cEIP REV04
cEIP User Manual 2010/04/21

Firmware Development Guide for COMIZOA Ethernet/IP Modules

Table of Contents

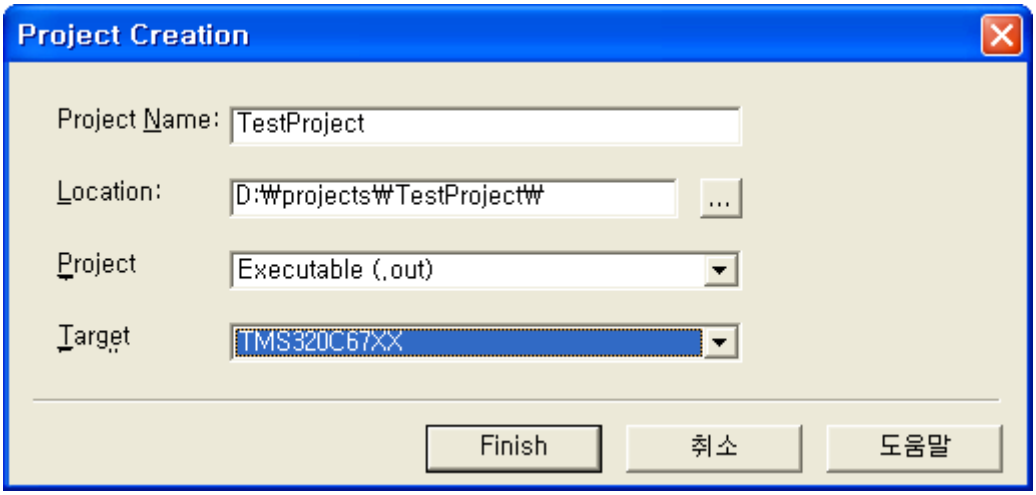
CHAPTER 1. CCS3.3 에서 개발 시작하기

1.1 새 프로젝트 만들기

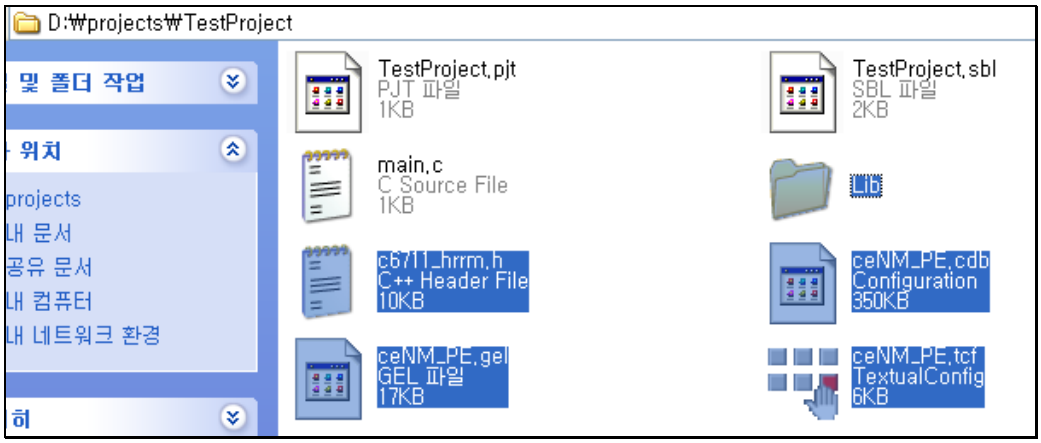
TMS320C6711 DSP 보드와 기본적으로 ㈜커미조아에서 제공하는 MK 라이브러리를 사용하여 사용자 펌웨어 프로그램을 개발하시려면 아래와 같은 순서대로 작업을 진행하시면 됩니다.

1. Project – New 메뉴를 선택 합니다.

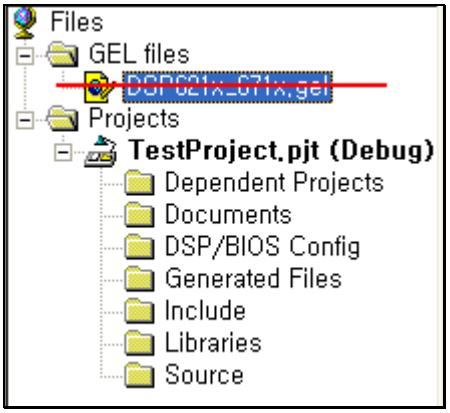
Project Name, Location 및 Target 보드를 선택하신 후 Finish 버튼을 눌러 프로젝트를 생성합니다.



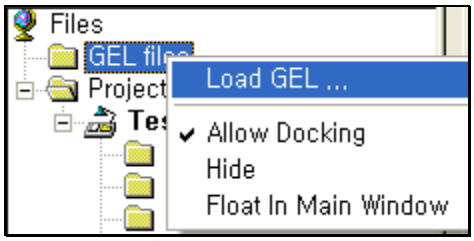
2. 생성된 프로젝트 폴더 안에 아래 그림처럼 필수 파일들을 복사해 넣습니다.



3. 디폴트 .gel 파일인 DSP621x_671x.gel 을 키보드의 <Delete> 버튼을 사용해서 GEL files 항목에서 제거 합니다.



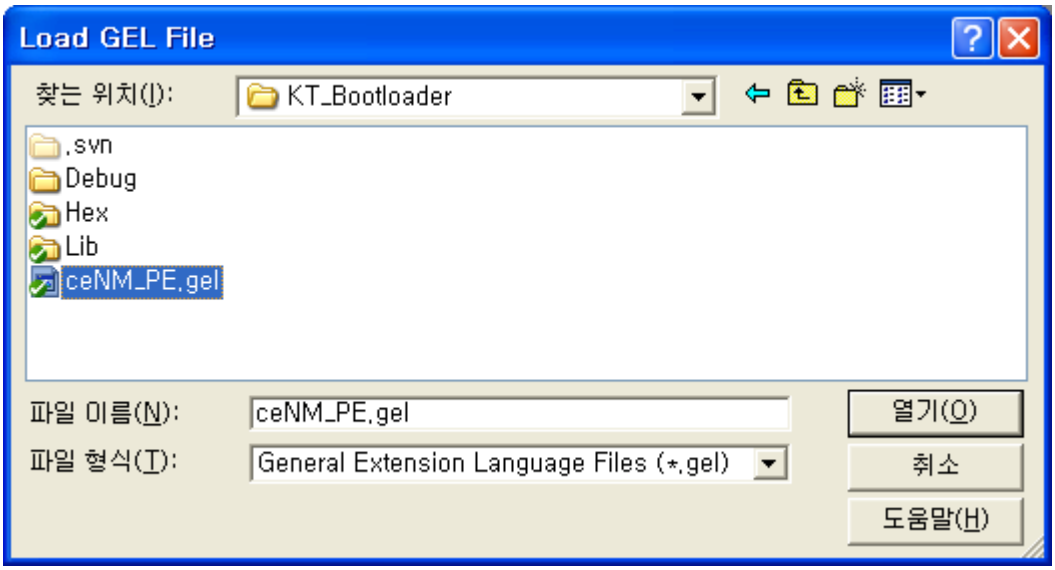
4. GEL files 항목에서 마우스 우측버튼을 클릭 후 "Load GEL …" 메뉴를 선택합니다.



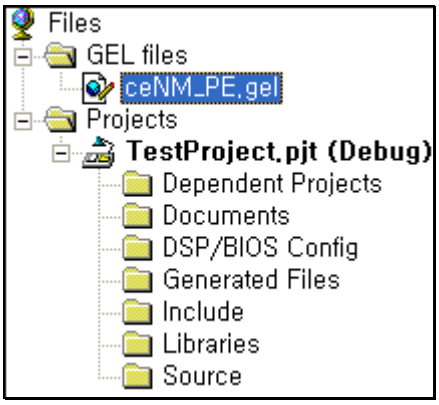
5. ㈜커미조아에서 배포하는 TMS320C6711 용 ceNM_PE.gel 파일을 선택합니다.

이를 위해서는 사전에 D:\projects\TestProject\ceNM_PE.gel 경로에 복사해 넣으셔야 합니다.

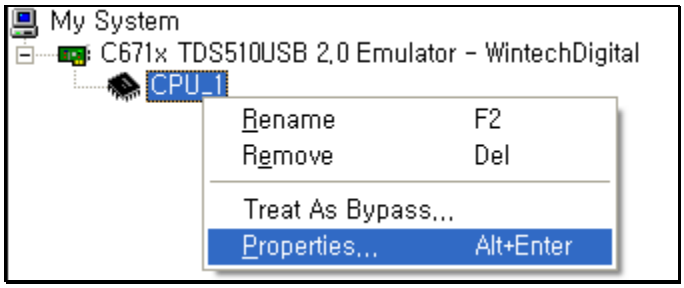
㈜커미조아에서는 JTAG 를 사용한 디버깅 개발 환경에서 TMS320C6711 모듈에 대한 SDRAM 메모리 인터페이스 초기화 함수들을 별도로 제작한 GEL 파일을 통해 제공해 드리고 있습니다. GEL 파일에 관한 일반적인 내용은 CCS3.3 의 도움말 파일을 참고하시기 바랍니다.

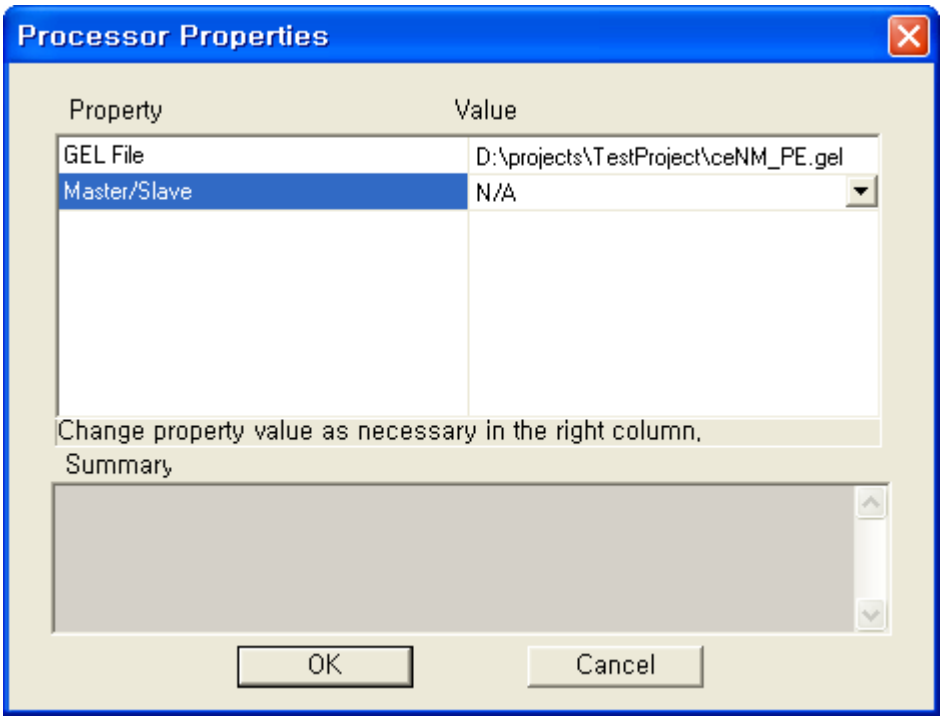


6. 아래 그림 처럼 ceNM_PE.gel 파일 하나만 GEL files 리스트에 들어가 있어야 합니다.

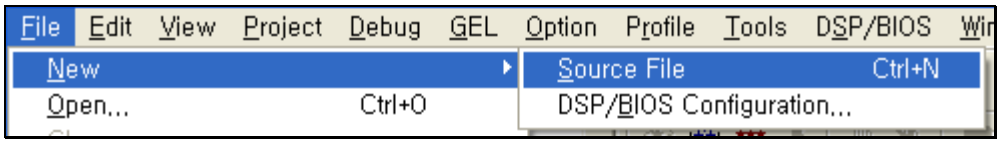


[팁] .gel 파일을 추가시키는 다른 방법으로는 "Setup CCStudio v3.3" 프로그램을 수행 시켜 아래 그림과 같이 GEL 파일 경로를 셋업해주시면 디폴트 GEL 파일로 자동으로 설정됩니다. 이렇게 한번 설정을 해놓고 나면 이후부터는 위의 3 ~ 6번 작업을 하지 않으셔도 됩니다.



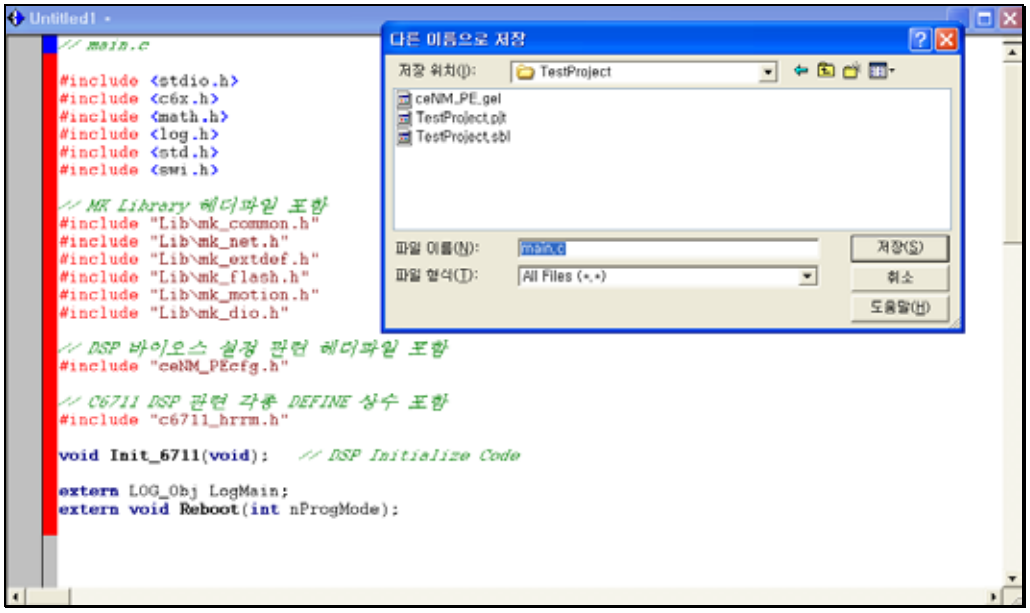


7. 프로그램 Entry Point 인 main.c 파일을 새로 작성합니다.

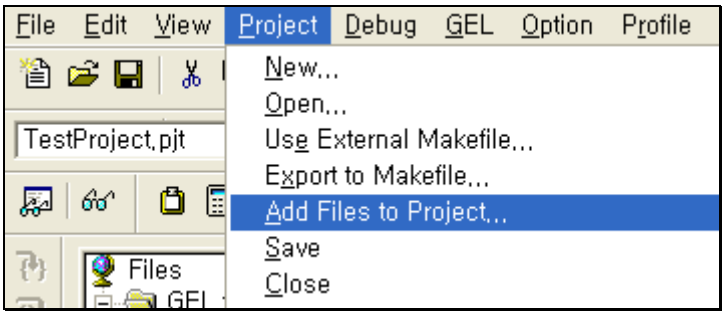


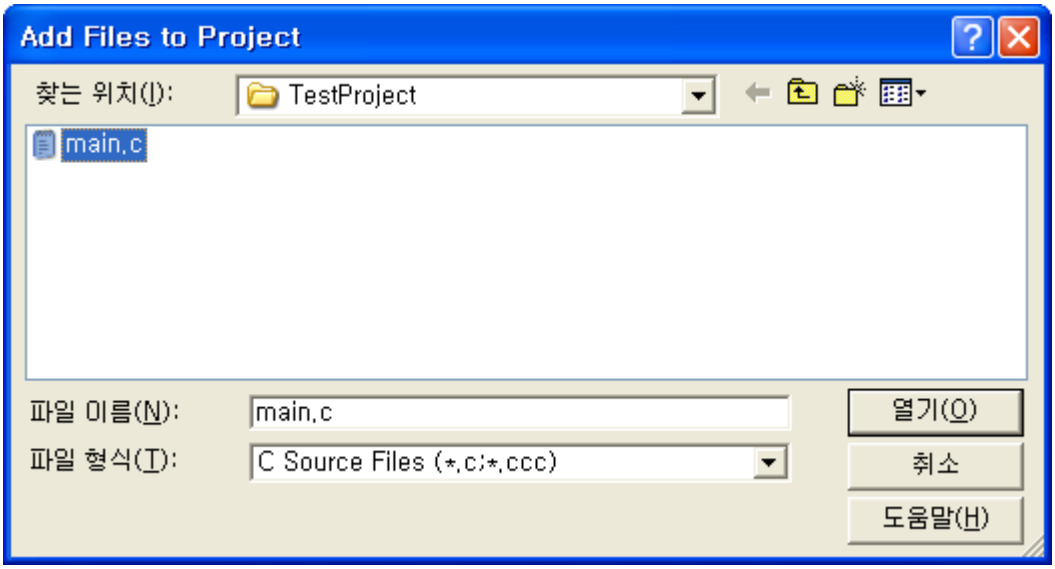
8. 필요한 코드들, 나중에 프로젝트에서 사용하게 될 함수나 변수, 정의된 상수값을 포함하는 각종 헤더파일(.h) 을 #include 문을 사용해서 추가합니다. 또한 main 함수에서 사용할 함수의 원형을 정의합니다.

작성이 끝나면 main.c 파일로 저장합니다.



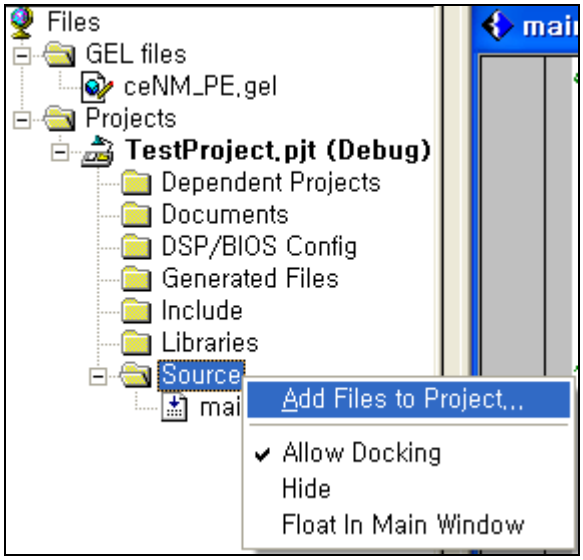
9. 작성해서 저장시킨 main.c 파일을 프로젝트에 추가시킵니다.

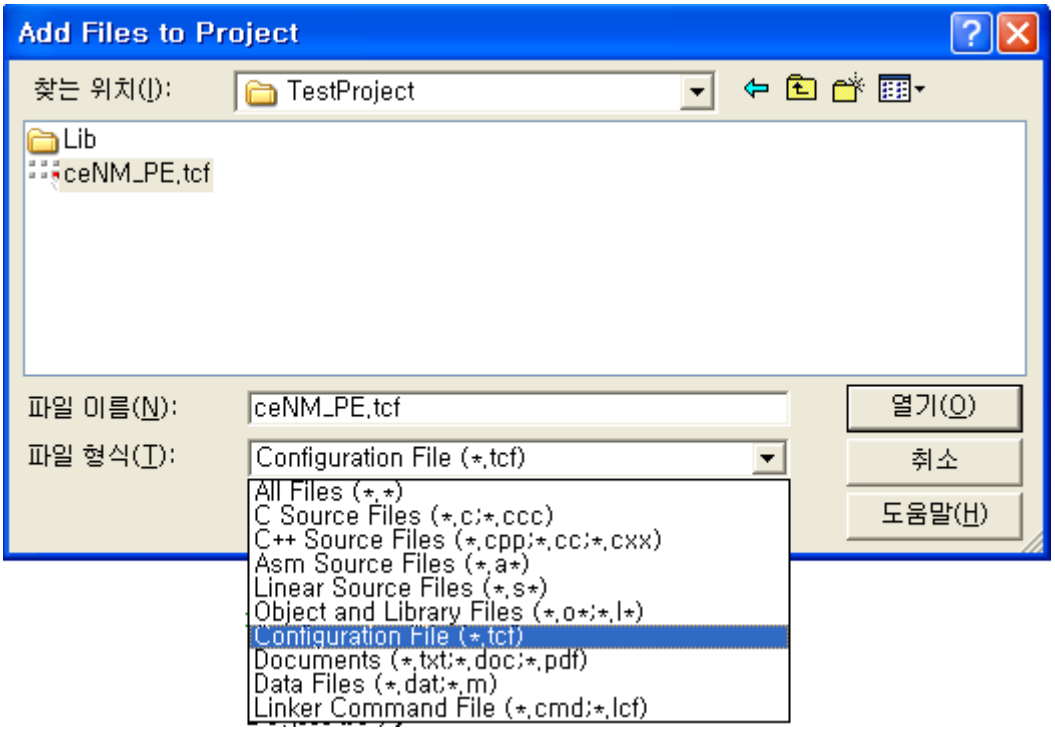




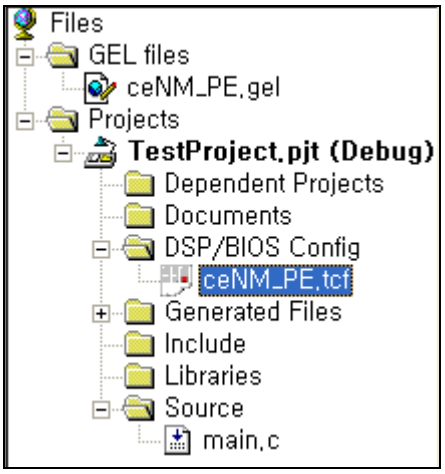
10. 프로젝트에 소스파일들을 추가합니다.

먼저 DSP 바이오스 환경 설정 파일을 추가합니다. 위의 2번 항목에서 이미 ceNM_PE.cdb 파일과 ceNM_PE.tcf 파일을 프로젝트 폴더에 추가하였고 여기서는 .tcf 파일만 선택해주시면 됩니다.

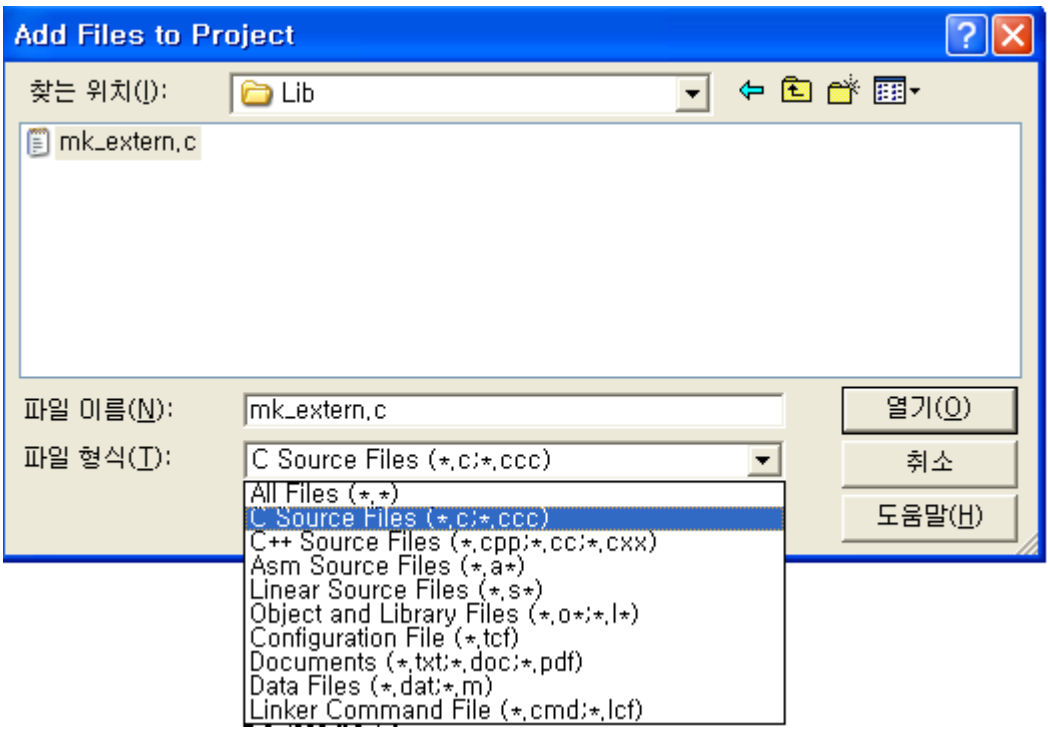




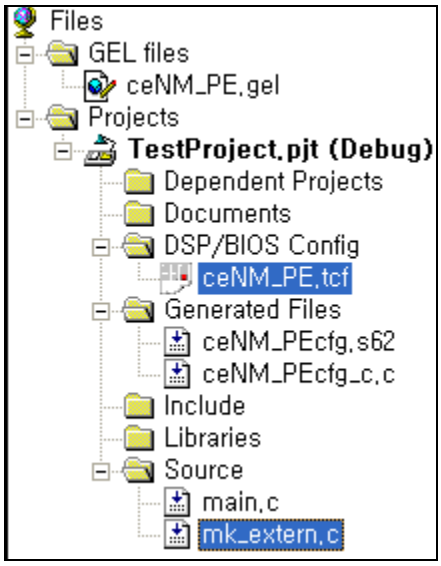
DSP/BIOS Config 항목에 ceNM_PE.tcf 파일이 추가된 것을 확인합니다.



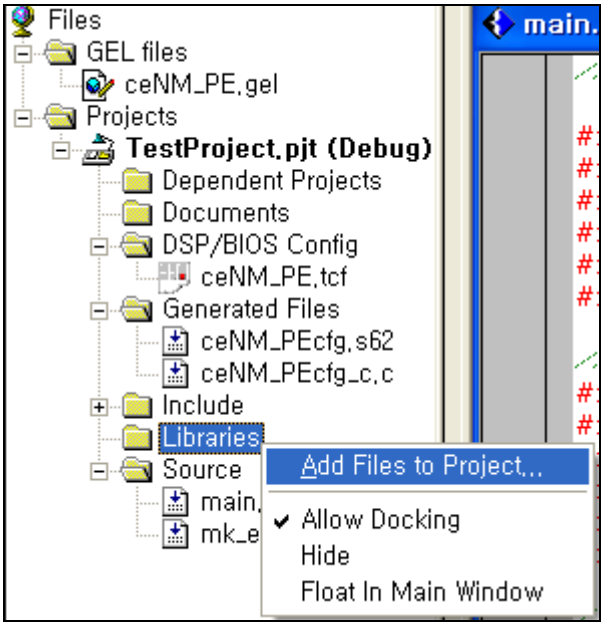
다음으로 .c 형태인 MK 라이브러리 파일, mk_extern.c 파일을 프로젝트에 추가합니다.

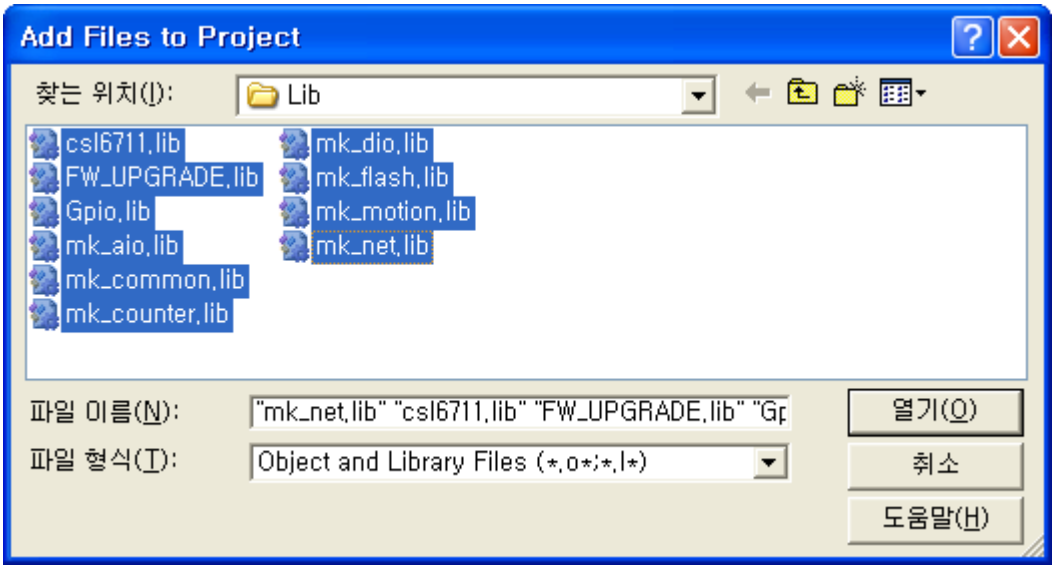


추가된 소스파일을 확인합니다.

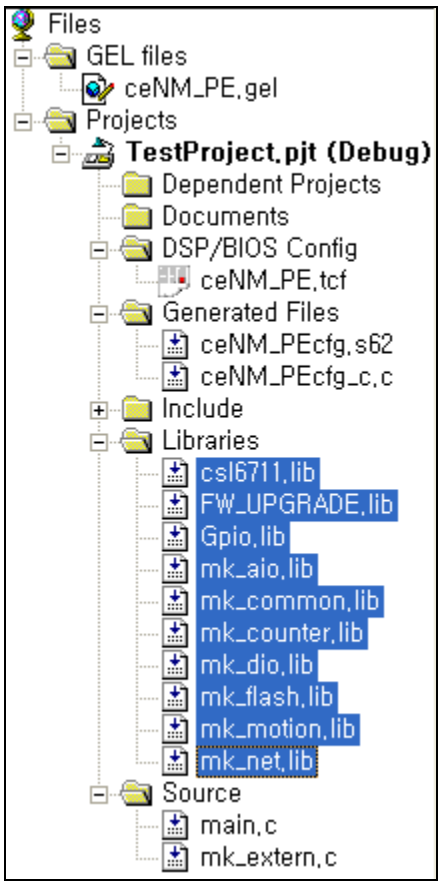


11. 프로젝트에 라이브러리 파일을 추가합니다.





프로젝트에 추가된 라이브러리를 확인합니다.



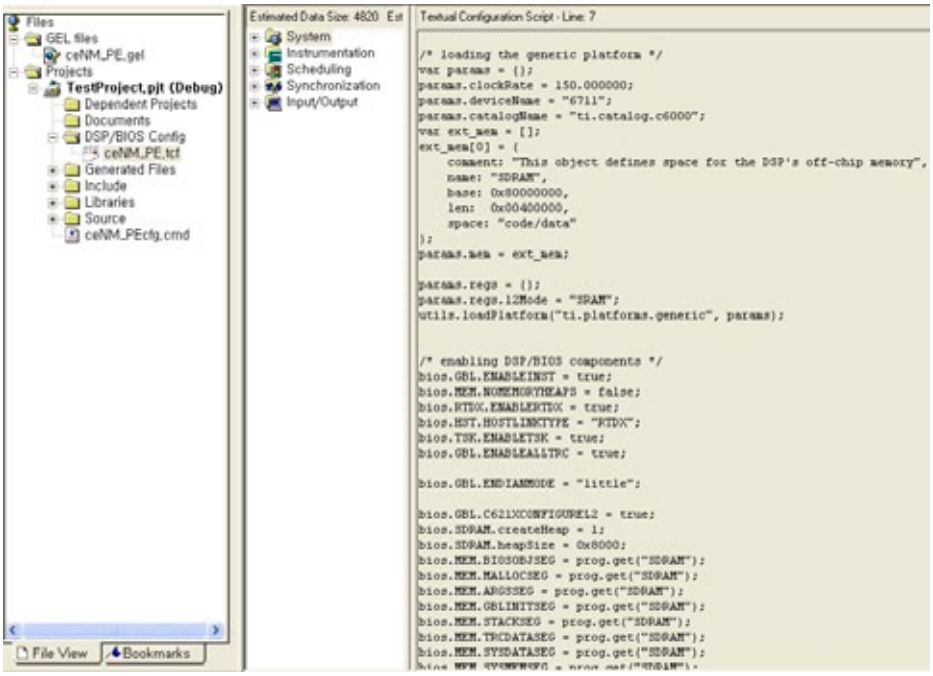
1.2 DSP/BIOS 설정

DSP/BIOS Config 설정을 통해서 시스템 메모리, 외부 메모리, 플래쉬 메모리, 타이머 함수, 인터럽트 서비스루틴, 작업함수(TASK) 등을 설정, 사용을 하실 수 있습니다.

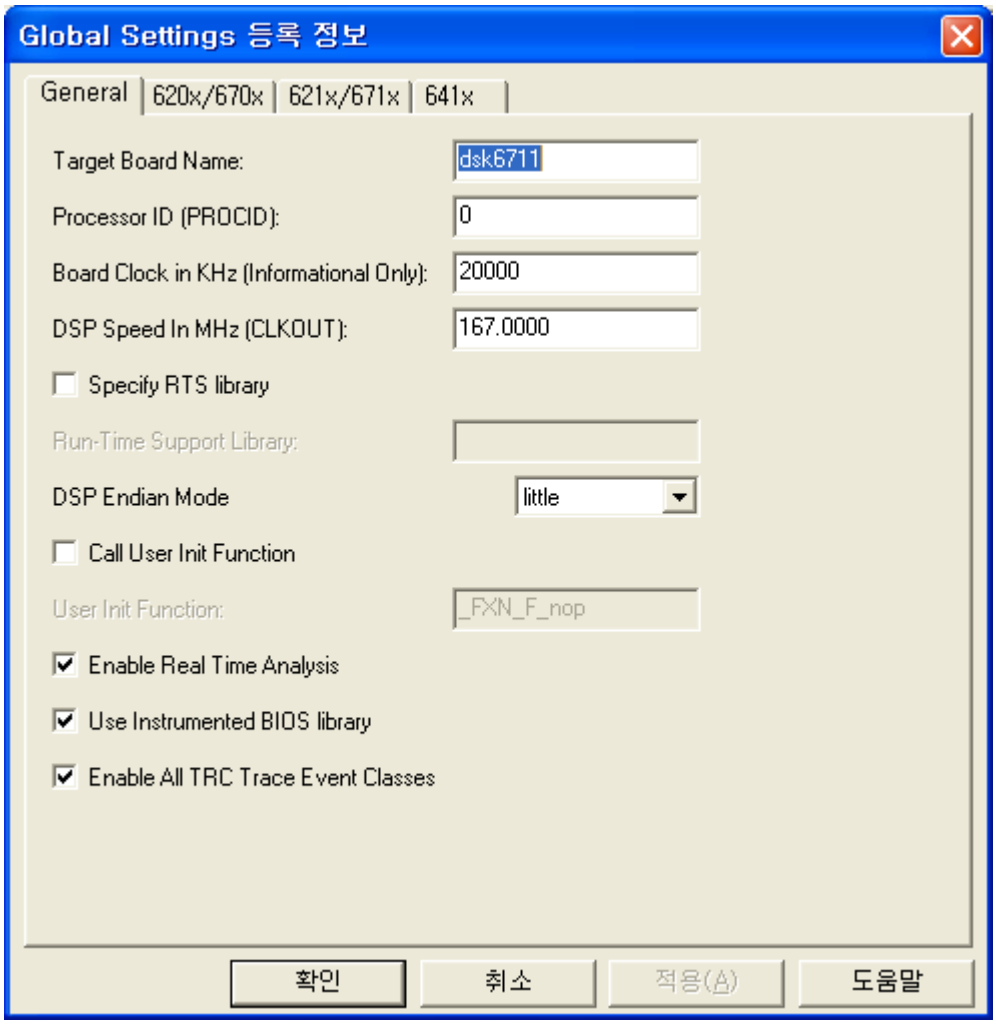
위 절에서 프로젝트에 필요한 파일들은 대부분 생성 및 추가되었으나 아직 프로젝트를 빌드 하시면 안됩니다. 빌드 시 오류가 발생합니다. 아래의 순서대로 DSP/BIOS 설정을 통해 메모리 섹션 구성 및 각종 설정을 하신 뒤에 메뉴에서 "Save" 버튼을 클릭하시면 ceNM_PEcfg.cmd 및 ceNM_PEcfg.h 파일이 생성되며 이 파일들을 프로젝트에 포함시켜야 합니다.

[설명] .cmd 파일은 소스파일 추가로, .h 파일은 main.c 에 #include 문을 통해 프로젝트에 포함시키시면 됩니다.

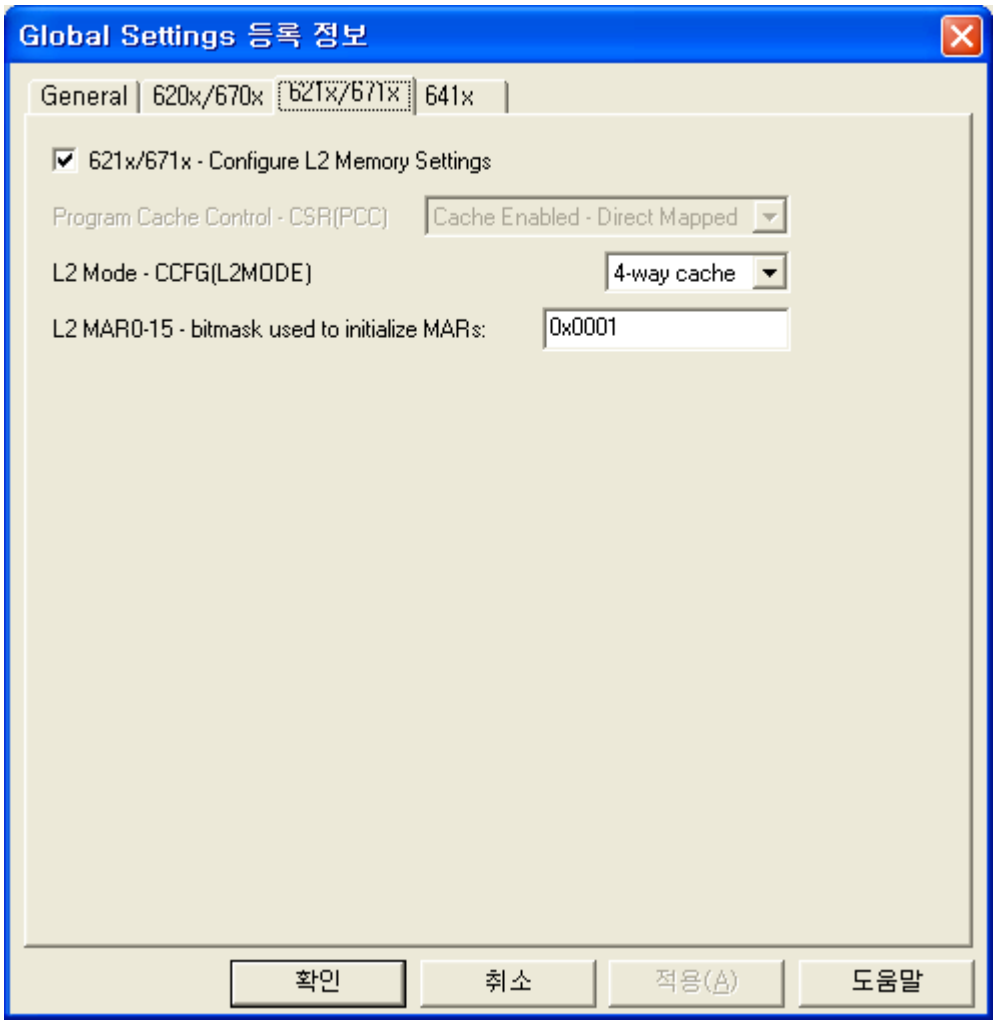
1. System – Global Settings 설정



Target 보드에 대한 일반적인 정보, CPU 클럭속도 등을 설정합니다.



내부 시스템 메모리를 모두 L2 Cache 메모리로 사용합니다. Cache 메모리를 활용하여 DSP 의 메모리 입출력 속도를 캐쉬 메모리를 통해 향상시키기 위한 설정입니다.

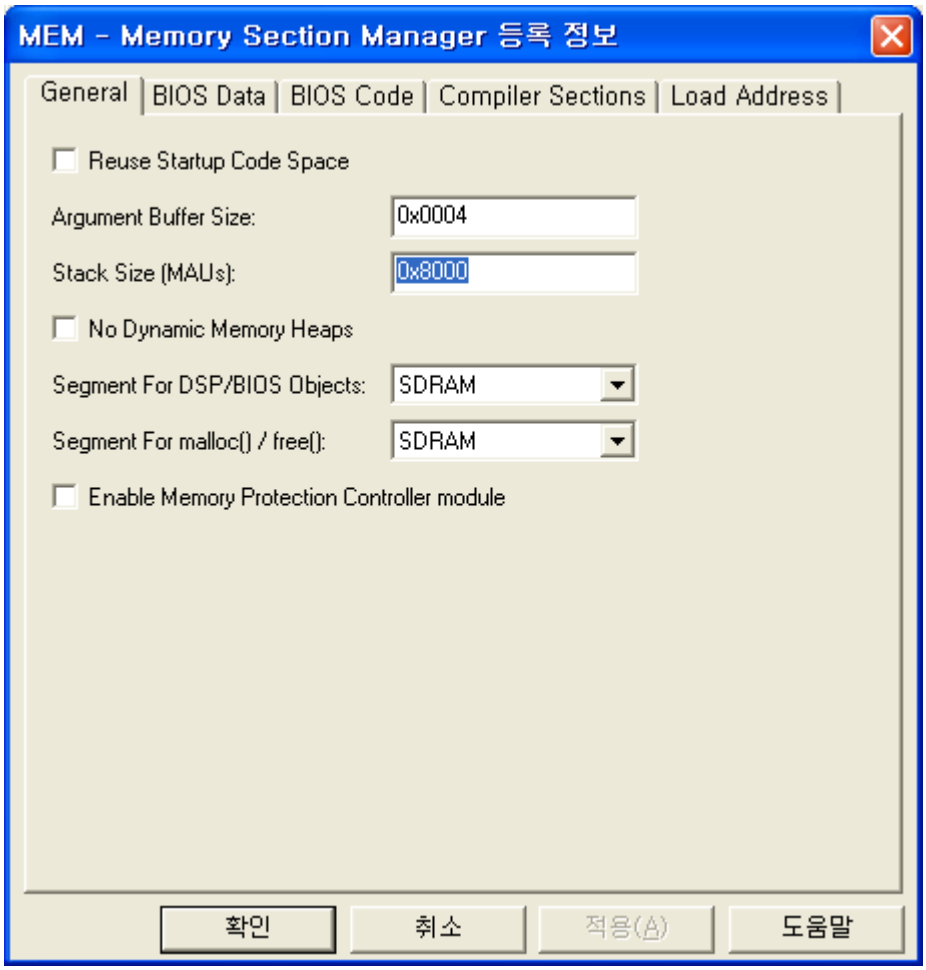


2. System - MEM – Memory Section Manager 설정.

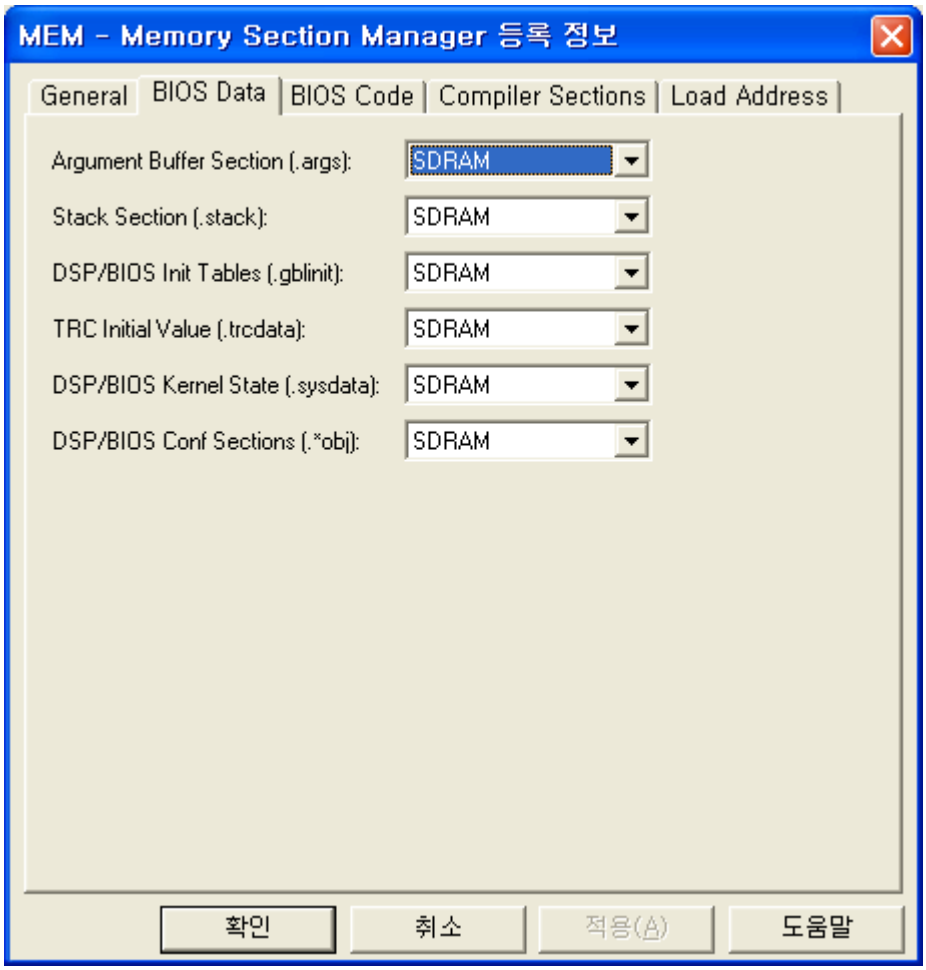
스택 메모리 크기 설정. 나머지 탭들에 대한 설정은 아래 그림들 처럼 변경 없이 그대로 두시면 됩니다.

만일 Stack 크기가 너무 작을 경우는 프로그램 코드에서 로컬 변수나 함수 할당을 위한 메모리 공간이 부족하게 되어 코드가 오동작하는 경우도 있습니다. 따라서 Stack 크기는 너무 작게 잡아주면 안됩니다.

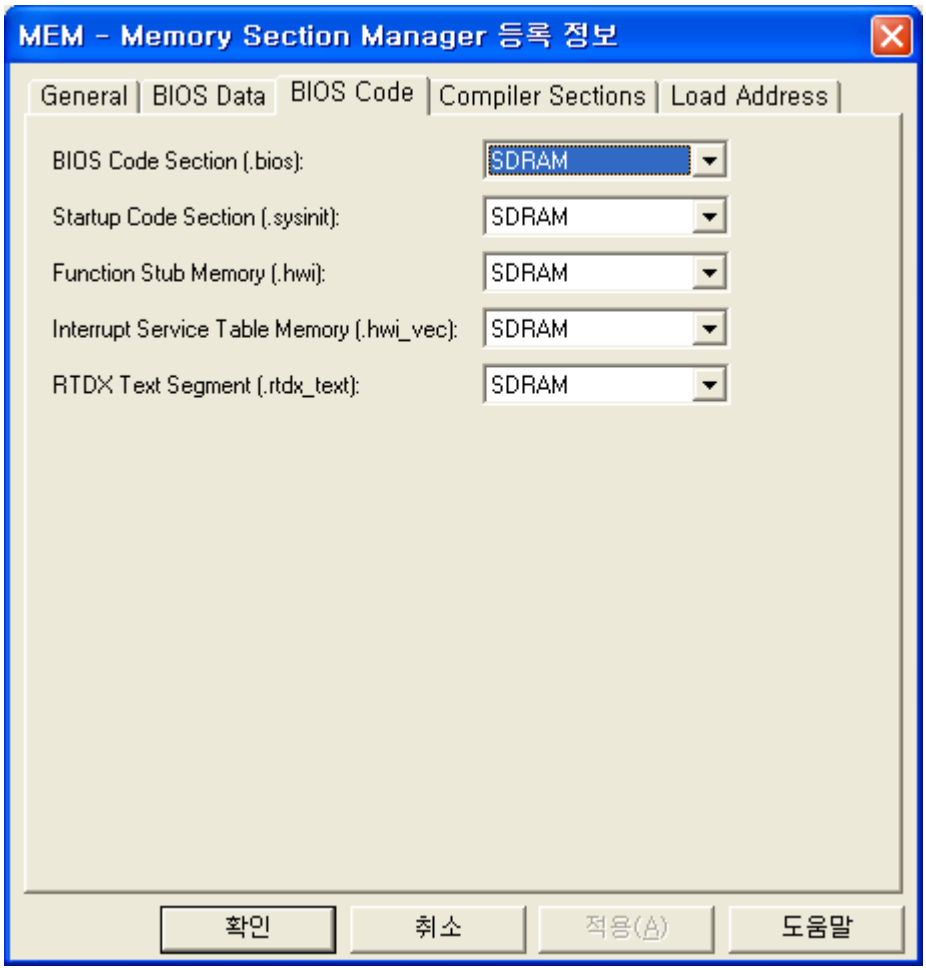
2.1 Memory Section Manager 등록정보 - General



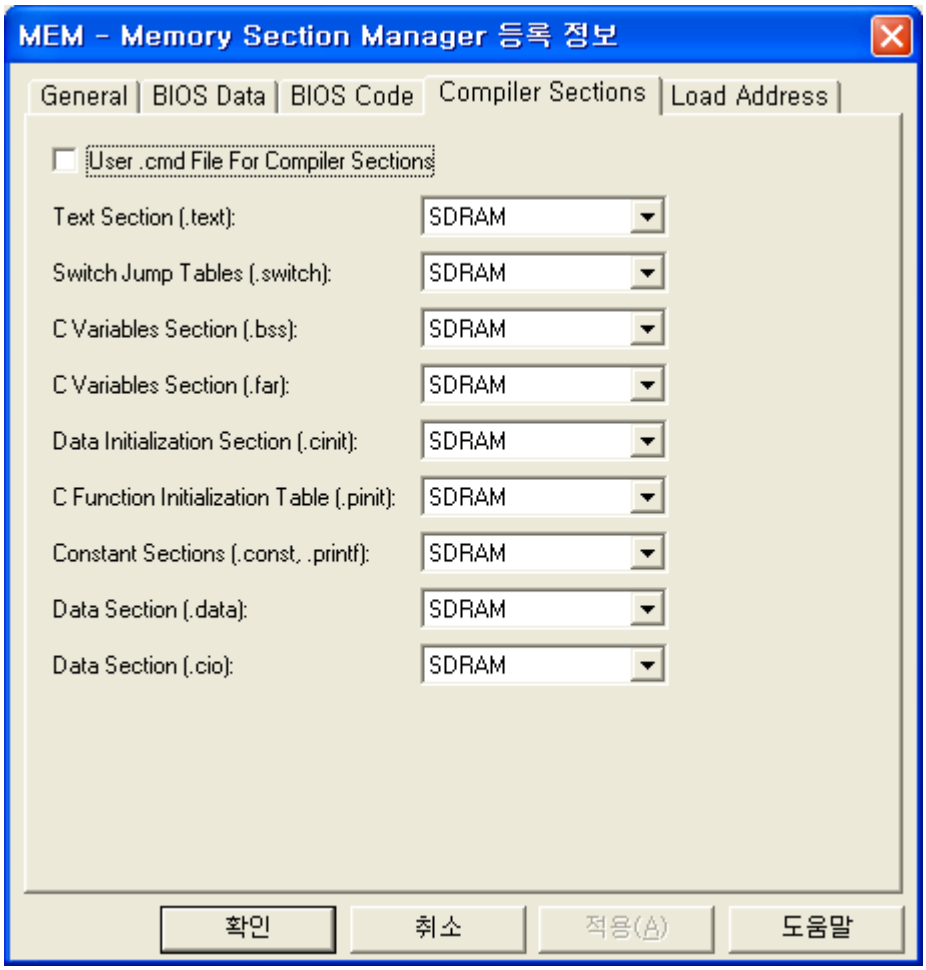
2.2 Memory Section Manager 등록정보 – BIOS Data



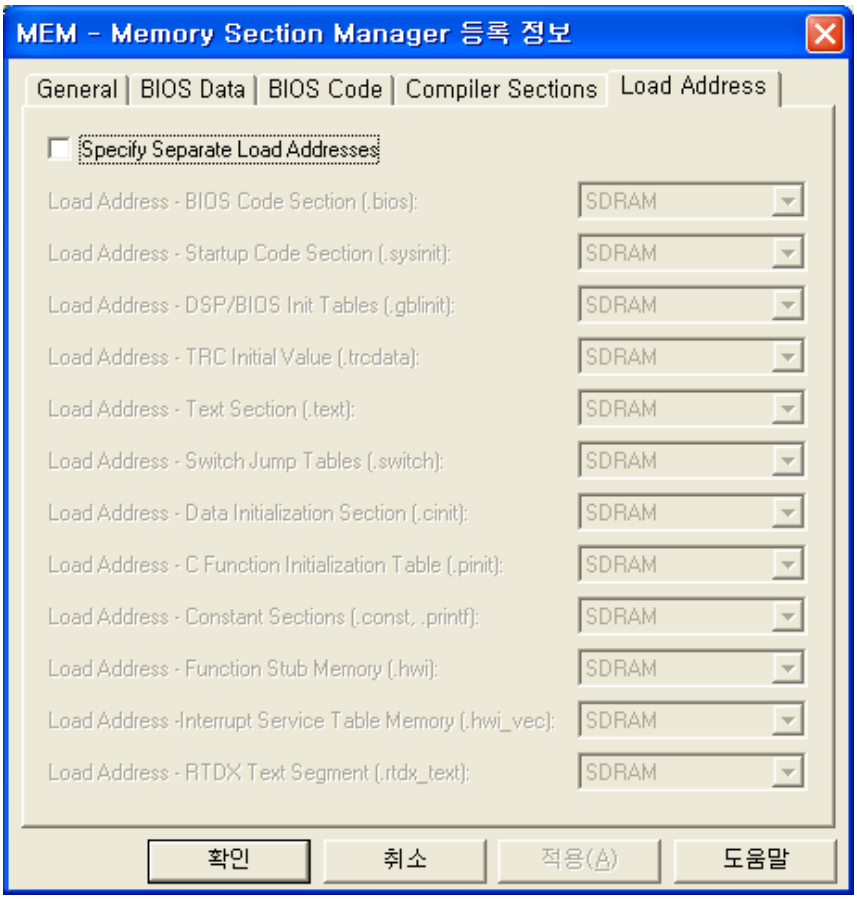
2.3 Memory Section Manager 등록정보 – BIOS Code



2.4 Memory Section Manager 등록정보 – Compiler Sections

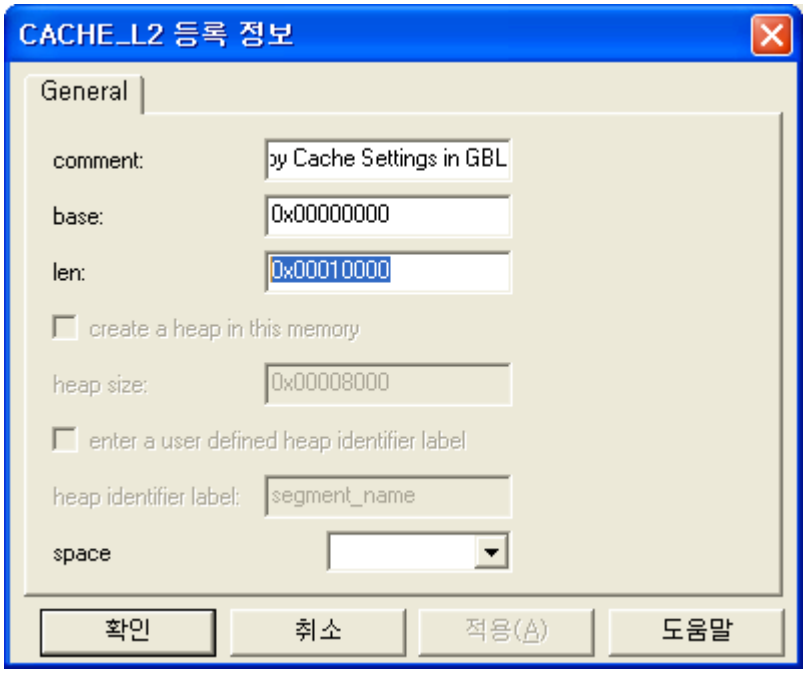


2.5 Memory Section Manager 등록정보 – Load Address



2. System - MEM – Memory Section Manager – CACHE_L2 메모리 크기 설정

L2 Cache 메모리 범위는 0x00000000 ~ 0x0000FFFF (length = 0x10000) 으로 설정하셔야 합니다.



