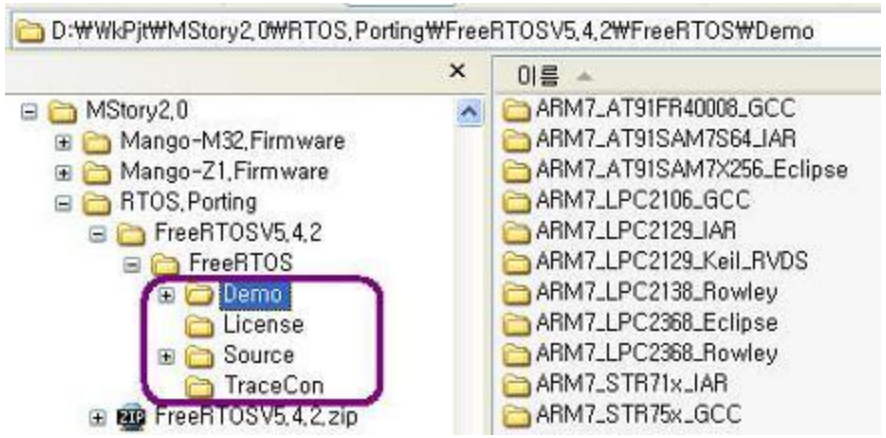
FreeRTOS porting 실습

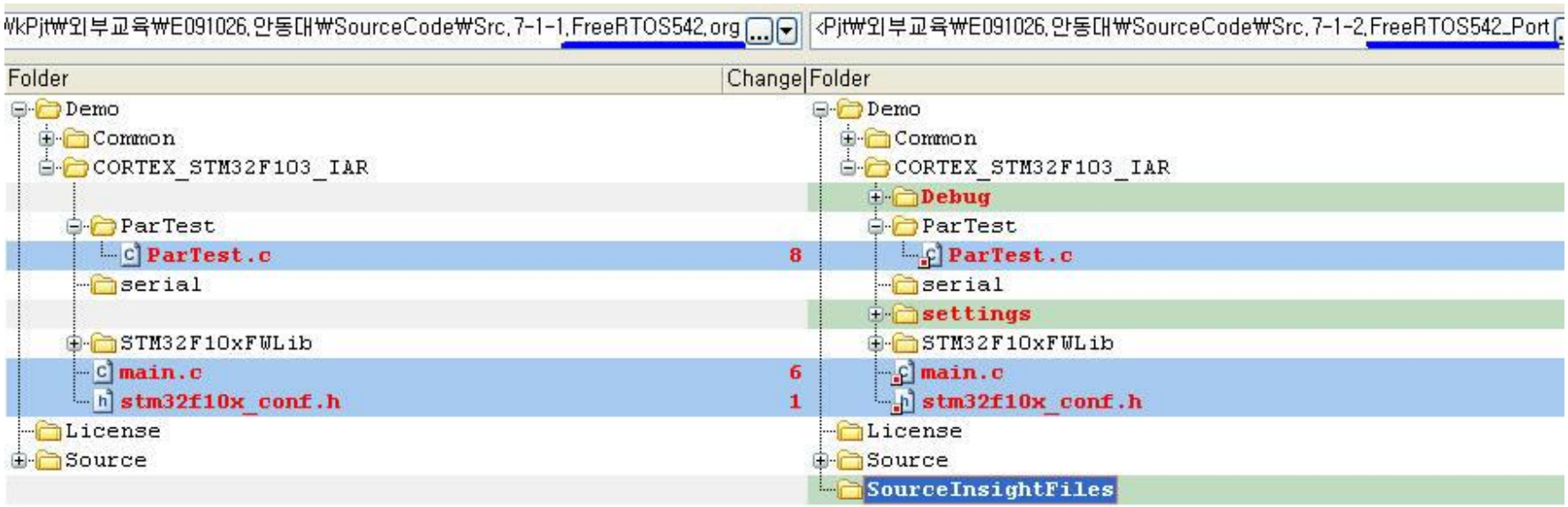
2009.11.20

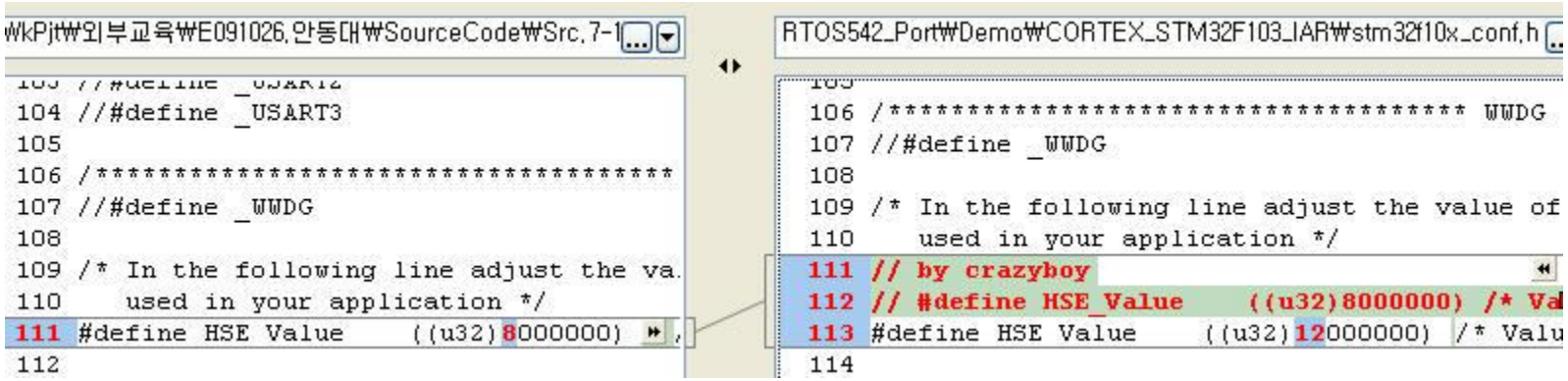
FreeRTOS 소스 다운로드

- http://sourceforge.net/projects/freertos/files/FreeRTOS/

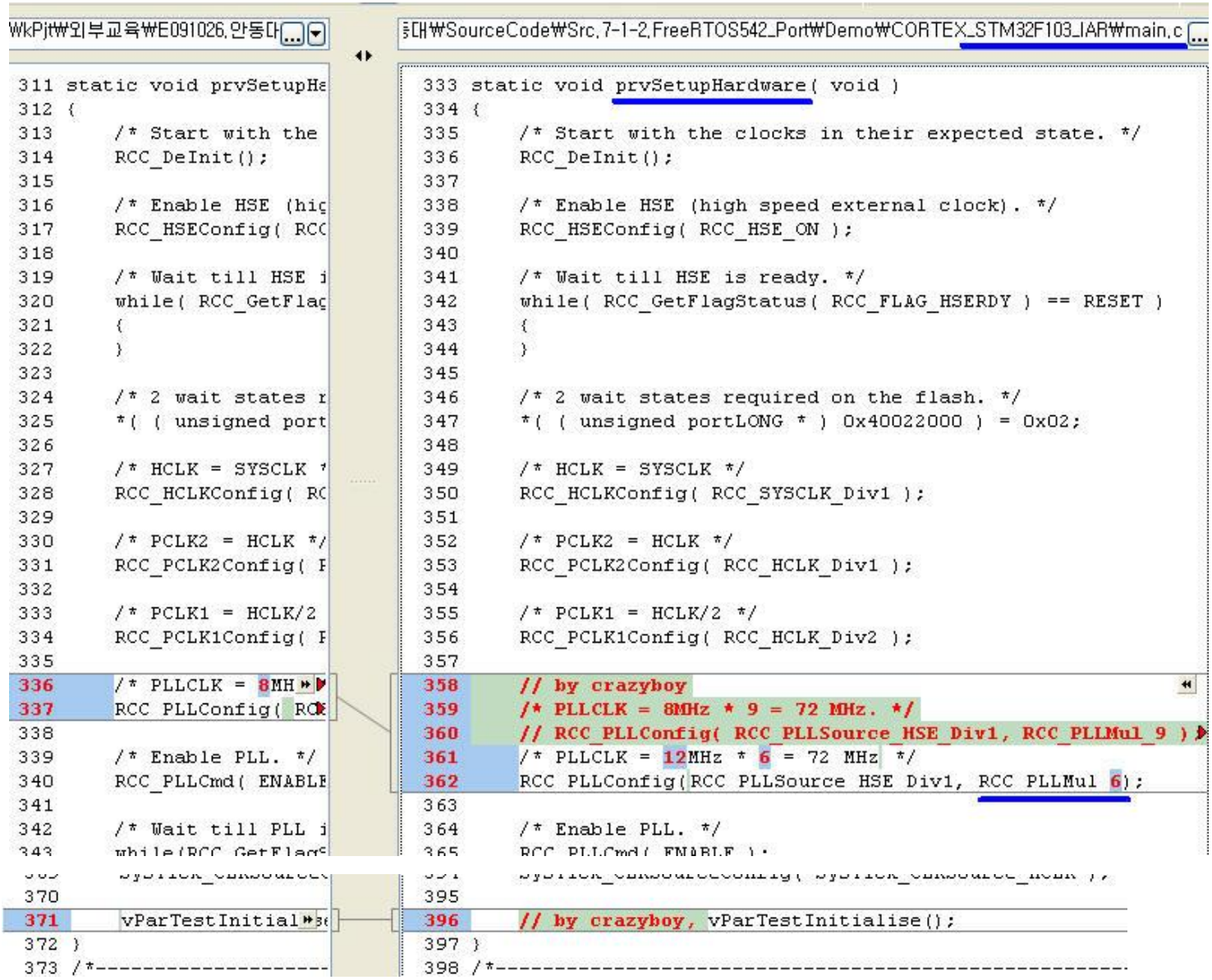


소스 수정 내역 (1)


WkPjt₩외부교육₩E091026,안동대₩SourceCode₩Src,7-1-1,FreeRTOS542.org
<Pjt₩외부교육₩E091026,안동대₩SourceCode₩Src,7-1-2,FreeRTOS542_Port
Folder
Change
Folder
Demo
Common
CORTEX_STM32F103_IAR
ParTest
ParTest.c
8
serial
STM32F10xFWLib
main.c
6
stm32f10x_conf.h
1
License
Source
Debug
settings
SourceInsightFiles


WkPjt₩외부교육₩E091026,안동대₩SourceCode₩Src,7-1
RTOS542_Port₩Demo₩CORTEX_STM32F103_IAR₩stm32f10x_conf.h
104 //#define _USART3
105
106 /**********
107 //#define _WWDG
108
109 /* In the following line adjust the va
110 used in your application */
111 #define HSE_Value ((u32)8000000)
112
106 /********** WWDG
107 //#define _WWDG
108
109 /* In the following line adjust the value of
110 used in your application */
111 // by crazyboy
112 // #define HSE_Value ((u32)8000000) /* Va
113 #define HSE_Value ((u32)12000000) /* Valu
114

