

FreeRTOS Interrupt & Critical Section

- 인터럽트 priority 와 태스크 priority 는 별개의 설정이므로 분리해서 설정한다
- 태스크는 configUSE_PREEMPTION 로 태스크들 사이의 스케줄링 방식을 preemptive 또는 cooperative 로 선택할수 있다. 인터럽트는 항상 preemptive 방식을 사용하며 높은 priority 인터럽트가 낮은 priority 인터럽트를 pre-empt 할수 있다
- 태스크 priority 는 낮은 숫자가 (예, 0 = osPriorityIdle(-3)) 낮은 priority 태스크를 의미하며, 높은 숫자가 (예, 6 = osPriorityRealtime(3)) 높은 priority 태스크를 의미한다
- 인터럽트 priority 는 낮은 숫자가 (예, 0) 높은 priority 인터럽트를 의미하며, 높은 숫자가 (예, 15) 낮은 priority 인터럽트를 의미한다
- FreeRTOS 는 기능은 같고 끝에 FromISR 이름이 붙는 별도의 함수를 제공하는데 인터럽트 핸들러에서는 해당 함수를 사용해야 한다 (예를 들어 xQueueSend() 함수 대신에 xQueueSendFromISR() 함수를 사용한다)
- CMSIS RTOS 는 인터럽트 핸들러 전용 함수가 나뉘어 있지 않다 (함수 내부에 if 문으로 인터럽트 상태인지 아닌지 구분하므로 똑같은 함수 예를 들어 osMessagePut() 를 사용한다)

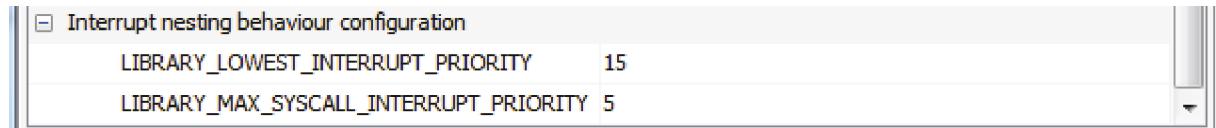
- STM32F4 (Cortex-M4) 는 아래와 같은 5개의 인터럽트 priority 그룹 정책중에 하나를 선택 할 수 있다 (SCB->AIRCR)
 - 0 bit for preemption priority and 4 bits for sub priority
 - 1 bit for preemption priority and 3 bits for sub priority
 - 2 bit for preemption priority and 2 bits for sub priority
 - 3 bit for preemption priority and 1 bits for sub priority
 - 4 bit for preemption priority and 0 bits for sub priority
- 위에 설정한 priority 정책에 맞게 각각의 인터럽트에 preemption priority 와 sub priority 숫자를 설정한다 (NVIC->IPR[x])
- Sub priority 와 무관하게 preemption priority 가 높은 인터럽트가 preemption priority 가 낮은 인터럽트를 pre-empt 할 수 있다 (예를 들어 인터럽트 priority 를 낮게 설정한 UART 인터럽트 핸들러 수행 도중에 priority 를 높게 설정한 USB 인터럽트 핸들러로 pre-empt 할 수 있다)
- Preemption priority 는 동일하고 sub priority 가 다른 인터럽트끼리는 pre-empt 하지 않는다

- Cortex-M4 의 Reset, NMI, Hard fault 인터럽트 등은 priority 가 고정되어 있는 반면, 나머지 Bus fault, Usage fault 등과 같은 system handler 인터럽트들과 MCU 제조사 인터럽트($IRQ \geq 0$) 의 priority 는 펌웨어로 설정할 수 있다

Table 17. Properties of the different exception types

Exception number ⁽¹⁾	IRQ number ⁽¹⁾	Exception type	Priority	Vector address or offset ⁽²⁾	Activation
1	-	Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	Hard fault	-1	0x0000000C	-
4	-12	Memory management fault	Configurable ⁽³⁾	0x00000010	Synchronous
5	-11	Bus fault	Configurable ⁽³⁾	0x00000014	Synchronous when precise Asynchronous when imprecise
6	-10	Usage fault	Configurable ⁽³⁾	0x00000018	Synchronous
7-10	-	-	-	Reserved	-
11	-5	SVCall	Configurable ⁽³⁾	0x0000002C	Synchronous
12-13	-	-	-	Reserved	-
14	-2	PendSV	Configurable ⁽³⁾	0x00000038	Asynchronous
15	-1	SysTick	Configurable ⁽³⁾	0x0000003C	Asynchronous
16 and above	0 and above	Interrupt (IRQ)	Configurable ⁽⁴⁾	0x00000040 and above ⁽⁵⁾	Asynchronous

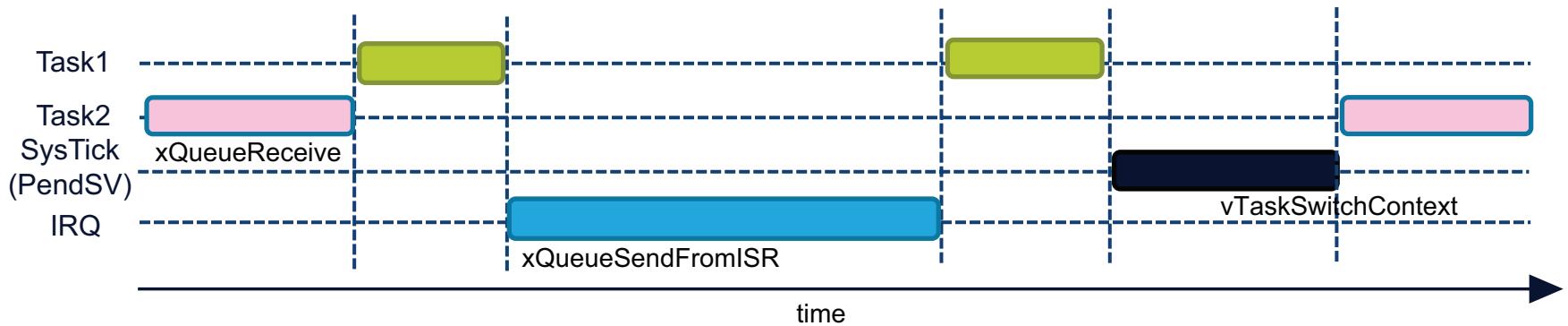
- FreeRTOS 는 인터럽트 priority 와 관련해서 두가지 상수 설정을 할 수 있으며 CubeMX 에서는 아래 설정에 해당한다



