1. UMAC(PMAC)에서 DPRAM 사용의 일반적인 사항

UMAC에서 DPRAM 옵션을 이용하면 보다 빠른 통신 속도로 Host PC와 데이터를 주고 받을 수 있습니다. 일반적으로 초당 100개 이상의 데이터를 주고 받을 필요가 있을 때에 DPRAM을 사용하면 효율적입니다.

UMAC에서 DPRAM 옵션 사용시에 제공되는 DPRAM은 32K x 16 Bits 입니다. UMAC의 데이터 버스는 24Bit 이므로, 실제로는 하위 16Bit만을 DPRAM에 접속해서 사용하게 됩니다. 이렇게 해서 X,Y (16 Bit Data) 또는 DP(32Bit Fixed Point), F(32Bit Floating Point)를 통해서 M변수에 맵핑해서 사용합니다. 반면 PC(HOST) 쪽에서는 통신 CPU를 통해서 해당 DPRAM영역에 접근하게 되는데, 16Bit 데이터 버스를 사용하고 있습니다.

아래 그림을 참고하시기 바랍니다.

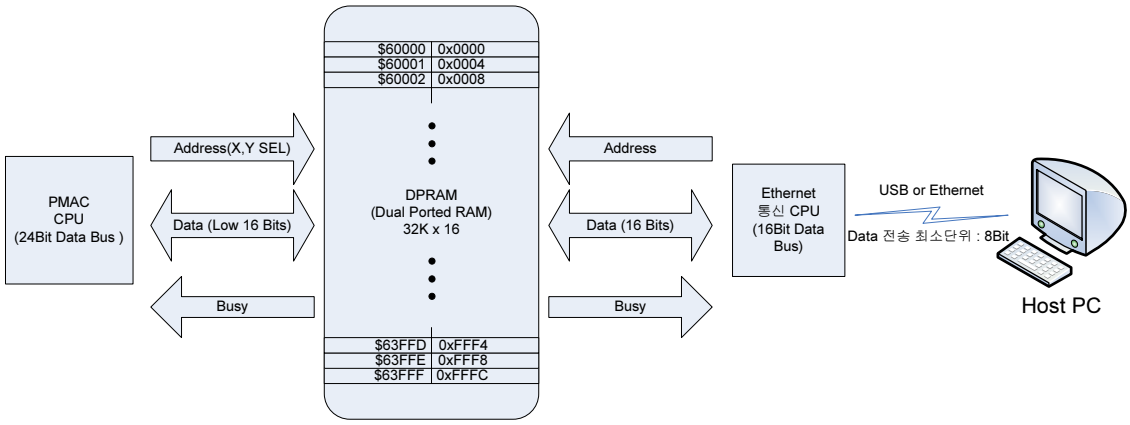

그림1. UMAC(PMAC)의 DPRAM 구조

DPRAM 사용 시에는 PMAC CPU의 통신 버퍼를 거치지 않고 통신 CPU에서 바로 데이터를 처리해서 HOST PC에 전달해 주기 때문에 통신 속도가 훨씬 빨라지게 되는 것입니다.

위의 구조에서 볼 수 있듯이 PMAC에서는 DPRAM에 최대 32Bit 정수 또는 실수로 맵핑해서 사용할 수 있으며, 이때 하나의 데이터를 쓰거나 읽기 위해서는 16Bit 데이터를 두 번 처리하는 구조로 되어 있습니다. 또한 통신 CPU에서 32Bit 정수 또는 실수를 처리하기 위해서는 데이터 버스가 16Bit이기 때문에 마찬가지로 두 번 접근해서 처리해야 합니다.

이러한 점(두 번 접근)을 감안하더라도, PMAC CPU에서 Parsing(명령 해석)하는 부분이 생략되기 때문에 속도는 상당히 빨라지게 됩니다.

