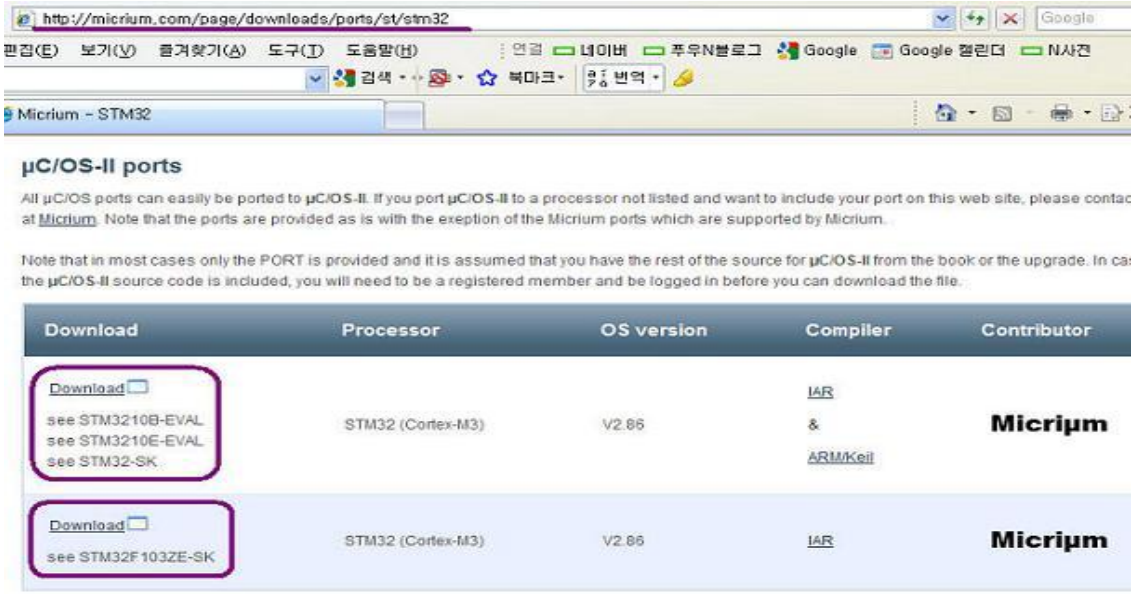
uCOS-II porting 실습

2009.11.20

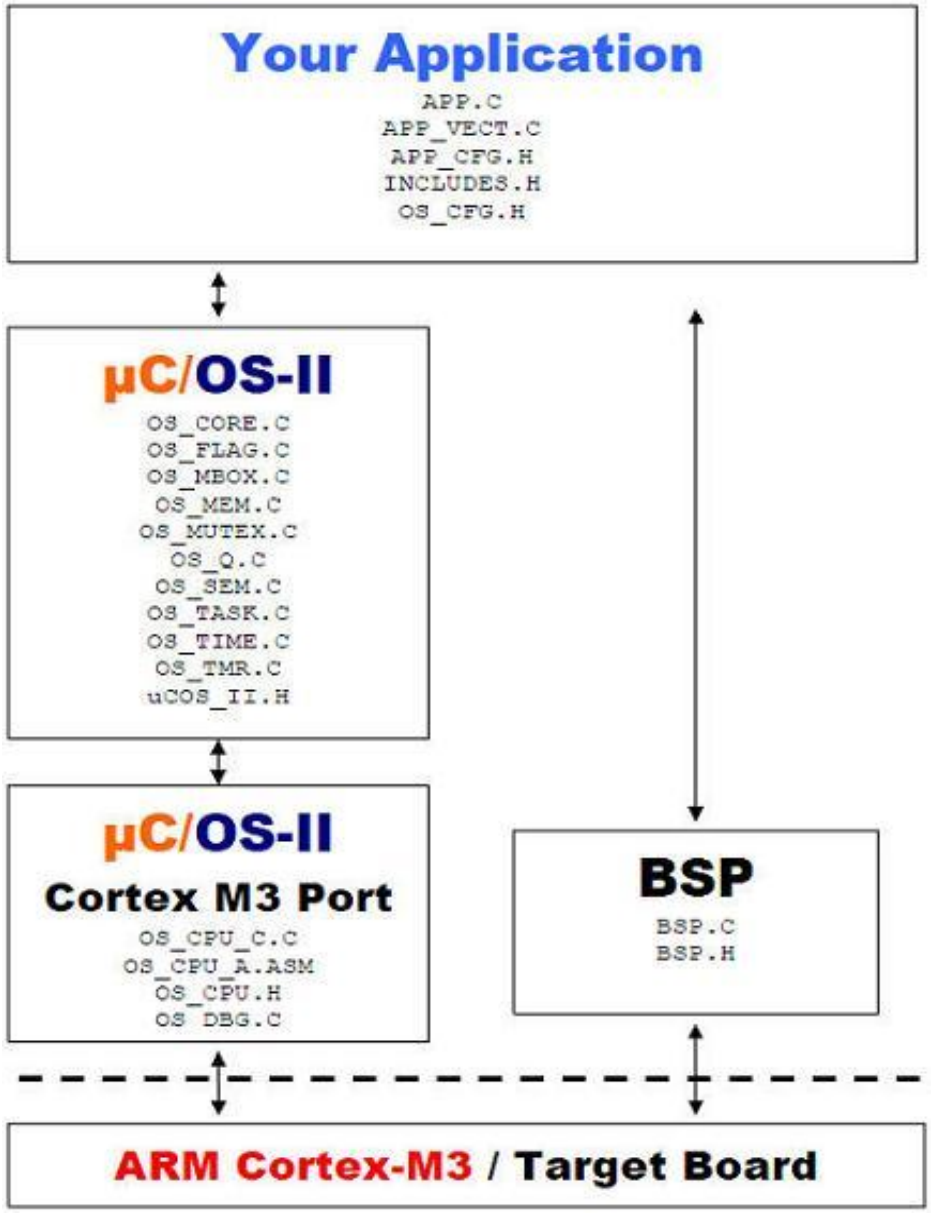
uC/OS-II 소스 다운로드

- 완전한 Free RTOS는 아니다. 양산 제품을 판매할 때 license fee
- 공부 목적으로 사용하는 데 있어서는 아무런 제약이 없다.
- Micrium 홈페이지(http://micrium.com/page/home)
  - ➢ 회원 가입 & 이메일을 통해 인증 절차
