

소프트웨어 참조 설명서

Power PMAC

Power PMAC Software Reference Manual

050-PRPMAC-0S0

January 6, 2015



DELTA TAU
Data Systems, Inc.

NEW IDEAS IN MOTION ...

저작권

© 2017 Delta Tau Data Systems, Inc. All rights reserved.

이 설명서는 Delta Tau Data Systems 의 정규 사용자용으로 작성되었습니다. 정규 사용자 이외의 사람이 Delta Tau Data Systems 의 서면 허가 없이 이 설명서를 이용하는 것은 금지됩니다. 이 설명서에 기재된 정보는 제품의 개선 등에 따라 갱신되는 경우가 있으나 제품이 개선될 때마다 갱신되는 것은 아니며 일정 기간마다 갱신됩니다.

이 설명서의 내용에 잘못된 기재나 불일치가 있는 경우는 다음 연락처로 전화하거나 전자 메일로 알려주십시오.

Delta Tau Data Systems 기술 지원

전화번호: (818) 717-5656

Fax: (818) 998-7807

전자 메일: support@deltatau.com

웹 사이트: <http://www.deltatau.com>

동작 조건

Delta Tau Data Systems 의 모든 모션 컨트롤러 제품, 액세서리 및 앰프에는 정전기의 영향을 받기 쉬운 부품이 포함되어 있으므로 잘못 취급하면 손상될 가능성이 있습니다. Delta Tau Data Systems 의 제품을 설치하거나 취급할 때는 절연성이 높은 물질과의 접촉을 피하십시오. 이 제품은 자격을 갖춘 담당자만 취급하십시오.

이 제품을 산업용으로 사용하는 경우는 위험한 물질이나 도전성 물질에서 보호하여 컨트롤러가 손상되는 원인이나 단락이 발생하기 쉬운 환경에서 보호해야 합니다. 이 제품을 산업용으로 사용하는 경우 산업용 제어반 내 또는 산업용 PC 내에 이 제품을 설치하여 과도한 습도 또는 부식성 습기로부터 보호하고 주변 온도를 유지하며 도전성 물질에서 보호할 수 있도록 하십시오. 당사의 제품이 위험한 물질이나 도전성 물질에 직접 노출되는 환경에 설치한 경우 당사는 이 제품의 동작을 보증하지 않습니다.

안전상의 주의 사항

이 제품의 운송, 조립, 설치 및 보수는 자격을 갖춘 담당자가 실시하십시오. 자격을 갖춘 담당자란, 제품의 운송, 조립, 설치 및 조작을 숙지하고 있는 사람을 가리킵니다. 자격을 갖춘 담당자는 다음 규격 및 규제를 숙지하고 이에 따라야 합니다.

IEC364resp.CENELEC HD 384 또는 DIN VDE 0100

IEC report 664 또는 DIN VDE 0110

안전 및 사고 방지에 관한 일본 국내의 규제 또는 VBG 4

잘못된 취급은 인명 사고나 기계 손상의 원인이 됩니다. 반드시 설명서의 설치 방법을 따르십시오. 저저항 접지에 따라 전기적 안전을 확보하십시오. 반드시 모든 시스템의 부품을 접지하십시오.

이 제품에는 정전기의 영향을 받기 쉬운 부품도 있습니다. 따라서 잘못 취급하면 이 제품이 손상될 가능성이 있습니다. 절연성이 높은 물질(인공 소재, 플라스틱 필름 등)과의 접촉을 피하십시오. 도전성이 있는 곳에 설치하십시오. 이 제품을 만지기 전에 접지된 금속의 표면 등을 만져서 신체의 정전기를 방전하십시오.

운전 중에는 모든 커버 및 캐비닛의 도어를 닫으십시오. 이 제품은 운전 중에 대전되거나 고온이 될 수 있습니다. 모터가 회전하지 않는 경우에도 컨트롤 케이블이나 전원 케이블에는 높은 전압이 인가됩니다. 통전 중에는 아크 방전을 피하기 위해 이 제품의 전원 플러그를 빼거나 꽂지 마십시오.



이 그림 기호는 인명 사고나 사망 사고로 연결될 우려가 있는 위험을 나타냅니다. 제품 사용 시 우선 확인하십시오.

경고



이 그림 기호는 제품 손상으로 연결될 우려가 있는 위험을 나타냅니다. 제품 사용 시 우선 확인하십시오.

주의



이 그림 기호는 제품에 대한 지식 또는 제품을 사용할 때의 중요한 정보를 나타냅니다. 제품 사용 시의 보충 정보입니다.

참고사항

개정 이력				
개정	설명	날짜	변경	승인
1	초판	2011 년 11 월 27 일	SS	Curt Wilson
2	펌웨어 버전 1.5 출시에 따른 설명서 갱신	2012 년 8 월 10 일	SS	Curt Wilson
3	펌웨어 버전 1.6 출시에 따른 설명서 갱신	2014 년 3 월 17 일	SS	Curt Wilson
4	펌웨어 버전 2.0 출시에 따른 설명서 갱신	2014 년 12 월 5 일	SS	Curt Wilson

목차

POWER PMAC 명령 구문 요약	43
주의.....	43
정의.....	43
수학적 요소.....	44
산술 연산자	44
비트 단위 논리 연산자	44
표준 대입 연산자	44
동기 대입 연산자	45
조건 비교 연산자	45
논리 연산자	45
스칼라 함수	45
벡터 함수	46
매트릭스 산술 함수	46
매트릭스 변환 함수	47
문자(바이트) 버퍼 함수	47
문자열 조작 함수	47
온라인 명령	48
온라인 글로벌 명령	48
온라인 좌표계 명령	50
온라인 모터 명령	53
버퍼 스크립트 프로그램 명령	55
이동 명령	55
동작 모드 명령	55
축 속성 명령	55
이동 속성 명령	56
커터(공구) 직경 보정 명령	56
변수 대입 명령	56
프로그램의 로직 제어	57
스크립트 PLC의 실행 제어	58
C PLC의 실행 제어	58
포트 통신	58
다이렉트 모터 명령	58
좌표계의 다이렉트 명령	59
프로그램 쿼리 명령	60
Acc84E[i].저장 데이터 구조체 요소(V1.6 2014.03)	61
Acc84E[i].멀티채널 설정 요소	61
Acc84E[i].SerialEncCtrl	61
BiSS B/C(단일 방향) 프로토콜	63
EnDat 2.1/2.2 프로토콜	64
Mitutoyo 프로토콜	64
Panasonic 프로토콜	65
SSI 프로토콜	65
Tamagawa 프로토콜	66

Yaskawa 시그마 II/III/IV 프로토콜	66
Acc84E[i].싱글 채널 설정 요소	67
Acc84E[i].Chan[j].SerialEncCmd	67
BiSS-B/C(단일 방향) 프로토콜	69
EnDat2.1/2.2 프로토콜	70
Mitutoyo 프로토콜	73
Panasonic 프로토콜	73
SSI 프로토콜	74
Tamagawa FA 코더 프로토콜	75
Yaskawa 시그마 II/III/IV 프로토콜	75
CamTable[m].저장 데이터 구조체 요소(V1.6 2014.03)	77
CamTable[m].DacData[i] 캠 테이블의 토크 오프셋 명령	77
CamTable[m].DacEnable 캠 테이블의 토크 출력 유효	78
CamTable[m].DacSf 캠 테이블의 토크 스케일 계수	78
CamTable[m].Dx 캠 테이블의 소스 위치 범위	78
CamTable[m].Nx 캠 테이블의 구간 수	79
CamTable[m].OutBits 캠 테이블의 범용 출력 비트 수	79
CamTable[m].OutData[i] 캠 테이블의 범용 출력 명령	80
CamTable[m].OutLeftShift 캠 테이블의 범용 출력 좌측 시프트 수	80
CamTable[m].PosData[i] 캠 테이블의 위치 명령	81
CamTable[m].PosSf 캠 테이블의 위치 스케일 계수	81
CamTable[m].pOut 캠 테이블의 범용 출력 레지스터 주소	82
CamTable[m].pOutBuf 캠 테이블의 범용 출력 유지 레지스터 주소	83
CamTable[m].SlewPosOffset 캠 테이블의 위치 오프셋 변화율	84
CamTable[m].SlewX0 캠 테이블의 소스 시작 위치 변화율	84
CamTable[m].Source 캠 테이블의 소스 모터 번호	85
CamTable[m].Target 캠 테이블의 대상 모터 번호	85
CamTable[m].X0 캠 테이블의 소스 시작 위치	86
CompTable[m].저장 데이터 구조체 요소(V1.6 2014.03)	87
CompTable[m].Ctrl 보정 테이블의 모드 제어	87
CompTable[m].Data[i] 보정 테이블의 1 차원 등록	88
CompTable[m].Data[j][i] 보정 테이블의 2 차원 등록	89
CompTable[m].Data[k][j][i] 보정 테이블의 3 차원 등록	90
CompTable[m].Dx[n] 보정 테이블의 소스 구간 거리	91
CompTable[m].Nx[n] 보정 테이블의 구간 수	92
CompTable[m].OutCtrl 보정 테이블의 보정 출력 제어	93
CompTable[m].Sf[q] 보정 테이블의 목표값 스케일 계수	94
CompTable[m].Source[n] 보정 테이블의 소스 모터 수	94
CompTable[m].SourceCtrl 보정 테이블의 소스 레지스터 제어	95
CompTable[m].Target[q] 보정 테이블의 대상 주소	95
CompTable[m].X0[n] 보정 테이블의 소스 시작 위치	97
Coord[x].저장 데이터 구조체 요소(V1.6 2014.03)	98

Coord[x].AbortAllMode 좌표계의 전체 Abort 동작.....	98
Coord[x].AbortTimeBase 좌표계의 Abort 타임 베이스.....	99
Coord[x].AddedDwellTime 좌표계의 블렌드 무효 시 드웰 시간.....	101
Coord[x].AltFeedrate 좌표계의 비 벡터축 프로그램 속도.....	102
Coord[x].AutoTxyzScale 좌표계의 변환 매트릭스 재스케일링.....	103
Coord[x].CCAddedArcBp 좌표계의 공구 직경 보정 추가 원호 브레이크 포인트.....	103
Coord[x].CCCtrl 좌표계의 공구 직경 보정 제어.....	105
Coord[x].CCDistance 좌표계의 공구 직경 보정 룩 어헤드 거리.....	106
Coord[x].CCSize 좌표계의 공구 직경 보정 이동 버퍼 크기.....	107
Coord[x].Control[i] 좌표계의 제어 요소.....	108
Coord[x].CornerAccel 좌표계의 블렌드 코너 가속도.....	108
Coord[x].CornerBlendBp 좌표계의 코너 블렌드 브레이크 포인트.....	110
Coord[x].CornerDwellBp 좌표계의 코너 드웰 브레이크 포인트.....	111
Coord[x].CornerError 좌표계의 코너 오차.....	112
Coord[x].CornerRadius 좌표계의 블렌드 코너 반경.....	114
Coord[x].Dprog 좌표계의 D 코드 프로그램 번호.....	115
Coord[x].FeedHoldSlew 좌표계의 피드 홀드의 슬루 레이트.....	115
Coord[x].FeedTime 좌표계 속도의 시간 단위.....	116
Coord[x].GoBack 좌표계의 점프 복귀 횟수.....	117
Coord[x].Gprog 좌표계의 G 코드 프로그램 번호.....	118
Coord[x].HomeRequired 좌표계의 원점 복귀 필요 조건.....	118
Coord[x].InPosTimeout 좌표계의 인포지션 타임아웃.....	119
Coord[x].LHDistance 좌표계의 룩 어헤드 거리.....	120
Coord[x].MaxCirAccel 좌표계의 최대 구심가속도.....	122
Coord[x].MaxFeedrate 좌표계의 최대 피드 레이트.....	123
Coord[x].MinArcLen 좌표계의 최소 원호 길이.....	124
Coord[x].Mprog 좌표계의 M 코드 프로그램 번호.....	125
Coord[x].Ndisplay 좌표계의 동기 행 레이블 표시 제어.....	125
Coord[x].NoBlend 좌표계의 블렌드 무효.....	126
Coord[x].NoCornerBp 좌표계의 코너 브레이크 포인트 무효.....	126
Coord[x].pDesTimeBase 좌표계의 타임 베이스 소스 포인터.....	127
Coord[x].PosRollOver[i] 좌표계의 축 위치 롤 오버 범위.....	127
Coord[x].RadiusErrorLimit 좌표계의 원호 반경 오차 한계.....	129
Coord[x].RapidVelCtrl 좌표계의 Rapid 속도 제어.....	129
Coord[x].SegLinToPvt 좌표계의 Linear 이동 PVT 실행.....	130
Coord[x].SegMoveTime 좌표계의 세그먼트 시간.....	131
Coord[x].SegOverrideSlew 좌표계의 세그먼트 오버라이드 슬루 레이트.....	132
Coord[x].SoftLimitStopDis 좌표계의 소프트웨어 리미트 동작 제어.....	133
Coord[x].SplineTimeRotate 좌표계의 스플라인 동작 시간 제어.....	134
Coord[x].StepMode 좌표계의 모션 프로그램 싱글 스텝 모드.....	134
Coord[x].SyncOps 좌표계의 동기 할당 버퍼 크기.....	136
Coord[x].Ta 좌표계의 블렌드 이동 가속 시간.....	137
Coord[x].Td 좌표계의 블렌드 이동 최종 감속 시간.....	137
Coord[x].TimeBaseSlew 좌표계의 타임 베이스 슬루 레이트.....	138
Coord[x].Tm 좌표계의 블렌드 이동 시간/피드 레이트.....	140

Coord[x].TPCoords 좌표계의 목표 위치 통지	140
Coord[x].Tprog 좌표계의 T 코드 서브 프로그램 번호	141
Coord[x].TPSize 좌표계의 목표 위치 버퍼 크기	141
Coord[x].Ts 좌표계의 블렌드 이동 S 자 곡선 시간	142
ECAT[j].저장 데이터 구조체 요소	144
ECAT[j].네트워크 전반적인 구성 요소	144
ECAT[j].AmpEnaTimeout	144
ECAT[j].DCRefBand	144
ECAT[j].DCRefMinus	145
ECAT[j].DCRefPlus	145
ECAT[j].DCRefSlave	145
ECAT[j].DistrClocks	146
ECAT[j].DistrClocksCount	146
ECAT[j].Enable	146
ECAT[j].IOCount	147
ECAT[j].LPIOCount	148
ECAT[j].LPnotLRW	148
ECAT[j].LPStateCheck	148
ECAT[j].LPStateCheckCount	149
ECAT[j].RTnotLRW	149
ECAT[j].RTStateCheck	150
ECAT[j].RTStateCheckCount	150
ECAT[j].ServoExtension	150
ECAT[j].SlaveCount	151
ECAT[j].사이클릭 I/O 구성 요소	152
ECAT[j].IO[k].BitLength	152
ECAT[j].IO[k].BitPosition	152
ECAT[j].IO[k].Index	153
ECAT[j].IO[k].Input	153
ECAT[j].IO[k].Slave	153
ECAT[j].IO[k].SubIndex	154
ECAT[j].낮은 우선도 I/O 모듈의 구성 요소	155
ECAT[j].LPIO[k].BitLength	155
ECAT[j].LPIO[k].BitPosition	155
ECAT[j].LPIO[k].Index	155
ECAT[j].LPIO[k].Input	156
ECAT[j].LPIO[k].Slave	156
ECAT[j].LPIO[k].SubIndex	156
ECAT[j].의 슬레이브 구성 요소	157
ECAT[j].Slave[j].Alias	157
ECAT[j].Slave[j].AssignActivate	157
ECAT[j].Slave[j].Enable	157
ECAT[j].Slave[j].Position	158
ECAT[j].Slave[j].ProductCode	158
ECAT[j].Slave[j].Sync0Cycle	159
ECAT[j].Slave[j].Sync0Shift	159
ECAT[j].Slave[j].Sync1Cycle	159
ECAT[j].Slave[j].Sync1Shift	159
ECAT[j].Slave[j].VendorID	160
ECAT[j].슬레이브 프로세스 데이터 개체의 구성 요소	161
ECAT[j].Slave[j].PDO[k].BitLength	161

ECAT[i].Slave[j].PDO[k].Index	161
ECAT[i].Slave[j].PDO[k].Input	161
ECAT[i].Slave[j].PDO[k].SubIndex	162
ECAT[i].슬레이브 PDO 의 매핑 구성 요소	163
ECAT[i].Slave[j].PDOMapping[m].Index	163
ECAT[i].Slave[j].PDOMapping[m].PDOCount	163
ECAT[i].Slave[j].PDOMapping[m].pPDO	163
ECAT[i].슬레이브 동기 매니저의 구성 요소	164
ECAT[i].Slave[j].SyncManager[n].Dir	164
ECAT[i].Slave[j].SyncManager[n].Index	164
ECAT[i].Slave[j].SyncManager[n].PDOMappingCount	164
ECAT[i].Slave[j].SyncManager[n].pPDOMapping	165
ECAT[i].Slave[j].SyncManager[n].WatchdogMode	165
EncTable[n].저장 데이터 구조체 요소	166
EncTable[n].CosBias	166
EncTable[n].CoverError	167
EncTable[n].EncBias	168
EncTable[n].index1	170
EncTable[n].index2	172
EncTable[n].index3	176
EncTable[n].index4	177
EncTable[n].index5	179
EncTable[n].index6	180
EncTable[n].MaxDelta	181
EncTable[n].pEnc	183
EncTable[n].pEnc1	186
EncTable[n].PrevDelta	188
EncTable[n].ScaleFactor	189
EncTable[n].SinBias	191
EncTable[n].TanHalfPhi	192
EncTable[n].type	194
Gate1[i].(PMAC2 스타일 서보 IC)저장 데이터 구조체 요소(V1.6 2014.03)	197
Gate1[i].멀티채널 설정 요소	197
Gate1[i].AdcStrobe 게이트 1 의 ADC 스트로브	198
Gate1[i].ClockCtrl 게이트 1 의 클록 제어	199
Gate1[i].DacStrobe 게이트 1 의 DAC 스트로브	199
Gate1[i].HardwareClockCtrl 게이트 1 의 하드웨어 클록 제어	200
Gate1[i].PhaseClockDiv 게이트 1 의 위상 클록 주파수 분주	202
Gate1[i].PhaseServoDir 게이트 1 의 위상/서보 클록 방향	203
Gate1[i].PwmCtrl 게이트 1 의 PWM 제어 워드	204
Gate1[i].PwmDeadTime 게이트 1 의 PWM 데드 타임/PFM 펄스 폭	205
Gate1[i].PwmPeriod 게이트 1 의 최대 위상/PWM 주기 제어	206
Gate1[i].ServoClockDiv 게이트 1 의 서보 클록 주파수 분주	208
Gate1[i].채널 고유의 설정 요소	210
Gate1[i].Chan[j].CaptCtrl 게이트 1 채널의 위치 포착 제어	210
Gate1[i].Chan[j].CaptFlagSel 게이트 1 채널의 위치 포착 플래그 선택	211

Gate1[i].Chan[j].Ctrl 게이트 1 채널의 제어 풀 워드	212
Gate1[i].Chan[j].EncCtrl 게이트 1 채널의 엔코더 디코드 제어.....	213
Gate1[i].Chan[j].Equ1Ena 게이트 1 채널의 비교 엔코더 선택	214
Gate1[i].Chan[j].GatedIndexSel 게이트 1 채널의 게이트 인덱스 포착 선택.....	215
Gate1[i].Chan[j].IndexGateState 게이트 1 채널의 게이트 인덱스 포착 상태.....	216
Gate1[i].Chan[j].OneOverTEna 게이트 1 채널의 하드웨어 1/T 유효.....	217
Gate1[i].Chan[j].OutputMode 게이트 1 채널의 출력 모드 선택.....	218
Gate1[i].Chan[j].OutputPol 게이트 1 채널의 출력 극성.....	218
Gate1[i].Chan[j].PfmDirPol 게이트 1 채널의 PFM 방향 출력 극성.....	220
Gate3[j].(PMAC3 스타일 IC)저장 데이터 구조체 요소(V1.6 2014.03).....	221
Gate3[j].멀티채널 설정 요소.....	222
Gate3[i].AdcAmpClockDiv 게이트 3 의 앰프용 ADC 클럭 분주	222
Gate3[i].AdcAmpCtrl 게이트 3 의 앰프용 ADC 제어.....	223
Gate3[i].AdcAmpDelay 게이트 3 의 앰프용 ADC 시작 지연	223
Gate3[i].AdcAmpHeaderBits 게이트 3 의 앰프용 ADC 헤더 비트.....	224
Gate3[i].AdcAmpStrobe 게이트 3 의 앰프 ADC 스트로브 워드.....	224
Gate3[i].AdcAmpUtoS 게이트 3 의 앰프용 ADC 부호 변환	225
Gate3[i].AdcEncClockDiv 게이트 3 의 엔코더용 ADC 클럭 분주.....	226
Gate3[i].AdcEncCtrl 게이트 3 의 엔코더용 ADC 제어	226
Gate3[i].AdcEncDelay 게이트 3 의 엔코더용 ADC 시작 지연.....	227
Gate3[i].AdcEncHeaderBits 게이트 3 의 엔코더용 ADC 헤더 비트.....	228
Gate3[i].AdcEncStrobe 게이트 3 의 엔코더용 ADC 스트로브 워드.....	228
Gate3[i].AdcEncUtoS 게이트 3 의 엔코더용 ADC 부호 변환.....	229
Gate3[i].ClockPol 게이트 3 의 클럭 극성 제어.....	229
Gate3[i].DacClockDiv 게이트 3 의 DAC 클럭 분주.....	230
Gate3[i].DacStrobe 게이트 3 의 DAC 스트로브 워드	231
Gate3[i].EncClockDiv 게이트 3 의 엔코더 클럭 분주.....	232
Gate3[i].EncLatchDelay 게이트 3 의 엔코더 래치 지연	233
Gate3[i].FiltClockDiv 게이트 3 의 제 2 필터 클럭 분주.....	234
Gate3[i].GpioCtrl 게이트 3 의 범용 IO 제어.....	235
Gate3[i].GpioDir[j] 게이트 3 의 범용 I/O 방향.....	236
Gate3[i].GpioMode[j] 게이트 3 의 범용 I/O 모드.....	236
Gate3[i].GpioPol[j] 게이트 3 의 범용 I/O 극성.....	238
Gate3[i].HardwareClockCtrl 게이트 3 의 하드웨어 클럭 제어.....	238
Gate3[i].MacroEnableA 게이트 3 의 MACRO 뱅크 A 구성.....	239
Gate3[i].MacroEnableB 게이트 3 의 MACRO 뱅크 B 구성.....	240
Gate3[i].MacroModeA 게이트 3 의 매크로 A 모드.....	241
Gate3[i].MacroModeB 게이트 3 의 매크로 B 모드.....	242
Gate3[i].PfmClockDiv 게이트 3 의 펄스 주파수 변조 클럭 분주.....	244
Gate3[i].PhaseClockDiv 게이트 3 의 위상 클럭 입력 분주 계수.....	245
Gate3[i].PhaseClockMult 게이트 3 의 위상 클럭 출력 배율.....	246
Gate3[i].PhaseFreq 게이트 3 의 내부 위상 클럭 주파수.....	246
Gate3[i].PhaseServoClockCtrl 게이트 3 의 위상서보 클럭 제어.....	247

Gate3[i].PhaseServoDir 게이트 3 의 위상/서보 클록 방향 제어.....	248
Gate3[i].ResolverCtrl 게이트 3 의 리졸버 제어.....	249
Gate3[i].SerialEncCtrl 게이트 3 의 시리얼 엔코더 제어.....	250
SPI 프로토콜.....	252
SSI 프로토콜.....	252
EnDat 2.1/2.2 는 프로토콜	252
Hiperface 프로토콜	253
Yaskawa 시그마 I 프로토콜	253
Yaskawa 시그마 II/III/IV 프로토콜	254
Tamagawa FA 코더 프로토콜	255
Panasonic 프로토콜.....	255
Mitutoyo 프로토콜	256
Kawasaki 프로토콜	256
Gate3[i].ServoClockDiv 게이트 3 의 서보 클록 분주	257
Gate3[i].채널 특유의 설정 요소	258
Gate3[i].Chan[j].AdcOffset[k] 게이트 3 채널의 ADC 오프셋.....	258
Gate3[i].Chan[j].AtanEna 게이트 3 채널의 아크탄젠트 유효.....	258
Gate3[i].Chan[j].CaptCtrl 게이트 3 채널의 포착 제어.....	259
Gate3[i].Chan[j].CaptFlagChan 게이트 3 채널의 포착 플래그 채널 선택.....	261
Gate3[i].Chan[j].CaptFlagSel 게이트 3 채널의 위치 포착 플래그 선택.....	261
Gate3[i].Chan[j].EncCtrl 게이트 3 채널의 엔코더 제어.....	262
Gate3[i].Chan[j].Equ1Ena 게이트 3 채널의 비교 엔코더 선택.....	264
Gate3[i].Chan[j].EquOutMask 게이트 3 채널의 비교 상태 출력 마스크.....	264
Gate3[i].Chan[j].EquOutPol 게이트 3 채널의 비교 출력 극성.....	265
Gate3[i].Chan[j].FlagFilt2Ena 게이트 3 채널의 제 2 필터 유효.....	266
Gate3[i].Chan[j].GatedIndexSel 게이트 3 채널의 게이트 인덱스 선택.....	266
Gate3[i].Chan[j].InCtrl 게이트 3 채널의 입력 제어.....	267
Gate3[i].Chan[j].IndexDemuxEna 게이트 3 채널 다중화 해제 유효.....	268
Gate3[i].Chan[j].IndexGateState 게이트 3 채널의 인덱스 게이트 상태.....	269
Gate3[i].Chan[j].OutCtrl 게이트 3 채널의 출력 제어.....	270
Gate3[i].Chan[j].OutputMode 게이트 3 채널의 위상 출력 모드.....	271
Gate3[i].Chan[j].OutputPol 게이트 3 채널의 위상 출력 반전	271
Gate3[i].Chan[j].PackInData 게이트 3 채널의 입력 데이터 압축 유효	273
Gate3[i].Chan[j].PackOutData 게이트 3 채널의 출력 데이터 압축 유효.....	274
Gate3[i].Chan[j].Pfm 게이트 3 채널의 PFM 명령.....	275
Gate3[i].Chan[j].PfmDirPol 게이트 3 채널의 PFM 방향 극성.....	276
Gate3[i].Chan[j].PfmFormat 게이트 3 채널의 PFM 출력 형식.....	277
Gate3[i].Chan[j].PfmWidth 게이트 3 채널의 PFM 출력 펄스 폭.....	277
Gate3[i].Chan[j].PwmDeadTime 게이트 3 채널의 PWM 데드 타임	278
Gate3[i].Chan[j].PwmFreqMult 게이트 3 채널의 PWM 주파수 승수	279
Gate3[i].Chan[j].SerialEncCmd 게이트 3 채널의 시리얼 엔코더 명령.....	279
SPI 프로토콜.....	281
SSI 프로토콜.....	281
EnDat2.1/2.2 프로토콜	282
Hiperface 프로토콜	283

Yaskawa 시그마 I 프로토콜	284
Yaskawa 시그마 II/III/V 프로토콜	285
Tamagawa FA 코더 프로토콜	285
Panasonic 프로토콜	286
Mitutoyo 프로토콜	287
Kawasaki 프로토콜	287
Gate3[i].Chan[j].SerialEncEna 게이트 3 채널의 시리얼 엔코더 유효.....	288
Gate3[i].Chan[j].TimerMode 게이트 3 채널의 타이머 모드.....	288
GateIo[i].저장 데이터 구조체 요소(V1.6 2014.03)	291
GateIo[i].Init.CtrlReg 게이트 IO의 레지스터 방향 초기화 제어.....	291
GateIo[i].Init.DataReg0[j] 게이트 IO의 전원 투입 기본값.....	292
GateIo[i].Init.DataReg64[j] 게이트 IO의 레지스터 반전 초기화 제어.....	292
GateIo[i].Init.DataReg128[j] 게이트 IO의 레지스터 읽기 초기화 제어.....	294
GateIo[i].Init.DataReg192[j] 게이트 IO의 레지스터 래치 초기화 제어.....	295
GateIo[i].Init.IntrReg64 게이트 IO의 인터럽트 라인 반전 초기화 제어.....	296
GateIo[i].Init.IntrReg128 게이트 IO의 인터럽트 라인 마스크 초기화 제어.....	296
GateIo[i].Init.IntrReg192 게이트 IO의 인터럽트 라인 엣지/레벨 초기화 제어.....	297
Motor[x].저장 데이터 구조체 요소(V1.6 2014.03)	298
Motor[x].AbortTa 모터의 Abort 감속 시간.....	298
Motor[x].AbortTs 모터의 Abort S자 감속 시간.....	299
Motor[x].AbsPhasePosForce 모터의 절대 위상 위치의 강제값.....	300
Motor[x].AbsPhasePosFormat 모터의 절대 위상 위치의 형식.....	300
Motor[x].AbsPhasePosOffset 모터의 절대 위상 위치 오프셋.....	302
Motor[x].AbsPhasePosSf 모터의 절대 위상 위치의 스케일 팩터.....	304
Motor[x].AbsPosFormat 모터의 절대 위치 데이터 형식.....	305
Motor[x].AbsPosSf 모터의 절대 위치 스케일 팩터.....	308
Motor[x].AdcMask 모터의 전류 피드백 마스크.....	308
Motor[x].AdvGain 모터의 전류(commutation) 위상 전진 게인.....	309
Motor[x].AmpEnableBit 모터의 앰프 Enable 신호 비트 번호.....	310
Motor[x].AmpFaultBit 모터의 앰프 폴트 신호의 비트 번호.....	310
Motor[x].AmpFaultLevel 모터의 앰프 이상 논리 상태.....	311
Motor[x].AuxFaultBit 모터의 보조 이상 비트 번호.....	312
Motor[x].AuxFaultLevel 모터의 보조 이상 논리 레벨.....	312
Motor[x].AuxFaultLimit 모터의 보조 이상 검출 제한.....	313
Motor[x].BIHysteresis 모터의 백래시 히스테리시스.....	314
Motor[x].BISize 모터의 백래시 크기.....	314
Motor[x].BISlewRate 모터의 백래시 슬루 레이트.....	315
Motor[x].BrakeOffDelay 모터의 브레이크 개방 지연 시간.....	315
Motor[x].BrakeOnDelay 모터의 브레이크 구동 지연 시간.....	316
Motor[x].BrakeOutBit 모터의 브레이크 출력 신호 비트 번호.....	317
Motor[x].CaptControl 모터의 포착 위치 제어.....	318
Motor[x].CaptFlagBit 모터의 포착 플래그 비트 번호.....	318
Motor[x].CaptPosLeftShift 모터의 포착 위치 좌측 시프트 수.....	319

Motor[x].CaptPosRightShift	모터의 포착 위치 우측 시프트 수.....	320
Motor[x].CaptPosRound	모터의 포착 위치 반올림 제어.....	321
Motor[x].CaptureMode	모터의 트리거 동작 위치 포착 모드.....	322
Motor[x].CmdMotor	모터의 갠트리 주축 모터 번호.....	323
Motor[x].Control[i]	모터의 요소 유효 제어.....	324
Motor[x].Ctrl	모터의 서보 알고리즘 선택.....	325
Motor[x].CurrentNullPeriod	모터의 전류 제로 조정 사이클 수.....	325
Motor[x].DacBias	모터의 서보 출력 오프셋.....	327
Motor[x].DacShift	모터의 출력 우측 시프트 양.....	327
Motor[x].DtOverRotorTc	모터의 슬립 상수.....	328
Motor[x].EncLossBit	모터의 엔코더 소실 비트 번호.....	329
Motor[x].EncLossLevel	모터의 엔코더 소실 레벨.....	329
Motor[x].EncLossLimit	모터의 엔코더 소실 제한.....	330
Motor[x].EncType	모터의 엔코더 타입.....	331
Motor[x].ExtraMotors	모터의 추가 모터.....	332
Motor[x].FatalFeLimit	모터의 중대한 편차 이상 리미트.....	333
Motor[x].FaultMode	모터의 이상 동작 제어.....	333
Motor[x].GantrySlewRate	모터의 갠트리 슬루 레이트.....	334
Motor[x].HomeOffset	모터의 원점 오프셋.....	335
Motor[x].HomeVel	모터의 원점 복귀 속도.....	336
Motor[x].I2tSet	모터의 연속 전류 제한.....	336
Motor[x].I2tTrip	모터의 적분 전류 제한.....	338
Motor[x].IaBias	모터의 A 상전류 바이어스.....	340
Motor[x].IbBias	모터의 B 상전류 바이어스.....	340
Motor[x].IdCmd	모터의 자화 전류.....	341
Motor[x].IiGain	모터의 전류 루프 적분 게인.....	342
Motor[x].InPosBand	모터의 인포지션 밴드.....	342
Motor[x].InPosTime	모터의 인포지션 스캔 횟수.....	343
Motor[x].InvAmax	모터의 최대 프로그램 가속도.....	343
Motor[x].InvDmax	모터의 최대 프로그램 감속도.....	344
Motor[x].InvJmax	모터의 최대 프로그램 저크.....	345
Motor[x].IiGain	모터의 전류 루프 백 패스 비례 게인.....	346
Motor[x].IpfGain	모터의 전류 루프 포워드 패스 비례 게인.....	347
Motor[x].IxCoupleGain	모터의 전류 루프 크로스 커플링 게인.....	347
Motor[x].JogOffset	모터의 JOG 오프셋.....	348
Motor[x].JogSpeed	모터의 JOG 속도.....	348
Motor[x].JogTa	모터의 JOG 가감속 시간.....	349
Motor[x].JogTs	모터의 JOG S 자 곡선 시간.....	350
Motor[x].LimitBits	모터의 리미트 신호 비트 수.....	351
Motor[x].MasterCtrl	모터의 위치 추종 제어 플래그.....	351
Motor[x].MasterMaxAccel	모터의 위치 추종 최대 가속도.....	352
Motor[x].MasterMaxSpeed	모터의 위치 추종 최고 속도.....	353
Motor[x].MasterPosSf	모터의 명령 마스터 위치 스케일 계수.....	354
Motor[x].MaxDac	모터의 최대 DAC 출력.....	354
Motor[x].MaxPos	모터의 양의 오버트래블 제한.....	356

Motor[x].MaxSpeed	모터의 프로그램 최대 속도.....	357
Motor[x].MinPos	모터의 음의 오버트래블 제한.....	358
Motor[x].MotorMode	모터의 동작 모드.....	358
Motor[x].MotorNodeOffset	모터의 네트워크 명령 오프셋.....	360
Motor[x].pAbsPhasePos	모터의 절대 전류(commutation) 위치 포인터.....	361
Motor[x].pAbsPos	모터의 절대 위치 포인터.....	362
Motor[x].pAdc	모터의 전류 피드백 포인터.....	363
Motor[x].pAmpEnable	모터의 앰프 Enable 포인터.....	364
Motor[x].pAmpFault	모터의 앰프 이상 포인터.....	365
Motor[x].pAuxFault	모터의 보조 이상 포인터.....	366
Motor[x].pBrakeOut	모터의 브레이크 출력 포인터.....	367
Motor[x].pBufPos	모터의 아우터 루프 확장 버퍼 포인터.....	369
Motor[x].pBufPos2	모터의 이너 루프 확장 버퍼 포인터.....	369
Motor[x].pCaptFlag	모터의 포착 플래그 포인터.....	370
Motor[x].pCaptPos	모터의 포착 위치 포인터.....	371
Motor[x].Pdi	모터의 사전 필터 분모항.....	372
Motor[x].pDac	모터의 명령 출력 포인터.....	372
Motor[x].pEnc	모터의 위치 루프 피드백 포인터.....	373
Motor[x].pEnc2	모터의 속도 루프 피드백 포인터.....	374
Motor[x].pEncCtrl	모터의 엔코더 포착 제어 포인터.....	375
Motor[x].pEncLoss	모터의 엔코더 소실 포인터.....	376
Motor[x].pEncStatus	모터의 부모 입력 플래그 포인터.....	377
Motor[x].PhaseCtrl	모터의 전류(commutation) 제어.....	378
Motor[x].PhaseEncLeftShift	모터의 페이즈 위치 좌측 시프트.....	379
Motor[x].PhaseEncRightShift	모터의 페이즈 위치 우측 시프트.....	380
Motor[x].PhaseFindingDac	모터의 페이즈 서치 출력.....	381
Motor[x].PhaseFindingTime	모터의 페이즈 서치 시간.....	381
Motor[x].PhaseLoadEncLeftShift	모터의 페이즈 부하 위치 좌측 시프트.....	383
Motor[x].PhaseLoadEncRightShift	모터의 페이즈 부하 위치 우측 시프트.....	384
Motor[x].PhaseMode	모터의 전류(commutation) 모드 제어.....	384
Motor[x].PhaseOffset	모터의 페이즈 오프셋.....	385
Motor[x].PhasePosSf	모터의 페이즈 위치 스케일 계수.....	387
Motor[x].PhaseSplineCtrl	모터의 페이즈 스플라인 보간.....	389
Motor[x].pLimits	모터의 오버트래블 리미트 포인터.....	390
Motor[x].pMasterEnc	모터의 마스터 엔코더 포인터.....	391
Motor[x].pMotorNode	모터의 네트워크 명령 포인터.....	391
Motor[x].Pni	모터의 사전 필터 분자항.....	392
Motor[x].PosReportMode	모터의 위치 통지 모드.....	393
Motor[x].PosSf	모터의 아우터 루프 스케일 계수.....	394
Motor[x].Pos2Sf	모터의 이너 루프 스케일 계수.....	395
Motor[x].PowerOnMode	모터의 전원 셋업 제어.....	396
Motor[x].pPhaseEnc	모터의 페이즈 엔코더 포인터.....	397
Motor[x].pPhaseLoadEnc	모터의 페이즈 부하 위치 포인터.....	398
Motor[x].PreFilterEna	모터의 사전 필터 유효.....	399
Motor[x].ProgJogPos	모터의 가변 JOG 위치.....	399

Motor[x].pSineTable	모터의 사인파 테이블 포인터.....	400
Motor[x].pVoltSineTable	모터의 전압 사인파 테이블 포인터.....	400
Motor[x].PwmDbComp	모터의 PWM 불감대 보정.....	401
Motor[x].PwmDbI	모터의 PWM 불감대 전류 최소치	402
Motor[x].PwmSf	모터의 PWM 출력 스케일 계수.....	402
Motor[x].RapidSpeedSel	모터의 Rapid 모드 속도 선택.....	403
Motor[x].ServoCapTimeOffset	모터의 서보 포착 시간 오프셋	404
Motor[x].ServoCtrl	모터의 서보 제어.....	405
Motor[x].SlewMasterPosSf	모터의 위치 추종비 슬루 레이트.....	406
Motor[x].SlipGain	모터의 슬립 게인.....	406
Motor[x].SoftLimitOffset	모터의 소프트웨어 리미트 오프셋.....	407
Motor[x].Stime	모터의 서보 사이클 연장.....	408
Motor[x].TraceSize	모터의 트레이스 크기.....	409
Motor[x].WarnFeLimit	모터의 편차 경고 리미트.....	410
모터 서보 루프항의 요소		411
Motor[x].Servo.BreakPosErr	모터 서보의 불감대 크기.....	411
Motor[x].Servo.EstMinDac	모터 서보의 추정 최소 서보 명령.....	411
Motor[x].Servo.EstTime	모터 서보의 추정 시간.....	412
Motor[x].Servo.Kafb	모터 서보의 가속도 피드백 게인.....	413
Motor[x].Servo.Kaff	모터 서보의 가속도 Feed-forward 게인	413
Motor[x].Servo.Kai	모터 서보의 명령 위치 필터 분자항.....	414
Motor[x].Servo.Kbi	모터 서보의 실제 위치 필터 분자항.....	415
Motor[x].Servo.Kbreak	모터 서보의 불감대 게인.....	415
Motor[x].Servo.Kci	모터 서보의 편차 필터 분자항.....	416
Motor[x].Servo.Kdi	모터 서보의 편차 필터 분모항.....	417
Motor[x].Servo.Kei	모터 서보의 이너 루프 필터 분모항.....	417
Motor[x].Servo.Kfi	모터 서보의 명령 속도 필터 분모항.....	418
Motor[x].Servo.Kfff	모터 서보의 마찰 Feed-forward 게인.....	418
Motor[x].Servo.Ki	모터 서보의 적분 게인.....	419
Motor[x].Servo.Kp	모터 서보의 비례 게인.....	420
Motor[x].Servo.Kvfb	모터 서보의 속도 피드백 게인.....	421
Motor[x].Servo.Kvff	모터의 속도 Feed-forward 게인	422
Motor[x].Servo.Kvifb	모터 서보의 속도 피드백 게인(적분기로)	423
Motor[x].Servo.Kviff	모터 서보의 속도 Feed-forward 게인(적분기로).....	424
Motor[x].Servo.Kxig	모터 서보의 크로스 커플링 적분 게인.....	425
Motor[x].Servo.Kxpg	모터 서보의 크로스 커플링 비례 게인.....	425
Motor[x].Servo.Kxvg	모터 서보의 크로스 커플링 미분 게인	426
Motor[x].Servo.MaxDR	모터 서보의 최대 게인 감쇠율.....	426
Motor[x].Servo.MaxGainFactor	모터 서보의 최대 게인 팩터.....	427
Motor[x].Servo.MaxInt	모터 서보의 최대 적분 편차.....	427
Motor[x].Servo.MaxPosErr	모터 서보의 최대 위치 편차.....	428
Motor[x].Servo.MaxW	모터 서보의 최대 게인 고유 주파수.....	429
Motor[x].Servo.MinDR	모터 서보의 최소 게인 감쇠율.....	430
Motor[x].Servo.MinGainFactor	모터 서보의 최소 게인 팩터.....	431

Motor[x].Servo.MinW	모터 서보의 최소 게인 고유 주파수.....	431
Motor[x].Servo.NominalGain	모터 서보의 공칭 게인.....	432
Motor[x].Servo.OutDbOff	모터 서보의 출력 불감대 OFF.....	433
Motor[x].Servo.OutDbOn	모터 서보의 출력 불감대 ON.....	433
Motor[x].Servo.OutDbSeed	모터 서보의 출력 불감대 시드.....	434
Motor[x].Servo.SwFfInt	모터 서보의 마찰 Feed-forward 선택.....	434
Motor[x].Servo.SwPoly7	모터 서보의 7 차 다항식 선택.....	435
Motor[x].Servo.SwZvInt	모터 서보의 적분 모드 선택.....	435
Sys.저장 데이터 구조체 요소(V1.6 2014.03)		437
Sys.AbortAllBit	시스템의 글로벌 Abort 입력 비트 번호.....	437
Sys.AbortAllLimit	시스템의 글로벌 Abort 검출 최대 누적 수.....	437
Sys.BgSleepTime	시스템의 백그라운드 사이클 휴지 시간.....	438
Sys.BgWDTReset	시스템의 백그라운드 워치독 타이머 최소치.....	439
Sys.BusCtrl[n]	시스템의 I/O 버스 타이밍 제어.....	439
Sys.CamEnable	시스템의 유효 캠 테이블 수.....	441
Sys.CompEnable	시스템의 유효 보정 테이블 수.....	441
Sys.CompMotor	시스템의 보정 테이블 갱신 전의 모터 수.....	442
Sys.CpuTimerIntr	시스템의 프로세서 내부 타이머 인터럽트 유효.....	442
Sys.FirstEnc	시스템의 시작 ECT 등록 번호.....	443
Sys.I[i]	시스템의 I 변수 배열 요소.....	444
Sys.MaxCoords	시스템의 최대 좌표 계수.....	444
Sys.MaxEcats	시스템의 EtherCAT 네트워크 사용 수.....	445
Sys.MaxMotors	시스템의 사용 최대 모터 수.....	445
Sys.MaxRtPlc	시스템의 포어그라운드 PLC 의 최대 번호.....	446
Sys.MaxTimedUnderflow	시스템의 음의 타임 베이스 초과 시간.....	446
Sys.MotorsPerRtInt	시스템의 RTI 모터 Status 수.....	446
Sys.NoShortCmds	시스템의 생략형 명령 무효 제어.....	449
Sys.pAbortAll	시스템의 글로벌 Abort 입력 레지스터 포인터.....	449
Sys.PhaseCycleExt	시스템의 페이즈 사이클 연장 기간.....	450
Sys.PhaseOverServoPeriod	시스템의 페이즈에 대한 서보 갱신 주기비.....	451
Sys.PreCalc	시스템의 동작 전 계산 버퍼 시간.....	452
Sys.RtIntPeriod	시스템의 실시간 인터럽트 주기.....	453
Sys.SendFileMode	시스템의 송신 메시지 형식.....	453
Sys.ServoPeriod	시스템의 보간용 서보 갱신 주기.....	454
Sys.SimConfigOk	시스템의 시뮬레이트 구성 다운로드 유효.....	455
Sys.WDTReset	시스템의 포어그라운드 워치독 타이머 최소치.....	455
POWER PMAC 의 비저장 설정 데이터 구조체 요소.....		457
Acc5E[i].비저장 설정 데이터 구조체 요소.....		457
Acc5E3[i].비저장 설정 데이터 구조체 요소.....		457
Acc5EP3[i].비저장 설정 데이터 구조체 요소		457

Acc11C[j].비저장 설정 데이터 구조체 요소.....	457
Acc11E[i].비저장 설정 데이터 구조체 요소.....	457
Acc14E[j].비저장 설정 데이터 구조체 요소.....	457
Acc24C2[j].비저장 설정 데이터 구조체 요소.....	457
Acc24C2A[j].비저장 설정 데이터 구조체 요소	458
Acc24E2[j].비저장 설정 데이터 구조체 요소.....	458
Acc24E2A[j].비저장 설정 데이터 구조체 요소	458
Acc24E2S[j].비저장 설정 데이터 구조체 요소	458
Acc24E3[j].비저장 설정 데이터 구조체 요소.....	458
Acc36E[j].비저장 설정 데이터 구조체 요소.....	458
Acc36E[j].ConvertCode	458
Acc51C[j].비저장 설정 데이터 구조체 요소.....	459
Acc51E[j].비저장 설정 데이터 구조체 요소.....	459
Acc58E[j].비저장 설정 데이터 구조체 요소.....	460
Acc59E[j].비저장 설정 데이터 구조체 요소.....	460
Acc59E[j].ConvertCode	460
Acc59E[j].DAC[j].....	461
Acc59E[j].DACHighLow[j]	461
Acc59E3[j].비저장 설정 데이터 구조체 요소.....	462
Acc65E[j].비저장 설정 데이터 구조체 요소.....	462
Acc66E[j].비저장 설정 데이터 구조체 요소.....	462
Acc67E[j].비저장 설정 데이터 구조체 요소.....	462
Acc68E[j].비저장 설정 데이터 구조체 요소.....	462
Acc72EX[j].비저장 설정 데이터 구조체 요소	462
Acc72EX[j].Idata16[j].....	463
Acc72EX[j].Idata32[j]	463
Acc72EX[j].Udata8[j].....	464
Acc72EX[j].Udata16[j].....	464
Acc72EX[j].Udata32[j].....	465
Acc84B[j].비저장 설정 데이터 구조체 요소.....	466
Acc84C[j].비저장 설정 데이터 구조체 요소.....	466

Acc84E[i].비저장 설정 데이터 구조체 요소.....	466
<i>Acc84E[i].AuxSerialEncCtrl</i>	466
<i>Acc84E[i].AuxChan[j].SerialEncCmd</i>	467
Acc84S[i].비저장 설정 데이터 구조체 요소.....	467
BrickAC.비저장 데이터 구조체 요소	468
<i>BrickAC.멀티채널 비저장 설정 요소</i>	468
BrickAC.Config	468
BrickAC.Monitor	469
BrickAC.Reset	471
BrickLV.비저장 데이터 구조체 요소.....	472
<i>BrickLV.멀티채널 설정 요소</i>	472
BrickLV.Config.....	472
BrickLV.Monitor	474
BrickLV.Reset.....	475
Clipper[j].비저장 설정 데이터 구조체 요소.....	476
CamTable[m].비저장 설정 데이터 구조체 요소.....	477
CamTable[m].Enable.....	477
CamTable[m].PosOffset	478
Cid[j].비저장 설정 데이터 구조체 요소.....	479
Cid[j].PartCtrl[k]	479
Cid[j].PartData[k]	480
Clipper[j].비저장 데이터 구조체 요소.....	480
Coord[x].비저장 설정 데이터 구조체 요소	481
Coord[x].DesTimeBase	481
Coord[x].InvTimeMode	482
Coord[x].Q[i]	484
Coord[x].SegOverride.....	484
Coord[x].TxyzScale	484
<i>Coord[x].CC3Data[i].3D 공구 보정 요소</i>	485
Coord[x].CC3Data[i].CutRadius	485
Coord[x].CC3Data[i].NdotT	486
Coord[x].CC3Data[0].ToolOffset.....	487
Coord[x].CC3Data[0].ToolRadius	487
<i>Coord[x].Ldata.비저장 로컬 데이터 요소</i>	488
ECAT[i].비저장 설정 데이터 구조체 요소.....	489
ECAT[i].IOBuffer[m].....	489
<i>EtherCAT 주기적 I/O 의 비저장 요소</i>	489
ECAT[i].IO[k].Data.....	489
ECAT[i].IO[k].Offset.....	490
<i>EtherCAT 낮은 우선 순위 I/O 모듈의 비저장 요소</i>	490
ECAT[i].LPIO[k].Data.....	490

Gate1[i].(PMAC2 스타일 Servo IC)비저장 설정 데이터 구조체 요소.....	491
Gate1[i].Chan[j].AmpEna.....	491
Gate1[i].Chan[j].CompA.....	491
Gate1[i].Chan[j].CompAdd.....	492
Gate1[i].Chan[j].CompB.....	492
Gate1[i].Chan[j].Dac[k]	493
Gate1[i].Chan[j].EquWrite	494
Gate1[i].Chan[j].Pfm	494
Gate1[i].Chan[j].Pwm[k]	495
Gate2[i].(PMAC2 스타일 MACRO IC)비저장 설정 데이터 구조체 요소.....	496
Gate2[i].DispData	496
Gate2[i].HighIoData.....	497
Gate2[i].LowIoData.....	497
Gate2[i].Macro[j][k]	498
Gate2[i].MuxData	498
Gate2[i]. 채널 고유의 비저장 설정 요소.....	500
Gate2[i].Chan[j].AmpEna.....	500
Gate2[i].Chan[j].CompA.....	500
Gate2[i].Chan[j].CompAdd.....	501
Gate2[i].Chan[j].CompB.....	502
Gate2[i].Chan[j].Dac[k]	502
Gate2[i].Chan[j].EquWrite	503
Gate2[i].Chan[j].Pfm	503
Gate2[i].Chan[j].Pwm[k]	504
Gate3[i].(PMAC3 스타일 인터페이스 IC)비저장 설정 데이터 구조체 요소	506
Gate3[i]. 멀티채널 비저장 설정 요소.....	506
Gate3[i].GpioData[j]	506
Gate3[i].IntCtrl	507
Gate3[i].MacroOutA[j][k]	507
Gate3[i].MacroOutB[j][k]	508
Gate3[i]. 채널 고유의 비저장 설정 요소.....	509
Gate3[i].Chan[j].AmpEna.....	509
Gate3[i].Chan[j].CompA.....	509
Gate3[i].Chan[j].CompAdd.....	510
Gate3[i].Chan[j].CompB.....	511
Gate3[i].Chan[j].Dac[k]	512
Gate3[i].Chan[j].EquWrite	513
Gate3[i].Chan[j].OutFlagB	513
Gate3[i].Chan[j].OutFlagC	514
Gate3[i].Chan[j].OutFlagD	514
Gate3[i].Chan[j].Pwm[k]	514
Gatelo[i].비저장 설정 데이터 구조체 요소	516
Gatelo[i].CtrlReg.....	516
Gatelo[i].DataReg[j]	516
Gatelo[i].IntrReg	519
Gather.비저장 설정 데이터 구조체 요소	521
Gather.Addr[i]	521
Gather.Enable	521

Gather.Items	522
Gather.MaxSamples	523
Gather.Period	523
Gather.PhaseAddr[i]	523
Gather.PhaseEnable	524
Gather.PhaseItems	525
Gather.PhaseMaxSamples	525
Gather.PhasePeriod	525
Gather.PhaseType[i]	526
Gather.Type[i]	527
Ldata.비저장 설정 데이터 구조체 요소	529
Ldata.C[i]	529
Ldata.Control	530
Ldata.coord	530
Ldata.D[i]	531
Ldata.GoBack	531
Ldata.L[i]	532
Ldata.motor	532
Ldata.R[i]	533
Motor[x].비저장 설정 데이터 구조체 요소	534
Motor[x].CapturePos	534
Motor[x].PhaseFindingStep	535
Motor[x].PhaseTableBias	535
Motor[x].VaBias	536
Motor[x].VbBias	536
Muxlo.비저장 데이터 구조체 요소	537
Muxlo.PortA[n].Data / Muxlo.PortB[n].Data	537
Plc[i].비저장 데이터 구조체 요소	538
<i>Plc[i].Ldata. 비저장 로컬 데이터 요소</i>	<i>538</i>
PowerBrick[i].비저장 데이터 구조체 요소	538
Sys.글로벌한 비저장 설정 데이터 구조체 요소	539
Sys.Cdata[i]	539
Sys.Ddata[i]	539
Sys.Fdata[i]	540
Sys.Idata[i]	541
Sys.Lock[i]	541
Sys.M[i]	542
Sys.P[i]	543
Sys.Udata[i]	543
Sys.Uhex[i]	544
Sys.WpKey	544
Tdata[i].비저장 설정 데이터 구조체 요소	546
Tdata[i].Bias[m]	546
Tdata[i].Diag[m]	546
Tdata[i].UUVVWW[m]	547
Tdata[i].UVW[m]	548

Tdata[i].XXYYZZ[m].....	549
Tdata[i].XYZ[m]	549
UserAlgo.비저장 설정 데이터 구조체 요소	551
UserAlgo.BgCplc[i]	551
UserAlgo.CaptComplntr	551
UserAlgo.CFunc	552
UserAlgo.RtiCplc	552
Coord[x].좌표계 상태 요소(V1.6 2014.03)	553
Coord[x].ActSegOverride 좌표계의 세그먼트 오버라이드.....	553
Coord[x].AddedDwellDis 좌표계의 드웰 무효 상태	553
Coord[x].AmpEna 좌표계의 앰프 Enable 상태.....	554
Coord[x].AmpFault 좌표계의 앰프 폴트 상태	554
Coord[x].AmpWarn 좌표계의 앰프 경고 상태.....	554
Coord[x].BlockActive 좌표계의 싱글 스텝 블록 유효 상태.....	554
Coord[x].BlockRequest 좌표계의 블록 요구 플래그 상태	555
Coord[x].BufferWarn 좌표계의 동작 방정식 버퍼 오버 경고 상태.....	555
Coord[x].CC3Active 좌표계의 3D 공구 직경 보정 유효 상태	556
Coord[x].CCAddedArc 좌표계의 공구 직경 보정 추가 원호 상태.....	556
Coord[x].CCMode 좌표계의 공구 직경 보정 모드 상태	556
Coord[x].CCMoveType 좌표계의 공구 직경 보정 동작 타입 상태.....	557
Coord[x].CCOffReq 좌표계의 공구 직경 보정 취소 요구 상태.....	557
Coord[x].cdata 좌표계의 원호 동작 모드.....	557
Coord[x].CdPos[] 좌표계의 축 동작 명령 위치	558
Coord[x].Cflags 좌표계의 조건부 실행 상태.....	558
Coord[x].ClosedLoop 좌표계의 클로즈드 루프 모드 상태	559
Coord[x].ContMotion 좌표계의 연속 동작 요구 상태	559
Coord[x].Csolve 좌표계의 축 정의 상태.....	559
Coord[x].DesVelZero 좌표계의 명령 속도 제로 상태.....	560
Coord[x].EncLoss 좌표계의 인코더가 소실 에러 상태.....	560
Coord[x].FeedHold 좌표계의 피드 홀드 상태	561
Coord[x].FeFatal 좌표계의 편차 이상 상태	562
Coord[x].FeWarn 좌표계의 편차 경고 상태.....	562
Coord[x].FRAxes 좌표계의 벡터 피드 레이트 축 상태.....	563
Coord[x].HomeComplete 좌표계의 기준점 확립 상태	563
Coord[x].HomeInProgress 좌표계의 원점 서치 실행 상태.....	564
Coord[x].I2tFault 좌표계의 전류 적분 이상 상태.....	564
Coord[x].IncAxes 좌표계의 증가 모드 상태.....	564
Coord[x].InPos 좌표계의 인포지션 상태	565
Coord[x].LHMotorSlots 좌표계의 록 어헤드 버퍼 모터 열수.....	565
Coord[x].LHSize 좌표계의 록 어헤드 버퍼 세그먼트 수	566
Coord[x].LHStatus 좌표계의 특별 록 어헤드 버퍼 상태.....	566
Coord[x].LimitStop 좌표계의 하드웨어 리미트 정지 상태	567
Coord[x].LinToPvtBuf 좌표계의 Linear-PVT 변환 상태	567
Coord[x].LinToPvtError 좌표계의 linear-PVT 변환 에러 상태	567

Coord[x].LookAheadActive 좌표계의 룩 어헤드 버퍼 실행 상태.....	568
Coord[x].LookAheadChange 좌표계의 룩 어헤드 실행 상태.....	568
Coord[x].LookAheadDir 좌표계의 룩 어헤드 방향 상태.....	568
Coord[x].LookAheadFlush 좌표계의 룩 어헤드 버퍼 순환 상태.....	568
Coord[x].LookAheadLookBack 좌표계의 룩 어헤드 버퍼 후방 상태.....	569
Coord[x].LookAheadReCalc 좌표계의 룩 어헤드 버퍼 재계산 상태.....	569
Coord[x].LookAheadStop 좌표계의 룩 어헤드 정지 상태.....	569
Coord[x].LookAheadWrap 좌표계의 룩 어헤드 버퍼 순환 상태.....	569
Coord[x].MinusLimit 좌표계의 음의 하드웨어 리미트 상태.....	570
Coord[x].Motors[i] 좌표계의 축 할당 모터 번호.....	570
Coord[x].MoveMode 좌표계의 프로그램 동작 모드 상태.....	570
Coord[x].Ncalc 좌표계의 동기 행 레이블 번호.....	571
Coord[x].Normal[i] 좌표계의 직교 벡터 성분값.....	571
Coord[x].Nsync 좌표계의 실행 행 레이블 번호.....	572
Coord[x].NumMotors 좌표계의 위치 결정 축 모터 수.....	572
Coord[x].NXYZ[i] 좌표계의 3D 보정 법선 벡터 성분값.....	572
Coord[x].PlusLimit 좌표계의 양의 하드웨어 리미트 상태.....	573
Coord[x].ProgActive 좌표계의 모션 프로그램 기동 상태.....	573
Coord[x].ProgProceeding 좌표계의 모션 프로그램 진척 상태.....	573
Coord[x].ProgRunning 좌표계의 모션 프로그램 실행 상태.....	574
Coord[x].PvtError 좌표계의 PVT 모드 에러 상태.....	574
Coord[x].RadiusError 좌표계의 Circle 반경 에러 상태.....	575
Coord[x].RunTimeError 좌표계의 런타임 에러 상태.....	575
Coord[x].SegEnabled 좌표계의 세그먼트 실행 상태.....	576
Coord[x].SegMove 좌표계의 세그먼트 처리 상태.....	576
Coord[x].SegMoveAccel 좌표계의 제 1 동작 세그먼트 실행 상태.....	576
Coord[x].SegMoveDecel 좌표계의 종료 세그먼트 실행 상태.....	577
Coord[x].SegStopReq 좌표계의 세그먼트 동작 정지 요구 상태.....	577
Coord[x].SharpCornerStop 좌표계의 예각 코너 블렌드 무효 상태.....	577
Coord[x].SoftLimit 좌표계의 소프트웨어 리미트 정지 상태.....	577
Coord[x].SoftMinusLimit 좌표계의 음의 소프트웨어 리미트 상태.....	578
Coord[x].SoftPlusLimit 좌표계의 양의 소프트웨어 리미트 상태.....	578
Coord[x].Status[0] 좌표계의 제 1 Status.....	578
Coord[x].Status[1] 좌표계의 제 2 Status.....	580
Coord[x].TimeBase 좌표계의 서보 사이클 경과 시간.....	581
Coord[x].TimerEnabled 좌표계의 동작 타이머 유효 상태.....	581
Coord[x].TimersEnabled 좌표계의 모든 동작 타이머 상태.....	582
Coord[x].Tsel 좌표계의 선택 변환 행렬.....	582
Coord[x].TriggerMove 좌표계의 트리거 서치 진행 중 상태.....	582
Coord[x].TriggerNotFound 좌표계의 트리거 미검출 상태.....	583
Coord[x].TXYZ[i] 좌표계의 3D 공구 직경 보정 자세 벡터 성분.....	583
Coord[x].XYZoff[i] 좌표계의 3D 공구 직경 보정 오프셋 벡터 성분.....	583
Coord[x].Ldata 로컬 데이터 상태 요소.....	584
Coord[x].CC3Data[i].3D 공구 직경 보정 상태 요소.....	584

Coord[x].CC3Data[i].ToolOffset 좌표계의 3D 공구 직경 보정 축 오프셋	584
Coord[x].CC3Data[i].ToolRadius 좌표계 3D 공구 직경 보정의 공구 직경.....	585
Coord[x].TPData[i].목표 위치 버퍼 상태 요소	586
Coord[x].TPData[i].Ncalc 좌표계의 목표 위치 버퍼 동기 레이블값.....	586
Coord[x].TPData[i].Pos[j] 좌표계의 목표 위치 버퍼 이동축 목표값.....	586
Coord[x].TPData[i].XYZPos[j] 좌표계의 목표값 버퍼 공구 직경 보정 오프셋.....	587
Coord[x].TPExec.목표 위치 상태 요소.....	588
Coord[x].TPExec.Ncalc 좌표계의 목표 위치 동기 레이블값.....	588
Coord[x].TPExec.Pos[j] 좌표계의 목표 위치 동작축 목표값.....	588
Coord[x].TPExec.XYZPos[j] 좌표계의 목표 위치 공구 직경 보정 오프셋.....	589
ECAT[i].STATUS 데이터 구조체 요소.....	590
ECAT[i].DCClockDiff.....	590
ECAT[i].Error	590
ECAT[i].LinkUp.....	591
ECAT[i].LPDomainOutputState	591
ECAT[i].LPDomainState	591
ECAT[i].LPRxTime	592
ECAT[i].LPTxTime.....	592
ECAT[i].MasterState.....	592
ECAT[i].RTDomainOutputState	593
ECAT[i].RTDomainState.....	593
ECAT[i].RxTime.....	594
ECAT[i].TxTime	594
EtherCAT 슬레이브의 Status 요소.....	595
ECAT[i].Slave[j].Online	595
ECAT[i].Slave[j].State	595
Gate1[i].(PMAC2 스타일 서보 ASIC)상태 요소(V1.6 2014.03)	596
Gate1[i].멀티채널 상태 요소.....	596
Gate1[i].PartData[k] 게이트 1 의 IC 보드 부품 원시 데이터.....	596
Gate1[i].PartNum 게이트 1 의 하드웨어 부품 번호.....	596
Gate1[i].PartOpt 게이트 1 의 옵션 구성 코드	597
Gate1[i].PartRev 게이트 1 의 하드웨어 개정 번호.....	597
Gate1[i].PartType 게이트 1 의 하드웨어 부품 타입	598
Gate1[i].채널 고유 상태 요소	599
Gate1[i].Chan[j].ABC 게이트 1 의 엔코더 채널 입력값.....	599
Gate1[i].Chan[j].Adc[k] 게이트 1 의 ADC 값.....	599
Gate1[i].Chan[j].CountError 게이트 1 의 엔코더 디코드 카운트 에러.....	600
Gate1[i].Chan[j].EncLossN 게이트 1 의 엔코더 소실 비트.....	600
Gate1[i].Chan[j].Equ 게이트 1 의 위치 비교 플래그 출력.....	601
Gate1[i].Chan[j].Fault 게이트 1 의 앰프 이상 플래그 입력값.....	601
Gate1[i].Chan[j].HallState 게이트 1 의 포착 홀 위치 상태.....	602
Gate1[i].Chan[j].HomeCapt 게이트 1 의 외부 트리거 래치 엔코더 카운트.....	602

Gate1[i].Chan[j].HomeFlag 게이트 1의 원점 플래그 입력.....	603
Gate1[i].Chan[j].MinusLimit 게이트 1의 MLIM 플래그 입력.....	603
Gate1[i].Chan[j].PhaseCapt 게이트 1의 위상 인터럽트 엔코더 카운트값.....	604
Gate1[i].Chan[j].PlusLimit 게이트 1의 PLIM 플래그 입력.....	604
Gate1[i].Chan[j].PosCapt 게이트 1의 위치 포착 트리거 플래그.....	604
Gate1[i].Chan[j].ServoCapt 게이트 1의 서보 인터럽트 엔코더 카운트.....	605
Gate1[i].Chan[j].Status 게이트 1의 에러 코드 상태 요소.....	605
Gate1[i].Chan[j].T 게이트 1의 T 플래그 입력.....	606
Gate1[i].Chan[j].TimeBetweenCts 게이트 1의 엔코더 카운트 간 시간.....	607
Gate1[i].Chan[j].TimeSinceCts 게이트 1의 엔코더 카운트 경과 시간.....	607
Gate1[i].Chan[j].UserFlag 게이트 1의 사용자 플래그 입력.....	608
Gate1[i].Chan[j].UVW 게이트 1의 UVW 플래그 입력.....	608
Gate3[i].(PMAC3 스타일 ASIC)상태 요소(V1.6 2014.03).....	609
Gate3[i].멀티채널 상태 요소.....	609
Gate3[i].ChipID 게이트 3의 IC 식별 번호.....	609
Gate3[i].EEPromData[k] 게이트 3의 EEPROM 데이터 워드.....	609
Gate3[i].GpioOutData[j] 게이트 3의 범용 I/O 뱅크 출력값.....	610
Gate3[i].MacroInA[j][k] 게이트 3의 MACRO 뱅크 A 입력 데이터.....	610
Gate3[i].MacroInB[j][k] 게이트 3의 MACRO 뱅크 B 입력 데이터.....	611
Gate3[i].MacroError 게이트 3의 MACRO 노드 에러.....	612
Gate3[i].PartNum 게이트 3의 하드웨어 부품 번호.....	612
Gate3[i].PartOptn 게이트 3의 옵션 하드웨어 구성.....	613
Gate3[i].PartRev 게이트 3의 하드웨어 개정 번호.....	613
Gate3[i].PartType.....	614
Gate3[i].채널 고유 상태 요소.....	615
Gate3[i].Chan[j].ABC 게이트 3 채널의 엔코더 ABC 채널 입력값.....	615
Gate3[i].Chan[j].ABPins 게이트 3 채널의 AB 신호 레벨.....	615
Gate3[i].Chan[j].AdcAmp[k] 게이트 3 채널의 앰프 ADC 값.....	616
Gate3[i].Chan[j].AdcEnc[k] 게이트 3 채널의 엔코더 ADC 값.....	616
Gate3[i].Chan[j].Atan 게이트 3 채널의 엔코더 사인/코사인 각도.....	617
Gate3[i].Chan[j].AtanSumOfSqr 게이트 3 채널의 엔코더 사인/코사인 각도와 진폭.....	618
Gate3[i].Chan[j].CountError 게이트 3 채널의 엔코더 디코드 카운트 에러.....	618
Gate3[i].Chan[j].DemuxInvalid 게이트 3 채널의 UVW 다중화 분리 무효.....	619
Gate3[i].Chan[j].Equ 게이트 3 채널의 위치 비교 내부 상태.....	619
Gate3[i].Chan[j].EquOut 게이트 3 채널의 위치 비교 종합 출력.....	620
Gate3[i].Chan[j].Fault 게이트 3 채널의 앰프 이상 플래그 입력.....	620
Gate3[i].Chan[j].HallState 게이트 3 채널의 포착 홀 위치 상태.....	621
Gate3[i].Chan[j].HomeCapt 게이트 3 채널의 외부 트리거 래치를 통한 엔코더 카운트값.....	621
Gate3[i].Chan[j].HomeFlag 게이트 3 채널의 원점 플래그 입력.....	622
Gate3[i].Chan[j].LossCapt 게이트 3 채널의 엔코더 소실 래치 에러.....	622
Gate3[i].Chan[j].LossStatus 게이트 3 채널의 현재 엔코더 소실 에러.....	623

Gate3[i].Chan[j].MinusLimit 게이트 3 채널의 MLIM 플래그 입력.....	623
Gate3[i].Chan[j].PhaseCapt 게이트 3 채널의 페이즈 인터럽트 래치의 엔코더 카운트값.....	623
Gate3[i].Chan[j].PlusLimit 게이트 3 채널의 PLIM 플래그 입력.....	624
Gate3[i].Chan[j].PosCapt 게이트 3 채널의 위치 포착 트리거 플래그.....	624
Gate3[i].Chan[j].SerialEncDataA 게이트 3 채널의 주 위치 피드백 시리얼 엔코더 데이터.....	625
AB 상 카운터.....	625
SPI 프로토콜.....	625
SSI 프로토콜.....	626
EnDat2.1/2.2 프로토콜.....	626
Hiperface 프로토콜.....	626
Yaskawa 시그마 I 프로토콜.....	627
Yaskawa 시그마 II/III/IV 프로토콜.....	627
Tamagawa FA 코더 프로토콜.....	628
Panasonic 프로토콜.....	628
Mitutoyo 프로토콜.....	629
Kawasaki 프로토콜.....	629
Gate3[i].Chan[j].SerialEncDataB 게이트 3 의 시리얼 엔코더 부위치 피드백 데이터.....	629
AB 상 엔코더.....	630
SPI 프로토콜.....	630
SSI 프로토콜.....	631
EnDat2.1/2.2 프로토콜.....	631
Hiperface 프로토콜.....	631
Yaskawa 시그마 I 프로토콜.....	631
Yaskawa 시그마 II/III/IV 프로토콜.....	632
Tamagawa FA 코더 프로토콜.....	633
Panasonic 프로토콜.....	634
Mitutoyo 프로토콜.....	634
Kawasaki 프로토콜.....	635
Gate3[i].Chan[j].ServoCapt 게이트 3 채널의 서보 인터럽트 래치 엔코더 카운트....	635
Gate3[i].Chan[j].SosError 게이트 3 채널의 사인/코사인 제곱합 에러.....	636
Gate3[i].Chan[j].Status 게이트 3 채널의 에러 코드 Status 요소.....	637
Gate3[i].Chan[j].SumOfSquares 게이트 3 채널의 엔코더 사인/코사인 진폭값.....	637
Gate3[i].Chan[j].T 게이트 3 채널의 T 플래그 입력.....	638
Gate3[i].Chan[j].TimerA 게이트 3 채널의 타이머 모드 정보.....	639
Gate3[i].Chan[j].TimerB 게이트 3 채널의 타이머 모드 정보.....	639
Gate3[i].Chan[j].TrigState 게이트 3 채널의 위치 포착의 현재 트리거 입력.....	640
Gate3[i].Chan[j].UserFlag 게이트 3 채널의 USER 플래그 입력.....	641
Gate3[i].Chan[j].UVW 게이트 3 채널의 UVW 플래그 입력.....	641
모터 상태 데이터 구조체 요소(V1.6 2014.03).....	642
Motor[x].ActiveMasterPos 모터의 누적 위치 추종 거리.....	642
Motor[x].ActiveMasterPosSf 모터 실행 중 위치 추종 계수.....	642
Motor[x].ActPos 모터의 외측 루프 피드백 위치.....	643
Motor[x].ActPos2 모터의 내측 루프 피드백 위치.....	643

Motor[x].ActVel 모터의 실제 속도	643
Motor[x].AmpEna 모터의 앰프 유효 Status 비트	644
Motor[x].AmpFault 모터의 앰프 이상 에러 Status 비트	644
Motor[x].AmpWarn 모터의 앰프 경고 Status 비트	644
Motor[x].AuxFault 모터의 보조 이상 에러 Status 비트	645
Motor[x].AuxFaultCount 모터의 보조 이상 검출 스캔 누적수	645
Motor[x].BICompSize 모터의 백래시 보정 테이블 보정값	645
Motor[x].BIDir 모터의 백래시 방향 Status 비트	646
Motor[x].BlockRequest 모터의 블록 요구 플래그 Status 비트	646
Motor[x].CapturedPos 모터의 포착 위치 모니터링	646
Motor[x].ClosedLoop 모터의 클로즈드 루프 모드 Status 비트	647
Motor[x].CompDac 모터의 토크 보정 출력 보정값	647
Motor[x].CompDesPos 모터의 명령 위치 보정 테이블 오프셋	647
Motor[x].CompPos 모터의 외측 루프 실제 위치 보정 테이블 보정값	648
Motor[x].CompPos2 모터의 내측 루프 실제 위치 보정 테이블 보정값	648
Motor[x].Coord 모터의 좌표계 할당	648
Motor[x].CoordSf[i] 모터의 축 정의 계수	649
Motor[x].Csolve 모터의 PMATCH 계산 Status 비트	649
Motor[x].CurrentNullTimer 모터의 자동 제로 전류 조정 잔여 사이클 수	650
Motor[x].DacLimit 모터의 서보 출력 제한 Status 비트	650
Motor[x].DesPos 모터의 Net 명령 위치	651
Motor[x].DesVel 모터의 Net 명령 속도	651
Motor[x].DesVelZero 모터의 명령 속도 제로 Status 비트	651
Motor[x].EncLoss 모터의 센서 소실 에러 비트	652
Motor[x].EncLossCount 모터의 센서 소실 검출 누적 스캔 수	652
Motor[x].FeFatal 모터의 편차 이상 Status 비트	652
Motor[x].FeWarn 모터의 경고 위치 편차 Status 비트	653
Motor[x].FiltrMasterPos[i] 모터의 마스터 위치 이력 배열 요소	653
Motor[x].FiltrMasterVel 모터의 마스터 필터 속도	653
Motor[x].FiltrPos[i] 모터의 외측 루프 피드백 위치 이력 배열 요소	654
Motor[x].FiltrPos2[i] 모터의 내측 루프 피드백 위치 이력 배열 요소	654
Motor[x].FiltrVel 모터의 외측 루프 피드백 필터 속도	654
Motor[x].FiltrVel2 모터의 내측 루프 피드백 필터 속도	655
Motor[x].GantryHomed 모터의 갠트리 원점 검출 완료 Status 비트	655
Motor[x].HomeComplete 모터의 기준 위치 확립 Status 비트	655
Motor[x].HomeInProgress 모터의 원점 서치 동작 실행 중 Status 비트	656
Motor[x].HomePos 모터의 기준 위치	656
Motor[x].I2tFault 모터의 적분 전류 이상 Status 비트	656
Motor[x].I2tSum 모터의 현재 적분 전류값	657
Motor[x].IaMeas 모터의 A(U)상 측정 전류	657
Motor[x].IaVolts 모터의 A(U)상 출력 명령	657
Motor[x].IbMeas 모터의 B(V)상 측정 전류	658
Motor[x].IbVolts 모터의 B(V)상 출력 명령	658
Motor[x].IcVolts 모터의 C(W)상 출력 명령	658
Motor[x].IdMeas 모터의 자화 전류 측정값	658

Motor[x].IdInt	659
Motor[x].IdVolts 모터의 다이렉트 전류 루프 출력.....	659
Motor[x].Imag 모터의 로터 자계 추정 전류.....	659
Motor[x].InPos 모터의 인포지션 Status 비트	660
Motor[x].InPosTimer 모터의 연속 인포지션 스캔 카운터.....	660
Motor[x].IqCmd 모터의 직교 전류 명령.....	660
Motor[x].IqInt 모터의 직교 전류 적분 출력.....	661
Motor[x].IqMeas 모터의 직교 전류 측정값.....	661
Motor[x].IqVolts 모터의 직교 전류 루프 출력.....	661
Motor[x].JogPos 모터의 JOG 종료 명령 위치	662
Motor[x].JogVel 모터의 도달 명령 속도 절대값.....	662
Motor[x].LimitStop 모터의 하드웨어 리미트 정지 Status 비트.....	662
Motor[x].MasterPos 모터의 무제한 위치 추종 누적 거리.....	662
Motor[x].MinusLimit 모터의 하드웨어 마이너스 리미트 Status 비트.....	663
Motor[x].MoveDesPos 모터의 프로그램 동작 종료 명령 위치	663
Motor[x].MoveTimer 모터의 현재 섹션 실행 경과 모터 시간.....	664
Motor[x].PhaseFound 모터의 위상 원점 확립 Status 비트	664
Motor[x].PhasePos 모터의 전류(commutation) 현재 위상각.....	664
Motor[x].PhaseVoltOffset 모터의 전류(commutation) 위상 전진 오프셋.....	665
Motor[x].PlusLimit 모터의 하드웨어 플러스 리미트 Status 비트.....	665
Motor[x].Pos 모터의 미처리 외측 루프 실제 위치.....	665
Motor[x].Pos2 모터의 미처리 내측 루프 실제 위치.....	666
Motor[x].PosError 모터의 서보 편차.....	666
Motor[x].Ppos 모터의 궤적 프리 필터	666
Motor[x].PresBlSize 모터의 현재 백래시 보정값.....	667
Motor[x].PrevDesPos 모터의 직전 Net 명령 위치	667
Motor[x].PrevMasterPos 모터의 직전 위치 추종 마스터 위치.....	667
Motor[x].PrevPhaseEnc 모터의 직전 위상 소스 위치.....	668
Motor[x].PrevPos2 모터의 직전 내측 루프 피드백 위치.....	668
Motor[x].Pui 모터의 i 사이클 이전의 모터 궤적 프리 필터 입력 위치.....	668
Motor[x].ResMasterPos 모터의 마스터 위치 잔차.....	668
Motor[x].ResPos 모터의 외측 루프 실제 위치 잔차.....	669
Motor[x].ResPos2 모터의 내측 루프 실제 위치 잔차.....	669
Motor[x].ServoOut 모터의 서보 루프 명령 출력	669
Motor[x].SoftLimit 모터의 소프트웨어 위치 리미트 정지 Status 비트.....	670
Motor[x].SoftLimitDir 모터의 소프트웨어 리미트 방향 Status 비트.....	670
Motor[x].SoftMinusLimit 모터의 소프트웨어 마이너스 리미트 Status 비트.....	671
Motor[x].SoftPlusLimit 모터의 소프트웨어 플러스 리미트 Status 비트	671
Motor[x].SpindleMotor 모터의 스핀들 모터 정의 상태	671
Motor[x].Status[0] 모터의 제 1 Status 워드	672
Motor[x].Status[1] 모터의 제 2 Status 워드	673
Motor[x].TriggerMove 모터의 트리거 동작 서치 중 Status 비트.....	673
Motor[x].TriggerNotFound 모터의 트리거 미검출 Status 비트.....	674
Motor[x].TriggerSpeedSel 모터의 트리거 속도 선택 Status 비트	674
Motor[x].VxCouple 모터의 전류 루프 크로스 커플링 주파수.....	674

Motor[x].Desired.궤적 서브 구조체 요소	675
Motor[x].Desired.Accel 모터 현재 실행의 가속도 계수	675
Motor[x].Desired.Jerk 모터 현재 실행의 저크 계수	675
Motor[x].Desired.Next 모터 현재 실행의 다음 인덱스	675
Motor[x].Desired.Pos 모터 현재 실행의 위치 계수	675
Motor[x].Desired.Time 모터 현재 실행의 시간 범위	676
Motor[x].Desired.TimerEnabled 모터 현재 실행의 시한 동작 Status 플래그	676
Motor[x].Desired.Vel 모터 현재 실행의 속도 계수	676
Motor[x].New[i].궤적 부구조체 요소	677
Motor[x].New[i].Accel 모터 직전 실행의 가속도 계수	677
Motor[x].New[i].Jerk 모터 직전 실행의 저크 계수	677
Motor[x].New[i].Pos 모터 직전 실행의 위치 계수	677
Motor[x].New[i].Time 모터 직전 실행의 시간 범위	678
Motor[x].New[i].Vel 모터 직전 실행의 속도 계수	678
Motor[x].Servo.부구조체 상태 요소	679
Motor[x].Servo.EstGain 모터 서보의 현재 추정 플랜트 게인	679
Motor[x].Servo.EstTimer 모터 서보의 적응 제어 서보 사이클 수	679
Motor[x].Servo.GainFactor 모터 서보의 적응 제어 기준 서보 게인	680
Motor[x].Servo.Integrator 모터 서보의 편차 적분 출력	680
Motor[x].Servo.Status 모터 서보의 불감대 출력 상태	680
Motor[x].Servo.uai 모터 서보의 다항식 A 입력	681
Motor[x].Servo.ubi 모터 서보의 다항식 B 입력	681
Motor[x].Servo.uci 모터 서보의 다항식 C/D 입력	681
Motor[x].Servo.udi 모터 서보의 다항식 C/D 출력	682
Motor[x].Servo.uei 모터 서보의 다항식 E 출력	682
Motor[x].Servo.ufi 모터 서보의 다항식 F 출력	682
Motor[x].Servo.Xint 모터 서보의 크로스 커플링 적분 편차 출력	683
Motor[x].TraceData[i].궤적 서브 구조체 요소	684
Motor[x].TraceData[i].Accel 모터 트레이스 데이터의 가속도 계수	684
Motor[x].TraceData[i].Jerk 모터 트레이스 데이터의 저크 계수	684
Motor[x].TraceData[i].Pos 모터 트레이스 데이터의 위치 계수	684
Motor[x].TraceData[i].Time 모터 트레이스 데이터의 시간 범위	685
Motor[x].TraceData[i].Vel 모터 트레이스 데이터의 속도 계수	685
Motor[x].TraceExec. 궤적 서브 구조체 요소	686
Motor[x].TraceExec.Accel 모터 트레이스 실행 가속도 계수	686
Motor[x].TraceExec.Jerk 모터 트레이스 실행 저크 계수	686
Motor[x].TraceExec.Pos 모터 트레이스 실행 위치 계수	686
Motor[x].TraceExec.Time 모터 트레이스 실행 시간 범위	687
Motor[x].TraceExec.Vel 모터 트레이스 실행 속도 계수	687
POWER PMAC 온라인 명령 사양	688

#.....	688
#{상수}	688
#{상수}->0	689
#{상수}->{축 정의}	689
#{상수}->I	691
#{영숫자}->S[{상수}]	691
#{목록}	693
#{목록}->	694
#*	694
\$	695
\$\$\$	696
\$\$\$***	696
%	697
%{상수}	697
&	698
&{상수}	699
&{목록}	699
&*	700
\	700
<	701
>	702
?	702
a	720
abort	721
adisable	721
b[{상수}]	722
backup	723
begin[{상수}]	725
bpclear	725
bpclearall	726
bpset	726
brickacver	727
bricklvver	727
buffer	728
clear all buffers	728
clear plc {상수}	729
clear prog {상수}	729
clear rotary	729
close	730
close all buffers	730
cpu	731
cpx	731
cx	732
{데이터 구조체 요소}	732
{데이터 구조체 요소}.a	733
{데이터 구조체 요소}={식}	734
d	734
D{데이터}	736
D{데이터}={식}	737
D{변수 목록}	737
D{변수 목록}={식}	739
date	740

ddisable	740
define lookahead	741
define rotary	742
delete all lookahead	743
delete all rotary	743
delete lookahead	744
delete rotary	744
disable	744
disable bgcplc	745
disable plc	745
disable rticplc	746
dkill	746
ecat alias	747
ecat assign	748
ecat config	749
ecat slaves	749
echo	750
echo{상수}	751
enable	752
enable bgcplc	752
enable plc	753
enable rticplc	754
f	754
free	755
g	755
h	756
hm	757
hmz	758
hold	758
home	759
homez	759
{데이터}	759
{데이터}={식}	760
{데이터}->	761
{변수 목록}	761
{변수 목록}={식}	763
{변수 목록}->	764
j+	765
j-	765
j/	766
j=	767
j={상수}	767
j={상수}	768
j=*	769
j:{상수}	770
j:*	771
j^{상수}	772
j^*	773
{JOG 명령}{상수}	774
{JOG 명령}^*	775
jog	776
jog+	777
jog-	777

jog/.....	777
jog=.....	778
jog={상수}.....	778
jog=={상수}.....	778
jog=*.....	779
jog:{상수}.....	779
jog:*.....	779
jog^{상수}.....	780
jog^*.....	780
k.....	780
kill.....	781
L{데이터}.....	782
L{데이터}={식}.....	782
L{변수 목록}.....	783
L{변수 목록}={식}.....	784
list apc.....	785
list {키네마틱스 버퍼}.....	786
list {프로그램 버퍼}.....	787
list pc.....	788
list rotary.....	789
M{데이터}.....	790
M{데이터}={식}.....	791
M{데이터}->.....	791
M{데이터}->{주소 정의}.....	792
M{데이터}->{데이터 구조체 요소}.....	794
M{데이터}->{자기 참조 정의}.....	795
M{변수 목록}.....	796
M{변수 목록}={식}.....	797
M{변수 목록}->.....	798
M{변수 목록}->{주소 정의}.....	799
M{변수 목록}->{데이터 구조체 요소}.....	801
M{변수 목록}->{자기 참조 정의}.....	802
open forward.....	803
open inverse.....	805
open plc.....	806
open prog.....	807
open rotary.....	809
open subprog.....	809
out{상수}.....	811
p.....	811
P{데이터}.....	813
P{데이터}={식}.....	813
P{변수 목록}.....	814
P{변수 목록}={식}.....	815
pause.....	816
pause plc.....	816
q.....	817
Q{데이터}.....	818
Q{데이터}={식}.....	818
Q{변수 목록}.....	819
Q{변수 목록}={식}.....	820

r.....	821
R{데이터}	822
R{데이터}={식}	823
R{변수 목록}.....	824
R{변수 목록}={식}.....	825
reboot.....	826
resetverbose	827
resume plc.....	827
rotfree.....	828
run.....	828
S	828
save	830
setverbose.....	830
size.....	831
start[{상수}].....	832
step	833
step plc.....	833
stop	834
string{상수}.....	834
t.....	834
type	835
undefine	836
undefine all.....	836
v	836
vers	837

POWER PMAC 프로그램 명령 사양..... 838

abort.....	838
abs	839
abs({벡터 목록})	840
adisable.....	840
{축}{데이터}[{축}{데이터}...].....	841
{축}{데이터}:{데이터}[{축}{데이터}:{데이터}...].....	842
{축}{데이터}~{데이터}[{축}{데이터}~{데이터}...].....	843
{축}{데이터}[{축}{데이터}...][{벡터}{데이터}[{벡터}{데이터}...].....	844
begin	846
break	847
bstart	848
bstop	848
C{데이터}{대입 연산자}{식}.....	849
call{데이터}.....	850
callsub{데이터}.....	851
case{상수}:	852
cclr{상수}.....	853
ccmode0.....	854
ccmode1.....	854
ccmode2.....	854
ccmode3.....	855
ccr{데이터}	855
cexec{상수}	856
circle.....	856

cmd	857
continue.....	859
cout:{데이터}	860
cset{상수}.....	861
cskip{상수}	861
{데이터 구조체 요소}{대입 연산자}{식}	862
D{데이터}	863
D{데이터}{대입 연산자}{식}.....	864
ddisable.....	865
default:	866
delay{데이터}.....	866
disable.....	867
disable bgcplc.....	868
disable plc	869
disable rticplc.....	869
dkill.....	870
do	871
dread	872
dtogread	872
dwel{데이터}.....	873
else	874
enable	874
enable bgcplc	875
enable plc.....	876
enable rticplc	876
F{데이터}.....	877
frax	878
fread.....	879
G{데이터}	880
gosub{데이터}	881
goto{데이터}	882
hold	883
home	884
homez	885
I{데이터}{대입 연산자}{식}	886
if({조건})	887
inc	888
inc({벡터 목록}).....	889
jog+	890
jog-	891
jog/	892
jog={데이터}	893
jog:{데이터}	894
jog^{데이터}.....	895
jogret	896
jogret={데이터}	897
{JOG 명령}{데이터}.....	898
kill.....	900
L{데이터}{대입 연산자}{식}	901
lh\	902
lh<	903
lh>	904

lhpurge	906
linear	906
M{데이터}	907
M{데이터}{대입 연산자}{식}	908
N{상수}	909
N{데이터}	910
nofrax	911
normal{벡터}{데이터}{[벡터]{데이터}...]}	911
nxyz	912
P{데이터}{대입 연산자}{식}	913
ause	914
pause plc	915
pclear	916
pload	916
pmatch	917
pread	917
pset{축}{데이터}{[축]{데이터}...]}	918
pstore	919
pvt{데이터}	919
Q{데이터}{대입 연산자}{식}	920
R{데이터}{대입 연산자}{식}	921
rapid	922
read	923
resume	924
resume plc	925
return	926
run	926
S{데이터}	928
send	928
sendall	930
sendallcmds	931
sendallsystemcmds	931
spline{데이터}	931
start	932
step	934
step plc	936
stop	936
switch({식})	937
system	938
T{데이터}	940
ta{데이터}	941
td{데이터}	942
tm{데이터}	943
tread	944
ts{데이터}	945
tset{데이터}	945
txyz	946
vread	946
while({조건})	947

POWER PMAC 프로젝트 강화 스크립트 언어 명령	949

스크립트 전처리 명령	949
#define {식별자}	949
#define {식별자} {토큰 문자열}	949
#define {식별자} {토큰 문자열} \	950
#define {식별자} ({인수 목록}) {토큰 문자열} [N]	951
#elif	952
#else	952
#endif	953
#if	953
#ifdef	954
#ifndef	954
#undef {식별자}	955
스크립트 변수 선언	956
csglobal	956
global	957
local	958
ptr	959
프로그램 및 서브루틴 선언	961
open forward	961
open inverse	962
open plc {식별자}	963
open prog {식별자}	963
open subprog {식별자}	964
sub:{식별자}	965
서브 프로그램 및 서브루틴 호출 명령	967
call {식별자}	967
callsub sub.{식별자}	968
POWER PMAC 스크립트 수학 기능 사양	970
연산자	970
산술 연산자	970
+	970
-	970
*	971
/	971
%	971
비트 단위의 논리 연산자	973
&	973
.....	973
^	974
<<	974
>>	975
표준 대입 연산자	976
=	976
+=	976

-=	976
*=	976
/=	977
%=	977
&=	977
=	977
=	978
>>=	978
<<=	978
++	978
--	978
동기 대입 연산자	979
==	979
+=	979
-=	980
*=	980
/=	980
+=	980
--	981
&=	981
=	981
=	981
조건 비교 연산자	982
==	982
!=	982
<	982
>	983
<=	983
>=	983
~	983
!~	984
<>	984
!>	984
!<	984
논리 연산자	985
&&	985
	985
!	985
함수	986
스칼라 함수	986
~	986
abs	987
acos	987
acosd	988
acosh	988
asin	989
asind	989
asinh	990
atan	990
atan2	991
atan2d	991
atand	992

atanh	993
cbrt	993
ceil	994
cos	994
cosd	995
cosh	995
exp	996
exp2	996
floor	997
int	997
isnan	998
ln	998
log	998
log10	999
log2	1000
madd	1000
pow	1001
qnrtd	1001
qrrtd	1002
randx	1002
rem	1003
rint	1003
rnd	1004
seed	1005
sin	1006
sincos	1006
sincosd	1007
sind	1007
sinh	1008
sgn	1008
sqrt	1009
tan	1009
tand	1010
tanh	1010
벡터 함수	1011
sum	1011
sumprod	1012
vcopy	1013
vmadd	1013
vscale	1014
매트릭스 산술 함수	1016
mdet	1016
minv	1017
mminor	1018
mmadd	1018
mmul	1019
msolve	1020
mtrans	1022
변환 행렬 수학 함수	1023
tinit	1023
tprop	1023
EtherCAT 네트워크 함수	1025
ecatsdo	1025

문자열 함수.....	1026
sprintf	1026
strcat	1027
strchr	1028
strcmp	1029
strcpy	1029
strcspn.....	1030
strlen	1031
strncat	1031
strncmp	1032
strncpy.....	1032
strpbrk	1033
strrchr.....	1033
strspn	1034
strstr	1035
strtod	1035
strtolower.....	1036
strtoupper	1037
문자 버퍼 함수.....	1038
memcpy.....	1038
memset	1038
POWER PMAC I/O 주소 오프셋.....	1040
칩 선택 라인의 개요	1040
PMAC2 스타일 Servo 카드의 UMAC.....	1041
PMAC2 스타일 MACRO 카드의 UMAC 주소.....	1042
PMAC2 스타일 I/O 카드의 UMAC 주소	1043
PMAC2 스타일 공유 메모리 카드의 UMAC 주소	1044
PMAC3 스타일 카드의 UMAC 주소.....	1045
Turbo 및 Power PMAC 의 UMAC 주소 지정의 개요	1046
Power PMAC ASIC 레지스터 요소 주소	1048
DSPGATE1(PMAC2 스타일 Servo ASIC) 레지스터 요소	1049
DSPGATE2(PMAC2 스타일 MACRO/I/O ASIC) 레지스터 요소.....	1051
DSPGATE3(PMAC3 스타일 범용 ASIC) 레지스터 요소.....	1054
범용 I/O 레지스터 요소.....	1058
IOGATE(PMAC2 스타일 I/O ASIC) 레지스터 요소	1058
ACC-28E(A/D 컨버터) 레지스터 요소	1059
ACC-36E(A/D 컨버터) 레지스터 요소	1061
ACC-59E(A/D 및 D/A 컨버터) 레지스터 요소.....	1062
POWER PMAC 의 I/O 포인터 선언의 권장 사항	1064
UMAC I/O 카드.....	1064

ACC-28E A/D 카드 0	1064
ACC-28E A/D 카드 1	1064
ACC-28E A/D 카드 2	1064
ACC-28E A/D 카드 3	1065
ACC-28E A/D 카드 4	1065
ACC-28E A/D 카드 5	1065
ACC-28E A/D 카드 6	1065
ACC-28E A/D 카드 7	1065
ACC-28E A/D 카드 8	1066
ACC-28E A/D 카드 9	1066
ACC-28E A/D 카드 10	1066
ACC-28E A/D 카드 11	1066
ACC-28E A/D 카드 12	1066
ACC-28E A/D 카드 13	1067
ACC-28E A/D 카드 14	1067
ACC-28E A/D 카드 15	1067
디지털 I/O 카드 0	1068
디지털 I/O 카드 1	1069
디지털 I/O 카드 2	1070
디지털 I/O 카드 3	1071
디지털 I/O 카드 4	1072
디지털 I/O 카드 5	1073
디지털 I/O 카드 6	1074
디지털 I/O 카드 7	1075
디지털 I/O 카드 8	1076
디지털 I/O 카드 9	1077
디지털 I/O 카드 10	1078
디지털 I/O 카드 11	1079
디지털 I/O 카드 12	1080
디지털 I/O 카드 13	1081
디지털 I/O 카드 14	1082
디지털 I/O 카드 15	1083
POWER PMAC TURBO 'I' 변수와 동등한 기능	1084
동등한 기능 관련 주의점	1084
글로벌 'I' 변수	1085
모터 'I' 변수	1088
서보 게인의 방정식	1095
데이터 개더링 'I' 변수	1096
ADC 다중 분리 'I' 변수	1096
좌표계의 'I' 변수	1097
PMAC2 스타일 MACRO IC 의 'I' 변수	1099

PMAC2 스타일 Servo IC의 'I' 변수	1100
엔코더 변환 테이블의 'I' 변수	1101
<i>Turbo PMAC 메소드의 수치 \$0: 소프트웨어에 따른 1/T 엔코더의 보간.....</i>	<i>1101</i>
<i>Turbo PMAC 메소드의 수치 \$1: Acc-28E 16 비트 ADC 변환.....</i>	<i>1101</i>
<i>Turbo PMAC 메소드의 수치 \$2: Y 레지스터의 가로 읽기, 최대 변화 필터 없음....</i>	<i>1101</i>
<i>Turbo PMAC 메소드의 수치 \$3: Y 레지스터의 가로 읽기, 최대 변화 필터 있음....</i>	<i>1102</i>
<i>Turbo PMAC 메소드의 수치 \$4: 타임 베이스 변환.....</i>	<i>1102</i>
<i>Turbo PMAC 메소드의 수치 \$5: 적분에 의한 Acc-28E 16 비트 ADC 변환.....</i>	<i>1102</i>
<i>Turbo PMAC 메소드의 수치 \$6: Y/X 레지스터의 가로 읽기, 최대 변화 필터 없음.</i>	<i>1103</i>
<i>Turbo PMAC 메소드의 수치 \$7: Y/X 레지스터의 가로 읽기, 최대 변화 필터 있음.</i>	<i>1103</i>
<i>Turbo PMAC 메소드의 수치 \$8: 인크리멘탈 엔코더의 병렬 확장.....</i>	<i>1103</i>
<i>Turbo PMAC 메소드의 수치 \$9/\$A/\$B: 트리거에 의한 타임 베이스 변환.....</i>	<i>1103</i>
프리즈 상태(Turbo PMAC 메소드 수치= \$9):	1104
작동 상태(Turbo PMAC 메소드 수치= \$B):	1104
작동 상태(Turbo PMAC 메소드 수치= \$A):	1104
<i>Turbo PMAC 메소드의 수치 \$C: 인크리멘탈 엔코더의 확장 없음.....</i>	<i>1104</i>
<i>Turbo PMAC 메소드의 수치 \$D: 로 패스 필터</i>	<i>1104</i>
<i>Turbo PMAC 메소드의 수치 \$E: 항목의 덧셈 또는 뺄셈</i>	<i>1105</i>
<i>Turbo PMAC 메소드의 수치 \$F/\$0: 고분해능 사인파 엔코더에 의한 보간.....</i>	<i>1106</i>
<i>Turbo PMAC 메소드의 수치 \$F/\$1: 고분해능 사인파 엔코더에 의한 보간 진단....</i>	<i>1106</i>
<i>Turbo PMAC 메소드의 수치 \$F/\$2: 바이트 폭의 가로 읽기, 최대 변화 필터 없음..</i>	<i>1107</i>
<i>Turbo PMAC 메소드의 수치 \$F/\$3: 바이트 폭의 가로 읽기, 최대 변화 필터 있음..</i>	<i>1107</i>
<i>Turbo PMAC 메소드의 수치 \$F/\$4: 리졸버에 의한 아크탄젠트 변환.....</i>	<i>1107</i>
POWER PMAC TURBO에 권장하는 'M' 변수와 동등한 기능.....	1108
동등한 기능 관련 주의점	1108
다양한 글로벌 변수	1108
서보 사이클 카운터	1109
PMAC2 Servo IC 레지스터	1109
모터 Status 비트	1111
모터 동작 레지스터	1112
모터 축 정의 레지스터	1113
다중 분리 ADC 레지스터	1113
좌표계 타이머	1114
좌표계의 계산된 동작의 종료 레지스터	1114
좌표계의 Status 비트	1115

좌표계의 타임 베이스 변수.....	1115
UMAC I/O 액세스러의 'M' 변수	1116
POWER PMAC UBUS32의 사양.....	1117
백플레인 커넥터의 핀 배열.....	1117
펌웨어의 갱신 이력	1118
V2.0 펌웨어 갱신(2014년 12월).....	1118
V1.6 펌웨어 갱신(2014년 2월).....	1121
V1.5 펌웨어 갱신(2012년 6월).....	1124
V1.4 펌웨어 갱신(2011년 9월).....	1126
V1.3 펌웨어 갱신(2011년 1월).....	1128

POWER PMAC 명령 구문 요약

이 장에서는 Power PMAC의 온라인(즉시 실행됨) 명령 및 버퍼 프로그램 명령 이 2 가지 명령을 예로 들어 Power PMAC 명령 구문의 개요를 설명합니다. 각 명령 및 수학적 기능에 대한 자세한 내용은 각 장에서 설명합니다.

주의

- Power PMAC의 스크립트 언어 구문에서는 대소문자 구별은 하지 않습니다.
- 특별히 설명이 있는 경우를 제외하고, 공백 문자는 Power PMAC 구문에서는 무시됩니다.
- 구문 중 이탤릭체로 표시된 문자 또는 기호는 실제 명령에는 포함되지 않지만 이탤릭체로 기술된 항목을 반드시 선택해야 합니다.
- **{ } - { }**로 묶은 이탤릭체 항목에는 정의에 맞는 내용을 지정할 수 있습니다.
- **[] - []**로 묶은 이탤릭체 항목은 생략할 수 있습니다.
- **[{item}...]** - 구문 중의 생략 기호는 직전의 항목을 반복할 수 있음을 의미합니다.
- **[.{item}]** - 구문에서 마침표를 사용하여 범위를 지정할 수 있음을 의미합니다.

정의

- **상수** - 일정한 수치입니다.
- **변수** - 가변하는 수치의 실체입니다.
- **함수** - 후속되는 소괄호에 포함되는 하나 이상의 인수에 대해 수학적 처리를 합니다.
- **연산자** - 값의 전후에 위치하여 수치 연산 또는 논리 연산을 합니다.
- **식** - 수치를 평가하는 상수, 변수, 함수 및 연산자의 일련의 조합입니다.
- **대입 연산자** - 선행하는 변수에 임의의 값을 대입하는 수학 기호로 표준 대입 연산자와 동기 대입 연산자가 있습니다.
- **표준 대입 연산자** - 식에 대입되는 연산자로 식을 평가합니다.
- **동기 대입 연산자** - 프로그램된 다음 동작이 실제로 실행될 때까지 실제 대입을 지연시킵니다.
- **데이터** - 괄호가 없는 상수 또는 괄호로 묶인 식을 나타내는 구문을 지정합니다.
- **비교 연산자** - 전후 식의 값의 상대값을 평가합니다.
- **조건** - 식의 값의 비교 결과를 바탕으로 True 또는 False로 평가합니다.
- **목록** - 여러 엔터티를 하나의 그룹으로 묶어 표현할 수 있는 정수의 상수 세트입니다. 모터 목록, 좌표계 목록, 변수 목록 등이 있습니다.
- **모터 목록** - 쉼표로 구분된 일련의 정수의 상수 또는 2 개의 마침표로 나타낸 일련의 정수의 상수(시작 상수 및 종료 상수)에 의한 범위로 이는 다음 명령으로 실행할 모터를 나타냅니다.
- **좌표계 목록** - 쉼표로 구분된 일련의 정수의 상수, 또는 2 개의 마침표로 나타낸 일련의 정수의 상수(시작 상수 및 종료 상수)에 의한 범위로 이는 다음 명령으로 실행할 좌표계를 나타냅니다.
- **변수 목록** - 쉼표로 구분되고 균등하게 구분된 변수 그룹을 나타내는 3 개의 정수 상수의 집합 또는 2 개의 마침표로 구분되고 일련의 변수의 범위를 나타내는 한 쌍의 정수의 상수입니다.
- **축** - 하나 또는 두 개의 영문자로 프로그래밍 축의 이름을 나타냅니다.
- **벡터** - 하나 또는 두 개의 영문자로 벡터 컴포넌트명을 나타냅니다.

수학적 요소

수학적인 연산자와 함수는 온라인 명령과 버퍼 프로그램 명령 양쪽에서 사용할 수 있습니다.

산술 연산자

+ – 덧셈 연산자
- – 뺄셈 또는 음수 연산자
* – 곱셈 연산자
/ – 나눗셈 연산자
% – 나머지 연산자

비트 단위 논리 연산자

& – 비트 단위 AND 연산자
| – 비트 단위 OR 연산자
^ – 비트 단위 XOR 연산자
~ – 비트 단위 NOT 연산자
<< – 왼쪽 시프트 연산자
>> – 오른쪽 시프트 연산자

표준 대입 연산자

= – 표준적인 값의 대입 연산자
+= – 덧셈 대입 연산자
-= – 뺄셈 대입 연산자
*= – 곱셈 대입 연산자
/= – 나눗셈 대입 연산자
%= – 나머지 대입 연산자
&= – 비트 단위 AND 대입 연산자
|= – 비트 단위 OR 대입 연산자
^= – 비트 단위 배타적 대입 연산자
>>= – 오른쪽 시프트 대입 연산자
<<= – 왼쪽 시프트 대입 연산자
++ – 증가 대입 연산자
-- – 감소 대입 연산자

동기 대입 연산자

== – 동기 대입 연산자
+= – 동기 덧셈 대입 연산자
-= – 동기 뺄셈 대입 연산자
***=** – 동기 곱셈 대입 연산자
/= – 동기 나눗셈 대입 연산자
&= – 동기 비트 단위 AND 대입 연산자
|= – 동기 비트 단위 OR 대입 연산자
^= – 동기 비트 단위 배타적 대입 연산자
++ – 동기 증가 대입 연산자
-- – 동기 감소 대입 연산자

조건 비교 연산자

== – 같음
!= – 같지 않음
< – 작음
> – 큼
<= – 작거나 같음
>= – 크거나 같음
~ – 거의 같음
!~ – 거의 같지 않음
<> – 같지 않음(!=)
!> – 작거나 같음(<=)
!< – 크거나 같음(>=)

논리 연산자

&& – 논리곱 연산자
|| – 논리합 연산자

스칼라 함수

abs – 절대값
acos – 삼각함수 아크코사인(각도)
acosc – 삼각함수 아크코사인(각도)
acosh – 쌍곡선 아크코사인
asin – 삼각함수 아크사인(라디안 단위)
asind – 삼각함수 아크사인(각도)
asinh – 쌍곡선 아크사인
atan – 삼각함수 아크탄젠트(라디안 단위)
atand – 삼각함수 아크탄젠트(각도)
atan2 – 삼각함수 4 상한 아크탄젠트(라디안 단위)
atan2d – 삼각함수 4 상한 아크탄젠트(각도)
atanh – 역 쌍곡선 탄젠트
cbrt – 세제곱근
ceil – 천장함수

cos – 삼각함수 코사인(인수는 라디안)
cosd – 삼각함수 코사인(인수는 각도)
cosh – 쌍곡선 코사인
exp – 밑이 e 인 거듭제곱(e^x)
exp2 – 밑이 2 인 거듭제곱(2^x)
floor – 바닥함수
int – 32 비트 정수화 함수
isnan – NaN(숫자가 아님) 확인
ln – 자연로그(대체 구문)
log – 자연로그
log10 – 밑이 10 인 상용로그
log2 – 밑이 2 인 로그
madd – 곱셈과 덧셈의 복합
pow – 거듭제곱
qnr – 다섯제곱근
qr – 네제곱근
rem – 나머지
randx – 64 비트 난수 함수
rint – 반올림
rnd – 32 비트 난수 함수
seed – 난수 시드 발생기
sin – 삼각함수 사인(인수는 라디안)
sind – 삼각함수 사인(인수는 각도)
sinh – 쌍곡선 사인
sincos – 삼각함수 사인과 코사인의 복합(인수는 라디안)
sincosd – 삼각함수 사인과 코사인의 복합(인수는 각도)
sgn – 부호함수
sqrt – 제곱근
tan – 삼각함수 탄젠트(인수는 라디안)
tand – 삼각함수 탄젠트(인수는 각도)
tanh – 쌍곡선 탄젠트

벡터 함수

sum – SUM 함수(벡터 요소의 합)
sumprod – SUMPRODUCT 함수(벡터 요소의 곱과 합)
vcopy – 벡터 복사
vmadd – 벡터의 곱셈과 덧셈
vscale – 벡터의 스칼라 곱

매트릭스 산술 함수

mdet – 매트릭스 행렬식
minv – 역행렬
minv – 매트릭스 소행렬식
mmadd – 매트릭스의 곱셈과 덧셈
mmul – 매트릭스 곱셈
msolve – 매트릭스 답 구하기
mtrans – 매트릭스 전치

매트릭스 변환 함수

tinit – 선택한 축 변환 행렬을 초기화합니다.

tprop – 선택한 축 변환 행렬을 결합합니다.

문자(바이트) 버퍼 함수

memcpy – 문자 버퍼 메모리의 내용을 복사합니다.

memset – 문자 버퍼 메모리를 지정된 문자의 값으로 설정합니다.

문자열 조작 함수

sprintf – 서식화된 문자열 변수를 작성합니다.

strcpy – (다른 변수 또는 리터럴에서) 문자열 변수를 복사합니다.

strncpy – (다른 변수에서) 제한된 길이의 문자열 변수를 복사합니다.

strtolower – 대문자를 소문자로 변환하고 제한된 길이의 문자열 변수를 복사합니다.

strtoupper – 소문자를 대문자로 변환하고 제한된 길이의 문자열 변수를 복사합니다.

strcat – (다른 변수 또는 리터럴에서) 문자열 변수의 마지막에 추가합니다.

strncat – 제한된 길이의 텍스트를 (다른 변수에서) 문자열 변수의 마지막에 추가합니다.

strchr – 문자열 변수 중에서 문자의 최초의 인스턴스를 검색합니다.

strrchr – 문자열 변수 중에서 문자의 최후의 인스턴스를 검색합니다.

strcmp – (다른 변수 또는 리터럴에 대해) 문자열 변수를 비교합니다.

strncmp – (다른 변수에 대해) 제한된 길이의 문자열 변수를 비교합니다.

strspn – 문자열 변수 중에서 지정된 문자의 가장 긴 종속 문자열을 검색합니다.

strcspn – 문자열 변수 중에서 지정된 문자를 포함하지 않는 가장 긴 종속 문자열을 검색합니다.

strlen – 문자열 변수 중에서 문자 수를 검색합니다.

strpbrk – 문자열 변수 중에서 지정된 임의의 문자의 최초 발생을 검색합니다.

strstr – 문자열 변수 중에서 다른 문자열의 최초 발생을 검색합니다.

strtod – 문자열 변수 중의 수치를 배정도 값으로 복원합니다.

온라인 명령

온라인 명령은 Power PMAC 에 의해 즉시 실행되며 실행 후에 파기됩니다. Power PMAC 에 보낸 온라인 명령의 일련의 시퀀스를 다시 표시할 수는 없습니다.

온라인 글로벌 명령

온라인 글로벌 명령의 실행은 현재 주소 지정된 모터나 좌표계에 의존하지 않으며 또한 직전의 모터나 좌표계 목록에도 의존하지 않습니다.

주소 지정 모드 명령

#{상수} – 특정한 모터를 현재의 주소 모터로 지정합니다.
– 현재 주소 지정된 모터 번호를 호스트에게 통지합니다.
&{상수} – 특정한 좌표계를 현재의 주소 좌표계로 지정합니다.
& – 현재 주소 지정된 좌표계를 호스트에게 통지합니다.

주소 지정의 비모달(Non-modal) 명령

#{목록} – 목록화된 모든 모터에 대해 직후의 모터 고유 명령을 실행합니다.
#* – Power PMAC 내에 있는 모든 모터에 대해 직후의 모터 고유 명령을 실행합니다.
&{목록} – 목록화된 모든 좌표계에 대해 직후의 좌표계 고유 명령을 실행합니다.
&* – Power PMAC 내에 있는 모든 좌표계에 대해 직후의 좌표계 고유 명령을 실행합니다.

일반적인 글로벌 명령

\$\$\$ – Power PMAC 을 리셋하여 저장된 설정값으로 복원합니다.
\$\$\$** – Power PMAC 을 리셋하여 공장 출하 설정으로 다시 초기화합니다.
reboot – Power PMAC 의 컴퓨터 및 제어 어플리케이션을 다시 기동합니다.
save – 활성 메모리 설정을 비휘발성 플래시 메모리에 복사합니다.
backup – 저장 설정 요소의 현재값을 통지합니다.
#{상수}> – 특정한 모터의 축 정의를 통지합니다.
undefine all – 모든 좌표계의 축 정의를 삭제합니다.
cx – 직후의 명령을 1 행의 PLC 프로그램으로 실행합니다.
echo{상수} – 명령 에코 모드를 설정합니다.
echo – 현재의 명령 에코 모드를 통지합니다.
string{상수} – 사용자 공유 메모리 버퍼 내의 지정된 위치에서 시작하는 문자열 변수의 내용을 통지합니다.

글로벌 구성 쿼리 명령

cpu – 프로세서 타입을 통지합니다.
date – 사용 중인 펌웨어 버전의 출시일을 통지합니다.
vers – 사용 중인 펌웨어 버전의 개정 번호를 통지합니다.
size – 사용자 버퍼로 확보된 메모리 용량을 통지합니다.
free – 사용자 버퍼로 확보된 이용 가능한(미사용) 메모리 용량을 통지합니다.
brickacver – 사용 중인 Power Brick AC 앰프의 펌웨어 버전의 개정 번호를 통지합니다.
bricklvver – 사용 중인 Power Brick LV 앰프의 펌웨어 버전의 개정 번호를 통지합니다.

글로벌 변수 명령

D{데이터} – 'D' 변수의 값을 호스트에게 통지합니다.
D{데이터}={식} – 'D' 변수의 값을 식의 값으로 설정합니다.
D{목록} – 'D' 변수의 값을 호스트에 통지합니다.
D{목록}={식} – 'D' 변수의 값을 식의 값으로 설정합니다.
I{데이터} – 'I' 변수의 값을 호스트에 통지합니다.
I{데이터}-> – 'I' 변수의 정의 내용을 호스트에 통지합니다.
I{데이터}={식} – 'I' 변수의 값을 식의 값으로 설정합니다.
I{목록} – 'I' 변수의 값을 호스트에 통지합니다.
I{목록}-> – 'I' 변수의 정의 내용을 호스트에 통지합니다.
I{목록}={식} – 'I' 변수의 값을 식의 값으로 설정합니다.
L{데이터} – 'L' 변수의 값을 호스트에 통지합니다.
L{데이터}={식} – 'L' 변수의 값을 식의 값으로 설정합니다.
L{목록} – 'L' 변수의 값을 호스트에 통지합니다.
L{목록}={식} – 'L' 변수의 값을 식의 값으로 설정합니다.
M{데이터} – 'M' 변수의 값을 호스트에 통지합니다.
M{데이터}-> – 'M' 변수의 정의 내용을 호스트에 통지합니다.
M{데이터}={식} – 'M' 변수의 값을 식의 값으로 설정합니다.
M{목록} – 'M' 변수의 값을 호스트에 통지합니다.
M{목록}-> – 'M' 변수의 정의 내용을 호스트에 통지합니다.
M{목록}={식} – 'M' 변수의 값을 식의 값으로 설정합니다.
P{데이터} – 'P' 변수의 값을 호스트에 통지합니다.
P{데이터}={식} – 'P' 변수의 값을 식의 값으로 설정합니다.
P{목록} – 'P' 변수의 값을 호스트에 통지합니다.
P{목록}={식} – 'P' 변수의 값을 식의 값으로 설정합니다.
R{데이터} – 'R' 변수의 값을 호스트에 통지합니다.
R{데이터}={식} – 'R' 변수의 값을 식의 값으로 설정합니다.
R{목록} – 'R' 변수의 값을 호스트에 통지합니다.
R{목록}={식} – 'R' 변수의 값을 식의 값으로 설정합니다.
{데이터 구조체 요소} – 특정한 데이터 구조체 요소의 값을 호스트에 통지합니다.
{데이터 구조체 요소}.a – 특정한 데이터 구조체 요소의 주소를 호스트에 통지합니다.
{데이터 구조체 요소}={식} – 특정한 데이터 구조체 요소의 값을 식의 값으로 설정합니다.

버퍼 제어 명령

open prog{상수}[[{상수}]][{상수}] – 임의의 기본값 이외의 스택 오프셋과 행의 점프 레이블의 테이블 크기를 지정하여 프로그램 명령 입력을 위해 활성 메모리상에 지정한 번호의 모션 프로그램 버퍼를 엽니다.
open plc{상수}[[{상수}]][{상수}] – 임의의 기본값 이외의 스택 오프셋과 행의 점프 레이블의 테이블 크기를 지정하여 프로그램 명령 입력을 위해 활성 메모리상에 지정한 번호의 PLC 프로그램 버퍼를 엽니다.
open subprog{상수}[[{상수}]][{상수}] – 임의의 기본값 이외의 스택 오프셋과 행의 점프 레이블의 테이블 크기를 지정하여 프로그램 명령 입력을 위해 활성 메모리상에 지정한 번호의 서브 프로그램 버퍼를 엽니다.
close – 현재 열려 있는 프로그램 버퍼를 닫습니다.
close all buffers – 다른 통신 스레드에서 열린 버퍼를 포함하여 열려 있는 프로그램 버퍼를 닫습니다.
delete all lookahead – 모든 좌표계의 룩 어헤드 버퍼를 삭제합니다.
delete all rotary – 모든 좌표계의 로터리 모션 프로그램의 버퍼 내용을 삭제합니다.

버퍼 쿼리 명령

list prog{상수}[, [{상수}]][, {상수}] – 임의의 시작 행 번호와 행 수를 지정하여 활성 메모리 내의 지정된 모션 프로그램의 버퍼 내용을 통지합니다.

list plc{상수}[, [{상수}]][, {상수}] – 임의의 시작 행 번호와 행 수를 지정하여 활성 메모리 내의 지정된 PLC 프로그램의 버퍼 내용을 통지합니다.

list subprog{상수}[, [{상수}]][, {상수}] – 임의의 시작 행 번호와 행 수를 지정하여 활성 메모리 내의 지정된 서브 프로그램의 버퍼 내용을 통지합니다.

스크립트 PLC 프로그램 제어 명령

enable plc{목록} – 지정된 번호의 PLC 프로그램을 프로그램 선두부터 실행 시작합니다.

disable plc{목록} – 지정된 번호의 PLC 프로그램을 프로그램 최종 행에서 실행 정지합니다.

pause plc{목록} – 지정된 번호의 PLC 프로그램을 현재의 실행 행에서 일시 정지합니다.

resume plc{목록} – 지정된 번호의 PLC 프로그램을 프로그램 일시 정지 부분부터 실행 재개합니다.

C PLC 프로그램 제어 명령

enable bgcplc{목록} – 지정된 백그라운드의 C PLC 프로그램 실행을 시작합니다.

enable rticplc – 포어그라운드의 C PLC 프로그램 실행을 시작합니다.

disable bgcplc{목록} – 지정된 백그라운드의 C PLC 프로그램 실행을 정지합니다.

disable rticplc – 포어그라운드의 C PLC 프로그램 실행을 정지합니다.

온라인 좌표계 명령

특별히 명시된 경우를 제외하고 좌표계 명령은 하나의 주소 지정 좌표계에서 동작합니다. 단, 직전에 여러 좌표계(&1, 2, 4r 등)의 목록이 있는 경우는 제외합니다. 여러 좌표계를 목록화한 경우에도 주소 지정한 좌표계에는 영향을 미치지 않도록 주의하십시오. 모든 좌표계를 동시에 주소 지정하기 위해 &* 구문을 사용할 수 있습니다(예를 들어, &*a는 모든 좌표계의 프로그램을 Abort 합니다).

축 정의 명령

축 정의 명령 앞에 여러 개의 좌표계 목록을 기재할 수 없다는 점에 주의하십시오.

#{상수}>{축 정의} – 가능한 스케일 계수 및 오프셋으로 지정된 모터를 하나 이상의 축에 할당합니다.

예

```
#1->X
#1->3333.333C
#1->8660X-5000Y
#2->5000X+8660Y+5000
```

#{상수}>I – 지정된 모터를 인버스 키네마틱스 축으로 할당합니다.

#{상수}>S[{상수}] – 지정된 모터를 주축으로 할당합니다.

#{상수}>0 – 좌표계 내에 있는 지정된 모터의 축 정의를 삭제합니다.

undefine – 좌표계 내의 모든 모터 축의 정의를 삭제합니다.

타임 베이스 명령

%{상수} – 주소 지정(또는 목록화)된 좌표계에 대해 타임 베이스 값을 지정합니다.

% – 주소 지정(또는 목록화)된 좌표계의 현재의 타임 베이스 값을 통지합니다.

모터 제어 명령

enable – 좌표계에 축으로 할당된 모든 모터의 서보 제어를 Enable(클로즈드 루프, 앰프 Enable)로 설정합니다.

disable – 좌표계에 축으로 할당된 모든 모터의 서보 제어를 Diabale(오픈 루프, 앰프 Disable)로 설정합니다.

ddisable – 브레이크의 작동 시간을 부여하기 위해 좌표계에 축으로 할당된 모든 모터의 서보 제어를 지연시켜 Disable(오픈 루프, 앰프 Disable)로 설정합니다.

adisable – 좌표계(Abort 와 동등) 내의 모든 모터가 매끄럽게 정지하고 그 다음으로 서보 제어(오픈 루프, 앰프 Disable)의 Disable 이 지연하여 브레이크가 완전히 연동할 시간을 지정합니다.

프로그램 제어 명령

r – 주소 지정한(또는 목록화된) 좌표계에서 현재 선택된 행부터 모션 프로그램을 실행합니다(비생략형 구문: **run**).

q – 주소 지정한(또는 목록화된) 좌표계에서 모션 프로그램의 연산을 현재 선택된 행에서 정지하고 **r** 또는 **s** 명령에 따른 재개를 대기 상태로 설정합니다(비생략형 구문: **pause**).

s[{{상수}}] – (스텝) 주소 지정한(또는 목록화된) 좌표계에서 현재 선택된 행부터 모션 프로그램을 시작하고 1 행(또는 지정된 행 수)을 실행한 후에 일시 정지합니다(비생략형 구문: **step**).

b[{{상수}}] – 지정된 모션 프로그램의 선두(상수에 소수 부분이 있는 경우에는 지정된 행 레이블)를 프로그램 카운터로 설정하고 실행 대기 상태로 설정합니다(비생략형 구문: **begin**).

stop – 주소 지정한 좌표계 또는 목록화된 좌표계의 모션 프로그램 연산을 현재 선택된 행에서 정지하고 현재 선택된 행을 그 프로그램의 선두행으로 설정합니다.

start[{{상수}}] – 주소 지정한(또는 목록화된) 좌표계에서 현재의 모션 프로그램 또는 지정된 모션 프로그램을 선두행부터 실행 시작합니다.

a – (Abort) 주소 지정한(또는 목록화된) 좌표계에서 현재 선택된 행에서 모션 프로그램 연산을 정지하고 좌표계의 모든 모터를 즉시 감속 시작합니다(비생략형 구문: **abort**).

h – (유지 정지) 주소 지정한(또는 목록화된) 좌표계에서 좌표계의 타임 베이스 값을 0 으로 하고 피드 홀드를 실행하여 **r** 또는 **s** 명령에 따른 재개를 기다립니다(비생략형 구문: **hold**).

l – (급정지) 주소 지정한(또는 목록화된) 좌표계에서 룩 어헤드 버퍼에서 가속도 제한에 접촉하지 않는 최단 정지 처리를 실행합니다.

< – (역행) 주소 지정한(또는 목록화된) 좌표계에서 룩 어헤드 버퍼 내의 역행 동작을 시작합니다.

> – (순행) 주소 지정한(또는 목록화된) 좌표계에서 룩 어헤드 버퍼 내의 순행 동작을 재개합니다.

cpx – 직후의 명령을 1 행의 모션 프로그램으로 즉시 실행합니다.

통지 명령

주의: 다음 좌표계 통지 명령의 경우 명령 직전에 좌표계 지정을 기재해야 합니다. 좌표계를 지정하지 않은 경우 해당 명령은 모터 통지 명령으로 처리됩니다.

&{목록}p – 목록 내 좌표계의 모든 활성 축의 현재 위치를 사용자 단위로 통지합니다.

&{목록}d – 목록 내 좌표계의 모든 활성 축의 명령 위치를 사용자 단위로 통지합니다.

&{목록}v – 목록 내 좌표계의 모든 활성 축의 현재 속도를 사용자 단위로 통지합니다.

&{목록}f – 목록 내 좌표계의 모든 활성 축의 위치 편차를 사용자 단위로 통지합니다.

&{목록}t – 목록 내 좌표계의 모든 버퍼링 축의 대상 위치를 사용자 단위로 통지합니다.

&{목록}g – 목록 내 좌표계의 모든 버퍼링 축의 동작에서 남은 이동량을 사용자 단위로 통지합니다.

&{목록}? – 좌표계의 Status 워드값을 통지합니다.

좌표계 변수 명령

Q{데이터} – 주소 지정한 좌표계의 'Q' 변수값을 호스트에게 통지합니다.

Q{데이터}={식} – 식과 대입 연산자를 사용하여 주소 지정한 좌표계의 'Q' 변수값을 설정합니다.

Q{목록} – 주소 지정한 좌표계의 'Q' 변수값을 호스트에게 통지합니다.

Q{목록}={식} – 주소 지정한 좌표계의 'Q' 변수값을 식의 값으로 설정합니다.

버퍼 제어 명령

open forward[,{상수}][,{상수}] – 임의의 기본값 이외의 스택 오프셋과 행의 점프 레이블의 테이블 크기를 지정하여 프로그램 명령 입력을 위해 활성 메모리상에 주소 지정한 좌표계에 관한 포워드 키네마틱스 버퍼를 엽니다.

open inverse[,{상수}][,{상수}] – 임의의 기본값 이외의 스택 오프셋과 행의 점프 레이블의 테이블 크기를 지정하여 프로그램 명령 입력을 위해 활성 메모리상에 주소 지정한 좌표계에 관한 인버스 키네마틱스 버퍼를 엽니다.

define rotary {상수}[,{상수}] – 임의의 기본값 이외의 스택 오프셋과 행의 점프 레이블의 테이블 크기를 지정하여 최초의 상수에 따라 지정된 바이트 크기를 갖는 주소 지정한 좌표계에 관한 로터리 모션 프로그램 버퍼를 정의합니다.

open rotary – 주소 지정한 좌표계에 관해 이미 정의된 로터리 모션 프로그램 버퍼를 입력용으로 엽니다.

close – 현재 열려 있는 고정 프로그램 버퍼를 닫습니다.

close rotary – 현재 열려 있는 로터리 모션 프로그램 버퍼를 닫습니다.

clear rotary – 버퍼 자체를 유지하고 주소 지정한 좌표계에 대한 로터리 모션 프로그램 버퍼 내용을 삭제합니다.

clear prog {상수} – 고정 모션 프로그램 버퍼를 삭제합니다.

clear plc {상수} – PLC 프로그램 버퍼를 삭제합니다.

rotfree – 주소 지정한 좌표계에 관한 로터리 모션 프로그램 버퍼 내에서 실행하지 않은 행 수를 통지합니다.

delete rotary – 주소 지정한 좌표계에 관한 로터리 모션 프로그램 버퍼를 삭제합니다.

define lookahead {상수} – 지정된 수의 모션 세그먼트의 주소 지정한 좌표계에 관한 룩 어헤드 버퍼를 정의합니다.

delete lookahead – 주소 지정한 좌표계에 관한 룩 어헤드 버퍼를 제거합니다.

버퍼 쿼리 명령

list forward[,{상수}][,{상수}] – 임의의 시작 행 번호와 행 수를 지정하고 활성 메모리 상에 주소 지정한 좌표계에 관한 포워드 키네마틱스 버퍼의 내용을 통지합니다.

list inverse[,{상수}][,{상수}] – 임의의 시작 행 번호와 행 수를 지정하고 활성 메모리 상에 주소 지정한 좌표계에 관한 인버스 키네마틱스 버퍼의 내용을 통지합니다.

list rotary – 주소 지정한 좌표계에 관한 로터리 모션 프로그램 버퍼의 실행하지 않은 행 내용을 통지합니다.

list pc – 프로그램 카운터 행부터 현재의 모션 프로그램 내용을 통지합니다.

list apc – Abort(명령 전송 또는 실패) 시에 프로그램 카운터 행에서 Abort 한 모션 프로그램 내용을 통지합니다.

축 명령

pmatch – 현재의 모터 위치에서 다음에 동작할 축의 시작 위치를 계산합니다.

pstore – **pload** 를 사용해서 향후 복원에 대비하여 **pset** 의 축/모터 오프셋 값을 저장합니다.

pload – **pstore** 를 사용해서 이전에 저장한 **pset** 의 축/모터 오프셋 값을 복원합니다.

온라인 모터 명령

직전에 여러 모터가 목록화(예: **#1..5hm**)되지 않은 한 모터 명령은 단일 주소 모터에 대해 실행됩니다. 여러 모터를 목록화한 경우라도 주소 지정 모터에는 영향을 미치지 않도록 유의하십시오. 모든 모터를 동시에 주소 지정하려면 **#*** 구문을 사용합니다. (예: **#*k** 는 모든 축 Kill)

통지 명령

p – 주소 지정한(또는 목록화된) 모터의 현재 위치를 모터 단위로 통지합니다.

d – 주소 지정한(또는 목록화된) 모터의 명령 위치를 모터 단위로 통지합니다.

v – 주소 지정한(또는 목록화된) 모터의 현재 속도를 모터 단위/밀리초로 통지합니다.

f – 주소 지정한(또는 목록화된) 모터의 위치 편차를 모터 단위로 통지합니다.

#{목록}? – 목록화된 모터의 Status 워드값을 통지합니다.

JOG 명령

j+ – 주소 지정한(또는 목록화된) 모터를 양의 방향으로 무한히 JOG 이송합니다(비생략형 구문: **jog+**).

j- – 주소 지정한(또는 목록화된) 모터를 음의 방향으로 무한히 JOG 이송합니다(비생략형 구문: **jog-**).

j/ – 주소 지정한(또는 목록화된) 모터의 JOG 이송을 정지하고 위치 제어를 복원합니다(비생략형 구문: **jog/**).

j= – 주소 지정한(또는 목록화된) 모터를 이전 프로그램(사전 JOG) 위치까지 JOG 이송합니다(비생략형 구문: **jog=**).

j={상수} – 주소 지정한(또는 목록화된) 모터를 지정된 위치까지 JOG 이송합니다(비생략형 구문: **jog={상수}**).

j=={상수} – 주소 지정한(또는 목록화된) 모터를 지정된 위치까지 JOG 이송하고, 해당 위치를 새로운(사전 JOG) 위치로 설정합니다(비생략형 구문: **jog=={상수}**).

j=* – 주소 지정한(또는 목록화된) 모터를 [ProgJogPos] 요소 위치에서 지정된 위치까지 JOG 이송합니다(비생략형 구문: **jog=***).

j:{상수} – 주소 지정한(또는 목록화된) 모터를 현재의 명령 위치에서부터 지정된 거리까지 JOG 이송합니다(비생략형 구문: **jog:{상수}**).

- j:*** – 주소 지정한(또는 목록화된) 모터를 현재의 명령 위치에서부터 [ProgJogPos] 요소에서 지정된 거리까지 JOG 이송합니다(비생략형 구문: **jog:***).
- j^상수** – 주소 지정한(또는 목록화된) 모터를 현재 위치에서부터 지정된 거리까지 JOG 이송합니다(비생략형 구문: **jog^상수**).
- j^*** – 주소 지정한(또는 목록화된) 모터를 현재 위치 [ProgJogPos] 요소에서 지정된 거리까지 JOG 이송합니다(비생략형 구문: **jog^***).
- jog** – 주소 지정한(또는 목록화된) 모터에 대해 현재 속도로(일반적으로는 오픈 루프에서 이동) 무한 JOG 이송을 시작합니다.

트리거까지 JOG 이송 명령

- j=상수** – 주소 지정한(또는 목록화된) 모터를 이전에 프로그램한(사전 JOG) 위치까지 JOG 이송하여 트리거를 검출한 경우에는 트리거 위치에서 지정 거리의 JOG 이송을 합니다(비생략형 구문: **jog=상수**).
- j{상수}상수** – 주소 지정한(또는 목록화된) 모터를 지정된 위치까지 JOG 이송하여 트리거를 검출한 경우에는 트리거 위치에서 지정 거리의 JOG 이송을 합니다(비생략형 구문: **jog={상수}상수**).
- j=상수** – 주소 지정한(또는 목록화된) 모터를 [ProgJogPos] 요소 내에서 지정된 위치까지 JOG 이송하여 트리거를 검출한 경우는 트리거 위치에서 지정 거리의 JOG 이송을 합니다(비생략형 구문: **jog=상수**).
- j:{상수}상수** – 주소 지정한(또는 목록화된) 모터를 현재의 지정 위치에서 지정된 거리까지 JOG 이송하여 트리거를 검출한 경우는 트리거 위치에서 지정 거리의 JOG 이송을 합니다(비생략형 구문: **jog:{상수}상수**).
- j:상수** – 주소 지정한(또는 목록화된) 모터를 현재의 명령 위치에서 [ProgJogPos] 요소 내에서 지정된 거리까지 JOG 이송하여 트리거를 검출한 경우는 트리거 위치에서 지정 거리의 JOG 이송을 합니다(비생략형 구문: **jog=상수**).
- j^상수상수** – 주소 지정한(또는 목록화된) 모터를 현재 위치에서 지정된 거리까지 JOG 이송하여 트리거를 검출한 경우는 트리거 위치에서 지정 거리의 JOG 이송을 합니다(비생략형 구문: **jog^상수상수**).
- j^상수** – 주소 지정한(또는 목록화된) 모터를 현재 위치에서 [ProgJogPos]가 지정한 거리까지 JOG 이송하여 트리거를 검출한 경우는 트리거 위치에서 지정 거리의 JOG 이송을 합니다(비생략형 구문: **jog^상수**).

일반적인 모터 명령

- \$** – 위상기준을 설정하고 필요하면 목록화된 모터의 클로즈드 루프를 닫습니다.
- hm** – 주소 지정한(또는 목록화된) 모터의 원점 서치 동작을 실시합니다.
- hmz** – 주소 지정한(또는 목록화된) 모터의 현재 명령 위치를 0으로 설정하거나 또는 절대 위치 센서를 읽어서 위치를 확립합니다.
- k** – 주소 지정한(또는 목록화된) 모터를 출력 정지로 설정합니다.
- dkill** – 주소 지정한(또는 목록화된) 모터의 서보 제어(오픈 루프, 앰프 Disable)를 지연 정지하여 브레이크가 완전히 연동할 시간을 제공합니다.
- out{상수}** – 주소 지정한(또는 목록화된) 모터에 대해 지정된 크기의 오픈 루프 서보 출력을 설정합니다.

버퍼 스크립트 프로그램 명령

이러한 버퍼 명령은 모션 프로그램과 PLC 프로그램 양쪽에서 사용할 수 있습니다.

이동 명령

{축}{데이터}{축}{데이터}... – 단순한 위치 이동 명령으로 직선 보간, Rapid 또는 스플라인 모드에서 사용할 수 있습니다.

.....예: **X1000 Y(P1) Z(P2*P3)**

{축}{데이터}:{데이터}{축}{데이터}:{데이터}... – 위치/속도 이동 명령으로 PVT 모드에서만 사용할 수 있습니다.

.....예: **X5000:750 Y3500:(P3) A(P5+P6):100**

{축}{데이터}^{데이터}{축}{데이터}^{데이터}... – 트리거 동작 명령으로 Rapid 모드에서만 사용할 수 있습니다.

{축}{데이터}{축}{데이터}...{벡터}{데이터}... – 원호 동작 명령으로 원호 보간 모드에서만 사용할 수 있습니다. 벡터가 원의 중심이 됩니다.

.....예: **X2000 Y3000 Z1000 I500 J300 K500**

.....**XX20 YY15 II10 JJ5**

{축}{데이터}{축}{데이터}... R{데이터} – 원호 동작 명령으로 주요 직교 축의 원호 보간 모드에서만 사용할 수 있습니다. R 값은 반경의 크기를 나타냅니다.

.....예: **X2000 Y3000 Z1000 R500**

dwell{데이터} – 제로(Zero) 동작 명령으로 고정 타임 베이스입니다.

delay{데이터} – 제로(Zero) 동작 명령으로 가변 타임 베이스입니다.

동작 모드 명령

linear – 혼합된 직선 보간 동작 모드를 설정합니다.

rapid – 최소 시간의 PTP 동작 모드를 설정합니다.

circle1 – 주 직교 축(X/Y/Z)에 대해 시계 방향 원호 보간 동작 모드를 설정합니다.

circle2 – 주 직교 축(X/Y/Z)에 대해 시계 반대 방향 원호 보간 동작 모드를 설정합니다.

circle3 – 보조 직교 축(XX/YY/ZZ)에 대해 시계 방향 원호 보간 동작 모드를 설정합니다.

circle4 – 보조 직교 축(XX/YY/ZZ)에 대해 시계 반대 방향 원호 보간 동작 모드를 설정합니다.

pvt{데이터} – 위치/속도/시간의 동작 모드(포물선 형상의 속도 프로파일)를 설정하고 동작 시간을 지정합니다.

spline{데이터}{spline{데이터}{spline{데이터}}} – 3 차 B 스플라인 동작 모드를 설정하고 스플라인의 세그먼트 시간을 지정합니다.

축 속성 명령

abs[{축},{축},...] – 모든(또는 지정한) 축에 대해 절대 동작 모드를 설정합니다.

inc[{축},{축},...] – 모든(또는 지정한) 축에 대해 인크리멘탈 동작 모드를 설정합니다.

frax[{축},{축},...] – 모든(또는 지정한) 축을 벡터 피드 레이트 축으로 설정합니다.

normal{벡터}{데이터}{벡터}{데이터}... – 원호 이동 및 공구 보정용으로 법선 벡터를 평면으로 지정합니다.

pset{축}{데이터}{축}{데이터}... – 축의 위치를 지정된 값으로 설정합니다.

pstore – 향후 복원용으로 현재의 축/모터의 오프셋 값을 저장합니다.

pload – 이전에 저장한 축/모터의 오프셋 값을 사용하기 위해 복원합니다.

pclr – 좌표계의 모든 축 위치를 0으로 설정합니다.

pmatch – 현재의 모터 위치에서 프로그램되어 다음에 동작할 축의 시작 위치를 계산합니다.

이동 속성 명령

F{데이터} – 직선 보간 및 원호 보간 동작 시의 이송 속도를 지정합니다.

tm{데이터} – 직선 보간 및 원호 보간 동작 시의 동작 시간을 지정합니다.

ta{데이터} – 직선 보간 및 원호 보간 동작 시의 동작 가속/감속 시간을 지정합니다.

td{데이터} – 직선 보간 및 원호 보간 동작 시의 최종적인 감속 시간을 지정합니다.

ts{데이터} – 직선 보간 및 원호 보간 동작 시의 가속/감속의 S 커브 시간을 지정합니다.

nxyz{벡터}{데이터}[{벡터}{데이터}...] – 3D 공구 보정용으로 면 법선 벡터를 지정합니다.

txyz{벡터}{데이터}[{벡터}{데이터}...] – 3D 공구 보정용으로 공구 기준 벡터를 지정합니다.

커터(공구) 직경 보정 명령

ccmode0 – 공구 직경 보정(2 차원 또는 3 차원)을 OFF 로 설정합니다.

ccmode1 – 좌측에 대한 2 차원 공구 직경 보정을 ON 으로 설정합니다.

ccmode2 – 우측에 대한 2 차원 공구 직경 보정을 ON 으로 설정합니다.

ccmode3 – 3 차원 공구 직경 보정 좌측 오프셋을 시작합니다.

ccr{데이터} – 2 차원 공구 직경 보정 반경을 지정합니다.

변수 대입 명령

I{데이터}{대입 연산자}{식} – 대입 연산자를 사용해서 식의 값을 지정된 'I' 변수에 대입합니다.

P{데이터}{대입 연산자}{식} – 대입 연산자를 사용해서 식의 값을 지정된 'P' 변수에 대입합니다.

Q{데이터}{대입 연산자}{식} – 대입 연산자를 사용해서 식의 값을 지정된 'Q' 변수에 대입합니다.

M{데이터}{대입 연산자}{식} – 대입 연산자를 사용해서 식의 값을 지정된 'M' 변수에 대입합니다.

L{데이터}{대입 연산자}{식} – 대입 연산자를 사용해서 식의 값을 지정된 'L' 변수에 대입합니다.

R{데이터}{대입 연산자}{식} – 대입 연산자를 사용해서 식의 값을 지정된 'R' 변수에 대입합니다.

C{데이터}{대입 연산자}{식} – 대입 연산자를 사용해서 식의 값을 지정된 'C' 변수에 대입합니다.

D{데이터}{대입 연산자}{식} – 대입 연산자를 사용해서 식의 값을 지정된 'D' 변수에 대입합니다.

{데이터 구조체 요소}{대입 연산자}{식} – 식 및 대입 연산자를 사용해서 지정된 데이터 구조 요소의 값을 설정합니다.

N{데이터} – 프로그램 연산 시에 {데이터} 값을 [Coord[x].Ncalc]에 대입하고 또한 후속 이동 실행 시에 [Coord[x].Nsync]에 대입합니다.

프로그램의 로직 제어

N{상수}: – 프로그램 점프 명령의 대상 위치를 나타내는 수치 행 레이블입니다.

goto{데/이/타} – 동일한 프로그램 내의 지정된 수치 행 레이블로 점프하며 복귀는 없습니다.

gosub{데/이/타} – 동일한 프로그램 내의 지정된 수치 행 레이블로 점프하고 복귀 시에 점프 백합니다.

callsub{데/이/타} – 동일한 프로그램 내의 지정된 수치 행 레이블로 점프하고 복귀 시에 점프 백합니다. 서브루틴에 대한 로컬 변수를 가능한 서브루틴에 전달합니다.

call{데/이/타} – 지정된 서브 프로그램(및 수치 행 레이블)으로 점프하고 복귀 시에 점프 백합니다. 로컬 변수를 가능한 서브루틴에 전달합니다.

return – call 한 서브루틴을 호출한 프로그램 행으로 점프 백하고 프로그램을 계속 실행시킵니다.

read({문자}[{문자}][{문자}[{문자}...]) – 인수(지정된 한 개 또는 두 개의 영문자로 이어지는 값)를 호출 행에서 서브루틴/서브 프로그램으로 읽어옵니다.

G{데/이/타} – **Gnnn[.mmm]**은 **CALL(Coord[x].Gprog).nnnnmmm**의 대체 명령([Coord[x].Gprog]로 지정된 프로그램에 의해 G 코드 처리용 서브루틴을 제공합니다.)

M{데/이/타} – **Mnnn[.mmm]**은 **CALL(Coord[x].Mprog).nnnnmmm**의 대체 명령([Coord[x].Mprog]로 지정된 프로그램에 의해 M 코드 처리용 서브루틴을 제공합니다.)

T{데/이/타} – **Tnnn[.mmm]**은 **CALL(Coord[x].Tprog).nnnnmmm**의 대체 명령([Coord[x].Tprog]로 지정된 프로그램에 의해 T 코드 처리용 서브루틴을 제공합니다.)

D{데/이/타} – **Dnnn[.mmm]**은 **CALL(Coord[x].Dprog).nnnnmmm**의 대체 명령([Coord[x].Dprog]로 지정된 프로그램에 의해 D 코드 처리용 서브루틴을 제공합니다.)

if({조건}){명령}[{명령}...] – 1행의 작업을 조건부로 실행합니다.

```
if({조건})
{
  {명령}
  [{명령}...]
}
```

– 여러 행의 작업을 조건부로 실행합니다.

else{명령}[{명령}...] – 앞의 IF 문의 조건이 거짓인 경우 1행의 작업을 실행합니다.

```
else
{
  {명령}
  [{명령}...]
}
```

– 앞의 IF 문의 조건이 거짓인 경우 여러 행의 처리를 실행합니다.

while({조건}){명령}[{명령}...] – 1행의 처리를 조건부로 반복합니다.

```
while({조건})
{
  {명령}
  [{명령}...]
}
```

– 여러 행의 처리를 조건부로 반복합니다.

do {명령}{명령...} while({조건}) – 1 행의 처리를 조건부로 반복합니다. 항상 1 회만 실행됩니다.

do

{

{명령}

{명령...}

}

while({조건}) – 1 행의 처리를 조건부로 반복합니다. 항상 1 회만 실행됩니다.

switch({식})

{

case{상수}:{명령}{명령...}[break]

[case{상수}:{명령}{명령...}[break]

[default:{명령}{명령...}

}

– 복수 분기의 조건부 실행 구조체

cset{상수} – 지정된 조건부 실행 플래그를 설정합니다.

cclr{상수} – 지정된 조건부 실행 플래그를 클리어합니다.

cexe{상수} – 지정된 플래그가 설정된 경우 나머지 행을 조건부로 실행합니다.

cskip{상수} – 지정된 플래그가 설정된 경우 나머지 행을 조건부로 건너뜁니다.

스크립트 PLC의 실행 제어

enable plc{목록} – 지정된 PLC 프로그램을 선두 행부터 실행 시작합니다.

disable plc{목록} – 지정된 PLC 프로그램을 최종 행에서 실행 정지합니다.

pause plc{목록} – 지정된 PLC 프로그램을 현재 실행 중인 행에서 일시 정지합니다.

resume plc{목록} – 지정된 PLC 프로그램을 일시 정지된 행부터 재개합니다.

C PLC의 실행 제어

enable bgcplc{목록} – 지정된 백그라운드의 C PLC 프로그램 실행을 시작합니다.

enable rticplc – 포어그라운드의 C PLC 프로그램 실행을 시작합니다.

disable bgcplc{목록} – 지정된 백그라운드의 C PLC 프로그램 실행을 정지합니다.

disable rticplc – 포어그라운드의 C PLC 프로그램 실행을 정지합니다.

포트 통신

send{상수},{문자열},{식}... – 서식화된 텍스트 메시지를 지정된 포트에 송신합니다.

다이렉트 모터 명령

cout{데이터}{목록} – 주소 지정한(또는 목록화된) 모터에 대해 지정한 크기의 오픈 루프 명령 출력을 설정합니다.

home{목록} – 주소 지정한(또는 목록화된) 모터의 원점 서치 동작을 실시합니다.

homez{목록} – 주소 지정한(또는 목록화된) 모터의 무동작 원점 복귀를 실행하거나 또는 절대 위치 센서를 읽어서 위치를 확립합니다.

jog+{목록} – 주소 지정한(또는 목록화된) 모터를 양의 방향으로 무한히 JOG 이송합니다.

jog-{목록} – 주소 지정한(또는 목록화된) 모터를 음의 방향으로 무한히 JOG 이송합니다.

jogl{목록} – 주소 지정한(또는 목록화된) 모터의 JOG 이송을 정지하고 위치 제어를 복원합니다.

jog{목록}={데이터} – 주소 지정한(또는 목록화된) 모터를 지정된 위치까지 JOG 이송합니다.

- jog[{{목록}}]:{{데이터}}** – 주소 지정한(또는 목록화된) 모터를 현재의 명령 위치에서 지정된 거리까지 JOG 이송합니다.
- jog[{{목록}}]^{{데이터}}** – 주소 지정한(또는 목록화된) 모터를 현재 위치에서 지정된 거리까지 JOG 이송합니다.
- jog[{{목록}}]={{데이터}}^{{데이터}}** – 주소 지정한(또는 목록화된) 모터를 지정된 위치까지 JOG 이송합니다. 트리거가 발생하면 대신 트리거 위치에서 지정된 거리를 JOG 이송합니다.
- jog[{{목록}}]:{{데이터}}^{{데이터}}** – 주소 지정한(또는 목록화된) 모터를 현재의 명령 위치에서 지정된 거리를 JOG 이송합니다. 트리거를 검출한 경우는 트리거 위치에서 지정된 거리를 JOG 이송합니다.
- jog[{{목록}}]^{{데이터}}^{{데이터}}** – 주소 지정한(또는 목록화된) 모터를 현재 위치에서 지정된 거리를 JOG 이송합니다. 트리거가 발생하면 대신 트리거 위치에서 지정된 거리를 JOG 이송합니다.
- jogret[{{목록}}]** – 주소 지정한(또는 목록화된) 모터를 이전에 프로그램한(사전 JOG) 위치까지 JOG 이송합니다.
- jogret[{{목록}}]={{데이터}}** – 주소 지정한(또는 목록화된) 모터를 지정된 위치까지 JOG 이송하고, 해당 위치를 새로운 사전 JOG 위치로 설정합니다.
- kill[{{목록}}]** – 주소 지정한(또는 목록화된) 모터를 즉시 출력 정지합니다(서보 제어를 Disable 로 설정합니다).
- dkill[{{목록}}]** – 주소 지정한(또는 목록화된) 모터의 지연 정지를 실시하여(서보 제어를 Dioble 로 설정) 브레이크가 완전히 연동할 시간을 제공합니다.

좌표계의 다이렉트 명령

- enable[{{목록}}]** – 주소 지정한(또는 목록화된) 좌표계에 할당된 모든 모터의 서보 제어를 Enable 로 설정합니다.
- disable[{{목록}}]** – 주소 지정한(또는 목록화된) 좌표계에 할당된 모든 모터의 서보 제어를 Disable 로 설정합니다(출력 정지합니다).
- ddisable[{{목록}}]** – 주소 지정한(또는 목록화된) 좌표계에 할당된 모든 모터에 대해 서보 제어를 지연 Disable(출력 정지)해서 브레이크가 완전히 연동할 시간을 제공합니다.
- abort[{{목록}}]** – 현재의 선택 포인트에서 주소 지정한(또는 목록화된) 좌표계의 모션 프로그램 연산을 현재 선택된 행에서 정지하고 좌표계 내의 모든 축을 즉시 감속합니다.
- adisable[{{목록}}]** – 현재 선택된 행부터 주소 지정한(또는 목록화된) 좌표계에서 모션 프로그램 연산을 정지하고 좌표계의 모든 축의 감속을 즉시 시작합니다. 다음으로 각 모터의 지연 Dioble 을 실시합니다.
- hold[{{목록}}]** – 주소 지정한(또는 목록화된) 좌표계에 대해 피드 홀드를 실행하여 **R** 또는 **S** 명령에 의한 재개를 기다립니다.
- pause[{{목록}}]** – 현재 선택된 행에서 주소 지정한(또는 목록화된) 좌표계의 모션 프로그램 연산을 현재 선택된 행에서 일시 정지하고 재개를 기다립니다.
- resume[{{목록}}]** – 주소 지정한(또는 목록화된) 좌표계에서 모션 프로그램을 중단한 행부터 실행 재개합니다.
- run[{{목록}}]** – 주소 지정된(또는 목록화된) 좌표계에서 현재 선택된 행부터 모션 프로그램을 실행합니다.
- start[{{목록}}]:{{데이터}}** – 주소 지정한(또는 목록화된) 좌표계에서 현재의(또는 지정된) 모션 프로그램의 실행을 선두 행부터 실행 시작합니다.
- step[{{목록}}]** – 주소 지정된(또는 목록화된) 좌표계에서 현재 선택된 행부터 시작하고, 1 행 또는 지정된 행 수를 실행한 후에 일시 정지합니다.

- stop[*{목록}*]** – 주소 지정한(또는 목록화된) 좌표계의 모션 프로그램 연산을 현재 선택된 행에서 정지하고, 해당 행을 프로그램의 선두 행으로 설정합니다.
- lh** – (급정지) 주소 지정한(또는 목록화된) 좌표계에서 록 어헤드 버퍼의 가속도 제한에 저촉되지 않는 최단 정지 처리를 실행합니다.
- lh<** – (역행) 주소 지정한(또는 목록화된) 좌표계에서 록 어헤드 버퍼의 역행 동작을 시작합니다.
- lh>** – (순행) 주소 지정한(또는 목록화된) 좌표계에서 록 어헤드 버퍼의 순행 동작을 재개합니다.

프로그램 쿼리 명령

- dread** – 주소 지정한 좌표계의 모든 활성 축의 명령 위치를 사용자 단위로 통지합니다.
- pread** – 주소 지정한 좌표계의 모든 활성 축의 현재 위치를 사용자 단위로 통지합니다.
- vread** – 주소 지정한 좌표계의 모든 활성 축의 (필터 처리한) 속도를 사용자 단위로 통지합니다.
- fread** – 주소 지정한 좌표계의 모든 활성 축의 위치 편차를 사용자 단위로 통지합니다.
- tread** – 주소 지정한 좌표계의 모든 버퍼링 축의 대상 위치를 사용자 단위로 통지합니다.
- dtogread** – 주소 지정한 좌표계의 모든 버퍼링 축의 남은 이동량을 사용자 단위로 통지합니다.

Acc84E[i].저장 데이터 구조체 요소(V1.6 2014.03)

ACC-84E 는 다양한 시리얼-엔코더 프로토콜을 지원하는 4 채널 시리얼-엔코더 인터페이스 보드입니다. 해당 레지스터는 **Acc84E[i].데이터** 구조체에 따라 액세스할 수 있습니다. 인덱스값 *i* 는 0 부터 15 의 범위에서 카드의 하드웨어 주소 DIP 스위치로 결정됩니다. 이 장에서는 ACC-84E 의 저장 설정 요소를 설명합니다.

Acc84E[i].멀티채널 설정 요소

시리얼 엔코더 인터페이스의 일부 기능은 보드의 모든 채널에서 공통적으로 기능합니다. 다음의 저장 설정 요소는 보드의 채널 전체에 동일하게 반영됩니다.

Acc84E[i].SerialEncCtrl

설명: 시리얼 엔코더 구성의 액세스러리 제어 워드

범위: \$0..\$FFFFFFF

단위: 비트 필드

기본값: 프로토콜 고유

Acc84E[i].SerialEncCtrl 은 ACC-84E 의 시리얼 엔코더 인터페이스용 멀티채널 설정으로 구성되는 풀 워드 요소입니다. 다음과 같은 구성 요소로 구성됩니다(이는 독립된 요소로 액세스할 수 없습니다).

구성 요소	스크립트 비트	16 진수 자리#	C 비트	기능
<i>SerialClockMDiv</i>	23 - 16	1 - 2	31 - 24	시리얼 클록의 선형 분주 계수
<i>SerialClockNDiv</i>	15 - 12	3	23 - 20	시리얼 클록 지수부의 분주 계수
(예약)	11 - 10	4	19 - 18	(추후 사용하기 위해 예약)
<i>SerialTrigClockSel</i>	09	4	17	시리얼 트리거 소스 선택
<i>SerialTrigEdgeSel</i>	08	4	16	시리얼 트리거 소스 엣지 선택
<i>SerialTrigDelay</i>	07 - 04	5	15 - 12	소스 엣지에서 시리얼 트리거 지연
<i>SerialProtocol</i>	03 - 00	6	11 - 08	시리얼 엔코더 프로토콜 선택(읽기 전용)

구성 요소 *SerialClockMDiv* 는 IC 고정 100MHz 클록 주파수에서 어떻게 중간 클록 주파수를 생성할지를 제어합니다. 뒤에서 설명하는 것처럼 산출 결과인 시리얼 엔코더의 클록 주파수는 이 중간 클록 주파수에서 구성 요소 *SerialClockNDiv* 를 사용하여 생성됩니다.

중간 클록 주파수 f_{int} 를 구하는 식은 다음과 같습니다.

$$f_{int}(\text{MHz}) = \frac{100}{M + 1}$$

여기에서 M 은 짧은 정수인 *SerialClockMDiv* 를 나타냅니다. 이 8 비트의 구성 요소는 0 에서 255 의 값으로 설정할 수 있습니다. 따라서 결과적으로 발생하는 중간 클록 주파수의 범위는 100MHz 부터 392kHz 까지입니다.

구성 요소 *SerialClockNDiv* 는 *SerialClockMDiv* 로 설정된 중간 클록 주파수에서 생성되는 최종 시리얼 엔코더의 클록 주파수를 제어합니다. 이 최종 주파수 f_{ser} 를 구하는 식은 다음과 같습니다.

$$f_{ser}(\text{MHz}) = \frac{f_{int}(\text{MHz})}{2^N} = \frac{100}{(M + 1) * 2^N}$$

여기에서 N 은 짧은 정수인 *SerialClockNDiv* 를 나타냅니다. 이 4 비트의 구성 요소는 0~15 의 값으로 설정할 수 있습니다. 따라서 결과적으로 발생하는 2^N 제수는 1 부터 32,768 의 값이 됩니다.

명시적인 클록 신호를 갖춘 대부분의 시리얼 엔코더의 프로토콜의 경우 얻은 주파수가 ACC-84E 에서 엔코더로 출력되는 클록 신호의 주파수입니다. 명시적인 클록 신호가 없는 "자가 클록" 프로토콜에서는 이 주파수가 입력 샘플링 주파수가 되어 입력 비트 전송 속도 f_{bit} 보다 20~25 배 높아집니다. 개별 프로토콜에 대한 자세한 내용은 설명서를 참조하십시오.

구성 요소 *SerialTrigClockSel* 은 엔코더를 트리거하는 Power PMAC 클록 신호를 제어합니다. 이 싱글 비트 구성 요소가 0 으로 설정될 경우 엔코더는 페이즈 클록으로 트리거되고 1 로 설정되는 경우 엔코더는 서보 클록으로 트리거됩니다. PMAC 에서 전류(commutation)를 수행하기 위해 엔코더 피드백이 필요한 경우에 페이즈 클록에서 트리거되어야 하며 PMAC 에서 전류(commutation)를 수행하면 서보 클록에서 트리거할 수 있습니다.

구성 요소 *SerialTrigEdgeSel* 는 *SerialTrigClockSel* 에서 선택된 클록 신호(페이즈 또는 서보)의 어느 엣지에서 트리거 프로세스를 시작할지 제어합니다. 이 싱글 비트 구성 요소를 0 으로 설정하는 경우 트리거 프로세스는 상승 엣지에서 시작하고 1 로 설정하는 경우 트리거 프로세스는 하강 엣지에서 시작합니다.

관련된 위상 페이즈 또는 서보 클록 신호의 하강 엣지 직후에 프로세서가 엔코더 데이터 읽기 동작을 시작할 수 있도록 인터럽트가 발생하며 Power PMAC 소프트웨어는 엔코더 데이터가 사용 가능하다고 판단합니다. 트리거에서는 지연이 최소화되는 것이 바람직하므로 하강 엣지 전에 완전히 데이터를 전송할 수 있다면 클록의 상승 엣지에서 트리거하는 것이 더 좋습니다. 이것이 불가능한 경우에는 트리거 프로세스를 시작하기 위해 하강 엣지를 사용해야 합니다.

구성 요소 **SerialTrigDelay** 는 시리얼 엔코더의 클록 단위이며 지정한 클록 엣지부터 실제로 시작하는 엔코더 응답을 트리거하는 출력 신호까지의 지연을 지정합니다. 엔코더를 트리거하는 타이밍과 Power PMAC 이 읽는 타이밍의 지연을 최소한으로 하기 위해 0 이 아닌 값을 사용할 수 있습니다.

구성 요소 **SerialProtocol** 은 IC 의 모든 채널에 어느 시리얼 엔코더의 프로토콜을 선택할지 제어합니다. 이 4 비트 구성 요소는 0 부터 15 의 값에서 설정됩니다. 이 값은 공장에서 보드에 설치된 프로토콜 인터페이스를 반영하므로 읽기 전용입니다.

다음 표에 이 구성 요소의 각 값으로 선택된 프로토콜이 나와 있습니다.

값	프로토콜	값	프로토콜	값	프로토콜	값	프로토콜
0	(예/약)	4	(예/약)	8	Panasonic	12	(예/약)
1	(예/약)	5	(예/약)	9	Mitutoyo	13	(예/약)
2	SSI	6	시그마 II/III	10	Kawasaki	14	(예/약)
3	EnDat	7	Tamagawa	11	Biss-B/C	15	(예/약)

스크립트 환경에서 사용할 경우 **Acc84E[i].SerialEncCtrl** 은 24 비트 요소입니다. C 환경에서 사용할 경우에는 상위 24 비트에 실제 데이터를 가지는 32 비트 요소가 되므로 C 환경에서의 값은 스크립트 환경에서의 값보다 256 배 커집니다.

BiSS B/C(단일 방향) 프로토콜

BiSS B 또는 BiSS C(단일 방향) 엔코더를 위한 **Acc84E[i].SerialEncCtrl** 의 대표적인 설정은 다음과 같습니다.

SerialClockMDiv: = $(100/f_{bit}) - 1$ // 시리얼 클록 주파수 = 비트 전송 주파수
SerialClockNDiv: = 0 // 주파수가 400kHz 미만이면 추가적인 나눗셈은
// 불필요
SerialTrigClockSel: = 0 // 가능하면 페이즈 클록 사용
SerialTrigEdgeSel: = 0 // 가능하면 클록 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 줄이기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$0B // BiSS 프로토콜에서 IC 프로그램

예를 들어 1.0MHz 의 비트 전송률에는 **SerialClockMDiv** = $(100/1.0) - 1 = 99$ (\$63)를 설정합니다. 그리고 지연 없이 페이즈 클록의 상승 엣지에서 트리거 하기 위해 **Acc84E[i].SerialEncCtrl** 을 \$63000B 로 설정합니다.

16 진수 자리(\$)	6				3				0				0				0				B				-	-		
스크립트 비트#	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	-	-	
C 비트#	3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	0	9	8	7-4	3-0
비트값 구성요소:	0	1	1	0	0	0	1	1	0	0	0	0	-	-	0	0	0	0	0	0	0	1	0	1	1	-	-	
	SerialClockMDiv								SerialClockNDiv				--	--	TC	TE	SerialTrigDelay				SerialProtocol							

EnDat 2.1/2.2 프로토콜

EnDat 엔코더용 **Acc84E[i].SerialEncCtrl**의 대표적인 설정은 다음과 같습니다. 시리얼 클럭 주파수는 입력 신호의 초과 샘플링을 가능하게 하기 위해 외부 클럭 주파수(비트 전송 주파수 f_{bit})보다 25 배 높아집니다.

SerialClockMDiv: = $(4/f_{bit}) - 1$ // 시리얼 클럭 주파수 = 25x 비트 전송 주파수
SerialClockNDiv: = 0 // 그 이상 나눗셈 없음
SerialTrigClockSel: = 0 // 가능하면 페이즈 클럭 사용
SerialTrigEdgeSel: = 0 // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 줄이기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$03 // EnDat 프로토콜에서 IC 프로그램

예를 들어 2.0MHz 의 비트 전송률에는 **SerialClockMDiv** = $(4/2) - 1 = 1$ (\$01)을 설정합니다. 그리고 지연 없이 페이즈 클럭의 상승 엣지에서 트리거 하기 위해 **Acc84E[i].SerialEncCtrl** 을 \$010003 으로 설정합니다.

16 진수 자리(\$)	0				1				0				0				0				3				-	-	
스크립트 비트#	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	-	-	
C 비트#	3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7-4	3-0	
비트값 구성요소:	0	0	0	0	0	0	0	0	1	0	0	0	0	0	-	-	0	0	0	0	0	0	0	1	1	-	-
	SerialClockMDiv				SerialClockNDiv				--	--	TC	TE	SerialTrigDelay				SerialProtocol										

Mitutoyo 프로토콜

Mitutoyo 시리얼 엔코더용 **Acc84E[i].SerialEncCtrl**의 대표적인 설정은 다음과 같습니다. 시리얼 클럭 주파수는 입력 신호의 초과 샘플링을 가능하게 하기 위해 외부 클럭 주파수(비트 전송 주파수 f_{bit})보다 20 배 높아집니다.

SerialClockMDiv: = $(5/f_{bit}) - 1$ // 시리얼 클럭 주파수 = 20x 비트 전송 주파수
SerialClockNDiv: = 0 // 나눗셈 없음
SerialTrigClockSel: = 0 // 가능하면 페이즈 클럭 사용
SerialTrigEdgeSel: = 0 // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 줄이기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$09 // Mitutoyo 프로토콜에서 IC 프로그램

예를 들어 2.5MHz 의 비트 전송률에는 **SerialClockMDiv** = $(5/2.5) - 1 = 1$ (\$01)을 설정합니다. 그리고 지연 없이 페이즈 클럭의 상승 엣지에서 트리거 하기 위해 **Acc84E[i].SerialEncCtrl** 을 \$010009 로 설정합니다.

16 진수 자리(\$)	0				1				0				0				0				9				-	-		
스크립트 비트#	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	-	-			
C 비트#	3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7-4	3-0		
비트값 구성요소:	0	0	0	0	0	0	0	0	1	0	0	0	0	-	-	0	0	0	0	0	0	0	1	0	0	1	-	-
	SerialClockMDiv				SerialClockNDiv				--	--	TC	TE	SerialTrigDelay				SerialProtocol											

Panasonic 프로토콜

Panasonic 엔코더용 **Acc84E[i].SerialEncCtrl** 의 대표적인 설정은 다음과 같습니다. 시리얼 클럭 주파수는 입력 신호의 초과 샘플링을 가능하게 하기 위해 외부 클럭 주파수(비트 전송 주파수 f_{bit})보다 20 배 높아집니다.

SerialClockMDiv: = $(5/f_{bit}) - 1$ // 시리얼 클럭 주파수 = 20x 비트 전송 주파수
SerialClockNDiv: = 0 // 나눗셈 없음
SerialTrigClockSel: = 0 // 가능하면 페이즈 클럭 사용
SerialTrigEdgeSel: = 0 // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 줄이기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$09 // Panasonic 프로토콜에서 IC 프로그램

예를 들어 2.5MHz 의 비트 전송률에는 **SerialClockMDiv** = $(5/2.5) - 1 = 1$ (\$01)을 설정합니다. 그리고 지연 없이 페이즈 클럭의 상승 엣지에서 트리거 하기 위해 **Acc84E[i].SerialEncCtrl** 을 \$010008 로 설정합니다.

16 진수 자리(\$)	0				1				0				0				0				8				-	-				
스크립트 비트#	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	-	-				
C 비트#	3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7-4	3-0			
비트값	0	0	0	0	0	0	0	0	1	0	0	0	0	0	-	-	0	0	0	0	0	0	0	0	1	0	0	0	-	-
구성 요소:	SerialClockMDiv								SerialClockNDiv								--	--	TC	TE	SerialTrigDelay				SerialProtocol					

SSI 프로토콜

SSI 엔코더용 **Acc84E[i].SerialEncCtrl** 의 대표적인 설정은 다음과 같습니다.

SerialClockMDiv: = $(100/f_{bit}) - 1$ // 시리얼 클럭 주파수 = 비트 전송 주파수
SerialClockNDiv: = 0 // 주파수가 400kHz 미만이면 추가적인 나눗셈은 // 불필요
SerialTrigClockSel: = 0 // 가능하면 페이즈 클럭 사용
SerialTrigEdgeSel: = 0 // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 줄이기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$02 // SSI 프로토콜에서 IC 프로그램

예를 들어 2.5MHz 의 비트 전송률에는 **SerialClockMDiv** = $(100/2.5) - 1 = 39$ (\$23)를 설정합니다. 그리고 지연 없이 페이즈 클럭의 상승 엣지에서 트리거 하기 위해 **Acc84E[i].SerialEncCtrl** 을 \$230002 로 설정합니다.

16 진수 자리(\$)	2				3				0				0				0				2				-	-
스크립트 비트#	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0	-	-
C 비트#	3 3	3 0	2 2	2 2	2 2	2 2	2 2	2 2	2 2	2 2	2 2	2 2	1 1	1 1	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7-4	3-0
비트값	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	-	-
구성 요소:	SerialClockMDiv								SerialClockNDiv								--	--	TC	TE	SerialTrigDelay				SerialProtocol	

Tamagawa 프로토콜

Tamagawa FA 코더 시리얼 엔코더용 **Acc84E[i].SerialEncCtrl** 의 대표적인 설정은 다음과 같습니다. 시리얼 클럭 주파수는 입력 신호의 초과 샘플링을 가능하게 하기 위해 외부 클럭 주파수(비트 전송 주파수 f_{bit})보다 20 배 높아집니다.

SerialClockMDiv: = $(5/f_{bit}) - 1$ // 시리얼 클럭 주파수 = 20x 비트 전송 주파수
SerialClockNDiv: = 0 // 나눗셈 없음
SerialTrigClockSel: = 0 // 가능하면 페이즈 클럭 사용
SerialTrigEdgeSel: = 0 // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 줄이기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$07 // Tamagawa 프로토콜에서 IC 프로그램

예를 들어 2.5MHz 의 비트 전송률에는 **SerialClockMDiv** = $(5/2.5) - 1 = 1$ (\$01)을 설정합니다. 그리고 지연 없이 위상 클럭의 상승 엣지에서 트리거 하기 위해 **Acc84E[i].SerialEncCtrl** 을 \$010007 로 설정합니다.

16 진수 자리(\$)	0				1				0				0				0				7				-	-	
스크립트 비트#	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-	
C 비트#	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0	
비트값 구성요소:	0	0	0	0	0	0	0	1	0	0	0	0	-	-	0	0	0	0	0	0	0	1	1	1	1	-	-
	SerialClockMDiv								SerialClockNDiv				--	--	TC	TE	SerialTrigDelay				SerialProtocol						

Yaskawa 시그마 II/III/V 프로토콜

Yaskawa II/III/V 엔코더용 **Acc84E[i].SerialEncCtrl** 의 대표적인 설정은 다음과 같습니다.

SerialClockMDiv: = 0 // 100MHz 시리얼 클럭 주파수 = 25x 비트 전송 주파수
SerialClockNDiv: = 0 // 나눗셈 없음
SerialTrigClockSel: = 0 // 가능하면 페이즈 클럭 사용
SerialTrigEdgeSel: = 0 // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 줄이기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$06 // Yaskawa II/III/V 프로토콜에서 프로그램

예를 들어 표준 4.0MHz 의 비트 전송률에서는 100MHz 의 시리얼 클럭 주파수를 사용하고 지연 없이 페이즈 클럭의 상승 엣지에서 트리거하기 위해 **Acc84E[i].SerialEncCtrl** 을 \$000006 으로 설정합니다.

16 진수 자리(\$)	0				0				0				0				0				6				-	-
스크립트 비트#	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C 비트#	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
비트값 구성 요소:	0	0	0	0	0	0	0	0	0	0	0	0	-	-	0	0	0	0	0	0	0	1	1	0	-	-
	SerialClockMDiv								SerialClockNDiv				--	--	TC	TE	SerialTrigDelay				SerialProtocol					

Acc84E[j].싱글 채널 설정 요소

시리얼 엔코더 인터페이스의 일부 기능은 채널마다 별도로 설정할 수 있습니다. 채널 인덱스 j 는 0~3 범위에서 보드의 하드웨어 채널 1~4 에 대응합니다. 채널 인덱스는 대응하는 하드웨어 채널 번호보다 1 이 작은 값입니다.

Acc84E[j].Chan[j].SerialEncCmd

설명: 시리얼 엔코더 명령의 액세스리 채널 제어 워드

범위: 0 ..\$FFFFFFF

단위: 비트 필드

기본값: (프로토콜 고유)

Acc84E[j].Chan[j].SerialEncCmd 는 각 채널의 시리얼 엔코더 인터페이스 명령 정보로 구성되는 풀 워드 요소입니다. 특정 프로토콜, 트리거 타이밍과 클록 주파수가 멀티채널 요소 **Acc84E[j].SerialEncCtrl** 로 결정된다는 점에 주의하십시오.

다음 구성 요소로 **Acc84E[j].Chan[j].SerialEncCmd** 는 구성됩니다. (이는 개별적인 요소로 액세스할 수 없습니다.)

구성 요소	스크립트 비트	16 진수 자리#	C 비트	기능
<i>SerialEncCmdWord</i>	23 - 16	1 - 2	31 - 24	시리얼 엔코더 출력 명령
<i>SerialEncParity</i>	15 - 14	3	23 - 22	시리얼 엔코더 패리티 타입
<i>SerialEncTrigMode</i>	13	3	21	시리얼 트리거 모드: 연속 또는 단발
<i>SerialEncTrigEna</i>	12	3	20	시리얼 트리거 유효
<i>SerialEncGtoB</i>	11	4	19	시리얼 SSI 데이터의 그레이-바이너리 변환 제어
<i>SerialEncEna/SerialEncDataReady</i>	10	4	18	시리얼 엔코더 회로 유효(쓰기) 시리얼 엔코더 수신 데이터 준비 완료(읽기)
<i>SerialEncStatusBits</i>	09 - 06	4 - 5	17 - 14	상태 비트의 시리얼 엔코더 SPI 번호
<i>SerialEncNumBits</i>	05 - 00	5 - 6	13 - 08	시리얼 엔코더 비트 길이 제어

프로토콜 고유의 방법으로 시리얼 엔코더로 송신하는 명령값을 정의하기 위해 8비트의 구성 요소 *SerialEncCmdWord* 를 사용합니다. 이 값은 어플리케이션을 사용하여 엔코더를 리셋하는 것과 같은 다양한 기능으로 변경할 수 있습니다. 단, 모든 프로토콜이 이 값을 필요로 하는 것은 아닙니다.

2 비트 구성 요소 *SerialEncParity* 는 수신 데이터 패킷(패리티 확인을 지원하는 프로토콜용)을 위해 필요한 패리티 타입을 정의합니다. 0 의 값은 패리티를 지정하지 않습니다. 1 의 값은 홀수 패리티를 지정합니다. 2 의 값은 짝수 패리티를 지정합니다. (추후 사용하기 위해 3 의 값이 예약됩니다.)

1 비트 구성 요소 **SerialEncTrigMode** 는 엔코더를 반복 샘플링할지 아니면 1 회만 샘플링할지 여부를 지정합니다. 0 의 값에서는 연속적인 샘플링(멀티채널 요소 **Acc84E[i].SerialEncCtrl** 로 설정되는 페이즈 사이클이나 서보 사이클마다)을 지정하고 1 의 값은 원샷 샘플링을 지정합니다.

1 비트 구성 요소 **SerialEncTrigEna** 은 엔코더를 샘플링할지 여부를 지정합니다. 0 의 값에서는 샘플링 되지 않습니다. 1 의 값에서는 엔코더 샘플링이 가능합니다. **SerialEncTrigMode** 가 0 으로 설정되고 **SerialEncTrigEna** 이 1 의 값으로 유지되면 반복 엔코더는 샘플링됩니다(멀티채널 요소 **Acc84E[i].SerialEncCtrl** 로 설정되는 페이즈 사이클이나 서보 사이클마다). 그러나 **SerialEncTrigMode** 가 1 로 설정되면 엔코더를 1 번만 샘플링합니다. 또한 ACC-84E IC 는 샘플링 후 자동으로 **SerialEncTrigEna** 을 0 으로 반환합니다.

1 비트 구성 요소 **SerialEncGtoB** 는 SSI 프로토콜로 반환되어 온 데이터를 그레이 형식에서 수치 바이너리 형식으로 변환할지 여부를 지정합니다. 0 의 값에서는 변환을 수행하지 않습니다. 1 의 값에서는 수신 데이터의 그레이-바이너리 변환을 수행합니다.

1비트 구성 요소 **SerialEncEna/SerialEncDataReady** 는 레지스터에 대한 쓰기와 읽기가 각각 따로 기능합니다. 레지스터에 쓸 때 이 비트는 **SerialEncEna** 을 나타냅니다. 그것은 시리얼 엔코더의 드라이버 회로를 유효로 합니다. 이 비트는 각 채널의 시리얼 엔코더에서 임의의 프로토콜을 사용하는 경우에는 1 로 설정해야 합니다. 동일한 신호 핀에 대체 사용이 있는 경우는 엔코더 드라이버가 대체 사용과 경합하지 않도록 이 비트는 0 으로 해야 합니다. **이 비트에 쓴 값은 다시 읽어들 수 없으므로 주의하십시오!**

레지스터에서 읽을 때 **SerialEncDataReady** Status 비트에 시리얼 데이터 수신 상태를 나타냅니다. 통지 0 은 데이터 전송 중의 유효한 새로운 데이터가 아직 준비 완료되지 않았음을 나타냅니다. 데이터를 모두 수신하여 처리했으면 1 을 통지합니다. 이는 읽기가 완료될 때까지 여러 개의 서보 사이클에 걸쳐 있는 더 느린 인터페이스에서 특히 중요합니다. 이러한 경우에는 데이터 준비 여부를 판정하기 위해 해당 비트는 폴링되어야 합니다.

4 비트 구성 요소 **SerialEncStatusBits** 는 인터페이스가 SPI 프로토콜의 엔코더에서 필요로 하는 Status 비트 수를 지정합니다. 설정의 유효 범위는 0~12 입니다.

6 비트 구성 요소 **SerialEncNumBits** 는 SSI, EnDat 또는 BiSS 프로토콜의 엔코더 인터페이스가 필요로 하는 데이터 비트 수를 지정합니다. 해당 프로토콜의 유효한 설정 범위는 12-63 입니다. 다른 프로토콜에서는 비트 수는 이와 같이 지정되지 않으며 이 값은 유효가 아닙니다. 따라서 이 구성 요소는 0 이 유지됩니다.

스크립트 환경에서 사용되는 경우 **Acc84E[i].Chan[j].SerialEncCmd** 는 24 비트 요소입니다. C 환경에서 사용할 경우에는 상위 24 비트에 실제 데이터를 가지는 32 비트 요소가 되므로 C 환경에서의 값은 스크립트 환경에서의 값보다 256 배 커집니다.

BiSS-B/C(단일 방향) 프로토콜

BiSS-B 와 BiSS-C(단일 방향) 프로토콜은 **Acc84E[i].Chan[j].SerialEncCmd** 의 **SerialEncCmdWord** 구성 요소는 위치와 상태의 데이터가 통지되었을 때 에러 검출에 사용되는 CRC 다항식을 지정합니다. 이는 4 비트에서 8 비트 CRC 다항식에 임의로 정의할 수 있는 8비트의 마스크 값 "M"입니다. 이는 특정 BiSS 엔코더에 사용된 다항식과 일치하도록 설정되어야 합니다.

마스크 비트 M_7 부터 M_0 은 CRC 다항식에서 x^8 부터 x^1 항의 계수를 각각 나타냅니다.

$$M_7x^8 + M_6x^7 + M_5x^6 + M_4x^5 + M_3x^4 + M_2x^3 + M_1x^2 + M_0x^1 + 1$$

CRC 다항식의 x^0 의 계수는 항상 1 이므로 마스크에 포함되지 않습니다. 모두 0 인 마스크는 CRC 비트가 엔코더 데이터에 포함되지 않는다는 것을 나타냅니다.

예를 들어 엔코더가 $x^6 + x^1 + 1$ CRC 다항식을 사용하는 경우(Renishaw Resolute™ 엔코더처럼) CRC 마스크 값 M 은 00100001(비트 M_5 와 M_0 이 1) 또는 \$21 로 설정합니다.

BiSS 프로토콜에서 **Acc84E[i].Chan[j].SerialEncCmd** 의 **SerialEncParity** 구성 요소는 BiSS-B 와 BiSS-C 프로토콜의 차이를 구별하기 위해 사용됩니다. 구성 요소(24 비트 요소인 비트 15)의 비트 1 은 0 일 때 BiSS-C 를 지정하며 1 일 때 BiSS-B 를 지정합니다. (BiSS-C 는 시작 비트와 위치 데이터 사이에 0 비트를 부여하지만 BiSS-B 에는 없습니다.)

구성 요소(24 비트의 요소인 비트 13)의 비트 0 은 BiSS-B 만 사용됩니다. 1 로 설정된 경우 클럭 사이클 출력을 추가로 부여하면 엔코더에서의 "멀티 사이클 데이터"(MCD) 비트를 수신합니다. 0 으로 설정된 경우 MCD 비트는 포착되지 않으며 사용되지 않습니다.

BiSS C 엔코더용 **Acc84E[i].Chan[j].SerialEncCmd** 의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord: = \$21	// $x^6 + x^1 + 1$ 의 CRC 다항식
SerialEncParity: = 0	// BiSS-C 프로토콜
SerialEncTrigMode: = 0	// 연속 트리거
SerialEncTrigEna: = 1	// 트리거 유효
SerialEncGtoB: = 0	// 그레이 코드는 EnDat 프로토콜에서는 // 지원되지 않습니다
SerialEncEna: = 1	// 드라이버 회로 유효
SerialEncStatusBits: = {엔코더 고유}	// 회신되는 위치 비트의 엔코더 고유의 // 비트 수
SerialEncNumBits: = {엔코더 고유}	// 회신되는 Status 비트의 엔코더 고유의 // 비트 수

예를 들어 36 비트의 위치 비트, 2 비트의 Status 비트 그리고 $x^6 + x^1 + 1$ 의 CRC 다항식을 갖춘 BiSSC 엔코더에서는 연속적으로 위치를 알리도록 **Acc84E[i].Chan[j].SerialEncCmd** 를 \$2114A4 로 설정합니다. (데이터 준비 완료 Status 비트가 설정되지 않은 경우 \$2110A4 로 통지됩니다.)

16 진수 자리(\$)	2				1				1				4				A				4				-	-	
스크립트 비트#	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	-	-	
C비트#	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	0	9	8	7-4	3-0
비트값	0	0	1	0	0	0	0	1	0	0	0	1	-	1	-	0	1	0	1	0	0	1	0	0	-	-	
구성 요소:	SerialEncCmdWord								Parity		TM	TE	GB Ena		Status				NumBits								

32 비트의 위치 비트, 4 비트의 Status 비트, MCD1 비트 그리고 $x^4 + x^1 + 1$ 의 CRC 다항식을 갖춘 BiSS-B 엔코더에서는 연속적인 위치를 알리도록 **Acc84E[i].Chan[j].SerialEncCmd** 를 \$09D520 으로 설정합니다. (데이터 준비 완료 Status 비트가 설정되지 않은 경우 \$09D120 으로 통지됩니다.)

16 진수 자리(\$)	0				9				D				5				2				0				-	-	
스크립트 비트#	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0	-	-	
C 비트#	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7- 4	3- 0	
비트값	0	0	0	0	1	0	0	1	1	1	0	1	-	1	-	1	0	0	0	1	0	0	0	0	0	-	-
구성 요소:	SerialEncCmdWord								Parity		TM	TE	GB Ena		Status				NumBits								

EnDat2.1/2.2 프로토콜

ACC-84E의 EnDat 인터페이스는 엔코더에 직접 송신되는 4개의 6비트 명령 코드를 지원합니다.

- 위치를 통지하기 위한 000111(\$07)(EnDat2.1)
- 엔코더를 리셋하기 위한 101010(\$2A)(EnDat2.1)
- 추가할 수 있는 정보를 가진 위치를 통지하기 위한 111000(\$38)(EnDat2.2)
- 엔코더를 리셋하기 위한 101101(\$2D)(EnDat2.2)

해당 6비트는 **Acc84E[i].Chan[j].SerialEncCmd**의 8비트의 *SerialEncCmdWord* 명령 필드의 하위 비트로 할당됩니다.



주의

EnDat 규격에서 EnDat2.2 엔코더는 EnDat2.1 명령 코드도 동일하게 처리되어야만 합니다. 그러나 EnDat2.2 규격에 대응하도록 판매된 모든 엔코더에서 이를 실행할 수는 없습니다.

나아가 EnDat2.2 엔코더의 ACC-84E(2014년 제 1/4분기 시작)는 메모리 영역 선택(MRS) 코드를 사용하여 엔코더에서 부가 정보를 얻는 컨트롤러 요청을 지원합니다. 이를 탑재하기 위해 *SerialEncCmdWord* 명령 필드는 MRS 코드를 포함합니다. (이 모드에서는 ACC-84E는 엔코더에 111000의 명령 코드-부가 정보를 가진 위치 통지-를 송신합니다.)

다음은 EnDat2.2 규격에서 지원되는 MRS 코드입니다.

- \$40 - 데이터 내용을 제외한 부가 정보 1 송신
- \$41 -진단 결과 송신
- \$42 -위치값 2 워드 1 LSB 송신
- \$43 -위치값 2 워드 2 송신
- \$44 -위치값 2 워드 3 MSB 송신
- \$45 -메모리 내용 LSB의 응답
- \$46 -메모리 내용 MSB의 응답
- \$47 -MRS 코드의 응답
- \$48 -테스트 명령의 응답
- \$49 -테스트값 워드 1 LSB 송신
- \$4A -테스트값 워드 2 송신
- \$4B -테스트값 워드 3 MSB 송신
- \$4C -온도 1 송신
- \$4D -온도 2 송신
- \$4F -부가 정보 1의 송신 정지
- \$50 - 데이터 내용을 제외한 부가 정보 2 송신
- \$51 -전류(commutation) 송신
- \$52 -가속도 송신
- \$53 -전류(commutation)와 가속도 송신
- \$54 -위치 리미트 신호 송신
- \$55 -위치 리미트 신호와 가속도 송신
- \$56 -현재 할당 안 함
- \$5F -부가 정보 2의 송신 정지

엔코더의 특정 MRS 코드 데이터 요청에 대한 응답은 엔코더 내에서 해당 데이터가 유효한지 여부에 따라 좌우됩니다. 지원된 MRS 코드에서 제공되는 부가 정보는 상태 요소 **Acc84E[i].Chan[j].SerialEncDataC** 와 **SerialEncDataD** 에서 확인할 수 있습니다.

EnDat 엔코더에서의 위치를 통지하기 위한 **Acc84E[i].Chan[j].SerialEncCmd** 의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord: = {cmd/MRS 코드}	// 명령 코드
SerialEncParity: = 0	// 패리티 확인은 EnDat 프로토콜에서는 // 지원되지 않습니다
SerialEncTrigMode: = 0	// 연속 트리거(EnDat2.2)
SerialEncTrigEna: = 1	// 트리거 유효
SerialEncGtoB: = 0	// 그레이 코드는 EnDat 프로토콜에서는 // 지원되지 않습니다
SerialEncEna: = 1	// 드라이버 회로 유효
SerialEncStatusBits: = 0	// Status 비트는 EnDat 프로토콜에서는 // 지원되지 않습니다
SerialEncNumBits: = {엔코더 고유}	// 회신되는 위치 비트의 엔코더 고유의 // 비트 수

예를 들어 37 비트의 위치 비트를 갖춘 EnDat2.2 엔코더에 대해서는 연속적으로 위치를 알리도록 **Acc84E[i].Chan[j].SerialEncCmd** 를 \$381425 로 설정합니다. (데이터 준비 완료 Status 비트가 설정되지 않은 경우 \$381025 로 통지됩니다.)

16 진수 자리(\$)	3								8				1				4				2				5				-	-
스크립트 비트#	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-				
C 비트#	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0				
비트값	-	0	1	1	1	0	0	0	-	-	0	1	-	1	-	-	-	-	1	0	0	1	0	1	-	-				
구성 요소:	SerialEncCmdWord								Parity				TM	TE	GB		Ena		Status				NumBits							

이 동일한 엔코더는 **Acc84E[i].Chan[j].SerialEncCmd** 를 \$2D3425 로 설정하고 원쇼트 모드에서 송신하는 45(\$2D)의 명령 워드로 리셋할 수 있습니다.

16 진수 자리(\$)	2				D				3				4				2				5				-	-
스크립트 비트#	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C 비트#	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
비트값	-	0	1	0	1	1	0	1	-	-	1	1	-	1	-	-	-	-	1	0	0	1	0	1	-	-
구성 요소:	SerialEncCmdWord								Parity		TM	TE	GB	Ena	Status				NumBits							

24 비트의 위치 비트를 갖춘 EnDat2.1 엔코더에 대해서는(전원 투입으로) 1 번만 위치를 알리도록 **Acc84E[i].Chan[j].SerialEncCmd** 를 \$073418 로 설정합니다. 이는 데이터가 수신될 때까지 \$073018 을 통지합니다.

16 진수 자리(\$)	0				7				3				4				1				8				-	-
스크립트 비트#	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0	-	-
C 비트#	3 1	3 0	3 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	2 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7-4	3-0
비트값	-	-	0	0	0	1	1	1	-	-	1	1	-	1	-	-	-	-	0	1	1	0	0	0	-	-
구성 요소:	SerialEncCmdWord								Parity		TM	TE	GB	Ena	Status				NumBits							

24 비트의 위치 비트를 갖춘 EnDat2.2 인크리멘탈 엔코더에 대해서는 위치 2 워드 1 의 부가 정보로 연속적으로 위치를 알릴 수 있도록 **Acc84E[i].Chan[j].SerialEncCmd** 를 \$421418 로 설정합니다. (데이터 준비 완료 Status 비트가 설정되지 않은 경우 \$421018 로 통지됩니다.)

16 진수 자리(\$)	4				2				1				4				1				8				-	-
스크립트 비트#	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-
C 비트#	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7-4	3-0
비트값	-	1	0	0	0	0	1	0	-	-	0	1	-	1	-	-	-	-	0	1	1	0	0	0	-	-
구성 요소:	SerialEncCmdWord								Parity		TM	TE	GB	Ena	Status				NumBits							

Mitutoyo 프로토콜

Mitutoyo 시리얼 엔코더용 **Acc84E[i].Chan[j].SerialEncCmd** 의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord: = \$01 // Mitutoyo 에서 위치를 통지시키는 명령 워드
SerialEncParity: = 0 // 패리티 확인은 Mitutoyo 프로토콜에서는
// 지원되지 않습니다
SerialEncTrigMode: = 0 // 연속 트리거
SerialEncTrigEna: = 1 // 트리거 유효
SerialEncGtoB: = 0 // 그레이 코드는 Mitutoyo 프로토콜에서는
// 지원되지 않습니다
SerialEncEna: = 1 // 드라이버 회로 유효
SerialEncStatusBits: = 0 // Status 비트는 Mitutoyo 프로토콜에서는
// 지원되지 않습니다
SerialEncNumBits: = 0 // 회신되는 고정 위치 비트 수

연속적인 위치 통지를 위해 **Acc84E[i].Chan[j].SerialEncCmd** 를 \$011400 으로 설정합니다. (준비 완료 Status 비트가 설정되지 않은 경우 \$011000 으로 통지됩니다.)

16 진수 자리(\$)	0				1				1				4				0				0				-	-
스크립트 비트#	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0	-	-
C 비트#	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 1	1 0	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7- 4	3- 0
비트값	0	0	0	0	0	0	0	1	0	0	1	-	1	-	-	-	-	0	0	0	0	0	0	0	0	-
구성 요소:	SerialEncCmdWord								Parity		TM	TE	GB	Ena	Status				NumBits							

SerialEncCmdWord 구성 요소를 \$89 로 설정하는 경우 8 회 송신된 후에 엔코더 중의 멀티턴 위치값이 0 으로 리셋됩니다. 이는 요소를 \$893400 으로 하여 8 회 반복하는 "원쇼트" 모드에서 수행됩니다. 리셋 동작이 수행될 때 구성 요소는 \$892000 으로 통지됩니다. 이 구성 요소가 \$9D 로 설정되는 경우 엔코더는 해당 ID 값을 통지합니다. 이는 "원쇼트" 모드에서 수행되며 IC 는 상태 요소 **Acc84E[i].Chan[j].SerialEncDataC** 에 이 값을 유지합니다.

Panasonic 프로토콜

Panasonic 시리얼 엔코더용 **Acc84E[i].Chan[j].SerialEncCmd** 의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord: = \$2A // Panasonic 멀티턴 위치의 명령 워드
SerialEncParity: = 0 // 패리티 확인은 Panasonic 프로토콜에서는
// 지원되지 않습니다
SerialEncTrigMode: = 0 // 연속 트리거
SerialEncTrigEna: = 1 // 트리거 유효
SerialEncGtoB: = 0 // 그레이 코드는 Panasonic 프로토콜에서는
// 지원되지 않습니다
SerialEncEna: = 1 // 드라이버 회로 유효
SerialEncStatusBits: = 0 // Status 비트는 Panasonic 프로토콜에서는
// 지원되지 않습니다
SerialEncNumBits: = 0 // 회신되는 고정 위치 비트 수

연속적인 위치 통지를 위해 **Acc84E[i].Chan[j].SerialEncCmd** 를 \$2A1400 으로 설정합니다. (준비 완료 Status 비트가 설정되지 않은 경우 \$2A1000 으로 통지됩니다.)

16 진수 자리(\$)	2				A				1				4				0				0				-	-	
스크립트 비트#	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0	-	-	
C 비트#	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7-4	3-0	
비트값	0	0	1	0	1	0	1	0	0	0	0	1	-	1	-	-	-	-	0	0	0	0	0	0	0	-	-
구성 요소:	SerialEncCmdWord								Parity		TM	TE	GB Ena		Status				NumBits								

이상 코드와 함께 싱글턴 위치를 통지하기 위해 **SerialEncCmdWord** 구성 요소를 \$52 로 설정하는 경우 **Acc84E[i].Chan[j].SerialEncCmd** 는 \$521400 으로 설정됩니다. (데이터 준비 완료 Status 비트가 설정되지 않은 경우 \$521000 으로 통지됩니다.) **SerialEncCmdWord** 구성 요소가 \$52 로 설정되는 경우 엔코더 ID 값도 통지됩니다.

SerialEncCmdWord 구성 요소가 \$4A, \$7A, \$DA 또는 \$F2 로 설정되는 경우 엔코더 내의 멀티턴 위치값은 0 으로 리셋됩니다. 이는 "원쇼트" 모드에서 수행해야 하며 요소를 \$4A3400, \$7A3400, \$DA3400 또는 \$F23400 으로 설정합니다. 리셋 동작을 수행할 때 구성 요소는 각각 \$4A2000, \$7A2000, \$DA2000 또는 \$F22000 을 통지합니다.

SSI 프로토콜

SSI 엔코더용 **Acc84E[i].Chan[j].SerialEncCmd** 의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord: = 0 // 명령 워드는 SSI 프로토콜에서는
//지원되지 않습니다
SerialEncParity: = ?? // 엔코더 특유의 패리티 확인
SerialEncTrigMode: = 0 // 연속 트리거
SerialEncTrigEna: = 1 // 트리거 유효
SerialEncGtoB: = ?? // 엔코더 특유의 데이터 형식
SerialEncEna: = 1 // 드라이버 회로 유효
SerialEncStatusBits: = 0 // Status 비트는 SSI 프로토콜에서는
// 지원되지 않습니다
SerialEncNumBits: = ?? // 회전되는 위치 비트의 엔코더 고유의
// 비트 수

예를 들어 짝수 패리티, 그레이 코드 형식에서의 25 비트 위치 비트를 갖춘 SSI 엔코더에 대해서는 **Acc84E[i].Chan[j].SerialEncCmd** 를 \$005C19 로 설정합니다. (데이터 준비 완료 Status 비트가 설정되지 않은 경우 \$005819 로 통지됩니다.)

16 진수 자리(\$)	0				0				5				C				1				9				-	-
스크립트 비트#	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0	-	-
C 비트#	3 1	3 0	3 2	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7-4	3-0
비트값	-	-	-	-	-	-	-	-	0	1	0	1	1	1	-	-	-	-	0	1	1	0	0	1	-	-
구성 요소:	SerialEncCmdWord								Parity		TM	TE	GB Ena		Status				NumBits							

Tamagawa FA 코더 프로토콜

Tamagawa FA 코더 시리얼 엔코더용 **Acc84E[i].Chan[j].SerialEncCmd** 의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord: = \$1A // Tamagawa 에서 위치를 통지하기 위한 명령 워드
SerialEncParity: = 0 // 패리티 확인은 Tamagawa 프로토콜에서는
// 지원되지 않습니다
SerialEncTrigMode: = 0 // 연속 트리거
SerialEncTrigEna: = 1 // 트리거 유효
SerialEncGtoB: = 0 // 그레이 코드는 Tamagawa 프로토콜에서는
// 지원되지 않습니다
SerialEncEna: = 1 // 드라이버 회로 유효
SerialEncStatusBits: = 0 // Status 비트는 Tamagawa 프로토콜에서는
// 지원되지 않습니다
SerialEncNumBits: = 0 // 회신되는 고정 위치 비트 수

연속적인 위치 통지를 위해 **Acc84E[i].Chan[j].SerialEncCmd** 를 \$1A1400 으로 설정합니다. (준비 완료 Status 비트가 설정되지 않은 경우 \$1A1000 으로 통지됩니다.)

16 진수 자리(\$)	1				A				1				4				0				0				-	-		
스크립트 비트#	2 3	2 2	2 1	2 0	1 9	1 2	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0	-	-	
C 비트#	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 1	1 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7- 4	3- 0
비트값 구성 요소:	0	0	0	1	1	0	1	0	0	0	0	1	-	1	-	-	-	-	0	0	0	0	0	0	0	0	-	-
	SerialEncCmdWord								Parity		TM	TE	GB	Ena	Status				NumBits									

SerialEncCmdWord 구성 요소를 \$BA, \$C2 또는 \$62 로 설정하는 경우 엔코더의 멀티턴 위치값은 0 으로 리셋됩니다. 이는 "원쇼트" 모드에서 수행해야 하며 요소를 \$BA3400, \$C23400 또는 \$623400 으로 설정합니다. 리셋 동작이 수행되면 구성 요소는 각각 \$BA2000, \$C22000 또는 \$622000 으로 통지될 것입니다.

Yaskawa 시그마 II/III/IV 프로토콜

Yaskawa 시그마 II/III/IV 인터페이스는 위치 통지와 이상 리셋 모드를 지원합니다. 위치 통지를 위한 명령 코드는 \$00 입니다. 이상 리셋을 위한 명령 코드는 \$04 입니다.

Yaskawa 시그마 II/III/IV 엔코더에서의 위치를 통지하기 위한 **Acc84E[i].Chan[j].SerialEncCmd** 의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord: = 0 // Yaskawa 에 위치를 통지하기 위한 명령 워드는
// 없습니다.
SerialEncParity: = 0 // 패리티 확인은 Yaskawa 프로토콜에서는
// 지원되지 않습니다
SerialEncTrigMode: = 0 // 연속 트리거
SerialEncTrigEna: = 1 // 트리거 유효
SerialEncGtoB: = 0 // 그레이 코드는 Yaskawa 프로토콜에서는
// 지원되지 않습니다
SerialEncEna: = 1 // 드라이버 회로 유효

SerialEncStatusBits:= 0 // Status 비트는 Yaskawa 프로토콜에서는
// 지원되지 않습니다
SerialEncNumBits:= 0 // 회전되는 고정 위치 비트 수

연속적인 위치 통지를 위해 **Acc84E[i].Chan[j].SerialEncCmd** 를 \$001400 으로 설정합니다. (준비 완료 Status 비트가 설정되지 않은 경우 \$001000 으로 통지됩니다.)

16 진수 자리(\$)	0								0								1								4								0								0								-	-
스크립트 비트#	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	-	-																								
C 비트#	3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7-4	3-0																							
비트값	-	-	0	0	0	0	0	0	-	-	0	1	-	1	0	0	0	0	0	0	0	0	0	0	0	-	-																							
구성 요소:	SerialEncCmdWord								Parity		TM	TE	GB Ena		Status				NumBits																															

엔코더를 리셋하기 위해 다음과 같이 구성 요소를 설정해야 합니다.

SerialEncCmdWord:= \$04 // 이상 리셋 명령 코드
SerialEncParity:= 0 // 패리티 확인은 Yaskawa 프로토콜에서는
// 지원되지 않습니다
SerialEncTrigMode:= 1 // 단일 트리거
SerialEncTrigEna:= 1 // 트리거 유효
SerialEncGtoB:= 0 // 그레이 코드는 Yaskawa 프로토콜에서는
// 지원되지 않습니다
SerialEncEna:= 1 // 드라이버 회로 유효
SerialEncStatusBits:= 4 // Yaskawa 프로토콜의 특별 리셋 명령
SerialEncNumBits:= \$01 // 리셋용 "엔코더 주소"

이상 리셋을 시작하기 위해 **Acc84E[i].Chan[j].SerialEncCmd** 는 \$043501 로 설정합니다. (준비 완료 Status 비트가 설정되지 않은 경우 \$043101 로 통지됩니다.)

16 진수 자리(\$)	0				4				3				5				0				1				-	-
스크립트 비트#	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0	-	-
C 비트#	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7-4	3-0
비트값	-	-	0	0	0	1	0	0	-	-	1	1	-	1	0	1	0	0	0	0	0	0	0	1	-	-
구성 요소:	SerialEncCmdWord								Parity		TM	TE	GB Ena		Status				NumBits							

엔코더의 이상 리셋을 시작하기 위해 이 값을 요소에 쓴 후 사용자는 다음을 해야 합니다.

1. 100msec 대기합니다.
2. 이 요소의 트리거 유효 구성 요소(비트 12)가 클리어될 때까지 대기합니다.
3. **Acc84E[i].Chan[j].SerialEncDataB** 의 사용 중 신호(비트 8)가 클리어될 때까지 대기합니다.
4. 요소에 \$003501 을 써서 이 요소의 명령 코드를 0 으로 클리어합니다.
5. 스텝 1~3 을 반복합니다.
6. 요소에 \$001400 을 써서 연속적인 위치 요구를 재개합니다.

CamTable[m].저장 데이터 구조체 요소(V1.6 2014.03)

CamTable[m].DacData[i] 캠 테이블의 토크 오프셋 명령

설명: 캠 테이블 토크 오프셋 명령 등록

범위: 부동 소수점

단위: 사용자 정의 토크 위치 단위

기본값: 0.0

CamTable[m].DacData[i]는 캠 테이블의 개별 토크 오프셋 명령 등록값을 지정합니다. 캠 테이블의 토크 오프셋 명령은 이 기능이 **CamTable[m].DacEnable**에서 유효가 되는 경우 소스 모터의 현재 위치에 따라서 인접한 등록 사이를 보간하는 방법으로 계산됩니다. 다음으로 명령은 대상 모터의 서보 루프 출력에 자동으로 가해지지만 대상 모터의 **Motor[x].CompDac** 레지스터에 쓰여지기 전에 스케일 계수항 **CamTable[m].DacSf**로 곱해집니다. 스케일 계수가 기본값인 1.0인 경우에 **DacData[i]** 등록은 대상 모터의 16비트 출력 단위입니다.

Note 최종적인 토크 명령은 대부분 D/A 컨버터에 쓰이지만 이 요소명대로 충실하게 실행할 필요는 없다는 점에 주의하십시오. 이 기능은 다른 형식의 명령 신호를 사용하는 경우와 PowerPMAC의 전류(commutation)나 전류 루프 알고리즘 입력의 토크 명령으로 사용되는 경우에도 동일하게 기능합니다.

CamTable[m].DacData[0]에는 소스 모터 위치가 **CamTable[m].X0**일 때의 토크 오프셋 명령을 지정합니다. **CamTable[m].DacData[1]**에는 소스 모터 위치가 $(X0 + Dx/Nx)$ 일 때의 토크 오프셋 명령을 지정합니다. **CamTable[m].DacData[2]**에는 소스 모터 위치가 $(X0 + 2*Dx/Nx)$ 일 때의 토크 오프셋 명령을 지정합니다. **CamTable[m].DacData[Nx-1]**에는 소스 모터 위치 $(X0 + [Nx-1]*Dx/Nx)$ 일 때의 토크 오프셋 명령을 지정합니다.

캠 테이블의 동작은 소스 모터 위치가 $(X0$ 에서 $X0 + Dx)$ 의 범위 밖인 경우에 항상 롤 오버합니다. 해당 롤 오버 기능은 항상 "재귀적"이므로 토크 오프셋은 각 사이클에서 항상 동일한 범위가 됩니다. 캠 테이블 종단에서 이 롤 오버를 유지 하기 위해 Power PMAC은 테이블 포인트 **CamTable[m].DacData[Nx-1]**과 **CamTable[m].DacData[0]** 사이를 보간합니다. **CamTable[m].DacData[Nx]** 등록이 존재하지만 아무 테이블 계산에서도 사용되지 않습니다.

CamTable[m].DacEnable 캠 테이블의 토크 출력 유효

설명: 캠 테이블 토크 출력 유효 제어

범위: 0 ..1

단위: 논리값

기본값: 0

CamTable[m].DacEnable 은 캠 테이블의 "토크" 오프셋 명령이 **DacData[i]** 등록에서 계산되어 대상 모터의 **CompDac** 레지스터에 기록할지 여부를 지정합니다. 그 기본값이 0 인 경우 토크 명령은 계산되지 않고 기록되지 않습니다. 이는 프로세서 시간을 절약하고 모터 위치를 사용하는 토크 보정 테이블이 대상 모터의 **CompDac** 레지스터에 보정을 가하지 않는 것을 의미합니다.

테이블이 유효이고 **CamTable[m].DacEnable** 이 1 로 설정된 경우 "토크" 오프셋 명령이 계산되어 대상 모터의 **CompDac** 레지스터에 기록됩니다. 이 경우 토크 보정 테이블은 동일한 대상 모터에서 사용되지 않는 것으로 합니다.

CamTable[m].DacSf 캠 테이블의 토크 스케일 계수

설명: 캠 테이블 토크 스케일 계수

범위: 부동 소수점

단위: 대상 모터 16 비트 단위/테이블 토크 단위

기본값: 1.0

CamTable[m].DacSf 는 캠 테이블에서 계산된 "토크" 오프셋 명령이 대상 모터의 서보 명령 출력 보정 레지스터(**Motor[x].CompDac**)에 기록되기 전에 명령에 곱해질 스케일 계수를 지정합니다. 그 기본값이 1.0 인 경우 테이블 토크 오프셋 등록은 부호가 있는 16 비트 출력 단위입니다.

CamTable[m].Dx 캠 테이블의 소스 위치 범위

설명: 캠 테이블 소스 위치 범위

범위: 음이 아닌 부동 소수점

단위: 소스 모터 단위

기본값: 0.0

CamTable[m].Dx 는 소스 모터 단위로 캠 테이블에서의 소스 모터의 전체 범위를 지정합니다. 소스 모터에 직접 사용되는 최소 위치는 **CamTable[m].X0** 으로 지정됩니다. 소스 모터에 직접 사용되는 최대 위치(**CamTable[m].X0 + CamTable[m].Dx**)입니다. 소스 모터 위치가 이 범위 밖이면 자동으로 범위 안으로 "롤 오버"됩니다. 테이블 포인트 간의 간격은 (**CamTable[m].Dx/CamTable[m].Nx**)입니다. **Nx** 는 테이블 내의 구간 수입니다.

CamTable[m].Nx 캠 테이블의 구간 수

설명:	캠 테이블 구간 수
범위:	음이 아닌 정수
단위:	테이블 구간
기본값:	0

CamTable[m].Nx 는 캠 테이블 내의 구간 수를 지정합니다. 구간은 테이블 내에 인접한 2 포인트 사이의 간격입니다. 데이터 포인트 **CamTable[m].PosData[i]**, **DacData[i]** 그리고 **OutData[i]**는 캠 테이블의 각 타입에서 전체 **Nx + 1** 데이터 포인트가 포함되도록 0에서 **Nx**까지의 인덱스값 **i**에 의해 캠 테이블에 등록되어야 합니다.

CamTable[m].OutBits 캠 테이블의 범용 출력 비트 수

설명:	캠 테이블의 범용 출력 비트 수
범위:	0 .. 32
단위:	비트 수
기본값:	0

CamTable[m].OutBits 는 캠 테이블의 현재 구간에 대응한 **CamTable[m].OutData[i]** 요소의 범용 출력 워드가 실제로 지정된 레지스터에 쓰여진 비트 0(LSB)에서 시작하는 연속된 비트 수를 지정합니다. **OutData[i]**의 지정 비트 이상의 비트는 모두 마스크 되고 **CamTable[m].OutLeftShift** 의 비트 수로 좌측 시프트 되며 **CamTable[m].pOut** 또는 **CamTable[m].pOutMask** 로 지정되어 레지스터에 마스크 된 쓰기가 수행됩니다.

예를 들어 **OutBits** 가 4이고 **OutLeftShift** 가 12인 경우에 **OutData[i]**의 0-3은 32비트 출력 레지스터의 비트 12-15에 기록됩니다. 출력 레지스터의 다른 비트는 이 캠 테이블의 영향을 받지 않습니다.

CamTable[m].OutData[i] 캠 테이블의 범용 출력 명령

설명: 캠 테이블의 범용 출력 명령 등록

범위: \$0..\$FFFFFFFF

단위: 비트 필드

기본값: nan(not-a-number)

CamTable[m].OutData[i]는 캠 테이블의 지정된 구간에 대한 범용 출력 워드를 지정합니다. 소스 모터의 위치가 아래에 지정된 범위인 경우 해당 출력 워드가 사용됩니다.

$$(X0 + i * Nx * Dx) \leq Pos < (X0 + [i+1] * Nx * Dx)$$

CamTable[m].OutData[i]는 32 비트 워드입니다. 따라서 각각 32 출력 비트 또는 32 비트 이내의 단일한 값을 지정할 수 있습니다. 유효한 **OutData[i]** 요소의 하위 *N* 비트만 사용되며 *N* 은 **CamTable[m].OutBits** 로 지정됩니다. 다른 비트는 마스크 됩니다. 결과적으로 값은 **CamTable[m].OutLeftShift** 의 비트 수로 좌측 시프트 되고 지시된 비트는 다음에 **CamTable[m].pOut** 의 출력 레지스터에 기록됩니다.

CamTable[m].OutLeftShift 캠 테이블의 범용 출력 좌측 시프트 수

설명: 캠 테이블의 범용 출력 워드 좌측 시프트 수

범위: 0 .. 31

단위: 비트

기본값: 0

CamTable[m].OutLeftShift 는 캠 테이블의 현재 구간에서의 **CamTable[m].OutData[i]** 요소 범용 출력 워드가 지정된 출력 레지스터에 기록되기 전에 좌측 시프트할 비트 수를 지정합니다. 따라서 32 비트 버스의 상위 비트로 매핑된 하드웨어에 반영하기 위해 테이블의 **OutData[i]** 값을 각각 시프트할 필요 없이 해당 하드웨어를 이용할 수 있습니다.

예를 들어 ACC-14E, ACC-65E 그리고 ACC-68E 와 같은 UMAC 의 범용 디지털 입출력 보드 출력은 32 비트 버스의 비트 8-15 에 매핑됩니다. **OutLeftShift** 를 8 로 설정하면 각 **OutData[i]** 의 값은 비트 0-7 을 사용할 수 있습니다. 이는 대부분의 사용자가 손쉽게 사용할 수 있는 방법입니다. 교대로 범용 출력이 32 비트 버스의 비트 16-31 에 매핑된 16 비트 D/A 변환기에 쓰기를 활용할 경우 각 **OutData[i]** 테이블 등록의 하위 16 비트에 16 비트 DAC 값을 쓰도록 **OutLeftShift** 를 16 으로 설정할 수 있습니다.

CamTable[m].PosData[j] 캠 테이블의 위치 명령

설명:	캠 테이블의 위치 명령 등록
범위:	부동 소수점
단위:	사용자 정의 테이블 위치 단위
기본값:	0.0

CamTable[m].PosData[j]는 캠 테이블의 개별 위치 명령 등록값을 지정합니다. 테이블에서의 Net 위치 명령은 소스 모터의 현재 위치에 따라서 인접한 등록 사이를 보간하는 방법으로 계산됩니다. 다음으로 명령은 대상 모터의 **Motor[x].CompDesPos** 레지스터에 기록되기 전에 스케일 계수항 **CamTable[m].ActivePosSf** 로 곱해집니다. 스케일 계수가 기본값 1.0 인 경우에 **PosData[j]** 등록은 대상 모터의 위치 단위입니다.

CamTable[m].PosData[0]에는 소스 모터 위치가 **CamTable[m].X0** 일 때의 위치 명령을 지정합니다. **CamTable[m].PosData[1]**에는 소스 모터 위치가 $(X0 + Dx/Nx)$ 일 때의 위치 명령을 지정합니다. **CamTable[m].PosData[2]**에는 소스 모터 위치가 $(X0 + 2*Dx/Nx)$ 일 때의 위치 명령을 지정합니다. **CamTable[m].PosData[Nx-1]**에는 소스 모터 위치가 $(X0 + [Nx-1]*Dx/Nx)$ 일 때의 위치 명령을 지정합니다.

캠 테이블의 동작은 소스 모터 위치가 $(X0$ 에서 $X0 + Dx)$ 의 범위 밖인 경우에 항상 롤 오버합니다. 이 롤 오버 기능은 "재귀적" 또는 "비재귀적"으로 할 수 있습니다. 보통 직선 동작에 사용되는 "재귀적" 기능에서는 각 사이클 종점과 시작점이 동일해야 합니다. 이 경우 **CamTable[m].PosData[Nx]**는 **CamTable[m].PosData[0]**과 동일한 값이어야 합니다.

보통 인쇄나 절단 휠과 같이 회전 동작에 사용되는 "비재귀적" 기능에서는 각 사이클 종점은 일반적인 회전 장치의 1 사이클 전체 주기의 시작점과 달라야 합니다. 이 경우 **CamTable[m].PosData[Nx]**는 **CamTable[m].PosData[0]**과 다른 값이어야 합니다. 이 차이는 사이클-보통 회전 장치의 주기-크기가 됩니다.

CamTable[m].PosSf 캠 테이블의 위치 스케일 계수

설명:	캠 테이블 위치 스케일 계수
범위:	부동 소수점
단위:	대상 모터 단위/테이블 위치 단위
기본값:	1.0

CamTable[m].PosSf 는 캠 테이블에서 보간, 계산된 위치 명령이 대상 모터의 명령 위치 보정 레지스터(**Motor[x].CompDesPos**)에 기록되기 전에 위치 명령에 곱할 명령 스케일 계수를 지정합니다. 그것이 기본값 1.0 인 경우 테이블 위치 등록은 대상 모터의 위치 단위가 됩니다.

CamTable[m].pOut 캠 테이블의 범용 출력 레지스터 주소

설명: 캠 테이블의 범용 출력 레지스터 주소

범위: 유효한 Power PMAC 주소

단위: Power PMAC 주소

기본값: 0(개별 출력 무효)

CamTable[m].pOut 은 테이블 현재 구간의 범용 출력이 마스크와 시프트 뒤에 기록할 레지스터 주소를 지정합니다. 이 변수값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 확장자를 해당 요소로 사용합니다. 사용자는 이 주소의 수적인 값을 알 필요가 없습니다. 단, 주소를 수적으로 지정할 수 있으며 레지스터가 정의된 데이터 구조체 요소(구 ACC- 11E 입출력 보드)가 아닌 경우처럼 이용할 수 있습니다.

예를 들어

CamTable[0].pOut = Acc68E[0].DataReg[4].a // Gatelo[0]

CamTable[1].pOut = PowerBrick[1].GpioData[0].a // Gate3[1]

CamTable[2].pOut = Acc24E3[2].Chan[3].Dac[1].a // Gate3[2]

CamTable[3].pOut = Sys.piom + \$A0000C // ACC-11E 용

CamTable[4].pOut = Acc5E[0].Macro[2][1].a // Gate2[0]

CamTable[5].pOut = Acc5EP3[1].MacroOutA[6][0].a // Gate3[1]

일부 어플리케이션은 캠 테이블의 직접 출력 기능을 소프트웨어 상태 기계를 구동하는 것과 같은 물리적인 출력 설정 이외의 목적으로 사용할 수 있습니다. 대부분의 경우 사용자 버퍼 중인 메모리 레지스터는 보통 "출력 레지스터"로 사용됩니다. 이 경우 유사한 설정이 됩니다.

CamTable[6].pOut = Sys.Udata[475].a

CamTable[6].pOut 으로 지정한 레지스터에 기록한 값을 읽을 수 없는 특성이 있는 경우(즉, "쓰기 전용" 레지스터) 캠 테이블에서 레지스터의 특정 비트를 수정할 수 없습니다. 이는 **Acc5E[j].Macro[j][k]** 레지스터와 같은 PMAC2 스타일 MACRO 출력 레지스터에 해당합니다. 또한 **PowerBrick[j].GpioData[j]** 레지스터에서의 PMAC3 스타일 출력 레지스터에 해당합니다.

이 경우 캠 테이블은 **CamTable[6].pOutBuf** 를 특정 비트를 변경할 수 있는 버퍼 레지스터의 주소로 설정하여 해당 버퍼 레지스터에 기록하도록 설정되어야 합니다.

이렇게 설정하면 버퍼 레지스터의 지정된 비트는 개별적으로 변경할 수 있습니다. 다음으로 버퍼 레지스터의 전체 32비트값은 **pOut** 으로 지정된 실제 출력 레지스터에 각 서보 사이클로 복사됩니다. 이 테이블 동작이 서보 사이클마다 출력 레지스터를 덮어쓰므로 다른 캠 테이블이나 사용자 프로그램의 태스크가 꼭 직접 출력 레지스터에 쓸 필요는 없습니다.

CamTable[m].pOutBuf 캠 테이블의 범용 출력 유지 레지스터 주소

설명: 캠 테이블의 범용 출력 유지 레지스터 주소

범위: 유효한 Power PMAC 주소

단위: Power PMAC 주소

기본값: 0(유지 레지스터 없음)

CamTable[m].pOutBuf 는 테이블의 범용 출력 유지 레지스터의 주소를 지정합니다. 여러 테이블이 동일한 물리적인 출력 레지스터로 출력 설정된 경우에 사용됩니다. 유지 레지스터를 사용하지 않으면 높은 번호가 부여된 테이블은 레지스터 내의 다른 출력 비트를 지정해도 낮은 번호의 테이블로 설정한 값을 덮어씁니다.

CamTable[m].pOutBuf 가 0 인 경우 캠 테이블은 유지 레지스터를 범용 출력에 사용하지 않습니다. 0 이외의 값인 경우 캠 테이블은 **CamTable[m].OutMask** 와 **CamTable[m].OutLeftShift** 를 사용해서 지정 레지스터에 읽기/변경/쓰기 동작을 수행하여 유지 레지스터의 비트를 세트하고 클리어합니다. 해당 동작은(임의의 메모리 레지스터와 같은) 읽기/쓰기 레지스터가 지정될 것을 조건으로 레지스터의 다른 비트를 변경하지 않습니다. 이는 2 개의 캠 테이블이 동일한 비트를 제어하지 않는 한, 여러 테이블이 경합 없이 동일한 레지스터의 비트를 제어할 수 있음을 의미합니다.

이 경우 캠 테이블은 **CamTable[m].pOut** 의 출력 레지스터에 직접 쓰지 않습니다. 또한 모든 테이블이 서보 사이클로 갱신된 다음 유지 레지스터의 값이 출력 레지스터에 복사됩니다.

Power Brick 의 I/O 와 같은 DSPGATE3 IC 범용 입출력 레지스터는 레지스터에 쓴 비트값을 반드시 읽을 수 있는 것은 아닙니다. 따라서 **CamTable[m].pOut** 가 **Gate3[j].GpioData[j].a** 로 설정되는 경우 유지 버퍼 레지스터를 사용해야 합니다. **Gate3[j]** 데이터 구조는 해당 각 I/O 레지스터용인 특별한 유지 레지스터를 제공합니다. 따라서 이 경우 **CamTable[m].pOutBuf** 는 유지 레지스터의 주소 **Gate3[j].GpioOutData[j].a** 로 설정됩니다.

ACC-5E 에서 사용되는 DSPGATE2 IC 의 MACRO 출력 레지스터는 레지스터에 쓴 후의 비트값을 읽을 수 없습니다. 따라서 **CamTable[m].pOut** 이 **Gate2[j].Macro[j][k].a** 로 설정되는 경우 유지 버퍼 레지스터가 사용됩니다. 이 경우 사용자 버퍼 내의 하나를 사용하는 메모리 레지스터를 사용합니다. 따라서 **CamTable[m].pOutBuf** 는 보통 **Sys.Udata[1297].a** 로 설정됩니다.

CamTable[m].SlewPosOffset 캠 테이블의 위치 오프셋 변화율

설명: 캠 테이블 위치 오프셋 변화율

범위: 부동 소수점

단위: 대상 모터 단위/서보 사이클

기본값: 0.0

CamTable[m].SlewPosOffset 은 캠 테이블이 유효할 때 대상 모터 위치를 소스 모터와 동기하기 위해 사용되는 실제 위치 오프셋(**Status** 요소 **CamTable[m].ActivePosOffset** 으로 표시)의 변화율을 지정합니다. 이는 효율적으로 동기를 확립하기 위해 사용되는 대상 모터 속도입니다. 테이블이 유효할 때 **CamTable[m].PosOffset** 을 변경하는 경우 이 요소는 실제 위치 오프셋이 이 새로운 값에 도달할 때의 변화율을 지정합니다.

CamTable[m].SlewX0 캠 테이블의 소스 시작 위치 변화율

설명: 캠 테이블 소스 시작 위치 변화율

범위: 부동 소수점

단위: 소스 모터 단위/서보 사이클

기본값: 0.0(슬루 제어 무효)

CamTable[m].SlewX0 은 명령 기준 위치 **CamTable[m].X0** 이 변경될 때 테이블 계산(**CamTable[m].ActiveX0**)에 각 서보 사이클에서 사용되는 소스 모터의 기준 위치 변화율을 지정합니다. 그것은 소스 모터 단위/서보 사이클로 표시됩니다. **SlewX0** 이 기본값 0.0 인 경우 **ActiveX0** 의 값은 **X0** 의 새로운 값으로 즉시 전환됩니다.

이 요소는 캠 테이블이 유효할 때도 과잉 반응 없이 "단계적으로 추종"할 수 있도록 소스 모터 기준 위치가 제어되어 변화할 수 있게 해 줍니다.

CamTable[m].Source 캠 테이블의 소스 모터 번호

설명: 캠 테이블 소스 모터 번호

범위: 0 .. 255

단위: 모터 번호

기본값: 0

CamTable[m].Source 는 캠 테이블 내의 "소스" 위치로 명령 위치에 사용되는 Power PMAC 모터 번호를 지정합니다. "대상" 모터의 캠 동작 및 임의의 테이블에서 구동되는 출력 상태는 테이블이 유효한 경우 해당 소스 모터 위치의 함수가 됩니다.

마스터로 사용되는 위치 센서가 부착된 물리적 모터가 없거나, Power PMAC 제어를 받는 물리적 모터가 없는 경우에도 유효한 Power PMAC "모터"의 위치로 센서가 사용되어야 한다는 점에 주의하십시오. 이러한 경우에는 Power PMAC 모터는 무효(kill 된) 상태가 유지되어야 하며 그 경우에는 Power PMAC 은 명령 위치 레지스터에 실제 위치를 서보 사이클마다 복사합니다.

어떤 경우에는 물리적인 액추에이터나 센서를 갖지 않은 가상 Power PMAC 모터는 기계적인 캠 시스템의 라인 샤프트를 수학적으로 대신 사용하기 위해 하나 이상의 소스 모터를 캠 테이블에 사용할 수 있습니다.

CamTable[m].Target 캠 테이블의 대상 모터 번호

설명: 캠 테이블 대상 모터 번호

범위: 0 .. 255

단위: 모터 번호

기본값: 0

CamTable[m].Target 은 캠 테이블에서 제어되는 모터 번호를 지정합니다. 이 모터의 명령 위치 오프셋 레지스터 **Motor[x].CompDesPos** 는 캠 테이블의 위치 데이터에 의해 설정됩니다. 또한 해당 토크 오프셋 레지스터 **Motor[x].CompDac** 은 토크 데이터에 의해 설정됩니다.

CamTable[m].X0 캠 테이블의 소스 시작 위치

설명: 캠 테이블 소스 시작 위치

범위: 부동 소수점

단위: 소스 모터 단위

기본값: 0.0

CamTable[m].X0 은 캠 테이블에서의 소스 모터 시작(최소) 명령 위치를 지정합니다. 데이터 포인트 **CamTable[m].PosData[0]**, **DacData[0]** 그리고 **OutData[0]**에서는 이 소스 모터 위치에서의 캠 테이블의 값을 나타냅니다. 캠 테이블에는 **CamTable[m].X0** 부터 (**CamTable[m].X0 + CamTable[m].Dx**)까지의 위치 범위가 직접 지정됩니다. 캠 테이블은 이 선언 범위를 초과한 위치에 대해 자동으로 롤 오버합니다.

이 위치는 소스 모터의 제로 위치를 기준으로 모터 단위로 나타냅니다. **CamTable[m].X0** 의 값은 언제나 변경할 수 있습니다. 변경되는 경우 각 서보 사이클에서 캠 테이블 계산에 사용되는 기준 위치의 Status 요소 **CamTable[m].ActiveX0** 은 저장 설정 요소 **CamTable[m].SlewX0** 의 속도로 새로운 명령값 **X0** 에 가까워집니다.

CompTable[m].저장 데이터 구조체 요소(V1.6 2014.03)

CompTable[m].데이터 구조체는 Power PMAC 보정 테이블을 탑재하여 모터의 측정 위치 또는 서보 출력 명령에 자동으로 보정을 적용할 수 있습니다. Power PMAC 에는 256 개의 보정 테이블이 있습니다. 따라서 테이블 인덱스값 m 은 0~255 의 범위입니다. 글로벌 저장 설정 변수 **Sys.CompEnable** 은 몇 개의 보정 테이블이 유효인지 지정합니다. 0 부터 **Sys.CompEnable** - 1 까지 인덱스 m 의 보정 테이블이 유효합니다.

CompTable[m].Ctrl 보정 테이블의 모드 제어

설명: 보정 테이블의 보정 모드 제어

범위: \$00..\$FF(0 - 255)

단위: 비트 폭

기본값: \$03

CompTable[m].Ctrl 은 보정 테이블 효과가 있는 모드의 몇 가지 기능을 지정하고 4 세트의 2 가지 제어 비트로 구성된 8 비트의 값입니다.

비트 0 과 1 은 각 소스 차원에서 보간의 차수를 결정합니다. 이것이 나타내는 값은 다음과 같습니다.

= 0(00): 모든 **Source** 에서 1 차 보간

= 1(01): **Source[0]**에서 3 차 보간, 다른 **Source** 에서 1 차 보간

= 2(10): **Source[0]**과 **Source[1]**에서 3 차 보간, **Source[2]**에서 1 차 보간

= 3(11) 모든 **Source** 에서 3 차 보간

소스 차원의 보정은 1 차 보간에서는 현 위치 양쪽 포인트 간의 선형 근사로 계산됩니다. 테이블 내의 포인트를 통과하기 때문에 보정 데이터가 급격하게 변화하여 현저하게 거친 동작이 발생할 수도 있습니다.

소스 차원의 보정은 3 차 보간에서는 현 위치 양쪽 2 점을 사용하고 3 차 근사로 계산됩니다. 보정의 기울기는 테이블 포인트를 통과해도 항상 연속적입니다. 이 보간은 1 차 보간의 약 2 배의 연산시간이 필요합니다.

사용하지 않는 소스($Nx[n]=0$)와 지정된 보간의 차원은 중요하지 않습니다. 그러나 "0 차원" 테이블(모든 $Nx[n]$ 값 = 0)에서는 이 2 비트의 제어 비트의 값이 0 보다 큰 경우 Power PMAC 은 **Source[0]**으로 지정된 모터에서 서보 출력 명령값을 얻어서 테이블(**Data[0]**)의 하나의 데이터 포인트에 배치합니다. 그때부터 서보 루프의 캐스케이드 연결을 가능하게 하여 위치 보정값으로 사용할 수 있습니다.

비트 2 와 3 은 **Source[0]** 차원에서 테이블 끝에서의 "경계 모드"를 결정합니다.

비트 4 와 5 는 **Source[1]** 차원에서의 테이블 끝에서의 "경계 모드"를 결정합니다.

비트 6 과 7 은 **Source[2]** 차원에서의 테이블 끝에서의 "경계 모드"를 결정합니다.

각 차원에서 2 비트가 나타내는 값은 다음과 같습니다.

- =0(00): 해당 차원의 테이블 경계에서의 롤 오버
- =1(01): 해당 차원의 테이블 경계를 초과하면 마지막 보정값을 유지
- =2(02): 해당 차원의 테이블 경계에서 보정 테이블을 미러링
- =3(03): (추후 사용하기 위해 예약)

CompTable[m].Data[j] 보정 테이블의 1 차원 등록

설명:	보정 테이블 1 차원 등록
범위:	부동 소수점
단위:	사용자 정의 테이블 단위
기본값:	0.0

CompTable[m].Data[j]는 1 차원 테이블의 개별 항목(보정)값을 지정합니다. 보정은 인접한 항목 간의 보간으로 산출됩니다. 다음으로 대상 레지스터에 기록하기 전에 해당 보정에 스케일 계수를 곱합니다. 보통 스케일 계수는 1.0 입니다. 따라서 해당 항목은 대상 레지스터(위치 또는 토크) 단위로 지정됩니다.

CompTable[m].Data[0]에서는 **Source[0]**으로 지정된 모터가 **X0[0]**으로 지정된 위치일 때의 보정값을 지정합니다. **CompTable[m].Data[1]**은 모터가 $(X0[0]+Dx[0]/Nx[0])$ 인 위치에서의 보정값을 지정합니다. **CompTable[m].Data[2]**는 모터가 $(X0[0]+2*Dx[0]/Nx[0])$ 인 위치에서의 보정값을 지정합니다. 이하 동일합니다.

테이블이 해당 차원에서 롤 오버로 지정되지 않는 경우 최종 테이블 등록 **CompTable[m].Data[Nx[0]]**는 모터가 $(X0[0]+Dx[0])$ 인 위치에서의 보정값을 지정합니다. 테이블이 해당 차원에서 롤 오버로 지정되는 경우 **CompTable[m].Data[Nx[0]-1]**에 의해서만 포인트 **CompTable[m].Data[0]**이 사용됩니다.

1 차원 데이터 포인트에서는 포인트 인덱스값 i 는 임의의 정수 상수값 0 에서 16,777,215(**Nx[n]**으로 선언된 범위 내) 또는 임의의 로컬 L-변수로 설정할 수 있습니다. 소수 상수 또는 수식은 허가되지 않습니다.

CompTable[m].Data[j][i] 보정 테이블의 2 차원 등록

설명:	보정 테이블 2 차원 등록
범위:	부동 소수점
단위:	사용자 정의 테이블 단위
기본값:	0.0

CompTable[m].Data[j][i]는 2 차원 테이블의 개별 항목(보정)값을 지정합니다. 보정은 양쪽 차원에서 인접한 항목 간의 보간으로 산출됩니다. 다음으로 대상 레지스터에 쓰여지기 전에 해당 보정에 스케일 계수를 곱합니다. 보통 스케일 계수는 1.0 입니다. 따라서 해당 항목은 대상 레지스터(위치 또는 토크) 단위로 지정됩니다.

CompTable[m].Data[0][0]에서는 **Source[0]**으로 지정된 모터가 **X0[0]**으로 지정된 위치에서 보정값을 지정하고 **Source[1]**로 지정된 모터가 **X0[1]**로 지정된 위치에서 보정값을 지정합니다.

CompTable[m].Data[0][1]에서는 **Source[0]**으로 지정된 모터가 $(X0[0] + Dx[0]/Nx[0])$ 의 위치에서 보정값을 지정하고 **Source[1]**로 지정된 모터가 **X0[1]**로 지정된 위치에서 보정값을 지정합니다.

CompTable[m].Data[1][0]에서는 **Source[0]**으로 지정된 모터가 **X0[0]**의 위치에서 보정값을 지정하고 **Source[1]**로 지정된 모터가 $(X0[1] + Dx[1]/Nx[1])$ 로 지정된 위치에서 보정값을 지정합니다.

테이블이 **Source[0]**으로 지정된 모터의 차원에서 롤 오버로 지정되지 않은 경우 항목 테이블 경계인 **CompTable[m].Data[0][Nx[0]]**에서는 **Source[0]**으로 지정된 모터가 $(X0[0] + Dx[0])$ 의 위치에서 보정값을 지정하고 **Source[1]**로 지정된 모터가 **X0[1]**로 지정된 위치에서의 보정값을 지정합니다. 테이블이 해당 차원에서 롤 오버로 지정되는 경우 **CompTable[m].Data[0][Nx[0]-1]**에 의해서만 포인트 **CompTable[m].Data[0][0]**이 사용됩니다.

테이블이 **Source[1]**로 지정된 모터의 차원에서 롤 오버로 지정되지 않은 경우 항목 테이블 경계의 **CompTable[m].Data[Nx[1]][0]**에서는 **Source[1]**로 지정된 모터가 $(X0[1] + Dx[1])$ 의 위치에서 보정값을 지정하고 **Source[0]**으로 지정된 모터가 **X0[0]**로 지정된 위치에서 보정값을 지정합니다. 테이블이 해당 차원에서 롤 오버로 지정되는 경우 **CompTable[m].Data[Nx[1]-1][0]**에 의해서만 포인트 **CompTable[m].Data[0][0]**이 사용됩니다.

2 차원 데이터 포인트를 위해 포인트 인덱스값 i 와 j 는 임의의 정수 상수값 0 에서 65,533($Nx[n]$ 으로 선언된 범위 내) 또는 로컬 L-변수 i 를 위한 L0 과 j 를 위한 L0 또는 L1 로 설정할 수 있습니다. 소수 상수 또는 수식은 허가되지 않습니다.

CompTable[m].Data[k][j][i] 보정 테이블의 3 차원 등록

설명:	보정 테이블 3 차원 등록
범위:	부동 소수점
단위:	사용자 정의 테이블 단위
기본값:	0.0

CompTable[m].Data[k][j][i]는 3 차원 테이블의 개별 항목(보정)값을 지정합니다. 보정은 모든 3 차원 내의 인접한 항목 간의 보간으로 산출됩니다. 다음으로 대상 레지스터에 쓰여지기 전에 해당 보정에 스케일 계수를 곱합니다. 보통 스케일 계수는 1.0 입니다. 따라서 해당 항목은 대상 레지스터(위치 또는 토크) 단위로 지정됩니다.

CompTable[m].Data[0][0][0]에서는 **Source[0]**으로 지정된 모터가 **X0[0]**으로 지정된 위치에서 **Source[1]**로 지정된 모터가 **X0[1]**로 지정된 위치에서 그리고 **Source[2]**로 지정된 모터가 **X0[2]**로 지정된 위치에서 보정값을 지정합니다.

CompTable[m].Data[0][0][1]에서는 **Source[0]**으로 지정된 모터가 $(X0[0]+Dx[0]/Nx[0])$ 의 위치에서 **Source[1]**로 지정된 모터가 **X0[1]**로 지정된 위치에서 그리고 **Source[2]**로 지정된 모터가 **X0[2]**로 지정된 위치에서 보정값을 지정합니다.

CompTable[m].Data[0][1][0]에서는 **Source[0]**으로 지정된 모터가 **X0[0]**의 위치에서 **Source[1]**로 지정된 모터가 $(X0[1]+Dx[1]/Nx[1])$ 로 지정된 위치에서 그리고 **Source[2]**로 지정된 모터가 **X0[2]**로 지정된 위치에서 보정값을 지정합니다.

CompTable[m].Data[1][0][0]에서는 **Source[0]**으로 지정된 모터가 위치 **X0[0]**에서 **Source[1]**로 지정된 모터가 **X0[1]**로 지정된 위치에서 그리고 **Source[2]**로 지정된 모터가 $(X0[2]+Dx[2]/Nx[2])$ 로 지정된 위치에서 보정값을 지정합니다.

테이블이 **Source[0]**으로 지정된 모터 차원에서 롤 오버로 지정되지 않은 경우 항목 테이블 경계의 **CompTable[m].Data[0][0][Nx[0]]**에서는 **Source[0]**으로 지정된 모터가 $(X0[0]+Dx[0])$ 의 위치에서 **Source[1]**로 지정된 모터가 **X0[1]**로 지정된 위치에서 그리고 **Source[2]**로 지정된 모터는 **X0[2]**로 지정된 위치에서 보정값을 지정합니다. 테이블이 해당 차원에서 롤 오버로 지정되는 경우 **CompTable[m].Data[0][0][Nx[0]-1]**에 의한 포인트 **CompTable[m].Data[0][0][0]**이 해당 차원 중에 사용됩니다.

테이블이 **Source[1]**로 지정된 모터의 차원에서 롤 오버로 지정되지 않은 경우 항목 테이블 경계의 **CompTable[m].Data[0][Nx[1]][0]**에서는 **Source[1]**로 지정된 모터가 $(X0[1]+Dx[1])$ 의 위치에서 **Source[0]**으로 지정된 모터가 **X0[0]**으로 지정된 위치에서 그리고 **Source[2]**로 지정된 모터가 **X0[2]**로 지정된 위치에서 보정값을 지정합니다. 테이블이 해당 차원에서 롤 오버로 지정되는 경우 **CompTable[m].Data[0][Nx[1]-1][0]**에 의한 포인트 **CompTable[m].Data[0][0][0]**이 해당 차원 중에 사용됩니다.

테이블이 **Source[2]**로 지정된 모터의 차원에서 롤 오버로 지정되지 않은 경우 항목 테이블 경계의 **CompTable[m].Data[Nx[1]][0][0]**에서는 **Source[2]**로 지정된 모터가 (**X0[2]+Dx[2]**)의 위치에서 **Source[0]**으로 지정된 모터가 **X0[0]**으로 지정된 위치에서 그리고 **Source[2]**로 지정된 모터가 **X0[2]**로 지정된 위치에서 보정값을 지정합니다. 테이블이 해당 차원에서 롤 오버로 지정되는 경우 **CompTable[m].Data[Nx[1]-1][0][0]**에 의한 포인트 **CompTable[m].Data[0][0][0]**이 해당 차원 중에 사용됩니다.

3 차원 데이터 포인트를 위해서 포인트 인덱스값 i 는 0 부터 65,533 까지 임의의 정수 상수값을 설정할 수 있고 포인트 인덱스값 j 와 k 는 0 부터 253 까지 임의의 정수 상수값(**Nx[n]**으로 선언된 범위 내) 또는 i 를 위한 로컬 L -변수 $L0$ 또는 j 를 위한 $L0$ 또는 $L1$ 그리고 k 를 위한 $L0$, $L1$ 또는 $L2$ 로 설정할 수 있습니다. 소수 상수 또는 수식은 허가되지 않습니다.

CompTable[m].Dx[n] 보정 테이블의 소스 구간 거리

설명: 보정 테이블의 차원 소스 위치 스패 거리

범위: 음이 아닌 부동 소수점

단위: 소스 모터 단위

기본값: 0.0

CompTable[m].Dx[n]은 **X0[n]**의 인덱스로 표시된 차원의 테이블에서 소스 모터의 스패 거리를 지정합니다. 이 인덱스는 테이블의 제 1 차원, 제 2 차원과 제 3 차원을 각각 지정하는 0, 1 또는 2 의 값을 설정할 수 있습니다.

2 차원 테이블의 **Dx[0]**는 데이터 포인트 **Data[j][i]**의 제 2 인덱스 $[i]$ 에 따라 변화하는 크기의 스패 거리를 지정합니다. 그리고 **Dx[1]**는 데이터 포인트의 제 1 인덱스 $[j]$ 에 따라 변화하는 크기의 스패 거리를 지정합니다. **Dx[2]**는 이 경우 사용되지 않습니다.

3 차원 테이블의 **Dx[0]**는 데이터 포인트 **Data[k][j][i]**의 3 번째 인덱스 $[i]$ 에 따라 변화하는 크기의 스패 거리를 지정합니다. **Dx[1]**는 데이터 포인트의 2 번째 인덱스 $[j]$ 에 따라 변화하는 크기의 스패 거리를 지정합니다. 그리고 **Dx[2]**는 데이터 포인트 **Data[k][j][i]**의 제 1 인덱스 $[k]$ 에 따라 변화하는 크기의 스패 거리를 지정합니다.

이 스패 거리는 소스 모터의 단위입니다. 차원에서의 소스 모터에서 사용되는 최소 위치는 **X0[n]**으로 지정됩니다. 사용되는 최대 위치는 **X0[n]+ Dx[n]**입니다. 차원에서의 테이블 포인트 간의 간격은 **Dx[n]/ Nx[n]**입니다.

CompTable[m].Nx[n] 보정 테이블의 구간 수

설명: 보정 테이블의 차원 구간 수

범위: 음이 아닌 정수

단위: 테이블의 포인트 구간

기본값: 0

CompTable[m].Nx[n]는 **Nx[n]**의 인덱스로 표시된 테이블 차원에서의 구간 수를 지정합니다. 해당 인덱스는 테이블의 제 1 차원, 제 2 차원과 제 3 차원을 각각 지정하여 0, 1 또는 2의 값으로 설정할 수 있습니다. 요소의 값이 0보다 큰 값으로 구간 수를 지정하는 경우 해당 차원은 유효가 됩니다. 그러나 0의 경우 차원은 유효가 아닙니다. (구간은 2개 차원의 인접한 테이블의 포인트 간의 공간입니다.) 예를 들어 1 차원의 테이블에서는 **CompTable[m].Nx[0]**은 0보다 크고 **CompTable[m].Nx[1]**과 **CompTable[m].Nx[2]**는 0이 됩니다.

2 차원 테이블의 **Nx[0]**는 데이터 포인트 **Data[j][i]**의 제 2 인덱스 **[i]**에 따라 변화하는 크기의 구간 수를 지정합니다. 그리고 **Nx[1]**는 데이터 포인트의 제 1 인덱스 **[j]**에 따라 변화하는 크기의 구간 수를 지정합니다.

3 차원 테이블의 **Nx[0]**는 데이터 포인트 **Data[k][j][i]**의 제 3 인덱스의 **[i]**에 따라 변화하는 크기의 구간 수를 지정합니다. **Nx[1]**은 데이터 포인트의 제 2 인덱스 **[j]**에 따라 변화하는 크기의 구간 수를 지정합니다. 그리고 **Nx[2]**는 데이터 포인트의 제 1 인덱스 **[k]**에 따라 변화하는 크기의 구간 수를 지정합니다.

CompTable[m].Ctrl 에서 특정 차원의 테이블을 롤 오버하지 않는다고 지정한 경우 데이터 포인트는 0 부터 **Nx[n](Data[0]~Data[Nx])**로 지정된 구간 수까지의 데이터 포인트를 설정합니다. 따라서 해당 차원의 데이터 포인트 수는 구간 수보다 하나씩 많아집니다.

CompTable[m].Ctrl 에서 특정 차원의 테이블을 롤 오버한다고 지정한 경우 데이터 포인트는 0 부터 해당 차원으로 지정된 구간 수-1까지의 인덱스로 입력되고 해당 차원의 데이터 포인트 수는 구간 수와 같아집니다. 해당 차원의 추가 포인트(구간 수와 같은 인덱스를 가진)는 오차 없이 입력될 수도 있습니다. 그러나 해당 차원에서 앞의 포인트와 0 인 인덱스 포인트 사이에서 보간하여 인접한 구간에서 보정값이 계산되므로 추가 포인트는 사용되지 않습니다.

CompTable[m].OutCtrl 보정 테이블의 보정 출력 제어

설명: 보정 테이블의 보정 출력 제어

범위: 0 ..255

단위: 비트 필드

기본값: 0

CompTable[m].OutCtrl 은 보정 테이블에서 해당 대상 레지스터의 처리를 지정합니다. 그것은 **CompTable[m].Target[q]**로 지정된 레지스터가 어떻게 처리되는지를 제어하는 비트 q 를 가진(2^q 의 값을 가진) 8 비트의 값입니다. 비트가 기본값 0 인 경우 보정값은 해당 레지스터에서 검출된 값에 덮어씁니다. 비트가 1 로 설정될 경우 보정값은 해당 레지스터에서 검출된 값에 더해집니다. 그리고 결과적으로 발생하는 합계가 레지스터에 기록됩니다.

덧셈 옵션은 여러 개의 보정값을 하나의 대상 레지스터에 적용할 수 있게 합니다. 이 기능의 일반적인 사용 방법 중의 하나는 모터 하나에 대해 "거친" 테이블과 "정밀한" 테이블을 적용하는 것입니다.

서보 사이클에 부여된 레지스터에 기록할 최초의 (가장 낮은 번호가 부여된) 테이블은 이전 사이클의 보정값에 덮어쓰기 0 에 **OutCtrl** 비트의 값이 설정되어야 한다는 점에 주의하십시오. 서보 사이클에서 동일한 레지스터에 기록할 후속 (높은 번호가 부여된)테이블도 **OutCtrl** 비트값을 1 로 설정하고 현재의 보정값에 해당 보정값을 더해야 합니다.



경고

보정값을 대상 레지스터에 더하도록 설정된 테이블은 (이전 서보 사이클에서) 현재값에 덮어쓰기 할 다른 테이블이 각 사이클에서 반드시 선행되어야 합니다. 그렇지 않으면 더하는 테이블은 수적인 적분기로 작용하여 동일한 방향으로 대상 레지스터의 값을 끊임없이 더하므로 위험한 폭주 상태를 유발할 가능성이 있습니다.

CompTable[m].Sf[q] 보정 테이블의 목표값 스케일 계수

설명:	보정 테이블 보정 목표 스케일 계수
범위:	부동 소수점
단위:	목표 레지스터 단위/테이블 보정 단위
기본값:	1.0

CompTable[m].Sf[q]는 대상 레지스터에 쓰여지기 전에 테이블에서 산출된 보정값을 곱할 스케일 계수를 지정합니다. 테이블에 여러 개의 대상 레지스터가 존재할 수 있으므로 대상 레지스터는 각각 그 자체의 스케일 계수를 가집니다. 주어진 인덱스(0~7)의 스케일 계수는 동일한 인덱스의 대상 레지스터에 작용합니다(예를 들어 **Sf[2]**는 **Target[2]**의 레지스터에 작용합니다).

테이블 항목이 대상 레지스터의 단위인 것과 같이 대부분의 어플리케이션은 **CompTable[m].Sf[n]** 요소를 1.0 인 기본값 그대로 둡니다. 그러면 대상 레지스터가 위치 레지스터의 테이블에도 적합해지므로 테이블 단위는 대상 모터의 위치 단위와 같아지고 대상 레지스터가 토크 레지스터의 테이블에도 적합해져서 테이블 단위는 서보 명령 단위와 동일한 16 비트 부호가 있는 출력이 됩니다.

CompTable[m].Source[n] 보정 테이블의 소스 모터 수

설명:	보정 테이블 차원에서의 소스 모터 수
범위:	0 ..255
단위:	모터 수
기본값:	0

CompTable[m].Source[n]은 **Source[n]**의 인덱스로 표시된 테이블 차원에서 "소스" 위치로 사용되는 모터 번호를 지정합니다. **Source[n]**의 인덱스는 0, 1 또는 2 의 값을 취할 수 있습니다. 1 차원 테이블에서는 **Source[0]**만 지정해야 합니다. 2D 테이블에서는 **Source[1]**이 지정되어야 합니다. 3D 테이블에서는 **Source [2]**가 지정되어야 합니다. 동일한 최종 인덱스를 위한 설정 요소 **CompTable[m].Nx[n]**이 0 보다 큰 경우에만 주어진 소스를 테이블 중에서 사용합니다.

2 차원 테이블에서 **Source[0]**은 해당 보정값이 데이터 포인트의 2 번째 인덱스에 따라 변화하는 소스 모터를 지정합니다. 그리고 **Source[1]**은 해당 보정값이 데이터 포인트의 제 1 인덱스에 따라 변화하는 소스 모터를 지정합니다. **Source[2]**는 이 경우에는 사용되지 않습니다.

3 차원 테이블에서 **Source[0]**은 해당 보정값이 데이터 포인트의 3 번째 인덱스에 따라 변화하는 소스 모터를 지정합니다. **Source[1]**은 해당 보정값이 데이터 포인트의 2 번째 인덱스에 따라 변화하는 소스 모터를 지정합니다. 그리고 **Source[2]**는 해당 보정값이 데이터 포인트의 제 1 인덱스에 따라 변화하는 소스 모터를 지정합니다.

CompTable[m].SourceCtrl 보정 테이블의 소스 레지스터 제어

설명: 보정 테이블 소스 레지스터 제어

범위: 0 ..7

단위: 비트 필드

기본값: 0

CompTable[m].SourceCtrl 은 소스 모터의 어느 위치를 각 차원의 보정 테이블에서 사용할지 지정합니다. 그것은 3비트값이며 비트 n 은 **CompTable[m].Source[n]**에 의해 모터의 어느 위치를 사용할지 결정합니다. 비트가 기본값 0 인 경우 순수 명령 위치(**Motor[x].DesPos**)가 사용됩니다. 비트가 1 로 설정되는 경우 보정되지 않은 실제 위치(**Motor[x].Pos**)가 사용됩니다.

명령 위치를 사용하면 서보 루프 운동 특성과 함께 보정의 상호 작용을 가능한 한 제거되고 보정에서 발생하는 노이즈의 레벨이 낮아집니다. 보통 이는 높은 서보 루프 게인의 사용을 가능하게 합니다. 실제 위치를 사용하면 더 신속하고 정확한 보정이 제공되고 서보 상호 작용 및 측정 노이즈가 낮으면 더 높은 정밀도를 제공할 수 있습니다.

CompTable[m].Nx[n]이 0 보다 커서 해당 차원의 보정 테이블을 사용하는 경우 **CompTable[m].SourceCtrl** 의 비트 n 이 사용됩니다. 따라서 가장 일반적인 1 차원 보정 테이블은 0 보다 크게 설정된 **Nx[0]**밖에 없으므로 **SourceCtrl** 비트 0 만 사용됩니다.

CompTable[m].SourceCtrl 은 V1.6 펌웨어의 새로운 기능입니다(2014 년 1/4 분기 출시). 기본값 0 에서는 소스 모터의 명령 위치를 사용하면 이전 버전의 펌웨어와 동작이 호환됩니다.

CompTable[m].Target[q] 보정 테이블의 대상 주소

설명: 보정 테이블의 대상 주소

범위: 선택된 Power PMAC 주소

단위: Power PMAC 주소

기본값: 0(대상 무효)

CompTable[m].Target[q]는 테이블에서 산출된 보정값이 기록될 레지스터의 주소를 지정합니다("대상"). 하나의 테이블에 여러 개의 대상 레지스터를 가질 수 있습니다. **Target[q]**의 인덱스는 0 부터 7 의 값을 설정할 수 있습니다. 따라서 테이블에는 대상이 8 개까지 있는 경우가 있습니다. **CompTable[m].Target[q]**가 0 으로 설정되는 경우 이 대상은 유효가 아닙니다.

이 변수값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 확장자를 해당 요소로 사용합니다. 사용자는 이 주소의 수적인 값을 알 필요가 없습니다.

기계적 또는 센서 오차를 보정하는 위치 보정 테이블에서는 **CompTable[m].Target[q]**는 대상 모터의 외측 루프 보정 레지스터(**CompPos**)의 주소로 설정됩니다. 그리고 대부분 내측 루프 보정 레지스터(**CompPos2**)의 주소로 설정되고 측정 오차에 의한 속도 루프의 리플을 감소시키는데 도움을 줄 수 있습니다. 해당 레지스터의 값은 Net 보정된 실제 위치 **Motor[x].ActPos** 그리고 **Motor[x].ActPos2** 를 각각 얻기 위해 **Motor[x].Pos** 와 **Motor[x].Pos2** 의 원시 측정값에 각각 더해집니다.

예는 다음과 같습니다.

CompTable[0].Target[0] = Motor[1].CompPos.a
CompTable[0].Target[1] = Motor[1].CompPos2.a

전자 캠용으로 모션을 생성하는 위치 보정 테이블의 경우 **CompTable[m].Target[q]**는 대상 모터의 명령 위치 보정 레지스터(**CompDesPos**)의 주소로 설정됩니다. 그때 이는 중합할 수 있습니다. 이 레지스터값을 수학적으로 계산된 궤적값과 Net 명령 위치의 **Motor[x].DesPos** 를 얻기 위한 마스터 위치의 값에 더합니다.

예는 다음과 같습니다.

CompTable[0].Target[0]=Motor[4].CompDesPos.a

토크 보상 테이블에서는 **CompTable[m].Target[q]**는 대상 모터의 서보 출력 보정 레지스터 **Motor[x].CompDac** 의 주소로 설정됩니다.

예는 다음과 같습니다.

CompTable[1].Target[0]=Motor[1].CompDac.a

백래시 보정 테이블에서는 **CompTable[m].Target[q]**는 대상 모터의 백래시 보정 레지스터 **Motor[x].BICompSize** 의 주소로 설정됩니다.

예는 다음과 같습니다.

CompTable[2].Target[0]=Motor[1].BICompSize.a

CompTable[m].X0[n] 보정 테이블의 소스 시작 위치

설명: 보정 테이블 차원의 소스 시작 위치

범위: 부동 소수점

단위: 소스 모터 단위

기본값: 0.0

CompTable[m].X0[n]은 **X0[n]**의 인덱스에 표시된 테이블 차원에서 소스 모터의 시작(최소) 위치를 지정합니다. 해당 인덱스는 테이블의 제 1 차원, 제 2 차원과 제 3 차원을 각각 지정하여 0, 1 또는 2의 값을 취할 수 있습니다.

2 차원 테이블에서 **X0[0]**은 데이터 포인트의 제 2 인덱스에 따라 변화하는 크기의 최소 소스 위치를 지정합니다. 그리고 **X0[1]**은 데이터 포인트의 제 1 인덱스에 따라 변화하는 크기의 최소 소스 위치를 지정합니다. **X0[2]**는 이 경우 사용되지 않습니다.

3 차원 테이블에서 **X0[0]**은 데이터 포인트의 제 3 인덱스에 따라 변화하는 크기의 최소 소스 위치를 지정합니다. **X0[1]**은 데이터 포인트의 제 2 인덱스에 따라 변화하는 크기의 최소 소스 위치를 지정합니다. 그리고 **X0[2]**는 데이터 포인트의 제 1 인덱스에 따라 변화하는 크기의 최소 소스 위치를 지정합니다.

이 위치는 모터의 제로 위치를 기준으로 소스 모터 단위로 나타냅니다.

Coord[x].저장 데이터 구조체 요소(V1.6 2014.03)

Coord[x].데이터 구조체는 지정된 좌표계의 소프트웨어 정보를 모두 제공합니다. 이 장에서는 비휘발성 메모리에 저장되는 데이터 구조체의 요소를 설명합니다.

주의: 각 괄호 안의 좌표계 인덱스값은 Power PMAC 시스템에서 유효한 좌표계 번호의 범위 내의 정수 상수 혹은 프로그램 또는 통신계에 의한 "로컬" L-변수(L0-L1007 사용)에 의해 지정됩니다. 인덱스값을 지정하는 다른 방법을 활용할 수 없습니다. 인덱스값은 0 부터 **Sys.MaxCoords** - 1 까지 지정할 수 있으며 **Sys.MaxCoords** 의 최대값은 128 입니다.

Coord[x].AbortAllMode 좌표계의 전체 Abort 동작

설명: "전체 Abort" 입력의 좌표계 동작

범위: 0 ..3

단위: 열거 형식

기본값: 0

Coord[x].AbortAllMode 는 Power PMAC 이 글로벌 저장 설정 요소 **Sys.pAbortAll**, **Sys.AbortAllBit** 그리고 **Sys.AbortAllLimit** 로 지정되는 "전체 Abort" 입력을 수신할 때 좌표계가 취하는 동작을 지정합니다. "전체 Abort" 입력 기능은 2014 년 1/4 분기에 출시한 V1.6 펌웨어의 새로운 기능입니다.

Coord[x].AbortAllMode 가 0 의 기본값인 경우 마치 **abort** 명령이 좌표계에 전송된 것처럼 좌표계의 모든 모터는 클로즈드 루프 제어되어 정지합니다. 이는 IEC-61800-5-2 기계 안전 규격에 기초한 "카테고리 2" 안전 정지 제어를 제공합니다. 이 규격에 기초한 "카테고리 1" 안전 정지(제어 정지 뒤에 이어지는 소프트웨어가 개재하지 않는 Disable)가 요구되는 경우 해당 입력을 전환하는 동일한 동작이 주요 회로(보통 버스 전원 또는 게이트 드라이버 전원 중 하나)의 전원을 차단하는 인증 지연 릴레이를 기동해야 합니다.

Coord[x].AbortAllMode 가 1 로 설정되는 경우 마치 **disable** 명령이 좌표계에 전송된 것처럼 좌표계의 모든 모터는 즉시 Disable 됩니다(kill 됨). "지연이 있는 Disable"이 설정되어도 브레이크 구동을 위한 지연은 없습니다.

Coord[x].AbortAllMode 가 2 로 설정된 경우 마치 **abort** 명령이 좌표계에 전송된 것처럼 좌표계의 모든 모터는 처음에 클로즈드 루프 제어되어 정지합니다. 다음으로 각각의 모터의 명령 속도가 0 에 도달하면 **dkill** 명령이 전송된 것과 동일하게 "지연이 있는 kill"을 실행하여 즉시 브레이크를 구동하고(브레이크가 **Motor[x].pBrakeOut** 으로 지정된 경우), **Motor[x].BrakeOnDelay** 로 지정된 간격 뒤에 모터의 Disable 이 다음으로 이어집니다.

Coord[x].AbortAllMode 가 3 으로 설정되는 경우 좌표계는 "전체 Abort" 입력의 영향을 받지 않습니다.



이 "글로벌한 Abort" 기능은 모터에서 동력을 분리할 필요가 있는 경우(IEC-61800-5-2 기계 안전 규격에 기초한 카테고리 0 또는 1 과 같은)에는 단독으로는 적응하지 않습니다.

Coord[x].AbortTimeBase 좌표계의 Abort 타임 베이스

설명: Abort 감속에서 사용되는 최소 타임 베이스값

범위: 음이 아닌 부동 소수점

단위: msec

기본값: 0.0

Coord[x].AbortTimeBase 는 Abort 처리가 소프트웨어, 하드웨어 명령 또는 제어 정지의 원인이 되는 이상에 의한 발생인지 관계없이 "Abort" 처리 중인 모터 감속 실행 시의 좌표계에서 사용되는 최소의 타임 베이스값을 지정합니다. 해당 파라미터의 목적은 Abort 처리 시작 시의 타임 베이스값에 관계없이 적절하게 제어된 Abort 감속을 즉시 명령하는 것입니다.

Coord[x].AbortTimeBase 는 2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능입니다. 기본값 0.0 의 경우 해당 동작은 타임 베이스값이 0% 이상, 100% 이하인 경우에는 이 파라미터가 없었던 이전 버전의 펌웨어와 호환성이 있습니다.

Abort 처리 시작 시에 순간 타임 베이스값과 목표 타임 베이스값 모두가 **AbortTimeBase** 보다 크고 100% 이하인 경우 (**Coord[x].AbortTimeBase < Coord[x].TimeBase**, **Coord[x].DesTimeBase <= Sys.ServoPeriod**)에 타임 베이스값은 변경되지 않습니다. 그리고 Abort 감속 곡선은 현재의 타임 베이스값을 사용합니다. 예를 들어 타임 베이스가 50%인 경우 감속 곡선이 100%로 감속하는 경우에 비해 1/4 의 감속률이며 시간은 2 배가 걸립니다.

Abort 처리 시작 시에 순간 또는 목표 타임 베이스값 중 하나가 음의 값을 포함하여 **AbortTimeBase** 와 +100% 사이의 범위 밖인 경우(**Coord[x].TimeBase** 또는 **Coord[x].DesTimeBase < Coord[x].AbortTimeBase**, **Coord[x].TimeBase** 또는 **Coord[x].DesTimeBase > Sys.ServoPeriod**)에 타임 베이스값은 +100%로 즉시 설정되고 **Coord[x].pDesTimeBase** 가 기본값인 **Coord[x].DesTimeBase.a** 와 달라진 경우 해당 기본값으로 변경되어 그 결과 "내부" 타임 베이스가 사용됩니다. 좌표계의 모든 모터의 현재 속도와 가속도는 연속성을 가진 채로 재스케일링되며 그 다음으로 감속 곡선은 재스케일링된 값으로 감속 시작을 산출하여 **Motor[x].AbortTa** 와 **Motor[x].AbortTs** 에 따라 실행됩니다.

타임 베이스가 Abort 처리 시작 시 정확하게 0.0 인 경우(좌표계가 피드 홀드 모드일 때와 같이) 현재 속도가 0.0 으로 재계산된다는 점에 주의하십시오. 따라서 Abort 감속 곡선은 필요 없습니다. 모션 프로그램이 0%의 타임 베이스에서 이미 일시 정지되어서 해당 프로그램을 재개하지 않고 대신 다른 프로그램을 로드하며 그리고/또는 실행하는 경우에 Abort 명령이 자주 사용됩니다.

이 상태에서는 최종적인 타임 베이스 소스와 값은 **Coord[x].AbortTimeBase** 가 0.0 과 같은지, 또는 0.0 보다 큰지에 따라 다릅니다. 0.0 과 같은 경우 순간 타임 베이스는 명령(비 피드 홀드(Non-feed hold)) 타임 베이스로 즉시 설정됩니다. 그리고 타임 베이스 소스 레지스터는 변경되지 않습니다. 0%의 타임 베이스가 피드 홀드에 따르지 않는 경우에는 그대로 0%입니다. 이 동작은 해당 파라미터가 없었던 펌웨어의 이전 버전과 호환성이 있습니다.

따라서 **Coord[x].AbortTimeBase** 가 0.0 보다 큰 경우 타임 베이스는 100%의 값으로 설정되고 **Coord[x].pDesTimeBase** 는 **Coord[x].DesTimeBase.a** 의 기본값으로 설정됩니다. 그리고 내부 타임 베이스가 사용됩니다.

다음 표에 **Coord[x].AbortTimeBase** 가 0.0 보다 큰 경우의 다른 시작 타임 베이스에 대한 타임 베이스 동작을 요약합니다. MATB(최소 Abort 타임 베이스)는 $(100\% * \text{Coord[x].AbortTimeBase} / \text{Sys.ServoPeriod})$ 와 같아집니다.

시작 시의 %값	시작 시의 타임 베이스 및 명령 타임 베이스	결과의 %값, 소스	감속 곡선의 실행?
<0.0%	어느 한쪽이 <0.0	+100%, 내부	예
= 0.0%	TimeBase=0.0	+100.0%, 내부	아니오
0.0%, <MATB%	양쪽>0.0 어느 한쪽이 < AbortTimeBase	+100.0%, 내부	예
>= MATB%, <=100.0%	양쪽>= AbortTimeBase 양쪽<= Sys.ServoPeriod	값 또는 소스의 변경 없음	예
>100%	어느 한쪽이 > Sys.ServoPeriod	+100%, 내부	예

항상 Abort 곡선이 100% 타임 베이스(즉, 항상 프로그램 레이트)로 실행되기를 바라는 경우 **Coord[x].AbortTimeBase** 는 **Sys.ServoPeriod** 와 같게 설정되어야 합니다.

다음 표에 **Coord[x].AbortTimeBase** 가 0.0 과 같은 경우의 다른 시작 타임 베이스에 대한 타임 베이스 동작을 요약합니다.

시작 시의 %값	시작 시의 타임 베이스 및 명령 타임 베이스	결과 %값, 소스	감속 곡선의 실행?
<0.0%	어느 한쪽이 <0.0	+100%, 내부	예
= 0.0%	TimeBase=0.0	(= DesTimeBase), 소스 변경 없음	아니오
> 0.0%, <=100%	양쪽>0.0 양쪽<= Sys.ServoPeriod	값 또는 소스의 변경 없음	예
>100%	어느 한쪽이 > Sys.ServoPeriod	+100%, 내부	예

시작 시의 타임 베이스가 0%보다 작거나 100%보다 큰 경우에 이전 펌웨어 버전과 호환성은 없지만 잘 제어된 감속 곡선을 생성합니다.

예

4kHz 의 서보 갱신 주파수를 가지는 시스템에서는 **Sys.ServoPeriod** 는 0.250(msec)으로 설정됩니다. 60% 이하의 임의의 타임 베이스값에서 Abort 곡선이 100%로 실행되어야 하는 경우 **Coord[x].AbortTimeBase** 는 0.250 의 * 0.6 = 0.150 으로 설정되어야 합니다.

Coord[x].AddedDwellTime 좌표계의 블렌드 무효 시 드웰 시간

설명: 블렌드 무효 시의 드웰 시간

범위: 음이 아닌 정수

단위: 실시간 인터럽트 주기

기본값: 0

기준 I 변수명: Isx82

Coord[x].AddedDwellTime에서는 블렌드 무효 시의 프로그램 동작 사이에 자동으로 삽입되는 드웰 시간을 지정합니다. 이 드웰 시간은 블렌드 되지 않은 프로그램 동작(**RAPID** 모드 동작, 프로그램된 원점 서치 동작) 뒤 또는 본래 블렌드 되지만 블렌드가 무효가 된 동작 뒤에 삽입됩니다. 그것은 **Coord[x].NoBlend** 가 1 로 설정되었거나 코너가 **Coord[x].CornerBlendBp** 와 **Coord[x].p**에 의해 지정된 각도보다 예각인 경우 무효가 됩니다(코너가 **Coord[x].CornerBlendBp**에 의해 지정된 각도보다 예각이지만 **Coord[x].CornerDwellBp**에 의해 지정된 각도만큼 예각이 아닌 경우 블렌드는 코너에서 무효가 됩니다. 그러나 드웰은 추가되지 않습니다). 이 드웰의 단위는 실시간 인터럽트 주기(**Sys.RtIntPeriod+1** 서보 사이클)에서 직접 프로그램의 드웰과 같이 msec 가 아닙니다.

Coord[x].InPosTimeout 가 0 인 경우 이 드웰 시간은 이전 동작 실행 완료 직후에 삽입됩니다. **Coord[x].InPosTimeout** 가 0 보다 큰 경우 좌표계의 모든 축이 "인포지션" 상태임이 확인한 후 이 드웰 시간이 삽입됩니다.

Coord[x].AltFeedrate 좌표계의 비 벡터축 프로그램 속도

설명: 비 벡터축의 프로그램 속도

범위: 음이 아닌 부동 소수점

단위: 사용자 축 속도 단위

기본값: 1.0

기준 I 변수명: Isx86

"비 피드 레이트" 축의 동작이 주요한 경우 **Coord[x].AltFeedrate** 는 피드 레이트 지정된 블렌드 이동(**linear** 또는 **circle** 모드)의 속도를 제어합니다. "피드 레이트" 또는 "벡터 피드 레이트" 축은 **frax** 명령으로 지정되며 X, Y 와 Z 축은 기본값인 피드 레이트 축입니다.

Coord[x].AltFeedrate 가 0.0 보다 큰 경우 Power PMAC에서는 피드 레이트 축의 벡터 거리를 지정된 피드 레이트(프로그램의 **F** 값 또는 **Coord[x].Tm**)로 나눠서 얻은 벡터 피드 레이트 축의 동작 시간과 비 피드 레이트 축의 최장 거리를 **Coord[x].AltFeedrate** 로 나눠서 얻은 비 피드 레이트 축의 동작 시간과 비교합니다. 그리고 피드 레이트와 비 피드 레이트의 모든 축에서 해당 2 가지 시간 중 긴 쪽을 동작 시간으로 사용합니다.

Coord[x].AltFeedrate 가 0.0 이고 Power PMAC 이 피드 레이트 지정된 동작의 벡터 거리가 제로(즉, 벡터 피드 레이트 축의 이동이 없음)라고 판단하는 경우에 비 피드 레이트 축의 동작은 제로의 동작 시간으로 지정되고 따라서 가속 시간으로 제어됩니다.

Coord[x].AltFeedrate 의 주요 사용 목적은 다음 2 가지입니다. 첫 번째는 피드 레이트 모드에서 1 행에 비 피드 레이트 축만 명령을 받는 경우 자동으로 "비 피드 레이트" 축의 동작으로 제어합니다. 보통 시스템은 직선 축만 벡터 피드 레이트 축이며 직선/회전축의 복합 시스템에서의 회전축에 적용됩니다.

두 번째는 프로그램의 피드 레이트가 무효가 되는 고속 "드라이 런" 모드를 가능하게 합니다. 좌표계의 어떤 축도 벡터 피드 레이트 축이 아닌 경우(**nofrax** 명령이 실행된 경우) **Coord[x].AltFeedrate** 는 프로그램의 **F** 값에 관계없이 모든 동작에 사용됩니다.

예:

```
Coord[1].FeedTime=1000 // 속도를 /초로 지정
Coord[1].AltFeedrate=5 // 5 사용자 단위/초의 대체 피드 레이트

inc ..... // 이동 거리로 지정된 동작
frax(X,Y,Z).... // X, Y 와 Z 축은 벡터 피드 레이트 축
X20 C5 F10.... // 벡터 축 동작 시간 = 20 단위/10(단위/sec) = 2sec
..... // 비벡터 축 동작 시간 = 5 단위/5(단위/sec) = 1sec
..... // 2sec 를 사용(벡터 피드 레이트 제어)
X10 C20..... // 벡터 축 동작 시간 = 10 단위/10(단위/sec) = 1sec
..... // 비벡터 축 동작 시간 = 20 단위/5(단위/sec) = 4sec
..... // 4sec 를 사용(대체 피드 레이트 제어)
C20..... // 동작 시간 = 20 단위/5(단위/sec)= 4sec
```

Coord[x].AutoTxyzScale 좌표계의 변환 매트릭스 재스케일링

설명: 자동적인 변환 매트릭스(행렬)의 재스케일링 제어

범위: 0 ..1

단위: 논리값

기본값: 0

Coord[x].AutoTxyzScale에서는 Power PMAC 이 좌표계의 XYZ 직행 좌표 공간의 선택된 변환 매트릭스 스케일링에 따라서 변환 매트릭스 재스케일링 계수 **Coord[x].TxyzScale**를 자동으로 계산할지 여부를 지정합니다. 기본값 0 인 경우 자동적인 연산은 수행되지 않습니다.

Coord[x].AutoTxyzScale 이 1 로 설정되는 경우 Power PMAC 은 선택된 변환 매트릭스인 3x3 XYZ 부분 행렬에 기초한 비저장 설정 요소 **Coord[x].TxyzScale**의 값을 자동으로 연산합니다. 변환 매트릭스가 프로그램 **tset{데이터}** 명령으로 좌표계에 선택되지 않은 경우 이 연산은 수행되지 않습니다.

기술적으로 말하면 **TxyzScale**은 선택된 변환 매트릭스의 3x3 XYZ 부분 행렬 행렬식의 세제곱근으로 연산됩니다. X, Y 그리고 Z 축의 프로그램된 단위가 선택된 변환 매트릭스에서 재스케일링되었을 때도 X, Y, Z 축의 "베이스"(변환되지 않은) 단위로 피드 레이트와 공구 반경을 유지합니다. 사용 시에는 피드 레이트와 공구 반경의 명령값은 동작에서 사용되기 전에 **TxyzScale**로 나누어집니다.

이러한 연산을 적절하게 하기 위해서는 변환된 X, Y 그리고 Z 축은 모두 동일한 스케일링을 가져야 합니다. 해당 축이 실제로는 1 개 또는 2 개만 사용되는 경우에도(예를 들어 2D XY 시스템) 추가되는 1 축 또는 여러 개의 축도 동일하게 재스케일링되어야 합니다.

Coord[x].CCAddedArcBp 좌표계의 공구 직경 보정 추가 원호 브레이크 포인트

설명: 공구 직경 보정의 바깥쪽 코너에서의 추가 원호에 대한 브레이크 포인트

범위: -1.0 ..1.0(부동 소수점)

단위: 각도 코사인

기본값: 0.0(90°에 상당하는 각도)

기존 I 변수명: Isx99

Coord[x].CCAddedArcBp는 2D 공구 직경 보정에서 원호 동작이 추가되는 경우의 바깥쪽 코너 각도와 진입하여 밖으로 나가는 이동이 보정된 교점에서 직접 결합되는 경우의 바깥쪽 코너 각도 사이의 좌표계의 최소치를 지정합니다.

Coord[x].CCAddedArcBp는 진입하여 밖으로 나가는 **linear** 모드 동작 간 유향각의 변화인 "등가" 코사인으로 표시됩니다. 그리고 이는 보정되지 않은 교점과 보정된 교점 간의 거리에 대한 공구 반경의 동일한 비율을 생성합니다(유향각의 변화는 180°에서 코너의 개선 각도를 뺀 각도와 같아집니다).

따라서 -1.0 과 +1.0 사이의 값이 됩니다. 2 가지 이동이 이동 경계에서 동일한 윗향각을 지나는 경우(즉, 동일한 방향으로 이동), 윗향각의 변화는 0°이며 코사인은 1.0 이 됩니다. 윗향각의 변화가 증가하면 코너는 점점 예각이 되고 윗향각의 변화인 코사인은 감소합니다. 완전한 역회전 궤적에서는 윗향각의 변화는 180°가 되고 코사인은 -1.0 이 됩니다. 윗향각의 변화는 normal 명령으로 정의된 평면(기본값은 XY 평면)으로 구할 수 있습니다. 나아가 코너가 이 평면에 수직인 축 동작을 포함하는 경우 관계하는 것은 이 평면상에 대한 투영 동작입니다.

2 개의 **linear** 모드 이동으로 형성되는 코너에 **Coord[x].CCAddedArcBp** 는 윗향각 변화의 최소치인 코사인으로 표시됩니다. **circle** 모드 이동에서 최소치 각도는 원호 이동이 "오목한 면"또는 "볼록한 면"을 생성하는지 여부에 따라 다르므로 보정 안 된 교점과 보정된 교점 간의 거리와 공구 직경 반경의 비율에 따라서 Power PMAC 이 선택하는 것과 같이 원호 반경에 따라 다릅니다. **circle** 모드 이동에서는 보정된 교점이 없는 경우도 있을 수 있으며 이 경우 바깥쪽 코너에 대응하기 위해 원호 이동이 항상 추가됩니다.

코너에서 윗향각의 변화(등가) 코사인이 **Coord[x].CCAddedArcBp** 보다 작은 경우(윗향각의 큰 변화로 좀 더 예각인 코너) 진입하여 밖으로 나가는 이동을 결합하기 위해 Power PMAC 은 공구 반경과 같은 반경의 원호 이동을 자동으로 추가합니다. 이는 예각 코너의 바깥쪽으로 도는 동작을 하는 경우 공구가 지나치게 밖으로 나가는 것을 방지합니다. 추가 원호의 최소 시간은 Ta 와 Ts 로 설정되는 가속 시간의 선언값이라는 점에 주의하십시오. 이는 추가 원호가 매우 짧은 경우 약간의 각도 변화에 대해서는 바람직하지 않은 감속을 일으킬 수도 있습니다.

코너에서 윗향각의 변화(등가) 코사인이 **Coord[x].CCAddedArcBp** 이상인 경우(윗향각의 작은 변화로 좀더 완만한 코너) Power PMAC 은 보정된 교점에서 진입하는 이동과 밖으로 나가는 이동을 직접 결합합니다. 이러한 이동이 서로 블렌드 될 때 이 시점에서 일반적인 블렌드 알고리즘이 사용됩니다.

Coord[x].CCAddedArcBp 의 동작은 **Coord[x].CornerBlendBp** 과 동일한 기능의 동작과 대부분 관계없습니다. **Coord[x].CornerBlendBp** 는 코너의 윗향각 변화에 따라서 진입하는 이동과 밖으로 나가는 이동이 서로 블렌드 되는지 여부를 제어합니다. 공구 직경 보정의 유·무효에 관계없이 또는 공구 직경 보정이 유효한 경우 코너가 안쪽인지 바깥쪽인지 관계없이 **Coord[x].CornerBlendBp** 는 동작합니다. 단, 원호가 추가된 바깥쪽의 보정된 코너의 경우 Power PMAC 은 추가된 원호 이동과 진입하여 밖으로 나가는 이동 사이의 "코너" 각도를 연산합니다. 해당 각도는 대부분 0°이며 따라서 항상 블렌드 됩니다.

예

윗향각의 변화가 60°보다 큰(120°보다 작은 개선했 각도) 경우 원호 이동이 바깥쪽 코너 보정에 추가되도록 하기 위해서는 **Coord[x].CCAddedArcBp** 는 0.5 로 설정해야 합니다. 왜냐하면 $\cos\Delta\theta = \cos 60^\circ = 0.5$ 가 되기 때문입니다.

Coord[x].CCCtrl 좌표계의 공구 직경 보정 제어

설명: 2D 공구 직경 보정의 제어 비트

범위: 0 ..7

단위: 비트 필드

기본값: 0

기존 I 변수명: Isx84

Coord[x].CCCtrl 은 2D 공구 직경 보정 알고리즘의 몇 가지 모달(modal)의 특징을 지정합니다. 이는 몇 가지 제어 비트로 구성됩니다.

Coord[x].CCCtrl 의 비트 0(값 1)은 2D 공구 직경 보정의 바깥쪽 코너에서 원호가 추가되고 블렌드가 무효가 되는 경우의 프로그램 이동이 정지하는 장소를 제어합니다. **Coord[x].CCCtrl** 의 비트 0 이 0 인 경우 프로그램 이동은 추가된 원호의 종료 시에 정지합니다. **Coord[x].CCCtrl** 의 비트 0 이 1 인 경우 프로그램 이동은 추가된 원호의 시작에서 정지합니다.

나아가 **Coord[x].CCCtrl** 의 비트 0 은 블렌드의 유·무효에 관계없이 버퍼링된 동기 변수의 대입 명령이 코너의 어디에서 실행되는지를 제어합니다. 0 으로 설정되는 경우 모든 버퍼링된 대입 명령은 추가된 원호가 종료 후 나가는 이동을 시작할 때 실행됩니다. 1 로 설정되는 경우 모든 버퍼링된 대입 명령은 추가된 원호를 시작할 때 실행됩니다. 또한 온라인 t 명령과 버퍼 tread 명령의 응답으로 통지되는 이동 목표 위치는 동기의 대입이 실행되는 위치가 됩니다.

싱글 스텝 조작(s), 프로그램 종료 조작(q 또는 l) 또는 **Coord[x].NoBlend** 를 1 로 설정하면 블렌드가 무효가 되는 경우 프로그램 이동은 추가된 원호 종료 시 항상 정지합니다. 보정 평면에서 코너를 형성하는 2 개의 궤적 사이에 평면 외 궤적이 있는 경우 이는 추가된 원호의 종료 시 항상 실행됩니다.

Coord[x].CCCtrl 의 비트 1(값 2)은 2D 공구 직경 보정 알고리즘에서 간섭이 감지된 경우 수행되는 동작을 지정합니다. 간섭 상태는 보정 안 된 이동과 비교하여 보정 이동의 방향이 역회전하면 감지됩니다. 또는 보정된 경로와 사전 버퍼의 또 하나의 이동의 보정된 경로가 교차하면 감지됩니다. 이는 보통 "오버 컷"이 발생함을 의미합니다.

Coord[x].CCCtrl 의 비트 1 이 0 인 경우 간섭이 감지되면 프로그램 동작은 에러 상태에서 정지합니다. 충분한 사전 계산이 **Coord[x].CCDistance** 로 지정된 경우 오버 컷이 발생하기 전에 해당 정지가 일어납니다. 간섭과 오버 컷이 프로그래밍 에러를 암시하는 경우 보통 이 동작 모드가 사용됩니다.

Coord[x].CCCtrl 의 비트 1 이 1 인 경우 프로그램 동작은 간섭을 "정정"하려고 시도하고 최초의 교점과 2 번째 교점 간의 보정된 경로를 제거합니다. 그 결과 교점에서의 진입 경로와 나가는 경로를 직접 결합합니다. 간섭을 정정할 수 없는 경우 프로그램은 정지합니다. 해당 동작 모드는 보통 부품의 작은 형상에 들어갈 수 없는 큰 공구를 사용한 "황삭" 경로에 사용됩니다.

Coord[x].CCCtrl 의 비트 2(값 4)는 간섭 확인이 "운전" 모드 또는 "테스트" 모드로 수행되는지 여부를 지정합니다. 비트 2 가 0 으로 설정되는 경우 "운전" 모드가 지정됩니다. 그리고 알고리즘은 간섭 감지에서 정지하거나(비트 1 = 0) 또는 간섭 상태를 "정정"합니다. 보통 해당 모드는 실제 가공에 사용됩니다.

Coord[x].CCCtrl 의 비트 2 가 1 인 경우 "테스트" 모드가 지정됩니다. 그 알고리즘은 간섭 확인을 바이패스하여(비트 1 = 0 인 경우) 오버 컷의 발생을 허용하거나 또는 가능하다면 정정하고(비트 1 = 1), 불가능한 경우에 속행하여 오버 컷의 발생을 허용합니다. 그 중 하나를 선택합니다. 해법을 계산할 수 없는 경우에만 프로그램은 정지합니다. 보통 파트 프로그램을 디버그하기 위해 이 모드가 사용됩니다.

Coord[x].CCCtrl 은 풀 워드 요소 **Coord[x].Control[0]**의 비트 18-21 을 구성합니다.

Coord[x].CCDistance 좌표계의 공구 직경 보정 룩 어헤드 거리

설명: 공구 직경 보정 평면 내의 룩 어헤드 거리

범위: 0 ..255

단위: 프로그램 이동

기본값: 0

Coord[x].CCDistance 는 Power PMAC 이 사전 계산하는 "평면 내"의 이동 동작 수를 지정합니다. 이는 2D 공구(커터) 직경 보정이 유효한 경우에 현재 동작 계산 종료 전에 수행됩니다. Power PMAC 은 다음의 보정된 2 개 교점을 발견하기 위해 적어도 2 개의 평면 내 이동을 항상 사전 계산해야 합니다. 최초의 교점은 현재 보정된 이동의 종료점을 구성합니다. 2 번째 교점에서는 다음의 보정한 이동에서 방향 역회전이 있는지 여부를 확인합니다. 이는 현재 이동이 간섭에 의해 오버 컷을 일으킨다는 것을 의미합니다. **Coord[x].CCDistance** 가 2 보다 작은 경우 Power PMAC 은 다시 미리 2 개의 이동을 확인합니다.

간섭이 감지되는 경우에 동작을 계속 하기 위해 1 개 이상의 보정된 "반전" 이동을 파기할 수 있다는 것을 사용자가 원하는 경우 **Coord[x].CCDistance** 는 해당 이동에 대응할 수 있도록 충분히 큰 값으로 설정되어야 합니다. 즉, **Coord[x].CCDistance** 는 적어도 폐기될 수 있는 평면 내 이동 수에 2 를 더한 값으로 설정되어야 합니다.

이러한 사전 계산된 이동을 저장하는 버퍼 크기를 정의하는 **Coord[x].CCSize** 에서는 **Coord[x].CCDistance** 이상의 값으로 설정되어야 합니다(이는 "평면 외"와 "이동 없음" 동작도 포함하는 데 충분한 크기가 필요합니다).

사용 시 이 길이의 사전 계산은 사용자에게는 거의 완전히 보이지 않습니다. 보정을 도입하는 경우 Power PMAC 은 도입 이동을 실행하기 전에 이 수의 평면 내 이동을 버퍼에 채웁니다. 보정된 이동 실행 중에 Power PMAC 은 자동으로 이 수의 평면 내 이동을 버퍼에 계속 채웁니다. 이것은 실행 중인 이동에 선행하여 수행됩니다. 보정이 무효가 되면 Power PMAC 은 버퍼에 남아 있는 이동을 사용하여 동작하고 버퍼를 자동으로 비웁니다.

싱글 스텝 모드에서 Power PMAC 은 버퍼의 1 개 이동을 실행하면서 버퍼를 필요한 범위로 계속 채우기 충분한 프로그램 이동을 계산합니다.

더 긴 사전 계산이 필요한 경우 더 짧은 계산과 비교하면 프로그램의 인접 이동 실행 전에 표준 변수의 값 대입 명령이 다시 실행됩니다. 동기 변수의 값 대입 명령의 실행은 룩 어헤드의 길이와 관계없이 프로그램의 다음 이동 명령의 실제 실행 시작까지 지연됩니다.

Coord[x].CCSize 좌표계의 공구 직경 보정 이동 버퍼 크기

설명: 공구 직경 보정의 이동 버퍼 길이

범위: 0 ..255

단위: 프로그램 이동

기본값: 0

2D 공구(커터) 직경 보정이 유효한 경우 **Coord[x].CCSize** 는 프로그램 이동의 버퍼 길이를 지정합니다. 이것은 2D 보정을 가능하게 하기 위해 적어도 2 로 설정해야 합니다.

2D 공구 직경 보정에서는 몇 가지 사유에 따라 이동이 버퍼링 되어야 합니다. 첫째, 이동의 보정된 종점을 산출하기 위해 다음 이동 시작 방향을 알고 있어야 합니다. 그래야 그 보정된 교점을 산출할 수 있게 됩니다. 둘째, 시간 내에 기본적인 오버 컷 상태를 포착하기 위해 다음 교점도 알고 있어야 합니다(왜냐하면 보정된 이동의 방향 역회전이 간섭을 나타내기 때문입니다). 그것은 다음 이동도 동일하게 계산해야 한다는 것을 의미합니다. 간섭 상태가 감지될 때 역회전되는 이동을 하나 이상 버려야 하는 경우 하나 이상의 이동이 버퍼에 추가되어야 합니다. 그래야 실제로 실행 중인 이동을 충분히 완전하게 유지할 수 있습니다.

마지막으로 보정 평면에 동작 성분이 없는 이동(평면에 수직인 이동 또는 드웰이나 딜레이와 같은 "이동 없는" 동작)의 가능성이 있는 경우 이것이 버퍼링 되어야 합니다. 그렇게 하면 Power PMAC 은 충분히 끝까지 계산할 수 있게 되고 이것 뒤의 다음 2개의(또는 더 많은) "평면 내" 이동을 계산합니다.

따라서 버퍼는 보정 교점의 산출과 기본적인 간섭 상태의 검출을 위해 충분한 "평면 내" 이동을 포함하는 충분한 크기이어야 합니다(최소한 2). 나아가 간섭의 원인이 되는 이동과 보정 평면에서의 이동 없는 동작도 버릴 수 있으려면 충분한 크기이어야 합니다.

예를 들어 평면 내 이동을 재개하기 전에 공구 직경 보정 중에 {dwell, cutter up, dwell, cutter down, dwell}과 같은 일련의 동작이 있는 경우 2개의 보정된 이동 사이에 보정 평면 내의 5개의 이동이 없는 동작이 있습니다. 그러므로 **Coord[x].CCSize** 는 적절한 보정을 위해 적어도 7 로 설정되어야 합니다.

Power PMAC 은 좌표계의 공구 직경 보정 버퍼를 위해 (880 * **CCSize**)바이트의 데이터를 예약합니다. 이 버퍼는 좌표계의 모든 버퍼(공구 직경 보정, 목표 위치 그리고 동적인 룩 어헤드)를 위해 할당된 16M 바이트 영역의 일부입니다.

공구 직경 보정 버퍼는 속도와 가속도와 같은 동적인 양을 제한하는 룩 어헤드 버퍼와는 독립됩니다. 모든 버퍼는 다른 버퍼와 관계없이 사용할 수 있습니다.

Coord[x].CCSize 는 얼마만큼 사전 계산된 이동이 실제로 버퍼링 되는지를 지정하는 것이 아닙니다. 관련되는 요소인 **Coord[x].CCDistance** 에 의해 지정됩니다. 정의된 버퍼 크기는 사전 계산된 이동 전체를 유지하기에 충분한 크기로 설정되어야 합니다.

Coord[x].Control[j] 좌표계의 제어 요소

설명: 요소를 유효로 하기 위한 좌표계의 풀 워드 요소

범위: \$00000000..\$FFFFFFFF

단위: 비트 필드

기본값: \$00000000

Coord[x].Control[j]는 좌표계의 기본적인 기능을 유효로 하는 설정 요소로 구성되는 풀 워드 요소 배열입니다. (현재는 **Control[0]**으로만 구성되지만 앞으로 더 많은 요소로 확장할 수도 있습니다.)

Coord[x].Control[0]은 다음과 같은 부분 워드 요소로 구성됩니다.

구성 요소	비트	기능
--	31	(추후 사용하기 위해 예약)
NoBlend	30	블렌드 이동 무효 제어
SoftLimitStopDis	29	소프트웨어 한계 제어에 대한 동작
RapidVelCtrl	28	Rapid 이동의 속도 모드
StepMode	27 -24	모션 프로그램의 싱글 스텝 모드 제어
SegLinToPvt	23 -22	Linear 동작의 PVT 실행 유효
CCCtrl	21 -18	공구 직경 보정 제어
Ndisplay	17	동기의 행 레이블 표시 제어
HomeRequired	16	모션 프로그램 실행 전의 원점 복귀
--	15 -08	(추후 사용하기 위해 예약)
GoBack	07 -00	블렌드 무효화 이전 모션 프로그램의 행 복귀

사용자는 요소의 설정과 해당 설정 확인을 위해 스크립트 환경에서 보통 이러한 부분 워드 요소에 액세스합니다. Power PMAC 은 효율화를 위해서 풀 워드 요소로 저장 및 복원합니다.

풀 워드 요소의 값이 스크립트 환경 중에 참조되는 경우 부분 워드 요소 값이 10 진수로 통지되어도 Power PMAC 은 16 진수 형식으로 값을 반환합니다.

해당 요소의 기능에 대한 자세한 설명은 각 부분 워드 요소의 항목을 참조하십시오.

Coord[x].CornerAccel 좌표계의 블렌드 코너 가속도

설명: 블렌드 이동에서의 코너 가속도

범위: 음이 아닌 부동 소수점

단위: 축 단위/msec²

기본값: 0.0(무효)

양의 **Coord[x].CornerAccel** 은 서로의 동작을 블렌드할 때의 Power PMAC 이 사용하는 벡터 코너 가속도를 지정합니다. 벡터 속도와 코너 각도에 따라서 Power PMAC 은 이 가속도를 실현하는 블렌드 시간을 계산합니다. 해당 블렌드 시간은 벡터 속도와 함께 코너 "크기"를 결정합니다. 이 파라미터의 가장 공통되는 사용 용도는 지정된 가속도를 사용하여 가능한 한 작은 코너 블렌드를 실행하는 것입니다.

Coord[x].CornerRadius 가 양의 값으로 설정된 경우 **Coord[x].CornerAccel** 이 사용되지 않는다는 점에 주의하십시오. 코너는 가속도가 아니라 크기로 정의되기 때문입니다. **Coord[x].CornerBlendBp** 로 정의된 최소치보다 더 예각인 코너의 경우도 포함하여 어떤 이유로 코너에서 블렌드가 무효가 되는 경우 **Coord[x].CornerAccel** 은 코너에 사용되지 않습니다.

코너 각도는 **normal** 벡터로 지정된 평면으로 계산됩니다. 한 쪽 이동이 **circle** 모드 이동인 경우 블렌드 되지 않은 코너 직후의 이동 방향이 코너 각도를 계산하기 위해 사용됩니다. 코너의 동작이 평면에 수직 성분을 포함하는 경우(예를 들어 지정된 XY 평면에 대한 Z 축 동작), 코너 각도의 계산에 사용되는 것은 지정된 평면상에 투영 궤적입니다. 벡터 피드 레이트가 진입하는 이동과 밖으로 나가는 이동 사이에서 변화하는 경우 두 피드 레이트의 RMS 평균이 계산 중에 사용됩니다. 단, 이 변화를 가져오는데 필요한 접선의 가속도는 코너 가속도 한계 계산에는 포함되지 않습니다.

Coord[x].CornerAccel 이 블렌드 시간을 제어하기 위해 사용되면 **Coord[x].Ta** (**ta{데이타}** 명령으로 설정할 수 있음)의 값은 사용되지 않습니다. 그렇지 않으면 **Coord[x].Ta** 로 제어합니다. 단, 그 값은 유지됩니다. 따라서 그 값을 재지정하지 않고 적절한 값으로(예를 들어 다음 이동 시퀀스의 최초의 가속도 등에) 그 이후에도 사용할 수 있습니다. **Coord[x].CornerAccel** 이 0.0 인 기본값으로 설정되는 경우 그때는 **Coord[x].Ta** 의 값이 유효가 되고 블렌드 시간을 지정합니다.

지정된 코너 가속도를 얻기 위해 필요한 블렌드 시간을 계산할 때 **Coord[x].Ts** 의 현재값(**ts{데이타}** 명령으로 설정할 수 있음)은 변경 없이 사용됩니다. 합계 블렌드 시간은 계산된 "Ta" 시간과 사전에 지정된 Ts 시간의 조합입니다.

지정된 코너 가속도를 얻기 위해 계산된 블렌드 시간이 **Coord[x].Td** 보다 작은 경우 **Coord[x].Td** 의 값이 대신 사용됩니다(**Coord[x].Td** 가 **Coord[x].Ts** 보다 작은 경우 **Ts** 시간이 유효한 **Td** 시간으로 기능합니다). 계산된 블렌드 시간이 매우 커서 진입하는 이동의 "일정 속도" 기간 시작 전에 블렌드를 시작해야 하는 경우 일정 속도 기간이 시작될 때 블렌드가 시작되도록 블렌드 시간이 축소됩니다.

이어서 모터 가속도 제한이 세그먼트의 룩 어헤드 알고리즘에 적용되는 경우 경로와 코너 크기가 변경되지 않는다는 점에 주의하십시오. 경로에 따른 속도는 모터 제한을 준수하기 위해 낮춰질 수도 있습니다.

Coord[x].CornerError(2014 년 1/4 분기에 출시한 V1.6 펌웨어의 새로운 기능)가 0.0 보다 큰 경우 Power PMAC 은 **Coord[x].CornerAccel** 을 사용하여 연산된 블렌드 경로와 프로그램의 코너 포인트 간의 오차량을 비교합니다. 이 오차가 **Coord[x].CornerError** 보다 큰 경우 블렌드 시간을 감소시켜서 이 오차 최소치를 초과하지 않게 합니다. 이 시간 변경은 프로그램 피드 레이트에서는 가속도값이 초과하는 것을 의미합니다. 이를 위해 룩 어헤드 알고리즘이 코너에서 명령 속도를 감소시켜 제한 내에 가속도를 유지하는 데에 사용할 수 있습니다.

Coord[x].CornerBlendBp 좌표계의 코너 블렌드 브레이크 포인트

설명: 블렌드 브레이크 포인트의 코너 각도

범위: -1.0 ..1.0(부동 소수점)

단위: 각도 코사인

기본값: 0.0(무효)

기준 I 변수명: Isx83

Coord[x].CornerBlendBp 는 **linear** 와 **circle** 모드 이동에서 직접 블렌드 되는 코너 각도와 코너로 진입하여 밖으로 나가는 이동 사이에서 축이 정지하는 각도 간의 좌표계 최소치를 지정합니다. **Coord[x].NoBlend** 가 블렌드를 허가하는 0 으로 설정되는 경우만 **Coord[x].CornerBlendBp** 는 사용됩니다. **Coord[x].NoBlend** 가 1 로 설정되는 경우 블렌드는 어떤 이동 사이에서도 일어나지 않습니다. 그리고 **Coord[x].CornerBlendBp** 는 사용되지 않습니다.

Coord[x].CornerBlendBp 가 0.0 인 기본값으로 정확하게 설정되는 경우 코너 각도에 기초한 블렌드 판정은 수행하지 않습니다. 90°의 코너 각도 최소치가 필요한 경우(**Coord[x].CornerBlendBp** 는 본래 0.0 이지만), **Coord[x].CornerBlendBp** 는 예를 들면 0.00001 과 같은 정확히 0.0 이외의 값으로 설정하십시오.

Coord[x].CornerBlendBp 는 진입하는 이동과 밖으로 나가는 이동 간의 유향각의 변화의 코사인으로 표시됩니다(유향각의 변화는 180°에서 코너의 개선 각도를 뺀 각도와 같아집니다). 따라서 이는 -1.0 과 +1.0 사이의 값입니다. 2 가지 이동이 이동 경계에서 동일한 유향각을 지니는 경우(즉, 동일한 방향으로 이동) 유향각의 변화는 0°입니다. 그리고 코사인은 1.0 입니다. 유향각의 변화가 증가하면 코너는 더 예각이 됩니다. 그리고 유향각의 변화 코사인은 감소합니다. 완전한 역회전 이동일 때 유향각의 변화는 180°입니다. 그리고 코사인은 -1.0 입니다. 유향각의 변화는 **normal** 명령으로 정의된 평면(기본값은 XY 평면)으로 구할 수 있습니다. 나아가 코너가 이 평면에 수직인 축 이동을 포함하는 경우 관계하는 것은 해당 평면상에 대한 투영 이동입니다.

코너에서의 유향각의 변화의 코사인이 **Coord[x].CornerBlendBp** 보다 작은 경우(유향각의 큰 변화 즉, 더 예각인 코너)에 Power PMAC 은 두 이동 간의 블렌드를 자동으로 무효로 하고 진입하는 이동이 종료될 때 명령 궤적을 일단 정지합니다. **Coord[x].InPosTimeout** 이 0 보다 큰 경우 Power PMAC 은 좌표계의 모든 모터가 "인포지션"임을 확인합니다. 다음으로 더욱이 이 각도의 코사인이 **Coord[x].CornerDwellBp** 보다 작은 경우 **Coord[x].AddedDwellTime** 의 실시간 인터럽트 주기로 드웰이 실행됩니다. 마지막으로 밖으로 나가는 이동이 자동으로 시작됩니다.

코너에서의 유향각의 변화 코사인이 **Coord[x].CornerBlendBp** 이상인 경우(유향각의 작은 변화 즉, 더 둔각 코너)에 Power PMAC 은 보통 블렌드 알고리즘으로 진입하는 이동과 밖으로 나가는 이동을 직접 블렌드합니다.

Coord[x].CornerBlendBp 의 동작은 **Coord[x].CCAddedCornerBp** 의 동일한 기능의 동작과는 대부분 독립됩니다. **Coord[x].CCAddedCornerBp** 는 2D 공구 직경 보정 중의 바깥쪽 코너에서 하나의 원호 동작이 유향각의 변화에 따라서 삽입되는지 여부를 제어합니다. 공구 직경 보정의 유·무효에 관계없이 그리고 공구 직경 보정이 유효한 경우 코너가 안쪽인지 바깥쪽인지 관계없이 **Coord[x].CornerBlendBp** 는 동작합니다. 단, 삽입 원호의 바깥쪽이 보정된 코너인 경우 Power PMAC 은 진입하는 이동, 밖으로 나가는 이동 그리고 삽입 원호 이동 간의 "코너" 각도를 모두 계산합니다. 해당 각도는 대부분 0°입니다. 따라서 항상 블렌드 됩니다.

예

유향각의 변화가 30°보다 큰(150° 미만의 개선 각도) 경우에 동작 정지가 필요하면 **Coord[x].CornerBlendBp** 는 0.866 으로 설정합니다. 왜냐하면 $\cos \Delta\theta = \cos 30^\circ = 0.866$ 이 되기 때문입니다.

Coord[x].CornerDwellBp 좌표계의 코너 드웰 브레이크 포인트

설명: 드웰 브레이크 포인트의 코너 각도

범위: -1.0 ..1.0(부동 소수점)

단위: 각도 코사인

기본값: 0.0(무효)

기준 I 변수명: Isx85

Coord[x].CornerDwellBp 는 **linear** 와 **circle** 모드에서 자동으로 드웰을 개재하지 않고 수행하는 코너 각도와 실시간 인터럽트 주기로 표시되는 **Coord[x].AddedDwellTime** 의 드웰이 자동으로 삽입되는 코너 각도 사이의 최소치를 지정합니다. **Coord[x].NoBlend** 가 0 으로 설정되어 블렌드가 허가되는 경우에만 **Coord[x].CornerDwellBp** 가 사용됩니다. **Coord[x].NoBlend** 가 1 로 설정되는 경우 모든 이동 사이에 드웰은 수행되지 않습니다. 그리고 **Coord[x].CornerBlendBp** 는 사용되지 않습니다.

코너의 블렌드가 **Coord[x].CornerBlendBp** 의 동작에 의해 무효가 된 경우에만 코너 각도가 **Coord[x].CornerDwellBp** 에 대해 판정됩니다. 이는 **Coord[x].CornerDwellBp** 가 작용하기 위해서는 **Coord[x].CornerBlendBp** 보다 작은 값으로 설정되어야 한다는 것을 의미합니다.

Coord[x].CornerDwellBp 가 0.0 인 기본값으로 정확하게 설정되는 경우 드웰은 자동으로 삽입되지 않습니다. 90°의 코너 각도 최소치가 필요한 경우(**Coord[x].CornerDwellBp** 는 본래 0.0 이 되지만)에 **Coord[x].CornerDwellBp** 는 예를 들면 0.00001 과 같이 정확하게 0.0 이외의 값으로 설정하십시오.

Coord[x].CornerDwellBp 는 진입하는 이동과 밖으로 나가는 이동 간의 유향각의 변화의 코사인으로 표시됩니다(유향각의 변화는 180°에서 코너의 개선 각도를 뺀 각도와 같아집니다). 따라서 -1.0 과 +1.0 사이의 값이 됩니다. 2 가지 이동이 이동 경계에서 동일한 유향각을 지니는 경우(즉, 동일한 방향으로 이동) 유향각의 변화는 0°입니다. 그리고 코사인은 1.0 입니다. 유향각의 변화가 증가하면 코너는 더 예각이 됩니다. 그리고 유향각의 변화 코사인은 감소합니다. 완전한 역회전 궤적에서는 유향각의 변화는 180°입니다. 그리고 코사인은 -1.0 입니다. 유향각의 변화는 **normal** 명령으로 정의된 평면(기본값은 XY 평면)으로 구할 수 있습니다. 나아가 코너가 이 평면에 수직인 축 이동을 포함하는 경우 관계하는 것은 해당 평면상에 대한 투영 이동입니다.

코너에서의 유향각의 변화 코사인이 **Coord[x].CornerDwellBp** 보다 작은 경우(유향각의 큰 변화, 즉 더 예각인 코너)에 밖으로 나가는 이동이 시작되기 전에 Power PMAC 은 자동으로 실시간 인터럽트 주기로 설정된 **Coord[x].AddedDwellTime** 의 드웰을 추가 삽입합니다.

코너에서의 유향각의 변화 코사인이 **Coord[x].CornerDwellBp** 이상인 경우(유향각의 작은 변화, 즉 더 둔각 코너)에 Power PMAC 은 자동으로 드웰을 추가하지 않습니다.

Coord[x].CornerDwellBp 의 동작은 **Coord[x].CCAddedCornerBp** 의 동일한 기능의 동작과는 대부분 독립됩니다. **Coord[x].CCAddedCornerBp** 에서는 2D 공구 직경 보정의 바깥쪽 코너에서 하나의 원호 동작이 유향각의 변화에 따라서 삽입되는지 여부를 제어합니다. 공구 직경 보정의 유·무효에 관계없이 그리고 공구 직경 보정이 유효한 경우에 코너가 안쪽인지 바깥쪽인지 관계없이 **Coord[x].CornerDwellBp** 는 동작합니다. 단, 삽입 원호의 바깥쪽이 보정된 코너인 경우 Power PMAC 은 진입하는 이동, 밖으로 나가는 이동 그리고 삽입 원호 이동 간의 "코너" 각도를 모두 계산합니다. 해당 각도는 대부분 항상 0°입니다. 따라서 드웰은 추가 삽입되지 않습니다.

예

유향각의 변화가 45°보다 큰 경우(135° 미만의 개선 각도)에 동작 정지가 필요하면 **Coord[x].CornerDwellBp** 는 0.707 로 설정해야 합니다. 왜냐하면 $\cos\Delta\theta = \cos 45^\circ = 0.707$ 이 되기 때문입니다.

Coord[x].CornerError 좌표계의 코너 오차

설명: 블렌드 이동에서의 코너 오차

범위: 음이 아닌 부동 소수점

단위: 축 단위

기본값: 0.0(무효)

양의 **Coord[x].CornerError** 는 서로의 동작을 블렌드할 때의 Power PMAC 이 사용 하는 코너 오차를 지정합니다. 이 오차는 코너의 블렌드 된 경로와 코너의 프로그램 포인트 간의 최소 거리입니다. 벡터 속도와 코너 각도에 따라서 Power PMAC 은 이 오차를 초래하는 블렌드 시간을 연산합니다.

Coord[x].CornerRadius 가 양의 값인 경우 **Coord[x].CornerError** 는 사용되지 않는다는 점에 주의하십시오. 코너는 블렌드 오차가 아니라 크기로 정의되기 때문입니다. **Coord[x].CornerBlendBp** 로 정의된 최소치보다 예각인 코너인 경우도 포함하여 어떤 이유로 코너에서 블렌드가 무효가 되면 **Coord[x].CornerError** 는 코너에서 사용되지 않습니다.

Coord[x].CornerAccel 이 양의 값인 경우 블렌드 시간은 지정된 코너 가속도를 얻기 위해 처음에 계산됩니다. **Coord[x].CornerError** 도 양의 값인 경우 이 시간으로 코너 오차를 연산하고 이 오차가 파라미터의 제한을 초과하는 경우 코너 오차 제한을 초과하지 않게 블렌드 시간이 축소됩니다(이는 프로그램 속도에서는 파라미터보다 높은 코너 가속도를 생성합니다. 이어지는 록 어헤드 연산은 코너에서의 속도를 떨어뜨려서, 가속도를 제한 이내로 유지하도록 사용할 수 있습니다).

Coord[x].CornerError 는 V1.6 펌웨어의 새로운 기능입니다(2014 년 1/4 분기 출시). 0.0 인 기본값에서는 동작이 해당 파라미터가 없었던 이전 펌웨어 버전과 호환성이 있습니다.

코너 각도는 **normal** 벡터로 지정된 평면으로 계산됩니다. 한 쪽 이동이 **circle** 모드 이동인 경우 블렌드 되지 않은 코너 직후의 이동 방향이 코너 각도를 계산하기 위해 사용됩니다. 코너의 동작이 평면에 수직 성분을 포함하는 경우(예를 들어 지정된 XY 평면에 대한 Z 축 동작), 코너 각도의 계산에 사용되는 것은 지정된 평면상에 투영 궤적입니다. 벡터 피드 레이트가 진입하는 이동과 밖으로 나가는 이동 사이에서 변화하는 경우 두 피드 레이트의 RMS 평균이 계산 중에 사용됩니다.

Coord[x].CornerError 가 블렌드 시간을 제어하기 위해 사용되면 **Coord[x].Ta(ta{데/이/타}** 명령으로 설정할 수 있음)의 값은 사용되지 않습니다. 그렇지 않으면 **Coord[x].Ta** 가 제어합니다. 단, 그 값은 유지됩니다. 따라서 그 값을 재지정하지 않고 적절한 값으로 하며(예를 들어 다음 이동 시퀀스의 최초 가속도 등) 그 이후에도 사용할 수 있습니다.

Coord[x].Ts 대 **Coord[x].Ta** 의 비율이 코너의 형상을 결정하므로 코너 오차 계산 루틴은 코너에서의 블렌드 파라미터를 계산할 때 이 비율을 유지합니다(더 큰 상대적인 **Ts** 의 값에서는 코너 형상은 더 직각이 됩니다). 코너 오차 알고리즘이 동작하기 위해 해당 값 중 적어도 하나는 0.0 보다 커야 합니다.

지정된 코너 오차를 얻기 위해 계산된 블렌드 시간이 **Coord[x].Td** 보다 작은 경우 **Coord[x].Td** 의 값이 대신 사용됩니다(**ta{데/이/타}** 명령이 **Coord[x].Ta** 와 동일하게 **Coord[x].Td** 의 값도 설정한다는 것을 잊지 마십시오). 계산된 블렌드 시간이 매우 크고 진입하는 이동의 "일정 속도" 기간 시작 전에 블렌드를 시작해야 하는 경우 일정 속도 기간이 시작될 때 블렌드가 시작될 수 있도록 블렌드 시간이 축소됩니다.

이어서 모터 가속도 제한이 세그먼트의 록 어헤드 알고리즘에 적용되는 경우 경로와 코너 크기가 변경되지 않는다는 점에 주의하십시오. 경로에 따른 속도는 모터 제한을 준수하기 위해 낮춰질 수도 있습니다.

Coord[x].CornerRadius 좌표계의 블렌드 코너 반경

설명: 블렌드 이동에서의 코너 반경

범위: 음이 아닌 부동 소수점

단위: 축 단위

기본값: 0.0(무효)

양의 값인 **Coord[x].CornerRadius** 는 서로의 동작을 블렌드할 때의 Power PMAC 이 사용하는 코너 "반경"을 지정합니다. 벡터 속도와 코너 각도에 따라서 Power PMAC 은 이 "반경"을 실현하는 블렌드 시간을 계산합니다. 해당 블렌드 시간은 벡터 속도와 함께 코너 "크기"를 결정합니다. 블렌드 된 코너 경로가 진(眞) 원호가 아니라는 점에 주의하십시오. 여기서 지정한 "반경"은 블렌드 된 코너의 시점과 종점에서부터 그때 나오는 궤적에 수직인 선이 교차하는 점까지의 거리입니다.

코너 각도는 **normal** 벡터로 지정된 평면으로 계산됩니다. 한 쪽 이동이 **circle** 모드 이동인 경우 블렌드 되지 않은 코너 직후의 이동 방향이 코너 각도를 계산하기 위해 사용됩니다. 코너의 이동이 평면에 수직 성분을 포함하는 경우(예를 들어 지정된 XY 평면에 대한 Z 축 동작)에 코너 각도의 계산에 사용되는 것은 지정된 평면상에 투영 이동입니다. 벡터 피드 레이트가 진입하는 이동과 밖으로 나가는 이동 사이에서 변화하는 경우 두 피드 레이트의 RMS 평균이 계산 중에 사용됩니다.

Coord[x].CornerRadius 가 블렌드 시간을 제어하기 위해 사용되면 **Coord[x].Ta**($ta_{\{데/이/타\}}$ 명령으로 설정할 수 있음)의 값은 사용되지 않습니다. 그렇지 않으면 **Coord[x].Ta** 가 제어합니다. 또한 S 자 곡선 시간은 **Coord[x].Ts** 의 값($ts_{\{데/이/타\}}$ 명령으로 설정할 수 있음)과 관계없이 블렌드 가속 중에 사용되지 않습니다. 단, 해당 요소의 값은 유지됩니다. 따라서 그 값을 재지정하지 않고 적절한 값으로 하며(예를 들어 다음 이동 시퀀스의 최초 가속도 등) 그 이후에도 사용할 수 있습니다. **Coord[x].CornerRadius** 가 0.0 의 기본값인 경우 그때는 **Coord[x].Ta** 와 **Coord[x].Ts** 의 값이 유효가 되어 블렌드 시간을 지정합니다.

지정된 코너 가속도를 얻기 위해 계산된 블렌드 시간이 **Coord[x].Td** 보다 작은 경우 **Coord[x].Td** 의 값이 대신 사용됩니다(**Coord[x].Td** 가 **Coord[x].Ts** 보다 작은 경우 **Ts** 시간이 유효한 **Td** 시간으로 기능합니다). 계산된 블렌드 시간이 매우 크고 진입하는 이동의 "일정 속도" 기간 시작 전에 블렌드를 시작해야 하는 경우 일정 속도 기간이 시작될 때 블렌드가 시작될 수 있도록 블렌드 시간이 축소됩니다.

Coord[x].Dprog 좌표계의 D 코드 프로그램 번호

설명:	D 코드 서브루틴을 위한 서브 프로그램 번호
범위:	양의 정수
단위:	서브 프로그램 번호
기본값:	1003

Coord[x].Dprog 는 "D 코드"의 실행 시에 어느 서브 프로그램이 호출되는지를 제어합니다. 프로그램 실행이 **D{데이터}** 구문을 검출하면 프로그램의 흐름은 **Coord[x].Dprog** 로 지정된 번호의 서브 프로그램의 **{데이터}**값의 1000 배인 점프 대상 행 레이블로 점프합니다. 예를 들어 **Coord[x].Dprog** 가 1003 인 기본값으로 설정되는 경우 **D43**에서는 **subprog 1003** 의 점프 대상 행 레이블 **N43000** 으로 점프합니다.

이 데이터 구조체 요소는 다른 좌표계가 동일한 D 코드에서 다른 서브루틴을 호출할 수 있게 합니다. 단, 기본설정에서는 모든 좌표계가 **subprog 1003** 을 D 코드 서브루틴에 사용합니다.

Coord[x].FeedHoldSlew 좌표계의 피드 홀드의 슬루 레이트

설명:	피드 홀드의 슬루 레이트
범위:	부동 소수점
단위:	msec/서보 사이클
기본값:	0.00001
기존 I 변수명:	Isx95

Coord[x].FeedHoldSlew 는 피드 홀드할 때나 피드 홀드에서 재개될 때의 좌표계의 타임 베이스 변화율을 제어합니다. 피드 홀드 명령이 좌표계에서 전송되면 실제 타임 베이스는 **Coord[x].FeedHoldSlew** 에서 결정된 비율로 0 을 향해 빠집니다("슬루").

즉 좌표계의 수치적인 시간의 값(msec 단위)을 더하기 위해 사용되는 **Coord[x].TimeBase** 의 값이 서보 사이클마다 0 에 도달할 때까지 **Coord[x].FeedHoldSlew** 의 값으로 빠집니다. 이어지는 재개 명령(예를 들어 **r** 또는 **resume**)이 좌표계에서 전송되는 경우에 타임 베이스값은 이 동일한 비율로 피드 홀드 전의 값으로 증가하여 복귀합니다.

"실시간" 타임 베이스값(%100)은 글로벌 데이터 구조체 요소의 **Sys.ServoPeriod** 로 결정됩니다. 이는 시스템의 클록 주파수를 제어하는 서보 또는 MACRO ASIC 으로 결정되는 서보 클록 사이클의 실제 주기로 설정되어야 합니다(msec 단위). 100% 변화에서의 시간(서보 사이클 단위)은 다음과 같이 계산합니다.

$$100\%SlewTime(servo\_cycles) = \frac{Sys.ServoPeriod}{Coord[x].FeedHoldSlew}$$

마찬가지로 이 변화에서 시간(msec 단위)은 다음과 같이 계산합니다.

$$100\%SlewTime(sec) = \frac{Sys.ServoPeriod^2}{Coord[x].FeedHoldSlew}$$

이 관계를 반전하여 주어진 슬루 타임에 필요한 요소의 값은 다음과 같습니다.

$$Coord[x].FeedHoldSlew = \frac{Sys.ServoPeriod^2}{100\%SlewTime(sec)}$$

보간 계산 중에 사용되는 실제 타임 베이스 변화율의 제어는 타임 베이스를 사용하여 축의 속도에 발생하는 스텝 변화가 아니라 원하는 타임 베이스값으로 스텝 변화를 가능하게 합니다.

Coord[x].FeedHoldSlew 가 양의 값으로 설정되는 경우 좌표계 타임 베이스는 좌표계 모터의 최종적인 가속도와 관계없이 각 서보 사이클에서 이 값과 같은 비율로 변경됩니다.

Coord[x].FeedHoldSlew 가 음수로 설정되는 경우(2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능)에 좌표계 타임 베이스의 변화율이 더욱 제한될 수 있습니다. 이 값의 절대값과 같은 비율로 변경되고 좌표계 중 한 모터 가속도가 그 **Motor[x].InvAmax** 제한을 초과하는 경우 또는 모터 감속도가 그 **Motor[x].InvDmax** 제한을 초과하는 경우에 변화율이 제한될 수 있습니다. 궤적이 원래 100%인 타임 베이스에서 이 제약을 초과하지 않도록 지정된 경우 0 보다 작은 **FeedHoldSlew** 에서는 +/-100% 이내의 변화는 이 제약을 초과하지 않습니다.

Coord[x].FeedHoldSlew 가 0.0 으로 정확하게 설정되는 경우 홀드 명령 또는 재시작 명령에 의한 실제 타임 베이스값은 변경할 수 없습니다.

Coord[x].FeedTime 좌표계 속도의 시간 단위

설명: 속도의 시간 단위

범위: 음이 아닌 부동 소수점

단위: msec

기본값: 1000(초)

기준 I 변수명: Isx90

Coord[x].FeedTime 은 좌표계에서 실행되는 모션 프로그램의 명령 속도(피드 레이트)에 사용되는 시간 단위를 정의합니다. 속도 단위는 시간 단위로 나뉜 길이 또는 각도의 단위로 구성됩니다. 좌표계의 길이/각도 단위는 축 정의 구문 또는 키네마틱스 서브루틴에 추가로 어떤 선택된 변환 매트릭스로 결정됩니다.

Coord[x].FeedTime 에는 시간 단위를 설정합니다. **Coord[x].FeedTime** 은 그 자체가 msec 의 단위입니다. 따라서 60,000 으로 설정되는 경우 시간 단위는 60,000msec 또는 1min 이 됩니다. **Coord[x].FeedTime** 의 기본값은 1sec 의 속도의 시간 단위를 지정하는 1000 입니다.

이 파라미터는 2 타입의 모션 프로그램의 값에 영향을 줍니다. 블렌드 되는 동작(**linear** 와 **circle** 모드)의 벡터 피드 레이트와 **PVT** 모드 동작의 부호가 있는 축 속도입니다.

Coord[x].GoBack 좌표계의 점프 복귀 횟수

설명: 블렌드 무효화 전의 모션 프로그램의 점프 복귀

범위: 0 ..255

단위: 열거 형식

기본값: 0

Coord[x].GoBack 은 좌표계의 모션 프로그램 실행에서 다음 이동에 블렌드를 자동으로 무효화 하기 전에 다음 이동이나 드웰, 딜레이 명령을 찾으면서 몇 회 "점프 백"할 수 있는지를 지정합니다. 블렌드를 무효로 하지 않고 (**GoBack** + 1)회분 점프 백할 수 있습니다.

발견한 모션 명령에서는 세그먼트 동작이 아닌 경우에 궤적을 다음 실시간 인터럽트(**Sys.RtIntPeriod** 로 설정)로 진행 또는 세그먼트 동작의 경우 궤적을 다음 세그먼트 주기(**Coord[x].SegMoveTime** 으로 설정)로 진행하도록 충분한 시간의 모션을 생성해야 합니다. 그에 따라 Power PMAC 은 현재의 일련의 이동 명령을 완전한 것으로 인정할 수 있습니다.

"점프 백"은 **while** 이나 **do..while** 루프의 마지막 또는 프로그램 중의 이전 라인에 **goto** 명령으로 생성됩니다. 서브루틴 또는 서브 프로그램에서의 복귀는 카운트 되지 않습니다.

여러 개의 "점프 백"에서의 자동적인 블렌드의 무효화는 프로그램이 모션 식을 생성하는 다음 명령을 찾으려는 무한 루프에 빠져 있는 동안 블렌드 된 이동이 계속 실행되는 사례를 방지하기 위한 것입니다. 직전의 이동의 실행이 본래 새로운 이동과 블렌드 되도록 예정된 위치에 도달하지만 그 새로운 이동의 동작 식이 아직 계산되지 않은 경우 "런타임 에러"가 발생합니다. 그리고 Power PMAC 은 자동으로 "Abort" 명령을 생성하여 프로그램 실행을 중단하고 **Motor[x].AbortTa** 와 **Motor[x].AbortTs** 의 파라미터에 의해 모든 모터를 감속 정지시킵니다. 감속이 반드시 프로그램 궤적에 따라가는 것은 아니므로 보통 프로그램 포인트에서 종료하지 않는다는 점에 주의하십시오.

이 기능이 충분한 시간의 동작 식을 생성하는 다음 명령을 찾는 동안 프로그램의 (**GoBack**+2)회분의 점프 백을 검출하면 마치 이동 후 드웰 명령이 있는 것처럼 프로그램된 종점에서 정지시키도록 최근에 계산된 이동의 종료를 자동으로 재계산합니다. 이동 실행이 이 포인트에서 정지할 때까지 프로그램 실행이 일시 정지되었다가 그 다음에 프로그램 실행은 자동으로 재개됩니다. 모터가 모두 정지한 무한 루프에서는 잠재적인 런타임 에러의 문제는 존재하지 않습니다.



주의

V1.6(2014 년 1/4 분기 출시) 이하의 펌웨어 버전에서는 너무 많은 점프 백에 의해 프로그램 실행이 일시 정지된 경우 재개할 수 없을 수도 있습니다.



저장 설정 요소 **Coord[x].GoBack** 은 모션 프로그램 실행에서의 점프 백을 허가하는 횟수를 제어하며 비저장 설정 요소 **Coord[x].Ldata.GoBack** 에 관계없습니다. **Ldata.GoBack** 은 모션 프로그램에서 호출되는 키네마틱스 서브루틴 중에서의 점프 백을 허가하는 횟수를 제어합니다.

Coord[x].GoBack 은 풀 워드 요소 **Coord[x].Control[0]**의 비트 0-7 을 구성합니다.

Coord[x].Gprog 좌표계의 G 코드 프로그램 번호

설명: G 코드 서브루틴을 위한 서브 프로그램 번호

범위: 양의 정수

단위: 서브 프로그램 번호

기본값: 1000

Coord[x].Gprog 는 "G 코드"의 실행에서 어느 서브 프로그램이 호출되는지를 제어합니다. 프로그램 실행이 **G{데이터}** 구문을 검출하면 프로그램의 흐름은 **Coord[x].Gprog** 로 지정된 번호의 서브 프로그램의 **{데이터}**값의 1000 배인 점프 대상 행 레이블로 점프합니다. 예를 들어 **Coord[x].Gprog** 가 1000 인 기본값으로 설정되는 경우 **G17**에서는 **subprog 1000** 의 점프 대상 행 레이블 **N17000** 으로 점프합니다.

이 데이터 구조체 요소는 다른 좌표계가 동일한 G 코드에서 다른 서브루틴을 호출할 수 있게 합니다. 단, 기본설정에서는 모든 좌표계가 **subprog1000** 을 G 코드 서브루틴에 사용합니다.

Coord[x].HomeRequired 좌표계의 원점 복귀 필요 조건

설명: 모션 프로그램 실행 전에 원점 복귀가 필요한지 여부를 제어

범위: 0 ..1

단위: 논리값

기본값: 0(무효)

Coord[x].HomeRequired 는 좌표계의 위치 결정 축에 할당된 모든 모터가 모션 프로그램 실행 전에 기준 위치를 확립시켜야 하는 또는 확립시키지 않아도 좋은지를 결정합니다.

Coord[x].HomeRequired 가 0 인 기본값인 경우에 좌표계의 하나 이상의 모터가 기준 위치를 확립하지 않아도 모션 프로그램의 실행이 좌표계에서 허가됩니다. 이 설정은 해당 설정 요소가 없었던 이전 버전의 Power PMAC 펌웨어와 하위 호환성이 있습니다.

Coord[x].HomeRequired 가 1 로 설정되는 경우 만일 좌표계에서 위치 결정 축에 할당된 모든 모터가 기준 위치를 확립하고 있지 않으면 모션 프로그램 실행은 좌표계에서 허가되지 않습니다. 이 모터 기준 위치는 원점 서치 이동의 정상 완료, 절대 위치 센서에서의 정상 읽기 완료 또는 "homez" 제로 동작 원점 복귀 완료로 확립됩니다. 어떤 경우에도 모터 Status 비트 **Motor[x].HomeComplete**(전원 투입/리셋 시에는 0)는 1 로 설정됩니다. 이 비트가 좌표계의 위치 결정 축에 할당된 모든 모터에서 1 인 경우 좌표계 Status 비트 **Coord[x].HomeComplete** 가 1 로 설정됩니다. 좌표계의 모션 프로그램 실행을 가능하게 하려면 해당 모드에서는 비트가 1 이어야 합니다.

좌표계에서 제로 정의된(#x- >0) 모터와 스핀들 축(#x->S, S0, S1)에 할당된 모터는 프로그램 실행을 실시하기 위해 기준 위치를 확립할 필요가 없습니다(단, 스핀들 축에 할당된 모터는 가끔씩 위치 결정 축에 동적으로 변경됩니다. 따라서 그 경우에 모션 프로그램 실행 전에 보통 기준 위치를 확립해야 한다는 점에 주의하십시오). 모터가 인버스 키네마틱스 축에 할당되어 위치 결정 축이라고 간주되면 이 모드에서는 모션 프로그램 실행을 가능하게 하기 위해 기준 위치를 확립해야 합니다.

Coord[x].HomeRequired 가 1 인 경우 Status 비트 **Coord[x].HomeComplete** 가 0 인 경우는 모션 프로그램 실행을 시작하는 명령(예를 들어 **r, s, start**)은 에러로 거절됩니다.

Coord[x].HomeRequired 는 V1.5 펌웨어에 추가되었습니다(2012 년 3/4 분기 출시). 0 인 기본값은 이전 펌웨어 버전과 호환성이 있습니다. 이것은 프로그램을 실행하기 전에 모든 축에서 기준 위치를 확립시킬 필요는 없었습니다.

Coord[x].InPosTimeout 좌표계의 인포지션 타임아웃

설명: 블렌드 무효로 하는 인포지션의 타임아웃값

범위: 음이 아닌 정수

단위: 실시간 인터럽트 주기

기본값: 0(무효)

기준 I 변수명: Isx81

양의 값인 **Coord[x].InPosTimeout** 에서는 프로그램 이동 간에 블렌드가 무효가 되는 경우에 좌표계의 모든 축에 대해 모션 프로그램에서 다음 이동을 시작하기 전에 Power PMAC 이 "인포지션" 확인을 지정합니다. 이 경우 **Coord[x].InPosTimeout** 의 값에는 인포지션 확인을 위한 "타임아웃" 값을 지정합니다. 그것은 실시간 인터럽트 주기 수로 표시합니다(**Sys.RtIntPeriod**+1 서보 사이클). 좌표계의 모든 축이 진입하는 이동 명령의 종점에서 이 시간 이내에 인포지션 되지 않는 경우에 프로그램은 런타임 에러로 정지합니다.

Coord[x].InPosTimeout 이 0 이고 블렌드가 무효인 경우 Power PMAC 은 다음 이동을 시작하기 전에 명령궤적을 순간 정지시킵니다. 이는 어떤 축도 실제 위치가 종점에 도달했다는 것을 의미하는 것은 아닙니다.

Coord[x].InPosTimeout 으로 지정되는 인포지션 확인은 절대 블렌드 되지 않은 프로그램 이동(**RAPID** 모드 이동, 프로그램 원점 서치 이동) 또는 본래는 블렌드 유효 동작이지만 **Coord[x].NoBlend** 가 1로 설정되어 블렌드 무효가 되었거나 또는 코너가 **Coord[x].CornerBlendBP** 로 지정된 각도보다 예각이므로 블렌드 무효가 된 동작 뒤에 실행됩니다.

Coord[x].InPosTimeout 에서 유효가 된 인포지션 확인은 각 모터의 **Motor[x].InPosBand** 최소치와 **Motor[x].InPosTime** 스캔 횟수를 사용합니다. 좌표계의 모든 모터가 "인포지션"에 있다고 확인되면 좌표계의 "인포지션" Status 비트가 설정됩니다. 인포지션 확인 후 다음 프로그램 이동의 실행이 시작되기 전에 **Coord[x].AddedDwellTime** 으로 지정된 시간의 드웰이 삽입됩니다.

Coord[x].LHDistance 좌표계의 룩 어헤드 거리

설명: 룩 어헤드 거리

범위: 음이 아닌 정수

단위: 이동 세그먼트

기본값: 0

기존 I 변수명: Isx20

Coord[x].LHDistance 는 좌표계 x의 룩 어헤드 버퍼 기능을 유효로 하고 유효인 경우 버퍼가 어느 정도 앞까지 예측할 수 있는지 결정합니다.

Coord[x].LHDistance 가 0(기본값)인 경우 룩 어헤드 버퍼가 정의된다고 해도 룩 어헤드 기능은 사용되지 않습니다.

Coord[x].LHDistance 가 0보다 큰 값으로 설정되고 좌표계가 세그먼테이션 모드(**Coord[x].SegMoveTime>0**)에서 룩 어헤드 버퍼가 정의된 경우에 PMAC 은 **linear** 와 **circle** 모드 이동에서 **Coord[x].LHDistance** 의 세그먼트 만큼을 예측합니다. 룩 어헤드 알고리즘은 **Motor[x].MaxSpeed** 제한 하에서 속도와 **Motor[x].InvAmax** 제한 하에서 가속도를 유지하는 데 필요한 버퍼의 각 세그먼트 시간을 연장할 수 있습니다.

적절한 룩 어헤드를 제어 하기 위해 Power PMAC 이 심도있게 예측할 수 있도록 **Coord[x].LHDistance** 는 충분히 큰 값으로 설정되어야 합니다. 그것은 가속도 제한 내에서 최대 속도부터 정지까지를 완전하게 제어하기 위해서입니다. 모터의 필요한 정지 시간은 다음과 같이 나타낼 수 있습니다.

$$StopTime(msec) = V_{max} * \frac{1}{A_{max}} = Motor[x].MaxSpeed * Motor[x].InvAmax$$

좌표계의 모든 모터에 대해 어느 모터가 가장 긴 정지 시간을 가지고 있는지 판단해야 합니다. 이 모터의 정지 시간은 **Coord[x].LHDistance** 를 계산하기 위해 사용됩니다.

이 정지 시간 간의 평균 속도는 $V_{\max}/2$ 입니다. 따라서 이동이 $V_{\max}$ (최악의 경우)일 때 록 어헤드 알고리즘에 들어가고 예측에 필요한 시간은 $\text{StopTime}/2$ 입니다. 따라서 록 어헤드 버퍼에서 항상 정정되는 세그먼트의 필요 수는 다음과 같이 나타낼 수 있습니다.

$$\text{SegmentsAhead} = \frac{\text{StopTime}(\text{msec})/2}{\text{SegTime}(\text{msec}/\text{seg})} = \frac{\text{Motor}[x].\text{MaxSpeed} * \text{Motor}[x].\text{InvAmax}}{2 * \text{Coord}[x].\text{SegMoveTime}}$$

각각의 세그먼트가 가해질 때마다 Power PMAC 은 완전하게 록 어헤드 버퍼를 정정하는 것은 아니므로 **Coord[x].LHDistance** 로 지정된 록 어헤드 거리는 이보다 약간 커야 합니다. 정지 거리에서의 충분한 록 어헤드를 보증하는 **Coord[x].LHDistance** 의 최소값의 식은 다음과 같습니다.

$$\text{Coord}[x].\text{LHDistance} = \frac{4}{3} * \text{SegmentsAhead}$$

소수값이 발생하는 경우는 다음 정수로 올림 처리하십시오. 이 수치보다 작은 **Coord[x].LHDistance** 의 값이 속도 또는 가속도의 제한 위반을 일으키는 것은 아닙니다. 단, 아무리 프로그램 되었다고 해도 알고리즘은 최고 속도에 도달하는 것을 허가하지 않습니다.

Coord[x].LHDistance 는 **define lookahead** 명령에서 예약된 세그먼트의 수보다 큰 값으로 설정해야 하는 것은 아닙니다. 이는 상태 요소 **Coord[x].LHSize** 에 저장됩니다. 록 어헤드 알고리즘은 버퍼 영역이 부족하면 Power PMAC 은 사용할 수 있는 영역의 양을 반영하기 위해 사용되는 록 어헤드 거리를 자동으로 줄입니다.

프로그램 실행 중에 **Coord[x].LHDistance** 의 값을 동적으로 변경할 수 있습니다. 이는 현재의 명령 피드 레이트와 피드 레이트 오버라이드값에 기초한 정지 거리에 상당하는 세그먼트의 최소 필요 수를 설정하여 록 어헤드 거리를 최적화하기 위해 사용할 수 있습니다. 동적으로 이 값을 설정하면 새로운 오버라이드값 등의 변경에 대해 시스템의 응답성을 최대한으로 높일 수 있습니다.

예

시스템에서의 축이 24,000mm/min 또는 400mm/sec(900 in/min 또는 15 in/sec)의 최대 속도입니다. 0.05g 또는 500mm/sec²(20 in/sec²)의 최대 가속도와 1μm 의 카운트 분해능입니다. 모터 단위는 카운트입니다. 200 블록/sec 의 최대 블록 레이트는 필요합니다. 따라서 **Motor[x].SegMoveTime** 은 5msec 로 설정됩니다. 파라미터는 다음과 같이 계산합니다.

- **Motor[x].MaxSpeed** = 400 mm/sec * 0.001 sec/msec * 1000 cts/mm
= 400 cts/msec
- **Motor[x].InvAmax** = 0.002 sec²/mm * 1000² msec²/sec² * 0.001 mm/ct
= 2.0 msec²/ct
- **Coord[x].LHDistance** = [4/3] * [(400 cts/msec * 2.0 msec²/ct) / (2 * 5 msec/seg)]
= 107 seg

Coord[x].MaxCirAccel 좌표계의 최대 구심가속도

설명: 원호 동작의 최대 구심가속도

범위: 음이 아닌 부동 소수점

단위: 축 단위/(좌표계 시간 단위)²

기본값: 0.0(무효)

기준 I 변수명: Isx78

양의 값인 **Coord[x].MaxCirAccel**에서는 좌표계의 피드 레이트(F)로 지정된 **circle** 이동에서 Power PMAC에서 허가되는 최대 구심가속도를 지정합니다. 프로그램 이동이 더 큰 구심가속도를 일으킨다면 제한을 초과하지 않도록 Power PMAC은 자동으로 이동의 프로그램 속도를 낮춥니다. 구심가속도는 다음과 같이 표시됩니다.

$$A = \frac{V^2}{R}$$

이 제한은 최초의 이동 연산 시 수행됩니다. 따라서 **Coord[x].MaxCirAccel**과 연동하여 룩 어헤드 버퍼를 사용할 필요는 없습니다. 특히 감속 원호 이동에 출입하는 접선가속도를 취급하기 위해 룩 어헤드 버퍼를 사용하는 것이 더 바람직합니다.

구심가속도 **Coord[x].MaxCirAccel**에서의 제한과 개별 모터 가속도 제한 **Motor[x].InvAmax**에서 구심가속도를 제한하는 것의 가장 중요한 차이는 정확한 궤적을 생성하는지 여부입니다. Power PMAC의 원호 보간 알고리즘은 다음 식으로 설명할 수 있는 반경 방향의 오차를 생성합니다.

$$E = \frac{V^2 T^2}{6R}$$

V가 모션 프로그램으로 결정되는 원호에 따른 (보통 **Coord[x].MaxCirAccel**로 제한된) 속도에서 T는 **Coord[x].SegMoveTime**의 세그멘테이션 시간입니다. 그리고 R은 원호의 반경입니다. **Coord[x].MaxCirAccel**이 원호의 속도를 제한하기 위해 사용된다면 이 오차는 작아집니다. 단, 룩 어헤드 기능이 속도 제한에 사용되는 경우 오차는 마치 원호 이동이 전속력으로 실행되는 것처럼 커집니다.

Coord[x].MaxCirAccel은 좌표계의 **Coord[x].FeedTime**에서 설정되는(보통 sec 또는 min) 사용자 "피드 레이트 시간 단위"의 제공으로 나눈 직선 축의 사용자 길이 단위(보통 mm 또는 inch)로 표시됩니다.

예 1:

Coord[x].MaxCirAccel 에서 구심가속도를 1.0g 으로 제한하고자 하는 경우를 상정합니다. 길이 단위는 mm 입니다. 그리고 시간 단위는 초입니다. **Coord[x].MaxCirAccel** 은 다음과 같이 계산합니다.

$$MaxCirAccel = 1.0g * \frac{9.8 \frac{m}{s^2}}{g} * \frac{1000mm}{m} = 9800 \left(\frac{mm}{s^2} \right)$$

예 2:

원호 보간 계산 오차를 0.001 인치로 제한하고자 하는 경우를 상정합니다. 길이 단위는 인치입니다. 시간 단위는 분입니다. 그리고 **Coord[x].SegMoveTime** 의 세그먼트이션 시간은 10msec 입니다. **Coord[x].MaxCirAccel** 은 다음과 같이 계산합니다.

$$MaxCirAccel = \frac{V^2}{R} = \frac{6E}{T^2} = \frac{6 * 0.001in}{0.01^2 s^2} * \frac{60^2 s^2}{min^2} = 216000 \left(\frac{in}{min^2} \right)$$

예 3:

시스템이 가속도 10m/s²(약 1g)에 대응할 수 있는 경우를 상정합니다. 길이 단위는 mm 입니다. 시간 단위는 분입니다. 그리고 **Coord[x].SegMoveTime** 의 세그먼트이션 시간은 2msec 입니다. **Coord[x].MaxCirAccel** 은 다음과 같이 계산합니다.

$$MaxCirAccel = 10 \frac{m}{s^2} * \frac{1000mm}{m} * \frac{3600s^2}{min^2} = 36,000,000 \left(\frac{mm}{min^2} \right)$$

해당 설정에서는 최대 원호 보간 계산 오차는 다음과 같이 계산합니다.

$$E = \frac{V^2 T^2}{6R} = MaxCirAccel * \frac{T^2}{6} = \frac{36,000,000}{6} \frac{mm}{min^2} * 0.002^2 s^2 * \frac{min^2}{60^2 s^2} = 0.0067mm$$

Coord[x].MaxFeedrate 좌표계의 최대 피드 레이트

설명: 최대 허용 벡터 피드 레이트

범위: 음이 아닌 부동 소수점

단위: 축 단위/좌표계 시간 단위

기본값: 0.0(비활성)

기준 I 변수명: Isx98

Coord[x].MaxFeedrate 는 좌표계에 지정할 수 있는 최대 벡터 속도("피드 레이트")를 지정하여 프로그램이 실수로 초과한 값을 명령하는 것을 방지합니다.

0 미만의 벡터 속도를 지정할 때의 **Coord[x].Tm**의 크기가 이 파라미터보다 큰 경우 이 파라미터가 대신 사용됩니다. 프로그램 명령 **F{데이터}**를 사용하여 **Coord[x].Tm**이 설정되는지 또는 대입 명령으로 직접 설정되는지에 관계없이 이와 같이 처리됩니다. **Coord[x].Tm** 요소의 값은 이 제한에 의해 변경되지 않습니다.

제한에 대한 명령 피드 레이트의 확인은 모터 연산을 수행하기 전에 프로그램 이동의 블록 연산 시 수행됩니다. 모터 계산에서는 룩 어헤드 기능의 유무에 관계없이 **Motor[x].MaxSpeed**의 제한과 각각의 모터 속도를 비교합니다. 그때 벡터 속도는 확인하지 않습니다. 직교 좌표계에서는 모터 제한이 더 빠른 **RAPID** 모드 이동 속도를 허용하도록 **linear**와 **circle** 모드에서의 프로그램 이동의 중요한 속도 제한은 보통 **Coord[x].MaxFeedrate**가 됩니다.

Coord[x].MaxFeedrate는 좌표계의 **Coord[x].FeedTime**(보통 sec 또는 min)으로 설정된 사용자 "피드 레이트 시간 단위"로 나누는 피드 레이트 축의 사용자 길이 단위(보통 mm 또는 inch)로 표시됩니다.

Coord[x].MaxFeedrate가 0.0인 기본값인 경우 Power PMAC은 프로그램 피드 레이트값의 제한을 확인하지 않습니다.

Coord[x].MinArcLen 좌표계의 최소 원호 길이

설명: circle 모드의 최소 원호 길이

범위: 음이 아닌 부동 소수점

단위: 라디안

기본값: 0.0

기존 I 변수명: Isx97

Coord[x].MinArcLen은 Power PMAC의 **circle** 모드가 생성할 수 있는 최단의 원호에 대한 각도의 크기를 지정합니다. **Coord[x].MinArcLen**보다 작은 각도로 중심의 IJK 벡터 표현으로 프로그램된 **circle** 모드 이동은 프로그램 각도 변화와 하나의 완전한 원의 합으로 실행됩니다. **Coord[x].MinArcLen**보다 큰 각도의 이동은 완전한 원보다 작은 하나의 원호로 실행됩니다.

Coord[x].MinArcLen의 목적은 단순히 종점을 시점(0도의 원호)과 같게 하여 하나의 완전한 원의 이동을 명령할 수 있는 표준 원호 프로그래밍을 지원하는 것입니다. 그러나 반올림 오차는 있습니다.

Coord[x].MinArcLen이 0.0인 기본값의 경우 완전한 원의 이동이 실행되기 위해서는 전체 배정도 부동 소수점 분해능으로 원호 보간축의 종점과 시작점이 정확하게 같아져야 한다는 점에 주의하십시오.

Coord[x].Mprog 좌표계의 M 코드 프로그램 번호

설명:	M 코드 서브루틴의 서브 프로그램 번호
범위:	양의 정수
단위:	서브 프로그램 번호
기본값:	1001

Coord[x].Mprog 는 "M 코드"의 실행에서 어느 서브 프로그램이 호출될지를 제어합니다. 프로그램 실행이 **M{데이터}** 구문을 검출하면 프로그램의 흐름은 **Coord[x].Mprog** 로 지정된 번호의 서브 프로그램의 **{데이터}**값의 1000 배인 점프 대상 행 레이블로 점프합니다. 예를 들어 **Coord[x].Mprog** 가 1001 인 기본값으로 설정되는 경우 **M24**에서는 **subprog 1001**의 점프 대상 행 레이블 **N24000**으로 점프합니다.

이 데이터 구조체 요소는 다른 좌표계가 동일한 M 코드에서 다른 서브루틴을 호출할 수 있게 합니다. 단, 기본설정에서는 모든 좌표계가 **subprog 1001**을 M 코드 서브루틴에 사용합니다.

Coord[x].Ndisplay 좌표계의 동기 행 레이블 표시 제어

설명:	동기 행 레이블 표시 제어
범위:	0 ..1
단위:	논리값
기본값:	0

Coord[x].Ndisplay 는 축 목표 위치가 **&xt** 명령 또는 축의 남은 이동량이 **&xg** 명령으로 쿼리가 오면 상태 요소 **Coord[x].Nsync**의 현재값이 응답의 처음에 통지되는지 여부를 제어합니다. **Coord[x].Nsync**의 값은 모션 프로그램의 다음 이동 실행 시작에서 동기 행 레이블 **N{데이터}**의 값으로 설정됩니다. 따라서 어느 이동이 현재 실행되고 있는지를 표시하기 위해 사용할 수 있습니다.

Coord[x].Ndisplay 가 0 인 기본값의 경우 **Coord[x].Nsync**의 값은 응답의 처음에 통지되지 않습니다. 이 경우 축의 값만 통지됩니다. 그리고 응답의 예로는 X10 Y20 Z30 이 통지됩니다.

Coord[x].Ndisplay 가 1 로 설정되는 경우 **Coord[x].Nsync**의 값은 응답의 처음에 통지됩니다. 그리고 응답의 예로는 N560 X10 Y20 Z30 이 통지됩니다.

Coord[x].Ndisplay 는 풀 워드 요소 **Coord[x].Control[0]**의 비트 17 을 구성합니다.

Coord[x].NoBlend 좌표계의 블렌드 무효

설명: 이동 블렌드의 무효 제어

범위: 0 ..1

단위: 없음

기본값: 0

기준 I 변수명: Isx92

Coord[x].NoBlend 는 좌표계의 프로그램 된 이동이 자동으로 블렌드 되는지 여부를 제어합니다. 0 인 기본값의 경우 **linear**, **circle** 그리고 **spline** 모드의 연속하는 프로그램 이동은 정지의 개입 없이 블렌드 됩니다. 다음 이동은 실행 중인 이동 도중에 계산됩니다. 단, 1 로 설정되는 경우 각 프로그램 이동 사이에 짧은 정지(사실상 **dwel 0** 명령 추가)가 들어가서 다음 이동은 그 사이에 계산됩니다.

실행을 시작하는 명령이 전송되는 경우에만 이 파라미터가 사용됩니다.

Coord[x].NoBlend 는 풀 워드 요소 **Coord[x].Control[0]**의 비트 30 을 구성합니다.

Coord[x].NoCornerBp 좌표계의 코너 브레이크 포인트 무효

설명: 블렌드와 드웰 계산을 위한 의사 코너 각도

범위: -1.0 ..1.0(부동 소수점)

단위: 유효한 각도 코사인

기본값: 0.0

Coord[x].NoCornerBp에서는 진입하여 밖으로 나가는 이동 중 하나가 계산 평면의 이동 성분을 갖지 않거나(거리 0 의 이동 또는 평면에 수직인 이동) 또는 존재하지 않는(최초 이동의 시작이나 일련의 이동의 마지막 이동의 종료) 경우 좌표계의 코너 계산에서 사용되는 실효 각도를 지정합니다. 그것은 **normal** 벡터로 정의된 평면에서 "코너" 동작의 유향각 변화의 코사인으로 표시됩니다. 1.0 의 값은 각도 변화 없음에 상당합니다. 0.0 의 값은 90° 각도에 상당합니다. -1.0 의 값은 180° 역회전에 상당합니다. 이 파라미터는 사용자가 해당 이동 경계에서의 동작을 지정할 수 있게 합니다.

이러한 "코너"에서는 **Coord[x].NoCornerBp** 의 값은 이동이 블렌드 되는지 여부를 결정하기 위해 **Coord[x].CornerBlendBp**(**CornerBlendBp** 가 0.0 이 아닌 설정에서 유효한 경우)의 값과 비교됩니다. **NoCornerBp** 가 **CornerBlendBp** 보다 작은 경우 Power PMAC 은 이동 경계에서의 블렌드를 자동으로 무효로 하고 진입하는 이동의 마지막에 명령 궤적을 정지시킵니다. **Coord[x].InPosTimeout** 이 0 보다 큰 경우 Power PMAC 은 좌표계의 모든 모터가 먼저 진행하기 전에 "인포지션"에 있는지 확인합니다. 또한 **NoCornerBp** 가 **Coord[x].CornerDwellBp** 보다 작은 경우(**CornerDwellBp** 가 0.0 이 아닌 설정에서 유효한 경우)에 실시간 인터럽트 주기에 의한 **Coord[x].AddedDwellTime** 의 드웰이 실행됩니다. 마지막으로 밖으로 나가는 이동이 자동으로 시작됩니다.

NoCornerBp 가 **CornerBlendBp** 이상인 경우 Power PMAC 은 보통 블렌드 알고리즘으로 진입하여 밖으로 나가는 이동을 직접 블렌드합니다.

Coord[x].pDesTimeBase 좌표계의 타임 베이스 소스 포인터

설명: 타임 베이스의 소스 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: **Coord[x].DesTimeBase.a**

기존 I 변수명: Isx93

Coord[x].pDesTimeBase에서는 좌표계가 궤적 보간 연산에서 필요한 타임 베이스값에 어느 레지스터를 사용할지 지정합니다. 여기에는 레지스터의 주소를 설정합니다. 이 레지스터의 내용은 msec 단위로 부동 소수점(배정도)이어야 합니다.

Coord[x].pDesTimeBase 가 기본값(동일한 좌표계의 **Coord[x].DesTimeBase** 의 주소)으로 설정되면 %{상수} 온라인 명령에 자동으로 응답하는 레지스터가 사용됩니다. 타임 베이스를 그대로 두거나 프로그램 제어 아래 두는 경우 **Coord[x].pDesTimeBase** 는 기본값 그대로 두어야 합니다.

좌표계의 타임 베이스값이 입력되는 엔코더 신호 또는 펄스 열의 주파수에 비례하는 "외부 타임 베이스"를 사용하기 위해서는 **Coord[x].pDesTimeBase** 가 **EncTable[j].DeltaPos.a** 로 설정되어야 합니다. 그때 엔코더 테이블의 "j번째" 항목에서 엔코더 또는 펄스 열이 처리되고 **DeltaPos** 는 서보 사이클에서의 카운트값의 변화를 스케일링한 값이 됩니다.

Coord[x].PosRollOver[j] 좌표계의 축 위치 롤 오버 범위

설명: 축 위치 롤 오버 범위

범위: 부동 소수점

단위: 축 단위

기본값: 0.0

Coord[x].PosRollOver[j]는 좌표계에서의 6 개 값의 배열(인덱스 0~5)이며 그 값은 6 개의 회전축(A, B, C, AA, BB 및 CC 에 인덱스 0~5 가 각각 대응)이 "롤 오버" 모드에서 프로그램되는지 여부를 지정합니다. 그리고 롤 오버가 지정되는 경우 해당 롤 오버 범위를 지정합니다. **Coord[x].PosRollOver[j]**가 0 인 기본값의 경우 롤 오버 모드는 무효가 되고 해당 축은 직선 축과 동일하게 처리됩니다.

각 배열의 요소에서 영향을 받는 축을 나타낸 표입니다.

요소	축	요소	축
Coord[x].PosRollover[0]	A	Coord[x].PosRollover[3]	AA
Coord[x].PosRollover[1]	B	Coord[x].PosRollover[4]	BB
Coord[x].PosRollover[2]	C	Coord[x].PosRollover[5]	CC

Coord[x].PosRollOver[i]가 0 보다 큰 경우 절대 위치(abs) 모드에서 프로그램된 축 이동에 대해 축은 목적 지점에 도착하기 위해 **Coord[x].PosRollOver[i]**로 정의된 원형 범위 주변의 최단경로를 잡습니다. 절대 위치 모드의 어떤 이동도 이 롤 오버가 유효인 경우 활성 롤 오버에서 1 회전의 절반(**Coord[x].PosRollOver[i]/2**)보다 길어지지 않습니다. 해당 타입의 회전축은 보통 도 단위로 프로그램되므로 **Coord[x].PosRollOver[i]**는 이 모드에서는 360 으로 설정됩니다.

Coord[x].PosRollOver[i]가 0 보다 작은 경우 절대 위치(abs) 모드에서 프로그램된 축 이동에 대해 이동 명령의 축값의 부호는 동작 방향으로 사용됩니다. 절대 위치 모드의 어떤 이동도 이 롤 오버가 유효인 경우 1 회전(**Coord[x].PosRollOver[i]**)보다 길어지지 않습니다. 이 타입의 회전축은 도 단위로 보통 프로그램되므로 **Coord[x].PosRollOver[i]**는 이 모드에서는 -360 으로 설정합니다.

또한 이 모드에서는 명령받은 절대 목표 위치의 부호는 목표값의 일부입니다. 따라서 이 모드에서의 **A-90** 명령은 역방향으로 -90 도(= +270 도) 이동 명령입니다. 양의 방향의 이동 명령을 위해 +부호가 필요 없이 허가됩니다(예를 들어 양의 방향으로 90 도 이동 명령은 **A90** 또는 **A+90** 중 하나를 사용할 수 있습니다).

Power PMAC 은 +0.0 과 -0.0 의 목표 명령 간의 차이를 관리할 수 없습니다. 그러므로 종료 위치에 아주 작은 0 이외 크기의 명령을 사용해야 합니다(예를 들어 **A+0.0000001** 과 **A-0.0000001**). 이 증가는 최종 목표값에 어떤 영향도 주지 않는 아주 작은 값이 됩니다. 정확하게 0.0 으로의 A 명령은 1 바퀴의 제로 위치로 최단 방향으로 이동시킵니다.

값의 부호가 아니라 크기만이 목표 위치를 지정하는 동일한 모드인 경우 다른 모드에서 크기를 360 도 뺀 크기가 되도록 역방향 이동의 목표 위치가 수정되어야 합니다. 예를 들어 만일 명령이 **C-120** 이고 역방향으로 (+)120 도 이동을 지정한다면 Power PMAC에서는 **C-240** 으로 수정되어야 합니다. 이는 역방향으로 -240 도(= +120 도)로의 이동을 지정하는 명령이 되기 때문입니다. 정방향에 대한 이동 명령을 수정할 필요는 없습니다.

인크리멘탈(inc) 모드의 축 이동은 어떤 롤 오버 모드에도 영향을 받지 않습니다. 해당 축에 할당된 모터의 조그 이동은 롤 오버에서 영향을 받지 않습니다. 통지되는 모터와 축 위치는 롤 오버에서 영향을 받지 않습니다(1 회전 내의 "롤 오버"된 위치 정보를 얻기 위해서는 Power PMAC 또는 호스트 컴퓨터 중 하나에서 모듈로(나머지) 연산자를 사용합니다).

Coord[x].RadiusErrorLimit 좌표계의 원호 반경 오차 한계

설명: circle 모드의 반경 오차 한계

범위: 음이 아닌 부동 소수점

단위: 축 단위

기본값: 0.0(무효)

기준 I 변수명: Isx96

Coord[x].RadiusErrorLimit에서는 실행할 수 있는 유효한 이동을 구성하는 **circle** 모드 이동의 시점 반경과 종점 반경 사이의 차의 최대 크기를 지정합니다. 차가 이 크기보다 큰 경우 모션 프로그램은 Abort 되고 실행은 이 이동이 시작할 때 정지합니다. 이 경우 상태 요소 **Coord[x].RadiusError**(이는 **Coord[x].ErrorStatus** 와 **Coord[x].Status[0]**의 일부에서 **&x?** 명령으로 조회 가능)의 비트가 설정됩니다. **Coord[x].RadiusErrorLimit**가 0.0 인 기본값의 경우 이 확인은 무효가 됩니다.

시점에서 중심점의 벡터 거리 지정(I/J/K 또는 II/JJ/KK)이 있는 **circle** 모드 이동에서는 중심에서 동일한 거리가 되도록 명령을 받는 종점을 제한하는 것은 아무것도 없습니다. 거리가 다르다면 Power PMAC 은 진(眞) 원호 대신 시작점에서 종점으로 지수 나선 이동을 실행합니다.

반경(R) 지정의 **circle** 모드 이동이며 시점부터 종점까지의 거리가 반경 크기의 2 배 이상인 경우 원호는 가능하지 않습니다. 그리고 Power PMAC 은 종점에 대한 지수 나선 이동을 생성합니다.

때로는 고의로 수행됩니다. 보통은 프로그래밍 에러가 원인입니다. 대부분의 경우 수치의 반올림에 의한 작은 오차가 있으므로 일부 어플리케이션에서는 반올림에 의한 오차와 실제 프로그래밍 에러를 구별하는 최소치의 설정이 필요합니다. **Coord[x].RadiusErrorLimit** 는 이 최소치에 사용됩니다.

Coord[x].RapidVelCtrl 좌표계의 Rapid 속도 제어

설명: Rapid 이동의 속도 모드 제어

범위: 0 ..1

단위: 없음

기본값: 0

기준 I 변수명: Isx79

Coord[x].RapidVelCtrl 은 Power PMAC 이 다축 RAPID 모드 이동의 "더 짧은" 축의 속도를 어떻게 계산할지 결정합니다. 그것이 0 인 기본값의 경우 다축 이동에 관여하는 모터에 모두 지정된 속도(**Motor[x].RapidSpeedSel** = 1 인 경우 **Motor[x].MaxSpeed**, **Motor[x].RapidSpeedSel** = 0 인 경우 **Motor[x].JogSpeed**)로 이동을 명령합니다. 이는 더 낮은 거리/속도 비율의 모터가 더 높은 비율의 모터보다 가깝게 이동 완료한다는 것을 의미합니다. 그리고 직교 좌표에서의 궤적은 보통 직선이 되지 않습니다.

Coord[x].RapidVelCtrl 이 1 로 설정되는 경우 Power PMAC 은 이동에서의 각 모터의 rapid 속도에 대한 이동 거리의 비율을 계산합니다. 가장 높은 거리/속도 비율을 가진 모터만 실제로 지정된 속도로 명령을 받습니다. 다른 모터에 명령을 내리는 속도는 속도에 대한 거리가 동일한 비율을 갖도록 낮춰집니다. 그것은 모든 모터를 동일한 이동 시간으로 하기 위한 것입니다(가속도와 감속도가 가해지기 전에). 이는 직교 좌표 시스템에서 이동 궤적을 최대한 직선으로 만듭니다(그리고 가속과 감속 시간이 동일한 경우 직선이 됩니다).

다축 이동에 관계하는 모든 경우에 각 모터는 해당 가속, 감속 프로파일을 계산하기 위해 그 자체의 가속도 파라미터인 **Motor[x].JogTa** 와 **Motor[x].JogTs** 를 사용합니다.

Coord[x].RapidVelCtrl 은 폴 워드 요소 **Coord[x].Control[0]**의 비트 28 을 구성합니다.

Coord[x].SegLinToPvt 좌표계의 Linear 이동 PVT 실행

설명: **Linear** 이동의 PVT 에서의 실행

범위: 0 ..3

단위: 없음

기본값: 0

Coord[x].SegLinToPvt 는 세그먼트이션 모드(**Coord[x].SegMoveTime**>0)에서 명령받은 **linear** 모드 이동이 처음에 자동으로 **PVT** 모드 이동으로 변환되는지 여부를 지정합니다. 0 인 기본값에서는 PVT 로는 변환되지 않으므로 **linear** 모드의 규칙에 따라 실행됩니다.

Coord[x].SegLinToPvt 가 0 보다 큰 값으로 설정되는 경우 **linear** 모드의 세그먼트화된 이동은 실행 전에 내부에서 **pvt** 모드 이동으로 변환됩니다. 각 이동이 명령을 받는 위치는 **linear** 모드에서의 위치와 동일합니다. 각 이동에서의 명령 속도와 시간은 명령 궤적이 프로그램 속도로 명령 위치를 올바르게 매끄럽게 통과하기 위해 계산됩니다.

해당 모드에서는 Power PMAC 은 **F** 또는 **tm** 의 프로그램 명령으로 설정되는 **Coord[x].Tm** 의 값을 사용합니다. 그러나 **Coord[x].Ta**, **Td** 또는 **Ts** 의 값은 사용하지 않습니다. 즉, **ta**, **td** 와 **ts** 의 프로그램 명령으로 설정한 값을 무시합니다.

많은 점이 윤곽 위에 밀접하게 배치되는 경우 이 변환은 우수한 궤적 정밀도를 제공합니다. 첫째, 명령을 받은 궤적은 프로그램된 점을 정확하게 통과합니다. 표준인 **linear** 모드에서는 항상 프로그램된 점의 안쪽을 통과합니다. 둘째, 바람직하지 않은 "평탄"은 생성되지 않고 더 매끄러운 궤적과 더 착실한 축 속도 곡선을 생성합니다.

Coord[x].SegLinToPvt 가 1 로 설정되는 경우 비교적 날카로운 코너에서는 속도는 더 양호한 궤적을 유지하기 위해 약간 증가됩니다. **Coord[x].SegLinToPvt** 가 2 로 설정되는 경우 비교적 날카로운 코너에서는 코너 주변 궤적의 "팽창"을 희생하여 속도가 더 양호하게 유지됩니다. 3 의 값은 추후 사용하기 위해 예약됩니다.

Coord[x].SegLinToPvt 가 풀 워드 요소 **Coord[x].Control[0]**의 비트 22-23 을 구성합니다.

Coord[x].SegMoveTime 좌표계의 세그먼트 시간

설명: 개략적인 보간의 세그먼트 시간

범위: 음이 아닌 부동 소수점

단위: msec

기본값: 0.0

기준 I 변수명: Isx13

Coord[x].SegMoveTime 은 좌표계가 "세그멘테이션 모드"인지 여부를 제어합니다. 그리고 세그멘테이션 모드인 경우 "세그멘테이션 시간"을 msec 단위로 표시합니다.

Coord[x].SegMoveTime 이 0.0 보다 큰 경우 좌표계는 세그멘테이션 모드가 되고 **linear**, **circle** 과 **pvt** 모드의 궤적은 모두 **Coord[x].SegMoveTime** 의 msec 마다 개략적인 보간 알고리즘에서 중간간의 "세그먼트" 포인트를 계산하여 생성합니다. 그리고 서보 사이클마다 3 차 스플라인 알고리즘을 사용하여 미세한 보간을 실행합니다.

이 세그멘테이션 시간보다 짧은 시간의 프로그램 이동("블록")을 실행할 수는 있지만 세그멘테이션 알고리즘에서는 해당 블록을 자동으로 건너뛰고 여러 개의 블록에 걸친 평활화를 효율적으로 실현합니다.

좌표계를 세그멘테이션 모드로 하면 유효가 되는 이 개략적인/정밀한 보간법은 좌표계가 다음 기능의 하나를 사용할 때 필요합니다.

- 원호 보간
- 공구 직경 보정
- 여러 블록의 룩 어헤드(예측)
- 인버스 키네마틱스

이 기능 중 하나라도 좌표계에서 필요 없는 경우 다른 태스크의 연산 시간을 해방하기 위해 **Coord[x].SegMoveTime** 을 0.0 인 기본값 그대로 두는 것을 권장합니다. **Coord[x].SegMoveTime** 이 0.0 인 경우 **circle** 모드 이동은 **linear** 이동으로 실행되고 공구 직경 보정은 실행되지 않습니다. 그리고 인버스 키네마틱스 연산도 이동 종료 위치에서만 실행됩니다.

세그먼테이션 모드인 **Coord[x].SegMoveTime** 의 표준값은 2~20msec 입니다. 값이 작을수록 더욱 실제 곡선에 정확하게 일치합니다. 단, 더 많은 계산이 이동에 필요해서 백그라운드 태스크에서 사용할 수 있는 시간은 더 적어집니다. **Coord[x].SegMoveTime** 이 극단적으로 낮게 설정되는 경우 Power PMAC 은 할당된 시간 내에 이동 연산 전체를 수행할 수 없게 되어 런타임 에러로 모션 프로그램을 정지시킵니다.

세그먼테이션 모드에서의 곡선 궤적 상에 도입되는 보간 오차의 식은 다음과 같습니다.

$$E = \frac{V^2 T^2}{6R}$$

V 는 속도, T 는 세그먼테이션 시간, R 는 곡소 반경으로 모두 일관된 단위로 표시됩니다. 직선 궤적에서는 R 이 무한해지고 오차는 0 과 같아집니다. 속도가 피드 레이트 F 로 표시되는 경우 단위/분이며 식은 다음과 같습니다.

$$E = \frac{F^2 \left(\frac{\text{units}^2}{\text{min}^2} \right) * \left(\frac{\text{min}^2}{60,000^2 \text{msec}^2} \right) * (\text{SegMoveTime})^2 (\text{msec}^2)}{6R(\text{units})} = \frac{F^2 (\text{SegMoveTime})^2}{2.16 \times 10^{10} R}$$

예

5000mm/min(200inch/min)의 피드 레이트와 반경 50mm(2 인치)에서는 10msec 의 **Coord[x].SegMoveTime** 의 값은 2.3μm(0.00009inch)의 보간 오차를 생성합니다.

Coord[x].SegOverrideSlew 좌표계의 세그먼트 오버라이드 슬루 레이트

설명: 세그먼테이션 오버라이드 변경의 슬루 레이트

범위: 음이 아닌 부동 소수점

단위: 세그먼트당 정규화된 오버라이드값

기본값: 0.0

기존 I 변수명: Isx16

Coord[x].SegOverrideSlew 는 좌표계의 "세그먼테이션 오버라이드"값의 변화율을 제어합니다. 세그먼트 오버라이드 기능은 동시에 가속도값을 오버라이드시키지 않고 세그먼트 이동(**Coord[x].SegMoveTime**>0 에서의 **linear**, **circle** 과 **pvt** 모드)의 고도 "피드 레이트 오버라이드" 기능을 가능하게 합니다.

이 파라미터는 세그먼트마다 사용되는 내부 오버라이드값이 **Coord[x].SegOverride**의 사용자 명령값에 어느 정도 빠르게 접근할지를 결정합니다(2가지가 다른 경우). 내부와 명령값의 양쪽이 정규화되므로 1.0의 값은 "실시간"으로 제공합니다. 세그먼트마다 "개략적인 보간" 기능은 **Coord[x].SegMoveTime**과 내부 오버라이드값의 곱이며 세그먼트 이동에 따라 시간이 증가하여 그 "시간"에 새로운 위치를 계산합니다. 예를 들어 **Coord[x].SegMoveTime**은 10msec로 설정되었으므로 내부 오버라이드가 0.6인 경우 수학적 시간 증가는 $10 * 0.6 = 6\text{msec}$ 가 됩니다. 이 이동 세그먼트는 물리적으로 10msec로 실행되므로 해당 이동은 프로그램 속도의 60%로 실행합니다.

Coord[x].ActSegOverride의 내부 오버라이드값이 명령받은 **Coord[x].SegOverride**값과 일치하지 않는 경우 각 세그먼트에서 **Coord[x].SegOverrideSlew**의 값은 명령받은 값에 도달할 때까지 내부의 값에 더해지거나 빼집니다.

예

Coord[x].SegMoveTime의 시간은 5msec로 하고 1.0sec에서 100%의 오버라이드 변경을 하기 위해서는 200의 세그먼트에 걸쳐 내부 오버라이드값이 1.0으로 변화하도록 합니다. 따라서 **Coord[x].SegOverrideSlew**는 1/200 또는 0.005로 설정되어야 합니다.

Coord[x].SoftLimitStopDis 좌표계의 소프트웨어 리미트 동작 제어

설명: 소프트웨어 리미트의 동작 모드 제어

범위: 0 ..1

단위: 없음

기본값: 0

Coord[x].SoftLimitStopDis는 좌표계의 모터가 해당 소프트웨어 오버트래블 리미트(**Motor[x].MinPos** 또는 **Motor[x].MaxPos**) 중 하나에 도달하면 어떤 동작을 취할지 결정합니다. 0인 기본값인 경우에 모션 프로그램은 정지하고 좌표계의 모든 모터는 "Abort" 상태가 됩니다. 즉 파라미터인 **Motor[x].AbortTa**와 **Motor[x].AbortTs**에 따라서 클로즈드 루프 정지 상태로 감속 정지시킬 수 있습니다.

단, **Coord[x].SoftLimitStopDis**가 1로 설정된 경우는 해당 소프트웨어 리미트에 도달한 모터가 그 위치를 유지한 상태로 모션 프로그램을 속행합니다.

Coord[x].SoftLimitStopDis는 풀 워드 요소 **Coord[x].Control[0]**의 비트 29를 구성합니다.

Coord[x].SplineTimeRotate 좌표계의 스플라인 동작 시간 제어

설명: 스플라인 모드의 동작 시간 테이블 제어

범위: 0 ..1

단위: 논리값

기본값: 0

Coord[x].SplineTimeRotate 는 스플라인 모드에서의 동작 시간 테이블이 어떻게 동작할지 지정하고 이동 시간을 바꾸는 스플라인 궤적을 어떻게 실행할지 제어합니다. 0 인 기본값의 경우 테이블은 스플라인 궤적의 지정에 더 많은 유연성을 제공합니다. 사용자 프로그램에서 완전하게 "수동" 제어되고 V1.6(2014 년 1/4 분기 출시) 이하의 펌웨어 버전에서는 해당 요소는 존재하지 않으며 이것은 동작 가능한 유일한 값이 됩니다.

Coord[x].SplineTimeRotate 가 1 로 설정되는 경우 새로운 동작을 명령받을 때마다 사용자 지정의 동작 시간이 궤적을 통하여 자동으로 "순환"됩니다. 이는 결과로 발생하는 궤적을 Turbo PMAC 의 동일한 방법으로 명령받은 것과 동등해집니다.

Coord[x].StepMode 좌표계의 모션 프로그램 싱글 스텝 모드

설명: 모션 프로그램의 싱글 스텝 모드

범위: 0 ..3

단위: 열거 형식

기본값: 0

Coord[x].StepMode 는 "싱글 스텝"(온라인 **s** 또는 버퍼 명령 **step**) 명령에 응답하는 좌표계 동작을 제어합니다. **Coord[x].StepMode** 는 새로운 V1.6 펌웨어(2014 년 1/4 분기 출시)의 기능입니다. 0 인 기본값일 때는 동작은 이 요소가 없었던 이전 펌웨어 버전과 호환성이 있습니다.

Coord[x].StepMode 가 0 인 기본값인 경우에 현재 실행 중인 프로그램에 싱글 스텝 명령이 주어지면 프로그램 연산은 하나의 프로그램 이동이 실행될 때까지 계속합니다(여기서의 "이동"은 드웰 또는 딜레이가 될 수도 있습니다). 이는 1 행, 1 행의 일부 또는 여러 행의 프로그램 연산을 발생시킬 수도 있습니다.

이 모드에서는 현재 실행 중인 프로그램에 싱글 스텝 명령이 주어지면 하나의 추가 프로그램 이동이 계산될 때까지 프로그램 연산은 속행되고 이미 계산된 이동이 실행됩니다. 이를 "종료"(온라인 **q** 또는 버퍼 명령 **pause**) 명령의 동작과 비교해보면 "종료" 명령에서는 그 이상의 프로그램 계산을 실행하지 않고 직전에 계산된 이동의 실행만 완료시킵니다.

계산이 종료되기 전에 프로그램에서 **bstart**(블록 시작) 명령과 만나는 경우 아무리 많은 프로그램 행 또는 이동 명령이 중간에 있어도 **bstop**(블록 정지) 명령과 만날 때까지 싱글 스텝 실행은 계속됩니다.

프로그램이 2D 공구 직경 보정 모드(**ccmode1** 또는 **ccmode2**)의 경우 하나의 이동 실행에 필요한 연산은 특히 보정 시작 시에는 하나 이상의 이동을 계산하는 경우가 있습니다. 또한 보정 종료 시에는 하나의 이동을 계산 또는 계산하지 않을 수도 있습니다(보정 모드의 상태에 따라 다른 이동 수를 계산할 가능성을 **bstart**와 **bstop**이 해결한다는 점에 주의하십시오).

Coord[x].StepMode가 1인 경우 현재 실행 중이 아닌 프로그램에 싱글 스텝 명령이 주어지면 프로그램 이동이 전혀 계산되지 않아도 실행을 시작한 "스택 레벨"(보통 최상위 모션 프로그램)에서 해당 프로그램 행의 종료와 만날 때까지 프로그램 계산을 계속합니다. 단, 이동 명령의 종료가 프로그램 행의 도중에 있으면 계산은 이동 명령의 종료에서 도중 정지합니다.

여러 프로그램 이동이 이 프로그램 행이 호출한 서브 프로그램 내에서 실행되고 하나의 싱글 스텝 명령에 의한 완전한 "고정 사이클"의 실행이 가능합니다.

이 모드에서 현재 실행 중인 프로그램에 싱글 스텝 명령이 주어지면 더는 프로그램 계산은 실행되지 않습니다. 그리고 이동 실행은 이미 계산된 이동을 단순히 종료시키기만 합니다.

프로그램이 2D 공구 직경 보정 모드(**ccmode1** 또는 **ccmode2**)인 경우에 특히 보정 시작 시에는 하나 이상의 이동을 계산하는 경우가 있습니다. 또한 보정 종료 시에는 하나의 이동을 계산 또는 계산하지 않을 수도 있습니다.

프로그램의 실행 행이 호출한 서브루틴에 몇 가지 이동 또는 드웰의 명령이 있으며(**G04** 드웰 서브루틴 또는 고정 사이클 중에서도 같이) 이는 서브루틴의 마지막을 지나서 호출 행의 마지막까지 실행을 계속시키고자 하는 경우 서브루틴은 **bstart** 명령으로 시작하고 **return** 직전에 **bstop** 명령으로 종료해야 합니다. 이는 코드(**G**, **M**, **T**와 **D** 코드)의 서브루틴을 탑재한 RS-274 스타일 CNC 프로그램이 일반적으로 기대되는 싱글 스텝 모드에서의 실행을 가능하게 합니다.

파트 프로그램의 테스트 런을 위해 오퍼레이터가 대화형 싱글 스텝 모드를 희망하는 경우에 해당 모드는 CNC 어플리케이션의 동작 중에서 가장 일반적으로 사용되는 모드입니다.

Coord[x].StepMode가 2인 경우에 현재 실행 중이 아닌 프로그램에 싱글 스텝 명령이 주어질 때 프로그램 이동이 계산되지 않아도 호출된 서브 프로그램을 포함하여 임의의 "스택 레벨"에서 프로그램 행의 종료와 만날 때까지 프로그램 계산을 계속합니다. 단, 이동 명령의 종료가 프로그램 행의 도중에 있으면 계산은 이동 명령의 종료에서 도중 정지합니다.

프로그램이 2D 공구 직경 보정 모드(**ccmode1** 또는 **ccmode2**)인 경우에 특히 보정 시작 시에는 하나 이상의 이동을 계산하는 경우가 있습니다. 또한 보정 종료 시에는 하나의 이동을 계산 또는 계산하지 않을 수도 있습니다.

이 모드에서 현재 실행 중인 프로그램에 싱글 스텝 명령이 주어지면 더는 프로그램 계산은 실행되지 않습니다. 그리고 이동 실행은 이미 계산된 이동을 단순히 종료시키기만 합니다.

고정 사이클과 같은 여러 이동의 서브루틴 개발과 디버그를 지원하기 위해 이 모드는 CNC 어플리케이션의 개발에서 가장 일반적으로 사용되는 모드입니다.

Coord[x].StepMode 가 3 인 경우 현재 실행 중이 아닌 프로그램에 싱글 스텝 명령이 주어질 때 프로그램 이동이 전혀 계산되지 않아도 호출된 서브 프로그램을 포함하여 임의의 "스택 레벨"에서 프로그램 행의 종료와 만날 때까지 프로그램 계산을 계속합니다. 단, 이동 명령의 종료가 프로그램 행의 도중에 있으면 계산은 이동 명령의 종료에서 도중 정지합니다.

프로그램이 2D 공구 직경 보정 모드(**ccmode1** 또는 **ccmode2**)인 경우에 보정된 이동의 사전 계산 버퍼가 완전하지는 않으므로 어떤 이동의 실제 실행을 발생시킬 수 없어도 많아도 하나의 보정된 이동 계산은 실행됩니다.

이 모드에서 현재 실행 중인 프로그램에 싱글 스텝 명령이 주어지면 더는 프로그램 계산은 실행되지 않습니다. 그리고 이동 실행은 이미 계산된 이동을 단순히 종료시키기만 합니다.

이 모드는 일련의 보정된 동작과 고정 사이클과 같은 여러 이동의 서브루틴 개발과 디버그를 지원하는 CNC 어플리케이션의 개발에서 가장 일반적으로 사용되는 모드입니다.

Coord[x].SyncOps 좌표계의 동기 할당 버퍼 크기

설명: 좌표계의 동기 할당 버퍼 크기

범위: $0 \text{ .. } 8192 * \text{Sys.MaxCoords}$

단위: 버퍼링 할당

기본값: 8192

Coord[x].SyncOps 는 좌표계에서의 동기 할당 버퍼의 크기를 지정합니다. 기본 버퍼 설정은 좌표계 하나당 8192 의 할당을 배치합니다. 좌표계 버퍼는 계산 시에 해당 명령과 만나는 때부터 좌표계 중에서 축 또는 모터를 사용하는 해당 명령으로 이어지는 프로그램 내의 다음 이동 명령이 실행될 때까지 보류되는 할당을 모두 저장할 수 있을 정도로 충분히 커야 합니다. 아무리 PLC 프로그램에서 명령을 받았다 해도 지연된 해당 동기 할당이 좌표계 기능이라는 점에 주의하십시오.

대다수의 Power PMAC 어플리케이션에서 8192 의 각 좌표계의 **Coord[x].SyncOps** 기본값은 동기 할당을 위해 필요한 항목을 모두 취급하기에 충분한 큰 값입니다. 단, 이동과 중요한 리트레이스 기능에 대해 여러 개의 동기 할당 가능성이 있으며 록 어헤드 버퍼가 많은 동작을 예측하는 어플리케이션에서는 특정 좌표계의 버퍼는 이보다 큰 크기가 필요할 수도 있습니다.

전원 투입/리셋 시에 Power PMAC CPU 는 동기 할당의 버퍼를 구성하기 위해 **Coord[x].SyncOps** 요소 전체가 저장된 값을 사용합니다. 합계 1M(1,048,576)의 동기 할당이 모든 좌표계를 위해 버퍼링할 수 있습니다. Power PMAC 은 좌표계마다 **Coord[x].SyncOffset** 값을 설정합니다. 그 값은 모든 버퍼 공간의 선두부터 해당 좌표계 버퍼의 오프셋값을 의미하지만 더 하위인 좌표계 전체에 배치된 할당 수의 합계와 같아집니다. 즉 $(0 \leq y < x)$ 에서 **Coord[y].SyncOps** 의 합계가 됩니다. 따라서 하나 이상의 **Coord[x].SyncOps** 요소의 값을 변경하고자 하는 경우 **save** 명령을 전송하고 동기 할당 버퍼를 사용하기 전에 Power PMAC 을 리셋해야 합니다.

Coord[x].Ta 좌표계의 블렌드 이동 가속 시간

설명: 블렌드 이동의 가속 시간

범위: 음이 아닌 부동 소수점

단위: msec

기본값: 100.0

기준 I 변수명: Isx87

Coord[x].Ta 는 msec 단위로 좌표계에 프로그램된 **linear** 또는 **circle** 모드 이동의 가속 명령 시간을 설정합니다. 이 시간은 일련의 블렌드 이동 초기의 정지부터 최초의 가속과 이어서 발생하는 이동 간의 블렌드에 사용됩니다(일련의 이동 종료 시의 정지로의 최종 감속은 **Coord[x].Td** 로 제어됩니다).

Coord[x].Ts 가 **Coord[x].Ta** 보다 작은 경우 가속에 사용되는 시간은 **Coord[x].Ta** 에 **Coord[x].Ts** 를 더한 시간입니다. **Coord[x].Ts** 가 **Coord[x].Ta** 보다 큰 경우 가속에 사용되는 시간은 $2 * \text{Coord[x].Ts}$ 가 됩니다. **Coord[x].Ta** 와 **Coord[x].Ts** 의 양쪽이 0.0 인 경우 명령 가속 곡선은 없어집니다. 그리고 명령 속도 곡선은 스텝 변화가 됩니다. 단, 세그먼테이션이 유효이면(**Coord[x].SegMoveTime**>0), 세그먼테이션 처리로 이 곡선은 약간 매끄러워집니다.

이 좌표계에서 실행되는 모션 프로그램 중의 **ta{데이터}** 명령은 자동으로 이 파라미터를 **{데이터}**값으로 설정합니다.

세그먼트 무효(**Coord[x].SegMoveTime** = 0)인 **linear** 모드 이동 또는 록 어헤드 버퍼 유효(**Coord[x].LHDistance**>0)인 **linear** 와 **circle** 모드 이동에서 해당 파라미터에서 동작하는 모션 프로그램이 모터에 요구하는 가속도가 **Motor[x].InvAmax** 로 지정되는 최대 허용 가속도를 초과하는 경우 제한을 위반하지 않도록 가속 시간이 증가됩니다. 완전한 협조를 유지하기 위해 좌표계의 모든 모터 시간이 확장된다는 점에 주의하십시오.



주의

V1.6(2014 년 1/4 분기 출시) 이하의 펌웨어 버전에서 이 요소의 공장 설정값은 0.0 이었습니다. V1.6 펌웨어에서 기본값이 변경됩니다. 이는 처음 사용하는 사용자가 순간 속도 변화 동작 명령을 최소한으로 전송하도록 하기 위함입니다.

Coord[x].Td 좌표계의 블렌드 이동 최종 감속 시간

설명: 블렌드 이동의 최종 감속 시간

범위: 음이 아닌 부동 소수점

단위: msec

기본값: 100.0

기준 I 변수명: Isx80

Coord[x].Td 는 msec 단위로 좌표계의 프로그램된 **linear** 또는 **circle** 모드 이동의 감속 명령 시간을 설정합니다. 이 시간은 일련의 블렌드 된 이동 종료 시 정지할 때까지의 최종 감속에 사용됩니다. 일련의 블렌드 이동 초기의 정지부터 최소의 가속과 이어서 일어나는 이동 사이의 블렌드 가속 시간은 **Coord[x].Ta** 로 제어합니다.

Coord[x].Ts 가 **Coord[x].Td** 보다 작은 경우 최종 감속에 사용되는 시간은 **Coord[x].Td** 에 **Coord[x].Ts** 를 더한 시간입니다. **Coord[x].Td** 가 **Coord[x].Ta** 보다 큰 경우 최종 감속에 사용되는 시간은 **2*Coord[x].Ts** 입니다. **Coord[x].Td** 와 **Coord[x].Ts** 의 양쪽이 0.0 인 경우 최종 명령 감속 곡선은 없어집니다. 그리고 명령 속도 곡선은 스텝 변화가 됩니다. 단, 세그먼테이션이 유효하면(**Coord[x].SegMoveTime>0**), 세그먼테이션 처리로 이 곡선은 약간 매끄러워집니다.

해당 좌표계에서 실행되는 모션 프로그램의 **td{데이타}** 명령 또는 **ta{데이타}** 명령 중 하나는 자동으로 이 파라미터를 **{데이타}**값으로 설정합니다. 별도의 가속과 감속 시간이 필요한 경우 **td{데이타}** 명령은 **ta{데이타}** 명령 뒤에 사용되어야 합니다.

Coord[x].CornerAccel > 0.0 또는 **Coord[x].CornerError > 0.0** 을 설정하고 코너 블렌드 시간의 자동 계산을 지정하는 경우 **Coord[x].Td** 는 해당 코너에서 사용할 수 있는 최소 시간으로 기능합니다.