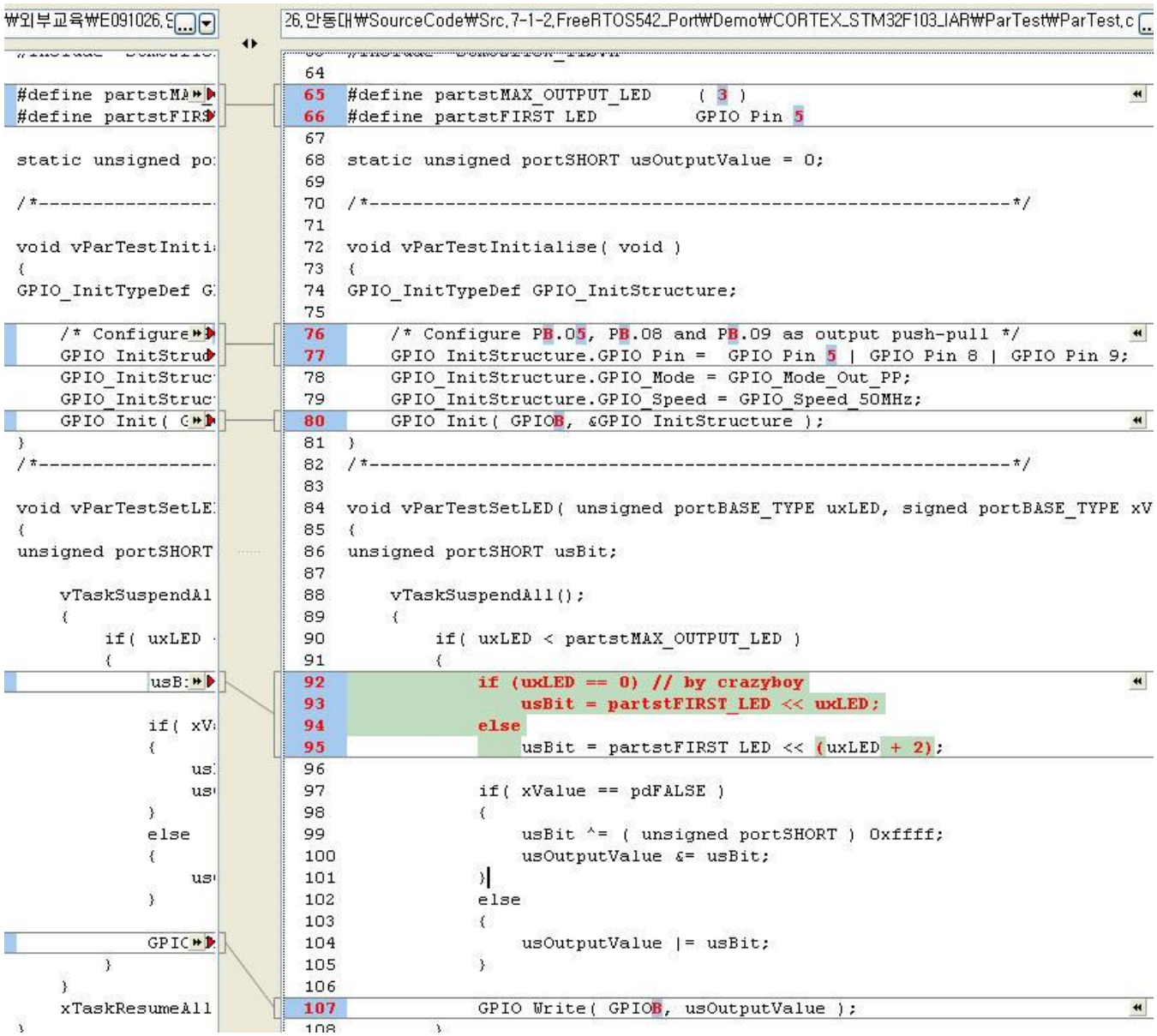
소스 수정 내역 (2)



```
311 static void prvSetupHa
312 {
313     /* Start with the
314     RCC_DeInit();
315
316     /* Enable HSE (hig
317     RCC_HSEConfig( RCC
318
319     /* Wait till HSE i
320     while( RCC_GetFlag
321     {
322     }
323
324     /* 2 wait states r
325     *( ( unsigned port
326
327     /* HCLK = SYSCLK *
328     RCC_HCLKConfig( RC
329
330     /* PCLK2 = HCLK */
331     RCC_PCLK2Config( P
332
333     /* PCLK1 = HCLK/2
334     RCC_PCLK1Config( P
335
336     /* PLLCLK = 8MH
337     RCC_PLLConfig( RC
338
339     /* Enable PLL. */
340     RCC_PLLCmd( ENABLE
341
342     /* Wait till PLL i
343     while(RCC_GetFlag
370
371     vParTestInitial
372 }
373 /*----------
```

```
333 static void prvSetupHardware( void )
334 {
335     /* Start with the clocks in their expected state. */
336     RCC_DeInit();
337
338     /* Enable HSE (high speed external clock). */
339     RCC_HSEConfig( RCC_HSE_ON );
340
341     /* Wait till HSE is ready. */
342     while( RCC_GetFlagStatus( RCC_FLAG_HSERDY ) == RESET )
343     {
344     }
345
346     /* 2 wait states required on the flash. */
347     *( ( unsigned portLONG * ) 0x40022000 ) = 0x02;
348
349     /* HCLK = SYSCLK */
350     RCC_HCLKConfig( RCC_SYSCLK_Div1 );
351
352     /* PCLK2 = HCLK */
353     RCC_PCLK2Config( RCC_HCLK_Div1 );
354
355     /* PCLK1 = HCLK/2 */
356     RCC_PCLK1Config( RCC_HCLK_Div2 );
357
358     // by crazyboy
359     /* PLLCLK = 8MHz * 9 = 72 MHz. */
360     // RCC_PLLConfig( RCC_PLLSource_HSE_Div1, RCC_PLLMul_9 );
361     /* PLLCLK = 12MHz * 6 = 72 MHz */