- http://micrium.com/page/downloads/ports/st/stm32



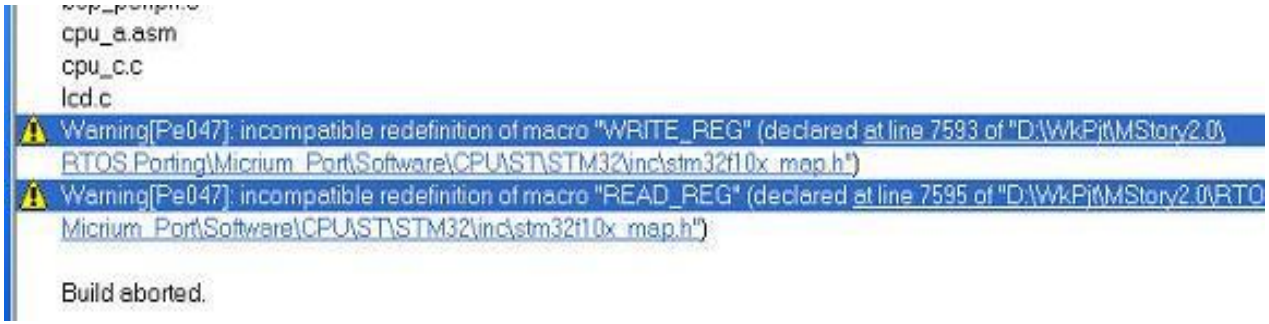
- 위 부분은 EVAL B/E, 아래 것은 EVAL ZE-SK 보드를 위한 것
  - ➢ EVAL ZE-SK 용이 보다 최신이어서 이를 이용했다

uC/OS-II 소스들간 관계



- uC/OS-II의 kernel과 관련된 부분 코드는 수정하지 않는다
- 대부분 BSP에 관련된 부분을 수정할 것이고, application 부분에서 task에 대한 설정 및 동작 부분만 수정 진행

