

1. CCS3.3 DSP/BIOS semaphore를 이용한 COM1 시리얼 통신

기존의 예제 프로그램을 바탕으로 실제 현장에 적용 가능한 BIOS프로그램 을 시작 하겠습니다.

이 장에서 사용할 H/W는 F28335내부에 있는 UART0 포트를 사용해 PC와 RS-232 통신을 프로세서간 동기를 맞추는데 사용하는 semaphore를 사용해 작성해 보겠습니다.

- Semaphore란

C 프로그램을 작성해 보신 분이라면 광역 변수의 0 과 1상태만 가지는 flag정도라 생각 하시면 됩니다. DSP/BIOS에서는 이 플래그 기능에 전 시간에 설명한 시간 지연(TSK_sleep()) 기능이 추가 되어 있습니다. 결론은 BIOS에서 각 TASK간 이벤트 동기와 시간 지연을 동시에 사용 할수 있게 만든 함수라 생각 하시면 됩니다.

- semaphore 설정 방법

CCS BIOS TOOL에서 1개 또는 여러 개의 semaphore를 사용자에게 맞춰 생성 합니다.

- 변수명, 초기치..

- semaphore 사용법

1. SEM_post(&sem_com1_rx_eflg);

이벤트를 발생 시킬때 사용하는 함수 입니다.

Item1: CCS BIOS에서 생성한 SEM 변수명 을 ()에 사용 합니다.

2. SEM_pend(&sem_com1_rx_eflg, SYS_FOREVER); <- 리턴 있음

이벤트를 대기 할때 사용하는 함수 입니다.

Item1: CCS BIOS에서 생성한 SEM 변수명 을 ()에 사용 합니다.

Item2: 함수내 시간 지연 방법 설정 입니다.

SYS_FOREVER : 이벤트가 SEM_post() 될때까지 영원히 기다림.

1 이상 숫자 : 1은 1TICK을 의미 하며 이 TICK동안 SEM_post()를 기다리다 탈출 합니다. 위 경우는 리턴 되는 값을 확인하여 처리 합니다.

리턴: 0 이면 지정한 TICK동안 이벤트가 발생 하지 않았음.

1 이면 지정한 TICK동안 이벤트가 발생함..

- TEST 환경(115200Bps, 8 Bit, NON Parity, 1STOP)

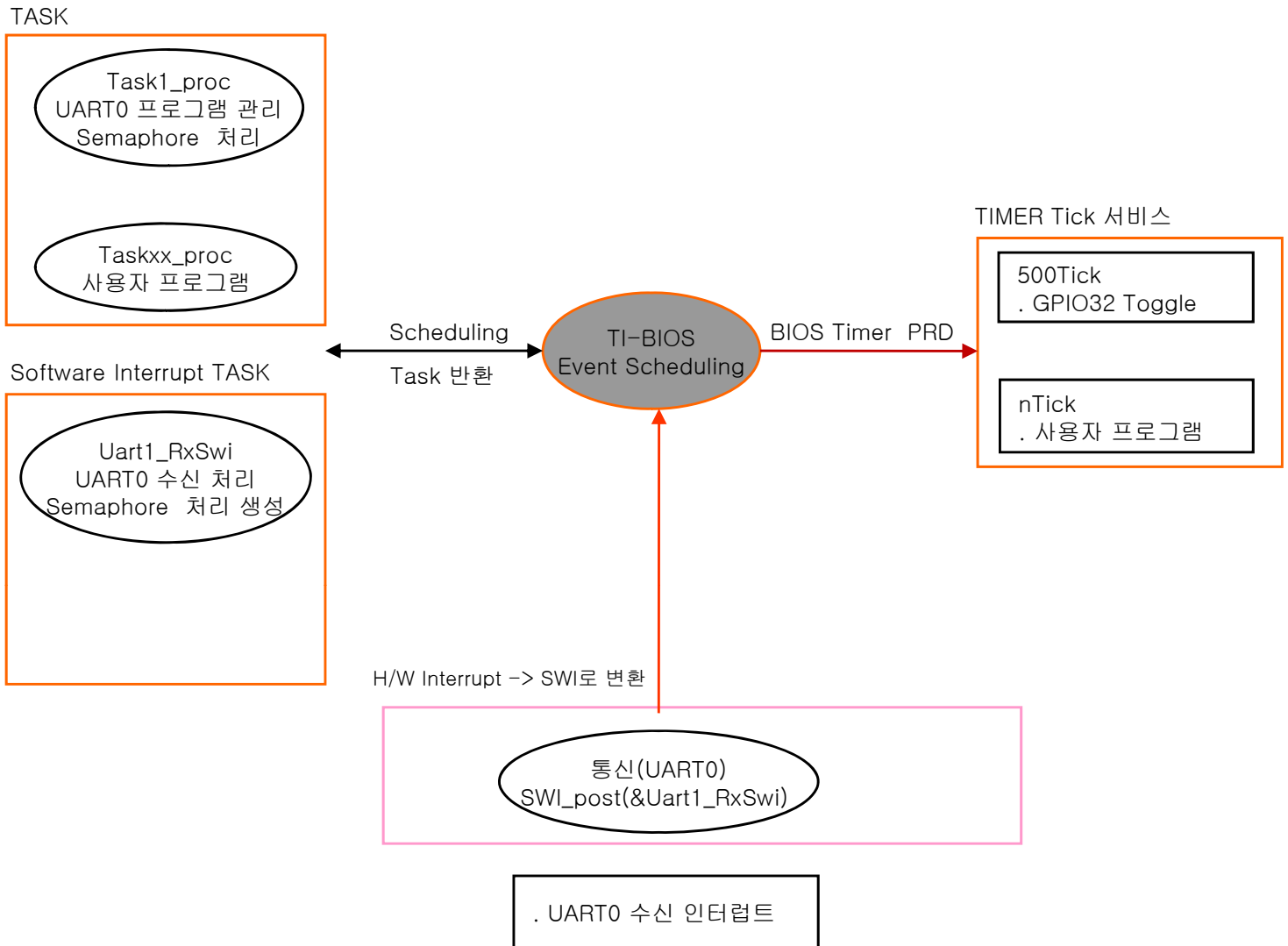
- PC : RS-232 PORT

- DSP 보드 : UART0 RS-232

1. 디렉토리 구성

..Wcmd	: Linker 커맨드 파일
..DSP2833x_headers	: Chip관련 헤더 파일 및 헤더용 Linker 커맨드 파일
..Winclude	: 사용자 인클루드 파일
..Wsemprj	: 사용자 프로젝트 파일 및 실행 파일(.HEX)
..Wsemsrc	: 사용자 소스 파일

- CCS3.3 DSP/BIOS Semaphore 구성



1. Main_Bios.c 설명

```

void Uart1_RxSwi(void)                                     <- SWI 스케줄 함수 명 :: 실제 UART0 수신 처리
{
    int i,rsr,len;

    rsr = SciaRegs.SCIRXST.all & 0xff;                   <- 수신 상태 레지스터
    if((rsr & 0x9c) != 0){                                <- 에러 검사
        SciaRegs.SCICTL1.bit.SWRESET=0;                 <- reset
        SciaRegs.SCICTL1.bit.SWRESET=1;                 <- non reset
    }
    else{                                                 <- 에러 없음.
        en = (SciaRegs.SCIFFRX.all >> 8) & 0x1f;       <- 수신 FIFO에서 수신된 개수 읽음.
        for(i = 0; i < len; i++){                         <- 데이터중 최종 만 읽음.
            com1_rx_data = SciaRegs.SCIRXBUF.all;
            SEM_post(&sem_com1_rx_eflg);                 <- semaphore 생성(수신된 데이터 있음.)
        }
    }
    SciaRegs.SCIFFRX.bit.RXFFOVRCLR=1;                  <- Clear Overflow flag
    SciaRegs.SCIFFRX.bit.RXFFINTCLR=1;                  <- Clear Interrupt flag
}
    
```

2. 소스코드 설명(Main_Bios.c)

```
#include "DSP2833x_Device.h" <- DSP 초기화 및 설정 관련
#include "F28335_example.h" <- 사용자 외부 함수, 변수, 정의 관리

void main(void)
{
    InitSysCtrl();           <- CPU 클럭 설정((30*10) / 2 = 150M)
    InitPieCtrl();          <- 인터럽트 관련 초기화
    InitWatchdog();        <- watch-dog 설정 및 초기화
    InitGpio();            <- CPU I/O 설정(IN,OUT,기본기능..) _EX_BUS_ON정의에 따라 외부 버스 ON
    InitXintf();           <- 내부 주변 디바이스 클럭 설정 및 외부 버스 타이밍 설정

    ** DSP/BIOS 관련 설정 **
    memcpy(&secureRamFuncs_runstart,
           &secureRamFuncs_loadstart,&secureRamFuncs_loadend - &secureRamFuncs_loadstart);
    InitFlash();

    com_init(_COM1,115200L,NON_P,8,1);<- COM1 설정(DSP2833x_sci.c)

    asm(" EALLOW");        <- Enable EALLOW protected register access
    GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 0;<- GPIO032 GPIO
    GpioCtrlRegs.GPBDIR.bit.GPIO32 = 1; <- GPIO32 output
    GpioDataRegs.GPBSET.bit.GPIO32 = 1; <- GPIO32 pin is set to 1
    asm(" EDIS");         // Disable EALLOW protected register access

    ** DSP/BIOS에서 TINT2,DLOGINT를 사용 하므로 BIOS사용 인터럽트 허가 **
    SetDBGIER(IER | 0x6000); <- Enable everything in IER, plus TINT2 and DLOGINT
    *(volatile unsigned int *)0x00000C14 |= 0x0C00;<- Set TIMER2 FREE=SOFT=1

    ** 아래 main()를 종료 하면 DSP/BIOS가 동작.. **
}

void UserInit(void){      <- 이 함수는 리셋시 DSP/BIOS 초기화 부분에서 한번 수행 후
                           DSP/BIOS관련 및 사용자 초기화 함수 추가
}

void task1_proc(void){    <- 이 함수는 스케줄러에 관리 되는 TASK (OM1 수신 처리)
}

void lo_Blink(void){     <- PRD 스케줄 함수 명
}

Void Uart1_RxSwi(void){  <- COM1_RX SWI 스케줄 함수 명
}