세그먼트 무효(**Coord[x].SegMoveTime = 0**)인 **linear** 모드 이동 또는 록 어헤드 유효(**Coord[x].LHDistance>0**)인 **linear** 와 **circle** 모드 이동은 해당 파라미터에서 동작하는 모션 프로그램이 모터에 요구하는 최종 감속도가 **Motor[x].InvDmax** 로 지정되는 최대 허용 최종 감속도를 초과하는 경우에 제한을 위반하지 않도록 최종 감속 시간을 확장합니다. 완전한 협조를 유지하기 위해 좌표계의 모든 모터 시간도 확장된다는 점에 주의하십시오.



주의

V1.6(2014 년 1/4 분기 출시) 이하의 펌웨어 버전에서 이 요소의 공장 설정값은 0.0 이었습니다. V1.6 펌웨어에서 기본값이 변경됩니다. 이는 처음 사용하는 사용자가 순간 속도 변화 동작 명령을 최소한으로 전송하도록 하기 위함입니다.

Coord[x].TimeBaseSlew 좌표계의 타임 베이스 슬루 레이트

설명: 타임 베이스의 슬루 레이트

범위: 부동 소수점

단위: msec/서보 사이클

기본값: 0.00001

기준 I 변수명: Isx94

Coord[x].TimeBaseSlew 는 좌표계의 타임 베이스의 변경율을 제어합니다. **Coord[x].pDesTimeBase** 로 주소 지정된 레지스터의 값에 변경이 있는 경우 실제 타임 베이스값은 **Coord[x].TimeBaseSlew** 로 결정된 비율로 이 "명령" 타임 베이스값을 향해 증가합니다("슬루").

즉, 서보 사이클마다 **Coord[x].TimeBase** 값은 좌표계의 시간적인 수치(msec)를 증가시키는데 사용되고 그 값이 명령 타임 베이스값에 도달할 때까지 **Coord[x].TimeBaseSlew** 의 값이 더해지거나 빼집니다.

"실시간" 타임 베이스값(%100)은 글로벌 데이터 구조체 요소 **Sys.ServoPeriod** 로 결정되고 그 값은 시스템의 클록 주파수를 제어하는 서보 또는 MACRO ASIC 으로 결정되는 서보 클록 사이클의 실제 주기로 설정되어야 합니다(msec 단위). 100% 변화가 발생하는 시간(서보 사이클)은 다음과 같이 계산합니다.

$$100\%SlewTime(servo\_cycles) = \frac{Sys.ServoPeriod}{Coord[x].TimeBaseSlew}$$

마찬가지로 이 변화가 생기는 시간(msec)은 다음과 같이 계산합니다.

$$100\%SlewTime(sec) = \frac{Sys.ServoPeriod^2}{Coord[x].TimeBaseSlew}$$

이 관계를 반대로 하여 주어진 슬루 레이트 시간에 필요한 요소의 값은 다음과 같습니다.

$$Coord[x].TimeBaseSlew = \frac{Sys.ServoPeriod^2}{100\%SlewTime(sec)}$$

보간 연산 중에 사용되는 실제 타임 베이스 변화율의 제어는 타임 베이스를 사용하여 축 속도의 스텝 변화를 유발하지 않고 명령 타임 베이스값의 스텝 변화를 가능하게 합니다.

Coord[x].TimeBaseSlew 가 양의 값으로 설정되는 경우 좌표계 타임 베이스는 좌표계 모터에 대한 최종적인 가속도에 관계없이 각 서보 사이클에 이 값과 같은 비율로 변경됩니다.

Coord[x].TimeBaseSlew 가 음수로 설정되는 경우(2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능)에 좌표계 타임 베이스의 변화율이 더욱 제한될 수 있습니다. 이 값의 절대값과 같은 비율로 변경되고 좌표계 중 한 모터 가속도가 그 **Motor[x].InvAmax** 제한을 초과하는 경우 또는 모터 감속도가 그 **Motor[x].InvDmax** 제한을 초과하는 경우에 변화율이 제한될 수 있습니다. 궤적이 원래 100%의 타임 베이스에서 해당 제약을 초과하지 않도록 지정되는 경우 0 보다 작은 **TimeBaseSlew** 에서는 +/-100% 이내의 변화는 해당 제약을 초과하지 않습니다.

Coord[x].TimeBaseSlew 가 0.0 으로 정확하게 설정되는 경우 실제 타임 베이스값은 변경할 수 없습니다.

Coord[x].Tm 좌표계의 블렌드 이동 시간/피드 레이트

설명:	블렌드 이동 시간/피드 레이트
범위:	부동 소수점
단위:	msec(0 이상인 경우) 또는 사용자 속도 단위(0 보다 작은 경우)
기본값:	-1.0(피드 레이트 모드, 1.0 의 축 단위/시간 단위)
기준 I 변수명:	Isx89

Coord[x].Tm 은 좌표계의 **linear**, **circle** 모드 이동 시간 또는 벡터 속도("피드 레이트")를 설정합니다. 0 보다 큰 경우에 이는 msec 으로 이동 시간을 의미합니다. 그리고 속도는 해당 지정 시간 내에 이동을 종료하기 위해 필요한 속도입니다. 0 보다 작은 경우 그 크기에 따라 **Coord[x].Feedtime** 으로 설정되는 좌표계 시간 단위당 벡터 축 단위의 속도 단위를 가지는 이동의 벡터 속도를 의미합니다. 이동 시간은 해당 지정 속도로 이동을 종료하기 위해 필요한 시간이 됩니다.

이 좌표계에서 실행된 모션 프로그램 중의 **tm{데이타}** 명령은 자동으로 이 파라미터를 **{데이타}**값으로 설정합니다. 해당 좌표계에서 실행된 모션 프로그램의 **F{데이타}** 명령은 자동으로 이 파라미터를 **{데이타}**값의 음수로 설정합니다.

해당 파라미터로 동작하는 모션 프로그램이 모터에 요구하는 속도의 크기가 **Motor[x].MaxSpeed** 로 지정되는 최대 허용 속도를 초과하는 경우에 제한을 위반하지 않도록 이동 시간이 증가됩니다. 완전한 협조를 유지하기 위해 좌표계의 모든 모터 이동 시간이 증가된다는 점에 주의하십시오.

Coord[x].TPCoords 좌표계의 목표 위치 통지

설명:	목표 위치 통지의 축 마스크 워드
범위:	\$0..\$FFFFFFFF
단위:	비트 필드
기본값:	\$0

Coord[x].TPSize 가 0 보다 크게 설정되어 버퍼링 기능이 유효한 경우에 **Coord[x].TPCoords** 는 온라인 **t(목표 위치)** 쿼리 명령에 따라 좌표계의 어느 축이 목표 위치 또는 거리를 통지할지 지정합니다. 좌표계의 모든 유효한 축의 값은 항상 계산되어 좌표계의 로컬 D 변수에 배치됩니다. 그러나 지정된 축의 값만 응답 문자열에 포함됩니다.

Coord[x].TPCoords 는 좌표계의 개별 유효 축을 위한 1비트가 있는 32 비트의 값입니다. 비트가 1로 설정될 때 해당하는 축의 목표 위치가 통지됩니다. 비트가 0으로 설정되는 경우 해당 축의 목표 위치는 통지되지 않습니다. 비트 **i** 와 그것이 표시하는 축은 다음 표와 같습니다.

축	i	축	i	축	i	축	i	축	i	축	i	축	i	축	i
A	0	V	4	Z	8	DD	12	HH	16	OO	20	SS	24	WW	28
B	1	W	5	AA	9	EE	13	LL	17	PP	21	TT	25	XX	29
C	2	X	6	BB	10	FF	14	MM	18	QQ	22	UU	26	YY	30
U	3	Y	7	CC	11	GG	15	NN	19	RR	23	VV	27	ZZ	31

설정되는 각 비트 i 는 2^i 의 값을 **Coord[x].TPCoords** 에 더합니다.

예

A, C, X, Y 와 Z 축의 목표 위치 통지를 유효로 하기 위해서는 비트 0, 2, 6, 7 과 8 이 설정됩니다. 따라서 **Coord[x].TPCoords** 는 \$1C5 가 됩니다.

Coord[x].Tprog 좌표계의 T 코드 서브 프로그램 번호

설명: T 코드 서브루틴의 서브 프로그램 번호

범위: 양의 정수

단위: 서브 프로그램 번호

기본값: 1002

Coord[x].Tprog 는 "T 코드"의 실행에서 어느 서브 프로그램이 호출되는지를 제어합니다. 프로그램 실행이 **T{테이타}** 구문을 검출하면 프로그램의 흐름은 **Coord[x].Tprog** 로 지정된 번호의 서브 프로그램의 **{테이타}**값의 1000 배인 점프 대상 행 레이블로 점프합니다. 예를 들어 **Coord[x].Tprog** 가 1002 인 기본값으로 설정되는 경우 **T5** 는 **subprog 1002** 의 점프 대상 행 레이블 **N5000** 으로 점프합니다.

이 데이터 구조체 요소는 다른 좌표계가 동일한 T 코드에서 다른 서브루틴을 호출할 수 있게 합니다. 단, 기본설정에서 모든 좌표계는 **subprog 1002** 를 T 코드 서브루틴에 사용합니다.

Coord[x].TPSize 좌표계의 목표 위치 버퍼 크기

설명: 이동 목표 위치 저장을 위한 버퍼 크기

범위: 음이 아닌 정수

단위: 프로그램 이동

기본값: 0(버퍼링 없음)

Coord[x].TPSize 는 좌표계 이동의 "목표 위치" 버퍼의 크기를 지정합니다. 이 버퍼는 CNC 가공과 같은 어플리케이션에서 현재 실행 중인 이동의 이동 종료(목표) 위치와 "남은 이동량"의 통지를 용이하게 합니다. 목표 위치는 **Coord[x].TPCoords** 로 지정된 축 버퍼에서 통지됩니다.

Power PMAC 은 이동 방정식을 계산할 때마다 목표 위치를 산출합니다. 블렌드, 스플라인, 동적인 룩 어헤드와 공구 직경 보정과 같은 기능에서는 이동 실행에 앞선 많은 이동에서 해당 계산이 생성됩니다. 이 버퍼는 해당 이동이 실행될 때까지 목표 위치가 저장될 수 있게 합니다. 따라서 사용자는 실행 중인 이동에 대한 관련성이 있는 데이터를 쉽게 얻을 수 있습니다. 이 버퍼는 어떤 이동의 실행에도 영향을 주지 않으며 적절한 이동 실행에도 관여하지 않는다는 점에 주의하십시오.

버퍼는 이동의 연산 시부터 이동 실행 시까지 프로그램 이동을 저장하기에 충분히 큰 크기여야 합니다. 단순한 블렌드 또는 스플라인을 위해서 버퍼는 적어도 3개의 이동을 저장할 수 있어야 합니다. 버퍼 크기가 생성되는 이동 세그먼트로 지정되는 동적인 룩 어헤드에는 버퍼 크기 뿐만 아니라 이동 하나당 세그먼트 최소수로 나뉘진 **Coord[x].LHDistance**(또는 역수가 사용되는 경우에는 **Coord[x].LHSize**)로 지정된 세그먼트 수와 같은 수의 이동을 저장해야 합니다. 2D 공구 직경 보정에서 버퍼 크기는 **Coord[x].CCSize** 에서 버퍼링된 이동과 같은 수의 이동을 저장해야 합니다. **Coord[x].CCSize** 는 간섭 확인의 길이와 가능한 평면 외/거리 0 이동 수도 포함합니다. 3D 공구 직경 보정에서는 추가 버퍼링이 필요 없습니다.

해당 이동 수는 모두 이동의 총수를 얻기 위해 **Coord[x].TPSize** 에서 버퍼에 더해져야 합니다. 이 버퍼에 필요한 메모리 크기는 비교적 작으므로 적절한 통지를 제공하는 데 충분히 많은 목표 위치를 항상 저장할 수 있다는 것을 보증하기 위해 이 버퍼는 상당히 여유 있게 "대형"으로 할 것을 권장합니다.

Coord[x].TPSize 가 0 인 기본값인 경우에 목표 위치의 버퍼링은 실행되지 않습니다.

현재 실행 중인 이동의 버퍼에서의 목표 위치는 온라인 **t** 명령, 버퍼 프로그램 **tread** 명령 또는 데이터 구조체 요소 **Coord[x].TPExec.Pos[i]**에서 구할 수 있습니다.

Power PMAC 은 좌표계의 목표 위치 버퍼를 위해 $(288 * \text{TPSize})$ 바이트의 데이터를 예약합니다. 이 버퍼는 좌표계 버퍼 전체에 할당되는 16M 바이트 공간의 일부입니다. 공구 직경 보정, 목표 위치 그리고 동적인 룩 어헤드에 상당합니다.

예

어플리케이션에서는 200 세그먼트의 동적인 룩 어헤드와 이동 하나당 5 가지 정밀도의 세그먼트를 가지는 블렌드 된 **linear** 와 **circle** 모드 이동을 사용합니다. 2D 공구 직경 보정은 공구 직경 보정 버퍼에 8 개까지의 이동이 버퍼링 됩니다. **Coord[x].TPSize** 는 적어도 $3 + 200/5 + 8 = 51$ 로 설정되어야 합니다. 여유를 더 제공하기 위해서는 75 로 설정합니다.

Coord[x].Ts 좌표계의 블렌드 이동 S 자 곡선 시간

설명: 블렌드 이동의 S 자 곡선 가속/감속 시간

범위: 음이 아닌 부동 소수점

단위: msec

기본값: 50.0

기본 변수명: Isx88

Coord[x].Ts 는 msec 단위로 좌표계에 프로그램된 **linear** 또는 **circle** 모드 이동의 "S 자 곡선" 명령 가속도의 절반의 시간을 설정합니다. 이 시간은 일련의 블렌드 이동 초기의 정지부터 최초의 가속과 일련의 이동 간의 블렌드와 일련의 종료 시의 최종 감속을 위해 사용됩니다.

Coord[x].Ts 가 **Coord[x].Ta**(혹은 **Coord[x].Td**)보다 작은 경우에 2 개의 "S" 시간의 절반 사이에 일정한 가속 또는 감속 시간이 있습니다. 그리고 전체적인 가속(또는 감속) 시간은 **Ta**(또는 **Td**)와 **Ts** 의 합계가 됩니다. **Coord[x].Ts** 가 **Coord[x].Ta**(또는 **Coord[x].Td**)보다 크면 가속 또는 감속에 사용되는 시간은 $2 * \text{Coord[x].Ts}$ 가 됩니다. **Coord[x].Ta**(또는 **Coord[x].Td**)와 **Coord[x].Ts** 의 양쪽이 0.0 인 경우에 명령을 받는 가속(또는 감속) 곡선은 없습니다. 그리고 명령을 받은 속도 곡선은 스텝 변화가 됩니다. 단, 세그먼테이션이 유효(**Coord[x].SegMoveTime**>0)인 경우 세그먼테이션 처리로 이 곡선은 약간 매끄러워집니다.

이 좌표계에서 실행되는 모션 프로그램의 **ts{데이터}** 명령은 자동으로 이 파라미터를 **{데이터}**값으로 설정합니다.

세그먼트 무효(**Coord[x].SegMoveTime** = 0)에서의 **linear** 모드 이동에서는 해당 파라미터에서 동작하는 모션 프로그램이 모터에 요구하는 "저크"(가속도의 변화율)가 **Motor[x].InvJmax** 로 지정되는 최대 허용 저크를 초과하는 경우 제한을 위반하지 않도록 S 자 곡선의 시간을 증가합니다. 완전한 협조를 유지하기 위해 좌표계의 모든 모터 시간이 증가된다는 점에 주의하십시오.



주의

V1.6(2014 년 1/4 분기 출시) 이하의 펌웨어 버전에서 이 요소의 공장 설정값은 0.0 이었습니다. V1.6 펌웨어에서 기본값이 변경됩니다. 이는 처음 사용하는 사용자가 순간 가속도 변화 동작 명령을 최소한으로 전송하도록 하기 위함입니다.

ECAT[i].저장 데이터 구조체 요소

해당 데이터 구조체 요소는 EtherCAT 을 사용할 때 여러 면에서 영향을 주지만 **save** 명령으로 비휘발성 메모리에 저장할 수 있습니다.

ECAT[i].네트워크 전반적인 구성 요소

해당 파라미터는 EtherCAT 네트워크 전체에 영향을 주는 설정 내용 구성에 사용할 수 있습니다.

ECAT[i].AmpEnaTimeout

설명: 앰프 Enable 타임아웃 대기 시간

범위: 음의 값이 아닌 정수

단위: 나노초

기본값: 0

[ECAT[i].AmpEnaTimeout]은 고장 난 앰프가 Enable 인지 아닌지를 판단하기 전인 앰프 Enable 명령을 발행한 뒤의 대기 시간(단위는 나노초)을 지정합니다. 기본값인 0 을 사용한 경우에 결과는 500,000 나노초(500μsec)가 사용됩니다. 이 변수는 앰프를 ON 상태로 하기 전에 더 긴 이동 시간이 필요한 상황에 유용합니다("IEC 61800-7-201"의 '*파워 드라이브 시스템의 유한 상태 오토마톤의 이동 상태*'를 참조).

ECAT[i].DCRefBand

설명: 분산 기준 클록의 최대 허용 드리프트

범위: 음의 값이 아닌 정수

단위: 나노초

기본값: 10,000

[ECAT[i].DCRefBand]는 Power PMAC 의 클록 주파수를 조정하고 드리프트를 줄이기 전인 Power PMAC 의 내부 클록과 분산 기준 클록 간의 최대 드리프트의 크기를 지정합니다. 이 조정은 내부 페이즈 클록 신호(이 신호로 서보 클록 신호가 생성됩니다)의 주파수를 설정하는 Power PMAC 의 파라미터값을 자동으로 변경하여 실행할 수 있습니다. 드리프트의 현재값(부호가 있는 값)은 Status 요소 [ECAT[i].DCClockDiff]에 있습니다.

ECAT[i].DCRefMinus

설명:	분산 기준 클록의 드리프트에 대한 역방향 보정
범위:	음의 값이 아닌 정수
단위:	나노초
기본값:	2

[ECAT[i].DCRefMinus]는 Power PMAC의 주파수가 매우 낮아서 해당 클록과 분산 기준 클록 간의 시간 드리프트가 [ECAT[i].DCRefBand]에 따라 설정된 리미트값을 초과한 경우에 Power PMAC의 인터럽트 주파수를 높이기 위한 조정을 지정합니다. 드리프트의 현재값(부호가 있는 값)은 Status 요소 [ECAT[i].DCClockDiff]에 있습니다.

ECAT[i].DCRefPlus

설명:	분산 기준 클록의 드리프트에 대한 정방향의 보정
범위:	음의 값이 아닌 정수
단위:	나노초
기본값:	2

[ECAT[i].DCRefPlus]는 Power PMAC의 주파수가 매우 높아서 해당 클록과 분산 기준 클록 간의 시간 드리프트가 [ECAT[i].DCRefBand]에 따라서 설정된 리미트값을 초과한 경우에 Power PMAC의 인터럽트 주파수를 낮추는 조정을 지정합니다. 드리프트의 현재값(부호가 있는 값)은 Status 요소 [ECAT[i].DCClockDiff]에 있습니다.

이 파라미터는 i번째의 EtherCAT 마스터의 클록 드리프트가 [ECAT[i].DCRefBand]보다 큰 경우에 PMAC의 위상 인터럽트 레이트를 낮추는 값을 지정합니다. 마스터와 슬레이브 간의 드리프트는 [ECAT[i].DCClockDiff]로 감시할 수 있습니다.

ECAT[i].DCRefSlave

설명:	분산 기준 클록용으로 사용하는 슬레이브
범위:	0 .. n, n = EtherCAT 네트워크상의 슬레이브 수
단위:	없음
기본값:	0

[ECAT[i].DCRefSlave]는 EtherCAT의 분산 클록 기능에 대하여 기준 클록을 제공하는 슬레이브 장치의 인덱스를 지정합니다. 지정한 슬레이브 장치를 찾을 수 없는 경우 또는 기준 클록을 제공할 수 없는 경우에 Power PMAC은 가능한 최초의 슬레이브 장치를 사용합니다. [ECAT[i].DCRefSlave]는 네트워크 상에서 처음에 호환 슬레이브 장치를 선택하기 위해서 보통은 기본값 0을 사용합니다.

ECAT[i].DistrClocks

설명: EtherCAT 분산 클록 기능의 유효화

범위: 0 .. 1

단위: 부울 형식

기본값: 0

[ECAT[i].DistrClocks]은 인덱스 'i'로 EtherCAT 네트워크의 분산 클록 기능의 유·무효 여부를 제어합니다. 기본값인 0 으로 설정하면 이 기능은 무효가 됩니다. 1 로 설정한 경우에 이 기능은 유효가 되며 저장 설정 요소 [ECAT[i].DistrClocksCount]에 의해 주기가 설정됩니다.

EtherCAT 네트워크의 분산 클록 기능은 모든 슬레이브 장치의 사이클 클록을 최초의 슬레이브 장치 로컬 기준 클록에 따라서 올바르게 동기됩니다. 이 동기에는 장치 간의 전송 지연 보정이 포함되며 또한 Power PMAC 의 마스터 클록에 대한 올바른 페이즈 클록이 포함됩니다. 여러 축을 적절하게 조정하려면 이 기능이 유효가 되어야 합니다.

ECAT[i].DistrClocksCount

설명: 분산 클록 기능용의 동기 주기

범위: 0 .. 255

단위: EtherCAT 의 사이클 갱신 주기

기본값: 0

[ECAT[i].DistrClocksCount]는 인덱스 'i'의 EtherCAT 네트워크상의 슬레이브 장치를 Power PMAC 의 마스터 클록에 올바르게 고정하는 연속된 동기 동작 간의 주기를 지정합니다. 이 동작은 ([ECAT[i].DistrClocksCount] + 1)의 네트워크 갱신 사이클마다 실행됩니다. 네트워크는 ([ECAT[i].ServoExtension] + 1)의 Power PMAC 서보 사이클마다 갱신되기 때문에 이 동기 주기는 ([ECAT[i].DistrClocksCount] + 1) * ([ECAT[i].ServoExtension] + 1)의 매 Power PMAC 서보 사이클입니다.

ECAT[i].Enable

설명: EtherCAT 네트워크 유효화

범위: 0 ... 3

단위: 열거

기본값: 0

[ECAT[i].Enable]은 인덱스 'i'에 의해 EtherCAT 네트워크에 대한 데이터 갱신의 유효 여부를 제어합니다. 이것은 다음 4 개의 값 중 하나가 됩니다.

- 0: 사이클릭 갱신과 백그라운드 갱신이 모두 무효입니다.
- 1: 사이클릭 갱신은 유효이지만 백그라운드 갱신은 무효입니다.
- 2: 사이클릭 갱신은 무효이지만 백그라운드 갱신은 유효입니다.
- 3: 사이클릭 갱신과 백그라운드 갱신이 모두 유효입니다.

EtherCAT 네트워크를 통해서 서보를 제어하려면 사이클릭 데이터 갱신을 해야 합니다. I/O 제어는 사이클릭 갱신(높은 우선도, a.k.a. HP) 또는 백그라운드(낮은 우선도, a.k.a. LP) 갱신 중 하나로 실행할 수 있습니다. 일반적으로 사용자는 1 또는 2 의 설정만 사용해야 합니다. 사이클릭 서보 제어인 경우에는 사이클릭 갱신으로 모든 I/O 제어를 하는 것을 권장합니다(예를 들어 해당 파라미터를 1로 설정하고 서보 제어와 I/O 양쪽에 대하여 사이클릭 갱신을 실행하십시오). 3의 설정은 백그라운드 갱신이 정확한 사이클릭 갱신 타이밍과 간섭할 가능성이 있기 때문에 일반적으로 권장하지 않습니다.

[ECAT[*j*].Enable=3]의 설정이 적절한 유일한 상황은 EtherCAT 경유로 실행하는 대량의 비 서보 데이터의 연산이 서보에 적용하는 듀티 사이클에 영향을 주는 경우입니다. EtherCAT의 수신/송신 사이클에는 보통 약 10μsec의 처리 시간이 필요합니다. 임의의 데이터를 서보 레이트로 교환하는 경우에는 모든 데이터를 서보 레이트로 교환해야 합니다. 그 이유는 백그라운드 레이트 및 서보 레이트로 교환하면 EtherCAT인 경우에는 약 20μsec의 프로세서 시간이 걸리기 때문입니다. 단, 서보 주기에 교환된 대용량의 비 서보 EtherCAT 데이터가 있는 경우에는 결과적으로 시간이 너무 오래 걸리기 때문에 [ECAT[*j*].Enable=3]을 사용하여 이 대량의 I/O 데이터를 백그라운드에 둘 수도 있습니다.

ECAT[*j*].IOCount

설명: 사이클릭 동작용으로 설정한 슬레이브 I/O 장치의 수

범위: 0 .. 2048

단위: 대수

기본값: 0

[ECAT[*j*].IOCount]는 EtherCAT 네트워크상의 사이클릭 동작용으로 구성된 입출력 장치의 수를 인덱스 '*i*'로 지정합니다.

동작 시 구조체 [ECAT[*j*].IO[*j*]]로 구성된 슬레이브 I/O 장치에 대하여 *j*가 0부터 [ECAT[*j*].IOCount] - 1의 인덱스값에 의해 사이클릭 데이터 전송은 유효합니다. 이 장치들은 요소 [ECAT[*j*].IO[*j*].Slave], [ECAT[*j*].IO[*j*].Index], [ECAT[*j*].IO[*j*].SubIndex], [ECAT[*j*].IO[*j*].BitLength] 및 [ECAT[*j*].IO[*j*].Input] 안에서 유효한 값을 가져야 합니다.

ECAT[i].LPIOCount

설명:	백그라운드 동작용으로 설정한 슬레이브 I/O 장치의 수
범위:	0 .. 2048
단위:	대수
기본값:	0

[ECAT[i].LPIOCount]는 EtherCAT 네트워크상의 백그라운드(낮은 우선도) 동작용으로 구성된 슬레이브의 I/O 장치 수를 인덱스 'i'로 지정합니다.

동작 시에 구조체의 [ECAT[i].LPIO[j].]로 구성된 슬레이브 I/O 장치에 대하여 j가 0부터 [ECAT[i].LPIOCount] - 1의 인덱스값에 의해 백그라운드의 데이터 전송은 유효가 됩니다. 요소 [ECAT[i].LPIO[j].Slave], [ECAT[i].LPIO[j].Index], [ECAT[i].LPIO[j].SubIndex], [ECAT[i].LPIO[j].BitLength] 및 [ECAT[i].LPIO[j].Input]은 유효한 값을 가집니다.

ECAT[i].LPnotLRW

설명:	조합의 낮은 우선도 읽기/쓰기 무효 제어
범위:	0 .. 1
단위:	부울 형식
기본값:	0

[ECAT[i].LPnotLRW]는 인덱스 'i'에 의해 EtherCAT 네트워크에 대하여 백그라운드(낮은 우선도) 장치에 대한 조합의 읽기/쓰기 동작의 유·무효를 제어합니다. 기본값인 0으로 설정한 경우에 EtherCAT LRW 명령을 사용한 조합의 읽기/쓰기 동작은 유효입니다. 1로 설정한 경우에 해당 조합 동작은 무효가 되며 개별 동작(예를 들어 읽기 동작에 대하여 LRD 또는 쓰기 동작에 대하여 LRW)을 사용해야 합니다.

일부 EtherCAT 슬레이브 장치(Hilscher slave IC를 사용하는 장치 등)는 조합의 읽기/쓰기 동작과 호환성이 없습니다. 이러한 장치가 네트워크상에 존재하는 경우에는 [ECAT[i].LPnotLRW]를 1로 설정해야 합니다.

ECAT[i].LPStateCheck

설명:	EtherCAT의 낮은 우선도 상태 확인의 유효화
범위:	0 .. 1
단위:	부울 형식
기본값:	0

[ECAT[i].LPStateCheck]는 인덱스 'i'에 의해 EtherCAT 네트워크의 낮은 우선도 장치에 대한 상태 확인 기능의 유·무효를 제어합니다. 기본값인 0 으로 설정하면 이 기능은 무효가 됩니다. 1 로 설정한 경우 저장 설정 요소 [ECAT[i].LPStateCheckCount]로 주기가 설정되어 해당 기능은 유효가 됩니다.

EtherCAT 네트워크 상태 확인 기능은 네트워크상의 낮은 우선도 장치의 상태를 정기적으로 감시합니다. 결과는 Status 요소 [ECAT[i].LPDomainState] 및 [ECAT[i].LPOutputDomainState]에 따라 통지됩니다.

ECAT[i].LPStateCheckCount

설명: 낮은 우선도의 상태 확인 기능에 대한 동기 주기

범위: 0 .. 255

단위: EtherCAT 의 사이클 갱신 주기

기본값: 0

[ECAT[i].LPStateCheckCount]는 EtherCAT 네트워크의 낮은 우선도 장치에 대한 연속된 상태 확인 동안의 주기를 인덱스 'i'로 지정합니다. 이 동작은 ([ECAT[i].LPStateCheckCount] + 1)의 네트워크 갱신 사이클 때마다 실행합니다. 네트워크는 ([ECAT[i].ServoExtension] + 1)의 Power PMAC 의 서보 사이클 때마다 갱신되기 때문에 이 동기 주기는 ([ECAT[i].LPStateCheckCount] + 1) * ([ECAT[i].ServoExtension] + 1)의 매 Power PMAC 서보 사이클입니다.

ECAT[i].RTnotLRW

설명: 조합의 실시간 읽기/쓰기 무효 제어

범위: 0 .. 1

단위: 부울 형식

기본값: 0

[ECAT[i].RTnotLRW]는 사이클릭(실시간) 장치 내의 처리 데이터 I/O 에 대한 조합의 읽기/쓰기 동작이 EtherCAT 네트워크에서 무효 여부를 인덱스 'i'에 의해 제어합니다. 기본값인 0 으로 설정한 경우에 EtherCAT LRW 명령을 사용한 조합의 읽기/쓰기 동작은 유효입니다. 1 로 설정한 경우에 해당 조합 동작은 무효가 되며 개별 동작(예를 들어 읽기 동작에 대하여 LRD 또는 쓰기 동작에 대하여 LRW)을 사용해야 합니다.

일부 EtherCAT 슬레이브 장치(Hilscher slave IC 를 사용하는 장치 등)는 조합의 읽기/쓰기 동작과 호환성이 없습니다. 이런 장치가 네트워크상에 존재하는 경우에는 [ECAT[i].RTnotLRW]를 1 로 설정해야 합니다.

ECAT[i].RTStateCheck

설명: EtherCAT의 실시간 상태 확인 유효화

범위: 0 .. 1

단위: 부울 형식

기본값: 0

[ECAT[i].RTStateCheck]는 인덱스 'i'에 의해 EtherCAT 네트워크 실시간 장치에 대한 상태 확인 기능의 유·무효를 제어합니다. 기본값인 0으로 설정하면 이 기능은 무효가 됩니다. 1로 설정한 경우에는 저장 설정 요소 [ECAT[i].RTStateCheckCount]로 주기가 설정되어 해당 기능은 유효가 됩니다.

EtherCAT 네트워크의 이 상태 확인 기능은 네트워크상의 마스터, 도메인 및 슬레이브의 각 장치 상태를 정기적으로 감시합니다. 결과는 Status 요소 [ECAT[i].MasterState], [ECAT[i].RTDomainState], [ECAT[i].RTOutputDomainState] 및 [ECAT[i].Slave[j].State]에 따라서 통지됩니다.

ECAT[i].RTStateCheckCount

설명: 실시간 상태 확인 기능에 대한 동기 주기

범위: 0 .. 255

단위: EtherCAT의 사이클 갱신 주기

기본값: 0

[ECAT[i].RTStateCheckCount]는 EtherCAT 네트워크의 실시간 장치에 대한 연속된 상태 확인 사이의 주기를 인덱스 'i'로 지정합니다. 이 동작은 ([ECAT[i].RTStateCheckCount] + 1)의 네트워크 갱신 사이클 때마다 실행합니다. 네트워크는 ([ECAT[i].ServoExtension] + 1)의 Power PMAC 서보 사이클 때마다 갱신되기 때문에 이 동기 주기는 ([ECAT[i].RTStateCheckCount] + 1) * ([ECAT[i].ServoExtension] + 1)의 매 Power PMAC 서보 사이클입니다.

ECAT[i].ServoExtension

설명: EtherCAT의 사이클릭 갱신 주기 연장

범위: 0 .. 255

단위: Power PMAC의 서보 사이클

기본값: 0

[ECAT[*i*].ServoExtension]은 EtherCAT 네트워크의 사이클릭 갱신 주기를 인덱스 '*i*'로 지정합니다. 이것은 ([ECAT[*i*].ServoExtension] + 1)의 Power PMAC 서보 인터럽트 사이클 때마다 갱신됩니다.

이 주기는 각 모터의 서보 갱신 주기의 연장값인 [Motor[*x*].Stime]와는 따로 제어할 수 있습니다. 이 연장값은 ([Motor[*x*].Stime] + 1)의 서보 인터럽트 주기의 모터 갱신 주기를 설정합니다. 단, 일반적으로 이 설정값은 하나의 어플리케이션 내에서는 같아야 하기 때문에 Power PMAC 은 네트워크 갱신 사이클 때마다 1 회만 네트워크에 대하여 새 데이터를 계산합니다.

ECAT[*i*].SlaveCount

설명: 사이클릭 동작용으로 설정한 슬레이브 서보 장치의 수

범위: 0 .. 512

단위: 대수

기본값: 0

[ECAT[*i*].SlaveCount]는 EtherCAT 네트워크상의 사이클릭 동작용으로 구성된 슬레이브 서보 장치의 수를 인덱스 '*i*'로 지정합니다. 이것은 보통 **ecat assign** 및 **ecat config** 명령 실행 시에 자동으로 설정됩니다. 잘못된 설정을 하면 네트워크 통신 이상의 원인이 되므로 사용자가 직접 이 요소를 설정하는 경우에는 특히 주의해야 합니다.

동작 시 구조체 [ECAT[*i*].Slave[*j*].] 로 구성된 슬레이브 서보 장치에 대하여 *j* 가 0 부터 [ECAT[*i*].SlaveCount] – 1 인 인덱스값에 의해 사이클릭 데이터 전송은 유효가 됩니다. 이 장치는 요소 [ECAT[*i*].Slave[*j*].VendorID], [ECAT[*i*].Slave[*j*].ProductCode], [ECAT[*i*].Slave[*j*].Position] 및 [ECAT[*i*].Slave[*j*].Alias] 안에서 유효한 값을 가져야 합니다. 또한 [ECAT[*i*].Slave[*j*].Enable]은 1 로 설정해야 합니다.

ECAT[*j*].사이클릭 I/O 구성 요소

다음 구조체는 우선 순위가 높은 인터럽트로 사용하는 I/O 장치의 구성에 관련됩니다. 이 장치는 슬레이브 앰프와는 다른 카테고리 분류됩니다.

ECAT[*j*].IO[*k*].BitLength

설명:	EtherCAT의 사이클릭 I/O뱅크 소스의 슬레이브 장치 PDO의 비트 수
범위:	0, 1, 2, 3, 4, 5, 6, 7, 8, 16, 32 또는 32를 넘는 정수
단위:	비트
기본값:	0

[ECAT[*j*].IO[*k*].BitLength]는 [ECAT[*j*].IO[*k*].Slave](이것은 EtherCAT 네트워크에 대하여 인덱스 '*j*'에 의해 I/O뱅크 *k*의 사이클릭 입출력인 경우의 소스)에 따라서 지정된 슬레이브 장치상의 프로세스 데이터 개체(PDO)에 대하여 송수신하는 비트 수를 지정합니다.

이 파라미터가 32를 넘는 경우에 I/O 데이터는 [ECAT[*j*].IO[*k*].Data] 내에 저장되지 않고 대신 [ECAT[*j*].IOBuffer[*n*]] 내에 저장됩니다.

ECAT[*j*].IO[*k*].BitPosition

설명:	슬레이브 장치 PDO에 대한 전송 비트 수
범위:	0 ... 31
단위:	비트
기본값:	0

[ECAT[*j*].IO[*k*].BitLength] = 1인 경우에 [ECAT[*j*].IO[*k*].BitPosition]은 [ECAT[*j*].IO[*k*].Slave](이것은 EtherCAT 네트워크에 대하여 인덱스 '*j*'로 I/O뱅크 *k*의 사이클릭 입출력에 대한 소스)에 따라서 지정된 슬레이브 장치 상에서 프로세스 데이터 개체(PDO)의 워드 중 어떤 비트를 읽을 것인지 또는 쓸 것인지 지정합니다.

ECAT[*j*].IO[*k*].Index

설명: EtherCAT 의 사이클릭 I/O बैंक 소스의 슬레이브 장치 PDO 의 인덱스

범위: \$0 ... \$FFFF

단위: 열거

기본값: \$0

[ECAT[*j*].IO[*k*].Index]는 [ECAT[*j*].IO[*k*].Slave](이것은 EtherCAT 네트워크에 대하여 인덱스 '*j*'로 I/O बैंक *k* 의 사이클릭 입출력을 위한 소스)에 따라서 지정된 슬레이브 장치상의 프로세스 데이터 개체(PDO)의 인덱스 번호를 지정합니다.

ECAT[*j*].IO[*k*].Input

설명: EtherCAT 의 사이클릭 I/O बैंक 소스의 슬레이브 장치 PDO 의 방향

범위: 0 ... 1

단위: 부울 형식

기본값: 0

[ECAT[*j*].IO[*k*].Input]은 EtherCAT 네트워크에 대하여 인덱스 '*j*'에 의해 I/O बैंक *k* 의 사이클릭 입출력인 경우의 데이터 전송 방향을 지정합니다. 이것을 1 로 설정한 경우에 이 I/O बैंक는 출력을 실행합니다. 이것을 0 으로 설정한 경우에 이 I/O बैंक는 입력을 실행합니다.

ECAT[*j*].IO[*k*].Slave

설명: EtherCAT 의 사이클릭 I/O बैंक 소스의 슬레이브 장치 번호

범위: 0 ... 255

단위: 열거

기본값: 0

[ECAT[*j*].IO[*k*].Slave]는 EtherCAT 네트워크에 대하여 인덱스 '*j*'에 의해 I/O बैंक *k* 의 사이클릭 입출력에 대한 소스가 되는 슬레이브 장치의 데이터 구조체의 인덱스값 '*j*'를 지정합니다. 즉 해당 I/O 포인트는 데이터 구조체 [ECAT[*j*].Slave[*j*]]에 할당된 장치상에 설정됩니다. 장치의 이 데이터 구조체는 올바르게 구성하고 또한 장치가 유효이어야 합니다. I/O बैंक의 인덱스값 '*k*'는 슬레이브 장치의 인덱스 '*j*'와 같지 않아도 됩니다.

ECAT[*i*].IO[*k*].SubIndex

설명: EtherCAT 의 사이클릭 I/O बैंक 소스의 슬레이브 장치 PDO 의 하위 인덱스

범위: \$0 ... \$FFFF

단위: 열거

기본값: \$0

[ECAT[*i*].IO[*k*].SubIndex]는 [ECAT[*i*].IO[*k*].Slave](이는 EtherCAT 네트워크에 대하여 인덱스 '*i*'로 I/O बैंक *k* 의 사이클릭 입출력을 위한 소스)에 따라서 지정된 슬레이브 장치상의 프로세스 데이터 개체(PDO)의 하위 인덱스 번호를 지정합니다.

ECAT[i].낮은 우선도 I/O 모듈의 구성 요소

다음 구조체는 백그라운드(즉, 우선 순위가 낮은 EtherCAT 인터럽트)에서 사용하는 슬레이브 I/O의 구성을 실행합니다.

ECAT[i].LPIO[k].BitLength

설명:	EtherCAT의 백그라운드의 I/O뱅크 소스의 슬레이브 장치 PDO의 비트 수
범위:	0, 1 ... 8, 16, 32, 64
단위:	비트
기본값:	0

[ECAT[i].LPIO[k].BitLength]는 [ECAT[i].LPIO[k].Slave](이것은 EtherCAT 네트워크에 대하여 인덱스 'i'에 의해 I/O뱅크 k의 백그라운드(낮은 우선도)의 입력 및/또는 출력인 경우의 소스)에 따라서 지정된 슬레이브 장치상의 프로세스 데이터 개체(PDO)에 대하여 송수신을 실행하는 비트 수를 지정합니다.

ECAT[i].LPIO[k].BitPosition

설명:	백그라운드 I/O 슬레이브 장치의 PDO에 대한 전송 비트 수
범위:	0 ... 31
단위:	비트
기본값:	0

[ECAT[i].LPIO[k].BitLength] = 1인 경우에 [ECAT[i].LPIO[k].BitPosition]은 [ECAT[i].LPIO[k].Slave](이것은 EtherCAT 네트워크에 대하여 인덱스 'i'로 I/O뱅크 k의 사이클릭 입출력에 대한 소스)에 따라서 지정된 슬레이브 장치 상에서 프로세스 데이터 개체(PDO)의 워드 중 어떤 비트를 읽을 것인지 또는 쓸 것인지 지정합니다.

ECAT[i].LPIO[k].Index

설명:	EtherCAT의 백그라운드 I/O뱅크 소스의 슬레이브 장치 PDO의 인덱스
범위:	\$0 ... \$FFFF
단위:	열거
기본값:	\$0

[ECAT[i].LPIO[k].Index]는 [ECAT[i].LPIO[k].Slave](이것은 EtherCAT 네트워크에 대하여 인덱스 'i'로 I/O뱅크 k의 백그라운드(낮은 우선도)의 입출력을 위한 소스)에 따라서 지정된 슬레이브 장치상의 프로세스 데이터 개체(PDO)의 인덱스 번호를 지정합니다.

ECAT[*j*].LPIO[*k*].Input

설명: EtherCAT 의 백그라운드 I/O BANK 소스의 슬레이브 장치 PDO 의 방향

범위: 0 ... 1

단위: 부울 형식

기본값: 0

[ECAT[*j*].LPIO[*k*].Input]은 EtherCAT 네트워크에 대하여 인덱스 '*j*'에 의해 I/O BANK *k* 의 백그라운드(낮은 우선도)의 입출력인 경우의 데이터 전송 방향을 지정합니다. 이것을 1 로 설정한 경우에 이 I/O BANK는 출력을 실행합니다. 이것을 0 으로 설정한 경우에 이 I/O BANK는 입력을 실행합니다.

ECAT[*j*].LPIO[*k*].Slave

설명: EtherCAT 의 백그라운드의 I/O BANK 소스의 슬레이브 장치 번호

범위: 0 ... 255

단위: 열거

기본값: 0

[ECAT[*j*].LPIO[*k*].Slave]는 EtherCAT 네트워크에 대하여 인덱스 '*j*'에 의해 I/O BANK *k* 의 백그라운드(낮은 우선도)의 입출력에 대한 소스가 되는 슬레이브 장치의 데이터 구조 인덱스값 '*j*'를 지정합니다. 즉 해당 I/O 포인트는 데이터 구조체 [ECAT[*j*].Slave[*j*]]에 할당된 장치상에 설정됩니다. 장치의 이 데이터 구조체는 올바르게 구성하고 또한 장치는 유효이어야 합니다. I/O BANK의 인덱스값 '*k*'는 슬레이브 장치의 인덱스 '*j*'와 같지 않아도 됩니다.

ECAT[*j*].LPIO[*k*].SubIndex

설명: EtherCAT 의 백그라운드 I/O BANK 소스의 슬레이브 장치 PDO 의 하위 인덱스

범위: \$0 ... \$FFFF

단위: 열거

기본값: \$0

[ECAT[*j*].LPIO[*k*].SubIndex]는 [ECAT[*j*].LPIO[*k*].Slave](이것은 EtherCAT 네트워크에 대하여 인덱스 '*j*'로 I/O BANK *k* 의 백그라운드(낮은 우선도)의 입출력을 위한 소스)에 따라서 지정된 슬레이브 장치상의 프로세스 데이터 개체(PDO)의 하위 인덱스 번호를 지정합니다.

ECAT[i]의 슬레이브 구성 요소

EtherCAT 네트워크상의 각 슬레이브 장치는 동작하기 전에 올바르게 구성된 구조를 갖추고 있어야 합니다. 이 섹션 내의 각 요소는 그 구조를 포함합니다.

ECAT[j].Slave[j].Alias

설명:	EtherCAT의 슬레이브 장치 위치의 별칭 값
범위:	0 ... 65,535
단위:	열거
기본값:	0

[ECAT[j].Slave[j].Alias]는 EtherCAT 네트워크 상에서 인덱스 'j'에 의해 슬레이브 인덱스 'j'에 할당된 슬레이브 장치에 대하여 위치의 별칭 값을 지정합니다. 해당 번호에 의해 슬레이브 장치는 EtherCAT 네트워크 내의 물리적인 위치와 관계없이 고유의 수치 식별자를 가질 수 있습니다. 장치의 해당 번호는 **ecat slaves** 명령으로 찾을 수 있습니다. 해당 번호는 **ecat assign** 또는 **ecat config** 명령에 따라서 이 요소에 자동으로 할당됩니다. 해당 번호는 네트워크 통신에 대하여 장치를 유효로 하기 전에 유효한 값으로 설정해야 합니다.

슬레이브 장치 자체의 별칭 값은 하드웨어 스위치 또는 **ecat alias** 명령으로 설정되어 장치에 따라서 비휘발성 메모리에 저장합니다. Power PMAC 내의 [ECAT[j].Slave[j].Alias]의 값은 장치 내에서 설정한 값과 일치해야 합니다.

이 파라미터에서 사용하는 [ECAT[j].Slave[j].Position]도 참조하십시오.

ECAT[j].Slave[j].AssignActivate

설명:	EtherCAT의 슬레이브 장치의 벤더 고유 요소
범위:	장치 고유
단위:	장치 고유
기본값:	0

[ECAT[j].Slave[j].AssignActivate]는 벤더 고유의 값으로 슬레이브 장치에 대한 XML 장치 설명 파일에서 취득해야 합니다. 보통 이는 슬레이브 장치의 기본적인 동작을 위해서는 필요 없습니다.

ECAT[j].Slave[j].Enable

설명:	EtherCAT의 슬레이브 장치의 유효화 제어
범위:	0 ... 1
단위:	부울 형식
기본값:	0

[ECAT[i].Slave[j].Enable]은 EtherCAT 네트워크 상에서 인덱스 'i'에 따라서 슬레이브 인덱스 'j'에 할당된 슬레이브 장치가 사이클릭 명령에 대한 유효 여부를 제어합니다. 이 값을 기본값인 0으로 설정한 경우 장치는 무효가 됩니다. 1로 설정한 경우 장치는 유효가 됩니다.

구성 요소 [ECAT[i].Slave[j].VendorID], [ECAT[i].Slave[j].ProductCode], [ECAT[i].Slave[j].Alias] 및 [ECAT[i].Slave[j].Position]은 장치를 유효로 하기 전에 올바르게 설정해야 합니다. 이 구조체를 올바르게 신속하게 설정하려면 Power PMAC IDE의 시스템 설정 툴을 사용하면 됩니다.

ECAT[i].Slave[j].Position

설명: EtherCAT의 슬레이브 장치 위치값

범위: 0 ... 65,535

단위: 열거

기본값: 0

[ECAT[i].Slave[j].Position]은 EtherCAT 네트워크 상에서 인덱스 'i'로 슬레이브 인덱스 'j'에 할당된 슬레이브 장치의 절대 위치 또는 상대 위치를 지정합니다. [ECAT[i].Slave[j].Alias]를 0으로 설정한 경우에 [ECAT[i].Slave[j].Position]은 네트워크상의 절대 위치를 지정합니다. [ECAT[i].Slave[j].Alias]를 0보다 큰 값으로 설정한 경우에 [ECAT[i].Slave[j].Position]은 같은 별칭 값을 가지는 최초 장치가 속하는 네트워크상의 위치를 지정합니다.

네트워크 상에서 슬레이브 장치의 절대 위치는 **ecat slaves**의 쿼리 명령에 응답하여 장치에 송신되는 통지의 최초값입니다. 슬레이브 장치의 별칭 값과 상대 위치(별칭 값이 0이외인 경우)는 장치에 대하여 송신되는 통지의 4번째와 5번째값으로 이것들은 콜론(:)으로 구분됩니다.

ECAT[i].Slave[j].ProductCode

설명: EtherCAT의 슬레이브 장치 제품 코드 번호

범위: \$0 ... \$FFFFFFFF

단위: 열거

기본값: 0

[ECAT[i].Slave[j].ProductCode]는 인덱스 'i'로 EtherCAT 네트워크상의 슬레이브 인덱스 'j'에 할당된 슬레이브 장치에 대한 제품 코드 번호를 지정합니다. 장치의 해당 번호는 **ecat slaves** 명령으로 찾을 수 있습니다. 해당 번호는 **ecat assign** 또는 **ecat config** 명령에 따라서 이 요소에 자동으로 할당됩니다. 해당 번호는 네트워크 통신에 대하여 장치를 유효로 하기 전에 유효한 값으로 설정해야 합니다.

ECAT[i].Slave[j].Sync0Cycle

설명: 분산 클록용으로 사용하는 SYNC0의 사이클 시간[나노초]

범위: 장치 고유

단위: 장치 고유

기본값: 0

[ECAT[i].Slave[j].Sync0Cycle]은 분산 클록용으로 사용하는 SYNC0 사이클 시간 [나노초]입니다.

ECAT[i].Slave[j].Sync0Shift

설명: 분산 클록용으로 사용하는 SYNC0 시프트 시간[나노초]

범위: 장치 고유

단위: 장치 고유

기본값: 0

[ECAT[i].Slave[j].Sync0Shift]는 분산 클록용으로 사용하는 SYNC0 시프트 시간 [나노초]입니다.

ECAT[i].Slave[j].Sync1Cycle

설명: 분산 클록용으로 사용하는 SYNC1 사이클 시간[나노초]

범위: 장치 고유

단위: 장치 고유

기본값: 0

[ECAT[i].Slave[j].Sync1Cycle]은 분산 클록용으로 사용하는 SYNC1 사이클 시간 [나노초]입니다.

ECAT[i].Slave[j].Sync1Shift

설명: 분산 클록용으로 사용하는 SYNC1 시프트 시간[나노초]

범위: 장치 고유

단위: 장치 고유

기본값: 0

[ECAT[i].Slave[j].Sync1Shift]는 분산 클록용으로 사용하는 SYNC1 시프트 시간 [나노초]입니다.

ECAT[i].Slave[j].VendorID

설명: EtherCAT 의 슬레이브 장치 벤더의 식별 번호

범위: \$0 ... \$FFFFFFFF

단위: 열거

기본값: 0

[ECAT[i].Slave[j].VendorID]는 인덱스 'i'로 EtherCAT 네트워크 상에서 슬레이브 인덱스 'j'에 할당된 슬레이브 장치에 대한 벤더 식별 번호를 지정합니다. 장치의 해당 번호는 **ecat slaves** 명령으로 찾을 수 있습니다. 해당 번호는 **ecat assign** 또는 **ecat config** 명령에 따라서 이 요소에 자동으로 할당됩니다. 해당 번호는 네트워크 통신에 대하여 장치를 유효로 하기 전에 유효한 값으로 설정해야 합니다.

ECAT[i].슬레이브 프로세스 데이터 개체의 구성 요소

ECAT[i].Slave[j].PDO[k].BitLength

설명:	슬레이브 장치의 프로세스 데이터 개체 길이(비트 수)
범위:	\$0 ... \$20(32)
단위:	비트
기본값:	0

[ECAT[i].Slave[j].PDO[k].BitLength]는 인덱스 'k'로 Power PMAC 의 PDO 이미지에 대하여 송수신을 실행하는 인덱스 'j'의 EtherCAT 네트워크 상에서 슬레이브 인덱스 'j'에 할당된 슬레이브 장치의 프로세스 데이터 개체(PDO) 내의 비트 수를 지정합니다. 이 요소의 값은 [ECAT[i].Slave[j].PDO[k].Index] 및 [ECAT[i].Slave[j].PDO[k].SubIndex]에 대응해야 합니다.

ECAT[i].Slave[j].PDO[k].Index

설명:	슬레이브 장치의 프로세스 데이터 개체의 인덱스값
범위:	\$0 ... \$FFFF
단위:	열거
기본값:	0

[ECAT[i].Slave[j].PDO[k].Index]는 인덱스 'k'로 Power PMAC 의 PDO 이미지에 할당되는 인덱스 'j'의 EtherCAT 네트워크 상에서 슬레이브 인덱스 'j'에 할당된 슬레이브 장치의 프로세스 데이터 개체(PDO)의 인덱스 번호를 지정합니다. 슬레이브당 최대 64 개의 PDO 가 가능합니다.

ECAT[i].Slave[j].PDO[k].Input

설명:	슬레이브 장치의 프로세스 데이터 개체의 방향 제어
범위:	0 ... 1
단위:	부울 형식
기본값:	0

[ECAT[i].Slave[j].PDO[k].Input]은 인덱스 'k'로 Power PMAC 의 PDO 이미지에 할당되는 인덱스 'j'의 EtherCAT 네트워크 상에서 슬레이브 인덱스 'j'에 할당된 슬레이브 장치의 프로세스 데이터 개체(PDO)의 방향을 지정합니다. 0 으로 설정한 경우에 PDO 는 출력 PDO 가 됩니다. 1 로 설정한 경우에 이것은 입력 PDO 가 됩니다. 이 요소의 값은 [ECAT[i].Slave[j].PDO[k].Index] 및 [ECAT[i].Slave[j].PDO[k].SubIndex]에 대응해야 합니다.

ECAT[*j*].Slave[*j*].PDO[*k*].SubIndex

설명: 슬레이브 장치의 프로세스 데이터 개체의 하위 인덱스값

범위: \$0 ... \$FFFF

단위: 열거

기본값: 0

[ECAT[*i*].Slave[*j*].PDO[*k*].SubIndex]는 인덱스 '*k*'로 Power PMAC 의 PDO 이미지에 할당되는 인덱스 '*i*'의 EtherCAT 네트워크 상에서 슬레이브 인덱스 '*j*'에 할당된 슬레이브 장치의 프로세스 데이터 개체(PDO)의 하위 인덱스 번호를 지정합니다. 이 요소의 값은 [ECAT[*i*].Slave[*j*].PDO[*k*].Index]에 대응해야 합니다.

ECAT[i].슬레이브 PDO의 매핑 구성 요소

ECAT[i].Slave[j].PDOMapping[m].Index

설명: 슬레이브 장치의 프로세스 데이터 개체의 매핑 인덱스값

범위: \$0 ... \$FFFF

단위: 열거

기본값: 0

[ECAT[i].Slave[j].PDOMapping[m].Index]는 인덱스 'i'로 EtherCAT 네트워크상의 슬레이브 인덱스 'j'에 할당된 슬레이브 장치의 인덱스 'm'으로 Power PMAC의 PDO 매핑에 할당된 프로세스 데이터 개체용인 인덱스 번호를 지정합니다. 보통 입력의 경우에 PDO 매핑은 '\$1600 ... \$17FF'의 범위 내 출력인 경우에는 '\$1A00 ... \$1BFF'의 범위 내입니다. 슬레이브 장치인 경우의 올바른 값은 슬레이브의 벤더에서 주어집니다.

ECAT[i].Slave[j].PDOMapping[m].PDOCount

설명: 매핑 시의 프로세스 데이터 개체의 슬레이브 장치 번호

범위: \$0 ... \$40(64)

단위: 개체

기본값: 0

[ECAT[i].Slave[j].PDOMapping[k].PDOCount]는 인덱스 'i'의 EtherCAT 네트워크 상에서 슬레이브 인덱스 'j'에 할당된 슬레이브 장치의 인덱스 'm'을 사용한 PDO의 매핑 시에 프로세스 데이터 개체(PDO)의 수를 지정합니다.

ECAT[i].Slave[j].PDOMapping[m].pPDO

설명: 최초 PDO의 슬레이브 장치의 프로세스 데이터 개체의 매핑 주소

범위: 유효한 [ECAT[i].Slave[j].PDO[k].a]의 값

단위: 개체

기본값: 0

[ECAT[i].Slave[j].PDOMapping[m].pPDO]는 인덱스 'i'로 EtherCAT 네트워크 상에서 인덱스 'j'의 슬레이브 장치의 인덱스 'm'에 따른 PDO 매핑에 할당된 최초의 프로세스 데이터 개체(PDO)의 주소를 지정합니다. 보통 주소는 '.a'의 접미사를 가지는 PDO에 대한 데이터 구조명으로 지정됩니다. 일반적으로 이 주소의 수치를 알 필요는 없습니다. 이 매핑에 할당된 다른 PDO는 이 주소에 따라서 지정된 PDO에 대하여 연속 번호의 인덱스값 'k'를 가져야 합니다.

ECAT[i].슬레이브 동기 매니저의 구성 요소

ECAT[i].Slave[j].SyncManager[n].Dir

설명: 슬레이브 장치의 동기 매니저의 방향 제어

범위: 0 ... 2

단위: 열거

기본값: 0

[ECAT[i].Slave[j].SyncManager[n].Dir]은 인덱스 'i'의 EtherCAT 네트워크 상에서 슬레이브 인덱스 'j'에 할당된 슬레이브 장치의 인덱스 'n'의 동기 매니저의 방향을 지정합니다. 출력인 경우에는 1 의 값을 사용하고 입력인 경우에는 2 의 값을 사용합니다. 즉, 입력 PDO 의 동기를 위해 사용하는 SyncManagers 는 2 의 값을 사용하고 또한 출력 PDO 의 동기인 경우에는 1 의 값을 사용합니다.

ECAT[i].Slave[j].SyncManager[n].Index

설명: 슬레이브 장치의 동기 매니저의 인덱스값

범위: 0 ... 8

단위: 열거

기본값: 0

[ECAT[i].Slave[j].SyncManager[n].Index]는 인덱스 'i'의 EtherCAT 네트워크 상에서 슬레이브 인덱스 'j'에 할당된 슬레이브 장치의 인덱스 'n'의 동기 매니저의 인덱스 번호를 지정합니다. 보통 사이클릭 출력 매핑인 경우에는 2 의 값을 사용하고 사이클릭 입력 매핑인 경우에는 3 의 값을 사용합니다. 보통 어떤 슬레이브를 포어그라운드 데이터 전송으로 사용하려면 2개의 SyncManager 가 사용되며 하나는 입력(입력 PDO 용), 다른 하나는 출력(출력 PDO 용)으로 할당됩니다.

ECAT[i].Slave[j].SyncManager[n].PDOMappingCount

설명: 슬레이브 장치의 동기 매니저의 PDO 매핑 횟수

범위: 0 ... 16

단위: 매핑

기본값: 0

[ECAT[i].Slave[j].SyncManager[n].Index]는 인덱스 'i'의 EtherCAT 네트워크 상에서 슬레이브 인덱스 'j'에 할당된 슬레이브 장치의 인덱스 'n'을 가지는 동기 매니저에 할당된 프로세스 데이터 개체(PDO)의 매핑 횟수를 지정합니다.

ECAT[i].Slave[j].SyncManager[n].pPDOMapping

설명: 슬레이브 장치의 동기 매니저의 최초 PDO 매핑 주소

범위: 유효한 [ECAT[i].Slave[j].PDOMapping[m].a]의 값

단위: 개체

기본값: 0

[ECAT[i].Slave[j].SyncManager[n].pPDOMapping]은 인덱스 '*i*'의 EtherCAT 네트워크 상에서 인덱스 '*j*'의 슬레이브 장치의 인덱스 '*n*'을 가지는 동기 매니저에 할당된 최초의 프로세스 데이터 개체(PDO)의 매핑 주소를 지정합니다. 보통 주소는 '*a*'의 접미사를 가지는 PDO에 대한 데이터 구조명으로 지정됩니다. 일반적으로 이 주소의 수치를 알 필요는 없습니다. 이 매니저에 할당된 다른 PDO 매핑은 이 주소에 따라서 지정된 PDO에 대하여 연속번호의 인덱스값 '*m*'을 가져야 합니다.

ECAT[i].Slave[j].SyncManager[n].WatchdogMode

설명: 슬레이브 장치의 동기 매니저의 워치독(감시) 모드

범위: 0 ... 2

단위: 열거

기본값: 0

[ECAT[i].Slave[j].SyncManager[n].Watchdog]은 인덱스 '*i*'의 EtherCAT 네트워크 상에서 슬레이브 인덱스 '*j*'에 할당된 슬레이브 장치의 인덱스 '*n*'을 가지는 동기 매니저의 워치독 타이머의 동작 모드를 지정합니다. 0의 값은 동기 매니저의 기본값 상태를 지정합니다. 1의 값은 워치독을 유효로 하고 2의 값은 워치독을 무효로 합니다.

EncTable[n].저장 데이터 구조체 요소

[EncTable[n].] 데이터 구조체는 엔코더 변환 테이블을 탑재합니다. 이 테이블은 서보 사이클마다 피드백 및 마스터 위치의 데이터를 사전 처리하고 서보 알고리즘으로 사용하는 원시 데이터를 생성합니다.

[EncTable[n]]의 각 인덱스값은 항목을 지정합니다. 이것은 특정 데이터를 처리하고 각 서보 사이클에서 모터가 보통 사용하는 단일 결과의 값을 피드백 또는 마스터 위치의 값으로 생성합니다. 인덱스값은 0 부터 [Sys.MaxEncoders] -1 까지의 값을 가질 수 있습니다. 이 시스템 상수값은 다른 타입의 Power PMAC에서는 다른 경우가 있지만 일반은 모터 최대 수의 몇 배의 값입니다.

EncTable[n].CosBias

설명: 엔코더 테이블 등록의 코사인항 오프셋

범위: -32,768 .. 32,767

단위: 16 비트 ADC

기본값: 0

[EncTable[n].CosBias]는 특정 타입의 피드백 시에 코사인파의 입력에 대하여 오프셋의 보정에 사용하는 보정률을 지정합니다. 이것은 부호가 있는 16 비트 정수값입니다. [EncTable[n].SinBias]와 다른 타입의 피드백으로 사용되는 32 비트의 설정 요소 [EncTable[n].MaxDelta]의 레지스터를 공유합니다.

Type = 0(테이블의 종료): [CosBias]로 지정된 값은 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): [CosBias] 요소의 워드는 이 타입의 오프셋 기능으로는 사용하지 않습니다. 대신 [MaxDelta] 요소(해당 섹션 참조)를 저장하기 위해서 사용합니다.

Type = 2(2 워드의 읽기): [CosBias] 요소의 워드는 이 타입의 오프셋 기능으로는 사용하지 않습니다. 대신 [MaxDelta] 요소(해당 섹션 참조)를 저장하기 위해서 사용합니다.

Type = 3(소프트웨어 1/T 엔코더 확장): [CosBias]로 지정한 값은 이 타입에서는 사용하지 않습니다. 대신 [MaxDelta] 요소(해당 섹션 참조)를 저장하기 위해서 사용합니다.

Type = 4(사인파 엔코더 아크탄젠트의 확장): [CosBias]의 값은 사인파 엔코더의 측정된 코사인항에 더하는 오프셋을 지정합니다. 이것은 해당하는 A/D 컨버터가 이상적인 0의 값을 통지할 때 [EncTable[n].Cos]에서 읽어낸 값을 부호 반전한 값에 설정해야 합니다. 측정항과 바이어스항은 모두 실제 A/D 컨버터가 다른 분해능을 가지는 경우에도 16 비트의 값으로 처리됩니다. 측정 데이터는 왼쪽 자리맞춤이므로 14 비트의 ADC를 사용하는 경우에 데이터는 16 비트 워드의 상위 14 비트에 배치됩니다.

Type = 5(4 바이트의 읽기): [CosBias] 요소의 워드는 이 타입의 오프셋 기능으로는 사용하지 않습니다. 대신 [MaxDelta] 요소(해당 섹션 참조)를 저장하기 위해서 사용합니다.

Type = 6(리졸버의 직접 변환): [CosBias]의 값은 리졸버의 측정된 코사인항에 더하는 오프셋을 지정합니다. 이것은 해당하는 A/D 컨버터가 이상적인 0 의 값을 통지할 때 [EncTable[n].Cos]에서 읽어낸 값을 부호 반전한 값에 설정해야 합니다. 측정항과 바이어스항은 모두 실제 A/D 컨버터가 다른 분해능을 가지는 경우에도 16비트의 값으로 처리됩니다. 측정 데이터는 왼쪽 자리맞춤이므로 14 비트의 ADC 를 사용하는 경우에 데이터는 16 비트 워드의 상위 14 비트에 배치됩니다.

Type = 7(확장 하드웨어 사인파 보간): [CosBias]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 8(2개의 소스 덧셈): [CosBias] 요소의 워드는 이 타입의 오프셋 기능으로는 사용하지 않습니다. 대신 [MaxDelta] 요소(해당 섹션 참조)를 저장하기 위해서 사용합니다.

Type = 9(2개의 소스 뺄셈): [CosBias] 요소의 워드는 이 타입의 오프셋 기능으로는 사용하지 않습니다. 대신 [MaxDelta] 요소(해당 섹션 참조)를 저장하기 위해서 사용합니다.

Type = 10(트리거 신호에 따른 타임 베이스): [CosBias]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 11(부동 소수점 값의 읽기): [CosBias]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): [CosBias]로 지정한 값은 이 타입에서는 사용하지 않습니다.

EncTable[n].CoverSerror

설명: 엔코더 테이블 등록의 사인파/코사인파 불일치 보정 계수

범위: -32,768 .. 32,767

단위: 32,768 * 등급비의 소수 부분

기본값: 0

[EncTable[i].CoverSerror](사인파와 코사인파의 진폭 오차)는 특정 타입의 피드백의 코사인파 대 사인파의 등급비 보정에 사용하는 보정 계수를 지정합니다. [CoverSerror] 요소는 2012 년 3/4 분기에 출시한 V1.5 펌웨어의 새 기능입니다. 기본값인 0 을 사용하면 모든 변환 방법의 실행은 이전 버전의 펌웨어와 같아집니다.

Type = 0(테이블의 종료): [CoverSerror]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): [CoverSerror]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 2(2 워드의 읽기): [CoverSerror]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 3(소프트웨어 1/T 엔코더 확장): [CoverSerror]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 4(사인파 엔코더 아크탄젠트의 확장): [CoverError]의 값은 측정된 코사인 신호(IC 채널의 [Adc[1]] 또는 [AdcEnc[1]] 레지스터)를 곱하여 측정한 사인과 코사인의 신호 간의 불일치를 보정하기 위해서 계수 (부호가 있는) 소수 부분을 지정합니다.

연산 시에 [CoverError]의 값은 32,768로 나뉘어 결과가 -1.0~+0.9999의 소수 범위에 놓입니다. 그 결과에 1.0이 더해지고 아크탄젠트의 연산이 이루어지기 전에 합계(0.0~1.9999의 범위 내)가 측정된 코사인 신호의 값에 곱해집니다. [CoverError] 0인 기본값에서는 보정 계수가 1.0으로 진폭 보정값이 실행되지 않는다는 점에 주의하십시오.

이 진폭 보정값은 [TanHalfPhi]를 사용하여 [SinBias]와 [CosBias]를 사용하는 오프셋 보정값 뒤, 위상 보정 전에 실행됩니다.

예: 측정된 사인 신호([Adc[0]] 또는 [AdcEnc[0]])의 진폭이 측정된 코사인 신호([Adc[1]] 또는 [AdcEnc[1]])의 진폭보다 5% 작은 경우에는 코사인 신호값에 계수 0.95를 곱해서 소수 성분 -0.05를 나타내는 [CoverError]를 -1638(= -0.05 * 32,768)로 설정합니다.

Type = 5(4 바이트의 읽기): [CoverError]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 6(리졸버의 직접 교환): [CoverError]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 7(확장 하드웨어 사인파 보간): [CoverError]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 8(2 개의 소스 덧셈): [CoverError]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 9(2 개의 소스 뺄셈): [CoverError]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 10(트리거 신호에 따른 타임 베이스): [CoverError]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 11(부동 소수점 값의 읽기): [CoverError]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): [CoverError]로 지정한 값은 이 타입에서는 사용하지 않습니다.

EncTable[n].EncBias

설명: 엔코더 테이블 등록의 사전 적분 오프셋

범위: $-2^{31} \dots 2^{31}-1$

단위: 32 비트 소스 레지스터의 수

기본값: 0

[EncTable[n].EncBias]는 항목이 소스 데이터를 적분하도록 설정된 경우([index4] > 0 및 [index2] < 32)에 사전 적분 오프셋을 지정합니다. 각 서보 사이클에는 [EncBias]의 값이 [index1] 및 [index2]가 지정한 대로 임의의 시프트 연산이 이루어지기 전과 [index4]가 지정한 대로 수치 적분이 이루어지기 전에 32 비트의 소스 데이터에 더합니다. 이 방법의 경우 [EncBias]는 소스 데이터 내의 아날로그 오프셋 등의 바이어스에 대하여 해당 보정을 실행할 수 있습니다.

[EncTable[n].EncBias]는 2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새 기능입니다. 이것은 (이전의 변경 내용을 소스 위치에 저장하기 위해서 적분이 없는 항목에서 사용하는)Status 요소 [EncTable[n].PrevDelta]와 함께 레지스터를 공유합니다. V1.6 이전의 펌웨어 버전에서는 사전 적분 오프셋값을 항목을 적분하기 위해서 [PrevDelta]로 저장했습니다.

Type = 0(테이블의 종료): [EncBias]로 설정한 값은 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): 항목이 적분용으로 설정된 경우에 [EncBias]로 지정한 값을 32 비트의 소스값에 더합니다.

Type = 2(2 워드의 읽기): 항목이 적분용으로 설정된 경우에 [EncBias]로 지정한 값을 결합된 32 비트의 소스값에 더합니다.

Type = 3(소프트웨어 1/T 엔코더 확장): [EncBias]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 4(소프트웨어의 아크탄젠트 엔코더 확장): [EncBias]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 5(4 바이트의 읽기): 항목이 적분용으로 설정된 경우에 [EncBias]로 지정한 값을 결합된 32 비트의 소스값에 더합니다.

Type = 6(리졸버의 직접 변환): [EncBias]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 7(확장 하드웨어 사인파 보간): [EncBias]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 8(2 개의 소스 덧셈): 항목이 적분용으로 설정된 경우에 [EncBias]로 지정한 값을 결합된 32 비트의 소스값에 더합니다.

Type = 9(2 개의 소스의 뺄셈): 항목이 적분용으로 설정된 경우에 [EncBias]로 지정한 값을 결합된 32 비트의 소스값에 더합니다.

Type = 10(트리거 신호에 따른 타임 베이스): [EncBias]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 11(부동 소수점 값의 읽기): 항목이 적분용으로 설정된 경우에 [EncBias]로 지정한 값을 소스의 부동 소수점 값의 레지스터에서 변환된 중간의 정수값에 더합니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): 항목이 적분용으로 설정된 경우에 [EncBias]로 지정한 값을 32 비트의 소스값에 더합니다.

EncTable[n].index1

설명: 엔코더 테이블 등록의 1 번째 변환 계수

범위: (타입 고유, 0 .. 255 의 범위)

단위: (타입 고유)

기본값: 하드웨어 베이스에 따라서 자동으로 구성됩니다.

[EncTable[n].index1]은 이 항목의 원시 데이터를 처리하기 위해서 사용하는 제 1 변환 계수(만약 있다면)를 지정합니다. 이 기능은 [EncTable[n].type]이 지정한 변환 방법으로 변환합니다.

Type = 0(테이블의 종료): [index1]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): [index1]의 값은 보통 소스의 32 비트 레지스터가 최초에 [index2]에 의해 우측 시프트 된 다음에 좌측 시프트 될 비트 수를 지정합니다. 이 동작의 목적은 최종적인 값인 MSB 에 소스 레지스터 실제 데이터의 최상위 비트를 배치하여 '불필요한 데이터'를 제거하여 소스 데이터를 적절하게 롤 오버 처리할 수 있도록 하는 것입니다. 이 동작의 처음에 실제 데이터인 LSB 는 32 비트 레지스터인 LSB 에 보통 배치됩니다. 그리고 보통 [index1]은 32 에서 실제 데이터의 비트 수를 뺀 값으로 설정합니다. 예를 들어 실제 데이터가 14 비트인 경우에 [index1]은 18 로 설정합니다. IDE 의 변환 테이블 설정 메뉴에서는 실제 데이터를 몇 비트 사용할지 설정하면 [index1]의 값이 산출됩니다.

단, [index2]의 값이 32 이상인 경우에 [index1]은 추적 필터의 적분 계인항 K_i 로 작용합니다. 추적 필터의 동작은 다음 [index2]에서 설명합니다. 소스 레지스터가 A/D 컨버터에서 나온 것이 아니면 추적 필터는 이 변환 방법으로 거의 사용하지 않습니다.

Type = 2(2 워드의 읽기): [index1]의 값은 보통 소스의 32 비트 레지스터가 최초에 [index2]에 의해 우측 시프트 된 다음에 좌측 시프트 될 비트 수를 지정합니다. 이 동작의 목적은 최종적인 값인 MSB 에 소스 레지스터 실제 데이터의 최상위 비트를 배치하여 '불필요한 데이터'를 제거하고 소스 데이터를 적절하게 롤 오버 처리할 수 있도록 하는 것입니다. 이 동작의 처음에 실제 데이터인 LSB 는 32 비트 레지스터인 LSB 에 보통 배치됩니다. 그리고 보통 [index1]은 32 에서 실제 데이터의 비트 수를 뺀 값으로 설정합니다. 예를 들어 실제 데이터가 28 비트인 경우에 [index1]은 4 로 설정합니다. IDE 의 변환 테이블 설정 메뉴에서는 실제 데이터를 몇 비트 사용할지 설정하면 [index1]의 값이 산출됩니다.

단, [index2]의 값이 32 이상인 경우에 [index1]은 추적 필터의 적분 계인항 K_i 로 작용합니다. 추적 필터의 동작은 다음 [index2]에서 설명합니다. 추적 필터는 이 변환 방법으로 거의 사용하지 않습니다.

Type = 3(소프트웨어 1/T 엔코더 확장): [index2]의 값이 32 미만인 경우에 [index1]은 이 타입에서는 사용되지 않습니다.

단, [index2]의 값이 32 이상인 경우에 [index1]은 추적 필터의 적분 계인항 K_i 로 작용합니다. 추적 필터의 동작은 다음 [index2]에서 설명합니다. 추적 필터는 이 변환 방법으로 거의 사용하지 않습니다.

Type = 4(사인파 엔코더 아크탄젠트의 확장): [index2]의 값이 32 미만인 경우에 [index1]은 이 타입에서는 사용되지 않습니다.

단, [index2]의 값이 32 이상인 경우에 [index1]은 추적 필터의 적분 게인항 K_i 로 작용합니다. 추적 필터는 다음 [index2]에서 설명합니다. 추적 필터는 사인파 엔코더의 아날로그 읽기값에 포함되는 노이즈 제거에 효과가 있습니다.

Type = 5(4 바이트의 읽기): [index1]의 값은 보통 소스의 32 비트 레지스터가 최초에 [index2]에 의해 우측 시프트 된 다음에 좌측 시프트 될 비트 수를 지정합니다. 이 동작의 목적은 최종적인 값인 MSB 에 소스 레지스터 실제 데이터의 최상위 비트를 배치하여 '불필요한 데이터'를 제거하여 소스 데이터를 적절하게 롤 오버 처리할 수 있도록 하는 것입니다. 이 동작의 처음에 실제 데이터인 LSB 는 32 비트 레지스터인 LSB 에 보통 배치됩니다. 그리고 보통 [index1]은 32 에서 실제 데이터의 비트 수를 뺀 값으로 설정합니다. 예를 들어 실제 데이터가 24 비트인 경우에 [index1]은 8 로 설정합니다. IDE 의 변환 테이블 설정 메뉴에서는 실제 데이터를 몇 비트 사용할지 설정하면 [index1]의 값이 산출됩니다.

단, [index2]의 값이 32 이상인 경우에 [index1]은 추적 필터의 적분 게인항 K_i 로 작용합니다. 추적 필터의 동작은 다음 [index2]에서 설명합니다. 추적 필터는 이 변환 방법으로 거의 사용하지 않습니다.

Type = 6(리졸버의 직접 교환): [index2]의 값이 32 미만인 경우에 [index1]은 이 타입에서는 사용되지 않습니다.

단, [index2]의 값이 32 이상인 경우에 [index1]은 추적 필터의 적분 게인항 K_i 로 작용합니다. 추적 필터의 동작은 다음 [index2]에서 설명합니다. 리졸버 권선의 아날로그 읽기값에 포함되는 노이즈 제거에 추적 필터 사용을 적극 권장합니다.

Type = 7(확장 하드웨어 사인파 보간): [index2]의 값이 32 미만인 경우에 [index1]은 이 타입에서는 사용되지 않습니다. 단, [index2]의 값이 32 이상인 경우에 [index1]은 추적 필터의 적분 게인항 K_i 로 작용합니다. 추적 필터의 동작은 다음 [index2]에서 설명합니다. 보통 보간은 하드웨어 자체에서 필터링하기 때문에 추적 필터는 이 변환 방법에서는 거의 사용되지 않습니다.

Type = 8(2 개의 소스 덧셈): [index1]의 값은 보통 소스의 32 비트 레지스터가 최초에 [index2]에 의해 우측 시프트 된 다음에 좌측 시프트 될 비트 수를 지정합니다. 이 동작의 목적은 최종적인 값인 MSB 에 소스 레지스터 실제 데이터의 최상위 비트를 배치하여 '불필요한 데이터'를 제거하여 소스 데이터를 적절하게 롤 오버 처리할 수 있도록 하는 것입니다. 이 동작의 처음에 실제 데이터인 LSB 는 32 비트 레지스터인 LSB 에 보통 배치됩니다. 그리고 보통 [index1]은 32 에서 실제 데이터의 비트 수를 뺀 값으로 설정합니다. 예를 들어 실제 데이터가 18 비트인 경우에 [index1]은 14 로 설정합니다. IDE 의 변환 테이블 설정 메뉴에서는 실제 데이터를 몇 비트 사용할지 설정하면 [index1]의 값이 산출됩니다.

단, [index2]의 값이 32 이상인 경우에 [index1]은 추적 필터의 적분 게인항 K_i 로 작용합니다. 추적 필터의 동작은 다음 [index2]에서 설명합니다. 소스 레지스터가 A/D 컨버터에서 나온 것이 아니면 추적 필터는 이 변환 방법으로 거의 사용하지 않습니다.

Type = 9(2 개의 소스 뺄셈): [index1]의 값은 보통 소스의 32 비트 레지스터가 처음에 [index2]에 의해 우측 시프트 된 다음에 좌측 시프트 되는 비트 수를 지정합니다. 이 동작의 목적은 최종적인 값인 MSB 에 소스 레지스터 실제 데이터의 최상위 비트를 배치하여 '불필요한 데이터'를 제거하여 소스 데이터를 적절하게 롤 오버 처리할 수 있도록 하는 것입니다. 이 동작의 처음에 실제 데이터인 LSB 는 32 비트 레지스터인 LSB 에 보통 배치됩니다. 그리고 보통 [index1]은 32 에서 실제 데이터의 비트 수를 뺀 값으로 설정합니다. 예를 들어 실제 데이터가 18 비트인 경우에 [index1]은 14 로 설정합니다. IDE 의 변환 테이블 설정 메뉴에서는 실제 데이터를 몇 비트 사용할지 설정하면 [index1]의 값이 산출됩니다.

단, [index2]의 값이 32 이상인 경우에 [index1]은 추적 필터의 적분 계인항 K_i 로 작용합니다. 추적 필터의 동작은 다음 [index2]에서 설명합니다. 소스 레지스터가 A/D 컨버터에서 나온 것이 아니면 추적 필터는 이 변환 방법으로 거의 사용하지 않습니다.

Type = 10(트리거 신호에 의한 타임 베이스): 이 일시적인 상태에서는 [index1]을 0에 설정한 경우에 타임 베이스는 출력 변화 없이 트리거에 응답하지 않습니다. 2로 설정한 경우에 타임 베이스는 동작 상태가 되며 출력은 변하지 않지만 [pEnc1]로 지정한 주소 위치에서 PMAC2 스타일 IC 의 트리거를 감시합니다. 3으로 설정한 경우에 타임 베이스는 동작 상태가 되며 출력은 변하지 않지만 [pEnc1]로 지정한 주소 위치에서 PMAC3 스타일 IC 의 트리거를 감시합니다. 트리거가 발생한 경우에 [index1]은 자동으로 0으로 변하고 Type 은 3 또는 1로 복귀합니다.

Type = 11(부동 소수점 값의 읽기): [index1]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): [index1]에서 지정되는 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

EncTable[n].index2

설명: 엔코더 테이블 등록의 2 번째 변환 계수

범위: (타입 고유, 0 .. 255 의 범위)

단위: (타입 고유)

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

[EncTable[n].index2]는 이 항목의 원시 데이터 처리에 사용하는 2 번째 변환 계수(있는 경우)를 지정합니다. 이 기능은 [EncTable[n].type]이 지정한 변환 방법으로 변화합니다.

Type = 0(테이블의 종료): [index2]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): [index2]의 값이 32 미만인 경우에 [index2]의 값은 소스의 32 비트 레지스터를 최초 연산에서 우측 시프트 할 비트 수를 지정합니다. 이 동작의 목적은 소스 레지스터 하위 비트의 '불필요한 데이터'를 가능한 한 제거하여 최종적인 값인 LSB 에 소스 레지스터 실제 데이터의 최하위 비트를 남기는 것입니다. 그 결과 보통 [index2]는 소스 레지스터에 있는 실제 데이터인 LSB 를 나타내는 비트 번호로 설정됩니다. 예를 들어 실제 데이터인 LSB 가 소스 레지스터의 비트 8 인 경우에 [index2]는 8 로 설정됩니다. IDE 의 변환 테이블 설정 메뉴에서는 '시작 비트 수'를 설정하면 [index2]에 설정된다는 점에 주의하십시오.

[index2]의 값이 32 이상인 경우에 [index2]의 값은 추적 필터의 비례 게인항 K_p 를 지정합니다. 이 경우에 [index1]의 값은 필터의 적분 게인항 K_i 를 지정합니다. 추적 필터의 동작에 대한 설명은 다음과 같습니다. 추적 필터가 유효할 때 시프트 동작은 하지 않습니다. 소스 레지스터가 A/D 컨버터가 아니면 추적 필터는 이 변환 방법으로 거의 사용하지 않습니다.

Type = 2(2 워드의 읽기): [index2]의 값이 32 미만인 경우에 [index2]의 값은 소스의 32 비트 레지스터를 최초 연산에서 우측 시프트 하는 비트 수를 지정합니다. 이 동작의 목적은 소스 레지스터 하위 비트의 '불필요한 데이터'를 가능한 한 제거하여 최종적인 값인 LSB 에 소스 레지스터 실제 데이터의 최하위 비트를 남기는 것입니다. 그 결과 보통 [index2]는 소스 레지스터에 있는 실제 데이터인 LSB 를 나타내는 비트 번호로 설정됩니다. 예를 들어 실제 데이터인 LSB 가 소스 레지스터의 비트 4 인 경우에 [index2]는 4 로 설정됩니다. IDE 의 변환 테이블 설정 메뉴에서는 '시작 비트 수'를 설정하면 [index2]에 설정된다는 점에 주의하십시오.

[index2]의 값이 32 이상인 경우에 [index2]의 값은 추적 필터의 비례 게인항 K_p 를 지정합니다. 이 경우에 [index1]의 값은 필터의 적분 게인항 K_i 를 지정합니다. 추적 필터의 동작에 대한 설명은 다음과 같습니다. 추적 필터가 유효할 때 시프트 동작은 하지 않습니다.

Type = 3(소프트웨어 1/T 엔코더 확장): [index2]의 값이 32 미만인 경우에 [index2]로 지정한 값은 이 타입에서는 사용되지 않습니다. [index2]의 값이 32 이상인 경우에 [index2]의 값은 추적 필터의 비례 게인항 K_p 를 지정합니다. 이 경우에 [index1]의 값은 필터의 적분 게인항 K_i 를 지정합니다. 추적 필터의 동작에 대한 설명은 다음과 같습니다. 추적 필터는 이 변환 방법으로는 거의 사용되지 않습니다.

Type = 4(사인파 엔코더 아크탄젠트의 확장): [index2]의 값이 32 미만인 경우에 [index2]로 지정한 값은 이 타입에서는 사용되지 않습니다. [index2]의 값이 32 이상인 경우에 [index2]의 값은 추적 필터의 비례 게인항 K_p 를 지정합니다. 이 경우에 [index1]의 값은 필터의 적분 게인항 K_i 를 지정합니다. 추적 필터의 동작에 대한 설명은 다음과 같습니다.

Type = 5(4 바이트의 읽기): [index2]의 값이 32 미만인 경우에 [index2]의 값은 소스의 32 비트 레지스터를 최초 연산에서 우측 시프트 하는 비트 수를 지정합니다. 이 동작의 목적은 소스 레지스터의 하위 비트의 '불필요한 데이터'를 가능한 한 제거하여 최종적 값 LSB 에 소스 레지스터의 실제 데이터의 최하위 비트를 남기는 것입니다. 그 결과 보통 [index2]는 소스 레지스터에 있는 실제 데이터인 LSB 를 나타내는 비트 번호로 설정됩니다. 예를 들어 실제 데이터 LSB 가 소스 레지스터의 비트 0 인 경우에 [index2]는 0 으로 설정됩니다. IDE 의 변환 테이블 설정 메뉴에서는 '시작 비트 수'를 설정하면 [index2]에 설정된다는 점에 주의하십시오.

[index2]의 값이 32 이상인 경우에 [index2]의 값은 추적 필터의 비례 게인항 K_p 를 지정합니다. 이 경우에 [index1]의 값은 필터의 적분 게인항 K_i 를 지정합니다. 추적 필터의 동작에 대한 설명은 다음과 같습니다. 추적 필터가 유효할 때 시프트 동작은 하지 않습니다.

Type = 6(리졸버 직접 변환): [index2]의 값이 32 미만인 경우에 [index2]로 지정한 값은 이 타입에서는 사용되지 않습니다. [index2]의 값이 32 이상인 경우에 [index2]의 값은 추적 필터의 비례 게인항 K_p 를 지정합니다. 이 경우에 [index1]의 값은 필터의 적분 게인항 K_i 를 지정합니다. 추적 필터의 동작은 [index2]에서 설명합니다.

Type = 7(확장 하드웨어 사인파 보간): [index2]의 값이 32 미만인 경우에 [index2]로 지정한 값은 이 타입에서는 사용되지 않습니다. [index2]의 값이 32 이상인 경우에 [index2]의 값은 추적 필터의 비례 게인항 Kp 를 지정합니다. 이 경우에 [index1]의 값은 필터의 적분 게인항 Ki 를 지정합니다. 추적 필터의 동작에 대한 설명은 다음과 같습니다. 추적 필터가 유효할 때 시프트 동작은 하지 않습니다. 보간은 하드웨어에서 이미 필터링되었으므로 이 변환 방법에서 추적 필터는 사용되지 않습니다.

Type = 8(2개의 소스 덧셈): [index2]의 값이 32 미만인 경우에 [index2]의 값은 2개의 소스 레지스터의 합계인 32 비트를 최초 연산에서 우측에 시프트 하는 비트 수를 지정합니다. 이 동작의 목적은 소스 레지스터의 하위 비트인 '불필요한 데이터'를 가능한 한 제거하여 결과값인 LSB에 소스 레지스터에 있는 실제 데이터의 최하위 비트를 남기는 것입니다. 그 결과 보통 [index2]는 소스 레지스터에 있는 실제 데이터인 LSB의 비트 번호에 설정됩니다. 예를 들어 실제 데이터인 LSB가 소스 레지스터의 비트 6인 경우에 [index2]는 6으로 설정됩니다.

[index2]의 값이 32 이상인 경우에 [index2]의 값은 추적 필터의 비례 게인항 Kp 를 지정합니다. 이 경우에 [index1]의 값은 필터의 적분 게인항 Ki 를 지정합니다. 추적 필터의 동작에 대한 설명은 다음과 같습니다. 추적 필터가 유효할 때 시프트 동작은 하지 않습니다. 소스 레지스터가 A/D 컨버터에서 나온 것이 아니면 추적 필터는 이 변환 방법으로 거의 사용하지 않습니다.

Type = 9(2개의 소스 뺄셈): [index2]의 값이 32 미만인 경우에 [index2]의 값은 2개의 소스 레지스터의 차인 32 비트 레지스터를 최초 연산에서 우측에 시프트 하는 비트 수를 지정합니다. 이 동작의 목적은 소스 레지스터의 하위 비트인 '불필요한 데이터'를 가능한 한 제거하여 결과값인 LSB에 소스 레지스터에 있는 실제 데이터의 최하위 비트를 남기는 것입니다. 그 결과 보통 [index2]는 소스 레지스터에 있는 실제 데이터인 LSB의 비트 번호에 설정됩니다. 예를 들어 실제 데이터인 LSB가 소스 레지스터의 비트 6인 경우에 [index2]는 6으로 설정됩니다.

[index2]의 값이 32 이상인 경우에 [index2]의 값은 추적 필터의 비례 게인항 Kp 를 지정합니다. 이 경우에 [index1]의 값은 필터의 적분 게인항 Ki 를 지정합니다. 추적 필터의 동작에 대한 설명은 다음과 같습니다. 추적 필터가 유효할 때 시프트 동작은 하지 않습니다. 소스 레지스터가 A/D 컨버터에서 나온 것이 아니면 추적 필터는 이 변환 방법으로 거의 사용하지 않습니다.

Type = 10(트리거 신호에 따른 타임 베이스): [index2]로 지정한 값은 이 일시적인 프리즈 상태로 있는 동안에는 사용되지 않지만 트리거 발생과 동시에 실행 취소하는 Type 1 또는 Type 3의 항목 내에서는 사용될 가능성이 있습니다.

Type = 11(부동 소수점 값의 읽기): [index2]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): [index2]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

추적 필터의 사용

엔코더 변환 테이블의 소프트웨어 추적 필드는 적분기를 가지는 디지털 로 패스 필터입니다. 리졸버-디지털(R/D) 변환기에서 일반적으로 사용하는 하드웨어 추적 필터와 동적으로 동등하면서 일정 속도 또는 위치에서 정상 오차를 내지 않고 측정 노이즈를 감소시키는 경우에 유용합니다.

각 서보 사이클 k 에 대한 추적 필터에 대하여 실행식은 다음과 같습니다.

$$\begin{aligned} Err(k) &= In(k) - Out(k-1) \\ Temp &= \left(\frac{256 - index2}{256} \right) * Err(k) \\ Int(k) &= Int(k-1) + \left(\frac{index1}{256 * 2^{index4}} \right) * Err(k) \\ Out(k) &= Out(k-1) + Temp + Int(k) \end{aligned}$$

[index1]에서 지정하는 적분 계인을 0으로 설정하면 $(256/256 - [index2]) - 1$ 서보 사이클의 시상수를 가지는 1차 로 패스 필터가 됩니다. 속도에서 정상 오차가 존재하기 때문에 1차 필터를 서보 피드백으로 사용하는 것을 권장하지 않지만 마스터 위치의 값을 필터 처리하는 경우에 어느 정도 유용합니다.

[index1]로 지정한 적분 계인이 0보다 큰 경우에 필터는 2차 필터가 됩니다. 이 필터의 고유 주파수 ω_n 는 다음의 방법으로 계산합니다.

$$\omega_n = \frac{1}{T_s} \sqrt{\frac{index1}{256 * 2^{index4}}}$$

여기서 T_s 는 서보 갱신 시간(단위:초)입니다($=[Sys.ServoPeriod]/1,000$).

이 필터의 감쇠비 ζ 는 다음의 방법으로 계산합니다.

$$\zeta = \frac{256 - index2}{2 \sqrt{256 * \frac{index1}{2^{index4}}}}$$

이 필터에서 희망하는 고유 주파수 ω_n 과 감쇠율 ζ 를 사용하여 역산하면 [index2]와 [index1]은 다음의 식을 사용하여 계산합니다.

$$index2 = 256 - 512 * \zeta * \omega * T_s$$

$$index1' = 256 * \omega_n^2 * T_s^2$$

[index1]의 값이 너무 작은 경우에 가장 가까운 정수값에 반올림하여 부정확한 결과를 가져올 수가 있습니다. [index1']이 64 미만인 경우에는 64보다 커지도록 충분한 시간을 배로 한 값을 [index1]로 설정하고 [index4]를 대비한 수로 설정하는 것을 권장합니다.

EncTable[n].index3

설명: 엔코더 테이블 등록의 3 번째 변환 계수

범위: (타입 고유, 0 .. 15 의 범위)

단위: (타입 고유)

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

[EncTable[n].index3]은 이 항목의 원시 데이터 처리에 사용하는 3 번째 변환 계수(있는 경우)를 지정합니다. 이 기능은 [EncTable[n].type]로 지정한 변환 방법으로 변화합니다.

Type = 0(테이블의 종료): [index3]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): [index3]의 값은 소스 데이터의 몇 차 미분으로 제한되는지 또는 그것이 어떻게 제한되는지를 지정합니다. [index3]이 0 과 동일한 경우에 소스 데이터의 1 차 미분(속도)은 0 보다 큰 값으로 지정된 [MaxDelta] 요소로 제한됩니다. 사이클 중에 제한을 초과하지만 이전 사이클에서는 초과하지 않은 경우에 Power PMAC 은 직전 사이클의 변화분을 사용하여 새로운 결과를 계산합니다. 사이클 중에 제한을 초과하고 이전 사이클에서도 초과한 경우에 변화분으로 [MaxDelta]값을 이 사이클에 사용합니다.

[index3]이 0 보다 큰 경우에 소스 데이터의 2 차 미분(가속도)은 0 보다 큰 값으로 지정된 [MaxDelta] 요소로 제한됩니다. [index3]은 가속도 제한을 초과한 경우에 Power PMAC 이 그 직전의 유효한 사이클의 변화를 사용하여 새로운 결과를 계산하고 연속하는 사이클 수를 지정합니다. 제한이 연속하는 사이클 수를 넘어서 초과한 경우에 소스 위치는 제한에 관계없이 사용됩니다.

단, [index4]를 0 보다 큰 값으로 설정하여 소스 데이터의 정수값을 지정한 경우에 [index3]은 사용되지 않는다는 점에 주의하십시오.

Type = 2(2 워드의 읽기): [index3]으로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 3(소프트웨어 1/T 엔코더 확장): [index3]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 4(사인파 엔코더 아크탄젠트의 확장): [index3]이 작용하는 값은 ASIC 채널(Gaten[j].Chan[j].EncCtrl)의 엔코더 디코드 제어값에 해당되어야 합니다. 이것은 변환이 동작하기 위해서 3 또는 7 이어야 합니다. [index3]의 값이 0 으로 설정된 경우에 항목은 다음 서보 사이클에서 ASIC 가 읽으며 자동으로 읽힌 값에 [index3]이 설정됩니다. [index3]은 항목의 초기 설정에서 ASIC 의 디코드 제어값이 변경될 때 0 으로 설정하는 것을 적극 권장합니다.

Type = 5(4 바이트의 읽기): [index3]으로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 6(리졸버의 직접 변환): [index3]의 값은 0 또는 1 로 지정하여 변환의 검출 방향을 지정해야 합니다. [index3] 값의 변경은 어느 회전 방향에서 변환값을 증가시킬지를 변화시킵니다.

Type = 7(확장 하드웨어 사인파 보간): [index3]으로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 8(2 개의 소스 덧셈): [index3]으로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 9(2 개의 소스 뺄셈): [index3]으로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 10(트리거 신호에 따른 타임 베이스): [index3]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 11(부동 소수점 값의 읽기): [index3]으로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): [index3]으로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

EncTable[n].index4

설명: 엔코더 테이블 등록의 4 번째 변환 계수

범위: (타입 고유, 0 .. 15 의 범위)

단위: (타입 고유)

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

[EncTable[n].index4]는 이 항목의 원시 데이터 처리에 사용하는 4 번째 변환 계수(있는 경우)를 지정합니다. 이 기능은 [EncTable[n].type]이 지정한 변환 방법으로 변화합니다.

Type = 0(테이블의 종료): [index4]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): [index2]의 값이 32 미만인 경우에 [index4]의 값은 소스 데이터에 대하여 실행되는 적분 차수를 지정합니다. 유효한 값은 0, 1 및 2 입니다. 기본값인 0(적분 없음)은 일반적으로 위치 센서를 위치 루프에서 사용하는 방법에 가장 적합합니다.

1 의 값은 소스 데이터에 1 회의 적분을 실행합니다. 이것은 위치 루프 내에서 속도 센서를 사용하는 경우(A/D 컨버터 경유의 회전 속도계의 읽기 등)에 적합합니다. 또한 시뮬레이트 한 장치를 1 회의 적분(속도부터 위치 등)으로 간단한 'Type 1'의 서보 루프를 시뮬레이트 하는 경우에도 적합합니다. 서보 루프의 출력은 사용자 공유 메모리 내의 레지스터에 작성되거나 또는 ASIC 의 출력 레지스터(DAC, PWM 또는 PFM)에 작성됩니다. 다음으로 이 항목에서 읽고 시뮬레이트 한 위치 피드백에 대하여 적분합니다.

2 의 값은 소스 데이터에 2 회의 적분을 실행합니다. 이것은 가속도 센서를 위치 루프에 사용하는 경우(A/D 컨버터 경유의 가속도계 등)에 적합합니다. 또한 시뮬레이트 한 장치로 2 회의 적분(토크/가속도부터 위치 등)으로 보다 복잡한 'Type 2'의 서보 루프를 시뮬레이트 하는 경우에도 적합합니다. 서보 루프의 출력은 사용자 공유 메모리 내의 레지스터에 작성되거나 또는 ASIC 의 출력 레지스터(DAC, PWM 또는 PFM)에 작성됩니다. 다음으로 이 항목에서 읽고 시뮬레이트 한 위치 피드백에 대하여 2 회의 적분을 실행합니다.

3~15의 값은 감쇠에 관하여 소스 데이터에 2회의 적분을 실행합니다(2의 값인 경우에 감쇠는 실행되지 않습니다). 이 경우 최초 적분 단계의 값은 $1/(2^{16-index4})$ 을 가중하여 인덱스 함수의 로 패스 필터에 입력됩니다. [index4]의 값이 높을수록 감쇠가 빨라집니다. 이 필터 처리는 2회의 적분을 실행하는 시뮬레이트 한 서보 루프에서 주로 사용되어 이 감쇠를 실행하지 않는 오픈 루프 모드에서는 변경 내용에서 출력을 정지하는 것은 매우 어렵습니다. 이 요소에 대한 3~15의 값은 2014년 4/4 분기에 출시된 V2.0 펌웨어의 새 기능입니다.

소스 데이터가 1 회 또는 2 회 적분된 경우에 항목 요소 [MaxDelta]는 [index3]의 설정 내용과 상관없이 속도 리미트(적분 데이터의 최초 미분 계수의 리미트 등)로 동작합니다. 이 리미트는 [index3]의 설정 내용과 관계없이 무제한으로 실행됩니다.

단, [index2]의 값이 32 이상인 경우에 [index4]는 [index1]과 함께 사용하는 지수를 지정하고 추적 필터로 적분 게인항 K_i 를 설정합니다. 추적 필터의 동작은 앞서 나온 [index2]에서 설명합니다. 추적 필터는 소스 레지스터가 A/D 컨버터에서 나온 것이 아니면 이 변환 방법에서는 거의 사용되지 않습니다.

Type = 2(2 워드의 읽기): [index4]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다. 추적 필터는 이 변환 방법으로는 거의 사용되지 않습니다.

Type = 3(소프트웨어 1/T 엔코더 확장): [index2]의 값이 32 미만인 경우에 [index4]로 지정한 값은 이 타입에서는 사용되지 않습니다. 단, [index2]의 값이 32 이상인 경우에 [index4]는 [index1]과 함께 사용하는 지수를 지정하고 추적 필터로 적분 게인항 K_i 를 설정합니다. 추적 필터의 동작은 앞서 나온 [index2]에서 설명합니다. 추적 필터는 이 변환 방법으로는 거의 사용되지 않습니다.

Type = 4(사인파 엔코더 아크탄젠트의 확장): [index2]의 값이 32 미만인 경우에 [index4]로 지정한 값은 이 타입에서는 사용되지 않습니다. 단, [index2]의 값이 32 이상인 경우에 [index4]는 [index1]과 함께 사용하는 지수를 지정하고 추적 필터로 적분 게인항 K_i 를 설정합니다. 추적 필터의 동작은 앞서 나온 [index2]에서 설명합니다.

Type = 5(4 바이트의 읽기): [index4]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다. 추적 필터는 이 변환 방법으로는 거의 사용되지 않습니다.

Type = 6(리졸버 직접 변환): [index2]의 값이 32 미만인 경우에 [index4]로 지정한 값은 이 타입에서는 사용되지 않습니다. 단, [index2]의 값이 32 이상인 경우에 [index4]는 [index1]과 함께 사용하는 지수를 지정하고 추적 필터로 적분 게인항 K_i 를 설정합니다. 추적 필터의 동작은 앞서 나온 [index2]에서 설명합니다.

Type = 7(확장 하드웨어 사인파 보간): [index4]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 8(2 개의 소스 덧셈): [index4]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 9(2 개의 소스 뺄셈): [index4]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 10(트리거 신호에 따른 타임 베이스): [index4]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 11(부동 소수점 값의 읽기): [index4]로 지정 한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): [index4]로 지정 한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

EncTable[n].index5

설명: 엔코더 테이블 등록의 5 번째 변환 계수

범위: 0 .. 255

단위: (타입 고유)

기본값: 0

[EncTable[i].index5]는 이 항목의 원시 데이터 처리에 사용하는 5 번째 변환 계수(있는 경우)를 지정합니다. 이 함수는 [EncTable[n].type]으로 지정 한 변환 방법으로 변화합니다. [index5] 요소는 2012 년 2/4 분기에 출시된 V1.5 펌웨어의 새 기능입니다. 기본값인 0 을 사용하면 모든 변환 메소드의 실행은 이전 버전의 펌웨어와 같아집니다.

Type = 0(테이블의 종료): [index5]로 지정 한 값은 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): [index5]로 지정 한 값은 이 타입에서는 사용하지 않습니다.

Type = 2(2 워드의 읽기): [index5]로 지정 한 값은 이 타입에서는 사용하지 않습니다.

Type = 3(소프트웨어 1/T 엔코더 확장): [index5]로 지정 한 값은 이 타입에서는 사용하지 않습니다.

Type = 4(사인파 엔코더 아크탄젠트의 확장): [index5]의 값은 소스 데이터에 사용하는 Servo IC 형식의 타입을 지정합니다. 기본값인 0 은 ACC-51E 보간 회로로 사용하는 PMAC2 스타일 IC 를 지정합니다. 이 경우에 항목에 관한 [pEnc] 및 [pEnc1]은 [Gate1[i]] 각 요소의 주소를 지정해야 합니다. 1 의 값은 ACC-24E3 보간 회로로 사용하는 PMAC3 스타일 IC 를 지정합니다. 이 경우에 항목에 관한 [pEnc] 및 [pEnc1]은 [Gate3[i]] 각 요소의 주소를 지정해야 합니다.

Type = 5(4 바이트의 읽기): [index5]로 지정 한 값은 이 타입에서는 사용하지 않습니다.

Type = 6(리졸버의 직접 변환): [index5]로 지정 한 값은 이 타입에서는 사용하지 않습니다.

Type = 7(확장 하드웨어 사인파 보간): [index5]로 지정 한 값은 이 타입에서는 사용하지 않습니다.

Type = 8(2 개의 소스 덧셈): [index5]로 지정 한 값은 이 타입에서는 사용하지 않습니다.

Type = 9(2 개의 소스 뺄셈): [index5]로 지정 한 값은 이 타입에서는 사용하지 않습니다.

Type = 10(트리거 신호에 따른 타임 베이스): [index5]로 지정 한 값은 이 타입에서는 사용하지 않습니다.

Type = 11(부동 소수점 값의 읽기): [index5]의 값은 소스 레지스터 내의 부동 소수점의 값에 대한 사전 스케일링 계수를 지정합니다. 이 값은 ([index5]+1)에서 곱한 다음에 32 비트의 정수값으로 변환되어 중간 처리가 실행됩니다. 이것은 입력값의 범위 내에서 보다 높은 유연성을 제공합니다. 다이렉트 마이크로 스테핑으로 사용하는 경우에 [index5]가 약 63(64의 곱셈 계수용)의 값으로 일반적으로 사용됩니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): [index5]로 지정한 값은 이 타입에서는 사용하지 않습니다.

EncTable[n].index6

설명: 엔코더 테이블 등록의 6 번째 변환 계수

범위: 0 .. 65,535

단위: (타입 고유)

기본값: 0

[EncTable[i].index6]은 이 항목의 원시 데이터 처리에 사용하는 6 번째 변환 계수(있는 경우)를 지정합니다. 이 기능은 [EncTable[n].type]로 지정한 변환 방법으로 변화합니다. [index6] 요소는 2014년 1/4 분기에 출시된 V1.6 펌웨어의 새 기능입니다. 기본값인 0을 사용하면 모든 변환 메소드의 실행은 이전 버전의 펌웨어와 같아집니다. [EncTable[i].index6]은 (타입 4 및 타입 6의 변환에서 사용하는)[EncTable[n].CoverError]와 같은 레지스터를 사용합니다.

Type = 0(테이블의 종료): [index6]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): [index6]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 2(2 워드의 읽기): [index6]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 3(소프트웨어 1/T 엔코더 확장): [index6]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 4(사인파 엔코더 아크탄젠트의 확장): 2 개의 요소가 동일한 레지스터를 공유하므로 [EncType[n].CoverError]가 이 방법으로 변경되면 [index6]의 값도 변화합니다.

Type = 5(4 바이트의 읽기): [index6]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 6(리졸버 직접 변환): [index6]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 7(확장 하드웨어 사인파 보간): [index6]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 8(2 개의 소스 덧셈): [index6]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 9(2 개의 소스 뺄셈): [index6]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 10(트리거 신호에 따른 타임 베이스): [index6]으로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 11(부동 소수점 값의 읽기): [index6]의 값은 소스 레지스터를 단정도(32 비트)의 부동 소수점 값으로 취급할지 배정도(64 비트)의 부동 소수점 값으로 취급할지 지정합니다. 기본값인 0 으로 설정한 경우에 소스값은 단정도로 취급합니다. 이것은 (단정도의 값만 사용하는)이전 펌웨어 버전과 호환성이 있습니다. [index6]을 1 로 설정한 경우에 소스값은 배정도로 취급합니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): [index6]에 의해 지정된 값은 비트 단위의 논리곱 연산에 의해 [pEnc1]로 지정한 레지스터에서 읽어낸 값과 결합된 32 비트 마스크 워드로 처리합니다. 결과인 32 비트값이 0 이 아닌 경우에 해당 서보 사이클의 위치 데이터는 무효가 됩니다. 이 데이터 대신 이전의 유효한 읽기에서 추정하여 사용됩니다.

[index6]의 비트 n ($n = 0 \sim 31$)을 1 로 설정한 경우에 [pEnc1]로 지정한 레지스터의 비트 n 이 에러 비트로 간주됩니다. 보통 이 레지스터는 시리얼 엔코더 인터페이스용 Status 레지스터입니다.

EncTable[n].MaxDelta

설명: 엔코더 테이블 등록의 입력 최대 허용 변화

범위: 0 .. 32,767

단위: 서보 사이클마다의 LSB 또는 서보 사이클/서보 사이클마다의 LSB

기본값: 0

Type = 0(테이블의 종료): [MaxDelta]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): [MaxDelta]의 값은 소스 데이터에 대하여 제한할 수 있는 미분 계수 또는 그 미분 계수를 제한하는 방법을 지정합니다. [MaxDelta]가 0 인 경우에 미분 계수의 제한은 무효가 됩니다.

[MaxDelta]가 0 보다 큰 경우에 미분 계수의 제한이 유효가 되고 대상인 미분 계수의 제한 범위를 지정합니다. 항목의 [index3] 요소가 0 인 경우에 [MaxDelta]는 1 차 미분(속도)의 범위를 지정합니다. 항목의 [index3] 요소가 0 보다 큰 경우에 [MaxDelta]는 2 차 미분(가속도)의 리미트 범위를 지정합니다.

어느 경우에서도 [MaxDelta]의 리미트에 대한 시작 (사전의 미분 계수)단위는 [index2]로 지정한 우측 시프트 다음의 소스 데이터 단위가 됩니다. 이 시프트의 목적은 실제 데이터인 LSB 를 레지스터의 비트 0 으로 두는 것이기 때문에 보통 리미트의 시작 단위는 실제 소스 데이터인 LSB 로 설정됩니다. [MaxDelta]가 속도 리미트를 지정하는 경우에 그 단위는 서보 사이클마다의 LSB 가 됩니다. [MaxDelta]가 가속도 리미트를 지정하는 경우에 그 단위는 서보 사이클/서보 사이클마다의 LSB 가 됩니다.

[MaxDelta]가 속도 리미트로 동작하며([index3] = 0) 그 리미트가 하나의 서보 사이클에서 초과하고 이전 서보 사이클에서는 초과하지 않은 경우에 위치 결과는 이전 서보 사이클의 속도(이것은 항목의 [PrevDelta]Status 요소에 있습니다)를 사용하여 계산합니다. 이 리미트가 직후의 서보 사이클에서도 다시 초과한 경우에 위치 결과는 속도의 [MaxDelta]를 사용하여 계산합니다. 이것은 위치 소스가 변하면 새 위치의 값을 신속하게 구할 수 있게 합니다.

[MaxDelta]가 가속도 리미트로 기능하고([index3]> 0) 그 리미트가 [index3]에서 지정한 연속 서보 사이클 수를 초과한 경우에 위치 결과는 이전 사이클의 가속도(이것은 항목의 [PrevDelta] Status 요소 내에 있습니다)를 사용하여 계산합니다. 이 리미트값이 직후의 서보 사이클에서도 다시 초과한 경우에 위치 결과는 위치 소스가 변하면 새로운 위치의 값을 허가합니다.

[index4]를 0 보다 큰 값으로 설정하고 소스 데이터의 수치 적분을 지정한 경우에 [index3]의 설정 내용에 관계없이 서보 사이클마다의 LSB 로 나타나는 속도 리미트(적분 데이터의 최초 미분 계수의 리미트 등)로 [MaxDelta]가 동작합니다. 이 리미트는 [index3]의 설정 내용과 관계없이 무제한으로 실행됩니다.

Type = 2(2 워드의 읽기): [MaxDelta]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 3(소프트웨어 1/T 엔코더 확장): [MaxDelta]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 4(사인파 엔코더 아크탄젠트의 확장): [MaxDelta] 요소의 워드는 이 타입의 제한 함수로는 사용하지 않습니다. 대신 레지스터는 [SinBias]와 [CosBias] 양쪽 요소(해당 섹션 참조)의 유지에 사용됩니다.

Type = 5(4 바이트의 읽기): [MaxDelta]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 6(리졸버 직접 변환): [MaxDelta]요소의 워드는 이 타입의 제한 기능으로는 사용하지 않습니다. 대신 레지스터는 [SinBias]와 [CosBias] 양쪽 요소(해당 섹션 참조)의 유지에 사용됩니다.

Type = 7(확장 하드웨어 사인파 보간): [MaxDelta]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 8(2 개의 소스 덧셈): [MaxDelta]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 9(2 개의 소스 뺄셈): [MaxDelta]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 10(트리거 신호에 따른 타임 베이스): [MaxDelta]로 지정한 값은 이 일시 Type 에서는 사용하지 않습니다. 단, 트리거 발생 시에 Type=1 로 돌아가는 항목에서 사용됩니다.

Type = 11(부동 소수점 값의 읽기): [MaxDelta]로 지정한 값은 부동 소수점의 소스를 [index5]에서 사전 스케일링하고 중간의 고정 소수점 값으로 변환된 뒤 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): [MaxDelta]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

EncTable[n].pEnc

설명: 엔코더 테이블 등록의 제 1 소스 주소

범위: 적정한 주소

단위: 데이터 구조체 요소의 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

[EncTable[n].pEnc]는 항목이 원시 데이터의 제 1 소스에 어느 레지스터를 사용할지를 지정합니다. 이것에는 해당 레지스터의 주소가 포함됩니다. 이 요소명이라 하더라도 엔코더를 피드백용으로 사용할 필요가 없습니다.

보통 이 변수의 값은 요소의 접미사인 '.a'를 사용하여 사용하는 레지스터를 나타내는 데이터 구조 요소의 주소에 할당하여 설정됩니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

사용하는 대표적인 주소는 다음 변환 타입에 따라서 다릅니다.

Type = 0(테이블의 종료): [pEnc]로 지정한 주소는 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): 많은 센서 장치가 하나의 레지스터에서 읽기 가능한 데이터를 제공합니다.

서브 카운트 데이터에 대한 확장 타입이 없는 원시 엔코더 카운터인 경우에 PMAC2 스타일 ASIC 채널의 위상 포착 레지스터를 읽어야 합니다. 이것은 실제 엔코더 피드백인 경우에도 특히 카운터를 채널의 펄스 출력을 사용한 시뮬레이트 피드백으로 사용하는 경우에도 두 개 모두 필요합니다. 다음은 예시입니다.

EncTable[1].pEnc=Gate1[4].Chan[0].PhaseCapt.a

EncTable[2].pEnc=Gate2[0].Chan[1].PhaseCapt.a

PMAC3 스타일 ASIC 채널의 엔코더 카운터값에 타이머 데이터(1/T) 또는 아크탄젠트 데이터(사인파/코사인파)의 하드웨어 보간을 실행하는 경우에 채널의 서보 포착 레지스터를 읽어야 합니다. 다음은 예시입니다.

EncTable[3].pEnc=Gate3[2].Chan[2].ServoCapt.a

PMAC3 스타일 ASIC 또는 ACC-84 FPGA 경유의 시리얼 엔코더를 읽는 경우에 채널용의 최초 데이터 레지스터를 읽어야 합니다. 다음은 예시입니다.

EncTable[3].pEnc=Gate3[4].Chan[3].SerialEncDataA.a

EncTable[4].pEnc=Acc84E[1].Chan[0].SerialEncDataA.a

PMAC ASIC 펄스(RS-422 모드)에 따라서 직접 구한 자왜식 선형 변위 변환기(MLDT)인 경우에 위치는 에코 펄스가 수신될 때까지 경과 시간에 따라서 측정되지만 이것은 다음과 같이 'TimeBetweenCts' 레지스터에서 읽을 수 있습니다.

EncTable[4].pEnc=Gate1[4].Chan[3].TimeBetweenCts.a

A/D 컨버터 레지스터인 경우에 전위차계 또는 선형 가변 변위 변환기(LVDT) 등에서 다음과 같이 ADC 레지스터(또는 역다중화된 결과)를 읽을 수 있습니다.

EncTable[5].pEnc=Gate1[6].Chan[0].Adc[0].a

EncTable[6].pEnc=AdcDemux.ResultLow[3].a

EncTable[7].pEnc=AdcDemux.ResultHigh[1].a

단일 레지스터 내 I/O 의 페러럴 데이터 레지스터인 경우에 단순히 레지스터 주소를 지정합니다.

EncTable[8].pEnc=Gate2[0].LowIoData.a

이전 항목의 결과를 더 처리하는 경우에 그 항목 결과의 위치 저장소를 다음과 같이 지정합니다.

EncTable[9].pEnc=EncTable[8].PrevEnc.a

사용자가 읽고자 하는 레지스터가 사전에 정의된 데이터 구조체 요소에 할당되지 않은 경우에 사용자는 주소 오프셋으로 레지스터를 지정할 수 있습니다.

EncTable[10].pEnc=Sys.piom+\$A00004

하드웨어 피드백 레지스터가 페이즈 전류(commutation) 위치의 피드백용으로 읽히고 서보 피드백에서 같은 위치의 값(추가 확장 없음)을 사용할 수 있는 경우에 항목은 메모리 내의 모터 전류(commutation) 알고리즘에 저장된 값을 읽을 수 있습니다. 이것은 하드웨어 레지스터의 읽기보다 훨씬 고속으로 실행됩니다. 이것은 보통 Power PMAC 의 계산상의 요건이 리미트에 가까운 경우에만 중요해집니다. 이 설정은 다음과 같습니다.

EncTable[11].pEnc=Motor[11].PrevPhaseEnc.a

Type = 2(2 워드의 읽기): 이 형식은 보통 32 비트의 피드백을 24 비트 버스의 인터페이스에서 읽을 때 사용합니다. 하위 24 비트는 [pEnc]로 지정한 레지스터에서 읽습니다(상위 8 비트는 [pEnc1]으로 지정한 레지스터에서 읽습니다). 다음은 예시입니다.

EncTable[11].pEnc=Gate2[0].LowIoData

Type = 3(소프트웨어 1/T 엔코더 확장): 이 변환 타입인 경우에 테이블은 엔코더의 카운트값과 타이머값을 PMAC2 스타일 ASIC 에서 개별로 읽은 다음에 그것을 수학적으로 조합합니다. 여기서 [pEnc] 요소는 ASIC 채널의 서보 포착 레지스터의 주소로 설정해야 합니다. 다음은 예시입니다.

EncTable[12].pEnc=Gate1[6].Chan[3].ServoCapt.a

EncTable[13].pEnc=Gate2[0].Chan[1].ServoCapt.a

Type = 4(사인파 엔코더 아크탄젠트의 확장): 이 변환 타입의 경우 테이블은 엔코더의 카운트값과 사인파/코사인파의 ADC 값을 읽고 그것을 수학적으로 조합합니다. 여기에서 [pEnc] 요소는 ASIC 채널의 Status 레지스터의 주소로 설정해야 합니다. 다음은 예시입니다.

EncTable[14].pEnc=Gate1[8].Chan[0].Status.a

Type = 5(4 바이트의 읽기): 이 변환 타입의 경우 테이블은 4 개의 레지스터 각각의 미들 8비트를 읽습니다(4 개의 레지스터 모두 사용하지 못하는 경우에는 일부를 마스크할 수 있습니다). 여기에서 [pEnc] 요소는 최하위 바이트의 주소로 설정해야 합니다. 이 데이터 형식의 가장 일반적인 소스는 ACC-14E 상의 IOGATE ASIC 또는 등가의 카드입니다. 다음은 예시입니다.

EncTable[15].pEnc=GateIo[0].DataReg[0].a // Port A LSByte

EncTable[16].pEnc=GateIo[0].DataReg[3].a // Port B LSByte

Type = 6(리졸버 직접 변환): 이 변환 타입인 경우에 테이블은 여기 레지스터의 래치한 현재값을 포함하는 레지스터를 읽습니다. 개별 설정 내용에 관해서는 리졸버의 액세스리 설명서를 참조하십시오.

Type = 7(확장 하드웨어 사인파 보간): 이 변환 타입인 경우에 테이블은 (서보 인터럽트로 래치하는)DSPGATE3 IC 내의 확장 엔코더 카운트값을 읽고 그것을 보다 높은 분해능의 아크탄젠트 요소와 조합합니다. 여기서 [pEnc] 요소는 ASIC 채널의 서보 포착 레지스터의 주소로 설정해야 합니다. 다음은 예시입니다.

EncTable[17].pEnc=Gate3[1].Chan[2].ServoCapt.a

Type = 8(2 개의 소스 덧셈): 이 변환 타입인 경우에 테이블은 함께 더하는 2 개의 32 비트값의 최초값을 포함하는 레지스터를 읽습니다. 대부분의 경우 이 레지스터는 테이블 내의 이전의 (더 낮은 번호)항목의 중간 결과 레지스터(최종적 출력 스케일링 전)입니다. 다음은 예시입니다.

EncTable[18].pEnc=EncTable[9].PrevEnc.a

Type = 9(2 개의 소스 뺄셈): 이 변환 타입인 경우에 (2 번째의 32 비트값을 빼는)32 비트의 값을 포함하는 레지스터를 읽습니다. 대부분의 경우 이 레지스터는 테이블 내의 이전 (더 낮은 번호)항목의 중간 결과 레지스터(최종적 출력 스케일링 전)입니다. 다음은 예시입니다.

EncTable[19].pEnc=EncTable[11].PrevEnc.a

Type = 10(트리거 신호에 따른 타임 베이스): 이 변환 타입인 경우에 Type=1 또는 Type=3 항목의 일시적 변형(프리즈 상태)으로 [EncTable[n].pEnc]의 값은 실행 상태 때와 같은 값으로 유지해야 합니다. Type=1 항목인 경우에 이것은 다음과 같아야 합니다.

EncTable[20].pEnc=Gate3[2].Chan[3].ServoCapt.a

Type=3 항목인 경우에 이것은 다음과 같아야 합니다.

EncTable[21].pEnc=Gate1[8].Chan[2].ServoCapt.a

Type = 11(부동 소수점 값의 읽기): 이 변환 타입인 경우에 테이블은 레지스터에서 부동 소수점의 값을 읽습니다. [index6]을 기본값인 0 으로 설정한 경우에 그것은 단정도(32 비트)의 부동 소수점 값이 되며 [index6]을 1 로 설정한 경우에 배정도(64 비트)의 부동 소수점 값이 됩니다.

단정도의 값인 경우에 [pEnc]는 모터의 서보 명령 출력인 [IqCmd] 레지스터 주소로 설정할 수 있으며 이것은 시뮬레이트의 서보 루프에 유용합니다. 또한 사용자 공유 메모리 버퍼 내의 [Fdata[n]] 레지스터의 주소로도 설정할 수 있습니다. 다음은 예시입니다.

EncTable[22].pEnc=Motor[6].IqCmd.a

EncTable[23].pEnc=Sys.Fdata[495].a

배정도의 값인 경우에 [pEnc]는 모터의 전류(commutation)각의 [PhasePos] 레지스터 주소로 설정할 수 있으며 모터의 최종적 명령 위치인 [DesPos] 레지스터(이것은 명령한 궤도의 추적에 유용합니다)의 주소로 설정할 수 있고 또 사용자 공유 메모리 버퍼 내의 [Ddata[n]] 레지스터의 주소로도 설정할 수 있습니다. 다음은 예시입니다.

EncTable[24].pEnc=Motor[2].PhasePos.a

EncTable[25].pEnc=Motor[19].DesPos.a

EncTable[26].pEnc=Sys.Ddata[3182].a

Type = 12(단일 워드의 읽기, 에러 확인 포함): 많은 센서 장치가 하나의 레지스터에서 읽기 가능한 데이터를 제공하지만 이 방법의 주된 목적은 시리얼 엔코더의 데이터를 처리하는 것입니다. 이 경우 시리얼 엔코더용인 최초의 데이터 레지스터를 읽어야 합니다. 다음은 예시입니다.

EncTable[27].pEnc=Gate3[0].Chan[3].SerialEncDataA.a

EncTable[28].pEnc=Acc84E[2].Chan[0].SerialEncDataA.a

EncTable[n].pEnc1

설명: 엔코더 테이블 등록의 제 2 소스 주소

범위: 적정한 주소

단위: 데이터 구조체 요소의 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

[EncTable[n].pEnc1]은 항목이 원시 데이터의 제 2 소스(만약 있다면)용으로 어느 레지스터를 사용할지 지정합니다. 여기에는 주소를 설정합니다. 이 요소명이라 하더라도 엔코더를 피드백용으로 사용할 필요가 없습니다.

보통 이 변수의 값은 요소의 접미사인 '.a'를 사용하여 사용하는 레지스터를 나타내는 데이터 구조 요소의 주소에 할당하여 설정됩니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

사용하는 대표적인 주소는 다음 변환 타입에 따라서 다릅니다.

Type = 0(테이블의 종료): [pEnc1]로 지정한 주소는 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 병렬 읽기): [pEnc1]로 지정한 주소는 이 타입에서는 사용하지 않습니다.

Type = 2(2 워드의 병렬 읽기): 이 형식은 보통 32 비트의 피드백을 24 비트 버스의 인터페이스에서 읽을 때 사용합니다. 상위 8 비트는 [pEnc1]로 지정한 레지스터에서 읽습니다. 다음은 예시입니다.

EncTable[11].pEnc1=Gate2[0].HighIoData

Type = 3(소프트웨어 1/T 엔코더 확장): 이 변환 타입인 경우에 테이블은 엔코더의 카운트값과 타이머값을 PMAC2 스타일 ASIC 에서 개별로 읽고 그것을 수학적으로 조합합니다. 여기에서 [pEnc1] 요소는 ASIC 채널의 카운트 간의 시간 레지스터 주소로 설정해야 합니다. 다음은 예시입니다.

EncTable[12].pEnc1=Gate1[6].Chan[3].TimeBetweenCts.a

EncTable[13].pEnc1=Gate2[0].Chan[1].TimeBetweenCts.a

Type = 4(사인파 엔코더 아크탄젠트의 확장): 이 변환 타입의 경우 테이블은 엔코더의 카운트값과 사인파/코사인파의 ADC 값을 읽고 그것을 수학적으로 조합합니다. 여기에서 [pEnc1] 요소는 ASIC 채널의 ADC 의 A 상 레지스터 주소로 설정해야 합니다. 다음은 예시입니다.

EncTable[14].pEnc1=Gate1[8].Chan[0].Adc[0].a

Type = 5(4 바이트의 읽기): 이 변환 타입의 경우 테이블은 4 개의 레지스터 각각의 미들 8비트를 읽습니다(4 개의 레지스터 모두 사용하지 못하는 경우에는 일부를 마스크할 수 있습니다). 여기에서 [pEnc1] 요소는 2 번째의 최하위 바이트 주소로 설정해야 합니다. 이 데이터 형식의 가장 일반적인 소스는 ACC-14E 상의 IOGATE ASIC 또는 동등한 카드입니다. 다음은 예시입니다.

EncTable[15].pEnc1=Gate1[0].DataReg[1].a // Port A 2nd LSByte

EncTable[16].pEnc1=Gate1[0].DataReg[4].a // Port B 2nd LSByte

Type = 6(리졸버 직접 변환): 이 변환 타입인 경우에 테이블은 여자값과 사인파/코사인파의 ADC 값을 개별로 읽고 그것들을 수학적으로 조합합니다. 여기에서 [pEnc1] 요소는 ASIC 채널의 ADC 의 A 상 레지스터 주소로 설정해야 합니다. 다음은 예시입니다.

EncTable[17].pEnc1=Gate1[10].Chan[2].Adc[0].a

Type = 7(확장 하드웨어 사인파 보간): 이 변환 타입인 경우에 테이블은 (서보 인터럽트로 래치하는)DSPGATE3 IC 내의 확장 엔코더 카운트값을 읽고 그것을 보다 높은 분해능의 아크탄젠트 요소와 조합합니다. 여기에서 [pEnc1] 요소는 ASIC 채널의 아크탄젠트 요소 주소로 설정해야 합니다. 다음은 예시입니다.

EncTable[17].pEnc1=Gate3[1].Chan[2].AtanSumOfSqr.a

Type = 8(2 개의 소스 덧셈): 이 변환 타입인 경우에 테이블은 함께 더하는 2 개의 32 비트값의 2 번째값을 포함한 레지스터를 읽습니다. 대부분의 경우 이 레지스터는 테이블 내의 이전의 (더 낮은 번호)항목의 중간 결과 레지스터(최종적 출력 스케일링 전)입니다. 다음은 예시입니다.

EncTable[18].pEnc1=EncTable[10].PrevEnc.a

Type = 9(2 개의 소스 뿔셈): 이 변환 타입인 경우에 (최초의 32 비트값에서 빼는)32 비트값을 포함한 레지스터를 읽습니다. 대부분의 경우 이 레지스터는 테이블 내의 이전의 (더 낮은 번호)항목의 중간 결과 레지스터(최종적 출력 스케일링 전)입니다. 다음은 예시입니다.

EncTable[19].pEnc1=EncTable[12].PrevEnc.a

Type = 10(트리거 신호에 따른 타임 베이스): 이 변환 타입인 경우에 트리거 조건을 확인하기 위해서 테이블은 엔코더 채널의 Status 레지스터를 읽습니다. 여기에서 [pEnc1] 요소는 ASIC 채널의 Status 레지스터 주소로 설정해야 합니다. 다음은 예시입니다.

EncTable[20].pEnc1=Gate1[12].Chan[3].Status.a

항목의 실행 중 상태가 Type=3 의 소프트웨어 1/T 엔코더 확장인 경우에 이 요소는 [Gate1[j].Chan[j].TimeBetweenCts.a]의 값으로 자동으로 변환하므로 주의하십시오.

Type = 11(부동 소수점 값의 읽기): [pEnc1]로 지정한 주소는 이 타입에서는 사용하지 않습니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): 이 변환 타입인 경우에 [pEnc1]은 피드백용으로 하나 이상의 에러 비트를 포함하는 레지스터 주소로 설정해야 합니다. 이 타입은 주로 시리얼 엔코더의 프로토콜용이기 때문에 채널의 시리얼 엔코더 인터페이스용인 2 번째 또는 3 번째 데이터 레지스터를 읽어야 합니다. 다음은 예시입니다.

EncTable[21].pEnc1=Gate3[6].Chan[0].SerialEncDataB.a

EncTable[22].pEnc1=Acc84E[2].Chan[1].SerialEncDataC.a

이 테이블 항목의 [index6] 요소는 지정된 레지스터의 어느 비트를 에러 비트로 취급할지 지정합니다.

EncTable[n].PrevDelta

설명: 엔코더 테이블 등록의 사전 적분 오프셋/사전 사이클 변화

범위: $-2^{31}..2^{31}-1$

단위: 32 비트 소스 레지스터의 단위

기본값: 0

[EncTable[n]]의 항목을 소스 데이터와 적분하도록 설정한 경우([index4] = 1 또는 2 및 [index2] < 32), [EncTable[n].PrevDelta]는 소스 데이터의 사전 적분 오프셋으로 기능하는 항목의 저장 설정 요소가 됩니다. 서보 사이클마다 [PrevDelta]의 값이 [index1]과 [index2]에 의한 시프트 동작과 [index4]로 지정한 수치 적분 전에 32 비트의 소스 데이터를 더합니다. 이 방법에서는 [PrevDelta]가 소스 데이터에 있어서 아날로그 오프셋 같은 바이어스 보정으로 기능합니다.

항목이 소스 데이터의 적분을 설정하지 않은 경우에 [PrevDelta]는 직전의 서보 사이클의 1 차 미분([index3]= 0 이고 [MaxDelta]> 0 이면 속도제한) 또는 2 차 미분([index3]> 0 이고 [MaxDelta]> 0 이면 가속도 제한)에 해당하는 상태 요소입니다. 1 차 미분인 경우에 단위는 서보 사이클마다 소스 데이터의 LSB 가 됩니다. 이것이 2 차 미분인 경우 그 단위는 서보 사이클/서보 사이클마다 소스 데이터의 LSB 가 됩니다. 어느 경우에도 실제 소스 데이터의 LSB 가 [index2]에 따른 우측 시프트 뒤의 32 비트 중간 레지스터의 비트 0에 있는 것으로 가정합니다.

2014 년 1/4 분기에 출시된 V1.6 펌웨어 이후 [EncTable[n].EncBias]는 이와 같은 레지스터의 대체명으로 채택되어 각 항목의 적분으로 사전 적분 오프셋용 저장 설정 요소로 사용할 수 있습니다. 이 항목 내에서 레지스터의 값은(이전 펌웨어 버전의 [PrevDelta]가 아니라) [EncBias]로 플래시 메모리에 저장됩니다. 항목 연산은 이 새로운 요소명의 결과로 변경되지 않습니다.

EncTable[n].ScaleFactor

설명: 엔코더 테이블 등록의 출력 스케일 계수

범위: 부동 소수점

단위: 출력단위/시프트 된 데이터의 LSB

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

변환 테이블 항목 직전의 동작에서는 출력을 계산하기 위해서 그때까지 처리된 중간 결과에[[EncTable[n].ScaleFactor]를 곱합니다. 이 동작은 나아가 중간 결과의 정수 형식을 최종 출력용 부동 소수점 형식으로 변환합니다. [ScaleFactor]에 관하여 정해진 규칙은 없습니다. [ScaleFactor]의 값은 항목의 출력값 단위를 실용적으로 정의합니다. 그 결과 해당 값들에 의미를 부여하기 위해서 상식적인 출력 단위가 되도록 각 타입의 변환으로 [ScaleFactor]를 설정하는 방법에 관한 가이드 라인이 있습니다.

부동 소수점 값으로 최종 결과가 서보 루프의 동작에 실질적으로 효과가 있는 소수점의 분해능을 가질 수 있다는 점을 기억하십시오. 소수의 값은 소실되지 않습니다. 피드백 루프로 사용되는 [ScaleFactor]의 변경이 루프 전체의 게인을 변화시킨다는 점에 주의하십시오. 그 결과 [ScaleFactor]가 변경되는 경우에 루프 게인 중 몇 가지는 변경을 보완하기 위해서 대부분의 경우 변경됩니다.

Type = 0(테이블의 종료): [ScaleFactor]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): 2 개의 시프트 연산이 [index2]와 [index1]에서 지정되면 보통 실제 데이터의 LSB 가 32 비트 워드의 비트(32에서 비트 수를 뺀 값)가 됩니다. 대부분 사용자는 실제 데이터 LSB 단위의 출력을 원하므로 이 중간 결과에 $2^{-(32-\text{비트 수})}$ 를 곱해야 합니다. 예를 들어 실제 데이터의 20 비트가 존재하는 경우에 [ScaleFactor]는 $2^{-(32-20)}$ 로 설정하거나 2^{-12} (이것은 0.00244140625 가 됩니다)으로 설정해야 합니다.

사용자가 이것을 요소 내에 직접 설정하는 경우에 [EncTable[1].ScaleFactor]=1/4,096 또는 [EncTable[1].ScaleFactor]=1/exp2(12) 등의 수식을 사용하는 편이 훨씬 간단합니다.

IDE 내의 변환 테이블 설정 메뉴에서 '출력 단위/LSB'를 설정합니다. 기본값은 1.0이며 [ScaleFactor]는 사용자 응답의 값에 $2^{-(32-[\text{비트 수}])}$ 를 곱한 값으로 설정합니다. 여기에서 '#비트'는 다른 메뉴에서 설정합니다.

Type = 2(2 워드의 읽기): [ScaleFactor]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 3(소프트웨어 1/T 엔코더 확장): 타이머에 따른 소수점 카운트가 정수 카운트값과 조합한 뒤에 중간 결과는 9 비트의 소수점 카운트를 가집니다. 그 결과 이것은 AB 상 카운트의 1/512 단위입니다. 대부분의 어플리케이션에서는 출력이 AB 상 카운트 단위이기 때문에 해당 중간 결과에 1/512 을 곱해야 합니다. 그것은 0.001953125 와 같습니다.

요소 내에 직접 설정하는 경우에 [EncTable[1].ScaleFactor]=1/512 또는 [EncTable[1].ScaleFactor]=1/exp2(9) 등의 수식을 사용하면 훨씬 간단해집니다.

IDE 내의 변환 테이블 설정 메뉴에서 '출력 단위/AB 상 카운트'를 설정합니다. 기본값은 1.0 입니다. [ScaleFactor]는 응답의 값에 1/512 을 곱한 값으로 설정됩니다.

Type = 4(사인파 엔코더 아크탄젠트의 확장): PMAC2 스타일 IC([index5]= 0)가 사용될 때 아크탄젠트에 따른 소수점 카운트가 정수 카운트값과 조합한 뒤에 중간 결과는 10 비트의 소수점 카운트를 가집니다. 그 결과 이는 AB 상 카운트의 1/1024(또는 엔코더 1 라인의 1/4096) 단위입니다. 다양한 어플리케이션에서는 항목에 각종 단위를 지정해도 됩니다.

단위를 보간 수의 LSB 로 하는 경우에는 [ScaleFactor]를 1.0 으로 설정해야 합니다. AB 상 카운트(1/4 엔코더 라인)의 단위를 원하는 경우에 [ScaleFactor]는 1/1024 또는 0.0009765625 로 설정되어야 합니다. 많은 서보 루프 게인항이 선택된 스케일 계수에 반비례하여 변화한다는 점을 기억하십시오.

PMAC3 스타일 IC([index5]= 1)가 사용될 때 아크탄젠트에 따른 소수점 카운트가 정수 카운트값과 조합한 뒤에 중간 결과는 14 비트의 소수점 카운트를 가집니다. 그 결과 이는 AB 상 카운트의 1/16,384(혹은 엔코더 1 라인의 1/65,536) 단위입니다. 다양한 어플리케이션은 항목에 각종 단위를 지정해도 됩니다.

보간된 카운트의 LSB 단위를 원하는 경우에 [ScaleFactor]는 1.0 으로 설정되어야 합니다. AB 상 카운트(1/4 엔코더 라인)의 단위를 원하는 경우에 [ScaleFactor]는 1/16,384 로 설정되어야 합니다. 많은 서보 루프 게인항이 선택된 스케일 계수에 반비례하여 변화한다는 점을 기억하십시오.

Type = 5(4 바이트의 읽기): [ScaleFactor]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 6(리졸버 직접 변환): 중간 결과는 32 비트 레지스터의 상위 16 비트 내에서 아크탄젠트 계산을 합니다. 여기에서 사용하는 [ScaleFactor]의 값은 측정 정밀도에 좌우됩니다. 일반적으로 [ScaleFactor]는 $2^{-(32-[\text{비트 수}])}$ 를 설정합니다. 여기에서 비트 수는 사용자가 선택하는 정밀도입니다.

사용자가 변환을 16 비트 변환으로 원하는 경우(16 비트의 ADC 가 필요)에 [ScaleFactor]를 2^{-16} 으로 설정하게 되지만 이것은 1/65,536 즉 0.0000152587890625 가 됩니다.

사용자가 변환을 14 비트 변환으로 원하는 경우 [ScaleFactor]를 2^{-18} 으로 설정하지만 이것은 1/262,144 즉 0.000003814697265625 로 설정합니다.

Type = 7(확장 하드웨어 사인파 보간): 확장한 아크탄젠트의 소수 부분값을 모든 카운트값과 조합한 뒤에 중간 결과는 14 비트의 소수점 카운트를 가집니다. 이것은 AB 상 카운트의 1/16,384 단위(또는 엔코더 라인의 1/65,536)가 됩니다. 항목에 대하여 다양한 단위를 지정해도 됩니다.

보간된 카운트의 LSB 단위를 원하는 경우 [ScaleFactor]는 1.0 으로 설정되어야 합니다. AB 상 카운트(1/4 엔코더 라인)의 단위를 원하는 경우 [ScaleFactor]는 1/16384 로 설정되어야 합니다. 엔코더 라인 단위를 원하는 경우 [ScaleFactor]는 165536 으로 설정되어야 합니다. 많은 서보 루프 게인항이 선택된 스케일 계수에 반비례하여 변화한다는 점을 기억하십시오.

Type = 8(2 개의 소스 덧셈): [ScaleFactor]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 9(2 개의 소스 뺄셈): [ScaleFactor]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

Type = 10(트리거 신호에 따른 타임 베이스): [ScaleFactor]로 지정한 값은 이 일시적인 프리즈 상태로 있는 동안에는 사용되지 않지만 트리거 발생과 동시에 Type 1 또는 Type 3 항목으로 돌아가고 그곳에서 사용됩니다.

Type = 11(부동 소수점 값의 읽기): [ScaleFactor]로 지정한 값은 중간 결과의 고정 소수점 값으로 곱해집니다. 이것은 소스의 부동 소수점 레지스터값에 256*을 곱한 값에 해당합니다([index5] + 1). 사전 스케일링항의 [index5]를 기본값인 0 으로 설정한 경우에 1/256 의 [ScaleFactor]값은 입력값과 동일한 항목 출력값을 제공합니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): [ScaleFactor]로 지정한 값은 Type 1(해당 섹션 참조)과 같은 방법으로 사용됩니다.

EncTable[n].SinBias

설명: 엔코더 테이블 등록의 사인항 오프셋

범위: -32,768 .. 32,767

단위: 16 비트 ADC

기본값: 0

Type = 0(테이블의 종료): [SinBias]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): [SinBias]요소의 워드는 이 타입의 오프셋 기능으로는 사용하지 않습니다. 대신 워드는 [MaxDelta] 요소(해당 섹션 참조)의 유지에 사용됩니다.

Type = 2(2 워드의 읽기): [SinBias] 요소의 워드는 이 타입의 오프셋 기능으로는 사용하지 않습니다. 대신 워드는 [MaxDelta] 요소(해당 섹션 참조)의 유지에 사용됩니다.

Type = 3(소프트웨어 1/T 엔코더 확장): [SinBias]로 지정한 값은 이 타입에서는 사용하지 않습니다. 대신 워드는 [MaxDelta] 요소(해당 섹션 참조)의 유지에 사용합니다.

Type = 4(사인파 엔코더 아크탄젠트의 확장): [SinBias]의 값은 사인파 엔코더의 측정된 사인항에 더하는 오프셋을 지정합니다. 이것은 일치하는 A/D 컨버터가 [EncTable[i].Sin] 내에서 통지하는 값과 반대의 값을 설정해야 합니다. 측정항과 바이어스항은 모두 실제 A/D 컨버터가 다른 분해능을 가지는 경우에도 16 비트의 값으로 처리됩니다. 측정 데이터는 왼쪽 자리맞춤으로 설명되므로 14 비트의 ADC 를 사용하는 경우에 데이터는 16 비트 워드의 상위 14 비트로 배치됩니다.

Type = 5(4 바이트의 읽기): [SinBias] 요소의 워드는 이 타입의 오프셋 기능으로는 사용하지 않습니다. 대신 워드는 [MaxDelta] 요소(해당 섹션 참조)의 유지에 사용합니다.

Type = 6(리졸버 직접 변환): [SinBias]의 값은 리졸버의 측정된 사인항에 더하는 오프셋을 지정합니다. 이것은 일치하는 A/D 컨버터가 [EncTable[n].Sin] 내에서 통지하는 값과 반대의 값을 포함해야 합니다. 측정항과 바이어스항은 모두 실제 A/D 컨버터가 다른 분해능을 가지는 경우에도 16 비트의 값으로 처리됩니다. 측정 데이터는 왼쪽 자리맞춤으로 설명되므로 14 비트의 ADC 를 사용하는 경우에 데이터는 16 비트 워드의 상위 14 비트로 배치됩니다.

Type = 7(확장 하드웨어 사인파 보간): [SinBias]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 8(2 개의 소스 덧셈): [SinBias]요소의 워드는 이 타입의 오프셋 기능으로는 사용하지 않습니다. 대신 워드는 [MaxDelta] 요소(해당 섹션 참조)의 유지에 사용합니다.

Type = 9(2 개의 소스 뺄셈): [SinBias] 요소의 워드는 이 타입의 오프셋 기능으로는 사용하지 않습니다. 대신 워드는 [MaxDelta] 요소(해당 섹션 참조)의 유지에 사용합니다.

Type = 10(트리거 신호에 따른 타임 베이스): [SinBias]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 11(부동 소수점 값의 읽기): [SinBias]로 지정한 값은 이 타입에서는 사용하지 않습니다. 대신 워드는 [MaxDelta] 요소(해당 섹션 참조)의 유지에 사용합니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): [SinBias]로 지정한 값은 이 타입에서는 사용하지 않습니다. 대신 워드는 [MaxDelta] 요소(해당 섹션 참조)의 유지에 사용합니다.

EncTable[n].TanHalfPhi

설명: 엔코더 테이블 등록의 사인파/코사인파의 위상 오차 보정 계수

범위: -32,768 .. 32,767

단위: 탄젠트 각도 * 65,536

기본값: 0

[EncTable[i].TanHalfPhi]는 피드백에 사용하는 특정 타입으로 코사인파 대 사인파의 위상 오차 보정에 사용하는 적절한 보정 계수를 지정합니다. [TanHalfPhi] 요소는 2012 년 3/4 분기에 출시된 V1.5 펌웨어의 새 기능입니다. 기본값인 0 을 사용하면 모든 변환 메소드의 실행은 이전 버전의 펌웨어와 같아집니다.

Type = 0(테이블의 종료): [TanHalfPhi]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 1(단일 워드의 읽기): [TanHalfPhi]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 2(2 워드의 읽기): [TanHalfPhi]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 3(소프트웨어 1/T 엔코더 확장): [TanHalfPhi]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 4(사인파 엔코더 아크탄젠트의 확장): [TanHalfPhi]의 값은 사인파 엔코더의 사인파 코사인 신호 간의 페이즈 위치 오차 절반의 탄젠트를 지정합니다. 페이즈 위치 오차(phi)는 2 개 신호의 이상적인 90° 분리로 생긴 차이입니다. 차이가 90° 이상인 경우에는 양의 값이 되고 90° 미만인 경우에는 음의 값이 됩니다. 이 계수는 보정된 신호가 90°로 분리하도록 신호 값의 조정에 사용됩니다.

연산 시에 [TanHalfPhi]의 값은 65,536 으로 나뉘어 결과가 -0.5~+0.49999 범위로 설정됩니다. 이 값은 측정된 신호의 값을 조정하기 위해서 사용되고 아크탄젠트의 계산이 실행되기 전에 이상적인 90°로 분리하는 신호가 되도록 보정된 값을 생성합니다. [TanHalfPhi]의 +/-0.5 범위는 +/-26.5° 절반의 각도를 충족하기 위해서 신호 사이클의 +/-53° 페이즈 위치 오차가 됩니다. 앞의 계산에서는 [TanHalfPhi]의 기본값이 0, 탄젠트가 0.0 으로 페이즈의 보정이 실행되지 않는다는 점에 주의하십시오.

이 페이즈의 보정은 오프셋 보정 뒤에는 [SinBias] 및 [CosBias]를 사용하고 진폭 보정 전에는 [CoverSerror]를 사용하여 실행됩니다.

예: 만약 2 개의 신호가 이상적인 90°가 아니라 105°로 분리된 경우에 위상 오차 *Phi* 는 +15°가 되므로 절반의 각도는 +7.5°입니다. 절반의 각도인 탄젠트는 +0.13165 가 되기 때문에 [TanHalfPhi]는 8628(= 0.13165 * 65,536)로 설정해야 합니다.

Type = 5(4 바이트의 읽기): [TanHalfPhi]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 6(리졸버 직접 변환): [TanHalfPhi]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 7(확장 하드웨어 사인파 보간): [TanHalfPhi]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 8(2 개의 소스 덧셈): [TanHalfPhi]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 9(2 개의 소스 뺄셈): [TanHalfPhi]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 10(트리거 신호에 따른 타임 베이스): [TanHalfPhi]로 지정한 값은 이 타입에서는 사용하지 않습니다.

Type = 11(부동 소수점 값의 읽기): [TanHalfPhi]로 지정 한 값은 이 타입에서는 사용하지 않습니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): [TanHalfPhi]로 지정 한 값은 이 타입에서는 사용하지 않습니다.

EncTable[n].type

설명: 엔코더 테이블 등록의 변환 방법

범위: 0 .. 15

단위: 없음

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

[EncTable[n].type]은 테이블 항목으로 사용되는 변환 방법을 지정합니다. 0 은 테이블의 종료를 지정하고 그것보다 큰 번호의 항목은 그 항목이 실제로 변환을 지정하는 경우에도 실행되지 않습니다.

다음의 변환 타입(방법)이 지원됩니다.

0: 테이블의 종료

1: 단일 워드(32 비트)의 읽기

2: 2 워드(24 비트+8 비트)의 읽기

3: 소프트웨어에 따른 1/T 엔코더 확장

4: 사인파 엔코더의 아크탄젠트 확장

5: 4 바이트의 읽기

6: 리졸버에 따른 직접적인 변환

7: 하드웨어에 따른 확장 사인파 보간

8: 2 개의 소스 덧셈

9: 2 개의 소스 뺄셈

10: 트리거 신호에 따른 타임 베이스

11: 부동 소수점 값의 읽기

12: 단일 워드의 읽기, 에러 확인 포함

13+: (추후 사용하기 위해 예약)

특정 항목에 사용하는 변환 타입에 따라서 항목용의 다른 설정 요소가 다른 의미를 가지는 경우가 있습니다. 이는 각 요소를 설명하는 장에 기재됩니다.

각 변환 타입의 세부 내용을 다음과 같습니다.

Type = 0(테이블의 종료): 서보 사이클마다 Power PMAC 은 항목 0 부터 테이블 처리를 시작하여 그 뒤에 증가한 인덱스의 각 항목을 처리합니다. type = 0 의 항목을 검출하면 테이블 처리를 종료합니다. 그 항목의 다른 설정 요소 또는 큰 번호의 항목 설정 내용과 관계없이 그 이후의 처리는 실행하지 않습니다.

Type = 1(단일 워드의 읽기): 이 변환 타입인 경우에 Power PMAC 은 지정된 주소에서 단일 32 비트 레지스터를 읽습니다. 대부분의 경우 실제 데이터는 이 레지스터의 일부만으로 구성됩니다. 항목의 후속 처리는 실제 데이터가 올바르게 실행하는 남은 워드나 스케일에서 가능성이 있는 부정확한 데이터를 제거합니다.

이 변환 타입은 대부분의 데이터 형식에서 편리합니다. 이것은 페러럴 또는 시리얼의 애플루트 엔코더(절대 위치 검출기)에서의 위치, A/D 컨버터에서의 위치 또는 MLDT 타이머 레지스터에서의 위치에 사용할 수 있습니다. PMAC3 스타일 DSPGATE3 ASIC 은 1/T 보간의 직교 엔코더 위치, 아크탄젠트 보간의 사인파 엔코더 위치 또는 아크탄젠트 처리의 리졸버 위치 등 ASIC 의 하드웨어 내에서 데이터의 전처리에 사용할 수 있습니다.

Type = 2(2 워드의 읽기): 이 변환 타입인 경우에 Power PMAC 은 2 개의 개별 레지스터를 읽고 예비 32 비트 데이터값을 어셈블합니다. 첫번째 읽기로 32 비트 레지스터의 상위 24 비트에서 데이터를 취득하고 그 비트를 예비값의 하위 24 비트로 사용합니다. 2 번째 읽기로 32 비트 레지스터의 2 번째 최하위 바이트(비트 8~15)에서 데이터를 취득하고 그 비트를 예비값의 상위 8 비트로 사용합니다. 실제 데이터가 이 예비값의 일부만을 형성하는 경우에 항목 내의 후속 처리는 실제 데이터가 올바르게 실행하는 남은 워드나 스케일에서 가능성이 있는 부정확한 데이터를 제거합니다.

이 변환 타입은 24 비트 레지스터를 가지는 PMAC2 ASIC 에서 32 비트의 데이터를 취득하기에 편리합니다. ASIC 의 핀 I/O00~I/O31 에 입력된 데이터는 이 변환 타입에서 처리할 수 있습니다.

Type = 3(소프트웨어 1/T 엔코더 확장): 이 변환 타입인 경우에 Power PMAC 은 카운터 및 타이머의 데이터를 PMAC2 스타일 DSPGATE1 또는 DSPGATE2 ASIC 의 엔코더 채널에서 읽고 그 타이머의 데이터를 사용하여 소수 부분의 카운트값을 계산하고 AB 상 카운터의 모든 카운트값을 확장합니다. 결과 데이터는 직교 카운터의 1/512 분해능을 가집니다(ACC-24E3 및 같은 인터페이스 상에서 사용하는 PMAC3 스타일 DSPGATE3 ASIC 은 하드웨어 내에서 그 자체의 처리를 실행하기 위해 이 타입의 피드백인 경우에는 간단한 32 비트의 단일 레지스터 변환 타입을 대신 사용합니다).

Type = 4(사인파 엔코더 아크탄젠트의 확장): 이 변환 타입인 경우에 Power PMAC 은 AB 상 카운터와 관련된 A/D 컨버터 레지스터를 PMAC2 스타일 또는 PMAC3 스타일 ASIC 의 채널에서 읽고 엔코더의 사인 및 코사인 신호의 ADC 데이터를 사용하며 아크탄젠트의 계산을 통해서 카운터의 AB 상 카운트(제로 크로싱) 데이터 간의 보간을 실행합니다. PMAC2 스타일 ASIC 인 경우에 결과 데이터는 직교 수의 1/1,024 분해능을 가지지만 이것은 엔코더 라인의 1/4,096 이 됩니다. PMAC3 스타일 ASIC 인 경우에 결과 데이터는 직교 수의 1/16,384 분해능을 가지지만 이것은 엔코더 라인의 1/65,536 이 됩니다(ACC-24E3 및 같은 인터페이스 상에서 사용하는 PMAC3 스타일 DSPGATE3 ASIC 은 하드웨어 내에서 그 자체의 처리를 실행하기 때문에 DSPGATE3 ASIC 을 사용하는 이 타입의 피드백인 경우에는 간단한 32 비트의 단일 레지스터 변환 타입을 대신 사용한다는 점에 주의하십시오). 단, 이 소프트웨어 변환에는 추가적인 수정 항목이 있습니다.

Type = 5(4 바이트의 읽기): 이 변환 타입인 경우에 Power PMAC 은 4 개의 개별 레지스터를 읽고 예비 32 비트 데이터값을 어셈블합니다. 각 레지스터를 읽으면 32 비트 레지스터의 2 번째 최하위 바이트(비트 8~15)에서 데이터를 취득하고 해당 4 바이트를 32 비트값으로 어셈블합니다. 최초 레지스터의 데이터는 이 값의 최하위 바이트를 형성하고 마지막 레지스터의 데이터는 최상위 바이트를 형성합니다. 실제 데이터가 이 예비값의 일부만을 형성하는 경우에 항목 내의 후속 처리는 실제 데이터가 올바르게 실행하는 남은 워드나 스케일에서 가능성이 있는 부정확한 데이터를 제거합니다.

이 변환 타입은 8 비트 레지스터를 가지는 IOGATE ASIC 을 사용하여 ACC-14E 등의 액세서리에서 위치 데이터를 취득하기에 편리합니다.

Type = 6(리졸버 직접 변환): 이 변환 타입인 경우에 Power PMAC 은 PMAC2 스타일 DSPGATE1 또는 DSPGATE2 ASIC 의 채널에서 A/D 컨버터 레지스터를 읽습니다. 또 동시에 여기 회로의 값을 읽고 리졸버의 사인 및 코사인 신호의 ADC 데이터를 사용하여 아크탄젠트의 계산을 통해서 리졸버의 위치를 계산합니다. (ACC-24E3 및 같은 인터페이스 상에서 사용하는 PMAC3 스타일 DSPGATE3 ASIC 은 하드웨어 내에서 그 자체의 처리를 실행하기 때문에 DSPGATE3 ASIC 을 사용하는 이 타입의 피드백인 경우에는 간단한 32 비트의 단일 레지스터변환 타입을 대신 사용한다는 점에 주의하십시오.)

Type = 7(확장 하드웨어 사인파 보간): 이 변환 타입인 경우에 Power PMAC 은 PMAC3 스타일 DSPGATE3 IC 의 채널에서 보간 수의 값을 읽고 그것을 같은 채널의 (하드웨어 계산을 한)아크탄젠트 요소에서 확장된 값과 조합합니다. 결과 데이터는 직교 수의 1/16,384 분해능을 가지지만 이것은 엔코더 라인의 1/65,536 이 됩니다.

Type = 8(2 개의 소스 덧셈): 이 변환 타입인 경우에 Power PMAC 은 지정된 레지스터에서 2 개의 32 비트 소스 레지스터를 읽고 각각의 값을 함께 더합니다. 이 변환 타입은 2 개의 센서를 함께 평균화하는 경우에 특히 편리합니다.

Type = 9(2 개의 소스 뺄셈): 이 변환 타입인 경우에 Power PMAC 은 지정된 레지스터에서 2 개의 32 비트 소스 레지스터를 읽고 최초값에서 2 번째값을 뺍니다. 이 변환 타입은 병렬 모터의 기울기 제어를 실행하는 경우에 특히 편리합니다.

Type = 10(트리거 신호에 따른 타임 베이스): 이 변환 타입은 type=3 의 소프트웨어 1/T 엔코더 확장(PMAC2 스타일 IC 를 사용하는 경우)의 일시적인 변형 또는 type=1 의 단일 레지스터 읽기(PMAC3 스타일 IC 경유로 인크리멘탈 엔코더를 사용하는 경우)의 일시적인 변형입니다. type=10 인 경우에는 입력 변화에 관계없이 출력은 프리즈됩니다(변화 없음). [index1]을 2 또는 3 으로 설정하면 대기 상태가 되고 지정된 트리거가 발생하면 항목은 자동으로 type=1 또는 type=3 으로 각각 복귀합니다.

Type = 11(부동 소수점 값의 읽기): 이 변환 타입인 경우에 Power PMAC 은 단정도(32 비트) 또는 배정도(64 비트)의 부동 소수점 레지스터를 ([index6]의 설정에 따라서)읽고 [index5]를 사용하여 사전 스케일링을 실행하고 그것을 중간 고정 소수점의 형식으로 변환한 뒤 type=1 인 경우의 고정 소수점 소스와 같이 처리합니다. 현재 이 방법에 대한 유효한 소스 레지스터는 모터의 [IqCmd] 단정도의 서보 출력 레지스터, 모터의 배정도 [DesPos]의 명령 위치 레지스터, [PhasePos]의 전류(commutation)각 레지스터 및 사용자 공유 메모리 버퍼의 부동 소수점 레지스터입니다. 이 방법으로는 가상 모터에 대한 시뮬레이트 완료 서보 루프 및 오픈 루프의 다이렉트 마이크로 스테핑이 가능합니다.

Type = 12(단일 워드의 읽기, 에러 확인 포함): 이 변환 타입인 경우에 Power PMAC 은 위치 데이터에 지정된 주소에서 하나의 32 비트 레지스터를 읽습니다. 이것은 가능성이 있는 에러 비트의 2 번째 레지스터를 읽고 해당 값을 [index6]의 값으로 마스크 하여 지정된 비트만을 에러로 처리합니다. 어떤 서보 사이클에서 에러 상태를 찾은 경우에 그 사이클에서 읽어낸 위치의 값은 사용되지 않고 대신 제일 마지막으로 유효한 위치의 값에서 추정된 데이터가 계산된 값으로 교체됩니다.

Gate1[i].(PMAC2 스타일 서보 IC)저장 데이터 구조체 요소(V1.6 2014.03)

이 장은 PMAC2 스타일 "DSPGATE1" 서보 IC 의 저장 데이터 구조체 요소를 설명합니다.

Power PMAC 스크립트 환경에서는 "**Gate1[i]**"구조체명 대신에 이 IC 가 탑재된 액세스서리의 이름을 사용할 수 있습니다. 지원하는 액세스서리의 이름은 다음과 같습니다. :

- **Acc24E2[i]**. ACC-24E2 UMAC PWM 축 인터페이스 보드
- **Acc24E2A[i]**. ACC-24E2A UMAC 아날로그 축 인터페이스 보드
- **Acc24E2S[i]**. ACC-24E2S UMAC 스테핑 모터 축 인터페이스 보드
- **Acc51E[i]**. ACC-51E UMAC 사인파 엔코더 보간 보드
- **Acc24C2[i]**. ACC-24C2 Compact UMAC PWM 축 인터페이스 보드
- **Acc24C2A[i]**. ACC-24C2A Compact UMAC 아날로그 축 인터페이스 보드
- **Acc51C[i]**. ACC-51C Compact UMAC 사인파 엔코더 보간 보드

이 이름은 단순히 **Gate1[i]** 이름의 "별칭"입니다. **Gate1[i]** 이름 또는 별칭이 사용되어도 인덱스값 *i* 는 동일합니다. 각 DSPGATE1 IC 에는 고유의 인덱스값이 사용됩니다. C 프로그램에서는 **Gate1[i]**의 이름을 사용해야 합니다. C 환경에서는 별칭을 사용할 수 없습니다. 인덱스값 *i* 는 현재 4~19(0~3 은 기존의 어떤 하드웨어에서도 사용되지 않습니다)의 범위에서 액세스서리 보드의 하드웨어 주소 DIP 스위치 설정에 따라서 결정됩니다.

주의: Turbo PMAC 어플리케이션에서 변환할 때 Power PMAC 시스템의 Gate1 IC 번호는 Turbo PMAC 시스템의 동일한 하드웨어 주소 설정의 IC 번호 "m"의 2 배입니다. 예를 들어 Turbo PMAC 시스템의 "서보 IC 2"는 Power PMAC 시스템에서는 **Gate1[4]**가 됩니다. 그리고 Turbo PMAC 시스템의 "서보 IC 6"은 Power PMAC 시스템에서는 **Gate1[12]**가 됩니다.

Turbo PMAC 의 1 개 간격의 IC 번호 "m*"에 대해서는 동등한 Power PMAC IC 번호는 $2m* + 1$ 이 됩니다. 예를 들어 Turbo PMAC 시스템의 "서보 IC 2*"는 Power PMAC 시스템에서는 **Gate1[5]**가 됩니다. 그리고 Turbo PMAC 시스템의 "서보 IC 6*"는 Power PMAC 시스템에서는 **Gate1[13]**이 됩니다.

Gate1[i].멀티채널 설정 요소

이 장에서는 PMAC2 스타일 "DSPGATE1" 서보 IC 의 "멀티채널" 저장 데이터 구조체 요소를 설명합니다. 이 요소는 ASIC 의 4 개 모든 채널에 영향을 줍니다.

Gate1[i].AdcStrobe 게이트 1 의 ADC 스트로브

설명:	서보 IC 의 ADC 스트로브 워드
범위:	\$000000..\$FFFFFF
단위:	시리얼 데이터열(MSB 를 선두로 위상 클록의 상승 엣지로 시작)
기본값:	\$3FFFFFF
기준 I 변수명:	I7m06

Gate1[i].AdcStrobe 는 IC 상의 모든 머신 인터페이스 채널의 ADC 스트로브 신호를 제어합니다. 이 변수에 따라서 설정된 24 비트 워드는 **ADC_STROB** 라인상에 연속적으로 시프트 출력됩니다. 이는 **MSB** 를 선두로 위상 클록의 상승에서 시작되는 **ADC_CLK** 사이클당 1 비트씩 출력됩니다.

Gate1[i].AdcStrobe 의 비트 0(LSB)은 ADC 의 수치 데이터에 선행하는 "헤더" 정보가 서보 IC 에 필요한지 여부를 결정하는 제어 비트입니다. 비트 0 이 0 인 경우에 헤더 정보는 필요 없습니다. 또한 이 비트에 의한 **low** 출력은 위상 클록의 다음 상승 엣지까지 유지됩니다.

Gate1[i].AdcStrobe 의 비트 0 이 1 인 경우에 4비트 이내의 헤더 정보는 ADC 의 반송 시리얼 데이터열로 받을 수 있습니다(Delta Tau 의 Geo 파워 블록 앰프의 ADC 와 유사). 이 비트는 "롤 오버" 되어서 서보 IC 의 ADC 레지스터의 비트 0-3 위치에 옵니다. 또한 수치 데이터는 ADC 레지스터 비트 23 의 위치에 **MSB** 가 오게 됩니다. 4 비트 미만의 헤더가 필요한 경우에 **Gate1[i].AdcStrobe** 의 최초 비트(여러 개 가능)를 0 으로 설정하여 스트로브 워드의 시작을 지연해야 합니다. 특히(4-n)개의 헤더 비트가 필요한 경우에 **Gate1[i].AdcStrobe** 의 최초 n비트는 0 으로 설정해야 합니다. 이 설정에서는 비트 0 이 시프트 출력된 뒤에 **ADC_STROB** 출력은 **low** 가 되고 **low** 를 유지합니다.

"1"인 최초 비트는 "변환 시작" 신호로서 보통 사용되는 **ADC_STROB** 출력의 상승 엣지를 생성합니다. 일부 A/D 변환기(ADC)는 변환에 해당 상승 엣지만 필요합니다. 그 밖에 변환하는 동안 신호를 **high** 로 해야 합니다. 일부 어플리케이션에서는 **ADC_STROB** 출력 중간의 비트는 다른 정보를 송신하기 위해서 사용할 수 있습니다.

\$3FFFFFF 의 **Gate1[i].AdcStrobe** 기본값은 대부분의 Delta Tau "Geo"파워 블록 앰프에 적합합니다. \$FFFFFFE 의 값은 Delta Tau 제품인 "Quad 앰프"나 대부분의 서드 파티 제품의 다이렉트 PWM 앰프 및 ACC-28B ADC 의 용도에 적합합니다. 상세한 내용은 개별 앰프 또는 ADC 설명서를 참조하십시오.

Gate1[i].ClockCtrl 게이트 1 의 클록 제어

설명: 서보 IC 풀 워드의 클록 제어 요소

범위: \$0..\$FFFFFF

단위: 비트 필드

기본값: \$3008D2

Gate1[i].ClockCtrl 은 IC 의 클록 신호 생성의 멀티채널 설정에서 구성되는 풀 워드 요소입니다. 이는 다음의 부분 워드 요소로 구성됩니다.

부분 워드 요소	스크립트 비트	C 비트	기능
ServoClockDiv	23 -20	31 -28	서보 클록 주파수 분주 제어
PhaseClockDiv	19 -16	27 -24	위상 클록 주파수 분주 제어
(예약)	15 -14	23 -22	(추후 사용하기 위해 예약)
PhaseServoDir	13 -12	21 -20	위상 및 서보 클록 방향 제어
HardwareClockCtrl	11 -00	19 -08	하드웨어 클록 주파수 분주 제어
(공백)	..	07 -00	(존재하는 하드웨어가 없습니다.)

사용자는 요소의 설정과 해당 설정 확인을 위해 스크립트 환경에서 보통 이러한 부분 워드 요소에 액세스합니다. Power PMAC 은 효율화를 위해서 풀 워드 요소로 저장 및 복원합니다.

C 프로그램에서는 사용자는 풀 워드 요소만 액세스해야 합니다. 그리고 설명 물리 레지스터에 24 비트만 있다 하더라도 32 비트 워드로 액세스해야 합니다. 이 24 비트는 32 비트 워드의 최상위에 배치됩니다.

풀 워드 요소의 값이 스크립트 환경 속에서 참조되는 경우에 부분 워드 요소 값이 10 진수로 통지되어도 Power PMAC 은 16 진수 형식으로 값을 반환합니다.

해당 요소의 기능에 대한 자세한 설명은 각 부분 워드 요소의 항목을 참조하십시오.

Gate1[i].DacStrobe 게이트 1 의 DAC 스트로브

설명: 서보 IC 의 DAC 스트로브 워드

범위: \$000000..\$FFFFFF

단위: 시리얼 데이터열(MSB 를 선두로 위상 클록의 상승 엣지로 시작)

기본값: \$7FFFC0(18 비트 DAC 에는)

기준 I 변수명: I7m05

Gate1[i].DacStrobe 는 선택된 PMAC2 스타일 "DSPGATE1"서보 IC의 머신 인터페이스 채널의 DAC 스트로브 신호를 제어합니다. **Gate1[i].DacStrobe** 에 따라서 설정된 24 비트 워드는 DAC_STROB 라인에 연속적으로 시프트 출력됩니다. MSB 를 선두로 위상 클록의 상승 엣지에서 시작하는 DAC_CLK 사이클당 1 비트씩 출력됩니다. LSB 의 값은 다음 위상 클록 사이클까지 유지됩니다.

대표적인 n-비트 DAC 에 대해서 스트로브 라인은 n-1 클록 사이클 동안 high 로 유지됩니다. 그 결과 이 변수의 일반 설정은 다음과 같습니다.

- 18 비트 DAC:\$7FFFC0(17 클록 사이클 동안 high)
- 16 비트 DAC:\$7FFF00(15 클록 사이클 동안 high)
- 12 비트 DAC:\$7FF000(11 클록 사이클 동안 high)

\$7FFFC0 의 **Gate1[i].DacStrobe** 기본값은 Delta Tau 제품의 18 비트 DAC 에 적합합니다. 만약 다른 DAC 를 사용하지 않는다면 **Gate1[i].DacStrobe** 의 기본값을 변경하면 안 됩니다.

Gate1[i].HardwareClockCtrl 게이트 1 의 하드웨어 클록 제어

설명: 서보 IC 의 하드웨어 클록 주파수 제어

범위: 0 ..4095

단위: 개별 클록 분주기

Gate1[i].HardwareClockCtrl = 엔코더 SCLK 분주기

+ 8 * PFM_CLK 분주기

+ 64 * DAC_CLK 분주기

+ 512 * ADC_CLK 분주기

따라서

엔코더 SCLK 주파수 = $39.3216 \text{ MHz} / (2^{\text{엔코더 SCLK 분주기}})$

PFM_CLK 주파수 = $39.3216 \text{ MHz} / (2^{\text{PFM_CLK 분주기}})$

DAC_CLK 주파수 = $39.3216 \text{ MHz} / (2^{\text{DAC_CLK 분주기}})$

ADC_CLK 주파수 = $39.3216 \text{ MHz} / (2^{\text{ADC_CLK 분주기}})$

기본값: $2258 = 2 + (8 * 2) + (64 * 3) + (512 * 4)$

엔코더 SCLK 주파수 = $39.3216 \text{ MHz} / (2^2) = 9.8304 \text{ MHz}$

PFM_CLK 주파수 = $39.3216 \text{ MHz} / (2^2) = 9.8304 \text{ MHz}$

DAC_CLK 주파수 = $39.3216 \text{ MHz} / (2^3) = 4.9152 \text{ MHz}$

ADC_CLK 주파수 = $39.3216 \text{ MHz} / (2^4) = 2.4576 \text{ MHz}$

기준 I 변수명: I7m03

Gate1[i].HardwareClockCtrl 은 PMAC2 스타일 "DSPGATE1" 서보 IC 의 4 개 머신 인터페이스 채널에 대해서 4 개의 하드웨어 클록 주파수(SCLK, PFM_CLK, DAC_CLK 및 ADC_CLK)를 제어합니다. 이는 4 개의 독립된 3 비트 제어에서 구성되는 12 비트 변수로 1 개가 각각의 클록을 제어합니다. 이 클록 주파수는 최초 39.3216MHz 주파수에서 2 의 거듭제곱, 2^N 에 따라서 1 부터 128 까($N = 0 \sim 7$) 분주할 수 있습니다. 그 결과 각 클록의 설정 가능한 주파수는 다음과 같습니다.

주파수	제수	$1/2^N$ 의 분주기 N	주파수	제수	$1/2^N$ 의 분주기 N
39.3216MHz	1	0	2.4576MHz	16	4
19.6608MHz	2	1	1.2288MHz	32	5
9.8304MHz	4	2	611.44 kHz	64	6
4.9152MHz	8	3	305.72 kHz	128	7

기본값에서 **Gate1[j].HardwareClockCtrl** 의 변경이 필요한 Power PMAC 어플리케이션은 거의 없습니다.

SCLK: 엔코더 샘플 클럭 신호 SCLK 는 서보 IC 의 디지털 하드웨어가 엔코더와 플래그 입력을 어느 정도의 빈도로 조사하는지 제어합니다. 서보 IC 는 많아도 SCLK 사이클당 1 카운트를 받을 수 있습니다. 그래서 SCLK 주파수는 절대적인 최대 엔코더 카운트 주파수가 됩니다. SCLK 는 또한 디지털 지연 필터를 통해서 엔코더와 플래그의 신호 전파를 제어합니다. SCLK 주파수가 낮을수록 필터로 컷할 수 있는 노이즈 펄스는 많아집니다. SCLK 주파수는 가능한 한 최고 속도로 엔코더 카운트를 받을 수 있는 최소값으로 설정해야 합니다.

PFM_CLK: 펄스 주파수 변조 클럭 PFM_CLK 는 일반적으로 스테핑 모터 드라이브에 사용되는 PFM 회로를 제어합니다. 설정 가능한 최고 펄스 주파수는 PFM_CLK 주파수의 1/4 입니다. PFM_CLK 주파수는 필요한 최고 주파수로 펄스를 생성할 수 있는 최소값에 최적화하여 설정해야 합니다.

DAC_CLK: DAC_CLK 는 D/A 컨버터(DAC)의 시리얼 데이터 주파수를 제어합니다. Delta Tau 에서 제공되는 액세서리상의 컨버터를 사용하는 경우에 DAC_CLK 설정은 기본값으로 해야 합니다.

ADC_CLK: ADC_CLK 는 ADC 의 시리얼 데이터 주파수를 제어합니다. Delta Tau 에서 제공되는 액세서리상의 컨버터를 사용하는 경우에 ADC_CLK 설정은 기본값으로 해야 합니다.

Gate1[j].HardwareClockCtrl 의 임의값으로 설정된 클럭 주파수를 구하려면 다음의 순서로 진행합니다.

- 1.) 512 로 **Gate1[j].HardwareClockCtrl** 을 나누고 가장 가까운 정수로 소수점 아래를 버립니다. 이 값 N1 은 ADC_CLK 분주기입니다.
- 2.) 512 와 N1 을 곱하고 **Gate1[j].HardwareClockCtrl** 에서 이 곱을 빼서 **Gate1[j].HardwareClockCtrl'**을 얻습니다. **Gate1[j].HardwareClockCtrl'**를 64 로 나누고 가장 가까운 정수로 버립니다. 이 값 N2 가 DAC_CLK 분주기입니다.
- 3.) 64 와 N2 를 곱하고 **Gate1[j].HardwareClockCtrl'**에서 이 곱을 빼서 **Gate1[j].HardwareClockCtrl''**를 얻습니다. **Gate1[j].HardwareClockCtrl''**를 8 로 나누고 가장 가까운 정수로 버립니다. 이 값 N3 이 PFM_CLK 분주기입니다.
- 4.) 8 과 N3 을 곱하고 **Gate1[j].HardwareClockCtrl''**에서 이 곱을 뺍니다. 최종값 N4 가 SCLK 분주기입니다.

Gate1[j].HardwareClockCtrl 은 폴 워드 요소 **Gate1[j].ClockCtrl** 의 비트 0-11 을 구성합니다. (C 에서는 32 비트 요소의 비트 8-19)

예

어플리케이션의 최대 엔코더 카운트 주파수는 800kHz 입니다. 그 결과 1.2288MHz 의 SCLK 주파수가 선택됩니다. 500kHz 까지 펄스열을 생성해야 하므로 2.4576MHz 의 PFM_CLK 주파수를 선택합니다. Delta Tau 에서 제공되는 기본값의 시리얼 DAC 및 ADC 를 사용하는 경우에 4.9152MHz 의 기본값 DAC_CLK 주파수 및 2.4576MHz 의 기본값 ADC_CLK 주파수를 선택합니다. 표에서

SCLK 분주기 N: 5
 PFM_CLK 분주기 N: 4
 DAC_CLK 분주기 N: 3
 ADC_CLK 분주기 N: 4

$$\text{Gate1}[\text{J}].\text{HardwareClockCtrl} = 5 + (8 * 4) + (64 * 3) + (512 * 4) \\ = 5 + 32 + 192 + 2048 = 2277$$

Gate1[J].HardwareClockCtrl 이 3429 로 설정되었다고 가정합니다. 이것은 어떤 클럭 주파수를 설정 할까요?

$$\begin{aligned} N1 &= \text{INT}(3429/512) = 6 & \text{ADC\_CLK} &= 611.44 \text{ kHz} \\ \text{Gate1}[\text{J}].\text{HardwareClockCtrl}' &= 3429 - (512*6) = 357 \\ N2 &= \text{INT}(357/64) = 5 & \text{DAC\_CLK} &= 1.2288 \text{ MHz} \\ \text{Gate1}[\text{J}].\text{HardwareClockCtrl}'' &= 357 - (64*5) = 37 \\ N3 &= \text{INT}(37/8) = 4 & \text{PFM\_CLK} &= 2.4576 \text{ MHz} \\ N4 &= 37 - (8*4) = 5 & \text{SCLK} &= 1.2288 \text{ MHz} \end{aligned}$$

Gate1[J].PhaseClockDiv 게이트 1 의 위상 클럭 주파수 분주

설명: 서보 IC 의 위상 클럭 주파수 분주 제어

범위: 0 ..15

단위: 위상 클럭 주파수 = 최대 위상 주파수/(Gate1[J].PhaseClockDiv+1)

기본값: 0
 위상 클럭 주파수 = 9.0346 kHz/ 1 = 9.0346 kHz
 (Gate1[J].PwmPeriod 의 기본값에서)

기존 I 변수명: I7m01

Gate1[J].PhaseClockDiv 는 **Gate1[J].PwmPeriod** 와 함께 각 PMAC2 스타일 "DSPGATE1" 서보 IC 의 내부에서 생성되는 위상 클럭 주파수를 결정합니다. 그러나 **Gate1[J].PhaseServoDir** 에서 그 자체의 위상 클럭을 사용하도록 지정된 서보 또는 MACRO IC 만 생성하는 위상 클럭 신호를 출력합니다. 위상 클럭의 각 사이클에서 모터 위상 전류(commutation) 및 디지털 전류 루프 알고리즘이 지정된 모터에서 실행됩니다.

특히 **Gate1[J].PhaseClockDiv** 는 위상 클럭 주파수가 "최대 위상" 클럭에서 얼마나 분주될지 제어합니다. "최대 위상" 주파수는 **Gate1[J].PwmPeriod** 에 따라서 설정됩니다. 위상 클럭 주파수는 (Gate1[J].PhaseClockDiv+1)에 의해 분주된 "최대 위상" 클럭 주파수와 같아집니다. **Gate1[J].PhaseClockDiv** 는 0~15 의 범위, 주파수 분주는 1~16 의 계수로 실행됩니다. **Gate1[J].PhaseClockDiv** 의 식은 다음과 같습니다.

$$Gate1[i].PhaseClockDiv = \frac{MaxPhaseFreq(kHz)}{PhaseFreq(kHz)} - 1$$

최대 위상 주파수와 위상 클럭 주파수의 비율은 정수여야 합니다.

주의: 위상 클럭 주파수가 너무 높게 설정되면 통신과 같은 우선 순위가 낮은 태스크는 처리 시간이 없어집니다. 백그라운드 태스크의 처리 시간이 완전히 부족하면 워치독 타이머가 작동하여 보드를 셧다운합니다. 일반적인 보드의 리셋으로 워치독 타이머도 트립하지 않고 통신도 정상적으로 동작하는 상태로 회복하지 않는 경우에 USB 메모리 스틱을 설치한 상태로 전원 셋업을 실행하여 보드를 재초기화해야 합니다. 이는 기본설정으로 복원하므로 통신도 가능해지고 **Gate1[j].PwmPeriod** 와 **Gate1[j].PhaseClockDiv** 를 지원 가능한 값으로 수정할 수 있습니다.

Gate1[j].PhaseClockDiv 는 풀 워드 요소 **Gate1[j].ClockCtrl** 의 비트 16-19 를 구성합니다. (C에서는 32 비트 요소의 비트 24-27)

예

Gate1[j].PwmPeriod 에 따라서 설정된 20kHz 의 최대 위상 클럭 주파수로 6.67kHz 의 위상 클럭 주파수를 구하려면 최대 위상과 위상 간의 비율은 3 이 됩니다.

$$Gate1[j].PhaseClockDiv = (20 / 6.67) - 1 = 3 - 1 = 2$$

Gate1[j].PhaseServoDir 게이트 1 의 위상/서보 클럭 방향

설명: 서보 IC 의 위상/서보 클럭 방향

범위: 0 ..3

단위: 비트 필드

기본값: IC 와 시스템에 좌우됨

기준 I 변수명: I7m07

Gate1[j].PhaseServoDir 은 IC 자체 내부에서 생성된 위상 및 서보 클럭 신호를 사용할지 또는 외부 소스의 위상 및 서보 클럭 신호를 사용할지 여부를 제어합니다. IC 내부에서 생성되는 해당 클럭은 **Gate1[j].PwmPeriod**, **Gate1[j].PhaseClockDiv** 및 **Gate1[i].ServoClockDiv** 에 따라서 제어됩니다.

모든 Power PMAC 시스템은 시스템에 서보 및 위상 클럭 신호 중 1 개의 소스만 존재합니다 - 서보 IC 또는 MACRO IC 중 1 개 또는 시스템 외의 소스 중 하나입니다.

Gate1[j].PhaseServoDir 은 2 비트값입니다. 비트 0 을 0 으로 설정하면 IC 가 그 자체의 위상 클럭 신호를 사용하여 출력합니다. 1 로 설정하면 외부 입력의 위상 클럭 신호를 사용합니다. 비트 1 를 0 으로 설정하면 IC 가 그 자체의 서보 클럭 신호를 사용하여 출력합니다. 1 로 설정하면 외부 입력의 서보 클럭 신호를 사용합니다. 이로 인해 **Gate1[j].PhaseServoDir** 은 아래에 있는 4 개의 값을 가집니다.

- **Gate1[i].PhaseServoDir = 0:** 내부 위상 클럭, 내부 서보 클럭
- **Gate1[i].PhaseServoDir = 1:** 외부 위상 클럭, 내부 서보 클럭
- **Gate1[i].PhaseServoDir = 2:** 내부 위상 클럭, 외부 서보 클럭
- **Gate1[i].PhaseServoDir = 3:** 외부 위상 클럭, 외부 서보 클럭

보통 사용에서는 모두 **Gate1[i].PhaseServoDir** 은 0(많아도 1 개의 IC) 혹은 3(다른 모든 IC)으로 설정됩니다. 보통 재초기화 시에 Power PMAC 은 시스템의 위상과 서보 클럭 신호의 소스로 어떤 IC 를 사용할지 자동으로 선택하고 이 변수값을 설정합니다. 대부분의 어플리케이션에서는 이 설정을 변경하지 않습니다.

Gate1[i].PhaseServoDir 은 풀 워드 요소 **Gate1[i].ClockCtrl** 의 비트 12-13 을 구성합니다. (C에서는 32 비트 요소의 비트 20-21)

Gate1[i].PwmCtrl 게이트 1 의 PWM 제어 워드

설명: 서보 IC 의 PWM 제어 풀 워드 요소

범위: \$0..\$FFFFFF

단위: 비트 필드

기본값: \$197F0F

Gate1[i].PwmCtrl 은 IC 의 PWM 신호 생성에 대한 멀티채널 설정에서 구성되는 풀 워드 요소입니다. 이는 다음의 부분 워드 요소로 구성됩니다.

부분 워드 요소	스크립트 비트	C 비트	기능
PwmPeriod	23 -08	31 -16	최대 위상/PWM 주기의 제어
PwmDeadTime	07 -00	15 -08	PWM 데드 타임/PFM 펄스 폭의 제어
(공백)	..	07 -00	(존재하는 하드웨어가 없습니다.)

사용자는 요소의 설정과 해당 설정 확인을 위해 스크립트 환경에서 보통 이러한 부분 워드 요소에 액세스합니다. Power PMAC 은 효율화를 위해서 풀 워드 요소로 저장 및 복원합니다.

C 프로그램에서는 사용자는 풀 워드 요소만 액세스해야 합니다. 그리고 물리 레지스터에 24 비트만 있다 하더라도 32 비트 워드로 액세스해야 합니다. 이 24 비트는 32 비트 워드의 최상위에 배치됩니다.

풀 워드 요소의 값이 스크립트 환경 속에서 참조되는 경우에 부분 워드 요소 값이 10 진수로 통지되어도 Power PMAC 은 16 진수 형식으로 값을 반환합니다.

해당 요소의 기능에 대한 자세한 설명은 각 부분 워드 요소의 항목을 참조하십시오.

Gate1[i].PwmDeadTime 게이트 1 의 PWM 데드 타임/PFM 펄스 폭

설명:	서보 IC 의 PWM 데드 타임/PFM 펄스 폭 제어
범위:	0 ..255
단위:	16*PWMCLK 사이클, PFMCLK 사이클 PWM 데드 타임 = $[16 / \text{PWMCLK 주파수(MHz)}] * \text{Gate1[i].PwmDeadTime}$ $= 0.135\mu\text{sec} * \text{Gate1[i].PwmDeadTime}$ PFM 펄스 폭 = $[1 / \text{PFMCLK 주파수(MHz)}] * \text{Gate1[i].PwmDeadTime}$ $= \text{PFMCLK 주기}(\mu\text{sec}) * \text{Gate1[i].PwmDeadTime}$
기본값:	15 PWM 데드 타임 = $0.135\mu\text{sec} * 15 = 2.03\mu\text{sec}$ PFM 펄스 폭 = $[1 / 9.8304\text{MHz}] * 15 = 1.526 \mu\text{sec}$ (기본값 Gate1[i].HardwareClockCtrl 일 때)

기존 I 변수명: I7m04

Gate1[i].PwmDeadTime 은 PMAC2 스타일 "DSPGATE1" 서보 IC 의 머신 인터페이스 채널의 자동 PWM 생성에서 Top On-time 과 Bottom On-time 사이의 데드 타임 기간을 제어합니다. 이것은 또한 **Gate1[i].HardwareClockCtrl** 과 연계하여 서보 IC 의 머신 인터페이스 채널의 자동 펄스 주파수 변조(PFM) 펄스 폭도 제어합니다.

PWM 데드 타임은 16PWM_CLK 사이클 단위로 지정됩니다. 이는 Top 신호의 하강과 Bottom 신호의 상승 사이 그리고 그 반대의 지연 시간입니다. 이것은 0.135 μsec 단위로 데드 타임을 지정할 수 있음을 뜻합니다. PWM 데드 타임 기능으로서의 **Gate1[i].PwmDeadTime** 의 식은 다음과 같습니다.

$$\text{Gate1[i].PwmDeadTime} = \frac{\text{DeadTime}(\mu\text{sec})}{0.135\mu\text{sec}}$$



주의

대부분의 다이렉트 PWM 드라이브는 안전 기능으로 최저 데드 타임 기간이 필요합니다. 단, 그것은 이 IC 의 PWM_CLK 신호에 비동기로 더욱 낮은 주파수의 클록으로 실행합니다. 이는 드라이브의 최소 데드 타임 설정을 신뢰하여 유효한 명령 분해능을 감소시키고 나아가 출력에 대한 맥놀이 현상을 초래한다는 것을 의미합니다. (해당 요소가 보다 작은 데드 타임에 지정된 경우에 발생합니다.)

Gate1[i].PwmDeadTime 에 따라서 정의되도록 PFM 펄스 폭은 PFM_CLK 사이클로 지정됩니다. PFM 펄스 폭 및 PFM_CLK 주파수의 함수로서 **Gate1[i].PwmDeadTime** 의 식은 다음과 같습니다.

$$\text{Gate1[i].PwmDeadTime} = \text{PfmClkFreq(MHz)} * \text{PfmPulseWidth}(\mu\text{sec}) - 1$$

DSPGATE1의 PFM 펄스 생성에서는 펄스 간의 최소 오프 타임은 펄스 폭과 같아집니다. 즉 최대 PFM 출력 주파수를 구하는 식은 다음과 같습니다.

$$PfmMaxFreq(MHz) = \frac{PfmClkFreq(MHz)}{2 * [Gate1[i].PwmDeadTime + 1]}$$

Gate1[i].PwmDeadTime은 풀 워드 요소 **Gate1[i].PwmCtrl**의 비트 0-7을 구성합니다. (C에서는 32비트 요소의 비트 8-15)

예/:

약 1μsec의 PWM 데드 타임을 구하는 방법은 다음과 같습니다.

$$Gate1[i].PwmDeadTime \cong 1 \mu\text{sec} / 0.135 \mu\text{sec} \cong 7$$

2.4576MHz의 PFM_CLK 주파수로 0.4μsec의 펄스 폭을 구하는 방법은 다음과 같습니다.

$$Gate1[i].PwmDeadTime \cong 2.4576 \text{ MHz} * 0.4 \mu\text{sec} \cong 1$$

Gate1[i].PwmPeriod 게이트 1의 최대 위상/PWM 주기 제어

설명: 서보 IC의 최대 위상/PWM 주기를 제어

범위: 0 ..32767

단위: MaxPhase 주파수 = 117,964.8 kHz / [2***Gate1[i].PwmPeriod**+3]
PWM 주파수 = 117,964.8 kHz / [4 * **Gate1[i].PwmPeriod**+6]

기본값: 6527
MaxPhase 주파수 = 117,964.8 / 13057 = 9.0346 kHz
PWM 주파수 = 117,964.8 / 26114 = 4.5173 kHz

기존 I 변수명: I7m00

Gate1[i].PwmPeriod는 PMAC2 스타일 "DSPGATE1" 서보 IC 상 4개의 머신 인터페이스 채널의 내부 "최대 위상" 클럭 주파수 및 PWM 주파수를 제어합니다. 서보 IC의 내부에서 생성되는 위상 및 서보 클럭은 최대 위상 클럭으로 만들어집니다.

서보 IC가 PMAC 시스템의 위상 및 서보 클럭을 생성하기 위해서 사용되는 경우에(**Gate1[i].PhaseServoDir** 변수로 인해 설정되는) 이 변수는 이 시스템 클럭의 주파수 제어 역할을 수행합니다.

Gate1[i].PwmPeriod는 PWM 업 다운 카운터의 한계 설정에 의해 해당 주파수를 제어합니다. 이 카운터에서는 117,964.8kHz(117.9648MHz)의 PWMCLK 주파수로 증감됩니다.

실제의 위상 클럭 주파수는 **Gate1[j].PhaseClockDiv** 의 설정에 따라서 최대 위상 클럭에서 분주됩니다. 위상 클럭의 하강 엣지에서 PMAC 이 서보 IC 에 연결된 시리얼 A/D 변환기(위상 전류 측정용)를 샘플링하고 필요한 위상 전류(commutation) 및 디지털 전류 루프 알고리즘을 시작시키도록 프로세서에 인터럽트를 가합니다. 위상 및 전류 루프 알고리즘이 사용되지 않아도 서보 클럭이 위상 클럭에서 생성되기 때문에 최대 위상과 위상 클럭 주파수는 중요합니다.

PWM 주파수는 다이렉트 PWM 명령으로 4 개의 머신 인터페이스 채널에 연결된 앰프의 실제 스위칭 주파수를 결정합니다. 다이렉트 PWM 명령 신호 형식이 사용되는 경우에 이 변수는 특히 중요합니다.

포화되지 않고 PWM 명령 레지스터에 작성할 수 있는 최대값은 양의 한계에서는 **Gate1[j].PwmPeriod + 1**, 음의 한계에서는 **- Gate1[j].PwmPeriod - 2**입니다. 보통 각 모터의 "PWM 스케일 팩터" **Motor[j].PwmSf** 는 최대 PWM 명령의 크기를 결정하고 **Gate1[j].PwmPeriod + 10%**로 설정됩니다.

보통 시스템 위상 클럭 주파수를 제어하지 않는 서보 IC 의 **Gate1[j].PwmPeriod** 는 제어하는 서보 IC 의 값과 같은 값으로 설정됩니다. 다른 PWM 주파수를 다른 서보 IC 의 PWM 출력으로 사용하는 경우에 **Gate1[j].PwmPeriod** 는 아래와 같이 설정됩니다.

$$\frac{2 * PwmFreq(kHz)}{PhaseFreq} = \{\text{정수}\}$$

이에 따라 해당 채널의 PWM 하드웨어와 시스템 위상 클럭에 따라서 동작하는 소프트웨어 알고리즘과의 동기를 유지합니다. 예를 들어 위상 주파수가 10kHz 인 경우에 다른 서보 IC 의 PWM 주파수는 5, 10, 15, 20(등)kHz 가 가능해집니다.

필요한 PWM 주파수에 **Gate1[j].PwmPeriod** 를 설정할 때는 다음의 식을 사용합니다.

$$Gate1[i].PwmPeriod = \frac{117,964.8(kHz)}{4 * PwmFreq(kHz)} - 1 (\text{소수점 이하 버림})$$

필요한 "최대 위상" 클럭 주파수에 **Gate1[j].PwmPeriod** 를 설정할 때는 다음의 식을 사용합니다.

$$Gate1[i].PwmPeriod = \frac{117,964.8(kHz)}{2 * MaxPhaseFreq(kHz)} - 1 (\text{소수점 이하 버림})$$

Gate1[j].PwmPeriod 는 풀 워드 요소 **Gate1[j].PwmCtrl** 의 비트 8-23 을 구성합니다. (C 에서는 32 비트 요소의 비트 16-31)

예/:

10kHz 의 PWM 주파수를 설정하고 그것에 따라서 20kHz 의 최대 위상 클럭 주파수를 설정하는 방법은 다음과 같습니다.

$$Gate1[j].PwmPeriod = (117,964.8 \text{ kHz} / [4 * 10 \text{ kHz}]) - 1 = 2948$$

7.5kHz 의 PWM 주파수를 설정하고 그것에 따라서 15kHz 의 최대 위상 클럭 주파수를 설정하는 방법은 다음과 같습니다.

$$\text{Gate1}[j].\text{PwmPeriod} = (117,964.8 \text{ kHz} / [4 \times 7.5 \text{ kHz}]) - 1 = 3931$$

Gate1[j].ServoClockDiv 게이트 1 의 서보 클럭 주파수 분주

설명: 서보 IC 의 서보 클럭 주파수 분주 제어

범위: 0 ..15

단위: 서보 클럭 주파수 = 위상 클럭 주파수 /
(Gate1[j].ServoClockDiv +1)

기본값: 3
서보 클럭 주파수 = 9.0346kHz / (3+1) = 2.2587 kHz
(Gate[j].PwmPeriod 와 Gate1[j].PhaseClockDiv 의 기본값으로)

기존 I 변수명: I7m02

Gate1[j].ServoClockDiv 는 Gate1[j].PhaseClockDiv 와 Gate1[j].PwmPeriod 와 함께 각 PMAC2 스타일 "DSPGATE1" 서보 IC 의 내부에서 생성되는 서보 클럭 주파수를 결정합니다. 그러나 Gate1[j].PhaseServoDir 에서 그 자체의 서보 클럭을 사용하도록 지정된 서보 IC 이지만 그 생성되는 서보 클럭 신호를 출력합니다. 서보 클럭의 각 사이클에서 Power PMAC 은 유효한 모터의 명령 위치를 갱신하고 앰프 또는 전류(commutation) 알고리즘의 명령을 연산하는 서보 알고리즘을 실행합니다.

특히 Gate1[j].ServoClockDiv 는 Gate1[j].PhaseClockDiv 와 Gate1[j].PwmPeriod 에 따라서 설정되는 위상 클럭 주파수로 서보 클럭 주파수가 얼마나 분주될지 제어합니다. 서보 클럭 주파수는 (Gate1[j].ServoClockDiv+1)로 분주된 위상 클럭 주파수와 같아집니다. Gate1[j].ServoClockDiv 는 0~15 의 범위, 따라서 주파수 분주는 1~16 의 계수로 실행됩니다. Gate1[j].ServoClockDiv 의 식은 다음과 같습니다.

$$\text{Gate1}[i].\text{ServoClockDiv} = \frac{\text{PhaseFreq(kHz)}}{\text{ServoFreq(kHz)}} - 1$$

위상 클럭 주파수와 서보 클럭 주파수의 비율은 정수여야 합니다.

적절한 속도로 궤적을 실행하기 위해서 Sys.ServoPeriod 는 서보 클럭의 사이클 시간을 궤적 생성 소프트웨어에 전달하기 위해서 올바르게 설정되어야 합니다. Sys.ServoPeriod 의 식은 다음과 같습니다.

$$\text{Sys.ServoPeriod} = \frac{1}{\text{ServoFreq(kHz)}}$$

PMAC2 스타일 IC의 서보 클럭 주파수를 결정하는 변수로 치환하면 **Sys.ServoPeriod**의 계산식은 다음과 같습니다.

$$\text{Sys.ServoPeriod} = \frac{(2 * \text{PwmPeriod} + 3)(\text{PhaseClockDiv} + 1)(\text{ServoClockDiv} + 1)}{117,964.8}$$

기본값의 서보 클럭 주파수에서는 Power PMAC의 보간 루틴이 적절한 서보 갱신 시간을 사용하기 위해서 **Sys.ServoPeriod**는 0.442718로 설정해야 합니다.

주의: 서보 클럭 주파수가 너무 높게 설정되면 통신과 같은 우선 순위가 낮은 태스크는 시간이 없어집니다. 백그라운드 태스크의 처리 시간이 완전히 부족하면 워치독 타이머가 작동하여 보드를 셧다운합니다. 일반적인 보드의 리셋으로 워치독 타이머도 작동하지 않고 통신도 정상적으로 동작하는 상태로 회복하지 않는 경우에 USB 메모리 스틱을 설치한 상태로 전원 셋업을 실행하여 보드를 재초기화해야 합니다. 이것은 기본설정으로 복원하므로 통신도 가능해지고 **Gate1[i].PwmPeriod**, **Gate1[i].PhaseClockDiv** 및 **Gate1[i].ServoClockDiv**를 지원 가능한 값으로 수정할 수 있습니다.

Gate1[i].ServoClockDiv는 풀 워드 요소 **Gate1[i].ClockCtrl**의 비트 20-23을 구성합니다. (C에서는 32비트 요소의 비트 28-31)

예/

Gate1[i].PwmPeriod와 **Gate1[i].PhaseClockDiv**에 따라서 설정된 6.67kHz의 위상 클럭 주파수 그리고 3.33kHz의 서보 클럭 주파수를 구하면:

$$\text{Gate1[i].ServoClockDiv} = (6.67 / 3.33) - 1 = 2 - 1 = 1$$

Gate1[i].채널 고유의 설정 요소

이 장에서는 PMAC2 스타일 "DSPGATE1" 서보 IC 채널에 고유 저장 데이터 구조체 요소를 설명합니다. IC의 4개의 각 채널은 이 변수로 독립적으로 설정할 수 있습니다.

이 설정 요소 중에서 Power PMAC의 채널 인덱스값 **Chan[j]**는 Turbo PMAC 소프트웨어 및 하드웨어 설명서의 채널 번호 "*n*"보다 1 개 작은 값입니다. Power PMAC 채널 인덱스값 "*j*"는 0~3의 범위에서 Turbo PMAC 채널 번호 "*n*"과 하드웨어 채널 번호 1~4에 각각 대응합니다.

Gate1[i].Chan[j].CaptCtrl 게이트 1 채널의 위치 포착 제어

설명: 서보 IC 채널의 위치 포착 제어

범위: 0 ..15

단위: 비트 필드

기본값: 1

기존 I 변수명: I7mn2

Gate1[i].Chan[j].CaptCtrl은 PMAC2 스타일 서보 IC 채널에 대해서 어떤 입력 신호 또는 어떤 신호의 조합인지 그리고 어떤 극성이 엔코더 카운터의 하드웨어 위치 포착 트리거가 될지 결정합니다. 플래그 입력이 사용되는 경우(원점, 리미트 또는 사용자 플래그)에 **Gate1[i].Chan[j].CaptFlagSel**에서 어떤 플래그를 사용할지 결정합니다.

매우 정확한 하드웨어 위치 포착 기능이 필요한 경우에 이 채널의 위치 루프 피드백과 플래그를 사용하는 모터가 원점 서치 동작 또는 다른 트리거가 있는 동작을 하기 위해 이 변수의 적절한 설정이 필요합니다. **Motor[x].CaptureMode**가 0, 1 또는 3으로 설정되어 하드웨어 트리거를 선택하는 경우(하드웨어 위치 포착에 상관없이)에 이 변수를 올바르게 설정해야 합니다.

Gate1[i].Chan[j].CaptCtrl은 다음의 설정을 사용할 수 있습니다.

- **Gate1[i].Chan[j].CaptCtrl = 0:** 연속 또는 홀 포착
- **Gate1[i].Chan[j].CaptCtrl = 1:** 인덱스(CHCn) high 로 포착
- **Gate1[i].Chan[j].CaptCtrl = 2:** 플래그 n high 로 포착
- **Gate1[i].Chan[j].CaptCtrl = 3:** (인덱스 high 및 플래그 n high)로 포착
- **Gate1[i].Chan[j].CaptCtrl = 4:** 연속 또는 홀 포착
- **Gate1[i].Chan[j].CaptCtrl = 5:** 인덱스(CHCn) low 로 포착
- **Gate1[i].Chan[j].CaptCtrl = 6:** 플래그 n high 로 포착
- **Gate1[i].Chan[j].CaptCtrl = 7:** (인덱스 low 및 플래그 n high)로 포착
- **Gate1[i].Chan[j].CaptCtrl = 8:** 연속 또는 홀 포착
- **Gate1[i].Chan[j].CaptCtrl = 9:** 인덱스(CHCn) high 로 포착
- **Gate1[i].Chan[j].CaptCtrl = 10:** 플래그 n low 로 포착

- **Gate1[i].Chan[j].CaptCtrl = 11:** (인덱스 high 및 플래그 n low)로 포착
- **Gate1[i].Chan[j].CaptCtrl = 12:** 연속 또는 홀 포착
- **Gate1[i].Chan[j].CaptCtrl = 13:** 인덱스(CHCn) low 로 포착
- **Gate1[i].Chan[j].CaptCtrl = 14:** 플래그 n low 로 포착
- **Gate1[i].Chan[j].CaptCtrl = 15:** (인덱스 low 및 플래그 n low)

엔코더와 같은 채널 번호의 플래그 및 인덱스 입력만 그 엔코더 위치의 하드웨어 포착으로 사용할 수 있다는 점에 주의하십시오. 이는 하드웨어 포착 기능을 원점 서치 동작에 사용하려면 **Motor[x].pEncCtrl** 은 **Motor[x].pEnc** 의 위치 루프 피드백에 사용하는 엔코더와 같은 채널 번호의 플래그를 사용해야 함을 뜻합니다.

이 변수의 비트 0 과 1 이 함께 0 으로 설정되어(값이 0, 4, 8 또는 12) 같은 채널의 **Gate1[i].Chan[j].GatedIndexSel** 이 0 인 경우에 트리거가 대기 상태가 되면 포착 트리거가 즉시 발생합니다. 같은 채널의 **Gate1[i].Chan[j].GatedIndexSel** 이 1 인 경우에 트리거가 U, V 또는 W 의 "홀" 입력 플래그 중 하나의 다음 신호 변화로 발생합니다.

위치 포착 레지스터 **Gate1[i].Chan[j].HomeCapt** 를 읽으면 트리거는 대기 상태가 됩니다. 이는 상태 요소 **Gate1[i].Chan[j].PosCapt** 를 0 으로 클리어하고 트리거가 대기 상태로 있음을 나타냅니다. 하지만 다음 트리거는 발생하지 않습니다. Power PMAC 의 트리거가 있는 동작 기능에서는 이 읽기는 동작 처음에 자동으로 실행됩니다. 그 결과 트리거는 항상 대기 상태가 됩니다. 그 다음 지정된 입력 라인이 지정된 상태임을 서보 IC 하드웨어가 판단하면 즉시 트리거가 발생합니다. - 이는 레벨 트리거로 엣지 트리거가 아닙니다. 신호 변화는 필요하지 않습니다.

Gate1[i].Chan[j].CaptCtrl 은 풀 워드 요소 **Gate1[i].Chan[j].Ctrl** 의 비트 4-7 을 구성합니다. (C에서는 32 비트 요소의 비트 12-15)

Gate1[i].Chan[j].CaptFlagSel 게이트 1 채널의 위치 포착 플래그 선택

설명: 서보 IC 의 채널 위치 포착 플래그 선택

범위: 0 ..3

단위: 열거 형식

기본값: 0

기존 I 변수명: I7mn3

Gate1[i].Chan[j].CaptFlagSel 은 4 개의 "플래그" 입력 중 어느 것이 PMAC2 스타일 서보 IC 채널의 엔코더 카운터의 하드웨어 위치 포착에 사용될지 여부를 결정합니다(1 개가 사용되는 경우). **Gate1[i].Chan[j].CaptCtrl** 은 플래그의 사용 여부, 플래그 중 어떤 극성이 트리거를 발생시킬지 결정합니다. **Gate1[i].Chan[j].CaptFlagSel** 의 유효값과 각각 선택되는 플래그는 다음과 같습니다.

- **Gate1[i].Chan[j].CaptFlagSel = 0:** HOMEn(홈 플래그 n)
- **Gate1[i].Chan[j].CaptFlagSel = 1:** PLIMn(양의 한계 플래그 n)
- **Gate1[i].Chan[j].CaptFlagSel = 2:** MLIMn(음의 한계 플래그 n)
- **Gate1[i].Chan[j].CaptFlagSel = 3:** USERn(사용자 플래그 n)

Gate1[i].Chan[j].CaptFlagSel 은 채널의 홈 플래그를 사용하는 원점 서치 동작을 위해서 보통 0 으로 설정됩니다. 프로빙이나 위치 등록과 같은 다른 용도에 하드웨어 위치 포착 기능을 사용하는 경우에 사용자 플래그를 선택하기 위해서 원점 서치 동작 다음에 3 으로 설정합니다. PLIMn 또는 MLIMn 의 오버트래블 한계 플래그로 포착하는 경우에 **Motor[x].pLimits** 를 0 으로 일시적으로 설정하고 일반적인 운전 정지 기능을 무효가 되게 할 수 있습니다.

Gate1[i].Chan[j].CaptFlagSel 은 풀 워드 요소 **Gate1[i].Chan[j].Ctrl** 의 비트 8-9 를 구성합니다. (C에서는 32 비트 요소의 비트 16-17)

Gate1[i].Chan[j].Ctrl 게이트 1 채널의 제어 풀 워드

설명: 서보 IC 채널의 풀 워드 제어 요소

범위: \$0..\$FFFFFF

단위: 비트 필드

기본값: \$C00017

Gate1[i].Chan[j].Ctrl 은 IC 채널의 신호 인터페이스를 위한 설정 요소에서 구성되는 풀 워드 요소입니다. 이는 다음의 부분 워드 요소로 구성됩니다.

부분 워드 요소	스크립트 비트	C 비트	기능
OutputMode	23 -22	31 -30	출력 모드 선택
OutputPol	21 -20	29 -28	출력 극성 제어
PfmDirPol	19	27	PFM 방향 출력 극성 제어
OneOverTEna	18	26	하드웨어 1/T 유효
IndexGateState	17 -16	25 -24	게이트된 인덱스 포착 상태 제어
GatedIndexSel	15	23	게이트된 인덱스 포착 선택
AmpEna	14	22	앰프 유효 출력 상태
Equ1Ena	13	21	비교 회로 엔코더 선택
EquWrite	12 -11	20 -19	비교 출력 작용 상태
PosClear	10	18	엔코더 카운터 리셋 제어
CaptFlagSel	09 -08	17 -16	위치 포착 플래그 선택
CaptCtrl	07 -04	15 -12	위치 포착 제어
EncCtrl	03 -00	11 -08	엔코더 디코드 제어
(공백)	..	07 -00	(존재하는 하드웨어가 없습니다.)

굵은 글씨로 표시된 요소는 설정 요소로 저장되며 이 장에 각각 기재합니다. 굵은 글씨가 아닌 것은 저장되지 않는 요소로 비저장 설정 요소 장에 기재합니다.

사용자는 요소의 설정과 해당 설정 확인을 위해 스크립트 환경에서 보통 이러한 부분 워드 요소에 액세스합니다. Power PMAC 은 효율화를 위해서 풀 워드 요소로 저장 및 복원합니다.

C 프로그램에서는 사용자는 풀 워드 요소만 액세스해야 합니다. 그리고 물리 레지스터에 24 비트만 있다 하더라도 32 비트 워드로 액세스해야 합니다. 이 24 비트는 32 비트 워드의 최상위에 배치됩니다.

풀 워드 요소의 값이 스크립트 환경 속에서 참조되는 경우에 부분 워드 요소 값이 10 진수로 통지되어도 Power PMAC 은 16 진수 형식으로 값을 반환합니다.

해당 요소의 기능에 대한 자세한 설명은 각 부분 워드 요소의 항목을 참조하십시오.

Gate1[i].Chan[j].EncCtrl 게이트 1 채널의 엔코더 디코드 제어

설명: 서보 IC 채널의 엔코더 디코드 제어

범위: 0 ..15

단위: 없음

기본값: 7

기존 I 변수명: I7mn0

Gate1[i].Chan[j].EncCtrl 은 PMAC2 스타일 서보 IC 채널의 엔코더 입력 신호가 어떻게 카운트에 디코드 될지 제어합니다. 따라서 여기에는 "카운트"의 부호와 크기를 정의합니다. 다음의 설정이 입력 신호를 디코드하기 위해서 사용됩니다.

- **Gate1[i].Chan[j].EncCtrl = 0:** 펄스와 방향 CW
- **Gate1[i].Chan[j].EncCtrl = 1:** x1 AB 상 CW 디코드
- **Gate1[i].Chan[j].EncCtrl = 2:** x2 AB 상 CW 디코드
- **Gate1[i].Chan[j].EncCtrl = 3:** x4 AB 상 CW 디코드
- **Gate1[i].Chan[j].EncCtrl = 4:** 펄스와 방향 CCW 디코드
- **Gate1[i].Chan[j].EncCtrl = 5:** x1 AB 상 CCW 디코드
- **Gate1[i].Chan[j].EncCtrl = 6:** x2 AB 상 CCW 디코드
- **Gate1[i].Chan[j].EncCtrl = 7:** x4 AB 상 CCW 디코드
- **Gate1[i].Chan[j].EncCtrl = 8:** 내부 펄스와 방향
- **Gate1[i].Chan[j].EncCtrl = 9:** 사용 안 함
- **Gate1[i].Chan[j].EncCtrl = 10:** 사용 안 함
- **Gate1[i].Chan[j].EncCtrl = 11:** x6 홀 형식 CW 디코드
- **Gate1[i].Chan[j].EncCtrl = 12:** MLDT 펄스 타이머 제어
(내부 펄스는 타이머를 리셋합니다.
외부 펄스는 타이머를 래치합니다)
- **Gate1[i].Chan[j].EncCtrl = 13:** 사용 안 함
- **Gate1[i].Chan[j].EncCtrl = 14:** 사용 안 함
- **Gate1[i].Chan[j].EncCtrl = 15:** x6 홀 형식 CCW 디코드

AB 상 디코드 모드 중 어느 것이든 서보 IC 는 각각 약 50%의 듀티 사이클로 서로 1 사이클의 약 1/4 위상 편차가 있는 CHAn 과 CHBn 2 개의 입력 파형이 필요합니다. "1 체배"(x1) 디코드는 1 사이클당 1 카운트합니다. x2 는 1 사이클당 2 카운트합니다. 그리고 x4 는 1 사이클당 4 카운트합니다. 대부분의 경우에 최고 분해능을 얻기 위해서 x4 디코드를 선택합니다.

"시계 방향"(CW)과 "시계 반대 방향"(CCW)의 옵션은 단순히 어느 방향으로 카운트 업을 할지 제어합니다. 잘못된 방향 검출인 경우에는 다른 옵션(예를 들어 7에서 3으로 또는 반대로)으로 변경하십시오.



경고

정상적으로 동작하는 모터의 피드백 엔코더의 디코드 방향 검출을 변경하는 것은 다른 변경이 없는 상태에서는 불안정한 정귀환이 되어 위험한 폭주 상태를 야기합니다. 안정된 부귀환을 회복하려면 동일하게 출력 극성도 변경해야 합니다.

펄스와 방향 디코드 모드에서는 서보 IC 는 CHAn 의 펄스열과 CHBn 의 방향(부호) 신호가 필요합니다. 신호가 단일 방향인 경우에 CHBn 라인은 high 상태로 풀업을 시키거나 또는 high 나 low 상태로 결선할 수 있습니다.

Gate1[j].Chan[j].EncCtrl 이 8 로 설정되는 경우에 디코더는 채널 자체의 펄스 주파수 변조(PFM) 출력 회로에 따라서 생성된 펄스와 방향 신호를 입력합니다. 이는 오픈 루프의 스테핑 모터 시스템을 구동하는 경우에 Power PMAC 이 가공의 클로즈드 루프 생성이 가능해집니다. 여기에는 점퍼 또는 케이블은 필요하지 않습니다. 연결은 완전히 서보 IC 내에서 이루어집니다. 카운터 극성은 자동으로 PFM 출력 극성에 적합합니다.

Gate1[j].Chan[j].EncCtrl 이 11 또는 15 로 설정되는 경우에 채널은 각각 약 50%의 듀티 사이클로 서로 1 사이클의 약 1/3(120 e) 위상 편차가 있는 CHAn, CHBn 과 CHCn 3 개의 홀 센서 형식 입력이 필요합니다. 디코드 회로는 각 신호의 각 엣지에서 1 카운트를 생성하고 신호 사이클당 6 카운트를 연습니다("x6 디코드"인 경우). 11 과 15 의 차이는 신호 중 어느 방향이 카운터를 카운트 업 시키느냐입니다.

Gate1[j].Chan[j].EncCtrl 이 12 로 설정되는 경우에 타이머 회로는 TemposonicsTM 과 같은 자왜식 선형 변위 변환기(MLDT)를 읽기 위해서 설정됩니다. 이 방식으로는 PFM 회로가 PULSEn 상에서 센서에 여기 펄스를 전송하면 타이머는 클리어됩니다. 또한 여기 펄스가 CHAn 으로 수신되면 타이머는 메모리 매핑된 레지스터에 래치됩니다.

Gate1[j].Chan[j].EncCtrl 은 풀 워드 요소 **Gate1[i].Chan[j].Ctrl** 의 비트 0-3 을 구성합니다. (C에서는 32 비트 요소의 비트 8-11)

Gate1[j].Chan[j].Equ1Ena 게이트 1 채널의 비교 엔코더 선택

설명: 서보 IC 채널의 비교 회로 엔코더 선택

범위: 0 ..1

단위: 논리값

기본값: 0

기존 I 변수명: I7mn1

Gate1[i].Chan[j].Equ1Ena 은 어느 채널의 엔코더 카운터가 PMAC2 스타일의 서보 IC 채널의 위치 비교 회로에 결합되는지 제어합니다. 다음의 설정이 가능합니다.

- **Gate1[i].Chan[j].Equ1Ena** = 0: 위치 비교 기능에 그 채널 자체의 엔코더 카운터를 사용
- **Gate1[i].Chan[j].Equ1Ena** = 1: 위치 비교 기능에 최초 채널(1)의 엔코더 카운터를 사용

Gate1[i].Chan[j].Equ1Ena 이 0 으로 설정되면 채널의 위치 비교 레지스터는 채널 자체의 엔코더 카운터와 결합되어 위치 비교 신호는 그 채널의 EQU 출력에만 출력됩니다.

Gate1[i].Chan[j].Equ1Ena 이 1 로 설정되면 채널의 위치 비교 레지스터는 서보 IC 의 최초 엔코더 카운터(채널 1)에 결합되어 위치 비교 신호는 이 채널 자체의 EQU 출력과 서보 IC 채널 1 의 결합된 EQU 출력의 양쪽으로 출력됩니다. 채널 1 에는 논리합으로 출력됩니다.

Gate1[i].Chan[0].Equ1Ena 는 유효한 기능으로 실행하지 않습니다. 따라서 항상 1 입니다. 0 으로는 설정할 수 없습니다.

Gate1[i].Chan[j].Equ1Ena 은 풀 워드 요소 **Gate1[i].Chan[j].Ctrl** 의 비트 13 을 구성합니다. (C 에서는 32 비트 요소의 비트 21)

Gate1[i].Chan[j].GatedIndexSel 게이트 1 채널의 게이트 인덱스 포착 선택

설명: 서보 IC 채널의 게이트된 인덱스 포착 선택

범위: 0 ..1

단위: 논리값

기본값: 0

기존 I 변수명: I7mn4

Gate1[i].Chan[j].GatedIndexSel 은 원시 엔코더 인덱스 채널 입력 또는 AB 상 상태에 따라서 게이트된 입력 중 어느 것을 PMAC2 스타일 서보 IC 채널의 엔코더 위치 포착에 사용할지 여부를 제어합니다. 다음의 설정이 가능합니다.

- **Gate1[i].Chan[j].GatedIndexSel** = 0: 엔코더 위치 포착에 게이트되지 않은 원시 인덱스를 사용
- **Gate1[i].Chan[j].GatedIndexSel** = 1: 엔코더 위치 포착에 AB 상 채널에 따라서 게이트된 인덱스를 사용

Gate1[i].Chan[j].GatedIndexSel 이 0 으로 설정되면 엔코더의 인덱스 채널 입력 (CHCn)은 위치 포착 회로에 직접 전달됩니다.

Gate1[j].Chan[j].GatedIndexSel 이 1로 설정되면 엔코더의 인덱스 채널 입력(CHCn)은 위치 포착 회로에 전달되기 전에 엔코더 n의 AB 상 신호와 논리 결합됩니다("게이트화됨"). 그 의도는 정확히 90° 위상 폭의 "게이트된 인덱스" 신호를 얻는 것입니다. 이는 더욱 정확하고 보다 재현성이 있는 포착을 제공하고 포착 기능의 사용에 의해 매 회전의 정확한 카운트 수를 아주 간단하게 얻을 수 있습니다.

게이트된 인덱스 포착이 확실히 작동하기 위해서 인덱스 펄스는 엔코더의 AB 상 펄스의 "high-high" 또는 "low-low" 1개의 상태 동안에만 안정적으로 유지되어야 합니다. **Gate1[j].Chan[j].IndexGateState** 는 이 2개의 가능성 중 어느 것을 사용할지 선택합니다.

주의: **Gate1[j].Chan[j].GatedIndexSel** 이 1로 설정되어도 **Gate1[j].Chan[j].CaptCtrl** 의 비트 0과 비트 1이 0으로 설정되는 경우에 인덱스는 위치 포착으로 사용되지 않습니다. 그 다음에 엔코더 위치는 채널의 U, V 혹은 W 플래그 입력 중 하나의 최초 엣지로 포착됩니다. 이 경우에 **Gate1[j].Chan[j].HallState**(채널 상태 워드의 비트 0, 1과 2)로 어느 홀 상태 엣지로 포착될지 판단합니다.

Gate1[j].Chan[j].GatedIndexSel 은 풀 워드 요소 **Gate1[j].Chan[j].Ctrl** 의 비트 15를 구성합니다. (C에서는 32비트 요소의 비트 23)

Gate1[j].Chan[j].IndexGateState 게이트 1 채널의 게이트 인덱스 포착 상태

설명: 서보 IC 채널의 게이트된 인덱스 포착 상태 제어

범위: 0..3

단위: 비트 필드

기본값: 0

기준 I 변수명: I7mn5

Gate1[j].Chan[j].IndexGateState 는 채널의 엔코더 인덱스 신호 2개의 기능을 제어하는 2비트의 변수입니다.

PMAC2 스타일 서보 IC의 "게이트된 인덱스" 기능을 더욱 정확한 위치 포착에 사용하는 경우(**Gate1[j].Chan[j].GatedIndexSel** = 1)에 **Gate1[j].Chan[j].IndexGateState** 의 비트 0은 서보 IC의 엔코더 입력에 보낸 원시 인덱스 채널 신호가 "high-high" AB 상 상태(비트 0 = 0) 혹은 "low-low" AB 상 상태(비트 0 = 1) 중 하나로 위치 포착 신호에 전달될지 여부를 지정합니다.

Gate1[j].Chan[j].IndexGateState 의 비트 1은 Yaskawa 인크리멘탈 엔코더처럼 서보 IC가 AB 상 상태에 따른 제 3의 채널의 인덱스 펄스와 3개의 홀 센서 스타일의 전류(commutation) 상태를 "다중화 해제"할지 여부를 제어합니다. 비트 1이 0으로 설정되는 경우에 이 다중화 해제 기능은 실행되지 않습니다. 그리고 엔코더의 "C" 채널 신호는 단순 인덱스로 사용됩니다. 비트 1이 1로 설정되는 경우에 서보 IC는 3번째 채널 신호를 4개의 다른 값으로 전개합니다. 그 1개는 4개의 가능한 AB 상 상태의 각각에 대응합니다. 다중화 해제된 홀 전류(commutation) 상태는 전원 투입 시의 위상 위치를 얻기 위해서 사용할 수 있습니다.

주의: 전원 투입 직후에 Yaskawa 엔코더는 자동으로 완전한 AB 상 사이클의 형식으로 AB 상 출력을 전후에 순환 출력합니다. 컨트롤러가 동작을 시작하기 전에 모든 홀 센서의 전류(commutation) 상태가 이용 가능하다는 것을 보증하기 위해서입니다. 그러나 Power PMAC 과 동시에 엔코더에 전원이 공급되면 Servo IC 가 이 신호를 받을 준비가 되기 전에 이 출력이 시작됩니다. 만약 준비가 되었을 때 서보 IC 가 이런 모든 상태를 받아들이지 못한다면 채널의 Status 워드 **Gate1[i].Chan[j].Status** 의 "다중화 해제 무효"의 비트 3 을 1 로 설정합니다. 이 기능을 사용하기 위해서 소프트웨어로 컨트롤되는 릴레이를 통해서 엔코더에 전원을 공급하는 것을 권장합니다. 이에 따라 전원 투입 시의 위상 읽기 전에 모든 상태의 유효한 읽기가 가능하다는 것이 보증되기 때문입니다.

Gate1[i].Chan[j].IndexGateState 는 아래와 같이 설정 가능합니다.

- **Gate1[i].Chan[j].IndexGateState** = 0: "high-high" AB 상 상태에서 게이트 인덱스(GI = A & B & C), 다중화 해제 없음
- **Gate1[i].Chan[j].IndexGateState** = 1: "low-low" AB 상 상태에서 게이트 인덱스(GI = A/ & B/ & C), 다중화 해제 없음
- **Gate1[i].Chan[j].IndexGateState** = 2 또는 3: 제 3 의 채널의 홀과 인덱스의 다중화 해제, 게이트 처리는 관계없음

Gate1[i].Chan[j].IndexGateState 는 풀 워드 요소 **Gate1[i].Chan[j].Ctrl** 의 비트 16-17 을 구성합니다. (C 에서는 32 비트 요소의 비트 24-25)

Gate1[i].Chan[j].OneOverTEna 게이트 1 채널의 하드웨어 1/T 유효

설명: 서보 IC 채널의 하드웨어 1/T 유효

범위: 0 ..1

단위: 논리값

기본값: 0

기준 I 변수명: I7mn9

Gate1[i].Chan[j].OneOverTEna 은 PMAC2 스타일 서보 IC 채널의 "하드웨어-1/T" 기능을 유효로 할지 여부를 제어합니다. **Gate1[i].Chan[j].OneOverTEna** 이 0 의 기본값인 경우에 하드웨어 -1/T 기능은 무효가 되고 엔코더 변환 테이블의 변환 방법 3 의 기본설정에 따라서 계산되는 "소프트웨어 -1/T" 확장의 사용을 가능하게 합니다. **Gate1[i].Chan[j].OneOverTEna** 이 1 로 설정되는 경우에 하드웨어 -1/T 기능이 유효가 됩니다. 그리고 소프트웨어 -1/T 는 사용할 수 없습니다.

하드웨어 -1/T 기능이 유효가 되면 IC 는 SCLK(엔코더 샘플 클럭) 사이클마다 타이머에 따른 새로운 소수점 카운트 위치 추정을 연산합니다. 이는 소수점 카운트 데이터가 하드웨어 포착과 비교 기능에 사용되어 그것의 분해능을 향상시킵니다. IC 가 ACC-51 고분해능 아날로그 엔코더 보간 보드로 사용될 때 이는 아주 유용합니다. 그러나 채널의 타이머 레지스터는 소수점 카운트 위치 데이터로 전환되므로 이것이 유효가 되는 경우에 변환 테이블의 기존 소프트웨어 -1/T 방식은 잘 기능하지 않습니다.

PMAC2 스타일 서보 IC 의 하드웨어 1/T 보간의 사용은 Power PMAC 펌웨어의 서보 위치에 대한 지원은 현재 없습니다.

Gate1[j].Chan[j].OneOverTEna 은 풀 워드 요소 **Gate1[j].Chan[j].Ctrl** 의 비트 18 을 구성합니다. (C 에서는 32 비트 요소의 비트 26)

Gate1[j].Chan[j].OutputMode 게이트 1 채널의 출력 모드 선택

설명: 서보 IC 채널의 출력 모드 선택

범위: 0 ..3

단위: 비트 필드

기본값: 0

기존 I 변수명: l7mn6

Gate1[j].Chan[j].OutputMode 는 PMAC2 스타일 서보 IC 채널의 명령 출력 신호 라인에서 어떤 출력 형식이 사용될지 제어합니다. 이에는 다음의 설정이 가능합니다.

- **Gate1[j].Chan[j].OutputMode** = 0: 출력 A & B 는 PWM, 출력 C 는 PWM
- **Gate1[j].Chan[j].OutputMode** = 1: 출력 A & B 는 DAC, 출력 C 는 PWM
- **Gate1[j].Chan[j].OutputMode** = 2: 출력 A & B 는 PWM, 출력 C 는 PFM
- **Gate1[j].Chan[j].OutputMode** = 3: 출력 A & B 는 DAC, 출력 C 는 PFM

3 상 다이렉트 PWM 명령 형식이 필요한 경우에 **Gate1[j].Chan[j].OutputMode** 는 0 으로 설정합니다. (외부의)D/A 컨버터에 대한 신호 출력이 필요한 경우에 **Gate1[j].Chan[j].OutputMode** 는 1 또는 3 으로 설정합니다. 이 경우 C 출력은 PWM 혹은 PFM 형식 중 하나로 보조(비서보) 출력으로 사용할 수 있습니다. 예를 들어 이것은 PFM 형식으로 MLDT 센서(예를 들어 Temposonics™)를 여기하기 위해서 사용할 수 있습니다.

Gate1[j].Chan[j].OutputMode 는 풀 워드 요소 **Gate1[j].Chan[j].Ctrl** 의 비트 22-23 을 구성합니다. (C 에서는 32 비트 요소의 비트 30-31)

Gate1[j].Chan[j].OutputPol 게이트 1 채널의 출력 극성

설명: 서보 IC 채널의 출력 극성 제어

범위: 0 ..3

단위: 비트 필드

기본값: 0

기존 I 변수명: l7mn7

Gate1[i].Chan[j].OutputPol 은 PMAC2 스타일 서보 IC 채널의 명령 출력 신호의 high/low 극성을 제어합니다. 다음의 설정이 가능합니다.

- **Gate1[i].Chan[j].OutputPol = 0:** 출력 A & B 를 반전하지 않음, 출력 C 를 반전하지 않음
- **Gate1[i].Chan[j].OutputPol = 1:** 출력 A & B 를 반전함, 출력 C 를 반전하지 않음
- **Gate1[i].Chan[j].OutputPol = 2:** 출력 A & B 를 반전하지 않음, 출력 C 를 반전함
- **Gate1[i].Chan[j].OutputPol = 3:** 출력 A & B 를 반전함, 출력 C 를 반전함

기본값의 비반전 출력은 정방향입니다. 출력 A, B 와 C 상의 PWM 신호에 대해서는 트랜지스터 ON 신호가 high 임을 뜻합니다. Delta Tau PWM 입력 앰프 및 대부분의 다른 PWM 입력 앰프는 이 비반전 출력 형식이 필요합니다. 이러한 3 상 모터 드라이브에는 **Gate1[i].Chan[j].OutputPol** 은 0 으로 설정합니다.



주의

PWM 신호의 high/low 극성이 특정 앰프에서 잘못된 경우에 **Gate1[i].PwmDeadTime** 에서 설정되는 Top 과 Bottom 의 ON 상태 간의 데드 타임으로 의도된 것은 오버랩 합니다. 앰프 입력 회로가 이것을 올바르게 배제하지 않는 경우에 이것은 버스 전원과 GND 사이의 순간적인 쇼트의 원인이 됩니다. 이것은 파워 트랜지스터를 순식간에 파괴합니다.

출력 C 의 PFM 신호에 대해서는 비반전은 펄스 ON 신호가 high 임을 뜻합니다(방향 극성은 **Gate1[i].Chan[j].PfmDirPol** 에 따라서 제어됩니다). 방향 전환 중에 방향 비트는 펄스의 선단 엣지에 동기하여 변화합니다. 비반전 출력 형식에서는 상승 엣지를 뜻합니다. 드라이버가 펄스의 상승 엣지 전에 방향 라인의 설정 시간이 필요한 경우에 펄스 출력을 반전시켜서 상승 엣지가 후방 엣지가 되고 펄스 폭(**Gate1[i].PwmDeadTime** 에 따라 설정됨)이 설정 시간이 됩니다.

출력 A 와 B 의 DAC 신호에 대해서는 비반전은 DAC 에 대한 1 의 값이 high 임을 뜻합니다. 다른 모든 기존에 알려진 DAC 와 마찬가지로 Delta Tau 액세서리 보드에서 사용되는 DAC 도 항상 비반전 입력이 필요합니다. 따라서 이 채널에서 DAC 를 사용하는 경우에 **Gate1[i].Chan[j].OutputPol** 은 0 또는 2 로 항상 설정해야 합니다.



경고

DAC 의 디지털 데이터의 high/low 극성을 변경하는 것은 DAC 의 아날로그 출력 전압 극성을 반대로 하는 효과가 있습니다. 이는 출력과 피드백 간의 극성 일치를 변경합니다. 만약 피드백 루프가 부귀환으로 안정되었다면 해당 변경은 불안정한 정귀환을 생성하고 위험한 폭주 상태를 야기합니다.

Gate1[i].Chan[j].OutputPol 은 풀 워드 요소 **Gate1[i].Chan[j].Ctrl** 의 비트 20-21 을 구성합니다. (C 에서는 32 비트 요소의 비트 28-29)

Gate1[i].Chan[j].PfmDirPol 게이트 1 채널의 PFM 방향 출력 극성

설명: 서보 IC 채널의 PFM 방향 출력 극성 제어

범위: 0 ..1

단위: 논리값

기본값: 0

기존 I 변수명: I7mn8

Gate1[i].Chan[j].PfmDirPol 은 PMAC2 스타일 서보 IC 채널의 펄스와 방향 형식에서 방향 출력 신호의 극성을 제어합니다. **Gate1[i].Chan[j].OutputMode** 가 2 또는 3의 펄스 주파수 변조(PFM) 출력으로 출력 C 를 사용하도록 설정된 경우에만 이 변수는 유효합니다. 다음의 설정이 가능합니다.

- **Gate1[i].Chan[j].PfmDirPol** = 0: 방향 신호를 반전하지 않음(+ = low, - = high)
- **Gate1[i].Chan[j].PfmDirPol** = 1: 방향 신호를 반전함(- = low, + = high)

Gate1[i].Chan[j].PfmDirPol 이 0인 기본값으로 설정되는 경우에 양의 방향 명령은 low 출력이 됩니다. **Gate1[i].Chan[j].PfmDirPol** 이 1로 설정되는 경우에 양의 방향 명령은 high 출력이 됩니다.

Gate1[i].Chan[j].PfmDirPol 은 풀 워드 요소 **Gate1[i].Chan[j].Ctrl** 의 비트 19 를 구성합니다. (C에서는 32비트의 요소인 비트 27)

Gate3[i].(PMAC3 스타일 IC)저장 데이터 구조체 요소(V1.6 2014.03)

Gate3[i]. 데이터 구조체는 Delta Tau 의 머신 인터페이스 ASIC "DSPGATE3" 중 메모리 맵의 레지스터로 구성됩니다. 이 IC 에는 서보와 범용 I/O 용도의 입출력 신호를 처리하는 많은 최신 기능이 장비됩니다.

Power PMAC 스크립트 환경에서는 "**Gate3[i]**" 구조체명 대신 장착된 액세스리의 이름을 사용할 수 있습니다. 현재 이를 지원하는 액세스리의 이름은 다음과 같습니다.

- **Acc24E3[i].** ACC-24E3 축 인터페이스 보드
- **Acc5E3[i].** ACC-5E3 MACRO 인터페이스 보드
- **Acc59E3[i].** Acc-59E3 A/D & D/A 변환기 보드

해당 이름들은 단순한 **Gate3[i]** 이름의 "별칭"입니다. **Gate3[i]** 또는 별칭이 사용되어도 인덱스값 *i* 는 동일합니다. 각 DSPGATE3 IC 에는 고유의 인덱스값이 사용됩니다. C 프로그램은 **Gate3[i]**의 이름을 사용해야 합니다. C 환경에서는 별칭을 사용할 수 없습니다.

IC 의 인덱스값 *i* 는 0~15 범위로 설정할 수 있습니다. 이는 보드 상의 하드웨어 주소 DIP 스위치 설정에 따라 결정됩니다.

주의: 권한이 없는 사용자가 **Gate3[i].** 데이터 구조체 설정 요소를 재가입할 수 있는 잠재적 위험을 방지하기 위해 많은 설정 요소는 "쓰기 방지"되었으므로 만약 쓰기 방지된 설정 요소에 쓰기 전에 IC 중의 "쓰기 방지 키"를 적절하게 설정하지 않으면 값을 변경할 수 없습니다.

쓰기 방지 키 레지스터에는 **Gate3[i].WpKey** 를 통해 액세스할 수 있습니다. 또한 보호된 설정 요소에 대한 쓰기 조작을 가능하게 하는 등록 값은 \$AAAAAAAA 입니다. IC 는 보호된 요소에 대한 쓰기 후에 자동으로 키값을 0 으로 클리어합니다. 따라서 보호된 요소에 대한 쓰기마다 키를 다시 설정해야 합니다.

C 프로그램에서는 이를 명시적으로 실시해야 합니다. 즉 **Gate3[i].WpKey** 에 \$AAAAAAAA 를 쓰는 것은 각각 보호된 설정 요소에 대한 쓰기에 선행하여 실시해야 합니다.

스크립트 환경(버퍼 프로그램 또는 온라인 명령 중 어느 하나)에서는 프로세스를 더 간단하게 할 수 있는 다른 방법이 있습니다. 쓰기 방지된 설정 요소에 쓰는 스크립트 동작 전에 Power PMAC 은 글로벌 메모리 요소 **Sys.WpKey** 의 값을 실제 쓰기 방지 키 레지스터 **Gate3[i].WpKey** 에 복사합니다. 변경 가능하게 하기 위해 사용자는 글로벌 요소에 \$AAAAAAAA 의 키를 한 번만 씁니다. 그리고 Power PMAC 은 쓰기가 필요할 때마다 자동으로 IC 레지스터 키를 복사합니다.

Sys.WpKey 는 저장되는 값이 아닙니다. 따라서 IC 의 보호된 설정 요소를 변경하려면 전원 투입/리셋 시마다 키를 설정해야 합니다. IC 를 설정하기 위한 통합 개발 환경(IDE) 프로그램의 특별 윈도우에서는 자동으로 **Sys.WpKey** 의 값을 처리합니다.

물론 실제 어플리케이션에서 이러한 설정 요소에 대한 변경을 방지하려면 **Sys.WpKey** 의 값을 설정하면 안 됩니다. 또한 권한이 없는 사용자에게는 키값이나 메커니즘에 대한 정보를 제공해서는 안 됩니다.

Gate3[i].멀티채널 설정 요소

이 장의 데이터 구조체 요소는 IC 의 4 가지 서보 채널 모두에서 공통적인 신호를 구성합니다. 여기에는 범용 I/O 와 MACRO 기능의 신호가 있습니다.

Gate3[i].AdcAmpClockDiv 게이트 3 의 앰프용 ADC 클럭 분주

설명: IC 의 "앰프용" ADC 클럭 주파수 제어

범위: 0 ..15

단위: 없음

기본값: 5(3.125MHz)

Gate3[i].AdcAmpClockDiv 는 IC 의 채널에 연결된 "앰프용" A/D 변환기(ADC)로부터의 시리얼 데이터 주파수를 제어하는 클럭 신호의 주파수를 지정합니다. 보통 모터의 "다이렉트 PWM" 제어의 전류 피드백에는 이러한 ADC 를 사용합니다. 이 요소는 내부 100MHz 클럭 신호를 2 로 몇 번 나눌지 지정하여 주파수를 제어합니다. 클럭 주파수를 구하는 식은 다음과 같습니다.

$$f_{AdcAmpClk} = \frac{100MHz}{2^{AdcAmpClockDiv}}$$

다음 표는 **Gate3[i].AdcAmpClockDiv** 에 할 수 있는 설정과 지원하는 주파수를 나타냅니다.

설정	제수	주파수	설정	제수	주파수
0	1	100MHz	8	256	390.6kHz
1	2	50MHz	9	512	195.3kHz
2	4	25MHz	10	1,024	97.65kHz
3	8	12.5MHz	11	2,048	48.82kHz
4	16	6.25MHz	12	4,096	24.41kHz
5	32	3.125MHz	13	8,192	12.21kHz
6	64	1.562MHz	14	16,384	6.104kHz
7	128	781.2kHz	15	32,768	3.052kHz

보통 데이터 취득 시의 지연을 최소한으로 하기 위해 ADC 정격의 가장 높은 클럭 주파수를 선택해야 합니다.

Gate3[i].AdcAmpClockDiv 는 풀 워드 요소 **Gate3[i].HardwareClockCtrl** 의 비트 28-31 의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].AdcAmpCtrl 게이트 3의 앰프용 ADC 제어

설명: 앰프용 ADC 구성의 IC 제어 워드

범위: \$0..\$FFFFFFFF

단위: 비트 필드

기본값: \$FFFFFFC02

Gate3[i].AdcAmpCtrl은 "앰프용 ADC"의 인터페이스를 제어하는 풀 워드 요소입니다. 보통 해당 ADC는 다이렉트 PWM 앰프의 "파워 블록"으로부터의 전류 피드백에 사용됩니다. 다음 부분 워드 요소로 **Gate3[i].AdcAmpCtrl**은 구성됩니다.

구성 요소	비트	기능
AdcAmpStrobe	31 - 08	앰프용 ADC의 스트로브 워드
AdcAmpDelay	07 - 04	앰프용 ADC 스트로브 지연
AdcAmpUtoS	03	앰프용 ADC 형식 변환 제어
AdcAmpHeaderBits	02 - 00	앰프용 ADC 헤더 비트 수

사용자는 부분 워드 요소의 설정과 설정 확인을 위해 보통 스크립트 환경에서 여기에 액세스합니다. Power PMAC은 효율화를 위해서 풀 워드 요소로 저장 및 복원합니다.

C 프로그램에서는 사용자는 32비트 워드의 풀 워드 요소에만 액세스해야 합니다.

풀 워드 요소의 값이 스크립트 환경 속에서 참조되는 경우에 부분 워드 요소 값이 10진수로 통지되어도 Power PMAC은 16진수 형식으로 값을 반환합니다.

Gate3[i].AdcAmpCtrl은 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey**에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey**에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

각각의 요소의 기능에 대한 자세한 설명은 각 부분 워드 요소 항목을 참조하십시오.

Gate3[i].AdcAmpDelay 게이트 3의 앰프용 ADC 시작 지연

설명: IC의 앰프용 AD 컨버터 스트로브의 시작 지연

범위: 0..15

단위: 앰프용 ADC 클록 사이클

기본값: 0

Gate3[i].AdcAmpDelay는 위상 클록의 상승 및 앰프용 ADC 스트로브 신호 간의 시간을 지정하여 앰프용 ADC 클록의 사이클로 표현합니다. 기본값 0에서는 위상 클록의 상승 엣지 후의 첫 앰프용 ADC 클록 엣지에서 변환을 시작합니다.

특히 파워 단에서의 PWM 출력 변환에 지연이 있는 경우나 위상 클록 엣지에서 노이즈가 발생하면 **Gate3[i].AdcAmpDelay** 로 앰프 위상 전류의 샘플링 타이밍을 최적화할 수 있습니다. 그러나 대부분의 시스템에서는 기본값 0에서 충분한 성능을 달성합니다.

Gate3[i].AdcAmpDelay 는 풀 워드 요소 **Gate3[i].AdcAmpCtrl** 의 비트 4-7의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].AdcAmpHeaderBits 게이트 3의 앰프용 ADC 헤더 비트

설명: IC의 앰프용 ADC의 헤더 비트 수

범위: 0..7

단위: 없음

기본값: 2

기존 I 변수명: 없음

Gate3[i].AdcAmpHeaderBits 는 IC 상의 모든 채널의 "앰프용" ADC에서의 시리얼 데이터 열의 처음에 요구되는 "헤더 비트" 수를 지정합니다. 헤더 비트의 지정된 수는 얻어진 데이터 워드의 최상위 비트에서 최하위 비트까지의 "배럴 시프터"입니다. 지정 수의 헤더 비트 다음에 최초로 수신된 비트가 레지스터의 최상위에 배치됩니다.

또한 이 파라미터로는 중간 처리 로직에 기인하여 일어나는 ADC 데이터를 반환할 때의 지연을 보정할 수 있습니다. 하나의 클록 사이클 지연인 경우에 이 파라미터는 1만큼 증가되어야 합니다.

이 요소에서는 실제 변환된 데이터 전에 1개 이상의 선두 비트를 추가하는 ADC에 Power PMAC을 간단히 연결할 수 있습니다. 가장 일반적인 Delta Tau 제품의 다이렉트 PWM 앰프와의 인터페이스에서는 1 헤더 비트를 갖춘 ADC를 이용하므로 1 클록 사이클 지연이 필요해집니다.

Gate3[i].AdcAmpHeaderBits 는 풀 워드 요소 **Gate3[i].AdcAmpCtrl** 의 비트 0-2의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].AdcAmpStrobe 게이트 3의 앰프 ADC 스트로브 워드

설명: IC의 앰프용 ADC의 스트로브 워드

범위: \$000000..\$FFFFFF

단위: 비트 필드

기본값: \$FFFFFF

Gate3[i].AdcAmpStrobe 로는 IC 상의 채널에 연결된 모든 "앰프용" ADC 의 스트로브 신호를 지정합니다. 이 요소로 지정된 24 비트 워드가 스트로브 라인 상에 연속적으로 시프트 출력됩니다. MSB 에서 앰프용 ADC 클록 사이클당 1 비트씩 위상 클록의 상승 엣지(**Gate3[i].AdcAmpDelay** 만큼 지연될 수 있음)부터 시작됩니다. 다음 위상 클록 사이클까지 LSB 값이 유지됩니다.

최초의 "1"에 의해 보통 "변환 시작" 신호로 사용되는 앰프용 ADC 스트로브 출력의 상승 엣지를 생성합니다. 일부 ADC 에서는 변환에 이 상승 엣지가 필요합니다. 그 외에는 변환을 하는 중에 신호를 high 로 해야 합니다. 다음 사이클로 상승 엣지를 생성하도록 항상 스트로브 워드의 LSB 를 0 으로 설정합니다. \$FFFFFFC 의 기본값은 실질적으로 모든 ADC 에 적합합니다.

Gate3[i].AdcAmpStrobe 는 풀 워드 요소 **Gate3[i].AdcAmpCtrl** 의 비트 8-31 의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].AdcAmpUtoS 게이트 3 의 앰프용 ADC 부호 변환

설명: IC 의 앰프용 ADC 데이터 변환

범위: 0 ..1

단위: 없음

기본값: 0

Gate3[i].AdcAmpUtoS 는 IC 의 모든 채널 상의 "앰프용" ADC 회로에서 프로세서의 읽기 가능한 레지스터로 래치하기 전에 수신 데이터의 형식 변환을 실시할지 여부를 지정합니다. 기본값 0 에서는 형식 변환을 실시하지 않습니다.

Gate3[i].AdcAmpUtoS 를 1 로 설정하면 IC 는 읽기 가능한 레지스터로 래치하기 전에 부호가 없는 데이터를 부호가 있는(2 의 보수) 형식으로 변환합니다. 디지털 전류 클로즈드 루프처럼 이러한 값을 사용하는 Power PMAC 의 자동 루틴에는 부호가 있는 데이터 형식이 필요합니다. 그러나 일부 ADC 는 부호가 없는 형식의 데이터로 제공됩니다. 이 변환은 부호가 없는 ADC 를 Power PMAC 의 자동 알고리즘에서 사용할 수 있게 합니다.

Gate3[i].AdcAmpUtoS 는 풀 워드 요소 **Gate3[i].AdcAmpCtrl** 의 비트 3 의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].AdcEncClockDiv 게이트 3의 엔코더용 ADC 클럭 분주

설명: IC의 "엔코더용" ADC 클럭 주파수 제어

범위: 0..15

단위: 없음

기본값: 5(3.125MHz)

Gate3[i].AdcEncClockDiv는 IC의 채널에 연결된 "엔코더용" ADC로부터의 시리얼 데이터 주파수를 제어하는 클럭 신호의 주파수를 지정합니다. 보통 이러한 ADC는 사인파 엔코더나 리졸버로부터의 위치 피드백에 사용됩니다. 이 요소는 내부 100MHz 클럭 신호를 2로 몇 번 나눌지 지정하여 주파수를 제어합니다. 클럭 주파수를 구하는 식은 다음과 같습니다.

$$f_{AdcEncClk} = \frac{100MHz}{2^{AdcEncClockDiv}}$$

다음 표는 **Gate3[i].AdcEncClockDiv**에 할 수 있는 설정과 지원하는 주파수를 나타냅니다.

설정	제수	주파수	설정	제수	주파수
0	1	100MHz	8	256	390.6kHz
1	2	50MHz	9	512	195.3kHz
2	4	25MHz	10	1,024	97.65kHz
3	8	12.5MHz	11	2,048	48.82kHz
4	16	6.25MHz	12	4,096	24.41kHz
5	32	3.125MHz	13	8,192	12.21kHz
6	64	1.562MHz	14	16,384	6.104kHz
7	128	781.2kHz	15	32,768	3.052kHz

보통 데이터 취득 시의 지연을 최소한으로 하기 위해 ADC 정격의 가장 높은 클럭 주파수를 선택해야 합니다.

Gate3[i].AmpEncClockDiv는 폴 워드 요소 **Gate3[i].HardwareClockCtrl**의 비트 24-27의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey**에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey**에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].AdcEncCtrl 게이트 3의 엔코더용 ADC 제어

설명: 엔코더용 ADC 구성의 IC 제어 워드

범위: \$0..\$FFFFFFFF

단위: 비트 필드

기본값: \$FFFFFFC01

Gate3[i].AdcEncCtrl 은 "엔코더용 ADC"의 인터페이스를 제어하는 풀 워드 요소입니다. 보통 이러한 ADC 는 사인파 엔코더나 리졸버와 같은 아날로그 위치 피드백 센서에 사용됩니다. 다음 부분 워드 요소로 **Gate3[i].AdcEncCtrl** 은 구성됩니다.

구성 요소	비트	기능
AdcEncStrobe	31 - 08	엔코더용 ADC 스트로브 워드
AdcEncDelay	07 - 04	엔코더용 ADC 스트로브 지연
AdcEncUtoS	03	엔코더용 ADC 형식 변환 제어
AdcEncHeaderBits	02 - 00	엔코더용 ADC 헤더 비트 수

사용자는 부분 워드 요소의 설정과 설정 확인을 위해 보통 스크립트 환경에서 여기에 액세스합니다. Power PMAC 은 효율화를 위해서 풀 워드 요소로 저장 및 복원합니다.

C 프로그램에서는 사용자는 32 비트 워드의 풀 워드 요소에만 액세스해야 합니다.

풀 워드 요소의 값이 스크립트 환경 속에서 참조되는 경우에 부분 워드 요소 값이 10 진수로 통지되어도 Power PMAC 은 16 진수 형식으로 값을 반환합니다.

Gate3[i].AdcEncCtrl 은 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

각각의 요소의 기능에 대한 자세한 설명은 각 부분 워드 요소 항목을 참조하십시오.

Gate3[i].AdcEncDelay 게이트 3 의 엔코더용 ADC 시작 지연

설명: IC 의 엔코더용 ADC 스트로브의 시작 지연

범위: 0 ..15

단위: 엔코더용 ADC 클록 사이클

기본값: 0

Gate3[i].AdcEncDelay 는 위상 클록 신호의 상승 엣지 및 엔코더용 ADC 스트로브 신호 간의 시간을 지정하여 엔코더용 ADC 클록의 사이클로 표현합니다. 기본값 0에서는 위상 클록의 상승 엣지 후의 첫 엔코더용 ADC 클록 엣지에서 변환을 시작합니다.

특히 위상 클록 엣지에서 노이즈가 혼입된 경우에 **Gate3[i].AdcEncDelay** 로 아날로그 위치 센서 값의 샘플링 타이밍을 최적화할 수 있게 합니다. 그러나 대부분의 시스템에서는 기본값 0에서 충분한 성능을 달성합니다.

Gate3[i].EncAmpDelay 는 풀 워드 요소 **Gate3[i].AdcEncCtrl** 의 비트 4-7 의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].AdcEncHeaderBits 게이트 3의 엔코더용 ADC 헤더 비트

설명:	IC의 엔코더용 ADC의 헤더 비트 수
범위:	0..7
단위:	없음
기본값:	1

Gate3[i].AdcEncHeaderBits는 IC상의 모든 채널의 "엔코더용" ADC로부터의 시리얼 데이터 열의 처음에 요구되는 "헤더 비트" 수를 지정합니다. 헤더 비트로 지정된 수는 얻어진 데이터 워드의 최상위 비트에서 최하위 비트까지의 "배럴 시프트"입니다. 지정 수의 헤더 비트 다음에 최초로 수신된 비트가 레지스터의 최상위에 배치됩니다.

또한 이 파라미터로는 중간 처리 로직에 기인하여 일어나는 ADC 데이터를 반환할 때의 지연을 보정할 수 있습니다. 하나의 클록 사이클 지연인 경우에 이 파라미터는 1만큼 증가되어야 합니다.

이 요소는 실제 변환된 데이터 전에 1개 이상의 선두 비트를 추가하는 ADC에 Power PMAC을 간단히 연결할 수 있습니다. 가장 일반적인 Delta Tau 아날로그 피드백 장치와의 인터페이스에서는 1 헤더 비트를 갖춘 ADC를 이용하므로 중간 지연이 없습니다.

Gate3[i].AdcEncHeaderBits는 풀 워드 요소 **Gate3[i].AdcEncCtrl**의 비트 0-2의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey**에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey**에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].AdcEncStrobe 게이트 3의 엔코더용 ADC 스트로브 워드

설명:	IC의 엔코더용 ADC의 스트로브 워드
범위:	\$000000..\$FFFFFF
단위:	비트 필드
기본값:	\$FFFFFFC

Gate3[i].AdcEncStrobe는 IC상의 채널에 연결된 모든 "엔코더용" ADC의 스트로브 신호를 지정합니다. 이 요소로 지정된 24비트 워드가 스트로브 라인 상에 연속적으로 시프트 출력됩니다. MSB에서 엔코더용 ADC 클록 사이클당 1비트씩 위상 클록의 상승 엣지(**Gate3[i].AdcEncDelay**만큼 지연될 수 있음)부터 시작됩니다. 다음 위상 클록 사이클까지 LSB 값이 유지됩니다.

최초의 "1"에 의해 보통 "변환 시작"신호로 사용되는 엔코더용 ADC 스트로브 출력의 상승 엣지를 생성합니다. 일부 ADC는 변환에 이 상승 엣지가 필요합니다. 그 밖에는 변환을 하는 중에 신호를 high로 해야 합니다. 다음 사이클로 상승 엣지를 생성하도록 항상 스트로브 워드의 LSB를 0으로 설정합니다. \$FFFFFFC의 기본값은 실질적으로 모든 ADC에 적합합니다.

Gate3[i].AdcEncStrobe는 풀 워드 요소 **Gate3[i].AdcEncCtrl**의 비트 8-31의 부분적인 구성 요소입니다.

Gate3[i].AdcEncUtoS 게이트 3의 엔코더용 ADC 부호 변환

설명: IC의 엔코더용 ADC 데이터 변환

범위: 0..1

단위: 없음

기본값: 0

Gate3[i].AdcEncUtoS는 IC의 모든 채널 상의 "엔코더용" ADC 회로에서 프로세서의 읽기 가능한 레지스터로 래치하기 전에 수신 데이터의 형식 변환을 실시할지 여부를 지정합니다. 기본값 0에서는 형식 변환을 실시하지 않습니다.

Gate3[i].AdcEncUtoS를 1로 설정하면 IC는 읽기 가능한 레지스터로 래치하기 전에 부호가 없는 데이터를 부호가 있는 (2의 보수)형식으로 변환합니다. 아크탄젠트 위치 계산처럼 이러한 값을 사용하는 Power PMAC의 자동 루틴에는 부호가 있는 데이터 형식이 필요합니다. 그러나 일부 ADC는 부호가 없는 형식 데이터로 제공됩니다. 이 변환은 부호가 없는 ADC를 Power PMAC의 자동 알고리즘에서 사용할 수 있게 합니다.

Gate3[i].AdcEncUtoS는 풀 워드 요소 **Gate3[i].AdcEncCtrl**의 비트 3의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey**에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey**에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].ClockPol 게이트 3의 클록 극성 제어

설명: IC의 DAC & ADC 클록의 극성 제어

범위: 0..7

단위: 비트 필드

기본값: 0

Gate3[i].ClockPol 은 IC 에 연결된 DAC 와 ADC 를 위한 데이터 주파수를 제어하는 3 가지 클록 신호의 high/low 극성을 지정합니다. 하나의 비트가 하나의 클록 신호를 제어하는 3 비트값입니다. 비트가 0 인 경우에 클록 신호는 반전되지 않습니다(클록 사이클 시작으로 인해 high 레벨이 됩니다). 비트가 1 인 경우에 클록 신호는 반전됩니다(클록 사이클 시작으로 인해 low 레벨이 됩니다.) 각각의 비트 및 제어되는 클록 신호는 다음과 같습니다.

- 비트 0(값 1): DAC 클록
- 비트 1(값 2): 엔코더용 ADC 클록
- 비트 2(값 4): 앰프용 ADC 클록

이 요소의 적절한 설정에 대해서는 특정한 하드웨어 매뉴얼을 참조하십시오. 일반적으로 이 IC 를 사용하는 Delta Tau 제품은 이 요소에 기본값 0 을 설정합니다.

Gate3[i].ClockPol 은 풀 워드 요소 **Gate3[i].HardwareClockCtrl** 의 비트 5-7 의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].DacClockDiv 게이트 3 의 DAC 클록 분주

설명: IC 의 DAC 클록 주파수 제어

범위: 0 ..15

단위: 없음

기본값: 5(3.125MHz)

Gate3[i].DacClockDiv 는 IC 의 채널에 연결된 D/A 변환기(DAC)에 대한 시리얼 데이터 주파수를 제어하는 클록 신호의 주파수를 지정합니다. 보통 이러한 DAC 은 서보 앰프에 대한 아날로그 속도, 토크 또는 위상 전류 신호의 출력에 사용됩니다. 이 요소는 내부 100MHz 클록 신호를 2 로 몇 번 나눌지 지정하여 주파수를 제어합니다. 클록 주파수를 구하는 식은 다음과 같습니다.

$$f_{DacClk} = \frac{100MHz}{2^{DacClockDiv}}$$

다음 표는 **Gate3[i].AdcAmpClockDiv** 에 할 수 있는 설정과 지원하는 주파수를 나타냅니다.

설정	제수	주파수	설정	제수	주파수
0	1	100MHz	8	256	390.6kHz
1	2	50MHz	9	512	195.3kHz
2	4	25MHz	10	1,024	97.65kHz
3	8	12.5MHz	11	2,048	48.82kHz
4	16	6.25MHz	12	4,096	24.41kHz
5	32	3.125MHz	13	8,192	12.21kHz
6	64	1.562MHz	14	16,384	6.104kHz
7	128	781.2kHz	15	32,768	3.052kHz

보통 데이터를 전송할 때의 지연을 최소한으로 하기 위해 DAC 정격의 가장 높은 클록 주파수를 선택해야 합니다.

Gate3[i].DacClockDiv 는 폴 워드 요소 **Gate3[i].HardwareClockCtrl** 의 비트 20-23 의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].DacStrobe 게이트 3 의 DAC 스트로브 워드

설명: IC 의 DAC 스트로브 워드

범위: \$00000000..\$FFFFFFFF

단위: 비트 필드

기본값: \$FFFF0000

Gate3[i].DacStrobe 는 IC 상의 채널에 연결된 모든 DAC 의 스트로브 신호를 지정합니다. 이 요소로 지정된 32 비트 워드가 DAC 스트로브 라인 상에 연속적으로 시프트 출력됩니다. MSB 에서 DAC 클록 사이클당 1 비트씩 위상 클록의 상승 엣지부터 시작됩니다. 다음 위상 클록 사이클까지 LSB 값이 유지됩니다.

n-비트의 데이터 폭을 전송하는 DAC 인 경우에 보통 스트로브 라인을 n-1 클록 사이클 동안 high 로 유지합니다. 따라서 이 변수의 공통 설정은 다음과 같습니다.

- 24 비트 폭(ACC-24E3 의 18 비트 DAC 에는): \$FFFFFF00
- 18 비트 폭(일부 18 비트 DAC 에는): \$FFFFC000
- 16 비트 폭: \$FFFF0000
- 12 비트 폭: \$FFF00000

ACC-24E3 상의 아날로그 앰프 보드에서 표준인 16 비트 DAC 은 16 비트의 데이터부를 갖추고 있으며 이를 위해 \$FFFF0000 값을 사용합니다. 이 보드 옵션의 18 비트 DAC 은 24 비트의 폭을 가지고 있으며 이를 위해 \$FFFFFF00 값을 사용합니다.

ACC-24E3 의 18 비트 DAC 옵션처럼 데이터부의 비트 수와 실제 DAC 비트 수 간에 차가 있는 경우에 **Motor[x].DacShift** 를 이 차이(즉 해당 옵션용의 6)값을 설정해야 합니다. 비트 수가 동일한 경우에 **Motor[x].DacShift** 를 그대로 기본값 0 으로 둡니다.

Gate3[i].DacStrobe 는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].EncClockDiv 게이트 3 의 엔코더 클록 분주

설명: IC 의 엔코더 샘플링 클록 주파수 제어

범위: 0 ..15

단위: 없음

기본값: 5(3.125MHz)

Gate3[i].EncClockDiv 로는 IC 의 채널의 엔코더 입력 샘플링을 제어하는 클록 신호의 주파수를 지정합니다. 엔코더 클록 사이클에서 "A", "B" 각각의 엔코더 디지털 입력을 샘플링합니다. 디코드 로직은 임의의 클록 사이클 중에 단 하나의 신호 엣지를 허가합니다. 따라서 클록 주파수보다 높은 엣지 레이트를 취급할 수 없습니다. 이는 최대 카운트 주파수가 이 클록 주파수보다 높아질 수 없음을 의미합니다. 절대 최대 신호 주파수가 해당 클록 주파수의 1/4 이라는 점에 주의하십시오. 일반적으로 신호의 불완전성 때문에 20%의 안전 여유를 두는 것을 권장합니다.

현재 ASIC 자체는 Power PMAC 하드웨어 상의 엔코더 리시버 회로보다 높은 신호 주파수를 읽을 수 있습니다. 이러한 리시버는 절대 최대 10MHz 의 신호 주파수(40MHz 의 카운트 주파수)로 제한됩니다. 안전 여유를 감안하여 주파수 8MHz 를 초과하는 신호를 부여하면 안 됩니다. 이러한 리시버는 디지털 AB 상 엔코더의 직접 사용 및 아날로그 사인파 엔코더의 콤파레이터 양쪽에 사용됩니다.

또한 이 클록 신호는 엔코더와 플래그 입력을 위한 디지털 지연 필터에 의해 신호 전달을 제어합니다. 1 클록 사이클만 존재하는 모든 입력 상태에서도 제거됩니다. 클록 주파수가 낮을수록 필터에서 제거되는 노이즈 펄스는 길어집니다. (플래그 입력을 제 2 디지털 지연 필터로 필터화하는 옵션이 있습니다. 그때 전달이 **Gate3[i].FiltClockDiv** 의 클록 주파수로 제어됩니다.)

이 요소는 내부 100MHz 클록 신호를 2 로 몇 번 나눌지 지정하여 주파수를 제어합니다. 클록 주파수를 구하는 식은 다음과 같습니다.

$$f_{EncClk} = \frac{100MHz}{2^{EncClockDiv}}$$

다음 표는 **Gate3[i].EncClockDiv** 에 할 수 있는 설정과 지원하는 주파수를 나타냅니다.

설정	제수	주파수	설정	제수	주파수
0	1	100MHz	8	256	390.6kHz
1	2	50MHz	9	512	195.3kHz
2	4	25MHz	10	1,024	97.65kHz
3	8	12.5MHz	11	2,048	48.82kHz
4	16	6.25MHz	12	4,096	24.41kHz
5	32	3.125MHz	13	8,192	12.21kHz
6	64	1.562MHz	14	16,384	6.104kHz
7	128	781.2kHz	15	32,768	3.052kHz

일반적으로 가능한 한 최대 계수율을 수신할 수 있는 최소 클록 주파수를 선택해야 합니다.

Gate3[i].EncClockDiv 는 폴 워드 요소 **Gate3[i].HardwareClockCtrl** 의 비트 12-15 의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].EncLatchDelay 게이트 3 의 엔코더 래치 지연

설명: IC 의 아날로그 엔코더의 래치/스트로브 지연

범위: 0 ..255

단위: 엔코더용 ADC 클록 사이클

기본값: 0

Gate3[i].EncLatchDelay 는 위상 클록의 상승 엣지에서 엔코더 ADC 가 스트로브 되어 엔코더 카운터값이 래치될 때까지의 지연을 제어합니다. 엔코더 ADC 의 클록 사이클 수로 설정합니다. 이 기능은 **Gate3[i].Chan[j].AtanEna** 을 1 로 설정한 채널에만 사용됩니다. 1 로 설정하면 엔코더 ADC 에서 아크탄젠트 소수점 카운트값과 카운트에서 래치된 정수 값의 자동 조합이 가능해집니다.

이 요소는 엔코더가 리졸브의 샘플링과 소프트웨어에 최종적인 값을 사용할 수 있게 하는 위상 또는 서보 인터럽트 지연을 최소한으로 할 수 있게 합니다. 수신과 ADC 데이터를 처리하기 위해 스트로브/래치 및 페이즈/서보 인터럽트 간에 최소 25 개의 엔코더용 ADC 클록 사이클이 필요합니다.

Gate3[i].EncLatchDelay 는 폴 워드 요소 **Gate3[i].PhaseServoClockCtrl** 의 비트 0-7 의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].FiltClockDiv 게이트 3 의 제 2 필터 클록 분주

설명: IC 의 제 2 필터 클록 주파수 제어

범위: 0 ..15

단위: 없음

기본값: $4(f_{EncClk}/16)$

Gate3[i].FiltClockDiv 로는 플래그 입력을 위한 제 2 디지털 지연 필터에 의한 신호 전달을 제어하는 클록 신호의 주파수를 지정합니다. 이 필터로는 1 개의 클록 사이클 간에만 존재하는 모든 입력 상태에서도 제거됩니다. 클록 주파수가 낮을수록 필터에서 제거되는 노이즈 펄스는 길어집니다. **Gate3[i].Chan[j].FlagFilt2Ena** 을 1 로 설정하는 경우에만 이 제 2 필터를 사용할 수 있습니다.

일부 어플리케이션에서는 프로세서에서 플래그 입력 상태를 읽기 전에 더 낮은 클록 주파수에서의 필터를 추가 사용해야 합니다. 제 1 디지털 지연 필터를 제어하는 엔코더의 샘플링 클록 주파수는 높은 계수율을 입력하기 위해 비교적 높아야 합니다. 플래그의 모든 포착과 트리거 기능도 제 2 필터 전에 실행된다는 점에 주의하십시오. 제 2 필터는 이러한 기능을 지연시키지는 않습니다.

이 요소는 샘플링 신호(100MHz 의 원본 클록 신호가 아님)를 2 로 몇 번 나눌지 여부를 지정하여 주파수를 제어합니다. 클록 주파수를 구하는 식은 다음과 같습니다.

$$f_{FiltClk} = \frac{f_{EncClk}}{2^{EncClockDiv}}$$

다음 표는 **Gate3[i].FiltClockDiv** 에 할 수 있는 설정과 지원하는 주파수를 나타냅니다.

설정	제수	주파수	설정	제수	주파수
0	1	f_{EncClk}	8	256	$f_{EncClk}/256$
1	2	$f_{EncClk}/2$	9	512	$f_{EncClk}/512$
2	4	$f_{EncClk}/4$	10	1,024	$f_{EncClk}/1024$
3	8	$f_{EncClk}/8$	11	2,048	$f_{EncClk}/2048$
4	16	$f_{EncClk}/16$	12	4,096	$f_{EncClk}/4096$
5	32	$f_{EncClk}/32$	13	8,192	$f_{EncClk}/8192$
6	64	$f_{EncClk}/64$	14	16,384	$f_{EncClk}/16384$
7	128	$f_{EncClk}/128$	15	32,768	$f_{EncClk}/32768$

일반적으로 실제 플래그 상태 이동에 충분히 빠른 응답 속도로 응답할 수 있는 최소 클록 주파수를 선택해야 합니다.

Gate3[i].FiltClockDiv 는 폴 워드 요소 **Gate3[i].HardwareClockCtrl** 의 비트 8-11 의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].GpioCtrl 게이트 3 의 범용 IO 제어

설명: IC 의 범용 I/O 제어

범위: \$0..\$FFFFFFFF

단위: 비트 필드

기본값: \$0

Gate3[i].GpioCtrl 은 IC 상의 범용 I/O 가 입력으로 사용될 때 어떤 식으로 처리할지 결정합니다. 이 32 비트 요소는 독립된 4 바이트로 구성되며 1 바이트는 I/O 신호의 각 "블록"입니다. 이 변수의 설정은 범용 입력으로 사용되는 IC 핀에만 관련됩니다.

4 바이트 **Gate3[i].GpioCtrl** 및 제어되는 I/O 뱅크는 다음과 같습니다.

바이트	비트	I/O 뱅크 인덱스	I/O 뱅크 핀 #	비고
3	31 - 24	3	4	Chan[3] 공통 핀
2	23 - 16	2	3	Chan[2] 공통 핀
1	15 - 08	1	2	Chan[1] 공통 핀
0	07 - 00	0	1	전용 I/O 핀

각 바이트 내에서 제어는 비트마다 분해됩니다.

- 비트 7 & 6: 그레이 코드 - 바이너리 변환 제어
 = 00: 그레이 코드 - 바이너리 변환을 실시하지 않음
 = 01: 입력 비트 31-08 및 07-00 에서의 독립된 그레이 코드 - 바이너리 변환
 = 10: 입력 비트 31-16 및 15-00 에서의 독립된 그레이 코드 - 바이너리 변환
 = 11: 입력 비트 31-00 에서의 모든 비트 그레이 코드 - 바이너리 변환
- 비트 5: 제 2 디지털 지연 필터(FiltClock 사용) 제어
 = 0: 제 2 필터 무효(바이패스됨)
 = 1: 제 2 필터 유효
- 비트 4: 제 1 디지털 지연 필터(EncClock 사용) 제어
 = 0: 제 1 필터 무효(바이패스됨)
 = 1: 제 1 필터 유효
- 비트 3: 래치 금지 레벨 제어
 = 0: High 에서 래치 금지
 = 1: Low 에서 래치 금지
- 비트 2: 래치 금지 제어:
 = 0: 래치 금지 무효
 = 1: 래치 금지 유효
- 비트 1: 래치의 클록 엣지 선택
 = 0: 클록의 하강 엣지에서 래치
 = 1: 클록의 상승 엣지에서 래치

- 비트 0: 래치 클록 선택
= 0: 래치에 위상 클록 사용
= 1: 래치에 서보 클록 사용

Gate3[j].GpioMode[0]으로 하면 **Gate3[j].GpioCtrl** 에 액세스할 수 있습니다. Power PMAC 은 **Gate3[j].GpioMode[0]**을 사용해서 이 요소의 값을 저장하고 복원합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[j].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경 중에서 자동으로 실행됩니다.)

Gate3[j].GpioDir[j] 게이트 3 의 범용 I/O 방향

설명: IC 의 범용 I/O BANK 입출력 방향

범위: \$0..\$FFFFFFFF

단위: 비트 필드

기본값: \$0 하드웨어 사양
\$00FF0000(Gate3[0].GpioDir[0])
\$00000000(기타)

Gate3[j].GpioDir[j]는 선택된 범용 I/O BANK의 IC 상의 I/O 핀이 입력 또는 출력 중 어느 쪽에서 사용될지 결정합니다. 이는 32 비트 폭을 갖는 4 개의 설정 요소 배열입니다. 각 요소 **GpioDir[0]~GpioDir[3]**은 동일한 인덱스값 0~3 의 GPIO BANK(각각 1 번째 BANK~4 번째 BANK)에 대응합니다.

각 요소 **Gate3[j].GpioDir[j]**는 독립된 32 비트 0-31 로 구성됩니다. 각각의 비트 n 은 BANK GPIO_n 의 신호 방향을 제어합니다. 비트값이 기본값 0 이면 입력으로 사용됩니다. 비트값이 1 이면 출력으로 사용됩니다. 신호 핀이 **Gate3[j].GpioMode[j]**가 대응하는 비트에 따라 GPIO 사용에 선택된 경우에 I/O 비트로만 사용됩니다. 항상 첫 BANK(인덱스값 0)는 GPIO 로 사용됩니다.

Gate3[j].GpioDir[j]는 쓰기 방지되었으므로 처음에 **Gate3[j].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[j].GpioMode[j] 게이트 3 의 범용 I/O 모드

설명: IC 의 범용 I/O BANK가 유효가 됩니다.

범위: \$0..\$FFFFFFFF

단위: 비트 필드

기본값: \$0

Gate3[j].GpioMode[j]는 IC 상의 선택된 बैं크의 범용 I/O 핀을 범용 I/O 가 전용 서보 기능으로 사용할지 여부를 결정합니다. 각 IC 에는 32 점의 I/O 를 갖춘 4 개의 बैं크가 있습니다. 첫 बैं크는 범용 I/O 를 위해 항상 사용됩니다. 2 번째, 3 번째 및 4 번째 बैं크는 IC 상의 2 번째, 3 번째 및 4 번째 서보 채널과 핀을 공유합니다. 다른 보드 설계에서는 해당 핀을 다른 목적으로 사용합니다.

범용 I/O 의 첫 बैं크(인덱스 0)에는 전용 핀이 있으므로 **Gate3[j].GpioMode[0]**에는 이 기능이 없습니다. **Gate3[j].GpioMode[0]**은 **Gate3[j].GpioCtrl** 과 레지스터를 공유하고 해당 요소의 대체명으로서 존재합니다. **GpioMode[1]**, **[2]** 또는 **[3]**을 사용하는 것처럼 **GpioMode[0]**을 사용하지 마십시오.

Gate3[j].GpioMode[1]은 2 번째 बैं크(인덱스값 1)의 핀을 범용 I/O 가 전용 서보 기능으로 사용할지 여부를 결정합니다.

Gate3[j].GpioMode[2]은 3 번째 बैं크(인덱스값 2)의 핀을 범용 I/O 가 전용 서보 기능으로 사용할지 여부를 결정합니다.

Gate3[j].GpioMode[3]은 4 번째 बैं크(인덱스값 3)의 핀을 범용 I/O 가 전용 서보 기능으로 사용할지 여부를 결정합니다.

Gate3[j].GpioMode[j]($j = 1\sim3$) 요소에는 각각 독립된 32 비트가 있습니다. 비트는 각각 개별 핀을 제어합니다. 각각의 비트와 1 로 설정했을 때의 GPIO 신호 및 0 으로 설정했을 때의 서보 신호는 다음 표와 같습니다. 하드웨어 채널 k 는 $j + 1$ 입니다. 다음 표는 각 비트에 대한 IC 핀의 사용을 표시합니다.

비트	GPIO 신호	서보 신호	비트	GPIO 신호	서보 신호
00	GPIO00	ENCAk -	16	GPIO16	AENAK
01	GPIO01	ENCAk+	17	GPIO17	OUTBk
02	GPIO02	ENCBk -	18	GPIO18	OUTCk
03	GPIO03	ENCBk+	19	GPIO19	OUTDk
04	GPIO04	ENCAk	20	GPIO20	ADCENCAk
05	GPIO05	ENCBk	21	GPIO21	ADCENCBk
06	GPIO06	ENCCK	22	GPIO22	ADCAMPAk
07	GPIO07	FAULTk	23	GPIO23	ADCAMPBk
08	GPIO08	HOMEk	24	GPIO24	PWMATk
09	GPIO09	MLIMk	25	GPIO25	PWMABk
10	GPIO10	PLIMk	26	GPIO26	PWMBTk
11	GPIO11	USERk	27	GPIO27	PWMBBk
12	GPIO12	FLAGTk	28	GPIO28	PWMCTk
13	GPIO13	FLAGUk	29	GPIO29	PWMCBk
14	GPIO14	FLAGVk	30	GPIO30	PWMDTk
15	GPIO15	FLAGWk	31	GPIO31	PWMDBk

각 **Gate3[j].GpioMode[j]** 요소의 적절한 설정은 IC 를 사용하는 하드웨어에 좌우됩니다. ACC-24E3 과 같은 축 인터페이스 하드웨어에서는 이러한 요소를 전용 서보 신호를 선택하는 것과 같이 \$0 으로 설정해야 합니다.

Gate3[j].GpioMode[j]는 쓰기 방지되었으므로 처음에 **Gate3[j].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].GpioPol[j] 게이트 3의 범용 I/O 극성

설명:	IC의 범용 I/O뱅크 극성
범위:	\$0..\$FFFFFFFF
단위:	비트 필드
기본값:	\$0

Gate3[i].GpioPol[j]는 IC 상의 선택된 뱅크의 범용 I/O 핀이 반전 I/O 인지 비반전 I/O 인지 여부를 결정합니다. 이는 32 비트 폭을 갖는 4 개의 설정 요소 배열입니다. 각 요소 **GpioPol[0]~GpioPol[3]**은 동일한 인덱스값 0~3의 GPIO 뱅크(각각 1 번째 뱅크~4 번째 뱅크)에 대응합니다.

요소 **Gate3[i].GpioPol[j]**는 각각 독립된 32 비트 0-31로 구성됩니다. 각각의 비트 n은 뱅크의 GPIO_n의 신호 방향을 제어합니다. 비트가 기본값 0이면 비반전(high 일 때 참) I/O로 사용됩니다. 비트가 1이면 반전(low 일 때 참) I/O로 사용됩니다. 신호 핀이 **Gate3[i].GpioMode[j]**가 대응하는 비트에 따라 GPIO로 사용하도록 선택된 경우에 범용 비트로 사용됩니다. 최초의 뱅크(인덱스값 0)는 항상 GPIO로 사용됩니다. 비트는 I/O가 입력 또는 출력으로 사용되는지와 관계없이 사용됩니다.

만약 비반전 I/O로 선택되고 동일한 뱅크의 **Gate3[i].GpioData[j]**와 일치하는 비트가 0이면 IC 신호 핀은 low 전압입니다. **Gate3[i].GpioData[j]**와 일치하는 비트가 1이면 IC 신호 핀은 high 전압입니다.

만약 반전 I/O로 선택되고 동일한 뱅크의 **Gate3[i].GpioData[j]**의 일치하는 비트가 0이면 IC 신호 핀은 high 전압입니다. **Gate3[i].GpioData[j]**의 일치하는 비트가 1이면 IC 신호 핀은 low 전압입니다.

해당 정의가 IC 자체의 I/O 극성에 관련된다는 점에 주의하십시오. 외부의 입출력 하드웨어의 별도의 반전 원인이 될 수도 있습니다.

Gate3[i].GpioPol[j]는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey**에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey**에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].HardwareClockCtrl 게이트 3의 하드웨어 클럭 제어

설명:	하드웨어 클럭 구성의 IC 제어 워드
범위:	\$0..\$FFFFFFFF
단위:	비트 필드
기본값:	\$55555403

Gate3[i].HardwareClockCtrl은 IC 중에서 사용되는 많은 클럭 주파수 생성을 위한 멀티채널 설정으로 구성되는 풀 워드 요소입니다. 다음 부분 워드 요소로 구성됩니다.

부분 워드 요소	비트	기능
Gate3[i].AdcAmpClockDiv	31 - 28	앰프용 ADC 클록 나눗셈 지수
Gate3[i].AdcEncClockDiv	27 - 24	엔코더용 ADC 클록 나눗셈 지수
Gate3[i].DacClockDiv	23 - 20	DAC 클록 나눗셈 지수
Gate3[i].PfmClockDiv	19 - 16	PFM 생성 클록 나눗셈 지수
Gate3[i].EncClockDiv	15 - 12	엔코더 샘플링 클록 나눗셈 지수
Gate3[i].FiltClockDiv	11 - 08	입력 필터 클록 나눗셈 지수
Gate3[i].ClockPol	07 - 05	DAC, ADC 클록 출력 극성
Gate3[i].ServoClockDiv	03 - 00	서보 클록 분주 계수

사용자는 부분 워드 요소의 설정과 설정 확인을 위해 보통 스크립트 환경에서 여기에 액세스합니다. Power PMAC은 효율화를 위해 풀 워드 요소로 저장하고 복원합니다.

C 프로그램에서는 사용자는 32 비트 워드의 풀 워드 요소에만 액세스해야 합니다.

풀 워드 요소의 값이 스크립트 환경 속에서 참조되는 경우에 부분 워드 요소 값이 10 진수로 통지되어도 Power PMAC은 16 진수 형식으로 값을 반환합니다.

Gate3[i].HardwareClockCtrl은 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey**에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey**에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

각각의 요소의 기능에 대한 자세한 설명은 각 부분 워드 요소 항목을 참조하십시오.

Gate3[i].MacroEnableA 게이트 3의 MACRO 뱅크 A 구성

설명: IC의 MACRO 뱅크 A의 유효 구성

범위: \$0 .. \$FFFFFFFF

단위: 비트 필드

기본값: \$0

Gate3[i].MacroEnableA는 IC의 MACRO 인터페이스("A")의 최초 설정의 유효한 구성 요소로 구성되는 풀 워드 요소입니다. 다음과 같은 구성 요소(이는 독립된 요소로 액세스할 수 없습니다)로 구성됩니다.

구성 요소	비트	16 진수 자리 #	기능
<i>MacroMasterNumA</i>	31 - 28	1	MACRO A 마스터 주소
<i>MacroSyncNodeA</i>	27 - 24	2	MACRO A 동기 패킷 노드 번호
<i>MacroNodeEnaA</i>	23 - 08	3 - 6	MACRO A 노드 유효 비트
(예약)	07 - 00	7 - 8	(추후 사용하기 위해 예약)

4 비트의 구성 요소 *MacroMasterNumA*는 IC의 16개의 "A" 노드 전부를 위한 마스터 주소 번호를 지정합니다. 4 비트의 노드 번호에 더해 각 노드 패킷의 헤더 바이트 일부로서 이 4 비트값을 송신합니다.

4 비트의 구성 요소 *MacroSyncNodeA* 는 "동기 패킷" 노드 번호를 지정합니다. 이 IC 가 링의 동기 마스터로 동작하지 않고 동일 장치 내의 다른 IC 로부터 위상 클록 신호를 직접 수신하지 않을 때 이 노드에서의 데이터 패킷 수신은 내부 동기 신호를 생성하여 링 타이밍에 올바르게 고정되고 유지됩니다.

16 비트의 구성 요소 *MacroNodeEnaA* 는 어떤 "A" 노드가 링 상에서 유효가 될지 지정합니다. 비트 *n* 으로 노드 *n* 을 제어합니다. 비트가 0 이면 노드를 무효로 설정하고 비트가 1 이면 노드가 유효가 됩니다.

Gate3[i].MacroEnableA 는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].MacroEnableB 게이트 3 의 MACRO BANK B 구성

설명: IC 의 MACRO BANK B 의 유효 구성

범위: \$0..\$FFFFFFFF

단위: 비트 필드

기본값: \$0

Gate3[i].MacroEnableB 는 IC 의 MACRO 인터페이스("B")의 2 번째 설정의 유효한 구성 요소로 구성되는 풀 워드 요소입니다. 다음과 같은 구성 요소(이는 독립된 요소로 액세스할 수 없습니다)로 구성됩니다.

구성 요소	비트	16 진수 자리 #	기능
<i>MacroMasterNumB</i>	31 - 28	1	MACRO B 마스터 주소
<i>MacroSyncNodeB</i>	27 - 24	2	MACRO B 동기 패킷 노드 번호
<i>MacroNodeEnaB</i>	23 - 08	3 - 6	MACRO B 노드 유효 비트
(예약)	07 - 00	7 - 8	(추후 사용하기 위해 예약)

4 비트의 구성 요소 *MacroMasterNumB* 는 IC 의 16 개의 "B" 노드 전부를 위한 마스터 주소 번호를 지정합니다. 4 비트의 노드 번호에 더해 각 노드 패킷의 헤더 바이트 일부로서 이 4 비트값을 송신합니다.

4 비트의 구성 요소 *MacroSyncNodeB* 는 "동기 패킷" 노드 번호를 지정합니다. 이 IC 가 링의 동기 마스터로 동작하지 않고 동일 장치 내의 다른 IC 로부터 위상 클록 신호를 직접 수신하지 않을 때 이 노드에서의 데이터 패킷 수신은 내부 동기 신호를 생성하여 링 타이밍에 올바르게 고정되고 유지됩니다.

16 비트의 구성 요소 *MacroNodeEnaB* 는 어느 "B" 노드가 링 상에서 유효가 될지 지정합니다. 비트 *n* 으로 노드 *n* 을 제어합니다. 비트가 0 이면 노드를 무효로 설정하고 비트가 1 이면 노드가 유효가 됩니다.

Gate3[i].MacroEnableB 는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].MacroModeA 게이트 3 의 매크로 A 모드

설명: IC 의 MACRO BANK A 의 상태 및 제어

범위: \$0..\$FFFFFFFF

단위: 비트 필드

기본값: \$0

Gate3[i].MacroModeA 는 IC 의 MACRO 인터페이스("A")의 최초 설정의 제어 및 상태의 구성 요소로 구성되는 풀 워드 요소입니다. 다음과 같은 구성 요소(이는 독립된 요소로 액세스할 수 없습니다)로 구성됩니다.

구성 요소	비트	16 진수 자리 #	기능
(예약)	31 - 24	1 - 2	(추후 사용하기 위해 예약)
<i>MacroMasterChkDisA</i>	23 - 16	3 - 4	MACRO A 마스터 확인 유효
<i>MacroSyncEnaA</i>	15	5	MACRO A 위상 클록 동기 유효
<i>MacroSyncRcvdA</i>	14	5	MACRO A 동기 패킷 수신 상태
<i>MacroStationTypeA</i>	13 - 12	5	MACRO A 스테이션 타입
<i>MacroUnderrunErrA</i>	11	6	MACRO A 데이터 언더런 에러 Status
<i>MacroParityErrA</i>	10	6	MACRO A 패리티/CRC 에러 Status
<i>MacroCodeErrA</i>	09	6	MACRO A 바이트 코드 에러 Status
<i>MacroOverrunErrA</i>	08	6	MACRO A 데이터 오버런 에러 Status
(예약)	07 - 00	7 - 8	(추후 사용하기 위해 예약)

8 비트의 구성 요소 *MacroMasterChkDisA* 는 IC 의 "A" 노드의 상위 번호 8 개 중에서 마스터 번호를 확인할지 여부를 지정합니다. 구성 요소의 비트 n 은(풀 워드 요소의 비트 n+16) IC 의 노드 n+8 을 제어합니다. 비트가 0 이면 IC 는 일치하는 노드 번호에 대한 수신 패킷의 마스터 번호를 확인합니다. 따라서 패킷은 링을 통해 1:1 통신에 사용할 수 있습니다. 비트가 1 이면 IC 는 일치하는 노드 번호에 대한 수신 패킷의 마스터 번호를 확인하지 않습니다. 따라서 패킷을 "브로드캐스트" 목적으로 사용할 수 있습니다.

1 비트의 구성 요소 *MacroSyncEnaA* 는 지정된 "동기 패킷"의 수신에 따라 IC 의 위상 클록이 동기될지 여부를 지정합니다. 0 으로 설정되면 동기는 실시되지 않습니다. 1 로 설정되면 IC 의 위상 클록 타이머는 링 상의 동기 패킷의 수신 타이밍과 동기됩니다. IC 가 링의 동기 마스터인 경우에 또는 동일 장치 내의 다른 IC 로부터 위상 클록을 수신하는 경우에 이 비트는 사용되지 않습니다.

1 비트의 구성 요소 *MacroSyncRcvdA* 는 지정된 동기 패킷이 수신된 경우에 1 로 설정되는 읽기 전용 Status 비트입니다. 이 레지스터가 읽히면 자동으로 0 으로 클리어됩니다. 따라서 마지막으로 레지스터가 읽힌 다음 동기 패킷이 수신되었는지 여부를 프로세서에 알릴 수 있습니다.

2 비트의 구성 요소 *MacroStationTypeA* 는 MACRO 링 상의 이 IC 를 포함하는 스테이션의 기능을 지정합니다. 지정 가능한 기능은 다음과 같습니다.

- 0: 슬레이브
- 1: 마스터
- 2: (추후 사용하기 위해 예약)
- 3: 마스터를 동기시킴

1 비트의 구성 요소 *MacroUnderrunErrA* 는 읽기 전용 Status 비트이며 IC 가 너무 짧은 바이트 수의 데이터 패킷을 수신하는 경우에 1 로 설정됩니다. 이 레지스터가 읽히면 자동으로 0 으로 클리어됩니다. 따라서 마지막으로 레지스터가 읽힌 다음 언더런 에러가 발생했는지 여부를 프로세서에 알릴 수 있습니다.

1 비트의 구성 요소 *MacroParityErrA* 는 읽기 전용 Status 비트이며 IC 가 패리티 또는 CRC 확인 에러가 있는 데이터 패킷을 수신하는 경우에 1 로 설정됩니다. 이 레지스터가 읽히면 자동으로 0 으로 클리어됩니다. 따라서 마지막으로 레지스터가 읽힌 다음 패리티/CRC 에러가 발생했는지 여부를 프로세서에 알릴 수 있습니다.

1 비트의 구성 요소 *MacroCodeErrA* 는 읽기 전용 Status 비트이며 IC 가 잘못된 코드 바이트가 있는 데이터 패킷을 수신하는 경우에 1 로 설정됩니다. 이 레지스터가 읽히면 자동으로 0 으로 클리어됩니다. 따라서 마지막으로 레지스터가 읽힌 다음 코드 에러가 발생했는지 여부를 프로세서에 알릴 수 있습니다.

1 비트의 구성 요소 *MacroOverrunErrA* 는 읽기 전용 Status 비트이며 IC 가 너무 많은 바이트 수의 데이터 패킷을 수신하는 경우에 1 로 설정됩니다. 이 레지스터가 읽히면 자동으로 0 으로 클리어됩니다. 따라서 마지막으로 레지스터가 읽힌 다음 오버런 에러가 발생했는지 여부를 프로세서에 알릴 수 있습니다.

Gate3[i].MacroModeA 의 제어 컴포넌트는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].MacroModeB 게이트 3 의 매크로 B 모드

설명: IC 의 MACRO बैंक B 의 상태 및 제어

범위: \$0..\$FFFFFFFF

단위: 비트 필드

기본값: \$0

Gate3[i].MacroModeB 는 IC 의 MACRO 인터페이스("B")의 최초 설정의 제어 및 상태의 구성 요소로 구성되는 풀 워드 요소입니다. 다음과 같은 구성 요소(이는 독립된 요소로 액세스할 수 없습니다)로 구성됩니다.

구성 요소	비트	16 진수 자리 #	기능
(예약)	31 - 24	1 - 2	(추후 사용하기 위해 예약)
<i>MacroMasterChkDisB</i>	23 - 16	3 - 4	MACRO B 마스터 확인 유효
<i>MacroSyncEnaB</i>	15	5	MACRO B 위상 클록 동기 유효
<i>MacroSyncRcvdB</i>	14	5	MACRO MacroModeA 동기 패킷 상태 수신
<i>MacroStationTypeB</i>	13 - 12	5	MACRO B 스테이션 타입
<i>MacroUnderrunErrB</i>	11	6	MACRO B 데이터 언더런 에러 Status
<i>MacroParityErrB</i>	10	6	MACRO B 패리티/CRC 에러 Status
<i>MacroCodeErrB</i>	09	6	MACRO B 바이트 코딩 시 에러 Status
<i>MacroOverrunErrB</i>	08	6	MACRO B 데이터 오버런 에러 Status
(예약)	07 - 00	7 - 8	(추후 사용하기 위해 예약)

8 비트의 구성 요소 *MacroMasterChkDisB* 는 IC 의 "B" 노드의 상위 번호 8 개 중에서 마스터 번호를 확인할지 여부를 지정합니다. 구성 요소의 비트 n 은(이는 풀 워드 요소의 비트 $n+16$ 입니다) IC 의 노드 $n+8$ 을 제어합니다. 비트가 0 이면 IC 는 일치하는 노드 번호에 대한 수신 패킷의 마스터 번호를 확인합니다. 따라서 패킷은 링을 통해 1:1 통신에 사용할 수 있습니다. 비트가 1 이면 IC 는 일치하는 노드 번호에 대한 수신 패킷의 마스터 번호를 확인하지 않습니다. 따라서 패킷을 "브로드캐스트" 목적으로 사용할 수 있습니다.

1 비트의 구성 요소 *MacroSyncEnaB* 는 지정된 "동기 패킷"의 수신에 의해 IC 의 위상 클록이 동기될지 여부를 지정합니다. 0 으로 설정되면 동기는 실시되지 않습니다. 1 로 설정되면 IC 의 위상 클록 타이머는 링 상의 동기 패킷의 수신 타이밍과 동기됩니다. IC 가 링의 동기 마스터인 경우에 또는 동일 장치 내의 다른 IC 로부터 위상 클록을 수신하는 경우에 이 비트는 사용되지 않습니다.

1 비트의 구성 요소 *MacroSyncRcvdB* 는 지정된 동기 패킷이 수신된 경우에 1 로 설정되는 읽기 전용 Status 비트입니다. 이 레지스터가 읽히면 자동으로 0 으로 클리어됩니다. 따라서 마지막으로 레지스터가 읽힌 다음 동기 패킷이 수신되었는지 여부를 프로세서에 알릴 수 있습니다.

2 비트의 구성 요소 *MacroStationTypeB* 는 MACRO 링 상의 이 IC 를 포함하는 스테이션의 기능을 지정합니다. 지정 가능한 기능은 다음과 같습니다.

- 0: **슬레이브**
- 1: **마스터**
- 2: **(추후 사용하기 위해 예약)**
- 3: **마스터를 동기시킴**

1 비트의 구성 요소 *MacroUnderrunErrB* 는 읽기 전용 Status 비트이며 IC 가 너무 짧은 바이트 수의 데이터 패킷을 수신하는 경우에 1 로 설정됩니다. 이 레지스터가 읽히면 자동으로 0 으로 클리어됩니다. 따라서 마지막으로 레지스터가 읽힌 다음 언더런 에러가 발생했는지 여부를 프로세서에 알릴 수 있습니다.

1 비트의 구성 요소 **MacroParityErrB** 는 읽기 전용 **Status** 비트이며 IC 가 패리티 또는 CRC 확인 에러가 있는 데이터 패킷을 수신하는 경우에 1 로 설정됩니다. 이 레지스터가 읽히면 자동으로 0 으로 클리어됩니다. 따라서 마지막으로 레지스터가 읽힌 다음 패리티/CRC 에러가 발생했는지 여부를 프로세서에 알릴 수 있습니다.

1 비트의 구성 요소 **MacroCodeErrB** 는 읽기 전용 **Status** 비트이며 IC 가 잘못된 코드 바이트가 있는 데이터 패킷을 수신하는 경우에 1 로 설정됩니다. 이 레지스터가 읽히면 자동으로 0 으로 클리어됩니다. 따라서 마지막으로 레지스터가 읽힌 다음 코드 에러가 발생했는지 여부를 프로세서에 알릴 수 있습니다.

1 비트의 구성 요소 **MacroOverrunErrB** 는 읽기 전용 **Status** 비트이며 IC 가 너무 많은 바이트 수의 데이터 패킷을 수신하는 경우에 1 로 설정됩니다. 이 레지스터가 읽히면 자동으로 0 으로 클리어됩니다. 따라서 마지막으로 레지스터가 읽힌 다음 오버런 에러가 발생했는지 여부를 프로세서에 알릴 수 있습니다.

Gate3[i].MacroModeB 의 제어 컴포넌트는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].PfmClockDiv 게이트 3 의 펄스 주파수 변조 클록 분주

설명: IC 의 펄스 주파수 변조의 클록 주파수 제어

범위: 0 ..15

단위: 없음

기본값: 5(3.125MHz)

Gate3[i].PfmClockDiv 는 IC 채널의 펄스 주파수 변조 회로를 제어하는 클록 신호의 주파수를 지정합니다. 각 PFM 클록 사이클에서 24 비트의 활성 위상 D 명령값은 24 비트의 어큐뮬레이터에 더해집니다. 어큐뮬레이터가 "롤 오버"될 때마다 출력 펄스가 발생합니다. 따라서 펄스 출력 주파수는 명령값(빈번하게 변경할 수 있음) 및 PFM 클록 주파수(보통 어플리케이션에 의해 고정됨) 양쪽에 비례합니다.

이 요소는 내부 100MHz 클록 신호를 2 로 몇 번 나눌지 지정하여 주파수를 제어합니다. 클록 주파수를 구하는 식은 다음과 같습니다.

$$f_{PfmClk} = \frac{100MHz}{2^{PfmClockDiv}}$$

다음 표는 **Gate3[i].PfmClockDiv** 에 할 수 있는 설정과 지원하는 주파수를 나타냅니다.

설정	제수	주파수	설정	제수	주파수
0	1	100MHz	8	256	390.6kHz
1	2	50MHz	9	512	195.3kHz
2	4	25MHz	10	1,024	97.65kHz
3	8	12.5MHz	11	2,048	48.82kHz
4	16	6.25MHz	12	4,096	24.41kHz
5	32	3.125MHz	13	8,192	12.21kHz
6	64	1.562MHz	14	16,384	6.104kHz
7	128	781.2kHz	15	32,768	3.052kHz

보통 최대 및 최소 펄스 출력 주파수 모두 가능한 클럭 주파수를 선택합니다.

Gate3[i].PfmClockDiv 는 풀 워드 요소 **Gate3[i].HardwareClockCtrl** 의 비트 16-19 의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].PhaseClockDiv 게이트 3 의 위상 클럭 입력 분주 계수

설명: IC 의 위상 클럭 입력 분주 계수

범위: 0 ..3

단위: 없음

기본값: 0

Gate3[i].PhaseClockDiv 는 외부 입력의 위상 클럭 주파수와 내부 위상 클럭 주파수의 분주 계수를 지정합니다. 외부 입력 위상 클럭 주파수는 (**Gate3[i].PhaseClockDiv+1**) 계수로 나누어져 내부 위상 클럭 주파수를 얻을 수 있습니다.

대부분의 어플리케이션은 외부 입력과 내부 위상 클럭 주파수가 동일해지도록 0 으로 설정합니다. 그러나 PWM 출력 주파수의 유효 범위가 내부 위상 클럭에 의해 결정되고 프로세서의 위상 인터럽트 주파수가 외부 입력 위상 클럭에 의해 결정되므로 해당 요소는 주파수 구성에 한층 유연성을 부여합니다.

Gate3[i].PhaseServoDir 의 비트 0(값 1)이 1 로 설정되는 경우에만 **Gate3[i].PhaseClockDiv** 가 사용되고 내부에서 생성하는 위상 클럭 신호 대신 외부에서 생성된 위상 클럭 신호를 입력하고 출력하도록 IC 에 지정합니다. 일반적으로 Power PMAC 시스템의 모든 IC 는 정확하게 동기를 유지하기 위해 단일 위상 클럭 신호를 사용해야 합니다.

Gate3[i].PhaseClockDiv 는 풀 워드 요소 **Gate3[i].PhaseServoClockCtrl** 의 비트 12-13 의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].PhaseClockMult 게이트 3 의 위상 클록 출력 배율

설명:	IC 의 위상 클록 출력 배율
범위:	0 ..3
단위:	없음
기본값:	0

Gate3[i].PhaseClockMult 는 내부의 위상 클록 주파수와 출력되는 위상 클록 주파수 간의 배율을 지정합니다. 내부 위상 클록 주파수에 (**Gate3[i].PhaseClockMult+1**) 계수를 곱해서 출력 위상 클록 주파수를 얻을 수 있습니다.

대부분의 어플리케이션은 내부 위상 클록 주파수와 출력 주파수가 동일해지도록 0 으로 설정합니다. 그러나 PWM 출력 주파수의 유효 범위가 내부 위상 클록에 의해 결정되고 프로세서의 위상 인터럽트 주파수가 출력 위상 클록에 의해 결정되므로 해당 요소는 주파수 구성에 한층 유연성을 부여합니다.

Gate3[i].PhaseServoDir 의 비트 0(값 1)이 0 으로 설정되는 경우에만 **Gate3[i].PhaseClockMult** 가 사용되고 외부에서 생성된 위상 클록 신호를 입력하는 대신 내부에서 위상 클록 신호를 생성하여 출력하도록 IC 에 지정합니다. 일반적으로 Power PMAC 시스템의 모든 IC 는 정확하게 동기를 유지하기 위해 단일 위상 클록 신호를 사용해야 합니다.

Gate3[i].PhaseClockMult 는 풀 워드 요소 **Gate3[i].PhaseServoClockCtrl** 의 비트 14-15 의 부분적인 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].PhaseFreq 게이트 3 의 내부 위상 클록 주파수

설명:	IC 의 내부 위상 클록 주파수
범위:	35.76 ..2,343,463
단위:	헤르츠
기본값:	9035.69

Gate3[i].PhaseFreq 는 헤르츠 단위로 IC 의 내부 위상 클록 신호의 주파수를 지정합니다. **Gate3[i].PhaseServoDir** 의 비트 0(값 1)이 0 으로 설정되는 경우에 외부에서 생성된 클록 신호를 입력하는 대신 내부에서 위상 클록 신호를 생성하여 출력하도록 IC 에 지정하고 또한 시스템 위상 클록의 주파수로 사용됩니다. 일반적으로 Power PMAC 시스템의 모든 IC 는 정확하게 동기를 유지하기 위해 단일 위상 클록 신호를 사용해야 합니다.

Gate3[i].PhaseClockMult 로 지정되는 IC 로부터의 페이즈 클록 신호 출력은(이는 프로세서의 "페이즈 인터럽트"를 부여함) 내부 페이즈 클록 주파수의 1~4 배입니다. 그러나 IC 채널 상에서 생성되는 PWM 주파수의 가능한 범위는 출력 신호 주파수가 아니라 내부 위상 클록 주파수에 좌우됩니다. **Gate3[i].Chan[j].PwmFreqMult** 로 지정되는 PWM 주파수는 내부 위상 클록 주파수의 0.5~4.0 배입니다.

Gate3[i].PhaseServoDir의 비트 0(값 1)을 1로 설정하는 경우에 IC는 외부 클록 신호를 입력하게 되므로 **Gate3[i].PhaseFreq**를 **Gate3[i].PhaseClockDiv**로 분주된 외부 클록 주파수에 가능한 한 가까운 값으로 설정해야 합니다. 이 경우에 내부 회로를 구동하기 위해 여전히 **Gate3[i].PhaseFreq**를 사용하지만 페이즈 록 루프(PLL) 회로를 통해(분주하여) 외부 클록 신호와 주기가 유지됩니다.

Gate3[i].PhaseFreq는 12비트의 가수부 및 4비트의 지수부를 갖는 사전에 스케일링된 16비트의 부동 소수점 값입니다. 이는 풀 워드 요소 **Gate3[i].PhaseServoClockCtrl**의 비트 16-31의 부분 구성 요소입니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey**에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey**에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].PhaseServoClockCtrl 게이트 3의 위상/서보 클록 제어

설명: 위상과 서보 클록의 IC 제어 워드

범위: \$0..\$FFFFFFFF

단위: 비트 필드

기본값: \$7F950C00

Gate3[i].PhaseServoCtrl은 IC의 위상 및 서보 클록 생성의 멀티채널 설정으로 구성되는 풀 워드 요소입니다. 다음 부분 워드 요소로 구성됩니다.

부분 워드 요소	비트	기능
PhaseFreq	31 - 16	내부 위상 클록 주파수
PhaseClockMult	15 - 14	위상 클록의 출력 배율
PhaseClockDiv	13 - 12	위상 클록의 입력 분주 계수
PhaseServoDir	11 - 10	위상 및 서보 클록 방향 제어
(예외)	09 - 08	(내부 상황 비트)
EncLatchDelay	07 - 00	엔코더 카운터 값을 래치할 때의 지연

부분 워드 요소 **PhaseFreq**는 비트 28-31의 4비트 지수부(기수 2) 및 비트 16-27의 12비트 가수부(암목의 "1" 포함)로 이루어지는 정규화된 16비트 부동 소수점 값으로 저장됩니다. 저장되는 값은 요소에 입력된 값에서 35.76(600,000,000/16,777,216) 계수로 나누어집니다.

사용자는 부분 워드 요소의 설정과 설정 확인을 위해 보통 스크립트 환경에서 여기에 액세스합니다. Power PMAC은 효율화를 위해서 풀 워드 요소로 저장 및 복원합니다.

C 프로그램에서는 사용자는 풀 워드 요소만 액세스해야 합니다.

풀 워드 요소의 값이 스크립트 환경 속에서 참조되는 경우에 부분 워드 요소 값이 10 진수로 통지되어도 Power PMAC 은 16 진수 형식으로 값을 반환합니다.

Gate3[i].PhaseServoCtrl 은 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

각각의 요소의 기능에 대한 자세한 설명은 각 부분 워드 요소 항목을 참조하십시오.

Gate3[i].PhaseServoDir 게이트 3 의 위상/서보 클록 방향 제어

설명: IC 의 위상 및 서보 클록 방향 제어

범위: 0 ..3

단위: 비트 필드

기본값: 0 또는 3(다시 초기화하면 자동 설정)

Gate3[i].PhaseServoDir 는 IC 의 내부에서 생성된 위상과 서보 클록 신호를 사용해서 출력할지 또는 외부에서 생성된 위상과 서보 클록 신호를 입력해서 사용할지를 지정합니다. 위상 클록 방향을 제어하는 비트 0(값 1) 및 서보 클록 방향을 제어하는 비트 1의 2비트값입니다. 0의 비트값으로는 내부에 생성된 클록 신호의 사용을 지정하며 1의 비트값으로는 외부 클록 신호의 사용을 지정합니다. 다음 4 가지 값이 가능합니다.

- **Gate3[i].PhaseServoDir** = 0: 내부 위상 클록, 내부 서보 클록
- **Gate3[i].PhaseServoDir** = 1: 외부 위상 클록, 내부 서보 클록
- **Gate3[i].PhaseServoDir** = 2: 내부 위상 클록, 외부 서보 클록
- **Gate3[i].PhaseServoDir** = 3: 외부 위상 클록, 외부 서보 클록

하나의 IC 중에서 2 개의 클록 신호가 상반되는 방향에서 사용되는 것은 매우 드문 일입니다. 따라서 일반적으로 사용되는 값은 0 과 3 뿐입니다.

어떤 Power PMAC 시스템에서도 시스템의 위상과 서보 클록의 신호에 하나의 소스가 있어야 합니다. **Gate1**, **Gate2** 또는 **Gate3** IC 중 하나 또는 시스템 외의 소스입니다. 다른 모든 IC 는 동기화를 유지하기 위해 이러한 클록 신호를 입력해야 합니다.

Gate3[i].PhaseServoDir 은 풀 워드 요소 **Gate3[i].PhaseServoClockCtrl** 의 비트 10-11 을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].ResolverCtrl 게이트 3의 리졸버 제어

설명: 리졸버 여자 구성을 위한 IC의 제어 워드

범위: \$0..\$FFFFFFFF

단위: 비트 필드

기본값: \$0

Gate3[i].ResolverCtrl은 리졸버 인터페이스용 멀티채널 설정으로 구성되는 풀 워드 요소입니다. 다음과 같은 구성 요소(이는 독립된 요소로 액세스할 수 없습니다)로 구성됩니다.

	비트	16진수 자리 #	기능
<i>ResolverExciteShift</i>	31 - 24	1 - 2	리졸버 여기 신호 위상 시프트
<i>ResolverExciteGain</i>	23 - 22	3	리졸버 여기 위상 제어
<i>ResolverExciteFreq</i>	21 - 20	3	리졸버 여기 주파수 제어
(예약)	19 - 00	4 - 8	(추후 사용하기 위해 예약)

8비트 구성 요소 *ResolverExciteShift*는 여기 신호의 위상 시프트(지연)를 지정합니다. 이는 여기 신호 주기(구성 요소 *ResolverExciteFreq*로 설정됨)의 1/512 단위로 0~255의 값으로 설정할 수 있습니다. 이 구성 요소는 리턴 신호(데이터 구조체 요소 **Gate3[i].Chan[j].SumOfSquares**를 읽어서 확인 가능)의 크기를 최대한이 되도록 보통 실험적으로 설정합니다.

2비트 구성 요소 *ResolverExciteGain*은 출력의 크기를 지정합니다. 지정 가능한 값의 크기는 다음과 같습니다.

- 0: **최대한의 1/4**
- 1: **최대한의 1/2**
- 2: **최대한의 3/4**
- 3: **최대한**

2비트 구성 요소 *ResolverExciteFreq*는 위상 클록 주파수에서 도출되는 출력의 주파수를 지정합니다. 지정 가능한 값의 주파수는 다음과 같습니다.

- 0: **위상 클록 주파수**
- 1: **위상 클록 주파수/2**
- 2: **위상 클록 주파수/4**
- 3: **위상 클록 주파수/6**

Gate3[i].ResolverCtrl은 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey**에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey**에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].SerialEncCtrl 게이트 3의 시리얼 엔코더 제어

설명: 시리얼 엔코더 구성의 IC 제어 워드

범위: \$0..\$FFFFFFFF

단위: 비트 필드

기본값: \$F4000000

Gate3[i].SerialEncCtrl 은 IC의 시리얼 엔코더 인터페이스용 멀티채널 설정으로 구성되는 풀 워드 요소입니다. 다음과 같은 요소(독립된 요소로 액세스할 수 없습니다)로 구성됩니다.

구성 요소	비트	16진수 자리 #	기능
<i>SerialClockMDiv</i>	31 - 24	1 - 2	시리얼 클록의 선형 분주 계수
<i>SerialClockNDiv</i>	23 - 20	3	시리얼 클록 지수부의 분주 계수
(예약)	19 - 18	4	(추후 사용하기 위해 예약)
<i>SerialTrigClockSel</i>	17	4	시리얼 트리거 소스 선택
<i>SerialTrigEdgeSel</i>	16	4	시리얼 트리거 소스 엣지 선택
<i>SerialTrigDelay</i>	15 - 08	5 - 6	소스 엣지에서 시리얼 트리거 지연
<i>SerialProtocol</i>	07 - 00	7 - 8	시리얼 엔코더 프로토콜 선택

구성 요소 *SerialClockMDiv* 는 IC 고정 100MHz 클록 주파수에서 어떻게 중간 클록 주파수를 생성할지 제어합니다. 뒤에서 설명하는 것처럼 산출 결과인 시리얼 엔코더의 클록 주파수는 이 중간 클록 주파수에서 구성 요소 *SerialClockNDiv* 를 사용하여 생성됩니다.

중간 클록 주파수를 구하는 식은 다음과 같습니다.

$$f_{\text{int}}(\text{MHz}) = \frac{100}{M + 1}$$

여기서 M은 짧은 정수인 *SerialClockMDiv* 를 나타냅니다. 이 8비트의 구성 요소는 0에서 255의 값으로 설정할 수 있습니다. 따라서 결과적으로 발생하는 중간 클록 주파수의 범위는 100MHz부터 392kHz까지입니다.

구성 요소 *SerialClockNDiv* 는 *SerialClockMDiv* 로 설정된 중간 클록 주파수에서 생성되는 최종 시리얼 엔코더의 클록 주파수를 제어합니다. 최종 주파수를 구하는 식은 다음과 같습니다.

$$f_{\text{ser}}(\text{MHz}) = \frac{f_{\text{int}}(\text{MHz})}{2^N} = \frac{100}{(M + 1) * 2^N}$$

여기서 N은 짧은 정수인 *SerialClockNDiv* 를 나타냅니다. 이 4비트의 구성 요소는 0~15의 값으로 설정할 수 있습니다. 따라서 결과적으로 발생하는 2^N 의 제수는 1~32,768의 값이 됩니다.

명시적인 클럭 신호를 갖춘 대부분의 시리얼 엔코더의 프로토콜의 경우 얻어진 주파수가 IC 에서 출력되는 클럭 신호의 주파수입니다. 명시적인 클럭 신호가 없는 "자가 클럭" 프로토콜 또는 약간의 사이클 지연 보상이 있는 클럭 프로토콜을 위해 이 주파수는 입력 샘플링 주파수가 되어 입력 비트 전송 속도보다 20~25 배 높아집니다. 개별 프로토콜에 대한 자세한 내용은 설명서를 참조하십시오.

구성 요소 *SerialTrigClockSel* 은 엔코더를 트리거하는 Power PMAC 클럭 신호를 제어합니다. 이 싱글 비트 구성 요소를 0 으로 설정하면 엔코더는 위상 클럭에 의해 트리거됩니다. 1 로 설정하면 엔코더는 서보 클럭에 의해 트리거됩니다. 엔코더 피드백이 전류(commutation)의 회전자 각도 피드백에 필요한 경우에 위상 클럭에 의해 트리거되어야 합니다. 전류(commutation)가 아니면 서보 클럭에 의해 트리거할 수 있습니다.

구성 요소 *SerialTrigEdgeSe* 는 *SerialTrigClockSel* 에서 선택된 클럭 신호(위상 또는 서보)의 어느 엣지에서 트리거 프로세스를 시작할지 제어합니다. 이 싱글 비트 구성 요소를 0 으로 설정하는 경우에 트리거 프로세스는 상승 엣지에서 시작합니다. 1 로 설정하는 경우에 트리거 프로세스는 하강 엣지에서 시작합니다.

관련된 위상 또는 서보 클럭 신호의 하강 엣지 직후에 프로세서가 엔코더 데이터 읽기 동작을 시작할 수 있도록 인터럽트가 발생하며 Power PMAC 소프트웨어는 엔코더 데이터가 사용 가능하다고 판단합니다. 트리거에서는 지연이 최소화되는 것이 바람직하므로 하강 엣지 전에 완전히 데이터를 전송할 수 있다면 클럭의 상승 엣지에서 트리거하는 것이 더 좋습니다. 이것이 불가능한 경우에는 트리거 프로세스를 시작하기 위해 하강 엣지를 사용해야 합니다.

구성 요소 *SerialTrigDelay* 는 시리얼 엔코더의 클럭 단위로 지정된 클럭 엣지에서 실제로 시작하는 엔코더 응답을 트리거하는 출력 신호의 실제 시작까지의 지연을 지정합니다. 엔코더를 트리거하는 타이밍과 Power PMAC 이 읽는 타이밍의 지연을 최소한으로 하기 위해 0 이 아닌 값을 사용할 수 있습니다. 더 작은 시간 지연은 서보 루프의 위상 지연을 축소하여 루프의 안정 특성을 개선합니다.

구성 요소 *SerialProtocol* 은 IC 의 모든 채널에 어느 시리얼 엔코더의 프로토콜을 선택할지 제어합니다. 이 4비트의 구성 요소는 0~15 의 값으로 설정할 수 있습니다. 다음 표에 이 구성 요소의 각 값으로 선택된 프로토콜이 나와 있습니다.

값	프로토콜	값	프로토콜	값	프로토콜	값	프로토콜
0	없음	4	Hiperface	8	Panasonic	12	(예약)
1	SPI	5	시그마 I	9	Mitutoyo	13	(예약)
2	SSI	6	시그마 II	10	Kawasaki	14	(예약)
3	EnDat	7	Tamagawa	11	(예약)	15	SW 제어

Gate3[i].SerialEncCtrl 은 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

자세한 내용은 시리얼 엔코더를 처리하기 위해 사용되는 특별한 인터페이스 보드(예를 들어 ACC-24E3)의 하드웨어 참조 설명서에 기재됩니다. 다음은 지원되는 각 프로토콜에 공통되는 설정입니다.

SPI 프로토콜

SPI 엔코더용 **Gate3[i].SerialEncCtrl** 의 전형적인 설정은 다음 목록과 같습니다.

SerialClockMDiv: $= (100/f_{\text{bit}}) - 1$ // 시리얼 클럭 주파수 = 비트 전송 주파수
SerialClockNDiv: = 0 // 주파수가 400kHz 이하라면 더 나눌 필요는 없음
SerialTrigClockSel: = 0 // 가능하면 위상 클럭 사용
SerialTrigEdgeSel: = 0 // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 축소하기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$01 // SPI 프로토콜 선택

예를 들어 4MHz 의 비트 전송률에는 **SerialClockMDiv** $= (100/4) - 1 = 24(\$18)$ 을 설정합니다. 또한 지연 없이 위상 클럭의 상승 엣지에서 트리거하기 위해 **Gate3[i].SerialEncCtrl** 을 \$18000001 로 설정합니다.

1				8				0				0				0				0				0				1			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	0	0	0	0	0	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
SerialClockMDiv								SerialClockNDiv				-- -- TC TE				SerialTrigDelay								SerialProtocol							

SSI 프로토콜

SSI 엔코더용 **Gate3[i].SerialEncCtrl** 의 전형적인 설정은 다음 목록과 같습니다.

SerialClockMDiv: $= (100/f_{\text{bit}}) - 1$ // 시리얼 클럭 주파수 = 비트 전송 주파수
SerialClockNDiv: = 0 // 주파수가 400kHz 이하라면 더 나눌 필요는 없음
SerialTrigClockSel: = 0 // 가능하면 위상 클럭 사용
SerialTrigEdgeSel: = 0 // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 축소하기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$02 // SSI 프로토콜 선택

예를 들어 2.5MHz 의 비트 전송률에는 **SerialClockMDiv** $= (100/2.5) - 1 = 39(\$23)$ 를 설정합니다. 또한 지연 없이 위상 클럭의 상승 엣지에서 트리거하기 위해 **Gate3[i].SerialEncCtrl** 을 \$23000002 로 설정합니다.

2		3		0		0		0		0		0		2																	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	1	1	0	0	0	0	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
SerialClockMDiv								SerialClockNDiv -- -- TC TE								SerialTrigDelay								SerialProtocol							

EnDat 2.1/2.2 는 프로토콜

EnDat 엔코더용 **Gate3[i].SerialEncCtrl** 의 전형적인 설정은 다음 목록과 같습니다. 지연 보상의 탑재를 가능하게 하려면 외부 클럭 주파수(이는 비트 전송 주파수입니다)보다 시리얼 클럭 주파수를 25 배 높게 합니다. 장거리에서 높은 주파수 변환이 가능해집니다.

SerialClockMDiv: $= (4/f_{\text{bit}}) - 1$ // 시리얼 클럭 주파수 = 25x 비트 전송 주파수
SerialClockNDiv: = 0 // 그 이상 나누지 않음
SerialTrigClockSel: = 0 // 가능하면 위상 클럭 사용
SerialTrigEdgeSel: = 0 // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 축소하기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$03 // EnDat 프로토콜 선택

예를 들어 2.0MHz 의 비트 전송률에는 **SerialClockMDiv** = (4/2) - 1 = 1(\$01)을 설정합니다. 또한 지연 없이 위상 클록의 상승 엣지에서 트리거하기 위해 **Gate3[i].SerialEncCtrl** 을 \$01000003 으로 설정합니다.

0				1				0				0				0				0				0				3			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	0	0	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
SerialClockMDiv								SerialClockNDiv -- -- TC TE								SerialTrigDelay								SerialProtocol							

Hiperface 프로토콜

Hiperface 엔코더용 **Gate3[i].SerialEncCtrl** 의 전형적인 설정은 다음 목록과 같습니다. 명시적인 클록 신호가 없으므로 시리얼 클록 주파수는 입력 데이터 스트림에서 "오버샘플링"하도록 비트 전송 주파수보다 20 배 높게 설정됩니다. Hiperface 엔코더의 기본 9600 보(baud)의 변환을 위해 이 클록 주파수는 192kHz 이어야 합니다. 이는 **SerialClockMDiv** 및 **SerialClockNDiv** 양쪽에서 2 단 나눗셈이 필요합니다. 가능한 한 이상적인 주파수에 근접하도록 제 1 단에서 가능한 한 낮은 값으로 분주할 것을 권장합니다.

SerialClockMDiv: = \$82 // 130 으로 분주 ~768kHz
SerialClockNDiv: = 2 // 4 로 분주 ~192kHz
SerialTrigClockSel: = 1 // 서보 클록 사용
SerialTrigEdgeSel: = 1 // 하강 클록 엣지 사용
SerialTrigDelay: = 0 // 지연 없음
SerialProtocol: = \$04 // Hiperface 프로토콜 선택

예를 들어 9600 비트/초의 전송 속도에는 **SerialClockMDiv** = 130(\$82), **SerialClockNDiv** = 2 를 설정합니다. 또한 지연 없이 서보 클록의 하강 엣지에서 트리거하기 위해 **Gate3[i].SerialEncCtrl** 을 \$82230004 로 설정합니다. 이는 "원쇼트" 읽기이므로 트리거 클록 엣지의 선택은 크게 중요하지 않습니다.

8								2								2								3								0								0								0								4							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																
1	0	0	0	0	0	1	0	0	0	1	0	-	-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0																															
SerialClockMDiv								SerialClockNDiv -- -- TC TE								SerialTrigDelay								SerialProtocol																																							

Yaskawa 시그마 I 프로토콜



주의

Yaskawa 는 현재 시그마 I 애플솔루트 엔코더를 생산하지 않습니다. 그러나 새로운 시그마 II, III 및 V 엔코더를 사용하는 중이라도 신세대 Yaskawa 시그마 서보 드라이버는 컨트롤러로 복귀하기 위해 Yaskawa 시그마 I 프로토콜을 합성합니다.

Yaskawa 시그마 I 애플루트 엔코더용 **Gate3[i].SerialEncCtrl**의 전형적인 설정은 다음 목록과 같습니다. 명시적인 클록 신호가 없으므로 시리얼 클록 주파수는 입력 데이터를 "오버 샘플링"하는 비트 전송 주파수보다 20 배 높게 설정됩니다. Yaskawa 시그마 I 엔코더의 기본 9600 보(baud)의 변환을 위해 이 클록 주파수는 192kHz 이어야 합니다. 이는 **SerialClockMDiv** 및 **SerialClockNDiv** 양쪽의 2 단 나눗셈이 필요합니다. 가능한 한 이상적인 주파수에 근접하도록 제 1 단에서 가능한 한 낮은 값으로 분주할 것을 권장합니다.

SerialClockMDiv:	= \$82	// 130 으로 분주 ~768kHz
SerialClockNDiv:	= 2	// 4 로 분주 ~192kHz
SerialTrigClockSel:	= 1	// 서보 클럭 사용
SerialTrigEdgeSel:	= 1	// 하강 클럭 엣지 사용
SerialTrigDelay:	= 0	// 지연 없음
SerialProtocol:	= \$05	// Yaskawa 시그마 I 선택

예를 들어 9600bit/sec 의 전송률에는 **SerialClockMDiv** = 130(\$82), **SerialClockNDiv** = 2 를 설정합니다. 또한 지연 없이 서보 클럭의 하강 엣지에서 트리거하기 위해 **Gate3[7].SerialEncCtrl** 을 \$82230005 로 설정합니다. 이는 "원쇼트" 읽기이므로 트리거 클럭 엣지의 선택은 그다지 중요하지 않습니다.

8								2								2								3								0								0								0								5							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																
1	0	0	0	0	0	1	0	0	0	1	0	-	-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1																															
SerialClockMDiv								SerialClockNDiv								-- -- TC TE								SerialTrigDelay								SerialProtocol																															

Yaskawa 시그마 III/III/V 프로토콜

Yaskawa II/III/V 엔코더용 **Gate3[7].SerialEncCtrl** 의 전형적인 설정은 다음 목록과 같습니다.

```

SerialClockMDiv:    = 0           // 100MHz 시리얼 클럭 주파수 =
                        // 25×비트 전송 주파수
SerialClockNDiv:    = 0           // 나눗셈 없음
SerialTrigClockSel: = 0           // 가능하면 위상 클럭 사용
SerialTrigEdgeSel: = 0           // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay:   = 0           // 지연을 축소하기 위해 가능하면 0 부터 증가시킴
SerialProtocol:    = $06         // Yaskawa II/III/V 프로토콜 선택

```

예를 들어 표준 4.0MHz 의 비트 전송률에서는 100MHz 의 시리얼 클록 주파수를 사용하고 지연 없이 위상 클록의 상승 엣지에서 트리거하기 위해 **Gate3[7].SerialEncCtrl** 을 \$00000006 으로 설정합니다.

0				0				0				0				0				0				0				6							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
0 0 0 0 0 0 0 0								0 0 0 0 0 0 0 0								--		--		0 0		0 0		0 0		0 0		0 0		0 0		1 1			
SerialClockMDiv								SerialClockNDiv								TC		TE		SerialTrigDelay								SerialProtocol							

Tamagawa FA 코더 프로토콜

Tamagawa FA 코더 엔코더용 **Gate3[i].SerialEncCtrl** 의 전형적인 설정은 다음 목록과 같습니다. 명시적인 클럭 신호가 없으므로 DSPGATE3 중의 시리얼 엔코더 클럭은 입력 데이터 스트림에서 "오버 샘플링"하도록 비트 전송 주파수의 20 배여야 합니다.

SerialClockMDiv: $= (5/f_{bit}) - 1$ // 시리얼 클럭 주파수 = 20x 비트 전송 주파수
SerialClockNDiv: = 0 // 나눗셈 없음
SerialTrigClockSel: = 0 // 가능하면 위상 클럭 사용
SerialTrigEdgeSel: = 0 // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 축소하기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$07 // Tamagawa FA 코더 프로토콜 선택

예를 들어 2.5MHz 의 비트 전송률에서는 50MHz 의 시리얼 클럭 주파수를 사용하고 지연 없이 위상 클럭의 상승 엣지에서 트리거하기 위해 **Gate3[i].SerialEncCtrl** 을 \$01000007 로 설정합니다.

0				1				0				0				0				0				7							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	0	0	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
SerialClockMDiv								SerialClockNDiv -- -- TC TE								SerialTrigDelay								SerialProtocol							

Panasonic 프로토콜

Panasonic 시리얼 엔코더용 **Gate3[i].SerialEncCtrl** 의 전형적인 설정은 다음 목록과 같습니다. 명시적인 클럭 신호가 없으므로 DSPGATE3 중의 시리얼 엔코더 클럭은 시리얼 입력 데이터 스트림에서 "오버 샘플링"하도록 비트 전송 주파수의 20 배여야 합니다.

SerialClockMDiv: $= (5/f_{bit}) - 1$ // 시리얼 클럭 주파수 = 20x 비트 전송 주파수
SerialClockNDiv: = 0 // 나눗셈 없음
SerialTrigClockSel: = 0 // 가능하면 위상 클럭 사용
SerialTrigEdgeSel: = 0 // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 축소하기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$08 // Panasonic 프로토콜 선택

예를 들어 2.5MHz 의 비트 전송률에서는 50MHz 의 시리얼 클럭 주파수를 사용하고 지연 없이 위상 클럭의 상승 엣지에서 트리거하기 위해 **Gate3[i].SerialEncCtrl** 을 \$01000008 로 설정합니다.

0				1				0				0				0				0				8							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	0	0	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	01
SerialClockMDiv								SerialClockNDiv -- -- TC TE								SerialTrigDelay								SerialProtocol							

Mitutoyo 프로토콜

Mitutoyo 시리얼 엔코더용 **Gate3[i].SerialEncCtrl** 의 전형적인 설정은 다음 목록과 같습니다. 명시적인 클럭 신호가 없으므로 DSPGATE3 중의 시리얼 엔코더 클럭은 시리얼 입력 데이터 스트림에서 "오버 샘플링"하도록 비트 전송 주파수의 20 배여야 합니다.

SerialClockMDiv: $= (5/f_{\text{bit}}) - 1$ // 시리얼 클럭 주파수 = 20x 비트 전송 주파수
SerialClockNDiv: = 0 // 나눗셈 없음
SerialTrigClockSel: = 0 // 가능하면 위상 클럭 사용
SerialTrigEdgeSel: = 0 // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 축소하기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$09 // Mitutoyo 프로토콜 선택

예를 들어 2.5MHz 의 비트 전송률에서는 50MHz 의 시리얼 클럭 주파수를 사용하고 지연 없이 위상 클럭의 상승 엣지에서 트리거하기 위해 **Gate3[i].SerialEncCtrl** 을 \$01000009 로 설정합니다.

0								1								0								0								0								0								9							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	0	0	0	0	0	0	1	0	0	0	0	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1																							
SerialClockMDiv								SerialClockNDiv								--		--		TC		TE		SerialTrigDelay								SerialProtocol																							

Kawasaki 프로토콜

Kawasaki 시리얼 엔코더용 **Gate3[i].SerialEncCtrl** 의 전형적인 설정은 다음 목록과 같습니다. 명시적인 클럭 신호가 없으므로 DSPGATE3 중의 시리얼 엔코더 클럭은 시리얼 입력 데이터 스트림에서 "오버 샘플링"하도록 비트 전송 주파수의 20 배여야 합니다.

SerialClockMDiv: $= (5/f_{\text{bit}}) - 1$ // 시리얼 클럭 주파수 = 20x 비트 전송 주파수
SerialClockNDiv: = 0 // 나눗셈 없음
SerialTrigClockSel: = 0 // 가능하면 위상 클럭 사용
SerialTrigEdgeSel: = 0 // 가능하면 클럭 상승 엣지 사용
SerialTrigDelay: = 0 // 지연을 축소하기 위해 가능하면 0 부터 증가시킴
SerialProtocol: = \$0A // Kawasaki 프로토콜 선택

예를 들어 2.5MHz 의 비트 전송률에서는 50MHz 의 시리얼 클럭 주파수를 사용하고 지연 없이 위상 클럭의 상승 엣지에서 트리거하기 위해 **Gate3[i].SerialEncCtrl** 을 \$0100000A 로 설정합니다.

0								1								0								0								0								0								A							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0	0	0	0	0	0	0	1	0	0	0	0	-	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0																							
SerialClockMDiv								SerialClockNDiv								-- -- TC TE								SerialTrigDelay								SerialProtocol																							

Gate3[i].ServoClockDiv 게이트 3 의 서보 클럭 분주

설명:	IC 의 서보 클럭 주파수 제어
범위:	0 ..15
단위:	없음
기본값:	3

Gate3[i].ServoClockDiv 는 내부에서 생성된 서보 클럭 신호의 주파수를 지정합니다. 이는 IC 의 내부 위상 클럭 주파수(**Gate3[i].PhaseFreq** 로 설정)로부터 몇 번 분주되는지에 따라 제어됩니다. 서보 클럭 주파수를 구하는 식은 다음과 같습니다.

$$f_{ServoClock} = \frac{f_{PhaseClock}}{ServoClockDiv + 1}$$

Gate3[i].PhaseServoDir 의 비트 1(값 2)이 0 으로 설정되는 경우에만 이 내부에서 생성된 서보 클럭이 사용되며 IC 가 그 자체의 서보 클럭으로 사용하여 출력하도록 지정됩니다. **Gate3[i].PhaseServoDir** 의 비트 1 이 1 로 설정되는 경우에 IC 는 외부에서 생성된 서보 클럭 신호를 요구하고 그 자체의 클럭 대신 사용합니다. 일반적으로 Power PMAC 시스템의 모든 IC 는 정확하게 동기를 유지하기 위해 단일 서보 클럭 신호를 사용해야 합니다.

적절한 속도로 궤적을 실행하기 위해 서보 클럭의 사이클 시간을 궤도 생성 소프트웨어에 전달하려면 **Sys.ServoPeriod** 는 올바르게 설정되었어야 합니다. **Sys.ServoPeriod** 의 식은 다음과 같습니다.

$$Sys.ServoPeriod = \frac{1}{ServoFreq(kHz)}$$

서보 클럭 주파수를 결정하는 변수로 치환하면 **Sys.ServoPeriod** 의 계산식은 다음과 같습니다.

$$Sys.ServoPeriod = 1000 * \frac{(Gate3[i].ServoClockDiv + 1)}{Gate3[i].PhaseFreq}$$

Gate3[i].ServoClockDiv 는 폴 워드 요소 **Gate3[i].HardwareClockCtrl** 의 비트 0-3 을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[j].채널 특유의 설정 요소

이 장의 설정 요소는 IC 상에 선택된 서보 채널의 입력 및 출력의 이용을 구성하기 위해 사용됩니다. 하드웨어 채널 번호 1~4 를 각각 나타내는 채널 인덱스값 j 가 0~3 범위가 된다는 점에 주의하십시오.

Gate3[j].Chan[j].AdcOffset[k] 게이트 3 채널의 ADC 오프셋

설명: IC 채널의 엔코더용 ADC 바이어스 보정

범위: $-2^{31}..2^{31} - 1$

단위: 32 비트의 ADC LSB

기본값: 0

Gate3[j].Chan[j].AdcOffset[k] 요소는 IC 가 나중의 아크탄젠트 및 제곱합 계산을 실시하기 전에 채널용의 본래 "엔코더용" ADC 데이터의 32 비트 레지스터에 자동으로 더해지는 바이어스 보정값을 지정합니다. n 비트 ADC 로 이루어지는 실제 ADC 데이터는 32 비트 레지스터의 상위 n 비트에 배치됩니다. 가장 일반적으로는 16 비트 ADC 가 사용됩니다. 그러나 다른 분해능도 사용할 수 있습니다.

AdcOffset[0]의 값은 **AdcEnc[0]**의 입력 데이터 값에 자동으로 더해집니다. **AdcOffset[1]**의 값은 **AdcEnc[1]**의 입력 데이터 값에 자동으로 더해집니다. 이는 A 상 및 B 상 신호의 위상 클럭의 상승 엣지에서 스트로브 된 값이며 사인파 피드백 타입(엔코더 및 리졸버)의 주요 위치 피드백값을 획득할 수 있습니다. 해당 오프셋 항목에서는 자동으로 계산된 아크탄젠트 및 제곱합값에 아날로그 회로의 바이어스 보정을 포함하는 것이 가능합니다.

만약 n 비트 ADC 의 LSB 단위로 필요한 오프셋값이 설정된다면 해당 오프셋 레지스터 중 하나에 쓰여야 하는 값은 32 비트값을 얻기 위해 이 값에 2^{32-n} 을 곱한 값입니다. 예를 들어 16 비트 ADC 의 -7LSB 의 오프셋을 원하는 경우에 $-7*65,536$ 의 값(즉, -458,752)이 **AdcOffset[k]**에 쓰여야 합니다.

읽기 가능한 레지스터 **AdcEnc[0]** 및 **AdcEnc[1]**값이 이 프로세스에 영향을 받지 않는다는 점에 주의하십시오. 해당 값은 보정되지 않은 값입니다.

"앰프용" ADC 값을 위해 IC 에 유사 하드웨어 바이어스 보정 항목은 없습니다. 모터의 디지털 전류 피드백을 위해 이러한 회로를 사용할 때는 바이어스 보정은 소프트웨어 항목 **Motor[x].IaBias** 와 **Motor[x].IbBias** 로 실시됩니다.

Gate3[j].Chan[j].AtanEna 게이트 3 채널의 아크탄젠트 유효

설명: IC 채널 아크탄젠트 확장을 유효로 설정함

범위: 0 ..1

단위: 논리값

기본값: 0

Gate3[i].Chan[j].AtanEna 는 채널의 **PhaseCapture** 및 **ServoCapture** 레지스터 중의 "소수점 카운트" 값을 계산하기 위해 "엔코더용 ADC" 레지스터에서 자동 아크탄젠트 계산을 사용할지 여부를 제어합니다. 이러한 계산은 IC 가 리졸버 및 사인파 엔코더의 "사인" 및 "코사인" 입력으로부터의 각도값을 직접 계산할 수 있게 합니다. 모든 위상 클록 사이클에서 2 가지 레지스터 **AdcEnc[k]**가 읽힙니다. 사용자 설정 **AdcOffset[k]** 바이어스 값을 여기에 더한 다음 아크탄젠트(2 개의 인수, +/-180°)가 제공합처럼 계산됩니다. 읽기 전용 채널 요소 **Atan** 및 **SumOfSquares** 에 아크탄젠트 및 제공합값이 각각 쓰입니다. 이러한 계산은 **AtanEna** 의 설정에 관계없이 실시됩니다.

Gate3[i].Chan[j].AtanEna 를 기본값 0 으로 설정하는 경우에 소수점 카운트값을 결정하기 위해 이러한 아크탄젠트 계산은 사용되지 않습니다. 이 모드에서 채널의 **PhaseCapture** 및 **ServoCapture** 레지스터의 소수점 카운트값을 어떻게 계산할지는 채널의 **TimerMode** 설정으로 결정됩니다. 이러한 레지스터에서의 정수값은 각각 위상과 서보 클록 신호의 하강 엣지에서 래치된 엔코더 카운터의 값입니다.

Gate3[i].Chan[j].AtanEna 를 1 로 설정하는 경우에 **PhaseCapture** 및 **ServoCapture** 레지스터의 소수점 카운트값을 계산하기 위해 아크탄젠트 계산을 사용합니다. 이 모드에서 이러한 2 가지 레지스터의 하위 14 비트는 아크탄젠트 계산에 의해 엔코더 라인당 16,384 상태 또는 AB 상 카운트당 4096 상태가 됩니다. 이러한 레지스터에서의 정수 카운트값은 위상 클록 신호의 상승 엣지 후에 래치된 엔코더 카운터의 값이며 위상 및 서보 클록의 신호의 하강 엣지 전의 **Gate3[i].EncLatchDelay** 로 각각 설정된 지연을 동반합니다. 이 모드와 같은 때에 엔코더용 ADC 를 읽습니다.

Gate3[i].Chan[j].AtanEna 는 풀 워드 요소 **Gate3[i].Chan[j].InCtrl** 의 비트 18 을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].CaptCtrl 게이트 3 채널의 포착 제어

설명: IC 채널의 위치 포착 제어

범위: 0 ..15

단위: 없음

기본값: 1

Gate3[i].Chan[j].CaptCtrl 은 이 PMAC3 스타일 인터페이스 IC 를 위해 어느 입력 신호 또는 어느 입력 신호의 조합 그리고 어느 극성이 이 채널의 엔코더 카운터의 하드웨어 위치 포착의 트리거가 될지를 결정합니다. 플래그 입력(홈, 리미트 또는 사용자)을 사용하는 경우에 **Gate3[i].Chan[j].CaptFlagChan** 은 어느 채널의 플래그가 사용될지를 결정합니다. 또한 **Gate3[i].Chan[j].CaptFlagSel** 은 선택된 채널의 어느 플래그가 사용될지를 결정합니다.

매우 정확한 하드웨어 위치 포착 기능이 사용되는 경우에 이 채널을 사용하는 모터에 대해 원점 서치 동작 또는 다른 트리거 포함 동작을 성공시키기 위해 이 변수의 적절한 설정이 필요합니다. **Motor[x].CaptureMode** 가 0, 1 또는 3 으로 하드웨어 트리거가 선택된 경우(하드웨어 위치 포착 유무에 관계없음) 이 변수를 적절히 설정해야 합니다.

Gate3[i].Chan[j].CaptCtrl 에서는 다음 설정을 사용할 수 있습니다.

- **Gate3[i].Chan[j].CaptCtrl = 0:** 즉시 또는 홀 센서로 포착
- **Gate3[i].Chan[j].CaptCtrl = 1:** 인덱스(CHCn) high 로 포착
- **Gate3[i].Chan[j].CaptCtrl = 2:** 플래그 high 로 포착
- **Gate3[i].Chan[j].CaptCtrl = 3:** (인덱스 low AND 플래그 low)로 포착
- **Gate3[i].Chan[j].CaptCtrl = 4:** 즉시 또는 홀 센서로 포착
- **Gate3[i].Chan[j].CaptCtrl = 5:** 인덱스(CHCn) low 로 포착
- **Gate3[i].Chan[j].CaptCtrl = 6:** 플래그 high 로 포착
- **Gate3[i].Chan[j].CaptCtrl = 7:** (인덱스 high AND 플래그 low)로 포착
- **Gate3[i].Chan[j].CaptCtrl = 8:** 즉시 또는 홀 센서로 포착
- **Gate3[i].Chan[j].CaptCtrl = 9:** 인덱스(CHCn) high 로 포착
- **Gate3[i].Chan[j].CaptCtrl = 10:** 플래그 low 로 포착
- **Gate3[i].Chan[j].CaptCtrl = 11:** (인덱스 low AND 플래그 high)로 포착
- **Gate3[i].Chan[j].CaptCtrl = 12:** 즉시 또는 홀 센서로 포착
- **Gate3[i].Chan[j].CaptCtrl = 13:** 인덱스(CHCn) low 로 포착
- **Gate3[i].Chan[j].CaptCtrl = 14:** 플래그 low 로 포착
- **Gate3[i].Chan[j].CaptCtrl = 15:** (인덱스 high AND 플래그 high)로 포착

이 변수의 비트 0 및 1 을 0(요소 값 0, 4, 8 또는 12)으로 설정하고 동일한 채널의 **Gate3[i].Chan[j].GatedIndexSel** 이 0 이면 트리거가 작동 가능해지고 포착 트리거가 즉시 발생합니다. 동일한 채널의 **Gate3[i].Chan[j].GatedIndexSel** 이 1 이면 트리거가 U, V 또는 W 의 "홀 센서" 입력 플래그의 변화에 따라 발생합니다.

위치 포착 레지스터 **Gate3[i].Chan[j].HomeCapt** 가 읽히면 트리거는 작동 가능해집니다. 이는 트리거가 작동 가능해지고 아직 트리거가 발생하지 않았다는 것을 나타내며 상태 요소 **Gate3[i].Chan[j].PosCapt** 가 0 이 됩니다. 그러나 Power PMAC 의 트리거 포함 동작 기능은 동작 시작으로 이 읽기를 자동으로 실시합니다. 따라서 트리거는 항상 작동 가능합니다. 그 후에 IC 하드웨어는 지정된 입력 라인이 지정된 상태로 변화했다고 판단하면 즉시 트리거가 발생합니다. - 이는 레벨 트리거가 아닌 엣지 트리거입니다. (**Gate1[i]** 데이터 구조체를 사용하는 구형 DSPGATE1 IC 와는 달리 트리거를 발생시키는 작동 가능 후에 상태 변화가 필요합니다.)

7 및 11 의 설정처럼 3 및 15 값의 설정은 **Gate1[i].Chan[j].CaptCtrl** 이 반대가 된다는 점에 주의하십시오.

Gate3[i].Chan[j].CaptCtrl 은 풀 워드 요소 **Gate3[i].Chan[j].InCtrl** 의 비트 6-9 를 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].CaptFlagChan 게이트 3 채널의 포착 플래그 채널 선택

설명:	IC 채널의 위치 포착 플래그 채널 선택
범위:	0 ..3
단위:	없음
기본값:	(이 채널의 인덱스값)

Gate3[i].Chan[j].CaptFlagChan 은 이 IC 의 4 개 채널 중 무엇이 하드웨어 위치 포착 기능의 플래그를 공급할지 결정합니다. **Gate3[i].Chan[j].CaptFlagSel** 은 선택된 채널의 어느 플래그가 사용될지 결정합니다. 또한 **Gate3[i].Chan[j].CaptCtrl** 은 플래그를 사용할지 여부와 플래그의 어느 극성이 트리거를 발생시킬지를 결정합니다.

Gate3[i].Chan[j].CaptFlagChan 은 선택된 채널의 인덱스를 지정하므로 0~3 의 값으로 설정할 수 있습니다. 신호 설명 중에서 사용되는 채널의 하드웨어 번호 할당이 1~4 까지의 범위라는 점에 주의하십시오. 따라서 이 변수는 채널의 하드웨어 번호-1 로 설정해야 합니다. 이 변수의 기본값은 채널 자신의 인덱스값입니다. 따라서 기본값에서 채널은 그 자신의 플래그로 위치 포착을 사용합니다.

Gate3[i].Chan[j].CaptFlagChan 은 풀 워드 요소 **Gate3[i].Chan[j].InCtrl** 의 비트 12-13 을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].CaptFlagSel 게이트 3 채널의 위치 포착 플래그 선택

설명:	IC 채널의 위치 포착 플래그 선택
범위:	0 ..3
단위:	없음
기본값:	0

Gate3[i].Chan[j].CaptFlagSel 은 PMAC3 스타일 인터페이스 IC 상에서 **Gate3[i].Chan[j].CaptFlagChan** 으로 선택된 채널의 4 개 "플래그" 입력 중 무엇을 엔코더 카운터의 하드웨어 위치 포착(1 개를 사용하는 경우)에 사용할지 결정합니다. **Gate3[i].Chan[j].CaptCtrl** 은 플래그를 사용할지 여부와 플래그의 어느 극성이 트리거를 발생시킬지를 결정합니다. **Gate3[i].Chan[j].CaptFlagSel** 의 가능한 값 및 플래그 각각의 선택은 다음과 같습니다.

- **Gate3[i].Chan[j].CaptFlagSel = 0:** HOMEn(홈 플래그 n)
- **Gate3[i].Chan[j].CaptFlagSel = 1:** PLIMn(양의 한계 리미트 플래그 n)
- **Gate3[i].Chan[j].CaptFlagSel = 2:** MLIMn(음의 한계 리미트 플래그 n)
- **Gate3[i].Chan[j].CaptFlagSel = 3:** USERn(사용자 플래그 n)

보통 채널의 홈 플래그를 사용하는 원점 서치 동작을 위해 **Gate3[i].Chan[j].CaptFlagSel** 을 0으로 설정합니다. 일반적으로 하드웨어 위치 포착 기능이 프로빙이나 위치 등록과 같은 다른 용도에서 필요한 경우에 사용자 플래그를 선택하기 위해 나중에 3으로 설정합니다. PLIMn 이 MLIMn 의 한계 리미트 플래그로 포착하는 경우는 **Motor[x].pLimits** 를 0으로 일시적으로 설정하여 일반적인 운전 정지 기능을 무효가 되도록 할 수 있습니다.

Gate3[i].Chan[j].CaptFlagSel 은 풀 워드 요소 **Gate3[i].Chan[j].InCtrl** 의 비트 10-11 을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].EncCtrl 게이트 3 채널의 엔코더 제어

설명: IC 채널의 엔코더 디코드 제어

범위: 0 ..15

단위: 없음

기본값: 7

Gate3[i].Chan[j].EncCtrl 은 채널의 엔코더 입력 신호가 어떻게 카운트에 디코드될지를 결정합니다. 따라서 "카운트"의 부호와 크기를 정의합니다. 다음의 설정이 입력 신호를 디코드하기 위해서 사용됩니다.

- **Gate3[i].Chan[j].EncCtrl = 0:** 외부 펄스 및 방향 CW
- **Gate3[i].Chan[j].EncCtrl = 1:** x1 AB 상 CW 디코드
- **Gate3[i].Chan[j].EncCtrl = 2:** x2 AB 상 CW 디코드
- **Gate3[i].Chan[j].EncCtrl = 3:** x4 AB 상 CW 디코드
- **Gate3[i].Chan[j].EncCtrl = 4:** 외부 펄스 및 방향 CCW
- **Gate3[i].Chan[j].EncCtrl = 5:** x1 AB 상 CCW 디코드
- **Gate3[i].Chan[j].EncCtrl = 6:** x2 AB 상 CCW 디코드
- **Gate3[i].Chan[j].EncCtrl = 7:** x4 AB 상 CCW 디코드
- **Gate3[i].Chan[j].EncCtrl = 8:** 내부 펄스 및 방향 CW
- **Gate3[i].Chan[j].EncCtrl = 9:** 펄스 업/펄스 다운 CW
- **Gate3[i].Chan[j].EncCtrl = 10:** (추후 사용하기 위해 예약)
- **Gate3[i].Chan[j].EncCtrl = 11:** x6 홀 형식 CW 디코드
- **Gate3[i].Chan[j].EncCtrl = 12:** 내부 펄스 및 방향 CCW
- **Gate3[i].Chan[j].EncCtrl = 13:** 펄스 업/펄스 다운 CCW
- **Gate3[i].Chan[j].EncCtrl = 14:** (추후 사용하기 위해 예약)
- **Gate3[i].Chan[j].EncCtrl = 15:** x6 홀 형식 CCW 디코드

AB 상 디코드 모드 중 무엇이든 IC 는 각각 약 50%의 듀티 사이클로 서로 1 사이클의 약 4분의 1 위상 편차가 있는 CHAn 및 CHBn 의 2 가지 입력 파형이 필요합니다. "1 체배"(x1) 디코드는 1 사이클당 1 카운트합니다. x2 는 1 사이클당 2 카운트합니다. 또한 x4 는 1 사이클당 4 카운트를 부여합니다. 많은 사용자는 최고 분해능을 얻기 위해 x4 디코드를 선택합니다.

어느 디코드 모드에서든 "시계 방향"(CW)과 "시계 반대 방향"(CCW) 옵션은 단순히 어느 방향으로 카운트 업할지를 제어합니다. 잘못된 방향 검출인 경우에는 다른 옵션(예를 들어 7 에서 3 으로 또는 반대로)으로 변경하십시오.



경고

정상 동작 중인 모터의 피드백 엔코더의 디코드 방향을 변경하는 것은 다른 변경이 없는 상태에서는 불안정한 정귀환이 되어 폭주 상태를 일으킵니다. 안정된 부귀환의 극성을 회복하려면 마찬가지로 출력 극성을 변경해야 합니다.

외부 펄스 및 방향의 디코드 모드에서는 IC 는 CHAn 의 펄스 열 및 CHBn 의 방향(부호) 신호가 필요합니다. 신호가 단일 방향인 경우에 CHBn 라인을 high 상태로 풀업시키거나 또는 high 나 low 상태로 결선할 수 있습니다.

내부 펄스와 방향 디코드 모드에서는 IC 는 디코더에 대한 입력으로 채널 자체의 펄스 주파수 변조(PFM) 펄스 열을 사용합니다. 이는 오픈 루프의 스테핑 모터 시스템을 구동할 때 Power PMAC 이 가상 클로즈드 루프를 생성하는 것이 가능해집니다. *점퍼 또는 케이블은 필요하지 않습니다. 연결은 완전히 IC 내에서 이루어집니다.*

펄스 업/펄스 다운 디코드 모드에서는 CHAn 에 한 방향으로 카운트를 생성하는 펄스가 필요하고 CHBn 에 다른 방향으로 카운트를 생성하는 펄스가 필요합니다. 이 펄스는 CW/CCW 선택에 의해 결정되는 방향으로 카운트를 생성합니다.

"x6"홀 형식의 디코드 모드에서는 IC 는 CHAn, CHBn 및 CHCn 에 각각 약 50%의 듀티 사이클로 서로 1 사이클의 약 1/3 위상 편차가 있는(120°e) 3 개의 홀 센서 형식 입력이 필요합니다. 디코드 회로는 각 신호의 각 엣지에서 1 카운트를 생성하고("x6 디코드"의 경우) 신호 1 사이클당 6 카운트를 부여합니다.

Gate3[i].Chan[j].EncCtrl 은 풀 워드 요소 **Gate3[i].Chan[j].InCtrl** 의 비트 0-3 을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].Equ1Ena 게이트 3 채널의 비교 엔코더 선택

설명: IC 채널의 비교 엔코더 선택

범위: 0 ..1

단위: 논리값

기본값: 0

Gate3[i].Chan[j].Equ1Ena 는 채널의 비교 회로가 해당 채널 자체의 엔코더 카운터나 또는 제 1 채널(chan[0])의 엔코더 카운트 중 무엇을 참조할지 결정합니다. **Gate3[i].Chan[j].Equ1Ena** 를 기본값 0 으로 설정하는 경우에 채널의 비교 회로는 채널 자체의 엔코더 카운터를 참조합니다. **Gate3[i].Chan[j].Equ1Ena** 를 1 로 설정하는 경우에 채널의 비교 회로는 제 1 채널의 엔코더 카운터를 참조합니다.

물론 **Gate3[i].Chan[0].Equ1Ena** 의 설정은 어느 쪽이든 동작에 차이는 없습니다.

단일 엔코더 카운터에 여러 개의 비교 회로를 참조시키는 것은 "원도잉"과 같은 특별한 위치 범위의 고주수 비교 출력 기능을 가능하게 합니다.

Gate3[i].Chan[j].Equ1Ena 는 풀 워드 요소 **Gate3[i].Chan[j].OutCtrl** 의 비트 5 를 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].EquOutMask 게이트 3 채널의 비교 상태 출력 마스크

설명: IC 채널의 비교 상태 출력 마스크

범위: \$0..\$F

단위: 비트 필드

기본값: **Gate3[i].Chan[0].EquOutMask=\$1**
Gate3[i].Chan[1].EquOutMask=\$2
Gate3[i].Chan[2].EquOutMask=\$4
Gate3[i].Chan[3].EquOutMask=\$8

Gate3[i].Chan[j].EquOutMask 는 이 채널의 위치 비교(EQU) 출력 생성에 어느 채널의 내부 위치 비교 상태를 사용할지 결정합니다. IC 의 채널에는 각각 엔코더 카운터와 **CompA/CompB** 의 레지스터 값의 비교 이력에 따른 내부 비교 상태가 있습니다.

채널의 비교 출력 레벨은 IC 의 4 가지 비교 상태 중 어느 것의 논리합입니다. **Gate3[i].Chan[j].EquOutMask** 의 비트(0~3)를 각각 1 로 설정하면 비트에 대응하는 채널의 내부 비교 상태를 논리합하고 이 채널의 비교 출력으로 출력합니다.

논리상의 비교 상태를 결합시키는 것은 다차원 비교 기능을 가능하게 하거나 또는 여러 개의 내부 비교 상태가 동일한 엔코더를 사용하는 경우 부여된 하나의 영역의 고주파 자동 증가 비교 출력의 정확한 "원도잉"을 가능하게 합니다.

기본설정에서는 채널의 비교 출력 레벨의 생성에 채널 자체의 내부 비교 상태만을 사용하도록 지정됩니다. 채널의 비교 출력에 4 가지 채널 내부 비교 상태 모두를 사용하려면 이 요소를 \$F(15)로 설정합니다.

논리합 후에 출력 레벨은 뒤에 설명하는 **Gate3[i].Chan[j].EquOutPol** 로 반전될 수도 있습니다.

Gate3[i].Chan[j].EquOutMask 는 풀 워드 요소 **Gate3[i].Chan[j].OutCtrl** 의 비트 0-3 을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].EquOutPol 게이트 3 채널의 비교 출력 극성

설명: IC 채널의 비교 출력 극성

범위: 0 ..1

단위: 비트 필드

기본값: 0

Gate3[i].Chan[j].EquOutPol 은 채널의 비교 출력 레벨이 논리상 반전될지 여부를 결정합니다. 채널의 비교 출력 레벨은 IC 의 개별 채널 내부 비교 상태 4 개까지의 논리합으로 최초에 생성됩니다. **Gate3[i].Chan[j].EquOutPol** 이 0 인 경우에 이 중간 상태는 채널의 EQU 신호에 출력되기 전에 반전되지 않습니다. **Gate3[i].Chan[j].EquOutPol** 이 1 인 경우에 이 중간 상태는 채널의 EQU 신호에 출력되기 전에 반전됩니다.

반전 제어 기능은 **Gate3[i].Chan[j].EquOutPol** 에 의해 제공되며 **Gate3[i].Chan[j].EquOutMask** 의 논리합 기능 및 **Gate3[i].Chan[j].EquWrite** 의 초기 상태 설정 기능과 조합되어 사용자에게 비교 상태에 어떤 목적의 논리 결합이라도 사실상 생성하는 기능을 부여합니다.

채널의 **Gate3[i].Chan[j].EquOutMask** 비트를 0 으로 설정하는 경우에 채널의 비교 출력 레벨의 생성에서 내부 비교 상태가 일체 사용되지 않는 것을 의미하므로 **Gate3[i].Chan[j].EquOutPol** 은 출력 레벨 제어에 직접 사용할 수 있으며 소프트웨어 제어의 범용 디지털 출력으로 EQU 출력을 사용할 수 있습니다.

Gate3[i].Chan[j].EquOutPol 은 풀 워드 요소 **Gate3[i].Chan[j].OutCtrl** 의 비트 5 를 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].FlagFilt2Ena 게이트 3 채널의 제 2 필터 유효

설명: IC 채널의 플래그 제 2 필터 유효

범위: 0 ..1

단위: 논리값

기본값: 0

Gate3[i].Chan[j].FlagFilt2Ena 은 프로세서로 읽을 수 있는 채널의 상태 레지스터 **Gate3[i].Chan[j].Status** 에 값을 래치하기 전에 채널의 입력 플래그를 제 2 디지털 지연 필터에 적용할지 여부를 제어합니다. 제 2 필터는 노이즈 내성을 향상시키므로 오작동을 방지하는 데 도움이 됩니다.

모든 플래그(엔코더 신호와 동일하게)는 자동으로 엔코더 샘플 클록 주파수로 신호를 샘플링하는 제 1 디지털 지연 필터를 통과합니다. 이는 보통 수 MHz 로 설정됩니다. 필터는 최신 샘플 상의 최적의 3 개 중 2 개가 일치하는 출력을 제공합니다. 따라서 1 개의 샘플에만 존재하는 스파이크 노이즈를 통과시키지 않습니다. 그러나 고주파수 엔코더 신호가 동일한 샘플링 속도의 필터를 통과하므로 샘플링 속도를 높게 설정해야 합니다. 또한 플래그 입력 중의 스파이크 노이즈는 여러 개의 샘플링 기간 동안 계속되어 제 1 필터를 통과할 가능성도 있습니다.

Gate3[i].Chan[j].FlagFilt2Ena 을 1 로 설정하는 경우에 플래그 입력은 계속해서 제 2 디지털 지연 필터로 전달됩니다. 제 1 필터와 구조는 동일하지만 IC 의 필터 클록에 의해 설정된 주파수로 샘플링됩니다. 이 클록 주파수는 **Gate3[i].FiltClockDiv** 에 의해 설정되어 엔코더 샘플링 클록 주파수에서 얼마나 분주될지 결정합니다. **Gate3[i].Chan[j].FlagFilt2Ena** 을 0 으로 설정하는 경우에 이 제 2 필터는 바이패스됩니다.

이 제 2 필터가 유효로 설정되어도 플래그의 포착 기능이 제 2 필터 전에 일어나므로 플래그의 포착 기능 타이밍은 변경되지 않는다는 점에 주의하십시오.

Gate3[i].Chan[j].FlagFilt2Ena 은 폴 워드 요소 **Gate3[i].Chan[j].InCtrl** 의 비트 17 을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].GatedIndexSel 게이트 3 채널의 게이트 인덱스 선택

설명: IC 채널의 게이트 제어된 인덱스 포착 선택

범위: 0 ..1

단위: 없음

기본값: 0

Gate3[i].Chan[j].GatedIndexSel 은 PMAC3 스타일 인터페이스 IC 의 원시 엔코더의 인덱스 채널 입력 또는 AB 상 상태에서 게이트된 입력 형식 중 어느 것을 채널의 엔코더 위치 포착에 사용할지 여부를 제어합니다. 이에는 다음의 설정이 가능합니다.

- **Gate3[i].Chan[j].GatedIndexSel = 0:** 게이트되지 않은 인덱스를 엔코더 위치 포착에 사용
- **Gate3[i].Chan[j].GatedIndexSel = 1:** 위치 포착에 AC 상 채널에서 게이트된 인덱스를 사용

Gate3[i].Chan[j].GatedIndexSel 을 0 으로 설정하는 경우에 엔코더의 인덱스 채널 입력(CHCn)은 위치 포착 회로에 직접 전달됩니다.

Gate3[i].Chan[j].GatedIndexSel 을 1 로 설정하는 경우에 엔코더의 인덱스 채널 입력(CHCn)은 위치 포착 회로에 전달되기 전에 엔코더 n 의 AB 상 신호와 논리 결합됩니다("게이트화됨"). 그 의도는 정확히 1 사이클의 AB 상 폭의 "게이트 인덱스" 신호를 얻는 것입니다. 더 정확하고 더 재현성이 높은 포착을 제공하고 1 회전당 적절한 카운트 수를 확인하는 포착 기능의 이용을 매우 간단하게 합니다.

제대로 게이트된 인덱스 포착이 확실히 작동할 수 있도록 인덱스 펄스는 확실히 1 회만이지만 "high-high" 또는 "low-low"의 엔코더 AB 상 상태를 유지해야 합니다. **Gate3[i].Chan[j].IndexGateState** 는 이 2 개의 가능성 중 어느 쪽을 사용할지 선택할 수 있습니다.

주의: **Gate3[i].Chan[j].GatedIndexSel** 이 1 로 설정되었더라도 **Gate3[i].Chan[j].CaptCtrl** 의 비트 0, 1 이 0 으로 설정되는 경우는 위치 포착에 인덱스를 사용하지 않습니다. 엔코더 위치는 채널의 입력 플래그 U, V 또는 W 중 어느 하나의 최초의 엣지에서 포착됩니다. 이 경우에 채널 상태 워드의 비트 16, 17 및 18 은 어느 홀 상태 엣지가 포착을 일으킬지 나타냅니다.

Gate3[i].Chan[j].GatedIndexSel 은 풀 워드 요소 **Gate3[i].Chan[j].InCtrl** 의 비트 14 를 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].InCtrl 게이트 3 채널의 입력 제어

설명: IC 채널의 입력 신호 제어 워드

범위: 0 ..\$FFFFFFFF

단위: 비트 필드

기본값: \$00400047

Gate3[i].Chan[j].InCtrl 은 IC 채널의 입력 신호 인터페이스 설정 요소로 구성되는 풀 워드 요소입니다. 다음 부분 워드 요소로 구성됩니다.

구성 요소	비트	기능
(예약)	31 - 23	(추후 사용하기 위해 예약)
PackInData	22 - 21	1 워드에 ADC 데이터를 압축 유효
SerialEncEna	20	채널 상의 시리얼 엔코더 인터페이스를 유효로 합니다.
PosClear	19	엔코더 카운터 및 타이머를 0 으로 리셋합니다.
AtanEna	18	엔코더용 ADC 로부터의 아크탄젠트 계산 유효
FlagFilt2Ena	17	플래그용 제 2 디지털 지연 필터 유효
IndexDemuxEna	16	인덱스로부터의 홀 데이터의 다중화 해제 유효
IndexGateState	15	인덱스 포착의 게이트에서의 AC 상 상태를 선택
GatedIndexSel	14	포착을 위한 AB 상 상태에 따른 인덱스의 게이트 유효
CaptFlagChan	13 - 12	채널의 포착에 대한 플래그의 채널을 선택
CaptFlagSel	11 - 10	채널의 포착에 대한 플래그를 선택
CaptCtrl	09 - 06	채널의 포착을 위한 트리거 상태를 선택
TimerMode	05 - 04	채널의 타이머 모드를 선택
EncCtrl	03 - 00	채널의 엔코더 디코드 방법 선택

굵은 글씨로 나타내는 요소는 설정 요소로 저장되고 각각 설명됩니다.

사용자는 부분 워드 요소의 설정과 설정 확인을 위해 보통 스크립트 환경에서 여기에 액세스합니다. Power PMAC 은 효율화를 위해서 풀 워드 요소로 저장 및 복원합니다.

C 프로그램에서는 사용자는 풀 워드 요소만 액세스해야 합니다.

풀 워드 요소의 값이 스크립트 환경 중에서 참조된 경우에 부분 워드 요소 값을 10 진수로 통지되도록 해도 Power PMAC 은 16 진수 형식으로 값을 반환합니다.

Gate3[i].InCtrl 은 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

각각의 요소의 기능에 대한 자세한 설명은 각 부분 워드 요소 항목을 참조하십시오.

Gate3[i].Chan[j].IndexDemuxEna 게이트 3 채널 다중화 해제 유효

설명: IC 채널의 인덱스 다중화 해제 유효

범위: 0 ..1

단위: 논리값

기본값: 0

Gate3[i].Chan[j].IndexDemuxEna 은 Yaskawa Σ 시그마 I의 인크리멘탈 엔코더처럼 AB 상 상태에 따른 3번째 채널로부터의 인덱스 펄스 및 3가지 홀 센서 전류(commutation) 상태를 "다중화 해제"할지 여부를 제어합니다. 0으로 설정하면 이 다중화 해제 기능은 실행되지 않고 엔코더의 "C"상 채널의 신호가 인덱스로 사용됩니다. 1로 설정하면 IC는 3번째 채널 신호로부터 4개의 독립된 값을 분해 출력하고 이 중 하나가 AB 상 신호의 가능한 4가지 상태 중 하나가 됩니다. 전원 투입 시의 위상 위치와 그 후의 위상 위치 모니터링을 가능하게 하기 위해 이 다중화 해제된 홀 소자 전류(commutation) 상태를 사용할 수 있습니다.

주의: 전원 투입 직후에 Yaskawa 엔코더는 자동으로 완전한 AB 상 사이클 형태로 AB 상 출력을 전후에 순환 출력합니다. 컨트롤러가 동작을 시작하기 전에 모든 홀 센서의 전류(commutation) 상태가 이용 가능하다는 것을 보증하기 위해서입니다. 그러나 Power PMAC과 동시에 엔코더에 전원이 공급되면 IC가 이러한 신호를 수신할 준비가 되기 전에 이 출력이 시작됩니다. 만약 준비가 되었을 때 서보 IC가 이러한 상태 모두를 수신할 수 없다면 채널의 Status 워드 **Gate3[i].Chan[j].Status** "다중화 해제 무효"의 비트 19를 1로 설정합니다. 이 기능을 사용하기 위해서 소프트웨어로 컨트롤되는 릴레이를 통해서 엔코더에 전원을 공급하는 것을 권장합니다. 이에 따라 전원 투입 시의 위상 읽기 전에 모든 상태의 유효한 읽기가 가능하다는 것이 보증되기 때문입니다.

Gate3[i].Chan[j].IndexDemuxEna 은 풀 워드 요소 **Gate3[i].Chan[j].InCtrl**의 비트 16을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey**에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey**에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].IndexGateState 게이트 3 채널의 인덱스 게이트 상태

설명:	IC 채널의 게이트된 인덱스 포착 상태 제어
범위:	0..1
단위:	없음
기본값:	0

Gate3[i].Chan[j].IndexGateState는 서보 IC의 엔코더 입력에 공급된 원시 인덱스 채널 신호가 "high-high"의 AB 상 상태(=0) 또는 "low-low"의 AB 상 상태(=1) 중 어느 상태를 위치 포착 신호에 전달시킬지 지정합니다. **Gate3[i].Chan[j].GatedIndexSel**이 1로 설정되면 이 "게이트 인덱스" 기능이 유효로 설정됩니다.

Gate3[i].Chan[j].IndexGateState는 풀 워드 요소 **Gate3[i].Chan[j].InCtrl**의 비트 15를 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey**에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey**에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].OutCtrl 게이트 3 채널의 출력 제어

설명: IC 채널의 출력 신호 제어 워드

범위: 0 ..\$FFFFFFFF

단위: 비트 필드

기본값: \$0F800001

Gate3[i].Chan[j].OutCtrl 은 IC 채널의 출력 신호 인터페이스 설정 요소로 구성되는 풀 워드 요소입니다. 다음 부분 워드 요소로 구성됩니다.

구성 요소	비트	기능
PwmDeadTime	31 - 24	PWM 출력용 데드 타임 전환
PackOutData	23	1 워드에 대한 PWM/DAC 데이터의 압축 유효
PwmFreqMult	22 - 20	PWM 주파수를 위한 승수
PfmFormat	19	펄스-방향 또는 AB 상 출력 선택
PfmDirPol	18	PFM 방향 신호의 출력 극성 선택
OutputPol	17 - 16	명령 신호의 출력 극성 선택
OutputMode	15 - 12	위상 명령 신호의 출력 형식 선택
OutFlagD	11	채널의 D 플래그 출력의 high/low 상태 설정
OutFlagC	10	채널의 C 플래그 출력의 high/low 상태 설정
OutFlagB	09	채널의 B 플래그 출력의 high/low 상태 설정
AmpEna	08	채널의 앰프 Enable 출력 플래그(A 플래그)의 high/low 상태 설정
EquWrite	07 - 06	비교 출력 상태의 강제 출력
Equ1Ena	05	비교를 위한 엔코더 채널 선택
EquOutPol	04	비교 출력 극성 선택
EquOutMask	03 - 00	어느 채널의 비교 상태를 논리합하여 출력할지 선택

굵은 글씨로 나타내는 요소는 설정 요소로 저장되고 이 장에서 각각 기재됩니다. 굵은 글씨가 아닌 요소는 저장되지 않으며 저장하지 않는 설정 요소에 대한 장에 기재합니다.

사용자는 부분 워드 요소의 설정과 설정 확인을 위해 보통 스크립트 환경에서 여기에 액세스합니다. Power PMAC 은 효율화를 위해서 풀 워드 요소로 저장 및 복원합니다.

C 프로그램에서는 사용자는 풀 워드 요소만 액세스해야 합니다.

풀 워드 요소의 값이 스크립트 환경 중에서 참조된 경우에 부분 워드 요소 값을 10 진수로 통지되도록 해도 Power PMAC 은 16 진수 형식으로 값을 반환합니다.

저장 요소인 **Gate3[i].Chan[j].OutCtrl** 은 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

각각의 요소의 기능에 대한 자세한 설명은 각 부분 워드 요소 항목을 참조하십시오.

Gate3[i].Chan[j].OutputMode 게이트 3 채널의 위상 출력 모드

설명: IC 채널의 위상 출력 모드 선택

범위: \$0..\$F

단위: 비트 필드

기본값: \$0

Gate3[i].Chan[j].OutputMode 는 IC 채널의 각 위상(A~D)의 명령 신호 라인에서 어느 출력 형식이 사용될지 결정합니다. 이는 4 비트값이며 각 비트에서 대응하는 위상의 출력 형식을 제어합니다.

비트 #	비트값	위상	출력 형식(0)	출력 형식(1)
0	1	A	PWM	DAC
1	2	B	PWM	DAC
2	4	C	PWM	DAC
3	8	D	PWM	PFM

3 상 모터에 대한 다이렉트 PWM 출력에 필요한 위상 A, B 및 C에 대한 PWM 출력을 위해서는 **Gate3[i].Chan[j].OutputMode** 를 0 또는 8로 설정해야 합니다.

전형적인 스텝핑 모터용으로서 2 개의 독립된 위상의 다이렉트 PWM 제어에 필요한 4 가지 위상 모두에 PWM 출력하려면 **Gate3[i].Chan[j].OutputMode** 를 0으로 설정해야 합니다.

아날로그 속도 또는 토크 모드 앰프에 필요한 A 상에 대한 싱글 DAC 출력 또는 Power PMAC 가 브러시리스 모터의 위상 전류(commutation)를 실시할 때 필요한 아날로그 "사인파"의 A 상, B 상에 대한 듀얼 DAC 출력을 위해서는 **Gate3[i].Chan[j].OutputMode** 를 3, 7, 11 또는 15로 설정해야 합니다.

기존의 스텝핑 모터 드라이브의 펄스 및 방향 제어에 필요한 D 상에 대한 PFM 출력 또는 MLDT 여기 펄스 출력을 위해서는 **Gate3[i].Chan[j].OutputMode** 를 8 이상으로 설정해야 합니다.

Gate3[i].Chan[j].OutputMode 는 풀 워드 요소 **Gate3[i].Chan[j].OutCtrl** 의 비트 12-15 를 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].OutputPol 게이트 3 채널의 위상 출력 반전

설명: IC 채널의 위상 출력 반전 제어

범위: 0..3

단위: 비트 필드

기본값: 0

Gate3[i].Chan[j].OutputPol 은 IC 채널의 명령 신호 라인을 반전시킬지 여부를 결정합니다. 비트 0(값 1)은 A, B 상 및 C 상의 반전을 제어하고(PWM 또는 DAC 모드 중 어느 쪽이든) 또한 비트 1 은 D 상의 반전을 제어하는(PWM 또는 PFM 모드 중 어느 쪽이든) 2 비트의 값입니다. 비트값이 0 이면 비반전 출력(high 일 때 참)을 지정합니다. 비트값이 1 이면 반전 출력(low 일 때 참)을 지정합니다. 변수로 설정 가능한 값은 다음과 같습니다.

- **Gate3[i].Chan[j].OutputPol = 0:** 비반전 A, B & C, 비반전 D
- **Gate3[i].Chan[j].OutputPol = 1:** 반전 A, B & C, 비반전 D
- **Gate3[i].Chan[j].OutputPol = 2:** 비반전 A, B & C, 반전 D
- **Gate3[i].Chan[j].OutputPol = 3:** 반전 A, B & C, 반전 D

기본값의 비반전 출력은 정방향입니다. 이는 PWM 신호에 대해 트랜지스터 ON 신호가 high 인 것을 의미합니다. Delta Tau PWM 입력 앰프 및 대부분의 다른 PWM 입력 앰프에는 이 비반전 출력 형식이 필요합니다. 이러한 3 상 모터 드라이브에는 **Gate3[i].Chan[j].OutputPol** 을 0 또는 8 로 설정해야 합니다.



주의

PWM 신호의 high/low 극성이 특정한 앰프에서 잘못된 경우에 **Gate3[i].Chan[j].PwmDeadTime** 에 의해 설정되는 Top 과 Bottom 의 ON 상태 간의 데드 타임으로 의도된 것은 오버랩합니다. 앰프 입력 회로가 이를 올바르게 배제하지 않는 경우에 버스 전원과 GND 간에 사실상의 순간 단락을 일으킵니다. 이는 파워 트랜지스터를 매우 빠르게 파괴하게 됩니다.

출력 D 의 PFM 신호에서는 비반전은 펄스 ON 신호가 high 인 것을 의미합니다(방향 극성은 **Gate3[i].Chan[j].PfmDirPol** 로 제어됩니다). 방향 전환 중에 방향 비트는 펄스의 선단 엣지에 동기하여 변화합니다. 비반전 형식에서는 상승 엣지를 의미합니다. 드라이버가 펄스의 상승 엣지 전에 방향 라인의 설정 시간을 필요로 하는 경우에 펄스 출력을 반전 설정하면 상승 엣지가 후방 엣지인 것처럼 되고 또한 펄스 폭(**Gate3[i].PwmDeadTime** 로 설정됨)은 설정 시간이 됩니다.

출력 A, B 및 C 의 DAC 신호에 대해 비반전은 DAC 에 대한 1 의 값이 high 인 것을 의미합니다. 다른 기존에 알려진 모든 DAC 와 동일하게 Delta Tau 액세서리 보드에서 사용되는 DAC 역시 항상 비반전 입력을 필요로 하며 따라서 이 채널에서 DAC 를 사용하는 경우에 **Gate3[i].Chan[j].OutputPol** 의 적절한 비트를 항상 0 으로 설정해야 합니다.



경고

DAC 의 디지털 데이터의 high/low 극성을 변경하면 DAC 의 아날로그 출력의 전압 극성을 반대로 하는 효과가 있습니다. 이는 출력과 피드백 간의 극성 일치를 변경합니다. 만약 피드백 루프가 부귀환으로 안정되었다면 해당 변경은 위험한 폭주 상태가 되어 불안정한 정귀환을 야기합니다.

Gate3[i].Chan[j].OutputPol 은 풀 워드 요소 **Gate3[i].Chan[j].OutCtrl** 의 비트 16-17 을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].PackInData 게이트 3 채널의 입력 데이터 압축 유효

설명: IC 채널의 ADC 입력 "압축" 유효

범위: 0 ..3

단위: 비트 필드

기본값: 2

Gate3[i].Chan[j].PackInData 는 IC 채널의 2 가지 상의 ADC 데이터를 하나의 32 비트 레지스터에 "압축"할지 여부를 제어합니다. 이는 2 비트의 변수입니다. 비트 0(값 1)에서는 채널의 "엔코더용" **AdcEnc[k]** 값을 압축할지 여부를 제어합니다. 비트 1(값 2)에서는 "앰프용" **AdcAmp[k]** 값을 압축할지 여부를 제어합니다. 비트값이 0 이면 압축을 무효로 합니다. 비트값이 1 이면 압축을 유효로 합니다.

데이터가 압축 형식인 경우에 하위 16 비트의 **AdcEnc[0]** 또는 **AdcAmp[0]**에 상위 16 비트의 **AdcEnc[1]** 또는 **AdcAmp[1]**을 각각 복사하고 하위 16 비트의 **AdcEnc[2]** 또는 **AdcAmp[2]**에 상위 16 비트의 **AdcEnc[3]** 또는 **AdcAmp[3]**을 각각 복사합니다. 이 모드에서 1 개의 32 비트 레지스터의 읽기는 프로세서 양쪽의 위상 정보에 대한 액세스를 가능하게 합니다. 각각의 ADC 에 16 비트 이상의 데이터가 있는 경우에 이 모드에서는 하위 비트가 상실됩니다.

Power PMAC 이 모드의 디지털 전류 루프를 실행하는 경우에 이 채널을 전류 피드백에 사용하면 데이터의 압축은 알고리즘의 효율을 비약적으로 향상시킵니다. 따라서 이 경우 **PackInData** 를 2 또는 3 으로 설정해야 합니다. **Motor[x].PhaseCtrl** 의 비트 0(값 1)에는 압축 데이터가 필요하며 또한 1 회의 읽기를 실시하도록 모터 알고리즘에 명령하기 위해 1 을 설정하십시오. 이 경우에 채널의 **PackOutData** 요소에도 하나의 레지스터에 2 개의 명령 출력을 압축하여 레지스터에 쓰도록 1 을 설정하십시오. 전류의 피드백 데이터가 압축되지 않는 경우에 **Motor[x].PhaseCtrl** 비트 2(값 4)에는 그 대신 압축 데이터를 필요로 하지 않으며 또한 2 개의 읽기를 실시하도록 모터 알고리즘에 명령하기 위해 1 을 설정하십시오.

현재 압축 모드에서 **AdcEnc[k]**를 사용하는 Power PMAC 의 자동 기능은 없습니다. 아날로그 사인파 엔코더용 엔코더 변환 테이블의 타입 4 소프트웨어 보간에서는 비압축 데이터가 필요합니다. 이 변환 방법을 사용하려면 **PackInData** 의 비트 0(값 1)은 0 이어야 합니다.

Gate3[i].Chan[j].PackInData 는 풀 워드 요소 **Gate3[i].Chan[j].InCtrl** 의 비트 21-22 를 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].PackOutData 게이트 3 채널의 출력 데이터 압축 유효

설명:	IC 채널의 PWM/DAC "압축" 유효
범위:	0 ..1
단위:	논리값
기본값:	1

Gate3[i].Chan[j].PackOutData 는 IC 채널의 4 상 출력(PWM 또는 DAC/PFM) 데이터가 2 개의 32 비트 레지스터에 "압축"될지 여부를 제어합니다. 0 으로 설정하는 경우에 데이터를 압축하지 않고 각 레지스터에 4 상 출력 명령값을 써야 합니다. 1 로 설정하는 경우에 데이터는 압축되고 2 개의 레지스터 각각의 상반부와 하반부에 4 상 출력 명령을 씁니다.

데이터가 압축되는 경우에 C 상 명령(압축되지 않았으면 **Pwm[2]** 또는 **Dac[2]**)의 상위 16 비트는 A 상 명령의 32 비트 레지스터(**Pwm[0]** 또는 **Dac[0]**)의 하위 16 비트와 D 상 명령(압축되지 않았으면 **Pwm[3]** 또는 **Pfm**)의 상위 16 비트는 B 상 명령의 32 비트 레지스터(**Pwm[1]** 또는 **Dac[1]**)의 하위 16 비트에 쓰일 것입니다. IC 는 자동으로 A, B 상의 메모리 맵의 32 비트 레지스터의 하위 16 비트를 C, D 상의 작업용 레지스터의 상위 16 비트에 각각 값을 전송합니다.

Power PMAC 이 모터의 전류(commutation)와 디지털 전류 루프 계산을 실시하는 경우에 이 채널을 여러 개의 위상 출력에 사용하면 데이터의 압축은 알고리즘의 효율을 비약적으로 향상시킵니다. 따라서 **PackOutData** 를 1 로 설정해야 합니다. **Motor[x].PhaseCtrl** 비트 0(값 1)에는 모터 알고리즘이 압축 데이터 형식으로 사용하기 위해 1로 설정해야 합니다. 이 경우에 채널의 **PackInData** 요소도 동일하게 레지스터에 2 개의 전류 피드백 입력을 압축하기 위해 1로 설정해야 합니다. 입출력 데이터가 압축되지 않는 경우에 **Motor[x].PhaseCtrl** 의 비트 2(값 4)는 그 대신 모터 알고리즘이 압축 데이터를 필요로 하지 않고 또한 각 입출력 레지스터용으로 개별 읽기 동작을 실시하도록 1로 설정하십시오.

Power PMAC 이 모터의 전류(commutation) 계산을 실시하지만 디지털 전류 루프를 실행하지 않는 경우에 이 채널을 여러 개의 위상 출력에 사용하려면 **PackOutData** 는 1로 설정하고 **Motor[x].PhaseCtrl** 의 비트 0 을 1로 설정하면 알고리즘의 효율이 더욱 향상됩니다. 그러나 이 "사인파 출력" 모드에서는 간혹 출력 데이터에 16 비트 이상의 분해능이 필요해집니다. (ACC-24E3 아날로그 인터페이스 보드는 표준 16 비트 DAC 및 18 비트 DAC 에서도 사용 가능합니다.) 압축 모드에서는 최대 16 비트까지의 분해능을 사용할 수 있습니다. 따라서 16 비트보다 큰 분해능이 필요하면서 해당 분해능을 온전히 활용하기 바란다면 **PackOutData** 를 0으로 설정해야 하며 **Motor[x].PhaseCtrl** 의 비트 2 는 그 대신 1로 설정해야 합니다.

Gate3[i].Chan[j].PackOutData 는 풀 워드 요소 **Gate3[i].Chan[j].OutCtrl** 의 비트 23 을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].Pfm 게이트 3 채널의 PFM 명령

설명:	IC 채널의 펄스 주파수 변조 레지스터의 명령값
범위:	-2,147,483,392..2,147,483,391($2^{31} + 256.. 2^{31} - 257$)
단위:	부호가 있는 32 비트의 PFM 단위
기본값:	0

Gate3[i].Chan[j].Pfm 은 지정된 IC 와 채널의 펄스 주파수 변조 레지스터에 대한 명령값을 지정합니다. 각각의 채널에는 명령값에 비례한 주파수의 펄스 및 방향 신호 페어를 생성하는 하나의 PFM 레지스터가 있습니다. PFM 출력은 일반적으로 기존의 스테핑 모터 드라이브 명령이나 MLDT 센서를 여기하거나 레이저 및 다른 출력을 변조하기 위해 사용됩니다. 시뮬레이트 된 피드백 루프, 정확한 트리거 타이밍(타이머 보조의 소프트웨어 위치 포착용) 그리고 다른 주파수/타이밍 용도를 위해 내부에서 채널의 엔코더 카운터 및 타이머에 피드백할 수 있습니다.

각 PFMCLK 클럭 사이클에서 24 비트의 어큐뮬레이터에 **Gate3[i].Chan[j].Pfm** 의 상위 24 비트값이 더해집니다(하위 8 비트는 사용되지 않습니다). 어큐뮬레이터가 "롤 오버"될 때마다 출력 펄스를 생성합니다. 양의 방향으로 롤 오버하는 경우에 방향 출력 신호는 "+(플러스)"로 설정됩니다. 음의 방향으로 롤 오버하는 경우에 방향 출력 신호는 "-(마이너스)"로 설정됩니다. 이처럼 생성된 펄스 주파수는 요소의 값에 비례합니다.

PFMCLK 신호의 주파수는 저장 설정 요소 **Gate3[i].PfmClockDiv** 로 설정됩니다. 기본 주파수는 3.125MHz 입니다. 부여되는 PFMCLK 주파수 및 32 비트의 명령값에서 결과적으로 발생하는 출력 주파수는 다음과 같이 산출할 수 있습니다.

$$f_{out} = \frac{Gate3[i].Chan[j].Pfm}{4,294,483,648} * f_{PFMCLK}$$

Gate3[i].Chan[j].OutputMode 의 비트 3(값 8)을 1 로 설정하면 PFM 모드 출력에 D 상이 설정되는 경우 명령값 **Gate3[i].Chan[j].Pwm[3]**에서 생성되는 PWM Top 과 Bottom 신호 핀 대신 펄스 및 방향 신호가 D 상 출력 핀에 출력됩니다. 내부 펄스 생성 및 채널의 엔코더 카운터 또는 타이머에 해당 펄스 신호를 보내는 능력이 출력 모드에 관계없이 부여된다는 점에 주의하십시오.