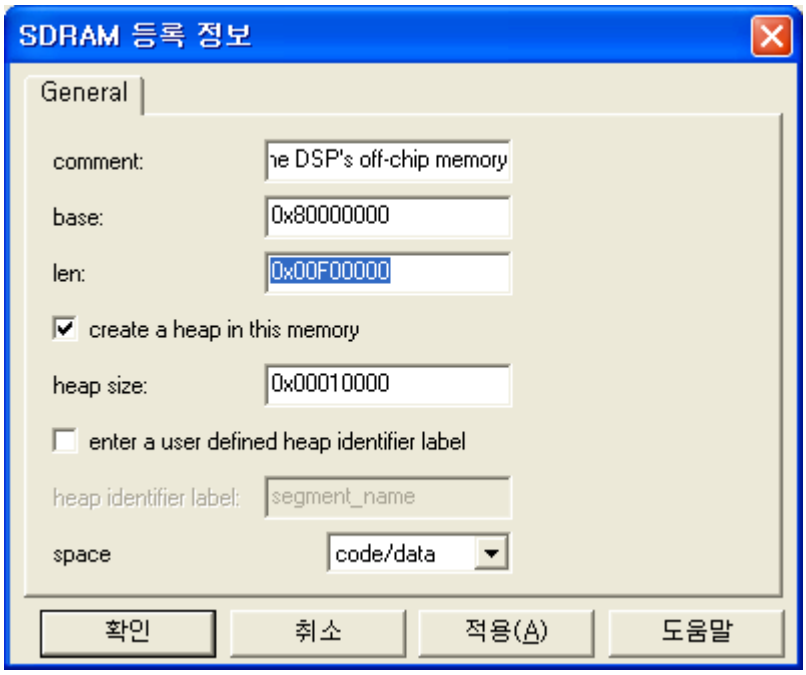
3. System - MEM – Memory Section Manager – SDRAM 메모리 설정

SDRAM 메모리는 전체 16MB (16 * 1024 * 1024 = 16777216 바이트) 크기로 Hexa Decimal 값으로 0x800000000 ~ 0x80FFFFFF (length = 0x01000000) 까지 사용 가능합니다. 하지만 유저 프로그램에서는 이 범위 중 0x800000000 ~ 0x80EFFFFF (length = 0x00F00000) 까지 15MB 크기만 사용 가능합니다.

[주의] 0x80F00000 ~ 0x80FFFFFF (length = 0x00100000) 까지 1MB 의 메모리 영역에 대해서는 부트로더 전용으로 사용되므로 이 영역을 유저 프로그램 영역으로 사용하실 수 없습니다. 만일 사용하시는 경우는 부트로더가 정상동작되지 않을 수 있습니다.

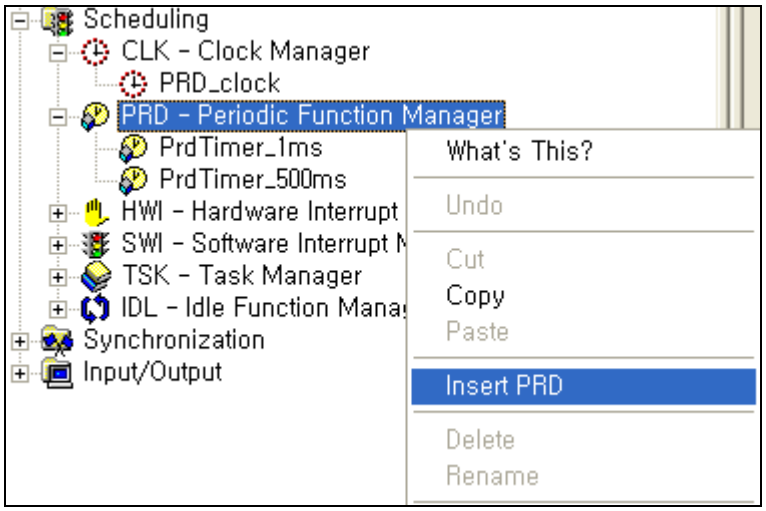
[참고] 플래쉬 메모리 설정 및 섹터 정보는 1.5절을 참고 하십시오.

아래 그림 처럼 SDRAM 메모리의 시작 주소와 크기를 설정 합니다. 또한 힙 메모리 크기를 설정하며 설정된 SDRAM 메모리 영역을 code 전용, data 전용 또는 code/data 겸용으로 사용할지를 선택합니다. 여기서는 code/data 겸용으로 설정합니다.



4. Scheduling – CLK – Clock Manager 설정

클럭 발생 주기를 설정합니다. 마이크로초 단위로 설정하며 이 값이 기준 클럭 주기가 됩니다. 여기서는 1000 마이크로초 이므로 1밀리초가 기준 클럭 주기가 됩니다.



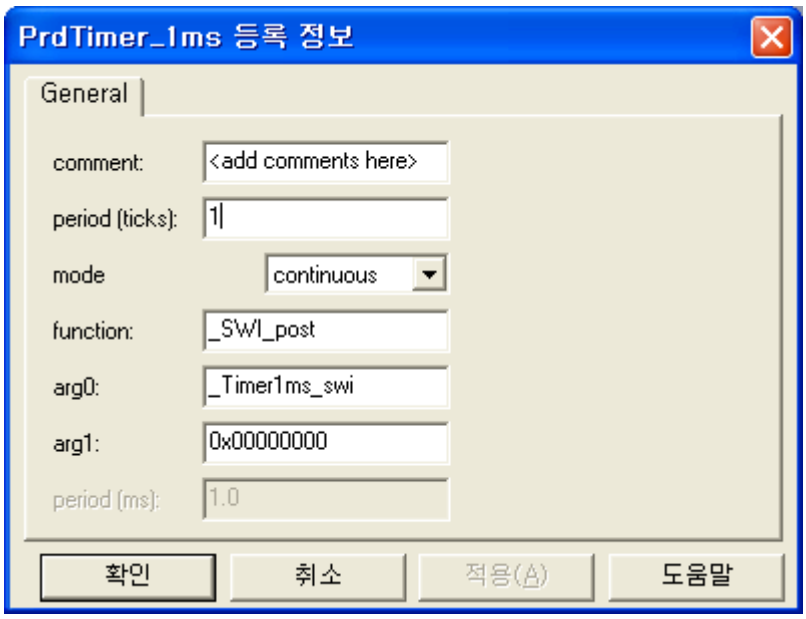
Scheduling – PRD – PrdTimer_1ms 의 Properties 창에서 다음과 같이 타이머 Tick 의 주기, period(ticks) 를 1 로 설정합니다. 위의 4번 항목에서 클럭 주기가 1ms 이므로 PrdTimer_1ms 타이머의 주기는 1 * 1ms 가 됩니다.

여기서 mode 는 단발성 타이머가 아닌 continuous 모드로 연속적으로 동작하도록 설정합니다.

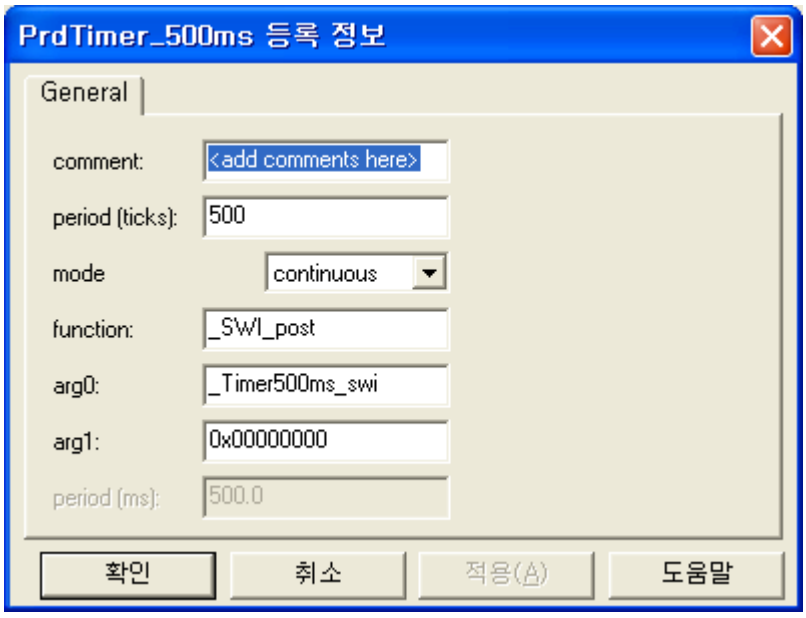
다음으로 function 란에 _SWI_post 라고 입력해 줍니다. 함수 이름 앞에 "_" (언더바) 가 붙은 이유는 어셈블리어 코드에서 실제 함수 이름 앞에 언더바를 사용하기 때문이라고만 알아 두시면 됩니다. SWI_post 함수는 DSP/BIOS 에서 지원하는 함수로 소프트웨어 인터럽트 타이머 함수를 설정된 타이머 주기 마다 호출해 주는 함수라고 이해하시면 됩니다. 이 함수 관련한 자세한 설명은 도움말을 참고 하시면 됩니다.

첫번째 argument, arg0 는 _Timer1ms_swi 라고 입력해 주시고 이 함수 이름은 다르게 입력하셔도 좋으나 아래에서 설명드리겠지만 SWI 항목의 function 이름에 들어갈 동일한 함수 이름으로 입력해주셔야 합니다. arg1 은 NULL 값 (0x0) 으로 두시면 됩니다.

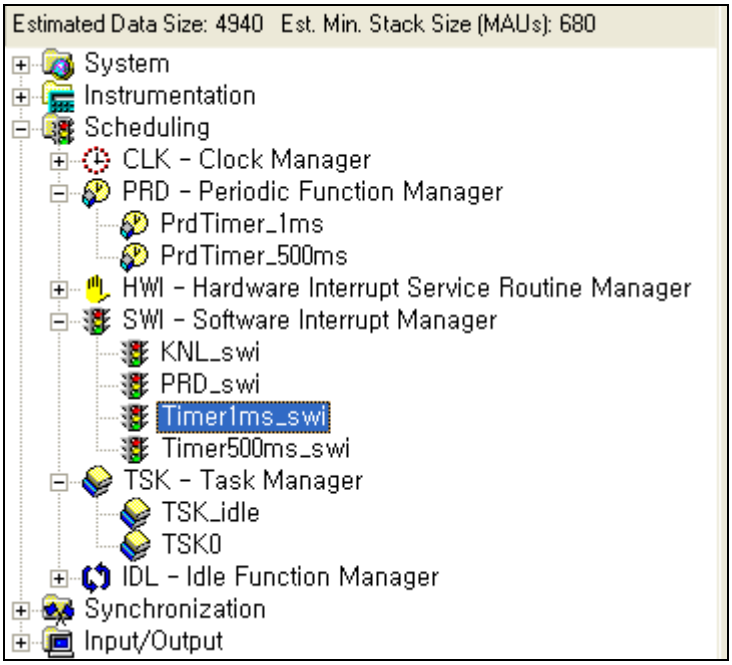
결과적으로 연속적으로 1ms 의 주기 마다 Timer1ms_swi 함수가 호출됩니다.

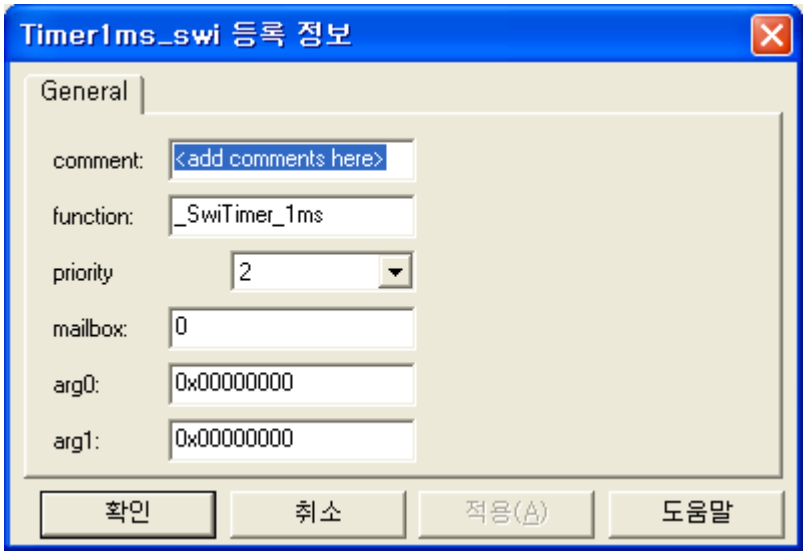


500ms 주기의 타이머도 동일한 방법대로 아래와 같이 추가시켜 주시면 됩니다.



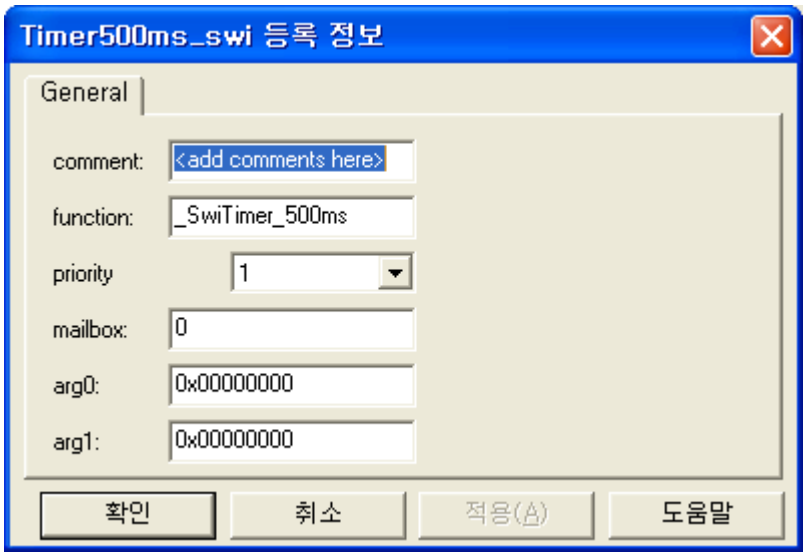
마지막으로 SWI 항목 아래에 해당 타이머 함수 인스턴스들을 추가시켜 주시면 됩니다.





function 항목은 실제 main.c 소스에서 구현할 함수 이름으로 "_" (언더바) 를 포함한 SwiTimer_1ms 로 입력하시고 priority 를 0 ~ 14 까지 설정하시면 됩니다. 숫자가 높을수록 함수 수행에 대한 우선 순위가 높아집니다. 나머지 argument 값들은 그대로 둡니다.

Timer500ms_swi 타이머 함수 인스턴스에 대해서도 아래 그림과 같이 main.c 에서 구현할 함수 이름을 입력 하고 우선순위를 1ms 주기 타이머 보다는 낮게 설정해 주시면 됩니다.

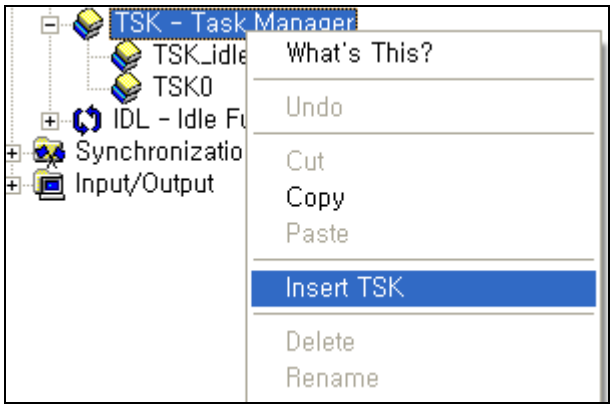


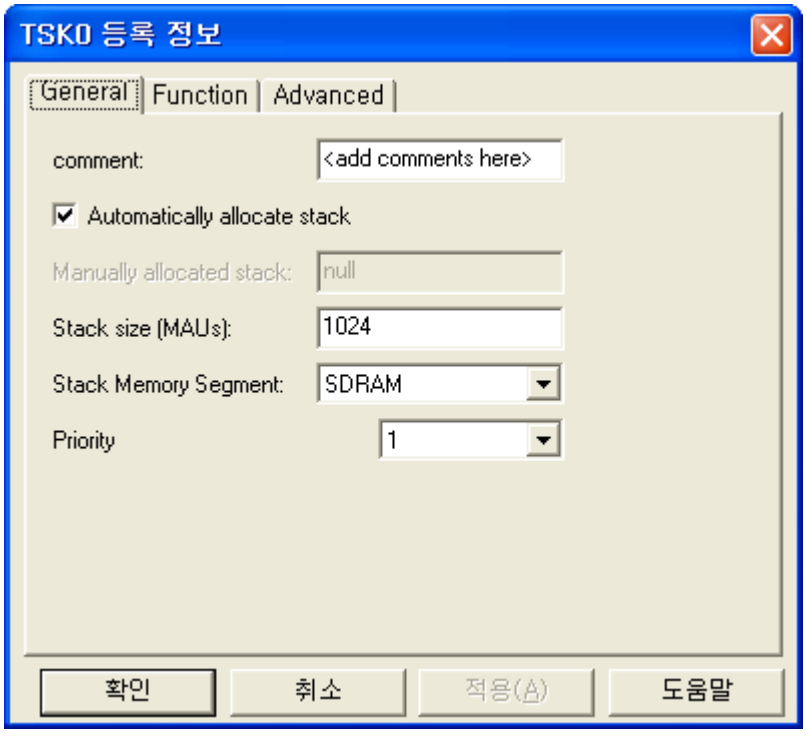
주기가 짧은 타이머일수록 우선 순위를 높게 설정해 주셔야 우선순위에 밀려 수행이 아예 안되거나 SKIP 되는 회수가 많아지는 것을 사전에 방지할 수 있습니다.

5. TASK 추가

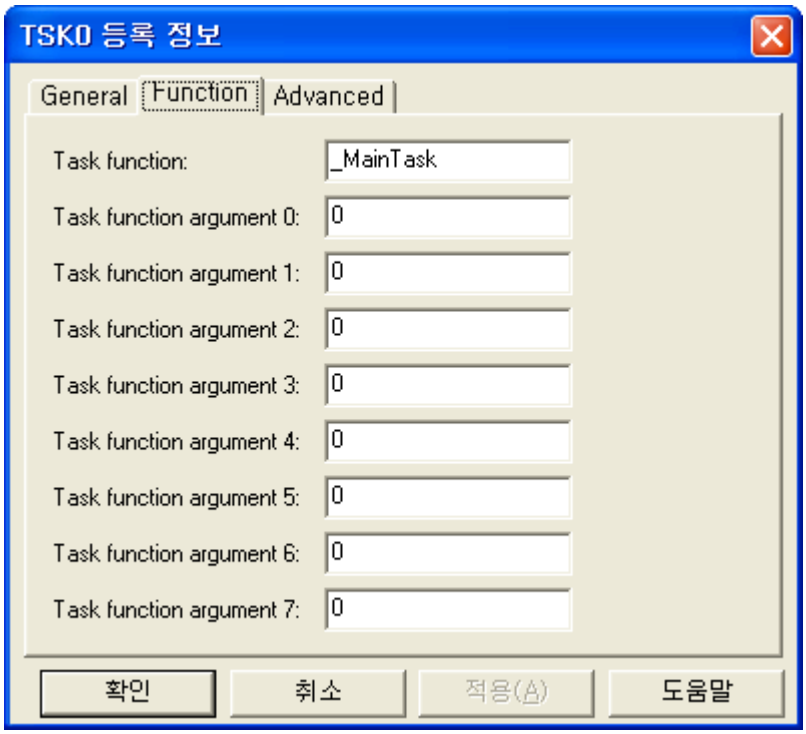
DSP/BIOS 설정을 통해 TASK (쓰레드 개념) 를 사용하실 수 있습니다. 이 경우 main 함수는 단지 엔트리 포인트 (프로그램 진입점) 에 불과하며 main 함수에서는 각종 초기화 작업과 같은 간단한 코드만 사용되며 실제 메인 작업은 TASK 함수를 통해 이루어지게 됩니다.

Main 함수에서 별도로 TASK 함수를 호출하지 않아도 기본적으로 자동으로 TASK 함수가 호출됩니다.



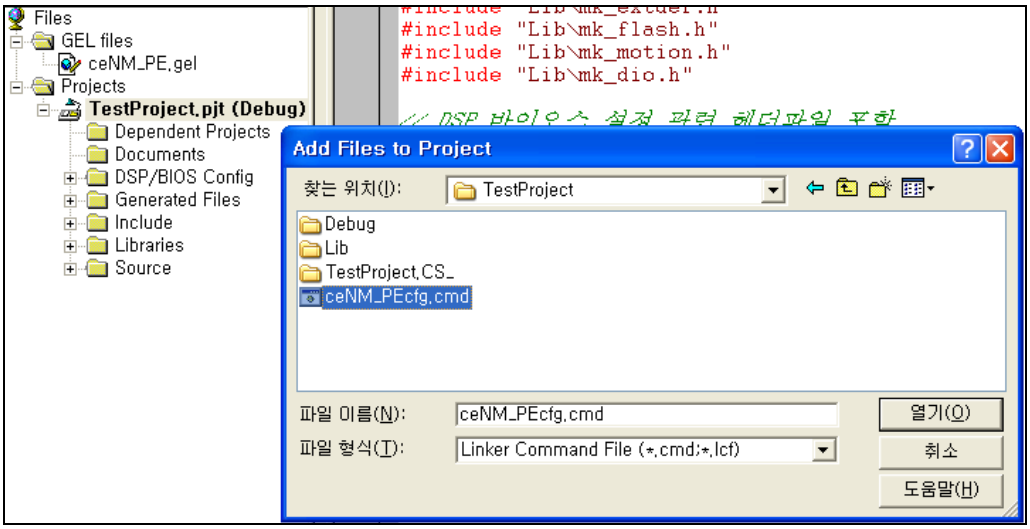


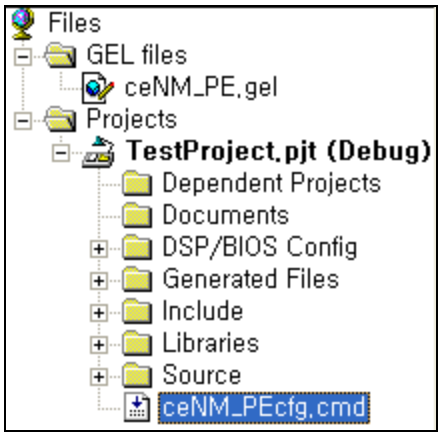
Function 탭에서 Task function 을 입력하면 됩니다. 여기에 입력되는 "_" (언더바) 를 제외한 "MainTask" 함수를 main.c 소스에 구현해서 사용하시면 됩니다. 함수 구현은 아래 main.c 소스 구성편을 참고 하시면 됩니다.



6. DSP/BIOS 설정 저장 및 프로젝트에 추가

DSP/BIOS Config 설정이 끝나셨으면 “Save” 메뉴를 클릭하시어 바이오스 설정을 저장합니다. 결과적으로 ceNM_PEcfg.cmd 파일이 생기며, 이 파일을 프로젝트에 추가시켜 주시면 됩니다.



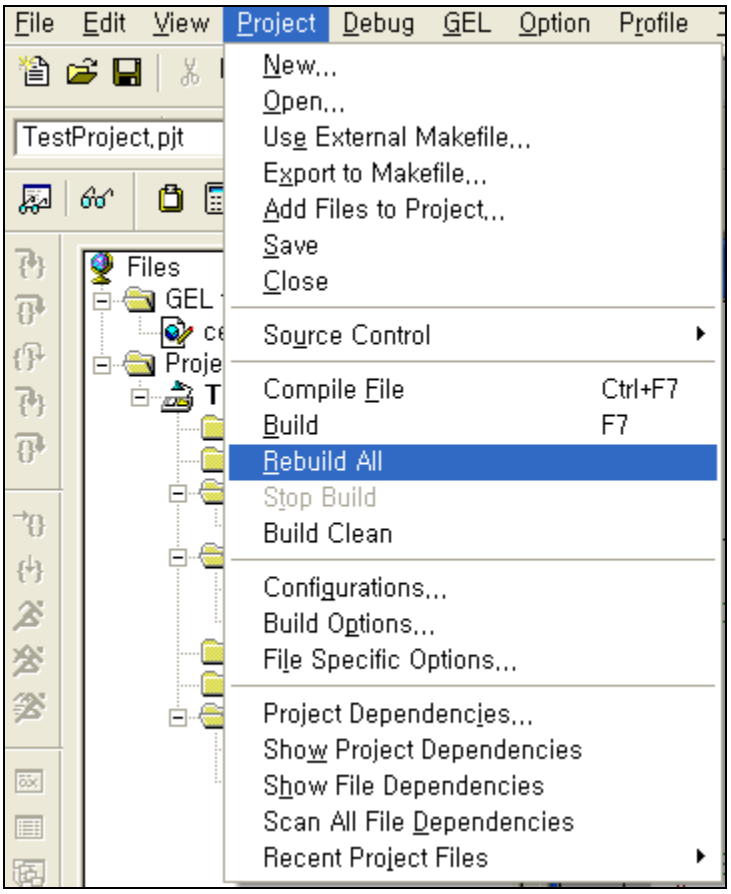


7. main.c 파일 구성 확인

```
#include <stdio.h>
#include <stdlib.h>
#include <c6x.h>
#include <math.h>
#include <log.h>
#include <std.h>
#include <swi.h>

// DSP/BIOS 설정 관련 헤더파일 포함. DSP/BIOS Config 설정 저장 시 생성되는 헤더파일로 여기에 포함시켜 주셔야 합니다.
#include "ceNM_PEcfg.h"

// C6711 DSP 관련 각종 DEFINE 상수 포함
#include "c6711_hrrm.h"

// ㈜커미조아 제공, MK Library 헤더파일 포함
#include "Lib\mk_common.h"
#include "Lib\mk_dio.h"
#include "Lib\mk_motion.h"
#include "Lib\mk_flash.h"
#include "Lib\mk_net.h"
#include "Lib\mk_extdef.h"
#include "Lib\mk_extgvar.h"

void Init_6711(void); // DSP Initialize 함수

void main(void)
{
    // CPU 초기화 루틴
    Init_6711() ;      // in c6711_hrrm.c => 인터럽트 초기화, PLL설정(167MHz), EMIF초기화,

    // CEIP (모션, 디지털입출력 등) 모듈 스캔 및 초기화 작업 수행
    mkcBootup();    // COMIZOA LIBRARY INITIALIZATION
}

void MainTask(Arg id_arg)
{
     while(1) {
```

```
            // 메인 작업을 여기에 구현합니다.
        }
}

// 1밀리초 주기로 동작하도록 구현된 소프트웨어 인터럽트 타이머 함수 입니다.
void SwiTimer_1ms(void)
{
    t_ui32 i;
    mkcTimerCallback();
    // 1밀리 타이머 Tick 을 발생시키는 타이머 콜백 함수로 본 함수가 호출되면 내부 타
    이머 Tick 변수값 이 1밀리초 마다 하나씩 증가되며 이 값은 언제든지 mkcTimerTick()
    함수를 사용해서 얻을 수 있습니다.

     for(i=0; i<g_uSerPortCnt; i++) {       // 시리얼 모듈의 포트 개수만큼 루프를 돌립니다.
          serISR(i);            // 시리얼 통신 패킷(RX / TX) 을 처리하기 위한 폴링 방식의
                                콜백 함수를 호출합니다. 시리얼 통신 패킷이 수신이 되었는
                                지를 본 콜백함수를 호출해 주셔야 정상적으로 처리가 됩니다.
     }
}

// 500 밀리초 주기로 동작하도록 구현된 소프트웨어 인터럽트 타이머 함수 입니다.
void SwiTimer_500ms(void)
{
    static bool bOnOff = true;
    Act_LED_On(bOnOff);       // 노드마스터 모듈(ceNM-PE) 의 Active LED 를 500밀리초
                              간격으로 On/Off 를 반복시킵니다.
    Err_LED_On(bOnOff);       // 노드마스터 모듈(ceNM-PE) 의 Error LED 를 500밀리초
                              간격으로 On/Off 를 반복시킵니다.
    bOnOff = !bOnOff;
}

void Init_6711(void)  // DSP Initialize Code
{
 // 내용 생략
}
```

8. 프로젝트 빌드



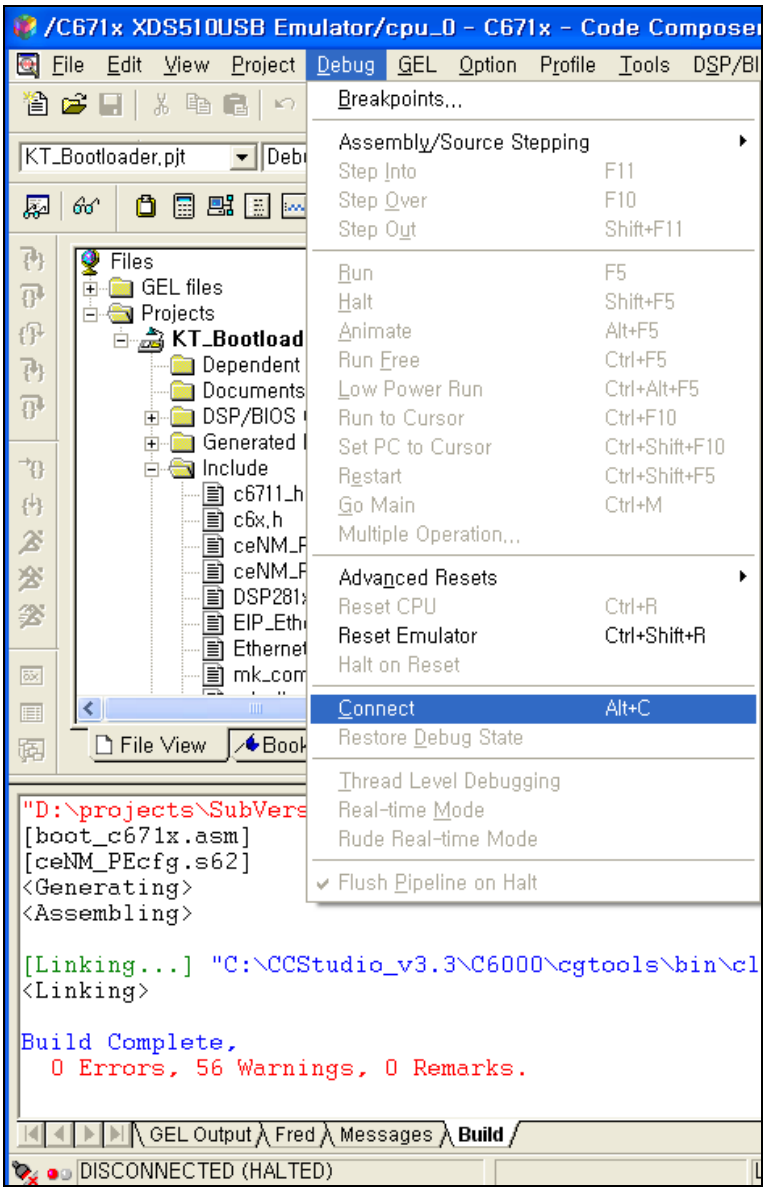
에러 없이 빌드가 성공하면 아래 그림과 같이 Build Complete 메시지가 출력창에 나타납니다.

```
"D:\projects\SubVersionProjects\_KOREATECHNO\SRC\Projects\KT_Bootloade
[boot_c671x.asm]
[ceNM_PEcfg.s62]
<Generating>
<Assembling>

[Linking...] "C:\CCStudio_v3.3\C6000\cgtools\bin\cl6x" -@"Debug.lkf"
<Linking>

Build Complete,
  0 Errors, 56 Warnings, 0 Remarks.
```

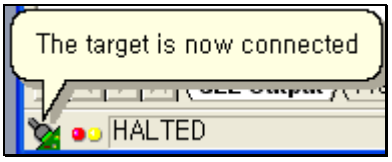
9. Connect

프로젝트 빌드가 성공했으면 프로그램을 메모리에 올리기 전에 코드컴포저 툴과 DSP 모듈이 JTAG 를 통해 연결된 상태인지 확인을 하셔야 합니다. Debug – Connect 명령은 프로그램 로드전에 아무때나 수행하셔도 됩니다.

Debug – Connect 명령 수행

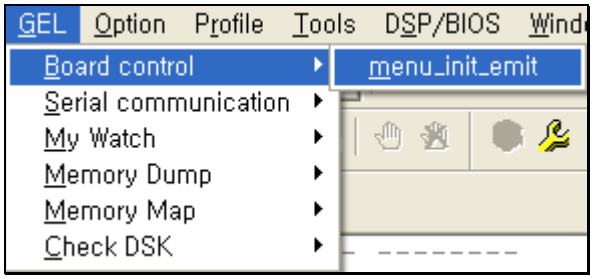


Connection 이 성공하면 아래 그림 처럼 접속상태가 표시되고 CPU 는 Halt 상태가 됩니다.



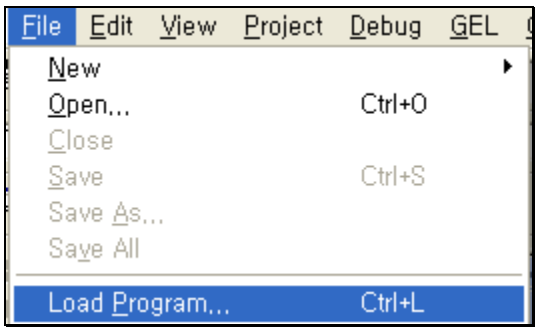
10. SDRAM 메모리 설정 초기화

프로그램이 SDRAM 메모리에 제대로 올라가려면 SDRAM 메모리 인터페이스 설정이 선행되어야 합니다. ㈜커미조아에서 제공해 드린 ceNM_PE.gel 파일에 SDRAM 메모리 설정 초기화 관련된 GEL 함수가 추가되어 있으며 이 GEL 함수 (menu_init_emit) 는 GEL 파일이 로드될 때 자동으로 호출되도록 이 역시 ceNM_PE.gel 파일에 구현이 되어 있습니다. 수동으로 이 함수를 호출하려면 "GEL – Board control – menu_init_emit" 메뉴를 클릭하시면 됩니다.

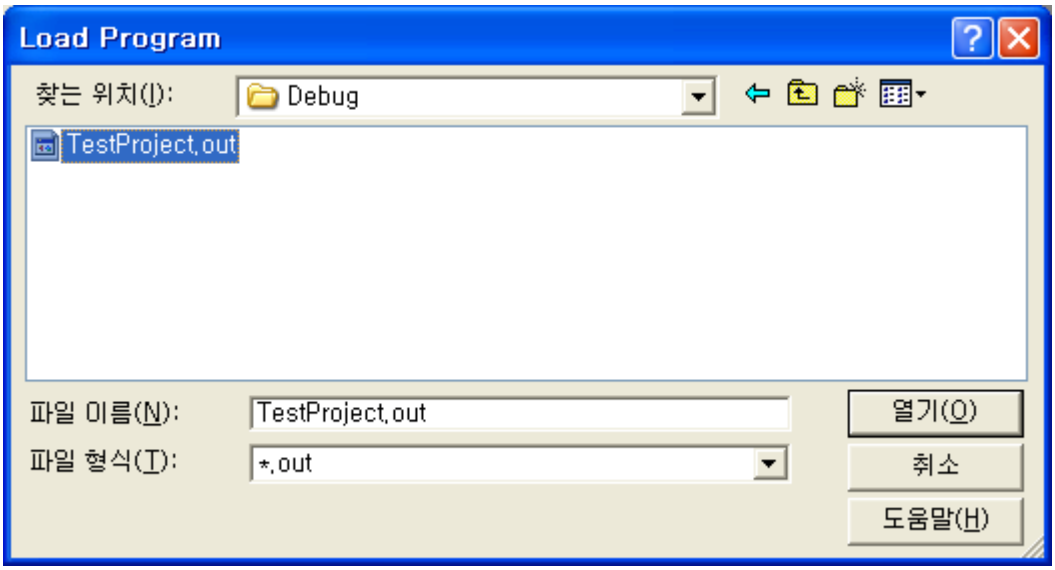


11. 프로그램 로드 (Load Program)

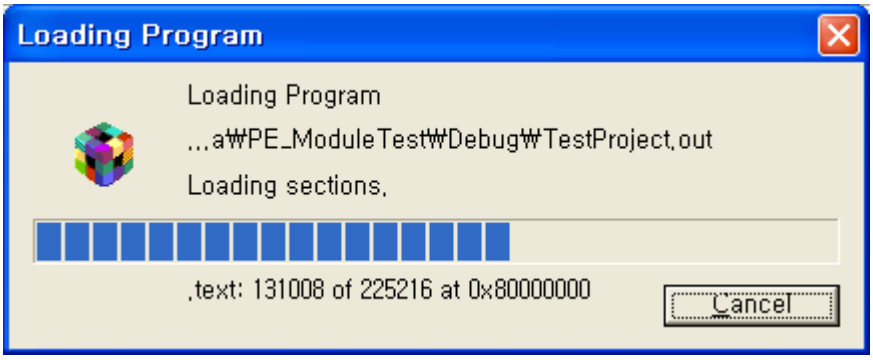
SDRAM 메모리 인터페이스 설정이 성공했으면 이제 프로그램을 메모리에 올리시면 됩니다. "File – Load Program" 메뉴를 클릭하시면 메모리에 로드되는 진행상태 표시창이 나타나며 프로그램 로드 작업이 시작됩니다.



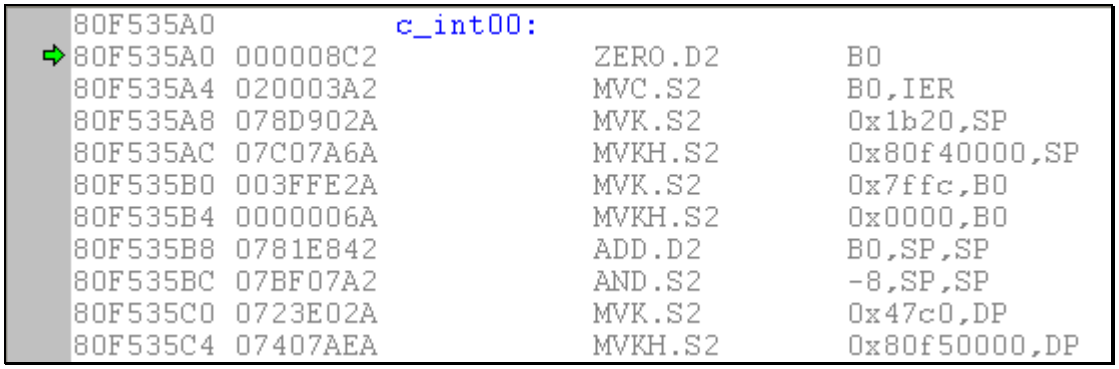
메모리에 로드할 프로그램을 선택합니다. 빌드가 완료된 .out 파일을 지정하면 됩니다.



"열기" 버튼을 누르면 프로그램 로딩 작업이 진행됩니다.

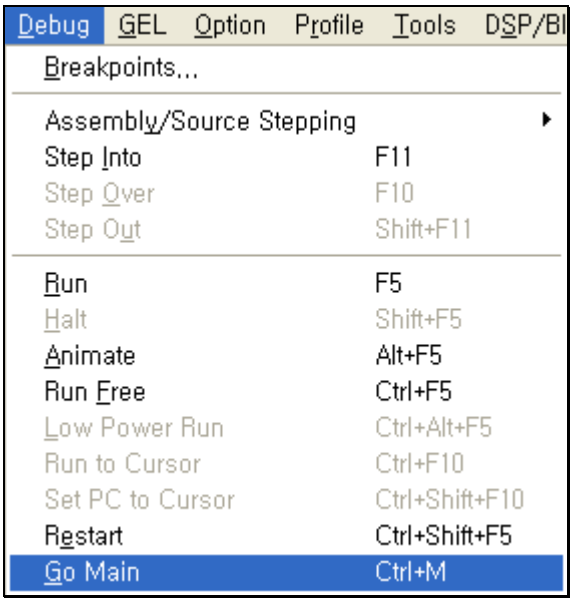


프로그램 로드가 정상적으로 완료되면 아래 그림과 같이 프로그램 시작 주소로 분기가 됩니다.

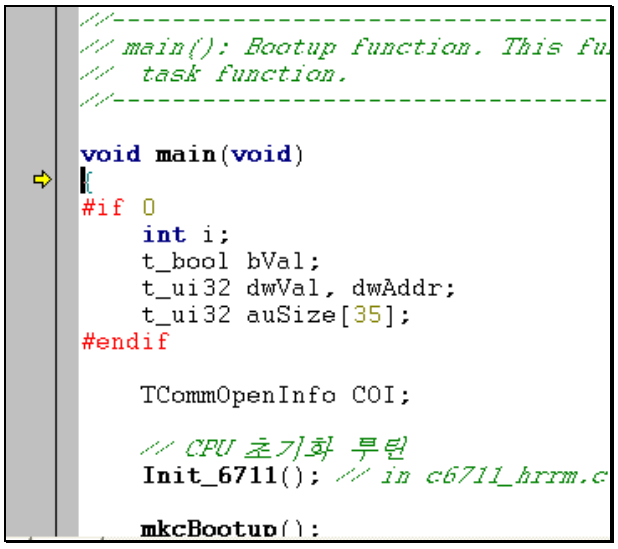


12. Go Main !!

프로그램 로드 (또는 재로드) 후에 Run 을 곧바로 수행하지 마시고 엔트리 포인트인 Main 함수로 정상적으로 진입하였는지 확인을 먼저 하는 것이 좋습니다. 이는 CCS3.3 IDE 환경에서 개발하면서 JTAG 에뮬레이터의 전원공급 상태 등의 외부 요인에 따라 가끔 프로그램이 제대로 로드가 안되는 경우가 발생할 수 있습니다. 따라서 만일 프로그램 로드 후 "Debug – Go Main" 명령으로 main 함수로 진입이 안되고 엉뚱한 메모리 주소로 분기된 경우가 발생하면 CPU 를 Halt 하거나 DSP Reset 후에 "File – Reload Program" 을 해 주신 후 다시한번 "Go Main" 을 시도하시면 됩니다.

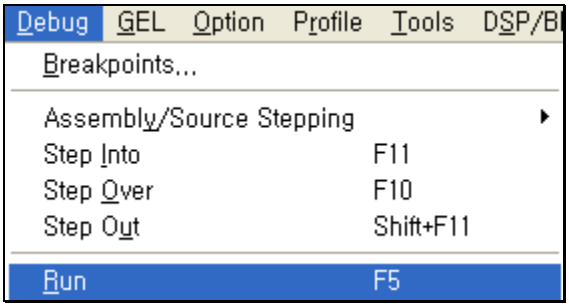


Main 함수로 진입이 성공하면 아래 그림과 같이 main 함수의 시작 부분을 커서가 가리키게 됩니다.



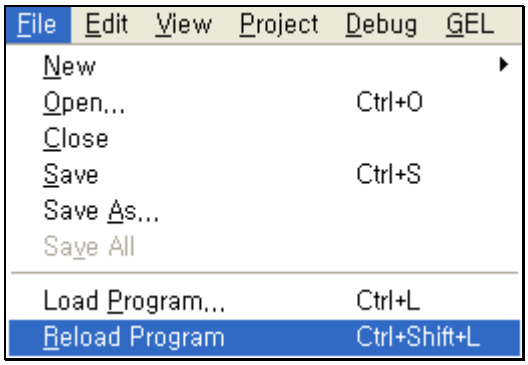
13. Run

엔트리 포인트인 main() 함수로 정상적으로 진입했으면 이제 브레이크 포인트를 설정해 놓고 디버깅 모드로 동작을 시킬 수 있습니다. 일반적인 디버깅 환경과 마찬가지로 “F5” 펑션 키를 누르시면 프로그램이 Run 상태로 됩니다.



14. 프로그램 재로드 (Reload Program)

프로그램 소스가 변경되지 않은 상태에서 다시 메모리에 로드하려면 File – Reload Program 메뉴를 클릭하시면 됩니다.

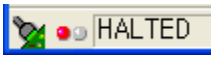


만일 해당 메뉴가 활성화 되지 않고 Dimmed 상태라면 DSP 모듈과 Connection 은 되어 있는지 그리고 CPU 가 Running 상태인지 확인하셔야 합니다. 만일 Connection 이 안되어 있다면 Debug – Connect 메뉴를 사용하셔서 연결을 하시면 되고,

하단 왼쪽 상태창에 CPU 가 Running 상태로 되어 있으면 Debug – Reset CPU 메뉴를 클릭하시거나 Debug – Halt 메뉴를 클릭하셔서 CPU 를 Halt 시키셔야 Load 및 Reload 가 가능합니다.

DSP 모듈이 연결되고 CPU 가 동작하고 있는 상태

DSP 모듈이 연결은 되었으나 CPU 가 동작하지 않고 멈춰 있는 상태



DSP 모듈과의 연결이 끊어진 상태

1.3 메모리 활용 편

1. ceNM-PE 메모리 맵

기본적인 메모리 맵 내용은 TMS320C6711D Memory Map을 참고하시기 바랍니다.

보드에 추가된 Peripheral에 대한 Memory Map입니다. 자세한 내용은 칩 datasheet를 참고하시기 바랍니다.

< Memory Map>

Memory Block Description	Data Bus Width	Hex Base Address
SDRAM(MT48LC16A2)	(16Mx16 Bit)	0x80000000
Flash(AM29LV160)	(1Mx16 Bit)	0x90000000
UART(16c550)	8 Bit	0xA0000000
E-NET(WT5100)	8 Bit	0xA0080000
IP Rotary S/W Read	8 Bit	0xA0100000
Extension BUS0	8 Bit	0xB0000000 (Add+4)
Extension BUS1	8 Bit	0xB0001000 (Add+4)
…..	…	…
Extension BUS9	8 Bit	0xB0009000 (Add+4)

<Interrupt>

E-NET(WT5100) INT	DSP INT4
UART(16c550) INT	DSP INT5
Extension BUS INT	DSP INT6

<GPIO>

Act LED Control	DSP GPIO(TOUT0)로 제어 합니다.
Error LED Control	DSP GPIO(TOUT1)로 제어 합니다.
Flash Busy Check	DSP GPIO2로 check 합니다.

2. SDRAM 메모리 사용

1.2절 DSP/BIOS 의 3. System – MEM – Memory Section Manager – SDRAM 메모리 설정 편에서 언급했던 SDRAM 메모리 영역에 대해 부트로드 전용으로 할당 된 시스템 전용 영역과 유저가 구현한 프로그램 코드 및 데이터 영역으로 사용 가능한 영역을 다음과 같이 구분하였습니다. 개발 시 착오 없으시기 바랍니다.

메모리 범위	크기	비고
0x80000000 ~ 0x80EFFFFF	0x00F00000 (15Mx16 Bit)	유저 코드 및 데이터 사용 영역 개발 시 사용 가능한 영역
0x80F00000 ~ 0x80FFFFFF	0x00100000 (1Mx16 Bit)	부트로더 사용 영역 개발 시 사용 불가한 영역

3. 플래쉬 섹터 구조

메모리 범위	크기	비고
0x90000000 ~ 0x900F7FFF	0x000F8000 (31 * 32 KWord)	SA0 ~ SA30 각 섹터별 32 KWord 의 동일 크기 SA0과 SA1 : 부트로더 코드 영역 개발 시 사용 불가한 영역 SA2 ~ SA30 : 유저 코드 및 데이터 다운로드 가능 영역, 개발 시 사용 가능한 영역
0x900F8000 ~ 0x900FBFFF	0x00004000 (16 KWord)	SA31 유저 코드 및 데이터 다운로드 가능 영역, 개발 시 사용 가능한 영역
0x900FC000 ~ 0x900FCFFF	0x00001000 (4 KWord)	SA32 유저 코드 및 데이터 다운로드 가능 영역, 개발 시 사용 가능한 영역
0x900FD000 ~ 0x900FDFFF	0x00001000 (4 KWord)	SA33 다운로드 정보 기록 영역 개발 시 사용 불가한 영역
0x900FE000 ~ 0x900FFFFF	0x00002000 (8 KWord)	SA34 유저 코드 및 데이터 다운로드 가능 영역, 개발 시 사용 가능한 영역

CHAPTER 2. MK Library API Manual

2-1. 데이터 형 표기 및 사용 시 유의사항

당사의 Ethernet I/P 모듈 펌웨어 라이브러리는 TMS320VC33, TMS320F2812 및 TMS320C6211 등 다양한 CPU 를 지원하며 Code Composer 개발 환경에서 C/C++ 문법을 동일하게 사용합니다. 하지만 CPU 및 컴파일러 특성에 따라 일부 데이터 형에서 Ansi C 문법에 비해 데이터 형의 크기(바이트 수)가 다소 차이가 있을 수 있으므로 아래 **표 2 데이터 형 표기** 같이 데이터 형을 통일하여 별칭으로 표기 하였습니다. 실제로 코딩 시에는 해당 플랫폼에 따라서 별칭으로 표기된 데이터 형에 대응되는 실제 데이터 타입을 사용하셔야 합니다. 또한 사용하시는 CPU 의 종류에 따라 아래와 같이 mk_common.h 파일에서 #define 문을 통해 사용 플랫폼을 설정해 주셔야 합니다.

```
#define PLF_miCUBE          0
#define PLF_miCUBE_E        1     // with TMS VC33 CPU
#define PLF_EtherIP_1       2     // with TMS320F2812 CPU
#define PLF_EtherIP_2       3     // with TMS320C6711 CPU
#define _PLATFORM   PLF_EtherIP_2 //_PLATFORM 정의로 플랫폼을 구분합니다.
```

표 1 mk_common.h 파일내에서의 플랫폼 정의 예

Data type	Description	VC33	C6711
t_bool	참(TRUE) 또는 거짓(FALSE) 상태 표현	char	char
t_char	1바이트 부호 있는 데이터 표현	char	char
t_uchar, t_byte	1바이트 부호 없는 데이터 표현	unsigned char	unsigned char
t_i16	2바이트 부호 있는 정수형 데이터 표현	short	short
t_ui16	2바이트 부호 없는 정수형 데이터 표현	unsigned short	unsigned short
t_i32	4바이트 부호 있는 정수형 데이터 표현	int, long	int
t_ui32	4바이트 부호 없는 정수형 데이터 표현	unsigned long	unsigned int
t_f32	4바이트 부호 있는 실수형 데이터 표현	double	float
HANDLE	4바이트 크기로 메모리에 유지하는 정보 블록에 붙은 고유 번호 표현	unsigned long	unsigned int
DWORD	4바이트 부호 없는 정수형 데이터 표현	unsigned long	unsigned int

표 2 데이터형 표기

2-2. 라이브러리 사용 법

당사의 Ethernet I/P 모듈 펌웨어 라이브러리를 사용하시려면 다음과 같이 개발 중이신 프로젝트에 라이브러리 파일 추가, 헤더파일 포함 및 라이브러리 함수 호출 사용의 순서로 하시면 됩니다.