```

3. 소스코드 설명

1. DefaultIsr_BIOS.c

```
void SCIRXINTA_ISR(void){ <- UART0 H/W(PIE9.1) 인터럽트를 바로 처리하지 않고 SWI로 변환하여 실행.
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP9; <- Must acknowledge the PIE group
    SWI_post(&UART1_Rx_Swi); <- SWI 로 변환
}
```

2. Main_Bios.c 설명

```
void Uart1_RxSwi(void)          <- SWI 스케줄 함수 명 :: 실제 UART0 수신 인터럽트 처리
{
  int i,rsr,len;

  rsr = SciaRegs.SCIRXST.all & 0xff;      <- 수신 상태 레지스터
  if((rsr & 0x9c) != 0){                <- 에러 검사
    SciaRegs.SCICTL1.bit.SWRESET=0;      <- reset
    SciaRegs.SCICTL1.bit.SWRESET=1;      <- non reset
  }
  else{                                  <- 에러 없음.
    en = (SciaRegs.SCIFFRX.all >> 8) & 0x1f; <- 수신 FIFO에서 수신된 개수 읽음.
    for(i = 0; i < len; i++){           <- 데이터중 최종 만 읽음.
      com1_rx_data = SciaRegs.SCIRXBUF.all;
      SEM_post(&sem_com1_rx_eflg);      <- semaphore 생성(수신된 데이터 있음.)
    }
  }
  SciaRegs.SCIFFRX.bit.RXFFOVRCLR=1;    <- Clear Overflow flag
  SciaRegs.SCIFFRX.bit.RXFFINTCLR=1;    <- Clear Interrupt flag
}

void task1_proc(void)            <- 사용자 TASK :: UART0 관리
{
  while(1) {
    SEM_pend(&sem_com1_rx_eflg,SYS_FOREVER);<- 1바이트가 수신될때 까지 무한정 기다림
    com1_putch(com1_rx_data);          <- 받은 데이터 송신(폴링 방식)
  }
}

↓ TICK을 지정할 경우

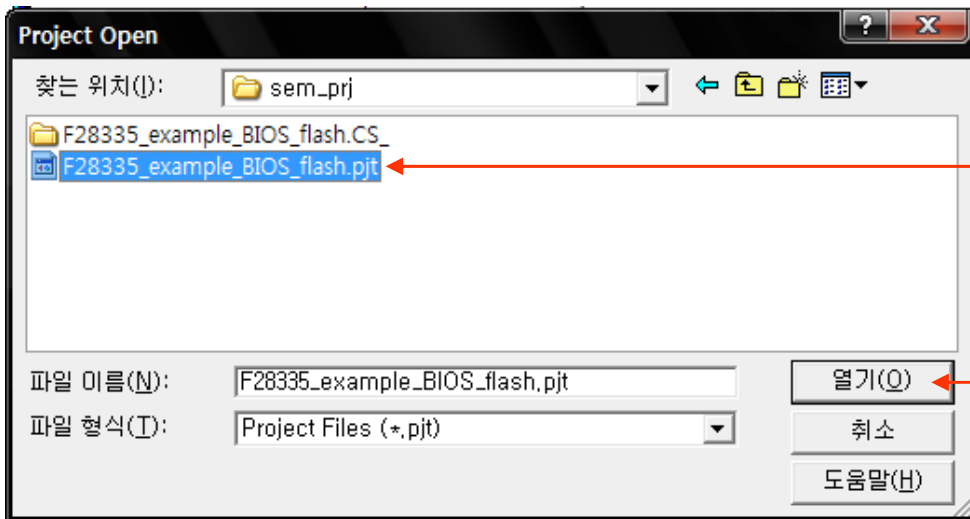
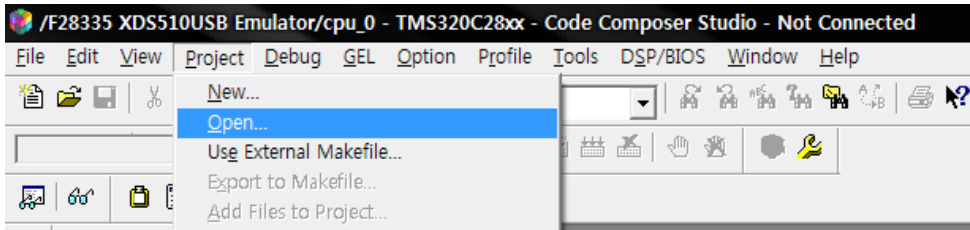
void task1_proc(void)           <- 사용자 TASK :: UART0 관리
{
  while(1) {
    if(SEM_pend(&sem_com1_rx_eflg,500)){ <- 1바이트가 수신될때 까지 500 TICK 기다린 후 처리(리턴참조)
      com1_putch(com1_rx_data);          <- 받은 데이터 송신(폴링 방식)
    }
  }
}
```

* CCS3.3 DSP/BIOS 구성 및 설명

1. Setup CCStudio v3.3 이나 CCSstudio3.3을 실행 합니다.



2. 아래와 같이 Project를 오픈 합니다.(Project->Open)



FLASH에서 실행되는
프로젝트

버튼 클릭

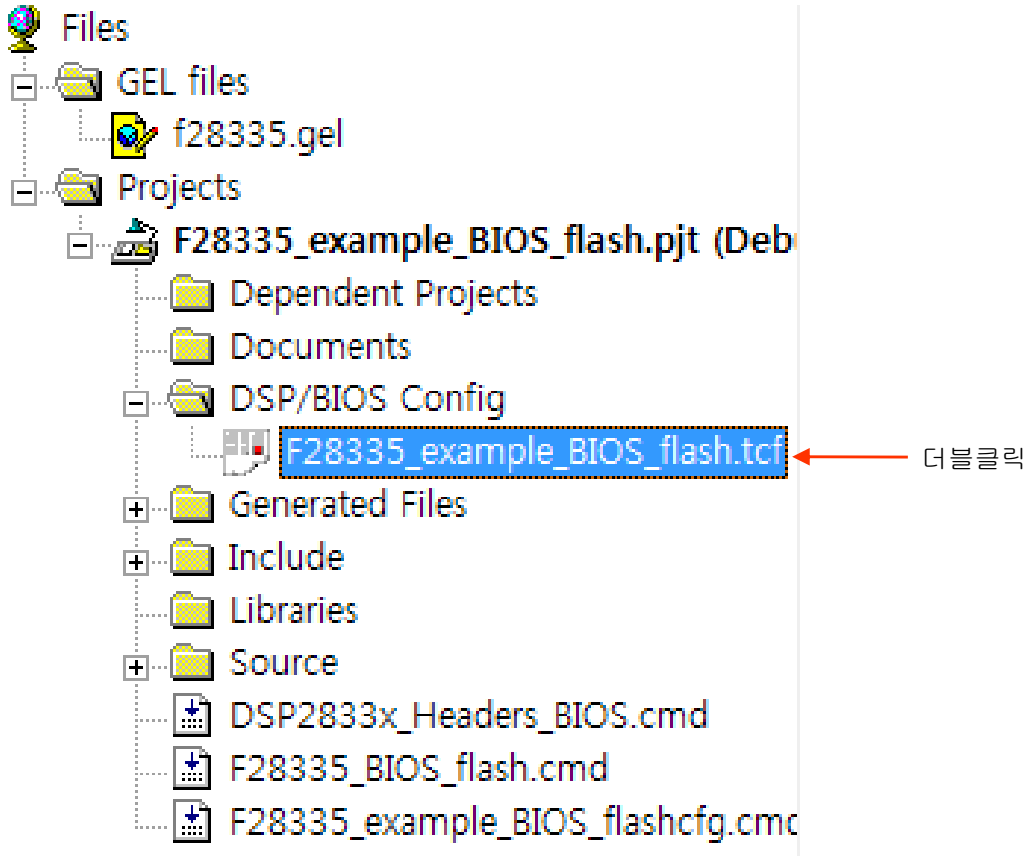
3. Projects Source 파일 구성

The image shows a file explorer window with the following structure:

