보이지 않는 곳에서 세상을 편리하게 하는힘
오토닉스 센서 & 제어기기

Autonics
Sensors & Controllers

Motion Controller

PMC-4B-PCI

라이브러리 매뉴얼

PMC-4B-PCI

저희 (주)오토닉스 제품을 구입해 주셔서 감사합니다.
사용 전에 안전을 위한 주의 사항을 반드시 읽고 정확하게 사용하여 주십시오.

제품 구입 감사 안내문

㈜오토닉스 제품을 구입해 주셔서 감사합니다.

먼저 **안전을 위한 주의사항**을 반드시 읽고 제품을 올바르게 사용해 주십시오.

본 사용자 매뉴얼은 제품에 대한 안내와 바른 사용 방법에 대한 내용을 담고 있으므로 사용자가 쉽게 찾아 볼 수 있는 장소에 보관하여 주십시오.

사용자 매뉴얼 안내

- 사용자 매뉴얼의 내용을 충분히 숙지한 후에 제품을 사용하여 주십시오.
- 사용자 매뉴얼은 제품 기능에 대해 자세하게 설명한 것으로, 사용자 매뉴얼 이외의 내용에 대해서는 보증하지 않습니다.
- 사용자 매뉴얼의 일부 또는 전부를 무단으로 편집 또는 복사하여 사용할 수 없습니다.
- 사용자 매뉴얼은 제품과 함께 제공하지 않습니다.
  당사 홈페이지(www.autonics.co.kr)에서 다운로드 하여 사용하십시오.
- 사용자 매뉴얼의 내용은 해당 제품의 성능 및 소프트웨어 개선에 따라 사전 예고없이 변경될 수 있으며, 업그레이드 공지는 당사 홈페이지를 통해 제공해 드립니다.
- 당사에서는 사용자 매뉴얼의 내용을 조금 더 쉽게, 정확하게 작성하고자 많은 노력을 기울였습니다. 그럼에도 불구하고 수정해야 될 부분이나 질문사항이 있으시면 당사 홈페이지를 통하여 의견을 주시기 바랍니다.

사용자 매뉴얼의 공통 기호

기호	설명
 Note	해당 기능에 대한 보충 설명
 Warning	지시 사항을 위반할 경우 심각한 상해나 사망 사고의 위험이 있는 내용
 Caution	지시 사항을 위반할 경우 경미한 상해나 제품 손상이 발생할 수 있는 내용
 Ex.	해당 기능에 대한 예시
※1	주석 설명 표시

안전을 위한 주의사항

- 안전을 위한 주의사항은 제품을 안전하고 올바르게 사용하여 사고나 위험을 미리 막기 위한 것이므로 반드시 지켜주십시오.
- 주의사항은 경고와 주의로 구분되어 있으며 각각의 의미는 다음과 같습니다.

Warning	경고	지시 사항을 위반하였을 때, 심각한 상해나 사망 사고가 발생할 가능성이 있는 경우
Caution	주의	지시 사항을 위반하였을 때, 경미한 상해나 제품 손상이 발생할 가능성이 있는 경우

Warning

- 인명이나 재산상에 영향이 큰 기기(예: 원자력 제어, 의료기기, 선박, 차량, 철도, 항공기, 연소장치, 안전장치, 방범/방재장치 등)에 사용할 경우 반드시 2 중으로 안전장치를 부착한 후 사용하십시오.
  화재, 인사 사고, 재산 상의 막대한 손실이 발생할 수 있습니다.
- 취급설명서에 기재된 일반사양의 환경에서 사용하십시오. 부식성 가스, 인화성 가스가 있는 장소, 고온, 다습, 진동, 화재, 오동작, 제품의 손상 또는 열화의 원인이 됩니다.
  화재, 인사 사고, 재산 상의 막대한 손실이 발생할 수 있습니다.
- 제품을 개조하지 마십시오.
  화재, 인사 사고, 재산 상의 막대한 손실이 발생할 수 있습니다.
- 운전 중에 전원을 차단하지 마십시오.
  인명사고, 재산상의 손실, 오작동의 원인이 됩니다.
- 운전시에는 항상 비상 정지가 가능하도록 하십시오.
  장치 파손 및 인명사고의 우려가 있습니다.
- 운전 중에 커넥터 및 점프핀을 분리하지 마십시오.
  인명사고, 재산상의 손실, 오작동의 원인이 됩니다.
- 제품의 폐기 시에는 산업 폐기물로써 처리하십시오.
  인명사고, 재산상의 손실이 발생할 수 있습니다.
- 반드시 PCI bus connector 에 장착하여 사용하십시오.
  장치 파손 및 È 재, 가전, 부상의 우려가 있습니다.
- 접속은 반드시 접속도를 기초로 하여 배선하십시오.
  화재나 감전 및 제품 파손의 우려가 있습니다.
- 리미트(한계) 스위치를 반드시 설치하십시오.
  인명사고, 재산상의 손실이 발생할 수 있습니다.

- 긴급 정지 스위치를 반드시 설치하십시오.

Caution

- 전원이 인가된 상태에서 결선 및 점검, 보수를 하지 마십시오.
  감전, 오동작의 원인이 됩니다.
- 자사 기술자 이외에는 제품을 수리하지 마십시오.
  감전, 화재의 우려가 있습니다. 수리가 필요할 경우 당사에 문의하십시오.
- 반드시 정격/성능 범위에서 사용하십시오.
  제품의 수명이 짧아지는 원인이 되며 화재의 우려가 있습니다.
- 청소 시 물, 유기용제를 사용하지 마십시오.
  감전, 화재, 제품 손상의 원인이 됩니다.
- 본 제품의 표면에 먼지나 배선 찌꺼기가 유입되지 않도록 하십시오.
  감전, 화재, 제품 손상의 원인이 됩니다.
- 보관 시에는 PC 와 I/O 케이블을 분리하여 정전기 방지용 포장지로 포장하여 정격/성능에 표기된 온도, 습도 이내에서 보관하십시오.
- 설치 전 배선 작업 시 반드시 전원을 차단한 후 설치하십시오.
  감전, 제품 손상의 우려가 있습니다.
- 설치 및 배선 작업 시 배선끼리 서로 단락되지 않도록 하십시오.
  감전, 제품 손상의 우려가 있습니다.
- 사용하지 않는 단자는 아무것도 결선하지 마시고 다른 단자와 단락되지 않도록 하십시오.
  감전, 제품 손상의 우려가 있습니다.

본 사용자 매뉴얼에 기재된 사양, 외형치수 등은 제품의 개선을 위하여 예고 없이 변경될 수 있습니다.

Table of Contents

1 초기화

1.1 open

MMC_INT16U **pmc4bpci_open**(int *id*, void (WINAPI *funcIntHandler)(void));

(1) 설명

MMC 보드의 초기화에 필요한 함수입니다. 인터럽트가 활성화 되면 호출되는 콜백함수를 설치할 수 있습니다.

아래는 PMC-4B-PCI 보드의 초기화순서 및 기본으로 설정되는 값입니다.

4 축 모두선택 → 리셋 → 레지스터청소 → 초기화

Range = 8,000,000, (배율=1:1)

K= 101 (가가속도=619kpps/sec²)

A= 1000 (가속도=125kpps/sec)

D= 1000 (감속도=125kpps/sec)

SV = 1000 (초기속도=1000pps)

V= 40000 (드라이브속도= 40000PPS)

P= 100000 (보간종점설정=100000)

LP = 0 (논리위치카운터설정=0)

보간모드 적용축 설정: 1 축→X 축, 2 축→Y 축, 3 축→Z 축

(2) 인자

- id: ID 번호를 입력합니다. (범위:0~15
- funcIntHandler: MMC 카드에 설정한 이벤트 발생시 사용자 함수를 호출 사용자 함수의 포인터를 전달

(3) 리턴값

드라이버 초기화 및 라이브러리 초기화 성공은 PMC4BPCI_OK 를 리턴하고, 실패할 경우에는 MMC_ERROR 을 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void WINAPI isr_sub0( void)
{
        // MMC Interrupt
}
void WINAPI isr_sub1( void)
{
```

```
		// MMC Interrupt
}
void WINAPI isr_sub2( void)
{
		// MMC Interrupt
}
void WINAPI isr_sub3( void)
{
		// MMC Interrupt
}
void WINAPI isr_sub4( void)
{
		// MMC Interrupt
}
void WINAPI isr_sub5( void)
{
		// MMC Interrupt
}
void WINAPI isr_sub6( void)
{
		// MMC Interrupt
}
void WINAPI isr_sub7( void)
{
		// MMC Interrupt
}
void WINAPI isr_sub8( void)
{
		// MMC Interrupt
}
void WINAPI isr_sub9( void)
{
		// MMC Interrupt
}
void WINAPI isr_sub10( void)
{
		// MMC Interrupt
}
void WINAPI isr_sub11( void)
{
		// MMC Interrupt
}
void WINAPI isr_sub12( void)
{
		// MMC Interrupt
}
void WINAPI isr_sub13( void)
{
		// MMC Interrupt
}
void WINAPI isr_sub14( void)
{
		// MMC Interrupt
}
void WINAPI isr_sub15( void)
{
		// MMC Interrupt
}
```



```
void main()
{
        MMC_INT16U stat;
        int i;

        // MMC 카드의 초기화 함수
        // 리턴값: 초기화 성공시에는 MMC_OK 를 리턴합니다.
        // 인자:
        // id - MMC 카드의 id 설정 스위치 값
        // funcIntHandler - MMC 카드에 설정한 이벤트 발생시 사용자 함수를 호출
        //                  사용자 함수의 포인터를 전달
        for(i=0;i<MAX_CARD;i++)
        {
                switch(i) {
                case  0: stat = pmc4bpci_open( 0, isr_sub0); break;
                case  1: stat = pmc4bpci_open( 1, isr_sub1); break;
                case  2: stat = pmc4bpci_open( 2, isr_sub2); break;
                case  3: stat = pmc4bpci_open( 3, isr_sub3); break;
                case  4: stat = pmc4bpci_open( 4, isr_sub4); break;
                case  5: stat = pmc4bpci_open( 5, isr_sub5); break;
                case  6: stat = pmc4bpci_open( 6, isr_sub6); break;
                case  7: stat = pmc4bpci_open( 7, isr_sub7); break;
                case  8: stat = pmc4bpci_open( 8, isr_sub8); break;
                case  9: stat = pmc4bpci_open( 9, isr_sub9); break;
                case 10: stat = pmc4bpci_open(10, isr_sub10); break;
                case 11: stat = pmc4bpci_open(11, isr_sub11); break;
                case 12: stat = pmc4bpci_open(12, isr_sub12); break;
                case 13: stat = pmc4bpci_open(13, isr_sub13); break;
                case 14: stat = pmc4bpci_open(14, isr_sub14); break;
                case 15: stat = pmc4bpci_open(15, isr_sub15); break;
                }
                if(stat==MMC_OK)
                {
printf("MESSAGE : Found and open 'PMC-4B-PCI(ID=%d)' driver\n", i);
                }
        }

        // 열린 윈도드라이버 닫기
        pmc4bpci_close_all();

        printf("\nComplete OK!\n");
}
```

(5) 참고함수

pmc4bpci_open, pmc4bpci_close_all

1.2 reset

MMC_INT16U **pmc4bpci_reset**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

PMC-4B-PCI 보드를 초기화합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U   rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn=pmc4bpci_set_reset(MMC_CARD_NO,PMC4BPCI_AXIS_X);
// ……

if(rtn!=MMC_OK)
{printf("Fail!\n");    }
else
{printf("Complete OK!\n");   }

pmc4bpci_close_all();

}
```

(5) 참고매크로

PMC4BPCI_CMD_RST 정지, 종료

2 정지, 종료

2.1 close

MMC_INT16U **pmc4bpci_close**(MMC_INT16U *id*);

(1) 설명

PMC-4B-PCI 보드를 종료합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

드라이버 종료에는 무조건 MMC_OK 을 리턴합니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        } else
        {
                printf("Open card(%d)!\n", MMC_CARD_NO);
        }

        // ……
        rtn = pmc4bpci_close(MMC_CARD_NO);
        if(rtn!=MMC_OK)
                printf("ERROR : CardID=%d\n", MMC_CARD_NO);
        else
                printf("Close card(%d)!\n", MMC_CARD_NO);
        // ……
}
```

(5) 참고함수

pmc4bpci_open

2.2 cloes all

MMC_INT16U pmc4bpci_close _all();

(1) 설명

모든 PMC-4B-PCI 보드를 종료합니다.

(2) 리턴값

드라이버 종료에는 무조건 MMC_OK 을 리턴합니다.

(3) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        } else
        {
                printf("Open card(%d)!\n", MMC_CARD_NO);
        }

        // ……
        rtn = pmc4bpci_close_all();
        if(rtn!=MMC_OK)
printf("ERROR : 열려진 드라이버를 찾을 수 없습니다!\n", MMC_CARD_NO);
        else
                printf("All card closed!\n");
        // ……
}
```

(4) 참고함수

pmc4bpci_open

2.3 dstop

MMC_INT16U **pmc4bpci_dstop**(MMC_INT16U id, MMC_INT16U *axis*);

(1) 설명

지정축의 드라이브를 감속 정지합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
MMC_INT16U    rtn;
        int OpenFlag;
        MMC_INT16U axis = PMC4BPCI_AXIS_X | PMC4BPCI_AXIS_Y |
                                PMC4BPCI_AXIS_Z | PMC4BPCI_AXIS_U ;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

rtn = pmc4bpci_dstop(MMC_CARD_NO, axis);

if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_dstop

2.4 stop

MMC_INT16U **pmc4bpci_stop**(MMC_INT16U id,MMC_INT16U *axis*);

(1) 설명

지정축의 드라이브를 즉시 종료합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
MMC_INT16U    rtn;
        int OpenFlag;
        MMC_INT16U axis = PMC4BPCI_AXIS_X | PMC4BPCI_AXIS_Y |
                                PMC4BPCI_AXIS_Z | PMC4BPCI_AXIS_U ;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

rtn = pmc4bpci_stop(MMC_CARD_NO, axis);

if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_dstop

3 레지스트값 읽기/쓰기

3.1 outw

MMC_INT16U **pmc4bpci_outw**(MMC_INT16U *id*, MMC_INT16U *wOffset*, MMC_INT16U *wData*);

(1) 설명

I/O 포트 입출력 함수입니다. pmc4bpci 용과 nova 호환용의 입출력주소에는 차이가 있으며, 위의 wOffset 은 I/O address 의 base address 를 기준으로 하는 상대 번지입니다. wOffset 번지의 시작은 0 번지부터입니다.

pmc4bpci_outw 는 pmc4bpci 의 해당번지(wOffset)에 wData(bData)를 입력합니다. 번지의 지정은 워드형식의 함수(pmc4bpci_outw())이며, 2byte 로 지정하면 됩니다. (예: pmc4bpci_outw(id, 0x00,0x8000))

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- wOffset: base address 를 기준으로 하는 상대 번지
- wData: I/O 포트에 써넣을 데이터

(3) 리턴값

주소입력의 범위가 벗어 날 경우 PMC4BPCI_IOADDRESS_ERR 를 리턴하며, 성공할 경우에는 PMC4BPCI_OK 를 리턴합니다. 에러의 확인은 전역변수 gError 를 확인하여 알 수 있습니다. 에러변수 gError 는 리턴값과 동일한 값을 가집니다.

(4) 주의

I/O 포트의 데이터를 읽어내는 함수일 경우에는 데이터 또는 에러를 리턴할 수 있습니다. 에러와 데이터의 구분은 오로지 에러를 표현하는 전역변수 gError 의 상태로 판단해야 합니다. 예로 리턴된 값이 PMC4BPCI_IOADDRESS_ERR 와 같을 경우 데이터인지 에러값인지의 판단은 gError 의 내용이 PMC4BPCI_OK 면 데이터로 판단하면 됩니다.

(5) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        MMC_INT16U    rtn, rv, i;
        int OpenFlag;
```

```
OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
if(OpenFlag!=MMC_OK)
{
        printf("Initialize error!\n");
        return;
}

// ……
rtn = pmc4bpci_outw(MMC_CARD_NO, wr0, PMC4BPCI_CMD_RST); // reset
if(MMC_ERROR()!=MMC_OK)
        printf("쓰기 설정주소 범위를 벗어났습니다!\n");

for(i=rr0;i<=rr7;i+=2) // word size 이므로 2 바이트씩 증가
{
        rv = pmc4bpci_inw(MMC_CARD_NO, i); // read status
        if(MMC_ERROR()!=MMC_OK)
                printf("읽기 설정주소 범위를 벗어났습니다!\n");
        else
                printf("id=%d, rr%d=0x%04x\n", MMC_CARD_NO, i/2, rv);
}
// ……

pmc4bpci_close_all();
}
```

(6) 참고함수

pmc4bpci_open, MMC_ERROR, pmc4bpci_close_all

3.2 inw

MMC_INT16U **pmc4bpci_inw**(MMC_INT16U *id*, MMC_INT16U *wOffset*);

(1) 설명

I/O 포트 입출력 함수입니다. pmc4bpci 용과 nova 호환용의 입출력주소에는 차이가 있으며, 위의 *wOffset* 은 I/O address 의 base address 를 기준으로 하는 상대 번지입니다. *wOffset* 번지의 시작은 0 번지부터입니다.

pmc4bpci_inw 는 pmc4bpci 의 해당번지(wOffset)에서 데이터를 읽습니다.
번지의 지정은 워드형식의 함수(pmc4bpci_inw())이며, 2byte 지정하면 됩니다.
(예: pmc4bpci_inw(id, 0x00))

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- wOffset: base address 를 기준으로 하는 상대 번지
- wData: I/O 포트에 써넣을 데이터

(3) 리턴값

주소입력의 범위가 벗어 날 경우 PMC4BPCI_IOADDRESS_ERR 를 리턴하며, 성공할 경우에는 PMC4BPCI_OK 를 리턴합니다. 에러의 확인은 전역변수 gError 를 확인하여 알 수 있습니다. 에러변수 gError 는 리턴값과 동일한 값을 갖는다.

I/O 포트의 데이터를 읽어내는 함수일 경우에는 데이터 또는 에러를 리턴할 수 있습니다. 에러와 데이터의 구분은 오로지 에러를 표현하는 전역변수 gError 의 상태로 판단해야 합니다. 예로 리턴된 값이 PMC4BPCI_IOADDRESS_ERR 와 같을 경우 데이터인지 에러값인지의 판단은 gError 의 내용이 PMC4BPCI_OK 면 데이터로 판단하면 됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        MMC_INT16U    rtn, rv, i;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
```

```
        }

        // ……
        rtn = pmc4bpci_outw(MMC_CARD_NO, wr0, PMC4BPCI_CMD_RST); // reset
        if(MMC_ERROR()!=MMC_OK)
                printf("쓰기 설정주소 범위를 벗어났습니다!\n");

        for(i=rr0;i<=rr7;i+=2) // word size 이므로 2 바이트씩 증가
        {
                rv = pmc4bpci_inw(MMC_CARD_NO, i); // read status
                if(MMC_ERROR()!=MMC_OK)
                        printf("읽기 설정주소 범위를 벗어났습니다!\n");
                else
                        printf("id=%d, rr%d=0x%04x\n", MMC_CARD_NO, i/2, rv);
        }
        // ……

        pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_open, MMC_ERROR, pmc4bpci_close_all

4 레지스트값 기록

4.1 wwr1

MMC_INT16U **pmc4bpci_wwr1**(MMC_INT16U *id*, MMC_INT16U *axis*, MMC_INT16U *wdata*)

(1) 설명

선택한 axis 의 mode register(1)에 drive 도중의 감속/정지 입력신호, IN0~IN3 의 유효/무효를 설정하는 bit 상태기록, interrupt 의 허가/금지를 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 선택한 축에 설정할 상태 값 (상태값은 사용자 매뉴얼 참조)

드라이브 정지 입력신호

WR1_IN0_L: 입력신호 IN0 에대한 논리레벨(0 이면 low level 에서정지, 1 이면 high level 에서감속정지)
WR1_IN0_E: 입력신호 IN0 에대한 유효/무효 설정(0-무효, 1-유효)
WR1_IN1_L: 입력신호 IN1 에대한 논리레벨(0 이면 low level 에서정지, 1 이면 high level 에서감속정지)
WR1_IN1_E: 입력신호 IN1 에대한 유효/무효 설정(0-무효, 1-유효)
WR1_IN2_L: 입력신호 IN2 에대한 논리레벨 (0 이면 low level 에서정지, 1 이면 high level 에서감속정지)
WR1_IN2_E: 입력신호 IN2 에대한 유효/무효 설정(0-무효, 1-유효)
WR1_IN3_L: 입력신호 IN3 에대한 논리레벨 (0 이면 low level 에서정지, 1 이면 high level 에서감속정지)
WR1_IN3_E: 입력신호 IN3 에대한 유효/무효 설정(0-무효, 1-유효)

인터럽트 발생조건

WR1_INT_PULSE: drive pulse 의 rising edge(펄스에서 올라가는 순간(rising edge))에서 발생 설정 인터럽트 발생(drive pulse 정논리 설정시)
WR1_INT_PGECM: 논리/실제위치 pulse 값 ≥ COMP-register 값 일때
WR1_INT_PLCM: 논리/실제위치 pulse 값 < COMP-register 값 일때
WR1_INT_PLCP: 논리/실제위치 pulse 값 < COMP+register 값 일때
WR1_INT_PGEP: 논리/실제위치 pulse 값 ≥ COMP+register 값 일때
WR1_INT_CEND: 정속지역에서 pulse 출력을 종료할때(가감속 drive 일때)
WR1_INT_CSTA: 정속지역에서 pulse 출력을 개시하였을때(가감속 drive 일때)
WR1_INT_DEND: drive 가 종료하였을때

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"


void WINAPI isr_sub( void)
{
        // MMC Interrupt
}

void main()
{
        #define MMC_CARD_NO     15
        MMC_INT16U     rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, isr_sub);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }
        // ……
        rtn = pmc4bpci_wwr1(MMC_CARD_NO, PMC4BPCI_AXIS_X,
WR1_INT_PGECM |
WR1_INT_CEND);
        // ……

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
}
```

(5) 참고매크로

WR1_IN0_L, WR1_IN0_E, WR1_IN1_L, WR1_IN1_E, WR1_IN2_L, WR1_IN2_E, WR1_IN3_L, WR1_IN3_E, WR1_INT_PULSE, WR1_INT_PGECM, WR1_INT_PLCM, WR1_INT_PLCP, WR1_INT_PGEP, WR1_INT_CEND, WR1_INT_CSTA, WR1_INT_DEND
IN0_SLOWSTOP, IN0_ENABLE, IN1_SLOWSTOP, IN1_ENABLE, IN2_SLOWSTOP, IN2_ENABLE, IN3_SLOWSTOP, IN3_ENABLE, INT_PULSE, INT_PGECM, INT_PLCM, INT_PLCP, INT_PGEP, INT_CEND, INT_CSTA, INT_DEND

4.2 wwr2

MMC_INT16U **pmc4bpci_wwr2**(MMC_INT16U *id*, MMC_INT16U *axis*, MMC_INT16U *wdata*);

(1) 설명

선택한 축의 mode register(2)에 limit 입력신호의 mode 설정, drive pulse 의 mode 설정 encoder 입력신호의 mode 설정, servo motor 용 신호의 논리 level 유효/무효 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: mode 설정값(wr2 의 레지스터 내용과 동일합니다.)

WR2_LMT_SLMTP: COMP+register 를 software limit 로 설정
WR2_LMT_SLMTM: COMP-register 를 software limit 로 설정
WR2_LMT_LMTMD: hardware limit(nLMTP, nLMTM)가 active 될 경우 정지방식 설정(0: 즉시정지, 1: 감속정지)
WR2_LMT_HLMTP: +방향 limit 입력신호(nLMTP)의 논리 level 설정 (0: low, 1: high 에서 active)
WR2_LMT_HLMTM: -방향 limit 입력신호(nLMTM)의 논리 level 설정 (0: low, 1: high 에서 active)
WR2_LMT_CMPSL: COMP+/- register 의 비교대상(0: 논리위치, 1: 실제위치와 비교)
WR2_DRV_PLSMD: drive pulse 의 출력방식(0: 독립 2 pulse 방식, 1: 1 pulse 방식)
WR2_DRV_PLS_L: drive pulse 의 논리 level 설정(0: 정논리, 1: 부논리)
WR2_DRV_DIR_L: drive pulse 의 방향출력신호의 논리 level 설정(0: +방향일때 low,-방향일때 hi,1:+방향일때 hi,-방향일때 low)
WR2_ENC_PINMD: 엔코더입력신호(nECA/PPIN, nECB/PMIN)의 출력방식 (0:2 상 1:up/dn pulse 입력)
WR2_ENC_PIND0: WR2_ENC_PIND1 와 조합에 의해 사용
WR2_ENC_PIND1: encoder 2 상 pulse 입력의 분주비를 설정(D1D0 00=1/1, 01=1/2, 10=1/4, 11=무효)
WR2_SRV_ALM_L: nALARM 입력 신호의 논리 level 을 설정(0: low, 1:high 에서 active)
WR2_SRV_ALM_E: servo motor alarm 용 입력신호 nALARM 의 유/무효 설정(0: 무효, 1: 유효)
WR2_SRV_INP_L: nINPOS 입력 신호의 논리 lovel 을 설정(0: low, 1: high 에서 active)
WR2_SRV_INP_E: servo motor 위치 결정 완료용 입력신호 nINPOS 의 유효/무효 설정(0: 무효, 1: 유효)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는

MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"


void main()
{
        #define MMC_CARD_NO     15
        MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }


rtn = pmc4bpci_wwr2(MMC_CARD_NO, PMC4BPCI_AXIS_X, WR2_LMT_SLMTP |
WR2_LMT_SLMTM);

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
}
```

(5) 참고매크로

WR2_LMT_SLMTP, WR2_LMT_SLMTM, WR2_LMT_LMTMD, WR2_LMT_HLMTP, WR2_LMT_HLMTM, WR2_LMT_CMPSL, WR2_DRV_PLSMD, WR2_DRV_PLS_L, WR2_DRV_DIR_L, WR2_ENC_PINMD, WR2_ENC_PIND0, WR2_ENC_PIND1, WR2_SRV_ALM_L, WR2_SRV_ALM_E, WR2_SRV_INP_L, WR2_SRV_INP_E

LMT_SLMTP, LMT_SLMTM, LMT_LMTMD, LMT_HLMTP, LMT_HLMTM, LMT_CMPSL, DRV_PLSMD, DRV_PLS_L, DRV_DIR_L, ENC_PINMD, ENC_PIND0, ENC_PIND1, SRV_ALM_L, SRV_ALM_E, SRV_INP_L, SRV_INP_E

4.3 wwr3

MMC_INT16U **pmc4bpci_wwr3**(MMC_INT16U *id*, MMC_INT16U *axis*, MMC_INT16U *wdata*);

(1) 설명

선택한 축의 mode register(3)에 manual 감속, 개별 감속도, S 자 가감속, 외부조작 mode 의 설정, 범용출력 OUT4~7 을 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위:0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- *wdata*: mode 설정값

WR3_DRV_MANLD: 가감속 정량 drive 의 감속방법(0: 자동감속,1: manual 감속)
WR3_DRV_DSNSE: 가감속 drive 감속시 영향을 받는 종류
(0: 가속도의 값에 영향, 1: 감속도의 값에 영향-manual 방식일때만 사용)
WR3_DRV_SACC: 직선가감속/S 자 가감속 설정(0: 직선가감속, 1: S 자 가감속)
WR3_DRV_EXOP0: WR3_DRV_EXOP1 와 조합에 의해 사용
WR3_DRV_EXOP1: 외부입력신호 (D4D3 00=외부입력신호에의한 drive 조작무효,
01=연속 drive mode, 10=정량 drive mode, 11=외부입력신호에의한 drive 조작무효)
WR3_OUT_OUTSL: 출력신호 nOUT4~7 을 범용/drive 상태의 여부결정
(0: 범용출력, 1: drive 상태표시)
WR3_OUT_OUT4: 출력신호로 사용할때 사용(0: low level, 1: high level)
WR3_OUT_OUT5: 출력신호로 사용할때 사용(0: low level, 1: high level)
WR3_OUT_OUT6: 출력신호로 사용할때 사용(0: low level, 1: high level)
WR3_OUT_OUT7: 출력신호로 사용할때 사용(0: low level, 1: high level)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"


void main()
{
```

```
#define MMC_CARD_NO     15
MMC_INT16U    rtn;
int OpenFlag;

OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
if(OpenFlag!=MMC_OK)
{
        printf("Initialize error!\n");
        return;
}

        rtn = pmc4bpci_wwr3(MMC_CARD_NO, PMC4BPCI_AXIS_X,
WR3_DRV_SACC);

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
}
```