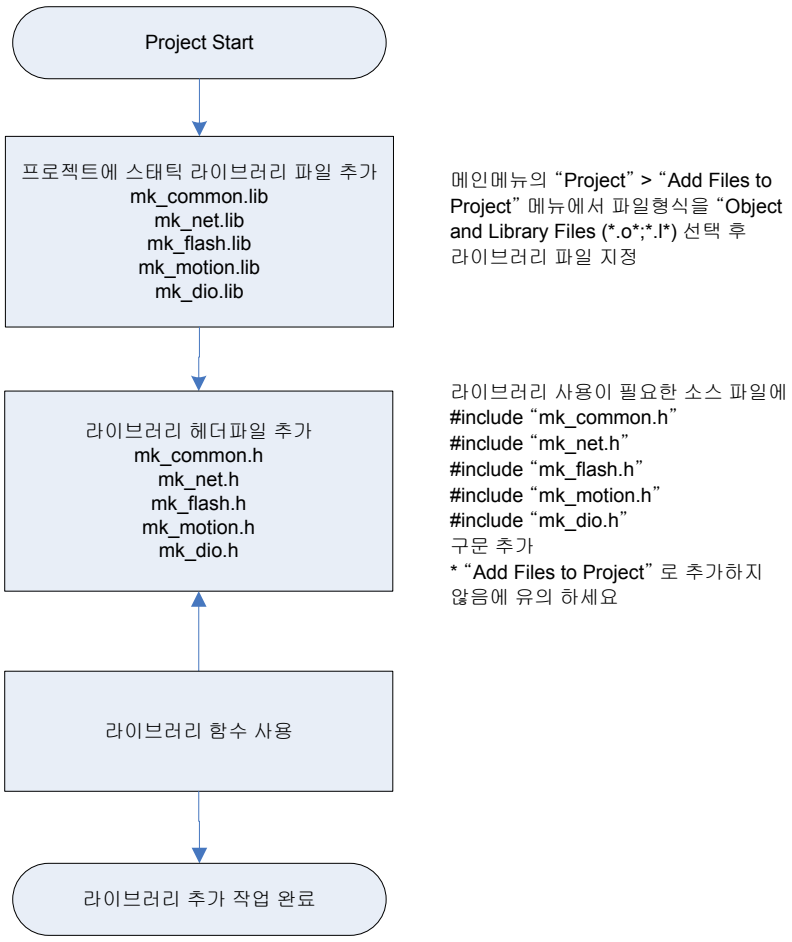


그림 1 라이브러리 사용하기

2-3. Common Functions

Summary of Functions
❒ void **mkcBootup** (void) main 함수의 시작부분에서 반드시 호출 해야 하는 시스템 초기화 함수입니다.
❒ t_ui32 **mkcTimerTick** (void) 1 ms 단위의 tick count 값을 반환하는 타이머 함수입니다.
❒ void **mkcTimerTick_Set** (t_ui32 dwSetValue) 1 ms 단위의 tick count 값을 설정하는 함수입니다.
❒ void **mkcTimerCallback** (void) 타이머 인터럽트 서비스 루틴에서 반드시 호출 해야 하는 타이머 콜백함수입니다.
❒ void **SetBit** (t_ui32 dwVal, t_ui32 dwBitIdx, t_bool bState) 지정한 비트의 값을 변경하는 함수입니다.
❒ void **Delay_us** (t_ui32 dwDelay) 1마이크로 초 단위로 지정된 Delay 수만큼 프로세스를 지연시키는 함수입니다.
❒ void **Delay_ms** (t_ui32 dwDelay) 1밀리 초 단위로 지정된 Delay 수만큼 프로세스를 지연시키는 함수입니다.
❒ t_ui32 **atoh** (t_char *str) 16진수 아스키 스트링을 부호 없는 4바이트 부호 없는 정수형 값으로 변환해주는 함수입니다.
❒ t_ui16 **Hex2Str** (t_ui32 dwValue, t_char *pszBuf, t_i16 nWidth, t_bool bFillZero, t_bool bCapital) 4바이트 부호 없는 정수형 값을 지정한 길이를 가진 문자열로 변환해주는 함수입니다.
❒ t_ui32 **FltCnvt_I2T** (t_ui32 dwIeee_v) IEEE 형식의 4바이트 부동소수점표현 식을 TI 형식의 4바이트 부동소수점 표현 식으로 변환시켜주는 함수입니다.
❒ t_ui32 **FltCnvt_T2I** (t_ui32 dwTms_v) TI 형식의 4바이트 부동소수점표현 식을 IEEE 형식의 4바이트 부동소수점 표현 식으로 변환시켜주는 함수입니다.
❒ t_f32 **Long2Float** (t_i32 nVal) 4바이트 정수형 변수에 담겨진 부동소수점표현 식을 부동소수점수로 변환합니다
❒ t_i32 **Float2Long** (t_f32 fVal) 부동소수점수를 4바이트 부동소수점 표현 식으로 변환합니다
❒ void **Outp32** (t_i32 nCS, t_ui32 dwOffset, t_i32 nVal) 해당 변수의 4바이트 크기 단위인 오프셋 주소에 4바이트 크기의 지정한 데이터를 쓰는 함수입니다.
❒ t_i32 **Inp32** (t_i32 nCS, t_ui32 dwOffset) 해당 변수의 4바이트 크기 단위인 오프셋 주소로부터 4바이트 크기의 저장된 데이터를 읽

어오는 함수입니다.
❒ void **Outp16** (t_i16 nCS, t_ui16 wOffset, t_i16 nVal) 해당 변수의 2바이트 크기 단위인 오프셋 주소에 2바이트 크기의 지정한 데이터를 쓰는 함수입니다.
❒ t_i32 **Inp16** (t_i16 nCS, t_ui16 wOffset) 해당 변수의 2바이트 크기 단위인 오프셋 주소로부터 2바이트 크기의 저장된 데이터를 읽어오는 함수입니다.
❒ void **Outp8** (t_char nCS, t_uchar uOffset, t_char nVal) 해당 변수의 1바이트 크기 단위인 오프셋 주소에 1바이트 크기의 지정한 데이터를 쓰는 함수입니다.
❒ t_i32 **Inp8** (t_char nCS, t_uchar uOffset) 해당 변수의 1바이트 크기 단위인 오프셋 주소로부터 1바이트 크기의 저장된 데이터를 읽어오는 함수입니다.
❒ void **Act_LED_On** (t_byte On) Gpio.lib 의 함수로 ceNM-PE 노드마스터 모듈의 Active LED 를 On/Off 시키는 함수입니다.
❒ void **Err_LED_On** (t_byte On) Gpio.lib 의 함수로 ceNM-PE 노드마스터 모듈의 Error LED 를 On/Off 시키는 함수입니다.
❒ t_i32 **GetSerDigInput** (t_i8 ch) Gpio.lib 의 함수로 ceNM-PE 노드마스터 모듈의 시리얼 포트에서 사용하지 않는 입력핀에 대해 Digital 입력채널용도로 사용하기 위한 함수입니다. (현재 3개의 입력 채널이 이용가능합니다. 자세한 내용은 노드마스터 모듈 하드웨어 매뉴얼을 참고하십시오)

▣ mkcBootup

함수 원형

void **mkcBootup** (void)

함수 설명

시스템을 초기화하는 함수입니다. 이 함수는 main()의 시작부분에서 반드시 호출해주어야 합니다.

참고

- ☐ 이 함수를 호출하기 전에 INT0(1st interrupt)를 활성화해주어야 합니다. 이 것은 이더넷칩을 초기화하는데 인터럽트가 사용되는데 이더넷칩이 사용하는 인터럽트가 INT0이기 때문입니다.
- ☐ mkcBootup() 함수는 mk_extern.c 소스파일에 있는 함수로서 다음과 같이 그 내용이 구성되어 있습니다. 일반적으로는 이 함수의 내부 구성은 수정될 필요가 없습니다.

```
void mkcBootup(void)
{
    flsBootup(); // flash memory 부팅
    mcBootup(); // 모션 모듈 부팅, 모션 모듈 비사용시 주석처리
    dioBootup(); // Digital I/O 모듈 부팅, I/O 모듈 비사용시 주석처리
    aioBootup(); // Analog I/O 모듈 부팅
    g_uSerPortCnt = serBootup(); //시리얼 모듈 부팅

    g_auIpAddr[3] = GetLastIpAddrFromGPIOF();

    if(g_auIpAddr[3] != 0 && g_auIpAddr[3] != 255)
    {
        netBootup(g_auIpAddr); // 이더넷 초기화, ip 및 Mac Address 설정
        Act_LED_On(1);
        Err_LED_On(1);
    }
    else
    {
        // IpAddr[3] 이 0 또는 255 인 경우는 에러 처리
        Act_LED_On(0);
        Err_LED_On(1);
        while(1)
        {
        }
    }
    commBootup(); // 통신 초기화
}
```

예제

```
//-----------------------------------------------------------------------------------------------------------
// Init_6711 (): initialize DSP processor. 이 함수는 아래와 같이 구성해줍니다.
//-----------------------------------------------------------------------------------------------------------
void Init_6711(void)
{
CSR = 0x100;                    // Disable all interrupts
IER = 1;                        // Disable all interrupts except NMI
ICR = 0xffff;                   // Clear all pending interrupts

*(unsigned volatile int *)PLL_M = 0x0007;          // PLLM x7 설정  167MHz (Base 25MHz)
*(unsigned volatile int *)PLL_CSR &= ~0x0008;  // PLL Reset Release
*(unsigned volatile int *)PLL_CSR |= 0x0001;      // PLL Enable
Delay_us(10);

*(unsigned volatile int *)EMIF_GCR = 0x3078;    // EMIF global control
*(unsigned volatile int *)EMIF_CE0 = 0x33; // EMIF CE0 control (32bit SDRAM 8M x16)
*(unsigned volatile int *)EMIF_CE0 = CE0_32;    // EMIF CE0 control (32bit SRAM)
*(unsigned volatile int *)EMIF_CE1 = CE1_32;    // EMIF CE1 control (8bit ROM) 16bit 로
*(unsigned volatile int *)EMIF_CE2 = CE2_32;    // EMIF CE2 control (I/O)
*(unsigned volatile int *)EMIF_CE3 = CE3_32;    // EMIF CE3 control (I/O)
*(unsigned volatile int *)EMIF_SDCTRL = 0x6348F000;     // EMIF SDRAM control
*(unsigned volatile int *)EMIF_SDRP = 0x0200030D;// EMIF SDRM refresh period, SDTIM
*(unsigned volatile int *)EMIF_SDEXT= 0x00058D29;       // EMIF SDRM extension
Delay_us(10);

/*****************************************************************************
 * Start state
 *****************************************************************************/
*(unsigned volatile int *)GPEN = 0x0084;          // Enable GPIO2   (bit2)(Flash Busy
Input), GPIO7(DI2)
*(unsigned volatile int *)GPDIR = 0x0000;         // IN : GPIO2 (bit2) }

void main()
{
    // CPU 초기화 루틴
    Init_6711(); // PLL 설정(167MHz), EMIF 초기화.

    mkcBootup();
    OpenSockets();
}
```

▣ mkcTimerTick

함수 원형

t_ui32 **mkcTimerTick** (void)

함수 설명

1 ms 단위의 tick count 값을 반환하는 함수입니다. 일반적으로 경과시간을 체크할 때 사용됩니다.

반환값

1 ms 단위의 tick count. 이 값은 시스템이 부팅된 이후에 절대적으로 증가합니다.

참고

□ 기본적으로 이 함수는 **mkcTimerCallback()** 함수가 호출된 횟수를 반환하도록 되어 있습니다. 따라서 **mkcTimerCallback()** 함수는 1ms 주기의 타이머 인터럽트 서비스 루틴에서 호출되어져야 합니다. 만일 **mkcTimerCallback()** 함수를 호출하는 타이머 ISR의 주기가 1ms와 다른 경우에는 **mk_common.c** 소스파일에서 **mkcTimerTick()** 함수의 내부를 수정하여 반환값의 단위가 1ms 되도록 조정하여야 합니다.

예제

```
t_ui32 uStartT, uElpsT;

uStartT = mkcTimerTick(); // 시작시간 저장
… // do something
uElpsT = mkcTimerTick() – uStartT; // 경과시간 계산
```

▣ mkcTimerTick_Set

함수 원형

void **mkcTimerTick_Set** (t_ui32 dwSetValue)

함수 설명

1 ms 단위의 tick count 값을 설정하는 함수입니다. 일반적으로 Host PC 로부터 Time Stamp 를 읽어서 동일한 시간 값으로 로컬 타임 스탬프를 설정할 때 사용됩니다.

매개 변수

▶ **dwSetValue:** 설정할 밀리초 단위의 값

반환값

없음

예제

```
void cnSetLocalTimestamp(t_ui32 dwTimetick)
{
    mkcTimerTick_Set(dwTimetick);
}

if(commReadDword(st, nSockChan, &dwTemp, 1) >= 1) // Read time-stamp //
    cnSetLocalTimestamp(dwTemp);
```

▣ mkcTimerCallback

함수 원형

void **mkcTimerCallback** (void)

함수 설명

이 함수는 라이브러리 내부적으로 활용되는 타이머 콜백함수입니다. 사용자는 반드시 타이머 인터럽트 서비스 루틴에서 이 함수를 호출해주어야 합니다.

참고

□ 사용자는 반드시 타이머 인터럽트 서비스 루틴에서 이 함수를 호출해주어야 합니다.

예제

아래와 같이 1ms 타이머 인터럽트 서비스 루틴에서 호출하여 주십시오

```
void Timer1ms_swi(void)
{
        t_ui32   i;
        for(i=0; i<5; i++)
        {
                serISR(i);
        }
        mkcTimerCallback();
}
```

▣ SetBit

함수 원형

void **SetBit** (t_ui32 dwVal, t_ui32 dwBitIdx, t_bool bState)

함수 설명

지정한 비트의 값을 변경하는 함수입니다.

매개 변수

- **dwVal:** 변경시킬 변수
- **dwBitIdx:** 0(zero) 기반의 해당 비트 Index
- **bState:** 변경값. 0(TRUE) 또는 1(FALSE)

반환값

없음

예제

```
t_i32 nDI, nLCH;
nDI = dioGetDevIdx(nChannel); // device index
nLCH = dioGetLocChan(nChannel); // local channel no.
bState = !bState;
SetBit(g_aDioDev[nDI].dwDioStates, nLCH, bState);
```

▣ Delay_us

함수 원형

void **Delay_us** (t_ui32 dwDelay)

함수 설명

1마이크로초 단위로 지정된 Delay 수 만큼 프로세스를 지연시키는 함수입니다.

매개 변수

▶ **dwDelay:** 마이크로초 단위의 지연 시간

반환값

없음

예제

```
val = nDioLedStatus[nModIdx] & 0x7;
Outp8(hDevice, 0x90, val);
Delay_us(10);
```

▣ Delay_ms

함수 원형

void **Delay_ms** (t_ui32 dwDelay)

함수 설명

1밀리초 단위로 지정된 Delay 수 만큼 프로세스를 지연시키는 함수입니다.

매개 변수

▶ **dwDelay:** 밀리초 단위의 지연 시간

반환값

없음

예제

```
flsEraseSector(34);
// Erase 작업이 완료될 때까지 기다린다. //
Delay_ms(1);
```

▣ atoh

함수 원형

t_ui32 **atoh** (t_char *str)

함수 설명

16진수 아스키 스트링을 부호없는 4바이트 부호없는 정수형 헥사 값으로 변환해주는 함수입니다.

매개 변수

▶ **str:** 16진수 아스키 문자열(스트링)

반환값

변환된 4바이트 부호 없는 정수형 헥사 값

예제

```
t_char sBuffer[4];
sBuffer[0] = 'A';
sBuffer[1] = 'A';
sBuffer[2] = '\0'; // 문자열의 끝임을 알리기 위해 반드시 널문자를 끝에 붙여줍니다.
nHex = atoh(sBuffer); // 결과적으로 0xAA 가 nHex 에 대입됩니다.
```

▣ Hex2Str

함수 원형

t_ui16 **Hex2Str** (t_ui32 dwValue, t_char *pszBuf, t_i16 nWidth, t_bool bFillZero, t_bool bCapital)

함수 설명

4바이트 부호없는 정수형 헥사 값을 지정한 길이를 가진 문자열로 변환해주는 함수입니다.

매개 변수

▶ **dwValue:** 변환대상 16진수

▶ **pszBuf:** 스트링을 저장할 버퍼

▶ **nWidth:** 스트링 폭. 이 값이 0 또는 -1이면 leading zero (0) 는 문자로 변환하지 않고 생략

▶ **bFillZero:** 이 값이 1(TRUE) 이면 leading zero 를 '0' 으로 변환

▶ **bCapital:** 이 값이 1(TRUE) 이면 A~F 를 대문자로 변환

반환값

변환된 문자 수

참고

□ 32비트 정수가 아닐 때는 반드시 비트마스크를 수행하여 원하는 비트만 유효한 값을 가지도록 합니다.

예제

```
t_ui16 i, nStrLen=0;

pszBuffer[nStrLen++] = GC_STX;
nStrLen += Hex2Str(nCommand&0xffff, pszBuffer+nStrLen, 4, true, true);
nStrLen += Hex2Str(nNumData&0xff, pszBuffer+nStrLen, 2, true, true);
pszBuffer[nStrLen++] = ':';

for(i=0; i<nNumData; i++){
    if(i==0)
    {
        nStrLen += Hex2Str(sData[i], pszBuffer+nStrLen, -1, false, true);
    }
    else
    {
        pszBuffer[nStrLen++] = ',';
        nStrLen += Hex2Str(sData[i]&0xff, pszBuffer+nStrLen, -1, false, true);
    }
}
```

▣ FltCnvt_I2T

함수 원형

t_ui32 **FltCnvt_I2T** (t_ui32 dwIeee_v)

함수 설명

IEEE 형식의 4바이트 부동소수점표현식을 TMS 형식의 4바이트 부동소수점 표현식으로 변환시켜주는 함수입니다.

매개 변수

▶ **dwIeee_v:** IEEE 형식의 부동 소수점 표현 값

반환값

TMS 형식으로 변환된 부동소수점 표현 값

예제

```
t_ui32 uTemp;
uTemp = flsReadDword(dwAddr);
uTemp = FltCnvt_I2T(uTemp);
```

▣ FltCnvt_T2I

함수 원형

t_ui32 **FltCnvt_T2I** (t_ui32 dwTms_v)

함수 설명

TMS 형식의 4바이트 부동소수점표현식을 IEEE 형식의 4바이트 부동소수점 표현식으로 변환시켜주는 함수입니다.

매개 변수

▶ **dwTms_v:** TMS 형식의 부동 소수점 표현 값

반환값

IEEE 형식으로 변환된 부동소수점 표현 값

예제

```
t_ui32 uTemp;
uTemp = Float2Long(fTmsValue);
uTemp = FltCnvt_T2I(uTemp);
```

▣ Long2Float

함수 원형

t_f32 **Long2Float** (t_i32 nVal)

함수 설명

4바이트 정수형 변수에 담겨진 부동소수점 표현식을 부동소수점 수로 변환합니다.

매개 변수

▶ **nVal:** 4바이트 부동소수점 표현식

반환값

부동소수점 수

예제

```
t_ui32 dwTemp;
dwTemp = FltCnvt_I2T(anParam[3]);
fVel = Long2Float(dwTemp);
```

▣ Float2Long

함수 원형

t_i32 **Float2Long** (t_f32 fVal)

함수 설명

부동소수점 수를 4바이트 부동소수점 표현식으로 변환합니다.

매개 변수

▶ **fVal:** 부동소수점 수

반환값

4바이트 부동소수점 표현식

예제

```
t_ui32 dwTemp;
mcIxGetSpeedPattern(&nMapIdx, &nIsVecMode, &nSpdMode, &fVel, &fAcc, &fDec);
dwTemp = Float2Long(fVel);
nReturn = FltCnvt_T2I(dwTemp);
```

▣ Outp32

함수 원형

void **Outp32** (t_i32 nCS, t_ui32 dwOffset, t_i32 nVal)

함수 설명

해당 변수의 4바이트 크기 단위인 오프셋 주소에 4바이트 크기의 지정한 데이터를 쓰는 함수입니다.

매개 변수

- **nCS:** 해당 변수의 주소
- **dwOffset**: 오프셋 번지
- **nVal:** 기록할 데이터

반환값

없음

예제

```
#define AM_SDRAM            0x80000000
volatile t_i32 *nTargetAddr = (t_i32 *)AM_SDRAM;

Outp32(nTargetAddr, 0x555, 0xaa00aa);
Outp32(nTargetAddr, 0x2aa, 0x550055);
```

▣ Inp32

함수 원형

t_i32 **Inp32** (t_i32 nCS, t_ui32 dwOffset)

함수 설명

해당 변수의 4바이트 크기 단위인 오프셋 주소로부터 4바이트 크기의 저장된 데이터를 읽어오는 함수입니다.

매개 변수

- **nCS:** 해당 변수의 주소
- **dwOffset**: 오프셋 번지

반환값

읽어온 값

예제

```
t_ui16 wDiData;
wDiData = 0xffff & Inp32(aDioDevice[nDI].hDevice, 0);
```

▣ Outp16

함수 원형

void **Outp16** (t_i32 nCS, t_ui16 wOffset, t_i16 nVal)

함수 설명

해당 변수의 2바이트 크기 단위인 오프셋 주소에 2바이트 크기의 지정한 데이터를 쓰는 함수입니다.

매개 변수

- **nCS:** 해당 변수의 주소
- **wOffset**: 오프셋 번지
- **nVal:** 기록할 데이터

반환값

없음

예제

```
#define AM_TEMP 0x2000
Outp16(AM_TEMP, 0, 0x5555);
Outp16(AM_TEMP, 1, 0xAAAA);
```

▣ Inp16

함수 원형

t_i16 **Inp16** (t_i32 nCS, t_ui16 wOffset)

함수 설명

해당 변수의 2바이트 크기 단위인 오프셋 주소로부터 2바이트 크기의 저장된 데이터를 읽어오는 함수입니다.

매개 변수

- **nCS:** 해당 변수의 주소
- **wOffset**: 오프셋 번지

반환값

읽어온 값

예제

```
#define AM_TEMP 0x2000
t_ui16 wDiData;
wDiData = Inp16(AM_TEMP, 0);
```

▣ Outp8

함수 원형

void **Outp8** (t_i32 nCS, t_uchar uOffset, t_char nVal)

함수 설명

해당 변수의 1바이트 크기 단위인 오프셋 주소에 1바이트 크기의 지정한 데이터를 쓰는 함수입니다.

매개 변수

- ▶ **nCS:** 해당 변수의 주소
- ▶ **uOffset**: 오프셋 번지
- ▶ **nVal:** 기록할 데이터

반환값

없음

예제

```
t_i16 val;
val = DioLedStates[0] & 0x7; // 0,1,2 번 비트값을 읽는다.
val |=0x4; // 기존 0,1,2 비트값을 그대로 두고 2 번 비트값(Enable 비트) 만 set 시킨다.
Outp8(aDioDevice[0].hDevice, 0xAB, val);
Delay_us(10);
```

▣ Inp8

함수 원형

t_char **Inp8** (t_i32 nCS, t_char nOffset)

함수 설명

해당 변수의 1바이트 크기 단위인 오프셋 주소로부터 1바이트 크기의 저장된 데이터를 읽어오는 함수입니다.

매개 변수

- **nCS:** 해당 변수의 주소
- **nOffset**: 오프셋 번지

반환값

읽어온 값

예제

```
tli32 dwLow, dwHigh, dwValue;
dwLow = 0xff & Inp8(aPclAxis[0].nBaseAddr, 6);
dwHigh = 0xff & Inp8(aPclAxis[0].nBaseAddr, 7);
dwValue |= (dwHigh<<24) | (dwLow<<16);
```

▣ Act_LED_On

함수 원형

void **Act_LED_On** (t_byte On)

함수 설명

Gpio.lib 의 함수로 ceNM-PE 노드마스터 모듈의 Active LED 를 On/Off 시키는 함수입니다.

매개 변수

▶ **On:** Active LED On/Off 상태 값

반환값

없음

예제

```
void SwiTimer_500ms(void)
{
        static bool bOnOff = true;
        Act_LED_On(bOnOff);        // 노드마스터 모듈(ceNM-PE) 의 Active LED 를
500 밀리초 간격으로 On/Off 를 반복시킵니다.
        Err_LED_On(bOnOff);        // 노드마스터 모듈(ceNM-PE) 의 Error LED 를
500 밀리초 간격으로 On/Off 를 반복시킵니다.
        bOnOff = !bOnOff;
}
```

▣ Err_LED_On

함수 원형

void **Err_LED_On** (t_byte On)

함수 설명

Gpio.lib 의 함수로 ceNM-PE 노드마스터 모듈의 Error LED 를 On/Off 시키는 함수입니다.

매개 변수

▶ **On:** Error LED On/Off 상태 값

반환값

없음

예제

```
void SwiTimer_500ms(void)
{
        static bool bOnOff = true;
        Act_LED_On(bOnOff);         // 노드마스터 모듈(ceNM-PE) 의 Active LED 를
500 밀리초 간격으로 On/Off 를 반복시킵니다.
        Err_LED_On(bOnOff);         // 노드마스터 모듈(ceNM-PE) 의 Error LED 를
500 밀리초 간격으로 On/Off 를 반복시킵니다.
        bOnOff = !bOnOff;
}
```

▣ GetSerDigInput

함수 원형

t_i32 **GetSerDigInput** (t_i8 ch)

함수 설명

Gpio.lib 의 함수로 ceNM-PE 노드마스터 모듈의 시리얼 포트에서 사용하지 않는 입력핀에 대해 Digital 입력채널용도로 사용하기 위한 함수입니다. (현재 3개의 입력 채널이 이용가능 합니다. 자세한 내용은 노드마스터 모듈 하드웨어 매뉴얼을 참고하십시오)

매개 변수

▶ **ch:** Input 채널 번호 (0 ~ 2)

반환값

해당 입력 채널의 On/Off 상태

예제

```
t_i32 val;
val = GetSerDigInput(0);
if(val != 1)
        return;

val = GetSerDigInput(1);
if(val != 1)
        return;

val = GetSerDigInput(2);
if(val != 1)
        return;
```

2-4. Motion Control Functions

Summary of Functions (1) 범용 함수들
❒ void **mcBootup** (void) 모션 장치 초기화 함수입니다. 이 함수는 시스템 부팅함수인 mkcBootup() 함수내부에서 호출되어져야 합니다.
❒ t_i32 **mcGnInitMotion** (t_i32 nNumDevs, TMotDevice aMotDevInfo[]) MK라이브러리에게 시스템에 장착된 모든 모션 장치의 정보(장치의 base address, 장치의갯수, 축의 수 등)를 알려줍니다. 그리고 모션 장치에 대한 정보를 전달받은 MK라이브러리에서는 모션 장치들을 초기화합니다.
❒ t_bool **mcGnResetDevice** (void) 모든 Motion 축(Axis)을 초기화 합니다
❒ t_i32 **mcGnGetNumAxes** (void) 현재 시스템에서 실제 하드웨어적으로 제공하는 모션제어 축(Axis) 수를 반환합니다.
❒ t_i32 **mcGnSetEmergency** (t_i32 nState) ❒ t_bool **mcGnGetEmergency** (void) mcGnSetEmergency() 함수는 소프트웨어적으로 모션컨트롤러를 Emergency 상태로 설정합니다. 비상정지(停止) 상태가 되면 모션컨트롤러는 현재 진행중인 작업을 모두 정지(停止) 합니다. 비상정지(停止)가 활성화되어 있는 동안에는 이동명령이 호출되어도 이송이 생략됩니다. mcGnGetEmergency() 함수는 Emergency의 set 상태를 체크하는 함수입니다.
❒ t_i32 **mcGnUserEmg_SetEnable** (t_bool bEnable) ❒ t_bool **mcGnUserEmg_GetEnable** (void) EMG-USER 입력을 활성화할지 설정(Set)하고 확인(Get)하는 함수입니다.
(2) 각종 설정 함수들
❒ t_i32 **mcCfgSetMioEnv** (t_i32 nAxis, t_i32 dwPropId, t_i32 dwPropVal) ❒ t_i32 **mcCfgGetMioEnv** (t_i32 nAxis, t_i32 dwPropId) 모션제어에 관련된 각종 I/O 환경을 설정하는 함수입니다. 이 함수는 여러가지의 환경을 설정하는데 공통적으로 사용되는 함수로서 dwPropId 파라미터를 통해서 어떠한 I/O 환경을 설정할 지를 결정합니다.
❒ t_i32 **mcCfgSetFilter** (t_i32 nAxis, t_bool bIsEnable) ❒ t_bool **mcCfgGetFilter** (t_i32 nAxis) 각종 I/O 신호에 필터 로직을 적용할지를 설정하는 함수입니다
❒ t_i32 **mcCfgSetFilterAB** (t_i32 nAxis, t_i32 nTarget, t_bool bIsEnable) ❒ t_bool **mcCfgGetFilterAB** (t_i32 nAxis, t_i32 nTarget) EA/EB 및 PA/PB 신호에 필터 로직을 적용할지를 설정하는 함수입니다
❒ t_i32 **mcCfgSetMaxPPS** (t_i32 nAxis, t_f32 fMaxPPS) ❒ t_f32 **mcCfgGetMaxPPS** (t_i32 nAxis) mcCfgSetMaxPPS() 함수는 지령펄스(command pulse)의 최대 주파수를 설정합니다. 이 함

수는 결과적으로 모션에 적용되는 최고 속도를 제한하게 됩니다. 출력 펄스의 주파수는 최대 6.5MHz까지 설정가능하며 기본적으로 설정되는 주파수 범위는 10Hz ~ 655,350Hz입니다. 최저속도는 최대속도 설정에 따라서 자동으로 결정됩니다.

mcCfgGetMaxPPS() 함수는 지령펄스의 최대 주파수 설정값을 읽어들이는 함수입니다

❒ t_i32 **mcCfgSetLogicDist** (t_i32 nAxis, t_f32 fPPUD)

❒ t_f32 **mcCfgGetLogicDist** (t_i32 nAxis)

mcCfgSetLogicDist () 함수는 논리적 단위 거리에 대한 펄스 수를 설정합니다. 여기서 논리적 단위 거리라 함은 Move함수에서 사용하는 거리 또는 위치에 대한 단위량을 의미합니다. 이 함수를 사용하여 특별히 지정하지 않는 경우에는 논리적 단위 거리에 대한 펄스 수는 초기 값인 '1' 로 사용됩니다.

mcCfgGetLogicDist () 함수는 논리적 단위 거리에 대한 펄스수 설정값을 읽어들이는 함수입니다.

❒ t_i32 **mcCfgSetLogicSpeed** (t_i32 nAxis, t3_f32 fPPUS)

❒ t_f32 **mcCfgGetLogicSpeed** (t_i32 nAxis)

mcCfgSetLogicSpeed () 함수는 논리적 단위 속도에 대한 실제 펄스 출력속도(PPS)를 설정합니다. 여기서 논리적 단위 속도라 함은 속도 지정함수에서 사용하는 속도 또는 가속도에 대한 단위량을 의미합니다. 이 함수를 사용하여 특별히 지정하지 않는 경우에는 단위 속도에 대한 펄스 출력속도는 1 PPS로 사용됩니다.

mcCfgGetLogicSpeed () 함수는 논리적 단위 속도에 대한 현재 설정값을 읽어들이는 함수입니다.

❒ t_i32 **mcCfgSetIniSpeed** (t_i32 nAxis, t_f32 fIniSpeed)

❒ t_f32 **mcCfgGetIniSpee3d** (t_i32 nAxis)

mcCfgSetIniSpeed () 함수는 지정한 축에 대한 초기속도를 설정합니다. 특별한 경우가 아니면 이 함수를 사용할 필요가 없으며 기본적으로 초기속도는 0으로 설정됩니다.

mcCfgGetIniSpeed () 함수는 지정한 축에 대한 초기속도의 현재 설정값을 읽어들이는 함수입니다.

❒ t_i32 **mcCfgSetSoftLimitRange** (t_i32 nAxis, t_f32 fNegLim, t_f32 fPosLim)

❒ void **mcCfgGetSoftLimitRange** (t_i32 nAxis, t_f32 *pfNegLim, t_f32 *pfPosLim)

mcCfgSetSoftLimitRange () 함수는 지정한 축의 소프트웨어 리미트 값을 설정합니다.

mcCfgGetSoftLimitRange () 함수는 지정한 축의 현재 설정된 소프트웨어 리미트 값을 읽어들이는 함수입니다.

❒ t_i32 **mcCfgSetCorrection** (t_i32 nAxis, t_i32 nCorrMode, t_f32 fCorrAmount, t_f32 fCorrVel, t_i32 dwCntrMask)

❒ t_i32 **mcCfgGetCorrection** (t_i32 nAxis, t_i32 *pnCorrMode, t_f32 *pfCorrAmount, t_f32 *pfCorrVel, t_i32 *pdwCntrMask)

mcCfgSetCorrection () 함수는 함수는 백래쉬(Backlash) 또는 슬립(Slip)에 대한 보정을 설정하는 함수입니다. 구조적으로 백래쉬나 슬립 현상이 심하게 일어나는 경우에는 이에 대한 보정이 필요할 수 있습니다.

mcCfgGetCorrection () 함수는 설정된 보정 정보를 얻어오는 함수입니다.

❒ t_i32 **mcCfgSetRingCntr** (t_i32 nAxis, t_i32 nTargCntr, t_i32 bIsEnable, t_f32 fMaxPos)

❒ t_i32 **mcCfgGetRingCntr** (t_i32 nAxis, t_i32 nTargCntr, t_i32* pbIsEnable, t_f32* pfMaxPos)

Ring Counter 기능 사용 여부 및 카운터 최대값을 설정합니다.

(3) 원점 복귀 관련 함수들

❒ t_i32 **mcHomeSetConfig** (t_i32 nAxis, t_i32 nHomeMode, t_i32 nEzCnt, t_f32 fEscDist)

❒ t_bool **mcHomeGetConfig** (t_i32 nAxis, t_i32* pnHomeMode, t_i32* pnEzCnt, t_f32*

pfEscDist)

mcHomeSetConfig () 함수는 원점복귀에 관련된 여러 가지 환경을 설정합니다.

mcHomeGetConfig () 함수는 현재 설정되어 있는 여러 가지 원점복귀 환경 설정 값을 읽어옵니다.

❒ t_i32 **mcHomeSetConfigEx** (t_i32 nAxis, t_i32 nHomeMode, t_i32 nDir, t_i32 nEzCnt, t_f32 fEscDist, t_f32 fOffset)

❒ t_i32 **mcHomeGetConfigEx** (t_i32 nAxis, t_i32* pnHomeMode, t_i32 *pnDir, t_i32* pnEzCnt, t_f32* pfEscDist, t_f32* pfOffset)

원점 복귀에 관련된 여러 가지 환경을 설정합니다. 기본 설정함수에 비해 원점복귀 방향 및 오프셋 이동거리값을 추가로 설정할 수 있습니다.

❒ t_i32 **mcHomeSetPosClrMode** (t_i32 nAxis, t_i32 nPosClrMode)

❒ t_i32 **mcHomeGetPosClrMode** (t_i32 nAxis)

원점복귀 완료 후에 Command 및 Feedback 위치를 재설정하는 함수입니다.

❒ t_i32 **mcHomeSetSpeedPattern** (t_i32 nAxis, t_i32 nVMode, t_f32 fVel, t_f32 fAcc, t_f32 fDec, t_f32 fRvsVel)

❒ void **mcHomeGetSpeedPattern** (t_i32 nAxis, t_i32* pnVMode, t_f32* pfVel, t_f32* pfAcc, t_f32* pfDec, t_f32* pfRvsVel)

mcHomeSetSpeedPattern () 함수는 원점복귀 작업시의 속도 패턴을 설정합니다.

mcHomeGetSpeedPattern () 함수는 현재 설정되어 있는 원점복귀 속도패턴을 읽어옵니다.

❒ t_bool **mcHomeMove** (t_i32 Axis, t_bool bWaitDone)

원점복귀 이송을 수행합니다

❒ t_bool **mcHomeIsBusy** (t_i32 nAxis)

현재 원점복귀가 진행중인지 확인을 하는 함수입니다.

❒ t_i32 **mcHomeWaitDone** (t_i32 nAxis)

해당축의 원점복귀가 완료될 때까지 기다리는 함수입니다.

❒ void **mcHomeSetSuccess** (t_i32 nAxis, t_bool bSuccess)

❒ t_bool **mcHomeGetSuccess** (t_i32 nAxis)

mcHomeSetSuccess () 이 함수가 호출되기 이전에 원점복귀가 성공적으로 완료되었는지를 알려주는 함수입니다.

mcCfgGetLogicDist () 함수는 원점복귀의 성공여부에 대한 플래그 값을 강제로 설정하는 함수입니다. 일반적으로는 이 플래그 값은 원점복귀의 실제 수행에 의해서 셋팅됩니다. 그러나 필요한 경우에 강제로 그 값을 셋(Set) 또는 리셋(Reset)할 수 있습니다.

(4) 단축 구동 관련 함수들

❒ t_i32 **mcSxSetSvon** (t_i32 nAxis, t_i32 nState)

❒ t_bool **mcSxGetSvon** (t_i32 nAxis)

mcSxSetSvon () 함수는 어느 한 축(Single axis)의 서보온(Servo-ON) 신호의 출력 상태를 제어합니다.

mcSxGetSvon () 함수는 어느 한 축의 현재 출력되고 있는 서보온 신호의 출력 상태를 읽어들입니다.

❒ t_i32 **mcSxSetAlmRst** (t_i32 nAxis, t_i32 nState)

❒ t_bool **mcSxGetAlmRst** (t_i32 nAxis)

mcSxSetAlmRst () 함수는 어느 한 축(Single axis)의 알람리셋(Alarm-reset) 신호의 출력 상태를 제어합니다. 알람리셋 신호는 서보드라이버의 알람 상태를 클리어(clear) 해주는 신호

입니다.

mcSxGetAlmRst () 어느 한 축의 현재 출력되고 있는 알람리셋(Alarm-reset) 신호의 출력 상태를 읽어들입니다.

❒ t_i32 **mcSxSetSpeedPattern** (t_i32 nAxis, t_i32 nVMode, t_f32 fVel, t_f32 fAcc, t_f32 fDec)

❒ void **mcSxGetSpeedPattern** (t_i32 nAxis, t_i32* pnVMode, t_f32* pfVel, t_f32* pfAcc, t_f32* pfDec)

mcSxSetSpeedPattern () 함수는 지정한 축의 정격속도 패턴을 설정합니다.

mcSxGetSpeedPattern () 함수는 지정한 축의 현재의 정격속도패턴 설정을 읽어들입니다.

❒ t_i32 **mcSxSetSpeedRatio** (t_i32 nAxis, t_f32 fVelR, t_f32 fAccR, t_f32 fDecR)

❒ void **mcSxGetSpeedRatio** (t_i32 nAxis, t_f32* pfVelR, t_f32* pfAccR, t_f32* pfDecR)

mcSxSetSpeedRatio () 함수는 지정한 축의 속도비(速度比, speed ratio)를 설정하는 함수입니다.

mcSxGetSpeedRatio () 함수는 지정한 축의 속도비 설정을 읽어들입니다.

❒ t_i32 **mcSxJog** (t_i32 nAxis, t_i32 nDir)

mcSxJog () 함수는 어느 한축에 대하여 조그(Jog) 이송을 시작시키는 함수입니다. 여기서 조그 이송이란 정지 함수가 호출되기 전까지 지정한 속도패턴을 유지하며 어느 한 방향으로 이송을 수행하는 것을 의미합니다. 이때 이송의 목표좌표는 없으며 정지 명령이 하달되기 전까지 계속해서 이송하게 됩니다.

이 함수는 이송을 시작 시킨 후에 바로 반환됩니다.

❒ t_i32 **mcSxMove** (t_i32 nAxis, t_f32 fDistance, t_bool bWaitDone)

하나의 축에 대하여 현재의 위치에서 지정한 거리(상대 위치)만큼 이동을 수행합니다.

❒ t_i32 **mcSxMoveTo** (t_i32 nAxis, t_f32 fPosition, t_bool bWaitDone)

하나의 축에 대하여 지정한 절대좌표로의 이송을 수행합니다.

❒ t_i32 **mcSxMove_2Vel** (t_i32 nAxis, t_f32 fDistance, t_f32 fVel2, t_bool bWaitDone)

2단계 가감속도 변경 값을 설정해서 구동하는 함수입니다. 상대좌표 이송입니다.

❒ t_i32 **mcSxMoveTo_2Vel** (t_i32 nAxis, t_f32 fPosition, t_f32 fVel2, t_bool bWaitDone)

2단계 가감속도 변경 값을 설정해서 구동하는 함수입니다. 절대좌표 이송입니다.

❒ t_bool **mcSxIsDone** (t_i32 nAxis)

지정한 축에 대한 이송을 완료하였는지(정지 상태에 있는지)를 알려주는 함수입니다

❒ t_i32 **mcSxSto**❒ t_bool **p** (t_i32 nAxis, t_bool bDecel, t_bool bWaitDone)

이송을 정지시키는 명령입니다

❒ t_f32 **mcSxGetTargPos** (t_i32 nAxis)

단축구동시 이동 거리에 해당하는 목표 좌표값을 반환하는 함수입니다.

(5) 다축 구동 관련 함수들

❒ t_i32 **mcMxMove** (t_i32 nNumAxes, t_i32 anAxes[], t_f32 afDistance[], t_bool bWaitDone)

여러 개의 축에 대하여 현재의 위치에서 지정한 거리만큼 이동을 동시에 시작합니다

❒ t_i32 **mcMxMoveTo** (t_i32 nNumAxes, t_i32 anAxes[], t_f32 afPosition[], t_bool bWaitDone)

절대치 다축구동 함수입니다.

❒ t_i32 **mcMxIsDone** (t_i32 nNumAxes, t_i32 anAxes[])
여러 개의 축에 대하여 지정한 모든 축의 모션이 완료됐는지를 체크합니다

❒ t_bool **mcMxIsDone_Mask** (t_ui32 dwAxisMask)
축(Axis) 마스크를 이용한 다축구동 완료 체크 함수입니다.

❒ t_i32 **mcMxStop** (t_i32 nNumAxes, t_i32 anAxes[], t_bool bDecel, t_bool bWaitDone)
다축구동을 정지시키는 함수입니다.

(6) 보간 제어 관련 함수들

❒ t_i32 **mcIxSetAxisMap** (t_i32 nMapIndex, t_ui32 dwMapMask)
보간제어를 위한 맵인덱스 설정 및 보간제어에 포함시킬 축(Axis) 마스크를 설정합니다.

❒ t_ui32 **mcIxGetAxisMap** (t_i32 nMapIndex)
보간제어의 설정된 맵인덱스를 통해 보간제어에 포함시킬 축(Axis) 마스크를 반환합니다

❒ t_i32 **mcIxSetVelCorrMode** (t_i32 nMapIndex, t_i32 nVelCorrOpt1, t_i32 nVelCorrOpt2)
보간제어 속도 보정을 위한 옵션을 설정합니다.

❒ t_i32 **mcIxGetVelCorrMode** (t_i32 nMapIndex, t_i32* pnVelCorrOpt1, t_i32* pnVelCorrOpt2)
보간제어 속도 보정을 위해 설정된 옵션을 읽어옵니다.

❒ t_i32 **mcIxSetSpeedPattern** (t_i32 nMapIndex, t_i32 bIsVectorSpeed, t_i32 nSpeedMode, t_f32 fVel, t_f32 fAcc, t_f32 fDec)
❒ void **mcIxGetSpeedPattern** (t_i32 nMapIndex, t_i32 *pbIsVectorSpeed, t_i32 *pnSpeedMode, t_f32 *pfVel, t_f32 *pfAcc, t_f32 *pfDec)
보간제어시 속도 패턴을 설정합니다.

❒ t_i32 **mcIxLine** (t_i32 nMapIndex, t_f32 afDistList[], t_bool bWaitDone, t_bool bIsBatchMode)
직선 보간제어를 시작합니다. 상대좌표 이송입니다.

❒ t_i32 **mcIxLineTo** (t_i32 nMapIndex, t_f32 afPosList[], t_bool bWaitDone, t_bool bIsBatchMode)
직선 보간제어를 시작합니다. 절대좌표 이송입니다.

❒ t_i32 **mcIxArc** (t_i32 nMapIndex, t_float32 fCenPos[], t_float32 fEndPos[], t_i32 nDir, t_bool bWaitDone)
중심점과 끝점을 이용해서 원호보간 이송을 수행합니다. 상대좌표 이송입니다.

❒ t_i32 **mcIxArcTo** (t_i32 nMapIndex, t_float32 fCenPos[], t_float32 fEndPos[], t_i32 nDir, t_bool bWaitDone)
중심점과 끝점을 이용해서 원호보간 이송을 수행합니다. 절대좌표 이송입니다.

❒ t_i32 **mcIxArcA** (t_i32 nMapIndex, t_f32 afCenPos[], t_f32 fAngle, BOOL bIsAbsPos, t_bool bWaitDone)
중심점과 각도를 이용해서 원호보간 이송을 수행합니다. 상대좌표 이송입니다.

❒ t_i32 **mcIxArc3P** (t_i32 nMapIndex, t_f32 P2[], t_f32 P3[], t_f32 fAngle, BOOL bIsAbsPos, t_bool bWaitDone)
세점을 이용해서 원호보간 이송을 수행합니다.

❒ t_i32 **mcIxSpline** (t_i32 nMapIndex, t_float32 **PP, t_i32 n_inp, t_i32 n_div)

스플라인 보간제어 이송을 수행합니다.

❒ t_bool **mcIxIsDone** (t_i32 nMapIndex)
보간제어가 완료되었는지 체크하는 함수입니다.

❒ t_i32 **mcIxStop** (t_i32 nMapIndex, t_bool bDecel, t_bool bWaitDone)
보간제어 이송을 정지시키는 함수입니다.

(7) MPG (매뉴얼 펄사) 구동 관련 함수들

❒ t_i32 **mcPlsrSetInMode** (t_i32 nAxis, t_i32 nInputMode, t_bool bIsRevDir)
❒ t_i32 **mcPlsrGetInMode** (t_i32 nAxis, t_i32* pnInputMode, t_bool* pbIsRevDir)
Manual Pulse 입력 신호에 대한 환경을 설정합니다.

❒ t_i32 **mcPlsrSetGain** (t_i32 nAxis, t_i32 nGainFactor, t_i32 nDivFactor)
❒ t_i32 **mcPlsrGetGain** (t_i32 nAxis, t_i32* pnGainFactor, t_i32* pnDivFactor)
PA/PB 입력 펄스 대비 Command 출력 펄스 수의 비를 사용자가 임의로 조절할 수 있도록 하는 함수입니다

❒ t_i32 **mcPlsrHomeMoveStart** (t_i32 nAxis, t_i32 nHomeType)
Manual Pulse 구동에 의한 원점복귀 이송명령을 수행합니다.

❒ t_i32 **mcPlsrJogStart** (t_i32 nAxis)
Manual Pulse 구동에 의한 속도 이송명령을 수행합니다.

❒ t_i32 **mcPlsrMove** (t_i32 nAxis, t_f32 fDistance, t_bool bWaitDone)
Manual Pulse 구동에 의한 상대좌표 이송명령을 수행합니다.

❒ t_i32 **mcPlsrMoveTo** (t_i32 nAxis, t_f32 fPosition, t_bool bWaitDone)
Manual Pulse 구동에 의한 절대좌표 이송명령을 수행합니다.

❒ t_bool **mcPlsrIsActive** (t_i32 nAxis)
PA/PB를 사용하는 이송모드가 설정되어 있는지 체크합니다.

(8) 속도 및 위치 오버라이드 함수들

❒ t_i32 **mcOverrideSpeedSet** (t_i32 nAxis)
해당축에 속도오버라이드 기능을 설정합니다.

❒ t_i32 **mcOverrideMove** (t_i32 nAxis, t_f32 fNewDist, t_bool bIsHardApply, t_i32 *pnState)
해당축에 오버라이드된 새로운 속도로 상대좌표 이송을 수행합니다.

❒ t_i32 **mcOverrideMoveTo** (t_i32 nAxis, t_f32 fNewPos, t_bool bIsHardApply, t_i32 *pnState)
해당축에 오버라이드된 새로운 속도로 절대좌표 이송을 수행합니다.

(9) 각종 상태 관련 함수들

❒ t_i32 **mcStSetCount** (t_i32 nAxis, t_i32 nSource, t_i32 nCount)
❒ t_i32 **mcStGetCount** (t_i32 nAxis, t_i32 nSource)
mcStSetCount() 함수는 지정한 축의 지령펄스(command pulse) 또는 궤환펄스(feedback pulse)의 카운트를 임의의 값으로 설정하는 함수입니다. 이때 지정하는 카운터값의 단위는

펄스수입니다.

mcStGetCount() 함수는 지정한 축의 지령펄스(command pulse) 또는 궤환펄스(feedback pulse)의 카운트값을 읽어들이는 함수입니다. 이때 카운트의 값은 펄스수입니다.

❒ t_i32 **mcStSetPosition** (t_i32 nAxis, t_i32 nSource, t_f32 fPosition)
❒ t_f32 **mcStGetPosition** (t_i32 nAxis, t_i32 nSource)

mcStSetPosition() 함수는 지정한 축의 지령위치(command position) 또는 궤환위치(feedback position) 값을 임의의 값으로 설정하는 함수입니다. 이때 지정하는 위치값의 단위는 논리거리 단위를 사용합니다.

mcStGetPosition() 함수는 지정한 축의 지령위치(command pulse) 또는 궤환위치(feedback pulse)의 현재값을 읽어들이는 함수입니다. 이때 위치값의 단위는 논리거리 단위입니다.

❒ t_f32 **mcStGetSpeed** (t_i32 nAxis, t_i32 nSource)

mcStGetSpeed() 함수는 현재 실제로 이송되고 있는 속도를 반환합니다. nSource 파라미터를 통해서 지령 속도와 궤환 속도를 모두 이 함수를 통해서 읽을 수 있습니다

❒ t_i32 **mcStGetMst** (t_i32 nAxis)

현재 모션의 동작 상태를 반환합니다

❒ t_i32 **mcStGetMio** (t_i32 nAxis)

현재 모션과 관련된 여러가지 I/O 상태를 나타내는 32비트 정수값을반환합니다. 이 값은 각 비트별로 할당된 I/O핀의 상태를 표시하므로 사용자는 비트마스크를 수행하여 원하는 I/O핀의 상태를 확인하여야 합니다.

(10) 에러 관련 함수들

❒ t_i32 **mcErrClearAxisError** (t_i32 nAxis)

에러코드 변수를 클리어합니다.

❒ t_i32 **mcErrGetAxisError** (t_i32 nAxis)

각 축에 모션 에러가 발생했는지 체크하는 함수입니다

❒ t_i32 **mcErrGetLastError** (void)

마지막에 발생한 에러코드를 반환합니다.

❒ void **mcErrGetErrorString** (t_i32 nErrCode, t_char* szBuffer)

에러코드에 해당하는 에러메시지를 얻는 함수입니다.

❒ void **mcErrClearLastError** (void)

마지막에 발생한 에러코드를 '에러없음' 으로 초기화합니다.

(11) 인터럽트 관련 함수들

❒ t_i32 **mcIntSetMask** (t_i32 nAxis, t_ui32 dwMask)
❒ t_ui32 **mcIntGetMask** (t_i32 nAxis)

인터럽트 마스크를 설정하고 얻어옵니다.

❒ void **mcISR** (void)

시스템 함수로서 모션장치의 인터럽트 서비스 루틴(t_i32errupt service routine, ISR)에서 처리해야 하는 작업들이 내장되어 있는 함수입니다.

사용자는 반드시 t_i32errupt #4의 서비스 루틴 함수인 c_t_i3204() 에서 이 함수를 호출해주어야 합니다

❒ void **mcTimerCallback** (void)

타이머 인터럽트 서비스루틴에서 처리해주어야 하는 모션제어 모듈 관련 작업들이 내장되어 있는 함수입니다.
사용자는 반드시 1ms 주기를 가지는 타이머 인터럽트 서비스 루틴(c_t_i3209())에서 이 함수를 호출해주어야 합니다

(12) Latch counter 관련 함수들

❒ t_bool **mcLtcIsLatched** (t_i32 nAxis)
래치카운터에 값이 들어왔는지 확인합니다.

❒ t_f32 **mcLtcReadLatch** (t_i32 nAxis, t_i16 nCounter)
래치카운터의 종류에 따라 카운터 값을 읽습니다.

❒ t_i32 **mcLtcQue_Alloc**(t_i32 nAxis, t_i16 nCounter)
Latch-que 버퍼 할당.

❒ t_i32 **mcLtcQue_Free**(t_i32 nAxis, t_i16 nCounter)
Latch-que 버퍼 해제.

❒ t_ui32 **mcLtcQue_GetSize**(t_i32 nAxis, t_i16 nCounter)
Latch-que 버퍼 크기 반환.

❒ t_i32 **mcLtcQue_Reset**(t_i32 nAxis, t_i16 nCounter)
Que의 push & pop count를 모두 리셋합니다.

❒ t_ui32 **mcLtcQue_Check**(t_i32 nAxis, t_i16 nCounter)
Que에 읽혀지지 않은 데이터가 몇 개 남아 있는지를 알아보는 함수.

❒ t_f32 **mcLtcQue_Pop**(t_i32 nAxis, t_i16 nCounter)
Que에서 읽혀지지 않은 데이터를 반환한다. 이때 pop-count는 자동으로 증가한다.

❒ t_f32 **mcLtcQue_GetAt**(t_i32 nAxis, t_i16 nCounter)
지정된 인덱스에 해당하는 Latch Que 버퍼의 데이터를 반환한다.

(13) 위치비교 출력 기능 관련 함수들

❒ t_i32 **mcCmpErr_Set** (t_i32 nAxis, t_f32 fCmpData, t_i32 nCmpAction)
error comparator 환경을 설정합니다.

❒ t_i32 **mcCmpErr_Get** (t_i32 nAxis, t_f32* pfCmpData, t_i32* pnCmpAction)
설정된 error comparator 환경 값을 읽어옵니다.

❒ t_i32 **mcCmpGen_Set** (t_i32 nAxis, t_i32 nCmpSrc, t_i32 nCmpMethod, t_f32 fCmpData, t_i32 nCmpAction)
general comparator 환경을 설정합니다.

❒ t_i32 **mcCmpGen_Get** (t_i32 nAxis, t_i32 *pnCmpSrc, t_i32 *pnCmpMethod, t_f32 *pfCmpData, t_i32 *pnCmpAction)
설정된 general comparator 환경 값을 읽어옵니다.

❒ t_i32 **mcCmpTrg_SetCfg** (t_i32 nAxis, t_i32 nCmpSrc, t_i32 nCmpMethod)
위치비교 출력 대상카운터 소스 종류 및 비교방법을 설정합니다.

❒ t_i32 **mcCmpTrg_GetCfg** (t_i32 nAxis, t_i32 *pnCmpSrc, t_i32 *pnCmpMethod)

설정된 위치비교 출력 대상카운터 소스 종류 및 비교방법을 읽어옵니다.
❒ t_i32 **mcCmpTrg_SetData** (t_i32 nAxis, t_f32 fCmpData) 위치비교 출력을 위한 비교데이터를 설정합니다.
❒ t_i32 **mcCmpTrg_GetData** (t_i32 nAxis, t_f32 *pfCmpData) 설정된 위치비교 출력을 위한 비교데이터를 읽어옵니다.
(14) 마스터 슬레이브 동기 구동 함수
❒ t_i32 **mcMs_RegSlave**(t_i32 nAxis, t_f32 fMaxSpeed, t_bool bIsRevDir) 마스터 슬레이브 구동을 위한 슬레이브 축을 등록합니다.
❒ t_i32 **mcMs_UnregSlave**(t_i32 nAxis) 마스터 슬레이브 구동에 대한 슬레이브 축을 해제합니다.
❒ t_i32 **mcMs_CheckSlaveState**(t_i32 nAxis) 슬레이브 축의 등록 상태를 체크합니다
❒ t_i32 **mcMs_GetMaster**(t_i32 nAxis) 슬레이브 축에 대한 마스터 축을 얻습니다
(15) 고급 확장 함수들
❒ t_i32 **mcAdvGetNumDevices** (void) 모션 모듈의 개수를 얻어오는 함수입니다.
❒ t_bool **mcAdvGetMotDevInfo** (t_i32 nDevIdx, TMotDevice* pMotDevBuf) 모션 모듈의 각종 정보를 얻어오는 함수입니다.
❒ t_i32 **mcAdvStaTrigger** (t_i32 nAxis) STA 트리거 신호를 출력하는 함수입니다.
❒ t_i32 **mcAdvErcOut** (t_i32 nAxis) ERC 신호를 출력하는 함수입니다.
❒ t_i32 **mcAdvErcReset** (t_i32 nAxis) ERC RESET 명령을 수행합니다.
(16) 범용 I/O 함수들
❒ t_bool **siGetOne**(t_ui32 nChannel) 단일 채널의 입력 센서 상태를 얻어오는 함수입니다.
❒ t_i32 **siGetMulti**(t_ui32 nIniChan, t_ui32 nNumChan) 여러 채널의 입력 센서 상태를 얻어오는 함수입니다.
❒ void **soPutOne**(t_ui32 nChannel, t_bool bState) 단일 채널을 통해 디지털 신호를 출력하는 함수입니다.
❒ t_bool **soGetOne**(t_ui32 nChannel) 단일 채널을 통해 출력된 디지털 신호를 확인하는 함수입니다.