2. 일반적인 DPRAM 사용 방법

PMAC CPU의 데이터 버스는 24Bit이며, X,Y 레지스터를 선택하게 되어 최대 48Bit의 데이터를 처리할 수 있습니다. 그러나 앞서 밝힌 바와 같이 DPRAM은 16Bit 데이터만 인터페이스 할 수 있기 때문에, 최대 32Bit의 데이터를 하나의 PMAC Address에서 처리할 수 있습니다. 따라서, PMAC 에서의 Address 하나는 PC(HOST)에서 4개의 Address를 가지게 됩니다. (그림1 참고, PC에서의 데이터 전송 최소단위가 8Bit이므로…)

이에 따라서 PMAC에서는 다음과 같은 형식으로 DPRAM을 사용할 수 있습니다.

```
M4000->DP:$60450      ; 32Bit 정수로 사용(부호사용)
M4001->F:$60451       ; 32Bit 실수로 사용
M4002->Y:$60452,0,16  ; 16Bit 정수로 사용(부호없음)
M4003->X:$60452,0,16  ; 16Bit 정수로 사용(부호없음)
M4004->Y:$60452,0,16,S    ; 16Bit 정수로 사용(부호사용)
M4005->X:$60452,0,16,S    ; 16Bit 정수로 사용(부호사용)

M4006->Y:$60453,0,1   ; 1Bit Flag로 사용 (Bit0)
…
M4021->Y:$60453,15,1  ; 1Bit Flag로 사용 (Bit15)
```

그림2. DPRAM영역 맵핑해서 사용하기

DPRAM 영역을 위와 같이 다양한 방법으로 맵핑해서 PLC나 모션프로그램 등에서 사용하실 수 있습니다. 특히, "DP", "F" 등으로 맵핑하게 되면, PC에서 바로 해당 영역의 데이터를 32비트 정수 또는 실수로 읽어서 처리할 수 있습니다. PC에서는 아래와 같은 함수들을 이용해서 DPRAM에 쓰기/읽기 동작을 수행합니다.

```
DeviceDPRSetMem(deviceNo, offset, size, pointer);
        - pointer가 가리키는 size만큼의 데이터를 쓴다.
DeviceDPRGetMem(deviceNo, offset, size, pointer);
        - size만큼의 데이터를 pointer가 가리키는 곳에 가져온다.
DeviceDPRWordBitSet(deviceNo, offset, bitNo);
        - 해당offset의 bitNo 번째 Bit가 On인지 확인한다.
DeviceDPRSetDWordBit(deviceNo, offset, bitNo);
        - 해당offset의 bitNo 번째 Bit를 On 시킨다.
DeviceDPRResetDWordBit(deviceNo, offset, bitNo);
        - 해당offset의 bitNo 번째 Bit를 Off 시킨다.

Ex)
  ; PMAC
  M4000->DP:$60450
  M4000 = $12345678
  ; PC
  DWORD dwData = 0;
  DeviceDPRGetMem(0, 0x450*4, sizeof(DWORD)*1, &dwData);
  ; 실행 후에 dwData = 0x12345678 이 됩니다.
```

그림3. PC에서 DPRAM 함수 사용하기

위의 예제에서 보시는 바와 같이 PMAC에서는 DPRAM의 Base Address가 "$60000"(Old CPU에서는 "$6C000")이 됩니다. 따라서, PMAC에서 "$450"만큼의 Offset을 가지게 되면, PC에서는 "0x450 * 4"만큼의 Offset을 가지게 됩니다. 이는 앞서 설명한 데이터 버스의 크기 때문에 발생하는 것입니다. (PMAC : 32Bit DPRAM 데이터 버스, PC : 8Bit DPRAM 데이터 단위 전송).

또한 대량의 데이터를 한꺼번에 교환할 때에는 "DeviceDPRGet/SetMem" 함수를 사용할 수 있습니다. 이 함수들을 사용하면, 큰 용량의 메모리를 한꺼번에 Dump 할 수 있습니다. 이외에도 Bit Operation 관련 함수들을 사용해서 간단하게 DPRAM Flag 통신을 구현하실 수 있습니다.