(5) 참고매크로

WR3_DRV_MANLD, WR3_DRV_DSNSE, WR3_DRV_SACC, WR3_DRV_EXOP0, WR3_DRV_EXOP1, WR3_OUT_OUTSL, WR3_OUT_OUT4, WR3_OUT_OUT5, WR3_OUT_OUT6, WR3_OUT_OUT7
DRV_MANLD, DRV_DSNSE, DRV_SACC, DRV_EXOP0, DRV_EXOP1, OUT_OUTSL, OUT_OUT4, OUT_OUT5, OUT_OUT6, OUT_OUT7

4.4 wwr4

MMC_INT16U **pmc4bpci_wwr4**(MMC_INT16U *id*, MMC_INT16U *axis*, MMC_INT16U *wdata*);

(1) 설명

각축 X, Y, Z, U * (nOUT0~3)4bit = 16bit 의 데이터 범위 내에서 각축의 데이터 설정 또는 16 비트 출력을 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위:0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 출력 값

WR4_OUT_XOUT0: X 축 출력신호 OUT0 의 출력 설정(0: low, 1: high)
WR4_OUT_XOUT1: X 축 출력신호 OUT1 의 출력 설정(0: low, 1: high)
WR4_OUT_XOUT2: X 축 출력신호 OUT2 의 출력 설정(0: low, 1: high)
WR4_OUT_XOUT3: X 축 출력신호 OUT3 의 출력 설정(0: low, 1: high)
WR4_OUT_YOUT0: Y 축 출력신호 OUT0 의 출력 설정(0: low, 1: high)
WR4_OUT_YOUT1: Y 축 출력신호 OUT1 의 출력 설정(0: low, 1: high)
WR4_OUT_YOUT2: Y 축 출력신호 OUT2 의 출력 설정(0: low, 1: high)
WR4_OUT_YOUT3: Y 축 출력신호 OUT3 의 출력 설정(0: low, 1: high)
WR4_OUT_ZOUT0: Z 축 출력신호 OUT0 의 출력 설정(0: low, 1: high)
WR4_OUT_ZOUT1: Z 축 출력신호 OUT1 의 출력 설정(0: low, 1: high)
WR4_OUT_ZOUT2: Z 축 출력신호 OUT2 의 출력 설정(0: low, 1: high)
WR4_OUT_ZOUT3: Z 축 출력신호 OUT3 의 출력 설정(0: low, 1: high)
WR4_OUT_UOUT0: U 축 출력신호 OUT0 의 출력 설정(0: low, 1: high)
WR4_OUT_UOUT1: U 축 출력신호 OUT1 의 출력 설정(0: low, 1: high)
WR4_OUT_UOUT2: U 축 출력신호 OUT2 의 출력 설정(0: low, 1: high)
WR4_OUT_UOUT3: U 축 출력신호 OUT3 의 출력 설정(0: low, 1: high)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
```

```
#include "../include/pmc4bpci.h"


void main()
{
        #define MMC_CARD_NO     15
        MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }
        rtn = pmc4bpci_wwr4(MMC_CARD_NO, PMC4BPCI_AXIS_X,
WR4_OUT_XOUT0 |
WR4_OUT_XOUT1);

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
}
```

(5) 참고매크로

WR4_OUT_XOUT0, WR4_OUT_XOUT1, WR4_OUT_XOUT2, WR4_OUT_XOUT3,
WR4_OUT_YOUT0, WR4_OUT_YOUT1, WR4_OUT_YOUT2, WR4_OUT_YOUT3,
WR4_OUT_ZOUT0, WR4_OUT_ZOUT1, WR4_OUT_ZOUT2, WR4_OUT_ZOUT3,
WR4_OUT_UOUT0, WR4_OUT_UOUT1, WR4_OUT_UOUT2, WR4_OUT_UOUT3

4.5 wwr5

MMC_INT16U **pmc4bpci_wwr5**(MMC_INT16U *id*, MMC_INT16U *axis*, MMC_INT16U *wdata*);

(1) 설명

보간 mode 를 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 출력 값

WR5_INP_LSPD0: 0x0100 보간 drive 의 선속일정 mode 를 설정
WR5_INP_LSPD1: 0x0200 D1D0(00: 선속 일정 무효, 01: 2 축선속 일정, 10: 설정불가, 11: 3 축선속일정)
WR5_INP_EXPLS: 0x0800 외부전송여부(0: 전송안함, 1: 보간 drive 를 외부신호-EXPLSN-로 step 전송)
WR5_INP_CMPLS: 0x1000(0: 전송안함, 1: 보간 drive 를 command 로 step 전송)
WR5_INP_CIINT: 0x4000
(연속보간시의 interrupt 발생 허가/금지 설정(0: 금지, 1: 허가)
WR5_INP_BPINT: 0x8000 bit pattern 보간시 interrupt 발생 허가/금지 설정
(0: 금지, 1: 허가)
WR5_INP_AXIS1_X: 보간시 제 1 축에 지정하고자 하는축 정의(X 축)
WR5_INP_AXIS1_Y: Y 축
WR5_INP_AXIS1_Z: Z 축
WR5_INP_AXIS1_U: U 축
WR5_INP_AXIS2_X: 보간시 제 2 축에 지정하고자 하는축 정의(X 축)
WR5_INP_AXIS2_Y: Y 축
WR5_INP_AXIS2_Z: Z 축
WR5_INP_AXIS2_U: U 축
WR5_INP_AXIS3_X: 보간시 제 3 축에 지정하고자 하는축 정의(X 축)
WR5_INP_AXIS3_Y: Y 축
WR5_INP_AXIS3_Z: Z 축
WR5_INP_AXIS3_U: U 축
WR5_INP_LCMODE_INVALIDATE: 보간 drive 의 무효 mode
WR5_INP_LCMODE_2AXIS: 2 축 선속 일정
WR5_INP_LCMODE_3AXIS: 3 축 선속 일정

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는
MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는

MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        // 보간축 선택(1 축=X, 2 축=Y, 3 축=U)
        rtn = pmc4bpci_wwr5(MMC_CARD_NO, 0, WR5_INP_AXIS1_X |
WR5_INP_AXIS2_Y
| WR5_INP_AXIS3_U);

        if(rtn!=MMC_OK)
{printf(“Fail!\n”);     }
else
{printf(“Complete OK!\n”);    }

        pmc4bpci_close_all();
}
```

(5) 참고매크로

WR5_INP_EXPLS, WR5_INP_CMPLS, WR5_INP_CIINT, WR5_INP_BPINT,
R5_INP_AXIS1_X, WR5_INP_AXIS1_Y, WR5_INP_AXIS1_Z, WR5_INP_AXIS1_U,
WR5_INP_AXIS2_X, WR5_INP_AXIS2_Y, WR5_INP_AXIS2_Z, WR5_INP_AXIS2_U,
WR5_INP_AXIS3_X, WR5_INP_AXIS3_Y, WR5_INP_AXIS3_Z, WR5_INP_AXIS3_U,
WR5_INP_LCMODE_INVALIDATE, WR5_INP_LCMODE_2AXIS,
WR5_INP_LCMODE_3AXIS, WR5_INP_AX10, WR5_INP_AX11, WR5_INP_AX20,
WR5_INP_AX21, WR5_INP_AX30, WR5_INP_AX31, WR5_INP_LSPD0,
WR5_INP_LSPD1, WR5_INP_EXPLS, WR5_INP_CMPLS, WR5_INP_CIINT,
WR5_INP_BPINT
INP_EXPLS, INP_CMPLS, INP_CIINT, INP_BPINT, INP_AXIS1_X, INP_AXIS1_Y,
INP_AXIS1_Z, INP_AXIS1_U, INP_AXIS2_X, INP_AXIS2_Y, INP_AXIS2_Z, INP_AXIS2_U,
INP_AXIS3_X, INP_AXIS3_Y, INP_AXIS3_Z, INP_AXIS3_U, INP_LCMODE_INVALIDATE,
INP_LCMODE_2AXIS, INP_LCMODE_3AXIS

4.6 write_data

MMC_INT16U **pmc4bpci_write_data**(MMC_INT16U *id*, MMC_INT16U *axis*, MMC_INT32U *wdata*);

(1) 설명

6.7 번 쓰기 레지스터(ww6, ww7)에 데이터를 기록합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 출력 데이터 값

(3) 리턴값

이 함수는 항상 성공하며 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
 #include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        // ……
        // 범위설정 예
        rtn = pmc4bpci_write_data(MMC_CARD_NO, PMC4BPCI_AXIS_X, 8000000);
        rtn = pmc4bpci_outw(MMC_CARD_NO, wr0, PMC4BPCI_AXIS_X |
PMC4BPCI_CMD_R);

        // ……

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
```

}

(5) 참고매크로

WR6_OUT_WD0, WR6_OUT_WD1, WR6_OUT_WD2, WR6_OUT_WD3, WR6_OUT_WD4, WR6_OUT_WD5, WR6_OUT_WD6, WR6_OUT_WD7, WR6_OUT_WD8, WR6_OUT_WD9, WR6_OUT_WD10, WR6_OUT_WD11, WR6_OUT_WD12, WR6_OUT_WD13, WR6_OUT_WD14, WR6_OUT_WD15 , WR7_OUT_WD0, WR7_OUT_WD1, WR7_OUT_WD2, WR7_OUT_WD3, WR7_OUT_WD4, WR7_OUT_WD5, WR7_OUT_WD6, WR7_OUT_WD7, WR7_OUT_WD8, WR7_OUT_WD9, WR7_OUT_WD10, WR7_OUT_WD11, WR7_OUT_WD12, WR7_OUT_WD13, WR7_OUT_WD14, WR7_OUT_WD15

5 레지스트값 읽기

5.1 rr0~rr5

MMC_INT16U **pmc4bpci_rr0(**MMC_INT16U id, MMC_INT16U *axis*);

MMC_INT16U **pmc4bpci_rr1(**MMC_INT16U id, MMC_INT16U *axis*);

MMC_INT16U **pmc4bpci_rr2(**MMC_INT16U id, MMC_INT16U *axis*);

MMC_INT16U **pmc4bpci_rr3(**MMC_INT16U id, MMC_INT16U *axis*);

MMC_INT16U **pmc4bpci_rr4(**MMC_INT16U id, MMC_INT16U *axis*);

MMC_INT16U **pmc4bpci_rr5(**MMC_INT16U id, MMC_INT16U *axis*);

(1) 설명

읽기 레지스터 rr0~5 번 에서 데이터를 읽습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.

(3) 리턴값

데이터 읽기에 성공하면 16 비트의 데이터를 리턴합니다. 리턴되는 각각의 코드는 “사용자 매뉴얼[4. Read register]를 참조합니다. PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void WINAPI isr_sub( void)
{
        // MMC Interrupt
}

void main()
{
                #define MMC_CARD_NO     15
                MMC_INT16U    rtn;
                int OpenFlag;

                OpenFlag = pmc4bpci_open(MMC_CARD_NO, isr_sub);
                if(OpenFlag!=MMC_OK)
                {
                        printf("Initialize error!\n");
```

```
                    return;
                }

                rtn = pmc4bpci_rr1(MMC_CARD_NO, PMC4BPCI_AXIS_X);
                // ……
                if(rtn!=MMC_OK)
{printf("Fail!\n");    }
else
{printf("Complete OK!\n");    }
                pmc4bpci_close_all();
}
```

6 파라미터 설정

6.1 wait

MMC_INT16U **pmc4bpci_wait**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

선택축의 drive pulse 가 출력중인 동안 대기합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정된 타임아웃시간에 의해 지정시간이 끝나 함수를 종료할 경우에는 PMC4BPCI_TIMEOUT_ERR 을 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
// 직선 가감속 정량 드라이버
pmc4bpci_pls_move(MMC_CARD_NO, PMC4BPCI_AXIS_X, 20000, 2000, 100);

// 드라이버 완료시까지 기다림
pmc4bpci_wait(MMC_CARD_NO, PMC4BPCI_AXIS_X);
// ……
```

```
printf("Complete OK!\n");

pmc4bpci_close_all();

}
```

(5) 참고함수

pmc4bpci_get_timeout, pmc4bpci_set_timeout, pmc4bpci_wait_timeout

6.2 wait time out

MMC_INT16U **pmc4bpci_wait_timeout**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32U *msec*);

(1) 설명

선택축의 drive pulse 가 출력중인 동안 대기합니다. 설정된 타임아웃시간(*msec*)에 의해 지정시간 이상을 기다릴 경우 **pmc4bpci_wait_timeout**()함수는 자동 종료됩니다. 타임아웃시간(*msec*)이 0 일 경우에는 무한정 기다린다. 인자 *msec* 에 의해 내부 타임아웃 시간이 변경됩니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- msec: msec(1/1000sec)단위의 타임아웃 시간설정.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정된 타임아웃시간에 의해 지정시간이 끝나 함수를 종료할 경우에는 PMC4BPCI_TIMEOUT_ERR 을 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
// 직선 가감속 정량 드라이버
pmc4bpci_pls_move(MMC_CARD_NO, PMC4BPCI_AXIS_X, 20000, 2000, 100);
```

```
// 타임아웃 시간만큼 기다림(1000msec)
pmc4bpci_wait_timeout(MMC_CARD_NO, PMC4BPCI_AXIS_X, 1000);
// ……

printf("Complete OK!\n");

pmc4bpci_close_all();

}
```

(5) 참고함수

pmc4bpci_set_timeout, pmc4bpci_get_timeout, pmc4bpci_wait

6.3 nextwait

MMC_INT16U **pmc4bpci_nextwait**(MMC_INT16U *id*, MMC_INT16U *msec*);

(1) 설명

연속보간 drive 에서 다음 node 를 위해 parameter data/보간 명령을 기입 가능할 동안 대기합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- msec: msec(0.001 초)단위의 타임아웃 시간설정.

(3) 리턴값

설정된 타임아웃시간에 의해 지정시간이 끝나 함수를 종료할 경우에는 PMC4BPCI_TIMEOUT_ERR 을 리턴합니다. 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

{
        // 보간시 1 축에지정축, 보간시 2 축에지정축
        MMC_INT16U work_plane_axis[2] = {WR5_INP_AXIS1_X,
WR5_INP_AXIS2_Y};
        MMC_VERTEX p = {20000, -20000, };
        MMC_INT16U axis = PMC4BPCI_AXIS1 | PMC4BPCI_AXIS2 |
PMC4BPCI_AXIS3 |
                        PMC4BPCI_AXIS4 ;
```

```
// ……
// 2 축 직선 보간 드라이브
pmc4bpci_pos_imove2(MMC_CARD_NO, work_plane_axis, &p);

// 다음 명령기입 가능까지 대기
pmc4bpci_nextwait(MMC_CARD_NO, 0);

p.x = 20000;
p.y = 0;

// 2 축 직선 보간 드라이브
pmc4bpci_pos_imove2(MMC_CARD_NO, work_plane_axis, &p);
// ……

printf("\nComplete OK!\n");

}

 // 열린 윈도드라이버 닫기
 pmc4bpci_close_all();
}
```


6.4 get_timeout

MMC_INT32U pmc4bpci_get_timeout();

(1) 설명

설정된 타임아웃시간을 리턴합니다.

(2) 리턴값

설정된 내부 타임아웃시간(0.001 초 단위)을 리턴합니다. 이 명령은 항상 성공하며 전역 에러 검사 변수인 gError 에는 PMC4BPCI_OK 가 기록됩니다.

(3) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
MMC_INT32U t = pmc4bpci_get_timeout();
// ……

 printf("\nComplete OK!\n");

 // 열린 윈도드라이버 닫기
 pmc4bpci_close_all();

}
```

(4) 참고함수

pmc4bpci_set_timeout, pmc4bpci_wait, pmc4bpci_wait_timeout

6.5 set_timeout

MMC_INT16U **pmc4bpci_set_timeout**(MMC_INT32U *msec*);

(1) 설명

드라이브 완료를 기다리는 무한루프를 빠져 나오기 위해 내부 타임아웃 시간을 적용합니다. 시간은 msec(0.001 초)단위입니다.

시스템 타이머를 카운트하는 방식이 아니기 때문에 타임아웃 시간은 부정확합니다.

(2) 인자

- msec: msec(0.001 초)단위의 타임아웃 시간설정.

(3) 리턴값

내부 타임아웃 시간 설정은 무조건 PMC4BPCI_OK 을 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
pmc4bpci_set_timeout(1000); // delay 1sec
// ……

printf("\nComplete OK!\n");

 // 열린 윈도드라이버 닫기
 pmc4bpci_close_all();

}
```

(5) 참고함수

pmc4bpci_get_timeout, pmc4bpci_wait, pmc4bpci_wait_timeout

6.6 bpwait

MMC_INT16U **pmc4bpci_bpwait**(MMC_INT16U *id*);

(1) 설명

bit pattern 보간 drive 에서 stack counter(SC)의 값을 가르킵니다. stack counter 가 11(SC=3) 일동안 대기합니다. 설정된 내부 타임아웃시간에 의해 지정시간 이상을 대기할 경우 pmc4bpci_bpwait()함수는 자동 종료됩니다. 내부 타임아웃시간이 0 일 경우에는 무한정 대기합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

bit pattern 보간을 위한 스택 공간에 여유가 있는지 검사합니다. 검사결과 다음 16bit 의 bit pattern 데이터를 기록가능하면 PMC4BPCI_OK 를 리턴합니다. 설정된 내부 타임아웃시간에 의해 지정시간이 끝나 함수를 종료할 경우에는 PMC4BPCI_TIMEOUT_ERR 을 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
 #include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn = pmc4bpci_bpwait(MMC_CARD_NO);
// ……

if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_set_timeout, pmc4bpci_get_timeout

6.7 set_range

MMC_INT16U **pmc4bpci_set_range**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32 *wdata*)

(1) 설명

드라이브 범위를 설정합니다. 설정된 범위에 따라 자동으로 배율이 결정됩니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 드라이브 범위 설정(R)
  설정범위: 8,000,000/R (설정범위 8,000,000(1 배율)~16,000(500 배율))

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"


void main()
{
        #define MMC_CARD_NO      15
        MMC_INT16U   rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        // ……
        // 범위설정 예
        rtn = pmc4bpci_set_range(MMC_CARD_NO,
 PMC4BPCI_AXIS_X|PMC4BPCI_AXIS_Y|PMC4BPCI_AXIS_Z|PMC4BPCI_AXIS_U,
        8000000);

        if(rtn!=MMC_OK)
{printf("Fail!\n");    }
else
{printf("Complete OK!\n");   }
```

```
        pmc4bpci_close_all();
    }
```

(5) 참고매크로

PMC4BPCI_CMD_R, PMC4BPCI_RANGE

6.8 set_axis

MMC_INT16U **pmc4bpci_set_axis**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

축을 선택합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위:0~15)
- axis: 선택 축은 매크로에 의해 선택합니다

PMC-4B-PCI 용 매크로 및 값

X 축: PMC4BPCI_AXIS1, PMC4BPCI_AXIS_X, 0x0100

Y 축: PMC4BPCI_AXIS2, PMC4BPCI_AXIS_Y, 0x0200

Z 축: PMC4BPCI_AXIS3, PMC4BPCI_AXIS_Z, 0x0400

U 축: PMC4BPCI_AXIS4, PMC4BPCI_AXIS_U, 0x0800

아래 매크로 정의는 NOVA 호환용 함수와 같이 사용합니다.

X 축: AXIS_X, 0x01

Y 축: AXIS_Y, 0x02

Z 축: AXIS_Z, 0x04

U 축: AXIS_U, 0x08

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
```

```
            }

// ……
rtn=pmc4bpci_set_axis(MMC_CARD_NO,
 PMC4BPCI_AXIS_X+PMC4BPCI_AXIS_Y);// X,Y 축 선택
// ……

if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

pmc4bpci_close_all();

}
```

(5) 참고매크로

PMC4BPCI_CMD_NOP

6.9 smove_stop

MMC_INT16U **pmc4bpci_smove_stop**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

지정축의 S 자 속도커브 드라이브의 완료 후 직선 가감속 드라이브를 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위:0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
MMC_INT16U    rtn;
        int OpenFlag;
        MMC_INT16U axis = PMC4BPCI_AXIS_X | PMC4BPCI_AXIS_Y |
                                PMC4BPCI_AXIS_Z | PMC4BPCI_AXIS_U ;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

rtn = pmc4bpci_smove_stop(MMC_CARD_NO, axis);

if(rtn!=MMC_OK)
{printf("Fail!\n");    }
else
{printf("Complete OK!\n");   }

pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_pos_smove, pmc4bpci_pos_smove_wait, pmc4bpci_set_mod3

6.10 set_msgdispatch

MMC_INT16U **pmc4bpci_set_msgdispatch**(PMC4BPCI_MSG_DISPATCH *funcHandler,* MMC_INT32U *wParam*);

(1) 설명

MMC 라이브러리에서 윈도함수를 사용할 수 있게 Hooking 기능을 제공합니다.

(2) 인자

- funcHandler: 함수형 포인터
- wParam: funcHandler 함수 전달인자

(3) 리턴값

무조건 성공하며 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

// pmc4bpci_open()함수에 의해 초기화 되는 인터럽트 처리함수
MMC_VOID MyHookFunc(MMC_INT32U wParam)
{
 // Message Dispatch
}

void main()
{
MMC_INT16U    rtn;
int OpenFlag;

OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
if(OpenFlag!=MMC_OK)
{
 printf("Initialize error!\n");
 return;
}

rtn = pmc4bpci_set_msgdispatch( MyHookFunc , 0);
if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

pmc4bpci_close_all();
}
```


6.11 command

MMC_INT16U **pmc4bpci_command**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT16U *wCmd*);

(1) 설명

선택한 축에 명령합니다. 명령코드의 자세한 설명은 사용자 매뉴얼을 참조하십시오.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wCmd: 명령값

명령코드(Command code)

PMC4BPCI_CMD_R: Range 설정(8,000,000:배율=1~16,000:배율:500),4 byte
PMC4BPCI_CMD_K: 가가속도 설정 (1~65535), 2 byte
PMC4BPCI_CMD_A: 가속도 설정(1~8000), 2 byte
PMC4BPCI_CMD_D: 감속도 설정(1~8000), 2 byte
PMC4BPCI_CMD_SV: 처음속도 설정(1~8000), 2 byte
PMC4BPCI_CMD_V: drive 속도 설정(1~8000), 2 byte
PMC4BPCI_CMD_P: 출력 pulse 수(0~268,435,455), 4 byte
보간종점(-8,388,608~8,388,607), 4 byte
PMC4BPCI_CMD_DP: manual 감속점 설정(0~2g), 4 byte
PMC4BPCI_CMD_C: 원호중심점 설정(-8m~8m), 4 byte
PMC4BPCI_CMD_LP: 논리위치 counter 설정(-2g~2g),4 byte
PMC4BPCI_CMD_EP: 실위치 counter 설정(-2g~2g), 4 byte
PMC4BPCI_CMD_CP: COMP +register 설정(-2g~2g), 4 byte
PMC4BPCI_CMD_CM: COMP -register 설정(-2g~2g), 4 byte
PMC4BPCI_CMD_AO: 가속 counter offset 설정(0~65535), 2 byte
PMC4BPCI_CMD_AD: 감가속도 설정(1~65535), 2 byte
PMC4BPCI_CMD_NOP: NOP(축교환용)
PMC4BPCI_CMD_HS_V: 원점 검출속도 설정(1~8,000), 2 byte
PMC4BPCI_CMD_HS_RUN: 원점복귀 실행
PMC4BPCI_CMD_RST: Reset

데이터 읽어내기 명령

PMC4BPCI_READ_LP: 논리위치 counter 읽어내기 (-2g~2g),4 byte
PMC4BPCI_READ_EP: 실제위치 counter 읽어내기 (-2g~2g),4 byte
PMC4BPCI_READ_CV: 현재 drive 속도 읽어내기(1~8000),2 byte
PMC4BPCI_READ_CA: 현재 가/감속도 읽어내기(1~8000),2 byte

드라이브 명령

PMC4BPCI_DRV_PF: +방향 정량 drive

PMC4BPCI_DRV_MF: -방향 정량 drive

PMC4BPCI_DRV_PC: +방향 연속 drive

PMC4BPCI_DRV_MC: -방향 연속 drive

PMC4BPCI_DRV_DH: drive 개시 hold

PMC4BPCI_DRV_DF: drive 개시 free/완료 status clear

PMC4BPCI_DRV_DDS: drive 감속 정지

PMC4BPCI_DRV_DS: drive 즉시 정지

보간 명령

PMC4BPCI_INP_2LD: 2 축 직선보간 drive

PMC4BPCI_INP_3LD: 3 축 직선보간 drive

PMC4BPCI_INP_CW: CW 원호 보간 drive

PMC4BPCI_INP_CCW: CCW 원호 보간 drive

PMC4BPCI_INP_2BP: 2 축 bit pattern 보간 drive

PMC4BPCI_INP_3BP: 3 축 bit pattern 보간 drive

PMC4BPCI_INP_BPE: BP register 기입 가능

PMC4BPCI_INP_BPD: BP register 기입 불가능

PMC4BPCI_INP_BPS: BP data stack

PMC4BPCI_INP_BPC: BP data clear

PMC4BPCI_INP_SS: 보간 single step

PMC4BPCI_INP_DV1: 감속 유효

PMC4BPCI_INP_DV2: 감속 무효

PMC4BPCI_INP_IC: 보간 interrupt clear

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
```

```
MMC_INT16U    rtn;
int OpenFlag;

OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
if(OpenFlag!=MMC_OK)
{
        printf("Initialize error!\n");
        return;
}

// ……
// 범위설정 예
rtn = pmc4bpci_write_data(MMC_CARD_NO, PMC4BPCI_AXIS_X, 8000000);
rtn = pmc4bpci_command(MMC_CARD_NO, PMC4BPCI_AXIS_X,
                                        PMC4BPCI_CMD_R);
//      outw 를 사용해도 같은결과
//      rtn = pmc4bpci_outw(MMC_CARD_NO, wr0, PMC4BPCI_AXIS_X |
                                        PMC4BPCI_CMD_R);

// 범위설정 예
//      pmc4bpci_set_range(MMC_CARD_NO, PMC4BPCI_AXIS_X, 8000000);

        if(rtn!=MMC_OK)
{printf("Fail!\n");    }
else
{printf("Complete OK!\n");    }
        pmc4bpci_close_all();
}
```

(5) 참고매크로

PMC4BPCI _CMD_R, PMC4BPCI _CMD_K, PMC4BPCI _CMD_A, PMC4BPCI _CMD_D, PMC4BPCI _CMD_SV, PMC4BPCI _CMD_V, PMC4BPCI _CMD_P, PMC4BPCI_CMD_DP, PMC4BPCI_CMD_C, PMC4BPCI_CMD_LP, PMC4BPCI_CMD_EP, PMC4BPCI_CMD_CP, PMC4BPCI_CMD_CM, PMC4BPCI_CMD_AO, PMC4BPCI_CMD_NOP, PMC4BPCI_CMD_RST,
PMC4BPCI_READ_LP, PMC4BPCI_READ_EP, PMC4BPCI_READ_CV, PMC4BPCI_READ_CA,
PMC4BPCI_DRV_PF, PMC4BPCI_DRV_MF, PMC4BPCI_DRV_PC, PMC4BPCI_DRV_MC, PMC4BPCI_DRV_DH, PMC4BPCI_DRV_DF, PMC4BPCI_DRV_DDS, PMC4BPCI_DRV_DS,
PMC4BPCI_INP_2LD, PMC4BPCI_INP_3LD, PMC4BPCI_INP_CW, PMC4BPCI_INP_CCW,
PMC4BPCI_INP_2BP, PMC4BPCI_INP_3BP, PMC4BPCI_INP_BPE, PMC4BPCI_INP_BPD, PMC4BPCI_INP_BPS, PMC4BPCI_INP_BPC, PMC4BPCI_INP_SS, PMC4BPCI_INP_DV1, PMC4BPCI_INP_DV2, PMC4BPCI_INP_IC

7 초기설정

7.1 set_acac

MMC_INT16U **pmc4bpci_set_acac**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT16U *wdata*);

(1) 설명

가가속도를 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 가가속도 설정값(K), 가가속도=((62.5*10^6)/K)*배율 (설정범위: 1~65,535)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

rtn = pmc4bpci_set_acac(MMC_CARD_NO,
PMC4BPCI_AXIS_X|PMC4BPCI_AXIS_Y|PMC4BPCI_AXIS_Z|PMC4BPCI_AXIS_U, 100);

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
```

```
}
```

(5) 참고매크로

PMC4BPCI_CMD_K

7.2 set_dcac

MMC_INT16U **pmc4bpci_set_dcac**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT16U *wdata*);

(1) 설명

가감속도를 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위:0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 가감속도 설정값(L), 가감속도=((62.5*10^6)/L)*배율 (설정범위: 1~65,535)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

rtn = pmc4bpci_set_dcac(MMC_CARD_NO,
PMC4BPCI_AXIS_X|PMC4BPCI_AXIS_Y|PMC4BPCI_AXIS_Z|PMC4BPCI_AXIS_U, 100);

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
}
```

(5) 참고매크로

PMC4BPCI_CMD_AD

7.3 set_acc

MMC_INT16U **pmc4bpci_set_acc**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT16 *wdata*);

(1) 설명

가속도를 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC4BPCIP 는 4 개의 축으로 이루어져 있습니다.
- wdata: 가속도 설정값(A), 가속도 = A*125*배율 (설정범위: 1~8,000)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

rtn = pmc4bpci_set_acc(MMC_CARD_NO,
PMC4BPCI_AXIS_X|PMC4BPCI_AXIS_Y|PMC4BPCI_AXIS_Z|PMC4BPCI_AXIS_U, 100);

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
}
```

(5) 참고매크로

PMC4BPCI_CMD_D

(6) 참고함수

pmc4bpci_set_dec

7.4 set_dec

MMC_INT16U **pmc4bpci_set_dec**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT16 *wdata*);

(1) 설명

감속도를 설정합니다.