❒ void **soPutMulti**(t_ui32 nIniChan, t_ui32 nNumChan, t_ui32 dwStates) 여러 채널을 통해 디지털 신호를 출력하는 함수입니다.
❒ t_i32 **soGetMulti**(t_ui32 nIniChan, t_ui32 nNumChan) 여러 채널을 통해 출력된 디지털 신호를 확인하는 함수입니다.

▣ mcBootup

함수 원형

void **mcBootup** ()

함수 설명

모션 장치 초기화 함수입니다. 이 함수는 시스템 부팅함수인 mkcBootup() 함수내부에서 호출되어져야 합니다.

매개 변수

▶ 없음

예제

□ 기본적으로 mk_extern.c 소스파일에 있는 mkcBootup() 함수 내부에서 아래와 같이 mcBootup() 함수를 수행하도록 되어 있습니다.

```
void mkcBootup(void)
{
    flsBootup(); // 전역 변수로 플래쉬 영역의 주소를 할당
    mcBootup(); // 모션장치 부팅
    dioBootup(); // 디지털 I/O 장치 부팅
    aioBootup(); // 아날로그 I/O 장치 부팅
    commBootup(); // Common communication 초기화
}
```

▣ mcGnInitMotion

함수 원형

t_i32 **mcGnInitMotion** (t_i32 nNumDevs, TMotDevice aMotDevInfo[])

함수 설명

MK라이브러리에게 시스템에 장착된 모든 모션 장치의 정보(장치의 base address, 장치의 갯수, 축의 수 등)를 알려줍니다. 그리고 모션 장치에 대한 정보를 전달받은 MK라이브러리에서는 모션 장치들을 초기화합니다.

매개 변수

▶ **nNumDevs:** 모션 모듈 장치의 수를 지정합니다. miCUBE-E4 장치는 모션 모듈이 하나입니다. 그리고 miCUBE-E6 장치는 외형적으로는 1개의 보드로 구성되었지만 실제 모션칩은 4축짜리 하나와 2축짜리 하나의 두개로 이루어져 있습니다. ceMC02P 는 2축이 하나의 모듈입니다.

▶ **aMotDevInfo:** 각 모듈의 장치 정보를 담고있는 TMotDevice 형의 구조체입니다. 사용자는 이 구조체에 각 모듈의 정보를 설정하여 MK라이브러리에게 전달하여야 합니다. 구조체는 모듈의 수에 해당하는 크기의 배열로 구성되어야 합니다. TMotDevice 구조체는 다음과 같은 구조를 가집니다.

```
typedef struct{
    HANDLE hDevice; // 장치의 base address 를 설정합니다.
    t_i32 nNumAxes; // 해당 모듈이 제공하는 축의 수를 설정합니다.
    t_i32 nAxisIni; // 해당 모듈의 첫번째 축의 전체 통합 축 번호상의 축 번호를 설정
    struct{
        t_ui32 Id: 4; // Device ID (4 bit 로 구성)
        t_ui32 Ver:          4; // H/W Version (4 bit 로 구성)
        t_ui32 Rsv: 24; // 사용되지 않음
    }u;
}TMotDevice;
```

반환값

전체 모듈이 제공하는 축의 총합을 반환합니다.

참고

□ 이 함수는 mk_extern.c 소스파일에 있는 **mcBootup()** 라고 하는 내부 사용 함수에서 호출됩니다. 참고로 **mcBootup()** 함수는 **mkcBootup()** 함수에서 호출됩니다.
mcBootup() 함수는 시스템에 장착된 모션 장치를 자동으로 열거하여 각 장치를 초기화하는 역할을 하는 함수입니다. 그 함수의 구성은 아래 예제에서 설명드리겠습니다.

예제

```
void mkcBootup(void)
{
    t_ui32 i, nBaseAddr, nDevId, nSerDevCnt=0;
    t_uchar auAddr[4]={192,168,1,100};

    flsBootup();
    mcBootup(); // 모션장치 부팅
    …
}

/------------/-----------------------------------------------------------------------------------------------------
// mcBootup(): 모션 장치 부팅 함수.
// *. 이 함수는 내부적으로만 사용되는 함수이다.
//-----------------------------------------------------------------------------------------------------------------
void mcBootup(void)
{
        TMotDevice aMotDev[16];
        t_i32 nBaseAddr, nDevId;
        t_i32 dc=0, ac=0;

        #if _PLATFORM == PLF_miCUBE_E   // miCUBE-E platform 인 경우.
        nBaseAddr = 0xfe0000; // miCUBE-E 시스템은 모션 모듈의 base address 는
무조건 0xfe0000 으로 시작함
        nDevId = 0xf & Inp32(nBaseAddr, 0x80); // 장착된 모션 모듈의 종류를
알아내기 위해서 장치 아이디를 읽는다(base address + 0x80).
        aMotDev[0].hDevice = nBaseAddr;
        aMotDev[0].nAxisIni = 0;
        aMotDev[0].u.Id = nDevId;
        if(nDevId == 2){ // 2 축 모션모듈
            aMotDev[0].nNumAxes = 2;
            aMotDev[0].u.Ver = (nDevId>>4)&0xf;
            ac = 2;                 dc = 1;
        }else if(nDevId == 4){ // 4 축 모션모듈
            aMotDev[0].nNumAxes = 4;
            aMotDev[0].u.Ver = (nDevId>>4)&0xf;
            ac = 4;                 dc = 1;
        }else if(nDevId == 6){ // 6 축 모션모듈(4 축 + 2 축의 두개 모듈로 구성)
            aMotDev[0].nNumAxes = 4;
            aMotDev[0].u.Ver = (nDevId>>4)&0xf;
            // 2nd 모듈(2 축짜리) 정보 설정 //
            aMotDev[1].hDevice = 0xfe0200;
            aMotDev[1].nAxisIni = 4;
            aMotDev[1].u.Id = 2;
            aMotDev[1].nNumAxes = 2;
            aMotDev[1].u.Ver = (nDevId>>4)&0xf;
            ac = 6;
            dc = 2;
        }
        #endif
        mcGnInitMotion (dc, aMotDev);
}
```

▣ mcGnResetDevice

함수 원형

t_bool **mcGnResetDevice** (void)

함수 설명

모든 Motion 축(Axis)을 초기화 합니다

매개 변수

▶ 없음

반환값

성공(TRUE) 또는 실패(FALSE)

참고

□ 동일 모션 모듈내의 전체 축(Axis) 에 칩 리셋이 적용됩니다.

▣ mcGnGetNumAxes

함수 원형

t_i32 **mcGnGetNumAxes** (void)

함수 설명

현재 시스템에서 실제 하드웨어적으로 제공하는 모션제어 축(Axis) 수를 반환합니다.

반환값

현재 시스템에서 실제 하드웨어적으로 제공하는 모션제어 축(Axis) 수를 반환합니다.

예제

```
t_bool bDecel = cmTRUE;
for(i=0; i<mcGnGetNumAxes(); i++){
        mcSxStop(i, bDecel, cmFALSE);
}
```

▣ mcGnSetEmergency / mcGnGetEmergency

함수 원형

t_i32 **mcGnSetEmergency** (t_i32 nState)

t_bool **mcGnGetEmergency** (void)

함수 설명

mcGnSetEmergency() 함수는 소프트웨어적으로 모션컨트롤러를 Emergency 상태로 설정합니다. 비상정지(停止) 상태가 되면 모션컨트롤러는 현재 진행중인 작업을 모두 정지(停止) 합니다. 비상정지(停止)가 활성화되어 있는 동안에는 이동명령이 호출되어도 이송이 생략됩니다.

mcGnGetEmergency() 함수는 Emergency의 set 상태를 체크하는 함수입니다.

매개 변수

▶ **nState:** 제어하고자 하는 Emergency set 상태를 설정합니다.

반환값

□ **mcGnSetEmergency()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

□ **mcGnGetEmergency()** 함수는 Emergency의 set 상태를 반환합니다.

Value	Meaning
0	Emergency가 set 되지 않은 상태(normal 상태)
1	Emergency가 set 된 상태 (모든 이송 명령이 무시됩니다)

예제

```
#define MAX_N_AXES 6
void InitMotionEnv(void)
{
        t_i32 i, nPosition;
        t_f32 afIniVel[MAX_N_AXES] = {0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
        t_f32 fNumRev;
        t_i32 nPos_cmd, nPos_feed;
        if(mcGnGetNumAxes() < 1)
                return;
        mcGnSetEmergency(cmFALSE);
}
```

▣ mcGnUserEmg_SetEnable / mcGnUserEmg_GetEnable

함수 원형

t_i32 **mcGnUserEmg_SetEnable** (t_bool bEnable)

t_bool **mcGnUserEmg_GetEnable** (void)

함수 설명

EMG-USER 입력을 활성화할지 설정(Set)하고 확인(Get)하는 함수입니다.

매개 변수

▶ **bEnable:** EMG-USER 입력 활성화 여부

반환값

□ **mcGnUserEmg_SetEnable ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

□ **mcGnUserEmg_GetEnable ()** 함수는 Emergency의 set 상태를 반환합니다.

Value	Meaning
0(FALSE)	EMG-USER 입력이 활성화 되지 않은 상태
1(TRUE)	EMG-USER 입력이 활성화 된 상태

▣ mcCfgSetMioEnv / mcCfgGetMioEnv

함수 원형

t_i32 **mcCfgSetMioEnv** (t_i32 nAxis, t_i32 dwPropId, t_i32 dwPropVal)

t_i32 **mcCfgGetMioEnv** (t_i32 nAxis, t_i32 dwPropId)

함수 설명

모션제어에 관련된 각종 I/O 환경을 설정하는 함수입니다. 이 함수는 여러가지의 환경을 설정하는데 공통적으로 사용되는 함수로서 **dwPropId** 파라미터를 통해서 어떠한 I/O 환경을 설정할 지를 결정합니다.

매개 변수

▶ **nAxis:** 대상 축 번호 (0-based)

▶ **dwPropId:** 설정하고자 하는 I/O 속성에 대한 아이디입니다. 이 속성 아이디에 대한 자세한 내용은 아래 표를 참고하십시오.

▶ **dwPropVal:** 설정하고자 하는 I/O 속성의 값입니다. 이 값의 범위는 속성 아이디에 따라서 다릅니다. 자세한 내용은 아래 표를 참고하십시오.

PropId	Meaning & PropVal
0 (cmMPID_ALM_LOGIC)	Alarm(ALM) 신호의 입력로직을 설정합니다. 설정가능한 PropVal 은 다음과 같습니다. ▪ **0 (cmLOGIC_A)** : A 접점 방식 => 평상시 Open, 감지되면 Close 되는 스위치 방식 (기본값) ▪ **1 (cmLOGIC_B)** : B 접점 방식 => 평상시 Close, 감지되면 Open 되는 스위치
1 (cmMPID_ALM_MODE)	Alarm 입력이 ON 되어 해당 축의 모션작업이 정지할때 정지되는 방식을 설정합니다. 설정가능한 PropVal 은 다음과 같습니다. ▪ **0** : 즉시정지 (기본값) ▪ **1** : 감속후 정지
2 (cmMPID_CMP_LOGIC)	**위치비교출력(CMP)** 신호의 출력방식을 설정합니다. ▪ **0** : Active low => 평상시 HIGH 상태를 유지하다가 트리거 시점에서 LOW 로 떨어졌다가 다시 HIGH 로 올라갑니다. (기본값) ▪ **1** : Active high => 평상시 LOW 상태를 유지하다가 트리거 시점에서 HIGH 로 올라갔다가 다시 LOW 로 떨어집니다.

PropId	Meaning & PropVal
3 (cmMPID_DR_LOGIC)	-/+ DR 신호의 입력로직을 설정합니다. 설정가능한 PropVal 은 다음과 같습니다. ▪ **0 (cmLOGIC_A)** : A 접점 방식 (기본값) ▪ 1 (cmLOGIC_B) : B 접점 방식
4 (cmMPID_EL_LOGIC)	-EL 과 +EL 신호의 입력로직을 설정합니다. 설정가능한 PropVal 은 다음과 같습니다. ▪ **0 (cmLOGIC_A)** : A 접점 방식 (기본값) ▪ 1 (cmLOGIC_B) : B 접점 방식
5 (cmMPID_EL_MODE)	-/+ EL 신호가 ON 되어 정시할 때 정지 방식을 설정합니다. 설정가능한 PropVal 은 다음과 같습니다. ▪ **0** : 즉시정지 (기본값) ▪ **1** : 감속후 정지
6 (cmMPID_ERC_LOGIC)	ERC 출력 신호의 출력로직을 설정합니다. 설정가능한 PropVal 은 다음과 같습니다. ▪ **0 (cmLOGIC_A)** : A 접점 방식 (기본값) ▪ 1 (cmLOGIC_B) : B 접점 방식
7 (cmMPID_ERC_OUT)	원점복귀 완료시에 ERC 출력을 내보낼지를 설정합니다. 설정가능한 PropVal 은 다음과 같습니다. ▪ **0 (FALSE)** : 원점복귀 완료시에 ERC 출력 없음. (기본값) ▪ **1 (TRUE)** : 원점복귀 완료시에 ERC 출력.
8 (cmMPID_EZ_LOGIC)	**EZ**(엔코더 Z 상) 입력 신호의 입력로직을 설정합니다. 설정가능한 PropVal 은 다음과 같습니다. ▪ **0 (cmLOGIC_A)** : A 접점 방식 (기본값) ▪ 1 (cmLOGIC_B) : B 접점 방식
9 (cmMPID_INP_EN)	**INP** 신호 입력을 활성화할 것인지를 설정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. ▪ **0 (FALSE)** : INP 비활성 (기본값) ▪ **1 (TRUE)** : INP 활성 => Command 출력이 완료되더라도 INP 신호가 ON 되기 전까지는 작업이 완료되지 않은 것으로 간주.

PropId	Meaning & PropVal
10 (cmMPID_INP_LOGIC)	**INP**(Inposition) 신호의 입력로직을 설정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. ▪ **0 (cmLOGIC_A)** : A 접점 방식 (기본값) ▪ 1 (cmLOGIC_B) : B 접점 방식
11 (cmMPID_LTC_LOGIC)	**LTC**(Latch) 신호의 입력로직을 설정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. ▪ **0 (cmLOGIC_A)** : A 접점 방식 (기본값) ▪ 1 (cmLOGIC_B) : B 접점 방식
12 (cmMPID_LTC_LTC2SRC)	LATCH COUNTER 2 의 대상카운터를 설정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. ▪ **0** : Deviation counter value (기본값) ▪ **1** : Presetn speed of command pulse
13 (cmMPID_ORG_LOGIC)	**ORG**(원점센서) 신호의 입력로직을 설정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. ▪ **0 (cmLOGIC_A)** : A 접점 방식 (기본값) ▪ 1 (cmLOGIC_B) : B 접점 방식
14 (cmMPID_SD_EN)	**SD**(Start of Deceleration) 신호의 입력을 활성 또는 비활성화 합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. ▪ **0 (FALSE)** : SD 입력을 비활성화합니다. (기본값) ▪ **1 (TRUE)** : SD 입력을 활성화합니다. 활성화되었을 때 SD 신호에 따른 동작방식은 ***cmSD_LATCH*** 와 ***cmSD_MODE*** 설정값에 의해 결정됩니다.
15 (cmMPID_SD_LOGIC)	**SD**(Start of Deceleration) 신호의 입력로직을 설정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. ▪ **0 (cmLOGIC_A)** : A 접점 방식 (기본값) ▪ 1 (cmLOGIC_B) : B 접점 방식

PropId	Meaning & PropVal
16 (cmMPID_SD_LATCH)	**SD**(Start of Deceleration) 신호를 래치(Latch)할 것인지에 대한 속성을 설정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. • **0 (FALSE)** : SD 가 ON 되어 감속중이거나 초기속도로 운전중일 때 SD 신호가 다시 OFF 상태로 변경되면 작업속도까지 다시 가속됩니다. (기본값) • **1 (TRUE)** : SD 가 ON 상태에서 OFF 상태로 바뀌어도 작업속도로 가속하지 않습니다.
17 (cmMPID_SD_MODE)	SD 신호에 따른 동작모드를 설정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. • **0** : SD 신호가 ON 되면 초기속도까지 감속합니다(정지하지 않음). (기본값) • **1** : SD 신호가 ON 되면 감속 후 정지합니다.
18 (cmMPID_STA_MODE)	Start mode 를 설정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. • **0** : STA 입력신호는 무시되며, 이동명령이 내려지면 바로 이동을 시작합니다. (기본값) • **1** : 이동명령이 내려지면 바로 시작하지 않고, STA 신호가 ON 이 되면 이동을 시작합니다.
19 (cmMPID_STA_TRG)	STA 신호가 ON 이 되는 신호의 형태를 설정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. • **0** : Level (LOW) => STA 신호가 LOW LEVEL 일때 ON (기본값) • **1** : Falling Edge => STA 입력신호가 HIGH 상태에서 LOW 상태로 천이될 때 ON
20 (cmMPID_STP_MODE)	Start mode 를 설정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. • **0** : Ignore STP => STP 입력신호 무시 (기본값) • **1** : Immediate stop => STP 입력이 ON 되면 즉시정지 • **2** : Stop after decel => STP 입력이 ON 되면 감속 후 정지

PropId	Meaning & PropVal
21 (cmMPID_CLR_CNTR)	CLR 신호가 입력되었을 때 CLEAR 되도록 할 모션컨트롤러의 카운터를 선택합니다. 이 값은 4 비트의 값으로 설정하며 각 비트는 다음과 같이 각 카운터의 클리어 여부를 설정합니다. • **Bit 0** : Command counter 의 클리어 여부를 설정 • **Bit 1** : Feedback counter 의 클리어 여부를 설정 • **Bit 2** : Deviation counter 의 클리어 여부를 설정 • **Bit 3** : General counter 의 클리어 여부를 설정
22 (cmMPID_CLR_SIGTYPE)	CLR 신호의 신호 형태를 설정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. • **0** : Falling edge => 스위치가 Open 상태에서 Close 되는 순간에 카운터가 클리어됩니다. (기본값) • **1** : Rising edge => 스위치가 Close 상태에서 Open 되는 순간에 카운터가 클리어됩니다. • **2** : Low level => 스위치가 Close 되어 있는 동안에는 계속 카운터가 클리어됩니다. • **3** : High level => 스위치가 Open 되어 있는 동안에는 계속 카운터가 클리어됩니다.
23 (cmMPID_CMP_PWIDTH)	CMP 출력은 One-shot pulse 로 출력되는데, 출력되는 펄스의 폭을 조절할 수 있습니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. • **0** : 트리거 시점의 Command 펄스의 펄스폭과 동일한 펄스폭을 가짐 (기본값) • **양수의값** : 이 값에 1.5us 가 곱해진 값이 펄스폭이 됩니다. 즉, 이 값을 1 로 하면 1.5us, 2 로 하면 3us...와 같이 됩니다.
24 (cmMPID_ERC_ONTIME)	ERC 출력펄스의 펄스폭을 결정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. **0** => 12us (기본값), **1** => 102us, **2** => 409us, **3** => 1.6ms, **4** => 13ms, **5** => 52ms, **6** => 104ms
25 (cmMPID_SLIM_EN)	**Software limit** 의 활성/비활성을 설정합니다. 설정가능한 ***PropVal*** 은 다음과 같습니다. • **0 (FALSE)** : Software limit 을 사용하지 않습니다. (기본값) • **1 (TRUE)** : Software limit 을 사용합니다. Software limit 의 범위 설정은 **mcCfgSetSoftLimitRange()** 함수를 사용합니다.

PropId	Meaning & PropVal
26 (cmMPID_OUT_MODE)	지령 펄스(command pulse)의 출력 모드를 설정합니다. 펄스 출력 모드에 대한 설명은 아래의 "**참고 1**" 항목을 참고하십시오. 설정가능한 ***PropVal*** 은 0 ~ 5 의 값입니다. 기본값은 4 번으로 설정됩니다.
27 (cmMPID_IN_MODE)	피드백(feedback) 신호 입력 모드를 설정합니다 설정가능한 설정가능한 ***PropVal*** 은 다음과 같습니다. • **0 (cmIMODE_AB1X)** : 1 채배 A/B 상 입력 모드(엔코더) (기본값) • **1 (cmIMODE_AB2X)** : 2 채배 A/B 상 입력 모드(엔코더) • **2 (cmIMODE_AB4X)** : 4 채배 A/B 상 입력 모드(엔코더) • **3 (cmIMODE_CWCCW)** : CW/CCW 펄스 입력 모드
28 (cmMPID_IN_INV)	엔코더펄스 입력의 방향을 설정합니다. • **0 (FALSE)** : 엔코더 방향을 정상으로 인식합니다. (기본값) • **1 (TRUE)** : 엔코더 방향을 반대로 합니다.

반환값

☐ **mcCfgSetMioEnv ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

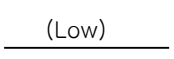
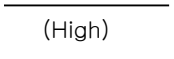
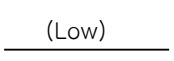
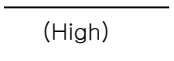
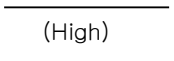
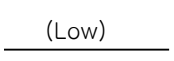
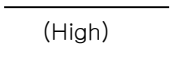
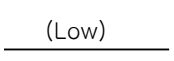
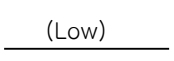
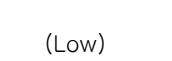
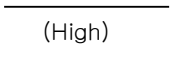
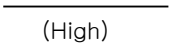
☐ **mcCfgGetMioEnv ()** 함수는 **dwPropId** 와 관련된 속성값을 반환합니다.

참고 1 <펄스출력 모드>

☐ 지령펄스(command pulse) 출력모드는 다음과 같이 6가지 모드가 있습니다. 이들 6가지 모드를 1-pulse 모드와 2-pulse 모드의 두 그룹으로 구분할 수 있습니다.

☐ 0 ~ 3번 모드는 1-pulse 모드입니다. 1-pulse 모드에서는 (+)방향 운전이든, (-)방향 운전이든지 간에 CW 단자에서만 펄스가 출력됩니다. 그리고 CCW 단자는 low 또는 high를 나타내는 레벨(level)로서 방향을 지시하게 됩니다.

☐ 4 ~ 5번 모드는 2-pulse 모드입니다. 2-pulse 모드에서는 운전 방향에 따라서 펄스가 출력되는 단자가 달라집니다. (+) 방향으로 운전할 때는 CW 단자에서 펄스가 출력되고, (-)방향으로 운전할 때는 CCW 단자에서 펄스가 출력됩니다.

Value	출력 형태			
	(+) 방향 운전 시		(-) 방향 운전 시	
	CW pin	CCW pin	CW pin	CCW pin
0				
1				
2				
3				
4				
5				

참고 2 <주요 I/O Property>

□ 대부분의 경우에 위에서 리스트된 I/O Property들은 기본값(default value)을 사용하며 변경하는 일이 거의 없습니다. 하지만 다음과 같은 Property는 시스템에 따라서 적절히 설정해주어야 하므로 주의깊게 확인해주어야 합니다.

- **0 (cmMPID_ALM_LOGIC)** : Alarm 신호의 입력 로직 설정.
- **4 (cmMPID_EL_LOGIC)**: -EL/+EL (리미트) 신호의 입력 로직 설정.
- **8 (cmMPID_EZ_LOGIC)**: Z 상 신호의 입력 로직 설정(Z상을 사용하는 원점복귀를 수행할 때만 설정 주의하면 됩니다).
- **9 (cmMPID_INP_EN)**: 서보드라이버의 INP 출력 신호를 활용할 것인지를 설정.
- **13 (cmMPID_ORG_LOGIC)**: 원점 센서 신호의 입력 로직 설정.
- **25 (cmMPID_SLIM_EN)**: 소프트웨어 리미트를 사용할 것인지를 설정.
- **26 (cmMPID_OUT_MODE)**: 지령펄스(command pulse) 출력 모드를 설정.
- **27 (cmMPID_IN_MODE)**: 피드백(feedback) 신호 입력 모드 설정.
- **28 (cmMPID_IN_INV)**: 피드백(feedback) 신호 입력의 방향을 반대로 인식할지를 설정.

예제

```
for(i=0; i<mcGnGetNumAxes(); i++){
        mcCfgSetMioEnv(i, cmMPID_EL_LOGIC, cmLOGIC_B);
        mcCfgSetMioEnv(i, cmMPID_ALM_LOGIC, cmLOGIC_A);
        mcCfgSetMioEnv(i, cmMPID_ERC_ONTIME, 6); // set ERC ontime as 104 ms
        mcCfgSetMioEnv(i, cmMPID_IN_MODE, cmIMODE_AB1X);
}
```

▣ mcCfgSetFilter / mcCfgGetFilter

함수 원형

t_i32 **mcCfgSetFilter** (t_i32 nAxis, t_bool bIsEnable)

t_bool **mcCfgGetFilter** (t_i32 nAxis)

함수 설명

각종 I/O 신호에 필터 로직을 적용할지를 설정하는 함수입니다

매개 변수

▶ **nAxis**: 축 번호

▶ **bIsEnable:** 필터 로직 설정 여부

반환값

☐ **mcCfgSetFilter ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

☐ **mcCfgGetFilter ()** 함수는 필터 설정 상태를 반환합니다.

Value	Meaning
0	필터 로직이 설정 되지 않은 상태
1	필터 로직이 설정 된 상태

참고

☐ 필터 로직 사용인 경우 +EL, -EL, SD, ORG, ALM, INP 입력 신호는 4us 미만 신호 무시-DR/+DR 신호는 3.2 us 미만 신호 무시

예제

```
t_i32 nAxis = 0;
mcCfgSetFilter(nAxis, 1);
```

▣ mcCfgSetFilterAB / mcCfgGetFilterAB

함수 원형

t_i32 **mcCfgSetFilterAB** (t_i32 nAxis, t_i32 nTarget, t_bool bIsEnable)

t_bool **mcCfgGetFilterAB** (t_i32 nAxis, t_i32 nTarget)

함수 설명

EA/EB 및 PA/PB 신호에 필터 로직을 적용할지를 설정하는 함수입니다

매개 변수

- ▶ **nAxis**: 축 번호
- ▶ **nTarget:** 필터 로직이 적용될 대상 신호
- ▶ **bIsEnable:** 필터 로직 설정 여부

반환값

□ **mcCfgSetFilterAB ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

□ **mcCfgGetFilterAB ()** 함수는 필터 설정 상태를 반환합니다.

Value	Meaning
0	필터 로직이 설정 되지 않은 상태
1	필터 로직이 설정 된 상태

예제

```
t_i32 nAxis = 0;
mcCfgSetFilterAB(nAxis, cmAB_ENC, 1);
mcCfgSetFilterAB(nAxis, cmAB_PULSAR, 1);
```

▣ mcCfgSetMaxPPS / mcCfgGetMaxPPS

함수 원형

t_i32 **mcCfgSetMaxPPS** (t_i32 nAxis, t_f32 fMaxPPS)

t_f32 **mcCfgGetMaxPPS** (t_i32 nAxis)

함수 설명

mcCfgSetMaxPPS() 함수는 지령펄스(command pulse)의 최대 주파수를 설정합니다. 이 함수는 결과적으로 모션에 적용되는 최고 속도를 제한하게 됩니다. 출력 펄스의 주파수는 최대 6.5MHz까지 설정가능하며 기본적으로 설정되는 주파수 범위는 10Hz ~ 655,350Hz입니다. 최저속도는 최대속도 설정에 따라서 자동으로 결정됩니다.

mcCfgGetMaxPPS() 함수는 지령펄스의 최대 주파수 설정값을 읽어들이는 함수입니다.

매개 변수

▶ **nAxis:** 대상 축 번호 (0-based)

▶ **fMaxPPS:** 모션의 최고 속도를 PPS 단위로 설정합니다.

주의	***fMaxPPS*** 값을 설정할 때는 로직속도 단위 설정에 관계없이 항상 **PPS** 단위로 설정하여야 합니다.

반환값

☐ **mcCfgSetMaxPPS ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

☐ **mcCfgGetMaxPPS ()** 함수는 현재 설정된 지령펄스의 최대 주파수값을 반환합니다.

참고

☐ fMaxPPS의 기본 설정값은 655,350 PPS입니다.

예제

```
t_i32 nAxis = 0;
mcCfgSetMaxPPS(nAxis, 65535);
```

▣ mcCfgSetLogicDist / mcCfgGetLogicDist

함수 원형

t_i32 **mcCfgSetLogicDist** (t_i32 nAxis, t_f32 fPPUD)

t_i32 **mcCfgGetLogicDist** (t_i32 nAxis)

함수 설명

mcCfgSetLogicDist () 함수는 논리적 단위 거리에 대한 펄스 수를 설정합니다. 여기서 논리적 단위 거리라 함은 Move함수에서 사용하는 거리 또는 위치에 대한 단위량을 의미합니다. 이 함수를 사용하여 특별히 지정하지 않는 경우에는 논리적 단위 거리에 대한 펄스 수는 초기 값인 '1' 로 사용됩니다.

mcCfgGetLogicDist () 함수는 논리적 단위 거리에 대한 펄스수 설정값을 읽어들이는 함수입니다.

매개 변수

▶ **nAxis:** 대상 축 번호 (0-based)

▶ **fPPUD:** 논리적거리 1을 이동하기 위해서 출력되어져야 하는 펄스 수를 지정합니다 (Pulses per Unit Distance). 예를 들어서 논리적거리 단위를 mm로 하고자 한다면, 1mm를 이송하기 위해서 필요한 펄스수를 지정합니다. 그러면 이후부터 거리의 단위는 mm가 됩니다.

반환값

☐ **mcCfgSetLogicDist ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

☐ **mcCfgGetLogicDist ()** 현재 설정되어 있는 논리적 단위 거리에 대한 펄스수 설정값을 반환합니다.

참고

☐ 모션컨트롤러가 서보드라이버에게 실제 출력하는 지령 펄스의 수는 (지령거리 x fPPUD)가 됩니다. 예를 들어서 기구물이 1mm 이송하는데 1000펄스가 필요하다면 아래 코드 예와 같이 fPPUD = 1000으로 설정합니다. 이렇게 논리적 거리 단위를 설정한 이후에는 모든 거리의 단위는 mm가 됩니다.

```
t_i32 nAxis = 0;
mcCfgSetLogicDist (nAxis, 1000);
mcSxMove (AXIS0, 50, true); // 50mm 이송한다. (실제 출력 펄스는 50,000 펄스가 된다)
mcSxMoveTo (AXIS0, 180, true); // 절대 위치가 180mm 가 되는 위치로 이송한다.
```

▣ mcCfgSetLogicSpeed / mcCfgGetLogicSpeed

함수 원형

t_i32 **mcCfgSetLogicSpeed** (t_i32 nAxis, t_f32 fPPUS)

t_f32 **mcCfgGetLogicSpeed** (t_i32 nAxis)

함수 설명

mcCfgSetLogicSpeed () 함수는 논리적 단위 속도에 대한 실제 펄스 출력속도(PPS)를 설정합니다. 여기서 논리적 단위 속도라 함은 속도 지정함수에서 사용하는 속도 또는 가속도에 대한 단위량을 의미합니다. 이 함수를 사용하여 특별히 지정하지 않는 경우에는 단위 속도에 대한 펄스 출력속도는 1 PPS로 사용됩니다.

mcCfgGetLogicSpeed () 함수는 논리적 단위 속도에 대한 현재 설정값을 읽어들이는 함수입니다.

매개 변수

▶ **nAxis:** 대상 축 번호 (0-based)

▶ **fPPUS:** 논리적 단위 속도에 대한 펄스 출력 속도(PPS)를 지정합니다(Pulses per Unit Speed).

반환값

□ **mcCfgSetLogicSpeed ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

□ **mcCfgGetLogicSpeed ()** 현재 설정되어 있는 논리적 단위 속도에 대한 설정값을 반환합니다.

참고

사용자의 특성에 따라 속도에 대한 단위가 다를 수 있습니다. 즉, 어떤 사용자는 속도 단위를 RPM으로 표현하는 것이 용이할 수 있고 어떤 사용자는 m/sec로 표현하는 것이 용이할 수 있습니다. CfgSetUnitSpeed() 함수는 사용자가 속도의 단위를 결정하도록 하는 함수입니다. 이 함수를 다음의 예를 참고하여 사용하십시오.

Ex 1) 1회전에 필요한 펄스 수가 3600 펄스인 경우에 속도의 단위를 RPM으로 하고자 한다면 fUnitDist값을 3600/60, 즉 60 PPS로 설정합니다(여기서 60으로 나누는 것은 RPM은 분당 회전수이므로 초당 3600/60펄스를 출력해야 1분에 3600펄스가 나가기 때문입니다).

Ex 2) 1cm 이송에 필요한 펄스 수가 1000 펄스인 경우에 이동량의 단위를 cm/sec로 하고자 한다면 fUnitDist값을 1000 PPS로 설정합니다.

예제

```
/* 1mm 이동에 필요한 펄스수가 100 인 경우, Logic Distance 를 100 으로 놓으면,
 Logic Speed 가 100 일때는 1mm 이동시 1 초가 걸린다고 가정하면, Logic Speed 가
10 일때는,  1mm 이동시 100/10 = 10 배의 시간이 더 걸립니다.*/

t_i32 nAxis = 0;
mcCfgSetLogicDist (nAxis, 100);
mcCfgSetLogicSpeed(nAxis, 10); // 0 번축: mm/sec 에 필요한 PPS
```

▣ mcCfgSetIniSpeed / mcCfgGetIniSpeed

함수 원형

t_i32 **mcCfgSetIniSpeed** (t_i32 nAxis, t_f32 fIniSpeed)

t_f32 **mcCfgGetIniSpeed** (t_i32 nAxis)

함수 설명

mcCfgSetIniSpeed () 함수는 지정한 축에 대한 초기속도를 설정합니다. 특별한 경우가 아니면 이 함수를 사용할 필요가 없으며 기본적으로 초기속도는 0으로 설정됩니다.

mcCfgGetIniSpeed () 함수는 지정한 축에 대한 초기속도의 현재 설정값을 읽어들이는 함수입니다.

매개 변수

▶ **nAxis:** 대상 축 번호 (0-based)

▶ **fIniSpeed:** 초기속도값. 초기속도의 단위는 논리적 속도 단위입니다.

반환값

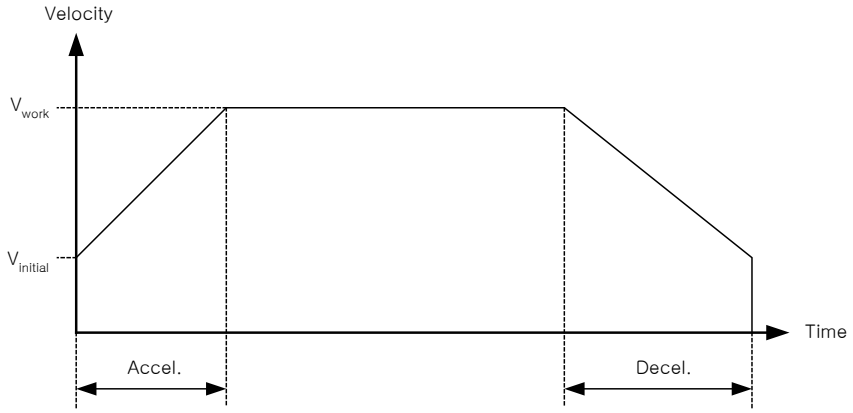
☐ **mcCfgSetIniSpeed ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

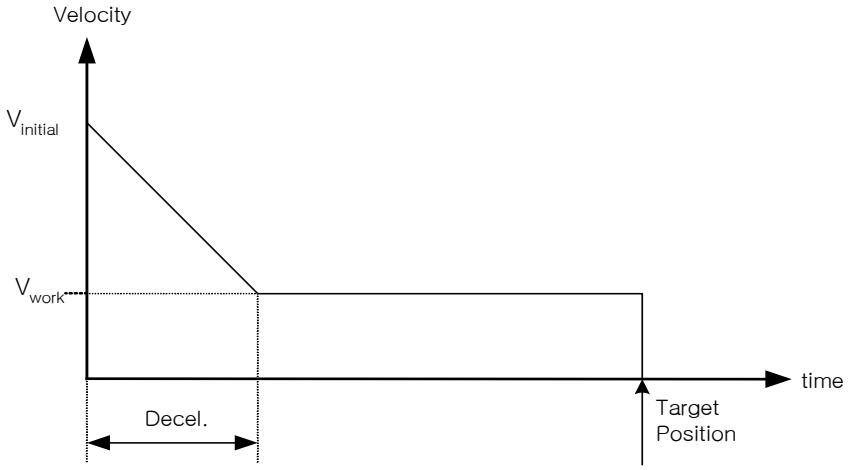
☐ **mcCfgGetIniSpeed ()**지정한 축에 대한 초기속도의 현재 설정값을 반환합니다.

참고

☐ 초기속도는 [그림 2-2]과 같이 가속의 시작속도와 감속의 종료속도를 의미합니다. 단, 초기속도가 작업속도보다 높게 설정된 경우에는 [그림 2-3]와 같이 초기속도로부터 출발하여 작업속도까지 가속 후에 작업속도를 유지하고 목표 위치까지 이동한 후에는 감속 없이 바로 정지하게 됩니다.



[그림 2-2] 작업속도가 초기속도보다 크게 설정된 경우의 속도 구성



[그림 2-3] 작업속도가 초기속도보다 작게 설정된 경우의 속도 구성

□ CONSTANT 속도모드에서는 초기속도가 무시됩니다.

예제

```
t_i32 nAxis = 0;
t_f32 fVel=1000.0, fVini=0.0;

// 1st-step 초기 속도는 0 으로 부터 시작하도록...
mcCfgSetIniSpeed (nAis, fVini);
mcSxSetSpeedRatio (nAxis, cmSMODE_KEEP, 50, 100, 0); // 감속구간을 없애기 위해서 감속도를 0 으로 설정
mcSxMove(nAxis,10000, cmTRUE);

// 2nd-step 은 1st-step 의 작업속도에서부터 가속을 시작하도록 초기속도를 1st-step 의 작업속도와 같은 값으로 설정한다.
fVini = fVel * 0.5;
mcCfgSetIniSpeed (nAxis, fVini);
mcSxSetSpeedRatio (nAxis, cmSMODE_KEEP, 70, 100, 0); // 감속구간을 없애기 위해서 감속도를 0 으로 설정
mcSxMove(nAxis,10000, cmTRUE);
```

▣ mcCfgSetSoftLimitRange / mcCfgGetSoftLimitRange

함수 원형

t_i32 **mcCfgSetSoftLimitRange** (t_i32 nAxis, t_f32 fNegLim, t_f32 fPosLim)

void **mcCfgGetSoftLimitRange** (t_i32 nAxis, t_f32 *pfNegLim, t_f32 *pfPosLim)

함수 설명

mcCfgSetSoftLimitRange () 함수는 지정한 축의 소프트웨어 리미트 값을 설정합니다.

mcCfgGetSoftLimitRange () 함수는 지정한 축의 현재 설정된 소프트웨어 리미트 값을 읽어들이는 함수입니다.

매개 변수

- ▶ **nAxis:** 대상 축 번호 (0-based)
- ▶ **fNegLim:** (-) 방향으로의 소프트웨어 리미트 값을 설정합니다.
- ▶ **fPosLim:** (+) 방향으로의 소프트웨어 리미트 값을 설정합니다.
- ▶ **pfNegLim:** (-) 방향으로의 소프트웨어 리미트 설정 값을 반환 받을 포인터.
- ▶ **pfPosLim:** (+) 방향으로의 소프트웨어 리미트 설정 값을 반환 받을 포인터.

반환값

☐ **mcCfgSetSoftLimitRange ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

☐ **mcCfgGetSoftLimitRange ()** 함수는 반환값이 없습니다.

예제

```
t_i32 nAxis = 0;

mcCfgSetSoftLimitRange (nAxis, -10000, 10000);
mcSxMoveTo(nAxis ,20000, cmTRUE); // 20000 위치로 이송하지 못하고 S/W Limit 을 만납니다.
```

▣ mcCfgSetCorrection / mcCfgGetCorrection

함수 원형

t_i32 **mcCfgSetCorrection** (t_i32 nAxis, t_i32 nCorrMode, t_f32 fCorrAmount, t_f32 fCorrVel, t_i32 dwCntrMask)

t_i32 **mcCfgGetCorrection** (t_i32 nAxis, t_i32 *pnCorrMode, t_f32 *pfCorrAmount, t_f32 *pfCorrVel, t_i32 *pdwCntrMask)

함수 설명

mcCfgSetCorrection () 함수는 함수는 백래쉬(Backlash) 또는 슬립(Slip)에 대한 보정을 설정하는 함수입니다. 구조적으로 백래쉬나 슬립 현상이 심하게 일어나는 경우에는 이에 대한 보정이 필요할 수 있습니다.

백래쉬는 일반적으로 모터의 구동 방향이 바뀔 때 발생됩니다. 따라서 백래쉬 보정은 모터의 제어 방향이 바뀔 때에만 적용됩니다. 백래쉬 보정을 활성화하면 모션컨트롤러에서 지령되는 이동 명령의 이동 방향이 이전의 이동 방향과 다른 경우에 자동적으로 백래쉬 보정 설정에 따라 보정 펄스가 출력된 후에 지정된 이동을 수행합니다. 이 설정은 단축구동뿐 아니라, 다축구동, 보간구동에서도 적용됩니다.

슬립은 일반적으로 정지 후 재기동시에 발생합니다. 따라서 슬립보정은 이동방향에 상관없이 기동시에 보정 펄스가 출력됩니다. 슬립보정을 활성화한 후에 이동명령이 하달되면 모션컨트롤러는 슬립 보정 설정에 따라 보정 펄스가 출력된 후에 지정된 이동을 수행합니다.

mcCfgGetSoftLimitRange () 함수는 지정한 축의 현재 설정된 소프트웨어 리미트 값을 읽어들이는 함수입니다.

매개 변수

▶ **nAxis:** 대상 축 번호 (0-based)

▶ **nCorrMode:** 보정 모드를 설정합니다. 이 값은 다음과 같습니다.

BIT No.	Meaning
0 (cmCORR_DIS)	보정기능을 비활성화합니다.
1 (cmCORR_BACK)	보정모드를 백래쉬 보정모드로 설정합니다.
2 (cmCORR_SLIP)	보정모드를 슬립 보정모드로 설정합니다.

▶ **fCorrAmount:** 보정 펄스의 수를 결정합니다. 단, 이 값은 논리적 거리 단위로 설정해야 합니다. 따라서 "Unit distance"(D_u)를 1이 아닌 값으로 설정한 경우에 실제 출력되는 보정 펄스수(N_c)는 다음과 같습니다.

$N_c = CorrAmount * D_u$

그리고 보정펄스의 수(N_c)는 0 ~ 4095의 값이어야 합니다.

▶ **fCorrVel:** 보정펄스 출력시의 주파수를 결정합니다. 단, 이 값은 논리적 속도 단위로 설정해야 합니다. 따라서 "Unit speed"(V_u)를 1이 아닌 값으로 설정한 경우에 실제 출력 주파수(F_c)는 다음과 같습니다.

F_u (PPS) = CorrVel * V_u

그리고, 하드웨어적으로 보정펄스 출력 주파수를 설정하는 레지스터는 원점복귀시의 Reverse Velocity (Vr)과 같은 레지스터를 사용합니다. 따라서 원점복귀시의 Vr이 보정펄스 출력시의 속도와 다른 경우에는 원점복귀를 수행한 후에 이 함수를 다시 수행해주어야 합니다.

▶ **dwCntrMask:** 보정펄스가 출력되는 동안에 각 카운터의 동작여부를 아래의 표와 같이 각 비트별로 설정하여 결정합니다. 예를 들어 이 값의 BIT0을 1로 하면 보정펄스가 출력되는 동안에도 Command Counter의 값은 증가 또는 감소합니다. 그리고 BIT0을 0으로 하면 보정펄스가 출력되는 동안에는 Command Counter의 값이 변화하지 않습니다.

Value	Meaning
BIT0	1 : 보정펄스 출력시에 Command Counter 가 동작합니다.
BIT1	1 : 보정펄스 출력시에 Feedback Counter 가 동작합니다.
BIT2	1 : 보정펄스 출력시에 Deviation Counter 가 동작합니다.
BIT3	1 : 보정펄스 출력시에 General Counter 가 동작합니다.

반환값

□ **mcCfgSetCorrection() / mcCfgGetCorrection()** 함수 모두 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcCfgSetRingCntr / mcCfgGetRingCntr

함수 원형

t_i32 **mcCfgSetRingCntr** (t_i32 nAxis, t_i32 nTargCntr, t_i32 bIsEnable, t_f32 fMaxPos)

t_i32 **mcCfgGetRingCntr** (t_i32 nAxis, t_i32 nTargCntr, t_i32* pbIsEnable, t_f32* pfMaxPos)

함수 설명

Ring Counter 기능 사용 여부 및 최대 카운터값을 설정합니다.

매개 변수

▶ **nAxis**: 축 번호

▶ **nTargCntr:** ring counter 기능 활성/비활성 설정 대상 카운터를 지정합니다.

cmCNT_COMM(0): command counter를 ring counter로 사용할지 말지를 결정합니다.

cmCNT_FEED(1): feedback counter를 ring counter로 사용할지 말지를 결정합니다.

▶ **bIsEnable:** 지정한 축의 지정한 카운터를 RING 카운터로 사용할지의 여부

▶ **fMaxPos:** Ring-counter의 카운트 범위를 설정합니다(논리 거리 단위).

반환값

☐ **mcCfgSetRingCntr () / mcCfgGetRingCntr ()** 함수 모두 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

참고

☐ Ring-counter 기능이 활성화되면 지정한 카운터는 0 ~ CntMax 사이의 값에서만 카운트 됩니다.

☐ 0, 1, 2, 3,...CntMax, 0, 1, 2,... 또는 역방향일때 0, CntMax, CntMax-1, ..., 2, 1, 0, CntMax, CntMax-1,... 과 같이 카운트합니다.

☐ command counter를 ring counter로 사용하는 경우에는 (+)software limit을 사용할 수 없습니다.

☐ feedback counter를 ring counter로 사용하는 경우에는 (-)software limit을 사용할 수 없습니다.

☐ 이 함수가 한번 실행되면 이전의 SOFTWARE LIMIT 설정은 무효화됩니다.

예제

```
t_i32 nErrCode = 0;
nErrCode = cnmCfgRingCntr_Set (0, cmCNT_COMM, cmTRUE, 7000);
```

▣ mcHomeSetConfig / mcHomeGetConfig

함수 원형

t_i32 **mcHomeSetConfig** (t_i32 nAxis, t_i32 nHomeMode, t_i32 nEzCnt, t_f32 fEscDist)

t_bool **mcHomeGetConfig** (t_i32 nAxis, t_i32* pnHomeMode, t_i32* pnEzCnt, t_f32* pfEscDist)

함수 설명

mcHomeSetConfig () 함수는 원점복귀에 관련된 여러 가지 환경을 설정합니다.

mcHomeGetConfig () 함수는 현재 설정되어 있는 여러 가지 원점복귀 환경 설정값을 읽어 옵니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **nHomeMode:** 원점복귀 모드 번호를 설정값입니다. 원점복귀 모드는 총 13가지(0 ~ 12)가 지원됩니다.

▶ **nEzCount:** 이 값은 ORG 신호 또는 EL 신호가 ON이 된 후에 실제로 복귀 작업을 완료하는데 필요한 EZ count값을 0 ~ 15 사이의 값으로 설정합니다. 이 값의 참조 여부는 원점복귀 모드에 따라서 다릅니다.

▶ **fEscDist:** 이 값은 원점 탈출 거리를 지정합니다. 이때 거리의 단위는 논리적 거리 단위를 사용합니다.

반환값

☐ **mcHomeSetConfig ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

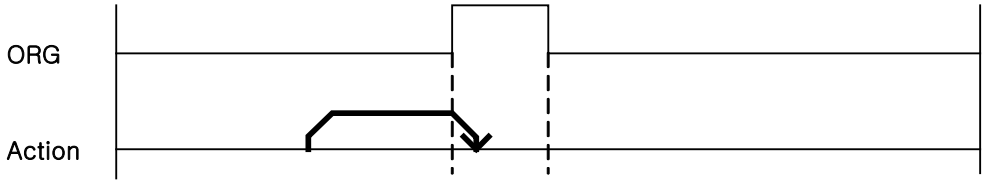
☐ **mcHomeGetConfig ()** 함수는 반환값이 없습니다.

참고 1 <원점복귀 모드>

□ **nHomeMode** 파라미터에 의하여 설정되는 원점복귀모드에 따른 동작방식은 아래 설명과 같습니다. 아래의 그림은 모두 속도모드를 Trapezoidal 모드로 설정한 상태임을 가정하여 그려진 것이며, 만일 Constant 속도 모드로 설정된 경우에는 감속이 없이 즉시 정지하게 됩니다.

□ MODE 0 : ORG ON => Stop

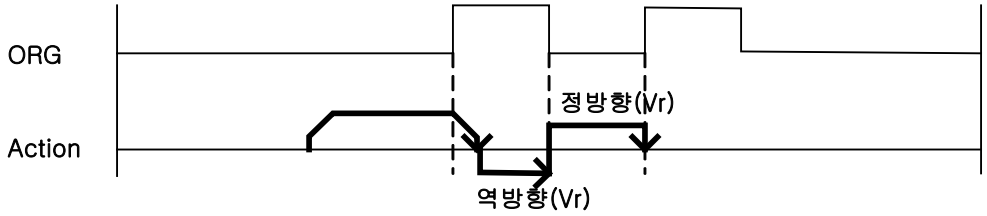
MODE 0 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌는 순간에 모션을 감속 후 정지하고 복귀 작업을 종료합니다.



□ MODE 1 : ORG ON => Stop => Back (Vr) => ORG OFF => Foward(Vr) => ORG ON => Stop

MODE 1 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌는 순간에 모션을 감속 후 정지한 후 ORG 신호가 OFF 가 될때까지 Vr(Reverse Speed)의 속도로 역방향 회전을 수행합니다. ORG 신호가 OFF 되면 다시 Vr 의 속도로 정방향 회전을 수행하다가 ORG 신호가 다시 ON 되는 순간에 복귀작업을 종료합니다.

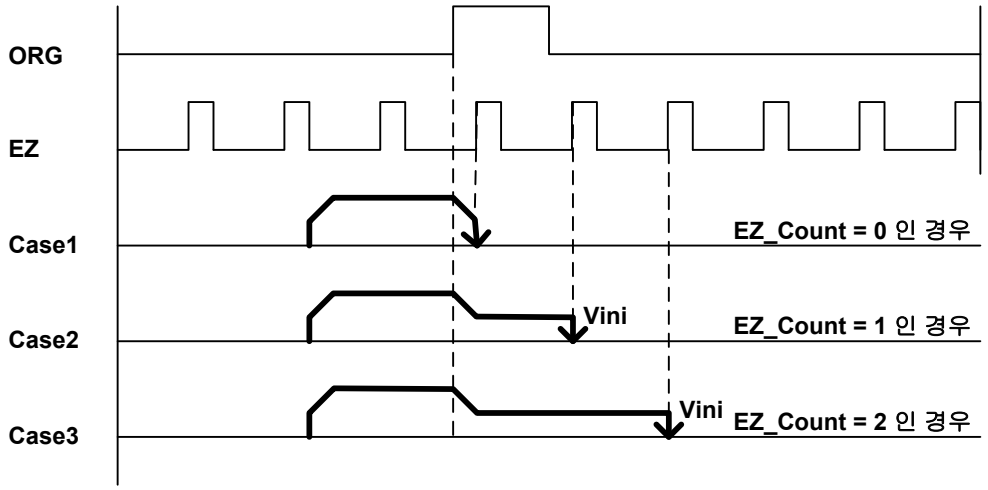
이 모드를 사용할 때 주의할 것은 처음 ORG 신호가 ON 으로 변경되어 감속후 정지하는 도중에 ORG 센서의 ON 으로 유지되는 시간이 짧아서 이미 OFF 상태로 변경되면 역방향 이동을 생략하고 최종적으로 원점센서를 찾으려는 단계로 넘어갑니다. 따라서 이러한 경우에는 (-)Limit 위치까지 이동하게 됩니다. 이러한 경우에도 자동으로 다시 탈출하여 정상적인 원점복귀 작업을 다시 수행하지만 의도하지 않게 원점복귀 시간이 길어질 수 있습니다. 이를 방지하는 방법은 구조적으로 ORG 신호가 ON 으로 유지되는 시간을 길게해주거나 또는 원점복귀시의 작업속도를 낮추거나 감속도를 크게해주어 감속시 이동되는 거리를 짧게해주면 됩니다.



Vr : Reverse Speed를 의미합니다.

□ MODE 2 : ORG ON => Slow down (Vini) => Stop on EZ Count

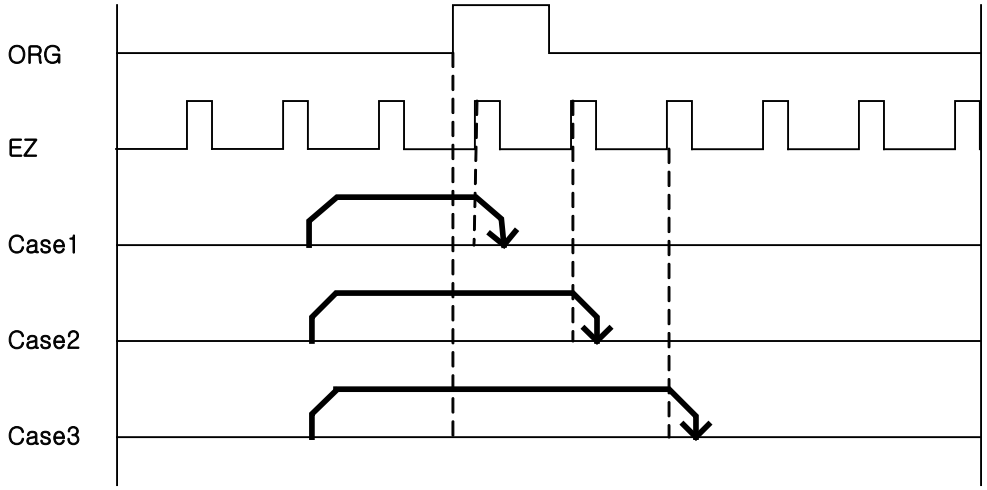
MODE 2 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌는 순간에 모션을 감속한 후 초기속도값으로 모션을 진행하다가 EZ 신호에 따라 복귀작업을 종료합니다. 미리 설정된 EzCount 값에 따라 종료하는 시점은 아래와같이 달라집니다.