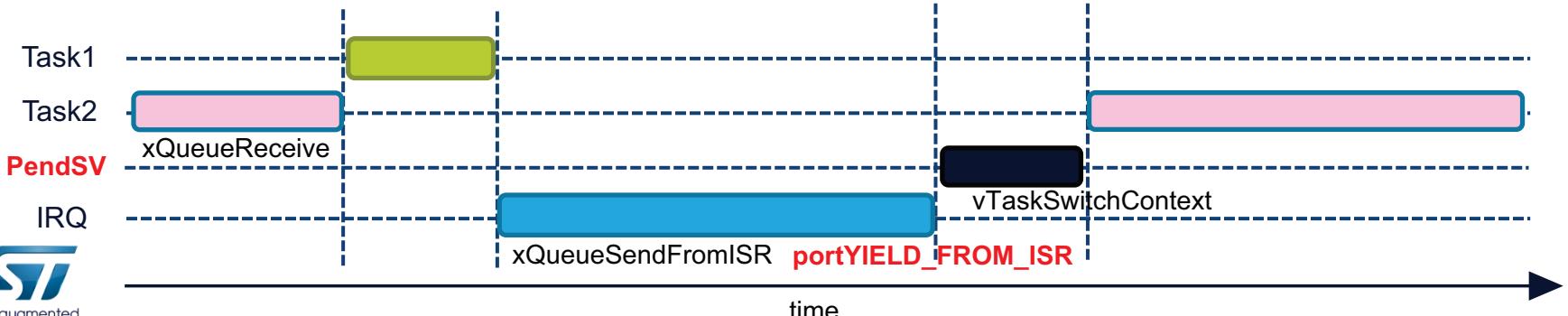
- configKERNEL_INTERRUPT_PRIORITY
 - CubeMX 에서 LIBRARY_LOWEST_INTERRUPT_PRIORITY 의 값 (Cortex-M4 디폴트 15)
 - FreeRTOS 커널 자체 (스케줄러) 의 인터럽트 priority 로서 Systick 타이머 인터럽트와 PendSV 인터럽트의 priority 값으로 설정된다 (SHPR3->PRI_14, SHPR3->PRI_15)
- configMAX_SYSCALL_INTERRUPT_PRIORITY
 - CubeMX 에서 LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 의 값 (Cortex-M4 디폴트 5)
 - FreeRTOS 의 API 함수를 호출하지 않는 인터럽트는 해당 상수값과 무관하게 어떠한 priority 숫자를 설정할 수 있다
 - FreeRTOS 의 API 함수를 호출하는 인터럽트는 해당 상수값보다 같거나 낮은 priority 를 설정해야 한다 (Cortex-M4 의 인터럽트 priority 숫자는 반대이기 때문에 예를 들어 priority 가 낮은 UART 인터럽트를 디폴트 5보다 큰 6 으로 설정)
 - 해당 상수보다 낮은 priority 인터럽트는 critical section 에서 disable (masking) 된다

- portYIELD_FROM_ISR (xHigherPriorityTaskWoken)

- 인터럽트 핸들러 처리 이후 태스크 스케줄링을 바로 해야되는 경우 해당 함수를 호출한다
- 예를 들어 인터럽트 핸들러에서 portYIELD_FROM_ISR 을 사용하지 않을 경우, Task2 가 xQueueReceive 로 blocked 상태로 되고나서 Task1 이 처리되는 도중 IRQ 인터럽트가 발생하면, 인터럽트 핸들러 내부에서 xQueueSendFromISR 로 Task2 를 unblock 시키는 경우 Task1 로 복귀후 다음번 Systick 으로 인한 태스크 스위칭시 Task2 로 전환된다



- 인터럽트 핸들러에서 xHigherPriorityTaskWoken 가 TRUE 이고 portYIELD_FROM_ISR 를 사용하면 아래와 같이 PendSV 핸들러가 바로 호출되어서 Task2 를 빨리 처리할수 있다



- 인터럽트 핸들러는 최대한 짧은 시간 동안 peripheral 과 time critical 한 사용자 처리만 하고 실제 데이터 전후 처리는 태스크로 위임하는게 바람직하다

```
QueueHandle_t xBinarySem;

void one_sec_interrupt(void)
{
BaseType_t higher_task_waken=pdFALSE;

    xSemaphoreGiveFromISR(
        xBinarySem,
        &higher_task_waken);

    portYIELD_FROM_ISR(higher_task_waken);
}
```

```
void Task1(void const * argument)
{
    for(;;){
        if(xSemaphoreTake(xBinarySem,2000)==pdPASS){
            //do something user logic
        }
        vTaskDelay(1000);
    }
}

void Task2(void const * argument)
{
    for(;;){
        vTaskDelay(100);
    }
}

void main(void)
{
...
    xBinarySem = xSemaphoreCreateBinary();
    xTaskCreate(Task1,"Task1", 1000, NULL, 1, NULL);
    xTaskCreate(Task2,"Task2", 1000, NULL, 2, NULL);
...
}
```