(2) 인자

- Id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 감속도 설정값(D), 감속도 = D*125*배율 (설정범위: 1~8,000)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

rtn = pmc4bpci_set_dec(MMC_CARD_NO,
PMC4BPCI_AXIS_X|PMC4BPCI_AXIS_Y|PMC4BPCI_AXIS_Z|PMC4BPCI_AXIS_U, 100);

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
}
```

(5) 참고매크로

PMC4BPCI_CMD_A

(6) 참고함수

pmc4bpci_set_acc

7.5 set_startv

MMC_INT16U **pmc4bpci_set_startv**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT16 *wdata*);

(1) 설명

선택한 축의 시작속도를 설정합니다.

(2) 인자

- Id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdat: 시작속도 설정값(SV), 시작속도 = SV*배율 (설정범위: 1~8,000)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"


void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn=pmc4bpci_set_startv(MMC_CARD_NO,PMC4BPCI_AXIS_X| PMC4BPCI_AXIS_Y,
                                                        1000);
// ……

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
```

```
}
```

(5) 참고매크로

PMC4BPCI_CMD_SV

7.6 set_speed

MMC_INT16U **pmc4bpci_set_speed**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT16 *wdata*);

(1) 설명

드라이브 속도를 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 속도 설정값(V), 드라이브 속도 = V*배율 (설정범위 1~8,000)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn=pmc4bpci_set_speed(MMC_CARD_NO, MC4BPCI_AXIS_X|PMC4BPCI_AXIS_Y,
1000);
// ……

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
```

```
}
```

(5) 참고매크로

PMC4BPCI_CMD_V

7.7 set_pulse, set_endpoint

MMC_INT16U pmc4bpci_set_pulse(MMC_INT16U id, MMC_INT16U axis,MMC_INT32 wdata);

MMC_INT16U pmc4bpci_set_endpoint(MMC_INT16U id, MMC_INT16U axis,MMC_INT32 wdata);

(1) 설명

출력 pulse 수를 지정합니다. 보간에서는 종점의 위치를 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 펄스 수/보간 종점 위치
  (설정범위: pulse 0~4,294,967,295, 보간종점 -2,147,483,646~2,147,483,646)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예 1

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn=pmc4bpci_set_pulse(MMC_CARD_NO, PMC4BPCI_AXIS_X, 100000);// 10 만 펄스
pmc4bpci_command(MMC_CARD_NO, PMC4BPCI_AXIS_X, PMC4BPCI_DRV_MF);
//-방향 정량 drive
// ……
```

```
        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
}
```

(5) 사용 예 2

```
    // 보간 종점위치 설정
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
pmc4bpci_outw(MMC_CARD_NO, wr5, WR5_INP_LSPD0 | 0x0104);
// 선속일정모드+보간축 선택
pmc4bpci_set_range(MMC_CARD_NO, PMC4BPCI_AXIS_X,4000000);// x2
pmc4bpci_set_range(MMC_CARD_NO, PMC4BPCI_AXIS_Y,5656000);
                                        // 4000000*1.414=5656000
pmc4bpci_set_startv(MMC_CARD_NO, PMC4BPCI_AXIS_X,500);          // x2 =1000pps
pmc4bpci_set_speed(MMC_CARD_NO, PMC4BPCI_AXIS_X,500);          // x2 =1000pps
pmc4bpci_set_center(MMC_CARD_NO, PMC4BPCI_AXIS_X,5000);
//중심축 X=5000
pmc4bpci_set_center(MMC_CARD_NO, PMC4BPCI_AXIS_Y,0);            // 중심축 Y=0
pmc4bpci_set_endpoint(MMC_CARD_NO, PMC4BPCI_AXIS_X,5000);
// X end point
pmc4bpci_set_endpoint(MMC_CARD_NO, PMC4BPCI_AXIS_Y,-5000);
// Y end point
pmc4bpci_command(MMC_CARD_NO, 0x0,PMC4BPCI_INP_CW);
// CW 원호보간드라이브 시작
// ……

                printf("Complete OK!\n");

                pmc4bpci_close_all();
}
```


(6) 참고매크로

PMC4BPCI_INP_CCW, PMC4BPCI_INP_CW, PMC4BPCI_CMD_P

7.8 set_decpoint

MMC_INT16U **pmc4bpci_set_decpoint**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32 *wdata*);

(1) 설명

매뉴얼 감속점을 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 매뉴얼 감속점 (설정범위: 0~4,294,967,295)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn=pmc4bpci_set_decpoint(MMC_CARD_NO, PMC4BPCI_AXIS_X, 20000);
// ……

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
}
```

(5) 참고매크로

PMC4BPCI_CMD_DP

7.9 set_center

MMC_INT16U **pmc4bpci_set_center**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32 *wdata*);

(1) 설명

원호 중심점을 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 원호 중심점(현재위치를 기준으로 합니다)
  (설정범위: -2,147,483,648~2,147,483,647)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn=pmc4bpci_set_center(MMC_CARD_NO, PMC4BPCI_AXIS_X, 20000);
rtn=pmc4bpci_set_center(MMC_CARD_NO, PMC4BPCI_AXIS_Y, -20000);
// ……

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }
```



```
        pmc4bpci_close_all();
    }
```

(5) 참고매크로

PMC4BPCI_CMD_C

7.10 set_lpcounter

MMC_INT16U **pmc4bpci_set_lpcounter**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32 *wdata*);

(1) 설명

논리위치 counter 값을 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 논리위치 counter 값(설정범위: –2,147,483,648~2,147,483,647)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
pmc4bpci_set_lpcounter(MMC_CARD_NO, PMC4BPCI_AXIS_X|PMC4BPCI_AXIS_Y, 0);
// ……

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
}
```


(5) 참고매크로

PMC4BPCI_CMD_LP

(6) 참고함수

pmc4bpci_set_epcounter

7.11 set_epcounter

MMC_INT16U **pmc4bpci_set_epcounter**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32 *wdata*);

(1) 설명

실제위치 counter 값을 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 실제위치 counter 값(설정범위: –2,147,483,648~2,147,483,647)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }
// ……
rtn=pmc4bpci_set_epcounter(MMC_CARD_NO,PMC4BPCI_AXIS_X|PMC4BPCI_AXIS_Y
,0);
// ……

        if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

        pmc4bpci_close_all();
}
```


(5) 참고매크로

PMC4BPCI_CMD_EP

(6) 참고함수

pmc4bpci_set_lpcounter

7.12 set_compplus

MMC_INT16U **pmc4bpci_set_compplus**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32 *wdata*);

(1) 설명

COMP+ register 의 범위를 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: COMP + register 설정값(설정범위: –2,147,483,648~2,147,483,647)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn=pmc4bpci_set_compplus(MMC_CARD_NO, PMC4BPCI_AXIS_X, 10000);
// COMP+ register  설정
// ……

if(rtn!=MMC_OK)
{printf(“Fail!\n”);     }
else
{printf(“Complete OK!\n”);    }
```



```
pmc4bpci_close_all();

}
```

(5) 참고매크로

PMC4BPCI_CMD_CP, PMC4BPCI_CMD_CM

7.13 set_compminus

MMC_INT16U **pmc4bpci_set_compminus**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32 *wdata*);

(1) 설명

COMP- register 의 범위를 설정합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: COMP - register 설정값 (설정범위: –2,147,483,648~2,147,483,647)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn=pmc4bpci_set_compminus(MMC_CARD_NO, PMC4BPCI_AXIS_X, 10000);
// COMP- register  설정
// ……

if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }
```



```
pmc4bpci_close_all();

}
```

(5) 참고매크로

PMC4BPCI_CMD_CP, PMC4BPCI_CMD_CM

7.14 set_accoffset

MMC_INT16U **pmc4bpci_set_accoffset**(MMC_INT16U *id*, MMC_INT16U *axis*, MMC_INT32 *wdata*);

(1) 설명

가속 counter offset 을 설정합니다. Reset 시에는 8 이 설정됩니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- wdata: 가속 counter offset 값(설정범위: 0~65,535),

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn= pmc4bpci_set_accoffset (MMC_CARD_NO, PMC4BPCI_AXIS_X, 8);
// 가속 counter offset 설정
// ……

if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }
```



```
pmc4bpci_close_all();

}
```

(5) 참고매크로

PMC4BPCI_CMD_AO

8 계산

8.1 calc_scale

MMC_INT32U **pmc4bpci_calc_scale**(MMC_INT32U *range*);

(1) 설명

범위 배율계산. 이 함수는 pmc4bpci_open()함수를 호출하기 전에 사용해도 됩니다.

(2) 인자

드라이브 범위값 *range* 를 입력합니다. (범위: 16,000~8,000,000)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 최소값, 최대값 벗어나면 MMC_ILLEGAL_PARAMETER 를 리턴합니다. 실행에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U scale;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        // 입력값이 800000 이므로 리턴값은 10 입니다.
        scale = pmc4bpci_calc_scale(800000);

        printf("%d\n", scale);

        pmc4bpci_close_all();
}
```

(5) 참고매크로

PMC4BPCI_RANGE

8.2 calc_acceleration

MMC_INT32U **pmc4bpci_calc_acceleration**(MMC_INT16U *acc*, MMC_INT32U *range*);

(1) 설명

가속도계산. 이 함수는 pmc4bpci_open()함수를 호출하기 전에 사용해도 됩니다.

(2) 인자

- acc: 가속도 설정값을 입력합니다. (범위: 1~8,000)
- range: 드라이브 범위를 입력합니다. (범위: 16,000~8,000,000)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 가속도(*acc*) 및 범위값(*range*)의 범위를 벗어나면 MMC_ILLEGAL_PARAMETER 를 리턴합니다. 실행에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT32U ac;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        // 가속도=acc*125*(8000000/range)
        ac = pmc4bpci_calc_acceleration(100, 800000);

        printf("가속도 : %d\n", ac);

        pmc4bpci_close_all();
}
```

(5) 참고매크로

PMC4BPCI_RANGE

8.3 calc_velocity

MMC_INT16U **pmc4bpci_calc_velocity**(MMC_INT16U *vel*, MMC_INT32U *range*);

(1) 설명

속도계산. 이 함수는 pmc4bpci_open()함수를 호출하기 전에 사용해도 됩니다.

(2) 인자

- vel: 속도값을 입력합니다. (범위: 1~8,000)
- range: 드라이브 범위를 입력합니다. (범위: 16,000~8,000,000)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 속도(vel) 및 범위값(range)의 범위를 벗어나면 PMC4BPCI_ILLEGAL_PARAMETER 를 리턴합니다. 실행에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U vel;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        //  속도=vel*(8000000/range)=1000 PPS
        vel = pmc4bpci_calc_velocity(100, 800000);

        printf("속도  : %d\n", vel);

        pmc4bpci_close_all();
}
```

(5) 참고매크로

PMC4BPCI_RANGE

8.4 calc_acac

MMC_INT16U **pmc4bpci_calc_acac**(MMC_INT16U *acac*, MMC_INT32U *range*);

(1) 설명

가가속도계산. 이 함수는 pmc4bpci_open()함수를 호출하기 전에 사용해도 됩니다.

(2) 인자

- acac: 가가속도값을 입력합니다. (범위: 1~65,535)
- range: 드라이브 범위를 입력합니다. (범위: 16,000~8,000,000)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 가가속도(*acac*) 및 범위값(*range*)의 범위를 벗어나면 PMC4BPCI_ILLEGAL_PARAMETER 를 리턴합니다. 실행에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT32U acac;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        // 가가속도=(62500000/acac) * (8000000/range) = 100000 PPS/SEC2
        acac = pmc4bpci_calc_acac(6250, 800000);

        printf("가가속도 : %d\n", acac);

        pmc4bpci_close_all();
}
```

(5) 참고매크로

PMC4BPCI_RANGE

8.5 util_math_dist

MMC_DOUBLE **util_math_dist**(MMC_VERTEX* *p1*, MMC_VERTEX* *p2*);

(1) 설명

점 p1 과 p2 의 거리를 구합니다.

이 함수는 pmc4bpci_open()함수를 호출하기 전에 사용해도 됩니다.

(2) 인자

- p1: 첫번째 점
- p2: 두번째 점

```
typedef struct {
        double x;               // X 좌표
        double y;               // Y 좌표
        union {
                double z;       // Z 좌표
                double k;       // 폴리라인 호의 bulge 값
        };
} MMC_VERTEX;
```

(3) 리턴값

두점사이의 거리를 리턴합니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"
#define MMC_CARD_NO     15

void main()
{
        MMC_DOUBLE dist;
        int OpenFlag;

        // p1 X 좌표 : 0, Y 좌표 0
        // p2 X 좌표 : 0, Y 좌표 5
        MMC_VERTEX p1 = {0,0}, p2={0,5};

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        dist = util_math_dist(&p1, &p2);
```

```
        printf("p1 , p2  거리  : %f\n", dist);
        pmc4bpci_close_all();
}
```

(5) 참고

MMC_VERTEX

8.6 util_math_angle

MMC_DOUBLE **util_math_angle**(MMC_VERTEX* *p1*, MMC_VERTEX* *p2*);

(1) 설명

점 p1 을 기준으로 점 p2 로의 각도를 구합니다.

이 함수는 pmc4bpci_open()함수를 호출하기 전에 사용해도 됩니다.

(2) 인자

- p1: 첫번째 점
- p2: 두번째 점

(3) 리턴값

두점사이의 각도를 리턴합니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_DOUBLE dist;
        int OpenFlag;

        // p1 X 좌표  : 1, Y 좌표  4
        // p2 X 좌표  : 5, Y 좌표  2
        MMC_VERTEX p1 = {1,4}, p2={5,2};

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        dist = util_math_angle(&p1, &p2);

        printf("p1 , p2  사이의  각  : %f\n", dist);

        pmc4bpci_close_all();
}
```

(5) 참고

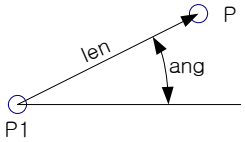
MMC_VERTEX, MMC_PI

8.7 util_math_polar

MMC_VOID **util_math_polar**(MMC_VERTEX* *p1*, MMC_DOUBLE *ang*, MMC_DOUBLE *len*, MMC_VERTEX* *p*);

(1) **설명**

점 *p1* 을 기준점으로 하고 ang 방향의, 길이 len 만큼 떨어져 있는 지점의 좌표를 구합니다. 이 함수는 pmc4bpci_open()함수를 호출하기 전에 사용해도 됩니다.



(2) **인자**

- p1: 기준점
- ang: 방향(Radian)
- len: 떨어진 거리
- p: 리턴값(점좌표)

(3) **리턴값**

점좌표를 리턴합니다. 함수 자체의 리턴값은 없습니다.

(4) **사용 예**

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15
#define MMC_PI     3.14159265358979323846

void main()
{
        int OpenFlag;

        MMC_VERTEX p1 = {0,0}, p;
        MMC_DOUBLE dist = 10.0;
        MMC_DOUBLE angle = MMC_PI * 30 / 180;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        // p1 을 기준점으로 30 도방향 10 만큼 떨어져 있는 점을 구합니다.
        util_math_polar(&p1, angle, dist, &p);
```

```
        printf("X 좌표 : %f, Y 좌표 : %f\n", p.x, p.y);
        pmc4bpci_close_all();
}
```

(5) 참고

MMC_VERTEX, MMC_PI

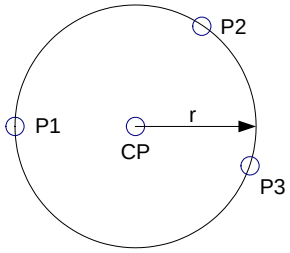
8.8 util_math_circle_3p

MMC_VOID **util_math_circle_3p**(MMC_VERTEX* *p1*, MMC_VERTEX* *p2*, MMC_VERTEX* *p3*, MMC_VERTEX* *cp*, MMC_DOUBLE* *r*);

(1) 설명

세점 p1, p2, p3 를 지나는 원의 중심점 cp 및 반지름 r 을 구합니다.

이 함수는 pmc4bpci_open()함수를 호출하기 전에 사용해도 됩니다.



(2) 인자

- p1: 원을 지나는 1 번째 점
- p2: 원을 지나는 2 번째 점
- p3: 원을 지나는 3 번째 점
- cp: 리턴값(중심점 좌표)
- r: 리턴값(반지름)

(3) 리턴값

계산결과인 중심점좌표 *cp* 및 반지름 *r* 을 리턴합니다. 함수 자체의 리턴값은 없습니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"
#define MMC_CARD_NO     15

void main()
{
        int OpenFlag;
        MMC_VERTEX p1 = {5,5}, p2 = {5,-5}, p3={10,0}, cp;
        MMC_DOUBLE r;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        util_math_circle_3p(&p1, &p2, &p3, &cp, &r);
```



```
        printf("중심점 X 좌표: %f, 중심점 Y 좌표: %f\n", cp.x, cp.y);
        printf("중심점에서 반지름 거리: %f\n", r);
        pmc4bpci_close_all();
}
```

(5) 참고

MMC_VERTEX, MMC_PI

9 상태읽기

9.1 check_valid_id

MMC_INT16U **pmc4bpci_check_valid_id**(MMC_INT16 *id*);

(1) 설명

입력받은 id 값이 유효한지 검사합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

id 번호에 해당되는 Handle 값이 존재할 경우는 MMC_OK 를 리턴합니다. 존재하지 않을 경우는 MMC_ERROR 를 리턴합니다. 성공할 경우는 전역변수 gError 값에 MMC_OK 값이 실패할 경우 MMC_ERROR 값이 설정됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }
        // ……
        rtn = pmc4bpci_check_valid_id(MMC_CARD_NO);
        if(rtn==MMC_ERROR) {
                printf("ERROR: Invalid ID number\n");
        } else {
                printf("Found!\n");
        }
        // ……
        pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_open, pmc4bpci_close_all

9.2 get_logicalposition

MMC_INT32 **pmc4bpci_get_logicalposition**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

현재 논리위치를 읽습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: PMC4BPCI 의 4 개 축에서 1 개의 축만을 선택합니다.

(3) 리턴값

읽기에 성공하면 MMC 보드(PMC-4B-PCI 보드)의 칩셋인 MCX314 에 의해 내부에서 카운트되는 논리위치를 리턴합니다. PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다. (읽기 성공이면 MMC_OK 가 기록됩니다.)

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
long x=pmc4bpci_get_logicalposition(MMC_CARD_NO, PMC4BPCI_AXIS_X);
long y=pmc4bpci_get_logicalposition(MMC_CARD_NO, PMC4BPCI_AXIS_Y);
// ……

printf("Complete OK!\n");

pmc4bpci_close_all();

}
```

(5) 참고매크로

PMC4BPCI_READ_LP

9.3 get_encoderposition

MMC_INT32 **pmc4bpci_get_encoderposition**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

현재 엔코더의 실제위치를 읽습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: PMC4BPCI 의 4 개 축에서 1 개의 축만을 선택합니다.

(3) 리턴값

읽기에 성공하면 PMC-4B-PCI 보드의 외부에 연결된 엔코더에 의해 카운트되는 실제위치를 리턴합니다. PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다. (읽기 성공이면 MMC_OK 가 기록됩니다.)

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
long x=pmc4bpci_get_encoderposition(MMC_CARD_NO, PMC4BPCI_AXIS_X);
long y=pmc4bpci_get_encoderposition(MMC_CARD_NO, PMC4BPCI_AXIS_Y);
// ……

printf("Complete OK!\n");

pmc4bpci_close_all();

}
```

(5) 참고매크로

PMC4BPCI_READ_EP

9.4 get_currentvelocity

MMC_INT16U **pmc4bpci_get_currentvelocity**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

현재 속도를 읽습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: PMC-4B-PCI 의 4 개 축에서 1 개의 축만을 선택합니다.

(3) 리턴값

읽기에 성공하면 PMC-4B-PCI 보드의 칩셋인 MCX314 에 의해 설정되어 있는 현재 속도값을 리턴합니다. PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다. (읽기 성공이면 MMC_OK 가 기록됩니다.)

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U velocity;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
velocity=pmc4bpci_get_currentvelocity(MMC_CARD_NO, PMC4BPCI_AXIS_X);
// ……

printf("Complete OK!\n");

pmc4bpci_close_all();

}
```

(5) 참고매크로

PMC4BPCI_READ_CV

9.5 get_currentacc

MMC_INT16U **pmc4bpci_get_currentacc**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

현재 가속도를 읽습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: PMC4BPCI 의 4 개 축에서 1 개의 축만을 선택합니다.

(3) 리턴값

읽기에 성공하면 PMC4BPCI 의 칩셋인 MCX314 에 의해 설정되어 있는 현재 가속도값을 리턴합니다. PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다. (읽기 성공이면 MMC_OK 가 기록됩니다.)

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U acc;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
acc = pmc4bpci_get_currentacc (MMC_CARD_NO, PMC4BPCI_AXIS_X);
// ……

printf("Complete OK!\n");

pmc4bpci_close_all();

}
```

(5) 참고매크로

PMC4BPCI_READ_CA

9.6 is_drive

MMC_INT16U **pmc4bpci_is_drive**(MMC_INT16U id, MMC_INT16U *axis*);

(1) 설명

각 축의 드라이브 상태를 표시합니다.

되돌림 값이 high level(1)이면 선택축의 드라이브 펄스가 출력중임을 나타내고 low level(0)을 되돌리면 선택축의 드라이브를 종료하고 있는 중임을 나타냅니다..

pmc4bpci_wait()함수와 **pmc4bpci_is_drive**()함수는 RR0 레지스터의 동일 비트를 검사하지만 pmc4bpci_wait()함수는 드라이브 펄스의 완료신호(low level(0))를 기다리고 **pmc4bpci_is_drive**()함수는 완료 신호를 기다리지 않고 현재의 상태만을 바로 리턴합니다.

서보모터 위치결정 완료용 입력신호(nINPOS)가 유효로 설정되어 있을 때는 드라이브 펄스 출력신호(nINPOS)가 활성화(active)후에 low level(0)으로 되돌아 갑니다..

위의 함수를 사용하지 않고 RR0 레지스터의 D0~D3 번 비트를 참조하여 검사 할 수 있습니다. (D0=X 축, D1=Y 축, D2=Z 축, D3=U 축)

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다.

출력상태의 논리가 high level(1) 이면 MMC_HIGH_LEVEL 를 low level(0) 이면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에는 성공과 에러에 관련된 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
```

```
                printf("Initialize error!\n");
                return;
        }

rtn = pmc4bpci_is_drive(MMC_CARD_NO,PMC4BPCI_AXIS_X+PMC4BPCI_AXIS_Y);

        if(rtn!=MMC_OK)
        {       printf("Not Move!\n");      }
        else
        {       printf("Move!\n");  }
                pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_wait

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL

9.7 is_idrive

MMC_INT16U **pmc4bpci_is_idrive**(MMC_INT16U *id*);

(1) 설명

보간 드라이브 상태를 표시합니다.

보간 드라이브 중이면 RR0 의 D8 번 비트가 high level(1), 그렇지 않으면 low level(0)입니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

보간 드라이브 상태의 논리가 high level 이면 MMC_HIGH_LEVEL 을 low level 이면 MMC_LOW_LEVEL 을 리턴합니다. 전역 에러 검사 변수인 gError 에는 성공과 에러에 관련된 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

rtn = pmc4bpci_is_idrive(MMC_CARD_NO);

        if(rtn!=MMC_OK)
        {       printf("Not Move!\n");          }
        else
        {       printf("Move!\n");  }

                pmc4bpci_close_all();
}
```

(5) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR0_I_DRV

9.8 flag_error

MMC_INT16U **pmc4bpci_flag_error**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

지정한 축의 에러 발생 상태를 표시합니다.