Vini : 초기속도를 의미하며, mcCfgSetIniSpeed 함수 참조

□ MODE 3 : ORG => Stop on EZ Count

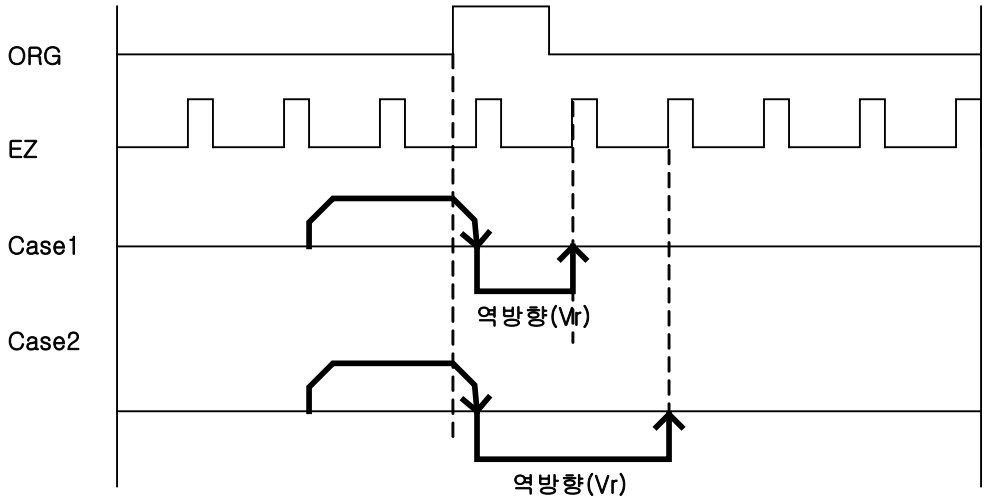
MODE 3 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌고 난 후 발생하는 EZ 신호에 따라 감속 후 정지합니다. 미리 설정된 EzCount 값에 따라 감속을 시작하는 시점은 아래와같이 달라집니다.



Case1 : EzCount = 0 인 경우
Case2 : EzCount = 1 인 경우
Case3 : EzCount = 2 인 경우

□ MODE 4 : ORG ON => Stop => Back (Vr) => Stop on EZ Count

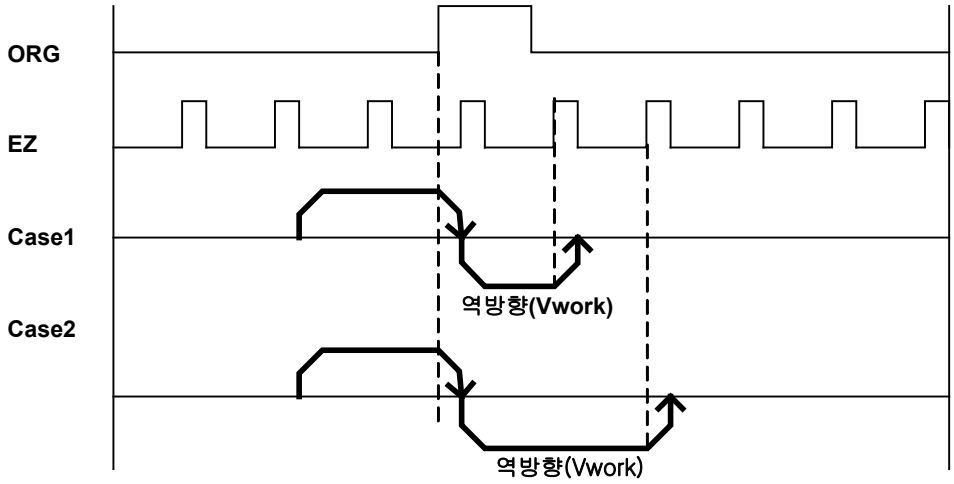
MODE 4 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌는 순간 감속 후 정지합니다. 그리고 Vr 의 속도로 역방향 회전한 후 EZ 신호에 따라 복귀 작업을 종료합니다.



Case1 : EzCount = 1 인 경우
Case2 : EzCount = 2 인 경우
Vr : Reverse Speed

□ MODE 5 : ORG ON => Stop => Back (Vwork) => Stop on EZ Count

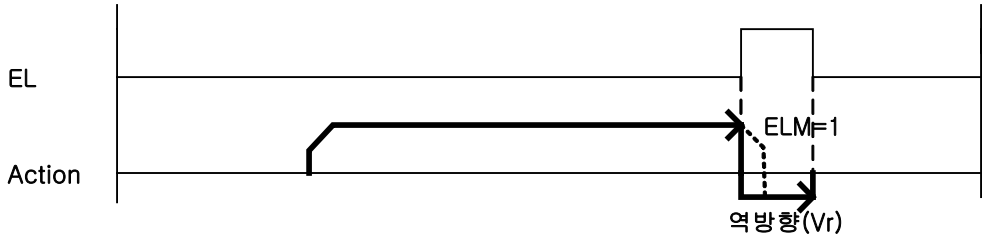
MODE 5 에서는 ORG 신호가 OFF 에서 ON 으로 바뀌는 순간 감속 후 정지합니다. 그리고 작업속도까지 가속하여 역방향 회전한 후 EZ 신호에 따라 감속 후 복귀 작업을 종료합니다.



Case1 : EzCount = 1 인 경우
Case2 : EzCount = 2 인 경우

□ MODE 6 : EL ON => Stop => Back (Vr) => EL OFF => Stop

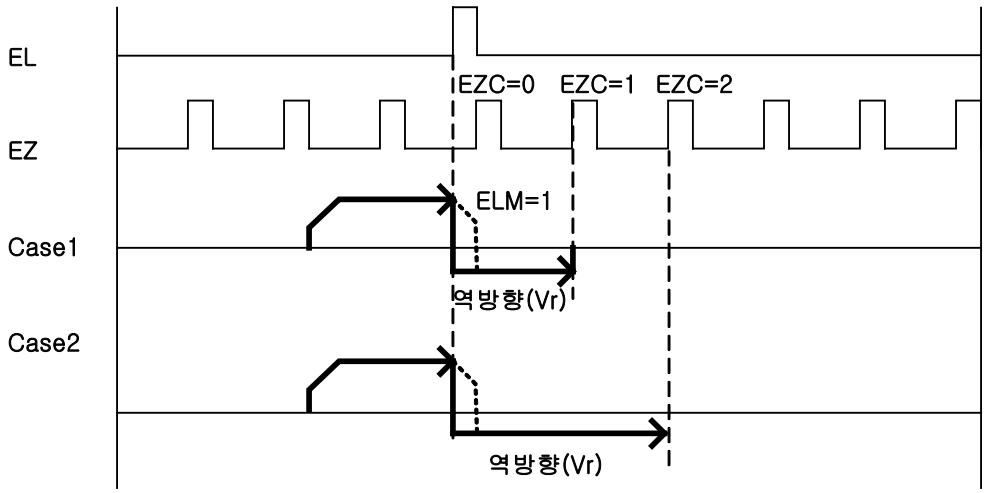
MODE 6 에서는 EL 신호가 ON 으로 바뀌는 순간 즉시 정지(또는 ELM=1 인 경우에 감속후 정지)합니다. 그리고 반대 방향으로 Vr 속도로 회전하다가 EL 신호가 OFF 되는 순간에 복귀작업을 종료합니다. 여기서 ELM=1 은 EL 의 "Stop mode"가 "Immediate stop" 모드로 설정됨을 의미합니다.



Vr : Reverse Speed

□ MODE 7 : EL ON => Stop => Back (Vr) => Stop on EZ count

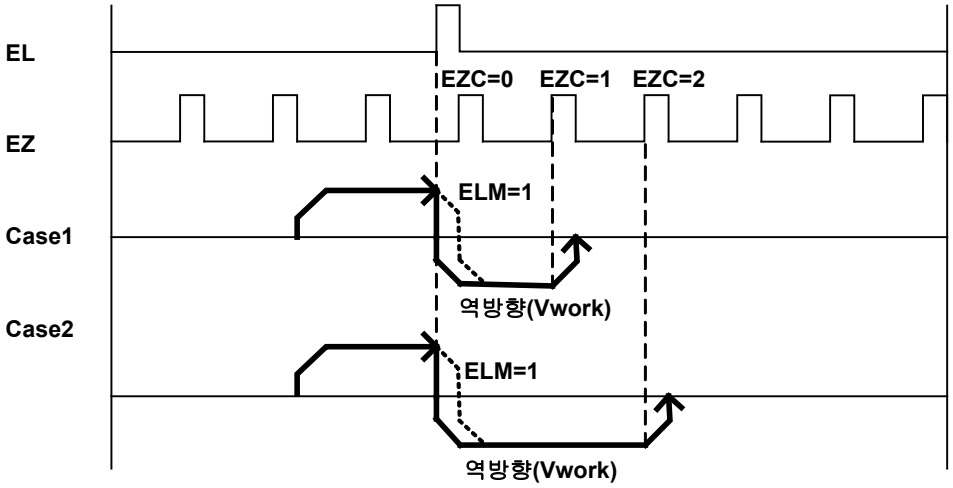
MODE 7 에서는 EL 신호가 ON 으로 바뀌는 순간 즉시 정지(또는 ELM=1 인 경우에 감속후 정지)합니다. 그리고 반대 방향으로 Vr 속도로 회전하다가 EZ_Count 에 따라 복귀작업을 종료합니다. 여기서 ELM=1 은 EL 의 "Stop mode"가 "Immediate stop" 모드로 설정됨을 의미합니다.



Case1 : EzCount = 1 인 경우
Case2 : EzCount = 2 인 경우
Vr : Reverse Speed

□ MODE 8 : EL ON => Stop => Back (Vwork) => Stop on EZ count

MODE 8 에서는 EL 신호가 ON 으로 바뀌는 순간 즉시 정지(또는 ELM=1 인 경우에 감속후 정지)합니다. 그리고 작업속도까지 가속하여 반대 방향으로 회전한 후 EZ 신호에 따라 감속 후 복귀 작업을 종료합니다. 여기서 ELM=1 은 EL 의 "Stop mode"가 "Immediate stop" 모드로 설정됨을 의미합니다.



Case1 : EzCount = 1 인 경우
Case2 : EzCount = 2 인 경우

□ MODE 9 : MODE0 => Operate till dev. counter 0

MODE 9 에서는 MODE 0 과 동일한 복귀 작업을 수행합니다. 그리고 난후, 편차카운터(Deviation Counter)가 0 이 되도록 모션을 다시 취한 후에 복귀 작업을 종료합니다.

□ MODE 10 : MODE3 => Operate till dev. counter 0

MODE 10 에서는 MODE 3 과 동일한 복귀 작업을 수행합니다. 그리고 난후 편차카운터(Deviation Counter)가 0 이 되도록 모션을 다시 취한 후에 복귀 작업을 종료합니다.

□ MODE 11 : MODE5 => Operate till dev. counter 0

MODE 11 에서는 MODE 5 과 동일한 복귀 작업을 수행합니다. 그리고 난후 편차카운터(Deviation Counter)가 0 이 되도록 모션을 다시 취한 후에 복귀 작업을 종료합니다.

□ MODE 12 : MODE8 > Operate till dev. counter 0

MODE 12에서는 MODE 8과 동일한 복귀 작업을 수행합니다. 그리고 난후 편차카운터(Deviation Counter)가 0이 되도록 모션을 다시 취한 후에 복귀 작업을 종료합니다.

▣ mcHomeSetConfigEx / mcHomeGetConfigEx

함수 원형

t_i32 **mcHomeSetConfigEx** (t_i32 nAxis, t_i32 nHomeMode, t_i32 nDir, t_i32 nEzCnt, t_f32 fEscDist, t_f32 fOffset)

t_i32 **mcHomeGetConfigEx** (t_i32 nAxis, t_i32* pnHomeMode, t_i32 *pnDir, t_i32* pnEzCnt, t_f32* pfEscDist, t_f32* pfOffset)

함수 설명

원점 복귀에 관련된 여러 가지 환경을 설정합니다. 기본 설정함수에 비해 원점복귀 방향 및 오프셋 이동거리값을 추가로 설정할 수 있습니다.

매개 변수

- ▶ **nAxis**: 축 번호
- ▶ **nHomeMode:** 원점 복귀 모드 (0~12)
- ▶ **nDir:** 원점 복귀 방향
- ▶ **nEzCnt:** Z 상 카운트 회수
- ▶ **fEscDist:** 원점 탈출 거리
- ▶ **fOffset:** 원점 복귀 완료 후 수정된 원점으로 설정할 Offset 거리.

반환값

☐ **mcHomeSetConfigEx () / mcHomeGetConfigEx ()** 함수 모두 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

참고

☐ 원점 복귀 모드 관련해서는 위의 mcHomeSetConfig 함수를 참고하세요

예제

```
t_i32 nAxis = 0;
mcHomeSetConfigEx(nAxis, 1, cmDIR_N, 0, 1000, 0); // 원점복귀 방향과 오프셋값까지 설정합니다.
```

▣ mcHomeSetPosClrMode / mcHomeGetPosClrMode

함수 원형

t_i32 **mcHomeSetPosClrMode** (t_i32 nAxis, t_i32 nPosClrMode)

t_i32 **mcHomeGetPosClrMode** (t_i32 nAxis)

함수 설명

원점복귀 완료 후에 Command 및 Feedback 위치를 재설정하거나 설정된 값을 얻는 함수입니다.

매개 변수

▶ **nAxis**: 축 번호

▶ **nPosClrMode:** 원점복귀 완료 후 Command 및 Feedback 위치가 클리어되는 모드를 결정합니다. 아래와 같이 3가지 모드를 사용하실 수 있습니다.

Value	Meaning
0 (cmHPCM_M0)	최종 완료 조건에 해당하는 외부 하드웨어 신호가 입력되는 순간에 Command 및 Feedback 의 위치가 0으로 소거됩니다. 일정량의 Feedback 위치 편차를 보입니다.
1 (cmHPCM_M1)	원점 복귀 모드에 상관없이 하드웨어 신호의 입력 상태와 더불어 소프트웨어 적인 모션완료 확인 동작을 포함한, 전체적인 원점복귀 작업이 모두 완료된 후에 Command 와 Feedback 위치가 모두 자동으로 0으로 소거됩니다. 일정량의 Feedback 위치 편차를 보입니다.
2 (cmHPCM_M2)	1차적으로는 cmHPCM_M0 일 때와 동일하게 동작합니다. 단, 원점복귀가 완료된 후에 Feedback 위치와 동일한 값으로 Command 위치를 셋팅하므로써 Command 와 Feedback 을 일치하도록 동작합니다. 이를 통해 서보드라이버에서 실제 이송한 위치에 대한 양을 Command 위치에 반영시켜서, 다음 모션 동작을 수행할 수 있도록 합니다.

반환값

☐ **mcHomeSetPosClrMode () / mcHomeGetPosClrMode ()** 함수 모두 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nAxis = 0;
mcHomeSetPosClrMode(nAxis, 2); // 홈복귀 완료후 Feedback 값을 Command 값에 덮어 씁니다.
```

▣ mcHomeSetSpeedPattern / mcHomeGetSpeedPattern

함수 원형

t_i32 **mcHomeSetSpeedPattern** (t_i32 nAxis, t_i32 nVMode, t_f32 fVel, t_f32 fAcc, t_f32 fDec, t_f32 fRvsVel)

void **mcHomeGetSpeedPattern** (t_i32 nAxis, t_i32* pnVMode, t_f32* pfVel, t_f32* pfAcc, t_f32* pfDec, t_f32* pfRvsVel)

함수 설명

mcHomeSetSpeedPattern () 함수는 원점복귀 작업시의 속도 패턴을 설정합니다.

mcHomeGetSpeedPattern () 함수는 현재 설정되어 있는 원점복귀 속도패턴을 읽어옵니다.

매개 변수

▶ **Axis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **nVMode:** 속도모드를 설정합니다. 설정값은 다음과 같습니다.

Value	Meaning
0	CONSTANT 속도 모드를 의미합니다. 즉, 가감속을 수행하지 않습니다.
1	TRAPEZODIAL 속도모드를 의미합니다. 즉, 사다리꼴의 가감속을 수행합니다.
2	S-CURVE 속도모드를 의미합니다. 즉, S-CURVE 형태의 가감속을 수행합니다.

▶ **fVel:** 이 값은 작업속도 설정값입니다. 작업속도의 단위는 논리적 속도 단위입니다.

▶ **fAcc:** 이 값은 가속도 설정값입니다. 가속도의 단위는 논리적 속도 단위에 대응됩니다. “Unit speed”를 1로 한 경우에 가속도의 단위는 Pulses/sec^2이 됩니다.

▶ **fDec:** 이 값은 감속도 설정값입니다. 감속도의 단위는 논리적 속도 단위에 대응됩니다. “Unit speed”를 1로 한 경우에 가속도의 단위는 Pulses/sec^2이 됩니다.

▶ **fRvsVel:** 이 값은 “Reverse Velocity”라고 하는 속도 설정값입니다. 원점복귀 모드에 따라서 “Reverse Velocity” 를 필요로 하는 모드가 있습니다.

반환값

☐ **mcHomeSetSpeedPattern ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

☐ **mcHomeGetSpeedPattern ()** 함수는 반환값이 없습니다.

예제

```
t_i32 nAxis = 0;

// 사다리꼴 속도패턴으로 작업속도 5000, 가감속 10000, 역방향속도 1000 으로
홈복귀 설정을 합니다.
mcHomeSetSpeedPattern (nAxis, cmSMODE_T , 5000, 10000, 10000, 1000);
```

▣ mcHomeMove

함수 원형

t_bool **mcHomeMove** (t_i32 Axis, t_bool bWaitDone)

함수 설명

mcHomeMove () 함수는 원점복귀 이송을 수행합니다.

매개 변수

▶ **Axis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **bWaitDone:** 이 값이 0이면 원점복귀 이송을 시작한 후에 함수가 바로 반환됩니다. 그러나 이 값이 1이면 원점복귀가 완료될 때까지 함수는 반환되지 않고 루프를 돌면서 대기합니다.

반환값

함수 수행 성공 여부를 반환합니다. 1을 반환하면 성공한 것이며, 0을 반환하면 실패한 것입니다.

참고 2 <자동원점 검색 기능>

□ **nHomeMode** 파라미터에 의하여 설정되는 원점복귀모드에 따른 동작방식은 아래 설명과 같습니다. 아래의 그림은 모두 속도모드를 Trapezoidal 모드로 설정한 상태임을 가정하여 그려진 것이며, 만일 Constant 속도 모드로 설정된 경우에는 감속이 없이 즉시 정지하게 됩니다.

일반적으로 기구물의 위치는 원점센서를 기준으로 (+) 방향의 위치에 있으며, 따라서 (-) 방향으로 원점복귀를 수행하면 됩니다. 하지만 원점복귀를 시작하는 시점에 기구물이 이미 원점센서가 ON이 되어 있는 위치에 있거나 원점센서와 (-)EL 센서 사이에 위치한 경우에는 원점센서를 기준으로 (+) 방향의 위치까지 탈출한 후에 정상적인 원점복귀 작업을 수행하여야 합니다. 이러한 기능을 "자동원점 검색" 기능이라 하며, ㈜커미조아의 모션제어보드는 이 기능을 자동으로 수행해주므로 기구물의 위치에 관계없이 원점복귀 작업을 수행할 수 있습니다.

기구물과 각 센서들의 위치에 따른 원점복귀 절차는 다음과 같이 약간씩 다릅니다. 단, 원점복귀 작업의 방향을 음의 방향으로 한경우에 대한 것입니다. 만일 원점복귀를 양의 방향으로 한 경우에는 –EL 신호 대신 +EL 신호가 사용됩니다.

□ CASE 1. 원점센서를 기준으로 양의 방향 위치에서 원점복귀를 시작한 경우

정상적인 원점복귀 작업을 수행합니다.

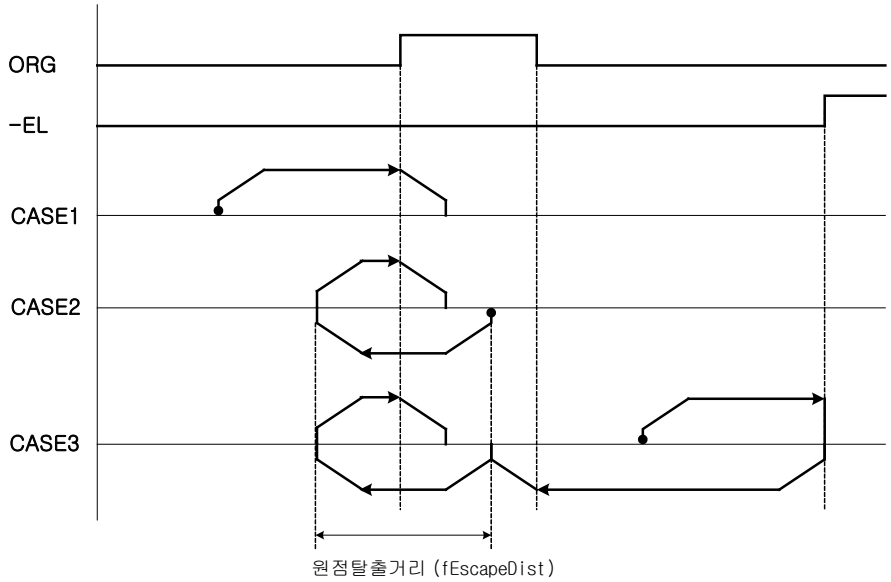
□ CASE 2. 원점센서가 ON인 상태에서 원점복귀를 시작한 경우

이러한 경우에는 먼저 원점 탈출거리(Escape distance) 만큼 양의 방향으로 이동한 후 다시 정상적인 원점복귀 작업을 수행합니다. 원점 탈출거리만큼 탈출하였어도 원점센서가 ON 상태이면 원점 탈출거리만큼 탈출하는 작업을 반복하므로 탈출거리가 짧아도 원점을 탈출하는데는 아무 문제가 없습니다.

□ CASE 3. 원점센서를 기준으로 양의 방향 위치에서 원점복귀를 시작한 경우

이러한 경우에는 먼저 음의 방향으로 이동을 시작합니다. 그리고 -EL(Negative Limit) 센서가 ON 되면 정지 후 양의 방향으로 이동합니다. 원점센서가 ON 되면 다시 정지 후 CASE 2 에서와 같이 원점 탈출거리만큼 양의 방향으로 이동한 후 다시 정상적인 원점복귀 작업을 수행합니다.

[그림 2-4]은 원점복귀모드를 0, 속도 패턴을 사다리꼴 방식으로 하고 원점복귀 작업을 수행할 때 위의 각 경우에 대하여 동작하는 방식을 도식적으로 나타낸 것입니다.



[그림 2-4] 기구물과 센서의 위치에 따른 원점복귀 작업

예제

```
t_i32 nAxis = 0;

// S-Curve 속도패턴으로 작업속도 5000, 가감속 10000, 역방향속도 1000 으로 홈복귀 설정을 합니다.
mcHomeSetSpeedPattern (nAxis, cmSMODE_S , 5000, 10000, 10000, 1000);
mcHomeMove (nAxis, cmTRUE);
```

▣ mcHomeIsBusy

함수 원형

t_bool **mcHomeIsBusy** (t_i32 nAxis)

함수 설명

지정된 축이 원점복귀를 수행하고 있는지 체크하는 함수입니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

원점복귀가 수행중인지를 알리는 플래그.

Value	Meaning
0 (FALSE)	원점복귀가 진행중이지 않음
1 (TRUE)	원점복귀가 진행중임.

예제

```
t_i32 nMstCode;
nMstCode = mcStGetMst(nAxis);
if(nMstCode < 0) return (nMstCode);
if(mcHomeIsBusy(nAxis)) return cmMST_HOMMING;
```

▣ mcHomeWaitDone

함수 원형

t_i32 **mcHomeWaitDone** (t_i32 nAxis)

함수 설명

해당축의 원점복귀가 완료될 때까지 기다리는 함수입니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode, nAxis=0;

nErrCode = mcHomeMove(nAxis, cmFALSE);
mcHomeWaitDone(nAxis);
```

▣ mcHomeSetSuccess / mcHomeGetSuccess

함수 원형

void **mcHomeSetSuccess**(t_i32 nAxis, t_bool bSuccess)

t_bool **mcHomeGetSuccess**(t_i32 nAxis)

함수 설명

mcHomeSetSuccess () 이 함수가 호출되기 이전에 원점복귀가 성공적으로 완료되었는지를 알려주는 함수입니다.

mcCfgGetLogicDist () 함수는 원점복귀의 성공여부에 대한 플래그 값을 강제로 설정하는 함수입니다. 일반적으로는 이 플래그 값은 원점복귀의 실제 수행에 의해서 셋팅됩니다. 그러나 필요한 경우에 강제로 그 값을 셋(Set) 또는 리셋(Reset)할 수 있습니다.

매개 변수

▶ **nAxis:** 대상 축 번호 (0-based)

▶ **bSuccess:** 이 함수가 호출된 시점을 기준으로 이전에 원점복귀가 성공적으로 완료된 상태인지를 알려주는 매개 변수(媒介變數)입니다.

Value	Meaning
0 (FALSE)	지정한 축은 원점복귀가 정상적으로 이루어지지 않은 상태를 의미합니다. 그 상태는 다음의 3가지 중에 하나일 수 있습니다. ① 원점복귀가 한번도 수행되지 않은 상태 ② 원점복귀가 현재 진해 중인 상태 ③ 원점복귀가 수행되었지만 수행 중에 에러(error)가 발생하였거나 사용자에 의해서 강제 종료된 상태
1(TRUE)	지정한 축이 원점복귀가 정상적으로 이루어진 상태를 의미합니다.

반환값

☐ **mcHomeSetSuccess ()** 함수는 에러코드를 반환하지 않습니다.

☐ **mcHomeSetSuccess ()** 함수는 원점복귀의 성공 여부를 나타내는 플래그값을 반환합니다. 플래그 값의 의미는 앞의 **bSuccess** 파라미터와 같은 의미를 지니게 됩니다.

참고

☐ 원점복귀의 성공 여부에 대한 플래그 값은 응용프로그램이 종료(終了)되어도 그대로 유지됩니다. 따라서 다시 응용프로그램이 시작되면 이전에 원점복귀를 정상적으로 수행했었는지를 알 수가 있습니다. 단, PC의 하드웨어적인 전원이 차단되거나 재 시작(Rebooting) 되면 그 값은 FALSE로 리셋됩니다. 따라서 cmmHomeGetSuccess() 함수의 이러한 특성(特性)을 활용하면 프로그램이 종료되었다가 다시 실행될 때 이전의 원점복귀 수행여부를 확인(確認)할 수가 없어서 매번 원점복귀를 수행해야 했던 불편을 보완(補完)할 수 있습니다.

예제

```
t_i32 nErrCode, nAxis=0;
t_bool bSuccess;

nErrCode = mcHomeMove(nAxis, cmTRUE); // 원점복귀가 완료될때까지 기다립니다.
bSuccess = mcHomeGetSuccess(nAxis);
```

▣ mcSxSetSvon / mcSxGetSvon

함수 원형

t_i32 **mcSxSetSvon** (t_i32 nAxis, t_i32 nState)

t_bool **mcSxGetSvon** (t_i32 nAxis)

함수 설명

mcSxSetSvon () 함수는 어느 한 축(Single axis)의 서보온(Servo-ON) 신호의 출력 상태를 제어합니다.

mcSxGetSvon () 함수는 어느 한 축의 현재 출력되고 있는 서보온 신호의 출력 상태를 읽어들입니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **nState:** 이 값은 서보온 출력 신호의 상태를 의미합니다. 0이면 서보온 출력 신호가 OFF 상태이며, 1이면 ON 상태를 의미합니다.

반환값

☐ **mcCfgSetLogicDist ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

☐ **mcCfgGetLogicDist ()** 서보온 출력 상태를 반환합니다.

Value	Meaning
0	서보온 출력 신호가 OFF 상태임을 의미합니다.
1	서보온 출력 신호가 ON 상태임을 의미합니다.

예제

```
t_i32 nErrCode, nAxis=0;
nErrCode = mcSxSetSvon(nAxis, 1); // 0 번 축을 서보 On 시킵니다.
nErrCode = mcSxSetSvon(nAxis+1, 0); // 1 번 축을 서보 Off 시킵니다.
```

▣ mcSxSetAlmRst / mcSxGetAlmRst

함수 원형

t_i32 **mcSxSetAlmRst** (t_i32 nAxis, t_i32 nState)

t_bool **mcSxGetAlmRst** (t_i32 nAxis)

함수 설명

mcSxSetAlmRst () 함수는 어느 한 축(Single axis)의 알람리셋(Alarm-reset) 신호의 출력 상태를 제어합니다. 알람리셋 신호는 서보드라이버의 알람 상태를 클리어(clear) 해주는 신호입니다.

mcSxGetAlmRst () 어느 한 축의 현재 출력되고 있는 알람리셋(Alarm-reset) 신호의 출력 상태를 읽어들입니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **nState:** 이 값은 알람리셋(Alarm-reset) 출력 신호의 상태를 의미합니다. 0이면 알람리셋 출력 신호가 OFF 상태이며, 1이면 ON 상태를 의미합니다.

반환값

☐ **mcSxSetAlmRst ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

☐ **mcSxGetAlmRst ()** 알람리셋 신호의 출력 상태를 반환합니다.

Value	Meaning
0	알람리셋 출력 신호가 OFF 상태임을 의미합니다.
1	알람리셋 출력 신호가 ON 상태임을 의미합니다.

예제

```
t_i32 nErrCode, nAxis=0;
nErrCode = mcSxSetAlmRst (nAxis, 1); // 0 번축의 알람리셋 신호 출력을 On 시킵니다
```

▣ mcSxSetSpeedPattern / mcSxGetSpeedPattern

함수 원형

t_i32 **mcSxSetSpeedPattern** (t_i32 nAxis, t_i32 nVMode, t_f32 fVel, t_f32 fAcc, t_f32 fDec)

void **mcSxGetSpeedPattern** (t_i32 nAxis, t_i32* pnVMode, t_f32* pfVel, t_f32* pfAcc, t_f32* pfDec)

함수 설명

mcSxSetSpeedPattern () 함수는 지정한 축의 정격속도 패턴을 설정합니다.

mcSxGetSpeedPattern () 함수는 지정한 축의 현재의 정격속도패턴 설정을 읽어들입니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **nVMode:** 속도모드를 설정합니다. 설정값은 다음과 같습니다.

Value	Meaning
0	CONSTANT 속도 모드를 의미합니다. 즉, 가감속을 수행하지 않습니다.
1	TRAPEZODIAL 속도모드를 의미합니다. 즉, 사다리꼴의 가감속을 수행합니다.
2	S-CURVE 속도모드를 의미합니다. 즉, S-CURVE 형태의 가감속을 수행합니다.

▶ fVel: 작업속도를 설정합니다.

▶ fAcc: 가속도를 설정합니다.

▶ fDec: 감속도를 설정합니다.

반환값

☐ **mcSxSetSpeedPattern ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

☐ **mcSxGetSpeedPattern ()** 함수는 반환값이 없습니다.

참고 1 <정격속도와 속도비 설정>

☐ 속도를 설정하는 함수는 mcSxSetSpeedPattern() 함수와 mcSxSetSpeedRatio() 함수의 두 가지가 있습니다.

☐ **mcSxSetSpeedPattern()** 함수는 정격속도를 설정하는 함수입니다. 그리고 **mcSxSetSpeedRatio()** 함수는 속도비(速度比, speed ratio)를 설정하는 함수입니다. 속도비

는 %값으로 설정하는데 정격속도의 몇 %의 속도로 이송할 것인지를 설정하는 것입니다. 따라서 정격속도를 V_m, 속도비를 R_V이라고 하면 실제 이송 속도 V는 다음과 같이 됩니다.

$$V = V_m \times R_V/100$$

예제

```
t_i32 nErrCode, nAxis=0;
t_f32 fWork=2000, fAcc=10000, fDec=10000;
nErrCode = mcSxSetSpeedPattern(nAxis, cmSMODE_S, fWork, fAcc, fDec);
```

▣ mcSxSetSpeedRatio / mcSxGetSpeedRatio

함수 원형

t_i32 **mcSxSetSpeedRatio** (t_i32 nAxis, t_f32 fVelR, t_f32 fAccR, t_f32 fDecR)

void **mcSxGetSpeedRatio** (t_i32 nAxis, t_f32* pfVelR, t_f32* pfAccR, t_f32* pfDecR)

함수 설명

mcSxSetSpeedRatio () 함수는 지정한 축의 속도비(速度比, speed ratio)를 설정하는 함수입니다.

mcSxGetSpeedRatio () 함수는 지정한 축의 속도비 설정을 읽어들입니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **fVelR:** 이 값은 작업속도에 대한 속도비입니다. 이 값의 단위는 (%)입니다. 예를 들어서 이 값이 '50' 이면 실제 이송 속도는 정격 작업속도의 50(%)가 됩니다.

▶ **fAccR:** 이 값은 가속도에 대한 속도비입니다. 이 값의 단위는 (%)입니다. 예를 들어서 이 값이 '50' 이면 실제 이송 가속도는 정격 가속도의 50(%)가 됩니다.

▶ **fDecR:** 이 값은 감속도에 대한 속도비입니다. 이 값의 단위는 (%)입니다. 예를 들어서 이 값이 '50' 이면 실제 이송 감속도는 정격 감속도의 50(%)가 됩니다.

반환값

☐ **mcSxSetSpeedRatio()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

☐ **mcSxGetSpeedPattern ()** 함수는 반환값이 없습니다.

참고 1 <정격속도와 속도비 설정>

☐ 속도를 설정하는 함수는 mcSxSetSpeedPattern() 함수와 mcSxSetSpeedRatio() 함수의 두 가지가 있습니다.

☐ **mcSxSetSpeedPattern()** 함수는 정격속도를 설정하는 함수입니다. 그리고 **mcSxGetSpeedRatio()** 함수는 속도비(速度比, speed ratio)를 설정하는 함수입니다. 속도비는 %값으로 설정하는데 정격속도의 몇 %의 속도로 이송할 것인지를 설정하는 것입니다. 따라서 정격속도를 V_m, 속도비를 R_V이라고 하면 실제 이송 속도 V는 다음과 같이 됩니다.

$$V = V_m \times R_V/100$$

예제

```
t_i32 nErrCode, nAxis=0;
t_f32 fWorkR=100, fAccR=100, fDecR=100;

// 100% 작업 속도로 이송
nErrCode = mcSxSetSpeedRatio(nAxis, fWorkR, fAccR, fDecR);
mcSxMove(nAxis, 10000, cmTRUE);

// 50% 작업 속도로 이송
nErrCode = mcSxSetSpeedRatio(nAxis, fWorkR / 2, fAccR, fDecR);
mcSxMove(nAxis, 10000, cmTRUE);
```

▣ mcSxJog

함수 원형

t_i32 **mcSxJog** (t_i32 nAxis, t_i32 nDir)

함수 설명

mcSxJog () 함수는 어느 한축에 대하여 조그(Jog) 이송을 시작시키는 함수입니다. 여기서 조그 이송이란 정지 함수가 호출되기 전까지 지정한 속도패턴을 유지하며 어느 한 방향으로 이송을 수행하는 것을 의미합니다. 이때 이송의 목표좌표는 없으며 정지 명령이 하달되기 전까지 계속해서 이송하게 됩니다.

이 함수는 이송을 시작 시킨 후에 바로 반환됩니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **nDir:** 이송 방향를 결정하는 파라미터입니다.

Value	Meaning
0	(-) 방향
1	(+) 방향

반환값

□ 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
void OnLButtonEvent(t_bool bDir)
{
        t_i32 nErrCode, nAxis=0;
        nErrCode = mcSxJog(nAxis, bDir); // bDir: cmDIR_N (-방향), cmDIR_P (+방향)
}
```

▣ mcSxMove

함수 원형

t_i32 **mcSxMove** (t_i32 nAxis, t_f32 fDistance, t_bool bWaitDone)

함수 설명

이 함수는 하나의 축에 대하여 현재의 위치에서 지정한 거리(상대 위치)만큼 이동을 수행합니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **fDistance:** 이동할 거리를 지정합니다. 이 값은 현재의 위치에 대한 상대 좌표이며, 거리의 단위는 논리적 거리(Logic distance) 단위를 사용합니다.

▶ **bWaitDone:** 이송완료를 기다릴지를 설정합니다.

Value	Meaning
0 (false)	이송을 시작시킨 후에 바로 반환됩니다.
1 (true)	이송이 완료될 때까지 함수는 바환되지 않습니다.

반환값

□ 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode=0, nAxis=0;
t_f32 fDistance = 10000; // 상대 이송 거리
nErrCode = mcSxMove(nAxis, fDistance, cmTRUE);
```

▣ mcSxMoveTo

함수 원형

t_i32 **mcSxMoveTo** (t_i32 nAxis, t_f32 fPosition, t_bool bWaitDone)

함수 설명

이 함수는 하나의 축에 대하여 지정한 절대좌표로의 이송을 수행합니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **fDistance:** 이송할 목표좌표를 절대좌표로 설정합니다. 거리의 단위는 논리적 거리(Logic distance) 단위를 사용합니다.

▶ **bWaitDone:** 이송완료를 기다릴지를 설정합니다.

Value	Meaning
0 (false)	이송을 시작시킨 후에 바로 반환됩니다.
1 (true)	이송이 완료될 때까지 함수는 바환되지 않습니다.

반환값

□ 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode=0, nAxis=1;
t_f32 fPosition = 100; // 절대 이송 위치
nErrCode = mcSxMoveTo(nAxis, fPosition, cmTRUE);
```

▣ mcSxMove_2Vel

함수 원형

t_i32 **mcSxMove_2Vel** (t_i32 nAxis, t_f32 fDistance, t_f32 fVel2, t_bool bWaitDone)

함수 설명

2단계 가감속도 변경 값을 설정해서 구동하는 함수입니다. 상대좌표 이송입니다.

매개 변수

- ▶ **nAxis**: 축 번호
- ▶ **fDistance**: 이송 거리
- ▶ **fVel2:** 2단계 작업속도 값
- ▶ **bWaitDone:** 이송완료를 기다릴지를 설정합니다

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode, nAxis=0;
t_f32 fWork=10000, fAcc=100000, fDec=100000;
t_f32 fWorkR=100, fAccR=100, fDecR=100;

// 0 번 축의 기본 속도를 설정합니다.
nErrCode = mcSxSetSpeedPattern(nAxis, cmSMODE_T, fWork, fAcc, fDec);

// 0 번 축의 속도 비율을 설정합니다.
nErrCode = mcSxSetSpeedRatio(nAxis, fWorkR, fAccR, fDecR);

// 이동할 거리와 2 단계 작업속도를 지정합니다.
mcSxMove_2Vel(nAxis, 20000, 30000, cmTRUE);
```

▣ mcSxMoveTo_2Vel

함수 원형

t_i32 **mcSxMoveTo_2Vel** (t_i32 nAxis, t_f32 fPosition, t_f32 fVel2, t_bool bWaitDone)

함수 설명

2단계 가감속도 변경 값을 설정해서 구동하는 함수입니다. 절대좌표 이송입니다.

매개 변수

- ▶ **nAxis**: 축 번호
- ▶ **fPosition**: 목표 위치
- ▶ **fVel2:** 2단계 작업속도 값
- ▶ **bWaitDone:** 이송완료를 기다릴지를 설정합니다

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode, nAxis=0;
t_f32 fWork=10000, fAcc=100000, fDec=100000;
t_f32 fWorkR=100, fAccR=100, fDecR=100;

// 0 번 축의 기본 속도를 설정합니다.
nErrCode = mcSxSetSpeedPattern(nAxis, cmSMODE_T, fWork, fAcc, fDec);

// 0 번 축의 속도 비율을 설정합니다.
nErrCode = mcSxSetSpeedRatio(nAxis, fWorkR, fAccR, fDecR);

// 목표 좌표와 2 단계 작업속도를 지정합니다.
mcSxMoveTo_2Vel(nAxis, 5000, 30000, cmTRUE);
```

▣ mcSxIsDone

함수 원형

t_bool **mcSxIsDone** (t_i32 nAxis)

함수 설명

mcSxIsDone () 함수는 지정한 축이 이송을 완료하였는지(정지 상태에 있는지)를 알려주는 함수입니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

반환값

지정한 축이 이송을 완료하였는지(정지 상태에 있는지)를 의미하는 값을 반환합니다.

Value	Meaning
0 (false)	모션작업이 완료되지 않음 (현재 이송이 진행중에 있음)
1 (true)	모션작업이 완료됨 (정지 상태임)

예제

```
t_bool mcMxIsDone(t_i32 nNumAxes, t_i32 nAxes[])
{
        t_ui16 i;
        for(i=0; i<nNumAxes; i++){
                if(!mcSxIsDone(nAxes[i])){
                        return false;
                }
        }
}
```

▣ mcSxStop

함수 원형

t_i32 **mcSxStop** (t_i32 nAxis, t_bool bDecel, t_bool bWaitDone)

함수 설명

mcSxStop () 함수는 이송을 정지시키는 명령입니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **bDecel:** 정지할 때 감속을 할 것인지를 설정합니다.

Value	Meaning
0	감속없이 급 정지를 합니다.
1	정지할 때 감속을 한 후에 정지합니다.

▶ **bWaitDone:** 완전히 정지할 때까지 기다릴지를 설정합니다.

Value	Meaning
0	정지 명령을 내린 후에 함수는 바로 반환됩니다. bDecel=1로 설정하였다면 이 함수가 반환된 이후에도 정지시의 감속 이송이 진행중일 수 있습니다.
1	완전히 정지할 때까지 함수는 반환되지 않습니다.

반환값

□ 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
for(i=0; i<mcGnGetNumAxes(); i++){
        mcSxStop(i, cmTRUE, cmTRUE); // 감속후 정지, 완료시까지 기다립니다.
}
```

▣ mcSxGetTargPos

함수 원형

t_f32 **mcSxGetTargPos** (t_i32 nAxis)

함수 설명

마지막 이송작업의 목표좌표를 절대값으로 반환하는 함수입니다.

매개 변수

▶ nAxis: 축 번호

반환값

□ 현재 이송이 진행중이면 진행중인 이송의 목표좌표를 반환하고, 이송이 진행중이 아니면 이전 이송의 목표좌표를 반환합니다.

예제

```
t_i32 nErrCode=0, nAxis=0;
t_f32 fPos;
fPos = mcSxGetTargPos(nAxis);
nErrCode = mcErrGetLastError();
```

▣ mcMxMove

함수 원형

t_i32 **mcMxMove** (t_i32 nNumAxes, t_i32 anAxes[], t_f32 afDistance[], t_bool bWaitDone)

함수 설명

mcMxMove () 함수는 여러 개의 축에 대하여 현재의 위치에서 지정한 거리만큼 이동을 동시에 시작합니다.

매개 변수

▶ **nNumAxes:** 동시에 작업을 수행할 대상 축의 수

▶ **anAxes:** 동시에 작업을 수행할 대상 축 번호를 담은 배열

▶ **afDistance:** 이동할 거리값을 담은 배열. 이 배열의 크기는 nNumAxes 값과 같거나 커야 합니다.

▶ **bWaitDone:** 완전히 정지할 때까지 기다릴지를 설정합니다.

Value	Meaning
0	정지 명령을 내린 후에 함수는 바로 반환됩니다. bDecel=1로 설정하였다면 이 함수가 반환된 이후에도 정지시의 감속 이송이 진행중일 수 있습니다.
1	완전히 정지할 때까지 함수는 반환되지 않습니다.

반환값

□ 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode=0;
t_i32 anAxes[] = {0, 1};
t_f32 afDist[] = {1000, 2000};
nErrCode = mcMxMove(2, anAxes, afDist, cmTRUE);
```

▣ mcMxMoveTo

함수 원형

t_i32 **mcMxMoveTo** (t_i32 nNumAxes, t_i32 anAxes[], t_f32 afPosition[], t_bool bWaitDone)

함수 설명

mcMxMoveTo () 함수는 여러 개의 축에 대하여 현재의 위치에서 지정한 절대 위치까지 이동을 동시에 시작합니다.

매개 변수

▶ **nNumAxes:** 동시에 작업을 수행할 대상 축의 수

▶ **anAxes:** 동시에 작업을 수행할 대상 축 번호를 담은 배열

▶ **afPosition:** 절대치 목표 좌표값을 담은 배열. 이 배열의 크기는 nNumAxes 값과 같거나 커야 합니다.

▶ **bWaitDone:** 완전히 정지할 때까지 기다릴지를 설정합니다.

Value	Meaning
0	정지 명령을 내린 후에 함수는 바로 반환됩니다. bDecel=1로 설정하였다면 이 함수가 반환된 이후에도 정지시의 감속 이송이 진행중일 수 있습니다.
1	완전히 정지할 때까지 함수는 반환되지 않습니다.

반환값

□ 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode=0;
t_i32 anAxes[] = {0, 1};
t_f32 afPos[] = {10000, 20000};
nErrCode = mcMxMoveTo(2, anAxes, afPos, cmTRUE);
```

▣ mcMxIsDone

함수 원형

t_bool **mcMxIsDone** (t_i32 nNumAxes, t_i32 anAxes[])

함수 설명

여러 개의 축에 대하여 지정한 모든 축의 모션이 완료됐는지를 체크합니다.

매개 변수

▶ **nNumAxes:** 동시에 작업을 수행할 대상 축의 수.

▶ **anAxes:** 작업완료를 체크할 대상 축의 배열 주소값. 이 배열의 크기는 NumAxes값과 일치하거나 커야 합니다.

반환값

지정한 축이 이송을 완료하였는지(정지 상태에 있는지)를 의미하는 값을 반환합니다.

Value	Meaning
0 (false)	하나 이상의 축의 모션작업이 완료되지 않음 (현재 이송이 진행 중에 있음)
1 (true)	모든 축의 모션작업이 완료됨 (정지 상태임)

예제

```
t_i32 nErrCode=0;
t_i32 anAxes[] = {0, 1};
while ( !mcMxIsDone(2, anAxes) )
{
        // 에러 처리..
}
```

▣ mcMxIsDone_Mask

함수 원형

t_bool **mcMxIsDone_Mask** (t_ui32 dwAxisMask)

함수 설명

축 마스크에 포함된 축에 한해서 해당 축들의 모션이 완료됐는지를 체크합니다.

매개 변수

▶ **dwAxisMask:** 축(Axis) 마스크 값

반환값

□ 축 마스크에 포함된 모든 축들의 모션이 완료되었는지 여부

Value	Meaning
0 (false)	하나 이상의 축의 모션작업이 완료되지 않음 (현재 이송이 진행 중에 있음)
1 (true)	모든 축의 모션작업이 완료됨 (정지 상태임)

예제

```
t_i32 nErrCode = 0;
t_i32 nAxisMask = 0x3; // 1 번과 2 번축 포함
if ( !mcMxIsDone_Mask(nAxisMask))
{
        // 에러 처리..
}
```

▣ mcMxStop

함수 원형

t_i32 **mcMxStop** (t_i32 nNumAxes, t_i32 anAxes[], t_bool bDecel, t_bool bWaitDone)

함수 설명

다축구동을 정지시키는 함수입니다.

매개 변수

- ▶ **nNumAxes**: 축 개수
- ▶ **anAxes**: 축 리스트
- ▶ **bDecel:** 감속후 정지할 것인지 여부
- ▶ **bWaitDone:** 완료시까지 기다릴 것인지 여부

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode=0;
t_i32 anAxes[] = {0, 1};

nErrCode = mcMxStop(2, anAxes, cmFALSE, cmTRUE); // 다축 구동중인 모든 축을 감속 없이 정지 시킵니다. 본 함수가 완료될때까지 윈도우 메시지는 Blocking 되며, 완료 후 함수가 리턴됩니다.
```

▣ mcIxSetAxisMap / mcIxGetAxisMap

함수 원형

t_i32 **mcIxSetAxisMap** (t_i32 nMapIndex, t_ui32 dwMapMask)

t_ui32 **mcIxGetAxisMap** (t_i32 nMapIndex)

함수 설명

보간제어를 위한 맵인덱스 설정 및 보간제어에 포함시킬 축(Axis) 마스크를 설정하거나(Set) 읽어옵니다(Get)

매개 변수

▶ **nMapIndex**: 보간맵 인덱스

▶ **dwMapMask:** 보간 제어에 포함시킬 축 마스크

반환값

☐ **mcIxSetAxisMap()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

☐ **mcIxGetAxisMap ()** 함수는 설정되어있는 축 마스크 값을 반환합니다.

예제

```
t_i32 nErrCode=0, nMapIdx=0;
t_i32 anAxes[] = {0, 1, 2, 3};

nErrCode = mcIxSetAxisMap(nMapIdx, (1<<anAxes[0]) | (1<<anAxes[1])); // 0 번축과
1 번축을 보간 맵으로 구성합니다. 결과적으로 map mask 값은 0x3 이 됩니다.
```

▣ mcIxSetVelCorrMode / mcIxGetVelCorrMode

함수 원형

t_i32 **mcIxSetVelCorrMode** (t_i32 nMapIndex, t_i32 nVelCorrOpt1, t_i32 nVelCorrOpt2)

t_i32 **mcIxGetVelCorrMode** (t_i32 nMapIndex, t_i32* pnVelCorrOpt1, t_i32* pnVelCorrOpt2)

함수 설명

슬레이브축의 속도가 정격속도를 초화하는 경우에 정격속도를 기준으로 마스터축의 속도를 보정하는 모드를 설정하거나 얻어오는 함수입니다.

매개 변수

▶ **nMapIndex**: 보간맵 인덱스

▶ **nVelCorrOpt1:** 보정을 할것인지를 결정하는 옵션

0=>보정안함

1=>슬레이브축의 속도가 정격속도를 초과하면 보정(가감속은 보정하지 않음)

2=>슬레이브축의 속도 및 가감속도까지 보정 조건으로 사용

▶ **nVelCorrOpt2:** 이 옵션은 마스터속도모드에서만 적용되는 옵션으로서, 보정을 하는 경우에 정격속도 및 가감속도만을 리미트로 사용할 것인지 정격속도(가감속도)에 보간속도비를 곱한 것을 리미트로 사용할 것인지를 결정하는 옵션

0=>슬레이브축의 속도 및 가속도 리미트를 정격속도와 정격가속도만을 사용,

1=>슬레이브축의 속도 리미트를 정격속도에 보간속도비를 곱한값을 사용한다. 그리고 가감속도 리미트는 정격을 사용

2=>슬레이브축의 속도 및 가속도 리미트를 모두 정격속도와 정격가속도에 보간속도비를 곱한값을 사용(마스터 속도 모드에서만 유효)

반환값

□ 두 함수 모두 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode=0, nMapIdx=0;
mcIxSetSpeedPattern (nMapIdx, cmFALSE, cmSMODE_T, 100, 100, 100); // 마스터 속도 모드일 때에는 정격 작업속도 및 가감속도에 대한 비율로 속도 값을 설정함에 주의
nErrCode = mcIxSetVelCorrMode (nMapIdx, 2, 1); // 슬레이브축의 속도 및 가감속도까지 보정 하며, 슬레이브축의 속도 리미트를 정격속도에 보간속도비를 곱한값을, 그리고 가감속도 리미트는 정격을 사용합니다.
```

▣ mcIxSetSpeedPattern / mcIxGetSpeedPattern

함수 원형

t_i32 **mcIxSetSpeedPattern** (t_i32 nMapIndex, t_i32 bIsVectorSpeed, t_i32 nSpeedMode, t_f32 fVel, t_f32 fAcc, t_f32 fDec)

void **mcIxSetSpeedPattern** (t_i32 nMapIndex, t_i32 *pbIsVectorSpeed, t_i32 *pnSpeedMode, t_f32 *pfVel, t_f32 *pfAcc, t_f32 *pfDec)

함수 설명

보간제어시 속도 패턴을 설정하거나 읽어옵니다.

매개 변수

▶ **nMapIndex**: 보간맵 인덱스

▶ **bIsVectorSpeed**: 벡터속도 모드 사용 여부

0(cmFALSE)=>벡터속도 모드 사용 안함, 마스터 속도 모드로 동작

1(cmTRUE)=>벡터속도 모드 사용함.

▶ nSpeedMode: 속도 모드

Value	Meaning
0	CONSTANT 속도 모드를 의미합니다. 즉, 가감속을 수행하지 않습니다.
1	TRAPEZODIAL 속도모드를 의미합니다. 즉, 사다리꼴의 가감속을 수행합니다.
2	S-CURVE 속도모드를 의미합니다. 즉, S-CURVE 형태의 가감속을 수행합니다.

▶ **fVel:** bIsVectorSpeed 모드가 1이면 작업 속도를, 0이면 작업 속도의 %비율을 사용합니다.

▶ **fAcc:** bIsVectorSpeed 모드가 1이면 가속도를, 0이면 가속도의 %비율을 사용합니다.

▶ **fDec:** bIsVectorSpeed 모드가 1이면 감속도를, 0이면 감속도의 %비율을 사용합니다.

자세한 내용은 예제를 참고 하세요.