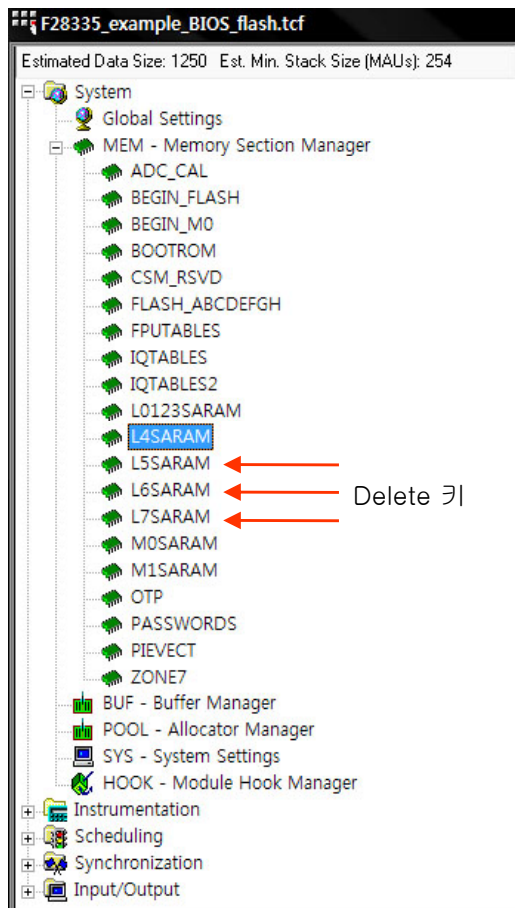
- Files
 - GEL files
 - Projects
 - F28335_example_BIOS_flash.pjt (Debug)
 - Dependent Projects
 - Documents
 - DSP/BIOS Config ← DSP/BIOS Config 파일
 - Generated Files
 - Include
 - Libraries
 - Source
 - CodeStartBranch.asm ← Watch-dog disable 후 C를 실행 할때 사용, 필요시 project->build option에서 code entry point에서 등록하여 사용
 - DefaultIsr_BIOS.c ← 인터럽트 서비스 루틴
 - DelayUs.asm ← DelayUs() 함수 지원
 - DSP2833x_GlobalVariableDefs.c ← 전역데이터 및 데이터 섹션 정의
 - DSP2833x_sci.c ← 통신 프로그램 소스
 - Flash.c ← Flash Memory 관련 지원 및 초기화
 - Gpio.c ← CPU I/O핀 초기화
 - Main_BIOS.c ← Main() 프로그램
 - Passwords.asm ← Flash Passwords 관련
 - PieCtrl_BIOS.c ← CPU 인터럽트 초기화
 - SetDBGIER.asm ← BIOS 인터럽트 지원
 - SysCtrl.c ← CPU 클럭 설정
 - Watchdog.c ← Watch-Dog 관련 초기화
 - Xintf.c ← 외부 버스 초기화(wait 설정)
 - DSP2833x-Headers_BIOS.cmd ← DSP/BIOS Config에서 컴파일시 생성
 - F28335_BIOS_flash.cmd ← 기본 CMD 파일 정의
 - F28335_example_BIOS_flashcfg.cmd ← DSP/BIOS Config에서 컴파일시 생성

4. DSP/BIOS Config->*.tcf 를 실행 한다.

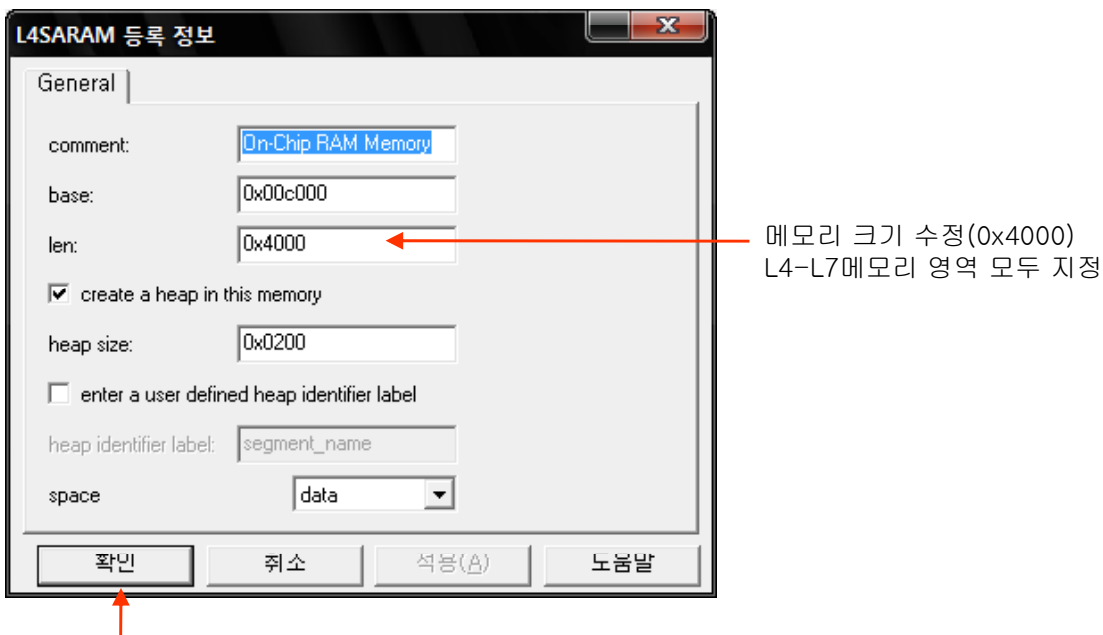
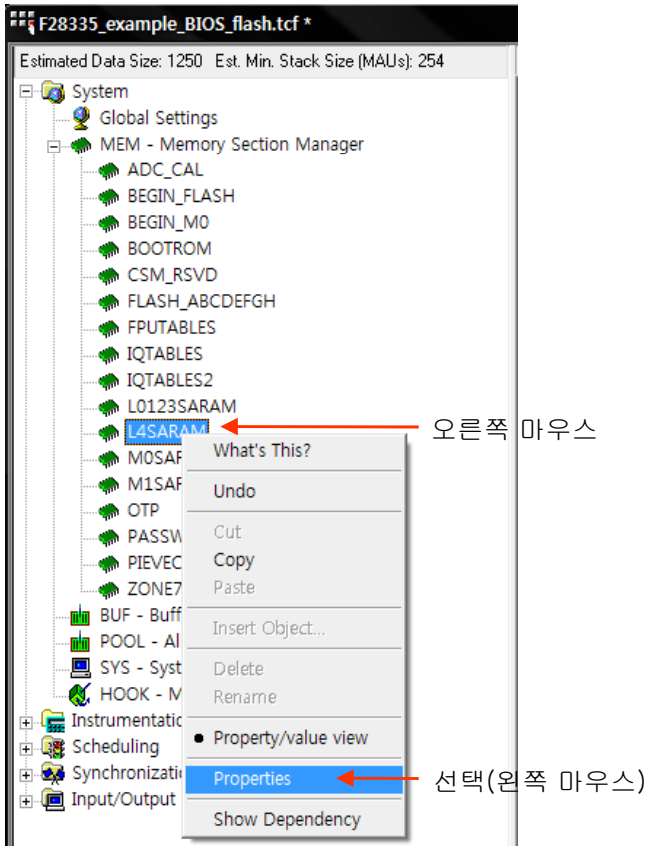
TASK생성, LOG_printf 생성법은 기존 자료 참조 하세요.



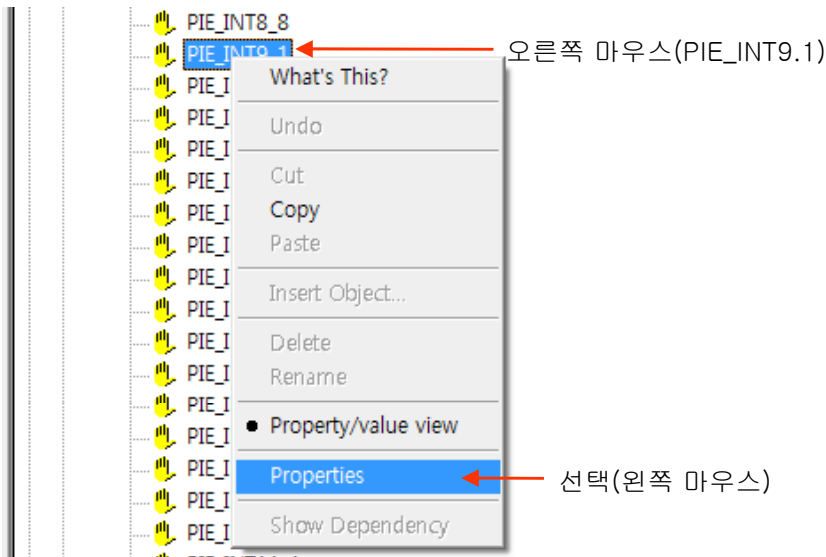
5. System->MEM->L5,L6,L7SARAM을 삭제 한다.(마우스 고정후 삭제 키)
BIOS 사용 메모리 확장