362     RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_6);
363
364     /* Enable PLL. */
395
396     // by crazyboy, vParTestInitialise();
397 }
398 /*----------
```

소스 수정 내역 (3)



```
64
65  #define partstMAX_OUTPUT_LED     ( 3 )
66  #define partstFIRST_LED          GPIO_Pin_5
67
68  static unsigned portSHORT usOutputValue = 0;
69
70  /*-----------------------------------------------------------*/
71
72  void vParTestInitialise( void )
73  {
74  GPIO_InitTypeDef GPIO_InitStructure;
75
76      /* Configure PB.05, PB.08 and PB.09 as output push-pull */
77      GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_5 | GPIO_Pin_8 | GPIO_Pin_9;
78      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
79      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
80      GPIO_Init( GPIOB, &GPIO_InitStructure );
81  }
82  /*-----------------------------------------------------------*/
83
84  void vParTestSetLED( unsigned portBASE_TYPE uxLED, signed portBASE_TYPE xV
85  {
86  unsigned portSHORT usBit;
87
88      vTaskSuspendAll();
89      {
90          if( uxLED < partstMAX_OUTPUT_LED )
91          {
92              if (uxLED == 0) // by crazyboy
93                  usBit = partstFIRST_LED << uxLED;
94              else
95                  usBit = partstFIRST_LED << (uxLED + 2);
96
97              if( xValue == pdFALSE )
98              {
99                  usBit ^= ( unsigned portSHORT ) 0xffff;
100                 usOutputValue &= usBit;
101             }
102             else
103             {
104                 usOutputValue |= usBit;
105             }
106
107             GPIO_Write( GPIOB, usOutputValue );
```