Critical Section

8

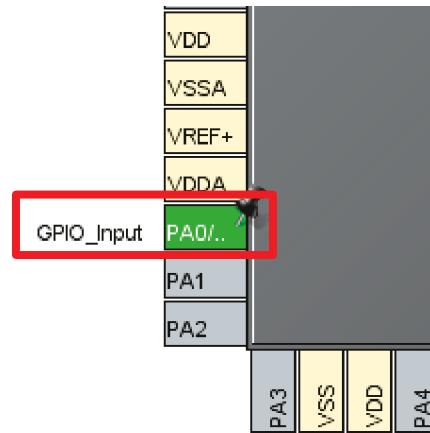
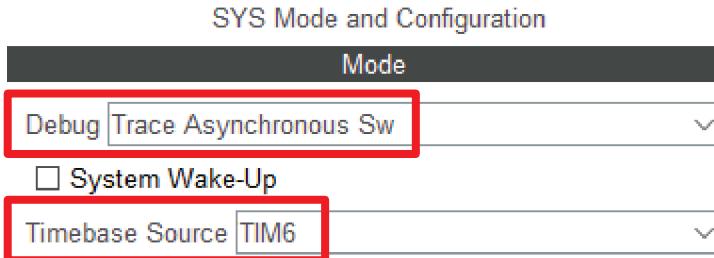
- FreeRTOS 는 전역 변수나 코드와 같은 공유 자원을 보호하기 위해 2가지 방법을 사용한다
- 인터럽트 끄기
 - taskENTER_CRITICAL() 함수와 taskEXIT_CRITICAL() 함수 사이에 공유 자원을 위치 시킨다
 - STM32L4 (Cortex-M4) 포팅은 taskENTER_CRITICAL() 함수의 구현을 configMAX_SYSCALL_INTERRUPT_PRIORITY 보다 priority 가 낮은 인터럽트를 BASEPRI 레지스터를 통해서 masking 한다
 - configMAX_SYSCALL_INTERRUPT_PRIORITY 보다 priority 가 높은 인터럽트로부터는 공유 자원을 보호할수 없다
- 스케줄러 끄기
 - vTaskSuspendAll() 함수와 xTaskResumeAll() 함수 사이에 공유 자원을 위치 시킨다
 - 다른 task 에서 해당 task 를 pre-empt 하는 것을 막을수는 있으나 인터럽트에서 pre-empt 하는 것은 막을수 없다

Interrupt & Critical Section 실습

- STM32F429I-DISC1 보드에 SWO 핀과 PB3 핀이 케이블로 연결되어 있는지 확인
- CubeMX 를 실행하고 새로운 프로젝트를 생성한다
- STM32F429ZITx 선택
- FREERTOS -> Interface -> CMSIS_V1 선택



- SYS -> Debug -> Trace Asynchronous SW 선택
- SYS -> Timebase Source -> TIM6 선택
- PA0 번핀 (USER 버튼) 을 입력핀 (GPIO_Input) 으로 설정



Interrupt & Critical Section 실습

- TIM2, TIM5 (32bit 타이머) 활성화

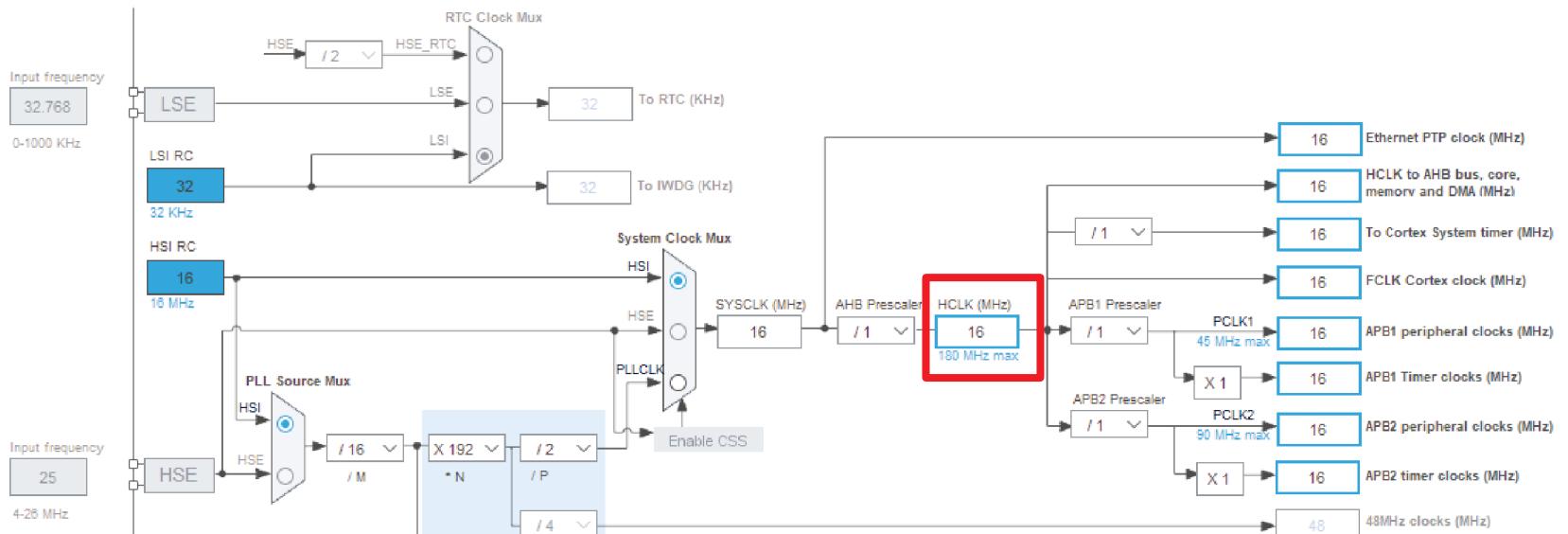
TIM2 Mode and Configuration

Mode
Slave Mode Disable
Trigger Source Disable
Clock Source Internal Clock
Channel1 Disable
Channel2 Disable
Channel3 Disable
Channel4 Disable

TIM5 Mode and Configuration

Mode
Slave Mode Disable
Trigger Source Disable
Internal Clock
Channel1 Disable
Channel2 Disable
Channel3 Disable
Channel4 Disable