6. System->MEM->L4SARAM 내용 수정



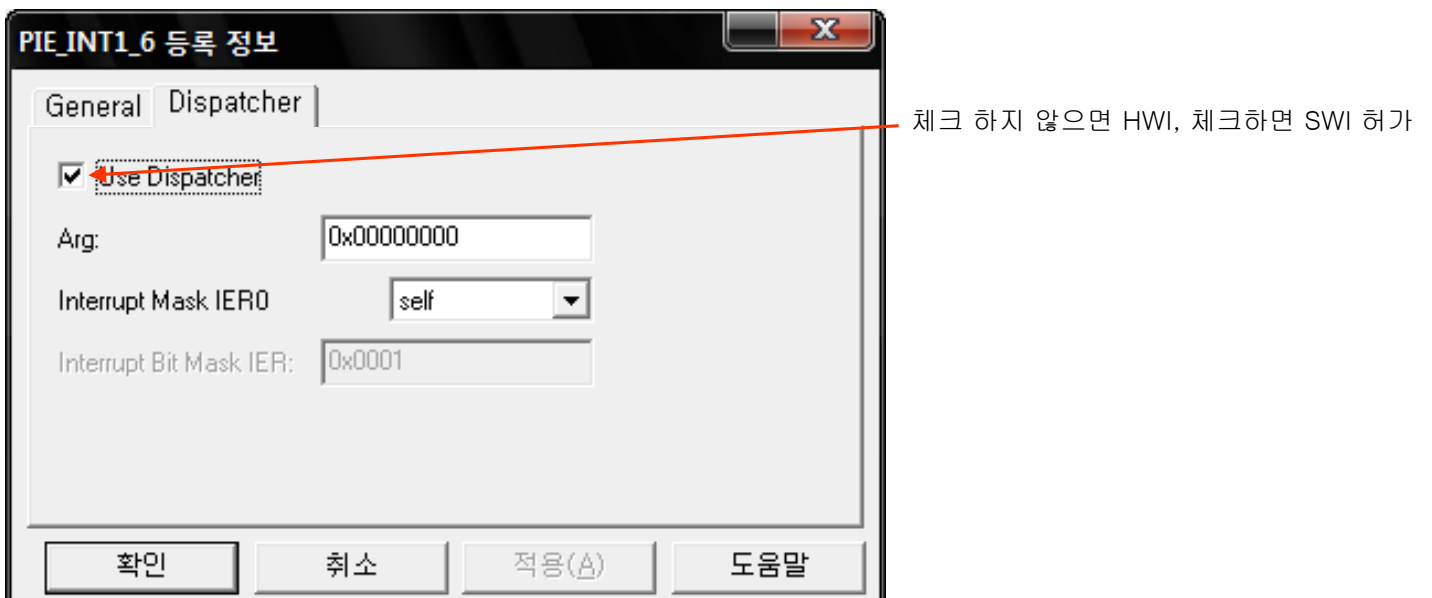
7. 하드웨어 인터럽트를 SWI로 변환(Scheduling ->HWI->PIE INTERRUPTS->PIE_INT9.1)



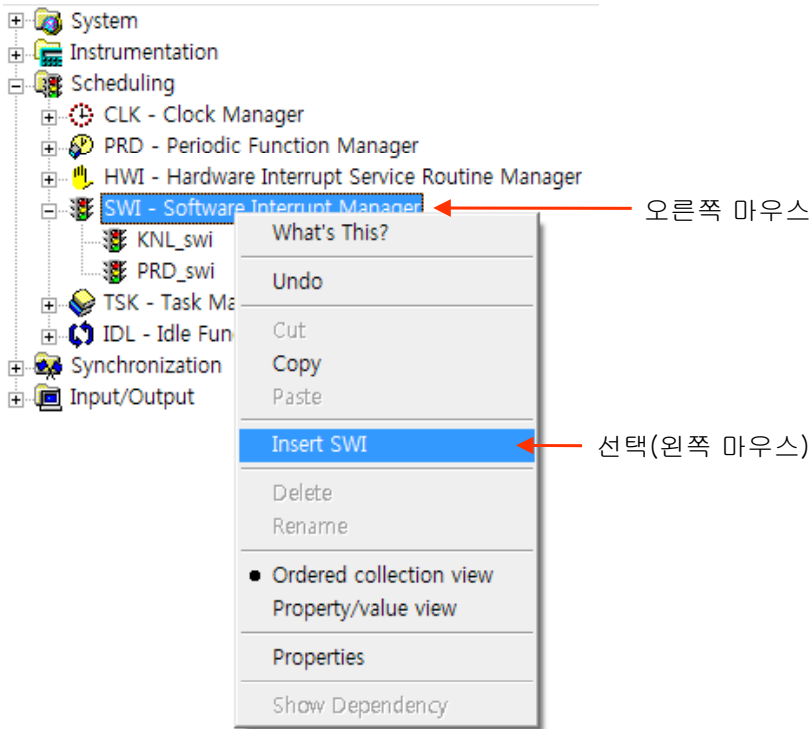
* 하드웨어 인터럽트 등록(HWI)



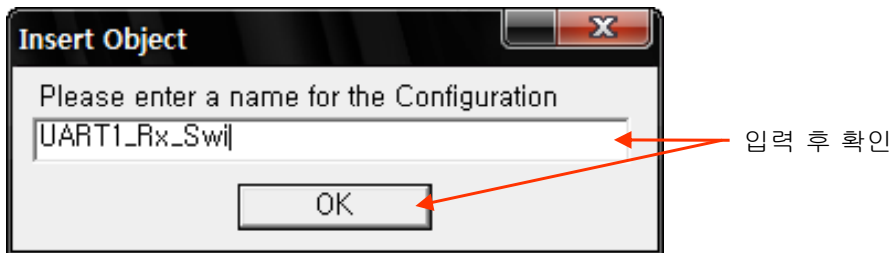
* 소프트웨어 인터럽트 허가(SWI)



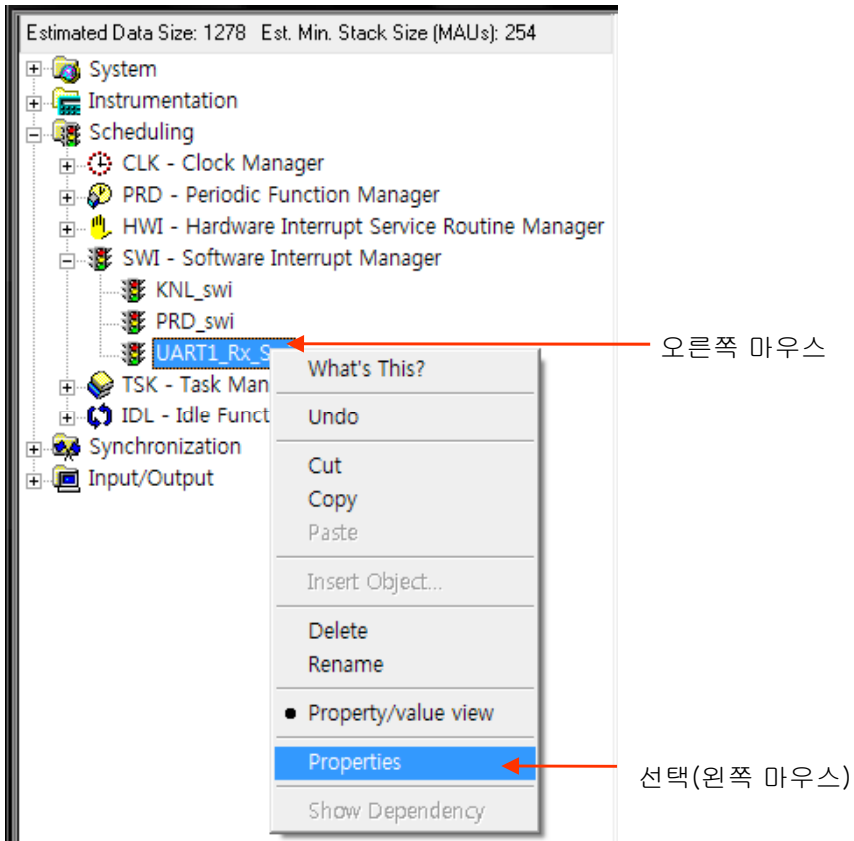
8. Scheduling SWI 등록(. Scheduling ->Software Interrupt Manager)



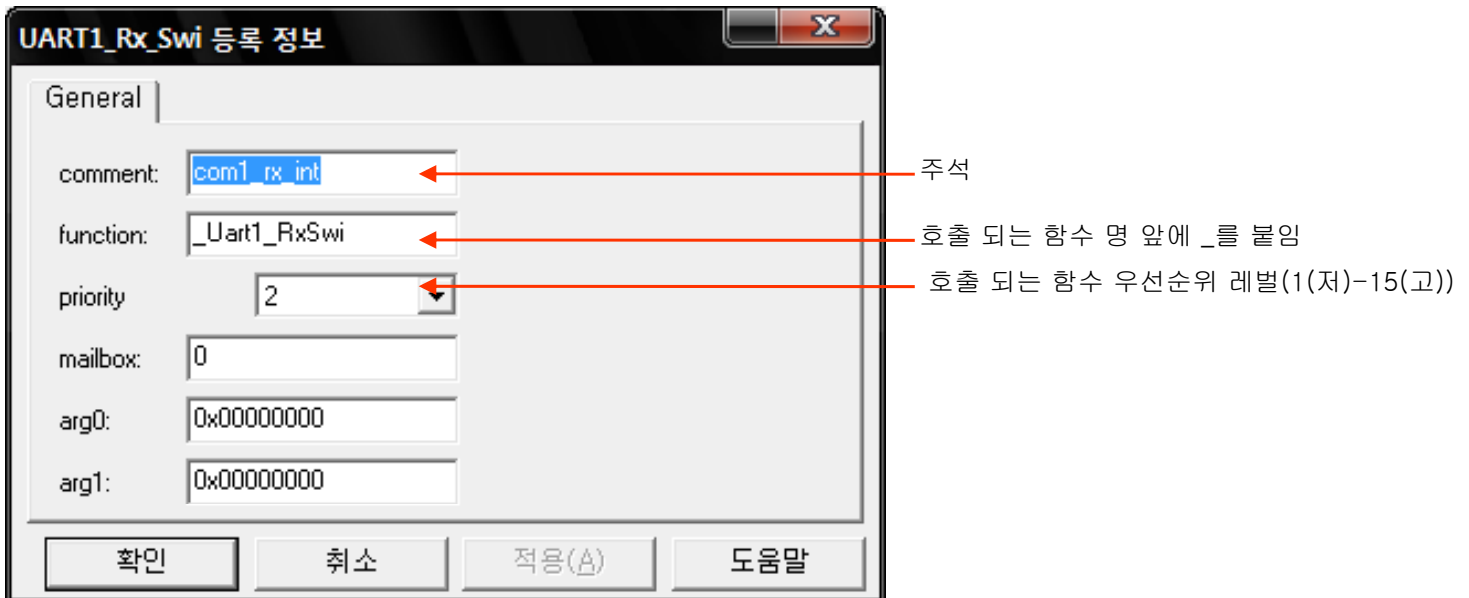
* SWI 관리 명을 입력 한다.(프로그램 에서 호출 주소가 됨.)