반환값

□ **mcIxSetSpeedPattern()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode=0, nMapIdx0=0, nMapIdx1=1;
t_bool bVecSpdMode0 = cmTRUE, bVecSpdMode1 = cmFALSE;

// 아래 보간제어는 벡터속도 모드 이며, 보간제어시의 작업속도, 가감속도를 PPS
단위로 그대로 넣어줍니다.
mcIxSetSpeedPattern (nMapIdx0, bVecSpdMode0, cmSMODE_T, 10000, 100000,
100000);

// 다음 보간제어는 마스터속도 모드이며, 정격속도 (mcSxSpeedPattern 함수로 설정된
각 축별 속도 패턴) 로 설정된 작업 및 가감속도에 대한 비율로 속도 파라미터값을
설정합니다.
mcIxSetSpeedPattern (nMapIdx1, bVecSpdMode1, cmSMODE_T, 100, 90, 90);
```

▣ mcIxLine

함수 원형

t_i32 **mcIxLine** (t_i32 nMapIndex, t_f32 afDistList[], t_bool bWaitDone, t_bool bIsBatchMode)

함수 설명

직선 보간제어를 시작합니다. 상대좌표 이송입니다.

매개 변수

▶ **nMapIndex**: 보간맵 인덱스

▶ **afDistList:** 축별 이송 거리를 담은 배열 값, 보간맵의 축 수와 동일한 배열 크기를 가집니다.

▶ **bWaitDone:** 보간제어가 완료시까지 기다릴지의 여부

▶ **bIsBatchMode:** 리스트모션 모드인지를 설정하는 플래그. 이 것은 소프트웨어적인 보간을 사용할 때 필요한 파라미터인데, 리스트모션을 진행할 때에는 두번째 작업부터는 각 축의 예약 레지스터에 예약되는데 선행 작업의 완료 타이밍이 축마다 약간씩 차이가 날 수 있으므로 예약작업의 이송시작을 동시에 하기 위해서 내부 동기 신호를 사용합니다.

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode=0, nMapIdx=0;
t_f32 afDistList[] = {10000, 20000};
t_bool bVecSpdMode = cmTRUE;

// 아래 보간제어는 벡터속도 모드 이며, 보간제어시의 작업속도, 가감속도를 PPS
단위로 그대로 넣어줍니다.
mcIxSetSpeedPattern (nMapIdx, bVecSpdMode, cmSMODE_T, 10000, 100000, 100000);
mcIxLine (nMapIdx, afDistList, cmTRUE, cmFALSE);
```

▣ mcIxLineTo

함수 원형

t_i32 **mcIxLineTo** (t_i32 nMapIndex, t_f32 afPosList[], t_bool bWaitDone, t_bool bIsBatchMode)

함수 설명

직선 보간제어를 시작합니다. 절대좌표 이송입니다.

매개 변수

▶ **nMapIndex**: 보간맵 인덱스

▶ **afPosList:** 축별 목표 좌표를 담은 배열 값, 보간맵의 축 수와 동일한 배열 크기를 가집니다.

▶ **bWaitDone:** 보간제어가 완료시까지 기다릴지의 여부

▶ **bIsBatchMode:** 리스트모션 모드인지를 설정하는 플래그. 이 것은 소프트웨어적인 보간을 사용할 때 필요한 파라미터인데, 리스트모션을 진행할 때에는 두번째 작업부터는 각 축의 예약 레지스터에 예약되는데 선행 작업의 완료 타이밍이 축마다 약간씩 차이가 날 수 있으므로 예약작업의 이송시작을 동시에 하기 위해서 내부 동기 신호를 사용합니다.

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcIxArc

함수 원형

t_i32 **mcIxArc** (t_i32 nMapIndex, t_float32 fCenPos[], t_float32 fEndPos[], t_i32 nDir, t_bool bWaitDone)

함수 설명

중심점과 끝점을 이용해서 원호보간 이송을 수행합니다. 상대좌표 이송입니다.

매개 변수

▶ **nMapIndex**: 보간맵 인덱스

▶ **fCentPos:** 축별 중심점을 담은 배열 값, 보간맵의 축 수와 동일한 배열 크기를 가집니다.

▶ **fEndPos:** 축별 끝점을 담은 배열 값, 보간맵의 축 수와 동일한 배열 크기를 가집니다.

▶ **nDir:** 회전 방향을 지정합니다.

Value	Meaning
0 (cmARC_CW)	시계 방향(CW) 으로 회전
1 (cmARC_CCW)	반시계 방향(CCW) 으로 회전

▶ **bWaitDone:** 보간제어가 완료시까지 기다릴지의 여부

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode=0, nMapIdx=0;
t_f32 afCentPos[] = {100, 100};
t_f32 afEndPos[] = {2000, 2000};
t_bool bVecSpdMode = cmTRUE;

// 아래 보간제어는 벡터속도 모드 이며, 보간제어시의 작업속도, 가감속도를 PPS 단위로 그대로 넣어줍니다.
mcIxSetSpeedPattern (nMapIdx, bVecSpdMode, cmSMODE_T, 10000, 100000, 100000);
mcIxArc (nMapIdx, afCenPos, afEndPos, cmARC_CW, cmTRUE);
```

▣ mcIxArcTo

함수 원형

t_i32 **mcIxArcTo** (t_i32 nMapIndex, t_float32 fCenPos[], t_float32 fEndPos[], t_i32 nDir, t_bool bWaitDone)

함수 설명

중심점과 끝점을 이용해서 원호보간 이송을 수행합니다. 절대좌표 이송입니다.

매개 변수

▶ **nMapIndex**: 보간맵 인덱스

▶ **fCentPos:** 축별 중심점을 담은 배열 값, 보간맵의 축 수와 동일한 배열 크기를 가집니다.

▶ **fEndPos:** 축별 끝점을 담은 배열 값, 보간맵의 축 수와 동일한 배열 크기를 가집니다.

▶ **nDir:** 회전 방향을 지정합니다.

Value	Meaning
0 (cmARC_CW)	시계 방향(CW) 으로 회전
1 (cmARC_CCW)	반시계 방향(CCW) 으로 회전

▶ **bWaitDone:** 보간제어가 완료시까지 기다릴지의 여부

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcIxArcA

함수 원형

t_i32 **mcIxArcA** (t_i32 nMapIndex, t_f32 afCenPos[], t_f32 fAngle, BOOL bIsAbsPos, t_bool bWaitDone)

함수 설명

중심점과 각도를 이용해서 원호보간 이송을 수행합니다. 상대좌표 이송입니다.

매개 변수

▶ **nMapIndex**: 보간맵 인덱스

▶ **fCentPos:** 축별 중심점을 담은 배열 값, 보간맵의 축 수와 동일한 배열 크기를 가집니다.

▶ **fAngle**: 회전 각

▶ **bIsAbsPos:** 중심점의 좌표를 현재좌표를 원점으로 하는 상대 좌표로 변경할지의 여부

▶ **bWaitDone:** 보간제어가 완료시까지 기다릴지의 여부

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode=0, nMapIdx=0;
t_f32 afCentPos[] = {100, 100};
t_f32 fAngle = 90;
t_bool bVecSpdMode = cmTRUE;

// 아래 보간제어는 벡터속도 모드 이며, 보간제어시의 작업속도, 가감속도를 PPS 단위로 그대로 넣어줍니다.
mcIxSetSpeedPattern (nMapIdx, bVecSpdMode, cmSMODE_T, 10000, 100000, 100000);
mcIxArcA (nMapIdx, afCenPos, fAngle, cmFALSE, cmTRUE);
```

▣ mcIxArc3P

함수 원형

t_i32 **mcIxArc3P** (t_i32 nMapIndex, t_f32 P2[], t_f32 P3[], t_f32 fAngle, BOOL bIsAbsPos, t_bool bWaitDone)

함수 설명

현재 좌표와 추가로 지정하는 두점을 포함한 세점을 이용해서 원호보간 이송을 수행합니다.

매개 변수

- ▶ **nMapIndex**: 보간맵 인덱스
- ▶ **P2:** 두번째 점의 X, Y 좌표
- ▶ **P3:** 세번째 점의 X, Y 좌표
- ▶ **fAngle**: 회전 각
- ▶ **bIsAbsPos:** 중심점의 좌표를 현재좌표를 원점으로 하는 상대 좌표로 변경할지의 여부
- ▶ **bWaitDone:** 보간제어가 완료시까지 기다릴지의 여부

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
// 현재 0 번축과 1 번축의 위치가 각각 0, 0 이라고 가정합니다. => P1(0, 0)
t_i32 nErrCode=0, nMapIdx=0;
t_f32 afPos2[] = {1000, 1000}; // P2(1000, 1000)
t_f32 afPos3[] = {2000, 0}; // P3(2000, 0)
t_f32 fAngle = 180;
t_bool bVecSpdMode = cmTRUE;

// 아래 보간제어는 벡터속도 모드 이며, 보간제어시의 작업속도, 가감속도를 PPS
단위로 그대로 넣어줍니다.
mcIxSetSpeedPattern (nMapIdx, bVecSpdMode, cmSMODE_T, 10000, 100000, 100000);
mcIxArc3P (nMapIdx, afPos2, afPos3, fAngle, cmFALSE, cmTRUE);
```

▣ mcIxSpline

함수 원형

t_i32 **mcIxSpline** (t_i32 nMapIndex, t_float32 **PP, t_i32 n_inp, t_i32 n_div)

함수 설명

스플라인 보간을 셋업하고 시작시키는 함수입니다.

매개 변수

- ▶ **nMapIndex**: 보간맵 인덱스
- ▶ **PP:** 포인트 데이터
- ▶ **n_inp:** Number of input points => 스플라인 보간의 입력 포인트의 수
- ▶ **n_div:** 몇 개의 세그먼트를 생성하여 스플라인을 만들지를 결정하는 개수

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcIxIsDone

함수 원형

t_bool **mcIxIsDone** (t_i32 nMapIndex)

함수 설명

보간제어가 완료되었는지 체크하는 함수입니다.

매개 변수

▶ **nMapIndex**: 보간맵 인덱스

반환값

□ 이 함수는 보간제어 완료 여부를 반환합니다.

Value	Meaning
0 (cmFALSE)	보간 제어 완료되지 않음
1 (cmTRUE)	보간 제어 완료됨

예제

```
t_i32 nErrCode=0, nMapIdx=0;
t_f32 afDistList[] = {10000, 20000};
t_bool bVecSpdMode = cmTRUE;

// 아래 보간제어는 벡터속도 모드 이며, 보간제어시의 작업속도, 가감속도를 PPS
단위로 그대로 넣어줍니다.
mcIxSetSpeedPattern (nMapIdx, bVecSpdMode, cmSMODE_T, 10000, 100000, 100000);
mcIxLine (nMapIdx, afDistList, cmFALSE, cmFALSE); // 함수 수행후 곧바로 리턴된다
Sleep(100);
while(!mcIxIsDone(nMaxIdx))
{
        // 처리..
}
```

▣ mcIxStop

함수 원형

t_i32 **mcIxStop** (t_i32 nMapIndex, t_bool bDecel, t_bool bWaitDone)

함수 설명

보간제어 이송을 정지시키는 함수입니다.

매개 변수

- ▶ **nMapIndex**: 보간맵 인덱스
- ▶ **bDecel:** 감속후 정지할 것인지의 여부
- ▶ **bWaitDone:** 완료될때까지 기다릴 것인지의 여부

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nErrCode=0, nMapIdx=0;
t_f32 afDistList[] = {10000, 20000};
t_bool bVecSpdMode = cmTRUE;

// 아래 보간제어는 벡터속도 모드 이며, 보간제어시의 작업속도, 가감속도를 PPS 단위로 그대로 넣어줍니다.
mcIxSetSpeedPattern (nMapIdx, bVecSpdMode, cmSMODE_T, 10000, 100000, 100000);
mcIxLine (nMapIdx, afDistList, cmFALSE, cmFALSE); // 함수 수행후 곧바로 리턴된다.
mcIxStop(nMapIdx, cmFALSE, cmFALSE); // 감속 없이 즉시 정지 시킨다.
```

▣ mcPlsrSetInMode / mcPlsrSetInMode

함수 원형

t_i32 **mcPlsrSetInMode** (t_i32 nAxis, t_i32 nInputMode, t_bool bIsRevDir)

t_i32 **mcPlsrGetInMode** (t_i32 nAxis, t_i32* pnInputMode, t_bool* pbIsRevDir)

함수 설명

Manual Pulse 입력 신호에 대한 환경을 설정하거나 읽어옵니다.

매개 변수

▶ **nAxis**: 축 번호

▶ **nInputMode:** PA 와 PB 입력 단자를 통하여 입력되는 Pulsar 입력 신호의 입력모드를 설정합니다.

Value	Meaning
0 (cmIMODE_AB1X)	A/B 상 1채배 모드
1 (cmIMODE_AB2X)	A/B 상 2채배 모드
2 (cmIMODE_AB4X)	A/B 상 4채배 모드
3 (cmIMODE_CWCCW)	CW/CCW 펄스

▶ **bIsRevDir:** 1(cmTRUE) 인 경우, 방향을 반대로 적용

반환값

□ 두 함수 모두 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcPlsrSetGain / mcPlsrGetGain

함수 원형

t_i32 **mcPlsrSetGain** (t_i32 nAxis, t_i32 nGainFactor, t_i32 nDivFactor)

t_i32 **mcPlsrGetGain** (t_i32 nAxis, t_i32* pnGainFactor, t_i32* pnDivFactor)

함수 설명

PA/PB 입력 펄스 대비 Command 출력 펄스 수의 비를 사용자가 임의로 조절할 수 있도록 값을 설정 하거나 얻어오는 함수입니다

매개 변수

▶ **nAxis**: 축 번호

▶ **nGainFactor:** PMG 회로에 설정되는 사용자 정수로서 PIM 회로를 거쳐서 생성된 1차 출력 펄스를 1~32배수의 펄스로 재 생성하는 회로입니다. 이 값은 1 ~ 32 사이의 값이어야 합니다. 이 값의 초기 기본값은 1입니다.

▶ **nDivFactor:** PDIV 회로에 설정되는 사용자 정수로서 PMG회로를 거쳐서 생성된 2차 출력 펄스에 (DivFactor/2048)가 곱해져서 최종 출력 펄스를 생성합니다. 이 값은 1 ~ 2048의 값을 설정할 수 있는데 2048을 제외한 나머지 값을 설정하면 결과적으로는 출력 펄스의 수를 줄이는 효과를 내므로 나누기 회로의 역할을 수행합니다. 이 값의 초기 기본값은 2048입니다

반환값

□ 두 함수 모두 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

참고

□ Pulsar신호는 [PIM 회로] -> [PMG 회로] -> [PDIV 회로를 거쳐서 Command 펄스로 출력됩니다. PIM회로는 채배회로이고, PMG는 곱셈회로, PDIV는 분주(나누기) 회로입니다.

▣ mcPlsrHomeMoveStart

함수 원형

t_i32 **mcPlsrHomeMoveStart** (t_i32 nAxis, t_i32 nHomeType)

함수 설명

Pulsar Input에 의한 원점 복귀 작업을 수행합니다. 원점 복귀 모드(Home Type)에 따라 원점 복귀가 완료되거나 cmmSxStopEmg() 함수가 호출되면 모션을 종료합니다.

매개 변수

▶ **nAxis**: 축 번호

▶ **nHomeType:** Pulsar Input에 의해 원점 복귀를 수행하는 모드를 설정합니다. 이 값은 다음 중 하나의 값이어야 합니다.

Value	Meaning
0	Command 카운터가 0이 될때 원점복귀를 종료합니다.
1	ORG 신호가 ON이 되면 원점복귀를 종료합니다.

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcPlsrJogStart

함수 원형

t_i32 **mcPlsrJogStart** (t_i32 nAxis)

함수 설명

Manual Pulse 구동에 의한 속도 이송명령을 수행합니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcPlsrMove

함수 원형

t_i32 **mcPlsrMove** (t_i32 nAxis, t_f32 fDistance, t_bool bWaitDone)

함수 설명

Manual Pulse 구동에 의한 상대좌표 이송명령을 수행합니다.

매개 변수

- ▶ **nAxis**: 축 번호
- ▶ **fDistance**: 이송 거리
- ▶ **bWaitDone:** 완료시 까지 기다릴것인지 여부

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcPlsrMoveTo

함수 원형

t_i32 **mcPlsrMoveTo** (t_i32 nAxis, t_f32 fPosition, t_bool bWaitDone)

함수 설명

Manual Pulse 구동에 의한 절대좌표 이송명령을 수행합니다.

매개 변수

- ▶ **nAxis**: 축 번호
- ▶ **fPosition**: 목표 좌표
- ▶ **bWaitDone:** 완료시 까지 기다릴것인지 여부

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcPlsrIsActive

함수 원형

t_bool **mcPlsrIsActive** (t_i32 nAxis)

함수 설명

PA/PB를 사용하는 이송모드가 설정되어 있는지 체크합니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcOverrideSpeedSet

함수 원형

t_i32 **mcOverrideSpeedSet** (t_i32 nAxis)

함수 설명

해당축에 속도오버라이드 기능을 설정합니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nAxis = 0;

// 0 번 축의 속도를 설정합니다.
mcSxSetSpeedPattern(nAxis, cmSMODE_T, 10000, 100000, 100000);

// 0 번 축을 구동합니다.
mcSxMove(nAxis, 100000, cmFALSE);

// 오버라이드 할 축 및 속도를 지정합니다.
mcSxSetSpeedPattern(nAxis, cmSMODE_T, 1000, 10000, 10000);

// mcSxSetSpeedRatio 함수를 이용하여 오버라이드 속도를 설정할 수 있습니다 //
// mcSxSetSpeedRatio (nAxis, cemSMODE_KEEP, 70, 70, 70);

// 속도를 오버라이드 합니다.
mcOverrideSpeedSet (nAxis);
```

▣ mcOverrideMove

함수 원형

t_i32 **mcOverrideMove** (t_i32 nAxis, t_f32 fNewDist, t_bool bIsHardApply, t_i32 *pnState)

함수 설명

해당축에 오버라이드된 새로운 속도로 상대좌표 이송을 수행합니다.

매개 변수

▶ **nAxis**: 축 번호

▶ **fNewDist:** 새로운 이송 거리

▶ **bIsHardApply:** 현재 이송이 완료되고 있지 않은 상태인 경우에도 이송을 적용할지를 설정

Value	Meaning
0	현재 이송이 진행중일 때만 이송 오버라이드를 하고, 그렇지 않은 경우에는 아무런 동작을 하지 않습니다.
1	현재 이송이 진행중일 때만 이송 오버라이드를 수행하고, 그렇지 않은 경우에는 새로운 이송 작업을 시작합니다.

▶ **pnState:** 적용 상태를 반환 받을 포인터. 이 포인터에 반환되는 값의 의미는 다음과 같다.

Value	Meaning
0	이송 오버라이드가 수행됩니다.
1	이미 이송이 완료되었거나 정지 상태이므로 이송 오버라이드가 생략됩니다.
2	이미 이송이 완료되었거나 정지 상태이므로 새로운 이송명령으로 처리합니다.

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nAxis = 0;

// 0 번 축의 속도를 설정합니다.
mcSxSetSpeedPattern(nAxis, cmSMODE_S, 5000, 30000, 30000);
mcSxSetSpeedRatio (nAxis, cemSMODE_KEEP, 100, 100, 100);

// 0 번 축을 구동합니다.
mcSxMove(nAxis, 10000, cmFALSE);

t_i32 nAppliedState = 0;
/* 이동할 거리를 20000 으로 오버라이드 합니다. bIsHardApply 인자를 cmTRUE 로
할 경우에는 모션이 종료된 상태에서도 강제로 이동시킵니다. */
mcOverrideMove (nAxis, 20000, cmTRUE, &nAppliedState);
```

▣ mcOverrideMoveTo

함수 원형

t_i32 **mcOverrideMoveTo** (t_i32 nAxis, t_f32 fNewPos, t_bool bIsHardApply, t_i32 *pnState)

함수 설명

해당축에 오버라이드된 새로운 속도로 절대좌표 이송을 수행합니다.

매개 변수

▶ **nAxis**: 축 번호

▶ **fNewPos:** 새로운 목표 좌표

▶ **bIsHardApply:** 현재 이송이 완료되고 있지 않은 상태인 경우에도 이송을 적용할지를 설정

Value	Meaning
0	현재 이송이 진행중일 때만 이송 오버라이드를 하고, 그렇지 않은 경우에는 아무런 동작을 하지 않습니다.
1	현재 이송이 진행중일 때만 이송 오버라이드를 수행하고, 그렇지 않은 경우에는 새로운 이송 작업을 시작합니다.

▶ **pnState:** 적용 상태를 반환 받을 포인터. 이 포인터에 반환되는 값의 의미는 다음과 같다.

Value	Meaning
0	이송 오버라이드가 수행됩니다.
1	이미 이송이 완료되었거나 정지 상태이므로 이송 오버라이드가 생략됩니다.
2	이미 이송이 완료되었거나 정지 상태이므로 새로운 이송명령으로 처리합니다.

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
// mcOverrideMove() 함수를 참고 하세요
```

▣ mcStSetCount / mcStGetCount

함수 원형

t_i32 **mcStSetCount** (t_i32 nAxis, t_i32 nSource, t_i32 nCount)

t_i32 **mcStGetCount** (t_i32 nAxis, t_i32 nSource)

함수 설명

mcStSetCount() 함수는 지정한 축의 지령펄스(command pulse) 또는 궤환펄스(feedback pulse)의 카운트를 임의의 값으로 설정하는 함수입니다. 이때 지정하는 카운터값의 단위는 펄스수입니다.

mcStGetCount() 함수는 지정한 축의 지령펄스(command pulse) 또는 궤환펄스(feedback pulse)의 카운트값을 읽어들이는 함수입니다. 이때 카운트의 값은 펄스수입니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **nSource:** 대상 카운터 종류를 나타내는 아이디값 . 이 값은 다음의 4가지 값중의 하나이어야 합니다.

Value	Meaning
0 (cmCNT_COMM)	지령펄스(command pulse) 카운터
1 (cmCNT_FEED)	궤환펄스(feedback pulse) 카운터
2 (cmCNT_DEV)	편차카운터(deviation counter): command count와 feedback count의 편차를 세는 카운터
3 (cmCNT_GEN)	General counter: 사용자의 정의에 따라 여러가지 용도로 사용될 수 있는 카운터

▶ **nCount:** 새로이 설정할 카운터의 값. 이 값은 논리거리 단위가 아닌 펄스 단위로 설정합니다.

반환값

□ **mcStSetCount ()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

□ **mcStGetCount ()** 함수는 지정한 카운터의 카운트값을 반환합니다.

예제

```
t_i32 nAxis = 0;
t_i32 nCmdCount = 0, nFeedCount = 0;

// 0 번 축의 Command Counter 를 0 으로 설정합니다.
mcStSetCount (nAxis, cmCNT_COMM, 0);

// 0 번 축의 Feedback Counter 를 0 으로 설정합니다.
mcStSetCount (nAxis, cmCNT_FEED, 0);

// 0 번 축의 Command Counter 값을 얻어옵니다.
mcStGetCount (nAxis, cmCNT_COMM, &nCmdCount);

// 0 번 축의 Feedback Counter 값을 얻어옵니다.
mcStGetCount (nAxis, cmCNT_FEED, &nFeedCount);
```

▣ mcStSetPosition / mcStGetPosition

함수 원형

t_i32 **mcStSetPosition** (t_i32 nAxis, t_i32 nSource, t_f32 fPosition)

t_f32 **mcStGetPosition** (t_i32 nAxis, t_i32 nSource)

함수 설명

mcStSetPosition() 함수는 지정한 축의 지령위치(command position) 또는 궤환위치(feedback position) 값을 임의의 값으로 설정하는 함수입니다. 이때 지정하는 위치값의 단위는 논리거리 단위를 사용합니다.

mcStGetPosition() 함수는 지정한 축의 지령위치(command pulse) 또는 궤환위치(feedback pulse)의 현재값을 읽어들이는 함수입니다. 이때 위치값의 단위는 논리거리 단위입니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **nSource:** 대상 카운터 종류를 나타내는 아이디값 . 이 값은 다음의 4가지 값중의 하나이어야 합니다.

Value	Meaning
0 (cmCNT_COMM)	지령펄스(command pulse) 카운터
1 (cmCNT_FEED)	궤환펄스(feedback pulse) 카운터
2 (cmCNT_DEV)	편차카운터(deviation counter): command count와 feedback count의 편차를 세는 카운터
3 (cmCNT_GEN)	General counter: 사용자의 정의에 따라 여러가지 용도로 사용될 수 있는 카운터

▶ **fPosition:** 새로이 설정할 위치값. 이 값의 단위는 논리거리를 사용합니다.

반환값

☐ **mcStSetPosition()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

☐ **mcStGetPosition()** 함수는 지정한 카운터의 위치값을 반환합니다. 이 값의 단위는 논리거리 단위를 사용합니다.

예제

```
t_i32 nAxis = 0;
t_i32 nCmdPos = 0, nFeedPos = 0;

// 0 번 축의 Command Position 을 0 으로 설정합니다.
mcStSetPosition (nAxis, cmCNT_COMM, 0);

// 0 번 축의 Feedback Position 을 0 으로 설정합니다.
mcStSetPosition (nAxis, cmCNT_FEED, 0);

// 0 번 축의 Command Position 값을 얻어옵니다.
mcStGetPosition (nAxis, cmCNT_COMM, &nCmdPos);

// 0 번 축의 Feedback Position 값을 얻어옵니다.
mcStGetPosition (nAxis, cmCNT_FEED, &nFeedPos);
```

▣ mcStGetSpeed

함수 원형

t_f32 **mcStGetSpeed** (t_i32 nAxis, t_i32 nSource)

함수 설명

mcStGetSpeed() 함수는 현재 실제로 이송되고 있는 속도를 반환합니다. nSource 파라미터를 통해서 지령 속도와 궤환 속도를 모두 이 함수를 통해서 읽을 수 있습니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

▶ **nSource:** 대상 카운터 종류를 나타내는 아이디값 .

Value	Meaning
0 (cmCNT_COMM)	지령속도(command speed)
1 (cmCNT_FEED)	궤환속도(feedback speed)

반환값

mcStGetSpeed() 함수는 지정한 축의 command 또는 feedback 속도를 반환합니다.

예제

```
t_i32 nAxis = 0;
t_i32 nCmdSpd = 0, nFeedSpd = 0;

// 0 번 축의 Command Speed 값을 얻어옵니다.
nCmdSpd = mcStGetSpeed (nAxis, cmCNT_COMM);

// 0 번 축의 Feedback Speed 값을 얻어옵니다.
nFeedSpd = mcStGetSpeed (nAxis, cmCNT_FEED);
```

▣ mcStGetMst

함수 원형

t_i32 **mcStGetMst** (t_i32 nAxis)

함수 설명

mcStGetMst() 함수는 현재 모션의 동작 상태를 반환합니다.

매개 변수

▶ **nAxis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

반환값

현재 모션의 동작 상태를 나태는 정수값을 반환합니다.

Value	Meaning
음수	직전의 모션 이송 명령 수행중에 에러가 발생함. 이 값은 에러코드와 일치합니다. 그리고 한번 읽으면 클리어(Clear)되어 다음에 다시 읽으면 nMstCode가 0이 됩니다.
0 (cmMST_STOP)	모션이 정지되어 있는 상태
1 (cmMST_WAIT_DR)	DR 신호 입력을 기다리고 있다.
2 (cmMST_WAIT_STA)	STA 신호 입력을 기다리고 있다.
3 (cmMST_WAIT_INSYNC)	내부동기신호(INSYNC)를 기다리고 있다.
4 (cmMST_WAIT_OTHER)	다른 축이 정지하기를 기다리고 있다.
5 (cmMST_WAIT_ERC)	ERC 출력 타이머가 완료되기를 기다리고 있다.
6 (cmMST_WAIT_DIR)	DIR 변경 타이머가 완료되기를 기다리고 있다.
7 (cmMST_RESERVED1)	백래쉬 보정을 수행중이다.
8 (cmMST_WAIT_PLSR)	PA/PB 신호 입력을 기다리고 있다.
9 (cmMST_IN_RVSSPD)	원점복귀의 역행속도(Reverse velocity)로 이송 중이다.
10 (cmMST_IN_INISPD)	초기속도로 이송 중이다.

11 (cmMST_IN_ACC)	가속 중이다.
12 (cmMST_IN_WORKSPD)	작업속도로 이송 중이다.
13 (cmMST_IN_DEC)	감속 중이다.
14 (cmMST_WAIT_INP)	INP 신호가 ON이 되기를 기다리고 있다.

예제

```
t_i32 cnmStGetMst(t_i32 nAxis)
{
        t_i32 nMstCode;
        nMstCode = mcStGetMst(nAxis);
        if(nMstCode < 0) return (nMstCode);
        if(mcHomeIsBusy(nAxis)) return 16;
        else return (nMstCode);
}
```

▣ mcStGetMio

함수 원형

t_ui32 **mcStGetMio** (t_i32 nAxis)

함수 설명

mcStGetMio() 함수는 현재 모션과 관련된 여러가지 I/O 상태를 나타내는 32비트 정수값을 반환합니다. 이 값은 각 비트별로 할당된 I/O핀의 상태를 표시하므로 사용자는 비트마스크를 수행하여 원하는 I/O핀의 상태를 확인하여야 합니다.

매개 변수

▶ **hMicomm:** 통신핸들. 이 값은 miOpenUdp()의 반환값을 사용합니다.

▶ **Axis:** 축 번호를 지정합니다. 축 번호는 0부터 시작합니다.

반환값

Bit No.	Meaning
0 (cmIOST_RDY)	Servo ready signal input status (1 = ON)
1 (cmIOST_ALM)	Alarm signal status (1 = ON)
2 (cmIOST_ELN)	Negative limit switch status (1 = ON)
3 (cmIOST_ELP)	Positive limit switch status (1 = ON)
4 (cmIOST_ORG)	Origin switch status (1 = ON)
5 (cmIOST_DIR)	Operating direction status (1 = ON)
6 (cmIOST_EZ)	Index signal status (1 = ON)
7 (cmIOST_LTC)	Latch signal input status (1 = ON)
8 (cmIOST_SD)	Slow Down signal input status (1 = ON)
9 (cmIOST_INP)	In-Position signal input status (1 = ON)
10 (cmIOST_DRN)	Negative Direction input signal stauts (1 = ON)
11 (cmIOST_DRP)	Positive Direction input signal status (1 = ON)
12 (cmIOST_STA)	Start input signal status (1 = ON)
13 (cmIOST_STP)	Stop input signal status (1 = ON)
14 (cmIOST_ALMR)	Alarm Reset signal input status (1 = ON)
15 (cmIOST_EMG)	Emergency signal input status (1 = ON)
16 (cmIOST_SVON)	Servo ON signal input status (1 = ON)

예제

```
t_i32 nTemp;
nTemp = mcStGetMio(nAxis);
#if         (_PLATFORM==PLF_miCUBE || _PLATFORM==PLF_miCUBE_E)
            dwTemp = soGetMulti(j*4, 4) & 0xf;
            dwTemp <<= 4;
            dwTemp |= (siGetMulti(j*4, 4) & 0xf);
            nTemp = (nTemp & 0xffffff) | ((dwTemp<<24) & 0xff000000); // MIO + SYS_IO
#elif (_PLATFORM==PLF_EtherIP_1)
            dwTemp = soGetMulti(j*3, 3) & 0xf;
            dwTemp <<= 4;
            dwTemp |= (siGetMulti(j*3, 3) & 0xf);
            nTemp = (nTemp & 0xffffff) | ((dwTemp<<24) & 0xff000000); // MIO + MOT_DIO
#endif
```

▣ mcErrClearAxisError

함수 원형

t_i32 **mcErrClearAxisError** (t_i32 nAxis)

함수 설명

REST를 대표하는 에러코드 변수를 클리어합니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcErrGetAxisError

함수 원형

t_i32 **mcErrGetAxisError** (t_i32 nAxis)

함수 설명

각 축에 모션 에러가 발생했는지 체크하는 함수입니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcErrGetAxisError

함수 원형

t_i32 **mcErrGetLastError** (void)

함수 설명

마지막에 발생한 에러코드를 반환합니다.

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcErrGetErrorString

함수 원형

void **mcErrGetErrorString** (t_i32 nErrCode, t_char* szBuffer)

함수 설명

에러코드에 해당하는 에러메시지를 얻는 함수입니다.

매개 변수

- ▶ **nErrCode**: 에러코드 값
- ▶ **szBuffer:** 에러코드에 해당하는 에러스트링 값

반환값

- □ 없음

▣ mcErrClearLastError

함수 원형

void **mcErrClearLastError** (void)

함수 설명

마지막에 발생한 에러코드를 '에러없음'으로 초기화합니다.

반환값

□ 없음

▣ mcIntSetMask / mcIntGetMask

함수 원형

t_i32 **mcIntSetMask** (t_i32 nAxis, t_ui32 dwMask)

t_ui32 **mcIntSetMask** (t_i32 nAxis)

함수 설명

인터럽트 마스크를 설정하거나 얻어옵니다.

매개 변수

▶ **nAxis**: 축 번호

▶ **dwMask:** 인터럽트 마스크 값

반환값

☐ **mcIntSetMask()** 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

☐ **mcIntGetMask()** 함수는 설정된 인터럽트 마스크값을 반환합니다.

▣ mcISR

함수 원형

void **mcISR** (void)

함수 설명

mcISR() 함수는 시스템 함수로서 모션장치의 인터럽트 서비스 루틴(interrupt service routine, ISR)에서 처리해야 하는 작업들이 내장되어 있는 함수입니다.

사용자는 반드시 interrupt **#4**의 서비스 루틴 함수인 **c_int04()** 에서 이 함수를 호출해주어야 합니다.

예제

```
// EBUS(Motion) INT.
interrupt void USER_XINT2_ISR(void)                    // 0x000D48   XINT2
{
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;     // Must acknowledge the PIE group

        mcISR();
}
```

▣ mcTimerCallback

함수 원형

void **mcTimerCallback** (void)

함수 설명

mcTimerCallback() 함수는 타이머 인터럽트 서비스루틴에서 처리해주어야 하는 모션제어 모듈 관련 작업들이 내장되어 있는 함수입니다.

사용자는 반드시 1ms 주기를 가지는 타이머 인터럽트 서비스 루틴(**c_int09()**)에서 이 함수를 호출해주어야 합니다.

예제

```
//-----------------------------------------------------------------------------------------
// mkcTimerCallback(): 1ms 간격의 타이머 인터럽트에서 호출되어져야 하는 콜백 함수.
만일 이 함수를 호출하는
// 타이머 인터럽트의 간격이 1ms 가 아닌 경우에는 이 안의 내용들을 적절히
수정하여야 한다.
//-----------------------------------------------------------------------------------------
void mkcTimerCallback(void)
{
        g_ulTimer++; // 내부적으로 사용되는 mkcTimerTick() 함수를 위해서...
        mcTimerCallback();
}
```

▣ mcLtcIsLatched

함수 원형

t_bool **mcLtcIsLatched** (t_i32 nAxis)

함수 설명

래치카운터에 값이 들어왔는지 확인합니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 래치된 상태를 얻어옵니다.

Value	Meaning
0 (cmFALSE)	래치 안됨
1 (cmTRUE)	래치 됨

▣ mcLtcReadLatch

함수 원형

t_f32 **mcLtcReadLatch** (t_i32 nAxis, t_i16 nCounter)

함수 설명

래치카운터의 종류에 따라 카운터 값을 읽습니다.

매개 변수

▶ **nAxis**: 축 번호

▶ **nCounter:** 래치카운터 종류

Value	Meaning
0 (cmCNT_COMM)	Command Counter
1 (cmCNT_FEED)	Feedback Counter
2 (cmCNT_DEV)	Deviation Counter
3 (cmCNT_GEN)	General Counter

반환값

□ 래치 카운트 값을 반환합니다.

▣ mcLtcQue_Alloc

함수 원형

t_i32 **mcLtcQue_Alloc**(t_i32 nAxis, t_ui32 nQueSize, t_ui16 nLtcSrcCntr)

함수 설명

Latch-que 버퍼 할당.

매개 변수

- ▶ **nAxis**: 축 번호
- ▶ **nQueSize:** 큐 버퍼 크기
- ▶ **nLtcSrcCntr:** 저장할 대상 카운터.

Value	Meaning
0 (cmCNT_COMM)	Command Counter
1 (cmCNT_FEED)	Feedback Counter
2 (cmCNT_DEV)	Deviation Counter
3 (cmCNT_GEN)	General Counter

반환값

□ 에러코드.

▣ mcLtcQue_Free

함수 원형

t_i32 **mcLtcQue_Free**(t_ui16 nAxis)

함수 설명

Latch-que 버퍼 해제.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 에러 코드.

▣ mcLtcQue_GetSize

함수 원형

t_ui32 **mcLtcQue_GetSize**(t_ui16 nAxis)

함수 설명

Latch-que 버퍼 크기 반환.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 지정된 축의 현재 할당된 Latch-que 버퍼 크기.

▣ mcLtcQue_Reset

함수 원형

t_i32 **mcLtcQue_Reset** (t_ui16 nAxis)

함수 설명

Que의 push & pop count를 모두 리셋합니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 에러 코드.

▣ mcLtcQue_Check

함수 원형

t_ui32 **mcLtcQue_Check** (t_ui16 nAxis)

함수 설명

Que에 읽혀지지 않은 데이터가 몇 개 남아 있는지를 알아보는 함수.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 현재 큐에 있는 읽혀지지 않은 데이터 수.

▣ mcLtcQue_Pop

함수 원형

t_f32 **mcLtcQue_Pop** (t_ui16 nAxis, t_i32 *pnErrCode)

함수 설명

Que에서 읽혀지지 않은 데이터를 반환한다. 이때 pop-count는 자동으로 증가한다.

매개 변수

- ▶ **nAxis**: 축 번호
- ▶ **pnErrCode:** 에러 코드

반환값

□ 큐에서 새로 읽혀진 데이터 값.

▣ mcLtcQue_GetAt

함수 원형

t_f32 **mcLtcQue_GetAt** (t_ui16 nAxis, t_ui32 nIndex, t_i32 *pnErrCode)

함수 설명

지정된 인덱스에 해당하는 Latch Que 버퍼의 데이터를 반환한다.

매개 변수

- ▶ **nAxis**: 축 번호
- ▶ **nIndex**: 읽고자 하는 Latch Que 버퍼 인덱스
- ▶ **pnErrCode:** 에러 코드

반환값

□ 지정된 인덱스에 해당하는 Latch Que 버퍼의 데이터.

▣ mcCmpErr_Set / mcCmpErr_Get

함수 원형

t_i32 **mcCmpErr_Set** (t_i32 nAxis, t_f32 fCmpData, t_i32 nCmpAction)

t_i32 **mcCmpErr_Set** (t_i32 nAxis, t_f32* pfCmpData, t_i32* pnCmpAction)

함수 설명

error comparator 환경을 설정하거나 읽어옵니다.

매개 변수

▶ **nAxis**: 축 번호

▶ **fCmpData**: 비교 데이터

▶ **nCmpAction:** 비교기 조건이 만족하였을 때 컨트롤러가 취하는 조치 모드

Value	Meaning
0	no action
1	immediate stop
2	deceleration stop

반환값

□ 두 함수 모두 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcCmpGen_Set / mcCmpGen_Get

함수 원형

t_i32 **mcCmpGen_Set** (t_i32 nAxis, t_i32 nCmpSrc, t_i32 nCmpMethod, t_f32 fCmpData, t_i32 nCmpAction)

t_i32 **mcCmpGen_Get** (t_i32 nAxis, t_i32 *pnCmpSrc, t_i32 *pnCmpMethod, t_f32 *pfCmpData, t_i32 *pnCmpAction)

함수 설명

general comparator 환경 값을 설정하거나 읽어옵니다.

매개 변수

▶ **nAxis**: 축 번호

▶ **nCmpSrc:** 비교기 카운터 소스

Value	Meaning
0 (cmCNT_COMM)	Command Counter
1 (cmCNT_FEED)	Feedback Counter
2 (cmCNT_DEV)	Deviation Counter
3 (cmCNT_GEN)	General Counter

▶ **nCmpMethod**: 비교 조건

Value	Meaning
0	disable comparator
1	fCmpData == source counter (regardless of counting direction)
2	fCmpData == source counter (while counting up)
3	fCmpData == source counter (while counting down)
4	fCmpData > source counter
5	fCmpData < source counter

▶ **fCmpData**: 비교 데이터

▶ **nCmpAction:** 비교기 조건이 만족하였을 때 컨트롤러가 취하는 조치 모드

Value	Meaning
0	no action
1	immediate stop
2	deceleration stop

반환값

□ 두 함수 모두 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcCmpTrg_SetCfg / mcCmpTrg_GetCfg

함수 원형

t_i32 **mcCmpTrg_SetCfg** (t_i32 nAxis, t_i32 nCmpSrc, t_i32 nCmpMethod)

t_i32 **mcCmpTrg_GetCfg** (t_i32 nAxis, t_i32 *pnCmpSrc, t_i32 *pnCmpMethod)

함수 설명

위치비교 출력 대상카운터 소스 종류 및 비교방법을 설정하거나 읽어옵니다.

매개 변수

▶ **nAxis**: 축 번호

▶ **nCmpSrc:** 비교기 카운터 소스

Value	Meaning
0 (cmCNT_COMM)	Command Counter
1 (cmCNT_FEED)	Feedback Counter
2 (cmCNT_DEV)	Deviation Counter
3 (cmCNT_GEN)	General Counter

▶ nCmpMethod: 비교 조건

Value	Meaning
0	disable comparator
1	fCmpData == source counter (regardless of counting direction)
2	fCmpData == source counter (while counting up)
3	fCmpData == source counter (while counting down)
4	fCmpData > source counter
5	fCmpData < source counter

반환값

□ 두 함수 모두 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcCmpTrg_SetData / mcCmpTrg_GetData

함수 원형

t_i32 **mcCmpTrg_SetData** (t_i32 nAxis, t_f32 fCmpData)

t_i32 **mcCmpTrg_GetData** (t_i32 nAxis, t_f32 *pfCmpData)

함수 설명

위치비교 출력을 위한 비교데이터를 설정하거나 읽어옵니다.

매개 변수

▶ **nAxis**: 축 번호

▶ **fCmpData**: 비교 데이터

반환값

□ 두 함수 모두 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcMs_RegSlave

함수 원형

t_i32 **mcMs_RegSlave(t_i32 nAxis, t_f32 fMaxSpeed, t_bool bIsRevDir)**

함수 설명

마스터 슬레이브 구동을 위한 슬레이브 축을 등록합니다.

매개 변수

- ▶ **nAxis**: 축 번호
- ▶ **fMaxSpeed**: 최대 속도를 설정하며, 이 값은 항상 마스터 축의 속도보다 커야 합니다.
- ▶ **bIsRevDir**: 피드백 입력 모드에 대한 CW/CCW 방향을 바꿀지 여부

반환값

□ 에러코드.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcMs_UnregSlave

함수 원형

t_i32 **mcMs_UnregSlave**(t_i32 nAxis)

함수 설명

마스터 슬레이브 구동에 대한 슬레이브 축을 해제합니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 에러코드.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcMs_CheckSlaveState

함수 원형

t_i32 **mcMs_CheckSlaveState**(t_i32 nAxis)

함수 설명

슬레이브 축의 등록 상태를 체크합니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 슬레이브 축 등록 상태.

Value	Meaning
-1	Slave 등록은 되어 있으나 Manual Pulsar 모드가 해제된 경우로 마스터 슬레이브 구동이 정상 수행 안됨.
0	Slave 로 등록되지 않음
1	정상 등록 됨

▣ mcMs_GetMaster

함수 원형

t_i32 **mcMs_GetMaster**(t_i32 nAxis)

함수 설명

슬레이브 축에 대한 마스터 축을 얻습니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 마스터 축 번호.

▣ mcAdvGetNumDevices

함수 원형

t_i32 **mcAdvGetNumDevices** (void)

함수 설명

모션 모듈의 개수를 얻어오는 함수입니다.

반환값

□ 사용 가능한 모션 모듈의 개수

예제

```
t_ui32 nNumModules;
////////// Motion Module Info ////////////////////////////
nNumModules = mcAdvGetNumDevices ();
commWriteDword_(st, nSockChan, nNumModules);
```

▣ mcAdvGetMotDevInfo

함수 원형

t_bool **mcAdvGetMotDevInfo** (t_i32 nDevIdx, TMotDevice* pMotDevBuf)

함수 설명

모션 모듈의 각종 정보를 얻어오는 함수입니다.

매개 변수

- ▶ **nDevIdx:** 모션 모듈 인덱스 (0 부터 시작합니다)
- ▶ **pMotDevBuf**: 모션 모듈 정보

반환값

□ 함수 실행 결과 성공 또는 실패 값

Value	Meaning
0 (cmFALSE)	함수 수행 실패, 해당 인덱스의 모션 모듈 정보 얻기 실패
1 (cmTRUE)	함수 수행 성공.

예제

```
TMotDevice MotDev;
for(i=0; i<nNumModules; i++){
        commWriteByte(st, nSockChan, cnmGetModuleId(i)); // Module ID
        mcAdvGetMotDevInfo(i, &MotDev);
        commWriteByte(st, nSockChan, cnIOMT_MOT); // Device type
        commWriteByte(st, nSockChan, MotDev.nNumAxes&0xff); // DI channel count
        commWriteByte(st, nSockChan, (MotDev.nNumAxes>>8)&0xff); // reserved
}
```

▣ mcAdvStaTrigger

함수 원형

t_i32 **mcAdvStaTrigger** (t_i32 nAxis)

함수 설명

STA 트리거 신호를 출력하는 함수입니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcAdvErcOut

함수 원형

t_i32 **mcAdvErcOut** (t_i32 nAxis)

함수 설명

ERC 신호를 출력하는 함수입니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ mcAdvErcReset

함수 원형

t_i32 **mcAdvErcReset** (t_i32 nAxis)

함수 설명

ERC RESET 명령을 수행합니다.

매개 변수

▶ **nAxis**: 축 번호

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

▣ siGetOne

함수 원형

t_bool **siGetOne** (t_ui32 nChannel)

함수 설명

단일 채널의 입력 센서 상태를 얻어오는 함수입니다.

모션 모듈에서 제공하는 범용 Digital Input Channel 은 일반적으로 외부로부터의 신호 입력, 예를들어 Emergency Stop 신호 또는 기타 센서 입력을 받아서 처리하도록 사용되어집니다.

매개 변수

▶ **nChannel**: 채널 번호

반환값

□ 이 함수는 수행결과를 반환합니다.

Value	Meaning
1	On
0	Off

▣ siGetMulti

함수 원형

t_i32 **siGetMulti**(t_ui32 nIniChan, t_ui32 nNumChan)

함수 설명

여러 채널의 입력 센서 상태를 얻어오는 함수입니다. (최대 32채널 센서 상태 동시 확인)

모션 모듈에서 제공하는 범용 Digital Input Channel 은 일반적으로 외부로부터의 신호 입력, 예를들어 Emergency Stop 신호 또는 기타 센서 입력을 받아서 처리하도록 사용되어집니다.

매개 변수

▶ nIniChan: 시작 채널 번호

▶ nNumChan: 시작 채널 번호부터의 채널 개수

반환값

□ 이 함수는 채널별 수행결과를 32비트 값으로 반환합니다.

Value (예)	Meaning
0x0	모두 Off 상태
0xFFFFFFFF	모두 On 상태
0xAAAAAAAA	10101010101010101010101010101010 이므로 짝수채널만 On 상태

▣ soPutOne

함수 원형

void **soPutOne** (t_ui32 nChannel)

함수 설명

단일 채널을 통해 디지털 신호를 출력하는 함수입니다.

모션 모듈에서 제공하는 범용 Digital Output Channel 은 일반적으로 서보 드라이버용 출력 신호로 사용되어집니다. 예를들어 서보 드라이버의 알람발생 시 알람 리셋(Clear) 신호 입력을 위해 Digital 신호 출력 채널로 사용할 수 있습니다. 이 외에도 테스트 목적으로 원점 센서입력(ORG) 또는 리미트 신호 입력(-EL, +EL) 에 수동으로 Digital 신호 출력채널을 통해 원점센서 또는 리미트 센서가 감지된 것처럼 테스트 할 경우 사용되기도 합니다.

매개 변수

▶ **nChannel**: 채널 번호

반환값

□ 없음

▣ soGetOne

함수 원형

t_bool **soGetOne** (t_ui32 nChannel)

함수 설명

단일 채널을 통해 출력된 디지털 신호를 확인하는 함수입니다.

매개 변수

▶ **nChannel**: 채널 번호

반환값

□ 해당 채널의 출력 상태

Value	Meaning
1	On
0	Off

▣ soPutMulti

함수 원형

void **soPutMulti**(t_ui32 nIniChan, t_ui32 nNumChan, t_ui32 dwStates)

함수 설명

여러 채널을 통해 디지털 신호를 출력하는 함수입니다 (최대 32채널 동시 출력 가능)

모션 모듈에서 제공하는 범용 Digital Output Channel 은 일반적으로 서보 드라이버용 출력 신호로 사용되어집니다. 예를들어 서보 드라이버의 알람발생 시 알람 리셋(Clear) 신호 입력을 위해 Digital 신호 출력 채널로 사용할 수 있습니다. 이 외에도 테스트 목적으로 원점 센서입력(ORG) 또는 리미트 신호 입력(-EL, +EL) 에 수동으로 Digital 신호 출력채널을 통해 원점센서 또는 리미트 센서가 감지된 것처럼 테스트 할 경우 사용되기도 합니다.

매개 변수

- ▶ nIniChan: 시작 채널 번호
- ▶ nNumChan: 시작 채널 번호부터의 채널 개수
- ▶ dwStates: 출력할 32비트 정수값 (각비트는 각채널에 해당)

반환값

- □ 없음

▣ soGetMulti

함수 원형

t_ui32 **soPutMulti**(t_ui32 nIniChan, t_ui32 nNumChan)

함수 설명

여러 채널을 통해 출력된 디지털 신호를 확인하는 함수입니다. (최대 32채널 동시 확인)

매개 변수

▶ nIniChan: 시작 채널 번호

▶ nNumChan: 시작 채널 번호부터의 채널 개수

반환값

□ 이 함수는 채널별 출력상태를 32비트 값으로 반환합니다.

Value (예)	Meaning
0x0	모두 Off 상태
0xFFFFFFFF	모두 On 상태
0xAAAAAAAA	10101010101010101010101010101010 이므로 짝수채널만 On 상태

2-5. Communication Functions

Summary of Functions
❒ void **commBootup** (void) mkcBootup 에서 호출되는 함수로 통신초기화 함수입니다.
❒ void **commOpenPort** (t_i32 nSockType, t_i32 nChannel, TCommProtocol CommProtocol, TCommOpenInfo* pCOI) 통신 포트를 생성하고 사용할 준비를 합니다.
❒ void **commClosePort** (t_i32 nSockType, t_i32 nChannel) 통신 포트 사용을 종료합니다.
❒ void **commGetPeerInfo** (t_i32 nSockType, t_i32 nChannel, t_uchar abyHostIp[], t_ui32 *pnPort) 이 함수는 nSockType==SOCK_ETHERNET일 때만 사용할 수 있는 함수이다. 이 함수는 UDP 모드와 TCP 모드에서 각각 다음과 같은 일을 수행한다. 1. UDP 모드: 가장 마지막에 본 소켓이 수신한 데이터를 송신한 노드의 IP주소를 반환합니다. 포트번호는 commOpenPort() 함수 실행시에 전달된 포트번호를 반환합니다. 2. TCP 모드: 사용자가 설정한 상대측(server 또는 client)의 IP주소 및 포트를 반환합니다.
❒ void **commSetTimeout** (t_i32 nSockType, t_i32 nChannel, t_ui32 dwTimeOut) ❒ t_ui32 **commGetTimeout** (t_i32 nSockType, t_i32 nChannel) 패킷 수신 시의 타임아웃을 설정하거나 얻어옵니다.
❒ void **commRxReset** (t_i32 nSockType, t_i32 nChannel) 수신 버퍼를 초기화 합니다.
❒ void **commTxReset** (t_i32 nSockType, t_i32 nChannel) 송신 버퍼를 초기화 합니다.
❒ t_bool **commIsDataReady** (t_i32 nSockType, t_i32 nChannel) 수신 데이터 패킷이 도착했는지 확인합니다.
❒ t_bool **commGetUnreadSize** (t_i32 nSockType, t_i32 nChannel) 수신 데이터 패킷중 읽지 않은 패킷의 크기를 반환합니다.
❒ t_bool **commPeekByte** (t_i32 nSockType, t_i32 nChannel, t_byte* pbyRetVal) pop 카운트는 증가하지 않고 Rx Que에 있는 Pop되지 않은 첫번째 데이터를 peek합니다.
❒ t_bool **commPeekByteEx** (t_i32 nSockType, t_i32 nChannel, t_byte* pbyRetVal, t_i32 nIndex) pop 카운트는 증가하지 않고 Rx Que에 있는 Pop되지 않은 첫번째 데이터를 peek합니다.
❒ t_bool **commReadByte** (t_i32 nSockType, t_i32 nChannel, void* pBuffer) Rx Que에 있는 Pop되지 않은 첫번째 데이터를 read한 뒤에 pop 카운트를 하나 증가시킵니다.
❒ t_bool **commWriteByte** (t_i32 nSockType, t_i32 nChannel, t_byte bData) Tx Que에 1바이트 크기의 데이터를 write한 뒤에 push 카운트를 하나 증가시킵니다.

❒ t_i32 **commPeekString** (t_i32 nSockType, t_i32 nChannel, t_char* pBuffer, t_ui32 nNumBytes)

pop 카운트는 증가하지 않고 Rx Que에 있는 Pop되지 않은 스트링 데이터를 peek합니다.

❒ t_i32 **commReadString** (t_i32 nSockType, t_i32 nChannel, t_char* pBuffer, t_ui32 nNumBytes)

Rx Que에 있는 Pop되지 않은 스트링 데이터를 read한 뒤에 pop 카운트를 바이트 크기만큼 증가시킵니다.

❒ t_i32 **commWriteString** (t_i32 nSockType, t_i32 nChannel, t_char* pBuffer, t_i32 nNumBytes)

Tx Que에 지정된 크기의 데이터를 write한 뒤에 push 카운트를 바이트 수만큼 증가시킵니다.

❒ t_i32 **commReadDword** (t_i32 nSockType, t_i32 nChannel, t_i32 *pBuffer, t_ui32 nNumDwords)

Rx Que에 있는 Pop되지 않은 연속된 4바이트 데이터를 DWORD 변수로 read한 뒤에 pop 카운트를 4바이트 증가시킵니다.

❒ t_i32 **commWriteDword** (t_i32 nSockType, t_i32 nChannel, t_i32 *pBuffer, t_ui32 nNumDwords)

Tx Que에 연속된 4바이트 크기의 DWORD 데이터를 write한 뒤에 push 카운트를 바이트 수만큼 증가시킵니다.

❒ t_i32 **commWriteDword_** (t_i32 nSockType, t_i32 nChannel, t_i32 dwData)

Tx Que에 연속된 4바이트 크기의 DWORD 데이터를 write한 뒤에 push 카운트를 바이트 수만큼 증가시킵니다.

❒ t_i32 **commCommit** (t_i32 nSockType, t_i32 nChannel)

실제 데이터를 전송하는 함수입니다. 마지막 수신된 UDP 패킷의 송신 IP 주소를 목적지로 합니다.

❒ t_i32 **commCommitTo** (t_i32 nSockType, t_i32 nChannel, t_uchar abyIpAddr[], t_ui32 nUdpPort)

이 함수는 UDP통신에서만 사용가능한 함수입니다. 목적지 IP 를 가지고 데이터를 전송합니다.

▣ commBootup

함수 원형

void **commBootup** (void)

함수 설명

common communication 라이브러리와 관련된 변수들을 초기화하는 함수. 이 함수는 시스템 부팅함수인 mkcBootup() 함수내부에서 호출되어져야 합니다.

예제

□ 기본적으로 mk_extern.c 소스파일에 있는 mkcBootup() 함수 내부에서 아래와 같이 commBootup() 함수를 수행하도록 되어 있습니다.