GPIO 설정

- 망고 M32의 LED는 GPIO Port B #5, #8, #9 이다. 이에 맞도록 GPIO에 대한 설정 작업을 수행해야 한다.
- 기존 ParTest.c는 GPIOC로 되어있는데 GPIOB로 수정 한다.

```
#define partstMAX_OUTPUT_LED ( 3 )    //      LED 3 EA
#define partstFIRST_LED              GPIO_Pin_5    //      첫번째 LED
void vParTestInitialise( void )
{
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_5 | GPIO_Pin_8 | GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init( GPIOB, &GPIO_InitStructure );
}
```

- 기존 소스는 LED가 GPIO 포트에 연속된 핀으로 연결되었다고 가정

```
수정 전,
usBit = partstFIRST_LED << uxLED;

수정 후, PB5 다음이 PB8, shift
if (uxLED == 0)      usBit = partstFIRST_LED << uxLED;
else                 usBit = partstFIRST_LED << (uxLED + 2);
```

vParTestToggleLED 변경

```
void vParTestToggleLED( unsigned portBASE_TYPE uxLED )
{
    unsigned portSHORT usBit;
    vTaskSuspendAll();
    {
        if( uxLED < partstMAX_OUTPUT_LED )
        {
            if (uxLED == 0)
                usBit = partstFIRST_LED << uxLED;
            else
                usBit = partstFIRST_LED << (uxLED + 2);
            if( usOutputValue & usBit )
            {
                usOutputValue &= ~usBit;
            }
            else
            {
                usOutputValue |= usBit;
            }
            GPIO_Write( GPIOB, usOutputValue );
        }
    }
    xTaskResumeAll();
}
```

main 함수

```
int main( void )
{
    prvSetupHardware();      // Clock 초기화
    vParTestInitialise();    // LED GPIO 포트 초기화

    // 세 개의 LED 테스크를 생성함
    vStartLEDFlashTasks( mainFLASH_TASK_PRIORITY );

    /* Start the scheduler. */
    vTaskStartScheduler();
}
```

prvSetupHardware

```
static void prvSetupHardware( void )
{
	RCC_DeInit(); /* Start with the clocks in their expected state. */
	RCC_HSEConfig( RCC_HSE_ON ); /* Enable HSE (high speed external clock). */
	while( RCC_GetFlagStatus( RCC_FLAG_HSERDY ) == RESET ) { }  /* Wait till HSE is ready. */
	*( ( unsigned portLONG * ) 0x40022000 ) = 0x02; /* 2 wait states required on the flash. */

	RCC_HCLKConfig( RCC_SYSCLK_Div1 ); /* HCLK = SYSCLK */
	RCC_PCLK2Config( RCC_HCLK_Div1 ); /* PCLK2 = HCLK */
	RCC_PCLK1Config( RCC_HCLK_Div2 ); /* PCLK1 = HCLK/2 */

	RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_6); /* PLLCLK = 12MHz * 6 = 72 MHz */

	RCC_PLLCmd( ENABLE ); /* Enable PLL. */
	while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) { }     /* Wait till PLL is ready. */

	RCC_SYSCLKConfig( RCC_SYSCLKSource_PLLCLK );  /* Select PLL as system clock source. */
	while( RCC_GetSYSCLKSource() != 0x08 ) { }       /* Wait till PLL is used as system clock source. */

	/* Enable GPIOA, GPIOB, GPIOC, GPIOD, GPIOE and AFIO clocks */
	RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |RCC_APB2Periph_GPIOC
			| RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE | RCC_APB2Periph_AFIO, ENABLE );
	RCC_APB1PeriphClockCmd( RCC_APB1Periph_SPI2, ENABLE ); /* SPI2 Periph clock enable */

	NVIC_SetVectorTable( NVIC_VectTab_FLASH, 0x0 );  /* Set the Vector Table base address at 0x08000000 */
	NVIC_PriorityGroupConfig( NVIC_PriorityGroup_4 );
	SysTick_CLKSourceConfig( SysTick_CLKSource_HCLK );  /* Configure HCLK clock as SysTick clock source. */

	// by crazyboy, vParTestInitialise();
}
```