빌드 에러 수정



- WRITE_REG와 READ_REG에 대한 것이 중복 정의
- <lcd.c>
  - #define WRITE_REG 0x02
  - #define READ_REG 0x03
- <stm32f10x_map.h>
  - #define WRITE_REG(REG, VAL) ((REG) = VAL)
  - #define READ_REG(REG) ((REG))
- 두 파일에서 같은 이름의 define을 사용하고 있다.
- <stm32f10x_map.h>
  - #define WRITE_REG**_STM**(REG, VAL) ((REG) = VAL)
  - #define READ_REG**_STM**(REG) ((REG))

main

```
int  main (void) {
    CPU_INT08U  os_err;
    BSP_IntDisAll(); /* Disable all ints until we are ready to accept them.  */
    OSInit();           /* Initialize "uC/OS-II, The Real-Time Kernel".        */
    os_err = OSTaskCreateExt((void (*)(void *)) App_TaskStart,
                (void         * ) 0,
                (OS_STK     * )&App_TaskStartStk[APP_TASK_START_STK_SIZE - 1],
                (INT8U          ) APP_TASK_START_PRIO,
                (INT16U         ) APP_TASK_START_PRIO,
                (OS_STK     * )&App_TaskStartStk[0],
                (INT32U         ) APP_TASK_START_STK_SIZE,
                (void         * )0,
                (INT16U         )(OS_TASK_OPT_STK_CLR | OS_TASK_OPT_STK_CHK));
#if (OS_TASK_NAME_SIZE >= 11)
    OSTaskNameSet(APP_TASK_START_PRIO,
                          (CPU_INT08U *)"Start Task", &os_err);
#endif
    OSStart();  /* Start multitasking (i.e. give control to uC/OS-II).  */
    return (0);
}
```

App_TaskStart

```
static  void  App_TaskStart (void *p_arg)
{
..... ..... ..... ..... .....
   os_err = OSTaskCreateExt((void (*)(void *)) App_LED1_Task,  /* led1 */
         (void         * ) 0,
         (OS_STK       * )&App_Task_LED1_Stack[APP_TASK_START_STK_SIZE - 1],
         (INT8U         ) APP_TASK_LED1_PRIO,
         (INT16U        ) APP_TASK_LED1_PRIO,
         (OS_STK       * )&App_Task_LED1_Stack[0],
         (INT32U        ) APP_TASK_START_STK_SIZE,
         (void         * )0,
         (INT16U        )(OS_TASK_OPT_STK_CLR | OS_TASK_OPT_STK_CHK));
#if (OS_TASK_NAME_SIZE >= 9)
   OSTaskNameSet(APP_TASK_LED1_PRIO, "LED 1", &os_err);
#endif
   OSTimeDlyHMSM(0, 0, 0, 100);

   os_err = OSTaskCreateExt((void (*)(void *)) App_LED2_Task,  /* led2 */
..... ..... ..... ..... .....
}
```

App_LED1_Task

```
static void App_LED1_Task(void *p_arg)
{
    (void)p_arg;

    BSP_LED_On(1);
    OSTimeDlyHMSM(0, 0, 0, 2000);
    BSP_LED_Off(1);

    while (DEF_TRUE) {
        BSP_LED_Toggle(1);
        OSTimeDlyHMSM(0, 0, 0, 100);
    }
}
```

OSTaskCreateExt (1)

```
INT8U  OSTaskCreate   (void (*task)(void *p_arg),
                       void *p_arg,
                       OS_STK *ptos,
                       INT8U prio)
INT8U  OSTaskCreateExt (void    (*task)(void *p_arg),
                        void     *p_arg,
                        OS_STK   *ptos,
                        INT8U     prio,
                        INT16U    id,
                        OS_STK   *pbos,
                        INT32U    stk_size,
                        void     *pext,
                        INT16U    opt)
```

- OSTaskCreate와 OSTaskCreateExt는 근본적으로 같은 행동
- OSTaskCreateExt의 경우가 보다 복잡한 설정 가능
- OSTaskCreate에서 설정하지 않는 부분은 default 값으로 처리