저장 설정 요소 **Gate3[i].Chan[j].PfmFormat** 을 기본값 0 으로 설정하는 경우에 펄스와 방향의 페어로 신호를 출력합니다. 각 펄스의 폭은 PFMCLK 사이클 단위로 저장 설정 요소 **Gate3[i].Chan[j].PfmWidth** 에 의해 결정됩니다. 앞 펄스 종료 전에 다음 펄스를 생성하면 사실상 출력을 포화시켜서 출력은 단순히 "ON"인 채로 유지됩니다. 양/음의 명령값 방향 출력의 극성은 저장 설정 요소 **Gate3[i].Chan[j].PfmDirPol** 로 결정됩니다.

Gate3[i].Chan[j].PfmFormat 을 1 로 설정하는 경우에 AB 상의 A 와 B 페어로 신호를 출력합니다. 펄스는 각각의 채널에 하나의 엣지를 생성합니다. 양/음의 명령값에서의 AB 상의 방향 극성은 저장 설정 요소 **Gate3[i].Chan[j].PfmDirPol** 로 결정됩니다.

"MLDT 펄스 엔코더의 타이밍" 모드를 위해 저장 설정 요소 **Gate3[i].Chan[j].TimerMode** 를 1 로 설정하는 경우에 **Gate3[i].Chan[j].Pfm** 의 값에 관계없이 하나의 펄스는 각 서보 클럭 사이클의 하강 엣지에서 자동으로 출력됩니다.

저장 설정 요소 **Gate3[i].Chan[j].EncCtrl** 을 8 또는 12 로 설정하는 경우에 내부 펄스 및 방향 신호는 1 펄스로 1 카운트를 생성하는 펄스로 채널의 엔코더 카운터를 구동합니다(외부 신호 대신). **Gate3[i].Chan[j].TimerMode** 를 3 으로 설정하는 경우에 내부 펄스 및 방향 신호는 엔코더 카운터를 외부 신호에 유효인 채로 두고(타임 베이스 확장이 없지만) 카운터로서 채널의 타이머 회로를 구동합니다.

Gate3[i].Chan[j].Pfm 은 펄스 폭 변조 명령값 **Gate3[i].Chan[j].Pwm[3]**과 레지스터를 공유합니다. 이 레지스터의 주소를 설정한 **Motor[x].pDac** 으로 모터 명령 출력하는 경우에 레지스터는 모든 서보 또는 위상 사이클에서 자동으로 갱신됩니다. 이 경우에는 범용으로 이용할 수 없습니다. 하지만 그렇지 않은 경우에 이 레지스터에 대해 값을 쓰면 언제든지 설정 가능한 레지스터로 범용 목적으로 이용 가능합니다.

저장 설정 요소 **Gate3[i].Chan[j].PackOutData** 를 1 로 설정하는 경우에 채널의 4 가지 상의 16 비트값은 이러한 32 비트 요소 2 개 중으로 "압축"됩니다. 따라서 프로세서는 불과 2 개의 쓰기 조작으로 4 개의 상 전부에 명령할 수 있습니다. 이 경우에 채널의 **Pwm[1]** 요소의 하위 16 비트에 PFM 명령값의 상위 16 비트를 씁니다. 24 비트의 PFM 명령값의 하위 8 비트는 이 모드에서는 항상 0 입니다. 이 모드에서는 채널의 **Pfm** 요소의 값은 사용되지 않습니다.

다른 ASIC 명령 출력 요소와 달리 비휘발성 플래시 메모리에 이 요소의 값을 저장할 수 있으며 따라서 전원 투입/리셋으로 자동 복원합니다. 이는 모든 전원 투입/리셋 후에 실행되며 명시적인 명령이 필요 없이 어플리케이션에서 고정 주파수가 생성될 수 있게 합니다. 이는 타이머 보조 소프트웨어 위치 포착 기능의 사용에 특히 효과가 있습니다.

스크립트 환경에서 이 요소에 대한 유효 범위 밖의 값을 쓰면 요소의 값은 해당 부호마다 최대값이 됩니다.

Gate3[i].Chan[j].PfmDirPol 게이트 3 채널의 PFM 방향 극성

설명: IC 채널의 PFM 방향 극성 제어

범위: 0 ..1

단위: 논리값

기본값: 0

Gate3[i].Chan[j].PfmDirPol 은 IC 채널의 PFM 방향 출력 신호 형식의 극성을 결정합니다. **Gate3[i].Chan[j].OutputMode** 가 D 상의 PFM 출력을 지정하는 8 이상의 값으로 설정된 경우에만 유효합니다.

Gate3[i].Chan[j].PfmDirPol 을 기본값 0 으로 설정하는 경우에 출력은 비반전입니다. 펄스 및 방향 모드에서는(**Gate3[i].Chan[j].PfmFormat** = 0 으로 설정됨) 양의 방향은 low 출력입니다.

Gate3[i].Chan[j].PfmDirPol 을 1 로 설정하는 경우에 출력은 반전입니다. 펄스 및 방향 모드에서는(**Gate3[i].Chan[j].PfmFormat** = 0 으로 설정됨) 양의 방향은 high 출력입니다.

AB 상 모드에서는(**Gate3[i].Chan[j].PfmFormat** = 1 로 설정됨) **Gate3[i].Chan[j].PfmDirPol** 은 제 2 의 B 상("방향" 라인의 출력)을 반전시키고 AB 상 신호의 방향 극성을 유효하게 제어합니다.

Gate3[i].Chan[j].PfmDirPol 은 풀 워드 요소 **Gate3[i].Chan[j].OutCtrl** 의 비트 18 을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].PfmFormat 게이트 3 채널의 PFM 출력 형식

설명: IC 채널의 PFM 출력 형식 제어

범위: 0 ..1

단위: 논리값

기본값: 0

Gate3[i].Chan[j].PfmFormat 은 IC 채널의 D 상의 PFM 회로의 출력 신호 형식을 결정합니다. **Gate3[i].Chan[j].OutputMode** 가 D 상의 PFM 출력을 지정하는 8 이상의 값으로 설정된 경우에만 유효합니다.

Gate3[i].Chan[j].PfmFormat 을 기본값 0 으로 설정하는 경우에 출력은 펄스 및 방향 모드입니다. 이 모드는 기존의 스테핑 모터 드라이브 제어 및 MLDT 센서의 여기에 도움이 됩니다.

Gate3[i].Chan[j].PfmFormat 을 1 로 설정하는 경우에 출력은 AB 상 모드입니다. 이 모드는 합성된 인크리멘탈 엔코더 신호를 생성하는 데 도움이 됩니다.

(Power PMAC 펌웨어의 몇 개 이전 버전에서는 이 요소는 **Gate3[i].Chan[j].PfmOutFormat** 으로 기재되었습니다.)

Gate3[i].Chan[j].PfmFormat 은 풀 워드 요소 **Gate3[i].Chan[j].OutCtrl** 의 비트 19 를 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].PfmWidth 게이트 3 채널의 PFM 출력 펄스 폭

설명: 채널의 펄스 주파수 변조의 출력 펄스 폭

범위: 0 ..4095

단위: PFM_CLK 사이클

기본값: 15

Gate3[i].Chan[j].PfmWidth 는 채널의 펄스 주파수 변조 출력의 각 펄스 출력의 지속 시간을 지정합니다. PFM_CLK 신호의 사이클 단위로 표시합니다. 해당 주파수는 IC 의 저장 설정 요소 **Gate3[i].PfmClockDiv** 로 결정됩니다.

원하는 PFM 펄스 폭과 PFM_CLK 주파수의 함수로서 **Gate3[i].Chan[j].PfmWidth** 를 구하는 식은 다음과 같습니다.

$$PfmWidth = PFM\_CLK\_Freq(MHz) * PFM\_Pulse\_Width(\mu sec) - 1$$

예

IC 의 PFM_CLK 주파수는 25MHz 입니다. 600 나노초(0.6μsec)의 펄스 폭으로 설정하면 다음과 같이 값을 산출할 수 있습니다.

$$Gate3[i].Chan[j].PfmWidth = 25 \text{ cyc}/\mu\text{sec} * 0.6\mu\text{sec} - 1 = 14$$

Gate3[i].Chan[j].PwmDeadTime 게이트 3 채널의 PWM 데드 타임

설명:	IC 채널의 데드 타임 제어
범위:	0 ..255
단위:	16 * PWM_CLK 사이클(53.33 nsec)
기본값:	15(800 nsec)

Gate3[i].Chan[j].PwmDeadTime 은 IC 채널의 각 상의 Top 신호와 Bottom 신호 페어 중에서 하나의 신호를 OFF 로 전환하는 것과 다른 신호를 ON 으로 전환하는 것 사이의 "데드 타임" 기간의 길이를 결정합니다. "다이렉트 PWM" 제어 중에서 PWM 신호를 파워 트랜지스터의 실제 ON/OFF 제어에 사용하는 경우에 충분히 OFF 로 전환하는 데 어느 정도 시간이 걸리고 페어 중의 다른 트랜지스터를 즉시 ON 으로 전환하는 것은 순간적인 단락을 일으키므로 이 지연은 필수적입니다.

16 개의 PWM_CLK 사이클 단위로 **Gate3[i].Chan[j].PwmDeadTime** 를 표시합니다. 300MHz 로 PWM_CLK 신호의 주파수가 고정됩니다. 따라서 1 사이클은 3.33 나노초이고 또한 16 사이클은 53.3 나노초입니다. 원하는 데드 타임의 함수로서의 이 파라미터 식은 다음과 같습니다.

$$Gate3[i].Chan[j].PwmDeadTime = \frac{DesiredDeadTime(\mu sec)}{0.0533\mu sec}$$



주의

대부분의 다이렉트 PWM 드라이브는 안전 기능으로 최소의 데드 타임 기간이 필요합니다. 단, 이 데드 타임 기간은 이 IC 의 PWM_CLK 신호에 비동기이며 더 낮은 주파수의 클럭으로 실시됩니다. 이는 드라이브의 최소 데드 타임 설정을 신뢰하여 유효한 명령 분해능을 줄이고 나아가서는 출력에 대한 맥놀이 현상을 초래한다는 것을 의미합니다(이 요소가 더 작은 데드 타임으로 지정되는 경우에 일어납니다).

Gate3[i].Chan[j].PwmDeadTime 은 풀 워드 요소 **Gate3[i].Chan[j].OutCtrl** 의 비트 24-31 을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].PwmFreqMult 게이트 3 채널의 PWM 주파수 승수

설명: IC 채널의 PWM 주파수의 승수

범위: 0 ..7

단위: 승수

기본값: 0

Gate3[i].Chan[j].PwmFreqMult 는 IC 채널의 PWM 출력 주파수가 IC 의 내부 위상 클록 주파수와 어떤 관계가 될지 결정합니다. 이 관계의 식은 다음과 같습니다.

$$f_{PWM} = \frac{PwmFreqMult + 1}{2} f_{IntPhase}$$

따라서 **Gate3[i].Chan[j].PwmFreqMult** 의 0~7 의 가능한 값으로는 내부 위상 클록 주파수의 0.5~4 배의 PWM 주파수 값을 설정할 수 있습니다.

Gate3[i].Chan[j].PwmFreqMult 는 풀 워드 요소 **Gate3[i].Chan[j].OutCtrl** 의 비트 20-22 를 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].SerialEncCmd 게이트 3 채널의 시리얼 엔코더 명령

설명: IC 채널의 시리얼 엔코더의 명령 제어 워드

범위: 0 ..\$FFFFFFFF

단위: 비트 필드

기본값: \$0

Gate3[i].Chan[j].SerialEncCmd 는 IC 채널의 시리얼 엔코더 인터페이스의 명령 정보로 구성되는 풀 워드 요소입니다. 특정한 프로토콜, 트리거 타이밍 및 클록 주파수가 멀티채널 요소 **Gate3[i].SerialEncCtrl** 로 결정된다는 점에 주의하십시오.

Gate3[i].Chan[j].SerialEncCmd 는 다음 구성 요소로 구성됩니다. 개별 요소로서 각각 별도로 구성 요소에 액세스할 수 없다는 점에 주의하십시오.

구성 요소	비트	기능
<i>SerialEncCmdWord</i>	31 - 16	시리얼 엔코더 출력 명령
<i>SerialEncParity</i>	15 - 14	시리얼 엔코더 패리티 타입
<i>SerialEncTrigMode</i>	13	시리얼 트리거 모드: 연속 또는 단발
<i>SerialEncTrigEna</i>	12	시리얼 트리거 유효
<i>SerialEncGtoB</i>	11	시리얼 SSI 데이터의 그레이-바이너리 변환 제어
<i>SerialEncDataReady</i>	10	시리얼 엔코더 수신 데이터 준비 완료
<i>SerialEncStatusBits</i>	09 - 06	상태 비트의 시리얼 엔코더 SPI 번호
<i>SerialEncNumBits</i>	05 - 00	시리얼 엔코더 비트 길이 제어

프로토콜 고유의 방법으로 시리얼 엔코더로 송신하는 명령값을 정의하기 위해 16 비트의 구성 요소 *SerialEncCmdWord* 를 사용합니다.

2 비트 구성 요소 *SerialEncParity* 는 수신 데이터 패킷(패리티 확인을 지원하는 프로토콜용)을 위해 필요한 패리티 타입을 정의합니다. 0 의 값은 패리티를 지정하지 않습니다. 1 의 값은 홀수 패리티를 지정합니다. 2 의 값은 짝수 패리티를 지정합니다. (추후 사용하기 위해 3 의 값이 예약됩니다.)

1 비트 구성 요소 *SerialEncTrigMode* 는 엔코더를 반복 샘플링할지 아니면 1 회만 샘플링할지 여부를 지정합니다. 0 의 값에서는 연속적인 샘플링(멀티채널 요소 **Gate3[i].SerialEncCtrl** 로 설정되는 위상 사이클이나 서보 사이클마다)을 지정합니다. 1 의 값은 원쇼트의 샘플링을 지정합니다.

1 비트 구성 요소 *SerialEncTrigEna* 은 엔코더를 샘플링할지 여부를 지정합니다. 0 의 값에서는 샘플링 되지 않습니다. 1 의 값에서는 엔코더 샘플링이 가능해집니다. *SerialEncTrigMode* 가 0 으로 설정되었으면 *SerialEncTrigEna* 이 1 의 값 그대로인 한 반복 엔코더는 샘플링됩니다(멀티채널 요소 **Gate3[i].SerialEncCtrl** 로 설정되는 위상 사이클이나 서보 사이클마다). 그러나 *SerialEncTrigMode* 가 1 로 설정되었으면 엔코더를 한 번만 샘플링합니다. 또한 IC 는 샘플링 후 자동으로 *SerialEncTrigEna* 을 0 으로 반환합니다.

1 비트 구성 요소 *SerialEncGtoB* 에서는 SSI 프로토콜로 반환되어 온 데이터가 그레이 형식에서 수치 바이너리 형식으로 변환될지 여부를 지정합니다. 0 의 값에서는 변환을 수행하지 않습니다. 1 의 값에서는 수신 데이터의 그레이-바이너리 변환을 수행합니다. 인터페이스 회로가 디지털 AB 상 엔코더에 사용되는 경우(**Gate3[i].Chan[j].SerialEncEna** = 0) 이 비트는 "4 체배" AB 상 디코드의 검출 방향을 제어합니다.

1 비트의 구성 요소 *SerialEncDataReady* 는 시리얼 데이터 수신 상태를 나타내는 읽기 전용 Status 비트입니다. 통지 0 은 데이터 전송 중의 유효한 새로운 데이터가 아직 준비 완료되지 않았음을 나타냅니다. 데이터를 모두 수신하여 처리했으면 1 을 통지합니다. 이는 읽기가 완료될 때까지 여러 개의 서보 사이클에 걸쳐 있는 더 느린 인터페이스에서 특히 중요합니다. 이러한 경우에는 데이터 준비 여부를 판정하기 위해 해당 비트는 폴링되어야 합니다. 인터페이스 회로가 디지털 AB 상 엔코더에 사용되는 경우(**Gate3[i].Chan[j].SerialEncEna** = 0) 이 비트를 1 로 설정하면 상태 요소 **Gate3[i].Chan[j].SerialEncDataA** 로 읽을 수 있는 카운트값을 0 클리어합니다.

4 비트 구성 요소 *SerialEncStatusBits* 는 인터페이스가 SPI 프로토콜의 엔코더에서 필요로 하는 Status 비트 수를 지정합니다. 설정의 유효 범위는 0~12 입니다.

6 비트 구성 요소 *SerialEncNumBits* 는 SPI, SSI 또는 EnDat 프로토콜의 엔코더 인터페이스가 필요로 하는 데이터 비트 수를 지정합니다. 설정의 유효한 범위는 12-63입니다.

자세한 내용은 시리얼 엔코더를 처리하기 위해 사용되는 특별한 인터페이스 보드(예를 들어 ACC-24E3)의 하드웨어 참조 설명서에 기재됩니다. 다음은 지원되는 각 프로토콜에 공통되는 설정입니다.

SPI 프로토콜

SPI 엔코더용 **Gate3[j].Chan[j].SerialEncCmd** 의 전형적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord:	= 0	// 명령 워드는 SPI 프로토콜에서는 // 지원되지 않습니다.
SerialEncParity:	= 0	// 패리티 확인은 SPI 프로토콜에서는 // 지원되지 않습니다.
SerialEncTrigMode:	= 0	// 연속 트리거
SerialEncTrigEna:	= 1	// 트리거 유효
SerialEncGtoB:	= 0	// 그레이 코드에서 바이너리 변환의 지원 없음
SerialEncDataReady:	= 0	// 읽기 전용 Status 비트
SerialEncStatusBits:	= ??	// 회신되는 Status 비트의 엔코더 특유의 비트 수
SerialEncNumBits:	= ??	// 회신되는 위치 비트의 엔코더 특유의 비트 수

예를 들어 28 비트의 위치 비트 및 4 비트의 Status 비트를 갖는 SPI 엔코더는 **Gate3[i].Chan[j].SerialEncCmd** 를 \$0000111C 로 설정합니다. (데이터 준비 완료 Status 비트가 설정된 경우 \$0000151C 로 통지됩니다.)

0				0				0				0				1		1		1		C									
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	1	-	-	0	1	0	0	0	1	1	1	0	0
SerialEncCmdWord																Parity		TM	TE	GB	Rdy	Status				NumBits					

SSI 프로토콜

SSI 엔코더용 **Gate3[7].Chan[7].SerialEncCmd** 의 전형적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord:	= 0	// 명령 워드는 SSI 프로토콜에서는 // 지원되지 않습니다.
SerialEncParity:	= ??	// 엔코더 특유의 패리티 확인
SerialEncTrigMode:	= 0	// 연속 트리거
SerialEncTrigEna:	= 1	// 트리거 유효
SerialEncGtoB:	= ??	// 엔코더 특유의 데이터 형식
SerialEncDataReady:	= 0	// 읽기 전용 Status 비트
SerialEncStatusBits:	= 0	// Status 비트는 SSI 프로토콜에서는 // 지원되지 않습니다.
SerialEncNumBits:	= ??	// 회신되는 위치 비트의 엔코더 특유의 비트 수

예를 들어 짝수 패리티, 그레이 코드 형식에서의 25 비트 위치 비트를 갖춘 SSI 엔코더에 대해서는 **Gate3[i].Chan[j].SerialEncCmd** 에 \$00005819 를 설정합니다. (데이터 준비 완료 Status 비트가 설정된 경우 \$0005C19 로 통지됩니다.)

0				0				0				0				5				8				1				9																																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																												
-								-								0				1				0				1				1				-				-				-				-				0				1				1				0				0				1			
SerialEncCmdWord																Parity TM TE GB Rdy Status												NumBits																																															

EnDat2.1/2.2 프로토콜

DSPGATE3 IC EnDat 인터페이스는 엔코더에 대한 2 개의 6 비트 명령 코드를 지원합니다. 위치를 보고하기 위한 000111(\$07) 및 엔코더를 리셋하기 위한 101010(\$2A)입니다. 이러한 6 비트는 **Gate3[i].Chan[j].SerialEncCmd** 의 16 비트의 **SerialEncCmdWord** 명령 필드의 하위 비트로 할당됩니다. 이는 EnDat2.1 또는 EnDat2.2 엔코더 중 하나에서 사용할 수 있는 명령 코드입니다.



주의

EnDat 규격에서 EnDat2.2 엔코더는 EnDat2.1 명령 코드도 동일하게 처리할 수 있어야 합니다. 그러나 EnDat2.2 규격에 대응하도록 판매된 모든 엔코더에서 이를 실행할 수는 없습니다.

SerialEncCmdWord 의 최상위 비트(풀 워드 요소의 비트 31)는 **StartDelayComp** 제어 비트입니다. 이 비트를 1로 설정하면 엔코더와 컨트롤러 간의 전송 지연을 측정하는 지연 확인 및 보정 사이클을 시작합니다. 지연 시간을 3회 측정하고 보정 중에 이러한 지연 값의 평균을 사용합니다. 이러한 계산이 끝나면 자동으로 **StartDelayComp** 비트가 리셋됩니다. 지연 확인 동작은 전원 투입 후에 언제나 실행되어야 합니다. 지연 보정은 매우 긴 케이블에서의 높은 비트 전송 속도를 가능하게 합니다.

EnDat 엔코더에서의 위치를 통지하기 위한 **Gate3[i].Chan[j].SerialEncCmd** 의 전형적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord: = 7	// 엔코더가 위치를 통지하는 명령 워드
SerialEncParity: = 0	// 패리티 확인은 EnDat 프로토콜에서는 // 지원되지 않습니다.
SerialEncTrigMode: = 0	// 연속 트리거(EnDat2.2)
SerialEncTrigEna: = 1	// 트리거 유효
SerialEncGtoB: = 0	// 그레이 코드는 EnDat 프로토콜에서는 지원 // 되지 않습니다.
SerialEncDataReady: = 0	// 읽기 전용 Status 비트
SerialEncStatusBits: = 0	// Status 비트는 EnDat 프로토콜에서는 // 지원되지 않습니다.
SerialEncNumBits: = ??	// 회신되는 위치 비트의 엔코더 특유의 // 비트 수

예를 들어 37 비트의 위치 비트를 갖춘 EnDat2.2 엔코더에 대해서는 연속적으로 위치를 알리도록 **Gate3[i].Chan[j].SerialEncCmd** 를 \$00071025 로 설정합니다. (데이터 준비 완료 Status 비트가 설정된 경우 \$00071425 로 통지됩니다.)

0								0								0				7				1				0					2			5				
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0									
0									0	0	0	1	1	1			0	1									1	0	0	1	0	1								
DC								SerialEncCmdWord								Parity				TM	TE	GB Rdy					Status					NumBits								

이 엔코더에서 지연 시간 확인과 보정 사이클을 실시하기 위해서는 **Gate3[i].Chan[j].SerialEncCmd** 를 \$80071025 로 설정한 후 MSB 가 클리어되도록 기다리십시오.

8								0							0				7							1				0							2				5						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																
1	-	-	-	-	-	-	-	-	-	0	0	0	0	1	1	1	-	-	0	1	-	-	-	-	-	-	1	0	0	1	0	1															
DC								SerialEncCmdWord								Parity				TM	TE	GB Rdy					Status					NumBits															

Gate3[i].Chan[j].SerialEncCmd 를 \$002A3025 로 설정하면 원쇼트 모드에서 송신하는 42(\$2A)의 명령 워드값으로 이 동일한 엔코더를 리셋할 수 있습니다.

0								0								2				A				3				0				2				5																			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																								
0								-								1				0				1				0				1				0				0				1				0				1			
DC								SerialEncCmdWord																Parity				TM	TE	GB Rdy				Status				NumBits																	

24 비트의 위치 비트를 갖춘 EnDat2.1 엔코더에 대해서는(전원 투입으로) 1 번만 위치를 알리도록 **Gate3[i].Chan[j].SerialEncCmd** 를 \$00073018 로 설정합니다.

0								0								0				7				3				0					1			8				
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0									
0										0	0	0	1	1	1			1	1								0	1	1	0	0	0								
DC								SerialEncCmdWord								Parity				TM	TE	GB Rdy					Status					NumBits								

Hiperface 프로토콜

DSPGATE3 IC Hiperface 인터페이스는 엔코더에 대한 3 개의 8 비트 명령 코드를 지원합니다. 위치를 알리기 위한 \$42, Status 를 알리기 위한 \$50 및 엔코더를 리셋하기 위한 \$53 입니다. 이러한 8 비트는 **Gate3[i].Chan[j].SerialEncCmd** 의 16 비트의 *SerialEncCmdWord* 명령 필드의 하위 비트로 할당됩니다.

SerialEncCmdWord 의 상위 8 비트에는 인터페이스의 엔코더 주소가 포함됩니다. Hiperface 프로토콜은 8 대까지의 독립된 엔코더가 1 개의 멀티 드롭 인터페이스로 "데이터 체인 연결"을 가능하게 합니다. ACC-24E3 을 사용하면 이 기능은 가능해지지만 배선을 단순화하기 위해 ACC-24E3 의 각각의 채널에 개별 엔코더를 연결하는 것도 가능합니다. 이 구성에서는 이 주소 필드는 엔코더의 주소 값(+ \$40)과 일치시키거나 또는 \$FF(브로드캐스트 모드)로 설정할 수 있습니다.

Hiperface 엔코더에서의 위치를 통지하기 위한 **Gate3[i].Chan[j].SerialEncCmd** 의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord: = \$4042 // 주소 0에서의 엔코더에 위치를 통지시킴
SerialEncParity: = ?? // 엔코더 고유의 패리티 확인
SerialEncTrigMode: = 1 // 원쇼트 트리거
SerialEncTrigEna: = 1 // 트리거 유효(원쇼트 후 클리어)
SerialEncGtoB: = 0 // 그레이 코드는 Hiperface 프로토콜에서는
// 지원되지 않습니다.
SerialEncDataReady: = 0 // 읽기 전용 Status 비트
SerialEncStatusBits: = 0 // Status 비트는 Hiperface 프로토콜에서는
// 지원되지 않습니다.
SerialEncNumBits: = ?? // 위치 비트의 엔코더 고유의 비트 수 통지

예를 들어 홀수 패리티를 갖춘 사용자 주소 0의 Hiperface 엔코더인 경우에 원쇼트의 위치 통지하는 것처럼 **Gate3[i].Chan[j].SerialEncCmd**를 \$40427000으로 설정합니다. (트리거 유효 비트가 클리어되고 데이터 준비 완료 Status 비트가 설정된 경우 \$40426400으로 통지됩니다.)

4				0				4				2				7				0				0				0			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-
SerialEncCmdWord																Parity		TM	TE	GB Rdy		Status				NumBits					

Yaskawa 시그마 I 프로토콜



Yaskawa는 현재 시그마 I 애플솔루트 엔코더를 제조하지 않습니다. 그러나 새로운 시그마 II, III 및 V 엔코더를 사용하는 중이라도 신세대 Yaskawa 시그마 서보 드라이브는 컨트롤러로 복귀하기 위해 Yaskawa 시그마 I 프로토콜을 합성합니다.

Yaskawa 시그마 I 엔코더용 **Gate3[i].Chan[j].SerialEncCmd**의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord: = 1 // 엔코더를 스트로브하기 위한 비트 설정
SerialEncParity: = 0 // Yaskawa 프로토콜의 고정 패리티 확인
SerialEncTrigMode: = 1 // 원쇼트 트리거
SerialEncTrigEna: = 1 // 트리거 유효(원쇼트 완료 시 클리어)
SerialEncGtoB: = 0 // 그레이 코드는 Yaskawa 프로토콜에서는
// 지원되지 않습니다.
SerialEncDataReady: = 0 // 읽기 전용 Status 비트
SerialEncStatusBits: = 0 // Status 비트는 Yaskawa 프로토콜에서는
// 지원되지 않습니다.
SerialEncNumBits: = 0 // 회신되는 고정 위치 비트 수

원쇼트의 위치 통지를 위해 **Gate3[i].Chan[j].SerialEncCmd** 를 \$00013000 으로 설정합니다. (원쇼트 종료 시 트리거 유효 비트가 클리어되고 준비 완료 Status 비트가 설정된 경우 \$00002400 으로 통지됩니다.)

0				0				0				1				3				0				0				0			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	1	1	-	-	-	-	-	-	-	-	-	-	-	-
SerialEncCmdWord																Parity		TM	TE	GB	Rdy	Status				NumBits					

Yaskawa 시그마 II/III/IV 프로토콜

Yaskawa 시그마 II/III/IV 엔코더용 **Gate3[i].Chan[j].SerialEncCmd** 의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord:	= 0	// Yaskawa 에 위치를 통지하기 위한 명령 워드는 // 없습니다.
SerialEncParity:	= 0	// 패리티 확인은 Yaskawa 프로토콜에서는 // 지원되지 않습니다.
SerialEncTrigMode:	= 0	// 연속 트리거
SerialEncTrigEna:	= 1	// 트리거 유효
SerialEncGtoB:	= 0	// 그레이 코드는 Yaskawa 프로토콜에서는 // 지원되지 않습니다.
SerialEncDataReady:	= 0	// 읽기 전용 Status 비트
SerialEncStatusBits:	= 0	// Status 비트는 Yaskawa 프로토콜에서는 // 지원되지 않습니다.
SerialEncNumBits:	= 0	// 회신되는 고정 위치 비트 수

연속적인 위치 통지를 위해 **Gate3[i].Chan[j].SerialEncCmd** 를 \$00001000 으로 설정합니다. (준비 완료 Status 비트가 설정된 경우 \$00001400 으로 통지됩니다.)

0		0				0				0				1		0				0				0							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SerialEncCmdWord																Parity		TM	TE	GB	Rdy	Status				NumBits					

Tamagawa FA 코더 프로토콜

Tamagawa FA 코더 엔코더용 **Gate3[i].Chan[j].SerialEncCmd** 의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord:	= \$1A	// Tamagawa 에서 위치를 통지하기 위한 명령 워드
SerialEncParity:	= 0	// 패리티 확인은 Tamagawa 프로토콜에서는 // 지원되지 않습니다.
SerialEncTrigMode:	= 0	// 연속 트리거
SerialEncTrigEna:	= 1	// 트리거 유효
SerialEncGtoB:	= 0	// 그레이 코드는 Tamagawa 프로토콜에서는 // 지원되지 않습니다.
SerialEncDataReady:	= 0	// 읽기 전용 Status 비트
SerialEncStatusBits:	= 0	// Status 비트는 Tamagawa 프로토콜에서는 // 지원되지 않습니다.
SerialEncNumBits:	= 0	// 회신되는 고정 위치 비트 수

연속적인 위치 통지를 위해 **Gate3[i].Chan[j].SerialEncCmd** 를 \$001A1000 으로 설정합니다. (준비 완료 Status 비트가 설정된 경우 \$001A1400 으로 통지됩니다.)

0		0				1				A				1				0				0				0							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
-	-	-	-	-	-	-	-	0	0	0	1	1	0	1	0	-	-	0	1	-	-	-	-	-	-	-	-	-	-	-	-		
SerialEncCmdWord																Parity		TM	TE	GB	Rdy	Status				NumBits							

SerialEncCmdWord 구성 요소를 \$BA, \$C2 또는 \$62 로 설정하는 경우에 엔코더의 멀티턴값을 0 으로 리셋합니다. 이는 "원쇼트" 모드에서 수행해야 하며 요소는 각각 \$00BA3000, \$00C23000 또는 \$00623000 입니다. 리셋 동작이 수행되면 구성 요소는 각각 \$00BA2000, \$00C22000 또는 \$00622000 으로 통지될 것입니다.

Panasonic 프로토콜

Panasonic 시리얼 엔코더용에서의 **Gate3[i].Chan[j].SerialEncCmd** 의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord: = \$2A // Panasonic 멀티턴 위치의 명령 워드
SerialEncParity: = 0 // 패리티 확인은 Panasonic 프로토콜에서는
// 지원되지 않습니다.
SerialEncTrigMode: = 0 // 연속 트리거
SerialEncTrigEna: = 1 // 트리거 유효
SerialEncGtoB: = 0 // 그레이 코드는 Panasonic 프로토콜에서는
// 지원되지 않습니다.
SerialEncDataReady: = 0 // 읽기 전용 Status 비트
SerialEncStatusBits: = 0 // Status 비트는 Panasonic 프로토콜에서는
// 지원되지 않습니다.
SerialEncNumBits: = 0 // 회신되는 고정 위치 비트 수

연속적인 멀티턴 위치 통지를 위해 **Gate3[i].Chan[j].SerialEncCmd** 를 \$002A1000 으로 설정합니다. (데이터 준비 완료 Status 비트가 설정된 경우 \$002A1400 으로 통지됩니다.)

0		0				2				A				1		0				0				0							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	0	0	1	0	1	0	1	0	-	-	0	1	-	-	-	-	-	-	-	-	-	-	-	-
SerialEncCmdWord																Parity		TM	TE	GB	Rdy	Status				NumBits					

이상 코드와 함께 싱글턴 위치를 통지하기 위해 *SerialEncCmdWord* 구성 요소를 \$52 로 설정하는 경우에 **Gate3[i].Chan[j].SerialEncCmd** 는 \$00521000 으로 설정됩니다. (데이터 준비 완료 Status 비트가 설정된 경우 \$00521400 으로 통지됩니다.) 이 경우에 보통 멀티턴 위치가 통지되는 곳에 엔코더 ID 값이 통지됩니다.

SerialEncCmdWord 구성 요소를 \$4A, \$7A, \$DA 또는 \$F2 로 설정하는 경우에 엔코더의 멀티턴 위치는 0 으로 리셋됩니다. 이는 "원쇼트" 모드에서 수행해야 하며 요소는 각각 \$004A3000, \$007A3000, \$00DA3000 또는 \$00F23000 입니다. 리셋 동작이 수행되면 구성 요소는 각각 \$004A2000, \$007A2000, \$00DA2000 또는 \$00F22000 으로 통지됩니다.

Mitutoyo 프로토콜

Mitutoyo 시리얼 엔코더용 **Gate3[i].Chan[j].SerialEncCmd** 의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord: = \$01 // Mitutoyo 에서 위치를 통지시키는 명령 워드
SerialEncParity: = 0 // 패리티 확인은 Mitutoyo 프로토콜에서는
// 지원되지 않습니다.
SerialEncTrigMode: = 0 // 연속 트리거
SerialEncTrigEna: = 1 // 트리거 유효
SerialEncGtoB: = 0 // 그레이 코드는 Mitutoyo 프로토콜에서는
// 지원되지 않습니다.
SerialEncDataReady: = 0 // 읽기 전용 Status 비트
SerialEncStatusBits: = 0 // Status 비트는 Mitutoyo 프로토콜에서는
// 지원되지 않습니다.
SerialEncNumBits: = 0 // 회신되는 고정 위치 비트 수

연속적인 위치 통지를 위해 **Gate3[i].Chan[j].SerialEncCmd** 를 \$00011000 으로 설정합니다. (데이터 준비 완료 Status 비트가 설정된 경우 \$00011400 으로 통지됩니다.)

0		0		0		1		1		0		0		0																	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	1	-	-	0	1	-	-	-	-	-	-	-	-	-	-	-	-
SerialEncCmdWord																Parity		TM	TE	GB	Rdy	Status				NumBits					

SerialEncCmdWord 구성 요소를 \$89 로 설정하는 경우에 엔코더 중의 멀티턴 위치값이 0 으로 리셋됩니다(8 사이클 후). **SerialEncCmdWord** 구성 요소를 \$9D 로 설정하는 경우에 **SerialEncDataB** 의 비트 8-15 에서 엔코더 ID 값을 통지합니다. **SerialEncCmdWord** 구성 요소를 \$85 로 설정하는 값이 \$01 과 동일하게 절대 위치가 통지됩니다.

Kawasaki 프로토콜

Kawasaki 시리얼 엔코더용 **Gate3[i].Chan[j].SerialEncCmd** 의 대표적인 설정은 다음 목록과 같습니다.

SerialEncCmdWord: = \$00 // Kawasaki 프로토콜에서는 명령 워드는
// 없습니다.
SerialEncParity: = 0 // 패리티 확인은 Kawasaki 프로토콜에서는
// 지원되지 않습니다.
SerialEncTrigMode: = 0 // 연속 트리거
SerialEncTrigEna: = 1 // 트리거 유효
SerialEncGtoB: = 0 // 그레이 코드는 Kawasaki 프로토콜에서는
// 지원되지 않습니다.
SerialEncDataReady: = 0 // 읽기 전용 Status 비트
SerialEncStatusBits: = 0 // Status 비트는 Kawasaki 프로토콜에서는
// 지원되지 않습니다.
SerialEncNumBits: = 0 // 회신되는 고정 위치 비트 수

연속적인 위치 통지를 위해 **Gate3[i].Chan[j].SerialEncCmd** 를 \$00001000 으로 설정합니다.

0		0		0		0		1		0		0		0																			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
SerialEncCmdWord																Parity		TM	TE	GB	Rdy	Status				NumBits							

Gate3[i].Chan[j].SerialEncEna 게이트 3 채널의 시리얼 엔코더 유효

설명: IC 채널의 시리얼 엔코더 유효

범위: 0 ..1

단위: 논리값

기본값: 0

Gate3[i].Chan[j].SerialEncEna 은 IC 채널의 시리얼 엔코더의 인터페이스를 유효로 설정할지 여부를 제어합니다. 1 로 설정하는 경우에 인터페이스가 유효가 되고 또한 채널용 SENC_MODEn 은 high 로 유지됩니다(이 신호로 많은 제품 중의 시리얼 엔코더 드라이버 및 리시브 회로를 유효로 설정할 수 있습니다). 0 으로 설정하는 경우에 시리얼 인터페이스는 무효가 되고 또한 채널용 SENC_MODEn 출력 핀은 low 로 유지됩니다. 그러나 이 경우에 인크리멘탈 엔코더에서 AB 상 입력을 수신하기 위해 시리얼 클록 및 데이터 핀을 사용할 수 있으므로 상태 요소 **Gate3[i].Chan[j].SerialEncDataA** 의 카운터 위치도 이용 가능해집니다.

Gate3[i].Chan[j].SerialEncEna 은 풀 워드 요소 **Gate3[i].Chan[j].InCtrl** 의 비트 20 을 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey** 에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 **Sys.WpKey** 에 들어 있는 경우에 스크립트 환경에서는 자동으로 실행됩니다.)

Gate3[i].Chan[j].TimerMode 게이트 3 채널의 타이머 모드

설명: IC 채널의 엔코더 타이머 제어

범위: 0 ..3

단위: 없음

기본값: 0

Gate3[i].Chan[j].TimerMode 는 어떻게 채널의 엔코더 타이머를 사용할지 또한 어떤 정보가 채널의 타이머 레지스터에 표시될지를 제어합니다. 다음 설정을 사용할 수 있습니다.

- **Gate3[i].Chan[j].TimerMode = 0:** 하드웨어 1/T
- **Gate3[i].Chan[j].TimerMode = 1:** MLDT 펄스 에코의 타이밍
- **Gate3[i].Chan[j].TimerMode = 2:** 트리거 입력 타이밍
- **Gate3[i].Chan[j].TimerMode = 3:** PFM 펄스 계수

Gate3[i].Chan[j].TimerMode=0: "하드웨어 1/T" 모드에서는 끝에서 2 개의 디코드 카운터 사이의 시간과 마지막으로 카운트했을 때부터의 시간을 측정하기 위해 채널의 타이머를 사용하고(디코드 모드에 관계없음) 양쪽은 엔코더 SCLK 사이클로 측정됩니다. 각 SCLK 사이클로 측정되는 소수점 카운트값을 산출하기 위해 이러한 값의 비율을 사용합니다. 또한 정수 카운트값과 결합된 값은 서보 인터럽트, 위상 인터럽트 또는 지정된 트리거 조건에서 언제든지 래치할 수 있습니다. 그리고 이러한 태스크 모두에 "서브 카운트"의 분해능을 부여합니다. 또한 서브 카운트 분해능 기능에서 "위치 비교" A와 B의 출력 레지스터를 이용할 수도 있습니다.

이 모드에서는 채널의 32 비트 **PhaseCapt**, **ServoCapt** 및 **HomeCapt** 레지스터는 모두 해당 하위 바이트의 1/T 회로로 산출된 8비트의 소수점 카운트값을 포함합니다. 채널의 **TimerA** 및 **TimerB** 레지스터는 양쪽 모두 해당 최하위 비트에서 1/T 회로로 산출된 12 비트의 소수점 카운트값을 포함합니다. **TimerA** 는 서보 인터럽트로 래치되고 **TimerB** 는 선택된 포착 트리거로 래치됩니다.

Gate3[i].Chan[j].TimerMode=1: "MLDT 펄스 에코의 타이밍" 모드에서는 자왜식 선형 변위 변환기(MLDT)에 대해 자동으로 생성된 PULSEn(PWM_BOT_Dn) 출력과 CHAn의 "에코 펄스" 수신까지의 시간을 측정하기 위해 채널의 타이머를 사용합니다. 이 시간은 베이스부터 가동부 센서까지의 거리에 비례합니다.

PFM 회로의 출력 펄스로 타이머가 0 으로 리셋됩니다. 이는 자동으로 서보 사이클과 동기합니다. 그 후 이는 600MHz 속도로 카운트 업합니다. 이 모드에서는 채널의 **TimerA** 레지스터는 "에코" 카운트 펄스를 수신하여 래치된 타이머값이 되고 센서의 절대 위치를 표시합니다. 채널의 **TimerB** 레지스터는 다음 출력 펄스를 전송하고 타이머를 0 으로 리셋하기 전에 래치된 타이머값이 들어 있습니다. 이 값은 주로 기준용입니다. 이 모드에서는 채널의 32 비트 **PhaseCapt**, **ServoCapt** 및 **HomeCapt** 레지스터의 하위 8비트는 모두 \$80 의 값이 됩니다. 1/2 의 사실상의 소수점 카운트값을 제공합니다. 1/T 서브 카운트 확장이 무효이기 때문입니다(만약 채널용 **AtanEna** 이 1 로 설정되지 않은 경우에 아크탄젠트 서브 카운트 확장은 사용되지 않습니다).

Gate3[i].Chan[j].TimerMode=2: "트리거 입력 타이밍" 모드에서는 채널의 카운터 회로를 사용하지 않는 센서의 서보 사이클 시작부터 "하드웨어 포착" 기능이 유효가 될 때까지의 시간을 계측하기 위해 채널의 타이머가 사용됩니다.

각 서보 사이클의 시작에 타이머가 0 으로 리셋되고 다음으로 SCLK 주파수로 카운트 업합니다. 지정된 포착 트리거 조건이 서보 사이클 중에 일어나면 **TimerA** 레지스터에 최대 타이머값을 래치합니다. 다음 서보 시작 시에 **TimerB** 레지스터에 제 2 타이머값을 래치합니다. 또한 트리거가 일어난 서보 사이클 내의 시기를 계산하기 위해 이러한 타이머의 비율을 사용할 수 있습니다. 그 후 트리거가 일어난 위치를 보간하기 위해서도 이를 사용할 수 있습니다. 이 모드에서는 채널의 32 비트 **PhaseCapt**, **ServoCapt** 및 **HomeCapt** 레지스터의 하위 8비트는 모두 \$80 의 값이 됩니다. 1/2 의 사실상의 소수점 카운트값을 제공합니다. 1/T 서브 카운트 확장이 무효이기 때문입니다(만약 채널용 **AtanEna** 이 1 로 설정되지 않은 경우에 아크탄젠트 서브 카운트 확장은 사용되지 않습니다).

Gate3[i].Chan[j].TimerMode=3: PFM 펄스 카운트 모드에서는 채널의 PFM 출력으로 생성된 펄스를 카운트하기 위해 채널의 타이머를 사용합니다. 연결은 ASIC 내부에서 이루어집니다. 외부 배선은 필요하지 않습니다. 이 모드에서는 채널의 PFM 신호를 펄스 및 방향 명령 입력 스텝핑 모터 드라이브에 출력합니다. 하나의 채널에서 Power PMAC에 타이머 레지스터값에 따른 시뮬레이트 루프를 실행시키고 또한 "확인 엔코더"로 사용하도록 할 수 있습니다.

이 모드에서는 채널의 **TimerA** 레지스터에는 서보 인터럽트로 래치된 펄스 카운트값이 있습니다. 시뮬레이트 서보 루프를 실행하기 위해 사용할 수 있습니다. 채널의 **TimerB** 레지스터에는 선택된 트리거로 래치된 펄스 카운트값이 있습니다. 이 모드에서는 채널의 32비트 **PhaseCapt**, **ServoCapt** 및 **HomeCapt** 레지스터의 하위 8비트는 모두 \$80의 값이 됩니다. 1/2의 사실상의 소수점 카운트값을 제공합니다. 1/T 서브 카운트 확장이 무효이기 때문입니다(만약 채널용 **AtanEna** 이 1로 설정되지 않은 경우에 아크탄젠트 서브 카운트 확장은 사용되지 않습니다).

Gate3[i].Chan[j].TimerMode 는 풀 워드 요소 **Gate3[i].Chan[j].InCtrl** 의 비트 4-5를 구성합니다. 이 요소는 쓰기 방지되었으므로 처음에 **Gate3[i].WpKey**에 대해 정상적인 키값을 쓰지 않으면 변경할 수 없습니다. (적절한 키값이 Sys.WpKey에 들어 있는 경우에 스크립트 환경 중에서 자동으로 발생합니다.)

Gatelo[j].저장 데이터 구조체 요소(V1.6 2014.03)

Gatelo[j].Init 서브 구조체의 저장 설정 요소는 IOGATE 디지털 I/O ASIC 의 설정 레지스터를 초기화하기 위해 사용됩니다. 해당 요소는 실제로는 IOGATE IC 에 존재하지 않습니다. 엄밀하게는 이는 메모리 레지스터이며 그 내용이 Power PMAC 전원 투입/리셋 처리 시에 IC 의 설정 레지스터에 복사됩니다. \$\$\$** 명령을 사용해서 Power PMAC 이 다시 초기화되면 이러한 요소는 IOGATE IC 가 존재하는 자동 검출된 액세스러리에 적합한 기본값으로 각각 설정됩니다.

표준 전원 투입/리셋 처리 중에 IOGATE IC 의 하드웨어 설정 레지스터는 최초로 공장 설정 기본값을 수신하고 다음으로 이러한 메모리 설정 요소에서 저장된 값을 수신합니다. 이러한 메모리 설정 요소가 전원 투입/리셋 처리 시에만 사용된다는 점에 주의하십시오. 따라서 IOGATE IC 의 구성을 변경하려면 먼저 메모리 설정 요소 값을 변경하고 해당 설정은 **save** 명령으로 플래시 메모리에 복사해야 합니다. 그리고 마지막으로 Power PMAC 을 리셋해야 합니다.

다음 별칭으로도 Gatelo 구조체를 나타낼 수 있다는 점에 주의하십시오. -**Acc14E[j]**, **Acc65E[j]**, **Acc66E[j]**, **Acc67E[j]** 및 **Acc68E[j]**- 그리고 백업 파일에 별칭으로 저장됩니다. 인덱스값 i 는 0~15 범위이며 액세스러리 카드의 하드웨어 주소 DIP 스위치에 의해 결정됩니다.

Gatelo[j].Init.CtrlReg 게이트 IO 의 레지스터 방향 초기화 제어

설명: IOGATE 레지스터의 방향 초기화 제어

범위: \$00..\$FF(0.. 255)

단위: 비트 필드

기본값: 구성에 좌우됨

소프트웨어 요소 **Gatelo[j].Init.CtrlReg** 에는 전원 투입/리셋 시의 IC 에 대한 구성 처리 끝에 IC 하드웨어 설정 레지스터 **Gatelo[j].CtrlReg** 에 자동으로 쓰여지는 값이 저장됩니다. 이 8 비트 요소의 하위 6 비트만 방향 제어에 사용됩니다. 6 개의 데이터 레지스터에 실제 데이터를 쓰거나 읽을 수 있도록 상위 2 비트는 이 구성 처리의 가장 마지막에 0 으로 설정되어야 합니다. 이 요소를 0~63 의 유효 범위로 설정합니다. 구성 처리 전반에 상위 2 비트는 6 개의 데이터 레지스터를 구성하는 처리 중에 자동으로 사용됩니다.

IC 인덱스값 i 는 0~15 범위이며 보드의 주소 스위치에 의해 설정되는 하드웨어 IC 의 인덱스와 일치해야 합니다.

Gatelo[j].Init.CtrlReg 의 비트 $j(j = 0\sim5)$ 는 **Gatelo[j].DataReg[j]**의 I/O 8 점의 방향을 결정합니다. 비트값이 0 인 경우에 데이터 레지스터에 대한 쓰기 조작을 가능하게 하고 데이터 레지스터에서의 각 라인의 출력 기능을 유효로 합니다. 출력이 OFF 인(비전도) 동안은 출력 기능을 유효로 설정해도 라인 중 어느 하나 또는 전부를 입력으로 사용하는 것을 막을 수 없습니다.

비트값이 1 인 경우에 데이터 레지스터에 대한 쓰기 조작을 허가하지 않습니다. 출력을 무효로 설정하여 입력을 위해 레지스터를 확보합니다. 이러한 라인을 입력으로 사용할 때는 데이터 레지스터에 대한 쓰기 조작이 입력을 무효로 설정하지 않도록 이 설정을 적극 권장합니다.

다시 초기화할 때에 자동 검출 처리에 의해 인식된 IOGATE IC 를 이용하는 디지털 I/O 액세스서를 위해 Power PMAC 은 이 요소의 값을 자동으로 설정합니다. 이 설정값은 특정한 액세스서의 가장 일반적인 사용 방법을 가능하게 합니다.

Gatelo[j].Init.DataReg0[j] 게이트 IO 의 전원 투입 기본값

설명: IOGATE 레지스터의 전원 투입 기본값

범위: \$00..\$FF(0.. 255)

단위: 비트 필드

기본값: 0

소프트웨어 요소 **Gatelo[j].Init.DataReg0[j]**에는 전원 투입/리셋 시의 하드웨어 설정 요소 **Gatelo[j].CtrlReg** 가 0 으로 설정될 때(비트 7 = 0, 비트 6 = 0) IC 하드웨어 설정 레지스터 **Gatelo[j].DataReg[j]**에 자동으로 쓰여지는 값이 저장됩니다. **Gatelo[j].Init.DataReg0[j]**의 새로운 값을 하드웨어에 반영시키려면 이 값을 **save** 명령으로 비휘발성 플래시 메모리에 저장하고 또한 Power PMAC 이 리셋되어야 합니다.

IC 인덱스값 i 는 0~15 범위이며 보드의 주소 스위치에 의해 설정되는 하드웨어 IC 의 인덱스와 일치해야 합니다. 레지스터 인덱스값 j 는 0~5 범위이며 IC 의 하드웨어 데이터 레지스터의 주소 오프셋값과 일치해야 합니다. 데이터 레지스터는 각각 8 점의 I/O 에 상당합니다.

8 비트 요소 **Gatelo[j].Init.DataReg0[j]**의 저장된 값은 하드웨어 레지스터 **Gatelo[j].DataReg[j]**에 관련된 8 개 출력의 초기값입니다. 비트에는 각각 번호 순서대로 일치하는 출력 포인트의 초기값을 지정합니다. 레지스터가 입력에 사용되는 경우에 이 요소의 저장된 값은 사용되지 않습니다.

다시 초기화할 때에 자동 검출 처리에 의해 인식된 IOGATE IC 를 이용하는 디지털 I/O 액세스서를 위해 Power PMAC 은 이 요소의 값을 자동으로 설정합니다. 기본값은 모든 출력을 "off" 상태로 설정합니다.

Gatelo[j].Init.DataReg64[j] 게이트 IO 의 레지스터 반전 초기화 제어

설명: IOGATE 레지스터의 반전 초기화 제어

범위: \$00..\$FF(0.. 255)

단위: 비트 필드

기본값: 구성에 좌우됨

소프트웨어 요소 **Gatelo[j].Init.DataReg64[j]**에는 전원 투입/리셋 시의 하드웨어 설정 요소 **Gatelo[j].CtrlReg** 가 64 로 설정될 때(비트 7 = 0, 비트 6 = 1) IC 하드웨어 설정 레지스터 **Gatelo[j].DataReg[j]**에 자동으로 쓰여지는 값이 저장됩니다. **Gatelo[j].Init.DataReg64[j]**의 새로운 값을 하드웨어에 반영시키려면 해당 값을 **save** 명령으로 비휘발성 플래시 메모리에 저장하고 또한 Power PMAC 이 리셋되어야 합니다.

IC 인덱스값 i 는 0~15 범위이며 IC 의 주소 스위치에 의해 설정되는 하드웨어 IC 의 인덱스와 일치해야 합니다. 레지스터 인덱스값 j 는 0~5 범위이며 IC 의 하드웨어 데이터 레지스터의 주소 오프셋과 일치해야 합니다. 데이터 레지스터는 각각 8 점의 I/O 에 상당합니다.

Gatelo[j].Init.DataReg64[j]의 저장된 값은 하드웨어 레지스터 **Gatelo[j].DataReg[j]**에 대응한 I/O 8 점의 반전을 결정합니다. 비트에는 각각 번호 순서대로 일치하는 I/O 포인트의 반전을 지정합니다. 비트값이 0 인 경우에 일치하는 비트에서의 I/O 포인트의 반전을 지정합니다. 즉 출력에 관해서는 0 의 출력값은 IC 자체에서 low(전도) 출력을 생성합니다. 또한 1 의 출력값은 high(비전도) 출력을 생성합니다. 입력에 관해서는 low 로 풀다운된 라인은 1 개의 입력값입니다. 또한 high 로 풀다운되거나 또는 high 로 플롯되는 라인은 0 의 입력값입니다.

Gatelo[j].Init.DataReg64[j]의 비트값이 1 인 경우에 일치하는 비트에서의 I/O 포인트의 비반전을 지정합니다. 즉 출력에 관해서는 0 의 출력값은 IC 자체에서 high(비전도) 출력을 생성합니다. 그리고 1 의 출력값은 low(전도) 출력을 생성합니다. 입력에 관해서는 low 로 풀다운된 IC 의 라인은 0 의 입력값이고 또한 high 로 풀업되거나 또는 high 로 플롯되는 라인은 1 의 입력값입니다.

외부 배선과의 연결에 아이솔레이터 및 드라이버 IC 를 사용하는 IOGATE 액세서리 보드(예를 들면 ACC-65E, ACC-66E, ACC-67E 및 ACC-68E)의 반전 설정은 "전도" 상태가 싱크 또는 소스에 관계없이 입력 및 출력 양쪽에서 데이터 레지스터 비트의 "1"에 상당한다는 것을 의미합니다.

IC 핀을 외부 배선에 직접 연결하고 보드 상에 풀업 저항이 있는 IOGATE 액세서리 보드(예를 들면 ACC-14E)의 출력의 반전 설정은 데이터 레지스터의 비트에 "1"을 썼을 때 IC 핀의 싱크 트랜지스터가 ON 되고 출력 라인이 low 로 떨어진다는 것을 의미합니다. 데이터 레지스터의 비트에 "0"을 썼을 때 트랜지스터가 OFF 되고 외부에 설치한 저항이 신호 라인을 high 로 풀업합니다. 입력의 반전 설정은 입력 핀이 활성 low 일 때 데이터 레지스터의 비트가 "1"로 쓰여진다는 것을 의미합니다. 입력 핀이 활성 high 로 되거나 외부에 설치한 저항기에 의해 풀업되면 데이터 레지스터의 비트는 "0"으로 쓰여집니다.

다시 초기화할 때에 자동 검출 처리에 의해 인식된 IOGATE IC 를 이용하는 디지털 I/O 액세서리를 위해 Power PMAC 은 이 요소의 값을 자동으로 설정합니다. 이 설정값은 특정한 액세서리의 가장 일반적인 사용 방법을 가능하게 합니다.

Gatelo[j].Init.DataReg128[j] 게이트 IO의 레지스터 읽기 초기화 제어

설명: IOGATE의 레지스터 읽기 초기화 제어

범위: \$00..\$FF(0.. 255)

단위: 비트 필드

기본값: 0

소프트웨어 요소 **Gatelo[j].Init.DataReg128[j]**에는 전원 투입/리셋 시의 하드웨어 설정 요소 **Gatelo[j].CtrlReg**가 128로 설정될 때(비트 7 = 1, 비트 6 = 0) IC 하드웨어 설정 레지스터 **Gatelo[j].DataReg[j]**에 자동으로 쓰여지는 값이 저장됩니다. **Gatelo[j].Init.DataReg128[j]**의 새로운 값을 하드웨어에 반영시키려면 해당 값을 save 명령으로 비휘발성 플래시 메모리에 저장하고 또한 Power PMAC이 리셋되어야 합니다.

IC 인덱스값 i 는 0~15 범위이며 보드의 주소 스위치에 의해 설정되는 하드웨어 IC의 인덱스와 일치해야 합니다. 레지스터 인덱스값 j 는 0~5 범위이며 IC의 하드웨어 데이터 레지스터의 주소 오프셋과 일치해야 합니다. 데이터 레지스터는 각각 8점의 I/O에 해당합니다.

Gatelo[j].Init.DataReg128[j]에 저장된 값은 **Gatelo[j].Init.DataReg192[j]**의 저장된 값과 연계하여 하드웨어 레지스터 **Gatelo[j].DataReg[j]**의 읽기 동작에 의해 8점의 I/O에 의해 무엇이 통지될지 결정합니다. 비트에는 각각 번호 순서대로 일치하는 I/O 포인트의 통지를 지정합니다.

Gatelo[j].Init.DataReg128[j]의 비트 동작은 동일한 인덱스 j 에서 **Gatelo[j].Init.DataReg192[j]**의 일치하는 비트 설정에 좌우됩니다. **Gatelo[j].Init.DataReg192[j]**의 일치하는 비트값이 0인 경우에 래치하지 않은 입력이 선택되고 **Gatelo[j].Init.DataReg128[j]**의 비트에 의해 핀의 값을 읽을지 또는 쓰기 가능한 레지스터의 값을 읽을지 여부를 제어합니다. **Gatelo[j].Init.DataReg128[j]**의 비트값이 0이면 인덱스 j 에서 **Gatelo[j].DataReg[j]**의 일치하는 비트에서 핀 값을 읽는 것을 선택합니다. 비트값이 1이면 쓰기 가능한 레지스터값을 읽는 것을 선택합니다.

Gatelo[j].Init.DataReg192[j]의 일치하는 비트값이 1인 경우에 래치 입력이 선택되고 **Gatelo[j].Init.DataReg128[j]**의 비트는 직접 래치된 데이터를 읽을지 또는 그레이 코드에서 바이너리로의 변환 결과를 읽을지 여부를 제어합니다. **Gatelo[j].Init.DataReg128[j]**의 비트값이 0이면 인덱스 j 에서 **Gatelo[j].DataReg[j]**의 일치하는 비트에서 직접 래치된 값을 읽는 것을 선택합니다. 비트값이 1이면 래치된 값을 그레이 코드에서 바이너리로 변환한 결과를 읽는 것을 선택합니다.

다시 초기화할 때에 자동 검출 처리에 의해 인식된 IOGATE IC를 이용하는 디지털 I/O 액세스를 위해 Power PMAC은 이 요소의 값을 자동으로 설정합니다. 이 설정값은 특정한 액세스의 가장 일반적인 사용 방법을 가능하게 합니다.

Gatelo[j].Init.DataReg192[j] 게이트 IO의 레지스터 래치 초기화 제어

설명: IOGATE 레지스터의 래치 초기화 제어

범위: \$00..\$FF(0.. 255)

단위: 비트 필드

기본값: 구성에 좌우됨

소프트웨어 요소 **Gatelo[j].Init.DataReg192[j]**에는 전원 투입/리셋 시의 하드웨어 설정 요소 **Gatelo[j].CtrlReg**가 192로 설정될 때(비트 7 = 1, 비트 6 = 1) IC 하드웨어 설정 레지스터 **Gatelo[j].DataReg[j]**에 자동으로 쓰여지는 값이 저장됩니다. **Gatelo[j].Init.DataReg192[j]**의 새로운 값을 하드웨어에 반영시키려면 해당 값을 **save** 명령으로 비휘발성 플래시 메모리에 저장하고 또한 Power PMAC이 리셋되어야 합니다.

IC 인덱스값 i 는 0~15 범위이며 보드의 주소 스위치에 의해 설정되는 하드웨어 IC의 인덱스와 일치해야 합니다. 레지스터 인덱스값 j 는 0~5 범위이며 IC의 하드웨어 데이터 레지스터의 주소 오프셋과 일치해야 합니다. 데이터 레지스터는 각각 8 점의 I/O에 해당합니다.

Gatelo[j].Init.DataReg192[j]에 저장된 값은 **Gatelo[j].Init.DataReg128[j]**의 저장된 값과 연계하여 하드웨어 레지스터 **Gatelo[j].DataReg[j]**의 읽기 동작에 의해 8 점의 I/O로 무엇이 통지될지 결정합니다. 비트에는 각각 번호 순서대로 일치하는 I/O 포인트의 통지를 지정합니다.

Gatelo[j].Init.DataReg192[j]의 비트 동작은 동일한 인덱스 j 에서 **Gatelo[j].Init.DataReg128[j]**의 일치하는 비트 설정에 좌우됩니다. **Gatelo[j].Init.DataReg128[j]**의 일치하는 비트값이 0인 경우에 **Gatelo[j].Init.DataReg192[j]**의 비트에 의해 핀의 값을 있는 그대로 읽을지 또는 직전에 래치된 값(보통 위상 또는 서보 클록의 엣지에서)을 읽을지 여부를 제어합니다. **Gatelo[j].Init.DataReg192[j]**의 비트값이 0이면 인덱스 j 에서 **Gatelo[j].DataReg[j]**의 일치하는 비트에서 핀의 값을 있는 그대로 읽는 것을 선택합니다. 비트값이 1이면 래치된 값을 읽는 것을 선택합니다.

Gatelo[j].Init.DataReg128[j]의 일치하는 비트값이 1인 경우에 **Gatelo[j].Init.DataReg192[j]**의 비트는 프로세서에 의해 레지스터에 쓰인 값을 읽을지 또는 래치 입력을 그레이 코드에서 바이너리로 변환한 값을 읽을지 여부를 제어합니다. **Gatelo[j].Init.DataReg192[j]**의 비트값이 0이면 인덱스 j 에서 **Gatelo[j].DataReg[j]**의 일치하는 비트에서 직전에 쓰인 비트의 값을 읽는 것을 선택합니다. 비트값이 1이면 래치된 값을 그레이 코드에서 바이너리로 변환한 결과를 읽는 것을 선택합니다.

다시 초기화할 때에 자동 검출 처리에 의해 인식된 IOGATE IC를 이용하는 디지털 I/O 액세스를 위해 Power PMAC은 이 요소의 값을 자동으로 설정합니다. 이 설정값은 특정한 액세스의 가장 일반적인 사용 방법을 가능하게 합니다.

Gatelo[i].Init.IntrReg64 게이트 IO의 인터럽트 라인 반전 초기화 제어

설명: IOGATE의 인터럽트 라인의 반전 초기화 제어

범위: \$00..\$FF(0.. 255)

단위: 비트 필드

기본값: 0

소프트웨어 요소 **Gatelo[i].Init.IntrReg64**에는 전원 투입/리셋 시의 하드웨어 설정 요소 **Gatelo[i].CtrlReg**가 64로 설정될 때(비트 7 = 0, 비트 6 = 1) IC 하드웨어 설정 레지스터 **Gatelo[i].IntrReg**에 자동으로 쓰여지는 값이 저장됩니다. **Gatelo[i].Init.IntrReg64**의 새로운 값을 하드웨어에 반영시키려면 해당 값을 **save** 명령으로 비휘발성 플래시 메모리에 저장하고 또한 Power PMAC이 리셋되어야 합니다.

IC 인덱스값 *i*는 0~15 범위이며 보드의 주소 스위치에 의해 설정되는 하드웨어 IC의 인덱스와 일치해야 합니다.

Gatelo[i].Init.IntrReg64의 저장된 값은 IOGATE IC에 입력되는 8개의 "En" 라인의 반전을 결정합니다. 각각의 비트는 번호 순서대로 일치하는 "En" 라인의 반전을 지정합니다. 비트값이 0인 경우에 일치하는 비트의 I/O 포인트를 반전하는 것을 지정합니다. IC에 대한 low 레벨의 신호는 1의 값을 생성하고 또한 high 레벨의 신호는 0의 값을 생성합니다.

Gatelo[i].Init.IntrReg64의 비트값이 1인 경우에 일치하는 비트의 I/O 포인트가 비반전인 것을 지정합니다. low 레벨 신호는 0의 값을 생성하고 또한 high 레벨 신호는 1의 값을 생성합니다.

다시 초기화할 때에 자동 검출 처리에 의해 인식된 IOGATE IC를 이용하는 디지털 I/O 액세스를 위해 Power PMAC은 이 요소의 값을 자동으로 설정합니다. 이 설정값은 특정한 액세스의 가장 일반적인 사용 방법을 가능하게 합니다.

현재의 Power PMAC I/O 액세스에서는 이 기능은 중요한 용도가 아닙니다. IC에 대한 En 입력은 시스템의 서보 또는 위상 클럭 신호로부터의 입력(점퍼에 의해 선택)뿐이며 입력 데이터를 래치하기 위해 사용됩니다.

Gatelo[i].Init.IntrReg128 게이트 IO의 인터럽트 라인 마스크 초기화 제어

설명: IOGATE의 인터럽트 라인의 마스크 초기화 제어

범위: \$00..\$FF(0.. 255)

단위: 비트 필드

기본값: 구성에 좌우됨

소프트웨어 요소 **Gatelo[i].Init.IntrReg128**에는 전원 투입/리셋 시의 하드웨어 설정 요소 **Gatelo[i].CtrlReg**가 128로 설정될 때(비트 7 = 1, 비트 6 = 0) IC 하드웨어 설정 레지스터

Gatelo[i].IntrReg 에 자동으로 쓰여지는 값이 저장됩니다. **Gatelo[i].Init.IntrReg128** 의 새로운 값을 하드웨어에 반영시키려면 해당 값을 **save** 명령으로 비휘발성 플래시 메모리에 저장하고 또한 Power PMAC 이 리셋되어야 합니다.

IC 인덱스값 i 는 0~15 범위이며 보드의 주소 스위치에 의해 설정되는 하드웨어 IC 의 인덱스와 일치해야 합니다.

Gatelo[i].Init.IntrReg128 의 저장된 값은 IOGATE IC 에 대한 "En" 라인 입력 중 어느 것이 IC 로부터의 "인터럽트" 출력을 생성할 수 있는지 결정합니다. 비트는 각각 번호 순서대로 일치하는 En 라인의 용도를 지정합니다. 비트값이 0 인 경우에 일치하는 En 라인의 사용을 무효로 합니다. 비트값이 1 인 경우에 일치하는 En 라인의 사용을 유효로 합니다. 해당 En 입력의 하강 엣지는 IC 의 "INT" 출력의 하강 엣지를 생성합니다.

현재의 Power PMAC I/O 액세스리에서는 이 기능은 중요한 용도가 아닙니다. IC 에 대한 En 입력은 시스템의 서보 또는 위상 클록 신호로부터의 입력(점퍼에 의해 선택)뿐이며 입력 데이터를 래치하기 위해 사용됩니다. IC 로부터의 "INT" 출력은 프로세서에 인터럽트를 걸 수 없습니다. 그러나 이는 액세스리의 "ID" 칩을 통해 읽을 수 있습니다.

Gatelo[i].Init.IntrReg192 게이트 IO 의 인터럽트 라인 엣지/레벨 초기화 제어

설명: IOGATE 의 인터럽트 라인의 엣지/레벨 초기화 제어

범위: \$00..\$FF(0.. 255)

단위: 비트 필드

기본값: 구성에 좌우됨

소프트웨어 요소 **Gatelo[i].Init.IntrReg192** 에는 전원 투입/리셋 시의 하드웨어 설정 요소 **Gatelo[i].CtrlReg** 가 192 로 설정될 때(비트 7 = 1, 비트 6 = 1) IC 하드웨어 설정 레지스터 **Gatelo[i].IntrReg** 에 자동으로 쓰여지는 값이 저장됩니다. **Gatelo[i].Init.IntrReg192** 의 새로운 값을 하드웨어에 반영시키려면 해당 값을 **save** 명령으로 비휘발성 플래시 메모리에 저장하고 또한 Power PMAC 이 리셋되어야 합니다.

IC 인덱스값 i 는 0~15 범위이며 보드의 주소 스위치에 의해 설정되는 하드웨어 IC 의 인덱스와 일치해야 합니다.

Gatelo[i].Init.IntrReg192 의 저장된 값은 IOGATE IC 에 대한 "En" 라인 입력 중 어느 것이 IC 로부터의 "인터럽트" 출력에서의 레벨 트리거 인터럽트가 될지 또는 엣지 트리거 인터럽트가 될지 여부를 결정합니다. 비트는 각각 번호 순서대로 일치하는 En 라인의 용도를 지정합니다. 비트값이 0 인 경우에 일치하는 En 라인의 레벨 트리거 인터럽트를 지정합니다. 비트값이 1 인 경우에 일치하는 En 라인의 엣지 트리거 인터럽트를 지정합니다. 입력이 인터럽트 유효일 때에만 이 설정이 유효가 됩니다.

현재의 Power PMAC I/O 액세스리에서는 이 기능은 중요한 용도가 아닙니다. IC 에 대한 En 입력은 시스템의 서보 또는 위상 클록 신호로부터의 입력(점퍼에 의해 선택)뿐이며 입력 데이터를 래치하기 위해 사용됩니다. IC 로부터의 "INT" 출력은 프로세서에 인터럽트를 걸 수 없습니다. 그러나 이는 액세스리의 "ID" 칩을 통해 읽을 수 있습니다.

Motor[x].저장 데이터 구조체 요소(V1.6 2014.03)

Motor[x] 데이터 구조체는 지정된 모터의 소프트웨어 정보를 모두 제공합니다. 이 장에서는 비휘발성 메모리에 저장되는 데이터 구조체의 요소를 설명합니다.

주의: 각 괄호 안의 모터 인덱스값은 Power PMAC 시스템의 유효한 모터 번호의 범위 내의 정수 상수로 또는 프로그램 또는 통신 계통에 의한 "로컬" L- 변수(L0-L1007 사용)에 의해 지정됩니다. 인덱스값을 지정하는 다른 방법을 활용할 수 없습니다. 인덱스값은 0 에서 **Sys.MaxMotors** - 1 까지 지정할 수 있습니다. **Sys.MaxMotors** 의 최대값은 256 입니다.

Motor[x].AbortTa 모터의 Abort 감속 시간

설명:	Abort 감속 시간 또는 감속률의 역수
범위:	부동 소수점
단위:	msec(0 이상인 경우) 또는 msec ² /모터 단위(0 미만인 경우)
기본값:	-2.0(= 0.5 모터 단위/msec ²)
기존 I 변수명:	lxx15

Motor[x].AbortTa에서는 "Abort" 시의 모터 정지까지의 시간 또는 감속도를 설정합니다. 값이 0 이상인 경우 msec 로 Abort 감속 시간을 지정합니다. 값이 0 미만인 경우 msec²/모터 단위(일반 원시 카운트)로 Abort 감속도의 최대값 역수를 지정합니다.

Motor[x].AbortTa 로 시간을 지정하고 **Motor[x].AbortTs**(S 자 곡선 시간)에서 더 작은 시간을 지정하는 경우에 전체 Abort 감속 소요 시간은 **AbortTa** 와 **AbortTs** 의 합계와 같아집니다. 그러나 **Motor[x].AbortTa** 로 시간을 지정하고 **Motor[x].AbortTs**에서 더 큰 시간을 지정하는 경우에 전체 Abort 감속 소요 시간은 **Motor[x].AbortTs** 의 2 배가 됩니다. 따라서 **Motor[x].AbortTa** 가 0 으로 설정되는 경우 **Motor[x].AbortTs**에서 "순수한" S 자 곡선 형식으로 감속 시간을 제어합니다.

Motor[x].AbortTs 가 0 미만에서 최대 "저크"를 지정한 경우에 **Motor[x].AbortTa** 도 0 미만으로 최대 가속도를 지정해야 합니다.

어떤 좌표계에서 온라인 명령 **a** 또는 다이렉트 프로그램 명령 **abort** 가 명령을 받는 경우에 해당 좌표계의 모터는 Abort 됩니다. 또한 하드웨어 또는 소프트웨어 오버트래블 리미트 중 하나를 초과하는 경우 또는 "run-time error"가 해당 좌표계에서 발생하는 경우에 그 좌표계의 모터는 자동으로 Abort 됩니다.

예/

미크론 모터 단위에서 20m/s²(약 2G)의 Abort 감속도를 설정하는 방법은 다음과 같습니다.

$$AbortTa = -\frac{s^2}{20m} * \frac{10^{-6}m}{motor\_unit} * \frac{10^6 ms^2}{s^2} = -0.05 \left(\frac{ms^2}{motor\_unit} \right)$$

Motor[x].AbortTs 모터의 Abort S 자 감속 시간

설명:	Abort S 자 곡선 감속 시간 또는 저크 레이트의 역수
범위:	부동 소수점
단위:	msec(0 이상인 경우) 또는 msec ³ /모터 단위(0 미만인 경우)
기본값:	0.0(S 자 곡선 시간 없음)
기존 I 변수명:	lxx19

Motor[x].AbortTs에서는 "Abort" 시의 모터 정지까지의 감속 S 자 곡선 부분의 시간 또는 "저크"(가속도의 변화율)의 최대값을 설정합니다. 값이 0 이상인 경우 Abort 감속의 S 자 곡선 시간을 msec 로 지정합니다. 값이 0 미만인 경우 msec³/모터 단위(일반 원시 카운트)로 Abort 저크의 최대값의 역수를 지정합니다.

Motor[x].AbortTs가 0 미만에서 최대 "저크"를 지정하는 경우에 **Motor[x].AbortTa**도 0 미만으로 최대 Abort 감속도를 지정해야 합니다. **Motor[x].AbortTs**가 0.0으로 설정되는 경우에 S 자 곡선 가속 부분은 없습니다. 그리고 조그 가속과 감속률은 일정해집니다.

Motor[x].AbortTa로 시간을 지정하고 **Motor[x].AbortTs**(S 자 곡선 시간)가 더 작은 시간을 지정하는 경우에 전체 가속 소요 시간은 **AbortTa**와 **AbortTs**의 합계와 같아집니다. 그러나 **Motor[x].AbortTa**로 시간을 지정하고 **Motor[x].AbortTs**에서 더 큰 시간을 지정하면 전체 Abort 감속 소요 시간은 **Motor[x].AbortTs**의 2 배가 됩니다. 따라서 **Motor[x].AbortTa**가 0으로 설정되는 경우 **Motor[x].AbortTs**로만 "순수한" S 자 곡선 형식으로 Abort 감속 시간을 제어합니다.

어떤 좌표계에서 온라인 명령 **a** 또는 **abort** 다이렉트 프로그램 명령이 내려지면 해당 좌표계의 모터는 Abort 됩니다. 또한 하드웨어 또는 소프트웨어 오버트래블 리미트 중 하나를 초과하는 경우 또는 "run-time error"가 해당 좌표계에서 발생하는 경우에 그 좌표계의 모터는 자동으로 Abort 됩니다.

예

미크론 모터 단위에서 50m/s³(약 5G/sec)의 Abort 저크 레이트를 설정하는 방법은 다음과 같습니다.

$$AbortTs = -\frac{s^3}{50m} * \frac{10^{-6}m}{motor\_unit} * \frac{10^9ms^3}{s^3} = -20\left(\frac{ms^3}{motor\_unit}\right)$$

Motor[x].AbsPhasePosForce 모터의 절대 위상 위치의 강제값

설명:	전류(commutation) 위치의 강제값
범위:	0.0 ..2047.999
단위:	1/2048 전류(commutation) 사이클
기본값:	0.0

Motor[x].AbsPhasePosForce에서는 위상 위치의 현재값을 "강제하기" 위해 저장할 수 있는 값을 제공합니다. 홀 센서 읽기 또는 고부하 모터에서의 위상 서치 동작과 같이 초기 위상 참조가 단지 근사인 경우에 이 변수는 일반적으로 사용됩니다. 보통 해당 방식은 기본 동작 기능을 얻기에는 충분합니다. 그러나 위상 참조에서의 나머지 오차는 최적의 모터 성능을 방해합니다. 이러한 경우에 엔코더 인덱스 펄스를 트리거로 하는 원점 서치 이동은 해당 모터를 전류(commutation) 사이클의 정확한 이미 알고 있는 위치로 하기에 충분한 기능입니다. 그 후 다음 명령을 실행할 수 있습니다(온라인 또는 프로그램 중 하나에서).

Motor[x].PhasePos = Motor[x].AbsPhasePosForce

보통 **Motor[x].AbsPhasePosForce**의 바람직한 값은 무부하 모터에서 "스테핑 모터" 위상 서치를 실행하고(무부하에서는 매우 정확) 다음으로 트리거로 엔코더 인덱스 펄스를 사용하는 원점 서치 동작을 수행하여 결정됩니다. 원점 위치에서의 **Motor[x].PhasePos**의 값을 **Motor[x].AbsPhasePosForce**에 복사하여 저장할 수 있습니다.



Motor[x].AbsPhasePosForce가 자동으로 사용되는 일은 없습니다. 사용자 어플리케이션 중에서 편리하게 저장된 값으로 단순하게 이용할 수 있습니다.

Motor[x].AbsPhasePosFormat 모터의 절대 위상 위치의 형식

설명:	전원 투입 시의 절대 전류(commutation) 위치의 데이터 형식
범위:	\$0..\$FFFFFFFF
단위:	바이트 필드
기본값:	\$0
기존 I 변수명:	lxx91

Motor[x].AbsPhasePosFormat은 절대 전류(commutation) 로터 각도 위치를 얻기 위해 **Motor[x].pAbsPhasePos**에 의해 지정된 주소의 데이터를 Power PMAC이 어떻게 번역할지 결정합니다. 이 데이터가 포맷된 후 **Motor[x].AbsPhasePosSf**에서 전류(commutation) 각도 단위(가능한 방향 반전을 포함)로 스케일링되고 또한 최종적인 전류(commutation)각을 얻기 위해 **Motor[x].AbsPhasePosOffset**의 오프셋이 가해집니다.

Motor[x].AbsPhasePosFormat 은 4 바이트 폭으로 구성되는 32 비트의 값이며 16 진수 형식 \$aabbccdd 로 간주됩니다. 그때 \$aa, \$bb, \$cc, \$dd 의 16 진수 문자 페어는 각각 4 바이트 표시합니다.

- \$dd 는 **Motor[x].pAbsPhasePos** 에 의해 주소 지정된(제 1) 레지스터 중에서 사용되는 시작 비트 번호를 지정합니다. \$00 에서 \$1F(0~31)의 값의 범위가 됩니다. 이 비트는 최종적인 값의 LSB 로 사용됩니다.
- \$cc 에서는 최종적인 값을 생성하기 위해 사용되는 전체 비트 수를 지정합니다. 이는 \$20(32)까지의 값의 범위가 됩니다. 위상 위치가 하나의 전류(commutation) 사이클 중에서 절대적인 필요가 있으며 Power PMAC 의 내부 전류(commutation) 사이클이 11 비트인 분해능(2048 분할)이라는 점을 기억하십시오. 홀 센서 형식이 지정되는 경우에 3 비트만 이 바이트 설정과 관계없이 사용됩니다. 그러나 명확히 하기 위해 이 바이트를 \$03 으로 설정하는 것을 권장합니다.
- \$bb 는 후속되는 레지스터의 데이터의 사용 방법을 지정합니다. 제 1 레지스터에서 \$cc 지정된 비트 수를 배치할 수 없는 경우에만 사용됩니다. \$00 부터 \$1F(0~31)의 \$bb 값에서는 다음 상위에 주소가 지정된 레지스터(**Motor[x].pAbsPhasePos** + 4)에서 사용되는 시작 비트 번호를 지정합니다. \$20(32)의 \$bb 의 값에는 "IOGATE" IC 에서 사용되는 바이트 폭의 형식을 지정합니다. 이 형식에서는 후속 레지스터(3 개까지)의 각각의 데이터는 제 1 레지스터(\$dd)와 동일한 시작 비트 번호가 사용됩니다. 위치 정보가 모두 하나의 레지스터로 배치되는 경우에 이 바이트는 사용되지 않고 보통 \$00 으로 설정됩니다.
- \$aa 는 Power PMAC 이 읽은 데이터를 어떻게 번역할지 지정합니다. 이는 다음 값과 같습니다.
 - o \$00: 바이너리값(변환은 필요 없음)으로 번역합니다.
 - o \$02: 그레이 코드로 번역하고 바이너리값으로 변환합니다.
 - o \$04: 120°의 신호 간격을 가진 3 상 디지털 홀 센서 데이터로 번역하고 바이너리값으로 변환합니다.
 - o \$05: 60°의 신호 간격을 가진 3 상 디지털 홀 센서 데이터로 번역하고 바이너리값으로 변환합니다.

전류(commutation) 위치 데이터가 주기적인 방법 중에서 사용되므로 서보 위치의 절대적인 형식과 같은 부호가 있거나 부호가 없는 형식을 지정할 필요가 없다는 점에 주의하십시오.

\$dd 바이트가 \$20(32)으로 설정되는 경우에 모터는 모터용 MACRO 슬레이브 노드의 MI920 파라미터의 쿼리를 통하여 MACRO 링 상의 해당 절대 위치를 얻도록 지정됩니다. (이 기능은 2012 년 3/4 분기에 출시된 V1.5 펌웨어의 새로운 기능입니다.) 이 경우에 \$cc 바이트는 MI920 파라미터의 통지되는 값에서 사용되는 비트 수를 지정합니다. 그리고 \$bb 바이트는 이 값에서 사용되는 시작 비트의 번호를 지정합니다. \$aa 바이트는 다른 데이터 소스와 동일하게 이 값이 어떻게 번역되는지 지정합니다.

모터의 절대 위상 위치를 MACRO 링에서 읽는 경우 **Motor[x].pEncStatus** 로 지정한 MACRO 노드에서 자동으로 읽습니다. 이 경우에 **Motor[x].pAbsPhasePos** 로 지정된 주소는 사용되지 않습니다. 그러나 **Motor[x].pAbsPhasePos** 는 절대 위치 읽기를 지정하는 0 이 아닌 값으로 설정해야 합니다. **Motor[x].pEncStatus** 와 동일한 주소로 설정하는 것을 권장합니다. (예를 들어 **Gate3[j].Macro[j][k].a** 또는 **Gate3[j].MacroInA[j][k].a**)

예

17 비트 로터리 엔코더에서 32 비트 레지스터의 비트 0 으로 시작하는 바이너리 형식 데이터가 사용되는 경우 **Motor[x].AbsPhasePosFormat** 은 \$00001100 으로 설정합니다. (aa = \$00 에서 바이너리값, bb = \$00 에서 확장 레지스터 없음, cc = \$11 에서 17 비트 데이터, dd = \$00 에서 시작 비트 0 을 설정합니다.)

13 비트의 로터리 엔코더에서 32 비트 레지스터의 비트 8 로 시작하는 그레이 코드 데이터가 사용되는 경우 **Motor[x].AbsPhasePosFormat** 은 \$02000D08 로 설정합니다. (aa = \$02 에서 그레이 코드, bb = \$00 에서 확장 레지스터 없음, cc = \$0D 에서 13 비트 데이터, dd = \$08 에서 시작 비트 8 을 설정합니다.)

ASIC 의 채널 레지스터의 비트 28-30 으로 읽는 120°의 3 상 디지털 홀 센서가 사용되는 경우 **Motor[x].AbsPhasePosFormat** 은 \$0400031C 로 설정합니다. (aa = \$04 에서 120°의 홀 형식, bb = \$00 에서 확장 레지스터 없음, cc = \$03 에서 3 비트 데이터, dd = \$1C 에서 시작 비트 28 을 설정합니다.)

ACC-14E 에서 바이트 폭의 레지스터를 통하여 읽는 24 비트의 바이너리 데이터가 사용되는 경우 **Motor[x].AbsPhasePosFormat** 은 \$00201808 로 설정합니다. (aa = \$00 에서 바이너리값, bb = \$20 에서 여러 개의 바이트 폭의 확장 레지스터, cc = \$18 에서 24 비트 데이터, dd = \$08 에서 시작 비트 8 을 설정합니다.)

17 비트의 절대 위치를 MACRO 링 상에서 읽고 회전된 위치의 비트 0 이 LSB 인 경우 **Motor[x].AbsPhasePosFormat** 은 \$00001120 으로 설정합니다. (aa = \$00 에서 바이너리값, bb = \$00 에서 확장 레지스터 없음, cc = \$11 에서 17 비트 데이터, dd = \$20 에서 MACRO 링의 쿼리를 설정합니다.)

Motor[x].AbsPhasePosOffset 모터의 절대 위상 위치 오프셋

설명: 전원 투입 시의 절대 전류(commutation) 위치 오프셋

범위: 부동 소수점

단위: 1/2048 전류(commutation) 사이클

기본값: 0.0

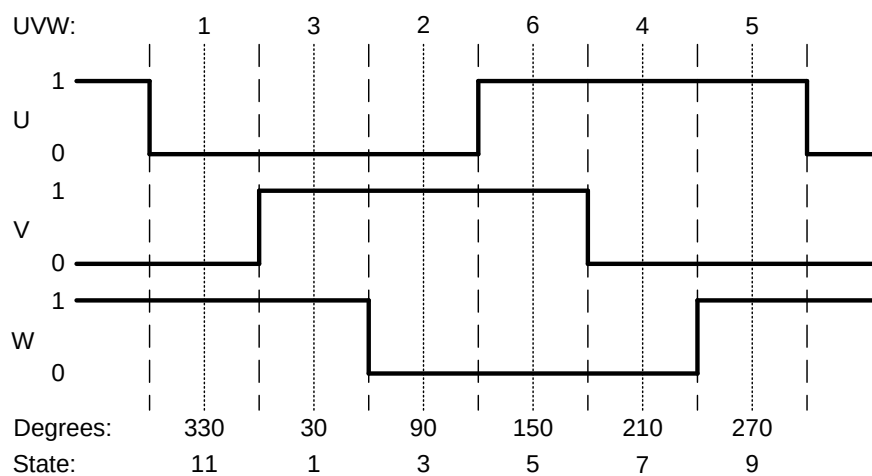
기존 I 변수명: lxx75

Motor[x].AbsPhasePosOffset 은 Power PMAC 에서 전류(commutation)되는 동기 모터(슬립 제로)가 전원 투입 시의 위상 위치를 확립하는 처리 중에 사용됩니다. 이 위치가 절대 위치 읽기에 의해 확립되거나(**Motor[x].PhaseFindingTime** = 0) 또는 위상 서치 동작(**Motor[x].PhaseFindingTime!** = 0)에 의해 확립되는지에 따라 이 요소는 달라집니다.

절대 위치 읽기인 경우에 하나의 전류(commutation) 사이클의 1/2048 단위에 **Motor[x].AbsPhasePosSf** 로 스케일링된 후에 **Motor[x].AbsPhasePosOffset** 에서는 절대 위치의 값에 더하는 양을 지정합니다. 따라서 이 요소는 사이클 하나의 1/2048 단위로 표시됩니다.

보통 전류(commutation) 사이클의 제로점으로 구동하는 "스테핑 모터" 위상 서치를 실행하고 이 점에서의 절대 위치 센서값을 읽어서 이 요소의 적정값이 설정됩니다. 이 값의 음수를 **Motor[x].AbsPosSf** 로 곱하고 이 결과를 **Motor[x].AbsPhasePosOffset** 에 씁니다.

홀 전류(commutation) 센서는 특별한 고려가 필요합니다. 다음 그림은 120 도 신호 간격에서의 홀 센서의 표준 구성을 나타냅니다. (60° 신호 간격인 경우에는 단순히 이 그림의 "V" 신호를 반전하십시오.)



U 가 low(0)이고 W 가 high(1)일 때 V 신호가 이동하면 Power PMAC 은 홀 사이클의 0 점으로 생각합니다. 이 그림(W 는 V 에 선행합니다. 그리고 V 는 U 에 선행합니다)에 표시된 "표준" 방향 검출에서 이 V 이동이 Power PMAC 의 전류(commutation) 사이클의 0 점인 경우 **Motor[x].AbsPhasePosOffset** 은 0.0 으로 설정됩니다.

일반적으로 이 V 이동은 Power PMAC 전류(commutation) 사이클의 제로 위치가 아닙니다. 따라서 이 이동이 어디가 되는지를 조사하는 것이 중요합니다. 다른 센서와 같이 이는 보통 위상 참조를 확립하기 위해 무부하 모터로 처음에 스텝핑 모터 위상 서치를 실행하여 수행됩니다. (가장 먼저 **Motor[x].AbsPhasePosSf** 가 올바르게 설정되었는지 확인하십시오.) 다음으로 **Motor[x].PhasePos** 의 전류(commutation) 각도와 U, V 와 W 상태를 모니터링하면서 이 V 이동이 발견될 때까지 모터를 천천히 이동시킵니다. (모터를 OFF 하고 손으로 이동시키면 대부분의 경우 수월하게 이동합니다.) 이 이동점에서의 **Motor[x].PhasePos** 의 값에 주의하십시오.

센서와 전류(commutation) 사이클의 방향 검출이 일치하는 경우(이 경우 **Motor[x].AbsPhasePosOffset** 은 양), **Motor[x].AbsPhasePosOffset** 에 이 값을 설정하십시오. 방향 검출이 일치하지 않는 경우(이 경우 **Motor[x].AbsPhasePosOffset** 은 음) **Motor[x].AbsPhasePosOffset** 에 이 값의 음수를 설정하십시오. 이 위상은 근사값일 뿐이며 나중에 수정되므로 여기서는 정확할 필요가 없다는 점에 주의하십시오.

위상 서치의 경우 **Motor[x].AbsPhasePosOffset**에서는 유효한 서치라고 판단하는 최소의 동작을 지정합니다. 이동이 이보다 작은 동작이 되는 경우에 Power PMAC은 서치가 실패했다고 생각합니다. 그 경우에는 모터의 **PhaseFound** Status 비트를 클리어하고 서치 종료 시 모터를 "kill"하며 어플리케이션으로 모터를 Enable 시키는 것을 허가하지 않습니다.

"스텝핑 모터" 위상 서치(**Motor[x].PhaseFindingTime**>255)인 경우에 1 번째와 2 번째 "스텝" 간의 이동이 평가됩니다. 이상적으로는 이것은 전류(commutation) 사이클의 $\pm 1/4$ 이어야 합니다. (~512 전류(commutation) 단위) (사이클의 $\sim 1/5$) 정도로 **Motor[x].AbsPhasePosOffset**을 설정하면 귀찮은 장애 리스크 없이 유효한 동작이 실행된다고 확약할 수 있습니다.

"4 가지 추측" 위상 서치(**Motor[x].PhaseFindingTime**<256)의 경우 "사인"과 "코사인" 추측의 이동량(전류(commutation) 단위가 아니라 모터 단위에서)의 제공의 합이 평가되어 **Motor[x].AbsPhasePosOffset**의 제공과 비교됩니다. 그 결과 **AbsPhasePosOffset**의 제공보다 작으면 서치가 실패했다고 판단합니다. 이 경우의 최적값은 서치 동작의 전형적인 크기에 좌우됩니다.

"사인" 추측의 동작의 크기는 **Motor[x].New[0].Vel**에 저장됩니다. "코사인" 추측의 동작의 크기는 **Motor[x].New[0].Accel**에 저장됩니다. 해당 크기는 사이클의 다양한 위치에서 시작하는 상당수의 서치 수로 평가되어야 합니다. 그리고 **Motor[x].AbsPhasePosOffset**은 해당 값의 제공합의 최소 유효 제공근보다 약간 작은 값으로 설정할 수 있습니다.

Motor[x].AbsPhasePosSf 모터의 절대 위상 위치의 스케일 팩터

설명: 전원 투입 시의 절대 전류(commutation) 위치의 스케일 팩터

범위: 부동 소수점

단위: 1/2048 전류(commutation) 사이클/소스 데이터

기본값: 1.0

Motor[x].AbsPhasePosSf는 **Motor[x].pAbsPhasePos**의 레지스터가 읽고 **Motor[x].AbsPhasePosFormat**에 따라 번역된 절대값 로터 각도 데이터를 1/2048 사이클의 전류(commutation) 단위로 스케일링하기 위해 사용되는 배율을 지정합니다. 따라서 그것은 소스 데이터의 LSB 당 전류(commutation) 단위가 됩니다. 이 스케일 팩터가 동작 중인 전류(commutation) 위치 데이터의 스케일 팩터와 동일할 필요가 없다는 점에 주의하십시오.

Power PMAC이 3 상 홀 전류(commutation) 센서를 디코드하면 결과의 위치값은 사이클마다 12 단위를 갖는다고 추측할 수 있습니다. - 6 개 상태와 6 개의 엣지

절대값 위상 포지션 센서의 업 다운 방향 검출이 전류(commutation) 사이클의 업 다운 방향 검출과 일치하지 않는 경우에 이 값은 음의 수로 설정합니다. 표준 120° 신호 위상차를 가지는 홀 센서인 경우에 전류(commutation) 사이클의 카운트 업 검출에서 W가 V에 선행하고 V가 U에 선행하는 경우에 방향 검출은 일치합니다. 양의 수를 설정합니다. 단, 카운트 업 검출에서 U가 V에 선행하고 V가 W에 선행하는 경우에 방향 검출은 일치하지 않습니다. 이 경우에 음의 수를 설정합니다. (드물게 60° 위상차 센서인 경우에 이 규칙은 반대가 됩니다.)

이와 다른 **Motor[x].PhasePosSf** 는 이동 중인 전류(commutation) 계산 중에 사용되고 다회전에 걸쳐 정확한 위치를 유지하기 위해 매우 정확해야 한다는 점에 주의하십시오. **Motor[x].AbsPhasePosSf** 는 한 번만 사용되므로 작은 반올림 오차를 허용할 수 있습니다.

예

17 비트 로터리 엔코더가 4 폴 모터로 전원 투입 시 사용되는 경우에 엔코더는 기계적인 1 회전당 131,072 카운트를 출력합니다. 2 전류(commutation) 사이클/회전에서 1 회전당 4,096 전류(commutation) 단위가 됩니다. **Motor[x].AbsPhasePosSf** 는 4096/131,072 또는 0.03125 로 설정됩니다.

120°의 신호 위상차에서 3 상 홀 센서가 전원 투입 시 사용되는 경우에 전류(commutation) 사이클의 카운트 업 검출에서 U 는 V 에 선행하고 V 는 W 에 선행합니다. 이 경우에 "역회전" 방향 검출이 있다고 생각할 수 있습니다. 홀 센서가 1 전류(commutation) 사이클(6 엣지와 6 스테이트)당 12 카운트를 출력한다고 생각할 수 있으므로 **Motor[x].AbsPhasePosSf** 는 -2048/12 또는 -170.667 로 설정됩니다.

Motor[x].AbsPosFormat 모터의 절대 위치 데이터 형식

설명: 전원 투입 시의 절대 위치 데이터 형식

범위: \$0..\$FFFFFFFF

단위: 바이트 필드

기본값: \$0

기존 I 변수명: lxx95

Motor[x].AbsPosFormat 은 모터의 절대 위치를 얻기 위해 **Motor[x].pAbsPos** 로 지정한 주소에서 Power PMAC 이 어떻게 데이터를 읽고 번역할지 결정합니다. 이 데이터가 포맷된 후에 **Motor[x].AbsPosSf** 에서 모터 단위로 스케일링되고 다음으로 **Motor[x].HomeOffset** 에서 오프셋되어 최종적인 모터의 절대 위치를 산출합니다.

Motor[x].AbsPosFormat 은 4 바이트 폭으로 구성되는 32 비트값입니다. 16 진수 형식 \$aabbccdd 라고 판단합니다. 그때 \$aa, \$bb, \$cc 와 \$dd 의 16 진수 문자 페어는 각각 4 바이트를 표시합니다.

- \$dd 는 **Motor[x].pAbsPos** 에 의해 주소 지정된(제 1) 레지스터에서 사용되는 시작 비트 번호를 지정합니다. 데이터를 직접 읽을 때 이 바이트는 \$00 에서 \$1F(0~31) 값의 범위가 됩니다. 이 비트는 최종적인 값의 LSB 로 사용됩니다.

\$dd 바이트가 \$20(32)으로 설정되는 경우에 모터는 모터용 MACRO 슬레이브 노드의 MI920 파라미터의 쿼리를 통하여 MACRO 링 상의 해당 절대 위치를 얻도록 지정됩니다. (이 기능은 2012 년 3/4 분기에 출시된 V1.5 펌웨어의 새로운 기능입니다.) 이 경우에 \$cc 바이트는 MI920 파라미터의 통지되는 값에서 사용되는 비트 수를 지정합니다. 그리고 \$bb 바이트는 이 값에서 사용되는 시작 비트의 번호를 지정합니다. \$aa 바이트는 다른 소스 데이터와 동일하게 이 값이 어떻게 번역되는지 지정합니다.

모터의 절대 위치를 MACRO 링에서 읽으면 **Motor[x].pEncStatus** 로 지정한 MACRO 노드에서 자동으로 읽습니다. 이 경우에 **Motor[x].pAbsPos** 로 지정된 주소는 사용되지 않습니다. 그러나 **Motor[x].pAbsPos** 는 절대 위치 읽기를 지정하는 0 이 아닌 값으로 설정해야 합니다. **Motor[x].pEncStatus** 와 동일한 주소로 설정하는 것을 권장합니다. (예를 들어 **Gate2[j].Macro[j][k].a** 또는 **Gate3[j].MacroInA[j][k].a**)

- \$cc 는 최종적인 값을 생성하기 위해 사용되는 전체 비트 수를 지정합니다. \$40(64) 이내의 값으로 할 수 있습니다. 유효로 하기 위해서는 모터의 전체 이동 구간에서 절대 위치가 보증되어야 한다는 점을 명심하십시오.
- \$bb 는 연속되는 레지스터에서의 데이터의 사용 방법을 지정합니다. 제 1 레지스터에서 \$cc 지정된 비트 수를 검출할 수 없는 경우에만 사용됩니다. \$00 에서 \$1F(0~31)의 \$bb 의 값에서는 다음 상위에 주소가 지정된 레지스터(**Motor[x].pAbsPos** + 4 의)에서 사용되는 시작 비트 번호를 지정합니다. \$20(32)의 \$bb 의 값에는 "IOGATE" IC 에서 사용되는 바이트 폭의 형식을 지정합니다. 이 형식에서는 이어지는 레지스터(3 개까지)의 각각의 데이터는 제 1 레지스터(\$dd)와 동일한 시작 비트 번호가 사용됩니다. 위치 정보가 모두 하나의 레지스터로 배치되는 경우에 이 바이트는 사용되지 않으며 보통 \$00 으로 설정됩니다.
- \$aa 는 Power PMAC 이 읽은 데이터를 어떻게 번역할지 지정합니다. 보통 다음 값 중 하나가 됩니다.
 - o \$00: 부호가 없는 바이너리값(변환은 필요 없음)으로 번역합니다.
 - o \$01: 부호가 있는 바이너리값(변환은 필요 없음)으로 번역합니다.
 - o \$02: 그레이 코드로 번역하고 부호가 없는 바이너리값으로 변환합니다.
 - o \$03: 그레이 코드로 번역하고 부호가 있는 바이너리값으로 변환합니다.

단, 8 비트 \$aa 값의 상위 5 비트 중 하나가 1 로 설정되는 경우에 **Motor[x].pAbsPos** 로 지정된 레지스터에서 읽은 데이터는 이어지는 처리가 실행되기 전에 좌측 시프트됩니다. (이 기능은 2012 년 3/4 분기에 출시된 V1.5 펌웨어의 새로운 기능입니다.) 이 기능의 목적은 일부 시리얼 엔코더 프로토콜에 발생하는 해당 레지스터와 후속 레지스터 간의 위치 데이터의 "간격"을 제거하는 것입니다. 중간 레지스터의 비트 31 에 위치 데이터의 최상위 비트를 시프트하여 실현합니다.

최초의 "a"로 표시되는 16 진수 자리의 상위 4 비트는 소스 레지스터에서 읽은 데이터를 좌측 시프트할 비트 수를 나타내는 0 에서 15 의 값을 지정합니다. 다음 비트 즉 2 번째의 16 진수 자리의 "8 의" 비트가 1 로 설정되면 소스 레지스터에서의 데이터는 추가 16 비트만큼 좌측 시프트됩니다. (현재 이 추가 시프트를 필요로 하는 프로토콜은 없습니다.) \$dd 로 지정된 "시작 비트" 번호는 이 초기 시프트 후 사용되는 최하위 비트 수를 나타낸다는 점에 주의하십시오.

예를 들어 일부 시리얼 엔코더 프로토콜은 **SerialEncDataA** 하위 17 비트의 "싱글턴" 위치 데이터와 **SerialEncDataB** 의 "멀티턴" 위치 데이터 16 비트를 더 제공합니다. 데이터가 32 비트 레지스터의 **Gate3[j].Chan[j].SerialEncDataA** 인 경우에 15 비트 좌측 시프트가 필요하며 최초의 "a"의 16 진법은 \$F 이고 \$dd 의 시작 비트값은 \$0F 가 됩니다. 데이터가 24 비트 레지스터의 **Acc84E[j].Chan[j].SerialEncDataA** 인 경우에 7 비트 좌측 시프트가 필요하며 최초의 "a"의 16 진법은 \$7 이고 \$dd 의 시작 비트값은 \$7 이 됩니다.

예

29 비트 멀티턴 로터리 엔코더에서는 32 비트 레지스터의 비트 0 에서 시작하는 바이너리 형식으로 데이터가 출력됩니다. 부호 없음으로 데이터를 번역하기 위해서는 **Motor[x].AbsPosFormat** 을 \$00001D00 으로 설정합니다. (aa = \$00 에서 부호가 없는 바이너리값, bb = \$00 에서 확장 레지스터 없음, cc = \$1D 에서 29 비트 데이터, dd = \$00 에서 시작 비트 0 을 설정합니다.)

27 비트 멀티턴 로터리 엔코더에서는 32 비트 레지스터의 비트 8 에서 시작하는 그레이 코드 데이터가 출력되고 다음 레지스터의 비트 0 에서 계속됩니다. 부호가 있는 데이터로 번역하기 위해서는 **Motor[x].AbsPosFormat** 을 \$03001B08 로 설정합니다. (aa = \$03 에서 부호가 있는 그레이 코드, bb = \$00 에서 확장 레지스터의 비트 0 에서 이어지고 cc = \$1B 에서 27 비트 데이터, dd = \$08 에서 시작 비트 8 을 설정합니다.)

33 비트 멀티턴 시리얼 엔코더에서는 32 비트 레지스터의 하위 비트인 17 비트 싱글턴 데이터와 다음 32 비트 레지스터의 하위 비트인 16 비트의 멀티턴 데이터를 출력합니다. 부호가 없는 바이너리값으로 데이터를 번역하기 위해서는 **Motor[x].AbsPosFormat** 을 \$F0002100 으로 설정합니다. (aa = \$F0 에서 제 1 레지스터의 15 비트 좌측 시프트, bb = \$00 에서 확장 레지스터의 시작 비트 0, cc = \$21 에서 33 비트 데이터, dd = \$0F 에서 시작 비트 15 를 설정합니다.)

ACC-14E 의 바이트 폭 레지스터를 통해 읽은 36 비트 패러럴 바이너리값 데이터가 사용됩니다. 부호가 있는 데이터로 번역하기 위해서는 **Motor[x].AbsPosFormat** 을 \$01202408 로 설정합니다. (aa = \$01 에서 부호가 있는 바이너리값, bb = \$20 에서 여러 개의 바이트 폭의 확장 레지스터, cc = \$24 에서 36 비트 데이터, dd = \$08 에서 시작 비트 8 을 설정합니다.)

ACC-24E2x 보드상의 23 비트 MLDT 타이머 레지스터는 32 비트 레지스터의 비트 8-30 에서 부호가 없는 위치를 출력합니다. **Motor[x].AbsPosFormat** 을 \$00001708 로 설정합니다. (aa = \$00 에서 부호가 없는 바이너리값, bb = \$00 에서 확장 레지스터 없음, cc = \$17 에서 23 비트 데이터, dd = \$08 에서 시작 비트 8 을 설정합니다.)

24 비트의 절대 위치를 MACRO 링 상에서 읽고 위치는 비트 0 을 LSB 로 한 부호가 있는 값으로 번역되기 위해서는 **Motor[x].AbsPosFormat** 을 \$01001820 으로 설정합니다. (aa = \$01 에서 부호가 있는 바이너리값, bb = \$00 에서 확장 레지스터 없음, cc = \$18 에서 24 비트 데이터, dd = \$20 에서 MACRO 쿼리를 설정합니다.)

Motor[x].AbsPosSf 모터의 절대 위치 스케일 팩터

설명: 전원 투입 시의 절대 위치 스케일 팩터

범위: 부동 소수점

단위: 모터 단위/소스 단위

기본값: 1.0

Motor[x].AbsPosSf 는 **Motor[x].pAbsPos** 로 지정된 레지스터에서 읽고 **Motor[x].AbsPosFormat** 에 따라 번역된 절대 위치 데이터를 사용자의 모터 단위로 스케일링하는 곱셈 스케일 팩터를 지정합니다. 따라서 단위는 모터 단위/소스 데이터의 LSB 가 됩니다. 이 스케일 팩터가 이동 중인 서보 위치 데이터의 스케일 팩터와 동일할 필요가 없다는 점에 주의하십시오.

절대 위치 검출기의 업 다운 방향 검출이 전류(commutation) 사이클의 업 다운 방향 검출과 일치하지 않는 경우에 이 값은 음의 수로 설정합니다.

이 곱셈이 실행된 후에 **Motor[x].HomeOffset** 의 값이 결과에 더해지고 결과의 합은 모터 단위의 모터 절대 위치를 확립하기 위해 사용됩니다.

Motor[x].AdcMask 모터의 전류 피드백 마스크

설명: 전류 피드백의 비트 패스 마스크 워드

범위: \$00000000..\$FFFFFFFF

단위: 비트 마스크

기본값: \$FFF00000(상위 12 비트 사용)

기존 I 변수명: lxx84

Motor[x].AdcMask 는 전류 루프식 중에서 실제 전류값으로 Power PMAC 에서 사용되는 것이 32 비트 전류 피드백 워드의 어느 비트인지 지정합니다. 클로즈 전류 루프가 이 모터에 대해 유효한 경우만 사용됩니다.

보통 전류의 피드백 데이터는 서보 IC 또는 MACRO IC 의 레지스터 최상위 비트에 출력되고 데이터는 시리얼 ADC 에서 시프트되어 왼쪽 자리맞춤됩니다. 가장 일반적인 분해능은 12 비트입니다. 그러나 다른 분해능도 가능합니다. 일부 애플은 동일한 워드의 더 하위 비트에서 상태와 이상 정보를 송신합니다. 이 경우에 전류 루프식에 해당 값을 마스크하는 것이 중요해집니다.

Motor[x].AdcMask 는 전류 루프식 중에서 사용되는 값을 생성하기 위해 논리적 연산을 통해 32 비트의 피드백 워드와 결합되는 32 비트의 마스크 워드를 지정합니다. 사용하는 모든 비트를 1 로 하고 사용되지 않은 모든 비트를 0 으로 합니다 데이터는 일반 왼쪽 자리맞춤이므로 **Motor[x].AdcMask** 는 1 에서 시작하고 0 에서 끝납니다. 보통 **Motor[x].AdcMask** 는 1 자리당 4 비트이고 합계 8 자리의 16 진법으로 표시됩니다.

*예*12 비트 ADC 에는: **Motor[x].AdcMask**=\$FFF0000014 비트 ADC 에는: **Motor[x].AdcMask**=\$FFFC000016 비트 ADC 에는: **Motor[x].AdcMask**=\$FFFF0000

Motor[x].AdvGain 모터의 전류(commutation) 위상 전진 게인

설명: 전류(commutation) 위상의 전진 게인

범위: 음이 아닌 부동 소수점

단위: 1/2048 전류(commutation) 사이클/(내측 루프의 모터 단위/16 서보 사이클)

기본값: 0.0

기존 I 변수명: lxx56

Motor[x].AdvGain 에서는 모터 각 상의 전기 회로의 지연 그리고/또는 모터 전류(commutation)에서의 계산과 전송 지연을 Power PMAC 에서 보정시킬 수 있게 합니다. 그 결과 고속도 성능을 향상시킵니다. 보정은 단순히 **Motor[x].FiltrVel2** 의 내측 루프의 필터 처리된 실제 모터 속도에 **Motor[x].AdvGain** 을 곱하게 하는 것입니다.

Power PMAC 이 모터 x 를 전류(commutation)하는(**Motor[x].PhaseCtrl** 비트 0 또는 비트 2 = 1) 경우에만 **Motor[x].AdvGain** 은 사용됩니다. 영구자석 브러시리스 모터와 같은 동기 모터(**Motor[x].SlipGain** = 0)의 전류(commutation)에만 사용됩니다. **Motor[x].AdvGain** 은 AC 유도 모터와 같은 비동기 모터(**Motor[x].SlipGain**>0)에는 0 으로 설정되어야 합니다.

Power PMAC 이 모터를 전류(commutation)하고 있지만 클로즈 전류 루프를 실행하고 있지 않으면(**Motor[x].pAdc** = 0) **Motor[x].AdvGain** 이 모터 권선 내 유도 지연을 보정하므로 보통 수천 RPM 에서 시동하는 성능을 향상시킵니다. Power PMAC 이 나아가 모터의 전류 루프를 실행 중인 경우(**Motor[x].pAdc**>0), DC 계자 프레임 전류 루프의 실행은 유도 지연을 보정하고 보정할 필요가 있는 근소한 계산 지연을 보정합니다. 이 파라미터의 사용은 폴 수가 낮은 서보 모터에서 10,000rpm 을 크게 초과하는 경우나 폴 수가 높은 스테핑 모터에서 약 1200rpm 이상이 아니면 그다지 의미가 없습니다.

이 파라미터는 고속으로 모터를 작동시키고 모터의 전류 흐름을 최소한으로 하는 설정을 조사하여 실험적으로 가장 좋은 값을 설정합니다.

Motor[x].AmpEnableBit 모터의 앰프 Enable 신호 비트 번호

설명:	pAmpEnable 레지스터의 앰프 Enable 신호의 비트 번호
범위:	0 ..31
단위:	비트 번호(리틀 엔디언)
기본값:	하드웨어에 따라서 자동으로 구성됩니다.

Motor[x].AmpEnableBit 는 **Motor[x].pAmpEnable** 로 지정되는 주소의 32 비트 레지스터의 어느 비트가 Power PMAC 에서 모터의 앰프 Enable 출력 제어에 사용되는지 결정합니다. 비트 번호는 "리틀 엔디언" 규약으로 지정하고 그때 비트 n 은 32 비트 워드의 2^n 의 값이 됩니다.

대다수 어플리케이션에서 이 값은 재초기화(\$\$\$\$** 명령)로 수행되는 하드웨어 자동 식별과 일부 설정에서 적절하게 지정됩니다. 이는 공장에서 또는 사용자에게 의한 시스템 어셈블리로 또는 그 후에 수행됩니다. 그리고 **Motor[x].EncType** 의 값이 Power PMAC 스크립트 환경을 통해 또는 온라인 명령, 버퍼 프로그램 명령 또는 전원 투입/리셋에서 저장된 값으로 설정될 때 **Motor[x].EncType** 의 값으로 지정된 하드웨어 인터페이스(예를 들어 PMAC2 스타일, PMAC3 스타일, MACRO) 타입에서 이 값을 표준값으로 설정됩니다.

ACC-24E2, ACC-24E2A 또는 ACC-24E2S 에서는 PMAC2 스타일 "DSPGATE1" 서보 IC 의 표준 앰프 Enable 출력 신호를 사용하기 위해 **Motor[x].AmpEnableBit** 는 22 로 설정됩니다. (IC 의 24 비트 워드의 비트 14 에 신호가 매핑되지만 이 워드는 32 비트 Power PMAC 버스의 상위 24 비트입니다. 따라서 이는 소프트웨어에서는 비트 22 로 표현된다는 점에 주의하십시오.)

ACC-24E3 또는 Power PMAC "Brick"에서는 PMAC3 스타일 "DSPGATE3" IC 의 표준 앰프 Enable 출력 신호를 사용하기 위해 **Motor[x].AmpEnableBit** 는 8 로 설정됩니다.

PMAC2 스타일 MACRO IC(ACC-5E 에서) 또는 PMAC3 스타일 IC(ACC-5E3 에서)를 사용하여 MACRO 링 프로토콜의 표준 앰프 Enable 비트를 사용하기 위해 **Motor[x].AmpEnableBit** 는 22 로 설정됩니다.

Motor[x].AmpFaultBit 모터의 앰프 폴트 신호의 비트 번호

설명:	pAmpFault 레지스터의 앰프 폴트 신호의 비트 번호
범위:	0 ..31
단위:	비트 번호(리틀 엔디언)
기본값:	하드웨어에 따라서 자동으로 구성됩니다.

Motor[x].AmpFaultBit 는 **Motor[x].pAmpFault** 로 지정되는 주소의 32 비트 레지스터의 어느 비트가 Power PMAC 에서 모터의 앰프 폴트 입력의 검출에 사용되는지 결정합니다. 비트 번호는 "리틀 엔디언" 규약으로 지정됩니다. 그때 비트 n 은 32 비트의 워드 중의 2^n 의 값입니다.

대다수 어플리케이션에서 이 값은 재초기화(\$\$\$*** 명령)로 수행되는 하드웨어 자동 식별과 일부 설정에서 적절하게 지정됩니다. 이는 공장에서 또는 사용자에게 의한 시스템 어셈블리로 또는 그 후에 수행됩니다. 그리고 **Motor[x].EncType**의 값이 Power PMAC 스크립트 환경을 통해 또는 온라인 명령, 버퍼 프로그램 명령 또는 전원 투입/리셋에서 저장된 값으로 설정될 때 **Motor[x].EncType**의 값으로 지정된 하드웨어 인터페이스(예를 들어 PMAC2 스타일, PMAC3 스타일, MACRO) 타입에서 이 값은 표준값으로 설정됩니다.

ACC-24E2, ACC-24E2A 또는 ACC-24E2S에서는 PMAC2 스타일 "DSPGATE1" 서보 IC의 표준 앰프 폴트 입력 신호를 사용하기 위해 **Motor[x].AmpFaultBit**는 23으로 설정됩니다. (IC의 24비트 워드의 비트 15에 신호가 매핑되지만 이 워드는 32비트 Power PMAC 버스의 상위 24비트입니다. 따라서 이는 소프트웨어에서는 비트 23으로 표현된다는 점에 주의하십시오.)

ACC-24E3 또는 Power PMAC "Brick"에서는 PMAC3 스타일 "DSPGATE3" IC의 표준 앰프 폴트 입력 신호를 사용하기 위해 **Motor[x].AmpFaultBit**는 7로 설정됩니다.

PMAC2 스타일 MACRO IC(ACC-5E에서) 또는 PMAC3 스타일 IC(ACC-5E3에서)를 사용하여 MACRO 링 프로토콜의 표준 앰프 폴트 비트를 사용하기 위해 **Motor[x].AmpFaultBit**는 23으로 설정됩니다.

Motor[x].AmpFaultLevel 모터의 앰프 이상 논리 상태

설명: 앰프 이상의 논리 상태

범위: 0 ..3

단위: 비트 필드

기본값: 1

Motor[x].AmpFaultLevel은 앰프 이상 에러의 요인이 되는 앰프 이상 입력 비트의 이상 상태와 이상 상태 검출에 필요한 수를 지정합니다.

비트 0(값 1)에서는 이상이라고 판단되는 앰프 이상 입력 비트의 바이너리값을 지정합니다. 그 값이 0으로 설정되는 경우에 0의 값이 이상으로 판단됩니다. 그 값이 1로 설정되는 경우에 1의 값이 이상으로 판단됩니다. 앰프 이상 비트는 **Motor[x].AmpFaultBit**로 지정한 비트 번호로 **Motor[x].pAmpFault**에 지정된 레지스터에서 읽습니다. **Motor[x].pAmpFault**가 0이고 앰프 이상 기능을 무효로 설정한 경우에 **Motor[x].AmpFaultLevel**의 설정은 관계가 없어집니다.

0 또는 1의 값을 생성하는 전압 레벨은 인터페이스 회로의 특성에 좌우됩니다. 많은 Power PMAC 앰프 이상 회로는 양방향 포토 커플러를 사용합니다. 해당 회로에서 어느 한 방향의 포토 커플러를 통해 한 방향으로 흐르는 전류는 0의 값을 생성시킵니다.

비트 1(값 2)은 앰프 이상 비트가 지정된 이상 상태임을 Power PMAC이 검출해야 하는 연속 스캔 수를 지정합니다. 그리고 검출되면 모터를 "kill"하는 앰프 이상 트립을 발생시킵니다. 0으로 설정되는 경우에 싱글 스캔에서 이상 상태를 검출하여 트립을 발생시킵니다. 1로 설정되는 경우에 2개의 연속 스캔(연속한 실시간 인터럽트 주기)에서 트립을 발생시키는 이상 상태를 검출해야 합니다. 싱글 검출에서는 **Motor[x].AmpWarn Status** 비트가 설정되며 래치됩니다.

Motor[x].AuxFaultBit 모터의 보조 이상 비트 번호

설명:	pAuxFault 레지스터에서의 센서 소실 플래그의 비트 번호
범위:	0 ..31
단위:	비트 번호(리틀 엔디언)
기본값:	0

Motor[x].AuxpFaultBit 는 **Motor[x].pAuxFault** 로 지정되는 주소의 32 비트 레지스터의 어느 비트가 모터의 보조 이상 검출에 사용되는지 결정합니다. 비트 번호는 "리틀 엔디언" 규약으로 지정됩니다. 그때 비트 n 은 32 비트 워드의 2^n 의 값이 됩니다. 이 보조 이상 입력 비트의 극성은 **Motor[x].AuxFaultLevel** 로 설정됩니다.

"T" 플래그 입력 처리 회로가 있는 Power Brick PMAC3 스타일 인터페이스 보드에 연결된 모터 서멀 센서에는 T 플래그 입력 비트가 레지스터의 비트 15 에서 검출되도록 **Motor[x].AuxFaultBit** 는 15 로 설정합니다.

"배타적 논리합"의 AB 상 소실 검출 회로를 가지는 ACC-24E2x PMAC2 스타일 축 인터페이스 보드에 연결된 AB 상 엔코더에는 **Gate1[j].Chan[j].EncLossN Status** 비트가 레지스터의 비트 13 에서 검출되도록 **Motor[x].AuxFaultBit** 는 13 으로 설정합니다.

"배타적 논리합"의 AB 상 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 AB 상 엔코더에는 **Gate3[j].Chan[j].LossStatus** 비트가 레지스터의 비트 28 에서 검출되도록 **Motor[x].AuxFaultBit** 는 28 로 설정합니다.

"제공함" 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 아날로그 사인파 엔코더 또는 리졸버에는 **Gate3[j].Chan[j].SosError** 비트가 레지스터의 비트 31 에서 검출되도록 **Motor[x].AuxFaultBit** 는 31 로 설정합니다.

일부 프로토콜에 좌우되는 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 시리얼 엔코더에는 채널의 "타임아웃" 에러 비트가 **Gate3[j].Chan[j].SerialEncDataB** 레지스터의 비트 31 에서 검출되도록 **Motor[x].AuxFaultBit** 는 일반 31 로 설정합니다. 그러나 프로토콜에 따라 다를 가능성도 있습니다.

Motor[x].AuxFaultLevel 모터의 보조 이상 논리 레벨

설명:	보조 이상의 논리 상태
범위:	0 ..1
단위:	논리값
기본값:	1

Motor[x].AuxFaultLevel 은 보조 이상 에러의 요인이 되는 보조 이상 입력 비트의 소실 상태를 지정합니다. 이 상태의 필요한 검출 수가 **Motor[x].AuxFaultLimit** 로 지정됩니다.

Motor[x].AuxFaultLevel 이 0 으로 설정되는 경우에 0 의 값이 이상으로 판단됩니다. 그 값이 1 로 설정되는 경우에 1 의 값이 이상으로 판단됩니다. 보조 이상 비트는 **Motor[x].AuxFaultBit** 로 지정한 비트 번호이며 **Motor[x].pAuxFault** 에서 지정한 레지스터에서 읽습니다. **Motor[x].pAuxFault** 가 0 으로 설정되고 보조 이상 검출 기능이 무효인 경우에 **Motor[x].AuxFaultLevel** 설정은 관계가 없어집니다.

"T" 플래그 입력 처리 회로를 가지는 Power Brick PMAC3 스타일 인터페이스 보드에 연결된 모터 서멀 센서에는 **Motor[x].AuxFaultLevel** 을 1 로 설정합니다.

"배타적 논리합"의 AB 상 소실 검출 회로를 가지는 ACC-24E2x PMAC2 스타일 축 인터페이스 보드에 연결된 AB 상 엔코더에는 **Motor[x].AuxFaultLevel** 을 0 으로 설정합니다.

"배타적 논리합"의 AB 상 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 AB 상 엔코더에는 **Motor[x].AuxFaultLevel** 을 1 로 설정합니다.

"제곱합" 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 아날로그 사인파 엔코더 또는 리졸버에는 **Motor[x].AuxFaultLevel** 을 1 로 설정합니다.

일부 프로토콜에 좌우되는 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드 또는 Power Brick 제어 보드에 연결된 시리얼 엔코더에는 **Motor[x].AuxFaultLevel** 을 1 로 설정합니다.

Motor[x].AuxFaultLimit 모터의 보조 이상 검출 제한

설명: 보조 이상 검출의 최대 누적 수

범위: 0 ..255

단위: 스캔

기본값: 0

Motor[x].AuxFaultLimit 는 보조 이상 에러에서 트립하지 않고 "이상" 상태를 Power PMAC 이 검출할 수 있는 누적 스캔의 최대수를 지정합니다. 이상 상태의 스캔 누적 수가 이 값을 초과하는 경우에 Power PMAC 은 해당 모터를 자동으로 "kill"하고 **Motor[x].FaultMode** 의 설정에 따라 동일한 좌표계의 다른 모터를 kill 또는 Abort 합니다. Power PMAC 은 지정된 엔코더 소실 비트의 상태를 실시간 인터럽트 주기마다 확인합니다.

보조 이상 검출 회로의 대부분은 일시적인 전기적 노이즈 또는 이 원인에 의한 이상을 검출하는 경우입니다. 소실 검출에 여러 번 연속 스캔을 필요로 한다는 것은 실제 소실을 확인할 수 있게 하며 오검출에 의한 트립을 제거 할 수 있다는 점에서 유익합니다.

보조 이상 비트의 레지스터는 **Motor[x].pAuxFault** 로 지정됩니다. 이 레지스터 내의 해당 비트는 **Motor[x].AuxFaultBit** 로 지정됩니다. 이 비트의 소실 상태는 **Motor[x].AuxFaultLevel** 로 지정됩니다. **Motor[x].pAuxFault** 를 0 으로 설정하여 보조 이상의 검출 기능을 무효로 한 경우 **Motor[x].AuxFaultLimit** 의 설정은 관계가 없어집니다. "이상" 상태에서 검출된 지정 비트의 스캔 누적 수는 상태 요소 **Motor[x].AuxFaultCount** 로 모니터링됩니다. 이 비트가 이상 상태로 검출될 때마다 이 요소는 더해지고 이상 상태가 검출되지 않을 때마다 감소됩니다. (0 보다 큰 경우)

Motor[x].BIHysteresis 모터의 백래시 히스테리시스

설명: 백래시의 히스테리시스 크기

범위: 부동 소수점

단위: 모터 단위

기본값: 0.0

기존 I 변수명: lxx87

Motor[x].BIHysteresis 는 Power PMAC 이 새로운 방향으로 백래시 보정을 도입하기 시작하기 전에 존재해야 하는 모터 명령 위치에서의 방향 반전의 크기를 지정합니다. 이 변수의 목적은 매우 작은 방향 반전(예를 들어 마스터 엔코더의 디더링에서)이 백래시 보정을 즉시 시작하지 않도록 보증하는 것입니다.

Motor[x].BISize 모터의 백래시 크기

설명: 백래시 크기

범위: 부동 소수점

단위: 모터 단위

기본값: 0.0

기존 I 변수명: lxx86

Motor[x].BISize 는 Power PMAC 이 모터 기어의 백래시 보정에 사용하는 보정 크기를 지정합니다. 명령 방향 반전 시 이 거리를 원시 측정 위치에 더하거나 뺍니다. 실제 위치가 조화되는 경우 이 오프셋은 표시되지 않습니다. 백래시를 더하거나 빼는("전송") 레이트는 **Motor[x].BISlewRate** 로 결정됩니다. **Motor[x].BISlewRate** 가 양의 값(기본값은 0.0)으로 설정되지 않으면 백래시 보정이 실제로 수행되지 않는다는 점에 주의하십시오.

Motor[x].BISlewRate 모터의 백래시 슬루 레이트

설명:	백래시의 전송 레이트
범위:	부동 소수점
단위:	모터 단위/실시간 인터럽트
기본값:	0.0
기준 I 변수명:	lxx85

Motor[x].BISlewRate 는 어떻게 빨리 백래시가 방향 반전 시 "전송"되는지를 지정합니다. 그 백래시의 크기는 **Motor[x].BISize** 로 결정됩니다. **Motor[x].BIHysteresis** 보다 큰 명령 방향의 반전이 실시간 인터럽트 주기로 검출되면 Power PMAC 은 모든 백래시값이 보정될 때까지 **Motor[x].BISlewRate** 와 같은 값으로 더하거나 뺍니다.

Motor[x].BISlewRate 는 보통 동적인 문제를 일으키지 않는 가능한 한 높은 값으로 대화식 그리고 실험적으로 설정됩니다. 기본값 0.0 은 실질적으로 백래시 보정을 무효로 만듭니다.

개별 모터 상태의 갱신이 실시간 인터럽트마다 실행되지 않도록 저장 설정 요소 **Sys.MotorsPerRtInt** 가 0 이외의 값으로 설정되는 경우(2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능), **Motor[x].BISlewRate** 의 스케일링은 0 인 기본값으로 설정되는 경우 또는 파라미터 **Sys.MotorsPerRtInt** 가 없었던 이전 펌웨어 버전의 경우와 다릅니다.

Sys.MotorsPerRtInt>0 인 경우에 **Motor[x].BISlewRate** 를 수정하기 위해 필요한 시간 확장 계수 N 은 다음 식으로 구합니다.

$$N = \text{ceil}\left(\frac{\text{Sys.ServoMotors} + 1}{\text{Sys.MotorsPerRtInt}}\right)$$

"ceil"은 다음 정수에 올림 처리하는 것(필요한 경우)을 나타내는 "CEILING" 함수입니다. 그리고 상태 요소 **Sys.ServoMotors** 는 유효한 모터의 최대 번호입니다. 실시간 인터럽트마다 실행되는 경우와 비교하여 **Motor[x].BISlewRate** 의 값은 N 으로 곱해야 합니다.

Motor[x].BrakeOffDelay 모터의 브레이크 개방 지연 시간

설명:	모터 Enable 후의 브레이크 개방 지연 시간
범위:	음이 아닌 부동 소수점
단위:	msec
기본값:	0.0

Motor[x].BrakeOffDelay 는 모터가 Enable 된 후에 브레이크가 자동으로 개방될 때까지의 지연 시간을 지정합니다. **Motor[x].pBrakeOut** 이 0 이 아닌 값으로 설정되고 해당 모터에 대해 자동 브레이크 제어 기능이 유효가 되는 경우에만 사용됩니다.

자동 브레이크 제어 기능이 유효인 모터가 **j/**, **outn** 또는 **enable** 명령으로 **Enable** 되면 브레이크는 **BrakeOffDelay** msec 후에 지정된 브레이크 제어 출력 비트를 1로 설정하여 개방됩니다. 이 지연은 브레이크가 개방되기 전에 모터의 완전한 제어를 달성하기 위해 시스템 시간을 부여하여 제어가 확립되기 전의 순간 동작을 방지합니다. (보통 수직 낙하)

자동 브레이크 제어 기능이 유효인 경우에 사용자는 **j+** 또는 **hm** 과 같은 실제 모터 이동 명령에서 **Disable** 상태의 모터를 **Enable** 로 하지 마십시오.

개별 모터 상태의 갱신이 실시간 인터럽트마다 실행되지 않도록 저장 설정 요소 **Sys.MotorsPerRtInt** 가 0 이외의 값으로 설정되는 경우(2014년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능), **Motor[x].BrakeOffDelay** 의 스케일링은 0인 기본값으로 설정되는 경우 또는 파라미터 **Sys.MotorsPerRtInt** 가 없었던 이전 펌웨어 버전의 경우와 다릅니다.

Sys.MotorsPerRtInt>0 인 경우에 **Motor[x].BrakeOffDelay** 를 수정하기 위해 필요한 시간 확장 계수 N 은 다음 식으로 구합니다.

$$N = \text{ceil}\left(\frac{\text{Sys.ServoMotors} + 1}{\text{Sys.MotorsPerRtInt}}\right)$$

"*ceil*"은 다음 정수에 올림 처리하는 것(필요한 경우)을 나타내는 "CEILING" 함수입니다. 그리고 상태 요소 **Sys.ServoMotors** 는 유효한 모터의 최대 번호입니다. 실시간 인터럽트마다 실행되는 경우와 비교하여 **Motor[x].BrakeOffDelay** 의 값은 N 으로 나뉘어야 합니다.

Motor[x].BrakeOnDelay 모터의 브레이크 구동 지연 시간

설명: 브레이크 구동 후 모터 **Disable** 의 지연 시간

범위: 음이 아닌 부동 소수점

단위: msec

기본값: 0.0

Motor[x].BrakeOnDelay 는 브레이크가 "지연 **Disable**" 명령으로 구동된 후에 모터가 **Disable** 될 때까지의 지연 시간을 지정합니다. **Motor[x].pBrakeOut** 이 0이 아닌 값으로 설정되고 해당 모터에 대해 자동 브레이크 제어 기능이 유효가 되는 경우에만 사용됩니다.

자동 브레이크 제어 기능을 가지는 모터가 클로즈드 루프 명령 속도 제로 상태에서 유효가 되며 전용 "지연 **Disable**" 명령으로 **Disable** 될 때 브레이크는 지정된 브레이크 제어 출력 비트가 0으로 설정되는 즉시 구동됩니다. 단, 모터 제어는 **BrakeOnDelay** msec 후까지 **Disable** 되지 않습니다. 이 지연은 모터 제어가 **Disable** 되기 전에 완전히 구동하는 브레이크 하드웨어 시간을 부여하여 브레이크가 완전히 구동되기 전에 순간 동작을 방지합니다. (보통 수직 낙하)

이 지연은 모터가 클로즈드 루프 명령 속도 0 상태인 경우에 모터 명령 **dkill** 과 좌표계 명령 **ddisable** 로 사용됩니다. 해당 명령은 온라인 형식(예를 들어 **#1dkill**, **&2ddisable**) 또는 버퍼형 프로그램 형식(예를 들어 **dkill1**, **ddisable2**) 중에서 사용할 수 있습니다.

표준 Disable 명령(**k**, **kill**, **disable**)이 사용되는 경우 또는 모터가 앰프 이상 또는 편차 이상 에러와 같은 이상에 의해 자동으로 Disable 되는 경우에 또는 모터가 클로즈드 루프 명령 속도 0 상태가 아닌 경우에 모터는 즉시 Disable 됩니다. 그리고 동일한 타이밍에 브레이크 출력이 구동됩니다. 지연은 없습니다.

개별 모터 상태의 갱신이 실시간 인터럽트마다 실행되지 않도록 저장 설정 요소 **Sys.MotorsPerRtInt** 가 0 이외의 값으로 설정되는 경우(2014년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능), **Motor[x].BrakeOnDelay**의 스케일링은 기본값 0으로 설정되는 경우 또는 파라미터 **Sys.MotorsPerRtInt**가 없었던 이전 펌웨어 버전의 경우와 다릅니다.

Sys.MotorsPerRtInt>0 인 경우에 **Motor[x].BrakeOnDelay**를 수정하기 위해 필요한 시간 확장 계수 N 은 다음 식으로 구합니다.

$$N = \text{ceil}\left(\frac{\text{Sys.ServoMotors} + 1}{\text{Sys.MotorsPerRtInt}}\right)$$

"ceil"은 다음 정수에 올림 처리하는 것(필요한 경우)을 나타내는 "CEILING" 함수입니다. 그리고 상태 요소 **Sys.ServoMotors**는 유효한 모터의 최대 번호입니다. 실시간 인터럽트마다 실행되는 경우와 비교하여 **Motor[x].BrakeOnDelay**의 값은 N 으로 나뉘야 합니다.

Motor[x].BrakeOutBit 모터의 브레이크 출력 신호 비트 번호

설명: **pBrakeOut** 레지스터의 브레이크 출력 신호의 비트 번호

범위: 0 ..31

단위: 비트 번호(리틀 엔디언)

기본값: 9

Motor[x].BrakeOutBit는 **Motor[x].pBrakeOut**으로 지정되는 주소의 32 비트 레지스터의 어느 비트가 Power PMAC에서 모터의 브레이크 제어 출력에 사용되는지 결정합니다. 비트 번호는 "리틀 엔디언" 규약으로 지정됩니다. 그때 비트 n 은 32 비트 워드 중의 2^n 의 값이 됩니다.

Power PMAC Brick에서는 PMAC3 스타일 "DSPGATE3" IC의 출력 플러그 "B"를 사용하기 위해 **Motor[x].BrakeOutBit**는 9로 설정됩니다.

ACC-65E, ACC-68E 또는 ACC-11E와 같은 UMAC 입출력 보드에서 범용 출력을 사용하기 위해 **Motor[x].BrakeOut**은 사용되는 특정 출력에 따라 8~15의 값으로 설정됩니다.

Motor[x].CaptControl 모터의 포착 위치 제어

설명: 포착 위치 처리의 모터 풀 워드 요소

범위: \$00000000..\$FFFFFFFF

단위: 비트 필드

기본값: \$00000000

Motor[x].CaptControl은 원점 서치 동작과 같은 트리거 포함 동작에서 포착 플래그와 포착 위치 처리에 대한 설정 요소로 구성되는 풀 워드 요소입니다.

Motor[x].CaptControl은 다음과 같은 부분 워드 요소로 구성됩니다.

성분	비트	기능
CaptFlagBit	31 - 24	포착 플래그의 비트 번호
CaptPosRightShift	23 - 16	포착 위치를 우측 시프트할 비트 수
CaptPosLeftShift	15 - 08	포착 위치를 좌측 시프트할 비트 수
--	07 - 01	(추후 사용하기 위해 예약)
CaptPosRound	00	포착 위치의 1/2 카운트 오프셋 제어

사용자는 요소의 설정과 그 설정 확인을 위해 보통 스크립트 환경에서 해당 부분 워드 요소에 액세스합니다. Power PMAC은 효율화를 위해서 그 풀 워드 요소로 저장 및 복원합니다.

풀 워드 요소인 값이 스크립트 환경 속에서 참조될 때 예를 들어 그 부분 워드 요소값이 10 진법으로 통지되더라도 Power PMAC은 16 진법 형식으로 값을 반환합니다.

그 요소 기능의 자세한 설명에 관해서는 각 부분 워드 요소의 항목을 참조하십시오.

Motor[x].CaptFlagBit 모터의 포착 플래그 비트 번호

설명: pCaptFlag 레지스터에서의 포착 플래그의 비트 번호

범위: 0 ..31

단위: 비트 번호(리틀 엔디언)

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

Motor[x].CaptFlagBit는 **Motor[x].pCaptFlag**로 지정되는 주소의 32 비트 레지스터의 어느 비트가 Power PMAC에서 모터의 포착 트리거 플래그를 검출하기 위해 사용되는지 결정합니다. 비트 번호는 "리틀 엔디언" 규약으로 지정됩니다. 그때 비트 n 은 32 비트 워드 중의 2^n 의 값이 됩니다.

대다수 어플리케이션에서 이 값은 재초기화(\$\$\$** 명령)로 수행되는 하드웨어 자동 식별과 일부 설정에서 적절하게 지정됩니다. 이는 공장에서 또는 사용자에게 의한 시스템 어셈블리로 또는 그 후에 수행됩니다. 그리고 **Motor[x].EncType**의 값이 Power PMAC 스크립트 환경을 통해 또는 온라인 명령, 버퍼 프로그램 명령 또는 전원 투입/리셋에서 저장된 값으로 설정될 때 **Motor[x].EncType**의 값으로 지정된 하드웨어 인터페이스(예를 들어 PMAC2 스타일, PMAC3 스타일, MACRO) 타입에서 이 값은 표준값으로 설정됩니다.

ACC-24E2, ACC-24E2A 또는 ACC-24E2S에서는 PMAC2 스타일 "DSPGATE1" 서보 IC의 표준 포착 트리거 플래그를 사용하기 위해 **Motor[x].CaptFlagBit**는 19로 설정되어야 합니다. (IC의 24비트 워드의 비트 11에 해당 신호가 매핑되지만 이 워드는 32비트 Power PMAC 버스의 상위 24비트가 됩니다. 따라서 소프트웨어에서 해당 비트 19로 표현된다는 점에 주의하십시오.)

ACC-24E3 또는 Power PMAC "Brick"에서는 PMAC3 스타일 "DSPGATE3" IC의 표준 포착 트리거 플래그를 사용하기 위해 **Motor[x].CaptFlagBit**는 20으로 설정되어야 합니다.

PMAC2 스타일 MACRO IC(ACC-5E에서) 또는 PMAC3 스타일 IC(ACC-5E3에서)와 관계없이 MACRO 링 프로토콜의 표준 포착 트리거 플래그를 사용하기 위해서는 **Motor[x].CaptFlagBit**는 19로 설정되어야 합니다.

Motor[x].CaptPosLeftShift 모터의 포착 위치 좌측 시프트 수

설명: 포착 위치 데이터를 좌측 시프트할 비트 수

범위: 0 ..31

단위: 비트

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

Motor[x].CaptPosLeftShift에서는 **Motor[x].pCaptPos**로 주소 지정된 32비트 레지스터에서 하드웨어 포착 위치를 **Motor[x].CaptPosRightShift**로 지정한 값으로 최초로 우측 시프트한 후에 좌측 시프트할 비트 수를 지정합니다. 이 조작의 목적은 변환 테이블 등록에 의한 **Motor[x].pEnc**로 지정된 동작 중의 위치 루프 피드백 데이터와 동일한 분해능을 가지는 값을 생성하는 것입니다.

대다수 어플리케이션에서 이 값은 재초기화(\$\$\$** 명령)로 수행되는 하드웨어 자동 식별과 일부 설정에서 적절하게 지정됩니다. 이는 공장에서 또는 사용자에게 의한 시스템 어셈블리로 또는 그 후에 수행됩니다. 그리고 **Motor[x].EncType**의 값이 Power PMAC 스크립트 환경을 통해 또는 온라인 명령, 버퍼 프로그램 명령 또는 전원 투입/리셋에서 저장된 값으로 설정될 때 **Motor[x].EncType**의 값으로 지정된 하드웨어 인터페이스(예를 들어 PMAC2 스타일, PMAC3 스타일, MACRO) 타입에서 이 값은 표준값으로 설정됩니다.

ACC-24E2, ACC-24E2A 또는 ACC-24E2S에서 서보 루프 위치 피드백이 1/T 보간을 가지는 디지털 AB 상 엔코더일 때 PMAC2 스타일 "DSPGATE1" 서보 IC의 표준 24비트 정수 카운트 포착 위치 레지스터를 사용하기 위해 **Motor[x].CaptPosLeftShift**는 9로 설정되어야 합니다. 왜냐하면 서보 피드백은 9비트의 소수점 카운트 확장을 가지기 때문입니다.

ACC-51E 에서 서보 루프 위치 피드백이 아크탄젠트 확장으로 사인파 엔코더일 때 PMAC2 스타일 "DSPGATE1"서보 IC 의 표준 24 비트의 정수 카운트 포착 위치 레지스터를 사용하기 위해서 **Motor[x].CaptPosLeftShift** 는 10 으로 설정되어야 합니다. 왜냐하면 서보 피드백은 10 비트의 소수점 카운트 확장을 가지기 때문입니다.

소수점 카운트 확장이 없는 IC 의 24 비트 정수 카운트 "위상 포착" 레지스터를 사용하고 서보 루프 위치 피드백이 위상 인터럽트(**Motor[x].PhaseCtrl=8**)로 처리되는 경우에 PMAC2 스타일 "DSPGATE1" 서보 IC 의 표준 24 비트 정수 카운트 포착 위치 레지스터를 사용하기 위해 **Motor[x].CaptPosLeftShift** 는 8 로 설정되어야 합니다. 왜냐하면 서보 피드백의 1"카운트"가 이 모드에서의 32 비트 Power PMAC 버스의 비트 8 로 검출되기 때문입니다.

ACC-24E3 또는 Power PMAC "Brick"에서 서보 피드백이 1/T 확장에서 AB 상 엔코더일 때 PMAC3 스타일 "DSPGATE3" IC 의 32 비트 포착 위치 레지스터의 24 비트 정수 카운트 부분만 사용하기 위해 **Motor[x].CaptPosLeftShift** 는 8 로 설정되어야 합니다. 왜냐하면 서보 피드백은 이 모드에서 8 비트의 소수점 카운트 확장을 가지기 때문입니다.

ACC-24E3 또는 Power PMAC "Brick"에서 서보 피드백이 아크탄젠트 확장에서 사인파 엔코더일 때 PMAC3 스타일 'DSPGATE3' IC 의 32 비트 포착 위치 레지스터의 24 비트 정수 카운트 부분만 사용하기 위해 **Motor[x].CaptPosLeftShift** 는 12 로 설정되어야 합니다. 왜냐하면 서보 피드백은 이 모드에서 12 비트의 소수점 카운트 확장을 가지기 때문입니다.

서보 피드백이 1/T 확장에서 AB 상 엔코더일 때 하위 8 비트에 1/T 의 소수점 카운트 추정을 포함하는 PMAC3 스타일 "DSPGATE3" IC 의 32 비트 포착 위치 레지스터 전체를 사용하기 위해 **Motor[x].CaptPosLeftShift** 는 0 으로 설정되어야 합니다. 왜냐하면 서보 피드백은 이 모드에서 8 비트의 소수점 카운트 확장을 가지기 때문입니다.

서보 피드백이 아크탄젠트 확장에서 사인파 엔코더일 때 하위 8 비트에 1/T 의 소수점 카운트 추정을 포함하는 PMAC3 스타일 "DSPGATE3" IC 의 32 비트 포착 위치 레지스터 전체를 사용하기 위해 **Motor[x].CaptPosLeftShift** 는 4 로 설정되어야 합니다. 왜냐하면 서보 피드백은 이 모드에서 12 비트의 소수점 카운트 확장을 가지기 때문입니다.

MACRO 링에서 수신된 포착 위치 데이터를 사용하기 위해 **Motor[x].CaptPosLeftShift** 는 13 으로 설정되어야 합니다. 왜냐하면 서보 루프 피드백이 비트 13 에서 정수 카운트 데이터를 가지기 때문입니다.

Motor[x].CaptPosRightShift 모터의 포착 위치 우측 시프트 수

설명: 포착 위치 레지스터를 우측 시프트할 비트 수

범위: 0 ..31

단위: 비트

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

Motor[x].CaptPosRightShift에서는 Power PMAC 이 **Motor[x].pCaptPos**에서 주소 지정된 32 비트 레지스터의 하드웨어 포착 위치를 초기 처리 스텝으로 우측 시프트할 비트 수를 결정합니다. 이 조작의 목적은 32 비트 소스 레지스터인 하위 비트의 "불필요한 데이터"를 가능한 한 많이 제거하여 최종적인 값인 비트 0(리틀 엔디언)을 32 비트 레지스터의 실제 데이터인 LSB에 남기는 것입니다. 따라서 **Motor[x].CaptPosRightShift**는 일반 32 비트 소스 워드에 실제 데이터인 LSB의 비트 번호로 설정됩니다.

대다수 어플리케이션에서 이 값은 재초기화(\$\$\$** 명령)로 수행되는 하드웨어 자동 식별과 일부 설정에서 적절하게 지정됩니다. 그것은 공장에서 또는 사용자에게 의한 시스템 어셈블리로 또는 그 후에 수행됩니다. 그리고 **Motor[x].EncType**의 값이 Power PMAC 스크립트 환경을 통해 또는 온라인 명령, 버퍼 프로그램 명령 또는 전원 투입/리셋에서 저장된 값으로 설정될 때 **Motor[x].EncType**의 값으로 지정된 하드웨어 인터페이스(예를 들어 PMAC2 스타일, PMAC3 스타일, MACRO) 타입에서 이 값은 표준값으로 설정됩니다.

ACC-24E2, ACC-24E2A, ACC-24E2S 또는 ACC-51E와 같은 PMAC2 스타일 "DSPGATE1" 서보 IC의 표준 24 비트 포착 위치 레지스터를 사용하기 위해 **Motor[x].CaptPosRightShift**는 8로 설정합니다. 왜냐하면 하위 8 비트가 32 비트인 Power PMAC 버스 상에서 불안정한 값이기 때문입니다.

ACC-24E3 또는 Power PMAC "Brick"에서 PMAC3 스타일 "DSPGATE3" IC의 32 비트 포착 위치 레지스터의 24 비트 정수 카운트 부분만 사용하기 위해서는 하위 8 비트의 1/T 소수점 카운트 추정값을 제거하기 위해 **Motor[x].CaptPosRightShift**를 8로 설정합니다. 이 설정은 원점 서치 동작의 포착에서 디지털 AB 상과 사인파 엔코더 양쪽에서 일반적으로 사용됩니다. 그것은 대부분의 사용자가 홈 포착 위치이므로 정수 카운트 엣지를 사용하기 때문입니다.

하위 8 비트에 1/T의 소수점 카운트 추정을 포함하는 PMAC3 스타일 "DSPGATE3" IC의 32 비트 포착 위치 레지스터 전체를 사용하기 위해 **Motor[x].CaptPosRightShift**는 0으로 설정합니다. 왜냐하면 데이터가 32 비트 레지스터의 LSB를 포함하기 때문입니다. 이 설정은 프로빙과 위치 등록의 포착에서 디지털 AB 상과 사인파 엔코더 양쪽에서 일반적으로 사용됩니다. 대부분의 사용자가 해당 포착에 대해 가능한 한 높은 분해능을 원하기 때문입니다.

MACRO 링 상에 송수신된 포착 위치 데이터를 사용하기 위해 **Motor[x].CaptPosRightShift**는 0으로 설정합니다. 왜냐하면 데이터는 반환 데이터 레지스터의 LSB를 포함하기 때문입니다.

Motor[x].CaptPosRound 모터의 포착 위치 반올림 제어

설명: 포착 위치 반올림 제어

범위: 0 ..1

단위: 논리값

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

Motor[x].CaptPosRound 는 Power PMAC 이 트리거 포함 동작의 포착 위치에 1/2 카운트를 오프셋할지 여부를 결정합니다. 0 으로 설정되는 경우에 Power PMAC 은 이 오프셋을 실행하지 않습니다. 1 로 설정되는 경우에 Power PMAC 은 이 오프셋을 실행합니다.

대다수 어플리케이션에서 이 값은 재초기화(\$\$\$** 명령)로 수행되는 하드웨어 자동 식별과 일부 설정에서 적절하게 지정됩니다. 이는 공장에서 또는 사용자에게 의한 시스템 어셈블리로 또는 그 후에 수행됩니다. 그리고 **Motor[x].EncType** 의 값이 Power PMAC 스크립트 환경을 통해 또는 온라인 명령, 버퍼 프로그램 명령 또는 전원 투입/리셋에서 저장된 값으로 설정될 때 **Motor[x].EncType** 의 값으로 지정된 하드웨어 인터페이스(예를 들어 PMAC2 스타일, PMAC3 스타일, MACRO) 타입에서 이 값은 표준값으로 설정됩니다.

모터의 서보 루프 피드백 위치가 서브 카운트 확장을 사용하는 경우에 이 오프셋이 실행됩니다. 그러나 트리거 포착된 위치에는 실행되지 않습니다. 이는 서브 카운트 확장이 정수 카운트값을 정수 카운트 엣지의 중간으로 하기 위한 1/2 카운트 오프셋을 자동으로 포함하기 때문입니다.

서보 루프 피드백 위치와 트리거 포착된 위치 양쪽이 서브 카운트 확장을 사용하거나 또는 사용하지 않는 경우에 오프셋은 실행되지 않습니다.

Motor[x].CaptureMode 모터의 트리거 동작 위치 포착 모드

설명: 트리거 동작 위치 포착 모드

범위: 0 ..3

단위: 없음

기본값: 0

기준 I 변수명: lx97

Motor[x].CaptureMode 는 위치 포착 기능이 트리거 동작 중인 모터에서 어떻게 기능하는지 지정합니다. 트리거 동작은 원점 서치 동작, 트리거 대기 조그 동작 그리고 모션 프로그램의 트리거 대기 실행 동작을 포함합니다.

Motor[x].CaptureMode 가 기본값 0 으로 설정되는 경우에 포착 트리거는 **Motor[x].pCaptFlag** 로 지정된 ASIC 채널의 트리거 입력(플래그와 인덱스 채널의 AND 나 OR 는 변경 가능)을 사용합니다. 포착 위치는 **Motor[x].pCaptPos** 로 주소 지정된 레지스터에서 얻을 수 있습니다. 그 값은 동일한 ASIC 채널의 "원점 포착" 엔코더 레지스터가 됩니다. 이 레지스터는 트리거 조건에서 즉시 래치됩니다. 따라서 정확하지만 이 ASIC 채널의 엔코더 카운터를 통해 처리된 센서에서 모터의 실제 위치가 도출돼 있어야 합니다.

Motor[x].CaptureMode 가 1 로 설정되는 경우에 포착 트리거는 **Motor[x].pCaptFlag** 로 지정된 ASIC 채널의 트리거 입력(플래그와 인덱스 채널의 AND 나 OR 는 변경 가능)을 사용합니다. 포착 위치는 위치값인 소스와 관계없이 모터의 현재 "실제 위치" 레지스터에서 얻을 수 있습니다. 이 "소프트웨어 포착" 기법은 모든 타입의 포지션 센서에서 사용할 수 있지만 포착 시에 1 개 서보 사이클 분량의 지연과 불확실성을 가집니다.

Motor[x].CaptureMode 가 2 로 설정되는 경우에 포착 트리거는 **Motor[x].WarnFeLimit** 로 설정한 경고 리미트를 모터 편차가 초과하면 발생합니다. 포착 위치는 위치값인 소스와 관계없이 모터의 "실제 위치" 레지스터에서 얻을 수 있습니다. 이 모드는 보통 물리적 정지에서의 원점 복귀나 단단히 나사를 조이는 것과 같은 "토크 한계" 트리거에 사용됩니다.

Motor[x].CaptureMode 가 3 으로 설정되는 경우에 포착 트리거는 **Motor[x].pCaptFlag** 로 지정된 PMAC3 스타일 ASIC 채널의 트리거 입력(플래그와 인덱스 채널의 AND 나 OR 는 변경 가능)을 사용합니다. 포착 위치는 트리거에서 래치된 시간값에 따라서 모터의 "실제 위치" 이력의 두 값 사이에서 보간되어 도출됩니다. 이 "타이머 보조 소프트웨어 포착" 기법은 채널의 엔코더 카운터를 통해 처리되지 않은 피드백에서조차 "하드웨어 포착" 기법에 가까운 정밀도를 제공할 수 있습니다.

이 타이머 보조 소프트웨어 포착 기법이 작동하기 위해서 **Gate3[j].Chan[j].Pfm** 의 적절한 설정으로 서보 사이클당 16 펄스가 생성되도록 채널의 PFM 출력을 구성해야 합니다. 이 펄스 열은 **Gate3[j].Chan[j].EncCtrl** 을 8 로 설정하여 채널의 카운터로 전송할 수 있어야 합니다. 펄스 카운트의 하드웨어 1/T 확장은 **Gate3[j].Chan[j].TimerMode** 를 기본값 0 으로 설정하여 유효가 되어야 합니다. PFM 명령이 각각의 레지스터에 있도록 **Gate3[j].Chan[j].PackOutData** 는 기본값 0 으로 설정되어야 합니다. 타이밍을 위해 펄스 카운터의 래치된 값을 사용하도록 **Motor[x].pCaptPos** 는 **Gate3[j].Chan[j].HomeCapt.a** 로 설정해야 합니다.

Motor[x].CaptureMode 는 풀 워드 요소 **Motor[x].Control[0]**의 비트 16-17 을 구성합니다.

Motor[x].CmdMotor 모터의 갠트리 주축 모터 번호

설명: 갠트리 추종 모드에서의 갠트리 주축의 모터 번호

범위: 0 ..**Sys.MaxMotors** - 1

단위: Power PMAC 모터 주소

기본값: 0

Motor[x].CmdMotor 에서는 모터가 "갠트리 추종" 모드일 때 어느 모터가 이 모터의 "갠트리 주축"인지 지정합니다. 이 모터의 **Motor[x].ServoCtrl** 이 8 로 설정되고 특별한 "갠트리 추종" 모드가 유효가 된 경우만 사용됩니다.

갠트리 추종 모드에서 이 모터는 그 자체의 궤적을 계산하는 대신 **Motor[x].CmdMotor** 로 지정된 모터에서 계산된 명령 궤적을 항상 사용합니다. 이처럼 여러 개의 모터가 여러 번 계산되지 않고도 동일한 명령 궤적을 사용할 수 있습니다. 각각의 모터가 각각의 피드백에 의해 독립하여 클로즈 서보 루프를 실행하며 지정된 모든 위상 인터럽트 태스크(전류(commutation)와 전류 루프)는 독립하여 실행한다는 점에 주의하십시오.

Motor[x].Control[j] 모터의 요소 유효 제어

설명: 요소를 유효로 하기 위한 모터의 풀 워드 요소

범위: \$00000000..\$FFFFFFFF

단위: 비트 필드

기본값: \$00000000

Motor[x].Control[j]는 모터의 기본적인 기능을 유효로 하기 위한 설정 요소로 구성되는 풀 워드 요소의 배열입니다.

Motor[x].Control[0]은 다음과 같은 부분 워드 요소로 구성됩니다.

성분	비트	기능
ServoCtrl	31 - 28	서보 태스크를 유효로 하는 제어 플래그
PhaseCtrl	27 - 24	위상 전류(commutation) 태스크를 유효로 하는 제어 플래그
MasterCtrl	23 - 20	마스터 추종 모드를 유효로 하는 제어 플래그
CaptureMode	19 - 16	트리거 포함 동작의 위치 포착 모드
RapidSpeedSel	15	Rapid 동작 모드의 속도 선택
PowerOnMode	14 - 12	전원 투입/리셋 시의 동작 제어
PhaseSplineCtrl	11 - 10	위상 인터럽트 스플라인 보간의 차수
AmpFaultLevel	09	앰프 이상 논리 상태
--	08	(추후 사용하기 위해 예약)
PhaseMode	07 - 05	위상 전류(commutation) 제어 모드의 플래그
FaultMode	04 - 02	이상 조건에서의 동작 제어
--	01 - 00	(추후 사용하기 위해 예약)

Motor[x].Control[1]은 다음과 같은 부분 워드 요소로 구성됩니다.

성분	비트	기능
DacShift	31 - 27	명령 출력값의 우측 시프트 거리
PhaseEncRightShift	26 - 22	위상 위치의 우측 시프트 비트 수 #
PhaseEncLeftShift	21 - 17	위상 위치의 좌측 시프트 비트 수 #
EncLossLevel	16	센서 소실 비트의 이상 상태
--	15 - 00	(추후 사용하기 위해 예약)

사용자는 요소의 설정과 그 설정을 확인하기 위해서 일반 스크립트 환경에서 이 부분 워드 요소에 액세스합니다. Power PMAC 은 효율화를 위해서 그 풀 워드 요소로 저장 및 복원합니다.

풀 워드 요소의 값이 스크립트 환경에서 참조되는 경우에 아무리 부분 워드 요소 값이 10 진법으로 통지되더라도 Power PMAC 은 16 진법 형식으로 값을 반환합니다.

그 요소 기능의 자세한 설명에 관해서는 각 부분 워드 요소의 항목을 참조하십시오.

Motor[x].Ctrl 모터의 서보 알고리즘 선택

설명: 선택된 서보 루프 알고리즘에 대한 포인터

범위: 정당한 주소

단위: Power PMAC의 메모리 주소

기본값: **Sys.ServoCtrl**

Motor[x].Ctrl 은 서보 알고리즘 코드의 저장 메모리 주소를 지정하여 어느 서보 알고리즘이 지정한 모터에 사용되는지 결정합니다. 표준 알고리즘이 사용되는 경우에 하나의 시스템 주소 상수를 사용하여 설정할 수 있습니다. (주소의 수치를 알 필요는 없습니다)

Motor[x].Ctrl 이 **Sys.PidCtrl**(하나의 주소)로 설정되는 경우에 이 모터에서는 아주 기본적인 PID 서보 알고리즘이 사용되고 아주 신속하게 실행됩니다.

이것이 **Sys.ServoCtrl**(또 하나의 주소)로 설정되는 경우에 많은 유익한 기능을 추가한 기본적인 PID 알고리즘의 상위판인 표준 서보 알고리즘이 사용됩니다. 이것은 기본설정입니다.

Motor[x].Ctrl 이 **Sys.GantryXCtrl**(또 하나의 주소)로 설정되는 경우에 이 모터(갠트리 주축)에서는 다음으로 큰 번호의 모터와 함께 크로스 커플링항을 가지는 특별한 2축의 갠트리 모터 알고리즘이 사용됩니다. 양쪽 모터의 서보 계산이 이 모터의 알고리즘으로 실행되도록 **Motor[x].ExtraMotors** 를 1로 설정합니다.

Motor[x].Ctrl 이 **Sys.AdaptiveCtrl**(또 하나의 주소)로 설정되는 경우에 이 모터에서는 적응 서보 알고리즘이 사용됩니다. 여기에서는 자동으로 관성의 변화를 검출하고 보정하기 위해 게인의 값을 변경할 수 있습니다

Motor[x].Ctrl 이 **Sys.PosCtrl**(또 하나의 주소)로 설정되는 경우에 이 모터에서는 서보 사이클마다 **Motor[x].pDac** 로 지정된 32 비트 레지스터에 해당 명령 위치를 단순히 출력합니다. 모터의 클로즈 서보 루프는 Power PMAC에서는 실행되지 않습니다. 이 설정은 주로 모든 루프가 네트워크형 드라이버로 실행되는 모션 네트워크용입니다.

고객의 전용 사용자 작성 서보를 위해 IDE 프로그램은 해당 이용 가능한 목록에서 한 개의 알고리즘을 선택하여 그 주소를 대화식으로 지정할 수 있습니다. 보통 이 설정은 **UserAlgo.ServoCtrlAddr[j]**가 됩니다.

Motor[x].CurrentNullPeriod 모터의 전류 제로 조정 사이클 수

설명: 전류 자동 제로 평균의 사이클 수

범위: -32,768..32,767

단위: 위상 인터럽트 주기

기본값: 0(무효)

Motor[x].CurrentNullPeriod 는 해당 기능의 유효호화와 모터의 디지털 전류 루프에서의 전류 피드백 "자동 제로 조정" 측정 처리 기간을 제어합니다. 디지털 전류 루프가 유효가 된 모터에만 사용됩니다. (**Motor[x].PhaseCtrl**>0 그리고 **Motor[x].pAdc**>0 인 경우) **Motor[x].CurrentNullPeriod** 가 기본값 0 으로 설정되는 경우에 자동 제로 조정 처리는 무효가 됩니다. **CurrentNullPeriod** 는 V1.5 펌웨어(2012 년 3/4 분기)에서 새로 추가되었고 기본값 0 일 때의 동작은 이전 펌웨어 버전과 완전히 호환됩니다.

Motor[x].CurrentNullPeriod 가 0 이 아닌 값으로 설정되는 경우에 모터가 오픈 루프 상태에서 클로즈 서보 루프 상태로 명령을 받을 때마다 Power PMAC 은 자동으로 제로 조정 처리를 실행합니다. 이 처리에서 모터의 A/B 상전류 피드백의 ADC 값이 제로 전류를 통지해야 할 때 반복 샘플링됩니다. **Motor[x].CurrentNullPeriod** 의 크기에는 샘플링되는 각각의 값의 횟수를 지정합니다. 각 상에 해당 샘플링의 평균이 계산되고 다음으로 각 상의 바이어스항(**Motor[x].IaBias** 와 **Motor[x].IbBias**)은 해당 평균의 음수로 자동 설정됩니다. 자동 제로 조정이 유효인 경우에 모터가 Enable 될 때마다 자동으로 덮어쓰기되므로 해당 바이어스항이 저장된 값은 사용되지 않는다는 점에 주의하십시오.

이 처리는 모터의 위상 참조에도 실행됩니다. 이는 모터의 온라인 명령 \$ 또는 **Motor[x].PhaseFindingStep** 을 직접 1 로 설정하여 기동시킬 수 있습니다. (또한 실제로 위상 참조하지 않고 전류의 자동 제로 조정을 실행하기 위해 **Motor[x].PhaseFindingStep** 을 8 로 설정할 수 있습니다.) 사용자가 직접 **Motor[x].PhaseFindingStep** 을 설정하는 경우에 자동 제로 조정을 실행하기 위해 모터가 적정한 "정지 상태"임을 사용자가 보증할 책임이 있습니다. **Motor[x].PowerOnMode** 의 비트 1(값 2)이 1 로 설정되는 경우에 이 위상 참조는 Power PMAC 의 전원 투입/리셋 시 자동으로 실행됩니다. 이 경우에 서보 드라이버에서 적절하게 자동 제로 조정 측정을 실행할 준비가 되었음을 확인할 수 없으므로 보통 자동 제로 조정은 권장하지 않습니다.

모터가 아직 Enable 되지 않은 경우에 모터가 할당된 좌표계에서 **enable** 명령(온라인 또는 프로그램)을 실행해도 자동 제로 조정은 실행됩니다. 또한 모터의 클로즈드 루프를 실행하는 온라인 j/명령 실행 시에도 자동 제로 조정은 실행됩니다.

이 모터에서 자동 브레이크 제어가 유효가 된 경우(**Motor[x].pBrakeOut** = 0), 자동 제로 조정 처리가 완료된 후에 제어 Enable 후의 지연 기간(**Motor[x].BrakeOffDelay** msec)이 시작됩니다.

Motor[x].CurrentNullPeriod 가 양의 값으로 설정되는 경우에 서보 드라이버로의 앰프 Enable 신호가 실제 출력되기 전에 해당 자동 제로 조정 처리가 수행됩니다. 이는 모든 상의 PWM 출력이 완전히 무효가 되었고 상의 전류가 흐르고 있지 않다는 것을 보증한 상태에서 측정이 수행됨을 의미합니다. 이 경우에 측정되는 바이어스는 전류 피드백 회로에만 관련된 값이 됩니다. 자동 제로 조정 처리가 완료된 후에 앰프 Enable 신호는 브레이크 해방 지연 시작 후 실제 출력되어 모터 제어를 시작할 수 있을 것입니다.

Motor[x].CurrentNullPeriod 가 음수로 설정되는 경우에 자동 제로 조정 처리가 수행되기 전에 서보 드라이버로의 앰프 Enable 신호는 실제 출력됩니다. 이 경우에 모든 상이 공칭 제로 전압값을 명령하도록 Power PMAC 은 일시적으로 자동 제로 조정 처리 기간 동안 강제적으로 **Motor[x].PwmSf** 를 0 으로 설정합니다. 이 경우에 측정되는 바이어스는 PWM 전압 명령 회로와 전류 피드백 회로의 조합이 됩니다. 자동 제로 조정 처리가 완료된 후 **Motor[x].PwmSf** 의 값은 이전 값으로 브레이크 해방 지연이 시작된 후 복원되며 모터 제어를 시작할 수 있습니다.

자동 제로 조정 처리 초기에 상태 요소 **Motor[x].CurrentNullTimer** 는 **CurrentNullPeriod** 의 크기로 설정됩니다. 이어서 일어나는 위상 사이클마다 한 개씩 뺍니다. 그 값이 0 보다 클 때는 상태 요소 **Motor[x].IaMeas** 와 **Motor[x].IbMeas** 는 단순히 현재 사이클의 값을 통지하는 대신 처리 초기에 읽은 값의 합을 누적합니다. **CurrentNullTimer** 가 0 에 도달하면 그 합을 **CurrentNullPeriod** 로 나눕니다. 그리고 최종적인 값의 음수가 **Motor[x].IaBias** 와 **Motor[x].IbBias** 에 자동으로 쓰여지고 자동 제로 조정 자체가 종료합니다. 그 후에 위상 참조 처리의 나머지 또는 Enable 처리를 계속합니다. 전체 프로세스의 종료를 감시하기 위해서는 **Motor[x].ClosedLoop** 또는 **Coord[x].ClosedLoop** Status 비트에서 확인할 수 있습니다.

Motor[x].DacBias 모터의 서보 출력 오프셋

설명: 서보 출력 오프셋

범위: -32,768.0..32,768.0

단위: 16 비트 DAC 와 등가

기본값: 0

Motor[x].DacBias 는 위치/속도 서보 루프에서 계산된 출력 명령에 더해지는 오프셋을 지정합니다. 명령값의 크기가 **Motor[x].MaxDac** 의 제한과 비교된 후에 이 오프셋은 더해집니다. 따라서 오프셋된 값이 그 제한보다 커질 수 있습니다.

이 요소의 이름에도 불구하고 D/A 컨버터가 출력에 사용될 필요가 없다는 점에 주의하십시오. PMAC 이 전류(commutation) 및(또는) 전류 루프 계산을 실행 중인 경우 **Motor[x].DacBias** 는 그러한 알고리즘으로 토크 입력값을 오프셋합니다. 이는 전류(commutation) 상 바이어스항의 **Motor[x].IaBias** 와 **Motor[x].IbBias** 와는 관계없습니다.

출력 장치의 실제 분해능이 달라도 **Motor[x].DacBias** 는 16 비트 출력 단위입니다. 예를 들어 18 비트 DAC 출력에서는 **Motor[x].DacBias** 의 100 의 값은 해당 DAC 에 쓰여지는 400 의 오프셋과 등가가 됩니다.

Motor[x].DacShift 모터의 출력 우측 시프트 양

설명: 출력값의 우측 시프트 양

범위: 0 ..31

단위: 비트

기본값: 0

Motor[x].DacShift 에서는 명령 출력값이 출력 레지스터에 쓰여지기 전에 우측 시프트되는 비트 수를 지정합니다. 자동으로 "왼쪽 자리맞춤"되지 않은 출력 장치의 사용을 가능하게 합니다.

대부분의 출력 장치에서는 **Motor[x].DacShift** 는 기본값 0 으로 설정됩니다. ACC-24E3 의 18 비트 DAC 옵션인 경우에 해당 DAC 은 24 비트 데이터 영역의 오른쪽 자리맞춤 18비트를 사용하고 있으므로 이 값은 6 으로 설정해야 합니다.

Motor[x].DacShift 는 폴 워드 요소 **Motor[x].Control[1]**의 비트 27-31 을 구성합니다.

Motor[x].DtOverRotorTc 모터의 슬립 상수

설명: 유도 모터의 슬립 상수

범위: 음이 아닌 부동 소수점

단위: 없음(시간의 비율)

기본값: 0.0

기존 I 변수명: lxx78

Motor[x].DtOverRotorTc 에서는 토크 명령과 AC 비동기(유도) 모터의 회전자 자계의 슬립 주파수와 관계의 제어를 제어합니다. 이는 보통 실험적으로 설정되지만 위상 갱신 주기와 회전자(고정자가 아닌)의 L/R 전기적 시상수 간의 비율로 계산합니다.

Motor[x].DtOverRotorTc 가 0 보다 큰 경우 Power PMAC 은 **Motor[x].IdCmd** 에서의 고정자 다이렉트 명령 전류값을 필터링하고 사이클마다 회전자 자화 전류 **Motor[x].Imag** 를 추정합니다. 이 전류값을 **DtOverRotorTc** 로 나누면 **Motor[x].SlipGain** 에 저장된 현재의 "슬립 상수"가 됩니다. 다음으로 **Motor[x].IqCmd**(서보 루프 출력)로 제어되는 직교 전류값을 이 값으로 곱하면 슬립 주파수가 됩니다. **Motor[x].PhaseCtrl** 의 비트 1(값 2)이 0 으로 설정되는 경우에 해당 슬립 연산은 직교 전류와 다이렉트 전류에 대해 측정값을 사용합니다. 이 비트가 1 로 설정되는 경우에 해당 계산은 양쪽 전류에 대해 명령값을 사용합니다.



DtOverRotorTc 가 0.0 으로 설정되는 경우에 **SlipGain** 은 사용자가 직접 설정하여 저장할 수 있습니다. 이는 오픈 루프의 다이렉트 마이크로 스테핑에 도움이 됩니다.

Motor[x].DtOverRotorTc 는 보통 데이터 수집 기능을 사용하면서 모터에 오픈 루프(토크) 출력 명령을 내려 속도 응답을 보고 실험적으로 설정할 수 있습니다. 역기전력이 공급 전압에 도달하면 속도가 포화하므로 속도는 안정 상태에 도달하기 위해 약 5% 내려갑니다. 이 이상 내려가면 슬립 시상수는 너무 높아집니다. 이보다 내려가지 않거나 전혀 내려가지 않으면 슬립 시상수는 너무 낮아집니다.

약 9kHz 의 위상 갱신 주파수에서의 표준 유도 모터의 **Motor[x].DtOverRotorTc** 표준값은 0.00015 입니다.

Power PMAC의 전류(commutation)를 지정하기 위해 **Motor[x].PhaseCtrl**의 비트 0(값 1) 또는 비트 2(값 4)가 1로 설정된 경우에만 **Motor[x].DtOverRotorTc**는 유효합니다. 영구자석 브러시리스 모터, 스텝핑 모터, SR(switched reluctance) 모터 등의 동기 모터를 전류(commutation)하는 경우에는 **Motor[x].DtOverRotorTc**는 0으로 설정되어야 합니다.

Motor[x].EncLossBit 모터의 엔코더 소실 비트 번호

설명: pEncLoss 레지스터에서의 센서 소실 플래그의 비트 번호

범위: 0 ..31

단위: 비트 번호(리틀 엔디언)

기본값: 0

Motor[x].EncLossBit는 Power PMAC이 모터의 엔코더 소실을 검출하기 위해 **Motor[x].pEncLoss**로 주소 지정되는 32비트 레지스터의 어느 비트가 사용되는지 결정합니다. 비트 번호는 "리틀 엔디언" 규약으로 지정됩니다. 그때 비트 n 은 32비트 워드 중의 2^n 의 값이 됩니다. 이 엔코더 소실 Status 비트의 극성은 **Motor[x].EncLossLevel**로 설정됩니다.

"배타적 논리합" 직교 소실 검출 회로를 가지는 ACC-24E2x PMAC2 스타일 축 인터페이스 보드에 연결된 AB 상(90도 위상차가 있는 직교) 엔코더에는 **Gate1[j].Chan[j].EncLossN** Status 비트가 해당 레지스터의 비트 13에서 검출되도록 **Motor[x].EncLossBit**는 13으로 설정합니다.

그리고 "배타적 논리합" 직교 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 AB 상 엔코더에는 **Gate3[j].Chan[j].LossStatus** 비트가 레지스터의 비트 28에서 검출되도록 **Motor[x].EncLossBit**는 28로 설정합니다.

"제곱합" 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 아날로그 사인파 엔코더 또는 리졸버에는 **Gate3[j].Chan[j].SosError** 비트가 해당 레지스터의 비트 31에서 검출되도록 **Motor[x].EncLossBit**는 31로 설정합니다.

일부 프로토콜에 좌우되는 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 시리얼 엔코더에는 채널의 "타임아웃" 에러 비트가 **Gate3[j].Chan[j].SerialEncDataB** 레지스터의 비트 31에서 검출되도록 일반 **Motor[x].EncLossBit**는 31로 설정합니다. 프로토콜에 따라 다를 가능성도 있습니다.

Motor[x].EncLossLevel 모터의 엔코더 소실 레벨

설명: 센서 소실 이상의 논리 상태

범위: 0 ..1

단위: 논리값

기본값: 1

Motor[x].EncLossLevel 은 엔코더 소실 이상의 원인이 되는 엔코더 소실 Status 비트의 소실 상태를 지정합니다. 이 상태의 검출에 필요한 수는 **Motor[x].EncLossLimit** 로 지정됩니다.

Motor[x].EncLossLevel 이 0 으로 설정되는 경우에 0 의 값이 이상으로 판단됩니다. 그 값이 1 로 설정되는 경우에 1 의 값이 이상으로 판단됩니다. 엔코더 소실 비트는 **Motor[x].pEncLoss** 로 지정된 레지스터의 **Motor[x].EncLossBit** 로 지정된 비트 번호에서 읽습니다. **Motor[x].pEncLoss** 가 0 으로 설정되고 엔코더 소실 검출 기능을 무효로 하는 경우 **Motor[x].EncLossLevel** 의 설정은 관계없습니다.

"배타적 논리합" 직교 소실 검출 회로를 가지는 ACC-24E2x PMAC2 스타일 축 인터페이스 보드에 연결된 AB 상(90 도 위상차가 있는 직교) 엔코더에는 **Motor[x].EncLossLevel** 은 0 으로 설정합니다.

그리고 "배타적 논리합" 직교 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 AB 상 엔코더에는 **Motor[x].EncLossLevel** 은 1 로 설정합니다.

"제곱합" 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 아날로그 사인파 엔코더 또는 리졸버에는 **Motor[x].EncLossLevel** 은 1 로 설정합니다.

일부 프로토콜에 좌우되는 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드 또는 Power Brick 제어 보드에 연결된 시리얼 엔코더에는 **Motor[x].EncLossLevel** 은 1 로 설정합니다.

Motor[x].EncLossLimit 모터의 엔코더 소실 제한

설명: 이상 검출의 센서 소실 최대 누적 수

범위: 0 ..255

단위: 스캔 수

기본값: 0

Motor[x].EncLossLimit 는 Power PMAC 이 엔코더 소실 이상에서 트립하지 않고 "소실" 상태의 엔코더를 검출할 수 있는 누적 스캔의 최대수를 지정합니다. 소실 상태에서의 누적 스캔 수가 이 값을 초과하는 경우에 Power PMAC 은 자동으로 이 모터를 "kill"하고 **Motor[x].FaultMode** 의 설정값에 의해 결정되지만 동일한 좌표계의 다른 모터를 kill 또는 Abort 합니다. Power PMAC 은 실시간 인터럽트마다 지정된 엔코더 소실 비트의 상태를 확인합니다.

엔코더 소실 검출 회로의 대부분은 전기적 노이즈 또는 유사한 요인에 의한 소실을 일시적으로 감지할 수 있습니다. 소실을 검출하는 데 필요한 여러 개의 연속되는 스캔 분량만 실제 소실을 확인하는 기능은 잘못된 트립의 제거에 도움이 됩니다.

엔코더 소실 비트의 레지스터는 **Motor[x].pEncLoss** 로 지정됩니다. 이 레지스터 내의 해당 비트는 **Motor[x].EncLossBit** 로 지정됩니다. 이 비트의 소실 상태는 **Motor[x].EncLossLevel** 로 지정됩니다. **Motor[x].pEncLoss** 가 0 으로 설정되고 엔코더 소실 검출 기능이 무효가 되는 경우 **Motor[x].EncLossLimit** 의 설정은 관계없습니다. 지정된 비트로 "소실" 상태를 검출한 누적 스캔 수는 상태 요소 **Motor[x].EncLossCount** 로 조사됩니다. 이 값은 소실 상태가 검출될 때마다 더해지고 정상 상태가 검출될 때마다 빼집니다(값이 0 보다 큰 경우).

Motor[x].EncType 모터의 엔코더 타입

설명: 포착 제어의 위치 피드백 타입

범위: 0 ..255

단위: 열거 형식

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

Motor[x].EncType에서는 원점 서치와 같은 트리거 포함 동작의 하드웨어 포착 위치를 사용하는 데 어떤 타입의 서보 피드백이 사용되는지 어떤 타입의 플래그 인터페이스가 사용되는지를 나타냅니다. **Motor[x].EncType**의 값은 그 자체로 어떤 기능도 제어하지 않습니다. 그러나 Power PMAC 스크립트 환경(온라인 명령, 버퍼 프로그램 명령, 전원 셋업/리셋 시의 플래시 메모리에서의 복사)에서 **Motor[x].EncType**의 값을 설정하는 조작은 여러 개의 설정 요소를 강제적으로 설정하여 그러한 기능을 제어하게 됩니다.

다음 표는 **Motor[x].EncType**의 가능한 값과 필요한 위치/플래그 인터페이스 타입을 나타냅니다.

타입	인터페이스 타입	피드백 타입	하드웨어
0	(없음)	(하드웨어 할당 없음)	(없음)
1	PMAC2 스타일 서보 IC	AB 상 엔코더, 원시 카운트	ACC 24E2x
2	PMAC2 스타일 서보 IC	AB 상 엔코더, 1/T 확장	ACC 24E2x
3	PMAC2 스타일 서보 IC	사인파 엔코더, 아크탄젠트 확장	ACC-51E
4	MACRO 인터페이스 IC	타입 1 MACRO 링 피드백	ACC-5E, 5E3
5	PMAC3 스타일 IC	AB 상 엔코더, 1/T 확장	ACC-24E3, Power Brick
6	PMAC3 스타일 IC	사인파 엔코더, 아크탄젠트 확장	ACC-24E3, Power Brick
7	PMAC3 스타일 IC	사인파 엔코더, 자동 수정 확장	ACC-24E3
8+	(예약)	(예약)	(예약)

다음 표는 **Motor[x].EncType**이 스크립트 환경에서 한 개의 탑재 타입으로 설정되면 **Motor[x]** "종속" 변수가 자동으로 어느 값을 수신하는지 나타냅니다.

EncType ->	1	2	3	4	5	6	7
AmpEnableBit	22	22	22	22	8	8	8
AmpFaultBit	23	23	23	23	7	7	7
CaptFlagBit	19	19	19	19	20	20	20
CaptPosLeftShift	8	9	10	13	8	4	6
CaptPosRightShift	8	8	8	0	8	0	0
CaptPosRound	0	1	1	1	1	0	0
LimitBits	25	25	25	25	9	9	9

또한 **Motor[x].EncType** 이 스크립트 환경에서 어느 하나의 탑재된 타입으로 설정되면 **Motor[x].pCaptFlag**(포착 플래그 레지스터의 주소)는 **Motor[x].pEncStatus** 의 값으로 자동 설정됩니다. 그리고 **Motor[x].pCaptPos**(포착 위치 레지스터의 주소)는 **Motor[x].pEncStatus(Gaten[j].Chan[j].HomeCapt.a)**로 지정된 IC 채널의 하드웨어 포착 레지스터 주소에 자동으로 설정됩니다. **Motor[x].pCaptPos** 가 **EncType** = 4(MACRO)로 자동 설정되지 않는다는 점에 주의하십시오. 포착된 위치가 단순히 레지스터 읽기가 아니라 이 모드에서는 하나의 MACRO 통신 요구에 의해 취득되기 때문입니다.

Motor[x].ExtraMotors 모터의 추가 모터

범위: 0 ..255

단위: Power PMAC 의 모터 수

기본값: 0

Motor[x].ExtraMotors 는 이 모터의 서보 알고리즘으로 실행되는 클로즈 서보 루프의 추가 모터 수를 지정합니다. 해당 추가 모터는 이 모터의 번호보다 하나 큰 값에서 시작하는 번호로 연속해서 번호를 부여해야 합니다. 예를 들어 **Motor[7].ExtraMotors** 가 4 로 설정되는 경우에 모터 7 의 서보 알고리즘이 모터 7 과 동일하게 모터 8, 9, 10 과 11 의 서보 루프를 실행하도록 요구됩니다.

Motor[x].ExtraMotors 의 0 이외의 값은 "멀티 입력, 멀티 출력"(MIMO) 및 "크로스 커플링" 서보 알고리즘을 쉽게 사용하기 위함입니다. **Motor[x].ExtraMotors** 는 내장된 "싱글 입력 및 싱글 출력"(SISO) 서보 알고리즘을 모두 0 으로 설정해야 합니다. 예를 들어 **Motor[x].Ctrl** 이 **Sys.PidCtrl**, **Sys.ServoCtrl** 또는 **Sys.PosCtrl** 로 설정될 때입니다. 내장된 크로스 커플링의 갠트리 알고리즘이 사용되는 경우(주축 모터의 **Motor[x].Ctrl** 은 **Sys.GantryXCtrl** 로 설정되고 종축 모터는 주축 모터보다 하나 큰 번호로 번호가 부여됩니다)에는 주축 모터의 이 요소는 1 로 설정되어야 합니다.

사용자 정의 MIMO 알고리즘에서는 그룹 내의 가장 작은 번호의 모터인 **Motor[x].ExtraMotors** 는 그룹 내의 모터 수보다 하나 작은 값으로 설정해야 합니다. 해당 사용자 정의 MIMO 알고리즘을 설명할 때는 모든 "추가" 모터의 서보 출력 명령을 해당 모터의 유지 레지스터(**Motor[x].ServoOut**)에 직접 써야 한다는 점에 주의하십시오. 이 모터의 서보 출력 명령은 해당 함수의 "복귀"값이 되어야 합니다.

Motor[x].ExtraMotors 가 0 보다 n 만큼 큰 값으로 설정되는 경우에 해당 다음 n 개의 모터에서는 다른 서보 알고리즘은 실행되지 않습니다.

Motor[x].FatalFeLimit 모터의 중대한 편차 이상 리미트

설명: 중대한(셋다운) 편차 이상 리미트

범위: 음이 아닌 부동 소수점

단위: 모터 단위

기본값: 2000.0

기준 I 변수명: lxx11

Motor[x].FatalFeLimit에서는 운전이 셋다운되도록 지정된 모터의 위치 편차의 크기를 설정합니다. 편차의 크기가 **Motor[x].FatalFeLimit**를 초과하면 지정된 모터는 "kill"됩니다(오픈 루프, 앰프 Disable, 제로 명령 출력). 해당 모터의 좌표계가 그때 모션 프로그램을 실행 중이면 해당 프로그램은 Abort 됩니다. 동일한 좌표계의 다른 모터는 **Motor[x].FaultMode**의 값에 의해 결정되지만 kill 되거나 Abort 됩니다. 그러나 다른 좌표계의 모터는 영향을 받지 않습니다.

Motor[x].FatalFeLimit를 0으로 설정하면 모터의 편차 이상 리미트를 무효로 합니다. 초기의 개발 단계에서는 편리할지도 모르지만 실제 어플리케이션에서는 반드시 피해야 합니다. 편차 이상 리미트는 대부분의 경우 직접 검출할 수 없지만 사람과 기기에 중대한 손해를 입힐 수 있는 피드백의 소실이나 피드백의 반전 등 다양한 타입의 이상으로부터 매우 중요한 보호가 됩니다.

모터 단위가 피드백의 "원시 카운트"로 정의되면 **Motor[x].FatalFeLimit**의 기본설정은 많은 시스템에서 타당한 값입니다. 모터 단위를 mm 또는 도(원시 카운트보다 보통 훨씬 큰) 등의 공업 단위로 변경하는 사용자는 유익한 안전 리미트를 유지하기 위해 **Motor[x].FatalFeLimit**에 비례해서 더 작게 해야 합니다.

Motor[x].FaultMode 모터의 이상 동작 제어

설명: 이상 동작 제어

범위: 0 ..7

단위: 비트 필드

기본값: 0

Motor[x].FaultMode는 모터의 어떤 이상 상태에서 취할 수 있는 동작을 지정합니다. 이는 3비트의 값이고 각각의 비트에서 해당 동작을 독립하여 제어합니다.

비트 0(값 1)에서는 이 모터가 **Motor[x].FatalFeLimit**로 설정되는 편차 이상 리미트를 초과하는 경우 또는 앰프 이상 상태(입력 신호 또는 **Motor[x].I2tTrip**으로 설정되는 적분 전류 리미트를 초과하는 경우 둘 중 하나)를 검출하는 경우 또는 "엔코더 소실" 상태를 감지한 때의 동작을 제어합니다. 이 모터는 이러한 경우 항상 "kill"됩니다(오픈 루프, 제로 명령 출력, 앰프 Disable로 합니다).

이 비트가 기본값 0 으로 설정되는 경우에 좌표계의 다른 모터는 이 경우에 "Abort"됩니다(**Motor[x].AbortTa** 와 **Motor[x].AbortTs** 에 따라 클로즈드 루프 정지 상태로 감속 정지됩니다). 이 비트가 1 로 설정되는 경우에 좌표계의 다른 모터도 이러한 경우에 kill 됩니다. 갠트리 시스템과 같이 해당 동작이 기계적으로 튼튼하게 연결되는 모터에서는 연결된 모든 모터가 그 중 하나의 이상이라도 kill 되는 것이 중요합니다. 이상 시 좌표계의 다른 모터가 이 모터와의 협조 모션을 수행하지 않아도 좌표계의 다른 모터가 그 이상 중 하나에서 Abort 되었거나 kill 된다는 점에 주의하십시오.

다른 좌표계의 모터는 이러한 사례에 결코 영향을 받지 않습니다.

비트 1(값 2)에서는 이 모터가 오픈 루프의 Enable 된 모드(온라인 outn 또는 버퍼 명령 counn 의 경우)에서 하드웨어 또는 소프트웨어의 위치 리미트에 도달했을 때 동작을 제어합니다. 이 비트가 기본값 0 으로 설정되는 경우에 모터는 "Abort"됩니다(**Motor[x].AbortTa** 와 **Motor[x].AbortTs** 에 따라 클로즈드 루프 정지 상태로 감속 정지합니다). 그 값이 1 로 설정되는 경우에 모터는 "kill"됩니다(오픈 루프, 제로 명령 출력, 앰프 Disable 로 합니다).

비트 2(값 4)에서는 이 모터가 클로즈드 루프에서 Enable 된 모드에서 소프트웨어 리미트에 아직 도달하지 않은 경우에 하드웨어 리미트에 도달했을 때의 동작을 제어합니다. 이 비트가 기본값 0 으로 설정되는 경우에 모터는 "Abort"됩니다(**Motor[x].AbortTa** 와 **Motor[x].AbortTs** 에 따라 클로즈드 루프 정지 상태로 감속 정지합니다). 그 값이 1 로 설정되는 경우에 모터는 "kill"됩니다(오픈 루프, 제로 명령 출력, 앰프 Disable 로 합니다). 이 경우의 모터를 kill 하는 배경에 있는 의도는 위치 정보를 잃게 되는 경우 즉 제어 정지가 불가능한 경우에만 수행되어야 한다는 것입니다.

Motor[x].GantrySlewRate 모터의 갠트리 슬루 레이트

설명: 갠트리 종축 모터의 원점 서치에서의 경사 제거 레이트

범위: 음이 아닌 부동 소수점

단위: 모터 단위/서보 사이클

기본값: 0.0

Motor[x].GantrySlewRate 는 "갠트리 종축" 모드(**Motor[x].ServoCtrl** = 8)에서 작동된 모터가 한번 그 자체의 원점 트리거를 검출하면 그 자체의 상대적인 원점 위치에 슬루하는 레이트를 지정합니다. 이 슬루 동작 이외에 갠트리 종축 모터는 그 자체의 명령 궤적을 생성하지 않습니다. 대신 이는 **Motor[x].CmdMotor** 로 지정된 "갠트리 주축" 모터에서 생성된 궤적을 사용합니다. 이 슬루 동작은 원점 트리거 위치의 차이에 따른 주축 모터와 상대적인 종축 모터의 "경사 제거"를 가능하게 합니다.

일반적인 사용에서 갠트리 종축 모터는 갠트리 주축 모터와 동시에 원점 서치 동작을 실행하라는 명령을 받습니다(예를 들어 #1, 2hm). 원점 서치 동작 중에 종축은 주축 모터에서 생성된 궤적을 단순하게 실행합니다. 단, 한번 주축과 종축의 양쪽 모터가 원점 트리거의 양쪽을 검출하면 종축 모터는 2 개 모터의 원점 위치 간의 상대적인 거리(양쪽 모터의 트리거 위치와 **Motor[x].HomeOffset** 의 값에 따른)를 계산합니다. 그리고 그때 갠트리 주축 모터에 추종하도록 실행된 모션(만약 있다면) 상에 **Motor[x].GantrySlewRate** 에서 지정한 레이트로 이 보정을 중첩합니다.

갠트리 주축 모터의 원점 서치 동작이 그 자체의 원점 트리거 위치와 종축 모터의 원점 트리거 위치 양쪽을 항상 통과하도록 구성해야 합니다. 실제로는 감속 거리는 모터 간 전원 셋업 시의 최악의 경사보다 더 길게 해야 합니다. 원점 서치의 루틴에서는 모든 갠트리 모터의 **Motor[x].HomeComplete** Status 비트가 1 이 되는 것을 확인하고(모터의 트리거가 검출되면 설정됩니다), 모든 갠트리 종축 모터의 **Motor[x].GantryHomed Status** 비트가 1 이 되는 것도 확인해야 합니다(경사가 완전히 제거되면 설정됩니다). 해당 확인은 처리가 진행되기 전에 이루어져야 합니다.

이미 완전히 원점 서치가 완료된(**Motor[x].GantryHomed** = 1 의) 갠트리 종축 모터에서는 오픈 루프(Enable 또는 Disable 에서도)일 때 발생할 수 있는 모든 경사를 제거하기 위해 클로즈드 루프 제어가 재확인된 경우는 언제나 **Motor[x].GantrySlewRate** 가 사용됩니다.



Motor[x].GantrySlewRate 가 기본값 0.0 일 때 이 보정은 수행되지 않습니다.

Motor[x].HomeOffset 모터의 원점 오프셋

설명: 위치 참조 오프셋

범위: 부동 소수점

단위: 모터 단위

기본값: 0.0

기존 I 변수명: lxx26

Motor[x].HomeOffset에서는 모터의 센서 제로 위치와 모터 자체의 제로 "원점" 위치 간의 (부호가 있는) 차를 지정합니다. 원점 서치 동작에서 그 위치 기준을 확립하는 모터에서 이는 원점 트리거 위치와 모터 제로 위치 간의 차이가 됩니다. 절대 위치를 읽을 때 그 위치 기준을 확립하는 모터에서 이는 절대 위치 센서의 제로 위치와 모터 제로 위치 간의 차이가 됩니다.

절대 위치 읽기에서 센서 위치의 값은 **Motor[x].AbsPosFormat** 으로 지정된 형식이며 **Motor[x].pAbsPos** 로 지정된 레지스터에서 읽습니다. 이 값에 **Motor[x].AbsPosSf** 가 곱해져서 모터 단위로 변환됩니다. 다음으로 **Motor[x].HomeOffset** 을 이 결과에서 빼서 절대 위치의 모터 위치를 산출합니다.

원점 서치 동작에서는 **Motor[x].HomeOffset** 은 원점 트리거가 검출된 실제 위치와 트리거 후 동작의 명령 종료 위치(여기서 모터는 정지) 간의 거리를 지정합니다. 이 값은 트리거 위치에 더해져서 종료 위치가 됩니다. 다음으로 트리거 후 동작의 명령 종료 위치는 모터 위치 제로로 판단됩니다. (프로그램상의 목적으로 다른 축 위치 제로를 생성하는 다른 오프셋을 사용하는 것도 가능합니다.)

홈 플래그로 오버트래블 리미트를 사용할 때 트리거 위치와 모터 제로 위치 간의 차는 특히 유의합니다(리미트 입력을 제한으로 다시 유효로 하기 전에 리미트에서 오프셋합니다). **Motor[x].HomeOffset** 이 충분히 큰 경우($1/2 \times$ 원점 복귀 속도 $\times$ 가속 시간보다 큼) 방향 반전이 없는 원점 서치 동작이 가능해집니다.

Motor[x].HomeVel 모터의 원점 복귀 속도

설명: 원점 서치의 부호가 있는 명령 속도

범위: 양의 부동 소수점

단위: 모터 단위/msec

기본값: 10.0

기존 I 변수명: lxx23

Motor[x].HomeVel에서는 모터의 원점 서치 동작의 명령 속도와 방향을 설정합니다. 부호 변경은 원점 복귀 동작의 방향을 역회전합니다. 음의 값은 역방향의 원점 서치를 지정합니다. 양의 값은 정방향의 원점 서치를 지정합니다.

원점 서치 동작 트리거 후의 부분이 충분히 긴 경우에 그 속도는 **Motor[x].HomeVel**로 설정된 크기의 피크를 달성하며 그 경우의 방향은 반드시 설정된 것은 아닙니다.

예

20 미크론 모터 단위에서 역방향으로 4m/min(약 160inch/min)의 원점 속도를 설정하는 식은 다음과 같습니다.

$$HomeVel = -\frac{4m}{min} * \frac{motor\_unit}{2 * 10^{-5} m} * \frac{min}{6 * 10^4 ms} = -3.333 \left(\frac{motor\_unit}{ms} \right)$$

Motor[x].I2tSet 모터의 연속 전류 제한

설명: 연속 전류 제한

범위: 음이 아닌 부동 소수점

단위: 16 비트의 DAC/ADC 와 등가

기본값: 0.0

기존 I 변수명: lxx57

Motor[x].I2tSet 은 Power PMAC 의 적분 전류 제한 기능이 유효할 때 모터의 최대 연속 전류 제한의 크기를 지정합니다(**Motor[x].I2tSet** 은 적분 전류 제한 기능을 유효로 하기 위해 0 보다 커야 합니다). Power PMAC 이 모터의 디지털 전류 루프를 실행 중이면 실제 전류 측정값을 이 기능에 사용합니다. 그 이외에서는 명령 전류값을 사용합니다. Power PMAC 에서의 실제 또는 명령 전류 레벨의 크기가 **Motor[x].I2tTrip** 으로 설정되는 기간에 **Motor[x].I2tSet** 의 크기를 초과하면 Power PMAC 은 적분 전류 앰프 이상 상태로 이 모터를 트립합니다.

실제 출력 또는 입력 장치가 다른 분해능을 가져도 **Motor[x].I2tSet** 은 부호가 있는 16비트 DAC 또는 ADC 의 단위입니다(32,767 의 최대 유효값). 일반 **Motor[x].I2tSet** 은 **Motor[x].MaxDac**(순간) 출력 제한의 1/3~1/2 로 설정됩니다. 순간과 연속 전류 제한의 사양에 대해서는 앰프와 모터의 설명서를 참조하십시오.

기술적으로는 **Motor[x].I2tSet** 은 직교 전류와 다이렉트 전류의 벡터합의 연속 제한이 됩니다. 직교 전류(토크 생성)는 위치/속도 서보 루프에서의 출력입니다. 다이렉트(자화) 전류는 **Motor[x].IdCmd** 로 설정됩니다.

사인파 출력 모드(**Motor[x].PhaseCtrl** 의 비트 0 또는 비트 2 = 1, **Motor[x].pAdc** = 0)에서는 보통 앰프 게인은 PMAC 출력의 1 볼트당 상전류의 암페어로 주어집니다. 그러나 보통 모터와 앰프의 제한은 RMS 암페어 수치(실효값)로 주어집니다. 이 경우에 피크 상전류값이 RMS 값보다 $\sqrt{2}$ (1.414)배가 된다는 점에 주의하십시오.

3 상 모터(**Motor[x].PhaseOffset** = +/-683)의 다이렉트 PWM 모드(**Motor[x].PhaseCtrl** 의 비트 0 = 1, **Motor[x].pAdc**>0)에서는 사인파 상전류에서의 ADC 읽기의 해당 파고값은 직교와 다이렉트 전류의 벡터합보다 $1/\cos(30^\circ)$ 또는 1.15 배 커집니다. 따라서 한번 파고값이 확립되고 연속적인 상전류를 A/D 변환값으로 보고자 하는 경우에 이 값에 $\cos(30^\circ)$ 또는 0.866 을 곱해야 하며 **Motor[x].I2tSet** 의 값을 구할 수 있습니다. 전류 제한이 RMS 값으로 주어지는 경우에 피크 상전류값을 얻기 위해서는 여기에 $\sqrt{2}$ (1.414)를 곱해야 한다는 점에 주의하십시오.

예

토크 모드의 브러시리스 모터 앰프는 최대 $15A_{rms}$ 의 단속적인 최대 전류 능력(즉 $\pm 10V$ 의 입력은 $\pm 15A_{rms}$ 전류를 명령)을 가지며 해당 브러시리스 모터는 $5A_{rms}$ 의 연속 전류 제한을 가지는 경우 **I2tSet** 은 다음과 같이 계산됩니다.

$$I2tSet = 32,768 * \frac{5A_{rms}}{15A_{rms}} = 10,923$$

각 상 2A/Volt 의 상호 컨덕턴스 게인을 가지는 사인파 입력 앰프는 Power PMAC 의 전체 범위($\pm 10V$)의 사인파 명령에서 각 위상에서 20A(피크)의 전류 출력을 생성합니다. **PwmSf** 는 기본값인 32,767 인 경우입니다. 이는 $20/\sqrt{2} = 14.1A_{rms}$ 의 RMS 의 크기에 상당합니다. 모터에 $4A_{rms}$ 의 연속 전류 제한이 있는 경우에 **I2tSet** 는 다음 식으로 설정할 수 있습니다.

$$I2tSet = 32,768 * \frac{4A_{rms}}{14.1A_{rms}} = 9,296$$

16.26A 의 상전류에서 ADC 가 32,768 의 최대 순간값을 통지하는 다이렉트 PWM 앰프에서는 $4A_{rms}$ 의 단속 전류 제한을 가지는 3 상 브러시리스 서보 모터인 경우에 **MaxDac** 은 다음과 같이 계산합니다.

$$MaxDac = 4 * \sqrt{2} * \frac{32,768}{16.26} * \cos(30^\circ) = 9,872$$

이 식의 $\sqrt{2}$ 의 계수는 순간 전류에서 RMS 전류(실효값)로의 변환입니다. $\cos(30^\circ)$ 의 계수는 3 상(abc)에서 2 상(dq)으로의 변환입니다.

Motor[x].I2tTrip 모터의 적분 전류 제한

설명: 적분 전류의 섯다운 제한
범위: 음이 아닌 부동 소수점
단위: (16 비트의 "DAC" 단위)²*sec
기본값: 0.0(Disable)

기존 I 변수명: Ixx58

Power PMAC 의 I²T 적분 전류 제한 기능이 유효한 경우에 **Motor[x].I2tTrip** 은 그 제한이 기능의 최대 적분 전류 제한을 설정합니다(**Motor[x].I2tSet** 가 0 인 경우 I²T 제한 기능은 무효가 됩니다). **Motor[x].I2tSet** 가 0 보다 큰 경우에 Power PMAC 은 명령 또는 실제 전류와 **Motor[x].I2tSet** 의 연속 전류 제한 간의 차를 계산하고 그 결과의 제곱값의 시간 적분을 **Motor[x].I2tTrip** 과 비교합니다. 적분값이 **Motor[x].I2tTrip** 으로 설정한 리미트를 초과하는 경우에 적시에 앰프 이상 신호를 수신한 것처럼 Power PMAC 은 모터를 이상 상태로 합니다. 이 경우에 앰프 이상과 적분 전류 이상의 모터 Status 비트의 양쪽이 설정됩니다.

Motor[x].I2tTrip 제한은 순간 전류 제한(16 비트 DAC 또는 등가 단위에서의 Power PMAC 의 **Motor[x].MaxDac**), 자화 전류(**Motor[x].IdCmd** 에 의한 명령, 보통 유도 모터의 벡터 제어 이외에는 0) 그리고 연속 전류 제한(16 비트 DAC 단위에서 Power PMAC 의 **Motor[x].I2tSet**) 사이에서 관련이 있으며 순간 제한에서의 허용 시간을 곱해서 설정됩니다. 계산식은 다음과 같습니다.

$$I2tTrip = (MaxDac^2 + IdCmd^2 - I2tSet^2) * PermittedTime(sec)$$



주의

Power PMAC 의 I²T 계산은 샘플 간의 경과 시간을 초로 계산하기 위해 **Sys.ServoPeriod** 의 값을 사용합니다. 따라서 **Sys.ServoPeriod** 의 값은 정확한 I²T 계산을 위해 올바른 값이어야 합니다.

개별 모터 상태의 갱신이 실시간 인터럽트마다 실행되지 않도록 저장 설정 요소 **Sys.MotorsPerRtInt** 가 0 이외의 값으로 설정되는 경우(2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능) **Motor[x].I2tTrip** 의 스케일링은 0 인 기본값으로 설정되는 경우 또는 파라미터 **Sys.MotorsPerRtInt** 가 없었던 이전 펌웨어 버전의 경우와 다릅니다.

Sys.MotorsPerRtInt>0 인 경우에 **Motor[x].I2tTrip** 을 수정하기 위해 필요한 시간 확장 계수 N 은 다음 식으로 구합니다.

$$N = \text{ceil}\left(\frac{\text{Sys.ServoMotors} + 1}{\text{Sys.MotorsPerRtInt}}\right)$$

"ceil"은 다음 정수에 올림 처리하는 것(필요한 경우)을 나타내는 "CEILING" 함수입니다. 그리고 상태 요소 **Sys.ServoMotors** 는 유효한 모터의 최대 번호입니다. 실시간 인터럽트마다 실행되는 경우와 비교하여 **Motor[x].I2tTrip** 의 값은 N 으로 나뉘야 합니다.

예

15A_{rms}의 피크 전류 능력을 가지는 토크 모드 앰프와 함께 사용되는 5A_{rms}의 연속 제한과 10A_{rms}의 단속 제한을 가지는 브러시리스 모터가 10A_{rms}에서 최대 허용 시간이 2 초인 경우에 **Motor[x].I2tTrip** 은 다음과 같이 계산합니다.

$$I2tTrip = \left[\left(32,768 * \frac{10}{15} \right)^2 + 0^2 - \left(32768 * \frac{5}{15} \right)^2 \right] * 2 = 7.16 \times 10^8$$

2A/Volt 게인의 사인파 앰프와 함께 사용되는 4A_{rms}의 연속 전류 제한과 12A_{rms}의 단속 전류 제한을 가지는 브러시리스 모터에서 단속 제한 시 최대 허용 시간이 3 초인 경우에 **Motor[x].I2tTrip** 은 다음과 같이 계산합니다.

$$I2tTrip = \left[\left(32,768 * \frac{12}{20} \right)^2 + 0^2 - \left(32768 * \frac{4}{20} \right)^2 \right] * 3 = 1.03 \times 10^9$$

16.26A의 상전류에서 32,768의 순간 최대값을 통지하는 ADC를 가지는 다이렉트 PWM 앰프와 함께 사용되는 4A_{rms}의 연속 전류 제한과 3초간의 8A_{rms}의 단속 전류 제한을 가지는 3상 브러시리스 서보 모터에서는 **I2tTrip** 은 다음과 같이 계산합니다.

$$I2tTrip = \left[\left(8 * \sqrt{2} * \frac{32,768}{16.26} * \cos(30^\circ) \right)^2 - \left(4 * \sqrt{2} * \frac{32,768}{16.26} * \cos(30^\circ) \right)^2 \right] * 3 = 8.77 \times 10^8$$

이 식의 $\sqrt{2}$ 의 계수는 순간 전류에서 RMS 전류로의 변환입니다. $\cos(30^\circ)$ 의 계수는 3상(abc)에서 2상(dq)으로의 변환입니다.

Motor[x].IaBias 모터의 A 상전류 바이어스

설명: A 상전류의 바이어스 오프셋

범위: +/-32,768(부동 소수점)

단위: 부호가 있는 16 비트 입출력의 LSB

기본값: 0.0

기존 I 변수명: Ixx29

Motor[x].IaBias 는 Power PMAC 으로 전류(commutation)되는 모터의 A 상전류의 오프셋 항입니다. 이는 아날로그 오프셋 전위차계의 디지털 등가로 하고자 하는 의도입니다.

Power PMAC 이 모터의 전류 루프를 실행하고 있지 않으면 이는 제 1 상의 명령 전류값으로의 오프셋으로 기능합니다. 출력 레지스터에 쓰여지기 전에 이것은 전류(commutation) 알고리즘으로 계산된 상의 값에 더해집니다.

Power PMAC 이 나아가 모터의 전류 루프를 실행 중이면 이것은 제 1 상의 측정된 전류값에 오프셋으로 기능합니다. 전류 루프 계산 중에서 사용되기 전에 이는 상의 전류 센서에서 AD 컨버터로 생성된 상의 값에 더해집니다.

실제 장치 분해능이 달라도 **Motor[x].IaBias** 는 항상 16 비트의 DAC 또는 ADC 단위가 됩니다. 예를 들어 18 비트 DAC 출력이 상전류 명령에 사용되는 경우에 0.25 인 **Motor[x].IaBias** 값은 DAC 의 1LSB 에相当합니다. 12 비트 ADC 입력이 상전류 측정에 사용되는 경우에 16.0 인 **Motor[x].IaBias** 값은 ADC 의 1LSB 에相当합니다.

Motor[x].CurrentNull 이 0 이외의 값으로 설정되는 경우 **Motor[x].IaBias** 의 값은 "자동 제로 조정" 기능에 의해 모터가 Enable 될 때 자동으로 설정됩니다. 이 경우에 그 저장된 값은 사용되지 않습니다.

Motor[x].IbBias 모터의 B 상전류 바이어스

설명: B 상전류의 바이어스 오프셋

범위: +/-32,768(부동 소수점)

단위: 부호가 있는 16 비트 출력의 LSB

기본값: 0.0

기존 I 변수명: Ixx79

Motor[x].IbBias 는 Power PMAC 으로 전류(commutation)되는 모터의 B 상전류의 오프셋 항입니다. 이는 아날로그 오프셋 전위차계의 디지털 등가로 하고자 하는 의도입니다.

Power PMAC 이 모터의 전류 루프를 실행하고 있지 않으면 이것은 제 2 상의 명령 전류값으로의 오프셋으로 기능합니다. 출력 레지스터에 쓰여지기 전에 이것은 전류(commutation) 알고리즘으로 계산된 상의 값에 더해집니다.

Power PMAC 이 또한 모터의 전류 루프를 실행 중이면 이것은 제 2 상의 측정된 전류값에 오프셋으로 기능합니다. 전류 루프 계산 중에서 사용되기 전에 이는 상의 전류 센서에서 AD 컨버터로 생성된 상의 값에 더해집니다.

실제 장치 분해능이 달라도 **Motor[x].IbBias** 는 항상 16 비트의 DAC 또는 ADC 단위가 됩니다. 예를 들어 18 비트 DAC 출력이 상전류 명령에 사용되는 경우에 0.25 인 **Motor[x].IbBias** 값은 DAC 의 1LSB 에相当합니다. 12 비트 ADC 입력이 상전류 측정에 사용되는 경우에 16.0 인 **Motor[x].IbBias** 값은 ADC 의 1LSB 에相当합니다.

Motor[x].CurrentNull 이 0 이외의 값으로 설정되는 경우 **Motor[x].IbBias** 의 값은 "자동 제로 조정" 기능에 의해 모터가 Enable 될 때 자동으로 설정됩니다. 이 경우에 그 저장된 값은 사용되지 않습니다.

Motor[x].IdCmd 모터의 자화 전류

설명: 명령 자화(다이렉트) 전류

범위: +/-32,768(부동 소수점)

단위: 부호가 있는 16 비트 출력의 LSB

기본값: 0

기준 I 변수명: Ixx77

Motor[x].IdCmd에서는 Power PMAC 으로 전류(commutation)되는 모터의 "다이렉트 전류" 성분의 명령 레벨을 지정합니다. 일반적으로 이 전류 성분을 "자화 전류"라고 부릅니다. 이것이 AC 유도 모터에서의 회전자의 자화 레벨을 설정하기 때문입니다. 이 성분은 회전자의 자계와 평행이며 토크를 생성하는 위치/속도 서보 루프에서의 "직교 전류" 명령 성분에 수직입니다.

Motor[x].IdCmd 는 유도 모터의 적절한 전류(commutation)를 위해 양의 값이어야 합니다. 주어지는 유도 모터의 최적값은 시스템에 좌우되지만 2500 은 대부분 해당 모터에서의 적절한 초기값이 됩니다. 회전자의 자기 포화보다 낮은 레벨에서는 회전자 자계의 강도는 이 값에 비례합니다. 모터의 토크 상수(K_T)와 역기전력 상수(K_E)는 회전자의 자계 강도에 비례합니다.

사용자는 전체 속도 범위에서 모터 성능을 최적화하기 위해 속도의 함수로 이 값을 역동적으로 변경할 수 있습니다.

Motor[x].liGain 모터의 전류 루프 적분 게인

설명:	전류 루프의 적분 게인
범위:	음이 아닌 부동 소수점
단위:	(16 비트 출력의 LSB)/(적분된 16 비트 전류 편차의 LSB)
기본값:	0.001
기존 I 변수명:	lxx61

Motor[x].liGain 은 디지털 전류 루프의 적분 게인항에서 명령과 실제 전류 레벨 간의 차에 곱하며 그 결과를 작동 중인 적분기에 더합니다. 그 적분된 값이 명령 출력에 더해집니다. **Motor[x].pAdc>0** 인 경우만 디지털 전류 루프가 실행되고 이 값도 사용됩니다. **Motor[x].liGain** 의 표준값은 약 0.2 입니다.

Motor[x].liGain 은 **Motor[x].IpfGain** 포워드 패스 비례 게인 또는 **Motor[x].IpbGain** 백 패스 비례 게인 중 하나와 함께 사용할 수 있습니다. **Motor[x].IpfGain** 과 함께 사용되는 경우 **Motor[x].IpfGain** 이 신속한 응답을 제공하고 **Motor[x].liGain** 은 바이어스를 보정할 때만 필요해지므로 이 값은 상당히 작아집니다. **Motor[x].IpbGain** 과 함께 사용되는 경우 **Motor[x].liGain** 은 명령 변화에 직접 응답하는 유일한 게인이 됩니다. 그러므로 더 신속하게 응답하게 하기 위해서는 이 게인을 상당히 높게 설정해야 합니다.

Motor[x].InPosBand 모터의 인포지션 밴드

설명:	인포지션 최소치
범위:	음이 아닌 부동 소수점
단위:	모터 단위
기본값:	0.0
기존 I 변수명:	lxx28

Motor[x].InPosBand 에서는 명령 동작을 실행하지 않을 때의 모터가 "인포지션" 상태라고 판단되는 최대 위치 편차의 크기를 지정합니다.

기술적으로는 모터가 "인포지션"이라고 판단되기 위해서는 5 가지 조건이 성립해야 합니다.

- 1.) 모터는 클로즈드 루프 제어 상태이어야 합니다.
- 2.) 모터 명령 속도는 0 이어야 합니다.
- 3.) 모터는 동작 또는 한정 시간의 드웰을 실행해서는 안 됩니다.
- 4.) 편차의 크기가 이 파라미터 이하이어야 합니다.
- 5.) 위의 4 가지 상태가 모두 연속하는 서보 사이클(**Motor[x].InPosTime** + 1)에서 성립해야 합니다.

모든 유효한 모터에 대해 Power PMAC 은 모터가 "인포지션"인지 여부를 서보 사이클마다 자동으로 판정합니다.

모터가 "인포지션" 상태인 경우에 데이터 구조체 요소 **Motor[x].InPos** 는 1 로 설정됩니다. 그렇지 않으면 0 으로 설정됩니다.

Motor[x].InPosTime 모터의 인포지션 스캔 횟수

설명: 인포지션의 연속 스캔 횟수

범위: 0 ..255

단위: 서보 사이클 수

기본값: 0

기존 I 변수명: lxx88

Motor[x].InPosTime 은 모터가 "인포지션" 상태로 판단되기 위해 인포지션 상태가 참이어야만 하는 연속하는 반복 횟수를 지정합니다. 기본값 0 에서는 1 회째에서 인포지션 상태(클로즈드 루프, 명령 속도 0, **Motor[x].InPosBand** 이하의 편차)의 조건이 성립하면 인포지션 비트가 설정됩니다.

Motor[x].InPosTime 이 0 보다 큰 경우에 해당 상태가 또한 연속 확인이 지정된 추가 횟수의 기간에 계속 참이어야 합니다. Power PMAC 은 모든 유효한 모터에 대해 서보 사이클마다 해당 조건을 확인합니다. 해당 추가 확인의 단 1 회라도 해당 조건 중 하나가 참이 아님을 검출한 경우에 해당 카운트는 0 부터 다시 실행합니다.

이 파라미터는 모터가 종료 위치에서 정말로 안정되었고 다음 운전을 실행하기 전에 인포지션 밴드 내외에서 진동하지 않는다고 사용자가 확인할 수 있습니다.

Motor[x].InvAmax 모터의 최대 프로그램 가속도

설명: 최대 프로그램 가속도의 역수

범위: 음이 아닌 부동 소수점

단위: msec²/모터 단위

기본값: 10.0(= 0.1 모터 단위/msec²)

기존 I 변수명: lxx17

Motor[x].InvAmax 는 프로그램 동작의 몇 가지 타입에서의 가속도의 최대 크기를 설정합니다. 세그먼트 무효(**Coord[x].SegMoveTime** = 0)인 직선 보간 모드에서는 **Ta** 와 **Ts** 파라미터에서 계산되는 피크 명령 가속도의 크기와 모터 속도의 변화에 대해 동작 계산 시에 확인됩니다. 이는 정지 상태에서의 초기 가속과 두 동작 명령 간의 블렌드 가속 또는 감속에 적용됩니다. **Motor[x].InvAmax** 는 연속 동작 종료 시의 정지로의 최종 감속에는 사용되지 않습니다. **Motor[x].InvDmax** 가 대신 사용됩니다.



직선 보간 모드에서는 0 이외의 가속 시간을 가져야 합니다(**Ta** 또는 **Ts**>0.0). 이 확인은 동작 계산 시 수행됩니다. 이 제한과 비교하여 확인되는 것은 가속 부분입니다.

프로그램의 직선 보간 동작이 이 모터의 더 높은 가속도의 크기를 명령하는 경우에 이 제한을 위반하지 않도록 충분한 시간만큼 가속 시간을 연장합니다. 또한 협조와 궤적이 유지되도록 비례하여 좌표계의 다른 모터의 가속도를 낮춥니다(해당 모터가 자체 제한을 위반하지 않은 경우라도).

세그먼트화 된 룩 어헤드 알고리즘이 좌표계에서 유효한 경우에 **Motor[x].InvAmax** 는 나아가 세그먼트별 직선 보간, 원호 보간과 **pvt** 모드 동작의 가속도의 최대값을 설정합니다. **Coord[x].SegMoveTime** 과 **Coord[x].LHDistance** 는 이 가속도 제한이 기능하기 위해 0 보다 커야 합니다. 명령 세그먼트의 가속도 크기가 이 크기를 초과하는 경우에 이 제한을 위반하지 않도록 세그먼트 시간이 충분한 시간만큼만 연장됩니다. 이 감속된 속도로의 감속도가 가속도 제한 내에 수행되도록 그 이전의 세그먼트 시간도 연장해야 하는 경우에 이도 자동으로 수행됩니다.

좌표계가 세그먼트 모드(**Coord[x].SegMoveTime**>0)이지만 룩 어헤드 버퍼가 유효가 아닌 경우(**Coord[x].LHDistance** = 0 또는 룩 어헤드 버퍼가 정의되지 않음) **Motor[x].InvAmax** 는 사용되지 않습니다.

Motor[x].InvAmax 는 최대 가속도 크기의 역수로 msec²/모터 단위(보통 원시 카운트)의 기본 단위로 표시된다는 점에 주의하십시오. 이는 Power PMAC 의 더 효율적인 계산을 가능하게 합니다. 0.0 의 값은 가속도 제한을 실행하지 않습니다.

예

mm 모터 단위에서 8m/s²(약 0.8G)의 최대 가속도를 설정하기 위한 식은 다음과 같습니다.

$$InvAmax = \frac{s^2}{8m} * \frac{10^{-3}m}{motor\_unit} * \frac{10^6 ms^2}{s^2} = 125 \left(\frac{ms^2}{motor\_unit} \right)$$

Motor[x].InvDmax 모터의 최대 프로그램 감속도

설명: 최대 프로그램 감속도의 역수

범위: 음이 아닌 부동 소수점

단위: msec²/모터 단위

기본값: 10.0(= 0.1 모터 단위/msec²)

좌표계가 세그먼트 모드가 아닐 때(**Coord[x].SegMoveTime** = 0), **Motor[x].InvDmax** 에서는 직선 보간 모드 정지까지의 최종 감속도의 최대값을 설정합니다(연속 동작 시퀀스의 종료). **Td** 와 **Ts** 의 파라미터에서 계산되는 피크의 명령 최종 감속도 크기와 모터 속도의 변화에 대해 동작 계산 시 확인됩니다.



직선 보간 모드에서는 0 이외의 감속 시간을 가져야 합니다(**Td** 또는 **Ts**>0.0). 이 확인은 동작 계산 시 수행됩니다. 이 제한과 비교하여 확인되는 것은 감속 부분입니다.

프로그램의 직선 보간 동작이 이 모터의 더 높은 최종 감속도의 크기를 명령하는 경우에 이 제한을 위반하지 않도록 충분한 시간만큼만 최종 감속 시간을 연장합니다. 또한 협조와 궤적이 유지되도록 비례하여 좌표계의 다른 모터의 최종 감속도를 낮춥니다(해당 모터가 자체 제한을 위반하지 않은 경우라도).

연속하는 직선 보간 모드 이동 간의 블렌드의 가속도는 결과적으로 감속되었다고 해도 **Motor[x].InvAmax** 로 제한됩니다. 룩 어헤드 버퍼에서는 모든 가속도와 감속도의 크기는 **Motor[x].InvAmax** 에 의해 관리됩니다. 그리고 **Motor.InvDmax** 는 사용되지 않습니다.

Motor[x].InvDmax 는 최대 감속도 크기의 역수로 msec²/모터 단위(보통 원시 카운트)의 기본 단위로 표시된다는 점에 주의하십시오. 이는 Power PMAC 의 더 효율적인 계산을 가능하게 합니다. 0.0 의 값은 감속도 제한을 실행하지 않습니다.

예

미크론 모터 단위에서 5m/s²(약 0.5G)의 최대 감속도를 설정하기 위한 식은 다음과 같습니다.

$$InvDmax = \frac{s^2}{5m} * \frac{10^{-6}m}{motor\_unit} * \frac{10^6 ms^2}{s^2} = 0.2 \left(\frac{ms^2}{motor\_unit} \right)$$

Motor[x].InvJmax 모터의 최대 프로그램 저크

설명: 최대 프로그램 저크의 역수

범위: 음이 아닌 부동 소수점

단위: msec³/모터 단위

기본값: 50(= 0.02 모터 단위/msec³)

기존 I 변수명: lxx18

좌표계가 세그먼트 모드가 아닐 때(**Coord[x].SegMoveTime** = 0) **Motor[x].InvDmax**에서는 직선 보간 모드 동작에서의 "저크"(가속도의 변화율)의 최대 크기를 설정합니다. **Ta**, **Td** 와 **Ts** 파라미터에서 계산되는 피크의 저크 크기와 모터 속도의 변화에 대해 동작 계산 시 확인됩니다.



직선 보간 모드에서는 0 이외의 S 자 곡선 시간을 가져야 합니다($T_s > 0.0$). 이 확인은 동작 계산 시 수행됩니다. 이 제한과 비교하여 확인되는 것은 S 자 곡선 부분입니다.

프로그램의 직선 보간 모드 동작이 이 모터의 더 높은 저크의 크기를 명령하는 경우에 이 제한을 위반하지 않도록 충분한 시간만큼 가속의 "S 자 곡선" 부분의 시간을 연장합니다. 또한 협조와 궤적이 유지되도록 비례하여 좌표계의 다른 모터의 저크를 낮춥니다(해당 모터가 자체 제한을 위반하지 않은 경우라도).

Motor[x].InvJmax 는 최대 저크 크기의 역수로 msec³/모터 단위(보통 원시 카운트)의 기본 단위로 표시된다는 점에 주의하십시오. 이는 Power PMAC 의 더 효율적인 계산을 가능하게 합니다. 0.0 의 값은 저크 제한을 실행하지 않습니다.

예

mm 모터 단위에서 100m/s³(약 10G/sec)의 최대 저크를 설정하기 위한 식은 다음과 같습니다.

$$InvJmax = \frac{s^3}{100m} * \frac{10^{-3}m}{motor\_unit} * \frac{10^9 ms^3}{s^3} = 10,000 \left(\frac{ms^3}{motor\_unit} \right)$$

Motor[x].liGain 모터의 전류 루프 백 패스 비례 게인

설명: 전류 루프의 백 패스 비례 게인

범위: 음이 아닌 부동 소수점

단위: (16 비트 출력의 LSB)/(16 비트 실제 전류의 LSB)

기본값: 1.0

기존 I 변수명: lxx76

Motor[x].IpbGain 은 디지털 전류 루프의 "백 패스"에 해당하는 비례 게인항에서 실제 전류 레벨에 곱하고 명령 출력에서 그 결과를 뺍니다. 전류 루프를 실행하기 위해 **Motor[x].IpbGain** 또는 **Motor[x].IpfGain**(포워드 패스 비례 게인) 중 하나를 사용해야 합니다. 보통 해당 비례 게인항의 하나만 사용되지만 양쪽이 사용되어도 상관 없습니다. **Motor[x].IpbGain** 의 표준값은 약 0.9 입니다.

Motor[x].IpbGain 이 유일한 비례 게인항으로 사용되는 경우에 **Motor[x].liGain** 의 적분 게인항만 명령 변화에 직접 응답합니다. 적분 동작은 명령상의 로 패스 필터로 작용하고 많은 노이즈를 제거하지만 실제 변화에 대한 응답성은 저하시킵니다. 위치/속도 루프 서보에서의 명령값에 높은 노이즈 레벨이 있고(보통 낮은 위치 분해능에 의한) 실제 전류 측정에는 낮은 노이즈 레벨이 있는 경우에 일반적으로 **Motor[x].IpbGain** 만 사용됩니다.

Motor[x].pAdc > 0 에서 디지털 전류 루프가 실행된 경우만 **Motor[x].IpbGain** 이 사용됩니다.

Motor[x].IpfGain 모터의 전류 루프 포워드 패스 비례 게인

설명:	전류 루프의 포워드 패스 비례 게인
범위:	음이 아닌 부동 소수점
단위:	(16 비트 출력의 LSB)/(16 비트 전류 편차의 LSB)
기본값:	1.0
기준 I 변수명:	Ix62

Motor[x].IpfGain 은 디지털 전류 루프의 "포워드 패스"에 해당하는 비례 게인항에서 명령과 실제 전류의 레벨 간의 차에 곱하고 그 결과가 명령 출력이 됩니다. 전류 루프를 실행하기 위해 **Motor[x].IpfGain** 또는 **Motor[x].IpbGain**(백 패스 비례 게인) 중 하나를 사용해야 합니다. 보통 해당 비례 게인항의 하나만 사용되지만 양쪽이 사용되어도 상관 없습니다. **Motor[x].IpfGain** 의 표준값은 약 0.9 입니다.

Motor[x].IpfGain 은 **Motor[x].IpbGain** 보다 위치/속도 서보 루프에서의 명령 변화에 대해 고응답성과 더 높은 전류 루프의 대역 폭이 있습니다. 단, 명령값에 노이즈가 많은 경우에 그것이 저분해능의 위치 센서인 경우에 일어날 수 있지만 그 대신 **Motor[x].IpbGain** 을 사용하면 더 효과가 있는 노이즈 필터링을 실현할 수 있습니다.

Motor[x].pAdc>0 에서 디지털 전류 루프가 실행된 경우만 **Motor[x].IpfGain** 이 사용됩니다.

Motor[x].IxCoupleGain 모터의 전류 루프 크로스 커플링 게인

설명:	전류 루프의 크로스 커플링 게인
범위:	음이 아닌 부동 소수점
단위:	(16 비트 출력의 LSB)/(16 비트 전류의 LSB*속도)
기본값:	0.0
기준 I 변수명:	Ixx89

Motor[x].IxCoupleGain 은 다이렉트 전류 루프와 직교 전류 루프 간의 크로스 커플링항입니다. 모터 권선의 인덕턴스를 위해 주파수가 높아지면 권선의 전압은 전류를 점점 더 많이 흘려보내야 합니다. 이는 수직 관계인 다이렉트 전류 루프와 직교 전류 루프 사이에 크로스 커플링(교차 결합) 효과를 생성하기 때문입니다. 이는 제로와 저주파수 영역에서는 실질적으로 독립됩니다.

이 항이 없으면 더 높은 주파수에서 필요한 전류를 얻기 위해서 즉 필요한 크로스 커플링의 전압을 제공하기 위해서 각 루프 내의 전류 루프의 적분 게인항에 의한 "차지업"으로 수행합니다. 이는 최종적으로는 기능하지만 원래 "편차에 의한 구동"이므로 보통 직접적인 처리가 아닙니다.

Motor[x].IxCoupleGain 은 이 효과를 보상하는 직접적인 방법을 제공합니다. 보통 급가속도 또는 급감속도의 고속(일반 10,000rpm 이상) 동작인 경우에만 사용됩니다. 그렇지 않으면 보통은 표준 전류 루프 게인만으로 충분합니다.

Motor[x].IxCoupleGain 은 직교 전류의 측정값과 전류(commutation) 주파수에 곱해지고 그 결과가 다이렉트 전압 명령값 오프셋이 됩니다. 또한 다이렉트 전류의 측정값과 전류(commutation) 주파수에 곱해지고 그 결과가 직교 전압 명령값 오프셋이 됩니다.

Motor[x].JogOffset 모터의 JOG 오프셋

설명: 트리거 포함 JOG 동작의 가변 트리거 후 오프셋

범위: 부동 소수점

단위: 모터 단위

기본값: 0.0

Motor[x].JogOffset 은 가변 트리거 후 거리(^*로 종료하는 명령)을 가지는 트리거 포함 JOG 동작의 부호가 있는 트리거 후 오프셋 거리를 지정합니다. 이 오프셋은 트리거 후 이동 부분에서 명령된 종료 위치와 트리거 조건이 성립한 실제 위치 간의 차가 됩니다.

Motor[x].JogSpeed 모터의 JOG 속도

설명: JOG 명령의 속도

범위: 양의 부동 소수점

단위: 모터 단위/msec

기본값: 32.0

기존 I 변수명: Ixx22

Motor[x].JogSpeed 는 모터의 JOG 이동 또는 프로그램의 **RAPID** 모드 이동 (**Motor[x].RapidSpeedSel** =0 인 경우)의 최고 명령 속도의 크기를 설정합니다. 이동 방향 즉 속도의 부호는 그 이동 명령 자체에서 제어됩니다.

이 파라미터의 변경은 다음 이동 명령까지 반영되지 않습니다. 예를 들어 동작 중에 JOG 속도를 변경하고자 하는 경우에 JOG 이동을 시작한 후 이 파라미터를 변경하고 다음으로 새로운 JOG 명령을 전송하게 됩니다.

예

5 미크론 모터 단위에서 200mm/sec(약 8inch/sec)의 최고 JOG 속도 크기로 설정하기 위한 식은 다음과 같습니다.

$$HomeVel = \frac{200mm}{sec} * \frac{motor\_unit}{5*10^{-3}mm} * \frac{sec}{10^3ms} = 40 \left(\frac{motor\_unit}{ms} \right)$$

Motor[x].JogTa 모터의 JOG 가/감속 시간

설명:	조그 가/감속 시간 또는 레이트의 역수
범위:	부동 소수점
단위:	msec(0 이상인 경우), msec ² /모터 단위(0 미만인 경우)
기본값:	10(= 0.1 모터 단위/msec ²)
기존 I 변수명:	lxx20

Motor[x].JogTa 는 JOG, 원점 서치 또는 프로그램의 **RAPID** 모드 이동에서의 기동, 정지, 속도 변경할 때의 가속 시간 또는 가속도를 설정합니다. 값이 0 이상인 경우에 밀리초 단위로 가속 시간을 지정합니다. 값이 0 미만인 경우에 msec²/모터 단위(일반 원시 카운트) 단위로 가속도의 최대값의 역수를 지정합니다.

이동 중인 동작에 매끄럽게 인터럽트하고자 하는 경우에 보통 가속도의 레이트를 지정하는 것이 효과적입니다. 여러 개의 모터에서 JOG 동작의 협조를 수행하고자 하는 경우에 보통 가속 시간을 지정하는 것이 효과적입니다.

Motor[x].JogTa 로 시간을 지정하고 **Motor[x].JogTs**(S 자 곡선 시간)로 더 작은 시간을 지정하는 경우에 가속에서 소비되는 합계 시간은 **JogTa** 와 **JogTs** 의 합과 같아집니다(이 규칙이 Turbo PMAC 과는 다르다는 점에 주의하십시오). 단, **Motor[x].JogTa** 로 시간을 지정하고 **Motor[x].JogTs** 로 더 큰 시간을 지정하는 경우에 가속에서 소비되는 합계 시간은 **Motor[x].JogTs** 의 2 배가 됩니다. 따라서 **Motor[x].JogTa** 가 0 인 경우에 **Motor[x].JogTs** 만으로 "순수한" S 자 곡선의 가속 시간을 제어합니다.

Motor[x].JogTs 가 0 미만으로 최대 "저크"가 지정되는 경우에 **Motor[x].JogTa** 도 0 미만으로 최대 가속도가 지정되어야 합니다.

이 파라미터의 변경은 다음 이동 명령까지 반영되지 않습니다. 예를 들어 **JOG** 이동에서 가속 시간과는 다른 감속 시간을 원하는 경우에 가속 시간을 지정한 다음 **JOG** 를 명령하고 감속 시간을 변경한 후에 다시 **JOG** 이동(예를 들어 **j=**)을 명령하거나 적어도 **JOG** 종료(**j!**)를 명령해야 합니다.

예

mm 모터 단위에서 5m/s²(약 0.5G)의 JOG 가속률을 설정하기 위한 식은 다음과 같습니다.

$$JogTa = -\frac{s^2}{5m} * \frac{10^{-3}m}{motor\_unit} * \frac{10^6 ms^2}{s^2} = -200 \left(\frac{ms^2}{motor\_unit} \right)$$

Motor[x].JogTs 모터의 JOG S 자 곡선 시간

설명:	JOG 가/감속 S 자 곡선 시간 또는 저크 레이트의 역수
범위:	부동 소수점
단위:	msec(0 이상인 경우) 또는 msec ³ /모터 단위(0 미만인 경우)
기본값:	-50(= 0.02 모터 단위/msec ³)
기준 I 변수명:	lxx21

Motor[x].JogTs 는 JOG, 원점 서치 또는 프로그램 **RAPID** 모드 이동에서의 기동, 정지, 속도 변경 시의 가속 S 자 곡선 부분의 시간을 설정하거나 해당 가속 중의 "저크"(가속도의 변화율)의 최대값을 설정합니다. 값이 0 이상인 경우에 msec 에서 가속의 각 S 자 곡선 부분의 시간을 지정합니다. 값이 0 미만인 경우에 msec³/모터 단위(일반 원시 카운트)에서 저크의 최대값의 역수를 지정합니다.

이동 중인 동작에 매끄럽게 인터럽트하고자 하는 경우에 저크의 레이트를 지정하는 것이 일반 효과적입니다. 여러 개의 모터에서의 JOG 동작의 협조를 수행하고자 하는 경우에 S 자 곡선 시간을 지정하는 것이 일반 효과적입니다.

Motor[x].JogTs 가 0 미만으로 최대 "저크"를 지정한 경우에 **Motor[x].JogTa** 도 0 미만으로 최대 가속도를 지정해야 합니다. **Motor[x].JogTs** 가 0.0 으로 설정되는 경우에 S 자 곡선 가속 부분은 없습니다. 그리고 JOG 가속도와 감속도 레이트는 일정해집니다.

Motor[x].JogTa 로 시간을 지정하고 **Motor[x].JogTs**(S 자 곡선 시간)로 더 작은 시간을 지정하는 경우에 가속에서 소비되는 합계 시간은 **JogTa** 와 **JogTs** 의 합과 같아집니다(이 규칙이 Turbo PMAC 과 다르다 점에 주의하십시오). 단, **Motor[x].JogTa** 로 시간을 지정하고 **Motor[x].JogTs** 로 더 큰 시간을 지정하는 경우에 가속에서 소비되는 합계 시간은 **Motor[x].JogTs** 의 2 배가 됩니다. 따라서 **Motor[x].JogTa** 가 0 인 경우에 **Motor[x].JogTs** 만으로 "순수한" S 자 곡선의 가속 시간을 제어합니다.

이 파라미터의 변경은 다음 이동 명령까지 반영되지 않습니다. 예를 들어 JOG 이동에서 가속 시간과는 다른 감속 시간을 원하는 경우에 가속 시간을 지정한 다음 JOG 를 명령하고 감속 시간을 변경한 후에 다시 JOG 이동(예를 들어 j=)을 명령하거나 적어도 JOG 종료(j/)를 명령해야 합니다.

예

mm 모터 단위에서 80m/s³(약 8G/sec)의 JOG 저크를 설정하기 위한 식은 다음과 같습니다.

$$JogTs = -\frac{s^3}{80m} * \frac{10^{-3}m}{motor\_unit} * \frac{10^9 ms^3}{s^3} = -12,500 \left(\frac{ms^3}{motor\_unit} \right)$$

Motor[x].LimitBits 모터의 리미트 신호 비트 수

설명:	pLimits 레지스터에서 제 1 리미트 신호의 비트 수
범위:	0 ..30
단위:	비트 수(리틀 엔디언)
기본값:	하드웨어에 따라서 자동으로 구성됩니다.

Motor[x].LimitBits 는 **Motor[x].pAmpFault** 에서 주소 지정되는 32 비트 레지스터의 어느 비트가 Power PMAC 에서 모터의 하드웨어 오버트래블 리미트 입력의 검출에 사용되는지 결정합니다. **Motor[x].LimitBits** 의 값은 양의 리미트 플래그 입력의 비트 번호를 지정합니다. PMAC 은 그 다음 고위의 비트를 음의 리미트 플래그 입력에 사용합니다. 비트 번호는 "리틀 엔디언" 규약으로 지정됩니다. 그때 비트 n 은 32 비트 워드의 2^n 의 값이 됩니다.

대다수 어플리케이션에서 이 값은 재초기화(\$\$\$** 명령)로 수행되는 하드웨어 자동 식별과 일부 설정에서 적절하게 지정됩니다. 이는 공장에서 또는 사용자에게 의한 시스템 어셈블리로 또는 그 후에 수행됩니다. 그리고 **Motor[x].EncType** 의 값이 Power PMAC 스크립트 환경을 통해 또는 온라인 명령, 버퍼 프로그램 명령 또는 전원 투입/리셋에서 저장된 값으로 설정될 때 **Motor[x].EncType** 의 값으로 지정된 하드웨어 인터페이스(예를 들어 PMAC2 스타일, PMAC3 스타일, MACRO) 타입에서 이 값은 표준값으로 설정됩니다.

ACC-24E2, ACC-24E2A 또는 ACC-24E2S 에서는 PMAC2 스타일 "DSPGATE1" 서보 IC 의 표준 오버트래블 리미트 입력 신호를 사용하기 위해 **Motor[x].LimitBits** 는 25 로 설정됩니다. (IC 의 24 비트 워드의 비트 17 에 신호가 매핑되지만 이 워드는 32 비트 Power PMAC 버스의 상위 24 비트가 됩니다. 따라서 이는 소프트웨어에서는 비트 25 로 표현된다는 점에 주의하십시오.)

ACC-24E3 또는 Power PMAC "Brick"에서는 PMAC3 스타일 "DSPGATE3" IC 의 표준 오버트래블 리미트 입력 신호를 사용하기 위해서 **Motor[x].LimitBits** 는 9 로 설정됩니다.

PMAC2 스타일 MACRO IC(ACC-5E 에서) 또는 PMAC3 스타일 IC(ACC-5E3 에서)를 통해 MACRO 링 프로토콜의 표준 오버트래블 리미트 비트를 사용하기 위해서 **Motor[x].AmpFaultBit** 는 25 로 설정됩니다.

Motor[x].MasterCtrl 모터의 위치 추종 제어 플래그

설명:	위치 추종을 유효로 하는 제어 플래그
범위:	0 ..15
단위:	비트 필드
기본값:	0
기존 I 변수명:	lxx06

Motor[x].MasterCtrl 은 위치 추종(전자 기어) 기능이 지정된 모터에 유효가 되었는지 여부 및 그 기능이 어떻게 실행되는지를 결정합니다. 그것은 4 비트값으로 구성되지만 현재 2 비트만 사용됩니다.

비트 0(값 1)에서는 위치 추종의 유효/무효를 제어합니다. 만약 1 로 설정되는 경우에 모터는 **Motor[x].MasterPosSf** 로 설정된 기어비에 따라서 **Motor[x].pMasterEnc** 로 주소 지정된 레지스터 값의 변화를 추적합니다. 만약 0 으로 설정되는 경우에 위치 추종 기능은 무효가 됩니다. 그리고 주소 지정된 마스터 레지스터의 변화를 추적하지 않습니다.

비트 1(값 2)에서는 "일반" 또는 "오프셋" 모드 중 어느 쪽에서 추종이 수행되는지 제어합니다. 0 으로 설정되는 경우에 일반 모드가 지정되고 모터가 마스터 위치를 추종할 때 프로그램 동작의 기준 위치는 변화하지 않습니다. 이는 이어서 일어나는 프로그램 동작이 추종 기능에 의한 변화를 취소한다는 것을 의미합니다.

비트 1 이 1 로 설정되는 경우에 오프셋 모드가 지정되고 프로그램 동작의 기준 위치는 추종 동작과 함께 변화합니다. 이 모드에서는 이어서 일어나는 프로그램 동작은 추종 기능에 의한 위치 변화에 추가됩니다. 프로그램 동작 및 추종 동작의 중첩이 가능합니다.

이어서 일어나는 프로그램 동작이 어떻게 계산되는지에 영향을 주므로 추종이 무효일 때도 추종 모드 비트가 중요하다는 점에 주의하십시오. 추종 모드가 변경되는 경우에만 다음 프로그램 동작이 계산되기 전에 **pmatch** 위치 적합 명령을 실행해야 합니다. 그렇지 않으면 그 동작은 시작 위치로 잘못된 값을 사용하게 되고 동작 처음에 위험한 점프 동작이 발생할 수 있습니다. 모션 프로그램이 시작되는 경우 **pmatch** 명령은 항상 자동으로 실행됩니다. 그러나 이 변경이 모션 프로그램 도중에 수행되는 경우에 이 명령을 명시적으로 명령해야 합니다.

비트 2 와 3 은 추후 사용하기 위해 예약됩니다.

Motor[x].MasterCtrl 은 풀 워드 요소 **Motor[x].Control[0]**의 비트 20-23 을 구성합니다.

Motor[x].MasterMaxAccel 모터의 위치 추종 최대 가속도

설명:	위치 추종의 최대 가속도
범위:	음이 아닌 부동 소수점
단위:	모터 단위/(서보 사이클*서보 사이클)
기본값:	0.0(가속도 제한은 무효)

Motor[x].MasterMaxAccel 은 임의의 서보 사이클에서의 위치 추종에서 발생하는 모터 속도의 최대 변화 크기의 지정 즉 위치 추종에서 발생하는 최대 가속도를 설정합니다. 또한 추종 속도 제한에서 발생하는 모든 "과잉" 추종의 폐기 또는 유지 여부도 제어합니다. **Motor[x].MasterMaxSpeed** 가 0.0 보다 큰 값으로 설정되는 경우에만 위치 추종에서의 속도 제한을 유효로 합니다.

Motor[x].MasterMaxAccel 이 기본값 0.0 으로 설정되는 경우에 이 가속도 제한은 무효가 되고 모터 명령 가속도에 위치 추종이 전적으로 영향을 줍니다. 또한 추종 속도 제한이 **Motor[x].MasterMaxSpeed** 에서 유효인 경우에 무제한 명령과 실제로 사용되는 제한 속도 간의 추종 거리의 모든 차가 버려집니다. 그 결과 마스터와의 위치 동기를 잃게 됩니다.

Motor[x].MasterMaxAccel 이 0.0 보다 큰 값으로 설정되는 경우에 가속도 제한이 유효가 되고 각 서보 사이클에서 이전 서보 사이클에서의 추종 크기의 변화는 마스터의 변화와 유효한 추종 비율로 연산되어 **Motor[x].MasterMaxAccel** 과 비교됩니다. 추종 변화의 크기가 이 제한보다 큰 경우에 제한값(추종 방향으로)이 대신 사용되고 이에 따라 추종에서 발생하는 가속도를 클램프합니다.

추종 가속도 제한이 유효인 경우에 무제한 명령과 실제로 사용되는 제한 속도 및/또는 제한 가속도 간의 추종 거리의 모든 차는 유지 및 누적됩니다. 그 결과 추종이 제한되지 않으면 "릴리스" 할 수 있습니다. 누적된 과잉을 릴리스하는 처리는 최대 추종 속도 및 최대 추종 가속도의 위반을 일으키지 않는 최대 속도로 진행됩니다. 과잉이 완전하게 릴리스되면 모터와 해당 마스터 간의 위치 동기가 복원됩니다.

제한에서의 과잉이 누적되었으므로 Power PMAC 은 단지 그 순간의 속도와 가속도를 고려할 뿐만 아니라 지정 제한 내에서 추종 프로파일을 정지 상태로 가져가는 속도와 가속도 프로파일도 고려합니다. 이는 각 서보 사이클에서 이 이상의 마스터 위치 변화는 발생하지 않는다고 가정하면서 수행됩니다. 이는 추종 속도와 가속도를 지정된 제한보다 낮은 값으로 제한시킬 수 있습니다.

Motor[x].MasterMaxSpeed 모터의 위치 추종 최고 속도

설명: 위치 추종의 최고 속도
범위: 음이 아닌 부동 소수점
단위: 모터 단위/서보 사이클
기본값: 0.0(속도 제한이 무효)

Motor[x].MasterMaxSpeed 는 임의의 서보 사이클에서의 위치 추종에서 발생하는 모터 위치의 최대 변화 크기를 지정 즉 위치 추종에서 발생하는 최대 속도를 설정합니다. **Motor[x].MasterMaxSpeed** 가 기본값 0.0 으로 설정되는 경우 속도 제한은 무효가 됩니다. 그리고 모터 명령 속도에 위치 추종이 전적으로 영향을 줍니다.

Motor[x].MasterMaxSpeed 가 0.0 보다 큰 값으로 설정되는 경우에 속도 제한이 유효가 되고 각 서보 사이클에서 추종 크기는 마스터의 변화와 유효한 추종 비율로 연산되어 **Motor[x].MasterMaxSpeed** 와 비교됩니다. 추종의 크기가 이 제한보다 큰 경우에 제한값(추종 방향으로)이 대신 사용되고 그에 따라 추종에서 발생하는 속도를 클램프합니다.

Motor[x].MasterMaxAccel 의 설정에 따라 추종 "과잉"은 폐기될 수도 있습니다. 그 결과 추종 속도가 제한되었으면 마스터와의 위치 동기를 잃게 되지만 마스터가 정지하면 나머지 모션은 없어집니다. 또는 추종 "과잉"은 유지될 수도 있습니다. 추종 속도가 더는 제한되지 않으면 마스터와의 완전한 위치 동기를 복원하도록 "릴리스됩니다".

모터의 궤적 명령 동작(조그 동작 또는 프로그램에서)이 위치 추종 상에 중합된 경우에 추종에서 발생하는 모터 모션의 성분만 이 기능에 의해 제한됩니다.

Motor[x].MasterPosSf 모터의 명령 마스터 위치 스케일 계수

설명: 명령 마스터 위치의 스케일 계수

범위: 부동 소수점

단위: 모터 단위/소스 데이터 단위

기본값: 1.0

기준 I 변수명: lxx07

Motor[x].MasterPosSf 는 마스터 위치가 **Motor[x].pMasterEnc** 로 지정된 레지스터에서 읽으며 곱해지는 명령 스케일 계수를 지정합니다. 이 곱셈 결과는 모터의 위치 추종 계산에서의 마스터값으로 사용됩니다. 이는 위치 추종 기능의 "기어비"를 실질적으로 설정하는 것입니다.

위치 데이터의 소스는 대부분의 경우 엔코더 변환 테이블(ECT) 항목의 결과 레지스터가 됩니다. 각각의 항목은 그 자체의 출력 스케일 계수를 가집니다. 따라서 처리의 각 스텝에서 각각 다른 단위를 사용하는 등 많은 유연성이 있습니다. 그러나 가장 일반적인 방법은 ECT 항목의 부동 소수점 출력이 카운터의 "카운트값" 이거나 데이터 워드나 A/D 컨버터의 LSB 로 스케일링되는 것입니다. 이 경우에 몇 가지 보간 기법 예를 들어 소수점의 분해능이 있었다고 해도 LSB 에서 스케일링됩니다.

이 요소는 소스 마스터 데이터의 증가분 단위에 대해 그 모터에 필요한 이동량의 모터 단위수로 설정합니다. 예를 들어 소스 마스터 데이터가 마스터 엔코더 카운터의 카운트값이며 모터는 피드백 엔코더 카운터의 카운트 단위인 경우입니다. 그리고 마스터의 각 카운트에서 모터를 5 단위 분량 동작시키고 싶으면 **Motor[x].MasterPosSf** 를 5.0 으로 설정합니다.

저장 설정 요소 **Motor[x].SlewMasterPosSf** 가 기본값 0.0 으로 설정되는 경우에 비율의 슬루 레이트 제어는 무효가 되고 **Motor[x].MasterPosSf** 의 값의 변화는 즉시 일어납니다. 그러나 **Motor[x].SlewMasterPosSf** 가 양의 값으로 설정되는 경우에 각 서보 사이클에서 사용되는 실제 비율(상태 요소 **Motor[x].ActiveMasterPosSf** 에서 확인 가능)은 서보 사이클마다 **Motor[x].SlewMasterPosSf** 의 크기만 변화합니다.

Motor[x].MaxDac 모터의 최대 DAC 출력

설명: 순간 서보 출력 제한

범위: 0 ..32,767.999

단위: 16 비트 DAC 와 등가

기본값: 28,000

기준 I 변수명: lxx69

Motor[x].MaxDac 은 모터의 위치/속도 서보 루프에서의 순간 최대 명령의 크기(절대값)를 지정합니다. 서보 알고리즘이 서보 사이클에서 더 큰 크기의 명령 출력값을 연산하는 경우에 사용되는 명령의 크기는 이 값으로 제한됩니다. **Motor[x].MaxDac** 은 **Motor[x].Ctrl** 에 의해 선택되는 알고리즘(표준 또는 사용자 설명)과 관계없이 사용됩니다.

이 요소 이름과도 관계없이 D/A 컨버터는 출력에 반드시 필요하지는 않습니다. Power PMAC 이 모터의 전류(commutation) 또는 전류 루프 계산을 실행하는 경우에 **Motor[x].MaxDac** 은 해당 알고리즘에 토크 입력값의 크기를 제한합니다.

클로즈드 루프 서보 출력의 크기가 있는 서보 사이클에서 **MaxDac** 에 의해 제한되는 경우에 내장된 서보 알고리즘의 **Motor[x].Servo.Ki** 로 제공되는 적분 동작은 다음 서보 사이클에서는 사용되지 않습니다. 이는 '안티 와인드업' 보호로 알려져 있는 기능을 제공하여 적분기가 "과충전"되는 것을 방지합니다.

out 명령에서의 오픈 루프 출력의 크기는 **MaxDac** 의 값으로 제한됩니다. 따라서 **out** 명령의 100(MaxDac 의 %)을 초과하는 값은 **out100** 명령과 동일한 결과가 됩니다.

출력 장치의 실제 분해능이 달라도 **Motor[x].MaxDac** 은 16 비트 출력 단위입니다. 예를 들어 18 비트 DAC 출력에서는 10,000 의 **Motor[x].MaxDac** 의 값은 DAC 에 쓰여지는 40,000 의 값과 등가가 됩니다.

서보 루프 출력이 속도 명령으로 "속도 모드" 앰프를 구동하는 경우에 **Motor[x].MaxDac** 은 순간 속도 명령 제한으로 기능합니다. 서보 출력이 토크/전류 명령으로 "토크 모드", "사인파 모드" 또는 "다이렉트 PWM 파워 블록" 앰프를 구동하는 경우에 **Motor[x].MaxDac** 은 순간 토크/전류 명령 제한으로 기능합니다.

사인파 출력 모드(**Motor[x].PhaseCtrl** 의 비트 0 또는 비트 2 = 1, **Motor[x].pAdc** = 0)에서는 앰프 게인은 보통 PMAC 출력의 1 볼트당 상전류의 암페어로 주어집니다. 그러나 모터와 앰프의 제한은 보통 RMS 암페어값으로 주어집니다. 이 경우에 피크 상전류값이 RMS 값보다 큰 $\sqrt{2}(1.414)$ 배라고 생각하면 됩니다.

3 상 모터(**Motor[x].PhaseOffset** = +/-683)의 다이렉트 PWM 모드 (**Motor[x].PhaseCtrl** 의 비트 0 = 1, **Motor[x].pAdc**>0)에서는 사인파 상전류에서의 ADC 읽기의 파고값은 직교 전류와 다이렉트 전류의 벡터합보다 커서 $1/\cos(30^\circ)$ 또는 1.15 배가 됩니다. 따라서 파고값을 한번 확립한 후에 연속적인 위상 전류를 A/D 컨버터에서 확인하고자 하는 경우에 이 값에 $\cos(30^\circ)$ 또는 0.866 을 곱하면 **Motor[x].MaxDac** 의 값이 됩니다. 전류 제한이 RMS 값으로 주어지는 경우에 이 값에 $\sqrt{2}(1.414)$ 를 곱하면 피크의 상전류값이 된다는 것을 기억하십시오.

다른 운전 모드에서의 **Motor[x].MaxDac** 계산의 자세한 내용은 사용자 설명서의 어플리케이션의 안전화 장을 참조하십시오.

예

토크 모드의 브러시리스 모터 앰프에는 $15A_{rms}$ 최대의 단속적인 전류 능력(즉 $\pm 10V$ 입력에서 $\pm 15A_{rms}$ 의 전류를 명령)이 있습니다. 단, 브러시리스 모터에는 그 자체 $10A_{rms}$ 의 단속적인 전류 제한이 있습니다. **MaxDac**은 다음과 같이 계산됩니다.

$$MaxDac = 32,768 * \frac{10A_{rms}}{15A_{rms}} = 21,845$$

각 상 2A/Volt의 상호 컨덕턴스 게인을 가지는 사인파 입력 앰프는 전체 범위($\pm 10V$)의 사인파 명령에서 각 상에서 20A(피크)의 전류 출력을 생성합니다. 이 경우에 Power PMAC의 **PwmSf**는 기본설정의 32767입니다. 이는 $20/\sqrt{2} = 14.1A_{rms}$ 의 RMS의 크기에 상당합니다. 모터에 $12A_{rms}$ 의 단속적인 전류 제한이 있는 경우에 **MaxDac**은 다음 식으로 설정할 수 있습니다.

$$MaxDac = 32,768 * \frac{12A_{rms}}{14.1A_{rms}} = 27,887$$

다이렉트 PWM 앰프에서 제어되는 $8A_{rms}$ 단속적인 전류 제한을 가지는 3상 브러시리스 서보 모터에서는 16.26A의 상전류에 대해 ADC가 32,768의 순간 최대값을 통지합니다. **MaxDac**은 다음과 같이 연산할 수 있습니다.

$$MaxDac = 8 * \sqrt{2} * \frac{32,768}{16.26} * \cos(30^\circ) = 19,745$$

이 식의 $\sqrt{2}$ 의 계수는 순간 전류에서 RMS 전류로의 변환입니다. $\cos(30^\circ)$ 의 계수는 3상(abc)에서 2상(dq)으로의 변환입니다.

Motor[x].MaxPos 모터의 양의 오버트래블 제한

설명: 양의 위치 오버트래블 제한

범위: 부동 소수점

단위: 모터 단위

기본값: 0.0(Motor[x].MinPos >= 0인 경우는 무효)

기준 I 변수명: lxx13

Motor[x].MaxPos는 지정된 모터의 정방향의 최대 허용 위치를 설정합니다. 모터가 정방향의 상한을 초과하는 위치로 명령을 받는 경우에 Power PMAC은 자동으로 그 동작을 Abort 합니다. 충분히 빨리 그 명령을 검출할 수 있는 경우에는 **Motor[x].MaxPos**에서 정의된 위치에서 모터를 정지시킵니다.

이동 연산 시 계산된 이동 또는 세그먼트의 종점이 위반인지 여부를 결정하기 위해 **Motor[x].MaxPos**와 비교됩니다. 이동 실행 시 순간의 최종적인 명령 위치가 위반인지 여부를 결정하기 위해 (**Motor[x].MaxPos + Motor[x].SoftLimitOffset**)과 비교됩니다.

Motor[x].MaxPos 는 모터 단위로 표시되며 다음 최신 명령, 동작 등으로 정의되는 모터 제로 위치에서의 상대 거리입니다. 해당 명령, 동작은 전원 셋업/리셋, 원점 복귀, 절대 위치 읽기 또는 **pset**, **pload** 명령, **pclr** 오프셋 명령입니다. 모터에 할당된 축의 매트릭스 변환은 이 제한의 동작에 영향을 주지 않습니다. 제한을 모두 유효로 하기 위해서는 **Motor[x].MaxPos** 는 **Motor[x].MinPos** 보다 커야 합니다(절대적인 의미에서). 따라서 해당 2가지 제한을 같게 하는 것은 이를 무효로 하는 좋은 방법입니다. 원점 서치 이동 중의 원점 트리거가 검출될 때까지 이 제한은 유효가 아닙니다. 이 제한은 트리거 이후 동작 중에 유효가 됩니다.

소프트웨어 리미트에 도달했을 때의 정확한 정지 동작은 제한 위반이 이동 연산 시 또는 동작 실행 시에 검출되는지에 따라 좌우되고 또한 만일 연산 시라면 동작의 타입과 **Coord[x].SoftLimitStopDis** 의 설정에 좌우됩니다. 사용자 설명서의 어플리케이션 안전화 장의 소프트웨어 위치 제한에 대한 내용을 참조하십시오.

Motor[x].MaxSpeed 모터의 프로그램 최대 속도

설명: 최대 프로그램 속도의 크기

범위: 부동 소수점

단위: 모터 단위/msec

기본값: 32

기존 I 변수명: lxx16

Motor[x].MaxSpeed 는 프로그램 동작의 몇 가지 타입의 최대 속도의 크기를 설정합니다. 직선 보간(**linear**) 모드 동작에서는 세그먼트화되지 않은 경우(**Coord[x].SegMoveTime** = 0)는 이동마다 모터의 제한으로 기능하고 세그먼트화된 경우(**Coord[x].SegMoveTime** > 0)는 세그먼트마다 모터의 제한으로 기능합니다. 프로그램의 직선 보간 모드 동작이 이 모터의 제한을 초과하는 속도를 명령하는 경우에 이 제한을 위반하지 않게 동작 시간이 충분한 시간만큼만 연장됩니다. 또한 협조와 궤적이 유지되도록 비례하여 좌표계의 다른 모터의 속도를 낮춥니다(해당 모터가 자체 제한을 위반하지 않은 경우라도).

세그먼트화된 룩 어헤드 알고리즘이 좌표계에서 유효한 경우에 **Motor[x].MaxSpeed** 는 나아가 세그먼트마다 원호 보간과 **pvt** 모드 동작(직선 보간 모드 동작과 동일하게) 속도의 최대 제한을 설정합니다.

Motor[x].RapidSpeedSel 이 1 인 기본값으로 설정되는 경우 **Motor[x].MaxSpeed** 는 **rapid** 모드 동작의 명령 속도 크기로 기능합니다. (그렇지 않으면 **Motor[x].JogSpeed** 가 사용됩니다).

예

10 마이크론 모터 단위에서 30m/min(약 1200inch/min)의 최대 속도를 설정하기 위한 식은 다음과 같습니다.

$$MaxSpeed = \frac{30m}{min} * \frac{motor\_unit}{10^{-5}m} * \frac{min}{6*10^4ms} = 50 \left(\frac{motor\_unit}{ms} \right)$$

Motor[x].MinPos 모터의 음의 오버트래블 제한

설명:	음의 위치 오버트래블 제한
범위:	부동 소수점
단위:	모터 단위
기본값:	0.0(Motor[x].MaxPos <= 0 인 경우는 무효)
기존 I 변수명:	lx14

Motor[x].MinPos에서는 지정된 모터의 역방향의 최대 허용 위치를 설정합니다. 모터가 역방향의 상한을 초과하는 위치로 명령을 받는 경우에 Power PMAC 은 자동으로 그 동작을 Abort 합니다. 충분히 빨리 그 명령을 검출할 수 있는 경우에는 **Motor[x].MinPos**에서 정의된 위치에서 모터를 정지시킵니다.

이동 연산 시 계산된 이동 또는 세그먼트의 종점이 위반인지 여부를 결정하기 위해 **Motor[x].MinPos**와 비교됩니다. 동작 실행 시 순간의 최종적인 명령 위치가 위반인지 여부를 결정하기 위해 (**Motor[x].MinPos + Motor[x].SoftLimitOffset**)과 비교됩니다.

Motor[x].MinPos는 모터 단위로 표시되며 다음 최신 명령, 동작 등으로 정의되는 모터 제로 위치에서의 상대 거리입니다. 해당 명령, 동작은 전원 셋업/리셋, 원점 복귀, 절대 위치 읽기 또는 **pset**, **pload** 명령, **pclr** 오프셋 명령입니다. 모터에 할당된 축의 매트릭스 변환은 이 제한의 동작에 영향을 주지 않습니다. **Motor[x].MinPos**는 제한을 모두 유효로 하기 위해 **Motor[x].MaxPos** 미만이어야 합니다(절대적인 의미에서). 따라서 해당 2가지 제한을 같게 하는 것은 이를 무효로 하는 좋은 방법입니다. 원점 서치 이동 중의 원점 트리거가 검출될 때까지 이 제한은 유효가 아닙니다. 이 제한은 트리거 이후 동작 중에 유효가 됩니다.

소프트웨어 리미트에 도달했을 때의 정확한 정지 동작은 제한 위반이 이동 연산 시 또는 동작 실행 시에 검출되는지에 따라 좌우되고 또한 만일 연산 시라면 동작의 타입과 **Coord[x].SoftLimitStopDis**의 설정에 좌우됩니다. 사용자 설명서의 어플리케이션 안전화 장의 소프트웨어 위치 제한에 대한 내용을 참조하십시오.

Motor[x].MotorMode 모터의 동작 모드

설명:	동작 모드
범위:	0 ..5
단위:	열거 형식
기본값:	0(독립 운전)

Motor[x].MotorMode는 "네트워크 슬레이브"로서의 모터 동작을 가능하게 하는 모터의 동작 모드를 지정합니다. 여기에서 모터는 MACRO 링과 같은 네트워크에서 주기적인 명령을 수신합니다. 이 요소는 2013 년 3/4 분기에 출시된 새로운 V1.6 펌웨어입니다. 기본값 0에서는 독립 운전을 지정하면 이 요소가 없었던 이전 펌웨어 버전과 호환됩니다.

Motor[x].MotorMode 는 다음 값이 되고 해당 운전 모드를 지정합니다.

- 0: 독립 운전(사이클릭 명령은 없음)
 - 1: 사이클릭 위치 명령*
 - 2: 사이클릭 속도 명령
 - 3: 사이클릭 토크/힘 명령
 - 4: 사이클릭 상전류("사인파") 명령
 - 5: 사이클릭 상 전압("다이렉트 PWM") 명령
- * 완전하게 지원되지는 않습니다.

0 보다 큰 값으로 설정되는 경우에 저장 설정 요소 **Motor[x].pMotorNode** 에서 주소 지정되는 레지스터에서 시작하는 지정된 타입의 사이클릭 명령이 모터는 필요합니다. 그리고 **Motor[x].pMotorNode** 의 값과 **Motor[x].MotorNodeOffset** 의 값의 합계값으로 주소 지정되는 레지스터에서 시작하는 피드백값이 배치됩니다.

Motor[x].MotorMode 가 1 로 설정되는 경우에 모터는 제 1 명령 레지스터에서 위치 명령값(궤적 생성부에서의)을 검출하고 제 4 명령 레지스터에서 명령 플래그를 서보 사이클마다 검출합니다. 2 번째와 3 번째 명령 레지스터는 이 모드에서는 자동적인 목적으로 사용되지 않습니다. 그리고 제 1 피드백 레지스터에서는 엔코더 변환 테이블에서 처리된 위치 피드백값이 배치되고 제 4 피드백 레지스터에서는 상태 플래그가 배치됩니다. 이 모드에서 서보 사이클마다 해당 궤적 명령 위치를 출력시키기 위해서는 Power PMAC 에서의 협조가 필요하지만 모든 클로즈드 루프 또는 상전류(phase commutation)는 실행하지 않습니다. 이 모드가 Power PMAC 또는 MACRO 링 프로토콜에서 완전하게는 지원되지 않는다는 점에 주의하십시오. 현 시점에서는 이 모드의 사용을 권장하지 않습니다.

Motor[x].MotorMode 가 2 로 설정되는 경우에 모터는 제 1 명령 레지스터에서 속도 명령값(위치 서보 루프에서의 출력과 등가)을 검출하고 제 4 명령 레지스터에서 명령 플래그를 서보 사이클마다 검출합니다. 2 번째와 3 번째 명령 레지스터는 이 모드에서는 자동적인 목적으로 사용되지 않습니다. 그리고 제 1 피드백 레지스터에서는 엔코더 변환 테이블에서 처리된 위치 피드백값이 배치되고 제 4 피드백 레지스터에서는 상태 플래그가 배치됩니다. 이 모드에서는 아우터 위치 서보 루프를 실행하는 데 Power PMAC에서의 협조가 필요하지만 모터의 이너 속도 루프, 상전류(phase commutation), 전류 루프를 실행하지 않습니다.

Motor[x].MotorMode 가 3 으로 설정되는 경우에 모터는 제 1 명령 레지스터에서 "토크" 명령값(위치/속도 서보 루프에서의 출력과 등가)을 검출하고 제 4 명령 레지스터에서 명령 플래그를 서보 사이클마다 검출합니다. 2 번째와 3 번째 명령 레지스터는 이 모드에서는 자동적인 목적으로 사용되지 않습니다. 그리고 제 1 피드백 레지스터에서는 엔코더 변환 테이블에서 처리된 위치 피드백값이 배치되고 제 4 피드백 레지스터에서는 상태 플래그가 배치됩니다. 이 모드에서는 완전한 위치/속도 서보 루프를 실행하는 데 Power PMAC 에서의 협조가 필요하지만 모터의 상전류(phase commutation), 전류 루프를 실행하지 않습니다.

Motor[x].MotorMode 가 4 로 설정되는 경우에 모터는 제 1 과 제 2 명령 레지스터에서 상전류 명령값을 검출하고 제 4 명령 레지스터에서 명령 플래그를 페이즈 사이클마다 검출합니다. 제 3 명령 레지스터는 이 모드에서는 사용되지 않습니다. 그리고 제 1 피드백 레지스터에서는 엔코더 변환 테이블에서 처리된 위치 피드백값이 배치되고 제 4 피드백 레지스터에서는 상태 플래그가 배치됩니다. 이 모드에서는 모터의 위치/속도 서보 루프와 상전류(phase commutation)를 실행하는 데 Power PMAC 의 협조가 필요하지만 모터의 전류 루프를 실행하지 않습니다.

Motor[x].MotorMode 가 5 로 설정되는 경우에 모터는 제 1, 제 2, 제 3 명령 레지스터에서 상 전압("PWM") 명령값을 검출하고 제 4 명령 레지스터에서 명령 플래그를 페이즈 사이클마다 검출합니다. 그리고 제 1 피드백 레지스터에서는 엔코더 변환 테이블에서 처리된 위치 피드백값이 배치되고 제 2 와 제 3 피드백 레지스터에서는 전류 피드백값이 배치되며 제 4 피드백 레지스터에서는 상태 플래그가 배치됩니다. 이 모드에서는 모터의 위치/속도 서보 루프, 상전류(phase commutation) 그리고 전류 루프를 실행하는 데 Power PMAC 의 협조가 필요합니다.

Motor[x].MotorNodeOffset 모터의 네트워크 명령 오프셋

설명: 사이클릭 네트워크 명령에서의 입출력 주소 간의 차

범위: 0 ..255

단위: Power PMAC 주소

기본값: 0

Motor[x].MotorNodeOffset 은 사이클릭 네트워크 슬레이브로 사용되는 모터의 명령과 피드백 주소 간의 수치적인 차를 지정합니다. **Motor[x].MotorMode** 가 0 보다 큰 값으로 설정되고 사이클릭 네트워크 명령을 받도록 설정된 경우에만 이 요소가 사용됩니다.

사이클릭 네트워크 명령은 이 모드의 모터에 **Motor[x].pMotorNode** 로 지정된 주소에서 시작하는 연속하는 4 개의 레지스터를 배치합니다. **Motor[x].pMotorNode** 의 값과 **Motor[x].MotorNodeOffset** 의 값의 합계값으로 지정된 주소에서 시작하는 연속하는 4 개의 레지스터에 해당 피드백값이 저장됩니다.

PMAC2 스타일 "DSPGATE2" MACRO IC 를 사용하는 경우에 피드백 레지스터는 명령 레지스터와 동일한 주소가 됩니다(동일한 주소라도 별도의 물리 레지스터). 이 경우에 **Motor[x].MotorNodeOffset** 은 0 으로 설정되어야 합니다.

PMAC3 스타일 "DSPGATE3"MACRO IC 를 사용하는 경우에 피드백 레지스터는 명령 레지스터와는 다른 주소가 됩니다. 피드백에 사용되는 노드의 **MacroOuta** 레지스터는 명령에 사용되는 동일한 노드의 **MacroIna** 레지스터에서 64 만큼 고위의 주소에서 시작합니다. 이 경우에 **Motor[x].MotorNodeOffset** 은 64 로 설정되어야 합니다.

Motor[x].pAbsPhasePos 모터의 절대 전류(commutation) 위치 포인터

설명: 전원 셋업 시의 절대 전류(commutation) 위치 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: 0

기준 I 변수명: lxx81

Motor[x].pAbsPhasePos 는 전원 셋업 시의 전류(commutation) 로터 각도 위치(절대 위치)를 얻기 위해 사용하는 레지스터의 주소를 지정합니다. 이것은 이동 중인 전류(commutation) 위치의 레지스터와는 다른 주소입니다(이는 **Motor[x].pPhaseEnc** 로 지정됩니다). 그리고 다시 전원 셋업 시에는 이동 중과 다른 분해능과 다른 방향 검출도 가능합니다.

Power PMAC 에서 전류(commutation)되는 동기 모터(보통 브러시리스 서보 모터)만 전원 셋업 시의 절대 페이즈 참조의 확립을 필요로 합니다. 페이즈 참조는 "페이즈 서치 동작" 또는 절대 위치 센서에서의 읽기 중 하나에서 실현할 수 있습니다. **Motor[x].pAbsPhasePos** 가 기본값 0 이거나 또는 **Motor[x].PhaseFindingTime** 이 4 보다 큰 값으로 설정되는 경우에 페이즈 참조는 페이즈 서치 동작을 통해 수행됩니다.

그러나 **Motor[x].pAbsPhasePos** 가 실제 주소로 설정되고 **Motor[x].PhaseFindingTime** 이 5 미만의 값으로 설정되는 경우에 페이즈 참조는 절대 위치 센서에서의 읽기로 수행됩니다. 이 경우에 지정된 주소에서의 데이터는 **Motor[x].AbsPhasePosFormat** 에 따라서 해석됩니다. 그 결과의 값은 **Motor[x].AbsPhasePosSf** 로 스케일링되고 **Motor[x].AbsPhasePosOffset** 에 의해 오프셋되면 로터 각도 위치가 됩니다.

Motor[x].pAbsPhasePos 의 일반적인 설정은 다음과 같습니다.

- PMAC2 스타일 ASIC 홀 센서(ACC-24E2x 등)에서는 **Gate1[j].Chan[j].Status.a**
- PMAC3 스타일 ASIC 홀 센서(ACC-24E3 등)에서는 **Gate3[j].Chan[j].Status.a**
- PMAC3 스타일 시리얼 엔코더 피드백(ACC-24E3 등)에서는 **Gate3[j].Chan[j].SerialEncDataAa**
- PMAC3 스타일 리졸버 피드백(ACC-24E3 등)에서는 **Gate3[j].Chan[j].AtanSumOfSqr.a**
- "IOGATE" 페러럴 피드백(ACC-14E 등)에서는 **Gatelo[j].DataReg[j].a**
- A/D 컨버터 피드백(ACC-28E 등)에서는 **Gatelo[j].AdcUdata[j].a**
- 정의된 요소가 없는 하드웨어 레지스터에서는 **Sys.piom+{offset}**
- 절대 페이즈 위치와 이동 중인 페이즈 위치가 동일한 센서를 사용하고 동일하게 읽기를 수행하는 경우에는 **Motor[x].PrevPhaseEnc.a**

- 이미 처리된 위치를 사용하는 경우에 특히 절대 위치가 이동 중인 서보 위치와 동일한 센서를 사용하는 경우에는(ACC-58E 리졸버 인터페이스 보드 등) **EncTable[j].PrevEnc.a**
- 전원 셋업 시의 페이즈 위치 데이터를 얻기 위해 자동으로 지원되지 않는 전송 또는 처리를 수행해야 하는 경우에 사용자 공유 메모리의 레지스터를 경유하여 수행하는데 그 경우의 **Sys.pushm+{offset}** 또는 **Sys.Udata[j].a**

이 위치 데이터에 대해 하나 이상의 레지스터를 읽어야 하는 경우에 **Motor[x].pAbsPhasePos**에는 최하위 수치를 저장하는 레지스터의 주소를 설정합니다.

온라인 모터 명령 \$이 전송되는 경우 또는 데이터 구조체 요소 **Motor[x].PhaseFindingStep**이 1로 설정되는 경우에 절대값 페이즈 위치를 읽습니다. **Motor[x].PowerOnMode**의 비트 1(값 2)이 1로 설정되는 경우에 전원 셋업/리셋 시 자동으로 읽습니다.

Motor[x].PowerOnMode의 비트 0(값 1)이 1로 설정되는 경우에 모터는 읽은(성공) 후 위치 루프를 닫고 자동으로 Enable 됩니다. 이 비트가 0으로 설정되는 경우에 모터는 읽은 후 kill된 상태 그대로입니다.

Motor[x].pAbsPos 모터의 절대 위치 포인터

설명: 전원 셋업 시의 절대 위치 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: 0

기존 I 변수명: lxx10

Motor[x].pAbsPos는 전원 셋업 시의 모터의 절대 위치를 얻기 위해 사용하는 레지스터의 주소를 지정합니다. 이것은 **Motor[x].pEnc**와 **Motor[x].pEnc2**로 지정되는 이동 중인 서보 위치 피드백과는 다른 주소를 사용할 수 있습니다. 그리고 전원 셋업 위치는 이동 중의 위치와는 다른 분해능과 다른 방향 검출도 가능합니다.

Motor[x].pAbsPos가 기본값 0으로 설정되는 경우에 모터의 절대 위치의 위치 읽기는 실행되지 않습니다. 이 경우에 모터의 위치 참조를 확립하기 위해 원점 서치 동작을 수행할 수 있어야 합니다. 이 요소가 0의 값이고 온라인 명령 **hmz** 또는 버퍼 프로그램 명령 **homez**를 실행하면 단순히 현재 명령 위치는 0으로 설정됩니다.

그러나 **Motor[x].pAbsPos**가 실제 주소로 설정되는 경우에 위치 참조는 절대 위치 센서의 읽기로 실현됩니다. 이 경우에 지정된 주소의 데이터는 **Motor[x].AbsPosFormat**에 따라서 해석됩니다. 그 결과의 값은 **Motor[x].AbsPosSf**로 스케일링되고 **Motor[x].HomeOffset**에 의해 오프셋되면 최종적인 모터 위치가 됩니다.

Motor[x].pAbsPos의 일반적인 설정은 다음과 같습니다.

- PMAC3 스타일 시리얼 엔코더 피드백(ACC-24E3 등)에서는 **Gate3[i].Chan[j].SerialEncDataAa**
- "IOGATE" 페러럴 피드백(ACC-14E 등)에서는 **Gatelo[i].DataReg[j].a**
- A/D 컨버터 피드백(ACC-28E 등)에서는 **Gatelo[i].AdcUdata[j].a**
- 정의된 요소가 없는 하드웨어 레지스터에서는 **Sys.piom+{offset}**
- 절대 위치가 이동 중인 서보 위치와 동일한 센서를 사용하는 경우에 이미 처리된 위치를 사용하는 경우에는 **EncTable[i].PrevEnc.a**
- 전원 셋업 시의 절대 위치 데이터를 얻기 위해 자동으로 지원되지 않는 전송 또는 처리를 수행해야 하는 경우에 사용자 공유 메모리의 레지스터를 경유하여 수행하는데 그 경우의 **Sys.pushm+{offset}** 또는 **Sys.Udata[i].a**

이 위치 데이터에 대해 하나 이상의 레지스터를 읽어야 하는 경우에 **Motor[x].pAbsPos** 에는 최하위 수치를 저장하는 레지스터의 주소를 설정합니다.

온라인 모터 명령 **hmz** 가 전송되는 경우 또는 버퍼 프로그램 명령 **homez** 가 실행되는 경우에 절대 위치 데이터를 읽습니다. **Motor[x].PowerOnMode** 의 비트 2(값 4)가 1로 설정되는 경우에 전원 셋업/리셋 시 자동으로 읽습니다. 이 자동적인 읽기가 지정되는 경우에 해당 시점에서 절대 위치 센서가 적절한 위치 데이터를 출력하는 준비가 되었음을 보증해야 합니다.

Motor[x].pAdc 모터의 전류 피드백 포인터

설명: 전류 피드백 포인터

범위: 0, 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

기준 I 변수명: lxx82

Motor[x].pAdc 는 디지털 전류 루프를 유효로 하고 Power PMAC 이 전류(commutation)를 실행할 때의 디지털 전류 루프 피드백값에 사용하는 레지스터를 지정합니다. 이것은 2 상 전류 피드백에 사용되는 2 개의 레지스터의 1 번째의 주소가 됩니다(다음 레지스터는 다음 고위의 주소가 됩니다). 이와 같이 전류 피드백을 수행하기 위해 어떤 피드백 장치가 사용되는지를 결정합니다.

Motor[x].pAdc 가 0 으로 설정되는 경우에 전류(commutation) 알고리즘이 유효라도 Power PMAC 은 이 모터의 전류 루프를 실행하지 않습니다. 전류(commutation) 알고리즘이 유효이고(**Motor[x].PhaseCtrl** 의 비트 0 또는 비트 2 = 1) 그리고 **Motor[x].pAdc = 0** 인 경우 "사인파 출력" 전류(commutation) 모드가 유효가 됩니다. 그 경우에 전류 루프는 앰프 내에서 실행됩니다. 전류(commutation) 알고리즘이 유효가 아닌 경우(**Motor[x].PhaseCtrl** 의 비트 0 과 비트 2 = 0)에 **Motor[x].pAdc** 의 설정은 관계없습니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

가장 일반적으로는 **Motor[x].pAdc** 는 서보 IC 채널의 최초의 ADC 레지스터의 주소(PMAC2 스타일 IC 의 **Adc[0]**, PMAC3 스타일 IC 의 **AdcAmp[0]**) 또는 MACRO IC 노드의 최초의 전류 피드백 레지스터(레지스터[1])로 설정됩니다. 예를 들어

```
Motor[1].pAdc = Gate1[4].Chan[0].Adc[0].a
Motor[2].pAdc = Gate1[4].Chan[1].Adc[0].a
Motor[3].pAdc = Gate3[0].Chan[0].AdcAmp[0].a
Motor[4].pAdc = Gate3[0].Chan[1].AdcAmp[0].a
Motor[5].pAdc = Gate2[0].Macro[4][1].a
Motor[6].pAdc = Gate2[0].Macro[5][1].a
Motor[7].pAdc = Gate3[0].MacroInA[4][1].a
Motor[8].pAdc = Gate3[0].MacroInB[5][1].a
Motor[9].pAdc = 0
```

Motor[x].pAmpEnable 모터의 앰프 Enable 포인터

설명: 앰프 Enable(출력) 플래그의 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

Motor[x].pAmpEnable 은 선택된 모터에서 앰프 Enable 의 출력 플래그 신호에 사용되는 레지스터를 지정합니다. 여기에 해당 레지스터 주소가 저장됩니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

가장 일반적으로는 **Motor[x].pAmpEnable** 은 서보 IC 의 채널 제어 레지스터 주소로 설정됩니다. 예를 들어

```
Motor[1].pAmpEnable = Gate1[4].Chan[0].Ctrl.a
```

```
Motor[2].pAmpEnable = Gate3[0].Chan[1].OutCtrl.a
```

모터가 MACRO 링을 통해 제어되는 경우 **Motor[x].pAmpEnable** 은 MACRO 노드 출력 플래그 레지스터(노드의 출력 레지스터 3) 주소로 일반 설정됩니다. 예를 들어

```
Motor[3].pAmpEnable = Gate2[0].Macro[4][3].a
```

```
Motor[4].pAmpEnable = Gate3[0].MacroOutA[5][3].a
```

Power PMAC 은 앰프 Enable 신호를 제어 하기 위해 그 지정된 레지스터의 **Motor[x].AmpEnableBit** 에 저장된 번호의 비트를 사용합니다.

그 비트 자체의 요소명(예를 들어 **Motor[5].pAmpEnable = Gate1[6].Chan[0].AmpEna.a**)이 있는 경우에 레지스터 내의 지정 비트의 주소를 지정하는 것도 가능하지만 값은 풀 워드 요소의 주소(예를 들어 **Gate1[6].Chan[0].Ctrl.a**)로 통지되므로 주의하십시오.

\$\$\$**의 재초기화 명령에서는 Power PMAC 은 검출된 서보 IC 와 MACRO IC 에 따라서 **Motor[x].pAmpEnable** 의 값을 자동으로 할당합니다.

Motor[x].pAmpEnable 이 0 으로 설정되는 경우에 이 모터에서는 자동 앰프 Enable 기능은 사용되지 않습니다.

Motor[x].pAmpFault 모터의 앰프 이상 포인터

설명: 앰프 이상(입력) 플래그의 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

(= 0 - 무효 - 할당하는 하드웨어가 없는 경우)

Motor[x].pAmpFault 에서는 선택된 모터에서 앰프 입력 신호에 사용되는 레지스터를 지정합니다. 여기에 해당 레지스터 주소가 저장됩니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

앰프 이상 입력의 극성은 **Motor[x].AmpFaultLevel** 로 설정됩니다.

가장 일반적으로는 **Motor[x].pAmpFault** 는 서보 IC 의 채널 Status 레지스터 주소로 설정됩니다. 예를 들어

Motor[1].pAmpFault = Gate1[4].Chan[0].Status.a

Motor[2].pAmpFault = Gate3[4].Chan[1].Status.a

모터가 MACRO 링을 통해 제어되는 경우 **Motor[x].pAmpFault** 는 MACRO 노드 입력 플래그 레지스터(노드의 입력 레지스터 3) 주소로 보통 설정됩니다. 예를 들어

Motor[3].pAmpFault = Gate2[0].Macro[4][3].a

Motor[4].pAmpFault = Gate3[0].MacroInA[5][3].a

PowerPMAC 은 앰프 이상 신호를 읽기 위해 지정된 레지스터의 **Motor[x].AmpFaultBit** 에 저장된 번호의 비트를 사용합니다.

그 비트 자체의 요소명(예를 들어 **Motor[5].pAmpFault = Gate1[6].Chan[0].AmpFault.a**)에 있는 경우에 레지스터 내의 지정 비트의 주소를 지정하는 것도 가능하지만 값은 풀 워드 요소의 주소(예를 들어 **Gate1[6].Chan[0].Status.a**)로 통지되므로 주의하십시오.

\$\$\$**의 재초기화 명령에서는 Power PMAC 은 검출된 서보 IC 와 MACRO IC 에 따라서 **Motor[x].pAmpFault** 의 값을 자동으로 할당합니다.

Motor[x].pAmpFault 가 0 으로 설정되는 경우에 이 모터에서는 자동 앰프 이상 기능은 사용되지 않습니다. 재초기화 시 모터에 자동 할당하는 하드웨어가 존재하지 않는 경우에 이 요소는 자동으로 0 으로 설정되고 앰프 이상 입력 기능은 무효가 됩니다(2014 년 1/4 분기에 출시한 V1.6 펌웨어의 새로운 기능입니다. 이전 버전에서는 이 경우에 사용자 공유 메모리 버퍼의 시작 레지스터의 **Sys.pushm** 에 할당되었습니다.).

Motor[x].pAuxFault 모터의 보조 이상 포인터

설명: 보조의 이상 레지스터 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: 0(보조의 이상 검출 무효)

Motor[x].pAuxFault 는 그 모터에서 "보조 이상" 비트를 감시하기 위해 사용되는 레지스터를 지정합니다. 여기에 해당 레지스터 주소가 저장됩니다. **Motor[x].pAuxFault** 가 기본값 0 으로 설정되는 경우에 이 모터에서는 자동 엔코더 소실 검출 기능은 사용되지 않습니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

PowerPMAC 은 앰프 이상 상태를 읽기 위해 지정된 레지스터의 **Motor[x].AuxFaultBit** 에 저장된 번호의 비트를 사용합니다. 엔코더 소실 Status 비트의 극성은 **Motor[x].AuxFaultLevel** 로 설정됩니다. 에러 발생을 단정하기 전에 이상 입력 상태가 검출되는 스캔의 누적 수가 **Motor[x].AuxFaultLimit** 로 지정됩니다.

Motor[x].pAuxFault 에는 몇 가지 사용 목적이 있습니다. 듀얼 엔코더 피드백을 사용하는 경우 **Motor[x].pEncLoss** 에서는 하나의 엔코더 소실을 검출할 수 있고 **Motor[x].pAuxFault** 는 또 하나를 검출하기 위해 사용됩니다. 물리적인 모터가 서멀 이상 출력을 가지는 경우에 Power Brick 에서는 검출할 수 있는데 **Motor[x].pAuxFault** 는 이것을 검출하기 위해 사용됩니다. 다른 이상 입력에도 동일하게 사용할 수 있습니다.

보조 이상이 검출되면 이 모터는 Power PMAC 에 의해 자동으로 "kill" 됩니다(Disable 됩니다). 동일한 좌표계의 다른 모터는 kill 되거나 "Abort" 됩니다(감속하여 제어된 정지 상태로). 이것은 이 모터의 **Motor[x].FaultMode** 의 비트 1(값 2)로 결정됩니다. 모터 Status 비트 **Motor[x].AuxFault** 는 1 로 설정됩니다.

"T" 플래그 입력 회로를 가지는 Power Brick PMAC3 스타일 인터페이스 보드에 연결된 모터의 서멀 센서에서는 **Motor[x].pAuxFault** 는 **Gate3[j].Chan[j].Status.a** 로 설정할 수 있습니다. 이 레지스터의 비트 15에 T 플래그 입력 비트가 있습니다.

ASIC 의 바깥쪽에 "배타적 논리합" 직교(AB 상) 소실 검출 회로를 가지는 ACC-24E2x PMAC2 스타일 축 인터페이스 보드에 연결된 직교(AB 상) 엔코더에서는 **Motor[x].pAuxFault** 는 **Gate1[j].Chan[j].EncLossN.a** 로 설정할 수 있습니다(대신 **ACC24E2x[j].Chan[j].EncLossN.a** 의 보드 구조체 별칭도 사용할 수 있습니다). 이는 실제로는 ASIC 중에 없으므로 복귀값으로 **Cid[n].PartData[j].a** 가 통지됩니다. 엔코더 소실 Status 비트는 이 레지스터의 비트 13에 있습니다.

일부 프로토콜에 좌우되는 소실 검출 회로를 가지는 ACC-84E PMAC2 스타일 시리얼 엔코더 인터페이스 보드에 연결된 시리얼 엔코더에서는 **Motor[x].pAuxFault** 는 **ACC84E[j].Chan[j].SerialEncDataB.a** 로 설정할 수 있습니다. 가장 일반적으로는 이 레지스터의 비트 31에서 "타임아웃" 에러 비트가 사용됩니다.

ASIC 에 "배타적 논리합" 직교(AB 상) 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 직교(AB 상) 엔코더에서는 **Motor[x].pAuxFault** 는 **Gate3[j].Chan[j].Status.a** 로 설정됩니다(대신 **ACC24E3[j].Chan[j].Status.a** 의 보드 구조체의 별칭도 사용할 수 있습니다). 이 레지스터의 비트 28에 **LossStatus** 의 투과적인 AB 상 소실 비트가 있습니다.

"제곱합"의 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 아날로그 사인파 엔코더 또는 리졸버에서는 **Motor[x].pAuxFault** 는 **Gate3[j].Chan[j].Status.a** 로 설정할 수 있습니다(대신 **ACC24E3[j].Chan[j].Status.a** 의 보드 구조체 별칭도 사용할 수 있습니다). 이 레지스터의 비트 31에 **SosError** 의 투과적인 "제곱합" 에러 비트가 있습니다.

일부 프로토콜에 좌우되는 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 시리얼 엔코더에서는 **Motor[x].pAuxFault** 는 **Gate3[j].Chan[j].SerialEncDataB.a** 로 설정할 수 있습니다(대신 **ACC24E3[j].Chan[j].SerialEncDataB.a** 의 보드 구조체 별칭도 사용할 수 있습니다). 가장 일반적으로는 이 레지스터의 비트 31에서 "타임아웃" 에러 비트가 사용됩니다.

그 비트 자체의 요소명(예를 들어 **Motor[5].pAuxFault = Gate3[1].Chan[0].T.a**)이 있는 경우에 레지스터 내의 지정 비트의 주소를 지정하는 것도 가능하지만 값은 폴 워드 요소의 주소(예를 들어 **Gate3[1].Chan[0].Status.a**)로 통지되므로 주의하십시오.

Motor[x].pBrakeOut 모터의 브레이크 출력 포인터

설명: 모터 브레이크(출력) 플래그 포인터

범위: 적당한 주소

단위: 데이터 구조체 요소 주소

기본값: 0

Motor[x].pBrakeOut에서는 선택된 모터에서 모터 브레이크 출력 플래그 신호에 사용되는 레지스터를 지정합니다. 여기에는 이 레지스터의 주소가 저장됩니다. **Motor[x].pBrakeOut**이 기본값 0인 경우에 이 모터에서는 자동 모터 브레이크 제어 기능은 사용되지 않습니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

가장 일반적으로는 **Motor[x].pBrakeOut**은 PMAC3 스타일 "DSPGATE3" IC의 채널 출력 제어 레지스터의 주소로 설정됩니다. 또한 ACC-11E 또는 ACC-68E와 같은 범용 I/O 보드의 출력 레지스터의 주소로 설정할 수 있습니다. 예를 들어

Motor[1].pBrakeOut=Gate3[0].Chan[0].OutCtrl.a

Motor[2].pBrakeOut=Gatelo[2].DataReg[3].a

Motor[3].pBrakeOut=Sys.piom + \$A0000C

Power PMAC은 브레이크 출력 신호를 제어하기 위해 그 지정된 레지스터의 **Motor[x].BrakeOutBit**에 설정된 번호의 비트를 사용합니다.

그 비트 자체의 요소명(예를 들어 **Motor[5].pBrakeOut = Gate3[1].Chan[0].OutFlagB.a**)이 있는 경우에 레지스터 내의 지정 비트의 주소를 지정하는 것도 가능하지만 값은 풀 워드 요소의 주소(예를 들어 **Gate3[1].Chan[0].OutCtrl.a**)로 통지되므로 주의하십시오.

지정된 브레이크 출력 비트는 항상 브레이크 구동 상태에서 0의 값이 되고 브레이크 개방 상태에서 1의 값이 됩니다. 이 소프트웨어의 극성은 변경할 수 없습니다. 사용자 책임으로 호환성이 있는 안전한 방법으로 이 출력을 사용하는 적절한 하드웨어를 실현하십시오. 이 비트는 전원 셋업/리셋 시 브레이크를 구동 상태로 하도록 항상 0으로 설정됩니다. 사용자 입력에 의하거나 위치독 타이머 트립과 같은 장애에 의해 컨트롤러가 "리셋" 상태가 되는 경우에 이 출력은 0의 상태로 강제적으로 설정됩니다.

모터가 Enable이나 Disable되는 경우에 자동적인 브레이크 릴리스와 브레이크 구동의 타이밍은 저장 설정 요소 **Motor[x].BrakeOffDelay**와 **Motor[x].BrakeOnDelay**에 의해 제어됩니다. 페이즈 서치 동작과 같은 특별한 동작인 경우는 출력 비트는 "수동"으로 설정해야 합니다(예를 들어 **Gate3[0].Chan[0].OutFlagB = 1**, **Gate3[0].Chan[0].OutCtrl |= \$20** 또는 **Gatelo[2].DataReg[0] |= 2**). 해당 조작은 브레이크를 필요로 하는 모터에는 권장하지 않으므로 주의하십시오.

자동 브레이크 제어 기능이 유효한 경우에 사용자는 **j+** 또는 **hm**과 같은 실제 모터 동작 명령에서 Disable 상태의 모터를 Enable로 하지 마십시오.

Motor[x].pBufPos 모터의 아우터 루프 확장 버퍼 포인터

설명: 모터의 아우터 루프 위치의 확장 저장 버퍼로의 포인터

범위: 0, **Sys.BufPos[7][0].a**($i = 0 \sim 7$)

단위: Power PMAC 주소

기본값: 0

Motor[x].pBufPos에서는 확장 저장 버퍼의 시작 주소를 지정하고(만일 있다면) 각 서보 사이클에서의 모터의 아우터 루프 현재 위치를 저장하기 위해 사용됩니다. 0으로 설정되는 경우에 이 모터에는 확장 저장 버퍼는 사용되지 않습니다.

Motor[x].pBufPos가 8개인 **Sys.BufPos** 확장 저장 버퍼 내의 하나의 최초 요소 주소로 설정되는 경우(즉, **Sys.BufPos[7][0].a**에 $i = 0 \sim 7$ 에서)에 각 서보 사이클에서 **Motor[x].ActPos**의 값은 상태 요소의 **Sys.Buf[7][7]**에 저장됩니다. 제 1 인덱스(i)는 버퍼 번호입니다. 제 2 인덱스(j)는 서보 사이클의 인덱스이고 범위는 0~255입니다. 이 인덱스값은 **Sys.ServoCount**의 하위 8 비트값과 같고 서보 사이클마다 더해집니다. 버퍼는 로터리 버퍼이며 사이클마다 그보다 256(2^8) 사이클 이전의 값에 덮어쓰기됩니다. 따라서 항상 가장 새로운 256 서보 사이클의 이력이 있습니다.

이 확장 버퍼링 기능은 모든 자동적인 기능에는 사용되지 않습니다. 그러나 이것은 상당한 위치 이력을 필요로 하는 어플리케이션 용도에 사용할 수 있습니다. 각 모터에서는 그 자체의 배열 **Motor[x].FltrPos[k]**에 최신 16 서보 사이클 분량의 현재 위치가 항상 저장되어 있으니 주의하십시오.

여러 개의 모터에서 해당 이너 또는 아우터 루프의 위치가 동일한 버퍼에 쓰여지게 설정되는 경우에 높은 번호의 모터값은 낮은 번호의 모터값에 서보 사이클마다 덮어쓰기됩니다. 이 타입의 덮어쓰기를 방지하는 방법은 없습니다.

Motor[x].pBufPos2 모터의 이너 루프 확장 버퍼 포인터

설명: 모터의 이너 루프 위치의 확장 저장 버퍼로의 포인터

범위: 0, **Sys.BufPos[7][0].a**, $i = 0 \sim 7$

단위: Power PMAC 주소

기본값: 0

Motor[x].pBufPos2는 확장 저장 버퍼의 시작 주소를 지정하고(만일 있다면) 각 서보 사이클에서의 모터의 이너 루프 현재 위치를 저장하기 위해 사용됩니다. 0으로 설정되는 경우에 이 모터에는 확장 저장 버퍼는 사용되지 않습니다.

Motor[x].pBufPos2가 8개인 **Sys.BufPos** 확장 저장 버퍼 내의 하나의 최초 요소 주소로 설정되는 경우(즉, **Sys.BufPos[7][0].a**에 $i = 0 \sim 7$ 에서)에 각 서보 사이클에서 **Motor[x].ActPos2**의 값은 상태 요소의 **Sys.Buf[7][7]**에 저장됩니다. 제 1 인덱스(i)는 버퍼 번호입니다. 제 2 인덱스(j)는 서보 사이클의 인덱스이고 범위는 0~255입니다. 이 인덱스값은 **Sys.ServoCount**의 하위 8 비트값과 같고 서보 사이클마다 더해집니다.

버퍼는 로터리 버퍼이며 사이클마다 그보다 256(2^8) 사이클 이전의 값에 덮어쓰기됩니다. 따라서 항상 가장 새로운 256 서보 사이클의 이력이 있습니다.

이 확장 버퍼링 기능은 모든 자동적인 기능에는 사용되지 않습니다. 그러나 이것은 상당한 위치 이력을 필요로 하는 어플리케이션 용도에 사용할 수 있습니다. 각 모터에서는 그 자체의 배열 **Motor[x].FltrPos2[k]**에 최신 16 서보 사이클 분량의 현재 위치가 항상 저장되어 있으니 주의하십시오.

여러 개의 모터에서 해당 이너 또는 아우터 루프의 위치가 동일한 버퍼에 쓰여지게 설정되는 경우에 높은 번호의 모터값은 낮은 번호의 모터값에 서보 사이클마다 덮어쓰기됩니다. 이 타입의 덮어쓰기를 방지하는 방법은 없습니다.

Motor[x].pCaptFlag 모터의 포착 플래그 포인터

설명: 엔코더 포착 트리거(입력) 플래그의 포인터

범위: 적당한 주소

단위: 데이터 구조체 요소 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

Motor[x].pCaptFlag에서는 선택된 모터에서 하드웨어 포착 트리거의 플래그 신호에 사용되는 주소를 지정합니다. 여기에는 이 레지스터의 주소가 저장됩니다. 이와 같이 플래그 신호의 어느 설정이 모터의 트리거 위치 포착에 사용되는지를 결정합니다. Power PMAC에서는 원점 서치와 같은 트리거 포함 동작에서의 포착 트리거의 검출에 이 레지스터가 사용됩니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

관련 요소 **Motor[x].CaptFlagBit**에서는 이 32 비트 레지스터의 어느 비트가 트리거 비트로 사용되는지 Power PMAC에 표시합니다. 트리거가 검출되면 하드웨어 포착이 지정되는 경우 Power PMAC은 그 포착된 위치로 **Motor[x].pCaptPos**로 지정된 레지스터를 사용합니다.

가장 일반적으로는 **Motor[x].pCaptFlag**에서는 서보 IC의 채널 Status 레지스터의 주소 또는 MACRO 노드의 입력 플래그 레지스터로 설정됩니다. 예를 들어

Motor[1].pCaptFlag = Gate1[4].Chan[0].Status.a

Motor[2].pCaptFlag = Gate3[0].Chan[1].Status.a

Motor[3].pCaptFlag = Gate2[0].Macro[4][3].a

Motor[4].pCaptFlag = Gate3[2].MacroInA[8][3].a

그 비트 자체의 요소명(예를 들어 **Motor[5].pCaptFlag = Gate1[6].Chan[0].PosCapt.a**)이 있는 경우에 레지스터 내의 지정 비트 주소를 지정하는 것도 가능하지만 값은 풀 워드 요소의 주소(예를 들어 **Gate1[6].Chan[0].Status.a**)로 통지되므로 주의하십시오.

\$\$\$***의 재초기화 명령에서는 **Motor[x].pCaptFlag** 는 사전 정의된 순서 규칙에 따라서 검출된 서보 채널 또는 MACRO 노드의 Status 레지스터에 자동으로 할당됩니다.

설정 요소 **Motor[x].EncType** 의 값이 스크립트 환경에 의해 설정되는 경우(C 프로그램에서가 아니라)에 **Motor[x].pCaptFlag** 는 **Motor[x].pEncStatus** 와 동일한 주소에 자동으로 설정됩니다.

Motor[x].pCaptPos 모터의 포착 위치 포인터

설명: 엔코더 포착 위치의 포인터

범위: 적당한 주소

단위: 데이터 구조체 요소 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

Motor[x].pCaptPos에서는 원점 서치 동작과 같은 트리거 포함 동작에서의 하드웨어 포착 위치를 얻기 위해 선택된 모터에서 읽은 레지스터를 지정합니다. 여기에는 이 레지스터의 주소가 저장됩니다. 이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

트리거 포함 동작에서의 포착 위치가 MACRO 링 상에서 얻어지는 경우에 Power PMAC 은 링 상의 소프트웨어 요청을 통해 해당 위치를 얻게 되고 이 경우 **Motor[x].pCaptPos**는 사용되지 않습니다.

관련 요소 **Motor[x].pCaptFlag**에서는 트리거가 발생했음을 전달하는 포착 트리거 비트로 어느 레지스터를 읽을지 Power PMAC 에 표시합니다. 보통 이는 포착된 위치 레지스터와 동일한 서보 IC 채널의 Status 플래그 레지스터를 사용합니다.

한번 포착된 위치가 이 레지스터에서 얻어지면 **Motor[x].CaptPosRightShift**, **Motor[x].CaptPosLeftShift** 그리고 **Motor[x].CaptPosRound** 의 값을 사용하여 처리됩니다.

가장 일반적으로는 **Motor[x].pCaptPos** 는 서보 IC 의 포착 위치 레지스터 주소에 설정됩니다. 예를 들어

Motor[1].pCaptPos = Gate1[4].Chan[0].HomeCapt.a

Motor[2].pCaptPos = Gate3[0].Chan[1].HomeCapt.a

\$\$\$***의 재초기화 명령에서는 **Motor[x].pCaptPos** 는 사전 정의된 순서 규칙에 따라서 검출된 서보 채널의 포착 위치 레지스터에 자동으로 할당됩니다.

설정 요소 **Motor[x].EncType** 의 값이 스크립트 환경에 의해 설정되는 경우(C 프로그램에서가 아니라)에 **Motor[x].pCaptPos** 는 **Motor[x].pEncStatus** 에서 사용되는 동일한 서보 채널의 포착 위치 레지스터의 주소에 자동으로 설정됩니다.

Motor[x].Pdi 모터의 사전 필터 분모항

설명: 사전 필터의 i 차 다항식의 분모항($i = 1 \sim 4$)

범위: 부동 소수점

단위: 없음(단위가 없는 z -변환 계수)

기본값: 0.0

Motor[x].Pdi 항은 서보 알고리즘에 전송되기 전의 명령 궤적 위치에 작용하는 4 차 다항식의 분모항의 배정도 계수입니다. 이 필터의 전달 함수는 다음과 같습니다.

$$P(z) = 1 - (1 - z^{-1}) \frac{Pn_0 + Pn_1 z^{-1} + Pn_2 z^{-2} + Pn_3 z^{-3} + Pn_4 z^{-4}}{1 + Pd_1 z^{-1} + Pd_2 z^{-2} + Pd_3 z^{-3} + Pd_4 z^{-4}}$$

해당 항을 사용하기 위해서는 모터의 사전 필터를 유효로 하기 위해 **Motor[x].PreFilterEna** 을 0 보다 큰 값으로 설정해야 합니다. **Motor[x].PreFilterEna** 의 값은 서보 인터럽트 주기에서의 필터 갱신 주기를 지정합니다.

필터가 궤적 "신호"에 어떠한 최종적인 스케일링 효과도 갖지 않도록 하기 위해서는(즉, 1.0 의 "DC 게인"을 가지도록) 전체 **Pni** 항의 합계값에서 전체 **Pdi** 항의 합계값을 뺀 값이 정확하게 1.0 과 같아야 합니다. 필요한 동특성을 얻기 위해 보통 **Pn1** 과 더 높은 항 그리고 **Pd1** 과 더 높은 항이 설정됩니다. 그때 **Pn0** 은 1.0 에서 더 고차인 **Pni** 항의 합계를 빼고 **Pdi** 항의 합계를 더한 값으로 설정됩니다. 필터에 의한 최종적인 스케일링은 없어집니다.

Motor[x].pDac 모터의 명령 출력 포인터

설명: 명령 출력의 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

기존 I 변수명: lxx02

Motor[x].pDac 은 선택된 모터에서 해당 명령 출력용으로 사용되는 레지스터(또는 1 세트의 레지스터의 처음)의 주소를 지정합니다. 여기에는 이 레지스터의 주소 또는 연속 주소의 여러 개 레지스터의 처음 주소(최하위 주소)가 저장됩니다. 이와 같이 명령 출력값을 전송하기 위해 어느 출력 신호가 사용되는지 결정합니다. 이 요소의 이름에도 불구하고 D/A 컨버터가 출력에 사용되어야 하는 것은 아닙니다. 이 점에 주의하십시오.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

가장 일반적으로 **Motor[x].pDac** 은 서보 IC 또는 MACRO IC 의 출력 레지스터 주소에 설정됩니다. 예를 들어

```
Motor[1].pDac = Gate1[4].Chan[0].Dac[0].a
Motor[2].pDac = Gate1[4].Chan[1].Pwm[0].a
Motor[3].pDac = Gate1[4].Chan[2].Pfm.a
Motor[4].pDac = Gate2[0].Macro[5][0].a
Motor[5].pDac = Gate3[0].Chan[0].Dac[0].a
Motor[6].pDac = Gate3[0].Chan[1].Pwm[0].a
Motor[7].pDac = Gate3[0].Chan[2].Pfm.a
Motor[8].pDac = Gate3[0].MacroOutA[8][0].a
```

재초기화 시 Power PMAC 은 존재하는 서보와 MACRO 인터페이스를 자동으로 식별하고 이용할 수 있는 인터페이스 채널의 주소를 **Motor[x].pDac** 요소에 할당합니다. 이용 가능한 인터페이스 채널이 없는 모터인 경우에 **Motor[x].pDac** 은 **Sys.pushm** 으로 설정되고 사용자 공유 메모리 버퍼의 선두 주소가 됩니다.

Motor[x].pEnc 모터의 위치 루프 피드백 포인터

설명: 아우터(위치) 루프의 위치 피드백의 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

기존 I 변수명: lxx03

Motor[x].pEnc 는 선택된 모터에서 아우터(보통은 위치) 루프의 피드백값에 사용되는 주소를 지정합니다. 여기에는 이 레지스터의 주소가 저장됩니다. 이와 같이 아우터 루프를 실행하기 위해 어떤 피드백 장치가 사용되는지를 결정합니다. 이 요소의 이름에도 불구하고 엔코더가 피드백에 사용되어야 하는 것은 아닙니다. 이 점에 주의하십시오.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 "a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

Motor[x].pEnc 는 엔코더 변환 테이블 항목의 주소로 설정되어야 합니다. 그때 아우터 루프는 그 항목이 처리된 결과를 사용할 수 있게 됩니다. 이를 수행하는 명령은 다음과 같습니다.

```
Motor[x].pEnc = EncTable[i].a
```

대부분의 경우 **x** 와 **i** 는 동일한 값이 됩니다. 그리고 이것은 보통 재초기화 시 자동으로 생성되는 기본설정이지만 그대로 할 필요는 없습니다.

예를 들어

Motor[1].pEnc = EncTable[2].a

서보 사이클마다 모터에서는 지정된 레지스터가 읽고 이 값에는 이 서보 사이클 간의 위치 변화가 포함됩니다. 이 값은 **Motor[x].PosSf** 로 곱하고 직전의 사이클의 모터 현재 위치의 값에 더합니다.

Motor[x].PhaseCtrl 의 비트 3 을 1 로 설정하고 모터의 서보 루프가 페이즈 인터럽트로 실행되는 경우에 아우터 루프의 피드백 레지스터가 대신 **Motor[x].pPhaseEnc** 로 지정된다는 점에 주의하십시오.

Motor[x].pEnc2 모터의 속도 루프 피드백 포인터

설명: 이너(속도) 루프의 위치 피드백의 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

기존 I 변수명: lxx04

Motor[x].pEnc2 는 선택된 모터에서 이너(보통은 속도) 루프의 피드백값에 사용되는 레지스터를 지정합니다. 여기에는 이 레지스터의 주소가 저장됩니다. 이와 같이 이너 루프를 실행하기 위해 어떤 피드백 장치가 사용되는지를 결정합니다. 이 요소의 이름에도 불구하고 엔코더가 피드백에 사용되어야 하는 것은 아닙니다. 이 점에 주의하십시오.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

Motor[x].pEnc2 는 엔코더 변환 테이블 항목의 주소로 설정되어야 합니다. 그때 이너 루프는 그 항목이 처리된 결과를 사용할 수 있게 됩니다. 이를 수행하는 명령은 다음과 같습니다.

Motor[x].pEnc2 = EncTable[i].a

대부분의 경우 **x** 와 **i** 는 동일한 값입니다. 그리고 이것은 보통 재초기화 시 자동으로 생성되는 기본설정이지만 그대로 할 필요는 없습니다.

예를 들어

Motor[1].pEnc2 = EncTable[2].a

서보 사이클마다 모터에서는 지정된 레지스터가 읽고 이 값에는 이 서보 사이클 간의 위치 변화가 포함됩니다. 이 값은 **Motor[x].Pos2Sf** 로 곱하고 직전의 사이클의 모터 현재 위치의 값에 더합니다.

대부분의 어플리케이션에서 이 변수는 **Motor[x].pEnc** 와 동일한 값으로 설정됩니다. 따라서 이너와 아우터 루프의 양쪽은 동일한 값을 사용합니다. 그러나 이 변수는 듀얼 루프 피드백을 수행하기 위한 간단한 방법을 제공합니다. 일반적으로 이너 루프에서는 최대한의 안정성을 얻기 위해 모터의 엔코더 또는 리졸버가 사용되고 아우터 루프에서는 최대한의 정밀도를 얻기 위해 부하측 센서가 사용됩니다.

Motor[x].PhaseCtrl의 비트 3을 1로 설정하고 모터의 서보 루프가 페이즈 인터럽트로 실행되는 경우에 이너 루프의 피드백 레지스터가 대신 **Motor[x].pPhaseEnc**로 지정된다는 점에 주의하십시오.

Motor[x].pEncCtrl 모터의 엔코더 포착 제어 포인터

설명: 엔코더 포착 제어 플래그의 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

Motor[x].pEncCtrl에서는 선택된 모터에서 포착 트리거 제어 신호에 사용되는 레지스터를 지정합니다. 여기에는 이 레지스터의 주소가 저장됩니다. 현재 이 레지스터는 MACRO 링 상의 트리거를 준비하기 위해서만 사용됩니다. Power PMAC은 원점 서치 동작과 같은 트리거 포함 동작에 이 레지스터를 사용합니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

가장 일반적으로는 **Motor[x].pEncCtrl**은 서보 IC의 채널 제어 레지스터의 주소 또는 MACRO 노드의 출력 레지스터 주소로 설정됩니다. 예를 들어

Motor[1].pEncCtrl = Gate1[4].Chan[0].Ctrl.a

Motor[2].pEncCtrl = Gate3[0].Chan[1].InCtrl.a

Motor[3].pEncCtrl = Gate2[0].Macro[4][3].a

Motor[4].pEncCtrl = Gate3[2]. MacroOutA[8][3].a

\$\$\$***의 재초기화 명령에서는 **Motor[x].pEncCtrl**은 사전 정의된 순서 규칙에 따라서 검출된 서보 채널 또는 MACRO 노드의 Status 레지스터에 자동으로 할당됩니다.

Motor[x].pEncLoss 모터의 엔코더 소실 포인터

설명:	센서 소실 레지스터의 포인터
범위:	적정한 주소
단위:	데이터 구조체 요소 주소
기본값:	0(엔코더 소실 검출 기능은 무효)

Motor[x].pEncLoss 는 그 모터에서 "엔코더 소실" 비트를 감시하는 데 사용되는 레지스터를 지정합니다. 여기에는 이 레지스터의 주소가 저장됩니다. **Motor[x].pEncLoss** 가 기본값 0 인 경우에 이 모터에서는 자동 엔코더 소실 검출 기능은 사용되지 않습니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

Power PMAC 은 엔코더 소실 상태를 읽기 위해 지정된 레지스터의 **Motor[x].EncLossBit** 로 설정된 번호의 비트를 사용합니다. 엔코더 소실 Status 비트의 극성은 **Motor[x].EncLossLevel** 로 설정됩니다. 에러 발생이 단정되기 전에 이상 입력 상태가 검출되는 스캔의 누적 수가 **Motor[x].EncLossLimit** 로 지정됩니다.

엔코더 소실 이상이 검출된 경우 이 모터는 Power PMAC 에 의해 자동으로 "kill"(Disable)됩니다. 동일한 좌표계의 다른 모터는 kill 되거나 "Abort" 됩니다(감속하여 제어된 정지 상태로). 이것은 이 모터의 **Motor[x].FaultMode** 의 비트 1(값 2)로 결정됩니다.

ASIC 의 바깥쪽에 "배타적 논리합" 직교(AB 상) 소실 검출 회로를 가지는 ACC-24E2x PMAC 스타일 축 인터페이스 보드에 연결된 직교(AB 상) 엔코더에서는 **Motor[x].pEncLoss** 는 **Gate1[j].Chan[j].EncLossN.a** 로 설정할 수 있습니다(대신 **ACC24E2x[j].Chan[j].EncLossN.a** 의 보드 구조체 별칭도 사용할 수 있습니다). 이는 실제로는 ASIC 중에 없으므로 복귀값으로 **Cid[n].PartData[j].a** 가 통지됩니다. 엔코더 소실 Status 비트는 이 레지스터의 비트 13에 있습니다.

일부 프로토콜에 좌우되는 소실 검출 회로를 가지는 ACC-84E PMAC2 스타일 시리얼 엔코더 인터페이스 보드에 연결된 시리얼 엔코더에서는 **Motor[x].pEncLoss** 는 **ACC84E[j].Chan[j].SerialEncDataB.a** 로 설정할 수 있습니다. 가장 일반적으로는 이 레지스터의 비트 31에서 "타임아웃" 에러 비트가 사용됩니다.

ASIC 의 "배타적 논리합" 직교(AB 상) 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 직교(AB 상) 엔코더에서는 **Motor[x].pEncLoss** 는 **Gate3[j].Chan[j].Status.a** 로 설정할 수 있습니다(대신 **ACC24E3[j].Chan[j].Status.a** 의 보드 구조체 별칭도 사용할 수 있습니다). 이 레지스터의 비트 28에 **LossStatus** 의 투과적인 AB 상 소실 비트가 있습니다.

"제곱합"의 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 아날로그 사인파 엔코더 또는 리졸버에서는 **Motor[x].pEncLoss** 는 **Gate3[j].Chan[j].Status.a** 로 설정할 수 있습니다(대신 **ACC24E3[j].Chan[j].Status.a** 의 보드 구조체 별칭도 사용할 수 있습니다). 이 레지스터의 비트 31 에 **SosError** 의 투과적인 "제곱합" 에러 비트가 있습니다.

일부 프로토콜에 좌우되는 소실 검출 회로를 가지는 ACC-24E3 PMAC3 스타일 축 인터페이스 보드에 연결된 시리얼 엔코더에서는 **Motor[x].pEncLoss** 는 **Gate3[j].Chan[j].SerialEncDataB.a** 로 설정할 수 있습니다(대신 **ACC24E3[j].Chan[j].SerialEncDataB.a** 의 보드 구조체 별칭도 사용할 수 있습니다). 가장 일반적으로는 이 레지스터의 비트 31 에서 "타임아웃" 에러 비트가 사용됩니다.

그 비트 자체의 요소명(예를 들어 **Motor[5].pEncLoss = Gate3[1].Chan[0].LossStatus.a**)이 있는 경우에 레지스터 내의 지정 비트 주소를 지정하는 것도 가능하지만 값은 풀 워드 요소의 주소(예를 들어 **Gate3[1].Chan[0].Status.a**)로 통지되므로 주의하십시오.

Motor[x].pEncStatus 모터의 부모 입력 플래그 포인터

설명: 모터 "부모" 입력 플래그의 포인터

범위: 적당한 주소

단위: 데이터 구조체 요소 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

기준 I 변수명: lxx25

Motor[x].pEncStatus에서는 모터의 입력 플래그용 "부모" 레지스터를 지정합니다. 여기에는 이 레지스터의 주소가 저장됩니다. 이것은 2 가지 주요한 기능에 도움을 줍니다.

첫 번째는 설정에서 **Motor[x].EncType** 이 스크립트 환경에서 설정될 때 포착 플래그 포인터 요소 **Motor[x].pCaptFlag** 는 **Motor[x].pEncStatus** 와 동일한 주소에 자동으로 설정됩니다. 그리고 **Motor[x].pEncStatus** 가 서보 IC 의 **Status** 레지스터를 지시하는 경우에 포착 위치 포인터 요소 **Motor[x].pCaptPos** 는 동일 채널의 하드웨어 포착 레지스터 주소에 자동으로 설정됩니다.

두 번째는 동작에서 모터의 상태와 안전 갱신 시 **Motor[x].pEncStatus** 로 지정된 레지스터의 값을 처음에 읽습니다. 다음으로 개별 입력 플래그를 확인해야 하지만 해당 플래그의 포인터 요소 - **Motor[x].pLimits**, **Motor[x].pAmpFault**, **Motor[x].pCaptFlag**, **Motor[x].pEncLoss** -가 **Motor[x].pEncStatus** 와 동일한 주소를 가지는 경우에 그(하드웨어) 레지스터는 다시 읽지는 않습니다. 대신 이미 읽어와서 메모리에 복사된 레지스터값을 사용합니다. 대부분의 경우에 전체는 아니지만 일부 입력 플래그가 동일한 레지스터에 할당되었지만 이 경우 상당한 액세스 시간을 절약할 수 있습니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

가장 일반적으로는 **Motor[x].pEncStatus**에서는 서보 IC의 채널 Status 레지스터의 주소 또는 MACRO 노드의 입력 플래그 레지스터로 설정됩니다. 예를 들어

Motor[1].pEncStatus = Gate1[4].Chan[0].Status.a

Motor[2].pEncStatus = Gate3[0].Chan[1].Status.a

Motor[3].pEncStatus = Gate2[0].Macro[4][3].a

Motor[4].pEncStatus = Gate3[2].MacroInA[8][3].a

\$\$\$***의 재초기화 명령에서는 **Motor[x].pEncStatus**는 사전 정의된 순서 규칙에 따라서 검출된 서보 채널 또는 MACRO 노드의 Status 레지스터에 자동으로 할당됩니다.

Motor[x].PhaseCtrl 모터의 전류(commutation) 제어

설명: 전류(commutation) 태스크를 유효로 하는 제어 플래그

범위: 0..15

단위: 없음

기본값: 0

기준 I 변수명: lxx01

Motor[x].PhaseCtrl은 페이즈(위상) 인터럽트에서 처리되는 모터의 태스크를 결정하고 그 태스크의 실행 방법도 결정합니다. 이것은 4개의 독립된 제어 비트의 집합입니다.

비트 0(값 1)에서는 Power PMAC이 "압축(팩)된" 데이터를 사용하여 모터의 전류(commutation)를 실행할지 여부를 제어합니다. 이 비트를 1로 설정한 경우에 Power PMAC은 페이즈(위상) 인터럽트마다 모터의 전류(commutation) 알고리즘을 실행합니다. 여기서는 입력값으로 서보 명령 출력을 사용하고 나아가 하나의 32비트 레지스터 중에서 16비트값으로 "압축된" 두 상의 정보를 사용하여 ASIC에서 /로 데이터를 전송합니다. 이것은 PMAC3 스타일 32비트 ASIC(DSPGATE3)의 기본값 모드입니다. 압축 전송은 상당한 액세스 시간을 절약합니다. 이 모드는 PMAC2 스타일 ASIC 또는 MACRO 전송에서는 사용할 수 없습니다. 비트 2가 1로 설정되는 경우에 이 비트는 1로 설정할 수 없습니다.

비트 1(값 2)에서는 슬립 연산이 AC 유도 모터의 전류(commutation) 또는 다이렉트 마이크로 스텝핑 제어에서 어떻게 실행되는지 제어합니다. 비트 0 또는 비트 2가 1로 설정되는 경우에만 사용됩니다. 비트 1이 기본값 0인 경우에 슬립 연산에서는 측정된 전류값이 사용됩니다. 비트 1이 1로 설정되는 경우에 슬립 연산에서는 명령 전류값이 사용됩니다(Turbo PMAC에서는 항상 측정된 전류값이 사용되었습니다). 다이렉트 마이크로 스텝핑 제어에서는 이 비트는 1로 설정되어야 합니다.

비트 2(값 4)에서는 Power PMAC 이 "비압축(언팩)된" 데이터를 사용하여 모터의 전류(commutation)를 실행할지 여부를 제어합니다. 이 비트를 1로 설정한 경우에 Power PMAC 은 페이즈(위상) 인터럽트마다 모터의 전류(commutation) 알고리즘을 실행합니다. 여기서는 입력값으로 서보 명령 출력을 사용하고 개별 레지스터에서 각 상의 정보를 사용하여 ASIC 에서/로 데이터를 전송합니다. 이 모드는 PMAC2 스타일 ASIC 또는 MACRO 전송에서 사용되어야 합니다. 비트 0 이 1로 설정되는 경우에 이 비트는 1로 설정할 수 없습니다.

Motor[x].PhaseCtrl 의 비트 3(값 8)이 1로 설정되는 경우에 이 모터에서는 페이즈 인터럽트에서 위치/속도 서보 루프를 실행합니다. 이는 "고속 톨 서보" 또는 직류 전기 구동과 같은 일부 Power PMAC 모터에서 시스템 내의 다른 모터보다 실질적으로 높은 주파수로 그 루프를 실행하는 것을 가능하게 합니다. 이 설정에서는 표준 **Motor[x].pEnc** 와 **Motor[x].pEnc2** 대신 아우터(위치) 루프와 이너(속도) 루프 양쪽에 피드백의 소스로 **Motor[x].pPhaseEnc** 로 지정된 레지스터를 사용합니다.

이 경우에 Power PMAC 은 모터의 페이즈 인터럽트로 간결한 보간을 실행하고 표준 서보 인터럽트 보간에서의 이동 포인트 사이에서 스플라인을 계산합니다. 스플라인의 다항식의 차수는 **Motor[x].PhaseSplineCtrl** 에서 설정됩니다. 이 보간을 적절하게 기능시키기 위해 **Sys.PhaseOverServoPeriod** 요소에서는 서보 주기에 대한 페이즈 주기의 비율을 설정해야 합니다.

따라서 **Motor[x].PhaseCtrl** 이 8로 설정되는 경우에 이 모터는 페이즈 인터럽트에서 서보 태스크를 실행하지만 모터의 페이즈 전류(commutation) 태스크를 실행하지 않습니다. 이 경우에 전류 피드백의 ADC 읽기를 무효로 하기 위해 **Motor[x].pAdc** 는 0으로 설정되는 것이 바람직합니다(이 경우에 ADC 를 읽는 것은 CPU 시간을 낭비하게 하지만 계산에는 영향을 주지 않습니다. 읽은 ADC 값은 사용되지 않기 때문입니다).

Motor[x].PhaseCtrl 이 9로 설정되는 경우에 이 모터에서는 페이즈 인터럽트에서 서보 태스크를 실행하고 페이즈 전류(commutation) 태스크도 실행합니다. 이 경우에 **Motor[x].pAdc** 의 설정은 전류 루프가 Power PMAC 에서 실행되는지 여부를 결정합니다. 그 루프도 실행하는 서보 태스크는 페이즈 태스크 전에 실행되고 표준 제어 모드에서 존재하는 1 페이즈 사이클의 "전송 지연"을 제거합니다.

Motor[x].PhaseCtrl 은 폴 워드 요소 **Motor[x].Control[0]**의 비트 24-27의 부분적인 구성 요소입니다.

Motor[x].PhaseEncLeftShift 모터의 페이즈 위치 좌측 시프트

설명: 페이즈 인터럽트에서의 위치 소스 데이터를 좌측 시프트할 비트 수

범위: 0 ..31

단위: 비트

기본값: 0

Motor[x].PhaseEncLeftShift 는 Power PMAC 이 **Motor[x].pPhaseEnc** 에서 주소 지정되는 32 비트 레지스터의 위치의 값을 먼저 **Motor[x].PhaseEncRightShift** 로 지정된 값으로 우측 시프트한 후에 다음으로 좌측 시프트할 비트 수를 지정합니다.

이 조작의 목적은 소스 레지스터의 실제 데이터의 최상위 비트(MSB)를 최종적인 32 비트값의 MSB에 배치하여 전류(commutation) 또는 서보 알고리즘 내의 소스 데이터를 오버의 적절한 처리를 가능하게 하는 것입니다.

전류(commutation) 페이즈 위치 정보의 가장 일반적인 소스는 DSPGATE1 과 DSPGATE3 ASIC 의 엔코더 카운터의 "페이즈 포착" 레지스터이며 이미 레지스터의 MSB에 실제 데이터인 MSB가 배치됩니다. 따라서 시프트는 필요 없고 그 소스에 대해 **Motor[x].PhaseEncLeftShift** 를 기본값 0 그대로 둘 수 있습니다. 시프트 조작은 전류(commutation) 알고리즘에 어떤 계산 시간도 더하지 않는다는 점에 주의하십시오.

그러나 그렇지 않은 몇 가지 소스가 있으며 그 경우 시프트 조작도 필요해집니다. 예를 들어 Panasonic 과 Tamagawa 의 시리얼 엔코더는 단일 레지스터에서 17 비트의 싱글턴 위치 정보를 제공합니다. ACC-24E3 과 Power Brick 제어 보드의 DSPGATE3 IC에서는 이는 **Gate3[j].Chan[j].SerialEncDataA** 레지스터의 비트 0-16 이 됩니다. 그리고 **Motor[x].PhaseEncLeftShift** 는 15 로 설정하고 이 MSB 를 결과의 비트 31 에 시프트시켜야 합니다. ACC-84E 에서 이는 **Acc84E[j].Chan[j].SerialEncDataA** 레지스터의 비트 8-24 가 됩니다. 그리고 **Motor[x].PhaseEncLeftShift** 는 7 로 설정하고(**Motor[x].PhaseEncRightShift** 의 값을 더해서) MSB 를 결과의 비트 31 에 시프트시켜야 합니다.

Motor[x].PhaseEncRightShift 모터의 페이즈 위치 우측 시프트

설명: 페이즈 위치의 소스 데이터를 우측 시프트할 비트 수

범위: 0 ..31

단위: 비트

기본값: 0

Motor[x].PhaseEncRightShift 는 Power PMAC 이 **Motor[x].pPhaseEnc** 에서 주소 지정되는 32 비트 레지스터의 위치값과 초기 처리 단계에서 우측 시프트할 비트 수를 지정합니다. 이 조작의 목적은 32 비트 레지스터에서의 실제 데이터의 최하위 비트(LSB)를 중간값인 비트 0에 배치하도록 시프트시켜 32 비트 소스 레지스터의 하위 비트에 있는 "불필요한 데이터"를 제거하는 것입니다. 그것을 수행하기 위해서는 **Motor[x].PhaseEncRightShift** 는 32 비트 소스 워드 중 실제 데이터인 LSB 의 비트 번호로 설정되어야 합니다.

Power PMAC 의 전류(commutation) 참조 "사인파" 테이블에는 2048(2^{11}) 항목이 있다는 점에 주의하십시오. 소스 데이터에 전류(commutation) 사이클당 적어도 11 비트(2048 "카운트")의 정확한 상태가 있는 경우에 참조 테이블 조작은 자동으로 하위 비트를 제거하므로 이 시프트 동작은 필요 없습니다. 이것은 가장 일반적인 경우입니다. 그러나 시프트 조작은 전류(commutation) 알고리즘에 어떤 계산 시간도 더하지 않습니다.

Motor[x].PhaseEncLeftShift 로 지정된 "좌측 시프트" 조작이 이 우측 시프트 후에 이어집니다. 그 결과는 소스 레지스터에서의 실제 데이터의 최상위 비트(MSB)를 최종적인 32 비트값의 MSB에 배치할 필요가 있고 소스 데이터를 오버의 적절한 처리를 가능하게 합니다.

Motor[x].PhaseFindingDac 모터의 페이즈 서치 출력

설명: 페이즈 서치 출력의 크기

범위: 음이 아닌 부동 소수점

단위: 16 비트 출력의 LSB

기본값: 0.0

기준 I 변수명: lxx73

Motor[x].PhaseFindingDac 은 Power PMAC 에서 전류(commutation)되는 동기 모터에 사용되는 전원 셋업 시의 페이즈 서치 동작에서의 출력 크기를 지정합니다. 동기 모터(영구자석 브러시리스 서보 모터와 같은)에는 전원 셋업 시의 페이즈 참조를 확립하는 방법이 필요해집니다. 절대 위치 센서 읽기 또는 페이즈 서치 동작 중 하나가 됩니다. 페이즈 서치 동작에는 이 파라미터는 0 보다 큰 값으로 설정해야 합니다. 또한 절대 위치 센서의 읽기에는 이 파라미터는 0 으로 설정해야 합니다.

이 요소의 이름에도 불구하고 D/A 컨버터가 출력에 사용될 필요가 없다는 점에 주의하십시오.

현재 2 가지 페이즈 서치 방식이 지원됩니다. "4 중 추정" 방식(Turbo PMAC 의 "2 중 추정" 방식을 업그레이드한 것)과 "스테퍼 모터" 방식입니다. 4 중 추정 방식은 **Motor[x].PhaseFindingDac** 에 전류(commutation) 페이즈 각도의 4 중 추정 각각에 사용되는 전류(commutation) 알고리즘에 토크 명령 입력 크기를 지정하고 **Motor[x].PhaseFindingTime** 이 지정한 기간 동안 그 크기는 유지됩니다.

스테퍼 모터 방식은 **Motor[x].PhaseFindingDac** 에 상에 강제 출력되는 상전류의 크기가 지정되고 스텝 모터와 같이 모터를 이미 알고 있는 전류(commutation) 페이즈 각도에 고정하기 위해 **Motor[x].PhaseFindingTime** 이 지정한 기간 동안 이 크기는 일정 비율로 증가합니다.

Motor[x].PhaseFindingTime 모터의 페이즈 서치 시간

설명: 페이즈 서치 시간

범위: 0 ..32,767

단위: 실시간 인터럽트 주기

기본값: 0

기준 I 변수명: lxx74

Motor[x].PhaseFindingTime 은 페이즈(위상) 서치 동작의 각 스텝에서의 계속 시간을 지정하고 나아가 페이즈 서치 동작이 수행되는지 여부를 지정하고 만약 수행된다면 사용되는 방식도 지정합니다. 단위는 (**Sys.RTIntPeriod+1**) 서보 사이클 시간의 실시간 인터럽트(RTI) 주기가 됩니다.

Motor[x].PhaseFindingTime 이 기본값 0 또는 1~3 의 값으로 설정되는 경우에 페이즈 서치 동작은 실행되지 않습니다.

Motor[x].PhaseFindingTime 이 4~255 의 값으로 설정되는 경우에 "4 중 추정" 페이즈 서치 방식이 사용되고 이 파라미터에서는 각 "추정"이 유지되는 RTI 사이클 수를 지정합니다. 이 방법에서 Power PMAC 은 정확한 로터 페이즈 각도에 대해 4 중 추정을 수행하고(각각 90 도 떨어진) **Motor[x].PhaseFindingDac** 으로 지정한 크기로 각 추정의 오픈 루프 출력을 명령하고 **Motor[x].PhaseFindingTime** 동안 이 추정을 유지합니다. 각 추정에서의 모터 동작을 평가하고 가장 좋은 해당 응답과 일치하는 로터 페이즈 각도를 산출합니다.

Motor[x].PhaseFindingTime 이 256 이상으로 설정되는 경우에 "스테퍼 모터" 페이즈 서치 방식이 사용됩니다. 그리고 이 파라미터는 각 상의 강제 전류가 유지되는 RTI 사이클 수를 지정합니다. 이것은 2 단계 처리입니다. 그리고 지정된 시간은 처리의 각 단계에서의 시간이 됩니다. 시간은 적절한 페이즈 서치 동작에 필요한 허용차 이내(일반 1 또는 2 도)에 모터를 확실하게 정정할 수 있게 충분히 커야 합니다.

개별 모터 상태의 갱신이 실시간 인터럽트 마다 실행될 수 있도록 저장 설정 요소 **Sys.MotorsPerRtInt** 가 0 이외의 값으로 설정되는 경우(2014 년 1/4 분기 출시된 V1.6 펌웨어의 새로운 기능)에 **Motor[x].PhaseFindingTime** 의 스케일링은 제로인 기본값으로 설정되는 경우 또는 파라미터 **Sys.MotorsPerRtInt** 가 없었던 이전 펌웨어 버전의 경우와 다릅니다.

Sys.MotorsPerRtInt>0 인 경우에 **Motor[x].PhaseFindingTime** 을 수정하기 위해 필요한 시간 확장 계수 N 은 다음 식으로 구합니다.

$$N = \text{ceil}\left(\frac{\text{Sys.ServoMotors} + 1}{\text{Sys.MotorsPerRtInt}}\right)$$

"*ceil*"은 다음 정수에 올림 처리하는 것(필요한 경우)을 나타내는 "CEILING" 함수입니다. 그리고 상태 요소 **Sys.ServoMotors** 는 유효한 모터의 최대 번호입니다. 실시간 인터럽트마다 실행되는 경우와 비교하여 **Motor[x].PhaseFindingTime** 의 값은 N 으로 나뉘어야 합니다.

Motor[x].PhaseLoadEncLeftShift 모터의 페이즈 부하 위치 좌측 시프트

설명:	페이즈 인터럽트에서의 부하 위치 소스 데이터를 좌측 시프트할 비트 수
범위:	0 ..31
단위:	비트
기본값:	0

Motor[x].PhaseLoadEncLeftShift에서는 Power PMAC 이 **Motor[x].pPhaseLoadEnc**에서 주소 지정되는 32 비트 레지스터의 위치의 값을 먼저 **Motor[x].PhaseLoadEncRightShift**로 지정된 값으로 우측 시프트한 후에 다음으로 좌측 시프트할 비트 수를 지정합니다. 이 조작의 목적은 소스 레지스터의 실제 데이터의 최상위 비트(MSB)를 최종적인 32 비트값의 MSB에 배치하는 것이고 페이즈 인터럽트 아래에서 실행되는 서보 알고리즘 내의 소스 데이터를 오버의 적절한 처리를 가능하게 하는 것입니다.

Motor[x].PhaseCtrl의 비트 3(값 8)이 1로 설정되고 페이즈 인터럽트 아래에서 서보 루프가 실행되는 경우에 **Motor[x].pPhaseLoadEnc**가 0 이외의 값으로 설정되고 다른 부하 위치 피드백을 지정하는 경우에만 **Motor[x].PhaseLoadEncLeftShift**는 사용됩니다.

Motor[x].PhaseLoadEncRightShift가 0보다 큰 값으로 설정되고 정의되지 않은 데이터의 하위 비트를 시프트 아웃하는 경우에 **Motor[x].PhaseLoadEncLeftShift**에서는 실제 데이터의 MSB를 최종적인 32 비트의 값인 비트 31에 배치하기 위해 적어도 우측 시프트값 이상의 값으로 설정해야 합니다. 예를 들어 DSPGATE1 서보 IC 채널의 "페이즈 포착" 레지스터가 사용되는 경우에 32 비트 버스의 비트 8-31에 실제 데이터가 있고 데이터는 처음에 8비트 우측 시프트된 후 실제 데이터의 MSB를 최종적인 데이터의 비트 31에 배치하도록 8비트 좌측 시프트됩니다. 그리고 최종적인 데이터의 하위 8비트는 모두 0이 됩니다.

다른 경우에는 실제 데이터의 MSB가 소스 레지스터의 비트 31에서 검출되지 않는 경우가 있습니다. 그 경우에 좌측 시프트값은 우측 시프트값보다 커야 합니다. 예를 들어 DSPGATE3 IC의 Panasonic과 Tamagawa 시리얼 엔코더 인터페이스에서는 소스 레지스터의 비트 0-16에서 17비트의 실제 데이터를 제공합니다. 우측 시프트는 필요 없습니다. 그러나 실제 데이터의 MSB를 비트 31에 배치하도록 **Motor[x].PhaseLoadEncLeftShift**는 15로 설정되어야 합니다.

FPGA 기반 ACC-84E의 Panasonic과 Tamagawa 시리얼 엔코더 인터페이스에서는 소스 레지스터의 비트 8-24에서 17비트의 실제 데이터를 제공합니다. 8비트 우측 시프트 후에 실제 데이터의 MSB를 비트 31에 배치하도록 **Motor[x].PhaseLoadEncLeftShift**는 15로 설정되어야 합니다.

Motor[x].PhaseLoadEncRightShift 모터의 페이즈 부하 위치 우측 시프트

설명:	페이즈 인터럽트에서 부하 위치 소스 데이터를 우측 시프트할 비트 수
범위:	0 ..31
단위:	비트
기본값:	0

Motor[x].PhaseLoadEncRightShift 는 Power PMAC 이 **Motor[x].pPhaseLoadEnc** 에서 주소 지정되는 32 비트 레지스터의 위치값과 초기 처리 단계에서 우측 시프트할 비트 수를 지정합니다. 이 조작의 목적은 32 비트 레지스터에서의 실제 데이터의 최하위 비트(LSB)를 중간값인 비트 0에 배치하도록 시프트시켜 32 비트 소스 레지스터의 하위 비트에 있는 "불필요한 데이터"를 제거하는 것입니다. 그것을 수행하기 위해서는 **Motor[x].PhaseLoadEncRightShift** 는 32 비트 소스 워드 중 실제 데이터인 LSB의 비트 번호로 설정되어야 합니다.

예를 들어 DSPGATE1 IC의 "페이즈 포착" 엔코더 카운터 레지스터는 32 비트 버스의 비트 8-31에 배치되며 프로세서가 그것을 읽을 때는 버스의 하위 8 비트는 정의되지 않습니다. 이 경우에 **Motor[x].PhaseLoadEncRightShift** 는 8로 설정하고 정의되지 않은 하위 8 비트를 "시프트 아웃"시킵니다.

Motor[x].PhaseLoadEncLeftShift로 지정된 "좌측 시프트" 조작은 이 우측 시프트 후에 이어집니다. 그 결과는 소스 레지스터에서의 실제 데이터의 최상위 비트(MSB)를 최종적인 32 비트값의 MSB(비트 31)에 배치할 필요가 있고 소스 데이터를 롤 오버의 적절한 처리를 가능하게 합니다.

Motor[x].PhaseCtrl의 비트 3(값 8)이 1로 설정되고 페이즈 인터럽트 아래에서 서보 루프가 실행되는 경우에 **Motor[x].pPhaseLoadEnc**가 0 이외의 값으로 설정되고 다른 부하 위치 피드백을 지정하는 경우에만 **Motor[x].PhaseLoadEncRightShift**는 사용됩니다.

Motor[x].PhaseMode 모터의 전류(commutation) 모드 제어

설명:	전류(commutation) 모드 제어 비트
범위:	0 ..3
단위:	비트 필드
기본값:	0

Motor[x].PhaseMode는 어떤 페이즈 전류(commutation) 태스크가 어떻게 실행되는지 지정합니다. 이것은 2비트값으로 전류(commutation)가 유효하며(**Motor[x].PhaseCtrl**의 비트 0 또는 비트 2 = 1) 디지털 전류 루프가 유효인 경우(**Motor[x].pAdc>0**)에만 사용됩니다.

비트 0(값 1)에서는 다이렉트 PWM 전압 명령 출력을 산출하는 데 "3 차 고조파 주입법"이 사용되는지 여부를 지정합니다. 기본값 0 인 경우에 3 차 고조파 주입법은 유효가 됩니다. 1 로 설정되는 경우에 3 차 고조파 주입법은 무효가 됩니다.

제 3 상파 투입은 주어진 모터 상전압에 대해 필요한 버스 전압을 약 15% 줄이는 3 상 모터를 위한 기법입니다. 버스 포화에 도달하기 전에 이것은 현저히 높은 속도 및/(또는) 더 높은 전류 레벨을 가능하게 합니다. 따라서 3 상 모터(**Motor[x].PhaseOffset** = +/- 683)의 다이렉트 PWM 제어에는 비트 0 은 0 으로 설정되어야 합니다.

그러나 3 차 고조파 주입법은 2 상 모터의 필요한 버스 전압을 감소시키지 않습니다. 그리고 상당한 토크 리플을 도입합니다. 따라서 2 상 모터(**Motor[x].PhaseOffset** = +/- 512)의 다이렉트 PWM 제어에는 이 기능을 무효로 하도록 비트 0 은 1 로 설정되어야 합니다.

Motor[x].PhaseMode 의 비트 1(값 2)에서는 "다이렉트 전류"의 루프 적분기가 유효인지 여부를 지정합니다. 기본값 0 인 경우에 이 적분기는 유효가 됩니다. 이 설정은 완전 다이렉트 전류의 루프 기능이 필요해지는 다상 모터(예를 들어 브러시리스의 서보 모터와 AC 유도 모터)에 사용됩니다(예를 들어 다이렉트 전류 레벨을 강제적으로 0 으로 하는 경우라도).

비트 1 이 1 로 설정되는 경우에 다이렉트 전류의 루프 적분기는 무효가 됩니다. 이 설정은 DC 브러시 모터(**Motor[x].PhasePosSf** = 0.0 의 설정에서)의 다이렉트 PWM 제어에 필요합니다. 여기서는 "직교 전류"의 루프만 실제로 실행되고 노이즈나 오프셋에서의 다이렉트 전류의 "편차"를 적분하는 것은 적절한 제어를 방해할 수 있기 때문입니다.

Motor[x].PhaseMode 는 풀 워드 요소 **Motor[x].Control[0]**의 비트 6-7 의 부분적인 구성 요소입니다.

Motor[x].PhaseOffset 모터의 페이즈 오프셋

설명:	A 상과 B 상 간의 오프셋
범위:	-1024..1023
단위:	1/2048 의 전류(commutation) 사이클
기본값:	-683
기존 I 변수명:	lxx72

Motor[x].PhaseOffset 에서는 Power PMAC 에서 전류(commutation)되는 다상 모터의 상간 각도 거리를 지정합니다. **Motor[x].PhaseOffset** 의 단위는 전류(commutation) 사이클의 1/2048 입니다. 사용되는 일반적인 값은 다음과 같습니다.

3 상: +/-683(+/- 120°e)

2 또는 4 상: +/-512(+/- 90°e)

(단상)DC 브러시 모터의 디지털 전류 루프를 실행할 때 **Motor[x].PhaseOffset** 은 2 또는 4 상 모터용의 +/-512 로 설정합니다.

주어진 상 수에는 선택 가능한 2 개의 유효한 값이 있으며 아래 설명과 같이 그 선택은 출력 명령과 피드백 사이의 극성 일치에 의해 결정됩니다.

Power PMAC 이 전류(commutation)와 동시에 전류 루프 계산도 실행하는지에 따라 **Motor[x].PhaseOffset** 은 약간 다르게 사용됩니다. 두 사례를 아래에서 설명합니다.

1. PowerPMAC 은 전류(commutation)를 실행하지만 전류 루프를 실행하지 않는 경우

Power PMAC 이 모터의 디지털 전류 루프를 실행하지 않는 경우에 이 파라미터와 모터와 앰프 간의 상 배선으로 결정되는 출력 방향 검출은 엔코더 디코드 변수(보통 **Gaten[j].Chan[j].EncCtrl**)와 엔코더 배선으로 결정되는 피드백 방향 검출과 일치해야 합니다. 방향 검출이 일치하지 않는 경우에 적절한 전류(commutation)와 서보 제어는 불가능해지고 모터는 주어진 위치에서 록합니다.

이 시스템에서는 주어진 상 수에서의 2 개의 값 중에서 변경은 모터의 리드선 교환과 같은 효과가 있으며 이는 서보 토크 명령에 주어진 부호로 모터 회전 방향을 바꾸게 됩니다. 한번 이 전류(commutation)/피드백 극성이 적절하게 일치하면 서보/피드백 극성도 자동으로 적절하게 일치합니다(서보 루프가 동일한 피드백 센서를 사용하고 있다고 가정하지만).

2. PowerPMAC 은 전류(commutation)와 전류 루프를 실행하는 경우

Power PMAC 이 모터의 디지털 전류 루프를 실행하는 경우에 이 파라미터에서 결정되는 출력 방향 검출은 상전류 센서와 이 데이터를 Power PMAC 에 가져오는 아날로그/디지털 변환(ADC) 회로의 극성과 일치해야 합니다. 이것은 모터 또는 앰프상 배선, 엔코더 배선 및 Power PMAC 엔코더 디코드 방향 검출에서 독립됩니다.

Power PMAC 에서 디지털 전류 루프를 수행하는 시스템에서는 상전류 ADC 레지스터가 그 상에 유입되는 전류를 양의 값이라고 통지하는 경우(즉, PWM 전압 명령값 및 전류 피드백값이 동일한 부호를 가짐)에 **Motor[x].PhaseOffset** 은 음수로 설정되어야 합니다(보통 3 상 모터에는 -683 또는 2, 4 상 모터에는 -512). 상전류 ADC 레지스터가 그 상에서 유출되는 전류를 양의 값이라고 통지하는 경우(즉, PWM 전압 명령값 및 전류 피드백값은 반대 부호를 가짐)에 **Motor[x].PhaseOffset** 은 양의 값으로 설정되어야 합니다(보통 3 상 모터에는 683 또는 2, 4 상 모터에는 512).



주의

Motor[x].PhaseOffset 설정의 적절한 검출을 확신할 때까지 Power PMAC 에서 모터의 디지털 전류 루프를 실행하지 마십시오(오픈 또는 클로즈드 루프에서 모터를 Enable 하여). 잘못된 검출의 **Motor[x].PhaseOffset** 설정은 전류 루프의 정구환을 일으키고 PMAC 출력의 포화 그리고 모터 및(또는) 앰프 파손의 가능성으로 이어집니다.

Power PMAC 에서 디지털 전류 루프를 수행하는 시스템에서는 전류(commutation)/피드백 극성이 서보/피드백 극성과 독립됩니다. 한번 **Motor[x].PhaseOffset** 가 적절한 전류(commutation)/피드백 극성으로 설정되었다 하더라도 적절한 위치 루프 서보/피드백 극성 또한 확립시켜야 합니다.

Motor[x].PhaseOffset 이 3 상 모터의 ± 683 으로 설정되는 경우에 **Motor[x].PhaseMode** 의 비트 0(값 1)은 "3 차 고조파 주입법"을 유효로 하기 위해 0 으로 설정되어야 한다는 점에 주의하십시오. **Motor[x].PhaseOffset** 이 1-, 2- 또는 4-상 모터의 ± 512 로 설정되는 경우에 3 차 고조파 주입법은 적합하지 않으므로 이 비트는 1 로 설정되어야 합니다.

Motor[x].PhasePosSf 모터의 페이즈 위치 스케일 계수

설명: 전류(commutation) 위치(각도)의 스케일 계수

범위: 양의 부동 소수점(배정도)

단위: 1/2048 전류(commutation) 사이클/32 비트 소스 레지스터의 LSB

기본값: 0.008

기존 I 변수명: lxx71

Motor[x].PhasePosSf 는 1 전류(commutation) 사이클의 크기를 정의하고 **Motor[x].pPhaseEnc** 로 지정되는 전류(commutation) 위치 피드백의 32 비트 소스 레지스터의 데이터를 곱해서 전류(commutation) 사이클의 단위로 변환합니다. **Motor[x].PhaseEncShiftRight** 와 **Motor[x].PhaseEncShiftLeft** 를 지정하여(공통이 아닌) 소스 데이터의 시프트가 실행된 후 이 곱셈이 수행됩니다. Power PMAC 은 1 전류(commutation) 사이클을 2048 개로 분할합니다. 따라서 결과의 단위(**Motor[x].PhasePos** 에서의)는 전류(commutation) 사이클의 1/2048 이 됩니다.

1 전류(commutation) 사이클은 모터의 N-S 자극 페어와 등가입니다. 그래서 일반적인 4 극 회전 브러시리스 또는 유도 모터에는 1 회전당 2 개의 전류(commutation) 사이클이 있습니다. 그리고 1 전류(commutation) 단위는 기계적인 1 회전의 1/4096 과 등가가 됩니다.

Power PMAC 의 전류(commutation) 알고리즘은 모든 32 비트에 실제 데이터가 없어도 완전한 32 비트의 레지스터로서 **Motor[x].pPhaseEnc** 로 지정된 전류(commutation) 위치 레지스터를 항상 읽습니다. 예를 들어 PMAC2 스타일 "DSPGATE1" 서보 IC 에는 32 비트의 Power PMAC 데이터 버스의 상위 24 비트에 24 비트의 위상 위치 레지스터가 있습니다. 하위 8 비트에는 실제 데이터가 없습니다. 이는 1 엔코더 카운트가 프로세서에서 읽는 32 비트 수치의 256LSB 와 등가임을 의미합니다.

아무리 정수 자체가 되지 않아도 **Motor[x].PhasePosSf** 는 항상 2 개의 정수의 비로 실질적으로는 표시할 수 있다는 점에 주의하십시오. 특히 최종적인 값이 부동 소수점 값 또는 매우 긴 수치 표현이 되어 정확한 수치 표현을 할 수 없는 경우에 사용자는 두 정수의 비로 입력하는 것이 더 간단하다고 생각할 수도 있습니다.

DC 브러시 모터의 다이렉트 PWM 제어를 실행하는 경우에 Power PMAC 은 모터의 전류 루프를 실행하기 위해 전류(commutation) 알고리즘을 실행해야 합니다. 이 경우에 DC 모터 전류(commutation)의 AC 성을 실질적으로 무효로 하기 위해 **Motor[x].PhasePosSf** 는 0.0 설정합니다.

예

4 극 회전 브러시리스 모터에 PMAC2 스타일 "DSPGATE1" 서보 IC의 채널로 전송되는 500pulse의 엔코더가 있어서 "4 체배" 디코드를 실행하고 기계적으로 1 회전당 2000 카운트한다고 가정합니다. 4 극 모터에는 기계적인 1 회전당 2 전류(commutation) 사이클이 있으므로 1 전류(commutation) 사이클당 1000 카운트하게 됩니다. 카운트 데이터는 32 비트 데이터 버스의 상위 24 비트에 있으므로 1 카운트는 256 LSB 카운트에 상당하며 전류(commutation) 사이클당 256,000LSB가 됩니다. 스케일 계수는 다음과 같이 계산합니다.

$$Motor[x].PhasePosSf = \frac{2048units/comm - cyc}{256,000LSBs/comm - cyc} = 0.008 \left(\frac{units}{LSB} \right)$$

리니어 브러시리스 모터가 60.96mm(2.4 인치)의 극 페어 간격이 있어서 10 미크론(0.01mm) 피치의 사인파 엔코더가 PMAC3 스타일 'DSPGATE3' IC에 전송되고 보간되어 엔코더 1 라인당 16,384LSB로 보간된다고 가정합니다. 보간된 값의 LSB는 32 비트 레지스터의 LSB 위치에 배치됩니다. 60.96/0.01=6,096의 엔코더 라인/전류(commutation) 사이클이 되고 따라서 6,096*16,384=99,876,864의 LSB/전류(commutation) 사이클이 됩니다. 스케일 계수는 다음과 같이 계산합니다.

$$Motor[x].PhasePosSf = \frac{2048units/comm - cyc}{99,876,864LSBs/comm - cyc} = 0.000020505249 \left(\frac{units}{LSB} \right)$$

Power PMAC이 정확한 값을 계산할 수 있도록 이 값은 2048/99876864이라고 입력되는 것이 가장 좋습니다.

4 극의 회전 브러시리스 모터가 시리얼 엔코더가 있어서 ACC-84E 채널의 "A" 데이터 레지스터의 비트 8~24에서 17 비트의 싱글턴 데이터를 읽고 "B" 데이터 레지스터에서 멀티턴 데이터를 읽는다고 가정합니다. Power PMAC이 데이터의 롤 오버를 올바르게 처리할 수 있도록 싱글 턴 데이터의 최상위 비트(MSB)를 비트 31에 배치할 필요가 있으나 그러기 위해서는 **Motor[x].PhaseEncShiftRight**와 **Motor[x].PhaseEncShiftLeft**를 사용하여 최종적으로는 7 비트 우측 시프트해야 합니다. 이는 기계적인 회전당 레지스터의 2^{32} LSB가 있고 그리고 전류(commutation) 사이클당 2^{31} LSB가 있음을 의미합니다. 스케일 계수는 다음과 같이 산출할 수 있습니다.

$$Motor[x].PhasePosSf = \frac{2^{11} units/comm - cyc}{2^{31} LSBs/comm - cyc} = \frac{1}{2^{20}} = \frac{1}{1,048,576} \left(\frac{units}{LSB} \right)$$

Power PMAC이 정확한 값을 계산할 수 있도록 이 값은 1/1048576이라고 입력되는 것이 가장 좋습니다.

Motor[x].PhaseSplineCtrl 모터의 페이즈 스플라인 보간

설명:	페이즈 인터럽트에서의 스플라인 보간의 차수
범위:	0 ..3
단위:	다항식의 차수
기본값:	0

Motor[x].PhaseSplineCtrl 은 페이즈 인터럽트로 실행되는 스플라인 보간의 차수를 지정합니다. **Motor[x].PhaseCtrl** 의 비트 3(값 8)이 1 로 설정되는 경우에 페이즈 인터럽트로 서보 계산이 실행되고 그 후 스플라인 보간이 실행됩니다. 서보 계산이 페이즈 인터럽트로 실행되지 않은 경우에 **Motor[x].PhaseSplineCtrl** 은 사용되지 않습니다.

서보 계산이 페이즈 인터럽트로 실행되는 경우에 보간의 단순한 추가 스텝만이 페이즈 인터럽트로 실행됩니다. 표준 보간 계산은 더 저주파수의 서보 인터럽트로 다시 실행됩니다. Power PMAC 의 페이즈 인터럽트 루틴은 서보 인터럽트로 계산된 포인트 간의 단순한 스플라인 보간을 실행합니다. 지정할 수 있는 스플라인의 차수는 다음과 같습니다.

- **Motor[x].PhaseSplineCtrl** = 0: 0 차-포인트 사이에 보간 없음
- **Motor[x].PhaseSplineCtrl** = 1: 1 차-포인트 사이에서 일정 속도 보간
- **Motor[x].PhaseSplineCtrl** = 2: 2 차-포인트 사이에서 일정 가속 보간
- **Motor[x].PhaseSplineCtrl** = 3: 3 차-포인트 사이에서 일정 저크 보간

고차 스플라인은 더 원활해지지만 좀 더 "후미부"를 궤적에 추가하여 총 시간을 연장합니다. **Motor[x].PhaseSplineCtrl** 의 사용은 사용자가 원활성과 응답성 간의 최적의 트레이드 오프를 찾을 수 있게 합니다.

서보 계산이 페이즈 인터럽트로 수행되고 **Motor[x].PreFilterEna** 이 0 보다 큰 경우에는 **Motor[x].PhaseSplineCtrl** 의 설정과 관계없이 3 차 스플라인 보간이 페이즈 인터럽트에서 사용되므로 주의하십시오.

이 페이즈 스플라인 보간이 올바르게 기능하기 위해 글로벌 저장 설정 요소 **Sys.PhaseOverServoPeriod** 는 페이즈 주기와 서보 주기의 비가 올바르게 설정되어야 합니다.

Motor[x].pLimits 모터의 오버트래블 리미트 포인터

설명: 오버트래블 리미트(입력) 플래그의 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

(할당하는 하드웨어가 없는 경우에 0 이 되고 기능은 무효)

Motor[x].pLimits 는 선택된 모터에서 하드웨어 오버트래블 리미트 입력 신호에 사용되는 레지스터를 지정합니다. 여기에는 이 레지스터의 주소가 저장됩니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

가장 일반적으로는 **Motor[x].pLimits** 는 서보 IC 의 채널 Status 레지스터 주소로 설정됩니다. 예를 들어

Motor[1].pLimits = Gate1[4].Chan[0].Status.a

Motor[2].pLimits = Gate3[0].Chan[1].Status.a

모터가 MACRO 링을 경유하여 제어되는 경우 보통 **Motor[x].pLimits** 는 MACRO 노드 입력 플래그 레지스터(노드의 입력 레지스터 3) 주소로 설정됩니다. 예를 들어

Motor[3].pLimits = Gate2[0].Macro[4][3].a

Motor[4].pLimits = Gate3[0].MacroInA[5][3].a

PowerPMAC 은 양의 리미트 신호를 읽기 위해 지정된 레지스터의 **Motor[x].LimitBit** 로 설정된 번호의 비트를 사용합니다. 음의 리미트 신호는 다음 고위 번호의 비트로 읽습니다.

그 비트 자체의 요소명(예를 들어 **Motor[5].pLimits = Gate1[6].Chan[0].PlusLimit.a**)이 있는 경우에 레지스터 내의 지정 비트 주소를 지정하는 것도 가능하지만 값은 폴 워드 요소의 주소(예를 들어 **Gate1[6].Chan[0].Status.a**)로 통지되므로 주의하십시오.

\$\$\$***의 재초기화 명령에서는 Power PMAC 은 검출된 서보 IC 와 MACRO IC 에 따라서 **Motor[x].pLimits** 의 값을 자동으로 할당합니다.

Motor[x].pLimits 가 0 으로 설정되는 경우에 이 모터에서는 자동 하드웨어 오버트래블 리미트 기능은 사용되지 않습니다(단, 소프트웨어 오버트래블 리미트는 유효로 할 수 있습니다). 재초기화 시 모터로의 자동 할당으로 하드웨어가 검출되지 않는 경우에 이 요소는 자동으로 0 으로 설정되고 모터의 하드웨어 오버트래블 리미트 입력 기능을 무효로 합니다(2014 년 1/4 분기에 출시한 V1.6 펌웨어의 새로운 기능입니다. 이전 버전에서는 이 경우에 사용자 공유 메모리 버퍼의 시작 레지스터 **Sys.pushm** 에 할당되었습니다).

Motor[x].pMasterEnc 모터의 마스터 엔코더 포인터

설명: 마스터 위치 소스의 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: **EncTable[0].a**

기존 I 변수명: lxx05

Motor[x].pMasterEnc에서는 선택된 모터에서 위치 추종(전자 기어) 기능의 마스터 위치에 사용되는 레지스터를 지정합니다. 여기에는 이 레지스터의 주소가 저장됩니다. 이와 같이 어느 센서(또는 의사 센서)가 마스터로 사용되는지를 결정합니다. 이 요소의 이름에도 불구하고 엔코더가 마스터 위치에 사용될 필요가 없다는 점에 주의하십시오.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

추종 알고리즘이 엔코더 변환 테이블 항목의 처리 결과를 사용하므로 **Motor[x].pMasterEnc**는 변환 테이블 항목의 주소로 설정되어야 합니다. 이를 수행하는 명령은 다음과 같습니다.

Motor[x].pMasterEnc = EncTable[i].a

예를 들어

Motor[1].pMasterEnc = EncTable[0].a

서보 사이클마다 위치 추종이 유효한 경우(**Motor[x].MasterCtrl**의 비트 0 = 1)에 모터에서는 지정된 레지스터를 읽으며 이 값은 서보 사이클 간의 위치 변화를 포함합니다. 모터에서는 이 값에 **Motor[x].MasterPosSf**를 곱해서 이전 사이클의 최종적인 명령 위치의 값에 더합니다. 따라서 모터는 이 마스터 위치의 값의 변화를 추적하게 됩니다.

기본값에서는 각 모터의 **pMasterEnc**는 **EncTable[0]**의 주소로 설정되고 그 값은 기본값 0이 됩니다.

Motor[x].pMotorNode 모터의 네트워크 명령 포인터

설명: 사이클릭 네트워크 명령 소스의 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소, 0

기본값: **Sys.pushm**(기능 없음)

Motor[x].pMotorNode 는 모터가 네트워크 명령을 받게 설정된 경우에 최초의 사이클릭 네트워크 명령값에 사용되는 레지스터를 지정합니다. 여기에는 이 레지스터의 주소가 저장됩니다. **Motor[x].MotorMode** 가 0 보다 큰 값으로 설정되고 사이클릭 네트워크 명령을 받게 모터가 설정되는 경우에만 이 저장 요소가 사용됩니다. 그 명령의 의미는 **Motor[x].MotorMode** 의 값에 좌우됩니다.

이 요소의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 "a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다. 가장 일반적으로는 **Motor[x].pMotorNode** 는 PMAC2 스타일 또는 PMAC3 스타일 MACRO IC 의 서보 노드를 위한 최초의 MACRO 입력 레지스터의 주소로 설정됩니다.

UMAC ACC-5E 등의 PMAC2 스타일 "DSPGATE2" MACRO IC 의 설정은 다음과 같습니다.

Motor[x].pMotorNode = Gate2[j].Macro[j][0].a

i 는 IC 인덱스(0-15)이고 *j* 는 노드 번호(이 타입의 데이터를 전달하기 위한 "서보 노드"로 0, 1, 4, 5, 8, 9, 12 와 13 이 표준적으로 사용됩니다)입니다. **Gate2** 대신 실제 장치의 "별칭"(예를 들어 Acc5E)을 사용할 수 있습니다. 참조되면 이 요소의 값을 통지하는 데 적절한 별칭이 사용됩니다(예를 들어 **Acc5E[0].Macro[8][0].a**).

Power Brick 또는 UMAC ACC-5E3 등의 PMAC3 스타일 "DSPGATE3" MACRO IC 의 설정은 다음과 같습니다.

Motor[x].pMotorNode = Gate3[j].MacroIn α [j][0].a

i 는 IC 인덱스(0-15)이고 *j* 는 노드 번호입니다(이 타입의 데이터를 전달하기 위한 "서보 노드"로 0, 1, 4, 5, 8, 9, 12 와 13 이 표준적으로 사용됩니다). α 는 "A" 또는 "B"입니다. Gate3 대신 실제 장치의 "별칭"(예를 들어 PowerBrick)을 사용할 수 있습니다. 참조되면 이 요소의 값을 통지하는 데 적절한 별칭이 사용됩니다(예를 들어 **PowerBrick[1].MacroInA[1][0].a**).

Motor[x].Pni 모터의 사전 필터 분자항

설명: 사전 필터의 *i* 차 다항식의 분자항(*i* = 0~4)

범위: 부동 소수점

단위: 없음(단위가 없는 z- 변환 계수)

기본값: 1.0(Pn0), 0.0(Pn1 - Pn4)

Motor[x].Pni 항은 서보 알고리즘에 전송되기 전의 명령 궤적 위치에 작용하는 4 차 다항식의 분자항의 배정도 계수입니다. 이 필터의 전달 함수는 다음과 같습니다.

$$P(z) = 1 - (1 - z^{-1}) \frac{Pn_0 + Pn_1 z^{-1} + Pn_2 z^{-2} + Pn_3 z^{-3} + Pn_4 z^{-4}}{1 + Pd_1 z^{-1} + Pd_2 z^{-2} + Pd_3 z^{-3} + Pd_4 z^{-4}}$$

해당 항목을 사용하기 위해서는 모터의 사전 필터를 유효로 하기 위해 **Motor[x].PreFilterEna** 은 0 보다 큰 값으로 설정해야 합니다. **Motor[x].PreFilterEna** 의 값에는 서보 인터럽트 주기 단위의 필터 갱신 주기를 지정합니다.

표준 필터 설계에서는 **Pn4** 항목은 항상 0 으로 설정됩니다. 따라서 분자는 분모보다 높은 유효한 차수를 갖지 않습니다.

필터가 궤적 "신호"에 어떠한 최종적인 스케일링 효과도 갖지 않도록 하기 위해서는(즉, 1.0 의 "DC 게인"을 가지도록) 전체 **Pni** 항목의 합계값에서 전체 **Pdi** 항목의 합계값을 뺀 값이 정확하게 1.0 과 같아야 합니다. 필요한 동특성을 얻기 위해 보통 **Pn1** 과 더 높은 항 그리고 **Pd1** 과 더 높은 항이 설정됩니다. 그때 **Pn0** 은 1.0 에서 더 고차인 **Pni** 항목의 합계를 빼고 **Pdi** 항목의 합계를 더한 값으로 설정됩니다. 필터에 의한 최종적인 스케일링은 없어집니다.

Motor[x].PosReportMode 모터의 위치 통지 모드

설명: 위치 통지 모드의 제어

범위: 0 ..1

단위: 논리값

기본값: 0

Motor[x].PosReportMode 는 모터의 위치를 통지하는 모드를 지정합니다. 기본값 0 으로 설정되는 경우에 최종적인 모터 명령 위치의 성분은 "오프셋 모드"이며 모터의 산출된 궤적의 기준 위치를 이동하여 축을 연결합니다. 위치가 모터의 조회 명령의 응답으로 통지될 때 그 성분을 뺍니다. 이 모드에서는 모터 위치는 프로그램 기준 위치에서의 상대값으로 통지됩니다.

Motor[x].PosReportMode 가 1 로 설정되는 경우에 위치가 모터의 조회 명령의 응답으로 통지될 때 모터 위치의 오프셋 모드 구성 성분은 빠지지 않습니다. 이 모드에서는 모터 위치는 모터 기준 위치("원점")에서의 상대값으로 통지됩니다.

Motor[x].MasterCtrl 의 비트 1 이 1 인 경우에 추종 기능은 오프셋 모드가 되고 **Motor[x].ActiveMasterPos** 의 위치 추종 기능의 마스터 위치 성분은 오프셋 모드 성분이 됩니다. 이 경우에 **Motor[x].PosReportMode** 는 그 값이 통지되는 위치에서 뺄지 여부를 결정합니다.

Motor[x].MasterCtrl 의 비트 1 이 0 인 경우에 추종 기능은 일반 모드가 되고 **Motor[x].ActiveMasterPos** 는 오프셋 모드 성분이 아닙니다. 이 경우에 **Motor[x].PosReportMode** 의 값과 관계없이 통지된 위치에서 뺄지 않습니다.

보통 보정 테이블 또는 캠 테이블에서 발생하는 **Motor[x].CompDesPos** 의 모터 위치 성분은 항상 오프셋 모드 성분이 됩니다. 따라서 **Motor[x].PosReportMode** 는 그 값이 통지된 위치에서 뺄지 여부를 결정합니다.

Motor[x].PosReportMode 는 모터 명령 위치(d, dread), 현재 위치(p, pread)와 대상 위치(t, tread)의 통지에 오프셋 모드 위치 성분이 어떻게 사용되는지를 제어합니다.

Motor[x].PosSf 모터의 아우터 루프 스케일 계수

설명: 아우터(위치) 루프의 스케일 계수

범위: 양의 부동 소수점(배정도)

단위: 모터 단위/소스 단위

기본값: 1.0

기존 I 변수명: lxx08

Motor[x].PosSf에서는 **Motor[x].pEnc**로 지정된 레지스터에서 읽은 실제 위치값에 곱할 스케일 계수를 지정합니다. 이것은 모터의 아우터(보통은 위치) 루프의 현재 위치 연산 중에서 사용되기 전에 곱합니다. 이 스케일 계수는 소프트웨어 용도에서의 모터 단위를 효과적으로 정의합니다. 기본값에서는 1.0의 스케일 계수에서 소스 데이터 단위와 일치하지만 이 모터 단위는 소스 데이터 단위와 일치할 필요는 없습니다.

위치 데이터의 소스는 대부분의 경우 엔코더 변환 테이블(ECT) 항목의 결과 레지스터가 됩니다. 각 항목에도 그 자체의 출력 스케일 계수가 있으며 따라서 처리의 각 단계에서 사용되는 단위에 대해 유연성이 많습니다. 그러나 몇 가지 보간 기법에서 소수점의 분해능이 가능해도 ECT 항목의 부동 소수점 출력은 카운터의 "카운트" 또는 데이터 워드나 A/D 컨버터의 LSB로 스케일링되는 것이 가장 일반적 방법입니다.

보통 **Motor[x].PosSf**는 기본값 1.0 그대로입니다. 따라서 모터 단위는 피드백 장치의 "카운트" 또는 "LSB"가 됩니다. 또는 모터 단위가 mm, 인치, 도(度) 또는 회전 등의 공학 단위가 되도록 설정합니다. 처음인 경우에 적합한 축의 단위는 단일성이 없는 스케일 계수에 의해 공학 단위로 변환됩니다(예를 들어 #1->1000X). 2번째인 경우에 축 정의에서 단일성 스케일 계수를 사용할 수 있습니다(예를 들어 #1->X).

위치 또는 그 미분(속도, 가속도, 저크)을 처리하는 모터의 모든 저장 설정 요소는 **Motor[x].PosSf**로 정의되는 모터 단위를 사용합니다. 따라서 이 요소의 변경은 그러한 요소의 모든 변화를 약하게 해야 합니다.



이 스케일 계수는 실질적으로는 피드백 알고리즘의 게인항입니다. 따라서 이 스케일 계수의 변경은 특히 **Motor[x].Servo.Kp**의 비례 게인항의 서보 게인의 변화를 약하게 해야 합니다.

Motor[x].Pos2Sf 모터의 이너 루프 스케일 계수

설명: 이너(속도) 루프의 스케일 계수

범위: 양의 부동 소수점(배정도)

단위: 모터 단위/소스 데이터 LSB

기본값: 1.0

기준 I 변수명: lxx09

Motor[x].Pos2Sf 는 **Motor[x].pEnc2** 로 지정된 레지스터에서 읽은 실제 위치값에 곱할 스케일 계수를 지정합니다. 이것은 모터의 이너(보통은 속도) 루프의 현재 위치 연산 중에서 사용되기 전에 곱합니다. 이 스케일 계수는 소프트웨어 용도에서의 모터 단위를 효과적으로 정의합니다. 기본값에서는 1.0 의 스케일 계수에서 소스 데이터 단위와 일치하지만 이 모터 단위는 소스 데이터 단위와 일치할 필요는 없습니다.

위치 데이터의 소스는 대부분의 경우 엔코더 변환 테이블(ECT) 항목의 결과 레지스터가 됩니다. 각 항목에도 그 자체의 출력 스케일 계수가 있으며 따라서 처리의 각 단계에서 사용되는 단위에 대해 유연성이 많습니다. 그러나 몇 가지 보간 기법에서 소수점의 분해능이 가능해도 ECT 항목의 부동 소수점 출력은 카운터의 "카운트" 또는 데이터 워드나 A/D 컨버터의 LSB 로 스케일링되는 것이 가장 일반적 방법입니다.

보통 **Motor[x].Pos2Sf** 는 기본값 1.0 그대로입니다. 따라서 이너 루프 단위는 피드백 장치의 "카운트" 또는 "LSB"가 됩니다. 또는 이너 루프 단위가 mm, 인치, 도 또는 회전 등의 공학 단위가 되도록 설정합니다. 대부분의 경우 단위 선택은 **Motor[x].PosSf** 로 설정되는 아우터 루프 단위와 일치합니다.



이 스케일 계수는 실질적으로는 피드백 알고리즘의 게인항입니다. 따라서 이 스케일 계수의 변경은 특히 **Motor[x].Servo.Kvfb** 및 **Kvifb** 의 비례 게인항의 서보 게인의 변화를 약하게 해야 합니다.

Motor[x].PowerOnMode 모터의 전원 셋업 제어

설명: 전원 셋업/리셋 시의 동작 제어

범위: 0 ..7

단위: 비트 필드

기본값: 0

기준 I 변수명: lxx80

Motor[x].PowerOnMode 는 모터의 전원 셋업 시의 동작을 결정합니다. 이는 3 비트의 독립된 제어 비트로 구성됩니다.

- 비트 0(값 1)은 \$의 온라인 명령 또는 **Motor[x].PhaseFindingStep** 을 1로 설정 이 어느 하나의 페이즈 참조 명령 후의 모터 상태를 결정합니다. 비트가 0 인 경우에 예를 들어 페이즈 서치 동작을 위해 모터가 **Enable** 되었다고 해도 모터는 **kill** 됩니다. 비트가 1 인 경우에 모터는 **Enable** 되고 위치 루프가 실행됩니다. 이것은 절대 위치 읽기와 페이즈 서치 동작 양쪽의 페이즈 참조 동작 후에 적용합니다. Power PMAC에서 전류(commutation)되지 않은 모터에서는 비트가 0 인 경우에 페이즈 참조 명령은 모터를 **Enable** 로 하지 않으며 루프를 실행하지 않습니다. 비트가 1 인 경우에 페이즈 참조 명령은 모터를 **Enable** 로 하고 루프를 실행합니다.
- 비트 1(값 2)은 페이즈 참조 명령이 전원 셋업/리셋 시에 모터에 자동으로 전송되는지 여부를 결정합니다. 비트가 0 인 경우에 명령은 Power PMAC 에서 자동으로 전송되지 않습니다. 그 경우에 사용자 어플리케이션에서 명령을 전송해야 합니다. 비트가 1 인 경우에 명령은 전원 셋업/리셋 시에 자동으로 전송됩니다. 따라서 사용자 어플리케이션 소프트웨어에서의 명시적인 조작은 필요 없습니다. Power PMAC 에서 전류(commutation)되지 않은 모터에서는 이 비트와 비트 0(양쪽 모두)이 1 인 경우 모터는 전원 투입/리셋 시에 자동으로 **Enable** 됩니다.

루프가 실행될 때 페이즈 참조의 실패는 위험한 폭주 상태로 이어질 가능성이 있다는 점에 주의하십시오. 페이즈 참조(읽기 또는 서치)가 전원 셋업/리셋 시에 즉시 실행되는 경우에 사용되는 모든 센서 및(또는) 앰프가 그때에는 반드시 정상적으로 운전 상태여야 합니다.

- 비트 2(값 4)는 절대값 서보 위치 읽기 명령이 전원 셋업/리셋 시에 모터에 자동으로 전송되는지 여부를 결정합니다. 비트가 0 인 경우에 이 명령은 Power PMAC 에서 자동으로 전송되지 않습니다. 이 경우에 사용자 어플리케이션에서 명령(온라인 **hmz** 또는 프로그램 **homez**)을 전송해야 합니다. 비트가 1 인 경우에 이 명령은 전원 셋업/리셋 시에 자동으로 전송됩니다. 따라서 사용자 어플리케이션 소프트웨어에서의 명시적인 조작은 필요 없습니다. **Motor[x].pAbsPos** 가 기본값 0 으로 설정되는 경우에 절대값 위치 읽기 기능을 무효로 하므로 이 비트의 설정은 관계없어집니다.

Motor[x].pPhaseEnc 모터의 페이즈 엔코더 포인터

설명: 이동 중인 전류(commutation) 위치의 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: 하드웨어에 따라서 자동으로 구성됩니다.

기준 I 변수명: lxx83

Motor[x].pPhaseEnc에서는 모터의 전류(commutation) 위치 피드백에 사용되는 32 비트 레지스터를 지정합니다. 여기에는 이 레지스터의 주소가 저장됩니다. 이와 같이 로터 각도(위치) 피드백을 수행하기 위해 어떤 피드백 장치가 사용되는지를 결정합니다. 이 요소의 이름에도 불구하고 엔코더가 이 피드백에 사용될 필요가 없다는 점에 주의하십시오.

Motor[x].PhaseCtrl의 비트 3(값 8)이 1로 설정되고 모터의 서보 루프가 페이즈 인터럽트로 실행되는 경우에 이 요소는 아우터(위치) 서보 루프와 이너(속도) 서보 루프에 사용되는 레지스터를 지정합니다. 그러나 **Motor[x].pPhaseLoadEnc(V1.6** 펌웨어에서의 새로운 기능)가 0 이외의 값으로 설정되는 경우에 이는 아우터(위치) 루프 피드백에 사용되는 레지스터를 지정합니다. 그리고 **Motor[x].pPhaseEnc**는 이너(속도) 루프 피드백에 사용되는 레지스터만 지정합니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

이 레지스터의 데이터는 고정 소수점(정수) 형식이 필요하며 유익하게 하기 위해 페이즈 사이클마다 갱신되어야 합니다. 엔코더 변환 테이블에서의 데이터는 이 기능에는 적절치 않습니다.

여기서 지정된 레지스터에서의 소스 데이터가 "롤 오버"할 수 있도록 이 레지스터를 사용한 최종적인 데이터는 32 비트 버스의 최상위 비트(MSB)에서 배치되어야 한다는 점에 주의하십시오. 이것이 소스 레지스터 내의 데이터에 해당하지 않는 경우에 **Motor[x].PhaseEncRightShift**와 **Motor[x].PhaseEncLeftShift**를 사용할 수 있습니다.

가장 일반적으로 **Motor[x].pPhaseEnc**는 서보 IC 채널의 인크리멘탈 엔코더 "페이즈 포착" 레지스터 또는 서보 IC 나 FPGA의 시리얼 엔코더 데이터 레지스터(페이즈 클럭으로 래치된) 또는 MACRO IC 노드의 위치 피드백 레지스터(레지스터[0])의 주소로 설정됩니다. 예를 들어

```
Motor[1].pPhaseEnc = Gate1[4].Chan[0].PhaseCapt.a
Motor[2].pPhaseEnc = Gate3[1].Chan[1].PhaseCapt.a
Motor[3].pPhaseEnc = Gate3[2].Chan[2].SerialEncDataA.a
Motor[4].pPhaseEnc = Acc84E[0].Chan[3].SerialEncDataA.a
Motor[5].pPhaseEnc = Gate2[0].Macro[4][0].a
Motor[6].pPhaseEnc = Gate3[1].MacroInA[5][0].a
```

Motor[x].pPhaseLoadEnc 모터의 페이즈 부하 위치 포인터

설명:	페이즈 인터럽트 서보에서 독립된 부하 위치의 포인터
범위:	적정한 주소
단위:	데이터 구조체 요소 주소
기본값:	0(무효)

Motor[x].pPhaseLoadEnc 는 페이즈 인터럽트에서 서보 루프를 실행할 때의 다른 아우터 루프의 서보 피드백 위치로 사용되는 32 비트 레지스터의 주소를 지정합니다. **Motor[x].PhaseCtrl** 의 비트 3(값 8)이 1 로 설정되는 경우에 페이즈 인터럽트로 서보 루프가 실행됩니다. 여기에는 이 레지스터의 주소가 저장됩니다. 이것이 0 으로 설정되는 경우에 **Motor[x].pPhaseEnc** 로 지정된 레지스터는 이너 루프와 아우터 루프의 서보 피드백 위치 양쪽에 사용됩니다. **Motor[x].pPhaseLoadEnc** 가 도입된 V1.6 보다 이전 펌웨어 버전인 경우는 항상 이 사례에 해당합니다. 이것은 이전 버전과의 호환성을 제공합니다.

고속 톨 서보 축과 같이 서보 루프가 위상 인터럽트로 실행되고 있을 때조차 **Motor[x].pPhaseLoadEnc** 는 듀얼 피드백 기능을 제공합니다. 이 경우에 **Motor[x].pPhaseLoadEnc** 는 보통 부하측의 피드백에 사용되고 정밀도를 위한 위치 루프를 실행합니다 그리고 **Motor[x].pPhaseEnc** 는 보통 모터측의 피드백에 사용되고 안정성을 위한 속도 루프를 실행합니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

이 레지스터의 데이터는 고정 소수점(정수) 형식이 필요하며 유익하게 하기 위해 페이즈 사이클마다 갱신되어야 합니다. 엔코더 변환 테이블에서의 데이터는 이 기능에는 적절치 않습니다.

여기서 지정된 레지스터에서의 소스 데이터가 "롤 오버"할 수 있도록 이 레지스터를 사용한 최종적인 데이터는 32 비트 버스의 최상위 비트(MSB)에서 배치되어야 한다는 점에 주의하십시오. 이것이 소스 레지스터 내의 데이터에 해당하지 않는 경우에 **Motor[x].PhaseEncRightShift** 와 **Motor[x].PhaseEncLeftShift** 를 사용할 수 있습니다.

가장 일반적으로는 **Motor[x].pPhaseLoadEnc** 는 서보 IC 채널의 인크리멘탈 엔코더 "페이즈 포착" 레지스터 또는 서보 IC 나 FPGA 의 시리얼 엔코더 데이터 레지스터(페이즈 클록으로 래치된) 또는 MACRO IC 노드의 위치 피드백 레지스터(레지스터[0])의 주소로 설정됩니다. 예를 들어

```
Motor[1].pPhaseLoadEnc = Gate1[4].Chan[1].PhaseCapt.a
Motor[2].pPhaseLoadEnc = Gate3[1].Chan[3].PhaseCapt.a
Motor[3].pPhaseLoadEnc = Gate3[2].Chan[0].SerialEncDataA.a
Motor[4].pPhaseLoadEnc = Acc84E[0].Chan[1].SerialEncDataA.a
Motor[5].pPhaseLoadEnc = Gate2[0].Macro[8][0].a
Motor[6].pPhaseLoadEnc = Gate3[0].MacroInA[1][0].a
```

Motor[x].PreFilterEna 모터의 사전 필터 유효

설명:	궤적 사전 필터의 유효/갱신 주기 제어
범위:	0 ..255
단위:	없음
기본값:	0

Motor[x].PreFilterEna에서는 궤적 사전 필터가 유효가 되는지 여부를 제어하고 유효한 경우 필터의 갱신 주기를 설정합니다. 기본값 0 인 경우에 사전 필터는 무효가 됩니다. 0 보다 큰 경우에 사전 필터는 유효가 되고 그 값은 서보 인터럽트 주기에서의 필터 갱신 주기를 지정합니다. 대개 필터가 저주파의 동특성을 보상하기 위해 사용되는 경우에 많은 서보 사이클 주기가 필요해집니다. 필터 주기가 1 서보 사이클보다 큰 경우에 필터 출력은 3 차 스플라인 알고리즘을 사용하여 서보 갱신 주기에 재보간됩니다.

궤적 사전 필터에서는 다항식항 **Motor[x].Pn0**에서 **Pn4**와 **Motor[x].Pd1**에서 **Pd4**를 사용하여 수학적으로 계산된 궤적에서의 명령 위치의 값을 조정합니다. 그 후에 그 값들은 서보 루프로 전송됩니다. 이 사전 필터는 시스템의 기계적인 공진 주파수에서의 주파수 성분 제거에 사용할 수 있습니다.

모터의 서보 루프가 실행되지 않게 되면(Enable 또는 Disable 상태 어느 쪽이랴도) Power PMAC 은 자동으로 **Motor[x].PreFilterEna**을 루프가 실행되고 있었던 때의 설정값의 음수로 설정합니다. 루프가 다시 실행될 때 Power PMAC 은 확실하게 필터의 적절한 초기화를 수행합니다. Enable 시 Power PMAC 은 자동으로 **Motor[x].PreFilterEna**을 양의 설정값으로 반환합니다.