vStartLEDFlashTasks

```
/* Task function macros as described on the FreeRTOS.org WEB site. */
#define portTASK_FUNCTION_PROTO( vFunction, pvParameters ) void vFunction( void *pvParameters )
#define portTASK_FUNCTION( vFunction, pvParameters ) void vFunction( void *pvParameters )

    /* Variable used by the created tasks to calculate the LED number to use, and
    the rate at which they should flash the LED. */
    static volatile unsigned portBASE_TYPE uxFlashTaskNumber = 0;

    /* The task that is created three times. */
    static portTASK_FUNCTION_PROTO( vLEDFlashTask, pvParameters );

    /*-----------------------------------------------------------*/

    void vStartLEDFlashTasks( unsigned portBASE_TYPE uxPriority )
    {
    signed portBASE_TYPE xLEDTask;

        /* Create the three tasks. */
        for( xLEDTask = 0; xLEDTask < ledNUMBER_OF_LEDS; ++xLEDTask )
        {
            /* Spawn the task. */
            xTaskCreate( vLEDFlashTask, ( signed portCHAR * ) "LEDx", ledSTACK_SIZE,
                         NULL, uxPriority, ( xTaskHandle * ) NULL );
        }
    }
    /*-----------------------------------------------------------*/

    static portTASK_FUNCTION( vLEDFlashTask, pvParameters )
    {
    portTickType xFlashRate, xLastFlashTime;
    unsigned portBASE_TYPE uxLED;

        /* The parameters are not used. */
        ( void ) pvParameters;

        /* Calculate the LED and flash rate. */
        portENTER_CRITICAL();
        {
            /* See which of the eight LED's we should use. */
            uxLED = uxFlashTaskNumber;

            /* Update so the next task uses the next LED. */
            uxFlashTaskNumber++;
        }
        portEXIT_CRITICAL();
```

portTASK_FUNCTION

```
/* Task function macros as described on the FreeRTOS.org WEB site. */
#define portTASK_FUNCTION_PROTO( vFunction, pvParameters ) ₩
        void vFunction( void *pvParameters )
#define portTASK_FUNCTION( vFunction, pvParameters ) ₩
        void vFunction( void *pvParameters )
```

- FreeRTOS\Source\portable\IAR\ARM_CM3\portmacro.h 매크로
- LED task의 main 함수는 vLEDFlashTask

```
/* The task that is created three times. */
static portTASK_FUNCTION_PROTO( vLEDFlashTask, pvParameters );
static portTASK_FUNCTION( vLEDFlashTask, pvParameters )
{
```

```
static vLEDFlashTask ( void * pvParameters );
static vLEDFlashTask ( void * pvParameters )
{
```

- 모든 task의 정의와 task의 main 함수의 이름을 portTASK_FUNCTION_PROTO와 portTASK_FUNCTION으로 통일

xTaskCreate (1)

```
portBASE_TYPE xTaskCreate( pdTASK_CODE pvTaskCode,
                           const portCHAR * const pcName,
                           unsigned portSHORT usStackDepth,
                           void *pvParameters,
                           unsigned portBASE_TYPE uxPriority,
                           xTaskHandle *pvCreatedTask );
```

- portBASE_TYPE - return value

```
#define portBASE_TYPE    long
#define pdPASS   ( 1 )
#define errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY (-1)
```

- portBASE_TYPE은 단순하게 long으로 정의되어 있는 type이다.
- xTaskCreate가 return해주는 값은 위의 2가지 중의 하나. pdPASS가 return되었을 경우는 이상 없이 task 생성 작업을 완수한 것, errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY가 return되면 memory가 부족해서 정상적으로 task 생성 작업을 하지 못했다는 것

xTaskCreate (2)

- pdTASK_CODE pvTaskCode
  - typedef void (*pdTASK_CODE)(void *);
  - 생성하는 task의 주된 동작을 위한 함수 포인터
- const portCHAR * const pcName
  - 생성하는 task의 이름 지정, "LEDx”로 3개 task 모두 같은 이름
  - 이름은 OS 입장에서 task를 구분하는 용도로 사용하는 것은 아니다.
- unsigned portSHORT usStackDepth
  - 생성되는 task의 stack에 대한 크기를 전달해주는 부분이다.
  - #define configMINIMAL_STACK_SIZE ((unsigned portSHORT)128)
  - #define ledSTACK_SIZE configMINIMAL_STACK_SIZE
  - 128 바이트. 우리의 프로그램은 단순히 LED를 하나 깜빡거리는 정도의 일만 수행하기 때문에 충분한 크기이다.
  - 모든 task는 자신의 stack을 따로 관리하게 되고, 이 stack에 대한 설정은 매우 중요한 부분이 되겠다.
- void *pvParameters
  - 생성되는 task에 생성 시점에 parameter로 넘겨주는 데이타를 적어줄 수 있는 부분. 특별히 넘겨줄 데이타가 없는 경우는 NULL로 처리

xTaskCreate (3)