OSTaskCreateExt (2) - return value

```
#define OS_ERR_NONE                  0u
#define OS_ERR_PRIO_EXIST           40u
#define OS_ERR_PRIO_INVALID         42u
#define OS_ERR_TASK_CREATE_ISR      60u
#define OS_ERR_TASK_NO_MORE_TCB     66u
```

- #define OS_LOWEST_PRIO 31
  /* Defines the lowest priority that can be assigned ... */
- os_cfg.h를 보면 위와 같이 OS_LOWEST_PRIO가 31로 정의
- 이 값보다 큰 값을 주어서 task를 만들려고 하면 OS_ERR_PRIO_INVALID 에러 발생
- uCOS의 경우 255개의 task를 만들 수 있다. 하지만 우리가 사용하는 버전은 이 값이 31로 설정되어 있어서 32개의 task만 만들 수 있다.
- define의 내용을 통해서도 알 수 있지만 값이 클수록 우선순위가 낮은 것임을 알 수 있다.

OSTaskCreateExt (3)

- void (*task)(void *p_arg) & void *p_arg
  - task의 주된 함수 부분이 이곳에 포인터로 들어가게 된다.
- OS_STK *ptos & OS_STK *pbos
  - task stack의 pointer 부분 (**Stack Top & Bottom**)
  - #define **OS_STK_GROWTH 1** /* Stack grows from HIGH to LOW memory on ARM */
  - stack의 방향이 아래 쪽으로 커지게 된다는 것을 의미한다. 즉, memory의 높은 곳에서 낮은 곳을 증가한다는 것을 의미한다. 이 경우 ptos에 전해지는 값은 stack에서 가장 높은 메모리 주소값을 넘겨주어야 한다.
  - 반대로 OS_STK_GROWTH 값이 0으로 설정되어 있다면 stack의 증가 방향이 위쪽으로 커지게 될 것이고, 이 경우 가장 낮은 메모리 주소값을 넘겨주어야 한다.
  - static OS_STK App_Task_LED1_Stack[APP_TASK_START_STK_SIZE];
  - OS_STK_GROWTH 값이 1로 설정되어 있기 때문에 ptos에는 (OS_STK *)&App_Task_LED1_Stack[**APP_TASK_START_STK_SIZE - 1**]와 같이 넣어서 호출하였고, pbos에는 (OS_STK *)&App_Task_LED1_Stack[**0**]으로 넣어서 호출하였다.

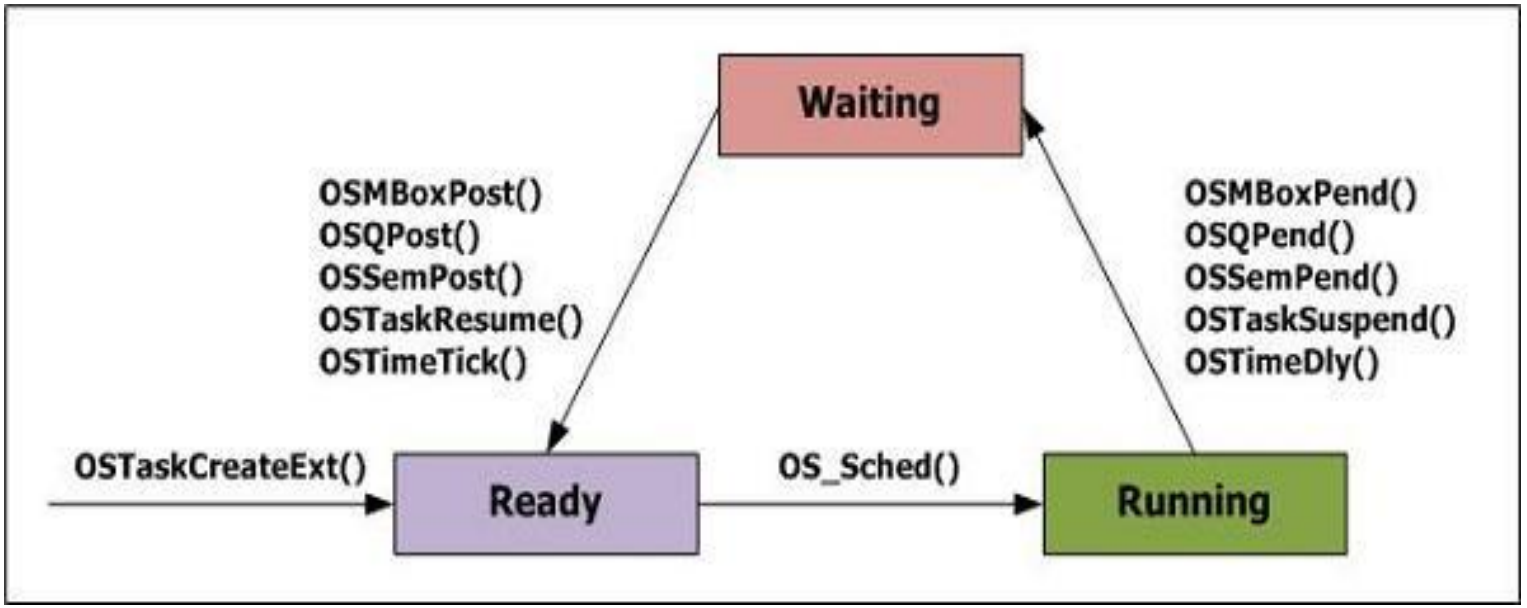
OSTaskCreateExt (4)