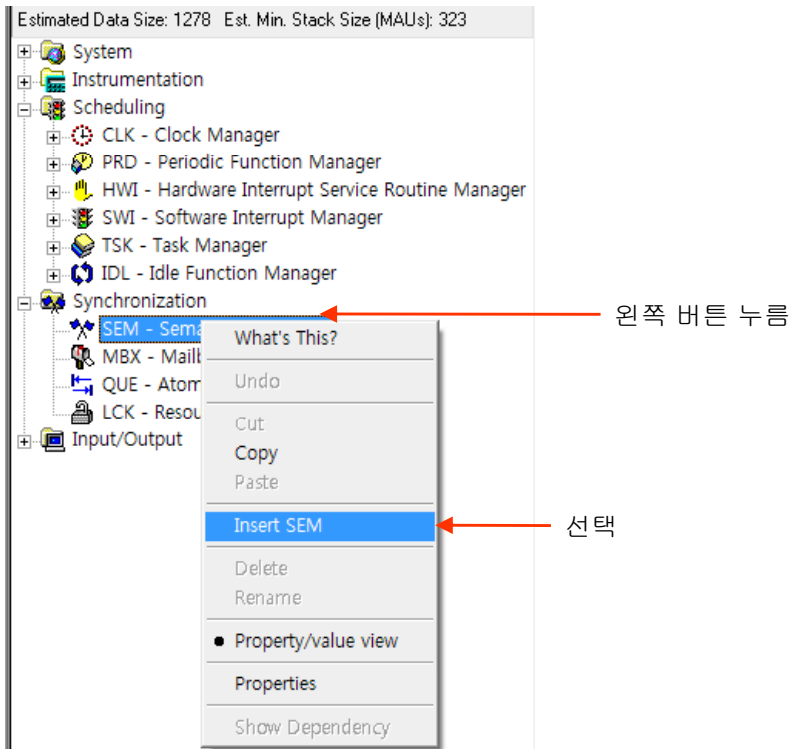
* 생성한 SWI에 사용자 환경을 설정 한다.(UART1_Rx_Swi 선택후 오른쪽 버튼)



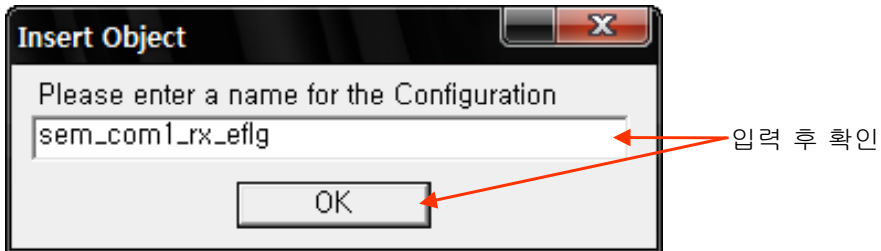
* General 에서 기본 정보를 설정 한다.



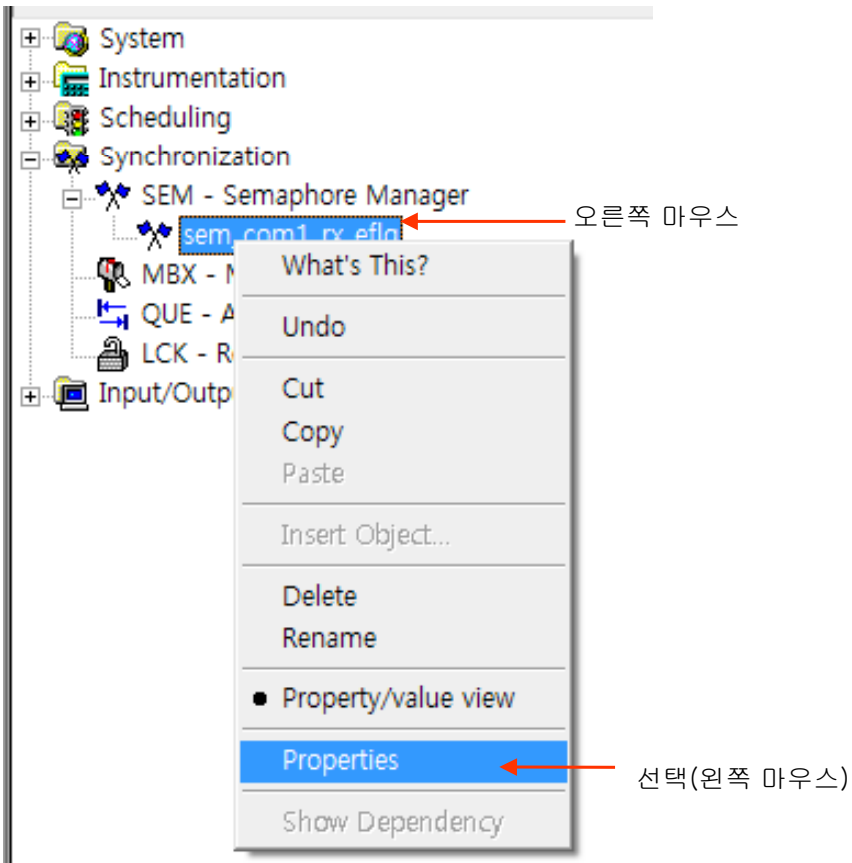
9. Semaphore 등록(Synchronization->SEM semaphore Manager)



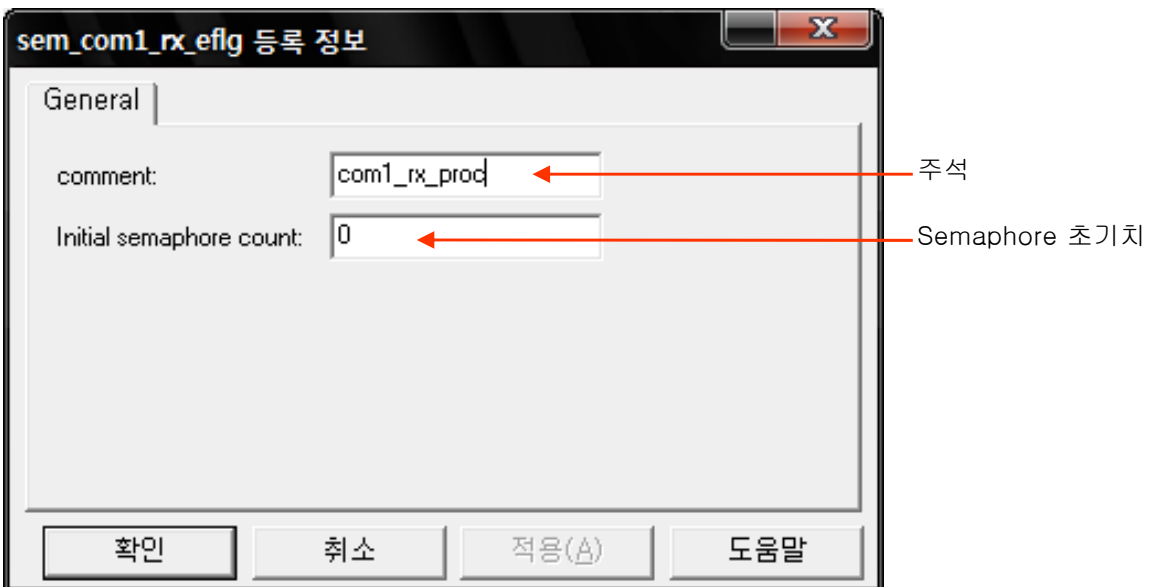
* semaphore 명을 입력 한다.(프로그램 에서 호출 변수가 됨.)



* 생성한 SEM에 사용자 환경을 설정 한다.(sem_com1_rx_eflg 선택후 오른쪽 버튼)



* General 에서 기본 정보를 설정 한다.



* Scheduling SWI에서 생성된 UART1_Rx_Swi 를 확인후 *.tcf 파일을 종료 합니다.