3. DPRAM에서 Bit 연산을 사용할 때의 주의 사항

일반적으로 DPRAM을 통해서 정수 또는 실수의 데이터를 교환할 때에는 별다른 문제가 없습니다. 하지만, DPRAM으로 Bit 연산을 수행할 때에는 매우 주의가 요구 됩니다. 그러한 사항을 구체적으로 살펴보면 아래와 같습니다.

우선 정수(32Bit, 16Bit) 또는 실수 데이터를 사용할 때를 살펴보겠습니다. 실제 Bit 연산에서 주의가 요구되는 부분은 PMAC과 PC에서 동시에 같은 Address영역의 Data를 쓸 때 입니다. Bit 데이터가 아닌 16Bit 단위의 Access 시에는 별다른 문제가 없습니다. 아래는 일반적인 16Bit Access시의 흐름도입니다.

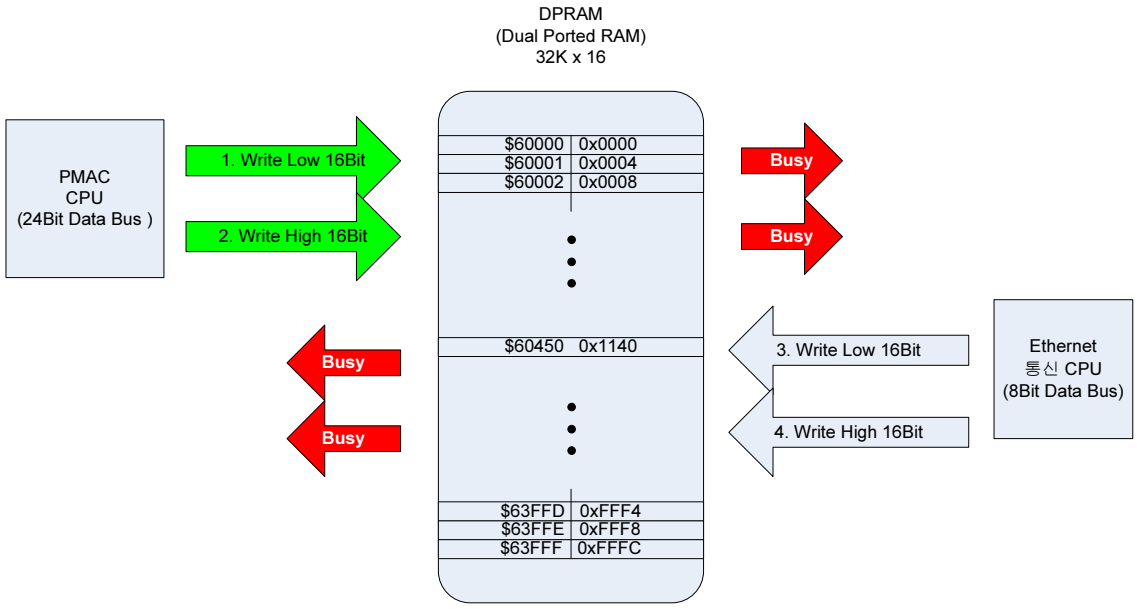

그림 4. 일반적인 DPRAM 접근시의 Busy 신호 발생

위 그림4 에서 보시는 바와 같이 일반적으로 16Bit 단위의 데이터를 Access할 때에는 서로 상대편 BUS쪽으로 Busy 신호를 내보내기 때문에 하드웨어적으로 동시에 데이터를 쓰는 경우는 발생할 수 없습니다.

이제, 문제가 될 수 있는 Bit 연산에 대해서 살펴보도록 하겠습니다. 우선, PMAC에서의 Bit연산은 Multi-Byte 데이터를 읽어서 해당 Bit만 Masking을 해서 다시 데이터를 쓰는 방식으로 수행할 수 있습니다. 그리고, 나머지 방법은 M변수에 정의할 때에 1Bit 크기만큼만 맵핑을 하는 것입니다. 아래는 두 가지 경우에 대해서 Bit 연산을 수행하는 부분을 설명하고 있습니다.

```
;;DPRAM $60450 어드레스의 Bit0 을 "1"로 설정하는 방법.

Method 1)
  M4000->Y:$60450      ; 16Bit 정수로 사용
  ; Bit 0 을 "1"로 설정할 경우
  P4000=M4000
  M4000=P4000|$0001
          or
  M4000=M4000|$0001

Method 2)
  M4001->Y:$60450,0,1   ; Bit0 을 맵핑하도록 정의함.
  ; Bit 0을 "1"로 설정할 경우
  M4001=1
```