RR0 의 D4~D7 번 비트중에 지정한 축을 검사하여 한축이라도 high level(1)이 되면 **pmc4bpci_flag_error**()는 에러를 리턴합니다.(D4=X 축, D5=Y 축, D6=Z 축, D7=U 축)

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 보간 드라이브 상태의 논리레벨이 high 이면 MMC_HIGH_LEVEL 를 low 이면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에는 성공과 에러에 관련된 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
                        MMC_INT16U    rtn;
                        int OpenFlag;

                        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
                        if(OpenFlag!=MMC_OK)
                        {
                                printf("Initialize error!\n");
                                return;
                        }

rtn = pmc4bpci_flag_error(MMC_CARD_NO,
            PMC4BPCI_AXIS_X|PMC4BPCI_AXIS_Y);

                        if(rtn!=MMC_OK)
                        {       printf("Not Error!\n");         }
                        else
                        {       printf("Error!\n");   }

                        pmc4bpci_close_all();
}
```


(5) 참고함수

pmc4bpci_is_error , Pmc4bpci_error_slimitplus, MMC_ERROR_slimitminus, MMC_ERROR_hlimitplus, MMC_ERROR_hlimitminus, MMC_ERROR_servoalarm, MMC_ERROR_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR0_X_ERR, RR0_Y_ERR, RR0_Z_ERR, RR0_U_ERR, RR2_SLMTP, RR2_SLMTM, RR2_HLMTP, RR2_HLMTM, RR2_ALARM, RR2_EMG

9.9 error

MMC_INT16U **pmc4bpci_error**();

(1) 설명

에러상태를 리턴하는 함수입니다.

(2) 리턴값

MMC_OK	1
MMC_FALSE	0
MMC_IOADDRESS_ERR	6
MMC_OPEN_ERR	5
MMC_TIMEOUT_ERR	7
MMC_INVALID_AXIS	8
MMC_ILLEGAL_PARAMETER	9
MMC_ZERO_PARAMETER	10
MMC_ERROR	11
MMC_QUIT	12
MMC_INVALID_CARD	13

(3) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_error ();
        printf("에러 값 : %d\n", level);
        pmc4bpci_close_all();
}
```

(4) 참고함수

pmc4bpci_error_slimitplus, pmc4bpci_error_slimitminus,
pmc4bpci_error_hlimitplus, pmc4bpci_error_hlimitminus,
pmc4bpci_error_servoalarm, pmc4bpci_error_emergency

(5) 참고매크로

MMC_OK, MMC_FALSE, MMC_IOADDRESS_ERR, MMC_OPEN_ERR,
MMC_TIMEOUT_ERR, MMC_INVALID_AXIS, MMC_ILLEGAL_PARAMETER,
MMC_ZERO_PARAMETER, MMC_ERROR, MMC_QUIT

9.10 flag_nextcommand

MMC_INT16U **pmc4bpci_flag_nextcommand**(MMC_INT16U *id*);

(1) 설명

연속보간에서 다음 데이터의 기입 가능임을 표시합니다.
연속보간에서 RR0 의 D9 (CNEXT)가 high level(1)이 되면 다음 노드를 위해 파라미터 및 보간명령을 기입할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 연속보간에서 다음 데이터의 기입 가능이면 MMC_HIGH_LEVEL 를 기입 불가능이면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에는 MMC_OK 로 설정됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        rtn = pmc4bpci_flag_nextcommand(MMC_CARD_NO);

        if(rtn!=MMC_OK)
        {       printf("Possible!\n");          }
        else
        {       printf("Impossible!\n");        }

                pmc4bpci_close_all();
}
```

(5) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR0_CNEXT

9.11 is_zone

MMC_INT16U **pmc4bpci_is_zone**(MMC_INT16U *id*);

(1) 설명

원호보간 드라이브에서 현재 드라이브위치를 나타냅니다.
현재의 위치를 RR0 의 D10~D12 번 비트로 표시하며 값의 의미는 다음과 같습니다.
아래 표에서 각각의 숫자는 그림의 원호보간 드라이브위치와 일치합니다.

표. 원호보간 드라이브 위치

D12	D11	D10	원호보간 드라이브위치
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

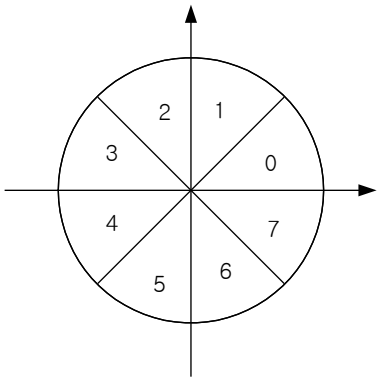


그림. 원호보간 드라이브 위치

(2) 인자

id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 연속보간에서 다음 데이터의 기입 가능이면 MMC_HIGH_LEVEL 를 기입 불가능이면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에는 PMC4BPCI_OK 로 설정됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        rtn = pmc4bpci_is_zone(MMC_CARD_NO);

        printf("원호보간 현재 위치 : %d\n", rtn);

        pmc4bpci_close_all();
}
```

(5) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR0_CNEXT, RR0_ZONE0, RR0_ZONE1, RR0_ZONE2

9.12 is_bpstackcounter

MMC_INT16U **pmc4bpci_is_bpstackcounter**(MMC_INT16U *id*);

(1) 설명

비트패턴 보간 드라이브에서 현재 스택카운터(Stack Counter(SC))의 위치를 나타냅니다. SC 의 위치를 RR0 의 D13, D14 번 비트로 표시하며 값의 의미는 아래 표와 같습니다.

표. Stack Counter(SC)의 값

D14	D13	Stack Counter(SC)의 값
0	0	0
0	1	1
1	0	2
1	1	3

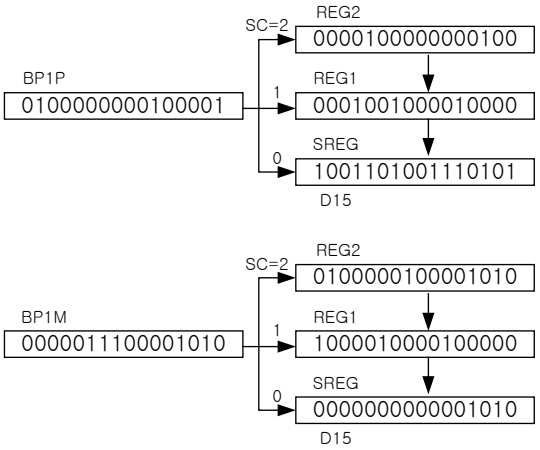


그림. 비트패턴 보간의 레지스터구성

비트패턴의 데이터를 BP1P 16 비트 레지스터에 기록하면 SC 의 값에 따라 기록되는 레지스터가 바뀝니다.
위의 예에서 SC=2 이면 BP1P 레지스터 내용이 REG2 레지스터로 이동하고 SC=1 이면 REG1 으로 이동합니다. SC=0 이면 BP1P 레지스터 내용이 SREG 레지스터로 이동합니다. PMC4BPCI 의 MCX314 칩은 SREG 레지스터 값을 하위비트부터 순서대로 드라이브펄스 출력합니다. SREG 레지스터의 내용이 모두 출력되고 SC 값이 0 보다 크다면 16 비트 레지스터의 내용은 REG2 → REG1 → SREG 레지스터의 순으로 이동합니다. BP1P 는 +방향/BP1M 은 –방향을 의미합니다.

(2) 인자

id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 비트패턴 보간에서 Stack Counter(SC)의 값을 리턴합니다. 전역 에러 검사 변수인 gError 에는 PMC4BPCI_OK 로 설정됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        rtn = pmc4bpci_is_bpstackcounter(MMC_CARD_NO);

        printf("Stack Counter(SC)  값  : %d\n", rtn);

        pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_is_bpdrive

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR0_BPSC0, RR0_BPSC1

9.13 is_bpdrive

MMC_INT16U **pmc4bpci_is_bpdrive**(MMC_INT16U *id*);

(1) 설명

비트패턴 보간 드라이브에서 비트패턴 데이터 기록가능 여부를 검사합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 비트패턴 보간에서 스택이 꽉 차있으면 MMC_HIGH_LEVEL 를 리턴하고, 스택이 1 개라도 비어있으면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에는 PMC4BPCI_OK 로 설정됩니다.

(4) 사용 예

```
 #include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        rtn = pmc4bpci_is_bpdrive (MMC_CARD_NO);

        if(rtn == MMC_OK)                       // 비트패턴에 데이터 기록을 할 수
없다면
        { printf("비트패턴데이터 기록 불가능!\n"); }
        else                                    // 비트패턴에 데이터 기록을 할 수 있다면
        { printf("비트패턴데이터 기록 가능!\n"); }

        printf("Stack Counter(SC) 값 : %d\n", rtn);

        pmc4bpci_close_all();
```

```
}
```

(5) 참고함수

pmc4bpci_is_bpstackcounter

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR0_BPSC0, RR0_BPSC1

9.14 error_servoalarm

MMC_INT16U **pmc4bpci_error_servoalarm**(MMC_INT16U *id*);

(1) 설명

서보모터용 알람신호(nALARM)가 유효설정에서 active level 로 되었을 때 high level(1)를 리턴합니다.

pmc4bpci_is_error()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_error_servoalarm**() 함수를 사용하십시오. pmc4bpci_is_error()함수를 사용하지 않고 **pmc4bpci_error_servoalarm**()함수를 사용하고자 한다면 축 선택을 먼저 하여야 합니다. RR2 레지스터의 D4(ALARM) 비트에 의해서 검출할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 서보모터용 알람신호(nALARM)가 유효설정에서 active level 로 되면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

pmc4bpci_error_slimitplus() 함수 참조.

(5) 참고함수

pmc4bpci_error_slimitplus, pmc4bpci_error_slimitminus,
pmc4bpci_error_hlimitplus, pmc4bpci_error_hlimitminus,
pmc4bpci_error_servoalarm, pmc4bpci_error_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR2_SLMTP, RR2_SLMTM, RR2_HLMTP, RR2_HLMTM, RR2_ALARM, RR2_EMG

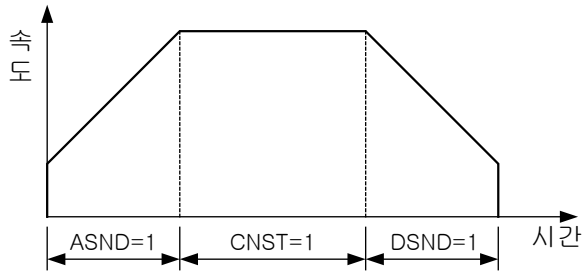
9.15 flag_aacc

MMC_INT16U **pmc4bpci_flag_aacc**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

가감속 drive 에서 가속여부를 판단합니다.

가속이면 high level(1)을 리턴합니다. RR1 레지스터의 D2(ASND)비트로 판단해도 됩니다.



(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다.

가감속 드라이브에서 가속중이면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_flag_aacc(MMC_CARD_NO, PMC4BPCI_AXIS_X);
```



```
        if(level == MMC_OK)        // 가감속 드라이브에서 가속중이면
        { printf("가속 드라이브중!\n"); }
pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_flag_aacc, pmc4bpci_flag_cons, pmc4bpci_flag_dacc

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_ASND, RR1_CNST, RR1_DSND

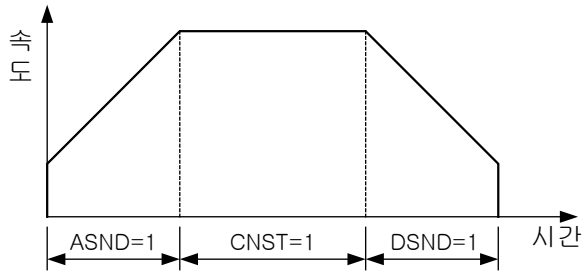
9.16 flag_cons

MMC_INT16U **pmc4bpci_flag_cons**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

가감속 drive 에서 정속여부를 판단합니다.

정속이면 high level(1)을 리턴합니다. RR1 레지스터의 D3(CNST)비트로 판단해도 됩니다.



(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다.

가감속 드라이브에서 정속중이면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

 level = pmc4bpci_flag_cons(MMC_CARD_NO, PMC4BPCI_AXIS_X);
```

```
 if(level == MMC_OK)        // 가감속 드라이브에서 정속중이면
 { printf("정속 드라이브중!\n"); }
 pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_flag_aacc, pmc4bpci_flag_cons, pmc4bpci_flag_dacc

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_ASND, RR1_CNST, RR1_DSND

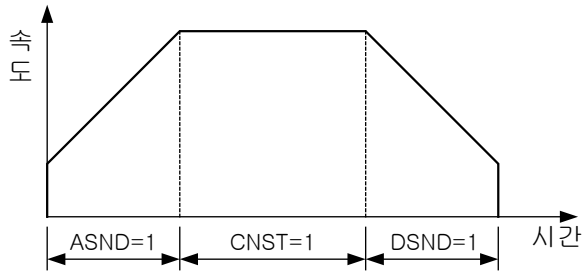
9.17 pmc4bpci_flag_dacc

MMC_INT16U **pmc4bpci_flag_dacc**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

가감속 drive 에서 감속여부를 판단합니다.

감속이면 high level(1)을 리턴합니다. RR1 레지스터의 D4(DSND)비트로 판단해도 됩니다.



(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 가감속 드라이브에서 감속중이면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }
```

```
 level = pmc4bpci_flag_dacc(MMC_CARD_NO, PMC4BPCI_AXIS_X);

 if(level == MMC_OK)        // 가감속 드라이브에서 감속중이면
 { printf("감속 드라이브중!\n"); }
 pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_flag_aacc, pmc4bpci_flag_cons, pmc4bpci_flag_dacc

(6) 참고매크로

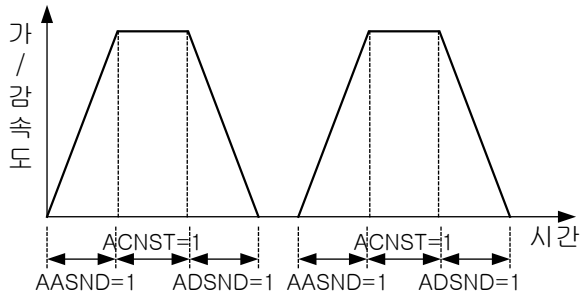
MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_ASND, RR1_CNST, RR1_DSND

9.18 flag_aacac

MMC_INT16U **pmc4bpci_flag_aacac**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

S 자 가감속 드라이브에서 가속도/감속도의 증가여부를 판단합니다.
증가하면 high level(1)을 리턴합니다. RR1 레지스터의 D5(AASND)비트로 판단해도 됩니다.



(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. S 자 가감속 드라이브에서 가속도/감속도값이 증가하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
```

```
        }

 level = pmc4bpci_flag_aacac(MMC_CARD_NO, PMC4BPCI_AXIS_X);

 if(level == MMC_OK)        // S 자  가감속 drive 에서  가속도/감속도가  증가할때
 { printf("가속도/감속도  증가!\n"); }
 pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_flag_aacac, pmc4bpci_flag_acons, pmc4bpci_flag_dacac

(6) 참고매크로

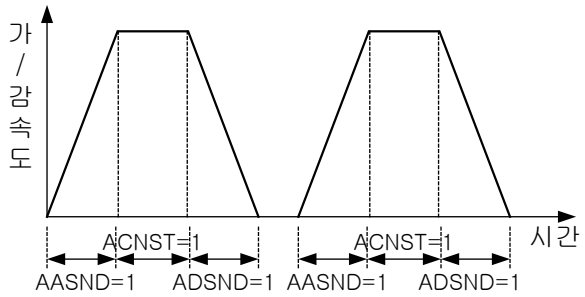
MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_AASND, RR1_ACNST, RR1_ADSND

9.19 flag_acons

MMC_INT16U **pmc4bpci_flag_acons**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

S 자 가감속 드라이브에서 가속도/감속도의 증가여부를 판단합니다.
일정하면 high level(1)을 리턴합니다. RR1 레지스터의 D6(ACNST)비트로 판단해도 됩니다.



(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. S 자 가감속 드라이브에서 가속도/감속도값이 일정하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
```

```
        }

 level = pmc4bpci_flag_acons(MMC_CARD_NO, PMC4BPCI_AXIS_X);

 if(level == MMC_OK)        // S 자  가감속 drive 에서  가속도/감속도가  일정할때
 { printf("가속도/감속도  일정!\n"); }
 pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_flag_aacac, pmc4bpci_flag_acons, pmc4bpci_flag_dacac

(6) 참고매크로

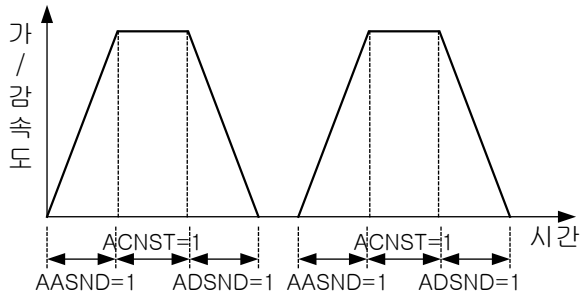
MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_AASND, RR1_ACNST, RR1_ADSND

9.20 flag_dacac

MMC_INT16U **pmc4bpci_flag_dacac**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

S 자 가감속 드라이브에서 가속도/감속도의 증가여부를 판단합니다.
감소하면 high level(1)을 리턴합니다. RR1 레지스터의 D7(ADSND)비트로 판단해도 됩니다.



(2) 인자

- id: ID 번호를 입력합니다. (범위:0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. S 자 가감속 드라이브에서 가속도/감속도값이 감소하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
```

```
        }

 level = pmc4bpci_flag_dacac(MMC_CARD_NO, PMC4BPCI_AXIS_X);

 if(level == MMC_OK)        // S 자  가감속 drive 에서  가속도/감속도가  감소할때
 { printf("가속도/감속도  감속!\n"); }
pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_flag_aacac, pmc4bpci_flag_acons, pmc4bpci_flag_dacac

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_AASND, RR1_ACNST, RR1_ADSND

9.21 flag_compp

MMC_INT16U **pmc4bpci_flag_compp**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

논리/실제위치 카운터와 COMP+레지스터의 대소관계를 나타냅니다.
high level(1): 논리/실제위치 카운터 ≥ COMP+ 레지스터
low level(0): 논리/실제위치 카운터 < COMP+ 레지스터
RR1 레지스터의 D0(CMP+)비트를 읽어 비교해도 됩니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다.
논리/실제위치 카운터의 값이 COMP+ 레지스터값 이상이면 MMC_HIGH_LEVEL 를
논리/실제위치 카운터의 값이 COMP+ 레지스터값보다 작으면 MMC_LOW_LEVEL 를
리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_flag_compp(MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(level == MMC_OK)        // 논리/실제위치 카운터 ≥ COMP+ 레지스터
        { printf("논리/실제위치 카운터 ≥ COMP+\n"); }
        else                       // 논리/실제위치 카운터 < COMP+ 레지스터
        { printf("논리/실제위치 카운터 < COMP+\n"); }
```



```
        pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_flag_compm

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_CMPP, RR1_CMPM

9.22 flag_compm

MMC_INT16U **pmc4bpci_flag_compm**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

논리/실제위치 카운터와 COMP-레지스터의 대소관계를 나타냅니다.
high level(1): 논리/실제위치 카운터 <= COMP- 레지스터
low level(0): 논리/실제위치 카운터 > COMP- 레지스터
RR1 레지스터의 D1(CMP-)비트를 읽어 비교해도 됩니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 논리/실제위치 카운터의 값이 COMP- 레지스터값 이하이면 MMC_HIGH_LEVEL 를 논리/실제위치 카운터의 값이 COMP- 레지스터값보다 크면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_flag_compm(MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(level == MMC_OK)         // 논리/실제위치 카운터 <= COMP- 레지스터
        { printf("논리/실제위치 카운터 <= COMP-\n"); }
        else                        // 논리/실제위치 카운터 > COMP- 레지스터
```

```
                { printf("논리/실제위치  카운터  > COMP-\n"); }
 pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_flag_compp

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_CMPP, RR1_CMPM

10 신호정지

10.1 flag_in0

MMC_INT16U **pmc4bpci_flag_in0**(MMC_INT16U *id*);

(1) 설명

드라이브가 외부감속정지(nIN0)에 의해서 정지 했을 때 high level(1)를 리턴합니다. pmc4bpci_is_stop()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_flag_in0**()함수를 사용합니다. pmc4bpci_is_stop()함수를 사용하지 않고 **pmc4bpci_flag_in0**()함수를 사용하고자 한다면 축 선택을 먼저 하여야 합니다. RR1 레지스터의 D8(IN0) 비트에 의해서 검출할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 드라이브가 외부감속정지(nIN0) 신호를 발생하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_is_stop(MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(MMC_ERROR()==MMC_OK)
        {
                if(level==MMC_OK)
                {
                        if(pmc4bpci_flag_in0(MMC_CARD_NO)==MMC_OK)
```

```
                                {       printf("in0 정지!\n");          }
                                else if(pmc4bpci_flag_in1(MMC_CARD_NO)==MMC_OK)
                                {       printf("in1 정지!\n");          }
                                else if(pmc4bpci_flag_in2(MMC_CARD_NO)==MMC_OK)
                                {       printf("in2 정지!\n");          }
                                else if(pmc4bpci_flag_in3(MMC_CARD_NO)==MMC_OK)
                                {       printf("in3 정지!\n");          }
                                else
    if(pmc4bpci_flag_limitplus(MMC_CARD_NO)==MMC_OK)
                                {       printf("limitplus 정지!\n");    }
                                else
    if(pmc4bpci_flag_limitminus(MMC_CARD_NO)==MMC_OK)
                                {       printf("limitminus 정지!\n");   }
                                else
                                if(pmc4bpci_flag_servoalarm(MMC_CARD_NO)==MMC_OK)
                                {       printf("servoalarm 정지!\n");   }
                                else
                                if(pmc4bpci_flag_emergency(MMC_CARD_NO)==MMC_OK)
                                {       printf("emergency 정지!\n");    }
                        }
                }

                pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_is_stop, pmc4bpci_flag_in0, pmc4bpci_flag_in1, pmc4bpci_flag_in2, pmc4bpci_flag_in3, pmc4bpci_flag_limitplus, pmc4bpci_flag_limitminus, pmc4bpci_flag_servoalarm, pmc4bpci_flag_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_IN0, RR1_IN1, RR1_IN2, RR1_IN3, RR1_LMTP, RR1_LMTM, RR1_ALARM, RR1_EMG

10.2 flag_in1

MMC_INT16U **pmc4bpci_flag_in1**(MMC_INT16U *id*);

(1) 설명

드라이브가 외부감속정지(nIN1)에 의해서 정지 했을 때 high level(1)를 리턴합니다. pmc4bpci_is_stop()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_flag_in1**()함수를 사용합니다. pmc4bpci_is_stop()함수를 사용하지 않고 **pmc4bpci_flag_in1**()함수를 사용하고자 한다면 축 선택을 먼저 하여야 합니다. RR1 레지스터의 D9(IN1) 비트에 의해서 검출할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 드라이브가 외부감속정지(nIN1) 신호를 발생하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

pmc4bpci_flag_in0() 함수 참조.

(5) 참고함수

pmc4bpci_is_stop, pmc4bpci_flag_in0, pmc4bpci_flag_in1, pmc4bpci_flag_in2, pmc4bpci_flag_in3, pmc4bpci_flag_limitplus, pmc4bpci_flag_limitminus, pmc4bpci_flag_servoalarm, pmc4bpci_flag_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_IN0, RR1_IN1, RR1_IN2, RR1_IN3, RR1_LMTP, RR1_LMTM, RR1_ALARM, RR1_EMG

10.3 flag_in2

MMC_INT16U **pmc4bpci_flag_in2**(MMC_INT16U *id*);

(1) 설명

드라이브가 외부감속정지(nIN2)에 의해서 정지 했을 때 high level(1)를 리턴합니다. pmc4bpci_is_stop()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_flag_in2**()함수를 사용합니다. pmc4bpci_is_stop()함수를 사용하지 않고 **pmc4bpci_flag_in2**()함수를 사용하고자 한다면 축 선택을 먼저 하여야 합니다. RR1 레지스터의 D10(IN2) 비트에 의해서 검출할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 드라이브가 외부감속정지(nIN2) 신호를 발생하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

pmc4bpci_flag_in0() 함수 참조.

(5) 참고함수

pmc4bpci_is_stop, pmc4bpci_flag_in0, pmc4bpci_flag_in1, pmc4bpci_flag_in2, pmc4bpci_flag_in3, pmc4bpci_flag_limitplus, pmc4bpci_flag_limitminus, pmc4bpci_flag_servoalarm, pmc4bpci_flag_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_IN0, RR1_IN1, RR1_IN2, RR1_IN3, RR1_LMTP, RR1_LMTM, RR1_ALARM, RR1_EMG

10.4 flag_in3

MMC_INT16U **pmc4bpci_flag_in3**(MMC_INT16U *id*);

(1) 설명

드라이브가 외부감속정지(nIN3)에 의해서 정지 했을 때 high level(1)를 리턴합니다. pmc4bpci_is_stop()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_flag_in3**()함수를 사용합니다. pmc4bpci_is_stop()함수를 사용하지 않고 **pmc4bpci_flag_in3**()함수를 사용하고자 한다면 축 선택을 먼저 하여야 합니다. RR1 레지스터의 D11(IN3) 비트에 의해서 검출할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 드라이브가 외부감속정지(nIN3) 신호를 발생하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

pmc4bpci_flag_in0() 함수 참조.

(5) 참고함수

pmc4bpci_is_stop, pmc4bpci_flag_in0, pmc4bpci_flag_in1, pmc4bpci_flag_in2, pmc4bpci_flag_in3, pmc4bpci_flag_limitplus, pmc4bpci_flag_limitminus, pmc4bpci_flag_servoalarm, pmc4bpci_flag_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_IN0, RR1_IN1, RR1_IN2, RR1_IN3, RR1_LMTP, RR1_LMTM, RR1_ALARM, RR1_EMG

10.5 flag_limitplus

MMC_INT16U **pmc4bpci_flag_limitplus**(MMC_INT16U *id*);

(1) 설명

드라이브가 +방향 리미트신호(nLMTP)에 의해서 정지 했을 때 high level(1)를 리턴합니다. pmc4bpci_is_stop()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_flag_limitplus**()함수를 사용합니다. pmc4bpci_is_stop()함수를 사용하지 않고 **pmc4bpci_flag_limitplus**()함수를 사용하고자 한다면 축 선택을 먼저 하여야 합니다. RR1 레지스터의 D12(LMT+) 비트에 의해서 검출할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 드라이브가 +방향 리미트신호(nLMTP) 신호를 발생하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

pmc4bpci_flag_in0() 함수 참조.

(5) 참고함수

pmc4bpci_is_stop, pmc4bpci_flag_in0, pmc4bpci_flag_in1, pmc4bpci_flag_in2, pmc4bpci_flag_in3, pmc4bpci_flag_limitplus, pmc4bpci_flag_limitminus, pmc4bpci_flag_servoalarm, pmc4bpci_flag_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_IN0, RR1_IN1, RR1_IN2, RR1_IN3, RR1_LMTP, RR1_LMTM, RR1_ALARM, RR1_EMG

10.6 flag_limitminus

MMC_INT16U **pmc4bpci_flag_limitminus**(MMC_INT16U *id*);

(1) 설명

드라이브가 -방향 리미트신호(nLMTM)에 의해서 정지 했을 때 high level(1)를 리턴합니다. pmc4bpci_is_stop()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_flag_limitminus**()함수를 사용합니다. pmc4bpci_is_stop()함수를 사용하지 않고 **pmc4bpci_flag_limitminus**()함수를 사용하고자 한다면 축 선택을 먼저 하여야 합니다. RR1 레지스터의 D13(LMT-) 비트에 의해서 검출할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 드라이브가 -방향 리미트신호(nLMTM) 신호를 발생하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

pmc4bpci_flag_in0() 함수 참조.

(5) 참고함수

pmc4bpci_is_stop, pmc4bpci_flag_in0, pmc4bpci_flag_in1, pmc4bpci_flag_in2, pmc4bpci_flag_in3, pmc4bpci_flag_limitplus, pmc4bpci_flag_limitminus, pmc4bpci_flag_servoalarm, pmc4bpci_flag_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_IN0, RR1_IN1, RR1_IN2, RR1_IN3, RR1_LMTP, RR1_LMTM, RR1_ALARM, RR1_EMG

10.7 flag_servoalarm

MMC_INT16U **pmc4bpci_flag_servoalarm**(MMC_INT16U *id*);

(1) 설명

드라이브가 서보모터용 알람신호(nALARM)에 의해서 정지 했을 때 high level(1)를 리턴합니다. pmc4bpci_is_stop()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_flag_servoalarm**()함수를 사용합니다. pmc4bpci_is_stop()함수를 사용하지 않고 **pmc4bpci_flag_servoalarm**()함수를 사용하고자 한다면 축 선택을 먼저 하여야 합니다. RR1 레지스터의 D14(ALARM) 비트에 의해서 검출할 수 있습니다.

(2) 인자

id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 드라이브가 서보모터용 알람신호(nALARM) 신호를 발생하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

pmc4bpci_flag_in0() 함수 참조.