The screenshot shows the Code Composer Studio interface for the file `F28335_example_BIOS_flash.tcf`. The left pane displays a project tree with the following structure:

- System
- Instrumentation
- Scheduling
 - CLK - Clock Manager
 - PRD - Periodic Function Manager
 - HWI - Hardware Interrupt Service Routine Manager
 - SWI - Software Interrupt Manager
 - KNL_swi
 - PRD_swi
 - UART1_Rx_Swi**
 - TSK - Task Manager
- IDL - Idle Function Manager
- Synchronization
- Input/Output

The middle pane shows the properties for `UART1_Rx_Swi`:

Property	Value
comment	com1_rx_int
function	_Uart1_RxSwi
priority	2
mailbox	0
arg0	0x00000000
arg1	0x00000000

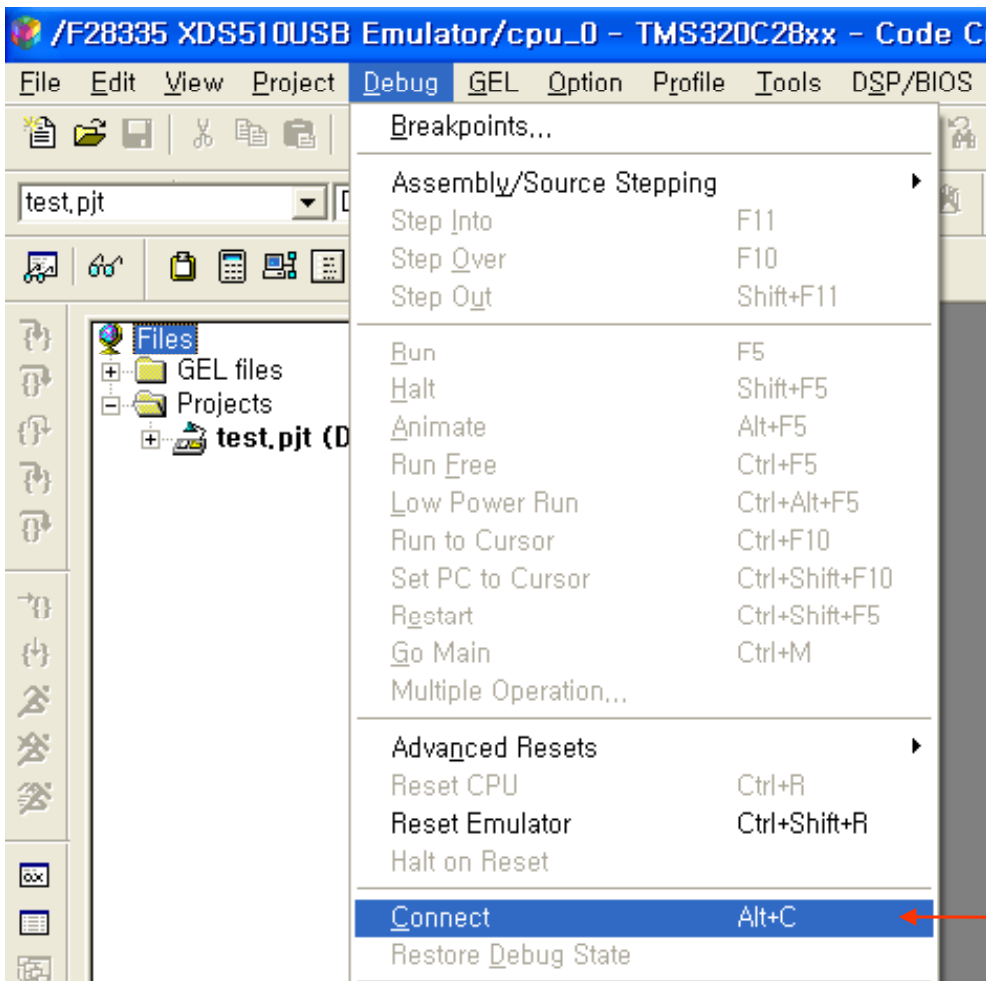
The right pane shows the textual configuration script with the following code:

```
utils.loadPlatform("ti.platforms.ezdsp28335");  
  
/* The following DSP/BIOS Features are enabled. */  
bios.enableRealTimeAnalysis(prog);  
bios.enableRtdw(prog);  
bios.enableTaskManager(prog);  
  
bios.GBL.CLKIN = 30000;  
bios.GBL.CALLUSERINITFXN = 1;  
bios.GBL.USERINITFXN = prog.extern("UserInit");  
bios.GBL.MODIFYPLL2 = 0;  
bios.MEM.NOMEMORYHEAPS = 0;  
bios.MEM.instance("SRAM").destroy();  
bios.MEM.create("ZONE7");  
bios.MEM.instance("ZONE7").comment = "External - 128Kw SRAM";  
bios.MEM.instance("ZONE7").base = 0x200000;  
bios.MEM.instance("ZONE7").len = 0x20000;
```

The screenshot shows a dialog box titled "Code Composer Studio" with a warning icon. The text inside the dialog asks: "변경 내용을 F28335_example_BIOS_flash.tcf에 저장할까요?" (Save changes to F28335_example_BIOS_flash.tcf?). There are three buttons: "예 (Y)" (Yes), "아니오 (N)" (No), and "취소" (Cancel). A red arrow points to the "예 (Y)" button.

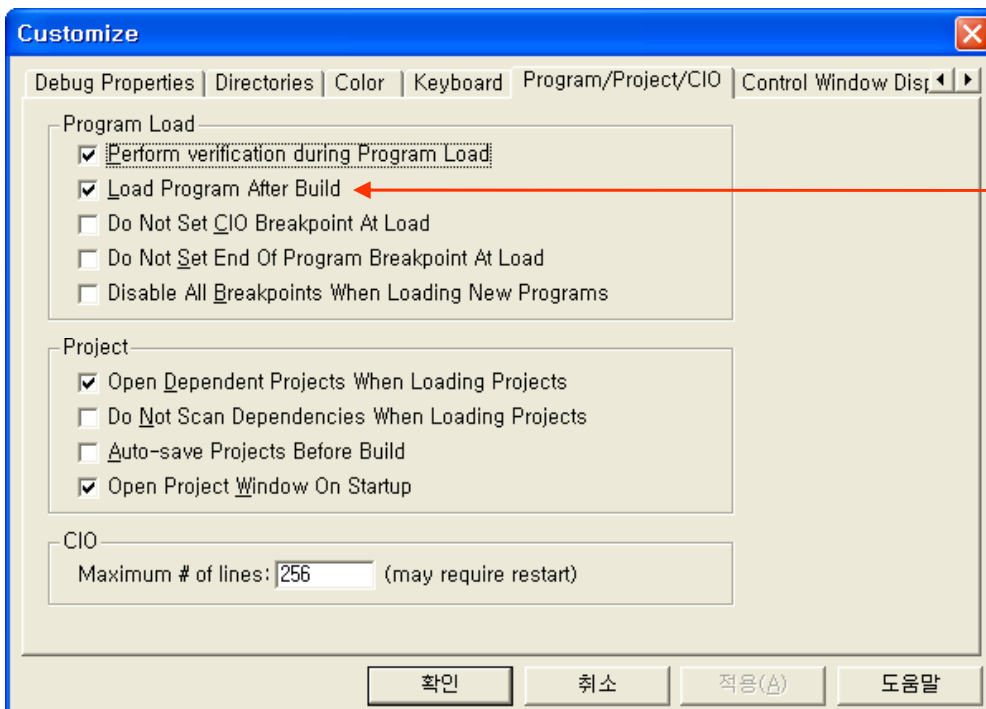
- CCS3.3 DSP/BIOS 실행

1. JTAG 및 에뮬레이터를 연결 합니다.



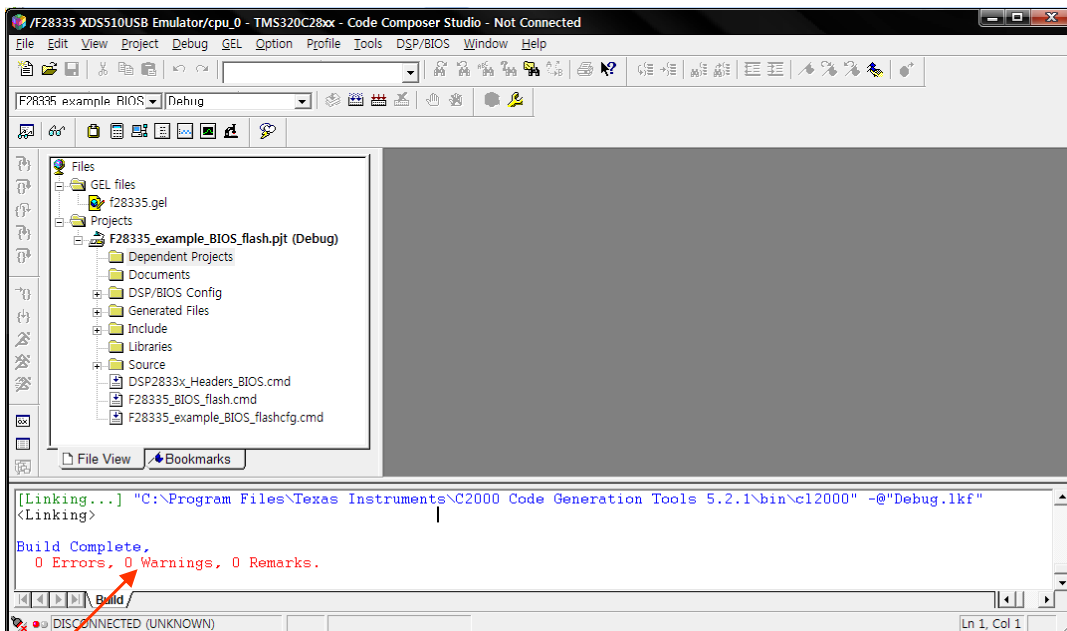
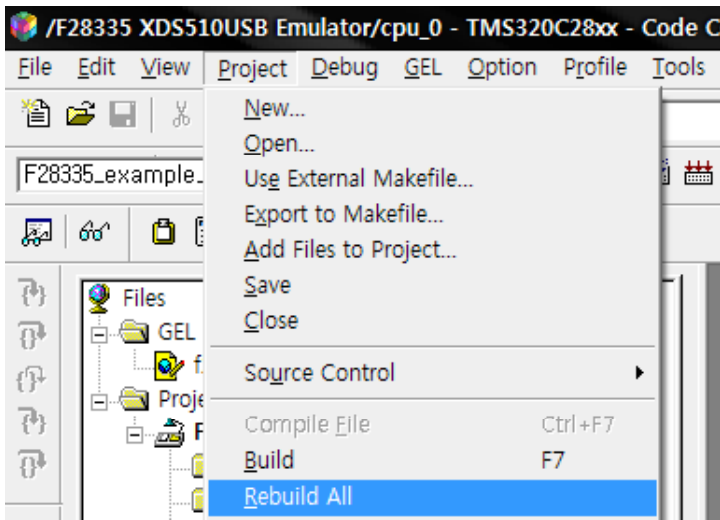
선택후
마우스 왼쪽 버튼 클릭