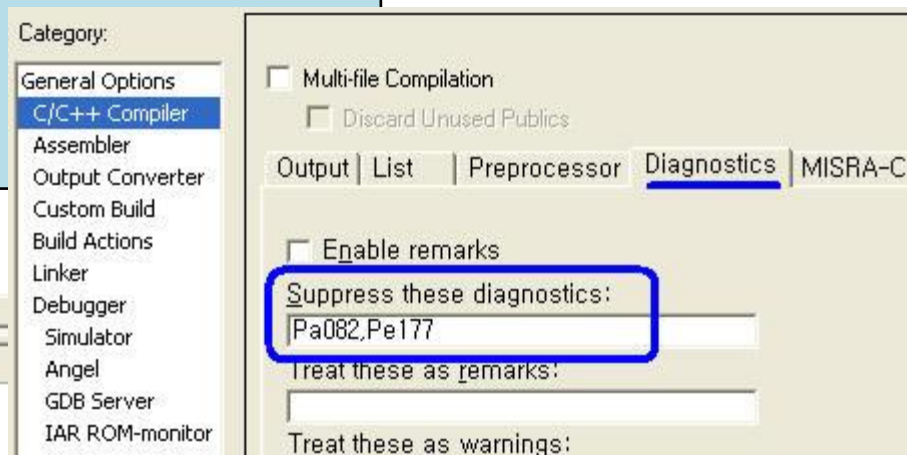
- unsigned portBASE_TYPE uxPriority
  - #define configMAX_PRIORITIES ((unsigned portBASE_TYPE)5)
  - #define tskIDLE_PRIORITY ((unsigned portBASE_TYPE)0)
  - #define mainFLASH_TASK_PRIORITY (tskIDLE_PRIORITY + 1)
  - OS의 가장 중요한 부분 중의 하나는 priority와 관련한 사항
  - 가장 높은 우선순위를 갖는 것은 큰 수치를 갖는 값
  - mainFLASH_TASK_PRIORITY로 같은 우선순위를 갖게 했다.
- xTaskHandle *pvCreatedTask
  - typedef void * xTaskHandle; // **task에 대한 handle 값**
  - 나중에 vTaskDelete 함수를 통해서 만들어진 task를 삭제할때 handle 값으로 넘겨주게 된다.
  - task를 삭제할 필요가 없는 경우라면 handle 값을 넘겨주지 않아도 된다.
  - handle값을 이용해서 task의 priority 값을 설정하거나 읽어올 수도 있고, task를 suspend 시키거나 resume 등의 동작을 수행할 때도 사용 가능
  - 특별히 이에 대한 동작이 필요 없는 단순한 형태이기 때문에 NULL을 주었다
  - vTaskDelete 호출 시 NULL을 주게 되면 현재 태스크를 소멸시킨다

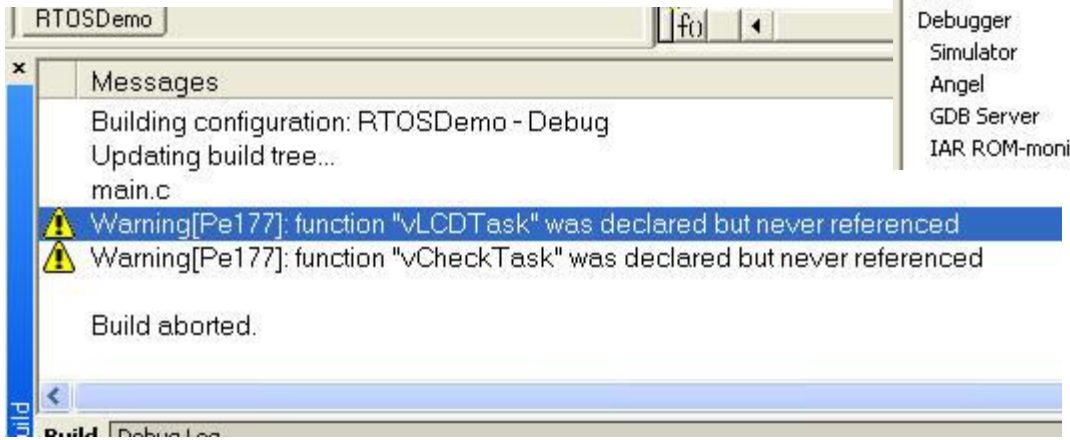
Compiler option

```
comtest.c
cortexm3_macro.s
death.c
Warning[Pa082]: undefined behavior: the order of volatile accesses is undefined in this statement

Build aborted.
```

```
static portTASK_FUNCTION( vSuicidalTask, pvParameters )
{
volatile portLONG l1, l2;
..... ..... ..... ..... ..... .....
                        l1 = 2;
                        l2 = 89;
                        l2 *= l1;
..... ..... ..... ..... ..... .....
```