(5) 참고함수

pmc4bpci_is_stop, pmc4bpci_flag_in0, pmc4bpci_flag_in1, pmc4bpci_flag_in2, pmc4bpci_flag_in3, pmc4bpci_flag_limitplus, pmc4bpci_flag_limitminus, pmc4bpci_flag_servoalarm, pmc4bpci_flag_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_IN0, RR1_IN1, RR1_IN2, RR1_IN3, RR1_LMTP, RR1_LMTM, RR1_ALARM, RR1_EMG

10.8 flag_emergency

MMC_INT16U **pmc4bpci_flag_emergency**(MMC_INT16U *id*);

(1) 설명

드라이브가 긴급정지신호(nEMG)에 의해서 정지 했을 때 high level(1)를 리턴합니다. pmc4bpci_is_stop()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_flag_emergency**()함수를 사용합니다. pmc4bpci_is_stop()함수를 사용하지 않고 **pmc4bpci_flag_emergency**()함수를 사용하고자 한다면 축 선택을 먼저 하여야 합니다. RR1 레지스터의 D15(EMG) 비트에 의해서 검출할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 드라이브가 긴급정지신호(nEMG) 신호를 발생하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

pmc4bpci_flag_in0() 함수 참조.

(5) 참고함수

pmc4bpci_is_stop, pmc4bpci_flag_in0, pmc4bpci_flag_in1, pmc4bpci_flag_in2, pmc4bpci_flag_in3, pmc4bpci_flag_limitplus, pmc4bpci_flag_limitminus, pmc4bpci_flag_servoalarm, pmc4bpci_flag_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_IN0, RR1_IN1, RR1_IN2, RR1_IN3, RR1_LMTP, RR1_LMTM, RR1_ALARM, RR1_EMG

10.9 is_error

MMC_INT16U **pmc4bpci_is_error**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

에러를 검사하는 함수입니다. 소프트웨어 리미트(SLMT+/-), 리미트(HLMT+/-), 서보모터 알람(ALARM), 긴급정지신호(EMG)중 한 개만 설정이 되어도 high level(1)을 리턴합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 소프트웨어 리미트(SLMT+/-), 리미트(HLMT+/-), 서보모터 알람(ALARM), 긴급정지신호(EMG)중 한 개만 설정 되어도 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_is_error(MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(level == MMC_OK)
        {       printf("에러 발생!\n");                 }

        pmc4bpci_close_all();
}
```


(5) 참고함수

pmc4bpci_error_slimitplus, pmc4bpci_error_slimitminus,
pmc4bpci_error_hlimitplus, pmc4bpci_error_hlimitminus,
pmc4bpci_error_servoalarm, pmc4bpci_error_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR2_SLMTP, RR2_SLMTM, RR2_HLMTP,
RR2_HLMTM, RR2_ALARM, RR2_EMG

10.10 error_slimitplus

MMC_INT16U **pmc4bpci_error_slimitplus**(MMC_INT16U *id*);

(1) 설명

드라이브가 +방향 소프트 리미트 신호(nSLMTP)에 의해서 정지할 때 high level(1)를 리턴합니다. pmc4bpci_is_error()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_error_slimitplus**()함수를 사용합니다. pmc4bpci_is_error()함수를 사용하지 않고 p**mc4bpci_error_slimitplus**()함수를 사용하고자 한다면 먼저 축 선택을 하여야 합니다. RR2 레지스터의 D0(SLMT+) 비트에 의해서 검출할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 드라이브가 +방향 소프트 리미트 신호(nSLMTP)에 의해서 정지 신호를 발생하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_is_error(MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(MMC_ERROR()==MMC_OK)
        {
                // 에러요소가 발생하면
                if(level==MMC_OK)
                {
```

```
                    if(pmc4bpci_error_slimitplus(MMC_CARD_NO)==MMC_OK)
                    {       printf("+방향 Soft Limit 에러!\n");         }
                    else
        if(pmc4bpci_error_slimitminus(MMC_CARD_NO)==MMC_OK)
                    {       printf("-방향 Soft Limit 에러!\n");         }
                    else
                    if(pmc4bpci_error_hlimitplus(MMC_CARD_NO)==MMC_OK)
                    {       printf("+방향 Limit 신호 에러!\n");         }
                    else
        if(pmc4bpci_error_hlimitminus(MMC_CARD_NO)==MMC_OK)
                    {       printf("-방향 Limit 신호 에러!\n");         }
                    else
        if(pmc4bpci_error_servoalarm(MMC_CARD_NO)==MMC_OK)
                    {       printf("in0 정지!\n");              }
                    else
        if(pmc4bpci_error_emergency(MMC_CARD_NO)==MMC_OK)
                    {       printf("긴급신호 에러!\n");         }
                }
            }

            pmc4bpci_close_all();
    }
```

(5) 참고함수

pmc4bpci_error_slimitplus, pmc4bpci_error_slimitminus,
pmc4bpci_error_hlimitplus, pmc4bpci_error_hlimitminus,
pmc4bpci_error_servoalarm, pmc4bpci_error_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL,
RR2_SLMTP, RR2_SLMTM, RR2_HLMTP, RR2_HLMTM, RR2_ALARM, RR2_EMG

10.11 error_slimitminus

MMC_INT16U **pmc4bpci_error_slimitminus**(MMC_INT16U *id*);

(1) 설명

드라이브가 -방향 소프트 리미트 신호(nSLMTM)에 의해서 정지할 때 high level(1)를 리턴합니다. pmc4bpci_is_error()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_error_slimitminus**()함수를 사용합니다. pmc4bpci_is_error()함수를 사용하지 않고 **pmc4bpci_error_slimitminus**()함수를 사용하고자 한다면 축 선택을 먼저 하여야 합니다. RR2 레지스터의 D1(SLMT-) 비트에 의해서 검출할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 드라이브가 -방향 소프트 리미트 신호(nSLMTM)에 의해서 정지 신호를 발생하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

pmc4bpci_error_slimitplus() 함수 참조.

(5) 참고함수

pmc4bpci_error_slimitplus, pmc4bpci_error_slimitminus,
pmc4bpci_error_hlimitplus, pmc4bpci_error_hlimitminus,
pmc4bpci_error_servoalarm, pmc4bpci_error_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR2_SLMTP, RR2_SLMTM, RR2_HLMTP, RR2_HLMTM, RR2_ALARM, RR2_EMG

10.12 error_hlimitplus

MMC_INT16U **pmc4bpci_error_hlimitplus**(MMC_INT16U *id*);

(1) 설명

드라이브가 +방향 리미트 신호(nLMTP)에 의해서 정지할 때 high level(1)를 리턴합니다. pmc4bpci_is_error()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_error_hlimitplus**()함수를 사용합니다. pmc4bpci_is_error()함수를 사용하지 않고 **pmc4bpci_error_hlimitplus**()함수를 사용하고자 한다면 축 선택을 먼저 하여야 합니다. RR2 레지스터의 D2(HLMT+) 비트에 의해서 검출할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 드라이브가 +방향 리미트 신호(nLMTP)에 의해서 정지 신호를 발생하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

pmc4bpci_error_slimitplus() 함수 참조.

(5) 참고함수

pmc4bpci_error_slimitplus, pmc4bpci_error_slimitminus,
pmc4bpci_error_hlimitplus, pmc4bpci_error_hlimitminus,
pmc4bpci_error_servoalarm, pmc4bpci_error_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR2_SLMTP, RR2_SLMTM, RR2_HLMTP, RR2_HLMTM, RR2_ALARM, RR2_EMG

10.13 error_hlimitminus

MMC_INT16U **pmc4bpci_error_hlimitminus**(MMC_INT16U *id*);

(1) 설명

드라이브가 -방향 리미트 신호(nLMTM)에 의해서 정지할 때 high level(1)를 리턴합니다. pmc4bpci_is_error()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_error_hlimitminus**()함수를 사용합니다. pmc4bpci_is_error()함수를 사용하지 않고 **pmc4bpci_error_hlimitminus**()함수를 사용하고자 한다면 축 선택을 먼저 하여야 합니다. RR2 레지스터의 D3(HLMT-) 비트에 의해서 검출할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 드라이브가 -방향 리미트 신호(nLMTM)에 의해서 정지 신호를 발생하면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

pmc4bpci_error_slimitplus() 함수 참조.

(5) 참고함수

pmc4bpci_error_slimitplus, pmc4bpci_error_slimitminus,
pmc4bpci_error_hlimitplus, pmc4bpci_error_hlimitminus,
pmc4bpci_error_servoalarm, pmc4bpci_error_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR2_SLMTP, RR2_SLMTM, RR2_HLMTP, RR2_HLMTM, RR2_ALARM, RR2_EMG

10.14 error_emergency

MMC_INT16U **pmc4bpci_error_emergency**(MMC_INT16U *id*);

(1) 설명

긴급정지신호(nEMG)가 low level(0)로 되었을 때 high level(1)를 리턴합니다. pmc4bpci_is_error()함수를 사용해 에러의 유무를 확인 후 **pmc4bpci_error_emergency**()함수를 사용합니다. . pmc4bpci_is_error()함수를 사용하지 않고 **pmc4bpci_error_emergency**()함수를 사용하고자 한다면 먼저 축 선택을 하여야 합니다. RR2 레지스터의 D5(EMG) 비트에 의해서 검출할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴합니다. 긴급정지신호(nEMG)가 low level(0)로 되면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

pmc4bpci_error_slimitplus() 함수 참조.

(5) 참고함수

pmc4bpci_error_slimitplus, pmc4bpci_error_slimitminus,
pmc4bpci_error_hlimitplus, pmc4bpci_error_hlimitminus,
pmc4bpci_error_servoalarm, pmc4bpci_error_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR2_SLMTP, RR2_SLMTM, RR2_HLMTP, RR2_HLMTM, RR2_ALARM, RR2_EMG

10.15 is_stop

MMC_INT16U **pmc4bpci_is_stop**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

드라이브가 외부감속정지(nIN0~3), +방향 리미트신호(nLMTP), -방향 리미트신호(nLMTM), 서보모터용 알람신호(nALARM), 긴급정지신호(nEMG)에 의해서 정지 했을 때 high level(1)를 리턴합니다. 드라이브 종료 상태비트 중 에러요인이 되는 nLMTP, nLMTM, nALARM, nEMG 가 하나라도 high level(1)이 되면 RR0 의 nERR 비트는 high level(1)이 됩니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다.
드라이브가 외부감속정지(nIN0~3), +방향 리미트신호(nLMTP), -방향 리미트신호(nLMTM), 서보모터용 알람신호(nALARM), 긴급정지신호(nEMG)등의 신호중 한 개 이상이 발생하면 MMC_HIGH_LEVEL 를 어떤 정지신호도 발생하지 않았다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

level = pmc4bpci_is_stop(MMC_CARD_NO, PMC4BPCI_AXIS_X);

if(level == MMC_OK)          // 정지를 발생할때
```

```
{ printf("외부요인으로  인한  정지!\n"); }
pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_is_stop, pmc4bpci_flag_in0, pmc4bpci_flag_in1, pmc4bpci_flag_in2, pmc4bpci_flag_in3, pmc4bpci_flag_limitplus, pmc4bpci_flag_limitminus, pmc4bpci_flag_servoalarm, pmc4bpci_flag_emergency

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR1_IN0, RR1_IN1, RR1_IN2, RR1_IN3, RR1_LMTP, RR1_LMTM, RR1_ALARM, RR1_EMG

11 인터럽트

11.1 is_interrupt

MMC_INT16U **pmc4bpci_is_interrupt**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

인터럽트가 발생되면 high level(1)이 됩니다.
인터럽트가 발생되는 요인은 펄스가 올라갈 때(드라이브 펄스가 정논리로 설정 되었다면), 논리/실위치가 COMP- 레지스터값 이상 일 때, 논리/실위치가 COMP- 레지스터 보다 작을 때, 논리/실위치가 COMP+ 레지스터보다 작을 때, 논리/실위치가 COMP+ 레지스터값 이상일 때, 가감속 드라이브중 정속지역으로 펄스출력을 종료할 때, 가감속 드라이브일 때 정속지역으로 펄스출력을 개시할 때입니다.
어느 인터럽트 발생요인에 의해 인터럽트가 발생하면, RR3 레지스터의 인터럽트 상태 비트들이 high level(1)이 되고, 인터럽트 출력신호(INTN)가 low level(0)이 됩니다.
pmc4bpci_is_interrupt()함수를 사용하게 되면 RR3 레지스터의 내용이 0 으로 채워져 인터럽트 출력신호는 non active level 로 돌아갑니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다.
인터럽트가 발생되면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
```

```
        }

        level = pmc4bpci_is_interrupt(MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(level == MMC_HIGH_LEVEL)
        {       printf("인터럽트 발생!\n");       }

        pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_interrupt_pulse, pmc4bpci_interrupt_pulseGEcompm, pmc4bpci_interrupt_pulseLcompm, pmc4bpci_interrupt_pulseLcompp, pmc4bpci_interrupt_pulseGEcompp, pmc4bpci_interrupt_cend, pmc4bpci_interrupt_cstart, pmc4bpci_interrupt_enddrive, pmc4bpci_get_interrupt

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR3_PULSE, RR3_PGECM, RR3_PLCM, RR3_PLCP, RR3_PGECP, RR3_CEND, RR3_CSTA, RR3_DEND

11.2 get_interrupt

MMC_INT16U **pmc4bpci_get_interrupt**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

인터럽트 상태 레지스터의 값을 돌려줍니다. 인터럽트가 발생되는 요인은 펄스가 올라갈 때(드라이브 펄스가 정논리로 설정 되었다면), 논리/실위치가 COMP- 레지스터값 이상 일 때, 논리/실위치가 COMP- 레지스터 보다 작을 때, 논리/실위치가 COMP+ 레지스터보다 작을 때, 논리/실위치가 COMP+ 레지스터값 이상일 때, 가감속 드라이브중 정속지역으로 펄스출력을 종료할 때, 가감속 드라이브일 때 정속지역으로 펄스출력을 개시할 때입니다.

어느 인터럽트 발생요인에 의해 인터럽트가 발생하면, RR3 레지스터의 인터럽트 상태 비트들이 high level(1)이 되고, 인터럽트 출력신호(INTN)가 low level(0)이 됩니다.

pmc4bpci_is_interrupt()함수를 사용하게 되면 RR3 레지스터의 내용이 0 으로 채워져 인터럽트 출력신호는 non active level 로 돌아갑니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다.

인터럽트의 상태 값을 리턴합니다. 전역 에러 검사 변수인 gError 는 성공 일 경우에는 MMC_OK 값이 기록되고, 그렇지 않으면 해당 에러가 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    intval;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }
```

```
        intval = pmc4bpci_get_interrupt(MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(intval & RR3_PULSE)
        {       printf("펄스에 의한 인터럽트 처리\n");      }
        else if(intval & RR3_PGECM)
        {       printf("drive pulse 가 rising edge(정논리에서 pulse 가 올라가는
순간)\n"); }
        else if(intval & RR3_PLCM)
        {       printf("논리/실위치≥COMP- register\n");     }
        else if(intval & RR3_PLCP)
        {       printf("논리/실위치<COMP- register\n");                 }
        else if(intval & RR3_PGECP)
        {       printf("논리/실위치<COMP+ register\n");                 }
        else if(intval & RR3_CEND)
        {       printf("논리/실위치≥COMP+ register\n");     }
        else if(intval & RR3_CSTA)
        {       printf("가감속 dirve 일때 정속지역으로 pulse 출력을 종료\n"); }
        else if(intval & RR3_DEND)
        {       printf("가감속 drive 일때 정속지역으로 pulse 출력을 개시\n"); }

        pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_interrupt_pulse, pmc4bpci_interrupt_pulseGEcompm, pmc4bpci_interrupt_pulseLcompm, pmc4bpci_interrupt_pulseLcompp, pmc4bpci_interrupt_pulseGEcompp, pmc4bpci_interrupt_cend, pmc4bpci_interrupt_cstart, pmc4bpci_interrupt_enddrive, pmc4bpci_get_interrupt

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR3_PULSE, RR3_PGECM, RR3_PLCM, RR3_PLCP, RR3_PGECP, RR3_CEND, RR3_CSTA, RR3_DEND

11.3 interrupt_pulse

MMC_INT16U **pmc4bpci_interrupt_pulse**(MMC_INT16U id, MMC_INT16U *axis*);

(1) 설명

인터럽트가 발생(드라이브 펄스가 정논리로 설정 되었다면 펄스가 올라갈(rising edge) 때)할 때 high level(1)을 리턴합니다. 인터럽트 상태에 관련된 함수를 사용하면 RR3 레지스터의 내용이 지워지므로 인터럽트 상태 관련 함수를 두 번 이상 연속으로 사용 할 수 없습니다. RR3 레지스터 D0(PULSE)비트로 판단 할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 인터럽트가 발생되면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예 1

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_interrupt_pulse(MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(level == MMC_HIGH_LEVEL)
        {       printf("펄스에 의한 인터럽트\n");    }
        else
        {       printf("인터럽트 X\n");       }

        pmc4bpci_close_all();
```

```
}
```

(5) 사용 예 2

pmc4bpci_get_interrupt() 함수 참조.

(6) 참고함수

pmc4bpci_interrupt_pulse, pmc4bpci_interrupt_pulseGEcompm, pmc4bpci_interrupt_pulseLcompm, pmc4bpci_interrupt_pulseLcompp, pmc4bpci_interrupt_pulseGEcompp, pmc4bpci_interrupt_cend, pmc4bpci_interrupt_cstart, pmc4bpci_interrupt_enddrive, pmc4bpci_get_interrupt

(7) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR3_PULSE, RR3_PGECM, RR3_PLCM, RR3_PLCP, RR3_PGECP, RR3_CEND, RR3_CSTA, RR3_DEND

11.4 interrupt_pulseGEcompm

MMC_INT16U **pmc4bpci_interrupt_pulseGEcompm**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

인터럽트 발생(논리/실위치 ≥ COMP- register)할 때 high level(1)을 리턴합니다.
인터럽트 상태에 관련된 함수를 사용하면 RR3 레지스터의 내용이 지워지므로 인터럽트 상태 관련 함수를 두 번 이상 연속으로 사용 할 수 없습니다. RR3 레지스터 D1(P≥C-) 비트로 판단 할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위:0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다.
인터럽트가 발생되면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예 1

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_interrupt_ pulseGEcompm
 (MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(level == MMC_HIGH_LEVEL)
        {       printf("논리/실위치 ≥ COMP- register 인터럽트\n");   }
        else
        {       printf("인터럽트 X\n");       }
```

```
        pmc4bpci_close_all();
}
```

(5) 사용 예 2

pmc4bpci_get_interrupt() 함수 참조.

(6) 참고함수

pmc4bpci_interrupt_pulse, pmc4bpci_interrupt_pulseGEcompm, pmc4bpci_interrupt_pulseLcompm, pmc4bpci_interrupt_pulseLcompp, pmc4bpci_interrupt_pulseGEcompp, pmc4bpci_interrupt_cend, pmc4bpci_interrupt_cstart, pmc4bpci_interrupt_enddrive, pmc4bpci_get_interrupt

(7) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR3_PULSE, RR3_PGECM, RR3_PLCM, RR3_PLCP, RR3_PGECP, RR3_CEND, RR3_CSTA, RR3_DEND

11.5 interrupt_pulseLcompm

MMC_INT16U **pmc4bpci_interrupt_pulseLcompm**(MMC_INT16U id, MMC_INT16U *axis*);

(1) 설명

인터럽트 발생(논리/실위치 < COMP- register)할 때 high level(1)을 리턴합니다. 인터럽트 상태에 관련된 함수를 사용하면 RR3 레지스터의 내용이 지워지므로 인터럽트 상태 관련 함수를 두 번 이상 연속으로 사용 할 수 없습니다. RR3 레지스터 D2(P<C-)비트로 판단 할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 인터럽트가 발생되면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예 1

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_interrupt_ pulseLcompm
 (MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(level == MMC_HIGH_LEVEL)
        {       printf("논리/실위치  < COMP- register  인터럽트\n");    }
        else
        {       printf("인터럽트  X\n");       }
```

```
        pmc4bpci_close_all();
}
```

(5) 사용 예 2

pmc4bpci_get_interrupt() 함수 참조.

(6) 참고함수

pmc4bpci_interrupt_pulse, pmc4bpci_interrupt_pulseGEcompm, pmc4bpci_interrupt_pulseLcompm, pmc4bpci_interrupt_pulseLcompp, pmc4bpci_interrupt_pulseGEcompp, pmc4bpci_interrupt_cend, pmc4bpci_interrupt_cstart, pmc4bpci_interrupt_enddrive, pmc4bpci_get_interrupt

(7) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR3_PULSE, RR3_PGECM, RR3_PLCM, RR3_PLCP, RR3_PGECP, RR3_CEND, RR3_CSTA, RR3_DEND

11.6 interrupt_pulseLcompp

MMC_INT16U **pmc4bpci_interrupt_pulseLcompp**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

인터럽트 발생(논리/실위치 < COMP+ register)할 때 high level(1)을 리턴합니다. 인터럽트 상태에 관련된 함수를 사용하면 RR3 레지스터의 내용이 지워지므로 인터럽트 상태 관련 함수를 두 번 이상 연속으로 사용 할 수 없습니다. RR3 레지스터 D3(P<C+)비트로 판단 할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 인터럽트가 발생되면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예 1

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_interrupt_ pulseLcompp
 (MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(level == MMC_HIGH_LEVEL)
        {       printf("논리/실위치  < COMP+ register  인터럽트\n");   }
        else
        {       printf("인터럽트  X\n");       }
```

```
        pmc4bpci_close_all();
}
```

(5) 사용 예 2

pmc4bpci_get_interrupt() 함수 참조.

(6) 참고함수

pmc4bpci_interrupt_pulse, pmc4bpci_interrupt_pulseGEcompm, pmc4bpci_interrupt_pulseLcompm, pmc4bpci_interrupt_pulseLcompp, pmc4bpci_interrupt_pulseGEcompp, pmc4bpci_interrupt_cend, pmc4bpci_interrupt_cstart, pmc4bpci_interrupt_enddrive, pmc4bpci_get_interrupt

(7) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR3_PULSE, RR3_PGECM, RR3_PLCM, RR3_PLCP, RR3_PGECP, RR3_CEND, RR3_CSTA, RR3_DEND

11.7 interrupt_pulseGEcompp

MMC_INT16U **pmc4bpci_interrupt_pulseGEcompp**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

인터럽트 발생(논리/실위치 ≥ COMP+ register)할 때 high level(1)을 리턴합니다.
인터럽트 상태에 관련된 함수를 사용하면 RR3 레지스터의 내용이 지워지므로 인터럽트 상태 관련 함수를 두 번 이상 연속으로 사용 할 수 없습니다. RR3 레지스터 D4(P≥C+)비트로 판단 할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다.
인터럽트가 발생되면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예 1

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_interrupt_ pulseGEcompp
 (MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(level == MMC_HIGH_LEVEL)
        {       printf("논리/실위치 ≥ COMP+ register 인터럽트\n");  }
        else
        {       printf("인터럽트 X\n");       }
```

```
        pmc4bpci_close_all();
}
```

(5) 사용 예 2

pmc4bpci_get_interrupt() 함수 참조.

(6) 참고함수

pmc4bpci_interrupt_pulse, pmc4bpci_interrupt_pulseGEcompm, pmc4bpci_interrupt_pulseLcompm, pmc4bpci_interrupt_pulseLcompp, pmc4bpci_interrupt_pulseGEcompp, pmc4bpci_interrupt_cend, pmc4bpci_interrupt_cstart, pmc4bpci_interrupt_enddrive, pmc4bpci_get_interrupt

(7) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR3_PULSE, RR3_PGECM, RR3_PLCM, RR3_PLCP, RR3_PGECP, RR3_CEND, RR3_CSTA, RR3_DEND

11.8 interrupt_cend

MMC_INT16U **pmc4bpci_interrupt_cend**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

인터럽트 발생(가감속 드라이브일 때 정속지역으로 펄스출력을 종료할 때)할 때 high level(1)을 리턴합니다. 인터럽트 상태에 관련된 함수를 사용하면 RR3 레지스터의 내용이 지워지므로 인터럽트 상태 관련 함수를 두 번 이상 연속으로 사용 할 수 없습니다. RR3 레지스터 D5(C-END)비트로 판단 할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 인터럽트가 발생되면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예 1

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_interrupt_cend
 (MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(level == MMC_HIGH_LEVEL)
        {   printf("가감속 드라이브일 때 정속지역으로 펄스출력을 종료할 때 \n");       }
        else
        {       printf("인터럽트 X\n");       }
```

```
        pmc4bpci_close_all();
}
```

(5) 사용 예 2

pmc4bpci_get_interrupt() 함수 참조.

(6) 참고함수

pmc4bpci_interrupt_pulse, pmc4bpci_interrupt_pulseGEcompm, pmc4bpci_interrupt_pulseLcompm, pmc4bpci_interrupt_pulseLcompp, pmc4bpci_interrupt_pulseGEcompp, pmc4bpci_interrupt_cend, pmc4bpci_interrupt_cstart, pmc4bpci_interrupt_enddrive, pmc4bpci_get_interrupt

(7) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR3_PULSE, RR3_PGECM, RR3_PLCM, RR3_PLCP, RR3_PGECP, RR3_CEND, RR3_CSTA, RR3_DEND

11.9 interrupt_cstart

MMC_INT16U **pmc4bpci_interrupt_cstart**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

인터럽트 발생(가감속 드라이브일 때 정속지역으로 펄스출력을 개시할 때)할 때 high level(1)을 리턴합니다. 인터럽트 상태에 관련된 함수를 사용하면 RR3 레지스터의 내용이 지워지므로 인터럽트 상태 관련 함수를 두 번 이상 연속으로 사용 할 수 없습니다. RR3 레지스터 D6(C-STA)비트로 판단 할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 인터럽트가 발생되면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예 1

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_interrupt_cstart(MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(level == MMC_HIGH_LEVEL)
        {   printf("가감속 드라이브일 때 정속지역으로 펄스출력을 개시할 때 \n");       }
        else
        {       printf("인터럽트 X\n");       }

        pmc4bpci_close_all();
```

```
}
```

(5) 사용 예 2

pmc4bpci_get_interrupt() 함수 참조.

(6) 참고함수

pmc4bpci_interrupt_pulse, pmc4bpci_interrupt_pulseGEcompm, pmc4bpci_interrupt_pulseLcompm, pmc4bpci_interrupt_pulseLcompp, pmc4bpci_interrupt_pulseGEcompp, pmc4bpci_interrupt_cend, pmc4bpci_interrupt_cstart, pmc4bpci_interrupt_enddrive, pmc4bpci_get_interrupt

(7) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR3_PULSE, RR3_PGECM, RR3_PLCM, RR3_PLCP, RR3_PGECP, RR3_CEND, RR3_CSTA, RR3_DEND

11.10 interrupt_enddrive

MMC_INT16U **pmc4bpci_interrupt_enddrive**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

인터럽트 발생(드라이브가 종료 할 때)할 때 high level(1)을 리턴합니다. 인터럽트 상태에 관련된 함수를 사용하면 RR3 레지스터의 내용이 지워지므로 인터럽트 상태 관련 함수를 두 번 이상 연속으로 사용 할 수 없습니다. RR3 레지스터 D7(D-END)비트로 판단 할 수 있습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 인터럽트가 발생되면 MMC_HIGH_LEVEL 를 그렇지 않다면 MMC_LOW_LEVEL 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예 1

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_interrupt_enddrive(MMC_CARD_NO, PMC4BPCI_AXIS_X);

        if(level == MMC_HIGH_LEVEL)
        {       printf("드라이브가 종료 할 때 인터럽트\n"); }
        else
        {       printf("인터럽트 X\n");       }

        pmc4bpci_close_all();
}
```

(5) 사용 예 2

pmc4bpci_get_interrupt() 함수 참조.

(6) 참고함수

pmc4bpci_interrupt_pulse, pmc4bpci_interrupt_pulseGEcompm, pmc4bpci_interrupt_pulseLcompm, pmc4bpci_interrupt_pulseLcompp, pmc4bpci_interrupt_pulseGEcompp, pmc4bpci_interrupt_cend, pmc4bpci_interrupt_cstart, pmc4bpci_interrupt_enddrive, pmc4bpci_get_interrupt

(7) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, RR3_PULSE, RR3_PGECM, RR3_PLCM, RR3_PLCP, RR3_PGECP, RR3_CEND, RR3_CSTA, RR3_DEND

12 입력신호 상태 읽기

12.1 input_status

MMC_INT16U **pmc4bpci_input_status**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

해당 축의 입력신호 상태를 8 비트 데이터로 돌려줍니다.