- Clock configuration -> HCLK 를 16 MHz 로 설정 (디폴트 설정 그대로)



Interrupt & Critical Section 실습

- TIM2 와 TIM5 를 1초 인터럽트 주기로 설정
- TIM2 와 TIM5 은 APB1 (PCLK1) 을 클럭 소스로 사용하므로 앞에서 설정한 16 MHz 를 16 분주 (Prescaler+1) 해서 1 MHz 를 만든후 up 또는 down 카운터로 reload 값 1,000,000 을 설정한다

User Constants NVIC Settings DMA Settings Parameter Settings

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

Prescaler (PSC - 16 bits value)	15
Counter Mode	Up
Counter Period (AutoReload ...)	1000000
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable

Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit)	Disable (Trigger input effect)
Trigger Event Selection	Reset (UG bit from TIMx_E

Section 28.5.10: SPI register map on page 920

Section 21.4.5: IWDG register map on page 712

Section 22.6.4: WWDG register map on page 719

Section 26.6.21: RTC register map on page 836

Section 19.5.12: TIM10/11/13/14 register map on page 694

Section 19.4.13: TIM9/12 register map on page 684

Section 20.4.9: TIM6 and TIM7 register map on page 707

Section 18.4.21: TIMx register map on page 648

0x4000 3800 - 0x4000 3BFF	SPI2 / I2S2	APB1
0x4000 3400 - 0x4000 37FF	I2S2ext	
0x4000 3000 - 0x4000 33FF	IWDG	
0x4000 2C00 - 0x4000 2FFF	WWDG	
0x4000 2800 - 0x4000 2BFF	RTC & BKP Registers	
0x4000 2000 - 0x4000 23FF	TIM14	
0x4000 1C00 - 0x4000 1FFF	TIM13	
0x4000 1800 - 0x4000 1BFF	TIM12	
0x4000 1400 - 0x4000 17FF	TIM7	
0x4000 1000 - 0x4000 13FF	TIM6	
0x4000 0C00 - 0x4000 0FFF	TIM5	
0x4000 0800 - 0x4000 0BFF	TIM4	
0x4000 0400 - 0x4000 07FF	TIM3	
0x4000 0000 - 0x4000 03FF	TIM2	

Interrupt & Critical Section 실습

- NVIC에서 TIM2와 TIM5의 인터럽트를 enable 한다
- CubeMX 툴에서는 FreeRTOS를 enable하면 preemption priority를 최대로 사용하는 4 bit for preemption priority, 0 bit for subpriority로 고정된다
- Uses FreeRTOS functions 체크박스에 따라서 TIM2, TIM5에 설정할수 있는 Preemptive Priority 범위가 configMAX_SYSCALL_INTERRUPT_PRIORITY 이상 또는 미만으로 바뀌는 것을 확인할 수 있다
- TIM2와 TIM5를 Uses FreeRTOS functions를 체크하고 Priority를 5로 설정한다

NVIC Code generation

Priority Group	4 bits for pre-emption priority 0 bits for subpriority	<input type="checkbox"/> Sort by Preemption Priority and		
Search	Search (Ctrl+F)	<input type="checkbox"/> Show only enabled interrupts <input checked="" type="checkbox"/> Force DMA channels Interrupts		
NVIC Interrupt Table	Enabled	Preemption ...	Sub Priority	Uses FreeRTOS functions
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Memory management fault	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Pre-fetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Debug monitor	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Pendable request for system service	<input checked="" type="checkbox"/>	15	0	<input checked="" type="checkbox"/>
System tick timer	<input checked="" type="checkbox"/>	15	0	<input checked="" type="checkbox"/>
PVD interrupt through EXTI line 16	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
Flash global interrupt	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
RCC global interrupt	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
TIM2 global interrupt	<input checked="" type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
TIM5 global interrupt	<input checked="" type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
Time base: TIM6 global interrupt, DAC1 and DAC...	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
FPU global interrupt	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>

Interrupt & Critical Section 실습

- FREERTOS에서 lowest priority 와 max syscall priority 를 확인할수 있다
- Project Manager에서 Project 이름 임의로 설정, Toolchain 은 TrueSTUDIO 로 설정한다
- Generate Code 후 프로젝트를 연다