Motor[x].ProgJogPos 모터의 가변 JOG 위치

설명:	가변 JOG 위치/거리
범위:	부동 소수점
단위:	모터 단위
기본값:	0.0

Motor[x].ProgJogPos는 모터의 온라인 "가변 JOG" 명령의 종료 위치 또는 동작 거리를 지정합니다. j=* 명령이 내려지면 **Motor[x].ProgJogPos**의 값은 그 JOG 동작의 종료 위치로 사용됩니다. j:* 명령이 내려지면 **Motor[x].ProgJogPos**의 값은 그 JOG 동작에서 현재 명령 위치에서의 거리로 사용됩니다. j^* 명령이 내려지면 **Motor[x].ProgJogPos**의 값은 그 JOG 동작에서 현재 위치에서의 거리로 사용됩니다.

Motor[x].pSineTable 모터의 사인파 테이블 포인터

설명: 전류(commutation)의 전류 참조 테이블로의 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: **Sys.SineTable[0].a**

Motor[x].pSineTable 은 Power PMAC 디지털 전류 루프 알고리즘에서의 전류 피드백값에 대한 2048-항목의 참조 "사인파" 테이블의 시작 주소를 지정합니다. 이 테이블은 클로즈 전류 루프에서의 실제 상전류 피드백 입력에서 "다이렉트" 및 "직교" 성분으로 변환하기 위해 사용됩니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

대다수의 경우에 **Motor[x].pSineTable**에서는 Power PMAC에서 제공되는 표준 사인파 테이블의 주소를 지정합니다. 다른 테이블이 요구되는 경우에 사용자 공유 메모리 버퍼의 연속한 2048 레지스터에 로드되어야 하며(360 도 대응하는 세트이며 각 레지스터는 4 바이트의 단정도 부동 소수점 값으로 구성) **Motor[x].pSineTable**은 이 테이블의 시작 주소로 설정됩니다.

예를 들어 사용자 정의 테이블이 **Sys.Fdata[4096]**(0 도에서의 항목)에서 **Sys.Fdata[6143]**(359.824 도에서의 항목)까지를 사용하여 사용자 공유 메모리 버퍼에 로드되는 경우에 **Motor[x].pSineTable**은 **Sys.pushm+16384**로 설정됩니다(각 **Fdata** 요소는 4 바이트를 점유하므로 $4096 * 4 = 16,384$ 가 됩니다).

Motor[x].pVoltSineTable 모터의 전압 사인파 테이블 포인터

설명: 전류(commutation)의 전압 참조 테이블로의 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: **Sys.SineTable[0].a**

Motor[x].pVoltSineTable은 Power PMAC 전류(commutation) 알고리즘의 상 출력값을 위한 2048 항목의 참조 테이블 시작 주소를 지정합니다. 이것은 Power PMAC의 모터 전류 루프가 실행되는지 여부와 관계없이 사용됩니다. 이 테이블은 계산된 "다이렉트" 및 "직교" 출력 명령에서 개별 상 명령으로 변환하기 위해 사용됩니다. Power PMAC이 전류 루프를 실행하는 경우에 실제 전류 피드백 입력에서 "다이렉트" 및 "직교" 성분으로 변환하기 위해 사용되는 테이블은 다른 요소 **Motor[x].pSineTable**로 지정된다는 점에 주의하십시오.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자가 이 주소의 수치를 알 필요는 없습니다.

대다수의 경우에 **Motor[x].pVoltSineTable**에서는 Power PMAC에서 제공되는 표준 사인파 테이블의 주소를 지정합니다. 다른 테이블이 요구되는 경우에 사용자 버퍼의 연속한 2048 레지스터에 로드되어야 하며(360 도 대응하는 세트이며 각 레지스터는 4 바이트의 단정도 부동 소수점 값으로 구성) **Motor[x].pVoltSineTable**은 이 테이블의 시작 주소로 설정됩니다.

예를 들어 사용자 정의 테이블이 **Sys.Fdata[6144]**(0 도에서의 항목)에서 **Sys.Fdata[8191]**(359.824 도에서의 항목)까지를 사용하여 사용자 공유 메모리 버퍼에 로드되는 경우에 **Motor[x].pVoltSineTable**은 **Sys.pushm+24576**으로 설정됩니다(각 **Fdata** 요소는 4 바이트를 점유하므로 $6144 * 4 = 24,576$ 이 됩니다).

Motor[x].PwmDbComp 모터의 PWM 불감대 보정

설명: PWM 불감대(Dead band) 보정 전압 오프셋의 크기

범위: -32,768..32,767

단위: PWM 출력 회로 단위

기본값: 0

Motor[x].PwmDbComp는 다이렉트 PWM 모드에서의 각 상 PWM 전압 명령 출력값에 적용되는 오프셋 전압의 크기를 지정합니다. 이것은 파워 트랜지스터의 턴 온/턴 오프 시간과 그 결과 발생하는 필요한 데드 타임에 의해 생성된 불감대 효과를 보정하기 위해 사용됩니다. 기본값 0인 경우에 해당 불감대 보정 연산은 실행되지 않습니다.

상의 명령 전류의 크기가 저장 설정 요소 **Motor[x].PwmDbI**의 값보다 큰 경우 보정이 상 전압에 적용됩니다. 상전류가 적어도 이 크기에서 양인 경우에 **Motor[x].PwmDbComp**의 2배 값이 **Motor[x].IaVolts**, **IbVolts** 또는 **IcVolts**의 해당 상 전압 명령의 값에 더해지고 **Motor[x].PwmDbComp**의 값을 다른 두 상 전압 명령에서 빼줍니다. 상전류가 적어도 이 크기에서 음인 경우 **Motor[x].PwmDbComp**의 2배 값을 해당 상 전압 명령의 값에서 빼지고 **Motor[x].PwmDbComp**의 값을 다른 2개의 상 전압 명령에 더해줍니다. 이 계산은 모터의 측정된 각 상전류에 따라서 3개 상 전체에서 수행됩니다.

Motor[x].PwmDbComp는 사용되는 서보 인터페이스의 PWM 출력 회로 단위로 표시됩니다. PMAC2 스타일 DSPGATE1 ASIC가 PWM 출력 생성에 사용되는 경우에 해당 단위는 저장 설정 요소 **Gate1[i].PwmPeriod**의 값으로 결정됩니다. 즉 +100%의 전압 명령(상 전압 명령은 $+V_{DCbus}$ 와 같음)은 **+Gate1[i].PwmPeriod**와 같은 명령값에 의해 생성되고 -100%의 전압 명령(상 전압 명령은 $-V_{DCbus}$ 와 같음)은 **Gate1[i].PwmPeriod**와 같은 명령값에 의해 생성됩니다.

PMAC3 스타일 DSPGATE3 ASIC가 PWM 출력 생성에 사용되는 경우에 단위는 PWM 주기/주파수와 관계없이 +/-16,384 범위로 고정됩니다. 즉 +100%의 전압 명령(상 전압 명령은 $+V_{DCbus}$ 와 같음)은 +16,383와 같은 명령값에 의해 생성되고 -100%의 전압 명령(상 전압 명령은 $-V_{DCbus}$ 와 같음)은 -16,384와 같은 명령값에 의해 생성됩니다.

Motor[x].PwmDbI 모터의 PWM 불감대 전류 최소치

설명:	PWM 불감대(Dead band) 보정 전류 최소치
범위:	-32,768..32,767
단위:	16 비트의 전류 LSB
기본값:	0

Motor[x].PwmDbI 는 **Motor[x].PwmDbComp** 의 오프셋 전압을 위해 사용되는 상 명령 전류의 최소 크기를 지정합니다. 다이렉트 PWM 모드에서의 제로 크로싱 점에서의 불감대 효과의 보정에 **Motor[x].PwmDbComp** 의 오프셋 전압이 적용됩니다. 페이즈 사이클마다 모터의 각 상의 명령 전류값은 **Motor[x].PwmDbI** 와 비교됩니다.

상의 명령 전류의 크기가 **Motor[x].PwmDb** 보다 큰 경우에 제로 크로싱 점의 불감대에서의 물리적인 전압 및 전류 파형의 왜곡을 최소한으로 하기 위해 **Motor[x].PwmDbComp** 의 값에 따른 오프셋이 모터 상 전압 출력값에 적용됩니다. 적용되는 오프셋의 방향은 전류값의 부호에 좌우됩니다. 그러나 상의 명령 전류 크기가 **Motor[x].PwmDbI** 이하인 경우에 오프셋은 이 상전류의 결과로서의 상 전압 출력값에 적용되지 않습니다.

Motor[x].PwmSf 모터의 PWM 출력 스케일 계수

설명:	PWM 출력의 스케일 계수
범위:	0 ..32,767
단위:	PWM 카운터 단위
기본값:	7636
기존 I 변수명:	lxx66

Motor[x].PwmSf 는 전류(commutation) 또는 디지털 전류 루프 알고리즘의 출력(-1.0~+1.0 의 범위로 정규화된 값으로 여기서 해석됩니다)에 곱하고 그 결과가 출력 레지스터에 쓰여집니다. 이와 같이 출력의 스케일링과 출력의 최대값의 양쪽을 결정합니다.

모터에서 디지털 전류 루프가 실행되지 않은 경우(**Motor[x].pAdc = 0**)에 보통 32,767 의 최대 허용값이 필요해지며 출력 D/A 컨버터의 전체 범위의 사용이 가능해집니다(실제 분해능이 달라도 부호가 있는 16 비트 분해능으로 취급). 더 작은 값도 사용할 수 있지만 그것이 시스템의 실효 계인을 감소시킨다는 점에 주의해야 합니다.

모터에서 디지털 전류 루프가 실행되지 않은 경우(**Motor[x].pAdc>0**)에 **Motor[x].PwmSf** 는 일반 PWM 생성 회로의 수치 범위와 관련되어 설정됩니다. PMAC2 스타일 "DSPGATE1" 서보 IC에서는 이 범위는 설정 요소 **Gate1[j].PwmPeriod** 에 의해 제어됩니다. PMAC3 스타일 "DSPGATE3" IC에서는 PWM 생성 회로의 수치 범위는 항상 +/-16,384 가 됩니다. 그때 **Motor[x].PwmSf** 는 이 값에 상대적으로 설정되어야 합니다.

전체 범위의 크기(**Gate1[j].PwmPeriod** 또는 16,384)와 같은 **Motor[x].PwmSf**의 설정은 그러한 파형의 피크에서 출력이 정확하게 0%와 100%의 듀티 사이클이 되도록 합니다(50%의 듀티 사이클은 최종적인 제로 명령이 됩니다).

다이렉트 PWM 앰프에서 그 파형이 결코 0% 또는 100%의 듀티 사이클에 도달하지 않아야 하는 경우(보통 충전 펌프의 유효 상태를 유지하기 위해)에 **Motor[x].PwmSf**는 보통 전체 범위의 크기보다 작은 95% 미만으로 설정되어야 합니다. 자세한 내용은 개별 앰프 설명서를 참조하십시오.

그렇지 않으면 **Motor[x].PwmSf**는 전체 범위의 크기 이상으로 설정할 수 있습니다. 전체 범위보다 큰 값인 경우에 입력 명령이 크면 전류(commutation) 사이클 부분을 초과하여 파형을 포화 상태에서 구동합니다. 그러나 이 경우 유효한 부가 토크를 생성할 수 있습니다. 기본적으로 큰 명령에서 파형은 사인파에서 사다리꼴로 변환됩니다. **Motor[x].PwmSf**는 **Gate1[j].PwmPeriod**보다 17%까지 큰 값이 유효가 됩니다. **Gate1[j].PwmPeriod**의 기본값 6527에서 **Motor[x].PwmSf**의 값 7636은 이 여유를 포함합니다.

PMAC3 스타일 "DSPGATE3" IC에서는 PWM 생성 회로의 수치 범위는 항상 +/- 16,384가 됩니다. 따라서 **Motor[x].PwmSf**는 이 값에 상대적으로 설정되어야 합니다.

Motor[x].RapidSpeedSel 모터의 Rapid 모드 속도 선택

설명: rapid 모드의 속도 선택

범위: 0 ..1

단위: 없음

기본값: 1

기존 I 변수명: lxx90

Motor[x].RapidSpeedSel은 모터의 RAPID 모드 동작의 속도를 결정하기 위해 어떤 설정 데이터 구조체 요소가 사용되는지 지정합니다. 이것이 0인 경우 **Motor[x].JogSpeed**가 사용됩니다. 이것이 기본값 1인 경우 **Motor[x].MaxSpeed**가 사용됩니다.

다축 rapid 동작을 수행하는 모터가 소속된 좌표계의 **Coord[x].RapidVelCtrl**이 1로 설정되고 이 모터에서는 다른 모터보다 작은 거리 대 속도비가 되는 경우에 그 모터의 동작 시간이 최대 거리 대 속도비의 모터 동작 시간과 일치하도록 그 모터의 명령 속도는 감소된다는 점에 주의하십시오.

Motor[x].RapidSpeedSel은 풀 워드 요소 **Motor[x].Control[0]**의 비트 9의 부분적인 구성 요소입니다.

Motor[x].ServoCaptTimeOffset 모터의 서보 포착 시간 오프셋

설명: 서보 피드백의 샘플링에서 인터럽트까지의 오프셋

범위: 32 비트의 정수

단위: 서보 사이클의 1/65536

기본값: 0

Motor[x].ServoCaptTimeOffset에서는(아우터 루프의) 위치 피드백 데이터가 샘플링될(래치 또는 스트로브될) 때부터 서보 인터럽트(서보 클록의 하강 엣지)까지의 시간을 지정합니다. 이 값은 타이머 보조 소프트웨어 포착 연산에서 사용됩니다(**Motor[x].CaptureMode**를 3으로 설정하는 경우). 다른 목적으로 설정할 필요는 없습니다.

피드백의 많은 형식 특히 시리얼 엔코더나 시리얼 ADC를 사용한 피드백에서 Power PMAC의 시리얼 데이터 인터페이스는 서보 인터럽트보다 상당히 앞서 샘플링되어 Power PMAC에 데이터를 전송할 시간을 제공해야 합니다. 그렇게 하면 서보 인터럽트 태스크에서 사용할 준비가 된 것입니다. 샘플링과 인터럽트 간의 지연이 고려되지 않은 경우에 이 모드에서의 포착 트리거로 래치된 타이머값을 사용하는 계산은 정확하지 않게 됩니다.

대부분의 Power PMAC의 시리얼 ADC는 페이즈 클록의 상승 엣지에서 아날로그 신호를 스트로브하고 Power PMAC에 데이터를 전송합니다. 그렇게 하면 페이즈 클록의 하강 엣지에서 해당 데이터는 준비가 완료됩니다. 페이즈 클록은 서보 클록의 n 배의 주파수가 되고(n 이 양의 정수) 서보 클록의 하강 엣지는(프로세서 인터럽트를 생성) 페이즈 클록의 하강 엣지와 항상 일치합니다.

ACC-28E 또는 ACC-59E3과 같이 페이즈 클록의 상승 엣지에서 샘플링되는 시리얼 A/D 피드백에서 서보 사이클당 4개의 페이즈 사이클이 되는 기본설정에서는(**aten[j].ServoClockDiv**의 기본값 3) 피드백 데이터는 서보 인터럽트의 1/8 서보 사이클 전에 샘플링되고 그때 **Motor[x].ServoCaptTimeOffset**은 $65,536/8 = 8,192$ 로 설정합니다.

ACC-84E UMAC 시리얼 엔코더 인터페이스 보드 또는 ACC-24E3 UMAC 축 인터페이스 보드와 Power Brick 제어 보드에서 사용되는 DSPGATE3 ASIC에 연결되는 시리얼 엔코더에서는 페이즈 클록 또는 서보 클록 중 하나의 상승 또는 하강 엣지에서 샘플링할 수 있습니다. 일반적으로 샘플링에서 인터럽트까지의 최소 지연을 제공하는 클록 엣지가 사용되지만 한편에서는 Power PMAC에 데이터를 전송하는 충분한 시간을 제공하여 인터럽트 연산에는 그 데이터의 준비가 완료됩니다.

엔코더가 페이즈 클록의 하강 엣지에서 샘플링되는 경우에 다음 서보 인터럽트에서 사용되기 전에 페이즈 클록 사이클의 1주기 분량의 지연이 발생합니다. 서보 사이클당 4개의 페이즈 사이클의 기본설정에서는 피드백은 서보 인터럽트의 1/4 서보 사이클 전에 샘플링됩니다. 그때 **Motor[x].ServoCaptTimeOffset**은 $65,536/4 = 16,384$ 로 설정합니다.

서보 클럭 신호는 페이지 클럭 신호에서 분주하여 구하며 그 하강 엣지는 페이지 클럭의 하강 엣지 중 하나와 일치합니다. 서보 클럭은 최대한 경우에도 1 페이지 클럭 사이클 동안 low 가 됩니다. 서보 사이클당 4 개의 페이지 사이클의 기본설정에서는 서보 클럭은 1 페이지 클럭 사이클 동안 low 가 되고 3 페이지 클럭 사이클 동안 high 가 됩니다. 엔코더가 서보 클럭의 상승 엣지에서 샘플링되는 경우에 서보 인터럽트의 3/4 서보 사이클 전에 샘플링됩니다. 그때 **Motor[x].ServoCaptTimeOffset** 은 $65,536 \times 3/4 = 49,152$ 로 설정합니다.

지정된 클럭 엣지에서의 시리얼 엔코더의 실제 샘플링을 지연하는 것도 가능하므로 주의하십시오. 이를 수행하는 경우에 **Motor[x].ServoCaptTimeOffset** 의 계산은 이 점을 고려해야 합니다.

ACC-14E I/O 보드를 사용하여 간섭계에서의 페러럴 데이터의 서보 피드백을 수행하는 경우에 보통 서보 클럭(인터럽트)의 하강 엣지에서 샘플링됩니다. 이 경우에 **Motor[x].ServoCaptTimeOffset** 은 0 으로 설정합니다.

Motor[x].ServoCtrl 모터의 서보 제어

설명: 서보 태스크를 유효로 하는 제어 플래그

범위: 0 ..15

단위: 비트 필드

기본값: 0

기존 I 변수명: lxx00

Motor[x].ServoCtrl 은 어느 태스크가 서보 루프에서 실행되는지 또한 어떻게 실행되는지 결정합니다. 이것은 4비트의 값이며 16의 운전 모드를 지정할 수 있는 가능성이 있지만 현재는 일부 모드만 탑재됩니다.

Motor[x].ServoCtrl 이 기본값 0 인 경우에 모터의 어떠한 계산도 실행되지 않고 위치 모니터링 기능조차 실행되지 않습니다. 따라서 위치 조회 명령에서는 위치 변화를 반영하지 않습니다. 어플리케이션에서 전혀 사용되지 않는 Power PMAC 의 모터는 모두 무효로 설정해야 합니다. 그래야 Power PMAC 은 그 모터의 계산 시간을 낭비하지 않게 됩니다.

Motor[x].ServoCtrl 이 1 인 경우에 모터는 표준 모드에서 유효가 됩니다. 이 모드에서는 위치 모니터링, 서보와 궤적의 연산이 이 모터에 대해 실행됩니다. 이 "유효" 모터에서는 명령 또는 이벤트에 의해 오픈 루프 또는 클로즈드 루프에서 "Enable" 또는 "Disable"(kill 상태)로 할 수 있습니다.

Motor[x].ServoCtrl 이 8 인 경우에 모터는 "갠트리 추종" 모드에서 유효가 됩니다. 이 모드에서는 위치 모니터링과 서보 계산이 이 모터에 대해 실행됩니다. 그러나 그 자체의 궤적 계산은 수행되지 않습니다. 대신 **Motor[x].CmdMotor** 로 지정된 모터의 계산된 궤적을 사용합니다. 이 기능에서는 축 프로그램 동작 또는 "갠트리 주축" 모터의 JOG 동작과 관계없이 여러 개 모터에서 동일한 명령 궤도를 사용할 수 있게 됩니다. "갠트리 주축" 모터는 표준 모드에서 유효로 해야 합니다(**Motor[x].ServoCtrl** = 1). "갠트리 종축"

모터는 "갠트리 주축" 모터에 대해 지연이 없으므로 주의하십시오(1 서보 사이클조차 없음).

유효로 Enable 된 모터를 **Motor[x].ServoCtrl** 을 0 으로 설정하여 무효로 하지 마십시오! 모터 출력은 직전의 명령 레벨에서 Enable 된 상태 그대로가 됩니다. 이 경우는 위험한 폭주 상태로 이어질 수 있습니다.

Motor[x].ServoCtrl 은 풀 워드 요소 **Motor[x].Control[0]**의 비트 28-31 의 부분적인 구성 요소입니다.

Motor[x].SlewMasterPosSf 모터의 위치 추종비 슬루 레이트

설명: 위치 추종 비율의 슬루 레이트
범위: 음이 아닌 부동 소수점
단위: (모터 단위/마스터 단위)/서보 사이클
기본값: 0.0(슬루 레이트 제어 무효)

Motor[x].SlewMasterPosSf 는 **Motor[x].MasterPosSf** 의 목표 비율 변화 시와 추종 기능의 무효/유효 전환 시 위치 추종 비율의 변화율을 지정합니다. **Motor[x].SlewMasterPosSf** 가 기본값 0.0 인 경우에 슬루 레이트 제어는 무효가 됩니다. 그리고 스텝 변화와 같이 추종 비율의 변화는 즉시 실행됩니다.

Motor[x].SlewMasterPosSf 가 0.0 보다 큰 값인 경우에 슬루 레이트 제어는 유효가 되고 서보 사이클에서 사용되는 실제의 위치 추종 비율은(상태 요소 **Motor[x].ActiveMasterPosSf**에서 확인 가능) 서보 사이클마다 **Motor[x].SlewMasterPosSf** 의 크기만 변화합니다. 이 변화는 **Motor[x].MasterPosSf** 의 목표값에 도달할 때까지 또는 추종 무효로 전환할 때의 0.0 에 도달할 때까지 일어납니다.

마스터 신호는 변화하지만("동적" 변화) 다시 추종의 유효 또는 무효로의 전환 또는 추종 비율을 변경하고자 하는 경우 **Motor[x].MasterPosSf** 를 적절한 양의 값으로 설정하는 것이 중요합니다. 변경 결과로 발생하는 모터 가속도 또는 감속도가 적절하게 제어되기 때문입니다.

Motor[x].SlipGain 모터의 슬립 게인

설명: 모터 전류(commutation) 명령/슬립 상수
범위: 음이 아닌 부동 소수점
단위: (전류(commutation) 각도 단위)/(16 비트 토크 명령 단위)
기본값: 0.0

Motor[x].SlipGain 은 서보 루프 출력 명령과 그 전류(commutation) 각도에서의 "슬립" 간의 비례 상수입니다. 슬립은 로터의 측정된 물리적인 각도에 비교해서 자계 각도가 진행되는 것입니다. **Motor[x].SlipGain** 은 다른 타입의 모터 제어에서 2 가지 다른 방법으로 사용할 수 있습니다.

비동기 유도 모터의 제어에는 **SlipGain** 은 페이즈 사이클마다 저장 설정 요소 **Motor[x].DtOverRotorTc** 와 **Motor[x].IdCmd** 의 값을 사용하고 중간값으로 자동 계산됩니다. 이 경우에 사용자는 보통 무시할 수 있는 내부 상태 요소로 사용합니다. 단 약한 계자와 같은 첨단 기술의 해석에는 필요할 것으로 보이며 **SlipGain** 저장값은 자동으로 덮어씁니다.

동기 AC 모터(보통 "스테핑" 모터라는 품목으로 판매됨)의 오픈 루프 "다이렉트 마이크로 스텝" 제어에서 **Motor[x].SlipGain** 은 사용자가 직접 설정합니다. 이 제어 모드에서 **Motor[x].DtOverRotorTc** 는 0.0 으로 설정되어야 하고 그에 따라 **SlipGain** 의 자동적인 내부 계산을 무효로 합니다. 다이렉트 마이크로 스텝 제어에서는 측정되는 전류(commutation) 각도 피드백은 없습니다. 따라서 이 파라미터에서 생성된 "슬립"만 모터를 구동하는 전류(commutation) 각도를 진행시킵니다. 이 경우에 (시뮬레이션된)서보 루프의 출력은 속도 명령입니다. **SlipGain** 을 출력에 곱하면 모터에 그 속도를 명령하는 데 필요한 만큼 전류(commutation) 각도는 진행됩니다.

다이렉트 마이크로 스텝에서의 **Motor[x].SlipGain** 의 설정은 속도 명령을 시뮬레이션된 서보 루프에서 전류(commutation) 각도의 적절한 진행으로 변환해야 합니다. 모터의 서보 단위가 개별 마이크로 스텝(2048/전류(commutation) 사이클)인 경우의 권장 설정에서는 **SlipGain** 은 1/(페이즈 사이클 수/서보 사이클)로 설정합니다. 4 페이즈 사이클/서보 사이클인 기본설정에서는 이 값은 1/4 또는 0.25 가 됩니다.

Motor[x].SoftLimitOffset 모터의 소프트웨어 리미트 오프셋

설명: 연산 시와 실행 시의 소프트웨어 리미트의 차

범위: 부동 소수점

단위: 모터 단위

기본값: 0.0

기존 I 변수명: lxx41

Motor[x].SoftLimitOffset 에는 동작 계산 시 사용되는 소프트웨어 위치 리미트와 동작 실행 시 사용되는 소프트웨어 위치 리미트 간의 부호가 있는 거리(차)를 지정합니다. 동작 계산 시에는 Power PMAC 은 음의 종점으로 **Motor[x].MinPos** 와 양의 종점으로 **Motor[x].MaxPos** 를 사용합니다. 동작의 종점이 이 제한을 초과하는 경우에 해당 동작은 실행되지 않거나 해당 종점이 리미트 위치로 변경됩니다. 이는 동작의 타입에 따라 결정됩니다. 세그먼트 동작(**Coord[x].SegMoveTime>0** 에서 동작하는 **linear**, **circle** 와 **pvt** 모드)에서는 각 세그먼트가 계산될 때마다 확인되고 초기 동작 계산 시에는 확인되지 않습니다.

동작 실행 시에는 Power PMAC 은 끊임없이 모터의 순간적인 최종적인 명령 위치(계산된 궤적과 마스터 추종의 양쪽에서(오픈 루프 모드에서는 실제 위치에서))를 음의 종단으로 (**Motor[x].MinPos - Motor[x].SoftLimitOffset**)에 대해 확인하고 양의 종단으로 (**Motor[x].MaxPos + Motor[x].SoftLimitOffset**)에 대해 확인합니다. 해당 최소치의 하나를 초과하는 경우(동일한 값이 아니라)에 해당 모터 Status 비트 **Motor[x].SoftMinusLimit** 또는 **Motor[x].SoftPlusLimit** 가 자동으로 설정됩니다.

Motor[x].SoftLimitOffset 가 0 또는 양인 경우에 동작 실행 시의 최소치는 동작 계산 시 최소치의 "외측"이 되고 동작 실행 시 해당 최소치의 하나를 초과하면 좌표계의 모든 모터는 Abort 되고 **Motor[x].AbortTa** 와 **Motor[x].AbortTs** 로 지정되는 제어로 정지시킬 수 있습니다.

Motor[x].SoftLimitOffset 이 음인 경우에 동작 실행 시의 최소치는 동작 계산 시의 최소치의 "내측"이 되고 모터 동작 실행 시의 최소치의 하나를 초과해도 좌표계의 모터는 Abort 되지 않습니다. 그러나 해당의 Status 비트는 설정된 그대로이며 해당 모터가 현재 "리미트 내"에 있음을 나타냅니다. 동작 계산 시 제한된 동작이 실제로 해당 제한에 도달할 때 Status 비트가 설정되는 것을 원하는 사용자는 **Motor[x].SoftLimitOffset** 을 매우 작은 음수로 설정하십시오(예를 들어 1 피드백 "카운트"와 같은 값).

Motor[x].MaxPos 가 **Motor[x].MinPos** 보다 크지 않은 경우에 동작 계산 시와 동작 실행 시 어느 경우에도 소프트웨어 위치 리미트 확인이 실행되지 않는다는 점에 주의하십시오.

Motor[x].SoftLimitOffset 과 동작 실행 시 확인의 대부분은 V1.5 펌웨어의 새로운 기능입니다(2012 년 3/4 분기에 출시).

Motor[x].Stime 모터의 서보 사이클 연장

설명: 서보 사이클 주기의 연장

범위: 0 ..2³¹-1

단위: 서보 인터럽트 주기

기본값: 0

기존 I 변수명: lxx60

Motor[x].Stime 에서는 모터의 연속하는 위치/속도 루프의 실행 사이에 건너뛰고 읽은 서보 인터럽트의 수를 지정합니다. 이는 1 사이클의 서보 인터럽트 주기를 초과하여 모터의 서보 갱신 시간의 연장을 가능하게 합니다. 서보 루프는 (**Motor[x].Stime + 1**)서보 인터럽트마다 실행됩니다. 기본값 0에서 루프는 서보 인터럽트마다 실행됩니다. 이 서보 갱신 시간의 연장은 느린 동특성 및/또는 제한된 피드백 분해능을 가지는 모터에 도움이 됩니다. 또한 실제 모터 대신 제어 루프가 느린 프로세스 제어 기능에 사용되는 경우에도 도움이 됩니다.

Motor[x].PhaseCtrl 의 비트 3(값 8)을 1 로 설정하여 서보 루프가 페이즈 인터럽트로 실행되는 경우에 **Motor[x].Stime** 에서는 모터의 연속하는 위치/속도 루프의 실행 사이에서 건너뛰고 읽은 페이즈 인터럽트의 수를 지정합니다.



Motor[x].Stime 의 값의 변경은 서보 루프의 동특성을 변경하는 것이 되어 예측할 수 없는 결과로 이어질 수 있습니다. 서보 루프는 변경한 후 (재)조정해야 합니다.

궤적(보간) 갱신과 전류(commutation) 갱신을 포함한 다른 갱신 시간은 **Motor[x].Stime** 에서 영향을 받지 않습니다 **Motor[x].Stime** 에 의해 글로벌 설정 요소 **Sys.ServoPeriod** 를 변경할 필요는 없습니다.

Motor[x].TraceSize 모터의 트레이스 크기

설명: 모터 동작의 트레이스 버퍼 길이

범위: 음이 아닌 정수

단위: 동작 세그먼트 또는 섹션

기본값: 0

Motor[x].TraceSize 는 기본값 0 보다 큰 경우에 모터 동작의 트레이스 버퍼 길이를 지정합니다. 이 버퍼에는 이미 실행된 동작의 모션식이 저장됩니다. 동작이 세그먼트화되지 않은 경우에 다른 모션식을 가지는 프로그램 동작의 각 섹션(예를 들어 일정 가속도, 일정 속도의 섹션)은 버퍼 중에 1 슬롯을 점유합니다. 동작이 세그먼트화된 경우에 **Coord[x].SegMoveTime** 의 각 세그먼트는 버퍼 중에 1 슬롯을 점유합니다.

트레이스 버퍼의 존재는 모터가 할당된 좌표계의 음의 "타임 베이스"에 의해 이미 실행된 동작의 역방향 실행을 가능하게 합니다. 역행의 실행은 저장된 트레이스 버퍼의 처음으로의 연속 동작 시퀀스를 통해 속행할 수 있습니다. 연속 동작 시퀀스는 제로 속도 동작과 **delay** 시간을 포함할 수 있지만 **dwell** 시간과 논리적인 이유에 의한 시퀀스의 브레이크(예를 들어 프로그램 종료)를 포함할 수 없습니다.

모터 트레이스 버퍼에 의한 역행 궤적은 룩 어헤드 버퍼 이력에 의한 역행 궤적과 유사하지만 이는 "상태 변화"의 후방/전방 제어 대신 "비례의" 후방/전방 제어에 보다 적합합니다.

좌표계에서 위치 결정 축에 할당되는 모든 모터는 좌표계에서 음의 타임 베이스를 사용하여 프로그램 궤적의 완전한 역행을 가능하게 하는 동일 크기의 트레이스 버퍼가 있어야 합니다.

Motor[x].TraceSize 는 V1.6 펌웨어의 새로운 기능입니다(2014 년 1/4 분기 출시).

Motor[x].WarnFeLimit 모터의 편차 경고 리미트

설명: 경고(트리거) 위치 편차의 리미트

범위: 음이 아닌 부동 소수점

단위: 모터 단위

기본값: 1000

기준 I 변수명: lxx12

Motor[x].WarnFeLimit 는 경고 플래그가 설정되는 지정된 모터에서의 편차 크기를 설정합니다. 편차의 크기가 **Motor[x].WarnFeLimit** 를 초과할 때 모터와 모터가 할당된 좌표계(소속하는 경우)의 Status 비트가 설정됩니다. 좌표계의 Status 비트는 해당 좌표계에 할당된 모든 모터의 Status 비트의 논리합입니다.

이 파라미터를 0 으로 설정하면 경고 편차 리미트 기능이 무효가 됩니다. 이 파라미터가 **Motor[x].FatalFeLimit** 보다 큰 경우에 편차 이상이 처음에 모터를 Disable 로 하므로 경고 Status 비트는 절대 ON 이 되지 않습니다.

Motor[x].CaptureMode 의 비트 1(값 2)이 1 인 경우에 원점 서치 동작, 트리거 포함 JOG 동작과 모션 프로그램의 트리거 포함 동작에서 편차가 **Motor[x].WarnFeLimit** 를 초과할 때 모터는 "트리거"됩니다. 트리거가 이 편차 리미트에 상당하는 토크 레벨에서 발생하므로 이는 "토크 모드" 트리거로 알려져 있습니다.

모터 서보 루프항의 요소

이 장의 설정 요소는 **Motor[x].Servo** 의 서브 구조체의 부분입니다. 이는 명령 궤적과 피드백값에 따른 서보 출력 명령을 계산하기 위해 사용됩니다. 해당 값은 보통 통합 개발 환경(IDE) 프로그램에서의 조정 유틸리티를 사용하고 자동 조정이나 대화형 조정 기능 중 하나에서 또는 양쪽을 사용하여 설정됩니다.

Motor[x].Servo.BreakPosErr 모터 서보의 불감대 크기

설명: 서보 게인 브레이크의 편차 크기

범위: 음이 아닌 부동 소수점

단위: 모터 단위

기본값: 0.0

기준 I 변수명: lxx65

Motor[x].Servo.BreakPosErr 는 모든 방향에서 제로 편차에서 측정된 위치 편차 영역의 크기를 정의합니다. 그 영역 내에서는 변경되는 제어 효과가 있어(또는 전혀 없고) "게인 브레이크" 또는 "불감대 보정"이라는 기능입니다. 게인 브레이크 영역은 보통 "불감대"를 알고리즘적으로 생성하거나 물리적인 불감대 효과를 보정하기 위해 사용됩니다.

Motor[x].Servo.Kbreak 는 이 영역 내의 실효 게인을 제어합니다. 게인 브레이크 영역 밖에서는 게인은 점근적으로 표준 비례 게인항 **Motor[x].Servo.Kp** 에 가까워집니다.

Motor[x].Ctrl 이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.BreakPosErr** 는 사용됩니다.

Motor[x].Servo.EstMinDac 모터 서보의 추정 최소 서보 명령

설명: 유효한 적응 제어를 위한 최소 서보 명령 레벨

범위: 0 ..32,767

단위: 16 비트 명령 단위

기본값: 0

Motor[x].Servo.EstMinDac에서는 적응 제어 알고리즘에 필요한 장치 게인을 추정하기 위해 Power PMAC 이 서보 사이클에서 사용하는 최소 서보 루프 명령 출력값을 지정합니다. 이것은 적응 서보 제어 알고리즘이 선택된 경우에만 사용됩니다. 이것은 +/- 32K 의 범위를 가지는 16 비트 출력 장치 단위이며 이 요소의 이름에 관계없이 출력 장치는 D/A 컨버터이어야 할 필요는 없습니다.

명령 가속도가 0 이외이고 서보 명령 출력의 크기가 이 값 이상인 서보 사이클 간에서 재귀적인 장치 게인 추정 계산이 실행됩니다. 명령 출력의 크기가 매우 낮을 때 장치 가속도는 오프셋, 노이즈 그리고 외란의 영향에 의해 명령 레벨과 그다지 관련이 없습니다.

그 경우에 장치 게인 추정 계산은 안정되지 않습니다. **Motor[x].Servo.EstMinDac** 의 표준값은 **Motor[x].MaxDac** 서보 출력 제한의 1~5%입니다.

Motor[x].Ctrl 이 **Sys.AdaptiveCtrl** 또는 **Sys.GantryXCtrl**(2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능)로 설정되는 경우에 **Motor[x].Servo.EstMinDac** 은 사용됩니다. **Motor[x].Ctrl** 이 **Sys.ServoCtrl**, **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.EstTime 모터 서보의 추정 시간

설명: 유효한 적응 제어에서의 서보 사이클 수

범위: 음이 아닌 정수

단위: 서보 사이클

기본값: 0(무효)

Motor[x].Servo.EstTime 은 적응 서보 제어 알고리즘의 추정 장치 게인을 계산하기 위해 재귀적 알고리즘이 사용되는 서보 사이클 수를 지정합니다. 이것은 적응 서보 제어 알고리즘이 선택된 경우에만 사용됩니다. 기본값 0 값에서는 아무리 적응 제어 알고리즘이 선택되었어도 이 추정/적응 알고리즘은 무효가 됩니다.

이 알고리즘은 명령 토크 출력과 그 결과로 측정되는 가속도 간의 비율을 추정하기 위해 과거의 연속하는 **Motor[x].Servo.EstTime** 서보 사이클에서의 서보 명령 출력과 위치 피드백 데이터를 사용합니다. 명령 가속도값이 0 이외이고 서보 명령 출력의 크기가 **Motor[x].EstMinDac** 이상일 때의 이 서보 사이클 사이에서만 사용됩니다. 장치 게인값의 결과는 상태 요소 **Motor[x].Servo.EstGain** 에 저장되고 서보 알고리즘의 게인 조정값 **Motor[x].Servo.GainFactor** 를 계산하기 위해 사용됩니다.

Motor[x].Servo.EstTime 의 더 낮은 값에서는 장치 관성 변화에 의해 개략적인 응답을 제공하지만 그러나 측정 오차, 양자화 노이즈, 물리적인 외란에 의해 민감해집니다. 일반적으로 이는 적어도 50 서보 사이클의 값으로 설정되어야 합니다. 가장 일반적인 범위는 2.26kHz 의 기본값 서보 주파수에서 100~500 이 됩니다.

Motor[x].Ctrl 이 **Sys.AdaptiveCtrl** 또는 **Sys.GantryXCtrl**(2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능)로 설정되는 경우에 **Motor[x].Servo.EstTime** 은 사용됩니다. **Motor[x].Ctrl** 이 **Sys.ServoCtrl**, **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.Kafb 모터 서보의 가속도 피드백 게인

설명:	PID 가속도 피드백 게인항
범위:	부동 소수점
단위:	16 비트 출력의 LSB/실제 가속도 (모터 단위/서보 사이클/서보 사이클)
기본값:	0.0

Motor[x].Servo.Kafb 는 서보 루프의 2 차 미분 게인항이며 모터의 실제 가속도 피드백("pEnc2"의 피드백 소스를 사용하여 이 사이클의 실제 속도에서 직전 사이클의 실제 속도를 뺀)에 비례한 제어량을 제어 출력에서 빼는 효과를 제공합니다. 이는 실질적으로 전자적 관성으로 기능합니다. **Motor[x].Servo.Kafb** 가 높으면 높을수록 관성 효과는 더 무거워집니다.

16 비트 부호가 있는 출력으로 스케일링되는 서보 출력에 제어량을 산출하기 위해서는 **Motor[x].Servo.Kafb** 는 실제 가속도에 단순히 곱해집니다. 즉 1.0 모터 단위/서보 사이클/서보 사이클의 실제 가속도에서는 1.0 인 **Motor[x].Servo.Kafb** 를 곱하면 16 비트 출력의 1.0LSB 값의 효과가 됩니다(이는 18 비트 DAC 에서의 4.0 LSB 에相当합니다).

2 차 디지털 미분은 많은 양자화 노이즈를 유도하므로 실제 속도가 "E" 다항식 필터 요소 **Motor[x].Servo.Ke1** 및 **Motor[x].Servo.Ke2** 에 의해 처음에 필터링되지 않는 한 이 항은 사용하지 마십시오.

Motor[x].Ctrl 이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.Kafb** 는 사용됩니다.

Motor[x].Servo.Kaff 모터 서보의 가속도 Feed-forward 게인

설명:	PID 가속도 Feed-forward 게인항
범위:	부동 소수점
단위:	16 비트 출력의 LSB/명령 가속도 (모터 단위/서보 사이클/서보 사이클)
기본값:	0.0
기존 I 변수명:	lxx35

Motor[x].Servo.Kaff 에서는 모터의 명령 가속도에 비례한 제어량을 명령 출력에 더합니다. 이것은 관성에서의 지연에 의한 추적 오차 감소를 목적으로 합니다.

서보 출력에 대한 효과를 계산하기 위해 **Motor[x].Servo.Kaff** 는 최종적인 명령 가속도로 단순히 곱해지고 16 비트의 부호가 있는 출력으로 스케일링됩니다. 즉 1.0 모터 단위/(서보 사이클*서보 사이클)의 명령 가속도는 1.0 인 **Motor[x].Servo.Kaff** 를 곱하면 16 비트 출력의 1.0LSB 의 값이 됩니다(이것은 18 비트 DAC 의 4.0LSB 에相当합니다).

이 효과는 적분 게인항의 출력에 더해진다는 점에 주의하십시오(적분기의 입력과 출력 양쪽에 속도 Feed-forward 항이 더해집니다).

서보 갱신 시간이 변경되는 경우에 동일한 효과를 유지하기 위해 제곱에 반비례하여 **Motor[x].Servo.Kaff** 를 변경해야 합니다. 예를 들어 서보 갱신 시간이 반감되는 경우에 동일한 효과를 유지하기 위해 **Motor[x].Servo.Kaff** 는 4 배로 해야 합니다.

Turbo PMAC 서보 알고리즘과 등가인 동작을 위해 이 게인은 다음과 같이 설정할 수 있습니다.

$$Motor[x].Servo.Kaff = I_{xx35} * \frac{I_{xx08} * I_{xx30}}{2^{26} * Motor[x].PosSf}$$

Motor[x].Ctrl 이 **Sys.PidCtrl**, **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.Kaff** 는 사용됩니다.

Motor[x].Servo.Kai 모터 서보의 명령 위치 필터 분자항

설명: 명령 위치의 i 차 다항식 분자항($i = 0 \sim 7$)

범위: 부동 소수점

단위: 없음(단위가 없는 z-변환 계수)

기본값: 1.0(Ka0), 0.0(Ka1~Ka7)

Motor[x].Servo.Kai 항은 서보 알고리즘의 최종적인 명령 위치에 기능하는 2 차 또는 7 차 "A" 다항식 필터 분자항의 배정도 계수입니다. 이 다항식의 전달 함수는 다음과 같습니다.

$$A(z) = Ka_0 + Ka_1 z^{-1} + Ka_2 z^{-2} + Ka_3 z^{-3} + Ka_4 z^{-4} + Ka_5 z^{-5} + Ka_6 z^{-6} + Ka_7 z^{-7}$$

사용되는 모든 **Kai** 항의 합계는 필터가 그 신호에 어떠한 최종적인 스케일링 효과도 갖지 않으므로 정확하게 1.0 과 같아야 합니다(즉, 1.0 의 "DC 게인"을 갖기 위해). 보통 **Ka1** 항과 그보다 높은 차수의 항은 희망하는 동특성을 얻을 수 있게 설정된 다음에 **Ka0** 항은 1 에서 해당 항의 합계를 뺀 값으로 설정합니다. 그렇게 하면 필터에 의한 모든 최종적인 스케일링도 없어집니다.

Motor[x].Ctrl 이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.Kafi** 항은 사용됩니다. **Motor[x].Ctrl** 이 **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.SwPoly7 이 기본값 0 인 경우에 **Ka1** 과 **Ka2** 의 항만 2 차 다항식으로 이 필터에서 사용됩니다. **Motor[x].Servo.SwPoly7** 이 1 로 설정되는 경우에 7 개의 **Kai** 항이 모두 사용되고 결과적으로 7 차 다항식이 됩니다. 더 많은 효과를 얻을 수 있게 되지만 실행 시간이 더 길어집니다.

Motor[x].Servo.Kbi 모터 서보의 실제 위치 필터 분자항

설명: 실제 위치의 i 차 다항식 분자항($i = 0 \sim 7$)

범위: 부동 소수점

단위: 없음(단위가 없는 z-변환 계수)

기본값: 1.0(Kb0), 0.0(Kb1 - Kb7)

Motor[x].Servo.Kbi 항은 서보 알고리즘의 최종적인 아우터 루프 실제 위치에 기능하는 2 차 또는 7 차 "B" 다항식 필터 분자항의 배정도 계수입니다. 이 다항식의 전달 함수는 다음과 같습니다.

$$B(z) = Kb_0 + Kb_1 z^{-1} + Kb_2 z^{-2} + Kb_3 z^{-3} + Kb_4 z^{-4} + Kb_5 z^{-5} + Kb_6 z^{-6} + Kb_7 z^{-7}$$

사용되는 모든 **Kbi** 항의 합계는 필터가 그 신호에 어떠한 최종적인 스케일링 효과도 갖지 않으므로 정확하게 1.0 과 같게 해야 합니다(즉, 1.0 의 "DC 게인"을 갖기 위해). 보통 **Kb1** 항과 그보다 높은 차수의 항은 희망하는 동특성을 얻을 수 있게 설정된 다음에 **Kb0** 항은 1 에서 해당 항의 합계를 뺀 값으로 설정합니다. 그렇게 하면 필터에 의한 모든 최종적인 스케일링도 없어집니다.

Motor[x].Ctrl 이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.Kbi** 항은 사용됩니다. **Motor[x].Ctrl** 이 **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.SwPoly7 이 기본값 0 인 경우에 **Kb1** 과 **Kb2** 의 항만 2 차 다항식으로 이 필터에서 사용됩니다. **Motor[x].Servo.SwPoly7** 이 1 로 설정되는 경우에 7 개의 **Kbi** 항이 모두 사용되고 결과적으로 7 차 다항식이 됩니다. 더 많은 효과도 얻을 수 있게 되지만 실행 시간이 더 길어집니다.

Motor[x].Servo.Kbreak 모터 서보의 불감대 게인

설명: 서보 "게인 브레이크" 시의 상대 게인

범위: 음이 아닌 부동 소수점

단위: 비례 게인 **Kp** 와의 비

기본값: 0.0

기존 I 변수명: lxx64

Motor[x].Servo.Kbreak 는 "게인 브레이크" 영역 내, 즉 위치 편차가 **Motor[x].Servo.BreakPosErr** 미만인 크기의 영역 내에서의 서보 루프 비례 게인을 제어합니다. 이 게인 브레이크 영역은 일반 "불감대"를 알고리즘적으로 생성하기 위함이거나 물리적인 불감대 효과를 보정하기 위해 사용됩니다.

Motor[x].Servo.Kbreak 는 표준 비례 게인항 **Motor[x].Servo.Kp** 와 관련되어 표시됩니다. 즉 **Motor[x].Servo.Kbreak** 의 1.0 의 값은 게인 브레이크하는 영역의 안쪽과 바깥쪽에서 동일한 게인을 제공합니다(이 영역의 목적을 무효로 합니다). 0.0 의 값은 이 영역 내에서 불감대를 생성합니다. 2.0 의 값은 영역 내에서 2 배의 게인을 제공합니다(물리적인 불감대를 보정하는 것입니다).

게인 브레이크 영역 밖에서는 게인은 점근적으로 표준 비례 게인항 **Motor[x].Servo.Kp** 에 가까워집니다.

Motor[x].Ctrl 이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.break** 는 사용됩니다. **Motor[x].Ctrl** 이 **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.Kci 모터 서보의 편차 필터 분자항

설명: 서보 에러(위치 편차)의 i 차 다항식 분자항($i = 1 \sim 7$)

범위: 부동 소수점

단위: 없음(단위가 없는 z-변환 계수)

기본값: 0.0

기존 I 변수명: lxx36(Kc1), lxx37(Kc2)

Motor[x].Servo.Kci 항은 서보 알고리즘의 아우터 루프(위치)의 위치 편차에 기능하는 2 차 또는 7 차 "C" 다항식 필터 분자항의 배정도 계수입니다. 이 다항식의 전달 함수는 다음과 같습니다.

$$C(z) = 1 + Kc_1 z^{-1} + Kc_2 z^{-2} + Kc_3 z^{-3} + Kc_4 z^{-4} + Kc_5 z^{-5} + Kc_6 z^{-6} + Kc_7 z^{-7}$$

이 필터의 "DC 게인"은 사용되는 계수 전체를 1 에 더한 합계와 같고 이 게인이 전체 서보 루프 게인의 일부라는 점에 주의하십시오.

Motor[x].Ctrl 이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.Kci** 항은 사용됩니다. **Motor[x].Ctrl** 이 **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.SwPoly7 이 기본값 0 인 경우에 **Kc1** 과 **Kc2** 의 항만 2 차 다항식으로 이 필터에서 사용됩니다. **Motor[x].Servo.SwPoly7** 이 1 로 설정되는 경우에 7 개의 **Kci** 항이 모두 사용되고 결과적으로 7 차 다항식이 됩니다. 더 많은 효과를 얻을 수 있게 되지만 실행 시간이 더 길어집니다.

Motor[x].Servo.Kdi 모터 서보의 편차 필터 분모항

설명: 서보 에러(위치 편차)의 i 차 다항식 분모항($i = 1 \sim 7$)

범위: 부동 소수점

단위: 없음(단위가 없는 z-변환 계수)

기본값: 0.0

기존 I 변수명: lxx38(Kd1), lxx39(Kd2)

Motor[x].Servo.Kdi 항은 서보 알고리즘의 아우터 루프(위치)의 위치 편차에 기능하는 2 차 또는 7 차 "D" 다항식 필터 분모항의 배정도 계수입니다. 이 다항식의 전달 함수는 다음과 같습니다.

$$D(z) = \frac{1}{1 + Kd_1 z^{-1} + Kd_2 z^{-2} + Kd_3 z^{-3} + Kd_4 z^{-4} + Kd_5 z^{-5} + Kd_6 z^{-6} + Kd_7 z^{-7}}$$

이 필터의 "DC 게인"은 사용되는 계수 전체를 1에 더한 합계로 1을 나눈 값과 같고 이 게인이 전체 서보 루프 게인의 일부라는 점에 주의하십시오.

Motor[x].Ctrl 이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.Kdi** 항은 사용됩니다. **Motor[x].Ctrl** 이 **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.SwPoly7 이 기본값 0 인 경우에 **Kd1** 과 **Kd2** 의 항만 2 차 다항식으로 이 필터에서 사용됩니다. **Motor[x].Servo.SwPoly7** 이 1 로 설정되는 경우에 7 개의 **Kdi** 항이 모두 사용되고 결과적으로 7 차 다항식이 됩니다. 더 많은 효과를 얻을 수 있게 되지만 실행 시간이 더 길어집니다.

Motor[x].Servo.Kei 모터 서보의 이너 루프 필터 분모항

설명: 이너 속도 루프의 i 차 다항식 분모항($i = 1 \sim 2$)

범위: 부동 소수점

단위: 없음(단위가 없는 z-변환 계수)

기본값: 0.0

Motor[x].Servo.Kbi 항은 서보 알고리즘의 이너(속도) 루프의 실제 위치로 기능하는 2 차 "E" 다항식 필터 분모항의 배정도 계수입니다. 이 다항식의 전달 함수는 다음과 같습니다.

$$E(z) = \frac{1}{1 + Ke_1 z^{-1} + Ke_2 z^{-2}}$$

이 필터의 "DC 게인"은 사용되는 계수 전체를 1에 더한 합계로 1을 나눈 값과 같고 이 게인이 전체 이너 루프 게인의 일부라는 점에 주의하십시오. 해당 항이 변경되는 경우에 전체 DC 게인을 유지하기 위해 **Kvfb** 및(또는) **Kvifb** 게인으로 변경을 부정해야 합니다.

Motor[x].Ctrl 이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.Kei** 항은 사용됩니다. **Motor[x].Ctrl** 이 **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.Kfi 모터 서보의 명령 속도 필터 분모항

설명: 명령 속도의 i 차 다항식 분모항($i = 1 \sim 2$)

범위: 부동 소수점

단위: 없음(단위가 없는 z-변환 계수)

기본값: 0.0

Motor[x].Servo.Kai 항은 서보 알고리즘의 Feed-forward 경로에서의 최종적인 명령 속도에 기능하는 2 차 "F" 다항식 필터 분모항의 배정도 계수입니다. 이 다항식의 전달 함수는 다음과 같습니다.

$$F(z) = \frac{1}{1 + Kf_1 z^{-1} + Kf_2 z^{-2}}$$

해당 항은 측정과 양자화 노이즈를 가지는 마스터 엔코더가 모터의 최종적인 명령 궤적 생성에 사용되는 경우(위치 추종과 외부 타임 베이스 모드와 같이)에 해당 Feed-forward 신호에 로 패스 필터를 적용하는 데 보통 사용됩니다. Feed-forward 게인은 해당 고주파수 노이즈 성분을 제거하고 결과적으로 개략적인 동작이 됩니다.

일반적으로 순수하게 수학적으로 생성되는 궤적에는 해당 항은 0 그대로 둡니다. 이 경우에 명령 궤적에는 현저한 노이즈가 존재하지 않기 때문입니다.

이 필터의 "DC 게인"은 2 개의 계수를 1 에 더한 합계로 1 을 나눈 값과 같고 이 게인이 전체의 Feed-forward 경로의 일부라는 점에 주의하십시오.

Motor[x].Ctrl 이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.Kfi** 항은 사용됩니다. **Motor[x].Ctrl** 이 **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.Kfff 모터 서보의 마찰 Feed-forward 게인

설명: 서보 마찰 Feed-forward 게인

범위: 부동 소수점

단위: 16 비트 출력의 LSB

기본값: 0.0

기존 I 변수명: lxx68

Motor[x].Servo.Kfff 는 최종적인 명령 속도의 부호에 비례하는 바이어스항을 모터의 서보 루프 출력에 더합니다. 즉 명령 속도가 양인 경우 **Motor[x].Servo.Kfff** 는 해당 출력에 더합니다. 명령 속도가 음인 경우 **Motor[x].Servo.Kfff** 는 해당 출력에서 뺍니다. 명령 속도가 제로인 경우에 어떤 값도 출력에서 더해지거나 빼지지 않습니다.

이 파라미터의 목적은 주로 기계적 마찰에 의한 오차를 극복하기 위한 것입니다. 이것은 "마찰 Feed-forward" 향으로 판단할 수 있습니다. 이것은 피드백 정보를 일절 사용하지 않는 Feed-forward 향이므로 시스템 안정성에 직접적인 효과는 없습니다. 이것은 마찰에서 발생하는 오차를 수정하기 때문에 특히 방향 전환성에서 적분 게인의 시상수와 잠재적인 안정성의 문제 등이 없이 사용할 수 있습니다.

소프트웨어 스위치의 **Motor[x].SwFffInt** 가 기본값 0 인 경우에 이 항의 동작은 서보 출력에 직접 더합니다. **Motor[x].SwFffInt** 가 1로 설정되는 경우에 서보 루프 적분기의 입력에 더합니다.

Motor[x].Servo.Kfff 는 16 비트 부호가 있는 출력에 스케일링됩니다. 즉 1.0 인 **Motor[x].Servo.Kfff** 는 16 비트 출력의 +/-1.0 LSB 의 값이 됩니다(이것은 18 비트 DAC에서의 +/-4.0 LSB 에相当합니다).

서보 알고리즘과 등가인 동작에는 **Motor[x].Servo.Kfff** 는 Turbo PMAC 의 lxx68 과 동일한 값으로 설정합니다.

Motor[x].Ctrl 이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.Kfff** 는 사용됩니다. **Motor[x].Ctrl** 이 **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.Ki 모터 서보의 적분 게인

설명: PID 의 적분 게인항

범위: 부동 소수점

단위: 16 비트 출력의 LSB/적분 오차(모터 단위*서보 사이클)

기본값: 0.001

기준 I 변수명: lxx33

Motor[x].Servo.Ki 는 (아마도 변경된)위치 편차의 시간 적분에 비례한 제어 출력을 제공하는 서보 루프의 주요 적분 게인항입니다. 적분기의 출력의 크기는 **Motor[x].Servo.MaxInt** 로 제한됩니다. **Motor[x].Servo.MaxInt** 가 0.0 의 값에서는 서보 출력에 적분기의 효과는 **Motor[x].Servo.Ki** 의 값과 관계없이 제로가 됩니다.

출력이 포화되는 경우(출력의 크기가 **Motor[x].MaxDac** 와 같은 경우)에 그 이상의 편차는 적분기에 더해지지 않습니다. 또한 **Motor[x].Servo.SwZvInt** = 1 인 경우에 하나의 동작이 명령되었을 때(명령 속도가 제로 이외) 모든 경우에 출력에 적분기의 제어량은 일정한 그대로가 됩니다.

서보 갱신 시간이 변경되는 경우에 동일한 효과를 유지하기 위해서는 **Motor[x].Servo.Ki** 를 동일한 방향에 비례하여 변경해야 합니다. 예를 들어 서보 갱신 시간이 반감되는 경우에 동일한 효과를 유지하기 위해 **Motor[x].Servo.Ki** 는 절반으로 해야 합니다.

적분기의 출력에 더해지는 속도 게인항의 **Motor[x].Servo.Kvfb**(피드백)와 **Motor[x].Servo.Kvff**(Feed-forward)가 사용되는 경우에 적분기는 실질적으로 위치 루프 중에 존재합니다. 그러나 적분기의 입력에 더해지는 속도 게인항의 **Motor[x].Servo.Kvfb**(피드백)와 **Motor[x].Servo.Kvff**(Feed-forward)가 사용되는 경우에 적분기는 실질적으로 속도 루프 중에 존재합니다. 일반적으로는 위치 루프의 적분기는 궤적 추종에 더 효과가 있고 속도 루프의 적분기는 외란 제거에 더 효과가 있습니다.

Turbo PMAC 서보 알고리즘과 등가인 동작에서는 이 게인은 다음과 같이 설정할 수 있습니다.

$$Motor[x].Servo.Ki = \frac{Ixx33}{2^{23} * Ixx08}$$

Motor[x].Ctrl 이 **Sys.PidCtrl**, **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.Ki** 는 사용됩니다.

Motor[x].Servo.Kp 모터 서보의 비례 게인

설명: PID 비례 게인항

범위: 부동 소수점

단위: 16 비트 출력의 LSB/위치 편차(모터 단위)

기본값: 4

기존 I 변수명: Ixx30

Motor[x].Servo.Kp 는 모터의 위치 편차(명령 위치와 현재 위치의 차)에 비례한 제어 출력을 제공하는 서보 루프의 주요한 비례 게인항입니다. 이는 실질적으로 전자 스프링으로 기능합니다. **Motor[x].Servo.Kp** 가 높으면 높을수록 "스프링"은 더 강해집니다. 너무 낮은 값이면 결과적으로 둔감한 성능이 됩니다. 지나치게 높은 값은 편차에 대한 연속적인 과잉 반응에서 "맥놀이"를 일으킬 수 있습니다.

서보 출력에 제어량을 계산하기 위해서는 단순히 **Motor[x].Servo.Kp** 에 위치 편차를 곱하고 16 비트 부호가 있는 출력으로 스케일링됩니다. 즉 1.0 모터 단위의 위치 편차에서는 1.0 인 **Motor[x].Servo.Kp** 를 곱하면 16 비트 출력의 1.0 LSB 의 값이 됩니다(이것은 18 비트 DAC 에서의 4.0 LSB 에相当합니다).

Turbo PMAC 서보 알고리즘과 등가인 동작에서는 이 게인은 다음과 같이 설정할 수 있습니다.

$$Motor[x].Servo.Kp = Ixx30 * \frac{Ixx08}{2^{19} * Motor[x].PosSf}$$

Motor[x].Ctrl 이 **Sys.PidCtrl**, **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.Kp** 는 사용됩니다.

Motor[x].Servo.Kvfb 모터 서보의 속도 피드백 게인

설명: PID 속도 피드백("미분") 게인항

범위: 부동 소수점

단위: 16 비트 출력의 LSB/실제 속도(모터 단위/서보 사이클)

기본값: 40

기준 I 변수명: Ixx31

Motor[x].Servo.Kvfb 는 서보 루프의 주요한 미분 게인항이며 모터의 실제 속도 피드백("Pos2"의 피드백 소스를 사용하여 이 사이클의 실제 위치에서 직전 사이클의 실제 위치를 뺀)에 비례한 제어량을 제어 출력에서 빼는 효과를 제공합니다. 이는 실질적으로 전자 댐퍼로 기능합니다. **Motor[x].Servo.Kvfb** 가 높으면 높을수록 감쇠 효과는 더 커집니다.

서보 출력에 제어량을 계산하기 위해서는 단순히 **Motor[x].Servo.Kvfb** 에 실제 속도를 곱하고 16 비트 부호가 있는 출력으로 스케일링됩니다. 즉 1.0 모터 단위/서보 사이클의 실제 속도에서는 1.0 인 **Motor[x].Servo.Kvfb** 를 곱하면 16 비트 출력의 1.0LSB 의 값이 됩니다(이는 18 비트 DAC 에서의 4.0 LSB 에 상당합니다).

이 제어량은 적분 게인항의 출력에서 빼진다는 점에 주의하십시오. 동류 항인 **Motor[x].Servo.Kvifb** 의 제어량은 적분 게인항의 입력에서 빼집니다. 일반적으로는 적분기의 출력에서 제어량을 빼는 것은 궤적 추종에 우수한 성능을 제공합니다. 적분기의 입력에서 제어량을 빼는 것은 외란 제거에 더 효과가 있습니다.

적절하게 조정된 속도 루프 앰프가 구동되는 경우에 해당 앰프는 충분한 감쇠를 제공하므로 **Motor[x].Servo.Kvfb** 는 제로로 설정해야 합니다. 전류 루프(토크) 앰프가 구동되는 경우 또는 Power PMAC 이 모터를 전류(commutation)하는 경우에 해당 앰프는 감쇠를 제공하지 않으므로 안정성에 감쇠를 제공하기 위해 **Motor[x].Servo.Kvfb** 는 제로보다 큰 값으로 설정해야 합니다.

글로벌한 서보 인터럽트 주파수의 변경 또는 **Motor[x].Stime** 에 의한 해당 모터 자체의 주기 변경에 관계없이 서보 갱신 시간이 변경되는 경우에 동일한 감쇠 효과를 유지하기 위해서는 **Motor[x].Servo.Kvfb** 를 반비례하여 변경해야 합니다. 예를 들어 서보 갱신 시간이 반감되는 경우에 동일한 효과를 유지하기 위해서는 **Motor[x].Servo.Kvfb** 는 2 배로 해야 합니다.

Ixx96 의 비트 1(값 2) = 0 에서 Turbo PMAC 서보 알고리즘과 등가인 동작에서는 이 게인은 다음과 같이 설정할 수 있습니다.

$$Motor[x].Servo.Kvfb = Ixx31 * \frac{Ixx09 * Ixx30}{2^{26} * Motor[x].Pos2Sf}$$

Turbo PMAC 의 Ixx96 의 비트 1 = 1 인 경우에 Power PMAC 의 **Kvifb** 게인이 대신 사용됩니다.

Motor[x].Ctrl 이 **Sys.PidCtrl**, **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.Kvfb** 는 사용됩니다.

Motor[x].Servo.Kvff 모터의 속도 Feed-forward 게인

설명: PID 속도 Feed-forward 게인항

범위: 부동 소수점

단위: 16 비트 출력의 LSB/명령 속도(모터 단위/서보 사이클)

기본값: 40

기준 I 변수명: Ixx32

Motor[x].Servo.Kvff에서는 모터의 명령 속도에 비례한 제어량을 제어 출력에 더합니다. 이 목적은 **Motor[x].Servo.Kvfb**, 아날로그 회전 속도계 피드백 또는 물리적인 감쇠 효과에서 발생하는 추종 오차를 감소시키는 것입니다.

서보 출력에 제어량을 계산하기 위해서는 단순히 **Motor[x].Servo.Kvff**에 최종적인 명령 속도를 곱하고 16 비트 부호가 있는 출력으로 스케일링됩니다. 즉 1.0 모터 단위/서보 사이클의 명령 속도에서는 1.0 인 **Motor[x].Servo.Kvff**를 곱하면 16 비트 출력의 1.0LSB 값이 됩니다(이는 18 비트 DAC에서의 4.0 LSB에相当합니다).

이 제어량이 적분 게인항의 출력에 더해진다는 점에 주의하십시오. 동류 항인 **Motor[x].Servo.Kviff**의 제어량은 적분 게인항의 입력에 더합니다. 일반적으로는 적분기의 출력에 제어량을 더하는 것은 궤적 추종에 우수한 성능을 제공합니다. 적분기의 입력에 제어량을 더하는 것은 외란 제거에 더 효과가 있습니다.

전류 루프(토크) 앰프가 구동되는 경우에 일반 추종 오차를 최소한으로 하기 위해 **Motor[x].Servo.Kvff**는 **Motor[x].Servo.Kvfb**와 같아야 합니다(또는 약간 크게). 속도 루프 앰프가 구동되는 경우에 추종 오차를 최소한으로 하기 위해 **Motor[x].Servo.Kvff**는 보통 **Motor[x].Servo.Kvfb**보다 실질적으로 크게 합니다.

서보 갱신 시간이 변경되는 경우에 동일한 효과를 유지하기 위해서는 **Motor[x].Servo.Kvff**를 비례하여 변경해야 합니다. 예를 들어 서보 갱신 시간이 반감되는 경우에 동일한 효과를 유지하기 위해서는 **Motor[x].Servo.Kvff**는 2배로 해야 합니다.

Ixx96의 비트 1(값 2) = 0에서 Turbo PMAC 서보 알고리즘과 등가인 동작에서는 이 게인은 다음과 같이 설정할 수 있습니다.

$$Motor[x].Servo.Kvff = Ixx32 * \frac{Ixx09 * Ixx30}{2^{26} * Motor[x].Pos2Sf}$$

Turbo PMAC의 Ixx96의 비트 1 = 1인 경우에 Power PMAC의 **Kviff** 게인이 대신 사용됩니다.

Motor[x].Ctrl이 **Sys.PidCtrl**, **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl**로 설정되는 경우에 **Motor[x].Servo.Kvff**는 사용됩니다.

Motor[x].Servo.Kvifb 모터 서보의 속도 피드백 게인(적분기로)

설명:	서보 레이트 피드백(적분기로의) 게인 항
범위:	부동 소수점
단위:	16 비트 출력의 LSB/실제 속도(모터 단위/서보 사이클)
기본값:	0

Motor[x].Servo.Kvifb 는 서보 루프의 2 번째 미분 게인항이며 모터의 실제 속도 피드백("Pos2"의 피드백 소스를 사용하여 이 사이클의 실제 위치에서 직전 사이클의 실제 위치를 뺀)에 비례한 제어량을 제공합니다. 이는 실질적으로 전자 댐퍼로 기능합니다. **Motor[x].Servo.Kvff** 가 높으면 높을수록 감쇠 효과는 더 강해집니다.

서보 출력에 제어량을 산출하기 위해서는 단순히 **Motor[x].Servo.Kvifb** 에 실제 속도를 곱하고 16 비트 부호가 있는 출력으로 스케일링됩니다. 즉 1.0 모터 단위/서보 사이클의 실제 속도에서는 1.0 인 **Motor[x].Servo.Kvifb** 를 곱하면 16 비트 출력의 1.0LSB 값이 됩니다(이는 18 비트 DAC 에서의 4.0 LSB 에相当합니다).

이 제어량이 적분 게인항의 입력에서 빠진다는 점에 주의하십시오. 동류 항인 **Motor[x].Servo.Kvff** 의 제어량은 적분 게인항의 출력에서 빠집니다. 일반적으로는 적분기의 출력에서 제어량을 빼는 것은 궤적 추종에 우수한 성능을 제공합니다. 적분기의 입력에서 제어량을 빼는 것은 외란 제거에 더 효과가 있습니다.

적절하게 조정된 속도 루프 앰프가 구동되는 경우에 앰프는 충분한 감쇠를 제공하므로 **Motor[x].Servo.Kvifb** 는 제로로 설정해야 합니다. 전류 루프(토크) 앰프가 구동되는 경우 또는 Power PMAC 이 모터를 전류(commutation)하는 경우에 앰프는 감쇠를 제공하지 않으므로 안정성에 감쇠를 제공하기 위해 **Motor[x].Servo.Kvifb** 는 0 보다 큰 값으로 설정해야 합니다.

글로벌 서보 인터럽트 주파수의 변경 또는 **Motor[x].Stime** 에 의한 해당 모터 자체의 주기 변경에 관계없이 서보 갱신 시간이 변경되는 경우에 동일한 감쇠 효과를 유지하기 위해서는 **Motor[x].Servo.Kvifb** 를 반비례하여 변경해야 합니다. 예를 들어 서보 갱신 시간이 반감되는 경우에 동일한 효과를 유지하기 위해서는 **Motor[x].Servo.Kvifb** 는 2 배로 해야 합니다.

Ixx96 의 비트 1(값 2) = 1 에서 Turbo PMAC 서보 알고리즘과 등가인 동작에서는 이 게인은 다음과 같이 설정할 수 있습니다.

$$Motor[x].Servo.Kvifb = Ixx31 * \frac{Ixx09 * Ixx30}{2^{26} * Motor[x].Pos2Sf}$$

Turbo PMAC 의 Ixx96 의 비트 1 = 0 인 경우에 Power PMAC 의 **Kvfb** 게인이 대신 사용됩니다.

Motor[x].Ctrl 이 **Sys.PidCtrl**, **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.Kvifb** 는 사용됩니다.

Motor[x].Servo.Kviff 모터 서보의 속도 Feed-forward 게인(적분기로)

설명:	서보 레이트 Feed-forward(적분기로) 게인항
범위:	부동 소수점
단위:	16 비트 출력의 LSB/명령 속도(모터 단위/서보 사이클)
기본값:	0

Motor[x].Servo.Kviff에서는 모터의 명령 속도에 비례한 제어량을 제어 출력에 더합니다. 목적은 **Motor[x].Servo.Kvifb**, 아날로그 회전 속도계 피드백 또는 물리적인 감쇠 효과에서 발생하는 추종 오차를 감소시키는 것입니다.

서보 출력에 제어량을 산출하기 위해서는 단순히 **Motor[x].Servo.Kviff**에 최종적인 명령 속도를 곱하고 16 비트 부호가 있는 출력으로 스케일링됩니다. 즉 1.0 모터 단위/서보 사이클의 명령 속도에서는 1.0인 **Motor[x].Servo.Kviff**를 곱하면 16 비트 출력의 1.0LSB 값이 됩니다(이는 18 비트 DAC에서의 4.0 LSB에 상당합니다).

이 제어량이 적분 게인항의 입력에 더해진다는 점에 주의하십시오. 동류 항인 **Motor[x].Servo.Kvff**의 제어량은 적분 게인항의 출력에 더합니다. 일반적으로는 적분기의 출력에 제어량을 더하는 것은 궤적 추종에 우수한 성능을 제공합니다. 적분기의 입력에 제어량을 더하는 것은 외란 제거에 더 효과가 있습니다.

전류 루프(토크) 앰프가 구동되는 경우에 추종 오차를 최소한으로 하기 위해 **Motor[x].Servo.Kviff**는 **Motor[x].Servo.Kvfb**와 보통 같아야 합니다(또는 약간 크게). 속도 루프 앰프가 구동되는 경우에 추종 오차를 최소한으로 하기 위해 **Motor[x].Servo.Kviff**는 **Motor[x].Servo.Kvifb**보다 실질적으로 크게 합니다.

서보 갱신 시간이 변경되는 경우에 동일한 효과를 유지하기 위해서는 **Motor[x].Servo.Kviff**를 비례하여 변경해야 합니다. 예를 들어 서보 갱신 시간이 반감되는 경우에 동일한 효과를 유지하기 위해서는 **Motor[x].Servo.Kviff**는 2배로 해야 합니다.

Ixx96의 비트 1(값 2) = 1에서 Turbo PMAC 서보 알고리즘과 등가인 동작에서는 이 게인은 다음과 같이 설정할 수 있습니다.

$$Motor[x].Servo.Kviff = Ixx32 * \frac{Ixx09 * Ixx30}{2^{26} * Motor[x].Pos2Sf}$$

Turbo PMAC의 Ixx96의 비트 1 = 0인 경우에 Power PMAC의 **Kvff** 게인이 대신 사용됩니다.

Motor[x].Ctrl이 **Sys.PidCtrl**, **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl**로 설정되는 경우에 **Motor[x].Servo.Kviff**는 사용됩니다.

Motor[x].Servo.Kxig 모터 서보의 크로스 커플링 적분 게인

설명:	크로스 커플링 적분 게인
범위:	부동 소수점
단위:	16 비트 출력의 LSB/적분 편차의 차(모터 단위*서보 사이클)
기본값:	0.0

Motor[x].Servo.Kxig 는 갠트리 모터에서의 크로스 커플링의 적분 게인항입니다. 주축과 종축 모터 간에서의 위치 편차의 차를 시간 적분에 비례한 해당 모터에서의 제어량을 제공합니다. 이것은 이 편차의 차를 제로로 하도록 기능합니다. 그 제어량은 모터 자체의 명령 위치와 실제 위치에 따른 서보 항의 제어 출력에 더해집니다.

크로스 커플링된 모터 페어의 낮은 번호(보통 주축 모터)의 **Motor[x].Ctrl** 이 **Sys.GantryXCtrl** 로 설정되는 경우에만 이 게인항이 사용됩니다. 종축 모터는 주축 모터의 번호보다 하나 큰 번호에 할당할 수 있어야 합니다. 주축 모터의 **Motor[x].ExtraMotors** 는 1 로 설정하십시오. 듀얼 모터 알고리즘의 각 모터에는 그 자체의 서보 출력에 영향을 주는 그 자체의 **Kxig** 항이 있습니다. 일반적으로는 두 모터의 해당 항이 동일하지 않은 경우에 매우 가까운 값이 됩니다.

Motor[x].Servo.Kxpg 모터 서보의 크로스 커플링 비례 게인

설명:	크로스 커플링 비례 게인
범위:	부동 소수점
단위:	16 비트 출력의 LSB/편차의 차(모터 단위)
기본값:	0.0

Motor[x].Servo.Kxpg 는 갠트리 모터에서의 크로스 커플링의 비례 게인항입니다. 주축과 종축 모터 간 위치 편차의 차에 비례한 이 모터에서의 제어량을 제공합니다. 이것은 이 편차의 차를 제로로 하도록 기능합니다. 그 제어량은 모터 자체의 명령 위치와 실제 위치에 따른 서보 항의 제어 출력에 더해집니다.

크로스 커플링된 모터 페어의 낮은 번호(보통 주축 모터)의 **Motor[x].Ctrl** 이 **Sys.GantryXCtrl** 로 설정되는 경우에만 이 게인항이 사용됩니다. 종축 모터는 주축 모터의 번호보다 하나 큰 번호에 할당할 수 있어야 합니다. 주축 모터의 **Motor[x].ExtraMotors** 는 1 로 설정하십시오. 듀얼 모터 알고리즘의 각 모터에는 그 자체의 서보 출력에 영향을 주는 그 자체의 **Kxpg** 항이 있습니다. 일반적으로는 두 모터의 해당 항이 동일하지 않은 경우에 매우 가까운 값이 됩니다.

Motor[x].Servo.Kxvg 모터 서보의 크로스 커플링 미분 게인

설명:	크로스 커플링 미분 게인
범위:	부동 소수점
단위:	16 비트 출력의 LSB/편차의 차(모터 단위/서보 사이클)
기본값:	0.0

Motor[x].Servo.Kxvg 는 갠트리 모터에서의 크로스 커플링의 미분 게인항입니다. 주축과 종축 모터 간 위치 편차의 변화율의 차에 비례한 이 모터의 제어량을 제공합니다. 이것은 이 편차의 차를 제로로 하도록 기능합니다. 그 제어량은 모터 자체의 명령 위치와 실제 위치에 따른 서보 항의 제어 출력에 더해집니다.

크로스 커플링된 모터 페어의 낮은 번호(보통 주축 모터)의 **Motor[x].Ctrl** 이 **Sys.GantryXCtrl** 로 설정되는 경우에만 이 게인항이 사용됩니다. 종축 모터는 주축 모터의 번호보다 하나 큰 번호에 할당할 수 있어야 합니다. 주축 모터의 **Motor[x].ExtraMotors** 는 1 로 설정하십시오. 듀얼 모터 알고리즘의 각 모터에는 그 자체의 서보 출력에 영향을 주는 그 자체의 **Kxvg** 항이 있습니다. 일반적으로는 두 모터의 해당 항이 동일하지 않은 경우에 매우 가까운 값이 됩니다.

Motor[x].Servo.MaxDR 모터 서보의 최대 게인 감쇠율

설명:	최대 장치 게인에서의 적응 서보 클로즈드 루프의 감쇠율
범위:	음이 아닌 부동 소수점
단위:	없음(비율)
기본값:	0.0(무효)

Motor[x].Servo.MaxDR 는 모터에 생성하기 위해 알고리즘이 시도해야 하는 클로즈드 루프 감쇠율(ζ)을 지정합니다. 이것은 적응 서보 알고리즘으로 추정된 장치 게인이 최대값(최소 관성을 의미)이며 알고리즘이 보정할(**Motor[x].Servo.MaxGainFactor** 의 설정) 때의 값입니다. 단, **Motor[x].Servo.MinDR** 가 기본값 0.0 인 경우에 적응 알고리즘에서는 클로즈드 루프 감쇠율(또는 고유 주파수) 조정은 수행하지 않습니다. 그 대신 공칭값에서 그 값을 일정하게 유지합니다.

Motor[x].Servo.MinDR 가 0.0 보다 큰 값이고 감쇠율 자동 조정을 유효로 하는 경우 **Motor[x].Servo.MaxDR**, **Motor[x].Servo.MinW** 그리고 **Motor[x].MaxW** 에서는 고유 주파수와 감쇠율의 변화 범위를 정의하기 위해 최적의 설정을 수행해야 합니다. 해당 항은 보통 IDE 의 조정 제어의 대화형 창에 의해 설정됩니다.



경고

Motor[x].Servo.MinDR 과 **Motor[x].Servo.MinW** 가 0.0 보다 큰 값으로 설정되어 게인 스케줄의 적응 제어가 유효한 경우에 **Motor[x].Servo.MaxDR** 은 기본값 0.0 그대로 두면 안 됩니다. 이것은 최소 시스템 관성에서의 감쇠율을 지정하지 않는 것입니다. 이것은 시스템의 위험한 진동으로 이어질 수도 있습니다.

자동적인 감쇠율 조정이 유효한 경우에 적응 서보 알고리즘은 추정된 장치 게인의 함수로 그 내부 게인을 설정하고 최대 장치 게인(최소 관성)에서의 **MaxDR** 의 값과 최소의 장치 게인(최대 관성)에서의 **MinDR** 의 값 사이에서 클로즈드 루프 감쇠율을 직선적으로 변화시킵니다. 자동적인 감쇠율 조정은 관성 변화에서의 일정한 성능을 유지하고자 하는 요구와 시스템의 물리적 제한 간의 트레이드 오프의 관계가 됩니다.

Motor[x].Servo.MaxDR 은 V1.6 펌웨어(2014 년 1/4 분기 출시)의 새로운 기능입니다. **Motor[x].Servo.MinDR** 이 기본값 0.0 인 경우에 동작은 이전 펌웨어 버전과 호환성이 있고 적응 서보 알고리즘의 자동적인 감쇠율 조정을 지원하지 않습니다.

Motor[x].Ctrl 이 **Sys.AdaptiveCtrl** 또는 **Sys.GantryXCtrl** 로 설정되는 경우에 **Motor[x].Servo.MaxDR** 은 사용됩니다. **Motor[x].Ctrl** 이 **Sys.ServoCtrl**, **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.MaxGainFactor 모터 서보의 최대 게인 팩터

설명: 최대 허용 상대 게인

범위: 음이 아닌 부동 소수점

단위: 없음(비율)

기본값: 1.0

Motor[x].Servo.MaxGainFactor 에서는 적응 제어 알고리즘에서 사용되는 최대 상대 서보 게인을 지정합니다. 그리고 표준 저장 서보 게인과의 비로 표시됩니다. 이것은 추정되는 최고 시스템 관성(최저 장치 게인)에서의 변화를 제한합니다. 이것은 적응 서보 제어 알고리즘이 선택된 경우에만 사용됩니다. 예를 들어 3.0 으로 설정되는 경우에 결과로 발생하는 게인은 표준 저장 서보 게인값의 3 배를 상회하지 않습니다.

Motor[x].Servo.MaxGainFactor 의 표준값은 1.0~10.0 의 범위가 됩니다. 이 파라미터는 적응 제어 알고리즘에서의 잘못된 결과를 방지하는 중요한 보호 방법입니다.

Motor[x].Ctrl 이 **Sys.AdaptiveCtrl** 또는 **Sys.GantryXCtrl**(2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능)로 설정되는 경우에 **Motor[x].Servo.MaxGainFactor** 는 사용됩니다. **Motor[x].Ctrl** 이 **Sys.ServoCtrl**, **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.MaxInt 모터 서보의 최대 적분 편차

설명: 서보의 최대 적분 편차의 크기

범위: 음이 아닌 부동 소수점

단위: 적분기의 출력 단위(16 비트 출력의 LSB)

기본값: 28,000

기존 I 변수명: lxx63

Motor[x].Servo.MaxInt 는 PID 서보 알고리즘의 적분 위치 편차(적분기의 출력)의 크기를 제한하고 서보 루프 출력이 포화되는 경우의 "안티 와인드업" 보호에 유익합니다. **Motor[x].Servo.MaxInt** 의 기본값에서는 기본적으로는 제한은 없습니다(적분 계인의 **Motor[x].Servo.Ki** 에서는 편차가 어느 정도 빠르게 적분되는지 제어합니다).

Motor[x].Servo.MaxInt 의 제로값은 적분기의 출력을 강제로 제로로 하고 PID 필터의 적분 기능을 실질적으로 무효로 합니다. 이것은 일정한 힘을 계속 적용하고 정상 상태의 위치 편차를 필요로 하는 기간에 유익합니다(대조적으로 **Motor[x].Servo.Ki** 를 0 으로 설정하면 적분기로의 그 이상의 입력을 방지하지만 그 출력은 유지됩니다).

Turbo PMAC 서보 알고리즘과 등가인 동작을 위해서는 이 항은 다음과 같이 설정할 수 있습니다.

$$\text{Motor}[x].\text{Servo.MaxInt} = I_{xx63} * \frac{I_{xx08} * I_{xx30}}{2^{23}}$$

Motor[x].Ctrl 가 **Sys.PidCtrl**, **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.MaxInt** 는 사용됩니다.

Motor[x].Servo.MaxPosErr 모터 서보의 최대 위치 편차

설명: 서보 편차의 입력 한계값

범위: 음이 아닌 부동 소수점

단위: 모터 단위

기본값: 10,000

기존 I 변수명: Ixx67

Motor[x].Servo.MaxPosErr 에서는 서보 필터에서 허용되는 최대 위치 편차의 크기를 지정합니다. 이 목적은 극단적인 상태가 필터의 안정성을 해치는 것을 방지하는 것입니다. 그러나 이 값이 너무 낮은 경우에 본래의 명령 궤적 또는 외란에 대한 시스템의 응답을 제한합니다(이 상태는 매우 고분해능인 시스템인 경우에 알게 됩니다).

순수한 속도 제어가 모터에 필요한 경우 **Motor[x].Servo.MaxPosErr** 를 0 으로 설정할 수 있고 실질적으로 위치 루프를 무효로 합니다.

이 파라미터는 **Motor[x].FatalFeLimit** 또는 **Motor[x].WarnFeLimit**(편차 리미트 파라미터)와 혼동하지 마십시오. 그러한 파라미터는 실제 편차(제한하기 전의)에 따라서 서보 루프의 바깥쪽에서 기능하기 때문입니다.

Motor[x].Ctrl 이 **Sys.PidCtrl**, **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 에 설정되는 경우에 **Motor[x].Servo.MaxPosErr** 가 사용됩니다.

Motor[x].Servo.MaxW 모터 서보의 최대 게인 고유 주파수

설명:	최대 장치 게인에서의 적응 서보 클로즈드 루프 고유 주파수
범위:	음이 아닌 부동 소수점
단위:	라디안/sec
기본값:	0.0(무효)

Motor[x].Servo.MaxW 는 모터에 생성하기 위해 알고리즘이 시도해야 하는 클로즈드 루프 고유 주파수(ω)를 지정합니다. 이것은 적응 서보 알고리즘으로 추정된 장치 게인이 최대값(최소 관성을 의미)이며 알고리즘이 보정할(**Motor[x].Servo.MaxGainFactor** 의 설정) 때의 값입니다. 단, **Motor[x].Servo.MinW** 가 기본값 0.0 인 경우에 적응 알고리즘에서는 클로즈드 루프 고유 주파수(또는 감쇠율) 조정은 수행하지 않습니다. 그 대신 공칭값에서의 그 값을 일정하게 유지합니다.

Motor[x].Servo.MinW 가 0.0 보다 큰 값이고 고유 주파수 자동 조정을 유효로 하는 경우 **Motor[x].Servo.MaxW**, **Motor[x].Servo.MinDR** 그리고 **Motor[x].MaxDR** 에서는 고유 주파수와 감쇠율의 변화 범위를 정의하기 위해 최적인 설정을 수행해야 합니다. 해당 항목은 보통 IDE 의 조정 제어의 대화형 창에 의해 설정됩니다.



경고

Motor[x].Servo.MinDR 와 **Motor[x].Servo.MinW** 가 0.0 보다 큰 값으로 설정되어 게인 스케줄의 적응 제어가 유효한 경우에 **Motor[x].Servo.MaxW** 는 기본값 0.0 그대로 두면 안 됩니다. 이것은 최소 시스템 관성에서의 제어 효과를 지정하지 않는 것입니다. 이것은 시스템의 위험한 상태로 이어질 수도 있습니다.

자동적인 고유 주파수 조정이 유효한 경우에 적응 서보 알고리즘은 추정된 장치 게인의 함수로 그 내부 게인을 설정하고 최대 장치 게인(최소 관성)에서의 **MaxW** 의 값과 최소의 장치 게인(최대 관성)에서의 **MinW** 의 값 사이에서 클로즈드 루프 고유 주파수를 직선적으로 변화시킵니다. 자동적인 고유 주파수 조정은 관성 변화에서의 일정한 성능을 유지하고자 하는 요구와 시스템의 물리적 제한 간의 트레이드 오프의 관계가 됩니다.



주의

Motor[x].Servo.MaxW 의 단위는 라디안/sec 입니다. 라디안/sec 로 표시된 주파수에서는 헤르츠(사이클/sec)로 표시된 동일 주파수보다 2π 만큼 큰 계수의 값이 됩니다.

Motor[x].Servo.MaxW 는 V1.6 펌웨어(2014 년 1/4 분기 출시)의 새로운 기능입니다. **Motor[x].Servo.MinW** 가 기본값 0.0 인 경우에 동작은 이전 펌웨어 버전과 호환성이 있고 적응 서보 알고리즘의 고유 주파수의 자동 조정을 지원하지 않습니다.

Motor[x].Ctrl 이 **Sys.AdaptiveCtrl** 또는 **Sys.GantryXCtrl** 로 설정되는 경우 **Motor[x].Servo.MaxW** 는 사용됩니다. **Motor[x].Ctrl** 이 **Sys.ServoCtrl**, **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

**주의**

고유 주파수의 조정 알고리즘은 초와 서보 사이클 간의 시간 단위를 변환하기 위해 **Sys.ServoPeriod** 의 값을 사용합니다. 따라서 **Sys.ServoPeriod** 의 조정 계산을 위해 정확해야 합니다.

Motor[x].Servo.MinDR 모터 서보의 최소 게인 감쇠율

설명:	최소 장치 게인에서의 적응 서보 클로즈드 루프의 감쇠율
범위:	음이 아닌 부동 소수점
단위:	없음
기본값:	0.0(무효)

Motor[x].Servo.MinDR 는 모터에 생성하기 위해 알고리즘이 시도해야 하는 클로즈드 루프 감쇠율(ζ)을 지정합니다. 이것은 적응 서보 알고리즘으로 추정된 장치 게인이 최소값(최대 관성을 의미)이며 알고리즘이 보정할(**Motor[x].Servo.MinGainFactor** 의 설정) 때의 값입니다. 단, 이것이 기본값 0.0 인 경우에 적응 알고리즘에서는 클로즈드 루프 감쇠율(또는 고유 주파수) 조정은 수행하지 않습니다. 그 대신 공칭값에서 그 값을 일정하게 유지합니다.

Motor[x].Servo.MinDR 가 0.0 보다 큰 값이고 감쇠율 자동 조정을 유효로 하는 경우 **Motor[x].Servo.MaxDR**, **Motor[x].Servo.MinW** 그리고 **Motor[x].MaxW** 에서는 고유 주파수와 감쇠율의 변화 범위를 정의하기 위해 최적인 설정을 수행해야 합니다. 해당 항은 보통 IDE 의 조정 제어의 대화형 창에 의해 설정됩니다.

자동적인 감쇠율 조정이 유효한 경우에 적응 서보 알고리즘은 추정된 장치 게인의 함수로 그 내부 게인을 설정하고 최대 장치 게인(최소 관성)에서의 **MaxDR** 의 값과 최소의 장치 게인(최대 관성)에서의 **MinDR** 의 값 사이에서 클로즈드 루프 감쇠율을 직선적으로 변화시킵니다.

Motor[x].Servo.MinDR 는 V1.6 펌웨어(2014 년 1/4 분기 출시)의 새로운 기능입니다. **Motor[x].Servo.MinDR** 가 기본값 0.0 인 경우에 동작은 이전 펌웨어 버전과 호환성이 있고 적응 서보 알고리즘의 감쇠율 자동 조정을 지원하지 않습니다.

Motor[x].Ctrl 이 **Sys.AdaptiveCtrl** 또는 **Sys.GantryXCtrl** 로 설정되는 경우 **Motor[x].Servo.MinDR** 는 사용됩니다. **Motor[x].Ctrl** 이 **Sys.ServoCtrl**, **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.MinGainFactor 모터 서보의 최소 게인 팩터

설명:	최소 허용 상대 게인
범위:	음이 아닌 부동 소수점
단위:	없음(비율)
기본값:	1.0

Motor[x].Servo.MinGainFactor에서는 적응 제어 알고리즘에서 사용되는 최소 상대 서보 게인을 지정합니다. 그리고 표준 저장 서보 게인과의 비로 표시됩니다. 이것은 추정되는 최저 시스템 관성(최고 장치 게인)에서의 변화를 제한합니다. 이것은 적응 서보 제어 알고리즘이 선택된 경우에만 사용됩니다. 예를 들어 그것이 0.25로 설정되는 경우에 결과로 발생하는 게인은 표준 저장 서보 게인의 1/4를 하회하지 않습니다.

Motor[x].Servo.MinGainFactor의 표준값은 0.1~1.0의 범위가 됩니다. 이 파라미터는 적응 제어 알고리즘에서의 잘못된 결과를 방지하는 중요한 보호 방법입니다.

Motor[x].Ctrl이 **Sys.AdaptiveCtrl** 또는 **Sys.GantryXCtrl**(2014년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능)로 설정되는 경우에 **Motor[x].Servo.MinGainFactor**는 사용됩니다. **Motor[x].Ctrl**이 **Sys.ServoCtrl**, **Sys.PidCtrl** 또는 **Sys.PosCtrl**로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.MinW 모터 서보의 최소 게인 고유 주파수

설명:	최소 장치 게인에서의 적응 서보 클로즈드 루프 고유 주파수
범위:	음이 아닌 부동 소수점
단위:	라디안/sec
기본값:	0.0(무효)

Motor[x].Servo.MinW는 모터에 생성하기 위해 알고리즘이 시도해야 하는 클로즈드 루프 고유 주파수(ω)를 지정합니다. 이것은 적응 서보 알고리즘으로 추정된 장치 게인이 최소값(최대 관성을 의미)이며 알고리즘이 보정할(**Motor[x].Servo.MinGainFactor**의 설정) 때의 값입니다. 단, 이 값이 기본값 0.0인 경우에 적응 알고리즘은 클로즈드 루프 고유 주파수(또는 감쇠율) 조정을 수행하지 않습니다. 그 대신 공칭값에서 그 값을 일정하게 유지합니다.

Motor[x].Servo.MinW가 0.0보다 큰 값이고 고유 주파수 자동 조정을 유효로 하는 경우 **Motor[x].Servo.MaxW**, **Motor[x].Servo.MinDR** 그리고 **Motor[x].MaxDR**에서는 고유 주파수와 감쇠율의 변화 범위를 정의하기 위해 최적의 설정을 수행해야 합니다. 해당 항목은 보통 IDE의 조정 제어의 대화형 창에 의해 설정됩니다.

자동적인 고유 주파수 조정이 유효한 경우에 적응 서보 알고리즘은 추정된 장치 게인의 함수로 그 내부 게인을 설정하고 최대 장치 게인(최소 관성)에서의 **MaxW** 의 값과 최소의 장치 게인(최대 관성)에서의 **MinW** 의 값 사이에서 클로즈드 루프 고유 주파수를 직선적으로 변화시킵니다. 자동적인 고유 주파수 조정은 관성 변화에서의 일정한 성능을 유지하고자 하는 요구와 시스템의 물리적 제한 간의 트레이드 오프의 관계가 됩니다.



Motor[x].Servo.MinW 의 단위는 라디안/sec 입니다. 라디안/sec 로 표시된 주파수는 헤르츠(사이클/sec)로 표시된 동일 주파수보다 2π 만큼 큰 계수의 값이 됩니다.

Motor[x].Servo.MinW 는 V1.6 펌웨어(2014 년 1/4 분기 출시)의 새로운 기능입니다. 이 값이 기본값 0.0 인 경우에 동작은 이전 펌웨어 버전과 호환성이 있고 적응 서보 알고리즘의 고유 주파수의 자동 조정을 지원하지 않습니다.

Motor[x].Ctrl 이 **Sys.AdaptiveCtrl** 또는 **Sys.GantryXCtrl** 로 설정되는 경우 **Motor[x].Servo.MinW** 는 사용됩니다. **Motor[x].Ctrl** 이 **Sys.ServoCtrl**, **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.



고유 주파수의 조정 알고리즘은 초와 서보 사이클 간의 시간 단위를 변환하기 위해 **Sys.ServoPeriod** 의 값을 사용합니다. 따라서 **Sys.ServoPeriod** 의 조정 계산을 위해 정확해야 합니다.

Motor[x].Servo.NominalGain 모터 서보의 공칭 게인

설명: 플랜트(장치) 기준 게인
범위: 음이 아닌 부동 소수점
단위: (모터 단위/sec²)/16 비트 명령 단위
기본값: 0.0

Motor[x].Servo.NominalGain 은 적응 서보 제어 알고리즘에서 서보 루프 출력을 명령할 때의 물리적인 장치의 기준 "게인"을 지정합니다. 이것은 적응 서보 제어 알고리즘이 선택된 경우에만 사용됩니다. 이 게인은 결과로 발생하는 모터 가속도(모터 단위/sec²)를 16 비트 단위의 서보 루프 출력 명령(+/- 32,768)으로 나눈 값으로 표시됩니다. 이 값은 모터와 부하 관성 게인뿐만 아니라 서보 명령 신호 게인, 앰프 게인 그리고 피드백 센서와 그 스케일링 게인도 포함합니다.

이 전체 게인의 성분 중 일반적으로는 관성값만 변화합니다(관성에 반비례하는 관성 "게인"). 가능한 관성의 범위 내에서 어떤 특정 관성값이 사용됐는지는 중요하지 않지만 제한항인 **Motor[x].Servo.MinGainFactor** 와 **Motor[x].Servo.MaxGainFactor** 는 값의 선택에는 필요합니다. 그리고 표준 저장 서보 루프 게인항은 이 장치 게인값에서의 필요한 성능에 대해 설정되어야 합니다. 실효 서보 루프 게인은 이 값과 **Motor[x].Servo.EstGain** 에서의 최신의 추정된 장치 게인 간의 비에 따릅니다.

Motor[x].Servo.NominalGain 의 값은 분석적으로 결정할 수 있지만 보통 실험적으로 결정하는 편이 더 간단하고 더 정확합니다. IDE 의 서보 조정 제어의 "위치 루프 자동 조정" 기능은 "추정 게인"으로 그 진동 발생에 따라서 장치의 계산값을 통지합니다. 이 통지된 값은 **Motor[x].Servo.NominalGain** 에 직접 사용할 수 있습니다.

Motor[x].Ctrl 이 **Sys.AdaptiveCtrl** 또는 **Sys.GantryXCtrl**(2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능)로 설정되는 경우에 **Motor[x].Servo.NominalGain** 은 사용됩니다. **Motor[x].Ctrl** 이 **Sys.ServoCtrl**, **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.OutDbOff 모터 서보의 출력 불감대 OFF

설명: 서보의 출력 불감대 OFF 의 편차 크기

범위: 음이 아닌 부동 소수점

단위: 모터 단위

기본값: 0.0

Motor[x].Servo.OutDbOff 에서는 제로 편차에서 양쪽 방향에서 측정되는 위치 편차 영역의 크기를 정의하고 여기서는 위치 편차의 크기가 이 크기보다 커지면 불감대 출력은 무효화됩니다. 이 영역 밖에서는 서보 출력은 보통 제로 이외가 됩니다.

출력 불감대에는 히스테리시스가 있고 따라서 한번 이 영역 밖으로 나가면 다시 출력 불감대(제로 서보 출력)를 생성하기 위해서는 위치 편차의 크기가 그 후에 **Motor[x].Servo.OutDbOn**(보통 더 작은 값)보다 작아져야 합니다.

Motor[x].Ctrl 이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.OutDbOff** 는 사용됩니다. **Motor[x].Ctrl** 이 **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.OutDbOn 모터 서보의 출력 불감대 ON

설명: 서보의 출력 불감대 ON 의 편차 크기

범위: 음이 아닌 부동 소수점

단위: 모터 단위

기본값: 0.0

Motor[x].Servo.OutDbOn에서는 제로 편차에서 양쪽 방향에서 측정되는 위치 편차 영역의 크기를 정의하고 여기서는 위치 편차의 크기가 이 크기보다 작아지면 출력 불감대가 유효가 됩니다. 이 영역 밖에서는 서보 출력은 정확하게 제로가 됩니다.

출력 불감대에는 히스테리시스가 있고 따라서 한번 이 영역 안으로 들어오면 다시 제로 이외의 서보 출력을 유효로 하기 위해서는 위치 편차의 크기가 그 후에 **Motor[x].Servo.OutDbOff**(보통 더 큰 값)보다 커져야 합니다.

Motor[x].Ctrl이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl**로 설정되는 경우에 **Motor[x].Servo.OutDbOn**은 사용됩니다. **Motor[x].Ctrl**이 **Sys.PidCtrl** 또는 **Sys.PosCtrl**로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.OutDbSeed 모터 서보의 출력 불감대 시드

설명: 서보의 출력 불감대 적분기 시드

범위: 음이 아닌 부동 소수점

단위: 16 비트 출력의 LSB

기본값: 0.0

Motor[x].Servo.OutDbSeed에서는 위치 편차가 **Motor[x].Servo.OutDbOff**의 최소치를 초과하고 서보 제어가 다시 유효화될 때 사용되는 적분기의 "시드(선택)"의 크기를 정의합니다. 즉 이 크기(와 적합한 부호)의 값은 이 시점에서 적분기의 출력 레지스터에 더하고 거기에 어떤 값이 쓰여 있어도 덮어씁니다. 이 값은 최종 위치로의 정정을 향상하기 위해 사용할 수 있습니다.

Motor[x].Ctrl이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl**로 설정되는 경우에 **Motor[x].Servo.OutDbSeed**는 사용됩니다. **Motor[x].Ctrl**이 **Sys.PidCtrl** 또는 **Sys.PosCtrl**로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.SwFffInt 모터 서보의 마찰 Feed-forward 선택

설명: 마찰 Feed-forward의 명령 대상을 선택하는 소프트웨어 스위치

범위: 0..1

단위: 논리값

기본값: 0

Motor[x].Servo.SwFffInt항에서는 서보 루프의 "마찰 Feed-forward"항의 출력 대상을 선택합니다. **Servo.SwFffInt**가 기본값 0인 경우에 마찰 Feed-forward 명령은 서보 루프의 출력에 직접 더해집니다. **Servo.SwFffInt**가 1로 설정되는 경우에 마찰 Feed-forward 명령은 서보 루프의 적분기 입력에 더해집니다.

마찰 Feed-forward, 속도 Feed-forward 그리고 속도 피드백항이 서보 출력에 직접 더해지는 경우에 적분기는 실질적으로 위치 루프에 존재하고 보통 명령 궤적의 추종에 더 효과가 있습니다. 마찰 Feed-forward, 속도 Feed-forward 그리고 속도 피드백항이 적분기의 입력에 더해지는 경우에 적분기는 실질적으로 속도 루프에 존재하고 보통 외란 제거에 더 효과가 있습니다.

Motor[x].Ctrl 이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.SwFfInt** 는 사용됩니다. **Motor[x].Ctrl** 이 **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.SwPoly7 모터 서보의 7 차 다항식 선택

설명: 2 차 또는 7 차 다항식을 선택하는 소프트웨어 스위치

범위: 0 ..1

단위: 논리값

기본값: 0

Motor[x].Servo.SwPoly7 항은 표준 서보 알고리즘의 다항식 블록 "A", "B", "C"와 "D"가 2 차 또는 7 차식인지 선택합니다. 기본값 0 인 경우에 2 차식으로 실행되고 최초의 2 항만 사용됩니다(예를 들어 그밖에 0 이외의 값으로 설정된 경우). 1 로 설정되는 경우 7 차식으로 실행됩니다.

보통 **Motor[x].Servo.SwPoly7** 은 고차 다항식의 계산을 필요로 하는 곤란한 동특성(예를 들어 다중 공진)의 모터를 제어하는 경우에만 1 로 설정됩니다. 대부분의 시스템에서 이 변수는 계산 시간을 절약하기 위해 기본값인 0 그대로 둡니다.

Motor[x].Ctrl 이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl** 로 설정되는 경우에 **Motor[x].Servo.SwPoly7** 는 사용됩니다. **Motor[x].Ctrl** 이 **Sys.PidCtrl** 또는 **Sys.PosCtrl** 로 설정되는 경우는 사용되지 않습니다.

Motor[x].Servo.SwZvInt 모터 서보의 적분 모드 선택

설명: 적분 모드의 선택 소프트웨어 스위치

범위: 0 ..1

단위: 논리값

기본값: 0

기존 I 변수명: Ixx34

Motor[x].Servo.SwZvInt 항에서는 위치 편차 적분항(**Ki**)이 모든 시간에 ON 이 되거나 모터의 명령 속도가 0 일 때만 ON 이 되는지 선택합니다. 0 으로 설정되는 경우에 위치 오차 적분은 모든 시간에 실행됩니다.

1로 설정되는 경우에 명령 속도가 정확하게 0일 때만 위치 편차 적분이 실행됩니다. 명령 속도가 반드시 0과 같지 않을 때(즉 명령 동작 중) 적분기로의 입력은 정지되고 적분기에서의 출력 제어 효과가 이 기간 일정하게 유지되는 것을 의미합니다(보통 0이 아님). 합계의 제어 출력이 **Motor.MaxDac**의 값에서 포화될 때는 언제라도 이 동일한 동작이 실행됩니다.

Motor[x].Servo.SwZvInt는 통합 개발 환경(IDE) 프로그램에서 최초로 조정 유틸리티를 사용할 때 보통 설정됩니다. 그 유틸리티의 대화형 Feed-forward 조정을 실행하는 경우는 시스템의 동특성을 적분 동작 없이 관찰할 수 있도록 1로 설정하는 것이 중요합니다.

Motor[x].Ctrl이 **Sys.ServoCtrl**, **Sys.GantryXCtrl** 또는 **Sys.AdaptiveCtrl**로 설정되는 경우에 **Motor[x].Servo.SwZvInt**는 사용됩니다. **Motor[x].Ctrl**이 **Sys.PidCtrl** 또는 **Sys.PosCtrl**로 설정되는 경우는 사용되지 않습니다.

Sys.저장 데이터 구조체 요소(V1.6 2014.03)

Sys. 데이터 구조체는 Power PMAC 의 글로벌 설정으로 구성됩니다. 이 구조체 내의 요소는 전체 Power PMAC 에 작용하며 특정 모터, 좌표 시스템 또는 ASIC 에 제한되지 않습니다.

Turbo PMAC 의 글로벌 설정은 I 변수의 I0-I99 로 관리되었습니다. I 변수의 해당 범위는 Power PMAC 에서는 모터 0 에 할당된다는 점에 주의하십시오. 해당 요소를 "계승한" Power PMAC 의 I 변수 번호는 Turbo PMAC 의 해당 번호보다 8000 이 큰 번호입니다.

Sys.AbortAllBit 시스템의 글로벌 Abort 입력 비트 번호

설명: **pAbortAll** 레지스터의 글로벌 Abort 입력 비트 번호

범위: 0 .. 31

단위: 비트 수(리틀 엔디언)

기본값: Power Brick:31

다른 Power PMAC:0

Sys.AbortAllBit 는 Power PMAC 이 글로벌 Abort 입력 검출에 사용하는 비트가 **Sys.pAbortAll** 로 지정되는 32 비트 레지스터의 어느 비트인지 결정합니다. 비트 번호는 "리틀 엔디언" 규약으로 지정됩니다. 그때 비트 n 은 32 비트의 워드 중의 2^n 의 값입니다. **Sys.pAbortAll** 이 0 에서 글로벌 입력 기능이 무효가 된 경우에 **Sys.AbortAllBit** 의 설정은 참조되지 않습니다.

지정된 비트가 **Sys.AbortAllLimit** 의 누적 스캔 수 사이 "1" 상태(극성 제어는 제공되지 않습니다)로 검출된 경우에 좌표계는 각각 해당 저장 설정 요소 **Coord[x].AbortAllMode** 의 설정에 따라서 동작하고 서보 루프 제어되는 정지인지, 제어되지 않은 정지인지 지정할 수 있으며 또는 입력을 무시할 수 있습니다.

Sys.AbortAllLimit 시스템의 글로벌 Abort 검출 최대 누적 수

설명: 실제 글로벌 Abort 검출의 최대 누적 수

범위: 0 .. 255

단위: 스캔

기본값: 0

Sys.AbortAllLimit 는 Power PMAC 이 글로벌 Abort 입력을 "Abort" 상태라고 검출해도 글로벌 Abort 명령을 전송하지 않고 누적되는 스캔의 최대 수를 지정합니다. Abort 상태에서 누적되는 스캔 수가 이 값을 초과하면 좌표계는 각각 해당 저장 설정 요소 **Coord[x].AbortAllMode** 의 설정에 따라서 동작합니다. 그것은 서보 루프 제어되는 정지인지 제어되지 않은 정지인지 지정할 수 있고 또는 입력을 무시할 수 있습니다. Power PMAC 은 실시간 인터럽트 주기로 지정된 글로벌 Abort 비트의 상태를 확인합니다.

글로벌 Abort 입력 검출 회로는 일시적인 전기적 노이즈 또는 동일한 요인에 의한 이상도 감지하게 됩니다. Abort 상태를 연속해서 여러 번 검출하는 스캔을 요구하여 실제 Abort 입력을 확정하는 것으로 오검출에 의한 트립의 제거에 효과가 있습니다.

글로벌 Abort 비트의 레지스터는 **Sys.pAbortAll** 로 지정됩니다. 이 레지스터 내의 비트의 입력 검출의 사용 상태를 **Sys.AbortAllBit** 로 지정합니다. **Sys.pAbortAll** 이 0 으로 설정되고 글로벌 Abort 검출 기능을 무효인 경우에 **Sys.AbortAllLimit** 의 설정은 관계없습니다. 지정된 비트로 "Abort" 상태를 검출한 스캔의 누적된 수는 상태 요소 **Sys.AbortAllCount** 로 조사됩니다. 이 값은 Abort 상태가 검출될 때마다 더하고 Abort 상태가 검출되지 않을 때마다(0 보다 큰 경우) 뺍니다.

Sys.BgSleepTime 시스템의 백그라운드 사이클 휴지 시간

설명: 백그라운드 사이클 휴지 시간

범위: 0, 250 ..10,000

단위: μsec

기본값: 0(1,000 μsec 를 지정)

Sys.BgSleepTime 는 Power PMAC 어플리케이션이 백그라운드 태스크를 선별하여 실행하도록 스케줄이 조정된 연속하는 Power PMAC 백그라운드 사이클 간의 "슬립" 기간을 지정합니다. 이 기간은 범용 운영체제(GPOS)가 GPOS 어플리케이션을 실행할 수 있게 합니다. Power PMAC 인터럽트 태스크가 동일하게 이 기간에 실행합니다.

스케줄 관리된 1 회의 백그라운드 사이클에서 Power PMAC 은 하나의 유효한 백그라운드 스크립트 PLC 프로그램을 1 회 스캔하고 모든 유효한 백그라운드 C PLC 프로그램을 1 회 스캔한 후 "하우스 키팅" 상태의 갱신과 안전성 확인을 실행합니다.

Sys.BgSleepTime 이 기본값 0 인 경우에 1,000 μsec (1msec)의 값이 사용되고 이 값은 초기 펌웨어 버전의 고정값에 해당합니다. 이 요소의 유효 범위는 250~10,000 μsec (0.25~10msec)입니다. 명령에서 250 μsec 미만으로 설정해도 250 μsec 로 설정됩니다. 또한 명령에서 10,000 μsec 보다 크게 설정해도 10,000 μsec 로 설정됩니다.

이 요소에 의해 스케줄 관리된 백그라운드 사이클 태스크와 독립한 C 어플리케이션을 포함하는 GPOS 태스크에 할당되는 백그라운드 시간의 합계를 결정할 때의 어플리케이션에 유연성을 부여합니다.

Sys.BgWDTReset 시스템의 백그라운드 워치독 타이머 최소치

설명: 백그라운드 워치독 타이머 최소치

범위: 0 .. $2^{32}-1$

단위: 백그라운드 소프트웨어 사이클

기본값: 0(10 사이클을 지정)

Sys.BgWDTReset 는 Power PMAC 이 "소프트웨어" 워치독 타이머 이상을 발생시키지 않고 "실시간 인터럽트"(RTI) 없이 실행할 수 있는 백그라운드 최대 사이클 수를 지정합니다. 기본값 0 인 경우에 10 백그라운드 사이클값이 사용됩니다. 이 모니터링 기능은 안전한 제어를 방해하는 하드웨어 또는 소프트웨어 인터럽트의 문제와 과부하를 검출하는 것을 목적으로 켜둔 메커니즘을 제공합니다.

RTI 는 (**Sys.RtIntPeriod** + 1)서보 주기마다 발생하는 "소프트웨어 인터럽트"입니다. 다음 RTI 가 발생하기 전에 RTI 에서 시작하는 1 개의 소프트웨어 태스크가 종료되지 않은 경우에 다음 RTI 는 "건너뛰니다".

유효한 기본값인 10 백그라운드 사이클은 대다수의 Power PMAC 어플리케이션에서 적절합니다. 단, 특수한 타이밍 요건을 가지는 어플리케이션에서는 다른 설정이 필요할 수 있습니다.

Sys.BgWDTReset 이 너무 큰 값인 경우에 Power PMAC 을 완전히 셧다운하는 "하드" 워치독 트립이 발생하기 전의 인터럽트 태스크 문제를 파악하지 못할 수도 있습니다.

소프트웨어 워치독 타이머의 트립에서는 모든 모션 프로그램이 **Abort** 되고 모든 모터는 소프트웨어에서 **kill** 됩니다(오픈 루프, 제로 출력, 앰프 무효). 그리고 인터페이스 ASIC 는 모두 "리셋" 상태로 고정되어 앰프 유효를 포함하는 개별 출력을 모두 중지하고 모든 연속 가변 명령 출력(DAC, PWM, PFM)은 제로 명령값으로 강제로 출력됩니다. 단, Power PMAC 프로세서는 유효한 상태 그대로 PLC 프로그램을 실행할 수 있고 호스트 컴퓨터와 통신할 수 있습니다.

임의의 시점에서의 최신의 RTI 에서 실행된 백그라운드 사이클의 수가 읽기 전용 요소 **Sys.BgWdTimer** 에 저장됩니다. 이 타입의 이상이 발생하면 읽기 전용 요소 **Sys.WDTFault** 는 2 로 설정됩니다.

Sys.BusCtrl[n] 시스템의 I/O 버스 타이밍 제어

설명: I/O 버스 타이밍 제어

범위: \$0000 .. \$FFFF

단위: 비트 필드

기본값: \$6666 (n = 0.. 9, 14.. 15)
\$6747 (n = 10.. 13)

Sys.BusCtrl[n]은 Power PMAC 상의 메모리- 매핑 I/O 의 읽기와 쓰기 동작의 타이밍 특성을 지정합니다. I/O 액세스에 사용 가능한 16 의 "칩 선택" 라인($n=0\sim15$)의 각각에 별도의 요소가 있습니다. 모든 칩 선택 라인이 무엇을 액세스하며 또 해당 인덱스 n 을 나타내는 표는 "소프트웨어 참조"의 "I/O 주소 오프셋" 장에 나와 있습니다. 인덱스 n 의 값은 칩 선택 라인을 사용하는 I/O 의 (**sys.piom** 의 I/O 기준 주소에서의)주소 오프셋의 처음 16 진수 자리에 해당합니다.

다음 표는 현재 지원되는 I/O 의 각 **Sys.BusCtrl[n]** 요소에서 제어되는 액세스 타이밍을 정리한 것입니다.

I/O 타입	BusCtrl[n] 인덱스	기준 주소 오프셋	I/O 타입	BusCtrl[n] 인덱스	기준 주소 오프셋
PMAC2 스타일 서보 IC	6	\$600000	범용 I/O	11	\$B00000
PMAC2 스타일 서보 IC	7	\$700000	범용 I/O	12	\$C00000
PMAC2 스타일 MACRO IC	8	\$800000	범용 I/O	13	\$D00000
PMAC3 스타일 IC	9	\$900000	공유 메모리	14	\$E00000
범용 I/O	10	\$A00000	공유 메모리	15	\$F00000

Sys.BusCtrl[n]은 칩 선택 라인의 쓰기 동작을 관리하는 하위 8 비트(마지막 16 진수 2 자리)와 읽기 동작을 관리하는 상위 8 비트(처음 16 진수 2 자리)의 16 비트의 값입니다.

각 8 비트 필드의 하위 5 비트(비트 0-4)는 칩 선택의 읽기 또는 쓰기 라인에서 부가되는 대기 상태 수(\$00~\$1F, 10 진수에서 0~31)를 지정합니다. 0 대기에서는 읽기 또는 쓰기 라인은 40nsec 동안 low 가 되고 칩 선택 라인은 읽기 또는 쓰기 펄스 시작 후에 50nsec 동안 low 로 유지됩니다. 각 대기 상태는 칩 선택과 읽기 또는 쓰기 라인의 양쪽 low 기간의 마지막에 10nsec 가 부가됩니다. 대기 상태가 6 개 부가되는 공장 출하 값에서는 읽기 또는 쓰기 라인은 100nsec 동안 low 가 되고 칩 선택 라인은 읽기 또는 쓰기 펄스 시작 후에 110nsec 동안 low 로 유지됩니다.

각 8 비트 필드의 비트 5 와 6 은 칩 선택 라인이 low 가 된 후에 읽기 또는 쓰기 라인이 low 가 될 때의 지연을 제어합니다. 해당 비트가 0 의 값에서 30-nsec 가 지연됩니다. 1 의 값에서 20-nsec 가 지연됩니다. 2 의 값에서 10-nsec 가 지연됩니다. 또한 3(기본값)의 값에서는 지연은 없습니다.

쓰기 필드의 비트 7 이 1 인 경우에 쓰기 신호는 이 비트가 0 으로 설정되는 경우보다 20nsec 빠르게 high 로 복귀합니다. 칩 선택 라인 펄스의 길이에 영향을 주지 않습니다. 읽기 필드의 비트 7 은 추후 사용을 위해 예약됩니다.

**주의**

Delta Tau 하드웨어용으로 사용된 **Sys.BusCtrl[n]** 요소의 값은 Delta Tau 에서의 명확한 지시가 없으면 공장 출하 설정에서 변경할 필요는 없습니다. 이러한 요소에는 특수한 상황 및 사용자 정의 하드웨어에 대한 사용자 액세스 권한이 있습니다. 대부분의 상황에서 동작하지만 신뢰성 있는 동작에 충분히 안전한 여유가 없는 "경계선"을 설정하면 간헐적인 오작동을 일으키며 이것은 대단히 위험한 결과를 초래할 수 있으니 주의하십시오.

Sys.CamEnable 시스템의 유효 캠 테이블 수

설명: 유효 캠 테이블 수

범위: 0 ..255

단위: 테이블

전원 셋업 기본값: 0

Sys.CamEnable 은 유효가 되는 최대 캠 테이블 수를 지정합니다. 0 보다 큰 경우 인덱스가 0에서 (**Sys.CamEnable** - 1)까지의 캠 테이블이 서보 사이클마다 확인됩니다. 개별 테이블인 **CamTable[m].Enable** 이 0 보다 큰 경우에 유효 테이블은 서보 사이클로 실행됩니다. 개별 테이블인 **CamTable[m].Enable** 이 0 인 경우에 해당 테이블은 정상적으로 다시 유효가 되도록 대기됩니다. **CamTable[m].Enable** 의 값은 저장되지 않아 전원 셋업/리셋에서 항상 0 으로 설정됩니다.

Sys.CompEnable 시스템의 유효 보정 테이블 수

설명: 유효 보정 테이블 수

범위: 0 ..256

단위: 테이블

기본값: 0

기존 I 변수명: I8051(Turbo PMAC 의 I51)

Sys.CompEnable 은 (존재하는 경우)유효로 하는 보정 테이블의 수를 지정합니다. n 의 값으로 설정되는 경우에 **CompTable[0]**에서 **CompTable[n-1]**까지의 보정 테이블이 유효가 되며 이 번호보다 높은 번호에 할당된 모든 보정 테이블은 유효가 아닙니다. 보정 테이블이 유효가 될 때 Power PMAC 은 서보 인터럽트마다 보정을 계산합니다.

Sys.CompMotor 시스템의 보정 테이블 갱신 전의 모터 수

설명:	보정 테이블이 갱신되기 전의 모터 수
범위:	0 ..255
단위:	모터 수
기본값:	0

Sys.CompMotor 은 유효한 보정 테이블이 어느 서보 사이클에서 계산될지 지정합니다. 모든 유효한 보정 테이블의 보정은 **Sys.CompMotor** 로 지정되는 모터의 서보 루프가 닫히기 직전에 계산됩니다.

대부분의 어플리케이션은 **Sys.CompMotor** 를 기본값 0 그대로 둡니다. 따라서 어떤 모터의 서보 루프가 닫히기 전에 보정 테이블이 계산됩니다(단, 모든 모터의 명령 궤적이 갱신된 후에 보정은 이 사이클의 명령 위치에 따라서 계산됩니다).

단, 캐스케이드 서보 루프에 보정 테이블을 이용하는 어플리케이션은 아우터 루프 모터의 서보 루프는 닫고 이너 루프 모터의 서보 루프는 닫기 전에 보정 테이블이 실행되도록 **Sys.CompMotor** 를 설정할 수 있습니다. 이것은 외측 루프에서 내측 루프로의 명령 위치를 얻을 때 서보 사이클의 지연을 제거합니다. 사용자는 모터 명령 위치에 기초한 보정 테이블의 표준 적용법에서는 **Sys.CompMotor** 미만으로 번호가 부여된 모터에 의한 지연이 서보 사이클에 추가된다는 것을 이해하십시오.

Sys.CpuTimerIntr 시스템의 프로세서 내부 타이머 인터럽트 유효

설명:	프로세서 내부 타이머 인터럽트 유효
범위:	0 ..1
단위:	논리값
기본값:	0(클록 생성 IC 를 검출) 1(클록 생성 IC 는 검출 안 함)

Sys.CpuTimerIntr 는 Power PMAC 프로세서가 페이즈와 서보의 인터럽트에 외부 신호를 사용할지 내부 타이머가 생성한 인터럽트를 사용할지 여부를 제어합니다.

Sys.CpuTimerIntr 가 기본값 0 인 경우에 페이즈와 서보 인터럽트 신호에 외부 클록 신호를 사용해야 합니다. 보통 해당 신호는 시스템의 서보 또는 MACRO IC **Gaten[j]** "DSPGATE1", "DSPGATE2" 또는 "DSPGATE3" 중 하나에서 발생합니다. **Gaten[j].PhaseServoDir** 가 0 으로 설정되어 페이즈와 서보 클록 신호 출력에 지정된 (하나의)IC 가 프로세서에서의 인터럽트 소스가 됩니다. 해당 클록 신호는 ASIC 하드웨어와 Power PMAC 소프트웨어의 완전한 동기를 유지합니다.

이 경우에 CPU 가 전원 셋업/리셋 초기화 처리 종료에서 클록 신호의 존재를 검출하지 않은 경우에 글로벌 Status 비트 **Sys.NoClocks** 를 1 로 설정합니다. 호스트 통신과 같은 백그라운드 태스크는 이어서 동작 가능하지만 포어그라운드(인터럽트에 기초한)

태스크는 실행하지 않습니다. 그리고 모터는 유효가 되지 않습니다. 위치독 타이머는 무효가 됩니다. 클록 신호가 초기화 처리 동안은 존재했다가 그 후 없어지는 경우에 위치독 타이머가 트립합니다(하드웨어 트립인지 소프트웨어 트립인지 설정에 따라 좌우됨).

Sys.CpuTimerIntr 가 1로 설정되는 경우에 Power PMAC CPU는 내부 타이머에서 자체 페이즈와 서보 인터럽트를 생성합니다. 양쪽 인터럽트가 저장 설정 요소 **Sys.ServoPeriod(msec)**로 설정된 주기에 동일한 주기로 발생합니다. 이 설정은 하드웨어 클록 신호를 생성하는 "DSPGATEn" 서보 IC 또는 MACRO IC도 시스템에 없는 경우를 위해 Power PMAC이 시뮬레이션 또는 데이터 취득에 사용될 때나 모터가 해당 ASIC를 하나도 이용하지 않는 EtherCAT과 같은 네트워크에 의해 제어될 때 사용됩니다.

CPU가 자체 인터럽트를 생성하고 있을 때 CPU와 ASIC의 동기를 유지하는 신호를 출력할 수 없다는 점에 주의하십시오.

V1.6 펌웨어(2014년 1/4분기 출시) 이상일 때 재초기화에서 클록을 생성하는 서보 또는 MACRO IC가 검출되지 않은 경우 **Sys.CpuTimerIntr**는 자동으로 1로 설정되고 프로세서는 즉시 그 자체의 타이머 베이스 인터럽트 생성을 시작합니다. 이전 펌웨어 버전에서는 이 경우에 시스템은 **Sys.NoClocks** 에러에 따라 재초기화됩니다. 그리고 **Sys.CpuTimerIntr**의 값을 변경하여 구성을 저장하고 프로세서에 해당 인터럽트를 발생시키기 위해 Power PMAC을 리셋해야 합니다.

Sys.FirstEnc 시스템의 시작 ECT 등록 번호

설명: 최초에 처리하는 엔코더 변환 테이블 등록 번호

범위: 0 .. **Sys.MaxEncoders** - 1

단위: 항목 수

기본값: 0

Sys.FirstEnc는 서보 인터럽트마다 처리하는 엔코더 변환 테이블(ECT)의 최초 등록 번호를 지정합니다. Power PMAC은 **EncTable[n]** 등록 인덱스값($n = \text{Sys.FirstEnc}$)으로 시작하는 테이블 등록의 실행을 시작하고 **Type = 0**(테이블 종료) 등록을 검출할 때까지 수행됩니다.

기본값 0에서 **Sys.FirstEnc**를 변경하는 주요 이유는 개발 중에 다른 변환방법을 사용한 실험을 용이하게 하는 것입니다. 다른 테이블의 섹션은 다른 변환 방법과 파라미터를 포함할 수 있습니다. 그리고 **Sys.FirstEnc**를 변경하면 "서브 테이블" 교환이 간단해집니다. 완성된 어플리케이션에서는 실제로 사용되는 등록만 테이블 중에 남기고 **Sys.FirstEnc**는 테이블의 처음부터 시작하도록 0으로 설정하는 것을 권장합니다.

Sys.I[i] 시스템의 I 변수 배열 요소

설명:	I 변수 배열 요소
범위:	정의에 좌우됨
단위:	정의에 좌우됨
전원 셋업 기본값:	정의에 좌우됨

Sys.I[i]는 "제 *i*"로 표시되는 I 변수 배열 요소입니다. I-변수는 Turbo PMAC 저장 설정 변수의 기능에 해당하는 사전 정의된 저장 설정 요소 변수입니다. 인덱스값 *i* 는 0~8,191 의 범위로 지정할 수 있습니다. 이 배열은 I-변수에 액세스하기 위한 대체 방법을 제공합니다.

Sys.MaxCoords 시스템의 최대 좌표 계수

설명:	사용할 수 있는 최대 좌표계 수
범위:	1 ..128
단위:	좌표계
기본값:	16

Sys.MaxCoords 는 이 Power PMAC 에서 사용할 수 있는 최대 좌표계 수를 지정합니다. 사용할 수 있는 좌표계는 C.S.0 에서 C.S.(**Sys.MaxCoords** - 1)입니다. 데이터 구조체 **Coord[0]**에서 **Coord[Sys.MaxCoords - 1]**을 사용할 수 있고 해당 데이터 구조체의 저장 설정 요소는 **save** 명령으로 비휘발성 플래시 메모리에 복사됩니다.

Sys.MaxCoords 의 값을 증가시킬 때에 새로 사용할 수 있게 되는 좌표계의 저장 설정 요소 설정에 대해 추정으로 수행해서는 안 됩니다.

사용자가 스크립트 명령으로 **Sys.MaxCoords** 의 값을 줄여서 모터가 할당된 좌표계를 이용할 수 없게 되는 경우 명령은 에러로 거절됩니다. 모터가 할당된 좌표계의 최대 번호는 읽기 전용 요소 **Sys.Coords** - 1 과 같으므로 **Sys.MaxCoords** 를 **Sys.Coords** 의 현재값 미만으로 설정할 수 없습니다.

필요 이상으로 큰 **Sys.MaxCoords** 의 설정(> **Sys.Coords**)은 계산 부하를 다소 증가시켜서 백업 파일을 필요 이상으로 커지게 합니다.

Sys.MaxEcats 시스템의 EtherCAT 네트워크 사용 수

설명:	사용할 수 있는 EtherCAT 네트워크의 수
범위:	0 ...8
단위:	단위
기본값:	0

Sys.MaxEcats 는 Power PMAC 에서 유효가 가능한 EtherCAT 네트워크 수를 지정합니다. 인덱스값 i 가 0 부터 **Sys.MaxEcats** - 1 까지의 데이터 구조체 **ECAT[i]**로 표시되는 네트워크를 유효로 할 수 있습니다.

Sys.MaxMotors 시스템의 사용 최대 모터 수

설명:	사용할 수 있는 최대 모터 수
범위:	1 .. 256
단위:	모터
기본값:	32

Sys.MaxMotors 는 Power PMAC 에서 사용할 수 있는 최대 모터 수를 지정합니다. 사용할 수 있는 모터는 **Motor0** 부터 **Motor(Sys.MaxMotors - 1)**입니다. 데이터 구조체 **Motor[0]**부터 **Motor[Sys.MaxMotors - 1]**을 사용할 수 있으며 해당 데이터 구조체의 저장 설정 요소는 **save** 명령으로 비휘발성 플래시 메모리에 복사됩니다.

Sys.MaxMotors 의 값을 증가시킬 때에 새로 사용할 수 있게 되는 모터의 저장 설정 요소 설정에 대해 추정으로 수행해서는 안 됩니다.

사용자가 스크립트 명령으로 **Sys.MaxMotors** 의 값을 줄임으로 인해 유효한 모터(**Motor[x].ServoCtrl** > 0)나 페이즈 태스크가 유효한 모터(**Motor[x].PhaseCtrl** > 0)를 이용할 수 없게 되는 경우에 명령은 에러로 거절됩니다. 유효화되는 모터의 최대 번호는 읽기 요소 **Sys.ServoMotors** - 1 과 같고 페이즈 태스크가 유효한 모터의 최대 번호는 읽기 요소 **Sys.PhaseMotors** - 1 과 같으므로 **Sys.MaxMotors** 를 **Sys.ServoMotors** 또는 **Sys.PhaseMotors** 중 하나를 현재값 미만으로는 설정할 수 없습니다.

필요 이상으로 큰 **Sys.MaxMotors** 의 설정(> **Sys.ServoMotors** 와 > **Sys.PhaseMotors**)은 계산 부하를 다소 증가시켜서 백업 파일을 필요 이상으로 커지게 합니다.

Sys.MaxRtPlc 시스템의 포어그라운드 PLC 의 최대 번호

설명:	포어그라운드 PLC 의 최대 번호
범위:	0 .. 3
단위:	PLC 프로그램 수
기본값:	0

Sys.MaxRtPlc 은 실시간 인터럽트로 실행되는 PLC 프로그램의 최대 번호를 지정합니다. 그보다 큰 번호를 가진 모든 PLC 프로그램은 백그라운드로 실행됩니다. 기본값 0에서는 PLC 0 만 실시간 인터럽트의 포어그라운드 태스크로 실행됩니다. 다른 모든 PLC 는 백그라운드로 실행됩니다. 값 3 의 최대 허가 수에서는 PLC0-3 이 포어그라운드로 실행되고 PLC4-31 이 백그라운드로 실행됩니다.

Sys.MaxTimedUnderflow 시스템의 음의 타임 베이스 초과 시간

설명:	음의 타임 베이스 최대 초과 동작 시간
범위:	음이 아닌 부동 소수점
단위:	명목상 msec
기본값:	0.0(제한 무효)

Sys.MaxTimedUnderflow 는 기본값인 0.0 보다 큰 값인 경우에 트레이스 버퍼가 없고(**Motor[x].TraceSize** = 0) 현재 동작 섹션 또는 세그먼트의 시작을 지나서 실행할 수 있는 최대허가 "시간"을 지정합니다. 그것은 음의 타임 베이스값을 사용하여 반대로 동작할 때 허가됩니다. 음의 타임 베이스값을 사용하는 보간이 이 섹션의 시작을 지나서 **Sys.MaxTimedUnderflow** msec 이상에 모터를 연장하는 경우에 모션은 정지하고 좌표계 내의 모든 모터는 "Abort" 감속으로 정지됩니다.

Sys.MaxTimedUnderflow 는 2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능입니다. 0.0 인 기본값에서는 이 보호가 없었던 이전 펌웨어와 동작의 호환성이 있습니다. 이 보호가 유효가 아니면 음의 타임 베이스 상태에서의 실행은 현재 섹션 또는 세그먼트의 시작을 지나서 무제한으로 계속할 수 있습니다. 그것은 예측 불가능한 결과를 가져올 수도 있습니다.

Sys.MotorsPerRtInt 시스템의 RTI 모터 Status 수

설명:	실시간 인터럽트마다 확인되는 모터 Status 수
범위:	0 .. 255
단위:	모터
기본값:	0(RTI 에서 모두 확인)

Sys.MotorsPerRtInt 는 실시간 인터럽트(RTI)마다 어느 정도의 모터가 그러한 상태와 안전 갱신을 확인할지 지정합니다. 기본값 0에서는 모든 모터가 RTI 마다 확인되고 해당 요소가 없었던 이전 펌웨어 버전과 호환성이 있는 동작을 합니다. (**Sys.MotorsPerRtInt** 는 2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능입니다.)

Sys.MotorsPerRtInt 가 0 보다 큰 값인 경우에 Power PMAC 은 RTI 마다 이 숫자만큼 모터를 확인합니다. 그에 따라 여러 개의 RTI 사이클에 걸쳐 이 확인의 폭을 넓힐 수 있으며 그 밖의 계산에 의해 많은 CPU 시간을 확보할 수 있습니다. 모션 프로그램에서의 높은 블록 레이트, 키네마틱스와(또는) 록 어헤드를 가진 높은 세그먼트이션 레이트와 같은 기능을 필요로 하는 대단히 높은 RTI 주파수를 가지는 일부 어플리케이션에서는 모든 모터를 RTI 마다 상태와 안전성을 확인할 필요가 없어 매우 큰 프로세서 시간을 확보할 수 있습니다. 특히 안전 입력 플래그의 읽기와 같은 I/O 액세스가 필요할 때 매우 많은 프로세서 시간을 획득할 수 있습니다. 이러한 어플리케이션인 경우에 **Sys.MotorsPerRtInt** 를 0 보다 큰 값으로 설정하고 각각의 모터를 연속된 상태와 안전 갱신 기간을 보다 긴 시간 간격으로 하는 것이 유용 할 수 있습니다.

Power PMAC 은 Motor 0 에서 시작해서 가장 높은 번호가 부여된 유효한 모터에서 종료하는 사이클 중에서 모터의 안전과 상태 확인을 실행합니다. 최고 번호는 상태 요소 **Sys.ServoMotors** 에서 확인할 수 있습니다. 이것은(**Sys.ServoMotors+1**) 모터 분량의 사이클이 있음을 의미합니다. 사이클 중의 해당 모터의 순서에서 Power PMAC 은 모터가 유효(**Motor[x].ServoCtrl > 0**)인지 여부를 가장 먼저 확인합니다. 유효한 경우에 해당 모터의 안전성 확인과 상태 갱신이 속행됩니다. 보통 이러한 확인과 갱신은 모터 1 개당 0.5~1.0μsec 가 각 사이클에서 필요합니다.

Sys.MotorsPerRtInt 가 0 보다 큰 경우에 단일의 RTI 사이클에서 Power PMAC 은 지정된 모터 숫자만 확인합니다. 사이클의 총수(**Sys.ServoMotors + 1**)가 **Sys.MotorsPerRtInt** 로 균등하게 나눠 떨어지지 않는 경우에 전체 사이클의 마지막 RTI에서 끝수의 모터가 확인됩니다. 예를 들어 **Sys.ServoMotors** 가 6 인 경우에 전체 사이클에는 7 개의 모터가 있습니다(0~6 을 포함). **Sys.MotorsPerRtInt** 가 3 으로 설정되는 경우에 전체 사이클 최초의 RTI에서는 Motor0, 1 그리고 2 가 확인됩니다. 2 번째 RTI에서는 Motor3, 4 그리고 5 가 확인됩니다. 그리고 3 번째 RTI에서는 Motor6 만 확인됩니다. 이 경우에 전체 사이클은 3RTI 주기입니다. 따라서 각각의 모터는 3RTI 마다 확인됩니다.

실시간 인터럽트 레벨에서 실행된 안전과 상태의 확인은 하드웨어와 소프트웨어의 오버트래블 제한, 앰프 이상, I²T 적분 전류, 엔코더 소실 검출, 브레이크 제어, 백래시 보정 그리고 트리거가 있는 동작을 위한 포착 트리거 모니터링을 포함합니다. (편차 에러와 인포지션 모니터링은 우선도가 높은 서보 인터럽트로 실행되며 **Sys.MotorsPerRtInt** 의 영향을 받지 않습니다.)

RTI 레벨로 수행된 몇 가지 확인과 갱신은 연속하는 확인 간의 주기를 고려하여 계산됩니다. **Sys.MotorsPerRtInt** 가 0 보다 커서 연속 확인 기간을 연장한 경우에 사용자는 이 연장을 고려하여 계산을 수정해야 합니다.



주의

특히 이러한 시간에 좌우되는 안전과 상태 확인의 고종 그리고 이를 위한 자동 설정 소프트웨어가 만약 해당 요소 탑재 전에 쓰인 경우에는 **Sys.MotorsPerRtInt** 로 연장된 확인 주기가 그 기능의 정확한 동작을 위해 필요한 사용자 계산의 수정을 반영하지 않았을 가능성이 있습니다. **Sys.MotorsPerRtInt** 가 확인 주기를 연장하기 위해 사용되는 경우에 사용자는 그것이 정확한지 확인하기 위해 확인 시간의 계산을 수동으로 확인해야 합니다.

이 계산을 수정하기 위해 필요한 시간 연장 요인 N 은 다음 식으로 구할 수 있습니다.

$$N = \text{ceil} \left(\frac{\text{Sys.ServoMotors} + 1}{\text{Sys.MotorsPerRtInt}} \right)$$

이 식의 "ceil"은(필요한 경우) 다음 정수에 올림 처리하는 것을 나타내는 "상한" 기능입니다.

백래시 보정이 모터에서 사용되는 경우 백래시 보정은 "모터 단위/RTI"로 표시되는 **Motor[x].BISlewRate** 의 레이트에서 방향 변화에 의해 덧셈과 뺄셈이 수행됩니다. **Sys.MotorsPerRtInt** 가 0 보다 큰 경우 이것은 기술적으로 "모터/NRTI 주기"입니다. 따라서 **Motor[x].BISlewRate** 의 값은 연장이 없었던 경우와 비교하여 N 의 요인만큼 증가되어야 합니다.

브레이크 제어가 모터에서 사용되는 경우에 지연 주기 **Motor[x].BrakeOffDelay** 그리고 **Motor[x].BrakeOnDelay** 는 msec 로 표시됩니다. Power PMAC 은 RTI 의 확인과 갱신에서 지연에 추종하고 1RTI 주기의 값을 경과 시간으로 추가합니다. **Sys.MotorsPerRtInt** 가 0 보다 큰 경우에 마지막 확인에서 경과한 실시간이 실제로는 N RTI 주기이므로 **Motor[x].BrakeOffDelay** 와 **Motor[x].BrakeOnDelay** 는 확장이 없었던 경우와 비교하여 N 의 요인만큼 줄여야 합니다.

"I²T" 적분 전류 제한 보호가 모터에서 사용되는 경우에 적분 전류 섀다운 제한 **Motor[x].I2tTrip** 은 전류 단위의 제곱×초로 표시됩니다. Power PMAC 은 RTI 의 확인과 갱신 시에 마지막은 1RTI 기간의 값을 확인한 후의 경과 시간을 사용하여 시간을 누적합니다. **Sys.MotorsPerRtInt** 가 0 보다 큰 경우에 마지막 확인에서 경과한 실시간이 실제로는 N RTI 주기이므로 **Motor[x].I2tTrip** 은 연장이 없었던 경우와 비교하여 N 의 요인만큼 줄여야 합니다.

PMAC 전류(commutation) 동기 모터에서 페이즈 서치 동작이 페이즈 기준을 확립하기 위해 사용되는 경우 **Motor[x].PhaseFindingTime** 은 RTI 주기에서 서치의 각 단의 시간을 지정합니다. **Sys.MotorsPerRtInt** 가 0 보다 큰 경우에 이 시간이 실제로는 N RTI 주기가 되므로 **Motor[x].PhaseFindingTime** 은 연장이 없었던 경우와 비교하여 N 의 요인만큼 줄여야 합니다. 단, 이 요소의 수적인 값은 나아가 서치의 타임을 제어합니다. 따라서 설정 변경할 때 모드를 변경하지 않는 것이 중요합니다.

Sys.NoShortCmds 시스템의 생략형 명령 무효 제어

설명: 생략형 온라인 명령 무효 제어

범위: 0 ..1

단위: 논리값

기본값: 0

Sys.NoShortCmds 는 "생략형" 온라인 "동작" 명령, 특히 1 문자의 영문자 명령이 허가되는지 여부를 제어합니다. 기본값 0 인 경우에 생략형 명령이 허가됩니다. 1 로 설정되는 경우에는 허가되지 않고 잘못된 구문 에러로 거절됩니다(에러 20). 이 모드에서는 해당하는 "비생략형" 온라인 명령이 대신 사용되어야 합니다. 값이 0 으로 설정되는 경우는 비생략형 명령도 사용할 수 있다는 점에 주의하십시오.

생략형 명령은 아주 편리하지만 대화식 단말 모드 조작 시에 특히 사용자 선언 변수명을 사용할 때 이를 잘못 발행할 가능성이 있습니다. 명령으로 사용되는 이름이 프로젝트 데이터 베이스에 하나도 해당되지 않는 경우에 Power PMAC 은 이름 속의 문자를 번역하려는 특유의 명령을 합니다. 이것은 예측 불가능하며 경우에 따라서는 위험한 결과를 초래할 수 있습니다.

다음 표는 **Sys.NoShortCmds** 를 1 로 설정하여 무효가 되는 생략형 "동작" 명령과 그에 해당하는 비생략형 명령을 나타낸 것입니다.

생략형 명령	비생략형 명령	생략형 명령	비생략형 명령
a	abort	j=={상수}	jog=={상수}
b	begin	j:{상수}	jog:{상수}
h	hold	j^{상수}	jog^{상수}
hm	home	j=*	jog=*
hmz	homez	j:*	jog:*
j+	jog+	j^*	jog^*
j-	jog-	k	kill
j/	jog/	q	pause
j=	jog=	r	run
j={상수}	jog={상수}	s	step

Sys.NoShortCmds 는 V1.5 펌웨어의 새로운 기능입니다(2012 년 3/4 분기에 출시). 0 인 기본값일 때 동작은 펌웨어의 이전 버전과 호환성이 있습니다.

Sys.pAbortAll 시스템의 글로벌 Abort 입력 레지스터 포인터

설명: 글로벌 Abort 입력 레지스터 포인터

범위: 적정한 주소

단위: 데이터 구조체 요소 주소

기본값: Power Brick: **Gate3[i].GpioData[0].a**
다른 Power PMAC: 0(글로벌 Abort 입력 무효)

Sys.pAbortAll 은 Power PMAC 이 그 "글로벌 Abort" 입력 비트를 감시하는 레지스터를 지정합니다. 여기에는 레지스터의 주소를 설정합니다. **Sys.pAbortAll** 이 기본값 0 인 경우에 Power PMAC 에서 자동 글로벌 Abort 입력 기능은 사용되지 않습니다.

이 변수의 값은 보통 사용되는 레지스터를 나타내는 데이터 구조체 요소의 주소를 할당하여 설정되고 ".a" 접미사를 해당 요소로 사용합니다. 사용자는 이 주소의 수적인 값을 알 필요가 없습니다.

지정된 레지스터에서 Power PMAC 은 입력 Status 를 읽기 위해 **Sys.AbortAllBit** 에 표시되는 번호의 비트를 사용합니다. 글로벌 Abort 가 전송되기 전에 지정된 입력 상태임을 감지하는 스캔의 누적 수는 **Sys.AbortAllLimit** 로 지정됩니다.

지정된 비트가 "1" 상태(극성 제어는 미제공)로 지정된 누적 스캔 수로 검출된 경우 각각의 좌표계는 저장 설정 요소 **Coord[x].AbortAllMode** 의 설정에 따라 동작합니다. 이는 제어되고 정지할지 제어되지 않고 정지할지 지정할 수 있으며 입력을 무시하도록 좌표계를 설정할 수 있습니다.

Sys.PhaseCycleExt 시스템의 페이즈 사이클 연장 기간

설명: 페이즈 사이클 연장 기간

범위: 0 .. 255

단위: 페이즈 인터럽트 주기

기본값: 0

기준 I 변수명: I8007(Turbo PMAC 의 I7)

Sys.PhaseCycleExt 는 연속하는 모터 "페이즈" 소프트웨어 태스크(선택된 모터의 전류(commutation)와 전류 루프 클로즈)의 실행 사이에서 줄어드는 페이즈 인터럽트 횟수를 지정합니다. 해당 태스크는 (**Sys.PhaseCycleExt+1**)페이즈 인터럽트 주기마다 실행됩니다.

페이즈 인터럽트 주기는 시스템 중에서 서보 또는 MACRO IC 중 하나에서 설정됩니다. 사용 파라미터는 PMAC2 스타일 "DSPGATE1" 서보 IC에서는 **Gate1[j].PwmPeriod** 과 **Gate1[j].PhaseClockDiv** 이고 PMAC2 스타일 "DSPGATE2" MACRO IC에서는 **Gate2[j].PwmPeriod** 과 **Gate2[j].PhaseClockDiv** 이며 PMAC3 스타일 "DSPGATE3" 머신 인터페이스 IC에서는 **Gate3[j].PhaseFreq** 입니다.

대부분의 Power PMAC 어플리케이션에서는 **Sys.PhaseCycleExt** 를 기본값 0 그대로 둡니다. 그 결과 모터 페이즈 알고리즘은 페이즈 인터럽트 주기로 실행됩니다. **Sys.PhaseCycleExt** 를 0 보다 크게 해서 모터 페이즈 갱신 사이클을 연장하는 이유는 2 가지입니다.

첫째, Power PMAC 이 MACRO 링에서 다이렉트 PWM 모터 제어를 하는 경우에 **Sys.PhaseCycleExt** 를 1로 설정하는 것을 권장합니다. 이는 하드웨어 페이즈 클록에서 동작하는 MACRO 링으로 소프트웨어 페이즈 사이클당 2 하드웨어 사이클이 필요한 설정입니다. 그에 따라 전류 루프를 닫을 때 1개의 페이즈 사이클 지연이 없어지고 높은 게인과 고성능이 가능해집니다. 예를 들어 하드웨어 페이즈 클록은 18kHz로 설정할 수 있지만 **Sys.PhaseCycleExt** = 1의 지정에서 전류 루프는 9kHz에서 닫힙니다.

둘째, ACC-36 보드의 멀티플렉서 A/D 변환기가 서보 피드백에 사용되는 경우에 각각의 A/D 변환기가 서보 사이클에서 반드시 1회는 처리될 수 있도록 **Sys.PhaseCycleExt** 를 0 보다 크게 할 수 있습니다. 1 세트의 멀티플렉서 ADC 가 하드웨어 페이즈 클록 사이클마다 처리됩니다.

예를 들어 440μsec(2.25kHz) 매 서보 사이클 처리 시에 멀티플렉서 ADC 가 8 세트 필요한 경우와 소프트웨어 페이즈 갱신에 220μsec(4.5 kHz)가 필요한 경우에 페이즈 클록 갱신은 매 서보 사이클에 8 세트의 ADC 를 모두 얻을 수 있도록 18kHz(18/8 = 2.25)로 설정되고 소프트웨어 페이즈 갱신이 4.5 kHz 로 실시될 수 있도록 **Sys.PhaseCycleExt** 는 3 으로 $(18/[3+1]= 4.5)$ 설정됩니다. 또한 서보 사이클 클록 분주기는 8 분주되도록 설정됩니다(**Gaten[j].ServoClockDiv** = 7).

매끄러운 동작을 위해 서보 클록 주기는 소프트웨어 페이즈 갱신의 정수 배가 되어야 합니다. 예를 들어 서보 클록 주파수가 페이즈 클록 주파수의 1/4 인 경우(**Gaten[j].ServoClockDiv** = 3)에 **Sys.PhaseCycleExt** 가 본래 값인 0 이며 서보 클록 주기당 4 소프트웨어 페이즈 갱신이 됩니다. 1 인 경우에는 2 페이즈/서보 주기이고 3 인 경우에는 1 페이즈/서보 주기입니다. 이 규칙에 따라 소프트웨어 페이즈 갱신 주기가 절대 서보 클록 주기 이상으로 되지 않도록 주의하십시오.

Sys.PhaseOverServoPeriod 시스템의 페이즈에 대한 서보 갱신 주기비

설명: 페이즈 갱신 주기에 대한 서보 갱신 주기의 비

범위: 0.0 .. 1.0

단위: 없음

기본값: 0.25

Sys.PhaseOverServoPeriod 는 부동 소수점 숫자로 표현되는 서보 갱신 주기에 대한 페이즈 갱신 주기의 비율을 지정합니다. 모터가 페이즈 갱신으로 서보 알고리즘을 실행 중인 경우(**Motor[x].PhaseCtrl** 의 비트 3 을 1로 설정하여 유효가 된 경우)에만 이 값은 사용됩니다. 이는 검류계와 같은 아주 고대역 액추에이터를 위한 특수 기능입니다. 이 경우에 매 페이즈 사이클의 보간이 서보 사이클의 몇 분할이 되는지를 페이즈 갱신의 보간 알고리즘으로 표시합니다.

페이즈와 서보 주기 사이의 실제 비율은 Power PMAC 시스템에 해당 클록 신호를 제공하는 서보 IC 또는 MACRO IC 의 저장 설정 요소 **Gaten[j].ServoClockDiv** 로 가장 일반적으로 설정됩니다. **Sys.PhaseOverServoPeriod** 에서는 이 비율이 몇 개를 페이즈 갱신 알고리즘으로 표시되도록 사용자가 설정해야 합니다. 다음과 같이 계산합니다.

$$\text{Sys.PhaseOverServoPeriod} = \frac{1}{\text{Gaten}[i].\text{ServoClockDiv} + 1}$$

Gaten[i].ServoClockDiv 가 기본값 3 이면 $1/(3+1) = 0.25$ 가 **Sys.PhaseOverServoPeriod** 의 값이 됩니다.

Sys.PreCalc 시스템의 동작 전 계산 버퍼 시간

설명: 동작 전 계산 버퍼 시간

범위: 0 ..15

단위: 서보 클럭 사이클

기본값: 1

Sys.PreCalc 는 Power PMAC 이 계산하는 연속적인 프로그램 궤적의 전방에 대해 최소 간격을 지정합니다. 즉 Power PMAC 이 모션 프로그램에서 동작 방정식을 산출할 때에 실행 준비가 된 궤적 내의 계산 완료 포인트에서 1 실시간 인터럽트(RTI) 주기 더하기 **Sys.PreCalc** 서보 사이클 분량의 앞까지 대응하는 동작 방정식이 있을 때까지 계산을 계속합니다. 즉 $(\text{Sys.RtIntPeriod} + 1) + \text{Sys.PreCalc}$ 사이클 전방입니다.

대부분의 Power PMAC 어플리케이션은 기본값인 1 을 **Sys.PreCalc** 에 사용할 수 있습니다. 그 시간이 단일 서보 사이클에 가깝고 다수의 미소 프로그램된 동작·동작 세그먼트에 따른 어플리케이션만이 항상 충분한 사전계산이 실행되었다는 것을 보증하기 위해서 해당 값을 증가시켜야 하는 경우가 있습니다.

어플리케이션에서 **Sys.PreCalc** 가 너무 작은 경우에 사전 계산이 동작 실행(보간) 전에 종료되지 않을 때에 프로그램 실행 중에 "런타임 에러"가 발생하는 경우가 있습니다. 실제 장애는 전형적으로 많은 계산을 필요로 하는 미소 이동의 집중 및(또는) 계산의 과부하에 의해 RTI 사이클이 누락하기 때문에 발생합니다. 이 경우에 좌표계의 상태 요소 **Coord[x].RuntimeError** 는 1 로 설정됩니다(**Coord[x].ErrorStatus** 는 16 으로 설정됩니다).

어플리케이션에서 **Sys.PreCalc** 가 너무 큰 경우에 계산과 실행 사이에 지연이 증가하고 현재 시스템 상태에 기초한 조건부 실행이 있는 경우에 문제가 될 수도 있습니다. 또한 추가 사전 계산 거리는 16 세그먼트 버퍼를 오버플로시킵니다. 이 경우에 좌표계의 상태 요소 **Coord[x].BufferError** 가 1 로 설정됩니다(**Coord[x].ErrorStatus** 는 2 로 설정됩니다).

펌웨어 버전 1.6(2014 년 1/4 분기 출시) 이상에서 좌표계 상태 요소 **Coord[x].BufferWarn** 을 높은 블록 레이트의 어플리케이션에서 **Sys.PreCalc** 의 설정을 최적화하기 위해 사용할 수 있습니다. 가끔 **BufferWarn** 이 1 이 되도록 **Sys.PreCalc** 의 설정을 충분히 높게 해야 할 필요가 있으나(동작 방정식 버퍼에 다른 동작 블록을 로드할 수 없을 수도 있다는 것을 의미) **BufferWarn** 이 2 가 되면 너무 높습니다(버퍼 오버플로를 회피하기 위해 다음 RTI 까지 동작 계산 요구를 연기해야 한다는 것을 의미).

Sys.RtIntPeriod 시스템의 실시간 인터럽트 주기

설명: 실시간 인터럽트 주기

범위: 0 .. 255

단위: 서보 클록 사이클 - 1

기본값: 2

기준 I 변수명: I8008(Turbo PMAC 의 I8)

Sys.RtIntPeriod 는 Power PMAC 의 "실시간 인터럽트" 주기를 지정합니다. 실시간 인터럽트는 (**Sys.RtIntPeriod** + 1) 서보 인터럽트마다 발생합니다. 기본값인 2 에서는 3 서보 인터럽트마다 실시간 인터럽트가 발생합니다.

모든 실시간 인터럽트에서 Power PMAC 은 새로운 프로그램된 동작 또는 프로그램 동작에서 파생한 새로운 동작 세그먼트에 모션 프로그램 계산이 필요하게 되는지 여부를 확인합니다. 또한 각각의 유효한 포어그라운드 PLC 프로그램이 스캔됩니다.

실시간 인터럽트 주기는 어떤 좌표계에서도 최소의 동작 세그먼트 시간(**Coord[x].SegMoveTime**)보다 작아야 합니다. 그렇지 않으면 Power PMAC 이 세그먼트 모션식을 시간 내에 계산할 수 없을 수도 있습니다. 그리고 모션 프로그램은 "런타임 에러"로 정지합니다.

Power PMAC 은 실시간 인터럽트 중에 위치독 타이머 카운터를 뺍니다. 이 뺄셈이 적어도 초당 50 회 발생하지 않을 정도로 큰 **Sys.RtIntPeriod** 가 설정되는 경우에 위치독 타이머가 트립하고 Power PMAC 이 셧다운할 수도 있습니다.

Sys.SendFileMode 시스템의 송신 메시지 형식

설명: 송신 메시지의 형식 모드

범위: 0 ..31

단위: 비트 필드

기본값: 0

Sys.SendFileMode 는 **sendn** 명령을 사용하여 전송된 텍스트 문자열을 어떻게 종결할지 지정합니다. **Sys.SendFileMode** 는 비트 0~4 의 독립된 5 비트로 구성됩니다. 각각의 비트 n 은 **sendn** 명령에서 사용되는 포트 n 에 대응합니다.

비트 n 이 기본값 0 인 경우에 추가 문자는 해당 포트에 송신된 텍스트 문자열에 추가되지 않고 **sendn** 명령으로 명시적으로 지정된 문자만 송신됩니다. 비트 n 이 1 로 설정되는 경우에 문자 "- n :"(하이픈, 포트 번호 숫자, 콜론)이 텍스트 문자열의 처음에 자동으로 추가되고 "null" 문자(ASCII 바이트값인 0)가 텍스트 문자열의 마지막에 자동으로 추가되어 개별 **sendn** 명령마다 전송됩니다.

null 문자의 추가는 수신 소프트웨어에서 개별 메시지를 확실하게 식별하기에 용이하게 합니다. 이와 같은 이유로 IDE의 "Unsolicited Messages" 창은 자동으로 해당 비트를 1로 설정합니다. 단, 그에 따라 다른 **sendn** 명령에서의 문자열의 다이렉트 연결이 불가능해집니다.

Sys.ServoPeriod 시스템의 보간용 서보 갱신 주기

설명: 보간용 서보 갱신 주기

범위: 양의 부동 소수점

단위: msec

기본값: 0.44274211

기존 I 변수명: I8010(Turbo PMAC의 I10)

Sys.ServoPeriod는 서보 인터럽트 간에 어느 정도의 시간이 있는지 Power PMAC 소프트웨어에 통지합니다(이는 하드웨어 회로에서 제어됨). 그 결과 보간 소프트웨어는 서보 인터럽트 사이에 필요한 시간을 인식합니다. 또한 초 단위로 경과 시간을 계산하고 "I²T" 적분 전류 제한이나 적응 서보 제어 알고리즘에서 사용됩니다. 부동 소수점 값의 msec로 표시됩니다. 수치적으로 서보 인터럽트 주파수 킬로헤르츠(kHz)의 역수입니다.

서보 인터럽트 주기가 PMAC2 스타일 서보 IC(DSPGATE1)로 결정되는 경우에 **Gate1[i].PwmPeriod**, **Gate1[i].PhaseClockDiv** 그리고 **Gate1[i].ServoClockDiv**로 제어되고 *i*는 시스템의 서보 클록을 제어하는 IC의 번호가 됩니다. (보통 Power UMAC 시스템에서 *i* = 4.) 다음 식으로 산출할 수 있습니다.

$$\text{Sys.ServoPeriod} = \frac{(2 * \text{PwmPeriod} + 3)(\text{PhaseClockDiv} + 1)(\text{ServoClockDiv} + 1)}{117,964.8}$$

서보 인터럽트 주기가 PMAC2 스타일 MACRO IC (DSPGATE2)로 결정되는 경우에 **Gate2[i].PwmPeriod**, **Gate2[i].PhaseClockDiv** 그리고 **Gate2[i].ServoClockDiv**로 제어되고 *i*는 시스템의 서보 클록을 제어하는 IC의 번호가 됩니다. (보통 Power UMAC 시스템에서 *i* = 0.) 위와 동일한 식으로 계산합니다.

서보 인터럽트 주기가 PMAC3 스타일 머신 인터페이스 IC(DSPGATE3)로 결정되는 경우 **Gate3[i].PhaseFreq**와 **Gate3[i].ServoClockDiv**로 제어되고 *i*는 시스템의 서보 클록을 제어하는 IC의 번호가 됩니다. (보통 Power UMAC 시스템에서 *i* = 0.) 다음 식으로 산출할 수 있습니다.

$$\text{Sys.ServoPeriod} = 1000 * \frac{(\text{Gate3}[i].\text{ServoClockDiv} + 1)}{\text{Gate3}[i].\text{PhaseFreq}}$$

저장 설정 요소 **Sys.CpuTimerIntr**가 1로 설정되고 Power PMAC 프로세서가 자체 페이지와 서보 인터럽트를 내부에서 생성하는 경우에 동일하게 **Sys.ServoPeriod**에는 인터럽트의 물리적인 주기를 지정합니다. 이 주기는 전원 셋업/리셋의 시에만 설정되므로 설정을 저장하고 Power PMAC을 리셋하면 변경이 반영됩니다.

Sys.SimConfigOk 시스템의 시뮬레이트 구성 다운로드 유효

설명: 시뮬레이트 된 구성의 다운로드 유효

범위: 0 .. 1

단위: 논리값

기본값: 0

Sys.SimConfigOk에서는 존재하지 않는 하드웨어 구성의 주소 설정을 다운로드하는 것이 허용되는지 여부를 결정합니다. 기본설정인 0에서는 시스템에 존재하지 않는 하드웨어(예를 들어 **Acc24E2A[6].Chan[0].Dac[0].a**)를 저장 설정 요소(예를 들어 **Motor[5].pDac**)에 주소 지정하는 명령을 Power PMAC에 송신하면 명령은 에러로 거절됩니다.

그러나 **Sys.SimConfigOk**를 1로 설정한 경우에 시스템에 존재하지 않는 하드웨어를 저장 설정 요소에 주소 지정하는 명령에도 에러가 되지 않고 수신됩니다. 이때 해당 요소의 값이 참조되는 경우에 Power PMAC은 요소명(예를 들어 **Acc24E2A[6].Chan[0].Dac[0].a**)을 사용하는 대신 존재하지 않는 하드웨어 주소의 오프셋값(예를 들어 **Sys.piom+\$700040**)을 통지합니다.

Sys.SimConfigOk를 1로 설정하면 프로세서만의 시뮬레이션으로 소프트웨어 프로젝트 구성을 테스트하는 데 이용할 수 있습니다. 다양한 시스템 구성이 모듈 양식으로 생성되는 경우에도 동일하게 이용할 수 있습니다. 완전한 시스템 구성을 부분적인 시스템으로 에러 없이 다운로드할 수 있습니다.

Sys.SimConfigOk는 V1.6 펌웨어(2014년 1/4분기 출시)의 새로운 기능입니다. 기본값 0에서는 이 요소가 없었던 이전 펌웨어 버전과 동작 호환성이 있습니다.

Sys.WDTRReset 시스템의 포어그라운드 워치독 타이머 최소치

설명: 포어그라운드 워치독 타이머 최소치

범위: 0 .. $2^{32}-1$

단위: 실시간 인터럽트 주기

기본값: 0(5000 사이클을 지정)

Sys.WDTRReset은 Power PMAC이 "소프트웨어" 워치독 타이머 이상을 일으키지 않고 백그라운드 소프트웨어 사이클을 종료시킬 수 있는 "실시간 인터럽트"(RTI) 최대 사이클 수를 지정합니다. 기본값 0인 경우(또는 100 미만의 임의의 값)에 5000RTI 사이클의 값이 사용됩니다. 이 감시 기능은 프로세서의 소프트웨어 과부하를 검출, 방지하여 안전한 제어를 실시하는 것을 목적으로 하며 나아가 섀다운 기능을 제공합니다.

백그라운드 소프트웨어는 페이지, 서보 그리고 실시간 인터럽트 상황에서 실행하는 태스크 사이의 이용 가능한 시간을 실행합니다. 이는 유효한 한 개의 백그라운드 스크립트 PLC 프로그램의 단일 스캔, 유효한 모든 백그라운드 C PLC 프로그램의 단일 스캔 그리고 여러가지 "하우스키핑" 태스크를 각 사이클에서 실행하고 주기적으로 그 여러가지 태스크를 실행합니다. 그런 특성상 해당 백그라운드 태스크의 사이클 시간은 정확하게 예측할 수 없지만 해당 태스크는 항상 합리적으로 적절한 타이밍에 실행해야 합니다.

유효한 기본값인 5000RTI 사이클은 대다수의 Power PMAC 어플리케이션에서 적합합니다. 단, 특수한 타이밍 요건을 가진 어플리케이션에서는 다른 설정이 필요할 수 있습니다.

소프트웨어 워치독 타이머의 트립에서는 모든 모션 프로그램이 Abort 되고 모든 모터는 소프트웨어에서 kill 됩니다(오픈 루프, 제로 출력, 앰프 무효). 그리고 인터페이스 ASIC 는 모두 "리셋"상태로 고정되고 이는 앰프 Enable 을 포함하는 각각의 출력을 모두 정지하며 모든 연속 가능 명령 출력(DAC, PWM, PFM)은 제로 명령값으로 강제로 출력됩니다. 단, Power PMAC 프로세서는 유효한 상태 그대로 PLC 프로그램을 실행할 수 있고 호스트 컴퓨터와 통신할 수 있습니다.

백그라운드 사이클마다 읽기 전용 요소 **Sys.WdTimer** 는 **Sys.WDTReset** 의 값(또는 **Sys.WDTReset** 이 100 미만인 경우는 5000)으로 설정됩니다. RTI 사이클마다 **Sys.WdTimer** 에서 1 만큼 뺍니다. **Sys.WdTimer** 의 값이 0에 도달하는 경우에 Power PMAC 은 소프트웨어 워치독 타이머 트립을 실행합니다. 이 타입의 이상이 발생한 경우에 읽기 전용 요소 **Sys.WDTFault** 는 1로 설정됩니다.

POWER PMAC의 비저장 설정 데이터 구조체 요소

이 장에서는 어플리케이션의 작성에 유용한 데이터 구조체에 대하여 설명합니다. 여기에서 설명하는 데이터 구조체의 값은 저장 명령을 사용하여 값을 플래시 메모리에 저장할 수 없습니다. 또한 전원 셋업/리셋 시에 플래시 메모리에서 복사되지 않습니다.

Acc5E[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc5E[i]]는 기본적인 데이터 구조체 [Gate2[i]]의 스크립트 환경에 대한 별칭입니다. [Gate2[i]] 데이터 구조체의 항목으로 ACC-5E MACRO 인터페이스 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc5E3[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc5E3[i]]는 기본적인 데이터 구조체 [Gate3[i]]의 스크립트 환경에 대한 별칭입니다. [Gate3[i]] 데이터 구조체의 항목으로 ACC-5E3 MACRO 인터페이스 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc5EP3[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc5EP3[i]]는 기본적인 데이터 구조체 [Gate3[i]]의 스크립트 환경에 대한 별칭입니다. [Gate3[i]] 데이터 구조체의 항목으로 ACC-5EP3 Etherlite MACRO 인터페이스 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc11C[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc11C[i]]는 기본적인 데이터 구조체 [Gatelo[i]]의 스크립트 환경에 대한 별칭입니다. [Gatelo[i]] 데이터 구조체의 항목으로 ACC-11C 디지털 I/O 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc11E[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc11E[i]]는 기본적인 데이터 구조체 [Gatelo[i]]의 스크립트 환경에 대한 별칭입니다. [Gatelo[i]] 데이터 구조체의 항목으로 ACC-11E 디지털 I/O 보드의 데이터 구조체 요소를 목록화하여 후술합니다. ACC-11E는 Power UMAC CPU에 의해서 자동으로 식별되지 않기 때문에 이 데이터 구조체를 사용하는 경우에는 수동으로 [Gatelo[i].PartNum]을 603,307로 설정하고 [Gatelo[i].PartType]을 8로 설정하여 **save** 명령을 발행한 다음 컨트롤러를 리셋해야 합니다.

Acc14E[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc14E[i]]는 기본적인 데이터 구조체 [Gatelo[i]]의 스크립트 환경에 대한 별칭입니다. [Gatelo[i]] 데이터 구조체의 항목으로 ACC-14E 디지털 I/O 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc24C2[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc24C2[i]]는 기본적인 데이터 구조체 [Gate1[i]]의 스크립트 환경에 대한 별칭입니다. [Gate1[i]] 데이터 구조체의 항목으로 ACC-24C2 디지털 서보 인터페이스 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc24C2A[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc24C2A[i]]는 기본적인 데이터 구조체 [Gate1[i]]의 스크립트 환경에 대한 별칭입니다. [Gate1[i]] 데이터 구조체의 항목으로 ACC-24C2A 아날로그 서보 인터페이스 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc24E2[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc24E2[i]]는 기본적인 데이터 구조체 [Gate1[i]]의 스크립트 환경에 대한 별칭입니다. [Gate1[i]] 데이터 구조체의 항목으로 ACC-24E2 디지털 서보 인터페이스 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc24E2A[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc24E2A[i]]는 기본적인 데이터 구조체 [Gate1[i]]의 스크립트 환경에 대한 별칭입니다. [Gate1[i]] 데이터 구조체의 항목으로 ACC-24E2A 아날로그 서보 인터페이스 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc24E2S[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc24E2S[i]]는 기본적인 데이터 구조체 [Gate1[i]]의 스크립트 환경에 대한 별칭입니다. [Gate1[i]] 데이터 구조체의 항목으로 ACC-24E2S 스테퍼 인터페이스 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc24E3[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc24E3[i]]는 기본적인 데이터 구조체 [Gate3[i]]의 스크립트 환경에 대한 별칭입니다. [Gate3[i]] 데이터 구조체의 항목으로 ACC-24E3 서보 인터페이스 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc36E[i].비저장 설정 데이터 구조체 요소

[Acc36E[i]] 데이터 구조체는 16 채널 12 비트의 ACC-36E A/D 컨버터 보드의 하드웨어 레지스터에 대한 액세스를 제공합니다.

Acc36E[i].ConvertCode

설명: ADC의 선택 및 변환 코드

범위: \$000000 .. \$00F00F

단위: 비트 필드

전원 셋업 시의 기본값: 0

[Acc36E[i].ConvertCode]는 보드 인덱스 'i'로 지정된 ACC-36E 보드상의 다중화 A/D 컨버터(ADC) 보드의 페어에서 다음에 선택할 페어를 지정함과 동시에 유니폴라 또는 바이폴라 모드 중 하나로 변환할지 여부를 지정합니다. 이는 6 자릿수 16 진법으로 나타내는 24 비트값입니다.

[Acc36E[i].ConvertCode]의 정규 값은 \$00m00n의 형식을 가집니다. *m* 및 *n*에는 0부터 F 까지 임의의 16 진법을 사용할 수 있습니다. *m* 값은 다음 방정식에 따라서 상위의 아날로그 입력 ANAI08~ANAI15에서 읽을 값 및 변환 방법을 결정합니다.

$m = \text{ANAI\#} - 8$ // 0V~+20V 유니폴라 입력, 부호가 없는 결과
 $m = \text{ANAI\#}$ // -10V~+10V 바이폴라 입력, 부호가 있는 결과

이 변환의 결과는 Status 비트[Acc36E[i].ADCRdyHigh]가 1로 설정되면(변환 코드가 설정되고 약 5 마이크로초 후) Status 요소 [Acc36E[i].ADCuHigh]에는 부호가 없는 결과가 나타나고 [Acc36E[i].ADCsHigh]에는 부호가 있는 결과가 나타납니다.

n 값은 다음 방정식에 따라서 하위 아날로그 입력 ANAI08~ANAI15에서 읽을 값 및 그 값의 변환 방법을 결정합니다.

$m = \text{ANAI\#}$ // 0V~+20V 유니폴라 입력, 부호가 없는 결과
 $m = \text{ANAI\#} + 8$ // -10V~+10V 바이폴라 입력, 부호가 있는 결과

이 변환의 결과는 Status 비트 [Acc36E[i].ADCRdyLow]가 1로 설정되면(변환 코드가 설정되고 약 5 마이크로초 후) Status 요소 [Acc36E[i].ADCuLow]에는 부호가 없는 결과가 나타나고 [Acc36E[i].ADCsLow]에는 부호가 있는 결과가 나타납니다.

[Acc36E[i].ConvertCode]에 쓰인 값을 되읽을 수는 없습니다. 이 값을 읽으려면 Status 요소 [Acc36E[i].ADCHighLow] 내의 값을 읽습니다.

[Acc36E[i].ConvertCode]는 저장 설정 요소 [AdcDemux.Address[i]]에 따라서 주소 지정되고 [AdcDemux.Enable]에 따라서 유효가 된 경우에 자동 ADC 다중화 해제 알고리즘에 따라서 페이지 사이클마다 씁니다. 이 경우에 이 요소에 직접 쓰기를 시행하면 안 됩니다.

스크립트 환경에서는 [Acc36E[i].ConvertCode]는 32 비트 데이터 버스의 상위 24 비트 내의 24 비트 요소입니다. C 환경에서는 하위 8 비트가 'don't care'값의 32 비트 요소이므로 이 요소의 값은 스크립트 환경값의 256 배가 됩니다(이 때문에 C에서 형식은 0x00m00n00).

Acc51C[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc51C[i]]는 기본적인 데이터 구조체 [Gate1[i]]의 스크립트 환경에 대한 별칭입니다. [Gate1[i]] 데이터 구조체의 항목으로 ACC-51C 사인파 엔코더 인터페이스 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc51E[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc51E[i]]는 기본적인 데이터 구조체 [Gate1[i]]의 스크립트 환경에 대한 별칭입니다. [Gate1[i]] 데이터 구조체의 항목으로 ACC-51E 사인파 엔코더 인터페이스 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc58E[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc58E[i]]는 기본적인 데이터 구조체 [Gate1[i]]의 스크립트 환경에 대한 별칭입니다. [Gate1[i]] 데이터 구조체의 항목으로 ACC-58E 리졸버/디지털 컨버터 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc59E[i].비저장 설정 데이터 구조체 요소

[Acc59E[i]] 데이터 구조체는 8 채널 12 비트의 ACC-59E A/D 컨버터 보드, D/A 컨버터 보드의 하드웨어 레지스터에 대한 액세스를 제공합니다.

Acc59E[i].ConvertCode

설명: ADC의 선택 및 변환 코드

범위: \$000 .. \$00F

단위: 비트 필드

전원 셋업 시의 기본값: 0

[Acc59E[i].ConvertCode]는 보드 인덱스 'i'로 지정된 ACC-59E 보드상의 다중화 A/D 컨버터(ADC) 보드에서 다음에 선택할 보드를 지정함과 동시에 유니폴라 또는 바이폴라 모드 중 하나로 변환할지 여부를 지정합니다. 이는 3 자릿수 16 진법으로 나타내는 12 비트값입니다.

[Acc59E[i].ConvertCode]의 정규 값은 \$00m의 형식을 가집니다. m에는 0부터 F까지 임의의 16 진법을 사용할 수 있습니다. m 값은 다음 방정식에 따라서 아날로그 입력 ANAI1~ANAI8에서 읽을 값 및 변환 방법을 결정합니다.

```
m = ANAI# - 1      // 0V~+20V 유니폴라 입력, 부호가 없는 결과
m = ANAI# + 7      // -10V~+10V 바이폴라 입력, 부호가 있는 결과
```

이 변환의 결과는 Status 비트 [Acc59E[i].ADCRdy]가 1로 설정되면(변환 코드가 설정되고 약 5 마이크로초 후) Status 요소 [Acc59E[i].ADCu]에는 부호가 없는 결과가 나타나고 [Acc59E[i].ADCs]에는 부호가 있는 결과가 나타납니다.

[Acc59E[i].ConvertCode]에 쓰인 값을 되읽을 수 없습니다. 이 값을 읽으려면 Status 요소 [Acc59E[i].ADCu] 또는 [ADCs] 내의 값을 읽습니다.

[Acc59E[i].ConvertCode]는 저장 설정 요소 [AdcDemux.Address[i]]에 따라서 주소 지정되고 [AdcDemux.Enable]에 따라서 유효가 된 경우에 자동 ADC 다중화 해제 알고리즘에 따라서 페이지 사이클마다 씁니다. 이 경우에 이 요소에 직접 쓰기를 시행하면 안 됩니다.

스크립트 환경에서는 [Acc36E[i].ConvertCode]는 24 비트 데이터 버스의 하위 12 비트 내의 12 비트 요소입니다. C 환경에서는 하위 8 비트 및 상위 12 비트가 'don't care'값의 32 비트 요소이므로 이 요소의 값은 스크립트 환경 값의 256 배가 됩니다(이 때문에 C에서 형식은 0x00000m00).

Acc59E[i].DAC[j]

설명: DAC 명령값

범위: 0 .. 4095

단위: 12 비트 DAC LSB

전원 셋업 시의 기본값: 0

[Acc59E[i].DAC[j]]는 인덱스 'j'에 따라 보드 인덱스 'i'로 지정된(DAC_n 출력 핀에 대응, $j = n - 1$) ACC-59E 보드상에 있는 12 비트의 D/A 컨버터에 쓰일 값을 지정합니다. 이는 부호가 없는 12비트값으로 값 0 은 DAC_{j+} 및 DAC_{j-} 핀의 0V 출력에 대응하고 값 4,095 는 DAC_{j+} 핀의 +10V 출력 및 DAC_{j-} 핀의 -10V 출력에 대응합니다.

C 환경에서 [Acc59E[i].DAC[j]]에 액세스할 수 없습니다. 대신 C 프로그램에서는 듀얼 DAC 요소 [Acc59E[i].DacHighLow[j]]를 사용해야 합니다.

Acc59E[i].DACHighLow[j]

설명: DAC 페어 결합 명령값

범위: 0 .. 16,777,215

단위: 듀얼 12 비트 DAC LSB

전원 셋업 시의 기본값: 0

[Acc59E[i].DACHighLow[j]]는 인덱스 'j'에 따라 12 비트 D/A 컨버터에 쓸 값을 지정하고(DAC_n 출력 핀에 대응, $j = n - 1$) 인덱스 'j+4'에 따라 보드 인덱스 'i'로 지정된 ACC-59E 보드상의 12 비트 아날로그 컨버터에 쓸 값을 지정합니다(DAC_{n+4} 출력핀에 대응). 이는 부호가 없는 24 비트값이며 하위 12 비트는 인덱스 'j'에 따라 DAC 에 쓸 부호 없는 값을 지정하고 상위 12 비트는 인덱스 'j+4'에 따라 DAC 에 쓸 부호 없는 값을 지정합니다.

각 DAC 에 대하여 값 0 은 DAC_{j+} 및 DAC_{j-} 핀의 0V 출력에 대응하고 값 4,095 는 DAC_{j+} 핀의 +10V 출력 및 DAC_{j-} 핀의 -10V 출력에 대응합니다. [Acc59E[i].DACHighLow[j]]의 값을 어셈블하려면 인덱스 'j+4'로 지정된 DAC 의 명령값에 4,096 을 곱하고 거기에 인덱스 'j'로 지정된 DAC 의 값을 더합니다.

스크립트 환경에서는 [Acc59E[i].DACHighLow[j]]는 32 비트 데이터 버스의 상위 24 비트 내의 24 비트값입니다. C 환경에서는 하위 8 비트가 'don't care'비트의 32 비트값이기 때문에 이 요소의 값은 스크립트 환경 값의 256 배 값이 됩니다.

Acc59E3[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc59E3[i]]는 기본적인 데이터 구조체 [Gate3[i]]의 스크립트 환경에 대한 별칭입니다. [Gate3[i]] 데이터 구조체의 항목으로 ACC-59E3 A/D 및 D/A 컨버터 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc65E[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc65E[i]]는 기본적인 데이터 구조체 [Gatelo[i]]의 스크립트 환경에 대한 별칭입니다. [Gatelo[i]] 데이터 구조체의 항목으로 ACC-65E 디지털 I/O 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc66E[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc66E[i]]는 기본적인 데이터 구조체 [Gatelo[i]]의 스크립트 환경에 대한 별칭입니다. [Gatelo[i]] 데이터 구조체의 항목으로 ACC-66E 디지털 입력 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc67E[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc67E[i]]는 기본적인 데이터 구조체 [Gatelo[i]]의 스크립트 환경에 대한 별칭입니다. [Gatelo[i]] 데이터 구조체의 항목으로 ACC-67E 디지털 출력 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc68E[i].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Acc68E[i]]는 기본적인 데이터 구조체 [Gatelo[i]]의 스크립트 환경에 대한 별칭입니다. [Gatelo[i]] 데이터 구조체의 항목으로 ACC-68E 디지털 I/O 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Acc72EX[i].비저장 설정 데이터 구조체 요소

Power PMAC은 ACC-72EX 필드 버스 인터페이스 카드와 이 카드에 고유의 필드 버스 통신 옵션을 전부 지원합니다. Power PMAC에서 ACC-72EX 듀얼 포트 RAM에 액세스하기 위해서 내장된 데이터 구조체에 따라 추가 소프트웨어를 사용하지 않고 메모리 매핑이나 식별을 할 수 있습니다.

이 섹션에서는 ACC-72EX 용으로 Power PMAC에 내장된 데이터 구조체에 대해서 설명하고 헤더 파일, 스타트업 및 핸드셰이킹 PLC의 예시를 나타냅니다.

ACC-72EX와의 상호 작용은 모두 ACC-72EX 전용으로 정의된 Power PMAC 펌웨어에 내장된 데이터 구조체를 사용하여 실행할 수 있습니다. 다음 구조체는 비트, 바이트, 2바이트 및 4바이트 폭의 액세스 모드로 DPRAM에 액세스할 수 있습니다. 비트 단위의 읽기 및 쓰기는 [Acc72EX[i].Udata16[j]] 데이터 구조체를 사용한 경우에만 지원됩니다.

Acc72EX[j].Idata16[j]

설명: ACC-72EX DPRAM 부호가 있는 16 비트 정수 데이터 배열 요소

범위: $-2^{15} \dots 2^{15}-1$

단위: 주소에 좌우됨

전원 셋업 시의 기본값: 주소에 좌우됨

[Acc72EX[j].Idata16[j]]는 [Acc72EX[j]] 듀얼 포트 RAM 내의 'j' 번째 부호가 있는 16 비트 정수 데이터 배열 요소입니다. 이 각 요소는 DPRAM 내에서 2 바이트를 차지하고 [Acc72EX[j].a] 내의 주소에 배치된 버퍼의 시작에서 $2*j$ 번째의 주소에서 시작됩니다. 이 배열은 Hilscher ComX 메모리 맵에 따라서 정의됩니다.

대괄호로 묶인 인덱스값 'j'는 0~262,143 범위의 정수 상수 또는 로컬 'L' 변수입니다. 식 또는 정수 이외의 상수는 허락되지 않습니다. DPRAM의 크기는 ACC-72EX 통신 옵션 및 설치된 Hilscher ComX 모듈에 좌우됩니다.

[Acc72EX[j].Idata16[j]]는 [Acc72EX[j].Udata8[2*j]]에서 [Acc72EX[j].Udata8[2*j+1]], [Acc72EX[j].Udata16[j]], [Acc72EX[j].Idata32[j/2]] 및 [Acc72EX[j].Udata32[j/2]]와 같은 레지스터에 배치됩니다. 같은 레지스터를 여러 용도로 사용하지 않도록 주의하십시오.

C에서 이 요소는 unsigned short 변수의 포인터로 piom+DPRCSBase 또는 piom+DPRCSBase+i*0x100000 기준 주소에서 $2*j$ 주소 오프셋하여 액세스해야 합니다(MyUdata16Ptr = (short *) piom+DPRCSBase+i*0x100000+2*j).

Acc72EX[j].Idata32[j]

설명: ACC-72EX DPRAM 부호가 있는 32 비트 정수 데이터 배열 요소

범위: $-2^{31} \dots 2^{31}-1$

단위: 주소에 좌우됨

전원 셋업 시의 기본값: 주소에 좌우됨

[Acc72EX[j].Idata32[j]]는 [Acc72EX[j]] 듀얼 포트 RAM 내의 'j' 번째 부호가 있는 32 비트 정수 데이터 배열 요소입니다. 이 각 요소는 DPRAM 내에서 4 바이트를 차지하고 [Acc72EX[j].a] 내의 주소에 배치된 버퍼의 시작에서 $4*j$ 번째의 주소에서 시작됩니다. 이 배열은 Hilscher ComX 메모리 맵에 따라서 정의됩니다.

대괄호로 묶인 인덱스값 'j'는 0~131,072 범위의 정수 상수 또는 로컬 'L' 변수입니다. 식 또는 정수 이외의 상수는 허락되지 않습니다. DPRAM의 크기는 ACC-72EX 통신 옵션 및 설치된 Hilscher ComX 모듈에 좌우됩니다.

[Acc72EX[j].Idata32[j]]는 [Acc72EX[j].Udata8[4*j]]에서 [Acc72EX[j].Udata8[4*j+3]], [Acc72EX[j].Idata16[2*j]]에서 [Acc72EX[j].Idata16[2*j+1]], [Acc72EX[j].Udata16[2*j]]에서 [Acc72EX[j].Udata16[2*j+1]] 및 [Acc72EX[j].Udata32[j]]와 같은 레지스터 내에 배치됩니다. 같은 레지스터를 여러 용도로 사용하지 않도록 주의하십시오.

C에서는 이 요소는 int 변수의 포인터로 `piom+DPRCSBase` 또는 `piom+DPRCSBase+i*0x100000` 기준 주소에서 **4*j** 주소 오프셋하여 액세스해야 합니다(`MyUdata16Ptr = (int *) piom+DPRCSBase+i*0x100000+4*j`).

Acc72EX[j].Udata8[j]

설명: ACC-72EX DPRAM 부호가 없는 8 비트 정수 데이터 배열 요소

범위: $0 \dots 2^8-1$

단위: 주소에 좌우됨

전원 셋업 시의 기본값: 주소에 좌우됨

[Acc72EX[j].Udata8[j]]는 [Acc72EX[j]] 듀얼 포트 RAM 내의 **j** 번째 부호가 없는 8 비트 정수 데이터 배열 요소입니다. 이 각 요소는 DPRAM 내에서 1 바이트를 차지하고 [Acc72EX[j].a] 내의 주소에 배치된 버퍼의 시작에서 **j** 번째의 주소에서 시작됩니다. 이 배열은 Hilscher ComX 메모리 맵에 따라서 정의됩니다.

대괄호로 묶인 인덱스값 **j**는 0~524,287 범위의 정수 상수 또는 로컬 'L' 변수입니다. 식 또는 정수 이외의 상수는 허락되지 않습니다. DPRAM의 크기는 ACC-72EX 통신 옵션 및 설치된 Hilscher ComX 모듈에 좌우됩니다.

[Acc72EX[j].Udata8[j]]는 [Acc72EX[j].ldata16[j/2]], [Acc72EX[j].Udata16[j/2]], [Acc72EX[j].ldata32[j/4]] 및 [Acc72EX[j].Udata32[j/4]]와 같은 레지스터 내에 배치됩니다. 같은 레지스터를 여러 용도로 사용하지 않도록 주의하십시오.

C에서는 이 요소는 char 변수의 포인터로 `piom+DPRCSBase` 또는 `piom+DPRCSBase+i*0x100000` 기준 주소에서 **j** 주소 오프셋하여 액세스해야 합니다(`MyUdata8Ptr = (char *) piom+DPRCSBase+i*0x100000+j`).

Acc72EX[j].Udata16[j]

설명: ACC-72EX DPRAM 부호가 없는 16 비트 정수 데이터 배열 요소

범위: $0 \dots 2^{16}-1$

단위: 주소에 좌우됨

전원 셋업 시의 기본값: 주소에 좌우됨

[Acc72EX[j].Udata16[j]]는 [Acc72EX[j]] 듀얼 포트 RAM 내의 **j** 번째 부호가 없는 16 비트 정수 데이터 배열 요소입니다. 이 각 요소는 DPRAM 내에서 2 바이트를 차지하고 [Acc72EX[j].a] 내의 주소에 배치된 버퍼의 시작에서 **2*j** 번째의 주소에서 시작됩니다. 이 배열은 Hilscher ComX 메모리 맵에 따라서 정의됩니다.

대괄호로 묶인 인덱스값 **j**는 0~262,143 범위의 정수 상수 또는 로컬 'L' 변수입니다. 식 또는 정수 이외의 상수는 허락되지 않습니다. DPRAM의 크기는 ACC-72EX 통신 옵션 및 설치된 Hilscher ComX 모듈에 좌우됩니다.

[Acc72EX[i].Udata16[j]]는 [Acc72EX[i].Udata8[2*j]]에서 [Acc72EX[i].Udata8[2*j+1]], [Acc72EX[i].Idata16[j]], [Acc72EX[i].Idata32[j/2]] 및 [Acc72EX[i].Udata32[j/2]]와 같은 레지스터 내에 배치됩니다. 같은 레지스터를 여러 용도로 사용하지 않도록 주의하십시오.

이 요소의 각 비트에 읽기 또는 쓰기 액세스할 경우에 스크립트 환경에서는 구문 [Acc72EX[i].Udata16[j].k]를 사용합니다. 여기서 **k**는 0~15 범위의 정수 상수입니다.

C 에서 이 요소는 unsigned short 변수의 포인터로 piom+DPRCSBase 또는 piom+DPRCSBase+i*0x100000 기준 주소에서 **2*j** 주소 오프셋하여 액세스해야 합니다(MyUdata16Ptr = (short *) piom+DPRCSBase+i*0x100000+2*j).

Acc72EX[i].Udata32[j]

설명: ACC-72EX DPRAM 부호가 없는 16 비트 정수 데이터 배열 요소

범위: 0 .. 2³²-1

단위: 주소에 좌우됨

전원 셋업 시의 기본값: 주소에 좌우됨

[Acc72EX[i].Udata32[j]]는 [Acc72EX[i]] 듀얼 포트 RAM 내의 **j** 번째 부호가 없는 32 비트 정수 데이터 배열 요소입니다. 이 각 요소는 DPRAM 내에서 4 바이트를 차지하고 [Acc72EX[i].a] 내의 주소에 배치된 버퍼의 시작에서 **4*j** 번째의 주소에서 시작됩니다. 이 배열은 Hilscher ComX 메모리 맵에 따라서 정의됩니다.

대괄호로 묶인 인덱스값 **j**는 0~262,143 범위의 정수 상수 또는 로컬 'L' 변수입니다. 식 또는 정수 이외의 상수는 허락되지 않습니다. DPRAM의 크기는 ACC-72EX 통신 옵션 및 설치된 Hilscher ComX 모듈에 좌우됩니다.

[Acc72EX[i].Udata32[j]]는 [Acc72EX[i].Udata8[4*j]]에서 [Acc72EX[i].Udata8[4*j+3]], [Acc72EX[i].Idata16[2*j]]에서 [Acc72EX[i].Idata16[2*j+1]], [Acc72EX[i].Udata16[2*j]]에서 [Acc72EX[i].Udata16[2*j+1]] 및 [Acc72EX[i].Idata32[j]]와 같은 레지스터 내에 배치됩니다. 같은 레지스터를 여러 용도로 사용하지 않도록 주의하십시오.

C 에서 이 요소는 unsigned int 변수의 포인터로 piom+DPRCSBase 또는 piom+DPRCSBase+i*0x100000 기준 주소에서 **4*j** 주소 오프셋하여 액세스해야 합니다(MyUdata16Ptr = (short *) piom+DPRCSBase+i*0x100000+4*j).

Acc84B[i].비저장 설정 데이터 구조체 요소

[Acc84B[i]] 데이터 구조체는 Power Brick 시스템의 시리얼 엔코더 인터페이스 보드의 레지스터에 대한 액세스를 제공합니다. 이 요소에 대해서는 동등한 [Acc84E[i]] 데이터 구조체의 항목에 나와 있습니다.

Acc84C[i].비저장 설정 데이터 구조체 요소

[Acc84C[i]] 데이터 구조체는 Compact UMAC 시스템의 시리얼 엔코더 인터페이스 보드의 레지스터에 대한 액세스를 제공합니다. 이 요소에 대해서는 동등한 [Acc84E[i]] 데이터 구조체의 항목에 나와 있습니다.

Acc84E[i].비저장 설정 데이터 구조체 요소

[Acc84E[i]] 데이터 구조체는 UMAC 시스템의 시리얼 엔코더 인터페이스 보드의 레지스터에 대한 액세스를 제공합니다. 폼 팩터가 다른 동등 인터페이스에는 같은 내용의 [Acc84C[i]] 및 [Acc84S[i]] 데이터 구조체가 있습니다. 이 섹션의 비저장 설정 데이터 구조체 요소는 옵션으로 ACC-84 보드의 2 번째 4 개의 채널을 구성합니다. 현재 이 옵션은 ACC-84C Compact UMAC 보드에서만 사용할 수 있습니다.

Acc84E[i].AuxSerialEncCtrl

설명: 2 번째의 시리얼 엔코더 구성용 액세스리 제어 워드

범위: \$0 .. \$FFFFFF

단위: 비트 필드

전원 셋업 시의 기본값: 프로토콜 고유

[Acc84E[i].AuxSerialEncCtrl]은 ACC-84x 보드상의 옵션의 2 번째 엔코더 인터페이스 IC 상의 시리얼 엔코더의 멀티채널 설정을 지정합니다. 현재 이 2 번째의 IC 를 지원하는 것은 ACC-84C Compact UMAC 보드뿐입니다.

[Acc84E[i].AuxSerialEncCtrl] 설정은 저장 설정 요소 [Acc84E[i].SerialEncCtrl]의 설정과 동등합니다. 이것은 보드상의 1 번째 엔코더 IC 의 설정을 지정합니다. 단, [AuxSerialEncCtrl]의 값은 저장되지 않기 때문에 전원 셋업/리셋을 실행할 때마다 사용자 어플리케이션을 통해 설정해야 합니다.

이 요소의 설정에 대해서는 [Acc84E[i].SerialEncCtrl]의 설명을 참조하십시오. 이 요소는 2014 년 4/4 분기에 출시된 버전 2.0 펌웨어에서 새로 도입되었습니다.

Acc84E[i].AuxChan[j].SerialEncCmd

설명: 2 번째의 시리얼 엔코더 명령용의 액세스리 채널 제어 워드

범위: \$0 .. \$FFFFFF

단위: 비트 필드

전원 셋업 시의 기본값: 프로토콜 고유

[Acc84E[i].AuxChan[j].SerialEncCmd]는 ACC-84x 보드상의 옵션인 2 번째의 엔코더 인터페이스 IC 채널의 시리얼 엔코더 인터페이스의 명령 정보를 지정합니다. 현재 이 2 번째의 IC 를 지원하는 것은 ACC-84C Compact UMAC 보드뿐입니다. 특정 프로토콜, 트리거 타이밍 및 클록 주파수는 멀티채널 요소 [Acc84E[i].AuxSerialEncCtrl]에 따라서 결정된다는 점에 주의하십시오.

[Acc84E[i].AuxChan[j].SerialEncCmd] 설정은 저장 설정 요소 [Acc84E[i].Chan[j].SerialEncCmd]의 설정과 동등합니다. 이것은 보드상의 1 번째 엔코더 IC 채널 설정을 지정합니다. 단, [Acc84E[i].AuxChan[j].SerialEncCmd]의 값은 저장되지 않기 때문에 전원 셋업/리셋을 실행할 때마다 사용자 어플리케이션을 통해서 설정해야 합니다.

이 요소의 설정에 대해서는 [Acc84E[i].Chan[j].SerialEncCmd] 설명을 참조하십시오. 이 요소는 2014 년 4/4 분기에 출시된 버전 2.0 펌웨어에서 새로 도입되었습니다.

Acc84S[i].비저장 설정 데이터 구조체 요소

[Acc84S[i]] 데이터 구조체는 Power Clipper 시스템의 시리얼 엔코더 인터페이스 보드의 레지스터에 대한 액세스를 제공합니다. 이 요소에 대해서는 동등한 [Acc84E[i]] 데이터 구조체의 항목에 나와 있습니다.

BrickAC.비저장 데이터 구조체 요소

Power Brick AC 는 인텔리전트 앰프의 Power Brick 제품군의 클래스 중 하나입니다. AC 전원 입력으로 동작하는 여러 타입의 브러시, 브러시리스 및 AC 유도 모터를 지원합니다. 이 레지스터에는 [BrickAC.] 데이터 구조체를 사용하여 액세스합니다.



참고사항

이 섹션에서 설명하는 [BrickAC.] 데이터 구조체 요소는 Power Brick AC 의 ASIC 제어 및 Status 레지스터를 형성하는 [PowerBrick[*i*].] 하드웨어 데이터 구조체 요소와는 다른 소프트웨어 요소입니다. ([PowerBrick[*i*].] 데이터 구조체는 하드웨어 데이터 구조체 [Gate3[*i*].]의 별칭입니다.)

BrickAC.멀티채널 비저장 설정 요소

Brick AC 앰프의 일부 요소는 보드상의 모든 채널에 공통입니다. 이 섹션의 비저장 설정 요소는 모든 채널에 영향을 줍니다.

BrickAC.Config

설명: 앰프 구성/초기화 제어

범위: -7 .. 1

단위: 없음

전원 셋업 시의 기본값: 0

[BrickAC.Config]는 [BrickAC.] 저장 설정 요소에 따라서 Power Brick AC 앰프의 초기화를 제어하는 Power PMAC 펌웨어의 플래그로 동작합니다. 앰프 스테이지는 전원 셋업 시에 자동으로 구성되지 않기 때문에 사용자 어플리케이션에 따라서 명시적으로 구성 프로세스를 명령하지 않으면 앰프 스테이지를 사용할 수 없습니다.

스크립트 명령에서 [BrickAC.Config]를 1로 설정하면 Power PMAC CPU 의 백그라운드 태스크로 초기화 프로세스를 시작합니다. 이 요소는 초기화 프로세스가 올바르게 완료([BrickAC.Config]의 값이 0 으로 설정됨)하거나 구성 에러가 검출([BrickAC.Config]가 음의 값으로 설정되어서 프로세스에 에러가 발생한 것을 나타냄)될 때까지 설정된 값이 유지됩니다. 발생할 가능성이 있는 에러 코드는 다음 목록과 같습니다.

에러 코드	설명
-1	할당된 값을 받아들이지 않았다. 사용자는 이 데이터 구조체에 1 또는 0 의 값만 할당할 수 있다.
-2	[BrickAC.Reset] 또는 [BrickAC.Config] 프로세스가 활성화되어 있는 동안 [BrickAC.Monitor]가 호출되었다.
-3	구성 프로세스가 호환되지 않는 하드웨어에 시행했다. Power Brick 부품 번호가 일치하는 앰프 하드웨어를 검출할 수 없었다.
-4	Power Brick 하드웨어를 검출할 수 없었다. 호환되지 않는 출력 스테이지가 검출되면 이 에러가 생성된다.
-7	구성 프로세스가 호환되지 않는 하드웨어 상에서 사용되었다. DPSGATE3 인터페이스 ASIC 를 검출할 수 없었다.

온라인 명령으로 [BrickAC.Config]를 1로 설정하면 구성이 올바르게 완료했는지 여부에 대한 응답이 텍스트로 반환되고 올바르게 완료하지 않은 경우에는 발생한 에러에 대한 응답이 텍스트로 반환됩니다.

[BrickAC.Config]의 값을 1로 설정하는 경우에는 초기화 프로세스의 성공/실패 Status를 확인할 것을 적극 권장합니다.



참고사항

전원 셋업에 따라서 구성 파라미터가 앰프 제어 회로에 읽힌 뒤의 표준적인 시스템 초기화 프로세스의 일부로 [BrickAC.Config]가 1로 설정되는 경우에 그 대신 [BrickAC.Reset]을 1로 설정할 것을 권장합니다. 이렇게 하면 구성 파라미터를 읽을 뿐만 아니라 전원 셋업 시의 과도적 상태가 원인으로 발생하는 모든 폴트가 클리어됩니다.



참고사항

[BrickAC.Config]를 1로 설정하면 앰프 구성 프로세스에 따라서 자동으로 앰프 감시 프로세스가 정지되어 구성이 완료되어도 감시 프로세스는 자동으로 재개되지 않습니다. [BrickAC.Monitor]를 다시 사용자 스크립트 어플리케이션에서 1로 설정하지 않으면 감시 프로세스를 재개할 수 없습니다.

예

```
BrickAC.Config = 1
while (BrickAC.Config > 0) {}
if (BrickAC.Config < 0)
{
    // take necessary action in case of a fault
}
// continue with script process
```

BrickAC.Monitor

설명: 앰프 상태의 감시 갱신 제어

범위: -7 .. 1

단위: 없음

전원 셋업 시의 기본값: 0

[BrickAC.Monitor]는 Power Brick AC 앰프의 상태를 감시하는 백그라운드 태스크의 실행을 제어하는 Power PMAC 펌웨어의 플래그로 동작합니다. 이 태스크는 [BrickAC.]Status 요소를 저장 설정 요소 [BrickAC.MonitorPeriod]에 따라서 설정된 일정 주기로 갱신합니다.

[BrickAC.Monitor]가 전원 셋업 시의 기본값인 0으로 설정되면 [BrickAC.] Status 요소는 갱신되지 않습니다. 이 모드에서는 해당 요소의 값은 갱신되지 않고 다음 리셋 또는 전원의 ON/OFF 까지 이전 갱신 시의 값이 유지됩니다.

스크립트 명령으로 [BrickAC.Monitor]를 1 로 설정하면 백그라운드의 [BrickAC.] Status 갱신 태스크가 [BrickAC.MonitorPeriod]에 따라서 설정된 주기로 시작됩니다. 이 요소는 사용자 어플리케이션이 값을 0 으로 설정하여 갱신 프로세스를 정지하거나 사용자 어플리케이션이 [BrickAC.Config] 또는 [BrickAC.Reset] 값을 1 로 설정하여 프로세스의 초기화 또는 리셋을 명령하거나 둘 중 하나가 실행될 때까지 설정된 값이 유지됩니다.

감시 프로세스 중에 에러가 발생하면 [BrickAC.Monitor]가 음의 값으로 설정되어 프로세스에 에러가 발생한 것이 통지됩니다. 통지될 가능성이 있는 에러는 다음 표와 같습니다. [BrickAC.Monitor]의 값을 1 로 설정하는 경우에는 초기화 프로세스 감시 성공/실패 Status 를 확인할 것을 적극 권장합니다.

에러 코드	설명
-1	할당된 값을 받아들이지 않았다. 사용자는 이 데이터 구조체에 1 또는 0 의 값만 할당할 수 있다.
-2	[BrickAC.Reset] 또는 [BrickAC.Config] 프로세스가 활성화되어 있는 동안 [BrickAC.Monitor]가 호출되었다.
-3	구성 프로세스가 호환되지 않는 하드웨어에 시행했다. Power Brick 부품 번호가 일치하는 앰프 하드웨어를 검출할 수 없었다.
-4	Power Brick 하드웨어를 검출할 수 없었다. 호환되지 않는 출력 스테이지가 검출되면 이 에러가 생성된다.
-6	팩(압축) 데이터 모드가 검출되었다([PowerBrick[j].Chan[j].PackInData] > 0). 이 에러는 감시 프로세스가 요구된 경우에만 생성됩니다.
-7	구성 프로세스가 호환되지 않는 하드웨어 상에서 사용되었다. DPSGATE3 인터페이스 ASIC 를 검출할 수 없었다.



참고사항

Power Brick AC 앰프 내의 감시 데이터는 컨트롤러의 [Gate3[j].Chan[j].AdcAmp[k]] 레지스터의 하위 비트, 현재의 피드백값 아래에 제공됩니다. 2 개의 페이지가 하나의 레지스터에 팩된 경우에는 이 데이터를 읽을 수 없습니다. 그 때문에 CPU 가 모든 레지스터를 읽을 수 있도록 [Gate3[j].Chan[j].PackInData] 및 [Gate3[j].Chan[j].PackOutData]를 0 으로 설정하여 데이터의 팩을 무효화하는 것이 중요합니다.



참고사항

앰프 구성의 갱신이나 앰프 상태의 리셋 등을 실행하기 위해서 [BrickAC.Config] 또는 [BrickAC.Reset] 중 하나가 1 로 설정되면 감시 프로세스가 자동으로 중단됩니다. 그 때문에 각각의 [BrickAC.Monitor]를 0 으로 설정합니다. 구성 또는 리셋 프로세스가 종료되어도 감시 프로세스는 자동으로 재개되지 않기 때문에 다른 프로세스 중 하나가 종료 뒤에는 명시적으로 재개해야 합니다.

예

```
BrickAC.Monitor = 1
MyEndTime = Sys.Time + 0.1           // 100 msec period
while (BrickAC.Monitor > 0 || MyEndTime < Sys.Time) {}
if (BrickAC.Monitor != 0)
{
    // take necessary action in case of a fault
}
// continue with script process
```


BrickAC.Reset

설명: 앰프 리셋/폴트 클리어 제어

범위: -7 .. 1

단위: 없음

전원 셋업 시의 기본값: 0

[BrickAC.Reset]은 Power Brick AC 앰프의 리셋 프로세스를 제어하는 Power PMAC 펌웨어의 플래그로 동작합니다. 이 리셋 프로세스는 래치된 모든 폴트를 클리어하고 [BrickAC] 저장 설정 요소에 따라서 활성 앰프 제어 회로에 구성을 로드합니다.

스크립트 명령으로 [BrickAC.Reset]을 1로 설정하면 Power PMAC CPU의 백그라운드 태스크로 리셋 프로세스가 시작됩니다. 이 값은 리셋 프로세스가 완료되고 [BrickAC.Reset]의 값이 0으로 설정되거나 에러가 발생하고 [BrickAC.Reset]의 값이 프로세스에 에러가 있는 것을 나타내는 음의 값으로 설정될 때까지 설정된 값이 유지됩니다. 에러 코드 목록의 자세한 내용에 대해서는 [BrickAC.Config]에 관한 항목을 참조하십시오.

[BrickAC.Reset]의 값을 1로 설정하는 경우에는 리셋 프로세스의 성공/실패 Status를 확인할 것을 적극 권장합니다.



참고사항

[BrickAC.Reset]을 1로 설정하면 앰프 구성 프로세스에 따라서 자동으로 앰프 감시 프로세스가 정지되어 구성이 완료되어도 감시 프로세스는 자동으로 재개되지 않습니다. 감시 프로세스를 재개하려면 [BrickAC.Monitor]를 다시 사용자 스크립트 어플리케이션에서 1로 설정해야 합니다.

예

```
BrickAC.Reset = 1
while (BrickAC.Reset > 0) {}
if (BrickAC.Reset < 0)
{
    // take necessary action in case of a fault
}
// continue with script process
```

BrickLV.비저장 데이터 구조체 요소

Power Brick LV 는 인텔리전트 앰프의 Power Brick 제품군의 클래스 중 하나입니다. 저전압 DC 입력으로 동작하는 여러 타입의 브러시, 브러시리스 및 스테핑 모터를 지원합니다. 이 레지스터에는 [BrickLV.] 데이터 구조체를 사용하여 액세스합니다.



참고사항

이 섹션에서 설명하는 [BrickLV.] 데이터 구조체 요소는 Power Brick LV 의 ASIC 제어 및 Status 레지스터를 형성하는 [PowerBrick[i].] 하드웨어 데이터 구조체 요소와 다른 소프트웨어 요소입니다. ([PowerBrick[i].] 데이터 구조체는 하드웨어 데이터 구조체 [Gate3[i].]의 별칭입니다.)

BrickLV.멀티채널 설정 요소

Brick LV 앰프의 일부 요소는 보드상의 모든 채널에 공통입니다. 이 섹션의 설정 요소는 모든 채널에 영향을 줍니다.

BrickLV.Config

설명: 앰프 구성/초기화 제어

범위: -7 .. 1

단위: 없음

전원 셋업 시의 기본값: 0

[BrickLV.Config]는 [BrickLV.] 저장 설정 요소에 따라서 Power Brick AC 앰프의 초기화를 제어하는 Power PMAC 펌웨어의 플래그로 동작합니다. 앰프 스테이지는 전원 셋업 시에 자동으로 구성되지 않기 때문에 사용자 어플리케이션에 따라서 명시적으로 구성 프로세스를 명령하지 않는 경우에 앰프 스테이지를 사용할 수 없습니다.

스크립트 명령으로 [BrickLV.Config]를 1로 설정하면 Power PMAC CPU의 백그라운드 태스크로 초기화 프로세스를 시작합니다. 이 요소는 초기화 프로세스가 올바르게 완료([BrickLV.Config]의 값이 0으로 설정됨)하거나 구성 에러가 검출([BrickLV.Config]가 음의 값으로 설정되어 프로세스에 에러가 발생한 것을 나타냄)될 때까지 설정된 값이 유지됩니다. 발생할 가능성이 있는 에러 코드는 다음 목록과 같습니다.

에러 코드	설명
-1	할당된 값을 받아들이지 않았다. 사용자는 이 데이터 구조체에 1 또는 0의 값만 할당할 수 있다.
-2	[BrickLV.Reset] 또는 [BrickLV.Config] 프로세스가 활성화되어 있는 동안 [BrickLV.Monitor]가 호출되었다.
-3	구성 프로세스가 호환되지 않는 하드웨어에 시행했다. Power Brick 부품 번호가 일치하는 앰프 하드웨어를 검출할 수 없었다.
-4	Power Brick 하드웨어를 검출할 수 없었다. 호환되지 않는 출력 스테이지가 검출되면 이 에러가 생성된다.
-7	구성 프로세스가 호환되지 않는 하드웨어 상에서 사용되었다. DPSGATE3 인터페이스 ASIC를 검출할 수 없었다.

온라인 명령으로 [BrickLV.Config]를 1로 설정하면 구성이 올바르게 완료했는지 여부에 대한 응답이 텍스트로 반환되고 올바르게 완료하지 않은 경우에는 발생한 에러에 대한 응답이 텍스트로 반환됩니다.

[BrickLV.Config]의 값을 1로 설정하는 경우에는 초기화 프로세스의 성공/실패 Status를 확인할 것을 적극 권장합니다.



참고사항

전원 셋업에 따라서 구성 파라미터가 앰프 제어 회로에 읽힌 뒤의 표준적인 시스템 초기화 프로세스의 일부로 [BrickLV.Config]가 1로 설정되는 경우에 그 대신 [BrickLV.Reset]을 1로 설정할 것을 권장합니다. 이렇게 하면 구성 파라미터를 읽을 뿐만 아니라 전원 셋업 시의 과도적 상태가 원인으로 발생하는 모든 폴트가 클리어됩니다.



참고사항

[BrickLV.Config]를 1로 설정하면 앰프 구성 프로세스에 따라서 자동으로 앰프 감시 프로세스가 정지되어 구성이 완료되어도 감시 프로세스는 자동으로 재개되지 않습니다. 감시 프로세스를 재개하려면 [BrickLV.Monitor]를 다시 사용자 스크립트 어플리케이션에서 1로 설정해야 합니다.

예

```
BrickLV.Config = 1
while (BrickLV.Config > 0) {}
if (BrickLV.Config != 0)
{
    // take necessary action in case of a fault
}
// continue with script process
```

BrickLV.Monitor

설명: 앰프 상태의 감시 갱신 제어

범위: -7 .. 1

단위: 없음

전원 셋업 시의 기본값: 0

[BrickLV.Monitor]는 Power Brick AC 앰프의 상태를 감시하는 백그라운드 태스크의 실행을 제어하는 Power PMAC 펌웨어의 플래그로 동작합니다. 이 태스크는 [BrickLV.] Status 요소를 저장 설정 요소 [BrickLVC.MonitorPeriod]에 따라서 설정된 일정 주기로 갱신합니다.

[BrickLV.Monitor]가 전원 셋업 시의 기본값인 0 으로 설정되면 [BrickLV.] Status 요소는 갱신되지 않습니다. 이 모드에서는 해당 요소의 값은 갱신되지 않고 다음 리셋 또는 전원의 ON/OFF 까지 이전 갱신 시의 값이 유지됩니다.

스크립트 명령으로 [BrickLV.Monitor]를 1 로 설정하면 백그라운드의 [BrickLV.] Status 갱신 태스크가 [BrickLV.MonitorPeriod]에 따라서 설정된 주기로 시작됩니다. 이 요소는 사용자 어플리케이션이 값을 0 으로 설정하여 갱신 프로세스를 정지하거나 사용자 어플리케이션이 [BrickLV.Config] 또는 [BrickLV.Reset]의 값을 1 로 설정하여 프로세스의 초기화 또는 리셋을 명령하거나 둘 중 하나가 실행될 때까지 설정된 값이 유지됩니다.

감시 프로세스 중에 에러가 발생하면 [BrickLV.Monitor]가 음의 값으로 설정되어 프로세스에 에러가 발생한 것이 통지됩니다. 통지될 가능성이 있는 에러는 다음 표와 같습니다. [BrickLV.Monitor]의 값을 1 로 설정하는 경우에는 초기화 프로세스 감시의 성공/실패 Status 를 확인할 것을 적극 권장합니다.

에러 코드	설명
-1	할당된 값을 받아들이지 않았다. 사용자는 이 데이터 구조체에 1 또는 0 의 값만 할당할 수 있다.
-2	[BrickLV.Reset] 또는 [BrickLV.Config] 프로세스가 활성화되어 있는 동안 [BrickLV.Monitor]가 호출되었다.
-3	구성 프로세스가 호환되지 않는 하드웨어에 시행했다. Power Brick 부품 번호가 일치하는 앰프 하드웨어를 검출할 수 없었다.
-4	Power Brick 하드웨어를 검출할 수 없었다. 호환되지 않는 출력 스테이지가 검출되면 이 에러가 생성된다.
-6	팩 데이터 모드가 검출되었다([PowerBrick[j].Chan[j].PackInData]> 0). 이 에러는 감시 프로세스가 요구된 경우에만 생성됩니다.
-7	구성 프로세스가 호환되지 않는 하드웨어 상에서 사용되었다. DPSGATE3 인터페이스 ASIC 를 검출할 수 없었다.



참고사항

Power Brick LV 앰프 내의 감시 데이터는 컨트롤러의 [Gate3[j].Chan[j].AdcAmp[k]] 레지스터의 하위 비트, 현재의 피드백값 아래에 제공됩니다. 2 개의 페이지가 하나의 레지스터에 팩된 경우에는 이 데이터를 읽을 수 없습니다. 그 때문에 CPU 가 모든 레지스터를 읽을 수 있도록 [Gate3[j].Chan[j].PackInData] 및 [Gate3[j].Chan[j].PackOutData]를 0 으로 설정하여 데이터의 팩을 무효화하는 것이 중요합니다.



참고사항

앰프 구성의 갱신이나 앰프 상태의 리셋 등을 실행하기 위해서 [BrickLV.Config] 또는 [BrickLV.Reset] 둘 중 하나가 1 로 설정되면 감시 프로세스가 자동으로 중단됩니다. 그 때문에 각각의 [BrickLV.Monitor]를 0 으로 설정합니다. 구성 또는 리셋 프로세스가 종료되어도 감시 프로세스는 자동으로 재개되지 않기 때문에 다른 프로세스 중 하나가 종료 뒤에는 명시적으로 재개해야 합니다.

예

```
BrickLV.Monitor = 1
MyEndTime = Sys.Time + 0.1           // 100 msec period
while (BrickLV.Monitor > 0 || MyEndTime < Sys.Time) {}
if (BrickLV.Monitor < 0)
{
    // take necessary action in case of a fault
}
// continue with script process
```

BrickLV.Reset

설명: 앰프 리셋/폴트 클리어 제어

범위: -7 .. 1

단위: 없음

전원 셋업 시의 기본값: 0

[BrickLV.Reset]은 Power Brick LV 앰프의 리셋 프로세스를 제어하는 Power PMAC 펌웨어의 플래그로 동작합니다. 이 리셋 프로세스는 래치된 모든 폴트를 클리어하고 [BrickLV] 저장 설정 요소에 따라서 활성 앰프 제어 회로에 구성을 로드합니다.

스크립트 명령으로 [BrickLV.Reset]을 1 로 설정하면 Power PMAC CPU 의 백그라운드 태스크로 리셋 프로세스가 시작됩니다. 이 값은 리셋 프로세스가 완료되고 [BrickLV.Reset]의 값이 0 으로 설정되거나 에러가 발생하고 [BrickLV.Reset]의 값이 프로세스에 에러가 있는 것을 나타내는 음의 값으로 설정될 때까지 설정된 값이 유지됩니다. 에러 코드 목록의 자세한 내용은 [BrickLV.Config]에 관한 항목을 참조하십시오.

[BrickLV.Reset]의 값을 1로 설정하는 경우에는 리셋 프로세스의 성공/실패 Status를 확인할 것을 적극 권장합니다.



참고사항

[BrickLV.Reset]을 1로 설정하면 앰프 구성 프로세스에 따라서 자동으로 앰프 감시 프로세스가 정지되어 구성이 완료되어도 감시 프로세스는 자동으로 재개되지 않습니다. 감시 프로세스를 재개하려면 [BrickLV.Monitor]를 다시 사용자 스크립트 어플리케이션에서 1로 설정해야 합니다.

예

```
BrickLV.Reset = 1
while (BrickLV.Reset > 0) {}
if (BrickLV.Reset < 0)
{
    // take necessary action in case of a fault
}
// continue with script process
```

Clipper[*i*].비저장 설정 데이터 구조체 요소

데이터 구조체명인 [Clipper[*i*]]는 기본적인 데이터 구조체 [Gate3[*i*]]의 스크립트 환경에 대한 별칭입니다. [Gate3[*i*]] 데이터 구조체의 항목으로 Power Clipper 내장 컨트롤러의 데이터 구조체 요소를 목록화하여 후술합니다.

CamTable[m].비저장 설정 데이터 구조체 요소

이 섹션에서는 **save** 명령으로 값이 플래시 메모리에 복사되지 않는 전자 캠 테이블 데이터 구조체의 설정 요소에 대해서 설명합니다.

CamTable[m].Enable

설명: 캠 테이블 유효화 제어

범위: 0 .. 3

단위: 비트 필드

전원 셋업 시의 기본값: 0

[CamTable[m].Enable]은 캠 테이블의 계산을 실행할지 여부를 지정합니다. 전원 셋업/리셋 시의 기본값인 0으로 설정하면 테이블의 계산은 실행되지 않습니다. 값이 1, 2 또는 3으로 설정되면 테이블의 계산이 실행됩니다. [CamTable[m].Enable]은 모든 테이블의 전원 셋업/리셋 시에 자동으로 0으로 설정됩니다. 이 때문에 테이블을 유효로 하려면 명시적인 명령이 필요합니다.

복귀 테이블에 대해서는 테이블의 동작은 제로 이외의 값 중 하나와 동일합니다. 비복귀 테이블의 경우에 지정된 제로 이외의 값에 따라서 테이블이 처음으로 유효화되었을 때 어떻게 동기화를 확립할지 결정됩니다.

비복귀 테이블의 경우에 Power PMAC 이 처음으로 [CamTable[m].Enable]의 값이 제로 이외라고 인식하면 테이블을 유효화하고 계산된 캠 위치의 값에 따라서 초기 위치에서 동기화된 위치로 이동하기 위한 방향을 결정합니다. 이 경우에 다음 3 개의 가능성이 있습니다.

- 1: 항상 양의 방향으로 이동하여 동기화함
- 2: 항상 음의 방향으로 이동하여 동기화함
- 3: 최단 거리로 동기화할 수 있는 방향으로 이동함

[CamTable[m].Enable]을 0 보다 큰 값으로 설정하면 테이블이 유효화되어 0 으로 설정하면 그 위치, 토크 및 다이렉트 출력 명령값이 이전 값인 채로 유지됩니다.

[CamTable[m].Enable]은 글로벌 저장 설정 요소 [Sys.CamEnable]이 *m* 보다 큰 값으로 설정된 경우에만 사용할 수 있습니다. 이 경우에 서보 사이클마다 [CamTable[m].Enable]이 확인됩니다.

CamTable[m].PosOffset

설명: 캠 테이블의 명령 슬레이브 위치 오프셋

범위: 부동 소수점

단위: 대상 모터의 위치 단위

전원 셋업 시의 기본값: 0.0

[CamTable[m].PosOffset]은 캠 테이블의 대상 모터 명령 위치를 지정합니다. 이것은 [CamTable[m].PosSf]를 곱해서 대상 모터의 [Motor[x].CompDesPos] 위치 레지스터에 쓴 뒤에 각 서보 사이클의 테이블 [PosData[i]] 항목에서 계산된 위치의 값에 더하는 정상 상태의 값입니다.

[CamTable[m].PosOffset]은 주로 Power PMAC 에 따라서 테이블이 유효화되었을 때 자동으로 지정하는 비복귀 테이블의 사이클을 결정하기 위해서 사용됩니다. 테이블이 유효화되면 서보 사이클에서는 Power PMAC 이 자동으로 [PosOffset]을 $N * (PosData[Nx] - PosData[0])$ 으로 설정합니다. 여기서 이전의 포인트와 최초의 포인트의 차는 테이블 사이클의 크기로 N 은 선택된 사이클 번호에 따릅니다(복귀 테이블의 경우에 N 은 항상 제로로 간주되기 때문에 [PosOffset]은 0으로 설정됩니다). 이것은 그 시점에서 요소 내에 존재하는 모든 값을 덮어씁니다(이전 테이블이 유효화되었을 때의 주기에 따라서 사용자가 지정한 오프셋 등).

[CamTable[m].Enable]을 1로 설정하고 테이블을 유효화하면 N 이 선택되기 때문에 대상 모터는 항상 양의 방향으로 움직여서 동기화를 확립합니다. 2로 설정하면 N 이 선택되고 항상 음의 방향으로 움직여서 동기화를 확립합니다. 3으로 설정하면 N 이 선택되고 항상 최단 거리의 방향으로 움직여서 동기화를 확립합니다.

지정된 서보 사이클에서 테이블 항목에서 계산된 값에 더하는 실제 값은 Status 요소 [CamTable[m].ActivePosOffset]으로 확인할 수 있습니다. [CamTable[m].PosOffset]의 값이 변경되면 [ActivePosOffset]의 값은 저장 설정 요소 [CamTable[m].SlewPosOffset]의 값에 따라서 새롭게 지정된 값에 도달할 때까지 서보 사이클마다 변화합니다. 이것으로 현재 위치 오프셋의 변경을 제어할 수 있습니다.

또 테이블이 유효할 때 [CamTable[m].PosOffset]의 값을 직접 지정하여 오프셋을 변경할 수도 있습니다. 단, 이것으로 테이블이 다시 유효화되면 앞에 설명한 바와 같이 Power PMAC 이 자동으로 [PosOffset]의 값을 설정하고 이전 설정한 값을 덮어씁니다. 일반적으로 다른 유효한 오프셋을 작성하는 경우에는 저장 설정 요소 [CamTable[m].PosBias](2014년 4/4분기 버전 V2.0 펌웨어에서 도입)를 설정하거나 JOG 이송 동작을 사용하여 궤적 모션을 명령해야 합니다.

Cid[j].비저장 설정 데이터 구조체 요소

이 섹션에서는 **save** 명령으로 값이 플래시 메모리에 복사되지 않는 카드 식별 데이터 구조체 내의 편리한 설정 요소에 대해서 설명합니다.

인덱스값 *j*는 0~63의 값을 가집니다. 이는 다음 표에 따라서 다양한 타입의 액세스리 인덱스값을 매핑합니다.

Cid[j]	Gatexx[j]	Cid[j]	Gatexx[j]	Cid[j]	Gatexx[j]	Cid[j]	Gatexx[j]
Cid[0]	{예/약 완료}	Cid[16]	{예/약 완료}	Cid[32]	{예/약 완료}	Cid[48]	{예/약 완료}
Cid[1]	{예/약 완료}	Cid[17]	{예/약 완료}	Cid[33]	{예/약 완료}	Cid[49]	{예/약 완료}
Cid[2]	Gate1[4]	Cid[18]	Gate1[8]	Cid[34]	Gate1[12]	Cid[50]	Gate1[16]
Cid[3]	Gate1[6]	Cid[19]	Gate1[10]	Cid[35]	Gate1[14]	Cid[51]	Gate1[18]
Cid[4]	Gate2[0]	Cid[20]	Gate2[1]	Cid[36]	Gate2[2]	Cid[52]	Gate2[3]
Cid[5]	Gate2[4]	Cid[21]	Gate2[5]	Cid[37]	Gate2[6]	Cid[53]	Gate2[7]
Cid[6]	Gate2[8]	Cid[22]	Gate2[9]	Cid[38]	Gate2[10]	Cid[54]	Gate2[11]
Cid[7]	Gate2[12]	Cid[23]	Gate2[13]	Cid[39]	Gate2[14]	Cid[55]	Gate2[15]
Cid[8]	Dpr[0]	Cid[24]	Dpr[2]	Cid[40]	Dpr[4]	Cid[56]	Dpr[6]
Cid[9]	Dpr[1]	Cid[25]	Dpr[3]	Cid[41]	Dpr[5]	Cid[57]	Dpr[7]
Cid[10]	Gate1[5]	Cid[26]	Gate1[9]	Cid[42]	Gate1[13]	Cid[58]	Gate1[17]
Cid[11]	Gate1[7]	Cid[27]	Gate1[11]	Cid[43]	Gate1[15]	Cid[59]	Gate1[19]
Cid[12]	Gatelo[0]	Cid[28]	Gatelo[4]	Cid[44]	Gatelo[8]	Cid[60]	Gatelo[12]
Cid[13]	Gatelo[1]	Cid[29]	Gatelo[5]	Cid[45]	Gatelo[9]	Cid[61]	Gatelo[13]
Cid[14]	Gatelo[2]	Cid[30]	Gatelo[6]	Cid[46]	Gatelo[10]	Cid[62]	Gatelo[14]
Cid[15]	Gatelo[3]	Cid[31]	Gatelo[7]	Cid[47]	Gatelo[11]	Cid[63]	Gatelo[15]

Cid[j].PartCtrl[k]

설명: 카드 식별 IC 정보

범위: \$00 .. \$FF

단위: 비트 필드

전원 셋업 시의 기본값: 카드에 좌우됨

[Cid[j].PartCtrl[k]]는 카드에 관한 설정 정보와 Status 정보 모두 다 포함된 8 비트 요소입니다. 인덱스 '*k*'는 0~3 범위의 값을 가집니다.

[PartCtrl[0]] 및 [PartCtrl[1]]의 비트 0~3 그리고 [PartCtrl[2]] 및 [PartCtrl[3]]의 비트 0~4에는 공장 출하 시에 설정된 카드 정보가 포함됩니다. Power PMAC은 [Cid[j]]의 해당 정보를 자동으로 Status 요소 [Cid[j].num], [Cid[j].opt], [Cid[j].rev] 및 [Cid[j].ven] 및 [PartNum], [PartOpt], [PartRev] 및 [PartType] 등의 액세스리 Status 요소로 읽습니다.

[PartCtrl[1]]의 비트 4는 बैं크 선택 제어 비트입니다. 이 비트에 쓰인 값에 따라 [PartCtrl[k]] 사용 시에 बैं크 0 정보 또는 बैं크 1 정보 중 어디에 액세스할지 결정됩니다.

[PartCtrl[k]]의 상위 비트의 기능은 카드에 따라 좌우됩니다.

Cid[j].PartData[k]

설명: 카드 식별 IC 정보

범위: \$00 .. \$FF

단위: 비트 필드

전원 셋업 시의 기본값: 카드에 좌우됨

[Cid[j].PartData[k]]는 카드의 설정 정보가 포함된 16 비트 요소입니다. 인덱스 'k'는 0~3 범위의 값을 가집니다.

[PartData[k]]의 비트의 기능은 카드에 따라 좌우됩니다. ACC-5E 에서 이 비트는 보드상의 디지털 I/O 의 바이트 폭 버퍼 IC 의 방향을 설정합니다.

Clipper[j].비저장 데이터 구조체 요소

데이터 구조체명인 [Clipper[j]]는 기본적인 데이터 구조체 [Gate3[j]]의 스크립트 환경에 대한 별칭입니다. [Gate3[j]] 데이터 구조체의 항목으로 Power Clipper 제어 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Coord[x].비저장 설정 데이터 구조체 요소

이 섹션에서는 **save** 명령으로 값이 플래시 메모리에 복사되지 않는 좌표계 데이터 구조체 내의 편리한 설정 요소에 대해서 설명합니다.

Coord[x].DesTimeBase

설명: 지정된 타임 베이스값

범위: 부동 소수점

단위: 밀리초

전원 셋업 시의 기본값: = Sys.ServoPeriod

[Coord[x].DesTimeBase]는 주소 지정된 저장 설정 요소 [Coord[x].pDesTimeBase]에 이 요소가 주소 지정되어(Coord[x].DesTimeBase.a) 사용이 지정된 경우에 좌표계의 사용자 타임 베이스값을 지정합니다. 이는 공장 출하 시의 주소 지정 기본값입니다.

[Coord[x].DesTimeBase] 내의 값은 좌표계에 고유의 온라인 %명령을 사용하여 설정할 수 있습니다. 이 경우에 값은 방정식에 따라서 설정됩니다.

$$Coord[x].DesTimeBase = \frac{\%value}{100} * Sys.ServoPeriod$$

예를 들어 **%50** 명령은 [Coord[x].DesTimeBase]를 [Sys.ServoPeriod]/2 로 설정합니다.

버퍼 프로그램 명령에서는 값이 [Coord[x].DesTimeBase]에 직접 할당됩니다. 예를 들어 0~15 의 수치를 생성하는 4 비트 스위치에서 타임 베이스값을 0~150%의 값으로 설정하려면 다음의 버퍼 프로그램 명령을 사용합니다.

Coord[x].DesTimeBase = Sys.ServoPeriod * SwitchValue / 10;

[Coord[x].DesTimeBase]를 직접 설정하는 경우에 **%{상수}** 온라인 명령을 사용하기 위해서 값이 0~200%로 제한되지 않습니다.

궤적에서 경과한 시간을 증가시키기 위해서 각 서보 사이클에서 사용되는 실제 타임 베이스값은 Status 요소 [Coord[x].TimeBase]에서 확인할 수 있습니다. [Coord[x].DesTimeBase] 내의 값이 이 값과 다른 경우에 [TimeBase]는 [DesTimeBase]의 값과 일치할 때까지 저장 설정 요소 [Coord[x].TimeBaseSlew]의 크기만큼 증가됩니다.

2014 년 1/4 분기에 출시된 버전 1.6 펌웨어부터 Power PMAC 은 음의 타임 베이스값을 완전히 지원하므로 모터 트레이스 버퍼를 사용하여 역행할 수 있게 되었습니다.

Coord[x].InvTimeMode

설명: 역타임 이송 속도 모드 제어

범위: 0 .. 3

단위: 열거

전원 셋업 시의 기본값: 0

비저장 설정 요소 [Coord[x].InvTimeMode]는 좌표계에서 직선 보간 모드와 원호 보간 모드가 블렌드된 동작으로 이송 속도 지정 모드를 결정합니다. 이것은 **F**(이송 속도) 프로그램 명령을 어떻게 해석할지 지정하는 데 따라 실현됩니다. 모든 모드에서 **F** 명령 내의 음의 값이 요소 [Coord[x].Tm]에 저장되고 그것이 **tm**(동작 시간) 명령이 아니라 **F** 명령에 의한 것임을 나타내는 부호가 붙습니다.

[Coord[x].InvTimeMode]의 전원 셋업 시의 기본값이 0 으로 설정된 경우에 역타임 모드가 무효가 되고 직선 보간 모드 및 원호 보간 모드의 동작 모두 **F** 프로그램 명령 내의 값이 벡터 속도로 좌표계 시간 단위당 축의 길이 단위로 해석됩니다. 이것을 일반적으로 이송 속도 모드라고 합니다.

축 단위는 좌표계의 축 정의에 따라서 결정되지만 벡터 속도 계산에 사용되는 것은 벡터 이송 속도 축뿐입니다. 이는 기본값으로 X, Y 및 Z 이지만 **frax** 명령을 사용하여 변경할 수 있습니다. 시간 단위는 저장 설정 요소 [Coord[x].FeedTime]에 따라서 설정됩니다. 이것이 기본값인 1,000 으로 설정된 경우에 단위는 초입니다. 60,000 으로 설정된 경우에 시간 단위는 분입니다. 이 모드에서는 동작의 시간은 이송 속도 축의 벡터 거리를 이전 **F** 명령으로 지정된 벡터 속도로 나눈 값으로 계산됩니다.

[Coord[x].InvTimeMode]를 1 로 설정하면 역행 시간 모드는 유효가 되고 직선 보간 모드 및 원호 보간 모드의 동작 모두 **F** 프로그램 명령 내의 값이 명령된 동작 시간의 역수로 해석됩니다. 두 동작 모드에서 속도를 축의 거리로 나눈 값으로 볼 수 있습니다. 시간 단위는 저장 설정 요소 [Coord[x].FeedTime]에 따라서 설정됩니다. 이것이 기본값인 1,000 으로 설정된 경우에 단위는 초입니다. 60,000 으로 설정된 경우에 시간 단위는 분입니다. 이 모드에서는 이전 **F** 명령으로 얻은 값을 [Coord[x].FeedTime]의 값으로 나누고 그 결과가 명령 동작 시간(밀리초)으로 사용됩니다.

[Coord[x].InvTimeMode]를 2 로 설정하면 역행 시간 모드는 유효가 되고 직선 보간 모드의 동작에 대해서 **F** 프로그램 명령 내의 값이 명령된 동작 시간의 역수로 해석됩니다. 이 동작 모드에서는 속도를 축의 거리로 나눈 값으로 볼 수 있습니다. 이 경우에 이전 **F** 명령으로 얻은 값을 [Coord[x].FeedTime]의 값으로 나누고 그 결과가 명령 동작 시간(밀리초)으로 사용됩니다.

[Coord[x].InvTimeMode]를 2 로 설정하면 원호 보간 모드의 동작에 대해 **F** 명령 내의 값은 속도를 X/Y/Z 원 반경으로 나눈 값으로 봅니다(동작 중에 반경이 변동하는 경우에는 시작 시의 반경이 사용됩니다). 이 경우에 **F** 명령 내의 값이 [Coord[x].FeedTime]으로 나뉘고 이 중간 결과에 원호 동작(라디안)에 의한 원호를 곱합니다. 이것으로 얻은 결과가 밀리초에 의한 명령 동작으로 사용됩니다.

[Coord[x].InvTimeMode]가 3 으로 설정된 경우에 동작은 2 로 설정된 경우와 같지만 원호 보간 모드 동작의 시간이 XX/YY/ZZ 원 반경에 기초합니다.

[Coord[x].InvTimeMode]가 제로와 제로 이외의 사이에서 변경되면 모드가 다음의 이송 속도 베이스 동작에 맞춰서 변경됩니다(모드가 변경되기 전에 실행되었던 **F** 명령의 경우도 포함). [Coord[x].InvTimeMode]의 설정은 직선 보간 모드 또는 원호 보간 모드의 동작 중 하나여도 **tm**(동작 시간) 명령값의 해석에 영향을 주지 않습니다. 명령 내의 값은 요소 [Coord[x].Tm]에 직접 저장됩니다.

일반적으로 역타임 모드는 RS-274 CNC 코드의 G93 코드에 탑재됩니다. 보통 G94 코드는 이송 속도 모드의 지정에 사용됩니다.

컨트롤러에 따라서는 역타임 모드 동안에 프로그램된 동작마다 새로운 **F** 명령이 필요한 경우가 있습니다. Power PMAC에서는 필수는 아니지만 사용하는 것을 적극 권장합니다. 새로운 동작에 오래된 동작 시간을 사용하면 예상할 수 없는 작업이 발생할 때가 있습니다.

역타임 모드의 좌표계에서는 공구 직경 보정 모드를 사용하지 마십시오.

[Coord[x].InvTimeMode]는 2012 년 2/4 분기에 출시된 버전 1.5 펌웨어에서 도입된 새로운 요소입니다. 기본값인 0 을 사용하면 오래된 버전의 펌웨어와 완전히 호환 운용할 수 있습니다.

예

다음 예에서는 이송 속도 및 역타임 모드의 직선 보간 모드 동작 및 원호 보간 모드 동작에서 어떻게 동작 시간이 계산되는지 나타냅니다.

```
Coord[1].FeedTime = 60000;           // Time units of minutes
frax(X,Y,Z);                         // Vector feedrate axes

Coord[1].InvTimeMode = 0;            // Feedrate mode (G94)
inc; linear; X10 Y10 Z5 F50;        // (G91 G01)
```

동작 거리= $\sqrt{10^2 + 10^2 + 5^2} = 15.00$. 동작 시간= $15.00/50 = 0.300$ 분= 18,000 밀리초

```
Coord[1].InvTimeMode = 0;            // Feedrate mode (G94)
inc; circle1; X10 Y10 Z5 I10 F50;    // (G91 G02)
```

동작 거리= $\sqrt{(10 \cdot \pi/2)^2 + 5^2} = 16.48$. 동작 시간= $16.48/50 = 0.329$ 분= 19,781 밀리초

```
Coord[1].InvTimeMode = 1;            // Inverse time mode (G93)
inc; linear; X10 Y10 Z5 F50;        // (G91 G01)
```

동작 시간= $60,000/50 = 12,000$ 밀리초

```
Coord[1].InvTimeMode = 1;            // Inverse time mode (G94)
inc; circle1; X10 Y10 Z5 I10 F50;    // (G91 G02)
```

동작 시간= $60,000/50 = 12,000$ 밀리초

```
Coord[1].InvTimeMode = 2;            // Inverse time mode (G94)
inc; circle1; X10 Y10 Z5 I10 F50;    // (G91 G02)
```

동작 시간= $(60,000/50) \cdot (\pi/2) = 18,849$ 밀리초

Coord[x].Q[i]

설명: 'Q' 변수 배열 요소

범위: 부동 소수점

단위: 사용자 정의

전원 켜짐 시의 기본값: 0

[Coord[x].Q[i]]는 좌표계의 'i' 번째 'Q' 변수 배열 요소입니다. 'Q' 변수는 좌표계 고유의 글로벌한 배정도 부동 소수점의 사용자 변수입니다. 인덱스값 'i'는 각 좌표계에서 0~8,191의 값을 가질 수 있습니다. 이 변수는 'Q' 변수에 액세스하기 위한 대체 방법을 제공합니다.

Coord[x].SegOverride

설명: 세그먼트 이송 속도 오버라이드 명령값

범위: 음의 값이 아닌 부동 소수점

단위: 실시간에 대한 비율

전원 켜짐 시의 기본값: 1.0

[Coord[x].SegOverride]는 세그먼트 또는 개략적인 보간, 세그먼트 동작의 궤적 생성 스테이지에 대한 이송 속도 오버라이드의 사용자 명령값을 지정합니다 ([Coord[x].SegMoveTime] > 0 인 경우의 **linear**, **circle** 및 **pvt**). 이것은 백분율이 아니라 정규화된 값이기 때문에 값이 1.0 인 경우에는 실시간 실행을 명령합니다.

세그먼트 동작에서는 실제 시간의 [Coord[x].SegMoveTime] 밀리초마다 Power PMAC 이 다음의 세그먼트 위치를 계산합니다. 이 때 [Coord[x].SegMoveTime]의 생성 결과와 오버라이드값에 따라서 모션 방정식의 수치 시간을 가속합니다. 예를 들어 [Coord[x].SegMoveTime]이 5.0 밀리초로 설정되어 오버라이드값이 1.5로 설정된 경우에 5.0 밀리초마다 실행되는 각 세그먼트 포인트의 계산은 모션의 방정식으로 $5.0 * 1.5 = 7.5$ 밀리초가 산출됩니다. 따라서 동작은 프로그램 속도의 150%로 실행됩니다.

명령값 [Coord[x].SegOverride]가 변경되면 실제로 세그먼트의 동작 방정식으로 진행되는 시간 계산에 사용되는 내부 값 [Coord[x].ActSegOverride]가 새로운 명령값에 도달할 때까지 각 세그먼트의 저장 설정 요소 [Coord[x].SegOverrideSlew]의 양만큼 변화합니다. 이것으로 결과의 속도가 단계적으로 변경되는 것을 방지합니다.

Coord[x].TxyzScale

설명: 변환 매트릭스 재스케일링 보정

범위: 부동 소수점

단위: 없음(비율)

전원 켜짐 시의 기본값: 0.0(무효)

[Coord[x].TxyzScale]을 사용하면 벡터 이송 속도 및 2D 공구 보정 반경의 스케일링을 좌표계의 XYZ 직교 공간의 변환 매트릭스 스케일링과 구별할 수 있습니다. 이것은 주로 X, Y 및 Z 축의 베이스(변환되지 않음) 단위에 의한 이송 속도 및 공구 반경을 유지하기 위해서 사용됩니다. 프로그램된 해당 축의 단위가 선택된 변환 매트릭스에 따라서 재스케일링된 경우에도 유지할 수 있습니다.

[Coord[x].TxyzScale]은 프로그램의 **tset{데/이/터}** 명령으로 좌표계의 변환 매트릭스가 선택되고(Status 요소 [Coord[x].Tsel] > -1) [TxyzScale]이 전원 셋업 시의 기본값 0.0 보다 큰 경우에만 사용됩니다.

이 기능이 유효한 경우에 프로그램의 **F{데/이/터}** 명령(이 기능을 사용하지 않고 변환된 축의 위치 단위를 사용)에 따라서 지정된 벡터 이송 속도의 값은 동작이 계산되기 전에 [Coord[x].TxyzScale]의 값으로 나뉩니다. 마찬가지로 프로그램의 **ccr{데/이/터}** 명령(이 기능을 사용하지 않고 변환된 축의 위치 단위를 사용)으로 지정된 2D 공구 보정 반경의 값은 보정 오프셋이 계산되기 전에 [Coord[x].TxyzScale]의 값으로 나뉩니다.

X, Y 및 Z 축의 베이스 단위에 따른 벡터 이송 속도 및 보정 반경을 유지하려면 [Coord[x].TxyzScale]을 기본축 단위에 대한 변환된 축 단위의 상대량에 설정해야 합니다. 따라서 변환된 단위가 베이스 단위보다 3 배 큰 경우에는 [TxyzScale]을 3.0 으로 설정합니다. 또 변환된 단위가 베이스 단위의 절반인 경우에는 [TxyzScale]을 0.5 으로 설정합니다.

저장 설정 요소 [Coord[x].AutoTxyzScale]을 1 로 설정하면 Power PMAC 은 자동으로 이송 속도 및 보정 반경이 좌표계의 베이스 단위로 유지되도록 [Coord[x].TxyzScale]을 계산합니다.

Coord[x].CC3Data[i].3D 공구 보정 요소

[Coord[x].CC3Data[i].]서브 구조체에는 3D 공구 직경 보정의 공구 노즈 지오메트리를 지정하는 요소가 포함됩니다.

Coord[x].CC3Data[i].CutRadius

설명: 3D 공구 보정의 공구 선단부 세그먼트의 로컬 반경

범위: 음의 값이 아닌 부동 소수점

단위: 직선적인 축의 사용자 단위

전원 셋업 시의 기본값: 0.0

[Coord[x].CC3Data[i].CutRadius]는 좌표계 **x** 의 3D 공구 직경 보정에 사용되는 공구 선단부 지오메트리를 지정합니다. 테이블의 원호 부분 반경이 인덱스 'i'로 지정됩니다. 이것은 좌표계의 기본축 X, Y 및 Z 의 단위로 나타나지만 대부분의 경우에 밀리미터 또는 인치가 됩니다. 이것은 축 단위가 변환 매트릭스로 재스케일링된 경우에도 재스케일링되지 않습니다.

공구 선단부 테이블에는 최대 16 의 원호 부분이 있으며 인덱스 'i'는 0~15 가 됩니다. 정의된 원호 부분에는 각각 [Coord[x].CC3Data[i].NdotT] 및 [Coord[x].CC3Data[i].CutRadius]의 값을 지정해야 합니다. 이 값을 입력하면 Power PMAC 은 자동으로 해당 원호 부분의 [ToolOffset] 및 [ToolRadius] 요소의 값을 설정합니다.

[Coord[x].CC3Data[i].CutRadius]에는 값 0.0 을 지정할 수 있습니다. 이것으로 크기가 제로인 원호 부분이 정의됩니다. 이 제로 값을 지정하면 공구 선단부의 각도가 순식간에 변경(코너)됩니다.

세부 내용에 대해서는 "사용자 설명서"의 'Power PMAC 의 동작 모드 궤적' 장의 '3 차원 공구 직경 보정' 섹션을 참조하십시오.

Coord[x].CC3Data[i].NdotT

설명: 3D 공구 보정의 공구 선단부 세그먼트 시작각의 코사인

범위: -1.0 .. 1.0

단위: 각도의 코사인

전원 셋업 시의 기본값: 0.0 ($i = 0 \dots 15$), 1.0 ($i = 16$)

[Coord[x].CC3Data[i].NdotT]는 좌표계 x 의 3D 공구 직경 보정에 사용되는 공구 선단부 지오메트리를 지정합니다. 테이블의 원호 부분 시작각의 코사인이 인덱스 ' i '로 지정됩니다. 이 각도는 공구 선단부에서 분리된 공구 회전축과 공구 표면에서 공구 중심을 향해서 공구 선단부에서 가장 먼 원호 부분의 반경 방향의 직선과의 각도로 정의됩니다.

이 각도의 코사인은 공구 선단부상의 해당 포인트와 파트 프로그램으로 **nxyz** 명령에 따라서 지정되는 부품 표면의 법선 벡터 **N** 과 파트 프로그램의 **txyz** 명령으로 지정되는 공구 선단부 방향 벡터 **T** 의 스칼라곱과 같기 때문에 요소명은 [NdotT]가 됩니다.

이 요소의 값은 유효한 코사인값이어야 하므로 범위는 -1.0~+1.0 이 됩니다. 스크립트 명령을 사용하여 이 범위 외의 값을 입력하면 에러로 거부됩니다. 원호 부분 i 의 해당 요소의 값은 그보다 전의 모든 원호 부분의 이 요소의 값보다 커야 합니다(인덱스값이 더 작음).

[Coord[x].CC3Data[0].NdotT]의 값은 테이블의 최초의 원호 부분의 시작각을 지정합니다. 이 요소 내의 값에 0.0 을 지정하면 공구 축 원통과 공구 선단부의 시작 부분과의 블렌드가 매끄러워집니다(탄젠트). 여기서 제로 이외의 값을 사용할 수 있으며 축과 공구 선단부의 시작점과의 각이 예각이 된다는 것을 의미합니다. 음의 값을 지정하면 축의 최대 반경보다 큰 반경의 둥근 공구 선단부가 된다는 것을 뜻합니다.

공구 선단부 테이블에는 최대 16 의 원호 부분이 있으며 인덱스 ' i '는 0~15 가 됩니다. 정의된 원호 부분에는 각각 [Coord[x].CC3Data[i].NdotT] 및 [Coord[x].CC3Data[i].CutRadius]의 값을 지정해야 합니다. 이 값을 입력하면 Power PMAC 은 자동으로 해당 원호 부분의 [ToolOffset] 및 [ToolRadius] 요소의 값을 설정합니다.

n 에 따라서 원호 부분이 정의된 테이블에서 인덱스가 ' $0 \sim n-1$ '인 경우에 [Coord[x].CC3Data[n].NdotT]를 1.0 으로 설정해야 합니다. 이 코사인 값은 원호 부분 $n-1$ 의 끝부분에서 반경 방향의 직선과 공구 회전축과의 각도에 0 도(평행)를 지정합니다. 즉 이 부분의 끝부분에 있는 공구 선단부가 회전의 중심에 대하여 수직이 되도록 지정합니다. 이렇게 하면 보통은 매끄러운 공구 선단부가 됩니다. 예리한 공구 선단부가 필요한 경우(특수한 케이스)에 마지막 원호 부분인 [Coord[x].CC3Data[i].CutRadius]의 값에 0.0 을 지정합니다.

[Coord[x].CC3Data[16].NdotT]는 마지막 원호 부분의 끝부분 각도($i = 15$)를 기본값 1.0 으로 설정하면 모든 테이블에서 적절한 끝부분을 제공합니다. 이 값은 사용자가 변경할 수 없습니다.

세부 내용에 대해서는 "사용자 설명서"의 '*Power PMAC 의 동작 모드 궤적*' 장의 '*3 차원 공구 직경 보정*' 섹션을 참조하십시오.

Coord[x].CC3Data[0].ToolOffset

설명: 3D 공구 보정의 최초 세그먼트의 축 오프셋

범위: 부동 소수점

단위: 직선적인 축의 사용자 단위

전원 셋업 시의 기본값: 0.0

[Coord[x].CC3Data[0].ToolOffset]은 좌표계 x 에서 공구 선단부 지오메트리를 3D 공구 직경 보정으로 사용하도록 지정합니다. 이는 공구 선단부의 처음에 정의된 원호 부분의 시작점부터 공구의 중심점까지의 부호가 있는 축 방향의 거리입니다. [Coord[x].CC3Data[0].ToolOffset] 및 [Coord[x].CC3Data[0].ToolRadius]에 따라서 정의되는 시작점은 공구 선단부 테이블의 다른 섹션의 참조점이 됩니다.

[Coord[x].CC3Data[0].ToolOffset]은 좌표계의 기본축 X, Y 및 Z 의 단위로 나타나지만 대부분의 경우에 밀리미터 또는 인치가 됩니다. 축 단위가 변환 매트릭스로 재스케일링된 경우에서도 재스케일링되지 않습니다. 이것이 양의 값인 경우에 테이블의 참조점은 공구 중심점의 위쪽(중심점보다 공구 선단부 쪽이 멀)이 됩니다. 음의 값인 경우에 테이블의 참조점은 공구 중심점의 아래쪽(중심점보다 공구 선단부에 가까움)이 됩니다. 3D 공구 직경 보정에서는 공구 중심점이 가공 표면에 접하도록 모션 프로그램의 경로를 설명합니다.

Power PMAC 에는 0 보다 큰 인덱스값 i 를 가지는 공구 선단부에 이어지는 원호 부분에 관한 [Coord[x].CC3Data[i].ToolOffset]의 항이 있지만 이것은 [Coord[x].CC3Data[i].NdotT] 및 [Coord[x].CC3Data[i].CutRadius]에 입력된 값에 따라서 Power PMAC 이 자동으로 계산하기 때문에 이 값을 덮어쓰지 않도록 하십시오.

세부 내용에 대해서는 "사용자 설명서"의 '*Power PMAC 의 동작 모드 궤적*' 장의 '*3 차원 공구 직경 보정*' 섹션을 참조하십시오.

Coord[x].CC3Data[0].ToolRadius

설명: 3D 공구 보정의 최초 세그먼트의 축 반경

범위: 양의 부동 소수점

단위: 직선적인 축의 사용자 단위

전원 셋업 시의 기본값: 0.0

[Coord[x].CC3Data[0].ToolRadius]는 좌표계 x 에서 공구 선단부 지오메트리를 3D 공구 직경 보정에 사용하도록 지정합니다. 이것은 공구 중심선(회전축)부터 공구 선단부의 처음에 정의된 원호 부분의 시작점까지 수직 방향의 거리입니다. 보통은 공구의 축 반경이 됩니다. [Coord[x].CC3Data[0].ToolOffset] 및 [Coord[x].CC3Data[0].ToolRadius]에 따라서 정의되는 시작점은 공구 선단부 테이블의 다른 섹션의 참조점이 됩니다.

[Coord[x].CC3Data[0].ToolRadius]는 좌표계의 베이스축 X, Y 및 Z의 단위로 나타나지만 대부분의 경우에 밀리미터 또는 인치가 됩니다. 축 단위가 변환 매트릭스 재스케일링된 경우에서도 재스케일링되지 않습니다.

Power PMAC에는 0보다 큰 인덱스값 'i'를 가지는 공구 선단부에 이어지는 원호 부분에 관한 [Coord[x].CC3Data[i].ToolRadius]의 항이 있지만 이것은 [Coord[x].CC3Data[i].NdotT] 및 [Coord[x].CC3Data[i].CutRadius]에 입력된 값에 따라서 Power PMAC이 자동으로 계산하기 때문에 이 값을 덮어쓰지 않도록 하십시오.

세부 내용에 대해서는 "사용자 설명서"의 '*Power PMAC의 동작 모드 궤적*' 장의 '*3 차원 공구 직경 보정*' 섹션을 참조하십시오.

Coord[x].Ldata.비저장 로컬 데이터 요소

[Coord[x].Ldata.] 서브 구조체에는 다양한 좌표계의 계산에 로컬라이즈된 요소가 있습니다. PLC 프로그램 및 통신 스레드에는 동일 로컬 데이터의 서브 구조체가 있습니다. 이 때문에 해당 서브 구조체는 이 장에서는 공통 [Ldata.] 섹션에 기재되었습니다.

ECAT[*j*].비저장 설정 데이터 구조체 요소

다음의 EtherCAT 데이터 구조체에는 Slave I/O 레지스터에 있는 데이터가 포함됩니다. 이 정보는 **save** 명령으로 저장할 수 없습니다.

ECAT[*j*].IOBuffer[*m*]

설명: I/O 비트 길이가 32 를 초과한 경우의 I/O 버퍼

범위: \$0 ... \$F

단위: 장치에 좌우됨

[ECAT[*j*].IO[*k*].BitLength]가 32 를 초과한 경우에 Power PMAC 은 I/O 데이터를 문자열로 처리하고 요소마다 1 바이트씩 I/O 데이터를 바이트 단위로 [ECAT[*j*].IOBuffer[*m*]]의 각 요소에 둡니다. 여기서 $m = \text{[ECAT[}j\text{].IO[}k\text{].Data}} + j$ 에서 j 는 0 부터 $(\text{[ECAT[}j\text{].IO[}k\text{].BitLength}/8 - 1)$ 의 값을 가집니다. 예를 들어 [ECAT[*j*].IO[*k*].BitLength] = 128 이고 [ECAT[*j*].IO[*k*].Data] = 1000 인 경우에 I/O 데이터는 바이트 단위로 [ECAT[*j*].IOBuffer[0]], [ECAT[*j*].IOBuffer[1]], ... [ECAT[*j*].IOBuffer[1015]]에 둡니다. 이 경우에 [ECAT[*j*].IO[*k*].Data]는 사용자 지정의 오프셋으로 사용되어 이 오프셋이 I/O 데이터를 보관하는 [ECAT[*j*].IOBuffer[*m*]] 배열 요소의 시작 인덱스를 결정합니다.

EtherCAT 주기적 I/O 의 비저장 요소

다음의 구조체는 우선 순위가 높은 인터럽트로 사용하는 I/O 장치에 관련합니다. 이 장치는 슬레이브 앰프와는 다른 카테고리 분류됩니다.

ECAT[*j*].IO[*k*].Data

설명: EtherCAT 슬레이브 장치의 주기적 I/O 데이터

범위: \$0 ... \$FFFFFFFF

단위: 장치에 좌우됨

[ECAT[*j*].IO[*k*].Data]에는 인덱스 '*i*'의 EtherCAT 네트워크의 I/O बैंक *k* 의 주기적 입력과 주기적 출력 데이터 전송의 내용이 포함됩니다. बैंक의 [ECAT[*j*].IO[*k*].Input]이 0 인 경우에 이 요소에 쓴 내용은 네트워크 갱신 사이클 때마다 बैंक의 슬레이브 장치에 복사됩니다. बैंक의 [ECAT[*j*].IO[*k*].Input]이 1 인 경우에 이 요소의 내용은 네트워크 갱신 사이클 때마다 बैंक의 슬레이브 장치에서 복사된 것입니다.

ECAT[*j*].IO[*k*].Offset

설명: EtherCAT I/O 데이터의 메모리 오프셋

범위: \$0 ... \$FFFFFFFF

단위: 바이트

이 파라미터는 프로세스 데이터 버퍼 선두에서 [ECAT[*j*].IO[*k*].Data]의 메모리 오프셋을 나타냅니다. 내부의 진단으로만 사용합니다.

EtherCAT 낮은 우선 순위 I/O 모듈의 비저장 요소

다음의 구조체는 백그라운드(우선 순위가 낮은 EtherCAT 인터럽트 등)에서 사용되는 슬레이브 I/O에 관련합니다.

ECAT[*i*].LPIO[*k*].Data

설명: EtherCAT 슬레이브 장치의 백그라운드 I/O 데이터

범위: \$0 ... \$FFFFFFFF

단위: 장치에 좌우됨

[ECAT[*j*].LPIO[*k*].Data]에는 인덱스 '*j*'의 EtherCAT 네트워크의 I/O बैंक *k*의 백그라운드(낮은 우선 순위)의 입력과 출력의 데이터 전송 내용이 포함됩니다. बैंक의 [ECAT[*j*].LPIO[*k*].Input]이 0인 경우에 이 요소에 쓴 내용은 백그라운드에서 정기적으로 बैंक의 슬레이브 장치에 복사됩니다. बैंक의 [ECAT[*j*].LPIO[*k*].Input]이 1인 경우에 이 요소의 내용은 백그라운드에서 정기적으로 बैंक의 슬레이브 장치에서 복사됩니다.

Gate1[j].(PMAC2 스타일 Servo IC)비저장 설정 데이터 구조체 요소

이 섹션에서는 **save** 명령으로 값이 플래시 메모리에 복사되지 않는 DSPGATE1 Servo ASIC의 설정 요소에 대해서 설명합니다.

IC의 인덱스값 *j*의 범위는 4~19로 할 수 있습니다(0~3은 예약 완료). 이것은 보드의 하드웨어 주소 DIP 스위치 설정에 따라서 결정됩니다. 채널 인덱스값 *j*의 범위는 0~3으로 할 수 있으며 각각 하드웨어 채널 번호 1~4를 나타냅니다.

Gate1[j].Chan[j].AmpEna

설명: IC 채널의 앰프 Enable 출력 플레그 값

범위: 0 .. 1

단위: 부울 형식

전원 셋업 시의 기본값: 0

[Gate1[j].Chan[j].AmpEna]는 채널의 앰프 Enable 출력의 상태를 제어하는 단일 비트값입니다.

[Motor[x].pAmpEnable] 및 [Motor[x].AmpEnableBit]로 지정되도록 이 출력 플레그를 모터의 자동 앰프 Enable 기능에 대하여 사용하는 경우에 보통 사용자 어플리케이션 코드에서 이 요소로 쓰지 마십시오.

이 하드웨어 출력 비트를 소프트웨어 모터 Status 비트[Motor[x].AmpEna]와 혼동하지 마십시오.

[Gate1[j].Chan[j].AmpEna]는 24 비트 요소 [Gate1[j].Chan[j].Ctrl]의 비트 14입니다(C 액세스인 경우에는 비트 22). C 언어 소프트웨어는 필요에 따라서 마스크와 시프트를 사용하여 풀 워드 요소를 사용해야 합니다.

Gate1[j].Chan[j].CompA

설명: Servo IC 채널의 비교 위치 A

범위: 0 .. 16,777,215

단위: 엔코더 카운트

전원 셋업 시의 기본값: 0

[Gate1[j].Chan[j].CompA]는 채널의 EQUn 비교 출력을 토글하여 위치 비교 이벤트가 실행되는 실제 엔코더 위치를 지정합니다. 단위는 채널의 엔코더 디코드 회로의 카운트입니다. [Gate1[j].Chan[j].EncCtrl]이 어떻게 디코드할지 정의합니다. 위치는 전원 셋업/리셋 시의 위치에 대한 상대 위치입니다. 이 엔코더가 모터의 위치 피드백 엔코더인 경우에 그 위치는 모터의 원점(제로)에서 엔코더 위치인 [Motor[x].HomePos]의 양만큼 모터 위치에서 떨어져 있습니다.

[Gate1[j].Chan[j].OneOverTEna]가 1로 설정된 경우에 하드웨어 1/T 서브 카운트 보간이 채널에 대하여 유효하고 [Gate1[j].Chan[j].TimeSinceCts]의 비트 0~11은 [CompA] 위치의 소수 카운트값을 유지합니다.

[Gate1[j].Chan[j].CompA]의 엔코더 위치를 통과하면 [Gate1[j].Chan[j].CompB]의 값은 [Gate1[j].Chan[j].CompAdd]의 크기만큼 동작의 방향으로 변경됩니다. [Gate1[j].Chan[j].CompB]의 엔코더 위치를 통과하면 [Gate1[j].Chan[j].CompA]의 값은 [Gate1[j].Chan[j].CompAdd]의 크기만큼 동작의 방향으로 변경됩니다. 이것으로 연속적 펄스 열을 출력하기 위해서 비교 위치를 자동 증가할 수 있습니다.

스크립트 환경에서는 $-8,388,608(-2^{23})$ 부터 $+16,777,215(+2^{24}-1)$ 의 값을 이 요소에 쓸 수 있습니다(이 범위 외의 값을 쓰려고 하면 요소값은 이 정규 범위의 종단에서 포화합니다). 쓰기의 경우에 값은 $-2^{23} \sim 2^{23}-1$ 범위의 부호가 있는 24 비트값 또는 $0 \sim 2^{24}-1$ 범위의 부호가 없는 24 비트값 중 하나로 처리할 수 있습니다. 한편 읽기의 경우에 항상 부호가 없는 값으로 통지되기 때문에 음의 값으로 설정된 경우에 통지되는 값은 그 값보다 2^{24} 커집니다.

C 언어 환경에서 이것은 상위 24 비트에 실제 데이터를 가지는 32 비트 요소입니다.

Gate1[j].Chan[j].CompAdd

설명: Servo IC 채널의 비교 위치 자동 증가

범위: 0 .. 16,777,215

단위: 엔코더 카운트

전원 셋업 시의 기본값: 0

[Gate1[j].Chan[j].CompAdd]는 다른 레지스터의 위치를 통과하면 자동으로 채널의 [CompA] 및 [CompB]의 비교 위치 레지스터에 더하거나 레지스터에서 빼는 값의 크기를 지정합니다(덧셈인지 뺄셈인지는 동작의 방향에 좌우됩니다). 이것으로 연속적 펄스 열을 출력하기 위해서 비교 위치를 자동 증가할 수 있습니다. 단위는 채널의 엔코더 디코드 회로의 카운트입니다. [Gate1[j].Chan[j].EncCtrl]이 어떻게 디코드할지 정의합니다.

[Gate1[j].Chan[j].CompA]의 엔코더 위치를 통과하면 [Gate1[j].Chan[j].CompB]의 값은 [Gate1[j].Chan[j].CompAdd]의 크기만큼 동작의 방향으로 변경됩니다. [Gate1[j].Chan[j].CompB]의 엔코더 위치를 통과하면 [Gate1[j].Chan[j].CompA]의 값은 [Gate1[j].Chan[j].CompAdd]의 크기만큼 동작의 방향으로 변경됩니다.

스크립트 환경에서는 $-8,388,608(-2^{23})$ 부터 $+16,777,215(+2^{24}-1)$ 의 값을 이 요소에 쓸 수 있습니다(이 범위 외의 값을 쓰려고 하면 요소값은 이 정규 범위의 종단에서 포화합니다). 쓰기의 경우에 값은 $-2^{23} \sim 2^{23}-1$ 범위의 부호가 있는 24 비트값 또는 $0 \sim 2^{24}-1$ 범위의 부호가 없는 24 비트값 중 하나로 처리할 수 있습니다. 한편 읽기의 경우에 항상 부호가 없는 값으로 통지되기 때문에 음의 값으로 설정된 경우에 통지되는 값은 그 값보다 2^{24} 커집니다.

C 언어 환경에서 이것은 상위 24 비트에 실제 데이터를 가지는 32 비트 요소입니다.

Gate1[j].Chan[j].CompB

설명: Servo IC 채널의 비교 위치 B

범위: 0 .. 16,777,215

단위: 엔코더 카운트

전원 셋업 시의 기본값: 0

[Gate1[j].Chan[j].CompB]는 채널의 EQU_n 비교 출력을 토글하여 위치 비교 이벤트가 실행되는 실제 엔코더 위치를 지정합니다. 단위는 채널의 엔코더 디코드 회로의 카운트입니다. [Gate1[j].Chan[j].EncCtrl]이 어떻게 디코드할지 정의합니다. 위치는 전원 셋업/리셋 시의 위치에 대한 상대 위치입니다. 이 엔코더가 모터의 위치 피드백 엔코더인 경우에 그 위치는 모터의 원점(제로)에서 엔코더 위치인 [Motor[x].HomePos]의 양만큼 모터 위치에서 떨어져 있습니다.

[Gate1[j].Chan[j].OneOverTEna]가 1로 설정된 경우에 하드웨어 1/T 서브 카운트 보간이 채널에 대하여 유효하고 [Gate1[j].Chan[j].TimeBetweenCts]의 비트 0~11은 [CompB] 위치의 소수 카운트값을 유지합니다.

[Gate1[j].Chan[j].CompB]의 엔코더 위치를 통과하면 [Gate1[j].Chan[j].CompA]의 값은 [Gate1[j].Chan[j].CompAdd]의 크기만큼 동작의 방향으로 변경됩니다. [Gate1[j].Chan[j].CompA]의 엔코더 위치를 통과하면 [Gate1[j].Chan[j].CompB]의 값은 [Gate1[j].Chan[j].CompAdd]의 크기만큼 동작의 방향으로 변경됩니다. 이것으로 연속적 펄스 열을 출력하기 위해서 비교 위치를 자동 증가할 수 있습니다.

스크립트 환경에서는 $-8,388,608(-2^{23})$ 부터 $+16,777,215(+2^{24}-1)$ 의 값을 이 요소로 쓸 수 있습니다(이 범위 외의 값을 쓰려고 하면 요소값은 이 정규 범위의 종단에서 포화합니다). 쓰기의 경우에 값은 $-2^{23} \sim 2^{23}-1$ 범위의 부호가 있는 24비트값 또는 $0 \sim 2^{24}-1$ 범위의 부호가 없는 24비트값 중 하나로 처리할 수 있습니다. 한편 읽기의 경우에 항상 부호가 없는 값으로 통지되기 때문에 음의 값으로 설정된 경우에 통지되는 값은 그 값보다 2^{24} 커집니다.

C언어 환경에서 이것은 상위 24비트에 실제 데이터를 가지는 32비트 요소입니다.

Gate1[j].Chan[j].Dac[k]

설명: Servo IC 채널의 D/A 컨버터 레지스터의 명령값

범위: $-8,388,608 \dots 8,388,607$

단위: 부호가 있는 24비트 DAC 단위

전원 셋업 시의 기본값: 0

[Gate1[j].Chan[j].Dac[k]]는 지정한 IC, 채널, 페이지의 D/A 컨버터 레지스터의 명령값을 지정합니다. 페이지 인덱스 'k'는 0 또는 1의 값을 가질 수 있으며 각각 채널의 A상 또는 B상에 대응합니다.

[Gate1[j].Chan[j].OutputMode]의 비트 0을 1로 설정하여 페이지가 DAC 모드 출력용으로 구성된 경우에 각 페이지 사이클에서 이 24비트값은 MSB를 선두로 시프트 아웃됩니다. 보통 n 비트 DAC의 데이터는 이 24비트 요소의 상위 n 비트에 놓입니다.

[Gate1[j].Chan[j].Dac[k]]는 PWM 명령값 [Gate1[j].Chan[j].Pwm[k]]와 레지스터를 공유합니다. 레지스터의 주소로 설정된 [Motor[x].pDac]에서 이 레지스터가 모터 명령 출력용으로 지정된 경우에 레지스터는 각 서보 사이클 또는 페이지 사이클마다 자동으로 쓰이기 때문에 범용으로 사용할 수 없습니다. 한편 그렇지 않은 경우에 레지스터는 범용으로 사용할 수 있으며 이 레지스터에 값을 쓰고 언제든지 값을 설정할 수 있습니다.

스크립트 환경에서는 정규 범위 외의 값이 이 요소에 쓰이면 요소의 값은 그 부호의 최대 정규 값에 포화합니다.

C언어 환경에서 이것은 상위 24비트에 실제 데이터를 가지는 32비트 요소입니다.

Gate1[j].Chan[j].EquWrite

설명: Servo IC 채널의 비교 초기 상태의 쓰기

범위: 0 .. 3

단위: 비트 필드

전원 셋업 시의 기본값: 0

[Gate1[j].Chan[j].EquWrite]에 따라 채널(EQUn)의 비교 출력의 현재 상태를 강제할 수 있습니다. 이 요소는 2 비트값입니다. 비트 1(값 2)은 강제하는 출력 상태입니다. 비트 0(값 1)은 강제 비트입니다. 강제 비트를 1로 설정하면 Status 비트가 출력에 놓입니다. Status 비트가 출력이 되면 강제 비트는 자동으로 0으로 클리어됩니다.

채널의 비교 출력을 1로 강제하려면 [EquWrite]를 3으로 설정합니다(이것은 2로 통지됩니다). 채널의 비교 출력을 0으로 강제하려면 [EquWrite]를 1로 설정합니다(이것은 0으로 통지됩니다). 출력값은 Status 비트 [Gate1[j].Chan[j].Equ]로 언제든지 읽을 수 있습니다.

출력값이 설정된 경우에 채널의 엔코더 위치가 [Gate1[j].Chan[j].CompA] 또는 [Gate1[j].Chan[j].CompB] 중 하나의 위치를 통과하면 출력값은 반대 상태로 토글됩니다.

[Gate1[j].Chan[j].EquWrite]는 풀 워드 요소 [Gate1[j].Chan[j].Ctrl]의 비트 11~12에 해당합니다(C에서는 32 비트 요소의 비트 19~20). C에서는 풀 워드 요소에서 액세스해야 합니다.

Gate1[j].Chan[j].Pfm

설명: Servo IC 채널의 펄스 주파수 변조 레지스터의 명령값

범위: -8,388,608 .. 8,388,607

단위: 부호가 있는 24비트 PFM 단위

전원 셋업 시의 기본값: 0

[Gate1[j].Chan[j].Pfm]은 지정한 IC와 채널의 펄스 주파수 변조 레지스터용의 명령값을 지정합니다. 각 채널은 펄스-방향 출력 신호 페어를 생성하는 단일 PFM 레지스터를 가집니다.

[Gate1[j].Chan[j].OutputMode]의 비트 1를 1로 설정하여 페이지가 PFM 모드 출력용으로 구성된 경우에 각 PFMCLK 사이클마다 [Gate1[j].Chan[j].Pfm]의 값이 24비트 어큐뮬레이터에 추가됩니다. 어큐뮬레이터가 롤 오버할 때마다 출력 펄스가 생성됩니다. 양의 방향으로 롤 오버한 경우에 방향 출력 신호는 플러스로 설정됩니다. 음의 방향으로 롤 오버한 경우에 방향 출력은 마이너스로 설정됩니다. 출력 펄스 주파수는 요소의 값에 비례합니다.

PFMCLK 신호의 주파수는 저장 설정 요소 [Gate1[j].HardwareClockCtrl]의 3 비트에 따라서 설정됩니다. 기본값 주파수는 9.83MHz 입니다. 특정 명령값의 출력 주파수는 다음과 같이 산출할 수 있습니다.

$$f_{out} = \frac{Gate1[i].Chan[j].Pfm}{16,777,216} * f_{PWMCLK}$$

PFMCLK 사이클에 대한 각 펄스의 폭은 이중 사용의 저장 설정 요소 [Gate1[j].PwmDeadTime]에 따라서 결정됩니다. IC 는 펄스 폭과 동일한 OFF 시간이 경과할 때까지 이후의 펄스 출력을 금지하기 때문에 펄스 열의 듀티 사이클이 50%를 초과하지 않습니다.

[Gate1[j].Chan[j].Pfm]은 펄스 폭 변조 명령값 [Gate1[j].Chan[j].Pwm[2]]와 레지스터를 공유합니다. 레지스터의 주소로 설정된 [Motor[x].pDac]에서 이 레지스터가 모터 명령 출력용으로 지정된 경우에 레지스터는 각 서보 사이클 또는 페이지 사이클마다 자동으로 쓰이기 때문에 범용으로 사용할 수 없습니다. 한편 그렇지 않은 경우에 레지스터는 범용으로 사용할 수 있으며 이 레지스터에 값을 쓰고 언제든지 값을 설정할 수 있습니다.

스크립트 환경에서는 정규 범위 외의 값이 이 요소에 쓰이면 요소의 값은 그 부호의 최대 정규 값에 포화합니다.

C 언어 환경에서 이것은 상위 24 비트에 실제 데이터를 가지는 32 비트 요소입니다.

Gate1[j].Chan[j].Pwm[k]

설명: Servo IC 채널의 펄스 폭 변조 레지스터의 명령값

범위: -8,388,608 .. 8,388,607

단위: 부호가 있는 24 비트 PWM 단위

전원 셋업 시의 기본값: 0

[Gate1[j].Chan[j].Pwm[k]]는 지정한 IC, 채널, 페이지의 펄스 폭 변조 레지스터용의 명령값을 지정합니다. 페이지 인덱스 'k'는 0, 1 또는 2 의 값을 가질 수 있고 각각 채널의 A 상, B 상 또는 C 상에 대응합니다.

[Gate1[j].Chan[j].OutputMode]의 비트 0(A 상과 B 상의 경우) 또는 비트 1(C 상의 경우)을 0 으로 설정하여 페이지가 PWM 모드용으로 구성된 경우에 각 페이지 사이클마다 이 값의 상위 16 비트가 페이지의 PWM 생성 회로에 자동으로 로드되고 거기서 그 값이 IC 의 실행 중인 PWM 업/다운 카운터와 디지털적으로 비교됩니다(이 요소의 하위 8 비트는 PWM 회로에서는 사용되지 않습니다).

이 PWM 카운터는 PWM 사이클마다 +[Gate1[j].PwmPeriod] +1 및 -[Gate1[j].PwmPeriod] -2 사이에서 증가됩니다. 또 페이지의 디지털 PWM 출력은 각 방향에서 카운터가 이 명령값을 통과하면 토글합니다. 페이지 PWM 신호의 최종적인 듀티 사이클은 [Gate1[j].Chan[j].Pwm[k]]의 이 값에 비례합니다. 상위 16 비트의 값이 +[Gate1[j].PwmPeriod] +1 이상인 경우(요소값은 이것의 256 배), 듀티 사이클은 100%가 됩니다(Top 신호는 항상 ON, Bottom 신호는 항상 OFF). 상위 16 비트의 값이

-[Gate1[i].PwmPeriod] -2 이하인 경우(요소값은 이것의 256 배), 듀티 사이클은 0%가 됩니다(Top 신호는 항상 OFF, Bottom 신호는 항상 ON).

[Gate1[i].Chan[j].Pwm[k]]는 $k = 0$ 또는 1 인 경우에는 D/A 컨버터 명령값 [Gate1[i].Chan[j].Dac[k]]와 $k = 2$ 인 경우에는 PFM 명령값[Gate1[i].Chan[j].Pfm]과 레지스터를 공유합니다. 레지스터의 주소로 설정된 [Motor[x].pDac]에서 이 레지스터가 모터 명령 출력용으로 지정된 경우에 레지스터는 각 서보 사이클 또는 페이즈 사이클마다 자동으로 쓰이기 때문에 범용으로 사용할 수 없습니다. 한편 그렇지 않은 경우에 레지스터는 범용으로 사용할 수 있으며 이 레지스터에 값을 쓰고 언제든지 값을 설정할 수 있습니다.

스크립트 환경에서는 정규 범위 외의 값이 이 요소에 쓰이면 요소의 값은 그 부호의 최대 정규 값에 포화합니다.

C 언어 환경에서 이것은 상위 24 비트에 실제 데이터를 가지는 32 비트 요소입니다.

Gate2[i].(PMAC2 스타일 MACRO IC)비저장 설정 데이터 구조체 요소

이 섹션에서는 **save** 명령으로 값이 플래시 메모리에 복사되지 않는 DSPGATE2 MACRO ASIC의 설정 요소에 대해서 설명합니다. IC의 인덱스값 'i' 범위는 0~15로 할 수 있습니다. 이 IC는 UMAC ACC-5E 상에 있습니다.

Gate2[i].DispData

설명: MACRO IC의 디스플레이 포트의 데이터 워드

범위: \$0 .. \$FFF

단위: 사용자 정의

전원 켜짐 시의 기본값: \$0

[Gate2[i].DispData]는 12 비트값으로 하위 8 비트는 IC의 디스플레이 포트상의 8개의 I/O 포인트 DISP0~DISP7의 상태를 나타내며 상위 4 비트는 4개의 제어행 CTRL0~CTRL3을 나타냅니다. 디스플레이 포트는 ACC-5E 보드의 JDISP 커넥터상에 있지만 4개의 CTRLn 행은 현재 어느 Power PMAC 장치에서도 사용되지 않습니다. 포트는 저장 설정 요소 [Gate2[i].DispDir] 및 [Gate2[i].DispPol]에 따라 구성되며 그 기본값은 포트를 비반전 출력으로 설정합니다.

[Gate2[i].DispDir]의 매칭 비트가 1로 설정되었기 때문에 I/O 포인트가 출력으로 설정된 경우에 [Gate2[i].DispData]의 비트값은 출력 상태를 제어합니다. [Gate2[i].DispDir]의 매칭 비트가 0으로 설정되었기 때문에 I/O 포인트가 입력으로 설정된 경우에 [Gate2[i].DispData]의 비트값은 단순히 입력 상태를 통지합니다. 이 경우에 비트에 대한 값의 쓰기는 출력 상태와 그 뒤에 통지되는 비트값에 모두 영향을 주지 않습니다.

Power PMAC의 이 포트에는 오래된 PMAC 컨트롤러에 장비된 자동 표시 기능이 없기 때문에 이 포트는 항상 범용 I/O로 사용할 수 있습니다.

C 언어 환경에서는 이것은 비트 8~19에 실제 데이터를 가지는 32 비트 요소입니다.

스크립트 환경에서는 I/O 뱅크의 각 I/O 포인트는 구문 `[Gate2[j].DispData.k]`를 사용하여 참조할 수 있습니다. **k**는 레지스터의 비트 번호를 나타내는 0~11 범위의 정수 상수입니다. 이것은 'M' 변수 포인터를 I/O 포인트에 할당할 때 특히 편리합니다. 예시는 다음과 같습니다.

ptr SolenoidOn->Gate2[0].DispData.3

Gate2[j].HighIoData

설명: MACRO IC의 I/O 포트 상위 데이터 워드

범위: \$0 .. \$FF

단위: 사용자 정의

전원 셋업 시의 기본값: \$0

[Gate2[j].HighIoData]는 IC의 32 비트 I/O 포트상의 상위 8개의 I/O 포인트의 상태를 나타내는 8비트값입니다. 이 포트는 ACC-5E 보드의 JI/O 커넥터상에 있습니다. 포트는 저장 설정 요소 [Gate2[j].HighIoMode], [Gate2[j].HighIoDir], [Gate2[j].HighIoPol]에 따라 구성되며 그 기본값은 포트를 비반전 범용 입력으로 설정합니다.

[Gate2[j].HighIoDir]의 매칭 비트가 1로 설정되었기 때문에 I/O 포인트가 출력으로 설정된 경우에 [Gate2[j].HighIoData]의 비트값은 출력 상태를 제어합니다. [Gate2[j].HighIoDir]의 매칭 비트가 0으로 설정되었기 때문에 I/O 포인트가 입력으로 설정된 경우에 [Gate2[j].HighIoData]의 비트값은 단순히 입력 상태를 통지합니다. 이 경우에 비트에 대한 값의 쓰기는 출력 상태와 그 뒤에 통지되는 비트값에 모두 영향을 주지 않습니다.

C언어 환경에서는 이것은 비트 8~15에 실제 데이터를 가지는 32비트 요소입니다.

스크립트 환경에서는 I/O 뱅크의 각 I/O 포인트는 구문 `[Gate2[j].HighIoData.k]`를 사용하여 참조할 수 있습니다. **k**는 레지스터의 비트 번호를 나타내는 0~7 범위의 정수 상수입니다. 이것은 'M' 변수 포인터를 I/O 포인트에 할당할 때 특히 편리합니다. 예시는 다음과 같습니다.

ptr ContactSwitch->Gate2[1].HighIoData.6

Gate2[j].LowIoData

설명: MACRO IC의 I/O 포트의 하위 데이터 워드

범위: \$0 .. \$FFFFFF

단위: 사용자 정의

전원 셋업 시의 기본값: \$0

[Gate2[j].LowIoData]는 IC의 32 비트 I/O 포트상의 하위 24개의 I/O 포인트의 상태를 나타내는 24비트값입니다. 이 포트는 ACC-5E 보드의 JI/O 커넥터상에 있으며 이 요소의 I/O00~I/O23을 나타냅니다. 포트는 저장 설정 요소 [Gate2[j].LowIoMode], [Gate2[j].LowIoDir], [Gate2[j].LowIoPol]에 따라 구성되며 그 기본값은 포트를 비반전 입력으로 설정합니다.

[Gate2[j].LowIoDir]의 매칭 비트가 1로 설정되었기 때문에 I/O 포인트가 출력으로 설정된 경우에 [Gate2[j].LowIoData]의 비트값은 출력 상태를 제어합니다. [Gate2[j].LowIoDir]의 매칭 비트가 0으로 설정되었기 때문에 I/O 포인트가 입력으로 설정된 경우에 [Gate2[j].LowIoData]의 비트값은 단순히 입력 상태를 통지합니다. 이 경우에 비트에 대한 값의 쓰기는 출력 상태와 그 뒤에 통지되는 비트값에 모두 영향을 주지 않습니다.

C 언어 환경에서는 이것은 비트 8~31 에 실제 데이터를 가지는 32 비트 요소입니다.

스크립트 환경에서는 I/O 뱅크의 각 I/O 포인트는 구문 `[Gate2[j].LowIoData.k]`를 사용하여 참조할 수 있습니다. *k* 는 레지스터의 비트 번호를 나타내는 0~7 범위의 정수 상수입니다. 이것은 'M' 변수 포인터를 I/O 포인트에 할당할 때 특히 편리합니다. 예시는 다음과 같습니다.

ptr OverTemp->Gate2[1].LowIoData.20

Gate2[j].Macro[j][k]

설명: MACRO 노드 *j* 의 레지스터 *k* 의 입출력 데이터

범위: $-2^{23} .. 2^{23}-1$

단위: 사용자 정의

전원 셋업 시의 기본값: \$0

`[Gate2[j].Macro[j][k]]`는 IC 의 MACRO 뱅크의 노드 *j* 의 입출력 데이터 레지스터 *k* 입니다. 데이터 레지스터 인덱스 '*k*'의 범위는 0~3 입니다. 노드 인덱스 '*j*'의 범위는 0~15 입니다. 이 뱅크는 MACRO 링의 마스터 번호를 가지며 저장 설정 요소 `[Gate2[j].MacroEnable]`의 비트 20~23 에 따라서 설정됩니다. `[Gate2[j].MacroEnable]`의 비트 *j* 가 1 로 설정된 경우에 각 페이즈 사이클마다 노드 *j* 의 4 개 모든 레지스터의 출력 데이터가 자동으로 송신되어 4 개 모든 레지스터의 입력 데이터가 수신됩니다.

이것은 스크립트 환경에서는 24 비트 요소입니다. `[Gate2[j].Macro[j][0]]`은 24 비트 모든 실제 링 데이터를 가집니다. `[Gate2[j].Macro[j][1]]`, `[Gate2[j].Macro[j][2]]`, `[Gate2[j].Macro[j][3]]`은 상위 16 비트의 실제 링 데이터만 가집니다.

입력 레지스터와 출력 레지스터는 하나의 요소를 공유한다는 점에 주의하십시오. 요소에 대한 쓰기 조작은 출력 레지스터에 액세스하고 이 값은 링을 통해서 송출됩니다. 요소에서 읽기 조작은 입력 레지스터에 액세스하고 링을 통해서 받은 값을 취득합니다. 따라서 이 요소 중 하나에 쓴 출력값을 되읽을 수 없습니다.

노드가 자동 서보 제어에 사용하는 경우에 저장 설정 요소 `[Motor[x].pDac]`는 `[Gate2[j].Macro[j][0].a]`로 설정되고, `[Motor[x].pEncCtrl]`은 높은 확률로 `[Gate2[j].Macro[j][3].a]`로 설정됩니다. 이 경우에 자동 Power PMAC 태스크는 이 레지스터에 씁니다. 또 보통 사용자 어플리케이션 코드로 이 레지스터에 쓰지 마십시오.

C 언어 환경에서 이것은 상위 24 비트 또는 16 비트에 실제 데이터를 가지는 32 비트 요소입니다.

Gate2[j].MuxData

설명: MACRO IC 의 I/O 멀티플렉서 포트의 데이터 워드

범위: \$0 .. \$FFFF

단위: 사용자 정의

전원 셋업 시의 기본값: \$0

[Gate2[*i*].MuxData]는 IC의 멀티플렉서 포트상의 16개 I/O 포인트의 상태를 나타내는 16비트값입니다. 이 포트는 ACC-5E 보드의 JTHW 커넥터상에 있으며 이 요소의 하위 바이트는 DAT0~DAT7, 상위 바이트는 SEL0~SEL7을 나타냅니다. 포트는 저장 설정 요소 [Gate2[*i*].MuxMode], [Gate2[*i*].MuxDir], [Gate2[*i*].MuxPol]에 따라 구성되며 그 기본값은 포트를 비반전 입력으로 설정합니다.

[Gate2[*i*].MuxDir]의 매칭 비트가 1로 설정되었기 때문에 I/O 포인트가 출력으로 설정된 경우에 [Gate2[*i*].MuxData]의 비트값은 출력 상태를 제어합니다. [Gate2[*i*].MuxDir]의 매칭 비트가 0으로 설정되었기 때문에 I/O 포인트가 입력으로 설정된 경우에 [Gate2[*i*].MuxData]의 비트값은 단순히 입력 상태를 통지합니다. 이 경우에 비트에 대한 값의 쓰기는 출력 상태와 그 뒤에 통지되는 비트값에 모두 영향을 주지 않습니다.

2012년 3/4 분기에 출시된 V1.5 펌웨어부터 이 포트의 I/O 포인트는 [MuxIO] 데이터 구조체에 따라서 리모트 I/O 보드의 Delta Tau 제품의 ACC-34 제품군에서 멀티플렉스 I/O를 위해서 자동으로 사용됩니다. 이 자동 기능을 사용하는 경우에 [Gate2[*i*].MuxData] 요소를 이 레지스터에 대한 수동 쓰기에 사용하지 마십시오. 또 이 요소의 읽기는 필요 없으며 보통은 사용하지 않습니다.

C언어 환경에서는 이것은 비트 8~23에 실제 데이터를 가지는 32비트 요소입니다.

스크립트 환경에서는 I/O बैं크의 각 I/O 포인트는 구문 [Gate2[*i*].MuxData.*k*]를 사용하여 참조할 수 있습니다. *k*는 레지스터의 비트 번호를 나타내는 0~15 범위의 정수 상수입니다. 이것은 'M' 변수 포인터를 I/O 포인트에 할당할 때 특히 편리합니다. 예시는 다음과 같습니다.

ptr HoldingClamp->Gate2[3].MuxData.14

Gate2[i].채널 고유의 비저장 설정 요소

이 섹션의 비저장 설정 요소는 IC 상에 선택한 서보 채널에서 입출력의 사용법을 구성하기 위해서 사용됩니다. 채널 인덱스값 *j*의 범위는 0~1로 할 수 있으며 각각 하드웨어 채널 번호 1~2를 나타냅니다.

구적 엔코더 입력은 ACC-5E 상의 핸드 휠 입력으로 전용 핀을 가집니다. C 상([Pfm], [Pwm[2]]) 출력은 펄스 출력으로 전용 핀을 가집니다. 그 외의 디지털 채널 신호는 범용 디지털 I/O 와 핀을 공유하여 저장 설정 요소 [Gate2[i].LowIoMode], [Gate2[i].HighIoMode], [Gate2[i].MuxMode]에 따라서 제어됩니다.

Gate2[i].Chan[j].AmpEna

설명: IC 채널의 앰프 Enable 출력 플래그 값

범위: 0 .. 1

단위: 부울 형식

전원 셋업 시의 기본값: 0

[Gate2[i].Chan[j].AmpEna]는 채널의 앰프 Enable 출력 상태를 제어하는 단일 비트값입니다.

[Motor[x].pAmpEnable] 및 [Motor[x].AmpEnableBit]로 지정되도록 이 출력 플래그를 모터의 자동 앰프 Enable 기능에 대하여 사용하는 경우에 보통 사용자 어플리케이션 코드에서 이 요소로 쓰지 마십시오.

이 하드웨어 출력 비트를 소프트웨어 모터 Status 비트[Motor[x].AmpEna]와 혼동하지 마십시오.

[Gate2[i].Chan[j].AmpEna]는 24 비트 요소 [Gate2[i].Chan[j].Ctrl]의 비트 14 입니다 (C 액세스인 경우에는 비트 22). C 언어 소프트웨어는 필요에 따라서 마스크와 시프트를 사용하여 풀 워드 요소를 사용해야 합니다.

Gate2[i].Chan[j].CompA

설명: MACRO IC 채널의 비교 위치 A

범위: 0 .. 16,777,215

단위: 엔코더 카운트

전원 셋업 시의 기본값: 0

[Gate2[i].Chan[j].CompA]는 채널의 EQUn 비교 출력을 토글하여 위치 비교 이벤트가 실행되는 실제 엔코더 위치를 지정합니다. 단위는 채널의 엔코더 디코드 회로의 카운트입니다. [Gate2[i].Chan[j].EncCtrl]이 어떻게 디코드할지 정의합니다. 위치는 전원 셋업/리셋 시의 위치에 대한 상대 위치입니다. 이 엔코더가 모터의 위치 피드백 엔코더인 경우에 그 위치는 모터의 원점(제로)에서 엔코더 위치인 [Motor[x].HomePos]의 양만큼 모터 위치에서 떨어져 있습니다.

[Gate2[i].Chan[j].OneOverTEna]가 1로 설정된 경우에 하드웨어 1/T 서브 카운트 보간 채널에 대하여 유효하고 [Gate2[i].Chan[j].TimeSinceCts]의 비트 0~11은 [CompA] 위치의 소수 카운트값을 유지합니다.

[Gate2[i].Chan[j].CompA]의 엔코더 위치를 통과하면 [Gate2[i].Chan[j].CompB]의 값은 [Gate2[i].Chan[j].CompAdd]의 크기만큼 동작 방향으로 변경됩니다. [Gate2[i].Chan[j].CompB]의 엔코더 위치를 통과하면 [Gate2[i].Chan[j].CompA]의 값은 [Gate2[i].Chan[j].CompAdd]의 크기만큼 동작 방향으로 변경됩니다. 이것으로 연속적 펄스 열을 출력하기 위해서 비교 위치를 자동 증가할 수 있습니다.

스크립트 환경에서는 $-8,388,608(-2^{23})$ 부터 $+16,777,215(+2^{24}-1)$ 의 값을 이 요소로 쓸 수 있습니다(이 범위 외의 값을 쓰려고 하면 요소값은 이 정규 범위의 종단에서 포화합니다). 쓰기의 경우에 값은 $-2^{23} \sim 2^{23}-1$ 범위의 부호가 있는 24비트값 또는 $0 \sim 2^{24}-1$ 범위의 부호가 없는 24비트값 중 하나로 처리할 수 있습니다. 한편 읽기의 경우에 항상 부호가 없는 값으로 통지되기 때문에 음의 값으로 설정된 경우에 통지되는 값은 그 값보다 2^{24} 커집니다.

C 언어 환경에서 이것은 상위 24비트에 실제 데이터를 가지는 32비트 요소입니다.

Gate2[i].Chan[j].CompAdd

설명: MACRO IC 채널의 비교 위치 자동 증가

범위: 0 .. 16,777,215

단위: 엔코더 카운트

전원 셋업 시의 기본값: 0

[Gate2[i].Chan[j].CompAdd]는 다른 레지스터의 위치를 통과하면 자동으로 채널의 [CompA]와 [CompB]의 비교 위치 레지스터에 더하거나 레지스터에서 빼는 값의 크기를 지정합니다(덧셈인지 뺄셈인지는 동작의 방향에 따릅니다). 이것으로 연속적 펄스 열을 출력하기 위해서 비교 위치를 자동 증가할 수 있습니다. 단위는 채널의 엔코더 디코드 회로의 카운트입니다. [Gate2[i].Chan[j].EncCtrl]이 어떻게 디코드할지 정의합니다.

[Gate2[i].Chan[j].CompA]의 엔코더 위치를 통과하면 [Gate2[i].Chan[j].CompB]의 값은 [Gate2[i].Chan[j].CompAdd]의 크기만큼 동작 방향으로 변경됩니다. [Gate2[i].Chan[j].CompB]의 엔코더 위치를 통과하면 [Gate2[i].Chan[j].CompA]의 값은 [Gate2[i].Chan[j].CompAdd]의 크기만큼 동작 방향으로 변경됩니다.

스크립트 환경에서는 $-8,388,608(-2^{23})$ 부터 $+16,777,215(+2^{24}-1)$ 의 값을 이 요소로 쓸 수 있습니다(이 범위 외의 값을 쓰려고 하면 요소값은 이 정규 범위의 종단에서 포화합니다). 쓰기의 경우에 값은 $-2^{23} \sim 2^{23}-1$ 범위의 부호가 있는 24비트값 또는 $0 \sim 2^{24}-1$ 범위의 부호가 없는 24비트값 중 하나로 처리할 수 있습니다. 한편 읽기의 경우에 항상 부호가 없는 값으로 통지되기 때문에 음의 값으로 설정된 경우에 통지되는 값은 그 값보다 2^{24} 커집니다.

C 언어 환경에서 이것은 상위 24비트에 실제 데이터를 가지는 32비트 요소입니다.

Gate2[j].Chan[j].CompB

설명: MACRO IC 채널의 비교 위치 B

범위: 0 .. 16,777,215

단위: 엔코더 카운트

전원 셋업 시의 기본값: 0

[Gate2[j].Chan[j].CompB]는 채널의 EQUn 비교 출력을 토글하여 위치 비교 이벤트가 실행되는 실제 엔코더 위치를 지정합니다. 단위는 채널의 엔코더 디코드 회로의 카운트입니다. [Gate2[j].Chan[j].EncCtrl]이 어떻게 디코드할지 정의합니다. 위치는 전원 셋업/리셋 시의 위치에 대한 상대 위치입니다. 이 엔코더가 모터의 위치 피드백 엔코더인 경우에 그 위치는 모터의 원점(제로)에서 엔코더 위치인 [Motor[x].HomePos]의 양만큼 모터 위치에서 떨어져 있습니다.

[Gate2[j].Chan[j].OneOverTEna]가 1로 설정된 경우에 하드웨어 1/T 서브 카운트 보간이 채널에 대하여 유효하고 [Gate2[j].Chan[j].TimeBetweenCts]의 비트 0~11은 [CompB] 위치의 소수 카운트값을 유지합니다.

[Gate2[j].Chan[j].CompB]의 엔코더 위치를 통과하면 [Gate2[j].Chan[j].CompA]의 값은 [Gate2[j].Chan[j].CompAdd]의 크기만큼 동작 방향으로 변경됩니다. [Gate2[j].Chan[j].CompA]의 엔코더 위치를 통과하면 [Gate2[j].Chan[j].CompB]의 값은 [Gate2[j].Chan[j].CompAdd]의 크기만큼 동작 방향으로 변경됩니다. 이것으로 연속적 펄스 열을 출력하기 위해서 비교 위치를 자동 증가할 수 있습니다.

스크립트 환경에서는 $-8,388,608(-2^{23})$ 부터 $+16,777,215(+2^{24}-1)$ 의 값을 이 요소로 쓸 수 있습니다(이 범위 외의 값을 쓰려고 하면 요소값은 이 정규 범위의 종단에서 포화합니다). 쓰기의 경우에 값은 $-2^{23} \sim 2^{23}-1$ 범위의 부호가 있는 24비트값 또는 $0 \sim 2^{24}-1$ 범위의 부호가 없는 24비트값 중 하나로 처리할 수 있습니다. 한편 읽기의 경우에 항상 부호가 없는 값으로 통지되기 때문에 음의 값으로 설정된 경우에 통지되는 값은 그 값보다 2^{24} 커집니다.

C 언어 환경에서 이것은 상위 24비트에 실제 데이터를 가지는 32비트 요소입니다.

Gate2[j].Chan[j].Dac[k]

설명: MACRO IC 채널의 D/A 컨버터 레지스터의 명령값

범위: -8,388,608 .. 8,388,607

단위: 부호가 있는 24비트 DAC 단위

전원 셋업 시의 기본값: 0

[Gate2[j].Chan[j].Dac[k]]는 지정한 IC, 채널, 페이지의 D/A 컨버터 레지스터의 명령값을 지정합니다. 페이지 인덱스 'k'는 0 또는 1의 값을 가질 수 있으며 각각 채널의 A상 또는 B상에 대응합니다.

[Gate2[j].Chan[j].OutputMode]의 비트 0을 1로 설정하여 페이지가 DAC 모드 출력용으로 구성된 경우에 각 페이지 사이클에서 이 24비트값은 MSB를 선두로 시프트 아웃됩니다. 보통 n 비트 DAC의 데이터는 이 24비트 요소의 상위 n 비트에 놓입니다.

[Gate2[j].Chan[j].Dac[k]]는 PWM 명령값[Gate2[j].Chan[j].Pwm[k]]와 레지스터를 공유합니다. 레지스터의 주소로 설정된 [Motor[x].pDac]에서 이 레지스터가 모터 명령 출력용으로 지정된 경우에 레지스터는 각 서보 사이클 또는 페이즈 사이클마다 자동으로 쓰이기 때문에 범용으로 사용할 수 없습니다. 한편 그렇지 않은 경우에 레지스터는 범용으로 사용할 수 있으며 이 레지스터에 값을 쓰고 언제든지 값을 설정할 수 있습니다.

스크립트 환경에서는 정규 범위 외의 값이 이 요소에 쓰이면 요소의 값은 그 부호의 최대 정규 값에 포화합니다.

C 언어 환경에서 이것은 상위 24 비트에 실제 데이터를 가지는 32 비트 요소입니다.

Gate2[j].Chan[j].EquWrite

설명: MACRO IC 채널의 비교 초기 상태의 쓰기

범위: 0 .. 3

단위: 비트 필드

전원 셋업 시의 기본값: 0

[Gate2[j].Chan[j].EquWrite]에 따라 채널(EQUn)의 비교 출력의 현재 상태를 강제할 수 있습니다. 이 요소는 2 비트값입니다. 비트 1(값 2)은 강제하는 출력 상태입니다. 비트 0(값 1)은 강제 비트입니다. 강제 비트를 1로 설정하면 Status 비트가 출력에 놓입니다. Status 비트가 출력이 되면 강제 비트는 자동으로 0으로 클리어됩니다.

채널의 비교 출력을 1로 강제하려면 [EquWrite]를 3으로 설정합니다(이것은 2로 통지됩니다). 채널의 비교 출력을 0으로 강제하려면 [EquWrite]를 1로 설정합니다(이것은 0으로 통지됩니다). 출력값은 Status 비트[Gate2[j].Chan[j].Equ]로 언제든지 읽을 수 있습니다.

출력값이 설정된 경우에 채널의 엔코더 위치가 [Gate2[j].Chan[j].CompA] 또는 [Gate2[j].Chan[j].CompB] 중 하나의 위치를 통과하면 출력값은 반대 상태로 토글됩니다.

[Gate2[j].Chan[j].EquWrite]는 풀 워드 요소 [Gate2[j].Chan[j].Ctrl]의 비트 11~12에 상당합니다(C에서는 32 비트 요소의 비트 19~20). C에서는 풀 워드 요소에서 액세스해야 합니다.

Gate2[j].Chan[j].Pfm

설명: MACRO IC 채널의 펄스 주파수 변조 레지스터의 명령값

범위: -8,388,608 .. 8,388,607

단위: 부호가 있는 24 비트 PFM 단위

전원 셋업 시의 기본값: 0

[Gate2[i].Chan[j].Pfm]은 지정한 IC와 채널의 펄스 주파수 변조 레지스터용의 명령값을 지정합니다. 각 채널은 펄스-방향 출력 신호 페어를 생성하는 단일 PFM 레지스터를 가집니다.

[Gate2[i].Chan[j].OutputMode]의 비트 1을 1로 설정하여 페이즈가 PFM 모드 출력용으로 구성된 경우에 각 PFMCLK 사이클마다 [Gate2[i].Chan[j].Pfm]의 값이 24비트 어큐뮬레이터에 추가됩니다. 어큐뮬레이터가 롤 오버할 때마다 출력 펄스가 생성됩니다. 양의 방향으로 롤 오버한 경우에 방향 출력 신호는 플러스로 설정됩니다. 음의 방향으로 롤 오버한 경우에 방향 출력은 마이너스로 설정됩니다. 출력 펄스 주파수는 요소의 값에 비례합니다.

PFMCLK 신호의 주파수는 저장 설정 요소 [Gate2[i].HardwareClockCtrl]의 3비트에 따라서 설정됩니다. 기본값 주파수는 9.83MHz입니다. 특정 명령값의 출력 주파수는 다음과 같이 산출할 수 있습니다.

$$f_{out} = \frac{Gate2[i].Chan[j].Pfm}{16,777,216} * f_{PWMCLK}$$

PFMCLK 사이클에 대한 각 펄스의 폭은 이중 사용의 저장 설정 요소 [Gate2[i].PwmDeadTime]에 따라서 결정됩니다. IC는 펄스 폭과 동일한 OFF 시간이 경과할 때까지 이후의 펄스 출력을 금지하기 때문에 펄스 열의 듀티 사이클이 50%를 초과하지 않습니다.

[Gate2[i].Chan[j].Pfm]은 펄스 폭 변조 명령값[Gate2[i].Chan[j].Pwm[2]]와 레지스터를 공유합니다. 레지스터의 주소로 설정된 [Motor[x].pDac]에서 이 레지스터가 모터 명령 출력용으로 지정된 경우에 레지스터는 각 서보 사이클 또는 페이즈 사이클마다 자동으로 쓰이기 때문에 범용으로 사용할 수 없습니다. 한편 그렇지 않은 경우에 레지스터는 범용으로 사용할 수 있으며 이 레지스터에 값을 쓰고 언제든지 값을 설정할 수 있습니다.

스크립트 환경에서는 정규 범위 외의 값이 이 요소에 쓰이면 요소의 값은 그 부호의 최대 정규 값에 포화합니다.

C언어 환경에서 이것은 상위 24비트에 실제 데이터를 가지는 32비트 요소입니다.

Gate2[i].Chan[j].Pwm[k]

설명: MACRO IC 채널의 펄스 폭 변조 레지스터의 명령값

범위: -8,388,608 .. 8,388,607

단위: 부호가 있는 24비트 PWM 단위

전원 셋업 시의 기본값: 0

[Gate2[i].Chan[j].Pwm[k]]는 지정한 IC, 채널, 페이즈의 펄스 폭 변조 레지스터용의 명령값을 지정합니다. 페이즈 인덱스 'k'는 0, 1 또는 2의 값을 가질 수 있고 각각 채널의 A상, B상 또는 C상에 대응합니다.

[Gate2[*i*].Chan[*j*].OutputMode]의 비트 0(A 상과 B 상인 경우) 또는 비트 1(C 상인 경우)을 0으로 설정하여 페이즈가 PWM 모드용으로 구성된 경우에 각 페이즈 사이클마다 이 값의 상위 16 비트가 페이즈의 PWM 생성 회로에 자동으로 로드되고 거기서 그 값이 IC의 실행 중인 PWM 업/다운 카운터와 디지털적으로 비교됩니다(이 요소의 하위 8 비트는 PWM 회로에서는 사용되지 않습니다).

이 PWM 카운터는 각 PWM 사이클마다 $+[Gate2[i].PwmPeriod] + 1$ 및 $-[Gate2[i].PwmPeriod] - 2$ 사이에서 증가됩니다. 또 페이즈의 디지털 PWM 출력은 각 방향에서 카운터가 이 명령값을 통과하면 토글합니다. 페이즈 PWM 신호의 최종적인 듀티 사이클은 [Gate2[*i*].Chan[*j*].Pwm[*k*]]의 이 값에 비례합니다. 상위 16 비트의 값이 $+[Gate2[i].PwmPeriod] + 1$ 이상인 경우(요소값은 이것의 256 배), 듀티 사이클은 100%가 됩니다(Top 신호는 항상 ON, Bottom 신호는 항상 OFF). 상위 16 비트의 값이 $-[Gate2[i].PwmPeriod] - 2$ 이하인 경우(요소값은 이것의 256 배), 듀티 사이클은 0%가 됩니다(Top 신호는 항상 OFF, Bottom 신호는 항상 ON).

[Gate2[*i*].Chan[*j*].Pwm[*k*]]는 $k = 0$ 또는 1 인 경우에는 D/A 컨버터 명령값 [Gate2[*i*].Chan[*j*].Dac[*k*]]와 $k = 2$ 인 경우에는 PFM 명령값 [Gate2[*i*].Chan[*j*].Pfm]과 레지스터를 공유합니다. 레지스터의 주소로 설정된 [Motor[*x*].pDac]에서 이 레지스터가 모터 명령 출력용으로 지정된 경우에 레지스터는 각 서보 사이클 또는 페이즈 사이클마다 자동으로 쓰이기 때문에 범용으로 사용할 수 없습니다. 한편 그렇지 않은 경우에 레지스터는 범용으로 사용할 수 있으며 이 레지스터에 값을 쓰고 언제든지 값을 설정할 수 있습니다.

스크립트 환경에서는 정규 범위 외의 값이 이 요소에 쓰이면 요소의 값은 그 부호의 최대 정규 값에 포화합니다.

C 언어 환경에서 이것은 상위 24 비트에 실제 데이터를 가지는 32 비트 요소입니다.

Gate3[j].(PMAC3 스타일 인터페이스 IC)비저장 설정 데이터 구조체 요소

이 섹션에서는 **save** 명령으로 값이 플래시 메모리에 복사되지 않는 DSPGATE3 머신 인터페이스 ASIC의 설정 요소에 대해서 설명합니다.

IC의 인덱스값 'j' 범위는 0~15로 할 수 있습니다. 이것은 보드의 하드웨어 주소 DIP 스위치 설정에 따라서 결정됩니다.

Gate3[j].멀티채널 비저장 설정 요소

이 섹션의 비저장 설정 요소는 서보 채널 특유의 신호 이외의 신호를 구성합니다. 범용 I/O이나 MACRO 기능을 위한 신호가 포함됩니다.

Gate3[j].GpioData[j]

설명: IC의 범용 I/O뱅크의 데이터값

범위: \$0 .. \$FFFFFFFF

단위: 비트 필드

전원 셋업 시의 기본값: \$0

[Gate3[j].GpioData[j]]는 IC의 I/O뱅크 'j'의 32개 I/O포인트의 상태를 나타내는 32비트값입니다. I/O뱅크 0은 IC상에 전용 핀을 가집니다. I/O뱅크 1, 2, 3은 전용 서보 기능과 핀을 공유합니다. 범용 I/O로 사용하려면 저장 설정 요소 [Gate3[j].GpioMode[j]]의 매칭 비트를 1로 설정해야 합니다. 어떤 뱅크에서도 범용 I/O포인트의 목적은 하드웨어 탑재와 어플리케이션에 따라서 결정됩니다.

[Gate3[j].GpioDir[j]]의 매칭 비트가 1로 설정되었기 때문에 I/O포인트가 출력으로 설정된 경우에 [Gate3[j].GpioData[j]]의 비트에 쓴 값은 출력 상태를 제어합니다. [Gate3[j].GpioData[j]]의 비트값을 되읽은 경우에 반드시 출력에 쓰인 값을 반영하지는 않는다는 점에 주의하십시오. 단, [Gate3[j].GpioData[j]]가 스크립트 환경에서 쓰인 경우에 이 같은 값이 자동으로 Status 요소 [Gate3[j].GpioOutData[j]]의 메모리 유지 레지스터에도 써지기 때문에 출력 비트값을 그 요소에서 읽을 수 있습니다.

[Gate3[j].GpioDir[j]]의 매칭 비트가 0으로 설정되었기 때문에 I/O포인트가 입력으로 설정된 경우에 [Gate3[j].GpioData[j]]의 비트값은 단순히 입력 상태를 통지합니다. 이 경우에 비트에 대한 값의 쓰기는 출력 상태와 그 뒤에 통지되는 비트값에 모두 영향을 주지 않습니다.

I/O뱅크의 각 I/O포인트는 구문 [Gate3[j].GpioData[j].k]를 사용하여 참조할 수 있습니다. k는 레지스터의 비트 번호를 나타내는 0~31범위의 정수 상수입니다. 이것은 'M' 변수 포인터를 I/O포인트에 할당할 때 특히 편리합니다. 예시는 다음과 같습니다.

ptr CoolingAirOn->Gate3[1].GpioData[0].13

Power PMAC은 실제 I/O레지스터 [Gate3[j].GpioData[j]]의 비트값을 설정하는 경우에 자동으로 유지 레지스터 [Gate3[j].GpioOutData[j]]의 출력 비트값을 사용하기 때문에 다른 출력 비트값을 실수로 변경하지 않습니다.

Gate3[j].IntCtrl

설명: IC의 인터럽트 제어/Status 레지스터

범위: \$0 .. \$0FFFFFFF

단위: 비트 필드

전원 셋업 시의 기본값: \$0

[Gate3[j].IntCtrl]은 DSPGATE3 IC의 프로그래머블 인터럽트 컨트롤러의 제어/Status 레지스터입니다. IC의 4개 서보 채널에서 위치 포착 이벤트 또는 위치 비교 이벤트에서 IC가 프로세서에 인터럽트하는 것을 허가합니다. 이 레지스터는 32비트 버스의 하위 24비트를 사용하여 3바이트로 구성됩니다.

상위 바이트(비트 16~23)는 인터럽트 유효 바이트입니다. 가능한 4개의 포착 이벤트와 4개의 비교 이벤트 중 어느 것이 인터럽트를 작성할지 제어할 수 있도록 합니다.

중간 바이트(비트 8~15)는 인터럽트 소스 바이트입니다. 이 읽기 전용 바이트에 따라 어떤 신호가 인터럽트를 트리거했는지 ISR이 확인할 수 있습니다.

하위 바이트(비트 0~7)는 인터럽트 Status 바이트입니다. 이 바이트의 비트에 1을 쓰면 대응하는 인터럽트를 소거하여 다음 인터럽트를 위해서 재작동 상태가 됩니다. 이 하위 바이트의 비트에 1을 쓰면 그 바이트에 무엇을 써도 인터럽트 소스의 바이트로 변경은 할 수 없습니다.

각 바이트 내에서 인터럽트를 작성할 수 있는 8개의 신호 각각에 대한 비트는 다음과 같이 배치됩니다.

- 비트 0: Chan[0].PosCapt(1번째의 채널 포착 플래그)
- 비트 1: Chan[1].PosCapt(2번째의 채널 포착 플래그)
- 비트 2: Chan[2].PosCapt(3번째의 채널 포착 플래그)
- 비트 3: Chan[3].PosCapt(4번째의 채널 포착 플래그)
- 비트 4: Chan[0].Equ(1번째의 채널 비교 플래그)
- 비트 5: Chan[1].Equ(2번째의 채널 비교 플래그)
- 비트 6: Chan[2].Equ(3번째의 채널 비교 플래그)
- 비트 7: Chan[3].Equ(4번째의 채널 비교 플래그)

이 레지스터는 주로 사용자가 작성하는 C 인터럽트 서비스 루틴 'CaptCompISR'과 조합하여 사용됩니다. 이 루틴의 실행은 IC의 인터럽트에 따라서 트리거됩니다. 물론 ISR(인터럽트 서비스 루틴)을 실행하기 위해서는 처음에 다른 루틴에서 해당 인터럽트 중 하나 이상을 유효하게 해야 합니다. ISR은 여러 소스가 유효가 된 경우에 인터럽트 소스 바이트를 사용하여 어떤 인터럽트 이벤트가 발생했는지 확인할 수 있습니다. 또 다음 인터럽트 이벤트의 준비를 위해서 인터럽트 Status 바이트에 씁니다.

Gate3[j].MacroOutA[j][k]

설명: MACRO 뱅크 A의 노드 j의 레지스터 k의 출력 데이터

범위: $-2^{31} .. 2^{31}-1$

단위: 사용자 정의

전원 셋업 시의 기본값: \$0

[Gate3[j].MacroOutA[j][k]]는 IC의 MACRO 뱅크 A의 노드 j 의 출력 데이터 레지스터 k 입니다. 데이터 레지스터 인덱스 ' k '의 범위는 0~3입니다. 노드 인덱스 ' j '의 범위는 0~15입니다. 이 뱅크는 MACRO 링에서 마스터 번호를 가지며 저장 설정 요소 [Gate3[j].MacroEnableA]의 비트 28~31에 따라서 설정됩니다. [Gate3[j].MacroEnableA]의 비트($j + 8$)가 1로 설정된 경우에 각 페이지 사이클마다 노드 j 의 4개 모든 레지스터의 출력 데이터가 자동으로 전송됩니다.

오리지널 MACRO 프로토콜에서는 [Gate3[j].MacroOutA[j][0]]은 상위 24비트만 링을 통해서 전송되고 [Gate3[j].MacroOutA[j][1]], [Gate3[j].MacroOutA[j][2]], [Gate3[j].MacroOutA[j][3]]은 상위 16비트만 링을 통해서 전송됩니다. 새로운 MACRO2 프로토콜에서는 4개의 노드 레지스터 모든 32비트 모두 링을 통해서 전송됩니다.

노드가 자동 서보 제어에 사용되는 경우에 저장 설정 요소 [Motor[x].pDac]은 [Gate3[j].MacroOutA[j][0].a]에 설정되고 [Motor[x].pEncCtrl]은 높은 확률로 [Gate3[j].MacroOutA[j][3].a]에 설정됩니다. 이 경우에 자동 Power PMAC 태스크는 이 레지스터에 씁니다. 또 보통 사용자 어플리케이션 코드로 이 레지스터에 쓰지 마십시오.

DSPGATE3 IC에서는 이 레지스터에 쓰인 내용을 되읽을 수 있습니다. 대응하는 입력이 다른 레지스터 [Gate3[j].MacroInA[j][k]]에 있기 때문입니다. 오래된 DSPGATE2 IC에서는 다른 출력값과 입력값이 하나의 레지스터를 공유하고 있기 때문에 레지스터에 쓰인 출력값을 되읽을 수 없습니다.

Gate3[j].MacroOutB[j][k]

설명: MACRO 뱅크 B의 노드 j 의 레지스터 k 의 출력 데이터

범위: $-2^{31} \dots 2^{31}-1$

단위: 사용자 정의

전원 셋업 시의 기본값: \$0

[Gate3[j].MacroOutB[j][k]]는 IC의 MACRO 뱅크 B의 노드 j 의 출력 데이터 레지스터 k 입니다. 데이터 레지스터 인덱스 ' k '의 범위는 0~3입니다. 노드 인덱스 ' j '의 범위는 0~15입니다. 이 뱅크는 MACRO 링에서 마스터 번호를 가지며 저장 설정 요소 [Gate3[j].MacroEnableB]의 비트 28~31에 따라서 설정됩니다. [Gate3[j].MacroEnableB]의 비트($j + 8$)가 1로 설정된 경우에 각 페이지 사이클마다 노드 j 의 4개 모든 레지스터의 출력 데이터가 자동으로 전송됩니다.

오리지널의 MACRO 프로토콜에서는 [Gate3[j].MacroOutB[j][0]]은 상위 24비트만 링을 통해서 전송되고 [Gate3[j].MacroOutB[j][1]], [Gate3[j].MacroOutB[j][2]], [Gate3[j].MacroOutB[j][3]]은 상위 16비트만 링을 통해서 전송됩니다. 새로운 MACRO2 프로토콜에서는 4개의 노드 레지스터 모든 32비트 모두 링을 통해서 전송됩니다.

노드가 자동 서보 제어에 사용하는 경우에 저장 설정 요소 [Motor[x].pDac]은 [Gate3[j].MacroOutB[j][0].a]에 설정되고 [Motor[x].pEncCtrl]은 높은 확률로 [Gate3[j].MacroOutB[j][3].a]에 설정됩니다. 이 경우에 자동 Power PMAC 태스크는 이 레지스터에 씁니다. 또 보통 사용자 어플리케이션 코드로 이 레지스터에 쓰지 마십시오.

DSPGATE3 IC에서는 이 레지스터에 쓰인 내용을 되읽을 수 있습니다. 대응하는 입력이 다른 레지스터 [Gate3[j].MacroInB[j][k]]에 있기 때문입니다. 오래된 DSPGATE2 IC에서는 다른 출력값과 입력값이 하나의 레지스터를 공유하고 있기 때문에 레지스터에 쓰인 출력값을 되읽을 수 없습니다.

Gate3[i].채널 고유의 비저장 설정 요소

이 섹션의 비저장 설정 요소는 IC 상의 선택한 서보 채널에서 입출력 사용을 구성하기 위해서 사용됩니다. 채널 인덱스값 j 의 범위는 0~3 으로 할 수 있으며 각각 하드웨어 채널 번호 1~4 를 나타냅니다.

Gate3[j].Chan[j].AmpEna

설명: IC 채널의 앰프 Enable 출력 플래그(A)의 값

범위: 0 .. 1

단위: 부울 형식

전원 셋업 시의 기본값: 0

[Gate3[j].Chan[j].AmpEna]는 채널의 출력 플래그 A의 상태를 제어하는 단일 비트값입니다. 이 출력 플래그(있는 경우)의 기능은 하드웨어 탑재에 따라 다른 것이 있지만 대부분의 경우에 채널의 앰프 Enable 출력으로 사용됩니다.

[Motor[x].pAmpEnable] 및 [Motor[x].AmpEnableBit]로 지정되도록 이 출력 플래그를 모터의 자동 앰프 Enable 기능에 대하여 사용하는 경우에 보통 사용자 어플리케이션 코드에서 이 요소로 쓰지 마십시오.

이 하드웨어 출력 비트를 소프트웨어 모터 Status 비트[Motor[x].AmpEna]와 혼동하지 마십시오.

[Gate3[j].Chan[j].AmpEna]는 풀 워드 요소 [Gate3[j].Chan[j].OutCtrl]의 비트 8 입니다. C 언어 소프트웨어는 필요에 따라서 마스크와 시프트를 사용하여 풀 워드 요소를 사용해야 합니다.

Gate3[j].Chan[j].CompA

설명: IC 채널의 비교 위치 A

범위: 0 .. $2^{32} - 1$

단위: 엔코더 카운트의 1/4,096

전원 셋업 시의 기본값: 0

[Gate3[j].Chan[j].CompA]는 채널의 EQUn 비교 출력을 토글하여 위치 비교 이벤트가 실행되는 실제 엔코더 위치를 지정합니다. 단위는 채널의 엔코더 디코드 회로의 카운트의 1/4,096 입니다(12 비트의 소수 부분을 사용). [Gate3[j].Chan[j].EncCtrl]이 어떻게 디코드할지 정의합니다. 위치는 전원 셋업/리셋 시의 위치에 대한 상대 위치입니다. 이 엔코더가 모터의 위치 피드백 엔코더인 경우에 그 위치는 모터의 원점(제로)에서 엔코더 위치인 [Motor[x].HomePos]의 양만큼 모터 위치에서 떨어져 있습니다.

엔코더의 현재(최신 서보 사이클) 위치는 Status 레지스터 [Gate3[j].Chan[j].ServoCapt]에 있습니다. [Gate3[j].Chan[j].AtanEna]가 기본값인 0 으로 설정된 경우에 이 레지스터는 8 비트의 소수 카운트값을 가지기 때문에 [CompA]와 같은 단위가 아닙니다. [AtanEna]가 1 로 설정된 경우에 [ServoCapt]는 12 비트의 소수 부분을 가지며(엔코더의 사인 ADC 값과 코사인 ADC 값의 아크탄젠트로 산출), [CompA]와 같은 단위입니다.

[Gate3[j].Chan[j].TimerMode]가 기본값인 0 으로 설정된 경우에 하드웨어 1/T 서브 카운트 보간이 채널에 대하여 유효가 되어 [Gate3[j].Chan[j].CompA]의 소수 부분 비트(0~11)가 각 엔코더 SCLK 사이클마다 1/T 회로에 따라서 자동으로 산출된 소수 카운트값과 비교됩니다. 이 경우에 Status 레지스터 [Gate3[j].Chan[j].TimerA]는 12 비트의 타이머 베이스 소수로 현재의 엔코더 위치를 유지하기 때문에 [CompA]와 같은 단위입니다.

[Gate3[j].Chan[j].CompA]의 엔코더 위치를 통과하면 [Gate3[j].Chan[j].CompB]의 값은 [Gate3[j].Chan[j].CompAdd]의 크기만큼 동작 방향으로 변경됩니다. [Gate3[j].Chan[j].CompB]의 엔코더 위치를 통과하면 [Gate2[j].Chan[j].CompA]의 값은 [Gate3[j].Chan[j].CompAdd]의 크기만큼 동작 방향으로 변경됩니다. 이것으로 연속적 펄스 열을 출력하기 위해서 비교 위치를 자동 증가할 수 있습니다.

[CompA] 요소에 쓰는 동작은 다음의 비교 이벤트 시의 자동 증가를 자동으로 유효로 합니다. [EquWrite] 요소에 쓰는 동작은 다음의 비교 이벤트 시의 자동 증가를 자동으로 무효로 합니다(단, 그 뒤의 이벤트 시에 유효가 됩니다).

스크립트 환경에서는 -2^{31} 부터 $+2^{31}-1$ 의 값을 이 요소에 쓸 수 있습니다(이 범위 외의 값을 쓰려고 하면 요소값은 이 정규 범위의 종단에서 포화합니다). 쓰기인 경우에 값은 $-2^{31} \sim 2^{31}-1$ 범위의 부호가 있는 32 비트값 또는 $0 \sim 2^{32}-1$ 범위의 부호가 없는 32 비트값 중 하나로 처리할 수 있습니다. 한편, 읽기의 경우에 항상 부호가 없는 값으로 통지되기 때문에 음의 값으로 설정된 경우에 통지되는 값은 그 값보다 2^{32} 커집니다.

Gate3[j].Chan[j].CompAdd

설명: IC 채널의 비교 위치 자동 증가

범위: $0 \dots 2^{32} - 1$

단위: 엔코더 카운트의 1/4,096

전원 셋업 시의 기본값: 0

[Gate3[j].Chan[j].CompAdd]는 다른 레지스터의 위치를 통과하면 자동으로 채널의 [CompA]와 [CompB]의 비교 위치 레지스터에 더하거나 레지스터에서 빼는 값의 크기를 지정합니다(덧셈인지 뺄셈인지는 동작의 방향에 따릅니다). 이것으로 연속적 펄스 열을 출력하기 위해서 비교 위치를 자동 증가할 수 있습니다. 단위는 채널의 엔코더 디코드 회로의 카운트의 1/4,096 분입니다(12 비트의 소수 부분). [Gate3[j].Chan[j].EncCtrl]이 어떻게 디코드할지 정의합니다.

[Gate3[j].Chan[j].CompA]의 엔코더 위치를 통과하면 [Gate3[j].Chan[j].CompB]의 값은 [Gate3[j].Chan[j].CompAdd]의 크기만큼 동작 방향으로 변경됩니다. [Gate3[j].Chan[j].CompB]의 엔코더 위치를 통과하면 [Gate3[j].Chan[j].CompA]의 값은 [Gate3[j].Chan[j].CompAdd]의 크기만큼 동작 방향으로 변경됩니다.

[Gate3[j].Chan[j].EquWrite]를 사용하여 내부 비교 상태로 값을 강제하면(현재 값과 같더라도) 그 뒤의 최초의 비교 이벤트 시의 자동 증가 조작은 금지됩니다. 즉, 최초의 [CompA]와 [CompB]의 값은 현재 위치를 끼워 넣지 않아도 된다는 것을 의미합니다. 그에 따라서 엔코더가 활성으로 카운트하는 동안 시퀀스의 지정은 쉬워지지만 동작 방향을 미리 알아야 합니다.

[CompA] 레지스터 또는 [CompB] 레지스터 중 하나의 쓰기는 최초의 후속 비교 이벤트 시에 자동 증가 조작을 유효(다시 유효)로 합니다. 그러기 위해서는 [CompA]와 [CompB]의 값이 현재 위치의 반대쪽이어야 하지만 그것은 현재 위치에서 어떤 방향으로 자동 증가 시퀀스를 허락하는 것이 됩니다.

스크립트 환경에서는 -2^{31} 부터 $+2^{31}-1$ 의 값을 이 요소에 쓸 수 있습니다(이 범위 외의 값을 쓰려고 하면 요소값은 이 정규 범위의 종단에서 포화합니다). 쓰기인 경우에 값은 $-2^{31} \sim 2^{31}-1$ 범위의 부호가 있는 32 비트값 또는 $0 \sim 2^{32}-1$ 범위의 부호가 없는 32 비트값 중 하나로 처리할 수 있습니다. 한편, 읽기의 경우에 항상 부호가 없는 값으로 통지되기 때문에 음의 값으로 설정된 경우에 통지되는 값은 그 값보다 2^{32} 커집니다.

Gate3[j].Chan[j].CompB

설명: IC 채널의 비교 위치 B

범위: $0 \dots 2^{32} - 1$

단위: 엔코더 카운트의 1/4,096

전원 셋업 시의 기본값: 0

[Gate3[j].Chan[j].CompB]는 채널의 EQUn 비교 출력을 토글하여 위치 비교 이벤트가 실행되는 실제 엔코더 위치를 지정합니다. 단위는 채널의 엔코더 디코드 회로의 카운트의 1/4,096 입니다(12 비트의 소수 부분을 사용). [Gate3[j].Chan[j].EncCtrl]이 어떻게 디코드할지 정의합니다. 위치는 전원 셋업/리셋 시의 위치에 대한 상대 위치입니다. 이 엔코더가 모터의 위치 피드백 엔코더인 경우에 그 위치는 모터의 원점(제로)에서 엔코더 위치인 [Motor[x].HomePos]의 양만큼 모터 위치에서 떨어져 있습니다.

엔코더의 현재(최신 서보 사이클) 위치는 Status 레지스터 [Gate3[j].Chan[j].ServoCapt]에 있습니다. [Gate3[j].Chan[j].AtanEna]가 기본값인 0 으로 설정된 경우에 이 레지스터는 8 비트의 소수 카운트값을 가지기 위해서 [CompB]와 같은 단위가 아닙니다. [AtanEna]가 1 로 설정된 경우에 [ServoCapt]는 12 비트의 소수 부분을 가지며(엔코더의 사인 ADC 값과 코사인 ADC 값의 아크탄젠트로 산출), [CompB]와 같은 단위입니다.

[Gate3[j].Chan[j].TimerMode]가 기본값인 0 으로 설정된 경우에 하드웨어 1/T 서브 카운트 보간이 채널에 대하여 유효가 되어 [Gate3[j].Chan[j].CompB]의 소수 부분 비트(0~11)가 각 엔코더 SCLK 사이클마다 1/T 회로에 따라서 자동으로 산출된 소수 카운트값과 비교됩니다. 이 경우에 Status 레지스터 [Gate3[j].Chan[j].TimerA]는 12 비트의 타이머 베이스 소수로 현재 엔코더 위치를 유지하기 위해서 [CompB]와 같은 단위입니다.

[Gate3[j].Chan[j].CompB]의 엔코더 위치를 통과하면 [Gate3[j].Chan[j].CompA]의 값은 [Gate3[j].Chan[j].CompAdd]의 크기만큼 동작 방향으로 변경됩니다. [Gate3[j].Chan[j].CompA]의 엔코더 위치를 통과하면 [Gate3[j].Chan[j].CompB]의 값은 [Gate3[j].Chan[j].CompAdd]의 크기만큼 동작 방향으로 변경됩니다. 이것으로 연속적 펄스 열을 출력하기 위해서 비교 위치를 자동 증가할 수 있습니다.

[CompB]요소에 쓰는 동작은 다음의 비교 이벤트 시의 자동 증가를 자동으로 유효로 합니다. [EquWrite] 요소에 쓰는 동작은 다음의 비교 이벤트 시의 자동 증가를 자동으로 무효로 합니다(단, 그 뒤의 이벤트 시에 유효가 됩니다).

스크립트 환경에서는 -2^{31} 부터 $+2^{31}-1$ 의 값을 이 요소에 쓸 수 있습니다(이 범위 외의 값을 쓰려고 하면 요소값은 이 정규 범위의 종단에서 포화합니다). 쓰기인 경우에 값은 $-2^{31} \sim 2^{31}-1$ 범위의 부호가 있는 32 비트값 또는 $0 \sim 2^{32}-1$ 범위의 부호가 없는 32 비트값 중 하나로 처리할 수 있습니다. 한편, 읽기의 경우에 항상 부호가 없는 값으로 통지되기 때문에 음의 값으로 설정된 경우에 통지되는 값은 그 값보다 2^{32} 커집니다.

Gate3[i].Chan[j].Dac[k]

설명: IC 채널의 D/A 변환기 컨버터의 명령값

범위: -2,147,483,648 .. 2,147,483,647

단위: 부호가 있는 32 비트 DAC 단위

전원 셋업 시의 기본값: 0

[Gate3[i].Chan[j].Dac[k]]는 지정한 IC, 채널, 페이지의 D/A 컨버터 레지스터의 명령값을 지정합니다. 페이지 인덱스 'i'는 0, 1 또는 2의 값을 가질 수 있고 각각 채널의 A 상, B 상 또는 C 상에 대응합니다.

[Gate3[i].Chan[j].OutputMode]의 비트 *k* 를 1로 설정하여 페이지가 DAC 모드 출력용으로 구성된 경우에 각 페이지 사이클에서 이 32 비트값은 MSB를 선두로 시프트 아웃됩니다. 보통 *n* 비트 DAC 필드의 데이터는 이 32 비트 요소의 상위 *n* 비트에 놓입니다(ACC-24E3 아날로그 앰프 인터페이스 보드의 옵션에서 사용 가능한 18 비트 DAC의 데이터는 24 비트 필드의 일부임에 주의하십시오. DAC에 따라 사용되는 이 비트는 24 비트 필드의 하위 18 비트에 있지만 24 비트 필드는 32 비트 레지스터의 상위 24 비트에 있습니다).

[Gate3[i].Chan[j].Dac[k]]는 PWM 명령값 [Gate3[i].Chan[j].Pwm[k]]와 레지스터를 공유합니다. 레지스터의 주소로 설정된 [Motor[x].pDac]에서 이 레지스터가 모터 명령 출력용으로 지정된 경우에 레지스터는 각 서보 사이클 또는 페이지 사이클마다 자동으로 쓰이기 때문에 범용으로 사용할 수 없습니다. 한편 그렇지 않은 경우에 레지스터는 범용으로 사용할 수 있으며 이 레지스터에 값을 쓰고 언제든지 값을 설정할 수 있습니다.

저장 설정 요소 [Gate3[i].Chan[j].PackOutData]가 1로 설정된 경우에 채널의 4상의 16 비트값은 이 32 비트 요소 중 2개에 팩되기 때문에 프로세서는 2회의 쓰기 조작만으로 4상 전부를 명령할 수 있습니다. 이 경우에 C 상 명령값의 상위 16 비트는 상위 16 비트에 있는 A 상 명령값과 함께 [Dac[0]] 요소의 하위 16 비트에 쓰이고 D 상 명령값의 16 비트는 상위 16 비트에 있는 B 상 명령값과 함께 [Dac[1]]의 하위 16 비트에 쓰이기 때문에 [Dac[2]]의 값은 사용되지 않습니다. 각 페이지 결과의 명령값은 이 모드에서는 16 비트에 제한되어(출력 장치의 분해능이 높은 경우에도) 하위 비트는 자동으로 0으로 설정됩니다.

스크립트 환경에서는 정규 범위 외의 값이 이 요소에 쓰이면 요소의 값은 그 부호의 최대 정규 값에 포화합니다.

Gate3[j].Chan[j].EquWrite

설명: IC 채널의 비교 초기 상태의 쓰기

범위: 0 .. 3

단위: 비트 필드

전원 셋업 시의 기본값: 0

[Gate3[j].Chan[j].EquWrite]에 따라 채널의 내부 비교 상태의 현재 값을 강제할 수 있습니다. 이 요소는 2 비트값입니다. 비트 1(값 2)은 강제하는 내부 상태의 값입니다. 비트 0(값 1)은 강제 비트입니다. 강제 비트를 1로 설정하면 Status 비트 채널의 활성 내부 값에 놓입니다. 내부 값에 놓이면 강제 비트는 자동으로 0으로 클리어됩니다.

채널의 비교 상태를 1로 강제하려면 [EquWrite]를 3으로 설정합니다(이것은 2로 통지됩니다). 채널의 비교 상태를 0으로 강제하려면 [EquWrite]를 1로 설정합니다(이것은 0으로 통지됩니다). 내부 상태의 값은 Status 비트[Gate3[j].Chan[j].Equ]에서 언제든지 읽을 수 있습니다.

채널의 이 내부 비교 상태의 값은 IC 상 4개의 채널 중 하나의 비교 출력 상태의 작성으로 사용할 수 있고 4비트의 저장 설정 요소 [Gate3[j].Chan[j].EquOutMask] 및 [Gate3[j].Chan[j].EquOutPol]에 따라서 제어됩니다(여기서는 채널 인덱스 'j'는 출력 채널을 참조한다는 점에 주의하십시오. 각 요소의 비트 번호는 제공 채널을 참조합니다).

[EquWrite] 요소에 쓰는 동작은 다음의 비교 이벤트 시의 자동 증가를 자동으로 무효로 합니다(단, 그 뒤의 이벤트 시에 유효가 됩니다). [CompA] 요소 또는 [CompB] 요소에 쓰는 동작은 다음의 비교 이벤트 시의 자동 증가를 자동으로 유효로 합니다.

내부 상태값이 설정된 경우에 채널의 엔코더 위치가 [Gate3[j].Chan[j].CompA] 또는 [Gate3[j].Chan[j].CompB] 중 하나의 위치를 통과하면 상태의 값은 반대의 값에 토글됩니다.

[Gate3[j].Chan[j].EquWrite]는 풀 워드 요소 [Gate3[j].Chan[j].OutCtrl]의 비트 06~07에 해당합니다. C에서는 풀 워드 요소에서 액세스해야 합니다.

Gate3[j].Chan[j].OutFlagB

설명: IC 채널의 출력 플래그 B의 값

범위: 0 .. 1

단위: 부울 형식

전원 셋업 시의 기본값: 0

[Gate3[j].Chan[j].OutFlagB]는 채널의 출력 플래그 B의 상태를 제어하는 단일 비트값입니다. 이 출력 플래그(있는 경우)의 기능은 하드웨어 탑재에 따라 다르지만 대부분의 경우에 채널의 브레이크 제어 출력으로 사용됩니다.

[Gate3[j].Chan[j].OutFlagB]는 풀 워드 요소 [Gate3[j].Chan[j].OutCtrl]의 비트 9입니다. C 언어 소프트웨어는 필요에 따라서 마스크와 시프트를 사용하여 풀 워드 요소를 사용해야 합니다.

Gate3[i].Chan[j].OutFlagC

설명: IC 채널의 출력 플래그 C의 값

범위: 0 .. 1

단위: 부울 형식

전원 셋업 시의 기본값: 0

[Gate3[i].Chan[j].OutFlagC]는 채널의 출력 플래그 C의 상태를 제어하는 단일 비트값입니다. 이 출력 플래그(있는 경우)의 기능은 하드웨어 탑재에 따라 다르지만 대부분의 경우에 채널의 브레이크 제어 출력으로 사용됩니다.

[Gate3[i].Chan[j].OutFlagC]는 풀 워드 요소 [Gate3[i].Chan[j].OutCtrl]의 비트 10입니다. C 언어 소프트웨어는 필요에 따라서 마스크와 시프트를 사용하여 풀 워드 요소를 사용해야 합니다.

Gate3[i].Chan[j].OutFlagD

설명: IC 채널의 출력 플래그 D의 값

범위: 0 .. 1

단위: 부울 형식

전원 셋업 시의 기본값: 0

[Gate3[i].Chan[j].OutFlagD]는 채널의 출력 플래그 B의 상태를 제어하는 단일 비트값입니다. 이 출력 플래그(있는 경우)의 기능은 하드웨어 탑재에 따라 다르지만 대부분의 경우에 채널의 브레이크 제어 출력으로 사용됩니다.

[Gate3[i].Chan[j].OutFlagD]는 풀 워드 요소 [Gate3[i].Chan[j].OutCtrl]의 비트 11입니다. C 언어 소프트웨어는 필요에 따라서 마스크와 시프트를 사용하여 풀 워드 요소를 사용해야 합니다.

Gate3[i].Chan[j].Pwm[k]

설명: IC 채널의 펄스 폭 변조 레지스터의 명령값

범위: -2,147,483,648 .. 2,147,483,647

단위: 부호가 있는 32 비트 PWM 단위

전원 셋업 시의 기본값: 0

[Gate3[i].Chan[j].Pwm[k]]는 지정한 IC, 채널, 페이지의 펄스 폭 변조 레지스터용의 명령값을 지정합니다. 페이지 인덱스 'k'는 0, 1, 2 또는 3의 값을 가질 수 있고 각각 채널의 A, B, C 또는 D 상에 대응합니다. 이것은 32 비트 레지스터이지만 상위 16 비트만 실제로 PWM 생성으로 사용됩니다.

저장 설정 요소 `[Gate3[j].Chan[j].PackOutData]`의 값이 1로 설정된 경우에 4상의 16비트값은 이 32비트 요소 중 2개에 팩되기 때문에 프로세서는 2회의 쓰기 조작만으로 4상 전부를 명령할 수 있습니다. 이 경우에 C상의 16비트 명령값은 상위 16비트에 있는 A상의 명령값과 함께 `[Pwm[0]]`의 하위 16비트에 쓰이고 D상의 16비트 명령값은 상위 16비트에 있는 B상의 명령값과 함께 `[Pwm[1]]`의 하위 16비트에 쓰이기 때문에 `[Pwm[2]]`와 `[Pwm[3]]`은 사용되지 않습니다.

`[Gate3[j].Chan[j].OutputMode]`의 비트 k 를 0으로 설정하여 페이지가 PWM 모드용으로 구성되어 있는 경우에 각 페이지 사이클에서 이 요소의 값은 페이지의 PWM 생성 회로로 자동으로 읽히고 필요에 따라서 언팩됩니다. 활성 16비트가 각각 개별로 채널의 실행 중인 PWM 업/다운 카운터와 비교됩니다.

이 PWM 카운터는 PWM 주파수에 관계없이 각 PWM 사이클마다 +16,383과 -16,384의 사이에서 증가됩니다. 또 페이지의 디지털 PWM 출력은 각 방향에서 카운터가 이 명령값을 통과하면 토글합니다. 페이지 PWM 신호의 최종적인 듀티 사이클은 `[Gate3[j].Chan[j].Pwm[k]]`의 이 값에 비례합니다. 상위 16비트의 값이 +16,383 이상인 경우(요소값은 +1,073,676,288 이상), 듀티 사이클은 100%가 됩니다(Top 신호는 항상 ON, Bottom 신호는 항상 OFF). 상위 16비트의 값이 -16,384 이하인 경우(요소값은 -1,073,741,824 이하), 듀티 사이클은 0%가 됩니다(Top 신호는 항상 OFF, Bottom 신호는 항상 ON).

`[Gate3[j].Chan[j].Pwm[k]]`는 $k = 0\sim 2$ 인 경우에는 D/A 컨버터 명령값 `[Gate3[j].Chan[j].Dac[k]]`과 $k = 3$ 인 경우에는 PFM 명령값 `[Gate3[j].Chan[j].Pfm]`과 레지스터를 공유합니다([Pfm]요소의 값은 일정 펄스 주파수 출력을 필요로 하는 어플리케이션을 위해서 저장되기 때문에 실제로는 `[Pwm[3]]`의 값도 동일하게 저장됩니다).

레지스터의 주소로 설정된 `[Motor[x].pDac]`에서 이 레지스터 중 하나가 모터 명령 출력용으로 지정되어 있는 경우에 레지스터는 각 서보 사이클 또는 페이지 사이클마다 자동으로 쓰이기 때문에 범용으로 사용할 수 없습니다. 한편 그렇지 않은 경우에 레지스터는 범용으로 사용할 수 있으며 이 레지스터에 값을 쓰고 언제든지 값을 설정할 수 있습니다.

스크립트 환경에서는 정규 범위 외의 값이 이 요소에 쓰이면 요소의 값은 그 부호의 최대 정규 값에 포화합니다.

Gatelo[j].비저장 설정 데이터 구조체 요소

이 섹션에서는 **save** 명령으로 값이 플래시 메모리에 복사되지 않는 IOGATE 디지털 I/O ASIC의 쓰기 가능한 요소를 설명합니다.

Gatelo[j].CtrlReg

설명: IOGATE 제어 레지스터

범위: \$00 .. \$FF (0 .. 255)

단위: 비트 필드

전원 셋업 시의 기본값: ([Gatelo[j].Init.CtrlReg]에 따름)

[Gatelo[j].CtrlReg]는 IOGATE 디지털 I/O 인터페이스 IC의 8비트의 요소로 IC의 설정과 동작을 제어합니다. 전원 셋업/리셋 시에 [Gatelo[j].Init] 서브 스트럭처의 저장된 설정에 기반하여 IC의 구성으로 자동으로 사용됩니다. 자동 설정의 절차가 종료되면 소프트웨어 요소 [Gatelo[j].Init.CtrlReg]의 저장된 값은 자동으로 이 요소에 복사됩니다.

[Gatelo[j].CtrlReg]의 비트 j ($j = 0 \sim 5$)는 IC의 [Gatelo[j].DataReg[j]]에 관련된 8개의 I/O 포인트의 방향을 제어합니다. 제어 비트의 값을 0으로 하면 데이터 레지스터의 쓰기 조작이 허락되어 레지스터의 각 행의 출력 함수는 유효가 됩니다. 출력 함수를 유효해도 출력이 OFF(비전도)인 경우에는 행 중 하나 또는 전부를 입력으로 사용할 수 있습니다.

제어 비트의 값을 1로 하면 데이터 레지스터에서 쓰기 조작은 허락되지 않고 출력은 무효가 되며 레지스터는 입력을 위해서 예약됩니다. 이 행을 입력에 사용하는 경우에는 이 설정을 적극 권장합니다. 데이터 레지스터에서 쓰기 조작이 입력을 무효로 할 수 없기 때문입니다.

[Gatelo[j].CtrlReg]의 비트 6과 7은 [Gatelo[j].DataReg[j]] 및 [Gatelo[j].IntrReg]에서 액세스하는 정보를 결정합니다. 비트 6과 7 둘 다 0으로 설정되면 이 레지스터의 실제 입력/출력 정보에 액세스합니다. IC가 실제로 동작하려면 2개의 비트가 둘 다 0으로 설정되는 것이 불가결합니다.

[Gatelo[j].CtrlReg]의 비트 6과 7 중 하나 또는 둘 다 1로 설정되면 [Gatelo[j].DataReg[j]] 및 [Gatelo[j].IntrReg]의 설정 레지스터에 액세스합니다. 보통 이 설정은 전원 셋업/리셋 시에만 또는 카드의 초기 설정 중에 실행합니다.

Gatelo[j].DataReg[j]

설명: IOGATE 데이터 레지스터

범위: \$00 .. \$FF (0 .. 255)

단위: 비트 필드

전원 셋업 시의 기본값: ([Gatelo[j].Init.DataReg0[j]]에 따름)

[Gatelo[j].DataReg[j]]는 IOGATE 디지털 I/O 인터페이스 IC 8비트의 요소로 인덱스 j 를 가지는 8비트의 데이터 레지스터에 관한 정보를 나타냅니다. 이 요소가 나타내는 이 레지스터의 정보는 IC의 설정과 구성에 따라 다릅니다.

레지스터의 각 I/O 포인트는 구문 `[Gatelo[j].DataReg[j].k]`를 사용하여 참조할 수 있습니다. 여기서 **k**는 레지스터의 비트 번호를 나타내는 0~7 범위의 정수 상수입니다. 이것은 'M' 변수 포인터를 I/O 포인트에 할당할 때 특히 편리합니다. 예시는 다음과 같습니다.

ptr ConveyorOn->Gatelo[2].DataReg[3].5

데이터값

`[Gatelo[j].CtrlReg]`의 비트 6 과 7 의 현재 값이 둘 다 0 인 경우에 이 요소는 레지스터인 8 개의 I/O 라인에 대하여 실제 데이터를 나타냅니다. 이것은 레지스터의 구성 상태가 아니라 동작 상태에 관한 정보입니다.

읽기 조작의 경우에는 레지스터의 구성 방법에 따라서 이 요소는 레지스터에 관련한 각 I/O 라인의 데이터 상태에 관한 4 종류의 정보 중 하나를 제공합니다.

1. 핀의 현재 고/저 전압 상태, 핀이 입력 또는 출력으로 사용되고 있는지 여부와 전압 상태와 결과의 비트값 매칭은 비트가 반전 또는 비반전으로 설정되고 있는지 여부에 따라 다름
2. 핀의 상태에 상관없이 프로세서가 레지스터의 비트에 쓴 최신 값
3. 카드 하드웨어의 구성 방법에 따라서 최근 페이지 또는 서보 클럭 엣지에 래치된 핀의 고/저 전압 상태
4. 그레이 코드에서 바이너리로의 변환에 따라 카드 하드웨어의 구성 방법에 따라서 최근 페이지 또는 서보 클럭 엣지에 래치된 핀의 고/저 전압 상태에서 취득된 값

쓰기 조작인 경우에 `[Gatelo[j].CtrlReg]`의 값에 따라 레지스터의 출력이 유효로 설정되면 이 요소는 레지스터에 관련하는 각 I/O 라인의 출력 상태를 제어할 수 있습니다.

반전 제어값

`[Gatelo[j].CtrlReg]`의 비트 6 의 현재 값이 1 이고 비트 7 가 0 인 경우에 `[Gatelo[j].DataReg[j]]`의 값은 레지스터의 8 개 라인의 반전 제어 상태를 나타냅니다.

각 비트는 번호 순서대로 매칭되어 대응하는 I/O 포인트의 반전을 지정합니다. 값을 0 으로 하면 매칭 비트의 반전 I/O 포인트가 지정됩니다. 즉, 출력의 경우에 값을 0 으로 하면 IC 자체에서 낮은(전도) 출력이 생성되고 값을 1 로 하면 높은 (비전도)출력이 생성됩니다. 입력의 경우에 라인을 Low 로 풀다운 하면 값은 1 이 되고 라인을 High 로 풀업 또는 높게 플롯시키면 값은 0 이 됩니다.

`[Gatelo[j].DataReg[j]]`의 비트값을 1 로 하면 매칭 비트의 비반전 I/O 포인트가 지정됩니다. 즉, 출력의 경우에 값을 0 으로 하면 IC 자체에서 높은 (비전도) 출력이 생성되고 값을 1 로 하면 낮은(전도) 출력이 생성됩니다. 입력인 경우에 라인을 Low 로 풀다운 하면 값은 0 이 되고 라인을 High 로 풀업 또는 높게 플롯시키면 값은 1 이 됩니다.

아이솔레이터 및 드라이버 IC 를 사용하여 구내의 배선(ACC-65E, ACC-66E, ACC-67E, ACC-68E 등)에 연결하는 IOGATE 액세스리 보드의 경우에 반전 설정은 입력과 출력 양쪽에서 싱크도 소스도 전도 상태가 데이터 레지스터 비트의 '1'에 대응한다는 것을 의미합니다.

IC 핀이 구내의 배선에 직접 연결되어 보드(ACC-14E 등)에 저항기가 풀업된 IOGATE 액세스리 보드의 경우에 출력의 반전 설정은 데이터 레지스터의 비트에 '1'이 쓰여 있으면 IC 핀의 싱크형 트랜지스터가 ON 이 되고 출력 라인이 Low 로 풀다운된다는 것을 의미합니다. 데이터 레지스터의 비트에 0 이 쓰여 있으면 트랜지스터는 OFF 가 되고 외부의 저항기가 라인을 Low 로 풀다운할 수 있습니다. 입력의 반전 설정은 입력핀이 능동적으로 Low 로 풀다운되면 데이터 레지스터의 비트에서 '1'을 읽을 수 있다는 것을

의미합니다. 입력핀이 능동적으로 High 로 풀업된 경우 또는 외부 저항기에 의한 풀업이 허가된 경우에 데이터 레지스터의 비트에서 '0'을 읽을 수 있습니다.

요소의 해당 값은 재초기화 때에 Power PMAC 에 따라서 자동으로 설정되고 디지털 I/O 액세스러리에 따라 자동 검출된 IOGATE 가 사용됩니다. 설정된 값에 따라 특정 액세스러리의 가장 일반적인 사용이 가능해집니다.

전원 셋업/리셋 시에 요소의 해당 값은 [GateIo[j].Init.DataReg64[j]]의 저장된 값에서 자동으로 복사됩니다.

레지스터의 읽기 제어값

[GateIo[j].CtrlReg] 비트 6 의 현재 값이 0 이고 비트 7 이 1 인 경우에 [GateIo[j].DataReg[j]]의 값은 레지스터 8 개의 라인 읽기 제어 상태를 나타냅니다.

레지스터의 읽기 제어값은 레지스터의 래치 제어값과 함께 레지스터의 데이터값 읽기 조작의 8 개 I/O 포인트에 무엇을 통지할지 결정합니다. 각 비트는 번호 순서대로 매칭되어 대응하는 I/O 포인트의 통지를 지정합니다.

레지스터의 읽기 제어값 비트의 처리는 같은 인덱스 'j'의 레지스터 제어값의 매칭 비트 설정에 좌우됩니다. 레지스터의 래치 제어값의 매칭 비트가 0 인 경우에 래치되지 않은 입력을 선택하면 레지스터의 읽기 제어값 비트가 핀의 값 또는 쓰기 가능한 레지스터의 값을 읽을지 여부를 결정합니다. 레지스터의 읽기 제어값 비트의 값을 0 으로 하면 레지스터의 데이터값 매칭 비트에서 읽히는 핀의 값이 선택됩니다. 비트의 값을 1 로 하면 쓰기 가능한 레지스터의 값이 선택됩니다.

레지스터의 래치 제어값 매칭 비트가 1 인 경우에 래치된 입력을 선택하면 레지스터의 읽기 제어값 비트가 직접 래치된 데이터 또는 그레이 코드에서 바이너리로 변환된 값을 읽을지 여부를 결정합니다. 레지스터의 래치 제어값 비트의 값을 0 으로 하면 레지스터의 데이터값 매칭 비트에서 읽히는 직접 래치된 값이 선택됩니다. 비트의 값을 1 로 하면 그레이 코드에서 바이너리로 변환된 값이 선택됩니다.

요소의 해당 값은 재초기화 때에 Power PMAC 에 따라서 자동으로 설정되고 디지털 I/O 액세스러리에 따라 자동 검출된 IOGATE 가 사용됩니다. 설정된 값에 따라 특정 액세스러리의 가장 일반적인 사용이 가능해집니다.

전원 셋업/리셋 시에 요소의 해당 값은 [GateIo[j].Init.DataReg128[j]]의 저장된 값에서 자동으로 복사됩니다.

레지스터의 래치 제어값

[GateIo[j].CtrlReg]의 비트 6 및 7 양쪽의 현재 값이 1 인 경우에 [GateIo[j].DataReg[j]]의 값은 레지스터의 8 개 라인의 읽기 제어 상태를 나타냅니다.

레지스터의 래치 제어값은 레지스터의 읽기 제어값과 함께 레지스터의 데이터값의 읽기 조작 8 개의 I/O 포인트에 무엇을 통지할지 결정합니다. 각 비트는 번호 순서대로 매칭되어 대응하는 I/O 포인트의 통지를 지정합니다. 각 비트 4 개의 동작은 앞의 레지스터의 읽기 제어값에 관한 섹션에서 설명했습니다.

요소의 해당 값은 재초기화 때에 Power PMAC 에 따라서 자동으로 설정되고 디지털 I/O 액세스러리에 따라 자동 검출된 IOGATE 가 사용됩니다. 설정된 값에 따라 특정 액세스러리의 가장 일반적인 사용이 가능해집니다.

전원 셋업/리셋 시에 요소의 해당 값은 [GateIo[j].Init.DataReg192[j]]의 저장된 값에서 자동으로 복사됩니다.

Gatelo[*i*].IntrReg

설명: IOGATE 인터럽트 레지스터

범위: \$00 .. \$FF (0 .. 255)

단위: 비트 필드

전원 셋업 시의 기본값: 해당 없음(입력)

[Gatelo[*i*].IntrReg]는 IOGATE 디지털 I/O 인터페이스 IC의 8비트 인터럽트 레지스터에 관한 정보를 나타냅니다. 이 요소가 나타내는 이 레지스터의 정보는 IC의 설정과 구성에 따라 다릅니다. IOGATE를 사용하는 제품의 현재 탑재로는 이 요소는 실용적이라고는 할 수 없습니다.

데이터값

[Gatelo[*i*].CtrlReg]의 비트 6과 7의 현재 값이 둘 다 0인 경우에 이 요소는 레지스터의 8개 'En'입력 라인의 실제 데이터에 관한 정보를 나타냅니다. 이것은 레지스터의 구성 상태가 아니라 동작 상태에 관한 정보입니다. 이 모드에서는 레지스터는 읽기 전용입니다. IOGATE를 사용하는 제품의 현재 탑재로는 보드의 하드웨어 구성에 따라서 이 라인은 전부 페이지 또는 서보 클럭 신호에 연결됩니다.

반전 제어값

[Gatelo[*i*].CtrlReg]의 비트 6의 현재 값이 1이고 비트 7가 0인 경우에 [Gatelo[*i*].DataReg[*j*]]의 값은 레지스터의 8개 라인의 반전 제어 상태를 나타냅니다.

반전 제어값은 IOGATE IC의 8개 'En'라인 입력 반전을 결정합니다. 각 비트는 번호 순서대로 매칭되어 대응하는 'En'라인의 반전을 지정합니다. 값을 0으로 하면 매칭 비트의 반전 I/O 포인트가 지정됩니다. IC의 레벨을 내리면 값은 1이 되고 레벨을 올리면 값은 0이 됩니다. 값을 1로 하면 매칭 비트의 비반전 I/O 포인트가 지정됩니다. 레벨을 내리면 값은 0이 되고 레벨을 올리면 값은 1이 됩니다.

요소의 해당 값은 재초기화 때에 Power PMAC에 따라서 자동으로 설정되고 디지털 I/O 액세스러리에 따라 자동 검출된 IOGATE가 사용됩니다. 설정된 값에 따라 특정 액세스러리의 가장 일반적인 사용이 가능해집니다.

전원 셋업/리셋 시에 요소의 해당 값은 [Gatelo[*i*].Init.IntrReg64]의 저장된 값에서 자동으로 복사됩니다.

마스크 제어값

[Gatelo[*i*].CtrlReg] 비트 6의 현재 값이 0이고 비트 7이 1인 경우에 [Gatelo[*i*].IntrReg]의 값은 레지스터의 8개 라인의 마스크 제어 상태를 나타냅니다.

마스크 제어값은 IOGATE IC에 대한 'En'라인 입력 중 IC의 인터럽트 출력을 작성할 수 있는 라인 입력을 결정합니다. 각 비트는 번호 순서대로 매칭되어 대응하는 'En'라인의 사용을 지정합니다. 값을 0으로 하면 매칭된 En라인의 사용이 무효가 됩니다. 값을 1로 하면 매칭된 En라인의 사용이 유효가 됩니다. En입력의 하강 엣지는 IC의 'INT' 출력 하강 엣지를 작성합니다.

현재 Power PMAC I/O 액세스러리에서는 이 기능에는 중요한 용도가 없습니다. IC의 En입력은 점퍼에 따라서 선택된 시스템 서보 또는 페이지 클럭 신호에서 얻어지며 이 신호는 입력 데이터의 래치에 사용할 수 있습니다. IC에서 'INT' 출력은 프로세서에 인터럽트할 수 없지만 액세스러리의 ID칩으로 읽을 수 있습니다.

요소의 해당 값은 재초기화 때에 Power PMAC 에 따라서 자동으로 설정되고 디지털 I/O 액세스러리에 따라 자동 검출된 IOGATE 가 사용됩니다. 설정된 값에 따라 특정 액세스러리의 가장 일반적인 사용이 가능해집니다.

전원 셋업/리셋 시에 요소의 해당 값은 [Gatelo[j].Init.DataReg128]의 저장된 값에서 자동으로 복사됩니다.

엣지/레벨 제어

[Gatelo[j].CtrlReg]의 비트 6 및 7 양쪽의 현재 값이 1 인 경우에 [Gatelo[j].IntrReg[j]]의 값은 레지스터의 8 개 라인의 엣지/레벨 제어 상태를 나타냅니다.

엣지/레벨 제어값은 IOGATE IC 에 대한 'En' 라인 입력 중 레벨 트리거의 인터럽트를 제공할 수 있는 라인 입력과 IC 의 인터럽트 출력에 엣지 트리거의 인터럽트를 제공할 수 있는 라인 입력을 결정합니다. 각 비트는 번호 순서대로 매칭되어 대응하는 'En' 라인의 사용을 지정합니다. 값을 0 으로 하면 매칭된 En 라인의 레벨 트리거 인터럽트가 지정됩니다. 값을 1 로 하면 매칭된 En 라인의 엣지 트리거 인터럽트가 지정됩니다. 이 설정은 입력이 인터럽트에 사용할 수 있는 경우에만 유효합니다.

현재 Power PMAC I/O 액세스러리에서는 이 기능에는 중요한 용도가 없습니다. IC 의 En 입력은 점퍼에 따라서 선택된 시스템 서보 또는 페이지 클록 신호에서 얻어지며 이 신호는 입력 데이터의 래치에 사용할 수 있습니다. IC 에서 'INT' 출력은 프로세서에 인터럽트할 수 없지만 액세스러리의 ID 칩으로 읽을 수 있습니다.

요소의 해당 값은 재초기화 때에 Power PMAC 에 따라서 자동으로 설정되고 디지털 I/O 액세스러리에 따라 자동 검출된 IOGATE 가 사용됩니다. 설정된 값에 따라 특정 액세스러리의 가장 일반적인 사용이 가능해집니다.

전원 셋업/리셋 시에 요소의 해당 값은 [Gatelo[j].Init.DataReg192]의 저장된 값에서 자동으로 복사됩니다.

Gather.비저장 설정 데이터 구조체 요소

이 섹션에서는 데이터 개더링 구조체 내부의 편리한 설정 요소에 대해서 설명합니다. 이 값은 **save** 명령으로 플래시 메모리에 복사되지 않습니다. 이 요소의 값은 대부분의 사용 방법으로 IDE의 플롯, 조정 및 스코프 기능에 따라 자동으로 설정됩니다. IDE는 나중에 사용하기 위해서 이 요소의 값을 Power PMAC의 플래시 메모리의 다른 파일에 저장합니다.

데이터 개더링은 서보 인터럽트 또는 페이지 인터럽트 중에 실행할 수 있습니다. 요소에는 양쪽의 환경을 위해서 동등한 세트가 있습니다.

Gather.Addr[i]

설명: 서보 인터럽트 데이터 개더링 소스 주소

범위: Power PMAC 주소

단위: 유효한 주소

전원 셋업 시의 기본값: Sys.ServoCount.a

[Gather.Addr[i]]는 서보 인터럽트 중의 데이터 개더링용 *i*번째 레지스터의 소스 주소를 지정합니다. *i*의 정규 값은 0부터 127의 범위입니다. 그 때문에 샘플마다 최대 128의 레지스터를 수집할 수 있습니다.

보통 [Gather.Addr[i]]의 값을 설정하는 경우에는 레지스터의 데이터 구조체 요소 이름을 지정하고 그 뒤에 주소임을 나타내는 '.a'를 부가합니다. 이것으로 주소를 지정할 수 있습니다. 사용자는 레지스터의 수치 주소를 알 필요는 없습니다. 주소의 수치를 직접 지정도 할 수 있습니다. [Gather.Addr[i]]의 값에 대한 쿼리가 있으면 어떤 경우라도 Power PMAC은 주소의 수치를 통지합니다.

Power PMAC은 [Gather.Type[i]]의 설정에 따라서 [Gather.Addr[i]]가 지정하는 주소의 데이터를 해석합니다. [Gather.Type[i]]는 스크립트 명령을 사용하여 [Gather.Addr[i]]가 요소의 주소에 설정되면 Power PMAC에 따라 자동으로 설정됩니다. 스크립트 환경 또는 C에서 [Gather.Addr[i]]가 직접 숫자에 설정된 경우에는 [Gather.Type[i]]를 명시적으로 설정하고 데이터 형식을 최적화해야 합니다.

[Gather.Items]가 *i*보다 큰 경우에 개더링이 유효화되어 있으면 지정된 주소의 데이터는 샘플마다 수집됩니다.

Gather.Enable

설명: 서보 인터럽트 데이터 개더링 유효화 제어

범위: 0 .. 3

단위: 없음

전원 셋업 시의 기본값: 0

[Gather.Enable]은 서보 인터럽트의 데이터 개더링 함수를 유효로 할지 여부를 제어합니다. 0 으로 설정된 경우에 서보 인터럽트는 유효가 아닙니다. 유효한 경우에 데이터의 저장은 버퍼의 선두부터 시작되어 버퍼 내에 이전 데이터가 있는 경우에는 덮어씁니다.

[Gather.Enable]이 1 로 설정된 경우에 서보 인터럽트 중의 개더링은 유효가 아닙니다. 유효한 경우에 데이터 저장은 이전 수집이 정지한 부분부터 시작됩니다. 그 때문에 새로 수집된 데이터는 버퍼의 마지막에 추가됩니다. 개더링하는 항목의 주소와 수를 지정한 뒤에 이 요소를 1로 설정하면 유용한 경우가 있습니다. 왜냐하면 1로 설정하면 Power PMAC 이 각 샘플을 저장하기 위해서 필요한 워드 수(Gather.LineLength) 및 저장할 수 있는 샘플의 수(Gather.MaxLines)를 계산하기 때문입니다.

[Gather.Enable]이 2 로 설정된 경우에 서보 인터럽트 중의 유한한 개더링은 유효이며 다른 설정 요소의 값에 따라 확립된 규칙에 따라서 속행됩니다. 수집된 샘플의 수가 [Gather.MaxSamples]와 같은 경우 또는 개더링 버퍼가 가득 찬 경우에 개더링은 자동으로 무효가 되고 [Gather.Enable]은 0 으로 설정됩니다.

[Gather.Enable]이 3 으로 설정된 경우에 서보 인터럽트 중의 무한 개더링은 유효이며 다른 설정 요소의 값에 따라 확립된 규칙에 따라서 속행됩니다. 개더링은 [Gather.Enable]의 3 의 값을 변경하여 정지될 때까지 계속합니다. 이 모드에서는 [Gather.MaxSamples]는 사용되지 않습니다. 버퍼는 로터리식으로 사용되어 필요에 따라서 새로운 샘플이 오래된 샘플을 덮어씁니다.

[Gather.PhaseEnable]의 값이 0 보다 큰 경우에 [Gather.Enable]은 0 이상의 값으로 설정할 수 없습니다.

예

```
open prog 73
Gather.Enable = 2;           // Start gathering
abs; linear;                 // Specify move mode
X25 Y15 F10 ta100 td200 ts50; // Command move
dwell 100;                   // Stop pre-calc, allow settling
Gather.Enable = 0;           // Stop gathering
```

Gather.Items

설명: 서보 인터럽트 중에 샘플마다 개더링하는 데이터 항목의 수

범위: 0 .. 128

단위: 없음

전원 셋업 시의 기본값: 0

[Gather.Items]는 서보 인터럽트의 데이터 개더링 함수가 유효한 경우에 샘플마다 개더링하는 항목의 수를 지정합니다. [Gather.Addr[0]]부터 [Gather.Addr[Items-1]]에 따라 지정되는 주소의 항목은 샘플마다 수집됩니다.

Gather.MaxSamples

설명: 서보 인터럽트 중에 개더링하는 데이터 샘플의 최대 수

범위: 음의 값이 아닌 정수

단위: 없음

전원 셋업 시의 기본값: 0

[Gather.MaxSamples]는 서보 인터럽트의 데이터 개더링 함수가 유효 모드([Gather.Enable] = 2)에서 유효한 경우에 수집할 수 있는 샘플의 최대 수를 지정합니다. 이처럼 많은 샘플이 버퍼에 저장되고도 데이터 개더링이 아직 유효한 경우에 Power PMAC 은 [Gather.Enable]을 0으로 설정하고 개더링을 자동으로 무효로 합니다.

무한 모드([Gather.Enable] = 3)로 수집하는 경우에 [Gather.MaxSamples]는 사용되지 않습니다.

Gather.Period

설명: 서보 인터럽트 데이터 개더링의 샘플링 주기

범위: 0 .. 65,535

단위: 서보 인터럽트 주기

전원 셋업 시의 기본값: 0

[Gather.Period]는 서보 인터럽트의 데이터 개더링 함수가 유효이고 서보 인터럽트 주기(서보 사이클)인 경우에 연속하는 샘플의 간격을 지정합니다. [Gather.Period]가 1로 설정된 경우에 개더링은 서보 사이클마다 실행됩니다. 2로 설정된 경우에는 1 서보 사이클 간격이고 이후 동일하게 설정값에 따른 간격으로 실행됩니다. 기본값 0으로 설정된 경우에 개더링은 주기적이 아니라 단발로 실행됩니다.

Gather.PhaseAddr[i]

설명: 페이즈 인터럽트 데이터 개더링 소스 주소

범위: Power PMAC 주소

단위: 유효한 주소

전원 셋업 시의 기본값:

[Gather.PhaseAddr[i]]는 페이즈 인터럽트 중의 데이터 개더링용의 *i*번째 레지스터의 소스 주소를 지정합니다. *i*의 정규 값은 0부터 7의 범위입니다. 이 때문에 샘플마다 최대 8의 레지스터를 수집할 수 있습니다.

보통 [Gather.PhaseAddr[i]]의 값을 설정하는 경우에는 레지스터의 데이터 구조체 요소의 이름을 지정하고 그 다음에 주소임을 나타내는 '.a'를 부가합니다. 이것으로 주소를 지정할 수 있습니다. 사용자는 레지스터의 수치 주소를 알 필요는 없습니다. 주소의 수치를 직접

지정할 수 있습니다. [Gather.PhaseAddr[*i*]]의 값에 대한 쿼리가 있으면 어떤 경우라도 Power PMAC은 주소의 수치를 통지합니다.

Power PMAC은 [Gather.PhaseType[*i*]]의 설정에 따라서 [Gather.PhaseAddr[*i*]]가 지정하는 주소의 데이터를 해석합니다. [Gather.PhaseType[*i*]]는 스크립트 명령을 사용하여 [Gather.PhaseAddr[*i*]]가 요소의 주소로 설정되면 Power PMAC에 따라 자동으로 설정됩니다. [Gather.PhaseAddr[*i*]]가 스크립트 환경 또는 C에서 숫자로 직접 설정된 경우에는 [Gather.PhaseType[*i*]]를 명시적으로 설정하고 데이터 형식을 최적화해야 합니다.

[Gather.PhaseItems]가 *i* 보다 큰 경우에 개더링이 유효화되어 있으면 지정된 주소의 데이터는 샘플마다 수집됩니다.

Gather.PhaseEnable

설명 : 페이지 인터럽트 데이터 개더링 유효화 제어

범위: 0 .. 3

단위: 없음

전원 셋업 시의 기본값: 0

[Gather.PhaseEnable]은 페이지 인터럽트의 데이터 개더링 함수를 유효로 할지 여부를 제어합니다. 0으로 설정된 경우에 페이지 인터럽트는 유효가 아닙니다. 유효한 경우에 데이터의 저장은 버퍼의 선두부터 시작되어 버퍼 내에 이전 데이터가 있는 경우에는 덮어씁니다.

[Gather.PhaseEnable]이 1로 설정된 경우에 페이지 인터럽트 중의 개더링은 유효가 아닙니다. 유효한 경우에 데이터 저장은 이전 수집이 정지한 부분부터 시작됩니다. 그 때문에 새로 수집된 데이터는 버퍼의 마지막에 추가됩니다. 수집하는 항목의 주소와 수를 지정한 뒤에 이 요소를 1로 설정하면 유용한 경우가 있습니다. 왜냐하면 1로 설정하면 Power PMAC이 각 샘플을 저장하기 위해서 필요한 워드 수(Gather.PhaseLineLength) 및 저장할 수 있는 샘플 수(Gather.PhaseMaxLines)를 계산하기 때문입니다.

[Gather.PhaseEnable]이 2로 설정된 경우에 페이지 인터럽트 중의 유한 개더링은 유효이며 다른 설정 요소의 값에 의해 확립된 규칙에 따라서 속행됩니다. 수집된 샘플의 수가 [Gather.PhaseMaxSamples]와 같은 경우 또는 개더링 버퍼가 가득 찬 경우에 개더링은 자동으로 무효가 되고 [Gather.PhaseEnable]은 0으로 설정됩니다.

[Gather.PhaseEnable]이 3으로 설정된 경우에 페이지 인터럽트 중의 무한의 개더링은 유효이며 다른 설정 요소의 값에 의해 확립된 규칙에 따라서 속행됩니다. 개더링은 [Gather.PhaseEnable]의 3의 값을 변경하여 정지될 때까지 계속합니다. 이 모드에서는 [Gather.PhaseMaxSamples]는 사용되지 않습니다. 버퍼는 로터리식으로 사용되어 필요에 따라서 새로운 샘플이 오래된 샘플을 덮어씁니다.

[Gather.Enable]의 값이 0보다 큰 경우에 [Gather.PhaseEnable]은 0 이상의 값으로 설정할 수 없습니다.

Gather.PhaseItems

설명 : 페이지 인터럽트 중에 샘플마다 수집하는 데이터 항목의 수

범위: 0 .. 8

단위: 없음

전원 셋업 시의 기본값: 0

[Gather.Items]는 페이지 인터럽트의 데이터 개더링 함수가 유효한 경우에 샘플마다 수집하는 항목의 수를 지정합니다. [Gather.Addr[0]]부터 [Gather.Addr[Items-1]]에 따라 지정되는 주소의 항목은 샘플마다 수집됩니다.

Gather.PhaseMaxSamples

설명 : 페이지 인터럽트 중에 수집하는 데이터 샘플의 최대 수

범위: 음의 값이 아닌 정수

단위: 없음

전원 셋업 시의 기본값: 0

[Gather.PhaseMaxSamples]는 페이지 인터럽트의 데이터 개더링 함수가 유효한 모드([Gather.PhaseEnable] = 2)에서 유효한 경우에 수집할 수 있는 샘플의 최대 수를 지정합니다. 이처럼 많은 샘플이 버퍼에 저장되고도 데이터 개더링이 아직 유효한 경우에 Power PMAC 은 [Gather.PhaseEnable]을 0 으로 설정하고 개더링을 자동으로 무효로 합니다.

무한 모드([Gather.PhaseEnable] = 3)에서 수집하는 경우에 [Gather.PhaseMaxSamples]는 사용되지 않습니다.

Gather.PhasePeriod

설명 : 페이지 인터럽트 데이터 개더링의 샘플링 주기

범위: 0 .. 65,535

단위: 페이지 인터럽트 주기

전원 셋업 시의 기본값: 0

[Gather.PhasePeriod]는 서보 인터럽트의 데이터 개더링 함수가 유효하고 페이지 인터럽트 주기(페이지 사이클)인 경우에 연속하는 샘플 간격을 지정합니다. [Gather.PhasePeriod]가 1 로 설정된 경우에 개더링은 페이지 사이클마다 실행됩니다. 2 로 설정된 경우에는 1 페이지 사이클 간격이고 이후 동일하게 설정값에 따른 간격으로 실행됩니다. 기본값 0 으로 설정된 경우에 개더링은 주기적이 아니라 단발로 실행됩니다.

Gather.PhaseType[i]

설명 : 페이지 인터럽트 데이터 개더링 소스 형식

범위: 0 .. 65,535

단위: 열거

전원 셋업 시의 기본값: 0

[Gather.PhaseType[i]]는 페이지 인터럽트 중의 데이터 개더링용 *i*번째 레지스터의 데이터를 해석하기 위해서 Power PMAC 이 사용하는 데이터 형식 규칙을 지정합니다. *i*의 정규 값은 0 부터 7의 범위입니다. gather_csv 유틸리티 프로그램을 사용하여 수집한 데이터를 호스트 컴퓨터에 업로드하면 그 프로그램은 이 요소의 값을 사용하여 통지되는 값을 수집한 데이터에서 도출합니다. 수집한 데이터를 업로드하는 대부분의 사용자 프로그램과 동일하게 IDE의 플롯 및 조정 함수는 이 유틸리티를 사용합니다.

Power PMAC의 스크립트 환경에서는 [Gather.PhaseAddr[i]]가 요소의 주소에 설정되면 Power PMAC은 그 요소에 대하여 가장 적절한 형식이 되도록 [Gather.PhaseType[i]]를 자동으로 설정합니다. 단, 주소가 스크립트 환경의 수치에 따라서 직접 지정되었거나 C 명령을 사용하여 지정되어 있는 경우에는 [Gather.PhaseType[i]]를 명시적으로 지정해야 합니다.

다음의 폴 워드 형식이 지원됩니다.

- 0: 32 비트의 정수, 부호가 없음
- 1: 32 비트의 정수, 부호가 있음
- 2: 24 비트의 정수, 부호가 없음(32 비트 버스의 상위 24 비트)
- 3: 24 비트의 정수, 부호가 있음(32 비트 버스의 상위 24 비트)
- 4: 32 비트의 부동 소수점(단정도)
- 5: 64 비트의 부동 소수점(배정도)

데이터를 포맷하여 32 비트의 정수 레지스터에서 연속하는 번호가 있는 비트의 임의의 세트값을 작성할 수 있습니다. 이 경우에 [Gather.PhaseType[i]]는 다음 3개의 필드에 분할됩니다.

- 비트 0~2: =6(부호가 없음), =7(부호가 있음)
- 비트 6~10: =32 -(비트의 번호)
- 비트 11~15: =시작 비트 번호

비트 3~5는 앞으로의 사용을 위해서 예약되었습니다. 0으로 두십시오.

예를 들어 레지스터의 비트 20을 0 또는 1만으로 통지하는 경우에 비트 0~2는 부호가 붙지 않은 비트에 대하여 6의 값(110)을 형성하고 비트 6~10은 하나의 비트에 대하여 31의 값(11111)을 형성합니다. 또한 비트 11~15는 시작 비트 20에 대하여 20의 값(10100)을 형성합니다. [Gather.PhaseType[i]]의 결과 값은 1010,0111,1100,0110(기수 2), \$A7C6(기수 16) 또는 49,250(기수 10)입니다.

다른 예로 레지스터의 비트 8~15 를 부호가 있는 값으로 통지하는 경우에 비트 0~2 는 부호가 있는 비트에 대하여 7 의 값(111)을 형성하고 비트 6~10 은 8 개의 비트에 대하여 24 의 값(11000)을 형성합니다. 또한 비트 11~15 는 시작 비트 8 에 대하여 8 의 값(01000)을 형성합니다. [Gather.PhaseType[i]]의 결과 값은 0100,0110,0000,0111(기수 2), \$4607(기수 16) 또는 17,927(기수 10)입니다.

Gather.Type[i]

설명: 서보 인터럽트 데이터 개더링 소스 형식

범위: 0 ..65,535

단위: 열거

전원 셋업 시의 기본값: 0

[Gather.Type[i]]는 서보 인터럽트 중의 데이터 개더링용 *i*번째 레지스터의 데이터를 해석하기 위해서 Power PMAC 이 사용하는 데이터 형식 규칙을 지정합니다. *i*의 정규 값은 0 부터 127 의 범위입니다. gather_csv 유틸리티 프로그램을 사용하여 수집한 데이터를 호스트 컴퓨터에 업로드하면 그 프로그램은 이 요소의 값을 사용하여 통지되는 값을 수집한 데이터에서 도출합니다. 수집한 데이터를 업로드하는 대부분의 사용자 프로그램과 동일하게 IDE 의 플롯 및 조정 함수는 이 유틸리티를 사용합니다.

Power PMAC 의 스크립트 환경에서는 [Gather.Addr[i]]가 요소의 주소에 설정되면 Power PMAC 은 그 요소에 대하여 가장 적절한 형식이 되도록 [Gather.Type[i]]를 자동으로 설정합니다. 단, 주소가 스크립트 환경의 수치에 따라서 직접 지정되었거나 C 명령을 사용하여 지정된 경우에는 [Gather.Type[i]]를 명시적으로 지정해야 합니다.

다음의 폴 워드 형식이 지원됩니다.

- 0: 32 비트의 정수, 부호가 없음
- 1: 32 비트의 정수, 부호가 있음
- 2: 24 비트의 정수, 부호가 없음(32 비트 버스의 상위 24 비트)
- 3: 24 비트의 정수, 부호가 있음(32 비트 버스의 상위 24 비트)
- 4: 32 비트의 부동 소수점(단정도)
- 5: 64 비트의 부동 소수점(배정도)

데이터를 포맷하여 32 비트의 정수 레지스터에서 연속하는 번호가 있는 비트의 임의의 세트값을 작성할 수 있습니다. 이 경우에 [Gather.Type[i]]는 다음 3 개의 필드에 분할됩니다.

- 비트 0~2: =6(부호가 없음), =7(부호가 있음)
- 비트 6~10: =32 -(비트의 번호)
- 비트 11~15: =시작 비트 번호

비트 3~5 는 앞으로의 사용을 위해서 예약되었습니다. 0 으로 두십시오.

예를 들어 레지스터의 비트 20 을 0 또는 1 만으로 통지하는 경우에 비트 0~2 는 부호가 붙지 않은 비트에 대하여 6 의 값(110)을 형성하고 비트 6~10 은 하나의 비트에 대하여 31 의 값(11111)을 형성합니다. 또한 비트 11~15 는 시작 비트 20 에 대하여 20 의 값(10100)을 형성합니다. [Gather.Type[i]]의 결과 값은 1010,0111,1100,0110(기수 2), \$A7C6(기수 16) 또는 49,250(기수 10)입니다.

다른 예로 레지스터의 비트 8~15 를 부호가 있는 값으로 통지하는 경우에 비트 0~2 는 부호가 있는 비트에 대하여 7 의 값(111)을 형성하고 비트 6~10 은 8 개의 비트에 대하여 24 의 값(11000)을 형성합니다. 또한 비트 11~15 는 시작 비트 8 에 대하여 8 의 값(01000)을 형성합니다. [Gather.Type[i]]의 결과 값은 0100,0110,0000,0111(기수 2), \$4607(기수 16) 또는 17,927(기수 10)입니다.

Ldata.비저장 설정 데이터 구조체 요소

좌표계 PLC 프로그램 및 통신 스레드에는 각각 고유의 [Ldata](로컬 데이터) 데이터 구조체가 있습니다. 실행 중인 프로그램 또는 스레드의 내부에서 [Ldata.{요소}]로 이 구조체의 요소에 직접 액세스할 수 있습니다. [Coord[x].Ldata.{요소}]나 [Plc[i].Ldata.{요소}]등 고레벨의 구조체를 사용하여 외부에서 액세스해야 합니다.

Ldata.C[i]

설명: 로컬 변수 'C'의 값

범위: 부동 소수점

단위: 사용자 정의

전원 셋업 시의 기본값: 0.0

[Ldata.C[i]]는 좌표계 PLC 프로그램 또는 스레드의 로컬 변수 'C'에 액세스하는 대체 수단입니다. 변수 'C'는 사용자 설명 키네마틱스 알고리즘 간에서 축 위치($i = 0 \sim 31$) 및 속도($i = 32 \sim 63$)를 전하기 위해 사용됩니다. 이 로컬 데이터 요소는 보통 [Coord[x]] 데이터 구조체 내부에서만 사용됩니다.

각 축 위치의 인덱스 'i'는 다음과 같습니다.

축	인덱스	축	인덱스	축	인덱스	축	인덱스
A	0	Z	8	HH	16	SS	24
B	1	AA	9	LL	17	TT	25
C	2	BB	10	MM	18	UU	26
U	3	CC	11	NN	19	VV	27
V	4	DD	12	OO	20	WW	28
W	5	EE	13	PP	21	XX	29
X	6	FF	14	QQ	22	YY	30
Y	7	GG	15	RR	23	ZZ	31

각 축의 속도 인덱스 'i'는 다음과 같습니다.

축	인덱스	축	인덱스	축	인덱스	축	인덱스
A	32	Z	40	HH	48	SS	56
B	33	AA	41	LL	49	TT	57
C	C4	BB	42	MM	50	UU	58
U	C5	CC	43	NN	51	VV	59
V	C6	DD	44	OO	52	WW	60
W	C7	EE	45	PP	53	XX	61
X	C8	FF	46	QQ	54	YY	62
Y	C9	GG	47	RR	55	ZZ	63

스크립트 환경에서는 'Ci'를 사용하는 직접 액세스는 보다 빠르고 간단해야 하므로 대부분의 경우 프로그램 또는 스레드 내부에서 사용됩니다. 따라서 [Ldata] 구조체를 사용하는 액세스는 보통 디버그를 위해 주로 외부 액세스로 사용됩니다.

Ldata.Control

설명: 벡터 및 매트릭스 함수의 변수 형식

범위: 0 .. 2

단위: 열거

전원 셋업 시의 기본값: 0

[Ldata.Control]은 프로그램 또는 통신 스레드에서 스크립트의 벡터 및 매트릭스 함수에 사용하는 변수의 형식을 지정합니다. 변수의 번호를 지정하는 함수의 인수는 이 요소에 따라서 결정되는 변수 형식의 번호를 지정합니다.

[Ldata.Control]이 전원 셋업 시의 기본값 0 으로 설정된 경우에 벡터 및 매트릭스 함수는 글로벌한 'P' 변수를 사용합니다.

[Ldata.Control]이 1 로 설정된 경우에 벡터 및 매트릭스 함수는 로컬의 'L' 변수를 사용합니다.

2012 년 3/4 분기에 출시된 V1.5 의 새로운 요소인 [Ldata.Control]이 2 로 설정된 경우에 벡터 및 매트릭스 함수는 사용자 버퍼의 Ddata 요소(Sys.Ddata[i])를 사용합니다.

모든 경우에서 변수는 배정도의 부동 소수점 변수입니다.