그림 5. DPRAM의 특정 Bit를 설정하는 방법

위의 그림5 에서 보시는 바와 같이 Bit 연산을 하는 방법은 두 가지가 있습니다. 보기에는 Method 2가 훨씬 간단해 보입니다. 하지만, 내부적으로는 아래의 그림과 같이 Method 1과 같은 연산을 수행하고 있습니다.

```
;; Method 2) 의 내부적인 연산 순서

 1) Temp 에 $60450 어드레스의 Y 레지스터 값을 읽어서 저장
 2) 이 Temp 값과 Bit 연산을 수행
     Temp = Temp | $0001
 3) DPRAM의 $60450 어드레스의 Y레지스터에 Temp값 설정
```

그림 6. Method 2)의 내부 연산 과정

위 그림6 의 연산 과정에서 알 수 있듯이 PMAC CPU는 최소 3번 이상의 Instruction을 수행하게 되며, DPRAM에는 Bit 연산을 위해서 두 번 접근하게 됩니다. 이러한 과정은 Bit 연산 시에 통신 CPU에서도 마찬가지로 수행되게 됩니다.

그러면, 앞으로 이러한 Bit 연산을 수행 시에 문제가 발생하는 경우를 예를 들어 설명해 보도록 하겠습니다. 예에서는 DPRAM의 특정 Address의 여러 Bit들을 제어용 Flag Bit로 사용하는 것으로 가정하고 설명하도록 하겠습니다.

먼저 PMAC에서는 아래의 두 변수를 Flag용으로 사용하고 있으며, 각각 서로 다른 PLC에서 제어하고 있습니다.

```
M4000->Y:$60450,0,1      ; 제어 Flag 1 번
M4010->Y:$60450,7,1      ; 제어 Flag 2 번
```

위와 같은 Bit Flag를 사용할 때에, "M4000"을 PMAC에서 Write하고, "M4010"을 PC에

서 동시에 Write하는 경우가 생기게 되면, 문제가 발생합니다. 아래에 문제가 발생하는 경우의 Flow를 그림으로 설명하겠습니다.