```
void mkcBootup(void)
{
    flsBootup(); // flash memory 부팅
    mcBootup(); // 모션 모듈 부팅, 모션 모듈 비사용시 주석처리
    dioBootup(); // Digital I/O 모듈 부팅, I/O 모듈 비사용시 주석처리
    aioBootup(); // Analog I/O 모듈 부팅
    g_uSerPortCnt = serBootup(); //시리얼 모듈 부팅

    g_auIpAddr[3] = GetLastIpAddrFromGPIOF();

    if(g_auIpAddr[3] != 0 && g_auIpAddr[3] != 255)
    {
        netBootup(g_auIpAddr); // 이더넷 초기화, ip 및 Mac Address 설정
        Act_LED_On(1);
        Err_LED_On(1);
    }
    else
    {
        // IpAddr[3] 이 0 또는 255 인 경우는 에러 처리
        Act_LED_On(0);
        Err_LED_On(1);
        while(1)
        {
        }
    }
    commBootup(); // 통신 초기화
}
```

▣ commOpenPort

함수 원형

void **commOpenPort** (t_i32 nSockType, t_i32 nChannel, TCommProtocol CommProtocol, TCommOpenInfo* pCOI)

함수 설명

통신 포트 하나를 open합니다. 이 함수는 Serial 과 Ethernet 통신의 공통으로 사용할 수 있도록 만들어진 함수입니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ **CommProtocol:** 프로토콜의 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (COMM_RS232)	RS232 통신 포트
1 (COMM_RS422)	RS422 통신 포트
2 (COMM_RS485)	RS485 통신 포트
3 (COMM_UDP)	UDP 통신 포트
4 (COMM_TCP)	TCP 통신 포트

▶ **pCOI:** 통신 포트를 open하기 위한 여러 가지 정보를 전달하는 공용체 파라미터입니다. TCommOpenInfo 데이터 형은 아래와 같이 여러 개의 구조체가 공존하는 공용체입니다.

```
typedef union{
    struct{
        t_ui32 uBaudRate;
        t_ui32 nDataBits;
        t_ui32 nStopBits;
        t_ui32 nParity;
        t_ui32 dwReserved[16];
    }RS232;
```

```
    struct{
        t_ui32 uBaudRate;
        t_ui32 nDataBits;
        t_ui32 nStopBits;
        t_ui32 nParity;
        t_ui32 dwReserved[16];
    }RS422;
    struct{
        t_ui32 uBaudRate;
        t_ui32 nDataBits;
        t_ui32 nStopBits;
        t_ui32 nParity;
        t_ui32 dwReserved[16];
    }RS485;
    struct{
        t_ui32 uLocalPort;
        t_ui32 uHostPort;
        t_ui32 uSockFlag;
        t_ui32 dwReserved[17];
    }UDP;
    struct{
        t_ui32 uLocalPort;
        t_ui32 uHostPort;
        t_ui32 uSockFlag;
        t_ui32 bIsServer; // 0->client,    1->server
        union{
            struct{
                t_ui32 b4: 8;
                t_ui32 b3: 8;
                t_ui32 b2: 8;
                t_ui32 b1: 8;
            }addr_1;
            t_ui32 addr_4;
        }HostIp;
        t_ui32 dwReserved[16];
    }TCP;
}TCommOpenInfo;
```

반환값

없음

참고

□ TCommOpenInfo 공용체의 각 멤버들의 의미는 다음과 같습니다(RS422과 RS485 구조체의 멤버들은 RS232 구조체의 멤버들과 동일하므로 설명을 생략합니다).

Value		Meaning
RS232 RS422 RS485	uBaudRate	시리얼통신의 baud rate를 설정합니다. baud rate의 아이디는 mk_net.h 헤더 파일에서 다음과 같이 선언되어 있으므로 이를 이용하시기 바랍니다. enum {BAUD_2400, BAUD_3600, BAUD_4800, BAUD_7200, BAUD_9600, BAUD_19200, BAUD_38400, BAUD_56000};
	nDataBits	시리얼통신의 데이터 비트수를 설정합니다. 이 값은 8, 7, 6, 5 중의 한 값이어야 합니다.
	nStopBits	시리얼통신의 Stop 비트 수를 설정합니다. 이 값은 1 또는 2 이어야 합니다.
	nParity	시리얼통신의 패리티 체크 옵션을 설정합니다. 0 => 패리티 체크 하지 않음 1 => 홀수 패리티 체크를 수행함 2 => 짝수 패리티 체크를 수행함
UDP	uLocalPort	장치에서 사용하는 UDP 포트를 설정합니다.
	uHostPort	이 항목은 현재 사용되지 않으므로 무시됩니다. Host 쪽의 UPD 포트 번호는 자동으로 검출하므로 사용자는 이 값을 설정할 필요가 없습니다.
	uSockFlag	소켓의 옵션플래그입니다. 일반적으로는 0으로 설정합니다. 만일 브로드캐스트(Broadcast) 송수신을 지원하려면 이 값에 SOCK_BROADCAST 또는 0x80을 지정하십시오.
TCP	bIsServer	0 – client mode 1 – server mode
	uLocalPort	UDP에서와 동일합니다.
	uHostPort	UDP에서와 동일합니다.
	uSockFlag	UDP에서와 동일합니다.
	HostIp	Peer(상대)측 IP주소를 설정합니다. 설정하는 방법은 다음과 같이 두 가지 방법이 있습니다(192.168.0.1을 설정한다고 가정). TCommOpenInfo COI; COI.TCP.HostIp. addr_4 = (192<<24) | (168<<16) | (0<<8) | 3; 또는 COI.TCP.HostIp.addr_1.b1 = 192; COI.TCP.HostIp.addr_1.b2 = 168; COI.TCP.HostIp.addr_1.b3 = 0; COI.TCP.HostIp.addr_1.b4 = 3;

□ 브로트캐스트(Broadcast)를 지원하기 위해서는 모든 Open된 이더넷 채널에 SOCK_BROADCAST 옵션을 지정해야 합니다.

예를 들어서 이더넷 채널 중에서 CH0와 CH1을 Open하며, CH0는 TCP모드로서

Broadcast가 필요없고, CH1은 UDP모드로서 Broadcast를 지원하도록 하고자할 때에 CH1의 UDP.uSockFlag=SOCK_BROADCAST를 설정하는 것은 물론이고 CH0에도 TCP.uSockFlag=SOCK_BROADCAST를 설정해주어야 합니다.

예제

```
#define UDP_PORT0          35040
#define UDP_PORT1          3000

void OpenSockets(void)
{
        TCommOpenInfo COI;
        // open UDP port //
        COI.UDP.uHostPort = UDP_PORT1 + g_auIpAddr[3]; // PORT 는
3000 번(BASE) + IP ADDR 4 번째 자리값 으로 한다.
        COI.UDP.uLocalPort = UDP_PORT1 + g_auIpAddr[3]; // PORT 는
3000 번(BASE) + IP ADDR 4 번째 자리값 으로 한다.
        COI.UDP.uSockFlag = 0;
        commOpenPort(SOCK_ETHERNET, 0, COMM_UDP, &COI);

        // 유저 프로그램 동작 시 펌웨어 업데이트 전용 소켓으로 1 번 사용.(필수!!)
        COI.UDP.uHostPort = UDP_PORT0; // 전용 포트번호:35040 고정 및 소켓 번호
1 번 고정 사용(아래).
        COI.UDP.uLocalPort = UDP_PORT0;
        COI.UDP.uSockFlag = 0;
        commOpenPort(SOCK_ETHERNET, 1, COMM_UDP, &COI); // 펌웨어 업데이트
전용 소켓 및 포트(35040) 필수 사용.

        // open TCP port //
        COI.TCP.bIsServer = true;
        COI.TCP.uHostPort = TCP_PORT + g_auIpAddr[3]; // PORT 는 3000 번(BASE)
+ IP ADDR 4 번째 자리값 으로 한다. UDP 포트와 중복되도 상관 없음.
        COI.TCP.uLocalPort = TCP_PORT + g_auIpAddr[3]; // PORT 는 3000 번(BASE)
+ IP ADDR 4 번째 자리값 으로 한다.
        COI.TCP.uSockFlag = 0;      // W5100 에서는 socket flag 를 0x80 으로 하면
Broadcast 가 안됨. 그냥 0x00 으로 설정해야 함.
        commOpenPort(SOCK_ETHERNET, 2, COMM_TCP, &COI);

        // open RS232 port //
        COI.RS232.uBaudRate = BAUD_9600;
        COI.RS232.nDataBits = 8;
        COI.RS232.nStopBits = 1;
        COI.RS232.nParity = 0; //0:none, 1:odd, 2:even
        commOpenPort(SOCK_SERIAL, 0, COMM_RS232, &COI);
  #ifndef _DBG_WO_SC04
        if(g_uSerPortCnt > 1) // 통신 확장 모듈 ceSC04A 가 붙었을 때
        {
                // Ch.01 (시리얼채널은 zero base)
                // baud rate: 19200, data bit:8bit, Stop bit:1bit 및 패리티:Even 으로
설정해야 함.
                COI.RS232.uBaudRate = BAUD_19200;
                COI.RS232.nDataBits = 8;
```

```
                    COI.RS232.nStopBits = 1;
                    COI.RS232.nParity = 2; //0:none, 1:odd, 2:even
                    commOpenPort(SOCK_SERIAL, 1, COMM_RS485, &COI);   // RS-485

                    // Ch.02 모듈 #2
                    // baud rate: 9600, data bit:8bit, Stop bit:1bit 및 패리티:none 으로
설정해야 함.
                    COI.RS232.uBaudRate = BAUD_9600;
                    COI.RS232.nDataBits = 8;
                    COI.RS232.nStopBits = 1;
                    COI.RS232.nParity = 0; //0:none, 1:odd, 2:even
                    commOpenPort(SOCK_SERIAL, 2, COMM_RS485, &COI);   // RS-485

                    // Ch.03 RFID 모듈
                    // baud rate: 9600, stop bit:8bit 및 패리티:none 으로 설정해야 함.
                    COI.RS232.uBaudRate = BAUD_9600;
                    COI.RS232.nDataBits = 8;
                    COI.RS232.nStopBits = 1;
                    COI.RS232.nParity = 0; //0:none, 1:odd, 2:even
                    commOpenPort(SOCK_SERIAL, 3, COMM_RS232, &COI);   // RF-ID
전용

                    // Ch.04 GOT 설정에 따름.
                    COI.RS232.uBaudRate = BAUD_19200;
                    COI.RS232.nDataBits = 8;
                    COI.RS232.nStopBits = 1;
                    COI.RS232.nParity = 1; //0:none, 1:odd, 2:even
                    commOpenPort(SOCK_SERIAL, 4, COMM_RS422, &COI);   // RS422,
미쯔비시 서보 통신용.

        }
  #endif
}
```

▣ commClosePort

함수 원형

void **commClosePort** (t_i32 nSockType, t_i32 nChannel)

함수 설명

통신포트의 연결을 끊습니다. 이 함수는 시리얼통신이나 TCP/IP 통신에서 연결을 끊을 때 사용합니다. UDP 통신 모드에서는 사용할 필요가 없습니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

예제

```
#define TCP_PORT          3000
#define cnTCP_SOCKET      2

t_bool cnTcpOpenServerSocket(void)
{
        TCommOpenInfo COI;
        g_nCmndSeqNo = 0;

        COI.TCP.bIsServer = true;
        COI.TCP.uHostPort = TCP_PORT + g_auIpAddr[3]; // PORT 는 3000 번(BASE)
+ IP ADDR 4 번째 자리값 으로 한다. UDP 포트와 중복되도 상관 없음.
        COI.TCP.uLocalPort = COI.TCP.uHostPort;

#if     (_PLATFORM==PLF_miCUBE || _PLATFORM==PLF_miCUBE_E)
        COI.TCP.uSockFlag = SOCKOPT_BROADCAST; // UDP socket 이 broadcast
메세지를 받으려면 TCP 소켓도 broadcast 옵션이 활성화되어야 합니다.
#elif (_PLATFORM==PLF_EtherIP_1)
        COI.TCP.uSockFlag = 0x00;  // W5100 에서는 socket flag 를 0x80 으로 하면
Broadcast 가 안됨. 그냥 0x00 으로 설정해야 함.
#else
#endif

        commClosePort(SOCK_ETHERNET, cnTCP_SOCKET);
```

```
        commOpenPort(SOCK_ETHERNET, cnTCP_SOCKET, COMM_TCP, &COI);
        return true;
}
```

▣ commGetPeerInfo

함수 원형

void **commGetPeerInfo** (t_i32 nSockType, t_i32 nChannel, t_uchar abyHostIp[], t_ui32 *pnPort)

함수 설명

연결된 상대측의 IP 주소 및 포트 번호를 반환합니다. 이 함수는 Ethernet 통신 소켓에서만 사용할 수 있는 함수입니다.

TCP 통신 모드에서는 사용자가 지정한 상대측 IP주소 및 Port번호를 반환합니다.

UDP 통신 모드에서는 가장 마지막에 지정한 소켓이 수신한 데이터를 송신한 노드의 IP주소를 반환합니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ **abyHostIp:** 상대측 IP 주소를 반환받을 배열 버퍼

▶ **pnPort:** 상대측 포트 번호를 반환받을 변수의 포인터

예제

```
if(commReadDword(st, nSockChan, &dwTemp, 1) < 1) break; // Read time-stamp //
cnSetLocalTimestamp(dwTemp);
if(commReadDword(st, nSockChan, &dwTemp, 1) < 1) break; // Read node-id //
if(g_NodeRegInf.F1.nState != cnNRS_OK){
//---------- Init TCP/IP server socket ----------//
commGetPeerInfo(st, nSockChan, auCnHostIp, NULL); // Save Host IP address to global
variable
// ..(중략)
                                    }
```

▣ commSetTimeout / commGetTimeout

함수 원형

void **commSetTimeout** (t_i32 nSockType, t_i32 nChannel, t_ui32 dwTimeOut)

t_ui32 **commGetTimeout** (t_i32 nSockType, t_i32 nChannel)

함수 설명

패킷 수신 시의 타임아웃을 설정하거나 얻어옵니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ **dwTimeOut**: 타임아웃

반환값

□ 이 함수는 에러코드를 반환합니다.

Value	Meaning
0 (cmERR_NONE)	에러 없음(수행 성공).
음수	에러 코드

예제

```
t_i32 nLen;

commSetTimeout(5000); // 타임 아웃을 5 초로 설정합니다.
nLen = 1;
if(commReadString(sBuffer, nLen) < nLen){
        return cnERR_TIMEOUT;
}
```

▣ commRxReset

함수 원형

void **commRxReset** (t_i32 nSockType, t_i32 nChannel)

함수 설명

수신 버퍼를 리셋합니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

예제

```
void commOpenPort(t_i32 nSockType, t_i32 nChannel, TCommProtocol CommProtocol,
TCommOpenInfo* pCOI)
{
        commRxReset(nSockType, nChannel);
        commTxReset(nSockType, nChannel);
        // ..(중략)
}
```

▣ commTxReset

함수 원형

void **commTxReset** (t_i32 nSockType, t_i32 nChannel)

함수 설명

송신 버퍼를 리셋합니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

예제

```
void commOpenPort(t_i32 nSockType, t_i32 nChannel, TCommProtocol CommProtocol,
TCommOpenInfo* pCOI)
{
        commRxReset(nSockType, nChannel);
        commTxReset(nSockType, nChannel);
        // ..(중략)
}
```

▣ commIsDataReady

함수 원형

t_bool **commIsDataReady** (t_i32 nSockType, t_i32 nChannel)

함수 설명

새로이 수신된 데이터가 있는지를 알려주는 함수입니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

반환값

새로이 수신된 데이터가 있는지를 반환합니다.

Value	Meaning
0 (FALSE)	수신 데이터 없음
1 (TRUE)	수신 데이터 있음

예제

```
nTemp = commPeekString(st, nSockChan, szTemp, 4);
commSetTimeout(st, nSockChan, dwTemp);
if(nTemp > 0 && (szTemp[0]=='e' || szTemp[0]=='E') && (szTemp[1]=='c' || szTemp[1]=='C')
        && (szTemp[2]=='h' || szTemp[2]=='H') && (szTemp[3]=='o' || szTemp[3]=='O'))
{
        do{
                commReadByte(st, nSockChan, &byData);
                commWriteByte(st, nSockChan, byData);
        }while(commIsDataReady(st, nSockChan));
        commCommit(st, nSockChan);
}
```

▣ commGetUnreadSize

함수 원형

t_bool **commGetUnreadSize** (t_i32 nSockType, t_i32 nChannel)

함수 설명

수신 데이터 패킷중 읽지 않은 패킷의 크기를 반환합니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

반환값

읽지 않은 데이터 바이트 수

Value	Meaning
0	읽지 않은 데이터 없음
양수	읽지 않은 데이터 바이트 수 (push 카운트 – pop 카운트)

예제

```
t_char cTemp;
while (commGetUnreadSize (SOCK_ETHERNET, nCh) > 0)
{
        if(commReadByte(SOCK_ETHERNET, nCh, &cTemp) < 1){
                return cnERR_TIMEOUT;

        // ..(처리)
}
```

▣ commPeekByte

함수 원형

t_bool **commPeekByte** (t_i32 nSockType, t_i32 nChannel, t_byte* pbyRetVal)

함수 설명

pop 카운트는 증가하지 않고 Rx Que에 있는 Pop되지 않은 첫번째 데이터를 peek합니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ **pbyRetVal**: 읽은 데이터

반환값

읽은 데이터 여부

Value	Meaning
0 (FALSE)	읽은 데이터 없음
1 (TRUE)	읽은 데이터 있음

예제

```
t_byte byTemp[1];
if ( !commPeckByte (SOCK_ETHERNET, 0, byTemp) )
{
        // ..(에러처리)
}
```

▣ commPeekByteEx

함수 원형

t_bool **commPeekByteEx** (t_i32 nSockType, t_i32 nChannel, t_byte* pbyRetVal, t_i32 nIndex)

함수 설명

pop 카운트는 증가하지 않고 Rx Que에 있는 Pop되지 않은 n번째 데이터를 peek합니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ **pbyRetVal**: 읽은 데이터

▶ **nIndex:** pop 되지 않은 데이터열 시작부터 n번째 값

반환값

읽은 데이터 여부

Value	Meaning
0 (FALSE)	읽은 데이터 없음
1 (TRUE)	읽은 데이터 있음

예제

```
//-------------------------------------------------------------------------------------------
// CommPeekByte(): Rx Que 에 있는 Pop 되지 않은 첫번째 데이터를 peek 한다.
// 이 함수가 CommReadByte()와 다른 것은 pop count 를 증가하지 않는다는 것이다.
//-------------------------------------------------------------------------------------------
t_i32 commPeekString(t_i32 nSockType, t_i32 nChannel, t_char* pBuffer, t_ui32
nNumBytes)
{
        t_i32 i, nNumRead = 0;
        for(i=0; i<nNumBytes; i++){
                if(!commPeekByteEx(nSockType, nChannel, pBuffer+i, i)){
                        return (nNumRead);
                }
                nNumRead++;
        }
}
```

▣ commReadByte

함수 원형

t_bool **commReadByte** (t_i32 nSockType, t_i32 nChannel, void* pBuffer)

함수 설명

Rx Que에 있는 Pop되지 않은 첫번째 데이터를 read한 뒤에 pop 카운트를 하나 증가시킵니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ **pBuffer**: 읽은 데이터

반환값

읽은 데이터 여부

Value	Meaning
0 (FALSE)	읽은 데이터 없음
1 (TRUE)	읽은 데이터 있음

예제

```
t_char cTemp;
while(nNumParams < cnMAX_NUM_CMD_PARAM){
        if(commReadByte(st, nSockChan, &cTemp) < 1){
                return cnERR_TIMEOUT;
        }
}
```

▣ commWriteByte

함수 원형

t_bool **commWriteByte** (t_i32 nSockType, t_i32 nChannel, t_byte bData)

함수 설명

Tx Que에 1바이트 크기의 데이터를 write한 뒤에 push 카운트를 하나 증가시킵니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ **bData:** 송신할 바이트 데이터

반환값

바이트 기록 성공 여부

Value	Meaning
0 (FALSE)	기록 실패
1 (TRUE)	기록 성공

예제

```
nNumModules = dioGetNumDevice();
commWriteDword_(st, nSockChan, nNumModules);
for(i=0; i<nNumModules; i++){
        ////// DIO Information ////////
        commWriteByte(st, nSockChan, cndGetModuleId(i)); // Module ID
        pDioDev = dioGetDeviceDesc(i);
        commWriteByte(st, nSockChan, cnIOMT_DIO); // Device type
        commWriteByte(st, nSockChan, pDioDev->nNumDio & 0xff); // DI channel count
        commWriteByte(st, nSockChan, (pDioDev->nNumDio>>8) & 0xff);
}
```

▣ commPeekString

함수 원형

t_i32 **commPeekString** (t_i32 nSockType, t_i32 nChannel, t_char* pBuffer, t_ui32 nNumBytes)

함수 설명

pop 카운트는 증가하지 않고 Rx Que에 있는 Pop되지 않은 스트링 데이터를 peek합니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ **pBuffer**: 읽은 데이터

▶ **nNumBytes:** 읽을 데이터 크기(바이트 수)

반환값

읽은 데이터의 크기 (바이트 수)

예제

```
t_i32 nTemp;
t_ui32 dwTemp, dwSignature;
t_char szTemp[32];

while(commIsDataReady(st, nSockChan)){
        //------- ECHO FUNCTION SUPPORT ---------------------------------//
        if(commReadDword(st, nSockChan, &dwSignature, 1) < 1)
                break;

        dwTemp = commGetTimeout(st, nSockChan);
        commSetTimeout(st, nSockChan, 0);
        nTemp = commPeekString(st, nSockChan, szTemp, 4);
}
```

▣ commReadString

함수 원형

t_i32 **commReadString** (t_i32 nSockType, t_i32 nChannel, t_char* pBuffer, t_ui32 nNumBytes)

함수 설명

Rx Que에 있는 Pop되지 않은 스트링 데이터를 read한 뒤에 pop 카운트를 바이트 크기만큼 증가시킵니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ pBuffer: 읽은 데이터

▶ **nNumBytes:** 읽을 데이터의 크기 (바이트 수)

반환값

읽은 데이터의 크기 (바이트 수)

예제

```
t_char acBuffer[20];
commReadString(st, nSockChan, acBuffer, 1);
if(acBuffer[0] == GC_STX || acBuffer[0] == GC_SOH){ // 올바른 패킷 인 경우
        // ..(처리)
}
```

▣ commWriteString

함수 원형

t_i32 **commWriteString** (t_i32 nSockType, t_i32 nChannel, t_char* pBuffer, t_i32 nNumBytes)

함수 설명

Tx Que에 지정된 크기의 데이터를 write한 뒤에 push 카운트를 바이트 수만큼 증가시킵니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ **pBuffer**: 기록할 데이터

▶ **nNumBytes:** 기록할 데이터 크기 (바이트 수)

반환값

기록한 데이터의 크기 (바이트 수)

예제

```
/* 아래 cnParseCtrlMsg() 함수와 cnMakeReplyFrame() 함수는 지원되지 않는 임의의
함수이므로 참고만 하십시오*/
nErrCode = cnParseCtrlMsg(st, nSockChan, &nNode, &nCmdSeq, &nCommand,
anCnParam, &nNumParams);
if(nErrCode < 0){
        cnMakeReplyFrame(sBuffer, nCmdSeq, nErrCode, anCnParam, 0);
        commWriteString(st, nSockChan, g_sBuffer, strlen(sBuffer));
        commCommit(st, nSockChan);
        return (nErrCode);
}
```

▣ commReadDword

함수 원형

t_i32 **commReadDword** (t_i32 nSockType, t_i32 nChannel, t_i32 *pBuffer, t_ui32 nNumDwords)

함수 설명

Rx Que에 있는 Pop되지 않은 연속된 4바이트 데이터를 DWORD 변수로 read한 뒤에 pop 카운트를 4바이트 증가시킵니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ **pBuffer**: 읽은 데이터

▶ **nNumDwords:** 읽을 데이터의 크기(더블워드 수)

반환값

읽은 데이터의 크기 (더블워드 수)

예제

```
t_i32 nTemp;
t_ui32 dwTemp, dwSignature;
t_char szTemp[32];

while(commIsDataReady(st, nSockChan)){
        //------- ECHO FUNCTION SUPPORT ---------------------------------//
        if(commReadDword(st, nSockChan, &dwSignature, 1) < 1)
                break;

        dwTemp = commGetTimeout(st, nSockChan);
        commSetTimeout(st, nSockChan, 0);
        nTemp = commPeekString(st, nSockChan, szTemp, 4);
}
```

▣ commWriteDword

함수 원형

t_i32 **commWriteDword** (t_i32 nSockType, t_i32 nChannel, t_i32 *pBuffer, t_ui32 nNumDwords)

함수 설명

Tx Que에 연속된 4바이트 크기의 DWORD 데이터 n개를 write한 뒤에 push 카운트를 전체 바이트 수만큼 증가시킵니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ **pBuffer**: 기록할 데이터

▶ **nNumDwords:** 기록할 데이터의 크기(더블워드 수)

반환값

기록한 데이터의 크기 (더블워드 수)

예제

```
t_i32 commWriteDword_(t_i32 nSockType, t_i32 nChannel, t_i32 dwData)
{
        return commWriteDword(nSockType, nChannel, &dwData, 1);
}
```

▣ commWriteDword_

함수 원형

t_i32 **commWriteDword_** (t_i32 nSockType, t_i32 nChannel, t_i32 dwData)

함수 설명

Tx Que에 연속된 4바이트 크기의 DWORD 데이터 1개를 write한 뒤에 push 카운트를 4 바이트 수만큼 증가시킵니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ **dwData:** 기록할 더블워드 데이터

반환값

기록한 데이터의 크기 (더블워드 수), 성공 시 항상 1.

예제

```
t_i32 cnSendRegisterNode(t_i32 st, t_i32 nSockChan, t_bool bIsBroadCast)
{
        t_ui32 dwData;
        t_uchar abyHostAddr[4]={255, 255, 255, 255};

        //--------   send registration packet ----------------------------------//
        commWriteDword_(st, nSockChan, 0x1234); // signature //
        commWriteDword_(st, nSockChan, cnGetLocalTimestamp()); // Time-stamp //
        commWriteDword_(st, nSockChan, cnGetNodeId()); // Node ID //
        if(bIsBroadCast) commCommitTo(st, nSockChan, abyHostAddr, cnUDP_PORT);
        else commCommit(st, nSockChan);
        // .. (중략)
        return cnERR_NONE;
}
```

▣ commCommit

함수 원형

t_i32 **commCommit** (t_i32 nSockType, t_i32 nChannel)

함수 설명

실제 데이터를 전송하는 함수입니다. 마지막 수신된 UDP 패킷의 송신 IP 주소를 목적지로 합니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

반환값

송신데이터의 크기

예제

```
nTemp = commPeekString(st, nSockChan, szTemp, 4);
commSetTimeout(st, nSockChan, dwTemp);
if(nTemp > 0 && (szTemp[0]=='e' || szTemp[0]=='E') && (szTemp[1]=='c' || szTemp[1]=='C')
        && (szTemp[2]=='h' || szTemp[2]=='H') && (szTemp[3]=='o' || szTemp[3]=='O'))
{
        do{
                commReadByte(st, nSockChan, &byData);
                commWriteByte(st, nSockChan, byData);
        }while(commIsDataReady(st, nSockChan));
        commCommit(st, nSockChan);
}
```

▣ commCommitTo

함수 원형

t_i32 **commCommitTo** (t_i32 nSockType, t_i32 nChannel, t_uchar abyIpAddr[], t_ui32 nUdpPort)

함수 설명

이 함수는 UDP통신에서만 사용가능한 함수입니다. 목적지 IP 를 가지고 데이터를 전송합니다.

매개 변수

▶ **nSockType:** 통신 종류를 설정합니다. 이 값은 다음과 같은 값 중의 하나이어야 합니다.

Value	Meaning
0 (SOCK_SERIAL)	Serial 통신 포트
1 (SOCK_ETHERNET)	Ethernet 통신 포트

▶ **nChannel:** 통신 포트 채널 번호를 지정합니다. 여기서 채널 번호는 Serial 통신 포트와 Ethernet 통신 포트가 따로 번호 매겨집니다. 예를 들어서 장치에서 제공하는 Serial 통신 포트가 5개이고, Ethernet 통신 포트가 하나라고 가정하면, Serial 통신 포트의 채널 번호는 0 ~ 4이고, Ethernet 통신 포트의 채널 번호는 0이 됩니다.

▶ **abyIpAddr:** 목적지 IP Address 입니다.

▶ **nUdpPort:** 목적지 UDP Port 번호 입니다.

반환값

송신데이터의 크기

예제

```
t_i32 cnSendRegisterNode(t_i32 st, t_i32 nSockChan, t_bool bIsBroadCast)
{
        t_ui32 dwData;
        t_uchar abyHostAddr[4]={255, 255, 255, 255};

        //--------   send registration packet ----------------------------------//
        commWriteDword_(st, nSockChan, 0x1234); // signature //
        commWriteDword_(st, nSockChan, cnGetLocalTimestamp()); // Time-stamp //
        commWriteDword_(st, nSockChan, cnGetNodeId()); // Node ID //
        if(bIsBroadCast) commCommitTo(st, nSockChan, abyHostAddr, cnUDP_PORT);
        else commCommit(st, nSockChan);
        // .. (중략)
        return cnERR_NONE;
}
```

2-6. Flash API Functions

Summary of Functions
❒ void **flsBootup** (void) mkcBootup 에서 호출되는 함수로 Flash 메모리 초기화 함수입니다.
❒ t_bool **flsCopySect** (t_ui32 nSrcSectIdx, t_ui32 nTargSectIdx, t_i32 nSize) 소스 섹터의 데이터를 대상 섹터로 지정된 크기 만큼 복사합니다
❒ t_ui32 **flsGetSectBaseAddr** (t_i32 nSectIdx) 섹터 번호를 주소값으로 변환해주는 함수입니다.
❒ t_bool **flsCheckReady** (void) Flash 장치가 Read/Write 가능한 상태인지를 체크합니다
❒ t_i32 **flsReadDword** (t_ui32 dwAddr) 해당 주소에서 더블워드 크기의 데이터를 읽어옵니다.
❒ void **flsReadDwordMulti** (t_ui32 dwAddr, t_ui32 *plBuffer, t_ui32 nSize) 해당 주소에서 시작해서 지정된 더블워드 크기만큼 더블워드 데이터배열값을 읽어 옵니다.
❒ void **flsWriteDword** (t_ui32 dwAddr, t_ui32 dwData) 해당 주소에 더블워드 크기의 데이터를 기록합니다.
❒ void **flsWriteDwordMulti** (t_ui32 dwAddr, t_ui32 *plBuffer, t_ui32 nSize) 해당 주소에서 시작해서 지정한 더블워드 크기만큼 더블워드 데이터배열값을 기록합니다.
❒ t_bool **flsEraseSector** (t_ui16 nSectIdx) 해당 섹터를 삭제합니다.
❒ t_ui32 **flxGetAddrOfsect** (t_ui32 nSectIdx) 지정된 flash의 섹터의 시작주소를 반환한다. 여기서 주소는 flxRead_ / flxWrite_ 함수에 파라미터로 전달하는 주소를 말하며 flsGetSectBaseAddr() 함수와 달리 1-byte 단위를 가지는 주소체계로 반환한다.
❒ t_bool **flxCheckReady** (void) Flash 장치가 Read/Write 가능한 상태인지를 체크합니다
❒ t_byte **flxReadByte** (t_ui32 ulByteAddr) 해당 주소에서 바이트 크기의 데이터를 읽어옵니다.
❒ void **flxReadByteMulti** (t_ui32 ulByteAddr, t_byte *pbyBuffer, t_ui32 nNumData) 해당 주소에서 시작해서 지정된 바이트 크기만큼 바이트 데이터배열값을 읽어 옵니다.
❒ t_ui16 **flxReadWord**(t_ui32 ulByteAddr) 해당 주소에서 워드 크기의 데이터를 읽어옵니다.
❒ void **flxReadWordMulti**(t_ui32 ulByteAddr, t_ui16 *pwBuffer, t_ui32 nNumData) 해당 주소에서 시작해서 지정된 워드 크기만큼 워드 데이터배열값을 읽어 옵니다.
❒ t_ui32 **flxReadDword**(t_ui32 ulByteAddr)

해당 주소에서 더블워드 크기의 데이터를 읽어옵니다.
❒ void **flxReadDwordMulti**(t_ui32 ulByteAddr, t_ui32 *plBuffer, t_ui32 nNumData) 해당 주소에서 시작해서 지정된 더블워드 크기만큼 더블워드 데이터배열값을 읽어 옵니다.
❒ t_f32 **flxReadFlt**(t_ui32 ulByteAddr) Flash 에서 하나의 32비트 float형 데이터를 읽어서 반환한다.
❒ void **flxWriteByte**(t_ui32 ulByteAddr, t_byte byData) 해당 주소에 바이트 크기의 데이터를 기록합니다.
❒ void **flxWriteByteMulti**(t_ui32 ulByteAddr, t_byte *pbyBuffer, t_ui32 nNumData) 해당 주소에서 시작해서 지정한 바이트 크기만큼 바이트 데이터배열값을 기록합니다.
❒ void **flxWriteWord**(t_ui32 ulByteAddr, t_ui16 wData) 해당 주소에 워드 크기의 데이터를 기록합니다.
❒ void **flxWriteWordMulti**(t_ui32 ulByteAddr, t_ui16 *pwBuffer, t_ui32 nNumData) 해당 주소에서 시작해서 지정한 워드 크기만큼 워드 데이터배열값을 기록합니다.
❒ void **flxWriteDword**(t_ui32 ulByteAddr, t_ui32 dwData) 해당 주소에 더블워드 크기의 데이터를 기록합니다.
❒ void **flxWriteDwordMulti**(t_ui32 ulByteAddr, t_ui32 *plBuffer, t_ui32 nNumData) 해당 주소에서 시작해서 지정한 더블워드 크기만큼 더블워드 데이터배열값을 기록합니다.
❒ void **flxWriteFlt**(t_ui32 ulByteAddr, t_f32 fTmsValue) Flash 에 하나의 32비트 float형 데이터를 기록한다.
❒ t_bool **flxEraseSector**(t_ui16 nSectIdx) 해당 섹터를 삭제합니다.

▣ flsBootup

함수 원형

void **flsBootup** ()

함수 설명

Flash 메모리 초기화 함수입니다. 이 함수는 시스템 부팅함수인 mkcBootup() 함수내부에서 호출되어져야 합니다.

매개 변수

▶ 없음

예제

□ 기본적으로 mk_extern.c 소스파일에 있는 mkcBootup() 함수 내부에서 아래와 같이 flsBootup() 함수를 수행하도록 되어 있습니다.

```
void mkcBootup(void)
{
    flsBootup(); // 전역 변수로 플래쉬 영역의 주소를 할당
    mcBootup(); // 모션장치 부팅
    dioBootup(); // 디지털 I/O 장치 부팅
    aioBootup(); // 아날로그 I/O 장치 부팅
    commBootup(); // Common communication 초기화
}
```

▣ flsCopySect

함수 원형

t_bool **flsCopySect** (t_ui32 nSrcSectIdx, t_ui32 nTargSectIdx, t_i32 nSize)

함수 설명

소스 섹터의 데이터를 대상 섹터로 지정된 크기 만큼 복사합니다

매개 변수

- ▶ **nSrcSectIdx**: 소스 섹터 인덱스
- ▶ **nTargSectIdx**: 대상 섹터 인덱스
- ▶ **nSize:** 백업할 데이터의 크기(32비트 단위)

  섹터별 Dword 크기는 '1.3 메모리 활용편' 을 참고하십시오.

반환값

□ 함수 실행 결과 성공 여부

Value	Meaning
0 (cmFALSE)	해당 섹터 삭제 실패 등의 이유로 복사 실패
1 (cmTRUE)	성공

예제

```
#define FLS_NODECFG 29
#define FLS_BACKUP 30
#define NODECFG_FLASH_SECT_SIZE 0x8000

t_i32 SectBackup()
{
        if(!flsCopySect(FLS_NODECFG, FLS_BACKUP,
NODECFG_FLASH_SECT_SIZE/*dword 크기단위*/))
                return cmERR_FLASH_COPYSECT_FAIL;
}
```

▣ flsGetSectBaseAddr

함수 원형

t_ui32 **flsGetSectBaseAddr** (t_i32 nSectIdx)

함수 설명

섹터 번호를 주소값으로 변환해주는 함수입니다.

매개 변수

▶ **nSectIdx**: 섹터 인덱스

반환값

□ 해당 섹터의 베이스 주소를 반환합니다.

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
uSectAddr = flsGetSectBaseAddr(FLS_NODECFG);
```

▣ flsCheckReady

함수 원형

t_bool **flsCheckReady** (void)

함수 설명

Flash 장치가 Read/Write 가능한 상태인지를 체크합니다

본 함수는 flsEraseSector 및 flsWrite~ 함수에서 알아서 처리해 주므로 유저가 이 함수를 직접 사용하시면 안됩니다

반환값

□ 플래쉬 장치가 사용 가능한지의 여부

Value	Meaning
0 (cmFALSE)	Read/Write/Erase의 작업 중 하나가 진행중이어서 Read/Write를 할 수 없습니다. Busy 상태임.
1 (cmTRUE)	현재 Read/Write 가능한 상태입니다.

예제

별도의 예제 코드가 없습니다.

▣ flsReadDword

함수 원형

t_i32 **flsReadDword** (t_ui32 dwAddr)

함수 설명

해당 주소에서 더블워드 크기의 데이터를 읽어옵니다..

매개 변수

▶ **dwAddr:** Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소

반환값

□ 해당 주소로부터 얻어온 4바이트 데이터 값

예제

```
flsEraseSector(0);
        flsWriteDword(0, 0xAAAAAAAA);
        nVal32 = flsReadDword(0);
        if(nVal32 != 0xAAAAAAAA)
                g_uErrCnt1++;

        flsWriteDword(2, 0x55555555); // 플래쉬 주소는 16 비트 체계이므로 앞에서
4 바이트(1Dword)를 기록했으므로 2 만큼 증가 시킵니다.
        nVal32 = flsReadDword(2);
        if(nVal32 != 0x55555555)
                g_uErrCnt2++;
```

▣ flsReadDwordMulti

함수 원형

void **flsReadDwordMulti** (t_ui32 dwAddr, t_ui32 *plBuffer, t_ui32 nSize)

함수 설명

해당 주소에서 시작해서 지정된 더블워드 크기만큼 더블워드 데이터배열값을 읽어 옵니다.

매개 변수

- ▶ **dwAddr:** Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소
- ▶ **plBuffer:** 얻어온 데이터 버퍼
- ▶ **nSize:** 버퍼 크기

반환값

없음

예제

```
typedef struct{
        t_i32 nVal[5];
        t_ui32 dwSign;
        t_ui32 dwComment[8];
        t_ui32 dwCommentCnt;
        t_f32 fData[2];
        t_ui32 dwReserved[10];
}TSampleData;


void rwTest()
{
        TSampleData tcfgW, tcfgW1[2], tcfgR, tcfgR1[2];
        t_ui32 dwAddr;
        t_i32 nDwordSize, nSize_t_ui32, nSizeBytes;
        t_i16 i,j;

        flsEraseSector(29);
        dwAddr = flsGetSectBaseAddr(29);

        nSize_t_ui32 = sizeof(t_ui32); // 4
        nSizeBytes = sizeof(TSampleData); // 108 bytes
        nDwordSize = nSizeBytes / nSize_t_ui32; // TSampleData 크기는 108 바이트,
따라서 nDwordSize = 108/4 = 27

        // 임의의 값을 대입
        for(i=0; i<8; i++){
```

```
                tcfgW.dwComment[i] = (t_ui32)i;
        }
        tcfgW.dwCommentCnt = 8;
        tcfgW.dwSign = 0xABCD1234;
        tcfgW.fData[0] = 4.5;
        tcfgW.fData[1] = 3.9;

        for(i=0; i<5; i++){
                tcfgW.nVal[i] = (t_i32)i * 2;
        }

        for(i=0; i<10; i++){
                tcfgW.dwReserved[i] = 0;
        }

        for(j=0;j<2;j++){
                // 임의의 값을 대입
                for(i=0; i<8; i++){
                        tcfgW1[j].dwComment[i] = (t_ui32)i;
                }
                tcfgW1[j].dwCommentCnt = 8;
                tcfgW1[j].dwSign = 0xABCD1234;
                tcfgW1[j].fData[0] = 4.5;
                tcfgW1[j].fData[1] = 3.9;

                for(i=0; i<5; i++){
                        tcfgW1[j].nVal[i] = (t_i32)i * 2;
                }
                for(i=0; i<10; i++){
                        tcfgW1[j].dwReserved[i] = 0;
                }
        }

        // tcfgW 구조체 내용(108 바이트 == 27 DWords) 을 플래쉬에 기록합니다.

        flsWriteDwordMulti(dwAddr, (t_ui32 *)(&tcfgW), nDwordSize);
        flsWriteDwordMulti(dwAddr+(nDwordSize*2), (t_ui32 *)(&tcfgW1),
nDwordSize*2);

        // 플래쉬의 내용을 tcfgR 구조체로 얻어옵니다.
        flsReadDwordMulti(dwAddr, (t_ui32 *)(&tcfgR), nDwordSize);
        flsReadDwordMulti(dwAddr+(nDwordSize*2), (t_ui32 *)(&tcfgR1), nDwordSize*2);
        // 플래쉬 주소는 16 비트 체계이므로 앞에서 4 바이트(1Dword)를
기록했으므로 1Dword 당 2 배만큼 주소를 증가 시킵니다.
}
```

▣ flsWriteDword

함수 원형

void **flsWriteDword** (t_ui32 dwAddr, t_ui32 dwData)

함수 설명

해당 주소에 더블워드 크기의 데이터를 기록합니다.

매개 변수

- ▶ **dwAddr:** Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소
- ▶ **dwData:** 기록할 더블 워드형 데이터

반환값

없음

예제

```
flsEraseSector(0);
flsWriteDword(0, 0xAAAAAAAA);
nVal32 = flsReadDword(0);
if(nVal32 != 0xAAAAAAAA)
        g_uErrCnt1++;

flsWriteDword(2, 0x55555555); // 플래쉬 주소는 16 비트 체계이므로 앞에서
4 바이트(1Dword)를 기록했으므로 2 만큼 증가 시킵니다.
nVal32 = flsReadDword(2);
if(nVal32 != 0x55555555)
        g_uErrCnt2++;
```

▣ flsWriteDwordMulti

함수 원형

void **flsWriteDwordMulti** (t_ui32 dwAddr, t_ui32 *plBuffer, t_ui32 nSize)

함수 설명

해당 주소에서 시작해서 지정한 더블워드 크기만큼 더블워드 데이터배열값을 기록합니다.

매개 변수

- ▶ **dwAddr:** Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소
- ▶ **plBuffer:** 기록할 데이터 버퍼
- ▶ **nSize:** 버퍼 크기

반환값

없음

예제

```
typedef struct{
        t_i32 nVal[5];
        t_ui32 dwSign;
        t_ui32 dwComment[8];
        t_ui32 dwCommentCnt;
        t_f32 fData[2];
        t_ui32 dwReserved[10];
}TSampleData;


void rwTest()
{
        TSampleData tcfgW, tcfgW1[2], tcfgR, tcfgR1[2];
        t_ui32 dwAddr;
        t_i32 nDwordSize, nSize_t_ui32, nSizeBytes;
        t_i16 i,j;

        flsEraseSector(29);
        dwAddr = flsGetSectBaseAddr(29);

        nSize_t_ui32 = sizeof(t_ui32); // 4
        nSizeBytes = sizeof(TSampleData); // 108 bytes
        nDwordSize = nSizeBytes / nSize_t_ui32; // TSampleData 크기는 108 바이트,
따라서 nDwordSize = 108/4 = 27

        // 임의의 값을 대입
        for(i=0; i<8; i++){
```

```
                tcfgW.dwComment[i] = (t_ui32)i;
        }
        tcfgW.dwCommentCnt = 8;
        tcfgW.dwSign = 0xABCD1234;
        tcfgW.fData[0] = 4.5;
        tcfgW.fData[1] = 3.9;

        for(i=0; i<5; i++){
                tcfgW.nVal[i] = (t_i32)i * 2;
        }

        for(i=0; i<10; i++){
                tcfgW.dwReserved[i] = 0;
        }

        for(j=0;j<2;j++){
                // 임의의 값을 대입
                for(i=0; i<8; i++){
                        tcfgW1[j].dwComment[i] = (t_ui32)i;
                }
                tcfgW1[j].dwCommentCnt = 8;
                tcfgW1[j].dwSign = 0xABCD1234;
                tcfgW1[j].fData[0] = 4.5;
                tcfgW1[j].fData[1] = 3.9;

                for(i=0; i<5; i++){
                        tcfgW1[j].nVal[i] = (t_i32)i * 2;
                }
                for(i=0; i<10; i++){
                        tcfgW1[j].dwReserved[i] = 0;
                }
        }

        i++;

        // tcfgW 구조체 내용(108 바이트 == 27 DWords) 을 플래쉬에 기록합니다.

        // 플래쉬 주소는 16 비트 체계이므로 앞에서 4 바이트(1Dword)를
기록했으므로 1Dword 당 2 배만큼 주소를 증가 시킵니다.
        flsWriteDwordMulti(dwAddr, (t_ui32 *)(&tcfgW), nDwordSize);
        flsWriteDwordMulti(dwAddr+(nDwordSize*2), (t_ui32 *)(&tcfgW1),
nDwordSize*2);

        // 플래쉬의 내용을 tcfgR 구조체로 얻어옵니다.
        flsReadDwordMulti(dwAddr, (t_ui32 *)(&tcfgR), nDwordSize);
        flsReadDwordMulti(dwAddr+(nDwordSize*2), (t_ui32 *)(&tcfgR1), nDwordSize*2);

        i++;
}
```

▣ flsEraseSector

함수 원형

t_bool **flsEraseSector** (t_ui16 nSectIdx)

함수 설명

해당 섹터를 삭제합니다.

매개 변수

▶ **nSectIdx**: 섹터 인덱스

반환값

□ 섹터 삭제 결과 성공 여부

Value	Meaning
0 (cmFALSE)	삭제 실패
1 (cmTRUE)	삭제 성공

예제

```
#define FLASH_I  34
if( !flsEraseSector (FLASH_I) )
{
    // ..에러 처리
}
```

▣ flxGetAddrOfsect

함수 원형

t_ui32 **flxGetAddrOfsect** (t_ui32 nSectIdx)

함수 설명

지정된 flash의 섹터의 시작주소를 반환한다. 여기서 주소는 flxRead_ / flxWrite_ 함수에 파라미터로 전달하는 주소를 말하며 flsGetSectBaseAddr() 함수와 달리 1-byte 단위를 가지는 주소체계로 반환한다.

매개 변수

▶ **nSectIdx**: 섹터 인덱스

반환값

□ 해당 섹터의 베이스 주소를 반환합니다.

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
```

▣ flxCheckReady

함수 원형

t_bool **flxCheckReady** (void)

함수 설명

Flash 장치가 Read/Write 가능한 상태인지를 체크합니다.

본 함수는 flxEraseSector 및 flxWrite~ 함수에서 알아서 처리해 주므로 유저가 이 함수를 직접 사용하시면 안됩니다.

매개 변수

▶ 없음

반환값

□ 플래쉬 장치가 사용 가능한지의 여부

Value	Meaning
0 (cmFALSE)	Read/Write/Erase의 작업 중 하나가 진행중이어서 Read/Write를 할 수 없습니다. Busy 상태임.
1 (cmTRUE)	현재 Read/Write 가능한 상태입니다.

예제

```
uStartT = mkcTimerTick();
while(!flsCheckReady()){
        if(mkcTimerTick() - uStartT > 2000){
                return;
        }
}
```

▣ flxReadByte

함수 원형

t_byte **flxReadByte** (t_ui32 ulByteAddr)

함수 설명

해당 주소에서 바이트 크기의 데이터를 읽어옵니다.

매개 변수

▶ **ulByteAddr**: Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소

반환값

□ 해당 주소로부터 얻어온 1바이트 데이터 값.

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
t_byte byData;
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
byData = flxReadByte (uSectAddr);
```

▣ flxReadByteMulti

함수 원형

void **flxReadByteMulti** (t_ui32 ulByteAddr, t_byte *pbyBuffer, t_ui32 nNumData)

함수 설명

해당 주소에서 시작해서 지정된 바이트 크기만큼 바이트 데이터배열값을 읽어 옵니다.

매개 변수

- ▶ **ulByteAddr**: 읽어올 플래쉬 선두 주소
- ▶ **pbyBuffer**: 읽은 결과를 저장할 변수 주소
- ▶ **nNumData**: 읽어올 바이트 수

반환값

□ 없음.

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
t_byte abyData[10];
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
flxReadByteMulti (uSectAddr, abyData, 10);
```

▣ flxReadWord

함수 원형

t_ui16 **flxReadWord**(t_ui32 ulByteAddr)

함수 설명

해당 주소에서 16bit 워드 크기의 데이터를 읽어옵니다.

매개 변수

▶ **ulByteAddr**: Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소

반환값

□ 해당 주소로부터 얻어온 2바이트 워드 데이터 값

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
t_ui16 wData;
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
wData = flxReadWord (uSectAddr);
```

▣ flxReadWordMulti

함수 원형

void **flxReadWordMulti**(t_ui32 ulByteAddr, t_ui16 *pwBuffer, t_ui32 nNumData)

함수 설명

해당 주소에서 시작해서 지정된 워드 크기만큼 워드 데이터배열값을 읽어 옵니다.

매개 변수

- ▶ **ulByteAddr**: 읽어올 플래쉬 선두 주소
- ▶ **pwBuffer**: 읽은 결과를 저장할 변수 주소
- ▶ **nNumData**: 읽어올 워드 수

반환값

□ 없음.

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
t_ui16 awData[10];
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
flxReadWordMulti (uSectAddr, awData, 10);
```

▣ flxReadDWord

함수 원형

t_ui32 **flxReadDWord**(t_ui32 ulByteAddr)

함수 설명

해당 주소에서 32bit 더블워드 크기의 데이터를 읽어옵니다.

매개 변수

▶ **ulByteAddr**: Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소

반환값

□ 해당 주소로부터 얻어온 4바이트 더블워드 데이터 값.

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
t_ui32 dwData;
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
dwData = flxReadDWord (uSectAddr);
```

▣ flxReadDWordMulti

함수 원형

void **flxReadDwordMulti**(t_ui32 ulByteAddr, t_ui32 *plBuffer, t_ui32 nNumData)

함수 설명

해당 주소에서 시작해서 지정된 더블워드 크기만큼 더블워드 데이터배열값을 읽어 옵니다.

매개 변수

- ▶ **ulByteAddr**: 읽어올 플래쉬 선두 주소
- ▶ **plBuffer**: 읽은 결과를 저장할 변수 주소
- ▶ **nNumData**: 읽어올 더블워드 수

반환값

□ 없음.

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
t_ui32 adwData[10];
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
flxReadDWordMulti (uSectAddr, adwData, 10);
```

▣ flxReadFlt

함수 원형

t_f32 **flxReadFlt**(t_ui32 ulByteAddr)

함수 설명

해당 Flash 주소에서 에서 하나의 32비트 float형 데이터를 읽어서 반환한다.

매개 변수

▶ **ulByteAddr**: Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소

반환값

□ 해당 주소로부터 얻어온 하나의 32비트 float 형 데이터 값.

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
t_f32 fData;
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
fData = flxReadFlt (uSectAddr);
```

▣ flxWriteByte

함수 원형

void **flxWriteByte**(t_ui32 ulByteAddr, t_byte byData)

함수 설명

해당 주소에 바이트 크기의 데이터를 기록합니다.

매개 변수

- **ulByteAddr:** Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소
- **byData:** 기록할 바이트형 데이터

반환값

없음

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
flxWriteByte(uSectAddr, 0xAA);
```

▣ flxWriteByteMulti

함수 원형

void **flxWriteByteMulti**(t_ui32 ulByteAddr, t_byte *pbyBuffer, t_ui32 nNumData)

함수 설명

해당 주소에서 시작해서 지정한 바이트 크기만큼 바이트 데이터배열값을 기록합니다.

매개 변수

- **ulByteAddr:** Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소
- **pbyBuffer:** 기록할 데이터 버퍼
- **nNumData:** 버퍼 크기

반환값

없음

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
t_byte abyBuffer[10];
// 적절한 abyBuffer 값 입력
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
flxWriteByteMulti (uSectAddr, abyBuffer, 10);
```

▣ flxWriteWord

함수 원형

void **flxWriteWord**(t_ui32 ulByteAddr, t_ui16 wData)

함수 설명

해당 주소에 워드 크기의 데이터를 기록합니다.

매개 변수

- **ulByteAddr:** Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소
- **wData:** 기록할 워드형 데이터

반환값

없음

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
flxWriteWord(uSectAddr, 0xAA55);
```

▣ flxWriteWordMulti

함수 원형

void **flxWriteWordMulti**(t_ui32 ulByteAddr, t_ui16 *pwBuffer, t_ui32 nNumData)

함수 설명

해당 주소에서 시작해서 지정한 워드 크기만큼 워드 데이터배열값을 기록합니다.

매개 변수

- **ulByteAddr:** Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소
- **pwBuffer:** 기록할 데이터 버퍼
- **nNumData:** 버퍼 크기

반환값

없음

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
t_ui16 awBuffer[10];
// 적절한 awBuffer 값 입력
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
flxWriteWordMulti (uSectAddr, awBuffer, 10);
```

▣ flxWriteDWord

함수 원형

void **flxWriteDword**(t_ui32 ulByteAddr, t_ui32 dwData)

함수 설명

해당 주소에 더블 워드 크기의 데이터를 기록합니다.