예를 들어 변수 '6'부터 시작하여 한 개 간격으로 3 개의 변수 합계를 계산하는 벡터 함수 **sum(6,3,1)**은 [Ldata.Control]이 0 인 경우에 글로벌 변수 'P6', 'P7', 'P8'의 합계를 반환합니다. [Ldata.Control]이 1 인 경우에는 프로그램 또는 스레드의 로컬 변수 'L6', 'L7', 'L8'의 합계를 반환합니다. [Ldata.Control]이 1 인 경우에는 (글로벌한)사용자 버퍼 요소 [Sys.Ddata[6]], [Sys.Ddata[7]], [Sys.Ddata[8]]의 합계를 반환합니다.

Ldata.coord

설명: 프로그램 또는 스레드를 위해서 주소 지정된 좌표계

범위: 0 .. 127

단위: 열거

전원 셋업 시의 기본값: 0(PLC 및 통신 스레드)
좌표계의 번호(좌표계)

[Ldata.coord]는 프로그램 또는 스레드에 모달로 주소 지정된 좌표계를 지정합니다. **run** 이나 **abort** 등 프로그램의 다이렉트 명령에 따라서 어떤 좌표계에 영향을 주는지 **pread** 나 **tread** 등의 명령에 따라서 어떤 좌표계가 쿼리의 대상이 될지 'Q' 변수의 어떤 좌표계의 세트가 액세스될지 결정합니다.

[Ldata.coord]는 PLC 프로그램 및 호스트 통신 스레드의 사용자에게 따라서 설정됩니다. 전원 셋업 시의 기본값은 0으로 좌표계 0을 지정합니다. 이는 거의 사용되지 않기 때문에 좌표계 고유 정보의 액세스를 시도하기 전에 이 요소의 값을 명시적으로 설정하는 것이 중요합니다.

좌표계의 [Ldata.coord]의 값은 해당 좌표계의 번호로 고정됩니다. 사용자는 이것을 변경할 수 없습니다. 즉, 해당 좌표계에서 실행되는 모션 프로그램은 해당 좌표계의 정보에 자동으로 액세스합니다.

통신 스레드에서는 스레드의 [Ldata.coord]의 값은 온라인 **&{상수}** 명령에 따라 설정됩니다.

Ldata.D[i]

설명: 로컬 변수 'D'의 값

범위: 부동 소수점

단위: 사용자 정의

전원 셋업 시의 기본값: 0.0

[Ldata.D[i]]는 좌표계 PLC 프로그램 또는 스레드의 로컬(비스택) 변수 **'D'**에 액세스하는 대체 수단입니다. 인덱스의 값 **'i'**는 0부터 54 범위가 됩니다. 스크립트 환경에서는 **'D'**를 사용하는 직접 액세스는 보다 빠르고 간단해야 하므로 대부분의 경우 프로그램 또는 스레드 내부에서 사용됩니다. [Ldata] 구조체를 사용하는 액세스는 보통 디버그를 위해 주로 외부 액세스로 사용됩니다.

Ldata.GoBack

설명: 프로그램의 일시 정지/Abort 전의 점프 백의 수

범위: 0 .. 255

단위: 열거

전원 셋업 시의 기본값: 10(Coord[x].Ldata.GoBack)
0(Plc[x].Ldata.GoBack)

[Ldata.GoBack]은 프로그램 실행을 일시 정지 또는 Abort 하기 전에 점프 백 할 수 있는 횟수를 지정합니다. 일시 정지 또는 Abort 하지 않고 실행을 점프 백 할 수 있는 횟수는 ([GoBack]+1)회입니다.

점프 백은 **while** 또는 **do..while** 루프의 마지막 또는 프로그램 전의 행에 대한 **goto** 명령 중 하나에 따라서 발생합니다. 서브루틴 또는 서브 프로그램에서 회귀는 점프 백에 포함되지 않습니다.

좌표계에서 [Ldata.GoBack]은 포워드 키네마틱스와 인버스 키네마틱스의 서브루틴을 실행하기 위해서 사용됩니다(이것은 모션 프로그램 자체의 실행에 사용되는 저장 설정 요소 [Coord[x].GoBack]과는 다릅니다). 키네마틱스 알고리즘의 일부는 실제로 반복형이며 솔루션에 집중하는 경향이 있습니다. 이 파라미터는 알고리즘이 수렴하지 않는 경우에 실행을 정지하기 위해서 사용할 수 있습니다. 이 경우에 점프 백이 너무 많으면 키네마틱스 서브루틴 및 그것을 호출하는 모션 프로그램이 에러로 정지합니다.

PLC 프로그램에서는 점프 백이 너무 많은 경우에는 PLC 프로그램의 스캔이 종료하고 이 PLC 실행이 다음의 사이클(RTI 또는 백그라운드)에서 재개됩니다.

Ldata.L[i]

설명: 로컬 변수 'L'의 값

범위: 부동 소수점

단위: 사용자 정의

전원 셋업 시의 기본값: 0.0

[Ldata.L[i]]는 좌표계 PLC 프로그램 또는 스레드의 로컬 스택 변수 '**L**'에 액세스하는 대체 수단입니다. 인덱스의 값 '**i**'는 0 부터 16,383 범위이며 다른 베이스를 가지는 서브 프로그램이 현재 실행 중이어도 최상위 프로그램 스택을 기준으로 합니다. Status 요소 [Ldata.Lindex]는 현재 실행 중인 프로그램 또는 서브 프로그램의 베이스 인덱스를 나타냅니다.

스크립트 환경에서 '**L**'를 사용하는 직접 액세스는 보다 빠르고 간단해야 하므로 대부분의 경우에 프로그램 또는 스레드 내부에서 사용됩니다. [Ldata] 구조체를 사용하는 액세스는 보통 디버그를 위해 주로 외부 액세스로 사용됩니다.

Ldata.motor

설명: 프로그램 또는 스레드를 위해서 주소 지정된 모터

범위: 0 .. 255

단위: 열거

전원 셋업 시의 기본값: 0

[Ldata.motor]는 프로그램 또는 스레드에 모달에 주소 지정된 모터를 지정합니다. jog 나 home 등 프로그램의 다이렉트 명령에 따라서 어떤 모터가 영향받을지 결정합니다.

[Ldata.motor]는 좌표계용의 모션 프로그램, PLC 프로그램 및 호스트 통신 스레드에서 설정됩니다. 전원 셋업 시의 기본값 0 으로 모터 0 을 지정합니다. 이는 거의 사용되지 않기 때문에 모터를 명령하기 전에 이 요소의 값을 명시적으로 설정하는 것이 중요합니다.

통신 스레드에서는 스레드의 [Ldata.motor]의 값은 온라인 **#상수** 명령에 따라 설정됩니다.

Ldata.R[i]

설명: 로컬 변수 'R'의 값

범위: 부동 소수점

단위: 사용자 정의

전원 셋업 시의 기본값: 0.0

[Ldata.R[i]]는 좌표계 PLC 프로그램 또는 스레드의 로컬 스택의 pass/return 변수 '**R**'에 액세스하는 대체 수단입니다. 인덱스의 값 '**i**'에는 0 부터 255 범위가 됩니다. 루틴 실행의 **R0** 은 루틴의 [L](Ldata.Lsize)에 상응하며 이것은 스택 전체의 [Ldata.L[Ldata.Lindex + Ldata.Lsize]]에 상응합니다.

스크립트 환경에서는 '**R**'를 사용하는 직접 액세스는 보다 빠르고 간단해야 하므로 대부분의 경우에 프로그램 또는 스레드 내부에서 사용됩니다. [Ldata] 구조체를 사용하는 액세스는 보통 디버그를 위해 주로 외부 액세스로 사용됩니다.

Motor[x].비저장 설정 데이터 구조체 요소

이 섹션에서는 **save** 명령으로 값이 플래시 메모리에 복사되지 않는 모터 데이터 구조체 내의 편리한 설정 요소에 대해서 설명합니다.

Motor[x].CapturePos

설명: 모터 위치 포착 모니터링 제어

범위: 0 .. 1

단위: 부울 형식

전원 켜짐 시의 기본값: 0

[Motor[x].CapturePos]는 모터의 위치 포착 모니터링 기능 상태를 지정합니다. 1로 설정하면 Power PMAC은 위치 포착 트리거 이벤트의 모든 실시간 인터럽트에 관한 모터 상태를 감시합니다. 모터가 Servo IC 중 하나로 하드웨어 위치 포착을 위해서 설정되면 포착 위치의 레지스터를 더미로 읽어서 트리거 회로를 다시 작동 상태로 합니다. 이벤트를 검출하면 이벤트가 포착한 센서 위치를 읽습니다. 그 위치를 모터 위치로 변환하고 그 결과를 Status 요소 [Motor[x].CapturedPos]에 저장합니다. 다음으로 [Motor[x].CapturePos]를 자동으로 0으로 반환하고 태스크 완료를 나타냅니다.

이 위치 포착 모니터링 기능은 모터 모션에 전혀 영향을 주지 않습니다. 이 기능은 오픈 루프 또는 정지 상태의 모터에서도 동작하지만 0보다 큰 [Motor[x].ServoCtrl]에서 모터를 기동해야 합니다. 모터가 원점 서치 동작 등 트리거까지의 동작 기능에서 트리거를 검색하는 경우에는 간섭 가능성이 있으므로 사용하지 마십시오.

포착 트리거 조건의 지정 및 포착 위치의 값 처리는 트리거까지의 동작 기능과 같습니다. 이 지정에 대해서는 "사용 설명서"의 '*각 모터 동작의 실행*' 장의 트리거가 있는 모터 동작에 관한 섹션에서 자세하게 설명합니다.

[Motor[x].CaptureMode]는 모터 위치 포착 모니터링 기능의 전체적인 기능을 지정합니다. 입력 트리거에서 포착이 발생하도록 지정하는 경우에 [Motor[x].pCaptFlag] 및 [Motor[x].CaptFlagBit]는 모터가 트리거 비트를 검색할 장소를 지정합니다. 이 비트가 Servo IC 중 하나인 경우에 [Gaten[j].Chan[j].CaptCtrl] 및 [Gaten[j].Chan[j].CaptFlagSel]에 따라서 트리거 이벤트의 원인이 되는 상태와 입력이 특정됩니다.

하드웨어 위치 포착이 지정된 경우에 [Motor[x].CaptPosRightShift], [Motor[x].CaptPosLeftShift] 및 [Motor[x].CaptPosRound]에 따라서 하드웨어에서 포착된 센서 위치를 처리하고 피드백 위치와 일치하는 방법이 설정됩니다.

[Motor[x].CapturePos]는 2014년 1/4 분기에 출시된 V1.6 펌웨어의 새 기능입니다.

Motor[x].PhaseFindingStep

설명: 모터 전류(commutation)의 페이즈 검출 상태

범위: 0 .. 7

단위: 열거

전원 셋업 시의 기본값: 0

[Motor[x].PhaseFindingStep]은 Power PMAC 에 따라서 전류(commutation)되는 동기화 모터의 위상 검출 상태를 지정합니다. 1 로 설정하면 페이즈 검출 동작 또는 절대 페이즈 위치 읽기를 불문하고 페이즈 검출 알고리즘이 시작됩니다. 이것은 온라인 \$명령의 발행에 상당하지만 프로그램 내부에서 직접 실행할 수 있습니다.

페이즈 검출이 종료되면 Power PMAC 은 [Motor[x].PhaseFindingStep]을 자동으로 0 으로 반환합니다. 이것을 사용하여 페이즈 검출이 언제 완료했는지 검출할 수 있습니다. 페이즈 검출 방법에 따라서 페이즈 검출 프로세스 중에 제로 이외의 값으로 설정할 수 있는 경우도 있습니다. Power PMAC 은 페이즈 검출이 정상적으로 실행되지 않았다고 인식하면 페이즈 검출 프로세스 종료 시에 Status 비트 [Motor[x].PhaseFound]를 그대로 0 으로 둡니다.

[Motor[x].PhaseFindingStep]을 8 로 설정하면 Power PMAC 은 [Motor[x].CurrentNullPeriod] > 0 인 경우에 페이즈 검출을 실행하지 않고 전류 루프의 자동 Null 처리를 실행합니다. Null 처리가 완료되면 Status 비트는 자동으로 0 으로 복귀합니다.

Motor[x].PhaseTableBias

설명: 모터 전류(commutation)의 페이즈 위치 상태

범위: -2048 .. 2047

단위: 1/2,048 전류(commutation) 사이클

전원 셋업 시의 기본값: 0

[Motor[x].PhaseTableBias]는 전류(commutation) 알고리즘(디지털 전류 루프의 유무에 상관없이)의 로터 페이즈 위치의 사용자 설정 가능한 오프셋 값을 지정합니다. 1 사이클의 1/2,048 단위의 내부 전류(commutation) 각도 로터 각도에 스케일링된 뒤에 로터 각도의 센서에서 계산되는(및 유도 모터에 의한 슬립 계산) 값에 자동으로 더해집니다.

이 값은 커스터마이즈 페이즈 전진 알고리즘이나 페이즈 검출 알고리즘의 정밀도의 대화형 테스트와 동일하게 표준적인 알고리즘에 대한 특수한 조정용을 목적으로 합니다.

Motor[x].VaBias

설명: 모터의 다이렉트 PWM의 최초의 페이즈 출력 오프셋

범위: -32,768 .. 32,767

단위: PWM 출력 유니트

전원 셋업 시의 기본값: 0.0

[Motor[x].VaBias]는 디지털 전류 루프의 최초의 페이즈 명령 출력의 사용자 설정 가능한 오프셋 값을 지정합니다. [Motor[x].PwmSf]에 따라 스케일링되는 PWM 출력 레지스터를 단위로 합니다. 다이렉트 PWM 전류 루프 알고리즘에 따라 페이즈에 대하여 계산된 값에 자동으로 더합니다. 최종적인 최초 페이즈의 명령은 [Motor[x].pDac]에 따라서 주소가 지정되는 레지스터에 씁니다. 보통 이것은 A 상 PWM 출력 명령입니다.

이 항은 보다 높은 고조파의 보정과 동일하게 표준적인 알고리즘에 대한 특수한 조정용을 목적으로 합니다.

2 번째의 페이즈(Motor[x].VbBias)에는 상당하는 오프셋 값이 있지만 3 번째 페이즈에는 없습니다. 그 이유는 후속하는 내장된 알고리즘(3 차 고조파 주입이라고 합니다)이 페이즈 간의 상대적인 전압을 유지하면서 사용 가능한 버스 전압의 최적 사용에 3 개의 페이즈 전압 명령을 자동으로 중앙에 위치시키기 때문입니다.

Motor[x].VbBias

설명: 모터의 다이렉트 PWM의 제 2 상의 출력 오프셋

범위: -32,768 .. 32,767

단위: PWM 출력 유니트

전원 셋업 시의 기본값: 0.0

[Motor[x].VbBias]는 디지털 전류 루프의 제 2 페이즈의 명령 출력용으로 사용자 설정 가능한 오프셋 값을 지정합니다. [Motor[x].PwmSf]에 따라 스케일링되는 PWM 출력 레지스터를 단위로 합니다. 다이렉트 PWM 전류 루프 알고리즘에 따라 페이즈에 대하여 계산된 값에 자동으로 더합니다. 최종적인 제 2 상의 명령은 [Motor[x].pDac]에 따라서 지정된 레지스터 중 다음으로 우선도가 높은 주소 지정된 레지스터에 씁니다. 보통 이것은 B 상 PWM 출력 명령입니다.

이 항은 보다 높은 고조파의 보정과 동일하게 표준적인 알고리즘에 대한 특수한 조정용을 목적으로 합니다.

최초 페이즈(Motor[x].VaBias)에는 상당하는 오프셋 값이 있지만 3 번째 페이즈에는 없습니다. 그 이유는 후속하는 내장된 알고리즘(3 차 고조파 주입이라고 합니다)이 페이즈 간의 상대적인 전압을 유지하면서 사용 가능한 버스 전압의 최적 사용에 3 개의 페이즈 전압 명령을 자동으로 중앙에 위치시키기 때문입니다.

Muxlo.비저장 데이터 구조체 요소

[Muxlo.]데이터 구조체는 Power PMAC 의 멀티 플렉스 I/O(섬유펜 포트) 설정으로 구성됩니다. 이 구조체의 요소에 따라 ACC-34A 나 ACC-34AA 등 시리얼 섬유펜 멀티플렉서 프로토콜을 사용하는 멀티 플렉스 I/O 장치에 액세스할 수 있습니다.

Muxlo.PortA[n].Data / Muxlo.PortB[n].Data

설명: 멀티 플렉스 I/O 이미지 워드

범위: \$00000000 .. \$FFFFFFFF (0 .. 4,294,967,295)

단위: 비트 필드

전원 셋업 시의 기본값: 0

기존 'I' 변수: 없음

[Muxlo.PortA[n].Data]/[Muxlo.PortB[n].Data]는 DIP 스위치로 선택된 주소가 **n** 의 장치의 멀티 플렉스 I/O 포트 A/B 의 Power PMAC 메모리 내의 이미지 워드입니다. [Muxlo.PortA[n].Dir]/[Muxlo.PortB[n].Dir]의 설정에 따라서 이 값은 읽히거나 씁집니다. 이 이미지 워드가 액세스되는 간격은 [Muxlo.UpdatePeriod]의 설정에 따라 다릅니다.

I/O बैं크의 각 I/O 포인트는 구문 [Muxlo.PortA[n]. Data.k]/[Muxlo.PortB[n]. Data.k]를 사용하여 참조할 수 있습니다. 여기서 **k** 는 레지스터의 비트 번호를 나타내는 0~31 인 정수 상수입니다. 이것은 'M' 변수 포인터를 I/O 포인트에 할당할 때 특히 편리합니다. 예시는 다음과 같습니다.

ptr ErrorLight->Muxlo.PortB[1].Data.18

Plc[i].비저장 데이터 구조체 요소

[Plc[i]] 데이터에는 스크립트 프로그램 PLC*i*에 관한 요소가 포함됩니다.

Plc[i].Ldata.비저장 로컬 데이터 요소

[Plc[i].Ldata.] 서브 구조체에는 다양한 PLC 프로그램의 계산에 로컬라이즈된 요소가 포함됩니다. 좌표계 및 통신 스레드에는 동일한 로컬 데이터의 서브 구조체가 있습니다. 이 때문에 해당 서브 구조체는 이 장에서는 공통 [Ldata.] 섹션에 기재되었습니다.

PowerBrick[i].비저장 데이터 구조체 요소

데이터 구조체명인 [PowerBrick[i]]는 기본적인 데이터 구조체 [Gate3[i]]의 스크립트 환경에 대한 별칭입니다. [Gate3[i]] 데이터 구조체의 항목으로 Power Brick 서보 인터페이스 보드의 데이터 구조체 요소를 목록화하여 후술합니다.

Sys.글로벌한 비저장 설정 데이터 구조체 요소

이 섹션에서는 **save** 명령으로 값이 플래시 메모리에 복사되지 않는 글로벌 설정 요소에 대해서 설명합니다.

Sys.Cdata[i]

설명: 사용자 공유 메모리 문자(바이트) 데이터 배열 요소

범위: 0 .. 255

단위: 사용자 지정

전원 켜짐 시의 기본값: 0

[Sys.Cdata[i]]는 사용자의 공유 메모리 버퍼 내의 'i' 번째의 부호가 없는 8 비트 정수 데이터 배열 요소입니다. 이 각 요소는 사용자의 공유 메모리 버퍼 내에서 1 바이트를 차지하고 [Sys.pushm] 내의 주소에 배치된 버퍼의 시작에서 'i' 번째의 주소부터 시작됩니다. 이 배열은 어플리케이션에서 범용 목적으로 사용됩니다. 자동으로 사용되지 않습니다.

대괄호로 묶인 인덱스값 'i'는 0~16,777,215 범위인 정수 상수 또는 로컬 'L' 변수입니다. 식 또는 정수 이외의 상수는 허락되지 않습니다. pp_proj.ini 파일에서 명령으로 설정되는 사용자 공유 메모리 버퍼의 크기는 기본값일 때는 1MB(1,048,576 바이트)입니다. 이 때문에 최대 유효한 인덱스값은 262,143 이 됩니다.

값은 단일 인용 부호가 있는 리터럴 문자(Sys.Cdata[170] = 'A' 등) 또는 정수값(Sys.Cdata[170] = 65 또는 Sys.Cdata[170] = 65 + LowerCase*32 등)으로 [Sys.Cdata[i]]요소에 할당할 수 있습니다. Power PMAC 은 쿼리에 대한 값을 정수 상수로 통지합니다.

연속하는 [Sys.Cdata[i]] 요소는 Power PMAC 스크립트 문자열의 조작 함수에서 사용할 수 있습니다.

[Sys.Cdata[i]]는 [Sys.Ddata[i/8]], [Sys.Fdata[i/4]], [Sys.Ldata[i/4]], [Sys.Udata[i/4]] 및 [Sys.Uhex[i/4]]와 같은 레지스터에 배치됩니다. 같은 레지스터를 여러 용도로 사용하지 않도록 주의하십시오.

C 에서 이 요소는 char 변수의 포인터로 pushm 기준 주소에서 i 주소 오프셋하여 액세스해야 합니다(MyCharPtr = (char *) pushm + i).

Sys.Ddata[i]

설명: 사용자 공유 메모리 double 형 데이터 배열 요소

범위: 배정도 부동 소수점

단위: 사용자 지정

전원 켜짐 시의 기본값: 0.0

[Sys.Ddata[*i*]]는 사용자의 공유 메모리 버퍼 내의 '*i*' 번째의 배정도 부동 소수점의 데이터 배열 요소입니다. 이 각 요소는 사용자의 공유 메모리 버퍼 내에서 8 바이트를 차지하고 [Sys.pushm] 내의 주소에 배치된 버퍼의 시작에서 $8*i$ 번째의 주소부터 시작됩니다. 이 배열은 어플리케이션에서 범용 목적으로 사용됩니다. 자동으로 사용되지 않습니다.

대괄호로 묶인 인덱스값 '*i*'는 0~16,777,215 범위인 정수 상수 또는 로컬 'L' 변수입니다. 식 또는 정수 이외의 상수는 허락되지 않습니다. pp_proj.ini 파일에서 명령으로 설정되는 사용자 공유 메모리 버퍼의 크기는 기본값일 때는 1MB(1,048,576 바이트)입니다. 이 때문에 최대 유효한 인덱스값은 131,071 이 됩니다.

[Sys.Ddata[*i*]]는 [Sys.Cdata[$i/8$]]부터 [Sys.Cdata[$i/8+7$]], [Sys.Fdata[$2*i$]], [Sys.Fdata[$2*i+1$]], [Sys.Ldata[$2*i$]], [Sys.Ldata[$2*i+1$]], [Sys.Udata[$2*i$]] 및 [Sys.Udata[$2*i+1$]]과 같은 레지스터 내에 배치됩니다. 같은 레지스터를 여러 용도로 사용하지 않도록 주의하십시오.

C에서는 이 요소는 double 변수의 포인터로 pushm 기준 주소에서 *i* 주소 오프셋하여 액세스해야 합니다(MyDoublePtr = (double *) pushm + *i*).

Sys.Fdata[*i*]

설명: 사용자 공유 메모리 float 형 데이터 배열 요소

범위: 단정도 부동 소수점

단위: 사용자 지정

전원 셋업 시의 기본값: 0.0

[Sys.Fdata[*i*]]는 사용자의 공유 메모리 버퍼 내의 '*i*' 번째의 단정도 부동 소수점의 데이터 배열 요소입니다. 이 각 요소는 사용자의 공유 메모리 버퍼 내에서 4 바이트를 차지하고 [Sys.pushm] 내의 주소에 배치된 버퍼의 시작에서 $4*i$ 번째 주소부터 시작됩니다. 이 배열은 어플리케이션에서 범용 목적으로 사용됩니다. 자동으로 사용되지 않습니다.

대괄호로 묶인 인덱스값 '*i*'는 0~16,777,215 범위인 정수 상수 또는 로컬 'L' 변수입니다. 식 또는 정수 이외의 상수는 허락되지 않습니다. pp_proj.ini 파일에서 명령으로 설정되는 사용자 공유 메모리 버퍼의 크기는 기본값일 때는 1MB(1,048,576 바이트)입니다. 이 때문에 최대 유효한 인덱스값은 262,143 이 됩니다.

[Sys.Fdata[*i*]]는 [Sys.Ddata[$i/2$]], [Sys.Ldata[*i*]] 및 [Sys.Udata[*i*]]와 같은 레지스터에 배치됩니다. 같은 레지스터를 여러 용도로 사용하지 않도록 주의하십시오.

[Sys.Fdata[*i*]]는 [Sys.Cdata[$i/4$]]부터 [Sys.Cdata[$i/4+3$]], [Sys.Ddata[$i/2$]], [Sys.Ldata[*i*]], [Sys.Udata[*i*]] 및 [Sys.Uhex[*i*]]와 같은 레지스터에 배치됩니다. 같은 레지스터를 여러 용도로 사용하지 않도록 주의하십시오.

C에서는 이 요소는 float 변수의 포인터로 pushm 기준 주소에서 *i* 주소 오프셋하여 액세스해야 합니다(MyFloatPtr = (float *) pushm + *i*).

Sys.Idata[*i*]

설명: 사용자 공유 메모리 부호가 있는 정수 데이터 배열 요소

범위: $-2^{31} \dots 2^{31}-1$

단위: 사용자 지정

전원 셋업 시의 기본값: 0

[Sys.Idata[*i*]]는 사용자의 공유 메모리 버퍼 내의 '*i*' 번째의 부호가 있는 32 비트 정수 데이터 배열 요소입니다. 이 각 요소는 사용자의 공유 메모리 버퍼 내에서 4 바이트를 차지하고 [Sys.pushm] 내의 주소에 배치된 버퍼의 시작에서 $4*i$ 번째 주소부터 시작됩니다. 이 배열은 어플리케이션에서 범용 목적으로 사용됩니다. 자동으로 사용되지 않습니다.

대괄호로 묶인 인덱스값 '*i*'는 0~16,777,215 범위인 정수 상수 또는 로컬 'L' 변수입니다. 식 또는 정수 이외의 상수는 허락되지 않습니다. pp_proj.ini 파일에서 명령으로 설정되는 사용자 공유 메모리 버퍼의 크기는 기본값일 때는 1MB(1,048,576 바이트)입니다. 이 때문에 최대 유효한 인덱스값은 262,143 이 됩니다.

[Sys.Idata[*i*]]는 [Sys.Ddata[*i*/2]], [Sys.Fdata[*i*]] 및 [Sys.Udata[*i*]]와 같은 레지스터에 배치됩니다. 같은 레지스터를 여러 용도로 사용하지 않도록 주의하십시오.

[Sys.Idata[*i*]]는 [Sys.Cdata[*i*/4]]부터 [Sys.Cdata[*i*/4+3]], [Sys.Ddata[*i*/2]], [Sys.Fdata[*i*]], [Sys.Udata[*i*]] 및 [Sys.Uhex[*i*]]와 같은 레지스터에 배치됩니다. 같은 레지스터를 여러 용도로 사용하지 않도록 주의하십시오.

C에서는 이 요소는 int 변수의 포인터로 pushm 기준 주소에서 *i* 주소 오프셋하여 액세스해야 합니다(MyIntPtr = (int *) pushm + *i*).

Sys.Lock[*i*]

설명: 잠금 제어 비트 프로세스 *i*

범위: 0 .. 1

단위: 부울 형식

전원 셋업 시의 기본값: 0

[Sys.Lock[*i*]]는 스크립트 프로그램의 조작으로 사용자 실행 프로세스 *i*의 제어 및 검출을 허가하는 싱글 비트의 요소입니다. 프로세스 잠금은 32 이고 인덱스 '*i*'는 0 부터 31 입니다.

[Sys.Lock[*i*]]에서 읽기 액세스는 Status 비트의 현재 값을 반환합니다. 값이 1 인 경우에 태스크(보통 사용에서는 다른 태스크)가 프로세스 *i*를 제어합니다.

0 이 반환된 경우에는 태스크가 이 스레드를 제어하지 않습니다. 0 이 반환된 경우에 자동으로 비트를 즉시 읽는 어토믹한 동작에 따라 비트를 1 로 설정하고 이 태스크가 스레드를 확실하게 제어할 수 있게 합니다.

[Sys.Lock[*i*]]를 0 으로 설정하는 쓰기 액세스는 허가됩니다. 보통 사용에서는 현재 스레드를 제어하는 태스크에서 이것을 실행하십시오. 단, 실행한 모든 태스크에서 에러가 발생한 경우에 관리 태스크가 스레드를 릴리스하는 것은 가능합니다.

[Sys.Lock[*i*]]를 1로 설정하는 쓰기 액세스는 허가되지 않습니다. 읽기 액세스가 비트를 1로 설정하는 유일한 방법입니다.

스크립트 프로그램은 프로세스의 제어를 대기하고 제어합니다. 그리고 다음 코드를 사용하여 제어를 해제합니다.

```
while (Sys.Lock[4] == 1) {}           // Wait until process 4 free
{Code that uses control of process 4}
Sys.Lock[4] = 0;                     // Release process 4 control
```

비트가 1로 설정되어 있는 한 다른 태스크가 프로세스를 잠그고 있다고 가정하고 프로그램은 while 루프에서 나올 수 없습니다. 이 예시에서는 스레드의 제어를 대기하는 동안에 아무것도 하지 않는 비어 있는 루프가 있지만 대기한 루프 내에서 다른 태스크를 실행할 수 있습니다. 루프는 [Sys.Lock[4]]가 0임을 검출한 경우에 조건 비교를 위한 읽기 동작에 따라서 이 요소는 자동으로 1로 설정됩니다. 그 때문에 이 비트를 참조하는 다른 태스크는 이 프로그램이 프로세스의 제어를 사용하는 태스크를 실행 중에는 프로세스가 잠겨 있다는 것을 확인합니다. 이 태스크가 종료하면 프로세스의 제어는 해제됩니다.

다른 경우에는 스크립트 프로그램은 다음과 같은 코드를 사용할 가능성이 있습니다.

```
if (Sys.Lock[15] == 0) {               // Process 15 free?
{Code that uses control of process 15}
Sys.Lock[15] = 0;                     // Release process 15 control
}
```

프로그램은 [Sys.Lock[15]]가 0임을 검출한 경우에 조건 비교를 위한 읽기 동작에 따라서 이 요소는 자동으로 1로 설정됩니다. 그 때문에 이 비트를 참조하는 다른 태스크는 이 프로그램이 프로세스의 제어를 사용하는 태스크를 실행 중에는 프로세스가 잠겨 있다는 것을 확인합니다. 이 태스크가 종료하면 프로세스의 제어는 해제됩니다. 프로그램은 [Sys.Lock[15]]가 1임을 검출한 경우에는 다른 프로그램이 이 실행 프로세스를 잠근 것을 인식하고 이 태스크를 건너뛵니다.

[Sys.Lock[*i*]]의 현재 상태는 32비트의 Status 요소 [Sys.Lock]을 읽고 그 요소의 비트 *i* 값을 평가하여 그 상태를 변경하지 않고 읽을 수 있습니다.

Sys.M[*i*]

설명: 'M' 변수 배열 요소

범위: 정의에 좌우됨

단위: 사용자 정의

전원 셋업 시의 기본값: 정의에 좌우됨

[Sys.M[*i*]]는 '*i*'번째의 'M' 변수 배열 요소입니다. 'M' 변수는 메모리와 I/O 레지스터를 주소 지정할 수 있는 글로벌 포인트 사용자 변수입니다. 인덱스값 '*i*'는 0부터 65,535의 범위입니다. 이 배열은 'M' 변수의 액세스 대체 방법을 제공합니다.

이 요소는 C에서 직접 액세스할 수 없습니다. C에서는 대상이 되는 레지스터는 포인터 변수로 직접 주소 지정합니다.

Sys.P[i]

설명: 'P' 변수 배열 요소

범위: 부동 소수점

단위: 사용자 정의

전원 셋업 시의 기본값: 0

[Sys.P[i]]는 *i*번째의 'P' 변수 배열 요소입니다. 'P' 변수는 글로벌한 배정도 부동 소수점의 사용자 변수입니다. 인덱스값 *i*는 0 부터 65,535 의 범위입니다. 이 배열은 'P' 변수의 액세스 대체 방법을 제공합니다.

C에서는 이 요소는 [pshm->P[i]]로 액세스합니다.

Sys.Udata[i]

설명: 사용자 공유 메모리 부호가 없는 정수 데이터 배열 요소

범위: 0 .. $2^{32}-1$

단위: 사용자 지정

전원 셋업 시의 기본값: 0

[Sys.Udata[i]]는 사용자의 공유 메모리 버퍼 내의 *i*번째의 부호가 없는 32 비트 정수 데이터 배열 요소입니다. 이 각 요소는 사용자의 공유 메모리 버퍼 내에서 4 바이트를 차지하고 [Sys.pushm] 내의 주소에 배치된 버퍼의 시작에서 $4*i$ 번째 주소부터 시작됩니다. 이 배열은 어플리케이션에서 범용 목적으로 사용됩니다. 자동으로 사용되지 않습니다.

[Sys.Udata[i]]는 [Sys.Uhex[i]]와 같지만 [Sys.Udata]인 경우에 스크립트 환경에서 값의 쿼리에 대하여 Power PMAC 은 값을 16 진법(\$100 등)가 아니라 10 진법으로(256 등) 통지합니다.

대괄호로 묶인 인덱스값 *i*는 0~16,777,215 범위인 정수 상수 또는 로컬 'L' 변수입니다. 식 또는 정수 이외의 상수는 허락되지 않습니다. pp_proj.ini 파일에서 명령으로 설정되는 사용자 공유 메모리 버퍼의 크기는 기본값일 때는 1MB(1,048,576 바이트)입니다. 이 때문에 최대 유효한 인덱스값은 262,143 이 됩니다.

[Sys.Udata[i]]는 [Sys.Cdata[i/4]]부터 [Sys.Cdata[i/4+3]], [Sys.Ddata[i/2]], [Sys.Fdata[i]], [Sys.Ldata[i]] 및 [Sys.Uhex[i]]와 같은 레지스터에 배치됩니다. 같은 레지스터를 여러 용도로 사용하지 않도록 주의하십시오.

C에서는 이 요소는 unsigned int 변수의 포인터로 pushm 기준 주소에서 *i* 주소 오프셋하여 액세스해야 합니다(MyUIntPtr = (int *) pushm + i).

Sys.Uhex[*i*]

설명: 사용자 공유 메모리 부호가 없는 16 진법의 정수 데이터 배열 요소

범위: 0 .. $2^{32}-1$

단위: 사용자 지정

전원 셋업 시의 기본값: 0

[Sys.Uhex[*i*]]는 사용자의 공유 메모리 버퍼 내의 '*i*' 번째의 부호가 없는 32 비트 정수 데이터 배열 요소입니다. 이 각 요소는 사용자의 공유 메모리 버퍼 내에서 4 바이트를 차지하고 [Sys.pushm] 내의 주소에 배치된 버퍼의 시작에서 $4*i$ 번째 주소부터 시작됩니다. 이 배열은 어플리케이션에서 범용 목적으로 사용됩니다. 자동으로 사용되지 않습니다.

[Sys.Uhex[*i*]]는 [Sys.Udata[*i*]]와 같지만 [Sys.Udata]인 경우에 스크립트 환경에서 값의 쿼리에 대하여 Power PMAC 은 값을 10 진법(256 등)가 아니라 16 진법(\$100 등)로 통지합니다.

대괄호로 묶인 인덱스값 '*i*'는 0~16,777,215 범위인 정수 상수 또는 로컬 'L' 변수입니다. 식 또는 정수 이외의 상수는 허락되지 않습니다. pp_proj.ini 파일에서 명령으로 설정되는 사용자 공유 메모리 버퍼의 크기는 기본값일 때는 1MB(1,048,576 바이트)입니다. 이 때문에 최대 유효한 인덱스값은 262,143 이 됩니다.

[Sys.Uhex[*i*]]는 [Sys.Cdata[*i*/4]]부터 [Sys.Cdata[*i*/4+3]], [Sys.Ddata[*i*/2]], [Sys.Fdata[*i*]], [Sys.Idata[*i*]] 및 [Sys.Udata[*i*]]와 같은 레지스터에 배치됩니다. 같은 레지스터를 여러 용도로 사용하지 않도록 주의하십시오.

C 에서는 이 요소는 unsigned int 변수의 포인터로 pushm 기준 주소에서 *i* 주소 오프셋하여 액세스해야 합니다(MyUIntPtr = (int *) pushm + *i*).

Sys.WpKey

설명: DSPGATE3 ASIC 의 쓰기 방지 키의 값

범위: 0 .. \$FFFFFFFF

단위: 없음

전원 셋업 시의 기본값: 0

[Sys.WpKey]는 스크립트 명령이 ASIC 의 쓰기 방지된 레지스터에 대하여 쓰기 조작을 실행하면 DSPGATE3 머신 인터페이스 ASIC *i* 의 쓰기 방지 키의 레지스터[Gate3[*i*].WpKey]에 자동으로 복사되는 값을 지정합니다. 스크립트 언어로 쓰기 방지된 레지스터에 쓰기 조작을 문제없이 실행하려면 [Sys.WpKey]를 \$AAAAAAAA 로 설정해야 합니다. 이러한 쓰기 조작 뒤에 ASIC 의 쓰기 방지 키의 레지스터는 자동으로 소거되어 0 이 되지만 이 요소의 값은 변하지 않습니다.

C 언어의 명령이 이러한 ASIC 의 쓰기 방지된 레지스터에 쓰는 경우에 이 요소는 사용되지 않습니다. C에서는 프로그램은 ASIC의 쓰기 방지된 레지스터에 쓰기 조작을 실행하기 전에 매번 [Gate3[i].WpKey] 레지스터에 키의 값을 명시적으로 써야 합니다.

이 요소에 따라 머신 개발자는 개발 중에 이 쓰기 방지된 레지스터를 간단하게 변경할 수 있지만 권한이 없는 사용자가 그것을 변경할 수 없습니다.

Tdata[i].비저장 설정 데이터 구조체 요소

이 섹션에서는 **save** 명령으로 값이 플래시 메모리에 복사되지 않는 모터 변환 매트릭스 구조체 내의 편리한 설정 요소에 대해서 설명합니다.

Tdata[i].Bias[m]

설명: 변환 매트릭스의 오프셋 항

범위: 부동 소수점

단위: (미변환)축 단위

전원 셋업 시의 기본값: 0.0

[Tdata[i].Bias[m]]은 인덱스 '*m*'을 가지는 축의 변환 매트릭스 *i*의 오프셋 항입니다. 그 값은 미변환의 기본축 위치를 얻기 위해서 프로그램된(변환된 것) 축 위치에 더해집니다. 변환 초기화의 프로그램 명령 **tinit*i***에 따라 변환 매트릭스 *i*의 모든 축의 바이어스 항은 0.0으로 설정됩니다.

256의 변환 매트릭스에 대한 인덱스의 값 '*i*'는 0부터 255의 범위입니다. 32의 축에 대한 인덱스의 값 '*m*'은 0부터 31의 범위입니다. 32의 축 명칭에 대한 축 인덱스 '*m*'을 다음 표에 나타냅니다.

축	<i>m</i>	축	<i>m</i>	축	<i>m</i>	축	<i>m</i>	축	<i>m</i>	축	<i>m</i>	축	<i>m</i>	축	<i>m</i>
A	0	V	4	Z	8	DD	12	HH	16	OO	20	SS	24	WW	28
B	1	W	5	AA	9	EE	13	LL	17	PP	21	TT	25	XX	29
C	2	X	6	BB	10	FF	14	MM	18	QQ	22	UU	26	YY	30
U	3	Y	7	CC	11	GG	15	NN	19	RR	23	VV	27	ZZ	31

활성(선택된) 변환 매트릭스의 항이 변경된 경우 또는 새로운 변환 매트릭스가 좌표계를 위해서 선택된 경우에 다음 프로그램 동작이 명령되기 전에 좌표계에 **pmatch** 명령을 실행해야 합니다(모션 프로그램의 실행을 시작하는 명령으로 자동으로 실행됩니다).

세부 내용에 대해서는 "사용 설명서"의 '*좌표계의 설정*' 장의 '*축 변환 매트릭스*' 섹션을 참조하십시오.

Tdata[i].Diag[m]

설명: 변환 매트릭스의 스케일링 항

범위: 부동 소수점

단위: 변환된 축 단위마다 변환되지 않은 축 단위

전원 셋업 시의 기본값: 1.0

[Tdata[i].Diag[m]]은 인덱스 'm'을 가지는 축의 변환 매트릭스 *i*의 (대각)스케일링 항입니다. 그 값에 프로그램된(변환된) 축 위치를 곱합니다. 그 곱은 미변환의 기본축 위치를 얻기 위해서 다른 항에 더합니다. 변환 초기화의 프로그램 명령 **tinit*i***에 따라 변환 매트릭스 *i*의 모든 축의 대각항은 1.0으로 설정됩니다.

256의 변환 매트릭스에 대한 인덱스의 값 '*i*'는 0부터 255의 범위입니다. 32의 축에 대한 인덱스의 값 '*m*'은 0부터 31의 범위입니다. 32의 축 명칭에 대한 축 인덱스 '*m*'을 다음 표에 나타냅니다.

축	<i>m</i>	축	<i>m</i>	축	<i>m</i>	축	<i>m</i>	축	<i>m</i>	축	<i>m</i>	축	<i>m</i>	축	<i>m</i>
A	0	V	4	Z	8	DD	12	HH	16	OO	20	SS	24	WW	28
B	1	W	5	AA	9	EE	13	LL	17	PP	21	TT	25	XX	29
C	2	X	6	BB	10	FF	14	MM	18	QQ	22	UU	26	YY	30
U	3	Y	7	CC	11	GG	15	NN	19	RR	23	VV	27	ZZ	31

활성(선택된) 변환 매트릭스의 항이 변경된 경우 또는 새로운 변환 매트릭스가 좌표계를 위해서 선택된 경우에 다음 프로그램 동작이 명령되기 전에 좌표계에 **pmatch** 명령을 실행해야 합니다(모션 프로그램의 실행을 시작하는 명령으로 자동으로 실행됩니다).

세부 내용에 대해서는 "사용 설명서"의 '*좌표계의 설정*' 장의 '*축 변환 매트릭스*' 섹션을 참조하십시오.

Tdata[i].UUVVWW[m]

설명: 변환 매트릭스 UU/VV/WW 축의 크로스 커플링 항

범위: 부동 소수점

단위: 변환된 축 단위마다 변환되지 않은 축 단위

전원 셋업 시의 기본값: 0.0

[Tdata[i].UUVVWW[m]]은 UU/VV/WW 축 트리플릿의 변환 매트릭스 *i*의(비대각) 크로스 커플링 항 중 하나입니다. 이 축 트리플릿의 3x3 크로스 커플링의 서브 매트릭스에는 비대각항이 6개 있습니다. 비대각항의 값에는 트리플릿 중 하나가 변환된 위치를 곱합니다. 결과의 값은 트리플릿의 다른 축의 변환되지 않은 위치의 값에 더합니다. 크로스 커플링 항은 주로 3D 축 공간에서 회전에 사용됩니다.

변환 매트릭스는 256, 인덱스 '*i*'는 0부터 255입니다.

트리플릿의 완전한 서브 매트릭스를 다음 매트릭스 방정식에 나타냅니다. 여기서 좌측 축 명칭이 변환되지 않은 위치고 우측 축 명칭이 변환된 위치입니다.

$$\begin{bmatrix} UU \\ VV \\ WW \end{bmatrix} = \begin{bmatrix} \text{Diag}[26] & UUVVWW[0] & UUVVWW[1] \\ UUVVWW[2] & \text{Diag}[27] & UUVVWW[3] \\ UUVVWW[4] & UUVVWW[5] & \text{Diag}[28] \end{bmatrix} \begin{bmatrix} UU' \\ VV' \\ WW' \end{bmatrix} + \begin{bmatrix} \text{Bias}[26] \\ \text{Bias}[27] \\ \text{Bias}[28] \end{bmatrix}$$

변환 초기화의 프로그램 명령 **tinit*i***에 따라 변환 매트릭스 *i*의 모든 축 트리플릿의 비대각항은 0.0으로 설정됩니다.

활성(선택된) 변환 매트릭스의 항이 변경된 경우 또는 새로운 변환 매트릭스가 좌표계를 위해서 선택된 경우에 다음 프로그램 동작이 명령되기 전에 좌표계에 **pmatch** 명령을 실행해야 합니다(모션 프로그램의 실행을 시작하는 명령으로 자동으로 실행됩니다).

세부 내용에 대해서는 "사용 설명서"의 '*좌표계의 설정*' 장의 '*축 변환 매트릭스*' 섹션을 참조하십시오.

Tdata[i].UVW[m]

설명: 변환 매트릭스 U/V/W 축의 크로스 커플링 항

범위: 부동 소수점

단위: 변환된 축 단위마다 변환되지 않은 축 단위

전원 셋업 시의 기본값: 0.0

[Tdata[i].UVW[m]]은 U/V/W 축 트리플렛의 변환 매트릭스 *i*의(비대각) 크로스 커플링 항 중 하나입니다. 이 축 트리플렛의 3x3 크로스 커플링의 서브 매트릭스에는 비대각항이 6개 있습니다. 비대각항의 값에는 트리플렛 중 하나가 변환된 위치를 곱합니다. 결과의 값은 트리플렛의 다른 축의 변환되지 않은 위치의 값에 더합니다. 크로스 커플링 항은 주로 3D 축 공간에서 회전에 사용됩니다.

변환 매트릭스는 256, 인덱스 '*i*'는 0 부터 255 입니다.

트리플렛의 완전한 서브 매트릭스를 다음 매트릭스 방정식에 나타냅니다. 여기서 좌측 축 명칭이 변환되지 않은 위치고 우측 축 명칭이 변환된 위치입니다.

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} Diag[3] & UVW[0] & UVW[1] \\ UVW[2] & Diag[4] & UVW[3] \\ UVW[4] & UVW[5] & Diag[5] \end{bmatrix} \begin{bmatrix} U' \\ V' \\ W' \end{bmatrix} + \begin{bmatrix} Bias[3] \\ Bias[4] \\ Bias[5] \end{bmatrix}$$

변환 초기화의 프로그램 명령 **tinit*i***에 따라 변환 매트릭스 *i*의 모든 축 트리플렛의 비대각항은 0.0으로 설정됩니다.

활성(선택된) 변환 매트릭스의 항이 변경된 경우 또는 새로운 변환 매트릭스가 좌표계를 위해서 선택된 경우에 다음 프로그램 동작이 명령되기 전에 좌표계에 **pmatch** 명령을 실행해야 합니다(모션 프로그램의 실행을 시작하는 명령으로 자동으로 실행됩니다).

세부 내용에 대해서는 "사용 설명서"의 '*좌표계의 설정*' 장의 '*축 변환 매트릭스*' 섹션을 참조하십시오.

Tdata[i].XXYYZZ[m]

설명: 변환 매트릭스 XX/YY/ZZ 축의 크로스 커플링 항

범위: 부동 소수점

단위: 변환된 축 단위마다 변환되지 않은 축 단위

전원 셋업 시의 기본값: 0.0

[Tdata[i].XXYYZZ[m]]은 XX/YY/ZZ 축 트리플렛의 변환 매트릭스 *i* 의(비대각) 크로스 커플링 항 중 하나입니다. 이 축 트리플렛의 3x3 크로스 커플링의 서브 매트릭스에는 비대각항이 6 개 있습니다. 비대각항의 값에는 트리플렛 중 하나가 변환된 위치를 곱합니다. 결과의 값은 트리플렛의 다른 축의 변환되지 않은 위치의 값에 더합니다. 크로스 커플링 항은 주로 3D 축 공간에서 회전에 사용됩니다.

변환 매트릭스는 256, 인덱스 '*i*'는 0 부터 255 입니다.

트리플렛의 완전한 서브 매트릭스를 다음 매트릭스 방정식에 나타냅니다. 여기서 좌측 축 명칭이 변환되지 않은 위치고 우측 축 명칭이 변환된 위치입니다.

$$\begin{bmatrix} XX \\ YY \\ ZZ \end{bmatrix} = \begin{bmatrix} Diag[29] & XXYYZZ[0] & XXYYZZ[1] \\ XXYYZZ[2] & Diag[30] & XXYYZZ[3] \\ XXYYZZ[4] & XZYYZZ[5] & Diag[31] \end{bmatrix} \begin{bmatrix} XX' \\ YY' \\ ZZ' \end{bmatrix} + \begin{bmatrix} Bias[29] \\ Bias[30] \\ Bias[31] \end{bmatrix}$$

변환 초기화의 프로그램 명령 **tinit**에 따라 변환 매트릭스 *i* 의 모든 축 트리플렛의 비대각항은 0.0 으로 설정됩니다.

활성(선택된) 변환 매트릭스의 항이 변경된 경우 또는 새로운 변환 매트릭스가 좌표계를 위해서 선택된 경우에 다음 프로그램 동작이 명령되기 전에 좌표계에 **pmatch** 명령을 실행해야 합니다(모션 프로그램의 실행을 시작하는 명령으로 자동으로 실행됩니다).

세부 내용에 대해서는 "사용 설명서"의 '**좌표계의 설정**' 장의 '**축 변환 매트릭스**' 섹션을 참조하십시오.

Tdata[i].XYZ[m]

설명: 변환 매트릭스 X/Y/Z 축의 크로스 커플링 항

범위: 부동 소수점

단위: 변환된 축 단위마다 변환되지 않은 축 단위

전원 셋업 시의 기본값: 0.0

[Tdata[i].XYZ[m]]은 X/Y/Z 축 트리플렛의 변환 매트릭스 *i* 의(비대각) 크로스 커플링 항 중 하나입니다. 이 축 트리플렛의 3x3 크로스 커플링의 서브 매트릭스에는 비대각항이 6 개 있습니다. 비대각항의 값에는 트리플렛 중 하나가 변환된 위치를 곱합니다. 결과의 값은 트리플렛의 다른 축의 변환되지 않은 위치의 값에 더합니다. 크로스 커플링 항은 주로 3D 축 공간에서 회전에 사용됩니다.

변환 매트릭스는 256, 인덱스 '*i*'는 0 부터 255 입니다.

트리플릿의 완전한 서브 매트릭스를 다음 매트릭스 방정식에 나타냅니다. 여기서 좌측 축 명칭이 변환되지 않은 위치고 우측 축 명칭이 변환된 위치입니다.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} Diag[6] & XYZ[0] & XYZ[1] \\ XYZ[2] & Diag[7] & XYZ[3] \\ XYZ[4] & XYZ[5] & Diag[8] \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} + \begin{bmatrix} Bias[6] \\ Bias[7] \\ Bias[8] \end{bmatrix}$$

변환 초기화의 프로그램 명령 **tiniti**에 따라 변환 매트릭스 *i*의 모든 축 트리플릿의 비대각항은 0.0으로 설정됩니다.

활성(선택된) 변환 매트릭스의 항이 변경된 경우 또는 새로운 변환 매트릭스가 좌표계를 위해서 선택된 경우에 다음 프로그램 동작이 명령되기 전에 좌표계에 **pmatch** 명령을 실행해야 합니다(모션 프로그램의 실행을 시작하는 명령으로 자동으로 실행됩니다).

세부 내용에 대해서는 "사용 설명서"의 '*좌표계의 설정*' 장의 '*축 변환 매트릭스*' 섹션을 참조하십시오.

UserAlgo.비저장 설정 데이터 구조체 요소

이 섹션에서는 **save** 명령으로 값이 플래시 메모리에 복사되지 않는 사용자 알고리즘 제어 구조체 내의 편리한 설정 요소에 대해서 설명합니다.

UserAlgo.BgCplc[i]

설명: 백그라운드의 C PLC 프로그램 *i*의 유효화

범위: 0 .. 1

단위: 부울 형식

전원 켜짐 시의 기본값: 0

[UserAlgo.BgCplc[i]]는 C 언어에 의한 백그라운드의 PLC 프로그램의 CPLC*i*가 유효인지 무효인지 제어합니다. 전원 켜짐/리셋 시의 기본값인 0으로 설정하면 프로그램은 무효가 되고 Power PMAC에 로드되어도 실행되지 않습니다. 1로 설정하면 프로그램은 유효가 되고 Power PMAC의 백그라운드 소프트웨어 사이클마다 실행됩니다.

이 요소의 값은 **enable bgcplc i** 명령으로 1, **disable bgcplc i** 명령으로 0을 설정할 수 있습니다.

전원 켜짐/리셋과 동시에 이 프로그램을 실행하는 경우에는 이 요소의 값을 1로 설정하는 명령을 pp_startup.txt 프로젝트 구성 파일에 포함해야 합니다.

UserAlgo.CaptComplntr

설명: 유효한 인터럽트 서비스 루틴의 포착/비교

범위: 0 .. 1

단위: 부울 형식

전원 켜짐 시의 기본값: 0

[UserAlgo.CaptComplntr]은 C 언어의 포착/비교 인터럽트 서비스 루틴이 유효인지 무효인지 제어합니다. 전원 켜짐/리셋 시의 기본값인 0으로 설정하면 프로그램은 무효가 되고 Power PMAC에 로드되어도 실행되지 않습니다. 1로 설정하면 프로그램은 유효가 되며 PMAC3 스타일 DSPGATE3 IC가 포착 또는 비교 인터럽트를 프로세서에 송신할 때마다 실행됩니다.

전원 켜짐/리셋과 동시에 이 프로그램을 실행하는 경우에는 이 요소의 값을 1로 설정하는 명령을 pp_startup.txt 프로젝트 구성 파일에 포함해야 합니다.

UserAlgo.CFunc

설명: CfromScript 함수의 유효화

범위: 0 .. 1

단위: 부울 형식

전원 셋업 시의 기본값: 0

[UserAlgo.CFunc]은 C 언어의 함수 'CfromScript'를 백그라운드 프로그램에서 유효로 할지 여부를 제어합니다. 전원 셋업/리셋 시의 기본값인 0으로 설정하면 함수는 무효가 되고 Power PMAC에 로드되어도 실행되지 않습니다. 1로 설정하면 함수는 유효가 되고 Power PMAC의 스크립트 프로그램에서 호출될 때마다 실행됩니다.

포어그라운드 PLC 프로그램 및 모션 프로그램에 따라 실행되는 키네마틱스 루틴 등 포어그라운드의 스크립트 프로그램에서 'CfromScript' 함수를 호출하는 경우에는 [UserAlgo.CFunc]을 1로 설정할 필요는 없습니다. 백그라운드 PLC 프로그램 및 온라인 쿼리 명령 또는 백그라운드 PLC 프로그램 명령에 따라 실행되는 키네마틱스 루틴 등 백그라운드의 스크립트 프로그램에서 함수를 호출하는 경우에는 [UserAlgo.CFunc]을 1로 설정해야 합니다.

전원 셋업/리셋과 동시에 이 프로그램을 실행할 수 있는 상태로 하는 경우에는 이 요소의 값을 1로 설정하는 명령을 pp_startup.txt 프로젝트 구성 파일에 포함해야 합니다.

UserAlgo.RtiCplc

설명: 백그라운드의 C PLC 프로그램의 유효화

범위: 0 .. 1

단위: 부울 형식

전원 셋업 시의 기본값: 0

[UserAlgo.RtiCplc]은 C 언어의 포어그라운드 PLC 프로그램이 유효인지 무효인지 제어합니다. 전원 셋업/리셋 시의 기본값인 0으로 설정하면 프로그램은 무효가 되고 Power PMAC에 로드되어도 실행되지 않습니다. 1로 설정하면 프로그램은 유효가 되고 Power PMAC의 실시간 인터럽트 소프트웨어 사이클마다 실행됩니다.

이 요소의 값은 **enable rticplc** 명령으로 1, **disable rticplc** 명령으로 0을 설정할 수 있습니다.

전원 셋업/리셋과 동시에 이 프로그램을 실행하는 경우에는 이 요소의 값을 1로 설정하는 명령을 pp_startup.txt 프로젝트 구성 파일에 포함해야 합니다.

Coord[x].좌표계 상태 요소(V1.6 2014.03)

Power PMAC 의 좌표계 데이터 구조체는 어플리케이션 개발이나 디버그 또는 실제 어플리케이션 실행에 중요한 많은 상태 요소를 사용자에게 제공합니다. 본 장에서는 중요한 좌표계 상태 요소를 기재 합니다.

Coord[x].ActSegOverride 좌표계의 세그먼트 오버라이드

설명: 현재의 "세그먼트 피드 레이트 오버라이드" 값

범위: 부동 소수점

단위: msec

Coord[x].ActSegOverride 는 최신의 개략적인 보간 계산에서 세그먼트 동작 (**Coord[x].SegMoveTime** > 0 일 때의 **linear**, **circle**, **pvt**)에 사용된 현재의 "세그먼트 오버라이드" 값을 나타냅니다. 이 값은 정규화된 값(%가 아님)으로 1.0 의 값은 "실시간" 실행 명령이 됩니다.

세그먼트 동작에서 **Coord[x].SegMoveTime** msec 마다 Power PMAC 은 다음의 세그먼트 위치를 계산합니다. 이때 **Coord[x].SegMoveTime** 과 오버라이드 값의 곱으로 동작 방정식 수치적 시간을 앞당깁니다. 예를 들어 **Coord[x].SegMoveTime** 이 5.0msec 이고 오버라이드 값이 1.5 인 경우에 5.0msec 마다 세그먼트 위치 계산과 실행이 이루어지지만 동작 방정식의 시간 간격은 $5.0 * 1.5 = 7.5\text{msec}$ 마다 처리되게 됩니다. 따라서 동작은 프로그램된 속도의 150%로 실행됩니다.

Coord[x].SegOverride 가 변경되었을 때 세그먼트로 인해 동작 방정식에 선행하여 시간 계산에 사용되는 실제 내부값 **Coord[x].ActSegOverride** 는 새 명령값에 도달할 때까지 세그먼트마다 저장 설정 요소 **Coord[x].SegOverrideSlew** 의 값으로 변화합니다. 이는 계산된 속도가 스텝 형상으로 변화하는 것을 방지합니다.

Coord[x].AddedDwellDis 좌표계의 드웰 무효 상태

설명: "추가 코너 드웰 무효" Status 비트

범위: 0 ..1

단위: 논리값

코너 각도가 **Coord[x].CornerBlendBp** 로 지정한 각도보다 예각이기 때문에 블렌드가 무효로 설정된 경우이지만 **Coord[x].CornerDwellBp** 로 지정한 각도만큼 예각이 아니고 드웰이 추가되지 않은 경우에 **Coord[x].AddedDwellDis** 는 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 0 에 상당합니다.

Coord[x].AmpEna 좌표계의 앰프 Enable 상태

설명: "앰프 Enable" Status 비트

범위: 0 ..1

단위: 논리값

Coord[x].AmpEna 은 좌표계 내의 모든 모터가 Enable 상태인 경우(클로즈드 루프 또는 오픈 루프 제어에서)에 1 이 됩니다. 또한 이 상태일 때는 새로 앰프 Enable 출력 신호를 유효로 설정하지 않아도 됩니다. 좌표계 중의 모터에 Enable 이 아닌 모터가 있는 경우에 이 비트는 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 12 에 해당합니다.

Coord[x].AmpFault 좌표계의 앰프 폴트 상태

설명: "앰프 폴트 에러" Status 비트

범위: 0 ..1

단위: 논리값

Coord[x].AmpFault 는 이미 앰프 폴트 상태가 아니더라도 앰프 폴트 에러로 또는 "I²T" 적분 전류의 이상(그 경우에 비트 21 도 설정됨)으로 좌표계 내의 어떤 모터가 Disable 로 설정된 경우에 1 이 됩니다. 그 이외에는 0 이며 모터가 다시 Enable 로 설정될 때 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 24 에 해당합니다.

Coord[x].AmpWarn 좌표계의 앰프 경고 상태

설명: "앰프 경고" Status 비트

범위: 0 ..1

단위: 논리값

좌표계의 어떤 모터에서 앰프 폴트를 2 회 연속 읽었을 때 에러로 처리하는 설정인 **Motor[x].AmpFaultLevel** 의 비트 1(값 2)이 1 인 경우에 앰프 폴트 비트를 1 회 읽었을 때에 **Coord[x].AmpWarn** 은 1 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 19 에 해당합니다.

Coord[x].BlockActive 좌표계의 싱글 스텝 블록 유효 상태

설명: "싱글 스텝 블록 유효" Status 비트

범위: 0 ..1

단위: 논리값

Coord[x].BlockActive 는 좌표계가 "싱글 스텝" 모드에서 **bstart**(블록 스타트)와 **bstop**(블록 스톱)의 프로그램 명령 사이를 계산 중이면 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 주로 내부 용도용입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 27 에 상당합니다.

Coord[x].BlockRequest 좌표계의 블록 요구 플래그 상태

설명: "블록 요구 플래그 세트" Status 비트

범위: 0 ..1

단위: 논리값

Coord[x].BlockRequest 는 좌표계의 어떤 모터가 적시에 새로운 동작 섹션에 들어가 있거나 프로그램 시작 명령을 받은 경우로 계산된 모션 대기 행렬에 다음의 동작을 갱신하기 위해 식을 요구하는 경우에 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 주로 내부 용도용입니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 9 에 상당합니다.

Coord[x].BufferWarn 좌표계의 동작 방정식 버퍼 오버 경고 상태

설명: 동작 방정식 버퍼 오버플로 경고 Status

범위: 0 ..2

단위: 열거 형식

Coord[x].BufferWarn 은 좌표계의 동작 방정식 버퍼 오버플로의 가능성 Status 를 나타냅니다. 1 회 실시간 인터럽트(RTI)에서 아주 무거운 계산 부하를 필요로 하는 매우 고속의 블록 레이트(매초 수천 블록)가 필요한 어플리케이션의 설정을 최적화하는 데 활용됩니다.

오버플로 가능성이 없는 경우에 **Coord[x].BufferWarn** 은 0 이 됩니다. 또 하나의 동작으로 버퍼가 오버플로할 정도인 경우에는 1 이 됩니다. 최신 RTI 에서 버퍼가 오버플로하는 동작 방정식을 요구한 경우에는 2 가 됩니다.

Coord[x].BufferWarn 은 펌웨어 V1.6 의 새로운 기능입니다(2014 년 1/4 분기 출시). 이전 버전의 펌웨어에서는 **BufferWarn** 이 2 로 설정되어 있을 때 동작 버퍼는 오버플로하고 프로그램은 에러로 정지했습니다. V1.6 이후에서는 이 상태가 발생하면 Power PMAC 은 1RTI 주기만큼 요구를 지연시킵니다.

저장 설정 요소 **Sys.PreCalc** 는 각 좌표계에서 모션 프로그램을 실행할 때 동작 방정식 버퍼를 충족하도록 서보 사이클에서 예측하는 최소 시간을 설정합니다. 아주 높은 블록 레이트 어플리케이션에서 사용자는 동작 방정식 버퍼에 유효한 보간 처리 준비가 된 동작 섹션이 항상 존재할 수 있도록 기본값의 1 보다 큰 값으로 설정하도록 합니다.

Coord[x].BufferWarn 은 이러한 높은 블록 레이트 방정식상의 **Sys.PreCalc** 설정을 최적화하는 데 이용할 수 있습니다. **Sys.PreCalc** 에는 **Coord[x].BufferWarn** 이 때때로 1 이 되도록 충분히 높이 설정하지만 2 가 될 정도로 높이 설정하지 않습니다.

Sys.PreCalc 가 너무 낮은 경우에 유효한 보간 처리가 시간 내에 다음의 동작 방정식을 얻을 수 없어서 모션 프로그램은 런타임 에러로 정지합니다(**Coord[x].RunTimeError** = 1). **Sys.PreCalc** 가 너무 높은 경우에 동작 방정식 버퍼가 오버플로하고 모션 프로그램은 버퍼 에러로 정지합니다(**Coord[x].ErrorStatus** = 2).

Coord[x].CC3Active 좌표계의 3D 공구 직경 보정 유효 상태

설명: "3D 공구 직경 보정 유효" Status 비트

범위: 0 ..1

단위: 논리값

Coord[x].CC3Active 는 3 차원 공구 직경 보정이 현재 유효한 경우에 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 2 에 상당합니다.

Coord[x].CCAddedArc 좌표계의 공구 직경 보정 추가 원호 상태

설명: "공구 직경 보정의 추가 원호 동작" Status 비트

범위: 0 ..1

단위: 논리값

Coord[x].CCAddedAr 는 현재 계산된 동작이 2D 공구 직경 보정에 따라 코너 바깥쪽에 자동으로 추가된 원호 동작인 경우 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 7 에 상당합니다.

Coord[x].CCMode 좌표계의 공구 직경 보정 모드 상태

설명: "공구 직경 보정 모드" Status 요소

범위: 0 ..3

단위: 열거 형식

Coord[x].CCMode 는 좌표계의 현재 공구 직경 보정 모드를 2 비트로 나타냅니다.

- 0: 공구 직경 보정 무효(**ccmode0**)
- 1: 2D 공구 직경 보정 좌측 유효(**ccmode1**)
- 2: 2D 공구 직경 보정 우측 유효(**ccmode2**)
- 3: 3D 공구 직경 보정 유효(**ccmode3**)

이 **Coord[x].CCMode** 는 32 비트 요소 **Coord[x].Status[1]**의 Bit 24, 25 에 상당합니다.

Coord[x].CCMoveType 좌표계의 공구 직경 보정 동작 타입 상태

설명: "공구 직경 보정 동작 타입" Status 요소

범위: 0 .. 3

단위: 열거 형식

Coord[x].CCMoveType 은 2D 공구 직경 보정이 유효이며 계산된 동작 타입을 2 비트로 나타냅니다. 유효한 값 및 동작 타입은 다음에 지정한 바와 같습니다.

- 0: 드웰
- 1: 거리 제로 또는 평면 외 동작
- 2: 직선 보간 모드 동작
- 3: 원호 보간 모드 동작

이 **Coord[x].CCMoveType** 은 32 비트 요소 **Coord[x].Status[1]**의 Bit 3, 4에 해당합니다.

Coord[x].CCOffReq 좌표계의 공구 직경 보정 취소 요구 상태

설명: "공구 직경 보정 취소 요구" Status 비트

범위: 0 ..1

단위: 논리값

Coord[x].CCOffReq 는 좌표계에서 **CC0** 명령에 따라 공구 직경 보정을 취소하라는 명령을 받고 보정을 원래대로 돌리는 lead-out 동작 계산이 아직 되지 않은 경우에 1 이 됩니다. 그 이외에는 0 입니다.

이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 6에 해당합니다.

Coord[x].cdata 좌표계의 원호 동작 모드

설명: 원호 동작 모드 데이터

범위: 0 .. 255

단위: 비트 필드

Coord[x].cdata 에는 좌표계 상의 원호 동작에 관한 모달 정보를 나타냅니다. 이는 비트마다 특정한 의미를 가진 8 비트값입니다.

- Bit 0: XX/YY/ZZ 방향 모드(0 = 시계 방향, 1 = 시계 반대 방향)
- Bit 1: II 벡터 사양 모드(0 = 인크리멘탈, 1 = 앵솔루트)
- Bit 2: JJ 벡터 사양 모드(0 = 인크리멘탈, 1 = 앵솔루트)
- Bit 3: KK 벡터 사양 모드(0 = 인크리멘탈, 1 = 앵솔루트)
- Bit 4: X/Y/Z 방향 모드(0 = 시계 방향, 1 = 시계 반대 방향)
- Bit 5: I 벡터 사양 모드(0 = 인크리멘탈, 1 = 앵솔루트)
- Bit 6: J 벡터 사양 모드(0 = 인크리멘탈, 1 = 앵솔루트)
- Bit 7: K 벡터 사양 모드(0 = 인크리멘탈, 1 = 앵솔루트)

Bit 0 및 Bit 4 는 **circlen** 프로그램 명령으로 설정됩니다. 다른 비트는 **abs({vector list})** 및 **inc({vector list})** 프로그램 명령으로 설정됩니다.

Coord[x].CdPos[j] 좌표계의 축 동작 명령 위치

설명: 축 동작 명령 위치

범위: 부동 소수점

단위: 축 위치 단위

Coord[x].CdPos[j]는 최신 계산된 동작 명령 목표 위치를 나타냅니다. 좌표계가 현재 연속 이동 시퀀스를 실행 중인 경우에 여기에 저장된 데이터의 동작은 현재 실행 중인 동작보다 1개 이상 앞 선 동작일 수도 있습니다. (목표 위치 버퍼가 설정된 경우에 현재 실행 중인 동작의 목표 위치는 **Coord[x].TPExec.Pos[j]**가 됩니다)

축 인덱스값 j 는 0에서 31까지의 범위 내로 각 인덱스값에 대한 축 명칭은 다음 표와 같습니다.

축	j	축	j	축	j	축	j	축	j	축	j	축	j	축	j
A	0	V	4	Z	8	DD	12	HH	16	OO	20	SS	24	WW	28
B	1	W	5	AA	9	EE	13	LL	17	PP	21	TT	25	XX	29
C	2	X	6	BB	10	FF	14	MM	18	QQ	22	UU	26	YY	30
U	3	Y	7	CC	11	GG	15	NN	19	RR	23	VV	27	ZZ	31

Coord[x].Cflags 좌표계의 조건부 실행 상태

설명: "조건부 실행" Status 비트 배열

범위: \$0.. \$FFFFFFFF

단위: 비트 필드

Coord[x].Cflags 는 좌표계의 32 가지 조건부 실행 플래그 상태를 나타내는 1 비트 단위의 32 비트 Status 입니다. 대응하는 플래그가 설정되면 그 비트는 1 이 되고 대응하는 플래그가 클리어되면 그 비트는 0 이 됩니다. 전원 투입/리셋으로 플래그는 모두 클리어됩니다.

Coord[x].Cflags 의 Bit n 은 좌표계의 조건부 실행 버퍼 프로그램 명령 **cexecn** 및 **cskipn** 의 동작을 결정하기 위해 사용됩니다.

Coord[x].Cflag 의 값은 보통 버퍼 프로그램 명령 **csetn** 및 **cclrn** 을 사용하여 설정됩니다. 그러나 **Coord[x].Cflags** 비트를 직접 세트하거나 클리어할 수 있습니다.

Coord[x].ClosedLoop 좌표계의 클로즈드 루프 모드 상태

설명: "클로즈드 루프 모드" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].ClosedLoop 는 좌표계의 모든 모터가 클로즈드 루프인 경우에 1 이 됩니다. 좌표계의 어떤 모터가 오픈 루프 모드인 경우에는 제로가 됩니다(Enable 또는 Disable 인 경우에도). 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 13 에 해당합니다.

Coord[x].ContMotion 좌표계의 연속 동작 요구 상태

설명: "연속 동작 요구" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].ContMotion 은 좌표계가 도중에 정지하지 않고 블렌드된 일련의 동작 프로그램을 실행하는 경우에 1 이 됩니다. 어떠한 이유로 그와 같은 일련의 동작을 실행하지 않은 경우에는 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 26 에 해당합니다.

Coord[x].Csolve 좌표계의 축 정의 상태

설명: "PMATCH 계산상의 정당한 축 정의" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].Csolve 는 좌표계의 축 정의 구문이 (모션 프로그램 시작, pmatch 명령 실행, 축 위치/속도/편차 조회)"PMATCH"(위치 일치) 함수로 축 위치를 계산하는 조건을 충족하는 경우에 1 이 됩니다. 그 이외에는 0 입니다.

축 정의 구문을 사용하여 좌표계 축이 할당되는 경우에 이 비트는 모션 프로그램을 실행하기 위해 1 이어야 합니다. 단, 좌표계가 모터 위치에서 축 위치를 계산하는 포워드 키네마틱스 서브루틴이 있는 경우에 모션 프로그램의 실행 가능 여부의 결정은 이 **Csolve** Status 비트로는 판정되지 않습니다.

이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 31 에 해당합니다.

Coord[x].DesVelZero 좌표계의 명령 속도 제로 상태

설명: "명령 속도 제로" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].DesVelZero 는 좌표계의 모든 모터가 클로즈드 루프 제어이고 명령 속도가 완전히 제로인 경우(다시 말해 위치를 유지하려고 함) 또는 오픈 루프 모드(Enable 또는 Disable 인 경우에도)이고 실제 속도가 정확히 제로인 경우에 1 이 됩니다. 좌표계의 모터 중 하나가 클로즈드 루프 모드로 명령 속도가 제로가 아닌 경우 또는 오픈 루프 모드(Enable 또는 Disable 인 경우에도)로 실제 속도가 제로가 아닌 경우에 이 비트는 제로가 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 14 에 해당합니다.

Coord[x].EncLoss 좌표계의 인코더가 소실 에러 상태

설명: "인코더 소실 에러" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].EncLoss 는 이미 인코더 소실 신호 상태가 아니더라도 좌표계 중 한 모터가 "인코더 소실" 에러로 Disable 로 설정된 경우에 1 이 됩니다. 그 이외에는 0 입니다. 모터가 다시 Enable 로 설정되면 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 18 에 해당합니다.

Coord[x].ErrorStatus 좌표계의 에러 코드 상태

설명: "에러 코드" Status 요소

범위: 0 .. 255

단위: 열거 형식

Coord[x].ErrorStatus 는 좌표계에서 발생한 에러 타입을 나타내는 8 비트의 Status 요소입니다. 유효한 값 및 동작 타입은 다음에 지정한 바와 같습니다.

값	에러명	설명
0	NoError	정상
1	SyncBufferError	동기 변수 할당 버퍼 에러
2	BufferError	모션 프로그램 버퍼 에러
3	CCMoveTypeError	공구 직경 보정 유효 시의 올바르지 않은 동작 모드 또는 명령(rapid, pvt, spline, lin-to-pvt, new normal, pmatch, pclr, pset, pload)
4	LinToPvtError	자동 linear-to-PVT 모드 에러
5	CCLeadOutMoveError	부정 공구 직경 보정 lead-out 동작(circle 모드 또는 공구 반경 미만의 길이)
6	CCLeadInMoveError	부정 공구 직경 보정 lead-in 동작(circle 모드 또는 공구 반경 미만의 길이)
7	CCBufSizeError	불충분한 공구 직경 보정 이동 버퍼 크기(평면 내 동작에서 다음의 동작에는 불충분)
8	PvtError	올바르지 않게 지정된 PVT 모드 동작
9	CCFeedRateError	공구 보정을 위한 이송 속도로 지정되지 않음
10	CCDirChangeError	프로그램된 동작과 반대 방향으로 보정된 동작, 간섭 상태
11	CCNoSolutionError	보정 이동의 해법을 검출하지 못함
12	CC3NdotTError	3D 공구 직경 보정 벡터 계산 에러(공구 선단 테이블 내의 1 st 등록 미만의 'NdotT' 값)
13	CCDistanceError	리무브 동작을 통해 오버 컷을 해결하지 못함
14	CCNoIntersectError	보정된 궤적의 교점을 검출하지 못함
15	CCNoMovesError	lead-in 과 lead-out 이동 간의 보정 동작 없음
16	RunTimeError	불충분한 동작 계산 시간
17	InPosTimeOutError	인포지션의 제한 시간 초과
18	LHNumMotorsError	사용 모터의 #과 룩 어헤드 버퍼 내의 # 간의 불일치
19-31	(추후 사용하기 위해 예약)	
32	SoftLimitError	소프트웨어 위치 제한을 초과한 정지
33-63	(추후 사용하기 위해 예약)	
64	RadiusError bit 0	X/Y/Z 공간 circle 동작 시의 반경 에러
65-127	(추후 사용하기 위해 예약)	
128	RadiusError bit 1	XX/YY/ZZ 공간 circle 동작 시의 반경 에러
129-255	(추후 사용하기 위해 예약)	

Coord[x].ErrorStatus 가 2 의 거듭제곱과 같은 경우(단일 비트만 세트)에 발생한 에러는 테이블 내의 굵은 글자로 표시되는 해당하는 비트의 에러입니다. 단, 2 의 거듭제곱과 같지 않은 경우(여러 비트가 세트)에 표시된 에러는 8비트의 모든 합계값입니다.

이 **Coord[x].ErrorStatus** 는 32 비트 요소 **Coord[x].Status[0]**의 Bit 0~7 에 상당합니다.

Coord[x].FeedHold 좌표계의 피드 홀드 상태

설명: "피드 홀드 상태" Status 요소

범위: 0 .. 3

단위: 비트 필드

Coord[x].FeedHold 는 좌표계에 피드 홀드 상태를 표시하는 2 비트의 Status 를 나타냅니다.

"피드 홀드"(h, hold) 명령 또는 록 어헤드가 정지하는 "급정지"(\, lh) 명령에 따라 정지하기 위해 감속 중이거나 원래의 상태로 돌아가기 위해 재가속 중이면 Bit 1(값 2)은 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 29 에 상당합니다.

보통 지정되는 타임 베이스가 아니라 "피드 홀드 타임 베이스"를 사용한 경우에 Bit 0(값 1)은 1 이 됩니다. 이는 "피드 홀드"(h, hold) 명령 또는 록 어헤드가 정지하는 "급정지"(\, lh) 명령으로 감속 중인 경우와 완전 정지한 경우입니다. 그 이외에는 0 입니다. 피드 홀드 상태에서 재가속 중인 경우도 포함합니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 28 에 상당합니다.

2 비트의 **Coord[x].FeedHold** 값은 좌표계가 홀드 상태를 향해 감속하고 있으면 3, 홀드 상태에서 정지할 때 1 이 되고 홀드 상태가 해제되어 가속할 때 2 가 되며 보통 동작 중에는 0 이 됩니다.

Coord[x].FeFatal 좌표계의 편차 이상 상태

설명: "편차 이상" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].TriggerMove 는 좌표계 중 한 모터의 편차량이 **Motor[x].FatalFeLimit** 로 설정한 편차 이상 제한을 초과하여 Disable 로 설정된 경우에 1 이 됩니다. 그 이외에는 0 이 됩니다. 모터가 다시 유효가 되면 다시 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 26 에 상당합니다.

Coord[x].FeWarn 좌표계의 편차 경고 상태

설명: "편차 경고" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].FeWarn 은 좌표계 중 한 모터의 편차량이 **Motor[x].WarnFeLimit** 로 설정한 경고 편차 제한을 초과하는 경우에 1 이 됩니다. 편차량이 이 제한 미만인 경우 또는 모터가 그 편차 이상 제한을 초과하여 Disable 로 설정된 경우 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 27 에 상당합니다.

Coord[x].FRAxes 좌표계의 벡터 피드 레이트 축 상태

설명: "벡터 피드 레이트" 축 Status 비트 배열

범위: \$0 .. \$FFFFFFFF

단위: 비트 필드

Coord[x].FRAxes 는 좌표계에서 사용할 수 있는 32 축의 "피드 레이트 축" 상태를 각각 표시하는 32 비트 Status 입니다. linear 또는 circle 모드 동작에서 벡터 피드 레이트 계산에 사용되도록 지정된 축의 해당하는 비트는 1 이 됩니다. 여기서 동작 시간은 모든 축의 이동 벡터 거리를 명령 이송 속도로 나뉘어서 계산됩니다. 벡터 피드 레이트 계산에 사용되지 않는 축에 해당하는 비트는 0 이 됩니다. 전원 투입/리셋 시에는 X, Y 및 Z 축만 피드 레이트 축입니다.

각 축 명칭에 대한 **Coord[x].FRAxes** 의 비트 번호 및 비트값은 다음 표와 같습니다.

축 명칭	비트#	비트값	축 명칭	비트#	비트값
A	0	\$1	HH	16	\$10000
B	1	\$2	LL	17	\$20000
C	2	\$4	MM	18	\$40000
U	3	\$8	NN	19	\$80000
V	4	\$10	OO	20	\$100000
W	5	\$20	PP	21	\$200000
X	6	\$40	QQ	22	\$400000
Y	7	\$80	RR	23	\$800000
Z	8	\$100	SS	24	\$1000000
AA	9	\$200	TT	25	\$2000000
BB	10	\$400	UU	26	\$4000000
CC	11	\$800	VV	27	\$8000000
DD	12	\$1000	WW	28	\$10000000
EE	13	\$2000	XX	29	\$20000000
FF	14	\$4000	YY	30	\$40000000
GG	15	\$8000	ZZ	31	\$80000000

Coord[x].FRAxes 의 값은 보통 버퍼 프로그램 명령 **frax** 를 사용하여 설정됩니다. 그러나 **Coord[x].FRAxes** 의 비트를 직접 세트하거나 클리어할 수 있습니다.

Coord[x].HomeComplete 좌표계의 기준점 확립 상태

설명: "기준점 확립" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].HomeComplete 는 좌표계의 위치 결정 축에 할당된 모든 모터에서 원점 서치 동작 또는 절대 위치 읽기 중 하나에서 기준 위치가 확립된 경우에 1 이 됩니다. 이 비트는 전원 투입/리셋 및 모터의 원점 서치 동작 시작 시에 자동으로 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 15 에 해당합니다.

Coord[x].HomeInProgress 좌표계의 원점 서치 실행 상태

설명: "원점 서치 동작 실행 중" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].HomeInProgress 는 좌표계 중 한 모터가 원점 서치 동작을 시작할 때에 1 이 됩니다. 미리 지정된 트리거 조건이 검출되고 트리거 후의 동작이 시작될 때 또는 트리거가 검출되지 않고 동작 종료한 경우에는 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 30 에 해당합니다.

Coord[x].I2tFault 좌표계의 전류 적분 이상 상태

설명: "전류 적분(I^2T) 이상" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].I2tFault 는 좌표계 중 한 모터가 **Motor[x].I2tTrip** 으로 설정한 전류 적분 제한을 초과하여 Disable 이 된 경우에 1 이 됩니다. 이때 앰프 폴트 비트 **Coord[x].AmpFault** 도 세트됩니다. 그 이외에는 0 이 되며 모터가 다시 Enable 로 설정될 때 0 이 됩니다(앰프 자체의 전류 적분 기능인 앰프 이상이 발생한 경우에는 이 비트는 세트 설정되지 않습니다). 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 21 에 해당합니다.

Coord[x].IncAxes 좌표계의 증가 모드 상태

설명: "인크리멘탈 모드" 축 Status 비트 배열

범위: \$0 .. \$FFFFFFFF

단위: 비트 필드

Coord[x].IncAxes 는 좌표계에서 사용할 수 있는 32 축의 증가/앰솔루트 모드 상태를 각각 나타내는 32 비트의 Status 입니다. 대응하는 축이 인크리멘탈 모드인 경우 비트는 1 이 되며 축의 동작 명령값은 현재 명령 위치에서의 거리를 지정합니다. 대응하는 축이 앰솔루트 모드인 경우 비트는 0 이 되며 축의 동작 명령값은 프로그램 원점 기준의 목표 위치를 지정합니다.

각 축 명칭에 대한 **Coord[x].IncAxes** 의 비트 번호 및 비트값은 다음 표와 같습니다.

축 명칭	비트#	비트값	축 명칭	비트#	비트값
A	0	\$1	HH	16	\$10000
B	1	\$2	LL	17	\$20000
C	2	\$4	MM	18	\$40000
U	3	\$8	NN	19	\$80000
V	4	\$10	OO	20	\$100000
W	5	\$20	PP	21	\$200000
X	6	\$40	QQ	22	\$400000
Y	7	\$80	RR	23	\$800000
Z	8	\$100	SS	24	\$1000000
AA	9	\$200	TT	25	\$2000000
BB	10	\$400	UU	26	\$4000000
CC	11	\$800	VV	27	\$8000000
DD	12	\$1000	WW	28	\$10000000
EE	13	\$2000	XX	29	\$20000000
FF	14	\$4000	YY	30	\$40000000
GG	15	\$8000	ZZ	31	\$80000000

Coord[x].IncAxes 의 값은 보통 버퍼 프로그램 명령 **inc** 및 **abs** 를 사용하여 설정됩니다. 그러나 **Coord[x].IncAxes** 의 비트를 직접 세트하거나 클리어할 수 있습니다.

Coord[x].InPos 좌표계의 인포지션 상태

설명: "인포지션" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].InPos 는 좌표계의 모든 모터에서 다음 "인포지션" 조건을 모두 만족했을 때 1 이 됩니다. (모터는 클로즈드 루프, 명령 속도 제로, 동작 타이머가 실행 중이 아님(동작, 드웰 또는 딜레이가 실행 중이지 않아야 함), 편차량이 **Motor[x].InPosBand** 이하 그리고 이 모든 조건이 (**Motor[x].InPosTime** - 1)서보 사이클 동안 연속하여 참일 것) 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 11 에 해당합니다.

Coord[x].LHMotorSlots 좌표계의 록 어헤드 버퍼 모터 열수

설명: 록 어헤드 버퍼에 필요한 모터 열의 수

범위: 0 .. 256

단위: 모터

Coord[x].LHMotorSlots 는 록 어헤드 계산이 모터를 조작할 수 있도록 록 어헤드 버퍼 내의 "열"로 필요한 모터 수입니다. 이는 이 좌표계의 축에 할당된 모터 수와 같아집니다. 모터가 좌표계의 축 정의 구문으로 1 축 이상 직접 할당된 경우(예를 들어 **#4- >1000C**) 또는 좌표계에서 인버스 키네마틱스 축으로 정의된 경우(예를 들어 **#5->I**) 또는 좌표계에서 스핀들로 정의된 경우(예를 들어 **#4->S**)에 해당 합계의 수가 됩니다. 모터가 좌표계(예를 들어 **#6- >0**)에서 "무효"로 정의된 경우에는 수로 카운트하지 않습니다.

스핀들 축은 룩 어헤드 계산을 실시하지 않아도 되지만 모터가 좌표계에서 위치 결정 축으로 재정의될 때 테이블을 삭제하고 재작성하지 않고 완료되도록 테이블이 확보됩니다.

Coord[x].LHSize 좌표계의 룩 어헤드 버퍼 세그먼트 수

설명: 룩 어헤드 버퍼 내의 1 모터당 세그먼트 수

범위: 음이 아닌 정수(0 또는 양의 정수)

단위: 동작 세그먼트

Coord[x].LHSize 는 좌표계의 룩 어헤드 버퍼 내의 각 모터를 위해 확보되는 동작 세그먼트 수입니다. 이 값은 좌표계의 **define lookahead{상수}** 명령값으로 설정됩니다. 룩 어헤드 버퍼가 좌표계에서 정의되지 않은 경우에는 0 이 됩니다. 이 값은 버퍼에 현재 실행 위치의 전방으로 어느 정도의 세그먼트 수가 저장되었는지 나타내는 저장 설정 요소 **Coord[x].LHDistance** 의 값과는 다릅니다.

이 요소의 값은 **save** 명령으로 플래시 메모리에 저장됩니다. 그러나 스크립트 환경에서는 직접 쓸 수 없습니다.

Coord[x].LHStatus 좌표계의 특별 룩 어헤드 버퍼 상태

설명: 특별한 룩 어헤드 버퍼 Status 바이트

범위: 0 .. 255

단위: 비트 필드

Coord[x].LHStatus 는 특별한 룩 어헤드 버퍼 동작에 관한 8 가지의 Status 비트를 결합한 것입니다. 각 비트는 각각 개별 이름으로 별도로 액세스할 수도 있습니다. 각 비트의 기능은 개별적인 비트의 설명에 기재했습니다. **Coord[x].LHStatus** 를 구성하는 8 가지의 개별 Status 비트는 다음과 같습니다.

- Bit 0(값 1): **Coord[x].LookAheadActive**
- Bit 1(값 2): **Coord[x].LookAheadFlush**
- Bit 2(값 4): **Coord[x].LookAheadReCalc**
- Bit 3(값 8): **Coord[x].LookAheadChange**
- Bit 4(값 16): **Coord[x].LookAheadStop**
- Bit 5(값 32): **Coord[x].LookAheadDir**
- Bit 6(값 64): **Coord[x].LookAheadLookBack**
- Bit 7(값 128): **Coord[x].LookAheadWrap**

Coord[x].LHStatus 는 32 비트 요소 **Coord[x].Status[1]**의 Bit 8~15 에 해당합니다.

Coord[x].LimitStop 좌표계의 하드웨어 리미트 정지 상태

설명: "하드웨어 위치 리미트 정지" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].LimitStop 은 좌표계 중 한 모터가 양 또는 음 중 하나의 하드웨어의 오버트래블 리미트에 도달하여 정지했거나 정지 중이면 그 리미트 위치에 현재는 있지 않더라도 1 이 됩니다. 리미트에서 빠져 나온 경우를 제외하고 리미트 위치에 있는 경우를 포함한 다른 모든 경우에는 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 25 에 해당합니다.

Coord[x].LinToPvtBuf 좌표계의 Linear-PVT 변환 상태

설명: "Linear-PVT 변환" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].LinToPvtBuf 는 (**Coord[x].SegLinToPvt**>0 에서)PVT 동작으로 변환된 linear 모드 동작이 현재 버퍼에 들어가 있고 실행 대기 상태인 경우에 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 주로 내부 용도용입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 30 에 해당합니다.

Coord[x].LinToPvtError 좌표계의 linear-PVT 변환 에러 상태

설명: "linear-PVT 변환 모드 에러" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].LinToPvtError 는 좌표계의 모션 프로그램이 linear-PVT 변환 모드 동작 실행 중에 에러로 Abort 된 경우에 1 이 됩니다(**Coord[x].ErrorStatus** 의 다른 7 비트가 세트되지 않으면 **Coord[x].ErrorStatus** 는 4 가 됩니다). 다른 비트가 세트된 경우에는 단순히 에러 코드의 Bit 2(값 4)가 1 이 됩니다. **Coord[x].ErrorStatus** 항목의 에러 코드 목록을 참조하십시오. 그 이외에는 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 2 및 8 비트 요소 **Coord[x].ErrorStatus** 의 Bit 2 에 해당합니다.

Coord[x].LookAheadActive 좌표계의 룩 어헤드 버퍼 실행 상태

설명: "룩 어헤드 버퍼 실행" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].LookAheadActive 는 좌표계가 룩 어헤드 버퍼 내에서 실행된 경우에 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 8 및 8 비트 요소 **Coord[x].LHStatus** 의 Bit 0 에 상당합니다.

Coord[x].LookAheadChange 좌표계의 룩 어헤드 실행 상태

설명: "룩 어헤드 실행 변화" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].LookAheadChange 는 룩 어헤드 버퍼의 실행 모드가 전방, 역행, 정지 모드 중 어느 2 가지 사이에서 현재 변화하는 중이면 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 11 및 8 비트 요소 **Coord[x].LHStatus** 의 Bit 3 에 상당합니다.

Coord[x].LookAheadDir 좌표계의 룩 어헤드 방향 상태

설명: "룩 어헤드 방향" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].LookAheadDir 은 룩 어헤드 버퍼가 역행 방향으로 실행하는 중이면 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 13 및 8 비트 요소 **Coord[x].LHStatus** 의 Bit 5 에 상당합니다.

Coord[x].LookAheadFlush 좌표계의 룩 어헤드 버퍼 순환 상태

설명: "룩 어헤드 버퍼 순환" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].LookAheadFlush 는 룩 어헤드 버퍼가 일련의 동작의 종점에 도달하고 다음의 새로운 세그먼트가 버퍼에 추가되지 않는 경우에 1 이 됩니다(그러나 계속해서 기존의 세그먼트는 실행 중인 경우). 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 9 및 8 비트 요소 **Coord[x].LHStatus** 의 Bit 1 에 상당합니다.

Coord[x].LookAheadLookBack 좌표계의 룩 어헤드 버퍼 후방 상태

설명: "룩 어헤드 버퍼 룩 백" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].LookAheadLookBack 은 마지막에 추가된 동작 끝에 제어된 정지가 가능하도록 룩 어헤드 알고리즘이 버퍼에서 "후방" 계산한 경우에 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 14 및 8 비트 요소 **Coord[x].LHStatus** 의 Bit 6 에 상당합니다.

Coord[x].LookAheadReCalc 좌표계의 룩 어헤드 버퍼 재계산 상태

설명: "룩 어헤드 버퍼 재계산" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].LookAheadReCalc 는 새로운 세그먼트를 버퍼에 추가하지 않고 역행 또는 역행에서 "복귀"할 때에 기존의 버퍼 내에서 룩 어헤드의 계산을 하는 경우에는 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 10 및 8 비트 요소 **Coord[x].LHStatus** 의 Bit 2 에 상당합니다.

Coord[x].LookAheadStop 좌표계의 룩 어헤드 정지 상태

설명: "룩 어헤드 정지" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].LookAheadStop 은 룩 어헤드 버퍼의 실행이 "급정지"(\, lh) 명령으로 정지 또는 감속하는 경우에 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 12 및 8 비트 요소 **Coord[x].LHStatus** 의 Bit 4 에 상당합니다.

Coord[x].LookAheadWrap 좌표계의 룩 어헤드 버퍼 순환 상태

설명: "룩 어헤드 버퍼 순환" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].LookAheadWrap 은 룩 어헤드 버퍼가 가득 차서 현재의 연속 동작 시퀀스가 "순환"하는 중이면 1 이 됩니다. 이는 완전한 시퀀스의 역행이 불가능하다는 것을 의미합니다. 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 15 및 8 비트 요소 **Coord[x].LHStatus** 의 Bit 7 에 상당합니다.

Coord[x].MinusLimit 좌표계의 음의 하드웨어 리미트 상태

설명: "음의 하드웨어 리미트 설정" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].MinusLimit 는 좌표계 중 한 모터가 음의 하드웨어 리미트 상에 있는 경우에 1 이 됩니다. 모터가 사전에 이 리미트에 도달하여 정지했더라도 상기 이외에는 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 29 에 해당합니다.

Coord[x].Motors[i] 좌표계의 축 할당 모터 번호

설명: 좌표계 상의 축 할당 모터 번호

범위: 0 .. 255

단위: 모터 번호

Coord[x].Motors[i]는 이 좌표계의 축에 할당된 "제 i" 모터 번호를 나타냅니다. **Coord[x].Motors[0]**은 이 좌표계 상의 축에 할당된 최하위 번호의 모터 번호를 나타냅니다. **Coord[x].Motors[1]**은 다음의 하위 번호의 모터 번호를 나타냅니다. 이 값은 $i = \text{Coord[x].NumMotors} - 1$ 까지 할당됩니다. 이러한 요소는 이 좌표계에서 "무효" 정의 또는 "스핀들" 정의된 모터는 통지하지 않습니다.

Coord[x].MoveMode 좌표계의 프로그램 동작 모드 상태

설명: "프로그램 동작 모드" 상태 요소

범위: 0 .. 3

단위: 열거 형식

Coord[x].MoveMode 는 좌표계에서 최근에 계산된 동작 타입을 나타내는 2 비트 상태 요소입니다. 유효한 값 및 동작 타입은 다음에 지정한 바와 같습니다.

- 0: 블렌드 동작(linear 또는 원호 모드 동작)
- 1: 스플라인 모드 동작
- 2: PVT 모드 동작
- 3: Rapid 모드 동작

Coord[x].MoveMode 가 0 인 경우에 각각의 동작 모드를 식별하기 위해서는 다른 상태 요소를 사용해야 합니다. 블렌드된 동작 모드가 **linear** 인 경우에 **Coord[x].Omega0[0]**은 0.0 이 됩니다. 원호 모드인 경우에 **Coord[x].Omega0[0]**은 제로가 아닌 값이 됩니다. 원호 모드인 경우에 X/Y/Z 직행 좌표 공간에서 **Coord[x].cdata** 의 Bit 4(값 16)는 **circle1**(시계 방향) 모드에서는 0 이고 **circle2**(시계 반대 방향) 모드에서는 1 이 됩니다. Bit 0(값 1)은 동일하게 XX/YY/ZZ 직행 좌표 공간에서 모드를 통지합니다.

Coord[x].MoveMode 는 32 비트 요소 **Coord[x].Status[1]**의 Bit 22 및 23 에 해당합니다.

Coord[x].Ncalc 좌표계의 동기 행 레이블 번호

설명: 마지막에 계산된 행의 동기 행 레이블 번호

범위: 0 .. $2^{32} - 1$

단위: 열거 형식

Coord[x].Ncalc 는 마지막에 계산된 모션 프로그램 행에 동기한 행 레이블의 값을 나타내며(예를 들어 마지막 행이 N560 이었다면 560) 모션 프로그램의 실행을 모니터링하는 간단한 방법을 제공합니다. 이 행에 동작 명령이 있는 경우에 그것은 반드시 이 동작이 실행을 시작했다는 것을 의미하는 것은 아닙니다. 관련된 상태 요소 **Coord[x].Nsync** 는 그 목적으로 사용할 수 있습니다.

Coord[x].Normal[j] 좌표계의 직교 벡터 성분값

설명: 조작 평면 직교 벡터 성분값

범위: 부동 소수점

단위: 3D 단위 벡터 성분

Coord[x].Normal[j]는 좌표계에서 원호 보간 및 2D 공구 직경 보정 평면의 현재의 법선 벡터로 지정한 차원 성분의 값을 나타냅니다. 각각의 좌표계에는 2 가지의 법선 벡터를 가지고 있으며 하나는 X/Y/Z 공간이고 다른 하나는 XX/YY/ZZ 공간으로 이에 따라 좌표계에는 6 가지의 직교하는 벡터 성분이 있습니다.

6 가지의 성분은 다음과 같습니다.

- **Coord[x].Normal[0]**: X/Y/Z 공간의 I-성분
- **Coord[x].Normal[1]**: X/Y/Z 공간의 J-성분
- **Coord[x].Normal[2]**: X/Y/Z 공간의 K-성분
- **Coord[x].Normal[3]**: XX/YY/ZZ 공간의 II-성분
- **Coord[x].Normal[4]**: XX/YY/ZZ 공간의 JJ-성분
- **Coord[x].Normal[5]**: XX/YY/ZZ 공간의 KK-성분

해당 요소의 값은 **normal** 프로그램 명령으로 설정됩니다. 명령으로 지정된 벡터가 단위 벡터가 아니어도 저장된 성분값은 같은 방향의 단위 벡터가 됩니다.

Coord[x].Nsync 좌표계의 실행 행 레이블 번호

설명: 마지막에 계산된 행의 동기 행 레이블 번호

범위: 0 .. $2^{32}-1$

단위: 열거 형식

Coord[x].Nsync에는 모션 프로그램에서 현재 실행 중이거나 마지막에 실행된 동작에 가장 가까운 선행하는 행 레이블의 값을 나타내며(예를 들어 마지막의 레이블이 N740 이었다면 740) 모션 프로그램에 의한 동작 실행을 모니터링하는 간단한 방법을 제공합니다. 프로그램 계산은 이 행을 초과하여 진행 중일 수도 있습니다. 관련된 상태 요소 **Coord[x].Ncalc**는 계산점을 모니터링하기 위해 사용할 수 있습니다.

Coord[x].NumMotors 좌표계의 위치 결정 축 모터 수

설명: 좌표계에서 위치 결정 축에 할당된 모터의 수

범위: 0 .. 256

단위: 모터

Coord[x].NumMotors는 이 좌표계에서 위치 결정 축에 할당된 모터 수에 나타냅니다. 모터가 좌표계의 축 정의 구문으로 1 축 이상이 직접 할당된 경우(예를 들어 **#4- >1000C**) 또는 좌표계에서 인버스 키네마틱스 축으로 정의된 경우(예를 들어 **#5->I**)에 해당 합계의 수가 됩니다. 좌표계에서 "무효"로 정의된 경우(예를 들어 **#6- >0**) 또는 좌표계에서 스핀들로 정의된 경우(예를 들어 **#4->S**)에는 수로 카운트하지 않습니다.

Coord[x].NXYZ[i] 좌표계의 3D 보정 법선 벡터 성분값

설명: 3D 공구 직경 보정 면 법선 벡터 성분값

범위: 부동 소수점

단위: 3D 단위 벡터 성분

Coord[x].NXYZ[i]는 X/Y/Z 공간상의 3D 공구(커터) 직경 보정의 보정 평면에 대한 현재의 법선 벡터를 지정하는 차원 성분의 값을 나타냅니다. 이 벡터에는 3 가지의 성분이 있습니다.

- **Coord[x].NXYZ[0]**: X/Y/Z 공간의 I-성분
- **Coord[x].NXYZ[1]**: X/Y/Z 공간의 J-성분
- **Coord[x].NXYZ[2]**: X/Y/Z 공간의 K-성분

해당 요소의 값은 **nxyz** 프로그램 명령으로 설정됩니다. 명령으로 지정된 벡터가 단위 벡터가 아니어도 저장된 성분값이 같은 방향의 단위 벡터가 됩니다.

Coord[x].PlusLimit 좌표계의 양의 하드웨어 리미트 상태

설명: "양의 하드웨어 리미트 설정" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].PlusLimit 는 좌표계 중 한 모터가 양의 하드웨어 리미트 상에 있는 경우에 1 이 됩니다. 모터가 사전에 이 리미트에 도달하여 정지했더라도 상기 이외에는 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 28 에 해당합니다.

Coord[x].ProgActive 좌표계의 모션 프로그램 기동 상태

설명: 모션 프로그램 기동 상태

범위: 0 .. 1

단위: 논리값

Coord[x].ProgActive 는 좌표계 상의 현재 모션 프로그램 "기동" 상태를 나타냅니다. 모션 프로그램이 현재 기동 중이면 1 입니다. 기동 중인 모션 프로그램은 실행하거나 실행을 재개할 수 있는 방법(**q[pause]** 또는 **s[step]** 명령)으로 일시 정지할 수 있습니다. 실행 부분은 모션 프로그램의 서브 프로그램일 수도 있습니다. 모션 프로그램이 좌표계에서 기동 중이지 않은 경우에 **Coord[x].ProgActive** 는 0 이 됩니다. 약간 다른 정보에 대해서는 관련된 Status 비트 **Coord[x].ProgRunning** 및 **Coord[x].ProgProceeding** 을 참조하십시오.

모션 프로그램이 기동 중이면 그 버퍼는 클리어하거나 새 프로그램을 다운로드하기 위해 열 수 없습니다. **a[abort]** 명령은 자동으로 프로그램 실행을 중지하는 에러 상태와 같이 모션 프로그램 실행을 정지합니다.

이 Status 는 정기적으로 산출된 Status 가 아닙니다. 함수를 통해 요구가 있을 때 몇 개의 비트에서 산출됩니다. 따라서 C 환경에서 직접 액세스할 수 없습니다. 대신에 API 함수 호출을 통해 액세스해야 합니다.

Coord[x].ProgProceeding 좌표계의 모션 프로그램 진척 상태

설명: 모션 프로그램 진척 상태

범위: 0 .. 1

단위: 논리값

Coord[x].ProgProceeding 은 좌표계의 모션 프로그램 진척 실행 상태를 나타냅니다. 모션 프로그램 또는 그로부터 호출된 서브 프로그램이 실행 중으로 프로그램 계산이 진행 중이면 1 이 됩니다. 모션 프로그램 실행이 재개 가능한 상태에서 (**q[pause]**, **s[step]**, **\[lh]** 또는 **h[hold]** 명령으로)일시 정지된 경우에 프로그램은 "실행 가능" (**Coord[x].ProgActive** = 1) 상태이지만 "진행 중"인 상태는 아닙니다.

모션이 타임 베이스값 %0 에서 정지한 경우("hold" 이외의 방법으로) 프로그램은 아직 내부에서는 실행 중으로 해당 Status 는 1 이 됩니다. **Coord[x].ProgProceeding** 과 **Coord[x].ProgRunning** 과의 차이는 **Coord[x].ProgRunning** 은 "hold" 중에는 1 인 상태이지만 **Coord[x].ProgProceeding** 은 프로그램이 "hold"에서 일시 정지하면 0 이 됩니다.

좌표계의 모션 프로그램이 "기동 중"(**Coord[x].ProgActive** = 1)이더라도 계산처리가 현재 진행 중이 아니면 **Coord[x].ProgProceeding** 은 0 입니다.

이 Status 는 정기적으로 산출된 Status 가 아닙니다. 함수를 통해 요구가 있을 때 몇 개의 비트에서 산출됩니다. 따라서 C 환경에서 직접 액세스할 수 없습니다. 대신에 API 함수 호출을 통해 액세스해야 합니다.

Coord[x].ProgRunning 좌표계의 모션 프로그램 실행 상태

설명: 모션 프로그램 실행 상태

범위: 0 .. 1

단위: 논리값

Coord[x].ProgRunning 은 좌표계의 모션 프로그램 실행 상태를 나타냅니다. 모션 프로그램 또는 그로부터 호출된 서브 프로그램을 현재 실행 중이면 1 이 됩니다. 모션 프로그램 실행을 재개할 수 있는 상태에서 (**q[pause]**, **s[step]** 또는 **\\[h]** 명령으로) 일시 정지된 경우에 프로그램은 "실행 가능"(**Coord[x].ProgActive** = 1) 상태이지만 "실행 중" 상태는 아닙니다.

모션이 **h[hold]** 또는 %0 명령으로 정지한 경우에 프로그램은 아직 내부에서는 실행 중이며 이 Status 는 1 이 됩니다. **Coord[x].ProgRunning** 과 **Coord[x].ProgProceeding** 과의 차이는 **Coord[x].ProgRunning** 은 "hold" 중에는 1 인 상태이지만 **Coord[x].ProgProceeding** 은 프로그램이 "hold"에서 일시 정지될 때는 0 이 됩니다.

좌표계의 모션 프로그램이 "실행 가능"(**Coord[x].ProgActive** = 1)이더라도 프로그램이 현재 "실행 중"이 아니면 **Coord[x].ProgRunning** 은 0 이 됩니다.

이 Status 는 정기적으로 산출된 Status 가 아닙니다. 함수를 통해 요구가 있을 때 몇 개의 비트에서 산출됩니다. 따라서 C 환경에서 직접 액세스할 수 없습니다. 대신에 API 함수 호출을 통해 액세스해야 합니다.

Coord[x].PvtError 좌표계의 PVT 모드 에러 상태

설명: "PVT 모드 에러" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].PvtError 는 좌표계의 모션 프로그램이 PVT 모드 동작 실행 중의 에러로 인해 Abort 된 경우에 1 이 됩니다(**Coord[x].ErrorStatus** 의 다른 7 비트 모두 세트되어 있지 않으면 **Coord[x].ErrorStatus** 는 8 이 됩니다). 다른 비트가 세트된 경우에는 단순히 에러 코드의 Bit 3(값 8)이 1 이 됩니다. **Coord[x].ErrorStatus** 항목의 에러 코드 목록을 참조하십시오. 그 이외에는 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 3 및 8 비트 요소 **Coord[x].ErrorStatus** 의 Bit 3 에 해당합니다.

Coord[x].RadiusError 좌표계의 Circle 반경 에러 상태

설명: "Circle 반경 에러" 상태 요소

범위: 0 .. 3

단위: 비트 필드

Coord[x].RadiusError 는 좌표계 상의 circle 동작 반경 에러 유무를 나타내는 2 비트의 상태 요소입니다. 이 Status 는 8 비트 상태 요소 **Coord[x].ErrorStatus** 의 2 비트에 해당합니다.

Bit 0(값 1)은 X/Y/Z 축 공간의 circle 동작에서 **Coord[x].RadiusErrorLimit** 를 초과하는 반경 에러가 있는 경우 1 이 됩니다(**Coord[x].ErrorStatus** 의 다른 7 비트모두 세트되어 있지 않으면 **Coord[x].ErrorStatus** 는 64 가 됩니다). 다른 비트가 세트된 경우에는 단순히 에러 코드의 Bit 6(값 64)이 1 이 됩니다. **Coord[x].ErrorStatus** 항목의 에러 코드 목록을 참조하십시오. Bit 6 이 세트된 에러 코드가 아닌 경우에 이 비트는 0 입니다. 이 비트는 그것은 32 비트 요소 **Coord[x].Status[0]**의 Bit 6, 8 비트 요소 **Coord[x].ErrorStatus** 의 Bit 6 에 해당합니다.

Bit 1(값 2)은 XX/YY/ZZ 축 공간의 circle 동작에서 **Coord[x].RadiusErrorLimit** 를 초과하는 반경 에러가 있는 경우 1 이 됩니다(그 경우 **Coord[x].ErrorStatus** 의 다른 7 비트 모두 세트되어 있지 않으면 **Coord[x].ErrorStatus** 는 128 이 됩니다). 다른 비트가 세트되어 있는 경우에는 단순히 에러 코드의 Bit 7(값 128)이 1 이 됩니다. **Coord[x].ErrorStatus** 항목의 에러 코드 목록을 참조하십시오. Bit 7 이 세트된 에러 코드가 아닌 경우에 이 비트는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 7 및 8 비트 요소 **Coord[x].ErrorStatus** 의 Bit 7 에 해당합니다.

Coord[x].RunTimeError 좌표계의 런타임 에러 상태

설명: "런타임 에러" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].RunTimeError 는 좌표계의 모션 프로그램 계산 시간 부족으로 인해 Abort 된 경우에 1 이 됩니다(**Coord[x].ErrorStatus** 의 다른 7 비트 모두 세트되어 있지 않으면 **Coord[x].ErrorStatus** 는 16 이 됩니다). 다른 비트가 세트된 경우에는 단순히 에러 코드의 Bit 4(값 16)가 1 이 됩니다. **Coord[x].ErrorStatus** 항목의 에러 코드 목록을 참조하십시오. 그 이외에는 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 4 및 8 비트 요소 **Coord[x].ErrorStatus** 의 Bit 4 에 해당합니다.

Coord[x].SegEnabled 좌표계의 세그먼트 실행 상태

설명: "세그먼트 동작 실행" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].SegEnabled 는 좌표계가 세그먼트 동작(**Coord[x].SegMoveTime**>0 에서 linear, circle 또는 pvt 동작)을 현재 실행한(또는 마지막에 실행한) 경우에 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 17 에 해당합니다.

Coord[x].SegMove 좌표계의 세그먼트 처리 상태

설명: "세그먼트 처리 중의 동작" 상태 요소

범위: 0 .. 3

단위: 비트 필드

Coord[x].SegMove 는 좌표계에서의 세그먼트 동작 실행 상태를 나타내는 2 비트의 상태 요소입니다.

Bit 1(값 2)은 좌표계의 세그먼트 모드에서 PVT 동작을 현재 실행 중이면 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 주로 내부 용도용입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 21 에 해당합니다.

Bit 0(값 1)은 좌표계의 세그먼트 모드에서 linear 또는 circle 동작을 현재 실행 중이면 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 주로 내부 용도용입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 20 에 해당합니다.

Coord[x].SegMoveAccel 좌표계의 제 1 동작 세그먼트 실행 상태

설명: "제 1 동작 세그먼트 실행" Status 비트

범위: 0 ..1

단위: 논리값

Coord[x].SegMoveAcce 은 좌표계가 세그먼트 동작이나 일련의 동작 정지 상태에서 최초(추가)의 세그먼트를 현재 실행하는 중이면 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 주로 내부 용도용입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 19 에 해당합니다.

Coord[x].SegMoveDecel 좌표계의 종료 세그먼트 실행 상태

설명: "마지막 동작 세그먼트 실행 중" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].SegMoveDecel 은 좌표계가 세그먼트 동작이나 일련의 동작을 정지하기 위한 마지막(추가) 세그먼트를 현재 실행하는 중이면 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 주로 내부 용도용입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 18에 상당합니다.

Coord[x].SegStopReq 좌표계의 세그먼트 동작 정지 요구 상태

설명: "세그먼트 동작 정지 요구" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].SegStopReq 는 좌표계가 세그먼트화된 일련의 동작 중에 정지하도록 요구된 경우에 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 주로 내부 용도용입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 16에 상당합니다.

Coord[x].SharpCornerStop 좌표계의 예각 코너 블렌드 무효 상태

설명: "예각 코너에 따른 블렌드 무효" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].SharpCornerStop 은 코너 각도가 **Coord[x].CornerBlendBp** 로 설정한 각도보다 예리하고 블렌드가 무효로 설정된 경우에 1 이 됩니다. 그 이외에는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[1]**의 Bit 1에 상당합니다.

Coord[x].SoftLimit 좌표계의 소프트웨어 리미트 정지 상태

설명: "소프트웨어 리미트 정지" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].SoftLimit 는 좌표계의 모터가 양 또는 음 중 하나의 소프트웨어 오버트래블 리미트에 도달하여 정지했기 때문에 모션 프로그램이 정지한 경우에 1 이 됩니다(**Coord[x].ErrorStatus** 의 다른 7 비트 모두 세트되어 있지 않으면 **Coord[x].ErrorStatus** 는 32 가 됩니다). 다른 비트가 세트된 경우에는 단순히 에러 코드의 Bit 5(값 32)가 1 이 됩니다. **Coord[x].ErrorStatus** 항목의 에러 코드 목록을

참조하십시오. 리미트에서 빠져 나온 경우를 제외하고 리미트 위치에 있는 경우를 포함한 다른 모든 경우에 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 5 및 8 비트 요소 **Coord[x].ErrorStatus** 의 Bit 5 에 해당합니다.

모터 명령 위치가 소프트웨어 오버트래블 리미트로 인해 단순히 정지된 경우(저장 설정 요소 **Coord[x].SoftLimitStopDis** 가 1 인 설정에서) 또는 좌표계의 모터가 JOG 와 같은 독립된 동작으로 소프트웨어 오버트래블 리미트에 도달하여 정지한 경우에 **Coord[x].SoftLimit** 는 세트되지 않습니다.

Coord[x].SoftMinusLimit 좌표계의 음의 소프트웨어 리미트 상태

설명: "음의 소프트웨어 리미트 세트" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].SoftMinusLimit 는 좌표계 중 한 모터가 (**Motor[x].MinPos** - **Motor[x].SoftLimitOffset**)에서 설정되는 음의 소프트웨어 리미트에 도달했거나 초과한 경우에 1 이 됩니다(이 제한을 유효로 설정하려면 **Motor[x].MinPos** 는 **Motor[x].MaxPos** 미만이어야 합니다). 모터가 사전에 이 리미트에 도달하여 정지했더라도 그 이외에는 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 23 에 해당합니다.

Coord[x].SoftPlusLimit 좌표계의 양의 소프트웨어 리미트 상태

설명: "양의 소프트웨어 리미트 세트" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].SoftPlusLimit 는 좌표계 중 한 모터가 (**Motor[x].MaxPos** + **Motor[x].SoftLimitOffset**)에서 설정되는 양의 소프트웨어 리미트에 도달했거나 초과한 경우에 1 이 됩니다(이 제한을 유효로 설정하려면 **Motor[x].MaxPos** 는 **Motor[x].MinPos** 보다 커야 합니다). 모터가 사전에 이 리미트에 도달하여 정지했더라도 그 이외에는 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 22 에 해당합니다.

Coord[x].Status[0] 좌표계의 제 1 Status

설명: 좌표계 제 1 Status 워드

범위: \$0.. \$FFFFFFFF

단위: 비트 필드

Coord[x].Status[0]은 좌표계의 많은 글로벌한 Status 비트를 저장한 최초의 32 비트 Status 워드입니다. 워드 속의 Status 비트 목록은 다음 표와 같습니다.

Bit #	16 진수값	요소명: Coord[x].{요소}	설명
31	\$80000000	TriggerMove	트리거 서치 중(좌표계 중 한 모터)
30	\$40000000	HomeInProgress	원점 서치 중(좌표계 중 한 모터)
29	\$20000000	MinusLimit	음의 하드웨어 리미트 세트(좌표계 중 한 모터)
28	\$10000000	PlusLimit	양의 하드웨어 리미트 세트(좌표계 중 한 모터)
27	\$8000000	FeWarn	편차 경고(좌표계 중 한 모터)
26	\$4000000	FeFatal	편차 이상(좌표계 중 한 모터)
25	\$2000000	LimitStop	하드웨어 리미트 정지(좌표계 중 한 모터)
24	\$1000000	AmpFault	앰프 이상(좌표계 중 한 모터)
23	\$800000	SoftMinusLimit	음의 소프트웨어 리미트 세트(좌표계 중 한 모터)
22	\$400000	SoftPlusLimit	양의 소프트웨어 리미트 세트(좌표계 중 한 모터)
21	\$200000	I2tFault	적분 전류(I^2T) 이상(좌표계 중 한 모터)
20	\$100000	TriggerNotFound	트리거 미검출(좌표계 중 한 모터)
19	\$80000	AmpWarn	앰프 경고(좌표계 중 한 모터)
18	\$40000	EncLoss	인코더 소실 에러(좌표계 내 하나의 모터)
17	\$20000	-	(추후 사용하기 위해 예약)
16	\$10000	TimerEnabled	동작 타이머 유효
15	\$8000	HomeComplete	원점 복귀 완료(좌표계의 모든 모터)
14	\$4000	DesVelZero	명령 속도 제로(좌표계의 모든 모터)
13	\$2000	ClosedLoop	클로즈드 루프 모드(좌표계의 모든 모터)
12	\$1000	AmpEna	앰프 Enable(좌표계의 모든 모터)
11	\$800	InPos	인포지션(좌표계의 모든 모터)
10	\$400	-	(추후 사용하기 위해 예약)
9	\$200	BlockRequest	블록 요구 플래그 세트
8	\$100	TimersEnabled	타이머 유효
7	\$80	RadiusError(bit 1) ErrorStatus(bit 7)	XX/YY/ZZ 축 Circle 반경 에러/에러 워드 bit 7
6	\$40	RadiusError(bit 0) ErrorStatus(bit 6)	X/Y/Z 축 Circle 반경 에러/에러 워드 bit 6
5	\$20	SoftLimit ErrorStatus(bit 5)	소프트웨어 위치 리미트 정지(좌표계 중 한 모터)/에러 워드 bit 5
4	\$10	RunTimeError ErrorStatus(bit 4)	런타임 에러/에러 워드 bit 4
3	\$8	PvtError ErrorStatus(bit 3)	PVT 모드 에러/에러 워드 bit 3
2	\$4	LinToPvtError ErrorStatus(bit 2)	PVT - 직선 모드 에러/에러 워드 bit 2
1	\$2	ErrorStatus(bit 1)	버퍼 에러/에러 워드 bit 1
0	\$1	ErrorStatus(bit 0)	동기 할당 버퍼 에러/에러 워드 bit 0

이 요소값은 **&x?** 조회 명령에 응답하여 통지됩니다. 그 내용은 응답의 최초 16 진수 8 자리(1 개의 숫자당 4 비트)로 구성됩니다. 온라인 조회 명령 **backup Coord[x].Status** 는 Power PMAC 에 모든 요소를 통지시킵니다.

각각의 요소는 각 사양 항목을 통해 자세하게 설명합니다.

Coord[x].Status[1] 좌표계의 제 2 Status

설명: 좌표계 제 2 Status 워드

범위: \$0.. \$FFFFFFFF

단위: 비트 필드

Coord[x].Status[1]은 좌표계의 많은 글로벌 Status 비트를 저장하는 2 번째의 32 비트 Status 워드입니다. 워드 속의 Status 비트 목록은 다음 표와 같습니다.

비트#	16 진수값	요소명:Coord[x].{요소}	설명
31	\$80000000	Csolve	PMATCH 를 위한 유효한 좌표계 축 정의
30	\$40000000	LinToPvtBuf	Linear-PVT 변환 동작이 버퍼됨
29	\$20000000	FeedHold(비트 1)	피드 홀드를 통한 가속/감속 중
28	\$10000000	FeedHold(비트 0)	피드 홀드 타임 베이스 소스
27	\$8000000	BlockActive	블록 유효
26	\$4000000	ContMotion	연속 동작 요구
25	\$2000000	CCMode(bit 1)	커터 보정 모드 bit 1
24	\$1000000	CCMode(bit 0)	커터 보정 모드 bit 0
23	\$800000	MoveMode(bit 1)	동작모드 bit 1(=0 블렌드와 스플라인, =1 rapid 와 pvt 모드)
22	\$400000	MoveMode(bit 0)	동작모드 bit 0(=0 블렌드와 pvt, =1 rapid 와 스플라인 모드)
21	\$200000	SegMove(bit 1)	세그먼트한 PVT 모드 동작 중
20	\$100000	SegMove(bit 0)	세그먼트한 linear 모드 동작 중
19	\$80000	SegMoveAccel	최초의 세그먼트 동작
18	\$40000	SegMoveDecel	마지막 세그먼트 동작
17	\$20000	SegEnabled	세그먼트 유효
16	\$10000	SegStopReq	세그먼트 정지 요구
15	\$8000	LookAheadWrap LHStatus(bit 7)	룩 어헤드 순환
14	\$4000	LookAheadLookBack LHStatus(bit 6)	룩 어헤드 역행
13	\$2000	LookAheadDir LHStatus(bit 5)	룩 어헤드 방향
12	\$1000	LookAheadStop LHStatus(bit 4)	룩 어헤드 정지
11	\$800	LookAheadChange LHStatus(bit 3)	룩 어헤드 변화
10	\$400	LookAheadReCalc LHStatus(bit 2)	룩 어헤드 재계산
9	\$200	LookAheadFlush LHStatus(bit 1)	룩 어헤드 플러시
8	\$100	LookAheadActive LHStatus(bit 0)	룩 어헤드 유효
7	\$80	CCAddedArc	커터 보정 원호 추가
6	\$40	CCOffReq	커터 보정 턴 오프 요구
5	\$20		(예약)
4	\$10	CCMoveType(bit 1)	커터 보정 동작 타입 bit 1
3	\$8	CCMoveType(bit 0)	커터 보정 동작 타입 bit 0
2	\$4	CC3Active	3D 커터 보정은 유효
1	\$2	SharpCornerStop	예각에 따른 블렌드 무효
0	\$1	AddedDwellDis	추가 드웰 무효

이 요소값은 **&x?** 조회 명령에 응답하여 통지됩니다. 그 내용은 응답의 2 번째 16 진수 8 자리(1 개의 숫자당 4 비트)로 구성됩니다. 온라인 조회 명령 **backup Coord[x].Status** 는 Power PMAC 에 모든 요소를 통지시킵니다.

각각의 요소는 각 사양 항목을 통해 자세하게 설명합니다.

Coord[x].TimeBase 좌표계의 서보 사이클 경과 시간

설명: 보간 계산의 현재 "서보 사이클 경과 시간"

범위: 부동 소수점

단위: msec

Coord[x].TimeBase 는 최근의 서보 사이클에서 좌표계 궤적 보간을 계산할 때에 사용된 "경과 시간"(Δt) 값을 나타냅니다. 다시 말해 직전의 서보 사이클에서 보간 알고리즘이 이 값을 시간에 더합니다. 이 값은 서보 사이클의 물리적인 경과 시간과 일치하지 않아도 됩니다. 이 시간은 어플리케이션을 위해 고정됩니다. 물리적인 경과 시간에서 **Coord[x].TimeBase** 값으로 변경하는 것은 "타임 베이스 제어"에 "속도 오버라이드"와 같은 기법이 가능합니다.

타임 베이스를 위한 "목표" 값은 **Coord[x].pDesTimeBase** 로 지정한 주소의 레지스터 값이 됩니다. 이 레지스터값이 **Coord[x].TimeBase** 와 다른 경우에 목표값에 도달할 때까지 **Coord[x].TimeBase** 의 값에 저장 설정 요소 **Coord[x].TimeBaseSlew** 의 값이 서보 사이클마다 더해집니다.

(1 서보 사이클만큼 반드시 지체되는)슬루 레이트 제한을 회피하여 **Coord[x].TimeBase** 에 직접 쓸 수 있습니다. 이 경우에 자동으로 슬루 알고리즘이 직접 설정을 취소하지 않도록 보통 **TimeBaseSlew** 를 0 으로 설정하는 것이 바람직합니다.

Coord[x].TimerEnabled 좌표계의 동작 타이머 유효 상태

설명: "동작 타이머 유효" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].TimerEnabled 는 프로그램된 축 동작 또는 독립된 모터의 조그 동작이나 원점 동작을 통해 좌표계 중 한 모터가 유한 시간의 동작 또는 동작 섹션을 실행하는 경우에 1 이 됩니다. 그 이외에는 0 입니다. 이는 좌표계의 위치 결정 축에 할당된 모든 모터의 **Motor[x].Desired.TimerEnabled** Status 비트의 논리합입니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 16 에 해당합니다.

"무한" 조그 동작(j+ 또는 j-) 또는 원점 서치 동작의 트리거 전 동작에서는 동작의 정속도 부분이 유한 시간 섹션에 해당하지 않습니다. 단, 모터의 소프트웨어 리미트가 유효한 경우(**Motor[x].MaxPos > Motor[x].MinPos**) 조그 동작의 정속도 부분은 자동으로 유한 시간 섹션으로 변환됩니다.

Coord[x].TimersEnabled 좌표계의 모든 동작 타이머 상태

설명: "모든 동작 타이머 유효" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].TimersEnabled 는 좌표계에서 위치 결정 축에 할당된 모든 모터가 유한 시간 동작 또는 동작 섹션을 실행하는 경우에 1 이 됩니다. 그 이외에는 0 입니다. 이는 좌표계의 위치 결정 축에 할당된 모든 모터의 **Motor[x].Desired.TimerEnabled** Status 비트의 논리곱입니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 8에 해당합니다.

좌표계가 모션 프로그램 내의 명령 동작을 실행하면 동작에 명시적으로 명령하지 않았더라도 좌표계에 위치 결정 축에 할당된 모터는 모두 유한 시간 동작을 실시합니다.

Coord[x].Tsel 좌표계의 선택 변환 행렬

설명: 현재 선택된 변환 행렬

범위: -1 ..255

단위: 행렬 인덱스

Coord[x].Tsel 은 좌표계에서 현재 선택된 축 변환 행렬의 인덱스값을 나타냅니다(-1의 값은 변환 행렬이 선택되지 않았다는 것을 나타냅니다). **i**의 값이 0에서 255의 범위이면 이 인덱스로 지정한 변환 행렬을 포함한 **Tdata[i]** 데이터 구조체 요소가 프로그램 축 위치를 변환하기 위해 사용됩니다.

Coord[x].Tsel의 값은 버퍼 프로그램 명령 **tset{data}**를 사용하여 설정됩니다. 이 요소에 직접 쓸 수 없습니다.

Coord[x].TriggerMove 좌표계의 트리거 서치 진행 중 상태

설명: "트리거 서치 동작 진행중" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].TriggerMove 는 좌표계 중 한 모터가 트리거가 있는 동작(원점 서치, 트리거가 있는 JOG, 프로그램 RAPID 모드 트리거가 있는 동작)의 시작이며 1 이 됩니다. 미리 지정된 트리거 조건이 검출되어 트리거 후 동작이 시작될 때 또는 트리거 검출 없이 동작 종료하면 0 이 됩니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 31에 해당합니다.

Coord[x].TriggerNotFound 좌표계의 트리거 미검출 상태

설명: "트리거 미검출" Status 비트

범위: 0 .. 1

단위: 논리값

Coord[x].TriggerNotFound 는 좌표계 중 한 모터가 트리거가 있는 JOG 또는 프로그램 RAPID 모드 트리거가 있는 동작으로 사전에 지정된 트리거 조건을 검출하지 않고 종료한 경우에 1 이 됩니다. 이는 에러 상태가 아닙니다. 그러나 후속 동작은 이 비트의 세트 여부에 따라 달라집니다. 다음의 동작이 시작될 때 0 으로 복귀하고 그 이외일 때는 0 입니다. 이 비트는 32 비트 요소 **Coord[x].Status[0]**의 Bit 20 에 상당합니다.

Coord[x].TXYZ[j] 좌표계의 3D 공구 직경 보정 자세 벡터 성분

설명: 3D 공구 직경 보정 자세 벡터 성분값

범위: 부동 소수점

단위: 3D 단위 벡터 성분

Coord[x].TXYZ[j]는 X/Y/Z 공간의 3D 공구(커터) 반경 보정에서 현재의 공구 자세 벡터의 지정된 차원 성분값을 나타냅니다. 이 벡터에는 3 가지의 성분이 있습니다.

- **Coord[x].TXYZ[0]**: X/Y/Z 공간의 I-성분
- **Coord[x].TXYZ[1]**: X/Y/Z 공간의 J-성분
- **Coord[x].TXYZ[2]**: X/Y/Z 공간의 K-성분

이러한 값은 **txyz** 프로그램 명령으로 설정됩니다. 명령으로 지정된 벡터가 단위 벡터가 아니어도 저장된 성분값이 같은 방향의 단위 벡터가 됩니다.

Coord[x].XYZoff[j] 좌표계의 3D 공구 직경 보정 오프셋 벡터 성분

설명: 3D 공구 직경 보정 오프셋 벡터 성분

범위: 부동 소수점

단위: 직행 축 단위

Coord[x].XYZoff[j]는 X/Y/Z 공간의 3D 공구(커터) 직경 보정으로 마지막에 계산된 오프셋 벡터의 지정된 차원 성분값을 나타냅니다. 이 벡터에는 3 가지의 성분이 있습니다.

- **Coord[x].XYZoff[0]**: X-성분
- **Coord[x].XYZoff[1]**: Y-성분
- **Coord[x].XYZoff[2]**: Z-성분

이러한 요소의 값은 선언된 면 법선 벡터, 선언된 공구 자세 벡터 및 공구 선단 형상 테이블에 따른 3D 공구 직경 보정 알고리즘으로 설정됩니다. 이러한 성분으로 정의되고 얻을 수 있는 벡터는 동작에서 프로그램 위치와 "공구 중심" 명령 위치의 차이입니다.

Coord[x].Ldata 로컬 데이터 상태 요소

Coord[x].Ldata 서브 구조체에는 좌표계를 위해 일반화된 로컬 데이터를 지정하는 요소를 저장합니다. 이 서브 구조체는 각각 스크립트 PLC 프로그램 및 커뮤니케이션 스레드의 **LData** 서브 구조체와 동일합니다. **Ldata** 구조체는 아래의 설명을 참조하십시오.

Coord[x].CC3Data[i].3D 공구 직경 보정 상태 요소

Coord[x].CC3Data[i].서브 구조체에는 3D 커터(공구) 직경 보정용의 공구 선단 형상을 지정하는 요소를 저장합니다. 그 일부는 Power PMAC 으로 자동 계산되며 스크립트 환경 내의 읽기 전용입니다.

Coord[x].CC3Data[i].ToolOffset 좌표계의 3D 공구 직경 보정 축 오프셋

설명: 3D 공구 직경 보정 세그먼트 축 오프셋

범위: 부동 소수점

단위: 직선 축 사용자 단위

전원 투입 후 기본값: 0.0

Coord[x].CC3Data[i].ToolOffset 은 좌표계 **x** 에서 3 차원 공구 직경 보정에 사용되는 공구 선단 형상이 공구 중심점부터 "제 *i*"로 정의된 공구 선단 원호형 시작점까지의 부호화된 축 방향 거리를 나타냅니다.

사용자가 직접 지정한 비저장 설정 요소에서 정의된 전체 공구 선단 테이블의 시작점 **Coord[x].CC3Data[0].ToolOffset** 및 **Coord[x].CC3Data[0].ToolRadius** 는 공구 선단 테이블의 다른 섹션을 위한 기준점입니다. 후속 원호 섹션(*i*>0)에서 동등한 요소는 사용자 설정에 따른 **Coord[x].CC3Data[i].CutRadius** 및 **Coord[x].CC3Data[i].NdotT** 에 따라 PMAC 에서 자동으로 계산됩니다. 또한 해당 요소는 사용자 읽기 전용입니다.

Coord[x].CC3Data[i].ToolOffset 은 좌표계의 X, Y 및 Z 축 베이스로 대부분의 경우 밀리미터 또는 인치의 단위로 표시됩니다. 축 단위가 변환 행렬을 사용하여 재스케일링하는 경우에도 이는 재스케일링되지 않습니다. 양의 값인 경우에 테이블의 기준점은 공구 중심점에서 "상위 방향"입니다(중심점보다 공구 선단에서 멀어지는 방향). 음수인 경우에 기준점은 공구 중심점에서 "하위 방향"입니다(중심점보다 공구 선단에 가까운 방향). 3D 공구 직경 보정에서는 마치 공구 중심점이 표면에 접해 있는 것처럼 모션 프로그램 궤적이 쓰여져 있습니다.

보다 자세한 내용은 Power PMAC 사용자 설명서 내의 Power PMAC 동작 모드 궤적장의 3 차원 공구 직경 보정 항목을 참조하십시오.

Coord[x].CC3Data[i].ToolRadius 좌표계 3D 공구 직경 보정의 공구 직경

설명: 3D 공구 직경 보정 세그먼트 공구 반경

범위: 양의 부동 소수점

단위: 직선 축 사용자 단위

전원 투입 시 기본값: 0.0

Coord[x].CC3Data[i].ToolRadius 는 좌표계 **x** 의 3 차원 공구 직경 보정에 사용되는 공구 선단 형상에서 공구 선단의 "제 *i*"에서 정의된 원호 섹션의 공구 중심선 (회전축선)에서 시점까지의 수직 거리를 나타냅니다.

사용자가 직접 지정한 비저장 설정 요소에서 정의된 전체 공구 선단 테이블의 시작점 **Coord[x].CC3Data[0].ToolOffset** 및 **Coord[x].CC3Data[0].ToolRadius** 는 공구 선단 테이블의 다른 섹션을 위한 기준점입니다. 후속 원호 섹션(*i*>0)에서 동등한 요소는 사용자 설정에 따른 **Coord[x].CC3Data[i].CutRadius** 및 **Coord[x].CC3Data[i].NdotT** 에 따라 PMAC 에서 자동으로 계산됩니다. 또한 해당 요소는 사용자 읽기 전용 상태 요소입니다.

Coord[x].CC3Data[0].ToolRadius 는 좌표계의 X, Y 및 Z 축 베이스에서 대부분의 경우 밀리미터 또는 인치 단위로 표시됩니다. 축 단위가 변환 행렬을 사용하여 재스케일링되는 경우에도 이는 재스케일링되지 않습니다.

보다 자세한 내용은 Power PMAC 사용자 설명서 내의 Power PMAC 동작 모드 궤적장의 3 차원 공구 직경 보정 항목을 참조하십시오.

Coord[x].TPData[i].목표 위치 버퍼 상태 요소

Coord[x].TPData[i].서브 구조체에는 좌표계의 프로그램 동작에 이동 계산 시와 동작 실행 시 사이의 목표 위치 데이터 유지 요소를 저장합니다. 이 정보는 현재 실행 중인 이동 목표 위치 및 "이동 거리"의 통지를 지원하기 위해 유지되고 특히 CNC 스타일 어플리케이션에 도움이 됩니다.

Coord[x].TPData[i].Ncalc 좌표계의 목표 위치 버퍼 동기 레이블값

설명: 목표 위치 버퍼 동작 동기 레이블값

범위: 음이 아닌 정수(0 또는 양의 정수)

단위: 열거 형식

전원 투입 기본값: 0

Coord[x].TPData[i].Ncalc 는 좌표계 **x** 의 목표 위치 버퍼 내 "제 *i*" 프로그램 동작상의(있는 경우) 동기 레이블의 값을 나타냅니다. 만일 그 동작에 동기 레이블이 없는 경우에 0 이 됩니다.

인덱스값 *i* 는 0 에서 **Coord[x].TPSize** - 1 의 범위입니다. *i* = 0 의 서브 구조체는 현재 실행 중인 동작용입니다. *i* = 1 의 서브 구조체는 일련의 동작의 다음 동작용으로 그 후에도 마찬가지로입니다.

저장 설정 요소 **Coord[x].TPSize** 가 0 인 기본값인 경우에 목표 위치 버퍼는 사용되지 않습니다. 또한 이 요소의 값은 "nan"(번호가 아님)으로 통지합니다.

Coord[x].TPData[i].Pos[j] 좌표계의 목표 위치 버퍼 이동축 목표값

설명: 목표 위치 버퍼의 이동축 목표값

범위: 부동 소수점

단위: 축 위치 단위

전원 투입 기본값: 0.0

Coord[x].TPData[i].Pos[j]는 좌표계 **x** 의 목표 위치 버퍼 내 프로그램 동작상의 인덱스 "*j*"축 목표 위치를 나타냅니다. 축 인덱스값 *j* 의 범위는 0 에서 31 까지로 각각의 인덱스값에 대한 축 명칭은 다음 표와 같습니다.

축	<i>j</i>	축	<i>j</i>	축	<i>j</i>	축	<i>j</i>	축	<i>j</i>	축	<i>j</i>	축	<i>j</i>	축	<i>j</i>
A	0	V	4	Z	8	DD	12	HH	16	OO	20	SS	24	WW	28
B	1	W	5	AA	9	EE	13	LL	17	PP	21	TT	25	XX	29
C	2	X	6	BB	10	FF	14	MM	18	QQ	22	UU	26	YY	30
U	3	Y	7	CC	11	GG	15	NN	19	RR	23	VV	27	ZZ	31

인덱스값 i 는 0 에서 **Coord[x].TPSize** - 1 의 범위입니다. $i = 0$ 의 서브 구조체는 현재 실행 중인 동작용입니다. $i = 1$ 의 서브 구조체는 일련의 동작의 다음 동작용으로 그 후에도 마찬가지로입니다.

저장 설정 요소 **Coord[x].TPSize** 가 0 인 기본값인 경우에 목표 위치 버퍼는 사용되지 않습니다. 또한 이 요소의 값은 "nan"(번호가 아님)으로 통지합니다.

보통 이런 값은 사용자가 직접 액세스할 수 없습니다. 이는 이동 계산 시부터 동작 실행 시까지의 좌표를 저장하기 위해 사용됩니다. 이때 이동 목표 위치 및 "이동 거리" 조회로 통지되는 값을 계산하기 위해 Power PMAC 에서 사용됩니다.

Coord[x].TPData[j].XYZPos[j] 좌표계의 목표값 버퍼 공구 직경 보정 오프셋

설명: 목표 위치 버퍼 이동축 공구 직경 보정 오프셋

범위: 부동 소수점

단위: 축 위치 단위

전원 투입 기본값: 0.0

Coord[x].TPData[j].XYZPos[j]는 좌표계 x 의 목표 위치 버퍼 내의 "제 i " 프로그램 동작상의 인덱스 " j "축 목표 공구 직경 보정 오프셋을 나타냅니다. 축 인덱스값 j 는 0 부터 2 까지의 범위의 값으로 $j = 0$ 은 X 축 오프셋을 나타내며 $j = 1$ 은 Y 축 오프셋을 나타내며 또한 $j = 2$ 는 Z 축 오프셋을 나타냅니다.

인덱스값 i 는 0 에서 **Coord[x].TPSize** - 1 의 범위입니다. $i = 0$ 의 서브 구조체는 현재 실행 중인 동작용입니다. $i = 1$ 의 서브 구조체는 일련의 동작의 다음 동작용으로 그 후에도 마찬가지로입니다.

저장 설정 요소 **Coord[x].TPSize** 가 0 인 기본값인 경우에 목표 위치 버퍼는 사용되지 않습니다. 또한 이 요소의 값은 "nan"(번호가 아님)으로 통지합니다.

보통 이런 값은 사용자가 직접 액세스할 수 없습니다. 이는 이동 계산 시부터 이동 실행 시까지의 좌표를 저장하기 위해 사용됩니다. 이때 이동 목표 위치 및 "이동 거리" 조회로 통지되는 값을 계산하기 위해 Power PMAC 에서 사용됩니다.

Coord[x].TPExec.목표 위치 상태 요소

Coord[x].TPExec.서브 구조체에는 좌표계의 현재 실행 중인 프로그램 동작에서 동작 계산 시부터 동작 실행 시까지 버퍼된 목표 위치 데이터 유지 요소를 저장합니다. 이 정보는 현재 실행 중인 이동 목표 위치 및 "이동 거리" 통지를 지원하기 위해 유지되고 특히 CNC 스타일 어플리케이션에 도움이 됩니다.

Coord[x].TPExec.Ncalc 좌표계의 목표 위치 동기 레이블값

설명: 현재 실행 중인 동기 레이블값

범위: 음이 아닌 정수(0 또는 양의 정수)

단위: 열거 형식

전원 투입 기본값: 0

Coord[x].TPExec.Ncalc 는 좌표계 x 의 목표 위치 버퍼 내의 현재 실행 중인 프로그램 동작에 (있는 경우) 동기 레이블값을 나타냅니다. 만일 그 동작에 동기 레이블이 없는 경우에 0 이 됩니다.

저장 설정 요소 **Coord[x].TPSize** 가 0 인 기본값인 경우에 목표 위치 버퍼는 사용되지 않습니다. 또한 이 요소의 값은 "nan"(번호가 아님)으로 통지합니다.

Coord[x].TPExec.Pos[j] 좌표계의 목표 위치 동작축 목표값

설명: 현재 동작 중인 축의 목표값

범위: 음이 아닌 정수(0 또는 양의 정수)

단위: 축 위치 단위

전원 투입 기본값: 0.0

Coord[x].TPExec.Pos[j]는 좌표계 x 의 목표 위치 버퍼를 통한 현재 실행 중인 프로그램 동작상의 인덱스 " j "축의 목표 위치를 나타냅니다. 축 인덱스값 j 의 범위는 0 부터 31 까지로 각 인덱스값에 대한 축 명칭은 다음 표와 같습니다.

축	j	축	j	축	j	축	j	축	j	축	j	축	j	축	j
A	0	V	4	Z	8	DD	12	HH	16	OO	20	SS	24	WW	28
B	1	W	5	AA	9	EE	13	LL	17	PP	21	TT	25	XX	29
C	2	X	6	BB	10	FF	14	MM	18	QQ	22	UU	26	YY	30
U	3	Y	7	CC	11	GG	15	NN	19	RR	23	VV	27	ZZ	31

저장 설정 요소 **Coord[x].TPSize** 가 0 인 기본값인 경우에 목표 위치 버퍼는 사용되지 않습니다. 또한 이 요소의 값은 "nan"(번호가 아님)으로 통지합니다.

해당 값은 목표 위치 및 "이동 거리" 조회로 통지되는 값을 계산하기 위해 Power PMAC 에서 사용됩니다.

Coord[x].TPExec.XYZPos[j] 좌표계의 목표 위치 공구 직경 보정 오프셋

설명: 현재 실행 중인 축 공구 직경 보정 오프셋

범위: 음이 아닌 정수(0 또는 양의 정수)

단위: 축 위치 단위

전원 투입 기본값: 0.0

Coord[x].TPExec.XYZPos[j]는 좌표계 **x** 의 목표 위치 버퍼를 통한 현재 실행 중인 프로그램 동작상의 인덱스 "**j**"축 목표 공구 직경 보정 오프셋을 나타냅니다. 축 인덱스값 **j**는 0 부터 2 까지의 범위 값으로 **j = 0** 은 축 오프셋을 나타내며 **j = 1** 은 Y 축 오프셋을 나타내며 또한 **j = 2** 는 Z 축 오프셋을 나타냅니다.

저장 설정 요소 **Coord[x].TPSize** 가 0 인 기본값인 경우에 목표 위치 버퍼는 사용되지 않습니다. 또한 이 요소의 값은 "nan"(번호가 아님)으로 통지합니다.

목표 위치 및 "이동 거리" 조회로 통지하는 값을 계산하기 위해 해당 값은 Power PMAC 에서 사용됩니다.

ECAT[i].STATUS 데이터 구조체 요소

다음의 구조체는 EtherCAT 네트워크의 Status 에 관한 자세한 내용을 제공합니다.

ECAT[i].DCClockDiff

설명	Power PMAC 클럭과 분산 기준 클럭의 시간차
범위:	정수
단위:	나노초
기본값:	0

[ECAT[i].DCClockDiff]에서는 Power PMAC 의 내부 클럭과 선택된 슬레이브 장치 분산 기준 클럭의 시간차를 지정합니다. 이 차가 저장 설정 요소 [ECAT[i].DCRefBand]에 따라 지정된 값을 초과하면 Power PMAC 의 내부 페이즈 클럭 주파수는 방향에 따라 지정된 [ECAT[i].DCRefPlus] 또는 [ECAT[i].DCRefMinus]의 양으로 이 차이가 작아지도록 조정합니다.

ECAT[i].Error

설명:	EtherCAT 네트워크 유효화의 에러 코드
범위:	-99 ... 0
단위:	열거

[ECAT[i].Error]에는 최근 EtherCAT 네트워크의 유효화를 시행했을 때의 에러 코드가 저장됩니다. [ECAT[i].Enable]의 인덱스 'i' 를 1 로 설정하면 에러가 검출됩니다. [ECAT[i].Error]가 0 인 경우에는 유효화가 성공하여 에러는 발생하지 않습니다. [ECAT[i].Error]에 포함된 값이 0 보다 작은 경우에 유효화가 실패했다는 것을 나타내며 값에 따라 특정한 에러가 표시됩니다. 현재 탑재된 에러 코드는 다음과 같습니다.

- -1: EtherCAT 마스터 어플리케이션 라이브러리가 존재하지 않음
- -2: EtherCAT 마스터 커널 라이브러리가 존재하지 않음
- -3: 마스터에 액세스할 수 없음. EtherCAT 어댑터의 구성이 올바르지 않을 가능성이 있음
- -8: 구성에 실수가 있는 슬레이브 장치가 유효화([ECAT[i].Slave[j].Enable] = 1 이지만 다른 슬레이브 설정에 실수가 있음)
- -9: 구성 내의 PDO 요구가 무효([ECAT[i].IO[k]]가 올바르게 구성되지 않음)
- -10: PDI 구성이 무효 [ECAT[i].LPIO[k]]가 올바르게 구성되지 않음.
[ECAT[i].LPIO[k].Slave]가 [ECAT[i].Slave]에 할당되었지만 일치하는 [ECAT[i].Slave[j].PDO[k].Index]의 [ECAT[i].LPIO[k].Index]가 없음
- -11: EtherCAT 마스터를 시작할 수 없음
- -12: I/O 가 지정된 슬레이브 장치(ECAT[i].IO[k].Slave)가 유효화되지 않음
- -13: [ECAT[i].BGIO[j].Slave]가 유효화되지 않음
- -14: EtherCAT 의 라이선스에 문제가 있음. 구입한 EtherCAT 축 수를 초과하면 안 됨

ECAT[i].LinkUp

설명: 마스터 연결 Status

범위: 0, 1

단위: 부울 형식

[ECAT[i].LinkUp]은 인덱스 'i'의 EtherCAT 마스터가 장치에 연결되어 있는지에 여부를 나타냅니다. [ECAT[i].RTStateCheck] = 1 이어야 합니다. 값은 다음과 같습니다.

- 0: 장치에 연결되지 않음
- 1: 장치에 연결됨

ECAT[i].LPDomainOutputState

설명: EtherCAT 백그라운드 프로세스 출력 교환의 현재 상태

범위: 0 ... 3

단위: 열거

[ECAT[i].LPDomainOutputState]에는 인덱스 'i'의 EtherCAT 네트워크에서 실행되는 백그라운드(우선도가 낮음) 프로세스 데이터 출력 전송의 현재 상태를 나타내는 값이 포함됩니다. [ECAT[i].LPStateCheck]가 1로 설정되고 실시간 전송 상태의 자동 확인이 유효한 경우에 한해 이 요소값이 설정됩니다. 상태를 나타내는 값은 다음과 같습니다.

- 0: 등록된 프로세스 데이터가 교환되지 않음
- 1: 등록된 프로세스 데이터의 일부가 교환됨
- 2: (예약 완료)
- 3: 등록된 모든 프로세스 데이터가 교환됨

[ECAT[i].LPnotLRW]가 기본값인 0으로 설정되면 공통의 입력/출력 교환이 유효화되고 입력 및 출력 교환의 공통 상태가 [ECAT[i].LPDomainState]로 통지됩니다. 따라서 이 요소는 사용되지 않습니다. [ECAT[i].LPnotLRW]가 1로 설정되고 일반적인 입력/출력 교환이 무효화된 경우에는 개별적인 출력 교환 상태가 [ECAT[i].LPDomainOutputState]로 통지됩니다.

ECAT[i].LPDomainState

설명: EtherCAT 백그라운드 프로세스 I/O 교환의 현재 상태

범위: 0 ... 3

단위: 열거

[ECAT[i].LPDomainState]에는 인덱스 'i'의 EtherCAT 네트워크에서 실행되는 백그라운드(우선도가 낮음) 프로세스 데이터 출력 전송의 현재 상태를 나타내는 값이 포함됩니다. [ECAT[i].LPStateCheck]가 1로 설정되고 백그라운드 전송 상태의 자동 확인이 유효한 경우에 한해 이 요소값이 설정됩니다. 상태를 나타내는 값은 다음과 같습니다.

- 0: 등록된 프로세스 데이터가 교환되지 않음
- 1: 등록된 프로세스 데이터의 일부가 교환됨
- 2: (예약 완료)
- 3: 등록된 모든 프로세스 데이터가 교환됨

[ECAT[i].LPnotLRW]가 기본값인 0 으로 설정된 경우에 이 통지된 상태는 입력과 출력 양쪽의 교환에 유효합니다. [ECAT[i].LPnotLRW]가 1 로 설정되고 공통 입력/출력 교환이 무효화된 경우에 여기서 통지된 상태는 입력 교환에 한해 유효하고 출력 교환에는 [ECAT[i].LPOutputDomainState]가 사용됩니다.

ECAT[i].LPRxTime

설명: EtherCAT 네트워크의 백그라운드 수신 시간

범위: 음의 값이 아닌 부동 소수점

단위: 마이크로초

[ECAT[i].LPRxTime]에는 인덱스 'i'의 EtherCAT 네트워크에서 이전의 백그라운드 네트워크 사이클 중의 입력 데이터와 Status 데이터의 수신에 사용된 시간이 마이크로초로 포함됩니다([ECAT[i].Enable] = 2 또는 3 인 경우). 이는 주로 진단에 사용됩니다.

ECAT[i].LPTxTime

설명: EtherCAT 네트워크의 백그라운드 전송 시간

범위: 음의 값이 아닌 부동 소수점

단위: 마이크로초

[ECAT[i].LPTxTime]에는 인덱스 'i'의 EtherCAT 네트워크에서 이전의 백그라운드 네트워크 사이클 중의 출력 데이터와 명령 데이터 전송에 사용한 시간이 마이크로초로 포함됩니다([ECAT[i].Enable] = 2 또는 3 인 경우). 이는 주로 진단에 사용됩니다.

ECAT[i].MasterState

설명: EtherCAT 네트워크 마스터의 현재 상태

범위: 0, 1, 2, 4, 8

단위: 열거

[ECAT[i].MasterState]에는 인덱스 'i'의 EtherCAT 네트워크에서 마스터의 현재 옵션 상태를 나타내는 값이 포함됩니다. 이 요소값은 [ECAT[i].RTStateCheck]가 1 로 설정되고 마스터 상태의 자동 확인이 유효화된 경우에 한해 설정됩니다. 상태를 나타내는 값은 다음과 같습니다.

- 1: INIT(초기화)
- 2: PREOP(동작 준비 중)
- 4: SAFEOP(세이프티 모드에서 동작 중)
- 8: OP(완전히 동작 중)

ECAT[i].RTDomainOutputState

설명: EtherCAT 의 주기적인 프로세스 출력 교환의 현재 상태

범위: 0 ... 3

단위: 열거

[ECAT[i].RTDomainOutputState]에는 인덱스 'i'의 EtherCAT 네트워크에서 실행되는 주기적인 (실시간)프로세스 데이터 출력 전송의 현재 상태를 나타내는 값이 포함됩니다. [ECAT[i].RTStateCheck]가 1로 설정되고 실시간 전송 상태의 자동 확인이 유효한 경우에만 이 요소값이 설정됩니다. 상태를 나타내는 값은 다음과 같습니다.

- 0: 등록된 프로세스 데이터가 교환되지 않음
- 1: 등록된 프로세스 데이터의 일부가 교환됨
- 2: (예약 완료)
- 3: 등록된 모든 프로세스 데이터가 교환됨

[ECAT[i].RTnotLRW]가 기본값인 0으로 설정되면 공통 입력/출력 교환이 유효화되고 입력 및 출력 교환의 공통 상태가 [ECAT[i].RTDomainState]로 통지됩니다. 따라서 이 요소는 사용되지 않습니다. [ECAT[i].RTnotLRW]가 1로 설정되고 일반적인 입력/출력 교환이 무효화된 경우에는 개별적인 출력 교환 상태가 [ECAT[i].RTDomainOutputState]로 통지됩니다.

ECAT[i].RTDomainState

설명: EtherCAT 의 주기적인 프로세스 I/O 교환의 현재 상태

범위: 0 ... 3

단위: 열거

[ECAT[i].RTDomainState]에는 인덱스 'i'의 EtherCAT 네트워크에서 실행되는 주기적인 (실시간)프로세스 데이터 전송의 현재 상태를 나타내는 값이 포함됩니다. [ECAT[i].RTStateCheck]가 1로 설정되고 실시간 전송 상태의 자동 확인이 유효한 경우에만 이 요소값이 설정됩니다. 상태를 나타내는 값은 다음과 같습니다.

- 0: 등록된 프로세스 데이터가 교환되지 않음
- 1: 등록된 프로세스 데이터의 일부가 교환됨
- 2: (예약 완료)
- 3: 등록된 모든 프로세스 데이터가 교환됨

[ECAT[i].RTnotLRW]가 기본값인 0으로 설정된 경우에 이 통지된 상태는 입력과 출력 양쪽의 교환에 유효합니다. [ECAT[i].RTnotLRW]가 1로 설정되고 공통 입력/출력 교환이 무효화된 경우에 여기서 통지된 상태는 입력 교환에 한해 유효하고 [ECAT[i].RTOutputDomainState]의 출력 교환을 감시할 수 있습니다.

ECAT[i].RxTime

설명: EtherCAT 네트워크의 포어그라운드 수신 시간

범위: 음의 값이 아닌 부동 소수점

단위: 마이크로초

[ECAT[i].RxTime]에는 인덱스 'i'의 EtherCAT 네트워크에서 이전의 포어그라운드 네트워크 사이클 중의 입력 데이터와 Status 데이터의 수신에 사용된 시간이 마이크로초로 포함됩니다([ECAT[i].Enable] = 1 인 경우). 이는 주로 진단에 사용됩니다.

ECAT[i].TxTime

설명: EtherCAT 네트워크의 포어그라운드 전송 시간

범위: 음의 값이 아닌 부동 소수점

단위: 마이크로초

[ECAT[i].TxTime]에는 인덱스 'i'의 EtherCAT 네트워크에서 이전의 포어그라운드 네트워크 사이클 중의 출력 데이터와 명령 데이터 전송에 사용된 시간이 마이크로초로 포함됩니다([ECAT[i].Enable] = 1 인 경우). 이는 주로 진단에 사용됩니다.

EtherCAT 슬레이브의 Status 요소

ECAT[j].Slave[j].Online

설명: 슬레이브의 온라인 여부를 나타내는 Status 비트

범위: 0, 1

단위: 없음

기본값: 0

레거시 'I' 변수: 없음

슬레이브가 물리적으로 EtherCAT 네트워크에 연결되고 전원이 켜진 상태인 경우에 이 파라미터의 값은 1 이 됩니다. 그렇지 않은 경우에는 0 이 됩니다.

ECAT[j].Slave[j].State

설명: 슬레이브의 현재 상태

범위: 1, 2, 4, 8

단위: 없음

기본값: 0

레거시 'I' 변수: 없음

슬레이브의 현재 상태를 나타내며 1 = INIT, 2 = PREOP, 4 = SAFEOP, 8 = OP 입니다. 이 변수는 [ECAT[j].RTStateCheck] = 1 인 경우에 한해 사용됩니다. 그렇지 않은 경우에 이 변수는 감시되지 않습니다.

Gate1[i].(PMAC2 스타일 서보 ASIC)상태 요소(V1.6 2014.03)

PMAC2 스타일 DSPGATE1 서보 ASIC에는 4 채널의 서보 인터페이스를 갖추고 있습니다. 본 장의 상태 요소에서는 프로세서가 이 ASIC의 외부 입력으로 얻어진 값에 액세스할 수 있습니다. 인덱스값 i 는 4~19까지의 범위가 됩니다(0 - 3은 예약되었습니다).

Gate1[i].멀티채널 상태 요소

본 장의 상태 요소는 ASIC 채널에 관한 것이 아닙니다.

Gate1[i].PartData[k] 게이트 1의 IC 보드 부품 원시 데이터

설명: DSPGATE1 IC 보드의 원시 부품 데이터

범위: \$0..\$FFFF

단위: 없음

Gate1[i].PartData[k]에는 DSPGATE1 ASIC를 내장한 보드상의 식별 IC에서의 4 레지스터($k = 0 \sim 3$) 중 하나의 원시 데이터를 저장됩니다. 이는 상태 요소 **Gate1[i].PartNum**, **Gate1[i].PartOpt** 그리고 **Gate1[i].PartRev**를 구하기 위해 사용됩니다.

C에 따라 요소에 직접 액세스할 수 없습니다. C에서는 **Sys.piom** + (IC 기준 주소) + 0x40, 0x44, 0x48 또는 0x4C의 주소에 대한 포인터 세트가 각각 **Gate1[i].PartData[0]**에서 **PartData[3]**의 레지스터에 액세스하기 위해 사용됩니다.

Gate1[i].PartData[k].{start}.{width} 구문을 사용하여 **Gate1[i].PartData[k]**의 특정한 비트 범위값을 참조할 수 있습니다. 여기서 **{start}**는 범위 내의 하위 비트 번호로 **{width}**는 비트 수입니다.

Gate1[i].PartNum 게이트 1의 하드웨어 부품 번호

설명: DSPGATE1 IC 보드의 하드웨어 부품 번호

범위: 음이 아닌 정수

단위: 열거 형식

Gate1[i].PartNum에는 DSPGATE1 ASIC를 내장한 보드의 6자리 부품 번호를 저장합니다. 이는 ASIC와 관련한 식별 IC에서 구해지며 전원 투입/리셋 시 자동으로 읽힙니다. 이 인덱스의 ASIC가 존재하지 않는 경우에 0이 통지됩니다.

현재 기존의 부품 번호는 다음과 같습니다.

- ACC-24C2: 603681
- ACC-24C2A: 603611
- ACC-24E2: 603397
- ACC-24E2A: 603398
- ACC-24E2S: 603441
- ACC-51C: 603680
- ACC-51E: 603438

이 요소는 실제 ASIC의 하드웨어 레지스터가 아닙니다. 이는 ASIC와 관련한 소프트웨어 레지스터입니다. 주로 IDE의 시스템 설정 유틸리티로 사용됩니다.

C에서 이 요소는 **Gate1PartData[i].Num**으로 액세스됩니다.

Gate1[i].PartOpt 게이트 1의 옵션 구성 코드

설명: DSPGATE1 IC 보드 옵션 하드웨어 구성 코드

범위: 음이 아닌 정수

단위: 열거 형식

Gate1[i].PartOpt에는 DSPGATE1 IC를 내장한 보드에 존재하는 하드웨어 옵션을 나타내는 코드를 저장합니다. 이는 ASIC와 관련한 식별 IC에서 구해지며 전원 투입/리셋 시 자동으로 읽힙니다. 이 인덱스의 ASIC가 존재하지 않는 경우에 "nan"(not-a-number 숫자가 아님)으로 통지됩니다.

이 요소는 실제 ASIC의 하드웨어 레지스터가 아닙니다. 이는 ASIC와 관련한 소프트웨어 레지스터입니다. 주로 IDE의 시스템 설정 유틸리티로 사용됩니다.

C에서는 이 요소는 **Gate1PartData[i].Opt**로 액세스됩니다.

Gate1[i].PartRev 게이트 1의 하드웨어 개정 번호

설명: DSPGATE1 IC 보드의 하드웨어 개정 번호

범위: 음이 아닌 정수

단위: 열거 형식

Gate1[i].PartRev에는 DSPGATE1 ASIC를 내장한 보드의 개정 번호를 저장합니다. 이는 ASIC와 관련한 식별 IC에서 구해지며 전원 투입/리셋 시 자동으로 읽힙니다. 이 수는 보드의 3자리 부품 개정 번호의 마지막 자릿수에 해당합니다.

이 요소는 실제 ASIC의 하드웨어 레지스터가 아닙니다. 이는 ASIC와 관련한 소프트웨어 레지스터입니다. 주로 IDE의 시스템 설정 유틸리티로 사용됩니다.

C에서는 이 요소는 **Gate1PartData[i].Rev**로 액세스됩니다.

Gate1[i].PartType 게이트 1의 하드웨어 부품 타입

설명: DSPGATE1 IC 보드의 하드웨어 부품 타입

범위: 음이 아닌 정수

단위: 비트 필드

Gate1[i].PartType에는 DSPGATE1 ASIC를 내장한 보드의 하드웨어 인터페이스 타입을 나타내는 정수를 저장합니다. 이는 ASIC와 관련한 식별 IC에서 구해지며 전원 투입/리셋 시 자동으로 읽힙니다. 이 인덱스의 ASIC가 존재하지 않는 경우에 "nan"(not-a-number 숫자가 아님)으로 통지됩니다.

각각의 비트는 존재하는 하나의 인터페이스 타입을 나타냅니다. IC에 그 인터페이스 타입이 존재하는 경우에 비트가 설정됩니다. 여러 인터페이스 타입이 존재할 수 있습니다.

- 비트 0(값 1): 서보
- 비트 1(값 2): MACRO
- 비트 2(값 4): 아날로그 I/O
- 비트 3(값 8): 디지털 입출력

현재 모든 DSPGATE1 IC 보드에 대한 **PartType**의 값은 1(서보 인터페이스만 해당)입니다.

이 요소는 실제 ASIC의 하드웨어 레지스터가 아닙니다. 이는 ASIC와 관련한 소프트웨어 레지스터입니다. 주로 IDE의 시스템 설정 유틸리티로 사용됩니다.

C에서는 이 요소는 **Gate1PartData[i].Type**으로 액세스됩니다.

Gate1[j].채널 고유 상태 요소

이 채널의 상태 요소에서는 ASIC 채널에 연결된 입력 센서값을 저장한 레지스터 요소나 채널의 현황 정보를 나타내는 개별적인 Status 비트를 포함합니다. 채널 인덱스 j 는 0~3 의 범위에서 각각 하드웨어 채널의 1~4 를 나타냅니다.

Gate1[j].Chan[j].ABC 게이트 1 의 엔코더 채널 입력값

설명: A, B 와 C 엔코더 채널 입력값

범위: 0 .. 7

단위: 비트 필드

Gate1[j].Chan[j].ABC 에는 최신 서보 인터럽트에서 래치된 IC 채널의 A, B 그리고 C 엔코더 입력의 현재 논리 상태를 저장합니다. 비트 2(값 4)는 C(인덱스) 채널의 상태를 나타냅니다. 비트 1(값 2)은 B 채널의 상태를 나타냅니다. 그리고 비트 0(값 1)은 A 채널의 상태를 나타냅니다.

비트가 1 인 값은 IC 상의 high 전압에 해당합니다. 0 인 값은 IC 상의 low 전압에 해당합니다. 대부분의 Delta Tau 하드웨어에서 이는 차동 입력 회로로 그 경우에는 "+" 입력이 "-" 입력보다 높은 전압일 때 1 의 값이 됩니다.

Gate1[j].Chan[j].ABC에서는 스크립트 환경에서 풀 워드 요소 **Gate1[j].Chan[j].Status** 의 비트 12 - 14 를 구성합니다(C 에서는 비트 20 - 22). C 환경에서는 풀 워드 요소로 액세스해야 합니다.

Gate1[j].Chan[j].Adc[k] 게이트 1 의 ADC 값

설명: 아날로그/디지털 변환값

범위: -8,388,608 .. 8,388,607(스크립트)
-2³¹ .. 2³¹-1 (C)

단위: 24 비트 ADC LSB(스크립트)
32 비트 ADC LSB(C)

Gate1[j].Chan[j].Adc[k]에는 최신의 위상 클록 사이클에서 래치된 IC 채널의 위상 아날로그-디지털 변환값을 저장합니다. 요소 **Adc[0]**에는 채널 Phase A 입력에 연결된 A/D 변환값을 저장합니다. **Adc[1]**에는 채널 Phase B 입력에 연결된 A/D 변환값을 저장합니다.

회로와 레지스터는 24 비트 ADC 까지 지원합니다. 인터페이스가 올바르게 설정되는 경우에 n-비트 ADC 데이터는 이 24 비트 요소의 상위 n 비트에 존재합니다. Power PMAC 의 "다이렉트 PWM" 전류(commutation) 알고리즘의 전류 피드백에 사용될 때 본래의 데이터는 최상위 비트에 존재해야 합니다.

ADC 가 트루 데이터에 선행하는 "헤더 비트"를 가지는 경우에 적절한 저장 설정 요소 **Gate1[j].AdcStrobe** 의 설정은 헤더 데이터를 이 요소의 하위 1 - 4 비트에 "롤 오버"시킵니다.

C 프로그램과 함수는 본래의 데이터는 상위(최대) 24 비트에 한하지만 32 비트값으로 이 요소를 액세스해야 한다는 점에 주의하십시오.

Gate1[i].Chan[j].CountError 게이트 1의 엔코더 디코드 카운트 에러

설명: 엔코더 디코드 카운트 에러

범위: 0 .. 1

단위: 논리값

Gate1[i].Chan[j].CountError에는 IC의 "엔코더 카운트 에러" 상태 플래그의 현재 논리 상태를 저장합니다. 1의 값은 엔코더 샘플 클럭(SCLK) 사이클에 엔코더에서 A, B 채널 양쪽의 입력 상태가 동시에 변화하는 무효의 AB 상 이동이 발생했다는 것을 나타냅니다. 이 무효 이동이 발생할 때 이 플래그는 설정되고 다음에 계속되는 어떤 엔코더 카운터의 읽기 래치된 값은 잘못된 값이라는 것을 나타냅니다.

Gate1[i].Chan[j].CountError는 Power PMAC의 전원 투입/리셋으로 강제로 0으로 설정됩니다. 일단 설정되면 이 플래그는 **Gate1[i].Chan[j].PosClear**를 1로 설정하고 채널의 엔코더 카운터만 클리어하면 0으로 리셋할 수 있습니다. 이를 실시하기 위해 **Gate1[i].Chan[j].AmpEna**을 0으로 설정해야 합니다. 서보 루프가 위치 피드백값의 급격한 변화에도 과민하게 반응하므로 엔코더 카운터 위치를 클리어하는 것은 매우 위험한 동작일 가능성이 있다는 점을 주의하십시오.

Gate1[i].Chan[j].CountError는 스크립트 환경에서는 풀 워드 요소 **Gate1[i].Chan[j].Status**의 비트 8입니다(C에서는 비트 16입니다). C 환경에서는 풀 워드 요소로 액세스해야 합니다.

Gate1[i].Chan[j].EncLossN 게이트 1의 엔코더 소실 비트

설명: 엔코더 소실 비트

범위: 0 .. 1

단위: 논리값

Gate1[i].Chan[j].EncLossN에는 ASIC를 동반한 AB 상 엔코더 검출 회로가 유효한 엔코더 신호를 검출했는지에 대해 나타냅니다. 그것이 유효 신호로 보는 경우에는 1입니다. 그것이 유효 신호로 간주되지 않는 경우에는 0입니다. 회로는 차동 신호 페어 상의 배타적 논리합 게이트를 사용합니다. 따라서 이 검출은 차동 라인 드라이버 출력을 가진 엔코더에 한해 유효합니다.

이 Status 비트가 ASIC 자체에는 실제로 존재하지 않는다는 점에 주의하십시오. 이는 ACC-24E2, ACC-24E2A 그리고 ACC-24E2S를 포함한 어떤 액세서리상의 ASIC를 동반하는 회로에서 생성됩니다.

Gate1[i].Chan[j].Equ 게이트 1 의 위치 비교 플래그 출력

설명: 위치 비교 플래그 출력값

범위: 0 .. 1

단위: 논리값

Gate1[i].Chan[j].Equ 에는 IC 채널의 "EQU" 위치 비교 플래그 출력의 현재 논리 상태를 저장합니다. 1 의 값은 IC 의 high 전압 출력에 대응합니다. 0 의 값은 IC 의 low 전압 출력에 대응합니다. 대부분의 Delta Tau 하드웨어는 이 신호로 싱크 출력 드라이버를 관리합니다. 따라서 1 은 출력 드라이버에 전류가 유입되고 low 출력 전압에 인입하는 온 상태입니다.

출력 상태를 직접 설정하기 위해 이 요소를 사용할 수 없습니다. 출력값을 강제로 설정하기 위해서는 2 비트의 비저장 설정 요소 **Gate1[i].Chan[j].EquWrite** 를 사용해야 합니다. 엔코더 카운터값이 **Gate1[i].Chan[j].CompA** 와 **Gate1[i].Chan[j].CompB** 로 설정한 위치를 통과할 때 플래그의 값이 자동으로 변화합니다.

Gate1[i].Chan[j].Equ 는 스크립트 환경에서 풀 워드 요소 **Gate1[i].Chan[j].Status** 의 비트 9 입니다(C에서는 비트 17 입니다). C 환경에서는 풀 워드 요소에 따라 액세스해야 합니다.

Gate1[i].Chan[j].Fault 게이트 1 의 앰프 이상 플래그 입력값

설명: 앰프 이상 플래그 입력값

범위: 0 .. 1

단위: 논리값

Gate1[i].Chan[j].Fault 에는 최신 서보 인터럽트에서 래치된 IC 채널의 FAULT 플래그 입력(앰프 이상)의 현재 논리 상태를 저장합니다. 1 의 값은 IC 에 대한 high 전압에 대응합니다. 0 의 값은 IC 에 대한 low 전압에 대응합니다. 많은 Delta Tau 하드웨어에서는 이는 광절연 입력입니다. 그리고 플래그가 "단함"일 때 광 아이솔레이터에 전류가 흐르고 0 이 됩니다. 이는 또한 차동 입력으로 그 경우에는 FAULT+입력이 FAULT-입력보다 높은 전압일 때 1 이 됩니다.

이 요소가 단순한 IC 에 대한 입력 레벨을 나타낸다는 점에 주의하십시오. 0 또는 1 상태 중 하나에서 앰프 이상으로 판단할지 여부는 **Motor[x].AmpFaultLevel** 의 설정에 따릅니다.

이 하드웨어 입력 요소는 모터 소프트웨어 상태 요소 **Motor[x].AmpFault** 와는 다릅니다. **Motor[x].AmpFault** 는 모터가 이 입력을 사용하는 설정이 되어 있으면 이 입력에 따른 상태를 나타냅니다.

Gate1[i].Chan[j].Fault 는 스크립트 환경의 풀 워드 요소 **Gate1[i].Chan[j].Status** 의 비트 17 입니다(C에서는 비트 25 입니다). C 환경에서는 풀 워드 요소로 액세스해야 합니다.

Gate1[j].Chan[j].HallState 게이트 1의 포착 홀 위치 상태

설명: 포착 홀 위치 상태값

범위: 0 .. 7

단위: 열거 형식

채널이 U, V 또는 W 중 어느 하나의 이동으로 엔코더 위치를 포착하도록 설정될 때(**Gate1[j].Chan[j].CaptCtrl** 비트 0과 1이 0으로 **Gate1[j].Chan[j].GatedIndexSel**이 1로 설정) **Gate1[j].Chan[j].HallState**는 최신 홀 사이클에서 포착된 U, V 및 W 플래그 입력에 의한 위치 상태를 저장합니다.

홀 사이클 6 상태의 유효한 위치 상태값 범위는 0~5입니다. U, V 그리고 W 플래그 입력의 유효한 입력값에 이러한 6개의 값이 어떻게 대응하는지는 다음 표와 같습니다.

Gate1[j].Chan[j].HallState	0	1	2	3	4	5
Gate1[j].Chan[j].UVW	1	3	2	6	4	5
U 플래그 입력 상태	0	0	0	1	1	1
V 플래그 입력 상태	0	1	1	1	0	0
W 플래그 입력 상태	1	1	0	0	0	1

Power PMAC은 상태 0과 상태 1 간의 이동을 홀 사이클 기준 영도점으로 간주합니다. 이것이 전류(commutation) 사이클의 영도점으로 규정하는 경우에 센서 위치에서 전류(commutation) 각도의 오프셋은 필요하지 않습니다.

어떤 이동이 발생하기 전 또는 무효인 UVW 플래그 입력 상태로 이동한 경우에 **Gate1[j].Chan[j].HallState**는 7의 값을 통지합니다.

Gate1[j].Chan[j].HallState는 스크립트 환경의 풀 워드 요소 **Gate1[j].Chan[j].Status**의 비트 0 - 2를 구성합니다(C에서는 비트 8 - 10입니다). C 환경에서는 풀 워드 요소로 액세스해야 합니다.

Gate1[j].Chan[j].HomeCapt 게이트 1의 외부 트리거 래치 엔코더 카운트

설명: 외부 트리거 래치된 엔코더 카운트값

범위: 0 .. 16777215

단위: 엔코더 카운트

Gate1[j].Chan[j].HomeCapt에는 최신의 외부 트리거에서 래치된 IC 채널의 하드웨어 엔코더 카운트값을 저장합니다. 이는 **Gate1[j].Chan[j].EncCtrl**로 지정한 엔코더 디코드에서 결정되는 엔코더 카운트 단위를 통한 24 비트값입니다. 카운터는 전원 투입/리셋으로 0으로 설정되고 그때부터 카운트됩니다. 따라서 항상 전원 투입/리셋 위치가 기준이 됩니다. 이는 모든 방향으로 롤 오버할 수 있습니다.

트리거 상태는 저장 설정 요소 **Gate1[j].Chan[j].CaptCtrl** 그리고 **Gate1[j].Chan[j].CaptFlagSel**에서 선택됩니다.

Gate1[i].Chan[j].HomeCapt 를 읽는 행동은 다음의 포착을 위한 트리거 회로를 다시 작동 할 수 있습니다. 레벨 트리거가 사용되므로 트리거하는 입력이 계속 지정된 상태라면 바로 다시 트리거가 발생하고(가능성으로) 이 요소에 새로운 카운트값을 래치합니다.

C 프로그램에서 액세스되는 경우에 이 요소는 상위 24 비트에 실제 데이터를 배치한 32 비트값입니다.

Gate1[i].Chan[j].HomeFlag 게이트 1 의 원점 플래그 입력

설명: 원점 플래그 입력값

범위: 0 .. 1

단위: 논리값

Gate1[i].Chan[j].HomeFlag 에는 최신 서보 인터럽트에서 래치된 IC 채널의 원점 플래그 입력의 현재 논리 상태를 저장합니다. 1 의 값은 IC 에 대한 high 전압에 대응합니다. 0 의 값은 IC 에 대한 low 전압에 대응합니다. 대부분의 Delta Tau 하드웨어에서는 이는 광절연 입력입니다. 그리고 플래그가 "단함"일 때 광 아이솔레이터에 전류가 흐르고 0 이 됩니다.

Gate1[i].Chan[j].HomeFlag 는 스크립트 환경 풀 워드 요소 **Gate1[i].Chan[j].Status** 의 비트 16 입니다(C 에서는 비트 24 입니다). C 환경에서는 풀 워드 요소로 액세스해야 합니다.

Gate1[i].Chan[j].MinusLimit 게이트 1 의 MLIM 플래그 입력

설명: MLIM 플래그 입력값

범위: 0 .. 1

단위: 논리값

Gate1[i].Chan[j].MinusLimit 에는 최신 서보 인터럽트에서 래치된 IC 채널의 MLIM(음의 오버트래블 리미트) 플래그 입력의 현재 논리 상태를 저장합니다. 1 의 값은 IC 에 대한 high 전압에 대응합니다. 0 의 값은 IC 에 대한 low 전압에 대응합니다. 대부분의 Delta Tau 하드웨어에서는 이는 광절연 입력입니다. 그리고 플래그가 "단함"일 때 광 아이솔레이터에 전류가 흐르고 0 이 됩니다.

Gate1[i].Chan[j].MinusLimit 는 스크립트 환경 풀 워드 요소 **Gate1[i].Chan[j].Status** 의 비트 18 입니다(C 에서는 비트 26 입니다). C 환경에서는 풀 워드 요소로 액세스해야 합니다.

Gate1[i].Chan[j].PhaseCapt 게이트 1 의 위상 인터럽트 엔코더 카운트값

설명: 위상 인터럽트에서 래치된 엔코더 카운트값

범위: 0 .. 16777215

단위: 엔코더 카운트

Gate1[i].Chan[j].PhaseCapt에는 최신의 위상 인터럽트에서 래치된 IC 채널의 하드웨어 엔코더 카운트값을 저장합니다. 이는 **Gate1[i].Chan[j].EncCtrl**로 지정한 엔코더 디코드에서 결정되는 엔코더 카운트 단위를 통한 24 비트값입니다. 카운터는 전원 투입/리셋으로 0으로 설정되고 그때부터 카운트됩니다. 따라서 항상 전원 투입/리셋 위치가 기준이 됩니다. 이는 모든 방향으로 롤 오버할 수 있습니다. 순간적인 카운터값이 프로세서로 액세스할 수 없으므로 이 요소는 현재 카운트값의 가장 다이렉트하게 액세스 가능한 대체값을 저장합니다.

만일 C 프로그램에서 액세스되는 경우에 이 요소는 상위 24 비트에 실제 데이터를 배치한 32 비트값입니다.

Gate1[i].Chan[j].PlusLimit 게이트 1 의 PLIM 플래그 입력

설명: PLIM 플래그 입력값

범위: 0 .. 1

단위: 논리값

Gate1[i].Chan[j].PlusLimit에는 최신 서보 인터럽트에서 래치된 IC 채널의 PLIM(양의 오버트래블 리미트) 플래그 입력의 현재 논리 상태를 저장합니다. 1의 값은 IC에 대한 high 전압에 대응합니다. 0의 값은 IC에 대한 low 전압에 대응합니다. 대부분의 Delta Tau 하드웨어에서는 이는 광절연 입력입니다. 그리고 플래그가 "닫힘"일 때 광 아이솔레이터에 전류가 흐르고 0이 됩니다.

Gate1[i].Chan[j].PlusLimit는 스크립트 환경 풀 워드 요소 **Gate1[i].Chan[j].Status**의 비트 17입니다(C에서는 비트 25입니다). C 환경에서는 풀 워드 요소로 액세스해야 합니다.

Gate1[i].Chan[j].PosCapt 게이트 1 의 위치 포착 트리거 플래그

설명: 위치 포착 트리거 플래그

범위: 0 .. 3

단위: 비트 필드

Gate1[j].Chan[j].PosCapt에는 채널 2개의 트리거 플래그의 현재 상태를 저장합니다. 비트 1(값 2)은 채널의 "범용" 트리거 플래그 상태입니다. 이는 **Gate1[j].Chan[j].CaptCtrl** 과 **Gate1[j].Chan[j].CaptFlagSel** 로 지정한 트리거 조건이 발생한 경우에는 항상 설정됩니다. 비트 0(값 1)은 "게이트된 인덱스"의 트리거 플래그 상태입니다. 이는 (**Gate1[j].Chan[j].CaptCtrl** 비트 0 = 1 과 **Gate1[j].Chan[j].GatedIndexSel** = 1 로 설정한) 1AB 상 쪽에서 "게이트화"된 엔코더의 인덱스 채널을 포함한 트리거 조건이 발생한 경우에는 항상 설정됩니다. 이 트리거는 주로 정확한 카운트 수가 인덱스 펄스 폭 사이에서 카운트를 확립하기 위해 사용됩니다.

두 플래그는 트리거 순간에 포착된 엔코더 위치가 저장된 레지스터를 읽으면 자동으로 클리어됩니다(**Gate1[j].Chan[j].HomeCapt**). 양쪽의 플래그는 레벨 트리거로 엣지 트리거가 아닙니다. 따라서 플래그가 클리어될 때 선택된 입력이 계속 지정된 트리거 상태이면 즉시 다시 트리거됩니다.

Gate1[j].Chan[j].PosCapt 는 스크립트 환경 폴 워드 요소 **Gate1[j].Chan[j].Status** 의 비트 10 - 11 을 구성합니다(C에서는 비트 18 - 19 입니다). C 환경에서는 이는 폴 워드 요소로 액세스되어야 합니다.

Gate1[j].Chan[j].ServoCapt 게이트 1 의 서보 인터럽트 엔코더 카운트

설명: 서보 인터럽트에서 래치된 엔코더 카운트값

범위: 0 .. 16777215

단위: 2*엔코더 카운트

Gate1[j].Chan[j].ServoCapt에는 직전에 수신된 카운트 방향을 더한 최신 서보 인터럽트에서 래치된 IC 채널의 하드웨어 엔코더 카운트값을 저장합니다. 이는 24 비트값입니다. 비트 0에는 수신된 직전의 카운트 방향을 저장합니다(0 = 상향, 1 = 하향). 비트 1~23에는 **Gate1[j].Chan[j].EncCtrl**으로 지정한 디코드 방법에 의한 엔코더 카운트 단위로 래치된 카운트값을 저장합니다. 카운터는 전원 투입/리셋으로 0으로 설정되고 그때부터 카운트됩니다. 따라서 항상 전원 투입/리셋 위치가 기준이 됩니다. 이는 모든 방향으로 롤 오버할 수 있습니다.

C 프로그램에서 액세스되는 경우에 이 요소는 상위 24 비트에 실제 데이터를 배치한 32 비트값입니다.

Gate1[j].Chan[j].Status 게이트 1 의 에러 코드 상태 요소

설명: "에러 코드" 상태 요소

범위: \$0 .. \$FFFFFF(스크립트)
\$0 .. \$FFFFFFFF(C)

단위: 비트 필드

Gate1[i].Chan[j].Status 는 IC 채널의 하드웨어 상태 레지스터 성분을 표시하는 24 비트의 Status 워드입니다(C 프로그램 속에서 액세스될 때는 32 비트 워드의 상위 24 비트). 이것에는 많은 개별적인 Status 비트와 요소를 저장합니다. 굵은 글자로 표시되는 각각의 Status 비트와 요소는 스크립트 프로그램 및 명령으로 별도의 요소로 액세스할 수 있지만 C 프로그램 및 함수에서는 액세스할 수 없습니다.

워드 속의 Status 비트 목록은 다음 표와 같습니다.

스크립트 비트#	스크립트 16 진수값	C 비트 #	C 16 진수값	요소명: Gate1[i].Chan[j]. {요소}	설명
23	\$800000	31	\$80000000	T	T 플래그 입력값
20 - 22	\$700000	28 - 30	\$70000000	UVW	U, V, W 플래그 입력값
19	\$80000	27	\$8000000	UserFlag	사용자 플래그 입력값
18	\$40000	26	\$4000000	MinusLimit	MLIM 플래그 입력값
17	\$20000	25	\$2000000	PlusLimit	PLIM 플래그 입력값
16	\$10000	24	\$1000000	HomeFlag	원점 플래그 입력값
15	\$8000	23	\$800000	Fault	앰프 이상 플래그 입력값
12 - 14	\$7000	20 - 22	\$700000	ABC	엔코더 A, B 그리고 C 채널 입력값
10 - 11	\$C00	18 - 19	\$C0000	PosCapt	위치 포착 트리거 플래그
9	\$200	17	\$20000	Equ	위치 비교 출력 상태
8	\$100	16	\$10000	CountError	엔코더 디코드 카운트 에러
4 - 7	\$F0	12 - 15	\$F000	-	(추후 사용하기 위해 예약)
3	\$8	11	\$800	DemuxInvalid	무효 UVW 디멀티플렉싱
0 - 2	\$7	8 - 10	\$700	HallState	포착 홀 상태
-	-	0 - 7	\$FF	-	(여기에 존재하는 하드웨어는 없습니다.)

각 요소는 해당하는 개별 사항 항목에서 상세하게 설명합니다.

Gate1[i].Chan[j].T 게이트 1의 T 플래그 입력

설명: T 플래그 입력값

범위: 0 .. 1

단위: 논리값

Gate1[i].Chan[j].T에는 최신 서보 인터럽트에서 래치된 IC 채널의 T 플래그 입력의 현재 논리 상태를 저장합니다. 1의 값은 IC에 대한 high 전압에 대응합니다. 0의 값은 IC에 대한 low 전압에 대응합니다. 대부분의 Delta Tau 하드웨어에서 IC에 대한 외부 버퍼 입력은 비반전입니다. 따라서 이는 컨트롤러에 대한 전압과 일치합니다.

Gate1[i].Chan[j].T는 스크립트 환경 풀 워드 요소 **Gate1[i].Chan[j].Status**의 비트 23입니다(C에서는 비트 31입니다). C 환경에서는 풀 워드 요소로 액세스해야 합니다.

Gate1[i].Chan[j].TimeBetweenCts 게이트 1 의 엔코더 카운트 간 시간

설명: 직전의 2 개의 엔코더 카운트 간 시간

범위: 0 .. 16777215

단위: 2 * SCLK 사이클

Gate1[i].Chan[j].TimeBetweenCts 에는 보통 IC 채널의 최신 서보 인터럽트에서 래치된 마지막부터 2 개의 엔코더 카운트 간 시간과 이 2 개의 카운트가 같은지 반대 방향인지를 나타내는 플래그를 함께 저장합니다. 이는 24 비트값입니다. 비트 0 에는 방향 변화 플래그를 저장합니다(0 =변화 없음, 1 = 변화 있음). 비트 1~23 에는 엔코더 샘플 클럭(SCLK) 사이클 단위의 래치된 타이머값을 저장합니다. 해당 분주/주파수는 **Gate1[i].HardwareClockCtrl** 으로 지정됩니다. 이 타이머값은 일반적으로 소수점 카운트 위치를 추정하기 위해 기본 설정의 "소프트웨어 1/T 보간"에서 사용됩니다.

저장 설정 요소 **Gate1[i].Chan[j].EncCtrl** 이 "MLDT 펄스 타이밍"을 지정하는 12 로 설정된 경우에 이 요소에는 내부 117.9648MHz 클럭 신호 단위로 채널상의 마지막에 생성된 출력 펄스와 MLDT 를 통한 "에코 펄스" 수신 간의 시간을 저장합니다. 이 시간은 센서 종료 시부터의 가동 자석 MLDT 의 거리에 비례합니다.

저장 설정 요소 **Gate1[i].Chan[j].OneOverTEna** 가 1 에서 "하드웨어 1/T 보간"이 유효로 설정되는 경우에 이 24 비트 요소는 2 개의 12 비트 성분으로 분할됩니다. 비트 0~11 은 "비교 A" 위치의 소수 카운트 부분을 나타내는 쓰기 가능한 성분으로 그 정수 카운트 부분은 **Gate1[i].Chan[j].CompA** 에 있습니다. 비트 12~23 은 서보 포착 위치의 소수 카운트 부분으로 그 정수 카운트 부분은 **Gate1[i].Chan[j].ServoCapt** 에 있습니다. 양쪽의 소수점 성분은 1/4096 카운트 단위입니다.

C 프로그램에서 액세스되는 경우에 이 요소는 상위 24 비트에 실제 데이터를 배치한 32 비트값입니다.

Gate1[i].Chan[j].TimeSinceCts 게이트 1 의 엔코더 카운트 경과 시간

설명: 마지막 엔코더 카운트 경과 시간

범위: 0 .. 16777215

단위: SCLK 사이클

Gate1[i].Chan[j].TimeSinceCts 에는 보통 IC 채널의 최신 서보 인터럽트에서의 엔코더 카운트부터의 경과 시간을 저장합니다. 이는 엔코더 샘플 클럭(SCLK) 사이클 단위의 24 비트값으로 해당 분주/주파수는 **Gate1[i].HardwareClockCtrl** 로 지정됩니다. 이 타이머값은 일반적으로 소수 카운트 위치를 추정하기 위해 기본 설정의 "소프트웨어 1/T 보간"에서 사용됩니다.

저장 설정 요소 **Gate1[i].Chan[j].OneOverTEna** 가 1 에서 "하드웨어 1/T 보간"이 유효로 설정되는 경우에 이 24비트 요소는 2 개의 12 비트 성분으로 분할됩니다. 비트 0~11 은 "비교 B" 위치의 소수 카운트 부분을 나타내는 쓰기 가능한 성분으로 그 정수 카운트 부분은 **Gate1[i].Chan[j].CompB** 에 있습니다. 비트 12~23 은 서보 포착 위치의 소수 카운트 부분으로 그 정수 카운트 부분은 **Gate1[i].Chan[j].HomeCapt** 에 있습니다. 양쪽의 소수점 성분은 1/4096 카운트 단위입니다.

C 프로그램에서 액세스되는 경우에 이 요소는 상위 24 비트에 실제 데이터를 배치한 32 비트값입니다.

Gate1[i].Chan[j].UserFlag 게이트 1 의 사용자 플래그 입력

설명: 사용자 플래그 입력값

범위: 0 .. 1

단위: 논리값

Gate1[i].Chan[j].UserFlag 에는 최신 서보 인터럽트에서 래치된 IC 채널의 USER 플래그 입력의 현재 논리 상태를 저장합니다. 1 의 값은 IC 에 대한 high 전압에 대응합니다. 0 의 값은 IC 에 대한 low 전압에 대응합니다. 대부분의 Delta Tau 하드웨어에서는 이는 광절연 입력입니다. 그리고 플래그가 "단함"일 때 광 아이솔레이터에 전류가 흐르고 0 이 됩니다.

Gate1[i].Chan[j].UserFlag 는 스크립트 환경 풀 워드 요소 **Gate1[i].Chan[j].Status** 의 비트 19 입니다(C 에서는 비트 27 입니다). C 환경에서는 풀 워드 요소로 액세스해야 합니다.

Gate1[i].Chan[j].UVW 게이트 1 의 UVW 플래그 입력

설명: U, V 그리고 W 플래그 입력값

범위: 0 .. 7

단위: 비트 필드

Gate1[i].Chan[j].UVW 에는 최신 서보 인터럽트에서 IC 채널의 U, V 및 W 플래그 입력의 현재 논리 상태를 저장합니다. 비트 2(값 4)는 U 플래그의 상태를 나타냅니다. 비트 1(값 2)은 V 플래그의 상태를 나타냅니다. 그리고 비트 0(값 1)은 W 플래그의 값을 나타냅니다.

비트의 1 의 값은 IC 에 대한 high 전압 입력을 나타냅니다. 0 의 값은 IC 에 대한 low 전압에 나타냅니다. 대부분의 Delta Tau 하드웨어에서 IC 에 대한 외부 버퍼 입력은 비반전입니다. 따라서 이는 컨트롤러에 대한 전압과 일치합니다.

Gate1[i].Chan[j].UVW 는 스크립트 환경 풀 워드 요소 **Gate1[i].Chan[j].Status** 의 비트 20 - 22 를 구성합니다(C 에서는 비트 28 - 30 입니다). C 환경에서는 풀 워드 요소로 액세스해야 합니다.

Gate3[i].(PMAC3 스타일 ASIC)상태 요소(V1.6 2014.03)

PMAC3 스타일 DSPGATE3 머신 인터페이스 ASIC에서는 범용 I/O 및 MACRO 링 인터페이스를 추가한 4 개 채널의 서보 인터페이스를 갖추고 있습니다. 본 장의 상태 요소에서는 프로세서가 이 ASIC의 외부 입력으로 얻어진 값에 액세스할 수 있습니다.

Gate3[i].멀티채널 상태 요소

본 장의 요소는 채널에 개별이 아니며 따라서 **Chan[j]** 서브 구조체를 필요로 하지 않습니다.

Gate3[i].ChipID 게이트 3의 IC 식별 번호

설명: DSPGATE3 IC 식별 번호

범위: \$0..\$FFFFFFFF

단위: 없음

Gate3[i].ChipID에는 IC의 식별 번호값을 저장합니다. DSPGATE3 IC에서는 이 값은 \$7145300n으로 n은 개정 번호입니다. 이 요소는 프로세서에서 액세스 가능하고 IC의 존재와 개정을 확인할 수 있습니다.

Gate3[i].EepromData[k] 게이트 3의 EEPROM 데이터 워드

설명: EEPROM 데이터 워드

범위: \$0..\$FFFFFFFF

단위: 없음

Gate3[i].EepromData[k]에는 DSPGATE3 IC에 연결된 128 비트 EEPROM IC에서 수신된 정보인 4개의 32 비트 워드 중의 1개를 저장합니다. 인덱스 *k*는 0부터 3까지의 값입니다. DSPGATE3 IC에는 8개까지의 I²C EEPROM IC를 연결할 수 있습니다. 그중 한 개는 **Gate3[i].EepromCtrl** 요소를 사용하여 주소가 지정됩니다. **Gate3[i].EepromData[k]** 레지스터 값에서는 유효한 EEPROM IC의 데이터라는 것은 알 수 없습니다.

전원 투입/리셋에서 Power PMAC은 ASIC에 연결 가능한 EEPROM IC의 8개 모든 읽기를 시도하기 위해 DSPGATE3 IC의 EEPROM 인터페이스를 사용합니다. 이러한 EEPROM에는 ASIC가 사용되는 하드웨어 구성 데이터를 저장합니다. 데이터는 상태 요소 **Gate3[i].PartNum**, **Gate3[i].PartOpt0**에서 **Gate3[i].PartOpt7**, **Gate3[i].PartRev** 및 **Gate3[i].PartType**의 값을 설정하기 위해 사용됩니다.

Gate3[j].GpioOutData[j] 게이트 3의 범용 I/O뱅크 출력값

설명: IC의 범용 I/O뱅크 출력 데이터 유지값

범위: \$0..\$FFFFFFFF

단위: 비트 필드

전원 투입 기본값: \$0

Gate3[j].GpioOutData[j]는 IC의 I/O뱅크 "j"의 32 I/O 포인트 명령 출력 상태를 나타내는 32 비트값입니다. 이는 메모리 내의 실제 출력용 유지 레지스터입니다. 스크립트 환경에서는 명령이 실제 I/O뱅크 데이터 레지스터 **Gate3[j].GpioData[j]**에 쓸 때 Power PMAC은 이 유지 레지스터에도 같은 값을 복사합니다.

I/O뱅크 데이터 레지스터 자체의 읽기가 출력 상태를 올바르게 반영하지 않는 I/O bank pins 설정이 될 가능성이 있으므로 **Gate3[j].GpioOutData[j]**는 유용합니다. 따라서 이 유지값은 출력값을 모니터링할 때 그리고 레지스터의 특정 비트를 세트 또는 클리어하는 경우의 초기값 레지스터로 양쪽에 사용하는 것이 좋습니다. Power PMAC은 IC의 실제 I/O 레지스터 특정 비트 **Gate3[j].GpioData[j].k**를 세트할 때 이 동작을 실시합니다.

Gate3[j].MacroInA[j][k] 게이트 3의 MACRO뱅크 A입력 데이터

설명: MACRO뱅크 A노드 j 레지스터 k입력 데이터

범위: $-2^{31}..2^{31}-1$

단위: 사용자 정의

Gate3[j].MacroInA[j][k]는 IC의 MACRO뱅크 A노드 j입력 데이터 레지스터 k입니다. 데이터 레지스터 인덱스 k의 범위는 0~3입니다. 노드 인덱스 j의 범위는 0~15입니다. 이 뱅크는 저장 설정 요소 **Gate3[j].MacroEnableA**의 비트 28 - 31에서 설정되는 MACRO링 상의 마스터 번호입니다. 노드 j의 4개 레지스터 모든 입력 데이터는 **Gate3[j].MacroEnableA**의 비트(j+8)가 1로 설정되는 경우에 자동으로 수신되고 위상 사이클마다 래치됩니다.

오리지널 MACRO 프로토콜에서는 **Gate3[j].MacroInA[j][0]**의 상위 24비트만 링 상에서 수신되고 **Gate3[j].MacroInA[j][1]**, **Gate3[j].MacroInA[j][2]** 및 **Gate3[j].MacroInA[j][3]**의 상위 16비트만 링 상에서 수신됩니다. 새로운 MACRO2 프로토콜에서는 4개 노드 레지스터의 모든 32비트 데이터가 링 상에서 수신됩니다.

노드가 자동 서보 제어에 사용될 때 저장 설정 요소 **EncTable[j].pEnc**는 보통 **Gate3[j].MacroInA[j][0].a**로 설정됩니다. 또한 **Motor[x].pEncStatus**(나아가 다른 관련된 주소 변수)는 보통 **Gate3[j].MacroInA[j][3].a**로 설정됩니다. 이 경우에 자동으로 Power PMAC 태스크는 이러한 레지스터에서 읽고 일반적으로 사용자·어플리케이션 코드는 레지스터에서 읽지 않아도 됩니다.

DSPGATE3 IC에서 이는 **Gate3[j].MacroOutA**의 출력값에 대응하는 레지스터를 별도로 주소로 지정되고 해당 요소에 쓸 수는 없습니다. 이전 DSPGATE2 IC에서는 별도의 출력 및 입력값은 레지스터를 공유합니다.

Gate3[i].MacroInA[j][k].{start}.{width} 구문을 사용하여 **Gate3[i].MacroInA[j][k]** 비트의 특정 범위값을 참조할 수 있습니다. 여기서 **{start}**는 범위 내의 하위 비트 번호이고 **{width}**는 범위 내의 비트 수입니다. 또한 이와 같이 "start"와 "width"값을 사용하여 요소 내의 비트 범위에 대한 M-변수를 정의할 수 있습니다. 예는 다음과 같습니다.

ptr OverPressure->Gate3[0].MacroInB[2][0].12.1

Gate3[i].MacroInB[j][k] 게이트 3의 MACRO 뱅크 B 입력 데이터

설명: MACRO 뱅크 B 노드 *j* 레지스터 *k* 입력 데이터

범위: $-2^{31}..2^{31}-1$

단위: 사용자 정의

Gate3[i].MacroInB[j][k]는 IC의 MACRO 뱅크 B의 노드 *j* 입력 데이터 레지스터 *k*입니다. 데이터 레지스터 인덱스 *k*의 범위는 0~3입니다. 노드 인덱스 *j*의 범위는 0~15입니다. 이 뱅크는 저장 설정 요소 **Gate3[i].MacroEnableB**의 비트 28 - 31에서 설정되는 MACRO 링 상의 마스터 번호입니다. 노드 *j*의 4개 레지스터의 모든 입력 데이터는 **Gate3[i].MacroEnableB**의 비트(*j* + 8)가 1로 설정되는 경우에 자동으로 수신되고 위상 사이클마다 래치됩니다.

오리지널 MACRO 프로토콜에서는 **Gate3[i].MacroInB[j][0]**의 상위 24비트만 링 상에서 수신되고 또한 **Gate3[i].MacroInB[j][1]**, **Gate3[i].MacroInB[j][2]** 및 **Gate3[i].MacroInB[j][3]**의 상위 16비트만 링 상에서 수신됩니다. 새로운 MACRO2 프로토콜에서는 4개 노드 레지스터의 모든 32비트 데이터가 링 상에서 수신됩니다.

노드가 자동 서보 제어에 사용될 때 저장 설정 요소 **EncTable[i].pEnc**는 보통 **Gate3[i].MacroInB[j][3].a**로 설정됩니다. 또한 **Motor[x].pEncStatus**(또한 다른 관련된 주소 변수)는 보통 **Gate3[i].MacroInB[j][0].a**로 설정됩니다. 이 경우에 자동으로 Power PMAC 태스크는 이러한 레지스터에서 읽고 일반적으로 사용자-어플리케이션 코드는 레지스터에서 읽지 않아도 됩니다.

DSPGATE3 IC에서는 이는 **Gate3[i].MacroOutB**의 출력값에 대응하는 레지스터를 별도로 주소로 지정되고 그래서 이러한 요소에 쓸 수 없습니다. 이전 DSPGATE2 IC에서는 별도의 출력 및 입력값은 레지스터를 공유합니다.

Gate3[i].MacroInB[j][k].{start}.{width} 구문을 사용하여 **Gate3[i].MacroInB[j][k]** 비트의 특정 범위값을 참조할 수 있습니다. 여기서 **{start}**가 범위 내의 하위 비트 번호로 **{width}**는 범위 내의 비트 수입니다. 또한 이와 같이 "start"와 "width"값을 사용하여 요소 내의 비트 범위에 대한 M-변수를 정의할 수 있습니다. 예는 다음과 같습니다.

ptr OverPressure->Gate3[0].MacroInB[2][0].12.1

Gate3[i].MacroError 게이트 3의 MACRO 노드 에러

설명: MACRO 노드 에러 Status

범위: \$0..\$FFFFFFFF

단위: 비트 필드

만일 있다면 **Gate3[i].MacroError**에는 노드가 마지막 위상 사이클에서 MACRO 링 위에 수신된 데이터 패킷에서 발생한 에러 정보를 저장합니다. 이는 32비트값으로 1비트가 IC에서 지원되는 MACRO 노드를 각각 나타냅니다. 마지막 사이클에서 데이터 수신 에러가 없었던 경우에 비트는 0이 됩니다. 마지막 사이클에서 데이터 수신 에러가 있었던 경우에 1이 됩니다. 노드가 유효하지 않은 경우에 비트는 항상 0입니다.

비트 0~15는 IC의 MACRO 뱅크 A의 노드 0~15의 에러 Status를 각각 나타냅니다. 비트 16~31은 IC의 MACRO 뱅크 B의 노드 0~15의 에러 Status를 각각 나타냅니다.

Gate3[i].MacroError에서는 단지 MACRO 데이터 에러의 유무를 나타냅니다. 만약 있다면 에러 타입은 저장 설정 요소 **Gate3[i].MacroModeA** 및 **Gate3[i].MacroModeB** Status 부분으로 검출할 수 있습니다.

Gate3[i].PartNum 게이트 3의 하드웨어 부품 번호

설명: DSPGATE3 IC 보드의 하드웨어 부품 번호

범위: 음이 아닌 정수

단위: 열거 형식

Gate3[i].PartNum에는 DSPGATE3 ASIC를 내장한 보드의 6자리 부품 번호를 저장합니다. 이는 ASIC에 연결된 EEPROM IC에서 추출되어 전원 투입/리셋으로 자동으로 읽힙니다. 이 인덱스의 ASIC가 존재하지 않는 경우에 0이 통지됩니다.

현재 기존의 부품 번호는 다음과 같습니다.

- ACC-24E3 604002
- ACC-5E3 604017
- ACC-5EP3 604035
- ACC-59E3 604027
- Power Brick 604030
- PowerClipper 604050

이 요소는 실제 ASIC의 하드웨어 레지스터가 아닙니다. 이는 ASIC와 관련한 소프트웨어 레지스터입니다. 이는 IDE에서 주로 시스템 설정 유틸리티로 사용됩니다.

C에서는 이 요소는 **Gate3PartData[i].Num**으로 액세스합니다.

Gate3[i].PartOptn 게이트 3의 옵션 하드웨어 구성

설명:	DSPGATE3 IC 보드의 옵션 하드웨어 구성 코드
범위:	음이 아닌 정수
단위:	열거 형식

Gate3[i].PartOptn에는 어느 하드웨어가 DSPGATE3 IC에 연결되어 있는지를 나타내는 코드를 저장합니다. 이는 ASIC에 연결된 EEPROM IC에서 추출되어 전원 투입/리셋으로 자동으로 읽힙니다. 마지막 숫자 *n*은 정보를 읽은 특정 EEPROM 번호와 일치하는 0부터 7의 값을 취합니다. 각 *n* 값의 EEPROM 장소는 다음과 같습니다.

- 0: 베이스 보드
- 1: 최초의 2 채널 1 세트의 피드백 인터페이스
- 2: 최초의 2 개 채널 1 세트의 출력 인터페이스
- 3: (추후 사용하기 위해 예약)
- 4: 2 번째 2 개 채널 (별도로 있으면) 1 세트의 중심 회로
- 5: 2 번째 2 개 채널 (별도로 있으면) 1 세트의 피드백 인터페이스
- 6: 2 번째 2 개 채널 (별도로 있으면) 1 세트의 출력 인터페이스
- 7: (추후 사용하기 위해 예약)

일반적으로 **Gate3[i].PartOptn**의 값은 가격 목록 부품 번호 내의 옵션용 제품 코드와 같습니다. 이 인덱스를 가진 ASIC가 존재하지 않는 경우에 "nan"(not-a-number)으로 통지합니다.

이 요소는 실제 ASIC의 하드웨어 레지스터가 아닙니다. 이는 ASIC와 관련한 소프트웨어 레지스터입니다. 이는 IDE에서 주로 시스템 설정 유틸리티로 사용됩니다.

C에서는 이 요소는 **Gate3PartData[i].Optn**으로 액세스합니다.

Gate3[i].PartRev 게이트 3의 하드웨어 개정 번호

설명:	DSPGATE3 IC 보드의 하드웨어 개정 번호
범위:	음이 아닌 정수
단위:	열거 형식

Gate3[i].PartRev에는 DSPGATE3 ASIC를 내장한 보드의 개정 번호를 저장합니다. 이는 ASIC에 연결된 EEPROM IC에서 추출되어 전원 투입/리셋으로 자동으로 읽힙니다. 이 번호는 보드의 3 자리 부품 개정 번호의 마지막 자리에 해당합니다.

이 요소는 실제 ASIC의 하드웨어 레지스터가 아닙니다. 이는 ASIC와 관련한 소프트웨어 레지스터입니다. 이는 IDE에서 주로 시스템 설정 유틸리티로 사용됩니다.

C에서는 이 요소는 **Gate3PartData[i].Rev**로 액세스합니다.

Gate3[i].PartType

설명: DSPGATE3 IC 보드의 하드웨어 부품 타입

범위: 음이 아닌 정수

단위: 비트 필드

Gate3[i].PartType 은 DSPGATE3 ASIC 를 내장한 보드의 하드웨어 인터페이스 타입을 나타내는 정수를 저장합니다. 이는 ASIC 에 연결된 EEPROM IC 에서 추출되어 전원 투입/리셋으로 자동으로 읽힙니다. 이 인덱스를 가진 ASIC 가 존재하지 않는 경우에 "nan"(not-a-number)으로 통지합니다.

각 비트는 존재하는 인터페이스 타입 중 1 개를 지정합니다. IC 에 그 인터페이스 타입이 존재하는 경우에 비트가 세트됩니다. 여러 인터페이스 타입이 존재할 수 있습니다.

- 비트 0(값 1): 서보
- 비트 1(값 2): MACRO
- 비트 2(값 4): 아날로그 I/O
- 비트 3(값 8): 디지털 입출력

이 요소는 실제 ASIC 의 하드웨어 레지스터가 아닙니다. 이는 ASIC 와 관련한 소프트웨어 레지스터입니다. 이는 IDE 에서 주로 시스템 설정 유틸리티로 사용됩니다.

C 에서는 이 요소는 **Gate3PartData[i].Type** 으로 액세스합니다.

Gate3[j].채널 고유 상태 요소

ASIC 내의 각 서보 인터페이스 채널에는 채널에 대한 입력 상태에 관한 정보를 내장한 몇 개의 읽기 전용 레지스터가 있습니다. 채널 인덱스 j 는 0~3 의 범위에서 하드웨어 채널 No.1~4 에 각각 대응합니다.

Gate3[j].Chan[j].ABC 게이트 3 채널의 엔코더 ABC 채널 입력값

설명: A, B 와 C 엔코더 채널 입력값

범위: 0 .. 7

단위: 비트 필드

Gate3[j].Chan[j].ABC 에는 최신의 서보 인터럽트에서 래치된 IC 채널에 대한 A, B 및 C 엔코더 입력의 현재 논리 상태를 저장합니다. 비트 2(값 4)는 C(인덱스) 채널의 상태를 나타냅니다. 비트 1(값 2)은 B 채널의 상태를 나타냅니다. 그리고 비트 0(값 1)은 A 채널의 값을 나타냅니다.

비트가 1 인 값은 IC 상의 high 전압에 대응합니다. 0 인 값은 IC 에 대한 low 전압에 대응합니다. 대부분의 Delta Tau 하드웨어에서 이는 차동 입력 회로로 그 경우에는 "+" 입력이 "-" 입력보다 높은 전압일 때 1 의 값이 됩니다.

Gate3[j].Chan[j].ABC 에서는 풀 워드 요소 **Gate3[j].Chan[j].Status** 의 비트 4 - 6 을 구성합니다. C 언어 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[j].Chan[j].ABPins 게이트 3 채널의 AB 신호 레벨

설명: 엔코더 A+, A-, B+, B- 신호 레벨

범위: 0 .. 15

단위: 비트 필드

Gate3[j].Chan[j].ABPins 에는 최신의 위상 인터럽트에서 래치된 IC 채널에 대한 A+, A-, B+ 및 B- 엔코더 입력의 현재 논리 상태를 저장합니다. 비트 3(값 8)은 B+ 신호의 상태를 나타내며 비트 2(값 4)는 B- 신호의 상태를 나타냅니다. 그리고 비트 1(값 2)은 A+ 신호의 상태를 나타내며 비트 0(값 1)은 A- 신호의 값을 나타냅니다.

비트가 1 인 값은 IC 상의 high 전압에 대응합니다. 0 인 값은 IC 에 대한 low 전압에 대응합니다. 대부분의 Delta Tau 하드웨어에서는 이는 차동 라인 리시버 앞에서 전송된 엔코더 차동 라인 드라이버 페어의 한쪽 신호입니다. 이러한 신호는 "엔코더 소실" 검출에 사용되는 ASIC 의 배타적 논리합 게이트에 대한 입력입니다.

Gate3[j].Chan[j].ABPins 는 풀 워드 요소 **Gate3[j].Chan[j].Status** 의 비트 0 - 3 을 구성합니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[i].Chan[j].AdcAmp[k] 게이트 3 채널의 앰프 ADC 값

설명: 앰프 아날로그/디지털 변환값

범위: $-2^{31}..2^{31}-1$

단위: 32 비트 ADC LSB

Gate3[i].Chan[j].AdcAmp[k]에는 최신의 위상 클록 사이클에서 래치된 IC 채널의 위상 및 하프 클록 사이클마다 "앰프용" 아날로그/디지털 변환(ADC)값을 저장합니다. 각 IC 채널에는 4 개의 **AdcAmp** 레지스터가 있습니다. A 위상용이 2 개이고 B 위상용이 2 개입니다. 각각의 모터/앰프 위상에서 1 개는 위상 클록 상승 엣지로 스트로브되고 1 개는 하강 엣지로 스트로브됩니다. 4 개의 요소와 그 의미는 다음과 같습니다.

- **AdcAmp[0]** A 위상, 위상 클록의 상승 엣지로 스트로브
- **AdcAmp[1]** B 위상, 위상 클록의 상승 엣지로 스트로브
- **AdcAmp[2]** A 위상, 위상 클록의 하강 엣지로 스트로브
- **AdcAmp[3]** B 위상, 위상 클록의 하강 엣지로 스트로브

요소 **AdcAmp[0]** 및 **AdcAmp[1]**은 최신의 유효값(1/2 위상 사이클 전)이 되므로 Power PMAC "다이렉트 PWM" 알고리즘에서 디지털 전류 클로즈드 루프용 모터의 전류 피드백에 보통 사용됩니다. 외부의 ADC 회로가 위상 클록 신호 상태에 따라 아날로그 입력을 다중화할 수 있는 경우에 **AdcAmp[2]** 및 **AdcAmp[3]** 요소는 버스 전압 및 온도와 같은 완전히 다른 값에 이용할 수 있습니다.

이는 최고 24 비트의 ADC 를 지원하는 회로를 통한 32 비트 요소입니다. 인터페이스가 올바르게 설정된 경우에 n 비트 ADC 의 데이터는 이 32 비트 요소의 상위 n 비트에 배치됩니다. Power PMAC 의 "다이렉트 PWM" 전류(commutation) 알고리즘 중에서 전류 피드백에 사용될 때 트루 데이터는 최상위 비트에 존재해야 합니다.

ADC 가 트루 데이터에 선행하는 "헤더 비트"를 가지는 경우에 저장 설정 요소 **Gate1[i].AdcAmpHeaderBits** 의 적절한 설정이 헤더 데이터를 이 요소의 하위 비트에 "롤 오버"시킵니다.

Gate3[i].Chan[j].AdcEnc[k] 게이트 3 채널의 엔코더 ADC 값

설명: 엔코더 아날로그/디지털 변환값

범위: $-2^{31}..2^{31}-1$

단위: 32 비트 ADC LSB

Gate3[i].Chan[j].AdcEnc[k]에는 최신의 위상 클록 사이클에서 래치된 IC 채널의 위상 및 하프 클록 사이클마다 "엔코더용" 아날로그/디지털 변환(ADC)값을 저장합니다. 각 IC 채널에는 4 개 **AdcEnc** 레지스터가 있습니다. A 위상용이 2 개이고 B 위상용이 2 개입니다. 각 엔코더 위상에서 1 개는 위상 클록의 상승 엣지로 스트로브되고 1 개는 하강 엣지로 스트로브됩니다. 4 개의 요소와 그 의미는 다음과 같습니다.

- **AdcEnc[0]** A 위상, 위상 클록의 상승 엣지로 스트로브
- **AdcEnc[1]** B 위상, 위상 클록의 상승 엣지로 스트로브
- **AdcEnc[2]** A 위상, 위상 클록의 하강 엣지로 스트로브
- **AdcEnc[3]** B 위상, 위상 클록의 하강 엣지로 스트로브

요소 **AdcEnc[0]** 및 **AdcEnc[1]**은 각각 최신 유효값(1/2 위상 사이클 전)이므로 사인파 엔코더 및 리졸버의 "사인" 및 "코사인"에 사용됩니다. 외부의 ADC 회로가 위상 클록 신호 상태에 따라 아날로그 입력을 다중화할 수 있는 경우에 **AdcAmp[2]** 및 **AdcAmp[3]** 요소는 사인파 엔코더 상의 아날로그 전류(commutation) 트랙 또는 감속 기어가 장착된 "멀티 턴" 리졸버와 같은 다른 값에 이용할 수 있습니다.

Gate3[j].Chan[j].AdcEnc[0]과 **Gate3[j].Chan[j].AdcEnc[1]**의 값은 **Gate3[j].Chan[j].Atan** 과 **Gate3[j].Chan[j].SumOfSquares** 생성에 자동으로 사용되고 저장 설정 요소 **Gate3[j].Chan[j].AtanEna** 가 1 인 경우에 **Gate3[j].Chan[j].PhaseCapt** 와 **Gate3[j].Chan[j].ServoCapt** 에서 서브 사이클 위치 정보에 자동으로 사용됩니다.

이는 32bit 요소로 최고 24 비트의 ADC 를 지원합니다. 인터페이스가 올바르게 설정되면 n 비트 ADC 의 데이터는 이 32 비트 요소의 상위 n 비트에 배치됩니다. 사인파 엔코더 또는 리졸버에 따른 위치 피드백에 사용될 때 트루 데이터는 최상위 비트에 존재해야 합니다.

ADC 데이터 앞에 "헤더 비트"가 있는 경우에 저장 설정 요소 **Gate1[j].AdcEncHeaderBits** 를 적절하게 설정하여 헤더 데이터를 이 요소의 하위 비트에 "롤 오버"시킵니다.

Gate3[j].Chan[j].Atan 게이트 3 채널의 엔코더 사인/코사인 각도

설명: 엔코더의 사인/코사인 각도

범위: 0 ..65,535

단위: 1/65,536 사이클

Gate3[j].Chan[j].Atan 에는 주 엔코더의 ADC 값에서 ASIC 로 자동 산출된 각도값을 저장합니다. 이러한 값은 **Gate3[j].Chan[j].AdcEnc[0]**(사인) 및 **Gate3[j].Chan[j].AdcEnc[1]**(코사인)에 있습니다. 이러한 값은 위상 클록의 상승 엣지로 스트로브되는 A/D 변환기에서 얻으며 위상 클록의 하강 엣지(프로세서의 인터럽트)로 데이터가 사용 가능해집니다. **Atan** 값은 위상 인터럽트에서 즉시 사용 가능합니다(위상 사이클의 절반에 25ADC 클록 사이클과 2μsec 의 충분한 시간이 있는 경우).

아크탄젠트값을 산출하기 전에 저장 설정 요소 **Gate3[j].Chan[j].AdcOffset[0]**과 **Gate3[j].Chan[j].AdcOffset[1]**의 값을 각각의 입력에 더합니다. (해당 항은 어플리케이션이 아날로그 바이어스를 보정 할 수 있습니다.) 이것이 완전한 360 도 범위가 되는 결과를 부여하는 "2 인수" ATAN 계산(많은 경우 ATAN2 라고 불림)이라는 점에 주의하십시오. 이는 변환기의 실제 분해능에 관계없이 ADC 입력 레지스터의 상위 16 비트를 이용합니다. 의미가 있는 정보가 되는 이 요소의 비트 수는 변환기의 분해능 및 아날로그 회로의 잡음 레벨에 좌우됩니다. 이 값이 서보 피드백에 사용될 때 엔코더 변환 테이블 상의 소프트웨어 추적 필터가 잡음 레벨을 낮추기 위해 자주 이용됩니다.

리졸버-디지털 변환 처리에서 ASIC 를 사용할 때 **Gate3[i].Chan[j].Atan** 은 리졸버 위치로 일반적으로 사용됩니다.

저장 설정 요소 **Gate3[i].Chan[j].AtanEna** 가 1 인 경우에 이 16 비트값의 상위 14 비트는 **Gate3[i].Chan[j].PhaseCapt** 및 **Gate3[i].Chan[j].ServoCapt** 의 하위 14 비트에 동일하게 복사됩니다. 여기서는 이는 사인파 엔코더에 대한 소수 사이클값을 나타냅니다. **AtanEna** 가 1 인 경우에 **Atan** 에는 정수 카운트 데이터와 결합 처리되는 데이터의 소수 부분을 포함하며 이는 원시 아크탄젠트 데이터와 방향이 다를 수 있습니다.

Gate3[i].Chan[j].Atan 는 풀 워드 요소 **Gate3[i].Chan[j].AtanSumOfSqr** 의 비트 16 - 31 을 구성합니다. C 프로그램과 함수는 풀 워드 요소를 사용해야 합니다.

Gate3[i].Chan[j].AtanSumOfSqr 게이트 3 채널의 엔코더 사인/코사인 각도와 진폭

설명: 엔코더 사인/코사인 각도 및 진폭 레지스터

범위: 32 비트의 정수

단위: (특정 구성 요소 참조)

Gate3[i].Chan[j].AtanSumOfSqr 은 32 비트의 읽기 전용 요소로 채널의 주 엔코더 사인, 코사인 ADC 값에서 자동으로 계산되는 정보를 가진 2 개의 16 비트 요소를 포함합니다.

Gate3[i].Chan[j].Atan(비트 16 - 31)에는 2 개 값의 아크탄젠트에서 산출된 각도를 저장합니다. **Gate3[i].Chan[j].SumOfSquares**(비트 0 - 15)에는 입력값의 제곱합으로 산출된 신호의 "진폭"을 저장합니다.

스크립트 프로그램과 명령은 보통 부분 워드 요소를 사용합니다. 그러나 C 프로그램과 함수는 이 풀 워드 요소를 사용하여 특정 구성 요소를 분리하는 데 필요할 때에 마스크하고 시프트해야 합니다.

특정 구성 요소에 대한 자세한 내용은 해당 부분 워드 요소의 설명을 참조하십시오.

Gate3[i].Chan[j].CountError 게이트 3 채널의 엔코더 디코드 카운트 에러

설명: 엔코더 디코드 카운트 에러

범위: 0 ..1

단위: 논리값

Gate3[i].Chan[j].CountError 에는 IC 채널의 "엔코더 카운트 에러" 상태 플래그의 현재 논리 상태를 저장합니다. 1 의 값에서는 같은 엔코더 샘플 클럭(SCLK) 사이클에서 엔코더의 A, B 채널 양쪽 입력 상태가 변화하는 무효의 AB 상 이동이 발생했다는 것을 나타냅니다. 이 무효 이동이 발생할 때 이 플래그는 설정되고 다음에 계속되는 어떤 엔코더 카운터의 읽기 래치된 값은 잘못된 값이라는 것을 나타냅니다.

Gate3[j].Chan[j].CountError 는 Power PMAC 의 전원 투입/리셋으로 강제로 0 으로 설정됩니다. 어플리케이션에서는 비트에 0 을 쓰면 이 Status 비트를 클리어할 수 있습니다.

Gate3[j].Chan[j].CountError 는 풀 워드 요소 **Gate3[j].Chan[j].Status** 의 비트 30 입니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[j].Chan[j].DemuxInvalid 게이트 3 채널의 UVW 다중화 분리 무효

설명: AB 상에 의한 UVW 의 다중 분리 무효

범위: 0 ..1

단위: 논리값

Gate3[j].Chan[j].DemuxInvalid 에는 IC 채널의 "홀 다중 분리 무효" 플래그의 현재 논리 상태를 저장합니다. 저장 설정 요소 **Gate3[j].Chan[j].IndexDemuxEna** 가 1 인 경우에 IC 는 엔코더 "C" 신호와 AB 상의 4 상태에 따른 인덱스 펄스와 AB 상에서 다중화 분리하여 U, V 및 W 플래그 상태를 추출합니다. 전원 투입/리셋 후에 이 회로가 AB 상의 4 상태를 모두 조사하기 전에 **Gate3[j].Chan[j].DemuxInvalid** 는 이러한 플래그 상태의 완전히 유효한 1 세트를 제공할 수 없다는 것을 나타내는 1 로 설정됩니다. 일단 그것이 AB 상의 4 상태를 모두 조사했다면 **Gate3[j].Chan[j].DemuxInvalid** 는 0 으로 설정됩니다.

Gate3[j].Chan[j].DemuxInvalid 는 풀 워드 요소 **Gate3[j].Chan[j].Status** 의 비트 19 입니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[j].Chan[j].Equ 게이트 3 채널의 위치 비교 내부 상태

설명: 위치 비교 내부 상태

범위: 0 ..1

단위: 논리값

Gate3[j].Chan[j].Equ 에는 IC "EQU" 위치 비교 내부 상태의 현재 논리 상태를 저장합니다. 저장 설정 요소 **Gate3[j].Chan[j].EquOutMask** 및 **Gate3[j].Chan[j].EquOutPol** 에서 지정되도록 이 내부 상태는 IC 채널 중 하나 또는 모든 위치 비교 출력 신호 생성에 사용할 수 있습니다.

내부 상태를 직접 설정하기 위해 이 요소를 사용할 수 없습니다. 강제로 출력값을 변경하려면 2 비트의 비저장 설정 요소 **Gate3[j].Chan[j].EquWrite** 를 사용해야 합니다. 엔코더 카운터값이 **Gate3[j].Chan[j].CompA** 와 **Gate3[j].Chan[j].CompB** 로 설정한 위치를 통과하면 그 내부값은 자동으로 변화합니다.

Gate3[j].Chan[j].Equ 는 풀 워드 요소 **Gate3[j].Chan[j].Status** 의 비트 24 를 구성합니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[i].Chan[j].EquOut 게이트 3 채널의 위치 비교 종합 출력

설명: 위치 비교의 종합 출력 상태

범위: 0 ..1

단위: 논리값

Gate3[i].Chan[j].EquOut에는 채널상의 IC "EQU" 위치 비교 플래그 출력의 현재 논리 상태를 저장합니다. 1의 값은 IC의 high 전압 출력에 대응합니다. 0의 값은 IC의 low 전압 출력에 대응합니다. 대부분의 Delta Tau 하드웨어에서는 이 신호가 싱크 출력 드라이버를 제어합니다. 그래서 1의 상태가 출력 드라이버를 턴 온하고 전류를 인입하여 출력 전압을 low로 낮춥니다.

저장 설정 요소 **Gate3[i].Chan[j].EquOutMask** 및 **Gate3[i].Chan[j].EquOutPol**에서 지정되고 채널 출력은 IC의 (가능성으로)4 채널 모든 내부 비교 상태의 논리 결합으로 만들어집니다.

출력 상태를 직접 설정하기 위해 이 요소를 사용할 수 없습니다. 강제로 출력값을 변경하려면 2비트의 비저장 설정 요소 **Gate3[i].Chan[j].EquWrite**를 사용해야 합니다. 엔코더 카운터값이 **Gate3[i].Chan[j].CompA**와 **Gate3[i].Chan[j].CompB**로 설정한 위치를 통과하면 그 플래그 값은 자동으로 변화합니다.

Gate3[i].Chan[j].EquOut은 풀 워드 요소 **Gate3[i].Chan[j].Status**의 비트 25를 구성합니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[i].Chan[j].Fault 게이트 3 채널의 앰프 이상 플래그 입력

설명: 앰프 이상 플래그 입력값

범위: 0 ..1

단위: 논리값

Gate3[i].Chan[j].Fault에는 최신의 서보 인터럽트에서 래치된 채널 상의 IC 이상(앰프 이상) 플래그 입력의 현재 논리 상태를 저장합니다. 1의 값은 IC에 대한 high 전압에 대응합니다. 0의 값은 IC에 대한 low 전압에 대응합니다. 많은 Delta Tau 하드웨어에서는 이는 광절연 입력입니다. 그리고 플래그가 "단함"일 때 광 아이솔레이터에 전류가 흐르고 0이 됩니다. 이는 또한 차동 장치 입력과 사용할 수 있으며 그 경우에는 FAULT+입력이 FAULT-입력보다 높은 전압일 때 1의 값이 됩니다.

이 요소가 단순히 IC에 대한 입력 레벨을 나타낸다는 점에 주의하십시오. 0 또는 1 중에서 앰프 이상으로 판단하는지 **Motor[x].AmpFaultLevel**의 설정에 따릅니다.

이 하드웨어 요소는 소프트웨어 상태 요소의 **Motor[x].AmpFault**와는 다릅니다. **Motor[x].AmpFault**는 모터가 이 입력을 사용하도록 설정되면 이 입력에 대한 모터의 상태를 나타냅니다.

Gate3[i].Chan[j].Fault는 풀 워드 요소 **Gate3[i].Chan[j].Status**의 비트 7을 구성합니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[j].Chan[j].HallState 게이트 3 채널의 포착 홀 위치 상태

설명: 포착 홀 위치 상태값

범위: 0 ..7

단위: 열거 형식

채널이 U, V 또는 W 의 이동 중 어느 하나에서 엔코더 위치를 포착하도록 설정된 때(**Gate3[j].Chan[j].CaptCtrl** 비트 0 과 1 이 0 으로 **Gate3[j].Chan[j].GatedIndexSel** 이 1 로 설정) **Gate3[j].Chan[j].HallState** 에는 채널의 U, V 및 W 플래그 입력에 의한 홀 사이클에서 최신 포착된 위치 상태를 저장합니다.

6 상태 홀 사이클의 유효한 위치 상태값은 0~5 범위입니다. U, V 및 W 플래그 입력의 유효한 입력값에 대한 6 개의 값이 어디에 해당하는지는 다음 표와 같습니다.

Gate3[j].Chan[j].HallState	0	1	2	3	4	5
Gate3[j].Chan[j].UVW	1	3	2	6	4	5
U 플래그 입력상태 4	0	0	0	1	1	1
V 플래그 입력상태 2	0	1	1	1	0	0
W 플래그 입력상태 1	1	1	0	0	0	1

Power PMAC 은 홀 사이클 상태 0 과 상태 1 간의 이동을 제로 기준으로 간주합니다. 이것이 전류(commutation) 사이클의 영도점으로 규정하는 경우에 센서 위치에서 전류(commutation) 각도에 대한 오프셋은 필요하지 않습니다.

어떤 이동이 발생하기 전 또는 무효인 UVW 플래그 입력 상태로 이동한 경우에 **Gate3[j].Chan[j].HallState** 는 7 의 값을 통지합니다.

Gate3[j].Chan[j].HallState 에서는 풀 워드 요소 **Gate3[j].Chan[j].Status** 의 비트 16 - 18 을 구성합니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[j].Chan[j].HomeCapt 게이트 3 채널의 외부 트리거 래치를 통한 엔코더 카운트값

설명: 외부 트리거 래치를 통한 엔코더 카운트값

범위: 0 ..2³²-1

단위: 1/256 엔코더 카운트

Gate3[j].Chan[j].HomeCapt 에는 최신의 외부 트리거에서 래치된 IC 채널의 확장 하드웨어 엔코더 카운트값을 저장합니다. 이는 **Gate3[j].Chan[j].EncCtrl** 에서 지정되는 디코드 방법에 의한 카운트 단위의 엔코더값을 상위 24 비트에 가지는 32 비트값입니다. 카운터는 전원 투입/리셋으로 0 으로 설정되고 그때부터 카운트됩니다. 따라서 항상 전원 투입/리셋 위치가 기준이 됩니다. 이는 모든 방향으로 롤 오버할 수 있습니다.

트리거 상태는 저장 설정 요소 **Gate3[j].Chan[j].CaptCtrl**, **Gate3[j].Chan[j].CaptFlagSel** 및 **Gate3[j].Chan[j].CaptFlagChan** 에서 선택됩니다.

소수점 카운트 위치를 나타내는 하위 8 비트의 성분은 **Gate3[i].Chan[j].TimerMode** 설정에 좌우됩니다. **TimerMode** 가 0 의 기본값인 경우에 하위 8 비트는 카운트 타이밍에 따라 소수 카운트값을 추정하는 IC 의 "1/T"서브 카운트 보간 알고리즘에서 산출됩니다. **TimerMode** 가 제로가 아닌 값인 경우에 하위 8 비트는 1/2 카운트를 나타내는 값으로 고정됩니다. **Gate3[i].Chan[j].AtanEna** 의 설정이 채널 **PhaseCapt** 및 **ServoCapt** 요소에 부여하는 다양한 영향은 **HomeCapt** 요소의 하위 비트 성분에는 없습니다.

TimerMode 가 0 인 경우에 **Gate3[i].Chan[j].TimerB** 요소가 같은 트리거에서 래치된 위치 정보를 저장하지만 상위 20 비트에만 정수 카운트 정보를 저장하고 하위 12 비트에 1/T 확장 소수 카운트 위치를 저장합니다.

Gate3[i].Chan[j].HomeCapt 를 읽는 동작은 다음의 포착을 위해 트리거 회로를 재작동할 수 있습니다. 레벨 트리거에서 사용되므로 트리거 입력이 계속 지정된 상태인 경우에 즉시 재트리거가 발생하여 (가능성으로)이 요소에 새로운 카운트값을 래치합니다.

Gate3[i].Chan[j].HomeFlag 게이트 3 채널의 원점 플래그 입력

설명: 원점 플래그 입력값

범위: 0 ..1

단위: 논리값

Gate3[i].Chan[j].HomeFlag 에는 최신 서보 인터럽트에서 래치된 IC 채널의 원점 플래그 입력의 현재 논리 상태를 저장합니다. 1 의 값은 IC 에 대한 high 전압에 대응합니다. 0 의 값은 IC 에 대한 low 전압에 대응합니다. 대부분의 Delta Tau 하드웨어에서는 이는 광절연 입력입니다. 그리고 광 아이솔레이터가 전도하여 플래그가 "단함"일 때 0 이 됩니다.

Gate3[i].Chan[j].HomeFlag 는 풀 워드 요소 **Gate3[i].Chan[j].Status** 의 비트 8 을 구성합니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[i].Chan[j].LossCapt 게이트 3 채널의 엔코더 소실 래치 에러

설명: 엔코더 소실 래치 에러

범위: 0 ..1

단위: 논리값

Gate3[i].Chan[j].LossStatus 에서는 ASIC 의 AB 상 엔코더 검출 회로가 유효한 엔코더 신호가 소실되었다는 것을 검출했는지 나타냅니다. 유효 신호를 항상 검출하는 경우에는 0 입니다. 유효 신호를 검출할 수 없는 경우("엔코더 소실")에는 1 입니다. 회로는 차동 페어 신호상의 배타적 논리합 게이트를 사용합니다. 따라서 이 검출은 차동 라인 드라이버 출력을 가진 엔코더에 한해 유효합니다.

이 비트는 Power PMAC 의 전원 투입/리셋으로 자동으로 0 으로 설정됩니다. 어플리케이션은 이 비트에 0 을 쓰면 이 Status 비트를 클리어할 수 있습니다.

Gate3[i].Chan[j].LossCapt 는 풀 워드 요소 **Gate3[i].Chan[j].Status** 의 비트 29 입니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[j].Chan[j].LossStatus 게이트 3 채널의 현재 엔코더 소실 에러

설명: 현재의 엔코더 소실 에러

범위: 0 ..1

단위: 논리값

Gate3[j].Chan[j].LossStatus에서는 ASIC의 AB 상 엔코더 검출 회로가 현재 유효한 엔코더 신호를 검출했는지 나타냅니다. 유효 신호로 판단한 경우에는 0입니다. 유효 신호로 판단하지 않은("엔코더 소실") 경우에는 1입니다. 회로는 차동 페어 신호상의 배타적 논리합 게이트를 사용합니다. 따라서 이 검출은 차동 라인 드라이버 출력을 가진 엔코더에 한해 유효합니다.

Gate3[j].Chan[j].LossStatus는 풀 워드 요소 **Gate3[j].Chan[j].Status**의 비트 28입니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[j].Chan[j].MinusLimit 게이트 3 채널의 MLIM 플래그 입력

설명: MLIM 플래그 입력값

범위: 0 ..1

단위: 논리값

Gate3[j].Chan[j].MinusLimit에는 최신 서보 인터럽트에서 래치된 IC 채널의 MLIM(음의 오버트래블 리미트) 플래그 입력의 현재 논리 상태를 저장합니다. 1의 값은 IC에 대한 high 전압에 대응합니다. 0의 값은 IC에 대한 low 전압에 대응합니다. 대부분의 Delta Tau 하드웨어에서는 이는 광절연 입력입니다. 그리고 광 아이솔레이터가 전도하여 플래그가 "닫힘"일 때 0이 됩니다.

Gate3[j].Chan[j].MinusLimit는 풀 워드 요소 **Gate3[j].Chan[j].Status**의 비트 10입니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[j].Chan[j].PhaseCapt 게이트 3 채널의 페이즈 인터럽트 래치의 엔코더 카운트값

설명: 페이즈 인터럽트에서 래치된 엔코더 카운트값

범위: 0 ..2³²-1

단위: 1/256 엔코더 카운트(디지털)
 1/4096 엔코더 카운트(아날로그)

Gate3[j].Chan[j].PhaseCapt에는 최신 위상 인터럽트에서 래치된 IC 채널의 확장 하드웨어 엔코더 카운트값을 저장합니다. 이는 **Gate3[j].Chan[j].EncCtrl**에서 지정되는 디코드 방법에 의한 카운트 단위의 엔코더값을 상위 20 또는 24 비트에 가진 32비트값입니다. 카운터는 전원 투입/리셋으로 0으로 설정되고 그때부터 카운트됩니다. 따라서 항상 전원 투입/리셋 위치가 기준이 됩니다. 이는 모든 방향으로 롤 오버할 수

있습니다. 순간적인 카운터값이 프로세서에서 액세스할 수 없으므로 이 요소에는 현재 카운터값의 가장 직접적으로 액세스할 수 있는 대체값을 저장합니다.

저장 설정 요소 **Gate3[j].Chan[j].AtanEna** 가 0 의 기본값인 경우에 이 요소의 상위 24 비트는 래치된 엔코더 카운터값이 설정되고 또한 소수 카운트 위치를 나타내는 하위 8 비트의 성분은 **Gate3[j].Chan[j].TimerMode** 의 설정에 좌우됩니다. **TimerMode** 가 0 의 기본값인 경우에 하위 8 비트는 카운트 타이밍에 따른 소수점 카운트값을 추정하는 IC 의 "1/T" 서브 카운트 보간 알고리즘에서 산출됩니다. **TimerMode** 가 제로가 아닌 경우에 하위 8 비트는 1/2 카운트를 나타내는 값으로 고정됩니다.

Gate3[j].Chan[j].AtanEna 가 1 인 경우에 이 요소의 상위 20 비트는 래치된 엔코더 카운터값이 설정되고 하위 12 비트는 **Gate3[j].Chan[j].Atan** 값이 설정됩니다. 이는 사인과 코사인 입력의 아크탄젠트로 자동으로 산출된 소수점 카운트 정보를 저장합니다.

Gate3[j].Chan[j].PlusLimit 게이트 3 채널의 PLIM 플래그 입력

설명: PLIM 플래그 입력값

범위: 0 ..1

단위: 논리값

Gate3[j].Chan[j].PlusLimit 에는 최신의 서보 인터럽트에서 래치된 IC 채널의 PLIM(양의 오버트래블 리미트) 플래그 입력의 현재 논리 상태를 저장합니다. 1 의 값은 IC 에 대한 high 전압에 대응합니다. 0 의 값은 IC 에 대한 low 전압에 대응합니다. 대부분의 Delta Tau 하드웨어에서는 이는 광절연 입력입니다. 그리고 플래그가 "단함"일 때 광 아이솔레이터에 전류가 흐르고 0 이 됩니다.

Gate3[j].Chan[j].PlusLimit 는 풀 워드 요소 **Gate3[j].Chan[j].Status** 비트 9 입니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[j].Chan[j].PosCapt 게이트 3 채널의 위치 포착 트리거 플래그

설명: 위치 포착 트리거 플래그

범위: 0 ..3

단위: 비트 필드

Gate3[j].Chan[j].PosCapt 에는 채널 2 개의 트리거 플래그의 현재 상태를 저장합니다. 비트 1(값 2)은 채널의 "범용" 트리거 플래그 상태입니다. 이는 **Gate3[j].Chan[j].CaptCtrl**, **Gate3[j].Chan[j].CaptFlagSel** 및 **Gate3[j].Chan[j].CaptFlagChan** 으로 지정한 트리거 조건이 발생한 경우에는 항상 설정됩니다. 비트 0(값 1)은 "게이트화 인덱스"의 트리거 플래그 상태입니다. 이는 (**Gate3[j].Chan[j].CaptCtrl** 비트 0 = 1 및 **Gate3[j].Chan[j].GatedIndexSel** = 1 에서 설정된) 1 AB 상 쪽에서 "게이트화"된 엔코더의 인덱스 채널을 포함한 트리거 조건이 발생한 경우에는 항상 설정됩니다. 이 트리거는 주로 카운트 수가 연속되는 인덱스 펄스 폭 간의 정확한 카운트를 확립하기 위해 사용됩니다.

두 플래그는 트리거 순간에 포착된 엔코더 위치가 저장된 레지스터를 읽으면 자동으로 클리어됩니다. **Gate3[i].Chan[j].HomeCapt** 양쪽의 플래그는 레벨 트리거로 엣지 트리거가 아닙니다. 따라서 플래그가 클리어될 때 선택된 입력이 계속 지정된 트리거 상태이면 즉시 다시 트리거됩니다.

Gate3[i].Chan[j].PosCapt 는 풀 워드 요소 **Gate3[i].Chan[j].Status** 의 비트 20 - 21 을 구성합니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[i].Chan[j].SerialEncDataA 게이트 3 채널의 주 위치 피드백 시리얼 엔코더 데이터

설명: 주 위치 피드백의 시리얼 엔코더 데이터

범위: $0 \dots 2^{32}-1$

단위: 엔코더 위치 LSB

Gate3[i].Chan[j].SerialEncDataA 에는 채널의 최신 위상 또는 서보 클럭 사이클에서 시리얼 엔코더 인터페이스에서 수신된 하위 위치 피드백 데이터를 저장합니다. 이 요소에서는 몇 개의 다른 시리얼 엔코더 프로토콜 중의 1 개를 통한 32 비트 이내의 엔코더 데이터를 유지할 수 있습니다. 엔코더에서 수신된 32비트 이상의 데이터가 있는 경우에 상위 데이터는 **Gate3[i].Chan[j].SerialEncDataB** 에 있습니다.

AB 상 카운터

시리얼 엔코더 인터페이스가 유효하지 않은 경우(**Gate3[i].Chan[j].SerialEncEna** = 0) 디지털 AB 상 엔코더는 시리얼 엔코더 핀에 연결할 수 있고 이 32 비트 요소에는 IC 채널의 확장 하드웨어 엔코더 카운트값을 저장합니다. 이는 최신 서보 인터럽트에서 래치되고 상위 24비트는 엔코더 카운트 단위를 가지며 하위 8비트는 카운트 타이밍에 따라 소수 카운트값을 추정하는 IC의 "1/T" 서브 카운트 보간 알고리즘에서 산출됩니다. 카운트는 "4 채널" 디코드가 항상 사용됩니다(1 라인당 4 카운터).

SPI 프로토콜

DSPGATE3 IC에서는 채널 명령 워드에서 지정되도록 SPI 엔코더를 통한 32비트까지의 위치 데이터 및 12 비트 상태 데이터를 지원합니다. 위치 비트의 몇 비트가 "싱글 턴" 데이터이고 몇 비트가 "멀티 턴"인가 하는 일반 사양은 없습니다. 어느 특정 Status 비트가 무엇을 의미하는가 하는 일반 사양은 없습니다.

SPI 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataA** 는 아래와 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

싱글/멀티 턴 위치

비트 Pn 이 싱글 턴 및 멀티 턴 위치의 비트를 나타냅니다.

SSI 프로토콜

DSPGATE3 IC에서는 채널 명령 워드에서 지정되도록 SSI 엔코더를 통한 32비트까지의 위치 데이터 및 패리티 에러 비트를 지원합니다. 위치 비트의 몇 비트가 "싱글 턴" 데이터이고 몇 비트가 "멀티 턴"인지를 나타내는 사양은 없습니다.

SSI 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataA** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

싱글/멀티 턴 위치

비트 Pn 이 싱글 턴 및 멀티 턴 위치의 비트를 나타냅니다.

EnDat2.1/2.2 프로토콜

DSPGATE3 IC에서는 채널 명령 워드에서 지정되도록 EnDat 엔코더를 통한 48비트까지의 위치 데이터를 지원합니다. 위치 비트의 몇 비트가 "싱글 턴" 데이터이고 몇 비트가 "멀티 턴"인지를 나타내는 사양은 없습니다.

EnDat 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataA** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

싱글/멀티 턴 위치

비트 Pn 이 싱글 턴 및 멀티 턴 위치의 비트를 나타냅니다.

Hiperface 프로토콜

DSPGATE3 IC에서는 Hiperface 엔코더를 통한 32비트 위치 데이터 및 12비트 Status 데이터를 지원합니다.

Hiperface 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataA** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

싱글/멀티 턴 위치

비트 Pn 이 싱글 턴 및 멀티 턴 위치의 비트를 나타냅니다.

Yaskawa 시그마 I 프로토콜

애플루트 시그마 I 엔코더에서는 **Gate3[7].Chan[7].SerialEncDataA** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D	D	D	D	D	D	D	D	C	C	C	C	C	C	C	C	B	B	B	B	B	B	B	B	A	A	A	A	A	A	A	A
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

멀티 턴 위치

비트 An 이 회전수 카운트의 "1 위 자리"를 나타내는 ASCII 코드의 비트입니다. 비트 Bn 이 "10 위 자리"의 비트를 나타내며 비트 Cn 이 "100 위 자리"의 비트를 나타내며 비트 Dn 이 "1000 위 자리"의 비트를 나타냅니다.

수치와 ASCII 숫자 각각의 자리에 대해 수치로는 ASCII 코드값에서 48(\$30)을 빼서 구할 수 있습니다.

Yaskawa 시그마 III/III/V 프로토콜

1 회전당 17 비트를 가진 애플루트 시그마 II 엔코더에서는 **Gate3[j].Chan[j].SerialEncDataA** 는 아래와 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	M	M	M	M	M	M	M	M	M	M	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	-	-	-	-
10	9	8	7	6	5	4	3	2	1	0	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				

멀티 턴 위치

싱글 턴 위치

비트 Sn 이 싱글 턴 위치의 비트를 나타내며 비트 Mn 이 멀티 턴 위치("회전 카운트")의 비트를 나타냅니다. 전체 위치는 싱글 턴과 멀티 턴 값을 결합하여 "매끄럽게" 읽을 수 있습니다.

1 회전당 20 비트를 가진 앱솔루트 시그마 III 또는 V 엔코더에서는 **Gate3[j].Chan[j].SerialEncDataA** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	M	M	M	M	M	M	M	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
7	6	5	4	3	2	1	0	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-	-	-	-
멀티 턴 위치								싱글 턴 위치																							

비트 분해능은 17 비트/회전 절대값 엔코더와 같습니다.

1 회전당 17 비트를 가진 인크리멘탈 시그마 II 엔코더에서는 **Gate3[7].Chan[7].SerialEncDataA** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C	C	C	C	C	C	C	C	-	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	-	-	U	V	W	Z
13	12	11	10	9	8	7	6		16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
보정 위치								싱글 터 위치								홀인덱스															

비트 Sn 이 싱글 턴 위치의 비트를 나타내며 비트 Cn 이 보정 위치(인덱스에서 검출된 위치)의 비트를 나타냅니다. 인덱스가 검출된 경우(알람 코드 비트 A6 은 0 이 됨) 보정 위치는 적절한 기준 위치를 얻기 위해 싱글 턴 위치 데이터에서 뺍니다. 보정 위치의 11 비트는 17 비트 싱글 턴 위치의 상위 11 비트와 일치한다는 점에 주의하십시오.

U, V 및 W 는 3 개의 위상 "홀" 전류(commutation) 센서 신호를 나타냅니다. 이는 마치 실제 홀 센서처럼 애플루트 전원 투입 위상 참조에 사용할 수 있습니다. Z 는 엔코더 인덱스(제로) 펄스 상태를 나타냅니다.

Tamagawa FA 코더 프로토콜

1 회전당 17 비트 및 16 비트의 "회전 카운트"를 가진 Tamagawa FA 코더 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataA** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	M	M	M	M	M	M	M	0	0	0	0	0	0	0	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
7	6	5	4	3	2	1	0								16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
멀티 턴 위치								싱글 턴 위치																							

비트 Sn 이 싱글 턴 및 멀티 턴 위치의 비트를 나타냅니다. 전체 위치는 싱글 턴과 멀티 턴값을 결합하여 읽을 수 있습니다. 그러나 해당 값이 서로 직접 인접해 있지 않다는 점에 주의하십시오. 따라서 이들을 결합할 때에 주의해야 합니다.

Panasonic 프로토콜

1 회전당 17 비트 및 16 비트의 "회전 카운트"를 가진 Panasonic 시리얼 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataA** 는 멀티 턴 위치 통지를 위해 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	M	M	M	M	M	M	M	0	0	0	0	0	0	0	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
7	6	5	4	3	2	1	0								16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
멀티 턴 위치								싱글 턴 위치																							

비트 Sn 이 싱글 턴 위치의 비트를 나타내며 비트 Mn 이 멀티 턴 위치("회전 카운트")의 비트를 나타냅니다. 전체 위치는 싱글 턴과 멀티 턴값을 결합하여 읽을 수 있습니다. 그러나 해당 값이 서로 직접 인접해 있지 않다는 점에 주의하십시오. 따라서 이들을 결합할 때에 주의해야 합니다.

1 회전당 17 비트 및 16 비트의 "회전 카운트"를 가진 Panasonic 시리얼 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataA** 는 알람 코드로 통지하는 싱글 턴 위치를 위해 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID	ID	ID	ID	ID	ID	ID	ID	0	0	0	0	0	0	0	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
7	6	5	4	3	2	1	0								16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
엔코더 ID 부호								싱글 턴 위치																							

비트 Sn 이 싱글 턴 및 멀티 턴 위치의 비트를 나타냅니다. 엔코더 ID 코드의 비트 ID0 - ID7 이 \$11 로 고정됩니다.

Mitutoyo 프로토콜

Mitutoyo 엔코더에서는 **Gate3[j].Chan[j].SerialEncDataA** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

절대 위치

비트 Pn 이 절대 위치의 비트를 나타냅니다.

Kawasaki 프로토콜

Kawasaki 엔코더에서는 **Gate3[j].Chan[j].SerialEncDataA** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	M	M	M	M	M	M	M	M	M	M	M	C	C	S	S	S	S	S	S	S	S	S	S	S	S	S	S	I	I	I	I
11	10	9	8	7	6	5	4	3	2	1	0	1	0	13	12	11	10	9	8	7	6	5	4	3	2	1	0	3	2	1	0

멀티 턴 위치 보정 값 싱글 턴 위치 보간 위치

비트 In 이 보간된 위치의 비트를 나타냅니다. 비트 Sn 이 싱글 턴 및 멀티 턴 위치의 비트를 나타냅니다. 비트 Cn 이 멀티 턴 보정 정보의 비트를 나타냅니다. 비트 Mn 이 멀티 턴 위치의 비트를 나타냅니다.

Gate3[j].Chan[j].SerialEncDataB 게이트 3 의 시리얼 엔코더 부위치 피드백 데이터

설명: 시리얼 엔코더의 부위치 피드백 데이터

범위: $0 \dots 2^{32}-1$

단위: 2^{32} *엔코더 위치 LSB

존재한다면 **Gate3[j].Chan[j].SerialEncDataB** 에는 프로토콜 좌우의 Status 및 에러 정보를 더한 채널의 시리얼 엔코더 피드백에서 상위 위치 피드백데이터를 저장합니다. 32 비트의 하위 위치 데이터(비트 0 - 31)는 **Gate3[j].Chan[j].SerialEncDataA** 에서 구할 수 있습니다. 피드백에 32 비트 이상의 데이터가 있는 경우에 이 데이터는 **Gate3[j].Chan[j].SerialEncDataB** 의 최하위 비트(데이터의 비트 32 는 이 요소의 비트 0 에 대응하며 이하 동일)에서 구할 수 있습니다.

Status 와 에러 정보는 이 요소의 최상위부터 배치됩니다. 이 정보의 중요성 및 의미는 특정 프로토콜에 좌우됩니다.

코드	프로토콜	Status 와 에러 정보
1	SPI	비트 31 - 20: 엔코더에 좌우됨
2	SSI	비트 31: 패리티 에러
3	EnDat	비트 31: 타임아웃 에러 비트 30: CRC 에러 비트 29: 에러 비트

4	Hiperface	비트 31: 타임아웃 에러 비트 30: 검사합 에러 비트 29: 패리티 에러 비트 28: 에러 비트 비트 27 - 20: 에러 코드
5	시그마 I	비트 31: 타임아웃 에러 비트 30: 패리티 에러
6	시그마 II	비트 31: 타임아웃 에러 비트 30: CRC 에러 비트 29: 코딩 시 에러 비트 28: (예약) 비트 27 - 20: 이상 코드
7	Tamagawa	비트 31: 타임아웃 에러 비트 30: CRC 에러 비트 29 - 28: (예약) 비트 27 - 24 : Status 영역
8	Panasonic	비트 31: 타임아웃 에러 비트 30: CRC 에러 비트 29 - 28: (예약) 비트 27 - 24: Status 영역
9	Mitutoyo	비트 31: 타임아웃 에러 비트 30: CRC 에러 비트 29 - 28: (예약) 비트 27 - 24: Status 영역
10	Kawasaki	비트 31: 타임아웃 에러 비트 30: CRC 에러 비트 29: 코딩 시 에러 비트 28 - 27: (예약) 비트 26: 보간 오차 비트 25: 위치 에러 비트 24: 사용 중 플래그
-	(AB 상)	비트 31: 카운트 에러

AB 상 엔코더

AB 상 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataB** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
0																															
CE																															

비트 E0 은 AB 상 이동이 무효일 때 설정되는 카운트 에러 비트를 나타냅니다.

SPI 프로토콜

SPI 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataB** 는 다음과 같이 설정됩니다.

비트 Fn 이 Status 영역의 비트를 나타냅니다.

SSI 프로토콜

SSI 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataB** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
0																															
PE																															

비트 E0 은 패리티 에러 비트를 나타냅니다.

EnDat2.1/2.2 프로토콜

EnDat 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataB** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0					
E	E	E	-	-	-	-	-	-	-	-	-	-	-	-	-	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P						
2	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32					
TE CE EB																																				

비트 Pn 이 싱글 턴 및 멀티 턴 위치의 비트를 나타냅니다. 비트 En 은 에러 비트를 나타냅니다(E0 은 엔코더에서 통지된 에러, E1 은 IC 에서검출된 CRC 에러, E2 가 IC에서 검출된 타임 아웃 에러).

Hiperface 프로토콜

DSPGATE3 IC에서는 Hiperface 엔코더를 통한 32 비트 위치 데이터 및 12 비트 Status 데이터를 지원합니다.

Hiperface 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataB** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	E	E	E	C	C	C	C	C	C	C	C	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
3	2	1	0	7	6	5	4	3	2	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
에러 비트 에러 코드																															

비트 Cn 이 엔코더의 에러 코드 비트를 나타냅니다(Status 요구에 한해 통지). 비트 En 은 에러 비트를 나타냅니다(E0 은 엔코더에서 통지된 에러, E1 이 IC 에서 검출된 패리티 에러, E2 가 IC에서 검출된 검사합 에러 그리고 E3 이 타임아웃 에러).

Yaskawa 시그마 I 프로토콜

Yaskawa 시그마 I 절대값 엔코더 프로토콜에서는 특정 요구에 따라 ASCII 텍스트 형식으로 절대값 회전 카운트 위치 데이터를 제공합니다. 절대값 시그마 I 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataB** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F	F							P	P	P	P	P	P	P	P	S	S	S	S	S	S	S	S	E	E	E	E	E	E	E	E
1	0	-	-	-	-	-	-	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Err Bits								"P"								부호(+/-)								멀티 턴 위치							

비트 En 이 회전 카운트의 "10000 위 자리" 비트를 나타냅니다. 비트 Sn 이 플러스 또는 마이너스의 ASCII 코드 비트를 나타냅니다. 비트 Pn 이 문자 "P"용 ASCII 코드 비트를 나타냅니다. 비트 Fn 이 에러 영역(F0 은 패리티 에러, F1 은 타임아웃 에러)의 비트를 나타냅니다.

5 개의 수치와 ASCII 숫자 각각의 자리에 대해 수치로는 ASCII 코드값에서 48(\$30)을 빼서 구할 수 있습니다.

Yaskawa 시그마 III/IV 프로토콜

1 회전당 17 비트를 가진 절대값 시그마 II 엔코더에서는 Gate3[j].Chan[j].SerialEncDataB 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	E	E	-	A	A	A	A	A	A	A	A	T	T	T	T	T	T	T	T	-	-	-	-	-	-	-	M	M	M	M	M
2	1	0		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	-	-	-	-	-	-	-	15	14	13	12	11
에러 비트				알람 코드								온도								멀티 턴 위치											

비트 Mn 이 멀티 턴 위치("회전 카운트")의 비트를 나타냅니다. 전체 위치는 싱글 턴과 멀티 턴 값을 결합하여 "매끄럽게" 읽을 수 있습니다. 비트 Tn 은 °C로 통지되는 온도의 비트를 나타냅니다(만일 제공되는 경우).

비트 An 은 절대값 엔코더를 위한 알람 코드의 비트를 나타내며 각각의 비트가 가진 의미는 다음과 같습니다.

- A0: 배터리 백업된 회전 데이터의 소실
- A1: 전원 투입 자체 진단 에러
- A2: 배터리 전압 저하 경고
- A3: 절대값 위치 에러
- A4: 과속도 에러
- A5: 온도 상승 에러
- A6: 엔코더 리셋 중
- A7: (예약)

비트 En 은 IC 에서 검출된 에러 비트를 나타내며 각각의 비트가 가진 의미는 다음과 같습니다.

- E0: 코드 에러
- E1: CRC 에러
- E2: 타임아웃 에러

1 회전당 20 비트를 가진 절대값 시그마 III/IV 엔코더에서는 Gate3[j].Chan[j].SerialEncDataB 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	E	E	-	A	A	A	A	A	A	A	A	T	T	T	T	T	T	T	T	-	-	-	-	-	-	-	M	M	M	M	M
2	1	0		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	-	-	-	-	-	-	-	15	14	13	12	11
에러 비트				알람 코드								온도								멀티 턴 위치											

비트의 의미는 17-비트/회전 절대값 엔코더와 같습니다.

1 회전당 17 비트를 가진 인크리멘탈 시그마 II 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataB** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	E	E	-	A	A	A	A	A	A	A	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	C	C	C
2	1	0		7	6	5	4	3	2	1	0																	16	15	14	
에러 비트				알람 코드																								보정 위치			

비트 Cn 은 보정 위치(인덱스에서 검출된 위치)의 비트를 나타냅니다. 인덱스가 검출된 경우(알람 코드 비트 A6 은 0 이 됨) 보정 위치는 적절한 기준 위치를 얻기 위해 싱글 턴 위치 데이터에서 뺍니다. 보정 위치의 11 비트는 17 비트 싱글 턴 위치의 상위 11 비트와 일치한다는 점에 주의하십시오.

비트 An 은 인크리멘탈.엔코더를 위한 알람 코드의 비트를 나타내며 각각의 비트가 가진 의미는 다음과 같습니다.

- A0: (예약)
- A1: 전원 투입 자체 진단 에러
- A2: (예약)
- A3: 부정 회전 카운트(인덱스에서 인덱스)
- A4: (예약)
- A5: (예약)
- A6: 기준 위치(인덱스) 미검출
- A7: (예약)

비트 En 은 IC 에서 검출된 에러 비트를 나타내며 각각의 비트가 가진 의미는 다음과 같습니다.

- E0: 코드 에러
- E1: CRC 에러
- E2: 타임아웃 에러

Tamagawa FA 코더 프로토콜

1 회전당 17 비트 및 16 비트의 "회전수 카운트"를 가진 Tamagawa FA 코더 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataB** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	E	-	-	F	F	F	F	A	A	A	A	A	A	A	A	-	-	-	-	-	-	-	-	M	M	M	M	M	M	M	M
1	0	-	-	7	6	5	4	7	6	5	4	3	2	1	0	-	-	-	-	-	-	-	-	15	14	13	12	11	10	9	8
에러 비트 Status 영역 알람 코드																멀티 턴 위치															

비트 Mn 은 멀티 턴 위치("회전수 카운트")의 비트를 나타냅니다. 전체 위치는 싱글 턴과 멀티 턴값을 결합하여 읽을 수 있습니다. 그러나 해당 값이 서로 직접 인접해 있지 않다는 점에 주의하십시오. 따라서 이들을 결합할 때에 주의해야 합니다. 비트 An 은 알람 코드의 비트를 나타내며 비트 Fn 은 Status 영역의 비트를 나타내고 비트 En 은 에러 비트를 나타냅니다(E0 은 CRC 에러, E1 은 타임아웃 에러).

Panasonic 프로토콜

1 회전당 17 비트 및 16 비트의 "회전수 카운트"를 가진 Panasonic 시리얼 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataB** 는 멀티 턴 위치 통지를 위해 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	E	-	-	F	F	F	F	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	M	M	M	M	M	M	M	M
1	0	-	-	7	6	5	4	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	15	14	13	12	11	10	9	8

에러 비트 Status 영역 멀티 턴 위치

비트 Mn 은 멀티 턴 위치("회전수 카운트")의 비트를 나타냅니다. 전체 위치는 싱글 턴과 멀티 턴값을 결합하여 읽을 수 있습니다. 그러나 해당 값이 서로 직접 인접해 있지 않다는 점에 주의하십시오. 따라서 이들을 결합할 때에 주의해야 합니다. 비트 Fn 은 Status 영역의 비트를 나타내고 비트 En 은 에러 비트를 나타냅니다(E0 은 CRC 에러, E1 은 타임아웃 에러).

1 회전당 17 비트 및 16 비트의 "회전수 카운트"를 가진 Panasonic 시리얼 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataB** 는 알람 코드를 동반하여 싱글 턴 위치를 통지하기 위해 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	E	-	-	F	F	F	F	-	-	-	-	-	-	-	-	A	A	A	A	A	A	A	A	ID	ID	ID	ID	ID	ID	ID	ID
1	0	-	-	7	6	5	4	-	-	-	-	-	-	-	-	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8

에러 비트 Status 영역 알람 코드 엔코더 ID 코드

비트 ID8-ID15 는 엔코더 ID 코드의 비트를 나타냅니다. 비트 A0 -A7 은 프로세서에서 통지된 알람 코드를 나타내며 각각의 비트가 가진 의미는 다음과 같습니다.

- A0: 과속도 에러
- A1: 완전한 분해능 = 1 의 경우 100rpm 이상일 때 분해능 저하를 통지
- A2: 카운트 에러
- A3: 카운터 오버플로
- A4: (예약)
- A5: 다중 회전 에러
- A6: 시스템 전압 부족 에러(<2.5V)
- A7: 배터리 저하(<3.1V)

Mitutoyo 프로토콜

Mitutoyo 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataB** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	E	-	-	F	F	F	F	A	A	A	A	A	A	A	A	ID	ID	ID	ID	ID	ID	ID	ID	-	-	-	-	-	-	-	-
1	0	-	-	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	-	-	-	-	-	-	-	-

에러 비트 Status 영역 알람 코드 엔코더 ID

비트 IDn 은 엔코더 ID 의 비트를 나타냅니다(특정 요구에 한해 제공). 비트 An 은 엔코더에서 통지된 알람 코드의 비트를 나타내며 각각의 비트가 가진 의미는 다음과 같습니다.

- A0: 초기화 에러
- A1: 광학 및 용량성 센서의 불일치
- A2: 광센서 에러
- A3: 용량성 센서 에러
- A4: CPU 에러(AT303), CPU/ROM/RAM 에러(AT503)
- A5: EEPROM 에러
- A6: ROM/RAM 에러(AT303), 통신 에러(AT503)
- A7: 과속도 에러

비트 Fn 은 엔코더에서 통지된 Status 영역의 비트를 나타내며 각각의 비트가 가진 의미는 다음과 같습니다.

- F0: 치명적인(회복 불능) 엔코더 에러
- F1: (예약)
- F2: 컨트롤러에 의한 부정 명령 코드
- F3: (예약)

비트 En 은 IC 에서 검출된 에러 코드의 비트를 나타내며 각각의 비트가 가진 의미는 다음과 같습니다.

- E0: CRC 에러
- E1: 타임아웃 에러

Kawasaki 프로토콜

Kawasaki 엔코더에서는 **Gate3[i].Chan[j].SerialEncDataB** 는 다음과 같이 설정됩니다.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	E	E	-	-	A	A	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M	M	M	M
2	1	0			2	1	0																					15	14	13	12
에러 비트 알람 코드																멀티 터 위치															

에러 비트 알람 코드

멀티 턴 위치

비트 Mn 은 멀티 턴 위치의 비트를 나타냅니다. 비트 An 은 엔코더에서 통지된 알람 코드 비트를 나타냅니다(A0 은 보간 회로 에러, A1 은 절대값 트랙 에러, A2 는 사용 중 플래그). 비트 En 은 IC 에서 검출된 에러 코드의 비트를 나타냅니다(E0 은 코드 에러, E1 은 CRC 에러, E2 는 타임아웃 에러).

Gate3[i].Chan[j].ServoCapt 게이트 3 채널의 서보 인터럽트 래치 엔코더 카운트

설명: 서보 인터럽트에서 래치된 엔코더 카운트값

범위: 0 ..2³²- 1

단위: 1/256 엔코더 카운트(디지털)
1/4096 엔코더 카운트(아날로그)

Gate3[j].Chan[j].ServoCapt 에는 최신 서보 인터럽트에서 래치된 IC 채널의 확장 하드웨어 엔코더 카운트값을 저장합니다. 이는 **Gate3[j].Chan[j].EncCtrl** 에서 지정되는 디코드 방법에 의한 카운트 단위의 엔코더값을 상위 20 또는 24 비트에 가진 32 비트값입니다. 카운터는 전원 투입/리셋으로 0으로 설정되고 그때부터 카운트됩니다. 따라서 항상 전원 투입/리셋 위치가 기준이 됩니다. 이는 모든 방향으로 롤 오버할 수 있습니다.

저장 설정 요소 **Gate3[j].Chan[j].AtanEna** 가 0 의 기본값인 경우에 이 요소의 상위 24 비트는 래치된 엔코더 카운터값이 설정되고 또한 소수 카운트 위치를 나타내는 하위 8 비트의 성분은 **Gate3[j].Chan[j].TimerMode** 의 설정에 좌우됩니다. **TimerMode** 가 0 의 기본값인 경우에 하위 8 비트는 카운트 타이밍에 따른 소수 카운트값을 추정하는 IC 의 "1/T" 서브 카운트 보간 알고리즘에서 산출됩니다. **TimerMode** 가 제로가 아닌 값인 경우에 하위 8 비트는 1/2 카운트를 나타내는 값으로 고정됩니다.

TimerMode 가 0 인 경우에 **Gate3[j].Chan[j].TimerA** 요소가 같은 서보 인터럽트에서 래치된 위치 정보를 저장하지만 상위 20 비트만 정수 카운트 정보를 구성하고 하위 12 비트에 1/T 확장 소수 카운트 위치를 구성합니다.

Gate3[j].Chan[j].AtanEna 가 1 인 경우에 이 요소의 상위 20 비트는 래치된 엔코더 카운터값이 설정되고 하위 12 비트는 **Gate3[j].Chan[j].Atan** 값이 설정됩니다. 이는 사인과 코사인 입력의 아크탄젠트로 자동으로 산출된 소수 카운트 정보를 저장합니다.

Gate3[j].Chan[j].SosError 게이트 3 채널의 사인/코사인 제곱합 에러

설명: 사인/코사인 "제곱합" 에러

범위: 0 ..1

단위: 논리값

Gate3[j].Chan[j].SosError 에는 IC 채널의 "제곱합" 에러 Status 플래그의 현재 논리 상태를 저장합니다. 1 의 값은 사인과 엔코더 또는 리졸버의 "사인" 및 "코사인"을 대상으로 하는 채널의 엔코더 ADC 입력(**Gate3[j].Chan[j].AdcEnc[0]**과 **Gate3[j].Chan[j].AdcEnc[1]**)의 제곱합이 유효 신호로 간주하기에 충분히 크지 않다는 것을 나타냅니다. 0 의 값은 유효 신호를 확립하기 위해 충분히 큰 제곱합임을 나타냅니다.

IC 는 자동으로 위상 사이클마다 해당 2 개의 부호가 있는 ADC 레지스터의 제곱합을 산출하고 부호가 없는 16 비트 요소 **Gate3[j].Chan[j].SumOfSquares** 에 결과를 입력합니다. 이 요소의 상위 4 비트가 위상 사이클에서 모두 0 인 경우에 **Gate3[j].Chan[j].SosError** 는 1 로 설정됩니다. 이는 값이 발생할 수 있는 최대값의 1/16 미만임을 나타내며 사인과 코사인값은 발생할 수 있는 최대값의 1/4 미만입니다. 사이클에서 16비트의 상위 4비트 중 하나가 1 인 경우에 **Gate3[j].Chan[j].SosError** 는 0 이 됩니다.

Gate3[j].Chan[j].SosError 는 풀 워드 요소 **Gate3[j].Chan[j].Status** 의 비트 31 입니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[i].Chan[j].Status 게이트 3 채널의 에러 코드 Status 요소

설명: "에러 코드" Status 요소

범위: \$0..\$FFFFFFFF

단위: 비트 필드

Gate3[i].Chan[j].Status 는 IC 의 채널용 하드웨어 상태 레지스터 성분을 나타내는 32 비트 Status 워드입니다. 이는 많은 특정 Status 비트 및 요소를 저장합니다. 특정 Status 비트 및 요소는 스크립트 프로그램 및 스크립트 명령에서는 별도의 요소로 액세스할 수 있지만 C 프로그램이나 C 함수에서는 할 수 없습니다.

워드 속의 Status 비트 목록은 다음 표와 같습니다.

비트#	16 진수값	요소명: Gate3[i].Chan[j]. {요소}	설명
31	\$80000000	SosError	사인/코사인 "제공함" 에러
30	\$40000000	CountError	엔코더 디코드 카운트 에러
29	\$20000000	LossCapt	엔코더 소실 래치 에러
28	\$10000000	LossStatus	엔코더 소실 현재 에러
26 - 27	\$C0000000	-	(추후 사용하기 위해 예약)
25	\$20000000	EquOut	위치 비교 출력 결합 상태
24	\$10000000	Equ	위치 비교 내부 상태
23	\$8000000	-	(추후 사용하기 위해 예약)
22	\$4000000	TrigState	현재의 트리거 입력 결합 상태
20 - 21	\$3000000	PosCapt	위치 포착 트리거 플래그
19	\$800000	DemuxInvalid	AB 상에 의한 UVW 무효 디멀티플렉싱
16 - 18	\$700000	HallState	포착된 홀의 상태
15	\$80000	T	T 플래그 입력값
12 - 15	\$7000	UVW	U, V 및 W 플래그 입력값
11	\$800	UserFlag	사용자 플래그 입력값
10	\$400	MinusLimit	MLIM 플래그 입력값
9	\$200	PlusLimit	PLIM 플래그 입력값
8	\$100	HomeFlag	원점 플래그 입력값
7	\$80	Fault	앰프 폴트 플래그 입력값
4 - 6	\$70	ABC	엔코더 A, B 그리고 C 채널 입력값
0 - 3	\$F	ABPins	엔코더 A+, A-, B+, B- 신호 레벨

각각의 요소는 각 사양 항목을 통해 자세하게 설명합니다.

Gate3[i].Chan[j].SumOfSquares 게이트 3 채널의 엔코더 사인/코사인 진폭값

설명: 엔코더 사인/코사인 진폭값

범위: 0 ..65,535

단위: (16 비트 ADC 단위)²/32,768

Gate3[i].Chan[j].SumOfSquares에는 주 엔코더 아날로그/디지털 변환값에서 ASIC로 자동으로 산출된 진폭값을 저장합니다. 이러한 값은 **Gate3[i].Chan[j].AdcEnc[0]**(사인) 및 **Gate3[i].Chan[j].AdcEnc[1]**(코사인)에 있습니다. 이러한 값은 위상 클록의 상승 엣지로 스트로브되고 위상 클록의 하강 엣지로 데이터를 사용할 수 있도록 (이는 프로세서 인터럽트에서 제공하는) A/D 변환기에서 구할 수 있습니다. **SumOfSquares** 값은 위상 인터럽트에서 즉시 사용 가능합니다(위상 사이클의 절반에 25ADC 클록 사이클과 2μsec의 충분한 시간이 있는 경우).

이 값을 생성하는 계산은 ADC의 실제 분해능에 관계없이 입력으로 ADC 레지스터의 상위 16비트를 이용하여 부호가 있는 값으로 다룹니다. 최초로 저장 설정 요소 **Gate3[i].Chan[j].AdcOffset[0]** 및 **Gate3[i].Chan[j].AdcOffset[1]**의 값을 각각의 입력에 더합니다(이러한 항은 어플리케이션이 아날로그 바이어스를 보정하는 것을 가능하게 합니다). 다음에 각각의 32비트 결과의 보정값을 제공하고 2개의 값을 더하여 상위 16비트의 결과를 사용합니다.

사인 및 코사인의 ADC 값이 해당 전체 범위(이러한 레지스터 상위 16비트에서 피크 +/- 32,768에 달함)를 사용하는 경우에는 **SumOfSquares** 값은 약 16,384입니다. 16,384를 초과하는 **SumOfSquares** 값은 아날로그 입력의 포화 문제를 나타냅니다.

Gate3[i].Chan[j].SumOfSquares는 사인과 엔코더 또는 리졸버와 같은 사인/코사인 피드백 장치 설정과 디버그를 실시할 때에 진단값으로 주로 사용됩니다. 이 16비트값의 최상위 4비트가 모두 0인 경우에 채널 Status 비트 **Gate3[i].Chan[j].SosError**가 1로 설정된다는 점에 주의하십시오. 이 비트는 센서 소실 지표로 사용할 수 있습니다.

Gate3[i].Chan[j].SumOfSquares는 풀 워드 요소 **Gate3[i].Chan[j].AtanSumOfSqr**의 비트 0 - 15를 구성합니다. C 프로그램과 함수는 풀 워드 요소를 사용해야 합니다.

Gate3[i].Chan[j].T 게이트 3 채널의 T 플래그 입력

설명: T 플래그 입력값

범위: 0 ..1

단위: 논리값

Gate3[i].Chan[j].T에는 최신의 서보 인터럽트에서 래치된 IC 채널의 T 플래그 입력의 현재 논리 상태를 저장합니다. 1의 값은 IC에 대한 high 전압에 대응합니다. 0의 값은 IC에 대한 low 전압에 대응합니다. 대부분의 Delta Tau 하드웨어에서 IC에 대한 외부 버퍼 입력은 비반전입니다. 따라서 이는 컨트롤러에 대한 전압과 일치합니다.

Gate3[i].Chan[j].T는 풀 워드 요소 **Gate3[i].Chan[j].Status**의 비트 15입니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[i].Chan[j].TimerA 게이트 3 채널의 타이머 모드 정보

설명: 타이머 모드 정보

범위: $0 \dots 2^{32}-1$

단위: (모드에 좌우됨)

Gate3[i].Chan[j].TimerA 에는 4 개 모드 중 1 개의 타이머 베이스 피드백 정보를 저장합니다.

저장 설정 요소 **Gate3[i].Chan[j].TimerMode** 가 0 의 기본값인 경우에 타이머 회로는 "1/T" 카운트 보간용으로 설정되고 32 비트의 **TimerA** 요소는 서보 인터럽트에서 래치된 엔코더 카운트값을 저장합니다. 이는 상위 20 비트에 정수 카운트 정보를 갖추고 있으며 하위 12 비트에 1/T 확장 소수 카운트 위치를 갖추고 있습니다. 이는 소수 카운트 분해능의 4 비트만큼 많은 것 이외에는(그리고 풀 카운트 범위의 4 비트만큼 적음) **Gate3[i].Chan[j].ServoCapt** 의 값과 같아집니다.

TimerMode 가 1 인 경우에 타이머 회로는 MLDT "펄스 에코" 모드를 위해 설정됩니다. 또한 **TimerA** 요소의 상위 24 비트는 이전의 서보 사이클 시작에 발생한 출력 펄스와 MLDT 센서를 통한 "에코 펄스" 수신 사이의 시간에 비례한 값을 저장합니다. 이 시간은 600MHz 클록 사이클(1.66667nsec) 단위로 베이스부터 MLDT 가동 자석의 거리에 정비례합니다.

TimerMode 가 2 인 경우에 타이머 회로는 트리거 입력 타이밍을 위해 설정됩니다. 그리고 **TimerA** 요소의 상위 24 비트는 마지막 서보 인터럽트로 지정된 트리거 조건 접수 사이의 시간을 저장합니다. 이는 엔코더 샘플 클록 사이클(SCLK) 단위이며 그 주기는 **Gate3[i].EncClockDiv** 에서 설정됩니다. 이 값은 트리거가 발생한 서보 사이클 시간의 비율을 검출하기 위해 **TimerB** 요소의 값으로 나눌 수 있습니다.

TimerMode 가 3 인 경우에 타이머 회로는 내부 펄스 계수를 위해 설정되고 **TimerA** 요소의 상위 24 비트에는 최신의 서보 인터럽트에서 래치된 채널의 D 위상 펄스 주파수 변조(PFM) 회로를 통한 누적된 펄스 카운트값을 저장합니다. 하위 8 비트는 모두 0 으로 읽히고 서브 카운트 보간은 없습니다. 위치 확인용으로 실제 엔코더에 이용할 수 있는 엔코더 카운터 회로를 남긴 상태에서 종래의 스테핑 모터 드라이브에 대한 펄스와 방향 출력을 생성할 때 이 값은 유사 피드백으로 사용할 수 있습니다.

Gate3[i].Chan[j].TimerB 게이트 3 채널의 타이머 모드 정보

설명: 타이머 모드를 통한 정보

범위: $0 \dots 2^{32}-1$

단위: (모드에 좌우됨)

Gate3[i].Chan[j].TimerB 에는 4 개 모드 중 1 개의 타이머 베이스 피드백 정보를 저장합니다.

저장 설정 요소 **Gate3[j].Chan[j].TimerMode** 가 0 의 기본값인 경우에 타이머 회로는 "1/T" 카운트 보간용으로 설정되고 32 비트의 **TimerB** 요소는 트리거 래치된 엔코더 카운트값을 저장합니다. 이는 상위 20 비트에 정수 카운트 정보를 갖추고 있으며 하위 12 비트에 1/T 확장 소수 카운트 위치를 갖추고 있습니다. 이는 소수 카운트 분해능의 4 비트만큼 많은 것 이외에는(또한 풀 카운트 범위의 4 비트만큼 적음) **Gate3[j].Chan[j].HomeCapt** 의 값과 같아집니다.

TimerMode 가 1 인 경우에 타이머 회로는 MLDT "펄스 에코" 모드를 위해 설정됩니다. 또한 **TimerB** 요소의 상위 24 비트는 이전의 서보 사이클 시작에 발생한 출력 펄스와 현재 서보 사이클 시작에 발생한 펄스 사이의 시간에 비례한 값을 저장합니다. 이 시간은 600MHz 클럭 사이클(1.66667nsec) 단위로 **TimerA** 요소의 스케일링을 확장하기 위해 사용되며 이는 그 모드와 관련된 측정값을 저장합니다.

TimerMode 가 2 인 경우에 타이머 회로는 트리거 입력 타이밍을 위해 설정되고 **TimerB** 요소의 상위 24 비트는 연속되는 서보 인터럽트 사이의 시간을 저장합니다. 이는 엔코더 샘플 클럭 사이클(SCLK) 단위이며 그 주기는 **Gate3[j].EncClockDiv** 에서 설정됩니다. 이 값은 트리거가 발생한 서보 사이클 시간의 비율을 검출하기 위해 **TimerA** 요소의 값을 나눌 수 있습니다.

Gate3[j].Chan[j].TrigState 게이트 3 채널의 위치 포착의 현재 트리거 입력

설명: 위치 포착의 현재 트리거 입력 상태

범위: 0 .. 1

단위: 논리값

Gate3[j].Chan[j].TrigState 에는 채널의 위치 포착 함수 트리거 입력 신호의 현재 상태를 저장합니다. 트리거 신호는 **Gate3[j].Chan[j].CaptCtrl**, **Gate3[j].Chan[j].CaptFlagSel** 및 **Gate3[j].Chan[j].CaptFlagChan** 에서 지정되고 인덱스 채널 및 입력 플래그의 논리 결합입니다.

1 의 값은 트리거 상태를 나타냅니다. 0 의 값은 비트리거 상태를 나타냅니다. 위치 포착의 실제 트리거는 **Gate3[j].Chan[j].PosCapt** 의 1 개 또는 양쪽의 비트를 1 로 설정한 후 이 값이 0 에서 1 로 변화하는 순간에서 발생하며 또는 이 요소가 1 에서 포착 위치 레지스터 **Gate3[j].Chan[j].HomeCapt** 를 읽었을 때 자동으로(회로가 레벨 트리거이므로) 다시 트리거를 발생시킵니다.

두 플래그는 트리거 순간에 포착된 엔코더 위치가 저장된 레지스터를 읽으면 자동으로 클리어됩니다. **Gate3[j].Chan[j].HomeCapt**. 양쪽의 플래그는 레벨 트리거로 엣지 트리거가 아닙니다. 따라서 플래그가 클리어될 때 선택된 입력이 계속 지정된 트리거 상태이면 즉시 다시 트리거됩니다.

Gate3[j].Chan[j].TrigState 는 풀 워드 요소 **Gate3[j].Chan[j].Status** 의 비트 22 입니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[i].Chan[j].UserFlag 게이트 3 채널의 USER 플래그 입력

설명: USER 플래그 입력값

범위: 0 .. 1

단위: 논리값

Gate3[i].Chan[j].UserFlag에는 최신의 서보 인터럽트에서 래치된 IC 채널의 USER 플래그 입력의 현재 논리 상태를 저장합니다. 1의 값은 IC에 대한 high 전압에 대응합니다. 0의 값은 IC에 대한 low 전압에 대응합니다. 대부분의 Delta Tau 하드웨어에서는 이는 광절연 입력입니다. 그리고 플래그가 "닫힘"일 때 광 아이솔레이터에 전류가 흐르고 0이 됩니다.

Gate3[i].Chan[j].UserFlag는 풀 워드 요소 **Gate3[i].Chan[j].Status**의 비트 11입니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

Gate3[i].Chan[j].UVW 게이트 3 채널의 UVW 플래그 입력

설명: U, V 및 W 플래그 입력값

범위: 0 .. 7

단위: 비트 필드

Gate3[i].Chan[j].UVW에는 최신의 서보 인터럽트에서 래치된 IC 채널의 U, V 및 W 플래그 입력의 현재 논리 상태를 저장합니다. 비트 2(값 4)는 U 플래그의 상태를 나타내며 비트 1(값 2)은 V 플래그의 상태를 나타내고 비트 0(값 1)은 W 플래그의 값을 나타냅니다.

비트가 1인 값은 IC에 대한 high 전압에 대응합니다. 0인 값은 IC에 대한 low 전압에 대응합니다. 대부분의 Delta Tau 하드웨어에서 IC에 대한 외부 버퍼 입력은 비반전입니다. 따라서 이는 컨트롤러에 대한 전압과 일치합니다.

Gate3[i].Chan[j].UVW는 풀 워드 요소 **Gate3[i].Chan[j].Status**의 비트 12 - 14를 구성합니다. C 환경에서는 이는 풀 워드 요소로 액세스되어야 합니다.

모터 상태 데이터 구조체 요소(V1.6 2014.03)

Power PMAC의 모터 상태 데이터 구조체는 어플리케이션 개발이나 디버그에 또는 실제 어플리케이션 실행에서 중요한 많은 요소를 제공합니다. 본 장은 중요한 모터 상태 요소를 기재합니다.

Motor[x].ActiveMasterPos 모터의 누적 위치 추종 거리

설명: 누적된(레이트 제한된) 위치 추종 거리

범위: 부동 소수점

단위: 모터 단위

Motor[x].ActiveMasterPos에는 위치 추종 기능에 의한 실행 중의 누적 추종 거리를 저장합니다. 추종이 속도와 가속도로 제한되고(**Motor[x].MaxSpeed**와 **Motor[x].MaxAccel**의 양쪽이 0.0보다 큰 설정) 그 제한이 발생한 경우 제한이 없는 추종 거리 **Motor[x].MasterPos**와 다른 경우가 있습니다. 이 차이는 추종 기능이 제한되지 않을 때 "릴리스"되는 누적된 "과잉량"입니다.

각 서보 사이클에서 **Motor[x].ActiveMasterPos**의 값은 **Motor[x].MasterPos**의 값에 원래대로 덮어쓰기되므로 **Motor[x].ActiveMasterPos**에 직접 쓸 수 없습니다.

위치 추종이 "오프셋" 모드(**Motor[x].MasterCtrl** 비트 1 = 1)인 경우에 이 요소는 모터의 "기본" 프로그래밍 원점과 실제로 사용되는 "오프셋" 프로그래밍 원점의 차를 저장합니다.

Motor[x].ActiveMasterPosSf 모터 실행 중 위치 추종 계수

설명: 실행 중(슬루 제한) 위치 추종 계수

범위: 부동 소수점

단위: 모터 단위/마스터 단위

Motor[x].ActiveMasterPosSf에는 최신 서보 사이클에 사용된 실제 위치 추종 계수를 저장합니다. 저장 설정 요소 **Motor[x].SlewMasterPosSf**를 0.0보다 크게 설정하여 추종 계수의 슬루 레이트 제어가 유효한 경우에 위치 추종 기능을 유효 또는 무효로 설정한 후 또는 명령 계수 변경 후에는 저장 설정 요소 **Motor[x].MasterPosSf**의 명령 계수와는 다른 값이 됩니다. 위치 추종이 무효일 때 이 값은 0.0이 됩니다.

Motor[x].ActPos 모터의 외측 루프 피드백 위치

설명: 모터 외측 루프 피드백 위치

범위: 부동 소수점

단위: 모터 단위

Motor[x].ActPos에는 외측 서보 루프상의 현재 서보 사이클 Net 피드백 위치를 저장합니다. 이 값은 위치 테이블 보정이나 백래시 보정 등이 추가된 측정 위치 **Motor[x].Pos**에서 구할 수 있습니다. 또한 모터 단위로 환산되어 전원 투입 위치의 기준이 됩니다. 편차량 **Motor[x].PosError**는 **Motor[x].DesPos**에서 이 값을 빼서 구할 수 있습니다.

모터 제로 위치와의 상대적 위치를 계산하려면 이 값보다 **Motor[x].HomePos**를 빼십시오.

Motor[x].ActPos2 모터의 내측 루프 피드백 위치

설명: 모터 내측 루프 피드백 위치

범위: 부동 소수점

단위: 내측 루프 단위

Motor[x].ActPos2에는 내측(보통은 속도) 서보 루프 상의 현재 서보 사이클 Net 피드백 위치를 저장합니다. 이 값은 보정 테이블 등이 추가된 측정 위치 **Motor[x].Pos2**에서 구할 수 있습니다. 또한 내측 서보 루프 단위로 환산되고 전원 투입 위치에 참조됩니다. 반드시 외측 루프 모터 단위와 같지 않아도 됩니다.

Motor[x].ActVel 모터의 실제 속도

설명: 모터 실제 속도

범위: 부동 소수점

단위: 모터 단위/서보 인터럽트 주기

Motor[x].ActVel에는 현재 서보 사이클의 실제 속도를 저장합니다. 이 값은 내측 루프의 현재 서보 사이클과 이전 서보 사이클 상의 실제 위치차로 구할 수 있으며 단위는 모터 단위/서보 인터럽트 주기로 환산됩니다. **Motor[x].Stime**이 0보다 크게 설정되고 매번 서보 루프를 닫지 않고 있을 때에도 이 값은 모든 서보 인터럽트 주기로 계산된다는 점에 주의하십시오. 이 값은 모터의 서보 알고리즘에서 속도와 가속 피드백항의 계산에 사용됩니다.

Motor[x].AmpEna 모터의 앰프 유효 Status 비트

설명: "앰프 유효" Status 비트

범위: 0 .. 1

단위: 논리값

모터가 유효인 경우(클로즈드 루프 또는 오픈 루프 제어 중에서)에 **Motor[x].AmpEna** Status 비트는 1로 설정됩니다. 이 경우에 유효한 앰프 유효 출력 신호가 존재하지 않아도 됩니다. 모터가 무효인 경우 이 비트는 0입니다. 이 Status 비트는 32비트 요소 **Motor[x].Status[0]**의 비트 12에 해당합니다.

이 소프트웨어 Status 비트가 실제로 앰프 유효 출력 신호를 제어하는 하드웨어 출력 비트와는 다르다는 점에 주의하십시오.

Motor[x].AmpFault 모터의 앰프 이상 에러 Status 비트

설명: "앰프 이상 에러" Status 비트

범위: 0 ..1

단위: 논리값

모터가 앰프 이상으로 무효로 설정된 경우에 앰프 이상 신호 상태가 없더라도 Status 비트 **Motor[x].AmpFault** 는 1이 됩니다. 또한 계산된 " I^2T " 적분 전류 이상에서도 1이 됩니다(그 경우에는 비트 21도 설정됩니다). 그 이외에는 0이 됩니다. 모터가 다시 유효가 되면 0이 됩니다. 이 Status 비트는 32비트 요소 **Motor[x].Status[0]**의 비트 24에 해당합니다.

이 소프트웨어 Status 비트가 앰프 이상 입력 신호 상태를 반영하는 하드웨어 입력 비트와는 다르다는 점에 주의하십시오.

Motor[x].AmpWarn 모터의 앰프 경고 Status 비트

설명: "앰프 경고" Status 비트

범위: 0 ..1

단위: 논리값

Motor[x].AmpFaultLevel 비트 1(값 2)이 1로 설정되는 경우에 에러가 발생하려면 앰프 이상 비트의 이상 상태가 2번 연속 읽히고 1번의 읽기라면 **Motor[x].AmpWarn** Status 비트가 1로 설정됩니다. 이 Status 비트는 32비트 요소 **Motor[x].Status[0]**의 비트 19에 해당합니다.

Motor[x].AuxFault 모터의 보조 이상 에러 Status 비트

설명: 보조 이상 에러 Status 비트

범위: 0 ..1

단위: 논리값

저장 설정 요소 **Motor[x].pAuxFault**, **Motor[x].AuxFaultBit**, **Motor[x].AuxFaultLevel** 그리고 **Motor[x].AuxFaultLimit** 에서 지정되는 "보조 이상" 에러 상태를 검출하여 모터가 무효로 설정된 경우에 **Motor[x].AuxFault** Status 비트는 1 이 됩니다. 그 이외에는 0 이 됩니다. 모터가 다시 유효가 되면 0 이 됩니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 17 에 상당합니다.

Motor[x].AuxFaultCount 모터의 보조 이상 검출 스캔 누적수

설명: 보조 이상 검출 스캔의 누적수

범위: 0 .. 255

단위: 스캔

Motor[x].AuxFaultCount 에는 "보조 이상" 상태가 검출된 스캔의 현재 누적수를 저장합니다. 이상 상태가 검출되는 스캔마다 **Motor[x].AuxFaultLimit** 의 이상 최소치를 초과할 때까지 **Motor[x].AuxFaultCount** 는 1 만 더합니다. 이상 상태가 검출되지 않는 스캔마다 **Motor[x].AuxFaultCount** 는 제로가 될 때까지 1 만 더합니다.

보조 이상 입력 비트 레지스터는 **Motor[x].pAuxFault** 에서 지정됩니다. 이 레지스터 내의 비트는 **Motor[x].AuxFaultBit** 에서 지정됩니다. 이 비트의 이상 상태는 **Motor[x].AuxFaultLevel** 에서 지정됩니다.

Motor[x].BICompSize 모터의 백래시 보정 테이블 보정값

설명: 모터 백래시 보정 테이블 보정값

범위: 부동 소수점

단위: 모터 단위

Motor[x].BICompSize 에는 대상 레지스터로 이 요소를 사용하는 모든 보정 테이블에 의한 현재 서보 사이클에서의 Net 백래시 보정값을 저장합니다. 모터가 반대 방향으로 이동한 경우에 이 요소와 저장 설정 요소 **Motor[x].BISize** 의 합에서 도출된 슬루 레이트 제한값은 서보 계산 중에서 사용되는 보정된 값 **Motor[x].ActPos** 를 얻기 위해 원시 측정 위치 **Motor[x].Pos** 에서 빼집니다.

보정 테이블이 이 레지스터를 대상으로 사용하지 않는 경우에 사용자는 오프셋 위치를 생성하기 위해 이 레지스터에 직접 쓸 수 없습니다.

Motor[x].BIDir 모터의 백래시 방향 Status 비트

설명: "백래시 방향" Status 비트

범위: 0 ..1

단위: 논리값

모터의 백래시 기능이 유효이고 모터가 반대 방향으로 실행되고 있거나 마지막 위치 동작이 반대 방향으로 실행되었던 경우에 Status 비트 **Motor[x].BIDir** 은 1 이 됩니다. 그 이외에는 0 입니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[1]**의 비트 28 에 상당합니다.

Motor[x].BlockRequest 모터의 블록 요구 플래그 Status 비트

설명: "블록 요구 플래그" Status 비트

범위: 0 .. 1

단위: 논리값

모터가 새로운 동작 섹션에 들어간 직후에 모션 신호 다음에 오는 동작 섹션을 위한 운동식을 계산하도록 요구하는 경우에 **Motor[x].BlockRequest** Status 비트는 1 로 설정됩니다. 그 이외에는 0 입니다. 주로 내부 용도용입니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 9 에 상당합니다.

Motor[x].CapturedPos 모터의 포착 위치 모니터링

설명: 모터의 포착 위치 모니터링

범위: 부동 소수점

단위: 모터 단위

Motor[x].CapturedPos 에는 감시 위치 포착 기능으로 계산된 최신 모터 위치를 저장합니다. 이 기능은 비저장 설정 요소 **Motor[x].CapturePos** 를 1 로 설정하여 사용됩니다. 포착 현상이 발생하면 Power PMAC 은 자동으로 모터 위치에 센서값을 포착하는 처리를 하고 이 요소에 값을 저장합니다. 이 감시 기능은 어떠한 경우에도 모터 동작에 영향을 미치지 않습니다.

포착 트리거 그리고 후속 위치 처리는 트리거가 있는 동작 기능과 완전히 동일하게 지정됩니다. 이 지정은 "사용자 설명서"의 '특정 모터 동작 실행' 장의 '트리거 모터 동작' 항목에서 자세하게 설명합니다. 감시 위치 포착 기능은 트리거가 있는 동작과 동시에 사용해서는 안 됩니다. 트리거가 있는 동작은 이 요소값에 영향을 미치지 않습니다.

이 위치는 모터 제로 위치가 기준이 되고(반드시 전원 투입 위치가 아닌) 모터에 할당된 축의 어느 프로그램 오프셋에도 영향을 받지 않습니다.

Motor[x].CapturedPos 는 2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새로운 기능입니다.

Motor[x].ClosedLoop 모터의 클로즈드 루프 모드 Status 비트

설명: "클로즈드 루프 모드" Status 비트

범위: 0 ..1

단위: 논리값

모터가 클로즈드 루프 제어 중이면 Status 비트 **Motor[x].ClosedLoop** 는 1 로 설정됩니다. 모터가 오픈 루프 모드인 경우(유효나 무효 모두)는 제로입니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 13에 해당합니다.

Motor[x].CompDac 모터의 토크 보정 출력 보정값

설명: 모터의 ("토크")보정 서보 루프 출력 보정값

범위: 부동 소수점

단위: 16 비트 출력 단위

Motor[x].CompDac에는 대상 레지스터로 이 요소를 사용하는 모든 보정 테이블에 의한 현재 서보 사이클에서의 Net 서보 루프 출력 보정값을 저장합니다. 이 값은 보정하는 값을 얻기 위해 원시 서보 루프 출력 **Motor[x].ServoOut**에 더합니다. 이는 Power PMAC 이 전류(commutation)를 실시하지 않는 경우에 **Motor[x].pDac**으로 지정한 레지스터에 출력되거나 전류(commutation)를 실시한 경우에 토크 명령 **Motor[x].IqCmd**로 전류(commutation) 알고리즘에 입력됩니다. **Motor[x].ServoOut**과 **Motor[x].CompDac**의 합계값의 크기는 저장 설정 요소 **Motor[x].MaxDac**과 비교되고 필요한 경우 그 값으로 제한됩니다.

보정 테이블이 이 레지스터를 대상으로 사용하지 않는 경우에 사용자는 오프셋 위치를 생성하기 위해 이 레지스터에 직접 쓸 수 없습니다.

Motor[x].CompDesPos 모터의 명령 위치 보정 테이블 오프셋

설명: 모터 명령 위치 보정 테이블 오프셋

범위: 부동 소수점

단위: 모터 단위

Motor[x].CompDesPos에는 대상 레지스터로 이 요소를 사용하는 모든 보정 테이블에 의한 현재 서보 사이클의 Net 명령 위치 "보정값"을 저장합니다. 이 값은 서보 계산에 사용되는 Net 명령 위치 **Motor[x].DesPos**를 얻기 위해 궤적 명령 위치 **Motor[x].Desired.Pos**와 마스터 위치 **Motor[x].ActiveMasterPos**에 더합니다.

보정 테이블이 "전자캠" 테이블과 같이 이 모터에서 생성된 동작 명령을 별도의 모터 위치 함수로 사용될 때는 보통 대상 레지스터로 **Motor[x].CompDesPos**가 사용됩니다. 모터의 측정 오차를 수정하기 위해서는 실제로 측정된 위치에 대한 오프셋으로 **Motor[x].CompPos**를 사용하도록 권장합니다.

보정 테이블이 이 레지스터를 대상으로 사용하지 않는 경우에 사용자는 오프셋 위치를 생성하기 위해 이 레지스터에 직접 쓸 수 없습니다.

Motor[x].CompPos 모터의 외측 루프 실제 위치 보정 테이블 보정값

설명: 모터 외측 루프 실제 위치 보정 테이블 보정값

범위: 부동 소수점

단위: 모터 단위

Motor[x].CompPos에는 대상 레지스터로 이 요소를 사용하는 모든 보정 테이블에 의한 현재 서보 사이클의 Net 외측 루프 실제 위치 보정값을 저장합니다. 이 값은 서보 계산에 사용되는 보정값 **Motor[x].ActPos**를 얻기 위해 원시 측정 위치 **Motor[x].Pos**에 더합니다.

보정 테이블이 이 레지스터를 대상으로 사용하지 않는 경우에 사용자는 오프셋 위치를 생성하기 위해 이 레지스터에 직접 쓸 수 없습니다.

Motor[x].CompPos2 모터의 내측 루프 실제 위치 보정 테이블 보정값

설명: 모터 내측 루프 실제 위치 보정 테이블 보정값

범위: 부동 소수점

단위: 내측 루프 단위

Motor[x].CompPos2에는 대상 레지스터로 이 요소를 사용하는 모든 보정 테이블에 의한 현재 서보 사이클의 Net 내측 루프 실제 위치 보정값을 저장합니다. 이 값은 서보 계산에 사용되는 보정값 **Motor[x].ActPo2s**를 얻기 위해 원시 측정 위치 **Motor[x].Pos2**에 더합니다.

이 레지스터를 보정 테이블이 대상으로 사용하지 않는 경우에 사용자는 내측 루프 오프셋을 생성하기 위해 이 레지스터에 직접 쓸 수 없습니다.

Motor[x].Coord 모터의 좌표계 할당

설명: 모터의 좌표계 할당

범위: 0 .. 127

단위: 좌표계 번호

Motor[x].Coord에는 특정 축 정의 구문을 사용하여 모터가 할당되는 좌표계의 번호를 저장합니다. 좌표계에서 모터는 위치 결정축, 명시적으로 또는 인버스 키네마틱스 축으로 또한 스핀들 축 또는 무효 정의로 정의됩니다. 재초기화에서 모터는 모두 좌표계 0의 무효 정의로 자동으로 설정됩니다. 따라서 **Motor[x].Coord**는 재초기화에 따라 모든 모터에서 0을 나타냅니다.

Motor[x].CoordSf[i] 모터의 축 정의 계수

설명: 축 정의 계수

범위: 부동 소수점

단위: 모터 단위/축 단위

Motor[x].CoordSf[i]에는 인덱스 *i* 축과 관련된 모터 *x*의 축 정의 계수를 저장합니다. *i*는 0~32의 범위입니다. 32의 축 명칭에 대한 각각의 인덱스값은 다음 표와 같습니다.

축 명칭	인덱스 <i>i</i>	축 명칭	인덱스 <i>i</i>	축 명칭	인덱스 <i>i</i>	축 명칭	인덱스 <i>i</i>
A	0	Z	8	HH	16	SS	24
B	1	AA	9	LL	17	TT	25
C	2	BB	10	MM	18	UU	26
U	3	CC	11	NN	19	VV	27
V	4	DD	12	OO	20	WW	28
W	5	EE	13	PP	21	XX	29
X	6	FF	14	QQ	22	YY	30
Y	7	GG	15	RR	23	ZZ	31

Motor[x].CoordSf[32]는 "축 오프셋" 값입니다.

이 값은 보통 온라인 축 정의 명령으로 설정됩니다. 예를 들어 축 정의 명령은 다음과 같습니다.

#1 -> 866X + 500Y -333.333

에서는 **Motor[1].CoordSf[6]**을 866으로 설정하고 **Motor[1].CoordSf[7]**을 500으로 설정하고 **Motor[1].CoordSf[32]**를 -333.333으로 설정합니다. 다른 **Motor[1].CoordSf[i]** 요소는 모두 0으로 설정됩니다.

보통 이러한 요소는 사용자를 통한 단순한 읽기값이지만 직접 쓸 수도 있습니다. 값을 변경하는 것이 모터와 축의 관계를 변화시킨다는 것을 명심하십시오. 그 결과 다음에 프로그램된 축 동작 앞에 위치 비교 기능 "pmatch"가 정상적인 관계를 재건하기 위해 실행되어야 합니다. 모션 프로그램이 시작될 때 이는 자동으로 실시됩니다. 그러나 그렇지 않으면 명시적으로 **pmatch** 명령을 실행해야 합니다.

또한 제로 그리고 비제로값 간의 값을 변경하더라도 모터와 축 위치를 적절하게 맞추기 위해 전체의 로직을 변경하므로 가능한 한 사용하지 마십시오.

Motor[x].Csolve 모터의 PMATCH 계산 Status 비트

설명: "PMATCH 계산 중의 모터"의 Status 비트

범위: 0 ..1

단위: 논리값

모터 위치가 좌표계 "pmatch"(위치 비교) 함수 내의 축 위치를 계산하기 위해 (모션 프로그램 시작, **pmatch** 명령 실행, 축 위치/속도/편차 참조에서) 사용되는 경우에 **Status** 비트 **Motor[x].Csolve** 는 1로 설정됩니다. 모터가 "무효" 정의 또는 여유 축 정의 시에는 0으로 설정되고 축 위치 계산에 사용되지 않습니다. 이 **Status** 비트는 32비트 요소 **Motor[x].Status[1]**의 비트 31에 해당합니다.

갠트리 중 어느 모터가 사용될지 변경할 목적으로 사용자는 어느 모터가 "pmatch" 함수 속에서 사용되는지 비트에 쓰기 변경할 수 있습니다. 단, 무효인 함수를 생성할 가능성이 커서 위험하므로 이를 실시할 때는 충분한 주의가 필요합니다.

Motor[x].CurrentNullTimer 모터의 자동 전류 제로 조정 잔여 사이클 수

설명: 자동 전류 제로 조정 잔여 사이클 수

범위: 0 .. 32767

단위: 위상 인터럽트 주기

Motor[x].CurrentNullTimer에는 모터의 자동 전류 제로 조정 중의 잔여 위상 사이클 수를 저장합니다. 첫 자동 제로 조정 처리에서 저장 설정 요소 **Motor[x].CurrentNullPeriod**의 값으로 설정되고 처리가 완료될 때까지 각 위상 사이클에서 1씩 뺍니다. **CurrentNullTimer**가 0보다 큰 시간 동안 실제 전류 루프 제어는 무효가 됩니다. 또한 상태 요소 **Motor[x].IaMeas**와 **Motor[x].IbMeas**에는 현재 사이클값 대신에 처리 동안에 측정된 상전류의 누적합을 저장합니다. **CurrentNullTimer**가 0일 때 자동 제로 조정은 실행되지 않습니다.

Motor[x].CurrentNullTimer는 펌웨어 버전 V1.5의 새로운 기능입니다(2012년 2/4분기 출시). 주된 목적은 상태 요소로 표준 자동 제로 조정의 진행을 추적하는 것이지만 사용자는 표준 명령으로 지원받지 못할 때 자동 제로 조정 처리를 시작하기 위해서 양의 값을 쓸 수 있습니다. 이 경우에 사용자 책임하에 적정 상태의 모터에서 자동 제로 조정이 실시된다는 것을 확인하십시오.

Motor[x].DacLimit 모터의 서보 출력 제한 **Status** 비트

설명: "서보 출력 제한" **Status** 비트

범위: 0 .. 1

단위: 논리값

모터의 서보 출력 명령 현재값이 **Motor[x].MaxDac**의 크기로 포화된 경우에 **Status** 비트 **Motor[x].DacLimit**는 1입니다. 그 이외에는 0입니다. 이 **Status** 비트는 32비트 요소 **Motor[x].Status[1]**의 비트 29에 해당합니다.

Motor[x].DesPos 모터의 Net 명령 위치

설명: 모터의 Net 명령 위치

범위: 부동 소수점

단위: 모터 단위

Motor[x].DesPos에는 현재 서보 사이클의 Net 명령 위치 즉, 계산된 궤적 위치와 위치 추종 기능에 의한 마스터 위치의 합을 저장합니다. 이 값은 모터 단위로 환산되고 전원 투입/리셋 위치 기준이 됩니다. **Motor[x].PosError**의 편차량을 얻기 위해 이 값에서 **Motor[x].ActPos**의 값을 뺍니다.

모터 제로 위치에서 기준 위치를 계산하려면 **Motor[x].HomePos**의 값을 빼십시오.

Motor[x].DesVel 모터의 Net 명령 속도

설명: 모터의 Net 명령 속도

범위: 부동 소수점

단위: 모터 단위/서보 인터럽트 주기

Motor[x].DesVel에는 현재 서보 사이클의 Net 명령 속도값 즉, 해당 서보 사이클의 명령 위치와 이전 서보 사이클의 명령 위치와의 차를 저장합니다. 이 값은 모터 단위/서보 인터럽트 주기로 환산됩니다. **Motor[x].Stime**이 0보다 크게 설정되고 매 서보 사이클에서 루프가 닫히지 않았을 때에도 이 값은 모든 서보 인터럽트 주기로 계산된다는 점에 주의하십시오. 또한 모터의 서보 알고리즘에서 속도, 가속과 마찰 Feed-forward 항의 계산에 사용됩니다.

Motor[x].DesVelZero 모터의 명령 속도 제로 Status 비트

설명: "명령 속도 제로" Status 비트

범위: 0 .. 1

단위: 논리값

모터가 클로즈드 루프 제어 중에 Net 명령 속도가 정확하게 제로인 경우(즉 그 위치를 유지) 또는 오픈 루프 모드에서 실제 속도가 정확하게 제로인 경우에 Status 비트 **Motor[x].DesVelZero**는 1로 설정됩니다. 모터가 제로 명령 속도가 아닌 클로즈드 루프 모드인 경우 또는 실제 속도가 제로가 아니고 오픈 루프 모드인 경우(유·무효 모두)에 Status 비트는 제로입니다. 이 Status 비트는 32비트 요소 **Motor[x].Status[0]**의 비트 14에 해당합니다.

Motor[x].EncLoss 모터의 센서 소실 에러 비트

설명: "센서 소실" 에러 Status 비트

범위: 0 .. 1

단위: 논리값

저장 설정 요소 **Motor[x].pEncLoss**, **Motor[x].EncLossBit**, **Motor[x].EncLossLevel** 그리고 **Motor[x].EncLossLimit** 설정을 통해 "엔코더 소실" 상태를 검출하고 모터가 무효가 된 경우 Status 비트 **Motor[x].EncLoss** 가 1 입니다. 그 이외에는 0 이 됩니다. 모터가 다시 유효가 되면 0 이 됩니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 18 에 상당합니다.

Motor[x].EncLossCount 모터의 센서 소실 검출 누적 스캔 수

설명: 센서 소실을 검출한 누적 스캔 수

범위: 0 .. 255

단위: 스캔

Motor[x].EncLossCount 에는 "엔코더 소실" 상태가 검출된 스캔의 현재 누적수를 저장합니다. 소실 상태가 검출되는 스캔마다 **Motor[x].EncLossLimit** 의 이상 최소치를 초과할 때까지 **Motor[x].EncLossCount** 는 1 만 더합니다. 소실 상태가 검출되지 않는 스캔마다 **Motor[x].EncLossCount** 는 제로에 도달할 때까지 1 만 뺍니다.

엔코더 소실 비트 레지스터는 **Motor[x].pEncLoss** 로 지정됩니다. 이 레지스터 내의 비트는 **Motor[x].EncLossBit** 에서 지정됩니다. 이 비트의 소실 상태는 **Motor[x].EncLossLevel** 로 지정됩니다.

Motor[x].FeFatal 모터의 편차 이상 Status 비트

설명: "편차 이상" Status 비트

범위: 0 .. 1

단위: 논리값

모터의 편차량이 **Motor[x].FatalFeLimit** 의 편차 이상 제한을 초과하여 모터가 무효가 된 경우에 Status 비트 **Motor[x].FeFatal** 은 1 이 됩니다. 그 이외에는 0 이 됩니다. 모터가 다시 유효가 되면 0 이 됩니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 26 에 상당합니다.

Motor[x].FeWarn 모터의 경고 위치 편차 Status 비트

설명: "경고 위치 편차" Status 비트

범위: 0 .. 1

단위: 논리값

모터의 편차량이 **Motor[x].WarnFeLimit** 의 경고 편차 제한을 초과한 경우에 Status 비트 **Motor[x].FeWarn** 은 1 이 됩니다. 편차량이 이 제한 미만이거나 모터가 편차 이상 제한을 초과하여 무효가 된 경우에 Status 비트는 0 이 됩니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 27 에 해당합니다.

Motor[x].FiltrMasterPos[i] 모터의 마스터 위치 이력 배열 요소

설명: 모터의 마스터 위치 이력 배열 요소

범위: 부동 소수점

단위: 모터 단위

Motor[x].FiltrMasterPos[i]에는 과거 16 회의 서보 사이클 중 1 개의 마스터 위치를 저장합니다. 각 서보 사이클에서 Power PMAC 은 요소 **Motor[x].FiltrMasterPos[i]**에 **Motor[x].ActiveMasterPos** 를 복사합니다. 이때 *i* 의 값은 서보 사이클 카운터 **Sys.ServoCount** 의 하위 4 비트와 같고 따라서 *i* 는 0~15 까지의 값이 됩니다.

Motor[x].FiltrMasterPos[i]는 모터 단위로 환산되고 전원 투입 시 마스터 위치 기준이 됩니다. 또한 통지 목적을 위해 필터링된 마스터 속도 **Motor[x].FiltrMasterVel** 의 계산에 사용됩니다.

Motor[x].FiltrMasterVel 모터의 마스터 필터 속도

설명: 모터 마스터 필터 속도

범위: 부동 소수점

단위: 모터 단위/16 서보 인터럽트 주기

Motor[x].FiltrMasterVel 에는 최신 서보 사이클에서 "필터링된" 마스터 속도를 저장합니다. 각 서보 사이클에서 Power PMAC 은 **Motor[x].FiltrMasterVel** 을 산출하기 위해 현재 사이클의 마스터 위치(**Motor[x].MasterPos**)에서 사전에 저장된 16 서보 사이클만큼의 마스터 위치(**Motor[x].FiltrMasterPos[i]**)를 뺍니다. **Motor[x].Stime** 이 0 보다 크게 설정되고 매 서보 사이클에서 루프가 닫히지 않았을 때에도 이 값은 모든 서보 인터럽트 주기로 계산된다는 점에 주의하십시오.

Motor[x].FltrPos[i] 모터의 외측 루프 피드백 위치 이력 배열 요소

설명: 모터 외측 루프 피드백 위치 이력 배열 요소

범위: 부동 소수점

단위: 모터 단위

Motor[x].FltrPos[i]에는 과거 16 회의 서보 사이클 중 1 개인 외측 서보 루프의 Net 피드백 위치를 저장합니다. 각 서보 사이클에서 Power PMAC 은 요소 **Motor[x].FltrPos[i]**에 **Motor[x].ActPos**를 복사합니다. 이때 *i*의 값은 서보 사이클 카운터 **Sys.ServoCount**의 하위 4비트와 같고 따라서 *i*는 0~15까지의 값이 됩니다.

Motor[x].FltrPos[i]는 모터 단위로 환산되고 전원 투입 위치 기준이 됩니다. 또한 통지 목적을 위해 필터링된 속도 **Motor[x].FltrVel**의 계산에 사용됩니다. 모터 제로 위치에서 기준 위치를 계산하려면 **Motor[x].HomePos**의 값을 빼십시오.

Motor[x].FltrPos2[i] 모터의 내측 루프 피드백 위치 이력 배열 요소

설명: 모터 내측 루프 피드백 위치 이력 배열 요소

범위: 부동 소수점

단위: 내측 루프 단위

Motor[x].FltrPos2[i]에는 과거 16 회의 서보 사이클 중 1 개인 내측 서보 루프의 Net 피드백 위치를 저장합니다. 각 서보 사이클에서 Power PMAC 은 요소 **Motor[x].FltrPos2[i]**에 **Motor[x].ActPos2**를 복사합니다. 이때 *i*의 값은 서보 사이클 카운터 **Sys.ServoCount**의 하위 4비트와 같고 따라서 *i*는 0~15의 값이 됩니다.

Motor[x].FltrPos2[i]는 내측 루프 단위로 환산됩니다. 이 값은 반드시 외측 루프에서 사용되는 모터 단위 또는 전원 투입 위치 기준과 동일하게 만들지 않아도 됩니다. 또한 통지 목적으로 필터링된 속도 **Motor[x].FltrVel2**의 계산에 사용됩니다.

Motor[x].FltrVel 모터의 외측 루프 피드백 필터 속도

설명: 모터 외측 루프 피드백 필터 속도

범위: 부동 소수점

단위: 모터 단위/16 서보 인터럽트 주기

Motor[x].FltrVel에는 최신 서보 사이클의 외측 서보 루프상의 "필터" 속도를 저장합니다. 각 서보 사이클에서 Power PMAC 은 **Motor[x].FltrVel**을 산출하기 위해 현재 사이클 위치(**Motor[x].ActPos**)에서 사전에 저장된 16 서보 사이클만큼의 실제 위치(**Motor[x].FltrPos[i]**)를 뺍니다. **Motor[x].Stime**이 0보다 크게 설정되고 매 서보 사이클에서 루프가 닫히지 않았을 때에도 이 값은 모든 서보 인터럽트 주기로 계산된다는 점에 주의하십시오.

Motor[x].FiltrVel 은 온라인 **v** 명령 또는 버퍼를 이용한 **vread** 명령용의 통지 속도를 계산하기 위해 사용됩니다. 통지되는 값은 모터 단위/msec 에 재스케일링됩니다. **Motor[x].FiltrVel** 의 값을 이러한 단위로 변환하려면 16 과 **Sys.ServoPeriod** 로 나누십시오.

Motor[x].FiltrVel2 모터의 내측 루프 피드백 필터 속도

설명: 모터 내측 루프 피드백 필터 속도

범위: 부동 소수점

단위: 내측 루프 단위/16 서보 인터럽트 주기

Motor[x].FiltrVel2에는 최신 서보 사이클 내측 서보 루프상의 "필터" 속도를 저장합니다. 각 서보 사이클에서 Power PMAC 은 **Motor[x].FiltrVel2** 를 산출하기 위해 현재 사이클 위치(**Motor[x].ActPos2**)에서 사전에 저장된 16 서보 사이클만큼의 실제 위치(**Motor[x].FiltrPos2[i]**)를 뺍니다. **Motor[x].Stime** 이 0 보다 크게 설정되고 매 서보 사이클에서 루프가 닫히지 않았을 때에도 이 값은 모든 서보 인터럽트 주기로 계산된다는 점에 주의하십시오.

Motor[x].GantryHomed 모터의 갠트리 원점 검출 완료 Status 비트

설명: "갠트리 원점 검출 완료" Status 비트

범위: 0 .. 1

단위: 논리값

이 모터가 주축/종축 갠트리 시스템 상의 주축 모터에 해당하고 주축과 해당 종축 모터의 원점 트리거가 검출된 상태이며 해당 종축 모터와 주축 사이에 전원 투입 시에 경사가 완전히 제거된 경우에 Status 비트 **Motor[x].GantryHomed** 는 1 로 설정됩니다. 그 이외에는 0 입니다. 이 Status 비트는 32 비트의 요소 **Motor[x].Status[0]**의 비트 6 에 해당합니다.

Motor[x].HomeComplete 모터의 기준 위치 확립 Status 비트

설명: "기준 위치 확립" Status 비트

범위: 0 .. 1

단위: 논리값

원점 서치 동작 또는 절대 위치 읽기 중 어느 하나에서 모터의 기준 위치가 확립된 경우에 Status 비트 **Motor[x].HomeComplete** 는 1 로 설정됩니다. 이 값은 전원 투입/리셋 그리고 첫 원점 서치 동작 시에 자동으로 0으로 설정됩니다. 원점 서치 동작에서는 미리 지정된 트리거 상태가 검출되었을 때 트리거 검출 후 동작 부분이 완료되기 전에 1 로 설정됩니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 15에 해당합니다.

Motor[x].HomeInProgress 모터의 원점 서치 동작 실행 중 Status 비트

설명: "원점 서치 동작 실행 중" Status 비트

범위: 0 .. 1

단위: 논리값

Status 비트 **Motor[x].HomeInProgress** 는 첫 원점 서치 동작에 1로 설정됩니다. 미리 지정된 트리거 상태가 검출되고 트리거 검출 후 동작이 시작될 때 또는 트리거 검출 없이 동작이 종료되었을 때 0으로 설정됩니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 30에 해당합니다.

Motor[x].HomePos 모터의 기준 위치

설명: 모터 기준 위치

범위: 부동 소수점

단위: 모터 단위

Motor[x].HomePos에는 전원 투입/리셋 위치의 상대적인 모터 제로 위치를 저장합니다. 이 값은 모터 원점 서치 동작 또는 절대값 위치 읽기 동안에 설정됩니다. 많은 모터 위치 요소가 전원 투입/리셋 위치 기준이므로 모터 제로 위치 기준 위치를 구하려면 해당 값에서 이 요소의 값을 빼서 구합니다.

보통 읽기 전용 상태 요소로 간주되지만 사용자 정의 위치 기준 알고리즘으로 이용할 수 있도록 이 요소에 쓰는 것이 허용됩니다.

Motor[x].I2tFault 모터의 적분 전류 이상 Status 비트

설명: "적분 전류 (I^2T) 이상" Status 비트

범위: 0 .. 1

단위: 논리값

Motor[x].I2tTrip의 적분 전류 제한을 초과하여 모터가 무효로 설정될 때 Status 비트 **Motor[x].I2tFault**는 1로 설정됩니다. 앰프 이상 Status 비트(**Motor[x].AmpFault**)도 이때 설정됩니다. 모터가 다시 유효로 설정될 때 0으로 복귀하고 다른 모든 경우에는 0이 됩니다(앰프 자체가 적분 전류 이상을 검출하는 경우에 이 비트는 세트되지 않는다는 점에 주의하십시오). 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 21에 해당합니다.

Motor[x].I2tSum 모터의 현재 적분 전류값

설명:	현재 적분 전류값
범위:	음이 아닌 부동 소수점
단위:	(16 비트 DAC/ADC 단위) ² * sec

Motor[x].I2tSum에는 Power PMAC의 I²T 적분 전류 제한 기능에 의한 현재 적분 전류값을 저장합니다. 저장 설정 요소 **Motor[x].I2tTrip**의 최소치를 초과하는 경우에 I²T 이상이 발생하여 모터를 kill 하고 모터 Status 비트 **Motor[x].I2tFault**를 1로 설정합니다.

현재 전류 레벨이 저장 요소 **Motor[x].I2tSet** 이상인 경우에 이 전류 레벨의 제공에서 **Motor[x].I2tSet**의 제공을 빼고 스캔 시간을 곱한 결과가 **Motor[x].I2tSum**에 더해집니다(제한을 초과하여 모터 이상이 발생할 때까지). 현재 전류 레벨이 저장 요소 **Motor[x].I2tSet** 미만인 경우에 이 전류 레벨² - **Motor[x].I2tSet**² × 스캔 시간의 결과가 **Motor[x].I2tSum**에서 빼집니다(제로에 도달할 때까지).

Motor[x].IaMeas 모터의 A(U)상 측정 전류

설명:	모터 A(U)상 측정 전류
범위:	-32,768.0..32767.999
단위:	16 비트 입력 단위

Motor[x].IaMeas에는 모터의 A(U)상의 현재 실제 전류값을 저장합니다. 디지털 전류 루프 제어가 유효인 경우에 한해 저장 설정 요소 **Motor[x].IaBias**의 오프셋이 저장 설정 요소 **Motor[x].pAdc**으로 지정한 레지스터에 더해서 전류값을 구할 수 있습니다. 다른 분해능의 ADC가 사용되더라도 단위는 16 비트 ADC 단위입니다.

Motor[x].IaVolts 모터의 A(U)상 출력 명령

설명:	모터 A(U)상 출력 명령
범위:	-32,768.0..32767.999
단위:	출력 단위

디지털 전류 루프 제어가 유효나 무효인 경우 모두 **Motor[x].IaVolts**에는 모터의 A(U)상의 현재 명령 출력값을 저장합니다. 이 값은 저장 설정 요소 **Motor[x].PwmSf**에서 사전에 환산됩니다. 디지털 전류 루프 제어가 유효인 경우에 비저장 설정 요소 **Motor[x].VaBias**의 오프셋이 추가됩니다. 값은 정수로 변환되고 **Motor[x].pDac**으로 지정한 레지스터에 쓰여집니다.

Motor[x].lbMeas 모터의 B(V)상 측정 전류

설명: 모터 B(V)상 측정 전류

범위: -32,768.0..32767.999

단위: 16 비트 입력 단위

Motor[x].lbMeas에는 모터의 B(V)상의 현재 실제 전류값을 저장합니다. 디지털 전류 루프 제어가 유효인 경우에 한해 저장 설정 요소 **Motor[x].lbBias**의 오프셋이 저장 설정 요소 **Motor[x].pAdc**으로 지정한 레지스터에 더해서 전류값을 구할 수 있습니다. 다른 분해능의 ADC가 사용되더라도 단위는 16 비트 ADC 단위입니다.

Motor[x].lbVolts 모터의 B(V)상 출력 명령

설명: 모터 B(V)상 출력 명령

범위: -32,768.0 .. 32767.999

단위: 출력 단위

디지털 전류 루프 제어의 유·무효 모두 **Motor[x].lbVolts**에는 모터 B(V)상의 현재 명령 출력값을 저장합니다. 이 값은 저장 설정 요소 **Motor[x].PwmSf**가 사전에 스케일링합니다. 디지털 전류 루프 제어가 유효인 경우에 비저장 설정 요소 **Motor[x].VbBias**의 오프셋이 추가됩니다. 값은 정수로 변환되고 **Motor[x].pDac**으로 지정한 주소의 1개 뒤의 레지스터에 쓰입니다.

Motor[x].lcVolts 모터의 C(W)상 출력 명령

설명: 모터 C(W)상 출력 명령

범위: -32,768.0 .. 32767.999

단위: 출력 단위

Motor[x].lcVolts에는 모터 C(W)상의 현재 명령 출력값을 저장합니다. 이 값은 디지털 전류 루프 제어가 유효인 경우에 한해 계산됩니다. 또한 이 값은 저장 설정 요소 **Motor[x].PwmSf**에서 사전에 스케일링됩니다. 값은 정수로 변환되고 **Motor[x].pDac**으로 지정한 주소의 2개 뒤의 레지스터에 쓰입니다.

Motor[x].ldMeas 모터의 자화 전류 측정값

설명: 모터의 다이렉트(자화) 전류 측정값

범위: -32,768.0..32767.999

단위: 16 비트 입력 단위

Motor[x].IdMeas에는 전류(commutation) 알고리즘상의 현재의 실제 "자화" 전류 입력을 저장합니다. 디지털 전류 루프 제어가 유효인 경우에 한해 계산됩니다. 로터 자기장 각도 **Motor[x].PhasePos**에 따라 상전류 측정값 **Motor[x].IaMeas**과 **Motor[x].IbMeas**에서 도출됩니다. 이는 전류 루프를 구동하기 위해 **Motor[x].IdCmd**의 명령 자화 전류값과 비교됩니다. 다른 분해능의 ADC가 사용되더라도 단위는 16비트 ADC 단위입니다.

Motor[x].IdInt

설명: 모터의 다이렉트(자화) 전류 적분기 출력

범위: -32,768.0..32767.999

단위: 16비트의 단위

Motor[x].IdInt에는 전류(commutation) 알고리즘의 "자화" 전류 적산기의 현재 출력값을 저장합니다. 디지털 전류 루프 동작이 유효인 경우에 한해 계산됩니다. 각 위상 사이클에서 자화 전류 오차는 전류 적분 게인항 **Motor[x].IIGain**으로 곱하고 그리고 결과는 전 사이클의 요소값을 더합니다. 다음에 루프 사이클 상의 Net 자화 출력 명령 **Motor[x].IdVolts**를 얻기 위해 적산값은 비례 게인항에 의한 출력값에 더합니다.

Motor[x].IdVolts 모터의 다이렉트 전류 루프 출력

설명: 모터의 다이렉트(자화) 전류 루프 출력

범위: -32,768.0 .. 32767.999

단위: 16비트 입력 단위

Motor[x].IdVolts에는 전류(commutation) 알고리즘의 "자화" 전류 루프의 현재 출력값을 저장합니다. 디지털 전류 루프 동작이 유효인 경우에 한해 계산됩니다. 각 위상 사이클에서 전류 루프 비례와 적분 게인항은 이 Net 출력항을 얻기 위해 계산되어 더해집니다. 다음에 각각의 상출력 전압을 산출하기 위해 이 값은 **Motor[x].IqVolts**와 로터 자계 각도 **Motor[x].PhasePos**와 조합되어 사용됩니다.

Motor[x].Imag 모터의 로터 자계 추정 전류

설명: 모터 로터 자계(자화) 추정 전류

범위: -32,768.0 .. 32767.999

단위: 16비트 단위

Motor[x].Imag에는 비동기(유도) 모터용의 현재 추정 로터 자화 전류를 저장합니다. 이는 저장 설정 요소 **Motor[x].DtOverRotorTc**의 로터 시상수로 필터링되고 자화 전류 명령 **Motor[x].IdCmd**에서 추정됩니다. 이 값은 모터 슬립 주파수 계산에 사용됩니다.

Motor[x].InPos 모터의 인포지션 Status 비트

설명: "인포지션" Status 비트

범위: 0 .. 1

단위: 논리값

다음의 모든 "인포지션" 상태가 충족될 때 Status 비트 **Motor[x].InPos** 는 1 로 설정됩니다.(모터가 클로즈드 루프, 명령 속도 제로, 동작 타이머가 실행 중이지 않음(동작, 드웰 또는 딜레이가 실행되지 않음), 편차량이 **Motor[x].InPosBand** 이하 그리고 이 모든 상태가 (**Motor[x].InPosTime** - 1)서보 사이클 동안 연속적으로 참일 것) 이 이외에서는 0 입니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 11 에 해당합니다.

Motor[x].InPosTimer 모터의 연속 인포지션 스캔 카운터

설명: 연속 인포지션 스캔 카운터

범위: 음이 아닌 정수

단위: 서보 사이클

Motor[x].InPosTimer 에는 모터가 "인포지션"으로 판단되는 데에 필요한 순간적인 "인포지션" 조건을 충족하는 연속되는 서보 사이클 수를 저장합니다. 각 서보 사이클에서 Power PMAC 은 모터가 인포지션 기능과 관련된 4 개의 조건에 해당하는지를 판정합니다.

- 1.) 모터는 클로즈드 루프 제어 일 것
- 2.) 모터 명령 속도가 제로일 것
- 3.) 모터가 이동 또는 시간 설정된 드웰을 실행하지 않을 것
- 4.) 편차량이 **Motor[x].InPosBand** 이하일 것

이 조건 4 개에 모두 해당할 때 Power PMAC 은 0 까지 **Motor[x].InPosTimer** 를 뺍니다. 또는 이미 0 인 경우 Power PMAC 은 모터 Status 비트 **Motor[x].InPos** 를 1 로 설정하고 모터가 다음에 계속되는 동작을 위해 "인포지션" 상태로 간주합니다.

이러한 4 개의 조건 중 하나라도 해당하지 않는 경우에 Power PMAC 은 **Motor[x].InPosTimer** 를 저장 설정 요소 **Motor[x].InPosTime** 로 설정하고 모터 Status 비트 **Motor[x].InPos** 를 0 으로 설정합니다.

Motor[x].IqCmd 모터의 직교 전류 명령

설명: 모터 직교(토크) 전류 명령

범위: -32,768.0 .. 32767.999

단위: 16 비트 출력 단위

Motor[x].IqCmd에는 전류(commutation) 알고리즘에서 사용하는 현재 "직교 방향" 전류 명령 입력을 저장합니다. 이 값은 모터의 서보 알고리즘에서 출력되고 토크 명령값을 나타냅니다. 디지털 전류 루프 제어가 유효인 경우에 전류 루프를 구동하기 위해 실제 직교 전류값 **Motor[x].IqMeas**와 비교됩니다.

Motor[x].IqInt 모터의 직교 전류 적분 출력

설명: 모터 직교(토크) 전류 적분 출력

범위: -32,768.0 .. 32767.999

단위: 16 비트 단위

Motor[x].IqInt에는 전류(commutation) 알고리즘에서 사용하는 "직교 방향" 전류 적분의 현재 출력값을 저장합니다. 디지털 전류 루프 제어가 유효인 경우에 한해 계산됩니다. 각 위상 사이클에서 직교 전류 오차는 전류 적분 게인항 **Motor[x].IiGain**으로 곱해지고 그 결과는 이전 사이클의 요소값에 더합니다. 다음에 사이클루프상의 **Net** 직교 출력 명령 **Motor[x].IqVolts**를 얻기 위해 적산값은 비례 게인항에 의한 출력값에 더합니다.

Motor[x].IqMeas 모터의 직교 전류 측정값

설명: 모터 직교(토크) 전류 측정값

범위: -32,768.0 .. 32767.999

단위: 16 비트 입력 단위

Motor[x].IqMeas에는 전류(commutation) 알고리즘에서 사용하는 현재의 실제 "직교 방향" 전류 입력값을 저장합니다. 디지털 전류 루프 제어가 유효인 경우에 한해 계산됩니다. 이 값은 로터 자계 각도 **Motor[x].PhasePos**에 따라 상전류 측정 **Motor[x].IaMeas**와 **Motor[x].IbMeas**에서 산출됩니다. 다른 분해능의 ADC가 사용되더라도 단위는 16 비트 ADC 단위입니다.

Motor[x].IqVolts 모터의 직교 전류 루프 출력

설명: 모터 직교(토크) 전류 루프 출력

범위: -32,768.0..32767.999

단위: 16 비트 입력 단위

Motor[x].IqVolts에는 전류(commutation) 알고리즘에서 사용하는 "직교 방향" 전류 루프의 현재 출력값을 저장합니다. 디지털 전류 루프 제어가 유효인 경우에 한해 계산됩니다. 각 위상 사이클에서 전류 루프 비례와 적분 게인항은 이 **Net** 출력항을 얻기 위해 계산되어 더해집니다. 다음에 각각의 위상 출력 전압을 산출하기 위해 이 값은 **Motor[x].IdVolts** 그리고 로터 자계 각도 **Motor[x].PhasePos**와 조합하여 사용됩니다.

Motor[x].JogPos 모터의 JOG 종료 명령 위치

설명: 모터 JOG 종료 명령 위치

범위: 부동 소수점

단위: 모터 단위

Motor[x].JogPos에는 현재 또는 마지막에 실행된 "유한" JOG 동작의 종료 위치를 저장합니다. 이 값은 모터 단위로 환산되고 전원 투입/리셋 위치 기준이 됩니다.

모터 제로 위치의 상대적 위치를 계산하기 위해서는 **Motor[x].HomePos**의 값을 빼십시오.

Motor[x].JogVel 모터의 도달 명령 속도 절대값

설명: 모터 동작의 도달 명령 속도 절대값

범위: 양의 부동 소수점

단위: 모터 단위

Motor[x].JogVel에는 현재 또는 마지막에 실행된 모터 JOG 또는 원점 서치 동작의 도달 명령 속도 절대값을 저장합니다. 이 값은 모터 단위/msec로 환산됩니다.

Motor[x].LimitStop 모터의 하드웨어 리미트 정지 Status 비트

설명: "하드웨어 리미트 정지" Status 비트

범위: 0 .. 1

단위: 논리값

양 또는 음 방향의 하드웨어 오버트래블 리미트가 ON 되어 모터가 정지했거나 정지 중이면 그 리미트에 현재 없더라도 Status 비트 **Motor[x].LimitStop**은 1입니다. 이는 리미트에서 빠져 나온 경우를 제외하고 리미트 위치에 있을 때를 포함한 모든 경우에는 0이 됩니다. 이 Status 비트는 32비트 요소 **Motor[x].Status[0]**의 비트 25에 해당합니다.

Motor[x].MasterPos 모터의 무제한 위치 추종 누적 거리

설명: 제한이 없는 위치 추종 누적 거리

범위: 부동 소수점

단위: 모터 단위

Motor[x].MasterPos에는 모터의 위치 추종 기능에 의한 누적된 제한이 없는 추종 거리를 저장합니다. 추종이 속도와 가속도로 제한되어(**Motor[x].MaxSpeed**와 **Motor[x].MaxAccel** 모두 0.0 보다 큰 설정) 제한이 발생하면 실행 추종 거리 **Motor[x].ActiveMasterPos**와 다른 경우가 있습니다. 이 차이는 추종 기능이 제한되지 않을 때 "릴리스"되는 누적된 "초과량"입니다.

추종이 속도로 제한되지만 가속 제한이 되지 않은 경우(**Motor[x].MaxSpeed**가 0.0 보다 크고 **Motor[x].MaxAccel**이 0.0 과 같음) 속도 제한이 발생할 때 "초과량"은 누적되지 않습니다. 또한 이 값은 실행 추종 거리 **Motor[x].ActiveMasterPos**와 같아집니다.

보통 **Motor[x].MasterPos**는 어플리케이션에서는 (진단 목적만을 위해)읽기 전용이지만 보통 모터를 사용자 정의로 추종, 여기, 오프셋할 목적으로 알고리즘으로 쓸 수도 있습니다.

단, 모터가 갠트리 '종축'(**Motor[x].ServoCtrl** = 8)으로 설정되는 경우에 **Motor[x].MasterPos**는 이 갠트리 종축 모터와 주축 모터 간의 "경사 보정" 원점 오프셋을 저장합니다. 이 경우에 사용자가 쓴 임의의 값은 자동 알고리즘을 통해 즉시 덮어씹힙니다.

Motor[x].MinusLimit 모터의 하드웨어 마이너스 리미트 Status 비트

설명: "하드웨어 마이너스 리미트 세트" Status 비트

범위: 0 .. 1

단위: 논리값

모터가 현재 마이너스 하드웨어 리미트 상에 있을 때 Status 비트 **Motor[x].MinusLimit**는 1로 설정됩니다. 모터가 이 리미트를 ON 하여 정지했더라도 리미트 상에 없는 경우에는 0이 됩니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 29에 해당합니다.

리미트 스위치를 위한 하드웨어 입력 비트(보통 **Gaten[j].Chan[j].Mlim**)와 이 소프트웨어 Status 비트는 다르다는 점에 주의하십시오.

Motor[x].MoveDesPos 모터의 프로그램 동작 종료 명령 위치

설명: 모터 프로그램 동작 종료 명령 위치

범위: 부동 소수점

단위: 모터 단위

Motor[x].MoveDesPos에는 현재 또는 마지막에 실행한 프로그램 동작의 종료 위치를 저장합니다. 또는 세그먼트 동작인 경우에 현재 또는 마지막에 실행한 동작 세그먼트의 종료 위치를 저장합니다. 이 값은 모터 단위로 환산되고 전원 투입/리셋 위치의 기준이 됩니다.

모터 제로 위치의 상대적 위치를 산출하려면 **Motor[x].HomePos**의 값을 빼십시오.

Motor[x].MoveTimer 모터의 현재 섹션 실행 경과 모터 시간

설명: 현재 섹션 또는 세그먼트 실행 경과 모터 시간

범위: 부동 소수점

단위: msec

Motor[x].MoveTimer에는 모터 상의 모션의 현재 실행 중인 섹션 또는 세그먼트의 시작 경과 시간(+100% 시간 베이스)을 저장합니다. 각 서보 사이클에서 이 값은 모터가 할당되는 좌표계가 가진 **Coord[x].TimeBase**의 현재값으로 더합니다. 이 섹션 또는 세그먼트의 총 시간은 **Motor[x].Desired.Time**에 저장됩니다.

Motor[x].PhaseFound 모터의 위상 원점 확립 Status 비트

설명: "위상 원점 확립" Status 비트

범위: 0 .. 1

단위: 논리값

Motor[x].PhaseFound Status 비트는 Power PMAC에서 동기하는 (**Motor[x].DtOverRotorTc** = 0.0) 전류(commutation) 모터(**Motor[x].PhaseCtrl** 비트 0 또는 비트 2 = 1)의 전원 투입/리셋으로 0으로 설정됩니다. 위상기준 위치가 위상 서치 동작 또는 절대값 위치 읽기 중 어느 하나에서 확립되는 경우에 1로 설정됩니다. 이 Status 비트는 32비트 요소 **Motor[x].Status[0]**의 비트 8에 해당합니다.

Motor[x].PhasePos 모터의 전류(commutation) 현재 위상각

설명: 모터 전류(commutation) 현재 위상각

범위: 0 .. 2047.999

단위: 전류(commutation) 각도 단위

Motor[x].PhasePos에는 전류(commutation) 사이클의 1/2048 단위로 현재 전류(commutation) 각도를 저장합니다. 가장 가까운 정수값이 저장 설정 요소 **Motor[x].pSineTable**과 **Motor[x].pVoltSineTable**에서 지정되는 "사인 함수의 룩업 테이블" 내의 등록을 선택하기 위해 사용됩니다.

보통 읽기 전용의 상태 요소로 간주되지만 위상 참조 사용자 정의 알고리즘으로 사용할 목적으로 이 요소에 쓰는 것이 허용됩니다.



경고

실제 전류(commutation) 각도와 크게 다른 값을 **Motor[x].PhasePos** 요소에 값을 쓰면 잘못된 방향으로 토크나 힘을 가할 수 있는 가능성이 있고 불안정한 양의 피드백이 되어 위험한 폭주 상태에 이를 수 있습니다.

Motor[x].PhaseVoltOffset 모터의 전류(commutation) 위상 전진 오프셋

설명: 모터 전류(commutation) 위상 전진 오프셋

범위: -512 .. 512

단위: 전류(commutation) 각도 단위

Motor[x].PhaseVoltOffset에는 속도에 좌우되는 "위상 전진"에 의한 현재 전류(commutation) 각도 오프셋을 저장합니다. 이는 현재 모터 속도 그리고 저장 설정 요소 **Motor[x].AdvGain**에서 산출됩니다.

Motor[x].PlusLimit 모터의 하드웨어 플러스 리미트 Status 비트

설명: "하드웨어 플러스 리미트 세트" Status 비트

범위: 0 .. 1

단위: 논리값

모터가 양의 하드웨어 리미트 상에 있을 때 Status 비트 **Motor[x].PlusLimit**는 1로 설정됩니다. 모터가 이 리미트를 ON하여 정지했더라도 리미트 상에 없는 경우에는 0이 됩니다. 이 Status 비트는 32비트 요소 **Motor[x].Status[0]**의 비트 28에 해당합니다.

리미트 스위치를 위한 하드웨어 입력 비트(보통 **Gaten[j].Chan[j].Plim**)와 이 소프트웨어 Status 비트는 다르다는 점에 주의하십시오.

Motor[x].Pos 모터의 미처리 외측 루프 실제 위치

설명: 모터의 미처리 외측 루프 실제 위치

범위: 부동 소수점

단위: 모터 단위

Motor[x].Pos에는 백래시 보정이나 위치 테이블 보정과 같은 오프셋을 더하는 이전의 원시 외측 루프 실제 위치를 저장합니다. 덧셈 후의 값 **Motor[x].ActPos**는 서보 루프에서 편차를 얻기 위해 명령 위치와 비교됩니다. 각 사이클에서 새로운 사이클의 "원시" 모터 위치를 얻기 위해 **Motor[x].pEnc**에서 주소로 지정된 엔코더 변환 테이블 등록에 따른 "위치 변화량"을 **Motor[x].Pos**에 더합니다.

모터 제로 위치의 상대적 위치를 산출하기 위해서는 **Motor[x].HomePos**의 값을 빼십시오.

대부분의 어플리케이션에서는 **Motor[x].Pos**는 단순히 실제 위치 처리 내의 중간값이므로 무시되거나 대부분은 디버그 분석을 위해 가끔 감시되는 정도입니다. 단, 모터가 무효 상태("kill 상태")이면 **Motor[x].Pos**에 값을 쓸 수 있습니다. 이는 절대값 위치 센서를 읽는 것과 같은 효과가 있고 자동적인 위치 읽기 알고리즘을 지원되지 않는 소스에서 위치를 교정하기 위해 사용할 수 있습니다. 모터를 무효로 설정하면

Motor[x].Pos 에 쓰여진 값은 경우에 따라서는 백래시와 보정으로 수정되고 모터의 명령 위치 레지스터에 동일하게 복사됩니다. 모터가 유효인 경우(오픈 루프 또는 클로즈드 루프) **Motor[x].Pos** 에 쓰려고 하면 에러로 거부됩니다.

Motor[x].Pos2 모터의 미처리 내측 루프 실제 위치

설명: 모터의 미처리 내측 루프 실제 위치

범위: 부동 소수점

단위: 내측 루프 단위

Motor[x].Pos2 에는 위치 테이블 보정과 같은 오프셋을 더하기 전의 원시 내측 루프 실제 위치를 저장합니다. 덧셈 후의 값 **Motor[x].ActPos2** 는 속도 피드백 서보 계산에서 사용됩니다. 각 사이클에서 새로운 사이클의 "원시" 모터 위치를 얻기 위해 **Motor[x].pEnc2** 에서 주소로 지정된 엔코더 변환 테이블 등록에 따른 "위치 변화량"을 **Motor[x].Pos2** 에 더합니다.

Motor[x].PosError 모터의 서보 편차

설명: 모터 서보 편차

범위: 부동 소수점

단위: 모터 단위

Motor[x].PosError 에는 자동으로 **Motor[x].DesPos** 에서 **Motor[x].ActPos** 를 뺀 현재 서보 사이클의 위치 오차("편차")값을 저장합니다. 단위는 모터 단위로 환산됩니다. 이 요소의 절대값이 저장 설정 요소 **Motor[x].FatalFeLimit** 를 초과하는 경우에 모터는 자동으로 "kill"됩니다.

Motor[x].Ppos 모터의 궤적 프리 필터

설명: 모터의 궤적 프리 필터

범위: 부동 소수점

단위: 모터 단위

Motor[x].Ppos 에는 최신 필터 사이클의 궤적 프리 필터에 대한 입력인 모터의 명령 위치를 저장합니다. 저장 설정 요소 **Motor[x].PreFilterEna** 가 0 보다 큰 값으로 필터를 유효로 설정한 경우에 한해 사용됩니다. 필터 출력값은 **Motor[x].DesPos** 를 산출하기 위해 사용됩니다.

Motor[x].PresBISize 모터의 현재 백래시 보정값

설명: 모터의 현재 백래시 보정값

범위: 부동 소수점

단위: 모터 단위

Motor[x].PresBISize에는 저장 설정 요소 **Motor[x].BISize**와 대상 레지스터로 **Motor[x].BIcompSize** 요소를 사용하는 모든 보정 테이블에서 도출한 현재 서보 사이클의 슬루 레이트 제한이 있는 백래시 보정값을 저장합니다. 이 값은 서보 계산에 사용되는 보정된 값 **Motor[x].ActPos**를 구하는 계산으로 원시 측정 위치 **Motor[x].Pos**에 더합니다.

모터가 음의 방향으로 움직이고 있는 경우에 **Motor[x].BISize**와 **Motor[x].BIcompSize**의 합계값에 도달할 때까지 이 값은 실시간 인터럽트마다 **Motor[x].BISlewRate**의 값으로 증가합니다. 모터가 양의 방향으로 움직이고 있는 경우에 0에 도달할 때까지 이 값은 실시간 인터럽트마다 **Motor[x].BISlewRate**의 값으로 감소합니다.

Motor[x].PrevDesPos 모터의 직전 Net 명령 위치

설명: 모터의 직전 Net 명령 위치

범위: 부동 소수점

단위: 모터 단위

Motor[x].PrevDesPos에는 현재 사이클에서 1 사이클 전의 서보 사이클에서 계산된 Net 명령 위치를 저장하고 있으며 이는 계산된 궤적 위치와 위치 추종 기능에 의한 마스터 위치의 합계값입니다. 모터 단위로 환산되고 전원 투입/리셋 위치 기준이 됩니다. 이 값은 모터의 서보 알고리즘에서 속도, 가속과 마찰 Feed-forward 항 계산에 사용됩니다.

모터 제로 위치의 상대적 위치를 계산하기 위해서는 **Motor[x].HomePos**의 값을 빼십시오.

Motor[x].PrevMasterPos 모터의 직전 위치 추종 마스터 위치

설명: 모터의 직전 위치 추종 마스터 위치

범위: 부동 소수점

단위: 모터 단위

Motor[x].PrevMasterPos에는 현재 사이클에서 1 사이클 전의 서보 사이클 상의 누적 위치 추종 거리를 저장합니다. 이 값은 모터 단위로 환산됩니다.

Motor[x].PrevPhaseEnc 모터의 직전 위상 소스 위치

설명: 모터의 직전 위상 소스 위치

범위: 0 .. $2^{32}-1$

단위: 전류(commutation) 소스 레지스터 단위

Motor[x].PrevPhaseEnc에는 현재 사이클에서 1 사이클 전의 위상 사이클 상의 **Motor[x].pPhaseEnc**에서 지정되는 주소의 32 비트 레지스터 성분을 저장합니다. 이 값은 롤 오버와 전류(commutation)의 주파수 계산에 사용됩니다.

Motor[x].PrevPos2 모터의 직전 내측 루프 피드백 위치

설명: 모터의 직전 내측 루프 피드백 위치

범위: 부동 소수점

단위: 내측 루프 단위

Motor[x].PrevPos2에는 내측(보통 속도) 서보 루프를 위한 현재 사이클에서 1 사이클 이전의 서보 사이클 Net 피드백 위치를 저장합니다. 이 값은 직전 서보 사이클상의 **Motor[x].ActPos2**의 값을 사용합니다. 내측 서보 루프 단위로 환산되고 반드시 외측 루프에서 사용되는 모터 단위와 같지 않아도 되며 전원 투입/리셋 위치 기준이 됩니다. 이 값은 모터의 실제 속도 계산 중에서 사용됩니다.

Motor[x].Pui 모터의 *i* 사이클 이전의 모터 궤적 프리 필터 입력 위치

설명: *i* 사이클 이전의 모터 궤적 프리 필터 입력 위치

범위: 부동 소수점

단위: 모터 단위

Motor[x].Pui에는 "*i*" 필터 사이클 이전의 궤적 프리 필터에 대한 입력으로 사용된 모터 명령 위치를 저장합니다. 숫자 *i*는 1~4 까지의 값이 됩니다. 이 요소는 저장 설정 요소 **Motor[x].PreFilterEna**가 0 보다 큰 값으로 필터를 유효로 설정한 경우에 한해 사용됩니다. 최신 입력값은 **Motor[x].Ppos**에서 검출할 수 있습니다. 필터 출력값은 **Motor[x].DesPos**를 산출하기 위해 사용됩니다.

Motor[x].ResMasterPos 모터의 마스터 위치 잔차

설명: 모터 마스터 위치 잔차

범위: 부동 소수점

단위: 모터 단위

Motor[x].ResMasterPos에는 모터 위치 추종 마스터 위치 "잔차"를 저장합니다. 배정도 부동 소수점 요소 **Motor[x].MasterPos**의 값이 그 52 비트 가수로 전체 분해능을 유지할

수 없을 정도로 큰 경우에 **Motor[x].ResMasterPos** 는 그 "잔차"값을 유지하기 위해 사용되고 이에 따라 위치 계산의 전체 정밀도를 유지합니다. **Motor[x].MasterPos** 의 크기가 모터 단위로 약 2^{50} 또는 10^{15} 값을 초과하는 경우에 한해 **Motor[x].ResMasterPos** 는 제로가 아닌 값이 됩니다.

Motor[x].ResPos 모터의 외측 루프 실제 위치 잔차

설명: 모터의 외측 루프 실제 위치 잔차

범위: 부동 소수점

단위: 모터 단위

Motor[x].ResPos 에는 모터의 외측 루프 실제 위치 "잔차"를 저장합니다. 배정도 부동 소수점 요소 **Motor[x].ActPos** 의 값이 그 52 비트 가수로 전체 분해능을 저장할 수 없을 정도로 큰 경우에 **Motor[x].ResPos** 는 그 "잔차"값을 유지하기 위해 사용되고 이에 따라 위치 계산의 전체 정밀도를 유지합니다. **Motor[x].ActPos** 의 크기가 모터 단위로 약 2^{50} 또는 10^{15} 을 초과하는 경우에 한해 **Motor[x].ResPos** 는 제로가 아닌 값이 됩니다.

Motor[x].ResPos2 모터의 내측 루프 실제 위치 잔차

설명: 모터의 내측 루프 실제 위치 잔차

범위: 부동 소수점

단위: 모터 단위

Motor[x].ResPos2 에는 모터의 내측 루프 실제 위치 "잔차"를 저장합니다. 배정도 부동 소수점 요소 **Motor[x].ActPos2** 의 값이 그 52 비트 가수로 전체 분해능을 저장할 수 없을 정도로 큰 경우에 **Motor[x].ResPos2** 는 그 "잔차"값을 유지하기 위해 사용되고 이에 따라 위치 계산의 전체 정밀도를 유지합니다. **Motor[x].ActPos2** 의 크기가 모터 단위로 약 2^{50} 또는 10^{15} 을 초과하는 경우에 한해 **Motor[x].ResPos2** 는 제로가 아닌 값이 됩니다.

Motor[x].ServoOut 모터의 서보 루프 명령 출력

설명: 모터 서보 루프 명령 출력

범위: 부동 소수점

단위: 16 비트 단위

Motor[x].ServoOut 에는 모터의 서보 루프의 현재 출력값을 저장합니다. Power PMAC 이 모터의 위상 전류(commutation)를 실시하지 않는 경우(**Motor[x].PhaseCtrl** 비트 0, 2 = 0), 이 값과 오프셋항 **Motor[x].CompDac** 과 **Motor[x].DacBias** 의 합이 **Motor[x].pDac** 으로 지정한 레지스터에 쓰여집니다. Power PMAC 이 모터의 위상 전류(commutation)를 실시한 경우(**Motor[x].PhaseCtrl** 비트 0 또는 2 = 1), 이 값과 오프셋항 **Motor[x].CompDac** 의 합은 전류 알고리즘에 대한 "직교 방향" 전류(commutation) 명령 입력 **Motor[x].IqCmd** 로 사용됩니다.

Motor[x].SoftLimit 모터의 소프트웨어 위치 리미트 정지 Status 비트

설명: "소프트웨어 위치 리미트 정지" Status 비트

범위: 0 .. 1

단위: 논리값

모터가 양 또는 음 중 어느 한쪽의 소프트웨어 오버트래블 리미트에 도달하여 정지한 경우 또는 그 리미트 위치에 현재 없더라도 이러한 리미트 중 1 개에서 현재 모션을 정지해야 한다고 지시된 경우에 Status 비트 **Motor[x].SoftLimit** 는 1 이 됩니다. 리미트에서 빠져 나온 경우를 제외하고 리미트 위치에 있는 경우를 포함하여 모든 경우에는 0 이 됩니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[1]**의 비트 30 에 해당합니다.



주의

소프트웨어 오버트래블 리미트가 실행되면 무한 JOG 명령 j+ 또는 j-의 동작은 자동으로 변경되고 **Motor[x].SoftLimit** 는 1 로 설정되고 동작 시작하면 바로 **Motor[x].SoftLimitDir** 이 적정값으로 설정됩니다. 리미트에 도달하기 전에 그 동작이 리미트에서 변경되지 않는 별도의 JOG 명령(예를 들어 j/)이 전송되는 경우에 **Motor[x].SoftLimit** 는 즉시 0 으로 클리어됩니다.

Motor[x].SoftLimitDir 모터의 소프트웨어 리미트 방향 Status 비트

설명: "소프트웨어 리미트 방향" Status 비트

범위: 0 .. 1

단위: 논리값

Motor[x].SoftLimitDir Status 비트에는 어느 소프트웨어 오버트래블 리미트가 모터 동작을 정지, 변경, 거부되었는지를 나타냅니다. 이 Status 는 리미트 동작이 실시되었다는 것을 나타내는 **Motor[x].SoftLimit** 가 1 일 때에 한해 유효합니다. 이는 32 비트 요소 **Motor[x].Status[1]**의 비트 27 에 해당합니다.

양의 소프트웨어 오버트래블 리미트 **Motor[x].MaxPos** 에서 이 동작을 일으킨 경우에 **Motor[x].SoftLimitDir** 은 0 입니다. 음의 소프트웨어 오버트래블 리미트 **Motor[x].MinPos** 에서 이 동작을 일으킨 경우에는 1 입니다.



주의

소프트웨어 오버트래블 리미트가 실행되면 무한 JOG 명령 j+ 또는 j-의 동작은 자동으로 변경되고 **Motor[x].SoftLimit** 는 1 로 설정되고 동작 시작하면 바로 **Motor[x].SoftLimitDir** 이 적정값으로 설정됩니다. 리미트에 도달하기 전에 그 동작이 리미트에서 변경되지 않는 별도의 JOG 명령(예를 들어 j/)이 전송되는 경우에 **Motor[x].SoftLimit** 는 즉시 0 으로 클리어됩니다.

Motor[x].SoftMinusLimit 모터의 소프트웨어 마이너스 리미트 Status 비트

설명: "소프트웨어 마이너스 리미트 세트" Status 비트

범위: 0 .. 1

단위: 논리값

모터가 **Motor[x].MinPos** 와 **Motor[x].SoftLimitOffset**(유효로 설정하기 위해서는 **Motor[x].MaxPos** 미만이어야 함)으로 설정한 음의 소프트웨어 리미트에 도달했거나 초과하면 **Motor[x].SoftMinusLimit** Status 비트는 1 이 됩니다. 모터가 이 리미트를 ON 하여 정지했더라도 그 이외에는 0 이 됩니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 23 에 상당합니다.

모터의 Net 명령 위치가 (**Motor[x].MinPos** - **Motor[x].SoftLimitOffset**)을 통과할 때에 이 Status 비트가 설정되기 위해서는 **Motor[x].SoftLimitOffset** 은 음의 값으로 설정해야 한다는 점에 주의하십시오.

Motor[x].SoftPlusLimit 모터의 소프트웨어 플러스 리미트 Status 비트

설명: "소프트웨어 플러스 리미트 세트" Status 비트

범위: 0 .. 1

단위: 논리값

모터가 **Motor[x].MaxPos** 와 **Motor[x].SoftLimitOffset**(유효로 설정하기 위해서는 **Motor[x].MinPos** 보다 크게 설정해야 함)으로 설정한 양의 소프트웨어 리미트에 도달했거나 초과하면 **Motor[x].SoftPlusLimit** Status 비트는 1 이 됩니다. 모터가 이 리미트를 ON 하여 정지했더라도 그 이외에는 0 이 됩니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 22 에 상당합니다.

모터의 Net 명령 위치가 (**Motor[x].MaxPos** + **Motor[x].SoftLimitOffset**)을 통과할 때에 이 Status 비트가 설정되기 위해서는 **Motor[x].SoftLimitOffset** 은 음의 값으로 설정해야 한다는 점에 주의하십시오.

Motor[x].SpindleMotor 모터의 스핀들 모터 정의 상태

설명: 스핀들 모터 정의 상태

범위: 0 .. 3

단위: 열거 형식

모터가 좌표계에서 "스핀들 축"으로 정의될 때(**#x->S**, **#x->S0**, **#x->S1**) 상태 요소 **Motor[x].SpindleMotor** 는 0 보다 큰 값으로 설정됩니다. 어느 좌표계에서도 스핀들 축으로 정의되지 않는 경우에는 0 으로 설정됩니다.

모터가 스핀들 축으로 정의된 경우에 이 요소값은 사용하는 "타임 베이스"(오버라이드)에 따라 결정됩니다. 설정되는 값은 다음과 같습니다.

- 1: 지정된 좌표계의 타임 베이스(#x->S)를 사용
- 2: 좌표계 0의 타임 베이스(#x->S0)를 사용
- 3: 100%로 고정된 타임 베이스(#x->S1)를 사용

Motor[x].SpindleMotor 는 32 비트 요소 **Motor[x].Status[0]**의 비트 4, 5에 해당합니다.

Motor[x].Status[0] 모터의 제 1 Status 워드

설명: 모터 제 1 Status 워드

범위: \$0..\$FFFFFFFF

단위: 비트 필드

Motor[x].Status[0]은 모터의 개별적인 글로벌 Status 비트를 저장한 1 번째의 32 비트 Status 워드입니다. 워드의 Status 비트 목록은 아래의 표와 같습니다.

비트#	16 진수값	요소명:Motor[x].{요소}	설명
31	\$80000000	TriggerMove	진행 중의 트리거 서치 동작
30	\$40000000	HomeInProgress	진행 중의 원점 서치 동작
29	\$20000000	MinusLimit	하드웨어 음의 리미트 세트
28	\$10000000	PlusLimit	하드웨어 양의 리미트 세트
27	\$8000000	FeWarn	편차 경고
26	\$4000000	FeFatal	편차 이상
25	\$2000000	LimitStop	하드웨어 위치 리미트 정지
24	\$1000000	AmpFault	앰프 이상 에러
23	\$800000	SoftMinusLimit	소프트웨어 음의 리미트 세트
22	\$400000	SoftPlusLimit	소프트웨어 양의 리미트 세트
21	\$200000	I2tFault	적분 전류(I ² T) 이상
20	\$100000	TriggerNotFound	트리거 미검출
19	\$80000	AmpWarn	앰프 경고
18	\$40000	EncLoss	센서 손실 이상
16 - 17	-	-	(추후 사용하기 위해 예약)
15	\$8000	HomeComplete	원점 복귀 완료
14	\$4000	DesVelZero	명령 속도 제로
13	\$2000	ClosedLoop	클로즈드 루프 모드
12	\$1000	AmpEna	앰프 유효
11	\$800	InPos	인포지션
10	-	-	(추후 사용하기 위해 예약)
9	\$200	BlockRequest	블록 요구 플래그 세트
8	\$100	PhaseFound	위상 참조 위치 확립
7	\$80	TriggerSpeedSel	트리거가 있는 동작 속도 선택
6	\$40	GantryHomed	갠트리 원점 완료
5	\$20	SpindleMotor (bit1)	스핀들 축 정의 상태
4	\$10	SpindleMotor (bit0)	스핀들 축 정의 상태
0 - 3	-	-	(추후 사용하기 위해 예약)

이 요소값은 조회 명령 **#x?**으로 통지됩니다. 그 성분은 응답의 최초 16 진수 8 자리(1 개의 숫자당 4 비트)로 구성됩니다. 온라인 조회 명령 **backup Motor[x].Status** 는 Power PMAC 이 영어 텍스트와 개별적인 요소의 모든 값을 통지합니다.

요소는 그 자체의 개별 지정 부분에서 상세하게 설명합니다.

Motor[x].Status[1] 모터의 제 2 Status 워드

설명: 모터 제 2 Status 워드

범위: \$0 .. \$FFFFFFFF

단위: 비트 필드

Motor[x].Status[1]은 많은 개별적인 글로벌 Status 비트를 저장하는 2 번째의 32 비트 Status 워드입니다. 워드의 Status 비트 목록은 다음과 같습니다.

비트#	16 진수값	요소명:Motor[x].{요소}	설명
31	\$80000000	Csolve	PMATCH 계산 중에 사용되는 모터
30	\$40000000	SoftLimit	소프트웨어 위치 리미트 정지
29	\$20000000	DacLimit	서보 출력 제한
28	\$10000000	BIDir	백래시 방향
27	\$80000000	SoftLimitDir	소프트웨어 리미트에 의한 동작 수정(0 = 정방향, 1 = 역방향)
0 – 26	-	-	(예약)

이 요소값은 조회 명령 **#x?**으로 통지됩니다. 그 성분은 참조의 2 번째 16 진수 8 자리(1 개의 숫자당 4 비트)로 구성됩니다. 온라인 조회 명령 **backup Motor[x].Status** 는 Power PMAC 이 영어 텍스트와 개별적인 요소의 모든 값을 통지합니다.

요소는 그 자체의 개별 지정 부분에서 상세하게 설명합니다.

Motor[x].TriggerMove 모터의 트리거 동작 서치 중 Status 비트

설명: "트리거 동작 서치 중" Status 비트

범위: 0 .. 1

단위: 논리값

Motor[x].TriggerMove Status 비트는 트리거가 있는 동작(원점 서치, 트리거가 있는 JOG, 프로그램 RAPID 모드 트리거가 있는 동작)의 시작에서 1 로 설정됩니다. 미리 지정된 트리거 조건이 검출되고 트리거 검출 후의 동작이 시작되거나 트리거가 검출되지 않고 동작이 종료한 경우에 0 으로 설정됩니다. 이 Status 비트는 32 비트 요소 **Motor[x].Status[0]**의 비트 31 에 상당합니다.

Motor[x].TriggerNotFound 모터의 트리거 미검출 Status 비트

설명: "트리거 미검출" Status 비트

범위: 0 .. 1

단위: 논리값

미리 지정된 트리거 조건이 검출되지 않고 트리거가 있는 JOG 또는 프로그램 RAPID 모드 트리거가 있는 동작이 종료한 경우에 **Motor[x].TriggerNotFound** Status 비트가 1로 설정됩니다. 이는 에러 조건이 아닙니다. 그러나 많은 경우 후속 동작은 이 비트의 설정 여부에 좌우됩니다. 이 값은 다음 동작이 시작될 때 0으로 복귀하고 그 이외에서는 0입니다. 이 Status 비트는 32비트 요소 **Motor[x].Status[0]**의 비트 20에 해당합니다.

Motor[x].TriggerSpeedSel 모터의 트리거 속도 선택 Status 비트

설명: "트리거 속도 선택" Status 비트

범위: 0 ..1

단위: 논리값

트리거가 있는 동작이 **Motor[x].MaxSpeed**의 속도로 실시되는 경우에 **Motor[x].TriggerSpeedSel** Status 비트는 1로 설정됩니다. 동작이 **Motor[x].JogSpeed**의 속도로 실시되는 경우에 0으로 설정됩니다. 프로세서가 어느 속도로 트리거 후 동작 부분에 사용하는지 알리기 위한 내부 용도용입니다. 이 Status는 32비트 요소 **Motor[x].Status[0]**의 비트 7에 해당합니다.

Motor[x].VxCouple 모터의 전류 루프 크로스 커플링 주파수

설명: 모터 전류 루프 크로스 커플링 주파수

범위: 부동 소수점

단위: 전류(commutation) 주파수

Motor[x].VxCouple에는 자화와 직교 전류 루프의 크로스 커플에 사용되는 현재 전류(commutation) 주파수를 저장합니다. 디지털 전류 루프 제어가 유효인 경우에 한해 계산됩니다. 이 값은 현재 모터 속도 및 저장 설정 요소 **Motor[x].IxCoupleGain**에서 산출됩니다.

Motor[x].Desired.궤적 서브 구조체 요소

각 모터는 모션 명령 상의 현재 실행 중인 방정식에 관한 중요한 정보가 저장된 서브 구조체 **Motor[x].Desired** 가 있습니다. 이러한 동작 방정식은 JOG 나 홈 명령의 직접적인 동작 명령 또는 모터에 할당된 프로그램 축에서 도출됩니다.

Motor[x].Desired.Accel 모터 현재 실행의 가속도 계수

설명: 모터의 현재 궤적 방정식 가속도 계수

범위: 부동 소수점

단위: 모터 위치 단위/msec²/2

Motor[x].Desired.Accel 에는 모션 궤적 명령의 현재 실행 중인 모터의 3 차 방정식 가속도 계수를 저장합니다. 이 계수는 모션의 현재 섹션 또는 세그먼트 상의 방정식 시작 시 가속의 1/2 에 해당합니다.

Motor[x].Desired.Jerk 모터 현재 실행의 저크 계수

설명: 모터의 현재 궤적 방정식 저크 계수

범위: 부동 소수점

단위: 모터 위치 단위/msec³/6

Motor[x].Desired.Jerk 에는 모션 궤적 명령의 현재 실행 중인 모터의 3 차 방정식 저크(가속 변화 레이트) 계수를 저장합니다. 이 계수는 모션의 현재 섹션 또는 세그먼트 상의 방정식 시작 시 저크의 1/6 에 해당합니다.

Motor[x].Desired.Next 모터 현재 실행의 다음 인덱스

설명: 모터의 다음 궤적 방정식 인덱스

범위: 0 .. 15

단위: 인덱스값

Motor[x].Desired.Next 에는 현재 실행 중인 섹션 또는 세그먼트에서 다음에 실행되는 섹션 또는 세그먼트의 모션 방정식을 저장한 서브 구조체 **Motor[x].New[i]**의 인덱스값 *i* 를 저장합니다.

Motor[x].Desired.Pos 모터 현재 실행의 위치 계수

설명: 모터의 현재 궤적 방정식 위치 계수

범위: 부동 소수점

단위: 모터 위치 단위

Motor[x].Desired.Pos에는 모션 궤적 명령의 현재 실행 중인 모터의 3차 방정식 위치 계수를 저장합니다. 이 계수는 모션의 현재 섹션 또는 세그먼트 상의 방정식 시작 시 위치에 해당합니다.

Motor[x].Desired.Time 모터 현재 실행의 시간 범위

설명: 모터의 현재 궤적 방정식 시간 범위

범위: 부동 소수점

단위: msec

Motor[x].Desired.Time에는 모션 궤적 명령의 현재 실행 중인 모터의 3차 방정식 시간 범위를 저장합니다. 100%의 타임 베이스에서 실행할 때 이는 섹션 또는 세그먼트의 처음부터 끝까지의 시간입니다.

Motor[x].Desired.TimerEnabled 모터 현재 실행의 시한 동작 Status 플래그

설명: 모터의 시한 동작 Status 플래그

범위: 0 .. 1

단위: 논리값

모터 명령 또는 프로그램된 축 명령에 의한 유한 시간의 클로즈드 루프 동작 섹션을 모터가 현재 실행 중이면 **Motor[x].Desired.TimerEnabled** 상태 플래그는 1로 설정됩니다. 그 이외에는 0으로 설정됩니다. 원점 서치 동작의 트리거 전 일정 속도 섹션 그리고 무한 JOG 명령의 일정 속도 섹션이 유한 시간이 아니라는 점에 주의하십시오. 따라서 이 비트는 이러한 섹션 실행 중에는 0입니다.

Motor[x].Desired.Vel 모터 현재 실행의 속도 계수

설명: 모터의 현재 궤적 방정식 속도 계수

범위: 부동 소수점

단위: 모터 위치 단위/msec

Motor[x].Desired.Vel에는 모션 궤적 명령의 현재 실행 중인 모터의 3차 방정식 속도 계수를 저장합니다. 이 계수는 모션의 현재 섹션 또는 세그먼트 상의 방정식 시작 시 속도에 해당합니다.

Motor[x].New[i].궤적 부구조체 요소

각 모터에는 즉시 실행되는 모션 명령의 방정식 버퍼에 관한 중요한 정보가 저장된 서브 구조체 **Motor[x].New[i]**가 있습니다. 이러한 동작 방정식은 JOG 나 홈 명령의 직접적인 동작 명령 또는 모터에 할당된 프로그램 축에서 도출됩니다. 이러한 서브 구조체는 16 개가 있으며($i = 0 \sim 15$) 즉시 동작하는 운동식의 실체가 버퍼링됩니다. 버퍼는 로터리 버퍼로 사용됩니다.

Motor[x].New[i].Accel 모터 직전 실행의 가속도 계수

설명: 모터 버퍼 궤적 방정식 가속도 계수

범위: 부동 소수점

단위: 모터 위치 단위/msec²/2

Motor[x].New[i].Accel에는 모션 궤적 명령의 버퍼링된 모터의 3 차 방정식 가속도 계수를 저장합니다. 이 계수는 모션의 해당 섹션 또는 세그먼트 상의 방정식 시작 시 가속의 1/2에 해당합니다.

Motor[x].New[i].Jerk 모터 직전 실행의 저크 계수

설명: 모터 버퍼 궤적 방정식 저크 계수

범위: 부동 소수점

단위: 모터 위치 단위/msec³/6

Motor[x].New[i].Jerk에는 모션 궤적 명령의 현재 버퍼링된 현재의 3 차 방정식 저크(가속 변화 레이트) 계수를 저장합니다. 이 계수는 모션의 해당 섹션 또는 세그먼트 상의 방정식 시작 시 저크의 1/6에 해당합니다.

Motor[x].New[i].Pos 모터 직전 실행의 위치 계수

설명: 모터 버퍼 궤적 방정식 위치 계수

범위: 부동 소수점

단위: 모터 위치 단위

Motor[x].New[i].Pos에는 모션 궤적 명령의 버퍼링된 3 차 방정식의 위치 계수를 저장합니다. 이 계수는 모션의 해당 섹션 또는 세그먼트 상의 방정식 시작 시 위치에 해당합니다.

Motor[x].New[j].Time 모터 직전 실행의 시간 범위

설명: 모터 버퍼는 궤적 방정식 시간 범위

범위: 부동 소수점

단위: msec

Motor[x].New[j].Time 에는 모션 궤적 명령의 버퍼링된 3 차 방정식 시간 범위를 저장합니다. 100%의 타임 베이스에서 실행할 때 이는 섹션 또는 세그먼트의 처음부터 끝까지의 시간입니다.

Motor[x].New[j].Vel 모터 직전 실행의 속도 계수

설명: 모터 버퍼 궤적 방정식 속도 계수

범위: 부동 소수점

단위: 모터 위치 단위/msec

Motor[x].New[j].Vel 에는 모션 궤적 명령의 버퍼링된 3 차 운동식 속도 계수를 저장합니다. 이 계수는 모션의 해당 섹션 또는 세그먼트 상의 방정식 시작 시 속도에 해당합니다.

Motor[x].Servo.부구조체 상태 요소

이 섹션의 요소는 Power PMAC 에서 자동으로 계산되는 서보 관련 정보를 제공합니다. 해당 요소는 모터의 서보 부구조체의 일부입니다.

Motor[x].Servo.EstGain 모터 서보의 현재 추정 플랜트 게인

설명: 현재 추정 플랜트 게인
범위: 음이 아닌 부동 소수점
단위: (모터 단위/sec²)/16 비트 명령 단위

Motor[x].Servo.EstGain에는 직전 **Motor[x].Servo.EstTime** 서보 사이클의 결과에 따라 적응 서보 제어의 귀납적인 추정 알고리즘에서 계산된 현재 추정 "플랜트 게인"을 저장합니다. 적응 서보 제어 알고리즘이 선택된 경우에 한해 사용됩니다. 이 값은 모터 단위/sec²/16 비트 출력 명령 단위로 가속도 비율을 표시합니다.

이 값은 적응 게인 보정값 **Motor[x].Servo.GainFactor**를 산출하기 위해 사용되므로 함께 루프 게인이 가능한 한 일정값을 유지합니다.

Motor[x].Servo.EstTimer 모터 서보의 적응 제어 서보 사이클 수

설명: 적응 제어에서 현재 사용 중인 서보 사이클 수
범위: 음이 아닌 정수
단위: 서보 사이클

Motor[x].Servo.EstTimer에는 모터의 적응 서보 제어 알고리즘에서 플랜트 게인 추정이 정상적인 조건으로 유지된 연속되는 서보 사이클 수를 저장합니다. 적응 서보 제어 알고리즘이 선택된 경우에 한해 사용됩니다.

플랜트 게인 추정을 위한 조건은 모터 명령 가속도가 제로가 아니며 서보 명령 출력의 크기가 **Motor[x].Servo.EstMinDac**보다 커야 합니다. **Motor[x].Servo.EstTimer**의 값이 저장 설정 요소 **Motor[x].Servo.EstTime**을 초과할 때 Power PMAC은 적응 서보 루프 제어에서 플랜트 게인(**Motor[x].Servo.EstGain**)을 계산하는 데 사용합니다.

플랜트 게인 추정 조건이 적절한 조건을 충족하지 않는 서보 사이클에서는 **Motor[x].Servo.EstTimer**은 0으로 설정됩니다.

Motor[x].Servo.GainFactor 모터 서보의 적응 제어 기준 서보 게인

설명: 현재 기준 서보 게인

범위: 음이 아닌 부동 소수점

단위: 없음(비율)

Motor[x].Servo.GainFactor에는 적응 제어 알고리즘의 직전 추정 장치 게인 계산에서 결과적으로 발생하는 현재 기준 서보 루프 게인값을 저장합니다. 적응 서보 제어 알고리즘이 선택된 경우에 한해 사용됩니다. 취득한 루프 게인과 서보 루프 성능을 가능한 한 일정하게 유지할 목적으로 표준 저장된 서보 루프 게인을 재스케일링하기 위해 사용됩니다. 이 값이 저장되지 않는다는 점에 주의하십시오. 전원 투입/리셋에서는 자동으로 1.0으로 설정됩니다.

Motor[x].Servo.GainFactor는 저장 설정 요소 **Motor[x].Servo.MinGainFactor**의 값 미만이지 않습니다. 또한 저장 설정 요소 **Motor[x].Servo.MaxGainFactor**의 값을 초과하는 경우는 없습니다.

Motor[x].Servo.Integrator 모터 서보의 편차 적분 출력

설명: 위치 편차 적분 출력

범위: 부동 소수점

단위: 16비트 서보 명령 출력 비트

Motor[x].Servo.Integrator에는 표준 서보 위치 편차 적분의 현재 출력값을 저장합니다. 이 값은 Net 서보 출력을 얻기 위해 다른 서보항의 출력을 더합니다. 크기는 저장 설정 요소 **Motor[x].Servo.MaxInt**로 제한됩니다.

Motor[x].Servo.Status 모터 서보의 불감대 출력 상태

설명: 히스테리시스를 가진 불감대 출력 상태

범위: 0 .. 1

단위: 논리값

Motor[x].Servo.Status에는 서보 루프의 히스테리시스를 가진 불감대 출력의 현재 상태를 저장합니다. 불감대 기능이 "OFF"로 서보 루프 출력이 유효인 경우에는 0입니다. 불감대 기능이 "ON"으로 서보 루프 출력이 강제로 0으로 출력되는 경우에는 1입니다. 모터 명령 속도가 정확하게 0.0과 같은 경우에 한해 Power PMAC은 이 불감대 상태를 산출합니다.

Motor[x].Servo.uai 모터 서보의 다항식 A 입력

설명: i 사이클 이전의 명령 위치 다항식 입력

범위: 부동 소수점

단위: 모터 단위

Motor[x].Servo.uai에는 서보 알고리즘에서 Net 명령 위치에 작용하는 "A" 서보 다항식의 i 서보 사이클 이전 입력값을 저장합니다. 이 값은 대응하는 저장 설정 요소 서보 게인 **Motor[x].Servo.Kai**로 곱합니다.

Power PMAC이 해당 항목을 사용하기 위해서는 저장 설정 요소 **Motor[x].Ctrl**에서 다항식 필터를 사용하는 서보 알고리즘을 선택해야 합니다. 이러한 요소는 **ua1**부터 **ua7**까지 7개가 있습니다. 저장 설정 요소 **Motor[x].Servo.SwPoly7**이 기본값 0인 경우에 **ua1**과 **ua2**에 한해 사용됩니다. 값이 1인 경우에 7개의 값 모두가 사용됩니다.

Motor[x].Servo.ubi 모터 서보의 다항식 B 입력

설명: i 사이클 이전의 실제 위치 다항식 입력

범위: 부동 소수점

단위: 모터 단위

Motor[x].Servo.ubi에는 서보 알고리즘에서 Net 실제 위치에 작용하는 "B" 서보 다항식의 i 서보 사이클 이전 입력값을 저장합니다. 이 값은 대응하는 저장 설정 요소 서보 게인 **Motor[x].Servo.Kbi**로 곱합니다.

Power PMAC이 해당 항목을 사용하기 위해서는 저장 설정 요소 **Motor[x].Ctrl**에서 다항식 필터를 사용하는 서보 알고리즘을 선택해야 합니다. 이러한 요소는 **ub1**부터 **ub7**까지 7개가 있습니다. 저장 설정 요소 **Motor[x].Servo.SwPoly7**이 기본값 0인 경우에 **ub1**과 **ub2**에 한해 사용됩니다. 값이 1인 경우에 7개의 값 모두가 사용됩니다.

Motor[x].Servo.uci 모터 서보의 다항식 C/D 입력

설명: i 사이클 이전의 편차 다항식 입력

범위: 부동 소수점

단위: 모터 단위

Motor[x].Servo.uci에는 서보 알고리즘에서 위치(추종) 편차에 작용하는 "C"와 "D" 서보 다항식 분자항과 분모항의 i 서보 사이클 이전 입력값을 저장합니다. 이 값은 대응하는 저장 설정 요소 서보 게인 **Motor[x].Servo.Kci**로 곱합니다.

Power PMAC이 해당 항목을 사용하기 위해서는 저장 설정 요소 **Motor[x].Ctrl**에서 다항식 필터를 사용하는 서보 알고리즘을 선택해야 합니다. 이러한 요소는 **uc1**부터 **uc7**까지 7개가 있습니다. 저장 설정 요소 **Motor[x].Servo.SwPoly7**이 기본값 0인 경우에 **uc1**과 **uc2**에 한해 사용됩니다. 값이 1인 경우에 7개의 값 모두가 사용됩니다.

Motor[x].Servo.udi 모터 서보의 다항식 C/D 출력

설명: i 사이클 이전의 편차 다항식 출력

범위: 부동 소수점

단위: 모터 단위

Motor[x].Servo.udi에는 서보 알고리즘에서 Net 명령 위치에 작용하는 "C"와 "D" 서보 다항식 분자항과 분모항의 i 서보 사이클 이전 출력값을 저장합니다. 이 값은 대응하는 저장 설정 요소 서보 게인 **Motor[x].Servo.Kdi**로 곱합니다.

Power PMAC이 해당 항목을 사용하기 위해서는 저장 설정 요소 **Motor[x].Ctrl**에서 다항식 필터를 사용하는 서보 알고리즘을 선택해야 합니다. 이러한 요소는 **ud1**부터 **ud7**까지 7개가 있습니다. 저장 설정 요소 **Motor[x].Servo.SwPoly7**이 기본값 0인 경우에 **ud1**과 **ud2**에 한해 사용됩니다. 값이 1인 경우에 7개의 값 모두가 사용됩니다.

Motor[x].Servo.uei 모터 서보의 다항식 E 출력

설명: i 사이클 이전의 내측 루프 속도 다항식 출력

범위: 부동 소수점

단위: 모터 단위

Motor[x].Servo.uei에는 서보 알고리즘에서 내측 (속도) 루프 위치에 작용하는 "E" 서보 다항식의 i 서보 사이클 이전 출력값을 저장합니다. 이 값은 대응하는 저장 설정 요소 서보 게인 **Motor[x].Servo.Kei**로 곱합니다.

Power PMAC이 해당 항목을 사용하기 위해서는 저장 설정 요소 **Motor[x].Ctrl**에서 다항식 필터를 사용하는 서보 알고리즘을 선택해야 합니다. 이러한 요소는 **ue1**과 **ue7** 2개가 있습니다.

Motor[x].Servo.ufi 모터 서보의 다항식 F 출력

설명: i 사이클 이전의 명령 속도 다항식 출력

범위: 부동 소수점

단위: 모터 단위

Motor[x].Servo.ufi에는 서보 알고리즘에서 Feed-forward 경로에 Net 명령 속도에 작용하는 "F" 서보 다항식의 i 서보 사이클 이전 출력값을 저장합니다. 이 값은 대응하는 저장 설정 요소 서보 게인 **Motor[x].Servo.Kfi**로 곱합니다.

Power PMAC이 해당 항목을 사용하기 위해서는 저장 설정 요소 **Motor[x].Ctrl**에서 다항식 필터를 사용하는 서보 알고리즘을 선택해야 합니다. 이러한 요소는 **uf1**과 **uf2** 2개가 있습니다.

Motor[x].Servo.Xint 모터 서보의 크로스 커플링 적분 편차 출력

설명: 크로스 커플링 적분 위치 편차 차이 출력

범위: 부동 소수점

단위: 16 비트 서보 명령 출력 비트

Motor[x].Servo.Xint 에는 현재 크로스 커플링 서보 위치 편차 차이 적분 출력값을 저장합니다. 이 값은 Net 서보 출력을 얻기 위해 다른 서보항의 출력을 더합니다. 크로스 커플링된 갠트리 제어 알고리즘에서 한해 사용됩니다.

Motor[x].TraceData[j].궤적 서브 구조체 요소

각 모터는 **Motor[x].TraceSize0** 이 0 보다 크게 설정되어 트레이스 버퍼가 유효인 경우에 모션 명령이 이미 실행된 운동식에 관한 중요한 정보가 저장된 서브 구조체 **Motor[x].TraceData[j]**를 가집니다. 이러한 동작 방정식은 JOG 나 홈 명령의 직접적인 동작 명령 또는 모터에 할당된 프로그램 축에서 도출됩니다. 인덱스값 **i** 은 0 부터 **TraceSize - 1** 의 범위 내에서 지정할 수 있습니다. 트레이스 버퍼는 음의 타임 베이스값으로 모션의 역실행을 할 수 있습니다. 버퍼는 로터리 버퍼로 사용됩니다.