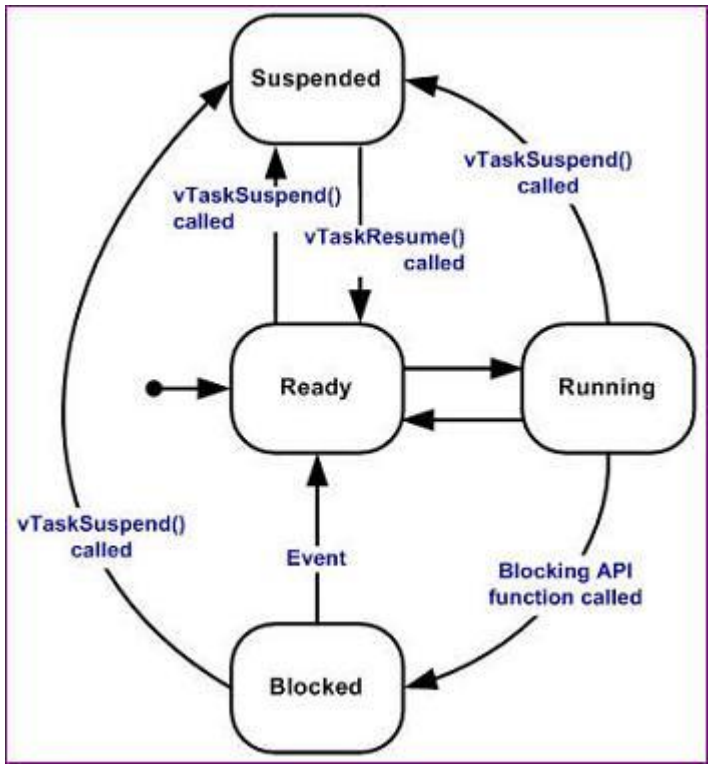




FreeRTOS Overview

- preemptive (선점형) 리소스 점유 방식
  - 먼저 선점하면, 수행이 끝나야 다음 태스크가 수행 된다
  - task를 delay를 주어서 잠시 쉬게 만들 수도 있고, 어떤 event에 대한 waiting 작업 등도 task를 쉬게 만들게 된다.
- Stack overflow 검사 옵션
  - Task stack은 xTaskCreate() API에 의해서 자동으로 할당
  - configCHECK_FOR_STACK_OVERFLOW
- 생성할 수 있는 Task 개수는 소프트웨어적으로 제한이 없다.
- 사용할 수 있는 Priorities 의 개수도 소프트웨어적으로 제한이 없다.
  - 물론 이론적으로 그렇더라도 무제한으로 사용할 수 있는 것은 아니다.
- 우선순위 할당에 있어서 제한이 없다.
  - 같은 우선순위의 Task를 하나 이상 할당이 가능하다.
  - 몇몇 RTOS의 경우 같은 우선순위 할당이 불가능 (uC/OS-II 등)
- 다양한 task간 통신 방법을 가지고 있다.
  - Queues, binary semaphores, counting semaphores, recursive semaphores와 mutex 사용
- 모든 것이 Free다.

task state



- Running – 실제적 수행 상태
  - CPU processor 사용하는 task 상태
- Ready – 실행 가능 준비 상태
  - scheduling에 포함되어 기다리는 상태
- Blocked – event wait 상태
  - vTaskDelay() 호출 시 지정 delay 소모까지 Block 상태가 된다.
  - queue나 semaphore event를 기다리는 동안에도 Blocked 상태
  - 'timeout' period를 가지고 있다.
  - scheduling에 포함되지 않는다.
- Suspended
  - scheduling에 포함되지 않는다.
  - 'timeout' period가 지정되지 않는다.
  - 명시적 API call로 작동
    - vTaskSuspend(): Suspended state 진입
    - xTaskResume(): Ready state 진입