Timers and Semaphores Mutexes FreeRTOS Heap Usage

Include parameters User Constants Tasks and Queues

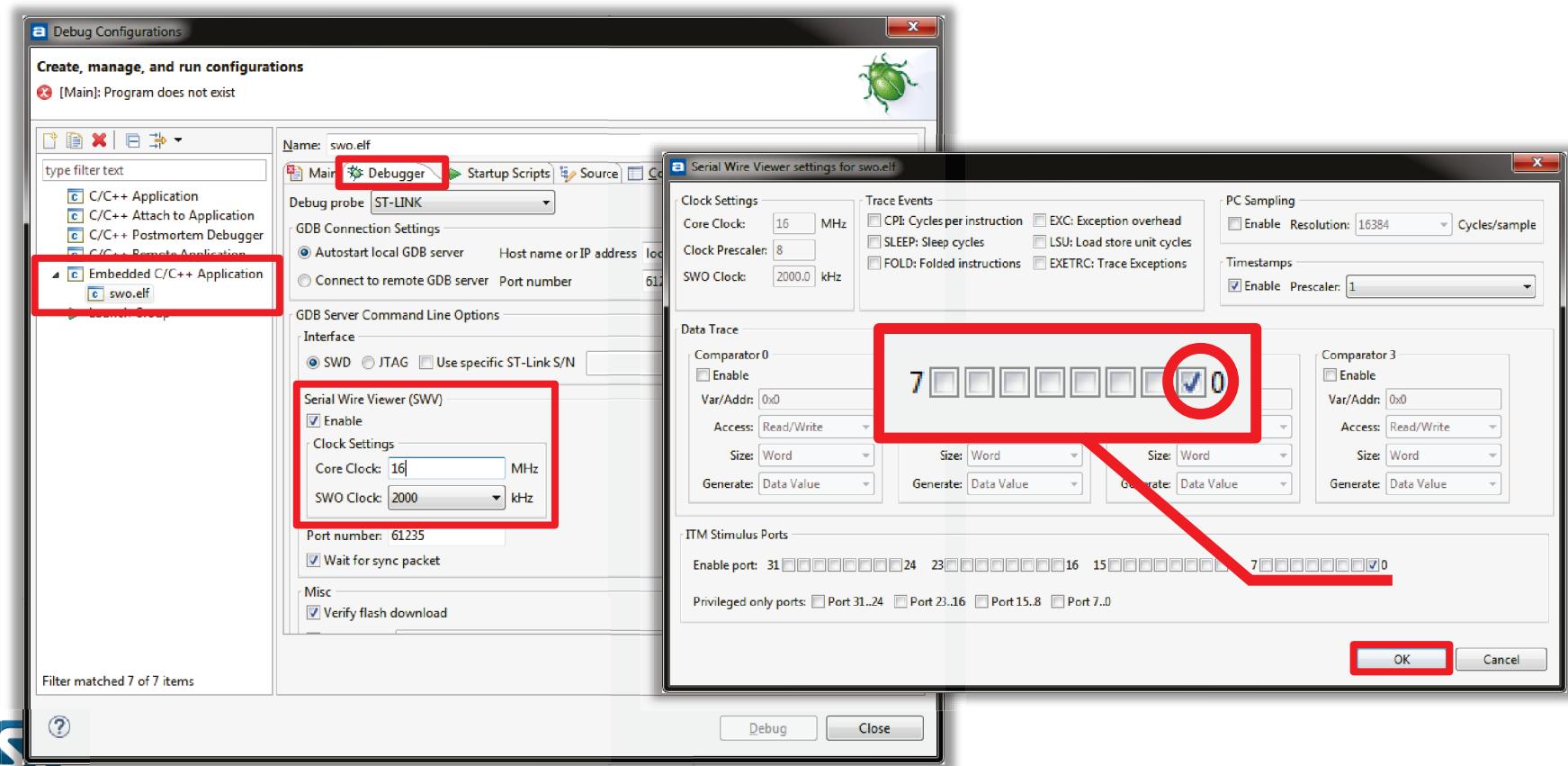
Config parameters

Configure the below parameters :

<input type="text" value="Search (Ctrl+F)"/> CHECK_FOR_STACK_OVERFLOW Disabled				
Run time and task stats gathering related definitions <ul style="list-style-type: none"> GENERATE_RUN_TIME_STATS Disabled USE_TRACE_FACILITY Disabled USE_STATS_FORMATTING_FUNCTIONS Disabled 				
Co-routine related definitions <ul style="list-style-type: none"> USE_CO_Routines Disabled MAX_CO_ROUTINE_PRIORITIES 2 				
Software timer definitions <ul style="list-style-type: none"> USE_TIMERS Disabled 				
Interrupt nesting behaviour configuration <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">LIBRARY_LOWEST_INTERRUPT_PRIORITY</td> <td style="padding: 2px;">15</td> </tr> <tr> <td style="padding: 2px;">LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY</td> <td style="padding: 2px;">5</td> </tr> </table>	LIBRARY_LOWEST_INTERRUPT_PRIORITY	15	LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY	5
LIBRARY_LOWEST_INTERRUPT_PRIORITY	15			
LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY	5			

Interrupt & Critical Section 실습

- 디버깅 편의를 위해서 Project properties -> C/C++ Build Settings -> Tool Settings -> C Compiler -> Optimization level 을 None (-O0) 으로 낮춘다
- printf 출력을 SWO 로 redirect 하는 SWV trace (ITM) 설정을 한다 (오션 세션과 동일)



Interrupt & Critical Section 실습

- ITM (SWO) 디버그 프린트 함수를 하나 추가한다

```
/* USER CODE BEGIN 0 */
#include <stdio.h>
char buff[5][64];
void itm_print(char *p, int len)
{
    for(int i=0;i<len;i++){
        ITM_SendChar(*p++);
    }
}
/* USER CODE END 0 */
```

- CubeMX 는 타이머 설정에 대한 코드를 자동 생성해 주지만 실제 시작 함수는 사용자가 원하는 시점과 장소에 위치 시켜야 한다

```
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim2);
HAL_TIM_Base_Start_IT(&htim5);
/* USER CODE END 2 */
```

Interrupt & Critical Section 실습

16

- 타이머2, 타이머5의 인터럽트 핸들러가 수행되는 1초마다 앞에서 추가한 itm_print 함수를 호출한다

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
/* USER CODE BEGIN Callback 0 */

/* USER CODE END Callback 0 */
    if (htim->Instance == TIM6) {
        HAL_IncTick();
    }
/* USER CODE BEGIN Callback 1 */
    if (htim->Instance == TIM2) {
        sprintf(buff[0], "%s", "tim2\n");
        itm_print(buff[0], strlen(buff[0]));
    }
    if (htim->Instance == TIM5) {
        sprintf(buff[1], "%s", "tim5\n");
        itm_print(buff[1], strlen(buff[1]));
    }
/* USER CODE END Callback 1 */
}
```

Interrupt & Critical Section 실습

17

- CubeMX로 생성된 default 태스크에 아래와 같은 코드를 추가하고 PA0 (USER 버튼)을 누르면 SWV Console 출력이 어떻게 바뀌는지 확인한다
- 태스크와 인터럽트의 itm_print 호출 타이밍이 겹치게 되면 출력 글자가 밀리는 현상이 발생할수 있다