Motor[x].TraceData[j].Accel 모터 트레이스 데이터의 가속도 계수

설명: 모터 트레이스 버퍼 궤적 방정식 가속도 계수

범위: 부동 소수점

단위: 모터 위치 단위/msec³/2

Motor[x].TraceData[j].Accel 에는 모터의 버퍼링된 모션 궤적 명령의 3 차 방정식 가속도 계수를 저장합니다. 이 계수는 모션의 해당 섹션 또는 세그먼트 상의 방정식 시작 시 가속도의 1/2 에 상당합니다.

Motor[x].TraceData[j].Jerk 모터 트레이스 데이터의 저크 계수

설명: 모터 트레이스 버퍼 궤적 방정식 저크 계수

범위: 부동 소수점

단위: 모터 위치 단위/msec²/6

Motor[x].TraceData[j].Jerk 에는 버퍼링된 궤적 명령 모션의 모터 3 차 방정식 저크(가속도 변화 레이트) 계수를 저장합니다. 이 계수는 모션의 해당 섹션 또는 세그먼트 상의 방정식 시작 시 저크의 1/6 에 상당합니다.

Motor[x].TraceData[j].Pos 모터 트레이스 데이터의 위치 계수

설명: 모터 트레이스 버퍼 궤적 방정식 위치 계수

범위: 부동 소수점

단위: 모터 위치 단위

Motor[x].TraceData[j].Pos 에는 버퍼링된 궤적 명령 모션의 모터의 3 차 방정식 위치 계수를 저장합니다. 이 계수는 모션의 해당 섹션 또는 세그먼트 상의 방정식 시작 시 위치에 해당합니다.

Motor[x].TraceData[j].Time 모터 트레이스 데이터의 시간 범위

설명: 모터 트레이스 버퍼 궤적 방정식 시간 범위

범위: 부동 소수점

단위: msec

Motor[x].TraceData[j].Time에는 버퍼링된 궤적 명령 모션의 모터의 3 차 방정식을 위한 시간 범위를 저장합니다. 100%의 타임 베이스에서 실행할 때 이는 섹션 또는 세그먼트의 처음부터 끝까지의 시간입니다.

Motor[x].TraceData[j].Vel 모터 트레이스 데이터의 속도 계수

설명: 모터 트레이스 버퍼 궤적 방정식 속도 계수

범위: 부동 소수점

단위: 모터 위치 단위/msec

Motor[x].TraceData[j].Vel에는 버퍼링된 궤적 명령 모션 모터의 3 차 방정식 속도 계수를 저장합니다. 이 계수는 모션의 해당 섹션 또는 세그먼트 상의 방정식 시작 시 속도에 해당합니다.

Motor[x].TraceExec. 궤적 서브 구조체 요소

각 모터에는 트레이스 버퍼에서 현재 실행 중인 모션 명령의 운동식에 관한 중요한 정보를 저장한 서브 구조체 **Motor[x].TraceExec** 이 있습니다. 이는 **Motor[x].TraceSize** 가 0 보다 크게 설정되어 트레이스 버퍼가 유효인 경우에 사용됩니다. 이러한 동작 방정식은 JOG 나 홈 명령의 직접적인 동작 명령 또는 모터에 할당된 프로그램 축에서 도출됩니다. 트레이스 버퍼는 음의 타임 베이스값으로 모션의 역실행을 할 수 있습니다. 버퍼는 로터리 버퍼로 사용됩니다.

Motor[x].TraceExec.Accel 모터 트레이스 실행 가속도 계수

설명: 모터의 트레이스 실행 버퍼 궤적 방정식 가속도 계수

범위: 부동 소수점

단위: 모터 위치 단위/msec²/2

Motor[x].TraceExec.Accel 에는 트레이스 버퍼의 현재 실행 중인 섹션 또는 세그먼트 상의 모터 궤적 명령 모션의 3 차 방정식 가속도 계수를 저장합니다. 이 계수는 모션의 이 섹션 또는 세그먼트 상의 방정식 시작 시 가속의 1/2 에 상당합니다.

Motor[x].TraceExec.Jerk 모터 트레이스 실행 저크 계수

설명: 모터의 트레이스 실행 버퍼 궤적 방정식 저크 계수

범위: 부동 소수점

단위: 모터 위치 단위/msec³/6

Motor[x].TraceExec.Jerk 에는 트레이스 버퍼의 현재 실행 중인 섹션 또는 세그먼트 상의 모터 궤적 명령 모션의 3 차 방정식 저크(가속도 변화 레이트) 계수를 저장합니다. 이 계수는 모션의 해당 섹션 또는 세그먼트 상의 방정식 시작 시 저크의 1/6 에 상당합니다.

Motor[x].TraceExec.Pos 모터 트레이스 실행 위치 계수

설명: 모터의 트레이스 실행 버퍼 궤적 방정식 위치 계수

범위: 부동 소수점

단위: 모터 위치 단위

Motor[x].TraceExec.Pos 에는 트레이스 버퍼의 현재 실행 중인 섹션 또는 세그먼트 상의 모터 궤적 명령 모션의 3 차 방정식 위치 계수를 저장합니다. 이 계수는 모션의 이 섹션 또는 세그먼트 상의 방정식 시작 시 위치에 상당합니다.

Motor[x].TraceExec.Time 모터 트레이스 실행 시간 범위

설명: 모터의 트레이스 실행 버퍼 궤적 방정식 시간 범위

범위: 부동 소수점

단위: msec

Motor[x].TraceExec.Time에는 트레이스 버퍼의 현재 실행 중인 섹션 또는 세그먼트 상의 모터 궤적 명령 모션의 3 차 방정식 시간 범위를 저장합니다. 100%의 타임 베이스에서 실행할 때 이는 섹션 또는 세그먼트의 처음부터 끝까지의 시간입니다.

Motor[x].TraceExec.Vel 모터 트레이스 실행 속도 계수

설명: 모터의 트레이스 실행 버퍼 궤적 방정식 속도 계수

범위: 부동 소수점

단위: 모터 위치 단위/msec

Motor[x].TraceExec.Vel에는 트레이스 버퍼에서 현재 실행 중인 섹션 또는 세그먼트 상의 모터 궤적 명령 모션의 3 차 방정식 속도 계수를 저장합니다. 이 계수는 모션의 해당 섹션 또는 세그먼트 상의 방정식 시작 시 속도에 상당합니다.

POWER PMAC 온라인 명령 사양

이 섹션에서는 Power PMAC 스크립트 언어의 온라인 명령에 대해 설명합니다. 온라인 명령은 즉시 실행된 후 파기됩니다. 버퍼 프로그램 명령과는 달리 Power PMAC에 송신한 일련의 온라인 명령은 다시 호출할 수 없습니다.

#

기능: 현재 스레드의 주소 지정된 모터를 통지함

범위: 통신 스레드 고유

구문: #

명령을 사용하면 Power PMAC은 이 명령이 송신되는 통신 스레드에 주소 지정된 모터의 수를 반환합니다. 다른 모터가 **#{상수}** 명령으로 주소 지정될 때까지 해당 모터가 대상 스레드 내에서 송신되는 후속 모터 고유의 명령에 작용합니다.

이 명령은 스레드의 데이터 구조체 요소 [Ldata.Motor]의 값을 조사합니다. 이는 [Ldata.Motor] 명령을 사용하여 요소를 직접 조사하는 것과 같습니다.

다른 통신 스레드가 동시에 다른 모터를 주소 지정할 가능성이 있습니다.

Power PMAC은 전원 셋업/리셋 시에 기본값으로 Motor 0(#0)을 주소 지정합니다.

#{상수}

기능: 스레드가 주소 지정한 모터를 선택함

범위: 통신 스레드 고유

구문: #{상수}

여기에서

- **{상수}**는 0부터 [Sys.MaxMotors]-1 범위 내의 정수입니다.

#{상수} 명령을 사용하면 **{상수}**로 지정된 모터가 이 명령을 송신한 통신 스레드에 대해 주소 지정되는 모터가 됩니다. 다른 모터가 다른 **#{상수}** 명령으로 주소 지정될 때까지 이 모터가 대상 스레드 내에서 송신되는 후속 모터 고유의 명령에 작용합니다.

이 명령은 스레드의 데이터 구조체 요소 [Ldata.Motor]의 값을 설정합니다. 이는 **Ldata.Motor={식}** 명령을 사용하여 요소를 직접 설정하는 것과 같습니다.

#{목록} 명령을 사용하여 목록 직후의 모터 고유 명령으로 기능하는 여러 모터를 지정할 수 있습니다. 주소 지정된 모터가 목록 내에 없는 경우에는 그 모터는 명령으로 기능하지 않습니다. **#{목록}** 명령에서는 이후의 명령으로 주소 지정되는 모터를 변경할 수 없습니다.

다른 통신 스레드가 동시에 다른 모터를 주소 지정할 가능성이 있습니다.

전원 셋업/리셋 시에는 Power PMAC은 기본값으로 Motor 0(#0)을 주소 지정합니다.

예

```
#1j+          // Address Motor 1, command it to jog positive
j-            // Command addressed motor (#1) to jog negative
#2j+          // Address Motor 2, command it to jog positive
#3..6hm       // Command Motors 3 - 6 to start homing-search moves
j/            // Command addressed motor (#2) to stop jogging
```

#{상수}->0

기능: 모터를 좌표계, Null 정의에 할당함

범위: 좌표계 고유

구문: **#{상수}->0**

여기에서

- **{상수}**는 0 부터 [Sys.MaxMotors]-1 범위 내의 정수입니다.

#{상수}->0 명령에서는 모터가 자신의 궤적 계산을 실행하지 않는 모드인 경우를 제외하고 Power PMAC 은 지정된 모터를 주소 지정된 좌표계에 할당합니다. 대부분의 경우에 이 모드는 비활성 모터([Motor[x].ServoCtrl] = 0 인 모터)로 사용되고 해당 모터가 어느 축과도 관련되어 있지 않다는 것을 나타냅니다. 기본값에서는 모든 모터는 이 정의에 따라 좌표계 0에 할당됩니다.

이 설정은 [Motor[x].ServoCtrl]을 8 로 설정하면 활성 모터가 갠트리 추종 모드인 경우에도 적용됩니다. 이는 [Motor[x].CmdMotor]로 지정된 모터에서 계산된 갠트리를 사용합니다. 이 경우에는 모터가 갠트리 주축과 같은 좌표계에 있고 장애 시의 모든 갠트리 모터 처리가 동일하다는 것이 중요합니다. 모든 갠트리 모터에서 [Motor[x].FaultMode]의 비트 0 을 1 로 설정하고 갠트리 중 하나의 모터에 편차 이상, 앰프 폴트 또는 적분 전류 폴트가 발생한 경우에 모두 출력 정지해야 합니다.

어떤 좌표계에서 모터의 정의가 0 이외인 경우에 다른 좌표계에서 이 정의의 사용을 시도하면 에러로 거부됩니다. 단, 어떤 좌표계에서 모터의 정의가 0 인 경우에 Power PMAC 에서는 다른 좌표계에서 이 축 정의(또는 다른 축 정의 중 하나)가 허용됩니다. 따라서 실제 축 정의를 가진 좌표계에서 모터를 삭제하는 경우에는 처음에 대상 좌표계에서 모터를 '0'으로 정의하고 다음에 다른 좌표계에서 축 정의(실제로 또는 0 으로)를 설정합니다.

이 명령 처리는 Turbo PMAC 의 명령 처리와는 달리 좌표계에서 모터를 완전히 삭제한다는 점에 주의하십시오.

#{상수}->{축 정의}

기능: 지정된 모터의 축 정의를 설정함

범위: 좌표계 고유

구문: **#{상수}->{축 정의}**

여기에서

- **{상수}**는 0 부터 [Sys.MaxMotors]-1 범위 내의 정수입니다.

- **{축 정의}**는 +또는 -의 산술 연산자로 구별된 1 개 이상의 **[[상수]]{축}**으로 구성됩니다. 여기에서는
 - o 임의의 부동 소수점 **{상수}**는 스케일 계수의 축을 의미하며 축 단위(공학 단위 또는 사용자 단위)별 모터 단위를 나타냅니다. 명시적으로 아무것도 지정되지 않은 경우에 Power PMAC에서는 값을 1.0으로 간주됩니다.
 - o **{축}**은 모터가 할당되는 축을 지정하는 1 문자 또는 2 문자(A, B, C, U, V, W, X, Y, Z, AA, BB, ... HH, LL, MM, ... ZZ)로 나타냅니다.
 - o +또는 -연산자 다음의 마지막 부동 소수점 **{상수}**는 모터의 제로 위치와 모터 단위에서 축 제로 위치 사이의 오프셋을 나타냅니다. 이 값이 명시적으로 지정되지 않은 경우에는 Power PMAC은 값을 0.0으로 간주합니다.

#{상수}>{축 정의} 명령은 지정된 모터를 주소 지정된 좌표계의 축 세트에 할당합니다. 또한 1 개 이상의 축에 대한 스케일링 및 시작 프로그래밍 오프셋을 정의합니다. 대부분의 경우에 Power PMAC 모터와 축은 1:1로 대응되므로 축 정의에서도 모터에 대해 1개의 축 명칭만 사용합니다.

스케일 계수는 보통 축을 나타내는 문자와 함께 사용합니다. 따라서 축 동작은 표준 단위(밀리미터, 인치, 도수 등)로 지정할 수 있습니다. *이 수치는 이 축의 사용자 단위를 정의하기 위한 것입니다.* 스케일 계수를 지정하지 않는 경우에 그 축의 사용자 단위는 1 모터 카운트입니다.

또한 오프셋 파라미터를 사용하여 프로그래밍 목적으로 축의 제로 위치를 모터의 홈 제로 위치 이외로 설정할 수도 있습니다(이를 초기 오프셋이라 하며 오프셋 명령으로 추후에 변경할 수 있습니다).

프로그램을 통한 동작이나 세그먼트마다 Power PMAC은 이 축 정의 구문으로 지정된 식을 해결하여 명령된 모터 위치를 계산합니다.

지정된 모터가 다른 좌표계의 축에 할당된 경우에 Power PMAC은 이 명령을 거부하고 에러를 통지합니다. 지정된 모터가 주소 지정된 좌표계의 축에 할당된 경우에 이전의 정의를 새로운 정의로 덮어씁니다.

예

```
&1 // Address C.S.1, definitions will be to axes in this C.S.
#1->X // Motor 1 assigned to X-axis, user units of motor units
#4->2000A // Motor 4 assigned to A-axis, 2000 counts per user unit
#9->333.333ZZ-666.667 // Non-integer values OK

&2 // Address C.S.2, definitions will be to axes in this C.S.
#2->Y // Multiple motors may be assigned to same axis
#3->Y // Both motors will move when Y-axis is commanded

&3 // Address C.S.3, definitions will be to axes in this C.S.
#5->8660X-5000Y // These provide a 30° rotation of X and Y
#6->5000X+8660Y // With 10,000 counts per user unit

&4 // Address C.S.4, definitions will be to axes in this C.S.
// These next two definitions create a 2D Cartesian correcting for
// a 1 arc-minute squareness error
#12->10000X // 10,000 counts per user unit
#13->-29.1X+10000Y // The "-29.1X" term is the correction factor
```

#{상수}->I

기능: 지정된 모터를 인버스 키네마틱스 축으로 정의함

범위: 좌표계 고유

구문: **#{상수}->I**

여기에서

- **{상수}**는 0 부터 [Sys.MaxMotors]-1 범위 내의 정수입니다.

#{상수}->I 명령을 통해 Power PMAC 은 지정된 모터를 주소 지정된 좌표계의 인버스 키네마틱스 축으로 할당합니다. 이처럼 할당된 모터는 프로그램 동작마다 단순한 축 정의 방정식에서가 아닌 좌표계에 대해 사용자가 설명한 인버스 키네마틱스 서브루틴에서 명령 위치를 취득해야 합니다. 인버스 키네마틱스 서브루틴 실행이 종료할 때마다 Power PMAC 은 대상 좌표계에 로컬 변수 'Lxx'로 Motor xx 의 명령 위치를 검색합니다.

지정된 모터가 다른 좌표계의 축에 할당된 경우에 Power PMAC 은 이 명령을 거부하고 에러를 통지합니다. 지정된 모터가 주소 지정된 좌표계의 축에 할당된 경우에 이전의 정의를 새로운 정의로 덮어씁니다.

#{영숫자}->S[{상수}]

기능: 지정된 모터를 주축으로 정의함

범위: 좌표계 고유

구문: **#{상수}->S[{상수}]**

여기에서

- 최초의 **{상수}**는 0 에서 [Sys.MaxMotors]-1 범위 내의 정수입니다.
- 2 번째 옵션의 **{상수}**는 0 또는 1 입니다.

#{상수}->S[{상수}] 명령을 통해 Power PMAC 은 지정된 모터를 주소 지정된 좌표계의 주축으로 할당합니다. 이렇게 정의된 모터는 좌표계의 위치 결정축과는 별도로 단독 모터로 명령할 수 있지만 장애 시의 응답을 위해 좌표계의 일부에 둡니다. 이는 보통 해당 모드의 모터 JOG 이송 또는 오픈 루프 명령을 받게 됩니다.

모터를 스핀들로 사용하려면 모터를 주축으로 정의하지 않아도 됩니다. 어플리케이션 내에서 모터가 스핀들로만 사용되는 경우에 모터를 주축으로 정의하는 이점이 없습니다. 룩 어헤드 버퍼 내에서 메모리가 예약되지만 사용되는 경우는 없습니다. 단, 이 정의는 위치 결정축 및 속도 모드 스핀들로도 사용되는 회전축에서는 유용합니다. 프로그램 속에서 회전 위치 결정축과 주축 정의를 변경하더라도 룩 어헤드 버퍼 등의 좌표계 구조체를 재구성하지 않아도 됩니다.

S의 다음 옵션인 'constant'는 스핀들 모드인 경우의 축 타임 베이스 제어를 나타냅니다. **#4->S**와 같이 **S** 다음에 상수 지정 없이 축이 정의된 경우에 모터는 정의된 좌표계의 타임 베이스를 사용합니다. 이 모드에서는 좌표계의 타임 베이스가 변경되면 스핀들 속도도 변경됩니다. **h [hold]** 명령은 좌표계의 타임 베이스를 강제로 제로로 설정하지만 이 명령을 통해 위치 결정축이 정지하면 스핀들도 정지합니다.

#4->S0처럼 **S** 다음에 **0**이 부가되어 축이 정의된 경우에 모터는 축이 정의된 좌표계에 관계없이 좌표계 **0**의 타임 베이스를 사용합니다. 이 모드에서는 좌표계의 타임 베이스가 변경되더라도 스핀들의 속도는 변경되지 않습니다. **h [hold]** 명령은 좌표계의 타임 베이스를 강제로 제로로 설정하지만 이 명령을 통해 위치 결정축이 정지하더라도 스핀들은 정지하지 않습니다. 단, 이 모드에서는 좌표계 **0**의 타임 베이스값을 변경하면 스핀들의 속도를 대화식으로 변경할 수 있습니다.

#4->S1처럼 **S** 다음에 **1**이 부가되어 축이 정의된 경우에 모터는 축이 정의된 좌표계의 타임 베이스값에 관계없이 고정 타임 베이스값%100을 사용합니다. 이 모드에서는 좌표계의 타임 베이스가 변경되더라도 스핀들의 속도는 변경되지 않습니다. **h [hold]** 명령은 좌표계의 타임 베이스를 강제로 제로로 설정하지만 이 명령을 통해 위치 결정축이 정지하더라도 스핀들은 정지하지 않습니다. 이 모드에서는 좌표계의 타임 베이스값을 변경하여 스핀들의 속도를 대화식으로 변경할 수 없습니다.

이 모드에서는 축 위치에서 모터 위치로 변환이 실시되지 않으므로 이 정의로는 축의 스케일 계수(문자 앞의 'constant') 및 축의 오프셋(문자 뒤의 + 또는 -)을 지정하지 않아도 됩니다. 주축으로 정의된 모터는 **S** 명령에 직접 응답하지 않습니다(단, **S** 명령으로 설정된 변수값을 사용할 수 있습니다).

버퍼 내에서 발생할 가능성이 있는 불일치를 회피하기 위해 좌표계의 룩 어헤드 버퍼의 동작 데이터는 좌표계의 모터 축 정의를 주축으로(또는 주축에서) 변경하기 전에 충분히 퍼지해야 합니다. **lhurge** 명령(2012년 2/4 분기에 출시된 V1.5 펌웨어의 새 기능)은 버퍼의 내용을 클리어하는 최적의 방법입니다.

이 이전 버전인 펌웨어에서는 룩 어헤드 버퍼를 사용하지 않는 동작 모드(Rapid 모드 등)를 선언하여 버퍼 내용을 클리어할 수 있습니다. 이 모드는 새로운 축 정의 준비에서 룩 어헤드 버퍼를 퍼지하기 위해 일시적으로 유효로 설정할 뿐입니다. 다음 프로그램 명령으로 동작 모드를 다른 모드로 돌릴 수 있습니다.

모터의 현재 스핀들 Status는 Status 요소 [Motor[x].SpindleMotor]에 있습니다. 0의 값은 스핀들 모터가 아님을 나타내며 1의 값은 정의된 좌표계 타임 베이스를 사용하는 스핀들 모터, 2의 값은 좌표계 0의 타임 베이스를 사용하는 스핀들 모터, 3의 값은 고정 100% 타임 베이스를 사용하는 스핀들 모터를 나타냅니다.

스핀들 모터에 관한 좌표계의 룩 어헤드 버퍼의 현재 상태는 Status 요소 [Coord[x].LHMotorSlots] 및 [Coord[x].NumMotors]로 평가할 수 있습니다. 요소 [LHMotorSlots]에는 모터의 열수가 테이블의 구조에 포함되고 좌표계의 임의의 형식으로 정의된 모터의 수(주축도 포함)와 같습니다. 요소 [NumMotors]에는 현재 사용된 테이블의 모터 열수(주축은 제외)가 포함됩니다.

예

모터는 보통 [cmd""] 명령을 사용하여 모션 프로그램 또는 (대부분의 경우 G 코드 서브루틴)서브 프로그램에서 위치 결정축과 주축 사이에서 변경됩니다. 이 변경 방법은 다음과 같습니다.

```

dwell 0; // Stop blending and lookahead
lhpurge; // Purge lookahead buffer
Ldata.CmdStatus = 1; // Will change when command executed
cmd"&1#4->S0"; // Define Motor 4 as spindle in CS1
sendallcmds; // Wait for command buffer to empty
do dwell 0; // Loop quickly while waiting
while (Ldata.CmdStatus == 1); // Until command fully executed
// Could check for error here (if < 0)

```

회전 위치 결정축으로 되돌리기 위한 동일한 순서는 다음과 같습니다.

```

dwell 0; // Stop blending and lookahead
lhpurge; // Purge lookahead buffer
Ldata.CmdStatus = 1; // Will change when command executed
cmd"&1#4->100C"; // Define Motor 4 as C-axis in CS1
sendallcmds; // Wait for command buffer to empty
do dwell 0; // Loop quickly while waiting
while (Ldata.CmdStatus == 1); // Until command fully executed
// Could check for error here (if < 0)

```

#{목록}

기능: 다음의 명령을 위한 모터를 지정함

범위: 통신 스레드 고유

구문: **#{목록}**

여기에서

- **{목록}**은 쉼표로 구분된 일련의 정수 상수 또는 2 개의 마침표로 표시된 정수 상수(시작 상수와 종료 상수)에 따른 범위입니다. 모든 상수는 0 에서 [Sys.MaxMotors] -1 의 범위입니다.

#{목록} 명령은 목록 직후의 모터 고유의 명령을 통해 영향을 받는 모터를 지정합니다. 이 구조에서는 1 개의 모터 명령을 여러 모터 상에서 작용할 수 있습니다.

같은 명령행에서 목록 직후에 모터 고유의 명령이 없는 경우에는 이 명령은 에러로 거부됩니다.

#{목록} 명령이 여러 모터에 영향을 주는 것은 직후의 명령만 해당합니다. 이는 해당 통신 스레드에서는 모달에 주소 지정되는 모터는 변경되지 않습니다.

예

```

#1j+ // Modally address Motor 1, command to jog positive
#2,3,4j- // Command Motors 2, 3, and 4 to jog negative
j/ // Stop jog on modally addressed motor (Motor 1)
#5..8,12,14..16hm // Start homing-search move on all motors in list
#3..6 // Specify list without command
error 20:ILLEGAL CMD // Power PMAC rejects with error

```

#{목록}->

기능: 지정된 모터의 축 정의를 통지함

범위: 글로벌

구문: **#{목록}->**

여기에서

- **{목록}**은 헵표로 구분된 1 개 이상의 정수 상수 또는 2 개의 마침표로 표시된 정수 상수(시작 상수와 종료 상수)에 따른 범위입니다. 모든 상수는 0 에서 [Sys.MaxMotors] -1 의 범위입니다.

Power PMAC 은 **#{목록}->**명령으로 지정된 1 대 이상의 모터의 현재 축 정의를 통지할 수 있습니다. 주소 지정된 좌표계에서 모터가 정의된 경우에는 축 정의만 통지됩니다. 다른 좌표계에서 모터가 정의된 경우에 Power PMAC 은 좌표계의 번호 및 좌표계의 정의를 통지합니다.

이 모터의 정의가 어느 좌표계에도 없는 경우에 Power PMAC 은 모터가 좌표계 0(&0)에 속해 있는 것으로 간주하지만 축 정의는 '0'이 아닙니다.

이 명령은 이전의 PMAC 및 Turbo PMAC 의 좌표계와는 약간 다릅니다. 해당 컨트롤러에서는 주소 지정된 좌표계에서 모터가 정의된 경우에 한해 축 정의가 통지됩니다.

예

```
&1 // Address Coordinate System 1
#1-> // Request axis definition of Motor 1
#1->500X // Power PMAC reports definition (in addressed C.S.)

#5-> // Request axis definition of Motor 5
&2#5->500Y // Power PMAC reports definition (in different C.S.)

#1..7-> // Request axis definitions of Motors 1 - 7
#1->500X // Power PMAC reports definition (in addressed C.S.)
#2->500Y // Power PMAC reports definition (in addressed C.S.)
#3->500Z // Power PMAC reports definition (in addressed C.S.)
&2#4->500X // Power PMAC reports definition (in separate C.S.)
#5->500Y // Power PMAC reports definition (in separate C.S.)
#6->500Z // Power PMAC reports definition (in separate C.S.)
&0#7->0 // Power PMAC reports definition (in separate C.S.)
```

#*

기능: 다음의 명령을 위한 모든 모터를 지정함

범위: 통신 스레드 고유

구문: **#***

#* 명령은 목록 직후의 모터 고유 명령으로 모든 활성 모터를 지정합니다. 이 구조에서는 1 개의 모터 명령을 모든 모터 상에서 작용할 수 있습니다.

같은 명령행에서 이 명령 직후에 모터 고유 명령이 없는 경우에는 이 명령은 에러로 거부됩니다.

****** 명령은 모든 모터에 영향을 주는 것은 직후의 명령만 해당합니다. 이는 해당 통신 스레드에서는 모달에 주소 지정되는 모터는 변경되지 않습니다.

예/

#1j+	// Modally address Motor 1, command to jog positive
#*j-	// Command all motors to jog negative
j/	// Stop jog on modally addressed motor (Motor 1)

\$

기능: 모터의 위상기준을 확립함

범위: 모터 고유

구문: \$

\$ 명령을 통해 Power PMAC 은 위상기준을 확립하려고 하거나 또는 PMAC 이 전류(commutation)하고 있는 ([Motor[x].PhaseCtrl]비트 0 [value 1] 또는 비트 2 [value 4] = 1)동기 ([Motor[x].DtOverRotorTc] = 0)모터에 대한 서보 루프를 클로즈합니다. 위상기준을 확립하지 않아도 되는 다른 타입의 모터에서는 **\$** 명령을 사용하여 서보 루프를 클로즈할 수 있습니다(이러한 모터에는 **j/** 명령도 적용할 수 있습니다).

이 명령은 **#1** 처럼 모터 번호의 직전 또는 **#2,4** 나 **#5..8** 처럼 모터 번호의 목록 직전에 설명합니다. 이와 같이 설명하지 않은 경우에는 에러로 거부됩니다.

위상기준을 확립하려면 모터를 여자하여 움직임에서 얻어진 결과로부터 로터 페이즈 각도를 파악하는 페이즈 서치 동작 또는 절대 위치 센서를 읽어 현재의 로터 페이즈 각도를 산출합니다. 페이즈 읽기를 지정하려면 절대 페이즈 위치를 읽기 위한 레지스터의 주소가 포함된 제로 이외의 값에 [Motor[x].pAbsPhasePos]를 설정합니다. 또한 [Motor[x].PhaseFindingTime]은 5 보다 작은 값으로 설정하지 않아도 됩니다.

페이즈 읽기에서는 저장 설정 요소 [Motor[x].pAbsPhasePosFormat], [Motor[x].pAbsPhasePosSf] 및 [Motor[x].AbsPhasePosOffset]을 적절하게 설정하고 올바르게 읽은 데이터를 처리할 수 있어야 합니다.

[Motor[x].PhaseFindingTime]이 5 이상인 경우에는 페이즈 서치 동작이 선택됩니다. 이 동작은 [Motor[x].PhaseFindingDac] 및 [Motor[x].PhaseFindingTime]에 저장된 설정값을 사용합니다. 이러한 값은 페이즈 서치의 메소드(4 추측 또는 스테퍼) 및 파라미터를 지정합니다.

페이즈 서치 동작 또는 절대 페이즈 읽기의 진척은 Status 요소 [Motor[x].PhaseFindingStep]으로 감시할 수 있습니다(서치 또는 읽기가 진행 중인 경우에는 > 0, 그 이외인 경우에는 <= 0). 참조가 실시되면 서치 또는 읽기에서 문제가 발생하지 않았던 경우에 이 요소는 0 으로 설정되고 문제가 발생한 경우에는 음의 수로 설정됩니다. Power PMAC 은 페이즈 서치나 읽기에서 발생할 가능성이 있는 모든 문제를 검출할 수 없으므로 [Motor[x].FatalFeLimit]를 적절하게 설정하는 등 이 이외에도 문제를 방지하는 수단이 필요합니다.

서치 또는 읽기가 성공했는지 여부는 Status 요소 [Motor[x].PhaseFound] 다음에서 확인할 수도 있습니다. Power PMAC 이 올바른 패턴의 충분한 모션이 있고 위상기준을 검출할 수 있다고 판단한 경우에는 1 로 설정됩니다. 그 밖의 경우에는 0 으로 설정됩니다. 이 비트가 1 로 설정되지 않는 경우에 모터는 유효가 되지 않습니다.



경고

처음에 유효한 위상기준을 확립하지 않고 동기적으로 전류(commutation)된 모터 상에서 서보 루프를 클로즈하면 위험한 폭주 상태가 될 수 있습니다.

[Motor[x].PhaseFindingStep]을 1 로 설정하여 서치 또는 읽기 중 하나의 위상기준을 명령할 수 있습니다.

[Motor[x].PowerOnMode]의 비트 0(value 1)은 위상기준이 정상적으로 종료한 후의 모터 상태를 판단합니다. 기본값인 0 으로 설정된 경우에 모터는 정지 상태(오픈 루프, 제로 출력, 앰프 Disable)로 유지됩니다. 1 로 설정된 경우에 모터는 클로즈드 루프가 유효한 상태가 됩니다. 단, Power PMAC 이 서치 또는 읽기에서 문제가 발생한 것으로 판단한 경우에 모터는 이 비트의 설정에 관계없이 정지 상태가 됩니다.

\$\$\$

기능: 컨트롤러를 완전히 리셋함

범위: 글로벌

구문: \$\$\$

\$\$\$ 명령으로 Power PMAC 내의 제어 어플리케이션에서 완전한 리셋을 실행합니다. 어플리케이션 상의 해당 명령 효과는 전원을 일단 OFF 로 설정한 후 ON 으로 설정한 것과 같습니다. 단, 컴퓨터의 하드웨어 및 운영체제는 리셋되더라도 조작을 속행합니다(온라인의 **reboot** 명령으로 컴퓨터를 재시작할 수도 있습니다).

Power PMAC 은 이 명령의 실행 처음에 자신의 인터페이스 ASIC 를 강제로 리셋 모드로 설정하여 모든 출력값을 제로로 설정합니다. 이에 따라 처리의 끝이 다가오면 저장된 설정값을 리로드해야 할 때 동작 모드로 릴리스됩니다.

이 명령 실행 중에 Power PMAC 은 마지막으로 저장된 컨트롤러의 설정값(설정 데이터 구조체 요소, 테이블, 프로그램 등)을 비휘발성 플래시 메모리에서 활성 메모리(RAM)로 복사합니다. 이는 해당 명령을 실행할 때 플래시 메모리에 저장되지 않은 Power PMAC 의 활성 메모리 내 설정 정보가 소실된다는 것을 의미합니다.

이 명령을 통해 저장 설정 요소 중 마지막에 저장한 값이 활성 레지스터에 가장 먼저 복사되고 다음으로 저장한 프로젝트 파일에서 명령이 실행됩니다. 해당 파일의 온라인 명령으로 설정 요소의 저장된 값을 덮어써서 설정 요소에 기록할 수 있습니다.

이 명령을 통해 Power PMAC 어플리케이션은 즉시 전원 셋업/리셋 사이클에 들어갈 수 있으므로 해당 명령에 대해 응답 문자를 보내지 않습니다.

\$\$\$***

기능: 컨트롤러의 완전한 리셋과 재초기화를 실시함

범위: 글로벌

구문: \$\$\$***

\$\$\$*** 명령을 통해 Power PMAC 은 완전한 리셋 및 활성 메모리의 재초기화를 실시할 수 있습니다. 모든 사용자 프로그램 및 그 밖의 버퍼는 활성 메모리 내에서 삭제됩니다. 모든 설정 데이터 구조체 요소는 공장 출하 시의 값으로 리셋됩니다. 플래시 메모리 내에서 마지막으로 저장된 설정은 이 명령을 통해 영향을 받지 않는다는 점과 이 설정은 이 이후의 **\$\$\$** 리셋 명령 또는 전원의 재투입을 통해 활성 메모리로 복원된다는 점에 주의하십시오.

\$\$\$*** 명령 시에 Power PMAC CPU 는 검출한 하드웨어에 따라 기본값 설정을 작성합니다. Power PMAC 시스템의 하드웨어 설정이 변경된 경우에는 수시로 이 명령을 전송할 것을 적극 권장합니다.

Power PMAC 은 이 명령의 실행 처음에 자신의 인터페이스 ASIC 를 강제로 리셋 모드로 설정하여 모든 출력값을 제로로 설정합니다. 이에 따라 처리의 끝이 다가오면 기본값의 설정값을 리로드하기 시작할 때에 동작 모드로 릴리스됩니다.

이 명령을 통해 Power PMAC 어플리케이션은 즉시 전원 셋업/리셋 사이클에 들어갈 수 있으므로 해당 명령에 대해 응답 문자를 보내지 않습니다.

%

기능: 타임 베이스의 오버라이드값을 통지함

범위: 좌표계 고유

구문: %

% 명령을 통해 Power PMAC 은 주소 지정된 좌표계의 현재 타임 베이스 오버라이드값을 통지합니다. 값이 100 인 경우에는 실시간 즉 좌표계에서 동작 속도와 시간이 지정대로임을 의미합니다. 값이 50 인 경우에는 동작 속도가 지정의 절반으로 실시간에서는 동작 시간이 2 배 걸린다는 것을 나타냅니다.

좌표계의 오버라이드를 제어한 것이 무엇인지에 관계없이(소스가 **%{상수}** 명령이 아닌 경우에도) Power PMAC 은 이 명령에 응답하고 오버라이드값을 통지합니다.

축 정의 구문을 통해 모터가 할당되지 않은 좌표계는 항상 %값 0 을 통지합니다.

%{상수}

기능: 타임 베이스의 오버라이드값을 설정함

범위: 좌표계 고유

구문: %{상수}

여기에서

- **{상수}**는 -200.0 ~ +200.0 범위의 부동 소수점 값으로 희망하는 타임 베이스의 오버라이드값을 실시간 비율값으로 지정합니다(+100 은 실시간을 나타냅니다).

%{상수} 명령을 사용하여 주소 지정된 좌표계의 타임 베이스의 오버라이드값을 설정할 수 있습니다. Power PMAC 은 이 명령으로 데이터 구조체 요소 [Coord[x].DesTimeBase]의 값을 설정하고 실제 서보 갱신 시간(설정 데이터 구조체 요소 [Sys.ServoPeriod]의 값으로 정의된 밀리초)에 **{상수}**의 비율을 곱한 값으로 설정합니다.

(기본값)[Coord[x].DesTimeBase]의 주소가 포함된 포인터 변수 [Coord[x].pDesTimeBase]로 타임 베이스값에 해당 레지스터를 사용하라고 좌표계가 지정된 경우에 좌표계 타임 베이스를 제어하기 위해 해당 값이 사용됩니다. 요소 [Coord[x].TimeBase]에서는 실제 타임 베이스값은 요소 [Coord[x].TimeBaseSlew]로 설정한 비율로 해당 값에 대해 변화합니다. 각각의 서보 갱신 주기(물리적으로는 고정 주기)에 Power PMAC 은 좌표계의 각 모터에 대해 명령된 궤적에 따라 요소 [Coord[x].TimeBase] 값씩 시간을 늘립니다. 사용되는 요소 [Coord[x].TimeBase]의 값이 대상의 [Coord[x].DesTimeBase] 요소값과 같아지면 슬루잉(slewing)이 정지합니다.

명령으로 값이 +100 인 경우에는 실시간 즉 좌표계 상의 동작 속도와 시간이 지정대로임을 의미합니다. 값이 50 인 경우에는 동작 속도가 지정의 절반으로 실시간에서는 동작 시간이 2 배 걸린다는 것을 나타냅니다. 값이 0 인 경우에는(hold 명령과 마찬가지로) 좌표계의 시간이 프리즈됩니다. 값이 -200 미만 또는 +200 을 초과하는 경우에는 에러로 거부됩니다.

2014 년 1/4 분기에 출시된 V1.6 펌웨어부터 모터의 트레이스 버퍼가 저장 설정 요소 [Motor[x].TraceSize]로 정의된 경우에 궤적에 따른 반전을 위한 음의 타임 베이스값이 지원됩니다.

프로그램 내에서는 요소 [Coord[x].DesTimeBase]에 값을 쓰면 동등한 효과를 얻을 수 있습니다. 예를 들어 다음의 프로그램 명령

Coord[1].DesTimeBase = Sys.ServoPeriod * MyPercent / 100

은 좌표계 1 의 비율값을 선언한 사용자 변수 'MyPercent'의 값으로 설정합니다(이 타입의 명령은 온라인 명령으로 사용하여 변수값에 대한 퍼센트의 오버라이드를 설정할 수도 있습니다). 이와 같이 설정한 경우에 값은 +/-200% 범위에는 제한되지 않습니다.

&

기능: 현재 스레드의 주소 지정된 좌표계를 통지함

범위: 통신 스레드 고유

구문: **&**

& 명령을 사용하면 Power PMAC 은 이 명령이 송신되는 통신 스레드에 대해 주소 지정된 좌표계의 수를 반환합니다. 다른 모터가 **&{상수}**명령으로 주소 지정될 때까지 이 좌표계가 대상 스레드 내에서 송신되는 이후의 모터 고유 명령으로 작용합니다.

이 명령은 스레드의 데이터 구조체 요소 [Ldata.Coord]의 값을 조사합니다. 이는 **Ldata.Coord** 명령을 사용하여 요소에 직접 쿼리하는 것과 같습니다.

다른 통신 스레드가 동시에 다른 좌표계를 주소 지정하는 중일 가능성이 있습니다.

전원 셋업/리셋 시에는 Power PMAC 은 기본값으로 좌표계 0(**&0**)을 주소 지정합니다.

&{상수}

기능: 스레드가 주소 지정한 좌표계를 선택함

범위: 통신 스레드 고유

구문: **&{상수}**

여기에서

- **{상수}**는 0에서 [Sys.MaxCoords]-1 범위 내의 정수입니다.

&{상수} 명령을 사용하면 **{상수}**가 지정한 좌표계가 해당 명령을 송신한 통신 스레드를 주소 지정하는 좌표계가 됩니다. 다른 좌표계가 **&{상수}** 명령으로 주소 지정될 때까지 이 좌표계가 대상 스레드 내에서 송신되는 후속 좌표계 고유 명령에 작용합니다.

이 명령은 스레드의 데이터 구조체 요소 [Ldata.Coord]의 값을 설정합니다. 이는 **Ldata.Coord={식}** 명령을 사용하여 요소를 직접 설정하는 것과 같습니다.

&{목록} 명령을 사용하여 목록 직후의 좌표계 고유 명령으로 기능하는 여러 좌표계를 지정할 수 있습니다. 주소 지정된 좌표계가 목록 내에 없는 경우에는 그 좌표계는 명령으로는 기능하지 않습니다. **&{목록}** 명령에서는 이후의 명령으로 주소 지정되는 좌표계를 변경할 수 없습니다.

다른 통신 스레드가 동시에 다른 좌표계를 주소 지정하는 중일 가능성이 있습니다.

예

```
&1b1r      // Address C.S. 1, command it to run from the beginning of PROG 1
q          // Command addressed C.S. (&1) to quit running program
&2%50      // Address C.S. 2, command it to set override of 50%
&3..6a     // Command C.S. 3 - 6 to abort programs and moves
%100       // Command addressed C.S. (&2) to set override of 100%
```

&{목록}

기능: 다음의 명령을 위한 좌표계를 지정함

범위: 통신 스레드 고유

구문: **&{목록}**

여기에서

- **{목록}**은 쉼표로 구분된 일련의 정수 상수 또는 2 개의 마침표로 표시된 정수 상수(시작 상수와 종료 상수)에 따른 범위입니다. 모든 상수는 0에서 [Sys.MaxCoords] -1의 범위입니다.

&{목록} 명령은 목록 직후의 좌표계 고유 명령을 통해 영향을 받는 좌표계를 지정합니다. 이 구조에서는 1개의 좌표계 명령을 여러 좌표계 상에서 작용할 수 있습니다.

같은 명령행에서 목록 직후에 좌표계 고유 명령이 없는 경우에는 이 명령은 에러로 거부됩니다.

&{목록} 명령이 여러 좌표계에 영향을 주는 것은 직후의 명령만 해당합니다. 이는 해당 통신 스레드에서는 모달에 주소 지정되는 모터는 변경되지 않습니다.

예

&1r	// Modally address C.S. 1, command to run selected program
&2,3,4a	// Command C.S. 2, 3, and 4 to abort programs and moves
%50	// Set override of 50% on modally addressed C.S. (&1)
&5..8,12,14..16h	// Command hold on all C.S.'s in list
&3..6	// Specify list without command
error 20:ILLEGAL CMD	// Power PMAC rejects with error

&*

기능: 다음 명령을 위한 모든 좌표계를 지정함

범위: 통신 스레드 고유

구문: **&***

&* 명령은 목록 직후의 좌표계 고유 명령을 통해 모든 활성 좌표계가 영향을 받도록 지정합니다. 이 구조체에서는 1 개의 좌표계 명령이 모든 좌표계 상에서 작용하도록 할 수 있습니다.

같은 명령행에서 이 명령 직후에 좌표계 고유 명령이 없는 경우에는 이 명령은 에러로 거부됩니다.

&* 명령으로 모든 좌표계를 지정하더라도 명령 직후의 좌표계만 영향을 받습니다. 이 통신 스레드에서는 모달에 주소 지정되는 좌표계는 변경되지 않습니다.

예

&1b1r	// Address C.S. 1, command it to run from the beginning of PROG 1
&*a	// Command all active C.S. to abort programs and moves
b2r	// Command C.S 1 to run from beginning of PROG 2

기능: 룩 어헤드 급정지

범위: 좌표계 고유

구문: ****

**** 명령을 통해 Power PMAC 은 지정된 좌표계(좌표계의 모든 모터는 가속도 제한을 위반하지 않음)의 룩 어헤드 버퍼 내에서 급정지를 계산 및 실행합니다. 모션은 프로그램된 경로에 따라 제어 정지를 계속하지만 프로그램 포인트에서 반드시 정지하는 것은 아닙니다. 좌표계 목록이 없는 경우에 앞서 말한 동작은 주소 지정된 좌표계에 적용됩니다. 좌표계 목록이 있는 경우에 앞서 말한 동작은 목록 내의 모든 좌표계에 적용됩니다.

**** 급정지 명령은 일반적으로 룩 어헤드에서 모션을 대화적으로 정지시키기 위한 최적의 명령 입니다. 이 기능은 기존 피드 홀드 명령 기능과 비슷하지만 Power PMAC 의 표준적인 **h** 피드 홀드 명령과는 달리 이 명령에서는 가속도 제한이 준수됩니다.

일단 정지하면 다음의 옵션이 가능합니다.

- JOG 이송 명령 중 하나로 축을 JOG 이송합니다. 온라인 JOG 명령을 사용하여 정지에서 좌표계 중 하나의 모터를 JOG 어웨이할 수 있습니다. 단, 프로그램된 경로 실행을 재개할 수 있도록 하기 전에 **j**=(JOG 리턴) 명령으로 모든 모터를 원래 정지점으로 복귀시켜야 합니다.
- **<** 명령으로 경로에 따라 역행을 시작합니다.
- **>**, **r** 또는 **s** 명령으로 정방향 실행을 재개합니다.
- **a** 명령으로 프로그램 실행을 종료합니다.

세그먼트화된 룩 어헤드 버퍼 내에서 동작을 실행하지 않는 좌표계에 대해 **** 명령이 지정되면 Power PMAC 은 대신에 **h**(피드 홀드) 명령을 실행합니다.

동등한 버퍼 프로그램의 다이렉트 명령은 **lh**입니다.

<

기능: 룩 어헤드 버퍼 내에서 역행을 시작함

범위: 좌표계 고유

구문: **<**

<명령을 통해 Power PMAC 은 지정된 좌표계에 대해 룩 어헤드 버퍼 내에서 역행을 시작합니다. 프로그램이 정방향으로 실행된 경우에 처음에(**** 명령과 등가인)급정지가 실행됩니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다.

전진 운동(존재하는 경우)의 감속 및 역방향 가속에서는 **[Motor[x].InvAmax]**의 가속 제한을 준수합니다.

실행은 룩 어헤드 버퍼에 버퍼된 포인트에 따라 역방향으로 진행하고 순방향과 동일하게 감속 및 가속 제한을 준수합니다. 이 실행은 다음 중 하나가 발생할 때까지 계속 실행합니다.

- 가장 오래된 저장점이 룩 어헤드 버퍼에 남아 있는 상태로 역행이 룩 어헤드 버퍼의 선두에 도달하고 가속 제한을 준수하여 정지를 위해 감속
- ****(급정지) 명령이 지정되어 룩 어헤드 버퍼에서 Power PMAC 이 급속하게 정지
- **>**(재개-정방향), **r**(실행) 또는 **s**(스텝) 명령이 지정되어 Turbo PMAC 이 프로그램 및 필요에 따라 룩 어헤드 버퍼의 정방향 실행을 재개
- 에러 상태 발생 또는 회복 불가능한 정지 명령이 지정됨

급정지점에서 어느 한 모터가 JOG 이송되고 **j**= 명령으로 복귀되지 않는 경우에는 Power PMAC 은 **<**(백업) 명령을 거부하고 에러를 통지합니다.

세그먼트화된 룩 어헤드 버퍼 내에서 동작을 실행하지 않는 좌표계에 대해 **<**명령이 지정되면 Power PMAC 은 대신에 **h**(피드 홀드) 명령을 실행합니다.

동등한 버퍼 프로그램의 다이렉트 명령은 **lh<**입니다.

>

기능: 룩 어헤드 버퍼에서 정방향 실행을 재개함

범위: 좌표계 고유

구문: >

> 명령을 통해 Power PMAC 은 주소 지정된 좌표계에 룩 어헤드 버퍼 내에서 정방향 실행을 재개합니다. 이는 보통 \(\급정지) 명령 또는 <(백업) 명령 뒤의 보통의 처리를 재개하기 위해 사용됩니다. 프로그램이 역방향으로 실행된 경우에 처음에 \(\ 명령과 등가인) 급정지가 실행됩니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다.

앞의 정방향 실행이 **r** 명령으로 시작되는 계속 모드인 경우에 > 명령은 계속 모드로 실행을 재개합니다. 앞의 정방향 실행이 **s** 명령으로 시작되는 싱글 스텝 모드인 경우에 > 명령은 싱글 스텝 모드로 실행을 재개합니다. 정방향 실행을 재개하기 위해 **r** 명령과 **s** 명령을 사용할 수 있지만 이러한 명령을 통해 계속/싱글 스텝 모드가 변경될 가능성이 있습니다.

역행(존재하는 경우)에서 감속 및 정방향에서 가속 시에 [Motor[x].InvAmax] 가속 제한을 준수합니다.

급정지점에서 어느 한 모터가 JOG 이송 되어 **j=** 명령으로 복귀되지 않는 경우에는 Power PMAC 은 >(재개) 명령을 거부하고 에러를 통지합니다.

세그먼트화된 룩 어헤드 버퍼 내에서 동작을 실행하지 않는 좌표계에 대해 \ 명령이 지정되면 Power PMAC 은 대신에 **r**(실행) 명령을 실행합니다.

동등한 버퍼 프로그램의 다이렉트 명령은 **lh>**입니다.

?

기능: Status 의 워드값을 통지함

범위: 글로벌, 모터/좌표계 고유

구문: ? (글로벌 Status)
 #{목록}? (모터 Status)
 &{목록}? (좌표계 Status)

? 명령으로 Power PMAC, 글로벌 Power PMAC 시스템 Status 워드의 현재값 또는 지정된 모터 또는 좌표계의 현재값을 통지합니다. 모터 목록 또는 좌표계 목록이 없는 경우에는 글로벌 Status 워드값이 통지됩니다. 모터 목록이 있는 경우에는 목록 내의 모든 모터에 대한 Status 워드값이 통지됩니다. 좌표계 목록이 있는 경우에는 목록 내의 모든 좌표계에 대한 Status 워드값이 통지됩니다.

여러 모터 목록 또는 여러 좌표계 목록을 지정해도 후속 모터 고유의 명령 또는 좌표계 고유의 명령으로 모달에 주소 지정된 모터 또는 좌표계는 변경되지 않습니다.

모터 및 좌표계의 Status 값은 16 자리의 16 진법 ASCII 문자로 통지되고 32 비트의 2 개의 Status 워드를 나타냅니다. 글로벌 Status 의 값은 8 자리의 16 진법 ASCII 문자로 통지되고 32 비트의 1 개의 Status 워드를 나타냅니다.

모터의 Status

모터의 Status 에 통지되는 Status 값이 각 Status 비트를 어떻게 나타내는지는 다음 표와 같습니다. 좌측의 2 열은 이 명령에 대한 16 진법의 ASCII 응답에서 Status 비트를 어떻게 해석할 수 있는지를 나타냅니다. 그 다음 열은 Status 비트의 정보에 관한 기본적인 설명입니다. 그 다음 2 열은 데이터 구조체 요소 [Motor[x].Status[i]]의 수치에서 Status 비트를 어떻게 해석할 수 있는지를 나타내고 처음 열은 비트의 데이터 구조체 요소 이름을 나타냅니다.

명령의 요소명 [backup Motor[x].Status]를 사용하여 같은 정보를 텍스트 형식으로 취득할 수 있습니다. 각 요소명은 현재의 값으로 복귀하여 통지됩니다.

Char #	16 진수값	설명	Word #	Bit #	요소명: [Motor[x].]
1	8	진행 중의 서치 동작 트리거	0	31	TriggerMove
1	4	진행 중의 원점 서치 동작	0	30	HomeInProgress
1	2	하드웨어 음의 리미트	0	29	MinusLimit
1	1	하드웨어 양의 리미트	0	28	PlusLimit
2	8	경고 위치 편차	0	27	FeWarn
2	4	편차 이상	0	26	FeFatal
2	2	하드웨어 위치 리미트에서 정지	0	25	LimitStop
2	1	앰프 폴트 에러	0	24	AmpFault
3	8	소프트웨어 음의 리미트	0	23	SoftMinusLimit
3	4	소프트웨어 양의 리미트	0	22	SoftPlusLimit
3	2	적분 전류(I^2T) 폴트	0	21	I2tFault
3	1	트리거가 발견되지 않음	0	20	TriggerNotFound
4	8	앰프 경고	0	19	AmpWarn
4	4	센서 소실 에러	0	18	EncLoss
4	2	(예약 완료)	0	17	
4	1	(예약 완료)	0	16	
5	8	원점 복귀 완료	0	15	HomeComplete
5	4	명령 속도 제로	0	14	DesVelZero
5	2	클로즈드 루프 모드	0	13	ClosedLoop
5	1	앰프 Enable	0	12	AmpEna
6	8	인포지션	0	11	InPos
6	4	(예약 완료)	0	10	
6	2	블록 요구 플래그	0	9	BlockRequest
6	1	위상기준의 확립	0	8	PhaseFound
7	8	트리거 장착 동작 속도의 선택	0	7	TriggerSpeedSel
7	4	갠트리 원점 복귀 완료	0	6	GantryHomed
7	2	스핀들 모터 정의 비트 1	0	5	SpindleMotor(bit 1)
7	1	스핀들 모터 정의 비트 0	0	4	SpindleMotor(bit 0)
8	8	(예약 완료)	0	3	
8	4	(예약 완료)	0	2	
8	2	(예약 완료)	0	1	
8	1	(예약 완료)	0	0	
9	8	PMATCH 계산에서 사용한 모터	1	31	Csolve
9	4	소프트웨어 위치 리미트에서 정지	1	30	SoftLimit
9	2	서보 출력 제한	1	29	DacLimit
9	1	백래시 방향	1	28	BIDir

10	8	SW 제한에 따른 동작 변경	1	27	SoftLimitDir
10	4	(예약 완료)	1	26	
10	2	(예약 완료)	1	25	
10	1	(예약 완료)	1	24	
11	8	(예약 완료)	1	23	
11	4	(예약 완료)	1	22	
11	2	(예약 완료)	1	21	
11	1	(예약 완료)	1	20	
12	8	(예약 완료)	1	19	
12	4	(예약 완료)	1	18	
12	2	(예약 완료)	1	17	
12	1	(예약 완료)	1	16	
13	8	(예약 완료)	1	15	
13	4	(예약 완료)	1	14	
13	2	(예약 완료)	1	13	
13	1	(예약 완료)	1	12	
14	8	(예약 완료)	1	11	
14	4	(예약 완료)	1	10	
14	2	(예약 완료)	1	9	
14	1	(예약 완료)	1	8	
15	8	(예약 완료)	1	7	
15	4	(예약 완료)	1	6	
15	2	(예약 완료)	1	5	
15	1	(예약 완료)	1	4	
16	8	(예약 완료)	1	3	
16	4	(예약 완료)	1	2	
16	2	(예약 완료)	1	1	
16	1	(예약 완료)	1	0	

첫 문자:

bit 31 트리거 서치 중 동작(TriggerMove): 이 비트는 트리거까지의 동작 시작(원점 서치, 트리거까지의 JOG 이송, 트리거까지의 프로그램 Rapid 모드 동작)에서 1로 설정됩니다. 사전에 지정된 트리거 상태가 검출되고 트리거 후의 동작이 시작된 경우 또는 트리거가 검출되지 않고 동작이 종료한 경우에는 0으로 설정됩니다.

bit 30 원점 서치 중 동작(HomeInProgress): 이 비트는 첫 원점 서치 동작에 1로 설정됩니다. 사전에 지정된 트리거 상태가 검출되고 트리거 후의 동작이 시작된 경우 또는 트리거가 검출되지 않고 동작이 종료한 경우에는 0으로 설정됩니다.

bit 29 하드웨어의 음의 리미트(MinusLimit): 이 비트는 모터가 하드웨어의 음의 리미트에 도달한 경우에 1로 설정됩니다. 그 밖에 모터가 이전에 해당 리미트에 도달하더라도 현재는 도달해 있지 않으면 0으로 설정됩니다.

bit 28 하드웨어의 양의 리미트(PlusLimit): 이 비트는 모터가 하드웨어의 양의 리미트에 있는 경우에 1로 설정됩니다. 그 밖에 모터가 이전에 해당 리미트에 도달하더라도 현재는 도달해 있지 않으면 0으로 설정됩니다.

2 번째 문자:

bit 27 경고 위치 편차 (FeWarn): 모터의 위치 편차 크기가 [Motor[x].WarnFeLimit]로 설정한 경고 위치 편차 리미트를 초과한 경우에 이 비트는 1이 됩니다. 위치 편차 크기가 이 제한 미만인 경우 또는 편차 이상 리미트를 초과했기 때문에 모터가 무효로 설정된 경우에는 이 비트는 0이 됩니다.

bit 26 편차 이상(FeFatal): 모터의 위치 편차 크기가 [Motor[x].FatalFeLimit]로 설정한 편차 이상 리미트를 초과했기 때문에 모터가 무효로 설정된 경우에 이 비트는 1 이 됩니다. 그 밖에는 0 이며 모터가 다시 Enable 된 경우에도 0 이 됩니다.

bit 25 하드웨어의 위치 리미트에서 정지(LimitStop): 양 또는 음의 하드웨어 오버트래블 리미트에 도달해서 모터가 정지한 경우에 현재는 그 리미트에 도달해 있지 않더라도 이 비트는 1 이 됩니다. 그 밖에 모두 리미트에 도달했다가 벗어난 경우도 포함하여 이 비트는 0 이 됩니다.

bit 24 앰프 폴트 에러(AmpFault): (앰프 폴트의 시그널 상태가 존재하지 않더라도) 앰프 폴트 에러 때문에 모터가 무효로 설정된 경우 또는 산출된 I^2T 의 적분 전류 폴트때문에(이 경우에는 비트 21 도 설정됨) 모터가 무효가 된 경우에는 이 비트는 1 이 됩니다. 그 밖에는 0 이며 모터가 다시 Enable 된 경우에도 0 이 됩니다.

3 번째 문자:

bit 23 소프트웨어의 음의 리미트(SoftMinusLimit): [Motor[x].MinPos]로 설정한 소프트웨어의 음의 리미트에 모터가 도달했거나 그것을 초과하면 이 비트는 1 로 설정됩니다(활성화하려면 [Motor[x].MaxPos]보다 작은 값으로 설정해야 합니다). 그 밖에 모터가 이전에 해당 리미트에 도달하더라도 현재는 도달해 있지 않으면 0 으로 설정됩니다.

bit 22 소프트웨어의 양의 리미트(SoftPlusLimit): [Motor[x].MaxPos]로 설정한 소프트웨어의 양의 리미트에 모터가 도달했거나 그것을 초과하면 이 비트는 1 로 설정됩니다(활성화하려면 [Motor[x].MinPos]보다 큰 값으로 설정해야 합니다). 그 밖에 모터가 이전에 해당 리미트에 도달하더라도 현재는 도달해 있지 않으면 0 으로 설정됩니다.

bit 21 적분 전류(I^2T) 폴트(I2tFault): [Motor[x].I2tTrip]으로 설정한 적분 전류 리미트를 초과하여 모터가 무효가 된 경우에 이 비트는 0 으로 설정됩니다. 이 경우에 앰프 폴트의 비트(비트 24)도 설정됩니다. 그 밖에는 모두 0 이며 모터가 다시 Enable 된 경우에도 0 이 됩니다(적분 전류 폴트 계산을 통해 앰프 폴트가 발생한 경우에 이 비트는 설정되지 않는다는 점에 주의하십시오).

bit 20 트리거가 발견되지 않음(TriggerNotFound): 트리거까지의 JOG 이송 또는 트리거까지 프로그램 Rapid 모드 동작이 사전에 지정된 트리거 상태가 검출되지 않고 종료된 경우에 이 비트는 1 로 설정됩니다. 이는 에러 상태는 아니지만 후속 처리가 많은 경우에 해당 비트의 설정 여부에 따라 다릅니다. 그 밖에는 모두 이 비트는 0 이며 다음 동작이 시작될 때 0 으로 복귀합니다.

4 번째 문자:

bit 19 앰프 경고(AmpWarn): [Motor[x].AmpFaultLevel]의 비트 1(값 2)이 1 로 설정된 경우에 이 비트는 1 로 설정됩니다. 에러를 트리거하기 위해 지정된 폴트 상태에서 앰프 폴트 비트를 연속으로 2 회 읽어야 하지만 앰프 폴트 비트의 하나를 읽었을 때 폴트 상태였습니다.

bit 18 .. 16 {추후 사용하기 위해 예약}

5 번째 문자:

bit 15 위치 참조의 확립(HomeComplete): 원점 서치 동작 또는 절대 위치 읽기 중 어느 하나에서 모터에 대해 위치 참조가 올바르게 확립된 경우에 이 비트는 1로 설정됩니다. 전원 셋업/리셋 시 및 첫 원점 서치 동작에는 자동으로 0으로 설정됩니다. 원점 서치 동작에서는 사전에 지정된 트리거 조건이 발견된 경우에 동작 트리거 후의 부분이 완료되기 전에 0으로 설정됩니다.

bit 14 명령 속도 제로(DesVelZero): 모터가 클로즈드 루프 제어에서(위치를 유지하려고 하는 등일 때에) 명령 속도가 제로로 설정된 경우 또는 모터가 오픈 루프 모드(유효 또는 무효)로 현재 속도가 제로와 같은 경우에 이 비트는 1로 설정됩니다. 모터가 클로즈드 루프 모드이고 명령 속도가 제로 이외인 경우 또는 모터가 오픈 루프 모드이고 현재 속도가 제로 이외인 경우에는 이 비트는 0이 됩니다.

bit 13 클로즈드 루프 모드(ClosedLoop): 모터가 클로즈드 루프 제어 상태이면 이 비트는 1로 설정됩니다. 모터가 오픈 루프 모드(유효 또는 무효) 상태이면 0이 됩니다.

bit 12 앰프 Enable(AmpEna): 모터가 유효인(클로즈드 루프 제어 또는 오픈 루프 제어) 경우에 이 비트는 1로 설정됩니다. 이 경우에는 활성 앰프의 Enable 출력 신호가 아니어도 된다는 점에 주의하십시오. 모터가 무효인 경우에 이 비트는 0이 됩니다.

6 번째 문자:

bit 11 인포지션(InPos): 인포지션의 모든 조건을 만족한 경우에 이 비트는 1로 설정됩니다. 모든 조건이란, 모터가 클로즈드 루프이고 명령 속도가 제로이며 이동 타이머가 활성이 아니고(동작 없음, 드웰 또는 지연 중), 위치 편차의 크기가 [Motor[x].InPosBand] 이하이며 이러한 모든 조건이([Motor[x].InPosTime] -1)인 연속 서보 사이클에 대해 True임을 말합니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 10 {추후 사용하기 위해 예약}

bit 9 블록 요구 플래그(BlockRequest): 모터가 새로운 동작 섹션에 들어간 시점에서 모션 신호의 다음 동작 섹션식을 계산하라는 요청이 있는 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다. 이는 주로 내부에서 사용됩니다.

bit 8 위상기준의 확립(PhaseFound): 동기적인 ([Motor[x].DtOverRotorTc] = 0.0)의 Power PMAC에 따라 전류(commutation)되는 모터의 전원 셋업/리셋 시([Motor[x].PhaseCtrl]의 bit 0 또는 bit 2 = 1)에 이 비트는 0으로 설정됩니다. 위상기준 동작 또는 절대 위치 읽기 중 어느 하나에서 모터에 대해 위상기준이 올바르게 확립된 경우에 이 비트는 1로 설정됩니다.

7 번째 문자:

bit 7 속도 선택 트리거(TriggerSpeedSel): [Motor[x].MaxSpeed]로 지정된 속도로 동작이 실시된 경우에 트리거까지의 동작 중 이 비트는 1로 설정됩니다. [Motor[x].JogSpeed]로 지정된 속도로 동작이 실시된 경우에는 0으로 설정됩니다.

bit 6 갠트리의 원점 복귀 완료(GantryHomed): 이 모터가 주축/종축 갠트리 시스템의 종축일 때 주축과 종축의 모터 원점 트리거가 검출된 경우 및 모터 간의 스쿠가 완전히 삭제된 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 5 스피들 정의 비트 1(SpindleMotor 비트 1): 이 모터가 주축으로 정의되고 좌표계 0 의 타임 베이스(**S0**) 또는 100% 고정 타임 베이스(**S1**) 중 하나를 사용한 경우에 이 비트는 1 로 설정됩니다. 모터가 주축으로 정의되지 않은 경우 또는 정의된 좌표계의 타임 베이스(**S**)를 사용하여 주축으로 정의된 경우에는 0 으로 설정됩니다.

bit 4 스피들 정의 비트 0(SpindleMotor 비트 0): 이 모터가 주축으로 정의되고 정의된 좌표계의 타임 베이스(**S**) 또는 100% 고정 타임 베이스(**S1**) 중 하나를 사용한 경우에 이 비트는 1 로 설정됩니다. 모터가 주축으로 정의되지 않은 경우 또는 좌표계 0 인 타임 베이스(**S0**)를 사용하여 주축으로 정의된 경우에는 0 으로 설정됩니다.

8 번째 문자:

bit 3 .. 0 {추후 사용하기 위해 예약}

9 번째 문자:

bit 31 PMATCH 계산에서 사용한 모터(Csolve): 이 모터의 위치를 사용하여 좌표계 'pmatch'(위치 일치) 함수로 (모션 프로그램 시작 시, **pmatch** 명령 실행 시, 축 위치/속도/편차 쿼리 시)축 위치를 계산하는 경우에 이 비트는 1 로 설정됩니다. 모터가 Null 정의 또는 여유 축 정의이므로 이러한 계산에서는 사용되지 않는 경우에는 0 으로 설정됩니다.

bit 30 소프트웨어의 위치 리미트에서 정지(SoftLimit): 모터가 정지했거나 양 또는 음의 소프트웨어 오버트래블 리미트에 도달했거나 장래적으로 도달하기 위해 현재의 동작이 정지할 예정인 경우에 현재는 리미트에 도달하지 않더라도 이 비트는 1 이 됩니다. 그 밖에 모두 리미트에 도달했다가 벗어난 경우도 포함하여 이 비트는 0 이 됩니다.

bit 29 서보 출력 제한(DacLimit): 모터의 서보 출력 명령값이 [Motor[x].MaxDac]의 값에 도달한 경우에 이 비트는 1 이 됩니다. 그 밖의 경우에는 0 으로 설정됩니다.

bit 28 백래시 방향(BIDir): 모터의 백래시 기능이 유효이고 모터가 음의 방향으로 위치 이동을 실행 중이거나 방금 실행한 경우에 이 비트는 1 이 됩니다. 그 밖의 경우에는 0 으로 설정됩니다.

10 번째 문자:

bit 27 엔코더 소실 에러(EncLoss): (현재는 에러 상태가 아니더라도)엔코더 소실 에러로 이 모터가 무효로 설정된 경우에는 이 비트는 1 로 설정됩니다. 그 밖에는 0 이며 모터가 다시 Enable 된 경우에도 0 이 됩니다.

bit 26 소프트웨어 리미트 방향(SoftLimitDir): 음의 소프트웨어 리미트에 따라 소프트웨어의 위치 리미트가 영향을 받고 마지막 동작이 정지, 수정 또는 거부된 경우에 이 비트는 1 로 설정됩니다. 이와 같은 동작이 양의 소프트웨어 리미트에 따라 제한된 경우 또는 전원 셋업/리셋 이후 이와 같은 리미트가 발생하지 않는 경우에는 이 비트는 0 이 됩니다.

bit 25 .. 24 {추후 사용하기 위해 예약}

11 번째 문자:

bit 23 .. 20 {추후 사용하기 위해 예약}

12 번째 문자:

bit 19 .. 16 {추후 사용하기 위해 예약}

13 번째 문자:

bit 15 .. 12 {추후 사용하기 위해 예약}

14 번째 문자:

bit 11 .. 8 {추후 사용하기 위해 예약}

15 번째 문자:

bit 7 .. 4 {추후 사용하기 위해 예약}

16 번째 문자:

bit 3 .. 0 {추후 사용하기 위해 예약}

좌표계의 Status

좌표계의 Status 에 통지되는 Status 값이 각 Status 비트를 어떻게 나타내는지 다음 표와 같습니다. 좌측의 2 열은 이 명령에 대한 16 진법의 ASCII 응답에서 Status 비트를 어떻게 해석할 수 있는지를 나타냅니다. 그 다음 열은 Status 비트의 정보에 관한 기본적인 설명입니다. 그 다음의 2 열은 데이터 구조체 요소 [Coord[x].Status[i]]의 수치에서 Status 비트를 어떻게 해석할 수 있는지를 나타내고 마지막 열은 비트의 데이터 구조체 요소 이름을 나타냅니다.

명령 [backup Coord[x].Status]에서 요소명을 사용하여 같은 정보를 텍스트 형식으로 취득할 수 있습니다. 각 요소명은 현재의 값으로 복귀하여 통지됩니다.

Char #	16 진수값	설명	Word #	Bit #	요소명: [Coord[x].]
1	8	진행 중의 서치 동작 트리거(좌표계 내의 어느 한 모터)	0	31	TriggerMove
1	4	진행 중의 원점 서치 동작(좌표계 내의 어느 한 모터)	0	30	HomeInProgress
1	2	하드웨어의 음의 리미트(좌표계 내의 어느 한 모터)	0	29	MinusLimit
1	1	하드웨어의 양의 리미트(좌표계 내의 어느 한 모터)	0	28	PlusLimit
2	8	경고 위치 편차 (좌표계 내의 어느 한 모터)	0	27	FeWarn
2	4	편차 이상(좌표계 내의 어느 한 모터)	0	26	FeFatal
2	2	하드웨어 리미트에서 정지(좌표계 내의 어느 한 모터)	0	25	LimitStop
2	1	앰프 폴트 (좌표계 내의 어느 한 모터)	0	24	AmpFault
3	8	소프트웨어의 음의 리미트(좌표계 내의 어느 한 모터)	0	23	SoftMinusLimit
3	4	소프트웨어 양의 리미트(좌표계 내의 어느 한 모터)	0	22	SoftPlusLimit
3	2	적분 전류(I^2T) 폴트(좌표계 내의 어느 한 모터)	0	21	I2tFault
3	1	트리거가 발견되지 않음(좌표계 내의 어느 한 모터)	0	20	TriggerNotFound
4	8	앰프 폴트 (좌표계 내의 어느 한 모터)	0	19	AmpWarn
4	4	엔코더 소실 에러(좌표계 내의 어느 한 모터)	0	18	EncLoss
4	2	(예약 완료)	0	17	
4	1	이동 타이머 유효화	0	16	TimerEnabled

5	8	원점 복귀 완료(좌표계 내의 모든 모터)	0	15	HomeComplete
5	4	원점 복귀 완료(좌표계 내의 모든 모터)	0	14	DesVelZero
5	2	클로즈드 루프 모드(좌표계 내의 모든 모터)	0	13	ClosedLoop
5	1	앰프 Enable(좌표계 내의 모든 모터)	0	12	AmpEna
6	8	인포지션 (좌표계 내의 모든 모터)	0	11	InPos
6	4	(예약 완료)	0	10	
6	2	블록 요구 플래그	0	9	BlockRequest
6	1	타이머 Enable	0	8	TimersEnabled
7	8	XX/YY/ZZ 축의 원의 반경 에러/에러 워드 비트 7	0	7	RadiusError (비트 1) ErrorStatus (비트 7)
7	4	X/Y/Z 축 Circle 반경 에러/에러 워드 비트 6	0	6	RadiusError (비트 0) ErrorStatus (비트 6)
7	2	소프트웨어 위치 리미트에서 정지(좌표계 내의 어느 한 모터)/에러 워드 비트 5	0	5	SoftLimit ErrorStatus(비트 5)
7	1	런타임 에러/에러 워드 비트 4	0	4	RunTimeError ErrorStatus(비트 4)
8	8	PVT 모드 에러/에러 워드 비트 3	0	3	PvtError ErrorStatus(비트 3)
8	4	Linear-to-PVT 모드 에러/에러 워드 비트 2	0	2	LinToPvtError ErrorStatus(비트 2)
8	2	버퍼 에러/에러 워드 비트 1	0	1	ErrorStatus(비트 1)
8	1	동기 대입 버퍼 에러/에러 워드 비트 0	0	0	ErrorStatus(비트 0)
9	8	PMATCH의 유효인 좌표계 축 정의	1	31	Csolve
9	4	버퍼 Linear-to-PVT 동작	1	30	LinToPvtBuf
9	2	피드 홀드 가속/감속	1	29	FeedHold(비트 1)
9	1	피드 홀드의 타임 베이스 소스	1	28	FeedHold(비트 0)
10	8	블록 활성화	1	27	BlockActive
10	4	연속 모션 요구	1	26	ContMotion
10	2	공구 보정 모드 비트 1	1	25	CCMode(비트 1)
10	1	공구 보정 모드 비트 0	1	24	CCMode(비트 0)
11	8	동작 모드 비트 1(블렌드 및 스플라인인 경우에는 0, Rapid 및 PVT 모드인 경우에는 1)	1	23	MoveMode (비트 1)
11	4	동작 모드 비트 0(블렌드 및 pvt 인 경우에는 0, Rapid 및 스플라인 모드인 경우에는 1)	1	22	MoveMode (비트 0)
11	2	진행 중의 세그먼트화 PVT 모드 동작	1	21	SegMove(비트 1)
11	1	진행 중의 세그먼트화 직선 보간 모드 동작	1	20	SegMove(비트 0)
12	8	최초 세그먼트 동작	1	19	SegMoveAccel
12	4	마지막 세그먼트 동작	1	18	SegMoveDecel
12	2	세그먼트 유효화	1	17	SegEnabled
12	1	세그먼트 정지 요구	1	16	SegStopReq
13	8	룩 어헤드 랩	1	15	LookAheadWrap LHStatus(비트 7)
13	4	룩 어헤드 룩 백	1	14	LookAheadLookBack LHStatus(비트 6)

13	2	룩 어헤드 방향	1	13	LookAheadDir LHStatus(비트 5)
13	1	룩 어헤드 정지	1	12	LookAheadStop LHStatus(비트 4)
14	8	룩 어헤드 변경	1	11	LookAheadChange LHStatus(비트 3)
14	4	룩 어헤드 재계산	1	10	LookAheadReCalc LHStatus(비트 2)
14	2	룩 어헤드 플러시	1	9	LookAheadFlush LHStatus(비트 1)
14	1	룩 어헤드 활성화	1	8	LookAheadActive LHStatus(비트 0)
15	8	공구 보정이 추가된 원호	1	7	CCAddedArc
15	4	공구 보정의 OFF 요구	1	6	CCOffReq
15	2	(예약 완료)	1	5	
15	1	공구 보정 동작 타입 비트 1	1	4	CCMoveType(비트 1)
16	8	공구 보정 동작 타입 비트 0	1	3	CCMoveType(비트 0)
16	4	3D 공구 보정 활성화	1	2	CC3Active
16	2	날카로운 코너에서 블렌드 무효	1	1	SharpCornerStop
16	1	추가 드웰 무효	1	0	AddedDwellDis

첫 문자:

bit 31 트리거 서치 중 동작(TriggerMove): 이 비트는 좌표계의 임의의 모터에 대해 트리거까지의 동작 시작(원점 서치, 트리거까지의 JOG 이송, 트리거까지의 프로그램 Rapid 모드 동작)에 1로 설정됩니다. 사전에 지정된 트리거 상태가 검출되고 트리거 후의 동작이 시작된 경우 또는 트리거가 검출되지 않고 동작이 종료한 경우에는 0으로 설정됩니다.

bit 30 원점 서치 중 동작(HomeInProgress): 이 비트는 좌표계의 임의의 모터에 대해 첫 원점 서치 동작에 1로 설정됩니다. 사전에 지정된 트리거 상태가 검출되고 트리거 후의 동작이 시작된 경우 또는 트리거가 검출되지 않고 동작이 종료한 경우에는 0으로 설정됩니다.

bit 29 하드웨어의 음의 리미트(MinusLimit): 좌표계의 어느 한 모터가 하드웨어의 음의 리미트에 도달한 경우에 이 비트는 1로 설정됩니다. 그 밖에 모터가 이전에 해당 리미트에 도달하더라도 현재는 도달해 있지 않으면 0으로 설정됩니다.

bit 28 하드웨어의 양의 리미트(PlusLimit): 좌표계의 어느 한 모터가 하드웨어의 양의 리미트에 도달한 경우에 이 비트는 1로 설정됩니다. 그 밖에 모터가 이전에 해당 리미트에 도달하더라도 현재는 도달해 있지 않으면 0으로 설정됩니다.

2 번째 문자:

bit 27 경고 위치 편차 (FeWarn): 좌표계의 어느 한 모터의 위치 편차 크기가 [Motor[x].WarnFeLimit]로 설정한 경고 위치 편차 리미트를 초과한 경우에 이 비트는 1이 됩니다. 위치 편차의 크기가 이 제한 미만인 경우 또는 편차 리미트를 초과했기 때문에 모터가 무효로 설정된 경우에 이 비트는 0이 됩니다.

bit 26 편차 이상(FeFatal): 좌표계의 어느 한 모터의 위치 편차 크기가 [Motor[x].FatalFeLimit]로 설정한 편차 리미트를 초과했기 때문에 모터가 무효가 된 경우에 이 비트는 1이 됩니다. 그 밖에는 0이며 모터가 다시 Enable 된 경우에도 0이 됩니다.

bit 25 하드웨어의 위치 리미트에서 정지(LimitStop): 양 또는 음의 하드웨어 오버트래블 리미트에 도달했기 때문에 좌표계의 어느 한 모터가 정지한 경우에 현재는 리미트에 도달하지 않았더라도 이 비트는 1 이 됩니다. 그 밖에 모두 리미트에 도달했다가 벗어난 경우도 포함하여 이 비트는 0 이 됩니다.

bit 24 앰프 폴트 에러(AmpFault): (앰프 폴트의 시그널 상태가 존재하지 않더라도) 앰프 폴트 에러 때문에 좌표계의 어느 한 모터가 무효가 된 경우 또는 산출된 I^2T 의 적분 전류 폴트때문에(이 경우에는 비트 21 도 설정됨) 모터가 무효가 된 경우에 이 비트는 1 이 됩니다. 그 밖에는 0 이며 모터가 다시 Enable 된 경우에도 0 이 됩니다.

3 번째 문자:

bit 23 소프트웨어의 음의 리미트(SoftMinusLimit): [Motor[x].MinPos]로 설정한 소프트웨어의 음의 리미트에 좌표계의 어느 한 모터가 도달했거나 초과하면 이 비트는 1 로 설정됩니다(활성화하려면 [Motor[x].MaxPos]보다 작은 값으로 설정해야 합니다). 그 밖에 모터가 이전에 해당 리미트에 도달하더라도 현재는 도달해 있지 않으면 0 으로 설정됩니다.

bit 22 소프트웨어의 양의 리미트(SoftPlusLimit): [Motor[x].MaxPos]로 설정한 소프트웨어의 양의 리미트에 좌표계의 어느 한 모터가 도달했거나 초과하면 이 비트는 1 로 설정됩니다(활성화하려면 [Motor[x].MinPos]보다 큰 값으로 설정해야 합니다). 그 밖에 모터가 이전에 해당 리미트에 도달하더라도 현재는 도달해 있지 않으면 0 으로 설정됩니다.

bit 21 적분 전류(I^2T) 폴트(I2tFault): [Motor[x].I2tTrip]로 설정한 적분 전류 리미트를 초과하여 좌표계의 어느 한 모터가 무효가 된 경우에 이 비트는 1 로 설정됩니다. 이 경우에 앰프 폴트의 비트(비트 24)도 설정됩니다. 그 밖에는 모두 0 이며 모터가 다시 Enable 된 경우에도 0 이 됩니다(적분 전류 폴트 계산을 통해 앰프 폴트가 발생한 경우에 이 비트는 설정되지 않는다는 점에 주의하십시오).

bit 20 트리거가 발견되지 않음(TriggerNotFound): 좌표계의 어느 한 모터에서 트리거까지의 JOG 이송 또는 트리거까지의 프로그램 Rapid 모드 동작이 사전에 지정된 트리거 상태를 검출하지 않고 종료한 경우에 이 비트는 1 로 설정됩니다. 이는 에러 상태는 아니지만 후속 처리가 많은 경우에 해당 비트의 설정 여부에 따라 다릅니다. 그 밖에는 모두 이 비트는 0 이며 다음 동작이 시작될 때 0 으로 복귀합니다.

4 번째 문자:

bit 19 앰프 경고(AmpWarn): 좌표계의 어느 한 모터에 대해 [Motor[x].AmpFaultLevel]의 비트 1(값 2)이 1 로 설정된 경우에 이 비트는 1 로 설정됩니다. 에러를 트리거하기 위해 지정된 폴트 상태에서 앰프 폴트 비트를 연속으로 2 회 읽어야 하지만 앰프 폴트 비트의 하나를 읽었을 때 폴트 상태였습니다.

bit 18 엔코더 소실 에러(EncLoss): 엔코더 소실 에러로 좌표계의 어느 한 모터가 무효가 된 경우에 이 비트는 1 이 됩니다. 그 밖에는 0 이며 모터가 다시 Enable 된 경우에도 0 이 됩니다.

bit 17 {추후 사용하기 위해 예약}

bit 16 이동 타이머 유효화(TimerEnabled): 좌표계의 어느 한 모터가 정한시의 동작 또는 동작 섹션을 실행 중인 경우에 이 비트는 1 로 설정됩니다. 그 밖의 경우에는 0 으로 설정됩니다.

5 번째 문자:

bit 15 위치 참조 확립(HomeComplete): 원점 서치 동작 또는 절대 위치 읽기 중 어느 하나에서 좌표계의 모든 모터에 대해 위치 참조가 올바르게 확립된 경우에 이 비트는 1로 설정됩니다. 전원 셋업/리셋 시 및 모터의 첫 원점 서치 동작에는 자동으로 0으로 설정됩니다.

bit 14 명령 속도 제로(DesVelZero): 좌표계의 모든 모터가 클로즈드 루프 제어로(위치 유지 시행 중 등) 명령 속도가 제로가 된 경우 또는 모터가 오픈 루프 모드(유효 또는 무효)로 현재 속도가 제로와 같은 경우에 이 비트는 1로 설정됩니다. 좌표계의 어느 한 모터가 클로즈드 루프 모드이고 명령 속도가 제로 이외인 경우 또는 모터가 오픈 루프 모드이고 현재 속도가 제로 이외인 경우에 이 비트는 0이 됩니다.

bit 13 클로즈드 루프 모드(ClosedLoop): 좌표계의 모든 모터가 클로즈드 루프 제어인 경우에 이 비트는 1로 설정됩니다. 좌표계의 어느 한 모터가 오픈 루프 모드(유효 또는 무효)인 경우에는 0이 됩니다.

bit 12 앰프 Enable(AmpEna): 좌표계의 모든 모터가 유효인(클로즈드 루프 제어 또는 오픈 루프 제어) 경우에 이 비트는 1로 설정됩니다. 이 경우에는 활성 앰프의 Enable 출력 신호가 아니어도 된다는 점에 주의하십시오. 좌표계의 어느 한 모터가 무효로 설정된 경우에 이 비트는 0이 됩니다.

6 번째 문자:

bit 11 인포지션(InPos): 좌표계의 모든 모터가 모든 인포지션의 조건을 만족한 경우에 이 비트는 1로 설정됩니다. 모든 조건이란, 모터가 클로즈드 루프이고 명령 속도가 제로이며 이동 타이머가 활성이 아니고(동작 없음, 드웰 또는 지연 중), 위치 편차의 크기가 [Motor[x].InPosBand] 이하이며 이러한 모든 조건이([Motor[x].InPosTime]- 1)인 연속 서보 사이클에 대해 True임을 말합니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 10 {추후 사용하기 위해 예약}

bit 9 블록 요구 플래그(BlockRequest): 좌표계의 어느 한 모터가 새로운 동작 섹션에 들어간 시점에서 모션 신호 다음의 동작 섹션 식을 계산하라는 요청이 있는 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다. 이는 주로 내부에서 사용됩니다.

bit 8 동작 타이머 유효화(TimersEnabled): 좌표계의 모든 모터가 정한시의 동작 또는 동작 섹션을 실행 중인 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

7 번째 문자:

bit 7 XX/YY/ZZ 축 Circle 반경 에러/에러 워드 비트 7(RadiusError 비트 1/ErrorStatus 비트 7): XX/YY/ZZ 축 공간의 원호 보간 동작이 [Coord[x].RadiusErrorLimit]로 설정한 값을 초과하는 반경 에러가 된 경우에 이 비트는 1로 설정됩니다(이 경우에 [ErrorStatus]의 다른 7개 비트는 설정되지 않습니다). 이 중에 다른 비트도 설정된 경우에는 단순히 에러 코드의 비트 7(값 128)이 1임을 나타냅니다. 다음의 에러 목록을 참조하십시오. 비트 7에 에러 코드가 설정되지 않은 경우에는 비트는 0이 됩니다.

bit 6 X/Y/Z 축 Circle 반경 에러/에러 워드 비트 6(RadiusError 비트 0/ErrorStatus 비트 6): X/Y/Z 축 공간의 원호 보간 동작이 [Coord[x].RadiusErrorLimit]로 설정한 값을 초과하는 반경 에러가 된 경우에 이 비트는 1로 설정됩니다(이 경우에 [ErrorStatus]의 다른 7개 비트는 설정되지 않습니다). 이 중에 다른 비트도 설정된 경우에는 단순히 에러 코드의 비트 6(값 64)이 1임을 나타냅니다. 다음의 에러 목록을 참조하십시오. 비트 6에 에러 코드가 설정되지 않은 경우에는 비트는 0이 됩니다.

bit 5 소프트웨어의 위치 리미트에서 정지/에러 워드 비트 5(SoftLimit/ErrorStatus 비트 5): 좌표계의 모터가 양 또는 음 중 하나의 소프트웨어 오버트래블 리미트에 도달했기 때문에 좌표계의 모션 프로그램이 정지한 경우에 (현재 그 리미트에 도달하지 않은 경우에도) 이 비트는 1이 됩니다(현재 리미트에 도달하지 않은 경우에는 [ErrorStatus]의 다른 7개 비트는 설정되지 않습니다). 이 중에 다른 비트도 설정된 경우에는 단순히 에러 코드의 비트 5(값 32)가 1임을 나타냅니다. 다음의 에러 목록을 참조하십시오. 그 밖에 모두 리미트에 도달했다가 벗어난 경우도 포함하여 이 비트는 0이 됩니다.

bit 4 런타임 에러/에러 워드 비트 4(RunTimeError/ErrorStatus 비트 4): 계산 시간이 부족하기 때문에 좌표계의 모션 프로그램이 이상 종료한 경우에는 이 비트는 1이 됩니다(이 경우에는 [ErrorStatus]의 다른 7개 비트는 설정되지 않습니다). 이 중에 다른 비트도 설정된 경우에는 단순히 에러 코드의 비트 4(값 16)가 1임을 나타냅니다. 다음의 에러 목록을 참조하십시오. 그 밖의 경우에 이 비트는 0이 됩니다.

8 번째 문자:

bit 3 PVT 모드 에러/에러 워드 비트 3(PvtError/ErrorStatus 비트 3): PVT 모드의 동작 실행 중에 에러가 발생하여 좌표계의 모션 프로그램이 이상 종료한 경우에는 이 비트는 1이 됩니다(이 경우에는 [ErrorStatus]의 다른 7개 비트는 설정되지 않습니다). 이러한 가운데 다른 비트도 설정된 경우에는 단순히 에러 코드의 비트 3(값 8)이 1임을 나타냅니다. 다음의 에러 목록을 참조하십시오. 그 밖의 경우에 이 비트는 0이 됩니다.

bit 2 Linear-to-PVT 모드 에러/에러 워드 비트 2(LinToPvtError/ErrorStatus 비트 2): linear-to-PVT 모드의 동작 실행 중에 에러가 발생하여 좌표계의 모션 프로그램이 이상 종료한 경우에 이 비트는 1이 됩니다(이 경우에는 [ErrorStatus]의 다른 7개 비트는 설정되지 않습니다). 그중에서 다른 비트도 설정된 경우에는 단순히 에러 코드의 비트 2(값 4)가 1임을 나타냅니다. 다음의 에러 목록을 참조하십시오. 그 밖의 경우에 이 비트는 0이 됩니다.

bit 1 버퍼 에러/에러 워드 비트 1(ErrorStatus 비트 1): 프로그램 버퍼에서 에러 때문에 좌표계의 모션 프로그램이 이상 종료한 경우에 이 비트는 1이 됩니다(이 경우에는 [ErrorStatus]의 다른 7개 비트는 설정되지 않습니다). 이 중에 다른 비트도 설정된 경우에는 단순히 에러 코드의 비트 1(값 2)이 1임을 나타냅니다. 다음의 에러 목록을 참조하십시오. 그 밖의 경우에 이 비트는 0이 됩니다.

bit 0 동기 대입 버퍼 에러/에러 워드 비트 0(ErrorStatus 비트 0): 동기 대입 버퍼에서 에러 때문에 좌표계의 모션 프로그램이 이상 종료한 경우에 이 비트는 1이 됩니다(이 경우에는 [ErrorStatus]의 다른 7개 비트는 설정되지 않습니다). 이 중에 다른 비트도 설정된 경우에는 단순히 에러 코드의 비트 1(값 2)이 1임을 나타냅니다. 다음의 에러 목록을 참조하십시오. 그 밖의 경우에 이 비트는 0이 됩니다.

9 번째 문자:

bit 31 PMATCH 계산의 유효 측 정의(Csolve): 이 좌표계의 측 정의가 좌표계 PMATCH(위치 일치) 함수이고 (모션 프로그램 시작 시, pmatch 명령 실행 시 측 위치/속도/위치 편차 쿼리 시) 측 위치를 계산하는 데 충분한 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 30 버퍼 Linear-to-PVT 동작(LinToPvtBuf): ([Coord[x].SegLinToPvt] > 0 이기 때문에)PVT 동작으로 변환된 직선 보간 모드 동작이 현재 버퍼되었고 실행 대기인 경우에 이 비트는 1 이 됩니다. 그 밖의 경우에는 0 으로 설정됩니다. 이는 주로 내부에서 사용됩니다.

bit 29 피드 홀드 가속감속(FeedHold 비트 1): 룩 어헤드 외부에서의 h, hold(피드 홀드) 명령이나 lh(급정지) 명령을 통해 좌표계가 감속하거나 이 상태 이외에서 재가속한 경우에 이 비트는 1 이 됩니다. 그 밖의 경우에는 0 으로 설정됩니다.

bit 28 피드 홀드의 타임 베이스(FeedHold 비트 0): 좌표계가 보통 지정된 타임 베이스가 아니라 피드 홀드의 타임 베이스를 사용한 경우에 이 비트는 1 이 됩니다. 이는 룩 어헤드 외부에서의 h, hold(피드 홀드) 명령이나 lh(급정지) 명령을 통해 감속 및 완전 정지 중인 경우입니다. 그 이외의 경우에는 (피드 홀드 상태에서 재감속 중인 경우도 포함하여)0 이 됩니다.

10 번째 문자:

bit 27 블록 활성(BlockActive): 좌표계가 bstart(블록 스타트) 명령과 bstop(블록 스톱) 명령 사이에서 계산을 실행 중인 경우에 이 비트는 1 로 설정됩니다. 그 밖의 경우에는 0 으로 설정됩니다. 이는 주로 내부에서 사용됩니다.

bit 26 연속 모션 요구(ContMotion): 좌표계가 일련의 프로그램 동작을 실행 중이며 그러한 프로그램의 사이는 정지하지 않고 블렌드된 경우에 이 비트는 1 로 설정됩니다. 이유에 관계없이 이와 같은 일련의 프로그램을 실행하지 않은 경우에 비트는 0 이 됩니다.

bit 25 공구 보정 모드 비트 1(CCMODE 비트 1): (ccmode2 또는 ccmode3 을 통해)공구 보정 모드가 2 또는 3 으로 설정된 경우에 이 비트는 1 로 설정됩니다. 그 밖의 경우에는 0 으로 설정됩니다.

bit 24 공구 보정 모드 비트 0(CCMODE 비트 0): (ccmode1 또는 ccmode3 을 통해)공구 보정 모드가 1 또는 3 으로 설정된 경우에 이 비트는 1 로 설정됩니다. 그 밖의 경우에는 0 으로 설정됩니다.

11 번째 문자:

bit 23 동작 모드 비트 1(MoveMode 비트 1): 좌표계가 PVT 또는 Rapid 동작을 실행 중인 경우(PVT 인 경우에는 MoveMode = 2, Rapid 인 경우에는 = 3) 또는 이러한 모드 중 하나를 마지막으로 실행 중인 경우에 이 비트는 1 로 설정됩니다. 좌표계가 블렌드(직선 보간 또는 원호 보간) 또는 스플라인 동작을 실행한(블렌드인 경우에는 MoveMode = 0, 스플라인인 경우에는 = 1) 경우 또는 이러한 모드 중 하나를 마지막으로 실행 중인 경우에는 이 비트는 0 으로 설정됩니다.

bit 22 동작 모드 비트 0(MoveMode 비트 0): 좌표계가 스플라인 또는 Rapid 동작을 실행 중인 경우(스플라인인 경우에는 MoveMode = 1, Rapid 인 경우에는 = 3) 또는 이러한 모드 중 하나를 마지막으로 실행 중인 경우에는 이 비트는 1 로 설정됩니다. 좌표계가 블렌드(직선 보간 또는 원호 보간) 또는 PVT 동작을 실행한(블렌드인 경우에 MoveMode = 0, 스플라인인 경우에는 = 2) 경우 또는 이러한 모드 중 하나를 마지막으로 실행 중인 경우에 이 비트는 0 으로 설정됩니다.

bit 21 진행 중인 세그먼트화 PVT 모드 동작(SegMove 비트 1): 좌표계가 세그먼트 모드로 PVT 동작을 실행 중인 경우에 이 비트는 1 로 설정됩니다. 그 밖의 경우에는 0 으로 설정됩니다. 이는 주로 내부에서 사용됩니다.

bit 20 진행 중인 세그먼트화 블렌드 모드 동작(SegMove 비트 0): 좌표계가 세그먼트 모드로 직선 보간 모드 동작 또는 원호 보간 모드 동작을 실행 중인 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다. 이 비트는 주로 내부에서 사용됩니다.

12 번째 문자:

bit 19 최초의 동작 세그먼트 실행(SegMoveAccel): 좌표계가 세그먼트화된 동작 또는 동작 시퀀스이고 정지 상태에서 최초의(추가) 세그먼트를 실행 중인 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다. 이 비트는 주로 내부에서 사용됩니다.

bit 18 마지막 동작 세그먼트 실행(SegMoveDecel): 좌표계가 세그먼트화된 동작 또는 동작 시퀀스이고 정지 상태를 목표로 마지막 (추가) 세그먼트를 실행 중인 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다. 이 비트는 주로 내부에서 사용됩니다.

bit 17 세그먼트 동작 실행(SegEnabled): 좌표계가 세그먼트 동작(Coord[x].SegMoveTime > 0에서 직선 보간, 원호 보간 또는 PVT 동작)을 실행한(또는 조금 전에 실행한) 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 16 세그먼트 동작의 정지 요구(SegStopReq): 좌표계에서 세그먼트 동작 시퀀스 사이에서 정지하라는 요청이 있는 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다. 이 비트는 주로 내부에서 사용됩니다.

13 번째 문자:

bit 15 룩 어헤드 버퍼 랩 어라운드 룩 어헤드 Status 비트 7(LookAheadWrap/LHStatus 비트 7): 룩 어헤드 버퍼가 많고 현재의 연속 동작 시퀀스에서 랩 어라운드한 경우에는 이 비트는 1로 설정됩니다. 이는 시퀀스 전체는 역행할 수 없음을 의미합니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 14 룩 어헤드 룩 백 룩 어헤드 Status 비트 6(LookAheadLookBack/LHStatus 비트 6): 마지막에 추가된 동작의 마지막에서 제어 정지가 가능하다는 것을 확인하기 위해 룩 어헤드 알고리즘이 버퍼 내의 백 워드를 계산한 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 13 룩 어헤드 방향 룩 어헤드 Status 비트 5(LookAheadDir/LHStatus 비트 5): 역방향으로 룩 어헤드 버퍼를 실행 중인 경우에는 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 12 룩 어헤드 정지/룩 어헤드 Status 비트 4(LookAheadStop/LHStatus 비트 4): \, lh\ (급정지) 명령을 통해 룩 어헤드 버퍼의 실행이 정지했거나 정지를 위해 감속 중인 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

14 번째 문자:

bit 11 룩 어헤드 실행 변경 룩 어헤드 Status 비트 3(LookAheadChange/LHStatus 비트 3): 룩 어헤드 버퍼의 실행모드가 순방향과 역방향 2개의 모드 간에 전환된 경우 및 룩 어헤드 버퍼의 실행이 정지한 경우에는 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 10 룩 어헤드 버퍼의 재계산 룩 어헤드 Status 비트 2(LookAheadReCalc/LHStatus 비트 2): 이미 존재하는 버퍼 내에서 버퍼에 새로운 세그먼트의 추가가 아니라 룩 어헤드 계산이 역행 또는 역행에서 리커버리하여 실행 중인 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 9 룩 어헤드 버퍼 플러시/룩 어헤드 Status 비트 1(LookAheadFlush/LHStatus 비트 1): 룩 어헤드 버퍼가 시퀀스의 마지막에 도달하고 버퍼에 새로운 세그먼트 추가가 없는(단 기존의 세그먼트는 아직 실행 중) 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 8 룩 어헤드 활성/룩 어헤드 Status 비트 0(LookAheadActive/LHStatus 비트 0): 좌표계 룩 어헤드 버퍼 내에서 실행 중인 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

15 번째 문자:

bit 7 공구 보정에서 추가된 원호 동작(CCArcAdded): 계산된 동작이 2D 공구 직경 보정의 외측 코너 주위에서 자동으로 추가되는 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 6 공구 보정의 OFF 요구 (CCOffReq): CC0 명령을 통해 좌표계가 공구 보정을 OFF로 설정하도록 통지된 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 5 {추후 사용하기 위해 예약}

bit 4 공구 보정 동작 타입 비트 1(CCMoveType 비트 1): 좌표계가 공구 보정이 활성화이고 직선 보간 또는 원호 보간 동작을 실행 중인(직선 보간인 경우에는 **CCMoveType** = 2, circle 인 경우에는 3) 경우에 이 비트는 1로 설정됩니다. 공구 보정이 활성화로 좌표계가 드웰 또는 제로 거리/평면 외 동작을 실행 중(드웰인 경우에는 **CCMoveType** = 0, 제로 거리/평면 외인 경우에는 1)인 경우 또는 공구 보정이 활성화가 아닌 경우에는 이 비트는 0이 됩니다. 이는 주로 내부에서 사용됩니다.

16 번째 문자:

bit 3 공구 보정 동작 타입 비트 0(CCMoveType 비트 0): 공구 보정이 활성화이고 좌표계가 제로 거리/평면 외 동작(제로 거리/평면 외인 경우에는 **CCMoveType** = 1, circle 인 경우에는 3) 또는 원호 보간 동작을 실행 중인 경우에 이 비트는 1로 설정됩니다. 공구 보정이 활성화이고 좌표계가 드웰 또는 직선 보간 동작을 실행 중(드웰인 경우에는 **CCMoveType** = 0, 직선 보간인 경우에는 2)인 경우에 이 비트는 0이 됩니다. 이는 주로 내부에서 사용됩니다.

bit 2 3D 공구 보정 활성(CC3Active): 3D 공구 보정이 활성화인 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 1 날카로운 코너에서 블렌드 무효(SharpCornerStop): [Coord[x].CornerBlendBp]로 지정된 것보다 각도가 날카롭기 때문에 블렌드가 코너에서 무효가 된 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 0 추가 드웰 무효(AddedDwellDis): [Coord[x].CornerBlendBp]가 지정한 각도보다 날카롭고 [Coord[x].CornerDwellBp]가 지정한 각도보다 둔한 코너에서는 블렌드가 무효로 설정되지만 코너에 추가 드웰이 없는 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

Coord[x].ErrorStatus 코드의 값:

값	에러명	설명
0	NoError	정상 실행
1	SyncBufferError	동기 변수 할당 버퍼에서 에러
2	BufferError	모션 프로그램 버퍼에서 에러
3	CCMoveTypeError	공구 보정이 활성화일 때의 올바르지 않은 동작 모드 또는 명령(rapid, pvt, spline, lin-to-pvt, new normal, pmatch, pclr, pset, pload)
4	LinToPvtError	자동적인 linear-to-PVT 모드 변환에서 에러
5	CCLeadOutMoveError	올바르지 않은 공구 보정 취소 동작(원호 보간 모드 또는 길이가 공구 직경 미만)
6	CCLeadInMoveError	올바르지 않은 공구 보정 도입 동작(원호 보간 모드 또는 길이가 공구 직경 미만)
7	CCBufSizeError	공구 보정 동작 버퍼의 크기 부족(다음의 평면 내 동작 검출에 불충분)
8	PvtError	올바르지 않게 지정된 PVT 모드 동작
9	CCFeedRateError	공구 보정용 피드 레이트가 지정하지 않은 동작
10	CCDirChangeError	공구 보정 중의 완전 역행
11	CCNoSolutionError	보정 완료 동작에 해결 방법을 찾지 못함
12	CC3NdotTError	3D 공구 보정 벡터 계산 에러(공구 선단 테이블의 최초 항목보다 'NdotT'의 값이 작음)
13	CCDistanceError	취소 동작에 따른 오버컷 해소 불능
14	CCNoIntersectError	보정된 경로의 교점 검출 불능
15	CCNoMovesError	도입 동작과 취소 동작 간에 보정된 동작 없음
16	RunTimeError	불충분한 동작 계산 시
17	InPosTimeOutError	인포지션으로 지정된 타임 리미트를 초과했음
18	LHNumMotorsError	사용한 모터 수와 룩 어헤드 버퍼 내의 모터 수가 일치하지 않음
19-31	(추후 사용하기 위해 예약)	
32	SoftLimitError	소프트웨어의 위치 리미트를 초과하지 않았음
33-63	(추후 사용하기 위해 예약)	
64	XYZRadiusError	X/Y/Z 평면 원호 보간 동작에서 반경 에러
65-127	(추후 사용하기 위해 예약)	
128	XXYYZZRadiusError	XX/YY/ZZ 평면 원호 보간 동작에서 반경 에러
129-255	(추후 사용하기 위해 예약)	

글로벌 Status

글로벌 Status 에 대해 통지되는 Status 값이 각 Status 비트를 어떻게 나타내는지는 다음 표와 같습니다. 좌측의 2 열은 이 명령에 대한 16 진법의 ASCII 응답에서 Status 비트를 어떻게 해석할 수 있는지를 나타냅니다. 다음의 열은 Status 비트의 정보에 관한 기본적인 설명입니다. 다음의 2 열은 데이터 구조체 요소 [Sys.Status]의 수치에서 Status 비트를 어떻게 해석할 수 있는지를 나타내고 마지막 열은 비트의 데이터 구조체 요소 이름을 나타냅니다.

명령의 요소명 [backup Sys.Status]를 사용하여 같은 정보를 텍스트로 취득할 수 있습니다. 각 요소명은 현재의 값으로 복귀하여 통지됩니다.

Char #	16 진수값	설명	Word #	Bit #	요소명: Sys.
1	8	(예약 완료)	0	31	
1	4	(예약 완료)	0	30	
1	2	(예약 완료)	0	29	
1	1	(예약 완료)	0	28	
2	8	(예약 완료)	0	27	
2	4	(예약 완료)	0	26	
2	2	(예약 완료)	0	25	
2	1	(예약 완료)	0	24	
3	8	(예약 완료)	0	23	
3	4	(예약 완료)	0	22	
3	2	(예약 완료)	0	21	
3	1	(예약 완료)	0	20	
4	8	(예약 완료)	0	19	
4	4	(예약 완료)	0	18	
4	2	(예약 완료)	0	17	
4	1	(예약 완료)	0	16	
5	8	(예약 완료)	0	15	
5	4	(예약 완료)	0	14	
5	2	(예약 완료)	0	13	
5	1	(예약 완료)	0	12	
6	8	(예약 완료)	0	11	
6	4	(예약 완료)	0	10	
6	2	(예약 완료)	0	9	
6	1	시스템 클럭이 발견되지 않음	0	8	NoClocks
7	8	공장 출하 시의 설정(cmd 또는 에러에 따름)	0	7	Default
7	4	시스템 파일의 설정 에러	0	6	FileConfigErr
7	2	저장한 후에 하드웨어 변경이 검출되었음	0	5	HWChangeErr
7	1	저장된 설정의 로드 에러	0	4	ConfigLoadErr
8	8	프로젝트 로드 에러	0	3	ProjectLoadErr
8	4	전원 셋업/리셋의 로드 폴트(비트 3-6의 OR)	0	2	PwrOnFault
8	2	소프트웨어 워치독 폴트 비트 1	0	1	WDTFault (bit 1)
8	1	소프트웨어 워치독 폴트 비트 0	0	0	WDTFault (bit 0)

첫 문자:

bit 31 .. 28 {추후 사용하기 위해 예약}

2 번째 문자:

bit 27 .. 24 {추후 사용하기 위해 예약}

3 번째 문자:

bit 23 .. 20 {추후 사용하기 위해 예약}

4 번째 문자:

bit 19 .. 16 {추후 사용하기 위해 예약}

5 번째 문자:**bit 15 .. 12** {추후 사용하기 위해 예약}**6 번째 문자:****bit 11 .. 9** {추후 사용하기 위해 예약}

bit 8 시스템 클럭이 발견되지 않음(NoClocks): 전원 셋업/리셋 프로세스의 마지막에 프로세서가 페이즈 또는 서보 클럭을 검출하지 않는 경우에 이 비트는 1로 설정됩니다. 이때 이러한 클럭 시그널을 검출한 경우에는 이 비트는 0으로 설정됩니다.

7 번째 문자:

bit 7 공장 출하시의 설정(Default): 재초기화 명령 또는 프로세스 중의 에러에 따라 전원 셋업/리셋 프로세스 중에 프로세서가 시스템을 전원 출하시의 설정으로 한 경우에 이 비트는 1로 설정됩니다. 그 밖의 경우에는 0으로 설정됩니다.

bit 6 시스템 파일의 설정 에러(FileConfigErr): 전원 셋업/리셋 프로세스 중에 프로세서가 시스템 파일 설정에서 에러를 검출한 경우에 이 비트는 1로 설정됩니다. 이 경우에 시스템은 공장 출하 시의 설정이 됩니다. 이때에 이와 같은 에러가 검출되지 않는 경우에 이 비트는 0이 됩니다.

bit 5 하드웨어 변경이 검출되었음(HWChangeErr): 전원 셋업/리셋 프로세스 중에 마지막 저장 이후에 시스템 하드웨어 설정이 변경되었음을 프로세서가 검출한 경우에 이 비트는 1로 설정됩니다. 이 경우에 시스템은 공장 출하 시의 설정이 됩니다. 이때에 이와 같은 에러가 검출되지 않는 경우에 이 비트는 0이 됩니다.

bit 4 저장된 설정의 로드 에러(ConfigLoadErr): 전원 셋업/리셋 프로세스 중에 프로세서가 사용자 저장 설정에서 에러를 검출한 경우에 이 비트는 1로 설정됩니다. 이 경우에 시스템은 공장 출하 시의 설정이 됩니다. 이때에 이와 같은 에러가 검출되지 않는 경우에 이 비트는 0이 됩니다.

8 번째 문자:

bit 3 프로젝트 로드 에러(ProjectLoadErr): 전원 셋업/리셋 프로세스 중에 프로세서가 사용자 프로젝트 파일을 활성 메모리에 로드할 때에 에러를 검출한 경우에 이 비트는 1로 설정됩니다. 이 경우에 시스템은 공장 출하 시의 설정이 됩니다. 이때에 이와 같은 에러가 검출되지 않는 경우에 이 비트는 0이 됩니다.

bit 2 전원 셋업/리셋의 로드 폴트(PwrOnFault): 전원 셋업/리셋 프로세스 중에 프로세서가 에러를 검출한 경우에 이 비트는 1로 설정됩니다. 이 비트는 비트 3-6의 논리 OR이기 때문에 이 중 하나의 에러가 검출된 경우에 설정됩니다. 이 경우에 시스템은 공장 출하 시의 설정이 됩니다. 이때에 이와 같은 에러가 검출되지 않는 경우에 이 비트는 0이 됩니다.

bit 1 백그라운드 소프트웨어 워치독 타이머 폴트(WDTFault 비트 1): 마지막 실시간 인터럽트 후 경과한 백그라운드 사이클이 너무 많으므로 프로세서가 소프트웨어 워치독 폴트를 검출한 경우에 이 비트는 1로 설정됩니다. 이 경우에 모든 사용자 프로그램은 무효가 되고 모든 하드웨어 인터페이스가 리셋 상태가 됩니다. 이와 같은 에러가 발견되지 않는 경우에는 이 비트는 0이 됩니다.

bit 0 포어그라운드 소프트웨어 워치독 타이머 폴트(WDTFault 비트 0): 백그라운드 사이클 후 경과한 실시간 인터럽트 사이클이 너무 많으므로 프로세서가 소프트웨어 워치독 폴트를 검출한 경우에 이 비트는 1로 설정됩니다. 이 경우에 모든 사용자 프로그램은 무효가 되고 모든 하드웨어 인터페이스가 리셋 상태가 됩니다. 이와 같은 에러가 발견되지 않는 경우에는 이 비트는 0이 됩니다.

예

?	// Request global status word
000000A4	// Factory default configuration due to hardware change
#1?	// Request Motor 1 status words
0000380000000000	// "Closed loop", "amp enabled", "in-position" bits set
&1?	// Request C.S. 1 status words
0000300000000000	// "Closed loop", "amp enabled" bits set

a

기능: 모션 프로그램과 동작을 Abort 함

범위: 좌표계 고유

구문: **a, abort**

a 명령을 통해 Power PMAC은 지정한 좌표계의 모션 프로그램과 동작을 Abort시킵니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계에 대해 모션 프로그램과 동작을 Abort합니다. 좌표계 목록이 있는 경우에 목록 내의 좌표계 모두에 대해 모션 프로그램의 실행과 동작을 Abort합니다. 좌표계가 모션 프로그램 또는 그 서브 프로그램 1개를 실행 중인 경우에 모션 프로그램의 프로그램 카운터는 최초의 (최상위의) 모션 프로그램으로 설정됩니다.

여러 좌표계 목록을 지정하더라도 후속 좌표계 고유의 명령에 대해 모달에 주소 지정된 좌표계는 변경되지 않습니다.

Abort한 모터는 제어(클로즈드 루프) 정지를 위해 데이터 구조체 요소 [Motor[x].AbortTa] 및 [Motor[x].AbortTs]로 설정한 속도 또는 시간으로 감속합니다. Power PMAC은 이 명령을 받는 즉시 지정된 좌표계 내의 모든 모터의 명령 동작 프로파일에 인터럽트, 각 모터의 개별 감속 프로파일을 계산합니다. 이러한 감속 프로파일은 독립되어 있으므로 정지 시간 또는 경로에 대해 반드시 어떤 조건이 존재하는 것은 아닙니다.

V1.6보다 이전 버전의 펌웨어에서 타임 베이스가 100%가 아닌 경우에는 모터의 감속 프로파일은 그 시점의 타임 베이스 비율로 실행됩니다. 2014년 1/4 분기에 출시된 V1.6 펌웨어부터 [Coord[x].AbortTimeBase]에 따라 사용자가 최소 타임 베이스의 값을 지정할 수 있으며 모터의 감속 프로파일은 이 값으로 실행됩니다. 명령이 전송됐을 때 타임 베이스가 이 값보다도 작은 경우에 프로파일은 100% 타임 베이스로 실행됩니다.

abort 명령은 제어 정지가 가능한 경우 긴급 정지에 유용합니다(어플리케이션에 따라서는 최초의 정지에서 **Abort** 하고 보통의 요건을 충족하기 위해 지연 릴레이를 사용하여 드라이브 전원을 OFF 할 수 있습니다).

모션 프로그램 조작이 (quit 나 hold 명령 등의) 다른 방법으로 이미 보류된 경우에 abort 명령으로 프로그램 실행을 완전히 정지시키면 다른 프로그램에서 시작이나 Power PMAC에 새 프로그램 다운로드 등을 할 수 있습니다.

정지한 포인트에서 모션 프로그램 실행을 재개하는 경우에 프로그램을 정지하기 위한 **abort** 명령을 지정하지 마십시오. 보통 **abort** 명령은 프로그램한 포인트에서 모션을 정지시키거나 프로그램한 경로로 모션을 정지시킬 수 없습니다.

a 명령은 정지한 모터를 **Enable** 로 하지 않습니다. PMAC 및 Turbo PMAC에서는 **Enable** 로 했지만 대신에 좌표계고유의 **enable** 명령 또는 모터 고유의 **j/** 명령을 사용합니다. 단, **a** 명령은 오픈 루프 대응의 모터이고 루프를 클로즈합니다. 명령 전송 시에 모터의 속도가 제로가 아닌 경우에 모터는 데이터 구조체 요소 **[Motor[x].AbortTa]** 및 **[Motor[x].AbortTs]**로 설정한 속도나 시간으로 정지를 위해 감속합니다.

이 명령의 생략형(**a**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**abort**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 **[Sys.NoShortCmds]**(2012년 3/4 분기에 출시된 V1.5의 새 기능)가 1로 설정된 경우에는 이 명령의 비생략형(**abort**)을 사용해야 합니다. 생략형(**a**)은 올바르지 않은 명령이며 에러 20으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

동등한 버퍼 프로그램의 다이렉트 명령은 **abort**입니다.

예

a	// Abort motors and programs in addressed C.S.
&2a	// Address C.S.2 and abort motors and programs there
&1, 3, 5a	// Abort motors and programs in C.S. 1, 3, and 5
&*a	// Abort motors and programs in all C.S.

abort

기능: 모션 프로그램과 동작을 Abort 함

범위: 좌표계 고유

구문: **a, abort**

abort 명령은 **a** 명령의 비생략형입니다. Power PMAC은 이 명령을 사용하여 지정된 좌표계의 모션 프로그램과 동작을 Abort 하도록 할 수 있습니다. **[Sys.NoShortCmds]**가 기본값인 0으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. **[Sys.NoShortCmds]**가 1로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

adisable

기능: 지연 Disable 다음에 Abort 함

범위: 좌표계 고유

구문: **adisable**

Power PMAC은 **adisable** 명령으로 좌표계의 모든 모터를 지연 **Disable**로 설정한 후에 지정된 좌표계의 모션 프로그램과 동작을 자동으로 Abort 시킵니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 좌표계가 모션 프로그램 또는 그 서브 프로그램 1개를 실행 중인 경우에 모션 프로그램의 프로그램 카운터는 최초의 (최상위의)모션 프로그램으로 설정됩니다.

여러 좌표계 목록을 지정하더라도 후속 좌표계 고유의 명령에 대해 모달에 주소 지정된 좌표계는 변경되지 않습니다.

adisable 명령에서 최초의 처리는 **abort** 명령 처리와 같으며 좌표계의 모든 모터는 제어(클로즈드 루프) 정지를 위해 [Motor[x].AbortTa] 및 [Motor[x].AbortTs]로 설정된 속도 또는 시간으로 즉시 감속을 시작합니다.

단, 각 모터가 명령 정지([Motor[x].DesVelZero] = 1) 상태가 된 경우에 **dkill** 명령이 지정되면 모터는 자동으로 무효가 됩니다. 모터에 브레이크 출력이 지정된([Motor[x].pBrakeOut] > 0) 경우에 브레이크는 이 시점에서 즉시 사용되어 [Motor[x].BrakeOnDelay] 밀리초의 지연 후에 모터는 정지합니다(오픈 루프, 제로 명령 출력, 앰프 Disable).

adisable 명령에 의한 처리는 [Coord[x].AbortAllMode]가 2 로 설정된 경우의 글로벌 Abort 에 해당합니다.

동등한 버퍼 프로그램의 다이렉트 명령은 **adisable** 입니다.

adisable 명령은 2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새 기능입니다.



참고사항

adisable 명령의 결과는 IEC-61800-5-2 의 기계 안전 규격 카테고리 1 의 안전 정지와 실질적으로 같으나 이 규격에서는 제어 정지 후에 모터 또는 드라이브에서 전원을 차단해야 합니다. 이 규격에서 이 명령을 사용하면 제어 정지 후 버스 파워 또는 게이트 드라이버 전원 중 하나를 차단하기 위한 지연 릴레이도 필요합니다.

b[{상수}]

기능: 지정된 모션 프로그램의 프로그램 카운터를 포인트함

범위: 좌표계 고유

구문: **b[{상수}], begin[{상수}]**

여기에서

- **{상수}**는 양의 값으로 정수 부분은 모션 프로그램의 번호를 나타냅니다. 소수 부분은 프로그램 내의 행으로 점프하는 레이블 번호를 나타냅니다.

b[{상수}]로 지정된 좌표계는 프로그램 카운터를 현재 프로그램의 최초로 설정하거나(번호가 지정되지 않은 경우) 또는 지정된 모션 프로그램의 포인트로 설정(번호가 지정된 경우)합니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 다음의 **r** 또는 **s** 명령에 따라 이 포인트에서 모션 프로그램의 실행이 시작됩니다.

여러 좌표계 목록을 지정하더라도 후속 좌표계 고유의 명령에 대해 모달에 주소 지정된 좌표계는 변경되지 않습니다.

{상수}의 정수 부분은 프로그램 카운터가 포인트하는 모션 프로그램의 번호를 나타냅니다. 고정 모션 프로그램(**prog**)은 1 에서 $2^{31}-1$ 범위의 번호를 가질 수 있습니다. 좌표계의 로터리 모션 프로그램 버퍼는 이 명령에서는 **prog 0** 으로 간주됩니다.

{상수}의 소수 부분이 있는 경우에는 프로그램 카운터가 포인트하는 프로그램 내의 행에 점프하는 레이블 번호를 나타냅니다. 이 소수 부분은 1,000,000 을 곱하여(필요한 경우에는 끝수 버림) 행에 점프하는 레이블 번호를 취득합니다. 따라서 사용할 수 있는 행에 점프하는 레이블 번호의 범위는 **N0**:에서 **N999999**:까지입니다.

N0:에서 행에 점프하는 레이블이 명시적으로 선언되지 않은 경우에 프로그램 최초로 암묵적인 **N0**:가 설명됩니다. 따라서 **N0**:가 명시적으로 선언되지 않더라도 정수값만 지정된 **b** 명령에서는 지정된 프로그램 번호 선두에 프로그램 카운터를 포인트합니다(이 명령에서는 이 사용법이 가장 일반적입니다).

b 명령을 실행하면 **[Coord[x].ProgActive]** 플래그가 클리어되고 덮어쓰기 또는 삭제를 위해 모션 프로그램의 버퍼를 오픈할 수 있습니다.

이 명령의 생략형(**b**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**begin**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 **[Sys.NoShortCmds]**(2012 년 3/4 분기 출시된 V1.5 의 새 기능)가 1 로 설정된 경우에는 이 명령의 비생략형(**begin**)을 사용해야 합니다. 생략형(**b**)은 올바르지 않은 명령이며 예러 20 으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

동등한 버퍼 프로그램의 다이렉트 명령은 **begin** 입니다.

예/

```
b1          // Set program counter of addressed C.S. to top of prog 1
&2b5r      // Set C.S.2 program counter to top of prog 5 and run
&1b9.2      // Set C.S.1 program counter to N200000: of prog 9
&1b9.200000 // Set C.S.1 program counter to N200000: of prog 9
&1b9.200001 // Set C.S.1 program counter to N200001: of prog 9
&1b9.000002 // Set C.S.1 program counter to N2: of prog 9
&3,4b0r    // Set C.S.3 & 4 program counters to top of own rotary program
            // buffers and run
```

backup

기능: 저장 설정 요소 또는 Status 요소의 값을 통지함

범위: 글로벌

구문: **backup {구조체}[.]**

여기에서

- {구조체}**는 다음에 표시한 데이터 구조체의 이름으로 이 구조체의 저장 설정 요소가 통지됩니다.

- o [Motor[x].](모터의 데이터 구조체)
- o [Motor[x].Servo.](모터의 서보 데이터 구조체)
- o [Motor[x].Status](모터의 Status 비트)
- o [Coord[x].](좌표계의 데이터 구조체)
- o [Coord[x].Status](좌표계의 Status 비트)
- o [Gaten](ASIC 데이터 구조체)
- o [EncTable[n].](엔코더 변환표의 항목 데이터 구조체)
- o **CompTables.**(모든 보정 테이블)
- o **Sys.**(글로벌 요소)
- o **Sys.Status**(글로벌 Status 비트)
- o **Mdefs**('M' 변수 정의)

Power PMAC 은 **backup** 명령으로 지정된 데이터 구조체 또는 모든 데이터 구조체의 저장 설정 요소 또는 Status 통지 데이터 구조체 요소의 현재 활성값을 통지합니다. (이 명령은 값을 플래시 메모리에 복사하기 위한 프로세스의 일부로 **save** 명령을 통해 사용되지만)이 명령에서 값은 비휘발성 플래시 메모리에 복사되지 않습니다.

나중에 구조체명을 지정하지 않고 **backup** 명령을 사용하면 저장 데이터 구조체 요소 모든 값이 통지됩니다. 구조체명이 지정된 경우에 그 구조체의 저장 설정 요소가 통지됩니다. 인덱스 구조체([Motor[x].], [Coord[x].], [EncTable[n].])에서는 인덱스값을 상수로 지정하고 대괄호로 닫은 다음 마침표(.)를 지정해야 합니다. 2 번째 마침표를 사용하면 지정된 인덱스에서 가능한 범위의 최대 인덱스까지 모든 타입의 구조체값이 통지됩니다.

구조체명이 Status 데이터 구조체의 이름(글로벌, 모터 또는 좌표계)인 경우에 통지되는 값은 저장 설정 요소의 이름이 아니라 읽기 전용의 Status 요소 이름이 됩니다(해당 명령은 IDE 의 Status 창에서 사용됩니다).

요소의 값은 반드시 ([Coord[1].FeedTime=1000]과 같이)요소명과 부호로 시작되는 형식으로 통지되므로 이 텍스트 문자는 나중에 Power PMAC 에 값을 복원하는 경우에도 사용됩니다. 이는 다른 쿼리 명령이 통지되는 형식에 영향을 주는 **echo** 설정에 관계없이 적용됩니다.

예

```

backup Motor[5].           // Report values for Motor 5 only
backup Motor[2]..          // Report values for Motors 2 and higher
backup Motor[13].Servo.    // Report values for Motor 13 servo algorithm
backup Coord[3].           // Report values for C.S. 3
backup Coord[0]..          // Report values for all C.S.
backup Gates               // Report values for all ASICs
backup EncTable[8].        // Report values for ECT entry 8
backup EncTable[0]..       // Report values for all ECT entries
backup CompTables          // Report values for all compensation tables
backup Sys.               // Report values of global elements
backup Motor[3].Status     // Report values of Motor 3 status elements
backup Coord[2].Status     // Report values of C.S. 2 status elements

```

begin[{{상수}}]

기능: 지정된 모션 프로그램의 프로그램 카운터를 포인트함

범위: 좌표계 고유

구문: **b[{{상수}}], begin[{{상수}}]**

여기에서

- **{{상수}}**는 양의 값으로 정수 부분은 모션 프로그램의 번호를 나타냅니다. 소수 부분은 프로그램 내의 행으로 점프하는 레이블 번호를 나타냅니다.

begin[{{상수}}]는 **b[{{상수}}]** 명령의 비생략형입니다. 이 명령으로 지정된 좌표계는 프로그램 카운터를 현재 프로그램의 최초로 설정하거나(번호가 지정되지 않은 경우) 또는 지정된 모션 프로그램의 포인트로 설정(번호가 지정된 경우)합니다. [Sys.NoShortCmds]가 기본값인 0으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

bpclear

기능: 지정된 프로그램의 브레이크 포인트를 삭제함

범위: 글로벌

구문: **bpclear {프로그램명}, {행 오프셋}[.{{명령 오프셋}}]**

여기에서

- **{{프로그램명}}**은 브레이크 포인트를 취득하는 프로그램의 레이블로 **prog** 또는 **plc** 와 프로그램 번호로 구성됩니다.
- **{{행 오프셋}}**은 브레이크 포인트가 설정된 프로그램의 선두부터 행 번호를 지정하는 음의 값이 아닌 정수값입니다(3 번째의 프로그램행은 오프셋 2가 됩니다).
- **{{명령 오프셋}}**은 브레이크 포인트가 설정된 프로그램행의 선두부터 명령 번호를 지정하는 음의 값이 아니라 정수값입니다(4 번째 명령은 오프셋 3이 됩니다).

bpclear 명령에 따라 Power PMAC 은 지정된 포인트에서 지정된 스크립트 프로그램의 브레이크 포인트를 삭제합니다. 명령이 어떤 처리를 실행하려면 프로그램명, 행의 오프셋 및 명령 오프셋에 따라 명령으로 지정된 브레이크 포인트가 기존의 브레이크 포인트와 일치해야 합니다. 브레이크 포인트가 일치하지 않는 경우에 처리는 발생하지 않습니다. 기존의 브레이크 포인트를 클리어하면 대상 브레이크 포인트의 데이터 구조체 [Bp[i]]가 클리어됩니다. 상위 인덱스값에 아직 브레이크 포인트가 남아 있는 경우에 해당 브레이크 포인트의 인덱스값이 감소하여 남은 브레이크 포인트의 인덱스와의 불일치가 없어집니다.

프로그램은 **prog 7** 또는 **plc 3** 등의 이름으로 지정됩니다. 프로그램 내의 브레이크 포인트는 행 오프셋과 명령 오프셋으로 지정됩니다. 행 오프셋은 브레이크가 발생하는 행 번호를 프로그램의 선두에서 지정합니다. 이는 행 번호보다 1 적은 값이 됩니다. 옵션의 명령 오프셋은 브레이크가 설정되는 행의 시작에서 다른 명령의 수를 지정합니다. 이는 1개의 프로그램 명령에 다수의 명령이 설정된 경우에 편리합니다. 이는 명령 번호보다 1개 적은 값이 됩니다.

bpclearall

기능: 모든 브레이크 포인트를 삭제함

범위: 글로벌

구문: **bpclearall**

bpclearall 명령을 통해 Power PMAC 은 확립된 모든 브레이크 포인트를 삭제합니다. 모든 [Bp[i]] 데이터 구조체가 클리어되므로 이러한 구조체의 모든 요소값은 'NaN'(숫자가 아님)으로 통지됩니다.

bpset

기능: 지정된 프로그램의 브레이크 포인트를 정의함

범위: 글로벌

구문: **bpset {프로그램명}, {행 오프셋}[{명령 오프셋}]:{반복}**

여기에서

- **{프로그램명}**은 브레이크 포인트를 취득하는 프로그램의 레이블이며 **prog** 또는 **plc** 및 프로그램 번호로 구성됩니다.
- **{행 오프셋}**은 브레이크 포인트가 설정된 프로그램 선두의 행 번호를 지정하는 음의 값이 아닌 정수값입니다(3 번째 프로그램 행은 오프셋 2 가 되는 등).
- **{명령 오프셋}**은 브레이크 포인트가 설정된 프로그램 행 선두의 명령 번호를 지정하는 음의 값이 아닌 정수값입니다(4 번째 명령은 오프셋 3 이 되는 등).
- **{반복}**은 명령의 실행에서 브레이크 포인트가 몇 회 있으면 프로그램이 중단할지 그 수를 지정하는 음의 값이 아닌 정수값입니다({반복}이 3 인 경우 4 회째의 브레이크 포인트에서 브레이크가 중단하는 등).

Power PMAC 은 **bpset** 명령에 따라 지정된 반복 다음에 지정된 포인트에서 지정된 스크립트 프로그램의 브레이크 포인트를 설정합니다. 이 명령은 데이터 구조체 [Bp[i]]를 설정하여 브레이크 포인트에 대응합니다. 최초의 정의된 브레이크 포인트는 [Bp[0]]을 사용합니다. 후속 명령에서는 다음의 이용 가능한 인덱스값이 사용됩니다.

프로그램은 **prog 7** 또는 **plc 3** 등의 이름으로 지정됩니다.

프로그램 내의 브레이크 포인트는 행 오프셋과 명령 오프셋으로 지정됩니다. 행 오프셋은 브레이크가 발생하는 행 번호를 프로그램의 선두에서 지정합니다. 이는 행 번호보다 1 적은 값이 됩니다. 프로그램의 n 행째에 브레이크를 발생시키려면 행 오프셋의 값을 $n-1$ 로 설정합니다. 즉 프로그램의 17 행째에 브레이크를 발생시키는 경우에는 행 오프셋의 값을 16 으로 설정합니다.

오프셋의 명령 오프셋은 브레이크가 설정되는 행의 시작에서 다른 명령의 수를 지정합니다. 이는 1 개의 프로그램 명령에 다수의 명령이 설정된 경우에 편리합니다. 이 값은 명령 번호보다 1 적은 값이 됩니다. 프로그램행 n 번째 명령 전에 브레이크를 발생시키려면 명령 오프셋의 값을 $n-1$ 로 설정합니다. 즉 프로그램의 3 번째 명령 전에 브레이크를 발생시키는 경우에는 명령 오프셋의 값을 2 로 설정합니다. 명령 오프셋을 지정하지 않는 경우에는 제로값이 사용되며 브레이크는 그 행의 최초 명령 전에서 발생합니다.

반복값은 프로그램의 실행이 브레이크 포인트에서 중단되기 전에 지정한 명령(즉 브레이크 포인트 직후의 명령)이 실행되는 횟수를 지정합니다. 이 값을 제로로 설정하면 브레이크 포인트가 최초로 출현했을 때 프로그램의 실행은 중단됩니다. 이 값을 99로 설정하면 99번의 브레이크 포인트는 지나치고 100번째 브레이크 포인트에서 프로그램의 실행이 중단됩니다.

bpset 명령이 전송되면 데이터 구조체 요소 **[Bp[i].Count]**는 명령 내에 지정된 반복 수로 설정됩니다. 이 요소값이 제로보다 큰 경우에 프로그램 실행 중의 브레이크 포인트마다 요소값을 1씩 줄입니다. 이 요소값이 제로로 설정되면 브레이크 포인트마다 프로그램의 실행이 중단됩니다. 이 반복값을 -1로 설정하면 **[Bp[i].Count]**는 감량하더라도 0이 되지 않으므로 브레이크 포인트는 프로그램을 중단하지 않고 사이클 카운터로 사용할 수 있습니다.

[Bp[i].Count]의 원래 값은 유지되지 않는다는 점에 주의하십시오. 프로그램의 실행을 단순히 재개한 경우에 최초로 지정한 반복 횟수와는 관계없이 프로그램은 다음의 브레이크 포인트에서 중단됩니다. 프로그램이 다시 중단되기 전에 반복 횟수가 필요한 경우에는 같은 브레이크 포인트에 제로 이외의 값의 반복 횟수로 **bpset** 명령을 재전송하거나 요소 **[Bp[i].Count]**에 직접 값을 할당합니다.

brickacver

기능: Brick AC 앰프의 펌웨어 버전을 통지함

범위: 글로벌

구문: **brickacver**

Power PMAC 프로세서는 **brickacver** 명령으로 Brick AC 앰프의 펌웨어를 쿼리하여 앰프 펌웨어의 현재 버전을 통지합니다. 버전은 **{버전}.{출사}** 형식으로 통지됩니다. 이 값은 비저장 설정 요소 **[BrickAC.Monitor]**를 1로 설정한 경우에 한해 통지됩니다.



앰프의 펌웨어 버전은 **vers** 명령을 사용하여 쿼리할 수 있는 Power PMAC CPU의 펌웨어 버전과는 다릅니다.

참고사항

bricklvver

기능: Brick LV 앰프의 펌웨어 버전을 통지함

범위: 글로벌

구문: **bricklvver**

Power PMAC 프로세서는 **bricklvver** 명령으로 Brick LV 앰프의 펌웨어를 쿼리하여 앰프 펌웨어의 현재 버전을 통지합니다. 버전은 **{버전}.{출사}** 형식으로 통지됩니다. 이 값은 비저장 설정 요소 **[BrickLV.Monitor]**를 1로 설정한 경우에 한해 통지됩니다.



앰프의 펌웨어 버전은 **vers** 명령을 사용하여 쿼리할 수 있는 Power PMAC CPU의 펌웨어 버전과는 다릅니다.

참고사항

buffer

기능: Power PMAC 버퍼 내의 프로그램을 통지함

범위: 글로벌

구문: **buffer**

Power PMAC은 **buffer** 명령으로 프로그램 메모리 내의 프로그램 버퍼를 통지합니다. 각 프로그램의 버퍼마다 이름, 프로그램 버퍼 메모리 선두에서의 오프셋(바이트 단위) 및 바이트 단위의 버퍼 크기를 통지합니다.

예

```
buffer
Prog1 offset 38655, size 436
Prog5 offset 254, size 151
Prog6 offset 405, size 2179
Prog10 offset 38580, size 75
Prog15 offset 41246, size 413
Prog25 offset 2950, size 39
Prog26 offset 2989, size 50
SubProg1000 offset 37205, size 1375
SubProg100000 offset 39091, size 150
SubProg100001 offset 39241, size 283
SubProg100002 offset 39524, size 166
SubProg100003 offset 39690, size 127
SubProg100004 offset 39817, size 172
SubProg100005 offset 39989, size 615
SubProg100006 offset 40604, size 642
Plc1 offset 2584, size 158
Plc2 offset 42367, size 396
Plc3 offset 42763, size 152
Plc5 offset 41659, size 384
Plc6 offset 42043, size 324
&1Inverse offset 37136, size 69
&1Forward offset 37023, size 113
```

clear all buffers

기능: 로터리 모션 프로그램의 모든 버퍼 내용을 삭제함

범위: 글로벌

구문: **clear all buffers**

Power PMAC은 **clear all buffers** 명령으로 모든 로터리 모션 프로그램의 버퍼 내용을 삭제하고 나아가 모션 프로그램의 버퍼, PLC 프로그램의 버퍼 및 키네마틱스 서브루틴의 버퍼 모두를 삭제합니다.

이 명령이 전송된 경우에 로터리 모션 프로그램의 버퍼는 모두 클로즈해야 합니다. **h**(유지) 명령, **q**(종료) 명령, **s**(스텝) 명령을 사용하거나 프로그램 실행이 로드된 로터리 버퍼의 종료점에 도달한 경우와 마찬가지로 모든 모션 프로그램(로터리 및 고유)은 일시 중단하는 것이 아니라 완전히 정지해야 합니다. 이는 **a**(Abort) 명령 사용 또는 프로그램의 실행이 고정 프로그램 버퍼 종료점에서 종료한 경우와 마찬가지로입니다.

로터리 모션 프로그램의 버퍼의 구조 자체는 남기 때문에 구조에 새로운 내용을 입력할 수 있습니다.

clear plc {상수}

기능: 지정한 PLC 프로그램의 버퍼의 내용을 삭제함

범위: 글로벌

구문: **clear plc {상수}**

여기에서

- **{상수}**는 0~31 범위의 정수에서 삭제할 PLC 프로그램의 번호를 지정합니다.

Power PMAC 은 **clear plc {상수}** 명령으로 지정한 스크립트 PLC 프로그램의 버퍼 내용을 삭제합니다. 이 처리는 명령 조합 **open plc {상수} close**에 해당합니다.

clear prog {상수}

기능: 지정한 모션 프로그램의 버퍼 내용을 삭제함

범위: 글로벌

구문: **clear prog {상수}**

여기에서

- **{상수}**는 음의 값이 아닌 32 비트 정수에서 삭제할 모션 프로그램의 번호를 지정합니다.

Power PMAC 은 **clear prog {상수}** 명령으로 지정한 스크립트 모션 프로그램의 버퍼 내용을 삭제합니다. 이 처리는 명령 조합 **open prog {상수} close**에 해당합니다.

clear rotary

기능: 로터리 모션 프로그램의 버퍼 내용을 삭제함

범위: 좌표계 고유

구문: **clear rotary**

Power PMAC 은 **clear rotary** 명령으로 주소 지정된 좌표계의 로터리 모션 프로그램의 버퍼 내용을 삭제합니다.

이 명령이 전송된 경우에 좌표계의 로터리 버퍼는 클로즈해야 합니다. **h**(유지) 명령, **q**(종료) 명령, **s**(스텝) 명령을 사용하거나 프로그램의 실행이 버퍼 종료점에 도달한 경우와 마찬가지로 프로그램의 실행은 일시 중단하는 것이 아니라 완전히 정지해야 합니다. 이는 **a**(Abort) 명령을 사용한 경우와 마찬가지로입니다.

로터리 버퍼의 구조 자체는 남기 때문에 구조에 새로운 내용을 입력할 수 있습니다. 프로그램 포인터는 버퍼의 선두로 설정되므로 **r**(실행) 명령 또는 **s**(스텝) 명령으로 실행을 즉시 재개할 수 있습니다.

close

기능: 현재 오픈한 프로그램 버퍼를 클로즈함

범위: 글로벌

구문: **close**

close 명령을 사용하면 Power PMAC 은 어떤 타입(**prog**, **plc**, **subprog**, **forward**, **inverse**)의 프로그램이라도 같은 통신 스레드에서 오픈된 경우에 현재 오픈한 프로그램 버퍼를 클로즈합니다.

버퍼 입력 및 삭제 준비에 사용하는 **open** 명령(**open prog 73 close** 등) 직후에 **close** 명령이 전송된 경우에는 공 버퍼로 남지 않고 프로그램 버퍼 자체가 삭제됩니다.

이 명령이 전송되어 오픈된 프로그램 버퍼가 없는 경우에 명령은 에러 없이 수용되지만 아무것도 처리하지 않습니다.

close all buffers

기능: 임의의 스레드가 현재 오픈된 모든 프로그램 버퍼를 클로즈함

범위: 글로벌

구문: **close all buffers**

close all buffers 명령을 사용하면 Power PMAC 은 어느 타입(**prog**, **plc**, **subprog**, **forward**, **inverse**)의 프로그램 버퍼인지 그리고 어느 통신 스레드가 오픈되었는지 여부와 관계없이 현재 오픈된 프로그램 버퍼를 클로즈합니다. 이 명령을 사용할 때에는 다른 스레드 처리와 간섭을 일으키는 경우가 있으므로 충분히 주의하십시오. 이 명령은 다른 스레드가 충돌된 경우의 복구에 사용됩니다.

오픈한 프로그램 버퍼가 없는 경우에 명령은 에러 없이 수용되지만 아무것도 처리하지 않습니다.

cpu

기능: CPU의 타입을 통지함

범위: 글로벌

구문: **cpu**

Power PMAC은 **cpu** 명령으로 사용된 CPU의 타입과 번호를 통지합니다.

예

```
cpu // Query CPU type
Power PC, 460EX // Power PMAC response
cpu // Query CPU type
Power PC, 86xxx // Power PMAC response
```

cpx

기능: 직후의 프로그램 명령을 컴파일하여 모션 프로그램으로 실행함

범위: 좌표계 고유

구문: **cpx {프로그램 명령}**

cpx 명령을 사용하면 Power PMAC은 같은 행의 직후에 이어지는 프로그램 명령 시퀀스를 컴파일(구문 해석)하여 모션 프로그램으로 실행합니다. 이에 따라 오픈/다운로드/클로즈/실행의 사이클 전체를 경유하지 않고 여러 기본적인 버퍼 프로그램 명령을 실행할 수 있는 간단한 방법이 제공됩니다.

cpx 명령으로 실행할 수 있는 버퍼 프로그램 명령은 다음과 같습니다.

- [while] 루프의 단일행
- [do .. while] 루프의 단일행
- [if]조건문에서 분기한 단일행
- 변수 수정의 할당(**P1+=1** 등)
- 축 동작(모든 동작 모드)

버퍼 프로그램 명령 또는 온라인 명령 중 하나에서 적절한 명령(**{변수}={식}**)은 **cpx** 명령을 먼저 설명하지 않아도 됩니다.

축 동작 또는 좌표계 변수 변경 등의 좌표계 고유 처리에서는 현재 통신 스레드에서 주소 지정된 좌표계에 한해 영향을 받습니다.

명령을 모션 프로그램으로 실행하기 위해 **pmatch** 명령은 시작 축 위치를 자동으로 계산하여 어느 내장 명령보다 먼저 자동적, 묵시적으로 실행됩니다.

예

```
cpx while (P100<40) P(P100)=0 P100++ // Execute WHILE loop
cpx if (Q10<0) Q300=72 // Execute conditional branch
cpx Motor[4].ProgJogPos+=5000 // Modify data structure element value
cpx linear X10 Y20 Z30 F5 // Execute linear-mode move
```


CX

기능: 직후의 프로그램 명령을 컴파일하여 PLC 로 실행함

범위: 좌표계 고유

구문: **cx {프로그램 명령}**

cx 명령을 사용하면 Power PMAC 은 같은 행의 직후에 이어지는 프로그램 명령 시퀀스를 컴파일(구문 해석)해서 PLC 프로그램으로 실행합니다. 이에 따라 오픈/다운로드/클로즈/실행의 사이클 전체를 경유하지 않고 여러 기본적인 버퍼 프로그램 명령을 실행할 수 있는 간단한 방법이 제공됩니다.

cx 명령으로 실행할 수 있는 버퍼 프로그램 명령은 다음과 같습니다.

- [while] 루프의 단일행
- [do .. while] 루프의 단일행
- [if]조건문에서 분기한 단일행
- 변수 수정의 할당(**P1+=1** 등)
- 축 동작(좌표계를 강제로 Rapid 모드로 설정)

버퍼 프로그램 명령 또는 온라인 명령 중 하나에서 적절한 명령(예를 들어 **{변수}={식}**)은 **cx** 명령을 먼저 설명하지 않아도 됩니다.

축 동작 또는 좌표계 변수 변경 등의 좌표계 고유 처리에서는 현재 통신 스레드에서 주소 지정된 좌표계에 한해 영향을 받습니다.

명령을 PLC 프로그램으로 실행하기 위해 **pmatch** 명령은 동작 명령의 시작 축 위치와 현재 모터 위치의 올바른 일치 여부를 확인하도록 자동적, 묵시적으로 실행되는 경우는 없습니다. 자동적, 묵시적인 실행이 필요한 경우에는 명령을 명시적으로 설명해야 합니다.

예

```
cx while (P100<40) P(P100)=0 P100++ // Execute WHILE loop
cx if (Q10<0) Q300=72 // Execute conditional branch
cx Motor[4].ProgJogPos+=5000 // Modify data structure element value
cx pmatch X10 Y20 Z30 // Execute RAPID-mode move
```

{데이터 구조체 요소}

기능: 데이터 구조체 요소의 값을 통지함

범위: 글로벌

구문: **{데이터 구조체 요소}**

여기에서

- **{데이터 구조체 요소}**는 사전에 정의된 데이터 구조체에 포함된 개별적인 요소의 이름입니다.

Power PMAC 은 **{데이터 구조체 요소}** 명령으로 지정된 데이터 구조체 요소의 현재값을 통지합니다. Power PMAC 의 스크립트 언어에서는 요소의 이름은 대문자와 소문자를 구별하지 않습니다(C 프로그램에서 데이터 구조체 요소에 액세스하는 경우의 이름은 대문자와 소문자를 구별한다는 점에 주의하십시오). 이 통신 스레드의 로컬 데이터 구조체의 일부인 요소를 제외하고 요소의 정식 이름은 해당 요소가 속한 구조체를 포함하여 지정해야 합니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 0(값 1)은 데이터 구조체의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

```
echo0 // Include variable names in query responses
Sys.ServoPeriod // Query value of system's servo update period
Sys.ServoPeriod=0.250 // Power PMAC response of value (in msec)
Motor[3].JogSpeed // Query value of Motor 3's commanded jog speed
Motor[3].JogSpeed=38 // Power PMAC response of value (in motor units/ms)
echo7 // Do not include variable names in query responses
Coord[2].MaxFeedrate // Query value of C.S. 3's maximum feedrate
25 // Power PMAC response of the value (in user units)
Plc[5].Ldata.Motor // Query value of PLC 5's addressed motor
7 // Power PMAC response of the value (Motor 7)
Ldata.Motor // Query value of thread's own addressed motor
4 // Power PMAC response of the value (Motor 4)
```

{데이터 구조체 요소}.a

기능: 데이터 구조체 요소의 주소를 통지함

범위: 글로벌

구문: **{데이터 구조체 요소}.a**

여기에서

- **{데이터 구조체 요소}**는 사전에 정의된 데이터 구조체에 포함된 개별적인 요소의 이름입니다.

Power PMAC 은 **{데이터 구조체 요소}.a** 명령으로 지정된 데이터 구조체 요소의 절대 주소를 통지합니다. 이 명령은 주로 진단을 목적으로 사용됩니다. 따라서 사용자는 데이터 구조체 요소의 주소 수치를 알 필요는 없습니다. 펌웨어 버전이 변경되면 요소의 절대 주소도 변경되는 경우가 있습니다.

또한 **{데이터 구조체 요소}.a** 는 수식의 변수값으로도 사용할 수 있습니다.

Power PMAC 의 스크립트 언어에서는 요소의 이름은 대문자와 소문자를 구별하지 않습니다(C 프로그램에서 데이터 구조체 요소에 액세스하는 경우의 이름은 대문자와 소문자를 구별한다는 점에 주의하십시오). 이 통신 스레드의 데이터 구조체 일부인 요소를 제외하고 요소의 정식명은 해당 요소가 속한 구조체를 포함하여 지정해야 합니다.

요소의 주소가 0 으로 반환된 경우에 그 요소는 함수 요소이며 레지스터 요소가 아닙니다. 요소값에 액세스하면 단순히 레지스터 내의 값을 읽는 것이 아니라 수학/논리 함수의 실행을 호출합니다. 함수 요소는 동기 대입으로 사용할 수 없으며 C 프로그램에서 직접 액세스할 수도 없습니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 0(값 1)은 데이터 구조체의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

```
echo0 // Include variable names in query responses
Gate1[4].Chan[0].Dac[0].a // Query value of DAC output address
Gate1[4].Chan[0].Dac[0].a=$f968008 // Power PMAC response of address

echo7 // Do not include variable names in query responses
Gate1[4].Chan[0].Dac[0].a // Query value of DAC output address
$f968008 // Power PMAC response of address
```

{데이터 구조체 요소}={식}

기능: 데이터 구조체 요소의 값을 설정함

범위: 글로벌

구문: {데이터 구조체 요소}={식}

여기에서

- {데이터 구조체 요소}는 사전에 정의된 데이터 구조체에 포함된 개별적인 요소의 이름입니다.
- {식}은 지정된 데이터 구조체 요소에 할당되는 값을 포함하는 수식입니다.

Power PMAC 은 {데이터 구조체 요소}={식} 명령으로 데이터 구조체 요소의 값을 부호의 오른쪽 식의 값으로 설정합니다. Power PMAC 의 스크립트 언어에서는 요소의 이름은 대문자와 소문자를 구별하지 않습니다(C 프로그램에서 데이터 구조체 요소에 액세스하는 경우의 이름은 대문자와 소문자를 구별한다는 점에 주의하십시오). 이 통신 스레드의 데이터 구조체 일부인 요소를 제외하고 요소의 정식명은 해당 요소가 속한 구조체를 포함하여 지정해야 합니다.

예

```
Sys.ServoPeriod=0.250 // Set val of system's servo update period to 0.250 (ms)
Motor[3].JogSpeed=38 // Set value of Motor 3's commanded jog speed
Coord[2].MaxFeedrate=25 // Set value of C.S. 2's maximum feedrate
Plc[5].Ldata.Motor=7 // Set value of PLC 5's addressed motor
Ldata.Motor=4 // Set value of thread's own addressed motor
```

d

기능: 명령 위치의 값을 통지함

범위: 모터 또는 좌표계 고유

구문: d

Power PMAC 은 **d** 명령으로 지정된 모터 또는 좌표계의 명령 위치의 현재값을 통지합니다. 모터 목록 또는 좌표계 목록이 없는 경우에는 현재 주소 지정된 모터의 명령 위치값을 통지합니다. 모터 목록이 있는 경우에는 목록 내의 모든 모터의 명령 위치값을 통지합니다. 좌표계 목록이 있는 경우에는 목록 내의 모든 좌표계의 활성인 모든 축의 명령 위치값을 통지합니다.

여러 모터 목록 또는 여러 좌표계 목록을 지정해도 후속 모터 고유의 명령 또는 좌표계 고유의 명령으로 모달에 주소 지정된 모터 또는 좌표계는 변경되지 않습니다.

모터 위치를 통지하는 경우에 위치는 기본 모터 단위로 표시됩니다. 이 통지의 참조(제로)점은 오프셋 모드 명령 위치 레지스터 내의 값의 사용 방법을 결정하는 2개의 제어 설정에 좌우됩니다. 오프셋 모드 명령 위치 레지스터에서는 몇 개의 위치 명령 소스 중첩이 허용됩니다.

일반적으로 캠 테이블 또는 캠 스타일의 보정 테이블 출력인 **[Motor[x].CompDesPos]**는 항상 오프셋 모드의 레지스터입니다. 일반적으로 위치 추종(전자 기어)의 결과인 **[Motor[x].ActiveMasterPos]**는 **[Motor[x].MasterCtrl]**의 비트 1(값 2)을 1로 설정한 경우에 오프셋 모드의 레지스터입니다.

[Motor[x].PosReportMode](2013 년 3/4 분기에 출시된 V1.6 펌웨어의 새 기능)를 기본값인 0으로 설정한 경우 또는 이 새로운 요소가 없는 이전 버전의 펌웨어를 사용한 경우에는 이러한 오프셋 모드의 레지스터값을 최종적인 명령 위치에서 뺍니다. **[Motor[x]]**의 소스 레지스터에서 작업하는 경우에 통지된 위치는 다음과 같이 계산됩니다.

```
+ DesPos           // 전원 셋업/리셋 시의 위치에 대한 명령 상대 위치
- CompDesPos       // 보정 테이블 명령 위치 명령
- ActiveMasterPos * MasterCtrlOffsetBit // 오프셋 모드 시의 마스터 위치
- HomePos          // 전원 셋업/리셋 시의 위치에 대한 모터 제로 상대 위치
```

[Motor[x].PosReportMode]를 1로 설정하면 오프셋 모드의 이러한 레지스터값을 최종적인 명령 위치에서 빼지 않습니다. 이 경우에 통지되는 위치는 다음과 같이 계산됩니다.

```
+ DesPos           // 전원 셋업/리셋 시의 위치에 대한 명령 상대 위치
- HomePos          // 전원 셋업/리셋 시의 위치에 대한 모터 제로 상대 위치
```

양쪽의 경우에 명령 위치값 **[DesPos]**는 궤적 명령 위치 **[Desired.Pos]**, 테이블에 기초하여 명령 위치 **[CompDesPos]** 및 다음의 명령 위치 **[ActiveMasterPos]**의 합계가 됩니다.

축 위치를 통지하는 경우에 위치는 축의 프로그램 원점 위치(반드시 기본 모터의 제로 위치와 같지 않음)에 대해 스케일링된 사용자(공학) 단위로 표시됩니다. 각 축에 통지된 값은 축의 이름 (1개 또는 2개의 영문자)앞에 설명합니다. Power PMAC 은 그 위치가 이 명령으로 계산하는 각 축의 값을 통신 스레드 $L(256 + [\text{Sys.MaxMotors} + n])$ 을 위한 로컬 변수로 남겨 둡니다. 여기서 n 은 0~31까지의 축 인덱스값(A는 0, B는 1 등)입니다.

그 좌표계의 축에 모터가 할당되지 않은 좌표계에서 통지가 요구되면 Power PMAC 은 문자열 'No Motors'를 반환합니다. 좌표계의 축 정의 셋팅 또는 포워드 키네마틱스 서브루틴으로 축 데이터를 적절하게 계산할 수 없는 경우에 Power PMAC 은 문자열 'No Solution'을 반환합니다.

예

```
d // Query desired position of presently addressed motor
1001 // Power PMAC response in counts

#1..4d // Query desired positions of Motors 1 - 4
982 -3462 27 86643 // Power PMAC response in counts

&1d // Query desired positions of axes in C.S. 1
X0.982 Y-3.462 Z0.027 C86.643 // Power PMAC response in user units
```

D{데이터}

기능: 'D' 변수값을 통지함

범위: 통신 스레드 고유

구문: **D{데이터}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.

Power PMAC 은 **D{데이터}** 명령으로 통신 스레드가 지정된 로컬 'D' 변수의 현재값을 통지합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값(0~54)이어야 합니다. 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

통신 스레드용의 'D' 변수는 어느 좌표계용의 'D' 변수에도 대응하지 않으므로 좌표계의 'D' 변수값 쿼리에는 이 명령은 사용할 수 없습니다. 쿼리를 하는 경우에는 데이터 구조체 요소 [Coord[x].Ldata.D[i]]의 값을 사용하십시오.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

```
echo0 // Include variable names in query responses
D1 // Query value of variable D1
D1=17.5 // Power PMAC response of the value of D1

D(P1) // Query value of D-variable numbered by P1
D17=3 // Power PMAC response of the value of D17

D(P1+P17) // Query value of D-variable numbered by P1+P17
D20=-5.35 // Power PMAC response of the value of D20

echo7 // Do not include variable names in query responses
D1 // Query value of variable D1
17.5 // Power PMAC response of the value of D1
```

D{데이터}={식}

기능: 값을 'D' 변수에 할당함

범위: 통신 스레드 고유

구문: **D{데이터}={식}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.
- **{식}**은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC 은 **D{데이터}={식}** 명령으로 통신 스레드가 지정된 로컬 'D' 변수에 부호의 오른쪽 식의 값을 설정합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값(0~54)이어야 합니다. 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

명령이 Power PMAC 에 송신되고 프로그램 버퍼가 오픈되면 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

각 통신 스레드에는 자신의 'D' 변수 세트가 있습니다. 또한 각 좌표계에는 모션 프로그램에서 사용하는 좌표계 자체의 세트가 있으며 각 PLC 자체에도 세트가 있습니다. 'D' 변수의 모든 세트는 각각 독립적입니다. 통신 스레드용의 'D' 변수는 어느 좌표계용의 'D' 변수에도 대응하지 않으므로 좌표계의 'D' 변수값 설정에 이 명령은 사용할 수 없습니다.

예

```
D1=17.5           // Set variable D1 to 17.5
D(P1)=3           // Set D-variable numbered by P1 to 3
D(P1+P17)=5*sqrt(P100) // Set D-var numbered by (P1+P17) to expression value
```

D{변수 목록}

기능: 목록의 'D' 변수값을 통지함

범위: 통신 스레드 고유

구문: **D{변수 목록}[:{상수}]**

여기에서

- **{변수 목록}**은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.
 - 0 **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작되고 2 번째 정수값(최초보다 큰 번호여야 합니다)으로 번호를 할당된 변수로 종료하는 연속적인 번호의 변수 범위를 지정합니다.

- **{상수}[{상수}[{상수}]}**에서 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- 옵션의 **{상수}**는 응답 행마다 통지된 값의 수를 지정하는 양의 정수입니다. 여기서 값을 지정하지 않는 경우에 응답 행마다 하나의 값만 통지됩니다. 값을 지정하지 않고 사용할 수 있는 것은 목록이 연속된 수의 범위로 지정된 경우에 한합니다.

Power PMAC 은 **D{변수 목록}** 명령으로 통신 스레드 목록 내의 로컬 'D' 변수의 현재값을 통지합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC に송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

각 통신 스레드에는 자신의 'D' 변수 세트가 있습니다. 또한 각 좌표계에는 모션 프로그램에서 사용하는 좌표계 자체의 세트가 있으며 각 PLC 자체에도 세트가 있습니다. 'D' 변수의 모든 세트는 각각 독립적입니다. 통신 스레드용의 'D' 변수는 어느 좌표계용의 'D' 변수에도 대응하지 않으므로 좌표계의 'D' 변수값 쿼리에는 이 명령은 사용할 수 없습니다. 쿼리를 하는 경우에는 데이터 구조체 요소 [Coord[x].Ldata.D[i]]의 값을 사용하십시오.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

```
D1 // Query value of variable D1 (set quantity 1, spacing 1)
D1=17.5 // Power PMAC response of the value of D1

D10..12 // Query value of D100, D101, D102
D10=-4 // Power PMAC response of the value of D10
D11=3.14159 // Power PMAC response of the value of D11
D12=0 // Power PMAC response of the value of D12

D30,3 // Query value of 3 D-vars, starting at D30 (spacing 1)
D30=-5.35 // Power PMAC response of the value of D30
D31=7.12 // Power PMAC response of the value of D31
D32=376452 // Power PMAC response of the value of D32

D30,3,3 // Query value of 3 D-vars, starting at D30, spacing 3
D30=-5.35 // Power PMAC response of the value of D30
D33=99 // Power PMAC response of the value of D33
D36=-0.002 // Power PMAC response of the value of D36

D1..10:5 // Query value of D1 - D10, 5 values per response line
D1=7,3,5,2,11 // Power PMAC response of the value of D1 - D5
D6=8,33,4,9,-2 // Power PMAC response of the value of D6 - D10
```

D{변수 목록}={식}

기능: 목록의 'D' 변수값을 설정함

범위: 통신 스레드 고유

구문: **D{변수 목록}={식}**

여기에서

- **{변수 목록}**은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.
 - o **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작되고 2 번째 정수값(최초보다 큰 번호여야 합니다)으로 번호를 할당된 변수로 종료하는 연속적인 번호의 변수 범위를 지정합니다.
 - o **{상수}[,{상수}[,{상수}]**에서 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- **{식}**은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC 은 **D{변수 목록}={식}** 명령으로 통신 스레드가 지정된 목록 로컬 'D' 변수에 부호의 오른쪽 식의 값으로 설정합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC に송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

각 통신 스레드에는 자신의 'D' 변수 세트가 있습니다. 또한 각 좌표계에는 모션 프로그램에서 사용하는 좌표계 자체의 세트가 있으며 각 PLC 자체에도 세트가 있습니다. 'D' 변수의 모든 세트는 각각 독립적입니다. 통신 스레드용의 'D' 변수는 어느 좌표계용의 'D' 변수에도 대응하지 않으므로 좌표계의 'D' 변수값 설정에 이 명령은 사용할 수 없습니다.

예

D1=17.5	// Set value of D1 (set quantity 1, spacing 1) to 17.5
D10..12=P1+5	// Set value of D10, D11, D12 to (P1+5)
D30,3=-7	// Set value of 3 D-vars, starting at D30 (spacing 1) to -7
D30,3,3=sqrt(P25)	// Set value of 3 D-vars, starting at D30, spacing 3
	// (D30, D33, D36) to the square root of P25

date

기능: 펌웨어의 발행일을 통지함

범위: 글로벌

구문: **date**

Power PMAC 은 **date** 명령으로 사용한 펌웨어의 발행일을 통지합니다. 발행일은 다음의 형식으로 통지됩니다.

- 월을 나타내는 영문 명칭의 최초 3 문자(Apr 등)
- 2 자리 숫자로 나타낸 월의 날짜
- 4 자리 숫자로 나타낸 년(서기)

예

date	// Query date of issuance of firmware
Jan 30 2008	// Power PMAC response

ddisable

기능: 좌표계의 모든 모터의 Disable 을 지연함

범위: 좌표계 고유

구문: **ddisable**

Power PMAC 은 **ddisable** 명령으로 지정한 좌표계의 모터 모든 제어에 대한 Disable(출력 정지)의 지연을 실행합니다. 이는 구체적으로 좌표계의 모터 모두에 대한 모터 고유의 **dkill**(지연된 출력 정지) 명령에 상당하지만 모터 명령과는 달리 모션 프로그램을 실행하는 좌표계에 작용하여 처리 중인 모션 프로그램을 중단시킵니다.

좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. **&*ddisable** 명령은 Power PMAC 상의 모든 좌표계의 모터를 Disable 합니다. **dkill** 모터 명령은 지정한 모터에 지연된 출력 정지를 실행한다는 점에 주의하십시오.

모터를 출력 정지하면 서보 루프가 오픈되고 서보 출력값이 강제로 제로가 됩니다(단, 출력 바이어스 항은 적용됩니다). 다음으로 앰프 Enable 신호가 강제로 무효 상태가 됩니다.

[Motor[x].pBrakeOut]을 제로 이외의 값(브레이크 출력 레지스터의 주소)으로 설정하고 모터의 자동 브레이크 제어 기능이 유효한 경우에 브레이크 출력은 **ddisable** 명령으로 즉시 연동하도록 명령합니다. 게다가 모터가 클로즈드 루프, 제로 속도 상태인 경우에 모터의 실제 출력 정지는 [Motor[x].BrakeOnDelay] 밀리초만큼 지연되어 브레이크 시스템이 완전히 연동될 시간을 제공합니다.

이 지연 Disable 명령은 브레이크가 연동하는 충분한 시간이 필요한 브레이크 장착 모터의 무효화를 목적으로 하고 있습니다. 모터의 긴급한 Disable 은 마찬가지로 즉시 **disable** 명령을 사용해야 합니다.

모터의 클로즈드 루프를 Enable 하는 제어는 **j/** 명령으로 재개할 수 있습니다. 좌표계 내 모터의 모든 클로즈드 루프를 Enable 하는 제어는 **enable** 명령으로 재개할 수 있습니다. 모터의 오픈 루프를 유효화하는 제어는 **out{상수}** 명령으로 재개할 수 있습니다.

온라인 지연 Disable 명령(&xddisable) 및 버퍼 프로그램 지연 Disable 명령(ddisablex)의 구문이 다르므로 주의하십시오.

define lookahead

기능: 룩 어헤드 버퍼를 정의함

범위: 좌표계 고유

구문: **define lookahead {상수}**

여기에서

- **{상수}**는 32 비트 양의 정수로 좌표계의 각 모터용 버퍼에 저장할 수 있는 동작 세그먼트의 수를 나타냅니다. 이 수치는 1,024 이상이어야 합니다.

Power PMAC 은 **define lookahead** 명령을 사용하여 주소 지정된 좌표계의 룩 어헤드 버퍼를 생성하고 지정한 수의 동작 세그먼트 정보를 저장하는 충분한 메모리를 할당합니다. Power PMAC 은 룩 어헤드 버퍼로 좌표계 내의 모든 모터에 위치, 속도 및 가속도 리미트의 위반이 없는지 먼저 프로그램된 궤적을 확실히 확인합니다. 필요에 따라 리미트 위반이 발생하지 않도록 프로그램된 궤적 속도를 감속합니다.

이 명령이 전송되면 Power PMAC 은 필요한 메모리량의 할당을 결정하기 위해 축 정의 구문 또는 키네마틱스 서브루틴을 통해 좌표계 내의 축에 할당된 모터 수를 확인합니다. 따라서 좌표계용의 룩 어헤드 버퍼를 생성하기 전에 좌표계를 정의해야 합니다.

이 명령이 전송되었을 때 이 좌표계에 정의된 룩 어헤드 버퍼가 이미 존재하는 경우에 명령은 에러로 거부됩니다. 저장하는 세그먼트 수의 변경 또는 좌표계 내의 모터 수 변경으로 좌표계의 룩 어헤드 버퍼를 재정의하는 경우에는 최초로 **delete lookahead** 명령 또는 **delete all lookahead** 명령을 사용하여 기존의 버퍼를 삭제해야 합니다.

명령의 상수값은 좌표계의 각 모터의 룩 어헤드 버퍼에 저장할 수 있는 동작 세그먼트 수를 지정합니다. 각 세그먼트는 프로그램 속도이며 [Coord[x].SegMoveTime] 밀리초의 시간이 필요합니다. 룩 어헤드 버퍼는 속도 및 가속도 리미트의 위반을 방지하기 위해 해당 세그먼트 시간을 연장하는 경우가 있습니다. Status 요소 [Coord[x].LHSize]는 이 값으로 설정 됩니다. 버퍼는 적어도 1,024 개의 세그먼트를 저장해야 합니다.

저장 설정 요소 [Coord[x].LHDistance]는 운전 중에 좌표계가 실제로 어느 정도의 세그먼트를 룩 어헤드하는지 지정합니다. 버퍼는 산출된 룩 어헤드 세그먼트 모두를 저장할 수 있는 크기여야 합니다. 따라서 이 상수에는 아무리 적어도 [Coord[x].LHDistance]로 지정된 같은 크기가 필요함을 의미합니다. 백업(리트레이스) 기능이 필요한 경우에 버퍼는 명령된 룩 어헤드 크기에 명령된 백업 크기를 더한 값에 대응하는 충분한 공간이 필요합니다.

확실한 가속 제한을 위해서는 룩 어헤드 알고리즘은 적어도 가장 나쁜 상태에서의 정지 거리를 선행하여 체크해야 합니다. 이는 시간으로 표시되며 최고 속도를 최대 가속도로 나누어서 구합니다. 최고 속도는 `[Motor[x].MaxSpeed]` 또는 `[Coord[x].MaxFeedrate]` (대부분 더 낮아짐)에서 산출할 수 있습니다. 좌표계 모터의 모든 최고 속도 대 가속도 비율 중에서 가장 높은 비율로 룩 어헤드하는 세그먼트의 최저수를 구하는 식은 다음과 같습니다.

$$AheadSegments = \frac{4}{3} * \frac{V_{max}}{A_{max}} * \frac{1}{2 * SegMoveTime}$$

명령된 백업 크기에 대응하여 저장에 필요한 세그먼트의 수를 구하는 식은 다음과 같습니다.

$$BackSegments = \frac{BackupDist (motor\_units)}{V_{max} (motor\_units / msec) * SegMoveTime (msec / seg)}$$

버퍼의 크기는 적어도 해당 2 개의 값의 합계와 같은 수의 세그먼트를 저장할 수 있어야 합니다. 버퍼 내의 각 세그먼트는 룩 어헤드 버퍼 공간에 $(40 + 16*N)$ 바이트가 필요합니다. 여기서 N 은 좌표계의 축(위치 결정축 또는 주축)에 할당된 모터의 수입니다. 예를 들어 좌표계의 축에 5 대의 모터가 할당된 경우에 각 세그먼트에는 120 바이트가 필요하므로 세그먼트 수 2,000 인 버퍼는 240,000 바이트의 메모리가 필요합니다. 룩 어헤드 버퍼 공간의 기본값은 16MB(16,777,216 바이트)입니다. 이는 IDE 의 프로젝트 매니저를 사용하여 변경할 수 있습니다. 이 버퍼 공간은 모든 좌표계의 룩 어헤드 세그먼트 버퍼, 공구 직경 보정 동작 블록 버퍼 및 목표위치 버퍼용에 사용됩니다.

define rotary

기능: 로터리 모션 프로그램의 버퍼를 정의함

범위: 좌표계 고유

구문: **define rotary {상수} [{상수}][{상수}]**

여기에서

- 최초의 {상수}는 2,048 이상의 32 비트 양의 정수이며 이 버퍼에 예약된 메모리의 바이트 수를 나타냅니다.
- 2 번째 옵션의 {상수}는 1,024 이상의 32 비트의 양의 정수이며 단일 프로그램행의 예비 처리를 위해 예약된 메모리의 바이트 수를 나타냅니다. 값이 지정되지 않은 경우에는 1,024의 값이 사용됩니다. 이 값은 최초의 상수로 지정된 버퍼 크기의 1/2 이하여야 합니다.
- 3 번째 옵션의 {상수}는 양의 정수이며 프로그램이 서브루틴 또는 서브 프로그램을 호출하는 경우에 사용하는 로컬 변수 스택 오프셋의 크기를 지정합니다. 값이 지정되지 않은 경우에는 256의 값이 사용됩니다.

Power PMAC 은 **define rotary** 명령으로 주소 지정된 좌표계의 로터리 모션 프로그램의 버퍼를 생성하고 지정된 양의 메모리를 할당합니다. 로터리 버퍼에 따라 프로그램 실행 중에 모션 프로그램 명령행의 다운로드가 가능합니다.

상수의 목적지 또는 거리(**X10** 등)를 가진 단일 축 동작 명령에는 9 바이트의 메모리가 필요합니다. 최초의 상수로 지정되는 버퍼의 크기는 호스트 컴퓨터에서 다음 다운로드 이벤트 전에 계산이 오버플로하지 않도록 프로그램의 계산보다 훨씬 앞인 시점에서 다운로드할 수 있는 충분한 크기가 필요합니다. 그 크기는 2번째 상수로 지정되는 예비행의 처리 버퍼의 적어도 2 배, 모든 상황을 상정하여 2,048 이상이 필요합니다.

2번째 옵션의 상수로 지정되는 예비행의 처리 버퍼의 크기는 각 명령행이 처리된 형식으로 로터리 버퍼 자체에 놓일 때까지 보관하는 일시 기억 버퍼 크기를 결정합니다. 값이 지정되지 않은 경우에는 1,024 바이트가 예약됩니다. 단일 명령행 처리에 1,024 바이트 이상을 필요로 하는 어플리케이션은 거의 없습니다.

3번째 옵션의 상수로 지정되는 스택 오프셋의 크기는 로터리 프로그램 자체를 사용할 수 있는 로컬 변수의 최대수를 결정합니다. 로컬 변수는 로터리 프로그램이 호출하는 모든 서브 프로그램의 로컬 변수에 따라 재사용되는 경우는 없습니다. 값이 지정되지 않은 경우에 256의 스택 오프셋이 사용됩니다.

로터리 프로그램의 버퍼는 RAM 스크립트 프로그램용 메모리 공간 전체의 메모리를 사용합니다. 이 공간은 기본값으로 16 메가바이트이지만 그 크기는 IDE 프로젝트 매니저를 사용하여 변경할 수 있습니다.

delete all lookahead

기능: 모든 룩 어헤드 버퍼를 삭제함

범위: 글로벌

구문: **delete all lookahead**

Power PMAC은 **delete all lookahead** 명령으로 모든 좌표계에 정의된 룩 어헤드 버퍼를 삭제하여 메모리를 다른 용도로 해방합니다.

delete all rotary

기능: 모든 로터리 모션 버퍼를 삭제함

범위: 글로벌

구문: **delete all rotary**

Power PMAC은 **delete all rotary** 명령으로 모든 좌표계의 정의된 로터리 모션의 버퍼를 삭제하고 메모리를 다른 용도로 해방합니다.

이 명령이 전송된 경우에 모든 로터리 모션 프로그램의 버퍼는 클로즈해야 합니다. **h**(유지) 명령, **q**(종료) 명령, **s**(스텝) 명령 사용하거나 프로그램 실행이 버퍼 종료점에 도달한 경우와 마찬가지로 모든 로터리 모션 프로그램의 실행은 일시 중단해야 하는 것이 아니라 완전히 정지해야 합니다. 이는 **a**(Abort) 명령을 사용한 경우와 마찬가지로입니다.

delete lookahead

기능: 지정된 좌표계의 룩 어헤드 버퍼를 삭제함

범위: 좌표계 고유

구문: **delete lookahead**

Power PMAC 은 **delete lookahead** 명령으로 지정된 좌표계의 정의된 룩 어헤드 버퍼를 삭제하고 메모리를 다른 용도로 해방합니다.

좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다.

delete rotary

기능: 지정된 좌표계의 로터리 모션 프로그램의 버퍼를 삭제함

범위: 좌표계 고유

구문: **delete rotary**

Power PMAC 은 **delete rotary** 명령으로 지정된 좌표계의 정의된 로터리 모션 프로그램의 버퍼를 삭제하고 메모리를 다른 용도로 해방합니다.

좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다.

이 명령이 전송된 경우에 지정된 로터리 모션 프로그램의 버퍼는 클로즈해야 합니다. **h**(유지) 명령, **q**(종료) 명령, **s**(스텝) 명령을 사용하거나 프로그램의 실행이 버퍼 종료점에 도달한 경우와 마찬가지로 로터리 모션 프로그램의 실행은 일시 중지하는 것이 아니라 완전히 정지해야 합니다. 이는 **a**(Abort) 명령을 사용한 경우와 마찬가지로입니다.

disable

기능: 좌표계의 모든 모터를 무효로 함

범위: 좌표계 고유

구문: **disable**

Power PMAC 은 **disable** 명령으로 지정한 좌표계 모터의 모든 제어를 무효로 합니다. 이는 기본적으로 좌표계의 모터 모두에 대한 모터 고유의 **k**(출력 정지) 명령에 상당하지만 모터 명령과는 달리 모션 프로그램을 실행하는 좌표계에 작용하여 처리 중인 모션 프로그램을 중단시킵니다.

좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다.

모터 제어 Disable 은 앰프 Enable 상태의 False(무효) 설정, 위치/속도 서보 루프의 오픈, 제로 서보 출력값의 강제로 구성됩니다. 이 상태는 클로즈드 루프 유효 상태 또는 오픈 루프 유효 상태 중 하나에서 직접 이행 할 수 있습니다.

즉시 출력 정지 명령은 모터의 긴급 정지용입니다. 자동 브레이크 제어를 이용하여 모터 Disable 을 계획하는 경우에는 브레이크가 완전히 연동할 때까지 시간이 필요하기 때문에 동일하게 지연된 **ddisable** 명령으로 실행해야 합니다. 자동 브레이크 제어가 장착되어 있지 않은 모터는 어느 명령을 사용해도 문제 없습니다.

disable 명령은 모든 무효 상태의 모터에는 영향을 미치지 않습니다.

disable bgcplc

기능: 지정된 백그라운드 C PLC 프로그램을 무효로 함

범위: 글로벌

구문: **disable bgcplc {목록}**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수 또는 시작 상수에서 종료 상수 사이를 2 개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 모든 상수는 0~31 까지의 범위이고 무효로 할 백그라운드 C PLC 프로그램을 지정합니다.

Power PMAC 은 **disable bgcplc** 명령으로 후속 스캔 시작을 금지하고 지정된 백그라운드 C PLC 프로그램 실행을 정지합니다. 이 명령이 프로그램 도중에 실행을 중단(**sleep** 명령 등을 사용)한 경우에도 프로그램의 선두에서 실행을 재개할 수 있습니다.

명령으로 지정된 C PLC 프로그램이 Power PMAC 에 존재하지 않는 경우에 에러가 통지되지 않습니다. 명령은 명령으로 지정된 다른 C PLC 프로그램에 작용합니다.

예/

disable bgcplc 0	// Disable execution of CPLC 0
disable bgcplc 2,4,6	// Disable execution of CPLCs 2, 4, and 6
disable bgcplc 7..10	// Disable execution of CPLCs 7, 8, 9, and 10
disable bgcplc 11,13..16,20	// Disable execution of CPLCs 11, 13, 14, 15, 16, and 20

disable plc

기능: 지정된 스크립트 PLC 프로그램을 무효로 함

범위: 글로벌

구문: **disable plc {목록}**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수 또는 시작 상수에서 종료 상수 사이를 2 개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 모든 상수는 0~31 까지의 범위에서 무효로 하는 스크립트 PLC 프로그램을 지정합니다.

Power PMAC 은 **disable plc** 명령으로 후속 스캔 시작을 금지하고 지정된 스크립트 PLC 프로그램의 실행을 정지합니다. 이 명령이 프로그램 도중에 실행을 중단(프로그램이 [while] 루프 내에서 프리즈 등)한 경우에도 프로그램 선두에서 실행을 재개할 수 있습니다.

pause plc 명령은 마찬가지로 PLC 프로그램의 실행을 정지하기 위해 사용하지만 이 명령은 프로그램을 프로그램 선두 이외의 정지한 곳에서 재개할 수 있도록 합니다.

명령으로 지정된 PLC 프로그램이 Power PMAC 에 존재하지 않는 경우에 에러가 통지되지 않습니다. 명령은 명령으로 지정된 다른 PLC 프로그램에 작용합니다.

예

```
disable plc 1           // Disable execution of PLC 1
disable plc 2,4,6       // Disable execution of PLCs 2, 4, and 6
disable plc 7..10       // Disable execution of PLCs 7, 8, 9, and 10
disable plc 11,13..16,20 // Disable execution of PLCs 11, 13, 14, 15, 16, and 20
```

disable rticplc

기능: 포어그라운드 C PLC 프로그램을 무효로 함

범위: 글로벌

구문: **disable rticplc**

Power PMAC 은 **disable rticplc** 명령으로 후속 스캔 시작을 금지하고 실시간 인터럽트로 실행하는 포어그라운드 C PLC 프로그램 실행을 정지합니다. 이 명령이 프로그램 도중에 실행을 중단(**sleep** 명령 등을 사용)한 경우에도 프로그램의 선두에서 실행을 재개할 수 있습니다.

포어그라운드 C PLC 프로그램이 Power PMAC 에 존재하지 않는 경우에 에러가 통지되지 않습니다.

dkill

기능: 모터 출력 정지 지연

범위: 모터 고유

구문: **dkill**

Power PMAC 은 **dkill** 명령으로 지정된 모터의 서보 출력을 출력 정지로 합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다. **#*dkill** 명령은 Power PMAC 의 모든 모터를 출력 정지로 합니다. 좌표계의 **ddisable** 명령은 좌표계의 모터 모두를 지연된 출력 정지로 한다는 점에 주의하십시오.

모터를 출력 정지하면 서보 루프가 오픈되고 서보 출력값이 강제로 제로가 됩니다(단, 출력 바이어스 항은 적용됩니다). 다음으로 앰프 **Enable** 신호가 강제로 무효 상태가 됩니다. Power PMAC 은 앰프 폴트 시나 편차 이상 시에는 자동으로 모터를 출력 정지합니다.

[Motor[x].pBrakeOut]을 제로 이외의 값(브레이크 출력 레지스터의 주소)으로 설정하면 모터의 자동 브레이크 제어 기능이 유효한 경우에 브레이크 출력은 **dkill** 명령으로 즉시 연동하도록 명령됩니다. 게다가 모터가 클로즈드 루프, 제로 속도 상태인 경우에 모터의 실제 출력 정지는 [Motor[x].BrakeOnDelay] 밀리초만큼 지연되어 브레이크 시스템이 완전히 연동될 시간을 제공합니다.

이 지연 **Disable** 명령은 브레이크 장착 모터의 **Disable** 을 계획하는 경우에 브레이크가 완전히 연동하기까지 시간을 확보하는 용도입니다. 모터의 긴급한 출력 정지에는 마찬가지로 즉시 **k** 명령을 사용해야 합니다.

현재 모션 프로그램을 실행 중인 좌표계 내의 모터인 경우에 Power PMAC 은 에러로 인해 모터 고유의 **dkill** 명령을 거부합니다. 단, 2011 년 7 월에 출시된 V1.4 의 **#*dkill** 명령은 모션 프로그램을 실행 중인 좌표계의 모터도 포함하여 모든 모터에 작용하여 처리 중인 모션 프로그램을 중단합니다.

모터의 클로즈드 루프를 **Enable** 하는 제어는 **j/** 명령으로 재개할 수 있습니다. 좌표계 내 모터의 모든 클로즈드 루프를 **Enable** 하는 제어는 **enable** 명령으로 재개할 수 있습니다. 모터의 오픈 루프를 유효화하는 제어는 **out{상수}** 명령으로 재개할 수 있습니다.

#xdkill(온라인 지연 Disable) 명령 및 **dkillx**(버퍼 프로그램 지연 출력 정지) 명령의 구문이 다르므로 주의하십시오.

ecat alias

기능: EtherCAT 슬레이브 장치에 별칭 주소를 설정함

구문: **ecat alias {상수} {상수} [{상수}]**

여기에서

- 최초의 **{상수}**는 별칭 주소를 부여하는 슬레이브 장치의 절대 위치값을 지정합니다.
- 2 번째 **{상수}**는 이 슬레이브 장치에 부여하는 별칭 주소 번호를 지정합니다.
- 마지막 **{상수}**는 옵션으로 이 슬레이브 장치가 배치된 EtherCAT 마스터의 번호이기도 한 EtherCAT 네트워크 인덱스(0~7)를 지정합니다. 값이 지정되지 않은 경우에 장치는 [ECAT[0]]으로 간주됩니다.

Power PMAC 은 **ecat alias** 명령으로 EtherCAT 네트워크 상의 슬레이브 장치에 별칭 주소를 할당합니다. 네트워크 배선을 변경할 가능성이 있는 경우에 네트워크 상의 슬레이브 장치에 별칭 주소를 설정할 것을 적극 권장합니다. 그렇게 하면 장치 네트워크 상의 물리적 위치에 관계없이 장치 소프트웨어 주소를 지정할 수 있으며 네트워크 배선을 재편성한 경우에도 구성된 드라이브로 통신을 계속할 수 있습니다.

장치가 소프트웨어의 별칭 주소 설정에 대응하지 않는 경우에는 이 명령은 사용할 수 없습니다. EtherCAT 슬레이브 장치 중에는 온보드 스위치를 통한 하드웨어의 별칭 주소 설정에만 대응하는 장치도 있습니다. 이와 같은 장치에서는 이 명령은 효과가 없습니다.

Power PMAC 에서는 [ECAT[*i*].Enable]이 1 로 설정되고 네트워크에서 사이클릭 통신이 유효한 경우에 이 명령은 실행할 수 없습니다.

별칭의 값을 설정한 후에 Power PMAC 은 네트워크에서 자동으로 **ecat slaves** 명령을 실행하고 네트워크 상의 슬레이브 구성 결과를 표시합니다.

예

```
ecat alias 7 16
0 VID=$00000002 PC=$044D2C52 0:0 PREOP + EK1101 EtherCAT-Koppler (2A E-Bus)
1 VID=$00000002 PC=$10103052 0:1 PREOP + EL4112-0010 2K.Ana.Ausgang +/-1t
2 VID=$00000002 PC=$10243052 0:2 PREOP + EL4132 2Ch.Ana.Ausgang +/-10V, t
3 VID=$00000002 PC=$101A3052 0:3 PREOP + EL4122 2K.Ana.Ausgang 4-20mA, 1t
4 VID=$00000002 PC=$07E83052 0:4 PREOP + EL2024 4K.Dig.Ausgang 24V, 2A
5 VID=$00000002 PC=$0C503052 0:5 PREOP + EL3152 2K.Ana.Eingang 4-20mA
6 VID=$00000002 PC=$05F12C52 0:6 PREOP + EK1521 1-Port EtherCAT-Abzweig (F)
7 VID=$00000002 PC=$05DD2C52 16:0 PREOP + EK1501 EtherCAT-Koppler (2A E-Bus)
8 VID=$00000002 PC=$138A3052 16:1 PREOP + EL5002 2K.SSI Encoder
9 VID=$00000002 PC=$04562C52 16:2 PREOP + EK1110 EtherCAT-Verlängerung
```

ecat assign

기능: 슬레이브 장치를 구조체 [ECAT[i].Slave[j]]에 할당함

구문: **ecat assign** [{상수}] [{상수}]

최초의 {상수}는 구조체에 할당하는 슬레이브 장치의 절대 위치값을 지정합니다. 값이 지정되지 않은 경우에는 지정된 네트워크 상의 모든 슬레이브가 [ECAT[0]]으로 통지됩니다. -1 의 값을 지정하면 모든 슬레이브가 최대 8 개의 EtherCAT 네트워크 전체에 할당됩니다.

2 번째 옵션의 {상수}는 슬레이브 장치를 할당하는 EtherCAT 네트워크의 인덱스(0~7)(EtherCAT 마스터 번호 등)를 지정합니다. 값이 지정되지 않은 경우에는 슬레이브는 [ECAT[0]]에 할당됩니다.

Power PMAC 은 **ecat assign** 명령으로 다음에 이용 가능한 네트워크의 구조체 [ECAT[i].Slave[j]]에 지정된 슬레이브 장치를 할당합니다. 음의 값이 아닌 상수값이 명령 다음에 있는 경우에 그 상수는 구조체에 할당하는 네트워크 상의 단일 장치의 절대 위치를 지정합니다. 2 번째 상수가 지정된 경우에 이 상수는 할당된 네트워크의 인덱스(0~7)를 지정합니다. 지정되지 않은 경우에는 인덱스는 0 으로 간주됩니다.

명령 뒤에 상수가 없는 경우에 EtherCAT 네트워크 0 인 슬레이브 장치 모두가 구조체 [ECAT[0].Slave[j]]에 할당되어 이미 할당된 장치도 재할당됩니다. 명령 뒤에 -1 의 값이 있는 경우에는 모든 EtherCAT 네트워크의 슬레이브 장치 모두가 구조체 [ECAT[i].Slave[j]]에 할당되어 이미 할당된 장치도 재할당됩니다.

구조체 [ECAT[i].Slave[j]]에 할당된 각 장치에 대해 **ecat slaves** 명령으로 취득한 장치의 정보를 사용하여 다음의 요소값이 자동으로 설정됩니다.

- ECAT[i].Slave[j].VendorID
- ECAT[i].Slave[j].ProductCode
- ECAT[i].Slave[j].Enable
- ECAT[i].Slave[j].Position
- ECAT[i].Slave[j].Alias

단일 장치를 할당한 경우에 서브 구조체 [Slave[j]]의 인덱스값 j는 명령이 전송되었을 때의 [ECAT[i].SlaveCount]의 값과 같습니다. 구조체 요소값이 설정된 후에 [ECAT[i].SlaveCount]의 값은 자동으로 1 개만 증가됩니다.

네트워크 상의 모든 슬레이브 장치가 할당되면 서브 구조체 [Slave[j]]의 인덱스값 *j*의 0 은 슬레이브 장치 수에서 1 을 뺀 값이 되며 [Ecat[j].SlaveCount]의 값이 자동으로 슬레이브 장치 수로 설정됩니다.

일반적으로는 모든 장치가 슬레이브의 서브 구조체에 할당되도록 네트워크의 초기 설정에서는 파라미터가 없는 **ecat assign** 명령이 사용됩니다. 파라미터가 있는 **ecat assign** 명령은 보통 이미 구성된 네트워크에서 새로운 장치를 네트워크에 추가해야 하는 경우에 사용됩니다. 이 명령은 EtherCAT 에서 사이클릭 조작이 실시된 경우([ECAT[j].Enable] = 1)에는 실행할 수 없습니다.

ecat config

기능: 자동으로 구조체 및 서브 구조체 [Ecat[j].Slave[j]]를 구성함

구문: **ecat config**

Power PMAC 은 **ecat config** 명령으로 모든 슬레이브 장치를 그 네트워크의 구조체 [Ecat[j].Slave[j]]에 할당하여 이미 할당된 장치도 재할당합니다.

구조체 [Ecat[j].Slave[j]]에 할당된 각 장치에 대해 **ecat slaves** 명령을 사용한 경우와 마찬가지로 취득한 장치의 정보를 사용하여 다음의 요소값이 자동으로 설정됩니다.

- ECAT[j].Slave[j].VendorID
- ECAT[j].Slave[j].ProductCode
- ECAT[j].Slave[j].Enable
- ECAT[j].Slave[j].Position
- ECAT[j].Slave[j].Alias

또한 Power PMAC 은 이 명령으로 다음의 서브 구조체를 할당합니다.

- ECAT[j].Slave[j].PDO[k].
- ECAT[j].Slave[j].PDOMapping[k].
- ECAT[j].Slave[j].SyncManager[k].

서브 구조체 [Slave[j]]의 인덱스값 *j*의 0 은 슬레이브 장치 수에서 1 을 뺀 값이 되며 [ECAT[j].SlaveCount]의 값은 자동으로 슬레이브 장치 수로 설정됩니다.

ecat slaves

기능: EtherCAT 네트워크의 슬레이브 장치를 표시함

구문: **ecat slaves [{상수}]**

여기에서

- 옵션의 {상수}는 슬레이브 장치를 통지하는 EtherCAT 마스터 번호이기도 한 EtherCAT 네트워크의 인덱스(0~7)를 지정합니다. 값이 지정되지 않은 경우에는 슬레이브는 ECAT[0]으로 통지됩니다. -1 의 값을 지정하면 모든 슬레이브가 최대 8 개의 EtherCAT 네트워크 전체에 통지됩니다.

Power PMAC 은 **ecat slaves** 명령으로 지정된 EtherCAT 네트워크에 현재 배치된 슬레이브 장치의 정보를 통지합니다. 1 슬레이브의 정보가 1 행 텍스트 형식으로 통지됩니다. 각 행에는 다음의 정보가 포함됩니다.

- **Absolute position(절대 위치):** 네트워크 상의 슬레이브의 물리적 위치순이 정수로 통지됩니다.
- **Vendor ID(벤더 ID):** 슬레이브 장치의 제조자 ID 번호가 VID=\$xxxxxxx 형식으로 통지됩니다.
- **Product code(제품 코드):** 슬레이브 장치로 사용되는 제품의 코드 번호가 PC=\$xxxxxxx 형식으로 통지됩니다.
- **Alias(별칭):** 슬레이브 장치의 별칭 번호가 정수값으로 통지됩니다.
- **Relative position(상대 위치):** 슬레이브 장치의 상대 위치값이 콜론에 이어지는 정수로 통지됩니다.
- **State(상태):** 슬레이브 장치의 운전 상태가 짧은 텍스트로 통지됩니다.
- **Device description(장치의 설명):** 슬레이브 장치로 사용되는 제품의 설명이 텍스트 형식으로 통지됩니다.

예

ecat slaves 0

Absolute Position

Vendor ID

Product Code

Alias

Relative Position

State

Device Description

```

0 VID=$000000002 PC=$044D2C52 0:0 PREOP + EK1101 EtherCAT-Koppler (2A E-Bus)
1 VID=$000000002 PC=$10103052 0:1 PREOP + EL4112-0010 2K.Ana.Ausgang +/-1t
2 VID=$000000002 PC=$10243052 0:2 PREOP + EL4132 2Ch.Ana.Ausgang +/-10V, t
3 VID=$000000002 PC=$101A3052 0:3 PREOP + EL4122 2K.Ana.Ausgang 4-20mA, 1t
4 VID=$000000002 PC=$07E83052 0:4 PREOP + EL2024 4K.Dig.Ausgang 24V, 2A
5 VID=$000000002 PC=$0C503052 0:5 PREOP + EL3152 2K.Ana.Eingang 4-20mA
6 VID=$000000002 PC=$05F12C52 0:6 PREOP + EK1521 1-Port EtherCAT-Abzweig (F)
7 VID=$000000002 PC=$05DD2C52 1:0 PREOP + EK1501 EtherCAT-Koppler (2A E-Bus)
8 VID=$000000002 PC=$138A3052 1:1 PREOP + EL5002 2K.SSI Encoder
9 VID=$000000002 PC=$04562C52 1:2 PREOP + EK1110 EtherCAT-Verlungerung

```

echo

기능: 명령 에코 모드를 통지함

범위: 통신 스레드 공유

구문: **echo**

Power PMAC 은 **echo** 명령으로 통신 스레드에 대해 에코 제어 파라미터의 현재값을 반환합니다. 이 값은 임의의 쿼리 명령에 응답 서식, 주로 쿼리 대상의 변수 이름을 응답에 기재할지 값만 기재할지를 결정합니다. 반환값이 무엇을 의미하는지를 이해하기 위해 다음에 나타난 **echo{상수}** 명령의 설명을 참조하십시오.

echo{상수}

기능: 명령 에코 모드를 설정함

범위: 통신 스레드 고유

구문: **echo{상수}**

여기에서

- **{상수}**는 0~63 까지 범위의 정수로 그 6 개의 각 비트(0~5)가 임의의 쿼리 명령에 대한 응답의 에코 백을 제어합니다.

echo{상수} 명령에 따라 Power PMAC 은 통신 스레드에 대한 에코 제어 파라미터의 값을 지정된 상수의 값으로 설정합니다. 이 값은 임의의 쿼리 명령에 응답 서식, 주로 쿼리 대상의 변수 이름을 응답에 기재할지 값만 기재할지를 결정합니다. 각 통신 스레드는 독자적인 에코 제어 파라미터를 가진다는 점에 주의하십시오.

에코 제어 파라미터는 5 개의 독립된 제어 비트로 구성됩니다.

bit 0(값 1)은 쿼리 대상의 요소값 응답에 데이터 구조체 요소의 이름을 기재할지 여부를 제어합니다. 이 비트를 0 으로 설정하면 이름이 기재됩니다. 예를 들어 쿼리[Motor[1].JogSpeed]의 응답은 [Motor[1].JogSpeed=200]과 같습니다. 이 비트를 1 로 설정하면 이름은 기재되지 않습니다. 예를 들어 쿼리[Motor[1].JogSpeed]의 응답은 [200]과 같습니다.

bit 1(값 2)은 쿼리 대상의 변수값 응답에 번호 지정된 변수의 이름을 기재할지 여부를 제어합니다. 이 비트를 0 으로 설정하면 이름이 기재됩니다. 예를 들어 쿼리[P1]의 응답은 [P1=10]과 같습니다. 이 비트를 1 로 설정하면 이름은 기재되지 않습니다. 예를 들어 쿼리[P1]의 응답은 [10]과 같습니다.

bit 2(값 4)은 쿼리 대상의 'I' 변수 또는 'M' 변수 *[definition]*의 응답에 'I' 변수 또는 'M' 변수의 이름을 기재할지 여부를 제어 합니다. 이 비트를 0 으로 설정하면 이름이 기재됩니다. 예를 들어 쿼리 [M1->]의 응답은 [M1->Sys.ServoCount]와 같습니다. 이 비트를 1 로 설정하면 이름은 기재되지 않습니다. 예를 들어 쿼리 [M1]의 응답은 [Sys.ServoCount]와 같습니다.

bit 3(값 8)은 그 값이 주소 또는 비트 논리값인 데이터 구조체 요소가 이러한 값을 16 진법(기수 16) 또는 10 진법(기수 10) 중 하나로 통지를 반환할지 여부를 제어합니다. 이 비트를 0 으로 설정하면 해당 값은 16 진법으로 반환됩니다. 예를 들어 쿼리 [Gate1[4].Chan[0].Dac[0].a]의 응답은 [\$d5700008]과 같습니다. 이 비트를 1 로 설정하면 해당 값은 10 진법으로 반환됩니다. 예를 들어 쿼리 [Gate1[4].Chan[0].Dac[0].a]의 응답은 [3580887048]과 같습니다.

bit 4(값 16)는 **backup xxx.status** 명령의 응답에 기재되는 Status 비트의 텍스트 설명이 열거 형식(False 또는 True) 또는 정수값(0 또는 1) 중 하나인지 여부를 제어합니다. 이 비트를 0 으로 설정하면 정수값으로 통지됩니다. 이 비트를 1 로 설정하면 열거 형식으로 통지됩니다(IDE 의 [Status] 창은 이 비트를 자동으로 1 로 설정하여 열거 형식을 받습니다).

bit 5(값 32)는 기동하지 않은 모터([Motor[x].ServoCtrl] = 0) 및 모터가 할당되지 않은 좌표계를 **backup** 및 **save** 명령을 사용하여 완전히 백업할지 여부를 제어합니다. 이 비트를 0으로 설정하면 이러한 모터 및 좌표계는 완전히 백업되지 않아 저장 공간과 시간이 절약됩니다. 이 비트를 1로 설정하면 이러한 모터 및 좌표계는 완전히 백업되어 개정 비교를 관리하기가 보다 쉬워집니다. 이 제어 비트는 2012년 2/4 분기에 출시된 펌웨어 V1.5의 새 기능이며 이 비트를 기본값인 0으로 설정하면 개정 간의 호환성을 유지할 수 있습니다.

bit 6(값 64)은 여러 개의 차분 요소 또는 폴 요소로 백업할지 여부를 단일 폴 워드 요소의 일부로서의 중복 차이 저장 설정 요소로 제어합니다. 이 비트를 0으로 설정하면 폴 백업으로 보다 빠르고 더 콤팩트하게 백업됩니다. 이 비트를 1로 설정하면 차이 백업으로 읽혀지고 필요한 부분에 대한 정보의 내용만 보다 빠르게 이해할 수 있습니다. 이 제어 비트는 2014년 4/4 분기에 출시된 펌웨어 V2.0의 새 기능이며 이 비트를 기본값인 0으로 설정하면 개정 간의 호환성을 유지할 수 있습니다.

에코 제어 파라미터의 전원 셋업 시 기본값은 0입니다. 따라서 쿼리 응답에서는 모든 이름이 에코 백됩니다(IDE의 [Watch] 창에서는 자동으로 7로 설정됩니다). 에코 제어 파라미터의 현재값은 **echo** 쿼리 명령으로 확인할 수 있습니다.

enable

기능: 좌표계의 모터 모두를 유효로 함

범위: 좌표계 고유

구문: **enable**

enable 명령을 사용하면 Power PMAC은 지정된 좌표계로 정의되고 현재 Disable된(출력 정지된) 모든 모터의 클로즈드 루프 서보 제어를 유효로 할 수 있습니다. 오픈 루프 또는 클로즈드 루프에 관계없이 이 명령은 좌표계에서 이미 유효한 모터에는 영향을 주지 않습니다.

좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다.

서보 제어의 유효화는 앰프 Enable 상태의 True(유효) 설정, 위치/속도 서보 루프의 클로즈로 구성됩니다. 모터는 출력 정지된 상태 또는 오픈 루프 유효 상태 어느 쪽에서든 최종적으로 유효화되고 클로즈드 루프, 제로 명령 속도 상태가 됩니다.

V1.4(2011년 7월 시점) 이전의 펌웨어 버전에서는 **enable** 명령은 유효한 오픈 루프 상태인 모터의 루프를 클로즈하는 명령이었다는 점에 주의하십시오.

enable bgcplc

기능: 지정된 백그라운드 C PLC 프로그램을 유효로 함

범위: 글로벌

구문: **enable bgcplc {목록}**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수 또는 시작 상수에서 종료 상수 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 모든 상수는 0~31까지의 범위이며 유효로 할 백그라운드 C PLC 프로그램을 지정합니다.

Power PMAC 은 **enable bgcplc** 명령으로 보통의 우선 순위와 타이밍에 따라 지정된 백그라운드 C PLC 프로그램 실행을 허가합니다. 이 명령이 프로그램 도중에 실행을 중단(**sleep** 명령 등을 사용)한 경우에도 프로그램의 선두에서 실행을 재개할 수 있습니다.

명령으로 지정된 C PLC 프로그램이 Power PMAC 에 존재하지 않는 경우에 에러가 통지되지 않습니다. 명령은 명령으로 지정된 다른 C PLC 프로그램에 작용합니다.

예

```
enable bgcplc 0           // Enable execution of CPLC 0
enable bgcplc 2,4,6       // Enable execution of CPLCs 2, 4, and 6
enable bgcplc 7..10       // Enable execution of CPLCs 7, 8, 9, and 10
enable bgcplc 11,13..16,20 // Enable execution of CPLCs 11, 13, 14, 15, 16, and 20
```

enable plc

기능: 지정된 스크립트 PLC 프로그램을 유효로 함

범위: 글로벌

구문: **enable plc {목록}**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수 또는 시작 상수에서 종료 상수 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 모든 상수는 0~31까지의 범위이며 유효로 하는 스크립트 PLC 프로그램을 지정합니다.

Power PMAC 은 **enable plc** 명령으로 보통의 우선 순위와 타이밍에 따라 지정된 스크립트 PLC 프로그램의 실행을 허가합니다. 프로그램 도중에 실행이 중단(**pause plc** 명령 등의 사용)한 경우 또는 (이미 유효로 설정된 경우에)실행 직후의 스캔이 프로그램 종료점에서 정지하지 않은 경우에도 프로그램 선두에서 실행을 시작합니다.

resume plc 명령은 마찬가지로 PLC 프로그램 실행을 재개하기 위해 사용하지만 이 명령은 프로그램을 프로그램 선두 이외의 정지한 곳에서 재개할 수 있도록 합니다.

명령으로 지정된 PLC 프로그램이 Power PMAC 에 존재하지 않는 경우에 에러가 통지되지 않습니다. 명령은 명령으로 지정된 다른 PLC 프로그램에 작용합니다.

예

```
enable plc 1           // Enable execution of PLC 1
enable plc 2,4,6       // Enable execution of PLCs 2, 4, and 6
enable plc 7..10       // Enable execution of PLCs 7, 8, 9, and 10
enable plc 11,13..16,20 // Enable execution of PLCs 11, 13, 14, 15, 16, and 20
```

enable rticplc

기능: 포어그라운드 C PLC 프로그램을 유효로 함

범위: 글로벌

구문: **enable rticplc**

Power PMAC 은 **enable rticplc** 명령으로 보통의 우선 순위와 타이밍에 따라 실시간 인터럽트로 실행되는 포어그라운드 C PLC 프로그램 실행을 허가합니다. 이 명령이 프로그램 도중에 실행을 중단(**sleep** 명령 등을 사용)한 경우에도 프로그램의 선두에서 실행을 재개할 수 있습니다.

포어그라운드 C PLC 프로그램이 Power PMAC 에 존재하지 않는 경우에 에러가 통지되지 않습니다.

f

기능: 위치 편차값을 통지함

범위: 모터 또는 좌표계 고유

구문: **f**

Power PMAC 은 **f** 명령으로 지정된 모터 또는 좌표계의 위치 편차의 현재값을 통지합니다. 여러 통지도 할 수 있습니다. 이 값은 명령 위치와 현재 위치 간의 순간적인 차이입니다. 모터 목록 또는 좌표계 목록이 없는 경우에는 현재 주소 지정된 모터의 위치 편차값을 통지합니다. 모터의 목록이 있는 경우에는 목록 내 모든 모터의 위치 편차값을 통지합니다. 좌표계의 목록이 있는 경우에는 목록 내 모든 좌표계의 활성인 모든 축의 위치 편차값을 통지합니다.

여러 모터 목록 또는 여러 좌표계 목록을 지정해도 후속 모터 고유의 명령 또는 좌표계 고유의 명령으로 모달에 주소 지정된 모터 또는 좌표계는 변경되지 않습니다.

위치 편차를 통지하는 경우에 편차는 정의된 모터 단위로 표시됩니다. 축의 위치 편차를 통지하는 경우에 편차는 스케일링된 축 단위로 표시됩니다. 각 축에 통지된 값은 축의 이름 (1 개 또는 2 개의 영문자)앞에 설명합니다.

그 좌표계의 축에 모터가 할당되지 않은 좌표계에서 통지가 요구되면 Power PMAC 은 문자열 'No Motors'를 반환합니다. 좌표계의 축 정의 셋팅 또는 포워드 키네마틱스 서브루틴으로 축 데이터를 적절하게 계산할 수 없는 경우에 Power PMAC 은 문자열 'No Solution'을 반환합니다.

예

```
f // Query following error of presently addressed motor
-23 // Power PMAC response in motor units

#1..4f // Query following errors of Motors 1 - 4
33 -22 4 173 // Power PMAC response in motor units

&1f // Query following errors of axes in C.S. 1
X0.033 Y-0.022 Z0.004 C0.173 // Power PMAC response in user axis units
```

free

기능: 버퍼의 여유 메모리를 통지함

범위: 글로벌

구문: **free**

Power PMAC 은 **free** 명령으로 RAM 의 여러 사용자 메모리 버퍼에서 이용 가능한 (미사용의)메모리 바이트 수를 통지합니다. 관련된 **size** 명령은 이러한 각 메모리 버퍼에 대해 사용 완료 또는 미사용에 관계없이 메모리의 합계 바이트 수를 반환합니다.

예

free	// Query free buffer memory
Program Buffer = 16740722	// Power PMAC response
User Buffer = 1048576	
Table Buffer = 1002305	
Lookahead Buffer = 16777216	

g

기능: 현재 실행 중인 동작 축의 남은 이동량을 통지함

범위: 좌표계 고유

구문: **g**

Power PMAC 은 **g** 명령으로 현재 실행 중인 동작 축의 남은 이동량을 통지합니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계의 남은 이동량을 통지합니다. 좌표계 목록이 있는 경우에 목록 내 모든 좌표계의 남은 이동량을 통지합니다.

축의 남은 이동량은 축의 목표 위치에서 현재 실행 중인 동작으로 그 축의 근접 명령 위치를 빼고 계산됩니다. X, Y, Z 축에 공구 직경 보정이 유효한 경우에 이 계산에 공구 직경 보정에 따른 목표 위치 오프셋 값이 사용됩니다. 축의 현재 명령 위치는 좌표계 정의에 따라 처리되어(축 정의 구문을 반전하거나 포워드 키네마틱스 루틴의 사용 둘 중 하나) 대응할 모터의 명령 위치 및 유효한 매트릭스 변환에서 계산됩니다. 이 차는 양이며 부호가 있는 값으로 통지됩니다.

저장 설정 요소 [Coord[x].TPSize]를 0 이상의 동작 계산 시와 동작 실행 시 사이의 모든 동작 위치를 저장하기 위한 충분한 값으로 설정하고 목표 위치의 버퍼를 유효로 해야 합니다. 좌표계의 정의된 모든 축의 남은 이동량이 통지됩니다.

남은 이동량은 축의 이름을 나타내는 문자 다음에 수치가 붙은 텍스트 형식으로 통지됩니다. 거리는 동작이 계산된 시점에서 유효한 축 변환을 사용하여 계산되어 통지됩니다.

저장 설정 요소 [Coord[x].Ndisplay]를 1 로 설정하면 Status 요소 [Coord[x].Nsync]의 현재값이 응답 선두에서 문자 'N' 다음에 통지됩니다. 이 값은 레이블에 이어지는 프로그램에서 동작 실행 시작 시 직후의 동기 라인 레이블값에 자동으로 설정됩니다. 따라서 레이블이 사용되었을 때의 특정한 동작을 간단하게 식별할 수 있습니다.

[Coord[x].TPSize]가 0 으로 설정된 경우에 이 명령이 전송되면 Power PMAC 은 에러 메시지 'No targets defined'를 반환합니다. [Coord[x].TPSize]가 0 이상으로 설정되고 그 값이 계산과 실행 간의 모든 동작 버퍼에 충분하지 않은 경우에는 잘못된 값이 통지될 가능성이 있습니다.

g(온라인) 명령은 **dtogread**(버퍼 프로그램) 명령과 같은 계산을 실행합니다. 또한 텍스트 문자열을 반환하기 위해 버퍼 프로그램 명령과 마찬가지로 축의 값을 'D' 변수에 넣습니다.

예/

```
Coord[1].Ndisplay = 0          // Do not report N label
&1g                          // Request distance to go
A5.72 C0 X-0.007 Y3.227 Z3.753 // Power PMAC responds w/o N label

Coord[1].Ndisplay = 1          // Report N label
&1g                          // Request distance to go
N430 A5.72 C0 X-0.007 Y3.227 Z3.753 // Power PMAC responds w/ N label
```

h

기능: 피드 홀드를 실행함

범위: 좌표계 고유

구문: **h, hold**

Power PMAC 은 **h** 명령을 사용하면 좌표계 타임 베이스를 제로로 하여 지정된 좌표계의 모션 프로그램의 실행을 일시 정지하고 경로에 따른 시작을 즉시 감속시킵니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계의 실행을 유지 정지합니다. 좌표계 목록이 있는 경우에 목록 내의 모든 좌표계 실행을 유지 정지합니다.

피드 홀 모드 중 모션 프로그램의 실행은 일시 정지하지만 이론적으로는 아직 실행 중인 상태입니다. 일시 정지 후에도 프로그램의 실행을 재개하지 않는 경우에는 **a** 명령을 사용하여 프로그램의 실행을 완전히 Abort 해야 합니다.

감속 및 그 후의 재가속이 [Coord[x].TimeBaseSlew]가 아니라 [Coord[x].FeedHoldSlew]로 제어되는 경우를 제외하고 결과적으로 **h** 명령은 **%0** 명령과 아주 유사합니다. 또한 **%100** 대신에 **R** 또는 **S** 명령을 사용하여 실행을 재개할 수 있습니다. 또한 **h** 명령은 **%0** 명령을 실행할 수 없는 외부 타임 베이스에서도 기능합니다.

일반적으로 **h** 명령으로는 모션이 프로그램 포인트에서 정지하지 않습니다. **r** 또는 **s** 명령을 사용하면 경로에 따른 전체 속도에서 실행이 재개됩니다. 전체 타임 베이스값(내부 설정 또는 외부 설정에 관계없이)까지의 변화는 [Coord[x].FeedHoldSlew]로 설정되는 속도가 됩니다. 전체 타임 베이스값에 도달하면 [Coord[x].TimeBaseSlew]는 임의의 타임 베이스로 변경할 수 있습니다.

여러 좌표계 목록을 지정하더라도 후속 좌표계 고유의 명령에 대해 모달에 주소 지정된 좌표계는 변경되지 않습니다.

이 명령의 생략형(**h**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**hold**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012년 3/4 분기에 출시된 V1.5의 새 기능)가 1로 설정된 경우에는 이 명령의 비생략형(**hold**)을 사용해야 합니다. 생략형(**h**)은 올바르지 않은 명령이며 에러 20으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

동등한 버퍼 프로그램의 다이렉트 명령은 **hold**입니다.

예

h	// Hold program execution in addressed C.S.
&2h	// Address C.S.2 and hold program execution there
&1, 3, 5h	// Hold program execution in C.S. 1, 3, and 5
&*h	// Hold program execution in all C.S.

hm

기능: 원점 서치 동작을 시작함

범위: 모터 고유

구문: **hm, home**

Power PMAC은 **hm** 명령으로 지정된 모터의 원점 서치 동작을 시작합니다. 모터 목록이 없는 경우에 현재 주소 지정된 모터의 원점 서치 동작을 시작합니다. 모터 목록이 있는 경우에 목록 내 모든 모터의 원점 서치 동작을 시작합니다.

여러 개의 모터 목록을 지정해도 후속 모터 고유의 명령에 대해 모달에 주소 지정된 모터는 변경되지 않습니다.

임의의 모터의 원점 서치 동작 프로파일 특성은 모터의 데이터 구조체 설정 요소 [Motor[x].HomeVel], [Motor[x].JogTa], [Motor[x].JogTs] 및 [Motor[x].HomeOffset]으로 제어됩니다.

home(온라인) 명령은 원점 서치 루틴을 시작만 합니다. Power PMAC에는 서치 완료 또는 서치 동작의 성공 여부를 자동으로 표시하는 기능은 없습니다. 서치 동작 완료 및 서치의 성공을 판단하려면 모터의 상태를 모니터링해야 합니다.

한편, 원점 서치 동작이 모션 프로그램 내의 버퍼 명령, **home1, 2** 등에서 명령된 경우에는 모션 프로그램은 자체적으로 시퀀스 알고리즘의 일부로 완료 Status를 끊임없이 감시합니다. 원점 서치 동작이 PLC 프로그램 내의 버퍼 명령에서 명령된 경우에 온라인 명령인 경우와 마찬가지로 서치 동작 완료 및 서치의 성공을 판단하려면 모터의 상태를 모니터링해야 합니다.

이 명령의 생략형(**hm**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**home**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012년 3/4 분기에 출시된 V1.5의 새 기능)가 1로 설정된 경우에는 이 명령의 비생략형(**home**)을 사용해야 합니다. 생략형(**hm**)은 올바르지 않은 명령이며 에러 20으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 원점 서치 동작 명령(**#xhm**) 및 버퍼 프로그램 원점 서치 동작 명령(**homex**)의 구문이 다르므로 주의하십시오.

예

hm	// Start homing-search move on the addressed motor
#1hm	// Address Motor 1, start homing-search move on it
#2, 4, 6hm	// Command Motors 2, 4, and 6 to start homing move
#*hm	// Command all active motors to start homing move

hmz

기능: 제로 동작 원점 복귀를 실행함

범위: 모터 고유

구문: **hmz, homez**

Power PMAC 은 **hmz** 명령으로 지정된 모터의 제로 동작 원점 복귀를 시작합니다. 모터 목록이 없는 경우에 현재 주소 지정된 모터의 제로 동작 원점 복귀를 시작합니다. 모터 목록이 있는 경우에 목록 내 모든 모터의 제로 동작 원점 복귀를 시작합니다.

여러 개의 모터 목록을 지정해도 후속 모터 고유의 명령에 대해 모달에 주소 지정된 모터는 변경되지 않습니다.

[Motor[x].pAbsPos]를 기본값 0 으로 설정하면 **hmz** 명령에 따라 Power PMAC 은 단순히 현재 명령 위치로부터의 동작 없이 원점(모터 제로)으로 재정의합니다.

단, [Motor[x].pAbsPos]를 제로 이외의 값으로 설정하면 Power PMAC 은 모터의 현재 절대 위치값에 따라 지정되는 주소를 가지는 레지스터를 읽습니다. 이 주소에서의 위치 데이터 읽기는 [Motor[x].AbsPosFormat], [Motor[x].AbsPosSf] 및 [Motor[x].HomeOffset] 설정에 따라 처리됩니다.

이 명령의 생략형(**hmz**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**homez**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012 년 3/4 분기에 출시된 V1.5 의 새 기능)가 1 로 설정된 경우에는 이 명령의 비생략형(**homez**)을 사용해야 합니다. 생략형(**hmz**)은 올바르지 않은 명령이며 에러 20 으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 제로 동작 원점 복귀 명령(**#xhmz**) 및 버퍼 프로그램 제로 동작 원점 복귀 명령(**homezx**)의 구문이 다르므로 주의하십시오.

예

hmz	// Perform zero-move homing on the addressed motor
#1hmz	// Address Motor 1, perform zero-move homing on it
#2, 4, 6hmz	// Command Motors 2, 4, and 6 to do zero-move homing
#*hmz	// Command all active motors to do zero-move homing

hold

기능: 피드 홀드를 실행함

범위: 좌표계 고유

구문: **h, hold**

hold 명령은 **h** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 좌표계 타임 베이스를 제로로 하여 지정된 좌표계의 모션 프로그램의 실행을 일시 정지하고 경로에 따른 시작을 즉시 감속시킵니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

home

기능: 원점 서치 동작을 시작함

범위: 모터 고유

구문: **hm, home**

home 명령은 **hm** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정한 모터의 원점 서치 동작을 시작합니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

homez

기능: 제로 동작 원점 복귀를 실행함

범위: 모터 고유

구문: **hmz, homez**

homez 명령은 **hmz** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정한 모터의 제로 동작 원점 복귀를 실행합니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

{데이터}

기능: 'I' 변수값을 통지함

범위: 글로벌

구문: **{데이터}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.

Power PMAC은 **{데이터}** 명령으로 지정된 'I' 변수의 현재값을 통지합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~65,535). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

echo0	// Include variable names in query responses
I1	// Query value of variable I1
I1=4	// Power PMAC response of the value of I1
I(P1)	// Query value of I-variable numbered by P1
I122=3	// Power PMAC response of the value of I122
I(200+P17)	// Query value of I-variable numbered by 200+P17
I220=-5.35	// Power PMAC response of the value of I220
echo2	// Do not include variable names in query responses
I1	// Query value of variable I1
4	// Power PMAC response of the value of I1

{데이터}={식}

기능: 값을 'I' 변수에 할당함

범위: 글로벌

구문: **{데이터}={식}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.
- **{식}**은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC은 **{데이터}={식}** 명령으로 지정된 'I' 변수를 부호의 오른쪽 식의 값으로 설정합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~65,535). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수 번호가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

명령이 Power PMAC에 송신되고 프로그램 버퍼가 오픈되면 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

예

I1=4	// Set variable I1 to 4
I(P1)=3	// Set I-variable numbered by P1 to 3
I(P1+P17)=5*sqrt(P100)	// Set I-var numbered by (P1+P17) to expression value

{데이터}->

기능: 'I' 변수의 정의를 통지함

범위: 글로벌

구문: **{데이터}->**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.

Power PMAC은 **{데이터}->** 명령으로 지정된 'I' 변수의 정의를 통지합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~65,535). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수 번호가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

각 'I' 변수는 Power PMAC의 데이터 구조체 요소에 정의됩니다. 'M' 변수와는 달리 'I' 변수 정의는 사용자가 변경할 수 없습니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 2(값 4)는 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0인 경우에 이름을 기재합니다. 비트가 1인 경우에 이름은 기재되지 않습니다.

예

```
echo0 // Include variable names in query responses
I122-> // Query definition of variable I122
I122->Motor[1].JogSpeed // Power PMAC response of the definition of I122

I(P1)-> // Query definition of I-variable numbered by P1 (=222)
I222->Motor[2].JogSpeed // Power PMAC response of the definition of I222

echo4 // Do not include variable names in query responses
I122-> // Query definition of variable I122
Motor[1].JogSpeed // Power PMAC response of the definition of I122
```

{변수 목록}

기능: 목록의 'I' 변수값을 통지함

범위: 글로벌

구문: **{변수 목록}[:{상수}]**

여기에서

- **{변수 목록}**은 1개 이상의 변수 세트를 지정합니다. 목록은 다음 2개의 형식 중 1개를 취할 수 있습니다.

- o **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작되고 2 번째 정수값(최초보다 큰 번호여야 합니다)으로 번호를 할당된 변수로 종료하는 연속적인 번호의 변수 범위를 지정합니다.
- o **{상수}[,{상수}[,{상수}]**에서 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- 옵션의 **{상수}**는 응답 행마다 통지된 값의 수를 지정하는 양의 정수입니다. 여기서 값을 지정하지 않는 경우에 응답 행마다 하나의 값만 통지됩니다. 값을 지정하지 않고 사용할 수 있는 것은 목록이 연속된 수의 범위로 지정된 경우에 한합니다.

Power PMAC 은 **I{변수 목록}** 명령으로 목록 내 'I' 변수의 현재값을 통지합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC に송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

I100 I100=1	// Query value of var I100 (set quantity 1, spacing 1) // Power PMAC response of the value of I100
I120..I122 I120=-4 I121=3.5 I122=25	// Query value of I120, I121, I122 // Power PMAC response of the value of I120 // Power PMAC response of the value of I121 // Power PMAC response of the value of I122
I120,3 I120=-4 I121=3.5 I122=25	// Query value of 3 I-vars, starting at I120 (spacing 1) // Power PMAC response of the value of I120 // Power PMAC response of the value of I121 // Power PMAC response of the value of I122
I122,3,100 I122=25 I222=25 I322=25	// Query value of 3 I-vars, starting at I120, spacing 100 // Power PMAC response of the value of I122 // Power PMAC response of the value of I222 // Power PMAC response of the value of I322
I3100..3109:5 I3100=0,0,0,0,0 I3105=0,0,0,0,0	// Query value of I3100 - I3109, 5 values per line // Power PMAC response of the value of I3100 - I3104 // Power PMAC response of the value of I3105 - I3109

{변수 목록}={식}

기능: 목록의 'I' 변수값을 설정함

범위: 글로벌

구문: ***{변수 목록}={식}***

여기에서

- ***{변수 목록}***은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.
 - o ***{상수}..{상수}***에서는 최초의 정수값으로 번호를 매긴 변수로 시작되고 2 번째 정수값(최초보다 큰 번호여야 합니다)으로 번호를 할당된 변수로 종료하는 연속적인 번호의 변수 범위를 지정합니다.
 - o ***{상수}[,{상수}[,{상수}]***에서 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- ***{식}***은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC 은 ***{변수 목록}={식}*** 명령으로 이 통신 스레드에 로컬 목록 내 'I' 변수값을 부호의 오른쪽 식의 값으로 설정합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC に송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

예

I100=1	// Set value of I100 (set quantity 1, spacing 1) to 1
I122..123=P1+5	// Set value of I122, I123 to (P1+5)
I122,2=17.5	// Set val of 2 I-vars, starting at I122 (spacing 1) to 17.5
I122,3,100=P25+7	// Set value of 3 I-vars, starting at I122, spacing 100
	// (I122, I222, I322) to (P25+7)

{변수 목록}->

기능: 목록의 'I' 변수 정의를 통지함

범위: 글로벌

구문: **{변수 목록}->**

여기에서

- **{변수 목록}**은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.
 - o **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작되고 2 번째 정수값(최초보다 큰 번호여야 합니다)으로 번호를 할당된 변수로 종료하는 연속적인 번호의 변수 범위를 지정합니다.
 - o **{상수},{상수},{상수}**에서 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).

Power PMAC 은 **{변수 목록}->** 명령으로 목록 내 'I' 변수의 정의를 통지합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC 에 송신되고 프로그램 버퍼가 오픈되면 명령은 에러로 거부됩니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 2(값 4)는 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

```
I122-> // Query definition of variable I122
I122->Motor[1].JogSpeed // Power PMAC response of the definition of I122

I120..I122-> // Query definitions of I120, I121, I122
I120->Motor[1].JogTa // Power PMAC response of the definition of I120
I121->Motor[1].JogTs // Power PMAC response of the definition of I121
I122->Motor[1].JogSpeed // Power PMAC response of the definition of I122

I120,3-> // Query defs of 3 I-vars starting with I120 (spacing 1)
I120->Motor[1].JogTa // Power PMAC response of the definition of I120
I121->Motor[1].JogTs // Power PMAC response of the definition of I121
I122->Motor[1].JogSpeed // Power PMAC response of the definition of I122

I122,3,100-> // Query defs of 3 I-vars starting with I120, spacing 100
I122->Motor[1].JogSpeed // Power PMAC response of the definition of I122
I222->Motor[2].JogSpeed // Power PMAC response of the definition of I222
I322->Motor[3].JogSpeed // Power PMAC response of the definition of I322
```

j+

기능: 양의 방향에 무한 JOG 이송을 함

범위: 모터 고유

구문: **j+, jog+**

j+ 명령을 사용하면 Power PMAC 은 지정된 모터에 대해 양의 방향으로 무한 JOG 이송을 시작합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

명령된 JOG 이송은 모터의 데이터 구조체 설정 요소 [Motor[x].JogSpeed], [Motor[x].JogTa] 및 [Motor[x].JogTs]의 값으로 명령 시에 강제로 제어됩니다. 모터가 현재 다른 JOG 이송 또는 원점 서치 동작을 실행 중이거나 모터가 오픈 루프 유효화 또는 출력 정지 상태인 경우에도 이 명령은 전송할 수 있습니다.

모터에 소프트웨어 양의 리미트가 유효화된 경우([Motor[x].MaxPos] > [Motor[x].MinPos]), 이 명령은 양의 리미트까지 JOG 이송하는 명령으로 처리되기 때문에 다른 명령이 전송되지 않은 경우에도 소프트웨어의 양의 리미트에서 자동으로 정지합니다(리미트 통과 시에 정지가 시작하는 것은 아닙니다).

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령의 생략형(**j+**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**jog+**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012 년 3/4 분기에 출시된 V1.5 의 새 기능)가 1 로 설정된 경우에는 이 명령의 비생략형(**jog+**)을 사용해야 합니다.. 생략형(**j+**)은 올바르지 않은 명령이며 에러 20 으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 JOG 명령(**#xj+**) 및 버퍼 프로그램 JOG 명령(**jog+x**)의 구문이 다르므로 주의하십시오.

j-

기능: 음의 방향에 무한 JOG 이송을 함

범위: 모터 고유

구문: **j-, jog-**

j- 명령을 사용하면 Power PMAC 은 지정된 모터에 대해 음의 방향으로 무한 JOG 이송을 시작합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

명령된 JOG 이송은 모터의 데이터 구조체 설정 요소 [Motor[x].JogSpeed], [Motor[x].JogTa] 및 [Motor[x].JogTs]의 값으로 명령 시에 강제로 제어됩니다. 모터가 현재 다른 JOG 이송 또는 원점 서치 동작을 실행 중이거나 모터가 오픈 루프 유효화 또는 출력 정지 상태인 경우에도 이 명령은 전송할 수 있습니다.

모터에 소프트웨어 음의 리미트가 유효화된 경우([Motor[x].MinPos] < [Motor[x].MaxPos]), 이 명령은 음의 리미트까지 JOG 이송하는 명령으로 처리되기 때문에 다른 명령이 전송되지 않은 경우에도 소프트웨어 음의 리미트에서 자동으로 정지합니다(리미트 통과 시에 정지가 시작하는 것은 아닙니다).

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령의 생략형(**j-**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**jog-**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012 년 3/4 분기에 출시된 V1.5 의 새 기능)가 1 로 설정된 경우에는 이 명령의 비생략형(**jog-**)을 사용해야 합니다. 생략형(**j-**)은 올바르지 않은 명령이며 에러 20 으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 JOG 명령(**#xj-**) 및 버퍼 프로그램 JOG 명령(**jog-x**)의 구문이 다르므로 주의하십시오.

j/

기능: JOG 정지 또는 루프를 클로즈함

범위: 모터 고유

구문: **j/, jog/**

j/ 명령을 사용하면 명령 시에 모터가 클로즈드 루프 상태 또는 오픈 루프 상태에 관계없이 Power PMAC 은 지정한 모터를 제어된 클로즈드 루프로 정지시킵니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

명령된 정지는 모터의 데이터 구조체 설정 요소 [Motor[x].JogTa] 및 [Motor[x].JogTs]의 값으로 명령 시에 강제로 제어됩니다. 명령이 클로즈드 루프 동작(JOG 이송 또는 원점 서치) 중에 전송되면 모터의 명령 속도가 감속 기울기의 시작값으로 사용됩니다. 모터가 오픈 루프 상태일 때에 명령이 전송되면 모터의 현재 속도가 시작값으로 사용됩니다.

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령의 생략형(**j/**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**jog/**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012 년 3/4 분기에 출시된 V1.5 의 새 기능)가 1 로 설정된 경우에는 이 명령의 비생략형(**jog/**)을 사용해야 합니다. 생략형(**j/**)은 올바르지 않은 명령이며 에러 20 으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 JOG 명령(**#xj/**) 및 버퍼 프로그램 JOG 명령(**jog/x**)의 구문이 다르므로 주의하십시오.

j=

기능: 사전에 JOG 위치로 JOG 이송함

범위: 모터 또는 좌표계 고유

구문: **j=, jog=**

j= 명령을 사용하면 Power PMAC 은 지정된 모터에 대해 최신 프로그램 위치로 JOG 이송을 시작합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내 각 좌표계의 모든 모터가 대상이 됩니다.

명령된 JOG 이송은 모터의 데이터 구조체 설정 요소 [Motor[x].JogSpeed], [Motor[x].JogTa] 및 [Motor[x].JogTs]의 값으로 명령 시에 강제로 제어됩니다. 모터가 현재 다른 JOG 이송 또는 원점 서치 동작을 실행 중이거나 모터가 오픈 루프 유효화 또는 출력 정지 상태인 경우에도 이 명령은 전송할 수 있습니다.

j= 명령은 모션 프로그램이 일시 정지된 경우나 1 대 이상의 모터가 일시 정지 위치에서 JOG 이송된 경우에 사용할 수 있습니다. 이 명령은 모터를 일시 정지 위치로 되돌리고 프로그램을 재개하기 위해 사용할 수 있습니다.

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령의 생략형(**j=**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**jog=**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012 년 3/4 분기에 출시된 V1.5 의 새 기능)가 1 로 설정된 경우에는 이 명령의 비생략형(**jog=**)을 사용해야 합니다. 생략형(**j=**)은 올바르지 않은 명령이며 에러 20 으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 JOG 명령(**#xj=**) 및 버퍼 프로그램 JOG 명령(**jogretx**)의 구문이 다르므로 주의하십시오.

j={상수}

기능: 지정된 위치로 JOG 이송함

범위: 모터 고유

구문: **j={상수}, jog={상수}**

여기에서

- **{상수}**는 목적지 위치를 모터 단위로 지정하는 부동 소수점 값입니다.

j={상수} 명령을 사용하면 Power PMAC 은 지정된 모터에 지정한 위치(모터 제로 위치에 모터 단위로)로 JOG 이송을 시작합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

명령된 JOG 이송은 모터의 데이터 구조체 설정 요소 [Motor[x].JogSpeed], [Motor[x].JogTa] 및 [Motor[x].JogTs]의 값으로 명령 시에 강제로 제어됩니다. 모터가 현재 다른 JOG 이송 또는 원점 서치 동작을 실행 중이거나 모터가 오픈 루프 유효화 또는 출력 정지 상태인 경우에도 이 명령은 전송할 수 있습니다.

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령의 생략형(**j={상수}**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**jog={상수}**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012 년 3/4 분기에 출시된 V1.5 의 새 기능)가 1 로 설정된 경우에는 이 명령의 비생략형(**jog={상수}**)을 사용해야 합니다. 생략형(**j={상수}**)은 올바르지 않은 명령이며 에러 20 으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 JOG 명령(**#xj={상수}**) 및 버퍼 프로그램 JOG 명령(**jogx={데/이터}**)의 구문이 다르므로 주의하십시오.

예

j=0	// Jog addressed motor to zero position
#1j=10000	// Jog motor 1 to position of 10,000 units
#2j=-347.28	// Jog motor 2 to position of -347.28 units
#3,4,5j=76921	// Jog motors 3 – 5 to position of 76,921 units

j={상수}

기능: 지정된 위치로 JOG 이송하고 그 위치를 사전에 JOG 위치로 함

범위: 모터 고유

구문: **j={상수}, jog={상수}**

여기에서

- **{상수}**는 목적지 위치를 모터 단위로 지정하는 부동 소수점 값입니다.

j={상수} 명령을 사용하면 Power PMAC 은 지정된 모터에 대해 지정된 위치(모터 제로 위치에 대해 모터 단위로)에 대한 JOG 이송을 시작합니다. 이 목적지 위치는 사전에 JOG 위치도 되므로 후속 **J=**명령은 모터를 그 위치로 되돌립니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

명령된 JOG 이송은 모터의 데이터 구조체 설정 요소 [Motor[x].JogSpeed], [Motor[x].JogTa] 및 [Motor[x].JogTs]의 값으로 명령 시에 강제로 제어됩니다. 모터가 현재 다른 JOG 이송 또는 원점 서치 동작을 실행 중이거나 모터가 오픈 루프 유효화 또는 출력 정지 상태인 경우에도 이 명령은 전송할 수 있습니다.

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령의 생략형(**j={상수}**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**jog={상수}**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용됩니다.

저장 설정 요소 [Sys.NoShortCmds](2012 년 3/4 분기에 출시된 V1.5 의 새 기능)가 1 로 설정된 경우에는 이 명령의 비생략형(**jog={상수}**)을 사용해야 합니다. 생략형(**j={상수}**)은 올바르지 않은 명령이며 에러 20 으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 JOG 명령(**#xj={상수}**) 및 버퍼 프로그램 JOG 명령(**jogretx={데이터}**)의 구문이 다르므로 주의하십시오.

예

j=0	// Jog addressed motor to zero position
#1j=10000	// Jog motor 1 to position of 10,000 units
#2j=-347.28	// Jog motor 2 to position of -347.28 units
#3,4,5j=76921	// Jog motors 3 – 5 to position of 76,921 units

j=*

기능: 변수로 지정된 위치로 JOG 이송함

범위: 모터 고유

구문: **j=*, jog=***

j=* 명령을 사용하면 Power PMAC 은 지정된 모터에 모터 데이터 구조체 요소 [Motor[x].ProgJogPos](모터 제로 위치에 모터 단위로)에 설정된 위치로 JOG 이송을 시작합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

명령된 JOG 이송은 모터의 데이터 구조체 설정 요소 [Motor[x].JogSpeed], [Motor[x].JogTa] 및 [Motor[x].JogTs]의 값으로 명령 시에 강제로 제어됩니다. 모터가 현재 다른 JOG 이송 또는 원점 서치 동작을 실행 중이거나 모터가 오픈 루프 유효화 또는 출력 정지 상태인 경우에도 이 명령은 전송할 수 있습니다.

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령의 생략형(**j=***)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**jog=***)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용됩니다.

저장 설정 요소 [Sys.NoShortCmds](2012 년 3/4 분기에 출시된 V1.5 의 새 기능)가 1 로 설정된 경우에는 이 명령의 비생략형(**jog=***)을 사용해야 합니다. 생략형(**j=***)은 올바르지 않은 명령이며 에러 20 으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 JOG 명령(**#xj=***) 및 버퍼 프로그램 JOG 명령(**jogx={데이터}**)의 구문이 다르므로 주의하십시오.

예

Motor[1].ProgJogPos=10000	// Set jog position value to 10,000 units
#1j=*	// Jog motor 1 to preset position value
Motor[2].ProgJogPos=-347.28	// Set jog position value to -347.28 units
#2j=*	// Jog motor 2 to preset position value
Motor[3].ProgJogPos=P1	// Set jog position value to value of P1
Motor[4].ProgJogPos=P1	// Set jog position value to value of P1
Motor[5].ProgJogPos=P1	// Set jog position value to value of P1
#3,4,5j=*	// Jog motors 3 – 5 to position of P1 units

j:{상수}

기능: 명령 위치에서 지정된 상대 거리를 JOG 이송함

범위: 모터 고유

구문: j:{상수}, jog:{상수}

여기에서

- {상수}는 부호가 있는 목적지까지의 거리를 모터 단위로 지정하는 부동 소수점 값입니다.

j:{상수} 명령을 사용하면 Power PMAC 은 지정된 모터에 지정된 거리(현재 모터가 명령된 위치에 모터 단위로)의 JOG 이송을 시작합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

이 명령을 현재 위치와 관련된 유사한 명령 j\{상수}와 비교해 봅니다. 일반적으로 j:{상수} 명령은 명령 시에 목적지가 위치 편차에 따라 다르지 않으므로 실용적이라 할 수 있습니다.

명령된 JOG 이송은 모터의 데이터 구조체 설정 요소 [Motor[x].JogSpeed], [Motor[x].JogTa] 및 [Motor[x].JogTs]의 값으로 명령 시에 강제로 제어됩니다. 모터가 현재 다른 JOG 이송 또는 원점 서치 동작을 실행 중이거나 모터가 오픈 루프 유효화 또는 출력 정지 상태인 경우에도 이 명령은 전송할 수 있습니다.

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령의 생략형(j:{상수})은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(jog:{상수})은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012년 3/4 분기에 출시된 V1.5의 새 기능)가 1로 설정된 경우에는 이 명령의 비생략형(jog:{상수})을 사용해야 합니다. 생략형(j:{상수})은 올바르지 않은 명령이며 에러 20으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 JOG 명령(#xj:{상수}) 및 버퍼 프로그램 JOG 명령(jogx:{대입/타})의 구문이 다르므로 주의하십시오.

예

j:100	// Jog addressed motor a distance of +100 units
#1j:5000	// Jog motor 1 a distance of +5,000 units
#2j:-279.43	// Jog motor 2 a distance of -279.43 units
#3,4,5j:49232	// Jog motors 3 – 5 a distance of 49,232 units

j:*

기능: 명령 위치에서 변수로 지정된 상대 거리를 JOG 이송함

범위: 모터 고유

구문: **j:*** , **jog:***

j:* 명령을 사용하면 Power PMAC 은 지정된 모터에 모터의 데이터 구조체 요소 [Motor[x].ProgJogPos](현재 모터의 명령된 위치에 모터 단위로)로 설정한(부호 있음) 거리에 대한 JOG 이송을 시작합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

이 명령을 현재 위치와 관련된 유사한 명령 **j^***와 비교해 봅니다. 일반적으로 **j:*** 명령은 명령 시에 목적지가 위치 편차에 따라 다르지 않으므로 실용적이라 할 수 있습니다.

명령된 JOG 이송은 모터의 데이터 구조체 설정 요소 [Motor[x].JogSpeed], [Motor[x].JogTa] 및 [Motor[x].JogTs]의 값으로 명령 시에 강제로 제어됩니다. 모터가 현재 다른 JOG 이송 또는 원점 서치 동작을 실행 중이거나 모터가 오픈 루프 유효화 또는 출력 정지 상태인 경우에도 이 명령은 전송할 수 있습니다.

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령의 생략형(**j:***)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**jog:***)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012 년 3/4 분기에 출시된 V1.5 의 새 기능)가 1 로 설정된 경우에는 이 명령의 비생략형(**jog:***)을 사용해야 합니다. 생략형(**j:***)은 올바르지 않은 명령이며 에러 20 으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 JOG 명령(**#xj:***) 및 버퍼 프로그램 JOG 명령(**jogx:{데이터}**)의 구문이 다르므로 주의하십시오.

예

Motor[1].ProgJogPos=5000	// Set jog distance value to 5,000 units
#1j:*	// Jog motor 1 the preset distance
Motor[2].ProgJogPos=-822.67	// Set jog position value to -822.67 units
#2j:*	// Jog motor 2 the preset distance
Motor[3].ProgJogPos=P1+500	// Set jog distance value to (P1+500)
Motor[4].ProgJogPos=P1+500	// Set jog distance value to (P1+500)
Motor[5].ProgJogPos=P1+500	// Set jog distance value to (P1+500)
#3,4,5j:*	// Jog motors 3 – 5 the preset distance

j{상수}

기능: 현재 위치에서 지정된 상대 거리를 JOG 이송함

범위: 모터 고유

구문: j{상수}, jog{상수}

여기에서

- {상수}는 부호가 있는 목적지까지의 거리를 모터 단위로 지정하는 부동 소수점 값입니다.

j{상수} 명령을 사용하면 Power PMAC 은 지정된 모터에 대해 지정된 거리(모터의 현재 위치에 대해 모터 단위로)의 JOG 이송을 시작합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

이 명령을 현재 명령된 위치와 관련된 유사한 명령 j:{상수}와 비교해 봅시다. 일반적으로 j:{상수} 명령은 명령 시에 목적지가 위치 편차에 따라 다르지 않으므로 실용적이라 할 수 있습니다.

명령된 JOG 이송은 모터의 데이터 구조체 설정 요소 [Motor[x].JogSpeed], [Motor[x].JogTa] 및 [Motor[x].JogTs]의 값으로 명령 시에 강제로 제어됩니다. 모터가 현재 다른 JOG 이송 또는 원점 서치 동작을 실행 중이거나 모터가 오픈 루프 유효화 또는 출력 정지 상태인 경우에도 이 명령은 전송할 수 있습니다.

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령의 생략형(j{상수})은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(jog{상수})은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012 년 3/4 분기에 출시된 V1.5 의 새 기능)가 1 로 설정된 경우에는 이 명령의 비생략형(jog{상수})을 사용해야 합니다. 합니다. 생략형(j{상수})은 올바르게 않은 명령이며 에러 20 으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 JOG 명령(#xj:{상수}) 및 버퍼 프로그램 JOG 명령(jogx:{테이타})의 구문이 다르므로 주의하십시오.

예

j^100	// Jog addressed motor a distance of +100 units
#1j^5000	// Jog motor 1 a distance of +5,000 units
#2j^-279.43	// Jog motor 2 a distance of -279.43 units
#3,4,5j^49232	// Jog motors 3 – 5 a distance of 49,232 units

j^*

기능: 현재 위치에서 변수로 지정된 상대 거리를 JOG 이송함

범위: 모터 고유

구문: **j^***, **jog^***

j^* 명령을 사용하면 Power PMAC 은 지정된 모터에 모터의 데이터 구조체 요소 [Motor[x].ProgJogPos](모터의 현재 위치에 모터 단위로)로 설정한(부호 있음) 거리에 대한 JOG 이송을 시작합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

이 명령을 현재 명령된 위치와 관련된 유사한 명령 **j:***와 비교해 봅시다. 일반적으로 **j:*** 명령은 명령 시에 목적지가 위치 편차에 따라 다르지 않으므로 실용적이라 할 수 있습니다.

명령된 JOG 이송은 모터의 데이터 구조체 설정 요소 [Motor[x].JogSpeed], [Motor[x].JogTa] 및 [Motor[x].JogTs]의 값으로 명령 시에 강제로 제어됩니다. 모터가 현재 다른 JOG 이송 또는 원점 서치 동작을 실행 중이거나 모터가 오픈 루프 유효화 또는 출력 정지 상태인 경우에도 이 명령은 전송할 수 있습니다.

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령의 생략형(**j^***)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**jog^***)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012년 3/4 분기에 출시된 V1.5의 새 기능)가 1로 설정된 경우에는 이 명령의 비생략형(**jog^***)을 사용해야 합니다. 생략형(**j^***)은 올바르지 않은 명령이며 에러 20으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 JOG 명령(**#xj^***) 및 버퍼 프로그램 JOG 명령(**jogx^{테이타}**)의 구문이 다르므로 주의하십시오.

예

Motor[1].ProgJogPos=5000	// Set jog distance value to 5,000 units
#1j^*	// Jog motor 1 the preset distance
Motor[2].ProgJogPos=-822.67	// Set jog position value to -822.67 units
#2j^*	// Jog motor 2 the preset distance
Motor[3].ProgJogPos=P1+500	// Set jog distance value to (P1+500)
Motor[4].ProgJogPos=P1+500	// Set jog distance value to (P1+500)
Motor[5].ProgJogPos=P1+500	// Set jog distance value to (P1+500)
#3,4,5j^*	// Jog motors 3 – 5 the preset distance

{JOG 명령}^{상수}

기능: 트리거까지의 JOG 이송 동작

범위: 모터 고유

구문: $j=\{상수\}$, $jog=\{상수\}$
 $j=\{상수\}^{\{상수\}}$, $jog=\{상수\}^{\{상수\}}$
 $j:\{상수\}^{\{상수\}}$, $jog:\{상수\}^{\{상수\}}$
 $j^{\{상수\}^{\{상수\}}}$, $jog^{\{상수\}^{\{상수\}}}$
 $j=*\{상수\}$, $jog=*\{상수\}$
 $j:*\{상수\}$, $jog:*\{상수\}$
 $j^{*\{상수\}}$, $jog^{*\{상수\}}$

여기에서

- 명령 마지막의 **{상수}**는 트리거 위치에서 트리거 후 동작이 명령된 목적지까지 (부호 있음)거리를 지정하는 부동 소수점 값입니다.

트리거까지의 JOG 이송 명령을 사용하면 Power PMAC 은 지정된 모터에 대해 명확하게 지정된 타입의 한정 JOG 이송을 시작합니다. 이 동작 중에 사전 정의된 트리거 조건이 발생하면 Power PMAC 은 자동으로 동작 궤적에 인터럽트하여 남은 부분을 트리거 위치로부터의 거리가 명령 마지막의 **{상수}**로 결정되는 목적지까지의 JOG 이송과 치환합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

명령된 JOG 이송은 모터의 데이터 구조체 설정 요소 [Motor[x].JogSpeed], [Motor[x].JogTa] 및 [Motor[x].JogTs]의 값으로 명령 시에 강제로 제어됩니다. 모터가 현재 다른 JOG 이송 또는 원점 서치 동작을 실행 중이거나 모터가 오픈 루프 유효화 또는 출력 정지 상태인 경우에도 이 명령은 전송할 수 있습니다.

트리거 조건은 모터의 설정 데이터 구조체 요소 [Motor[x].CaptureMode]로 설정합니다.

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령의 생략형(**j...**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**jog...**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012년 3/4 분기에 출시된 V1.5의 새 기능)가 1로 설정된 경우에는 이 명령의 비생략형(**jog...**)을 사용해야 합니다. 생략형(**j...**)은 올바르지 않은 명령이며 에러 20으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 JOG 명령(**#xj={상수}^*** 등) 및 버퍼 프로그램 JOG 명령(**jogx={데이타}^{\{데이타\}}**)의 구문이 다르므로 주의하십시오.



```

j=^0 // Jog addressed motor to pre-jog position if no trigger,
// return to trigger position if found
#1j=25000^1000 // Jog Motor 1 to 25,000 units if no trigger,
// return to trigger position minus 1,000 units if found
#2j:30^0.2 // Jog Motor 2 a distance of 30 units if no trigger,
// return to trigger position plus 0.2 units if found
#3j^-75^1.1 // Jog Motor 3 a distance of -75 units if no trigger,
// return to trigger position plus 1.1 units if found
Motor[4].ProgJogPos=-5000 // Set (no-trigger) jog position to -5,000 units
#4j=^25 // Jog Motor 4 to preset position if no trigger,
// return to trigger position plus 25 units if found
Motor[5].ProgJogPos=32.1 // Set (no-trigger) jog distance to +32.1 units
#5j=^-0.01 // Jog Motor 5 the preset distance if no trigger,
// return to trigger position minus 0.01 units if found
Motor[6].ProgJogPos=-7.9 // Set (no-trigger) jog distance to -7.9 units
#6j=^225 // Jog Motor 6 the preset distance if no trigger,
// return to trigger position plus 225 units if found

```

{JOG 명령}^*

기능: 트리거까지의 JOG 이송 동작 변수

범위: 모터 고유

구문:
j=^*, jog=^*
j={상수}^*, jog={상수}^*
j:{상수}^*, jog:{상수}^*
j^{상수}^*, jog^{상수}^*
j=^*, jog=^*
j:~^*, jog:~^*
j^~^*, jog^~^*

트리거까지의 JOG 이송 명령을 사용하면 Power PMAC 은 지정된 모터에 대해 지정된 타입의 한정 JOG 이송을 시작합니다. 이 동작 중에 사전에 정의된 트리거 조건이 발생하면 Power PMAC 은 자동으로 동작 궤적에 인터럽트하여 남은 부분을 트리거 위치에서의 거리가 트리거 시의 [Motor[x].JogOffset]의 값으로 결정되는 목적지까지의 JOG 이송과 치환합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

명령된 JOG 이송은 모터의 데이터 구조체 설정 요소 [Motor[x].JogSpeed], [Motor[x].JogTa] 및 [Motor[x].JogTs]의 값으로 명령 시에 강제로 제어됩니다. 모터가 현재 다른 JOG 이송 또는 원점 서치 동작을 실행 중이거나 모터가 오픈 루프 유효화 또는 출력 정지 상태인 경우에도 이 명령은 전송할 수 있습니다.

트리거 조건은 모터의 설정 데이터 구조체 요소 [Motor[x].CaptureMode]로 설정합니다.

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령의 생략형(j...)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(jog...)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012년 3/4 분기에 출시된 V1.5의 새 기능)가 1로 설정된 경우에는 이 명령의 비생략형(**jog...**)을 사용해야 합니다. 생략형(**j...**)은 올바르지 않은 명령이며 에러 20으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 JOG 명령(**#xj={상수}^*** 등) 및 버퍼 프로그램 JOG 명령(**jogx={데/이/타}^{데/이/타}**)의 구문이 다르므로 주의하십시오.

예

Motor[1].JogOffset=2*P1	// Set post-trigger offset to 2*P1
#1j=25000^*	// Jog Motor 1 to 25,000 units if no trigger, // return to trigger position plus offset if found
#2j:30^*	// Jog Motor 2 a distance of 30 units if no trigger, // return to trigger position plus offset if found
#3j^-75^*	// Jog Motor 3 a distance of -75 units if no trigger, // return to trigger position plus offset if found
Motor[4].ProgJogPos=-5000	// Set (no-trigger) jog position to -5,000 units
Motor[4].JogOffset=P5-4	// Set post-trigger offset to P5-4
#4j=*^*	// Jog Motor 4 to preset position if no trigger, // return to trigger position plus offset if found
Motor[5].ProgJogPos=32.1	// Set (no-trigger) jog distance to +32.1 units
Motor[5].JogOffset=5*P9	// Set post-trigger offset to 5*P9
#5j=*^*	// Jog Motor 5 the preset distance if no trigger, // return to trigger position plus offset if found
Motor[6].ProgJogPos=-7.9	// Set (no-trigger) jog distance to -7.9 units
Motor[6].JogOffset=P8	// Set post-trigger offset to P8
#6j=*^225	// Jog Motor 6 the preset distance if no trigger, // return to trigger position plus 225 units if found

jog

기능: 오픈 루프에서 무한 JOG 이송함

범위: 모터 고유

구문: **jog**

jog 명령을 사용하면 Power PMAC은 지정된 모터에 오픈 루프 동작의 현재 속도를 사용하여 오픈 루프 동작에서(**out{상수}** 명령에서) 이동하는 무한 JOG 이송을 시작합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

모터가 이미 JOG 이송을 실행 중인 경우에 **jog** 명령은 모터에 아무 영향도 주지 않습니다.

명령 시에 모션 프로그램을 실행 중인 좌표계에서 모터가 위치 결정축에 정의된 경우에는 Power PMAC은 이 명령을 에러로 거부합니다.

jog+

기능: 양의 방향에 무한 JOG 이송을 함

범위: 모터 고유

구문: **j+, jog+**

jog+ 명령은 **j+** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정된 모터에 대해 양의 방향으로 무한 JOG 이송을 시작합니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

jog-

기능: 음의 방향에 무한 JOG 이송을 함

범위: 모터 고유

구문: **j-, jog-**

jog- 명령은 **j-** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정된 모터에 대해 음의 방향으로 무한 JOG 이송을 시작합니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

jog/

기능: JOG 정지 또는 루프를 클로즈함

범위: 모터 고유

구문: **j/, jog/**

jog/ 명령은 **j/** 명령의 비생략형입니다. 이 명령으로 명령 시에 모터가 클로즈드 루프 상태 또는 오픈 루프 상태에 관계없이 Power PMAC 은 지정한 모터를 제어된 클로즈드 루프로 정지시킵니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

jog=

기능: 사전에 JOG 위치로 JOG 이송함

범위: 모터 또는 좌표계 고유

구문: **j=, jog=**

jog= 명령은 **j=** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정된 모터에 대해 최신 프로그램 위치에 대한 JOG 이송을 시작합니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

jog={상수}

기능: 지정된 위치로 JOG 이송함

범위: 모터 고유

구문: **j={상수}, jog={상수}**

여기에서

- **{상수}**는 목적지 위치를 모터 단위로 지정하는 부동 소수점 값입니다.

jog={상수} 명령은 **j={상수}** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정한 모터에 대해 지정된 위치(모터 제로 위치에 대해 모터 단위로)에 대한 JOG 이송을 시작합니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

jog=={상수}

기능: 지정된 위치로 JOG 이송하고 그 위치를 사전에 JOG 위치로 함

범위: 모터 고유

구문: **j=={상수}, jog=={상수}**

여기에서

- **{상수}**는 목적지 위치를 모터 단위로 지정하는 부동 소수점 값입니다.

jog=={상수} 명령은 **j=={상수}** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정한 모터에 대해 지정된 위치(모터 제로 위치에 대해 모터 단위로)에 대한 JOG 이송을 시작합니다. 이 목적지 위치는 사전에 JOG 위치도 되므로 후속 **J=**명령은 모터를 그 위치로 되돌립니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

jog=*

기능: 지정된 변수 위치로 JOG 이송함

범위: 모터 고유

구문: **j=*, jog=***

jog=* 명령은 **j=*** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정된 모터에 대해 모터의 데이터 구조체 요소 [Motor[x].ProgJogPos](모터 제로 위치에 모터 단위로)로 설정한 위치에 대한 JOG 이송을 시작합니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

jog:{상수}

기능: 명령 위치에서 지정된 상대 거리를 JOG 이송함

범위: 모터 고유

구문: **j:{상수}, jog:{상수}**

여기에서

- **{상수}**는 부호가 있는 목적지까지의 거리를 모터 단위로 지정하는 부동 소수점 값입니다.

jog:{상수} 명령은 **j:{상수}** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정한 모터에 대해 지정된 거리(현재 모터의 명령된 위치에 대해 모터 단위로)의 JOG 이송을 시작합니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

jog:*

기능: 명령 위치에서 변수로 지정된 상대 거리를 JOG 이송함

범위: 모터 고유

구문: **j:*, jog:***

jog:* 명령은 **j:*** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정한 모터에 대해 모터의 데이터 구조체 요소 [Motor[x].ProgJogPos](현재 모터의 명령된 위치에 대해 모터 단위로)로 설정한(부호 있음) 거리의 JOG 이송을 시작합니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

jog^{상수}

기능: 현재 위치에서 지정된 상대 거리를 JOG 이송함

범위: 모터 고유

구문: **j^{상수}, jog^{상수}**

여기에서

- **{상수}**는 부호가 있는 목적지까지의 거리를 모터 단위로 지정하는 부동 소수점 값입니다.

jog^{상수} 명령은 **j^{상수}** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정한 모터에 대해 지정된 거리(모터의 현재 위치에 모터 단위로)의 JOG 이송을 시작합니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

jog^{*}

기능: 현재 위치에서 변수로 지정된 상대 거리를 JOG 이송함

범위: 모터 고유

구문: **j^{*}, jog^{*}**

jog^{*} 명령은 **j^{*}** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정한 모터에(모터의 현재 위치에) 모터의 데이터 구조체 요소 [Motor[x].ProgJogPos]에서 설정한(부호 있음) 거리의 JOG 이송을 시작합니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

k

기능: 모터를 출력 정지로 함

범위: 모터 고유

구문: **k, kill**

Power PMAC 은 **k** 명령으로 지정된 모터의 서보를 출력 정지로 합니다. 모터 목록이 없는 경우에 현재 주소 지정된 모터의 출력을 정지합니다. 모터 목록이 있는 경우에 목록 내 모든 모터의 출력을 정지합니다. 좌표계의 **disable** 명령인 경우에는 좌표계 내의 모든 모터를 출력 정지한다는 점에 주의하십시오.

모터를 출력 정지하면 서보 루프가 오픈되고 서보 출력값이 강제로 제로가 됩니다(단, 출력 바이어스 항은 적용됩니다). 다음으로 앰프 Enable 신호가 강제로 무효 상태가 됩니다. Power PMAC 은 앰프 폴트 시나 편차 이상 시에는 자동으로 모터를 출력 정지합니다.

즉시 출력 정지 명령은 모터의 긴급 정지용입니다. 자동 브레이크 제어가 장착된 모터의 무효화를 계획한 경우에는 브레이크가 연동할 충분한 시간이 필요하므로 동일하게 지연된 **dkill** 명령으로 실행해야 합니다. 자동 브레이크 제어가 장착되어 있지 않은 모터는 어느 명령을 사용해도 문제 없습니다.

현재 모션 프로그램을 실행한 좌표계 내의 모터인 경우에 Power PMAC 은 에러로 인해 모터 고유의 **k** 명령을 거부합니다. 단, **#*k** 명령은 모션 프로그램을 실행하는 좌표계 내의 모터도 포함하여 모든 모터에 작용해 처리 중인 모션 프로그램을 중단합니다.

모터의 클로즈드 루프를 Enable 하는 제어는 **j/** 명령으로 재개할 수 있습니다. 좌표계 내 모든 모터의 클로즈드 루프를 Enable 하는 제어는 **enable** 명령으로 재개할 수 있습니다. 모터의 오픈 루프를 Enable 하는 제어는 **out{상수}** 명령으로 재개할 수 있습니다.

이 명령의 생략형(**k**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**kill**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012 년 3/4 분기에 출시된 V1.5 의 새 기능)가 1 로 설정된 경우에는 이 명령의 비생략형(**kill**)을 사용해야 합니다. 생략형(**k**)은 올바르지 않은 명령이며 에러 20 으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

온라인 출력 정지 명령(**#xk**) 및 버퍼 프로그램 출력 정지 명령(**killx**)의 구문이 다르므로 주의하십시오.

kill

기능: 모터를 출력 정지로 함

범위: 모터 고유

구문: **k, kill**

kill 명령은 **k** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정한 모터의 서보를 출력 정지로 합니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

L{데이터}

기능: 'L' 변수값을 통지함

범위: 통신 스레드 고유

구문: **L{데이터}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.

Power PMAC 은 **L{데이터}** 명령으로 통신 스레드가 지정된 로컬 'L' 변수의 현재값을 통지합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~8,191). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

각 통신 스레드에는 자신의 'L' 변수 세트가 있습니다. 또한 각 좌표계에는 모션 프로그램에서 사용하는 좌표계 자체의 세트가 있으며 각 PLC 자체에도 세트가 있습니다. 'L' 변수의 모든 세트는 각각 독립적입니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

echo0	// Include variable names in query responses
L1	// Query value of variable L1
L1=17.5	// Power PMAC response of the value of L1
L(P1)	// Query value of L-variable numbered by P1
L17=3	// Power PMAC response of the value of L17
L(P1+P17)	// Query value of L-variable numbered by P1+P17
L20=-5.35	// Power PMAC response of the value of L20
echo2	// Do not include variable names in query responses
L1	// Query value of variable L1
17.5	// Power PMAC response of the value of L1

L{데이터}={식}

기능: 값을 'L' 변수에 할당함

범위: 통신 스레드 고유

구문: **L{데이터}={식}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.

- **{식}**은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC은 **L{데이터}={식}** 명령으로 통신 스레드가 지정된 로컬 'L' 변수에 부호의 오른쪽 식의 값을 설정합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~8,191). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

명령이 Power PMAC에 송신되고 프로그램 버퍼가 오픈되면 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

각 통신 스레드에는 자신의 'L' 변수 세트가 있습니다. 또한 각 좌표계에는 모션 프로그램에서 사용하는 좌표계 자체의 세트가 있으며 각 PLC 자체에도 세트가 있습니다. 'L' 변수의 모든 세트는 각각 독립적입니다.

예

L1=17.5	// Set variable L1 to 17.5
L(P1)=3	// Set L-variable numbered by P1 to 3
L(P1+P17)=5*sqrt(P100)	// Set L-var numbered by (P1+P17) to expression value

L{변수 목록}

기능: 목록의 'L' 변수값을 통지함

범위: 통신 스레드 고유

구문: **L{변수 목록}[:{상수}]**

여기에서

- **{변수 목록}**은 1개 이상의 변수 세트를 지정합니다. 목록은 다음 2개의 형식 중 1개를 취할 수 있습니다.
 - **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작하여 2번째 정수값(최초의 정수값보다 큰 번호여야 합니다)으로 번호를 매긴 변수로 종료하는 연속된 번호의 변수 범위를 지정합니다.
 - **{상수},{상수},{상수}**에서는 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1이 됩니다), 옵션의 3번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1이 됩니다).
- 옵션인 **{상수}**는 응답 행마다 통지된 값의 수를 지정하는 양의 정수입니다. 여기서 값을 지정하지 않는 경우에 응답 행마다 하나의 값만 통지됩니다. 값을 지정하지 않고 사용할 수 있는 것은 목록이 연속된 수의 범위로 지정된 경우에 한합니다.

Power PMAC 은 **L{변수 목록}** 명령으로 통신 스레드가 지정된 목록의 로컬 'L' 변수의 현재값을 통지합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC に송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

각 통신 스레드에는 자신의 'L' 변수 세트가 있습니다. 또한 각 좌표계에는 모션 프로그램에서 사용하는 좌표계 자체의 세트가 있으며 각 PLC 자체에도 세트가 있습니다. 'L' 변수의 모든 세트는 각각 독립적입니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

L1 L1=17.5	// Query value of variable L1 (set quantity 1, spacing 1) // Power PMAC response of the value of L1
L100..102 L100=-4 L101=3.14159 L102=0	// Query value of L100, L101, L102 // Power PMAC response of the value of L100 // Power PMAC response of the value of L101 // Power PMAC response of the value of L102
L50,3 L50=-5.35 L51=7.12 L52=376452	// Query value of 3 L-vars, starting at L50 (spacing 1) // Power PMAC response of the value of L50 // Power PMAC response of the value of L51 // Power PMAC response of the value of L52
L50,3,3 L50=-5.35 L53=99 L56=-0.002	// Query value of 3 L-vars, starting at L50, spacing 3 // Power PMAC response of the value of L50 // Power PMAC response of the value of L53 // Power PMAC response of the value of L56
L1..10:5 L1=7,3,5,2,11 L6=8,33,4,9,-2	// Query value of L1 – L10, 5 values per response line // Power PMAC response of the value of L1 – L5 // Power PMAC response of the value of L6 – L10

L{변수 목록}={식}

기능: 목록의 'L' 변수값을 설정함

범위: 통신 스레드 고유

구문: **L{변수 목록}={식}**

여기에서

- {변수 목록}**은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.

- o **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작하여 2 번째 정수값(최초의 정수값보다 큰 번호여야 합니다)으로 번호를 매긴 변수로 종료하는 연속된 번호의 변수 범위를 지정합니다.
 - o **{상수}[,{상수}[,{상수}]**에서는 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- **{식}**은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC 은 **L{변수 목록}={식}** 명령으로 통신 스레드가 지정된 목록의 로컬 'L' 변수에 부호의 오른쪽 식의 값을 설정합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC に송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

각 통신 스레드에는 자신의 'L' 변수 세트가 있습니다. 또한 각 좌표계에는 모션 프로그램에서 사용하는 좌표계 자체의 세트가 있으며 각 PLC 자체에도 세트가 있습니다. 'L' 변수의 모든 세트는 각각 독립적입니다.

예

L1=17.5	// Set value of L1 (set quantity 1, spacing 1) to 17.5
L100..102=P1+5	// Set value of L100, L101, L102 to (P1+5)
L50,3=-7	// Set value of 3 L-vars, starting at L50 (spacing 1) to -7
L50,3,3=sqrt(P25)	// Set value of 3 L-vars, starting at L50, spacing 3 // (L50, L53, L56) to the square root of P25

list apc

기능: Abort 시의 프로그램 카운터에서 프로그램을 목록화함

범위: 좌표계 고유

구문: **list apc [{상수}]**

여기에서

- 옵션인 **{상수}**는 프로그램을 Abort 했을 때 프로그램 카운터 장소에서 통지되는 행의 수를 지정합니다.

Power PMAC 은 **list apc** 명령으로 좌표계의 프로그램 카운터 포인트에서 시작하는 시점에서 주소 지정된 좌표계에서 지시하는 모션 프로그램 또는 서브 프로그램이 Abort 되었을 때 그 내용을 통지합니다. 내용은 ASCII 텍스트 형식으로 통지됩니다. 사용자 선언 변수명이나 텍스트 치환은 사용되지 않습니다.

Abort 는 사용자 명령에서 또는 어떤 폴트에서 자동으로 실행됩니다. **Abort** 가 발생하면 좌표계의 프로그램 카운터는 최상위 프로그램 시작으로 자동으로 리셋됩니다. 이 명령을 통해 사용자는 진단이나 디버그 목적으로 **Abort** 발생 시에 프로그램이 계산한 부분을 알 수 있습니다.

명령의 마지막에서 값도 지정되지 않고 쉼표도 사용되지 않는 경우(**list apc** 등)에 나머지 프로그램 전체가 통지되고 프로그램의 각 행에 선행하는 행 번호는 표시되지 않습니다.

명령의 마지막에서 값은 지정되지 않고 쉼표가 사용된 경우(**list apc,** 등)에 나머지 프로그램 전체가 통지되고 프로그램의 각 행에 선행하는 행 번호가 표시됩니다.

쉼표 뒤에 명령의 마지막에서 수치가 지정된 경우에는(**list apc, 5** 등)에 프로그램 카운터 장소에서 시작하는 수치로 지정된 행의 수가 통지됩니다. 행 번호는 프로그램이 통지된 각 행에 선행하여 표시됩니다. 또한 프로그램명은 선두에 표시되지만 그보다 먼저 현재 실행 중인 프로그램을 호출하는 고레벨의 프로그램명이 표시됩니다.

해당 명령으로 지정되는 행 번호는 루틴에서 사용되는 행 레이블('jump' 나 'sync')과 관계가 없습니다. 루틴의 최초 텍스트 행은 항상 행 번호 0 이고 2 번째 행은 행 번호 1 이며 이후에도 마찬가지로 이어집니다.

예

list apc Y20 dwell 50 X0 Y0	// List remaining program starting at program counter // Power PMAC response
list apc, prog 25 43:Y20 44:dwell 50 45:X0 Y0 46:end	// List remaining program starting at program counter // Power PMAC response
list apc,2 prog 25 43:Y20 44:dwell 50	// List 2 program lines starting at program counter // Power PMAC response

list {키네마틱스 버퍼}

기능: 지정된 키네마틱스 버퍼 내용을 목록화함

범위: 좌표계 고유

구문: **list {키네마틱스 버퍼}[,{{상수}},{상수}]**

여기에서

- **{키네마틱스 버퍼}**는 좌표계에 대한 Power PMAC 키네마틱스 루틴 타입을 지정합니다. 다음의 형식을 취할 수 있습니다.
 - 포워드 키네마틱스 루틴의 경우에 **forward**
 - 인버스 키네마틱스 루틴의 경우에 **inverse**

- 최초의 옵션 **{상수}**는 Top 에서 통지되는 행의 수(2 번째 상수가 지정되지 않은 경우) 또는 통지되는 최초의 행 번호를(2 번째 상수가 통지된 경우) 지정합니다.
- 2 번째 옵션 **{상수}**는 최초의 **{상수}**로 지정된 행 번호에서 통지되는 행의 수를 지정합니다.

Power PMAC 은 **list {키네마틱스 버퍼}** 명령으로 주소 지정된 좌표계에 대해 지정된 키네마틱스 루틴 버퍼 내용을 통지합니다. 내용은 ASCII 텍스트 형식으로 통지됩니다. 사용자 선언 변수명이나 텍스트 치환은 사용되지 않습니다.

루틴 타입 다음에 값도 지정되지 않고 쉼표도 사용되지 않는 경우(**list forward** 등)에 루틴 전체가 통지되고 행 번호는 루틴의 각 행에 선행하여 표시되지 않습니다.

루틴 타입 다음에 값이 지정되지 않고 쉼표가 사용된 경우(**list forward, 등**)에 루틴 전체가 통지되고 각 행 번호는 루틴의 각 행에 선행하여 표시됩니다.

루틴 타입 다음에 단일값이 지정된 경우(**list inverse, 5** 등)에 루틴 시작에서 시작되는 행 번호가 통지됩니다. 행 번호는 루틴이 통지된 각 행에 선행하여 표시됩니다.

루틴 타입 다음에 2 개의 값이 지정되는 경우(**list forward, 8, 20**)에 최초의 값은 통지되는 시작행의 번호를 지정하고 2 번째 값은 통지되는 행의 수를 지정합니다. 행 번호는 루틴이 통지된 각 행에 선행하여 표시됩니다.

해당 명령으로 지정되는 행 번호는 루틴에서 사용되는 행 레이블('jump' 나 'sync')과 관계가 없습니다. 루틴의 최초 텍스트 행은 항상 행 번호 0 이고 2 번째 행은 행 번호 1 이며 이후에도 마찬가지로 이어집니다.

list {프로그램 버퍼}

기능: 지정된 프로그램 버퍼 내용을 목록화함

범위: 글로벌

구문: **list {프로그램 버퍼}[,{상수},{상수}]**

여기에서

- **{프로그램 버퍼}**는 Power PMAC 프로그램의 이름을 지정합니다. 다음의 형식을 취할 수 있습니다.
 - o 모션 프로그램의 경우에는 **prog {상수}**로, 여기에서 **{상수}**는 모션 프로그램의 번호입니다.
 - o PLC 프로그램의 경우에는 **plc {상수}**로, 여기에서 **{상수}**는 PLC 프로그램의 번호입니다.
 - o 모션 프로그램의 경우에는 **subprog {상수}**로, 여기에서 **{상수}**는 서브 프로그램의 번호입니다.

- 최초의 옵션 **{상수}**는 Top 에서 통지되는 행의 수(2 번째 상수가 지정되지 않은 경우) 또는 통지되는 최초의 행 번호를(2 번째 상수가 통지된 경우) 지정합니다. 대신에 문자 **apc** 를 사용하여 실행이 Abort 된 프로그램 내의 포인트를 지정할 수 있습니다.
- 2 번째 옵션 **{상수}**는 최초의 **{상수}**로 지정된 행에서 시작하는 통지되는 행의 수를 지정합니다.

Power PMAC 은 **list {프로그램 버퍼}** 명령으로 지정된 프로그램 버퍼 내용을 통지합니다. 내용은 ASCII 텍스트 형식으로 통지됩니다. 사용자 선언 변수명이나 텍스트 치환은 사용되지 않습니다.

프로그램 타입 뒤에 값도 지정되지 않고 쉼표도 사용되지 않는 경우(**list prog 2000** 등)에 프로그램 전체가 통지되고 프로그램의 각 행에 선행하는 행 번호는 표시되지 않습니다.

프로그램 타입 다음에 값이 지정되지 않고 쉼표가 사용된 경우(**list prog 2000,** 등)에 프로그램 전체가 통지되고 프로그램의 각 행에 선행하는 행 번호가 표시됩니다.

프로그램 타입 다음에 단일값이 지정된 경우(**list plc 10, 5** 등)에 프로그램 시작을 시작하는 행 번호가 통지되고 행 번호는 프로그램의 통지된 각 행에 선행하여 표시됩니다.

프로그램 타입 다음에 2 개의 값이 지정되는 경우(**list subprog 1000, 8, 20** 등)에 최초의 값은 통지되는 시작 행의 번호를 지정하고 2 번째 값은 통지되는 행의 수를 지정합니다. 행 번호는 프로그램이 통지된 각 행에 선행하여 표시됩니다.

이러한 명령으로 지정되는 행 번호는 프로그램에서 사용되는 행 레이블('jump'나 'sync')과 관계는 없습니다. 프로그램 최초의 텍스트 행은 항상 행 번호 0 이고 2 번째 행은 행 번호 1 이며 이후에도 마찬가지로 이어집니다.

list pc

기능: 프로그램 카운터로 프로그램을 목록화함

범위: 좌표계 고유

구문: **list pc [{상수}]**

여기에서

- 옵션의 **{상수}**는 프로그램 카운터 장소에서 통지되는 행의 수를 지정합니다.

Power PMAC 은 **list pc** 명령으로 좌표계의 프로그램 카운터의 포인트에서 시작하는 주소 지정된 좌표계에서 현재 지시하는 모션 프로그램 또는 서브 프로그램의 내용을 통지합니다. 내용은 ASCII 텍스트 형식으로 통지됩니다. 사용자 선언 변수명이나 텍스트 치환은 사용되지 않습니다. 프로그램 카운터는 계산되는 모션 프로그램 또는 서브 프로그램에서 다음의 행을 지시합니다. 이에 따라 모션 프로그램의 계산 결과 현재 실행 중인 프로그램 동작보다 앞에 몇 개의 동작이 일어날 수 있습니다.

명령의 마지막에서 값이 지정되지 않은 경우(즉, **list pc**)에 남은 프로그램 전체가 통지되며 프로그램의 각 행에 선행하는 행 번호는 표시되지 않습니다.

명령의 마지막에 값이 지정되어 있지 않지만 쉼표가 붙어 있는 경우(즉, **list pc**)에 남은 프로그램 전체가 통지되고 프로그램의 각 행에 선행하는 행 번호가 표시됩니다.

쉼표 뒤인 명령의 마지막에 수치가 지정된 경우에는(예를 들어 **list pc, 5**) 프로그램 카운터 장소에서 시작하는 수치로 지정된 행의 수가 통지됩니다. 행 번호는 프로그램이 통지된 각 행에 선행하여 표시됩니다. 또한 프로그램명은 선두에 표시되지만 그보다 먼저 현재 실행 중인 프로그램을 호출하는 고레벨의 프로그램명이 표시됩니다.

해당 명령으로 지정되는 행 번호는 루틴에서 사용되는 행 레이블('jump' 나 'sync')과 관계가 없습니다. 루틴의 최초 텍스트 행은 항상 행 번호 0 이고 2 번째 행은 행 번호 1 이며 이후에도 마찬가지로 이어집니다.

예

list pc Y20 dwell 50 X0 Y0	// List remaining program starting at program counter // Power PMAC response
list pc, prog 25 43:Y20 44:dwell 50 45:X0 Y0 46:end	// List remaining program starting at program counter // Power PMAC response
list pc,2 prog 25 43:Y20 44:dwell 50	// List 2 program lines starting at program counter // Power PMAC response

list rotary

기능: 로터리 모션 프로그램의 버퍼 내용을 목록화함

범위: 좌표계 고유

구문: **list rotary** [{상수},{상수}]

여기에서

- 최초의 옵션 {상수}는 계산된 최신 행에서 통지되는 행의 수(2 번째 상수가 지정되지 않은 경우) 또는 계산되어 통지되는 최신 행 뒤의 최초의 번호를(2 번째 상수가 통지된 경우) 지정합니다. 대신에 문자 **apc** 를 사용하여 실행이 Abort 된 프로그램 내의 포인트를 지정할 수 있습니다.
- 2 번째 옵션 {상수}는 최초의 {상수}로 지정된 행 번호에서 통지되는 행의 수를 지정합니다.

Power PMAC 은 **list rotary** 명령으로 주소 지정된 좌표계에 대해 로터리 모션 프로그램의 버퍼 내용을 통지합니다. Power PMAC 에서 계산이 완료되지 않은 프로그램 명령행만 통지할 수 있습니다. 프로그램 명령의 계산이 완료되면(결과의 동작이 아직 실행되지 않은 경우에도) **list rotary** 명령에 따른 이 명령행의 통지는 없습니다.

명령의 마지막에서 값이 지정되지 않은 경우(즉, **list rotary**)에 나머지 프로그램 전체가 통지되고 행 번호는 프로그램의 각 행에 선행하여 표시되지 않습니다.

명령의 마지막에 값이 지정되지 않고 쉼표가 사용된 경우(다시 말해 **list rotary,**), 나머지 프로그램 전체가 통지되고 행 번호는 프로그램의 각 행에 선행하여 표시됩니다.

쉼표 뒤 명령의 마지막에 수치가 지정된 경우에는(**list rotary, 5** 등) 프로그램 카운터 장소에서 시작하는 수치로 지정된 행 번호가 통지됩니다. 행 번호는 프로그램이 통지된 각 행에 선행하여 표시됩니다. 또한 프로그램명은 선두에 표시되지만 그보다 먼저 현재 실행 중인 프로그램을 호출하는 고레벨의 프로그램명이 표시됩니다.

이 명령을 수취하면 좌표계의 로터리 버퍼는 클로즈해야 합니다.

M{데이터}

기능: 'M' 변수값을 통지함

범위: 글로벌

구문: **M{데이터}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.

Power PMAC 은 **M{데이터}** 명령으로 지정된 'M' 변수의 현재값을 통지합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이 되어야 합니다(표준 Power PMAC 펌웨어에서는 0~16,383). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

echo0	// Include variable names in query responses
M1	// Query value of variable M1
M1=17.5	// Power PMAC response of the value of M1
M(P1)	// Query value of I-variable numbered by M1
M122=3	// Power PMAC response of the value of M122
M(200+P17)	// Query value of M-variable numbered by P1+P17
M220=-5.35	// Power PMAC response of the value of M220
echo7	// Do not include variable names in query responses
M1	// Query value of variable M1
17.5	// Power PMAC response of the value of M1

M{데이터}={식}

기능: 값을 'M' 변수에 할당함

범위: 글로벌

구문: **M{데이터}={식}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.
- **{식}**은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC 은 **M{데이터}={식}** 명령으로 지정된 'M' 변수를 부호의 오른쪽 식의 값으로 설정합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이 되어야 합니다(표준 Power PMAC 펌웨어에서는 0~16,383). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

명령이 Power PMAC 에 송신되고 프로그램 버퍼가 오픈되면 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

예

M1=17.5	// Set variable M1 to 17.5
M(P1)=3	// Set M-variable numbered by P1 to 3
M(P1+P17)=5*sqrt(P100)	// Set M-var numbered by (P1+P17) to expression value

M{데이터}->

기능: 'M' 변수의 정의를 통지함

범위: 글로벌

구문: **M{데이터}->**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.

Power PMAC 은 **M{데이터}->**명령으로 지정된 'M' 변수의 정의를 통지합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이 되어야 합니다(표준 Power PMAC 펌웨어에서는 0~16,383). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

'I' 변수와는 달리 'M' 변수 정의는 사용자가 변경할 수 있습니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 2(값 4)는 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

echo0	// Include variable names in query responses
M162->	// Query definition of variable M162
M162->Motor[1].ActPos	// Power PMAC response of the definition of M162
M(P1)->	// Query definition of M-variable numbered by P1 (=0)
M0->u.io:\$A00000.8.1	// Power PMAC response of the definition of M0
echo7	// Do not include variable names in query responses
M162->	// Query definition of variable M162
Motor[1].ActPos	// Power PMAC response of the definition of M162

M{데이터}->{주소 정의}

기능: 주소에 따라 'M' 변수의 정의를 설정함

범위: 글로벌

구문: **M{데이터}->{주소 정의}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.
- **{주소 정의}**는 **{서식}.{메모리 뱅크}:{주소 오프셋}[{비트 오프셋}[{폭}]**으로 구성됩니다.
여기에서

- o **{서식}**은 변수 타입(이 주소의 값을 해석하는 방법)을 나타내는 단일 문자입니다. 다음을 선택할 수 있습니다.
 - **s** 는 명령이 범위보다 큰 값을 할당하려고 하면 포화하는(롤 오버하지 않음) 32 비트까지의 부호가 있는 정수를 지정합니다.
 - **i** 는 명령이 범위보다 큰 값을 할당하려고 하면 롤 오버하는(포화하지 않음) 32 비트까지의 부호가 있는 정수를 지정합니다.
 - **u** 는 명령이 범위보다 큰 값을 할당하려고 하면 롤 오버하는(포화하지 않음) 32 비트까지의 부호가 없는 정수를 지정합니다.
 - **f** 는 단정도(32 비트)의 부동 소수점 값을 지정합니다.
 - **d** 는 배정도(64 비트)의 부동 소수점 값을 지정합니다.
- o **{메모리 뱅크}**는 'M' 변수로 액세스하는 메모리 뱅크를 지정합니다. 다음을 선택할 수 있습니다.
 - **user** 는 Power PMAC RAM 이 예약된 사용자 버퍼를 지정합니다.
 - **io** 는 Power PMAC 의 주소 공간에서 메모리 매핑된 I/O 를 지정합니다.

- o **{주소 오프셋}**은 선택된 메모리 बैं크의 기준 주소에서 바이트 주소 오프셋을 지정하는 음의 값이 아닌 정수입니다.
- o 옵션의 **{비트 오프셋}**은 0~31 범위의 정수로 지정된 주소의 레지스터에 있는 변수의 시작 비트 번호를 나타냅니다. 이 지정은 정수의 변수 형식에서만 유효합니다. 값이 지정되지 않은 경우에는 0의 값을 사용합니다.
- o 옵션의 **{폭}**은 1~32 범위의 정수로 지정된 주소의 레지스터에 있는 변수로 사용되는 비트의 번호를 나타냅니다. 이 지정은 정수의 변수 형식에서만 유효합니다. 비트 오프셋 또는 폭에 값이 지정되지 않은 경우에는 폭에 32의 값이 사용됩니다. 비트 오프셋에 값이 설정되고 폭의 값이 지정되지 않은 경우에 폭에 1의 값이 사용됩니다.

Power PMAC은 **M{데이터}>{주소 정의}** 명령으로 주소 정의에 따라 지정된 'M' 변수의 정의를 설정합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이 되어야 합니다(표준 Power PMAC 펌웨어에서는 0~16,383). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다. Power PMAC에 송신되는 명령은 식의 값뿐이라는 것이 중요합니다. 그 후 식의 값을 변경한 경우에 정의는 다른 'M' 변수에 이식되지 않습니다.

주소 정의 최초의 항목은 변수 서식을 나타내는 문자입니다. 이 서식은 레지스터의 정보를 해석하여 처리하는 방법을 결정합니다. 3개의 정수 서식을 고려할 수 있습니다. **s**(부호가 있는 포화 정수), **i**(부호가 있는 오버플로 정수), **u**(부호가 없는 오버플로 정수)입니다. 정수 서식의 폭은 1~32 비트까지 가능합니다. 2개의 부동 소수점 서식을 고려할 수 있습니다. **f**(32비트 단정도), **d**(64비트 배정도)입니다.

주소 정의의 2 번째 항목은 변수가 정의되는 메모리 बैं크입니다. 여기에서도 2개의 선택지가 있습니다. **user**는 Power PMAC의 RAM에서 큰 영역으로 예약된 사용자 버퍼를 지정하고(사용자 설정 크기, 기본값은 1MB), **io**는 Power PMAC의 축 인터페이스 레지스터에 메모리 매핑된 I/O, 범용 아날로그 및 디지털 I/O, 듀얼 포트 RAM 레지스터의 각 주소를 지정합니다.

주소 정의의 3 번째 항목은 선택된 메모리 बैं크의 시작 주소에서 얻을 수 있는 관련된 레지스터의 수치 주소 오프셋입니다. 여기에서 오프셋을 지정하면 절대 물리 메모리 주소를 알 필요가 없다는 점에 주의하십시오. Power PMAC은 이 오프셋을 메모리 बैं크의 기준 주소에 더하여 절대 주소를 계산합니다. 예를 들어 사용자 메모리 버퍼의 기준 주소는 [Sys.pushm]에 기재되고 메모리 매핑된 I/O의 기준 주소는 [Sys.piom]에 기재됩니다.

모든 정수 서식과 단정도의 부동 소수점 서식 주소 오프셋은 long 정수 경계에 있고 4로 균등하게 나누어야 합니다. 배정도의 부동 소수점 서식 주소 오프셋은 long long 정수 경계에 있고 8로 균등하게 나누어야 합니다.

정수 서식에서는 주소 오프셋으로 정의되는 32 비트 레지스터의 일부만 사용할 수도 있습니다. 비트 오프셋이 명시적으로 선언된 경우에 사용되는 32 비트 레지스터의 시작(최하위) 비트 번호를 지정합니다. 이는 0~31의 값을 취해야 합니다. 여기에서 값이 선언되지 않는 경우에는 시작 비트 번호는 0이 됩니다(모든 32비트가 사용됩니다).

비트 오프셋이 명시적으로 선언된 경우에는 비트 폭도 선언됩니다. 정의 내에서 사용되는 비트 번호를 지정하고 비트 오프셋으로 선언된 비트로 시작할 수 있습니다. 오프셋 후에 비트 폭이 선언되지 않는 경우에는 1 의 폭이 사용됩니다. 비트 오프셋과 비트 폭의 합계는 32 이하여야 합니다.

예

M0->u.io:\$A00000.8.1	// Set def of M0 in I/O space, 1 bit starting at bit 8
M80->s.user:0	// Set def of M80 in user buffer, 32-bit signed int
M(P1)->d.user:\$100	// Set def of M(P1) in user buffer, 64-bit float
M5000->u.io:\$A10000.8.8	// Set def of M5000 in I/O space, 8-bit unsigned int

M{데이터}->{데이터 구조체 요소}

기능: 데이터 구조체 요소에 따라 [M] 변수의 정의를 설정함

범위: 글로벌

구문: **M{데이터}->{데이터 구조체 요소}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.
- **{데이터 구조체 요소}**는 사전에 정의된 Power PMAC 의 데이터 구조체 요소 이름입니다.

Power PMAC 은 **M{데이터}->{데이터 구조체 요소}** 명령으로 포인트까지 지정된 'M' 변수의 정의를 지정된 데이터 구조체 요소로 설정합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이 되어야 합니다(표준 Power PMAC 펌웨어에서는 0~16,383). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다. Power PMAC 에 송신되는 명령은 식의 값뿐이라는 것이 중요합니다. 그 후 식의 값을 변경한 경우에 정의는 다른 'M' 변수에 이식되지 않습니다.

이 타입의 'M' 변수 정의는 데이터 구조체 요소에 생략 표현명 즉, 변수명이 'M' 변수로 선언 또는 정의된 경우의 커스터마이징된 이름을 허가합니다.

이 타입의 정의에서는 'M' 변수는 지정된 데이터 구조체 요소의 서식을 자동으로 취득합니다.

예

M0->Sys.ServoCount	// Set definition of M0 to global servo cycle counter
M162->Motor[1].ActPos	// Set definition of M162 to #1 actual position register
M5187->Coord[1].InPos	// Set definition of M5187 to &1 in-position bit
M(P10)->Sys.Fdata[0]	// Set definition of M(P10) to user buffer location 0
	// used as single-precision float
M15000->Sys.Ddata[L0]	// Set definition of M15000 to user buffer indexable
	// location, used as double-precision float

M{데이터}->{자기 참조 정의}

기능: 'M' 변수의 정의를 자기 참조 정의로 설정함

범위: 글로벌

구문: **M{데이터}->{자기 참조 정의}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.
- **{자기 참조 정의}**는 ***[{서식}][{폭}]**으로 구성됩니다. 여기에서
 - o **{서식}**은 변수 타입(이 주소의 값을 해석하는 방법)을 나타내는 단일 문자입니다. 다음을 선택할 수 있습니다.
 - **s** 는 명령이 범위보다 큰 값을 할당하려고 하면 포화하는(롤 오버하지 않음) 32 비트까지의 부호가 있는 정수를 지정합니다.
 - **i** 는 명령이 범위보다 큰 값을 할당하려고 하면 롤 오버하는(포화하지 않음) 32 비트까지의 부호가 있는 정수를 지정합니다.
 - **u** 는 명령이 범위보다 큰 값을 할당하려고 하면 롤 오버하는(포화하지 않음) 32 비트까지의 부호가 없는 정수를 지정합니다.
 - **f** 는 단정도(32 비트)의 부동 소수점 값을 지정합니다.
 - **d** 는 배정도(64 비트)의 부동 소수점 변수(실제로는 가수의 마지막 비트가 소실되므로 기술적으로는 63 비트만 해당)를 지정합니다. 서식을 지정하지 않는 경우에는 Power PMAC 은 **d** 서식을 사용합니다.
 - o 옵션의 **{폭}**은 1~32 범위의 정수로 지정된 주소의 레지스터에 있는 변수로 사용되는 비트의 번호를 나타냅니다. 이 지정은 정수의 변수 형식에서만 유효합니다. 값이 지정되지 않은 경우에는 폭에 32의 값을 사용합니다.

Power PMAC 은 **M{데이터}->{자기 참조 정의}** 명령으로 변수값에 정의 레지스터의 일부를 사용하기 위해 지정된 'M' 변수의 정의를 설정합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이 되어야 합니다(표준 Power PMAC 펌웨어에서는 0~16,383). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다. Power PMAC 에 송신되는 명령은 식의 값뿐이라는 것이 중요합니다. 그 후 식의 값을 변경한 경우에 정의는 다른 'M' 변수에 이식되지 않습니다.

자기 참조하는 'M' 변수는 특정 서식의 범용 글로벌 사용자 변수입니다. 글로벌 변수의 합계수를 덧셈 또는 글로벌 'P' 변수와는 다른 서식의 변수 작성에 활용할 수 있습니다. 이는 모두 배정도의 부동 소수점 변수입니다. 이는 변수의 정의 클리어에도 사용할 수 있으므로 이 변수에 대한 값의 할당은 문제 발생의 우려가 있는 쓰기를 할 수 없습니다.

**참고사항**

자기 참조하는 'M' 변수에서는 동기 대입(**M100==1** 등)을 실행할 수 없습니다. Power PMAC 프로그램에서 동기 대입을 하려고 하면 'no-op'가 되고 에러는 통지되지 않습니다.

예

M70->*u	// Set M70 as self-defined unsigned 32-bit integer
M80->*s.16	// Set M80 as self-defined 16-bit signed integer
M8000->*f	// Set M8000 as self-defined 32-bit float
M(P1)->*d	// Set M(P1) as self-defined 64-bit float
M9000->*f	// Set M9000 as self-defined 64-bit float

M{변수 목록}

기능: 목록의 'M' 변수값을 통지함

범위: 글로벌

구문: **M{변수 목록}[:{상수}]**

여기에서

- **{변수 목록}**은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.
 - **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작하여 2 번째 정수값(최초의 정수값보다 큰 번호여야 합니다)으로 번호를 매긴 변수로 종료하는 연속된 번호의 변수 범위를 지정합니다.
 - **{상수}[,{상수}[,{상수}]]**에서는 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- 옵션인 **{상수}**는 응답 행마다 통지된 값의 수를 지정하는 양의 정수입니다. 여기서 값을 지정하지 않는 경우에 응답 행마다 하나의 값만 통지됩니다. 값을 지정하지 않고 사용할 수 있는 것은 목록이 연속된 수의 범위로 지정된 경우에 한합니다.

Power PMAC 은 **M{변수 목록}** 명령으로 목록 내 'M' 변수의 현재값을 통지합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

명령이 Power PMAC に송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

예

M100 M100=1	// Query value of var M100 (set quantity 1, spacing 1) // Power PMAC response of the value of M100
M120..122 M120=-4 M121=3.5 M122=25	// Query value of M120, M121, M122 // Power PMAC response of the value of M120 // Power PMAC response of the value of M121 // Power PMAC response of the value of M122
M120,3 M120=-4 M121=3.5 M122=25	// Query value of 3 M-vars, starting at M120 (spacing 1) // Power PMAC response of the value of M120 // Power PMAC response of the value of M121 // Power PMAC response of the value of M122
M120,3,100 M122=25 M222=25 M322=25	// Query value of 3 M-vars, starting at M120, spacing 100 // Power PMAC response of the value of M122 // Power PMAC response of the value of M222 // Power PMAC response of the value of M322
M3100..3109:5 M3100=0,0,0,0,0 M3105=0,0,0,0,0	// Query value of M3100 – M3109, 5 values per line // Power PMAC response of values of M3100 – M3104 // Power PMAC response of values of M3105 – M3109

M{변수 목록}={식}

기능: 목록의 'M' 변수값을 설정함

범위: 글로벌

구문: **M{변수 목록}={식}**

여기에서

- **{변수 목록}**은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.
 - **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작하여 2 번째 정수값(최초의 정수값보다 큰 번호여야 합니다)으로 번호를 매긴 변수로 종료하는 연속된 번호의 변수 범위를 지정합니다.
 - **{상수}[,{상수}[,{상수}]**에서는 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- **{식}**은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC 은 **M{변수 목록}={식}** 명령으로 이 통신 스레드에 로컬 목록 내 'M' 변수값을 부호의 오른쪽 식의 값으로 설정합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로

지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC に송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

예

M100=1	// Set value of M100 (set quantity 1, spacing 1) to 1
M122..123=P1+5	// Set value of M122, M123 to (P1+5)
M122,2=17.5	// Set val of 2 M-vars, starting at M122 (spacing 1) to 17.5
M122,3,100=P25+7	// Set value of 3 M-vars, starting at M122, spacing 100
	// (M122, M222, M1322) to (P25+7)

M{변수목록}>

기능: 목록의 'M' 변수 정의를 통지함

범위: 글로벌

구문: **M{변수 목록}>**

여기에서

- **{변수 목록}**은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.
 - **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작하여 2 번째 정수값(최초의 정수값보다 큰 번호여야 합니다)으로 번호를 매긴 변수로 종료하는 연속된 번호의 변수 범위를 지정합니다.
 - **{상수},{상수},{상수}**에서는 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).

Power PMAC 은 **M{변수 목록}>**명령으로 목록 내 'M' 변수의 정의를 통지합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 2(값 4)는 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

명령이 Power PMAC 에 송신되고 프로그램 버퍼가 오픈되면 명령은 에러로 거부됩니다.

예

```

M0->                                     // Query definition of M0
M0->u.io:$A00000.8.1                     // Power PMAC response of the definition of M0

M0..2->                                   // Query definitions of M0, M1, M2
M0->u.io:$A00000.8.1                     // Power PMAC response of the definition of M0
M1->u.io:$A00000.9.1                     // Power PMAC response of the definition of M1
M2->u.io:$A00000.10.1                    // Power PMAC response of the definition of M2

M0,3->                                   // Query defs of 3 M-vars starting with M0 (spacing 1)
M0->u.io:$A00000.8.1                     // Power PMAC response of the definition of M0
M1->u.io:$A00000.9.1                     // Power PMAC response of the definition of M1
M2->u.io:$A00000.10.1                    // Power PMAC response of the definition of M2

M0,3,100->                               // Query defs of 3 M-vars starting with M0, spacing 100
M0->u.io:$A00000.8.1                     // Power PMAC response of the definition of M0
M100->u.io:$A08000.8.1                   // Power PMAC response of the definition of M100
M200->u.io:$A10000.8.1                   // Power PMAC response of the definition of M200

```

M{변수 목록}->{주소 정의}

기능: 주소에 따라 목록 내 'M' 변수의 정의를 설정함

범위: 글로벌

구문: **M{변수 목록}->{주소 정의}**

여기에서

- **{변수 목록}**은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.
 - **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작하여 2 번째 정수값(최초의 정수값보다 큰 번호여야 합니다)으로 번호를 매긴 변수로 종료하는 연속된 번호의 변수 범위를 지정합니다.
 - **{상수}[,{상수}[,{상수}]]**에서는 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- **{주소 정의}**는 **{서식}.{메모리 뱅크}:{주소 오프셋}[,{비트 오프셋}[,{폭}]]**로 구성됩니다.

여기에서

 - **{서식}**은 변수 타입(이 주소의 값을 해석하는 방법)을 나타내는 단일 문자입니다. 다음을 선택할 수 있습니다.
 - **s** 는 명령이 범위보다 큰 값을 할당하려고 하면 포화하는(롤 오버하지 않음) 32 비트까지의 부호가 있는 정수를 지정합니다.
 - **i** 는 명령이 범위보다 큰 값을 할당하려고 하면 롤 오버하는(포화하지 않음) 32 비트까지의 부호가 있는 정수를 지정합니다.
 - **u** 는 명령이 범위보다 큰 값을 할당하려고 하면 롤 오버하는(포화하지 않음) 32 비트까지의 부호가 없는 정수를 지정합니다.

- **f**는 단정도(32 비트)의 부동 소수점 값을 지정합니다.
- **d**는 배정도(64 비트)의 부동 소수점 값을 지정합니다.
- **{메모리 뱅크}**는 'M' 변수로 액세스하는 메모리 뱅크를 지정합니다. 다음을 선택할 수 있습니다.
 - **user**는 Power PMAC RAM이 예약된 사용자 버퍼를 지정합니다.
 - **io**는 Power PMAC의 주소 공간에서 메모리 매핑된 I/O를 지정합니다.
- **{주소 오프셋}**은 선택된 메모리 뱅크의 기준 주소에서 바이트 주소 오프셋을 지정하는 음의 값이 아닌 정수입니다.
- 옵션의 **{비트 오프셋}**은 0~31 범위의 정수로 지정된 주소의 레지스터에 있는 변수의 시작 비트 번호를 나타냅니다. 이 지정은 정수의 변수 형식에서만 유효합니다. 값이 지정되지 않은 경우에는 0의 값을 사용합니다.
- 옵션의 **{폭}**은 1~32 범위의 정수로 지정된 주소의 레지스터에 있는 변수로 사용되는 비트의 번호를 나타냅니다. 이 지정은 정수의 변수 형식에서만 유효합니다. 비트 오프셋 또는 폭에 값이 지정되지 않은 경우에는 폭에 32의 값이 사용됩니다. 비트 오프셋에 값이 설정되고 폭의 값이 지정되지 않은 경우에 폭에 1의 값이 사용됩니다.

Power PMAC은 **M{변수 목록}>{주소 정의}** 명령으로 주소 정의에 따라 목록 내 'M' 변수의 정의를 설정합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

주소 정의 최초의 항목은 변수 서식을 나타내는 문자입니다. 이 서식은 레지스터의 정보를 해석하여 처리하는 방법을 결정합니다. 3개의 정수 서식을 고려할 수 있습니다. **s**(부호가 있는 포화 정수), **i**(부호가 있는 오버플로 정수), **u**(부호가 없는 오버플로 정수)입니다. 정수 서식의 폭은 1~32 비트까지 가능합니다. 2개의 부동 소수점 서식을 고려할 수 있습니다. **f**(32 비트 단정도), **d**(64 비트 배정도)입니다.

주소 정의의 2 번째 항목은 변수가 정의되는 메모리 뱅크입니다. 여기에서도 2 개의 선택지가 있습니다. **user**는 Power PMAC의 RAM에서 큰 영역으로 예약된 사용자 버퍼를 지정하고(사용자 설정 크기, 기본값은 1MB), **io**는 Power PMAC의 축 인터페이스 레지스터에 메모리 매핑된 I/O, 범용 아날로그 및 디지털 I/O, 듀얼 포트 RAM 레지스터의 각 주소를 지정합니다.

주소 정의의 3 번째 항목은 선택된 메모리 뱅크의 시작 주소에서 얻을 수 있는 관련된 레지스터의 수치 주소 오프셋입니다. 여기에서 오프셋을 지정하면 절대 물리 메모리 주소를 알 필요가 없다는 점에 주의하십시오. Power PMAC은 이 오프셋을 메모리 뱅크의 기준 주소에 더하여 절대 주소를 계산합니다. 예를 들어 사용자 메모리 버퍼의 기준 주소는 **Sys.pushm**에 기재되고 메모리 매핑된 I/O의 기준 주소는 **Sys.piom**에 기재됩니다.

모든 정수 서식과 단정도의 부동 소수점 서식 주소 오프셋은 long 정수 경계에 있고 4로 균등하게 나누어야 합니다. 배정도의 부동 소수점 서식 주소 오프셋은 long long 정수 경계에 있고 8로 균등하게 나누어야 합니다.

정수 서식에서는 주소 오프셋으로 정의되는 32 비트 레지스터의 일부만 사용할 수도 있습니다. 비트 오프셋이 명시적으로 선언된 경우에 사용되는 32 비트 레지스터의 시작(최하위) 비트 번호를 지정합니다. 이는 0~31의 값을 취해야 합니다. 여기에서 값이 선언되지 않는 경우에는 시작 비트 번호는 0이 됩니다(모든 32비트가 사용됩니다).

비트 오프셋이 명시적으로 선언된 경우에는 비트 폭도 선언됩니다. 정의 내에서 사용되는 비트 번호를 지정하고 비트 오프셋으로 선언된 비트로 시작할 수 있습니다. 오프셋 후에 비트 폭이 선언되지 않는 경우에는 1의 폭이 사용됩니다. 비트 오프셋과 비트 폭의 합계는 32 이하여야 합니다.

이 명령은 여러 변수가 단일 명령으로 같은 정의를 취할 수 있게 하지만 그 가능성은 거의 도움이 되지 않습니다.

예

M0->u.io:\$A00000.8.1	// Set def of M0 in I/O space, 1 bit starting at bit 8
M80->s.user:0	// Set def of M80 in user buffer, 32-bit signed int
M5000->u.io:\$A10000.8.8	// Set def of M5000 in I/O space, 8-bit unsigned int

M{변수 목록}->{데이터 구조체 요소}

기능: 데이터 구조체 요소로 목록의 'M' 변수 정의를 설정함

범위: 글로벌

구문: M{변수 목록}->{데이터 구조체 요소}

여기에서

- {변수 목록}은 1개 이상의 변수 세트를 지정합니다. 목록은 다음 2개의 형식 중 1개를 취할 수 있습니다.
 - {상수}..{상수}에서는 최초의 정수값으로 번호를 매긴 변수로 시작되고 2번째 정수값(최초보다 큰 번호여야 합니다)으로 번호를 할당된 변수로 종료하는 연속적인 번호의 변수 범위를 지정합니다.
 - {상수}[{상수}][{상수}]에서 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1이 됩니다), 옵션의 3번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1이 됩니다).
- {데이터 구조체 요소}는 사전에 정의된 Power PMAC의 데이터 구조체 요소 이름입니다.

Power PMAC은 M{변수 목록}->{데이터 구조체 요소} 명령으로 포인트까지의 목록 내 'M' 변수 정의를 지정된 데이터 구조체 요소로 설정합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다.

목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

이 타입의 'M' 변수 정의는 데이터 구조체 요소에 생략 표현명 즉, 변수명이 'M' 변수로 선언 또는 정의된 경우의 커스터마이징된 이름을 허가합니다.

이 타입의 정의에서는 'M' 변수는 지정된 데이터 구조체 요소의 서식을 자동으로 취득합니다.

이 명령은 여러 변수가 단일 명령으로 같은 정의를 취할 수 있게 하지만 그 가능성은 거의 도움이 되지 않습니다.

예

```
M0->Sys.ServoCount          // Set definition of M0 to global servo cycle counter
M162->Motor[1].ActPos        // Set definition of M162 to #1 actual position register
M5187->Coord[1].InPos        // Set definition of M5187 to #1 in-position bit
M15000->Sys.Ddata[L0]        // Set definition of M15000 to user buffer indexable
                             // location, used as double-precision float
```

M{변수 목록}->{자기 참조 정의}

기능: 목록 내의 'M' 변수 정의를 자기 참조 정의로 설정함

범위: 글로벌

구문: **M{변수 목록}->{자기 참조 정의}**

여기에서

- **{변수 목록}**은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.
 - o **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작되고 2 번째 정수값(최초보다 큰 번호여야 합니다)으로 번호를 할당된 변수로 종료하는 연속적인 번호의 변수 범위를 지정합니다.
 - o **{상수}[,{상수}[,{상수}]]**에서 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- **{자기 참조 정의}**는 ***[{서식}][{쪽}]**으로 구성됩니다. 여기에서
 - o **{서식}**은 변수 타입(이 주소의 값을 해석하는 방법)을 나타내는 단일 문자입니다. 다음을 선택할 수 있습니다.
 - **S** 는 명령이 범위보다 큰 값을 할당하려고 하면 포화하는(롤 오버하지 않음) 32 비트까지의 부호가 있는 정수를 지정합니다.
 - **i** 는 명령이 범위보다 큰 값을 할당하려고 하면 롤 오버하는(포화하지 않음) 32 비트까지의 부호가 있는 정수를 지정합니다.

- **u** 는 명령이 범위보다 큰 값을 할당하려고 하면 롤 오버하는 (포화하지 않음) 32 비트까지의 부호가 없는 정수를 지정합니다.
 - **f** 는 단정도(32 비트)의 부동 소수점 값을 지정합니다.
 - **d** 는 배정도(64 비트)의 부동 소수점 변수(실제로는 가수의 마지막 비트가 소실되므로 기술적으로는 63 비트만 해당)를 지정합니다. 서식을 지정하지 않는 경우에는 Power PMAC 은 **d** 서식을 사용합니다.
- o 옵션의 **{폭}**은 1~32 범위의 정수로 지정된 주소의 레지스터에 있는 변수로 사용되는 비트의 번호를 나타냅니다. 이 지정은 정수의 변수 형식에서만 유효합니다. 값이 지정되지 않은 경우에는 폭에 32 의 값을 사용합니다.

Power PMAC 은 **M{변수 목록->{자기 참조 정의}}** 명령으로 변수값에 정의 레지스터의 일부를 사용하므로 목록 내의 'M' 변수 정의를 설정합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

자기 참조하는 'M' 변수는 특정 서식의 범용 글로벌 사용자 변수입니다. 글로벌 변수의 합계수를 덧셈 또는 글로벌 'P' 변수와는 다른 서식의 변수 작성에 활용할 수 있습니다. 이는 모두 배정도의 부동 소수점 변수입니다. 이는 변수의 정의 클리어에도 사용할 수 있으므로 이 변수에 대한 값의 할당은 문제 발생의 우려가 있는 쓰기를 할 수 없습니다.

각 변수의 정의는 자기 참조 정의 내의 독자적인 레지스터를 사용하므로 단일 명령 내에서 여러 변수를 정의하는 경우에 도움이 됩니다.

예

```
M70->*u           // Set M70 as self-defined unsigned 32-bit integer
M80..89->*s.16     // Set M80 - M89 as self-defined 16-bit signed integers
M8000->*f          // Set M8000 as self-defined 32-bit float
M10000..10999->*d  // Set M10000 - M10999 as self-defined 64-bit floats
M0..16383->*       // Clear all M-variable definitions
```

open forward

기능: 입력하는 포워드 키네마틱스 프로그램의 버퍼를 오픈함

범위: 통신 스레드, 좌표계 고유

구문: **open forward**[[{상수}]]{상수}

여기에서

- 최초 옵션의 **{상수}**는 양의 정수이며 프로그램이 서브루틴 또는 서브 프로그램을 호출하는 경우의 로컬 변수 스택 오프셋 크기를 지정합니다. 값이 지정되지 않은 경우에는 256 의 값이 사용됩니다.
- 2 번째 옵션의 **{상수}**는 양의 정수이며 프로그램이 사용하는 행으로 점프하는 경우의 레이블 번호의 최대값을 지정합니다. 번호가 지정되지 않은 경우에는 1,024 의 값이 사용됩니다.

Power PMAC은 **open forward** 명령으로 이 통신 스레드의 프로그램 명령을 입력하기 위해 주소 지정된 좌표계의 포워드 키네마틱스 프로그램의 버퍼를 엽니다. 이 스레드로 송신된 후속 프로그램 명령이 프로그램 버퍼에 입력됩니다. 프로그램에 대한 입력이 종료된 후 그 이상 명령행이 입력되지 않도록 **close** 명령을 사용하여 클로즈합니다. 그 다음에 프로그램의 실행을 허가해야 합니다.

좌표계의 포워드 키네마틱스 프로그램에는 모터(관절) 좌표를 축(공구 선단) 좌표로 변환하는 계산이 포함됩니다. 이 프로그램에서는 축 정의 구문의 스케일링과 오프셋보다 높은 유연한 변환을 실행할 수 있습니다. 임의의 좌표축에 대해 Power PMAC으로 포워드 키네마틱스 프로그램이 설정된 경우에는 그 좌표계에서 모션 프로그램의 시작, **pmatch** 또는 **pset** 명령, 축 위치 또는 속도 요구 등의 모터와 축의 변환이 요구되면 포워드 키네마틱스 프로그램은 항상 자동으로 실행됩니다.

프로그램 번호 다음에 인수가 없는 경우에 Power PMAC은 서브루틴의 호출로 기본값의 로컬 변수 스택 오프셋(256)을 사용하여 행 점프 레이블의 기본값 테이블 크기(1,024)를 예약합니다. 최초의 쉽표 다음에 수치가 선언된 경우에 그 수치는 프로그램이 서브루틴을 호출하는 경우에 사용되는 스택 오프셋의 크기를 합니다(서브루틴의 'R0'은 이 프로그램의 **L(StackOffset)**에 해당합니다). 수치가 크면 보다 많은 인수를 지정된 서브루틴의 호출에 전달할 수 있지만 8,192 개의 변수 스택 전체가 적은 수의 서브루틴으로 모두 소진됩니다.

서브루틴이 IDE 프로젝트 매니저를 거쳐 다운로드된 경우에 IDE는 사용되는 로컬 변수의 수를 처리하기 위해 충분한 최저한의 스택 오프셋을 자동으로 선언합니다.

2 번째 쉽표 다음에 수치가 선언된 경우에 수치는 프로그램에서 사용 가능한 행으로 점프하는 레이블의 최대수를 지정합니다. 수치가 크면 보다 많은 메모리를 필요로 하고 점프 실행에 여유 시간이 필요합니다.

서브루틴이 IDE 프로젝트 매니저를 거쳐 다운로드된 경우에 IDE는 사용되는 행의 점프 레이블 수를 처리하기 위해 충분한 최소의 행 레이블 버퍼를 자동으로 선언합니다.

이 명령을 사용하여 프로그램 버퍼를 오픈하면 이 번호의 기존의 프로그램 버퍼 내용을 자동으로 삭제합니다. PMAC 또는 Turbo PMAC처럼 **clear** 명령은 사용하지 않습니다. 무효인 구문 또는 온라인 명령으로만 실시하는 구문 중 하나가 원인으로 버퍼가 오픈되어 있는 동안에 버퍼에 입력할 수 없는 명령행이 이 스레드로 송신되면 명령은 에러로 거부됩니다. 버퍼 내용은 삭제되고 버퍼는 자동으로 클로즈됩니다.

이 명령이 송신될 때에 모든 스레드는 다른 프로그램 버퍼를 오픈할 수 없습니다. Power PMAC은 이 명령을 에러로 거부합니다. 유지, 정지 또는 싱글 스텝 명령 등 중단된 시점에서 재기동할 수 있는 명령으로 프로그램을 중단한 경우에도 이 명령으로는 모션 프로그램은 실행할 수 없으며 PLC 프로그램도 유효가 되지 않습니다. Power PMAC은 이 명령을 에러로 거부합니다(**a(Abort)** 명령은 실행 중인 모션 프로그램 또는 중단된 모션 프로그램을 완전히 종료시키기 위해 사용할 수 있습니다).

예

open forward	// Open forward-kinematic program buffer for addressed
	// C.S. default stack offset, default number of labels
&2open forward,64	// Open F.K program buffer for C.S. 2,
	// stack offset of 64, default number of labels
&4open forward,,2048	// Open F.K. program buffer for C.S. 4,
	// default stack offset, up to 2048 labels
&12open forward,512,256	// Open F.K. program buffer for C.S. 12,
	// stack offset of 512, up to 2048 labels

open inverse

기능: 입력하는 인버스 키네마틱스 프로그램의 버퍼를 오픈함

범위: 통신 스레드, 좌표계 고유

구문: **open inverse**[**{상수}**][**{상수}**]

여기에서

- 최초 옵션의 **{상수}**는 양의 정수이며 프로그램이 서브루틴 또는 서브 프로그램을 호출하는 경우의 로컬 변수 스택 오프셋 크기를 지정합니다. 값이 지정되지 않은 경우에는 256의 값이 사용됩니다.
- 2 번째 옵션의 **{상수}**는 양의 정수이며 프로그램이 사용하는 행으로 점프하는 경우의 레이블 번호의 최대값을 지정합니다. 번호가 지정되지 않은 경우에는 1,024의 값이 사용됩니다.

Power PMAC은 **open inverse** 명령으로 이 통신 스레드의 프로그램 명령을 입력하기 위해 주소 지정된 좌표계의 인버스 키네마틱스 프로그램의 버퍼를 오픈합니다. 이 스레드로 송신된 후속 프로그램 명령이 프로그램 버퍼에 입력됩니다. 프로그램에 대한 입력이 종료된 후 그 이상 명령행이 입력되지 않도록 **close** 명령을 사용하여 클로즈합니다. 그 다음에 프로그램의 실행을 허가해야 합니다.

좌표계의 인버스 키네마틱스 프로그램에는 축(공구 선단) 좌표계를 모터(관절) 좌표계로 변환하는 계산이 포함됩니다. 이 프로그램에서는 축 정의 구문의 스케일링과 오프셋보다 높은 유연한 변환을 실행할 수 있습니다. 좌표계의 축이 인버스 키네마틱스 축으로 정의된 경우에(**#x->I**) 그 좌표계에 대해 요구된 축과 모터의 변환이 자동으로 실행됩니다(세그먼트화되지 않은 동작에 대해 동작마다 1 회, 세그먼트화된 동작에 대해 세그먼트마다 1 회, 변환이 실행됩니다).

프로그램 번호 다음에 인수가 없는 경우에 Power PMAC은 서브루틴의 호출로 기본값의 로컬 변수 스택 오프셋(256)을 사용하여 행 점프 레이블의 기본값 테이블 크기(1,024)를 예약합니다. 최초의 쉼표 다음에 수치가 선언된 경우에 그 수치는 프로그램이 서브루틴을 호출하는 경우에 사용되는 스택 오프셋의 크기를 합니다(서브루틴의 'R0'은 이 프로그램의 **L(StackOffset)**에 해당합니다). 수치가 크면 보다 많은 인수를 지정된 서브루틴의 호출에 전달할 수 있지만 8,192 개의 변수 스택 전체가 적은 수의 서브루틴으로 모두 소진됩니다.

서브루틴이 IDE 프로젝트 매니저를 거쳐 다운로드된 경우에 IDE는 사용되는 로컬 변수의 수를 처리하기 위해 충분한 최저한의 스택 오프셋을 자동으로 선언합니다.

2 번째 쉼표 다음에 수치가 선언된 경우에 수치는 프로그램에서 사용 가능한 행으로 점프하는 레이블의 최대수를 지정합니다. 수치가 크면 보다 많은 메모리를 필요로 하고 점프 실행에 여유 시간이 필요합니다.

서브루틴이 IDE 프로젝트 매니저를 거쳐 다운로드된 경우에 IDE는 사용되는 행의 점프 레이블 수를 처리하기 위해 충분한 최소의 행 레이블 버퍼를 자동으로 선언합니다.

이 명령을 사용하여 프로그램 버퍼를 오픈하면 이 번호의 기존의 프로그램 버퍼 내용을 자동으로 삭제합니다. PMAC 또는 Turbo PMAC처럼 **clear** 명령은 사용하지 않습니다. 무효인 구문 또는 온라인 명령으로만 실시하는 구문 중 하나가 원인으로 버퍼가 오픈되어 있는 동안에 버퍼에 입력할 수 없는 명령행이 이 스레드로 송신되면 명령은 에러로 거부됩니다. 버퍼 내용은 삭제되고 버퍼는 자동으로 클로즈됩니다.

이 명령이 송신될 때에 모든 스레드는 다른 프로그램 버퍼를 오픈할 수 없습니다. Power PMAC 은 이 명령을 에러로 거부합니다. 유지, 정지 또는 싱글 스텝 명령 등 중단된 시점에서 재기동할 수 있는 명령으로 프로그램을 중단한 경우에도 이 명령으로는 모션 프로그램은 실행할 수 없으며 PLC 프로그램도 유효가 되지 않습니다. Power PMAC 은 이 명령을 에러로 거부합니다(a(Abort) 명령은 실행 중인 모션 프로그램 또는 중단된 모션 프로그램을 완전히 종료시키기 위해 사용할 수 있습니다).

예

open inverse	// Open inverse-kinematic program buffer for addressed
	// C.S. default stack offset, default number of labels
&2open inverse,64	// Open I.K program buffer for C.S. 2,
	// stack offset of 64, default number of labels
&4open inverse,,2048	// Open I.K. program buffer for C.S. 4,
	// default stack offset, up to 2048 labels
&12open inverse,512,256	// Open I.K. program buffer for C.S. 12,
	// stack offset of 512, up to 2048 labels

open plc

기능: 입력하는 PLC 프로그램의 버퍼를 오픈함

범위: 통신 스레드 고유

구문: **open plc{상수}[, [{상수}]][, {상수}]**

여기에서

- 최초의 **{상수}**는 0~31 범위의 양의 정수이며 버퍼를 오픈하는 PLC 프로그램의 번호를 지정합니다.
- 2 번째 옵션의 **{상수}**는 양의 정수이며 프로그램이 서브루틴 또는 서브 프로그램을 호출하는 경우의 로컬 변수 스택 오프셋 크기를 지정합니다. 값이 지정되지 않은 경우에는 256의 값이 사용됩니다.
- 3 번째 옵션 **{상수}**는 양의 정수이며 프로그램이 사용하는 행으로 점프하는 경우의 레이블 번호의 최대값을 지정합니다. 번호가 지정되지 않은 경우에는 1,024의 값이 사용됩니다.

Power PMAC 은 **open plc** 명령으로 이 통신 스레드의 프로그램 명령을 입력하기 위해 지정된 PLC 프로그램의 버퍼를 오픈합니다. 이 스레드로 송신된 후속 프로그램 명령이 프로그램 버퍼에 입력됩니다. 프로그램에 대한 입력이 종료된 후 그 이상 명령행이 입력되지 않도록 **close** 명령을 사용하여 클로즈합니다. 그 다음에 프로그램의 실행을 허가해야 합니다.

프로그램이 IDE 프로젝트 매니저를 거쳐 다운로드된 경우에 텍스트명을 프로그램에 사용할 수 있으며(**open plc SafetyInterlocks** 등), IDE 는 자동으로 텍스트명을 프로그램 번호로 치환합니다. 텍스트명은 프로그램의 그 밖의 모든 참조에서 사용할 수 있습니다(**enable plc SafetyInterlocks** 등).

프로그램 번호 다음에 인수가 없는 경우에 Power PMAC 은 서브루틴의 호출로 기본값의 로컬 변수 스택 오프셋(256)을 사용하여 행 점프 레이블의 기본값 테이블 크기(1,024)를 예약합니다. 최초의 쉼표 다음에 수치가 선언된 경우에 그 수치는 프로그램이 서브루틴을 호출하는 경우에 사용되는 스택 오프셋 크기를 지정합니다(서브루틴의 'R0'은 이 프로그램의 [L(StackOffset)]에 상당합니다). 수치가 크면 보다 많은 인수를 지정된 서브루틴의 호출에 전달할 수 있지만 8,192 개의 변수 스택 전체가 적은 수의 서브루틴으로 모두 소진됩니다.

프로그램이 IDE 프로젝트 매니저를 거쳐 다운로드된 경우에 IDE 는 사용되는 로컬 변수의 수를 처리하기 위해 충분한 최저한의 스택 오프셋을 자동으로 선언합니다.

2 번째 쉼표 다음에 수치가 선언된 경우에 수치는 프로그램에서 사용 가능한 행으로 점프하는 레이블의 최대수를 지정합니다. 수치가 크면 보다 많은 메모리를 필요로 하고 점프 실행에 여유 시간이 필요합니다.

프로그램이 IDE 프로젝트 매니저를 거쳐 다운로드된 경우에 IDE 는 사용되는 행의 점프 레이블 수를 처리하기 위해 충분한 최소의 행 레이블 버퍼를 자동으로 선언합니다.

이 명령을 사용하여 프로그램 버퍼를 오픈하면 이 번호의 기존의 프로그램 버퍼 내용을 자동으로 삭제합니다. PMAC 또는 Turbo PMAC 처럼 clear 명령은 사용하지 않습니다. 무효인 구문 또는 온라인 명령으로만 실시하는 구문 중 하나가 원인으로 버퍼가 오픈되어 있는 동안에 버퍼에 입력할 수 없는 명령행이 이 스레드로 송신되면 명령은 에러로 거부됩니다. 버퍼 내용은 삭제되고 버퍼는 자동으로 클로즈됩니다.

이 명령이 송신될 때에 지정된 PLC 프로그램은 중단 또는 싱글 스텝 모드가 아니라 Disable 되어야 합니다. Disable 되지 않은 경우에 Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령이 송신될 때에 모든 스레드는 다른 프로그램 버퍼를 오픈할 수 없습니다. Power PMAC 은 이 명령을 에러로 거부합니다.

예

```
open plc 0           // Open PLC program buffer 0, default stack and labels
open plc 1,64        // Open PLC program buffer 1,
                     // stack offset of 64, default number of labels
open plc 7,,2048     // Open PLC program buffer 7,
                     // default stack offset, up to 2048 labels
open plc 23,512,256  // Open PLC program buffer 23,
                     // stack offset of 512, up to 2048 labels
```

open prog

기능: 입력하는 모션 프로그램의 버퍼를 오픈함

범위: 통신 스레드 고유

구문: **open prog{상수}[, [{상수}]][, {상수}]**

여기에서

- 최초의 {변수}는 음의 값이 아닌 32 비트 정수이며 버퍼를 오픈하는 모션 프로그램의 번호를 지정합니다.

- 2 번째 옵션의 {상수}는 양의 정수이며 프로그램이 서브루틴 또는 서브 프로그램을 호출하는 경우의 로컬 변수 스택 오프셋 크기를 지정합니다. 값이 지정되지 않은 경우에는 256의 값이 사용됩니다.
- 3 번째 옵션{상수}는 양의 정수이며 프로그램이 사용하는 행으로 점프하는 경우의 레이블 번호의 최대값을 지정합니다. 번호가 지정되지 않은 경우에는 1,024의 값이 사용됩니다.

Power PMAC은 **open prog** 명령으로 이 통신 스레드의 프로그램 명령을 입력하기 위해 지정된 모션 프로그램의 버퍼를 엽니다. 이 스레드로 송신된 후속 프로그램 명령이 프로그램 버퍼에 입력됩니다. 프로그램에 대한 입력이 종료된 후 그 이상 명령행이 입력되지 않도록 **close** 명령을 사용하여 클로즈합니다. 그 다음에 프로그램의 실행을 허가해야 합니다.

프로그램이 IDE 프로젝트 매니저를 거쳐 다운로드된 경우에 텍스트명을 프로그램에 사용할 수 있으며(**open prog CalibrationMove** 등), IDE는 자동으로 텍스트명을 프로그램 번호로 치환합니다. 텍스트명은 프로그램에 대한 그 밖의 모든 참조에서 사용할 수 있습니다(&1 start Calibration Moves 등).

open prog 0 명령은 **open rot** 명령에 상당하며 주소 지정된 로터리 모션 프로그램의 버퍼를 엽니다.

프로그램 번호 다음에 인수가 없는 경우에 Power PMAC은 서브루틴의 호출로 기본값의 로컬 변수 스택 오프셋(256)을 사용하여 행 점프 레이블의 기본값 테이블 크기(1,024)를 예약합니다. 최초의 쉼표 다음에 수치가 선언된 경우에 그 수치는 프로그램이 서브루틴을 호출하는 경우에 사용되는 스택 오프셋 크기를 지정합니다(서브루틴의 'R0'은 이 프로그램의 [L(StackOffset)]에 상당합니다). 수치가 크면 보다 많은 인수를 지정된 서브루틴의 호출에 전달할 수 있지만 8,192개의 변수 스택 전체가 적은 수의 서브루틴으로 모두 소진됩니다.

프로그램이 IDE 프로젝트 매니저를 거쳐 다운로드된 경우에 IDE는 사용되는 로컬 변수의 수를 처리하기 위해 충분한 최저한의 스택 오프셋을 자동으로 선언합니다.

2 번째 쉼표 다음에 수치가 선언된 경우에 수치는 프로그램에서 사용 가능한 행으로 점프하는 레이블의 최대수를 지정합니다. 수치가 크면 보다 많은 메모리를 필요로 하고 점프 실행에 여유 시간이 필요합니다.

프로그램이 IDE 프로젝트 매니저를 거쳐 다운로드된 경우에 IDE는 사용되는 행의 점프 레이블 수를 처리하기 위해 충분한 최소의 행 레이블 버퍼를 자동으로 선언합니다.

이 명령을 사용하여 프로그램 버퍼를 오픈하면 이 번호의 기존의 프로그램 버퍼 내용을 자동으로 삭제합니다. PMAC 또는 Turbo PMAC처럼 **CLEAR** 명령은 사용하지 않습니다. 무효인 구문 또는 온라인 명령으로만 실시하는 구문 중 하나가 원인으로 버퍼가 오픈되어 있는 동안에 버퍼에 입력할 수 없는 명령행이 이 스레드로 송신되면 명령은 에러로 거부됩니다. 버퍼 내용은 삭제되고 버퍼는 자동으로 클로즈됩니다.

유지, 정지 또는 싱글 스텝 명령 등 중단된 시점에서 재기동할 수 있는 명령으로 프로그램을 중단한 경우에도 모션 프로그램이 Power PMAC의 메모리에 이미 존재하는 경우에는 이 명령을 실행할 수 없습니다. Power PMAC은 이 명령을 에러로 거부합니다(**a**(Abort) 명령은 실행 중인 모션 프로그램 또는 중단된 모션 프로그램을 완전히 종료시키기 위해 사용할 수 있습니다).

이 명령이 송신될 때에 모든 스레드는 다른 프로그램 버퍼를 오픈할 수 없습니다. Power PMAC은 이 명령을 에러로 거부합니다.

예

```
open prog 1           // Open motion program buffer 1,
                      // default stack and labels
open prog 100,64       // Open motion program buffer 100,
                      // stack offset of 64, default number of labels
open prog 2001,,2048   // Open motion program buffer 2001,
                      // default stack offset, up to 2048 labels
open prog 5,512,256    // Open motion program buffer 5,
                      // stack offset of 512, up to 2048 labels
```

open rotary

기능: 입력하는 로터리 모션 프로그램의 버퍼를 오픈함

범위: 좌표계 고유

구문: **open rotary**

Power PMAC 은 **open rotary** 명령으로 프로그램에 입력하기 위해 주소 지정된 좌표계의 로터리 모션 프로그램의 버퍼를 오픈합니다. 이 포트로 송신된 로터리 모션 프로그램에 대해 유효한 후속 프로그램 명령이 로터리 모션 프로그램의 버퍼에 입력됩니다.

이 명령이 송신될 때에 다른 프로그램 버퍼를 오픈할 수 없습니다. 이 좌표계에 대해 로터리 버퍼가 정의된 상태여야 합니다.

로터리 모션 프로그램의 버퍼는 좌표계의 모션 프로그램 0으로 간주되므로 명령 **open prog 0** 은 같은 결과가 됩니다.

정의된 로터리 버퍼가 오픈되면 주소 지정된 이 좌표계에 송신된 프로그램 명령이 로터리 버퍼에 자동으로 입력됩니다.

이 좌표계가 주소 지정된 동안에 **close** 명령을 전송하면 입력을 위해 오픈한 로터리 버퍼가 클로즈합니다.

open subprog

기능: 입력하는 서브 프로그램의 버퍼를 오픈함

범위: 통신 스레드 고유

구문: **open subprog{상수}[, {상수}][, {상수}]**

여기에서

- 최초의 **{상수}**는 32 비트 양의 정수이며 버퍼를 오픈하는 서브 프로그램의 번호를 지정합니다.
- 2 번째 옵션의 **{상수}**는 양의 정수이며 프로그램이 서브루틴 또는 서브 프로그램을 호출하는 경우의 로컬 변수 스택 오프셋 크기를 지정합니다. 값이 지정되지 않은 경우에는 256 의 값이 사용됩니다.
- 3 번째 옵션 **{상수}**는 양의 정수이며 프로그램이 사용하는 행으로 점프하는 경우의 레이블 번호의 최대값을 지정합니다. 번호가 지정되지 않은 경우에는 1,024 의 값이 사용됩니다.

Power PMAC 은 **open subprog** 명령으로 이 통신 스레드의 프로그램 명령을 입력하기 위해 지정된 서브 프로그램의 버퍼를 엽니다. 이 스레드로 송신된 후속 프로그램 명령이 프로그램 버퍼에 입력됩니다. 서브 프로그램에 대한 입력이 종료된 후 그 이상 명령행이 입력되지 않도록 **close** 명령을 사용하여 클로즈합니다. 그 다음에 프로그램의 실행을 허가해야 합니다.

프로그램이 IDE 프로젝트 매니저를 거쳐 다운로드된 경우에 텍스트명을 프로그램에 사용할 수 있으며(**open subprog DrillCycle** 등), IDE 는 자동으로 텍스트명을 프로그램 번호로 치환합니다. 텍스트명은 프로그램의 그 밖의 모든 참조에서 사용할 수 있습니다(**call DrillCycle** 등).

서브 프로그램 번호 다음에 인수가 없는 경우에 Power PMAC 은 서브루틴의 호출로 기본값의 로컬 변수 스택 오프셋(256)을 사용하여 행 점프 레이블의 기본값 테이블 크기(1,024)를 예약합니다. 최초의 쉼표 다음에 수치가 선언된 경우에 그 수치는 프로그램이 서브루틴을 호출하는 경우에 사용되는 스택 오프셋 크기를 지정합니다 (서브루틴의 'R0'은 이 프로그램의 [L(StackOffset)]에 상당합니다). 수치가 크면 보다 많은 인수를 지정된 서브루틴의 호출에 전달할 수 있지만 8,192 개의 변수 스택 전체가 적은 수의 서브루틴으로 모두 소진됩니다.

프로그램이 IDE 프로젝트 매니저를 거쳐 다운로드된 경우에 IDE 는 사용되는 로컬 변수의 수를 처리하기 위해 충분한 최저한의 스택 오프셋을 자동으로 선언합니다.

2 번째 쉼표 다음에 수치가 선언된 경우에 수치는 프로그램에서 사용 가능한 행으로 점프하는 레이블의 최대수를 지정합니다. 수치가 크면 보다 많은 메모리를 필요로 하고 점프 실행에 여유 시간이 필요합니다.

프로그램이 IDE 프로젝트 매니저를 거쳐 다운로드된 경우에 IDE 는 사용되는 행의 점프 레이블 수를 처리하기 위해 충분한 최소의 행 레이블 버퍼를 자동으로 선언합니다.

이 명령을 사용하여 서브 프로그램 버퍼를 엽니다 이 번호의 기존의 프로그램 버퍼 내용을 자동으로 삭제합니다. PMAC 또는 Turbo PMAC 처럼 **CLEAR** 명령은 사용하지 않습니다. 무효인 구문 또는 온라인 명령으로만 실시하는 구문 중 하나가 원인으로 버퍼가 오픈되어 있는 동안에 버퍼에 입력할 수 없는 명령행이 이 스레드로 송신되면 명령은 에러로 거부됩니다. 버퍼 내용은 삭제되고 버퍼는 자동으로 클로즈됩니다.

이 명령이 송신될 때에 모든 스레드는 다른 프로그램 버퍼를 엽니다 수 없습니다. Power PMAC 은 이 명령을 에러로 거부합니다. 유지, 정지 또는 싱글 스텝 명령 등 중단된 시점에서 재기동할 수 있는 명령으로 프로그램을 중단한 경우에도 이 명령으로는 모션 프로그램은 실행할 수 없습니다. Power PMAC 은 이 명령을 에러로 거부합니다(**a(Abort)** 명령은 실행 중인 모션 프로그램 또는 중단된 모션 프로그램을 완전히 종료시키기 위해 사용할 수 있습니다). 이 명령으로는 PLC 프로그램을 유효로 설정할 수 없습니다. Power PMAC 은 이 명령을 에러로 거부합니다.

이 명령이 송신될 때에 모든 스레드는 다른 프로그램 버퍼를 엽니다 수 없습니다. Power PMAC 은 이 명령을 에러로 거부합니다.

예

```
open subprog 1           // Open motion program buffer 1,
                        // default stack and labels
open subprog 100,64      // Open motion program buffer 100,
                        // stack offset of 64, default number of labels
open subprog 2001,,2048  // Open motion program buffer 2001,
                        // default stack offset, up to 2048 labels
open subprog 5,512,256   // Open motion program buffer 5,
                        // stack offset of 512, up to 2048 labels
```

out{상수}

기능: 오픈 루프 출력

범위: 모터 고유

구문: **out{상수}**

여기에서

- **{상수}**는 부동 소수점의 값이며 출력 레벨을 모터의 서보 루프 최대값 비율로 지정합니다.

Power PMAC 은 **out{상수}** 명령으로 지정된 모터를 오픈 루프 유효 모드로 설정하고 모터의 최대 출력 파라미터 [Motor[x].MaxDac]의 비율로 지정된 값을 서보 루프에 강제로 출력합니다. 모터 목록이 없는 경우에 앞에서 설명한 동작은 현재 주소 지정된 모터가 대상이 됩니다. 모터 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 모터가 대상이 됩니다.

이 명령은 일반적으로 설정과 진단을 목적으로 사용됩니다(예를 들어 양의 **out** 명령으로 위치는 양의 방향으로 카운트되어야 합니다. 즉 클로즈드 루프 제어를 확립할 수 없습니다). 단, 실제 어플리케이션에서 사용할 수 있습니다.

모터가 Power PMAC 을 통해 전류(commutation)되지 않는 경우에 이 명령은 DAC 및 펄스 주파수 레지스터 등 모터의 단일 출력 레지스터 상수값을 작성합니다. 모터가 Power PMAC 을 통해 전류(commutation)되는 경우에 전류(commutation) 알고리즘이 계속 활성화되고 지정된 값이 전류(commutation) 알고리즘에 대한 직교(토크) 명령 입력이 됩니다.

Power PMAC 의 **out{상수}** 명령은 종래의 PMAC 및 Turbo PMAC 컨트롤러의 **o{상수}**에 상당합니다.

값이 +/-100.0 인 범위 밖으로 지정된 경우에 출력은 출력 리미트값의 +/-100%로 포화합니다. 예러는 통지되지 않습니다.

p

기능: 현재 위치값을 통지함

범위: 모터 또는 좌표계 고유

구문: **p**

Power PMAC 은 **p** 명령으로 지정된 모터 또는 좌표계의 현재 위치값을 통지합니다. 모터 목록 또는 좌표계 목록이 없는 경우에는 현재 주소 지정된 모터의 현재 위치값을 통지합니다. 모터 목록이 있는 경우에는 목록 내 모든 모터의 현재 위치값을 통지합니다. 좌표계 목록이 있는 경우에는 목록 내 모든 좌표계에 모든 활성축의 현재 위치값을 통지합니다.

여러 모터 목록 또는 여러 좌표계 목록을 지정해도 후속 모터 고유의 명령 또는 좌표계 고유의 명령으로 모달에 주소 지정된 모터 또는 좌표계는 변경되지 않습니다.

모터 위치를 통지하는 경우에 위치는 기본 모터 단위로 표시됩니다. 이 통지의 참조(제로)점은 오프셋 모드 명령 위치 레지스터 내의 값의 사용 방법을 결정하는 2개의 제어 설정에 좌우됩니다. 오프셋 모드 명령 위치 레지스터에서는 몇 개의 위치 명령 소스 중첩이 허용됩니다.

일반적으로 캠 테이블 또는 캠 스타일의 보정 테이블 출력인 `[Motor[x].CompDesPos]`는 항상 오프셋 모드의 레지스터입니다. 일반적으로 위치 추종(전자 기어)의 결과인 `[Motor[x].ActiveMasterPos]`는 `[Motor[x].MasterCtrl]`의 비트 1(값 2)을 1로 설정한 경우에 오프셋 모드의 레지스터입니다.

`[Motor[x].PosReportMode]`(2013 년 3/4 분기에 출시된 V1.6 펌웨어의 새 기능)를 기본값인 0으로 설정한 경우 또는 이 새로운 요소가 없는 이전 버전의 펌웨어를 사용한 경우에는 이러한 오프셋 모드의 레지스터값을 최종적인 명령 위치에서 뺍니다. `[Motor[x]]`의 소스 레지스터에서 작업하는 경우에 통지된 위치는 다음과 같이 계산됩니다.

```
+ Pos           // 전원 셋업/리셋 시의 위치에 대한 원시 피드백 상대 위치
+ CompPos       // 보정 테이블 측정의 수정
+ PresBlSize    // 백래시 수정
- CompDesPos     // 보정 테이블 명령 오프셋
- ActiveMasterPos * MasterCtrlOffsetBit // 오프셋 모드 시의 마스터 위치
- HomePos       // 전원 셋업/리셋 시의 위치에 대한 모터 제로 상대 위치
```

`[Motor[x].PosReportMode]`를 1로 설정하면 오프셋 모드의 이러한 레지스터값을 최종적인 명령 위치에서 빼지 않습니다. 이 경우에 통지되는 위치는 다음과 같이 계산됩니다.

```
+ Pos           // 전원 셋업/리셋 시의 위치에 대한 원시 피드백 상대 위치
+ CompPos       // 보정 테이블 측정의 수정
+ PresBlSize    // 백래시 수정
- HomePos       // 전원 셋업/리셋 시의 위치에 대한 모터 제로 상대 위치
```

축 위치를 통지하는 경우에 위치는 축의 프로그램 원점 위치(반드시 기본 모터의 제로 위치와 같다고 할 수 없음)에 대해 스케일링된 사용자 축 단위로 표시됩니다. 각 축에 통지된 값은 축의 이름 (1개 또는 2개의 영문자)앞에 설명합니다. Power PMAC은 그 위치가 이 명령으로 계산하는 각 축의 값을 통신 스레드 $L(256 + [\text{Sys.MaxMotors} + n])$ 을 위한 로컬 변수로 남겨 둡니다. 여기서 n 은 0~31까지의 축 인덱스값(A는 0, B는 1 등)입니다.

그 좌표계의 축에 모터가 할당되지 않은 좌표계에서 통지가 요구되면 Power PMAC은 문자열 'No Motors'를 반환합니다. 좌표계의 축 정의 셋팅 또는 포워드 키네마틱스 서브루틴으로 축 데이터를 적절하게 계산할 수 없는 경우에 Power PMAC은 문자열 'No Solution'을 반환합니다.

예

```
p           // Query actual position of presently addressed motor
1001        // Power PMAC response in counts

#1..4p      // Query actual positions of Motors 1 - 4
982 -3462 27 86643 // Power PMAC response in counts

&1p        // Query actual positions of axes in C.S. 1
X0.982 Y-3.462 Z0.027 C86.643 // Power PMAC response in user units
```

P{데이터}

기능: 'P' 변수값을 통지함

범위: 글로벌

구문: **P{데이터}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.

Power PMAC 은 **P{데이터}** 명령으로 지정된 'P' 변수의 현재값을 통지합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~65,535). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

echo0	// Include variable names in query responses
P1	// Query value of variable P1
P1=17.5	// Power PMAC response of the value of P1
P(P1)	// Query value of P-variable numbered by P1
P17=3	// Power PMAC response of the value of P17
P(P1+P17)	// Query value of P-variable numbered by P1+P17
P20=-5.35	// Power PMAC response of the value of P20
echo7	// Do not include variable names in query responses
P1	// Query value of variable P1
17.5	// Power PMAC response of the value of P1

P{데이터}={식}

기능: 값을 'P' 변수에 할당함

범위: 글로벌

구문: **P{데이터}={식}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.
- **{식}**은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC은 **P{데이터}={식}** 명령으로 지정된 'P' 변수를 부호의 오른쪽 식의 값으로 설정합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~65,535). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

명령이 Power PMAC에 송신되고 프로그램 버퍼가 오픈되면 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

예

P1=17.5	// Set variable P1 to 17.5
P(P1)=3	// Set P-variable numbered by P1 to 3
P(P1+P17)=5*sqrt(P100)	// Set P-var numbered by (P1+P17) to expression value

P{변수 목록}

기능: 목록의 'P' 변수값을 통지함

범위: 글로벌

구문: **P{변수 목록}[:{상수}]**

여기에서

- **{변수 목록}**은 1개 이상의 변수 세트를 지정합니다. 목록은 다음 2개의 형식 중 1개를 취할 수 있습니다.
 - **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작되고 2 번째 정수값(최초보다 큰 번호여야 합니다)으로 번호를 할당된 변수로 종료하는 연속적인 번호의 변수 범위를 지정합니다.
 - **{상수}[,{상수}[,{상수}]**에서 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- 옵션의 **{상수}**는 응답 행마다 통지된 값의 수를 지정하는 양의 정수입니다. 여기서 값을 지정하지 않는 경우에 응답 행마다 하나의 값만 통지됩니다. 값을 지정하지 않고 사용할 수 있는 것은 목록이 연속된 수의 범위로 지정된 경우에 한합니다.

Power PMAC은 **P{변수 목록}** 명령으로 목록 내 'P' 변수의 현재값을 통지합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC에 송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

```
P1 // Query value of variable P1 (set quantity 1, spacing 1)
P1=17.5 // Power PMAC response of the value of P1

P100..102 // Query value of P100, P101, P102
P100=-4 // Power PMAC response of the value of P100
P101=3.14159 // Power PMAC response of the value of P101
P102=0 // Power PMAC response of the value of P102

P50,3 // Query value of 3 P-vars, starting at P50 (spacing 1)
P50=-5.35 // Power PMAC response of the value of P50
P51=7.12 // Power PMAC response of the value of P51
P52=376452 // Power PMAC response of the value of P52

P50,3,3 // Query value of 3 P-vars, starting at P50, spacing 3
P50=-5.35 // Power PMAC response of the value of P50
P53=99 // Power PMAC response of the value of P53
P56=-0.002 // Power PMAC response of the value of P56

P1..10:5 // Query value of P1 - P10, 5 values per response line
P1=7,3,5,2,11 // Power PMAC response of the value of P1 - P5
P6=8,33,4,9,-2 // Power PMAC response of the value of P6 - P10
```

P{변수 목록}={식}

기능: 목록의 'P' 변수값을 설정함

범위: 글로벌

구문: **P{변수 목록}={식}**

여기에서

- **{변수 목록}**은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.
 - **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작되고 2 번째 정수값(최초보다 큰 번호여야 합니다)으로 번호를 할당된 변수로 종료하는 연속적인 번호의 변수 범위를 지정합니다.
 - **{상수},{상수},{상수}**에서 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- **{식}**은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC은 **P{변수 목록}={식}** 명령으로 목록 내의 'P' 변수에 부호의 오른쪽 식의 값으로 설정합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC에 송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

예

P1=17.5	// Set value of P1 (set quantity 1, spacing 1) to 17.5
P100..102=P1+5	// Set value of P100, P101, P102 to (P1+5)
P50,3=-7	// Set value of 3 P-vars, starting at P50 (spacing 1) to -7
P50,3,3=sqrt(P25)	// Set value of 3 P-vars, starting at P50, spacing 3
	// (P50, P53, P56) to the square root of P25

pause

기능: 모션 프로그램의 실행을 정지함

범위: 좌표계 고유

구문: **q, pause**

pause 명령은 **q** 명령의 비생략형입니다. Power PMAC은 이 명령으로 지정한 좌표계의 모션 프로그램의 실행을 마지막에 계산된 동작 끝에서 정지시킵니다. [Sys.NoShortCmds]가 기본값인 0으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

pause plc

기능: 지정된 PLC 프로그램의 실행을 일시 정지함

범위: 글로벌

구문: **pause plc {목록}**

여기에서

- {목록}**은 쉼표로 구분된 정수 상수 또는 시작 상수에서 종료 상수 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 모든 상수는 0~31까지의 범위이며 실행을 일시 정지하는 PLC 프로그램을 지정합니다.

Power PMAC은 **pause plc** 온라인 명령으로 후속 스캔의 시작을 금지하고 지정된 PLC 프로그램의 실행을 정지합니다. 이 명령이 프로그램 도중에 실행을 중단(프로그램이 [while] 루프 내에서 프리즈한 경우 등)한 경우에도 **resume plc** 명령을 사용하여 정지한 장소에 실행을 재개할 수 있습니다.

disable plc 명령은 마찬가지로 PLC 프로그램의 실행을 정지하기 위해 사용하지만 이 명령은 실행이 정지된 장소와 관계없이 프로그램 선두에서 실행을 재개하도록 합니다.

명령으로 지정된 PLC 프로그램이 Power PMAC 에 존재하지 않는 경우에 에러가 통지되지 않습니다. 명령은 명령으로 지정된 다른 PLC 프로그램에 작용합니다.

예

```
pause plc 1           // Pause execution of PLC 1
pause plc 2,4,6       // Pause execution of PLCs 2, 4, and 6
pause plc 7..10       // Pause execution of PLCs 7, 8, 9, and 10
pause plc 11,13..16,20 // Pause execution of PLCs 11, 13, 14, 15, 16, and 20
```

q

기능: 모션 프로그램의 실행을 정지함

범위: 좌표계 고유

구문: **q, pause**

Power PMAC 은 **q** 명령으로 지정된 좌표계의 모션 프로그램의 실행을 마지막에 계산된 동작 끝에서 정지시킵니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계에 대해 프로그램의 실행을 정지합니다. 좌표계 목록이 있는 경우에 목록 내의 모든 좌표계에 대해 프로그램의 실행을 정지합니다.

좌표계가 블렌드된 동작 중에서 모션 프로그램을 예측한 경우에는 현재 실행한 동작에 이어지는 1개 이상의 동작 계산을 실행하여 모션 신호에 둡니다. 이러한 동작은 **q** 명령에 따라 정지하기 전에 실행됩니다.

모션 프로그램의 실행은 해당 명령으로 정지 중인 동안에서는 일시 정지하지만 논리적으로는 아직 프로그램 내에 있으며 그 시점에서 실행을 재개할 수 있습니다. 일시 정지 후에도 프로그램의 실행을 재개하지 않는 경우에는 **a** 명령을 사용하여 프로그램의 실행을 완전히 Abort 해야 합니다.

이 명령의 생략형(**q**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**pause**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012년 3/4 분기에 출시된 V1.5의 새 기능)가 1로 설정된 경우에는 이 명령의 비생략형(**pause**)을 사용해야 합니다. 생략형(**q**)은 올바르지 않은 명령이며 에러 20 으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

동등한 버퍼 프로그램의 다이렉트 명령은 **pause** 입니다.

예

```
q           // Quit program execution in addressed C.S.
&2q        // Address C.S.2 and quit program execution there
&1,3,5q    // Quit program execution in C.S. 1, 3, and 5
&*q        // Quit program execution in all C.S.
```

Q{데이터}

기능: 'Q' 변수값을 통지함

범위: 좌표계 고유

구문: **Q{데이터}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.

Power PMAC은 **Q{데이터}** 명령으로 주소 지정된 좌표계(&n)에 대해 지정된 'Q' 변수의 현재값을 통지합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~8,191). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

```
&1 // Modally address C.S.1, will access C.S.1's Q-vars
echo0 // Include variable names in query responses
Q1 // Query value of variable Q1
Q1=17.5 // Power PMAC response of the value of Q1

Q(P1) // Query value of Q-variable numbered by P1
Q17=3 // Power PMAC response of the value of Q17

Q(P1+P17) // Query value of Q-variable numbered by P1+P17
Q20=-5.35 // Power PMAC response of the value of Q20

echo7 // Do not include variable names in query responses
Q1 // Query value of variable Q1
17.5 // Power PMAC response of the value of Q1
```

Q{데이터}={식}

기능: 값을 'Q' 변수에 할당함

범위: 좌표계 고유

구문: **Q{데이터}={식}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.
- **{식}**은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC은 **Q{데이터}={식}** 명령으로 주소 지정된 좌표계(&n)에 지정된 'Q' 변수를 부호의 오른쪽 식의 값으로 설정합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~8,191). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

명령이 Power PMAC에 송신되고 프로그램 버퍼가 오픈되면 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

예

&3	// Modally address C.S.3, will access C.S.3's Q-vars
Q1=17.5	// Set variable Q1 to 17.5
Q(P1)=3	// Set Q-variable numbered by P1 to 3
Q(P1+P17)=5*sqrt(P100)	// Set Q-var numbered by (P1+P17) to expression value

Q{변수 목록}

기능: 목록의 'Q' 변수값을 통지함

범위: 좌표계 고유

구문: **Q{변수 목록}[:{상수}]**

여기에서

- **{변수 목록}**은 1개 이상의 변수 세트를 지정합니다. 목록은 다음 2개의 형식 중 1개를 취할 수 있습니다.
 - **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작되고 2 번째 정수값(최초보다 큰 번호여야 합니다)으로 번호를 할당된 변수로 종료하는 연속적인 번호의 변수 범위를 지정합니다.
 - **{상수}[, {상수}[, {상수}]]**에서 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- 옵션의 **{상수}**는 응답 행마다 통지된 값의 수를 지정하는 양의 정수입니다. 여기서 값을 지정하지 않는 경우에 응답 행마다 하나의 값만 통지됩니다. 값을 지정하지 않고 사용할 수 있는 것은 목록이 연속된 수의 범위로 지정된 경우에 한합니다.

Power PMAC은 **Q{변수 목록}** 명령으로 주소 지정된 좌표계(&n) 목록 내의 'Q' 변수값을 통지합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC에 송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

```
&2 // Modally address C.S.2, will access C.S.2's Q-vars
Q1 // Query value of variable Q1 (set quantity 1, spacing 1)
Q1=17.5 // Power PMAC response of the value of Q1

Q100..102 // Query value of Q100, P101, P102
Q100=-4 // Power PMAC response of the value of Q100
Q101=3.14159 // Power PMAC response of the value of Q101
Q102=0 // Power PMAC response of the value of Q102

Q50,3 // Query value of 3 Q-vars, starting at Q50 (spacing 1)
Q50=-5.35 // Power PMAC response of the value of Q50
Q51=7.12 // Power PMAC response of the value of Q51
Q52=376452 // Power PMAC response of the value of Q52

Q50,3,3 // Query value of 3 Q-vars, starting at Q50, spacing 3
Q50=-5.35 // Power PMAC response of the value of Q50
Q53=99 // Power PMAC response of the value of Q53
Q56=-0.002 // Power PMAC response of the value of Q56

Q1..10:5 // Query value of Q1 - Q10, 5 values per response line
Q1=7,3,5,2,11 // Power PMAC response of the value of Q1 - Q5
Q6=8,33,4,9,-2 // Power PMAC response of the value of Q6 - Q10
```

Q{변수 목록}={식}

기능: 목록의 'Q' 변수값을 설정함

범위: 좌표계 고유

구문: **Q{변수 목록}={식}**

여기에서

- **{변수 목록}**은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.
 - **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작되고 2 번째 정수값(최초보다 큰 번호여야 합니다)으로 번호를 할당된 변수로 종료하는 연속적인 번호의 변수 범위를 지정합니다.
 - **{상수}[,{상수}[,{상수}]**에서 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- **{식}**은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC 은 **Q{변수 목록}={식}** 명령으로 주소 지정된 좌표계(&n) 목록 내의 'Q' 변수값을 부호의 오른쪽 식의 값으로 설정합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다.

목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC に송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

예

```
&4 // Modally address C.S.4, will access C.S.4's Q-vars
Q1=17.5 // Set value of Q1 (set quantity 1, spacing 1) to 17.5
Q100..102=P1+5 // Set value of Q100, Q101, Q102 to (P1+5)
Q50,3=-7 // Set value of 3 Q-vars, starting at Q50 (spacing 1) to -7
Q50,3,3=sqrt(P25) // Set value of 3 Q-vars, starting at Q50, spacing 3
// (Q50, Q53, Q56) to the square root of P25
```

r

기능: 모션 프로그램을 실행함

범위: 좌표계 고유

구문: **r, run**

Power PMAC 은 **r** 명령으로 지정된 좌표계의 계속되는 모션 프로그램의 실행을 좌표계의 프로그램 카운터의 현재 포인트에서 시작시킵니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계에 대해 프로그램의 실행을 시작합니다. 좌표계 목록이 있는 경우에 목록 내의 모든 좌표계에 대해 프로그램의 실행을 시작합니다.

프로그램 카운터는 보통 **b{상수}** 명령을 사용하여 초기 설정됩니다. **s, q** 또는 **h** 명령 등으로 프로그램의 실행을 일시 정지한 경우에는 실행이 정지된 포인트에서 재개됩니다.

좌표계의 이 명령이 Power PMAC 을 통해 수용되려면 좌표계가 적절한 상태여야 합니다. 적절한 상태가 아닌 경우에 Power PMAC 은 이 명령을 에러로 거부합니다. 이 명령이 수용되려면 좌표계가 다음 조건을 유지해야 합니다.

- 선택된 모션 프로그램(및 선택된 레이블)이 존재하고 유효할 것
- 좌표계의 위치 결정축에 할당된 모든 모터가 기동 중일 것($[Motor[x].ServoCtrl] > 0$)
- 좌표계의 위치 결정축에 할당된 모든 모터가 유효하며 클로즈드 루프일 것
- 좌표계의 위치 결정축에 할당된 모터에 양방향 오버트래블 리미트가 설정되지 않을 것
- 축 정의 구문이 사용된 경우에 Power PMAC 에 의한 현재 모터 위치에서 모든 시작축 위치 계산이 가능할 것($[Coord[x].Csolve] = 1$) 키네마틱스 정의가 사용된 경우($\#x > I$)), 좌표계에 대해 유효한 서브루틴이 존재할 것
- $[Coord[x].HomeRequired] = 1$ 인 경우에 좌표계의 위치 결정축에 할당된 모든 모터가 정상적인 원점 서치 동작 또는 절대 위치 읽기에 따라 위치 참조를 확립했을 것
- 프로그램된 포인트가 아닌 일시 정지 위치에서 재기동하는 경우에 모든 모터 위치가 최초의 정지 및 일시 정지가 명령된 위치와 같을 것

좌표계의 위치 결정축에 모터가 할당되지 않은 상태에서 모션 프로그램을 실행할 수 있지만 이는 아주 고속 리허설 모드에서 실행됩니다.

이 명령의 생략형(**r**)은 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**run**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012년 3/4 분기에 출시된 V1.5의 새 기능)가 1로 설정된 경우에는 이 명령의 비생략형(**run**)을 사용해야 합니다. 생략형(**r**)은 올바르지 않은 명령이며 에러 20으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

동등한 버퍼 프로그램의 다이렉트 명령은 **run** 입니다.

예

r	// Start program execution in addressed C.S.
b5r	// Point to beginning of Prog 5 and start program
	// execution in addressed C.S.
&2r	// Address C.S.2 and start program execution there
&1,3,5r	// Start program execution in C.S. 1, 3, and 5
&*r	// Start program execution in all C.S.

R{데이터}

기능: 'R' 변수값을 통지함

범위: 통신 스레드 고유

구문: **R{데이터}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.

Power PMAC 은 **R{데이터}** 명령으로 통신 스레드가 지정된 로컬 'R' 변수의 현재값을 통지합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~8,191). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

각 통신 스레드에는 자신의 'R' 변수의 세트가 있습니다. 또한 각 좌표계에는 모션 프로그램에서 사용하는 좌표계 자체의 세트가 있으며 각 PLC 자체에도 세트가 있습니다. 'R' 변수의 모든 세트는 각각 독립적입니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

echo0	// Include variable names in query responses
R1	// Query value of variable R1
R1=17.5	// Power PMAC response of the value of R1
R(P1)	// Query value of R-variable numbered by P1
R17=3	// Power PMAC response of the value of R17
R(P1+P17)	// Query value of R-variable numbered by P1+P17
R20=-5.35	// Power PMAC response of the value of R20
echo7	// Do not include variable names in query responses
R1	// Query value of variable R1
17.5	// Power PMAC response of the value of R1

R{데이터}={식}

기능: 값을 'R' 변수에 할당함

범위: 통신 스레드 고유

구문: **R{데이터}={식}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 변수 번호를 나타내는 소괄호 내의 수식입니다.
- **{식}**은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC 은 **R{데이터}={식}** 명령으로 통신 스레드가 지정된 로컬 'R' 변수에 부호의 오른쪽 식의 값을 설정합니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~8,191). 유효한 변수를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 에러로 거부됩니다.

명령이 Power PMAC 에 송신되고 프로그램 버퍼가 오픈되면 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

각 통신 스레드에는 자신의 'R' 변수의 세트가 있습니다. 또한 각 좌표계에는 모션 프로그램에서 사용하는 좌표계 자체의 세트가 있으며 각 PLC 자체에도 세트가 있습니다. 'R' 변수의 모든 세트는 각각 독립적입니다.

예

R1=17.5	// Set variable R1 to 17.5
R(P1)=3	// Set R-variable numbered by P1 to 3
R(P1+P17)=5*sqrt(P100)	// Set R-var numbered by (P1+P17) to expression value

R{변수 목록}

기능: 목록의 'R' 변수값을 통지함

범위: 통신 스레드 고유

구문: **R{변수 목록}[:{상수}]**

여기에서

- **{변수 목록}**은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.
 - o **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작되고 2 번째 정수값(최초보다 큰 번호여야 합니다)으로 번호를 할당된 변수로 종료하는 연속적인 번호의 변수 범위를 지정합니다.
 - o **{상수}[{상수}]{상수}**에서 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- 옵션의 **{상수}**는 응답 행마다 통지된 값의 수를 지정하는 양의 정수입니다. 여기서 값을 지정하지 않는 경우에 응답 행마다 하나의 값만 통지됩니다. 값을 지정하지 않고 사용할 수 있는 것은 목록이 연속된 수의 범위로 지정된 경우에 한합니다.

Power PMAC 은 **R{변수 목록}** 명령으로 통신 스레드 목록 내의 로컬 'R' 변수의 현재값을 통지합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC に송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

각 통신 스레드에는 자신의 'R' 변수의 세트가 있습니다. 또한 각 좌표계에는 모션 프로그램에서 사용하는 좌표계 자체의 세트가 있으며 각 PLC 자체에도 세트가 있습니다. 'R' 변수의 모든 세트는 각각 독립적입니다.

echo{상수} 명령으로 설정되는 통신 스레드의 에코 모드 파라미터 비트 1(값 2)은 변수의 이름을 응답에 기재할지 여부를 결정합니다. 비트가 0 인 경우에 이름을 기재합니다. 비트가 1 인 경우에 이름은 기재되지 않습니다.

예

```

R1 // Query value of variable R1 (set quantity 1, spacing 1)
R1=17.5 // Power PMAC response of the value of R1

R100..102 // Query value of R100, R101, R102
R100=-4 // Power PMAC response of the value of R100
R101=3.14159 // Power PMAC response of the value of R101
R102=0 // Power PMAC response of the value of R102

R50,3 // Query value of 3 R-vars, starting at R50 (spacing 1)
R50=-5.35 // Power PMAC response of the value of R50
R51=7.12 // Power PMAC response of the value of R51
R52=376452 // Power PMAC response of the value of R52

R50,3,3 // Query value of 3 R-vars, starting at R50, spacing 3
R50=-5.35 // Power PMAC response of the value of R50
R53=99 // Power PMAC response of the value of R53
R56=-0.002 // Power PMAC response of the value of R56

R1..10:5 // Query value of R1 - R10, 5 values per response line
R1=7,3,5,2,11 // Power PMAC response of the value of R1 - R5
R6=8,33,4,9,-2 // Power PMAC response of the value of R6 - R10

```

R{변수 목록}={식}

기능: 목록의 'R' 변수값을 설정함

범위: 통신 스레드 고유

구문: **R{변수 목록}={식}**

여기에서

- **{변수 목록}**은 1 개 이상의 변수 세트를 지정합니다. 목록은 다음 2 개의 형식 중 1 개를 취할 수 있습니다.
 - o **{상수}..{상수}**에서는 최초의 정수값으로 번호를 매긴 변수로 시작되고 2 번째 정수값(최초보다 큰 번호여야 합니다)으로 번호를 할당된 변수로 종료하는 연속적인 번호의 변수 범위를 지정합니다.
 - o **{상수}[, {상수}[, {상수}]**에서 최초의 정수값은 번호를 할당된 변수로 시작하는 변수 세트를 지정합니다. 이 경우에 옵션의 2 번째 정수값은 그 세트 내의 변수량을 지정하고(명시적으로 지정되지 않은 경우에는 양은 1 이 됩니다), 옵션의 3 번째 정수값은 세트 수의 간격을 지정합니다(명시적으로 지정되지 않은 경우에는 간격은 1 이 됩니다).
- **{식}**은 지정된 변수에 할당되는 값을 포함한 수식입니다.

Power PMAC 은 **R{변수 목록}={식}** 명령으로 통신 스레드가 지정된 목록의 로컬 'R' 변수에 부호의 오른쪽 식의 값으로 설정합니다. 목록은 시작과 종료 번호가 지정된 연속된 번호를 매긴 변수의 세트 또는 시작 변수 번호로 지정된 세트 내의 변수의 수 그리고 세트 내의 변수의 간격이 지정된 등간격의 변수 세트 중 하나로 할 수 있습니다. 목록으로 지정된 모든 변수 번호는 유효한 변수를 참조해야 합니다. 유효한 변수가 아닌 경우에는 명령 전체가 에러로 거부됩니다.

명령이 Power PMAC に송신되어 프로그램 버퍼가 오픈되면 목록 내에 1 개 이상의 변수가 있는 경우에 명령은 에러로 거부됩니다. 목록 내에 단일 변수만 있는 경우에는 명령은 다음의 실행을 위해 버퍼에 저장됩니다.

각 통신 스레드에는 자신의 'R' 변수의 세트가 있습니다. 또한 각 좌표계에는 모션 프로그램에서 사용하는 좌표계 자체의 세트가 있으며 각 PLC 자체에도 세트가 있습니다. 'R' 변수의 모든 세트는 각각 독립적입니다.

예

R1=17.5	// Set value of R1 (set quantity 1, spacing 1) to 17.5
R100..102=P1+5	// Set value of R100, R101, R102 to (P1+5)
R50,3=-7	// Set value of 3 R-vars, starting at R50 (spacing 1) to -7
R50,3,3=sqrt(P25)	// Set value of 3 R-vars, starting at R50, spacing 3
	// (R50, R53, R56) to the square root of P25

reboot

기능: 컨트롤러와 컴퓨터를 완전히 재기동함

범위: 글로벌

구문: **reboot**

reboot 명령으로 Power PMAC 의 컴퓨터가 완전히 재기동합니다. 컴퓨터에 대한 이 명령의 실행은 전원 스위치의 ON/OFF 와 같습니다. 액션에는 플래시 메모리에서 활성 RAM 으로의 운영체제의 리로드가 포함됩니다. Power PMAC 의 제어 어플리케이션 액션은 \$\$\$ 리셋 명령 액션과 동등합니다.

Power PMAC 은 이 명령의 실행 시작 시점에서 인터페이스 ASIC 를 강제로 리셋 모드로 하고 모든 출력값을 제로로 설정합니다. 재기동 처리 마지막에 저장된 설정값의 리로드가 시작되면 컴퓨터는 운영 모드가 됩니다.

이 명령 실행 중에 Power PMAC 은 마지막에 저장된 컨트롤러의 설정값(설정 데이터 구조체 요소, 테이블, 프로그램 등)을 비휘발성 플래시 메모리에서 활성 메모리(RAM)로 복사합니다. *이는 해당 명령을 실행하면 플래시 메모리에 저장되지 않은 Power PMAC 의 활성 메모리 내의 설정 정보가 소실된다는 것을 의미합니다.*

이 명령에 따라 저장 설정 요소 중 마지막에 저장된 값이 활성 레지스터에 처음에 복사되고 다음에 저장된 프로젝트 파일에서 명령이 실행됩니다. 해당 파일의 온라인 명령으로 설정 요소의 저장된 값을 덮어써서 설정 요소에 기록할 수 있습니다.

이 명령을 통해 Power PMAC 어플리케이션은 즉시 전원 셋업/리셋 사이클에 들어갈 수 있으므로 이 명령에 대해 응답 문자를 반환하지 않습니다.

resetverbose

기능: Power PMAC의 버보스 응답 모드를 종료함

범위: 글로벌

구문: **resetverbose**

Power PMAC은 **resetverbose** 명령으로 버보스 응답 모드를 종료하고 표준 응답 모드로 되돌 수 있습니다. 버보스 응답 모드란 Power PMAC이 **save** 명령 등 복잡한 조작의 각 순서를 통지하는 디버그 모드입니다. 실제 어플리케이션에서는 거의 사용하지 않습니다.

setverbose 온라인 명령을 사용하면 Power PMAC은 버보스 응답 모드가 됩니다.

resume plc

기능: 지정된 PLC 프로그램의 실행을 재개함

범위: 글로벌

구문: **resume plc {목록}**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수 또는 시작 상수에서 종료 상수 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 모든 상수는 0~31까지의 범위이며 실행을 재개하는 PLC 프로그램을 지정합니다.

Power PMAC은 **resume plc** 명령으로 보통 우선 순위와 타이밍으로 지정된 PLC 프로그램의 실행을 허가합니다. **pause plc** 명령을 사용하면 프로그램 도중에 실행이 정지되더라도 실행을 정지한 장소에서 시작됩니다.

마찬가지로 **enable plc** 명령은 PLC 프로그램의 실행을 재개하기 위해 사용하지만 이 명령은 프로그램 선두 이외의 다른 장소에서 실행이 정지되더라도 프로그램 선두에서 PLC 프로그램의 실행을 재개할 수 있습니다.

명령으로 지정된 PLC 프로그램이 Power PMAC에 존재하지 않는 경우에 에러가 통지되지 않습니다. 명령은 명령으로 지정된 다른 PLC 프로그램에 작용합니다.

예

```
resume plc 1           // Resume execution of PLC 1
resume plc 2,4,6        // Resume execution of PLCs 2, 4, and 6
resume plc 7..10        // Resume execution of PLCs 7, 8, 9, and 10
resume plc 11,13..16,20 // Resume execution of PLCs 11, 13, 14, 15, 16, and 20
```

rotfree

기능: 로터리 버퍼가 실행하지 못한 행의 수를 통지함

범위: 좌표계 고유

구문: **rotfree**

Power PMAC 은 **rotfree** 명령으로 지정된 좌표계의 로터리 모션 프로그램 버퍼 내에 있는 아직 실행되지 않은 프로그램 행의 수를 통지합니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다.

이 명령은 프로그램 행의 다음 세트를 로터리 버퍼에 로드하는 타이밍을 결정하는 경우에 도움이 됩니다. 보통 사용자가 로터리 버퍼를 실행하지 않은 행의 수가 2개의 최소치 사이가 되도록 어플리케이션의 규칙을 설정합니다. 예를 들어 규칙이 50에서 100을 실행하지 않은 행 버퍼로 유지하는 경우에 사용자는 100 행을 로드할 수 있습니다. 그 후 프로그램이 실행 중일 때에 실행하지 않은 행의 나머지가 50 미만이라는 응답이 돌아 올 때까지 **rotfree** 명령이 반복 전송됩니다. 이 시점에서 실행하지 않은 행의 수가 100 행에 근접할 때까지 프로그램행이 계속 로드되어 프로세스가 반복됩니다.

run

기능: 모션 프로그램을 실행함

범위: 좌표계 고유

구문: **r, run**

run 명령은 **r** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정한 좌표계의 계속되는 모션 프로그램의 실행을 좌표계 프로그램 카운터의 현재 포인트에서 시작할 수 있습니다. [Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다. [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

s

기능: 모션 프로그램의 싱글 스텝을 실행함

범위: 좌표계 고유

구문: **s, step**

Power PMAC 은 **s** 명령으로 지정된 좌표계의 싱글 스텝의 모션 프로그램 실행을 좌표계의 프로그램 카운터의 현재 포인트에서 시작할 수 있습니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계에 대해 프로그램의 실행을 시작합니다. 좌표계 목록이 있는 경우에 목록 내의 모든 좌표계에 대해 프로그램의 실행을 시작합니다.

이 명령에 응답하는 좌표계의 정확한 동작은 저장 설정 요소 `[Coord[x].StepMode]`의 설정에 따라 달라집니다(이 요소는 2014 년 1/4 분기에 출시된 V1.6 펌웨어의 신기능이며 동작은 항상 이 요소가 0으로 설정된 상태와 같습니다).

좌표계가 이미 모션 프로그램을 실행 중인 경우에 이 명령이 전송되면 프로그램은 싱글 스텝 모드가 됩니다. `[Coord[x].StepMode]`가 0으로 설정된 경우에 또 다른 한 동작이 계산되어 동작 실행은 이 동작이 끝날 때 정지합니다.. 이 경우에 **q** 명령 처리와 비슷하지만 **q** 와 다른 점은 추가적인 동작이 실행된다는 점입니다. `[Coord[x].StepMode]`가 0보다 큰 값으로 설정된 경우에 또 다른 한 동작이 계산되어 동작 실행은 이 직전에 계산된 동작이 끝날 때 정지합니다.

프로그램 실행이 이미 싱글 스텝 모드에서 정지된 경우에 `[Coord[x].StepMode]`가 0으로 설정되어 있으면 스텝 명령에 따라 하나의 동작이 완료될 때까지 프로그램은 실행을 계속합니다. `[Coord[x].StepMode]`가 0보다 큰 값으로 설정된 경우에 하나의 프로그램행이 실행됩니다(`[Coord[x].StepMode]`가 0 이외의 값으로 설정되고 2D 공구 반경 보정이 유효한 경우에는 값에 따라 동작이 조금 다릅니다).

`[Coord[x].StepMode]` 설정과 관계없이 싱글 스텝 실행 모션 프로그램에서 **bstart** 명령이 검출되면 포함된 동작이나 프로그램행의 수에 관계없이 모션 프로그램은 다음 **bstop** 명령까지 동작을 속행하고 정지합니다.

프로그램 카운터는 보통 **b{상수}** 명령을 사용하여 초기 설정됩니다. **s**, **q** 또는 **h** 명령 등으로 프로그램의 실행을 일시 정지한 경우에는 실행이 정지된 포인트에서 재개됩니다.

프로그램의 실행 중에 **bstart**(블록 스타트) 명령이 검출된 경우에 **bstop**(블록 스톱) 명령이 검출될 때까지 싱글 스텝의 실행이 속행됩니다. 그 동안의 프로그램 행이나 동작 명령의 수 또는 `[Coord[x].StepMode]`의 설정은 이 프로그램의 실행과는 관계없습니다.

모션 프로그램의 실행은 해당 명령 뒤에 정지 중인 동안은 일시 정지하지만 논리적으로는 아직 프로그램 내에 있으며 이 시점에서 실행을 재개할 수 있습니다. 일시 정지 후에도 프로그램의 실행을 재개하지 않는 경우에는 **a** 명령을 사용하여 프로그램의 실행을 완전히 Abort 해야 합니다.

좌표계의 이 명령이 Power PMAC 을 통해 수용되려면 좌표계가 적절한 상태여야 합니다. 적절한 상태가 아닌 경우에 Power PMAC 은 이 명령을 에러로 거부합니다. 이 명령이 수용되려면 좌표계가 다음 조건을 유지해야 합니다.

- 선택된 모션 프로그램(및 선택된 레이블)이 존재하고 유효할 것
- 좌표계의 위치 결정축에 할당된 모든 모터가 기동 중일 것(`[Motor[x].ServoCtrl] > 0`)
- 좌표계의 위치 결정축에 할당된 모든 모터가 유효하며 클로즈드 루프일 것
- 좌표계의 위치 결정축에 할당된 모터에 양방향 오버트래블 리미트가 설정되지 않을 것
- 축 정의 구문이 사용된 경우에 Power PMAC 에 의한 현재 모터 위치에서 모든 시작축 위치 계산이 가능할 것(`[Coord[x].Csolve] = 1`) 키네마틱스 정의가 사용된 경우(**#x->I**)), 좌표계에 대해 유효한 서브루틴이 존재할 것
- `[Coord[x].HomeRequired] = 1` 인 경우에 좌표계의 위치 결정축에 할당된 모든 모터가 정상적인 원점 서치 동작 또는 절대 위치 읽기에 따라 위치 참조를 확립했을 것

- 프로그램된 포인트가 아닌 일시 정지 위치에서 재기동하는 경우에 모든 모터 위치가 최초의 정지 및 일시 정지가 명령된 위치와 같을 것

좌표계의 위치 결정축에 모터가 할당되지 않은 상태에서 모션 프로그램을 실행할 수 있지만 이는 아주 고속 리허설 모드에서 실행됩니다.

이 명령의 생략형(**s**)는 터미널 모드에서 입력할 때 도움이 됩니다. 비생략형(**step**)은 프로그램에서 명령 전송 시에 명확성과 자기 문서화가 필요한 경우에 사용합니다.

저장 설정 요소 [Sys.NoShortCmds](2012년 3/4 분기에 출시된 V1.5의 새 기능)가 1로 설정된 경우에는 이 명령의 비생략형(**step**)을 사용해야 합니다. 생략형(**s**)은 올바르지 않은 명령이며 에러 20으로 거부됩니다. 이 설정에 따라 이 명령의 잘못된 전송을 최소한으로 할 수 있습니다.

동등한 버퍼 프로그램의 다이렉트 명령은 **step**입니다.

예

s	// Start single-step execution in addressed C.S.
b5s	// Point to beginning of Prog 5 and start single-step
	// execution in addressed C.S.
&2s	// Address C.S.2 and start single-step execution there
&1, 3, 5s	// Start single-step execution in C.S. 1, 3, and 5
&*s	// Start single-step execution in all C.S.

save

기능: 셋업의 설정을 비휘발성 메모리에 복사함

범위: 글로벌

구문: **save**

Power PMAC은 **save** 명령으로 활성인 메모리(RAM 및 ASIC 메모리 맵 레지스터)에서 비휘발성 메모리(플래시)에 셋업 설정을 복사합니다. 이에 따라 전원 끄기 또는 리셋 시에도 정보를 유지할 수 있습니다. 다음의 전원 셋업 또는 리셋 시에 이 정보가 비휘발성 메모리에서 활성 메모리로 복사됩니다.

save 명령으로 플래시 메모리에 저장되는 프로그램은 RAM의 파일 구조에서 플래시 메모리에 복사되므로 그 구조의 일부로 Power PMAC의 활성인 메모리에 다운로드되어야 합니다. IDE 프로젝트 다운로드는 자동으로 이를 실행합니다.

setverbose

기능: Power PMAC을 버보스 응답 모드로 함

범위: 글로벌

구문: **setverbose**

setverbose 명령을 사용하면 Power PMAC을 버보스 응답 모드로 설정할 수 있습니다. 버보스 응답 모드란 Power PMAC이 **save** 명령 등 복잡한 조작의 각 순서를 통지하는 디버그 모드입니다. 실제 어플리케이션에서는 거의 사용하지 않습니다.

resetverbose 온라인 명령을 사용하면 Power PMAC의 버보스 응답 모드가 종료되고 표준 응답 모드로 되돌릴 수 있습니다.

예

```
setverbose // Put into verbose response mode
save // Store project to flash memory
Changing echoMode to zero during Save operation
Successful:SaveConfiguration using /var/ftp/usrflash/Project/Configuration/pp_save.cfg

Successful:SaveCustomConfiguration using
/var/ftp/usrflash/Project/Configuration/pp_custom_save.tpl

SaveToFlash:Do NOT Power off until Finished!!!

SaveToFlash:cp

SaveToFlash:sync()

SaveToFlash:mount

SaveToFlash:Finish SAVING to Flash.

Save Complete

Restoring echoMode to original 0 value

resetverbose // Take out of verbose response mode
save // Store project to flash memory
Changing echoMode to zero during Save operation
Save Complete

Restoring echoMode to original 0 value
```

size

기능: 버퍼의 합계 메모리 크기를 통지함

범위: 글로벌

구문: **size**

Power PMAC은 **size** 명령으로 RAM에 있는 여러 사용자 메모리 버퍼에서 사용 완료 또는 미사용한 현재의 메모리 바이트 수를 통지합니다. 관련된 **free** 명령은 이러한 각 메모리 버퍼에 대해 미사용한 메모리의 합계 바이트 수를 반환합니다. 버퍼 크기는 IDE 프로젝트 매니저의 [pp_proj.ini] 파일의 지시문으로 설정됩니다.

예

```
size // Query total buffer memory
Program Buffer = 16777216 // Power PMAC response
User Buffer = 1048576
Table Buffer = 1048576
Lookahead Buffer = 16777216
```

start[{{상수}}]

기능: 지정된 모션 프로그램의 프로그램 카운터를 포인트하여 실행함

범위: 좌표계 고유

구문: **start[{{상수}}]**

여기에서

- **{{상수}}**는 모션 프로그램의 수를 지정하는 음의 값이 아닌 정수입니다.

start[{{상수}}] 명령으로 지정된 좌표계는 수치가 지정되지 않은 경우에는 현재 또는 지정된 모션 프로그램의 최초에 프로그램 카운터를 설정하고 그 프로그램의 계속적인 실행을 시작합니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다.

start[{{상수}}] 명령은 명령 시퀀스 **b[{{상수}}]r**(시작과 실행)과 같습니다.

좌표계의 이 명령이 Power PMAC 을 통해 수용되려면 좌표계가 적절한 상태여야 합니다. 적절한 상태가 아닌 경우에 Power PMAC 은 이 명령을 에러로 거부합니다. 이 명령이 수용되려면 좌표계가 다음 조건을 유지해야 합니다.

- 선택된 모션 프로그램(및 선택된 레이블)이 존재하고 유효할 것
- 좌표계의 위치 결정축에 할당된 모든 모터가 기동 중일 것([Motor[x].ServoCtrl] > 0)
- 좌표계의 위치 결정축에 할당된 모든 모터가 유효하며 클로즈드 루프일 것
- 좌표계의 위치 결정축에 할당된 모터에 양방향 오버트래블 리미트가 설정되지 않을 것
- 축 정의 구문이 사용된 경우에 Power PMAC 에 의한 현재 모터 위치에서 모든 시작축 위치 계산이 가능할 것([Coord[x].Csolve] = 1) 키네마틱스 정의가 사용된 경우(#x->l), 좌표계에 대해 유효한 서브루틴이 존재할 것
- [Coord[x].HomeRequired] = 1 인 경우에 좌표계의 위치 결정축에 할당된 모든 모터가 정상적인 원점 서치 동작 또는 절대 위치 읽기에 따라 위치 참조를 확립했을 것
- 프로그램된 포인트가 아닌 일시 정지 위치에서 재기동하는 경우에 모든 모터 위치가 최초의 정지 및 일시 정지가 명령된 위치와 같을 것

좌표계의 위치 결정축에 모터가 할당되지 않은 상태에서 모션 프로그램을 실행할 수 있지만 이는 아주 고속 리허설 모드에서 실행됩니다.

step

기능: 모션 프로그램의 싱글 스텝을 실행함

범위: 좌표계 고유

구문: **s, step**

step 명령은 **s** 명령의 비생략형입니다. Power PMAC 은 이 명령으로 지정한 좌표계의 싱글 스텝 모션 프로그램의 실행을 좌표계 프로그램 카운터의 현재 포인트에서 시작할 수 있습니다.[Sys.NoShortCmds]가 기본값인 0 으로 설정된 경우에 이 명령 중 하나의 형태도 사용할 수 있습니다.이 [Sys.NoShortCmds]가 1 로 설정된 경우에 비생략형만 사용할 수 있습니다.

이 명령에 관한 자세한 내용은 명령의 생략형 설명을 참조하십시오.

step plc

기능: 지정된 PLC 프로그램의 싱글 스텝을 실행함

범위: 글로벌

구문: **step plc {목록}**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수 또는 시작 상수에서 종료 상수 사이를 2 개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 모든 상수는 0~31 까지의 범위로 싱글 스텝을 실행하는 PLC 프로그램을 지정합니다.

Power PMAC 은 **step plc** 명령으로 지정된 PLC 프로그램의 싱글 스텝 실행을 허가합니다. **pause plc** 명령을 사용하면 프로그램 도중에 실행이 정지된 경우에도 지정된 PLC 의 실행이 정지한 장소에서 시작됩니다. 지정된 PLC 프로그램이 명령된 시점에서 계속 실행 모드였던 경우에 계속 실행은 정지됩니다.

싱글 스텝 모드에서는 단일 프로그램 행의 명령만 명령에 대한 응답으로 실행됩니다. 단일 프로그램 행을 실행할 수 없는 경우에는 실행이 중단됩니다. 싱글 스텝 모드는 보통 디버그 목적으로 사용됩니다.

resume plc 명령은 마찬가지로 PLC 프로그램의 실행을 재개하기 위해 사용하지만 실행이 정지된 장소에서 PLC 프로그램의 계속 실행을 재개할 수 있습니다.

명령으로 지정된 PLC 프로그램이 Power PMAC 에 존재하지 않는 경우에 에러가 통지되지 않습니다. 명령은 명령으로 지정된 다른 PLC 프로그램에 작용합니다.

예

```
step plc 1           // Execute single line of PLC 1
step plc 2,4,6       // Execute single line of PLCs 2, 4, and 6
step plc 7..10       // Execute single line of PLCs 7, 8, 9, and 10
step plc 11,13..16,20 // Execute single line of PLCs 11, 13, 14, 15, 16, and 20
```

stop

기능: 모션 프로그램의 실행을 정지하고 프로그램 카운터를 선두에 리셋함

범위: 좌표계 고유

구문: **stop**

Power PMAC 은 **stop** 명령으로 지정된 좌표계의 모션 프로그램의 실행을 정지하고 프로그램에 따라 이미 계산된 동작을 종료시킵니다. 또한 좌표계의 프로그램 카운터를 모션 프로그램의 선두(최상위)에 리셋합니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다.

string{상수}

기능: 지정된 텍스트 문자열의 내용을 통지함

구문: **string{상수}**

여기에서

- **{상수}**는 음의 값이 아닌 정수이며 통지하는 문자열의 최초 문자의 사용자 공유 메모리 시작부터 바이트 오프셋을 설정합니다.

Power PMAC 은 **string{상수}** 명령으로 지정된 문자열 변수의 현재값 내용을 텍스트로 통지합니다. 변수는 최초의 사용자 공유 메모리 버퍼에서 지정된 오프셋으로 시작되고 Null 종단 문자(바이트값 0)가 검출될 때까지 계속됩니다. 값은 ASCII 텍스트로 통지됩니다.

다른 대부분의 쿼리 명령과는 달리 Power PMAC 의 응답은 텍스트의 응답 마지막에 캐리지 리턴 문자를 자동으로 추가하지 않습니다.

예

string250	// Query string starting at 250
Overpressure Error	// Power PMAC response

t

기능: 현재 실행 중인 동작 축의 대상 위치를 통지함

범위: 좌표계 고유

구문: **t**

Power PMAC 은 **t** 명령으로 현재 실행 중인 동작 축의 대상 위치를 통지합니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계의 축 대상 위치를 통지합니다. 좌표계 목록이 있는 경우에 목록 내 모든 좌표계의 대상 위치를 통지합니다.

저장 설정 요소 [Coord[x].TPSize]를 0 이상의 동작 계산 시와 동작 실행 시 사이의 모든 동작 위치를 저장하기 위한 충분한 값으로 설정하고 목표 위치의 버퍼를 유효로 해야 합니다. 저장 설정 요소 [Coord[x].TPCoords]에 따라 지정된 축의 위치만 통지됩니다.

대상 위치는 축의 이름을 나타내는 문자 뒤에 수치가 있는 텍스트 형식으로 통지됩니다. 위치는 동작이 계산된 시점에서 유효한 축 변환을 사용하여 계산되며 통지됩니다. 저장 설정 요소 `[Coord[x].Ndisplay]`를 1로 설정하면 Status 요소 `[Coord[x].Nsync]`의 현재값이 응답 선두에서 문자 'N' 다음에 통지됩니다. 이 값은 레이블에 이어지는 프로그램에서 동작 실행 시작 시 직후의 동기 라인 레이블값에 자동으로 설정됩니다. 따라서 레이블이 사용되었을 때의 특정한 동작을 간단하게 식별할 수 있습니다.