RR4 와 RR5 레지스터를 8 비트씩 4 개의 바이트로 나누어서 판단합니다. (RR4 의 Low byte=X 축, RR4 의 High byte=Y 축, RR5 의 Low byte=Z 축, RR5 의 High byte=U 축을 나타냅니다..)

D7	D6	D5	D4	D3	D2	D1	D0
n-ALM	n-INP	n-EX-	n-EX+	n-IN3	n-IN2	n-IN1	n-IN0

(2) 인자

- id: id 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

해당 축의 상태를 8 비트 데이터로 돌려주며, 전역 에러 검사 변수인 gError 에는 MMC_OK 가 기록됩니다. PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD(13)를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS(8)를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U    stat;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        stat = pmc4bpci_input_status(MMC_CARD_NO, PMC4BPCI_AXIS_X);
```

```
        // Low Active 평상시에 High
        if(!(stat & n_IN0))
        {       printf("n_IN0 ON\n");       }
        else if(!(stat & n_IN1))
        {       printf("n_IN1 ON\n");       }
        else if(!(stat & n_IN2))
        {       printf("n_IN2 ON\n");       }
        else if(!(stat & n_IN3))
        {       printf("n_IN3 ON\n");       }
        else if(!(stat & n_EXPP))
        {       printf("n_EXPP ON\n");      }
        else if(!(stat & n_EXPM))
        {       printf("n_EXPM ON\n");      }
        else if(!(stat & n_INPOS))
        {       printf("n_INPOS ON\n");     }
        else if(!(stat & n_ALARM))
        {       printf("n_ALARM ON\n");     }

        pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_inputstatus_in0, pmc4bpci_inputstatus_in1, pmc4bpci_inputstatus_in2, pmc4bpci_inputstatus_in3, pmc4bpci_inputstatus_exp, pmc4bpci_inputstatus_exm, pmc4bpci_inputstatus_inpos, pmc4bpci_inputstatus_alarm

(6) 참고매크로

n_IN0, n_IN1, n_IN2, n_IN3, n_EXPP, n_EXPM, n_INPOS, n_ALARM

12.2 inputstatus_in0~in3, inputstatus_exp, inputstatus_exm, inputstatus_inpos, inputstatus_alarm

MC_INT16U **pmc4bpci_inputstatus_in0**(MMC_INT16U *id*, MMC_INT16U *axis*);

MC_INT16U **pmc4bpci_inputstatus_in1**(MMC_INT16U *id*, MMC_INT16U *axis*);

MC_INT16U **pmc4bpci_inputstatus_in2**(MMC_INT16U *id*, MMC_INT16U *axis*);

MC_INT16U **pmc4bpci_inputstatus_in3**(MMC_INT16U *id*, MMC_INT16U *axis*);

MC_INT16U **pmc4bpci_inputstatus_exp**(MMC_INT16U *id*, MMC_INT16U *axis*);

MC_INT16U **pmc4bpci_inputstatus_exm**(MMC_INT16U *id*, MMC_INT16U *axis*);

MC_INT16U **pmc4bpci_inputstatus_inpos**(MMC_INT16U *id*, MMC_INT16U *axis*);

MC_INT16U **pmc4bpci_inputstatus_alarm**(MMC_INT16U *id*, MMC_INT16U *axis*);

(1) 설명

해당 축의 입력신호의 상태를 나타냅니다.. RR4 와 RR5 레지스터 8 비트씩 4 개의 바이트로 나누어집니다.

RR4	D7	D6	D5	D4	D3	D2	D1	D0
Low	X-ALM	X-INP	X-EX-	X-EX+	X-IN3	X-IN2	X-IN1	X-IN0
RR4	D15	D14	D13	D12	D11	D10	D9	D8
High	Y-ALM	Y-INP	Y-EX-	Y-EX+	Y-IN3	Y-IN2	Y-IN1	Y-IN0
RR5	D7	D6	D5	D4	D3	D2	D1	D0
Low	Z-ALM	Z-INP	Z-EX-	Z-EX+	Z-IN3	Z-IN2	Z-IN1	Z-IN0
RR5	D15	D14	D13	D12	D11	D10	D9	D8
High	U-ALM	U-INP	U-EX-	U-EX+	U-IN3	U-IN2	U-IN1	U-IN0

(2) 인자

- id: id 번호를 입력합니다. (범위: 0~15)
- axis: 1 축만을 선택합니다.

(3) 리턴값

해당 축의 상태가 high level(1)이면 MMC_HIGH_LEVEL 을 그렇지 않으면 MMC_LOW_LEVEL 을 리턴합니다. PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"
```

```
#define MMC_CARD_NO     15

void main()
{
        MMC_INT16U level;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

        level = pmc4bpci_inputstatus_in0(MMC_CARD_NO, PMC4BPCI_AXIS_X);

        // IC 쪽으로 들어가는 신호는 항상 ON(Low Active), IN0 신호를 넣어주면 Low
 if(!(level == MMC_HIGH_LEVEL))
        {       printf("IN0 신호 ON\n");    }
        else if(level == MMC_HIGH_LEVEL)
        {       printf("IN0 신호 off\n");     }

        pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_inputstatus_in0, pmc4bpci_inputstatus_in1, pmc4bpci_inputstatus_in2, pmc4bpci_inputstatus_in3, pmc4bpci_inputstatus_exp, pmc4bpci_inputstatus_exm, pmc4bpci_inputstatus_inpos, pmc4bpci_inputstatus_alarm

(6) 참고매크로

MMC_HIGH_LEVEL, MMC_LOW_LEVEL, n_IN0, n_IN1, n_IN2, n_IN3, n_EXPP, n_EXPM, n_INPOS, n_ALARM

13 동작

13.1 pls_move

MMC_INT16U **pmc4bpci_pls_move**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32 pls,MMC_INT16 *vel*,MMC_INT16 *acc*);

(1) 설명

지정펄스만큼 직선 가감속 정량 드라이브를 합니다. 드라이브완료까지 기다리지 않는다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- pls: 펄스(범위: –268,435,455~268,435,455)
- vel: 속도(범위: 1~8,000)
- acc: 가(감)속도(범위: 1~8,000)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn = pmc4bpci_pls_move(MMC_CARD_NO,PMC4BPCI_AXIS_X,10000,
4000, 200);
// ……
```

```
if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

pmc4bpci_close_all();

}
```

(5) 참고함수

pmc4bpci_pls_move, pmc4bpci_pls_move_wait

13.2 pls_move_wait

MMC_INT16U **pmc4bpci_pls_move_wait**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32 *pls*,MMC_INT16 *vel*,MMC_INT16 *acc*);

(1) 설명

지정펄스만큼 직선 가감속 정량 드라이브를 합니다. 드라이브완료까지 기다린다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- pls: 펄스(범위: –268,435,455~268,435,455)
- vel: 속도(범위: 1~8,000)
- acc: 가(감)속도(범위: 1~8,000)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정된 내부 타임아웃시간에 의해 지정시간이 끝나 함수를 종료할 경우에는 MMC_TIMEOUT_ERR 을 리턴합니다. 실행결과 성공하면 MMC_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn = pmc4bpci_pls_move_wait
 (MMC_CARD_NO,PMC4BPCI_AXIS_X,10000,4000, 200);
// ……
```

```
if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_pls_move, pmc4bpci_pls_move_wait

13.3 pos_move

MMC_INT16U **pmc4bpci_pos_move**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32 *pos*,MMC_INT16 *vel*,MMC_INT16 *acc*);

(1) 설명

지정위치로 직선 가감속 정량 드라이브합니다. 드라이브완료까지 기다리지 않는다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- pos: 설정된 논리/실 위치값(범위: -2,147,483,648~2,147,483,647)을 기준으로 옵셋범위(–268,435,455~268,435,455)내에서 설정 가능.
- vel: 속도(범위 1~8,000)
- acc: 가(감)속도(범위 1~8,000)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn = pmc4bpci_ pos_move
 (MMC_CARD_NO, PMC4BPCI_AXIS_X,10000, 4000, 200);
// ……
```

```
if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_pos_move, pmc4bpci_pos_move_wait

13.4 pos_move_wait

MMC_INT16U **pmc4bpci_pos_move_wait**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32 *pos*, MMC_INT16 *vel*,MMC_INT16 *acc*);

(1) 설명

지정위치로 직선 가감속 정량 드라이브를 합니다. 드라이브완료까지 기다린다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- pos: 설정된 논리/실 위치값(범위: -2,147,483,648~2,147,483,647)을 기준으로 옵셋범위(–268,435,455~268,435,455)내에서 설정 가능.
- vel: 속도(범위: 1~8,000)
- acc: 가(감)속도(범위: 1~8,000)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정된 내부 타임아웃시간에 의해 지정시간이 끝나 함수를 종료할 경우에는 MMC_TIMEOUT_ERR 을 리턴합니다. 실행결과 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn = pmc4bpci_pos_move_wait
 (MMC_CARD_NO,PMC4BPCI_AXIS_X,10000,4000, 200);
// ……
```

```
if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_pos_move, pmc4bpci_pos_move_wait

13.5 cmove

MMC_INT16U **pmc4bpci_cmove**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT16 *vel*);

(1) 설명

지정축의 연속 드라이브를 속도(*vel*)로 합니다. pmc4bpci_stop() 또는 pmc4bpci_dstop()로 정지시킵니다. 외부 정지신호 IN0~3 에 의해 정지합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- vel: 속도(범위: 1~8,000)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn = pmc4bpci_cmove(MMC_CARD_NO,PMC4BPCI_AXIS_X, 2000);
// ……

if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }

pmc4bpci_close_all();
```

```
}
```

(5) 참고함수

pmc4bpci_stop, pmc4bpci_dstop

13.6 pls_smove

MMC_INT16U **pmc4bpci_pls_smove**(MMC_INT16U *id*, MMC_INT16U *axis*, MMC_INT32 *pls*, MMC_INT16 *vel*, MMC_INT16 *acc*, MMC_INT16 *acac,* MMC_INT16 *dec,* MMC_INT16 *dcac*);

(1) 설명

선택축에 지정펄스로 S 자 가감속도커브 드라이브를 합니다.

드라이브 완료까지 기다리지 않습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- pls: 펄스(범위: –268,435,455~268,435,455)
- vel: 속도(범위: 1~8,000)
- acc: 가(감)속도(범위: 1~8,000)
- acac: 가가속도(범위: 1~65,535)
- dec: 감속도(범위: 1~8,000)
- dcac: 감속도 증가율(범위: 1~65,535)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다.

설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }
// ……
rtn = pmc4bpci_pls_smove(MMC_CARD_NO, PMC4BPCI_AXIS_X, 10000,
4000, 1000, 300, 1000, 300);
```

```
// ……
if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }
pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_pls_smove_wait

13.7 pls_smove_wait

MMC_INT16U **pmc4bpci_pls_smove_wait**(MMC_INT16U *id*, MMC_INT16U *axis*, MMC_INT32 *pls*, MMC_INT16 *vel*, MMC_INT16 *acc*, MMC_INT16 *acac,* MMC_INT16 *dec,* MMC_INT16 *dcac*);

(1) 설명

선택축에 지정펄스로 S 자 가감속도커브 드라이브를 합니다. 드라이브완료까지 기다린다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- pls: 펄스(범위: –268,435,455~268,435,455)
- vel: 속도(범위: 1~8,000)
- acc: 가(감)속도(범위: 1~8,000)
- acac: 가가속도(범위: 1~65,535)
- dec: 감속도(범위: 1~8,000)
- dcac: 감속도 증가율(범위: 1~65,535)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }
// ……
rtn = pmc4bpci_pls_smove_wait(MMC_CARD_NO, PMC4BPCI_AXIS_X,
```

```
10000, 4000, 1000, 300, 1000, 300);

// ……
if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }
pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_pls_smove

13.8 pos_smove

MMC_INT16U **pmc4bpci_pos_smove**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32 *pos*,MMC_INT16 *vel*,MMC_INT16 *acc*, MMC_INT16 *acac,* MMC_INT16 *dec,* MMC_INT16 *dcac*);

(1) 설명

지정위치로 S 자 가감속도커브 드라이브를 합니다. 드라이브완료까지 기다리지 않습니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- pos: 설정된 논리/실 위치값(범위: -2,147,483,648~2,147,483,647)을 기준으
- 로 옵셋범위(–268,435,455~268,435,455)내에서 설정 가능.
- vel: 속도(범위: 1~8,000)
- acc: 가(감)속도(범위: 1~8,000)
- acac: 가가속도(범위: 1~65,535)
- dec: 감속도(범위: 1~8,000)
- dcac: 감속도 증가율(범위: 1~65,535)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }
```

```
// ……
rtn = pmc4bpci_pos_smove(MMC_CARD_NO, PMC4BPCI_AXIS_X, 10000,
                                    2000, 200, 200, 200, 200);
if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }
pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_pos_smove_wait, pmc4bpci_smove_stop

13.9 pos_smove_wait

MMC_INT16U **pmc4bpci_pos_smove_wait**(MMC_INT16U *id*, MMC_INT16U *axis*,MMC_INT32 *pls*,MMC_INT16 *vel*,MMC_INT16 *acc*, MMC_INT16 *acac,* MMC_INT16 *dec,* MMC_INT16 *dcac*);

(1) 설명

지정위치로 S 자 가감 속도커브 드라이브를 합니다. 드라이브완료까지 기다린다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- pos: 설정된 논리/실 위치값(범위: -2,147,483,648~2,147,483,647)을 기준으로 옵셋범위(–268,435,455~268,435,455)내에서 설정 가능.
- vel: 속도(범위: 1~8,000)
- acc: 가(감)속도(범위: 1~8,000)
- acac: 가가속도(범위: 1~65,535)
- dec: 감속도(범위: 1~8,000)
- dcac: 감속도 증가율(범위: 1~65,535)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정된 내부 타임아웃시간에 의해 지정시간이 끝나 함수를 종료할 경우에는 PMC4BPCI_TIMEOUT_ERR 을 리턴합니다. 실행결과 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }
```

```
// ……
rtn = pmc4bpci_pos_smove_wait(MMC_CARD_NO, PMC4BPCI_AXIS_X,
10000, 2000, 200, 200, 200, 200);
if(rtn!=MMC_OK)
{printf("Fail!\n");     }
else
{printf("Complete OK!\n");    }
pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_pos_smove, pmc4bpci_smove_stop

13.10 pos_imove2

MMC_INT16U **pmc4bpci_pos_imove2**(MMC_INT16U *id*, MMC_INT16U **axis*,MMC_VERTEX **p*)

(1) 설명

2 축 직선 보간 드라이브를 합니다. (절대위치 이동)

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. 보간하고자 하는 축을 1 개씩 순서대로 입력
  1 축은 INP_AXIS1_X, INP_AXIS1_Y, INP_AXIS1_Z 중 하나 선택
  2 축은 INP_AXIS2_X, INP_AXIS2_Y, INP_AXIS2_Z 중 하나 선택
- p: 축의 순서와 동일하게 목표 펄스 입력

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void WINAPI isr_sub( void)
{
        // MMC Interrupt
}

void main()
{
        MMC_INT16U stat;

        stat = pmc4bpci_open(MMC_CARD_NO, isr_sub);
        if(stat!=MMC_OK)
        {
                printf("\nERROR : Can't open 'PMC-4B-PCI(ID=%d)' driver\n",
                                                                MMC_CARD_NO);

        // 열린 윈도드라이버 닫기
        pmc4bpci_close_all();
        return;
```

```
        }

        // 2 축 직선 보간 드라이브
        // ex) X 축 +2000, Z 축 -200 만큼 직선 보간
        // axis     = [WR5_INP_AXIS1_X, WR5_INP_AXIS2_Y]
        // p        = [2000, -2000]
        {
        // 보간시 1 축에지정축, 보간시 2 축에지정축
        MMC_INT16U work_plane_axis[2] = {WR5_INP_AXIS1_X,
WR5_INP_AXIS2_Y};
        MMC_VERTEX p = {20000, -20000, };
        MMC_INT16U axis = PMC4BPCI_AXIS1 | PMC4BPCI_AXIS2 |
PMC4BPCI_AXIS3 | PMC4BPCI_AXIS4 ;

        // 초기화
        pmc4bpci_set_accoffset(MMC_CARD_NO, axis, 8);// AO = 8(reset 시 설정치 = 8)
        pmc4bpci_set_range(MMC_CARD_NO, axis, 8000000); // (배율 = 1      )
        pmc4bpci_set_acac(MMC_CARD_NO,
axis,101);//K=101(가가속도=619kpps/sec²)
        pmc4bpci_set_acc(MMC_CARD_NO, axis,1000);// A=1000(가속도=125kpps/sec)
        pmc4bpci_set_dec(MMC_CARD_NO, axis,1000);// D=1000(감속도=125kpps/sec)
        pmc4bpci_set_startv(MMC_CARD_NO, axis,1000);//
SV=1000(초기속도=1000pps)
        pmc4bpci_set_speed(MMC_CARD_NO, axis,4000);//V=4000(드라이브속도 4000PPS)
        pmc4bpci_set_pulse(MMC_CARD_NO, axis,100000);//P=100000(보간종점설정)
        pmc4bpci_set_lpcounter(MMC_CARD_NO, axis,0);// LP= 0(논리위치카운터설정)

        // 2 축 직선 보간 드라이브
        pmc4bpci_pos_imove2(MMC_CARD_NO, work_plane_axis, &p);

        // 운전을 멈출때까지 2000msec 기다림
        pmc4bpci_wait_timeout(MMC_CARD_NO, axis, 2000);

        printf("\nComplete OK!\n");
        }

        // 열린 윈도드라이버 닫기
        pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_pos_iarc, pmc4bpci_pos_iarca, pmc4bpci_pos_imove3,

(6) 참고매크로

INP_AXIS1_X, INP_AXIS1_Y, INP_AXIS1_Z, INP_AXIS2_X, INP_AXIS2_Y, INP_AXIS2_Z, INP_AXIS3_X, INP_AXIS3_Y, INP_AXIS3_Z

13.11 pls_imove2

MMC_INT16U **pmc4bpci_pls_imove2**(MMC_INT16U *id*, MMC_INT16U **axis*,MMC_VERTEX **p*)

(1) 설명

2 축 직선 보간 드라이브를 합니다. (상대위치 이동)

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. 보간하고자 하는 축을 1 개씩 순서대로 입력
  1 축은 INP_AXIS1_X, INP_AXIS1_Y, INP_AXIS1_Z 중 하나 선택
  2 축은 INP_AXIS2_X, INP_AXIS2_Y, INP_AXIS2_Z 중 하나 선택
- p: 축의 순서와 동일하게 목표 펄스 입력

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void WINAPI isr_sub( void)
{
        // MMC Interrupt
}

void main()
{
        MMC_INT16U stat;

        stat = pmc4bpci_open(MMC_CARD_NO, isr_sub);
        if(stat!=MMC_OK)
        {
                printf("\nERROR : Can't open 'PMC-4B-PCI(ID=%d)' driver\n",
                                                                MMC_CARD_NO);

        // 열린 윈도드라이버 닫기
        pmc4bpci_close_all();
        return;
        }
```

```
		// 2 축 직선 보간 드라이브
		// ex) X 축 +2000, Z 축 -200 만큼 직선 보간
		// axis     = [WR5_INP_AXIS1_X, WR5_INP_AXIS2_Y]
		// p        = [2000, -2000]
		{
		// 보간시 1 축에지정축, 보간시 2 축에지정축
		MMC_INT16U work_plane_axis[2] = {WR5_INP_AXIS1_X,
WR5_INP_AXIS2_Y};
		MMC_VERTEX p = {20000, -20000, };
		MMC_INT16U axis = PMC4BPCI_AXIS1 | PMC4BPCI_AXIS2 |
PMC4BPCI_AXIS3 | PMC4BPCI_AXIS4 ;

		// 초기화
		pmc4bpci_set_accoffset(MMC_CARD_NO, axis, 8);// AO = 8(reset 시 설정치 = 8)
		pmc4bpci_set_range(MMC_CARD_NO, axis, 8000000); // (배율 = 1 )
		pmc4bpci_set_acac(MMC_CARD_NO,
axis,101);//K=101(가가속도=619kpps/sec²)
		pmc4bpci_set_acc(MMC_CARD_NO, axis,1000);// A=1000(가속도=125kpps/sec)
		pmc4bpci_set_dec(MMC_CARD_NO, axis,1000);// D=1000(감속도=125kpps/sec)
		pmc4bpci_set_startv(MMC_CARD_NO, axis,1000);//
SV=1000(초기속도=1000pps)
		pmc4bpci_set_speed(MMC_CARD_NO, axis,4000);//V=4000(드라이브속도 4000PPS)
		pmc4bpci_set_pulse(MMC_CARD_NO, axis,100000);//P=100000(보간종점설정)
		pmc4bpci_set_lpcounter(MMC_CARD_NO, axis,0);// LP= 0(논리위치카운터설정)

		// 2 축 직선 보간 드라이브
		pmc4bpci_pls_imove2(MMC_CARD_NO, work_plane_axis, &p);

		// 운전을 멈출때까지 2000msec 기다림
		pmc4bpci_wait_timeout(MMC_CARD_NO, axis, 2000);

		printf("\nComplete OK!\n");
		}

		// 열린 윈도드라이버 닫기
		pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_pos_iarc, pmc4bpci_pos_iarca, pmc4bpci_pos_imove3,

(6) 참고매크로

INP_AXIS1_X, INP_AXIS1_Y, INP_AXIS1_Z, INP_AXIS2_X, INP_AXIS2_Y, INP_AXIS2_Z, INP_AXIS3_X, INP_AXIS3_Y, INP_AXIS3_Z

13.12 pos_imove3

MMC_INT16U **pmc4bpci_pos_imove3**(MMC_INT16U *id*, MMC_INT16U **axis*,MMC_VERTEX **p*);

(1) 설명

3 축 직선 보간 드라이브를 합니다. (절대위치 이동)

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. 보간하고자 하는 축을 1 개씩 순서대로 입력
  1 축은 INP_AXIS1_X, INP_AXIS1_Y, INP_AXIS1_Z 중 하나 선택
  2 축은 INP_AXIS2_X, INP_AXIS2_Y, INP_AXIS2_Z 중 하나 선택
  3 축은 INP_AXIS3_X, INP_AXIS3_Y, INP_AXIS3_Z 중 하나 선택
- p: 축의 순서와 동일하게 목표 펄스 입력

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../../include/pmc4bpci.h"

#define MMC_CARD_NO     15


void WINAPI isr_sub( void)
{
        // MMC Interrupt
}

void main()
{
        MMC_INT16U stat;

        stat = pmc4bpci_open(MMC_CARD_NO, isr_sub);
        if(stat!=MMC_OK)
        {
printf("\nERROR : Can't open 'PMC-4B-PCI(ID=%d)' driver\n",
                                        MMC_CARD_NO);
```

```
                    // 열린 윈도드라이버 닫기
                    pmc4bpci_close_all();
                    return;
          }

          // 3 축 직선 보간 드라이브
          // ex) X 축 +2000, Y 축 -200, Z 축 1000 만큼 직선 보간
          // work_plane_axis          = [WR5_INP_AXIS1_X, WR5_INP_AXIS2_Y,
WR5_INP_AXIS3_Z]
          // p        = [2000, -200, 1000]
          {
                    // 보간시 1 축에 X 축,      보간시 2 축에 Y, 보간시 3 축에 Z 축
                    MMC_INT16U work_plane_axis[3] = {WR5_INP_AXIS1_X,
                              WR5_INP_AXIS2_Y, WR5_INP_AXIS3_Z};
                    MMC_VERTEX p = {20000, -20000, 10000 };
                    MMC_INT16U axis = PMC4BPCI_AXIS1 | PMC4BPCI_AXIS2 |
                                        PMC4BPCI_AXIS3 | PMC4BPCI_AXIS4 ;

                    // 초기화
                    pmc4bpci_set_accoffset(MMC_CARD_NO, axis, 8);// AO = 8
(reset 시 설정치 = 8)
                    pmc4bpci_set_range(MMC_CARD_NO, axis,PMC4BPCI_RANGE);
// R = 8000000     (배율 = 1          )
                    pmc4bpci_set_acac(MMC_CARD_NO, axis,101);
                                        // K = 101(가가속도 = 619kpps/sec²)
                    pmc4bpci_set_acc(MMC_CARD_NO, axis,1000);
                                        // A = 1000(가속도 = 125kpps/sec)
                    pmc4bpci_set_dec(MMC_CARD_NO, axis,1000);
// D = 1000(감속도=125kpps/sec)
                    pmc4bpci_set_startv(MMC_CARD_NO, axis,1000);
// SV= 1000        (초기속도 = 1000pps)
                    pmc4bpci_set_speed(MMC_CARD_NO, axis,4000);
                                        // V = 4000(드라이브속도 = 4000PPS)
                    pmc4bpci_set_pulse(MMC_CARD_NO, axis,100000);
// P = 100000      (보간종점설정 = 100000)
                    pmc4bpci_set_lpcounter(MMC_CARD_NO, axis,0);
// LP= 0(논리위치카운터설정 = 0)

                    // 3 축 직선 보간 드라이브
                    pmc4bpci_pos_imove3(MMC_CARD_NO, work_plane_axis, &p);