2. 내부럼 으로 프로그램을 실행할 경우 아래와 같이 설정 합니다.(Option->Customize)



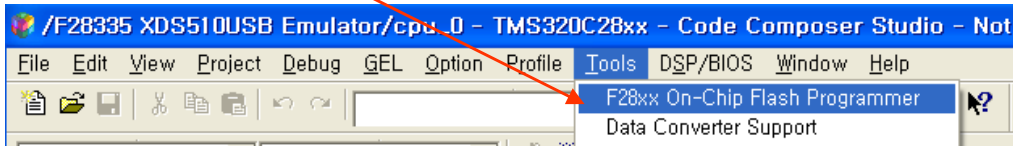
체크

3. 컴파일 하기(Project->Rebuild All)

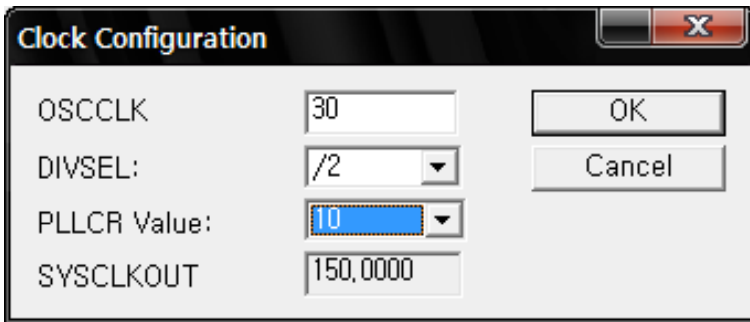


에러 확인

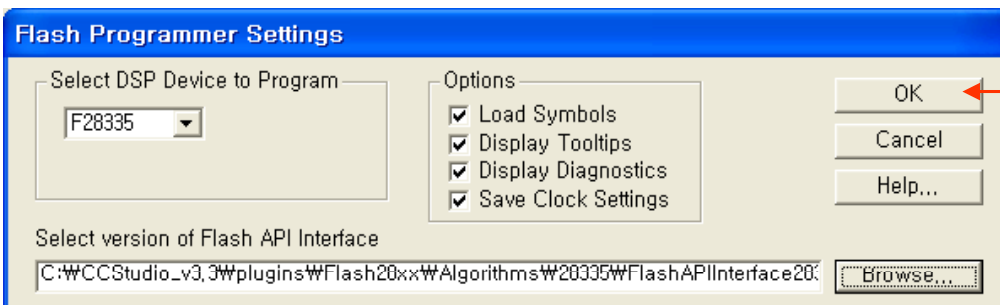
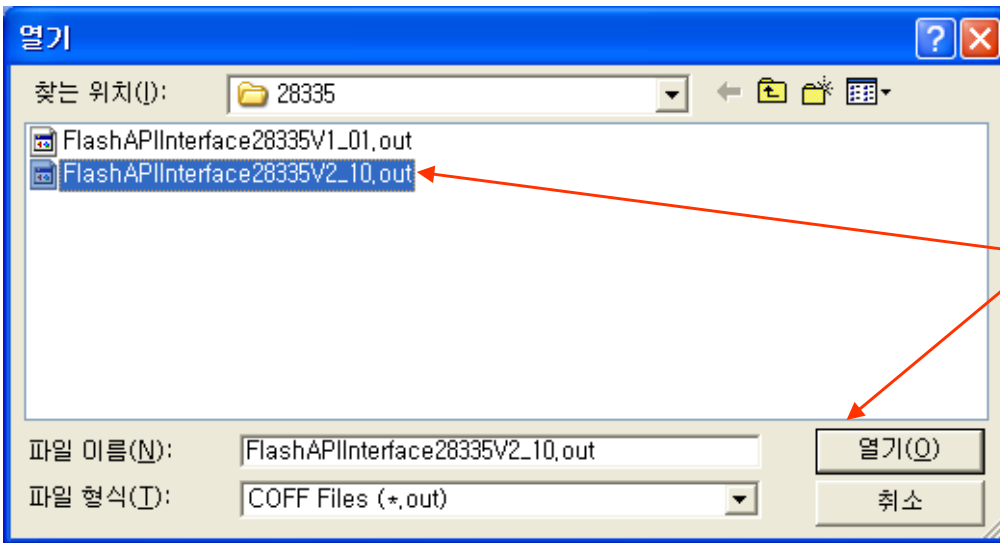
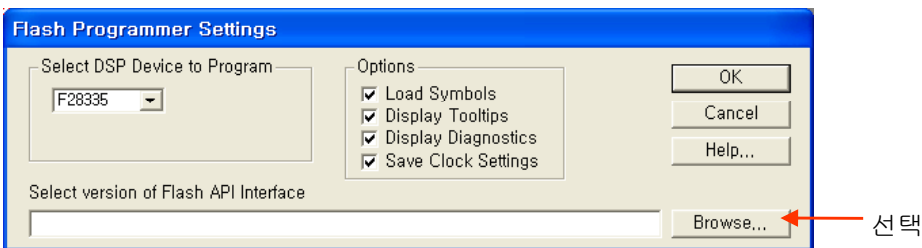
4. FLASH에 프로그램 하기

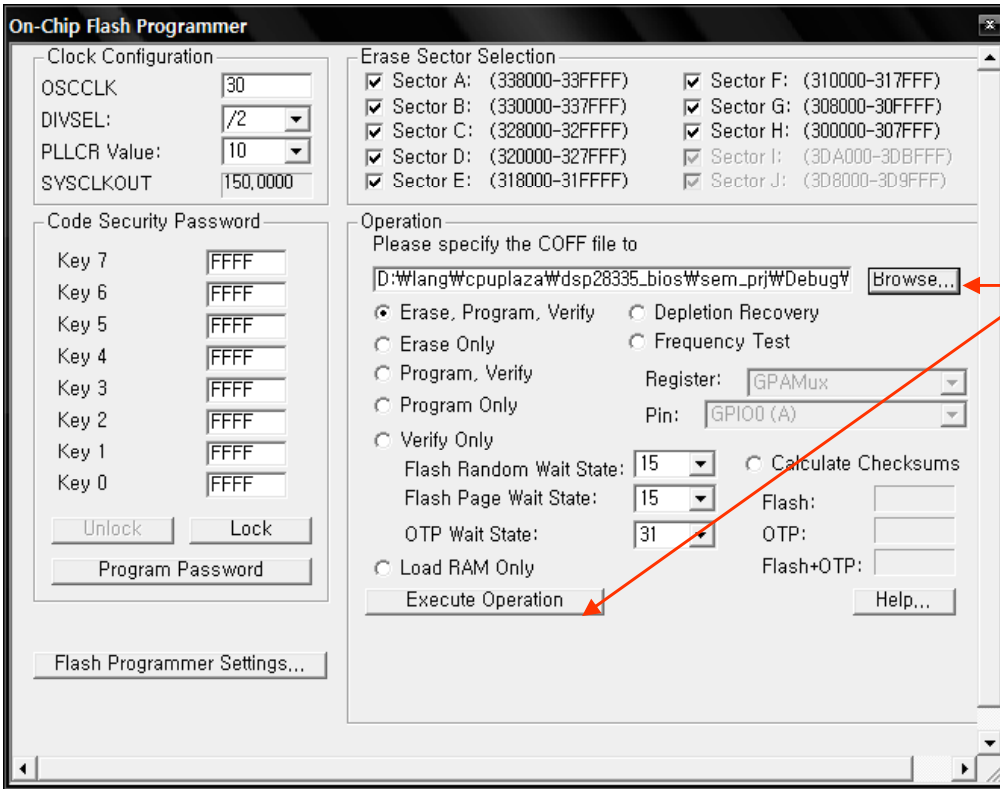


* 아래 CLOCK 설정 메뉴를 사용자에게 맞게 설정 합니다.



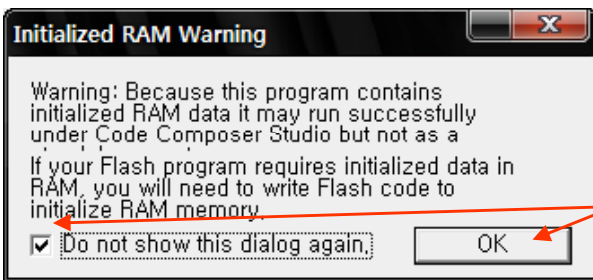
* API Interface 파일을 등록 합니다.