- INT8U prio & INT16U id - task의 priority 값과 ID
  - 반드시 고유한 값을 넣어야 한다. 값이 작으면 작을수록 높은 priority
  - ID는 0부터 65535까지 16 비트 공간
  - 다른 값을 가져도 되지만 같은 값으로 사용하였다. 근거는 **priority가 task 마다 다르게 설정되어야 한다**는 uCOS의 특징을 이용하는 것이다. 굳이 ID를 따로 부여할 필요가 없이 이 priority를 ID로 사용할 수 있다.
- INT32U stk_size - task에서 사용할 stack에 대한 크기
  - APP_TASK_START_STK_SIZE 전달 (코드에서 128로 정의)
  - typedef unsigned int OS_STK; /* Each stack entry 32-bit wide */
  - unsigned int로 정의. stack 증가 감소 크기로 **32 비트의 넓이**
  - 크기가 16 비트라면 16 비트의 폭에 맞는 stack의 크기를 기술하여야 한다
- void *pext - OS_TCBInit에 넘겨주는 값
  - TCB extension에서 사용될 사용자가 지정한 메모리 위치
  - 이 메모리에는 context switch 동안에 floating-point registers의 내용을 보관하고 있을 수도 있고, 각 task가 수행하는데 소모한 시간, switch가 일어난 횟수 등도 보관할 수 있다.
  - 우리의 경우는 이 부분을 사용하지 않는다. 0으로 주면 된다.

OSTaskCreateExt (5) - INT16U opt

- 이 값은 task의 행동에 대한 추가적인 정보
  - #define OS_TASK_OPT_NONE 0x0000u /* NO option */
  - #define OS_TASK_OPT_STK_CHK 0x0001u
    /* Enable stack checking for the task */
  - #define OS_TASK_OPT_STK_CLR 0x0002u
    /* Clear the stack when the task is create */
  - #define OS_TASK_OPT_SAVE_FP 0x0004u
    /* Save the contents of any floating-point registers */
- statistic task가 주기적으로 동작을 하게 만들 수 있는데 이 경우 **stack에 대해서 검사를 수행**할 수 있다. **OS_TASK_OPT_STK_CHK**이 정의되어 있을 경우 이 검사를 하게 할 것인가를 옵션으로 구분하는 것
- **OS_TASK_OPT_STK_CLR**는 task가 생성될 때 stack을 clear할 것인가를 정해주는 옵션
- **OS_TASK_OPT_SAVE_FP**는 CPU가 floating-point registers를 가지고 있을 때 이들 값을 context switch 동안 저장을 할 것인지 결정
- OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR와 같이 OR 연산으로 두 개의 옵션만 주어서 task를 생성하였다.

task state



- uC/OS-II의 경우 task 상태에 대한 것은 무척 간단한 편이다.
- 대부분의 경우 어떤 것에 대한 대기 상태가 되는 경우 Waiting이 되고, 그렇지 않을 경우는 Ready 상태에 있다가 scheduling 기능에 따라서 Running 상태로 가게 된다.