                    // 운전을 멈출때까지 기다림
                    pmc4bpci_wait(MMC_CARD_NO, axis);

                    printf("\nComplete OK!\n");
          }

          // 열린 윈도드라이버 닫기
          pmc4bpci_close_all();
}
```


(5) 참고함수

pmc4bpci_pos_iarc, pmc4bpci_pos_iarca, pmc4bpci_pos_imove2

(6) 참고매크로

INP_AXIS1_X, INP_AXIS1_Y, INP_AXIS1_Z, INP_AXIS2_X, INP_AXIS2_Y, INP_AXIS2_Z, INP_AXIS3_X, INP_AXIS3_Y, INP_AXIS3_Z

13.13 pls_imove3

MMC_INT16U **pmc4bpci_pls_imove3**(MMC_INT16U *id*, MMC_INT16U **axis*,MMC_VERTEX **p*);

(1) 설명

3 축 직선 보간 드라이브를 합니다. (상대위치 이동)

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. 보간하고자 하는 축을 1 개씩 순서대로 입력
  1 축은 INP_AXIS1_X, INP_AXIS1_Y, INP_AXIS1_Z 중 하나 선택
  2 축은 INP_AXIS2_X, INP_AXIS2_Y, INP_AXIS2_Z 중 하나 선택
  3 축은 INP_AXIS3_X, INP_AXIS3_Y, INP_AXIS3_Z 중 하나 선택
- p: 축의 순서와 동일하게 목표 펄스 입력

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../../include/pmc4bpci.h"

#define MMC_CARD_NO     15


void WINAPI isr_sub( void)
{
        // MMC Interrupt
}

void main()
{
        MMC_INT16U stat;

        stat = pmc4bpci_open(MMC_CARD_NO, isr_sub);
        if(stat!=MMC_OK)
        {
printf("\nERROR : Can't open 'PMC-4B-PCI(ID=%d)' driver\n",
                                        MMC_CARD_NO);
```

```
                    // 열린 윈도드라이버 닫기
                    pmc4bpci_close_all();
                    return;
          }

          // 3 축 직선 보간 드라이브
          // ex) X 축 +2000, Y 축 -200, Z 축 1000 만큼 직선 보간
          // work_plane_axis          = [WR5_INP_AXIS1_X, WR5_INP_AXIS2_Y,
WR5_INP_AXIS3_Z]
          // p       = [2000, -200, 1000]
          {
                    // 보간시 1 축에 X 축, 보간시 2 축에 Y, 보간시 3 축에 Z 축
                    MMC_INT16U work_plane_axis[3] = {WR5_INP_AXIS1_X,
                              WR5_INP_AXIS2_Y, WR5_INP_AXIS3_Z};
                    MMC_VERTEX p = {20000, -20000, 10000 };
                    MMC_INT16U axis = PMC4BPCI_AXIS1 | PMC4BPCI_AXIS2 |
                                        PMC4BPCI_AXIS3 | PMC4BPCI_AXIS4 ;

                    // 초기화
                    pmc4bpci_set_accoffset(MMC_CARD_NO, axis, 8);// AO = 8
(reset 시 설정치 = 8)
                    pmc4bpci_set_range(MMC_CARD_NO, axis,PMC4BPCI_RANGE);
// R = 8000000     (배율 = 1          )
                    pmc4bpci_set_acac(MMC_CARD_NO, axis,101);
                                        // K = 101(가가속도 = 619kpps/sec²)
                    pmc4bpci_set_acc(MMC_CARD_NO, axis,1000);
                                        // A = 1000(가속도 = 125kpps/sec)
                    pmc4bpci_set_dec(MMC_CARD_NO, axis,1000);
// D = 1000(감속도=125kpps/sec)
                    pmc4bpci_set_startv(MMC_CARD_NO, axis,1000);
// SV= 1000        (초기속도 = 1000pps)
                    pmc4bpci_set_speed(MMC_CARD_NO, axis,4000);
                                        // V = 4000(드라이브속도 = 4000PPS)
                    pmc4bpci_set_pulse(MMC_CARD_NO, axis,100000);
// P = 100000      (보간종점설정 = 100000)
                    pmc4bpci_set_lpcounter(MMC_CARD_NO, axis,0);
// LP= 0(논리위치카운터설정 = 0)

                    // 3 축 직선 보간 드라이브
                    pmc4bpci_pls_imove3(MMC_CARD_NO, work_plane_axis, &p);

                    // 운전을 멈출때까지 기다림
                    pmc4bpci_wait(MMC_CARD_NO, axis);

                    printf("\nComplete OK!\n");
          }

          // 열린 윈도드라이버 닫기
          pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_pos_iarc, pmc4bpci_pos_iarca, pmc4bpci_pos_imove2

(6) 참고매크로

INP_AXIS1_X, INP_AXIS1_Y, INP_AXIS1_Z, INP_AXIS2_X, INP_AXIS2_Y, INP_AXIS2_Z, INP_AXIS3_X, INP_AXIS3_Y, INP_AXIS3_Z

13.14 pos_iarc

MMC_INT16U **pmc4bpci_pos_iarc**(MMC_INT16U *id*, MMC_INT16U **axis*, MMC_VERTEX **cp*,MMC_VERTEX **ep*, MMC_INT16 *dir*);

(1) 설명

2 축 원호보간 드라이브를 합니다. 선택 축(*axis*)은 반드시 2 축이어야 하며 현재위치를 호의 시작으로 *cp* 점을 호의 중심점으로 하고 *dir* 방향으로 끝점(*ep* 점)의 수평위치까지 호를 그린다. *dir* 방향이 +이면 CCW, -이면 CW 로 호를 그린다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. 보간하고자 하는 축을 1 개씩 순서대로 입력
  1 축은 INP_AXIS1_X, INP_AXIS1_Y, INP_AXIS1_Z 중 하나 선택
  2 축은 INP_AXIS2_X, INP_AXIS2_Y, INP_AXIS2_Z 중 하나 선택
  cp: 현재위치를 기준으로 상대위치로 중심점으로 지정합니다.
- ep: 현재위치를 기준으로 상대위치로 목표점으로 지정합니다.
- dir: 원의 회전방향을 입력합니다. (만약 dir>0 이면 CCW, 그렇지 않다면 CW)

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void WINAPI isr_sub( void)
{
        // MMC Interrupt
}

void main()
{
        MMC_INT16U stat;

        stat = pmc4bpci_open(MMC_CARD_NO, isr_sub);
        if(stat!=MMC_OK)
        {
```

```
                printf("\nERROR : Can't open 'PMC-4B-PCI(ID=%d)' driver\n",
                                                        MMC_CARD_NO);

                // 열린 윈도드라이버 닫기
                pmc4bpci_close_all();
                return;
        }

        // 2 축 원호 보간드라이브
        // ex) R=20000, X 축 Center 0, Y 축 Center -20000, CCW
        // axis     = [X, Y, 0]
        // cp       = [-20000, 0, 0]
        // ep     = [-40000, 0, 0]
        // dir      = 1 (dir≥0 then ccw else cw)
        {
        //보간시 1 축에지정축, 보간시 2 축에지정축
        MMC_INT16U work_plane_axis[2] = {WR5_INP_AXIS1_X,
WR5_INP_AXIS2_Y};
        MMC_VERTEX cp = {-20000, 0, };
        MMC_VERTEX ep = {-40000, 0, };
        MMC_INT16U axis = PMC4BPCI_AXIS1 | PMC4BPCI_AXIS2 |
PMC4BPCI_AXIS3 |
                                                PMC4BPCI_AXIS4 ;

                // 2 축 원호 보간 드라이브
                pmc4bpci_pos_iarc(MMC_CARD_NO, work_plane_axis, &cp, &ep, 1);

                // 운전을 멈출때까지 기다림
                pmc4bpci_wait(MMC_CARD_NO, axis);

                printf("\nComplete OK!\n");
        }

        // 열린 윈도드라이버 닫기
        pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_pos_iarca

(6) 참고매크로

INP_AXIS1_X, INP_AXIS1_Y, INP_AXIS1_Z, INP_AXIS2_X, INP_AXIS2_Y, INP_AXIS2_Z, INP_AXIS3_X, INP_AXIS3_Y, INP_AXIS3_Z

13.15 pos_iarca

MMC_INT16U **pmc4bpci_pos_iarca**(MMC_INT16U *id*, MMC_INT16U **axis*, MMC_VERTEX **cp*,MMC_DOUBLE *ang*);

(1) 설명

2 축 원호보간 드라이브 합니다. 선택 축(*axis*)은 반드시 2 축이어야 합니다. 현재위치, 각을 기준으로 *ang* 만큼 회전합니다. *ang* 값이 +(plus)면 CCW 방향으로 회전, -(minus)면 CW 방향으로 회전합니다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. 보간하고자 하는 축을 1 개씩 순서대로 입력
  1 축은 INP_AXIS1_X, INP_AXIS1_Y, INP_AXIS1_Z 중 하나 선택
  2 축은 INP_AXIS2_X, INP_AXIS2_Y, INP_AXIS2_Z 중 하나 선택
- cp: 현재위치를 기준(0,0)으로 하고 상대좌표인 중심점을 지정합니다.
- ang: 현재각을 기준각으로 하고 상대각인 회전각(*ang*)을 지정합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../../include/pmc4bpci.h"

#define MMC_CARD_NO     15

void WINAPI isr_sub( void)
{
        // MMC Interrupt
}

void main()
{
        MMC_INT16U stat;

        stat = pmc4bpci_open(MMC_CARD_NO, isr_sub);
        if(stat!=MMC_OK)
        {
                printf("\nERROR : Can't open 'PMC-4B-PCI(ID=%d)' driver\n",
                                                        MMC_CARD_NO);
```

```
                    // 열린 윈도드라이버 닫기
                    pmc4bpci_close_all();
                    return;
            }

            // 2 축 원호 보간드라이브
            // ex) R=20000, X 축 Center 0, Y 축 Center -20000, CCW
            // axis     = [X, Y, 0]
            // cp       = [-20000, 0, 0]
            {
            // 보간시 1 축에지정축, 보간시 2 축에지정축
            MMC_INT16U work_plane_axis[2] = {WR5_INP_AXIS1_X,
WR5_INP_AXIS2_Y};
            MMC_VERTEX cp = {-20000, 0, };
            MMC_INT16U axis = PMC4BPCI_AXIS1 | PMC4BPCI_AXIS2 |
PMC4BPCI_AXIS3 |
                                                PMC4BPCI_AXIS4 ;

            // 2 축 원호 보간 드라이브
            pmc4bpci_pos_iarca(MMC_CARD_NO, work_plane_axis, &cp, 30.0);

            // 운전을 멈출때까지 기다림
            pmc4bpci_wait(MMC_CARD_NO, axis);

            printf("\nComplete OK!\n");
            }

            // 열린 윈도드라이버 닫기
            pmc4bpci_close_all();
}
```

(5) 참고함수

pmc4bpci_pos_iarca

(6) 참고매크로

INP_AXIS1_X, INP_AXIS1_Y, INP_AXIS1_Z, INP_AXIS2_X, INP_AXIS2_Y, INP_AXIS2_Z, INP_AXIS3_X, INP_AXIS3_Y, INP_AXIS3_Z

14 원점복귀

14.1 homesearch_sw

MMC_INT16U **pmc4bpci_homesearch_sw**(MMC_INT16U *id*, MMC_INT16U *axis*, MMC_INT16 *vel*);

(1) 설명

선택축의 원점복귀를 합니다(논리위치를 기준으로 원점을 찾음). 직선보간을 사용하지 않고 정량드라이브를 사용하므로 최단거리로 이동하지 않는다.

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: 축을 선택합니다. PMC-4B-PCI 는 4 개의 축으로 이루어져 있습니다.
- vel: 해당축의 원점복귀 속도를 지정합니다.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../include/pmc4bpci.h"

void main()
{
        #define MMC_CARD_NO     15
MMC_INT16U    rtn;
        int OpenFlag;

        OpenFlag = pmc4bpci_open(MMC_CARD_NO, NULL);
        if(OpenFlag!=MMC_OK)
        {
                printf("Initialize error!\n");
                return;
        }

// ……
rtn = pmc4bpci_homesearch_sw(MMC_CARD_NO,
 PMC4BPCI_AXIS_X|PMC4BPCI_AXIS_Y, 2000);
// ……

if(rtn!=MMC_OK)
{printf("Fail!\n");     }
```

```
else
{printf("Complete OK!\n");    }

pmc4bpci_close_all();

}
```


14.2 homesearch

MMC_INT16U **pmc4bpci_homesearch**(MMC_INT16U *id*, MMC_INT16U *axis*, stHomeSearch* *param*);

(1) 설명

선택축의 원점복귀를 합니다. 소프트웨어 리미트는 반드시 Disable 시켜야 하며, Move Speed 가 Homesearch Speed 보다 빨라야 합니다. 원점복귀에 관한 자세한 내용은 “사용자 매뉴얼 2.5 자동 원점 복귀 출력”을 참조합니다

(2) 인자

- id: ID 번호를 입력합니다. (범위: 0~15)
- axis: PMC4BPCI 의 4 개 축에서 1 개의 축만을 선택합니다.
- param: 해당축의 원점복귀 설정 구조체.

(3) 리턴값

PMC-4B-PCI 보드에서 지원하는 ID 의 범위를 벗어났을 경우에는 MMC_INVALID_CARD 를 리턴하며, 축의 범위를 벗어났을 경우에는 MMC_INVALID_AXIS 를 리턴합니다. 설정에 성공하면 PMC4BPCI_OK 를 리턴합니다. 전역 에러 검사 변수인 gError 에도 같은 값이 기록됩니다.

(4) 사용 예

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#include "../../include/pmc4bpci.h"

#define MMC_CARD_NO     15


void WINAPI isr_sub( void)
{
                        // MMC Interrupt
}

void main()
{
            MMC_INT16U stat;
            MMC_INT32U t = 0L;
            stHomeSearch hs = {
                        0x5D00,             // WR6
                        0x495F,             // WR7
                        0x00000DAC,         // 3500pulse
                        0x0032,             // home-search speed
                        0x07d0              // move speed
            };

            stat = pmc4bpci_open(MMC_CARD_NO, isr_sub);
```

```
		if(stat!=MMC_OK)
		{
printf("\nERROR : Can't open 'PMC-4B-PCI(ID=%d)' driver\n", MMC_CARD_NO);

			// 열린 윈도드라이버 닫기
			pmc4bpci_close_all();
			return;
		}
		printf("\nMOVE...\n", MMC_CARD_NO);

		{
			// X, Y 축 홈서치 기능
			MMC_INT16U axis = PMC4BPCI_AXIS_X | PMC4BPCI_AXIS_Y;
			MMC_INT32 x, y;
			MMC_INT16U xstat, ystat;

			// 기본값 지정후에 아래처럼 별도지정 가능
			hs.move_speed = 5000;

			pmc4bpci_homesearch(MMC_CARD_NO, axis, &hs);

			// 운전을 멈출때까지 기다림
			while(pmc4bpci_inw(MMC_CARD_NO, rr0) & (PMC4BPCI_AXIS_X>>8))
			{
			// 상태값은 반드시 한축씩만 읽기
			// 내부 카운터 위치
			x = pmc4bpci_get_logicalposition(MMC_CARD_NO,
PMC4BPCI_AXIS_X);
			y = pmc4bpci_get_logicalposition(MMC_CARD_NO,
PMC4BPCI_AXIS_Y);
			xstat = pmc4bpci_rr2(MMC_CARD_NO, PMC4BPCI_AXIS_X);
// home search 상태값
			ystat = pmc4bpci_rr2(MMC_CARD_NO, PMC4BPCI_AXIS_Y);
// home search 상태값
			printf("\rStat=0x%04X,0x%04X : Pos=%d,%d    ", xstat, ystat, x, y);
			}

			printf("\nComplete OK!\n");
		}

		// 열린 윈도드라이버 닫기
		pmc4bpci_close_all();
}
```


15 부록

15.1 'Autonics' 로고 연속 보간을 위한 DXF 만들기

AutoCAD 에서 폴리라인을 사용하여 드로잉한 도면을 DXF(Data eXchange Format)형식의 파일로 만들어 MMC 라이브러리에서 직접 사용할 수 있습니다.
아래의 작업순서를 참고하십시오.

- 작업순서

1st AutoCAD 에서 폴리라인 명령을 사용하여 그림 1.을 드로잉 합니다.

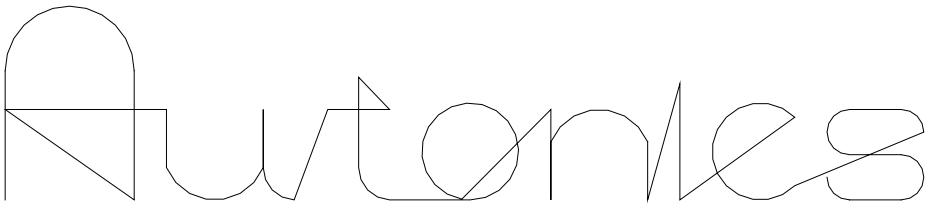


그림 A.1 오토캐드의 폴리라인으로 구성된 'Autonics' 로고

2nd AutoCAD 에서 "DXFOUT" 명령을 사용하여 DXF 형식의 파일 생성합니다.
Command : DXFOUT

3rd 2nd 에 의해 생성된 리스트 A.1 또는 리스트 A.2 를 텍스트 편집기에서 리스트 A.3 형식으로 편집합니다. (리스트 A.1 또는 리스트 A.2 는 출력된 파일의 일부분임.)

리스트 A.1 AutoCAD 2000 의 폴리라인 부분 발췌

`…	10	0.0	42
2	125.0	42	2.414213562373097
ENTITIES	20	22.90376554843131	10
0	25.0	10	611.9095991350035
LWPOLYLINE	42	353.8812881779981	20
5	0.6666666666666666	20	10.98349570550448
2B	10	0.569709262042181	10
330	200.0	3	711.9095991350035
1F	20	10	20
100	25.0	422.8930948405079	52.5
AcDbEntity	10	20	42
8	200.0	69.99999999999997	0.4142135623730951
0	20	10	10
100	69.99999999999997	422.8930948405079	694.4095991350035
AcDbPolyline	10	20	20
90	200.0	0.0	70.0
38	20	10	10
70	25.0	422.8930948405079	654.4095991350035
0	42	20	20
43	0.4016633628195203	44.99999999999997	70.0
0.0	10	42	42
10	223.9465474202539	-0.666666666666666	0.9999999999999998
0.0	20		10

```
 20
0.0
 10
0.0
 20
100.0
 42
-1.0
 10
100.0
 20
100.0
 10
99.99999999999998
 20
0.0
 10
0.0
 20
70.0
 10
99.99999999999998
 20
69.99999999999997
 10
125.0
 20
69.99999999999997
```

```
0.0
 10
250.0
 20
69.99999999999997
 10
297.8930948405079
 20
69.99999999999997
 10
273.9465474202539
 20
94.99999999999998
 10
273.9465474202539
 20
25.0
 42
0.4016633628195203
 10
297.8930948405079
 20
0.0
 10
360.3930948405079
 20
```

```
6
 10
497.8930948405079
 20
44.99999999999997
 10
497.8930948405079
 20
0.0
 10
522.8930948405079
 20
89.99999999999998
 10
522.8930948405079
 20
79.99999999999997
 10
522.8930948405079
 20
69.99999999999997
 10
522.8930948405079
 20
0.0
 10
611.9095991350035
 20
64.01650429449552
```

```
654.4095991350035
 20
35.0
 10
694.4095991350035
 20
35.0
 42
-0.9999999999999998
 10
694.4095991350035
 20
0.0
 10
654.4095991350035
 20
0.0
 42
-0.4142135623730951
 10
636.9095991350035
 20
17.5
  0
ENDSEC
...
```

리스트 A.2 AutoCAD r12 의 폴리라인 부분 발췌

```
...
0
SECTION
  2
ENTITIES
  0
POLYLINE
  5
2B
  8
0
 66
     1
 10
0.0
 20
0.0
 30
0.0
  0
VERTEX
  5
93
  8
0
 10
0.0
 20
0.0
 30
```

```
0.0
  0
VERTEX
  5
9C
  8
0
 10
200.0
 20
69.999999999999972
 30
0.0
  0
VERTEX
  5
9D
  8
0
 10
200.0
 20
25.0
 30
0.0
 42
0.4016633628195203
  0
VERTEX
  5
```

```
5
A6
  8
0
 10
422.8930948405079
 20
69.999999999999972
 30
0.0
  0
VERTEX
  5
A7
  8
0
 10
422.8930948405079
 20
0.0
 30
0.0
  0
VERTEX
  5
A8
  8
0
 10
422.8930948405079
```

```
611.90959913500353
 20
10.983495705504479
 30
0.0
  0
VERTEX
  5
B1
  8
0
 10
711.90959913500353
 20
52.5
 30
0.0
 42
0.4142135623730951
  0
VERTEX
  5
B2
  8
0
 10
694.40959913500353
 20
70.0
 30
```

```
0.0
  0
VERTEX
  5
94
  8
0
 10
0.0
 20
100.0
 30
0.0
 42
-1.0
  0
VERTEX
  5
95
  8
0
 10
100.0
 20
100.0
 30
0.0
  0
VERTEX
  5
96
  8
0
 10
99.999999999999986
 20
0.0
 30
0.0
  0
VERTEX
  5
97
  8
0
 10
0.0
 20
70.0
 30
0.0
  0
VERTEX
  5
98
  8
0
 10
99.999999999999986
 20
69.999999999999972
 30
0.0
  0
VERTEX
```

```
9E
  8
0
 10
223.94654742025389
 20
0.0
 30
0.0
  0
VERTEX
  5
9F
  8
0
 10
250.0
 20
69.999999999999972
 30
0.0
  0
VERTEX
  5
A0
  8
0
 10
297.8930948405079
 20
69.999999999999972
 30
0.0
  0
VERTEX
  5
A1
  8
0
 10
273.94654742025392
 20
94.999999999999986
 30
0.0
  0
VERTEX
  5
A2
  8
0
 10
273.94654742025392
 20
25.0
 30
0.0
 42
0.4016633628195203
  0
VERTEX
  5
A3
  8
0
```

```
 20
44.999999999999972
 30
0.0
 42
-0.6666666666666666
  0
VERTEX
  5
A9
  8
0
 10
497.8930948405079
 20
44.999999999999972
 30
0.0
  0
VERTEX
  5
AA
  8
0
 10
497.8930948405079
 20
0.0
 30
0.0
  0
VERTEX
  5
AB
  8
0
 10
522.89309484050796
 20
89.999999999999986
 30
0.0
  0
VERTEX
  5
AC
  8
0
 10
522.89309484050796
 20
79.999999999999972
 30
0.0
  0
VERTEX
  5
AD
  8
0
 10
522.89309484050796
 20
69.999999999999972
 30
```

```
0.0
  0
VERTEX
  5
B3
  8
0
 10
654.40959913500353
 20
70.0
 30
0.0
 42
0.9999999999999998
  0
VERTEX
  5
B4
  8
0
 10
654.40959913500353
 20
35.0
 30
0.0
  0
VERTEX
  5
B5
  8
0
 10
694.40959913500353
 20
35.0
 30
0.0
 42
-0.9999999999999998
  0
VERTEX
  5
B6
  8
0
 10
694.40959913500353
 20
0.0
 30
0.0
  0
VERTEX
  5
B7
  8
0
 10
654.40959913500353
 20
0.0
 30
0.0
```

```
  5
99
  8
0
 10
125.0
 20
69.999999999999972
 30
0.0
  0
VERTEX
  5
9A
  8
0
 10
125.0
 20
25.0
 30
0.0
 42
0.6666666666666666
  0
VERTEX
  5
9B
  8
0
 10
200.0
 20
25.0
 30
 10
297.8930948405079
 20
0.0
 30
0.0
  0
VERTEX
  5
A4
  8
0
 10
360.3930948405079
 20
0.0
 30
0.0
 42
22.903765548431309
  0
VERTEX
  5
A5
  8
0
 10
353.8812881779981
 20
0.5697092620421813
 30
0.0
  0
VERTEX
0.0
  0
VERTEX
  5
AE
  8
0
 10
522.89309484050796
 20
0.0
 30
0.0
  0
VERTEX
  5
AF
  8
0
 10
611.90959913500353
 20
64.016504294495519
 30
0.0
 42
2.4142135623730971
  0
VERTEX
  5
B0
  8
0
 10
 42
-0.4142135623730951
  0
VERTEX
  5
B8
  8
0
 10
636.90959913500353
 20
17.5
 30
0.0
  0
SEQEND
…
```

리스트 A.3 C++ 형태로 편집

```
        ……
    MMC_VERTEX Autonics_p[] = {
            // A
            {0.0, 0.0, 0},
            {0.0,100.0,-1.0},
            {100.0,100.0,0},
            {99.99999999999998,0.0,0},
            {0.0,70.0,0},
            {99.99999999999998,69.99999999999997,0},
            // u
            {125.0,69.99999999999997,0},
            {125.0,25.0,0.6666666666666666},
            {200.0,25.0,0},
            {200.0,69.99999999999997,0},
            {200.0,25.0,0.4016633628195203},
            {223.9465474202539,0.0,0},
            // t
            {250.0,69.99999999999997,0},
            {297.8930948405079,69.99999999999997,0},
            {273.9465474202539,94.99999999999998,0},
            {273.9465474202539,25.0,0.4016633628195203},
            {297.8930948405079,0.0,0},
            // o
            {360.3930948405079,0.0,22.90376554843131},
            {353.8812881779981,0.5697092620421813,0},
            // n
            {422.8930948405079,69.99999999999997,0},
            {422.8930948405079,0.0,0},
            {422.8930948405079,44.99999999999997,-
0.6666666666666666},
            {497.8930948405079,44.99999999999997,0},
            {497.8930948405079,0.0,0},
            // i
            {522.8930948405079,89.99999999999998,0},
            {522.8930948405079,79.99999999999997,0},
            {522.8930948405079,69.99999999999997,0},
            {522.8930948405079,0.0,0},
            // c

        {611.9095991350035,64.01650429449552,2.414213562373097},
```

```
                {611.9095991350035,10.98349570550448,0},
                // s
                {711.9095991350035,52.5,0.4142135623730951},
                {694.4095991350035,70.0,0},
                {654.4095991350035,70.0,0.9999999999999998},
                {654.4095991350035,35.0,0},
                {694.4095991350035,35.0,-0.9999999999999998},
                {694.4095991350035,0.0,0},
                {654.4095991350035,0.0,-0.4142135623730951},
                {636.9095991350035,17.5,0},
                // move to origin point
                {0.0, 0.0, 0}
        };
          ……..
```

DXF 출력 고려사항으로 연속된 하나의 폴리라인으로 그려야 제대로 된 순서를 찾을 수 있습니다. AutoCAD 의 drawing 순서는 대부분 그려진 순서를 따르지만 중간에 추가/편집 작업에 의해 그림의 순서가 바뀔 수 있기 때문에 위 그림의 모든 문자는 폴리라인에 의해 연결되어야 합니다. 그렇지 않으면 문자순서를 제대로 찾을 수가 없습니다.

폴리라인 DXF 형식.

표 A.1 DXF Format

인식코드	설명
0	명령/이름
100	명령/이름 (AutoCAD 2000)
10	X 좌표
20	Y 좌표
30	Z 좌표
42	호의 돌출 값

폴리라인은 AutoCAD r12 버전과 AutoCAD 2000 버전의 DXF 출력형식이 다르다. 위의 리스트.1 은 AutoCAD 2000 버전이며 리스트.2 는 AutoCAD r12 버전입니다.

15.2 매크로의 정의

(1) MMC 변수형

- 정의: MMC_INT8
  원형: typedef signed char MMC_INT8;
  설명: 부호있는 8bit 문자형입니다. 이 정의부분은 WinDriver 에서 생성한 소스와 호환됩니다.
- 정의: MMC_INT8U
  원형: typedef unsigned char MMC_INT8U;
  설명: 부호없는 8bit 문자형입니다. 이 정의부분은 WinDriver 에서 생성한 소스와 호환됩니다.
- 정의: MMC_INT16
  원형: typedef signed short int MMC_INT16;
  설명: 부호있는 16bit 정수형입니다. 이 정의부분은 WinDriver 에서 생성한 소스와 호환됩니다.
- 정의: MMC_INT16U
  원형: typedef unsigned long MMC_INT16U;
  설명: 부호없는 32bit 정수형입니다. 이 정의부분은 WinDriver 에서 생성한 소스와 호환됩니다.
- 정의: MMC_INT32
  원형: typedef long MMC_INT32;
  설명: 부호있는 32bit 정수형입니다.
- 정의: MMC_INT32U
  원형: typedef unsigned long MMC_INT32U;
  설명: 부호없는 32bit 정수형입니다.

- 정의: MMC_FLOAT
  원형: typedef float MMC_FLOAT;
  설명: 4 바이트 실수형입니다.
- 정의: MMC_DOUBLE
  원형: typedef double MMC_DOUBLE;
  설명: 8 바이트 실수형입니다.

- 정의: MMC_VOID
  원형: typedef void MMC_VOID;
  설명: void 형 입니다.
- 정의: MMC_VERTEX
  원형: typedef struct {

```
double x;    // X 좌표
double y;    // Y 좌표
union {
z;   // Z 좌표
k;   // 폴리라인에서 호의 돌출정도
}
} MMC_VERTEX;
```

설명: 폴리라인의 Vertex 정보와 호환됩니다.

(2) PMC4BPCI 레지스터 정의

- 정의: wr0~wr7, rr0~rr7
  설명: PMC4BPCI 에 사용되는 I/O address 의 offset 값을 갖는다. wr0~wr7(0x00~0x0e), rr0~rr7(0x00~0x0e)
- 정의: bp1p, bp1m, bp2p, bp2m, bp3p, bp3m
  설명: PMC4BPCI 에 사용되는 bit pattern 에 사용되는 I/O address 의 offset 값입니다.