```
void StartDefaultTask(void const * argument)
{
    /* USER CODE BEGIN 5 */
    /* Infinite loop */
    for(;;)
    {
        //taskENTER_CRITICAL();
        while(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0)==GPIO_PIN_SET){};
        //taskEXIT_CRITICAL();
        sprintf(buff[2], "%s", "default task\n");
        itm_print(buff[2], strlen(buff[2]));
        osDelay(1000);
    }
    /* USER CODE END 5 */
}
```

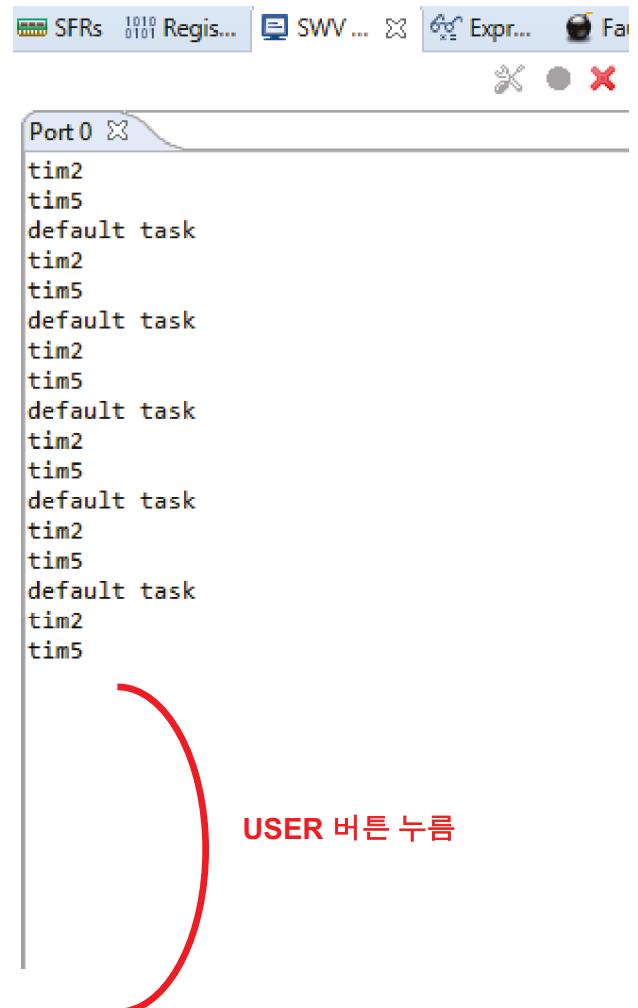


Interrupt & Critical Section 실습

18

- 아래 코드에서 taskENTER_CRITICAL() 과 taskEXIT_CRITICAL() 의 주석을 해제하고 PA0 (USER 버튼) 을 누르면 SWV Console 출력이 어떻게 바뀌는지 확인한다

```
void StartDefaultTask(void const * argument)
{
    /* USER CODE BEGIN 5 */
    /* Infinite loop */
    for(;;)
    {
        taskENTER_CRITICAL();
        while(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0)==SET){};
        taskEXIT_CRITICAL();
        sprintf(buff[2], "%s", "default task\n");
        itm_print(buff[2], strlen(buff[2]));
        osDelay(1000);
    }
    /* USER CODE END 5 */
}
```



Interrupt & Critical Section 실습

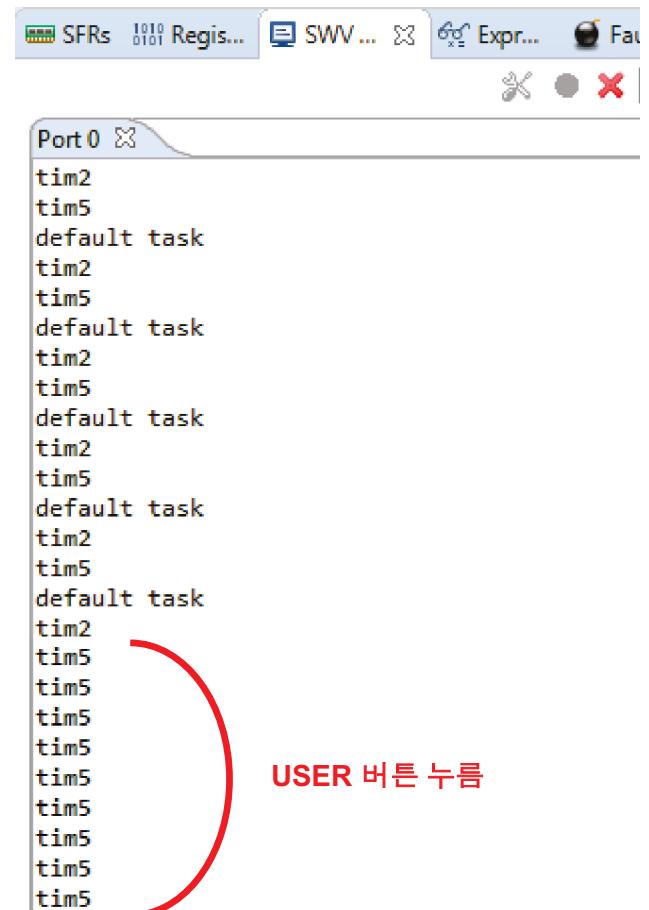
- HAL_TIM_Base_MspInit 함수에서 타이머 5의 인터럽트 priority 를 하나 작은 값(높은 인터럽트 priority) 으로 변경하고 위의 테스트를 다시 수행했을때 PA0 (USER 버튼) 을 누르면 SWV Console 출력이 어떻게 바뀌는지 확인한다

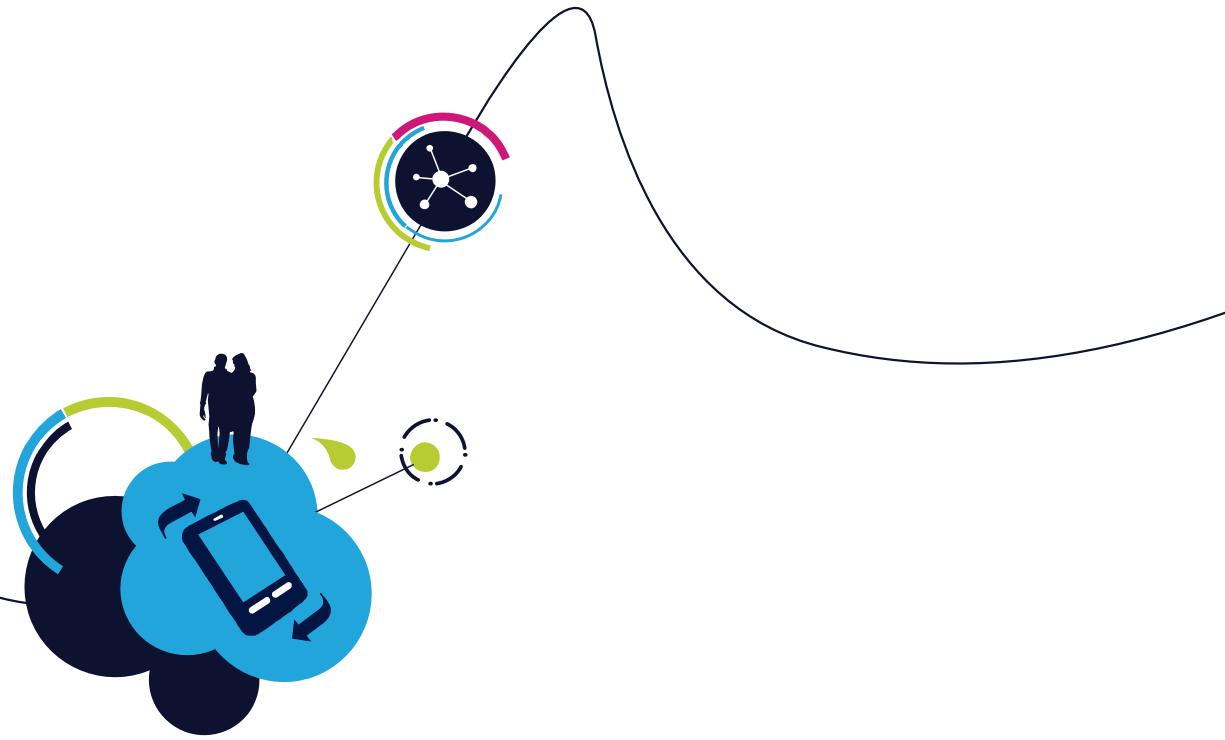
```
else if(htim_base->Instance==TIM5)
{
    /* USER CODE BEGIN TIM5_MspInit 0 */