공구 직경 보정이 동작에 대해 활성이며 축 보정에서 제로 이외의 오프셋이 있는 경우에 X, Y 또는 Z 위치는 2개의 값으로 통지됩니다. 축의 이름을 나타내는 문자 다음에 프로그램된 값이 이어지고 그 다음에 콜론, 마지막에 공구 보정에 대해 자동으로 계산된 축 오프셋 순입니다. 보정이 다음 동작에 대한 블렌드이며 외측 코너 주위에 원호 동작을 추가한 경우에 `[Coord[x].CCtrl]`의 비트 0(값 1)이 기본값인 0으로 설정되면 원호의 마지막 오프셋이 통지됩니다. 이 비트가 1로 설정되면 원호의 최초 오프셋이 통지됩니다.

`[Coord[x].TPSize]`가 0으로 설정된 경우에 이 명령이 전송되면 Power PMAC은 에러 메시지 'No targets defined'를 반환합니다. `[Coord[x].TPSize]`가 0 이상으로 설정되고 그 값이 계산과 실행 간의 모든 동작 버퍼에 충분하지 않은 경우에는 잘못된 값이 통지될 가능성이 있습니다.

t 온라인 명령은 기본적으로 **tread** 버퍼 프로그램 명령과 같은 계산을 실행합니다. 또한 텍스트 문자열을 반환하기 위해 버퍼 프로그램 명령과 마찬가지로 축의 값을 'D' 변수에 넣습니다.

예

```
Coord[1].Ndisplay = 0           // Do not report N label
Coord[1].TPCoords = $1C5       // Show A,C,X,Y,Z data
&1t
A20 C-12.25 X33.7223 Y-5.78 Z0

Coord[1].TPCoords = $1C0       // Show X,Y,Z data
&1t
X33.7223 Y-5.78 Z0

Coord[1].Ndisplay = 1          // Report N label
&1t
N1260 X45.1:3.72 Y-8:-1.34 Z0
```

type

기능: Power PMAC의 시스템 타입을 통지함

범위: 글로벌

구문: **type**

Power PMAC은 **type** 명령으로 현재의 Power PMAC 타입을 통지합니다.

예

```
type           // Query version number of firmware
PWR PMAC UMAC  // Power PMAC response
```

undefine

기능: 좌표계의 정의를 클리어함

범위: 좌표계 고유

구문: **undefine**

Power PMAC 은 **undefine** 명령으로 지정한 좌표계의 모든 축 정의를 클리어합니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 모터가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다.

이 명령은 다른 좌표계의 축 정의에 영향을 미치지 않습니다. 이 명령은 축 정의의 새로운 세트를 준비하는 경우에 도움이 됩니다.

각 좌표계의 모든 축 정의를 클리어하려면 **undefine all** 명령을 사용합니다.

undefine all

기능: 모든 좌표계의 정의를 클리어함

범위: 글로벌

구문: **undefine all**

Power PMAC 은 **undefine all** 명령으로 모든 좌표계의 모든 축 정의를 클리어합니다. 이 명령은 축 정의의 새로운 세트를 준비하는 경우에 도움이 됩니다.

지정된 좌표계의 축 정의 모두를 클리어하려면 **undefine** 명령을 사용합니다.

v

기능: 현재 속도값을 통지함

범위: 모터 또는 좌표계 고유

구문: **v**

Power PMAC 은 **v** 명령으로 지정된 모터 또는 좌표계의 현재 속도값을 통지합니다. 모터 목록 또는 좌표계 목록이 없는 경우에는 현재 주소 지정된 모터의 현재 속도값을 통지합니다. 모터 목록이 있는 경우에는 목록 내 모든 모터의 현재 속도값을 통지합니다. 좌표계 목록이 있으면 목록 내의 모든 좌표계의 모든 활성 축의 현재 속도값을 통지합니다.

여러 모터 목록 또는 여러 좌표계 목록을 지정해도 후속 모터 고유의 명령 또는 좌표계 고유의 명령으로 모달에 주소 지정된 모터 또는 좌표계는 변경되지 않습니다.

모터 속도를 통지하는 경우에 속도는 서보 사이클당 정의된 모터 단위로 표시됩니다. 축 속도를 통지하는 경우에 데이터 구조체 요소 **[Coord[x].FeedTime]**(보통 초 또는 분)을 통해 지정되도록 속도는 좌표계의 시간 단위당 스케일링된 축 위치 단위로 표시됩니다.

각 축에 통지된 값은 축의 이름 (1 개 또는 2 개의 영문자) 앞에 설명합니다. 어느 경우에도 실행 중인 필터링된 평균 속도값이 통지됩니다.

그 좌표계의 축에 모터가 할당되지 않은 좌표계에서 통지가 요구되면 Power PMAC 은 문자열 'No Motors'를 반환합니다. 좌표계의 축 정의 셋팅 또는 포워드 키네마틱스 서브루틴으로 축 데이터를 적절하게 계산할 수 없는 경우에 Power PMAC 은 문자열 'No Solution'을 반환합니다.

예

v	// Query actual velocity of presently addressed motor
1.21	// Power PMAC response in counts per msec
#1..4v	// Query actual velocities of Motors 1 - 4
3.62 -7.83 0 601	// Power PMAC response in counts per msec
&1v	// Query actual velocities of axes in C.S. 1
X36.2 Y-78.3 Z0 C6010	// Power PMAC response in user units

vers

기능: 펌웨어 버전을 통지함

범위: 글로벌

구문: **vers**

Power PMAC 은 **vers** 명령으로 사용 중인 펌웨어의 버전 번호를 통지합니다.

예

vers	// Query version number of firmware
1.4.0.62	// Power PMAC response

POWER PMAC 프로그램 명령 사양

이 섹션에서는 Power PMAC 의 버퍼 프로그램 명령을 설명합니다. Power PMAC 에 버퍼 프로그램 명령을 송신하면 명령은 오픈 버퍼에 입력됩니다. 프로그램이 실행되고 프로그램 시퀀스가 이 명령에 도달할 때까지 명령은 실제로 실행되지 않습니다.

abort

기능: 모션 프로그램과 동작을 Abort 함

구문: **abort**{**{목록}**}

여기에서

- **{목록}**은 씬표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.

abort 명령에 따라 주소 지정 또는 목록화된 좌표계의 모션 프로그램과 동작을 Abort 시킵니다.

좌표계 목록이 없는 경우에 현재 프로그램에 따라 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 목록 내의 좌표계를 지정해도 프로그램에 따라 모달에 주소 지정된 좌표계에는 영향을 미치지 않습니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 **[Plc[i].Ldata.Coord]**에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(**[Coord[x].Ldata.Coord]**)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

Abort 한 모터는 제어(클로즈드 루프) 정지를 위해 데이터 구조체 요소 **[Motor[x].AbortTa]** 및 **[Motor[x].AbortTs]**로 설정한 속도 또는 시간으로 감속합니다. Power PMAC 은 이 명령을 받는 즉시 지정된 좌표계 내에 있는 모든 모터의 명령 동작 프로파일에 인터럽트하여 각 모터의 개별 감속 프로파일을 계산합니다. 해당 감속 프로파일은 독립되어 있으므로 정지 시간 또는 궤적에 대해서 반드시 어떤 조건이 있는 것은 아닙니다.

V1.6 보다 이전 버전의 펌웨어에서 타임 베이스가 100%가 아닌 경우에는 모터의 감속 프로파일은 그 시점의 타임 베이스 비율로 실행됩니다. 2014 년 1/4 분기에 출시된 V1.6 펌웨어부터 **[Coord[x].AbortTimeBase]**에 따라 사용자가 최소 타임 베이스의 값을 지정할 수 있으며 모터의 감속 프로파일은 이 값으로 실행됩니다. 명령이 전송됐을 때 타임 베이스가 이 값보다도 작은 경우에 프로파일은 100% 타임 베이스로 실행됩니다.

abort 명령은 제어 정지가 가능한 경우에 긴급 정지에 대해 유효합니다(어플리케이션에 따라서는 최초 정지 시에 **Abort** 하고 보통의 요건을 충족하기 위해 지연 릴레이를 사용하여 드라이브 전원을 OFF 하는 경우도 있습니다).

모션 프로그램의 조작이 ('quit'이나 'hold' 명령 등)그 밖의 방법을 이미 보류 중인 경우에 'abort' 명령으로 프로그램의 실행을 완전히 정지시키면 다른 프로그램에서 시작하거나 Power PMAC 에 새 프로그램을 다운로드할 수 없습니다.

정지한 포인트에서 모션 프로그램 실행을 재개하는 경우에 프로그램을 정지하기 위한 **abort** 명령을 지정하지 마십시오. 보통 **abort** 명령은 프로그램한 포인트에서 모션을 정지시키거나 프로그램한 궤적에서 모션을 정지시킬 수 없습니다. **Abort** 명령에 따라 모션 프로그램의 선두에 프로그램 카운터가(최상위) 자동으로 리셋됩니다.

좌표계의 모터가 오픈 루프 유효 상태인 경우에 이 명령에 의해 루프가 클로즈하고 필요에 따라 제어된 감속으로 정지합니다. 단, 이 명령에 따라 무효인(출력 정지된) 모터가 유효가 되지 않습니다.

상당하는 온라인 명령은 생략형 **a** 를 사용하며 그 앞에 좌표계 목록이 놓인다는 점(&1,2a 등)에 주의하십시오.

예

```
abort;           // Abort motors in presently addressed C.S
abort1;          // Abort motors in C.S.1
abort3,5..7;     // Abort motors in C.S. 3, 5, 6, & 7
```

abs

기능: 절대 동작 모드 지정

구문: **abs**[{축 목록}]

여기에서

- {축 목록}**은 쉼표로 나눈 축 명칭(1 문자 또는 2 문자) 및/또는 시작 축 명칭에서 종료 축 명칭 사이를 마침표 2 개로 나타내는 연속한 축 범위입니다. 사용되는 모든 축 명칭은 알파벳 순이며 먼저 1 문자의 이름을 지정하고 그 다음에 2 문자의 이름을 지정해야 합니다.

인수를 가지지 않는 **abs** 명령에 따라 프로그램에서 명령된 좌표계의 모든 축 동작 명령 내의 이동 대상이 절대 위치로 처리됩니다. 이것이 앱솔루트 모드이며 전원 셋업 시의 기본값 상태입니다.

인수를 가지는 **abs** 명령에 따라 프로그램을 실행하는 좌표계의 지정된 축이 앱솔루트 모드가 됩니다. 그 외의 축은 변경할 수 없습니다. 이 명령 내에 공백은 사용할 수 없습니다.

abs 명령을 실행하면 32 비트 데이터 구조체 요소 **[Coord[x].IncAxes]**(인크리멘탈 모드 축)가 클리어됩니다.

예

```
abs; // Put all axes in C.S. in absolute mode
abs(X,Y); // Put X & Y axes in C.S. in abs mode, leave others
abs(V);
abs(XX,YY,ZZ,CC);
abs(A..Z);
abs(A,B,UU..ZZ);
```

abs({벡터 목록})

기능: 절대 원 중심 벡터 지정

구문: **abs({벡터 목록})**

여기에서

- **{벡터 목록}**은 쉼표로 구분된 벡터 컴포넌트명(I, J, K, II, JJ, KK) 세트 및/또는 시작 벡터 컴포넌트명에서 종료 컴포넌트명 사이를 2 개의 마침표로 나타낸 연속된 벡터 컴포넌트 범위입니다. 사용되는 모든 벡터 컴포넌트명은 알파벳 순으로 최초에 1 문자의 이름을 지정하고 그 다음에 2 문자의 이름을 지정해야 합니다.

abs({벡터 목록}) 명령에 따라 지정된 원 중심 벡터 컴포넌트가 애플루트 모드가 됩니다. 애플루트 모드는 중심 벡터 컴포넌트가 인크리멘탈 모드처럼 동작 시작점에서가 아니라 일치하는 축(I 는 X, J 는 Y, K 는 Z)의 프로그래밍 원점에서 원 중심까지 부호가 있는 거리를 지정합니다. 전원 셋업/리셋 시에 모든 벡터 컴포넌트는 인크리멘탈 모드입니다.

abs({벡터 목록}) 명령을 실행하면 좌표계의 Status 데이터 구조체 요소 [Coord[x].cdata]의 일부 비트가 지정됩니다.

목록이 없는 **abs** 명령에 따라 원 중심 벡터에 영향을 미치지 않으면서 좌표계 모든 축이 애플루트 모드가 된다는 점에 주의하십시오.

예

```
abs(I,J,K); // Put I/J/K vector set in abs mode
abs(I..K); // Put I/J/K vector set in abs mode
abs(I..KK); // Put both vector sets in abs mode
```

adisable

기능: 지연 Disable 다음에 Abort 함

범위: 좌표계 고유

구문: **adisable[{목록}]**

Power PMAC 은 **adisable** 명령으로 좌표계의 모든 모터를 지연 Disable 로 한 후에 주소 지정 또는 목록화된 좌표계 모션 프로그램과 동작을 자동으로 Abort 시킵니다.

좌표계 목록이 없는 경우에 현재 프로그램에 따라 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 목록 내의 좌표계를 지정해도 프로그램에 따라 모달에 주소 지정된 좌표계에는 영향을 미치지 않습니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 [Plc[i].Ldata.Coord]에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계([Coord[x].Ldata.Coord])를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

adisable 명령에서 최초의 처리는 **abort** 명령 처리와 같으며 좌표계의 모든 모터는 제어(클로즈드 루프) 정지를 위해 [Motor[x].AbortTa] 및 [Motor[x].AbortTs]로 설정된 속도 또는 시간으로 즉시 감속을 시작합니다.

단, 각 모터가 명령 정지([Motor[x].DesVelZero] = 1) 상태가 된 경우에 **dkill** 명령이 지정되면 모터는 자동으로 무효가 됩니다. 모터에 브레이크 출력이 지정된 ([Motor[x].pBrakeOut] > 0) 경우에 브레이크는 이 시점에서 즉시 사용되어 [Motor[x].BrakeOnDelay] 밀리초의 지연 후에 모터는 정지합니다(오픈 루프, 제로 명령 출력, 앰프 Disable).

adisable 명령에 의한 처리는 [Coord[x].AbortAllMode]가 2 로 설정된 경우의 글로벌 Abort에 해당합니다.

adisable 명령은 2014 년 1/4 분기에 출시된 V1.6 펌웨어의 새 기능입니다.



참고사항

adisable 명령의 결과는 IEC-61800-5-2 의 기계 안전 규격 카테고리 1 의 안전 정지와 실질적으로 같으나 이 규격에서는 제어 정지 후에 모터 또는 드라이브에서 전원을 차단해야 합니다. 이 규격에서 이 명령을 사용하면 제어 정지 후 버스 파워 또는 게이트 드라이버 전원 중 하나를 차단하기 위한 지연 릴레이도 필요합니다.

예

```
adisable;           // Abort and disable motors in presently addressed C.S
adisable1;          // Abort and disable motors in C.S.1
adisable3,5..7;      // Abort and disable motors in C.S. 3, 5, 6, & 7
```

{축}{데이터}{축}{데이터}...

기능: 위치 동작 명령

구문: {축}{데이터}{축}{데이터}...

여기에서

- **{축}**은 동작하는 축을 지정하는 1 문자 또는 2 문자(A, B, C, U, V, W, X, Y, Z, AA, BB, ... HH, LL, MM, ... ZZ)로 나타냅니다.

- **{데이터}**는 종료 위치 또는 거리를 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.
- **[축]{데이터}...**는 여러 축을 동시 동작시키는 옵션을 지정합니다.

이것은 기본적인 Power PMAC 동작 지정 명령입니다. 하나 이상의 축 레이블 그룹과 그에 관련된 값으로 구성됩니다. 축의 값은 (축 정의에 따라 결정된 단위로) 스케일링되고 축이 앵글루트(**abs**) 모드인 경우에 위치를 나타냅니다. 또한 축이 인크리멘탈(**inc**) 모드인 경우에 최근 명령 위치에서의 거리를 나타냅니다. 축이 지정된 순서는 관계없습니다.

이 명령은 축 동작 대상을 지정합니다. 동작 방법은 지정하지 않습니다. 그 밖의 프로그램 명령 및 파라미터에 따라 방법이 정의됩니다. 사전에 설정해야 합니다.

동작 명령이 생성하는 모션 타입은 모션 모드 및 동작 시작 시의 시스템 상태에 따라 결정됩니다.

서브루틴 또는 서브 프로그램의 호출과 같은 프로그램행 뒤에 하나 이상의 구조체 **{축}{데이터}**가 있고 루틴이 축 명칭을 지정하는 **read** 명령을 실행하는 경우에 **{데이터}**값을 인수로 서브루틴 또는 서브 프로그램에 전달합니다. 이 경우에 구조체는 **read** 명령에 따라 사용되며 동작 명령으로서는 실행되지 않습니다.

예

```
X1000
X(P1+P2)
Y(Q100+500) Z35 C(P100)
X1000 Y1000 XX2000 YY2000
A(P1) B(P2) C(P3)
X(Q1*sin(Q2/Q3)) U500
```

{축}{데이터}:{데이터}[축]{데이터}:{데이터}...

기능: 위치 및 속도 동작 명령

구문: **{축}{데이터}:{데이터}[축]{데이터}:{데이터}...**

여기에서

- **{축}**은 동작하는 축을 지정하는 1 문자 또는 2 문자(A, B, C, U, V, W, X, Y, Z, AA, BB, ... HH, LL, MM, ... ZZ)로 나타냅니다.
- **{데이터}**는 축의 종료 위치 또는 거리를 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.
- **:{데이터}**는 축의 부호가 있는 최종 속도를 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.
- **[축]{데이터}:{데이터}...**는 여러 축을 동시 동작시키는 옵션을 지정합니다.

PVT(위치, 속도, 시간) 동작 모드의 경우에 종료 위치와 속도는 각 축의 각 세그먼트로 지정됩니다. 명령은 2 개의 데이터가 콜론으로 구분된 1 개 이상 그룹의 축 레이블로 구성됩니다.

각 축의 최초 데이터는 스케일링된 종료 위치 또는 거리입니다. 위치 또는 거리를 선택하는 것은 축 모드가 앵글루트(**abs**) 모드인지 증가(**inc**) 모드인지에 따라 결정됩니다. 위치 스케일링은 축 정의 선언에 따라 결정됩니다. 두 번째 {데이터}(콜론 뒤)는 종료 속도입니다.

축 정의 구문으로 확립한 속도 단위는 스케일링된 위치 단위를 시간 단위로 나누어 저장 설정 요소 [Coord[x].FeedTime]으로 설정합니다. 여기에서 속도는 단순히 크기만이 아니라 부호가 있는 양입니다. '모션 프로그램 설명' 섹션에서 설명하는 PVT 모드의 예를 참조하십시오.

세그먼트 시간은 마지막으로 실행한 **pvt** 명령의 인수입니다.

PVT 모드에서 세그먼트에 속도가 설정되지 않은 경우에 Power PMAC 은 세그먼트의 속도를 0 으로 간주합니다.

예

```
X1000:50
Y500:-32 Z737.2:68.93
A(P1+P2):(P3) B(sin(Q1)):0
```

{축}{데이터}^^{데이터} [{축}{데이터}^^{데이터}...]

기능: 트리거까지의 동작 명령

구문: {축}{데이터}^^{데이터} [{축}{데이터}^^{데이터}...]

여기에서

- **{축}**은 동작하는 축을 지정하는 1 문자 또는 2 문자(A, B, C, U, V, W, X, Y, Z, AA, BB, ... HH, LL, MM, ... ZZ)로 나타냅니다.
- **{데이터}**는 트리거가 없는 축의 종료 위치 또는 거리를 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.
- **^^{데이터}**는 트리거가 발견된 경우에 트리거 위치에서 정지 위치까지의 거리를 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.
- **[{축}{데이터}^^{데이터}...]**는 여러 축을 동시 동작시키는 옵션을 지정합니다.

Rapid 동작 모드에서는 이 동작 지정에 따라 트리거까지의 동작 기능을 사용할 수 있습니다. ^ 부호보다 앞에 있는 축 동작 설명의 첫 부분은 트리거가 없는 경우의 동작 대상을 지정합니다. 축이 앵글루트 모드인 경우에는 위치이고 축이 인크리멘탈 모드인 경우에는 거리입니다. 어느 쪽도 단위는 스케일링된 축의 사용자 단위입니다. 동작 대상에 도달할 때까지 트리거가 발견되지 않는 경우에 동작은 표준 Rapid 동작이 됩니다.

^ 부호보다 뒤에 있는 축 동작 설명의 두 번째 부분은 트리거가 발견된 경우에 트리거 위치에서 트리거 후 동작 종단까지의 거리를 지정합니다. 양쪽 값은 스케일링된 축의 사용자 단위로 나타냅니다.

명령으로 지정되어 축에 할당된 각 모터는 트리거까지 동작을 개별로 실행합니다. 할당된 모터는 모두 동시에 시작하지만 트리거 조건은 각각 다른 경우가 있습니다. 공통의 트리거가 필요한 경우에는 모든 모터 인터페이스에 트리거 신호를 연결해야 합니다. 각 모터는 개별 시간에 종료할 수 있습니다. 모든 모터 동작이 완료할 때까지 프로그램의 다음 행은 실행을 시작하지 않습니다. 다음 동작에 블렌드할 수 없습니다.

트리거 조건은 저장 설정 데이터 구조체 요소 [Motor[x].CaptureMode]로 설정합니다.

같은 행에서 일부 축을 동시에 실행 시작하는 보통의 트리거가 없는 Rapid 동작을 지정할 수 있습니다.



참고사항

해당 트리거가 있는 동작은 키네마틱스 서브루틴을 통해서 축에 할당된 모터(모터의 축 정의가 #x->I 형식)에서 실행할 수 있습니다.

예

```
X1000^0
X10^-0.01 Y5.43^0.05
A(P1)^(P2) B10^200 C(P3)^0 X10
```

{축}{데이터}{축}{데이터}...{벡터}{데이터}{벡터}{데이터}...

기능: 원호 동작 명령

구문: {데이터}{데이터}{축}{데이터}...{벡터}{데이터}
{벡터}{데이터}...

여기에서

- **{축}**은 동작하는 축을 지정하는 1 문자 또는 2 문자(A, B, C, U, V, W, X, Y, Z, AA, BB, ... HH, LL, MM, ... ZZ)로 나타냅니다.
- **{데이터}**는 종료 위치 또는 거리를 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.
- **{축}{데이터}...**는 여러 축을 동시 동작시키는 옵션을 지정합니다.
- **{벡터}**는 각각 X, Y, Z 축과 평행한 벡터 컴포넌트인 1 문자(I, J, K)와 각각 XX, YY, ZZ 축과 평행한 벡터 컴포넌트를 지정하는 2 문자(II, JJ, KK) 또는 반경 벡터 크기를 지정하는 문자 R(X, Y, Z 축의 세트만 해당)입니다.
- **{데이터}**는 벡터 컴포넌트 또는 반경의 크기를 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.
- **{벡터}{데이터}...**는 벡터 컴포넌트 추가 옵션을 지정합니다.

원호 동작 모드에서는 동작 종점과 원호의 중심에 대한 두 벡터가 지정됩니다. 종점은 직선 보간 모드 동작과 마찬가지로 위치(좌표계 원점을 기준) 또는 거리(시작 위치를 기준)로 지정됩니다.

좌표계 X/Y/Z 직교축 세트 또는 XX/YY/ZZ 직교축 세트의 모든 평면에서 원호 보간이 실행됩니다. 평면은 **normal** 명령으로 지정합니다. 해당 축에서만 원호 보간을 실행할 수 있습니다.

원호 보간의 평면 상에 없는 축은 직선 보간됩니다. 예를 들면 XY 평면이 원호 보간 평면으로 지정된 경우에 Z 축 동작은 직선 보간되고 XYZ 공간에 나선 궤도를 작성합니다. 또한 회전축 동작도 직선 보간됩니다. 원호 궤적에 대해 공구 각도를 탄젠트 각도 또는 수직으로 유지할 수 있습니다.

또한 원호 동작의 원호 중심을 동작 명령으로 지정해야 합니다. 이것은 보통 원호 중심에 대해 벡터 정의로 지정할 수 있습니다. X/Y/Z 직교축 세트에는 I, J 및 K 컴포넌트를 사용하고 XX/YY/ZZ 직교축 세트에는 II, JJ 및 KK 컴포넌트를 사용합니다. 지정되지 않은 벡터 컴포넌트에 대해 Power PMAC 은 값 0.0 을 설정합니다.

벡터 시작점은 동작 시작점(기본값) 또는 좌표계의 프로그래밍 원점입니다. 인크리멘탈 벡터 모드에서는 동작 시작점을 사용하고 절대 위치 벡터 모드에서는 프로그래밍 원점을 사용합니다. **inc({벡터 목록})** 프로그램 명령에 따라 지정된 벡터 컴포넌트가 인크리멘탈 모드가 됩니다. 또한 **abs({벡터 목록})** 프로그램 명령에 따라 지정된 원 중심 벡터 컴포넌트가 앵슬루트 모드가 됩니다. 벡터 인크리멘탈/앵슬루트 모드는 축의 인크리멘탈/앵슬루트 모드와는 다르다는 점에 주의하십시오.

또는 명령행에 **R{데이타}**를 입력하면 중심까지의 벡터의 크기만을 지정할 수 있습니다(X/Y/Z 직교축 세트만 해당). 이 경우에 중심 위치를 Power PMAC 이 계산합니다. **{데이타}**에서 양의 값을 지정한 경우에 Power PMAC 은 목적지까지의 짧은 원호 궤적을 계산($\leq 180^\circ$)하고 음의 값을 지정한 경우에 Power PMAC 은 이동까지의 긴 원호 궤적을 계산($\geq 180^\circ$)합니다. R 벡터 지정자에 의한 하나의 명령으로 전체 원을 지정할 수 있습니다.

목적 지점에서의 원호 방향(시계 방향 또는 시계 반대 방향)은 X/Y/Z 축 세트에서는 좌표계가 **circle1**(시계 방향) 또는 **circle2**(시계 반대 방향) 모드인지에 따라 결정됩니다. XX/YY/ZZ 직교축 세트에서는 좌표계가 **circle3**(시계 방향) 또는 **circle4**(시계 반대 방향) 모드인지에 따라 결정됩니다. 평면의 시계 방향은 **normal** 벡터의 평면에 대한 방향으로 결정됩니다.

중심점에서 목적 지점까지 거리와 시작점까지와의 거리가 다른 경우에 지수 스파이럴 궤적으로 동작하면서 반경을 바꿔갑니다. 이는 원호에서 벗어난 곡선 오차 지정을 보정하기 위해 유효합니다. 저장 설정 요소 **[Coord[x].RadiusErrorLimit]**가 양의 수로 설정된 경우에 에러가 나는 차이의 크기 최소치를 **Status** 요소 **[Coord[x].RadiusError]**로 설정합니다. 모션 프로그램이 정지하고 동작은 실행되지 않습니다.

시작점에서 중심점까지의 벡터가 원호 보간 평면 내에 없는 경우에 평면에 대한 벡터 투영을 수행합니다. 목적 지점이 시작점과 같은 원호 보간 평면 상에 없는 경우에는 헬리컬 동작을 목적 지점까지 수행합니다.

목적 지점(또는 시작점을 포함하는 원호 보간 평면으로의 투영)이 시작점과 같은 경우에 360°인 전체 원이 지정된 방향(I, J, K 벡터가 지정된 경우)으로 작성됩니다. 이 경우에 동작 명령으로 벡터만 지정해야 합니다. 축에 목적 지점이 지정되지 않은 경우에 자동으로 목적 지점이 시작점과 같아지기 때문입니다.

벡터와 반경의 크기가 동작 명령으로 지정되지 않은 경우에는 프로그램이 원호 보간 모드라도 목적 지점까지의 직선 동작을 수행합니다.



참고사항

Power PMAC 은 원호를 세그먼트로 분할하여 각 세그먼트의 3 차원적 곡선상에서 원호 동작을 수행합니다. 좌표계 데이터 구조체 설정 요소 [Coord[x].SegMoveTime]은 각 세그먼트 시간을 결정합니다. 원호 동작을 실행하기 위해 좌표를 세그먼트 모드로 하지만 그때 이 요소를 0 보다 큰 값으로 설정해야 합니다. [Coord[x].SegMoveTime]을 0 으로 설정하면 원호 동작은 직선 동작이 됩니다.

예

```
normal K-1 // Specifies XY-plane for circles
X5000 Y3000 I1000 J1000 // Specifies arc move in XY-plane
X5 Y-1 R2 // Specifies arc move in XY-plane
X-20 Y10 Z3 J5 // Specifies arc move in XY-plane with linear Z move
X10 Y20 C5 I5 J5 // Specifies arc move in XY-plane with rotary axis move
J10 // Specifies a full circle of 10 unit radius

normal J-1 // Specifies ZX-plane for circles
X(P101) Z(P102) I(P201) K(P202) // Specifies arc move in ZX-plane
X10 I5

normal I-1 // Specifies YZ-plane for circles
Y5 Z3 R2 // Specifies arc move in YZ-plane
```

begin

기능: 지정된 모션 프로그램의 프로그램 카운터를 포인트함

구문: **begin[{목록}]:{데이터}**

여기에서

- {목록}**은 쉼표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.
- {데이터}**는 모션 프로그램 번호(정수 부분) 및 옵션에서 행 점프 레이블 번호(소수 부분을 1,000,000 배)를 지정하는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

begin 명령에 따라 주소 지정 또는 목록화된 좌표계, 즉 지정된 좌표계에서는 지정된 모션 프로그램의 선두 또는 지정된 행 점프 레이블 번호에 프로그램 카운터를 설정합니다.

좌표계 목록이 없는 경우에 현재 프로그램에 따라 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 목록 내의 좌표계를 지정해도 프로그램에 따라 모달에 주소 지정된 좌표계에는 영향을 미치지 않습니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 `[Plc[i].Ldata.Coord]`에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(`[Coord[x].Ldata.Coord]`)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

{데이터}의 정수부(버림 등에 따른)는 모션 프로그램 번호를 지정합니다. 소수 부분이 없는 경우에 모션 프로그램의 선두를 가리키듯 프로그램 카운터가 설정됩니다. 소수 부분이 있는 경우에 소수 부분에 1,000,000 을 곱하면 프로그램 카운터가 가리키는 행 점프 레이블 번호가 됩니다.

상당하는 온라인 명령은 생략형 **b** 를 사용하며 그 앞에 좌표계 목록이 놓이고 프로그램 번호 앞에 콜론이 붙지 않는다는 점(&1,2b75 등)에 주의하십시오.

예

```
begin:2;           // Point presently addressed C.S. to top of prog 2
begin1:75;         // Point C.S.1 to top of prog 75
begin3:75.001;     // Point C.S.3 to jump label N1000: of prog 75
begin8:(P1+P2/1000000); // Point C.S.8 to jump label specified by P2 in motion
                    // program specified by P1
```

break

기능: **case** 명령 분기의 종료점에서 **while** 루프를 종료함

구문: **break**

break 명령은 **switch** 여러 분기 구조체 내의 특정 **case** 분기에 따라 실행하는 명령 세트의 종료를 지정합니다. 또한 **while** 또는 **do..while** 루프 도중에 종료를 허가합니다.

프로그램 실행 중에 **break** 명령을 만나면 **switch** 구조체 전체 종료를 나타내는 우측의 중괄호(**}**) 뒤의 다음 명령으로 점프합니다. **case** 분기 후에 **break** 명령이 없는 경우에 프로그램은 다음 케이스를 계속 실행합니다.

while 또는 **do..while** 루프 내에서 명령 실행 시에 **break** 명령에 도달하면 **while** 조건의 현재 상태와 관계없이 프로그램 실행은 루프 종료 뒤의 첫 명령으로 점프합니다.

bstart

기능: 순차적인 스텝으로 실행하는 블록 시작 명령

구문: **bstart**

bstart(블록 스타트) 명령은 실행하는 프로그램 명령 세트의 시작을 하나의 스텝 명령으로 나타냅니다. 스텝 명령을 실행하면 다음 **bstop** 명령이 도달할 때까지 계속 실행합니다. **bstart..bstop** 구조가 없을 때 [Coord[x].StepMode]가 0 이면 스텝 명령은 다음 동작, dwell 또는 지연 명령까지만 계속 실행합니다. [Coord[x].StepMode]가 0 보다 큰 경우에 프로그램행의 끝까지 계속 실행합니다. 이 구조가 있으면 스텝 명령으로 0, 1 또는 그 이상의 동작, dwell 또는 지연 명령이 실행됩니다.

이 명령은 [Coord[x].BlendDisable]이 0 으로 설정되고 모달에 블렌드를 유효로 한 경우에 ('run' 명령에 따라)연속 모드로 실행 중인 프로그램에 영향을 미치지 않습니다. 단, [Coord[x].BlendDisable]이 1 로 설정되고 모달에 블렌드를 무효로 한 경우에 'run' 또는 'step' 명령으로 **bstart** 및 **bstop** 명령 동작이 블렌드됩니다.

이 명령은 PLC 프로그램에서는 'no op'가 됩니다. PLC 프로그램에서 스텝 명령을 실행하면 단순히 싱글 스텝 명령상에서 실행하는 지시(결과 없음)로 기능합니다. 다음 지시에 의한 실행에 대해 영향을 미치지 않습니다.

이 구조는 일련의 PVT 모드 동작 실행에 특히 유효합니다. 각 세그먼트가 속도 제로에서 멈추지 않고 보통 스텝에서의 실행이 매우 어렵기 때문입니다.

예

```
bstart;           // Begin section of continuous execution
inc pvt200;       // Move mode setting
x10:100;          // First specified move (accel)
x20:100;          // Second specified move (slew)
x10:0;            // Third specified move (decel)
bstop;           // End section of continuous execution
```

bstop

기능: 순차적 스텝으로 실행하는 블록 종료 명령

구문: **bstop**

bstop(블록 스톱) 명령은 실행하는 프로그램 명령 세트의 종료를 하나의 스텝 명령으로 나타냅니다. **bstart** 가 도달하면 스텝 명령 실행은 다음 **bstop** 명령에 도달할 때까지 계속 실행합니다. **bstart..bstop** 구조가 없을 때 [Coord[x].StepMode]가 0 이면 스텝 명령은 다음 동작, dwell 또는 지연 명령까지만 계속 실행합니다. [Coord[x].StepMode]가 0 보다 큰 경우에 프로그램행의 끝까지 계속 실행합니다. 이 구조가 있으면 스텝 명령으로 0, 1 또는 그 이상의 동작, dwell 또는 지연 명령이 실행됩니다.

이 명령은 PLC 프로그램에서는 'no op'가 됩니다. PLC 프로그램에서 스텝 명령을 실행하면 단순히 싱글 스텝 명령 상에서 실행하는 지시(결과 없음)로 기능합니다. 다음 지시에 의한 실행에 대해 영향을 미치지 않습니다.

C{데이터}{대입 연산자}{식}

기능: 지정된 'C' 변수에 값을 할당함

구문: **C{데이터}[,{식}[,{식}]**
{대입 연산자}{식}

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 값을 할당하는 첫 번째 변수 번호를 나타내는 소괄호 내의 수식입니다.
- 첫 번째 옵션 **{식}**은 값을 할당하는 변수의 수를 지정합니다. 명확하게 지정되지 않은 경우에는 1이 사용되고 값은 하나의 변수에만 할당됩니다.
- 두 번째 옵션 **{식}**은 값을 할당하는 변수의 숫자 간격을 지정합니다. 명확하게 지정되지 않은 경우에는 1이 사용되고 값은 연속된 번호의 변수에 할당됩니다.
- **{대입 연산자}**는 식에서 값을 변수에 할당하는 방법을 제어하는 숫자 연산자입니다.
- 마지막 **{식}**은 상수, 변수, 논리, 산술 연산자 및 함수를 조합한 수식이며 도출된 식의 값이 할당에 사용됩니다.

C{데이터}{대입 연산자}{식} 명령에 따라 지정된 'C' 변수에 식에서 구한 값이 들어갑니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수 번호를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~63). 유효한 변수 번호를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음 정수로 버림). 평가되지 않는 경우에 명령은 실행하지 않는 것으로 처리됩니다.

각 좌표계에는 키네마틱스 서브루틴 프로그램에 따라 사용되는 독자적인 'C' 변수 세트가 있습니다. 주로 축 위치 및 속도를 루틴과의 사이에서 전달하는 목적으로 사용됩니다. 일반적으로 'C' 변수는 키네마틱스 서브루틴 프로그램에서만 사용됩니다.

지정된 'C' 변수에 기본적인 문자/번호 형식(C6 등)의 이름을 직접 할당합니다. 또한 IDE 소프트웨어 사용 시에는 **#define** 텍스트의 대체이며 변수에 이름을 할당할 수 있습니다. IDE 프로젝트 매니저의 키네마틱스 서브루틴 폴더에 소프트웨어를 쓸 때 IDE는 모든 축의 대체인 **KinPosAxisa** 및 **KinVelAxisa**를 자동으로 정의합니다. 후자의 경우에 기본적인 문자/번호 형식의 변수명으로 대체하는 것은 다운로드 시 자동으로 이루어집니다.

보통 연산자 뒤에 있는 도출된 식은 **=** 대입 연산자를 사용하고 변수에 직접 대입됩니다. 단, 이 값은 대입 연산자 **+=**, **-=**, ***=**, **/=**, **%=**, **&=**, **|=**, **^=**, **>>=**, **<<=**를 사용하여 산술적 또는 논리적으로 변수의 기존 값과 조합할 수도 있습니다. 또한 증가 및 감소 연산자 **++** 및 **--**는 뒤에 식이 없어도 사용할 수 있습니다.

동일한 간격으로 나열되는 여러 변수에 같은 값을 할당할 수 있습니다. 초기 변수명 뒤에 두 번째 값이 사용되는 경우에 값이 할당되는 변수의 수를 지정합니다(두 번째 값이 사용되지 않는 경우에 기본값인 1이 되고 초기 변수가 지정됩니다). 세 번째 값이 사용되는 경우에 변수의 간격을 수치로 지정합니다(세 번째 값이 사용되지 않는 경우에 기본값인 1이 되며 연속된 번호의 변수가 사용됩니다).

'C' 변수는 로컬 변수이기 때문에 글로벌 변수에서 사용 가능한 지연 동기 대입 연산자를 사용할 수 없습니다. 실제로 할당 시에 컨텍스트가 소실될 가능성이 있기 때문입니다.

예

```
C1=17.5;           // Set variable C1 to 17.5
C(P1)+=3;          // Add 3 to C-variable numbered by P1
C(P1+P17)=5*sqrt(P100); // Set C-var numbered by (P1+P17) to expression value
C0,64=0.0;         // Initialize all 64 C-variables
C1,4,2=0.0;        // Initialize C1, C3, C5, and C7
```

call{데이터}

기능: 서브 프로그램으로 점프한 후에 복귀함

구문: **call{데이터}**

여기에서

- **{데이터}**는 종점 위치 또는 거리를 나타내는 부동 소수점 상수(괄호 없음) 또는 식(괄호 있음)입니다. 정수부는 호출하는 서브 프로그램 번호를 나타내고 소수 부분은 목적지 행의 행 점프 레이블(행 점프 레이블 번호는 소수 부분을 1,000,000 배한 것과 같음, 모든 프로그램 선두에는 암시적인 **N0** 가 있음)을 나타냅니다.

call 명령에 따라 **{데이터}**값의 정수부 및 **{데이터}**값의 소수값으로 지정된 행 점프 레이블(**N{상수}** 뒤에 콜론이 있음)로 지정된 서브 프로그램(subprog)까지 프로그램 실행을 점프합니다. 그 후에 다음 **return** 명령이 도달할 때까지 **call** 직후의 명령으로 복귀합니다. Power PMAC 은 소수값을 1,000,000 배 하여 점프 행 레이블값을 구합니다. 곱셈 후에 남은 소수는 버립니다. 따라서 점프 행 레이블의 값은 값의 소수점 이하 6 자리로 보고 필요한 경우 0 을 삽입합니다.



이전 PMAC 및 Turbo PMAC 컨트롤러에서는 소수값을 1,000,000 배가 아니라 100,000 배 했습니다. 이러한 이전 컨트롤러의 프로그램에서는 이 점을 수정해야 합니다.

{데이터}의 값이 정수이고 서브 프로그램에 **N0**: 점프 레이블이 없는 경우에 지정된 번호의 서브 프로그램의 선두로 점프합니다. 서브 프로그램에 명확한 **N0**: 점프 레이블이 있는 경우에 레이블이 서브 프로그램의 선두에 있는지 여부와 상관없이 그 레이블까지 점프합니다.

모션 프로그램, PLC 프로그램 및 서브 프로그램은 **call** 명령을 사용할 수 있지만 호출할 수 있는 것은 서브 프로그램뿐입니다. 최상위 프로그램이 모션 프로그램인 경우에 호출된 서브 프로그램은 모션 프로그램의 규칙에 따라 실행됩니다. 최상위 프로그램이 PLC 프로그램인 경우에 호출된 서브 프로그램은 PLC 프로그램의 규칙에 따라 실행됩니다. 하나의 서브 프로그램을 모션 프로그램과 PLC 프로그램 양쪽에서 호출할 수도 있습니다.

모든 종류의 서브루틴 및 서브 프로그램 호출(**call**, **callsub**, **gosub**, **G**, **M**, **T**, **D**)은 255 레벨(최상위를 포함하여 256)로 네스트할 수 있습니다. 무한 루프는 사용하지 마십시오.

call 명령으로 지정된 서브루틴이 존재하지 않는 경우에 프로그램 실행은 에러로 인해 **call** 명령에서 정지합니다. 단, 서브 프로그램이 존재하더라도 지정된 점프 행 레이블이 존재하지 않는 경우에 서브 프로그램의 선두로 점프합니다.

call 명령에서는 로컬 변수 스택을 사용하는 서브 프로그램으로의 인수를 전달합니다. 서브루틴의 로컬 변수 '**L0**'는 호출 루틴의 로컬 변수 '**R0**'(및 로컬 변수 '**L{StackOffset}**')에 해당합니다. **StackOffset**의 기본값은 256입니다. 다른 값으로 할 경우에 프로그램 다운로드를 시작하는 **open prog**, **open plc** 또는 **open subprog** 명령 내에서 명확하게 선언해야 합니다. 다운로드 중에 IDE 프로젝트 매니저가 이상적인 **StackOffset** 값을 자동으로 할당합니다.

call{데이터} 구문 뒤에 문자 및 데이터(**X1000 DD50** 등)가 있는 경우에 이러한 값들은 서브 프로그램으로 전달하는 인수입니다. 이 인수를 사용하여 RS-274 문자/번호 구문의 프로그램이 서브루틴을 사용할 수 있습니다. 인수를 전달하는 경우에 서브루틴에서 실행되는 최초의 행에 **read** 명령을 사용하십시오. 이 명령은 지정된 문자와 연관된 값을 취득하고 그 값을 문자와 일치하는 적절한 '**D**' 변수에 넣습니다.

예를 들면 A 뒤의 값은 로컬 변수 '**D1**', B 뒤의 값은 로컬 변수 '**D2**', Z 뒤의 값은 로컬 변수 '**D26**'에 들어갑니다. 2 문자 AA 뒤의 값은 '**D27**', BB' 뒤의 값은 로컬 변수 '**D28**', ZZ 뒤의 값은 '**D52**'에 들어갑니다. 그 후에 서브 프로그램이 변수값을 사용할 수 있습니다. 각 좌표계(최상위 프로그램이 모션 프로그램인 경우) 및 각 PLC 프로그램에 '**D**' 변수 세트는 하나뿐입니다. **read** 명령을 사용한 서브루틴/서브 프로그램 네스트가 있는 경우에 매회 같은 '**D**' 변수 세트가 사용됩니다. 자세한 내용은 **read** 명령 사양을 참조하십시오.

예

```
call 250;           // Call to subprog 250
call (Q100);        // Call to subprogram numbered by value of Q100
call 1500 D10;      // Call to subprog 1500, ready to read D value
```

callsub{데이터}

기능: 프로그램 내에서 점프한 뒤에 변수 스택에 데이터를 전달하고 복귀함

구문: **callsub{데이터}**

여기에서

- **{데이터}**는 목적지 행의 점프 레이블을 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

callsub 명령에 따라 **{데이터}**에서 지정된 행 점프 레이블(**N{상수}** 뒤에 콜론이 있음)까지 프로그램 실행(모션, PLC, 서브 프로그램)이 점프됩니다. 그 후에 다음 **return** 명령이 도달할 때까지 **callsub** 직후의 명령으로 복귀합니다. **{데이터}** 식의 값을 완전한 정수로 구할 수 없는 경우에는 다음 정수로 버리면 됩니다. 상수값은 0~999,999 범위의 음의 값이 아닌 정수이어야 합니다. 그 이외의 값인 경우에 다운로드 시에 명령은 구문 에러로 거부됩니다.

{데이터}가 상수인 경우에 프로그램 실행 전에 프로그램 레이블로 궤적이 링크되어 있기 때문에 점프가 매우 빨라집니다. **{데이터}**가 변수 식인 경우에 실행 시에 구하며 적절한 레이블을 검색합니다. 이 방법은 처음 방법보다 시간이 약간 더 걸립니다.

모든 종류의 서브루틴 및 서브 프로그램 호출(**call**, **callsub**, **gosub**, **G**, **M**, **T**, **D**)은 255 레벨(최상위를 포함하여 256)로 네스트할 수 있습니다. 무한 루프는 사용하지 마십시오.

프로그램 입력 시에 항목이 클로즈 명령으로 종료되기까지의 프로그램행에 **callsub{상수}** 명령으로 지정된 점프 레이블 번호가 설명되어 있지 않은 경우에 에러가 생성되어 프로그램 버퍼가 클리어됩니다.

프로그램 실행 시에 식의 값이 계산된 **callsub** 명령으로 지정된 점프 행 레이블이 프로그램에 존재하지 않을 경우에 프로그램 실행은 에러로 인해 **callsub** 명령에서 정지합니다.

gosub 명령과 비슷하지만 **callsub** 명령에서는 로컬 변수 스택을 사용하고 인수를 서브루틴으로 전달합니다. 서브루틴의 로컬 변수 '**L0**'는 호출 루틴의 로컬 변수 '**R0**'(및 로컬 변수 '**L{StackOffset}**')에 해당합니다. **StackOffset**의 기본값은 256입니다. 다른 값으로 할 경우에 프로그램 다운로드를 시작하는 **open prog**, **open plc** 또는 **open subprog** 명령 내에서 명확하게 선언해야 합니다. 다운로드 중에 IDE 프로젝트 매니저가 이상적인 스택 오프셋값을 자동으로 할당합니다.

call{데이터} 구문의 뒤에 문자 및 데이터(**X1000 DD50** 등)가 있는 경우에 해당 값들은 서브 프로그램으로 전달하는 인수입니다. 이 인수를 사용하여 RS-274 문자/번호 구문의 프로그램이 서브루틴을 사용할 수 있습니다. 인수를 전달하는 경우에 서브루틴에서 실행되는 최초의 행에 **read** 명령을 사용하십시오. 이 명령은 지정된 문자와 연관된 값을 취득하고 그 값을 문자와 일치하는 적절한 '**D**' 변수에 넣습니다.

예를 들면 A 뒤의 값은 로컬 변수 '**D1**', B 뒤의 값은 로컬 변수 '**D2**', Z 뒤의 값은 로컬 변수 '**D26**'에 들어갑니다. 2 문자 AA 뒤의 값은 '**D27**', BB' 뒤의 값은 로컬 변수 '**D28**', ZZ 뒤의 값은 '**D52**'에 들어갑니다. 그 후에 서브 프로그램이 변수값을 사용할 수 있습니다. 각 좌표계(최상위 프로그램이 모션 프로그램인 경우) 및 각 PLC 프로그램에 '**D**' 변수 세트는 하나뿐입니다. **read** 명령을 사용한 서브루틴/서브 프로그램 네스트가 있는 경우에 매회 같은 '**D**' 변수 세트가 사용됩니다. 자세한 내용은 **read** 명령 사양을 참조하십시오.

예

```
callsub 400;           // Call to jump label N400: of this program
callsub (Q123);        // Call to jump label numbered by value of Q123
callsub 5000 E5;       // Call to jump label N5000:, ready to pass E value
```

case{상수}:

기능: **switch** 명령 분기

구문: **case{상수}:**

여기에서

- **{상수}**는 정수입니다.

case 명령에서는 특정한 분기를 **switch** 여러 분기 구조체 내에서 시작합니다. **switch** 명령 내 수식(필요에 따라 다음 정수로 버림)이 **case** 명령의 상수값과 같은 경우에 콜론 뒤에 있는 프로그램 명령이 실행됩니다. 명령이 이 프로그램행에 있는지 후속 프로그램행에 있는지와 관계없이 이 명령은 **break** 명령까지 또는 **switch** 구조체 전체 종료를 나타내는 닫는 중괄호(**}**)까지 실행됩니다. 특정 케이스에서 실행하는 명령을 구분하는 여는 중괄호 또는 닫는 중괄호는 필요 없습니다. 각 케이스 내부에서 괄호는 사용할 수 없습니다.

다음 **case** 명령의 앞에 **break** 명령이 없는 경우에 다음 **case** 명령의 뒤까지 프로그램 실행을 계속합니다.

예

```
switch (P1) {                                     // Branch based on value of P1
  case 1:X10 F5; break;                           // Action if P1 = 1
  case 2:X17 F7; break;                           // Action if P1 = 2
  case 3:X-3 F2; break;                           // Action if P1 = 3
  default:dwll 100; break;                        // Action if P1 != 1, 2, or 3
}
```

cclr{상수}

기능: 조건 실행 플래그를 클리어함

구문: **cclr{상수}**

여기에서

- **{상수}**는 조건 플래그 번호를 나타내는 0~31의 정수입니다.

cclr 명령에 따라 좌표계에 지정된 조건 실행 플래그가 클리어(0으로 설정)됩니다. 각 좌표계에 0~31의 번호가 있는 32개의 플래그가 있으며 개별적인 비트인 32비트 데이터 구조체 요소 [Coord[x].Cflags]에 저장됩니다. 플래그 n 은 비트 n 으로 이 요소 내에 2^n 값과 함께 저장됩니다.

플래그 n 이 클리어되면 **cexecn** 명령 뒤에 있는 같은 프로그램행의 프로그램 명령은 실행되지 않습니다. **cskipn** 명령 뒤에 있는 같은 프로그램행의 프로그램 명령은 실행되지 않습니다. 클리어된 플래그는 **cset** 명령으로 설정할 수 있습니다.

cclr 명령은 버퍼 프로그램 명령입니다. 온라인 명령으로 효과적으로 실행하려면 **cx** '컴파일&실행' 구조체(&x cx cclr*n* 등)를 사용하십시오.

ccmode0

기능: 공구 직경 보정을 OFF 로 함

구문: **ccmode0**

ccmode0 명령에 따라 2D 또는 3D 공구 직경 보정이 OFF 가 되고 다음 직선 보간 모드 동작부터 서서히 보정을 해제합니다. 공작기계 표준 RS-274 언어의 **G40** 명령에 상당합니다.



참고사항

이전 PMAC 및 Turbo PMAC 컨트롤러에서 상당하는 명령은 **cc0** 입니다. Power PMAC 에서 **cc0** 는 축 동작 명령입니다.

ccmode1

기능: 2D 공구 직경 보정을 ON 으로 함

구문: **ccmode1**

ccmode1 명령에 따라 (공구 동작 방향을 향해)좌측으로 2D 공구 직경 보정 오프셋이 ON 이 되고 다음 직선 보간 모드 동작부터 서서히 보정을 시작합니다. 공작기계 표준 RS-274 언어의 **G41** 명령에 상당합니다.

2D 보정을 유효로 하려면 좌표계를 세그먼트 모드([Coord[x].SegMoveTime] > 0)로 하고 동작을 시간(tm) 모드에서 피드 레이트(F) 모드로 하며 공구 보정 버퍼를 좌표계로 정의([Coord[x].CCSize] >= 2)해야 합니다. 2D 공구 직경 보정은 직선 보간 모드 및 원호 보간 모드의 동작에만 유효합니다.



참고사항

이전 PMAC 및 Turbo PMAC 컨트롤러에서 상당하는 명령은 **cc1** 입니다. Power PMAC 에서 **cc1** 은 축 동작 명령입니다.

ccmode2

기능: 2D 공구 직경 보정을 ON 으로 함

구문: **ccmode2**

ccmode2 명령에 따라 (공구 동작 방향을 향해)우측으로 2D 공구 직경 보정 오프셋이 ON 이 되고 다음 직선 보간 모드 동작부터 서서히 보정을 시작합니다. 공작기계 표준 RS-274 언어의 **G42** 명령에 상당합니다.

2D 보정을 유효로 하려면 좌표계를 세그먼트 모드([Coord[x].SegMoveTime] > 0)로 하고 동작을 시간(tm) 모드에서 피드 레이트(F) 모드로 하며 공구 보정 버퍼를 좌표계로 정의([Coord[x].CCSize] >= 2)해야 합니다. 2D 공구 직경 보정은 직선 보간 모드 및 원호 보간 모드의 동작에만 유효합니다.

[참고사항](#)

이전 PMAC 및 Turbo PMAC 컨트롤러에서 상당하는 명령은 **cc2** 입니다. Power PMAC 에서 **cc2** 는 축 동작 명령입니다.

ccmode3

기능: 3D 공구 직경 보정을 ON 으로 함

구문: **ccmode3**

ccmode3 명령에 따라 3D 공구 직경 보정이 ON 이 되고 다음 직선 보간 모드 동작부터 서서히 보정을 시작합니다. 3D 보정 오프셋은 부품 프로그램에서 명확하게 지정하므로 2D 보정과 같은 좌측 또는 우측 모드의 선언은 필요 없습니다.

ccmode3 명령이 실행되면 공구 방향 벡터 및 면 법선 벡터가 자동으로 Null 벡터로 설정됩니다. 그 후에 공구 방향 벡터는 **txyz** 프로그램 명령, 면 법선 벡터는 **nxyz** 프로그램 명령으로 지정할 수 있습니다. 실제로 보정을 수행하려면 양쪽의 벡터를 0 이외로 해야 합니다. 또한 공구 선단 형상을 구조체 요소 **[Coord[x].CC3Data]**로 지정할 필요가 없습니다.

3D 보정을 유효로 하려면 좌표계를 세그먼트 모드(**[Coord[x].SegMoveTime] > 0**)로 하고 동작을 시간(tm) 모드에서 피드 레이트(F) 모드로 하며 공구 보정 버퍼를 좌표계로 정의(**[Coord[x].CCSize] >= 2**)해야 합니다. 3D 공구 직경 보정은 직선 보간 모드 및 원호 보간 모드 동작에 유효하지만 일반적으로는 직선 보간 모드의 짧은 동작에만 사용됩니다.

[참고사항](#)

이전 PMAC 및 Turbo PMAC 컨트롤러에서 상당하는 명령은 **cc3** 입니다. Power PMAC 에서 **cc3** 는 축 동작 명령입니다.

ccr{데이터}

기능: 2D 보정 공구 직경을 지정함

구문: **ccr{데이터}**

여기에서

- **{데이터}**는 원통형 공구의 반경을 나타내는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

ccr{데이터} 명령은 2D 공구 직경 보정을 위한 공구의 반경 크기를 X, Y, Z 축 단위로 지정합니다. 많은 경우 이 기능은 D 공구 데이터 코드의 일부로서 공작기계 표준 RS-274 언어에서 사용됩니다.

동작 시에 정의된 보정 평면에 프로그램된 경로에서 이 거리만큼만 공구 중심 경로가 경로에 대해 수직 방향으로 오프셋됩니다.

이 값은 3D 공구 직경 보정에는 사용할 수 없습니다. 3D 의 복잡한 공구 선두 형태의 정의에는 구조체 [Coord[x].CC3Data]를 사용합니다.

cexec{상수}

기능: 남은 프로그램행을 조건에 따라 실행함

구문: **cexec{상수}**

여기에서

- **{상수}**는 조건 플래그 번호를 나타내는 0~31의 정수입니다.

cexec 명령은 좌표계로 지정된 조건 실행 플래그의 현재값에 따라 남은 프로그램행의 실행 여부를 결정합니다. 각 좌표계에 0~31의 번호가 있는 32개의 플래그가 있으며 개별적인 비트인 32비트 데이터 구조체 요소 [Coord[x].Cflags]에 저장됩니다. 플래그 n 은 비트 n 으로 이 요소 내에 2^n 값과 함께 저장됩니다.

Boolean 플래그 n 의 값이 0(클리어)인 경우에 **cexecn** 뒤에 있는 같은 프로그램행의 프로그램 명령은 실행되지 않습니다. Boolean 플래그 n 의 값이 1(설정)인 경우에 **cexecn** 뒤에 있는 같은 프로그램행의 프로그램 명령은 실행됩니다. 같은 프로그램행에 하나 이상의 **cexecn** 및/또는 **cskipn** 명령이 있는 경우가 있습니다.

클리어된 플래그는 **csetn** 명령으로 설정할 수 있습니다. 설정된 플래그는 **cclrn** 명령으로 클리어할 수 있습니다.

circle

기능: 원호 보간 동작 모드를 설정함

구문: **circle{상수}**

여기에서

- **{상수}**는 원호 보간 모드를 나타내는 1~4의 정수입니다.

circle 명령에 따라 직교축 세트 중 하나의 모션 프로그램이 원호 보간 동작 모드가 됩니다. 이 모드에서 다른 축은 직선 보간 모드로 동작합니다. 모션 프로그램의 후속 동작 명령은 이 모드의 규칙에 따라 처리됩니다.

circle 명령의 4가지 변형과 실행되는 모드는 다음과 같습니다.

- **circle1**: X/Y/Z 축 세트를 시계 방향의 원호 보간 모드로 설정
- **circle2**: X/Y/Z 축 세트를 시계 반대 방향의 원호 보간 모드로 설정
- **circle3**: XX/YY/ZZ 축 세트를 시계 방향의 원호 보간 모드로 설정
- **circle4**: XX/YY/ZZ 축 세트를 시계 반대 방향의 원호 보간 모드로 설정

하나의 직교축 세트를 원호 보간 모드로 설정해도 직선 또는 원호 모드로 설정된 다른 직교축에 영향을 미치지 않습니다. 단, 그 외의 직교축 세트가 다른 모드(**pvt**, **rapid**, **spline**)로 설정된 경우에 직선 모드가 됩니다. 다른 동작 모드(**linear**, **pvt**, **rapid**, **spline**) 명령을 실행하면 양쪽의 직교축 세트가 원호 모드에서 변경됩니다.

각 직교축 세트의 원호 보간 평면은 **normal** 명령에 따라 결정됩니다. 직교축 세트 중 하나의 축이 피드 레이트축인 경우에 원호 보간에 사용되는 그 세트의 모든 축은 명시적으로 피드 레이트축으로 선언됐는지 여부와 상관없이 원호 동작에서는 피드 레이트축으로 처리됩니다.

원호 보간을 실행하려면 좌표계를 세그먼트 동작 모드로 하고 데이터 구조체의 저장 설정 요소 `[Coord[x].SegMoveTime](lsx13)`를 1 이상으로 설정해야 합니다. 요소를 0으로 설정하면 세그먼트 모드가 무효가 되고 동작은 직선 보간됩니다.

cmd

기능: PMAC 명령 파서에 명령을 전송함

구문: **cmd**[,]"{문자열}"[, {식}...]

여기에서

- **{문자열}**은 리터럴 영숫자, 이스케이프 시퀀스 및 포맷된 변수 시퀀스로 구성되는 포맷된 ASCII 문자열입니다.
- **{식}**은 문자열과 일치하는 포맷된 변수 시퀀스로 송신되는 값을 포함하는 수식입니다. 포맷된 각 변수 시퀀스마다 하나의 수식이 필요합니다.

cmd 프로그램에 따라서 'gpascii' 스레드를 통해서 외부 컴퓨터에서 송신된 것처럼 Power PMAC 이 Power PMAC 에 명령으로 포맷된 문자열을 전송합니다. Power PMAC 에는 다른 통신 스레드와는 독립하여 해당 명령을 항상 처리할 수 있는 전용 통신 스레드가 있습니다.

프로그램 실행의 일부로서 명령이 처리되면 문자열이 작성되어 통신 스레드에서 처리하도록 버퍼됩니다. 문자열이 버퍼되면 문자열이 스레드에서 처리될 때까지 기다리지 않고 프로그램 실행을 계속합니다.

송신하는 문자열은 표준 ASCII 문자 '\'(백슬래시)로 시작하여 컨트롤 및 특수 문자의 출력을 허가하는 이스케이프 시퀀스, '%' (퍼센트) 문자로 시작하여 식의 수치를 문자열로 포맷하는 방법을 지정한 포맷 시퀀스의 조합입니다. 각 포맷 시퀀스에는 문자열 뒤에 일치하는 식이 필요합니다.

다음 이스케이프 시퀀스를 사용할 수 있습니다(전부 소문자).

- **\a** 경고음(ASCII 7)
- **\b** 백스페이스(ASCII 8)
- **\t** 가로 탭(ASCII 9)
- **\n** 줄 바꿈(ASCII 10)
- **\v** 세로 탭(ASCII 11)

- `\f` 폼 피드(ASCII 12)
- `\r` 캐리지 리턴(ASCII 13)
- `\\` 백슬래시 문자
- `\?` 물음표
- `\'` 작은따옴표
- `\"` 큰따옴표
- `\ooo` ASCII 문자 코드의 8 진법(000~377 범위)
- `\xhh` ASCII 문자 코드의 16 진법(x00~xff 범위)

다음과 같은 포맷 시퀀스를 사용할 수 있습니다(대·소문자 여부 본문 기재).

- `%d` 부호가 있는 정수, 10 진법 형식
- `%u` 부호가 없는 정수, 10 진법 형식
- `%x` 부호가 없는 정수, 16 진법 형식, 소문자 사용
- `%nx` 부호가 없는 정수, 16 진법 형식, n 자리 사용($n = 1\sim 8$), 소문자 사용
- `%X` 부호가 없는 정수, 16 진법 형식, 대문자 사용
- `%nX` 부호가 없는 정수, 16 진법 형식, n 자리 사용($n = 1\sim 8$), 대문자 사용
- `%f` 부동 소수점 값, 최대 6 자리까지
- `%nf` 부동 소수점 값, 최대 n 자리까지($n = 1\sim 31$)
- `%n.mf` 부동 소수점 값, 최대 n 자리까지($n = 1\sim 31$), 소수 최대 m 자리($m < n$)
- `%s` 문자열, 임의의 길이(Null 종단됨)
- `%ns` 문자열, 최대 n 문자(Null 종단 문자가 있는 경우에는 이보다 짧음)
- `%c` 지정된 바이트값의 1 문자
- `%nc` 지정된 바이트값의 1 문자의 n 회 반복
- `%%` 퍼센트 문자

선행 제로열은 10 진법 정수로는 작성되지 않습니다. 자릿수가 지정된 경우에만 선행 제로열이 16 진법 정수로 작성됩니다. 자릿수 지정 여부와 관계없이 선행 및 끝 제로열은 부동 소수점 값으로 작성되지 않습니다. 부동 소수점 값에 지정된 자릿수가 지수 없이 값을 나타내기에 불충분한 경우에 자동으로 지수가 붙습니다(2.345e8 등). 0 이 아닌 소수 부분이 있는 경우에 소수점은 부동 소수점 값만 사용됩니다. 부동 소수점 값의 소수점, 음의 부호, 지수는 자릿수로 세지 않습니다.

문자열이 포맷된 각 변수 시퀀스에는 문자에 변환되는 수치를 지정하는 **{식}**이 포함됩니다. 식은 단순히 하나의 상수 또는 변수인 경우도 있지만 여러 개의 수학 연산자나 식을 포함하는 경우도 있습니다. 문자열 변수에는 식의 값(필요에 따라 다음 정수로 버림)이 사용되는 문자열 변수의 사용자 공유 메모리 내에서의 시작 인덱스를 나타냅니다.

모터 또는 좌표계 목록에 따라 즉시 처리되지 않는 모터 고유 또는 좌표계 고유의 명령인 경우에 `[Ldata.Motor]` 또는 `[Ldata.Coord]`에 따라 각각 지정된 모터에서 동작합니다. 명령에는 명확하게 모터 또는 좌표계 목록을 포함할 것을 적극 권장합니다.

명령에 대해 응답이 없는 경우에도 처리된다는 점에 주의하십시오. 응답이 있는 경우에 명령은 사라지지만 에러는 발생하지 않습니다.

좌표계의 로컬 데이터 구조체(모션 프로그램 내인 경우)나 PLC 프로그램에서는 Power PMAC 스크립트 명령이 정상적으로 실행되면 요소 [Ldata.CmdStatus]가 0 으로 설정됩니다. 명령 실행 시에 에러가 발생하면 음의 번호로 설정됩니다. 명령 실행을 감시하는 경우에 명령을 실행하기 전에 이 요소를 양의 번호로 설정하십시오.

또한 로컬 데이터 구조체에서는 명령을 실행하면 명령의 정상적 실행 여부와 관계없이 요소 [Ldata.CmdCount]는 증가합니다. 명령 실행 시에 오류가 발생해도 아무 처리도 하지 않는다는 점에 주의하십시오. 카운트를 확인하기 전에 명령 버퍼를 확실히 플러시하려면 **sendallcmds** 명령을 사용하십시오. 사용자는 [Ldata.CmdCount]에 씁니다. 일부 사용자는 명령 세트를 전송하기 전에 0 으로 설정하고 기대되는 수의 명령이 실행되는지 감시합니다.

명령 처리 시에 에러가 되면 명령이 다음에 정상 실행될 때까지(또는 사용자가 음이 아닌 값으로 덮어쓸 때까지) [Ldata.CmdStatus]는 음의 값을 가지게 됩니다.

텍스트 명령 내부에서 로컬 변수는 사용되지 않습니다. 명령 프로세서가 로컬 처리의 일부가 아니기 때문입니다.

예

```
cmd"&1#4->1000C"; // Assign Motor 4 to C-axis in C.S. 1
cmd"&1#4->%fC", 24400*EngMode+1000; // Assign # 4 as English or metric

Ldata.CmdStatus=1; // Set so can look for successful execution
cmd"M1->u.io:$A00000.%u,1",P10; // Assign M-variable to bit specified by P10
sendallcmds; // Force commands from buffer
while (Ldata.CmdStatus > 0) {} // Wait for command to execute
if (Ldata.CmdStatus < 0) { // Error in execution?
    {error trapping commands} // Handle the error
};

Ldata.CmdCount=0; // Set so can look for command execution
cmd"&1#4->0"; // Clear axis definition in C.S. 1
cmd"&2#4->1000C"; // Make axis definition in C.S. 2
sendallcmds; // Force commands from buffer
while (Ldata.CmdCount < 2) {} // Wait for command to execute
```

continue

기능: 루프 조건 평가로 점프함

구문: **continue**

continue 명령에 따라 **while** 또는 **do..while** 루프 내부의 프로그램 실행이 사이에 있는 프로그램 명령을 바이패스하고 **while** 명령 내의 루프 조건 평가로 점프합니다.

이 명령을 **while** 또는 **do..while** 루프 바깥쪽의 Power PMAC 에 송신한 경우에 Power PMAC 에서 에러로 거부됩니다.

예

```
P1=0; P2=0; // Initialize loop counter and conditional value
while (P1<1000) { // Loop conditional
    P1++; // Increment loop counter
    if (P2>0) continue; // Jump back to loop conditional
    dwell1000; // Pause for 1 second
} // End of loop
```


cout:{데이터}

기능: 지정된 백분율의 오픈 루프 출력을 명령함

구문: **cout[{목록}]:{데이터}**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0 부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1 을 뺀 값까지입니다.
- **{데이터}**는 최대 서보 출력의 퍼센트를 지정하는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

cout:{데이터} 명령에 따라 Power PMAC 은 주소 지정 또는 지정된 모터를 오픈 루프 유효 모드로 하고 서보 루프 출력을 강제로 지정된 크기로 합니다. 출력 크기는 모터의 파라미터의 최대 출력 파라미터에 대한 (부호가 있는) 퍼센트이고 **[Motor[x].MaxDac]**으로 표시됩니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

상수를 사용하는지 식을 사용하는지에 관계없이 지정된 값이 100.0 보다 큰 경우에 값은 100.0 이 되고 지정된 값의 부호가 적용됩니다. Status 비트는 설정되지 않고 에러도 발생하지 않습니다. 따라서 **[Motor[x].MaxDac]**이 최대값보다 작은 경우라도 **[Motor[x].MaxDac]**보다 큰 오픈 루프 크기는 지정할 수 없습니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(**Plc[i].Ldata.Motor**)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(**Coord[x].Ldata.Motor**)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={식}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터가 명령을 받아 동시에 동작을 시작합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

일반적으로 이 명령은 PLC 프로그램에서만 사용하십시오. 모션 프로그램 표준 동작에서는 실행 중인 좌표계의 한 축에 할당된 모든 모터가 클로즈드 루프가 되어있어야 하기 때문입니다.

오픈 출력의 온라인 명령은 생략형 **out**(콜론 없음)을 사용하며 모터 목록이 그 앞에 놓인다는 점(**#1,2out25** 등)에 주의하십시오.

예

```
cout:10;           // 10% output from presently addressed motor
cout1:(Q1);        // Q1% output from Motor 1, regardless of addressed
cout1,2,3:-15;      // -15% output from Motors 1, 2, and 3
cout1..3,5..7:0;    // 0% output from Motors 1, 2, 3, 5, 6, and 7
```

cset{상수}

기능: 조건 실행 플래그를 설정함

구문: **cset{상수}**

여기에서

- **{상수}**는 조건 플래그 번호를 나타내는 0~31의 정수입니다.

cset 명령에 따라 좌표계에 지정된 조건 실행 플래그가 설정(1로 설정)됩니다. 각 좌표계에 0~31의 번호가 있는 32개의 플래그가 있으며 개별적인 비트인 32비트 데이터 구조체 요소 [Coord[x].Cflags]에 저장됩니다. 플래그 n 은 비트 n 으로 이 요소 내에 2^n 값과 함께 저장됩니다.

플래그 n 이 설정되면 **cexecn** 명령 뒤에 있는 같은 프로그램행의 프로그램 명령이 실행됩니다. **cskipn** 명령 뒤에 있는 같은 프로그램행의 프로그램 명령이 실행됩니다. 설정된 플래그는 **cclr** 명령으로 클리어할 수 있습니다.

cset 명령은 버퍼 프로그램 명령입니다. 온라인 명령으로 효과적으로 실행하려면 **cx** '컴파일&실행' 구조체(&x cx csetn)를 사용하십시오.

cskip{상수}

기능: 남은 프로그램행을 조건에 따라 건너뛸

구문: **cskip{상수}**

여기에서

- **{상수}**는 조건 플래그 번호를 나타내는 0~31의 정수입니다.

cskip 명령은 좌표계로 지정된 조건 실행 플래그의 현재값에 따라 남은 프로그램행의 실행 여부를 결정합니다. 각 좌표계에 0~31의 번호가 있는 32개의 플래그가 있으며 개별적인 비트인 32비트 데이터 구조체 요소 [Coord[x].Cflags]에 저장됩니다. 플래그 n 은 비트 n 으로 이 요소 내에 2^n 값과 함께 저장됩니다.

Boolean 플래그 n 의 값이 1(설정)인 경우에 **cskipn** 뒤에 있는 같은 프로그램행의 프로그램 명령이 실행됩니다. Boolean 플래그 n 의 값이 0(클리어)인 경우에 **cskipn** 뒤에 있는 같은 프로그램행의 프로그램 명령이 실행됩니다. 같은 프로그램행에 하나 이상의 **cskipn** 및/또는 **cexecn** 명령이 있는 경우가 있습니다.

클리어된 플래그는 **csetn** 명령으로 설정할 수 있습니다. 설정된 플래그는 **cclrn** 명령으로 클리어할 수 있습니다.

{데이터 구조체 요소}{대입 연산자}{식}

기능: 지정된 데이터 구조체 요소에 값을 할당함

구문: **{데이터 구조체 요소}{대입 연산자}{식}**

여기에서

- **{데이터 구조체 요소}**는 사전에 정의된 데이터 구조체에 포함되는 개별 요소의 이름입니다.
- **{대입 연산자}**는 식에서 값을 변수에 할당하는 방법을 제어하는 숫자 연산자입니다.
- **{식}**은 상수, 변수, 논리 및 산술 연산자, 함수를 조합한 수식이며 도출된 식의 값이 할당에 사용됩니다.

{데이터 구조체 요소}{대입 연산자}{식} 명령에 따라 지정된 데이터 구조체 요소에 식에서 구한 값이 들어갑니다.

보통 구한 값은 표준적인 대입 연산자(=)를 사용하여 요소에 직접 대입됩니다. 단, 이 값은 표준적인 대입 연산자 +=, -=, *=, /=, %=, &=, |=, ^=, >>=, <<=를 사용하여 산술적 또는 논리적으로 요소의 기존 값과 조합할 수도 있습니다. 또한 증가 및 감소 연산자 ++ 및 --는 뒤에 식이 없어도 사용할 수 있습니다.

변수의 실제 할당을 프로그램의 다음 명령 동작이 실제로 실행 시작할 때까지 지연되는 경우에 동기 대입 연산자를 사용할 수 있습니다. 모션 프로그램에서 여러 동작의 블렌드 또는 스플라인 시퀀스 중에 출력을 설정할 때 도움이 됩니다. 모션 프로그램에서는 동작을 어떻게 블렌드 또는 스플라인할지 구하기 위해 하나 이상의 선행 동작을 계산해야 하기 때문입니다. 표준적인 대입 연산자에서는 값이 설정되는 타이밍이 너무 빠릅니다. 동기 할당을 설정하면 다음 동작이 실행 시작할 때까지 실제 할당을 지연할 수 있기 때문에 직관적으로 기대되는 시간에 출력을 실시합니다.

또한 동기 할당은 동작 종료를 확인하기 전에 명령된 동작이 시작되도록 모터 또는 축 동작을 직접 명령하는 PLC 프로그램에서도 도움이 됩니다. 모터 동작에서는 모터를 좌표계에 할당해야 한다는 점에 주의하십시오. 지연 할당을 유지하는 동기 할당 큐는 좌표계에 속하기 때문입니다.

보통 명령 시에 구한 식의 값은 == 동기 대입 연산자를 사용하여 다음 동작의 실행 시작 시에 변수에 직접 대입됩니다. 단, 이 값은 산술 동기 변수 대입 연산자 +=, -=, *=, /=, %= 또는 논리 동기 대입 연산자 &=, |=, ^=를 사용하여 산술적 또는 논리적으로 변수의 기존 값과 조합할 수도 있습니다. 또한 동기 증가 및 감소 연산자 ++= 및 --=는 뒤에 식이 없어도 사용할 수 있습니다.

산술 동기 대입 연산자는 부동 소수점 요소만으로 사용할 수 있고 논리 동기 대입 연산자는 고정 소수점(정수) 요소만으로 사용할 수 있다는 점에 주의하십시오. 동기 대입 연산자는 로컬(Ldata) 요소에서는 사용할 수 없습니다.

예

```
Motor[3].JogSpeed=38;           // Set value of Motor 3's commanded jog speed
Coord[2].MaxFeedrate+=25;       // Add 25 to C.S. 2's maximum feedrate
Motor[5].Servo.Ki==7;           // Set #5's integral gain to 0 at next move
Ldata.Motor=4;                  // Set value of program's own addressed motor
```

D{데이터}

기능: 톨 데이터 코드(D 코드)

구문: **D{데이터}**

여기에서

- **{데이터}**는 코드 번호 및 그 코드를 탑재하는 서브루틴의 행 점프 레이블을 나타내는 0.000~999.999 의 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

Power PMAC 은 D 코드를 **call{Coord[x].Dprog}.{데이터}*1000** 명령(**{데이터}**는 D 코드의 번호)으로 해석합니다. 이 명령에 따라 코드 번호를 1,000 배 한 번호의 행 점프 레이블 위치에서 [Coord[x].Dprog]의 값에 따라 번호가 지정된 서브 프로그램으로 점프합니다. 예를 들어 [Coord[x].Dprog]의 기본값이 1003 인 경우에 **subprog 1003**의 행 점프 레이블 **N3000:**에서 **D03** 이 호출됩니다. 프로그램 실행은 다음 **return** 명령에서 호출자 프로그램으로 복귀합니다.

이 구조체에 따라 커스터마이징 가능한 D 코드 루틴을 CAD/CAM 프로그램의 RS-274 대응 코드 및 CNC 스타일의 어플리케이션에 탑재할 수 있습니다. 호출자 프로그램 D 코드의 뒤에 1 세트 이상의 **{문자}{데이터}**를 두면 해당 서브루틴에 인수를 전달할 수 있습니다. 각 문자의 뒤의 값은 **read** 명령을 사용하여 서브루틴으로 취득할 수 있습니다. 모든 좌표계의 [Coord[x].Dprog] 기본값은 1003 입니다. 따라서 모든 좌표계는 기본값으로 같은 D 코드 세트를 공유합니다. 단, 다른 좌표계에 다른 값을 할당하면 다른 코드를 탑재할 수 있습니다.

일반적으로 D 코드는 RS-274 프로그램에서 공구 데이터 코드로 사용되며 공구 길이, 반경 등을 지정합니다. D 코드의 코어 세트는 0~99 의 정수를 사용하지만 이 스킴에서는 999.999 까지의 소수 코드 번호를 사용할 수 있고 코어 번호에 수식을 사용할 수 있습니다.

모든 종류의 서브루틴 및 서브 프로그램 호출(**call**, **callsub**, **gosub**, **G**, **M**, **T**, **D**)은 255 레벨(최상위를 포함하여 256)로 네스트할 수 있습니다. 무한 루프는 사용하지 마십시오.

D 코드 명령에 따라 지정된 점프 행 레이블이 프로그램에 존재하지 않는 경우에 지정된 D 코드의 서브 프로그램 선두로 점프합니다. 에러는 발생하지 않습니다. 이를 통해 사용자 어플리케이션이 존재하지 않는 D 코드를 처리하는 방법을 결정할 수 있습니다.

예

```
D03           // Call to label N3000: of D-code subprogram
D23.7         // Call to label N23700: of D-code subprogram
D135          // Call to label N135000: of D-code subprogram
```

$D\{\text{데이터}\}\{\text{대입 연산자}\}\{\text{식}\}$

기능: 지정된 'D' 변수에 값을 할당함

구문: $D\{\text{데이터}\}\{\text{식}\}\{\text{식}\}$
 $\{\text{대입 연산자}\}\{\text{식}\}$

여기에서

- $\{\text{데이터}\}$ 는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 값을 할당하는 첫 번째 변수 번호를 나타내는 소괄호 내의 수식입니다.
- 첫 번째 옵션 $\{\text{식}\}$ 은 값을 할당하는 변수의 수를 지정합니다. 명확하게 지정되지 않은 경우에는 1 이 사용되고 값은 하나의 변수에만 할당됩니다.
- 두 번째 옵션 $\{\text{식}\}$ 은 값을 할당하는 변수의 숫자 간격을 지정합니다. 명확하게 지정되지 않은 경우에는 1 이 사용되고 값은 연속된 번호의 변수에 할당됩니다.
- $\{\text{대입 연산자}\}$ 는 식에서 값을 변수에 할당하는 방법을 제어하는 숫자 연산자입니다.
- 마지막 $\{\text{식}\}$ 은 상수, 변수, 논리, 산술 연산자 및 함수를 조합한 수식이며 도출된 식의 값이 할당에 사용됩니다.

$D\{\text{데이터}\}\{\text{대입 연산자}\}\{\text{식}\}$ 명령에 따라 지정된 'D' 변수에 식에서 구한 값이 들어갑니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수 번호를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~53). 유효한 변수 번호를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 실행하지 않는 것으로 처리됩니다.

각 좌표계에는 **read** 명령에 따라 사용되는 독자적인 'D' 변수 세트가 있습니다. 주로 G 코드 형식 프로그램에서 인수를 전달하는 목적으로 사용됩니다. 일반적으로 'D' 변수는 이 목적으로만 사용됩니다.

지정된 'D' 변수에 기본적인 문자/번호 형식(D6 등)의 이름을 직접 할당합니다. 또한 IDE 소프트웨어 사용 시에는 **#define** 텍스트의 대체로 변수에 이름을 할당합니다. 두 번째의 경우에 기본적인 문자/번호 형식의 변수명으로 대체하는 것은 다운로드 시 자동으로 이루어집니다.

동일한 간격으로 나열되는 여러 변수에 같은 값을 할당할 수 있습니다. 초기 변수명 뒤에 두 번째 값이 사용되는 경우에 값이 할당되는 변수의 수를 지정합니다(두 번째 값이 사용되지 않는 경우에 기본값인 1 이 되고 초기 변수가 지정됩니다). 세 번째 값이 사용되는 경우에 변수의 간격을 수치로 지정합니다(세 번째 값이 사용되지 않는 경우에 기본값인 1 이 되며 연속된 번호의 변수가 사용됩니다).

보통 연산자 뒤에 있는 도출된 식은 = 대입 연산자를 사용하고 변수에 직접 대입됩니다. 단, 이 값은 대입 연산자 +=, -=, *=, /=, %=, &=, |=, ^=, >>=, <<=를 사용하여 산술적 또는 논리적으로 변수의 기존 값과 조합할 수도 있습니다. 또한 증가 및 감소 연산자 ++ 및 --는 뒤에 식이 없어도 사용할 수 있습니다.

'D' 변수는 로컬 변수이기 때문에 글로벌 변수에서 사용 가능한 지연 동기 대입 연산자를 사용할 수 없습니다. 실제로 할당 시에 컨텍스트가 소실될 가능성이 있기 때문입니다.

예

```

D1=17.5;           // Set variable D1 to 17.5
D(P1)+=3;          // Add 3 to D-variable numbered by P1
D(P1+P17)=5*sqrt(P100); // Set D-var numbered by (P1+P17) to expression value
D0,53=0.0;         // Set all 53 D-variables to 0
D1,4,2=0.0;        // Set D1, D3, D5, and D7 to 0

```

ddisable

기능: 좌표계의 모든 모터를 무효로 함

구문: **ddisable**[{목록}]

여기에서

- **{목록}**은 십표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.

ddisable 명령에서 Power PMAC은 주소 지정 또는 지정된 좌표계에 정의된 모든 모터의 제어를 지연 Disable로 합니다. 이것은 기본적으로 좌표계의 모든 모터에 대한 모터 고유의 **dkill** 명령에 상당하지만 모터 명령과는 다르게 모션 프로그램을 실행하는 좌표계에 작용하여 처리 중인 모션 프로그램을 중단시킵니다.

좌표계 목록이 없는 경우에 현재 프로그램에 따라 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 목록 내의 좌표계를 지정해도 프로그램에 따라 모달에 주소 지정된 좌표계에는 영향을 미치지 않습니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 [Plc[i].Ldata.Coord]에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(Coord[x].Ldata.Coord)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

모터 제어 Disable은 앰프 Enable 상태의 'False'(무효) 설정, 위치/속도 서보 루프의 오픈, 제로 서보 출력값의 강제로 구성됩니다. 클로즈드 루프 또는 오픈 루프 유효 상태에서 모터는 출력 정지 상태가 됩니다.

[Motor[x].pBrakeOut]을 제로 이외의 값(브레이크 출력 레지스터의 주소)으로 설정하면 모터의 자동 브레이크 제어 기능이 유효한 경우에 브레이크 출력은 **ddisable** 명령으로 즉시 연동하도록 명령됩니다. 게다가 모터가 클로즈드 루프, 제로 속도 상태인 경우에 모터의 실제 출력 정지는 [Motor[x].BrakeOnDelay] 밀리초만큼 지연되어 브레이크 시스템이 완전히 연동될 시간을 제공합니다.

이 지연 **Disable** 명령은 브레이크 장착 모터의 **Disable** 을 계획하는 경우에 브레이크가 완전히 연동하기까지 시간을 확보하는 용도입니다. 모터의 긴급한 **Disable** 은 마찬가지로 즉시 **disable** 명령을 사용해야 합니다.

ddisable 명령은 모든 무효 상태의 모터에는 영향을 미치지 않습니다.

예

```
ddisable;           // Delayed disable of all motors in presently addressed C.S.
ddisable4;          // Delayed disable of all motors in C.S. 4
ddisable5..8;       // Delayed disable of all motors in C.S. 5, 6, 7, & 8
```

default:

기능: 일치하지 않는 경우의 **switch** 명령 분기의 시작

구문: **default:**

default 명령은 **switch** 식의 값(필요에 따라 다음 정수로 버림)이 **case** 명령의 상수와 일치하지 않을 때 실행되는 **switch** 여러 분기 구조체의 분기 시작을 나타냅니다. **default** 명령은 **switch** 구조체의 **case** 명령의 뒤에 반드시 필요한 것은 아니지만 관습적으로 그 위치에 둡니다. **switch** 구조체에 **default** 는 필요 없습니다.

일치하지 않는 경우에 **default** 명령의 콜론 뒤에 있는 프로그램 명령이 실행됩니다. 명령이 이 프로그램행에 있는지 후속 프로그램행에 있는지와 관계없이 이 명령은 **break** 명령까지 또는 **switch** 구조체 전체 종료를 나타내는 닫는 중괄호(**}**)까지 실행됩니다. **default** 케이스에서는 실행하는 명령을 구분하는 여는 중괄호 또는 닫는 중괄호가 필요 없습니다. 각 케이스 내부에서 괄호는 사용할 수 없습니다.

다음 **case** 명령의 앞에 **break** 명령이 없는 경우에 다음 **case** 명령의 뒤까지 프로그램 실행을 계속합니다.

예

```
switch (P1) {           // Branch based on value of P1
  case 1:X10 F5; break;  // Action if P1 = 1
  case 2:X17 F7; break;  // Action if P1 = 2
  case 3:X-3 F2; break;  // Action if P1 = 3
  default:dwel 100; break; // Action if P1 != 1, 2, or 3
}
```

delay{데이타}

기능: 모든 축을 지정된 시간만큼 지연시킴

구문: **delay{데이타}**

여기에서

- {데이타}**는 지연 시간을 밀리초로 나타내는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

delay 명령이 실행되면 모션 프로그램을 실행하는 좌표계의 모든 축에 대해 지정 시간 동안 현재 명령 위치를 유지하도록 명령합니다. 이 시간에 접근하는 동작의 감속 시간의 반 그리고 멀어지는 동작의 가속 시간의 반이 포함됩니다.

명령된 지연 시간은 오버라이드 또는 타임 베이스값에 영향을 받습니다. 50% 오버라이드일 때 실제로 지연 시간은 명령 시간의 배가 됩니다. 따라서 외부 타임 베이스 모드 중에 **delay** 명령을 사용해도 마스터 입력과의 동기를 유지할 수 있습니다.

delay 명령은 룩 어헤드 계산을 정지하지 않습니다. 지연 시간이 실제로 종료하기 전에 모션 프로그램의 후속행을 평가할 수 있습니다.

dealy 명령은 밀리초로 지정된 시간의 제로 거리 동작에 해당합니다. 동작 명령과 마찬가지로 지정된 지연 시간이 현재 유효한 가속 시간보다 적은 경우에 지정된 지연 시간이 아니라 가속 시간의 지연이 됩니다.

dwell 명령과 비슷하지만 약간 다릅니다. 드웰 시간에는 접근하는 동작의 감속 시간 및 멀어지는 동작의 가속 시간이 포함되지 않으며 오버라이드 및 타임 베이스값에 영향을 받지 않습니다. 또한 룩 어헤드 계산을 금지합니다.

예

```
delay250;  
delay(P1+P2);
```

disable

기능: 좌표계의 모든 모터를 무효로 함

구문: **disable**{**{목록}**}

여기에서

- {목록}**은 쉼표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.

disable 명령에서 Power PMAC은 주소 지정 또는 지정된 좌표계의 모든 모터의 제어를 무효로 합니다. 이것은 기본적으로 좌표계의 모든 모터에 대한 모터 고유의 **kill** 명령에 상당하지만 모터 명령과는 다르게 모션 프로그램을 실행하는 좌표계에 작용하여 처리 중인 모션 프로그램을 중단시킵니다.

좌표계 목록이 없는 경우에 현재 프로그램에 따라 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 목록 내의 좌표계를 지정해도 프로그램에 따라 모달에 주소 지정된 좌표계에는 영향을 미치지 않습니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 **[Plc[i].Ldata.Coord]**에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={**{**속**}**}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(Coord[x].Ldata.Coord)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

모터 제어 Disable 은 앰프 Enable 상태의 'False'(무효) 설정, 위치/속도 서보 루프의 오픈, 제로 서보 출력값의 강제로 구성됩니다. 클로즈드 루프 또는 오픈 루프 유효 상태에서 모터는 출력 정지 상태가 됩니다.

즉시 출력 정지 명령은 모터의 긴급 정지용입니다. 자동 브레이크 제어를 이용하여 모터 Disable 을 계획하는 경우에는 브레이크가 완전히 연동할 때까지 시간이 필요하기 때문에 동일하게 지연된 **ddisable** 명령으로 실행해야 합니다. 자동 브레이크 제어가 장착되어 있지 않은 모터는 어느 명령을 사용해도 문제 없습니다.

disable 명령은 모든 무효 상태의 모터에는 영향을 미치지 않습니다.

예

```
disable;           // Disable all motors in presently addressed C.S.
disable4;          // Disable all motors in C.S. 4
disable5..8;       // Disable all motors in C.S. 5, 6, 7, & 8
```

disable bgcplc

기능: 지정된 백그라운드 C PLC 프로그램을 무효로 함

구문: **disable bgcplc {목록}**

여기에서

- {목록}**은 쉼표로 구분된 정수 상수 또는 시작 상수에서 종료 상수 사이를 2 개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 모든 상수는 0~31 까지의 범위이고 무효로 할 백그라운드 C PLC 프로그램을 지정합니다.

Power PMAC 은 **disable bgcplc** 명령으로 후속 스캔 시작을 금지하고 지정된 백그라운드 C PLC 프로그램 실행을 정지합니다. 이 명령이 프로그램 도중에 실행을 중단(예를 들면 **sleep** 명령을 사용)한 경우에도 프로그램은 선두부터의 실행만 재개할 수 있습니다.

명령으로 지정된 C PLC 프로그램이 Power PMAC 에 존재하지 않는 경우에 에러가 통지되지 않습니다. 명령은 명령으로 지정된 다른 C PLC 프로그램에 작용합니다.

예

```
disable plc 0       // Disable execution of CPLC 0
disable plc 2,4,6    // Disable execution of CPLCs 2, 4, and 6
disable plc 7..10    // Disable execution of CPLCs 7, 8, 9, and 10
disable plc 11,13..16,20 // Disable execution of CPLCs 11, 13, 14, 15, 16, and 20
```

disable plc

기능: 지정된 PLC 프로그램 동작을 무효로 함

구문: **disable plc {목록}**

여기에서

- **{목록}**은 무효로 할 PLC 프로그램을 지정하는 하나 이상의 상수 세트 또는 0~31의 범위 상수입니다.

Power PMAC은 **disable plc** 명령으로 지정된 PLC 프로그램을 무효로(실행을 정지)합니다. 지정된 프로그램의 후속 스캔은 프로그램이 다시 유효가 될 때까지 금지됩니다. **enable plc** 명령으로 프로그램의 선두에서부터 실행이 재개됩니다.

PLC 무효/유효 상태는 스캔 시작 시에만 체크된다는 점에 주의하십시오. 이 명령에 따라 스캔 중에 PLC 프로그램 자체가 무효가 된 경우에 실행 중인 스캔은 완료합니다.

PLC 프로그램은 번호(0~31)에 따라 지정됩니다. 이 명령에서는 하나의 번호만 지정된 경우에 여러 번호가 쉼표로 구분되어 지정되는 경우와 연속된 번호의 변수 범위가 지정되는 경우가 있습니다.

예

```
disable plc 0;  
disable plc 1,2,5;  
disable plc 1..16;  
disable plc 3..6,8;
```

disable rticplc

기능: 포어그라운드 C PLC 프로그램을 무효로 함

구문: **disable rticplc**

Power PMAC은 **disable rticplc** 명령으로 후속 스캔 시작을 금지하고 실시간 인터럽트로 실행하는 포어그라운드 C PLC 프로그램 실행을 정지합니다. 이 명령이 프로그램 도중에 실행을 중단(예를 들면 **sleep** 명령을 사용)한 경우에도 프로그램은 선두부터의 실행만 재개할 수 있습니다.

포어그라운드 C PLC 프로그램이 Power PMAC에 존재하지 않는 경우에 에러가 통지되지 않습니다.

dkill

기능: 모터 지연 출력 정지를 전송함

구문: **dkill**[{목록}]

여기에서

- **{목록}**은 십표로 구분된 정수 상수이며 또는 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1을 뺀 값까지입니다.

dkill 명령은 주소 지정 또는 지정된 모터에 대해 지연 출력 정지를 실행합니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(Plc[i].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(Coord[x].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={식}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터가 명령을 받아 동시에 동작을 시작합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

모터를 출력 정지하면 서보 루프가 오픈되고 서보 출력값이 강제로 제로가 됩니다(단, 출력 바이어스 항은 적용됩니다). 다음으로 앰프 Enable 신호가 강제로 Disable 상태가 됩니다. Power PMAC 은 앰프 폴트 시나 편차 이상 시에는 자동으로 모터를 출력 정지합니다.

프로그램 **dkill** 모터 명령은 모션 프로그램을 실행 중인 좌표계 내의 모터에는 영향을 미치지 않습니다. 프로그램 실행을 먼저 정지해야 합니다. 단, 프로그램 **ddisable** 좌표계 명령은 동일하게 동작하지만 모션 프로그램을 실행하는 좌표계 내의 모터도 포함하여 모든 모터에 작용합니다.

[Motor[x].pBrakeOut]을 제로 이외의 값(브레이크 출력 레지스터의 주소)으로 설정하면 모터의 자동 브레이크 제어 기능이 유효한 경우에 브레이크 출력은 **dkill** 명령으로 즉시 연동하도록 명령됩니다. 게다가 모터가 클로즈드 루프, 제로 속도 상태인 경우에 모터의 실제 출력 정지는 [Motor[x].BrakeOnDelay] 밀리초만큼 지연되어 브레이크 시스템이 완전히 연동될 시간을 제공합니다.

이 지연 출력 정지 명령은 브레이크가 완전히 연동하기까지 시간을 제공하고 브레이크가 있는 모터의 계획된 출력 정지를 위한 것입니다. 모터의 긴급한 출력 정지는 마찬가지로 즉시 **kill** 명령을 사용해야 합니다.

일반적으로 이 명령은 PLC 프로그램에서만 사용하십시오. 모션 프로그램 표준 동작에서는 실행 중인 좌표계의 한 축에 할당된 모든 모터가 유효하고 클로즈드 루프이어야 합니다.

모터 출력 정지의 온라인 명령은 같은 형식 **dkill** 을 사용하며 모터 목록이 그 앞에 놓인다는 점(#1,2dkill 등)에 주의하십시오.

예

```
dkill;           // Acts on presently addressed motor
dkill1;          // Acts on Motor 1, regardless of addressed
dkill1,2,3;      // Acts on Motors 1, 2, and 3
dkill1..3,5..7;  // Acts on Motor 1, 2, 3, 5, 6, and 7
```

do

기능: 조건 루프를 시작함

구문: **do**

완전한 구조 구문:

```
.....do {명령} [{명령}...] while({조건})

           do {
.....     {명령}
.....     [{명령}...]
           }
.....while({조건})
```

do 명령은 **while** 명령으로 유효가 되는 조건 루프 내에서 실행하는 명령의 시작을 나타냅니다. **while** 명령의 조건은 루프의 매회 마지막으로 평가되기 때문에 동작은 항상 적어도 1 회 실행됩니다.

같은 프로그램행의 직후에 명령이 있는 경우에 루프 내에서 같은 행의 명령만 실행됩니다. 이 경우에 이러한 명령 뒤에는 동일한 행 또는 다음 프로그램행의 선두에 **while** 명령이 필요합니다.

do 명령 직후에 여는 중괄호({)가 있는 경우에는 같은 행이거나 다음 프로그램행의 선두이더라도 그 중괄호는 루프에서 실행하는 명령 시작을 나타냅니다. 이 명령의 실행은 닫는 중괄호(})가 있는 행까지 계속합니다. 닫는 괄호 뒤에는 동일한 행 또는 다음 프로그램행의 선두에 **while** 명령이 필요합니다.

예

```
do dwell 50 while (M1 == 0);           // Dwell until input M1 goes true

do {                                     // Start do loop
  X100 F20;                             // Move out
  dwell 50;                             // Dwell in "out" position
  X0 F10;                               // Move back
  dwell 50;                             // Dwell in "back" position
}                                       // Dwell in "out" position
while (P100 < 10);                     // Condition for continuing loop
```

dread

기능: 축 명령 위치를 통지함

구문: **dread**

Power PMAC 은 **dread** 명령으로 프로그램의 [Ldata.Coord]에 따라 주소 지정된 좌표계의 축 명령 위치를 계산합니다. 좌표계의 모든 활성 축의 명령 위치가 계산됩니다.

축의 현재 명령 위치는 좌표계 정의에 따라 처리되어(축 정의 구문을 반전하거나 포워드 키네마틱스 루틴의 사용 둘 중 하나) 대응할 모터의 명령 위치 및 유효한 매트릭스 변환에서 계산됩니다.

축 명령 위치의 값은 프로그램의 로컬 'D' 변수에 복사되어 후속 계산에서 사용됩니다. 위치는 유효한 축 변환 매트릭스를 사용한 현재 축의 단위로 복귀합니다. 각 축에 사용되는 'D' 변수 'D_i'는 다음 표와 같습니다.

축	D _i	축	D _i	축	D _i	축	D _i	축	D _i	축	D _i	축	D _i	축	D _i
A	0	V	4	Z	8	DD	12	HH	16	OO	20	SS	24	WW	28
B	1	W	5	AA	9	EE	13	LL	17	PP	21	TT	25	XX	29
C	2	X	6	BB	10	FF	14	MM	18	QQ	22	UU	26	YY	30
U	3	Y	7	CC	11	GG	15	NN	19	RR	23	VV	27	ZZ	31

'D32'는 32 비트의 마스크를 반환하고 명령 위치가 통지된 축을 통지합니다. 'D32'의 각 비트 i 는 변수가 통지된 변수 'D_i'의 번호에 해당합니다. 'D32'의 반환값이 0 인 경우에 위치는 통지되지 않습니다.

버퍼 프로그램 **dread** 명령은 온라인 **d** 명령과 같은 계산을 실행합니다.

dtogread

기능: 현재 실행 중인 동작 축의 남은 이동량을 통지함

구문: **dtogread**

Power PMAC 은 **dtogread** 명령으로 프로그램의 [Ldata.Coord]에 따라 주소 지정된 좌표계에서 실행 중인 축의 남은 이동량을 계산합니다. 저장 설정 요소 [Coord[x].TPSize]를 0 이상의 동작 계산 시와 동작 실행 시 사이의 모든 동작 위치를 저장하기 위한 충분한 값으로 설정하고 목표 위치의 버퍼를 유효로 해야 합니다. 좌표계에 정의된 모든 축의 남은 이동량이 계산됩니다.

축의 남은 이동량은 축의 목표 위치에서 현재 실행 중인 동작으로 그 축의 근접 명령 위치를 빼고 계산됩니다. X, Y, Z 축에 공구 직경 보정이 유효한 경우에 이 계산에 공구 직경 보정에 따른 목표 위치 오프셋 값이 사용됩니다. 축의 현재 명령 위치는 좌표계 정의에 따라 처리되어(축 정의 구문을 반전하거나 포워드 키네마틱스 루틴의 사용 둘 중 하나) 대응할 모터의 명령 위치 및 유효한 매트릭스 변환에서 계산됩니다. 이 차는 양이며 부호가 있는 값으로 통지됩니다.

남은 이동량은 프로그램의 로컬 'D' 변수에 복사되고 후속 계산에서 사용됩니다. 거리는 동작이 계산된 시점에서 유효한 축 매트릭스 변환을 사용하여 계산되어 반환됩니다. 각 축에 사용되는 'D' 변수 'D_i'는 다음 표와 같습니다.

축	D _i	축	D _i	축	D _i	축	D _i	축	D _i	축	D _i	축	D _i	축	D _i
A	0	V	4	Z	8	DD	12	HH	16	OO	20	SS	24	WW	28
B	1	W	5	AA	9	EE	13	LL	17	PP	21	TT	25	XX	29
C	2	X	6	BB	10	FF	14	MM	18	QQ	22	UU	26	YY	30
U	3	Y	7	CC	11	GG	15	NN	19	RR	23	VV	27	ZZ	31

D32는 32비트의 마스크를 반환하고 남은 이동량이 통지된 축을 통지합니다. 'D32'의 각 비트 *i*는 변수가 통지된 변수 'D_i'의 번호에 해당합니다. 'D32' 값은 [Coord[x].TPCoords]와 같은 값이어야 합니다. 'D32'의 반환값이 0인 경우에 거리는 통지되지 않습니다.

버퍼 프로그램 **dtogread** 명령은 온라인 **g** 명령과 같은 계산을 실행합니다.

dwel{데이터}

기능: 모든 축을 지정된 시간만큼 드웰시킴

구문: **dwel{데이터}**

여기에서

- **{데이터}**는 드웰 시간을 밀리초로 나타내는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

dwel 명령이 실행되면 모션 프로그램을 실행하는 좌표계의 모든 축에 지정 시간 동안 현재 명령 위치를 유지하도록 명령합니다. 접근하는 동작의 감속 시간 및 멀어지는 동작의 가속 시간은 이 시간에 포함되지 않습니다.

명령된 드웰 시간은 오버라이드 또는 타임 베이스값에 영향을 받지 않습니다. 글로벌 데이터 구조체의 저장 설정 요소 [Sys.ServoPeriod]에 따라 정의된 실시간으로 동작합니다. 따라서 **dwel0** 등의 **dwel** 구문을 사용하면 외부 타임 베이스 모드 중에는 마스터 입력과 동기되지 않습니다.

dwel 명령은 룩 어헤드 계산을 정지합니다. 드웰 시간이 실제로 종료할 때까지 모션 프로그램의 후속행은 평가되지 않습니다.

delay 명령과 비슷하지만 약간 다릅니다. 지연 시간에는 접근하는 동작의 감속 시간의 반 그리고 멀어지는 동작의 가속 시간의 반이 포함되지 않으며 오버라이드 및 타임 베이스값에 영향을 받습니다. 또한 룩 어헤드 계산을 금지하지 않습니다.

예

```
dwel 250;
dwel (P1+P2);
dwel 0;
```

else

기능: False 조건 분기를 시작함

구문: **else**

완전한 구조 구문:

```
else {명령}{명령}...

.....else {
..... {명령}
..... [{명령}...]
..... }
```

else 명령은 선행하는 **if** 조건 평가가 **False** 인 경우에 실행하는 명령의 시작을 나타냅니다. **else** 명령 앞에 대응하는 **if** 명령이 필요하지만 **if** 명령 뒤에 반드시 **else** 명령이 필요한 것은 아닙니다.

if 조건이 **False** 일 때에 같은 프로그램행의 직후에 명령이 있는 경우에는 같은 행의 명령만 실행됩니다.

같은 프로그램행의 뒤에 명령이 없는 경우에 조건 **False** 에서 실행하는 명령의 시작을 나타내기 위해서 다음 프로그램행은 여는 중괄호(**{**)로 시작해야 합니다. 이 명령의 실행은 닫는 중괄호(**}**)가 있는 행까지 계속합니다. **else** 명령과 같은 행에 여는 중괄호를 두어도 무방합니다.

예

```
if (M1 == 0) P20++;           // Increment P20 if input M1 false
else P20--;                   // Otherwise decrement P20

if (P500 > 0) P500--;          // Decrement P500 if greater than 0
else {                         // Start of branch for P500 <= 0
  X50 tm200;                   // Execute X-axis move
  dwell 75;                    // Dwell for 75 msec
  P500 = 50;                   // Set P500 to 50
}                               // End of branch for P500 <= 0
```

enable

기능: 좌표계의 모터 모두를 유효로 함

구문: **enable**{**{목록}**}

여기에서

- **{목록}**은 씬표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.

enable 명령을 사용하면 Power PMAC은 주소 지정 또는 지정된 좌표계에 정의되고 현재 Disable된 모든 모터의 클로즈드 루프 서보 제어를 유효로 합니다.

좌표계 목록이 없는 경우에 현재 프로그램에 따라 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 목록 내의 좌표계를 지정해도 프로그램에 따라 모달에 주소 지정된 좌표계에는 영향을 미치지 않습니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 `[Plc[i].Ldata.Coord]`에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={x}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(`Coord[x].Ldata.Coord`)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

서보 제어 유효화는 앰프 Enable 상태의 'True'(유효) 설정, 위치/속도 서보 루프의 클로즈로 구성됩니다. 모터는 출력 정지된 상태 또는 오픈 루프 유효 상태 어느 쪽에서든 최종적으로 유효화되고 클로즈드 루프, 제로 명령 속도 상태가 됩니다.

enable 명령은 오픈 루프 또는 클로즈드 루프와 관계없이 이미 유효 상태인 모터에는 영향을 미치지 않습니다(V1.3 이전 펌웨어 버전에서는 오픈 루프 유효 모드로 루프를 클로즈합니다).

예

```
enable;           // Enable all motors in presently addressed C.S.
enable4;          // Enable all motors in C.S. 4
enable5..8;       // Enable all motors in C.S. 5, 6, 7, & 8
```

enable bgcplc

기능: 지정된 백그라운드 C PLC 프로그램을 유효로 함

구문: **disable bgcplc {목록}**

여기에서

- {목록}**은 쉼표로 구분된 정수 상수 또는 시작 상수에서 종료 상수 사이를 2 개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 모든 상수는 0~31 까지의 범위이고 무효로 할 백그라운드 C PLC 프로그램을 지정합니다.

Power PMAC 은 **enable bgcplc** 명령으로 보통의 우선 순위와 타이밍에 따라 지정된 백그라운드 C PLC 프로그램 실행을 허가합니다. 이 명령이 프로그램 도중에 실행을 중단(예를 들면 **sleep** 명령을 사용)한 경우에도 프로그램은 선두부터의 실행만 재개할 수 있습니다.

명령으로 지정된 C PLC 프로그램이 Power PMAC 에 존재하지 않는 경우에 에러가 통지되지 않습니다. 명령은 명령으로 지정된 다른 C PLC 프로그램에 작용합니다.

예

```
enable plc 0           // Enable execution of CPLC 0
enable plc 2,4,6       // Enable execution of CPLCs 2, 4, and 6
enable plc 7..10       // Enable execution of CPLCs 7, 8, 9, and 10
enable plc 11,13..16,20 // Enable execution of CPLCs 11, 13, 14, 15, 16, and 20
```

enable plc

기능: 지정된 PLC 프로그램 동작을 유효로 함

구문: **enable plc {목록}**

여기에서

- **{목록}**은 유효로 할 PLC 프로그램을 지정하는 하나 이상의 상수 세트 또는 0~31의 범위 상수입니다.

Power PMAC 은 **enable plc** 명령으로 지정된 PLC 프로그램의 보통 실행 순서를 유효로(연속 실행을 시작) 합니다. 지정된 각 PLC 프로그램은 프로그램 선두부터 실행을 시작합니다. PLC 프로그램은 번호(0~31)에 따라 지정됩니다. 이 명령에서는 하나의 번호만 지정된 경우에 여러 번호가 쉼표로 구분되어 지정되는 경우와 연속된 번호의 변수 범위가 지정되는 경우가 있습니다.

예

```
enable plc 0;
enable plc 1,2,5;
enable plc 1..16;
enable plc 3..6,8;
```

enable rticplc

기능: 포어그라운드 C PLC 프로그램을 유효로 함

범위: 글로벌

구문: **enable rticplc**

Power PMAC 은 **enable rticplc** 명령으로 보통의 우선 순위와 타이밍에 따라 실시간 인터럽트로 실행되는 포어그라운드 C PLC 프로그램 실행을 허가합니다. 이 명령이 프로그램 도중에 실행을 중단(예를 들면 **sleep** 명령을 사용)한 경우에도 프로그램은 선두부터의 실행만 재개할 수 있습니다.

포어그라운드 C PLC 프로그램이 Power PMAC 에 존재하지 않는 경우에 에러가 통지되지 않습니다.

F{데이터}

기능: 벡터 피드 레이트 지정

구문: **F{데이터}**

여기에서

- **{데이터}**는 벡터 피드 레이트를 시간 사용자 단위당의 길이 사용자 단위로 나타내는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

F 명령은 후속 직선 보간 모드 동작 및 원호 보간 모드 동작에서 사용하는 벡터 피드 레이트(벡터 크기)를 부동 소수점의 분해능으로 지정합니다. 좌표계는 동작 시간 모드가 아니라 피드 레이트 모드에서 동작합니다. 양의 값을 지정해야 합니다. 이미 지정된 **tm** 또는 **F** 값을 오버라이드합니다. 또한 이 명령 후에 **tm** 또는 **F** 구문에 따라 오버라이드됩니다.

모션 프로그램에서 **F** 명령을 실행하면 Power PMAC 은 [Coord[x].Tm]을 지정된 값의 음의 값으로 변경합니다(마이너스 기호는 좌표계가 동작 시간 모드가 아니라 피드 레이트 모드임을 나타냅니다). **F** 명령은 PLC 프로그램에서는 'no op'가 됩니다. 직접 PLC 프로그램에서 좌표계에 피드 레이트값을 설정하는 경우에 [Coord[x].Tm]에 직접 쓰십시오.

F 명령으로 지정한 속도 단위는 스케일링된 시간 단위(축 정의에 따라 설정)당 스케일링된 위치 단위(좌표계의 데이터 구조체 설정 요소 [Coord[x].FeedTime] – Isx90에 따라 정의)입니다.

여기서 지정된 속도는 좌표계의 벡터 피드 레이트 축의 벡터 속도입니다. 즉 동작 시간은 피드 레이트 축의 벡터 거리(각축의 제곱합의 제곱근)를 여기에서 지정한 피드 레이트의 값으로 나누어 계산합니다. 같은 동작 명령행에서 동작을 명령된 피드 레이트 이외의 축은 같은 시간에 종료할 수 있는 속도로 동작합니다.

축은 **frax** 명령으로 벡터 피드 레이트 축으로 지정됩니다. **frax** 명령이 없는 경우에 기본값인 피드 레이트 축은 X, Y 및 Z 축입니다. 원호 보간되는 축은 최신 **frax** 명령에 의한 지정 여부와 관계없이 자동으로 피드 레이트 축이 됩니다. 여러 축의 좌표계에서는 동작 피드 레이트 지정이 직교 배치 좌표계에서만 유용합니다. 직교 배치 좌표계에서는 이동 방향과 관계없이 평면 또는 3D 공간에서 일정 속도 동작을 하기 위함입니다.

**참고사항**

저장하지 않는 설정 요소 [Coord[x].InvTimeMode]가 제로 이외의 값으로 설정된 경우에 좌표계는 역타임 모드가 됩니다. 또한 **F** 로 지정된 값은 벡터 속도가 아니라 동작 시간의 역수를 지정합니다. 자세한 내용은 [Coord[x].InvTimeMode] 설명을 참조하십시오.

F 명령으로 프로그램된 피드 레이트가 좌표계의 데이터 구조체 설정 요소 [Coord[x].MaxFeedrate](Isx98)에 따라 지정된 제한을 초과한 경우에 대신에 제한값이 사용됩니다.

벡터 시간과 피드 레이트로부터 계산한 동작 시간이 가속 시간보다 짧은 경우에 가속 시간에 맞춰서 동작 시간이 증가합니다. 그 결과 축 거리를 동작 시간으로 나누어 결정한 속도보다도 축 속도가 느려집니다.

동작 거리를 동작 시간으로 나누어 결정한 좌표계 모터의 요구 속도가 데이터 구조체 요소 [Motor[x].MaxSpeed](활성일 때)에 따라 설정된 제한을 초과한 경우에 제한을 위반하지 않도록 동작 시간이 연장됩니다.

벡터 피드 레이트 축과 피드 레이트가 아닌 축이 같은 프로그램행에서 동시에 명령되어 좌표계의 데이터 구조체 설정 요소 [Coord[x].AltFeedrate](Isx86)에서 축 거리를 나누어 계산된 피드 레이트가 아닌 축의 시간이 벡터 피드 레이트 축에 계산된 동작 시간을 초과하는 경우에 긴 시간 쪽이 동작에 적용됩니다. 결과적으로 유효 벡터 피드 레이트는 느려집니다.

피드 레이트 지정 동작에서 피드 레이트가 아닌 축만 해당하는 동작이 명령된 경우에 Power PMAC 은 축에 명령된 것 중에서 가장 거리를 좌표계의 데이터 구조체 설정 요소 [Coord[x].AltFeedrate]로 나누어 동작 시간을 계산합니다. 이 파라미터가 0으로 설정된 경우에 가장 거리를 프로그램된 피드 레이트로 나누어 동작 시간을 계산합니다.

예

```
F100  
F31.25  
F(Q10)  
F(sin(P8*P9))
```

frax

기능: 벡터 피드 레이트 축 지정

구문: **frax**[{축 목록}]

여기에서

- **{축 목록}**은 쉼표로 나눈 축 명칭(1 문자 또는 2 문자) 및/또는 시작 축 명칭에서 종료 축 명칭 사이를 마침표 2 개로 나타내는 연속한 축 범위입니다. 사용되는 모든 축 명칭은 알파벳 순이며 먼저 1 문자의 이름을 지정하고 그 다음에 2 문자의 이름을 지정해야 합니다.

frax 명령은 다음의 지정된 피드 레이트(F)로 동작하기 위한 벡터 피드 레이트(속도) 계산에 관련된 축을 지정합니다. Power PMAC 은 모든 피드 레이트 축의 벡터 거리(축 거리의 제곱합의 제곱근)를 피드 레이트로 나누고 이러한 동작에 필요한 시간을 계산합니다. 같은 명령행에서 지정된 피드 레이트가 아닌 축은 같은 시간 내에 동작을 종료시키고 그 시간 내에 몇 개의 거리를 동작하기 위해 필요한 속도로 이동합니다.

벡터 피드 레이트는 분명하게 직교 좌표계에서만 기하학적인 의미를 가집니다. 직교 좌표계에서는 방향과 관계없이 공구 속도가 일정합니다. 단, 벡터 피드 레이트는 직교가 아닌 좌표계 및 3 축 이상의 좌표계에 대해 지정할 수 있습니다.

각 좌표계의 기본값인 벡터 피드 레이트 축은 X, Y 및 Z 축입니다. 이 설정은 거의 변경할 필요가 없습니다.



참고사항

피드 레이트 지정 동작에서 데이터 구조체 설정 요소 [Coord[x].AltFeedrate](Isx86)로 축 거리를 나누고 계산된 피드 레이트가 아닌 축의 동작 시간이 벡터 거리를 피드 레이트로 나누어 계산된 피드 레이트 축의 동작 시간을 초과한 경우에 Power PMAC 은 피드 레이트가 아닌 축의 동작 시간을 적용합니다.

인수를 가지지 않는 **frax** 명령은 후속 동작 명령에서 좌표계 X, Y, Z 축만을 피드 레이트 축으로 설정합니다. 인수를 가지는 **frax** 명령은 후속 동작 명령에서 지정된 축을 피드 레이트 축으로 설정하고 그 이외의 축을 피드 레이트가 아닌 축으로 설정합니다. 이 명령 내에 공백은 사용할 수 없습니다.

frax 명령은 피드 레이트 축의 32 비트 데이터 구조체 요소 [Coord[x].FRAxes] 비트를 1 로 설정하고 비트 레이트가 아닌 축의 비트를 0 으로 설정합니다.

이 명령이 Power PMAC 으로 전송됐을 때 모션 프로그램 버퍼가 오픈되어 있지 않은 경우에 온라인 좌표계 명령으로 실행됩니다.

예

```
frax(X,Y)
inc
X30 Y40 Z10 F100
```

벡터 거리는 $\sqrt{30^2 + 40^2} = 50\text{mm}$ 입니다. 100mm/초일 때 동작 시간(블렌드가 아닌)은 0.5 초입니다. X 축 속도는 $30/0.5 = 60\text{mm/초}$, Y 축 속도는 $40/0.5 = 80\text{mm/초}$, Z 축 속도는 $10/0.5 = 20\text{mm/초}$ 입니다.

Z20

벡터 거리는 $\sqrt{0^2 + 0^2} = 0\text{mm}$ 입니다. [AltFeedrate] = 50(mm/초)일 때 Z 축 속도는 50mm/초, 동작 시간(블렌드가 아닌)은 0.4 초입니다.

```
frax(X,Y,Z)
inc
X-30 Y-40 Z120 F65
```

벡터 거리는 $\sqrt{-30^2 + -40^2 + 120^2} = 130\text{mm}$ 입니다. 동작 시간은 $130/65 = 2.0$ 초입니다. X 축 속도는 $30/2.0 = 15\text{mm/초}$, Y 축 속도는 $40/2.0 = 20\text{mm/초}$, Z 축 속도는 $120/2.0 = 60\text{mm/초}$ 입니다.

fread

기능: 축 위치 편차를 통지함

구문: **fread**

Power PMAC 은 **fread** 명령으로 프로그램의 [Ldata.Coord]에 따라 주소 지정된 좌표계의 축 위치 편차를 계산합니다. 좌표계의 모든 활성 축의 위치 편차가 계산됩니다.

축의 현재 위치 편차는 좌표계 정의에 따라 처리되어(축 정의 구문을 반전하거나 포워드 키네마틱스 루틴을 사용하여) 대응할 모터의 편차 및 유효한 매트릭스 변환으로 계산됩니다. 포워드 키네마틱스 루틴을 사용하는 경우에 명령 위치와 현재 위치를 사용하여 편차를 계산할 수 있도록 더블 버스를 실행할 수 있어야 합니다.

축 위치 편차값은 프로그램의 로컬 'D' 변수에 복사되어 후속 계산에서 사용됩니다. 속도는 유효한 축 변환 매트릭스를 사용한 현재 축의 단위로 복귀합니다. 각 축에 사용되는 'D' 변수 'D_i'는 다음 표와 같습니다.

축	D _i	축	D _i	축	D _i	축	D _i	축	D _i	축	D _i	축	D _i	축	D _i
A	0	V	4	Z	8	DD	12	HH	16	OO	20	SS	24	WW	28
B	1	W	5	AA	9	EE	13	LL	17	PP	21	TT	25	XX	29
C	2	X	6	BB	10	FF	14	MM	18	QQ	22	UU	26	YY	30
U	3	Y	7	CC	11	GG	15	NN	19	RR	23	VV	27	ZZ	31

'D32'는 32 비트의 마스크를 반환하고 위치 편차가 통지된 축을 통지합니다. 'D32'의 각 비트 *i* 는 변수가 통지된 변수 'D_i'의 번호에 해당합니다. 'D32'의 반환값이 0 인 경우에 위치는 통지되지 않습니다.

버퍼 프로그램 **fread** 명령은 온라인 **f** 명령과 같은 계산을 실행합니다.

G{데이터}

기능: 표준 코드(G 코드)

구문: **G{데이터}**

여기에서

- {데이터}**는 코드 번호 및 그 코드를 탑재하는 서브루틴의 행 점프 레이블을 나타내는 0.000~999.999 의 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

Power PMAC 은 G 코드를 **call{Coord[x].Gprog}.{데이터}*1000** 명령(**{데이터}**는 G 코드의 번호)으로 해석합니다. 이 명령에 따라 코드 번호를 1,000 배 한 번호의 행 점프 레이블 위치에서 [Coord[x].Gprog]의 값에 따라 번호가 지정된 서브 프로그램으로 점프합니다. 예를 들어 [Coord[x].Gprog]의 기본값이 1000 인 경우에 **subprog 1000** 의 행 점프 레이블 **N3000**:에서 **G03** 이 호출됩니다. 프로그램 실행은 다음 **return** 명령에서 호출자 프로그램으로 복귀합니다.

이 구조체에 따라 커스터마이징 가능한 G 코드 루틴을 CAD/CAM 프로그램의 RS-274 대응 코드 및 CNC 스타일 어플리케이션에 탑재할 수 있습니다. 호출자 프로그램 G 코드의 뒤에 1 세트 이상의 **{문자}{데이터}**를 두면 해당 서브루틴에 인수를 전달할 수 있습니다. 각 문자의 뒤의 값은 **read** 명령을 사용하여 서브루틴으로 취득할 수 있습니다. 모든 좌표계의 [Coord[x].Gprog] 기본값은 1,000 입니다. 따라서 모든 좌표계는 기본값으로 같은 G 코드 세트를 공유합니다. 단, 다른 좌표계에 다른 값을 할당하면 다른 코드를 탑재할 수 있습니다.

일반적으로 G 코드는 RS-274 프로그램에서 표준 코드로 사용되며 동작 모드 등을 지정합니다. G 코드의 코어 세트는 0~99 의 정수를 사용하지만 이 스킴에서는 999.999 까지의 소수 코드 번호를 사용할 수 있고 코어 번호에 수식을 사용할 수 있습니다.

모든 종류의 서브루틴 및 서브 프로그램 호출(**call**, **callsub**, **gosub**, **G**, **M**, **T**, **D**)은 255 레벨(최상위를 포함하여 256)로 네스트할 수 있습니다. 무한 루프는 사용하지 마십시오.

G 코드 명령에 따라 지정된 점프 행 레이블이 프로그램에 존재하지 않는 경우에 지정된 G 코드의 서브 프로그램 선두로 점프합니다. 에러는 발생하지 않습니다. 이를 이용하여 사용자 어플리케이션이 존재하지 않는 G 코드를 처리하는 방법을 결정할 수 있습니다.

예

G01	// Call to label N1000: of G-code subprogram
G54.1	// Call to label N54100: of G-code subprogram
G122	// Call to label N122000: of G-code subprogram

gosub{데이터}

기능: 프로그램 내에서 점프하고 복귀함

구문: **gosub{데이터}**

여기에서

- {데이터}는 목적지 행의 점프 레이블을 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

gosub 명령에 따라 {데이터}에 지정된 행 점프 레이블(N{상수} 뒤에 콜론이 붙음)까지 모션 프로그램 실행이 점프됩니다. 그 후에 다음 **return** 명령에 도달할 때까지 **gosub** 직후의 명령으로 복귀합니다. {데이터} 식의 값을 완전한 정수로 평가할 수 없는 경우에는 다음 정수로 버리면 됩니다. 상수값은 0~999,999 범위의 음의 값이 아닌 정수이어야 합니다. 그 이외의 값인 경우에 다운로드 시에 명령은 구문 에러로 거부됩니다.

{데이터}가 상수인 경우에 프로그램 실행 전에 프로그램 레이블로 궤적이 링크되어 있기 때문에 점프가 매우 빨라집니다. {데이터}가 변수 식인 경우에 실행 시에 평가하며 적절한 레이블을 검색합니다. 이 방법은 처음 방법보다 시간이 약간 더 걸립니다.

모든 종류의 서브루틴 및 서브 프로그램 호출(**call**, **callsub**, **gosub**, **G**, **M**, **T**, **D**)은 255 레벨(최상위를 포함하여 256)로 네스트할 수 있습니다. 무한 루프는 사용하지 마십시오.

프로그램 입력 시에 항목이 클로즈 명령으로 종료되기까지의 프로그램행에 **gosub{상수}** 명령으로 지정된 점프 레이블 번호가 설명되어 있지 않은 경우에 에러가 생성되어 프로그램 버퍼가 클리어됩니다.

프로그램 실행 시에 식의 값이 계산된 **gosub** 명령으로 지정된 점프 행 레이블이 프로그램에 존재하지 않을 경우에 프로그램 실행은 에러로 인해 **gosub** 명령에서 정지합니다.

gosub 명령으로는 서브루틴과의 사이에서 로컬 스택 변수를 사용하는 인수를 전달할 수 없습니다. 대신에 **callsb** 또는 **call** 명령을 사용하십시오.

gosub{데이터} 구문의 뒤에 문자 및 데이터(**X1000 DD50** 등)가 있는 경우에 이러한 값들은 서브 프로그램으로 전달하는 인수입니다. 이 인수를 사용하여 **RS-274** 문자/번호 구문의 프로그램이 서브루틴을 사용할 수 있습니다. 인수를 전달하는 경우에 서브루틴에서 실행되는 최초의 행에 **read** 명령을 사용하십시오. 이 명령은 지정된 문자와 연관된 값을 취득하고 그 값을 문자와 일치하는 적절한 'D' 변수에 넣습니다.

예를 들면 A 뒤의 값은 로컬 변수 'D1', B 뒤의 값은 로컬 변수 'D2', Z 뒤의 값은 로컬 변수 'D26'에 들어갑니다. 2 문자 AA 뒤의 값은 'D27', BB' 뒤의 값은 로컬 변수 'D28', ZZ 뒤의 값은 'D52'에 들어갑니다. 그 후에 서브 프로그램이 변수값을 사용할 수 있습니다. 각 좌표계(최상위 프로그램이 모션 프로그램인 경우) 및 각 PLC 프로그램에 'D' 변수 세트는 하나뿐입니다. **read** 명령을 사용한 서브루틴/서브 프로그램 네스트가 있는 경우에 매회 같은 'D' 변수 세트가 사용됩니다. 자세한 내용은 **read** 명령 사양을 참조하십시오.

예

```
gosub 400;           // Call to jump label N400: of this program
gosub (Q123);        // Call to jump label numbered by value of Q123
gosub 5000 E5;        // Call to jump label N5000:, ready to pass E value
```

goto{데이터}

기능: 프로그램 내에서 점프하고 복귀하지 않음

구문: **goto{데이터}**

여기에서

- **{데이터}**는 목적지 행의 점프 레이블을 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

goto 명령에 따라 **{데이터}**에 지정된 행 점프 레이블(**N{상수}** 뒤에 콜론이 붙음)까지 모션 프로그램 또는 PLC 프로그램 실행이 점프됩니다. 그 뒤에 원래 위치로 복귀하지 않습니다. **{데이터}** 식의 값을 완전한 정수로 평가할 수 없는 경우에는 다음 정수로 버리면 됩니다. 상수값은 0~999,999 범위의 음의 값이 아닌 정수이어야 합니다. 그 이외의 값인 경우에 다운로드 시에 명령은 구문 에러로 거부됩니다.

{데이터}가 상수인 경우에 프로그램 실행 전에 프로그램 레이블로 궤적이 링크되어 있기 때문에 점프가 매우 빨라집니다. **{데이터}**가 변수 식인 경우에 실행 시에 평가하며 적절한 레이블을 검색합니다. 이 방법은 처음 방법보다 시간이 약간 더 걸립니다.

goto 명령으로 지정된 점프 행 레이블이 프로그램에 존재하지 않을 경우에 프로그램 실행은 에러로 인해 **goto** 명령에서 정지합니다.



참고사항

현대 컴퓨터의 최적 구조화 개념에서는 **goto** 를 지양합니다. 코드를 해독하지 못하는 경향이 있기 때문입니다.

예

```

gosub 400;           // Jump to jump label N400: of this program
gosub (Q123);        // Jump to jump label numbered by value of Q123

```

hold

기능: 피드 홀드를 실행함

구문: **hold**[{목록}]

여기에서

- **{목록}**은 쉽표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.

hold 명령으로 좌표계 타임 베이스를 제로로 하고 주소 지정 또는 목록화된 좌표계의 모션 프로그램 실행을 일시 정지하며 궤적에 따른 시작을 즉시 감속시킵니다.

좌표계 목록이 없는 경우에 현재 프로그램에 따라 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 목록 내의 좌표계를 지정해도 프로그램에 따라 모달에 주소 지정된 좌표계에는 영향을 미치지 않습니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 [Plc[i].Ldata.Coord]에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={x}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(Coord[x].Ldata.Coord)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

피드 홀드 모드 중에 모션 프로그램 실행은 일시 정지하지만 논리적으로는 아직 실행 중인 상태입니다. 일시 정지 후에도 프로그램 실행을 재개하지 않는 경우에 **Abort** 명령을 사용하여 프로그램 실행을 완전히 **Abort** 해야 합니다.

감속 및 그 후의 재가속이 [Coord[x].TimeBaseSlew](Isx94)가 아니라 [Coord[x].FeedHoldSlew](Isx95)로 제어되는 경우를 제외하고 결과적으로 **hold** 명령은 **%0** 명령과 아주 유사합니다. 또한 **%100** 대신에 **run** 또는 **step** 명령을 사용하여 실행을 재개할 수 있습니다. 그리고 **hold** 명령은 **%0** 명령을 실행할 수 없는 외부 타임 베이스에서도 기능합니다.

일반적으로 **hold** 명령으로는 모션이 프로그램 포인트에서 정지하지 않습니다. **run** 또는 **step** 명령을 사용하면 궤적에 따른 전체 속도로의 실행이 재개됩니다. 전체 타임 베이스값(내부 설정 또는 외부 설정에 관계없이)까지의 변화는 [Coord[x].FeedHoldSlew]로 설정되는 속도가 됩니다. 전체 타임 베이스값에 도달하면 [Coord[x].TimeBaseSlew]는 임의의 타임 베이스로 변경할 수 있습니다.

상당하는 온라인 명령은 생략형 **h** 를 사용하며 그 앞에 좌표계 목록이 놓인다는 점(&1,2h 등)에 주의하십시오.

예

<code>hold;</code>	<code>// Do feed hold in presently addressed C.S</code>
<code>hold1;</code>	<code>// Do feed hold in C.S.1</code>
<code>hold3,5..7;</code>	<code>// Do feed hold in C.S. 3, 5, 6, & 7</code>

home

기능: 원점 서치 동작을 실행함

구문: **home**{**목록**}

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수이며 또는 시작 상수에서 종료 상수까지의 사이를 2 개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0 부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1 을 뺀 값까지입니다.

home 명령은 주소 지정 또는 지정된 모터에 대해 원점 서치 동작을 실행시킵니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(Plc[i].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(Coord[x].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={x}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터가 명령을 받아 동시에 동작을 시작합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램의 다음 명령이 실행되기 전에 트리거 후 오프셋을 포함한 원점 서치 동작 전체 또는 동작 세트를 완료해야 합니다. 최상위 프로그램이 PLC 프로그램인 경우에 원점 서치 동작 시작 후에 바로 프로그램의 다음 명령이 실행됩니다.

원점 서치의 속도 및 방향은 설정 데이터 구조체 요소 [Motor[x].HomeVel]에 따라 결정됩니다. 원점 서치의 가속도 프로파일은 설정 데이터 구조체 요소 [Motor[x].JogTa] 및 [Motor[x].JogTs]에 따라 결정됩니다.

원점 서치 동작의 온라인 명령은 생략형 **hm** 을 사용하며 모터 목록이 그 앞에 놓인다는 점(&1,2hm 등)에 주의하십시오.

예

```

home;           // Home presently addressed motor
home1;          // Home Motor 1
home1,2,3;      // Home Motors 1, 2, & 3
home1..3,5..7;  // Home Motors 1, 2, 3, 5, 6, 7

```

homez

기능: 제로 동작 원점 복귀를 프로그램함

구문: **homez[{목록}]**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수이며 또는 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1을 뺀 값까지입니다.

homez 명령은 주소 지정 또는 지정된 모터에 대해 제로 동작 원점 복귀 동작을 실행시킵니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(Plc[i].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(Coord[x].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={식}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터가 명령을 받아 동시에 동작을 시작합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

갠트리 종축 모드에서 모터가 활성인([Motor[x].ServoCtrl] = 8) 경우에 **homez** 명령을 실행하면 원점 서치 동작의 갠트리 주축 모터를 추적하면서 원점 트리거 위치(오프셋 있음)까지 모터가 저속으로 이동합니다. 갠트리 모터 사이의 전원 셋업의 기울기를 삭제할 수 있습니다.

[Motor[x].pAbsPos]를 기본값 0으로 설정하면 **homez** 명령에 따라 Power PMAC은 단순히 현재 명령 위치로부터의 동작 없이 원점(모터 제로)으로 재정의합니다. 명령 시에 위치 편차가 있으면 결과의 현재 위치는 위치 편차의 음의 값이 됩니다.

단, [Motor[x].pAbsPos]를 제로 이외의 값으로 설정하면 Power PMAC은 모터의 현재 절대 위치값에 따라 지정되는 주소를 가지는 레지스터를 읽습니다. 이 주소에서의 위치 데이터 읽기는 [Motor[x].AbsPosFormat], [Motor[x].AbsPosSf] 및 [Motor[x].HomeOffset] 설정에 따라 처리됩니다.

원점 서치 동작의 온라인 명령은 생략형 **hmz**를 사용하며 모터 목록이 그 앞에 놓인다는 점(#1,2hmz 등)에 주의하십시오.

예

```
homez;           // Zero-move home of presently addressed motor
homez1;          // Zero-move home of Motor 1
homez1,2,3;       // Zero-move home of Motors 1, 2, & 3
homez1..3,5..7;   // Zero-move home of Motors 1, 2, 3, 5, 6, 7
```

{데이터}{대입 연산자}{식}

기능: 지정된 'I' 변수에 값을 할당함

구문: {데이터}[{식}]{식}
 {대입 연산자}{식}

여기에서

- {데이터}는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 값을 할당하는 첫 번째 변수 번호를 나타내는 소괄호 내의 수식입니다.
- 첫 번째 옵션 {식}은 값을 할당하는 변수의 수를 지정합니다. 명확하게 지정되지 않은 경우에는 1 이 사용되고 값은 하나의 변수에만 할당됩니다.
- 두 번째 옵션 {식}은 값을 할당하는 변수의 숫자 간격을 지정합니다. 명확하게 지정되지 않은 경우에는 1 이 사용되고 값은 연속된 번호의 변수에 할당됩니다.
- {대입 연산자}는 식에서 값을 변수에 할당하는 방법을 제어하는 숫자 연산자입니다.
- 마지막 {식}은 상수, 변수, 논리 및 산술 연산자, 함수를 조합한 수식이며 도출된 식의 값이 할당에 사용됩니다.

{데이터}{대입 연산자}{식} 명령에 따라 지정된 'I' 변수에 식에서 구한 값이 들어갑니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수 번호를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~16,383). 유효한 변수 번호를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 구해져야 합니다(필요에 따라 다음 정수로 버림). 구해지지 않는 경우에 명령은 실행하지 않는 것으로 처리됩니다.

'I' 변수는 이전 Turbo PMAC 과 기능이 일치하는 저장 설정 데이터 구조체 요소에 생략명을 제공합니다. 이에 따라 Turbo PMAC 에 정통한 사용자는 그 'I' 변수 번호를 Power PMAC 에 사용할 수 있습니다. 또한 'I' 변수명에 의한 저장 데이터 구조체 요소의 참조는 전혀 필요 없습니다.

지정된 'I' 변수에 기본이 되는 데이터 구조체 요소([Motor[1].JogSpeed] 등)의 이름을 사용하여 기본적인 문자/번호 형식(I122 등)의 이름을 직접 할당할 수 있습니다. 또한 IDE 소프트웨어 사용 시에는 #define 텍스트의 대체(Mtr1Vj 등)로 변수에 이름을 할당합니다. 마지막으로 기본적인 문자/번호 형식의 변수명으로 대체하는 것은 다운로드 시 자동으로 이루어집니다.

동일한 간격으로 나열되는 여러 변수에 같은 값을 할당할 수 있습니다. 초기 변수명 뒤에 두 번째 값이 사용되는 경우에 값이 할당되는 변수의 수를 지정합니다(두 번째 값이 사용되지 않는 경우에 기본값인 1 이 되고 초기 변수가 지정됩니다). 세 번째 값이 사용되는 경우에 변수의 간격을 수치로 지정합니다(세 번째 값이 사용되지 않는 경우에 기본값인 1 이 되며 연속된 번호의 변수가 사용됩니다).

보통 연산자 뒤에 있는 도출된 식은 표준적인 대입 연산자를 사용하여 변수에 직접 대입할 수 있습니다. 단, 이 값은 표준적인 대입 연산자 `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`, `>>=`, `<<=`를 사용하여 산술적 또는 논리적으로 변수의 기존 값과 조합할 수도 있습니다. 또한 증가 및 감소 연산자 `++` 및 `--`는 뒤에 식이 없어도 사용할 수 있습니다.

변수의 실제 할당을 프로그램의 다음 명령 동작이 실제로 실행 시작할 때까지 지연되는 경우에 동기 대입 연산자를 사용할 수 있습니다. 모션 프로그램에서 여러 동작의 블렌드 또는 스플라인 시퀀스 중에 출력을 설정할 때 도움이 됩니다. 모션 프로그램에서는 동작을 어떻게 블렌드 또는 스플라인할지 구하기 위해 하나 이상의 선행 동작을 계산해야 하기 때문입니다. 표준적인 대입 연산자에서는 값이 설정되는 타이밍이 너무 빠릅니다. 동기 할당을 설정하면 다음 동작이 실행 시작할 때까지 실제 할당을 지연할 수 있기 때문에 직관적으로 기대되는 시간에 출력을 실시합니다.

또한 동기 할당은 동작 종료를 확인하기 전에 명령된 동작이 시작되도록 모터 또는 축 동작을 직접 명령하는 PLC 프로그램에서도 도움이 됩니다. 모터 동작에서는 모터를 좌표계에 할당해야 한다는 점에 주의하십시오. 지연 할당을 유지하는 동기 대입 규는 좌표계에 속하기 때문입니다.

보통 명령 시에 구한 식의 값은 `==` 동기 대입 연산자를 사용하여 다음 동작의 실행 시작 시에 변수에 직접 대입됩니다. 단, 이 값은 산술 동기 변수 대입 연산자 `+=`, `-=`, `*=`, `/=`, `%=` 또는 논리 동기 대입 연산자 `&=`, `|=`, `^=`를 사용하여 산술적 또는 논리적으로 변수의 기존 값과 조합할 수도 있습니다. 또한 동기 증가 및 감소 연산자 `++=` 및 `--=`는 뒤에 식이 없어도 사용할 수 있습니다.

산술 동기 대입 연산자는 부동 소수점인 'f' 변수만으로 사용할 수 있고 논리 동기 대입 연산자는 고정 소수점(정수)인 'i' 변수만으로 사용할 수 있다는 점에 주의하십시오.

예

```
I122=17.5;           // Set variable I122 to 17.5
I(L10)+=3;           // Add 3 to I-variable numbered by L10
I(P10+P17)=5*sqrt(P100); // Set I-var numbered by (P10+P17) to expression value
I290==1;             // Set variable I290 to 1 at start of next move execution
I388+=;              // Increment I388 at start of next move execution
I122,3,I100=50;      // Set I122, I222, I322 to 50
```

if({조건})

기능: 조건 분기

구문: `if({조건})`

여기에서

- **{조건}**은 단일 또는 복합 조건 또는 조건 비교 연산자가 없는 수식입니다.

완전한 구조 구문:

```
.....if({조건}){명령}{명령...}

        if({조건}) {
.....    {명령}
.....    [{명령}...]
        }
```

if 명령으로 모션 프로그램과 PLC 프로그램에서 조건 분기를 사용할 수 있습니다. 소괄호로 둘러싸인 조건의 평가가 True 인 경우 또는 식의 평가가 제로 이외의 값인 경우에 이 명령 직후의 프로그램 명령이 실행됩니다.

같은 프로그램행의 직후에 명령이 있는 경우에는 조건이 True 일 때 같은 행의 명령만 실행됩니다.

조건을 닫는 소괄호 직후에 여는 중괄호(**{**)가 있는 경우에는 같은 행이나 다음 프로그램행의 선두라도 이 중괄호는 조건이 True 인 경우에 실행하는 명령의 시작을 나타냅니다. 이 명령의 실행은 닫는 중괄호(**}**)가 있는 행까지 계속합니다.

조건의 평가가 False 인 경우 또는 식의 평가가 정확하게 0.0 값인 경우에 조건이 True 일 때 실행하는 명령은 건너뜁니다. 건너뀀 명령 직후에 **else** 명령이 있는 경우에는 대신에 **else** 뒤의 명령이 실행됩니다. **else** 명령이 없고 조건이 False 인 경우에는 아무것도 처리되지 않습니다.

if 조건은 다른 조건 구조체(**if**, **while**, **switch**) 내부에서 네스트할 수 있습니다. 또한 다른 조건 구조체를 **if** 조건 내부에서 네스트할 수 있습니다. 조건 구조체는 최대 32 레벨까지 네스트할 수 있습니다.

예

```
if (M10 == 0) P20++;           // Increment P20 if input M1 false

if (P500 > 0) {                // Start branch if P500 if greater than 0
    X50 tm200;                 // Execute X-axis move
    dwell 75;                  // Dwell for 75 msec
}                               // End of branch for P500 > 0
else P500 = 100;               // Otherwise set P500 to 100
```

inc

기능: 인크리멘탈 동작 모드 설정

구문: **inc**[({축 목록})]

여기에서

- **{축 목록}**은 쉼표로 나눈 축 명칭(1 문자 또는 2 문자) 및/또는 시작 축 명칭에서 종료 축 명칭 사이를 마침표 2 개로 나타내는 연속한 축 범위입니다. 사용되는 모든 축 명칭은 알파벳 순이며 먼저 1 문자의 이름을 지정하고 그 다음에 2 문자의 이름을 지정해야 합니다.

인수를 가지지 않는 **inc** 명령에 따라 프로그램에서 명령된 좌표계의 모든 축 동작 명령 내의 모든 축 이동 대상이 이전 명령 위치에서의 상대 위치로 처리됩니다. 이것은 인크리멘탈 모드이며 기본값인 앱솔루트 모드의 반대입니다.

인수를 가지는 **inc** 명령에 따라 프로그램을 실행하는 좌표계의 지정된 축이 인크리멘탈 모드가 됩니다. 그 외의 축은 변경할 수 없습니다. 이 명령 내에 공백은 사용할 수 없습니다.

inc 명령을 실행하면 32 비트 데이터 구조체 요소 **[Coord[x].IncAxes]**(인크리멘탈 모드 축)가 설정됩니다.

예

```
inc;  
inc(X,Y);  
inc(V);  
inc(XX,YY,ZZ,CC);  
inc(A..Z);  
inc(A,B,UU..ZZ);
```

inc({벡터 목록})

기능: 인크리멘탈 원 중심 벡터를 지정함

구문: **inc({벡터 목록})**

여기에서

- {벡터 목록}**은 쉼표로 구분된 벡터 컴포넌트명(I, J, K, II, JJ, KK) 세트 및/또는 시작 벡터 컴포넌트명에서 종료 컴포넌트명 사이를 2 개의 마침표로 나타낸 연속된 벡터 컴포넌트 범위입니다. 사용되는 모든 벡터 컴포넌트명은 알파벳 순으로 최초에 1 문자의 이름을 지정하고 그 다음에 2 문자의 이름을 지정해야 합니다.

inc({벡터 목록}) 명령에 따라 지정된 원 중심 벡터 컴포넌트가 인크리멘탈 모드가 됩니다. 인크리멘탈 모드에서는 중심 벡터 컴포넌트가 앱솔루트 모드처럼 프로그래밍 원점에서가 아니라 일치하는 축(I 는 X, J 는 Y, K 는 Z)의 동작 시작점에서 원 중심까지 부호가 있는 거리를 지정합니다. 전원 셋업/리셋 시에 모든 벡터 컴포넌트는 인크리멘탈 모드입니다.

inc({벡터 목록}) 명령을 실행하면 좌표계의 Status 데이터 구조체 요소 **[Coord[x].cdata]**의 일부 비트가 클리어됩니다.

목록을 가지지 않는 **inc** 명령에 따라 원 중심 벡터에 영향을 미치지 않으면서 좌표계 모든 축이 인크리멘탈 모드가 된다는 점에 주의하십시오.

예

```
inc(I,J,K);           // Put I/J/K vector set in inc mode  
inc(I..K);            // Put I/J/K vector set in inc mode  
inc(I..KK);           // Put both vector sets in inc mode
```

jog+

기능: 양의 무한 JOG 이송을 실행함

구문: **jog+[{목록}]**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수이며 또는 시작 상수에서 종료 상수까지의 사이를 2 개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0 부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1 을 뺀 값까지입니다.

jog+ 명령을 사용하면 주소 지정 또는 지정된 모터에 대해 양의 방향으로 무한 JOG 이송을 시작합니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(Plc[j].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(Coord[x].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={식}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터가 명령을 받아 동시에 동작을 시작합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램의 다음 명령이 실행되기 전에 모든 축에서 최초의 명령 가속도 프로파일을 완료해야 합니다. 최상위 프로그램이 PLC 프로그램인 경우에 JOG 이송 동작 시작 후에 바로 프로그램의 다음 명령이 실행됩니다.

최상위 프로그램이 PLC 프로그램인 경우에 명령 시에 모터가 오픈 루프 상태(유효 또는 출력 정지)일 가능성이 있습니다. 그 경우 Power PMAC 은 가속도 프로파일 시작 속도로 명령 시 모터의 현재 속도를 사용합니다.

JOG 이송 속도는 설정 데이터 구조체 요소 [Motor[x].JogSpeed]에 따라 결정됩니다. JOG 이송의 가속도 프로파일은 설정 데이터 구조체 요소 [Motor[x].JogTa] 및 [Motor[x].JogTs]에 따라 결정됩니다.

상당하는 JOG 이송 동작의 온라인 명령은 생략형 **j+**를 사용하며 모터 목록이 그 앞에 놓인다는 점(**#1,2j+** 등)에 주의하십시오.

예

```
jog+; // Acts on presently addressed motor
jog+1; // Acts on Motor 1, regardless of addressed
jog+1,2,3;
jog+1..3,5..7;
```

jog-

기능: 음의 무한 JOG 이송을 실행함

구문: **jog-[{목록}]**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수이며 또는 시작 상수에서 종료 상수까지의 사이를 2 개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0 부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1 을 뺀 값까지입니다.

jog- 명령을 사용하면 주소 지정 또는 지정된 모터에 대해 음의 방향으로 무한 JOG 이송을 시작합니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(Plc[j].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(Coord[x].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={식}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터가 명령을 받아 동시에 동작을 시작합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램의 다음 명령이 실행되기 전에 모든 축에서 최초의 명령 가속도 프로파일을 완료해야 합니다. 최상위 프로그램이 PLC 프로그램인 경우에 JOG 이송 동작 시작 후에 바로 프로그램의 다음 명령이 실행됩니다.

최상위 프로그램이 PLC 프로그램인 경우에 명령 시에 모터가 오픈 루프 상태(유효 또는 출력 정지)일 가능성이 있습니다. 그 경우 Power PMAC 은 가속도 프로파일 시작 속도로 명령 시 모터의 현재 속도를 사용합니다.

JOG 이송 속도는 설정 데이터 구조체 요소 [Motor[x].JogSpeed]에 따라 결정됩니다. JOG 이송의 가속도 프로파일은 설정 데이터 구조체 요소 [Motor[x].JogTa] 및 [Motor[x].JogTs]에 따라 결정됩니다.

상당하는 JOG 이송 동작의 온라인 명령은 생략형 **j-**를 사용하며 모터 목록이 그 앞에 놓인다는 점(**#1,2j-** 등)에 주의하십시오.

예

```
jog-; // Acts on presently addressed motor
jog-1; // Acts on Motor 1, regardless of addressed
jog-1,2,3;
jog-1..3,5..7;
```


jog/

기능: JOG 정지 동작을 실행함

구문: **jog/****[{목록}]**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수이며 또는 시작 상수에서 종료 상수까지의 사이를 2 개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0 부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1 을 뺀 값까지입니다.

jog/ 명령을 사용하면 주소 지정 또는 지정된 모터가 클로즈드 루프의 제로 속도 상태가 되고 필요에 따라 JOG 이송 타입 동작을 정지합니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(Plc[i].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(Coord[x].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={x}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터가 명령을 받아 동시에 동작을 시작합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램의 다음 명령이 실행되기 전에 모든 축에서 명령 감속도 프로파일을 완료해야 합니다. 최상위 프로그램이 PLC 프로그램인 경우에 명령에 따른 프로파일 시작 후에 바로 프로그램의 다음 명령이 실행됩니다.

최상위 프로그램이 PLC 프로그램인 경우에 명령 시에 모터가 오픈 루프 상태(유효 또는 출력 정지)일 가능성이 있습니다. 그 경우 Power PMAC 은 감속 프로파일 시작 속도로서 명령 시 모터의 현재 속도를 사용합니다. 현재 속도가 정확하게 제로인 경우에 프로파일은 작성되지 않습니다.

JOG 이송의 가속도 프로파일은 설정 데이터 구조체 요소 [Motor[x].JogTa] 및 [Motor[x].JogTs]에 따라 결정됩니다.

상당하는 JOG 이송 동작의 온라인 명령은 생략형 **j/**를 사용하며 모터 목록이 그 앞에 놓인다는 점(**#1,2j/** 등)에 주의하십시오.

예

```
jog/; // Acts on presently addressed motor
jog/1; // Acts on Motor 1, regardless of addressed
jog/1,2,3;
jog/1..3,5..7;
```

jog={데이터}

기능: 지정된 위치에 JOG 이송을 실행함

구문: **jog[{목록}]= {데이터}**

jog[{목록}]=*

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수이며 또는 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0 부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1 을 뺀 값까지입니다.
- **{데이터}**는 목적지 위치를 모터 단위로 지정하는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

jog={데이터} 명령에 따라 주소 지정 또는 목록화된 모터가 **{데이터}**로 지정된 모터 위치까지 한정 JOG 이송 동작을 시작합니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

대신에 **jog=*** 구문을 사용하는 경우에 요소 **[Motor[x].ProgJogPos]** 값에 따라 목적지 위치가 지정됩니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(**Plc[j].Ldata.Motor**)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(**Coord[x].Ldata.Motor**)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={식}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터가 명령을 받아 동시에 동작을 시작합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램의 다음 명령이 실행되기 전에 모든 축에서 최초의 명령 가속도 프로파일을 완료해야 합니다. 최상위 프로그램이 PLC 프로그램인 경우에 JOG 이송 동작 시작 후에 바로 프로그램의 다음 명령이 실행됩니다.

최상위 프로그램이 PLC 프로그램인 경우에 명령 시에 모터가 오픈 루프 상태(유효 또는 출력 정지)일 가능성이 있습니다. 그 경우 Power PMAC 은 가속도 프로파일 시작 속도로 명령 시 모터의 현재 속도를 사용합니다.

JOG 이송 속도는 설정 데이터 구조체 요소 **[Motor[x].JogSpeed]**에 따라 결정됩니다. JOG 이송의 가속도 프로파일은 설정 데이터 구조체 요소 **[Motor[x].JogTa]** 및 **[Motor[x].JogTs]**에 따라 결정됩니다.

상당하는 JOG 이송 동작의 온라인 명령은 생략형 **j={상수}**를 사용하며 모터 목록이 그 앞에 놓인다는 점(**#1,2j=1000** 등)에 주의하십시오. JOG 이송의 온라인 명령은 상수를 사용하여 위치 또는 거리를 지정해야 합니다. 변수 및 식은 사용할 수 없습니다.

예

```
jog=5000;           // Acts on presently addressed motor
jog1=(Q1);          // Acts on Motor 1, regardless of addressed
jog1,2,3=-2468;
jog1..3,5..7=0;
jog25..27=*
```

jog:{데이터}

기능: 명령 위치와 관련되는 지정된 거리로 JOG 이송을 실행함

구문: jog[{목록}]:{데이터}

jog[{목록}]:*

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수이며 또는 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1을 뺀 값까지입니다.
- **{데이터}**는 부호가 있는 목적지 거리를 모터 단위로 지정하는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

jog :{데이터} 명령에 따라 주소 지정 또는 지정된 모터가 현재 명령 위치에서 **{데이터}**로 지정된 부호가 있는 거리의 한정 JOG 이송 동작을 시작합니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

대신에 **jog:*** 구문을 사용하는 경우에 요소 [Motor[x].ProgJogPos] 값에 따라 목적지 거리가 지정됩니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(Plc[j].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(Coord[x].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={식}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터가 명령을 받아 동시에 동작을 시작합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램의 다음 명령이 실행되기 전에 모든 축에서 최초의 명령 가속도 프로파일을 완료해야 합니다. 최상위 프로그램이 PLC 프로그램인 경우에 JOG 이송 동작 시작 후에 바로 프로그램의 다음 명령이 실행됩니다.

최상위 프로그램이 PLC 프로그램인 경우에 명령 시에 모터가 오픈 루프 상태(유효 또는 출력 정지)일 가능성이 있습니다. 그 경우 Power PMAC은 가속도 프로파일 시작 위치와 시작 속도로서 명령 시 모터의 현재 속도를 사용합니다.

JOG 이송 속도는 설정 데이터 구조체 요소 [Motor[x].JogSpeed]에 따라 결정됩니다. JOG 이송의 가속도 프로파일은 설정 데이터 구조체 요소 [Motor[x].JogTa] 및 [Motor[x].JogTs]에 따라 결정됩니다.

상당하는 JOG 이송 동작의 온라인 명령은 생략형 **j:{상수}**를 사용하며 모터 목록이 그 앞에 놓인다는 점(#1,2j:-1000 등)에 주의하십시오. JOG 이송의 온라인 명령은 상수를 사용하여 위치 또는 거리를 지정해야 합니다. 변수 및 식은 사용할 수 없습니다.

예

```
jog:5000;           // Acts on presently addressed motor
jog1:(Q1);          // Acts on Motor 1, regardless of addressed
jog1,2,3:-2468;
jog1..3,5..7:0;
jog25..27:*
```

jog^{데이터}

기능: 현재 위치와 관련되는 지정된 거리로 JOG 이송을 실행함

구문: **jog[{목록}]^{데이터}**

jog[{목록}]^*

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수이며 또는 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1을 뺀 값까지입니다.
- **{데이터}**는 부호가 있는 목적지 거리를 모터 단위로 지정하는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

jog^{데이터} 명령에 따라 주소 지정 또는 지정된 모터가 현재 위치에서 **{데이터}**로 지정된 부호가 있는 거리의 한정 JOG 이송 동작을 시작합니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

대신에 **jog^*** 구문을 사용하는 경우에 요소 [Motor[x].ProgJogPos] 값에 따라 목적지 거리가 지정됩니다.

명령 위치가 아니라 현재 실제 위치에 대해 거리가 지정되기 때문에 동작은 명령 시점의 위치 편차에 좌우됩니다. 따라서 동작을 예측할 수 없습니다. 많은 경우 현재 명령 위치로부터의 거리를 지정하는 유사한 명령인 **jog:{데이터}**를 선택합니다. 단, 기존 위치 편차를 메꾸는 경우에는 **jog^0**가 유효합니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(Plc[i].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(Coord[x].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={식}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터가 명령을 받아 동시에 동작을 시작합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램의 다음 명령이 실행되기 전에 모든 축에서 최초의 명령 가속도 프로파일을 완료해야 합니다. 최상위 프로그램이 PLC 프로그램인 경우에 JOG 이송 동작 시작 후에 바로 프로그램의 다음 명령이 실행됩니다.

최상위 프로그램이 PLC 프로그램인 경우에 명령 시에 모터가 오픈 루프 상태(유효 또는 출력 정지)일 가능성이 있습니다. 그 경우 Power PMAC 은 가속도 프로파일 시작 속도로 명령 시 모터의 현재 속도를 사용합니다.

JOG 이송 속도는 설정 데이터 구조체 요소 [Motor[x].JogSpeed]에 따라 결정됩니다. JOG 이송의 가속도 프로파일은 설정 데이터 구조체 요소 [Motor[x].JogTa] 및 [Motor[x].JogTs]에 따라 결정됩니다.

상당하는 JOG 이송 동작의 온라인 명령은 생략형 **j{상수}**를 사용하며 모터 목록이 그 앞에 놓인다는 점(**#1,2j^1000** 등)에 주의하십시오. JOG 이송의 온라인 명령은 상수를 사용하여 위치 또는 거리를 지정해야 합니다. 변수 및 식은 사용할 수 없습니다.

예

```
jog^5000;           // Acts on presently addressed motor
jog1^(Q1);          // Acts on Motor 1, regardless of addressed
jog1,2,3^2468;
jog1..3,5..7^0;
jog25..27=*
```

jogret

기능: 사전 JOG 이송 위치로 JOG 이송을 실행함

구문: **jogret[{목록}]**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수이며 또는 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1을 뺀 값까지입니다.

jogret 명령을 사용하면 주소 지정 또는 지정된 모터에 대해 이전의 사전 JOG 위치까지 한정 JOG 이송을 시작합니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(Plc[i].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(Coord[x].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={식}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터가 명령을 받아 동시에 동작을 시작합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램의 다음 명령이 실행되기 전에 모든 축에서 최초의 명령 가속도 프로파일을 완료해야 합니다. 최상위 프로그램이 PLC 프로그램인 경우에 JOG 이송 동작 시작 후에 바로 프로그램의 다음 명령이 실행됩니다.

최상위 프로그램이 PLC 프로그램인 경우에 명령 시에 모터가 오픈 루프 상태(유효 또는 출력 정지)일 가능성이 있습니다. 그 경우 Power PMAC 은 가속도 프로파일 시작 속도로 명령 시 모터의 현재 속도를 사용합니다.

JOG 이송 속도는 설정 데이터 구조체 요소 [Motor[x].JogSpeed]에 따라 결정됩니다. JOG 이송의 가속도 프로파일은 설정 데이터 구조체 요소 [Motor[x].JogTa] 및 [Motor[x].JogTs]에 따라 결정됩니다.

상당하는 JOG 이송 동작의 온라인 명령은 생략형 **j=**을 사용하며 모터 목록이 그 앞에 놓인다는 점(**#1,2j=** 등)에 주의하십시오.

예

```
jogret;                // Acts on presently addressed motor
jogret1;               // Acts on Motor 1, regardless of addressed
jogret1, 2, 3;
jogret1..3,5..7;
```

jogret={데이터}

기능: 지정된 위치로 JOG 이송하고 그 위치를 사전에 JOG 위치로 함

구문: **jogret[{목록}]{데이터}**

jogret[{목록}]=*

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수이며 또는 시작 상수에서 종료 상수까지의 사이를 2 개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0 부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1 을 뺀 값까지입니다.
- **{데이터}**는 목적지 위치를 모터 단위로 지정하는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

jogret={데이터} 명령에 따라 주소 지정 또는 지정된 모터에 대해 **{데이터}**로 지정된 모터 위치까지 한정 JOG 이송 동작을 시작시키고 그 위치를 후속 **jogret** 명령의 사전 JOG 위치로 합니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

대신에 **jogret=*** 구문을 사용하는 경우에 요소 [Motor[x].ProgJogPos] 값에 따라 목적지 위치가 지정됩니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(Plc[i].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(Coord[x].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={식}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터가 명령을 받아 동시에 동작을 시작합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램의 다음 명령이 실행되기 전에 모든 축에서 최초의 명령 가속도 프로파일을 완료해야 합니다. 최상위 프로그램이 PLC 프로그램인 경우에 JOG 이송 동작 시작 후에 바로 프로그램의 다음 명령이 실행됩니다.

최상위 프로그램이 PLC 프로그램인 경우에 명령 시에 모터가 오픈 루프 상태(유효 또는 출력 정지)일 가능성이 있습니다. 그 경우 Power PMAC 은 가속도 프로파일 시작 속도로 명령 시 모터의 현재 속도를 사용합니다.

JOG 이송 속도는 설정 데이터 구조체 요소 [Motor[x].JogSpeed]에 따라 결정됩니다. JOG 이송의 가속도 프로파일은 설정 데이터 구조체 요소 [Motor[x].JogTa] 및 [Motor[x].JogTs]에 따라 결정됩니다.

상당하는 JOG 이송 동작의 온라인 명령은 생략형 **j={상수}**를 사용하며 모터 목록이 그 앞에 놓인다는 점(**#1,2j=1000** 등)에 주의하십시오. JOG 이송의 온라인 명령은 상수를 사용하여 위치 또는 거리를 지정해야 합니다. 변수 및 식은 사용할 수 없습니다.

예

```
jogret=5000;           // Acts on presently addressed motor
jogret1=(Q1);          // Acts on Motor 1, regardless of addressed
jogret1,2,3=-2468;
jogret1..3,5..7=0;
jogret25..27=*
```

{JOG 명령}^{데이터}

기능: 트리거까지의 JOG 이송 동작을 실행함

범위: 모터 고유

구문:

```
jog[{목록}]={데이터}^{데이터}
jog[{목록}]:{데이터}^{데이터}
jog[{목록}]^{데이터}^{데이터}
jog[{목록}]=*^{데이터}
jog[{목록}]:*^{데이터}
jog[{목록}]^*^{데이터}
```


여기에서

- **{목록}**은 쉼표로 구분된 정수 상수이며 또는 시작 상수에서 종료 상수까지의 사이를 2 개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0 부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1 을 뺀 값까지입니다.
- 명령의 마지막인 **{데이터}**는 트리거 위치에서 트리거 후의 동작으로 명령된 목적지까지의 (부호가 있는) 거리를 모터 단위로 지정하는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

트리거까지의 JOG 이송 클래스 명령을 사용하면 주소 지정 또는 지정된 모터에 대해 지정된 타입의 한정 JOG 이송을 시작합니다. 이 동작 중에 사전 정의된 트리거 조건이 발생하면 Power PMAC 은 자동으로 동작 궤적에 인터럽트하여 남은 부분을 트리거 위치로부터의 거리가 명령 마지막의 **{데이터}**로 결정되는 목적지까지의 JOG 이송과 치환합니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

대신에 **jog={데이터}**, **jog:*^{데이터}** 또는 **jog^{*^{데이터}}** 구문을 사용하는 경우에 요소 [Motor[x].ProgJogPos] 값에 따라 트리거 앞의 목적지 위치 또는 거리가 지정됩니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(Plc[j].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(Coord[x].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={식}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터가 명령을 받아 동시에 동작을 시작합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램의 다음 명령이 실행되기 전에 모든 축에서 최초의 명령 가속도 프로파일을 완료해야 합니다. 최상위 프로그램이 PLC 프로그램인 경우에 JOG 이송 동작 시작 후에 바로 프로그램의 다음 명령이 실행됩니다.

최상위 프로그램이 PLC 프로그램인 경우에 명령 시에 모터가 오픈 루프 상태(유효 또는 출력 정지)일 가능성이 있습니다. 그 경우 Power PMAC 은 가속도 프로파일 시작 속도로 명령 시 모터의 현재 속도를 사용합니다.

JOG 이송 속도는 설정 데이터 구조체 요소 [Motor[x].JogSpeed]에 따라 결정됩니다. JOG 이송의 가속도 프로파일은 설정 데이터 구조체 요소 [Motor[x].JogTa] 및 [Motor[x].JogTs]에 따라 결정됩니다.

트리거 조건은 모터의 설정 데이터 구조체 요소 [Motor[x].CaptureMode]로 설정합니다.

상당하는 JOG 이송 동작의 온라인 명령은 **jog...** 대신에 생략형 **j...**을 사용하며 모터 목록이 그 앞에 놓인다는 점(**#1,2j=10000^50** 등)에 주의하십시오. JOG 이송의 온라인 명령은 상수를 사용하여 위치 또는 거리를 지정해야 합니다. 변수 및 식은 사용할 수 없습니다.

예

```
jog=10000^-50;           // Acts on presently addressed motor
jog1:-50000^(P10);        // Acts on Motor 1, regardless of addressed
jog1..3=*^0;              // Motors 1, 2, & 3 jog to ProgJogPos, but return to trigger pos
```

kill

기능: 모터 출력 정지 명령을 전송함

구문: **kill**{**목록**}

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수이며 또는 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0부터 특정 버전의 Power PMAC 펌웨어가 지원하는 모터 수에서 1을 뺀 값까지입니다.

kill 명령은 주소 지정 또는 지정된 모터를 출력 정지합니다. 명령 뒤에 모터의 목록이 없는 경우에 프로그램에서 주소 지정된 한 대의 모터에 대해 작용합니다.

최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소(Plc[j].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 최상위 프로그램이 모션 프로그램인 경우에 프로그램이 실행되는 좌표계의 데이터 구조체 내의 요소(Coord[x].Ldata.Motor)에 따라 주소 지정되는 모터가 결정됩니다. 모션 프로그램은 프로그램을 실행하는 좌표계의 축에 정의된 모터에만 명령할 수 있습니다. 어느 프로그램에서도 명령 **Ldata.Motor={식}**으로 모터를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

명령이 모터 목록을 사용하여 모터 세트를 지정하는 경우에 목록 내의 모든 모터에 동시에 명령합니다. 목록 내 모터를 지정해도 프로그램에 따라 모달에 주소 지정된 모터에는 영향을 미치지 않습니다.

모터를 출력 정지하면 서보 루프가 오픈되고 서보 출력값이 강제로 제로가 됩니다(단, 출력 바이어스 항은 적용됩니다). 다음으로 앰프 **Enable** 신호가 강제로 무효 상태가 됩니다. Power PMAC 은 앰프 폴트 시나 편차 이상 시에는 자동으로 모터를 출력 정지합니다.

프로그램 **kill** 모터 명령은 모션 프로그램을 실행 중인 좌표계 내의 모터에는 영향을 미치지 않습니다. 프로그램 실행을 먼저 정지해야 합니다. 단, 프로그램 **disable** 좌표계 명령은 동일하게 동작하지만 모션 프로그램을 실행하는 좌표계 내의 모터도 포함하여 모든 모터에 작용합니다.

즉시 출력 정지 명령은 모터의 긴급 정지용입니다. 자동 브레이크 제어를 이용하여 모터 **Disable** 을 계획하는 경우에는 브레이크가 완전히 연동할 때까지 시간이 필요하기 때문에 동일하게 지연된 **dkill** 명령으로 실행해야 합니다. 자동 브레이크 제어가 장착되어 있지 않은 모터는 어느 명령을 사용해도 문제 없습니다.

일반적으로 이 명령은 PLC 프로그램에서만 사용하십시오. 모션 프로그램 표준 동작에서는 실행 중인 좌표계의 한 축에 할당된 모든 모터가 유효하고 클로즈드 루프이어야 합니다.

상당하는 JOG 이송 동작의 온라인 명령은 생략형 **k** 를 사용하며 모터 목록이 그 앞에 놓인다는 점(#1,2k 등)에 주의하십시오.

예

```
kill; // Acts on presently addressed motor
kill1; // Acts on Motor 1, regardless of addressed
kill1,2,3;
kill1..3,5..7;
```

L{데이터}{대입 연산자}{식}

기능: 지정된 'L' 변수에 값을 할당함

구문: L{데이터}[,{식},{식}]
{대입 연산자}{식}

여기에서

- {데이터}는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 값을 할당하는 첫 번째 변수 번호를 나타내는 소괄호 내의 수식입니다.
- 첫 번째 옵션 {식}은 값을 할당하는 변수의 수를 지정합니다. 명확하게 지정되지 않은 경우에는 1 이 사용되고 값은 하나의 변수에만 할당됩니다.
- 두 번째 옵션 {식}은 값을 할당하는 변수의 숫자 간격을 지정합니다. 명확하게 지정되지 않은 경우에는 1 이 사용되고 값은 연속된 번호의 변수에 할당됩니다.
- {대입 연산자}는 식에서 값을 변수에 할당하는 방법을 제어하는 숫자 연산자입니다.
- 마지막 {식}은 상수, 변수, 논리 및 산술 연산자, 함수를 조합한 수식이며 도출된 식의 값이 할당에 사용됩니다.

L{데이터}{대입 연산자}{식} 명령에 따라 지정된 'L' 변수에 식에서 구한 값이 들어갑니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수 번호를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~8,191). 유효한 변수 번호를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 실행하지 않는 것으로 처리됩니다.

'L' 변수는 프로그램 내부에서 사용되는 로컬 스택 변수이며 고레벨 루틴에 값을 반환합니다. 각 좌표계에는 최상위 모션 프로그램 및 서브 프로그램에서 사용되는 독자적인 'L' 변수 세트가 있습니다. 또한 각 PLC 프로그램에는 모션 프로그램에서 사용하는 세트가 있으며 각 통신 스레드 자체에도 세트가 있습니다. 'L' 변수의 모든 세트는 각각 독립적입니다. 지정된 'L' 변수에 기본적인 문자/번호 형식(L10 등)으로 이름을 직접 할당합니다. 또한 IDE 소프트웨어 사용 시에는 선언된 local 변수명('L' 변수에 자동으로 할당됨) 또는 특정 'L' 변수에 부여되는 이름에 따라 #define 텍스트의

대체로 변수에 이름을 직접 할당할 수 있습니다. 후자의 경우에 기본적인 문자/번호 형식의 변수명으로 대체하는 것은 다운로드 시 자동으로 이루어집니다.

동일한 간격으로 나열되는 여러 변수에 같은 값을 할당할 수 있습니다. 초기 변수명 뒤에 두 번째 값이 사용되는 경우에 값이 할당되는 변수의 수를 지정합니다(두 번째 값이 사용되지 않는 경우에 기본값인 1 이 되고 초기 변수가 지정됩니다). 세 번째 값이 사용되는 경우에 변수의 간격을 수치로 지정합니다(세 번째 값이 사용되지 않는 경우에 기본값인 1 이 되며 연속된 번호의 변수가 사용됩니다).

보통 연산자 뒤에 있는 도출된 식은 = 대입 연산자를 사용하고 변수에 직접 대입됩니다. 단, 이 값은 대입 연산자 +=, -=, *=, /=, %=, &=, |=, ^=, >>=, <<=를 사용하여 산술적 또는 논리적으로 변수의 기존 값과 조합할 수도 있습니다. 또한 증가 및 감소 연산자 ++ 및 --는 뒤에 식이 없어도 사용할 수 있습니다.

'L' 변수는 로컬 변수이기 때문에 글로벌 변수에서 사용 가능한 지연 동기 대입 연산자를 사용할 수 없습니다. 실제로 할당 시에 컨텍스트가 소실될 가능성이 있기 때문입니다.

예

```
L1=17.5;           // Set variable L1 to 17.5
L(P1)+=3;          // Add 3 to L-variable numbered by P1
L(P1+P17)=5*sqrt(P100); // Set L-var numbered by (P1+P17) to expression value
L100,16=0;         // Set L100 - L115 to 0
L20,5,10++;        // Increment L20, L30, L40, L50, L60
```

lh\

기능: 룩 어헤드 급정지를 명령함

구문: lh\ [{목록}]

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.

Power PMAC 은 lh\ 명령으로 대상 또는 목록에 있는 좌표계(좌표계의 모든 모터가 가속도 제한을 위반하지 않는)의 룩 어헤드 버퍼 내에서 급정지를 계산하고 실행합니다. 모션은 프로그램된 궤적에 따라 제어 정지를 계속하지만 프로그램 포인트에서 반드시 정지하는 것은 아닙니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 [Plc[i].Ldata.Coord]에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(Coord[x].Ldata.Coord)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

lh 급정지 명령은 일반적으로 룩 어헤드에서 모션을 대화적으로 정지시키기 위한 최적의 명령입니다. 이 기능은 종래의 피드 홀드 명령 기능과 비슷하지만 Power PMAC 의 표준적인 **hold** 피드 홀드 명령과는 달리 [Motor[x].InvAmax]에 따라 설정되는 각 좌표계의 모든 모터 가속도 제한이 준수됩니다.

일단 정지하면 다음의 옵션이 가능합니다.

- JOG 이송 명령 중 하나로 축을 JOG 이송합니다. 온라인 JOG 명령을 사용하여 정지에서 좌표계 중 하나의 모터를 JOG 어웨이할 수 있습니다. 단, 프로그램된 궤적 실행을 재개할 수 있도록 미리 **j=** 명령으로 모든 모터를 원래 정지점으로 되돌려놔야 합니다.
- **<** 명령으로 궤적에 따라 역행을 시작합니다.
- **>**, **run** 또는 **step** 명령으로 정방향 실행을 재개합니다.
- **abort** 명령으로 프로그램 실행을 종료합니다.

룩 어헤드 버퍼 내에서 동작을 실행하지 않는 좌표계에 대해 **lh** 명령이 지정되면 Power PMAC 은 대신에 hold(피드 홀드) 명령을 실행합니다.

상당하는 온라인 명령은 생략형 ****를 사용하며 그 앞에 좌표계 목록이 놓인다는 점(&1,2\ 등)에 주의하십시오.

예

```
lh\; // Acts on presently addressed motor
lh\1; // Acts on C.S. 1, regardless of addressed
```

lh<

기능: 룩 어헤드 중 역행을 명령함

구문: **lh< {목록}**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.

Power PMAC 은 **lh<** 명령으로 대상 또는 목록에 있는 좌표계에 대해 룩 어헤드 버퍼 내에서 역행을 시작합니다. 프로그램이 정방향으로 실행된 경우에 처음에 (**lh** 명령과 등가인)급정지가 실행됩니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 `[Plc[i].Ldata.Coord]`에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(`Coord[x].Ldata.Coord`)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

전진 운동(존재하는 경우)의 감속 및 역방향 가속에서는 `[Motor[x].InvAmax]`의 가속 제한을 준수합니다.

실행은 룩 어헤드 버퍼에 버퍼된 포인트에 따라 역방향으로 진행하고 순방향과 동일하게 감속 및 가속 제한을 준수합니다. 이 실행은 다음 중 하나가 발생할 때까지 계속 실행합니다.

- 가장 오래된 저장점이 룩 어헤드 버퍼에 아직 남아있고 역행이 룩 어헤드 버퍼의 선두에 도달하여 정지를 향해 감속하면서 가속 제어를 준수하여 이 시점에서 제어 정지한 경우
- lh**(급정지) 명령이 지정되어 룩 어헤드 버퍼에서 Power PMAC 이 가장 급속으로 정지시킬 수 있었던 경우
- lh>**(재개-정방향), **run** 또는 **step** 명령이 지정되어 Power PMAC 이 프로그램 및 필요에 따라 룩 어헤드 버퍼의 정방향 실행을 재개한 경우
- 에러 상태가 발생한 경우 또는 회복 불가능한 정지 명령이 지정된 경우

급정지점에서 어느 한 모터가 JOG 이송되고 **j=** 명령으로 복귀되지 않는 경우에 Power PMAC 은 **lh<**(백업) 명령을 거부하고 에러를 통지합니다.

좌표계가 룩 어헤드 시퀀스 중이 아닌 경우에 Power PMAC 은 이 명령을 **hold**(피드 홀드) 명령으로 실행합니다.

상당하는 온라인 명령은 생략형 **<**를 사용하며 그 앞에 좌표계 목록이 놓인다는 점(**&1,2<** 등)에 주의하십시오.

예

<code>lh<;</code>	<code>// Acts on presently addressed motor</code>
<code>lh<1;</code>	<code>// Acts on C.S. 1, regardless of addressed</code>

lh>

기능: 순방향 룩 어헤드 실행을 재개함

구문: **lh> [{목록}]**

여기에서

- {목록}**은 쉼표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.

Power PMAC 은 **lh>** 명령으로 대상의 좌표계에 대해 룩 어헤드 버퍼 내에서 정방향 실행을 재개합니다. 이것은 보통 **lh**(급정지) 명령 또는 **lh<**(백업) 명령 뒤에 보통의 처리를 재개하기 위해 사용됩니다. 프로그램이 역방향으로 실행된 경우에 처음에 (**lh** 명령과 등가인)급정지가 실행됩니다. 좌표계 목록이 없는 경우에 현재 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 **[Plc[i].Ldata.Coord]**에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(**Coord[x].Ldata.Coord**)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

앞의 정방향 실행이 (**run** 명령으로 시작하는)계속 모드인 경우에 **lh>** 명령은 계속 모드로 실행을 재개합니다. 앞의 정방향 실행이 (**step** 명령으로 시작하는)싱글 스텝 모드인 경우에 **lh>** 명령은 싱글 스텝 모드로 실행을 재개합니다. 정방향 실행을 재개하기 위해 **run** 명령과 **step** 명령을 사용할 수 있지만 이러한 명령에 따라 계속/싱글 스텝 모드가 변경될 가능성이 있습니다.

역행(존재하는 경우)에서 감속 및 정방향에서 가속 시에 **[Motor[x].InvAmax]** 가속 제한을 준수합니다.

급정지점에서 어느 한 모터가 JOG 이송되고 **jog=** 명령으로 복귀되지 않는 경우에 Power PMAC 은 **lh>**(재개) 명령을 거부하고 에러를 통지합니다.

좌표계가 룩 어헤드 시퀀스 중이 아닌 경우에 Power PMAC 은 이 명령을 **run** 명령으로 실행합니다.

상당하는 온라인 명령은 생략형 **>**를 사용하며 그 앞에 좌표계 목록이 놓인다는 점(**&1,2>** 등)에 주의하십시오.

예/

lh>;	// Acts on presently addressed motor
lh>1;	// Acts on C.S. 1, regardless of addressed

lhpurge

기능: 룩 어헤드 버퍼 콘텐츠를 비움

구문: **lhpurge**

모션 프로그램 또는 그 서브 프로그램에서 **lhpurge** 명령을 실행하면 프로그램을 실행하는 좌표계의 룩 어헤드 버퍼 콘텐츠를 비웁니다. PLC 프로그램 또는 그 서브 프로그램에서 실행해도 처리가 수행되지 않습니다. 이 명령은 좌표계 모터의 정의를 위치 결정축(적극적으로 룩 어헤드 버퍼를 사용하는)과 주축(적극적으로 룩 어헤드 버퍼를 사용하지 않는) 사이에서 전환되기 전에 실행하십시오. **lhpurge** 명령을 사용하면 축 정의 변경 후에 룩 어헤드 버퍼에서 발생할 가능성이 있는 부적합을 회피할 수 있습니다.

lhpurge 명령 실행 후에 동작이 버퍼에서 삭제되기 때문에 이미 실행된 동작을 역행할 수 없습니다. **lhpurge** 명령을 실행하면 그 전후로 **dwel** 명령이 사용되지 않는 경우라도 명령된 모션이 강제로 일시 정지됩니다. 실행 후에는 좌표계 [Coord[x].LookAheadActive] Status 비트가 클리어됩니다.

예

모터는 보통 **cmd**"" 구조체를 사용하여 모션 프로그램 또는 (대부분의 경우 G 코드 서브루틴인)서브 프로그램에서 위치 결정축과 주축과의 사이에서 변경됩니다. 이 변경 방법은 다음과 같습니다.

```
dwel 0; // Stop blending and lookahead
lhpurge; // Purge lookahead buffer
Ldata.CmdStatus = 1; // Will change when command executed
cmd"&1#4->S0"; // Define Motor 4 as spindle in CS1
sendallcmds; // Wait for command buffer to empty
do dwel 0; // Loop quickly while waiting
while (Ldata.CmdStatus == 1); // Until command fully executed
// Could check for error here (if < 0)
```

회전 위치 결정축으로 되돌리기 위한 동일한 순서는 다음과 같습니다.

```
dwel 0; // Stop blending and lookahead
lhpurge; // Purge lookahead buffer
Ldata.CmdStatus = 1; // Will change when command executed
cmd"&1#4->100C"; // Define Motor 4 as C-axis in CS1
sendallcmds; // Wait for command buffer to empty
do dwel 0; // Loop quickly while waiting
while (Ldata.CmdStatus == 1); // Until command fully executed
// Could check for error here (if < 0)
```

linear

기능: 직선 보간 동작 모드를 설정함

구문: **linear**

linear 명령은 모션 프로그램을 직선 보간 동작 모드(전원 셋업/리셋 시의 기본값 모드)로 합니다. 모션 프로그램의 후속 동작 명령은 이 모드의 규칙에 따라 처리됩니다.

linear 명령을 실행하면 그 외의 동작 모드(**circle**, **pvt**, **rapid**, **spline**)가 종료됩니다. 그 외의 동작 모드를 명령하면 직선 보간 모드가 종료됩니다.

M{데이터}

기능: 머신 출력 코드(M 코드)

구문: **M{데이터}**

여기에서

- **{데이터}**는 코드 번호 및 그 코드를 탑재하는 서브루틴의 행 점프 레이블을 나타내는 0.000~999.999 의 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

Power PMAC 은 M 코드를 **call{Coord[x].Mprog}.{데이터}*1000** 명령(**{데이터}**는 M 코드의 번호)으로 해석합니다. 이 명령에 따라 코드 번호를 1,000 배 한 번호의 행 점프 레이블 위치에서 [Coord[x].Mprog]의 값에 따라 번호가 지정된 서브 프로그램으로 점프합니다. 예를 들어 [Coord[x].Mprog]의 기본값이 1001 인 경우에 **subprog 1001**의 행 점프 레이블 **N3000**:에서 **M03** 이 호출됩니다. 프로그램 실행은 다음 **return** 명령에서 호출자 프로그램으로 복귀합니다.

이 구조체에 따라 커스터마이징 가능한 M 코드 루틴을 CAD/CAM 프로그램의 RS-274 대응 코드 및 CNC 스타일 어플리케이션에 탑재할 수 있습니다. 호출자 프로그램 M 코드의 뒤에 1 세트 이상의 **{문자}{데이터}**를 두면 해당 서브루틴에 인수를 전달할 수 있습니다. 각 문자의 뒤의 값은 **read** 명령을 사용하여 서브루틴으로 취득할 수 있습니다. 모든 좌표계의 [Coord[x].Mprog] 기본값은 1001 입니다. 따라서 모든 좌표계는 기본값으로 같은 M 코드 세트를 공유합니다. 단, 다른 좌표계에 다른 값을 할당하면 다른 코드를 탑재할 수 있습니다.

일반적으로 M 코드는 RS-274 프로그램에서 머신 출력 코드로 사용되고 주축 동작, 냉각수, 컨베이어 등의 머신 기능을 제어합니다. M 코드의 코어 세트는 0~99 의 정수를 사용하지만 이 스킴에서는 999.999 까지의 소수 코드 번호를 사용할 수 있고 코어 번호에 수식을 사용할 수 있습니다.

모든 종류의 서브루틴 및 서브 프로그램 호출(**call**, **callsub**, **gosub**, **G**, **M**, **T**, **D**)은 255 레벨(최상위를 포함하여 256)까지 네스트할 수 있습니다. 무한 루프는 사용하지 마십시오.

M 코드 명령에 따라 지정된 점프 행 레이블이 프로그램에 존재하지 않는 경우에 지정된 M 코드의 서브 프로그램 선두로 점프합니다. 에러는 발생하지 않습니다. 이를 이용하여 사용자 어플리케이션이 존재하지 않는 M 코드를 처리하는 방법을 결정할 수 있습니다.

예/

M03	// Call to label N3000 : of M-code subprogram
M28.3	// Call to label N28300 : of M-code subprogram
M197	// Call to label N197000 : of M-code subprogram

M{데이터}{대입 연산자}{식}

기능: 지정된 'M' 변수에 값을 할당함

구문: **M{데이터}[{식}][{식}]**
{대입 연산자}{식}

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 값을 할당하는 첫 번째 변수 번호를 나타내는 소괄호 내의 수식입니다.
- 첫 번째 옵션 **{식}**은 값을 할당하는 변수의 수를 지정합니다. 명확하게 지정되지 않은 경우에는 1 이 사용되고 값은 하나의 변수에만 할당됩니다.
- 두 번째 옵션 **{식}**은 값을 할당하는 변수의 숫자 간격을 지정합니다. 명확하게 지정되지 않은 경우에는 1 이 사용되고 값은 연속된 변수에 할당됩니다.
- **{대입 연산자}**는 식에서 값을 변수에 할당하는 방법을 제어하는 숫자 연산자입니다.
- 마지막 **{식}**은 상수, 변수, 논리 및 산술 연산자, 함수를 조합한 수식이며 도출된 식의 값이 할당에 사용됩니다.

M{데이터}{대입 연산자}{식} 명령에 따라 지정된 'M' 변수에 식에서 구한 값이 들어갑니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수 번호를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~16,383). 유효한 변수 번호를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 실행하지 않는 것으로 처리됩니다.

'M' 변수는 Power PMAC 하드웨어 및 필요한 소프트웨어의 레지스터에 액세스할 때 사용할 수 있는 글로벌 포인터 변수입니다. 사용 전에 레지스터에 정의되어야 합니다.

동일한 간격으로 나열되는 여러 변수에 같은 값을 할당할 수 있습니다. 초기 변수명 뒤에 두 번째 값이 사용되는 경우에 값이 할당되는 변수의 수를 지정합니다(두 번째 값이 사용되지 않는 경우에 기본값인 1 이 되고 초기 변수가 지정됩니다). 세 번째 값이 사용되는 경우에 변수의 간격을 수치로 지정합니다(세 번째 값이 사용되지 않는 경우에 기본값인 1 이 되며 연속된 번호의 변수가 사용됩니다).

보통 연산자 뒤에 있는 도출된 식은 표준적인 대입 연산자를 사용하여 변수에 직접 대입할 수 있습니다. 단, 이 값은 표준적인 대입 연산자 **+=**, **-=**, ***=**, **/=**, **%=**, **&=**, **|=**, **^=**, **>>=**, **<<=**를 사용하여 산술적 또는 논리적으로 변수의 기존 값과 조합할 수도 있습니다. 또한 증가 및 감소 연산자 **++** 및 **--**는 뒤에 식이 없어도 사용할 수 있습니다.

변수의 실제 할당을 프로그램의 다음 명령 동작이 실제로 실행 시작할 때까지 지연되는 경우에 동기 대입 연산자를 사용할 수 있습니다. 모션 프로그램에서 여러 동작의 블렌드 또는 스플라인 시퀀스 중에 출력을 설정할 때 도움이 됩니다. 모션 프로그램에서는 동작을 어떻게 블렌드 또는 스플라인할지 구하기 위해 하나 이상의 선행 동작을 계산해야 하기 때문입니다. 표준적인 대입 연산자에서는 값이 설정되는 타이밍이 너무 빠릅니다. 동기 할당을 설정하면 다음 동작이 실행 시작할 때까지 실제 할당을 지연할 수 있기 때문에 직관적으로 기대되는 시간에 출력을 실시합니다.

또한 동기 할당은 동작 종료를 확인하기 전에 명령된 동작이 시작되도록 모터 또는 축 동작을 직접 명령하는 PLC 프로그램에서도 도움이 됩니다. 모터 동작에서는 모터를 좌표계에 할당해야 한다는 점에 주의하십시오. 지연 할당을 유지하는 동기 대입 큐는 좌표계에 속하기 때문입니다.

보통 명령 시에 구한 식의 값은 == 동기 대입 연산자를 사용하여 다음 동작의 실행 시작 시에 변수에 직접 대입됩니다. 단, 이 값은 산술 동기 변수 대입 연산자 +=, -=, *=, /=, %= 또는 논리 동기 대입 연산자 &=, |=, ^=를 사용하여 산술적 또는 논리적으로 변수의 기존 값과 조합할 수도 있습니다. 또한 동기 증가 및 감소 연산자 ++ 및 --는 뒤에 식이 없어도 사용할 수 있습니다.

산술 동기 대입 연산자는 부동 소수점인 'M' 변수만으로 사용할 수 있고 논리 동기 대입 연산자는 고정 소수점(정수)인 'M' 변수만으로 사용할 수 있다는 점에 주의하십시오. 어떤 종류의 동기 대입 연산자도 자기 참조 'M' 변수, 로컬 변수 또는 기능의 데이터 구조체 요소에서 사용할 수 없습니다.

예

```
M1=17.5;           // Set variable M1 to 17.5
M(L10)+=3;         // Add 3 to M-variable numbered by L10
M(P10+P17)=5*sqrt(P100); // Set M-var numbered by (P10+P17) to expression value
M25==1;           // Set variable M25 to 1 at start of next move execution
M50++;            // Increment M50 at start of next move execution
M100,24=0;        // Set M100 - M123 to 0
M20,5,10^=1;      // Logically invert M20, M30, M40, M50, M60
```

N{상수}:

기능: 행 점프 레이블 프로그램

구문: N{상수}:

여기에서

- {상수}는 부호가 없는 32 비트 정수(0~4,294,967,295)입니다.

이것은 숫자로 된 점프 행 레이블로 **goto**, **gosub**, **callsub**, **call**, **G**, **M**, **T**, **D** 프로그램 명령 또는 **B** 온라인 명령에 따라 모션 프로그램 실행 플로를 해당 행까지 점프할 수 있습니다.

점프 레이블로서 처리하려면 상수값 직후에 콜론(스페이스 없음)을 붙여야 한다는 점에 주의하십시오. 콜론이 없는 경우에는 동기 **Status** 레이블로 처리됩니다. 두 기능을 유효하게 하고자 하는 경우에 같은 행에 두 타입의 레이블을 배치할 수 있습니다.

어떤 행으로 점프할 경우에 그 행의 점프 레이블은 하나로 충분합니다. 행 레이블은 번호순으로 나열할 필요는 없습니다. 레이블은 행 선두에 두어야 합니다.

기본값에서는 프로그램에 행 점프 레이블용으로 주소 포인터 1,024 개가 예약됩니다. 그보다 더 많은 점프 레이블이 필요한 경우에는 프로그램 다운로드를 시작하는 **open prog** 또는 **open plc** 명령 내에서 예약할 주소 포인터 수를 명확하게 선언해야 합니다. 또한 예약수를 기본값보다 작게 하면 메모리 용량이 확보되어 변수 점프 검색 속도가 향상됩니다.



참고사항

기본값에서는 각 모션 프로그램 선두에 암묵적으로 **N0:**이 있습니다. 단, 프로그램의 다른 곳에서 명확하게 **N0:**을 두면 선두의 N0: 대신에 명확한 **N0:**이 점프에 사용됩니다.

N{데이터}

기능: 동기 Status 레이블 프로그램

구문: **N{데이터}**

여기에서

- **{데이터}**는 레이블의 값을 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

숫자로 된 동기 Status 레이블이며 Power PMAC 은 2 개의 사전 정의된 데이터 구조체 요소를 **{데이터}**값으로 자동으로 설정합니다. 하나는 프로그램 계산이 레이블에 도달하면 다른 하나는 다음 동작 실행이 실제로 시작됐을 때에 설정됩니다. 이를 통해 사용자는 프로그램 및 동작 실행 Status 를 간단하게 감시할 수 있습니다.

동기 Status 레이블 뒤에 콜론은 붙지 않습니다. 레이블 뒤에 콜론이 붙으면 대신 점프 레이블이 됩니다. 두 기능을 유효하게 하고자 하는 경우에 같은 행에 두 타입의 레이블을 배치할 수 있습니다.

프로그램 계산 중에 동기 Status 레이블이 출현하면 **{데이터}**값을 평가합니다. 필요에 따라 32 비트 정수값으로 버립니다. 즉시 데이터 구조체 요소 [Coord[x].Ncalc]를 이 값으로 설정합니다.

그리고 이 값으로 데이터 구조체 요소 [Coord[x].Nsync]의 동기 할당 지시를 동기 할당 큐에 넣습니다. 모션 프로그램 내의 다음 동작이 실제로 실행 시작한 시점에 이 지시가 큐에서 호출되어 실행됩니다. 프로그램이 블렌드 동작 시퀀스에서 예측하는 경우에 실제로는 초기 할당 후에 동기 할당이 실행됩니다.

이러한 2 개의 데이터 요소를 감시하면 프로그램 계산 및 동작 실행의 진척 상황을 언제든지 확인할 수 있습니다. 주로 모션 프로그램 실행을 감시하는 목적으로 사용되지만 PLC 프로그램에서도 사용할 수 있습니다(이것은 프로그램의 [Ldata.Coord]로 지정한 좌표계 요소에 영향을 끼칩니다).

nofrax

기능: 벡터 피드 레이트 축을 클리어함

구문: **nofrax**

nofrax 명령은 다음 피드 레이트 지정(F) 동작에서는 좌표계의 어떤 축도 벡터 피드 레이트축으로 처리하지 않는다는 것을 지정합니다. 이러한 동작에서는 Power PMAC 은 데이터 구조체 요소 [Coord[x].AltFeedrate](lsx86)를 거리가 최대인 축에 사용하여 동작 시간을 계산합니다.

nofrax 명령은 [Coord[x].FRAxes] 32 비트 워드(각 축당 1 비트)의 모든 비트를 클리어하면 기능합니다. 이러한 비트는 후속 **frax** 명령으로 어떠한 비트가 설정될 때까지 클리어됩니다.

normal{ 벡터}{데이터} [{ 벡터}{데이터}...]

기능: 법선 벡터 평면 지정

구문: **normal{ 벡터}{데이터} [{ 벡터}{데이터}...]**

여기에서

- **{ 벡터 }**는 각각 X, Y, Z 축과 평행한 벡터 컴포넌트 문자(I, J, K) 또는 각각 XX, YY, ZZ 축과 평행한 벡터 컴포넌트 2 문자(II, JJ, KK)입니다.
- **{ 데이터 }**는 벡터 컴포넌트를 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

normal 구문은 X/Y/Z 직교축 세트 또는 XX/YY/ZZ 축 직교축 세트인 3D 공간으로 원호 보간, 2D 공구(커터) 직경 보정 및 코너 각도 보정이 수행되는 평면의 방향을 지정합니다. 평면에 대한 법선(수직) 벡터 정의로 지정합니다.

X/Y/Z 축에 지정할 수 있는 벡터 컴포넌트는 I(X 축 방향), J(Y 축 방향), K(Z 축 방향)입니다. XX/YY/ZZ 축에 지정할 수 있는 벡터 컴포넌트는 II(X 축 방향), JJ(Y 축 방향), KK(Z 축 방향)입니다. 컴포넌트 값 사이의 비율이 법선 벡터 방향, 즉 보정 평면 방향을 결정합니다. 벡터 길이는 문제가 되지 않습니다. 단위 벡터일 필요는 없습니다(Power PMAC 은 저장 시에 자동으로 벡터를 단위 크기로 정규화합니다).

벡터 방향 감지는 문제가 됩니다. 원호 동작의 시계 방향 감지 및 공구 보정 오프셋 감지를 정의하기 위함입니다. Power PMAC 은 우회전 규칙, 즉 우회전 좌표계(**I** x **J** = **K**)를 사용합니다. 오른손 엄지가 여기서 지정한 법선 벡터의 방향을 가리키면 다른 손가락은 원형 평면의 시계 방향인 원호 방향 및 보정 평면에 대한 이동 방향에서 우측으로 오프셋한 방향으로 구부러집니다. 일반적으로 음의 법선 벡터는 예상되는 시계 방향/시계 반대 방향 감지를 생성합니다.

전원 셋업 시 기본값은 X/Y/Z 축 세트에서는 K-1.0 법선 벡터가 X/Y 평면을 지정하고 XX/YY/ZZ 축 세트에서는 KK-1.0 법선 벡터가 XX/YY 평면을 지정합니다.

예

주평면에 원을 그리기 위한 표준적인 설정은 다음과 같습니다.

```
normal K-1;           // XY plane -- equivalent to G17 NC code
normal J-1;           // ZX plane -- equivalent to G18 NC code
normal I-1;           // YZ plane -- equivalent to G19 NC code

normal KK-1;          // XX/YY plane
normal JJ-1;          // ZZ/XX plane
normal II-1;          // YY/ZZ plane
```

여러 벡터 컴포넌트를 사용하면 주평면으로 기울어진 평면을 정의할 수 있습니다.

```
normal I0.866 J0.500;
normal JJ25 KK-25;
normal J(-SIN(Q1)) K(-COS(Q1));
```

nxxyz

기능: 3D 공구 직경 보정용으로 면 법선 벡터를 지정함

구문: **nxxyz{벡터}{데이터}{{벡터}{데이터}...}**

여기에서

- **{벡터}**는 각각 X, Y, Z 축과 평행한 벡터 컴포넌트를 지정하는 문자(I, J, K)입니다.
- **{데이터}**는 벡터 컴포넌트를 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

nxxyz 명령은 3D 공구 직경 보정에 쓰이는 면 법선 벡터를 지정합니다. 벡터 컴포넌트를 X, Y, Z 축에 따라 설정하여 지정합니다. 각각 I, J, K 문자 뒤의 값으로 설정하는 이러한 컴포넌트의 절대량은 문제가 되지 않습니다. 컴포넌트의 상대값이 벡터 방향을 결정합니다. 벡터는 가공면에서 공구로 향하는 방향이어야 합니다.

면 법선 벡터는 기반이 되는 머신 좌표계에서 정의해야 합니다. 좌표계의 XYZ 공간을 변환 매트릭스로 회전하는 경우에 벡터 좌표는 프로그래밍 좌표와 다릅니다.

이 명령으로 정의한 면 법선 벡터는 새로운 면 법선 벡터가 지정될 때까지 모션 프로그램의 동일한 행에서 명령된 동작 및 모든 후속 동작의 보정 오프셋 결정에 쓰입니다. 단, 3D 보정에서는 명령된 동작마다 개별적으로 면 법선 벡터가 지정되는 것이 표준적인 방법입니다.

면 법선 벡터의 컴포넌트가 **nxxyz** 명령으로 명확하게 선언되지 않은 경우에 그 컴포넌트는 자동으로 0.0 으로 설정됩니다.

P{데이터}{대입 연산자}{식}

기능: 지정된 'P' 변수에 값을 할당함

구문: **P{데이터}[{식}]{대입 연산자}{식}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 값을 할당하는 첫 번째 변수 번호를 나타내는 소괄호 내의 수식입니다.
- 첫 번째 옵션 **{식}**은 값을 할당하는 변수의 수를 지정합니다. 명확하게 지정되지 않은 경우에는 1이 사용되고 값은 하나의 변수에만 할당됩니다.
- 두 번째 옵션 **{식}**은 값을 할당하는 변수의 숫자 간격을 지정합니다. 명확하게 지정되지 않은 경우에는 1이 사용되고 값은 연속된 번호의 변수에 할당됩니다.
- **{대입 연산자}**는 식에서 값을 변수에 할당하는 방법을 제어하는 숫자 연산자입니다.
- 마지막 **{식}**은 상수, 변수, 논리, 산술 연산자 및 함수를 조합한 수식이며 도출된 식의 값이 할당에 사용됩니다.

P{데이터}{대입 연산자}{식} 명령에 따라 지정된 'P' 변수에 식에서 구한 값이 들어갑니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수 번호를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~65,535). 유효한 변수 번호를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수 번호가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 구해야 합니다(필요에 따라 다음 정수로 버림). 구해지지 않는 경우에 명령은 실행하지 않는 것으로 처리됩니다.

'P' 변수는 글로벌 사용자 변수입니다. 선택한 모든 기능을 제공할 수 있습니다. 지정된 'P' 변수에 기본적인 문자/번호 형식(**P125** 등)으로 이름을 직접 할당합니다. 또한 IDE 소프트웨어 사용 시에는 선언된 global 변수명('P' 변수에 자동으로 할당됨) 또는 특정 'P' 변수에 부여되는 이름에 따라 **#define** 텍스트의 대체로 변수에 이름을 직접 할당할 수 있습니다. 후자의 경우에 기본적인 문자/번호 형식의 변수명으로 대체하는 것은 다운로드 시 자동으로 이루어집니다.

보통 연산자 뒤에 있는 도출된 식은 표준적인 대입 연산자를 사용하여 변수에 직접 대입할 수 있습니다. 단, 이 값은 표준적인 대입 연산자 **+=**, **-=**, ***=**, **/=**, **%=**, **&=**, **|=**, **^=**, **>>=**, **<<=**를 사용하여 산술적 또는 논리적으로 변수의 기존 값과 조합할 수도 있습니다. 또한 증가 및 감소 연산자 **++** 및 **--**는 뒤에 식이 없어도 사용할 수 있습니다.

변수의 실제 할당을 프로그램의 다음 명령 동작이 실제로 실행 시작될 때까지 늦추고자 하는 경우에 동기 대입 연산자를 사용할 수 있습니다. 모션 프로그램에서 여러 동작의 블렌드 또는 스플라인 시퀀스 중에 출력을 설정할 때 도움이 됩니다. 모션 프로그램에서는 동작을 어떻게 블렌드 또는 스플라인할지 구하기 위해 하나 이상의 선행 동작을 계산해야 하기 때문입니다. 표준적인 대입 연산자에서는 값이 설정되는 타이밍이 너무 빠릅니다. 동기 할당을 설정하면 다음 동작이 실행 시작할 때까지 실제 할당을 지연할 수 있기 때문에 직관적으로 기대되는 시간에 출력을 실시합니다.

또한 동기 할당은 동작 종료를 확인하기 전에 명령된 동작이 시작되도록 모터 또는 축 동작을 직접 명령하는 PLC 프로그램에서도 도움이 됩니다. 모터 동작에서는 모터를 좌표계에 할당해야 한다는 점에 주의하십시오. 지연 할당을 유지하는 동기 할당 규는 좌표계에 속하기 때문입니다.

보통 명령 시에 구한 식의 값은 == 동기 대입 연산자를 사용하여 다음 동작의 실행 시작 시에 변수에 직접 대입됩니다. 단, 이 값은 산술 동기 변수 대입 연산자 +=, -=, *=, /=, %=을 사용하여 산술적으로 변수의 기존 값과 조합할 수도 있습니다. 또한 동기 증가 및 감소 연산자 ++ 및 --는 뒤에 식이 없어도 사용할 수 있습니다. 'P' 변수는 부동 소수점인 논리 동기 대입 연산자 &=, |=, ^=을 사용할 수 없습니다.

동일한 간격으로 나열되는 여러 변수에 같은 값을 할당할 수 있습니다. 초기 변수명 뒤에 두 번째 값이 사용되는 경우에 값이 할당되는 변수의 수를 지정합니다(두 번째 값이 사용되지 않는 경우에 기본값인 1 이 되고 초기 변수가 지정됩니다). 세 번째 값이 사용되는 경우에 변수의 간격을 수치로 지정합니다(세 번째 값이 사용되지 않는 경우에 기본값인 1 이 되며 연속된 번호의 변수가 사용됩니다).

예

```
P1=17.5;           // Set variable P1 to 17.5
P(L10)+=3;         // Add 3 to P-variable numbered by L10
P(P10+P17)=5*sqrt(P100); // Set P-var numbered by (P10+P17) to expression value
P25==1;           // Set variable P25 to 1 at start of next move execution
P50++;            // Increment P50 at start of next move execution
P100,16=0;         // Set P100 - P115 to 0
P20,5,10++;        // Increment P20, P30, P40, P50, P60
```

ause

기능: 지정한 좌표계에서 모션 프로그램 실행을 일시 정지함

구문: **pause [{목록}]**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수 세트 및 시작 상수 번호에서 종료 상수 번호까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.

Power PMAC 은 **pause** 명령으로 지정된 좌표계의 모션 프로그램 실행을 마지막으로 계산된 동작의 실행 후에 일시 정지시킵니다. 프로그램의 **resume** 또는 **run** 명령으로 이 시점부터 실행을 재개할 수 있습니다.

좌표계 목록이 없는 경우에 현재 프로그램에 따라 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 목록 내의 좌표계를 지정해도 프로그램에 따라 모달에 주소 지정된 좌표계에는 영향을 미치지 않습니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 `[Plc[i].Ldata.Coord]`에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(`Coord[x].Ldata.Coord`)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

정지 모드 중에 모션 프로그램 실행은 일시 정지되지만 논리적으로는 아직 실행 중인 상태입니다. 일시 정지 후에도 프로그램 실행을 재개하지 않는 경우에 **Abort** 명령을 사용하여 프로그램 실행을 완전히 **Abort** 해야 합니다.

상당하는 온라인 명령은 생략형 **q** 를 사용하며 그 앞에 좌표계 목록이 놓인다는 점(&1,2q 등)에 주의하십시오.

예

```
pause;           // Pause motion program in addressed C.S.
pause 2;         // Pause motion program in C.S.2
pause 1..16;     // Pause motion program in C.S.1 thru 16
pause 3..6,8;    // Pause motion program in C.S.3, 4, 5, 6, 8
```

pause plc

기능: 지정된 PLC 프로그램 동작을 일시 정지함

구문: **pause plc {목록}**

여기에서

- **{목록}**은 무효로 할 PLC 프로그램을 지정하는 하나 이상의 상수 세트 또는 0~31의 범위 상수입니다.

pause plc 명령에 따라 Power PMAC 은 지정된 PLC 프로그램 실행을 현재 실행 시점에서 일시 정지합니다. **resume plc** 명령으로 이 시점부터 실행을 재개합니다.

PLC 프로그램은 번호(0~31)에 따라 지정됩니다. 이 명령에서는 하나의 번호만 지정된 경우에 여러 번호가 쉼표로 구분되어 지정되는 경우와 연속된 번호의 변수 범위가 지정되는 경우가 있습니다.

예

```
pause plc 0;
pause plc 1,2,5;
pause plc 1..16;
pause plc 3..6,8;
```

pclear

기능: 모든 축/모터 위치를 제로로 설정함

구문: **pclear**

pclear 명령에 따라 Power PMAC 은 좌표계 모든 축의 현재 위치를 제로로 재정의합니다. 또한 축 정의 관계를 유효한 상태로 유지하기 위해 이러한 축에 정의된 모든 모터 위치를 재정의합니다. 이 명령은 어느 축에 있어서도 동작은 발생하지 않습니다. 좌표계 내의 모든 축의 현재 명령 위치는 지정된 값으로 설정되고 축 정의 관계를 유효한 상태로 유지하기 위해 축에 정의된 모터 위치가 조정됩니다.

pclear 명령은 **pset{축}0{축}0...** 명령 또는 좌표계 모든 축의 **homez** 명령(절대 전원 셋업 위치 없음)에 상당합니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램을 실행하는 좌표계의 축 및 모터가 영향을 받습니다. 이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 요소 [Plc[i].Ldata.Coord]의 대상 좌표계의 축 및 모터가 영향을 받습니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

PMAC 및 Turbo PMAC 과 달리 Power PMAC 의 **pset** 및 유사한 명령(**pclear** 등)은 모터의 제로 위치를 변경합니다. 또한 소프트웨어 리미트 및 보정 테이블을 참조하는 프레임도 변경됩니다. 모터 제로를 변경하지 않으면서 참조하는 축 프로그래밍 프레임을 변경하려면 축 변환 매트릭스를 사용하십시오.

pload

기능: 저장된 **pset** 오프셋을 복원함

구문: **pload**

pload 명령에 따라 Power PMAC 은 마지막으로 저장한 **pset** 명령의(또는 **pset** 명령이 전송되기 전부터) 모터 오프셋의 영향을 받은 좌표계에 복원합니다. 이러한 오프셋은 최신 **pstore** 명령으로 저장할 수 있습니다. 이 명령은 좌표계 축 및 모터의 현재 위치를 변경하고 동작은 실행하지 않습니다. **pset** 명령과 같은 효과가 있습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램을 실행하는 좌표계의 축 및 모터가 영향을 받습니다. 이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 요소 [Plc[i].Ldata.Coord]의 대상 좌표계의 축 및 모터가 영향을 받습니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

PMAC 및 Turbo PMAC 과 달리 Power PMAC 의 **pset** 명령은 모터의 제로 위치를 변경합니다. 또한 소프트웨어 리미트 및 보정 테이블을 참조하는 프레임도 변경됩니다. 모터 제로를 변경하지 않으면서 참조하는 축 프로그래밍 프레임을 변경하려면 축 변환 매트릭스를 사용하십시오.

pmatch

기능: 축 위치와 모터 위치의 재위치 맞춤

구문: **pmatch**

Power PMAC 은 **pmatch** 명령으로 좌표계의 축 시작 위치를 재계산하고 현재 모터 명령 위치에 위치 맞춤을 합니다. 위치 맞춤은 축 정의 구문 방정식을 반전해서 축 위치를 계산하고 좌표축에 포워드 키네마틱스 서브루틴이 있는 경우에는 이 서브루틴을 실행합니다.

이 기능은 모션 프로그램이 실행을 명령받을 때마다 최신 동작이 올바르게 계산되는지 확인하기 위해 Power PMAC 에 따라 자동으로 실행됩니다(명확한 **pmatch** 명령이 없는 경우).

단, 최상위 프로그램이 PLC 프로그램인 경우에 축 동작이 프로그램에서 명령받을 때까지 Power PMAC 은 이 기능을 자동으로 실행하지 않습니다. 마지막으로 명령된 축 동작 후에 좌표축 모터가 동작한 가능성이 있는 경우에 또는 마지막으로 명령된 축 동작 후에 축/모터 관계가 변화한 가능성이 있는 경우에 Power PMAC 이 올바른 축 시작 위치를 계산하기 위해 최초로 **pmatch** 명령을 실행해야 합니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램을 실행하는 좌표계의 축 및 모터가 영향을 받습니다. 이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 요소 [Plc[i].Ldata.Coord]의 대상 좌표계의 축 및 모터가 영향을 받습니다. PLC 프로그램에서는 명령 **Ldata.Coord={속}**으로 좌표계를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

현재 축/모터의 관계에 따라 축과 모터의 위치가 일치하지 않을 때 축 동작을 명령하면 잘못된 축 시작 포인트를 사용하여 Power PMAC 이 동작을 계산하므로 동작 시작 시에 급격한 시작 포인트로의 조작이 발생합니다.

pread

기능: 축 현재 위치를 통지함

구문: **pread**

Power PMAC 은 **pread** 명령으로 프로그램의 [Ldata.Coord]에 따라 주소 지정된 좌표계의 축 현재 위치를 계산합니다. 좌표계의 모든 활성 축의 현재 위치가 계산됩니다.

축의 현재 위치는 좌표계 정의에 따라 처리되어(축 정의 명령문을 반전하거나 포워드 키네마틱스 루틴을 사용하여) 대응할 모터의 현재 위치 및 유효한 매트릭스 변환으로 계산됩니다.

축 현재 위치의 값은 프로그램의 로컬 'D' 변수에 복사되고 후속 계산에서 사용됩니다. 위치는 유효한 축 변환 매트릭스를 사용한 현재 축의 단위로 복귀합니다. 각 축에 사용되는 'D' 변수 'D'는 다음 표와 같습니다.

축	D_i	축	D_i	축	D_i	축	D_i	축	D_i	축	D_i	축	D_i	축	D_i
A	0	V	4	Z	8	DD	12	HH	16	OO	20	SS	24	WW	28
B	1	W	5	AA	9	EE	13	LL	17	PP	21	TT	25	XX	29
C	2	X	6	BB	10	FF	14	MM	18	QQ	22	UU	26	YY	30
U	3	Y	7	CC	11	GG	15	NN	19	RR	23	VV	27	ZZ	31

'D32'는 32 비트의 마스크를 반환하고 현재 위치가 통지된 축을 통지합니다. 'D32'의 각 비트 i 는 변수가 통지된 변수 'D'의 번호에 해당합니다. 'D32'의 반환값이 0 인 경우에 위치는 통지되지 않습니다.

버퍼 프로그램 **pread** 명령은 온라인 **p** 명령과 같은 계산을 실행합니다.

pset{축}{데이터}[[축]{데이터}...]

기능: 축과 모터의 위치를 재정의함

구문: **pset**{축}{데이터}[[축]{데이터}...]

여기에서

- **{축}**은 모터 위치를 재정의할 축을 지정하는 1 문자 또는 2 문자(A, B, C, U, V, W, X, Y, Z, AA, BB, ... HH, LL, MM, ... ZZ)입니다.
- **{데이터}**는 종료 위치 또는 거리를 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.
- **[[축]{데이터}...]**는 여러 축을 동시에 재정의하는 옵션을 지정합니다.

pset 명령을 사용하여 지정된 축과 그 축에 정의된 모터의 현재 위치를 재정의할 수 있습니다. 이 명령은 어느 축에 있어서도 동작은 발생하지 않습니다. 명령 목록 내의 모든 축의 현재 명령 위치값은 지정된 값으로 설정되며 축 정의 관계를 유효한 상태로 유지하기 위해 축에 정의된 모든 모터 위치가 조정됩니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램을 실행하는 좌표계의 축 및 모터가 영향을 받습니다. 이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 요소 [Plc[i].Ldata.Coord]의 대상 좌표계의 축 및 모터가 영향을 받습니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

PMAC 및 Turbo PMAC 과 달리 Power PMAC 의 **pset** 명령은 모터의 제로 위치를 변경합니다. 또한 소프트웨어 리미트 및 보정 테이블을 참조하는 프레임도 변경됩니다. 모터 제로를 변경하지 않으면서 참조하는 축 프로그래밍 프레임을 변경하려면 축 변환 매트릭스를 사용하십시오.

내부적으로는 이 명령으로 지정된 축에 정의된 각 모터의 데이터 구조체 요소 [Motor[x].HomePos] 값을 변경합니다. 이 레지스터가 모터 제로 위치와 기본이 되는 포지션 센서(엔코더 카운터 등)의 제로 위치와의 차이를 유지합니다.

예

<code>pset X10 Y20;</code>	<code>// Set present X position to 10 units, Y to 20</code>
<code>pset A(P1) Y(P2);</code>	<code>// Set present A position to value of P1, B to P2</code>

pstore

기능: 현재 **pset** 오프셋을 저장함

구문: **pstore**

pstore 명령은 Power PMAC 이 **pset** 명령에 영향을 받은(또는 **pset** 명령이 전송되기 전부터) 좌표계의 현재 모터 오프셋을 저장합니다. 이러한 오프셋은 나중에 **pload** 명령으로 복원할 수 있습니다. **pstore** 명령은 활성 오프셋의 변경 및 동작을 실행하지 않습니다. 한 좌표계당 1 세트의 오프셋만 저장됩니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 모션 프로그램을 실행하는 좌표계의 축 및 모터가 영향을 받습니다. 이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 요소 [Plc[i].Ldata.Coord]의 대상 좌표계의 축 및 모터가 영향을 받습니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

PMAC 및 Turbo PMAC 과 달리 Power PMAC 의 **pset** 명령은 모터의 제로 위치를 변경합니다. 또한 소프트웨어 리미트 및 보정 테이블을 참조하는 프레임도 변경됩니다. 모터 제로를 변경하지 않으면서 참조하는 축 프로그래밍 프레임을 변경하려면 축 변환 매트릭스를 사용하십시오.

pvt{데이터}

기능: 위치-속도-시간 동작 모드 설정

구문: **pvt{데이터}**

여기에서

- **{데이터}**는 동작 시간을 밀리초로 나타내는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

pvt 명령은 모션 프로그램을 위치-속도-시간 동작 모드로 합니다. **{데이터}**의 값을 구하여 밀리초로 나타내는 시간으로 프로그램 동작에 사용합니다. 모션 프로그램이 이미 PVT 모드인 경우에 명령을 사용하여 동작 시간을 변경할 수 있습니다(정지하지 않습니다). 모션 프로그램의 후속 동작 명령은 이 모드의 규칙에 따라 처리됩니다.

Turbo PMAC 은 PVT 모드 시간을 **ta** 또는 **tm** 명령으로 설정 또는 변경할 수 있지만 Power PMAC에서는 불가능합니다.

pvt 명령을 실행하면 그 외의 동작 모드(**circle**, **linear**, **rapid**, **spline**)가 종료됩니다. 그 외의 동작 모드를 명령하면 **pvt** 모드가 종료됩니다.

예

```
pvt200;           // Set PVT mode with move time of 200 msec
pvt(L10);         // Set PVT mode with move time of (L10) msec
```

Q{데이터}{대입 연산자}{식}

기능: 지정된 'Q' 변수에 값을 할당함

구문: **Q{데이터}{식}{식}**
{대입 연산자}{식}

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 값을 할당하는 첫 번째 변수 번호를 나타내는 소괄호 내의 수식입니다.
- 첫 번째 옵션 **{식}**은 값을 할당하는 변수의 수를 지정합니다. 명확하게 지정되지 않은 경우에는 1이 사용되고 값은 하나의 변수에만 할당됩니다.
- 두 번째 옵션 **{식}**은 값을 할당하는 변수의 숫자 간격을 지정합니다. 명확하게 지정되지 않은 경우에는 1이 사용되고 값은 연속된 번호의 변수에 할당됩니다.
- **{대입 연산자}**는 식에서 값을 변수에 할당하는 방법을 제어하는 숫자 연산자입니다.
- 마지막 **{식}**은 상수, 변수, 논리, 산술 연산자 및 함수를 조합한 수식이며 도출된 식의 값이 할당에 사용됩니다.

Q{데이터}{대입 연산자}{식} 명령에 따라 지정된 'Q' 변수에 식에서 구한 값이 들어갑니다. 변수 번호가 상수로 지정된 경우에 그 상수는 유효한 변수 번호를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~8,191). 유효한 변수 번호를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수 번호가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음 정수로 버림). 구해지지 않는 경우에 명령은 실행하지 않는 것으로 처리됩니다.

'Q'는 글로벌(여러 프로그램에서 사용 가능) 변수이지만 좌표계 고유(서로 다른 좌표계와 독립적)의 사용자 변수는 선택한 모든 기능을 제공할 수 있습니다. 지정된 'Q' 변수에 기본적인 문자/번호 형식(Q125 등)으로 이름을 직접 할당합니다. 또한 IDE 소프트웨어 사용 시에는 선언된 csglobal 변수명('Q' 변수에 자동으로 할당됨) 또는 특정 'Q' 변수에 부여되는 이름에 따라 #define 텍스트의 대체로 변수에 이름을 직접 할당할 수 있습니다. 후자의 경우에 기본적인 문자/번호 형식의 변수명으로 대체하는 것은 다운로드 시 자동으로 이루어집니다.

보통 연산자 뒤에 있는 도출된 식은 표준적인 대입 연산자를 사용하여 변수에 직접 대입됩니다. 단, 이 값은 표준적인 대입 연산자 +=, -=, *=, /=, %=, &=, |=, ^=, >>=, <<=를 사용하여 산술적 또는 논리적으로 변수의 기존 값과 조합할 수도 있습니다. 또한 증가 및 감소 연산자 ++ 및 --는 뒤에 식이 없어도 사용할 수 있습니다.

변수의 실제 할당을 프로그램의 다음 명령 동작이 실제로 실행 시작될 때까지 늦추고자 하는 경우에 동기 대입 연산자를 사용할 수 있습니다. 모션 프로그램에서 여러 동작의 블렌드 또는 스플라인 시퀀스 중에 출력을 설정할 때 도움이 됩니다. 모션 프로그램에서는 동작을 어떻게 블렌드 또는 스플라인할지 구하기 위해 하나 이상의 선행 동작을 계산해야 하기 때문입니다. 표준적인 대입 연산자에서는 값이 설정되는 타이밍이 너무 빠릅니다. 동기 할당을 설정하면 다음 동작이 실행 시작할 때까지 실제 할당을 지연할 수 있기 때문에 직관적으로 기대되는 시간에 출력을 실시합니다.

또한 동기 대입은 동작 종료를 확인하기 전에 명령된 동작이 시작되도록 모터 또는 축 동작을 직접 명령하는 PLC 프로그램에서도 도움이 됩니다. 모터 동작에서는 모터를 좌표계에 할당해야 한다는 점에 주의하십시오. 지연 할당을 유지하는 동기 할당 규는 좌표계에 속하기 때문입니다.

보통 명령 시에 구한 식의 값은 == 동기 대입 연산자를 사용하여 다음 동작의 실행 시작 시에 변수에 직접 대입됩니다. 단, 이 값은 산술 동기 변수 대입 연산자 +=, -=, *=, /=, %=을 사용하여 산술적으로 변수의 기존 값과 조합할 수도 있습니다. 또한 동기 증가 및 감소 연산자 ++= 및 --=는 뒤에 식이 없어도 사용할 수 있습니다. 'Q' 변수는 부동 소수점인 논리 동기 대입 연산자 &=, |=, ^=을 사용할 수 없습니다.

동일한 간격으로 나열되는 여러 변수에 같은 값을 할당할 수 있습니다. 초기 변수명 뒤에 두 번째 값이 사용되는 경우에 값이 할당되는 변수의 수를 지정합니다(두 번째 값이 사용되지 않는 경우에 기본값인 1 이 되고 초기 변수가 지정됩니다). 세 번째 값이 사용되는 경우에 변수의 간격을 수치로 지정합니다(세 번째 값이 사용되지 않는 경우에 기본값인 1 이 되며 연속된 번호의 변수가 사용됩니다).

예

```
Q1=17.5;           // Set variable Q1 to 17.5
Q(L10)+=3;         // Add 3 to Q-variable numbered by L10
Q(P10+P17)=5*sqrt(P100); // Set Q-var numbered by (P10+P17) to expression value
Q25==1;           // Set variable Q25 to 1 at start of next move execution
Q50++=;           // Increment Q50 at start of next move execution
Q100,16=0;        // Set Q100 - Q115 to 0
Q20,5,10++;       // Increment Q20, Q30, Q40, Q50, Q60
```

R{데이터}[대입 연산자]{식}

기능: 지정된 'R' 변수에 값을 할당함

구문: **R{데이터}[{식}]{대입 연산자}{식}**

여기에서

- **{데이터}**는 음의 값이 아닌 정수 상수(소괄호 없음) 또는 값을 할당하는 첫 번째 변수 번호를 나타내는 소괄호 내의 수식입니다.
- 첫 번째 옵션 **{식}**은 값을 할당하는 변수의 수를 지정합니다. 명확하게 지정되지 않은 경우에는 1 이 사용되고 값은 하나의 변수에만 할당됩니다.

- 두 번째 옵션 **{식}**은 값을 할당하는 변수의 숫자 간격을 지정합니다. 명확하게 지정되지 않은 경우에는 1이 사용되고 값은 연속된 번호의 변수에 할당됩니다.
- **{대입 연산자}**는 식에서 값을 변수에 할당하는 방법을 제어하는 숫자 연산자입니다.
- 마지막 **{식}**은 상수, 변수, 논리, 산술 연산자 및 함수를 조합한 수식이며 도출된 식의 값이 할당에 사용됩니다.

R{데이터}{대입 연산자}{식} 명령에 따라 지정된 'R' 변수에 식에서 구한 값이 들어갑니다. 변수가 상수 번호로 지정된 경우에 그 상수는 유효한 변수 번호를 나타내는 정수값이어야 합니다(표준 Power PMAC 펌웨어에서는 0~8,191). 유효한 변수 번호를 나타내는 정수값이 아닌 경우에 명령은 에러로 거부됩니다. 변수가 수식으로 지정된 경우에 그 식은 유효한 변수 번호로 평가되어야 합니다(필요에 따라 다음의 정수로 버림). 평가되지 않는 경우에 명령은 실행하지 않는 것으로 처리됩니다.

'R' 변수는 프로그램 내부에서 사용되는 로컬 스택 변수이며 저레벨 루틴에 값을 전달합니다. 각 좌표계에는 최상위 모션 프로그램 및 서브 프로그램에서 사용되는 독자적인 'R' 변수 세트가 있습니다. 또한 각 PLC 프로그램에는 모션 프로그램에서 사용하는 세트가 있으며 각 통신 스레드 자체에도 세트가 있습니다. 'R' 변수의 모든 세트는 각각 독립적입니다.

보통 연산자 뒤에 있는 도출된 식은 = 대입 연산자를 사용하여 변수에 직접 대입할 수 있습니다. 단, 이 값은 대입 연산자 +=, -=, *=, /=, %=, &=, |=, ^=, >>=, <<=를 사용하여 산술적 또는 논리적으로 변수의 기존 값과 조합할 수도 있습니다. 또한 증가 및 감소 연산자 ++ 및 --는 뒤에 식이 없어도 사용할 수 있습니다.

'R' 변수는 로컬 변수이기 때문에 글로벌 변수에서 사용 가능한 지연 동기 대입 연산자를 사용할 수 없습니다. 실제로 할당 시에 컨텍스트가 소실될 가능성이 있기 때문입니다.

동일한 간격으로 나열되는 여러 변수에 같은 값을 할당할 수 있습니다. 초기 변수명 뒤에 두 번째 값이 사용되는 경우에 값이 할당되는 변수의 수를 지정합니다(두 번째 값이 사용되지 않는 경우에 기본값인 1이 되고 초기 변수가 지정됩니다). 세 번째 값이 사용되는 경우에 변수의 간격을 수치로 지정합니다(세 번째 값이 사용되지 않는 경우에 기본값인 1이 되며 연속된 번호의 변수가 사용됩니다).

예

```
R1=17.5;           // Set variable R1 to 17.5
R(P1)+=3;          // Add 3 to R-variable numbered by P1
R(P1+P17)=5*sqrt(P100); // Set R-var numbered by (P1+P17) to expression value
R10,16=0;          // Set R10 - R25 to 0
R20,5,10++;        // Increment R20, R30, R40, R50, R60
```

rapid

기능: Rapid(빠른 이송) 동작 모드를 설정함

구문: **rapid**

rapid 명령은 모션 프로그램을 최소 시간점 간 동작을 실행하는 Rapid 동작 모드로 합니다. 모션 프로그램의 후속 동작 명령은 이 모드의 규칙에 따라 처리됩니다.

rapid 명령을 실행하면 그 외의 동작 모드(**circle**, **pvt**, **linear**, **spline**)가 종료됩니다. 그 외의 동작 모드를 명령하면 Rapid 모드가 종료됩니다.

read

기능: 서브루틴용으로 인수를 읽음

구문: **read**{문자 목록}

여기에서

- **{문자}**는 송신하는 인수를 지정할 1 문자 또는 2 문자 문자(A, B, ..., Z, AA, BB, ..., ZZ)입니다.
- **{문자 목록}**은 쉼표로 구분된 문자(1 문자 또는 2 문자) 세트 및/또는 시작 문자에서 종단 문자까지의 사이를 2 개의 마침표로 나타낸 연속된 축의 범위입니다. 사용되는 모든 문자는 알파벳 순으로 먼저 1 문자, 그 다음에 2 문자를 지정해야 합니다.

read 에 따라 서브루틴 또는 서브 프로그램은 호출 루틴에서 문자/번호 형식 인수를 취득합니다. CAD/CAM 및 CNC 어플리케이션에서 일반적으로 사용되는 표준 RS-274 'G 코드' 프로그램에서 호출 가능한 파라미터화된 서브루틴의 작성에 특히 유효합니다. 이 명령 내에 공백은 사용할 수 없습니다.

read 명령은 호출 명령(**gosub**, **callsub**, **call**, **G**, **M**, **T**, **D**)이 포함된 호출 프로그램행의 남은 부분을 보고 지정된 문자 뒤의 값을 취득하여 좌표계(최상위 프로그램이 모션 프로그램인 경우) 또는 PLC(최상위 프로그램이 PLC 프로그램인 경우)의 특정한 D 변수에 값을 넣습니다.

1 문자인 경우에 알파벳 '*n*'번째의 문자가 '**D*n***'에 들어갑니다. 2 문자인 경우에 알파벳 '*n*'번째의 문자가 '**D(26+*n*)**'에 들어갑니다. 각 좌표계(최상위 프로그램이 모션 프로그램인 경우) 및 각 PLC 프로그램에 'D' 변수 세트는 하나뿐입니다. **read** 명령을 사용한 서브루틴/서브 프로그램 네스트가 있는 경우에 매회 같은 'D' 변수 세트가 사용됩니다.

read 명령은 읽을 문자 목록에 포함되지 않는 문자 또는 문자/값의 조합이 아닌 부분이 출현할 때까지 호출 행을 스캔합니다(읽기 대상이 아닌 문자가 출현한 경우에 목록의 마지막까지 문자를 읽지 않았다 하더라도 명령은 정지한다는 점에 주의하십시오). 문자값이 'D' 변수로 읽혔어도 읽기 완료를 나타내는 비트가 'D0'으로 설정됩니다(1 문자인 경우 알파벳의 *n* 번째 문자에는 비트 *n*-1, 2 문자인 경우에는 비트 *n*+25). 최신 **read** 명령으로 취득하지 못한 문자가 있는 경우에 대응하는 비트 'D0'이 제로로 설정됩니다.

2014 년 4/4 분기에 출시된 V2.0 펌웨어부터 'D0'과 같은 비트가 'D54'에도 설정됩니다. 'D0'은 키네마틱스 루틴에도 사용되지만 여기서 설정한 'D0'값을 덮어쓸 가능성이 낮습니다. 'D54'는 **read** 명령 이외에는 사용할 수 없습니다.

호출 행에서 문자값을 취득한 경우에 문자가 서브루틴으로 파라미터를 전달만 하도록 보통의 문자 기능(축 동작 등)이 덮어씹니다. 취득되지 않은 문자값이 호출 행에 남은 경우에 서브루틴에서 복귀한 후에 보통 기능에 따라 실행됩니다.

각 문자와 연관된 'D' 변수 및 'D0'의 플래그 비트를 다음 표로 나타냅니다.

문 자	변수	D0 의 비트	비트값(1 0 진법)	비트값(1 6 진법)	문자	변수	D0 의 비트	비트값(10 진법)	비트값 (16 진법)
A	D1	0	1	\$1	AA	D27	26	67,108,864	\$4000000
B	D2	1	2	\$2	BB	D28	27	134,217,728	\$8000000
C	D3	2	4	\$4	CC	D29	28	268,435,456	\$10000000
D	D4	3	8	\$8	DD	D30	29	536,870,912	\$20000000
E	D5	4	16	\$10	EE	D31	30	1,073,741,824	\$40000000
F	D6	5	32	\$20	FF	D32	31	2,147,483,648	\$80000000
G	D7	6	64	\$40	GG	D33	32	4,294,967,296	\$100000000
H	D8	7	128	\$80	HH	D34	33	8,589,934,592	\$200000000
I	D9	8	256	\$100	II	D35	34	17,179,869,184	\$400000000
J	D10	9	512	\$200	JJ	D36	35	34,259,738,368	\$800000000
K	D11	10	1,024	\$400	KK	D37	36	68,719,476,736	\$1000000000
L	D12	11	2,048	\$800	LL	D38	37	137,438,953,472	\$2000000000
M	D13	12	4,096	\$1000	MM	D39	38	274,877,906,944	\$4000000000
N	D14	13	8192	\$2000	NN	D40	39	549,755,813,888	\$8000000000
O	D15	14	16,384	\$4000	OO	D41	40	1,099,511,627,776	\$10000000000
P	D16	15	32,768	\$8000	PP	D42	41	2,199,023,255,552	\$20000000000
Q	D17	16	65,536	\$10000	QQ	D43	42	4,398,046,511,104	\$40000000000
R	D18	17	131,072	\$20000	RR	D44	43	8,796,093,022,208	\$80000000000
S	D19	18	262,144	\$40000	SS	D45	44	17,592,186,044,416	\$100000000000
T	D20	19	524,288	\$80000	TT	D46	45	35,184,372,088,832	\$200000000000
U	D21	20	1,048,576	\$100000	UU	D47	46	70,368,744,177,664	\$400000000000
V	D22	21	2,097,152	\$200000	VV	D48	47	140,737,488,355,328	\$800000000000
W	D23	22	4,194,304	\$400000	WW	D49	48	281,474,976,710,656	\$1000000000000
X	D24	23	8,388,608	\$800000	XX	D50	49	562,949,953,421,312	\$2000000000000
Y	D25	24	16,777,216	\$1000000	YY	D51	50	1,125,899,906,842,624	\$4000000000000
Z	D26	25	33,554,332	\$2000000	ZZ	D52	51	2,251,799,813,685,248	\$8000000000000

예

```
read(a,b,c);
read(a..z);
read(d..h,q,aa..gg);
```

resume

기능: 지정한 좌표계에서 모션 프로그램을 재개함

구문: **resume**[{ 목록}]

여기에서

- **{ 목록}**은 쉼표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.

Power PMAC은 **resume** 명령으로 지정된 좌표계의 모션 프로그램 실행을 일시 정지한 시점부터 재개할 수 있습니다. 일반적으로 이 명령은 **pause** 명령으로 실행을 일시 정지한 경우에 사용합니다.

좌표계 목록이 없는 경우에 현재 프로그램에 따라 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 목록 내의 좌표계를 지정해도 프로그램에 따라 모달에 주소 지정된 좌표계에는 영향을 미치지 않습니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 `[Plc[i].Ldata.Coord]`에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(`Coord[x].Ldata.Coord`)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

예

```
resume;           // Resume motion program in addressed C.S.
resume 2;         // Resume motion program in C.S.2
resume 1..16;     // Resume motion program in C.S.1 thru 16
resume 3..6,8;    // Resume motion program in C.S.3, 4, 5, 6, 8
```

resume plc

기능: 지정된 PLC 프로그램의 실행을 재개함

구문: **resume plc {목록}**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수 또는 시작 상수에서 종로 상수까지의 사이를 2 개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 모든 상수는 0 에서 31 까지의 범위이고 실행을 재개할 PLC 프로그램을 지정합니다.

Power PMAC 은 **resume plc** 명령에 따라 지정된 PLC 프로그램을 보통의 우선순위와 타이밍으로 실행합니다. **pause plc** 명령을 사용하는 경우와 같이 프로그램 도중에 실행이 정지된 경우에도 실행을 정지한 곳에서 실행 시작됩니다.

유사한 **enable plc** 명령을 사용하면 다른 곳에서 실행이 정지한 경우에도 프로그램의 최초에서 PLC 프로그램 실행을 재개합니다.

명령으로 지정된 PLC 프로그램이 Power PMAC 에 존재하지 않는 경우에 에러가 통지되지 않습니다. 명령은 명령으로 지정된 다른 PLC 프로그램에 작용합니다.

예

```
resume plc 1;           // Resume execution of PLC 1
resume plc 2,4,6;       // Resume execution of PLCs 2, 4, and 6
resume plc 7..10;      // Resume execution of PLCs 7, 8, 9, and 10
resume plc 11,13..16,20; // Resume execution of PLCs 11, 13, 14, 15, 16, and 20
```

return

기능: 서브루틴으로 점프하고 복귀/메인 프로그램을 종료

구문: **return**

return 명령에 따라 프로그램 실행이 해당 루틴의 실행을 호출한 루틴으로 복귀합니다. **return** 명령이 최상위 프로그램에서 검출되면 프로그램 실행 포인터가 프로그램 선두로 복귀합니다. 모션 프로그램의 경우 프로그램 실행이 정지합니다. PLC 프로그램의 경우에 다음 스캔 전까지 무효가 되지 않는 한 다음 스캔 시에 프로그램 실행을 계속합니다.

프로그램 실행 중에 **return** 명령이 검출되면 실행이 호출 명령 직후의 호출 루틴 또는 프로그램으로 복귀합니다.

프로그램 버퍼에 명확하게 **return** 명령이 입력되지 않는 경우에도 프로그램 실행이 버퍼 종료 위치에 도달하면 명령이 실행되도록 각 프로그램 버퍼의 종료 위치에 암묵적인 **return** 이 있습니다.

run

기능: 지정한 좌표계에서 모션 프로그램을 실행함

구문: **run [{목록}]**

여기에서

- **{목록}**은 씬표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.

Power PMAC 은 **run** 명령으로 지정된 좌표계의 선택된 모션 프로그램 연속 실행을 프로그램 카운터의 현재 포인터에서 시작합니다.

좌표계 목록이 없는 경우에 현재 프로그램에 따라 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 목록 내의 좌표계를 지정해도 프로그램에 따라 모달에 주소 지정된 좌표계에는 영향을 미치지 않습니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 [Plc[i].Ldata.Coord]에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(Coord[x].Ldata.Coord)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

일반적으로 좌표계의 모션 프로그램은 최초에 **begin** 명령으로 선택됩니다. **step**, **pause** 또는 **hold** 명령 등으로 프로그램 실행이 정지된 경우에 **run** 명령으로 정지된 포인트에서 실행을 시작합니다.

좌표계의 이 명령이 Power PMAC 을 통해 수용되려면 좌표계가 적절한 상태여야 합니다. 적절한 상태가 아닌 경우에 Power PMAC 은 이 명령을 에러로 거부합니다. 이 명령이 수용되려면 좌표계가 다음 조건을 유지해야 합니다.

- 선택된 모션 프로그램(및 선택된 레이블)이 존재하고 유효할 것
- 좌표계의 위치 결정축에 할당된 모든 모터가 기동 중일 것([Motor[x].ServoCtrl] > 0)
- 좌표계의 위치 결정축에 할당된 모든 모터가 유효하며 클로즈드 루프일 것
- 좌표계의 위치 결정축에 할당된 모터에 양방향 오버트래블 리미트가 설정되어 있지 않을 것
- Power PMAC 에 의한 현재 모터 위치로부터의 모든 시작축 위치 계산이 가능할 것([Coord[x].Csolve] = 1)
- [Coord[x].HomeRequired] = 1 인 경우에 좌표계의 위치 결정축에 할당된 모든 모터가 정상적인 원점 서치 동작 또는 절대 위치 읽기에 따라 위치 참조를 확립했을 것
- 프로그램된 포인트가 아닌 일시 정지 위치에서 재기동하는 경우에 모든 모터 위치가 최초의 정지 및 일시 정지가 명령된 위치와 같을 것

상당하는 온라인 명령은 생략형 **r** 을 사용하며 그 앞에 좌표계 목록이 놓인다는 점(&1,2r 등)에 주의하십시오.

예

```
run;                // Start motion program in addressed C.S.
run 2;              // Start motion program in C.S.2
run 1..16;          // Start motion program in C.S.1 thru 16
run 3..6,8;         // Start motion program in C.S.3, 4, 5, 6, 8
```

S{데이터}

기능: 스핀들 데이터 코드(S 코드)

구문: **S{데이터}**

여기에서

- **{데이터}**는 나중에 사용하기 위해 저장되는 변수값을 나타내는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

Power PMAC은 **S{데이터}** 명령으로 **{데이터}**값을 프로그램을 실행 중인 좌표계의 로컬 변수 'D53'에 로드합니다. 그 이외의 동작은 실행되지 않습니다. RS-274 형식의 공작기계 프로그램에 주축 속도 정보를 전달합니다. 실제로 주축을 제어하는 알고리즘은 그 루틴 중에서 'D53'을 사용합니다. 예를 들면 클로즈드 루프의 주축에 JOG 속도를 설정 또는 오픈 루프의 주축에 전압 출력을 설정합니다.

그 이외에 'D53'은 자동으로 사용되지 않습니다. 따라서 사용자 어플리케이션이 다른 목적으로 사용하지 않는 한 값은 필요해질 때까지 이 변수에 안전하게 저장할 수 있습니다. 단, 나중에 사용하기 위해 값을 글로벌 변수에 복사할 것을 권장합니다. 모션 프로그램 및 그 서브루틴의 바깥쪽에서 데이터 구조체 요소 [Coord[x].Ldata.D[53]]을 사용하여 'D53'값에 액세스할 수 있습니다.

이 명령은 **read(S)** 명령을 사용하여 서브루틴 호출의 일부로서 전달되는 것이고 **S{데이터}** 정보와는 다르다는 점에 주의하십시오. 이 형식에서는 값이 좌표계의 로컬 변수 'D19'로 들어갑니다.

send

기능: 지정된 포트에 포맷된 문자열을 전송함

구문: **send{상수}[,]{문자열"}[, {식}...]**

여기에서

- **{상수}**는 0에서 4까지 범위 내의 정수로 문자열을 송신하는 소프트웨어 통신 포트를 지정합니다.
- **{문자열}**은 리터럴 영숫자, 이스케이프 시퀀스 및 포맷된 변수 시퀀스로 구성되는 포맷된 ASCII 문자열입니다.
- **{식}**은 문자열과 일치하는 포맷된 변수 시퀀스로 송신되는 값을 포함하는 수식입니다. 포맷된 각 변수 시퀀스마다 하나의 수식이 필요합니다.

Power PMAC은 **send** 명령으로 소프트웨어 통신 포트에 포맷된 문자열을 송신합니다. **send** 직후의 상수값(0, 1, 2, 3, 4)은 문자열을 송신하는 소프트웨어 포트 번호를 나타냅니다. 호스트 컴퓨터는 **send getsends -n(n**은 일치하는 포트 번호) **Telnet** 명령으로 문자열을 호스트로 송신할 수 있도록 Power PMAC의 'send getsends' 유틸리티를 시작합니다(IDE에서는 [Unsolicited Messages] 창을 통해 자동으로 설정되어 송신된 문자열이 표시됩니다. 따라서 디버그 어플리케이션으로 우수한 툴입니다).

프로그램 실행의 일부로서 명령이 처리되면 문자열이 작성되어 지정된 포트에서 송신되도록 버퍼됩니다. 문자열이 버퍼되면 지정된 포트에서 송신될 때까지 기다리지 않고 프로그램 실행이 계속됩니다. 단, 같은 포트를 사용하는 문자열이 이전에 작성되어 송신되지 않고 버퍼에 남아있는 경우에 새로운 문자열이 버퍼에 저장될 수 있을 때까지 기다린 다음 프로그램을 실행합니다.

송신하는 문자열은 표준 ASCII 문자 '\'(백슬래시)로 시작하여 컨트롤 및 특수 문자의 출력을 허가하는 이스케이프 시퀀스, '%' (퍼센트) 문자로 시작하여 식의 수치를 문자열로 포맷하는 방법을 지정한 포맷 시퀀스의 조합입니다. 각 포맷 시퀀스에는 문자열 뒤에 일치하는 식이 필요합니다.

다음 이스케이프 시퀀스를 사용할 수 있습니다(전부 소문자).

- \a 경고음(ASCII 7)
- \b 백스페이스(ASCII 8)
- \t 가로 탭(ASCII 9)
- \n 줄 바꿈(ASCII 10)
- \v 세로 탭(ASCII 11)
- \f 폼 피드(ASCII 12)
- \r 캐리지 리턴(ASCII 13)
- \\ 백슬래시 문자
- \? 물음표
- \' 작은따옴표
- \" 큰따옴표
- \ooo ASCII 문자 코드의 8 진법(000~377 범위)
- \xhh ASCII 문자 코드의 16 진법(x00~xff 범위)

다음과 같은 포맷 시퀀스를 사용할 수 있습니다(대·소문자 여부 본문 기재).

- %d 부호가 있는 정수, 10 진법 형식
- %u 부호가 없는 정수, 10 진법 형식
- %x 부호가 없는 정수, 16 진법 형식, 소문자 사용
- %nx 부호가 없는 정수, 16 진법 형식, n 자리 사용($n = 1\sim 8$), 소문자 사용
- %X 부호가 없는 정수, 16 진법 형식, 대문자 사용
- %nX 부호가 없는 정수, 16 진법 형식, n 자리 사용($n = 1\sim 8$), 대문자 사용
- %f 부동 소수점 값, 최대 6 자리까지
- %nf 부동 소수점 값, 최대 n 자리까지($n = 1\sim 31$)
- %n.mf 부동 소수점 값, 최대 n 자리까지($n = 1\sim 31$), 소수 최대 m 자리($m < n$)
- %s 문자열, 임의의 길이(Null 종단됨)
- %ns 문자열, 최대 n 문자(Null 종단 문자가 있는 경우에는 이보다 짧음)
- %c 지정된 바이트값의 1 문자
- %nc 지정된 바이트값의 1 문자의 n 회 반복
- %% 퍼센트 문자

선행 제로열은 10 진법 정수로는 작성되지 않습니다. 자릿수가 지정된 경우에만 선행 제로열이 16 진법 정수로 작성됩니다. 자릿수 지정 여부와 관계없이 선행 및 끝 제로열은 부동 소수점 값으로 작성되지 않습니다. 부동 소수점 값에 지정된 자릿수가 지수 없이 값을 나타내기에 불충분한 경우에 자동으로 지수가 붙습니다(2.345e8 등). 0 이 아닌 소수 부분이 있는 경우에 소수점은 부동 소수점 값만 사용됩니다. 부동 소수점 값의 소수점, 음의 부호, 지수는 자릿수로 세지 않습니다.

문자열이 포맷된 각 변수 시퀀스에는 문자에 변환되는 수치를 지정하는 **{식}**이 포함됩니다. 식은 단순히 하나의 상수 또는 변수인 경우도 있지만 여러 개의 수학 연산자나 식을 포함하는 경우도 있습니다. 문자열 변수에는 식의 값(필요에 따라 다음 정수로 버림)이 사용되는 문자열 변수의 사용자 공유 메모리 내에서의 시작 인덱스를 나타냅니다.

저장 설정 요소 [Sys.SendFileMode]의 비트 *n* 이 1 로 설정된 경우에 포트 *n* 에 송신되는 문자열 선두에 '-n:' 문자가 자동으로 추가됩니다. 또한 문자열 마지막에 Null 문자(ASCII 0)가 자동으로 추가됩니다.

Telnet 통신 세션에서는 일반적인 구문 `send getsends -n [-n2...] [-b] [-fname](n, n2` 등은 잠재적인 임의의 통신을 위해 유효로 하는 소프트웨어 통신 포트의 번호)에 따라 Power PMAC 의 'send getsends' 유틸리티가 시작됩니다. 옵션의 '-b'는 바이너리 모드를 탑재합니다(바이너리 모드가 없으면 Power PMAC 은 문자열을 중단시키지 않고는 Null 문자를 송신할 수 없으며 Power PMAC 은 문자열 선두에 포트 번호를 자동으로 추가합니다). 또한 옵션인 *fname* 은 파일명을 지정하고 문자열을 파일에 입력할 수 있도록 합니다.

저장 설정 요소 [Sys.SendFileMode]의 비트 *n* 이 1 로 설정된 경우에 포트 *n* 에 송신되는 각 문자열 선두에 '-n:' 문자가 추가됩니다. 또한 문자열 마지막에 Null 문자가 추가됩니다.

예

```
send0, "Hello, World!";           // Pure literal string to Port 0
send1, "Hello, World!\n";         // String with "new line" control character at end
send2, "Cycle counter=%u\n", P100; // String with unsigned integer value
send3, "Xpos:%9fmm Ypos:%9fmm\n", Q107*25.4, Q108*25.4;
                                   // String with 2 floating-point expressions
send4, "App error #%u, Code %s\n", P20, 1024;
                                   // Text with integer and string variables
```

sendall

기능: 버퍼된 송신 문자열을 모두 송신함

구문: **sendall**

sendall 명령에 따라 Power PMAC 은 모든 통신 포트에 대해 이전에 **send** 명령으로 송신된 보류 중인 문자열을 모두 강제 출력합니다. 보류 중인 문자열이 모두 송신될 때까지 프로그램 실행은 중지됩니다. 이 명령은 이전에 작성된 문자열을 버퍼에서 확실히 플러시할 경우에 유용합니다.

sendallcmds

기능: 버퍼된 'cmd' 문자열을 모두 송신함

구문: **sendallcmds**

Power PMAC 은 **sendallcmds** 명령으로 이전의 모든 프로그램에서 **cmd** 명령에 따라 송신된 보류 중인 문자열을 모두 강제 실행합니다. 보류 중인 문자열이 모두 실행될 때까지 프로그램 실행은 중지됩니다. 이 명령은 이전에 작성된 문자열을 버퍼에서 확실히 플래시할 경우에 유용합니다.

sendallsystemcmds

기능: 버퍼된 'system' 문자열을 모두 송신함

구문: **sendallsystemcmds**

Power PMAC 은 **sendallsystemcmds** 명령으로 이전의 모든 프로그램에서 **system** 명령에 따라 송신된 보류 중인 문자열을 모두 강제 실행합니다. 보류 중인 문자열이 모두 실행될 때까지 프로그램 실행은 중지됩니다. 이 명령은 이전에 작성된 문자열을 버퍼에서 확실히 플래시할 경우에 유용합니다.

spline{데이터}

기능: 3 차 B 스플라인 동작 모드 및 시간을 설정함

구문: **spline{데이터 0}[spline{데이터 1}[spline{데이터 2}]]**

여기에서

- **{데이터 n}**은 섹션마다 동작 시간을 밀리초로 나타내는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

spline 명령으로 모션 프로그램은 3 차 B 스플라인 동작 모드가 되고 스플라인 동작 시간을 밀리초 단위로 설정합니다. 모션 프로그램이 이미 3 차 B 스플라인 동작 모드인 경우에 명령을 사용하여 동작 시간을 변경할 수 있습니다(정지하지 않습니다). 모션 프로그램의 후속 동작 명령은 이 모드의 규칙에 따라 처리됩니다.

3 차 B 스플라인 동작 모드에서는 프로그램 동작은 3 개의 섹션으로 구성됩니다. 스플라인 모드 시퀀스 도중에 프로그램 동작 섹션은 전후의 프로그램 동작 섹션과 중복됩니다. 그 결과 3 개의 동작을 중첩시킨 프로파일을 얻을 수 있습니다.

최초의 섹션에 걸리는 시간은 Status 데이터 구조체 요소 **[Coord[x].T0Spline]**으로 유지됩니다. 두 번째 섹션에 필요한 시간은 **[Coord[x].T1Spline]**으로 유지됩니다. 세 번째 섹션에 필요한 시간은 **[Coord[x].T2Spline]**으로 유지됩니다.

spline 모드 프로그램 동작에서는 최초의 섹션만 완전히 계산됩니다(앞의 동작이 있는 경우에 그 뒤의 섹션에 중첩됩니다). 두 번째와 세 번째 섹션은 가계산만 실행됩니다. 후속 **spline** 명령 및 프로그램 동작 명령에 따라 이러한 섹션에 필요한 시간이 바뀌는 경우가 있기 때문입니다.

spline 명령의 정확한 동작은 프로그램이 이미 스플라인 모드 시퀀스를 실행 중인지 여부 또는 저장 설정 요소 [Coord[x].SplineTimeRotate] 설정에 따라 달라집니다.

시퀀스 시작 시에 하나의 시간값을 지정하면(**spline50** 등) 3 개의 섹션 전체에 이 지정값이 설정됩니다. 2 개의 시간값을 지정하면(**spline50 spline75** 등) 최초의 섹션(T0Spline)에는 첫 번째 값이 지정되고 두 번째와 세 번째 섹션([T1Spline] 및 [T2Spline])은 두 번째 값이 설정됩니다. 3 개의 시간값을 지정하면(**spline50 spline75 spline100** 등) 동작에 해당하는 섹션에 각 값이 설정됩니다.

[Coord[x].SplineTimeRotate]가 기본값인 0 으로 설정되고 시퀀스가 실행 중인 경우에 하나의 시간값을 지정하면 세 번째 섹션(T2Spline) 시간에 이 지정값이 설정됩니다. 최초의 두 번째 섹션([T0Spline] 및 [T1Spline])에 걸리는 시간은 이전 값 그대로입니다.

[Coord[x].SplineTimeRotate]를 1 로 설정하면 Power PMAC 은 스플라인 섹션 시간에 대해 이전 PMAC 과 호환 모드가 됩니다. 이 경우에 **spline{데이터}** 명령은 다음 동작의 세 번째 섹션(T2Spline) 시간만 변경합니다(동일한 행에 여러 시간을 지정하면 각 값이 하나 앞의 값을 덮어쓰기 때문에 마지막으로 지정된 값만 설정됩니다).

단, 이 명령 다음의 두 번째 동작에서는 이 새로운 값이 두 번째 섹션(T1Spline) 시간값으로 사용됩니다. 세 번째 동작에서는 이 새로운 값이 최초의 섹션(T0Spline) 시간값으로 사용됩니다. 이처럼 스플라인 시간은 3 개 동작에서 변경되고 섹션 시간은 재계산되지 않습니다.

Turbo PMAC 은 스플라인 섹션 시간을 **ta** 또는 **tm** 명령으로 설정 또는 변경할 수 있지만 Power PMAC 에서는 불가능합니다.

spline 명령을 실행하면 그 외의 동작 모드(**circle**, **linear**, **rapid**, **pvt**)가 종료됩니다. 그 외의 동작 모드를 명령하면 **spline** 모드가 종료됩니다.

예

```
spline 200;           // Set spline mode with all 3 section times of 200 msec
spline (L10);         // Set spline mode with all 3 section times of (L10) msec
spline 200 spline 300; // Set spline mode with 1st section time of 200 msec,
                      // 2nd and 3rd section times of 300 msec
spline 200 spline 300 spline 400; // Set spline mode with 1st section time
                                // of 200 msec, 2nd section time of 300 msec;
                                // 3rd section time of 400 ms
```

start

기능: 지정된 모션 프로그램의 프로그램 카운터를 포인트하여 실행함

구문: **start[{목록}]:{데이터}**

여기에서

- **{목록}**은 씬표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.
- **{데이터}**는 모션 프로그램 번호(정수 부분) 및 옵션에서 행 점프 레이블 번호(소수 부분을 1,000,000 배)를 지정하는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

start 명령에 따라 대상 또는 목록에 있는 좌표계에서는 지정된 모션 프로그램의 선두 또는 지정된 행 점프의 레이블 번호로 프로그램 카운터가 설정됩니다. 또한 그 시점부터 프로그램 연속 실행을 시작합니다.

좌표계 목록이 없는 경우에 현재 프로그램에 따라 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 목록 내의 좌표계를 지정해도 프로그램에 따라 모달에 주소 지정된 좌표계에는 영향을 미치지 않습니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 `[Plc[i].Ldata.Coord]`에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(`Coord[x].Ldata.Coord`)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

{데이터}의 정수부(버림 등에 따른)는 모션 프로그램 번호를 지정합니다. 소수 부분이 없는 경우에 모션 프로그램의 선두를 가리키듯 프로그램 카운터가 설정됩니다. 소수 부분이 있는 경우에 소수 부분에 1,000,000을 곱하면 프로그램 카운터가 가리키는 행 점프 레이블 번호가 됩니다.

좌표계의 이 명령이 Power PMAC을 통해 수용되려면 좌표계가 적절한 상태여야 합니다. 적절한 상태가 아닌 경우에 Power PMAC은 이 명령을 에러로 거부합니다. 이 명령이 수용되려면 좌표계가 다음 조건을 유지해야 합니다.

- 선택된 모션 프로그램(및 선택된 레이블)이 존재하고 유효할 것
- 좌표계의 위치 결정축에 할당된 모든 모터가 기동 중일 것(`[Motor[x].ServoCtrl] > 0`)
- 좌표계의 위치 결정축에 할당된 모든 모터가 유효하며 클로즈드 루프일 것
- 좌표계의 위치 결정축에 할당된 모터에 양방향 오버트래블 리미트가 설정되어 있지 않을 것
- Power PMAC에 의한 현재 모터 위치로부터의 모든 시작축 위치 계산이 가능할 것(`[Coord[x].Csolve] = 1`)

- [Coord[x].HomeRequired] = 1 인 경우에 좌표계의 위치 결정축에 할당된 모든 모터가 정상적인 원점 서치 동작 또는 절대 위치 읽기에 따라 위치 참조를 확립했을 것
- 프로그램된 포인트가 아닌 일시 정지 위치에서 재기동하는 경우에 모든 모터 위치가 최초의 정지 및 일시 정지가 명령된 위치와 같을 것

예

```
start:2;           // Point addressed C.S. to top of prog 2 and run
start1:75;         // Point C.S.1 to top of prog 75 and run
start3:75.001;     // Point C.S.3 to jump label N1000: of prog 75 and run
start8:(P1+P2/1000000); // Point C.S.8 to jump label specified by P2 in motion
                    // program specified by P1, and run
```

step

기능: 지정한 좌표계에서 모션 프로그램 싱글 스텝을 실행함

구문: **step** [{목록}]

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수 세트 및 시작 상수 번호에서 종료 상수 번호까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.

Power PMAC은 **step** 명령으로 지정된 좌표계의 선택된 모션 프로그램 싱글 스텝 실행을 프로그램 카운터의 현재 포인트에서 시작합니다.

이 명령에 응답하는 좌표계의 정확한 동작은 저장 설정 요소 [Coord[x].StepMode]의 설정에 따라 달라집니다(이 요소는 2014년 1/4 분기에 출시된 V1.6 펌웨어의 신기능이며 동작은 항상 이 요소가 0으로 설정된 상태와 같습니다).

좌표계가 이미 모션 프로그램을 실행 중인 경우에 이 명령이 전송되면 프로그램은 싱글 스텝 모드가 됩니다. [Coord[x].StepMode]가 0으로 설정된 경우에 또 다른 한 동작이 계산되어 동작 실행은 이 동작이 끝날 때 정지합니다.. 이 경우에 **q** 명령 처리와 비슷하지만 **q**와 다른 점은 추가적인 동작이 실행된다는 점입니다. [Coord[x].StepMode]가 0보다 큰 값으로 설정된 경우에 또 다른 한 동작이 계산되어 동작 실행은 이 직전에 계산된 동작이 끝날 때 정지합니다.

프로그램 실행이 이미 싱글 스텝 모드에서 정지된 경우에 [Coord[x].StepMode]가 0으로 설정되어 있으면 스텝 명령에 따라 하나의 동작이 완료될 때까지 프로그램은 실행을 계속합니다. [Coord[x].StepMode]가 0보다 큰 값으로 설정된 경우에 하나의 프로그램행이 실행됩니다([Coord[x].StepMode]가 0 이외의 값으로 설정되고 2D 공구 반경 보정이 유효한 경우에는 값에 따라 동작이 조금 다릅니다).

[Coord[x].StepMode] 설정과 관계없이 싱글 스텝 실행 모션 프로그램에서 **bstart** 명령이 검출되면 포함된 동작이나 프로그램행의 수에 관계없이 모션 프로그램은 다음 **bstop** 명령까지 동작을 속행하고 정지합니다.

좌표계 목록이 없는 경우에 현재 프로그램에서 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 목록 내의 좌표계를 지정해도 프로그램에 따라 모달에 주소 지정된 좌표계에는 영향을 미치지 않습니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 `[Plc[i].Ldata.Coord]`에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={*x*}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(`Coord[x].Ldata.Coord`)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

일반적으로 좌표계의 모션 프로그램은 최초에 **begin** 명령으로 선택됩니다. **step**, **pause** 또는 **hold** 명령 등으로 프로그램 실행이 정지된 경우에 **step** 명령으로 정지된 포인트에서 실행을 시작합니다.

스텝이 완료되면 모션 프로그램 실행은 일시 정지되지만 논리적으로는 아직 실행 중인 상태입니다. 일시 정지 후에도 프로그램 실행을 재개하지 않는 경우에 **Abort** 명령 또는 정지 명령을 사용하여 프로그램 실행을 완전히 종료해야 합니다.

좌표계의 이 명령이 Power PMAC 을 통해 수용되려면 좌표계가 적절한 상태여야 합니다. 적절한 상태가 아닌 경우에 Power PMAC 은 이 명령을 에러로 거부합니다. 이 명령이 수용되려면 좌표계가 다음 조건을 유지해야 합니다.

- 선택된 모션 프로그램(및 선택된 레이블)이 존재하고 유효할 것
- 좌표계의 위치 결정축에 할당된 모든 모터가 기동 중일 것(`[Motor[x].ServoCtrl] > 0`)
- 좌표계의 위치 결정축에 할당된 모든 모터가 유효하며 클로즈드 루프일 것
- 좌표계의 위치 결정축에 할당된 모터에 양방향 오버트래블 리미트가 설정되어 있지 않을 것
- Power PMAC 에 의한 현재 모터 위치로부터의 모든 시작축 위치 계산이 가능할 것(`[Coord[x].Csolve] = 1`)
- `[Coord[x].HomeRequired] = 1` 인 경우에 좌표계의 위치 결정축에 할당된 모든 모터가 정상적인 원점 서치 동작 또는 절대 위치 읽기에 따라 위치 참조를 확립했을 것
- 프로그램된 포인트가 아닌 일시 정지 위치에서 재기동하는 경우에 모든 모터 위치가 최초의 정지 및 일시 정지가 명령된 위치와 같을 것

상당하는 온라인 명령은 생략형 **s** 를 사용하며 그 앞에 좌표계 목록이 놓인다는 점(**&1,2s** 등)에 주의하십시오.

예

```
step; // Single-step motion program in addressed C.S.  
step 2; // Single-step motion program in C.S.2  
step 1..16; // Single-step motion program in C.S.1 thru 16  
step 3..6,8; // Single-step motion program in C.S.3, 4, 5, 6, 8
```

step plc

기능: 지정된 PLC 프로그램의 싱글 스텝을 실행함

구문: **step plc {목록}**

여기에서

- **{목록}**은 유효로 할 PLC 프로그램을 지정하는 하나 이상의 상수 세트 또는 0~31의 범위 상수입니다.

step plc 명령에 따라 Power PMAC 은 지정된 PLC 프로그램을 보통 실행 순서로 프로그램 싱글 스텝을 실행합니다. 지정된 각 PLC 프로그램은 일시 정지한 시점부터 실행을 시작합니다. PLC 프로그램은 번호(0~31)에 따라 지정됩니다. 이 명령에서는 하나의 번호만 지정된 경우에 여러 번호가 쉼표로 구분되어 지정되는 경우와 연속된 번호의 변수 범위가 지정되는 경우가 있습니다.

예

```
step plc 0;  
step plc 1,2,5;  
step plc 1..16;  
step plc 3..6,8;
```

stop

기능: 지정한 좌표계에서 모션 프로그램 실행을 정지함

구문: **stop [{목록}]**

여기에서

- **{목록}**은 쉼표로 구분된 정수 상수 및 시작 상수에서 종료 상수까지의 사이를 2개의 마침표로 나타낸 연속된 정수 상수 범위입니다. 상수 범위는 0에서 특정 버전의 Power PMAC 펌웨어가 지원하는 좌표계 수에서 1을 뺀 값까지입니다.

Power PMAC 은 **stop** 명령으로 지정된 좌표계의 모션 프로그램 실행을 마지막에 계산된 동작의 실행 후에 정지시킵니다. 그 후에 프로그램 카운터는 자동으로 모션 프로그램 선두로 복귀합니다. 따라서 실행을 정지한 포인트에서 재개할 수 없습니다.

좌표계 목록이 없는 경우에 현재 프로그램에 따라 주소 지정된 좌표계가 대상이 됩니다. 좌표계 목록이 있는 경우에 앞에서 설명한 동작은 목록 내의 모든 좌표계가 대상이 됩니다. 목록 내의 좌표계를 지정해도 프로그램에 따라 모달에 주소 지정된 좌표계에는 영향을 미치지 않습니다.

이 명령을 전송하는 최상위 프로그램이 PLC 프로그램인 경우에 PLC 프로그램의 데이터 구조체 내의 요소 [Plc[i].Ldata.Coord]에 따라 주소 지정되는 좌표계가 결정됩니다. PLC 프로그램에서는 명령 **Ldata.Coord={식}**으로 좌표계를 모달에 주소 지정할 수 있습니다. 부모 구조체명은 필요 없습니다.

이 명령을 전송하는 최상위 프로그램이 모션 프로그램인 경우에 프로그램을 실행하는 좌표계에서 모달에 주소 지정되는 좌표계(Coord[x].Ldata.Coord)를 변경할 수 없습니다. 따라서 모션 프로그램이 이 명령을 다른 좌표계에 전송하는 경우에는 명령 목록에 그 좌표계를 포함해야 합니다.

예

```
stop;                                // Halt motion program in addressed C.S.
stop 2;                             // Halt motion program in C.S.2
stop 1..16;                          // Halt motion program in C.S.1 thru 16
stop 3..6,8;                         // Halt motion program in C.S.3, 4, 5, 6, 8
```

switch({식})

기능: 조건부 다중 분기

구문: **switch({식})[상수]**

여기에서

- **{식}**은 실행할 분기를 제어하는 수식입니다.
- **[상수]**는 이 전환 명령으로 사용할 케이스의 최대수를 나타내는 옵션의 양의 정수 상수입니다. 값이 지정되지 않은 경우에는 256 을 값으로 사용합니다.

```
.....switch({식})
.....{
..... case{상수}:{명령}[명령...][break}
..... [case{상수}:{명령}[명령...][break]}
..... [default:{명령}[명령...]]
..... }
```

switch 명령으로 프로그램에서 여러 개의 케이스 분기를 사용할 수 있습니다. 명령 실행 시에 괄호 내의 식을 평가합니다. 식이 엄밀하게 정수값으로 평가되지 않는 경우에는 가까운 정수로 버립니다.

식 뒤에 있는 옵션 상수값은 이 명령으로 사용할 케이스의 최대수를 지정합니다. 최대값이 작을 수록(실제로 사용하는 케이스 수와 관계없이) 실행 속도가 향상됩니다. 값이 지정되지 않은 경우에는 Power PMAC 은 256 을 값으로 사용합니다. IDE 프로젝트 매니저는 자동으로 사용 케이스 수에 대응하는 최소 수를 지정합니다.

이 명령 직후의 문자는 동일행이거나 다음 프로그램행의 선두이거나 관계없이 가능성이 있는 분기의 시작을 나타내는 여는 중괄호(**{**)여야 합니다. 이 여는 괄호 뒤에는 하나 이상의 **case** 명령이 있습니다. 각 **case** 명령 뒤에는 정수 상수 및 콜론이 있습니다. 콜론 뒤에는 **switch** 명령 식의 값이 **case** 명령 상수값과 일치하는 경우에 실행할 명령 세트가 있습니다. 명령 실행은 **break** 명령이 설명된 행까지 또는 종료를 나타내는 닫는 중괄호가 설명된 곳까지 계속합니다.

여는 괄호 뒤에 분기 세트의 마지막에 **default** 명령이 있는 경우가 있습니다. 이 명령 뒤에는 콜론 및 **switch** 명령 식의 값이 **case** 명령 세트의 상수값과 일치하지 않는 경우에 실행할 명령 세트가 있습니다. 식의 값이 일치하지 않고 **default** 명령도 없는 경우에 **switch** 명령에 따라 아무것도 처리되지 않습니다.

default 케이스를 포함하는 **switch** 명령에서 선택할 수 있는 가능성이 있는 케이스 세트에서는 다음 프로그램행의 선두 전에 닫는 중괄호(**}**)가 필요합니다.

예/

```
switch (P1) {                                // Branch based on value of P1
  case 1:X10 F5; break;                      // Action if P1 = 1
  case 2:X17 F7; break;                      // Action if P1 = 2
  case 3:X-3 F2; break;                      // Action if P1 = 3
  default:dwll 100;                          // Action if P1 != 1, 2, or 3
}

switch (P0) {                                // Branch based on value of P0
  case 4:P4++;                               // Action if P0 = 4
  case 3:P3++;                               // Action if P0 = 3 or 4
  case 2:P2++;                               // Action if P0 = 2, 3, or 4
  case 1:P1++;                               // Action if P0 = 1, 2, 3, or 4
}
```

system

기능: 운영체제의 명령 파서에 명령을 전송합니다.

구문: **system[,]"{문자열}"[,]{식}...]**

여기에서

- **{문자열}**은 리터럴 영숫자, 이스케이프 시퀀스 및 포맷된 변수 시퀀스로 구성되는 포맷된 ASCII 문자열입니다.
- **{식}**은 문자열과 일치하는 포맷된 변수 시퀀스로 송신되는 값을 포함하는 수식입니다. 포맷된 각 변수 시퀀스마다 하나의 수식이 필요합니다.

system 프로그램 명령에 따라 Telnet 세션의 외부 컴퓨터로부터 전송된 것처럼 Power PMAC 이 Power PMAC 의 운영체제로 보내는 명령으로 포맷된 문자열을 전송합니다.

프로그램 실행의 일부로서 명령이 처리되면 문자열이 작성되어 운영체제에 따라 처리되도록 버퍼됩니다. 문자열이 버퍼되면 문자열이 스레드에서 처리될 때까지 기다리지 않고 프로그램 실행을 계속합니다.

송신하는 문자열은 표준 ASCII 문자 '\'(백슬래시)로 시작하여 컨트롤 및 특수 문자 출력을 허가하는 이스케이프 시퀀스, '%' (퍼센트) 문자로 시작하여 식의 수치를 문자열로 포맷하는 방법을 지정하는 포맷 시퀀스의 조합입니다. 각 포맷 시퀀스에는 문자열 뒤에 일치하는 식이 필요합니다.

다음 이스케이프 시퀀스를 사용할 수 있습니다(전부 소문자).

- \a 경고음(ASCII 7)
- \b 백스페이스(ASCII 8)
- \t 가로 탭(ASCII 9)
- \n 줄 바꿈(ASCII 10)
- \v 세로 탭(ASCII 11)
- \f 폼 피드(ASCII 12)
- \r 캐리지 리턴(ASCII 13)
- \\ 백슬래시 문자
- \? 물음표
- \' 작은따옴표
- \" 큰따옴표
- \ooo ASCII 문자 코드의 8 진법(000~377 범위)
- \xhh ASCII 문자 코드의 16 진법(x00~xff 범위)

다음과 같은 포맷 시퀀스를 사용할 수 있습니다(대·소문자 여부 본문 기재).

- %d 부호가 있는 정수, 10 진법 형식
- %u 부호가 없는 정수, 10 진법 형식
- %x 부호가 없는 정수, 16 진법 형식, 소문자 사용
- %nx 부호가 없는 정수, 16 진법 형식, n 자리 사용($n = 1\sim 8$), 소문자 사용
- %X 부호가 없는 정수, 16 진법 형식, 대문자 사용
- %nX 부호가 없는 정수, 16 진법 형식, n 자리 사용($n = 1\sim 8$), 대문자 사용
- %f 부동 소수점 값, 최대 6 자리까지
- %nf 부동 소수점 값, 최대 n 자리까지($n = 1\sim 31$)
- %n.mf 부동 소수점 값, 최대 n 자리까지($n = 1\sim 31$), 소수 최대 m 자리($m < n$)
- %s 문자열, 임의의 길이(Null 종단됨)
- %ns 문자열, 최대 n 문자(Null 종단 문자가 있는 경우에는 이보다 짧음)
- %c 지정된 바이트값의 1 문자
- %nc 지정된 바이트값의 1 문자의 n 회 반복
- %% 퍼센트 문자

선행 제로열은 10 진법 정수로는 작성되지 않습니다. 자릿수가 지정된 경우에만 선행 제로열이 16 진법 정수로 작성됩니다. 자릿수 지정 여부와 관계없이 선행 및 끝 제로열은 부동 소수점 값으로 작성되지 않습니다. 부동 소수점 값에 지정된 자릿수가 지수 없이 값을 나타내기에 불충분한 경우에 자동으로 지수가 붙습니다(2.345e8 등). 0 이 아닌 소수 부분이 있는 경우에 소수점은 부동 소수점 값만 사용됩니다. 부동 소수점 값의 소수점, 음의 부호, 지수는 자릿수로 세지 않습니다.

문자열이 포맷된 각 변수 시퀀스에는 문자에 변환되는 수치를 지정하는 **{식}**이 포함됩니다. 식은 단순히 하나의 상수 또는 변수인 경우도 있지만 여러 개의 수학 연산자나 식을 포함하는 경우도 있습니다. 문자열 변수에는 식의 값(필요에 따라 다음 정수로 버림)이 사용되는 문자열 변수의 사용자 공유 메모리 내에서의 시작 인덱스를 나타냅니다.

명령에 대해 응답이 없는 경우에도 처리된다는 점에 주의하십시오. 응답이 있는 경우에 명령은 사라지지만 에러는 발생하지 않습니다.

좌표계의 로컬 데이터 구조체(모션 프로그램 내인 경우) 또는 PLC 프로그램에서는 Power PMAC 스크립트 명령이 정상적으로 실행되면 요소 [Ldata.SystemCmdStatus]가 0으로 설정됩니다. 명령 실행 시에 에러가 발생하면 음의 번호로 설정됩니다. 명령 실행을 모니터링하는 경우에 명령을 실행하기 전에 이 요소를 양의 번호로 설정하십시오.

또한 로컬 데이터 구조체에서는 명령을 실행하면 명령의 정상적 실행 여부와 관계없이 요소 [Ldata.SystemCmdCount]는 증가합니다. 명령 실행 시에 오류가 발생해도 아무 처리도 하지 않는다는 점에 주의하십시오. 카운트를 확인하기 전에 명령 버퍼를 확실히 플러시하려면 **sendallsystemcmds** 명령을 사용하십시오. 사용자는 [Ldata.SystemCmdCount]에 값을 설정할 수 있습니다. 일부 사용자는 명령 세트를 전송하기 전에 0으로 설정하고 기대되는 수의 명령이 실행되는지 모니터링합니다.

명령 처리 시에 에러가 발생한 경우에 명령이 다음에 정상적으로 실행될 때까지(또는 사용자가 음이 아닌 값으로 덮어쓸 때까지) [Ldata.SystemCmdStatus]는 음의 값을 기다립니다.

예

```
system"/opt/ppmac/tune/parabolicmove 1 2000 500 0 0 0"; // Execute
provided "parabolic move" tuning C program

Ldata.SystemCmdStatus=1; // Set to start
system"/var/ftp/usrflash/Project/C\ Language/Background\ Programs/cplc1.out"; //
Execute user C application program
while (Ldata.SystemCmdStatus > 0) {}; // Wait until done
```

T{데이터}

기능: 공구 선택 코드(T 코드)

구문: **T{데이터}**

여기에서

- **{데이터}**는 코드 번호 및 그 코드를 탑재하는 서브루틴의 행 점프 레이블을 나타내는 0.000~999.999 의 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

Power PMAC 은 T 코드를 **call{Coord[x].Tprog}.{데이타}*1000** 명령(**{데이타}**는 T 코드의 번호)으로 해석합니다. 이 명령에 따라 [Coord[x].Tprog]의 값에 따라 지정된 번호의 서브 프로그램으로 점프하고 코드 번호를 1,000 배한 것이 행 점프 레이블이 됩니다. 예를 들어 [Coord[x].Tprog]의 기본값이 1002 인 경우에 **T03** 은 **subprog 1002** 의 행 점프 레이블 **N3000**:을 호출합니다. 프로그램 실행은 다음 **return** 명령에서 호출자 프로그램으로 복귀합니다.

이 구조체에 따라 커스터마이징 가능한 T 코드 루틴을 CAD/CAM 프로그램의 RS-274 대응 코드 및 CNC 스타일의 어플리케이션에 탑재할 수 있습니다. 호출자 프로그램 T 코드의 뒤에 1 세트 이상의 **{문자}{데이타}**를 두면 이러한 서브루틴에 인수를 전달할 수 있습니다. 각 문자의 뒤의 값은 **read** 명령을 사용하여 서브루틴으로 취득할 수 있습니다. 모든 좌표계의 [Coord[x].Tprog] 기본값은 1002 입니다. 따라서 모든 좌표계는 기본값으로 같은 T 코드 세트를 공유합니다. 단, 다른 좌표계에 다른 값을 할당하면 다른 코드를 탑재할 수 있습니다.

일반적으로 T 코드는 RS-274 프로그램에서 공구 데이터 코드로 사용되며 자동 공구 전환 장치 등을 제어합니다. T 코드의 코어 세트는 0~99 의 정수를 사용하지만 이 스킴에서는 999.999 까지의 소수 코드 번호를 사용할 수 있고 코어 번호에 수식을 사용할 수 있습니다.

모든 종류의 서브루틴 및 서브 프로그램 호출(**call**, **callsub**, **gosub**, **G**, **M**, **T**, **D**)은 255 레벨(최상위를 포함하여 256)로 네스트할 수 있습니다. 무한 루프는 사용하지 마십시오.

T 코드 명령에 따라 지정된 점프 행 레이블이 프로그램에 존재하지 않는 경우에 지정된 T 코드의 서브 프로그램 선두로 점프합니다. 에러는 발생하지 않습니다. 이를 이용하여 사용자 어플리케이션이 존재하지 않는 T 코드를 처리하는 방법을 결정할 수 있습니다.

ta{데이타}

기능: 가속/감속 시간 지정

구문: **ta{데이타}**

여기에서

- **{데이타}**는 가속/감속 시간을 밀리초로 나타내는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

ta 명령은 직선 보간 모드 동작 및 원호 보간 모드 동작의 명령 가속도 및 감속도 시간을 부동 소수점 분해능을 갖는 밀리초 단위로 지정합니다. 제로 또는 양의 값을 지정해야 합니다. 가속 시간에 제로를 설정해도 거부되지 않고 런타임 에러도 발생하지 않지만 명령 속도 프로파일이 급격하게 변화합니다(실제 가속 제어는 타임 베이스값의 기울기에 따라 제어되듯이 외부 타임 베이스의 어플리케이션에서는 정당한 사용 방법이 되는 경우가 있습니다).

td(감속 시간) 명령으로 직선 보간 동작 모드와 원호 보간 동작 모드에 각각의 최종 감속 시간을 설정할 수도 있습니다. 단, 이것을 실현하려면 **td** 명령 앞에 **ta** 명령을 두어야 합니다. **ta** 명령은 가속 시간과 감속 시간 양쪽을 설정하기 위한 것입니다.

모션 프로그램에서 **ta** 명령이 실행되면 프로그램을 실행하는 좌표계 데이터 구조체 요소 [Coord[x].Ta] 및 [Coord[x].Td]는 $\{데/이/타\}$ 에서 구해진 값으로 설정됩니다. 모션 프로그램에서 **ta** 명령이 실행되지 않는 경우에 데이터 구조체 요소 [Coord[x].Ta] 및 [Coord[x].Td]의 값이 직접 사용됩니다.

ta 명령은 PLC 프로그램에서는 'no op'가 됩니다. 직접 PLC 프로그램에서 좌표계에 가속 시간값을 설정하는 경우에 [Coord[x].Ta] 및 [Coord[x].Td](필요한 경우)에 직접 쓰십시오.

좌표계 모터의 가속도가 데이터 구조체 요소 [Motor[x].InvAmax]에 따라 설정된 제어를 초과한 경우에 리미트 위반을 회피하기 위해 충분한 만큼 가속 시간이 연장됩니다. 좌표계가 세그먼테이션 모드가 아닌 경우([Coord[x].SegMoveTime] = 0) 또는 세그먼테이션 모드에서 록 어헤드 버퍼가 유효한 경우([Coord[x].SegMoveTime] > 0)에 자동 제어가 기능합니다.

지정된 S 곡선 가속 시간(Coord[x].Ts)이 가속 또는 감속 시간의 반보다 큰 경우에 전체 가속 또는 감속 시간은 지정된 S 곡선 시간의 2 배가 됩니다.

tm 명령으로 직접 지정했지만 벡터 거리를 벡터 피드 레이트(**F**)로 나누어 구한 동작 시간이 가속 시간보다 작은 경우에 가속 시간과 일치하도록 동작 시간이 증가합니다. 가속 시간은 최소 허용 동작 시간으로 기능함을 의미합니다.

예

```
ta100;
ta(P20);
ta(45.3+sqrt(Q10));
```

td{데/이/타}

기능: 감속 시간 지정

구문: **td{데/이/타}**

여기에서

- $\{데/이/타\}$ 는 감속 시간을 밀리초로 나타내는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

td 명령은 직선 보간 모드 동작 및 원호 보간 모드 동작의 명령 감속도 시간을 부동 소수점 분해능을 갖는 밀리초 단위로 지정합니다. 제로 또는 양의 값을 지정해야 합니다. 감속 시간에 제로를 지정해도 거부되지 않고 런타임 에러도 발생하지 않습니다. 그러나 명령 속도 프로파일에 급격한 변화가 발생합니다(실제 감속 제어는 타임 베이스값 기울기에 따라 제어되듯이 외부 타임 베이스 어플리케이션에서는 정당한 사용 방법인 경우가 있습니다).

td 명령을 이용하여 직선 보간 모드 동작 및 원호 보간 모드 동작의 최종 감속 시간 또는 초기 가속 및 블렌드 시간과 다른 동작 시퀀스를 지정할 수 있습니다. **ta** 명령은 가속 및 감속 시간 모두 설정한다는 점에 주의하십시오. 감속 시간에 다른 값을 지정할 경우에 **ta** 명령 뒤에 **td** 명령이 필요합니다.

모션 프로그램에서 **td** 명령이 실행되면 프로그램을 실행하는 좌표계 데이터 구조체 요소 [Coord[x].Td]는 {데이터}에서 구해진 값으로 설정됩니다. 모션 프로그램에서 **td** 명령이 실행되지 않으면 데이터 구조체 요소 [Coord[x].Td]의 값이 직접 사용됩니다.

td 명령은 PLC 프로그램에서는 'no op'가 됩니다. 직접 PLC 프로그램에서 좌표계에 감속 시간값을 설정하는 경우에 [Coord[x].Td]에 직접 쓰십시오.

좌표계 모터의 가속도가 데이터 구조체 요소 [Motor[x].InvDmax]에 따라 설정된 제어를 초과한 경우에 리미트 위반을 회피하기 위해 충분할 만큼 감속 시간이 연장됩니다. 좌표계가 세그먼테이션 모드가 아닌 경우([Coord[x].SegMoveTime] = 0) 또는 세그먼테이션 모드에서 특별 룩 어헤드 버퍼가 유효한 경우([Coord[x].SegMoveTime] > 0)에 자동 제어가 기능합니다.

지정된 S 곡선 가속 시간(Coord[x].Ts)이 감속 시간의 반보다 큰 경우에 전체 감속 시간은 지정된 S 곡선 시간의 2 배가 됩니다.

tm 명령으로 직접 지정했지만 벡터 거리를 벡터 피드 레이트(F)로 나누어 구한 동작 시간이 가속 시간보다 작은 경우에 가속 시간과 일치하도록 동작 시간이 증가합니다. 가속 시간은 최소 허용 동작 시간으로 기능함을 의미합니다.

예

```
td100;
td(P20);
td(45.3+sqrt(Q10));
```

tm{데이터}

기능: 동작 시간 지정

구문: **tm{데이터}**

여기에서

- {데이터}는 동작 시간을 밀리초로 나타내는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

tm 명령은 후속되는 직선 보간 모드 동작 및 원호 보간 모드 동작에 걸리는 시간(가감속 시간이 추가되기 전)을 부동 소수점 분해능을 갖는 밀리초 단위로 지정합니다. 제로 또는 양의 값을 지정해야 합니다. 이미 지정된 **tm** 또는 **F** 값을 오버라이드합니다. 또한 이 명령 후에 **tm** 또는 **F** 명령에 따라 오버라이드됩니다.

모션 프로그램에서 **tm** 명령을 실행하면 Power PMAC 은 [Coord[x].Tm]을 지정된 값으로 변경합니다(플러스 기호는 좌표계가 동작 시간 모드가 아니라 피드 레이트 모드임을 나타냅니다). **tm** 명령은 PLC 프로그램에서는 'no op'가 됩니다. 직접 PLC 프로그램에서 좌표계에 피드 레이트값을 설정하는 경우에 [Coord[x].Tm]에 직접 쓰십시오.

지정된 동작 시간이 가속 시간보다 짧은 경우에 가속 시간에 맞춰서 동작 시간이 증가합니다. 그 결과 축 거리를 동작 시간으로 나누어 결정한 속도보다도 축 속도가 느려집니다.

동작 거리를 동작 시간으로 나누어 결정한 좌표계 모터의 요구 속도가 데이터 구조체 요소 [Motor[x].MaxSpeed]에 따라 설정된 제한을 초과한 경우에 리미트 위반을 회피하기에 충분한 만큼 동작 시간이 연장됩니다. 좌표계가 세그먼테이션 모드가 아닌 경우([Coord[x].SegMoveTime] = 0) 또는 세그먼테이션 모드에서 특별 록 어헤드 버퍼가 유효한 경우([Coord[x].SegMoveTime] > 0)에 자동 제어가 기능합니다.

예

```
tm100;
tm(P20);
tm(45.3+sqrt(Q10));
```

tread

기능: 현재 실행 중인 동작 축의 대상 위치를 통지함

구문: **tread**

Power PMAC 은 **tread** 명령으로 프로그램의 [Ldata.Coord]에 따라 주소 지정된 좌표계에서 실행 중인 축의 대상 위치를 계산합니다. 저장 설정 요소 [Coord[x].TPSize]를 0 보다 큰 동작 계산 시와 동작 실행 시 사이의 모든 동작의 위치를 저장하기 위해 충분한 값으로 설정하고 목표 위치의 버퍼를 유효로 해야 합니다. 저장 설정 요소 [Coord[x].TPCoords]에 따라 지정된 축의 위치만 계산됩니다.

축 대상 위치의 값은 프로그램의 로컬 'D' 변수에 복사되고 후속 계산에서 사용됩니다. 위치는 동작이 계산된 시점에 유효한 축 변환을 사용하여 프로그램에 반환됩니다. 각 축에 사용되는 'D' 변수 'D_i'를 다음 표로 나타냅니다.

축	D _i	축	D _i	축	D _i	축	D _i	축	D _i	축	D _i	축	D _i	축	D _i
A	0	V	4	Z	8	DD	12	HH	16	OO	20	SS	24	WW	28
B	1	W	5	AA	9	EE	13	LL	17	PP	21	TT	25	XX	29
C	2	X	6	BB	10	FF	14	MM	18	QQ	22	UU	26	YY	30
U	3	Y	7	CC	11	GG	15	NN	19	RR	23	VV	27	ZZ	31

D32 는 32 비트의 마스크를 반환하고 대상 위치가 통지된 축을 통지합니다. 'D32'의 각 비트 *i* 는 변수가 통지된 변수 'D_i'의 번호에 해당합니다. 'D32' 값은 [Coord[x].TPCoords]와 같은 값이어야 합니다. 'D32'의 반환값이 0 인 경우에 위치는 통지되지 않습니다.

동작의 공구 직경 보정이 유효한 경우에 동작 종료 시에 D6, D7, D8 로 반환되는 X, Y, Z 축 위치는 각각 축 보정의 오프셋이 포함됩니다. 2D 보정이 다음 동작으로의 블렌드로 외측 코너 주위에 원호 동작을 추가한 경우에 [Coord[x].CCCtrl]의 비트 0(값 1)이 기본값 0 으로 설정되어 있으면 원호 끝에서 축 보정된 대상 위치가 통지됩니다. 이 비트가 1 로 설정되면 동작 시작 시에 통지됩니다.

버퍼 프로그램 **tread** 명령은 온라인 **t** 명령과 같은 계산을 실행합니다.

ts{데이터}

기능: S 곡선 시간 지정

구문: **ts{데이터}**

여기에서

- **{데이터}**는 S 곡선 가감속 시간을 밀리초로 나타내는 부동 소수점 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

ts 명령은 직선 보간 모드 동작 및 원호 보간 모드 동작의 명령 가속도 및 감속도 시간 양쪽에서 시작 시 및 종료 시에 S 곡선 프로파일에 소비되는 시간을 부동 소수점 분해능을 갖는 밀리초 단위로 지정합니다. 제로 또는 양의 값을 지정해야 합니다. S 곡선 시간 동안 가감속 시작 시에는 가속률이 제로에서 최대 가속도까지 직선적으로 변화합니다. 또한 가감속 종료 시에는 최대 가속도에서 제로까지 직선적으로 변화합니다.

td(감속 시간) 명령으로 직선 보간 동작 모드와 원호 보간 동작 모드에 각각의 전체 최종 감속 시간을 설정할 수도 있지만 S 곡선 시간은 둘 다 같습니다.

모션 프로그램에서 **ts** 명령이 실행되면 프로그램을 실행하는 좌표계 데이터 구조체 요소 **[Coord[x].Ts]**는 **{데이터}**에서 구해진 값으로 설정됩니다. 모션 프로그램에서 **ts** 명령이 실행되지 않으면 데이터 구조체 요소 **[Coord[x].Ts]**의 값이 직접 사용됩니다.

ts 명령은 PLC 프로그램에서는 'no op'가 됩니다. 직접 PLC 프로그램에서 좌표계에 S 곡선 시간값을 설정하는 경우에 **[Coord[x].Ts]**에 직접 쓰십시오.

좌표계 모터의 가속도 변화율('jerk'라고 함)이 데이터 구조체 요소 **[Motor[x].InvJmax]**에 따라 설정된 제어를 초과한 경우에 리미트 위반을 회피하기 위해 충분한 S 곡선 시간으로 연장됩니다. 좌표계가 세그먼테이션 모드가 아닌 경우(**[Coord[x].SegMoveTime] = 0**)에만 직선 보간 모드 동작에 대해 자동 제어가 기능합니다.

지정된 S 곡선 가속 시간(**Coord[x].Ts**)이 가속 또는 감속 시간의 반보다 큰 경우에 전체 가속 또는 감속 시간은 지정된 S 곡선 시간의 2 배가 됩니다.

예

```
ts100;
ts(P20);
ts(45.3+sqrt(Q10));
```

tset{데이터}

기능: 좌표계의 활성 변환 매트릭스를 선택함

구문: **tset{데이터}**

여기에서

- **{데이터}**는 선택된 변환 매트릭스 수를 지정합니다.

tset 명령은 지정된 축 변환 매트릭스를 주소 지정된 좌표계의 활성 변환 매트릭스로 선택합니다(모션 프로그램인 경우에 프로그램을 실행 중인 좌표계입니다). 축 변환 매트릭스의 유효한 숫자는 0~255 입니다. **{데이터}**에 따라 지정된 값이 정수값으로 평가되지 않는 경우에는 가까운 정수로 버립니다.

좌표계에 대해 현재 선택된 매트릭스 수는 데이터 구조체 요소 [Coord[x].Tset]에서 확인할 수 있습니다.

-1 값을 지정하면 변환 매트릭스는 모두 선택 해제되고 계산 시간이 좌표계에 저장됩니다. 이것은 모든 좌표계의 전압 셋업/리셋 시 기본값입니다.

txyz

기능: 3D 공구 직경 보정용으로 툴 배향 벡터를 지정함

구문: **txyz{벡터}{데이터} [{벡터}{데이터}...]**

여기에서

- **{벡터}**는 각각 X, Y, Z 축과 평행한 벡터 컴포넌트를 지정하는 문자(I, J, K)입니다.
- **{데이터}**는 벡터 컴포넌트를 나타내는 상수(소괄호 없음) 또는 식(소괄호 있음)입니다.

txyz 명령은 3D 공구 직경 보정에 쓰이는 툴 배향 벡터를 지정합니다. 벡터 컴포넌트를 X, Y, Z 축에 따라 설정하여 지정합니다. 각각 I, J, K 문자 뒤의 값으로 설정하는 이러한 컴포넌트의 절대량은 문제가 되지 않습니다. 컴포넌트의 상대값이 벡터 방향을 결정합니다. 툴 배향 벡터 방향 감지(베이스에서 선단부 또는 선단부에서 베이스)는 중요하지 않습니다.

이 명령으로 정의한 툴 배향 벡터는 새로운 면 법선 벡터가 지정될 때까지 모션 프로그램의 동일한 행에서 명령된 동작 및 모든 후속 동작의 보정 오프셋 결정에 쓰입니다. 단, 3D 보정에서는 명령된 동작마다 개별적으로 면 법선 벡터가 지정되는 것이 표준적인 방법입니다.

툴 배향 벡터의 컴포넌트가 **txyz** 명령으로 명확하게 선언되지 않은 경우에 그 컴포넌트는 자동으로 0.0 으로 설정됩니다.

txyz 명령에 따라 공구 방향이 이 방향으로 되지 않는 경우에 축 동작 명령 컴포넌트를 이용해 실행할 필요가 있습니다. **txyz** 명령은 표준 형식을 사용하고 단순히 3D 보정 알고리즘으로 공구 방향을 통지합니다.

vread

기능: 축 현재 속도를 통지함

구문: **vread**

Power PMAC 은 **vread** 명령으로 프로그램의 [Ldata.Coord]에 따라 주소 지정된 좌표계의 축 실제 (필터링된)속도를 계산합니다. 좌표계의 모든 활성 축의 현재 속도가 계산됩니다.

축의 현재 속도는 좌표계 정의에 따라 처리되어(축 정의 명령문을 반전하거나 포워드 키네마틱스 루틴을 사용) 대응하는 모터의 명령의 이력 및 유효한 매트릭스 변환으로 계산됩니다. 포워드 키네마틱스 루틴을 사용하는 경우에 현재와 과거 위치로부터 속도를 계산할 수 있도록 더블 버스를 실행할 수 있어야 합니다.

축 현재 속도의 값은 프로그램의 로컬 'D' 변수에 복사되고 후속 계산에서 사용됩니다. 속도는 유효한 축 변환 매트릭스를 사용한 현재 축의 단위로 복귀합니다. 각 축에 사용되는 'D' 변수 'Di'는 다음 표와 같습니다.

축	Di	축	Di	축	Di	축	Di	축	Di	축	Di	축	Di	축	Di
A	0	V	4	Z	8	DD	12	HH	16	OO	20	SS	24	WW	28
B	1	W	5	AA	9	EE	13	LL	17	PP	21	TT	25	XX	29
C	2	X	6	BB	10	FF	14	MM	18	QQ	22	UU	26	YY	30
U	3	Y	7	CC	11	GG	15	NN	19	RR	23	VV	27	ZZ	31

'D32'는 32 비트의 마스크를 반환하고 현재 속도가 통지된 축을 통지합니다. 'D32'의 각 비트 *i* 는 변수가 통지된 변수 'Di'의 번호에 상당합니다. 'D32'의 반환값이 0 인 경우에 위치는 통지되지 않습니다.

버퍼 프로그램 **vread** 명령은 온라인 **v** 명령과 같은 계산을 실행합니다.

while({조건})

기능: 조건 그룹

구문: **while({조건})**

여기에서

- **{조건}**은 단일 또는 복합 조건 또는 조건 비교 연산자가 없는 수식입니다.

완전한 구조 구문:

```

while({조건}){명령}{명령}...
.....while({조건}) {
..... {조건}
..... [{조건}...]
..... }

```

while 명령으로 모션 프로그램과 PLC 프로그램에서 조건 루프를 사용할 수 있습니다. 소괄호로 둘러싸인 조건의 평가가 True 인 경우 또는 식의 평가가 제로 이외의 값인 경우에 루프 내부의 프로그램 명령이 실행됩니다.

while 명령은 루프 동작 전에 사용할 수 있습니다. 그 경우에 루프를 실행하기 전에 조건을 평가합니다. 조건 평가가 False 인 경우에 루프 내의 명령은 1 회도 실행되지 않습니다. 조건은 **do** 명령으로 시작하는 루프 마지막에도 사용할 수 있습니다. 이 경우에 루프 내의 명령은 항상 1 회 이상 실행됩니다.

루프 선두에서 **while** 명령을 사용하고 그 조건이 **True** 일 때 프로그램의 동일한 행 직후에 명령이 있는 경우에 동일한 행 명령만이 실행됩니다.

조건을 닫는 소괄호 직후에 여는 중괄호(**{**)가 있는 경우에는 같은 행이나 다음 프로그램행의 선두라도 이 중괄호는 조건이 **True** 인 경우에 실행하는 명령의 시작을 나타냅니다. 이 명령의 실행은 닫는 중괄호(**}**)가 있는 행까지 계속합니다.

do 명령으로 시작하는 루프 마지막에 **while** 명령을 사용하는 경우에 프로그램의 동일한 행의 후속 명령이 루프의 일부가 아니기 때문에 조건 평가가 **False** 가 될 때까지 실행되지 않습니다.

while 루프는 다른 조건 구조체(**if**, **while**, **switch**) 내부에서 네스트할 수 있습니다. 또한 다른 조건 구조체를 **while** 루프의 내부에서 네스트할 수 있습니다. 조건 구조체는 최대 32 레벨로 네스트할 수 있습니다.

예

```
while (M1 == 0) dwell 10;

while (P1 > 0) {}

while (Q100) {
X10;
X0;
}

P50 = 0;
do {
    YY200;
    dwell 50;
    YY10;
    P50++
}
while (P50 < 100);
```

POWER PMAC 프로젝트 강화 스크립트 언어 명령

PC 용 Power PMAC 의 Integrated Development Environment(IDE) 소프트웨어 프로젝트 매니저는 기본적인 Power PMAC 스크립트 언어 능력과 유연성을 대폭 강화시킵니다. 여기서는 강화된 내용을 자세하게 설명합니다. 강화된 스크립트를 Power PMAC 에서 문제없이 사용하기 위해서는 IDE 프로젝트 매니저의 다운로드 기능을 사용하여 Power PMAC 스크립트의 기본적인 처리를 실행해야 합니다. Power PMAC 용으로 커스터마이징하지 않은 명령을 직접 Power PMAC 으로 전송하면 에러로 거부됩니다.

스크립트 전처리 명령

이 섹션의 전처리 명령을 사용하면 사용자가 명령 다운로드를 조건부로 제어하기 위해 제어 플래그를 지정하고 사용할 수 있습니다. 또한 상수, 변수 및 식의 텍스트 변환을 작성할 수도 있습니다.

#define {식별자}

기능: 전처리기의 플래그를 작성함

범위: 글로벌에서 Power PMAC 프로젝트

구문: **#define {식별자}**

여기에서

- **{식별자}**는 전처리기의 플래그명을 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다.

사용자는 이 명령을 이용하여 IDE 프로젝트 매니저의 조건부 전처리기 명령하며 그중에서도 특히 조건부 명령 **#ifdef** 및 **#ifndef** 플래그를 작성할 수 있습니다. 이 명령의 결과는 **#undef** 명령에 따라 원래대로 복귀할 수 있습니다.

이 방법으로 정의된 식별자는 값이 할당되지 않은 경우에 자동으로 값 0이 부여됩니다. 이 경우에 **#ifdef {식별자}** 조건은 True 로 평가하고 **#if {식별자}** 조건은 False 로 평가합니다.

예

```
#define TestMode
#ifdef TestMode
    &1%25
#endif
```

#define {식별자} {토큰 문자열}

기능: 텍스트 대체 매크로를 작성함

범위: 글로벌에서 Power PMAC 프로젝트

구문: **#define {식별자} {토큰 문자열}**

여기에서

- **{식별자}**는 다운로드 중에 치환할 텍스트를 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다.
- **{토큰 문자열}**은 식별자를 대신하는 텍스트 문자열입니다. 이것은 세미콜론, 행바꿈 또는 명령 시작 문자(// 또는 /*)로 종료시킵니다.

사용자는 이 명령으로 IDE 프로젝트 매니저를 사용하여 다운로드한 프로젝트 내에서 텍스트 대체를 지정할 수 있습니다. 프로젝트 매니저는 **{식별자}**로 지정한 텍스트를 검출할 때마다 **{토큰 문자열}** 텍스트로 치환합니다. 식별자는 대문자와 소문자를 구분합니다. 이러한 대체에 따라 명령 및 프로그램 내에서 의미를 가지는 이름을 사용할 수 있습니다.

예

```
#define True 1;
#define _Length2 Q100+10
#define SixFactorial (6*5*4*3*2)
#define Cycle_Counter P10
#define Clear_Cycle_Counter P10=0
```

#define {식별자} {토큰 문자열} \

기능: 여러 행의 텍스트 대체 매크로를 작성함

범위: 글로벌에서 Power PMAC 프로젝트

구문: **#define {식별자} {토큰 문자열} **
{토큰 문자열} []...

여기에서

- **{식별자}**는 다운로드 중에 치환할 텍스트를 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다.
- **{토큰 문자열}**은 식별자를 대신하는 텍스트 문자열입니다. 이것은 세미콜론, 행바꿈 또는 명령 시작 문자(// 또는 /*)로 종료시킵니다.

사용자는 이 명령으로 IDE 프로젝트 매니저를 사용하여 다운로드한 프로젝트 내에서 여러 행의 텍스트 대체를 지정할 수 있습니다. 프로젝트 매니저는 **{식별자}**로 지정한 텍스트를 검출할 때마다 **{토큰 문자열}** 표시 세트의 텍스트로 치환합니다. 식별자는 대문자와 소문자를 구분합니다.

개별 토큰 문자열 뒤에 \ 백슬래시 문자가 있는 경우에 텍스트 대체는 다음 행에서 토큰 문자열을 이어갑니다(치환된 텍스트에는 행 구분 기호가 포함되지 않습니다). 대체는 뒤에 백슬래시 문자가 없는 토큰 문자열에서 끝납니다.

여러 행의 텍스트 대체 매크로는 다운로드된 IDE 프로젝트 이외에서는 기동할 수 없습니다. 그러한 매크로들은 1행 매크로와는 달리 IDE 터미널 창이나 다른 gpascii -2 통신 스레드와 같은 명령행에서 기동할 수 없습니다.

예

```
#define ClearCounters P100=0\
P101=0\
P102=0\
P103=0
```

#define {식별자} ({인수 목록}) {토큰 문자열} [N]

기능: 함수 텍스트 기능 대체 매크로를 작성함

범위: 글로벌에서 Power PMAC 프로젝트

구문: **#define {식별자} ({인수 목록}) {토큰 문자열}**

**#define {식별자}({인수 목록}) {토큰 문자열} **
{토큰 문자열} [N]...

여기에서

- **{식별자}**는 다운로드 중에 치환할 함수명을 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하는 영숫자와 _(스페이스 없음)가 이어집니다.
- **{인수 목록}**은 하나 이상의 텍스트 문자열입니다. 여러 개인 경우에는 쉼표로 구분하며 함수에서 사용하는 인수를 나타냅니다.
- **{토큰 문자열}**은 식별자를 대신하는 텍스트 문자열입니다. 이것은 세미콜론, 행바꿈 또는 명령 시작 문자(/ 또는 *)로 종료시킵니다.

사용자는 이 명령으로 IDE 프로젝트 매니저를 사용하여 다운로드한 프로젝트 내에서 인수를 가지는 함수의 텍스트 대체를 지정할 수 있습니다. 프로젝트 매니저는 **{식별자}**로 지정한 텍스트를 검출할 때마다 **{토큰 문자열}** 표시 세트의 텍스트를 지정된 인수로 치환합니다. 식별자는 대문자와 소문자를 구분합니다.

개별 토큰 문자열 뒤에 \ 백슬래시 문자가 있는 경우에 텍스트 대체는 다음 행에서 토큰 문자열을 이어갑니다(치환된 텍스트에는 행 구분 기호가 포함되지 않습니다). 대체는 뒤에 백슬래시 문자가 없는 토큰 문자열에서 끝납니다.

함수 텍스트 대체 매크로는 다운로드된 IDE 프로젝트 이외에서는 기동할 수 없습니다. 그러한 매크로들은 1 행 매크로와는 달리 IDE 터미널 창이나 다른 gpascii -2 통신 스레드와 같은 명령행에서 기동할 수 없습니다.

예

```
#define SindSquared(Theta) sind(Theta)*sind(Theta)

#define SindAngleSum(A,B) sind(A)*cosd(B) + cosd(A)*sind(B)

#define _Pi 3.14159265
#define sinc(x) sin(_Pi*x)/(_Pi*x)

#define Factorial(Pnum,X) P(Pnum)=X;\
while (X > 1) {\
X--;\
P(Pnum)=P(Pnum)*X;\
}
```

#elif

기능: 복합 False 조건부 명령 분기를 시작함

범위: 글로벌에서 Power PMAC 프로젝트

구문: **#elif {식별자} [{비교자} {상수}]**

여기에서

- **{식별자}**는 평가할 상수 또는 파라미터를 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다.
- **{비교자}**는 다음 조건부 콤퍼레이터 중 하나입니다. ==, !=, <, >, <=, >=
- **{상수}**는 부호가 있는 32 비트 범위 내의 정수입니다.

사용자는 이 명령을 사용하여 프로젝트 다운로드 명령 실행 중에 선행 조건의 False 분기 안에 네스트된 조건부 분기를 작성할 수 있습니다. 이것은 **#if**, **#ifdef**, **#ifndef** 및 **#elif** 조건부 명령으로 사용할 수 있습니다.

행의 조건 평가가 True 인 경우에 직후의 명령이 **#endif**, **#else** 또는 **#elif** 명령을 실행합니다.

명령이 단순한 형식 **#elif{식별자}**인 경우에는 **{식별자}**가 정의되고 제로 이외의 상수값이 할당된 경우에 한해 조건 평가가 True 가 됩니다.

명령 형식이 **#elif{식별자}{비교자}{상수}**인 경우에는 **{식별자}**의 엔터티값이 **{비교자}**에 따라 지시된 **{상수}**값과 관계있는 경우에 한해 조건 평가가 True 가 됩니다.

예

```
#if SafetyLevel == 1
    Motor[5].FatalFeLimit = 2000
#elif SafetyLevel == 2
    Motor[5].FatalFeLimit = 1000
#else
    Motor[5].FatalFeLimit = 500
#endif
```

#else

기능: False 조건부 명령 분기를 시작함

범위: 글로벌에서 Power PMAC 프로젝트

구문: **#else**

사용자는 이 명령을 이용하여 앞의 조건부 명령 평가가 False 인 경우에 실행되는 조건부 분기를 작성할 수 있습니다. 이것은 **#if**, **#ifdef**, **#ifndef** 및 **#elif** 조건부 명령으로 사용할 수 있습니다.

선행하는 조건부 명령 평가가 True 인 경우 실행은 이 **#else** 명령에 이어지는 명령으로 건너뛰고 다음 **#endif** 명령까지 계속 실행합니다. **#else** 와 **#endif** 사이에 다른 조건부 명령이 들어가지 않습니다.

예

```
#define TestMode
#ifdef TestMode
    &1%25
#else
    &1%100
#endif
```

#endif

기능: False 조건부 명령 분기를 종료함

범위: 글로벌에서 Power PMAC 프로젝트

구문: **#endif**

사용자는 이 명령에 따라 **#if**, **#ifdef**, **#ifndef**, **#else** 및 **#elif** 조건부 명령 중 하나로 시작하는 조건부 분기를 종료할 수 있습니다.

예

```
#if SafetyLevel >= 3
    Motor[5].FatalFeLimit = 100
#endif

#define TestMode
#ifdef TestMode
    &1%25
#else
    &1%100
#endif
```

#if

기능: 값에 따른 조건부 명령 분기를 시작함

범위: 글로벌에서 Power PMAC 프로젝트

구문: **#if {식별자} [{비교자} {상수}]**

여기에서

- **{식별자}**는 평가할 상수 또는 파라미터를 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다.
- **{비교자}**는 다음 조건부 콤퍼레이터 중 하나입니다. ==, !=, <, >, <=, >=
- **{상수}**는 부호가 있는 32 비트 범위 내의 정수입니다.

사용자는 이 명령을 사용하여 프로젝트 다운로드 명령 실행 중에 조건부 분기를 작성할 수 있습니다. 행의 조건 평가가 True 인 경우에 직후의 명령이 **#endif**, **#else** 또는 **#elif** 명령을 실행합니다.

명령이 단순한 형식 **#if{식별자}**인 경우에는 **{식별자}**가 정의되고 제로 이외의 상수값이 할당되어 있는 경우에 한해 조건 평가가 True 가 됩니다.

명령 형식이 **#if{식별자}{비교자}{상수}**인 경우에는 **{식별자}**의 엔터티값이 **{비교자}**에 따라 지시된 **{상수}**값과 관계있는 경우에 한해 조건 평가가 True가 됩니다.

예

```
#if SafetyLevel >= 3
    Motor[5].FatalFeLimit = 100
#endif

#if Expanded_Machine
    Sys.MaxMotors = 9
#else
    Sys.MaxMotors = 5
#endif
```

#ifdef

기능: 존재에 근거한 조건부 명령 분기를 시작함

범위: 글로벌에서 Power PMAC 프로젝트

구문: **#ifdef {식별자}**

여기에서

- {식별자}**는 평가할 상수 또는 파라미터를 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다.

사용자는 이 명령을 사용하여 프로젝트 다운로드 명령 실행 중에 조건부 분기를 작성할 수 있습니다. 행의 조건 평가가 True인 경우에 직후의 명령이 **#endif**, **#else** 또는 **#elif** 명령을 실행합니다.

(값의 할당 여부에 관계없이){식별자} 엔터티가 **#define** 명령에 따라 정의된 경우에 조건 평가는 True가 됩니다.

예

```
#define _AbsMode
#ifdef _AbsMode
    Motor[1].pAbsPos = Gate3[0].Chan[0].SerialEncDataA
    ...
#endif
```

#ifndef

기능: 비존재에 근거하여 조건부 명령 분기를 시작함

범위: 글로벌에서 Power PMAC 프로젝트

구문: **#ifndef {식별자}**

여기에서

- {식별자}**는 평가할 상수 또는 파라미터를 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다.

사용자는 이 명령을 사용하여 프로젝트 다운로드 명령 실행 중에 조건부 분기를 작성할 수 있습니다. 행의 조건 평가가 True 인 경우에 직후의 명령이 **#endif**, **#else** 또는 **#elif** 명령을 실행합니다.

(값의 할당 여부에 관계없이){**식별자**} 엔터티가 **#define** 명령으로 정의되지 않은 경우에 또는 그런 정의가 **#undef** 명령으로 삭제된 경우에 조건 평가는 True 가 됩니다.

예

```
#ifndef _AbsMode
    Motor[1].pAbsPos = 0
    ...
#endif
```

#undef {식별자}

기능: 전처리기의 플래그 정의 또는 매크로 대체 정의를 원래대로 복귀함

범위: 글로벌에서 Power PMAC 프로젝트

구문: **#undef {식별자} [{식별자}...]**

여기에서

- **{식별자}**는 전처리기의 플래그명 또는 매크로 대체명을 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다.

사용자는 이 명령을 이용하여 전처리기의 플래그 정의 또는 매크로 대체 정의를 원래대로 복귀할 수 있습니다. 매크로 대체 정의를 원래대로 복귀하면 전처리는 프로젝트 로드 중에 텍스트 대체를 실행하지 않습니다.

예

```
#define _DebugMode
...
#undef _DebugMode

#define Indicator2Bank M100
...
#undef Indicator2Bank
```


스크립트 변수 선언

이 섹션의 선언에서는 다양한 타입의 사용자 변수에 의미 있는 이름을 부여하기 위한 가장 간단한 방법을 소개합니다.

csglobal

기능: 좌표계 고유의 글로벌('Q') 변수에 사용자명을 자동으로 할당함

범위: 글로벌에서 프로젝트 선언, 좌표계 고유의 초기화 값

구문: **csglobal {식별자} [= {식}]**
[, {식별자} [= {식}]]...;
csglobal {식별자} ({array_size})
[, {식별자} ({array_size})]...;

여기에서

- **{식별자}**는 사용자 변수명을 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다.
- **{식}**은 수치와 평가되는 상수, 변수, 함수 및 연산자의 정합적인 조합입니다. 이것은 세미콜론, 행바꿈 또는 명령 시작 문자(// 또는 /*)로 종료시킵니다.
- **{array_size}**는 양의 정수 상수 또는 양의 정수 상수를 위해 이전에 정의된 식별자입니다.

csglobal 선언으로 사용자 정의 식별자명을 각 좌표계의 'Q' 변수 번호와 함께 Power PMAC 좌표계의 글로벌('Q') 변수 세트에 자동으로 할당합니다. 식별자는 대문자와 소문자를 구분합니다.

Power PMAC 각 좌표계에는 8,192 개의 'Q' 변수(Q0~Q8191)가 있습니다. 프로젝트 매니저가 이러한 'Q' 변수에 할당을 시작하지만 할당하는 번호는 IDE 프로젝트의 [Properties] 창에서 정의한 'Q' 변수의 시작점에서 지정됩니다. 이 기본값은 'Q1024'입니다. 보통 하위 번호 변수는 직접 사용되거나 #define 명령에 따라 사용자명이 부여됩니다. 이러한 사용자명 부여는 특히 이전 Turbo PMAC 에서 어플리케이션을 이식한 경우 등에 사용됩니다.

사용자명 직후에 괄호로 둘러싸인 배열 크기가 이어지는 경우(양의 정수 상수 또는 양의 정수 상수를 위해 이전에 정의된 식별자)에 이 선언을 사용하여 'Q' 변수의 사용자 배열을 선언할 수 있습니다. 이러한 경우에 프로젝트 매니저는 연속되는 번호가 있는 'Q' 변수를 지정 수만큼 배열에 할당합니다.

사용자는 단일 변수(배열이 아닌)를 선언하기 위해 프로젝트를 다운로드할 때 전원 셋업 시 또는 리셋 시에 자동으로 변수에 할당되는 값을 지정할 수 있습니다. 이 값은 수식(단일 수치 상수처럼 단순한 경우가 있습니다)에 따라 지정됩니다. 단, 이 값은 주소 지정된 좌표계의 'Q' 변수에만 할당할 수 있습니다.

csglobal 선언은 Power PMAC 스크립트 프로그램 또는 서브 프로그램 외부에 있어야만 합니다. 사용자에게 이러한 선언 전부를 [global definitions.pmh] 프로젝트 파일과 같은 단일 파일에 넣을 것을 권장합니다.

예

```

csglobal CycleCount;
&1
csglobal MaxErrors = 5;
csglobal Xscale, Yscale, Zscale;

#define True 1;
#define False 0;
&2
csglobal SysInitComplete = False;
csglobal OperationMode = True;

csglobal XyzOffset(3);
#define ListLength 500;
csglobal BatchRMS(ListLength);

```

global

기능: 글로벌('P') 변수에 사용자명을 자동으로 할당함

범위: 글로벌에서 프로젝트

구문: **global {식별자} [= {식}]**
 [, {식별자} [= {식}]]...;
 global {식별자} ({array_size})
 [, {식별자} ({array_size})]...;

여기에서

- **{식별자}**는 사용자 변수명을 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _ (스페이스 없음)가 이어집니다.
- **{식}**은 수치와 평가되는 상수, 변수, 함수 및 연산자의 정합적인 조합입니다. 이것은 세미콜론, 행바꿈 또는 명령 시작 문자(// 또는 /*)로 종료시킵니다.
- **{array_size}**는 양의 정수 상수 또는 양의 정수 상수를 위해 이전에 정의된 식별자입니다.

global 선언은 사용자 정의 식별자명이 IDE 프로젝트 매니저에 따라 Power PMAC 글로벌('P') 변수에 자동으로 할당되는 요인입니다. 식별자는 대문자와 소문자를 구분합니다.

Power PMAC에는 65,536 개의 'P' 변수(P0~P65535)가 있습니다. 프로젝트 매니저가 이러한 'P' 변수에 할당을 시작하지만 할당하는 번호는 IDE 프로젝트의 [Properties] 창에서 정의한 'P' 변수의 시작점에서 지정됩니다. 이 기본값은 'P8192'입니다. 보통 하위 번호 변수는 직접 사용되거나 **#define** 명령에 따라 사용자명이 부여됩니다. 이러한 사용자명 부여는 특히 이전 Turbo PMAC에서 어플리케이션을 이식한 경우 등에 사용됩니다.

사용자명 직후에 괄호로 둘러싸인 배열 크기가 이어지는 경우(양의 정수 상수 또는 양의 정수 상수를 위해 이전에 정의된 식별자)에 이 선언을 사용하여 'P' 변수의 사용자 배열을 선언할 수 있습니다. 이러한 경우에 프로젝트 매니저는 연속되는 번호가 있는 'P' 변수를 지정 수만큼 배열에 할당합니다.

사용자는 단일 변수(배열이 아닌)를 선언하기 위해 프로젝트를 다운로드할 때 전원 셋업 시 또는 리셋 시에 자동으로 변수에 할당되는 값을 지정할 수 있습니다. 이 값은 수식(단일 수치 상수처럼 단순한 경우가 있습니다)에 따라 지정됩니다. 이를 이용하여 어플리케이션의 변수값을 단순한 방법으로 초기화할 수 있습니다.

global 선언은 Power PMAC 스크립트 프로그램 또는 서브 프로그램 외부에 있어야만 합니다. 사용자에게 이러한 선언 전부를 [global definitions.pmh] 프로젝트 파일과 같은 단일 파일에 넣을 것을 권장합니다.

예

```
global CycleCount;
global MaxErrors = 5;
global Xscale, Yscale, Zscale;

#define True 1;
#define False 0;
global SysInitComplete = False;
global OperationMode = True;

global XyzOffset(3);
#define ListLength 500;
global BatchRMS(ListLength);
```

local

기능: 로컬('L') 변수에 사용자명을 자동으로 할당함

범위: 로컬에서 프로그램 또는 서브 프로그램

구문: **local {식별자} [= {식}]**
[, {식별자} [= {식}]]...;

local {식별자} ({array_size})
[, {식별자} ({array_size})]...;

여기에서

- **{식별자}**는 사용자 변수명을 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다.
- **{식}**은 수치와 평가되는 상수, 변수, 함수 및 연산자의 정합적인 조합입니다. 이것은 세미콜론, 행바꿈 또는 명령 시작 문자(// 또는 /*)로 종료시킵니다.
- **{array_size}**는 양의 정수 상수 또는 양의 정수 상수를 위해 이전에 정의된 식별자입니다.

local 선언은 사용자 정의 식별자명이 IDE 프로젝트 매니저에 따라 Power PMAC 로컬('L') 변수에 자동으로 할당되는 요인입니다. 식별자는 대문자와 소문자를 구분합니다.

Power PMAC에는 각 실행 스레드(좌표계, PLC 프로그램, 통신 스레드)용으로 8,192개의 로컬 변수 스택이 있습니다. 호출 측 서브루틴 또는 서브 프로그램의 프로그램 또는 서브 프로그램의 스택 오프셋 기본값은 256이지만 프로그램 선언으로 변경할 수 있습니다. IDE 프로젝트 매니저는 이 스택 오프셋을 자동으로 최소값에 설정합니다. 이 값은 전달될 선언된 로컬 변수를 수용하는 크기가 됩니다.

프로젝트 매니저는 서브 프로그램에 인수를 전달할 때 'L' 변수를 사용한 경우에 그 할당을 다음과 같이 시작합니다. 예를 들면 서브 프로그램으로 전달한 인수가 3 개, 반환된 인수가 1 개인 경우에 해당 인수는 서브 프로그램의 'L0'에서 'L3'에 할당되고 서브 프로그램 내에서 선언된 로컬 변수는 'L4'에서 시작합니다.

사용자명 직후에 괄호로 둘러싸인 배열 크기가 이어지는 경우(양의 정수 상수 또는 양의 정수 상수를 위해 이전에 정의된 식별자)에 이 선언을 사용하여 'L' 변수의 사용자 배열을 선언할 수 있습니다. 이러한 경우에 프로젝트 매니저는 연속되는 번호가 있는 'L' 변수를 지정 수만큼 배열에 할당합니다.

사용자는 단일 변수(배열이 아닌)를 선언하기 위해 프로그램 또는 서브 프로그램이 선두에 입력될 때마다 자동으로 변수에 할당되는 값을 지정할 수 있습니다. 이 값은 수식(단일 수치 상수처럼 단순한 경우가 있습니다)에 따라 지정됩니다. 이를 이용하여 루틴을 실행할 때 변수값을 단순한 방법으로 초기화할 수 있습니다.

local 선언은 Power PMAC 스크립트 프로그램 또는 서브 프로그램 내부에 있어야만 합니다.

예

```
open prog 100
local LoopCount;
local MaxErrors = 5;
local Xscale, Yscale, Zscale;

#define True 1;
#define False 0;
open subprog StepAndRepeat
local StepComplete = False;
local OperationMode = True;

#define NumLoops 500;
open plc PickAndPlace
local XyzOffset(3);
local LengthError(NumLoops);
```

ptr

기능: 포인터('M') 변수에 사용자명을 자동으로 할당함

범위: 글로벌에서 프로젝트

구문: **ptr {식별자}->{정의} [= {식}]**
[,{식별자}->{정의} [= {식}]]...;
ptr {식별자} ({array_size})->{정의}
[, {식별자} ({array_size})->{정의}]]...;

여기에서

- **{식별자}**는 사용자 변수명을 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _ (스페이스 없음)가 이어집니다.
- **{정의}**는 모든 유효한 타입의 Power PMAC의 M 변수 정의입니다.

- **{식}**은 수치와 평가되는 상수, 변수, 함수 및 연산자의 정합적인 조합입니다. 이것은 세미콜론, 행바꿈 또는 명령 시작 문자(`//` 또는 `/*`)로 종료시킵니다.
- **{array_size}**는 양의 정수 상수 또는 양의 정수 상수를 위해 이전에 정의된 식별자입니다.

ptr 선언은 사용자 정의 식별자명이 IDE 프로젝트 매니저에 따라 Power PMAC 포인터('M') 변수에 자동으로 할당되는 요인입니다. 식별자는 대문자와 소문자를 구분합니다.

Power PMAC에는 65,536 개의 'M' 변수(M0~M65535)가 있습니다. 프로젝트 매니저는 해당 'M' 변수에 할당을 시작하지만 할당하는 번호는 IDE 프로젝트의 [Properties] 창에서 정의한 'M' 변수의 시작점에서 지정됩니다. 이 기본값은 'P8192'입니다. 보통 하위 번호 변수는 직접 사용되거나 `#define` 명령에 따라 사용자명이 부여됩니다. 이러한 사용자명 부여는 특히 이전 Turbo PMAC에서 어플리케이션을 이식한 경우 등에 사용됩니다.

사용자명 직후에 괄호로 둘러싸인 배열 크기가 이어지는 경우(양의 정수 상수 또는 양의 정수 상수를 위해 이전에 정의된 식별자)에 이 선언을 사용하여 'M' 변수의 사용자 배열을 선언할 수 있습니다. 이러한 경우에 프로젝트 매니저는 연속되는 번호가 있는 'M' 변수를 지정 수만큼 배열에 할당합니다. 정의는 주소 타입이어야만 합니다. 또한 연속되는 번호가 있는 레지스터에 지정합니다.

사용자는 단일 변수(배열이 아닌)를 선언하기 위해 프로젝트를 다운로드할 때 전원 셋업 시 또는 리셋 시에 자동으로 변수에 할당되는 값을 지정할 수 있습니다. 이 값은 수식(단일 수치 상수처럼 단순한 경우가 있습니다)에 따라 지정됩니다. 이를 이용하여 어플리케이션의 변수값을 단순한 방법으로 초기화할 수 있습니다(물론 변수가 읽기 전용 레지스터 또는 자동적인 Power PMAC 알고리즘에 따라 쓰인 레지스터를 가리키는 경우에 이 할당이 유효하지 않을 가능성이 있습니다).

ptr 선언은 Power PMAC 스크립트 프로그램 또는 서브 프로그램 외부에 있어야만 합니다. 사용자에게 이러한 선언 전부를 [global definitions.pmh] 프로젝트 파일과 같은 단일 파일에 넣을 것을 권장합니다.

예

```
ptr ContactSwitch->Acc68E[0].DataReg[1].2;  
ptr LaserOn->Acc68E[1].DataReg[3].7 = 0;  
ptr FirstIoCardReg(8)->u.io:$A00000.8.8+
```

프로그램 및 서브루틴 선언

이 섹션의 Power PMAC 프로그램 및 서브루틴 정의는 어플리케이션에서 이러한 루틴에 의미 있는 이름을 부여하는 방법을 제공합니다.

open forward

기능: 포워드 키네마틱스 루틴 버퍼를 오픈함

범위: 좌표계 고유

구문: **open forward** [{상수}]

여기에서

- 옵션 {상수}는 0~127 범위 내의 정수로 이 루틴을 사용하는 좌표계 번호를 지정합니다.

open forward 명령은 좌표계의 포워드 키네마틱스 서브루틴 버퍼를 오픈합니다. 괄호로 둘러싸인 좌표계 번호가 명령 끝에서 지정된 경우에는 그 좌표계의 버퍼를 오픈합니다. 괄호로 둘러싸인 번호가 명령 끝에서 지정되지 않은 경우에는 모달에 주소 지정된 좌표계의 버퍼를 오픈합니다.

전처리기는 키네마틱스 루틴이 의미 있는 이름을 사용할 수 있도록 자동으로 다음과 같은 **#define** 치환을 제공합니다.

```
#define KinPosMotorx      Lx          // Motor # x = 0 .. 255
#define KinPosMotor(x)    L(x)        // Motor # x = 0 .. 255
#define KinVelMotorx      Rx          // Motor # x = 0 .. 255
#define KinVelMotor(x)    R(x)        // Motor # x = 0 .. 255
#define KinPosAxis $\alpha$     Cy          // Axis index y = 0 .. 31
#define KinPosAxis(y)     C(y)        // Axis index y = 0 .. 31
#define KinVelAxis $\alpha$       C{y+32}    // Axis index y = 0 .. 31
#define KinVelAxis(y)     C(y+32)    // Axis index y = 0 .. 31
#define KinEnaAxis $\alpha$       pow(2,y)   // Axis index y = 0 .. 31
#define KinAxisIndex $\alpha$     y          // Axis index y = 0 .. 31
#define KinVelEna          D0          // Velocity calcs needed
#define KinAxisUsed        D0          // Bits for axes used
```

축 문자 α 에 대응하는 변수 번호 'y'는 다음 표와 같습니다.

축 α	y	축 α	y	축 α	y	축 α	y	축 α	y	축 α	y	축 α	y	축 α	y
A	0	V	4	Z	8	DD	12	HH	16	OO	20	SS	24	WW	28
B	1	W	5	AA	9	EE	13	LL	17	PP	21	TT	25	XX	29
C	2	X	6	BB	10	FF	14	MM	18	QQ	22	UU	26	YY	30
U	3	Y	7	CC	11	GG	15	NN	19	RR	23	VV	27	ZZ	31

전처리기는 로컬 변수의 스택 오프셋 크기와 행 점프 레이블 수를 루틴 내용에 근거하여 자동으로 지정합니다.

예

이 간단한 예에서는 #3->1000X 및 #4->1000Y에 해당하는 설정으로 좌표계 2의 포워드 키네마틱스 루틴을 나타냅니다.

```
open forward (2)
if (KinVelEna) callsub 100;    // For double-pass calculations
KinAxisUsed = KinEnaAxisX + KinEnaAxisY;    // X and Y computed
N100:
KinPosAxisX = KinPosMotor3 / 1000;
KinPosAxisY = KinPosMotor4 / 1000;
close
```

open inverse

기능: 인버스 키네마틱스 루틴 버퍼를 오픈함

범위: 좌표계 고유

구문: **open inverse** [{상수}]

여기에서

- 옵션 **{상수}**는 0~127 범위 내의 정수로 이 루틴을 사용하는 좌표계 번호를 지정합니다.

open inverse 명령은 좌표계의 인버스 키네마틱스 서브루틴 버퍼를 오픈합니다. 괄호로 둘러싸인 좌표계 번호가 명령 끝에서 지정된 경우에는 그 좌표계의 버퍼를 오픈합니다. 괄호로 둘러싸인 번호가 명령 끝에서 지정되지 않은 경우에는 모달에 주소 지정된 좌표계의 버퍼를 오픈합니다.

전처리기는 키네마틱스 루틴이 의미 있는 이름을 사용할 수 있도록 앞서 설명된 **open forward** 설명의 #define 치환을 자동으로 제공합니다.

전처리기는 로컬 변수의 스택 오프셋 크기와 행 점프 레이블 수를 루틴 내용에 근거하여 자동으로 지정합니다.

예

이 간단한 예에서는 #3->1000X 및 #4->1000Y에 해당하는 설정으로 좌표계 2의 인버스 키네마틱스 루틴을 나타냅니다.

```
open inverse (2)
KinPosMotor3 = 1000 * KinPosAxisX;
KinPosMotor4 = 1000 * KinPosAxisY;
if (KinVelEna) {                // Velocity calcs needed (non-seg PVT)?
    KinVelMotor3 = 1000 * KinVelAxisX;
    KinVelMotor4 = 1000 * KinVelAxisY;
}
close
```

open plc {식별자}

기능: 입력하는 PLC 프로그램의 버퍼를 오픈함

범위: 글로벌

구문: **open plc {식별자}**

여기에서

- **{식별자}**는 사용자 프로그램명을 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다. 0~31 범위의 정수 상수 또는 이러한 상수 하나에 할당된 식별자인 경우도 있습니다.

open plc 명령은 Power PMAC 스크립트 PLC 프로그램 32 개 중 하나의 프로그램 버퍼를 오픈합니다. 식별자가 0~31 범위에 있는 상수 또는 이러한 상수값 하나에 할당된 텍스트 문자열인 경우 오픈되는 버퍼는 그 값에 따라 직접 지정됩니다(이 버퍼가 자동으로 할당된 식별자에 이미 사용된 경우에는 다운로드 에러가 표시됩니다).

식별자가 이때까지 상수값에 할당되지 않은 텍스트 문자열인 경우에 1~31 중에서 가장 작은 번호가 있으면서 사용되지 않은 PLC 프로그램 버퍼가 선택됩니다. 이것은 식별자가 RtPlc 가 아닌 경우이며 식별자가 RtPlc 인 경우에는 PLC 0 프로그램 버퍼가 선택됩니다. 식별자는 대문자와 소문자를 구분합니다.

전처리기는 로컬 변수의 스택 오프셋 크기와 행 점프 레이블 수를 프로그램 내용에 근거하여 자동으로 지정합니다.

IDE 에디터는 **plc** 코드 스니펫 지원을 제공합니다. **plc** 인텔리센스 목록에서 [TAB] 키를 누릅니다. 하나 이상의 파라미터 입력이 있는 경우에는 [TAB] 키로 이동합니다. [ENTER] 키를 누르면 스니펫이 종료되고 다른 명령을 입력할 수 있습니다. 마우스 포인터를 파라미터 위에 갖다 대면 입력과 관련된 정보가 표시됩니다. **plc** 명령 위에 마우스 포인터를 갖다 대면 명령과 인수 구문의 설명이 표시됩니다.

예

```
open plc Coolant_Control      // Auto-assigned to next available buffer
enable plc Coolant_Control
```

open prog {식별자}

기능: 입력하는 모션 프로그램의 버퍼를 오픈함

범위: 글로벌

구문: **open prog {식별자}**

여기에서

- **{식별자}**는 사용자 프로그램명을 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다. 0~2³²-1 범위의 정수 상수 또는 이러한 상수 하나에 할당된 식별자인 경우도 있습니다.

open prog 명령은 Power PMAC 스크립트 모션 프로그램 1,024 개 중 하나의 프로그램 버퍼를 엽니다. 식별자가 $0 \sim 2^{32}-1$ 범위에 있는 상수 또는 이러한 상수값 하나에 할당된 텍스트 문자열인 경우에 오픈되는 버퍼는 그 값에 따라 직접 지정됩니다(이 버퍼가 자동으로 할당된 식별자에 이미 사용된 경우에는 다운로드 에러가 표시됩니다).

식별자가 이때까지 상수값에 할당되지 않은 텍스트 문자열인 경우에는 사용되지 않은 100,000 이상의 가장 작은 번호가 있는 모션 프로그램 버퍼가 선택됩니다. 식별자는 대문자와 소문자를 구분합니다.

전처리기는 로컬 변수의 스택 오프셋 크기와 행 점프 레이블 수를 프로그램 내용에 근거하여 자동으로 지정합니다.

IDE 에디터는 **prog** 코드 스니펫 지원을 제공합니다. **prog** 인텔리센스 목록에서 [TAB] 키를 누릅니다. 하나 이상의 파라미터 입력이 있는 경우에는 [TAB] 키로 이동합니다. [ENTER] 키를 누르면 스니펫이 종료되고 다른 명령을 입력할 수 있습니다. 마우스 포인터를 파라미터 위에 갖다 대면 입력과 관련된 정보가 표시됩니다. **prog** 명령 위에 마우스 포인터를 갖다 대면 명령과 인수 구문의 설명이 표시됩니다.

예

```
open prog CalibrationSequence // Auto-assigned to next available buffer

&1 start CalibrationSequence // Command to start execution
```

open subprog {식별자}

기능: 입력하는 서브 프로그램의 버퍼를 엽니다

범위: 글로벌

구문: **open subprog {식별자} [{call_list}]**

여기에서

- **{식별자}**는 사용자 서브 프로그램명을 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _ (스페이스 없음)가 이어집니다. $0 \sim 2^{32}-1$ 범위의 정수 상수 또는 이러한 상수 하나에 할당된 식별자인 경우도 있습니다.
- 옵션 **{call_list}**는 서브 프로그램과 주고받는 인수 식별자 세트입니다.

open subprog 명령은 Power PMAC 스크립트 서브 프로그램 1,024 개 중 하나의 프로그램 버퍼를 엽니다. 식별자가 $0 \sim 2^{32}-1$ 범위에 있는 상수 또는 이러한 상수값 하나에 할당된 텍스트 문자열인 경우에 오픈되는 버퍼는 그 값에 따라 직접 지정됩니다(이 버퍼가 자동으로 할당된 식별자에 이미 사용된 경우에는 다운로드 에러가 표시됩니다). 서브 프로그램은 고레벨의 프로그램에서 **call** 명령을 사용하여 실행됩니다.

식별자가 이때까지 상수값에 할당되지 않은 텍스트 문자열인 경우에는 사용되지 않은 100,000 이상의 가장 작은 번호가 있는 서브 프로그램 버퍼가 선택됩니다. 식별자는 대문자와 소문자를 구분합니다.

옵션인 호출 목록에 따라 로컬 변수 스택의 서브 프로그램과 인수를 주고받을 수 있게 됩니다. 스택 인수를 전달하지 않는 경우에는 목록을 삭제할 수 있습니다. 또는 빈 목록(**void**)을 사용할 수 있습니다. 호출 목록에서는 32 까지의 식별자를 사용할 수 있습니다.

서브 프로그램과의 사이에서 인수를 주고받는 경우에는 호출 목록으로 인수를 지정합니다. 서브 프로그램으로 전달하는 인수의 앞에는 & 기호가 없습니다. 호출 프로그램으로 반환하는 인수의 앞에는 & 기호가 있습니다.

호출 목록 항목이 구문 **{식별자}({array_size})**로 지정된 배열이 될 수 있습니다. 배열 크기는 1~256 정수 범위의 상수 또는 이 범위의 정수에 할당된 식별자입니다. 목록 전체 중에서 최대 256의 합계 개별값을 전달할 수 있습니다.

전처리기는 로컬 변수의 스택 오프셋 크기와 행 점프 레이블 수를 프로그램 내용에 근거하여 자동으로 지정합니다.

IDE 에디터는 **subprog** 코드 스니펫 지원을 제공합니다. **subprog** 인텔리센스 목록에서 [TAB] 키를 누릅니다. 하나 이상의 파라미터 입력이 있는 경우에는 [TAB] 키로 이동합니다. [ENTER] 키를 누르면 스니펫이 종료되고 다른 명령을 입력할 수 있습니다. 마우스 포인터를 파라미터 위에 갖다 대면 입력과 관련된 정보가 표시됩니다. **subprog** 명령 위에 마우스 포인터를 갖다 대면 명령과 인수 구문의 설명이 표시됩니다.

예

```
open subprog Pythag (Adjacent, Opposite, &Hypotenuse)
Hypotenuse = sqrt(Adjacent*Adjacent + Opposite*Opposite);
return;
close

open subprog SquareArray (InputArray(20), &ReturnArray(20))
local Index = 0;
do {
    ReturnArray(Index) = InputArray(Index) * InputArray(Index);
    Index++;
} while (Index < 20);
return;
close
```

sub:{식별자}

기능: 프로그램 또는 서브 프로그램 내에서 서브루틴을 선언함

범위: 로컬에서 스크립트 프로그램

구문: **sub:{식별자} [{sublabel_call_list}]**

여기에서

- **{식별자}**는 사용자 서브루틴명을 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다.
- 옵션 **{sublabel_call_list}**는 서브루틴과 주고받는 인수 식별자 세트입니다.

sub 선언은 스크립트 프로그램 또는 서브 프로그램(**prog**, **plc**, **subprog**, **forward**, **inverse**) 내에서 시작합니다. 식별자는 할당되지 않은 10,000 이상에서 숫자가 가장 작은 점프 레이블에 자동으로 할당됩니다. 식별자는 대문자와 소문자를 구분합니다. 서브루틴은 같은 프로그램에서 **callsub** 명령을 사용하여 실행됩니다.

옵션인 호출 목록에 따라 로컬 변수 스택의 서브 프로그램과 인수를 주고받을 수 있게 됩니다. 스택 인수를 전달하지 않는 경우에는 목록을 삭제할 수 있습니다. 또는 빈 목록(**void**)을 사용할 수 있습니다. 호출 목록에서는 32 까지의 식별자를 사용할 수 있습니다.

서브 프로그램과의 사이에서 인수를 주고받는 경우에는 호출 목록으로 인수를 지정합니다. 서브 프로그램으로 전달하는 인수의 앞에는 & 기호가 없습니다. 호출 프로그램으로 반환하는 인수의 앞에는 & 기호가 있습니다.

호출 목록 항목이 구문 **{식별자}({array_size})**로 지정된 배열이 될 수 있습니다. 배열 크기는 1~256 정수 범위의 상수 또는 이 범위의 정수에 할당된 식별자입니다. 목록 전체 중에서 최대 256의 합계 개별값을 전달할 수 있습니다.

IDE 에디터는 **sub** 코드 스니펫 지원을 제공합니다. **sub** 인텔리센스 목록에서 [TAB] 키를 누릅니다. 하나 이상의 파라미터 입력이 있는 경우에는 [TAB] 키로 이동합니다. [ENTER] 키를 누르면 스니펫이 종료되고 다른 명령을 입력할 수 있습니다. 마우스 포인터를 파라미터 위에 갖다 대면 입력과 관련된 정보가 표시됩니다. **sub** 명령 위에 마우스 포인터를 갖다 대면 명령과 인수 구문의 설명이 표시됩니다.

예

```
open plc MyPlc                                // Top-level PLC program
local MyEndTime;                              // Local to top-level section
local MyList(32);
...
callsub sub.GetEndTime(1000,&MyEndTime);
callsub sub.DelayOneSec();
callsub ShiftList(MyList(32),&MyList(32));
return;                                        // End of top-level section

sub:GetEndTime(TimeInc,&NewTime);              // Subroutine within program
  local StartTime;                            // Local to subroutine
  StartTime = Sys.Time;
  NewTime = StartTime + TimeInc;              // Result in returned variable
return;                                       // End of subroutine
sub:DelayOneSec(void)                         // Subroutine within program
  local DelayTime = Sys.Time + 1;            // Declare and set variable
  while (Sys.Time < DelayTime) {}
return;                                       // End of subroutine
sub:ShiftList(InList(32),&OutList(32));
  local Index = 0;
  do {
    OutList(Index) = InList(Index+1);
    Index++;
  }
while (Index < 32);
OutList(32) = 0;
return;
close
```

서브 프로그램 및 서브루틴 호출 명령

전처리기는 Power PMAC 의 서브 프로그램과 서브루틴을 호출하는 스크립트 프로그램 명령의 실용성을 대폭 개선하고 이러한 인수를 단순하고도 직관적으로 루틴에 전달하고 루틴에서 반환되도록 합니다.

call {식별자}

기능: 서브 프로그램으로 점프하고 복귀함

구문: **call {식별자} [{call_list}]**

call {식별자}.{서브 레이블} [{sublabel_call_list}]

여기에서

- **{식별자}**는 사용자 서브 프로그램명을 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다. 0~2³²-1 범위의 정수 상수 또는 이러한 상수 하나에 할당된 식별자인 경우도 있습니다.
- 옵션 **{call_list}**는 서브 프로그램과 주고받는 인수 식별자 세트입니다.
- **{서브 레이블}**은 서브 프로그램 내의 서브루틴이 선언된 이름을 나타내는 텍스트 문자열입니다.
- 옵션 **{sublabel_call_list}**는 서브 프로그램 내(서브 프로그램의 시작 부분에 대해 호출 목록과 다른 경우가 있습니다) 서브루틴과 주고받는 인수의 식별자 세트입니다.

call 명령에 따라 프로그램 실행은 식별자가 지정하는 서브 프로그램(subprog)으로 점프합니다. 이 식별자는 서브 프로그램의 기본적인 번호이지만 전처리를 사용하여 선언된 서브 프로그램의 사용자명인 경우도 있습니다. 식별자는 대문자와 소문자를 구분합니다.

옵션인 호출 목록에 따라 로컬 변수 스택의 서브 프로그램과 인수를 주고받을 수 있게 됩니다. 호출 목록은 서브 프로그램에 선언된 호출 목록과 일치해야 합니다. 스택 인수를 전달하지 않는 경우에는 목록을 삭제해야 합니다. 또는 빈 괄호 ()를 사용할 수 있습니다.

서브 프로그램으로 전달된 인수의 앞에는 & 기호가 없습니다. **call** 명령에서(서브 프로그램 선언과는 달리) 서브 프로그램으로 전달된 인수는 변수가 아니라 상수 또는 식입니다. 프로그램에서 반환된 인수의 앞에는 & 기호가 있습니다. 전달된 인수는 호출 명령과 서브 프로그램 정의 둘 다에서 변수이어야 합니다.

호출 목록 항목이 구문 **{식별자}({start_index})**로 지정된 배열이 될 수 있습니다. 시작 인덱스는 음의 값이 아닌 정수값입니다. 전달된 배열의 길이는 서브 프로그램의 정의에서 지정합니다. 호출 명령의 인덱스는 사용하는 배열의 시작 요소를 지정합니다.

호출 명령으로 서브 레이블이 지정되지 않은 경우에는 지정된 서브 프로그램 처음으로 점프합니다. 서브 레이블이 지정된 경우에는 서브 프로그램 내에 적합한 서브루틴의 처음으로 점프합니다.

IDE 에디터는 **call** 코드 스니펫 지원을 제공합니다. **call** 인텔리센스 목록에서 [TAB] 키를 누릅니다. 하나 이상의 파라미터 입력이 있는 경우에는 [TAB] 키로 이동합니다. [ENTER] 키를 누르면 스니펫이 종료되고 다른 명령을 입력할 수 있습니다. 마우스 포인터를 파라미터 위에 갖다 대면 입력과 관련된 정보가 표시됩니다. **call** 명령 위에 마우스 포인터를 갖다 대면 명령과 인수 구문의 설명이 표시됩니다.

예

```
// Call commands that use example subprogs at "open subprog"
call Pythag (Xvel, Yvel, &VecVel);
call SquareArray (RawData(10), &ProcData(10));

// Call commands that use subprogs shown below
call GetTime(&LoggedStartTime);
call GetTime.GetEndTime(60,&ProcEndTime);
call GetTime.DelayTimeSec(WaitTime);
call GetTime.DelayOneSec();

// Subprogram with subroutines
open subprog GetTime(&ReturnTime)    // Start of subprogram
    ReturnTime = Sys.Time;
return;                             // Jump back to calling routine
sub:GetEndTime(TimeInc,&NewTime)    // Start of subroutine within
    NewTime = Sys.Time + TimeInc;
return;                             // Jump back to calling routine
sub:DelayTimeSec(DelayTime)        // Start of subroutine within
    local Timer = Sys.Time + DelayTime
    while (Sys.Time < Timer) {}
return;                             // Jump back to calling routine
sub:DelayOneSec(void)              // Start of subroutine within
    local DelayTime = Sys.Time + 1;
    while (Sys.Time < DelayTime) {}
return;                             // Jump back to calling routine
close
```

callsub sub.{식별자}

기능: 프로그램 내의 서브루틴 레이블로 점프하고 복귀함

구문: **callsub sub.{식별자} [{sublabel_call_list}]**

여기에서

- **{식별자}**는 사용자 서브루틴 레이블을 나타내는 텍스트 문자열입니다. 문자 또는 언더바 _로 시작하며 영숫자와 _(스페이스 없음)가 이어집니다.
- 옵션 **{sublabel_call_list}**는 서브루틴과 주고받는 인수 식별자 세트입니다.

callsub 명령에 따라 프로그램 실행은 식별자가 지정하는 같은 프로그램 내의 서브루틴 레이블로 점프합니다. 식별자는 전처리기에서 선언한 서브루틴의 사용자명이어야 합니다. 식별자는 대문자와 소문자를 구분합니다.

옵션인 서브 레이블 호출 목록에 따라 로컬 변수 스택의 서브루틴과 인수를 주고받을 수 있게 됩니다. 서브 레이블 호출 목록은 서브루틴에 선언된 서브 레이블 호출 목록과 일치해야 합니다. 스택 인수를 전달하지 않는 경우에는 목록을 삭제해야 합니다. 또는 빈 괄호 ()를 사용할 수 있습니다.

서브루틴으로 전달된 인수의 앞에는 & 기호가 없습니다. **callsub** 명령에서(서브 프로그램 선언과는 달리) 서브 프로그램으로 전달된 인수는 변수가 아니라 상수 또는 식입니다. 프로그램에서 반환된 인수의 앞에는 & 기호가 있습니다. 전달된 인수는 호출 명령과 서브 프로그램 정의 둘 다에서 변수이어야 합니다.

호출 목록 항목이 구문 **{식별자}{start_index}**로 지정된 배열이 될 수 있습니다. 시작 인덱스는 음의 값이 아닌 정수값입니다. 전달된 배열의 길이는 서브루틴의 정의에서 지정합니다. 호출 명령의 인덱스는 사용하는 배열의 시작 요소를 지정합니다.

IDE 에디터는 **callsub** 코드 스니펫 지원을 제공합니다. **callsub** 인텔리센스 목록에서 [TAB] 키를 누릅니다. 하나 이상의 파라미터 입력이 있는 경우에는 [TAB] 키로 이동합니다. [ENTER] 키를 누르면 스니펫이 종료되고 다른 명령을 입력할 수 있습니다. 마우스 포인터를 파라미터 위에 갖다 대면 입력과 관련된 정보가 표시됩니다. **callsub** 명령 위에 마우스 포인터를 갖다 대면 명령과 인수 구문의 설명이 표시됩니다.

예

```
open prog Myprog
local LoggedStartTime, ProcEndTime, WaitTime;
callsub sub.GetPresentTime(&LoggedStartTime);
callsub sub.GetEndTime(60,&ProcEndTime);
callsub sub.DelayTimeSec(WaitTime);
callsub sub.DelayOneSec();

return;
// End of main program

// Subroutines within same program
sub:GetPresentTime(&ReturnTime)
    ReturnTime = Sys.Time;
return;
sub:GetEndTime(TimeInc,&NewTime)
    NewTime = Sys.Time + TimeInc;
return;
sub:DelayTimeSec(DelayTime)
    local Timer = Sys.Time + DelayTime;
    while (Sys.Time < Timer) {}
return;
sub:DelayOneSec(void)
    local DelayTime = Sys.Time + 1;
    while (Sys.Time < DelayTime) {}
return;
close
```

POWER PMAC 스크립트 수학 기능 사양

Power PMAC의 스크립트 언어에는 강력하고 유연한 연산자와 함수가 있으며 사용자는 고도의 계산식을 폭넓게 직감적인 방법으로 간단하게 탑재할 수 있습니다. 이 수학적 기능은 모션 프로그램, PLC 프로그램 및 온라인 명령으로 이용할 수 있습니다.

연산자

Power PMAC의 스크립트 언어에는 산술 연산자, 논리 연산자, 표준 대입 연산자, 동기 대입 연산자, 조건 비교 연산자, 논리 연산자와 같은 연산자가 있습니다.

산술 연산자

Power PMAC 스크립트 언어에는 완전한 산술 연산자세트가 있으며 모션 프로그램, PLC 프로그램 및 온라인 명령으로 사용할 수 있습니다.

+

기능: 덧셈

+ 연산자는 선행하는 수치와 후속 수치를 더합니다.

곱셈, 나눗셈, 나머지 및 비트 단위의 AND 연산은 덧셈, 뺄셈, 비트 단위의 OR 연산 및 비트 단위의 배타적 논리합보다 높은 우선도를 가집니다. 같은 우선도를 가지는 연산자는 좌측에서 우측으로 순서대로 실행됩니다.



참고사항

후속 변수 또는 상수의 양의 값을 사용하는 경우(예를 들어 'Var2=+Var1'은 잘못된 것입니다)에 플러스(+) 부호를 단항 연산자로 사용할 수 없습니다. 이 구문은 에러로 거부됩니다.

-

기능: 뺄셈 또는 음수

- 연산자는 선행하는 수치에서 후속 수치를 나눕니다. 직전의 자릿수에 수치가 존재하지 않는 경우에 이 부호는 후속 수치를 부정하는 단항 연산자로 작용합니다('Var2=-Var1' 등).

곱셈, 나눗셈, 나머지 및 비트 단위의 AND 연산은 덧셈, 뺄셈, 비트 단위의 OR 연산 및 비트 단위의 배타적 논리합보다 높은 우선도를 가집니다. 같은 우선도를 가지는 연산자는 좌측에서 우측으로 순서대로 실행됩니다.

*

기능: 곱셈

* 연산자는 선행하는 수치와 후속 수치를 곱합니다.

곱셈, 나눗셈, 나머지 및 비트 단위의 AND 연산은 덧셈, 뺄셈, 비트 단위의 OR 연산 및 비트 단위의 배타적 논리합보다 높은 우선도를 가집니다. 같은 우선도를 가지는 연산자는 좌측에서 우측으로 순서대로 실행됩니다.

/

기능: 나눗셈

/ 연산자는 후속 수치를 사용하여 선행하는 수치를 나눕니다. 나눗셈 연산은(정수값이 사용되는 경우에도) 항상 부동 소수점 연산이 됩니다. 몫은 부동 소수점의 값으로 산출되며 같은 식의 다음 계산에서 그대로 사용됩니다. 그것이 정수 레지스터에 저장된 경우에는 저장 시에 반올림합니다.

곱셈, 나눗셈, 나머지 및 비트 단위의 AND 연산은 덧셈, 뺄셈, 비트 단위의 OR 연산 및 비트 단위의 배타적 논리합보다 높은 우선도를 가집니다. 같은 우선도를 가지는 연산자는 좌측에서 우측으로 순서대로 실행됩니다.

제수가 0 인 경우에 결과는 '∞ 무한대(inf)'의 특수한 부동 소수점 표시가 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 몫에 이어지는 수학 연산에서는 예측 가능한 결과를 얻을 수 없습니다. 제로 나눗셈에 의한 에러 가능성은 프로그래머가 확인합니다.

예

연산	결과
$10 * 2 / 3$	6.666666667
$10 * (2 / 3)$	6.666666667
$10 / 0$	inf
$-10 / 0$	-inf

%

기능: 모듈로(나머지)

% 연산자는 선행하는 수치를 후속 수치로 나눈 경우에 발생하는 나머지를 계산합니다. 나눗셈 연산은(정수값이 사용되는 경우에도) 항상 부동 소수점 연산이 됩니다. 몫은 부동 소수점의 값으로 계산된 뒤에 나머지를 계산할 수 있도록 다음의 최소(예를 들어 0 으로 향하는) 정수로 버립니다.

피제수 'n'이 양의 값인 경우에 모듈로의 결과는 $\{0 \leq \text{결과} < n\}$ 의 범위가 됩니다. 피제수 'n'이 음의 값인 경우에 모듈로의 결과는 $\{-n < \text{결과} \leq 0\}$ 의 범위가 됩니다.

음의 피제수 및 제수를 사용한 나머지 연산자의 연산은 Power PMAC에서는 종래의 PMAC 및 Turbo PMAC 과 다르니 주의하십시오. 일반 사용자는 음의 제수로 사용하는 경우에 이전 나머지 연산자의 연산을 에뮬레이트하는 **rem**(나머지) 함수를 이용할 가능성이 있습니다.

곱셈, 나눗셈, 나머지 및 비트 단위의 AND 연산은 덧셈, 뺄셈, 비트 단위의 OR 연산 및 비트 단위의 배타적 논리합 연산보다 높은 우선도를 가집니다. 같은 우선도를 가지는 연산자는 좌측에서 우측으로 순서대로 실행됩니다.

제수가 0 인 경우에 나눗셈은 포화하고 모듈로 결과는 특수한 부동 소수점 표기의 NaN(숫자가 아님)이 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 뒤에 이어지는 수학 연산에서는 예측 가능한 결과를 얻을 수 없습니다. 제로 나눗셈에 의한 에러 가능성은 프로그래머가 확인합니다.

예

연산	결과
11 % 4	3
-11 % 4	-3
11 % -4	3
-11 % -4	-3
3 % 2.5	0.5
-3 % 2.5	-0.5
3 % -2.5	0.5
-3 % -2.5	-0.5
3 % 0	NaN

비트 단위의 논리 연산자

Power PMAC 스크립트 언어에는 비트마다 연산하는 다음 3 개의 논리 연산자가 있습니다. AND 연산자, OR 연산자 및 XOR 연산자입니다. 이 연산자는 모션 프로그램, PLC 프로그램 및 온라인 명령으로 이용할 수 있습니다.

&

기능: 비트 단위의 논리곱

& 연산자는 비트마다 선행하는 수치와 후속 수치의 논리곱 연산을 실행합니다. 양쪽 피연산자가 일치하는 비트가 1 과 같은 경우에만 결과의 소정 비트가 1 이 됩니다. 연산은 정수 비트와 소수 비트(존재하는 경우)의 양쪽 모두에 실행됩니다.

곱셈, 나눗셈, 나머지 및 비트 단위의 AND 연산은 덧셈, 뺄셈, 비트 단위의 OR 연산 및 비트 단위의 배타적 논리합 연산보다 높은 우선도를 가집니다. 같은 우선도를 가지는 연산자는 좌측에서 우측으로 순서대로 실행됩니다.

수치 비트를 논리적으로 조합하는 이 비트 단위의 AND 연산자를 논리적으로 조건을 조합하는 && 연산자와 혼동하지 마십시오.

예

연산	결과
3 & 1	1
3 & 2	2
3 & 3	3
3 & 4	0
\$AA & \$F	\$A(10)
3 & -3	1
-3 & -2	-4
0.875 & 1.75	0.75
0.875 & -1.75	0.25

|

기능: 비트 단위의 논리합

| 연산자는 비트마다 선행하는 수치와 후속 수치의 논리합 연산을 실행합니다. 한쪽의 연산자가 일치하는 비트가 1 과 같은 경우에 결과의 소정 비트가 1 이 됩니다. 연산은 정수 비트와 소수 비트(존재하는 경우)의 양쪽 모두에 실행됩니다.

곱셈, 나눗셈, 나머지 및 비트 단위의 AND 연산은 덧셈, 뺄셈, 비트 단위의 OR 연산 및 비트 단위의 배타적 논리합 연산보다 높은 우선도를 가집니다. 같은 우선도를 가지는 연산자는 좌측에서 우측으로 순서대로 실행됩니다.

수치 비트를 논리적으로 조합하는 이 비트 단위의 OR 연산자를 논리적으로 조건을 조합하는 || 연산자와 혼동하지 마십시오.

예

연산	결과
4 3	7
3 2	3
3 3	3
\$F0 \$4	\$F4(244)
3 -3	-1
-3 -2	-1
0.5 0.375	0.875
0.875 -1.75	-0.375

^

기능: 비트 단위의 배타적 논리합

^ 부호는 비트마다 선행하는 수치와 후속 수치의 배타적 논리합 연산(XOR)을 실행합니다. 2개의 연산자가 일치하는 비트가 서로 다른 경우에만 결과의 소정 비트가 1이 됩니다. 연산은 정수 비트와 소수 비트(존재하는 경우)의 양쪽 모두에 실행됩니다.

곱셈, 나눗셈, 나머지 및 비트 단위의 AND 연산은 덧셈, 뺄셈, 비트 단위의 OR 연산 및 비트 단위의 배타적 논리합 연산보다 높은 우선도를 가집니다. 같은 우선도를 가지는 연산자는 좌측에서 우측으로 순서대로 실행됩니다.

예

연산	결과
2 ^ 1	3
2 ^ 2	0
5 ^ 7	2
\$AA ^ \$55	\$FF
3 ^ -3	-2
0.5 ^ 0.875	0.375

<<

기능: 좌측 시프트

<< 연산자는 좌측 시프트 연산하여 연산자 앞의 값을 연산자 뒤의 값으로 지정된 비트 수만큼 시프트합니다. 이것은 사실상 2^n 의 곱셈으로 여기서 n 은 연산자의 후속 값입니다.

연산자 선행의 값과 연산자의 후속 값 양쪽이 부동 소수점의 값이 될 가능성이 있으며 양쪽이 유효한 연산을 위해서 정수여야 할 필요는 없습니다.

이 연산자는 곱셈이나 나눗셈과 같은 우선순위입니다.

>>

기능: 우측 시프트

>> 연산자는 우측 시프트 연산하여 연산자 앞의 값을 연산자 뒤의 값으로 지정된 비트 수만큼 시프트합니다. 이것은 사실상 2^n 의 나눗셈으로 여기서 n 은 연산자 앞의 값입니다.

연산자 선행의 값과 연산자의 후속 값 양쪽이 부동 소수점의 값이 될 가능성이 있으며 양쪽이 유효한 연산을 위해서 정수여야 할 필요는 없습니다.

이 연산자는 곱셈이나 나눗셈과 같은 우선순위입니다.

표준 대입 연산자

Power PMAC의 스크립트 언어에는 큰 대입 연산자 세트가 있습니다. 이것으로 Power PMAC은 값을 명령 내의 연산자 직전에 있는 변수에 씁니다. 다수의 대입 연산자는 수학 연산이나 논리 연산과 대입을 결합하여 소형으로 효율적인 코드를 할 수 있습니다. 이 연산자는 모션 프로그램 및 PLC 프로그램에서 사용할 수 있습니다. 온라인 명령에서 '='의 기본적인 대입 연산자만 사용할 수 있습니다.

이 대입 연산자 세트는 표준형이며 보통 프로그램 플로우 값이 계산되면 즉시 변수에 씁니다. 이 연산자는 동기 대입 연산자와 다릅니다. 동기인 경우에 실제 수치 대입은 다음의 프로그램 동작의 실제 실행 시작까지 지연됩니다.

=

기능: 기본적인 대입 연산

구문: **{변수} = {식}**

=의 대입 연산자는 연산자의 다음 수식을 평가하고 그 값을 연산자 앞에 지정된 변수에 대입합니다.

+=

기능: 덧셈과 대입

구문: **{변수} += {식}**

+=의 대입 연산자는 연산자의 다음 수식을 평가하고 그 값을 연산자 앞에 지정된 변수의 현재 값에 더합니다. 또 결과 값을 이 변수에 대입합니다.

-=

기능: 뺄셈과 대입

구문: **{변수} -= {식}**

-=의 대입 연산자는 연산자의 다음 수식을 평가하고 그 값을 연산자 앞에 지정된 변수의 현재 값에서 뺍니다. 또 결과 값을 이 변수에 대입합니다.

***=**

기능: 곱셈과 대입

구문: **{변수} *= {식}**

***=**의 대입 연산자는 연산자의 다음 수식을 평가하고 그 값을 연산자 앞에 지정된 변수의 현재 값에 곱합니다. 또 결과 값을 이 변수에 대입합니다.

/=

기능: 나눗셈과 대입

구문: **{변수} /= {식}**

/=의 대입 연산자는 연산자의 다음 수식을 평가하고 그 값으로 연산자 앞에 지정된 변수의 현재 값을 나눕니다. 또 결과 값을 이 변수에 대입합니다.

제수가 0 인 경우에 결과는 '± 무한대(inf)'의 특수한 부동 소수점 표시가 됩니다. 그 경우에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 뒤에 이어지는 수학 연산에서는 예측 가능한 결과를 얻을 수 없습니다. 제로 나눗셈에 의한 에러 가능성은 프로그래머가 확인합니다.

%=

기능: 모듈로 연산과 대입

구문: **{변수} %= {식}**

%=의 대입 연산자는 연산자의 다음 수식을 평가하고 그 값으로 연산자 앞에 지정된 변수의 현재 값을 나눕니다. 또 이 나눗셈의 나머지와 결과 값을 이 변수에 대입합니다.

제수가 0 인 경우에 나눗셈은 포화하고 모듈로 결과는 특수한 부동 소수점 표기의 NaN(숫자가 아님)이 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 뒤에 이어지는 수학 연산에서는 예측 가능한 결과를 얻을 수 없습니다. 제로 나눗셈에 의한 에러 가능성은 프로그래머가 확인합니다.

&=

기능: 비트 단위의 논리곱 대입

구문: **{변수} &= {식}**

&=의 대입 연산자는 연산자의 다음 수식을 평가하고 그 값과 연산자 앞에 지정된 변수의 현재 값으로 논리곱 연산을 실행합니다. 결과 값을 이 변수에 대입합니다.

|=

기능: 비트 단위의 논리합 대입

구문: **{변수} |= {식}**

|=의 대입 연산자는 연산자의 다음 수식을 평가하고 그 값과 연산자 앞에 지정된 변수의 현재 값으로 논리합 연산을 실행합니다. 또 결과 값을 이 변수에 대입합니다.

`^=`

기능: 비트 단위의 배타적 논리합 대입

구문: **`{변수} ^= {식}`**

`^=`의 대입 연산자는 연산자의 다음 수식을 평가하고 그 값과 연산자 앞에 지정된 변수의 현재 값으로 배타적 논리합 연산을 실행합니다. 또 결과 값을 이 변수에 대입합니다.

`>>=`

기능: 우측 시프트 연산과 대입

구문: **`{변수} >>= {식}`**

`>>=`의 대입 연산자는 연산자의 다음 수식을 평가하고 그 값으로 연산자 앞에 지정된 변수의 현재 값을 우측 시프트하는 비트 수를 지정합니다. 또 결과 값을 이 변수에 대입합니다.

`<<=`

기능: 좌측 시프트 연산과 대입

구문: **`{변수} <<= {식}`**

`<<=`의 대입 연산자는 연산자의 다음 수식을 평가하고 그 값으로 연산자 앞에 지정된 변수의 현재 값을 좌측 시프트하는 비트 수를 지정합니다. 또 결과 값을 이 변수에 대입합니다.

`++`

기능: 증가 대입

구문: **`{변수}++`**

`++` 대입 연산자는 선행하는 변수값에 1.0 을 더합니다. 이것은 더 복잡한 식의 일부로 사용할 수 없습니다. 변수는 증가분의 올바른 연산을 위해서 정수값일 필요는 없습니다.

`--`

기능: 감소 대입

구문: **`{변수}--`**

`--` 대입 연산자는 선행하는 변수값에서 1.0 을 뺍니다. 이것은 더 복잡한 식의 일부로 사용할 수 없습니다. 변수는 감소분의 올바른 연산을 위해서 정수값일 필요는 없습니다.

동기 대입 연산자

동기 대입 연산자는 대응하는 명령 동작이 실제로 실행되기 훨씬 전에 모션 프로그램 내의 명령문의 평가가 필요한 블렌드 동작 시퀀스에서 사용합니다. 표준 대입 연산자를 사용하면 실제 동작을 실행하는 시퀀스 이외의 변수에 값을 대입할 수 있습니다. 이는 동작 방정식의 계산에서 사용되는 변수에 대해서는 적합하지만 물리적인 출력 값을 설정하려는 목적으로 해당 표준 대입이 사용된 경우에 출력은 관련하는 동작 시퀀스에서 제외됩니다.

동기 대입 연산자는 명령된 다음 동작의 실제 실행이 시작될 때까지 평가한 식의 실제 대입을 늦춰서 이 문제를 해결합니다. 동기 대입 연산자 뒤의 식은 프로그램 계산 플로의 읽기 때에 평가되지만 연산 결과는 큐에 넣어서 프로그램 내에서 명령된 다음 동작의 실제 실행이 시작될 때까지 실제로는 실행되지 않습니다.

또 동기 대입 연산자는 축 또는 모터의 동작 명령으로 명령된 동작이 실제로 시작된 것을 확인하는 PLC 프로그램에서 유용합니다.

동기 대입 연산자는 로컬 변수나 자기 정의의 'M' 변수의 대입에는 사용할 수 없습니다. 이 연산자는 온라인 명령으로는 사용할 수 없습니다.

==

기능: 표준 동기 대입

구문: **{변수} == {식}**

==의 동기 대입 연산자는 연산자의 다음 수식을 평가하고 결과 값을 출력 큐에 넣습니다. 모션 프로그램 내의 다음 동작이 실제로 실행을 시작한 시점에서 연산자 앞의 지정된 변수에 대입합니다.

조건문 중의 **==**기호는 평등 비교 연산자입니다.

+=

기능: 덧셈과 동기 대입

구문: **{변수} += {식}**

+=의 동기 대입 연산자는 연산자의 다음 수식을 평가하고 결과 값을 출력 큐에 넣습니다. 모션 프로그램 내의 다음 동작이 실제로 실행을 시작한 시점에서 연산자 앞의 지정된 변수의 현재 값에 더합니다. 또 결과 값을 이 변수에 대입합니다.

-==

기능: 뺄셈과 동기 대입

구문: **{변수} -= {식}**

-==의 대입 연산자는 연산자의 다음 수식을 평가하고 결과 값을 출력 큐에 넣습니다. 모션 프로그램 내의 다음 동작이 실제로 실행을 시작한 시점에서 연산자 앞의 지정된 변수의 현재 값에서 뺍니다. 또 결과 값을 이 변수에 대입합니다.

***==**

기능: 곱셈과 동기 대입

구문: **{변수} *= {식}**

***==**의 대입 연산자는 연산자의 다음 수식을 평가하고 결과 값을 출력 큐에 넣습니다. 모션 프로그램 내의 다음 동작이 실제로 실행을 시작한 시점에서 연산자 앞의 지정된 변수의 현재 값에 곱합니다. 또 결과 값을 이 변수에 대입합니다. 변수는 정수가 아니라 부동 소수점 형식이어야 합니다.

/==

기능: 나눗셈과 동기 대입

구문: **{변수} /= {식}**

/==의 대입 연산자는 연산자의 다음 수식을 평가하고 결과 값을 출력 큐에 넣습니다. 모션 프로그램 내의 다음 동작이 실제로 실행을 시작한 시점에서 연산자 앞의 지정된 변수의 현재 값을 나눕니다. 또 결과 값을 이 변수에 대입합니다. 변수는 정수가 아니라 부동 소수점 형식이어야 합니다.

제수가 0 인 경우에 결과는 '± 무한대(inf)'의 특수한 부동 소수점 표시가 됩니다. 그 경우에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 묶에 이어지는 수학 연산에서는 예측 가능한 결과를 얻을 수 없습니다. 제로 나눗셈에 의한 에러 가능성은 프로그래머가 확인합니다.

++=

기능: 동기 증가 대입

구문: **{변수} +=**

++=의 대입 연산자는 선행하는 변수값에 1.0 을 더하고 결과 값을 큐에 넣습니다. 프로그램된 다음 동작이 실제로 실행을 시작한 시점에서 변수에 대입합니다. 이것은 더 복잡한 식의 일부로 사용할 수 없습니다. 변수는 증가분의 올바른 연산을 위해서 정수값일 필요는 없습니다.

--=

기능: 동기 감소 대입

구문: **{변수}--=**

--=의 대입 연산자는 선행하는 변수값에서 1.0 을 빼고 결과 값을 큐에 넣습니다. 프로그램된 다음 동작이 실제로 실행을 시작한 시점에서 변수에 대입합니다. 이것은 더 복잡한 식의 일부로 사용할 수 없습니다. 변수는 감소분의 올바른 연산을 위해서 정수값일 필요는 없습니다.

&==

기능: 비트 단위의 논리곱연산과 동기 대입

구문: **{변수} &== {식}**

&==의 대입 연산자는 연산자의 다음 수식을 평가하고 결과 값을 출력 큐에 넣습니다. 또 비트 단위의 논리곱 연산을 사용한 논리적인 조합의 경우에 모션 프로그램 내의 다음 동작이 실제로 실행을 시작한 시점에서 연산자 앞의 지정된 변수의 현재 값을 사용하여 결과 값을 이 변수에 대입합니다. 변수는 부동 소수점의 값이 아니라 정수 형식이어야 합니다.

|==

기능: 비트 단위의 논리합 연산과 동기 대입

구문: **{변수} |== {식}**

|==의 대입 연산자는 연산자의 다음 수식을 평가하고 결과 값을 출력 큐에 넣습니다. 또 비트 단위의 논리합 연산을 사용한 논리적인 조합의 경우에 모션 프로그램 내의 다음 동작이 실제로 실행을 시작한 시점에서 연산자 앞의 지정된 변수의 현재 값을 사용하여 결과 값을 이 변수에 대입합니다. 변수는 부동 소수점의 값이 아니라 정수 형식이어야 합니다.

^==

기능: 비트 단위의 배타적 논리합 연산과 동기 대입

구문: **{변수} ^== {식}**

^==의 대입 연산자는 연산자의 다음 수식을 평가하고 결과 값을 출력 큐에 넣습니다. 또 비트 단위의 배타적 논리합 연산(XOR)을 사용한 논리적인 조합의 경우에 모션 프로그램 내의 다음 동작이 실제로 실행을 시작한 시점에서 연산자 앞의 지정된 변수의 현재 값을 사용하여 결과 값을 이 변수에 대입합니다. 변수는 부동 소수점의 값이 아니라 정수 형식이어야 합니다.

조건 비교 연산자

Power PMAC 스크립트 언어에는 강력하고 유연한 논리 기능을 제공하는 완전한 세트의 조건 비교 연산자가 있습니다. 비교 결과는 부울 형식의 **True** 또는 **False** 값으로 나타나고 이것은 복합 조건 중에서 다른 부울 형식의 **True** 또는 **False** 값과 논리적으로 조합되지만 일반적인 수식의 일부로 사용되지 않는 경우도 있습니다.

==

기능: 평등 비교 연산자

구문: **{exp1} == {exp2}**

==의 조건 비교 연산자는 연산자 양변의 수식 평가와 비교합니다. 2 개의 값이 정확하게 똑같은 경우에는 부울 형식의 **True** 를 반환하고 이 값이 정확하게 똑같지 않은 경우에는 부울 형식의 **False** 를 반환합니다.

변수 대입 명령 내의 **==** 부호는 동기 대입 연산자입니다.

!=

기능: 불평등 비교 연산자

구문: **{exp1} != {exp2}**

!=의 조건 비교 연산자는 연산자의 양변 수식 평가와 비교합니다. 2 개의 값이 정확하게 똑같지 않은 경우에는 부울 형식의 **True** 를 반환하고 이 값이 정확하게 똑같은 경우에는 부울 형식의 **False** 를 반환합니다.

<>의 구문도 마찬가지로 사용할 수 있지만 프로그램에서는 **!=**의 구문이 사용됩니다.

<

기능: '보다 작은' 비교 연산자

구문: **{exp1} < {exp2}**

<의 조건 비교 연산자는 연산자의 양변 수식 평가와 비교합니다. 최초의 값이 2 번째 값보다 작은 경우(예를 들어 음의 무한대에 가까운 경우)에는 부울 형식의 **True** 를 반환하고 2 개의 값이 정확하게 똑같은 경우 또는 2 번째 값이 최초의 값보다 작은 경우에는 부울 형식의 **False** 를 반환합니다.

>

기능: '보다 큰' 비교 연산자

구문: **$\{exp1\} > \{exp2\}$**

>의 조건 비교 연산자는 연산자의 양변 수식 평가와 비교합니다. 최초의 값이 2 번째 값보다 큰 경우(예를 들어 양의 무한대에 가까운 경우)에는 부울 형식의 True 를 반환하고 2 개의 값이 정확하게 똑같은 경우 또는 2 번째 값이 최초의 값보다 큰 경우에는 부울 형식의 False 를 반환합니다.

<=

기능: '보다 작거나 같은' 비교 연산자

구문: **$\{exp1\} <= \{exp2\}$**

<=의 조건 비교 연산자는 연산자의 양변 수식 평가와 비교합니다. 최초의 값이 2 번째 값보다 작은 경우(예를 들어 음의 무한대에 가까운 경우) 또는 2 개의 값이 정확하게 똑같은 경우에는 부울 형식의 True 을 반환하고 2 번째 값이 최초의 값보다 작은 경우에는 부울 형식의 False 를 반환합니다.

!>의 구문도 마찬가지로 사용할 수 있지만 프로그램에서는 <=의 구문이 사용됩니다.

>=

기능: '보다 크거나 같은' 비교 연산자

구문: **$\{exp1\} >= \{exp2\}$**

>=의 조건 비교 연산자는 연산자의 양변 수식 평가와 비교합니다. 최초의 값이 2 번째 값보다 큰 경우(예를 들어 양의 무한대에 가까운 경우) 또는 2 개의 값이 정확하게 똑같은 경우에는 부울 형식의 True 를 반환하고 2 번째 값이 최초의 값보다 큰 경우에는 부울 형식의 False 를 반환합니다.

!<의 구문도 마찬가지로 사용할 수 있지만 프로그램에서는 >=의 구문이 사용됩니다.

~

기능: 근사 평등 비교 연산자

구문: **$\{exp1\} \sim \{exp2\}$**

~의 조건 비교 연산자는 연산자의 양변 수식 평가와 비교합니다. 2 개의 값의 차가 0.5 미만인 경우에는 부울 형식의 True 를 반환하고 차가 0.5 이상인 경우에는 부울 형식의 False 를 반환합니다.

조건문 외측의 ~ 기호는 비트 단위의 반전 함수이므로 주의하십시오.

!~

기능: 근사 불평등 비교 연산자

구문: **$\{exp1\} !\sim \{exp2\}$**

!~의 조건 비교 연산자는 연산자의 양변 수식 평가와 비교합니다. 2 개의 값의 차가 0.5 이상인 경우에는 부울 형식의 True 를 반환하고 차가 0.5 미만인 경우에는 부울 형식의 False 를 반환합니다.

<>

기능: 불평등 비교 연산자(대체 구문)

구문: **$\{exp1\} <> \{exp2\}$**

{See !=}

<>의 조건 비교 연산자는 !=의 불평등 비교 연산자의 대체 구문입니다. <>는 != 대신에 사용할 수 있지만 !=로 저장되어 통지됩니다.

!>

기능: '보다 작거나 같은' 비교 연산자(대체 구문)

구문: **$\{exp1\} !>\{exp2\}$**

{See <=}

!>의 조건 비교 연산자는 <=의 '보다 작거나 같은' 비교 연산자의 대체 구문입니다. !>는 <= 대신에 사용할 수 있지만 <=로 저장되어 통지됩니다.

!<

기능: '보다 크거나 같은' 비교 연산자(대체 구문)

구문: **$\{exp1\} !< \{exp2\}$**

{See >=}

!<의 조건 비교 연산자는 >=의 '보다 크거나 같은' 비교 연산자의 대체 구문입니다. !<는 >= 대신에 사용할 수 있지만 >=로 저장되어 통지됩니다.

논리 연산자

Power PMAC 은 단순 조건의 부울 형식을 복합 조건으로 조합 가능하게 하는 논리 연산자를 제공합니다.

&&

기능: 논리곱 연산

구문: **{조건 1} && {조건 2}**

&&의 논리 연산자는 연산자 양변의 논리 조건 부울 형식의 결과를 논리곱 연산 내에서 조합합니다. 양쪽의 조건이 True 인 경우에 결과 값은 부울 형식의 True 가 됩니다. 조건 중 하나 또는 둘이 False 인 경우에는 부울 형식의 False 가 됩니다. 조건은 논리 비교 연산자 또는 제로 이외의 값에 대해서 평가한 경우에 True 가 되는 단순한 수식을 사용할 수 있습니다.

||

기능: 논리합 연산

구문: **{조건 1} || {조건 2}**

||의 논리 연산자는 연산자 양변의 논리 조건 부울 형식의 결과를 논리합 연산 내에서 조합합니다. 한쪽의 조건이 True 인 경우에 결과 값은 부울 형식의 True 가 됩니다. 조건인 양쪽이 False 인 경우에는 부울 형식의 False 값이 됩니다. 조건은 논리 비교 연산자를 사용한 명시적인 조건 또는 제로 이외의 값에 대해서 평가한 경우에 True 가 되는 단순한 수식을 사용할 수 있습니다.

!

기능: 논리 부정

구문: **!({조건})**

!의 논리 부정 연산자는 후속하는 괄호 안의 조건 부울 형식의 결과를 부정합니다. 조건이 True 인 경우 또는 식이 제로 이외의 값을 가지는 경우에 결과 값은 False 가 됩니다. 조건이 False 인 경우 또는 식이 정확하게 제로 값을 가지는 경우에 결과 값은 True 가 됩니다. 조건은 논리 비교 연산자를 사용한 명시적인 조건이어야 하며 단순한 수식은 허용되지 않습니다.

함수

Power PMAC 은 스크립트 언어의 부동 소수점 라이브러리의 일부로 다수의 수학 함수를 탑재하고 있습니다. 이 함수는 모션 프로그램, PLC 프로그램 및 온라인 명령으로 이용할 수 있습니다.

스칼라 함수

다음 그룹의 함수는 스칼라량을 연산하고 결과를 반환합니다(반의어는 벡터 또는 매트릭스입니다).

~

기능: 비트 단위의 반전

구문: **~({식})**

도메인: 모든 실수

도메인의 단위: 사용자 정의

범위: 모든 실수

범위의 단위: 사용자 정의

~ 함수는 직후 값인 각 비트를 비트 단위로 반전합니다. 이것은 주로 부호가 없는 정수값의 사용을 목적으로 하지만 다른 타입의 연산에도 사용할 수 있습니다.

조건문 중 ~기호는 근사 평등 비교 연산자입니다.

예

```
M1->*u.1           // Self-referenced 1-bit var
M1=0                // To start
M1=~(M1)            // Invert bit
M1                  // Query value
1                   // Power PMAC response
M1=~(M1)            // Invert bit
M1                  // Query value
0                   // Power PMAC response
M1->*u.4             // Self-referenced 4-bit var
M1=0                // To start
M1=~(M1)            // Invert bits
M1                  // Query value
15                  // Power PMAC response
M1=~(M1)            // Invert bits
M1                  // Query value
0                   // Power PMAC response
```

abs

기능:	절대값
구문:	abs({식})
도메인:	모든 실수
도메인의 단위:	사용자 정의
범위:	음의 값이 아닌 실수
범위의 단위:	사용자 정의

abs 는 후속하는 괄호 안에 포함되는 수식의 절대값(크기)의 함수를 실행합니다.

예

<pre>Var2=abs(Var1) if(Var3 <= 0) SgnVar3=abs(Var3)/Var3 else SgnVar3=0</pre>	<pre>// Computes mag of Var1 // Sign of non-zero num // Sign value is 0</pre>

acos

기능:	삼각 arccos(아크코사인)
구문:	acos({식})
도메인:	-1.0~+1.0
도메인의 단위:	없음
범위:	0~Pi
범위의 단위:	라디안

acos 는 후속하는 괄호 안에 포함되는 수식의 표준 삼각 함수의 arccos(아크코사인) 함수를 실행하여 라디안의 각도를 반환합니다.

결과의 단위가 도수의 arccos 함수인 경우에는 대신에 **acosd** 를 사용하십시오.

괄호 안의 인수가 -1.0~+1.0 의 규정 도메인을 벗어나는 경우에 결과는 NaN(숫자가 아님)의 특수한 부동 소수점 표기가 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

<pre>Angle=acos(CosValue) C(acos(Length/10))</pre>	<pre>// Finds angle whose cos is CosValue // Move C axis to specified angle</pre>

acosd

기능:	삼각 arccos(각도)
구문:	acosd({식})
도메인:	-1.0~+1.0
도메인의 단위:	없음
범위:	0~180
범위의 단위:	도수

acosd 는 후속하는 괄호 안에 포함되는 수식의 삼각 함수의 arccos(아크코사인) 함수를 구하여 각도(도수)를 반환합니다.

결과의 단위가 라디안의 표준 아크코사인 함수인 경우에는 대신에 **acos** 를 사용하십시오.

괄호 안의 인수가 -1.0~+1.0 의 규정 도메인을 벗어나는 경우에 결과는 NaN(숫자가 아님)의 특수한 부동 소수점 표기가 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

Angle=acosd(CosValue)	// Finds angle whose cos is CosValue
C(acosd(Length/10))	// Move C axis to specified angle

acosh

기능:	역쌍곡선 코사인
구문:	acosh({식})
도메인:	양의 실수 > 1.0
도메인의 단위:	없음
범위:	양의 실수
범위의 단위:	라디안

acosh 는 후속하는 괄호 안에 포함되는 수식의 역쌍곡선 코사인 함수를 실행합니다.

괄호 안의 인수가 +1.0~+무한대의 규정 도메인을 벗어나는 경우에 결과는 NaN(숫자가 아님)의 특수한 부동 소수점 표기가 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

Var2=acosh(Var1)

asin

기능: 삼각 arcsin(아크사인)

구문: **asin({식})**

도메인: -1.0~+1.0

도메인의 단위: 없음

범위: $-\pi/2 \sim \pi/2$

범위의 단위: 라디안

asin 은 후속하는 괄호 안에 포함되는 수식의 표준 삼각 함수의 arcsin(아크사인) 함수를 실행하여 라디안의 각도를 반환합니다.

결과물의 단위가 도수의 arcsin 함수인 경우에는 대신에 **asind** 를 사용하십시오.

괄호 안의 인수가 -1.0~+1.0 의 규정 도메인을 벗어나는 경우에 결과는 NaN(숫자가 아님)의 특수한 부동 소수점 표기가 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

Angle=asin(SinValue)	// Finds angle whose sin is SinValue
C(asin(Rise/10))	// Move C axis to specified angle

asind

기능: 삼각 arcsin(단위는 도수)

구문: **asind({식})**

도메인: -1.0~+1.0

도메인의 단위: 없음

범위: 0~180

범위의 단위: 도수

asind 는 후속하는 괄호 안에 포함되는 수식의 삼각 함수의 arcsin(아크사인) 함수를 실행하여 각도(도수)를 반환합니다.