(3) PMC-4B-PCI 보드 ID 설정 방법

- 함수: pmc4bpci_set_current_id(MMC_INT16U *id*);
  설명: PMC-4B-PCI 보드의 DIP S/W 로 설정한 ID 를 참고하여 현재 동작시킬 보드를 바꾼다.
  예: pmc4bpci_set_current_id(3);

(4) 축(axis) 지정 방법

사용 예: 축 1, 축 2 를 동시에 지정하는 방법

```
PMC4BPCI_AXIS1 | PMC4BPCI_AXIS2
```

또는 PMC4BPCI_AXIS1 + PMC4BPCI_AXIS2

또는 PMC4BPCI_AXIS_X | PMC4BPCI_AXIS_Y

또는 PMC4BPCI_AXIS_X + PMC4BPCI_AXIS_Y

또는 0x0300

```
pmc4bpci_set_axis(PMC4BPCI_AXIS1 | PMC4BPCI_AXIS2);
```

실제 mcx314 칩의 비트셋에 따릅니다. 아래 매크로 정의를 참고하십시오.

```
#define   PMC4BPCI_AXIS_NUM    4                  // 축의 수

#define   PMC4BPCI_AXIS1              0x0100
#define   PMC4BPCI_AXIS2              0x0200
#define   PMC4BPCI_AXIS3              0x0400
#define   PMC4BPCI_AXIS4              0x0800
#define   PMC4BPCI_AXIS_X             PMC4BPCI_AXIS1
#define   PMC4BPCI_AXIS_Y             PMC4BPCI_AXIS2
#define   PMC4BPCI_AXIS_Z             PMC4BPCI_AXIS3
#define   PMC4BPCI_AXIS_U             PMC4BPCI_AXIS4
```


(5) 보간 드라이브 축(axis) 지정 방법

사용 예: MMC_INT16U _ax[2]={INP_AXIS1_X, INP_AXIS2_Y};

```
#define INP_AXIS1_X          WR5_INP_AXIS1_X      // 제 1 축 지정 축 정의
#define INP_AXIS1_Y          WR5_INP_AXIS1_Y      //
#define INP_AXIS1_Z          WR5_INP_AXIS1_Z      //
#define INP_AXIS1_U          WR5_INP_AXIS1_U      //
#define INP_AXIS2_X          WR5_INP_AXIS2_X      // 제 2 축 지정 축 정의
#define INP_AXIS2_Y          WR5_INP_AXIS2_Y      //
#define INP_AXIS2_Z          WR5_INP_AXIS2_Z      //
#define INP_AXIS2_U          WR5_INP_AXIS2_U      //
#define INP_AXIS3_X          WR5_INP_AXIS3_X      // 제 3 축 지정 축 정의
#define INP_AXIS3_Y          WR5_INP_AXIS3_Y      //
#define INP_AXIS3_Z          WR5_INP_AXIS3_Z      //
#define INP_AXIS3_U          WR5_INP_AXIS3_U      //
```

(6) 드라이브 범위

PMC4BPCI 의 드라이브 범위는 8,000,000 입니다.

```
#define PMC4BPCI_RANGE       8000000
```

(7) 명령코드

```
#define PMC4BPCI_CMD_R                0x00    // Range 설정
#define PMC4BPCI_CMD_K                0x01    // 가가속도 설정 (1~65535),2
#define PMC4BPCI_CMD_A                0x02    // 가속도 설정(1~8000),2
#define PMC4BPCI_CMD_D                0x03    // 감속도 설정(1~8000),2
#define PMC4BPCI_CMD_SV       0x04    // 처음속도 설정(1~8000),2
#define PMC4BPCI_CMD_V                0x05    // drive 속도 설정(1~8000),2
#define PMC4BPCI_CMD_P                0x06    // 출력 pulse 수(0~268,435,455),4
                                      // 보간종점(-8,388,608~8,388,607),4
#define PMC4BPCI_CMD_DP       0x07    // manual 감속점 설정(0~2g),4
#define PMC4BPCI_CMD_C                0x08    // 원호중심점 설정(-8m~8m),4
#define PMC4BPCI_CMD_LP       0x09    // 논리위치 counter 설정(-2g~2g),4
#define PMC4BPCI_CMD_EP       0x0a    // 실위치 counter 설정(-2g~2g),4
#define PMC4BPCI_CMD_CP       0x0b    // COMP +register 설정(-2g~2g),4
#define PMC4BPCI_CMD_CM       0x0c    // COMP -register 설정(-2g~2g),4
#define PMC4BPCI_CMD_AO       0x0d    // 가속 counter offset 설정(0~65535),2
#define PMC4BPCI_CMD_NOP      0x0f    // NOP(축교환용)
#define PMC4BPCI_CMD_RST      0x8000  // Reset
```

(8) 데이터 읽어내기 명령 정의

```
#define PMC4BPCI_READ_LP      0x10    // 논리위치 counter 읽어내기 (-2g~2g),4
#define PMC4BPCI_READ_EP      0x11    // 실제위치 counter 읽어내기 (-2g~2g),4
#define PMC4BPCI_READ_CV      0x12    // 현재 drive 속도 읽어내기(1~8000),2
#define PMC4BPCI_READ_CA      0x13    // 현재 가압속도 읽어내기(1~8000),2
```

(9) 드라이브 명령 정의

```
#define PMC4BPCI_DRV_PF          0x20    // +방향 정량 drive
#define PMC4BPCI_DRV_MF   0x21   // -방향 정량 drive
#define PMC4BPCI_DRV_PC   0x22   // +방향 연속 drive
#define PMC4BPCI_DRV_MC   0x23   // -방향 연속 drive
#define PMC4BPCI_DRV_DH   0x24   // drive 개시 hold
#define PMC4BPCI_DRV_DF          0x25    // drive 개시 free/완료 status clear
#define PMC4BPCI_DRV_DDS  0x26   // drive 감속 정지
#define PMC4BPCI_DRV_DS   0x27   // drive 즉시 정지
```

(10) 보간 명령 정의

```
#define PMC4BPCI_INP_2LD  0x30   // 2 축 직선보간 drive
#define PMC4BPCI_INP_3LD  0x31   // 3 축 직선보간 drive
#define PMC4BPCI_INP_CW          0x32    // CW 원호 보간 drive
#define PMC4BPCI_INP_CCW  0x33   // CCW 원호 보간 drive
#define PMC4BPCI_INP_2BP  0x34   // 2 축 bit pattern 보간 drive
#define PMC4BPCI_INP_3BP  0x35   // 3 축 bit pattern 보간 drive
#define PMC4BPCI_INP_BPE  0x36   // BP register 기입 가능
#define PMC4BPCI_INP_BPD  0x37   // BP register 기입 불가능
#define PMC4BPCI_INP_BPS  0x38   // BP data stack
#define PMC4BPCI_INP_BPC  0x39   // BP data clear
#define PMC4BPCI_INP_SS          0x3a    // 보간 single step
#define PMC4BPCI_INP_DV1  0x3b   // 감속 유효
#define PMC4BPCI_INP_DV2  0x3c   // 감속 유효
#define PMC4BPCI_INP_IC          0x3d    // 보간 interrupt clear
```

(11) 쓰기 레지스터 비트 정의

- WR1 (Mode register 1)

```
#define WR1_IN0_L         0x0001  // 입력신호 IN0 에대한 논리레벨
#define WR1_IN0_E         0x0002  // 입력신호 IN0 에대한 유효/무효 설정
#define WR1_IN1_L         0x0004
#define WR1_IN1_E         0x0008
#define WR1_IN2_L         0x0010
#define WR1_IN2_E         0x0020
#define WR1_IN3_L         0x0040
#define WR1_IN3_E         0x0080
```

- interrupt 의 발생조건 설정 비트

```
#define WR1_INT_PULSE     0x0100  // drive pulse 의 rising edge
#define WR1_INT_PGECM     0x0200  // 논리/실제위치 pulse 값 ≥ COMP-register
#define WR1_INT_PLCM      0x0400  // 논리/실제위치 pulse 값 < COMP-register
#define WR1_INT_PLCP      0x0800  // 논리/실제위치 pulse 값 < COMP+register
```

```
#define WR1_INT_PGEP            0x1000  // 논리/실제위치 pulse 값 ≥
COMP+register
#define WR1_INT_CEND            0x2000  // 정속지역에서 pulse 출력을 종료할때
#define WR1_INT_CSTA            0x4000  // 정속지역에서 pulse 출력을 개시하였을때
#define WR1_INT_DEND            0x8000  // drive 가 종료하였을때
```

- WR2 (Mode register 2)

```
// COMP+register 를 software limit 로 설정
#define WR2_LMT_SLMTP 0x0001
// COMP-register 를 software limit 로 설정
#define WR2_LMT_SLMTM 0x0002
// hardware limit(nLMTP,nLMTM)가 active 될 경우 정지방식설정(0: 즉시정지,1: 감속정지)
#define WR2_LMT_LMTMD           0x0004
// +방향 limit 입력신호(nLMTP)의 논리 level 설정(0:low,1:high 에서 active)
#define WR2_LMT_HLMTP 0x0008
// -방향 limit 입력신호(nLMTM)의 논리 level 설정(0:low,1:high 에서 active)
#define WR2_LMT_HLMTM           0x0010
// COMP+/- register 의 비교대상(0:논리위치,1:실제위치와 비교)
#define WR2_LMT_CMPSL 0x0020
// drive pulse 의 출력방식(0:독립 2 pulse 방식,1:1 pulse 방식)
#define WR2_DRV_PLSMD 0x0040
// drive pulse 의 논리 level 설정(0:정논리,1:부논리)
#define WR2_DRV_PLS_L   0x0080
// drive pulse 의 방향출력신호의 논리 level 설정
// (0:+방향일때 low,-방향일때 hi,1:+방향일때 hi,-방향일때 low)
#define WR2_DRV_DIR_L   0x0100
// encoder 입력신호(nECA/PPIN,nECB/PMIN)의 출력방식(0:2 상 pulse,1:up/dn pulse 입력)
#define WR2_ENC_PINMD 0x0200
// encoder 2 상 pulse 입력의 분주비를 설정(D1D0 00=1/1, 01=1/2
#define WR2_ENC_PIND0  0x0400
// encoder 2 상 pulse 입력의 분주비를 설정        10=1/4, 11=무효)
#define WR2_ENC_PIND1  0x0800
// nALARM 입력 신호의 논리 level 을 설정(0: low, 1: high 에서 active)
#define WR2_SRV_ALM_L  0x1000
// servo motor alarm 용 입력신호 nALARM 의 유/무효 설정(0:무효,1:유효)
#define WR2_SRV_ALM_E  0x2000
// nINPOS 입력 신호의 논리 lovel 을 설정(0: low, 1: high 에서 active)
#define WR2_SRV_INP_L   0x4000
```

```
// servo motor 위치 결정 완료용 입력신호 nINPOS 의 유효/무효 설정(0: 무효,1: 유효)
#define WR2_SRV_INP_E   0x8000
```

- WR3 (Mode register 3)

```
// 가감속 정량 drive 의 감속방법(0: 자동감속, 1: manual 감속)
#define WR3_DRV_MANLD           0x0001
// 가감속 drive 감속시 영향을 받는 종류
// (0: 가속도의 값에 영향, 1: 감속도의 값에 영향-manual 방식일때만 사용-)
#define WR3_DRV_DSNSE 0x0002
// 직선가감속/S 자 가감속 설정(0: 직선가감속, 1: S 자가감속)
#define WR3_DRV_SACC   0x0004
// 외부입력신호 (D4D3 00=외부입력신호에의한 drive 조작무효, 01=연속 drive mode
#define WR3_DRV_EXOP0  0x0008
// 외부입력신호          10=정량 drive mode, 11=외부입력신호에의한 drive 조작무효
#define WR3_DRV_EXOP1  0x0010
// 출력신호 nOUT4~7 을 범용/drive 상태의 여부결정(0: 범용출력, 1: drive 상태표시)
#define WR3_OUT_OUTSL 0x0080
// 출력신호로 사용할때 사용(0: low level 사용, 1: high level 사용)
#define WR3_OUT_OUT4   0x0100
#define WR3_OUT_OUT5   0x0200
#define WR3_OUT_OUT6   0x0400
#define WR3_OUT_OUT7   0x0800
```

- WR4 (output register)

```
// 범용출력신호 nOUT0~3 의 출력을 설정(0: low level, 1: high level)
#define WR4_OUT_XOUT0 0x0001
#define WR4_OUT_XOUT1 0x0002
#define WR4_OUT_XOUT2 0x0004
#define WR4_OUT_XOUT3 0x0008
#define WR4_OUT_YOUT0 0x0010
#define WR4_OUT_YOUT1 0x0020
#define WR4_OUT_YOUT2 0x0040
#define WR4_OUT_YOUT3 0x0080
#define WR4_OUT_ZOUT0 0x0100
#define WR4_OUT_ZOUT1 0x0200
#define WR4_OUT_ZOUT2 0x0400
#define WR4_OUT_ZOUT3 0x0800
#define WR4_OUT_UOUT0 0x1000
#define WR4_OUT_UOUT1 0x2000
#define WR4_OUT_UOUT2 0x4000
#define WR4_OUT_UOUT3 0x8000
```


- WR5 (보간 mode register)

```
// 보간 drive 를 실행하는 제 1 축 지정
#define WR5_INP_AX10                    0x0001
// AX10AX11 00:X, 01:Y, 10:Z, 11:U
#define WR5_INP_AX11                    0x0002
// 보간 drive 를 실행하는 제 2 축 지정
#define WR5_INP_AX20                    0x0004
#define WR5_INP_AX21                    0x0008
// 보간 drive 를 실행하는 제 3 축 지정(2 축보간만 사용할경우 임의의 값을 사용)
#define WR5_INP_AX30                    0x0010
#define WR5_INP_AX31                    0x0020
// 보간 drive 의 선속일정 mode 를 설정
#define WR5_INP_LSPD0                   0x0100
// D1D0 00: 선속일정무효, 01: 2 축선속일정, 10: 설정불가, 11: 3 축선속일정
#define WR5_INP_LSPD1                   0x0200
// 외부전송여부(0: 전송안함, 1: 보간 drive 를 외부신호-EXPLSN-로 step 전송)
#define WR5_INP_EXPLS                   0x0800
// (0: 전송안함, 1: 보간 drive 를 command 로 step 전송)
#define  WR5_INP_CMPLS                  0x1000
// 연속보간시의 interrupt 발생 허가/금지 설정(0: 금지, 1: 허가)
#define WR5_INP_CIINT                   0x4000
// bit pattern 보간시 interrupt 발생 허가/금지 설정(0: 금지, 1: 허가)
#define WR5_INP_BPINT                   0x8000
// 보간시 제 1 축에 지정하고자 하는축 정의
#define WR5_INP_AXIS1_X                 0x0000
#define WR5_INP_AXIS1_Y                 WR5_INP_AX10
#define WR5_INP_AXIS1_Z                 WR5_INP_AX11
#define WR5_INP_AXIS1_U                 (WR5_INP_AX10|WR5_INP_AX11)
// 보간시 제 2 축에 지정하고자 하는축 정의
#define WR5_INP_AXIS2_X                 0x0000
#define WR5_INP_AXIS2_Y                 WR5_INP_AX20
#define WR5_INP_AXIS2_Z                 WR5_INP_AX21
#define WR5_INP_AXIS2_U                 (WR5_INP_AX20|WR5_INP_AX21)
// 보간시 제 3 축에 지정하고자 하는축 정의
#define WR5_INP_AXIS3_X                 0x0000
#define WR5_INP_AXIS3_Y                 WR5_INP_AX30
#define WR5_INP_AXIS3_Z                 WR5_INP_AX31
#define WR5_INP_AXIS3_U                 (WR5_INP_AX30|WR5_INP_AX31)
// 보간 drive 의 무효 mode
```

```
#define WR5_INP_LCMODE_INVALIDATE    0x0000
// 2 축 선속 일정
#define WR5_INP_LCMODE_2AXIS         WR5_INP_LSPD0
// 3 축 선속 일정
#define WR5_INP_LCMODE_3AXIS         (WR5_INP_LSPD0|WR5_INP_LSPD1)
```

- WR6 (write data register 1)

```
#define WR6_OUT_WD0          0x0001  // 범용 출력 register
#define WR6_OUT_WD1          0x0002
#define WR6_OUT_WD2          0x0004
#define WR6_OUT_WD3          0x0008
#define WR6_OUT_WD4          0x0010
#define WR6_OUT_WD5          0x0020
#define WR6_OUT_WD6          0x0040
#define WR6_OUT_WD7          0x0080
#define WR6_OUT_WD8          0x0100
#define WR6_OUT_WD9          0x0200
#define WR6_OUT_WD10         0x0400
#define WR6_OUT_WD11         0x0800
#define WR6_OUT_WD12         0x1000
#define WR6_OUT_WD13         0x2000
#define WR6_OUT_WD14         0x4000
#define WR6_OUT_WD15         0x8000
```

- WR7 (write data register 2)

```
#define WR7_OUT_WD0          0x0001  // 범용 출력 register
#define WR7_OUT_WD1          0x0002
#define WR7_OUT_WD2          0x0004
#define WR7_OUT_WD3          0x0008
#define WR7_OUT_WD4          0x0010
#define WR7_OUT_WD5          0x0020
#define WR7_OUT_WD6          0x0040
#define WR7_OUT_WD7          0x0080
#define WR7_OUT_WD8          0x0100
#define WR7_OUT_WD9          0x0200
#define WR7_OUT_WD10         0x0400
#define WR7_OUT_WD11         0x0800
#define WR7_OUT_WD12         0x1000
#define WR7_OUT_WD13         0x2000
#define WR7_OUT_WD14         0x4000
#define WR7_OUT_WD15         0x8000
#define WR6_OUT_WD16         WR7_OUT_WD0  // 범용 출력 register
```

```
#define WR6_OUT_WD17        WR7_OUT_WD1
#define WR6_OUT_WD18        WR7_OUT_WD2
#define WR6_OUT_WD19        WR7_OUT_WD3
#define WR6_OUT_WD20        WR7_OUT_WD4
#define WR6_OUT_WD21        WR7_OUT_WD5
#define WR6_OUT_WD22        WR7_OUT_WD6
#define WR6_OUT_WD23        WR7_OUT_WD7
#define WR6_OUT_WD24        WR7_OUT_WD8
#define WR6_OUT_WD25        WR7_OUT_WD9
#define WR6_OUT_WD26        WR7_OUT_WD10
#define WR6_OUT_WD27        WR7_OUT_WD11
#define WR6_OUT_WD28        WR7_OUT_WD12
#define WR6_OUT_WD29        WR7_OUT_WD13
#define WR6_OUT_WD30        WR7_OUT_WD14
#define WR6_OUT_WD31        WR7_OUT_WD15
```

(12) 읽기 레지스터 비트 정의

- RR0 (주 status register)

```
// X 축 drive 의 상태 표시(1: drive pulse 출력중, 0: drive 종료중)
#define RR0_X_DRV           0x0001
#define RR0_Y_DRV           0x0002  // Y
#define RR0_Z_DRV           0x0004  // Z
#define RR0_U_DRV           0x0008  // U
// X 축 error 발생상태 표시(1: RR1, RR2 register 의 error bit 들중 1 개만 설정되도 설정)
#define RR0_X_ERR           0x0010
#define RR0_Y_ERR           0x0020  // Y
#define RR0_Z_ERR           0x0040  // Z
#define RR0_U_ERR           0x0080  // U
// 보간 drive 상태 표시(1: 보간 drive pulse 출력중)
#define RR0_I_DRV           0x0100
// (1: 연속보간 drive 에서 다음 node 를 위해 parameter data/보간 명령을 기입가능)
#define RR0_CNEXT           0x0200
// 원호보간 drive 에 있어서 현재 drive 의 상한을 가르킵니다.
#define RR0_ZONE0           0x0400
// E2E1E0 000:0, 001:1, 010:2, 011:3, 100:4, 101:5, 110:6, 111:7
#define RR0_ZONE1           0x0800
#define RR0_ZONE2           0x1000
// bit pattern 보간 drive 에서 stack counter(SC)의 값을 가르킵니다.
#define RR0_BPSC0           0x2000
// C1C0 00:0, 01:1, 10:2, 11:3
#define RR0_BPSC1           0x4000

RR1 (status register 1)
// (1: 논리/실제위치 counter≥COMP+ register, 0:else)
#define RR1_CMPP            0x0001
// (1: 논리/실제위치 counter<COMP- register, 0:else)
#define RR1_COMM            0x0002
```

```
#define RR1_ASND                0x0004  // (1: 가감속 drive 에서 가속때)
#define RR1_CNST                0x0008  // (1: 가감속 drive 에서 정속때)
#define RR1_DSND                0x0010  // (1: 가감속 drive 에서 감속때)
// (1:S 자 가감속 drive 에서 가속도/감속도가 증가할때)
#define RR1_AASND               0x0020
// (1:S 자 가감속 drive 에서 가속도/감속도가 일정할때)
#define RR1_ACNST               0x0040
// (1:S 자 가감속 drive 에서 가속도/감속도가 감소할때)
#define RR1_ADSND               0x0080
// (1:drive 가 외부감속정지신호 nIN0 에의해서 정지하였을때)
#define RR1_IN0                 0x0100
#define RR1_IN1                 0x0200  // nIN1
#define RR1_IN2                 0x0400  // nIN2
#define RR1_IN3                 0x0800  // nIN3
// (1:drive 가 +방향 limit 신호(nLMTP)에 의해서 정지할때)
#define RR1_LMTP                0x1000
// (1:drive 가 -방향 limit 신호(nLMTM)에 의해서 정지할때)
#define RR1_LMTM                0x2000
// (1: servo motor 용 alarm 신호 (nALARM)에 의해 정지할때)
#define RR1_ALARM               0x4000
// (1: 긴급정지신호(EMGN)에 의해서 정지할때)
#define RR1_EMG                 0x8000
```

- RR2 (status register 2)

```
// (1: COMP+ register 의 값보다 커졌을때)
#define RR2_SLMTP               0x0001
// (1: COMP- register 의 값보다 커졌을때)
#define RR2_SLMTM               0x0002
// (1: +방향 limit 신호(nLMTP)가 active level 일때)
#define RR2_HLMTP               0x0004
// (1: -방향 limit 신호(nLMTM)가 active level 일때)
#define RR2_HLMTM               0x0008
// (1: servo motor 용 alarm 신호(nALARM)가 유효설정에서 active level 로 되었을때)
#define RR2_ALARM               0x0010
// (1: 긴급 정지신호(EMGN)가 low level 로 되어있을때)
#define RR2_EMG                 0x0020
```

- RR3 (status register 3)

```
// (1: drive pulse 가 rising edge(정논리에서 pulse 가 올라가는 순간)일때)
#define RR3_PULSE               0x0001
// (1: 논리/실위치≥COMP- register)
#define RR3_PGECM               0x0002
// (1: 논리/실위치<COMP- register)
#define RR3_PLCM                0x0004
// (1: 논리/실위치<COMP+ register)
#define RR3_PLCP                0x0008
```



```
// (1:  논리/실위치≥COMP+ register)
#define RR3_PGECP                0x0010
// (1:가감속  drive 일때  정속지역으로  pulse 출력을  종료)
#define RR3_CEND                 0x0020
// (1:가감속  drive 일때  정속지역으로  pulse 출력을  시작)
#define RR3_CSTA                 0x0040
// (1:drive 가  종료)
#define RR3_DEND                 0x0080
```

- RR4,5 (input register 1,2)

```
#define X_IN0                    0x0001  // X 축의  입력상태  표시
#define X_IN1                    0x0002  //
#define X_IN2                    0x0004  //
#define X_IN3                    0x0008  //
#define X_EXP                    0x0010  //
#define X_EXM                    0x0020  //
#define X_XINP                   0x0040  //
#define X_XALM                   0x0080  //
#define Y_IN0                    0x0100  // Y 축의  입력상태  표시
#define Y_IN1                    0x0200  //
#define Y_IN2                    0x0400  //
#define Y_IN3                    0x0800  //
#define Y_EXP                    0x1000  //
#define Y_EXM                    0x2000  //
#define Y_XINP                   0x4000  //
#define Y_XALM                   0x8000  //
#define Z_IN0                    0x0001  // Z 축의  입력상태  표시
#define Z_IN1                    0x0002  //
#define Z_IN2                    0x0004  //
#define Z_IN3                    0x0008  //
#define Z_EXP                    0x0010  //
#define Z_EXM                    0x0020  //
#define Z_XINP                   0x0040  //
#define Z_XALM                   0x0080  //
#define U_IN0                    0x0100  // U 축의  입력상태  표시
#define U_IN1                    0x0200  //
#define U_IN2                    0x0400  //
#define U_IN3                    0x0800  //
#define U_EXP                    0x1000  //
#define U_EXM                    0x2000  //
#define U_XINP                   0x4000  //
#define U_XALM                   0x8000  //
#define n_IN0                    X_IN0   // 축의  입력상태  표시
#define n_IN1                    X_IN1   //
#define n_IN2                    X_IN2   //
#define n_IN3                    X_IN3   //
#define n_EXPP                   X_EXP   //
#define n_EXPM                   X_EXM   //
#define n_INPOS                  X_XINP  //
#define n_ALARM                  X_XALM//
```

(13) PMC4BPCI 함수 리턴 매크로

```
// 실패시 결과
#define MMC_FALSE                       0
// 성공시 결과
#define MMC_TRUE                        1
// 함수 수행결과 성공시
#define MMC_OK                          1
// 논리 level 이 high 일 때
#define MMC_HIGH_LEVEL                  1
// 논리 level 이 low 일 때
#define MMC_LOW_LEVEL                   0
// MMC 드라이버의 호출 실패
#define MMC_OPEN_ERR                    5
// MMC 드라이버의 I/O address 의 범위가 넘었을 때
#define MMC_IOADDRESS_ERR               6
// MMC 의 drive 중 기다림 시간을 초과할 때
#define MMC_TIMEOUT_ERR                 7
// 축 지정을 잘못한 경우
#define MMC_INVALID_AXIS                8
// 파라미터 설정을 잘못한 경우
#define MMC_ILLEGAL_PARAMETER           9
// 움직임을 가질 수 없는 이동을 명령하였을 경우
#define MMC_ZERO_PARAMETER              10
// 다른 원인으로 인한 에러일 때
#define MMC_ERROR                       11
// 종료하는 이벤트 발생시
#define MMC_QUIT                        12
// CARD 인식 불가
#define MMC_INVALID_CARD                13
```

(14) PMC4BPCI 라이브러리 선택

표준 PMC4BPCI 라이브러리와 호환되는 라이브러리를 응용프로그램에 적재하기 위해 아래와 같이 매크로 문을 정의 합니다.

아래의 예는 PMC4BPCI 를 사용하기 위한 정의입니다.

```
#define MMC_USING_PMC4BPCI_DRIVER
#include "pmc4bpci.h"
```

(15) 축수를 나타내는 매크로 정의

```
// PMC4BPCI.H 에 포함
#define PMC4BPCI_AXIS_NUM       4

// MMC_AX.H 에 포함
#define AXIS_NUM                        PMC4BPCI_AXIS_NUM
```


(16) NOVA 호환용 정의

```
// PMC-4B-PCI 보드의 Base address
// PMC4BPCI.H 에 포함
// PMC4BPCI 라이브러리 내부에서는 적용되지 않는다.
#define   adr                    0x0320

// 축지정 매크로
// PMC4BPCI.H 에 포함
#define AXIS_X                   0x01
#define AXIS_Y                   0x02
#define AXIS_Z                   0x04
#define AXIS_U                   0x08
```

ISO-9001

Autonics
Sensors & Controllers

www.autonics.co.kr

Distributor

■ 주요생산품목

근접센서 · 포토센서 · 에리어센서 · 광화이버센서 · 도어센서 · 도어사이드센서 · 압력센서 · 로터리 엔코더 · 카운터 · 타이머 · 온도조절기 · 온/습도 센서 · 전력조정기 · 판넬메타 · 타코/스피드/펄스메타 · 디스플레이 유니트 · 센서 컨트롤러 · 스위칭 파워 서플라이 · 그래픽/로직 패널 · 스테핑 모터/드라이버/컨트롤러 · 필드 네트워크 기기 · 레이저 마킹 시스템(CO_2, Nd:YAG) · 레이저 웰딩/솔더링 시스템

Autonics Corporation

■ 본사(공장) / 부산사무소
TEL : 055-371-5051 / FAX : 055-372-4432
■ 서울사무소
TEL : 032-610-2700 / FAX : 032-323-3008
■ 대구사무소
TEL : 053-383-7673 / FAX : 053-383-7674
■ 광주
TEL : 062-521-6716~7, 010-9277-3023 / FAX : 062-521-6717
■ 기술 상담 센터
제품기술상담 : 1588-2333 (전국)
A / S 상 담 : 080-529-3333 (수도권/충청/강원)
080-519-3333 (영남/호남/제주)

■ 제품 개선 / 개발 제안 : Product@autonics.com

MTC-PMC4BL1-V1.0-1212KR