    /* USER CODE END TIM5_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_TIM5_CLK_ENABLE();
    /* Peripheral interrupt init */

    //HAL_NVIC_SetPriority(TIM5_IRQn, 5, 0);
    HAL_NVIC_SetPriority(TIM5_IRQn, 4, 0);
    HAL_NVIC_EnableIRQ(TIM5_IRQn);
    /* USER CODE BEGIN TIM5_MspInit 1 */

    /* USER CODE END TIM5_MspInit 1 */
}
```





FreeRTOS Event Group

Event Group

21

- Semaphore 나 Queue 와 다르게 하나 또는 여러개의 이벤트를 조합해서 전달하고 기다리는 동기화 방법으로 사용된다
- configUSE_16_BIT_TICKS 가 1 이면 Event Group (EventGroupHandle_t) 은 16 비트 크기의 데이터형이며 하위 8 비트만 Event Flag 로 사용된다
- configUSE_16_BIT_TICKS 가 0 이면 Event Group (EventGroupHandle_t) 은 32 비트 크기의 데이터형이며 하위 24 비트만 Event Flag 로 사용된다
- Event Group 내의 한 비트를 Event Flag 라고 부르며 1 또는 0 으로 이벤트 발생 여부를 나타낸다
- 인터럽트 핸들러에서 사용하는 xEventGroupSetBitsFromISR 와 xEventGroupClearBitsFromISR 함수는 Set 또는 Clear 할 Bit (List_t) 의 수가 가변이기 때문에 인터럽트 핸들러에서 non-deterministic 딜레이가 발생하지 않도록 flag 세팅만 하고 실제 동작은 타이머 태스크로 위임하도록 구현되어 있다. 해당 API 를 사용하려면 configUSE_TIMERS, configUSE_TRACE_FACILITY , INCLUDE_xTimerPendFunctionCall 도 같이 세팅되어야 한다

- 제공되는 API 는 아래와 같다

Event Groups and Event Bits API Functions

- `xEventGroupCreate`
- `xEventGroupCreateStatic`
- `xEventGroupWaitBits`
- `xEventGroupSetBits`
- `xEventGroupSetBitsFromISR`
- `xEventGroupClearBits`
- `xEventGroupClearBitsFromISR`
- `xEventGroupGetBits`
- `xEventGroupGetBitsFromISR`
- `xEventGroupSync`
- `vEventGroupDelete`

Event Group

23

- 아래는 Task1, Task2에서 Event Flag 1과 2를 각각 세팅하고 Task3은 Event Flag 1과 2가 모두 함께 세팅되는 것을 2초 동안 기다리는 예제이다

```
EventGroupHandle_t evtGrp;
uint32_t evt_flag_1 = 0x01; //bit 0
uint32_t evt_flag_2 = 0x04; //bit 2

void Task1(void const * argument)
{
    for(;;){
        xEventGroupSetBits(evtGrp, evt_flag_1);
        vTaskDelay(1000);
    }
}

void Task2(void const * argument)
{
    for(;;){
        xEventGroupSetBits(evtGrp, evt_flag_2);
        vTaskDelay(1000);
    }
}
```