결과물의 단위가 라디안의 표준 arcsin 함수인 경우에는 대신에 **asin** 을 사용하십시오.

괄호 안의 인수가 -1.0~+1.0 의 규정 도메인을 벗어나는 경우에 결과는 NaN(숫자가 아님)의 특수한 부동 소수점 표기가 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

```
Angle=asind(SinValue)      // Finds angle whose sin is SinValue
C(asind(Rise/10))          // Move C axis to specified angle
```

asinh

기능: 역쌍곡선 사인

구문: **asinh({식})**

도메인: 모든 실수

도메인의 단위: 없음

범위: 모든 실수

범위의 단위: 라디안

asinh 은 후속하는 괄호 안에 포함되는 수식의 역쌍곡선 사인 함수를 실행합니다.

예

```
Var2=asinh(Var1)
```

atan

기능: 삼각 arctan(아크탄젠트)

구문: **atan({식})**

도메인: 모든 실수

도메인의 단위: 없음

범위: $-\pi/2 \sim +\pi/2$

범위의 단위: 라디안

atan 은 후속하는 괄호 안에 포함되는 수식의 표준 삼각 함수의 arctan(아크탄젠트) 함수를 실행하여 라디안의 각도를 반환합니다. 이 표준의 arctan 함수는 $\pm\pi/2$ 의 라디안 범위 내에서만 값을 반환합니다. $\pm\pi$ 라디안의 모든 범위가 필요한 경우에는 대신에 **atan2** 함수를 사용하십시오.

결과의 단위가 도수의 arctan 함수인 경우에는 대신에 **atand** 를 사용하십시오.

예

```
Angle=atan(TanValue)      // Finds angle whose tan is TanValue
C(atan(Rise/Run))          // Move C axis to specified angle
```

atan2

기능:	2 인수 삼각 arctan
구문:	atan2 ({식1},{식2})
도메인:	양쪽의 인수 내의 모든 실수(단, 동시에 2 개가 0 이 되지는 않는다)
도메인의 단위:	없음
범위:	-Pi~+Pi
범위의 단위:	라디안

atan2 는 후속 괄호 안에 포함되는 2 개의 수식의 확장(2 인수)의 arctan(아크탄젠트) 함수를 실행하여 라디안의 각도를 반환합니다.

이 확장 arctan 함수는 $\pm\pi$ 라디안의 모든 범위 내에서만 값을 반환합니다. $\pm\pi/2$ 라디안 범위만 필요한 경우에는 표준 **atan** 함수를 대신에 사용하십시오. **atan2** 함수는 양쪽 인수의 부호를 사용하고 또 그 크기의 비율을 사용하여 범위를 전체 360 도로 확장합니다. **atan2** 의 후속 괄호 안의 최초의 값은 sin 의 인수이고 2 번째는 cos 인수입니다. 2 개의 인수 부호와 상대적인 크기만 중요한 점에 주의하십시오. 실제로는 모두 결과 각도의 사인 또는 코사인과 같은 값이 아닌 것이 필요합니다.

결과의 단위가 도수의 2 인수 삼각 아크탄젠트 함수인 경우에는 대신에 **atan2d** 를 사용하십시오.

atan2 함수의 양쪽 인수가 정확하게 0.0 과 같은 경우에 결과는 특수한 부동 소수점 표기의 NaN(숫자가 아님)이 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

```
Angle=atan2(SinAngle,CosAngle)
if(abs(XVel)>0 || abs(YVel)>0) VelAngle=atan2(YVel,XVel)
```

atan2d

기능:	2 인수 삼각 arctan(단위는 도수)
구문:	atan2d ({식1},{식2})
도메인:	양쪽의 인수 내의 모든 실수(단, 동시에 2 개가 0 이 되지는 않는다)
도메인의 단위:	없음
범위:	-180~+180
범위의 단위:	도수

atan2d 는 후속 괄호 안에 포함되는 2 개의 수식 확장(2 인수)의 arctan(아크탄젠트) 함수를 실행하여 각도(도수)를 반환합니다.

이 확장 아크탄젠트 함수는 전체 $\pm 180^\circ$ 의 범위 내에서 값을 반환합니다. $\pm 90^\circ$ 의 범위만 필요한 경우에는 대신에 **atand** 함수를 사용하십시오. **atan2d** 함수는 양쪽 인수의 부호를 사용하고 또 그 크기의 비율을 사용하여 범위를 전체 360 도로 확장합니다. **atan2d** 의 후속 괄호 안의 최초의 값은 sin 의 인수이고 2 번째는 cos 의 인수입니다. 2 개의 인수 부호와 상대적인 크기만 중요한 점에 주의하십시오. 실제로는 모두 결과 각도의 사인 또는 코사인과 같은 값이 아닌 것이 필요합니다.

결과의 단위가 라디안의 4 상한 아크탄젠트 함수인 경우에는 대신에 **atan2** 를 사용하십시오.

atan2d 함수의 양쪽 인수가 정확하게 0.0 과 같은 경우에 결과는 특수한 부동 소수점 표기의 NaN(숫자가 아님)이 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

```
Angle=atan2d(SinAngle,CosAngle)
if(abs(XVel)>0 || abs(YVel)>0) VelAngle=atan2d(YVel,XVel)
```

atand

기능: 삼각 arctan(단위는 도수)

구문: **atand({식})**

도메인: 모든 실수

도메인의 단위: 없음

범위: $-90 \sim +90$

범위의 단위: 도수

atand 는 후속하는 괄호 안에 포함되는 수식의 삼각 함수의 arctan(아크탄젠트) 함수를 실행하여 각도(도수)를 반환합니다.

이 단일 인수의 아크탄젠트 함수는 ± 90 도의 각도 값만 반환합니다. ± 180 도 모든 범위가 필요한 경우에는 대신에 **atan2d** 함수를 사용하십시오.

결과의 단위가 라디안의 4 상한 아크탄젠트 함수인 경우에는 대신에 **atan2** 를 사용하십시오.

예

```
Angle=atand(TanValue)           // Finds angle whose tan is TanValue
C(atand(Rise/Run))              // Move C axis to specified angle
```

atanh

기능:	역쌍곡선 탄젠트
구문:	atanh({식})
도메인:	-1.0~+1.0
도메인의 단위:	없음
범위:	모든 실수
범위의 단위:	라디안

atanh 는 후속하는 괄호 안에 포함되는 수식의 역쌍곡선 탄젠트 함수를 실행합니다.

괄호 안의 인수가 -1.0~+1.0 의 규정 도메인을 벗어나는 경우에 결과는 NaN(숫자가 아님)의 특수한 부동 소수점 표기가 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

Var2=atanh(Var1)

cbrt

기능:	세제곱근
구문:	cbrt({식})
도메인:	모든 실수
도메인의 단위:	사용자 정의
범위:	모든 실수
범위의 단위:	사용자 정의

cbrt 는 후속하는 괄호 안에 포함되는 수식의 실수 세제곱근 함수를 실행합니다.

예

Var1=cbrt(Var2)

ceil

기능: 보다 큰 정수로 반올림함

구문: **ceil({식})**

도메인: 모든 실수

도메인의 단위: 사용자 정의

범위: 모든 정수

범위의 단위: 사용자 정의

ceil 은 후속하는 괄호 안에 포함되는 수식에서 보다 큰 정수에 반올림하는 함수를 실행합니다. 반올림은 항상 정방향으로 실행됩니다.

결과는 정수이지만 부동 소수점의 값으로 나타냅니다.

예

```
Var1=2.5
Var2=ceil(Var1)           // Take ceil of positive value
Var2                      // Query resulting value
3                          // Next higher integer value
Var3=ceil(-Var1)          // Take ceil of negative value
Var3                      // Query resulting value
-2                         // Next higher integer value
```

cos

기능: 삼각 함수의 cos(코사인)

구문: **cos({식})**

도메인: 모든 실수

도메인의 단위: 라디안

범위: -1.0~+1.0

범위의 단위: 없음

cos 은 후속하는 괄호 안에 포함되는 수식(단위는 라디안)의 표준 삼각 함수의 cos 함수를 실행합니다.

인수가 도수 단위의 cos 함수인 경우에는 대신에 **cosd** 를 사용하십시오.

예

```
XDist=VecDist*cos(Angle)
X(Radius*cos(Azimuth))
```

cosd

기능: (도수를 사용하는)삼각 함수의 cos

구문: **cosd({식})**

도메인: 모든 실수

도메인의 단위: 도수

범위: -1.0~+1.0

범위의 단위: 없음

cosd 는 후속하는 괄호 안에 포함되는 수식(단위는 도수)의 삼각 함수의 cos 함수를 실행합니다.

인수가 라디안 단위의 표준 cos 함수인 경우에는 대신에 **cos** 을 사용하십시오.

예

```
XDist=VecDist*cosd(Angle)  
X(Radius*cosd(Azimuth))
```

cosh

기능: 쌍곡선 코사인

구문: **cosh({식})**

도메인: 모든 실수

도메인의 단위: 라디안

범위: 양의 실수 ≥ 1.0

범위의 단위: 없음

cosh 는 후속하는 괄호 안에 포함되는 수식의 쌍곡선 코사인 함수를 실행합니다.

예

```
Var2=cosh(Var1)
```

exp

기능: 밑 e 의 거듭제곱(e^x)

구문: **exp({식})**

도메인: 모든 실수

도메인의 단위: 사용자 정의

범위: 양의 실수

범위의 단위: 사용자 정의

exp 는 후속하는 괄호 안에 포함되는 수식을 지수에 설정하여 표준 지수 함수를 실행합니다.

$e^{x \ln(y)}$ 를 사용하여 y^x 함수를 구할 수 있지만 **pow(y,x)**도 사용할 수 있습니다.

예

<pre>Var2=exp(Var1) // Raises e to the power of Var1 Var3=exp(Var2*log(Var1)) // Raises Var1 to the power of Var2</pre>

exp2

기능: 밑 2 의 거듭제곱(2^x)

구문: **exp2({식})**

도메인: 모든 실수

도메인의 단위: 사용자 정의

범위: 양의 실수

범위의 단위: 사용자 정의

exp2 는 후속하는 괄호 안에 포함되는 수식을 지수에 설정하여 밑 2 에 대한 지수 함수를 실행합니다.

예

<pre>Var2=exp2(Var1) // Raises 2 to the power of Var1</pre>

floor

기능: 보다 작은 정수로 반올림함

구문: **floor**({식})

도메인: 모든 실수

도메인의 단위: 사용자 정의

범위: 모든 정수

범위의 단위: 사용자 정의

floor 는 후속하는 괄호 안에 포함되는 수식의 정수로 반올림을 실행합니다. 반올림은 항상 역방향으로 실행됩니다.

결과는 정수이지만 부동 소수점의 값으로 나타냅니다. 같은 **int** 함수와 다르며 **floor** 함수는 32 비트의 부호가 있는 정수의 범위 외의 값을 반환할 수 있습니다.

예

```
Var1=2.5
Var2=floor(Var1)           // Take floor of positive value
Var2                       // Query resulting value
2                          // Next lower integer value
Var3=floor(-Var1)         // Take floor of negative value
Var3                       // Query resulting value
-3                         // Next lower integer value
```

int

기능: 보다 작은 정수로 반올림함

구문: **int**({식})

도메인: 모든 실수

도메인의 단위: 사용자 정의

범위: 모든 정수

범위의 단위: 사용자 정의

int 는 후속하는 괄호 안에 포함되는 수식의 정수로 반올림을 실행합니다. 반올림은 항상 역방향으로 실행됩니다.

같은 **floor** 함수와 다르며 **int** 는 32 비트의 부호가 있는 정수의 값을 생성합니다. 식의 값이 32 비트의 부호가 있는 정수로 나타낼 수 있는 값보다 큰 경우에 결과는 최대의 양 또는 음의 32 비트 정수값($+2^{31}-1$ 또는 -2^{31})으로 반올림합니다. 다음으로 결과인 32 비트의 정수값은 후속 연산 또는 대입으로 부동 소수점 형식으로 변환됩니다.

예

```
Var1=2.5
Var2=int(Var1)           // Take int of positive value
Var2                     // Query resulting value
2                         // Next lower integer value
Var3=int(-Var1)          // Take int of negative value
Var3                     // Query resulting value
-3                       // Next lower integer value
```

isnan

기능: NaN(숫자가 아님)의 확인

구문: **isnan({식})**

도메인: 모든 실수로 NaN(숫자가 아님) 표시

도메인의 단위: 사용자 정의

범위: 0, 1

범위의 단위: 없음

isnan 은 식이 유효한 수치인지 여부 또는 무효한 값을 나타내는 'NaN'(숫자가 아님)인지 여부를 사용자가 확인할 수 있게 합니다. 유효한 수치인 경우에는 결과가 0 이 되고 'NaN'인 경우에는 1 이 됩니다.

'NaN'은 함수의 도메인 에러인 경우에 표시되어 음의 값의 제공근 또는 시스템 내에 물리적으로 존재하지 않는 하드웨어에 대한 데이터 구조체 요소값에서 얻을 수 있습니다.

예

```
InvalidSoln=isnan(acosd(Var2))
ICError=isnan(Gate1[Index].PwmPeriod)
```

ln

기능: 자연로그(대체 구문)

{See **log**}

ln 은 자연로그(밑은 'e')의 대체 구문입니다. **ln** 은 **log** 대신에 사용할 수 있지만 **log** 로 저장되어 통지됩니다.

log

기능: 자연로그

구문: **log({식}), or ln({식})**

도메인: 양의 실수

도메인의 단위: 사용자 정의

범위: 모든 실수

범위의 단위: 사용자 정의

log 는 후속하는 괄호 안에 포함되는 수식의 자연로그(밑 'e') 함수를 실행합니다. **ln** 의 구문도 마찬가지로 사용할 수 있지만 프로그램에서는 **log** 구문이 사용됩니다.

다른 밑을 사용하여 로그의 연산은 값의 자연로그를 밑의 자연로그($\log_y x = \ln x / \ln y$)로 나누어서 할 수 있습니다. Power PMAC 은 밑의 2 및 10 에 대해서 전용 함수의 **log2** 및 **log10** 을 각각 제공합니다.

괄호 안의 인수가 정확하게 0 인 경우에 결과는 '마이너스의 무한대(-inf)'의 특수한 부동 소수점 표시가 됩니다. 괄호 안의 인수가 음의 값인 경우에 결과는 NaN(숫자가 아님)의 특수한 부동 소수점 표시가 됩니다. 어떤 경우에도 에러는 통지되지 않고 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

```
Var2=log(Var1)           // Takes the natural log of Var1
Var2=ln(Var1)            // Takes the natural log of Var1
Var4=log(Var3)/log(8)     // Takes the log base 8 of Var3
```

log10

기능: 밑 10 의 로그

구문: **log10({식})**

도메인: 양의 실수

도메인의 단위: 사용자 정의

범위: 모든 실수

범위의 단위: 사용자 정의

log10 은 후속하는 괄호 안에 포함되는 수식의 밑 10 의 로그 함수를 실행합니다.

괄호 안의 인수가 정확하게 0 인 경우에 결과는 '마이너스의 무한대(-inf)'의 특수한 부동 소수점 표시가 됩니다. 괄호 안의 인수가 음의 값인 경우에 결과는 NaN(숫자가 아님)의 특수한 부동 소수점 표시가 됩니다. 어떤 경우에도 에러는 통지되지 않고 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

```
Var2=log10(Var1)         // Takes the log base 10 of Var1
```

log2

기능: 밑 2 의 로그

구문: **log2({식})**

도메인: 양의 실수

도메인의 단위: 사용자 정의

범위: 모든 실수

범위의 단위: 사용자 정의

log2 는 후속하는 괄호 안에 포함되는 수식의 밑 2 의 로그 함수를 실행합니다.

괄호 안의 인수가 정확하게 0 인 경우에 결과는 '마이너스의 무한대(-inf)'의 특수한 부동 소수점 표시가 됩니다. 괄호 안의 인수가 음의 값인 경우에 결과는 NaN(숫자가 아님)의 특수한 부동 소수점 표시가 됩니다. 어떤 경우에도 에러는 통지되지 않고 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

Var2=log2(Var1)	<i>// Takes the log base 2 of Var1</i>

madd

기능: 곱셈과 덧셈

구문: **madd({식},{식},{식})**

도메인: 모든 실수

도메인의 단위: 사용자 정의

범위: 모든 실수

범위의 단위: 사용자 정의

madd 는 중간 값을 확장 배정도로 유지하면서 곱셈과 덧셈을 조합해서 연산합니다. 최초의 2 개 식의 값을 곱하고 그(확장 배정도) 곱을 3 번째 식의 값에 더합니다. 그 결과의 합계가 반환됩니다.

이 함수는 특정 수치 연산의 정밀도를 평가하는 경우에 유용합니다.

예

P1=madd(sqrt(3),sqrt(3),-3) P1 P1=-3.47625514608046876e-16 P2=sqrt(3)*sqrt(3)-3 P2 P2=-4.44089209850062616e-16

pow

기능:	일반 거듭제곱
구문:	pow({식},{식})
도메인:	모든 실수(후술 조합의 제한을 참조)
도메인의 단위:	사용자 정의
범위:	모든 실수
범위의 단위:	사용자 정의

pow 는 최초의 식의 값을 2 번째 식의 지수에 설정하여 거듭제곱을 실행합니다.

최초의 식이 양의 값인 경우에 2 번째 식은 임의의 실수를 가질 수 있습니다. 또 이 함수는 유효한 수치를 반환합니다.

최초의 식이 제로 값인 경우에 2 번째 식은 함수가 유효한 수치를 반환할 수 있도록 반드시 음수가 아닌 값을 가져야 합니다. 2 번째 식이 음의 값인 경우에 결과는 'inf'(무한대)의 특수한 부동 소수점의 값이 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

최초의 식이 음의 값인 경우에 2 번째 식은 함수가 유효한 수치를 반환할 수 있도록 정확한 정수값을 반드시 가져야 합니다. 2 번째 식이 정수가 아닌 값인 경우에 결과는 NaN(숫자가 아님)의 특수한 부동 소수점의 값이 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

```
Var3=pow(Var1,Var2)
```

qnr

기능:	다섯제곱근
구문:	qnr({식})
도메인:	모든 실수
도메인의 단위:	사용자 정의
범위:	모든 실수
범위의 단위:	사용자 정의

qnr 는 후속하는 괄호 안에 포함되는 수식의 다섯제곱근 함수를 실행합니다.

예

Var2=qrrt(Var1)

qrrt

기능:	네제곱근
구문:	qrrt({식})
도메인:	음의 값이 아닌 실수
도메인의 단위:	사용자 정의
범위:	음의 값이 아닌 실수
범위의 단위:	사용자 정의

qrrt 는 후속하는 괄호 안에 포함되는 수식의(양의 실수) 네제곱근 함수를 실행합니다.

괄호 안의 인수가 음의 값이 아닌 규정 도메인을 벗어나는 경우에 결과는 NaN(숫자가 아님)의 특수한 부동 소수점 표기가 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

Var2=qrrt(Var1)

randx

기능:	64 비트 난수 함수
구문:	randx({식})
도메인:	모든 실수
도메인의 단위:	없음
범위:	0.0~인수의 값
범위의 단위:	없음

randx 는 난수 함수를 탑재하고 그 인수로 반환값의 범위를 지정합니다. 이 함수는 0.0 부터(양 또는 음) 인수의 값까지 범위에서 64 비트의 부동 소수점의 값을 반환합니다.

포어그라운드와 백그라운드의 우선도 레벨은(그 자체의 시드값을 가짐) 개별 난수 함수를 가집니다. 포어그라운드의 태스크에는 모션 프로그램 및 실시간 PLC 가 포함되어 백그라운드의 태스크에는 백그라운드 PLC 및 온라인 명령이 포함됩니다.

randx 함수를 호출할 때마다 우선도 레벨에 대한 4개의 모든 시드값(0, 1, 2, 3)이 사용되고 또 제너레이터에 대한 다음의 호출 준비로 갱신됩니다. 모든 시드값은 전원 셋업/리셋 시에 특정 값으로 설정되어 많은 사용자는 기본값에 따라 기동할 수 있습니다. 또 시드값은 **seed** 함수를 사용하여 언제든지 설정 가능합니다.

예

```
Var1=randx(2.0)-1.0           // Result is between -1.0 and +1.0
```

rem

기능: 나머지

구문: **rem({식},{식})**

도메인: 모든 실수(2^{nd} 인수인 경우에는 제로 이외의 값)

도메인의 단위: 사용자 정의

범위: 음의 값이 아닌 실수

범위의 단위: 사용자 정의

rem 은 나머지 함수를 탑재하고 최초의 식의 값을 2 번째 식의 값으로 나누었을 때 최소 나머지를 반환합니다. 반환값은 다음 범위가 됩니다.

$$-Divisor/2 < \text{Returned Value} \leq Divisor/2$$

이 범위는 % 나머지 연산자의 결과 범위와 다르므로 주의하십시오.

2 번째 식이 정확하게 0 의 값을 가지는 경우에 결과는 NaN(숫자가 아님)의 특수한 부동 소수점의 값이 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

```
Var1=5
Var2=rem(Var1,2)           // Find remainder modulo 2
Var2                         // Query resulting value
1                           // Reports remainder value
Var1=5.01
Var3=rem(Var1,2)           // Find remainder modulo 2
Var3                         // Query resulting value
-0.99                       // Reports rounded value
```

rint

기능: 반올림

구문: **rint({식})**

도메인: 모든 실수

도메인의 단위: 사용자 정의

범위: 모든 정수

범위의 단위: 사용자 정의

rint 는 후속하는 괄호 안에 포함되는 수식의 반올림 함수를 실행합니다. 식의 소수값이 정확하게 0.5 인 경우에 결과는 항상(플러스의 무한대 방향으로) 반올림합니다.

결과는 정수이지만 부동 소수점의 값으로 표시됩니다.

예

```
Var1=2.4
Var2=rint(Var1)           // Round to nearest integer
Var2                      // Query resulting value
2                          // Reports rounded value
Var3=rint(Var1+0.2)       // Round expression to nearest integer
Var3                      // Query resulting value
3                          // Reports rounded value
Var3=rint(Var1+0.1)       // Round expression to nearest integer
Var3                      // Query resulting value
3                          // Reports rounded(up) value
```

rnd

기능: 32 비트 난수 함수

구문: **rnd({식})**

도메인: 0~1

도메인의 단위: 없음

범위: 0.0~0.99999...

범위의 단위: 없음

rnd 는 난수 함수를 탑재하고 인수 1 의 비트를 제너레이터 번호인 0 또는 1 로 합니다. 모션 프로그램이나 실시간 PLC 등의 포어그라운드 태스크용에 2 개의 독립된 난수 함수(0 및 1)가 존재합니다. 또 백그라운드 PLC 나 온라인 명령 등의 백그라운드 태스크용에 그 외에도 2 개의 독립된 난수 함수(이것도 0 및 1)가 존재합니다. 인수는 임의의 실수이지만 어떤 제너레이터가 사용되는지 판단하기 위해서 1 의 비트만 평가됩니다. 보통 이 함수는 **rnd(0)** 또는 **rnd(1)**로 사용됩니다.

인수 값과 상관없이 이 함수는 0.0 보다 큰 또는 같은 값 및 1.0 보다 작은 값을 반환합니다. 반환되는 수치는 32 비트의 부동 소수점의 값입니다.

주어진 난수 함수에 액세스할 때마다 2 개의 시드값이 사용됩니다. 제너레이터 0 은 시드값의 0 및 1 을 사용하고 제너레이터 1 은 시드값의 2 및 3 을 사용합니다. 반환된 랜덤 값의 연산 프로세스로 이 제너레이터의 다음 액세스에 준비하기 위해서 이 시드값인 양쪽이 새로운 값으로 갱신됩니다. 모든 시드값은 전원 셋업/리셋 시에 특정 값으로 설정되어 많은 사용자는 기본값에 따라 기동할 수 있습니다. 또 시드값은 **seed** 함수를 사용하여 언제든지 설정 가능합니다.

예

```
Var1=0; Var2=0; // Initialize values
while(Var1<1000) { // Loop 1000 times
    Var2+=rnd(1); // Add random value to sum
    Var1++; // Increment cycle counter
} // Sum is ~on normal distribution
Var2=(Var2/500)-1; // Normalize and center on zero
```

seed

기능: 난수 시드 발생기

구문: **seed({s},{s})**

도메인 1: 부호가 있는 32 비트 정수

도메인 1의 단위: 없음

도메인 2: 0 .. 3

도메인 2의 단위: 시드 번호

범위: 부호가 없는 32 비트 정수

범위의 단위: 없음

seed 는 난수 발생을 위한 시드값을 생성합니다. 각 난수 함수용으로 2 개의 시드값이 사용됩니다. 시작 시의 기본값인 시드값을 사용하는 것도 가능하지만 사용자는 이 함수를 사용하여 시작 시의 사용자 독자적인 시드값을 설정할 수 있습니다.

2 번째 인수는 어떤 시드값을 생성할지 지정합니다. 0, 1, 2 또는 3 의 값을 가질 수 있고 이것은 우선도 레벨에 따라서 4 개의 시드값 중 어떤 것을 생성할지 지정합니다(양쪽 포어그라운드와 태스크(즉, 모션 프로그램과 실시간 PLC) 및 백그라운드 태스크(즉, 백그라운드 PLC 와 온라인 명령)는 각자가 4 개의 시드값의 독립된 세트를 가집니다). 각각의 우선도 레벨에서 **rnd(0)**은 시드값인 0 과 1 을 사용합니다. **rnd(1)**은 시드값인 2 와 3 을 사용합니다. 64 비트의 난수 발생 함수의 **randx** 는 우선도 레벨에 따라서 전체 4 세트의 시드값을 사용합니다.

최초의 인수는 시드값의 생성 방법을 지정합니다. 이 값이 0 보다 큰 경우에 시드값은 인수와 같은 값으로 설정됩니다. 이 값이 0 보다 작은 경우에 시드값은 이 함수가 호출되었을 때 프로세서 내부 타이머 레지스터의 값에서 설정됩니다. 최초의 사례에서는 반복 가능한 난수 시퀀스가 허용되지만 2 번째 사례에서는 반복 불가능한 시퀀스가 생성됩니다.

최초의 인수가 0 과 같은 경우에 시드값은 변경되지 않고 함수는 단순히 시드의 현재 값을 반환합니다.

모든 사례에서 시드의 결과 값은 양쪽 모두 내부 데이터 구조체에 저장되어 함수의 반환값으로 사용됩니다. 대부분의 사례에서는 반환값은 직접 사용하지 않기 때문에 일반적으로는 더미 변수에 대입됩니다.

sin

기능: sin(사인)

구문: **sin({식})**

도메인: 모든 실수

도메인의 단위: 라디안

범위: -1.0~+1.0

범위의 단위: 없음

sin 은 후속하는 괄호 안에 포함되는 수식(단위는 라디안)의 표준 삼각 함수의 사인 함수를 실행합니다.

인수가 도수 단위의 사인 함수인 경우에는 대신에 **sind** 를 사용하십시오.

예

```
YDist=VecDist*sin(Angle)
Y(Radius*sin(Azimuth))
```

sincos

기능: 삼각 함수의 sin 및 cos 의 조합 함수

구문: **sincos({식1},{식2})**

도메인 1: 모든 실수

도메인 1의 단위: 라디안

도메인 2: 음의 값이 아닌 정수

도메인 2의 단위: 로컬 변수

범위: -1.0~+1.0

범위의 단위: 없음

sincos 은 후속하는 괄호 안에 포함되는 최초의 수식(단위는 라디안)의 삼각 함수 sin/cos 의 특수한 조합 함수를 실행합니다. 이 함수의 결과는 최초의 식의 sin 이 됩니다.

2번째 식은 함수를 실행하는 프로그램 또는 태스크에 대하여 로컬('L') 변수값을 지정합니다. 산출된 cos 의 값이 들어가는 로컬 변수의 번호는 음의 값이 아닌 정수이며 2번째 식은 필요에 따라서 사용하는 변수를 선택하기 위해 가장 가까운 정수값으로 반올림됩니다.

각도의 sin 과 cos 양쪽을 계산하는 경우에 개별 **sin** 및 **cos** 을 사용하는 것보다 조합한 **sincos** 함수를 사용하는 편이 처리가 빠릅니다.

인수가 도수 단위의 sin/cos 의 조합 함수인 경우에는 대신에 **sincosd** 를 사용하십시오.

예

```
L1=VecDist*sincos(Angle,2)           // Sine into L1, cosine into L2
X(VecDist*L2)Y(L1)
```

sincosd

기능: 삼각 함수의 sin 및 cos 의 조합 함수(단위는 도수)

구문: **sincosd({식 1},{식 2})**

도메인 1: 모든 실수

도메인 1의 단위: 도수

도메인 2: 음의 값이 아닌 정수

도메인 2의 단위: 로컬 변수

범위: -1.0~+1.0

범위의 단위: 없음

sincosd 는 후속하는 괄호 안에 포함되는 최초의 수식(단위는 도수)의 삼각 함수 sin/cos 의 특수한 조합 함수를 실행합니다. 이 함수의 결과는 최초의 식의 sin 이 됩니다.

2 번째 식은 함수를 실행하는 프로그램 또는 태스크에 대하여 산출된 cos 의 값이 들어가는 로컬('L') 변수값을 지정합니다. 로컬 변수의 번호는 음의 값이 아닌 정수이며 2 번째 식은 필요에 따라 사용하는 변수를 선택하기 위해 가장 가까운 정수값으로 반올림됩니다.

각도의 sin 과 cos 양쪽을 계산하는 경우에 개별 **sind** 및 **cosd** 를 사용하는 것보다 조합한 **sincosd** 함수를 사용하는 편이 처리가 빠릅니다.

인수가 라디안 단위의 sin/cos 의 조합 함수인 경우에는 대신에 **sincos** 을 사용하십시오.

예

```
L1=VecDist*sincosd(Angle,2)           // Sine into L1, cosine into L2
X(VecDist*L2)Y(L1)
```

sind

기능: (도수를 사용하는)삼각 함수의 sin

구문: **sind({식})**

도메인: 모든 실수

도메인의 단위: 도수

범위: -1.0~+1.0

범위의 단위: 없음

sind 는 후속하는 괄호 안에 포함되는 수식(단위는 도수)의 삼각 함수의 sin 함수를 실행합니다.

인수가 라디안 단위의 표준 sin 함수인 경우에는 대신에 **sin** 을 사용하십시오.

예

```
YDist=VecDist*sind(Angle)
Y(Radius*sind(Azimuth))
```

sinh

기능: 쌍곡선 사인

구문: **sinh({식})**

도메인: 모든 실수

도메인의 단위: 라디안

범위: 모든 실수

범위의 단위: 없음

sinh 는 후속하는 괄호 안에 포함되는 수식의 쌍곡선 사인 함수를 실행합니다.

예

```
Var2=sinh(Var1)
```

sgn

기능: 산술 부호

구문: **sgn({식})**

도메인: 모든 실수

도메인의 단위: 사용자 정의

범위: -1, 0, +1

범위의 단위: 없음

sgn 은 후속하는 괄호 안에 포함되는 수식의 산술 부호(signum) 함수를 실행합니다. 식이 음의 값을 가지는 경우에 결과는 -1 이 됩니다. 식이 양의 값을 가지는 경우에 결과는 +1 이 됩니다. 식이 정확하게 0 의 값을 가지는 경우에 결과는 0 이 됩니다.

예

```
Direction=sgn(CmdVel)
```

sqrt

기능:	제곱근
구문:	sqrt({식})
도메인:	음의 값이 아닌 실수
도메인의 단위:	사용자 정의
범위:	음의 값이 아닌 실수
범위의 단위:	사용자 정의

sqrt 는 후속하는 괄호 안에 포함되는 수식의 양의 제곱근 함수를 실행합니다.

괄호 안의 인수가 음의 값이 아닌 규정 도메인을 벗어나는 경우에 결과는 NaN(숫자가 아님)의 특수한 부동 소수점 표기가 됩니다. 그 경우 에러는 통지되지 않으며 프로그램은 정지하지 않습니다. 이 값에 이어지는 수학 연산에서는 예측 불가능한 결과가 될 가능성이 있습니다. 도메인 에러의 가능성은 프로그래머가 확인합니다.

예

```
Var2=sqrt(Var1)
VecVel=sqrt(XVel*XVel+YVel*YVel)
```

tan

기능:	삼각 함수의 tan(탄젠트)
구문:	tan({식})
도메인:	모든 실수이며 '+/-(2N-1)*Pi/2' 이외
도메인의 단위:	라디안
범위:	모든 실수
범위의 단위:	없음

tan 는 후속하는 괄호 안에 포함되는 수식(단위는 라디안)의 표준 삼각 함수의 tan 함수를 실행합니다.

괄호 안의 인수가 '+/-(2N-1)*Pi/2'의 라디안(+/-Pi/2, 3Pi/2, 5Pi/2 등)인 경우에 **tan** 함수는 발산 상태가 되고 매우 큰 값이 반환됩니다. 이런 결과에 따른 후속 계산은 유효성에 문제가 있습니다. 이런 상태는 프로그래머가 확인합니다.

인수가 도수 단위의 tan 함수인 경우에는 대신에 **tand** 를 사용하십시오.

예

```
YDist=XDist*tan(Angle)
```

tand

기능: (도수를 사용하는)삼각 함수의 tan(탄젠트)

구문: **tand({식})**

도메인: 모든 실수이며 '+/-(2N-1)*90°' 이외

도메인의 단위: 도수

범위: 모든 실수

범위의 단위: 없음

tand 는 후속하는 괄호 안에 포함되는 수식(단위는 도수)의 삼각 함수의 tan 함수를 실행합니다.

괄호 안의 인수가 '+/-(2N-1)*90°'의 도수(+/-90, 270, 450 등)인 경우에 **tand** 함수는 발산 상태가 되고 매우 큰 값이 반환됩니다. 이런 결과에 따른 후속 계산은 유효성에 문제가 있습니다. 이런 상태는 프로그래머가 확인합니다.

인수가 라디안 단위의 표준 tan 함수인 경우에는 대신에 **tan** 를 사용하십시오.

예

`YDist=XDist*tand(Angle)`

tanh

기능: 쌍곡선 탄젠트

구문: **tanh({식})**

도메인: 모든 실수

도메인의 단위: 라디안

범위: -1.0~+1.0

범위의 단위: 없음

tanh 는 후속하는 괄호 안에 포함되는 수식의 쌍곡선 탄젠트 함수를 실행합니다.

예

`Var1=tanh(Var2)`

벡터 함수

Power PMAC의 스크립트 언어는 벡터의 각 요소를 동작하지 않고 강력하고 유연한 1 차원(1D)의 벡터 연산을 가능하게 하는 벡터 함수 세트를 제공합니다. Power PMAC 내의 벡터는 태스크(P 인 경우에는 0, L 인 경우에는 1, Ddata 인 경우에는 2)에 대해서 로컬에서 비저장인 [Ldata.Control]의 설정에 따라서 연속한 번호 세트의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열 배정도의 부동 소수점('Sys.Ddata[i]') 변수를 사용합니다.

변수값 요소의 공간 또는 요소 수를 지정하는 벡터 함수의 인수는 함수 내에서 사용하기 전에 항상 정수값(음의 무한대 방향)으로 반올림합니다. 정수로 직접 지정된 경우 또는 정수값만 사용하여 종료한 경우에 반올림 에러에 대한 문제는 없습니다. 단, 이 값의 계산에 임의의 부동 소수점의 값 또는 함수가 사용된 경우에는 어플리케이션으로 강제적인 올바른 반올림('int(Var+0.5)' 등)을 실행하여 필요한 정수값을 사용하기를 권장합니다. 그 이외의 경우에는 계산상의 미세한 정밀도의 저하로 인해 의도하지 않은 값을 사용할 가능성이 있습니다(예를 들어 '2.9999999999'인 경우에 '2'를 지정한 경우 등).

IDE 프로젝트 매니저로 사용한 경우에 &(앰퍼샌드) 문자의 다음 선언명을 사용하여(해당하는 변수값이 불명인 경우에도) 선언된 변수명을 참조할 수 있습니다. 그 예시는 다음과 같습니다.

```
global MyVector1(6), MyVector2(6);
...
vcopy(&MyVector1(0), &MyVector2(0), 6);
```

sum

기능: 벡터 요소의 합

구문: **sum({exp1},{exp2},{exp3})**

sum 함수는 후속하는 괄호 안에 포함되는 식의 값으로 지정된 벡터의 지정된 요소 수의(스칼라) 합을 계산하여 결과의 합을 반환합니다. 벡터는 태스크(P 인 경우에는 0, L 인 경우에는 1, Ddata 인 경우에는 2)에 대한 [Ldata.Control]의 설정에 따라서 연속 번호의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열의 배정도 부동 소수점('Sys.Ddata[i]') 변수의 세트에 따라서 형성됩니다.

최초의 식인 **exp1** 은 벡터의 시작 변수의 번호를 지정합니다.

2 번째 식인 **exp2** 는 추가하는 벡터 내의 요소 수를 지정합니다.

3 번째 식인 **exp3** 은 추가하는 요소 간의 증가분(공간)을 지정합니다. 이 식의 값이 1 인 경우에는 벡터 내에서 연속한 요소를 사용하는 것을 지정합니다.

sum 함수는 정방 행렬(대각 요소의 합)의 트레이스 계산에 유용합니다. 이 방법에서 2 번째 식을 2 번째 행렬의 행수와 열수와 같게 해야 하며 또 3 번째 식은 2 번째 식보다 하나 크게 해야 합니다.

예

```

P20=10 P21=5 P22=0           // Set first row of matrix
P23=5 P24=12 P25=0          // Set second row of matrix
P26=1 P27=2 P28=15          // Set third row of matrix
P30=sum(20,3,4)             // Add P20, P24, P28
P30
P30=37

```

sumprod

기능: 벡터 요소의 곱의 합

구문: **sumprod({exp1},{exp2},{exp3},{exp4},{exp5})**

sumprod 함수는 후속하는 괄호 안에 포함되는 식의 값으로 지정된 벡터의 지정된 요소 수의(스칼라) 합을 계산하여 결과의 합을 반환합니다. 이 함수는 (2D 행렬의 행과 열을 포함) 벡터의 스칼라 곱의 계산에 유용합니다.

벡터는 태스크(P 인 경우에는 0, L 인 경우에는 1, Ddata 인 경우에는 2)에 대한 [Ldata.Control]의 설정에 따라서 연속 번호의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열의 배정도 부동 소수점('Sys.Ddata[l]') 변수의 세트에 따라서 형성됩니다.

최초의 식인 **exp1** 은 연산으로 사용하는 최초의 벡터의 시작 변수 번호를 지정합니다.

2 번째 식인 **exp2** 는 연산으로 사용하는 2 번째 벡터의 시작 변수 번호를 지정합니다.

3 번째 식인 **exp3** 은 양쪽 소스 벡터로 사용하는 요소의 수를 지정합니다.

4 번째 식인 **exp4** 는 최초의 벡터 내에서 사용하는 요소 간의 증가분(공간)을 지정합니다.

5 번째 식인 **exp5** 는 2 번째 벡터 내에서 사용하는 요소 간의 증가분(공간)을 지정합니다. 이 식의 값이 1 인 경우에는 벡터 내에서 연속한 요소를 사용하는 것을 지정합니다.

예

```

P50=10 P51=5 P52=2
P60=20 P61=3 P62=1
P70=sumprod(50,60,3,1,1)
P70
P70=217

P20=1 P21=2 P22=3
P23=4 P24=5 P25=6
P26=7 P27=8 P28=9
P40=2 P41=3 P42=4
P10=sumprod(20,40,3,3,1)
P10
P10=42

```

vcopy

기능: 벡터 복사

구문: **vcopy({exp1},{exp2},{exp3})**

vcopy 함수는 하나의 벡터(소스의 벡터) 내의 모든 요소의 값을 다른 벡터(결과의 벡터)의 요소에 복사합니다. 벡터는 태스크(P 인 경우에는 0, L 인 경우에는 1, Ddata 인 경우에는 2)에 대한 [Ldata.Control]의 설정에 따라서 연속 번호의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열의 배정도 부동 소수점('Sys.Ddata[i]') 변수의 세트에 따라서 형성됩니다.

최초의 식인 **exp1** 은 결과 벡터의 시작 변수 번호를 지정합니다.

2 번째 식인 **exp2** 는 소스 벡터의 시작 변수 번호를 지정합니다.

3 번째 식인 **exp3** 은 소스 벡터와 결과의 벡터 양쪽의 요소의 수를 지정합니다.

이 함수의 반환값은 올바르게 연산된 것을 나타내는 부울 형식의 플래그입니다. 올바르게 연산된 경우에 이것이 1 로 설정되고 올바르게 연산되지 않은 경우에는 0 으로 설정됩니다(올바르게 연산되지 않는 경우에 보통은 요소 수가 제로 또는 음의 값이 되는 범위 외의 값이 지정됩니다). 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

또 **vcopy** 함수를 2D 행렬의 복사에 사용할 수 있습니다. 복사가 목적인 경우에 1D 벡터로 취급할 수 있습니다. 예를 들어 '3x4'의 행렬은 12 요소의 벡터로 처리할 수 있습니다.

예

```
P80=10 P81=5 P82=2
P90=vcopy(80, 90, 3)
P90, 3
P90=10
P91=5
P92=2
```

vmadd

기능: 벡터의 곱셈과 덧셈

구문: **vmadd({exp1},{exp2},{exp3},{exp4},{exp5})**

vmadd 함수는 후속하는 괄호 안에 포함되는 식의 값으로 지정된 2 개의 벡터의 합(추가하기 전에 최초의 벡터를 상수로 스케일링)을 계산하고 결과의 합 벡터를 저장합니다. 벡터는 태스크(P 인 경우에는 0, L 인 경우에는 1, Ddata 인 경우에는 2)에 대한 [Ldata.Control]의 설정에 따라서 연속 번호의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열의 배정도 부동 소수점('Sys.Ddata[i]') 변수의 세트에 따라서 형성됩니다.

최초의 식인 **exp1** 은 결과(합)의 벡터의 시작 변수 번호를 지정합니다.

2 번째 식인 **exp2** 는 추가하는 최초의 벡터의 어떤 요소를(벡터 추가 전에) 곱할지를 결정하는 스케일 계수를 지정합니다.

3 번째 식인 **exp3** 은 추가하는 최초의 벡터의 시작 변수 번호를 지정합니다.

4 번째 식인 **exp4** 는 추가하는 2 번째 벡터의 시작 변수 번호를 지정합니다.

5 번째 식인 **exp5** 는 소스 벡터와 결과의 벡터의 양쪽에 포함되는 요소 수를 지정합니다.

스케일 계수를 +1 로 설정하면 2 개의 소스 벡터를 직접 더할 수 있습니다. 스케일 계수를 -1 로 설정하면 2 번째 소스 벡터에서 최초의 소스 벡터를 뺍니다.

이 함수의 반환값은 올바르게 연산된 것을 나타내는 부울 형식의 플래그입니다. 올바르게 연산된 경우에 이것이 1 로 설정되고 올바르게 연산되지 않은 경우에는 0 으로 설정됩니다(올바르게 연산되지 않는 경우에 보통은 요소 수가 제로 또는 음의 값이 되는 범위 외의 값이 지정됩니다). 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

또 **vmadd** 함수는 2D 행렬을 함께 추가하기 위해서 사용할 수 있습니다. 추가가 목적인 경우에 1D 의 벡터로 사용할 수 있습니다. 예를 들어 '3x4'의 행렬은 12 요소의 벡터로 처리할 수 있습니다.

예

```
P30=5 P31=10 P32=3
P50=3 P51=6 P52=8
P0=vmadd(70,1,50,30,3) // Standard addition of vectors
P70,3
P70=8
P71=16
P72=11
P0=vmadd(70,-1,50,30,3) // Subtraction of vectors
P70,3
P70=2
P71=4
P72=-5
```

vscale

기능: 벡터의 스칼라 곱

구문: **vscale({exp1},{exp2},{exp3},{exp4})**

vscale 함수는 하나의 벡터(소스의 벡터) 내의 모든 요소의 값을 가지고 그것을 공통의 스케일 계수로 곱하고 그 곱의 값을 다른 벡터(결과의 벡터)의 요소에 씁니다. 벡터는 태스크(P 인 경우에는 0, L 인 경우에는 1, Ddata 인 경우에는 2)에 대한 [Ldata.Control]의 설정에 따라서 연속 번호의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열의 배정도 부동 소수점('Sys.Ddata[i]') 변수의 세트에 따라서 형성됩니다.

최초의 식인 **exp1** 은 결과 벡터의 시작 변수 번호를 지정합니다.

2 번째 식인 **exp2** 는 부동 소수점의 값으로 스케일 계수를 지정합니다.

3 번째 식인 **exp3** 은 소스 벡터의 시작 변수 번호를 지정합니다.

4 번째 식인 **exp4** 는 소스 벡터와 결과의 벡터 양쪽의 요소 수를 지정합니다.

이 함수의 반환값은 올바르게 연산된 것을 나타내는 부울 형식의 플래그입니다. 올바르게 연산된 경우에 이것이 1로 설정되고 올바르게 연산되지 않은 경우에는 0으로 설정됩니다(올바르게 연산되지 않는 경우에 보통은 요소 수가 제로 또는 음의 값이 되는 범위 외의 값이 지정됩니다). 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

또 **vscale** 함수를 2D 행렬의 스케일링에 사용할 수 있습니다. 스케일링이 목적인 경우에 1D의 벡터로 사용할 수 있습니다. 예를 들어 '3x4'의 행렬은 12 요소의 벡터로 처리할 수 있습니다.

예

```
P80=10 P81=5 P82=2
P0=vcopy(80,4,90,3)
P90,3
P90=40
P91=20
P92=8
```

매트릭스 산술 함수

Power PMAC의 스크립트 언어는 행렬의 각 요소를 동작하지 않고 강력하고 유연한 2차원(2D)의 행렬 연산을 가능하게 하는 행렬 함수의 세트를 제공합니다. Power PMAC 내의 행렬은 태스크(P인 경우에는 0, L인 경우에는 1, Ddata인 경우에는 2)에 대해서 로컬에서 비저장인 [Ldata.Control] 설정에 따라서 연속한 번호 세트의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열 배정도의 부동 소수점('Sys.Ddata[i]') 변수를 사용합니다.

같은 행의 행렬 요소는 연속한 번호의 변수를 사용하고 같은 열의 행렬 요소는 열의 수에 따라서 변수 번호(인덱스)로 분리됩니다.

행 번호와 열 번호의 지정자는 0부터 시작되기 때문에 3행과 4열로 지정된 행렬은 행 0, 1 및 2를 가지고 또 열 0, 1, 2 및 3을 가집니다.

행렬 함수의 반환값은 항상 하나의 스칼라값이 됩니다. 이 함수의 주요 결과가 행렬식 등의 스칼라값인 경우에 이 값이 함수의 반환값이 됩니다. 단, 이 함수의 주요 결과가 행렬인 경우에 반환값은(연산의 성공/실패를 나타냄) 부울 형식의 플래그 또는 계산에 관여하는 행렬의 행렬식 등의 보조값이 됩니다.

Power PMAC 내의 2D 행렬은 기본적으로 연속한 번호의 변수 배열이기 때문에 이것도 몇 개의 연산에서는 벡터로 취급할 수 있습니다. 벡터 함수의 `vadd`, `vscale` 및 `vcopy`는 각각이 덧셈, 스케일링 및 복사용으로 2D 행렬의 연산에도 매우 유용합니다.

변수 번호, 행 또는 열의 번호 또는 행과 열의 수를 지정하는 행렬 함수의 인수는 함수로 사용하기 전에 반드시 정수값(음의 무한대 방향)으로 반올림합니다. 정수로 직접 지정된 경우 또는 정수값만 사용하여 종료한 경우에 반올림 에러에 대한 문제는 없습니다. 단, 이 값의 계산에 임의의 부동 소수점의 값 또는 함수가 사용된 경우에는 어플리케이션에서 강제적인 올바른 반올림(`[int(Var+0.5)]` 등)을 실행하여 필요한 정수값을 사용하기를 권장합니다. 그 이외의 경우에는 계산상의 미세한 정밀도의 저하로 인해 의도하지 않은 값을 사용할 가능성이 있습니다(예를 들어 '2.9999999999'인 경우에 '2'를 지정한 경우 등).

mdet

기능: 매트릭스 행렬식

구문: **mdet({exp1},{exp2})**

mdet 함수는 후속하는 괄호 안에 포함되는 식의 값에 따라서 지정되는 정방 행렬의 행렬식을 계산하고 결과를 반환합니다. 행렬은 연속한 번호의 글로벌('P') 변수의 세트에 따라서 형성됩니다. 행렬은 태스크(P인 경우에는 0, L인 경우에는 1, Ddata인 경우에는 2)에 대한 [Ldata.Control]의 설정에 따라서 연속 번호의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열의 배정도 부동 소수점('Sys.Ddata[i]') 변수의 세트에 따라서 형성됩니다.

최초의 식인 **exp1**은 행렬의 시작 변수 번호를 지정합니다.

2번째 식인 **exp2**는 정방 행렬의 행수와 열수를 지정합니다.

예

```
P50=2 P51=0
P52=0 P53=2
P54=mdet(50,2)
P54
4
```

minv

기능: 역행렬

구문: **minv({exp1},{exp2},{exp3})**

minv 함수는 후속하는 괄호 안에 포함되는 식의 값에 따라서 지정되는 정방 행렬의 역행렬을 계산하여 결과를 저장합니다. 소스 행렬과 결과의 행렬은 태스크(P 인 경우에는 0, L 인 경우에는 1, Ddata 인 경우에는 2)에 대한 [Ldata.Control]의 설정에 따라서 연속 번호의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열의 배정도 부동 소수점('Sys.Ddata[i]') 변수의 세트에 따라서 형성됩니다.

최초의 식인 **exp1** 은 결과 (역)행렬의 시작 변수 번호를 지정합니다.

2 번째 식인 **exp2** 는 소스 행렬의 시작 변수 번호를 지정합니다.

3 번째 식인 **exp3** 은 정방 소스 행렬과 결과의 행렬 행수와 열수를 지정합니다.

이 함수의 반환값은 소스 행렬의 행렬식입니다. 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다. 이 값이 정확하게 0 인 경우에 반전은 무효가 됩니다. 또 0에 매우 가까운 경우('매우 가까운'의 의미의 정의는 어플리케이션에 따라서 달라집니다), 반전은 문제가 남습니다.

예

```
P50=2 P51=0
P52=0 P53=2
P54=minv(60,50,2)
P54
P54=4
P60,4
P60=0.5
P61=0
P62=0
P63=0.5
```

mminor

기능: 매트릭스 소행렬식

구문: **mminor**({*exp1*},{*exp2*},{*exp3*},{*exp4*})

mminor 함수는 후속하는 괄호 안에 포함되는 식의 값에 따라서 지정되는 정방 행렬의 소행렬식을 계산하여 결과를 반환합니다. 행렬은 태스크(P 인 경우에는 0, L 인 경우에는 1, Ddata 인 경우에는 2)에 대한 [Ldata.Control]의 설정에 따라서 연속 번호의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열의 배정도 부동 소수점('Sys.Ddata[i]') 변수의 세트에 따라서 형성됩니다.

최초의 식인 **exp1** 은 행렬 전체의 시작 변수 번호를 지정합니다.

2 번째 식인 **exp2** 는 정방 행렬의 행수와 열수를 지정합니다.

3 번째 식인 **exp3** 은 소행렬식에서 제거되는 행렬 전체의 행번호를 지정합니다.

4 번째 식인 **exp4** 는 소행렬식에서 제거되는 행렬 전체의 열번호를 지정합니다. 행과 열의 번호는 1 이 아니라 0 부터 시작되는 점에 주의하십시오.

예

```
P60=2 P61=0 P62=0
P63=0 P64=3 P65=0
P66=0 P67=0 P68=4
P70=mminor(60,3,0,0)
P70
12
P70=mminor(60,3,1,1)
P70
8
P70=mminor(60,3,2,2)
P70
6
```

mmadd

기능: 행렬의 곱셈과 덧셈

구문: **mmadd**({*exp1*},{*exp2*},{*exp3*},{*exp4*},{*exp5*},{*exp6*})

mmadd 함수는 후속하는 괄호 안에 포함되는 식의 값으로 지정된 2 개의 행렬의 곱을 계산하여 결과 곱의 행렬을 3 번째 행렬에 더하고 그 행렬의 합을 저장합니다. 소스 행렬과 결과의 행렬은 태스크(P 인 경우에는 0, L 인 경우에는 1, Ddata 인 경우에는 2)에 대한 [Ldata.Control]의 설정에 따라서 연속 번호의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열의 배정도 부동 소수점('Sys.Ddata[i]') 변수의 세트에 따라서 형성됩니다.

최초의 식인 **exp1** 은 결과(곱과 합)의 시작 변수 번호를 지정합니다. 함수를 실행하기 전의 이 행렬 내의 값은 다른 2 개의 행렬의 곱의 값에 더해집니다.

2 번째 식인 **exp2** 는 곱하는 최초의 행렬의 시작 변수 번호를 지정합니다.

3 번째 식인 **exp3** 은 곱하는 2 번째 행렬의 시작 변수 번호를 지정합니다. 행렬의 곱셈은 가환이 아니기 때문에 이 행렬의 순서가 중요하다는 점을 기억하십시오.

4 번째 식인 **exp4** 는 곱하는 최초의 소스 행렬 내의 행수를 지정하기 때문에 결과(곱)의 행렬의 행수를 지정합니다.

5 번째 식인 **exp5** 는 곱하는 최초의 행렬 내의 열수를 지정하기 때문에 곱하는 2 번째 행렬 내의 행수를 지정합니다.

6 번째 식의 **exp6** 은 곱하는 2 번째 행렬 내의 열수를 지정하기 때문에 결과(곱)의 행렬 내의 열수를 지정합니다.

이 함수의 반환값은 올바르게 연산된 것을 나타내는 부울 형식의 플래그입니다. 올바르게 연산된 경우에 이것이 1 로 설정되고 올바르게 연산되지 않은 경우에는 0 으로 설정됩니다(올바르게 연산되지 않은 경우에 보통은 행 또는 열의 수가 제로 또는 음의 값이 되는 범위 외의 값이 지정됩니다). 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

예

```
P10=1 P11=2 P12=3
P13=4 P14=5 P15=6

P20=7 P21=8
P22=9 P23=10
P24=11 P25=12

P30=58 P31=64
P32=139 P33=154
P40=mmadd(30,10,20,2,3,2)
P30,4
P30=116
P31=128
P32=278
P33=308
```

mmul

기능: 매트릭스 곱셈

구문: **mmul({exp1},{exp2},{exp3},{exp4},{exp5},{exp6})**

mmul 함수는 후속하는 괄호 안에 포함되는 식의 값에 따라서 지정되는 2 개의 행렬의 곱을 계산하여 결과 곱의 행렬을 저장합니다. 소스 행렬과 결과의 행렬은 태스크(P 인 경우에는 0, L 인 경우에는 1, Ddata 인 경우에는 2)에 대한 [Ldata.Control]의 설정에 따라서 연속 번호의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열의 배정도 부동 소수점('Sys.Ddata[i]') 변수의 세트에 따라서 형성됩니다.

최초의 식인 **exp1** 은 결과(곱)의 행렬의 시작 변수 번호를 지정합니다.

2 번째 식인 **exp2** 는 곱하는 최초의 행렬의 시작 변수 번호를 지정합니다.

3 번째 식인 **exp3** 은 곱하는 2 번째 행렬의 시작 변수 번호를 지정합니다. 행렬의 곱셈은 가환이 아니기 때문에 이 행렬의 순서가 중요하다는 점을 기억하십시오.

4 번째 식인 **exp4** 는 곱하는 최초의 소스 행렬 내의 행수를 지정하기 때문에 결과(곱)의 행렬의 행수를 지정합니다.

5 번째 식인 **exp5** 는 곱하는 최초의 행렬 내의 열수를 지정하기 때문에 곱하는 2 번째 행렬 내의 행수를 지정합니다.

6 번째 식의 **exp6** 은 곱하는 2 번째 행렬 내의 열수를 지정하기 때문에 결과(곱)의 행렬 내의 열수를 지정합니다.

이 함수의 반환값은 올바르게 연산된 것을 나타내는 부울 형식의 플래그입니다. 올바르게 연산된 경우에 이것이 1 로 설정되고 올바르게 연산되지 않은 경우에는 0 으로 설정됩니다(올바르게 연산되지 않은 경우에 보통은 행 또는 열의 수가 제로 또는 음의 값이 되는 범위 외의 값이 지정됩니다). 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

예

```
P10=1 P11=2 P12=3
P13=4 P14=5 P15=6
P20=7 P21=8
P22=9 P23=10
P24=11 P25=12
P40=mmul(30,10,20,2,3,2)
P30,4
P30=58
P31=64
P32=139
P33=154
P40=mmul(30,20,10,3,2,3)
P30,9
P30=39
P31=54
P32=69
P33=49
P34=68
P35=87
P36=59
P37=82
P38=105
```

msolve

기능: 매트릭스 답 구하기

구문: **msolve({exp1},{exp2},{exp3},{exp4})**

msolve 함수는 2 개의 행렬에서 표현된 식의 세트의 답을 계산하여 행렬의 한쪽을 덮어써서 결과를 저장합니다. 행렬은 태스크(P 인 경우에는 0, L 인 경우에는 1, Ddata 인 경우에는 2)에 대한 [Ldata.Control]의 설정에 따라서 연속 번호의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열의 배정도 부동 소수점('Sys.Ddata[i]') 변수의 세트에 따라서 형성됩니다.

최초의 식인 **exp1** 은 함수에 대한 입력으로 작용하는 상수 행렬(실제로는 열 벡터의 세트)의 시작 변수 번호를 지정합니다. 또 이것은 식 세트의 답을 포함하는 결과의 행렬(열 벡터의 세트)이 됩니다.

2 번째 식인 **exp2** 는 변수 계수를 포함하는 (정방)행렬의 시작 변수 번호를 지정합니다.

3 번째 식인 **exp3** 은 정방 변수 계수의 행렬 내의 행수와 열수를 지정합니다.

4 번째 식인 **exp4** 는 상수/결과의 행렬 내의 열수를 지정하기 때문에 답 구하는 식의 세트 수를 지정합니다. 이 행렬은 변수 계수 행렬과 같은 수의 행수를 가집니다. 단일 세트 식의 경우에 이 4 번째 식은 1 에 대해서 평가하기 때문에 상수/결과의 행렬은 실제로는 열 벡터가 됩니다.

msolve 함수는 각 세트가 다음과 같은 형식을 가지는 하나 또는 그 이상의 완전한 세트의 식을 푸는 경우에 유용합니다.

$$\bar{y} = A\bar{x}$$

여기서 A 는 n -by- n 정방 행렬에서 $\bar{y}$ 및 $\bar{x}$ 는 n 행의 열 벡터입니다. $\bar{y}$ 벡터(또는 벡터 세트)는 이 함수에 대해서 지정된 최초의 행렬에서 A 계수의 행렬은 지정된 2 번째 행렬입니다. 이것은 함수에 대한 입력값입니다. **msolve** 함수는 식의 세트에 대한 답을 구하는 $\bar{x}$ 벡터를 계산하고 이 답을 사용하여 입력의 $\bar{y}$ 벡터를 덮어씁니다.

식이 다음의 형식을 가지는 경우

$$\bar{y} = A\bar{x} + \bar{b}$$

vmadd 함수를 사용하여 $\bar{b}$ 벡터를 $\bar{y}$ 벡터에서 빼고 또 결과의 $(\bar{y} - \bar{b})$ 벡터를 **msolve** 함수에 대한 최초의 입력으로 사용해야 합니다.

함수의 반환값은 정방 계수 행렬의 행렬식입니다. 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 답의 유효성을 평가하는 경우에 유용합니다. 이 값이 정확하게 0 인 경우에 답은 무효가 됩니다. 또 0 에 매우 가까운 경우('매우 가까운 '의 정의는 어플리케이션에 따라서 다릅니다), 답은 문제가 남습니다.

예

```
P60=1000 P61=0
P62=0    P63=1000
P70=2
P71=3
P1=msolve(70,60,2,1)
P70,2
P70=0.002
P71=0.003
P1
P1=1000000(?)

P40=0.866 P41=-0.500 P42=0.000
P43=0.500 P44=0.866 P45=0.000
P46=0.000 P47=0.000 P48=1.000

P80=10.0 P81=7.071
P82=0.0 P83=7.071
P84=5.0 P85=1.000

P100=msolve(80,40,3,2)
```

mtrans

기능: 매트릭스 전치

구문: **mtrans({exp1},{exp2},{exp3},{exp4})**

mtrans 함수는 후속하는 괄호 안에 포함되는 식의 값에 따라서 지정되는 행렬의 전치를 계산하여 결과를 저장합니다. 소스 행렬과 결과의 행렬은 태스크(P 인 경우에는 0, L 인 경우에는 1, Ddata 인 경우에는 2)에 대한 [Ldata.Control]의 설정에 따라서 연속 번호의 글로벌('P') 변수, 로컬('L') 변수 또는 사용자 버퍼 배열의 배정도 부동 소수점('Sys.Ddata[i]') 변수의 세트에 따라서 형성됩니다.

최초의 식인 **exp1** 은 결과(전치)의 행렬의 시작 변수 번호를 지정합니다.

2 번째 식인 **exp2** 는 소스 행렬의 시작 변수 번호를 지정합니다.

3 번째 식인 **exp3** 은 결과의 행렬 내의 행의 수를 지정하기 때문에 소스 행렬 내의 열의 수를 지정합니다.

4 번째 식인 **exp4** 는 결과의 행렬 내의 열의 수를 지정하기 때문에 소스 행렬 내의 행의 수를 지정합니다.

이 함수의 반환값은 올바르게 전치된 것을 나타내는 부울 형식의 플래그입니다. 올바르게 전치된 경우에 이것이 1 로 설정되고 올바르게 전치되지 않은 경우에는 0 으로 설정됩니다(올바르게 전치되지 않은 경우에 보통은 행 또는 열의 수가 제로 또는 음의 값이 되는 범위 외의 값이 지정됩니다). 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

예

```
P40=1 P41=3 P42=5
P43=7 P44=9 P45=11
P46=mtrans(50,40,3,2)
P46
P46=1
P50,6
P50=1
P51=7
P52=3
P53=9
P54=5
P55=11
```

변환 행렬 수학 함수

Power PMAC 스크립트 언어에는 축 변환 데이터 구조체에서 동작하는 함수 세트가 있습니다. 이 함수는 회전, 오프셋, 스케일링 등의 각종 변환을 가능하게 하여 신속하고 간단하게 탑재 및 조합할 수 있습니다.

tinit

기능: 변환 행렬의 초기화

구문: **tinit({exp1})**

tinit 함수는(**exp1**의 값으로 지정된 인덱스를 가짐) 축 변환 행렬 데이터 구조체 **[Tdata[i]]**를 초기화합니다. 초기화에서는 모든 대각 요소 **[Tdata[i].Diag[m]]**을 1.0으로 설정하고 모든 오프셋 요소 **[Tdata[i].Bias[m]]**을 0.0으로 설정합니다. 또 모든 비대각 요소 **[Tdata[i].UVW[m]]**, **[Tdata[i].XYZ[m]]**, **[Tdata[i].UUUVWW[m]]** 및 **[Tdata[i].XXYYZZ[m]]**을 0.0으로 설정합니다. 이것으로 축 변환 행렬을 항등 행렬로 하여 실제로 변환되지 않습니다. 이것은 이미 알고 있는 시작 상태에서 행렬을 취득하는 경우에 유용합니다.

이 함수의 반환값은 초기화 성공을 나타내는 부울 형식 플래그로 취급할 수 있습니다. 유효한 변환 행렬의 인덱스 번호가 선택된 경우에 Power PMAC은 이 행렬의 행렬식을 계산하고 그것이 1.0과 같은지 조사합니다. 무효한 변환 행렬의 인덱스 번호가 선택된 경우에 Power PMAC은 반환값을 0.0으로 설정하고 연산이 실패했다는 것을 나타냅니다. 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

tprop

기능: 변환 행렬의 전송(조합)

구문: **tprop({exp1},{exp2},{exp3})**

tprop(변환의 전송) 함수는 2개의 축 변환 행렬의 변환을 조합하여 그 결과를 3번째 축 변환 행렬에 둡니다.

결과의 변환 행렬은 인덱스가 **exp1**로 지정된 데이터 구조체 **[Tdata[i]]**입니다. 이 결과의 변환 행렬은 조합하는 2개의 행렬 중 하나로 같은 행렬로 할 수 있습니다.

조합하는 최초의 변환 행렬은 인덱스가 **exp2**로 지정된 데이터 구조체 **[Tdata[i]]**입니다(변환 행렬의 조합은 가환이 아님을 잊지 마십시오).

조합하는 2번째 변환 행렬은 인덱스가 **exp3**으로 지정된 데이터 구조체 **[Tdata[i]]**입니다.

최초의 축 변환의 정방 회전 행렬은 2번째 축 변환의 정방 회전 행렬에서 곱해져 결과의 회전 행렬이 결과의 축 변환에 써집니다. 다음으로 최초의 축 변환의 정방 회전 행렬이 2번째 축 변환의 바이어스의 열 오프셋 벡터에 따라서 곱해집니다. 다음으로 결과의 열 벡터가 최초의 변환 오프셋 벡터에 더해져 결과의 합벡터가 결과의 축 변환의 오프셋 벡터에 써집니다.

조합은 2 개의 확장된 동차 행렬의 직접 곱셈으로 생각할 수 있습니다. 여기서 동차 행렬은 오프셋 벡터를(우측의 추가 열로) 정방 회전 행렬에 추가합니다. 또한 추가 행이 오리지널 회전 행렬 아래에서 모두 0의 값을 가지는 저부에 추가되며 또한 오프셋 벡터의 행 아래에서 값 1을 가지는 저부에 추가됩니다.

이 함수의 반환값은 결과의 축 변환 회전 행렬의 행렬식입니다. 0.0의 값은 인덱스 번호가 범위 외 또는 FAIL-SOFT한 결과의 변환이 원인이 되어 결과의 행렬이 무효가 된 것을 나타냅니다. 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

EtherCAT 네트워크 함수

다음 함수가 EtherCAT 네트워크의 데이터에 대한 액세스에 사용할 수 있습니다.

ecatsdo

기능: EtherCAT의 SDO(서비스 데이터 객체)에 액세스 함

구문: **ecatsdo({쓰기},{읽기},{쓰기},{읽기},{쓰기},{읽기})**

ecatsdo 함수는 프로그램 내에서 지정된 EtherCAT SDO(서비스 데이터 객체)에 대한 액세스(쓰기 또는 읽기)를 할 수 있습니다. 이 함수는 6개의 인수를 가집니다.

- 최초의 인수는 액세스의 방향을 지정합니다. SDO에 대한 쓰기의 경우에는 인수를 0으로 해야 하며 SDO에서 읽기인 경우에는 1로 해야 합니다.
- 2번째 인수는 장치의 슬레이브 인덱스값 'j'로 이것은 장치에 할당된 구조체 [ECAT[j].Slave[j]]에 대응합니다.
- 3번째 인수는 SDO의 인덱스 번호입니다.
- 4번째 인수는 SDO의 하위 인덱스 번호입니다.
- 5번째 인수는 SDO의 데이터 값입니다. 함수가 SDO에 쓰기에 사용된 경우(최초의 인수=0), 이것은 써지는 값이 됩니다. 함수가 SDO에서 읽기에 사용된 경우(최초의 인수=1), 이 값은 사용되지 않습니다(사용되지 않은 경우에도 함수를 실행하기 위해서 여기에는 임의의 값이 있어야 합니다).
- 6번째 인수는 장치의 마스터 인덱스값 'i'(EtherCAT 네트워크 인덱스 등)로 이것은 장치에 할당된 구조체 [ECAT[i].Slave[j]]에 대응합니다.

이 함수의 반환값은 읽기 액세스가 유효한 경우에 지정된 SDO의 데이터 값입니다. 읽기 액세스가 무효인 경우에 반환값은 NaN(숫자가 아님)이 됩니다.

유효한 쓰기 액세스인 경우에 이 함수의 반환값은 0이 됩니다. 쓰기 액세스가 무효인 경우에 반환값은 NaN(숫자가 아님)이 됩니다. 반환값이 어플리케이션 내에서 실제로 사용되지 않는 경우에도 어느 한곳에 반드시 저장해야 합니다. 예를 들어 다음 코딩 예시에 나타난 것처럼 더미 변수에 할당하여 저장하십시오.

예

```
global sdoresult;
sdoresult = ecatsdo(0,0,$6060,0,10,0); // Write to CAN object
sdoresult = ecatsdo(1,0,$6061,0,0,0); // Read from CAN object
```

문자열 함수

Power PMAC 스크립트 언어에는 문자열 조작 함수의 완전한 세트가 있습니다. 이것으로 텍스트 문자열을 작성, 조작 및 처리할 수 있습니다. Power PMAC 내에서 문자열은 사용자의 공유 메모리 버퍼 내의 문자(바이트) 배열(Sys.Cdata[i])로 취급되고 사용자 공유 메모리 버퍼의 시작의 공유(바이트) 주소 오프셋(인덱스)에 따라서 참조하십시오

sprintf

기능: 서식화된 텍스트 문자열을 작성함

구문: **sprintf({식},{문자열}[, {식}...])**

sprintf 함수는 수식의 수치를 서식화된 문자열로 변환합니다.

최초의 **{식}**은 목표의 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정하여 서식화된 텍스트 형식의 값을 거기에 둡니다.

큰따옴표 중의 **{문자열}**은 서식화된 ASCII 텍스트 문자열로 이것은 리터럴 영숫자 문자, 이스케이프 시퀀스 및 서식화된 변수 시퀀스에서 구성됩니다.

다음 이스케이프 시퀀스를 사용할 수 있습니다(전부 소문자).

- \a 경고음(ASCII 7)
- \b 백스페이스(ASCII 8)
- \t 가로 탭(ASCII 9)
- \n 줄 바꿈(ASCII 10)
- \v 세로 탭(ASCII 11)
- \f 폼 피드(ASCII 12)
- \r 캐리지 리턴(ASCII 13)
- \\ 백슬래시 문자
- \? 물음표
- \' 작은따옴표
- \" 큰따옴표
- \ooo ASCII 문자 코드의 8 진법(000~377 범위)
- \xhh ASCII 문자 코드의 16 진법(x00~xff 범위)

(Null 종단 문자('\0' 또는 '\x0')를 문자열에 명시적으로 삽입한 경우에 그 이후의 문자열 변수는 문자열의 끝으로 취급됩니다. Power PMAC 은 사용자 지정 텍스트 뒤에 Null 종단 문자를 자동으로 추가하지 않으니 주의하십시오.)

다음의 포맷 시퀀스를 사용할 수 있습니다(대·소문자 여부 본문 기재).

- %d 부호가 있는 정수, 10 진법 형식
- %u 부호가 없는 정수, 10 진법 형식
- %x 부호가 없는 정수, 16 진법 형식, 소문자 사용
- %nx 부호가 없는 정수, 16 진법 형식, n 자리 사용(n = 1~8), 소문자 사용
- %X 부호가 없는 정수, 16 진법 형식, 대문자 사용
- %nX 부호가 없는 정수, 16 진법 형식, n 자리 사용(n = 1~8), 대문자 사용
- %f 부동 소수점 값, 최대 6 자리까지

- `%nf` 부동 소수점 값, 최대 n 자리까지($n = 1\sim31$)
- `%n.mf` 부동 소수점 값 최대 n 자릿수까지($n = 1\sim31$), 소수 최대 m 자릿수($m < n$)
- `%s` 문자열, 임의의 길이(Null 종단됨)
- `%ns` 문자열, 최대 n 문자(Null 종단 문자가 있는 경우에는 이보다 짧음)
- `%c` 지정된 바이트값의 1 문자
- `%nc` 지정된 바이트값의 1 문자의 n 회 반복
- `%%` 퍼센트 문자

선행 제로열은 10 진법 정수로는 작성되지 않습니다. 자릿수가 지정된 경우에만 선행 제로열이 16 진법 정수로 작성됩니다. 자릿수 지정 여부와 관계없이 선행 및 끝 제로열은 부동 소수점 값으로 작성되지 않습니다. 부동 소수점 값에 지정된 자릿수가 지수가 없는 값으로 나타내기 불충분한 경우에는 자동으로 지수가 붙습니다(2.345e8 등). 0 이 아닌 소수 부분이 있는 경우에 소수점은 부동 소수점 값만 사용됩니다. 부동 소수점 값의 소수점, 음의 부호, 지수는 자릿수로 세지 않습니다.

문자열이 포맷된 각 변수 시퀀스에는 문자에 변환되는 수치를 지정하는 `{식}`이 포함됩니다. 식은 단순히 하나의 상수 또는 변수인 경우도 있지만 여러 개의 수학 연산자나 식을 포함하는 경우도 있습니다. 문자열 변수에는 식의 값(필요에 따라 다음 정수로 버림)이 사용되는 문자열 변수의 사용자 공유 메모리 내에서의 시작 인덱스를 나타냅니다.

Power PMAC 은 Null 종단 문자(ASCII 코드 0)를 작성한 문자열의 끝에 자동으로 추가합니다.

정상적으로 실행된 뒤에 함수 반환값은 문자열 변수 중 문자 수로 Null 종단 문자는 포함되지 않습니다. 함수가 정상적으로 실행되지 않은 경우에 -1 의 값이 반환됩니다. 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

예

```
EndIndex = sprintf(256, "%d", CycleNum)
EndIndex = sprintf(256, "Cycle # %d executing\n", CycleNum)
EndIndex = sprintf(MaskStringIndex, "%6x", OutputMask)
EndIndex = sprintf(XposStringIndex, "X:%9.4f", Motor[1].DesPos)
```

strcat

기능: 문자열 변수의 끝에 추가함

구문: `strcat({식},{식})`

`strcat({식},"{문자열}")`

strcat 함수는 제한된 길이의 텍스트 문자열을 문자열 변수 끝에 추가합니다. 이 문자열은 다른 문자열 변수도 가능하지만 직접 지정할 수도 있습니다.

최초의 `{식}`은(텍스트 문자열을 추가하는) 목표 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

쉼표 뒤에 2 번째 **{식}**이 있는 경우에 이 식에서(텍스트 문자열을 추가하는) 소스 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다. 추가는 이 인덱스에서 시작되어 Null 종단 문자가 검출될 때까지 계속합니다. 필요한 장소에 Null 종단 문자가 존재하지 않는 경우에는 매우 긴 바이트 시퀀스가 목표 문자열에 추가되어 사용자 공유 메모리 버퍼 내의 다른 요소를 덮어씁니다. 관련하는 **strncat** 함수는 추가 가능한 최대 문자 수를 제한하기 때문에 이 점에서 매우 안전합니다.

쉼표 뒤에 큰따옴표가 있는 리터럴 텍스트 문자열이 있는 경우에 이 문자열은 목표 변수의 끝에 추가됩니다. 또 마지막 문자 뒤에 Null 문자가 자동으로 추가됩니다. 텍스트 문자열에는 표준 텍스트 문자나 다음의 **sprintf** 함수의 형에서 나타내는 이스케이프 시퀀스를 포함할 수 있지만 **sprintf** 함수에서 허용되는 서식화된 변수 시퀀스는 포함할 수 없습니다.

정상적으로 실행된 뒤에 함수의 반환값은 현재는 목표 변수 중 문자열을 종료시키는 Null 문자의 사용자 공유 메모리 내의 인덱스입니다. 함수가 정상적으로 실행되지 않은 경우에 -1의 값이 반환됩니다. 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

예

```
EndIndex = strncat(256, 384)
EndIndex = strncat(256, 384+16*MsgNum)
EndIndex = strncat(800, "due to oscillation")
```

strchr

기능: 문자열 변수 중에서 문자의 최초의 인스턴스를 검색함

구문: **strchr({식},{식})**
strchr({식},'문자')

strchr 함수는 문자열 변수 중에 문자가 존재한다는 것을 통지하고 또 문자의 최초의 위치를 통지합니다. 문자는 그 ASCII 코드값을 사용하여 지정 또 직접 지정할 수 있습니다.

최초의 **{식}**은 검색 대상의 문자열 변수 사용자 공유 메모리 내의 시작 인덱스를 지정합니다. 문자열은 이 인덱스에서 다음의 Null 문자까지의 사이에서 검색됩니다.

쉼표 뒤에 2 번째 **{식}**이 있는 경우에 이 식으로 검색 대상의 문자인 ASCII 값을 지정합니다. 이 값 하위 바이트만 사용되기 때문에 식이 0~255 범위 외에서 평가한 경우에 그 범위 내에서 롤 오버가 발생합니다.

쉼표 뒤의 작은따옴표가 있는 리터럴 문자가 있는 경우에는 문자열 변수 중에서 검색하는 문자를 지정합니다. 버퍼 프로그램 내에서 이런 방법에 따라 지정된 함수의 반환값은 ASCII 코드 값으로 통지되므로 주의하십시오.

정상적인 실행 후의 함수 반환값은 지정된 문자의 최초의 인스턴스 사용자 공유 메모리 내의 인덱스입니다. 지정된 문자를 못 찾아서 함수가 정상적으로 실행되지 않은 경우에는 -1의 값이 반환됩니다.

예

```
CharIndex = strchr(256, 65)
CharIndex = strchr(256, 64+LtrNum)
CharIndex = strchr(256, 'A')
```

strcmp

기능: 2 개의 문자열 변수를 비교함

구문: **strcmp({식},{식})**
strcmp({식},"{문자열}")

strcmp 함수는 문자열 변수의 내용을 텍스트 문자열과 비교합니다. 이 2 번째 문자열은 다른 문자열 변수도 가능하지만 직접 지정을 할 수도 있습니다.

최초의 **{식}**은 비교 대상의 문자열 변수 사용자 공유 메모리 내의 시작 인덱스를 지정합니다. 문자열은 이 인덱스에서 다음의 Null 문자 사이에서 비교됩니다.

쉼표 뒤에 2 번째 **{식}**이 있는 경우에 이 식에서 비교 대상의 2 번째 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다. 문자열은 이 인덱스에서 다음의 Null 문자 사이에서 비교됩니다.

쉼표 뒤의 큰따옴표가 있는 리터럴 텍스트 문자열이 있는 경우에 이 문자열은 최초의 문자열 변수값과 비교됩니다. 텍스트 문자열에는 표준 텍스트 문자나 다음의 **sprintf** 함수의 형에서 나타내는 이스케이프 시퀀스를 포함할 수 있지만 **sprintf** 함수에서 허용되는 서식화된 변수 시퀀스는 포함할 수 없습니다.

2 개의 문자열이 정확하게 같은 경우에 함수의 반환값은 0 이 됩니다. 그것이 정확하게 같지 않은 경우에 반환값은 2 개의 문자열의 차가 됩니다. 최초의 문자열에서 2 번째 문자열을 뺀 결과는 정수값으로 취급됩니다.

예

```
StrDiff = strcmp(256, 384)
StrDiff = strcmp(256, 512+32*CmdNum)
if(!(strcmp(HostCmdIndex, "Insert Part") { ...
```

strcpy

기능: 문자열 변수를 복사함

구문: **strcpy({식},{식})**
strcpy({식},"{문자열}")

strcpy 함수는 텍스트 문자열을 문자열 변수에 복사합니다. 이 문자열은 다른 문자열 변수도 가능하지만 직접 지정할 수도 있습니다.

최초의 **{식}**은(텍스트 문자열을 복사하는) 목표 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

쉼표 뒤에 2 번째 **{식}**이 있는 경우에 이 식에서(텍스트 문자열을 복사하는) 소스 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다. 복사는 이 인덱스에서 시작되어 Null 종단 문자가 검출될 때까지 계속합니다. 필요한 장소에 Null 종단 문자가 존재하지 않는 경우에는 매우 긴 바이트 시퀀스가 목표 문자열에 복사되어 사용자 공유 메모리 버퍼 내의 다른 요소를 덮어씁니다. 관련하는 **strcpy** 함수는 복사 가능한 최대 문자 수를 제한하기 때문에 이 점에서 매우 안전합니다.

쉼표 뒤의 큰따옴표가 있는 리터럴 텍스트 문자열이 있는 경우에 이 문자열은 목표 변수에 복사됩니다. 또 마지막 문자 뒤에 Null 문자가 자동으로 추가됩니다. 텍스트 문자열에는 표준 텍스트 문자나 다음의 **sprintf** 함수의 항에서 나타내는 이스케이프 시퀀스를 포함할 수 있지만 **sprintf** 함수에서 허용되는 서식화된 변수 시퀀스는 포함할 수 없습니다.

정상적으로 실행된 뒤에 함수의 반환값은 목표 변수 중 문자열을 종료시키는 Null 문자의 사용자 공유 메모리 내의 인덱스입니다. 함수가 정상적으로 실행되지 않은 경우에 -1의 값이 반환됩니다. 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

예

```
EndIndex = strcpy(256, 384)
EndIndex = strcpy(256, 384+16*MsgNum)
EndIndex = strcpy(800, "Process timed out")
```

strcspn

기능: 문자열 변수 중에서 지정 문자를 포함하지 않는 가장 긴 종속 문자열을 검색함

구문: **strcspn({식},{식})**
strcspn({식},"{문자열}")

strcspn 함수는 그 위치에서 시작하여 다른 문자열에 포함되지 않는 문자만으로 구성되는 문자열의 최장 부분의 길이를 찾아냅니다.

최초의 **{식}**은 이 부분의 길이를 판단하는 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

쉼표 뒤에 2 번째 **{식}**이 있는 경우에 이 식에서(최초의 문자열 변수 내에서 존재의 유무를 확인하는) 문자를 포함하는 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

쉼표 뒤의 큰따옴표가 있는 리터럴 텍스트 문자열이 있는 경우에 이것에는 최초의 문자열 변수 내에서 존재의 유무를 확인하는 문자가 포함됩니다. 텍스트 문자열에는 표준 텍스트 문자나 다음의 **sprintf** 함수의 항에서 나타내는 이스케이프 시퀀스를 포함할 수 있지만 **sprintf** 함수에서 허용되는 서식화된 변수 시퀀스는 포함할 수 없습니다.

반환값은 그 위치에서 시작하여 2 번째 문자열(변수 또는 리터럴)에 포함되지 않는 문자만으로 구성되는 최초의 문자열 변수의 종속 문자열의 길이입니다.

예

```
SubLen = strcspn(256, 384)
SubLen = strcspn(256, 512+32*CmdNum)
SubLen = strcspn(RecipeIndex, "XYZ")
```

strlen

기능: 문자열 변수 중 문자 수를 찾는다

구문: **strlen({식})**

strlen 함수는 문자열 변수의 길이를 찾아냅니다.

{식}은 길이를 판단하는 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

반환값은 문자열 변수 중 문자 수이며 Null 종단 문자는 포함되지 않습니다.

예

```
Len = strlen(256)
Len = strlen(RecipeIndex)
```

strncat

기능: 지정한 문자열을 문자열 변수 끝에 추가함

구문: **strncat({식},{식},{식})**

strncat 함수는 텍스트 문자열의 변수를 다른 문자열 변수 끝에 추가합니다.

최초의 **{식}**은(텍스트 문자열을 추가하는) 목표 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

최초의 쉼표 뒤의 2번째 **{식}**은 이 식에서(텍스트 문자열을 추가하는) 소스 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다. 추가는 이 인덱스에서 시작되어 Null 종단 문자가 검출될 때까지 또는 맨 마지막 **{식}**에서 지정된 문자 수가 추가될 때까지 이 두 가지 경우 중에 하나가 처음에 발생할 때까지 계속됩니다. 이 함수에서 문자 수를 제한하는 경우에 Null 종단 문자가 목표 변수에 자동으로 추가됩니다.

정상적으로 실행된 뒤에 함수의 반환값은 현재는 목표 변수 중 문자열을 종료시키는 Null 문자의 사용자 공유 메모리 내의 인덱스입니다. 함수가 정상적으로 실행되지 않은 경우에 -1의 값이 반환됩니다. 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

예

```
EndIndex = strncat(256, 384, 8)
EndIndex = strncat(256, 384+16*MsgNum, 12)
```

strncmp

기능: 지정한 길이의 2 개의 문자열 변수를 비교함

구문: **strncmp({식},{식},{식})**

strncmp 함수는 지정한 길이의 2 개의 문자열 변수의 내용을 비교합니다.

최초의 **{식}**은 비교 대상의 문자열 변수 사용자 공유 메모리 내의 시작 인덱스를 지정합니다. 문자열의 비교는 이 인덱스에서 시작되어 Null 종단 문자가 검출될 때까지 또는 마지막 **{식}**에서 지정된 문자 수가 모두 비교될 때까지 이 두 가지 경우 중에 하나가 처음에 발생할 때까지 계속됩니다.

최초의 쉼표 뒤의 2 번째 **{식}**은 비교 대상의 2 번째 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다. 문자열의 비교는 이 인덱스에서 시작되어 Null 종단 문자가 검출될 때까지 또는 마지막 **{식}**에서 지정된 문자 수가 모두 비교될 때까지 이 두 가지 경우 중에 하나가 처음에 발생할 때까지 계속됩니다.

2 개의 지정한 문자열이 정확하게 같은 경우에 함수의 반환값은 0 이 됩니다. 그것이 정확하게 같지 않은 경우에 반환값은 2 개의 문자열의 차가 됩니다. 최초의 문자열에서 2 번째 문자열을 뺀 결과는 정수값으로 취급됩니다.

예

```
StrDiff = strncmp(256, 384, 8)
StrDiff = strncmp(256, 512+32*CmdNum, CmdCoreLen)
```

strncpy

기능: 지정한 문자열 변수를 복사함

구문: **strncpy({식},{식},{식})**

strncpy 함수는 지정한 길이의 텍스트 문자열의 변수를 다른 문자열 변수에 복사합니다.

최초의 **{식}**은(텍스트 문자열을 복사하는) 목표 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

최초의 쉼표 뒤의 2 번째 **{식}**은 이 식에서 (텍스트 문자열을 복사하는)소스 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다. 복사는 이 인덱스에서 시작되어 Null 종단 문자가 검출될 때까지 또는 마지막 **{식}**에서 지정된 문자 수가 복사될 때까지 이 두 가지 경우 중에 하나가 처음에 발생할 때까지 계속됩니다. 함수에서 문자 수를 제한하는 경우에 Null 종단 문자가 목표 변수에 자동으로 추가됩니다.

정상적으로 실행된 뒤에 함수의 반환값은 목표 변수 중 문자열을 종료시키는 Null 문자의 사용자 공유 메모리 내의 인덱스입니다. 함수가 정상적으로 실행되지 않은 경우에 -1 의 값이 반환됩니다. 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

예

```
EndIndex = strncpy(256, 384, 8)
EndIndex = strncpy(256, 384+16*MsgNum, 12)
```

strpbrk

기능: 문자열 변수 중에서 지정된 어느 문자의 최초의 출현을 찾아낸다

구문: **strpbrk({식},{식})**
strpbrk({식},{문자열})

strpbrk 함수는 다른 문자열에 포함되는 어느 문자가 문자열 변수에서 최초의 출현하는 위치를 찾아냅니다.

최초의 **{식}**은 문자의 출현을 판단하는 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

쉼표 뒤에 2 번째 **{식}**이 있는 경우에 이 식에서(최초의 문자열 변수 내에서 존재의 유무를 확인하는) 문자를 포함하는 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

쉼표 뒤의 큰따옴표가 있는 리터럴 텍스트 문자열이 있는 경우에 이것에는 최초의 문자열 변수 내에서 존재의 유무를 확인하는 문자가 포함됩니다. 텍스트 문자열에는 표준 텍스트 문자나 다음의 **sprintf** 함수의 형에서 나타내는 이스케이프 시퀀스를 포함할 수 있지만 **sprintf** 함수에서 허용되는 서식화된 변수 시퀀스는 포함할 수 없습니다.

반환값은 2 번째 문자열(변수 또는 리터럴) 중 어느 문자의 최초의 문자열 변수 내에서 처음에 출현한 사용자 공유 메모리 내의 인덱스입니다.

예

```
CharIndex = strpbrk(256, 384)
CharIndex = strpbrk(256, 512+32*CmdNum)
CharIndex = strpbrk(RecipeIndex, "XYZ")
```

strchr

기능: 문자열 변수 중에서 마지막에 출현한 문자를 검색함

구문: **strchr({식},{식})**
strchr({식},{문자})

strchr 함수는 문자열 변수 중에 문자가 존재한다는 것을 통지하고 또 문자의 마지막 위치를 통지합니다. 문자는 그 ASCII 코드값을 사용하여 지정 또 직접 지정할 수 있습니다.

최초의 **{식}**은 검색 대상의 문자열 변수 사용자 공유 메모리 내의 시작 인덱스를 지정합니다. 문자열은 이 인덱스에서 다음의 Null 문자까지의 사이에서 검색됩니다.

쉼표 뒤에 2 번째 **{식}**이 있는 경우에 이 식으로 검색 대상의 문자인 ASCII 값을 지정합니다. 이 값 하위 바이트만 사용되기 때문에 식이 0~255 범위 외에서 평가한 경우에 그 범위 내에서 롤 오버가 발생합니다.

쉼표 뒤의 작은따옴표가 있는 리터럴 문자가 있는 경우에는 문자열 변수 중에서 검색하는 문자를 지정합니다. 버퍼 프로그램 내에서 이런 방법에 따라 지정된 함수가 목록에 반환된 경우에 이것은 문자인 ASCII 코드 값으로 통지되므로 주의하십시오.

정상적인 실행 후에 함수의 반환값은 지정된 문자가 마지막에 출현한 사용자 공유 메모리 내의 인덱스입니다. 지정된 문자를 못 찾아서 함수가 정상적으로 실행되지 않은 경우에는 -1의 값이 반환됩니다.

예

```
CharIndex = strrchr(256, 65)
CharIndex = strrchr(256, 64+LtrNum)
CharIndex = strrchr(256, 'A')
```

strspn

기능: 문자열 변수 중에서 지정된 문자의 최장 종속 문자열을 검색함

구문: **strspn({식},{식})**
strspn({식},"{문자열}")

strspn 함수는 그 위치에서 시작하여 다른 문자열에 포함되는 문자만으로 구성되는 문자열의 최장 부분의 길이를 찾아냅니다.

최초의 **{식}**은 이 부분의 길이를 판단하는 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

쉼표 뒤에 2 번째 **{식}**이 있는 경우에 이 식에서(최초의 문자열 변수 내에서 존재의 유무를 확인하는) 문자를 포함하는 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

쉼표 뒤의 큰따옴표가 있는 리터럴 텍스트 문자열이 있는 경우에 이것에는 최초의 문자열 변수 내에서 존재의 유무를 확인하는 문자가 포함됩니다. 텍스트 문자열에는 표준 텍스트 문자나 다음의 **sprintf** 함수의 항에서 나타내는 이스케이프 시퀀스를 포함할 수 있지만 **sprintf** 함수에서 허용되는 서식화된 변수 시퀀스는 포함할 수 없습니다.

반환값은 그 위치에서 시작하여 2 번째 문자열(변수 또는 리터럴)에 포함되는 문자만으로 구성되는 최초의 문자열 변수 중 종속 문자열의 길이입니다.

예

```
SubLen = strspn(256, 384)
SubLen = strspn(256, 512+32*CmdNum)
SubLen = strspn(RecipeIndex, "XYZ")
```

strstr

기능: 문자열 변수 중에서 다른 문자열의 최초의 출현을 찾아냄

구문: **strstr({식},{식})**

strstr({식},"{문자열}")

strstr 함수는 다른 문자열의 문자열 변수 내의 최초의 출현을 찾아냅니다.

최초의 **{식}**은 다른 문자열의 출현을 판단하는 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

쉼표 뒤에 2 번째 **{식}**이 있는 경우에 이 식에서 (최초의 문자열 변수 내에서 존재의 유무를 확인하는)내용을 포함하는 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

쉼표 뒤의 큰따옴표가 있는 리터럴 텍스트 문자열이 있는 경우에 이것에는 최초의 문자열 변수 내에서 존재의 유무를 확인하는 문자가 포함됩니다. 텍스트 문자열에는 표준 텍스트 문자나 다음의 **sprintf** 함수의 항에서 나타내는 이스케이프 시퀀스를 포함할 수 있지만 **sprintf** 함수에서 허용되는 서식화된 변수 시퀀스는 포함할 수 없습니다.

반환값은 2 번째 문자열(변수 또는 리터럴)의 임의 문자의 최초의 문자열 변수 내에서 처음에 출현한 사용자 공유 메모리 내의 인덱스입니다.

예

```
CharIndex = strstr(256, 384)
CharIndex = strstr(256, 512+32*CmdNum)
CharIndex = strstr(RecipeIndex, "XYZ")
```

strtod

기능: 문자열 텍스트를 수치 변수값으로 변환함

구문: **strtod({식},{식})**

strtod 함수는 소수점 이하의 수치를 나타내는 텍스트 문자열을 해당 수치로 변환하고 그 값을 반환합니다.

최초의 **{식}**은 수치를 도출하는 소스 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다. Power PMAC에서는 수치의 일부로 해석할 수 없는 문자가 검출될 때까지 또는 Null 종단 문자가 검출될 때까지 모든 문자가 해당 인덱스에서 사용됩니다. 수치에 관한 텍스트는 PMAC에 대한 명령 내에서 수치 상수값의 지정에 유효한 임의의 서식을 가질 수 있습니다. 이에에는 과학적 기수법([1.234e8] 등)이나 16 진법 형식([\$3FFF] 등)이 포함됩니다.

최초의 쉼표 뒤의 2 번째 **{식}**은 프로그램이나 명령 스레드에 대해서 문자열의 종료를 나타내는 Null 문자 사용자 공유 메모리 내의 인덱스를 넣는 로컬('L') 변수 번호를 지정합니다. 함수가 지정된 문자열에서 임의의 수치를 해석할 수 없는 경우에 그 변수는 -1의 값을 가집니다.

반환값은 지정된 문자열에서 도출되어 해석된 수치입니다. 이것은 배정도 부동 소수점의 값으로 계산되기 때문에 임의의 범용 사용자 변수(로컬 또는 글로벌)에 대한 저장을 포함하여 그 이상 변환하지 않고 후속하는 임의의 프로그램 계산에서 직접 사용할 수 있습니다. 다른 서식의 변수 또는 요소에 저장하는 경우에 Power PMAC 은 필요한 형식으로 자동으로 변환합니다. 함수가 지정된 문자열에서 임의의 수치를 해석할 수 없는 경우에 반환값은 0 이 됩니다(단, 지정된 로컬 변수값에 따라서 올바르게 해석되었다는 것을 판단해야 합니다).

예

```
MyVar = strtod(256, 0)
InchLength = 25.4 * strtod(500, EndPtr)
```

strtolower

기능: 대문자를 소문자로 변환하여 지정된 문자열 변수를 복사함

구문: **strtolower**{식},{식},{식}

strtolower 함수는 지정된 길이의 문자열 변수에 대해서 그중 대문자(A~Z)를 등가의 소문자(a~z)로 변환하여 그 문자열 변수를 복사합니다.

최초의 {식}은(텍스트 문자열을 복사하는) 목표 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

최초의 쉼표 뒤의 2 번째 {식}은 이 식에서 (텍스트 문자열을 복사하는)소스 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다. 복사는 이 인덱스에서 시작되어 Null 종단 문자가 검출되거나 또는 마지막 {식}에서 지정된 문자 수가 복사되는 두 가지 경우 중에 하나가 처음에 발생할 때까지 계속됩니다. 함수에서 문자 수를 제한하는 경우에 Null 종단 문자가 목표 변수에 자동으로 추가됩니다.

소스와 목표 문자열 변수를 동일하게 할 수 있으며 이 경우에 함수는 문자열 변수를 변환할 수 있습니다.

정상적으로 실행된 뒤에 함수의 반환값은 목표 변수 중 문자열을 종료시키는 Null 문자의 사용자 공유 메모리 내의 인덱스입니다. 함수가 정상적으로 실행되지 않은 경우에 -1 의 값이 반환됩니다.

예

```
EndIndex = strtolower(256, 384, 8)
EndIndex = strtolower(256, 256, 8)
EndIndex = strtolower(256, 384+16*MsgNum, 12)
```

strtoupper

기능: 소문자를 대문자로 변환하여 지정 한 문자열 변수를 복사함

구문: **strtoupper({식},{식},{식})**

strtoupper 함수는 지정된 길이의 문자열 변수에 대해서 그중 소문자(a~z)를 등가의 대문자(A~Z)로 변환하여 그 문자열 변수를 복사합니다.

최초의 {식}은(텍스트 문자열을 복사하는) 목표 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다.

최초의 쉼표 뒤의 2 번째 {식}은 이 식에서 (텍스트 문자열을 복사하는)소스 문자열 변수의 사용자 공유 메모리 내의 시작 인덱스를 지정합니다. 복사는 이 인덱스에서 시작되어 Null 종단 문자가 검출되거나 또는 마지막 {식}에서 지정된 문자 수가 복사되는 두 가지 경우 중에 하나가 처음에 발생할 때까지 계속됩니다. 함수에서 문자 수를 제한하는 경우에 Null 종단 문자가 목표 변수에 자동으로 추가됩니다.

소스와 목표 문자열 변수를 동일하게 할 수 있으며 이 경우에 함수는 문자열 변수를 변환할 수 있습니다.

정상적으로 실행된 뒤에 함수의 반환값은 목표 변수 중 문자열을 종료시키는 Null 문자의 사용자 공유 메모리 내의 인덱스입니다. 함수가 정상적으로 실행되지 않은 경우에 -1 의 값이 반환됩니다.

예

```
EndIndex = strtoupper(256, 384, 8)
EndIndex = strtoupper(256, 256, 8)
EndIndex = strtoupper(256, 384+16*MsgNum, 12)
```

문자 버퍼 함수

Power PMAC에는 사용자 공유 메모리 버퍼 내의 문자 배열에 쓰기를 실행하는 2개의 함수가 있습니다. 이 함수는 문자열 변수에 대한 배열 방법을 배려하지 않고 문자 바이트에 직접 액세스 합니다.

memcpy

기능: 문자 버퍼 메모리 내용을 복사함

구문: **memcpy({식},{식},{식})**

memcpy 함수는 사용자 공유 메모리 내에서 하나 또는 그 이상의 연속하는 바이트 위치 내용을 임의의 문자 값에 강제로 복사합니다.

최초의 {식}은 복사 대상인 레지스터 사용자 공유 메모리 내의 (시작)인덱스를 지정합니다. 이 인덱스는 사용자 버퍼의 시작에서 주소 오프셋입니다.

최초의 쉼표 뒤의 2 번째 {식}은 복사원의 레지스터 사용자 공유 메모리 내의 (시작)인덱스를 지정합니다. 이 인덱스는 사용자 버퍼의 시작에서 주소 오프셋입니다.

마지막 {식}은 복사 대상의 메모리 바이트 수를 지정합니다.

정상적인 실행 후에 함수의 반환값은 값을 복사한 최초의 레지스터의 사용자 공유 메모리 내의 인덱스입니다. 함수가 정상적으로 실행되지 않은 경우에 -1 의 값이 반환됩니다. 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

예

```
CharIndex = memset(256, 65, 1)
CharIndex = memset(256, 64+LtrNum, 8)
CharIndex = memset(256, 'A', 1)
```

memset

기능: 문자 버퍼 메모리를 지정된 문자의 값에 설정함

구문: **memset({식},{식},{식})**

memset({식},{문자},{식})

memset 함수는 사용자 공유 메모리 내에서 하나 또는 그 이상의 연속하는 바이트 위치 내용을 임의의 문자 값에 강제로 복사합니다.

최초의 {식}은 강제로 복사하는 레지스터 사용자 공유 메모리 내의 (시작)인덱스를 지정합니다. 이 인덱스는 사용자 버퍼의 시작에서 주소 오프셋입니다.

최초의 쉼표 뒤에 2 번째 {식}이 있는 경우에 이 식으로 메모리에 복사하는 문자의 ASCII 값을 지정합니다. 이 값이 0~255 범위 내의 경우에 이것은 단일 문자(1 바이트)를 나타내고 한번에 1 바이트가 복사됩니다. 값이 이 범위 밖인 경우에는 32 비트의 워드를 나타내며 4 바이트마다 복사됩니다.

최초의 쉼표 뒤의 작은따옴표가 있는 리터럴 문자가 있는 경우에는 메모리에 복사하는 문자(바이트)를 지정하여 한번에 1 바이트가 복사됩니다. 버퍼 프로그램 내에서 이런 방법에 따라 지정된 함수가 목록에 반환된 경우에 이것은 문자인 ASCII 코드 값으로 통지되므로 주의하십시오.

마지막 **{식}**은 강제로 값을 복사하는 문자열 버퍼 메모리의 바이트 수를 지정합니다. 축취치이것은 1 바이트마다 복사되는 경우에도 4 바이트마다 복사되는 경우에도 함께 바이트 수로 표현됩니다.

정상적인 실행 후에 함수의 반환값은 값을 복사한 최초의 레지스터의 사용자 공유 메모리 내의 인덱스입니다. 함수가 정상적으로 실행되지 않은 경우에 -1 의 값이 반환됩니다. 확인하지 않는 경우에도 일반적으로 더미 변수의 대입처럼 반드시 무언가를 뒤야 합니다. 이것은 연산의 유효성을 평가하는 경우에 유용합니다.

예

```
CharIndex = memset(256, 65, 1)
CharIndex = memset(256, 64+LtrNum, 8)
CharIndex = memset(256, 'A', 1)
```

POWER PMAC I/O 주소 오프셋

이 섹션은 Power PMAC 이 물리적인 I/O 와 통신할 때 사용하는 인터페이스 IC 의 I/O 주소 오프셋에 대한 참조 내용입니다. 대부분의 사용자는 해당 IC 내 레지스터의 주소 수치를 확인할 필요가 없으므로 참조용으로 사용하십시오.

이 섹션에서 제공하는 주소 오프셋은 I/O 기준 주소의 기준이 됩니다. 이 I/O 기준 주소는 Power PMAC 의 하드웨어 및 소프트웨어의 버전에 따라 다른 경우가 있습니다. I/O 기준 주소의 절대값은 데이터 구조체 요소 [Sys.piom]으로 확인할 수 있습니다.

이러한 인터페이스 IC 가 탑재된 대부분의 제품은 이전 버전 Turbo PMAC CPU 에서도 사용할 수 있습니다. 또한 Turbo PMAC CPU 에서 Power PMAC CPU 로 이행하는 경우에 대비하여 Turbo PMAC 의 주소 및 설정에 대해서도 필요에 따라 참조용으로 기재하였습니다.

칩 선택 라인의 개요

Power PMAC CPU 는 칩 선택 라인 및 주소 라인을 사용하여 다양한 인터페이스 IC 의 주소를 지정합니다. 사용하는 칩 선택 라인은 인터페이스 IC 의 클래스에 따라 달라집니다. 다음 표에 Power PMAC 시스템에서 사용할 수 있는 칩 선택 라인, 각 라인의 기능 및 칩 선택의 기준 주소 오프셋([Sys.piom]이 기점)을 나타냅니다.

칩 선택 라인	기능	Sys.BusCtrl[n] Index	Power PMAC I/O 기준 주소 오프셋	Turbo PMAC(절대) 기준 주소
CS00-*	온보드 I/O	0	\$000000	\$078800
CS02-*	온보드 I/O	1	\$100000	\$078900
CS04-*	온보드 I/O	2	\$200000	\$078A00
CS06-*	온보드 I/O	3	\$300000	\$078B00
CS0-*	온보드 Servo IC	4	\$400000	\$078000
CS1-*	온보드 Servo IC	5	\$500000	\$078100
CS2-	확장 Servo IC	6	\$600000	\$078200
CS3-	확장 Servo IC	7	\$700000	\$078300
CS4-	MACRO IC	8	\$800000	\$078400
CS5-	DSPGATE3 IC	9	\$900000	--
CS10-	확장 I/O	10	\$A00000	\$078C00
CS12-	확장 I/O	11	\$B00000	\$078D00
CS14-	확장 I/O	12	\$C00000	\$078E00
CS16-	확장 I/O	13	\$D00000	\$078F00
DPRCS-(MemCS0-)	공유 메모리	14	\$E00000	\$060000
VMECS-(MemCS1-)	VME 인터페이스 또는 공유 메모리	15	\$F00000	\$070000

* 현재는 Power PMAC 하드웨어에 탑재 안 됨

PMAC2 스타일 Servo 카드의 UMAC

Power PMAC UMAC 시스템은 PMAC2 스타일 DSPGATE1 Servo IC 를 사용하는 여러 종류의 액세서리 보드에 대응합니다. 여기에는 ACC-24E2, ACC-24E2A, ACC-24E2S 및 ACC-51E 등이 있습니다. 이 클래스의 보드는 주소 충돌 발생을 방지하기 위해 하나의 UMAC 랙 내에서 각각 임의의 DIP 스위치 설정이 필요합니다. 다음 표에 사용 가능한 스위치 설정과 그 결과의 주소 오프셋 및 [Gate1[i]] 인덱스값 'i'를 나타냅니다.

SW1-1 (CS2/3)	SW1-2 (주소 비트 8)	SW1-3 (주소 비트 15)	SW1-4 (주소 비트 16)	Power PMAC Gate1 인덱스 번호	Power PMAC I/O 베이스 주소 오프셋	Turbo PMAC Servo IC 번호	Turbo PMAC 베이스 주소
ON(0)	ON(0)	ON(0)	ON(0)	4	\$600000	2	\$078200
ON(0)	OFF(1)	ON(0)	ON(0)	5	\$600100	2*	\$078220
OFF(1)	ON(0)	ON(0)	ON(0)	6	\$700000	3	\$078300
OFF(1)	OFF(1)	ON(0)	ON(0)	7	\$700100	3*	\$078320
ON(0)	ON(0)	OFF(1)	ON(0)	8	\$608000	4	\$079200
ON(0)	OFF(1)	OFF(1)	ON(0)	9	\$608100	4*	\$079220
OFF(1)	ON(0)	OFF(1)	ON(0)	10	\$708000	5	\$079300
OFF(1)	OFF(1)	OFF(1)	ON(0)	11	\$708100	5*	\$079320
ON(0)	ON(0)	ON(0)	OFF(1)	12	\$610000	6	\$07A200
ON(0)	OFF(1)	ON(0)	OFF(1)	13	\$610100	6*	\$07A220
OFF(1)	ON(0)	ON(0)	OFF(1)	14	\$710000	7	\$07A300
OFF(1)	OFF(1)	ON(0)	OFF(1)	15	\$710100	7*	\$07A320
ON(0)	ON(0)	OFF(1)	OFF(1)	16	\$618000	8	\$07B200
ON(0)	OFF(1)	OFF(1)	OFF(1)	17	\$618100	8*	\$07B220
OFF(1)	ON(0)	OFF(1)	OFF(1)	18	\$718000	9	\$07B300
OFF(1)	OFF(1)	OFF(1)	OFF(1)	19	\$718100	9*	\$07B320

Turbo PMAC 주소(*TurboAdr*) 및 Power PMAC 주소 오프셋(*PowerAdr*)은 다음 식으로 조합할 수 있습니다.

$$PowerAdr = \$600000 + (TurboAdr \& \$100) * \$1000 + (TurboAdr \& \$7CF8) * 8 + (TurboAdr \& 7) * 4 + "X" * \$20$$

여기서 'X'는 Turbo PMAC X 주소의 경우에 1, Turbo PMAC Y 주소의 경우에 0 입니다.

PMAC2 스타일 MACRO 카드의 UMAC 주소

Power PMAC UMAC 시스템은 하나 이상의 PMAC2 스타일 ACC-5E MACRO 인터페이스 액세서리 보드에 대응합니다. ACC-5E 보드는 주소 충돌 발생을 방지하기 위해 하나의 UMAC 랙 내에서 각각 임의의 DIP 스위치 설정이 필요합니다. 또한 ACC-5E 카드에 2 번째 IC(자동으로 최초의 IC 다음으로 높은 주소 및 바로 다음 인덱스가 할당됨)가 탑재된 경우에는 다른 ACC-5E 의 최초의 IC 가 이 IC 의 주소와 충돌하지 않는지 확인해야 합니다. 다음 표에 사용 가능한 스위치 설정, 결과의 주소 오프셋 및 [Gate2[i]] 인덱스값 'i'를 나타냅니다. *

SW1-1 (주소 비트 15)	SW1-2 (주소 비트 16)	SW1-3 (주소 비트 11)	SW1-4 (주소 비트 12)	Power PMAC Gate2 인덱스 번호*	Power PMAC I/O 기준 주소 오프셋	Turbo PMAC MACRO IC 번호	Turbo PMAC 베이스 주소
ON(0)	ON(0)	ON(0)	ON(0)	0	\$800000	0	\$078400
OFF(1)	ON(0)	ON(0)	ON(0)	1	\$808000	1	\$079400
ON(0)	OFF(1)	ON(0)	ON(0)	2	\$810000	2	\$07A400
OFF(1)	OFF(1)	ON(0)	ON(0)	3	\$818000	3	\$07B400
ON(0)	ON(0)	OFF(1)	ON(0)	4	\$800800	4	\$078500
OFF(1)	ON(0)	OFF(1)	ON(0)	5	\$808800	5	\$079500
ON(0)	OFF(1)	OFF(1)	ON(0)	6	\$810800	6	\$07A500
OFF(1)	OFF(1)	OFF(1)	ON(0)	7	\$818800	7	\$07B500
ON(0)	ON(0)	ON(0)	OFF(1)	8	\$801000	8	\$078600
OFF(1)	ON(0)	ON(0)	OFF(1)	9	\$809000	9	\$079600
ON(0)	OFF(1)	ON(0)	OFF(1)	10	\$811000	10	\$07A600
OFF(1)	OFF(1)	ON(0)	OFF(1)	11	\$819000	11	\$07B600
ON(0)	ON(0)	OFF(1)	OFF(1)	12	\$801800	12	\$078700
OFF(1)	ON(0)	OFF(1)	OFF(1)	13	\$809800	13	\$079700
ON(0)	OFF(1)	OFF(1)	OFF(1)	14	\$811800	14	\$07A700
OFF(1)	OFF(1)	OFF(1)	OFF(1)	15	\$819800	15	\$07B700

Turbo PMAC 주소(*TurboAdr*) 및 Power PMAC 주소 오프셋(*PowerAdr*)은 다음 식으로 조합할 수 있습니다.

$$PowerAdr = \$800000 + (TurboAdr \& \$7BF8) * 8 + (TurboAdr \& 7) * 4 + "X" * \$20$$

여기서 'X'는 Turbo PMAC X 주소의 경우에 1, Turbo PMAC Y 주소의 경우에 0 입니다.

* 데이터 구조체 [Gate2[i]]의 인덱스 번호 'i'는 가장 하위의 주소 오프셋 설정에 배치된 MACRO IC 를 사용하여 시스템이 구성된 경우에 한해 유효합니다(이 구성을 적극 권장합니다). Power PMAC CPU 이 전원 셋업/리셋 시에 PMAC2 스타일 MACRO 카드를 자동 검출하고 주소 오프셋이 최하위의 IC 에 인덱스 번호 '0'을 할당하고 그 다음으로 높은 주소 오프셋의 IC 에 '1'을 할당합니다.

ACC-5E 의 최초의 MACRO IC 는 카드의 DIP 스위치 설정에 기반하여 위 표에 기재된 주소 오프셋에 배치됩니다. ACC-5E 에 2 개째의 MACRO IC 가 탑재되어 있는 경우에 2 번째 IC 는 다음으로 높은 주소 오프셋에 배치되고 다음으로 높은 인덱스 번호가 자동으로 할당됩니다.

PMAC2 스타일 I/O 카드의 UMAC 주소

Power PMAC UMAC 시스템은 PMAC2 스타일 IOGATE I/O IC 를 사용하는 여러 종류의 액세스리 보드에 대응합니다. 여기에는 ACC-11E*, ACC-14E, ACC-65E, ACC-66E, ACC-67E 및 ACC-68E 등이 있습니다. 이 클래스의 보드는 주소 충돌 발생을 방지하기 위해 하나의 UMAC 랙 내에서 각각 임의의 DIP 스위치 설정이 필요합니다. 다음 표에 사용 가능한 스위치 설정과 그 결과의 주소 오프셋 및 [Gatelo[i]] 인덱스값 'i'를 나타냅니다(ACC-11E 는 데이터 구조체 [Gatelo[i]]에서 액세스할 수 없습니다. 'M' 변수 포인터에서 레지스터에 액세스해야 합니다).

ACC-28E 고분해능 A/D 컨버터 보드도 이와 동일한 주소 범위를 사용합니다. 이러한 레지스터는 데이터 구조체 [Acc28E[i]]에서 액세스할 수 있습니다. 다른 액세스리 보드의 경우에 특히 시리얼 엔코더 인터페이스에서도 이와 동일한 주소 범위를 사용하지만 사전 정의된 데이터 구조체는 없습니다. 'M' 변수 포인터를 사용해서 이러한 레지스터에 액세스해야 합니다.

SW1-1 (CS10 / 12)	SW1-2 (CS10 / 14)	SW1-3 (주소 비트 15)	SW1-4 (주소 비트 16)	Power PMAC Gatelo 인덱스 번호	Power PMAC I/O 베이스 주소 오프셋	Turbo PMAC IO IC 번호	Turbo PMAC 베이스 주소
ON(0)	ON(0)	ON(0)	ON(0)	0	\$A00000	0	Y:\$078C00
OFF(1)	ON(0)	ON(0)	ON(0)	1	\$B00000	1	Y:\$078D00
ON(0)	OFF(1)	ON(0)	ON(0)	2	\$C00000	2	Y:\$078E00
OFF(1)	OFF(1)	ON(0)	ON(0)	3	\$D00000	3	Y:\$078F00
ON(0)	ON(0)	OFF(1)	ON(0)	4	\$A08000	4	Y:\$079C00
OFF(1)	ON(0)	OFF(1)	ON(0)	5	\$B08000	5	Y:\$079D00
ON(0)	OFF(1)	OFF(1)	ON(0)	6	\$C08000	6	Y:\$079E00
OFF(1)	OFF(1)	OFF(1)	ON(0)	7	\$D08000	7	Y:\$079F00
ON(0)	ON(0)	ON(0)	OFF(1)	8	\$A10000	8	Y:\$07AC00
OFF(1)	ON(0)	ON(0)	OFF(1)	9	\$B10000	9	Y:\$07AD00
ON(0)	OFF(1)	ON(0)	OFF(1)	10	\$C10000	10	Y:\$07AE00
OFF(1)	OFF(1)	ON(0)	OFF(1)	11	\$D10000	11	Y:\$07AF00
ON(0)	ON(0)	OFF(1)	OFF(1)	12	\$A18000	12	Y:\$07BC00
OFF(1)	ON(0)	OFF(1)	OFF(1)	13	\$B18000	13	Y:\$07BD00
ON(0)	OFF(1)	OFF(1)	OFF(1)	14	\$C18000	14	Y:\$07BE00
OFF(1)	OFF(1)	OFF(1)	OFF(1)	15	\$D18000	15	Y:\$07BF00

Turbo PMAC 주소(*TurboAdr*) 및 Power PMAC 주소 오프셋(*PowerAdr*)은 다음 식으로 조합할 수 있습니다.

$$PowerAdr = \$A00000 + (TurboAdr \& \$300) * \$1000 + (TurboAdr \& \$70F8) * 8 + (TurboAdr \& 7) * 4$$

* ACC-11E 는 그 외의 카드의 DIP 스위치 설정의 기능과 동일한 기준 주소 오프셋의 규칙이 있지만 Power PMAC CPU 에 의한 자동 검출에 필요한 회로가 없으므로 데이터 구조체 [Gatelo[i]]를 사용할 수 없습니다. 이 레지스터에는 레지스터의 주소 오프셋에 기반하여 'M' 변수를 사용해서 액세스해야 합니다.

PMAC2 스타일 공유 메모리 카드의 UMAC 주소

Power PMAC UMAC 시스템은 프로세서와 외부의 통신 채널 간의 공유 메모리 인터페이스를 제공하는 액세스리 보드에 대응합니다. 이 중 가장 중요한 것은 ACC-72E 필드 버스 인터페이스 보드입니다. 이 클래스의 보드는 주소 충돌 발생을 방지하기 위해 하나의 UMAC 랙 내에서 각각 임의의 DIP 스위치 설정이 필요합니다. 다음 표에 사용 가능한 스위치 설정과 그 결과의 주소 오프셋을 나타냅니다.

SW1-1	SW1-2 (MEMCS 0 / 1)	SW1-3 (주소 비트 15)	SW1-4 (주소 비트 16)	공유 메모리 보드 번호	Power PMAC I/O 기준 주소 오프셋	Turbo PMAC 베이스 주소
Don't care	ON(0)	ON(0)	ON(0)	0	\$E00000	\$06C000
Don't care	OFF(1)	ON(0)	ON(0)	1	\$F00000	\$074000
Don't care	ON(0)	OFF(1)	ON(0)	2	\$E08000	\$06D000
Don't care	OFF(1)	OFF(1)	ON(0)	3	\$F08000	\$075000
Don't care	ON(0)	ON(0)	OFF(1)	4	\$E10000	\$06E000
Don't care	OFF(1)	ON(0)	OFF(1)	5	\$F10000	\$076000
Don't care	ON(0)	OFF(1)	OFF(1)	6	\$E18000	\$06F000
Don't care	OFF(1)	OFF(1)	OFF(1)	7	\$F18000	\$077000

Turbo PMAC 주소(*TurboAdr*) 및 Power PMAC 주소 오프셋(*PowerAdr*)은 다음 식으로 조합할 수 있습니다.

$$PowerAdr = \$E00000 + (TurboAdr \& \$10000) * \$10 + (TurboAdr \& \$3FF8) * 8 + (TurboAdr \& 7) * 4 + "X" * \$20$$

여기서 'X'는 Turbo PMAC X 주소의 경우에 1, Turbo PMAC Y 주소의 경우에 0 입니다.

PMAC3 스타일 카드의 UMAC 주소

Power PMAC UMAC 시스템은 PMAC3 스타일 DSPGATE3 머신 인터페이스 IC 를 사용하는 여러 종류의 액세스리 보드에 대응합니다. 여기에는 ACC-24E3, ACC-5E3 및 ACC-59E3 등이 있습니다. 이 클래스의 보드는 주소 충돌 발생을 방지하기 위해 하나의 UMAC 랙 내에서 각각 임의의 DIP 스위치 설정이 필요합니다. 다음 표에 사용 가능한 스위치 설정과 그 결과의 주소 오프셋 및 [Gate3[i]] 인덱스값 'i'를 나타냅니다.

SW1-1 (주소 비트 11)	SW1-2 (주소 비트 12)	SW1-3 (주소 비트 13)	SW1-4 (주소 비트 14)	Power PMAC Gate3 인덱스 번호	Power PMAC I/O 기준 주소 오프셋
OFF(0)	OFF(0)	OFF(0)	OFF(0)	0	\$900000
ON(1)	OFF(0)	OFF(0)	OFF(0)	1	\$904000
OFF(0)	ON(1)	OFF(0)	OFF(0)	2	\$908000
ON(1)	ON(1)	OFF(0)	OFF(0)	3	\$90C000
OFF(0)	OFF(0)	ON(1)	OFF(0)	4	\$910000
ON(1)	OFF(0)	ON(1)	OFF(0)	5	\$914000
OFF(0)	ON(1)	ON(1)	OFF(0)	6	\$918000
ON(1)	ON(1)	ON(1)	OFF(0)	7	\$91C000
OFF(0)	OFF(0)	OFF(0)	ON(1)	8	\$920000
ON(1)	OFF(0)	OFF(0)	ON(1)	9	\$924000
OFF(0)	ON(1)	OFF(0)	ON(1)	10	\$928000
ON(1)	ON(1)	OFF(0)	ON(1)	11	\$92C000
OFF(0)	OFF(0)	ON(1)	ON(1)	12	\$930000
ON(1)	OFF(0)	ON(1)	ON(1)	13	\$934000
OFF(0)	ON(1)	ON(1)	ON(1)	14	\$938000
ON(1)	ON(1)	ON(1)	ON(1)	15	\$93C000

Turbo PMAC CPU 는 DSPGATE3 IC 가 탑재된 보드에 주소 지정할 수 없습니다.

Turbo 및 Power PMAC의 UMAC 주소 지정의 개요

다음 표에 Turbo PMAC CPU 및 Power PMAC CPU 양쪽 모두와 사용하는 경우에 사용 가능한 모든 종류의 UMAC 액세스리 카드의 주소 지정을 나타냅니다. 양쪽의 CPU 타입에 대해 카드 기능 부분 및 식별 IC의 기준 주소를 나타냅니다.

메인 CS 번호	Turbo IC 번호	Turbo 카드 베이스 주소	Turbo ID 베이스 주소	Power IC의 타입 및 번호	Power 카드의 베이스 주소 오프셋	Power ID의 기준 주소 오프셋	Power 카드의 ID 번호
	{예약 완료}			{예약 완료}			0
	{예약 완료}			{예약 완료}			1
CS2	Servo 2	\$078200	\$078F08	Gate1[4]	\$600000	\$D00040	2
CS3	Servo 3	\$078300	\$078F0C	Gate1[6]	\$700000	\$D00050	3
CS4	Macro 0	\$078400	\$078F10	Gate2[0]~	\$800000	\$D00080	4
CS4	Macro 4	\$078500	\$078F14	Gate2[4]~	\$800800	\$D00090	5
CS4	Macro 8	\$078600	\$078F18	Gate2[8]~	\$801000	\$D000C0	6
CS4	Macro 12	\$078700	\$078F1C	Gate2[12]~	\$801800	\$D000D0	7
MemCS0	Mem 0	\$06C000	\$078F20	Dpr[0]	\$E00000	\$D00100	8
MemCS1	Mem 1	\$074000	\$078F24	Dpr[1]	\$F00000	\$D00110	9
CS2	Servo 2*	\$078220	\$078F28	Gate1[5]	\$600100	\$D00140	10
CS3	Servo 3*	\$078320	\$078F2C	Gate1[7]	\$700100	\$D00150	11
CS10	I/O 0	\$078C00	\$078F30	GateIo[0]	\$A00000	\$D00180	12
CS12	I/O 1	\$078D00	\$078F34	GateIo[1]	\$B00000	\$D00190	13
CS14	I/O 2	\$078E00	\$078F38	GateIo[2]	\$C00000	\$D001C0	14
CS16	I/O 3	\$078F00	\$078F3C	GateIo[3]	\$D00000	\$D001D0	15
	{예약 완료}			{예약 완료}			16
	{예약 완료}			{예약 완료}			17
CS2	Servo 4	\$079200	\$079F08	Gate1[8]	\$608000	\$D08040	18
CS3	Servo 5	\$079300	\$079F0C	Gate1[10]	\$708000	\$D08050	19
CS4	Macro 1	\$079400	\$079F10	Gate2[1]~	\$808000	\$D08080	20
CS4	Macro 5	\$079500	\$079F14	Gate2[5]~	\$808800	\$D08090	21
CS4	Macro 9	\$079600	\$079F18	Gate2[9]~	\$809000	\$D080C0	22
CS4	Macro 13	\$079700	\$079F1C	Gate2[13]~	\$809800	\$D080D0	23
MemCS0	Mem 2	\$06D000	\$079F20	Dpr[2]	\$E08000	\$D08100	24
MemCS1	Mem 3	\$075000	\$079F24	Dpr[3]	\$F08000	\$D08110	25
CS2	Servo 4*	\$079220	\$079F28	Gate1[9]	\$608100	\$D08140	26
CS3	Servo 5*	\$079320	\$079F2C	Gate1[11]	\$708100	\$D08150	27
CS10	I/O 4	\$079C00	\$079F30	GateIo[4]	\$A08000	\$D08180	28
CS12	I/O 5	\$079D00	\$079F34	GateIo[5]	\$B08000	\$D08190	29
CS14	I/O 6	\$079E00	\$079F38	GateIo[6]	\$C08000	\$D081C0	30
CS16	I/O 7	\$079F00	\$079F3C	GateIo[7]	\$D08000	\$D081D0	31
	{예약 완료}			{예약 완료}			32
	{예약 완료}			{예약 완료}			33
CS2	Servo 6	\$07A200	\$07AF08	Gate1[12]	\$610000	\$D10040	34
CS3	Servo 7	\$07A300	\$07AF0C	Gate1[14]	\$710000	\$D10050	35
CS4	Macro 2	\$07A400	\$07AF10	Gate2[2]~	\$810000	\$D10080	36
CS4	Macro 6	\$07A500	\$07AF14	Gate2[6]~	\$810800	\$D10090	37
CS4	Macro 10	\$07A600	\$07AF18	Gate2[10]~	\$811000	\$D100C0	38
CS4	Macro 14	\$07A700	\$07AF1C	Gate2[14]~	\$811800	\$D100D0	39
MemCS0	Mem 4	\$06E000	\$07AF20	Dpr[4]	\$E10000	\$D10100	40
MemCS1	Mem 5	\$076000	\$07AF24	Dpr[5]	\$F10000	\$D10110	41
CS2	Servo 6*	\$07A220	\$07AF28	Gate1[13]	\$610100	\$D10140	42
CS3	Servo 7*	\$07A320	\$07AF2C	Gate1[15]	\$710100	\$D10150	43

CS10	I/O 8	\$07AC00	\$07AF30	GateIo[8]	\$A10000	\$D10180	44
CS12	I/O 9	\$07AD00	\$07AF34	GateIo[9]	\$B10000	\$D10190	45
CS14	I/O 10	\$07AE00	\$07AF38	GateIo[10]	\$C10000	\$D101C0	46
CS16	I/O 11	\$07AF00	\$07AF3C	GateIo[11]	\$D10000	\$D101D0	47
	{예략 완료}			{예략 완료}			48
	{예략 완료}			{예략 완료}			49
CS2	Servo 8	\$07B200	\$07BF08	Gate1[16]	\$618000	\$D18040	50
CS3	Servo 9	\$07B300	\$07BF0C	Gate1[18]	\$718000	\$D18050	51
CS4	Macro 3	\$07B400	\$07BF10	Gate2[3]~	\$818000	\$D18080	52
CS4	Macro 7	\$07B500	\$07BF14	Gate2[7]~	\$818800	\$D18090	53
CS4	Macro 11	\$07B600	\$07BF18	Gate2[11]~	\$819000	\$D180C0	54
CS4	Macro 15	\$07B700	\$07BF1C	Gate2[15]~	\$819800	\$D180D0	55
MemCS0	Mem 6	\$06F000	\$07BF20	Dpr[6]	\$E18000	\$D18100	56
MemCS1	Mem 7	\$077000	\$07BF24	Dpr[7]	\$F18000	\$D18110	57
CS2	Servo 8*	\$07B220	\$07BF28	Gate1[17]	\$618100	\$D18140	58
CS3	Servo 9*	\$07B320	\$07BF2C	Gate1[19]	\$718100	\$D18150	59
CS10	I/O 12	\$07BC00	\$07BF30	GateIo[12]	\$A18000	\$D18180	60
CS12	I/O 13	\$07BD00	\$07BF34	GateIo[13]	\$B18000	\$D18190	61
CS14	I/O 14	\$07BE00	\$07BF38	GateIo[14]	\$C18000	\$D181C0	62
CS16	I/O 15	\$07BF00	\$07BF3C	GateIo[15]	\$D18000	\$D181D0	63

~: 시스템 내의 모든 ACC-5E 카드에 경합이 발생하지 않는 최하위 주소가 설정되어 있는 경우에 이러한 [Gate2[*i*]] 인덱스 번호는 이 주소만 유효합니다.

Power PMAC ASIC 레지스터 요소 주소

이 섹션에는 레지스터의 데이터 구조체 요소 및 Power PMAC 시스템에 존재하는 각 타입 ASIC의 레지스터 일부 및 그 주소가 기재됩니다. 요소의 일부는 저장 설정 요소(굵은 글씨로 표시)입니다. 이에 대해서는 설명서의 해당하는 장에서 자세하게 설명합니다. 요소 및 레지스터 대부분의 사용 방법에서는 레지스터의 수치 주소가 필요하지 않기 때문에 이러한 주소는 주로 참조용으로 기재됩니다.

I/O 기준 주소에서 레지스터 요소의 주소 오프셋을 계산하는 방법

1. IC의 종류 및 인덱스 번호에 기반하여 I/O 기준 주소에서 대상 IC의 기준 주소까지의 오프셋을 추가합니다. 이 오프셋 값은 각 타입 IC의 최상부 섹션에 상자로 둘러싸여 기재됩니다.
2. 채널 고유의 레지스터가 있는 경우에는 IC의 기준 주소에서 채널의 기준 주소까지의 오프셋을 추가합니다.
3. IC의 기준 주소에서 대상 레지스터까지의 오프셋을 추가하거나, 채널의 기준 주소에서 채널 고유의 레지스터까지의 오프셋을 추가합니다.

이 값은 'io' 형식의 'M' 변수를 선언할 때 사용할 수 있습니다.

Power PMAC의 메모리 맵에서 레지스터 요소의 절대 수치 주소를 찾는 방법

1. [Sys.piom]의 값을 쿼리하여 I/O의 시작 주소를 확인합니다.
2. 이 값을 위에서 계산한 주소 오프셋에 추가합니다.

이 값은 요소의 주소가 쿼리되었을 때 통지되는 수치의 값과 일치합니다.

주의:

1. 비트 번호는 'Intel'(리틀 엔디언) 스타일로 표시되고 비트 n 의 값을 워드로 나타내면 2^n 입니다. 하드웨어가 사용 중인 비트 수가 32비트 미만이라도 모든 비트 번호는 풀 워드 32비트입니다.
2. 굵은 글씨로 표시된 이름은 저장 설정 요소의 이름입니다. 이탤릭체로 표시된 이름은 직접 사용하는 실제 요소가 아니라 개별 기능을 갖는 풀 요소 내의 컴포넌트를 나타냅니다.

버퍼 프로그램 Status와 온라인 명령을 사용하는 Power PMAC의 스크립트 언어 환경에서는 32비트 미만의 요소는 요소의 실제 비트 폭을 사용하여 액세스됩니다. 예를 들어 [Gate1[j].Chan[j].Status]는 32비트 워드의 비트 08~31을 점유합니다. 스크립트 환경에서는 24비트값으로 처리됩니다. [Gate1[j].Chan[j].HomeFlag]는 이 동일한 워드의 24비트를 점유합니다. 이는 스크립트 환경에서는 단일 비트값으로 직접 액세스할 수 있습니다.

C언어에서는 레지스터가 32비트 워드 전부를 점유하지 않더라도 IC 레지스터는 풀 워드 32비트로만 액세스 가능합니다. 즉, [Gate1[j].Chan[j].Status]는 32비트값으로 처리되고 하위 8비트는 의미를 갖지 않습니다. C언어에서는 단일 비트값 [Gate1[j].Chan[j].HomeFlag]에는 직접 액세스할 수 없습니다. 다음과 같은 식에 따라 도출해야 합니다.

$(\text{Gate1}[j].\text{Chan}[j].\text{Status} \& \$01000000) \gg 24$

DSPGATE1(PMAC2 스타일 Servo ASIC) 레지스터 요소

DSPGATE1 Servo ASIC 는 UMAC 보드의 ACC-24E2, ACC-24E2A, ACC-24E2S 및 ACC-51E 에 탑재됩니다. IC 의 데이터 구조체는 일반명 [Gate1[*i*]] 또는 각각 보드명 [Acc24E2[*i*]], [Acc24E2A[*i*]], [Acc24E2S[*i*]] 또는 [Acc51E[*i*]]로 참조됩니다.

Gate1[#]	4	5	6	7	8	9	10	11
IC 베이스 오프셋	\$600000	\$600100	\$700000	\$700100	\$608000	\$608100	\$708000	\$708100
Gate1[#]	12	13	14	15	16	17	18	19
IC 베이스 오프셋	\$610000	\$610100	\$710000	\$710100	\$618000	\$618100	\$718000	\$718100

데이터 구조체	폴 요소	부분 요소	ASIC 베이스에서의 오프셋	비트
Gate1[<i>i</i>].	ClockCtrl	HardwareClockCtrl PhaseServoDir PhaseClockDiv ServoClockDiv	\$030	08-19 20-21 24-27 28-31
	DacStrobe		\$070	08-31
	AdcStrobe		\$0B0	08-31
	PwmCtrl		\$0F0	08-31
		PwmDeadTime		08-15
		PwmPeriod		16-31
	Chan[0].		\$000	
	Chan[1].		\$040	
	Chan[2].		\$080	
	Chan[3].		\$0C0	

Chan[#]	0	1	2	3
IC 베이스에서의 오프셋	\$000	\$040	\$080	\$0C0

데이터 구조체	폴 요소	부분 요소	채널 베이스에서의 오프셋	비트
Gate1[<i>i</i>].Chan[<i>j</i>].	TimeBetweenCts		\$000	08-31
		SCLK cycles between Change of direction		08-30 31
		Hardware 1/T values*		08-31
		CompA fraction*		08-19
		ServoCapt fraction*		20-31
	TimeSinceCts		\$004	08-31
		SCLK cycles since last {reserved}		08-30 31
		Hardware 1/T values*		08-31
		CompB fraction*		08-19
		HomeCapt fraction*		20-31
	Pwm[0] / Dac[0]		\$008	08-31
	Pwm[1] / Dac[1]		\$00C	08-31

Pwm[2] / Pfm	\$010	08-31
Adc[0]	\$014	08-31
Adc[1]	\$018	08-31
CompB	\$01C	08-31
Status	\$020	08-31
HallState		08-10
CountError		16
Equ		17
PosCapt		18-19
ABC		20-22
Fault (<i>Amp fault</i>)		23
HomeFlag		24
PlusLimit		25
MinusLimit		26
UserFlag		27
UVW		28-30
T		31
PhaseCapt	\$024	08-31
ServoCapt	\$028	08-31
HomeCapt	\$02C	08-31
Ctrl	\$034	08-31
EncCtrl		08-11
CaptCtrl		12-15
CaptFlagSel		16-17
PosClear		18
EquWrite		19-20
Equ1Ena		21
AmpEna		22
GatedIndexSel		23
IndexGateState		24-25
OneOverTEna		26
PfmDirPol		27
OutputPol		28-29
OutputMode		30-31
CompAdd	\$038	08-31
CompA	\$03C	08-31

(* [**OneOverTEna**]의 값을 '1'로 설정한 경우)

DSPGATE1 IC 가 탑재된 UMAC 카드에는 각각 카드의 정보가 포함된 ID 칩이 있습니다. 다음 표에 ID 칩의 기준 주소 오프셋 및 칩 내의 레지스터에 관한 정보를 기재합니다.

Gate1[#]	4	5	6	7	8	9	10	11
Cid[#]	2	10	3	11	18	26	19	27
ID 칩의 베이스 오프셋	\$D00040	\$D00140	\$D00050	\$D00150	\$D08040	\$D08140	\$D08050	\$D08150
Gate1[#]	12	13	14	15	16	17	18	19
Cid[#]	34	42	35	43	50	58	51	59
ID 칩의 베이스 오프셋	\$D10040	\$D10140	\$D10050	\$D10150	\$D18040	\$D18140	\$D18050	\$D18150

풀 레지스터	부분 컴포넌트	ID 칩 베이스에서의 오프셋	비트
Reg0		\$000	08-15
	Vendor ID bits 0-3*		08-11
	Revision number**		08-11
	Clock buffer direction (1=out)		12
	1st chan encoder loss (0 = loss)		13
Reg1		\$004	08-15
	Vendor ID bits 4-7*		08-11
	Card number bits 0-3**		08-11
	Bank select output		12
	2nd chan encoder loss (0 = loss)		13
Reg2		\$008	08-15
	Card options bits 0-4*		08-12
	Card number bits 4-8**		08-12
	3rd chan encoder loss (0 = loss)		13
Reg3		\$00C	08-15
	Card options bits 5-9*		08-12
	Card number bits 9-13**		08-12
	4th chan encoder loss (0 = loss)		13

(*[Bank select output]의 값을 '0'으로 설정한 경우)

(**[Bank select output]의 값을 '1'로 설정한 경우)

DSPGATE2(PMAC2 스타일 MACRO/I/O ASIC) 레지스터 요소

UMAC 보드 ACC-5E 에는 DSPGATE2 MACRO IC 가 탑재됩니다. IC 의 데이터 구조체는 일반명 [Gate2[i]] 또는 보드명 [Acc5E[i]]로 참조됩니다. 카드가 DIP 스위치에 따라 최하위의 주소 오프셋으로 설정되는 경우에 구조체 인덱스는 다음 목록에 있는 기준 주소 오프셋하고만 일치합니다.

Gate2[#]	0	1	2	3	4	5	6	7
IC 베이스 오프셋	\$800000	\$800800	\$801000	\$801800	\$808000	\$808800	\$809000	\$809800
Gate2[#]	8	9	10	11	12	13	14	15
IC 베이스 오프셋	\$810000	\$810800	\$811000	\$811800	\$818000	\$818800	\$819000	\$819800

데이터 구조체	풀 요소	부분 요소	ASIC 베이스에서의 오프셋	비트
Gate2[i].	LowIoData		\$000	08-31
	HighIoData		\$004	08-23
	MuxData		\$008	08-23
	DispData		\$00C	08-19
	LowIoMode		\$010	08-31
	HighIoMode		\$014	08-15
	MuxMode		\$018	08-23
	DispMode		\$01C	08-19
	LowIoDir		\$020	08-31
	HighIoDir		\$024	08-15
	MuxDir		\$028	08-23
	DispDir		\$02C	08-19

LowIoPol	\$030	08-31
HighIoPol	\$034	08-15
MuxPol	\$038	08-23
DispPol	\$03C	08-19
LowIoGray	\$050	08-31
IoGrayCtrl	\$054	08-12
MacroEnable	\$058	08-31
MacroMode	\$05C	08-23
DacStrobe	\$070	08-31
ClockCtrl	\$07C	08-31
	HardwareClockCtrl	08-19
	PhaseServoDir	20-21
	PhaseClockDiv	24-27
	ServoClockDiv	28-31
AdcStrobe	\$0B0	08-31
PwmCtrl	\$0F0	08-31
	PwmDeadTime	08-15
	PwmPeriod	16-31
Chan[0].	\$080	
Chan[1].	\$0C0	
Macro[m][n]	$m*\$10+n*4+\$100$	
(m=0 to 15, n=0 to 3)		

Chan[#]	0	1
IC	\$080	\$0C0
베이스에서의 오프셋		

데이터 구조체	폴 요소	부분 요소	채널 베이스에서의 오프셋	비트
Gate2[j].Chan[j].	TimeBetweenCts		\$000	08-31
	TimeSinceCts		\$004	08-31
	Pwm[0] / Dac[0]		\$008	08-31
	Pwm[1] / Dac[1]		\$00C	08-31
	Pwm[2] / Pfm		\$010	08-31
	Adc[0]		\$014	08-31
	Adc[1]		\$018	08-31
	CompB		\$01C	08-31
	Status		\$020	08-31
		HallState		08-10
		CountError		16
		Equ		17
		PosCapt		18-19
		ABC		20-22
		Fault (<i>Amp fault</i>)		23
		HomeFlag		24
		PlusLimit		25
		MinusLimit		26
		UserFlag		27
		UVW		28-30
		T		31
	PhaseCapt		\$024	08-31
	ServoCapt		\$028	08-31
	HomeCapt		\$02C	08-31

Ctrl	\$034	08-31
EncCtrl		08-11
CaptCtrl		12-15
CaptFlagSel		16-17
PosClear		18
EquWrite		19-20
Equ1Ena		21
AmpEna		22
GatedIndexSel		23
IndexGateState		24-25
OneOverTEna		26
PfmDirPol		27
OutputPol		28-29
OutputMode		30-31
CompAdd	\$038	08-31
CompA	\$03C	08-31

DSPGATE2 IC가 탑재된 UMAC 카드에는 각각 카드의 정보가 포함된 ID 칩이 있습니다. 다음 표에 ID 칩의 기준 주소 오프셋 및 칩 내의 레지스터에 관한 정보를 기재합니다.

Gate2[#]	0	1	2	3	4	5	6	7
Cid[#]	4	20	36	52	5	21	37	53
ID 칩의 베이스 오프셋	\$D00080	\$D08080	\$D10080	\$D18080	\$D00090	\$D08090	\$D10090	\$D18090
Gate2[#]	8	9	10	11	12	13	14	15
Cid[#]	6	22	38	54	7	23	39	55
ID 칩의 베이스 오프셋	\$D000C0	\$D080C0	\$D100C0	\$D180C0	\$D000D0	\$D080D0	\$D100D0	\$D180D0

풀 레지스터	부분 컴포넌트	ID 칩 베이스에서의 오프셋	비트
Reg0	Vendor ID bits 0-3*	\$000	08-15
	Revision number**		08-11
	Clock buffer direction (1=out)		12
Reg1	Vendor ID bits 4-7*	\$004	08-15
	Card number bits 0-3**		08-11
	Bank select output		12
Reg2	Card options bits 0-4*	\$008	08-15
	Card number bits 4-8**		08-12
Reg3	Card options bits 5-9*	\$00C	08-15
	Card number bits 9-13**		08-12

(*[Bank select output]의 값을 '0'으로 설정한 경우)

(**[Bank select output]의 값을 '1'로 설정한 경우)

DSPGATE3(PMAC3 스타일 범용 ASIC) 레지스터 요소

UMAC 보드의 ACC-24E3, ACC-5E3 및 ACC-59E3 에는 DSPGATE3 머신 인터페이스 IC 가 탑재됩니다. IC 의 데이터 구조체는 범용명 [Gate3[*i*]] 또는 각각 보드명 [Acc24E3[*i*]], [Acc5E3[*i*]], [Acc59E3[*i*]] 또는 [PowerBrick[*i*]]로 참조됩니다.

Gate3[#]	0	1	2	3	4	5	6	7
IC 베이스 오프셋	\$900000	\$904000	\$908000	\$90C000	\$910000	\$914000	\$918000	\$91C000
Gate3[#]	8	9	10	11	12	13	14	15
IC 베이스 오프셋	\$920000	\$924000	\$928000	\$92C000	\$930000	\$934000	\$938000	\$93C000

데이터 구조체	필 요소	부분 요소	ASIC 베이스에서의 오프셋	비트
Gate3[<i>i</i>].	Chan[0] Chan[1] Chan[2] Chan[3]		\$000 \$080 \$100 \$180	
	PhaseServoClockCtrl		\$200	00-31
	EncLatchDelay			00-07
	PhaseServoDir			10-11
	PhaseClockDiv			12-13
	PhaseClockMult			14-15
	PhaseFreq			16-31
	HardwareClockCtrl		\$204	00-31
	ServoClockDiv			00-03
	ClockPol			04-07
	FiltClockDiv			08-11
	EncClockDiv			12-15
	PfmClockDiv			16-19
	DacClockDiv			20-23
	AdcEncClockDiv			24-27
	AdcAmpClockDiv			28-31
	DacStrobe		\$208	00-31
	AdcAmpCtrl		\$20C	00-31
	AdcAmpHeaderBits			00-02
	AdcAmpUtoSConv			03
	AdcStrobeAmpDelay			04-07
	AdcStrobeAmpWord			08-31
	AdcEncCtrl		\$210	00-31
	AdcEncHeaderBits			00-02
	AdcEncUtoSConv			03
	AdcStrobeEncDelay			04-07
	AdcStrobeEncWord			08-31
	ResolverCtrl		\$214	00-31
	{reserved}			00-19
	ResolverExciteFreq			20-21
	ResolverExciteGain			22-23
	ResolverExcitePhase			24-31
	SerialEncCtrl		\$218	00-31
	SerialProtocol			00-03
	{reserved}			04-07
	SerialTrigDelay			08-15

	SerialTrigEdgeSel		16
	SerialTrigClockSel		17
	{reserved}		18-19
	SerialClockNDiv		20-23
	SerialClockMDiv		24-31
WpKey		\$21C	00-31
ChipID		\$220	00-31
IntCtrl		\$224	00-31
	InterruptStatus		00-07
	Chan[0] Capture		00
	Chan[1] Capture		01
	Chan[2] Capture		02
	Chan[3] Capture		03
	Chan[0] Compare		04
	Chan[1] Compare		05
	Chan[2] Compare		06
	Chan[3] Compare		07
	InterruptSource(Signal/Status)		08-15
	Chan[0] Capture		08
	Chan[1] Capture		09
	Chan[2] Capture		10
	Chan[3] Capture		11
	Chan[0] Compare		12
	Chan[1] Compare		13
	Chan[2] Compare		14
	Chan[3] Compare		15
	InterruptEnable		16-23
	Chan[0] Capture		16
	Chan[1] Capture		17
	Chan[2] Capture		18
	Chan[3] Capture		19
	Chan[0] Compare		20
	Chan[1] Compare		21
	Chan[2] Compare		22
	Chan[3] Compare		23
	{reserved}		24-31
MacroError		\$228	00-31
EEpromCtrl		\$22C	00-31
	EEpromReadEna		00
	EEpromWriteEna		01
	EEpromWP		02
	EEpromError		03
	EEpromSel		04-06
	(reserved)		02-31
EEpromData[0]		\$230	00-31
EEpromData[1]		\$234	00-31
EEpromData[2]		\$238	00-31
EEpromData[3]		\$23C	00-31
GpioCtrl / GpioMode[0]		\$240	00-31
GpioMode[1]		\$244	00-31
GpioMode[2]		\$248	00-31
GpioMode[3]		\$24C	00-31
GpioData[0]		\$250	00-31
GpioData[1]		\$254	00-31
GpioData[2]		\$258	00-31
GpioData[3]		\$25C	00-31
GpioDir[0]		\$260	00-31
GpioDir[1]		\$264	00-31
GpioDir[2]		\$268	00-31
GpioDir[3]		\$26C	00-31

GpioPol[0]	\$270	00-31
GpioPol[1]	\$274	00-31
GpioPol[2]	\$278	00-31
GpioPol[3]	\$27C	00-31
ReadAlias[n] (n = 0 to 14)	$n*4+\$280$	00-31
WriteAlias[n] (n = 0 to 14)	$n*4+\$2C0$	00-31
MemAlias[n] (n = 0 to 59)	$n*4+\$300$	00-31
MacroEnableA	\$3F0	00-23
MacroModeA	\$3F4	00-15
MacroEnableB	\$3F8	00-23
MacroModeB	\$3FC	00-15
MacroInA[m][n] (m=0 to 15, n=0 to 3)	$m*\$10+n*4+\$400$	00-31
MacroOutA[m][n] (m=0 to 15, n=0 to 3)	$m*\$10+n*4+\$500$	00-31
MacroInB[m][n] (m=0 to 15, n=0 to 3)	$m*\$10+n*4+\$600$	00-31
MacroOutB[m][n] (m=0 to 15, n=0 to 3)	$m*\$10+n*4+\$700$	00-31

Chan[#]	0	1	2	3
IC 베이스에서의 오프셋	\$000	\$080	\$100	\$180

데이터 구조체	폴 요소	부분 요소	채널 베이스에서의 오프셋	비트
Gate3[.].Chan[.].	Status		\$000	00-31
		ABPins		00-03
		ABC		04-06
		Fault (<i>Amp fault</i>)		07
		HomeFlag		08
		PlusLimit		09
		MinusLimit		10
		UserFlag		11
		UVW		12-14
		T		15
		HallState		16-18
		DemuxInvalid		19
		PosCapt		20-21
		TrigState		22
		Equ		24
		EquOut		25
		LossStatus		28
		LossCapt		29
		CountError		30
		SosError		31
	PhaseCapt		\$004	00-31
	ServoCapt		\$008	00-31
	AtanSumOfSqr		\$00C	00-31
		SumOfSquares		00-15
		Atan		16-31
	TimerA		\$010	00-31
	TimerB		\$014	00-31
	SerialEncDataA		\$018	00-31

SerialEncDataB	\$01C	00-31
AdcAmp[0]	\$020	00-31
AdcAmp[1]	\$024	00-31
AdcAmp[2]	\$028	00-31
AdcAmp[3]	\$02C	00-31
AdcEnc[0]	\$030	00-31
AdcEnc[1]	\$034	00-31
AdcEnc[2]	\$038	00-31
AdcEnc[3]	\$03C	00-31
Pwm[0] / Dac[0]	\$040	00-31
Pwm[1] / Dac[1]	\$044	00-31
Pwm[2] / Dac[2]	\$048	00-31
Pwm[3] / Pfm	\$04C	00-31
CompA	\$050	00-31
CompB	\$054	00-31
CompAdd	\$058	00-31
SerialEncCmd	\$05C	00-31
	<i>SerialPosNumBits</i>	00-05
	<i>SerialStatusNumBits</i>	06-09
	<i>SerialDataReady</i>	10
	<i>SerialGrayBinConv</i>	11
	<i>SerialTrigEna</i>	12
	<i>SerialTrigMode</i>	13
	<i>SerialParityType</i>	14-15
	<i>SerialCmdWord</i>	16-31
AdcOffset[0]	\$060	00-31
AdcOffset[1]	\$064	00-31
InCtrl	\$068	00-31
	EncCtrl	00-03
	TimerMode	04-05
	CaptCtrl	06-09
	CaptFlagSel	10-11
	CaptFlagChan	12-13
	GatedIndexSel	14
	IndexGateState	15
	IndexDemuxEna	16
	FlagFilt2Ena	17
	AtanEna	18
	PosClear	19
	SerialEncEna	20
	PackInData	21-22
	{reserved}	23-31
OutCtrl	\$06C	00-31
	EquOutMask	00-03
	EquOutPol	04
	Equ1Ena	05
	EquWrite	06-07
	AmpEna	08
	OutFlagB	09
	OutFlagC	10
	OutFlagD	11
	OutputMode	12-15
	OutputPol	16-17
	PfmDirPol	18
	PfmFormat	19
	PwmFreqMult	20-22
	PackOutData	23
	PwmDeadTime	24-31
PfmWidth	\$070	00-11
HomeCapt	\$074	00-31

다른 Power PMAC 인터페이스와는 달리 PMAC3 스타일 IC 가 조립된 경우에는 프로세서가 직접 액세스할 수 있는 개별 ID 칩이 없습니다. 그 대신 IC 의 요소 [EEpromData[k]] 내의 DSPGATE3 IC 에서 읽기 가능한 EEPROM IC 에 ID 정보가 저장됩니다.

범용 I/O 레지스터 요소

Power PMAC 은 아날로그 및 디지털 양쪽 모두의 범용 입력/출력 액세스를 다수 지원합니다. 이는 \$A00000~\$DFFFFFF 의 오프셋으로 I/O 주소 영역에 매핑됩니다. 이 페이지의 다양한 액세스를 조합할 수 있지만 서로 다른 타입의 카드를 조합할 때는 해당 주소가 충돌하지 않는지 반드시 확인하십시오.

IOGATE(PMAC2 스타일 I/O ASIC) 레지스터 요소

UMAC 보드의 ACC-14E, ACC-65E, ACC-66E, ACC-67E 및 ACC-68E 에는 IOGATE I/O ASIC 가 탑재됩니다. IC 의 데이터 구조체는 일반명 [GateIo[j]] 또는 각각 보드명 [Acc14E[j]], [Acc65E[j]], [Acc66E[j]], [Acc67E[j]], [Acc68E[j]]로 참조됩니다. ACC-11E I/O 보드도 IOGATE IC 를 사용하지만 CPU 가 사전 정의된 데이터 구조체를 사용할 때 필요한 ID 회로가 없습니다. ACC-11E 레지스터는 포인터(ptr) 'M' 변수를 사용해서 액세스해야 합니다.

GateIo[j]	0	1	2	3	4	5	6	7
IC 베이스 오프셋	\$A00000	\$B00000	\$C00000	\$D00000	\$A08000	\$B08000	\$C08000	\$D08000
GateIo[j]	8	9	10	11	12	13	14	15
IC 베이스 오프셋	\$A10000	\$B10000	\$C10000	\$D10000	\$A18000	\$B18000	\$C18000	\$D18000

데이터 구조체	폴 주소	부분 주소	ASIC 베이스에서의 오프셋	비트
GateIo[j].	DataReg[0]		\$000	08-15
		DataReg[0].0.1		08
		DataReg[0].1.1		09
		DataReg[0].2.1		10
		DataReg[0].3.1		11
		DataReg[0].4.1		12
		DataReg[0].5.1		13
		DataReg[0].6.1		14
		DataReg[0].7.1		15
	DataReg[1]		\$004	08-15
		{as for DataReg[0]}		
	DataReg[2]		\$008	08-15
		{as for DataReg[0]}		
	DataReg[3]		\$00C	08-15
		{as for DataReg[0]}		
	DataReg[4]		\$010	08-15
		{as for DataReg[0]}		
	DataReg[5]		\$014	08-15
		{as for DataReg[0]}		
	IntrReg		\$018	08-15
	CtrlReg		\$01C	08-15

IOGATE IC가 탑재된 현행 UMAC 카드(기존 버전 ACC-11E 이외)에는 카드의 정보가 포함된 ID 칩이 탑재됩니다. 다음 표에 ID 칩의 기준 주소 오프셋 및 칩 내의 레지스터에 관한 정보를 기재합니다.

Gatelo[#]	0	1	2	3	4	5	6	7
Cid[#]	12	13	14	15	28	29	30	31
ID 칩의 베이스 오프셋	\$D00180	\$D00190	\$D001C0	\$D001D0	\$D08180	\$D08190	\$D081C0	\$D081D0
Gatelo[#]	8	9	10	11	12	13	14	15
Cid[#]	44	45	46	47	60	61	62	63
ID 칩의 베이스 오프셋	\$D10180	\$D10190	\$D101C0	\$D101D0	\$D18180	\$D18190	\$D181C0	\$D181D0

플 레지스터	부분 컴포넌트	ID 칩 베이스에서의 오프셋	비트
<i>Reg0</i>	<i>Vendor ID bits 0-3*</i> <i>Revision number**</i>	\$000	08-15 08-11 08-11
<i>Reg1</i>	<i>Vendor ID bits 4-7*</i> <i>Card number bits 0-3**</i> <i>Bank select output</i>	\$004	08-15 08-11 08-11 12
<i>Reg2</i>	<i>Card options bits 0-4*</i> <i>Card number bits 4-8**</i>	\$008	08-15 08-12 08-12
<i>Reg3</i>	<i>Card options bits 5-9*</i> <i>Card number bits 9-13**</i>	\$00C	08-15 08-12 08-12

(*[Bank select output]의 값을 '0'으로 설정한 경우)

(**[Bank select output]의 값을 '1'로 설정한 경우)

ACC-28E(A/D 컨버터) 레지스터 요소

ACC-28E 고분해능 A/D 컨버터 카드에는 16비트의 읽기 전용 레지스터가 4개 탑재되며 선택한 요소에서 부호가 없는('U') 값 또는 부호가 있는('S') 값으로 액세스할 수 있습니다.

Acc28E[#]	0	1	2	3	4	5	6	7
IC 베이스 오프셋	\$A00000	\$B00000	\$C00000	\$D00000	\$A08000	\$B08000	\$C08000	\$D08000
Acc28E[#]	8	9	10	11	12	13	14	15
IC 베이스 오프셋	\$A10000	\$B10000	\$C10000	\$D10000	\$A18000	\$B18000	\$C18000	\$D18000

데이터 구조체	폴 요소	ASIC 베이스에서의 오프셋 비트	
Acc28E[i].	AdcUdata[0]	\$000	16-31
	AdcSdata[0]	\$000	16-31
	AdcUdata[1]	\$004	16-31
	AdcSdata[1]	\$004	16-31
	AdcUdata[2]	\$008	16-31
	AdcSdata[2]	\$008	16-31
	AdcUdata[3]	\$00C	16-31
	AdcSdata[3]	\$00C	16-31

ACC-28E에는 각각 카드에 관한 정보가 포함된 ID 칩이 탑재됩니다. 다음 표에 ID 칩의 기준 주소 오프셋 및 칩 내의 레지스터에 관한 정보를 기재합니다.

Acc28E[#]	0	1	2	3	4	5	6	7
Cid[#]	12	13	14	15	28	29	30	31
ID 칩의 베이스 오프셋	\$D00180	\$D00190	\$D001C0	\$D001D0	\$D08180	\$D08190	\$D081C0	\$D081D0
Acc28E[#]	8	9	10	11	12	13	14	15
Cid[#]	44	45	46	47	60	61	62	63
ID 칩의 베이스 오프셋	\$D10180	\$D10190	\$D101C0	\$D101D0	\$D18180	\$D18190	\$D181C0	\$D181D0

폴 레지스터	부분 컴포넌트	ID 칩 베이스에서의 오프셋 비트	
Reg0		\$000	08-15
	Vendor ID bits 0-3*		08-11
	Revision number**		08-11
Reg1		\$004	08-15
	Vendor ID bits 4-7*		08-11
	Card number bits 0-3**		08-11
	Bank select output		12
Reg2		\$008	08-15
	Card options bits 0-4*		08-12
	Card number bits 4-8**		08-12
Reg3		\$00C	08-15
	Card options bits 5-9*		08-12
	Card number bits 9-13**		08-12

(*[Bank select output]의 값을 '0'으로 설정한 경우)

(**[Bank select output]의 값을 '1'로 설정한 경우)

ACC-36E(A/D 컨버터) 레지스터 요소

ACC-36E 다중 A/D 컨버터 카드에는 단일 24 비트 레지스터가 있으며 제어 레지스터로서의 쓰기 및 데이터 레지스터로서의 읽기가 가능하고 선택한 요소에서 부호가 없는('U') 값 또는 부호가 있는('S') 값 중 어느 하나를 선택한 다중 ADC에 액세스할 수 있습니다.

Acc36E[#]	0	1	2	3	4	5	6	7
IC 베이스 오프셋	\$A00000	\$B00000	\$C00000	\$D00000	\$A08000	\$B08000	\$C08000	\$D08000
Acc36E[#]	8	9	10	11	12	13	14	15
IC 베이스 오프셋	\$A10000	\$B10000	\$C10000	\$D10000	\$A18000	\$B18000	\$C18000	\$D18000

데이터 구조체	폴 요소	부분 요소	ASIC 베이스에서의 오프셋	비트
Acc36E[<i>j</i>].	ConvertCode (<i>write-only</i>)		\$000	08-31
	ADCHighLow (<i>read-only</i>)		\$000	08-31
		ADCsLow		08-19
		ADCuLow		08-19
		ADCsHigh		20-31
		ADCuHigh		20-31

ACC-36E에는 각각 카드에 관한 정보가 포함된 ID 칩이 탑재됩니다. 다음 표에 ID 칩의 기준 주소 오프셋 및 칩 내의 레지스터에 관한 정보를 기재합니다.

Acc36E[#]	0	1	2	3	4	5	6	7
Cid[#]	12	13	14	15	28	29	30	31
ID 칩의 베이스 오프셋	\$D00180	\$D00190	\$D001C0	\$D001D0	\$D08180	\$D08190	\$D081C0	\$D081D0
Acc36E[#]	8	9	10	11	12	13	14	15
Cid[#]	44	45	46	47	60	61	62	63
ID 칩의 베이스 오프셋	\$D10180	\$D10190	\$D101C0	\$D101D0	\$D18180	\$D18190	\$D181C0	\$D181D0

폴 레지스터	부분 컴포넌트	ID 칩 베이스에서의 오프셋	비트
Reg0		\$000	08-15
	Vendor ID bits 0-3*		08-11
	ADCRdy*		13
	Revision number**		08-11
Reg1		\$004	08-15
	Vendor ID bits 4-7*		08-11
	Card number bits 0-3**		08-11
	Bank select output		12
Reg2		\$008	08-15
	Card options bits 0-4*		08-12
	Card number bits 4-8**		08-12
Reg3		\$00C	08-15
	Card options bits 5-9*		08-12
	Card number bits 9-13**		08-12

(*[Bank select output]의 값을 '0'으로 설정한 경우)

(**[Bank select output]의 값을 '1'로 설정한 경우)

ACC-59E(A/D 및 D/A 컨버터) 레지스터 요소

ACC-59E 컨버터 카드에는 단일 24 비트 레지스터가 있으며 A/D 레지스터의 제어 레지스터로서의 쓰기 및 데이터 레지스터로서의 읽기가 가능하고 선택한 요소에서 부호가 없는('U') 값 또는 부호가 있는('S') 값 중 어느 하나를 선택한 다중 ADC 에 액세스할 수 있습니다. 또한 여기에는 24 비트 레지스터가 4 개 있고 각각 12 비트의 DAC 값이 2 개 포함됩니다.

Acc59E[#]	0	1	2	3	4	5	6	7
IC 베이스 오프셋	\$A00000	\$B00000	\$C00000	\$D00000	\$A08000	\$B08000	\$C08000	\$D08000
Acc59E[#]	8	9	10	11	12	13	14	15
IC 베이스 오프셋	\$A10000	\$B10000	\$C10000	\$D10000	\$A18000	\$B18000	\$C18000	\$D18000

데이터 구조체	폴 요소	부분 요소	ASIC 베이스에서의 오프셋	비트
Acc36E[7].	ConvertCode (write-only)		\$000	08-31
	ADCs (read-only)		\$000	08-19
	ADCu (read-only)		\$000	08-19
	DACHighLow[0]		\$008	08-31
		DAC[0]		08-19
		DAC[4]		20-31
	DACHighLow[1]		\$009	08-31
		DAC[1]		08-19
		DAC[5]		20-31
	DACHighLow[2]		\$00A	08-31
		DAC[2]		08-19
		DAC[6]		20-31
	DACHighLow[3]		\$00B	08-31
		DAC[3]		08-19
		DAC[7]		20-31

ACC-59E 에는 각각 카드에 관한 정보가 포함된 ID 칩이 탑재됩니다. 다음 표에 ID 칩의 기준 주소 오프셋 및 칩 내의 레지스터에 관한 정보를 기재합니다.

Acc59E[#]	0	1	2	3	4	5	6	7
Cid[#]	12	13	14	15	28	29	30	31
ID 칩의 베이스 오프셋	\$D00180	\$D00190	\$D001C0	\$D001D0	\$D08180	\$D08190	\$D081C0	\$D081D0
Acc59E[#]	8	9	10	11	12	13	14	15
Cid[#]	44	45	46	47	60	61	62	63
ID 칩의 베이스 오프셋	\$D10180	\$D10190	\$D101C0	\$D101D0	\$D18180	\$D18190	\$D181C0	\$D181D0

<u>풀 레지스터</u>	<u>부분 컴포넌트</u>	<u>ID 칩 베이스에서의 오프셋</u>	<u>비트</u>
<i>Reg0</i>	<i>Vendor ID bits 0-3*</i>	\$000	08-15
	<i>ADCRdy*</i>		08-11
	<i>Revision number**</i>		13
<i>Reg1</i>	<i>Vendor ID bits 4-7*</i>	\$004	08-11
	<i>Card number bits 0-3**</i>		08-15
	<i>Bank select output</i>		12
<i>Reg2</i>	<i>Card options bits 0-4*</i>	\$008	08-12
	<i>Card number bits 4-8**</i>		08-15
<i>Reg3</i>	<i>Card options bits 5-9*</i>	\$00C	08-12
	<i>Card number bits 9-13**</i>		08-15

(*[Bank select output]의 값을 '0'으로 설정한 경우)

(**[Bank select output]의 값을 '1'로 설정한 경우)

POWER PMAC 의 I/O 포인터 선언의 권장 사항

Power PMAC 의 'M'(포인터) 변수는 특히 디지털 및 아날로그 I/O 등 대상의 정보에 액세스하는 데 도움이 되는 방법입니다. 이 섹션에서는 일반적으로 사용하는 정보를 위해 이러한 변수에 사용자명을 선언할 때의 권장 사항에 대해 설명합니다. 이것은 어디까지나 권장 사항에 불과합니다. 어플리케이션 내에서 어느 I/O 포인트를 사용할지 나타내는 어플리케이션 고유 변수명으로 이러한 이름을 치환하는 경우가 많습니다.

UMAC I/O 카드

UMAC 의 랙 마운트 컨트롤러 시스템은 여러 종류의 디지털 및 아날로그 I/O 보드를 제공합니다. 이러한 보드는 Power PMAC 의 I/O 영역의 기준 주소에서의 주소 오프셋이 \$A00000~\$DFFFFFF 사이에 매핑됩니다. 하나의 UMAC 랙에 이러한 보드를 합계 16 개 사용할 수 있습니다. 사용하는 각 I/O 보드에는 DIP 스위치 SW1-1~SW1-4 를 임의로 설정해야 합니다.

이 섹션에서 설명하는 것처럼 디지털 I/O 카드의 포인터 변수를 선언하여 IOGATE IC 를 사용하는 UMAC 디지털 I/O 보드의 I/O 포인트 및 설정 레지스터에 액세스할 수 있습니다. 이러한 보드에는 ACC-11E, ACC-14E, ACC-65E, ACC-66E, ACC-67E 및 ACC-68E 등이 있습니다.

이 섹션에서 설명하는 것처럼 ACC-28E A/D 카드의 포인터 변수를 선언해서 ACC-28E 고분해능 A/D 컨버터 보드의 4 개의 16 비트 A/D 컨버터 레지스터에 액세스할 수 있습니다.

ACC-28E A/D 카드 0

(SW1-1 ON, SW1-2 ON, SW1-3 ON, SW1-4 ON)(기본값 스위치 설정)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard0Reg0->u.io:$A00000.16.16 // ACC-28E Card 0 ADC1
ptr AdcCard0Reg1->u.io:$A00004.16.16 // ACC-28E Card 0 ADC2
ptr AdcCard0Reg2->u.io:$A00008.16.16 // ACC-28E Card 0 ADC3
ptr AdcCard0Reg3->u.io:$A0000C.16.16 // ACC-28E Card 0 ADC4
```

ACC-28E A/D 카드 1

(SW1-1 OFF, SW1-2 ON, SW1-3 ON, SW1-4 ON)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard1Reg0->u.io:$B00000.16.16 // ACC-28E Card 1 ADC1
ptr AdcCard1Reg1->u.io:$B00004.16.16 // ACC-28E Card 1 ADC2
ptr AdcCard1Reg2->u.io:$B00008.16.16 // ACC-28E Card 1 ADC3
ptr AdcCard1Reg3->u.io:$B0000C.16.16 // ACC-28E Card 1 ADC4
```

ACC-28E A/D 카드 2

(SW1-1 ON, SW1-2 OFF, SW1-3 ON, SW1-4 ON)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard2Reg0->u.io:$C00000.16.16 // ACC-28E Card 2 ADC1
ptr AdcCard2Reg1->u.io:$C00004.16.16 // ACC-28E Card 2 ADC2
ptr AdcCard2Reg2->u.io:$C00008.16.16 // ACC-28E Card 2 ADC3
ptr AdcCard2Reg3->u.io:$C0000C.16.16 // ACC-28E Card 2 ADC4
```

ACC-28E A/D 카드 3

(SW1-1 OFF, SW1-2 OFF, SW1-3 ON, SW1-4 ON)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard3Reg0->u.io:$D00000.16.16 // ACC-28E Card 2 ADC1
ptr AdcCard3Reg1->u.io:$D00004.16.16 // ACC-28E Card 2 ADC2
ptr AdcCard3Reg2->u.io:$D00008.16.16 // ACC-28E Card 2 ADC3
ptr AdcCard3Reg3->u.io:$D0000C.16.16 // ACC-28E Card 2 ADC4
```

ACC-28E A/D 카드 4

(SW1-1 ON, SW1-2 ON, SW1-3 OFF, SW1-4 ON)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard4Reg0->u.io:$A08000.16.16 // ACC-28E Card 4 ADC1
ptr AdcCard4Reg1->u.io:$A08004.16.16 // ACC-28E Card 4 ADC2
ptr AdcCard4Reg2->u.io:$A08008.16.16 // ACC-28E Card 4 ADC3
ptr AdcCard4Reg3->u.io:$A0800C.16.16 // ACC-28E Card 4 ADC4
```

ACC-28E A/D 카드 5

(SW1-1 OFF, SW1-2 ON, SW1-3 OFF, SW1-4 ON)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard5Reg0->u.io:$B08000.16.16 // ACC-28E Card 5 ADC1
ptr AdcCard5Reg1->u.io:$B08004.16.16 // ACC-28E Card 5 ADC2
ptr AdcCard5Reg2->u.io:$B08008.16.16 // ACC-28E Card 5 ADC3
ptr AdcCard5Reg3->u.io:$B0800C.16.16 // ACC-28E Card 5 ADC4
```

ACC-28E A/D 카드 6

(SW1-1 ON, SW1-2 OFF, SW1-3 OFF, SW1-4 ON)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard6Reg0->u.io:$C08000.16.16 // ACC-28E Card 6 ADC1
ptr AdcCard6Reg1->u.io:$C08004.16.16 // ACC-28E Card 6 ADC2
ptr AdcCard6Reg2->u.io:$C08008.16.16 // ACC-28E Card 6 ADC3
ptr AdcCard6Reg3->u.io:$C0800C.16.16 // ACC-28E Card 6 ADC4
```

ACC-28E A/D 카드 7

(SW1-1 OFF, SW1-2 OFF, SW1-3 OFF, SW1-4 ON)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard7Reg0->u.io:$D08000.16.16 // ACC-28E Card 7 ADC1
ptr AdcCard7Reg1->u.io:$D08004.16.16 // ACC-28E Card 7 ADC2
ptr AdcCard7Reg2->u.io:$D08008.16.16 // ACC-28E Card 7 ADC3
ptr AdcCard7Reg3->u.io:$D0800C.16.16 // ACC-28E Card 7 ADC4
```

ACC-28E A/D 카드 8

(SW1-1 ON, SW1-2 ON, SW1-3 ON, SW1-4 OFF)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard8Reg0->u.io:$A10000.16.16 // ACC-28E Card 8 ADC1
ptr AdcCard8Reg1->u.io:$A10004.16.16 // ACC-28E Card 8 ADC2
ptr AdcCard8Reg2->u.io:$A10008.16.16 // ACC-28E Card 8 ADC3
ptr AdcCard8Reg3->u.io:$A1000C.16.16 // ACC-28E Card 8 ADC4
```

ACC-28E A/D 카드 9

(SW1-1 OFF, SW1-2 ON, SW1-3 ON, SW1-4 OFF)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard9Reg0->u.io:$B10000.16.16 // ACC-28E Card 9 ADC1
ptr AdcCard9Reg1->u.io:$B10004.16.16 // ACC-28E Card 9 ADC2
ptr AdcCard9Reg2->u.io:$B10008.16.16 // ACC-28E Card 9 ADC3
ptr AdcCard9Reg3->u.io:$B1000C.16.16 // ACC-28E Card 9 ADC4
```

ACC-28E A/D 카드 10

(SW1-1 ON, SW1-2 OFF, SW1-3 ON, SW1-4 OFF)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard10Reg0->u.io:$C10000.16.16 // ACC-28E Card 10 ADC1
ptr AdcCard10Reg1->u.io:$C10004.16.16 // ACC-28E Card 10 ADC2
ptr AdcCard10Reg2->u.io:$C10008.16.16 // ACC-28E Card 10 ADC3
ptr AdcCard10Reg3->u.io:$C1000C.16.16 // ACC-28E Card 10 ADC4
```

ACC-28E A/D 카드 11

(SW1-1 OFF, SW1-2 OFF, SW1-3 ON, SW1-4 OFF)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard11Reg0->u.io:$D10000.16.16 // ACC-28E Card 11 ADC1
ptr AdcCard11Reg1->u.io:$D10004.16.16 // ACC-28E Card 11 ADC2
ptr AdcCard11Reg2->u.io:$D10008.16.16 // ACC-28E Card 11 ADC3
ptr AdcCard11Reg3->u.io:$D1000C.16.16 // ACC-28E Card 11 ADC4
```

ACC-28E A/D 카드 12

(SW1-1 ON, SW1-2 ON, SW1-3 OFF, SW1-4 OFF)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard12Reg0->u.io:$A18000.16.16 // ACC-28E Card 12 ADC1
ptr AdcCard12Reg1->u.io:$A18004.16.16 // ACC-28E Card 12 ADC2
ptr AdcCard12Reg2->u.io:$A18008.16.16 // ACC-28E Card 12 ADC3
ptr AdcCard12Reg3->u.io:$A1800C.16.16 // ACC-28E Card 12 ADC4
```

ACC-28E A/D 카드 13

(SW1-1 OFF, SW1-2 ON, SW1-3 OFF, SW1-4 OFF)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard13Reg0->u.io:$B18000.16.16 // ACC-28E Card 13 ADC1
ptr AdcCard13Reg1->u.io:$B18004.16.16 // ACC-28E Card 13 ADC2
ptr AdcCard13Reg2->u.io:$B18008.16.16 // ACC-28E Card 13 ADC3
ptr AdcCard13Reg3->u.io:$B1800C.16.16 // ACC-28E Card 13 ADC4
```

ACC-28E A/D 카드 14

(SW1-1 ON, SW1-2 OFF, SW1-3 OFF, SW1-4 OFF)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard14Reg0->u.io:$C18000.16.16 // ACC-28E Card 14 ADC1
ptr AdcCard14Reg1->u.io:$C18004.16.16 // ACC-28E Card 14 ADC2
ptr AdcCard14Reg2->u.io:$C18008.16.16 // ACC-28E Card 14 ADC3
ptr AdcCard14Reg3->u.io:$C1800C.16.16 // ACC-28E Card 14 ADC4
```

ACC-28E A/D 카드 15

(SW1-1 OFF, SW1-2 OFF, SW1-3 OFF, SW1-4 OFF)

```
// 16-bit variables used for accessing A/D values
ptr AdcCard15Reg0->u.io:$D18000.16.16 // ACC-28E Card 15 ADC1
ptr AdcCard15Reg1->u.io:$D18004.16.16 // ACC-28E Card 15 ADC2
ptr AdcCard15Reg2->u.io:$D18008.16.16 // ACC-28E Card 15 ADC3
ptr AdcCard15Reg3->u.io:$D1800C.16.16 // ACC-28E Card 15 ADC4
```


디지털 I/O 카드 0

(SW1-1 ON, SW1-2 ON, SW1-3 ON, SW1-4 ON)(기본값 스위치 설정)UMAC 랙의 디지털 I/O 카드가 하나인 경우에 거의 언제나 이 기준 주소로 사용됩니다.

```
// Single-bit variables used for accessing I/O points
ptr IoCard0Pt00->u.io:$A00000.8.1      // I/O Card 0 I/000
ptr IoCard0Pt01->u.io:$A00000.9.1      // I/O Card 0 I/001
ptr IoCard0Pt02->u.io:$A00000.10.1     // I/O Card 0 I/002
ptr IoCard0Pt03->u.io:$A00000.11.1     // I/O Card 0 I/003
ptr IoCard0Pt04->u.io:$A00000.12.1     // I/O Card 0 I/004
ptr IoCard0Pt05->u.io:$A00000.13.1     // I/O Card 0 I/005
ptr IoCard0Pt06->u.io:$A00000.14.1     // I/O Card 0 I/006
ptr IoCard0Pt07->u.io:$A00000.15.1     // I/O Card 0 I/007
ptr IoCard0Pt08->u.io:$A00004.8.1      // I/O Card 0 I/008
ptr IoCard0Pt09->u.io:$A00004.9.1      // I/O Card 0 I/009
ptr IoCard0Pt10->u.io:$A00004.10.1     // I/O Card 0 I/010
ptr IoCard0Pt11->u.io:$A00004.11.1     // I/O Card 0 I/011
ptr IoCard0Pt12->u.io:$A00004.12.1     // I/O Card 0 I/012
ptr IoCard0Pt13->u.io:$A00004.13.1     // I/O Card 0 I/013
ptr IoCard0Pt14->u.io:$A00004.14.1     // I/O Card 0 I/014
ptr IoCard0Pt15->u.io:$A00004.15.1     // I/O Card 0 I/015
ptr IoCard0Pt16->u.io:$A00008.8.1      // I/O Card 0 I/016
ptr IoCard0Pt17->u.io:$A00008.9.1      // I/O Card 0 I/017
ptr IoCard0Pt18->u.io:$A00008.10.1     // I/O Card 0 I/018
ptr IoCard0Pt19->u.io:$A00008.11.1     // I/O Card 0 I/019
ptr IoCard0Pt20->u.io:$A00008.12.1     // I/O Card 0 I/020
ptr IoCard0Pt21->u.io:$A00008.13.1     // I/O Card 0 I/021
ptr IoCard0Pt22->u.io:$A00008.14.1     // I/O Card 0 I/022
ptr IoCard0Pt23->u.io:$A00008.15.1     // I/O Card 0 I/023
ptr IoCard0Pt24->u.io:$A0000C.8.1      // I/O Card 0 I/024
ptr IoCard0Pt25->u.io:$A0000C.9.1      // I/O Card 0 I/025
ptr IoCard0Pt26->u.io:$A0000C.10.1     // I/O Card 0 I/026
ptr IoCard0Pt27->u.io:$A0000C.11.1     // I/O Card 0 I/027
ptr IoCard0Pt28->u.io:$A0000C.12.1     // I/O Card 0 I/028
ptr IoCard0Pt29->u.io:$A0000C.13.1     // I/O Card 0 I/029
ptr IoCard0Pt30->u.io:$A0000C.14.1     // I/O Card 0 I/030
ptr IoCard0Pt31->u.io:$A0000C.15.1     // I/O Card 0 I/031
ptr IoCard0Pt32->u.io:$A00010.8.1      // I/O Card 0 I/032
ptr IoCard0Pt33->u.io:$A00010.9.1      // I/O Card 0 I/033
ptr IoCard0Pt34->u.io:$A00010.10.1     // I/O Card 0 I/034
ptr IoCard0Pt35->u.io:$A00010.11.1     // I/O Card 0 I/035
ptr IoCard0Pt36->u.io:$A00010.12.1     // I/O Card 0 I/036
ptr IoCard0Pt37->u.io:$A00010.13.1     // I/O Card 0 I/037
ptr IoCard0Pt38->u.io:$A00010.14.1     // I/O Card 0 I/038
ptr IoCard0Pt39->u.io:$A00010.15.1     // I/O Card 0 I/039
ptr IoCard0Pt40->u.io:$A00014.8.1      // I/O Card 0 I/040
ptr IoCard0Pt41->u.io:$A00014.9.1      // I/O Card 0 I/041
ptr IoCard0Pt42->u.io:$A00014.10.1     // I/O Card 0 I/042
ptr IoCard0Pt43->u.io:$A00014.11.1     // I/O Card 0 I/043
ptr IoCard0Pt44->u.io:$A00014.12.1     // I/O Card 0 I/044
ptr IoCard0Pt45->u.io:$A00014.13.1     // I/O Card 0 I/045
ptr IoCard0Pt46->u.io:$A00014.14.1     // I/O Card 0 I/046
ptr IoCard0Pt47->u.io:$A00014.15.1     // I/O Card 0 I/047

// Byte-wide variables used for power-on configuration
ptr IoCard0Reg0->u.io:$A00000.8.8      // I/O Card 0 I/000-07 as byte
ptr IoCard0Reg1->u.io:$A00004.8.8      // I/O Card 0 I/008-15 as byte
ptr IoCard0Reg2->u.io:$A00008.8.8      // I/O Card 0 I/016-23 as byte
ptr IoCard0Reg3->u.io:$A0000C.8.8      // I/O Card 0 I/024-31 as byte
ptr IoCard0Reg4->u.io:$A00010.8.8      // I/O Card 0 I/032-39 as byte
ptr IoCard0Reg5->u.io:$A00014.8.8      // I/O Card 0 I/040-47 as byte
ptr IoCard0Reg6->u.io:$A00018.8.8      // I/O Card 0 latch inputs
ptr IoCard0Reg7->u.io:$A0001C.8.8      // I/O Card 0 control register
```

디지털 I/O 카드 1

(SW1-1 OFF, SW1-2 ON, SW1-3 ON, SW1-4 ON)

```
// Single-bit variables used for accessing I/O points
ptr IoCard1Pt00->u.io:$B00000.8.1      // I/O Card 1 I/000
ptr IoCard1Pt01->u.io:$B00000.9.1      // I/O Card 1 I/001
ptr IoCard1Pt02->u.io:$B00000.10.1     // I/O Card 1 I/002
ptr IoCard1Pt03->u.io:$B00000.11.1     // I/O Card 1 I/003
ptr IoCard1Pt04->u.io:$B00000.12.1     // I/O Card 1 I/004
ptr IoCard1Pt05->u.io:$B00000.13.1     // I/O Card 1 I/005
ptr IoCard1Pt06->u.io:$B00000.14.1     // I/O Card 1 I/006
ptr IoCard1Pt07->u.io:$B00000.15.1     // I/O Card 1 I/007
ptr IoCard1Pt08->u.io:$B00004.8.1      // I/O Card 1 I/008
ptr IoCard1Pt09->u.io:$B00004.9.1      // I/O Card 1 I/009
ptr IoCard1Pt10->u.io:$B00004.10.1     // I/O Card 1 I/010
ptr IoCard1Pt11->u.io:$B00004.11.1     // I/O Card 1 I/011
ptr IoCard1Pt12->u.io:$B00004.12.1     // I/O Card 1 I/012
ptr IoCard1Pt13->u.io:$B00004.13.1     // I/O Card 1 I/013
ptr IoCard1Pt14->u.io:$B00004.14.1     // I/O Card 1 I/014
ptr IoCard1Pt15->u.io:$B00004.15.1     // I/O Card 1 I/015
ptr IoCard1Pt16->u.io:$B00008.8.1      // I/O Card 1 I/016
ptr IoCard1Pt17->u.io:$B00008.9.1      // I/O Card 1 I/017
ptr IoCard1Pt18->u.io:$B00008.10.1     // I/O Card 1 I/018
ptr IoCard1Pt19->u.io:$B00008.11.1     // I/O Card 1 I/019
ptr IoCard1Pt20->u.io:$B00008.12.1     // I/O Card 1 I/020
ptr IoCard1Pt21->u.io:$B00008.13.1     // I/O Card 1 I/021
ptr IoCard1Pt22->u.io:$B00008.14.1     // I/O Card 1 I/022
ptr IoCard1Pt23->u.io:$B00008.15.1     // I/O Card 1 I/023
ptr IoCard1Pt24->u.io:$B0000C.8.1      // I/O Card 1 I/024
ptr IoCard1Pt25->u.io:$B0000C.9.1      // I/O Card 1 I/025
ptr IoCard1Pt26->u.io:$B0000C.10.1     // I/O Card 1 I/026
ptr IoCard1Pt27->u.io:$B0000C.11.1     // I/O Card 1 I/027
ptr IoCard1Pt28->u.io:$B0000C.12.1     // I/O Card 1 I/028
ptr IoCard1Pt29->u.io:$B0000C.13.1     // I/O Card 1 I/029
ptr IoCard1Pt30->u.io:$B0000C.14.1     // I/O Card 1 I/030
ptr IoCard1Pt31->u.io:$B0000C.15.1     // I/O Card 1 I/031
ptr IoCard1Pt32->u.io:$B00010.8.1      // I/O Card 1 I/032
ptr IoCard1Pt33->u.io:$B00010.9.1      // I/O Card 1 I/033
ptr IoCard1Pt34->u.io:$B00010.10.1     // I/O Card 1 I/034
ptr IoCard1Pt35->u.io:$B00010.11.1     // I/O Card 1 I/035
ptr IoCard1Pt36->u.io:$B00010.12.1     // I/O Card 1 I/036
ptr IoCard1Pt37->u.io:$B00010.13.1     // I/O Card 1 I/037
ptr IoCard1Pt38->u.io:$B00010.14.1     // I/O Card 1 I/038
ptr IoCard1Pt39->u.io:$B00010.15.1     // I/O Card 1 I/039
ptr IoCard1Pt40->u.io:$B00014.8.1      // I/O Card 1 I/040
ptr IoCard1Pt41->u.io:$B00014.9.1      // I/O Card 1 I/041
ptr IoCard1Pt42->u.io:$B00014.10.1     // I/O Card 1 I/042
ptr IoCard1Pt43->u.io:$B00014.11.1     // I/O Card 1 I/043
ptr IoCard1Pt44->u.io:$B00014.12.1     // I/O Card 1 I/044
ptr IoCard1Pt45->u.io:$B00014.13.1     // I/O Card 1 I/045
ptr IoCard1Pt46->u.io:$B00014.14.1     // I/O Card 1 I/046
ptr IoCard1Pt47->u.io:$B00014.15.1     // I/O Card 1 I/047

// Byte-wide variables used for power-on configuration
ptr IoCard1Reg0->u.io:$B00000.8.8      // I/O Card 1 I/000-07 as byte
ptr IoCard1Reg1->u.io:$B00004.8.8      // I/O Card 1 I/008-15 as byte
ptr IoCard1Reg2->u.io:$B00008.8.8      // I/O Card 1 I/016-23 as byte
ptr IoCard1Reg3->u.io:$B0000C.8.8      // I/O Card 1 I/024-31 as byte
ptr IoCard1Reg4->u.io:$B00010.8.8      // I/O Card 1 I/032-39 as byte
ptr IoCard1Reg5->u.io:$B00014.8.8      // I/O Card 1 I/040-47 as byte
ptr IoCard1Reg6->u.io:$B00018.8.8      // I/O Card 1 latch inputs
ptr IoCard1Reg7->u.io:$B0001C.8.8      // I/O Card 1 control register
```

디지털 I/O 카드 2

(SW1-1 ON, SW1-2 OFF, SW1-3 ON, SW1-4 ON)

```
// Single-bit variables used for accessing I/O points
ptr IoCard2Pt00->u.io:$C00000.8.1      // I/O Card 2 I/000
ptr IoCard2Pt01->u.io:$C00000.9.1      // I/O Card 2 I/001
ptr IoCard2Pt02->u.io:$C00000.10.1     // I/O Card 2 I/002
ptr IoCard2Pt03->u.io:$C00000.11.1     // I/O Card 2 I/003
ptr IoCard2Pt04->u.io:$C00000.12.1     // I/O Card 2 I/004
ptr IoCard2Pt05->u.io:$C00000.13.1     // I/O Card 2 I/005
ptr IoCard2Pt06->u.io:$C00000.14.1     // I/O Card 2 I/006
ptr IoCard2Pt07->u.io:$C00000.15.1     // I/O Card 2 I/007
ptr IoCard2Pt08->u.io:$C00004.8.1      // I/O Card 2 I/008
ptr IoCard2Pt09->u.io:$C00004.9.1      // I/O Card 2 I/009
ptr IoCard2Pt10->u.io:$C00004.10.1     // I/O Card 2 I/010
ptr IoCard2Pt11->u.io:$C00004.11.1     // I/O Card 2 I/011
ptr IoCard2Pt12->u.io:$C00004.12.1     // I/O Card 2 I/012
ptr IoCard2Pt13->u.io:$C00004.13.1     // I/O Card 2 I/013
ptr IoCard2Pt14->u.io:$C00004.14.1     // I/O Card 2 I/014
ptr IoCard2Pt15->u.io:$C00004.15.1     // I/O Card 2 I/015
ptr IoCard2Pt16->u.io:$C00008.8.1      // I/O Card 2 I/016
ptr IoCard2Pt17->u.io:$C00008.9.1      // I/O Card 2 I/017
ptr IoCard2Pt18->u.io:$C00008.10.1     // I/O Card 2 I/018
ptr IoCard2Pt19->u.io:$C00008.11.1     // I/O Card 2 I/019
ptr IoCard2Pt20->u.io:$C00008.12.1     // I/O Card 2 I/020
ptr IoCard2Pt21->u.io:$C00008.13.1     // I/O Card 2 I/021
ptr IoCard2Pt22->u.io:$C00008.14.1     // I/O Card 2 I/022
ptr IoCard2Pt23->u.io:$C00008.15.1     // I/O Card 2 I/023
ptr IoCard2Pt24->u.io:$C0000C.8.1      // I/O Card 2 I/024
ptr IoCard2Pt25->u.io:$C0000C.9.1      // I/O Card 2 I/025
ptr IoCard2Pt26->u.io:$C0000C.10.1     // I/O Card 2 I/026
ptr IoCard2Pt27->u.io:$C0000C.11.1     // I/O Card 2 I/027
ptr IoCard2Pt28->u.io:$C0000C.12.1     // I/O Card 2 I/028
ptr IoCard2Pt29->u.io:$C0000C.13.1     // I/O Card 2 I/029
ptr IoCard2Pt30->u.io:$C0000C.14.1     // I/O Card 2 I/030
ptr IoCard2Pt31->u.io:$C0000C.15.1     // I/O Card 2 I/031
ptr IoCard2Pt32->u.io:$C00010.8.1      // I/O Card 2 I/032
ptr IoCard2Pt33->u.io:$C00010.9.1      // I/O Card 2 I/033
ptr IoCard2Pt34->u.io:$C00010.10.1     // I/O Card 2 I/034
ptr IoCard2Pt35->u.io:$C00010.11.1     // I/O Card 2 I/035
ptr IoCard2Pt36->u.io:$C00010.12.1     // I/O Card 2 I/036
ptr IoCard2Pt37->u.io:$C00010.13.1     // I/O Card 2 I/037
ptr IoCard2Pt38->u.io:$C00010.14.1     // I/O Card 2 I/038
ptr IoCard2Pt39->u.io:$C00010.15.1     // I/O Card 2 I/039
ptr IoCard2Pt40->u.io:$C00014.8.1      // I/O Card 2 I/040
ptr IoCard2Pt41->u.io:$C00014.9.1      // I/O Card 2 I/041
ptr IoCard2Pt42->u.io:$C00014.10.1     // I/O Card 2 I/042
ptr IoCard2Pt43->u.io:$C00014.11.1     // I/O Card 2 I/043
ptr IoCard2Pt44->u.io:$C00014.12.1     // I/O Card 2 I/044
ptr IoCard2Pt45->u.io:$C00014.13.1     // I/O Card 2 I/045
ptr IoCard2Pt46->u.io:$C00014.14.1     // I/O Card 2 I/046
ptr IoCard2Pt47->u.io:$C00014.15.1     // I/O Card 2 I/047

// Byte-wide variables used for power-on configuration
ptr IoCard2Reg0->u.io:$C00000.8.8      // I/O Card 2 I/000-07 as byte
ptr IoCard2Reg1->u.io:$C00004.8.8      // I/O Card 2 I/008-15 as byte
ptr IoCard2Reg2->u.io:$C00008.8.8      // I/O Card 2 I/016-23 as byte
ptr IoCard2Reg3->u.io:$C0000C.8.8      // I/O Card 2 I/024-31 as byte
ptr IoCard2Reg4->u.io:$C00010.8.8      // I/O Card 2 I/032-39 as byte
ptr IoCard2Reg5->u.io:$C00014.8.8      // I/O Card 2 I/040-47 as byte
ptr IoCard2Reg6->u.io:$C00018.8.8      // I/O Card 2 latch inputs
ptr IoCard2Reg7->u.io:$C0001C.8.8      // I/O Card 2 control register
```

디지털 I/O 카드 3

(SW1-1 OFF, SW1-2 OFF, SW1-3 ON, SW1-4 ON)

```
// Single-bit variables used for accessing I/O points
ptr IoCard3Pt00->u.io:$D00000.8.1      // I/O Card 3 I/000
ptr IoCard3Pt01->u.io:$D00000.9.1      // I/O Card 3 I/001
ptr IoCard3Pt02->u.io:$D00000.10.1     // I/O Card 3 I/002
ptr IoCard3Pt03->u.io:$D00000.11.1     // I/O Card 3 I/003
ptr IoCard3Pt04->u.io:$D00000.12.1     // I/O Card 3 I/004
ptr IoCard3Pt05->u.io:$D00000.13.1     // I/O Card 3 I/005
ptr IoCard3Pt06->u.io:$D00000.14.1     // I/O Card 3 I/006
ptr IoCard3Pt07->u.io:$D00000.15.1     // I/O Card 3 I/007
ptr IoCard3Pt08->u.io:$D00004.8.1      // I/O Card 3 I/008
ptr IoCard3Pt09->u.io:$D00004.9.1      // I/O Card 3 I/009
ptr IoCard3Pt10->u.io:$D00004.10.1     // I/O Card 3 I/010
ptr IoCard3Pt11->u.io:$D00004.11.1     // I/O Card 3 I/011
ptr IoCard3Pt12->u.io:$D00004.12.1     // I/O Card 3 I/012
ptr IoCard3Pt13->u.io:$D00004.13.1     // I/O Card 3 I/013
ptr IoCard3Pt14->u.io:$D00004.14.1     // I/O Card 3 I/014
ptr IoCard3Pt15->u.io:$D00004.15.1     // I/O Card 3 I/015
ptr IoCard3Pt16->u.io:$D00008.8.1      // I/O Card 3 I/016
ptr IoCard3Pt17->u.io:$D00008.9.1      // I/O Card 3 I/017
ptr IoCard3Pt18->u.io:$D00008.10.1     // I/O Card 3 I/018
ptr IoCard3Pt19->u.io:$D00008.11.1     // I/O Card 3 I/019
ptr IoCard3Pt20->u.io:$D00008.12.1     // I/O Card 3 I/020
ptr IoCard3Pt21->u.io:$D00008.13.1     // I/O Card 3 I/021
ptr IoCard3Pt22->u.io:$D00008.14.1     // I/O Card 3 I/022
ptr IoCard3Pt23->u.io:$D00008.15.1     // I/O Card 3 I/023
ptr IoCard3Pt24->u.io:$D0000C.8.1      // I/O Card 3 I/024
ptr IoCard3Pt25->u.io:$D0000C.9.1      // I/O Card 3 I/025
ptr IoCard3Pt26->u.io:$D0000C.10.1     // I/O Card 3 I/026
ptr IoCard3Pt27->u.io:$D0000C.11.1     // I/O Card 3 I/027
ptr IoCard3Pt28->u.io:$D0000C.12.1     // I/O Card 3 I/028
ptr IoCard3Pt29->u.io:$D0000C.13.1     // I/O Card 3 I/029
ptr IoCard3Pt30->u.io:$D0000C.14.1     // I/O Card 3 I/030
ptr IoCard3Pt31->u.io:$D0000C.15.1     // I/O Card 3 I/031
ptr IoCard3Pt32->u.io:$D00010.8.1      // I/O Card 3 I/032
ptr IoCard3Pt33->u.io:$D00010.9.1      // I/O Card 3 I/033
ptr IoCard3Pt34->u.io:$D00010.10.1     // I/O Card 3 I/034
ptr IoCard3Pt35->u.io:$D00010.11.1     // I/O Card 3 I/035
ptr IoCard3Pt36->u.io:$D00010.12.1     // I/O Card 3 I/036
ptr IoCard3Pt37->u.io:$D00010.13.1     // I/O Card 3 I/037
ptr IoCard3Pt38->u.io:$D00010.14.1     // I/O Card 3 I/038
ptr IoCard3Pt39->u.io:$D00010.15.1     // I/O Card 3 I/039
ptr IoCard3Pt40->u.io:$D00014.8.1      // I/O Card 3 I/040
ptr IoCard3Pt41->u.io:$D00014.9.1      // I/O Card 3 I/041
ptr IoCard3Pt42->u.io:$D00014.10.1     // I/O Card 3 I/042
ptr IoCard3Pt43->u.io:$D00014.11.1     // I/O Card 3 I/043
ptr IoCard3Pt44->u.io:$D00014.12.1     // I/O Card 3 I/044
ptr IoCard3Pt45->u.io:$D00014.13.1     // I/O Card 3 I/045
ptr IoCard3Pt46->u.io:$D00014.14.1     // I/O Card 3 I/046
ptr IoCard3Pt47->u.io:$D00014.15.1     // I/O Card 3 I/047

// Byte-wide variables used for power-on configuration
ptr IoCard3Reg0->u.io:$D00000.8.8      // I/O Card 3 I/000-07 as byte
ptr IoCard3Reg1->u.io:$D00004.8.8      // I/O Card 3 I/008-15 as byte
ptr IoCard3Reg2->u.io:$D00008.8.8      // I/O Card 3 I/016-23 as byte
ptr IoCard3Reg3->u.io:$D0000C.8.8      // I/O Card 3 I/024-31 as byte
ptr IoCard3Reg4->u.io:$D00010.8.8      // I/O Card 3 I/032-39 as byte
ptr IoCard3Reg5->u.io:$D00014.8.8      // I/O Card 3 I/040-47 as byte
ptr IoCard3Reg6->u.io:$D00018.8.8      // I/O Card 3 latch inputs
ptr IoCard3Reg7->u.io:$D0001C.8.8      // I/O Card 3 control register
```

디지털 I/O 카드 4

(SW1-1 ON, SW1-2 ON, SW1-3 OFF, SW1-4 ON)

```
// Single-bit variables used for accessing I/O points
ptr IoCard4Pt00->u.io:$A08000.8.1      // I/O Card 4 I/000
ptr IoCard4Pt01->u.io:$A08000.9.1      // I/O Card 4 I/001
ptr IoCard4Pt02->u.io:$A08000.10.1     // I/O Card 4 I/002
ptr IoCard4Pt03->u.io:$A08000.11.1     // I/O Card 4 I/003
ptr IoCard4Pt04->u.io:$A08000.12.1     // I/O Card 4 I/004
ptr IoCard4Pt05->u.io:$A08000.13.1     // I/O Card 4 I/005
ptr IoCard4Pt06->u.io:$A08000.14.1     // I/O Card 4 I/006
ptr IoCard4Pt07->u.io:$A08000.15.1     // I/O Card 4 I/007
ptr IoCard4Pt08->u.io:$A08004.8.1      // I/O Card 4 I/008
ptr IoCard4Pt09->u.io:$A08004.9.1      // I/O Card 4 I/009
ptr IoCard4Pt10->u.io:$A08004.10.1     // I/O Card 4 I/010
ptr IoCard4Pt11->u.io:$A08004.11.1     // I/O Card 4 I/011
ptr IoCard4Pt12->u.io:$A08004.12.1     // I/O Card 4 I/012
ptr IoCard4Pt13->u.io:$A08004.13.1     // I/O Card 4 I/013
ptr IoCard4Pt14->u.io:$A08004.14.1     // I/O Card 4 I/014
ptr IoCard4Pt15->u.io:$A08004.15.1     // I/O Card 4 I/015
ptr IoCard4Pt16->u.io:$A08008.8.1      // I/O Card 4 I/016
ptr IoCard4Pt17->u.io:$A08008.9.1      // I/O Card 4 I/017
ptr IoCard4Pt18->u.io:$A08008.10.1     // I/O Card 4 I/018
ptr IoCard4Pt19->u.io:$A08008.11.1     // I/O Card 4 I/019
ptr IoCard4Pt20->u.io:$A08008.12.1     // I/O Card 4 I/020
ptr IoCard4Pt21->u.io:$A08008.13.1     // I/O Card 4 I/021
ptr IoCard4Pt22->u.io:$A08008.14.1     // I/O Card 4 I/022
ptr IoCard4Pt23->u.io:$A08008.15.1     // I/O Card 4 I/023
ptr IoCard4Pt24->u.io:$A0800C.8.1      // I/O Card 4 I/024
ptr IoCard4Pt25->u.io:$A0800C.9.1      // I/O Card 4 I/025
ptr IoCard4Pt26->u.io:$A0800C.10.1     // I/O Card 4 I/026
ptr IoCard4Pt27->u.io:$A0800C.11.1     // I/O Card 4 I/027
ptr IoCard4Pt28->u.io:$A0800C.12.1     // I/O Card 4 I/028
ptr IoCard4Pt29->u.io:$A0800C.13.1     // I/O Card 4 I/029
ptr IoCard4Pt30->u.io:$A0800C.14.1     // I/O Card 4 I/030
ptr IoCard4Pt31->u.io:$A0800C.15.1     // I/O Card 4 I/031
ptr IoCard4Pt32->u.io:$A08010.8.1      // I/O Card 4 I/032
ptr IoCard4Pt33->u.io:$A08010.9.1      // I/O Card 4 I/033
ptr IoCard4Pt34->u.io:$A08010.10.1     // I/O Card 4 I/034
ptr IoCard4Pt35->u.io:$A08010.11.1     // I/O Card 4 I/035
ptr IoCard4Pt36->u.io:$A08010.12.1     // I/O Card 4 I/036
ptr IoCard4Pt37->u.io:$A08010.13.1     // I/O Card 4 I/037
ptr IoCard4Pt38->u.io:$A08010.14.1     // I/O Card 4 I/038
ptr IoCard4Pt39->u.io:$A08010.15.1     // I/O Card 4 I/039
ptr IoCard4Pt40->u.io:$A08014.8.1      // I/O Card 4 I/040
ptr IoCard4Pt41->u.io:$A08014.9.1      // I/O Card 4 I/041
ptr IoCard4Pt42->u.io:$A08014.10.1     // I/O Card 4 I/042
ptr IoCard4Pt43->u.io:$A08014.11.1     // I/O Card 4 I/043
ptr IoCard4Pt44->u.io:$A08014.12.1     // I/O Card 4 I/044
ptr IoCard4Pt45->u.io:$A08014.13.1     // I/O Card 4 I/045
ptr IoCard4Pt46->u.io:$A08014.14.1     // I/O Card 4 I/046
ptr IoCard4Pt47->u.io:$A08014.15.1     // I/O Card 4 I/047

// Byte-wide variables used for power-on configuration
ptr IoCard4Reg0->u.io:$A08000.8.8      // I/O Card 4 I/000-07 as byte
ptr IoCard4Reg1->u.io:$A08004.8.8      // I/O Card 4 I/008-15 as byte
ptr IoCard4Reg2->u.io:$A08008.8.8      // I/O Card 4 I/016-23 as byte
ptr IoCard4Reg3->u.io:$A0800C.8.8      // I/O Card 4 I/024-31 as byte
ptr IoCard4Reg4->u.io:$A08010.8.8      // I/O Card 4 I/032-39 as byte
ptr IoCard4Reg5->u.io:$A08014.8.8      // I/O Card 4 I/040-47 as byte
ptr IoCard4Reg6->u.io:$A08018.8.8      // I/O Card 4 latch inputs
ptr IoCard4Reg7->u.io:$A0801C.8.8      // I/O Card 4 control register
```


디지털 I/O 카드 5

(SW1-1 OFF, SW1-2 ON, SW1-3 OFF, SW1-4 ON)

```
// Single-bit variables used for accessing I/O points
ptr IoCard5Pt00->u.io:$B08000.8.1      // I/O Card 5 I/000
ptr IoCard5Pt01->u.io:$B08000.9.1      // I/O Card 5 I/001
ptr IoCard5Pt02->u.io:$B08000.10.1     // I/O Card 5 I/002
ptr IoCard5Pt03->u.io:$B08000.11.1     // I/O Card 5 I/003
ptr IoCard5Pt04->u.io:$B08000.12.1     // I/O Card 5 I/004
ptr IoCard5Pt05->u.io:$B08000.13.1     // I/O Card 5 I/005
ptr IoCard5Pt06->u.io:$B08000.14.1     // I/O Card 5 I/006
ptr IoCard5Pt07->u.io:$B08000.15.1     // I/O Card 5 I/007
ptr IoCard5Pt08->u.io:$B08004.8.1      // I/O Card 5 I/008
ptr IoCard5Pt09->u.io:$B08004.9.1      // I/O Card 5 I/009
ptr IoCard5Pt10->u.io:$B08004.10.1     // I/O Card 5 I/010
ptr IoCard5Pt11->u.io:$B08004.11.1     // I/O Card 5 I/011
ptr IoCard5Pt12->u.io:$B08004.12.1     // I/O Card 5 I/012
ptr IoCard5Pt13->u.io:$B08004.13.1     // I/O Card 5 I/013
ptr IoCard5Pt14->u.io:$B08004.14.1     // I/O Card 5 I/014
ptr IoCard5Pt15->u.io:$B08004.15.1     // I/O Card 5 I/015
ptr IoCard5Pt16->u.io:$B08008.8.1      // I/O Card 5 I/016
ptr IoCard5Pt17->u.io:$B08008.9.1      // I/O Card 5 I/017
ptr IoCard5Pt18->u.io:$B08008.10.1     // I/O Card 5 I/018
ptr IoCard5Pt19->u.io:$B08008.11.1     // I/O Card 5 I/019
ptr IoCard5Pt20->u.io:$B08008.12.1     // I/O Card 5 I/020
ptr IoCard5Pt21->u.io:$B08008.13.1     // I/O Card 5 I/021
ptr IoCard5Pt22->u.io:$B08008.14.1     // I/O Card 5 I/022
ptr IoCard5Pt23->u.io:$B08008.15.1     // I/O Card 5 I/023
ptr IoCard5Pt24->u.io:$B0800C.8.1      // I/O Card 5 I/024
ptr IoCard5Pt25->u.io:$B0800C.9.1      // I/O Card 5 I/025
ptr IoCard5Pt26->u.io:$B0800C.10.1     // I/O Card 5 I/026
ptr IoCard5Pt27->u.io:$B0800C.11.1     // I/O Card 5 I/027
ptr IoCard5Pt28->u.io:$B0800C.12.1     // I/O Card 5 I/028
ptr IoCard5Pt29->u.io:$B0800C.13.1     // I/O Card 5 I/029
ptr IoCard5Pt30->u.io:$B0800C.14.1     // I/O Card 5 I/030
ptr IoCard5Pt31->u.io:$B0800C.15.1     // I/O Card 5 I/031
ptr IoCard5Pt32->u.io:$B08010.8.1      // I/O Card 5 I/032
ptr IoCard5Pt33->u.io:$B08010.9.1      // I/O Card 5 I/033
ptr IoCard5Pt34->u.io:$B08010.10.1     // I/O Card 5 I/034
ptr IoCard5Pt35->u.io:$B08010.11.1     // I/O Card 5 I/035
ptr IoCard5Pt36->u.io:$B08010.12.1     // I/O Card 5 I/036
ptr IoCard5Pt37->u.io:$B08010.13.1     // I/O Card 5 I/037
ptr IoCard5Pt38->u.io:$B08010.14.1     // I/O Card 5 I/038
ptr IoCard5Pt39->u.io:$B08010.15.1     // I/O Card 5 I/039
ptr IoCard5Pt40->u.io:$B08014.8.1      // I/O Card 5 I/040
ptr IoCard5Pt41->u.io:$B08014.9.1      // I/O Card 5 I/041
ptr IoCard5Pt42->u.io:$B08014.10.1     // I/O Card 5 I/042
ptr IoCard5Pt43->u.io:$B08014.11.1     // I/O Card 5 I/043
ptr IoCard5Pt44->u.io:$B08014.12.1     // I/O Card 5 I/044
ptr IoCard5Pt45->u.io:$B08014.13.1     // I/O Card 5 I/045
ptr IoCard5Pt46->u.io:$B08014.14.1     // I/O Card 5 I/046
ptr IoCard5Pt47->u.io:$B08014.15.1     // I/O Card 5 I/047

// Byte-wide variables used for power-on configuration
ptr IoCard5Reg0->u.io:$B08000.8.8      // I/O Card 5 I/000-07 as byte
ptr IoCard5Reg1->u.io:$B08004.8.8      // I/O Card 5 I/008-15 as byte
ptr IoCard5Reg2->u.io:$B08008.8.8      // I/O Card 5 I/016-23 as byte
ptr IoCard5Reg3->u.io:$B0800C.8.8      // I/O Card 5 I/024-31 as byte
ptr IoCard5Reg4->u.io:$B08010.8.8      // I/O Card 5 I/032-39 as byte
ptr IoCard5Reg5->u.io:$B08014.8.8      // I/O Card 5 I/040-47 as byte
ptr IoCard5Reg6->u.io:$B08018.8.8      // I/O Card 5 latch inputs
ptr IoCard5Reg7->u.io:$B0801C.8.8      // I/O Card 5 control register
```

디지털 I/O 카드 6

(SW1-1 ON, SW1-2 OFF, SW1-3 OFF, SW1-4 ON)

```
// Single-bit variables used for accessing I/O points
ptr IoCard6Pt00->u.io:$C08000.8.1      // I/O Card 6 I/000
ptr IoCard6Pt01->u.io:$C08000.9.1      // I/O Card 6 I/001
ptr IoCard6Pt02->u.io:$C08000.10.1     // I/O Card 6 I/002
ptr IoCard6Pt03->u.io:$C08000.11.1     // I/O Card 6 I/003
ptr IoCard6Pt04->u.io:$C08000.12.1     // I/O Card 6 I/004
ptr IoCard6Pt05->u.io:$C08000.13.1     // I/O Card 6 I/005
ptr IoCard6Pt06->u.io:$C08000.14.1     // I/O Card 6 I/006
ptr IoCard6Pt07->u.io:$C08000.15.1     // I/O Card 6 I/007
ptr IoCard6Pt08->u.io:$C08004.8.1      // I/O Card 6 I/008
ptr IoCard6Pt09->u.io:$C08004.9.1      // I/O Card 6 I/009
ptr IoCard6Pt10->u.io:$C08004.10.1     // I/O Card 6 I/010
ptr IoCard6Pt11->u.io:$C08004.11.1     // I/O Card 6 I/011
ptr IoCard6Pt12->u.io:$C08004.12.1     // I/O Card 6 I/012
ptr IoCard6Pt13->u.io:$C08004.13.1     // I/O Card 6 I/013
ptr IoCard6Pt14->u.io:$C08004.14.1     // I/O Card 6 I/014
ptr IoCard6Pt15->u.io:$C08004.15.1     // I/O Card 6 I/015
ptr IoCard6Pt16->u.io:$C08008.8.1      // I/O Card 6 I/016
ptr IoCard6Pt17->u.io:$C08008.9.1      // I/O Card 6 I/017
ptr IoCard6Pt18->u.io:$C08008.10.1     // I/O Card 6 I/018
ptr IoCard6Pt19->u.io:$C08008.11.1     // I/O Card 6 I/019
ptr IoCard6Pt20->u.io:$C08008.12.1     // I/O Card 6 I/020
ptr IoCard6Pt21->u.io:$C08008.13.1     // I/O Card 6 I/021
ptr IoCard6Pt22->u.io:$C08008.14.1     // I/O Card 6 I/022
ptr IoCard6Pt23->u.io:$C08008.15.1     // I/O Card 6 I/023
ptr IoCard6Pt24->u.io:$C0800C.8.1      // I/O Card 6 I/024
ptr IoCard6Pt25->u.io:$C0800C.9.1      // I/O Card 6 I/025
ptr IoCard6Pt26->u.io:$C0800C.10.1     // I/O Card 6 I/026
ptr IoCard6Pt27->u.io:$C0800C.11.1     // I/O Card 6 I/027
ptr IoCard6Pt28->u.io:$C0800C.12.1     // I/O Card 6 I/028
ptr IoCard6Pt29->u.io:$C0800C.13.1     // I/O Card 6 I/029
ptr IoCard6Pt30->u.io:$C0800C.14.1     // I/O Card 6 I/030
ptr IoCard6Pt31->u.io:$C0800C.15.1     // I/O Card 6 I/031
ptr IoCard6Pt32->u.io:$C08010.8.1      // I/O Card 6 I/032
ptr IoCard6Pt33->u.io:$C08010.9.1      // I/O Card 6 I/033
ptr IoCard6Pt34->u.io:$C08010.10.1     // I/O Card 6 I/034
ptr IoCard6Pt35->u.io:$C08010.11.1     // I/O Card 6 I/035
ptr IoCard6Pt36->u.io:$C08010.12.1     // I/O Card 6 I/036
ptr IoCard6Pt37->u.io:$C08010.13.1     // I/O Card 6 I/037
ptr IoCard6Pt38->u.io:$C08010.14.1     // I/O Card 6 I/038
ptr IoCard6Pt39->u.io:$C08010.15.1     // I/O Card 6 I/039
ptr IoCard6Pt40->u.io:$C08014.8.1      // I/O Card 6 I/040
ptr IoCard6Pt41->u.io:$C08014.9.1      // I/O Card 6 I/041
ptr IoCard6Pt42->u.io:$C08014.10.1     // I/O Card 6 I/042
ptr IoCard6Pt43->u.io:$C08014.11.1     // I/O Card 6 I/043
ptr IoCard6Pt44->u.io:$C08014.12.1     // I/O Card 6 I/044
ptr IoCard6Pt45->u.io:$C08014.13.1     // I/O Card 6 I/045
ptr IoCard6Pt46->u.io:$C08014.14.1     // I/O Card 6 I/046
ptr IoCard6Pt47->u.io:$C08014.15.1     // I/O Card 6 I/047
// Byte-wide variables used for power-on configuration
ptr IoCard6Reg0->u.io:$C08000.8.8      // I/O Card 6 I/000-07 as byte
ptr IoCard6Reg1->u.io:$C08004.8.8      // I/O Card 6 I/008-15 as byte
ptr IoCard6Reg2->u.io:$C08008.8.8      // I/O Card 6 I/016-23 as byte
ptr IoCard6Reg3->u.io:$C0800C.8.8      // I/O Card 6 I/024-31 as byte
ptr IoCard6Reg4->u.io:$C08010.8.8      // I/O Card 6 I/032-39 as byte
ptr IoCard6Reg5->u.io:$C08014.8.8      // I/O Card 6 I/040-47 as byte
ptr IoCard6Reg6->u.io:$C08018.8.8      // I/O Card 6 latch inputs
ptr IoCard6Reg7->u.io:$C0801C.8.8      // I/O Card 6 control register
```

디지털 I/O 카드 7

(SW1-1 OFF, SW1-2 OFF, SW1-3 OFF, SW1-4 ON)

```
// Single-bit variables used for accessing I/O points
ptr IoCard7Pt00->u.io:$D08000.8.1      // I/O Card 7 I/000
ptr IoCard7Pt01->u.io:$D08000.9.1      // I/O Card 7 I/001
ptr IoCard7Pt02->u.io:$D08000.10.1     // I/O Card 7 I/002
ptr IoCard7Pt03->u.io:$D08000.11.1     // I/O Card 7 I/003
ptr IoCard7Pt04->u.io:$D08000.12.1     // I/O Card 7 I/004
ptr IoCard7Pt05->u.io:$D08000.13.1     // I/O Card 7 I/005
ptr IoCard7Pt06->u.io:$D08000.14.1     // I/O Card 7 I/006
ptr IoCard7Pt07->u.io:$D08000.15.1     // I/O Card 7 I/007
ptr IoCard7Pt08->u.io:$D08004.8.1      // I/O Card 7 I/008
ptr IoCard7Pt09->u.io:$D08004.9.1      // I/O Card 7 I/009
ptr IoCard7Pt10->u.io:$D08004.10.1     // I/O Card 7 I/010
ptr IoCard7Pt11->u.io:$D08004.11.1     // I/O Card 7 I/011
ptr IoCard7Pt12->u.io:$D08004.12.1     // I/O Card 7 I/012
ptr IoCard7Pt13->u.io:$D08004.13.1     // I/O Card 7 I/013
ptr IoCard7Pt14->u.io:$D08004.14.1     // I/O Card 7 I/014
ptr IoCard7Pt15->u.io:$D08004.15.1     // I/O Card 7 I/015
ptr IoCard7Pt16->u.io:$D08008.8.1      // I/O Card 7 I/016
ptr IoCard7Pt17->u.io:$D08008.9.1      // I/O Card 7 I/017
ptr IoCard7Pt18->u.io:$D08008.10.1     // I/O Card 7 I/018
ptr IoCard7Pt19->u.io:$D08008.11.1     // I/O Card 7 I/019
ptr IoCard7Pt20->u.io:$D08008.12.1     // I/O Card 7 I/020
ptr IoCard7Pt21->u.io:$D08008.13.1     // I/O Card 7 I/021
ptr IoCard7Pt22->u.io:$D08008.14.1     // I/O Card 7 I/022
ptr IoCard7Pt23->u.io:$D08008.15.1     // I/O Card 7 I/023
ptr IoCard7Pt24->u.io:$D0800C.8.1      // I/O Card 7 I/024
ptr IoCard7Pt25->u.io:$D0800C.9.1      // I/O Card 7 I/025
ptr IoCard7Pt26->u.io:$D0800C.10.1     // I/O Card 7 I/026
ptr IoCard7Pt27->u.io:$D0800C.11.1     // I/O Card 7 I/027
ptr IoCard7Pt28->u.io:$D0800C.12.1     // I/O Card 7 I/028
ptr IoCard7Pt29->u.io:$D0800C.13.1     // I/O Card 7 I/029
ptr IoCard7Pt30->u.io:$D0800C.14.1     // I/O Card 7 I/030
ptr IoCard7Pt31->u.io:$D0800C.15.1     // I/O Card 7 I/031
ptr IoCard7Pt32->u.io:$D08010.8.1      // I/O Card 7 I/032
ptr IoCard7Pt33->u.io:$D08010.9.1      // I/O Card 7 I/033
ptr IoCard7Pt34->u.io:$D08010.10.1     // I/O Card 7 I/034
ptr IoCard7Pt35->u.io:$D08010.11.1     // I/O Card 7 I/035
ptr IoCard7Pt36->u.io:$D08010.12.1     // I/O Card 7 I/036
ptr IoCard7Pt37->u.io:$D08010.13.1     // I/O Card 7 I/037
ptr IoCard7Pt38->u.io:$D08010.14.1     // I/O Card 7 I/038
ptr IoCard7Pt39->u.io:$D08010.15.1     // I/O Card 7 I/039
ptr IoCard7Pt40->u.io:$D08014.8.1      // I/O Card 7 I/040
ptr IoCard7Pt41->u.io:$D08014.9.1      // I/O Card 7 I/041
ptr IoCard7Pt42->u.io:$D08014.10.1     // I/O Card 7 I/042
ptr IoCard7Pt43->u.io:$D08014.11.1     // I/O Card 7 I/043
ptr IoCard7Pt44->u.io:$D08014.12.1     // I/O Card 7 I/044
ptr IoCard7Pt45->u.io:$D08014.13.1     // I/O Card 7 I/045
ptr IoCard7Pt46->u.io:$D08014.14.1     // I/O Card 7 I/046
ptr IoCard7Pt47->u.io:$D08014.15.1     // I/O Card 7 I/047

// Byte-wide variables used for power-on configuration
ptr IoCard7Reg0->u.io:$D08000.8.8      // I/O Card 7 I/000-07 as byte
ptr IoCard7Reg1->u.io:$D08004.8.8      // I/O Card 7 I/008-15 as byte
ptr IoCard7Reg2->u.io:$D08008.8.8      // I/O Card 7 I/016-23 as byte
ptr IoCard7Reg3->u.io:$D0800C.8.8      // I/O Card 7 I/024-31 as byte
ptr IoCard7Reg4->u.io:$D08010.8.8      // I/O Card 7 I/032-39 as byte
ptr IoCard7Reg5->u.io:$D08014.8.8      // I/O Card 7 I/040-47 as byte
ptr IoCard7Reg6->u.io:$D08018.8.8      // I/O Card 7 latch inputs
ptr IoCard7Reg7->u.io:$D0801C.8.8      // I/O Card 7 control register
```


디지털 I/O 카드 8

(SW1-1 ON, SW1-2 ON, SW1-3 ON, SW1-4 OFF)

```
// Single-bit variables used for accessing I/O points
ptr IoCard8Pt00->u.io:$A10000.8.1      // I/O Card 8 I/000
ptr IoCard8Pt01->u.io:$A10000.9.1      // I/O Card 8 I/001
ptr IoCard8Pt02->u.io:$A10000.10.1     // I/O Card 8 I/002
ptr IoCard8Pt03->u.io:$A10000.11.1     // I/O Card 8 I/003
ptr IoCard8Pt04->u.io:$A10000.12.1     // I/O Card 8 I/004
ptr IoCard8Pt05->u.io:$A10000.13.1     // I/O Card 8 I/005
ptr IoCard8Pt06->u.io:$A10000.14.1     // I/O Card 8 I/006
ptr IoCard8Pt07->u.io:$A10000.15.1     // I/O Card 8 I/007
ptr IoCard8Pt08->u.io:$A10004.8.1      // I/O Card 8 I/008
ptr IoCard8Pt09->u.io:$A10004.9.1      // I/O Card 8 I/009
ptr IoCard8Pt10->u.io:$A10004.10.1     // I/O Card 8 I/010
ptr IoCard8Pt11->u.io:$A10004.11.1     // I/O Card 8 I/011
ptr IoCard8Pt12->u.io:$A10004.12.1     // I/O Card 8 I/012
ptr IoCard8Pt13->u.io:$A10004.13.1     // I/O Card 8 I/013
ptr IoCard8Pt14->u.io:$A10004.14.1     // I/O Card 8 I/014
ptr IoCard8Pt15->u.io:$A10004.15.1     // I/O Card 8 I/015
ptr IoCard8Pt16->u.io:$A10008.8.1      // I/O Card 8 I/016
ptr IoCard8Pt17->u.io:$A10008.9.1      // I/O Card 8 I/017
ptr IoCard8Pt18->u.io:$A10008.10.1     // I/O Card 8 I/018
ptr IoCard8Pt19->u.io:$A10008.11.1     // I/O Card 8 I/019
ptr IoCard8Pt20->u.io:$A10008.12.1     // I/O Card 8 I/020
ptr IoCard8Pt21->u.io:$A10008.13.1     // I/O Card 8 I/021
ptr IoCard8Pt22->u.io:$A10008.14.1     // I/O Card 8 I/022
ptr IoCard8Pt23->u.io:$A10008.15.1     // I/O Card 8 I/023
ptr IoCard8Pt24->u.io:$A1000C.8.1      // I/O Card 8 I/024
ptr IoCard8Pt25->u.io:$A1000C.9.1      // I/O Card 8 I/025
ptr IoCard8Pt26->u.io:$A1000C.10.1     // I/O Card 8 I/026
ptr IoCard8Pt27->u.io:$A1000C.11.1     // I/O Card 8 I/027
ptr IoCard8Pt28->u.io:$A1000C.12.1     // I/O Card 8 I/028
ptr IoCard8Pt29->u.io:$A1000C.13.1     // I/O Card 8 I/029
ptr IoCard8Pt30->u.io:$A1000C.14.1     // I/O Card 8 I/030
ptr IoCard8Pt31->u.io:$A1000C.15.1     // I/O Card 8 I/031
ptr IoCard8Pt32->u.io:$A10010.8.1      // I/O Card 8 I/032
ptr IoCard8Pt33->u.io:$A10010.9.1      // I/O Card 8 I/033
ptr IoCard8Pt34->u.io:$A10010.10.1     // I/O Card 8 I/034
ptr IoCard8Pt35->u.io:$A10010.11.1     // I/O Card 8 I/035
ptr IoCard8Pt36->u.io:$A10010.12.1     // I/O Card 8 I/036
ptr IoCard8Pt37->u.io:$A10010.13.1     // I/O Card 8 I/037
ptr IoCard8Pt38->u.io:$A10010.14.1     // I/O Card 8 I/038
ptr IoCard8Pt39->u.io:$A10010.15.1     // I/O Card 8 I/039
ptr IoCard8Pt40->u.io:$A10014.8.1      // I/O Card 8 I/040
ptr IoCard8Pt41->u.io:$A10014.9.1      // I/O Card 8 I/041
ptr IoCard8Pt42->u.io:$A10014.10.1     // I/O Card 8 I/042
ptr IoCard8Pt43->u.io:$A10014.11.1     // I/O Card 8 I/043
ptr IoCard8Pt44->u.io:$A10014.12.1     // I/O Card 8 I/044
ptr IoCard8Pt45->u.io:$A10014.13.1     // I/O Card 8 I/045
ptr IoCard8Pt46->u.io:$A10014.14.1     // I/O Card 8 I/046
ptr IoCard8Pt47->u.io:$A10014.15.1     // I/O Card 8 I/047

// Byte-wide variables used for power-on configuration
ptr IoCard8Reg0->u.io:$A10000.8.8      // I/O Card 8 I/000-07 as byte
ptr IoCard8Reg1->u.io:$A10004.8.8      // I/O Card 8 I/008-15 as byte
ptr IoCard8Reg2->u.io:$A10008.8.8      // I/O Card 8 I/016-23 as byte
ptr IoCard8Reg3->u.io:$A1000C.8.8      // I/O Card 8 I/024-31 as byte
ptr IoCard8Reg4->u.io:$A10010.8.8      // I/O Card 8 I/032-39 as byte
ptr IoCard8Reg5->u.io:$A10014.8.8      // I/O Card 8 I/040-47 as byte
ptr IoCard8Reg6->u.io:$A10018.8.8      // I/O Card 8 latch inputs
ptr IoCard8Reg7->u.io:$A1001C.8.8      // I/O Card 8 control register
```

디지털 I/O 카드 9

(SW1-1 OFF, SW1-2 ON, SW1-3 ON, SW1-4 OFF)

```
// Single-bit variables used for accessing I/O points
ptr IoCard9Pt00->u.io:$B10000.8.1      // I/O Card 9 I/000
ptr IoCard9Pt01->u.io:$B10000.9.1      // I/O Card 9 I/001
ptr IoCard9Pt02->u.io:$B10000.10.1     // I/O Card 9 I/002
ptr IoCard9Pt03->u.io:$B10000.11.1     // I/O Card 9 I/003
ptr IoCard9Pt04->u.io:$B10000.12.1     // I/O Card 9 I/004
ptr IoCard9Pt05->u.io:$B10000.13.1     // I/O Card 9 I/005
ptr IoCard9Pt06->u.io:$B10000.14.1     // I/O Card 9 I/006
ptr IoCard9Pt07->u.io:$B10000.15.1     // I/O Card 9 I/007
ptr IoCard9Pt08->u.io:$B10004.8.1      // I/O Card 9 I/008
ptr IoCard9Pt09->u.io:$B10004.9.1      // I/O Card 9 I/009
ptr IoCard9Pt10->u.io:$B10004.10.1     // I/O Card 9 I/010
ptr IoCard9Pt11->u.io:$B10004.11.1     // I/O Card 9 I/011
ptr IoCard9Pt12->u.io:$B10004.12.1     // I/O Card 9 I/012
ptr IoCard9Pt13->u.io:$B10004.13.1     // I/O Card 9 I/013
ptr IoCard9Pt14->u.io:$B10004.14.1     // I/O Card 9 I/014
ptr IoCard9Pt15->u.io:$B10004.15.1     // I/O Card 9 I/015
ptr IoCard9Pt16->u.io:$B10008.8.1      // I/O Card 9 I/016
ptr IoCard9Pt17->u.io:$B10008.9.1      // I/O Card 9 I/017
ptr IoCard9Pt18->u.io:$B10008.10.1     // I/O Card 9 I/018
ptr IoCard9Pt19->u.io:$B10008.11.1     // I/O Card 9 I/019
ptr IoCard9Pt20->u.io:$B10008.12.1     // I/O Card 9 I/020
ptr IoCard9Pt21->u.io:$B10008.13.1     // I/O Card 9 I/021
ptr IoCard9Pt22->u.io:$B10008.14.1     // I/O Card 9 I/022
ptr IoCard9Pt23->u.io:$B10008.15.1     // I/O Card 9 I/023
ptr IoCard9Pt24->u.io:$B1000C.8.1      // I/O Card 9 I/024
ptr IoCard9Pt25->u.io:$B1000C.9.1      // I/O Card 9 I/025
ptr IoCard9Pt26->u.io:$B1000C.10.1     // I/O Card 9 I/026
ptr IoCard9Pt27->u.io:$B1000C.11.1     // I/O Card 9 I/027
ptr IoCard9Pt28->u.io:$B1000C.12.1     // I/O Card 9 I/028
ptr IoCard9Pt29->u.io:$B1000C.13.1     // I/O Card 9 I/029
ptr IoCard9Pt30->u.io:$B1000C.14.1     // I/O Card 9 I/030
ptr IoCard9Pt31->u.io:$B1000C.15.1     // I/O Card 9 I/031
ptr IoCard9Pt32->u.io:$B10010.8.1      // I/O Card 9 I/032
ptr IoCard9Pt33->u.io:$B10010.9.1      // I/O Card 9 I/033
ptr IoCard9Pt34->u.io:$B10010.10.1     // I/O Card 9 I/034
ptr IoCard9Pt35->u.io:$B10010.11.1     // I/O Card 9 I/035
ptr IoCard9Pt36->u.io:$B10010.12.1     // I/O Card 9 I/036
ptr IoCard9Pt37->u.io:$B10010.13.1     // I/O Card 9 I/037
ptr IoCard9Pt38->u.io:$B10010.14.1     // I/O Card 9 I/038
ptr IoCard9Pt39->u.io:$B10010.15.1     // I/O Card 9 I/039
ptr IoCard9Pt40->u.io:$B10014.8.1      // I/O Card 9 I/040
ptr IoCard9Pt41->u.io:$B10014.9.1      // I/O Card 9 I/041
ptr IoCard9Pt42->u.io:$B10014.10.1     // I/O Card 9 I/042
ptr IoCard9Pt43->u.io:$B10014.11.1     // I/O Card 9 I/043
ptr IoCard9Pt44->u.io:$B10014.12.1     // I/O Card 9 I/044
ptr IoCard9Pt45->u.io:$B10014.13.1     // I/O Card 9 I/045
ptr IoCard9Pt46->u.io:$B10014.14.1     // I/O Card 9 I/046
ptr IoCard9Pt47->u.io:$B10014.15.1     // I/O Card 9 I/047

// Byte-wide variables used for power-on configuration
ptr IoCard9Reg0->u.io:$B10000.8.8      // I/O Card 9 I/000-07 as byte
ptr IoCard9Reg1->u.io:$B10004.8.8      // I/O Card 9 I/008-15 as byte
ptr IoCard9Reg2->u.io:$B10008.8.8      // I/O Card 9 I/016-23 as byte
ptr IoCard9Reg3->u.io:$B1000C.8.8      // I/O Card 9 I/024-31 as byte
ptr IoCard9Reg4->u.io:$B10010.8.8      // I/O Card 9 I/032-39 as byte
ptr IoCard9Reg5->u.io:$B10014.8.8      // I/O Card 9 I/040-47 as byte
ptr IoCard9Reg6->u.io:$B10018.8.8      // I/O Card 9 latch inputs
ptr IoCard9Reg7->u.io:$B1001C.8.8      // I/O Card 9 control register
```

디지털 I/O 카드 10

(SW1-1 ON, SW1-2 OFF, SW1-3 ON, SW1-4 OFF)

```
// Single-bit variables used for accessing I/O points
ptr IoCard10Pt00->u.io:$C10000.8.1 // I/O Card 10 I/000
ptr IoCard10Pt01->u.io:$C10000.9.1 // I/O Card 10 I/001
ptr IoCard10Pt02->u.io:$C10000.10.1 // I/O Card 10 I/002
ptr IoCard10Pt03->u.io:$C10000.11.1 // I/O Card 10 I/003
ptr IoCard10Pt04->u.io:$C10000.12.1 // I/O Card 10 I/004
ptr IoCard10Pt05->u.io:$C10000.13.1 // I/O Card 10 I/005
ptr IoCard10Pt06->u.io:$C10000.14.1 // I/O Card 10 I/006
ptr IoCard10Pt07->u.io:$C10000.15.1 // I/O Card 10 I/007
ptr IoCard10Pt08->u.io:$C10004.8.1 // I/O Card 10 I/008
ptr IoCard10Pt09->u.io:$C10004.9.1 // I/O Card 10 I/009
ptr IoCard10Pt10->u.io:$C10004.10.1 // I/O Card 10 I/010
ptr IoCard10Pt11->u.io:$C10004.11.1 // I/O Card 10 I/011
ptr IoCard10Pt12->u.io:$C10004.12.1 // I/O Card 10 I/012
ptr IoCard10Pt13->u.io:$C10004.13.1 // I/O Card 10 I/013
ptr IoCard10Pt14->u.io:$C10004.14.1 // I/O Card 10 I/014
ptr IoCard10Pt15->u.io:$C10004.15.1 // I/O Card 10 I/015
ptr IoCard10Pt16->u.io:$C10008.8.1 // I/O Card 10 I/016
ptr IoCard10Pt17->u.io:$C10008.9.1 // I/O Card 10 I/017
ptr IoCard10Pt18->u.io:$C10008.10.1 // I/O Card 10 I/018
ptr IoCard10Pt19->u.io:$C10008.11.1 // I/O Card 10 I/019
ptr IoCard10Pt20->u.io:$C10008.12.1 // I/O Card 10 I/020
ptr IoCard10Pt21->u.io:$C10008.13.1 // I/O Card 10 I/021
ptr IoCard10Pt22->u.io:$C10008.14.1 // I/O Card 10 I/022
ptr IoCard10Pt23->u.io:$C10008.15.1 // I/O Card 10 I/023
ptr IoCard10Pt24->u.io:$C1000C.8.1 // I/O Card 10 I/024
ptr IoCard10Pt25->u.io:$C1000C.9.1 // I/O Card 10 I/025
ptr IoCard10Pt26->u.io:$C1000C.10.1 // I/O Card 10 I/026
ptr IoCard10Pt27->u.io:$C1000C.11.1 // I/O Card 10 I/027
ptr IoCard10Pt28->u.io:$C1000C.12.1 // I/O Card 10 I/028
ptr IoCard10Pt29->u.io:$C1000C.13.1 // I/O Card 10 I/029
ptr IoCard10Pt30->u.io:$C1000C.14.1 // I/O Card 10 I/030
ptr IoCard10Pt31->u.io:$C1000C.15.1 // I/O Card 10 I/031
ptr IoCard10Pt32->u.io:$C10010.8.1 // I/O Card 10 I/032
ptr IoCard10Pt33->u.io:$C10010.9.1 // I/O Card 10 I/033
ptr IoCard10Pt34->u.io:$C10010.10.1 // I/O Card 10 I/034
ptr IoCard10Pt35->u.io:$C10010.11.1 // I/O Card 10 I/035
ptr IoCard10Pt36->u.io:$C10010.12.1 // I/O Card 10 I/036
ptr IoCard10Pt37->u.io:$C10010.13.1 // I/O Card 10 I/037
ptr IoCard10Pt38->u.io:$C10010.14.1 // I/O Card 10 I/038
ptr IoCard10Pt39->u.io:$C10010.15.1 // I/O Card 10 I/039
ptr IoCard10Pt40->u.io:$C10014.8.1 // I/O Card 10 I/040
ptr IoCard10Pt41->u.io:$C10014.9.1 // I/O Card 10 I/041
ptr IoCard10Pt42->u.io:$C10014.10.1 // I/O Card 10 I/042
ptr IoCard10Pt43->u.io:$C10014.11.1 // I/O Card 10 I/043
ptr IoCard10Pt44->u.io:$C10014.12.1 // I/O Card 10 I/044
ptr IoCard10Pt45->u.io:$C10014.13.1 // I/O Card 10 I/045
ptr IoCard10Pt46->u.io:$C10014.14.1 // I/O Card 10 I/046
ptr IoCard10Pt47->u.io:$C10014.15.1 // I/O Card 10 I/047
// Byte-wide variables used for power-on configuration
ptr IoCard10Reg0->u.io:$C10000.8.8 // I/O Card 10 I/000-07 as byte
ptr IoCard10Reg1->u.io:$C10004.8.8 // I/O Card 10 I/008-15 as byte
ptr IoCard10Reg2->u.io:$C10008.8.8 // I/O Card 10 I/016-23 as byte
ptr IoCard10Reg3->u.io:$C1000C.8.8 // I/O Card 10 I/024-31 as byte
ptr IoCard10Reg4->u.io:$C10010.8.8 // I/O Card 10 I/032-39 as byte
ptr IoCard10Reg5->u.io:$C10014.8.8 // I/O Card 10 I/040-47 as byte
ptr IoCard10Reg6->u.io:$C10018.8.8 // I/O Card 10 latch inputs
ptr IoCard10Reg7->u.io:$C1001C.8.8 // I/O Card 10 control register
```

디지털 I/O 카드 11

(SW1-1 OFF, SW1-2 OFF, SW1-3 ON, SW1-4 OFF)

```
// Single-bit variables used for accessing I/O points
ptr IoCard11Pt00->u.io:$D10000.8.1      // I/O Card 11 I/000
ptr IoCard11Pt01->u.io:$D10000.9.1      // I/O Card 11 I/001
ptr IoCard11Pt02->u.io:$D10000.10.1     // I/O Card 11 I/002
ptr IoCard11Pt03->u.io:$D10000.11.1     // I/O Card 11 I/003
ptr IoCard11Pt04->u.io:$D10000.12.1     // I/O Card 11 I/004
ptr IoCard11Pt05->u.io:$D10000.13.1     // I/O Card 11 I/005
ptr IoCard11Pt06->u.io:$D10000.14.1     // I/O Card 11 I/006
ptr IoCard11Pt07->u.io:$D10000.15.1     // I/O Card 11 I/007
ptr IoCard11Pt08->u.io:$D10004.8.1      // I/O Card 11 I/008
ptr IoCard11Pt09->u.io:$D10004.9.1      // I/O Card 11 I/009
ptr IoCard11Pt10->u.io:$D10004.10.1     // I/O Card 11 I/010
ptr IoCard11Pt11->u.io:$D10004.11.1     // I/O Card 11 I/011
ptr IoCard11Pt12->u.io:$D10004.12.1     // I/O Card 11 I/012
ptr IoCard11Pt13->u.io:$D10004.13.1     // I/O Card 11 I/013
ptr IoCard11Pt14->u.io:$D10004.14.1     // I/O Card 11 I/014
ptr IoCard11Pt15->u.io:$D10004.15.1     // I/O Card 11 I/015
ptr IoCard11Pt16->u.io:$D10008.8.1      // I/O Card 11 I/016
ptr IoCard11Pt17->u.io:$D10008.9.1      // I/O Card 11 I/017
ptr IoCard11Pt18->u.io:$D10008.10.1     // I/O Card 11 I/018
ptr IoCard11Pt19->u.io:$D10008.11.1     // I/O Card 11 I/019
ptr IoCard11Pt20->u.io:$D10008.12.1     // I/O Card 11 I/020
ptr IoCard11Pt21->u.io:$D10008.13.1     // I/O Card 11 I/021
ptr IoCard11Pt22->u.io:$D10008.14.1     // I/O Card 11 I/022
ptr IoCard11Pt23->u.io:$D10008.15.1     // I/O Card 11 I/023
ptr IoCard11Pt24->u.io:$D1000C.8.1      // I/O Card 11 I/024
ptr IoCard11Pt25->u.io:$D1000C.9.1      // I/O Card 11 I/025
ptr IoCard11Pt26->u.io:$D1000C.10.1     // I/O Card 11 I/026
ptr IoCard11Pt27->u.io:$D1000C.11.1     // I/O Card 11 I/027
ptr IoCard11Pt28->u.io:$D1000C.12.1     // I/O Card 11 I/028
ptr IoCard11Pt29->u.io:$D1000C.13.1     // I/O Card 11 I/029
ptr IoCard11Pt30->u.io:$D1000C.14.1     // I/O Card 11 I/030
ptr IoCard11Pt31->u.io:$D1000C.15.1     // I/O Card 11 I/031
ptr IoCard11Pt32->u.io:$D10010.8.1      // I/O Card 11 I/032
ptr IoCard11Pt33->u.io:$D10010.9.1      // I/O Card 11 I/033
ptr IoCard11Pt34->u.io:$D10010.10.1     // I/O Card 11 I/034
ptr IoCard11Pt35->u.io:$D10010.11.1     // I/O Card 11 I/035
ptr IoCard11Pt36->u.io:$D10010.12.1     // I/O Card 11 I/036
ptr IoCard11Pt37->u.io:$D10010.13.1     // I/O Card 11 I/037
ptr IoCard11Pt38->u.io:$D10010.14.1     // I/O Card 11 I/038
ptr IoCard11Pt39->u.io:$D10010.15.1     // I/O Card 11 I/039
ptr IoCard11Pt40->u.io:$D10014.8.1      // I/O Card 11 I/040
ptr IoCard11Pt41->u.io:$D10014.9.1      // I/O Card 11 I/041
ptr IoCard11Pt42->u.io:$D10014.10.1     // I/O Card 11 I/042
ptr IoCard11Pt43->u.io:$D10014.11.1     // I/O Card 11 I/043
ptr IoCard11Pt44->u.io:$D10014.12.1     // I/O Card 11 I/044
ptr IoCard11Pt45->u.io:$D10014.13.1     // I/O Card 11 I/045
ptr IoCard11Pt46->u.io:$D10014.14.1     // I/O Card 11 I/046
ptr IoCard11Pt47->u.io:$D10014.15.1     // I/O Card 11 I/047

// Byte-wide variables used for power-on configuration
ptr IoCard11Reg0->u.io:$D10000.8.8      // I/O Card 11 I/000-07 as byte
ptr IoCard11Reg1->u.io:$D10004.8.8      // I/O Card 11 I/008-15 as byte
ptr IoCard11Reg2->u.io:$D10008.8.8      // I/O Card 11 I/016-23 as byte
ptr IoCard11Reg3->u.io:$D1000C.8.8      // I/O Card 11 I/024-31 as byte
ptr IoCard11Reg4->u.io:$D10010.8.8      // I/O Card 11 I/032-39 as byte
ptr IoCard11Reg5->u.io:$D10014.8.8      // I/O Card 11 I/040-47 as byte
ptr IoCard11Reg6->u.io:$D10018.8.8      // I/O Card 11 latch inputs
ptr IoCard11Reg7->u.io:$D1001C.8.8      // I/O Card 11 control register
```

디지털 I/O 카드 12

(SW1-1 ON, SW1-2 ON, SW1-3 OFF, SW1-4 OFF)

```
// Single-bit variables used for accessing I/O points
ptr IoCard12Pt00->u.io:$A18000.8.1 // I/O Card 12 I/000
ptr IoCard12Pt01->u.io:$A18000.9.1 // I/O Card 12 I/001
ptr IoCard12Pt02->u.io:$A18000.10.1 // I/O Card 12 I/002
ptr IoCard12Pt03->u.io:$A18000.11.1 // I/O Card 12 I/003
ptr IoCard12Pt04->u.io:$A18000.12.1 // I/O Card 12 I/004
ptr IoCard12Pt05->u.io:$A18000.13.1 // I/O Card 12 I/005
ptr IoCard12Pt06->u.io:$A18000.14.1 // I/O Card 12 I/006
ptr IoCard12Pt07->u.io:$A18000.15.1 // I/O Card 12 I/007
ptr IoCard12Pt08->u.io:$A18004.8.1 // I/O Card 12 I/008
ptr IoCard12Pt09->u.io:$A18004.9.1 // I/O Card 12 I/009
ptr IoCard12Pt10->u.io:$A18004.10.1 // I/O Card 12 I/010
ptr IoCard12Pt11->u.io:$A18004.11.1 // I/O Card 12 I/011
ptr IoCard12Pt12->u.io:$A18004.12.1 // I/O Card 12 I/012
ptr IoCard12Pt13->u.io:$A18004.13.1 // I/O Card 12 I/013
ptr IoCard12Pt14->u.io:$A18004.14.1 // I/O Card 12 I/014
ptr IoCard12Pt15->u.io:$A18004.15.1 // I/O Card 12 I/015
ptr IoCard12Pt16->u.io:$A18008.8.1 // I/O Card 12 I/016
ptr IoCard12Pt17->u.io:$A18008.9.1 // I/O Card 12 I/017
ptr IoCard12Pt18->u.io:$A18008.10.1 // I/O Card 12 I/018
ptr IoCard12Pt19->u.io:$A18008.11.1 // I/O Card 12 I/019
ptr IoCard12Pt20->u.io:$A18008.12.1 // I/O Card 12 I/020
ptr IoCard12Pt21->u.io:$A18008.13.1 // I/O Card 12 I/021
ptr IoCard12Pt22->u.io:$A18008.14.1 // I/O Card 12 I/022
ptr IoCard12Pt23->u.io:$A18008.15.1 // I/O Card 12 I/023
ptr IoCard12Pt24->u.io:$A1800C.8.1 // I/O Card 12 I/024
ptr IoCard12Pt25->u.io:$A1800C.9.1 // I/O Card 12 I/025
ptr IoCard12Pt26->u.io:$A1800C.10.1 // I/O Card 12 I/026
ptr IoCard12Pt27->u.io:$A1800C.11.1 // I/O Card 12 I/027
ptr IoCard12Pt28->u.io:$A1800C.12.1 // I/O Card 12 I/028
ptr IoCard12Pt29->u.io:$A1800C.13.1 // I/O Card 12 I/029
ptr IoCard12Pt30->u.io:$A1800C.14.1 // I/O Card 12 I/030
ptr IoCard12Pt31->u.io:$A1800C.15.1 // I/O Card 12 I/031
ptr IoCard12Pt32->u.io:$A18010.8.1 // I/O Card 12 I/032
ptr IoCard12Pt33->u.io:$A18010.9.1 // I/O Card 12 I/033
ptr IoCard12Pt34->u.io:$A18010.10.1 // I/O Card 12 I/034
ptr IoCard12Pt35->u.io:$A18010.11.1 // I/O Card 12 I/035
ptr IoCard12Pt36->u.io:$A18010.12.1 // I/O Card 12 I/036
ptr IoCard12Pt37->u.io:$A18010.13.1 // I/O Card 12 I/037
ptr IoCard12Pt38->u.io:$A18010.14.1 // I/O Card 12 I/038
ptr IoCard12Pt39->u.io:$A18010.15.1 // I/O Card 12 I/039
ptr IoCard12Pt40->u.io:$A18014.8.1 // I/O Card 12 I/040
ptr IoCard12Pt41->u.io:$A18014.9.1 // I/O Card 12 I/041
ptr IoCard12Pt42->u.io:$A18014.10.1 // I/O Card 12 I/042
ptr IoCard12Pt43->u.io:$A18014.11.1 // I/O Card 12 I/043
ptr IoCard12Pt44->u.io:$A18014.12.1 // I/O Card 12 I/044
ptr IoCard12Pt45->u.io:$A18014.13.1 // I/O Card 12 I/045
ptr IoCard12Pt46->u.io:$A18014.14.1 // I/O Card 12 I/046
ptr IoCard12Pt47->u.io:$A18014.15.1 // I/O Card 12 I/047
// Byte-wide variables used for power-on configuration
ptr IoCard12Reg0->u.io:$A18000.8.8 // I/O Card 12 I/000-07 as byte
ptr IoCard12Reg1->u.io:$A18004.8.8 // I/O Card 12 I/008-15 as byte
ptr IoCard12Reg2->u.io:$A18008.8.8 // I/O Card 12 I/016-23 as byte
ptr IoCard12Reg3->u.io:$A1800C.8.8 // I/O Card 12 I/024-31 as byte
ptr IoCard12Reg4->u.io:$A18010.8.8 // I/O Card 12 I/032-39 as byte
ptr IoCard12Reg5->u.io:$A18014.8.8 // I/O Card 12 I/040-47 as byte
ptr IoCard12Reg6->u.io:$A18018.8.8 // I/O Card 12 latch inputs
ptr IoCard12Reg7->u.io:$A1801C.8.8 // I/O Card 12 control register
```


디지털 I/O 카드 13

(SW1-1 OFF, SW1-2 ON, SW1-3 OFF, SW1-4 OFF)

```
// Single-bit variables used for accessing I/O points
ptr IoCard13Pt00->u.io:$B18000.8.1 // I/O Card 13 I/000
ptr IoCard13Pt01->u.io:$B18000.9.1 // I/O Card 13 I/001
ptr IoCard13Pt02->u.io:$B18000.10.1 // I/O Card 13 I/002
ptr IoCard13Pt03->u.io:$B18000.11.1 // I/O Card 13 I/003
ptr IoCard13Pt04->u.io:$B18000.12.1 // I/O Card 13 I/004
ptr IoCard13Pt05->u.io:$B18000.13.1 // I/O Card 13 I/005
ptr IoCard13Pt06->u.io:$B18000.14.1 // I/O Card 13 I/006
ptr IoCard13Pt07->u.io:$B18000.15.1 // I/O Card 13 I/007
ptr IoCard13Pt08->u.io:$B18004.8.1 // I/O Card 13 I/008
ptr IoCard13Pt09->u.io:$B18004.9.1 // I/O Card 13 I/009
ptr IoCard13Pt10->u.io:$B18004.10.1 // I/O Card 13 I/010
ptr IoCard13Pt11->u.io:$B18004.11.1 // I/O Card 13 I/011
ptr IoCard13Pt12->u.io:$B18004.12.1 // I/O Card 13 I/012
ptr IoCard13Pt13->u.io:$B18004.13.1 // I/O Card 13 I/013
ptr IoCard13Pt14->u.io:$B18004.14.1 // I/O Card 13 I/014
ptr IoCard13Pt15->u.io:$B18004.15.1 // I/O Card 13 I/015
ptr IoCard13Pt16->u.io:$B18008.8.1 // I/O Card 13 I/016
ptr IoCard13Pt17->u.io:$B18008.9.1 // I/O Card 13 I/017
ptr IoCard13Pt18->u.io:$B18008.10.1 // I/O Card 13 I/018
ptr IoCard13Pt19->u.io:$B18008.11.1 // I/O Card 13 I/019
ptr IoCard13Pt20->u.io:$B18008.12.1 // I/O Card 13 I/020
ptr IoCard13Pt21->u.io:$B18008.13.1 // I/O Card 13 I/021
ptr IoCard13Pt22->u.io:$B18008.14.1 // I/O Card 13 I/022
ptr IoCard13Pt23->u.io:$B18008.15.1 // I/O Card 13 I/023
ptr IoCard13Pt24->u.io:$B1800C.8.1 // I/O Card 13 I/024
ptr IoCard13Pt25->u.io:$B1800C.9.1 // I/O Card 13 I/025
ptr IoCard13Pt26->u.io:$B1800C.10.1 // I/O Card 13 I/026
ptr IoCard13Pt27->u.io:$B1800C.11.1 // I/O Card 13 I/027
ptr IoCard13Pt28->u.io:$B1800C.12.1 // I/O Card 13 I/028
ptr IoCard13Pt29->u.io:$B1800C.13.1 // I/O Card 13 I/029
ptr IoCard13Pt30->u.io:$B1800C.14.1 // I/O Card 13 I/030
ptr IoCard13Pt31->u.io:$B1800C.15.1 // I/O Card 13 I/031
ptr IoCard13Pt32->u.io:$B18010.8.1 // I/O Card 13 I/032
ptr IoCard13Pt33->u.io:$B18010.9.1 // I/O Card 13 I/033
ptr IoCard13Pt34->u.io:$B18010.10.1 // I/O Card 13 I/034
ptr IoCard13Pt35->u.io:$B18010.11.1 // I/O Card 13 I/035
ptr IoCard13Pt36->u.io:$B18010.12.1 // I/O Card 13 I/036
ptr IoCard13Pt37->u.io:$B18010.13.1 // I/O Card 13 I/037
ptr IoCard13Pt38->u.io:$B18010.14.1 // I/O Card 13 I/038
ptr IoCard13Pt39->u.io:$B18010.15.1 // I/O Card 13 I/039
ptr IoCard13Pt40->u.io:$B18014.8.1 // I/O Card 13 I/040
ptr IoCard13Pt41->u.io:$B18014.9.1 // I/O Card 13 I/041
ptr IoCard13Pt42->u.io:$B18014.10.1 // I/O Card 13 I/042
ptr IoCard13Pt43->u.io:$B18014.11.1 // I/O Card 13 I/043
ptr IoCard13Pt44->u.io:$B18014.12.1 // I/O Card 13 I/044
ptr IoCard13Pt45->u.io:$B18014.13.1 // I/O Card 13 I/045
ptr IoCard13Pt46->u.io:$B18014.14.1 // I/O Card 13 I/046
ptr IoCard13Pt47->u.io:$B18014.15.1 // I/O Card 13 I/047
// Byte-wide variables used for power-on configuration
ptr IoCard13Reg0->u.io:$B18000.8.8 // I/O Card 13 I/000-07 as byte
ptr IoCard13Reg1->u.io:$B18004.8.8 // I/O Card 13 I/008-15 as byte
ptr IoCard13Reg2->u.io:$B18008.8.8 // I/O Card 13 I/016-23 as byte
ptr IoCard13Reg3->u.io:$B1800C.8.8 // I/O Card 13 I/024-31 as byte
ptr IoCard13Reg4->u.io:$B18010.8.8 // I/O Card 13 I/032-39 as byte
ptr IoCard13Reg5->u.io:$B18014.8.8 // I/O Card 13 I/040-47 as byte
ptr IoCard13Reg6->u.io:$B18018.8.8 // I/O Card 13 latch inputs
ptr IoCard13Reg7->u.io:$B1801C.8.8 // I/O Card 13 control register
```

디지털 I/O 카드 14

(SW1-1 ON, SW1-2 OFF, SW1-3 OFF, SW1-4 OFF)

```
// Single-bit variables used for accessing I/O points
ptr IoCard14Pt00->u.io:$C18000.8.1 // I/O Card 14 I/000
ptr IoCard14Pt01->u.io:$C18000.9.1 // I/O Card 14 I/001
ptr IoCard14Pt02->u.io:$C18000.10.1 // I/O Card 14 I/002
ptr IoCard14Pt03->u.io:$C18000.11.1 // I/O Card 14 I/003
ptr IoCard14Pt04->u.io:$C18000.12.1 // I/O Card 14 I/004
ptr IoCard14Pt05->u.io:$C18000.13.1 // I/O Card 14 I/005
ptr IoCard14Pt06->u.io:$C18000.14.1 // I/O Card 14 I/006
ptr IoCard14Pt07->u.io:$C18000.15.1 // I/O Card 14 I/007
ptr IoCard14Pt08->u.io:$C18004.8.1 // I/O Card 14 I/008
ptr IoCard14Pt09->u.io:$C18004.9.1 // I/O Card 14 I/009
ptr IoCard14Pt10->u.io:$C18004.10.1 // I/O Card 14 I/010
ptr IoCard14Pt11->u.io:$C18004.11.1 // I/O Card 14 I/011
ptr IoCard14Pt12->u.io:$C18004.12.1 // I/O Card 14 I/012
ptr IoCard14Pt13->u.io:$C18004.13.1 // I/O Card 14 I/013
ptr IoCard14Pt14->u.io:$C18004.14.1 // I/O Card 14 I/014
ptr IoCard14Pt15->u.io:$C18004.15.1 // I/O Card 14 I/015
ptr IoCard14Pt16->u.io:$C18008.8.1 // I/O Card 14 I/016
ptr IoCard14Pt17->u.io:$C18008.9.1 // I/O Card 14 I/017
ptr IoCard14Pt18->u.io:$C18008.10.1 // I/O Card 14 I/018
ptr IoCard14Pt19->u.io:$C18008.11.1 // I/O Card 14 I/019
ptr IoCard14Pt20->u.io:$C18008.12.1 // I/O Card 14 I/020
ptr IoCard14Pt21->u.io:$C18008.13.1 // I/O Card 14 I/021
ptr IoCard14Pt22->u.io:$C18008.14.1 // I/O Card 14 I/022
ptr IoCard14Pt23->u.io:$C18008.15.1 // I/O Card 14 I/023
ptr IoCard14Pt24->u.io:$C1800C.8.1 // I/O Card 14 I/024
ptr IoCard14Pt25->u.io:$C1800C.9.1 // I/O Card 14 I/025
ptr IoCard14Pt26->u.io:$C1800C.10.1 // I/O Card 14 I/026
ptr IoCard14Pt27->u.io:$C1800C.11.1 // I/O Card 14 I/027
ptr IoCard14Pt28->u.io:$C1800C.12.1 // I/O Card 14 I/028
ptr IoCard14Pt29->u.io:$C1800C.13.1 // I/O Card 14 I/029
ptr IoCard14Pt30->u.io:$C1800C.14.1 // I/O Card 14 I/030
ptr IoCard14Pt31->u.io:$C1800C.15.1 // I/O Card 14 I/031
ptr IoCard14Pt32->u.io:$C18010.8.1 // I/O Card 14 I/032
ptr IoCard14Pt33->u.io:$C18010.9.1 // I/O Card 14 I/033
ptr IoCard14Pt34->u.io:$C18010.10.1 // I/O Card 14 I/034
ptr IoCard14Pt35->u.io:$C18010.11.1 // I/O Card 14 I/035
ptr IoCard14Pt36->u.io:$C18010.12.1 // I/O Card 14 I/036
ptr IoCard14Pt37->u.io:$C18010.13.1 // I/O Card 14 I/037
ptr IoCard14Pt38->u.io:$C18010.14.1 // I/O Card 14 I/038
ptr IoCard14Pt39->u.io:$C18010.15.1 // I/O Card 14 I/039
ptr IoCard14Pt40->u.io:$C18014.8.1 // I/O Card 14 I/040
ptr IoCard14Pt41->u.io:$C18014.9.1 // I/O Card 14 I/041
ptr IoCard14Pt42->u.io:$C18014.10.1 // I/O Card 14 I/042
ptr IoCard14Pt43->u.io:$C18014.11.1 // I/O Card 14 I/043
ptr IoCard14Pt44->u.io:$C18014.12.1 // I/O Card 14 I/044
ptr IoCard14Pt45->u.io:$C18014.13.1 // I/O Card 14 I/045
ptr IoCard14Pt46->u.io:$C18014.14.1 // I/O Card 14 I/046
ptr IoCard14Pt47->u.io:$C18014.15.1 // I/O Card 14 I/047
// Byte-wide variables used for power-on configuration
ptr IoCard14Reg0->u.io:$C18000.8.8 // I/O Card 14 I/000-07 as byte
ptr IoCard14Reg1->u.io:$C18004.8.8 // I/O Card 14 I/008-15 as byte
ptr IoCard14Reg2->u.io:$C18008.8.8 // I/O Card 14 I/016-23 as byte
ptr IoCard14Reg3->u.io:$C1800C.8.8 // I/O Card 14 I/024-31 as byte
ptr IoCard14Reg4->u.io:$C18010.8.8 // I/O Card 14 I/032-39 as byte
ptr IoCard14Reg5->u.io:$C18014.8.8 // I/O Card 14 I/040-47 as byte
ptr IoCard14Reg6->u.io:$C18018.8.8 // I/O Card 14 latch inputs
ptr IoCard14Reg7->u.io:$C1801C.8.8 // I/O Card 14 control register
```

디지털 I/O 카드 15

(SW1-1 OFF, SW1-2 OFF, SW1-3 OFF, SW1-4 OFF)

```
// Single-bit variables used for accessing I/O points
ptr IoCard15Pt00->u.io:$D18000.8.1      // I/O Card 15 I/000
ptr IoCard15Pt01->u.io:$D18000.9.1      // I/O Card 15 I/001
ptr IoCard15Pt02->u.io:$D18000.10.1     // I/O Card 15 I/002
ptr IoCard15Pt03->u.io:$D18000.11.1     // I/O Card 15 I/003
ptr IoCard15Pt04->u.io:$D18000.12.1     // I/O Card 15 I/004
ptr IoCard15Pt05->u.io:$D18000.13.1     // I/O Card 15 I/005
ptr IoCard15Pt06->u.io:$D18000.14.1     // I/O Card 15 I/006
ptr IoCard15Pt07->u.io:$D18000.15.1     // I/O Card 15 I/007
ptr IoCard15Pt08->u.io:$D18004.8.1      // I/O Card 15 I/008
ptr IoCard15Pt09->u.io:$D18004.9.1      // I/O Card 15 I/009
ptr IoCard15Pt10->u.io:$D18004.10.1     // I/O Card 15 I/010
ptr IoCard15Pt11->u.io:$D18004.11.1     // I/O Card 15 I/011
ptr IoCard15Pt12->u.io:$D18004.12.1     // I/O Card 15 I/012
ptr IoCard15Pt13->u.io:$D18004.13.1     // I/O Card 15 I/013
ptr IoCard15Pt14->u.io:$D18004.14.1     // I/O Card 15 I/014
ptr IoCard15Pt15->u.io:$D18004.15.1     // I/O Card 15 I/015
ptr IoCard15Pt16->u.io:$D18008.8.1      // I/O Card 15 I/016
ptr IoCard15Pt17->u.io:$D18008.9.1      // I/O Card 15 I/017
ptr IoCard15Pt18->u.io:$D18008.10.1     // I/O Card 15 I/018
ptr IoCard15Pt19->u.io:$D18008.11.1     // I/O Card 15 I/019
ptr IoCard15Pt20->u.io:$D18008.12.1     // I/O Card 15 I/020
ptr IoCard15Pt21->u.io:$D18008.13.1     // I/O Card 15 I/021
ptr IoCard15Pt22->u.io:$D18008.14.1     // I/O Card 15 I/022
ptr IoCard15Pt23->u.io:$D18008.15.1     // I/O Card 15 I/023
ptr IoCard15Pt24->u.io:$D1800C.8.1      // I/O Card 15 I/024
ptr IoCard15Pt25->u.io:$D1800C.9.1      // I/O Card 15 I/025
ptr IoCard15Pt26->u.io:$D1800C.10.1     // I/O Card 15 I/026
ptr IoCard15Pt27->u.io:$D1800C.11.1     // I/O Card 15 I/027
ptr IoCard15Pt28->u.io:$D1800C.12.1     // I/O Card 15 I/028
ptr IoCard15Pt29->u.io:$D1800C.13.1     // I/O Card 15 I/029
ptr IoCard15Pt30->u.io:$D1800C.14.1     // I/O Card 15 I/030
ptr IoCard15Pt31->u.io:$D1800C.15.1     // I/O Card 15 I/031
ptr IoCard15Pt32->u.io:$D18010.8.1      // I/O Card 15 I/032
ptr IoCard15Pt33->u.io:$D18010.9.1      // I/O Card 15 I/033
ptr IoCard15Pt34->u.io:$D18010.10.1     // I/O Card 15 I/034
ptr IoCard15Pt35->u.io:$D18010.11.1     // I/O Card 15 I/035
ptr IoCard15Pt36->u.io:$D18010.12.1     // I/O Card 15 I/036
ptr IoCard15Pt37->u.io:$D18010.13.1     // I/O Card 15 I/037
ptr IoCard15Pt38->u.io:$D18010.14.1     // I/O Card 15 I/038
ptr IoCard15Pt39->u.io:$D18010.15.1     // I/O Card 15 I/039
ptr IoCard15Pt40->u.io:$D18014.8.1      // I/O Card 15 I/040
ptr IoCard15Pt41->u.io:$D18014.9.1      // I/O Card 15 I/041
ptr IoCard15Pt42->u.io:$D18014.10.1     // I/O Card 15 I/042
ptr IoCard15Pt43->u.io:$D18014.11.1     // I/O Card 15 I/043
ptr IoCard15Pt44->u.io:$D18014.12.1     // I/O Card 15 I/044
ptr IoCard15Pt45->u.io:$D18014.13.1     // I/O Card 15 I/045
ptr IoCard15Pt46->u.io:$D18014.14.1     // I/O Card 15 I/046
ptr IoCard15Pt47->u.io:$D18014.15.1     // I/O Card 15 I/047

// Byte-wide variables used for power-on configuration
ptr IoCard15Reg0->u.io:$D18000.8.8      // I/O Card 15 I/000-07 as byte
ptr IoCard15Reg1->u.io:$D18004.8.8      // I/O Card 15 I/008-15 as byte
ptr IoCard15Reg2->u.io:$D18008.8.8      // I/O Card 15 I/016-23 as byte
ptr IoCard15Reg3->u.io:$D1800C.8.8      // I/O Card 15 I/024-31 as byte
ptr IoCard15Reg4->u.io:$D18010.8.8      // I/O Card 15 I/032-39 as byte
ptr IoCard15Reg5->u.io:$D18014.8.8      // I/O Card 15 I/040-47 as byte
ptr IoCard15Reg6->u.io:$D18018.8.8      // I/O Card 15 latch inputs
ptr IoCard15Reg7->u.io:$D1801C.8.8      // I/O Card 15 control register
```


POWER PMAC TURBO 'I' 변수와 동등한 기능

이 섹션에서는 최소한의 변경으로 운전 중인 Turbo PMAC 어플리케이션에서 Power PMAC으로 변환하는 데 도움이 되는 정보를 제공합니다. Turbo PMAC의 설정 'I' 변수와 동등한 Power PMAC의 기능에 대해 설명하고 이러한 설정에 변경이 필요한지 여부를 기재합니다. 이러한 테이블을 직접 적용한 직접적인 변환을 실시하면 Power PMAC의 모든 기능을 활용할 수 없는 경우가 많습니다. 단, 향후 확장하는 경우에는 이 방법을 사용해도 출발점으로는 충분합니다.

동등한 기능 관련 주의점

비고란에는 다음 항목 중 해당 란에 기재된 번호와 동일한 번호의 항목이 해당합니다.

1. Power PMAC 변수의 주소는 보통 '{데이터 구조체 요소명}.a'로 지정합니다. 주소의 수치 값은 필요하지 않지만 Turbo PMAC의 동등한 기능과는 다릅니다.
2. Power PMAC에서는 이 변수의 위치 단위는 모터 단위이며 이를 엔코더의 카운터로 사용하는 것은 가능하지만 필수는 아닙니다. Turbo PMAC에서 가장 간단하게 사용 가능한 포트 설정을 실시하는 경우에 모터 단위를 엔코더 또는 동등 기기의 카운트로 설정합니다.
3. 이 변수는 Turbo PMAC에서는 정수(또는 적어도 고정 소수점 값)이지만 Power PMAC에서는 부동 소수점이므로 분해능이 향상될 뿐만 아니라 범위도 넓어지는 경우가 있습니다.
4. 고정 소수점을 사용하는 모터의 수식과 비교하면 Power PMAC의 부동 소수점 모터의 수식에서는 모터 위치의 스케일 계수의 동작이 완전히 다릅니다. Power PMAC의 스케일 계수를 변경하면 실제로 모터의 작업 단위가 변경됩니다(이에 따라 모터를 밀리미터나 각도 등의 공학 단위로 스케일링할 수 있습니다). 단, 가장 간단하게 Turbo PMAC 어플리케이션을 변환하려면 모터의 단위를 피드백 장치의 카운터 단위 상태로 두는 것을 권장합니다. 그러면 이러한 어플리케이션에서는 [Motor[x].PosSf] 및 [Motor[x].Pos2Sf]를 기본값인 1.0으로 설정합니다.
5. Turbo PMAC의 'I' 변수의 수치는 Power PMAC에서 이와 일치하는 설정 요소의 레거시와 동등한 기능으로 사용할 수 있습니다(모터 1~32, 좌표계 1~16, MACRO IC 0~3 및 Servo IC 0~9).

글로벌 'I' 변수

이 섹션에서는 Turbo PMAC의 글로벌 'I' 변수 및 Power PMAC에서 이와 동등한 기능을 목록으로 기재합니다.

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	Power PMAC 탑재 코멘트	비고
I0	--	Power PMAC에서는 시리얼 포트의 호스트 통신을 지원하지 않음	
I1	--	Power PMAC에서는 시리얼 포트의 호스트 통신을 지원하지 않음	
I2	--	Turbo PMAC2 또는 Power PMAC에는 컨트롤 패널의 포트가 없음	
I3	--	Power PMAC에서 핸드셰이킹 문자는 고정되어 있음	
I4	--	TCP/IP 레벨에서 통신의 정합성 처리	
I5	--	Power PMAC에서는 항상 I5 = 3과 동등(모든 PLC를 유효화할 수 있음) 파일 pp_startup.txt의 명령을 사용해서 기동 시에 유효화하는 PLC 프로그램을 제어	
I6	--	Power PMAC에서는 에러 통지 프로토콜은 고정되어 있음	
I7	Sys.PhaseCycleExt	동일한 값으로 호환 운용 제공	
I8	Sys.RtIntPeriod	동일한 값으로 호환 운용 제공	
I9 비트 0	echo 명령값 비트 0~2	echo 값에 따라 쿼리의 종류에 의해 short 형식, long 형식의 응답 옵션을 개별적으로 설정 가능	
I9 비트 1	echo 값 비트 3	echo 비트의 극성은 I9 비트와 반대. = 0은 16진법 응답, = 1은 10진법 응답	
I10	Sys.ServoPeriod	ServoPeriod = I10/8388608에 설정하면 호환 운용 가능	
I11	--	Power PMAC에서는 명령 실행 후 항상 가능한 한 즉시 모션 프로그램의 실행을 시작함	
I12	--	확장된 Power PMAC의 록 어헤드 알고리즘에 따라 이 변수의 필요성이 없어짐	
I13	--	Power PMAC의 포어그라운드에서 모든 인포지션 확인이 실행됨	
I14	--	pp_startup.txt 파일의 명령으로 버퍼정의 가능	
I15	--	'd'로 끝나는 삼각함수(sind, atan2d 등)는 각도를 사용하고 그 이외(sin, atan2)는 라디안을 사용함	

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	Power PMAC 탑재 코멘트	비고
I16~I18	--	프로그램 버퍼의 핸드셰이킹은 OS 유틸리티에 따라 자동으로 처리됨	
I19	--	각 저장 요소 [Gaten[I].PhaseServoDir]에 따라 ASI 클록 소스가 지정됨	
I20~I23	--	MACRO IC 번호는 주소에 기반하여 자동으로 할당됨 Power PMAC에서는 4 개의 IC 에 한정되지 않음	
I24	--	Power PMAC에는 DPRAM 이 없음	
I26	--	Power PMAC 에 따라 자동으로 지원되는 것은 화이버 MACRO 만 해당	
I27	Muxlo.InBit	InBit = 8, I27 = 0; InBit = 15, I27 = 1	
I28	--	Power PMAC 에서는 디스플레이 포트가 지원되지 않음	
I29	Muxlo.pln Muxlo.pOut	Power PMAC 에는 기본값 주소가 없음	1
I30	CompTable[m].Ctrl	각 테이블의 차원마다 롤 오버를 개별적으로 제어 가능	
I36	--	Power PMAC 에서는 Enable 및 Disable 은 항상 분리됨(I36 = 1 와 동일함)	
I37	Sys.BusCtrl[n]	메모리 대기 제어 없음 I/O 대기와는 크게 다름	
I38	--	향상된 Power PMAC 의 표준 기능에 따라 필요성이 없어짐	
I39	--	Power PMAC 에서는 구성 정보의 표시가 다름	
I40	Sys.WDTReset	기본값 설정인 0 에 따라 실질적으로 호환 운용 가능	
I41	--	Power PMAC 에서는 로그아웃 기능은 지원되지 않음	
I42	--	향상된 Power PMAC 의 표준 기능에 따라 필요성이 없어짐	
I43	--	Power PMAC 에서는 시리얼 포트의 호스트 통신을 지원하지 않음	
I44	--	Power PMAC 에서는 래더 로직의 프로그램이 독립적인 어플리케이션처럼 실행됨	
I45	--	향상된 Power PMAC 의 표준 기능에 따라 필요성이 없어짐	
I46	--	Power PMAC 에는 배터리 백업 탑재 RAM 이 없음	
I47~I50	--	Power PMAC 에는 DPRAM 이 없음	

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	Power PMAC 탑재 코멘트	비고
I51	Sys.CompEnable	CompEnable 을 최대 테이블 수보다 크게 설정하면 모든 테이블이 유효가 됨	
I52	--	CPU 의 동작 주파수는 자동으로 설정됨	
I53~I54	--	시리얼 포트의 전송 속도는 OS 파일 etc/inittab 에서 기본값인 115,200 으로 설정	
I55~I58		Power PMAC 에는 DPRAM 이 없음	
I59	--	Turbo PMAC2 또는 Power PMAC 에는 컨트롤 패널이 없음	
I60~I61	--	Power PMAC 에서는 필터링된 속도의 샘플 주기가 16 서보 사이클로 고정됨	
I62	Sys.SendFileMode	Power PMAC 에서는 동작이 약간 다름	
I63~I64	--	향상된 Power PMAC 의 표준 기능에 따라 필요성이 없어짐	
I65	--	Power PMAC 의 프로젝트 관리 파일에 따라 사용자 구성을 표시하는 것 외의 방법이 제공됨	
I67	Modbus[I] 구조체	Power PMAC 에서는 데이터 구조체 주소를 지정할 필요가 없음	
I68	Sys.MaxCoords Coord[x].SyncOps	Power PMAC 의 기본값 설정에 따라 모든 Turbo PMAC 설정과 호환 운용이 가능	
I69	--	Power PMAC 에는 Modbus 의 자동 제어 패널 구조가 없음	
I70, I72, I74, I76	--	향상된 Power PMAC 의 표준 기능에 따라 필요성이 없어짐	
I71, I73, I75, I77	--	Power PMAC 에서는 타입 1 의 MACRO 프로토콜만 지원함	
I78	Macro.IOTimeout	I78 은 서보 사이클, IOTimeout 은 밀리초	
I79	Macro.IOTimeout	I79 는 서보 사이클, IOTimeout 은 밀리초	
I80	Macro.TestPeriod	동일한 값으로 호환 운용 제공	
I81	Macro.TestMaxErrors	동일한 값으로 호환 운용 제공	
I82	Macro.TestReqdSynchs	동일한 값으로 호환 운용 제공	
I83	--	Power PMAC 에서는 자동으로 설정	
I84	--	Power PMAC 에서는 자동으로 설정	
I85	Macro.Station	자동으로 검출됨	
I90~I99	--	Power PMAC 에는 VME 버스 인터페이스가 없음	

모터 'I' 변수

이 섹션에서는 Turbo PMAC의 모터 고유의 'I' 변수 및 Power PMAC에서 이와 동등한 기능을 목록으로 기재합니다. Power PMAC의 모터 구조체 인덱스 'x'는 Turbo PMAC의 'I' 변수 번호의 백의 자리의 값과 동등합니다([Motor[17].ServoCtrl]은 'I1700'와 동등함).

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	Power PMAC 탑재 코멘트	비고
Ixx00	Motor[x].ServoCtrl	동일한 값으로 호환 운용 제공	5
Ixx01	Motor[x].PhaseCtrl	Power PMAC에서 PMAC2 스타일 IC와 통신하는 방법은 4로 설정(1이 아님)	5
Ixx02	Motor[x].pDac	보통 Power PMAC의 기본값은 Turbo PMAC 시스템의 기본값과 동등함	1,5
Ixx03	Motor[x].pEnc	보통 Power PMAC의 기본값은 Turbo PMAC 시스템의 기본값과 기능적으로 동등함	1,5
Ixx04	Motor[x].pEnc2	보통 Power PMAC의 기본값은 Turbo PMAC 시스템의 기본값과 기능적으로 동등함	1,5
Ixx05	Motor[x].pMasterEnc	엔코더 테이블 등록의 주소로 설정	1,5
Ixx06	Motor[x].MasterCtrl	동일한 값으로 호환 운용 제공	5
Ixx07	Motor[x].MasterPosSf	부동 소수점 값, 그 자체에서 추종의 기어비를 마스터 위치 단위당 모터 단위로 지정 (= Ixx07/Ixx08, Turbo PMAC의 경우)	5
Ixx08	Motor[x].PosSf	피드백 장치의 카운트에 모터의 단위를 설정하려면 PosSf 를 1.0로 설정	4,5
Ixx09	Motor[x].Pos2Sf	피드백 장치의 카운트에 모터의 단위를 설정하려면 Pos2Sf 를 1.0로 설정	4,5
Ixx10	Motor[x].pAbsPos	요소명에 따라 지정된 주소	1,5
Ixx11	Motor[x].FatalFeLimit	단위는 1/16 카운트가 아닌 정수의 모터 단위, 모터 단위가 카운트인 경우에는 FatalFeLimit = Ixx11/16으로 설정하면 호환 운용 가능	2,3,5
Ixx12	Motor[x].WarnFeLimit	단위는 1/16 카운트가 아닌 정수의 모터 단위, 모터 단위가 카운트인 경우에는 WarnFeLimit = Ixx12/16으로 설정하면 호환 운용 가능	2,3,5
Ixx13	Motor[x].MaxPos	모터 단위가 카운트인 경우에는 동일한 값으로 호환 운용 가능	2,3,5
Ixx14	Motor[x].MinPos	모터 단위가 카운트인 경우에는 동일한 값으로 호환 운용 가능	2,3,5

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	Power PMAC 탑재 코멘트	비고
Ixx15	Motor[x].AbortTa	이상 종료 시의 감속률(시간이 아님)을 지정하려면 AbortTa 를 0 보다 작은 값으로 설정하고 반대 비율을 표현하여 설정 모터 단위가 카운트인 경우에는 호환 운용을 위해 AbortTa = -1/Ixx15 로 설정	2,5
Ixx16	Motor[x].MaxSpeed	모터 단위가 카운트인 경우에는 동일한 값으로 호환 운용 가능	2,5
Ixx17	Motor[x].InvAmax Motor[x].InvDmax	역수이므로 Power PMAC 에서의 값은 Turbo PMAC 과 동등한 값의 역수, 모터 단위가 카운트인 경우에는 InvAmax = 1/Ixx17 로 설정하면 호환 운용 가능, 시퀀스의 마지막 감속에는 InvDmax 사용	2,5
Ixx19	Motor[x].JogTa	JOG 동작의 가속도 레이트를 지정(한정값이 아님)하려면 JogTa 를 레이트의 역수의 음의 값으로 설정	
Ixx20	Motor[x].JogTa	Ta > Ts 인 경우에 합계 가속 시간은 Ta+Ts 이며 Ta 만이 아님	5
Ixx21	Motor[x].JogTs	Ta > Ts 인 경우에 합계 가속 시간은 Ta+Ts 이며 Ta 만이 아님	5
Ixx22	Motor[x].JogSpeed	모터 단위가 카운트인 경우에는 동일한 값으로 호환 운용 가능	2,5
Ixx23	Motor[x].HomeVel	모터 단위가 카운트인 경우에는 동일한 값으로 호환 운용 가능	2,5
Ixx24 비트 23	Motor[x].AmpFaultLevel	동일한 값으로 호환 운용 제공	
Ixx24 비트 22-21	Motor[x].FaultMode	FaultMode 비트 0 은 이 모터의 고장을 해결할 때 좌표계 내의 다른 모터를 Abort 또는 Kill 할지 여부를 제어함, 다른 좌표계 내의 모터에는 영향이 없음	
Ixx24 비트 20	Motor[x].pAmpFault	pAmpFault 를 0 으로 설정하면 앰프의 고장 확인이 무효가 됨	
Ixx24 비트 19-18	--	Power PMAC 의 MACRO 플러그에 특별한 모드를 지정할 필요는 없음	
Ixx24 비트 17	Motor[x].pLimits	pLimits 를 0 으로 설정하면 하드웨어의 오버트래블 리미트 확인이 무효가 됨	
Ixx24 비트 16	Motor[x].pAmpEnable	pAmpEnable 을 0 으로 설정하면 앰프 Enable 신호의 출력이 무효가 됨	
Ixx24 비트 15	--	소프트웨어의 오버트래블 리미트에 대해 항상 순간적인 명령 위치가 확인됨	

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	Power PMAC 탑재 코멘트	비고
Ixx24 비트 14	Motor[x].SoftLimitStopDis	동일한 값으로 호환 운용 제공	
Ixx24 비트 13	--	Turbo PMAC2 또는 Power PMAC에서는 지원되지 않는 기능	
Ixx24 비트 12	Motor[x].CaptPosRightShift Motor[x].CaptPosLeftShift	포착한 소수 카운트값을 시프트 아웃할 수 있는지 여부는 CaptPosRightShift 및 CaptPosLeftShift 를 사용	
Ixx24 비트 11	Motor[x].CaptPosRightShift Motor[x].CaptPosLeftShift	포착한 위치의 분해능과 서보 위치의 분해능을 일치시키기 위해 CaptPosRightShift 및 CaptPosLeftShift 를 사용할 수 있음	
Ixx24 비트 10	Motor[x].PhaseMode 비트 0	동일한 값으로 호환 운용 제공	
Ixx24 비트 8	Motor[x].FaultMode 비트 2	동일한 값으로 호환 운용 제공	
Ixx24 비트 1	Motor[x].AmpFaultBit Motor[x].AmpEnableBit Motor[x].LimitBits Motor[x].CaptFlagBit	Power PMAC에서는 유연함을 최대한 확보하기 위해 각 플래그는 해당 레지스터 내의 레지스터 주소와 비트 번호에 따라 지정, 단순히 모든 플래그용의 하나의 스타일의 IC로 제한되는 것이 아님	
Ixx25	Motor[x].pCaptFlag	보통 Power PMAC의 기본값은 Turbo PMAC 시스템의 기본값과 기능적으로 동등함	1
Ixx26	Motor[x].HomeOffset	단위는 1/16 카운트가 아닌 정수의 모터 단위, 모터 단위가 카운트인 경우에는 HomeOffset = Ixx26/16으로 설정하면 호환 운용 가능	2,3,5
Ixx27	Coord[x].PosRollover[j]	Power PMAC에서는 회전축의 롤 오버 계산은 모터가 아닌 축에 대해 실시됨, 단위는 축 단위	
Ixx28	Motor[x].InPosBand	단위는 1/16 카운트가 아닌 정수의 모터 단위, 모터 단위가 카운트인 경우에는 InPosBand = Ixx28/16으로 설정하면 호환 운용 가능	2,3,5
Ixx29	Motor[x].DacBias Motor[x].IaBias	Power PMAC에 전류(commutation) 모터가 없는 경우에는 DacBias 를 사용, Power PMAC에 전류(commutation) 모터가 있는 경우에는 IaBias 를 사용, 동일한 값으로 호환 운용 제공	
Ixx30	Motor[x].Servo.Kp	뒤에 설명하는 '서보 게인의 방정식' 참조	3,5

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	Power PMAC 탑재 코멘트	비고
Ixx31	Motor[x].Servo.Kvfb	뒤에 설명하는 '서보 계인의 방정식' 참조	3,5
Ixx32	Motor[x].Servo.Kvff	뒤에 설명하는 '서보 계인의 방정식' 참조	3,5
Ixx33	Motor[x].Servo.Ki	뒤에 설명하는 '서보 계인의 방정식' 참조	3,5
Ixx34	Motor[x].Servo.SwZvInt	동일한 값으로 호환 운용 제공	5
Ixx35	Motor[x].Servo.Kaff	뒤에 설명하는 '서보 계인의 방정식' 참조	3,5
Ixx36	Motor[x].Servo.Kc1	동일한 값으로 호환 운용 제공	3,5
Ixx37	Motor[x].Servo.Kc2	동일한 값으로 호환 운용 제공	3,5
Ixx38	Motor[x].Servo.Kd1	동일한 값으로 호환 운용 제공	3,5
Ixx39	Motor[x].Servo.Kd2	동일한 값으로 호환 운용 제공	3,5
Ixx40	Motor[x].Pn0 Motor[x].Pd1 Motor[x].PreFilterEna	호환 운용을 가능하게 하려면 Pn0 = Ixx40, Pd1 = -Ixx40 로 설정 PreFilterEna = 1	
Ixx41	Motor[x].SoftLimitOffset	모터 단위가 카운트인 경우에는 동일한 값으로 호환 운용 가능	2,3,5
Ixx42	Motor[x].pAmpEnable Motor[x].pAmpFault	Power PMAC 에서 0 으로 설정한 경우에 플래그의 기능이 무효가 됨(Turbo PMAC 의 경우처럼 Ixx25 주소로 복귀하지 않음)	1
Ixx43	Motor[x].pLimits	Power PMAC 에서 0 으로 설정한 경우에 플래그의 기능이 무효가 됨(Turbo PMAC 의 경우처럼 Ixx25 주소로 복귀하지 않음)	1
Ixx44	Motor[x].pMotorNode Motor[x].MotorMode	Power PMAC 에서는 주소와 동작 모드가 2 개의 변수에 분할됨 (2013 년의 버전 1.6 펌웨어의 새로운 기능)	1
Ixx55	Motor[x].pSineTable Motor[x].pVoltSineTable	기본값 Ixx55 가 0 인 경우에 이를 기본값인 Sys.SineTable[0].a 로 설정함, 기본값 이외의 주소는 설명서 참조	1
Ixx56	Motor[x].AdvGain	AdvGain = Ixx56*Ixx09*32 로 설정하면 호환 운용 가능	3,5
Ixx57	Motor[x].I2tSet	동일한 값으로 호환 운용 제공	3,5
Ixx58	Motor[x].I2tTrip	I2tTrip = Ixx58*2 ³⁰ * Sys.ServoPeriod /1000 으로 설정하면 호환 운용 가능	3,5
Ixx59	Motor[x].Ctrl	Power PMAC 에서는 Motor[x].Ctrl 은 선택한 알고리즘의 주소를 지정, 내장된 여러 알고리즘 또는 커스터마이징 알고리즘 중에 선택 가능	

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	Power PMAC 탑재 코멘트	비고
Ixx60	Motor[x].Stime	동일한 값으로 호환 운용 제공	5
Ixx61	Motor[x].liGain	liGain = $8 \times Ixx61$ 로 설정하면 호환 운용이 가능	3,5
Ixx62	Motor[x].lpfGain	lpfGain = $4 \times Ixx62$ 로 설정하면 호환 운용이 가능	3,5
Ixx63	Motor[x].Servo.MaxInt	뒤에 설명하는 '서보 게인의 방정식' 참조, 음의 값의 MaxInt 는 Power PMAC 에서는 지원되지 않음	5
Ixx64	Motor[x].Servo.Kbreak	Kbreak = $(Ixx64+16)/16$ 으로 설정하면 호환 운용이 가능	3,5
Ixx65	Motor[x].Servo.BreakPosErr	단위는 정수의 모터 단위, $1/16$ 카운트가 아님, BreakPosErr = $Ixx65/16$ 으로 설정하면 호환 운용이 가능	2,3,5
Ixx66	Motor[x].PwmSf	동일한 값으로 호환 운용 제공	5
Ixx67	Motor[x].Servo.MaxPosErr	단위는 $1/16$ 카운트가 아닌 정수의 모터 단위, 모터 단위가 카운트인 경우에는 MaxPosErr = $Ixx67/16$ 으로 설정하면 호환 운용이 가능	5
Ixx68	Motor[x].Servo.Kfff	동일한 값으로 호환 운용 제공	3,5
Ixx69	Motor[x].MaxDac	동일한 값으로 호환 운용 제공	3,5
Ixx70	Motor[x].PhasePosSf	PhasePosSf = $256/(Ixx71/Ixx70)$ 으로 설정하면 호환 운용이 가능	3
Ixx71	Motor[x].PhasePosSf	PhasePosSf = $256/(Ixx71/Ixx70)$ 으로 설정하면 호환 운용이 가능	3,5
Ixx72	Motor[x].PhaseOffset	동일한 값으로 호환 운용 제공	5
Ixx73	Motor[x].PhaseFindingDac	동일한 값으로 호환 운용 제공	5
Ixx74	Motor[x].PhaseFindingTime	추측법의 동일한 값으로 호환 운용 가능, 스텝퍼법의 경우에는 Ixx74 값에 256 를 곱하면 호환 운용이 가능	5
Ixx75	Motor[x].AbsPhasePosOffset Motor[x].AbsPhasePosForce	절대 위치 읽기의 경우에는 AbsPhasePosOffset = $Ixx75 \times Ixx70$ 으로 설정하면 호환 운용이 가능, 보정의 경우에는 AbsPhasePosForce = $Ixx75 \times 2048 / (Ixx71 / Ixx70)$ 으로 설정하면 호환 운용이 가능	
Ixx76	Motor[x].lpbGain	lpbGain = $4 \times Ixx76$ 으로 설정하면 호환 운용이 가능	3,5
Ixx77	Motor[x].IdCmd	동일한 값으로 호환 운용 제공	3,5
Ixx78	Motor[x].DtOverRotorTc	동일한 값으로 호환 운용 제공	3,5
Ixx79	Motor[x].lbBias	동일한 값으로 호환 운용 제공	3,5

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	Power PMAC 탑재 코멘트	비고
Ixx80	Motor[x].PowerOnMode Motor[x].PhaseFindingTime	Ixx80의 비트 0이 PowerOnMode 의 비트 0 및 1에 분할됨, Ixx80의 비트 1은 PhaseFindingTime 의 최소치를 사용하여 메소드를 선택	
Ixx81	Motor[x].pAbsPhasePos	요소명에 따라 지정된 주소	1,5
Ixx82	Motor[x].pAdc	보통 Power PMAC의 기본값은 Turbo PMAC 시스템의 기본값과 기능적으로 동등함, 레지스터 쌍의 2번째가 아닌 1번째를 지정	1,5
Ixx83	Motor[x].pPhaseEnc	보통 Power PMAC의 기본값은 Turbo PMAC 시스템의 기본값과 기능적으로 동등함	1,5
Ixx84	Motor[x].AdcMask	Power PMAC의 32비트 마스크와 Turbo PMAC의 24비트 마스크, AdcMask = Ixx84*\$100 (*256)으로 설정하면 호환 운용이 가능	5
Ixx85	Motor[x].BISlewRate	단위는 실시간 인터럽트당 정수의 모터 단위, 백그라운드 사이클당 1/16 카운트가 아님	2,5
Ixx86	Motor[x].BISize	단위는 1/16 카운트가 아닌 정수의 모터 단위, 모터 단위가 카운트인 경우에는 BISize = Ixx86/16으로 설정하면 호환 운용이 가능	2,5
Ixx87	Motor[x].BIHysteresis	단위는 1/16 카운트가 아닌 정수의 모터 단위, 모터 단위가 카운트인 경우에는 BIHysteresis = Ixx87/16으로 설정하면 호환 운용이 가능	2,5
Ixx88	Motor[x].InPosTime	서보 사이클의 단위, 백그라운드 사이클이 아님	5
Ixx90	Motor[x].RapidSpeedSel	동일한 값으로 호환 운용 제공	5
Ixx91	Motor[x].AbsPhasePosFormat	Power PMAC에서는 이 변수의 구조는 크게 다름, 자세한 설명을 참조할 것	5
Ixx92	--	Power PMAC에서는 JOG 동작은 항상 다음 서보 사이클 내에서 계산 및 시작됨	
Ixx95	Motor[x].AbsPosFormat	Power PMAC에서는 이 변수의 구조는 크게 다름, 자세한 설명을 참조할 것	5
Ixx96 비트 0	Motor[x].PhaseMode 비트 1	동일한 값으로 호환 운용 제공	

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	Power PMAC 탑재 코멘트	비고
Ixx96 비트 1	Motor[x].Servo.SwFffInt Motor[x].Servo.Kvifb Motor[x].Servo.Kviff	SwFffInt 가 동일한 값이면 마찰 Feed-forward 의 호환 운용 가능, Ixx96 비트 1 = 0 에 따라 호환 운용을 가능하게 하려면 속도 게인 Kvfb 및 Kvff 사용, Ixx96 비트 1 = 1 에 따라 호환 운용을 가능하게 하려면 속도 게인 Kvfb 및 Kviff 사용	
Ixx97	Motor[x].CaptureMode	Ixx97 = 0 또는 1 인 경우에는 CaptureMode = Ixx97 로 설정하면 호환 운용이 가능 Ixx97 = 3 인 경우에는 CaptureMode = 2 로 설정하면 호환 운용이 가능	5
Ixx98	--	Power PMAC 에서는 기어 장착 리졸버에서의 절대 위치의 자동 계산은 지원되지 않음	
Ixx99	--	Power PMAC 에서는 기어 장착 리졸버에서의 절대 위치의 자동 계산은 지원되지 않음	

서보 게인의 방정식

Power PMAC의 기본적인 서보 루프는 Turbo PMAC의 서보 루프와 구조 및 스케일이 다릅니다. Turbo PMAC 'I' 변수의 동작과 일치시키기 위해 필요한 Power PMAC의 서보 요소값을 계산하려면 다음 방정식을 사용합니다.

$$Motor[x].Servo.Kp = Ixx30 * \frac{Ixx08}{2^{19} * Motor[x].PosSf}$$

$$Motor[x].Servo.Kvfb = Ixx31 * \frac{Ixx09 * Ixx30}{2^{26} * Motor[x].Pos2Sf}$$

$$Motor[x].Servo.Kvff = Ixx32 * \frac{Ixx08 * Ixx30}{2^{26} * Motor[x].PosSf}$$

$$Motor[x].Servo.Ki = \frac{Ixx33}{2^{23}}$$

$$Motor[x].Servo.Kaff = Ixx35 * \frac{Ixx08 * Ixx30}{2^{26} * Motor[x].PosSf}$$

$$Motor[x].Servo.MaxInt = Ixx63 * \frac{Ixx08 * Ixx30}{2^{23}}$$

주의:

- Turbo PMAC에서 Power PMAC으로 가장 간단하게 변환하는 방법이 필요한 경우에는 Power PMAC 모터를 피드백 장치의 카운트로 스케일링하고 [Motor[x].PosSf] 및 [Motor[x].Pos2Sf]를 기본값 1.0인 상태로 둡니다.
- Turbo PMAC의 변수 'Ixx96'의 비트 1(값 2)이 1로 설정되고 인티그레이터가 위치 루프가 아닌 속도 루프에 배치되어 있는 경우에는 앞에서 설명한 식에서 [Motor[x].Servo.Kvfb] 및 [Motor[x].Servo.Kvff] 대신 Power PMAC의 요소 [Motor[x].Servo.Kvifb] 및 [Motor[x].Servo.Kviff]를 각각 사용합니다.

데이터 개더링 'I' 변수

Power PMAC 의 데이터 개더링 설정 요소는 플래시 메모리에 저장할 수 없다는 점에 주의하십시오.

Turbo PMAC 'I' 변수	동등한 Power PMAC 설정 요소	코멘트	비고
I5000 비트 0	--	Power PMAC 에는 DPRAM 이 없으므로 항상 메인 메모리에 수집, Power PMAC 에서는 수집이 활성화된 동안 데이터의 업로드가 가능	
I5000 비트 1	Gather.Enable	Enable = 2 의 경우에 롤 오버 없는 수집을 설정 Enable = 3 의 경우에 롤 오버 있는 수집을 설정	
I5001~I5048	Gather.Addr[i]	보통은 숫자에 의한 주소값이 아니라 요소의 이름으로 지정	1
I5049	Gather.Period	동일한 값으로 호환 운용 제공	
I5050~I5051	Gather.Items	Items 은 I5050 및 I5051 과 같은 마스크가 아님, Addr[0]~Addr[Items-1] 의 소스를 수집	

ADC 다중 분리 'I' 변수

이러한 설정 변수는 ACC-36E 및 ACC-59E 의 다중화된 ADC 처리를 제어합니다.

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	코멘트	비고
I5060	AdcDemux.Enable	Enable 은 링 크기 및 ON/OFF 기능을 제어	
I5061~I5076	AdcDemux.Address[i]	보통은 숫자에 의한 주소값이 아니라 요소의 이름으로 지정	1
I5080	AdcDemux.Enable	Enable 은 링 크기 및 ON/OFF 기능을 제어	
I5081~I5096	AdcDemux.ConvertCode[i]	Power PMAC 에서는 ADC 의 페어 양쪽을 동일한 방법으로 사용해야 함(유니폴라 또는 바이폴라)	

좌표계의 'I' 변수

이 섹션에서는 Turbo PMAC의 좌표계 고유의 'I' 변수 및 Power PMAC에서 이와 동등한 기능을 목록으로 기재합니다. Power PMAC의 좌표계 구조체 인덱스 'x'는 Turbo PMAC의 'I' 변수 번호에서 50을 뺀 값의 백의 자리의 값과 동등합니다([Coord[3].SegMoveTime]은 'I5313'과 동등함).

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	코멘트	비고
Isx11	--	Power PMAC에서는 직접 지원되지 않음, Status 요소 [Sys.ServoCount] 및 사용자 변수를 지정	
Isx12	--	Power PMAC에서는 직접 지원되지 않음, Status 요소 [Sys.ServoCount] 및 사용자 변수를 지정	
Isx13	Coord[x].SegMoveTime	동일한 값으로 호환 운용 제공	3,5
Isx14	--	(Power PMAC에서는 지원되지 않음)	
Isx15	Coord[x].SegOverride	Power PMAC에서는 동등한 값은 1.0, Turbo PMAC의 경우보다 큼(1.0은 실시간, 0.0이 아님). Power PMAC에서는 저장되지 않음	5
Isx16	Coord[x].SegOverrideSlew	동일한 값으로 호환 운용 제공	5
Isx20	Coord[x].LHDistance	동일한 값으로 호환 운용 제공	5
Isx21	--	룩 어헤드 상태를 신속하게 변경하기 위해 프로그램 다이렉트 명령 lh\, lh<, lh>를 사용	
Isx50	--	Power PMAC의 좌표계에 키네마틱스 루틴이 존재하는 경우에 그것이 자동으로 실행	
Isx53	Coord[x].StepMode	동일한 값으로 호환 운용 제공 (버전 1.6 펌웨어부터 도입)	
Isx78	Coord[x].MaxCirAccel	동일한 값으로 호환 운용 제공	5
Isx79	Coord[x].RapidVelCtrl	동일한 값으로 호환 운용 제공	5
Isx81	Coord[x].InPosTimeOut	동일한 값으로 호환 운용 제공	
Isx82	Coord[x].AddedDwellTime	동일한 값으로 호환 운용 제공	5
Isx83	Coord[x].CornerBlendBp	동일한 값으로 호환 운용 제공	5
Isx84	Coord[x].CCCtrl 비트 0	동일한 값으로 호환 운용 제공	5
Isx85	Coord[x].CornerDwellBp	동일한 값으로 호환 운용 제공	5
Isx86	Coord[x].AltFeedRate	동일한 값으로 호환 운용 제공	5
Isx87	Coord[x].Ta Coord[x].Td	Ta > Ts 인 경우에 합계 가속 시간은 Ta+Ts이며 Ta만이 아님 Td는 블렌드된 동작 시퀀스의 최후의 감속에 사용	
Isx88	Coord[x].Ts	Ta > Ts 인 경우에 합계 가속 시간은 Ta+Ts이며 Ta만이 아님	5

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	코멘트	비고
Isx89	Coord[x].Tm	분리 비트(음은 이송 속도)가 아니며 이 요소의 부호에 따라 이송 속도/동작 시간의 차이를 설정, 보통 이 요소의 값은 프로그램 명령에 따라 자동으로 설정됨	5
Isx90	Coord[x].FeedTime	동일한 값으로 호환 운용 제공	5
Isx91	--	Turbo PMAC2 또는 Power PMAC에서는 컨트롤 패널이 지원되지 않음	
Isx92	Coord[x].NoBlend	동일한 값으로 호환 운용 제공	5
Isx93	Coord[x].pDesTimeBase	보통 Power PMAC의 기본값은 Turbo PMAC 시스템의 기본값과 동등함	1,5
Isx94	Coord[x].TimeBaseSlew	TimeBaseSlew 를 $Isx94/2^{23}$ 으로 설정하면 호환 운용이 가능	5
Isx95	Coord[x].FeedHoldSlew	FeedHoldSlew 를 $Isx95/2^{23}$ 으로 설정하면 호환 운용이 가능	5
Isx96	Coord[x].RadiusErrorLimit	동일한 값으로 호환 운용 제공	5
Isx97	Coord[x].MinArcLen	Isx97 > 0의 경우에는 MinArcLen 을 $\pi * Isx97$ 로 설정하면 호환 운용이 가능 Isx97 = 0의 경우에는 MinArcLen 을 $\pi * 2^{-20}$ 으로 설정하면 호환 운용이 가능	5
Isx98	Coord[x].MaxFeedRate	동일한 값으로 호환 운용 제공	5
Isx99	Coord[x].CCAddedArcBp	동일한 값으로 호환 운용 제공	5

PMAC2 스타일 MACRO IC 의 'I' 변수

IC 의 설정용 'I' 변수는 하드웨어 레지스터 안에 있으므로 Turbo PMAC 및 Power PMAC 에서 기능은 동일합니다. 이름만 변경되었습니다.

대부분의 경우에 Power PMAC MACRO IC 의 구조체 인덱스 'j'는 Turbo PMAC MACRO IC 번호(0~3)와 동등합니다. Power PMAC 채널의 서브 구조체 인덱스 'j'는 Turbo PMAC 의 'I' 변수 번호에서 1 을 빼 값의 십의 자리의 숫자 'n'과 동등합니다 ([Gate2[0].Chan[0].EncCtrl]은 'I6810'과 동등).

IC 를 사용한 하드웨어의 이름을 나타내는 데이터 구조체 [Gate2[j]]의 별칭이 대신 사용되는 경우가 있습니다([Acc5E[j]] 등).

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	코멘트	비고
I6m00/ I6m50	Gate2[j].PwmPeriod	동일한 값으로 호환 운용 제공	5
I6m01/ I6m51	Gate2[j].PhaseClockDiv	동일한 값으로 호환 운용 제공	5
I6m02/ I6m52	Gate2[j].ServoClockDiv	동일한 값으로 호환 운용 제공	5
I6m03/ I6m53	Gate2[j]. HardwareClockCtrl	동일한 값으로 호환 운용 제공	5
I6m04/ I6m54	Gate2[j].PwmDeadTime	동일한 값으로 호환 운용 제공	5
I6m05/ I6m55	Gate2[j].DacStrobe	동일한 값으로 호환 운용 제공	5
I6m06/ I6m56	Gate2[j].AdcStrobe	동일한 값으로 호환 운용 제공	5
I6m07/ I6m57	Gate2[j].PhaseServoDir	동일한 값으로 호환 운용 제공	5
I6mn0	Gate2[j].Chan[j].EncCtrl	동일한 값으로 호환 운용 제공	5
I6mn1	Gate2[j].Chan[j].Equ1Ena	동일한 값으로 호환 운용 제공	5
I6mn2	Gate2[j].Chan[j].CaptCtrl	동일한 값으로 호환 운용 제공	5
I6mn3	Gate2[j].Chan[j].CaptFlag	동일한 값으로 호환 운용 제공	5
I6mn4	Gate2[j].Chan[j]. GatedIndexSel	동일한 값으로 호환 운용 제공	5
I6mn5	Gate2[j].Chan[j]. IndexGateState	동일한 값으로 호환 운용 제공	5
I6mn6	Gate2[j].Chan[j]. OutputMode	동일한 값으로 호환 운용 제공	5
I6mn7	Gate2[j].Chan[j]. OutputPol	동일한 값으로 호환 운용 제공	5
I6mn8	Gate2[j].Chan[j]. PfmDirPol	동일한 값으로 호환 운용 제공	5
I6mn9	Gate2[j].Chan[j]. OneOverTEna	동일한 값으로 호환 운용 제공	5
I6m40/ I6m90	Gate2[j].MacroMode	동일한 값으로 호환 운용 제공	5
I6m41/ I6m91	Gate2[j].MacroEnable	동일한 값으로 호환 운용 제공	5

PMAC2 스타일 Servo IC 의 'I' 변수

Servo IC 의 설정용 'I' 변수는 하드웨어 레지스터 안에 있으므로 Turbo PMAC 및 Power PMAC 에서 기능은 동일합니다. 이름만 변경되었습니다.

Power PMAC 의 Servo IC 구조체 인덱스 'I'는 Turbo PMAC 의 'I' 변수 번호의 백의 자리 'm'의 2 배입니다([Gate1[4].PwmPeriod]는 'I7200'과 동등). Power PMAC 채널의 서브 구조체 인덱스 'j'는 Turbo PMAC 의 'I' 변수 번호에서 1 을 뺀 값의 십의 자리의 숫자 'n'과 동등합니다([Gate1[6].Chan[0].EncCtrl]은 'I7310'과 동등).

IC 를 사용하는 하드웨어의 이름을 나타내는 데이터 구조체 [Gate1[I]]의 별칭이 대신 사용되는 경우가 있습니다. 즉, [Acc24E2[I]], [Acc24E2A[I]], [Acc24E2S[I]] 또는 [Acc51E[I]]입니다.

Turbo PMAC 'I' 변수	동등한 Power PMAC 저장 설정 요소	코멘트	비고
I7m00	Gate1[I].PwmPeriod	동일한 값으로 호환 운용 제공	5
I7m01	Gate1[I].PhaseClockDiv	동일한 값으로 호환 운용 제공	5
I7m02	Gate1[I].ServoClockDiv	동일한 값으로 호환 운용 제공	5
I7m03	Gate1[I].HardwareClockCtrl	동일한 값으로 호환 운용 제공	5
I7m04	Gate1[I].PwmDeadTime	동일한 값으로 호환 운용 제공	5
I7m05	Gate1[I].DacStrobe	동일한 값으로 호환 운용 제공	5
I7m06	Gate1[I].AdcStrobe	동일한 값으로 호환 운용 제공	5
I7m07	Gate1[I].PhaseServoDir	동일한 값으로 호환 운용 제공	5
I7mn0	Gate1[I].Chan[j].EncCtrl	동일한 값으로 호환 운용 제공	5
I7mn1	Gate1[I].Chan[j].Equ1Ena	동일한 값으로 호환 운용 제공	5
I7mn2	Gate1[I].Chan[j].CaptCtrl	동일한 값으로 호환 운용 제공	5
I7mn3	Gate1[I].Chan[j].CaptFlag	동일한 값으로 호환 운용 제공	5
I7mn4	Gate1[I].Chan[j].GatedIndexSel	동일한 값으로 호환 운용 제공	5
I7mn5	Gate1[I].Chan[j].IndexGateState	동일한 값으로 호환 운용 제공	5
I7mn6	Gate1[I].Chan[j].OutputMode	동일한 값으로 호환 운용 제공	5
I7mn7	Gate1[I].Chan[j].OutputPol	동일한 값으로 호환 운용 제공	5
I7mn8	Gate1[I].Chan[j].PfmDirPol	동일한 값으로 호환 운용 제공	5
I7mn9	Gate1[I].Chan[j].OneOverTEna	동일한 값으로 호환 운용 제공	5

엔코더 변환 테이블의 'I' 변수

Power PMAC 의 엔코더 변환 테이블(ECT)의 구조는 Turbo PMAC 의 경우와 크게 다릅니다. Turbo PMAC 에서는 설정이 일련의 'I' 변수('I8000'~'I8191')로 구성되고 각 항목은 1~5 개의 'I' 변수로 구성됩니다.

Power PMAC 에서는 이 테이블은 일련의 인덱스가 있는 데이터 구조체의 형태를 취하고 테이블 내의 각 항목을 구성하는 하나의 데이터 구조체는 각각 서보에서 사용하는 하나의(프라이머리) 결과를 생성합니다. Power PMAC ECT 의 각 항목에는 동일한 구조체가 있지만 구조체 내의 모든 요소가 각각의 변환 방법에서 사용되는 것은 아닙니다. Power PMAC 은 최대 768 개의 [EncTable[n]] 항목을 지원합니다($n = 0 \sim 767$).

Power PMAC 에서는 변환 테이블의 결과 사용을 지정하는 모터의 서보 루프 설정 변수 [Motor[x].pEnc], [Motor[x].pEnc2] 및 [Motor[x].pMasterEnc]는 필요한 결과가 포함된 ECT 항목의 기준 주소로 설정됩니다(EncTable[n].a).

Turbo PMAC 메소드의 수치 \$0: 소프트웨어에 따른 1/T 엔코더의 보간

```
EncTable[n].type = 3
EncTable[n].pEnc = Gate1[j].Chan[j].ServoCapt.a
EncTable[n].pEnc1 = Gate1[j].Chan[j].TimeBetweenCts.a
EncTable[n].index1 = 0
EncTable[n].index2 = 0
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].ScaleFactor = 1/512
```

Turbo PMAC 메소드의 수치 \$1: Acc-28E 16 비트 ADC 변환

```
EncTable[n].type = 1
EncTable[n].pEnc = Acc28E.AdcSdata[j].a
EncTable[n].index1 = 16
EncTable[n].index2 = 16
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].ScaleFactor = 1/65536
```

Turbo PMAC 메소드의 수치 \$2: Y 레지스터의 가로 읽기, 최대 변화 필터 없음

```
EncTable[n].type = 1
EncTable[n].pEnc = {source element}.a
EncTable[n].index1 = 32 - Turbo 폭값
EncTable[n].index2 = Turbo 오프셋값 + 8
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].MaxDelta = 0
EncTable[n].ScaleFactor =  $1/2^{(32 - \text{Turbo 폭값})}$  (소스의 LSB 에 결과)
EncTable[n].ScaleFactor =  $1/2^{(27 - \text{Turbo 폭값})}$  (소스의 1/32 LSB 에 결과)
```

Turbo PMAC 메소드의 수치 \$3: Y 레지스터의 가로 읽기, 최대 변화 필터 있음

```
EncTable[n].type = 1
EncTable[n].pEnc = {source element}.a
EncTable[n].index1 = 32 - Turbo 폭값
EncTable[n].index2 = Turbo 오프셋값 + 8
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].MaxDelta = Turbo 최대 변경값 (서보 사이클당 소스 LSB)
EncTable[n].ScaleFactor =  $1/2^{(32 - \text{Turbo 폭값})}$  (소스의 LSB 에 결과)
EncTable[n].ScaleFactor =  $1/2^{(27 - \text{Turbo 폭값})}$  (소스의 1/32 LSB 에 결과)
```

Turbo PMAC 메소드의 수치 \$4: 타임 베이스 변환

이 섹션에서는 PMAC2 스타일 DSPGATE1 Servo IC 에서의 소프트웨어에 의한 1/T 확장을 사용하는 인크리멘탈 엔코더를 이용한 타임 베이스 변환을 실시하는 경우의 Power PMAC 설정을 설명합니다. 이는 엔코더의 위치값을 피드백용으로 처리하는 경우의 항목과 동일한 타입입니다. 좌표계는 피드백 또는 마스터 위치의 경우의 모터의 서보 알고리즘 사용의 경우와는 다른 레지스터를 타임 베이스 소스로 포인트합니다.

```
EncTable[n].type = 3
EncTable[n].pEnc = Gate1[j].Chan[j].ServoCapt.a
EncTable[n].pEnc1 = Gate1[j].Chan[j].TimeBetweenCts.a
EncTable[n].index1 = 0
EncTable[n].index2 = 0
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].ScaleFactor =  $1/(512 * \text{RTIF[cts/msec]})$ 
```

```
Coord[x].pDesTimeBase = EncTable[n].DeltaPos.a
```

Turbo PMAC 메소드의 수치 \$5: 적분에 의한 Acc-28E 16 비트 ADC 변환

```
EncTable[n].type = 1
EncTable[n].pEnc = Acc28E.AdcSdata[j].a
EncTable[n].index1 = 16
EncTable[n].index2 = 16
EncTable[n].index3 = 0
EncTable[n].index4 = 1
EncTable[n].index5 = 0
EncTable[n].PrevDelta =  $65536 \times \{16 \text{ 비트 바이어스값}\}$ 
EncTable[n].ScaleFactor =  $1/65536$ 
```

Turbo PMAC 메소드의 수치 \$6: Y/X 레지스터의 가로 읽기, 최대 변화 필터 없음

```
EncTable[n].type = 1
EncTable[n].pEnc = {source element}.a
EncTable[n].index1 = 32 - Turbo 폭값
EncTable[n].index2 = Turbo 오프셋값 - 16*
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].MaxDelta = 0
EncTable[n].ScaleFactor =  $1/2^{(32 - \text{Turbo 폭값})}$  (소스의 LSB 에 결과)
EncTable[n].ScaleFactor =  $1/2^{(27 - \text{Turbo 폭값})}$  (소스의 1/32 LSB 에 결과)
```

* 사용되는 모든 데이터는 Turbo PMAC 의 X 레지스터 안에 있다고 가정하였습니다.

Turbo PMAC 메소드의 수치 \$7: Y/X 레지스터의 가로 읽기, 최대 변화 필터 있음

```
EncTable[n].type = 1
EncTable[n].pEnc = {source element}.a
EncTable[n].index1 = 32 - Turbo 폭값
EncTable[n].index2 = Turbo 오프셋값 - 16*
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].MaxDelta = Turbo 최대 변경값 (서보 사이클당 소스 LSB)
EncTable[n].ScaleFactor =  $1/2^{(32 - \text{Turbo 폭값})}$  (소스의 LSB 에 결과)
EncTable[n].ScaleFactor =  $1/2^{(27 - \text{Turbo 폭값})}$  (소스의 1/32 LSB 에 결과)
```

* 사용되는 모든 데이터는 Turbo PMAC 의 X 레지스터 안에 있다고 가정하였습니다.

Turbo PMAC 메소드의 수치 \$8: 인크리멘탈 엔코더의 병렬 확장

이 변환 방법은 Power PMAC 에 업그레이드 가능한 Turbo PMAC 시스템 또는 모든 Power PMAC 시스템에서 지원되지 않습니다.

Turbo PMAC 메소드의 수치 \$9/\$A/\$B: 트리거에 의한 타임 베이스 변환

이 섹션에서는 PMAC2 스타일 DSPGATE1 Servo IC 에서의 소프트웨어에 의한 1/T 확장을 사용하는 인크리멘탈 엔코더를 이용한 트리거에 의한 타임 베이스에 따라 변환하는 경우의 Power PMAC 설정을 설명합니다.

프리즈 상태(Turbo PMAC 메소드 수치= \$9):

```
EncTable[n].type = 10
EncTable[n].pEnc = Gate1[j].Chan[j].ServoCapt.a
EncTable[n].pEnc1 = Gate1[j].Chan[j].TimeBetweenCts.a
EncTable[n].index1 = 0
EncTable[n].index2 = 0
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].ScaleFactor = 1/(512 × RTIF[cts/msec])
```

```
Coord[x].pDesTimeBase = EncTable[n].PrevDelta.a
```

작동 상태(Turbo PMAC 메소드 수치= \$B):

```
EncTable[n].pEnc1 = Gate1[j].Chan[j].Status.a
EncTable[n].index1 = 2
(Gate1[j].Chan[j].CaptCtrl 및 Gate1[j].Chan[j].FlagCtrl 에 따라 정의된 트리거 조건)
```

작동 상태(Turbo PMAC 메소드 수치= \$A):

지정된 트리거가 발생하면 Power PMAC 은 다음을 설정해서 자동으로 항목을 트리거되지 않은 타임 베이스와 동등한 실행 중 상태로 변경합니다.

```
EncTable[n].type = 3
EncTable[n].pEnc1 = Gate1[j].Chan[j].TimeBetweenCts.a
EncTable[n].index1 = 0
```

Turbo PMAC 메소드의 수치 \$C: 인크리멘탈 엔코더의 확장 없음

```
EncTable[n].type = 1
EncTable[n].pEnc = Gate1[j].Chan[j].PhaseCapt.a
EncTable[n].index1 = 8
EncTable[n].index2 = 8
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].MaxDelta= 0
EncTable[n].ScaleFactor = 1/256
```

Turbo PMAC 메소드의 수치 \$D: 로 패스 필터

Turbo PMAC 이 엔코더 변환 테이블에서는 로 패스 필터(인티그레이터가 없는 단순한 지수 필터 또는 인티그레이터가 있는 트래킹 필터)에 따라 거의 항상 실제 소스 데이터를 처리하는 별도의 테이블 항목의 결과가 처리됩니다. 이 섹션에서는 이를 어떻게 Power PMAC 테이블에서 실행하는지 설명합니다. Power PMAC 테이블은 소스 데이터를 처리하는 항목과 동일한 항목이며 보다 유연하게 필터링을 실행할 수 있습니다. 자세한 내용은 "사용 설명서"의 ECT 관련 장을 참조하십시오.

지수 필터(Turbo PMAC 의 최초의 설정 워드 비트 19 = 0)에 대해서는 이전 항목 '*m*'의 결과를 처리하도록 Power PMAC 을 설정하는 경우에는 다음과 같습니다.

```
EncTable[n].type = 1
EncTable[n].pEnc = EncTable[m].PrevEnc.a
EncTable[n].index1 = 0
EncTable[n].index2 = 256 - (Turbo Kp 계인/32768)
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].MaxDelta= Turbo 최대 변경값  $\times 2^{N*}$ 
EncTable[n].ScaleFactor = 1/256
```

* 'N'은 시프트 후의 소스 데이터의 LSB 의 비트 번호입니다.

트래킹 필터(Turbo PMAC 의 최초의 설정 워드 비트 19 = 1)에 대해서는 이전 항목 '**m**'의 결과를 처리하도록 Power PMAC 을 설정하는 경우에는 다음과 같습니다.

```
EncTable[n].type = 1
EncTable[n].pEnc = EncTable[m].PrevEnc.a
EncTable[n].index1 = Turbo Ki 계인/32768**
EncTable[n].index2 = 256 - (Turbo Kp 계인/32768)
EncTable[n].index3 = 0
EncTable[n].index4 = 0**
EncTable[n].index5 = 0
EncTable[n].MaxDelta= Turbo 최대 변경값  $\times 2^{N*}$ 
EncTable[n].ScaleFactor = 1/256
```

** [index1]의 식에 따라 얻어지는 값이 32 보다 작은 경우에 [index4]를 1 증가시키고 [index1]에 2 를 곱하여 [index1]이 32 이상이 되게 합니다.

Power PMAC 에서는 [index1] 및 [index2]는 정수여야만 하기 때문에 숫자를 반올림해야 합니다.

Turbo PMAC 메소드의 수치 \$E: 항목의 덧셈 또는 뺄셈

Turbo PMAC 의 엔코더 변환 테이블에서는 데이터 소스의 덧셈 또는 뺄셈에서는 덧셈 또는 뺄셈하는 소스로 이전 항목의 결과를 사용해야 합니다. Power PMAC ECT 에서는 이전 항목의 결과는 가장 일반적으로 덧셈 또는 뺄셈의 소스로 사용되지만 필수는 아닙니다. 이 섹션에서는 Power PMAC 테이블 외의 항목의 결과를 덧셈 또는 뺄셈하는 방법을 나타냅니다. 다른 소스를 더하거나 빼는 방법에 대해서는 "사용 설명서"의 ECT 관련 장을 참조하십시오.

2 개의 항목(Turbo PMAC 의 최초의 설정 워드 비트 17 = 0)의 결과를 더하는 경우에 이전 항목 '**m**' 및 '**p**'를 더하기 위한 Power PMAC 의 설정은 다음과 같습니다.

```
EncTable[n].type = 8
EncTable[n].pEnc = EncTable[m].PrevEnc.a
EncTable[n].pEnc1 = EncTable[p].PrevEnc.a
EncTable[n].index1 = 0
EncTable[n].index2 = 0
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].MaxDelta= 0
EncTable[n].ScaleFactor = 1.0
```

2 개의 항목의 결과를 더하는 경우(Turbo PMAC 의 최초의 설정 워드 비트 17 = 0), 이전 항목 '**m**'에서 이전 항목 '**p**'를 빼기 위한 Power PMAC 의 설정은 다음과 같습니다.

```
EncTable[n].type = 9
EncTable[n].pEnc = EncTable[m].PrevEnc.a
EncTable[n].pEnc1 = EncTable[p].PrevEnc.a
EncTable[n].index1 = 0
EncTable[n].index2 = 0
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].MaxDelta= 0
EncTable[n].ScaleFactor = 1.0
```

Turbo PMAC 메소드의 수치 \$F/\$0: 고분해능 사인파 엔코더에 의한 보간

```
EncTable[n].type = 4
EncTable[n].pEnc = Gate1[j].Chan[j].Status.a
EncTable[n].pEnc1 = Gate1[j].Chan[j].Adc[0].a
EncTable[n].index1 = 0
EncTable[n].index2 = 0
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].SinBias = 16 × Turbo 사인 바이어스 조건
EncTable[n].CosBias = 16 × Turbo 코사인 바이어스 조건
EncTable[n].ScaleFactor = 1/(512 × RTIF[cts/msec])
```

앞에서 설명한 설정을 사용하면 이 ECT 항목에서는 로 패스 필터링은 실행되지 않습니다. 이 타입의 필터링은 이 항목 내에 있는지 다른 항목 내에 있는지에 관계없이 사인파 엔코더 변환에 적합한 경우가 있습니다.

Turbo PMAC 메소드의 수치 \$F/\$1: 고분해능 사인파 엔코더에 의한 보간 진단

Power PMAC 에서는 표준 고분해능의 사인파 엔코더의 보간 항목(**type** = 4, 앞에서 설명)은 자동으로 서보 사이클마다 엔코더 신호의 벡터값(의 제곱)을 계산합니다. 이는 Status 요소 [EncTable[n].SumOfSqr]로 읽을 수 있습니다.

Power PMAC 의 ECT 에는 자동으로 사인파 엔코더의 사인 및 코사인 입력의 오프셋을 추측하는 방법은 없습니다.

Turbo PMAC 메소드의 수치 \$F/\$2: 바이트 폭의 가로 읽기, 최대 변화 필터 없음

```
EncTable[n].type = 5
EncTable[n].pEnc = Gatelo[i].DataReg[j].a
EncTable[n].pEnc1 = Gatelo[i].DataReg[j+1].a
EncTable[n].index1 = 32 - Turbo 폭값
EncTable[n].index2 = Turbo 오프셋값 + 8
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].MaxDelta = Turbo 최대 변경값(서보 사이클당 소스 LSB)
EncTable[n].ScaleFactor = 1/2(32-Turbo 폭값)
```

Turbo PMAC 메소드의 수치 \$F/\$3: 바이트 폭의 가로 읽기, 최대 변화 필터 있음

```
EncTable[n].type = 5
EncTable[n].pEnc = Gatelo[i].DataReg[j].a
EncTable[n].pEnc1 = Gatelo[i].DataReg[j+1].a
EncTable[n].index1 = 32 - Turbo 폭값
EncTable[n].index2 = Turbo 오프셋값 + 8
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].MaxDelta = 0
EncTable[n].ScaleFactor = 1/2(32-Turbo 폭값)
```

Turbo PMAC 메소드의 수치 \$F/\$4: 리졸버에 의한 아크탄젠트 변환

```
EncTable[n].type = 6
EncTable[n].pEnc = Gate1[i].a + 256
EncTable[n].pEnc1 = Gate1[i].Chan[j].Adc[0].a
EncTable[n].index1 = 0
EncTable[n].index2 = 0
EncTable[n].index3 = 0
EncTable[n].index4 = 0
EncTable[n].index5 = 0
EncTable[n].SinBias = 16 × Turbo 사인 바이어스 조건
EncTable[n].CosBias = 16 × Turbo 코사인 바이어스 조건
EncTable[n].ScaleFactor = 1 / 4
```

앞에서 설명한 설정을 사용하면 이 ECT 항목에서는 로 패스 필터링은 실행되지 않습니다. 이 타입의 필터링은 이 항목 내에 있는지 다른 항목 내에 있는지에 관계없이 리졸버 변환에 사용하는 것을 적극 권장합니다.

POWER PMAC TURBO 에 권장하는 'M' 변수와 동등한 기능

이 섹션에서는 최소한의 변경으로 운전 중인 Turbo PMAC 어플리케이션에서 Power PMAC 으로 변환하는 데 도움이 되는 정보를 제공합니다. Turbo PMAC 의 권장하는 'M' 변수의 정의 및 동등한 Power PMAC 기능에 대해 설명하고 이러한 설정에 변경이 필요한지 여부를 나타냅니다(이러한 Turbo PMAC 의 'M' 변수의 정의는 'I' 변수의 경우와 달리 권장 사항뿐이지만 널리 사용되고 있습니다).

이러한 테이블을 직접 적용한 직접적인 변환을 실시하면 Power PMAC 의 모든 기능을 활용할 수 없는 경우가 많습니다. 단, 향후 확장하는 경우에는 이 방법을 사용해도 출발점으로는 충분합니다.

동등한 기능 관련 주의점

1. Turbo PMAC 에서의 이 변수의 위치 단위는 카운트의 $1/[Ixx08*32]$ 또는 $1/[Ixx09*32]$ 등의 단위이며 고정 소수점 형식으로도 충분한 분해능을 제공합니다. Power PMAC 의 부동 소수점을 사용하는 동등한 기능에서는 소수 분해능은 정수의 단위로 제공됩니다.
2. Power PMAC 에서는 이 변수의 위치 단위는 모터 단위이며 이를 엔코더의 카운터로 사용하는 것은 가능하지만 필수는 아닙니다. Turbo PMAC 에서 가장 간단하게 사용 가능한 포트 설정을 실시하는 경우에 모터 단위를 엔코더 또는 동등 기기의 카운트로 설정합니다.
3. Turbo PMAC 에서는 전원 셋업/리셋 후에 위치를 참조하는 동작이 실행된 경우에 이 위치값은 모터의 원점이 기준입니다. Power PMAC 에서는 이 위치값은 항상 전원 셋업/리셋 위치가 기준입니다. 모터의 원점과 전원 셋업/리셋 위치의 차는 `[Motor[x].HomePos]`에 저장되어 있지만 이 값을 Power PMAC 의 값에서 빼면 Turbo PMAC 의 값과 동등합니다.

다양한 글로벌 변수

이 섹션에서는 다양한 태스크에서 경과하는 프로세서 시간의 모니터링에 사용하는 변수를 목록으로 나타냅니다.

Turbo PMAC 'M' 변수	설명	동등한 Power PMAC 요소	Power PMAC 탑재 코멘트	비고
M70	페이지 인터럽트의 간격 (CPU 사이클/2)	Sys.PhaseDeltaTime	PhaseDeltaTime (단위는 마이크로초)	
M71	페이지 태스크의 시간 (CPU 사이클/2)	Sys.PhaseTime, Sys.FltrPhaseTime	PhaseTime, FltrPhaseTime (단위는 마이크로초)	
M72	서보 태스크의 시간 (CPU 사이클/2)	Sys.ServoTime, Sys.FltrServoTime	ServoTime, FltrServoTime (단위는 마이크로초)	
M73	RTI 태스크의 시간 (CPU 사이클/2)	Sys.RtIntTime, Sys.FltrRtIntTime	RtIntTime, FltrRtIntTime (단위는 마이크로초)	

서보 사이클 카운터

서보 사이클 카운터는 서보 인터럽트가 1 회 발생할 때마다 1 증분되어 프로세스 경과 시간을 판단하는 데 사용할 수 있습니다. Power PMAC 카운터의 범위가 더 길다는 점에 주의하십시오. 어떤 경우든 견고한 알고리즘에 따라 롤 오버의 가능성이 고려됩니다.

Turbo PMAC 'M' 변수	설명	동등한 Power PMAC 요소	Power PMAC 탑재 코멘트	비고
M100	24 비트의 서보 사이클 카운터	Sys.ServoCount	ServoCount 는 32 비트의 정수	

PMAC2 Servo IC 레지스터

이 섹션에서는 DSPGATE1 PMAC2 스타일 Servo IC 의 주요 레지스터 및 제어/Status 비트를 목록으로 나타냅니다. Power PMAC 의 스크립트 환경에서는 풀 레지스터 요소는 32 비트 버스 상에서 액세스되지만 24 비트값입니다.

Turbo PMAC 'M' 변수	설명	동등한 Power PMAC 요소	Power PMAC 탑재 코멘트	비고
Mxx01	24 비트 카운터 위치	Gate1[j].Chan[j].PhaseCapt		
Mxx02	A 상 명령값, DAC 또는 PWM	Gate1[j].Chan[j].Dac[0], Gate1[j].Chan[j].Pwm[0]	Dac[0] 및 Pwm[0]과 동일한 레지스터, 16 비트가 아닌 24 비트, 범위	
Mxx03	포착한 위치	Gate1[j].Chan[j].HomeCapt		
Mxx04	B 상 명령값, DAC 또는 PWM	Gate1[j].Chan[j].Dac[1], Gate1[j].Chan[j].Pwm[1]	Dac[1] 및 Pwm[1]은 동일한 레지스터, 16 비트가 아닌 24 비트, 범위	
Mxx05	A 상의 ADC 입력값	Gate1[j].Chan[j].Adc[0]	Adc[0]은 16 비트가 아닌 24 비트, 범위	
Mxx06	B 상의 ADC 입력값	Gate1[j].Chan[j].Adc[1]	Adc[1]은 16 비트가 아닌 24 비트, 범위	
Mxx07	C 상 명령값, PFM 또는 PWM	Gate1[j].Chan[j].Pfm, Gate1[j].Chan[j].Pwm[2]	Pfm 및 Pwm[1]은 동일한 레지스터, 16 비트가 아닌 24 비트, 범위	
Mxx08	위치 비교의 위치 A	Gate1[j].Chan[j].CompA		
Mxx09	위치 비교의 위치 B	Gate1[j].Chan[j].CompB		
Mxx10	위치 비교의 자동 증분값	Gate1[j].Chan[j].CompAdd		
Mxx11	위치 비교의 초기 상태 쓰기 유효	Gate1[j].Chan[j].EquWrite 비트 0		
Mxx12	위치 비교의 초기 상태값	Gate1[j].Chan[j].EquWrite 비트 1		

Mxx14	AENA 출력 상태	Gate1[j].Chan[j].AmpEna		
Mxx15	사용자 플래그 입력 상태	Gate1[j].Chan[j].UserFlag		
Mxx16	위치 비교의 출력값	Gate1[j].Chan[j].Equ		
Mxx17	위치 포착 플래그	Gate1[j].Chan[j].PosCapt		
Mxx18	카운트 에러 플래그	Gate1[j].Chan[j]. CountError		
Mxx19	엔코더 C 채널 입력 상태	Gate1[j].Chan[j].ABC 비트 2		
Mxx20	원점 플래그 입력 상태	Gate1[j].Chan[j].HomeFlag		
Mxx21	플러스 리미트 입력 상태	Gate1[j].Chan[j].PlusLimit		
Mxx22	마이너스 리미트 입력 상태	Gate1[j].Chan[j]. MinusLimit		
Mxx23	이상 플래그 입력 상태	Gate1[j].Chan[j].Fault		
Mxx24	W 플래그 입력 상태	Gate1[j].Chan[j].UVW 비트 0		
Mxx25	V 플래그 입력 상태	Gate1[j].Chan[j].UVW 비트 1		
Mxx26	U 플래그 입력 상태	Gate1[j].Chan[j].UVW 비트 2		
Mxx27	T 플래그 입력 상태	Gate1[j].Chan[j].T		
Mxx28	4비트값으로 TUW 플래그 입력	8*Gate1[j].Chan[j].T + Gate1[j].Chan[j].UVW		

모터 Status 비트

이 섹션에서는 가장 일반적으로 사용되는 Turbo PMAC 의 모터 Status 비트 및 Power PMAC 에서 이와 동등한 기능을 목록으로 기재합니다. Power PMAC 의 모터 구조체 인덱스 'x'는 Turbo PMAC 의 권장하는 'M' 변수 번호의 백의 자리의 값과 동등합니다.

Turbo PMAC 'M' 변수	설명	동등한 Power PMAC 요소	Power PMAC 탑재 코멘트	비고
Mxx30	위치 리미트에서 정지	Motor[x].LimitStop, Motor[x].SoftLimit	Power PMAC 은 HW 리미트 및 SW 리미트에 개별 비트를 사용	
Mxx31	양의 엔드 리미트 설정	Motor[x].PlusLimit, Motor[x].SoftPlusLimit	Power PMAC 은 HW 리미트 및 SW 리미트에 개별 비트를 사용	
Mxx32	음의 엔드 리미트 설정	Motor[x].MinusLimit, Motor[x].SoftMinusLimit	Power PMAC 은 HW 리미트 및 SW 리미트에 개별 비트를 사용	
Mxx33	명령 속도 제로	Motor[x].DesVelZero		
Mxx35	드웰 진행 중	Motor[x].Desired.Dwell		
Mxx37	프로그램 실행 중 (동작 타이머가 활성화 상태)	Motor[x].Desired.TimerEnabled		
Mxx38	오픈 루프 모드	Motor[x].ClosedLoop	Power PMAC 에서는 논리적으로 반대 극성	
Mxx39	앰프 Enable	Motor[x].AmpEna		
Mxx40	백그라운드 인포지션	Motor[x].InPos	Power PMAC 의 모든 인포지션 확인은 포어그라운드	
Mxx41	위치 편차 경고	Motor[x].FeWarn		
Mxx42	편차 이상	Motor[x].FeFatal		
Mxx43	앰프 폴트	Motor[x].AmpFault		
Mxx44	포어그라운드 인포지션	Motor[x].InPos	Power PMAC 의 포어그라운드 인포지션 확인은 스캔 카운트	
Mxx45	원점 복귀 완료	Motor[x].HomeComplete		
Mxx46	적분 편차 이상	--	Power PMAC 에서는 적분 편차 이상이 없음	
Mxx47	I2T 폴트	Motor[x].I2tFault		
Mxx48	페이즈 에러 폴트	Motor[x].PhaseFound	Power PMAC 에서는 논리적으로 반대 극성	
Mxx49	페이즈 서치 중	Motor[x].PhaseFindingStep	Power PMAC 에서는 부울값이 아닌 계수, 단, 페이즈 서치 중은 제로 이외	
--	원점 서치 중	Motor[x].HomeInProgress		
--	트리거 동작 중	Motor[x].TriggerMove		
--	좌표계에 할당	Motor[x].Coord	Coord 는 모터가 할당되는 좌표계 번호를 지정, 보통 모터는 활성화된 좌표계가 아니라 좌표계 0 에 할당됨	

모터 동작 레지스터

이 섹션에서는 일반적으로 사용되는 Turbo PMAC의 모터 동작 레지스터의 위치, 속도 및 서보 명령을 목록으로 기재하고 Power PMAC에서 이와 동등한 기능에 대해서도 병기합니다. Power PMAC의 모터 구조체 인덱스 'x'는 Turbo PMAC의 권장하는 'M' 변수 번호의 백의 자리의 값과 동등합니다.

Turbo PMAC 'M' 변수	설명	동등한 Power PMAC 요소	Power PMAC 탑재 코멘트	비고
Mxx61	명령 위치	Motor[x].DesPos	DesPos 는 최종적인 명령 위치, 마스터 및 보정 포함	1,2,3
Mxx62	현재 위치	Motor[x].ActPos	ActPos 는 최종적인 현재 위치, 백래시 및 실제 보정 포함	1,2,3
Mxx63	대상(엔드) 위치	Motor[x].MoveDesPos		1,2,3
Mxx64	위치 바이어스	Motor[x].CoordSf[32], Tdata[j].Bias[m]	보통 모터 원점에서 축 원점을 오프셋하기 위한 다양한 방법이 사용됨	
Mxx66	현재 속도	Motor[x].ActVel, Motor[x].ActVel2	Mxx66 은 이너 루프 속도, ActVel 은 아우터 루프 속도, ActVel2 는 이너 루프 속도	1,2
Mxx67	마스터 위치	Motor[x].MasterPos, Motor[x].ActiveMasterPos	ActiveMasterPos 는 레이트 제한되어 있으며 서보가 직접 사용함	1,2
Mxx68	필터 출력	Motor[x].IqCmd	Power PMAC에서는 부동 소수점, 동등한 단위	
Mxx69	보정에 의한 수정	Motor[x].CompDesPos, Motor[x].CompPos	Mxx69 는 명령 위치에 추가됨, CompDesPos 는 명령 위치에 추가됨, CompPos 는 현재 위치에 추가됨, 따라서 역방향으로 인식됨	1,2
Mxx70	현재의 페이즈 위치 (소수 부분 포함)	Motor[x].PhasePos	PhasePos (전류(commutation)의 단위, 1/2048 사이클)	1
Mxx71	현재의 페이즈 위치	Motor[x].PhasePos	PhasePos (전류(commutation)의 단위, 1/2048 사이클)	
Mxx72	가변 JOG 위치/거리	Motor[x].ProgJogPos		
Mxx73	엔코더의 원점 포착 위치	Motor[x].HomePos	HomePos (항상 모터 단위)	
Mxx74	평균 현재 속도	Motor[x].FiltrVel, Motor[x].FiltrVel2	Mxx74 는 이너 루프 속도, FiltrVel 은 아우터 루프 속도, FiltrVel2 는 이너 루프 속도	1,2
Mxx75	실제 직교 전류	Motor[x].IqMeas	Power PMAC에서는 부동 소수점, 동등한 단위	
Mxx76	실제 측정된 전류	Motor[x].IdMeas	Power PMAC에서는 부동 소수점, 동등한 단위	

Mxx77	직교 전류 루프의 인티그레이터 출력	Motor[x].IqVolts	Power PMAC에서는 부동 소수점, 동등한 단위	
Mxx78	직류 전류 루프의 인티그레이터 출력	Motor[x].IdVolts	Power PMAC에서는 부동 소수점, 동등한 단위	
Mxx79	PID 내부 필터 결과	Motor[x].ServoOut	Power PMAC에서는 부동 소수점, 동등한 단위	

모터 축 정의 레지스터

이 섹션에서는 Turbo PMAC 이 권장하는 모터의 축 정의 계수인 'M' 변수 및 Power PMAC 에서 이와 동등한 기능을 목록으로 기재합니다. Power PMAC 의 모터 구조체 인덱스 'x'는 Turbo PMAC 이 권장하는 'M' 변수 번호의 백의 자리의 값과 동등합니다([Motor[12].CoordSf[6]]은 'M1291'과 동등함).

Turbo PMAC 'M' 변수	설명	동등한 Power PMAC 요소	Power PMAC 탑재 코멘트	비고
Mxx91	X/U/A/B/C 축의 스케일 계수	Motor[x].CoordSf[6] (X), Motor[x].CoordSf[3] (U), Motor[x].CoordSf[0] (A), Motor[x].CoordSf[1] (B), Motor[x].CoordSf[2] (C)	Power PMAC에는 축마다 개별 요소가 있음	
Mxx92	Y/V 축의 스케일 계수	Motor[x].CoordSf[7] (Y), Motor[x].CoordSf[4] (V)	Power PMAC에는 축마다 개별 요소가 있음	
Mxx93	Z/W 축의 스케일 계수	Motor[x].CoordSf[8] (Z), Motor[x].CoordSf[5] (W)	Power PMAC에는 축마다 개별 요소가 있음	
Mxx94	축 오프셋	Motor[x].CoordSf[32]		

다중 분리 ADC 레지스터

이 섹션에서는 ACC-36 또는 CC-59 보드 상의 A/D 컨버터의 자동 다중 분리 결과용 Turbo PMAC 권장 'M' 변수를 목록으로 나타냅니다.

Turbo PMAC 'M' 변수	설명	동등한 Power PMAC 요소	Power PMAC 탑재 코멘트	비고
M5061 – M5076	I5061~I5076 의 하위 ADC 레지스터의 다중 분리	AdcDemux.ResultLow[i]	인덱스 <i>i</i> 의 범위는 0~15, M5061~M5076 과 일치	
M5081 – M5096	I5061~I5076 의 상위 ADC 레지스터의 다중 분리	AdcDemux.ResultHigh[i]	인덱스 <i>i</i> 의 범위는 0~15, M5081~M5096 과 일치	

좌표계 타이머

Power PMAC 에는 Turbo PMAC 의 전용 카운트다운 타이머의 직접적인 동등 기능은 없습니다. 단, [Sys.ServoCount] 상태 레지스터를 사용하면 간단히 Power PMAC 에서 동등한 로직을 작성할 수 있습니다.

Turbo PMAC 의 로직과 동등한 로직을 작성하는 방법:

```
Msx11 == MyDelay
while (Msx11 > 0) endwhile
```

Power PMAC 의 로직을 사용한 경우:

```
MyEndtime = Sys.ServoCount + MyDelay;
while (Sys.ServoCount < MyEndTime) { }
```

Turbo PMAC 'M' 변수	설명	동등한 Power PMAC 요소	Power PMAC 탑재 코멘트	비고
Msx11	&x Isx11 타이머 (동기 대입용)	(직접적인 동등 기능은 없음)	상기 설명과 같이 서보 카운터 및 범용 변수를 사용	
Msx12	&x Isx12 타이머 (동기 대입용)	(직접적인 동등 기능은 없음)	상기 설명과 같이 서보 카운터 및 범용 변수를 사용	

좌표계의 계산된 동작의 종료 레지스터

이 섹션에서는 Turbo PMAC 이 권장하는 이전에 계산된 동작(현재 실행 중인 동작이 아닌 경우가 있음)의 축 정의의 값 'M' 변수 및 Power PMAC 에서 이와 동등한 기능을 목록으로 기재합니다. Power PMAC 의 모터 구조체 인덱스 'x'는 Turbo PMAC 이 권장하는 'M' 변수 번호에서 50 을 뺀 백의 자리의 값과 동등합니다([Coord[1].CdPos[6]]은 'M5147'과 동등함).

Turbo PMAC 'M' 변수	설명	동등한 Power PMAC 요소	Power PMAC 탑재 코멘트	비고
Msx41	A 축 대상 위치	Coord[x].CdPos[0]		
Msx42	B 축 대상 위치	Coord[x].CdPos[1]		
Msx43	C 축 대상 위치	Coord[x].CdPos[2]		
Msx44	U 축 대상 위치	Coord[x].CdPos[3]		
Msx45	V 축 대상 위치	Coord[x].CdPos[4]		
Msx46	W 축 대상 위치	Coord[x].CdPos[5]		
Msx47	X 축 대상 위치	Coord[x].CdPos[6]		
Msx48	Y 축 대상 위치	Coord[x].CdPos[7]		
Msx49	Z 축 대상 위치	Coord[x].CdPos[8]		

좌표계의 Status 비트

이 섹션에서는 가장 일반적으로 사용되는 Turbo PMAC 의 좌표계 Status 비트 및 Power PMAC 에서 이와 동등한 기능을 목록으로 기재합니다. Power PMAC 의 모터 구조체 인덱스 'x'는 Turbo PMAC 이 권장하는 'M' 변수 번호에서 50 을 뺀 백의 자리의 값과 동등합니다([Coord[2].RunTimeError]은 'M5282'와 동등함).

Turbo PMAC 'M' 변수	설명	동등한 Power PMAC 요소	Power PMAC 탑재 코멘트	비고
Msx80	프로그램 실행 중 비트	Coord[x].ProgRunning	ProgRunning 은 프로그램에 따라 계산된 모든 동작이 실행을 종료할 때까지 True 로 유지됨	
Msx81	Circle 반경 에러	Coord[x].RadiusError		
Msx82	런타임 에러	Coord[x].RunTimeError		
Msx84	계속 동작 요구	Coord[x].ContMotion		
Msx87	인포지션 비트 (모터의 AND)	Coord[x].InPos		
Msx88	경고 위치 편차 (모터의 OR)	Coord[x].FeWarn		
Msx89	편차 이상 (모터의 OR)	Coord[x].FeFatal		
Msx90	앰프 폴트 에러 (모터의 OR)	Coord[x].AmpFault		

좌표계의 타임 베이스 변수

이 섹션에서는 좌표계의 타임 베이스 변수를 목록으로 나타냅니다. Power PMAC 의 모터 구조체 인덱스 'x'는 Turbo PMAC 이 권장하는 'M' 변수 번호에서 50 을 뺀 백의 자리의 값과 동등합니다([Coord[4].TimeBase]는 'M5498'과 동등함).

Turbo PMAC 'M' 변수	설명	동등한 Power PMAC 요소	Power PMAC 탑재 코멘트	비고
Msx97	호스트가 명령한 타임 베이스	Coord[x].DesTimeBase	DesTimeBase 의 단위는 밀리초, 2^{-23} 밀리초는 아님	
Msx98	현재의 타임 베이스	Coord[x].TimeBase	TimeBase 의 단위는 밀리초, 2^{-23} 밀리초는 아님	

UMAC I/O 액세스리의 'M' 변수

이 섹션에서는 UMAC 백플레인 상의 첫 I/O 보드의 I/O 포인트 'M' 변수를 목록으로 나타냅니다. Power PMAC 어플리케이션에서는 I/O 포인트에 'ptr' 변수를 선언하여 'M' 변수를 할당하고 어플리케이션에 관련 있는 이름을 붙이는 것이 일반적입니다.

Turbo PMAC 'M' 변수	설명	동등한 Power PMAC 요소	Power PMAC 탑재 코멘트	비고
M7000	MI/O00	Gatelo[0].DataReg[0].0		
..	..	..		
M7007	MI/O07	Gatelo[0].DataReg[0].7		
M7008	MI/O08	Gatelo[0].DataReg[1].0		
..	..	..		
M7015	MI/O15	Gatelo[0].DataReg[1].7		
M7016	MI/O16	Gatelo[0].DataReg[2].0		
..	..	..		
M7023	MI/O23	Gatelo[0].DataReg[2].7		
M7024	MI/O24	Gatelo[0].DataReg[3].0		
..	..	..		
M7031	MI/O31	Gatelo[0].DataReg[3].7		
M7000	MI/O32	Gatelo[0].DataReg[4].0		
..	..	..		
M7007	MI/O39	Gatelo[0].DataReg[4].7		
M7000	MI/O40	Gatelo[0].DataReg[5].0		
..	..	..		
M7007	MI/O47	Gatelo[0].DataReg[5].7		

POWER PMAC UBUS32 의 사양

백플레인 커넥터의 핀 배열

핀 번호	열 A	열 B	열 C
1	+5V	+5V	+5V
2	GND	GND	GND
3	BD09 (BD01)	BD00 (rsrvd.)	BD08 (BD00)
4	BD11 (BD03)	BD01 (rsrvd.)	BD10 (BD02)
5	BD13 (BD05)	BD02 (rsrvd.)	BD12 (BD04)
6	BD15 (BD07)	BD03 (rsrvd.)	BD14 (BD06)
7	BD17 (BD09)	BD04 (rsrvd.)	BD16 (BD08)
8	BD19 (BD11)	BD05 (rsrvd.)	BD18 (BD10)
9	BD21 (BD13)	BD06 (rsrvd.)	BD20 (BD12)
10	BD23 (BD15)	BD07 (rsrvd.)	BD22 (BD14)
11	BD25 (BD17)	rsrvd	BD24 (BD16)
12	BD27 (BD19)	rsrvd	BD26 (BD18)
13	BD29 (BD21)	rsrvd	BD28 (BD20)
14	BD31 (BD23)	BCSDIR(rsrvd.)	BD30 (BD22)
15	rsrvd	BCS0-(rsrvd.)*	rsrvd
16	BA01	BCS1-(rsrvd.)*	BA00
17	BA04 (BA03)	BCS5-(rsrvd.)*	BA02
18	BA03 (BX/Y)	BA15 (BA14)	BA05 (BA04)
19	BCS3-	BA07 (BA06)	BCS2-
20	BA06 (BA05)	BA08 (BA07)	BCS4-
21	BCS12-	BA09 (BA08)	BCS10-
22	BCS16-	BA10 (BA09)	BCS14-
23	BA14 (BA13)	BA11 (BA10)	BA13 (BA12)
24	BRD-	BA12 (BA11)	BWR-
25	rsrvd	DPRCS1-	rsrvd
26	WAIT-	VMECS1-	BRESET
27	PHASE+	UMAC_INT-	SERVO+
28	PHASE-	INT1- (EQU1-)	SERVO-
29	AGND	INT2- (EQU2-)	AGND
30	A-15V	PWM_ENA	A+15V
31	GND	GND	GND
32	+5V	+5V	+5V

주의:

- 이탤릭체로 표시된 이름은 UBUS 백플레인 상에서 Turbo PMAC CPU 가 이 핀을 사용하는 것을 나타냅니다.
- 'rsrvd'는 '향후 사용하기 위한 예약 완료'를 의미합니다.
- 최초의 문자인 'B'는 백업 완료를 의미하며 이러한 신호는 이 신호를 사용하는 임의의 카드의 백플레인 커넥터에서 25mm 이내에 배치된 버퍼 IC 에 따라 전달되어야 합니다.
- (*) 2007 년 이후의 프로토타입 보드에서는 BCS0-는 핀 B11 에 BCS1-는 B12 에 BCS2-는 B13 에 배치됩니다.

펌웨어의 갱신 이력

이 섹션에서는 Power PMAC 펌웨어의 각 개정에서 제공된 추가 및 수정을 목록으로 나타냅니다.

V2.0 펌웨어 갱신(2014 년 12 월)

1. 싱글코어 및 듀얼코어 AMCC465(86xxx) CPU 지원을 추가
2. Soft Power PMAC 에 멀티코어 x86 CPU 지원을 추가. Microsoft Windows™ 와 Linux 양쪽을 다른 코어에서 동시에 실행 가능
3. 데이터 구조체 [Gate3[i]]에 별칭 [Clipper[i]]를 추가 새로운 Power Clipper 내장 컨트롤러를 지원
4. 새로운 Power Brick 제품 라인 지원을 더욱 확충
5. 기존의 데이터 구조체 [Acc84E[i]]와 동등한 데이터 구조체 [Acc84C[i]]를 사용하는 Compact UMAC ACC-84C 시리얼 엔코더의 인터페이스 보드의 데이터 구조체 지원을 추가
6. I/O 칩 선택 시에 수동으로 액세스러리의 유무를 지정할 수 있는 기능을 추가. 이로 인해 자동 식별용 ID 칩이 없는 구형 액세스러리가 데이터 구조체 액세스에서 사용 가능. ID Status 요소 [Gatelo[i]], [PartNum], [PartOpt], [PartRev] 및 [PartType]의 쓰기 액세스 제공을 통해 실현. ACC-84S Clipper 시리얼 엔코더 보드 및 ACC-11E UMAC I/O 보드의 지원이 주 목적
7. 기존의 데이터 구조체 [Acc84E[i]]와 동등한 데이터 구조체 [Acc84S[i]]를 사용하는 Clipper ACC-84S 시리얼 엔코더의 인터페이스 보드의 데이터 구조체 지원을 추가
8. 예비 채널 구조체 요소를 데이터 구조체 [Acc84x[i]]에 추가하여 2 개의 엔코더 인터페이스 IC 를 탑재한 ACC-84S 보드를 지원
9. 데이터 구조체 [Gatelo[i]]의 별칭 [Acc11E[i]]에 따라 UMAC ACC-11E 디지털 I/O 보드의 데이터 구조체 지원을 추가
10. 테이블 값에서 대상 모터의 오프셋을 사용자가 지정할 수 있도록 저장 설정 요소 [CamTable[m].PosBias]를 추가
11. 저장 설정 요소 [CamTable[m].DacGain], [CamTable[m].MinPosError] 및 [CamTable[m].MaxDac]를 추가하여 토크 출력을 최적화하는 반복 학습 제어를 지원
12. 모터 동작이 명령된 경우의 모터의 브레이크 상태 확인을 추가. 브레이크가 완전히 해제되지 않은 경우(해제가 명령되었음에도) 동작 명령은 에러로 거부
13. 'PosCtrl' 위치 출력 모드를 사용 중인 모터에 기본값 1.0 이외에 [Motor[x].PosSf] 값을 사용할 수 있는 기능을 추가
14. [BrakeOffDelay] 및 [BrakeOnDelay] 주기 동안에도 진행 상태를 감시할 수 있도록 [Motor[x].BrakeTimer]에 대한 스크립트 읽기 액세스를 추가

15. 안정성을 향상시키기 위해 엔코더 번호나 테이블의 이중 적분을 통해 댐핑을 제공할 수 있는 기능을 추가. 이 모드에서의 `[EncTable[n].index4]`의 최대값을 2 에서 15 로 확장. 값이 2 보다 큰 경우에 $1/(2^{16-index4})$ 을 가중하는 지수 필터를 사용하는 이중 적분이 가능. `[index4]`의 값이 높을수록 댐핑이 빨라짐
16. 개별 에러 레지스터에 의한 단일 레지스터 위치 읽기를 제공하기 위해 엔코더 변환 테이블에 새로운 타입 12 메소드를 추가. 주로 시리얼 엔코더 프로토콜을 고려함
17. `[MaxDelta]`의 리미트를 초과한 경우에 또는 타입 12 에러가 검출된 경우에 서보 사이클에서 비트 0 이 자동으로 1 로 설정되는 새로운 `[EncTable[n].Status]`를 추가. `[Motor[x].pEncLoss]`는 이 레지스터를 사용 가능
18. 비트 2(값 4)를 `[Motor[x].PhaseMode]`에 추가. 1 로 설정하면 2 개의 ASIC 채널을 사용하는 다이렉트 PWM 모드에서 3 레벨 앰프 단계의 제어가 가능
19. MACRO 슬레이브로 동작 중일 때 하드웨어 포착 위치를 MACRO 마스터에 통지하는 기능을 추가
20. 구성 요소로서의 부분 요소에 분할된 복합 폴 요소에 따라 백업/저장에 새로운 에코 모드를 추가. 에코 명령의 비트 6(값 64)을 1 로 설정하면 유효로 설정 가능
21. 프로그램의 **read** 명령에 로컬 변수 'D54' 및 'D0'의 문자 값을 정확하게 읽을 수 있도록 새로운 비트 필드를 추가. **read** 다음에 'D0'가 명령에 따라 사용 가능해지기 전에 다른 태스크에 의해 덮어쓰지는 것을 방지하기 위해 사용 가능
22. 전류 자동 Null 프로세스 중의 최대 허용 속도를 `[Motor[x].MaxSpeed]/10,000`에서 `[Motor[x].MaxSpeed]/1,000`으로 증가
23. 2D 보정 테이블의 데이터 포인트로 사용할 수 있는 로컬('L') 변수 번호의 범위를 L0~L1에서 L0~L1023으로 확장. 3D 보정 테이블의 데이터 포인트로 사용할 수 있는 로컬 변수 번호를 최초의 2 개 인덱스에 대해서는 L0~L2에서 L0~L31로 3 개째의 인덱스에 대해서는 L0~L63으로 확장. 이러한 테이블의 최대 상수 인덱스값을 약간 축소
24. 클로즈드 루프 내이지만 앰프 Disable 상태인 모터에 이상이 발생할 가능성을 배제(상태 이동하는 동안 발생하는 인터럽트에 따라 일어날 가능성이 있음)
25. 좌표계의 폴트 공유 로직을 수정하여 모션 프로그램을 실행하고 있지 않을 때 모터에 폴트(오버트래블 리미트, 앰프 폴트 등)가 발생해도 좌표계 내의 다른 모터에 영향을 주지 않도록 함
26. `[Coord[x].AltFeedrate]`의 기본값을 0.0 에서 1.0 으로 변경하고 이송 속도 동작 중의 속도 축이 비벡터 축과 비교해서 거리가 작거나 거리가 없는 경우에 대체 이송 속도 제어를 기본값으로 사용 가능
27. 서보 및 페이즈 위치 양쪽에 대해 전원 셋업 시의 자동 절대 위치 읽기 동작을 수정(`[Motor[x].PowerOnMode] = 6`)
28. `[Motor[x].AuxFaultLevel]`을 1 로 설정한 경우의 보조적인 폴트 검출 동작을 수정
29. 비활성 모터에 제어 명령을 발행한 경우에 에러 메시지를 통지하도록 수정
30. 프로그램을 목록화할 때 'C(i)', 'D(i)' 및 'R(i)' 배열의 변수 형식을 통지하도록 수정
31. 한 번도 실행되지 않은 프로그램에 대한 **list apc** 명령에 대한 응답을 수정

- 32. 정확히 $7\pi/4$ 라디안으로 라디안을 사용하는 삼각함수의 동작을 수정
- 33. Rapid 동작 모드에서 spline 동작 모드로의 직접 이동을 수정

V1.6 펌웨어 갱신(2014 년 2 월)

1. 싱글코어 및 듀얼코어의 새로운 Power PC 465 CPU 지원을 추가
2. 데이터 구조체 [CamTable[m]]을 사용하는 캠 테이블 기능을 추가
3. 새로운 부울형 저장 설정 요소 [Motor[x].PosReportMode]에 의한 오프셋 컴포넌트(마스터 위치 오프셋 모드, 캠 테이블 위치 오프셋)가 있는 경우에 모터 위치의 통지 방법 옵션을 추가
4. 새로운 저장 설정 요소 [Motor[x].pAuxFault], [Motor[x].AuxFaultBit], [Motor[x].AuxFaultLevel] 및 [Motor[x].AuxFaultLimit]에 의한 각 모터의 추가 폴트 비트 기능을 추가. 보조 폴트 기능은 엔코더 소실 기능과 동일한 내용. 특히 2 차 엔코더 소실 또는 모터의 열 폴트에 사용 가능
5. 새로운 저장 설정 요소 [Sys.pAbortAll], [Sys.AbortAllBit], [Sys.AbortAllLimit] 및 [Coord[x].AbortAllMode]에 의한 글로벌 Abort 입력 기능을 추가
6. 새로운 좌표계 **adisable** 명령(온라인 및 프로그램)에 의한 하이브리드 Abort 모드 기능을 추가. 좌표계 내의 모든 모터는 **abort** 명령의 경우와 동일한 방법으로 최초로 제어 정지로 이행. 각 모터가 명령 속도 제로 상태에 도달하면 모터에 지연 Disable 이 실행되고 사용 중인 경우에는 즉시 브레이크가 적용되어 [Motor[x].BrakeOnDelay] 시간이 경과한 후 출력 정지
7. Abort 감속 프로파일이 실행되는 최소 타임 베이스 비율을 지정하는 새로운 저장 설정 요소 [Coord[x].AbortTimeBase]를 추가
8. 새로운 저장 설정 요소 [Sys.MotorsPerRtInt]에 따라 모터의 상태 및 안전 갱신을 확인하는 빈도에 유연성을 부여. 기본값 0 에서 변경한 경우(하위 호환), [Motor[x].BISlewRate], [Motor[x].BrakeOffDelay], [Motor[x].BrakeOnDelay], [Motor[x].PhaseFindingTime] 및 [Motor[x].I2tTrip]의 설정을 변경하여 호환 운동을 유지해야 함
9. 새로운 저장 설정 요소 [Motor[x].pPhaseLoadEnc], [Motor[x].PhaseLoadEncLeftShift] 및 [Motor[x].PhaseLoadEncRightShift]를 추가하여 페이즈 인터럽트 시에 서보 루프를 닫은 경우에 듀얼(모터/로드) 피드백이 가능해짐
10. 모션 프로그램의 싱글 스텝 조작을 더 유연하게 실행할 수 있도록 [Coord[x].StepMode]를 추가. 기존의 싱글 동작 모드(= 0)에 더해 CNC 스타일의 싱글 행 모드(= 1) 및 디버그 모드(= 2, 3)를 지원
11. [Sys.SimConfigOk]를 추가하여 에러를 발생시키지 않고 존재하지 않는 하드웨어를 지정하는 주소 지정에 의한 설정 요소의 값을 다운로드하는 기능을 추가. 이로써 축소된 시스템 및 시뮬레이션 중인 시스템에서 프로젝트 구성을 사용할 수 있게 됨
12. 시스템의 재초기화 시에 클록을 생성하지 않는 ASIC 가 발견된 경우에 [Sys.CpuTimerIntr]이 자동으로 1 로 설정되고 프로세서가 내부에서 독자적으로 인터럽트를 생성. 따라서 EtherCAT 등의 네트워크 및 시뮬레이션 모드 사용 시에 ASIC 를 사용하지 않아도 Power PMAC 을 가동 가능
13. 새로운 저장 설정 요소 [Coord[x].CornerError]에 따라 프린트되지 않은 코너 포트와 관련해서 블렌드된 경로의 코너에서의 편차를 지정할 수 있는 기능을 추가

14. 새로운 비저장 설정 요소 [Coord[x].TxyzScale]에 따라 XYZ 직교 공간이 변환 매트릭스에 따라 다시 스케일링된 경우에 스케일링되지 않은 이송 속도 및 2D 공구 보정 반경을 유지하는 기능을 추가. 새로운 저장 설정 요소 [Coord[x].AutoTxyzScale]의 제어 비트에 따라 변환 매트릭스의 3x3 XYZ 부분에 기반하여 [TxyzScale]을 자동 계산
15. 새로 추가한 Status 비트 [Motor[x].SoftLimitDir]에 따라 동작의 정지, 변경 또는 거부(0 은 양의 리미트, 1 은 음의 리미트)의 원인이 된 소프트웨어 오버트래블 리미트를 식별 가능
16. 새로운 저장 설정 요소 [Sys.MaxTimedUnderflow]에 따라 상정되지 않은 과도한 음의 타임 베이스에 대한 보호를 제공. 임의의 모터의 축적된 음의 타임 베이스가 이 시간을 초과하여 저장되어 있는 식 범위의 최초로 복귀하면 모션이 이상 정지
17. 음의 타임 베이스 값일 때는 모터 트레이스 버퍼의 과거의 모터 궤적 정보를 버퍼링하여 역행할 수 있는 기능을 추가. 버퍼 크기는 새로운 저장 설정 요소 [Motor[x].TraceSize]에 따라 설정
18. 타임 베이스 변경의 슬루 레이트를 자동으로 제한하여 모터 가속 리미트에 위반되지 않게 하는 기능을 Power PMAC 에 추가. 이 모드는 [Coord[x].TimeBaseSlew] 및 [Coord[x].FeedHoldSlew]에 음이 값을 설정하면 유효로 설정됨
19. [Motor[x].EncType]의 새로운 값 7 과 확장 하드웨어 보간(65,536x 의 보간)용의 새로운 엔코더 변환 테이블 메소드([EncTable[n].type] = 7)에 따라 새롭게 확장 분해능의 자동 수집 사인파 보간 보드를 지원
20. [Sys.NoShortCmds]에 따라 단일 문자의 온라인 액션 명령을 금지하는 기능을 추가. 이 모드에 따라 이러한 명령을 의도치 않게 발생할 가능성이 감소(이 모드는 실제로는 V1.5 펌웨어에서 추가)
21. 비저장 설정 제어 요소 [Motor[x].CapturePos] 및 Status 요소 [Motor[x].CapturedPos](모터 단위)에 따라 새로운 위치 포착 모니터링 기능을 추가
22. 새로운 저장 설정 요소 [CompTable[m].SourceCtrl]에 따라 수정 계산에서 명령 위치 대신 소스 모터의 현재 위치를 사용하는 보정 테이블의 기능을 추가
23. 가변 시간 'spline' 모드에서 새로운 부울형 저장 설정 요소 [Coord[x].SplineTimeRotate]를 사용하여 Turbo PMAC 과 동일한 동작이 가능해짐
24. 새로운 요소 [Motor[x].pMotorNode], [Motor[x].MotorNodeOffset] 및 [Motor[x].MotorMode]에 따라 Power PMAC 모터를 MACRO 슬레이브로 실행할 수 있는 기능을 추가
25. [Motor[x].Ctrl] = [Sys.GantryXCtrl]을 사용 중인 모터가 각 모터에 적응 제어 계인을 사용할 수 있게 하고 크로스 커플링 및 적응 서보 제어 알고리즘을 조합하는 기능을 추가([GantryXCtrl] 알고리즘의 기존 사용자가 제로와 동등하지 않은 미사용 적응 계인을 가지게 되는 발생할 가능성이 적은 상태에서는 이 변경에 따라 완전한 후방 호환성을 얻을 수 없는 경우가 있음)

26. [Motor[x].Servo.MinW], [Motor[x].Servo.MaxW], [Motor[x].Servo.MinDR] 및 [Motor[x].MaxDr]에 따라 클로즈드 루프의 고유 주파수(W) 및 검출된 관성의 범위에서의 감쇠율(DR)의 제어 변화를 가능하게 하여 적응 서보 알고리즘의 유연성을 강화
27. 데이터 구조체 [BrickAC] 및 [BrickLV], 데이터 구조체 [Gate3[i]]의 별칭 [PowerBrick[i]] 및 온라인 쿼리 명령 **brickacver**, **bricklvver**에 따라 새로운 Power Brick 제품을 지원
28. 데이터 구조체 [Acc72EX[i]]가 추가되어 새로운 ACC-72EX 필드 버스 인터페이스 보드를 지원
29. 블록 레이트가 아주 높은 어플리케이션 계산에 견고함을 더 강화. 다음 프로그램된 동작 블록의 계산 요구에 따라 버퍼 오버플로의 가능성이 있는 경우에 즉시 런타임 에러를 발생시키지 않고 실시간 인터럽트가 1 회 지연. 새로운 Status 요소 [Coord[x].BufferWarn]에 따라 이 상태의 발생을 나타내며 이러한 어플리케이션의 [Sys.PreCalc] 변수의 설정이 최적화
30. 모션 프로그램 로터리 버퍼에서 동작 계산이 최후의 동작에 도달하거나, 고정 모션 프로그램 플로가 [Coord[x].GoBack]+1 회를 초과하여 점프 백했을 때의 액션을 수정. 실행을 일시 정지할 수 있으므로 자동으로 재개 가능(프로그램 실행이 중지된 후)
31. 좌표계의 타임 베이스가 정확히 0.0 인 경우에 계산이 완료된 동작 방정식을 큐에서 활성화된 실행 레지스터로 로드 가능. 이로서 트리거에 의한 타임 베이스 시작의 위치 참조가 더 정확해짐
32. [Coord[x].Ta]의 기본값을 0.0 에서 100.0(밀리초)으로 변경. [Coord[x].Td]의 기본값을 0.0 에서 100.0(밀리초)으로 변경. [Coord[x].Ts]의 기본값을 0.0 에서 50.0(밀리초)으로 변경. 제어된 동작의 작성이 쉬워짐
33. 자동 검출 하드웨어 인터페이스 채널에 할당되지 않은 모터의 재초기화 시에 모터의 주소 지정 변수 [Motor[x].pLimits] 및 [Motor[x].pAmpFault]가 0 으로 설정되게 함. 이로서 이러한 모터의 하드웨어 오버트래블 리미트 및 앰프 폴트 체크가 무효가 되어 가상 모터의 설정이 쉬워짐
34. 스크립트 PLC 프로그램에서 모달 명령(pvt, spline, ta, tm, ts, td 및 F)을 무효로 설정(no-ops). 이러한 명령을 PLC 프로그램에 입력할 수는 있지만 실행해도 아무 것도 동작하지 않음. 이러한 명령이 영향을 주는 데이터 구조체 요소에 직접 이러한 명령을 써서 동작 가능
35. 수치 적분이 실행되기 전에 소스 위치에 추가되는 오프셋 조건을 새로운 저장 설정 요소 [EncTable[n].EncBias]에 따라 저장하도록 함. 이전에는 이 목적에 사용되었던 Status 요소 [EncTable[n].PrevDelta]와 레지스터를 공유

V1.5 펌웨어 갱신(2012 년 6 월)

1. 보정 테이블의 사용 가능한 대상 레지스터로서 새로운 요소 [Motor[x].CompDesPos]를 추가. 궤적 및 마스크 위치와 함께 순수 명령 위치의 컴포넌트를 제공. 테이블을 전자 캡 테이블로 사용 가능
2. 프로그램에서 [Ldata.Control]을 2 로 설정하면 Script 의 속도 및 매트릭스 함수의 사용자 공유 메모리 버퍼 배열을 사용 가능
3. 온 더 플라이 알고리즘으로 블렌드하는 JOG 스타일(jog, home, rapid) 동작을 확장
4. Compact UMAC 액세스리 보드 ACC-11C, ACC-24C2, ACC-24C2A, ACC-51C 의 자동 검출을 추가
5. 새로운 요소 [Sys.FltrPhaseTime], [Sys.FltrServoTime], [Sys.FltrRtiTime] 및 [Sys.FltrBgTime]의 결과에 따라 필터링된(동작 평균) 태스크의 계산 시간의 자동 계산을 추가
6. 새로운 비저장 설정 요소 [Coord[x].InvTimeMode]에 따라 직선 보간 모드 및 원호 보간 모드의 동작에 역수 모드의 프로그래밍을 추가
7. 새로운 저장 설정 요소 [Coord[x].HomeRequired]를 추가. 1 로 설정하면 좌표계 내의 축에 할당된 모든 모터는 모션 프로그램이 좌표계에서 실행할 수 있게 하기 위해 기준 위치를 확립해야 함
8. 타입 11 의 엔코더 변환 테이블 항목을 추가하여 부동 소수점 레지스터 [Motor[x].IqCmd]를 유사 피드백으로 처리 가능. 이 메소드는 오픈 루프의 다이렉트 마이크로 스텝핑의 가상 모터 및 가상 피드백을 지원
9. 사인파 엔코더 변환(타입 4)용으로 새로운 보정을 위한 요소 [EncTable[n].CoverSerror] 및 [EncTable[n].TanHalfPhi]를 추가. 새로운 요소 [EncTable[n].index5]를 추가하여 사인파 엔코더에 대한 DSPGATE3 인터페이스에서 소프트웨어에 의한 아크탄젠트 변환이 가능
10. 새로운 저장 설정 요소 [Motor[x].CurrentNullPeriod]에 따라 전류 피드백 자동 Null 기능을 추가. 모터를 유효화하면 자동으로 [Motor[x].IaBias] 및 [Motor[x].IbBias] 항목을 ON 으로 설정
11. 에코 모드 파라미터에 새로운 비트 5(값 32)를 추가. 1 로 설정하면 비활성 모터 및 좌표계를 포함한 모터 및 좌표계가 완전히 백업되어 다른 구성과 비교하는 데 사용 가능
12. [Motor[x].AbsPosFormat] 및 [Motor[x].AbsPhasePosFormat]의 새로운 설정을 사용해서 MACRO 링 상에서 절대 위치 데이터를 취득할 수 있는 기능을 추가
13. [Motor[x].AbsPosFormat]의 새로운 설정을 사용하면 최초 및 2 번째의 레지스터 간에 간격이 있는 데이터에서 연속된 레지스터의 절대 위치 데이터를 사용할 수 있는 기능을 추가
14. 인포지션 확인 기능에서 무한 JOG 이송 및 원점 서치 동작의 정속 부분에서 인포지션을 True 로 통지하지 않도록 수정

15. 위치 추종 간의 소프트웨어 오버트래블 리미트 확인을 수정. 저장 설정 요소 [Motor[x].SoftLimitOffset]을 추가하여 계산 시간에 의한 소프트웨어 리미트와 실행 시간의 소프트웨어 리미트의 차를 허용
16. 매우 짧은 스플라인 동작으로 버퍼 오버플로를 방지하도록 계산을 수정
17. 서보 출력 불감대의 탑재를 수정. 적분기의 'seed' 부호를 변경하여 내측 불감대 리미트와 외측 불감대 리미트 간의 위치 편차에 따라 명령 속도 제로 상태에 도달했을 때의 적분기 시딩(seeding)을 추가

V1.4 펌웨어 갱신(2011 년 9 월)

1. 저장 설정 요소 [Sys.BgSleepTime]을 추가하여 독립적인 C 어플리케이션의 백그라운드 사이클 간의 합계 시간을 밀리초로 지정 가능해짐. 이전에는 1.0 밀리초로 고정되어 있었음
2. 모터의 스핀들 축 정의(#x->S, #x->S0, #x->S1)를 추가하여 좌표계 내의 다양한 번호의 위치 결정 모터의 록 어헤드 버퍼를 삭제 및 재정의하지 않고 회전 위치 결정 축과 속도 모드 스핀들 간에서 모터를 변경 가능. Status 요소 [Motor[x].SpindleMotor] 및 [Coord[x].LHMotorSlots]를 추가하고 이에 관련된 모터 및 좌표계의 현재 Status 를 저장 가능해짐
3. 저장 설정 요소 [Motor[x].PwmDbComp] 및 [Motor[x].PwmDbI]을 추가하여 PWM 에 필요한 데드 타임을 위한 현재의 불감대를 다이렉트 PWM 제어 모드에서 보정 가능해짐
4. 새로운 서브 구조체 [GateIo[j].Init]에 저장 설정 요소를 추가하여 I/O 자체의 동작에 영향을 주지 않고 설정을 저장 가능해짐
5. ACC-53E UMAC SSI 엔코더 인터페이스 보드용 데이터 구조체 [Acc53E[j]]를 추가
6. Status 요소 [PartType]를 액세서리 데이터 구조체 [Gate1[j]](및 별칭), [Gate2[j]](및 별칭), [GateIo[j]](및 별칭), [Acc28E[j]], [Acc36E[j]], [Acc59E[j]], [Acc72E[j]], [Acc84E[j]] 및 [DPR[j]]에 추가하여 설정 유틸리티의 편의성이 향상
7. 사용자 공유 메모리 버퍼에 비저장 데이터 구조체 요소 [Sys.Uhex[j]]를 추가. 쿼리하면 값이 16 진법으로 통지되는 것 이외에는 [Sys.Udata[j]]와 동등
8. 크로스 커플링된 갠트리 서보 제어 알고리즘을 추가. [Motor[x].Ctrl]을 새로운 Status 주소 요소 [Sys.GantryXCtrl]에 설정하면 선택 가능. 새로운 저장 게인 항목 [Motor[x].Servo.Kxpg], [Motor[x].Servo.Kxig] 및 [Motor[x].Kxvg]를 추가
9. Status 비트 [Coord[x].ProgProceeding]을 추가. 피드 홀드에 따라 정지된 경우에 0 이 되는 것 이외에는 기존의 Status 비트 [Coord[x].ProgRunning]과 동등
10. 데이터 구조체 요소에서 단항 부정 연산자로 '-'를 사용 가능해짐
11. 32 비트 I/O 요소 [Gate3[j].GpioData[j]]에 비트 필드 액세스를 추가. [Gate3[j].GpioData[j].m.n]은 요소 내의 비트 'm'으로 시작하는 'n' 비트의 사용을 지정
12. [Motor[x].PhaseMode]에 새로운 비트(비트 1, 값 2)를 추가. 이것을 설정하면 직류 전류 루프의 적분기를 무효로 설정하여 DC 브러시 모터의 다이렉트 PWM 제어가 쉬워짐
13. 'PosCtrl' 서보 알고리즘의 32비트 위치 출력이 롤 오버 가능해짐
14. 커널 내의 시리얼 드라이버에 USB 에서 RS232 컨버터로의 변환 기능을 추가
15. 비디오 드라이버 소프트웨어에 모니터 해상도의 자동 검출 기능을 추가. 이전에는 1024x768 로 고정되어 있었음
16. 전원 셋업 시에 설치되어 있는 USB 메모리 스틱에 네트워크 설정을 쓰는 기능을 추가. 통신 링크가 없어도 Power PMAC 의 IP 주소 검출이 가능

17. 소프트웨어에 의한 오버트래블 리미트 **Status** 비트의 투과적인 계산을 변경. 모터의 현재 위치가 아닌 명령 위치에 따라 좌우되도록 변경하여 리미트에 따라 정지했을 때의 토글을 방지
18. 독립적인 C 어플리케이션이 **\$\$\$** 또는 **\$\$\$***** 명령을 통해 반드시 자동으로 정지
19. **\$\$\$***** 재초기화 후에 공장 출하 시의 기본 설정이 포함되는 디렉토리 'ftp://usrflash/Project/Configuration'에 자동으로 파일 [pp_default.cfg]가 작성되게 함
20. ACC-24E2x 직교 엔코더 소실 비트에서 [Gate1[j].Chan[j].EncLossN.a] 또는 동등 주소 [Cid[n].PartData[j].a]에 설정된 경우의 주소 요소 [Motor[x].pEncLoss]의 통지를 수정
21. 회전축에서의 롤 오버 모드의 부호에 의한 방향의 동작을 수정 ([Coord[x].PosRollover[j] < 0)
22. gpascii 유틸리티에 의한 무효 시스템 명령의 에러 포착을 수정
23. 실시간 커널 세마포어 작성에서의 메모리 누설을 수정
24. 저장 소프트웨어 구조체 [Gatelo.Init]를 추가하여 전원 셋업/리셋 시에 하드웨어의 설정 요소 [Gatelo]의 자동 구성이 가능해짐

V1.3 펌웨어 갱신(2011 년 1 월)

1. 위치 추종 기능의 확장: 새로운 저장 설정 요소 [Motor[x].SlewMasterPosSf]에 따라 추종 기어비를 자동으로 단계적으로 변경 가능. 새로운 저장 설정 요소 [Motor[x].MasterMaxSpeed] 및 [Motor[x].MasterMaxAccel]에 따라 추종 속도의 자동 포화가 가능. 포화가 발생한 경우에는 속도 또는 위치에 대한 재동기화가 가능한 옵션을 제공
2. 새로운 저장 설정 요소 [Motor[x].pEncLoss], [Motor[x].EncLossBit], [Motor[x].EncLossLevel] 및 [Motor[x].EncLossLimit]에 따라 자동으로 엔코더 소실 검출 및 결과의 섀다운 유효화가 가능. 엔코더 소실 장애는 편차 이상 폴트 또는 앰프 폴트로 처리
3. 기존의 저장 설정 요소 [Motor[x].FaultMode]에 새로운 비트 2(값 4)를 추가하여 하드웨어 리미트에 도달하면(소프트웨어 리미트에는 도달하지 않음) 모터를 클로즈드 루프 정지(Abort)하는 대신 모터를 무효(출력 정지)화할 수 있는 기능을 추가. 이 모드는 보통 어떤 피드백 에러에 따라 하드웨어 리미트에 발생한 의도하지 않은 일탈을 처리하는 수단으로 권장
4. 새로운 저장 설정 요소 [Coord[x].TPSize] 및 [Coord[x].TPCoords]에 따라 축의 대상 위치의 복수 동작 버퍼를 탑재. 대상 위치의 통지에는 새로운 온라인 명령 **t** 및 새로운 버퍼 프로그램 명령 **tread**, 남은 이동량의 통지에는 새로운 온라인 명령 **g** 및 새로운 버퍼 프로그램 명령 **dtogread** 를 사용
5. 버퍼 프로그램의 축 쿼리 명령 **dread**(명령 위치), **fread**(위치 편차), **pread**(현재 위치) 및 **vread**(필터링된 현재 속도)를 탑재. 값은 로컬 프로그램용 'D' 변수 배열에 반환
6. 사용자 공유 메모리 내의 문자 배열 [Sys.Cdata[i]], 문자 배열 조작 함수 **memset** 및 **memcpy**, 문자열 변수 조작 함수 **sprintf**, **strcpy**, **strncpy**, **strtolower**, **strtoupper**, **strcat**, **strncat**, **strchr**, **strchr**, **strcmp**, **strncmp**, **strspn**, **strcspn**, **strlen**, **strbrk**, **strstr** 및 **strtod** 에 따라 문자열 변수 및 문자열 조작 기능을 추가. 기존의 프로그램 버퍼 명령 **cmd**, **send** 및 **system** 을 통해 문자열 변수를 사용 가능
7. 기존의 저장 설정 요소 [Motor[x].Ctrl]에는 새로운 설정 [Sys.AdaptiveCtrl]을 추가하고 새로운 저장 설정 요소 [Motor[x].EstTime], [Motor[x].EstMinDac], [Motor[x].Servo.NominalGain], [Motor[x].Servo.MinGainFactor] 및 [Motor[x].MaxGainFactor]를 추가해서 관성을 변경하고 시스템용 자동 적응 서보 제어 알고리즘을 추가
8. 기존의 저장 설정 요소 [Motor[x].CaptureMode]에 새로운 설정 3 을 추가. 트리거 장착 동작에 타이머 보조 소프트웨어 포착 기능을 탑재하여 ASIC 카운터에서 처리되지 않는 피드백 타입의 포착 정확성이 향상. 트리거(위치 피드백이 아님)는 PMAC3 스타일 DSPGATE3 ASIC 에서 처리되어야 함. 이는 해당 ASIC 가 서보 사이클 위치 간의 보간에 ASIC 내의 타이머 회로를 사용하기 때문. 새로운 저장 설정 요소 [Motor[x].ServoCaptTimeOffset]을 사용하여 보간 타이밍이 적절하게 설정

9. 실시간 C 루틴 'CaptComplSR'을 사용해서 PMAC3 스타일 DSPGATE3 ASIC 에서 생성되는 인터럽트에 C 언어에 의한 사용자 지정 인터럽트 루틴을 사용자가 탑재할 수 있는 기능을 추가. 루틴은 새로운 저장 설정 요소 [UserAlgo.CaptComplIntr]을 1 로 설정하면 유효로 설정됨
10. 새로운 저장 설정 요소 [Motor[x].pBufPos] 및 [Motor[x].pBufPos2]에 따라 최대 8 개의 서보 루프의 현재 위치 레지스터의 로터리 버퍼를 확장하는(256 서보 사이클) 기능을 추가
11. 새로운 저장 설정 요소 [Motor[x].PhaseEncRightShift] 및 [Motor[x].PhaseEncLeftShift]에 따라 전류(commutation) 위치 피드백 데이터의 시프트 조작을 실행할 수 있는 기능을 추가. 그러면 레지스터의 비트 31 에 피드백 데이터의 MSB 를 갖지 않는 레지스터 및 하위 비트에 가비지 데이터를 갖는 레지스터를 전류(commutation) 피드백에서 사용 가능
12. 여러 UMAC 액세서리의 하드웨어 레지스터에 대한 다이렉트 액세스를 실시하기 위한 데이터 구조체 [Acc59E[i]], [Acc59E3[i]], [Acc72E[i]] 및 [Acc84E[i]] 를 탑재함