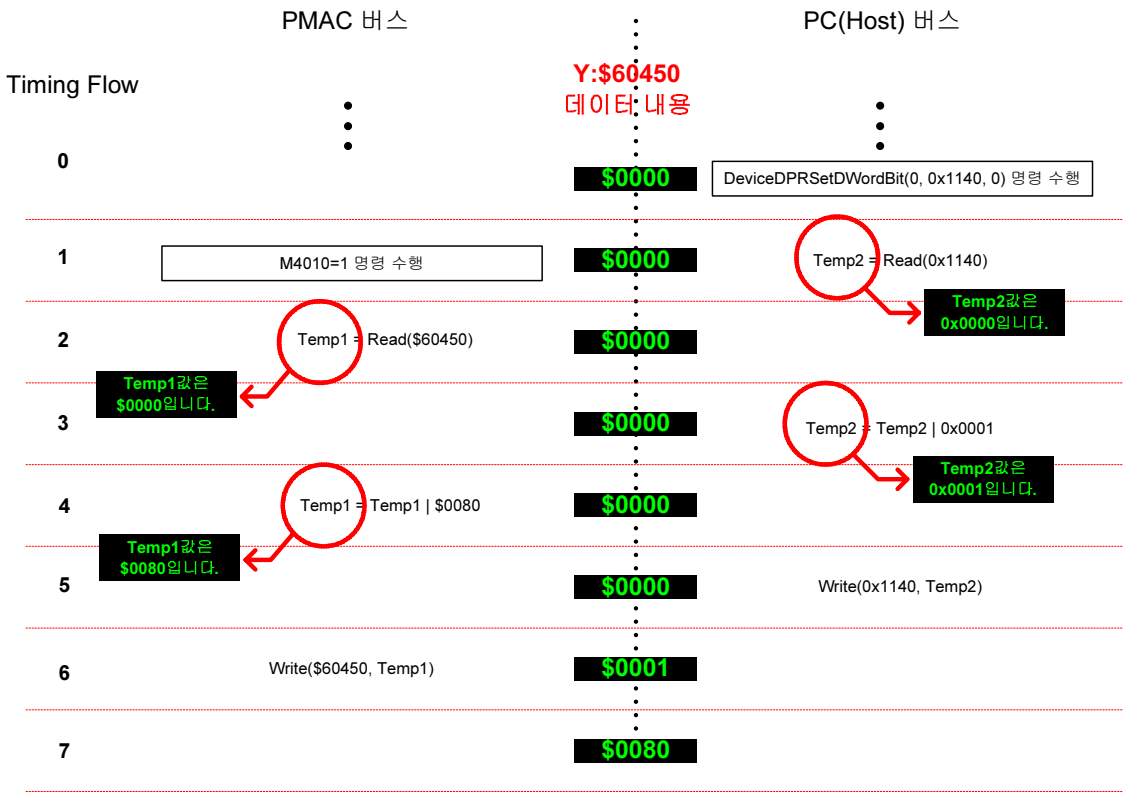

그림 7. 동시 Bit Write시의 Pseudo Timing Chart

위의 그림7 에서 보시는 바와 같이, 공교롭게도 "Y:$60450"의 데이터를 PMAC과 PC에서 동시에 Write하는 경우가 분명 발생할 수 있습니다. 물론, 이러한 경우가 발생하는 타이밍이 자주 있지는 않을 것입니다. (그러나, 분명 문제를 야기할 수 있습니다.)

위에서 보시면 Bit 연산을 하기 위해서는 우선은 해당 Address의 값을 먼저 읽어야 합니다. 그리고, 그 읽은 데이터에 Bit 연산을 수행한 다음, 결과를 다시 해당 Address에 Write하게 됩니다. 위와 같이 타이밍이 절묘하게 꼬이게 되면, PC에서 Bit0번을 설정하는 함수를 호출하게 되어도 순간적으로만 Bit0이 "1"로 되었다가 다음 CPU Cycle에는 바로 "0"으로 바뀌게 됩니다.

4. DPRAM Bit 연산 사용 시 발생하는 문제를 없애는 방법

앞서의 설명에서 보셨듯이, 같은 DPRAM Address를 PMAC과 PC에서 동시에 Write하는 경우만 피할 수 있다면, 문제는 발생하지 않을 것입니다. 따라서, 아래와 같은 여러 가지 방법으로 이 문제를 피해갈 수 있습니다.

a. Bit Flag로 사용할 M변수를 정의할 때에는 PMAC에서 Write하는 부분과 PC에서 Write하는 부분을 구분해서 서로 다른 어드레스를 지정하도록 합니다. (같은 어드레스라도 X/Y 레지스터를 구분하면 괜찮습니다.)

ex)

```
; PMAC -> PC Flag 정의
M4000->Y:$60450,0,1
M4010->Y:$60450,1,1
; PC -> PMAC Flag 정의
M4020->X:$60450,0,1
M4030->X:$60450,1,1
```

b. 같은 어드레스 영역의 Bit Flag를 사용할 경우, Write한 후에 반드시 Bit가 Set/Reset 되었는지를 확인하는 절차를 거칩니다. (이 방법은 별로 권장하지 않습니다.)

c. Bit Flag를 사용하지 않고, 16Bit 단위로 Flag를 맵핑해서 사용합니다.

ex)

```
; 16 Bit 단위로 M변수를 맵핑
M4000->Y:$60450,0,16
M4010->X:$60450,0,16
M4020->Y:$60451,0,16
M4030->X:$60451,0,16
        ...
```