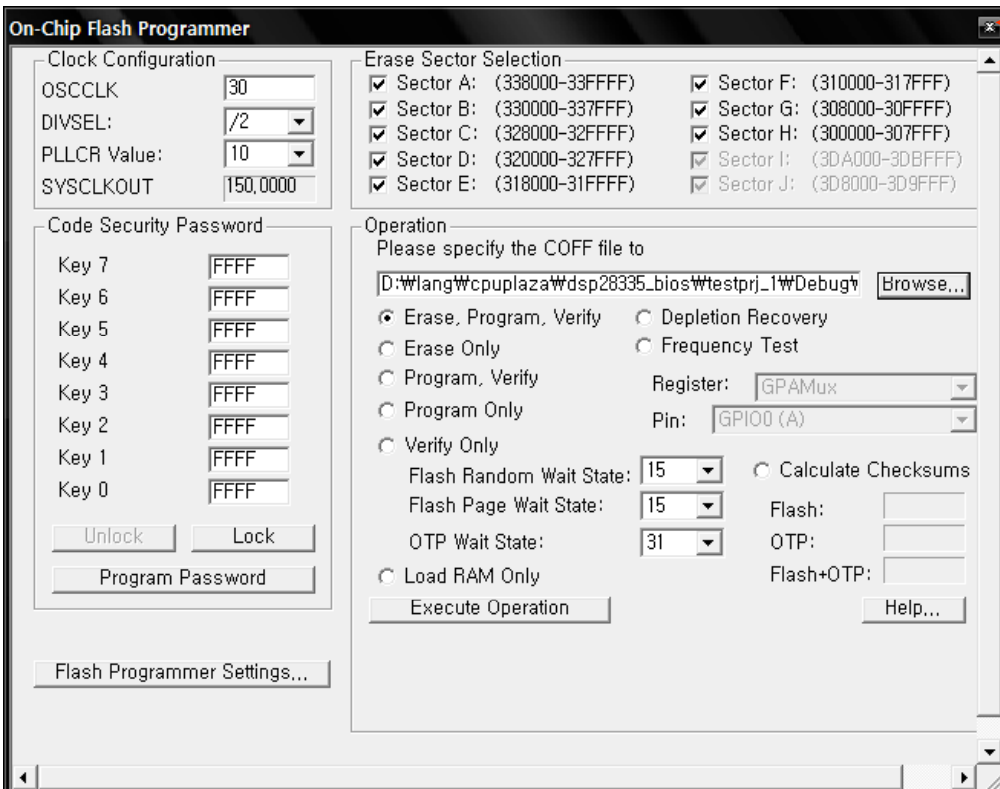


Browse.. 창에서 파일을 선택후 Execute Operation 탭을 실행합니다.

* TI 실행 파일은 *.OUT로 현재 작업 디렉토리 ..\Wdebug\ 에 있습니다.

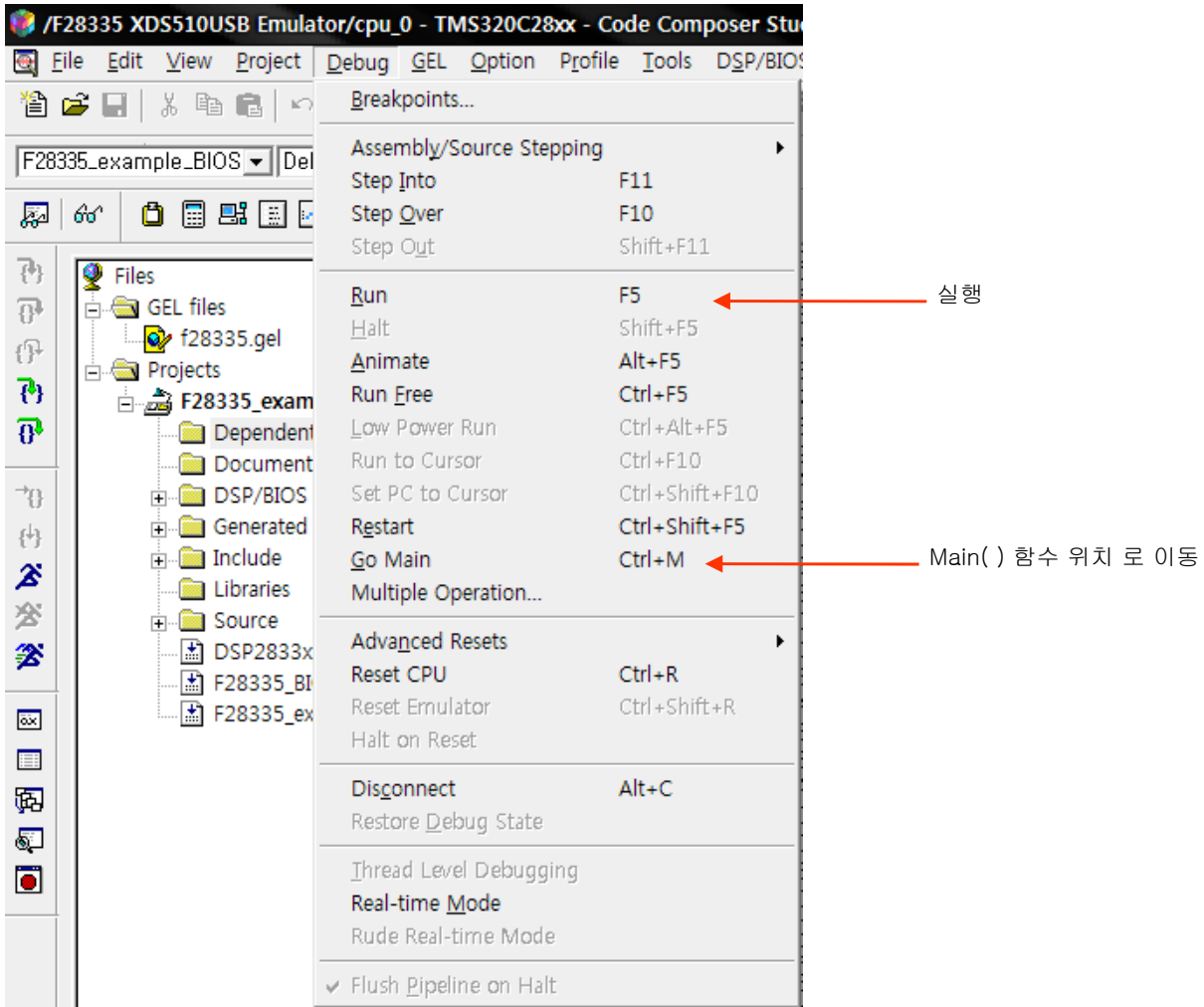


체크후 확인



닫음

5. 프로그램을 로딩후 Debug 탭에서 Go Main -> RUN 기능을 실행 합니다.



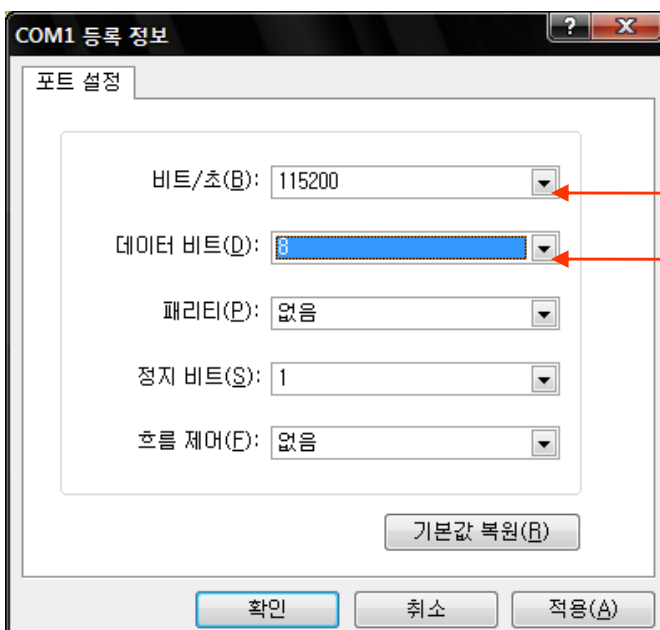
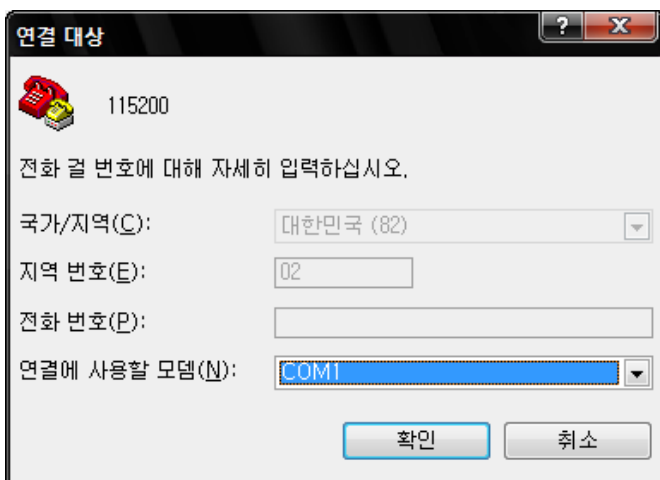
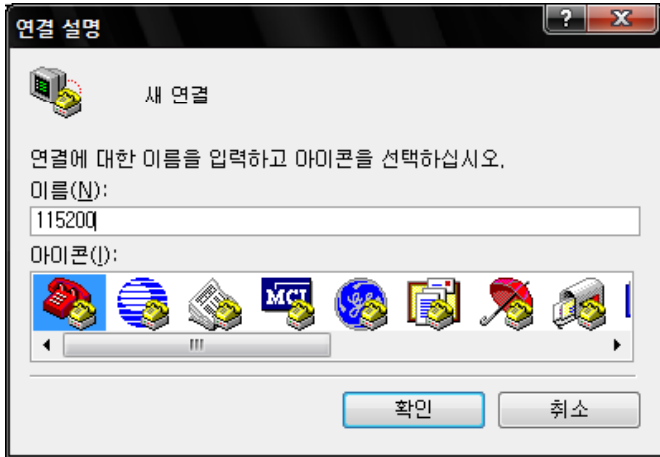
6. BIOS 디버그 방법은 기존 BIOS 자료를 참고 하세요.

* TEST 프로그램 실행

1. 하이퍼 터미널을 시작 합니다.



2. 하이퍼 터미널에서 통신 포트를 설정합니다.



보레이트

데이터 비트는 반드시 재 설정(하이퍼 에러)

3. 하이 터미널이 실행되면 아무 키나 누르면 반송 됩니다.