매개 변수

- **ulByteAddr:** Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소
- **dwData:** 기록할 더블 워드형 데이터

반환값

없음

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
flxWriteDWord(uSectAddr, 0xAABBCCDD);
```

▣ flxWriteDWordMulti

함수 원형

void **flxWriteDwordMulti**(t_ui32 ulByteAddr, t_ui32 *plBuffer, t_ui32 nNumData)

함수 설명

해당 주소에서 시작해서 지정한 더블워드 크기만큼 더블워드 데이터배열값을 기록합니다.

매개 변수

- **ulByteAddr:** Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소
- **plBuffer:** 기록할 데이터 버퍼
- **nNumData:** 버퍼 크기

반환값

없음

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
t_ui32 adwBuffer[10];
// 적절한 adwBuffer 값 입력
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
flxWriteDWordMulti (uSectAddr, adwBuffer, 10);
```

▣ flxWriteFlt

함수 원형

void **flxWriteFlt**(t_ui32 ulByteAddr, t_f32 fTmsValue)

함수 설명

해당 Flash 주소에 하나의 32비트 float 형 데이터를 기록합니다.

매개 변수

- **ulByteAddr:** Read 또는 Write 하려는 플래쉬 섹터 영역의 시작 주소
- **fTmsValue:** 기록할 32bit float형 데이터

반환값

없음

예제

```
#define FLS_NODECFG 29
t_i32 uSectAddr;
uSectAddr = flxGetAddrOfsect(FLS_NODECFG);
flxWriteFlt(uSectAddr, 123.45);
```

▣ flxEraseSector

함수 원형

t_bool **flxEraseSector** (t_ui16 nSectIdx)

함수 설명

해당 섹터를 삭제합니다.

매개 변수

▶ **nSectIdx**: 섹터 인덱스

반환값

□ 섹터 삭제 결과 성공 여부

Value	Meaning
0 (cmFALSE)	삭제 실패
1 (cmTRUE)	삭제 성공

예제

```
#define FLASH_I  34
if( !flxEraseSector (FLASH_I) )
{
    // ..에러 처리
}
```

2-7. Digital I/O Functions

Summary of Functions
❒ void **dioBootup** (void) mkcBootup 에서 호출되는 함수로 디지털 I/O 장치 초기화 함수입니다.
❒ t_i32 **dioGetNumDevice_Ex** (t_ui16 nModeType) Digital I/O 모듈 종류별 모듈 개수를 얻어옵니다.
❒ t_i32 **dioGetNumDevice** (void) Digital I/O 모듈의 개수를 반환합니다.
❒ t_ui16 **dioGetNumChannels** (void) Digital I/O 모듈의 전체 채널수를 반환합니다.
❒ void **dioSetIomode** (t_ui16 nChannel, t_bool nInOutMode) ❒ t_bool **dioGetIomode** (t_ui16 nChannel) 해당 한 채널을 입력 모드 또는 출력 모드로 설정 하거나 설정된 모드를 확인합니다.
❒ void **dioSetIomodeMulti** (t_ui16 nIniChan, t_ui16 nNumChan, t_ui32 dwIOMM) ❒ t_ui32 **dioGetIomodeMulti** (t_ui16 nIniChan, t_ui16 nNumChan) 해당 채널 범위의 채널들에 대해 입력 모드 또는 출력 모드로 설정 하거나 설정된 모드를 확인합니다.
❒ void **dioSetLogic** (t_ui16 nChannel, t_bool bLogic) ❒ t_bool **dioGetLogic** (t_ui16 nChannel) 해당 한 채널의 입력 또는 출력 로직(A접점/B접점) 을 설정하거나 설정된 로직을 얻어옵니다.
❒ void **dioSetLogicMulti** (t_ui16 nIniChan, t_ui16 nNumChan, t_ui32 dwLogicMask) ❒ t_ui32 **dioGetLogicMulti** (t_ui16 nIniChan, t_ui16 nNumChan) 해당 채널 범위의 채널들에 대해 입력 또는 출력 로직(A접점/B접점) 을 설정하거나 설정된 로직을 얻어옵니다.
❒ void **dioPutOne** (t_ui16 nChannel, t_bool bState) 해당 한 채널에 대해 출력 값을 내보냅니다. ❒ t_bool **dioGetOne** (t_ui16 nChannel) 해당 한 채널에 대해 입력 또는 출력 상태값을 읽어옵니다.
❒ void **dioPutMulti** (t_ui16 nIniChan, t_ui16 nNumChan, t_ui32 dwStates) 해당 채널 범위의 멀티 채널에 대해 출력 값을 각각 내보냅니다. ❒ t_ui32 **dioGetMulti** (t_ui16 nIniChan, t_ui16 nNumChan) 해당 채널 범위의 멀티 채널에 대해 입력 또는 출력 값을 각각 얻어옵니다.

▣ dioBootup

함수 원형

void **dioBootup** ()

함수 설명

디지털 I/O 장치 초기화 함수입니다. 이 함수는 시스템 부팅함수인 mkcBootup() 함수내부에서 호출되어져야 합니다.

매개 변수

▶ 없음

예제

□ 기본적으로 mk_extern.c 소스파일에 있는 mkcBootup() 함수 내부에서 아래와 같이 dioBootup() 함수를 수행하도록 되어 있습니다.

```
void mkcBootup(void)
{
    flsBootup(); // 전역 변수로 플래쉬 영역의 주소를 할당
    mcBootup(); // 모션장치 부팅
    dioBootup(); // 디지털 I/O 장치 부팅
    aioBootup(); // 아날로그 I/O 장치 부팅
    commBootup(); // Common communication 초기화
}
```

▣ dioGetNumDevice_Ex

함수 원형

t_i32 **dioGetNumDevice_Ex** (t_ui16 nModeType)

함수 설명

Digital I/O 모듈 종류별 모듈 개수를 얻어옵니다.

매개 변수

▶ **nModeType**: 모듈 종류

Value	Meaning
0	ceD16CM : 16채널 Input / Output 겸용 모듈
1	ceDI32N : 32채널 Input 전용 모듈
2	ceDO32N : 32채널 Output 전용 모듈

반환값

□ 해당 모듈 종류에 대한 Digital I/O 모듈의 개수

예제

```
#define MT_D16CM        0
#define MT_DI32N        1
#define MT_DO32N        2

t_ui16 nNumDioDev = dioGetNumDevice_Ex(MT_DI32N);
for(nIdx=0; nIdx<nNumDioDev; nIdx++)
{
        // ..처리
}
```

▣ dioGetNumDevice

함수 원형

t_i32 **dioGetNumDevice** (void)

함수 설명

Digital I/O 모듈의 개수를 반환합니다.

반환값

□ Digital I/O 모듈의 개수

예제

```
t_ui16 nNumDioDev = dioGetNumDevice();
for(nIdx=0; nIdx<nNumDioDev; nIdx++)
{
        // ..처리
}
```

▣ dioGetNumChannels

함수 원형

t_ui16 **dioGetNumChannels** (void)

함수 설명

Digital I/O 모듈의 전체 채널수를 반환합니다.

반환값

□ 전체 Digital I/O 모듈이 갖고 있는 전체 채널 수

예제

```
t_ui16 nNumDioCh = dioGetNumChannels();
for(nCh=0; nCh<nNumDioCh; nCh++)
{
        dioSetIomode(nCh, 1); // 모든 Digital I/O 채널의 모드를 1(출력) 로 설정
}
```

▣ dioSetIomode / dioGetIomode

함수 원형

void **dioSetIomode** (t_ui16 nChannel, t_bool nInOutMode)

t_bool **dioGetIomode** (t_ui16 nChannel)

함수 설명

해당 한 채널을 입력 모드 또는 출력 모드로 설정 하거나 설정된 모드를 확인합니다.

매개 변수

▶ **nChannel**: 채널 번호

▶ **nInOutMode:** 입출력 모드 (0 또는 1)

반환값

☐ **dioGetIomode()** 함수는 해당 채널이 입력 모드인지 출력 모드인지 여부를 알려줍니다.

Value	Meaning
0	입력 모드
1	출력 모드

예제

```
t_ui16 nCh = 0;
t_bool nMode = 0;

// 0 번 채널의 디지털 입출력 모드(Mode)를 1(출력모드) 로 설정
dioSetIomode(nCh, 1)

// 0 번 채널의 디지털 입출력 모드(Mode) 를 얻어옵니다.
nMode = dioGetIomode(nCh);

if( nMode != 1 )
{
        // 에러처리..
}
```

▣ dioSetIomodeMulti/ dioGetIomodeMulti

함수 원형

void **dioSetIomodeMulti** (t_ui16 nIniChan, t_ui16 nNumChan, t_ui32 dwIOMM)

t_ui32 **dioGetIomodeMulti** (t_ui16 nIniChan, t_ui16 nNumChan)

함수 설명

해당 채널 범위의 채널들에 대해 입력 모드 또는 출력 모드로 설정 하거나 설정된 모드를 확인합니다.

매개 변수

- ▶ **nIniChan:** 채널 범위의 시작 채널번호
- ▶ **nNumChan:** 채널 수 (채널 범위의 크기)
- ▶ **dwIOMM:** 채널별 입출력 모드가 설정된 32비트 마스크 값

반환값

- □ **dioGetIomodeMulti()** 함수는 32비트 값으로 설정된 각 채널별 입출력 모드 상태를 반환합니다.

예제

```
t_ui16 nStart=0, nNumCh=32;
t_ui32 nModes = 0;

/* 0 번채널부터 31 번 채널까지 (32 개 멀티채널) 입출력 모드를 0xAAAA
( 10101010101010101010101010101010) 값으로 Write 합니다.
각 비트별로 0 은 입력로직, 1 은 출력로직 을 의미합니다.*/
dioSetIomodeMulti(nStart, nNumCh, 0xAAAA)

// 0 ~ 31 번 채널에 대해 입출력 모드 값을 읽어 옵니다.
nModes = dioGetIomodeMulti(nStart, nNumCh);

if( ((nModes & 0xAAAA) & 0xFFFF) != 0xAAAA )
{
        // 에러처리..
}
```

▣ dioSetLogic / dioGetLogic

함수 원형

void **dioSetLogic** (t_ui16 nChannel, t_bool bLogic)

t_bool **dioGetLogic** (t_ui16 nChannel)

함수 설명

해당 한 채널의 입력 또는 출력 로직(A접점/B접점) 을 설정하거나 설정된 로직을 얻어옵니다.

매개 변수

▶ **nChannel**: 채널 번호

▶ **bLogic:** 로직 값 (0 또는 1), enum _TDioLogic {LOGIC_A, LOGIC_B}; 로 선언되어 있습니다.

반환값

□ **dioGetLogic()** 함수는 해당 채널의 로직 상태를 반환합니다.

Value	Meaning
0 (LOGIC_A)	A접점 (Normal Open) 방식 => 평상시 Open, 감지되면 Close 되는 스위치 방식
1 (LOGIC_B)	B접점 (Normal Close) => 평상시 Close, 감지되면 Open 되는 스위치 방식

예제

```
t_ui16 nCh = 0;
t_bool nLogic = 0;

// 0 번 채널의 논리(로직) 설정을 1(B 접점, Normal Close) 로 설정
dioSetLogic(nCh, LOGIC_B)

// 0 번 채널의 논리(로직) 설정을 얻어옵니다.
nLogic = dioGetLogic(nCh);

if( nLogic != LOGIC_B )
{
        // 에러처리..
}
```

▣ dioSetLogicMulti / dioGetLogicMulti

함수 원형

void **dioSetLogicMulti** (t_ui16 nIniChan, t_ui16 nNumChan, t_ui32 dwLogicMask)

t_ui32 **dioGetLogicMulti** (t_ui16 nIniChan, t_ui16 nNumChan)

함수 설명

해당 채널 범위의 채널들에 대해 입력 또는 출력 로직(A접점/B접점) 을 설정하거나 설정된 로직을 얻어옵니다.

매개 변수

- ▶ **nIniChan:** 채널 범위의 시작 채널번호
- ▶ **nNumChan:** 채널 수 (채널 범위의 크기)
- ▶ **dwLogicMask:** 채널별 로직 상태값이 설정된 32비트 마스크 값

반환값

☐ **dioGetLogicMulti()** 함수는 32비트 값으로 설정된 각 채널별 로직 상태를 반환합니다.

예제

```
t_ui16 nStart=0, nNumCh=32;
t_ui32 nLogicMask = 0;

/* 0 번채널부터 31 번 채널까지 (32 개 멀티채널) 논리(로직) 설정을 0xAAAA
( 10101010101010101010101010101010) 값으로 Write 합니다.
각 비트별로 0 은 LOGIC_A (A 접점), 1 은 LOGIC_B(B 접점) 을 의미합니다.*/
dioSetLogicMulti(nStart, nNumCh, 0xAAAA)

// 0 ~ 31 번 채널에 대해 논리(로직) 설정 값을 읽어 옵니다.
nLogicMask = dioGetLogicMulti(nStart, nNumCh);

if( ((nLogicMask & 0xAAAA) & 0xFFFF) != 0xAAAA )
{
        // 에러처리..
}
```

▣ dioPutOne / dioGetOne

함수 원형

void **dioPutOne** (t_ui16 nChannel, t_bool bState)

t_bool **dioGetOne** (t_ui16 nChannel)

함수 설명

dioPutOne() 함수는 대상 디지털 채널을 통해 디지털 출력을 발생시킵니다.

dioGetOne() 함수는 대상 디지털 채널의 용도(Mode) 에 따라 디지털 입력 또는 출력 상태를 반환합니다.

매개 변수

▶ **nChannel**: 채널 번호

▶ **bState:** 출력 상태 (0 또는 1)

Value	Meaning
0	OFF
1	ON

반환값

□ 해당 채널의 디지털 입력 또는 출력 상태값 (0 또는 1)

Value	Meaning
0	OFF
1	ON

예제

```
t_ui16 nCh = 0;
t_bool nState = 0;

// 0 번 채널의 디지털 출력 상태를 1(ON) 으로 설정
dioPutOne(nCh, 1)

// 0 번 채널의 디지털 출력 상태를 얻어옵니다.
nState = dioGetOne(nCh);

if( nState != 1 )
{
        // 에러처리..
}
```

▣ dioPutMulti / dioGetMulti

함수 원형

void **dioPutMulti** (t_ui16 nIniChan, t_ui16 nNumChan, t_ui32 dwStates)

t_ui32 **dioGetMulti** (t_ui16 nIniChan, t_ui16 nNumChan)

함수 설명

dioPutMulti() 함수는 대상 디지털 채널 범위의 다중 디지털 채널을 통해 디지털 출력을 발생시킵니다.

dioGetMulti() 함수는 다중 디지털 채널의 용도(Mode) 에 따라 다채널에 대한 입력 혹은 출력 상태를 반환합니다.

매개 변수

- ▶ **nIniChan:** 출력 시작 채널
- ▶ **nNumChan:** 출력 채널의 수 (범위의 크기)
- ▶ **dwStates:** 출력하고자 하는 각 채널의 상태 (비트별로 채널의 출력 상태를 나타냅니다)

반환값

dioGetMulti() 함수는 여러 채널에 대한 디지털 입력 또는 출력 상태를 반환합니다.

예제

```
t_ui16 nStart=0, nNumCh=32;
t_ui32 nStates = 0;

// 0 번채널부터 31 번 채널까지 (32 개 멀티채널) 디지털 출력을 0xAAAA
( 10101010101010101010101010101010) 값으로 Write 합니다.
dioPutMulti(nStart, nNumCh, 0xAAAA)

// 0 ~ 31 번 채널에 대해 디지털 출력 값을 읽어 옵니다.
nStates = dioGetMulti(nStart, nNumCh);

if( ((nStates & 0xAAAA) & 0xFFFF) != 0xAAAA )
{
        // 에러처리..
}
```

2-8. Analog I/O Functions

Summary of Functions
❒ void **aioBootup** (void) mkcBootup 에서 호출되는 함수로 아날로그 I/O 장치 초기화 함수입니다.
❒ t_i32 **aiGetNumDevice** (void) Analog Input (A/D) 모듈의 개수를 반환합니다.
❒ t_bool **aiIsValidChan** (t_ui16 nAiChan) 해당 채널이 유효한 채널 범위에 들어서 사용 가능한지 여부를 반환합니다.
❒ t_ui16 **aiGetNumChannels** (void) 동일 노드 내 Analog Input 모듈들의 전체 채널수를 반환합니다.
❒ t_i32 **aoGetNumDevice** (void) Analog Output (D/A) 모듈의 개수를 반환합니다.
❒ t_bool **aoIsValidChan** (t_ui16 nAoChan) 해당 채널이 유효한 채널 범위에 들어서 사용 가능한지 여부를 반환합니다.
❒ t_ui16 **aoGetNumChannels** (void) 동일 노드 내 Analog Output (D/A) 모듈들의 전체 채널수를 반환합니다.
❒ t_i32 **aiSetVoltRangeMode** (t_ui16 nChannel, t_i32 nMode) ❒ t_i32 **aiGetVoltRangeMode** (t_ui16 nChannel, t_i32* pnMode) 해당 채널에 대한 특정 전압 입력 범위에 대한 모드를 설정하거나 설정된 모드를 반환합니다. 모드는 아래와 같습니다. 0: Bipolar (-10 ~ 10V) 1: Bipolar (-5 ~ 5V) 2: Bipolar (-2.5 ~ 2.5V) 3: Unipolar (0 ~ 10V, 0 ~ 20mA) 4: Unipolar (0 ~ 5V) 5: Unipolar (1 ~ 5V) 6: Uniipolar (4 ~ 20mA) 초기값은 전체널에 대해 모드0번(-10V ~ +10V) 으로 설정되어 있습니다.
❒ t_i32 **aiGetRangeDigit** (t_ui16 nChannel, t_i32* Dmin, t_i32* Dmax) 해당 채널에 대한 A/D 값에 대한 Digit 범위를 반환합니다.
❒ t_i32 **aiGetDigit** (t_ui16 nChannel, t_i32* pDigit) 해당 채널에 대해 A/D 한 결과 Digit 값 (Raw Data)을 반환합니다.
❒ t_i32 **aiGetVolt** (t_ui16 nChannel, t_f32* pfVolt) 해당 채널에 대해 A/D 한 결과 전압값을 반환합니다.
❒ t_i32 **aiGetCurrent** (t_ui16 nChannel, t_f32* pfCurrent) 해당 채널에 대해 A/D 한 결과 전류값을 반환합니다.

❒ t_i32 **aoOutDigit** (t_ui16 nChannel, t_i32 OutDigit) 해당 채널을 통해 Digit 값을 출력합니다.
❒ t_i32 **aoOutVolt** (t_ui16 nChannel, t_f32 OutVolt) 해당 채널을 통해 전압값을 출력합니다.
❒ t_i32 **aoOutCurrent** (t_ui16 nChannel, t_f32 OutCurrent) 해당 채널을 통해 전류값을 출력합니다.

▣ aioBootup

함수 원형

void **aioBootup** ()

함수 설명

아날로그 I/O 장치 초기화 함수입니다. 이 함수는 시스템 부팅함수인 mkcBootup() 함수내부에서 호출되어져야 합니다.

매개 변수

▶ 없음

예제

□ 기본적으로 mk_extern.c 소스파일에 있는 mkcBootup() 함수 내부에서 아래와 같이 aioBootup() 함수를 수행하도록 되어 있습니다.

```
void mkcBootup(void)
{
    flsBootup(); // 전역 변수로 플래쉬 영역의 주소를 할당
    mcBootup(); // 모션장치 부팅
    dioBootup(); // 디지털 I/O 장치 부팅
    aioBootup(); // 아날로그 I/O 장치 부팅
    commBootup(); // Common communication 초기화
}
```

▣ aiGetNumDevice

함수 원형

t_i32 **aiGetNumDevice** (void)

함수 설명

Analog Input (A/D) 모듈의 개수를 반환합니다.

반환값

□ Analog Input (A/D) 모듈의 개수

예제

```
t_ui16 nNumAiDev = aiGetNumDevice();
for(nIdx=0; nIdx<nNumAiDev; nIdx++)
{
        // ..처리
}
```

▣ aiIsValidChan

함수 원형

t_bool **aiIsValidChan** (t_ui16 nAiChan)

함수 설명

해당 채널이 유효한 채널 범위에 들어서 사용 가능한지 여부를 반환합니다.

매개 변수

▶ **nAiChan**: 채널 번호

반환값

□ 채널 유효성 여부

Value	Meaning
0	무효한 채널
1	유효한 채널

예제

```
If ( aiIsValidChan (ch) ){
      aiGetDigit ( ch, &nDigit );
}else{
      // 에러 처리
}
```

▣ aiGetNumChannels

함수 원형

t_ui16 **aiGetNumChannels** (void)

함수 설명

동일 노드 내 모든 Analog Input 모듈들의 전체 채널수를 반환합니다.

반환값

□ 동일 노드 내 전체 Analog Input 채널 수

예제

```
t_ui16 nNumAiCh = aiGetNumChannels();
for(nCh=0; nCh<nNumAiCh; nCh++)
{
        // ..처리
}
```

▣ aoGetNumDevice

함수 원형

t_i32 **aoGetNumDevice** (void)

함수 설명

Analog Output (D/A) 모듈의 개수를 반환합니다.

반환값

□ Analog Output (D/A) 모듈의 개수

예제

```
t_ui16 nNumAoDev = aoGetNumDevice();
for(nIdx=0; nIdx<nNumAoDev; nIdx++)
{
        // ..처리
}
```

▣ aoIsValidChan

함수 원형

t_bool **aoIsValidChan** (t_ui16 nAoChan)

함수 설명

해당 채널이 유효한 채널 범위에 들어서 사용 가능한지 여부를 반환합니다.

매개 변수

▶ **nAoChan**: 채널 번호

반환값

□ 채널 유효성 여부

Value	Meaning
0	무효한 채널
1	유효한 채널

예제

```
If ( aoIsValidChan (ch) ){
      aoOutDigit ( ch, nDigit );
}else{
      // 에러 처리
}
```

▣ aoGetNumChannels

함수 원형

t_ui16 **aoGetNumChannels** (void)

함수 설명

동일 노드 내 모든 Analog Output 모듈들의 전체 채널수를 반환합니다.

반환값

□ 동일 노드 내 전체 Analog Output 채널 수

예제

```
t_ui16 nNumAoCh = aoGetNumChannels();
for(nCh=0; nCh<nNumAoCh; nCh++)
{
        // ..처리
}
```

▣ aiSetVoltRangeMode / aiGetVoltRangeMode

함수 원형

t_i32 **aiSetVoltRangeMode** (t_ui16 nChannel, t_i32 nMode)

t_i32 **aiGetVoltRangeMode** (t_ui16 nChannel, t_i32* nMode)

함수 설명

해당 채널에 대한 특정 전압 입력 범위에 대한 모드를 설정하거나 설정된 모드를 반환합니다.

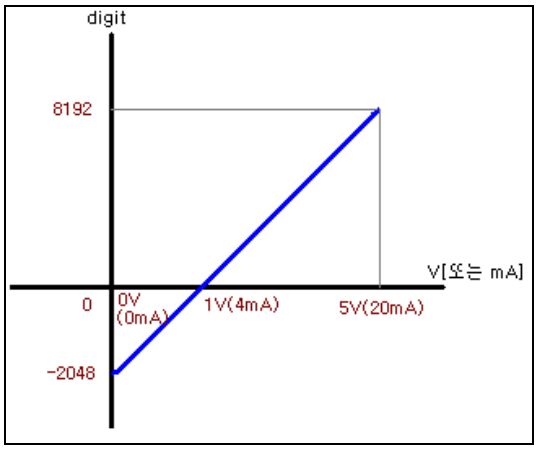
매개 변수

▶ **nChannel**: 채널 번호

▶ **nMode**: Range Mode [초기값은 전체널에 대해 모드0번(-10V ~ +10V) 으로 설정되어 있습니다]

Range Mode	Meaning
0	Bipolar (-10 ~ 10V)
1	Bipolar (-5 ~ 5V)
2	Bipolar (-2.5 ~ 2.5V)
3	Unipolar (0 ~ 10V, 0 ~ 20mA)
4	Unipolar (0 ~ 5V)
5	Unipolar (1 ~ 5V)
6	Uniipolar (4 ~ 20mA)

- 각 Rnage Mode 는 digit 범위 0 ~ 8192 로 선형 맵핑됩니다.
- 단, Range Mode 가 5 또는 6인 경우에는 0V~1V 사이 값 또는 0mA ~ 4mA 사이 값을 다음과 같이 음수 값으로 선형 보정해서 사용합니다.



반환값

□ 설정 성공 여부

Value	Meaning
0 (cmERR_NONE)	정상 설정됨
음수 값	cmERR_OUT_OF_RANGE (-1600), cmERR_INVALID_CHANNEL (-1105). 에러코드는 mk_common.h 참고

예제

```
t_i32 nMode;
if (aiSetVoltRangeMode (ch, 5) == cmERR_NONE) { // Unipolar (1 ~ 5V) Range 로 설정
{
    aiGetVoltRangeMode (ch, &nMode);
}
```

▣ aiGetRangeDigit

함수 원형

t_i32 **aiGetRangeDigit** (t_ui16 nChannel, t_i32* Dmin, t_i32* Dmax)

함수 설명

해당 채널에 대한 A/D 값에 대한 Digit 범위를 반환합니다.

매개 변수

- ▶ **nChannel**: 채널 번호
- ▶ **Dmin**: Digit 범위의 최소 값 [ceAI08A 기준 값 0]
- ▶ **Dmax**: Digit 범위의 최대 값 [ceAI08A 기준 값 8192]

반환값

□ 수행 결과

Value	Meaning
0 (cmERR_NONE)	정상 수행 됨
음수 값	cmERR_OUT_OF_RANGE (-1600), cmERR_INVALID_CHANNEL (-1105). 에러코드는 mk_common.h 참고

예제

```
t_i32 nMin, nMax;
if ( aiGetRangeDigit ( ch, &nMin, &nMax ) != cmERR_NONE ){
    // 에러 처리
}
```

▣ aiGetDigit

함수 원형

t_i32 **aiGetDigit** (t_ui16 nChannel, t_i32* pDigit)

함수 설명

해당 채널에 대해 A/D 한 결과 Digit 값 (Raw Data)을 반환합니다.

매개 변수

▶ **nChannel**: 채널 번호

▶ **pDigit**: A/D 값 (아날로그 입력 Raw Data)

반환값

□ 수행 결과

Value	Meaning
0 (cmERR_NONE)	정상 수행 됨
음수 값	cmERR_OUT_OF_RANGE (-1600), cmERR_INVALID_CHANNEL (-1105). 에러코드는 mk_common.h 참고

예제

```
t_i32 nDigit;
if ( aiGetDigit ( ch, &nDigit ) != cmERR_NONE ){
      // 에러 처리
}
```

▣ aiGetVolt

함수 원형

t_i32 **aiGetVolt** (t_ui16 nChannel, t_f32* pfVolt)

함수 설명

해당 채널에 대해 A/D 한 결과 전압값을 반환합니다.

매개 변수

▶ **nChannel**: 채널 번호

▶ **pfVolt**: A/D 값 (아날로그 입력 전압)

반환값

□ 수행 결과

Value	Meaning
0 (cmERR_NONE)	정상 수행 됨
음수 값	cmERR_OUT_OF_RANGE (-1600), cmERR_INVALID_CHANNEL (-1105). 에러코드는 mk_common.h 참고

예제

```
t_f32 fVolt;
if ( aiGetVolt ( ch, &fVolt ) != cmERR_NONE ){
    // 에러 처리
}
```

▣ aiGetCurrent

함수 원형

t_i32 **aiGetCurrent** (t_ui16 nChannel, t_f32* pfCurrent)

함수 설명

해당 채널에 대해 A/D 한 결과 전류값을 반환합니다.

매개 변수

▶ **nChannel**: 채널 번호

▶ **pfCurrent**: A/D 값 (아날로그 입력 전류)

반환값

□ 수행 결과

Value	Meaning
0 (cmERR_NONE)	정상 수행 됨
음수 값	cmERR_OUT_OF_RANGE (-1600), cmERR_INVALID_CHANNEL (-1105). 에러코드는 mk_common.h 참고

예제

```
t_f32 fCurrent;
if ( aiGetVolt ( ch, &fCurrent ) != cmERR_NONE ){
    // 에러 처리
}
```

▣ aoOutDigit

함수 원형

t_i32 **aoOutDigit** (t_ui16 nChannel, t_i32 OutDigit)

함수 설명

해당 채널을 통해 Digit 값을 출력합니다.

매개 변수

▶ **nChannel**: 채널 번호

▶ **OutDigit**: 아날로그 출력 Digit 값 (Raw Data). 출력 값의 Digit 범위는 16bits bipolar 로 -32768 ~ +32767 까지 입니다. 이 범위 밖의 값은 출력 될 수 없으며 자동으로 최소 또는 최대 값으로 조정 됩니다.

반환값

□ 수행 결과

Value	Meaning
0 (cmERR_NONE)	정상 수행 됨
음수 값	cmERR_INVALID_CHANNEL (-1105). 에러코드는 mk_common.h 참고

예제

```
if ( aoOutDigit ( ch, -32768 ) != cmERR_NONE ){ // Voltage 환산 -10V 를 출력합니다.
    // 에러 처리
}
```

▣ aoOutVolt

함수 원형

t_i32 **aoOutVolt** (t_ui16 nChannel, t_f32 OutVolt)

함수 설명

해당 채널을 통해 전압 값을 출력합니다.

매개 변수

▶ **nChannel**: 채널 번호

▶ **OutVolt**: 아날로그 출력 전압 값. 출력 값의 전압 범위는 -10V ~ +10V 까지 입니다. 이은 Digit 기준으로 -32768 ~ 32767 에 해당합니다. 이 범위 밖의 값은 출력 될 수 없으며 자동으로 최소 또는 최대 값으로 조정 됩니다.

반환값

□ 수행 결과

Value	Meaning
0 (cmERR_NONE)	정상 수행 됨
음수 값	cmERR_INVALID_CHANNEL (-1105). 에러코드는 mk_common.h 참고

예제

```
if ( aoOutVolt ( ch, 3.3 ) != cmERR_NONE ){ // 3.3V 를 출력합니다.
    // 에러 처리
}
```

▣ aoOutCurrent

함수 원형

t_i32 **aoOutCurrent** (t_ui16 nChannel, t_f32 OutCurrent)

함수 설명

해당 채널을 통해 전류 값을 출력합니다.

매개 변수

▶ **nChannel**: 채널 번호

▶ **OutCurrent**: 아날로그 출력 전류 값. 출력 값의 전류 범위는 4mA ~ 20mA 까지 입니다. 이 은 Digit 기준으로 0 ~ 32767 에 해당합니다. 이 범위 밖의 값은 출력 될 수 없으며 자동으로 최소 또는 최대 값으로 조정 됩니다.

반환값

□ 수행 결과

Value	Meaning
0 (cmERR_NONE)	정상 수행 됨
음수 값	cmERR_INVALID_CHANNEL (-1105). 에러코드는 mk_common.h 참고

예제

```
if ( aoOutCurrent ( ch, 10.5 ) != cmERR_NONE ){ // 10.5mA 를 출력합니다.
    // 에러 처리
}
```

2-9. Counter Functions

Summary of Functions
❒ t_i32 **cntGetNumDevice** (void) 카운터 모듈의 개수를 반환합니다.
❒ t_bool **cntIsValidChan** (t_ui16 nCntChan) 해당 채널이 유효한 채널 범위에 들어서 사용 가능한지 여부를 반환합니다.
❒ t_ui16 **cntGetNumChannels** (void) 카운터 모듈들의 전체 채널수를 반환합니다.
❒ void **cntSetCounterEdgeOne** (t_ui16 nChannel, t_i32 nEdgeMode) ❒ t_bool **cntGetCounterEdgeOne** (t_ui16 nChannel) 해당 채널의 Edge Mode (falling 또는 rising) 를 설정하거나 설정된 모드를 반환합니다. 디폴트 값은 전체널에 대해 falling edge (0) 입니다.
❒ void **cntSetCounterEdgeMulti** (t_ui16 nIniChan, t_ui16 nNumChan, t_ui32 dwEdgeModeMask) ❒ t_i32 **cntGetCounterEdgeMulti** (t_ui16 nIniChan, t_ui16 nNumChan) 다수 채널의 Edge Mode (falling 또는 rising) 를 설정하거나 설정된 모드를 반환합니다.
❒ t_i32 **cntCounterClearOne** (t_ui16 nChannel) 해당 채널의 카운터 값을 0으로 Reset 시킵니다.
❒ t_i32 **cntCounterClearMulti** (t_ui16 nIniChan, t_ui16 nNumChan) 다수 채널의 카운터 값을 0으로 Reset 시킵니다.
❒ t_i32 **cntCounterClearAll** () 전체 채널의 카운터 값을 0으로 Reset 시킵니다.
❒ t_i32 **cntGetCount** (t_ui16 nChannel) 해당 채널의 카운터 값을 반환합니다.
❒ void **cntSetCounterEnableOne** (t_ui16 nChannel, t_bool bEnable) ❒ t_bool **cntGetCounterEnableOne** (t_ui16 nChannel) 해당 채널의 카운터 기능 사용 여부를 설정하거나 설정된 사용 여부값을 반환합니다.
❒ void **cntSetCounterEnableMulti** (t_ui16 nIniChan, t_ui16 nNumChan, t_ui32 dwEnableMask) ❒ t_i32 **cntGetCounterEnableMulti** (t_ui16 nIniChan, t_ui16 nNumChan) 다수 채널의 카운터 기능 사용 여부를 설정하거나 설정된 사용 여부값을 반환합니다.
❒ t_i32 **cntGetOverflowFlagOne** (t_ui16 nChannel) 해당 채널의 카운터 값이 overflow 된 상태인지 반환합니다.
❒ t_i32 **cntGetOverflowFlagMulti** (t_ui16 nIniChan, t_ui16 nNumChan) 다수 채널의 카운터 값이 overflow 된 상태인지 반환합니다.
❒ t_i32 **cntOverflowFlagClearOne** (t_ui16 nChannel)

해당 채널의 카운터 값이 overflow 된 상태를 Clear 시킵니다.
❒ t_i32 **cntOverflowFlagClearMulti** (t_ui16 nInichan, t_ui16 nNumChan) 다수 채널의 카운터 값이 overflow 된 상태를 Clear 시킵니다.
❒ t_i32 **cntOverflowFlagClearAll** () 전체 채널의 카운터 값이 overflow 된 상태를 Clear 시킵니다.
❒ void **cntSetFilterFreq** (t_ui16 nChannel, t_i32 FreqSel) ❒ t_i32 **cntGetFilterFreq** (t_ui16 nChannel) 해당 채널의 필터 주파수 모드값을 설정하거나 설정된 모드값을 반환합니다. 사용 모드는 0 ~ 3 까지이며 디폴트는 0 입니다. 필터 주파수 모드에 대해서는 카운터 모듈의 하드웨어 매뉴얼 7.1절 Digital Filter 부분을 참고하십시오.

▣ cntGetNumDevice

함수 원형

t_i32 **cntGetNumDevice** (void)

함수 설명

카운터 모듈의 개수를 반환합니다.

반환값

□ 카운터 모듈의 개수

예제

```
t_ui16 nNumCntDev = cntGetNumDevice();
for(nIdx=0; nIdx<nNumCntDev; nIdx++)
{
        // ..처리
}
```

▣ cntIsValidChan

함수 원형

t_bool **cntIsValidChan** (t_ui16 nCntChan)

함수 설명

해당 채널이 유효한 채널 범위에 들어서 사용 가능한지 여부를 반환합니다.

매개 변수

▶ **nCntChan :** 채널 번호.

반환값

□ 채널 유효성 여부

Value	Meaning
0	무효한 채널
1	유효한 채널

예제

```
t_ui16 nNumCntCh;      // 카운터 값.

If ( cntIsValidChan (ch) ){
      nNumCntCh = cntGetCount ( ch );
}else{
      // 에러 처리
}
```

▣ cntGetNumChannels

함수 원형

t_ui16 **cntGetNumChannels** (void)

함수 설명

카운터 모듈들의 전체 채널수를 반환합니다.

반환값

□ 전체 카운터 모듈이 갖고 있는 전체 채널 수

예제

```
t_ui16 nNumCntCh = cntGetNumChannels ();

for(nCh=0; nCh<nNumCntCh; nCh++)
{
        // ..처리
}
```

▣ cntSetCounterEdgeOne / cntGetCounterEdgeOne

함수 원형

t_ui16 **cntSetCounterEdgeOne** (t_ui16 nChannel, t_i32 nEdgeMode)

t_ui16 **cntGetCounterEdgeOne** (t_ui16 nChannel)

함수 설명

해당 채널의 Edge Mode (falling 또는 rising) 를 설정하거나 설정된 모드를 반환합니다. 디폴트 값은 전체널에 대해 falling edge (0) 입니다.

매개 변수

▶ **nChannel :** 채널 번호

▶ **nEdgeMode :** 대상 카운터 채널의 Edge Mode.

Value	Meaning
0 [Default]	Falling Edge
1	Rising Edge

반환값

☐ **cntGetCounterEdgeOne()** 함수는 해당 채널의 Edge Mode 상태를 반환합니다.

Value	Meaning
0	Falling Edge.
1	Rising Edge

예제

```
t_ui16 nCh = 0;
t_bool nEdgeMode = 0;

// 0 번 채널의 Edge Mode 를 1(Rising Edge) 로 설정
cntSetCounterEdgeOne (nCh, 1)

// 0 번 채널의 Edge Mode 를 얻어옵니다.
nEdgeMode = cntGetCounterEdgeOne(nCh);

if( nEdgeMode != 1 )
{
        // 에러처리..
}
```

▣ cntSetCounterEdgeMulti / cntGetCounterEdgeMulti

함수 원형

Void **cntSetCounterEdgeMulti** (t_ui16 nIniChan, t_ui16 nNumChan, t_ui32 dwEdgeModeMask)

t_ui32 **cntGetCounterEdgeMulti** (t_ui16 nIniChan, t_ui16 nNumChan)

함수 설명

다수 채널의 Edge Mode (falling 또는 rising) 를 설정하거나 설정된 모드를 반환합니다. 디폴트 값은 전체널에 대해 falling edge (0) 입니다.

매개 변수

- ▶ **nIniChan :** 시작 채널 번호.
- ▶ **nNumChan :** 채널 수 (범위의 크기)
- ▶ **dwEdgeModeMask :** 채널별 Edge Mode 가 설정된 32비트 마스크 값.

반환값

☐ **cntGetCounterEdgeMulti()** 함수는 32비트 값으로 설정된 각 채널의 Edge Mode 상태를 반환합니다.

예제

```
t_ui16 nIniChan = 0, nNumChan = 32;
t_ui32 nEdgeModeMask = 0;

/* 0 번채널부터 31 번 채널까지 (32 개 멀티채널) EdgeMode 설정을 0xAAAA
( 10101010101010101010101010101010) 값으로 Write 합니다.
각 비트별로 0 은 Falling Edge, 1 은 Rising Edge 를 의미합니다.*/
cntSetCounterEdgeMulti(nIniChan, nNumChan, 0xAAAA)

// 0 ~ 31 번 채널에 대해 Edge Mode 설정 값을 읽어 옵니다.
nEdgeModeMask = cntGetCounterEdgeMulti(nIniChan, nNumChan);

if( ( (nEdgeModeMask & 0xAAAA) & 0xFFFF) != 0xAAAA )
{
        // 에러처리..
}
```

▣ cntCounterClearOne

함수 원형

t_i32 **cntCounterClearOne** (t_ui16 nChannel)

함수 설명

해당 채널의 카운터 값을 0으로 Reset 시킵니다.

매개 변수

▶ **nChannel :** 채널 번호.

반환값

Value	Meaning
0 (cmERR_NONE)	수행 성공
음수 값	수행 실패. 에러코드는 mk_common.h 참고

예제

```
t_ui16 nChannel = 0;

// 해당 채널 카운트 값을 '0'으로 리셋
if( cntCounterClearOne(nChannel) != cmERR_NONE ) {
    // 에러 처리
}
```

▣ cntCounterClearMulti

함수 원형

t_i32 **cntCounterClearMulti** (t_ui16 nIniChan, t_ui16 nNumChan)

함수 설명

다수 채널의 카운터 값을 0으로 Reset 시킵니다.

매개 변수

- ▶ **nIniChan :** 시작 채널 번호.
- ▶ **nNumChan :** 채널 수 (범위의 크기)

반환값

Value	Meaning
0 (cmERR_NONE)	수행 성공
음수 값	수행 실패. 에러코드는 mk_common.h 참고

예제

```
t_ui16 nIniChan = 0, nNumChan = 32;

// 다수 채널 카운트 값을 '0'으로 리셋
if( cntCounterClearMulti (nIniChan, nNumChan) != cmERR_NONE ) {
    // 에러 처리
}
```

▣ cntCounterClearAll

함수 원형

t_i32 **cntCounterClearAll** (void)

함수 설명

전체 채널의 카운터 값을 0으로 Reset 시킵니다.

반환값

Value	Meaning
0 (cmERR_NONE)	수행 성공
음수 값	수행 실패. 에러코드는 mk_common.h 참고

예제

```
// 전체 채널 카운트 값을 '0'으로 리셋
if( cntCounterClearAll () != cmERR_NONE ) {
    // 에러 처리
}
```

▣ cntGetCount

함수 원형

t_i32 **cntGetCount** (t_ui16 nChannel)

함수 설명

해당 채널의 카운터 값을 반환합니다.

매개 변수

▶ **nChannel :** 채널 번호.

반환값

□ 해당 채널의 카운터 값.

예제

```
t_ui16 nChannel = 0;

// 해당 채널 카운트 값을 확인합니다.
if( cntGetCount(nChannel) != cmERR_NONE ) {
    // 에러 처리
}
```

▣ cntSetCounterEnableOne / cntGetCounterEnableOne

함수 원형

void **cntSetCounterEnableOne** (t_ui16 nChannel, t_bool bEnable)

t_bool **cntGetCounterEnableOne** (t_ui16 nChannel)

함수 설명

해당 채널의 카운터 기능 사용 여부를 설정하거나 설정된 사용 여부값을 반환합니다.

매개 변수

▶ **nChannel :** 채널 번호.

▶ **nEnable :** 대상 카운터 채널의 카운트 기능 활성/비활성 상태를 설정합니다.

Value	Meaning
0 [Default]	Count Disable
1	Count Enable

반환값

☐ **cntGetCounterEnableOne()** 함수는 해당 채널의 카운트 기능 활성/비활성 상태를 반환합니다.

예제

```
t_ui16 nChannel = 0;
t_bool bCountEnable = 0;

// 0 번 채널의 카운터 기능 활성
cntSetCounterEnableOne (nChannel, 1)

// 0 번 채널의 카운터 기능 사용 여부 반환.
bCountEnable = cntGetCounterEnableOne(nChannel);

if( bCountEnable != 1 )
{
        // 에러처리..
}
```

▣ cntSetCounterEnableMulti / cntGetCounterEnableMulti

함수 원형

Void **cntSetCounterEnableMulti** (t_ui16 nIniChan, t_ui16 nNumChan, t_ui32 dwEnableMask)

t_ui32 **cntGetCounterEnableMulti** (t_ui16 nIniChan, t_ui16 nNumChan)

함수 설명

다수 채널의 카운터 기능 사용 여부를 설정하거나 설정된 사용 여부값을 반환합니다.

매개 변수

- ▶ **nIniChan :** 시작 채널 번호.
- ▶ **nNumChan :** 채널 수 (범위의 크기)
- ▶ **dwEnableMask :** 채널별 카운터 기능 활성/비활성 상태가 설정된 32비트 마스크 값.

반환값

- □ **cntGetCounterEnableMulti()** 함수는 32비트 값으로 설정된 각 채널의 카운터 기능 활성/비활성 상태를 반환합니다.

예제

```
t_ui16 nIniChan = 0, nNumChan = 32;
t_ui32 nEnableMask = 0;

/* 0 번채널부터 31 번 채널까지 (32 개 멀티채널) 카운터 활성/비활성 상태 설정을
0xAAAA ( 10101010101010101010101010101010) 값으로 Write 합니다.
각 비트별로 0 은 비활성, 1 은 활성 상태를 의미합니다.*/
cntSetCounterEdgeMulti(nIniChan, nNumChan, 0xAAAA)

// 0 ~ 31 번 채널에 대해 Edge Mode 설정 값을 읽어 옵니다.
nEdgeModeMask = cntGetCounterEdgeMulti(nIniChan, nNumChan);

if( ( (nEdgeModeMask & 0xAAAA) & 0xFFFF) != 0xAAAA )
{
        // 에러처리..
}
```

▣ cntGetOverflowFlagOne

함수 원형

t_i32 **cntGetOverflowFlagOne** (t_ui16 nChannel)

함수 설명

해당 채널의 카운터 값이 overflow 된 상태인지 반환합니다.

매개 변수

▶ **nChannel :** 채널 번호.

반환값

Value	Meaning
0 (cmFALSE)	Overflow 발생 안함.
1 (cmTRUE)	Overflow 발생.

예제

```
t_ui16 nChannel = 0;

if( cntGetOverflowFlagOne (nChannel) == cmTRUE ) {
      // Overflow 가 발생한 채널에 대한 처리
}
```

▣ cntGetOverflowFlagMulti

함수 원형

t_i32 **cntGetOverflowFlagMulti** (t_ui16 nIniChan, t_ui16 nNumChan)

함수 설명

다수 채널의 카운터 값이 overflow 된 상태인지 반환합니다.

매개 변수

- ▶ **nIniChan :** 시작 채널 번호.
- ▶ **nNumChan :** 채널 수 (범위의 크기)

반환값

□ 32비트 값으로 설정된 각 채널의 Overflow 상태를 반환합니다.

예제

```
t_ui16 nIniChan = 0, nNumChan = 32;
t_i32 dwOverflowStates;

dwOverflowStates = cntGetOverflowFlagMulti (nIniChan, nNumChan);

for(nIdx=0; nIdx< nNumChan; nIdx++)
{
    If( (dwOverflowStates >> nIdx)&0x1 == 1 )
    {
        // Overflow 가 발생한 채널에 대한 처리
    }
}
```

▣ cntOverflowFlagClearOne

함수 원형

t_i32 **cntOverflowFlagClearOne** (t_ui16 nChannel)

함수 설명

해당 채널의 카운터 값이 Overflow 된 상태를 Clear 시킵니다.

매개 변수

▶ **nChannel :** 채널 번호.

반환값

Value	Meaning
0 (cmERR_NONE)	수행 성공
음수 값	수행 실패. 에러코드는 mk_common.h 참고

예제

```
t_ui16 nChannel = 0;

if( cntGetOverflowFlagOne (nChannel) == cmTRUE ) {
    // Overflow 가 발생한 채널에 대한 처리
    // …
    // Overflow 상태 Clear.
    cntCounterClearOne(nChannel);
}
```

▣ cntOverflowFlagClearMulti

함수 원형

t_i32 **cntOverflowFlagClearMulti** (t_ui16 nIniChan, t_ui16 nNumChan)

함수 설명

다수 채널의 카운터 값이 overflow 된 상태를 Clear 시킵니다.

매개 변수

- ▶ **nIniChan :** 시작 채널 번호.
- ▶ **nNumChan :** 채널 수 (범위의 크기)

반환값

Value	Meaning
0 (cmERR_NONE)	수행 성공
음수 값	수행 실패. 에러코드는 mk_common.h 참고

예제

```
t_ui16 nIniChan = 0, nNumChan = 32;

// 다수 채널의 Overflow 상태 Clear
if( cntOverflowFlagClearMulti (nIniChan, nNumChan) != cmERR_NONE ) {
    // 에러 처리
}
```

▣ cntOverflowFlagClearAll

함수 원형

t_i32 **cntOverflowFlagClearAll** (void)

함수 설명

전체 채널의 카운터 값의 overflow 된 상태를 Clear 시킵니다.

반환값

Value	Meaning
0 (cmERR_NONE)	수행 성공
음수 값	수행 실패. 에러코드는 mk_common.h 참고

예제

```
// 전체 채널의 Overflow 상태 Clear
if( cntOverflowFlagClearAll () != cmERR_NONE ) {
    // 에러 처리
}
```

▣ cntSetFilterFreq / cntGetFilterFreq

함수 원형

void **cntSetFilterFreq** (t_ui16 nChannel, t_i32 FreqSel)

t_i32 **cntGetFilterFreq** (t_ui16 nChannel)

함수 설명

해당 채널의 필터 주파수 모드값을 설정하거나 설정된 모드값을 반환합니다.

사용 모드는 0 ~ 3 까지이며 디폴트는 0 입니다. 필터 주파수 모드에 대해서는 카운터 모듈의 하드웨어 매뉴얼 7.1절 Digital Filter 부분을 참고하십시오.

매개 변수

▶ **nChannel :** 채널 번호.

▶ **FreqSel :** 대상 카운터별 필터 주파수 모드를 설정합니다..

Value	필터 주파수	Cutoff 주파수 (50 Duty)	필터를 패스하기 위한 최소 시간
0 [Default]	10 MHz	500 kHz	1 usec
1	312 kHz	20 kHz	25 usec
2	39 kHz	4 kHz	125 usec
3	4.88 kHz	500 Hz	1 msec

반환값

☐ **cntGetFilterFreq()** 함수는 해당 채널의 필터 주파수 모드 설정 상태를 반환합니다.

예제

```
t_ui16 nChannel = 0;
t_i32 FreqSel = 0;

// 0 번 채널의 필터 주파수 모드를 0 으로 설정
cntSetFilterFreq (nChannel, 0)

// 0 번 채널의 필터 주파수 모드 설정 상태 확인.

if( cntGetFilterFreq(nChannel) != 0 )
{
        // 에러처리..
}
```

2-10. 에러 코드 편

에러 코드 정의 값	에러코드	에러 의미
cmERR_NONE	0	에러 없음
cmERR_MEM_ALLOC_FAIL	-290	Memory allocation fail
cmERR_GLOBAL_MEM_FAIL	-292	Global memory allocation fail
cmERR_ISR_CONNEC_FAIL	-310	ISR registration fail
cmERR_DIVIDE_BY_ZERO	-400	Cause divide by zero error
cmERR_WORNG_NUM_DATA	-500	Number of data is too small or too big
cmERR_VER_MISMATCH	-600	Version(of file or device) mismatch
cmERR_FLASH_ERASE_FAIL	-601	fail to erase a flash-memory sector
cmERR_FLASH_WRITE_FAIL	-602	fail to write a flash-memory sector
cmERR_FLASH_COPYSECT_FAIL	-603	fail to copy whole data from source sector to target sector
cmERR_FLASH_CANNOTMODIFY	-604	cannot modify the sector
cmERR_INVALID_DEVICE_ID	-1010	-
cmERR_INVALID_HANDLE	-1020	-
cmERR_UNSUPORTED_FUNC	-1030	-
cmERR_INVALID_PARAMETER	-1101	-
cmERR_INVALID_CHANNEL	-1105	-
cmERR_INVALID_INPUT_RANGE	-1111	Invalid range value (AI, AO)
cmERR_INVALID_FREQ_RANGE	-1121	Invalid input or output frequency
cmERR_FILE_CREATE_FAIL	-1501	File create fail
cmERR_FILE_OPEN_FAIL	-1511	File open fail
cmERR_FILE_READ_FAIL	-1522	File reading fail
cmERR_EVENT_CREATE_FAIL	-1550	Event handle creation fail
cmERR_INT_INSTANCE_FAIL	-1560	Interrupt event instance creation fail
cmERR_DITHREAD_CRE	-1570	D/I state change monitor thread creation fail
cmERR_BUFFER_SMALL	-1580	Buffer size is too small
cmERR_HIGH_TIMER_UNSUPP	-1590	The installed hardware does not support a high-resolution performance counter (cmmUtlDelayMicroSec() function fails)
cmERR_OUT_OF_RANGE	-1600	The range of some parameter is out of range
cmERR_ON_MOTION	-5001	-

cmERR_STOP_BY_SLP	-5002	Abnormally stopped by positive soft limit
cmERR_STOP_BY_SLN	-5003	Abnormally stopped by negative soft limit
cmERR_STOP_BY_CMP3	-5004	Abnormally stopped by comparator3
cmERR_STOP_BY_CMP4	-5005	Abnormally stopped by comparator4
cmERR_STOP_BY_CMP5	-5006	Abnormally stopped by comparator5
cmERR_STOP_BY_ELP	-5007	Abnormally stopped by (+) external limit
cmERR_STOP_BY_ELN	-5008	Abnormally stopped by (-) external limit
cmERR_STOP_BY_ALM	-5009	Abnormally stopped by alarm input signal
cmERR_STOP_BY_CSTP	-5010	Abnormally stopped by CSTP input signal
cmERR_STOP_BY_CEMG	-5011	Abnormally stopped by CEMG input signal
cmERR_STOP_BY_SD	-5012	Abnormally stopped by SD input signal
cmERR_STOP_BY_DERROR	-5013	Abnormally stopped by operation data error
cmERR_STOP_BY_IP	-5014	Abnormally stopped by other axis error during interpolation
cmERR_STOP_BY_PO	-5015	An overflow occurred in the PA/PB input buffer
cmERR_STOP_BY_AO	-5016	Out of range position counter during interpolation
cmERR_STOP_BY_EE	-5017	An EA/EB input error occurred (does not stop)
cmERR_STOP_BY_PE	-5018	An PA/PB input error occurred (does not stop)
cmERR_STOP_BY_SLVERR	-5019	Abnormally stopped because slave axis has been stopped
cmERR_STOP_BY_SEMG	-5020	Abnormally stopped by software emergency setting
cmERR_MOT_MAOMODE	-5110	Master output mode is not CW/CCW mode
cmERR_MOT_SLAVE_SET	-5120	Slave start fail (Motion state가 "Wait for Pulsar Input"으로 변하지 않습니다.)
cmERR_SPEED_RANGE_OVER	-5130	-
cmERR_INVALID_SPEED_SET	-5140	Speed setting value is not valid
cmERR_ACC_LOW_LIMIT_OVER	-5142	Acceleration setting value is too low
cmERR_ACC_HIGH_LIMIT_OVER	-5143	Acceleration setting value is too high
cmERR_DEC_LOW_LIMIT_OVER	-5144	Deceleration setting value is too low
cmERR_DEC_HIGH_LIMIT_OVER	-5145	Deceleration setting value is too high
cmERR_INVALID_IXMAP	-5150	Invalid interpolation map
cmERR_INVALID_LMMAP	-5160	Invalid List-Motion Map

cmERR_MOT_SEQ_SKIPPED	-5170	Motion command is skipped because the axis is already running.
cmERR_UNKNOWN	-9999	알수 없는 에러

2009년 10월 30일 3판 발행.

Hardware Support : hwteam@comizoa.com

Software Support : swteam@comizoa.com

㈜커미조아
www.comizoa.com
www.comizoa.co.kr
Tel) 042 – 936 – 6500~6
Fax) 042 – 936 – 6507