```
void Task3(void const * argument)
{
    uint32_t result;
    for(;;){
        result = xEventGroupWaitBits(
            evtGrp,
            (evt_flag_1 | evt_flag_2),
            pdTRUE, //xClearOnExit
            pdTRUE, //xWaitForAllBits
            2000); //xTicksToWait
        if(result & (evt_flag_1 | evt_flag_2) ==
           (evt_flag_1 | evt_flag_2)){
            //flag_1 and flag_2 are all set
        }
        else{
            if(result & evt_flag_1){//flag_1 is set}
            else if(result & evt_flag_2){//flag_2 is set}
            else { //none set}
        }
    }
}

void main(void)
{
    evtGrp = xEventGroupCreate();
}
```

Event Group 실습

24

- 앞장에서 사용한 CubeMX 와 코드를 그대로 사용하고 여기에 태스크만 2개 더 추가해서 코드 생성을 다시 한다
- 단순 테스트를 위해 태스크는 디폴트 설정을 사용한다

Reset Configuration

Timers and Semaphores Mutexes FreeRTOS Heap Usage
Config parameters Include parameters User Constants Tasks and Queues

Tasks

Task Name	Priority	Sta...	Entry Function	Code ...	Para...	Allocation	Buffer ...	Control Bl...
defaultTask	osPriorityNormal	128	StartDefaultTask	Default	NULL	Dynamic	NULL	NULL
myTask02	osPriorityIdle	128	StartTask02	Default	NULL	Dynamic	NULL	NULL
myTask03	osPriorityIdle	128	StartTask03	Default	NULL	Dynamic	NULL	NULL

Event Group 실습

25

- Event Group, Event Flag 를 선언한다

```
/* USER CODE BEGIN PV */  
/* Private variables ----- */  
EventGroupHandle_t evtGrp;  
uint32_t evt_flag_1 = 0x01; //bit 0  
uint32_t evt_flag_2 = 0x02; //bit 1  
/* USER CODE END PV */
```

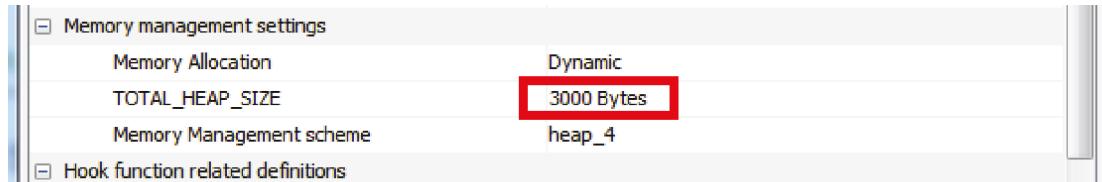
- 앞장 예제 코드에서 타이머 5 시작을 주석처리 하고 Event Group 을 생성하는 코드를 호출한다

```
/* USER CODE BEGIN 2 */  
HAL_TIM_Base_Start_IT(&htim2);  
//HAL_TIM_Base_Start_IT(&htim5);  
  
evtGrp = xEventGroupCreate();  
/* USER CODE END 2 */
```

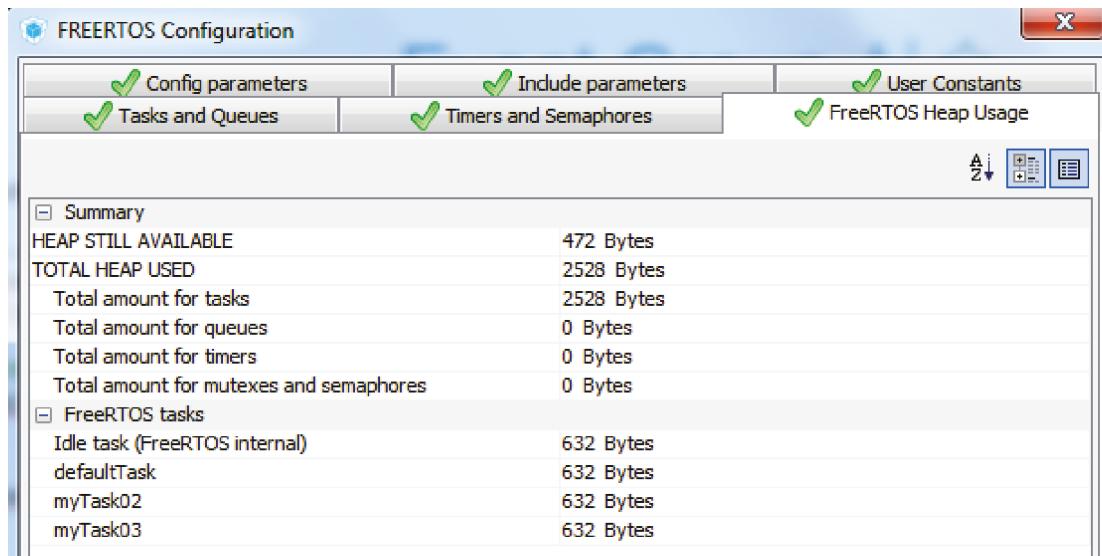
Event Group 실습

26

- 태스크를 여러개 생성하는 경우 HEAP 사이즈가 충분한지 유의한다



- CubeMX 를 사용해서 태스크를 생성할 경우 위의 TOTAL_HEAP_SIZE 대비 현재 사용중인 HEAP 사이즈와 남아있는 HEAP 사이즈를 볼수 있다



Event Group 실습

27

- 태스크2 은 1초 마다 Event Group 의 Flag 1 을 세팅하도록 코드를 추가하고
- 태스크3 에 3초 마다 Event Group 의 Flag 2 를 세팅하도록 코드를 추가한다

```
/* USER CODE END Header_StartTask02 */
void StartTask02(void const * argument)
{
    /* USER CODE BEGIN StartTask02 */
    /* Infinite loop */
    for(;;)
    {
        xEventGroupSetBits(evtGrp, evt_flag_1);
        osDelay(1000);
    }
    /* USER CODE END StartTask02 */
}
void StartTask03(void const * argument)
{
    /* USER CODE BEGIN StartTask03 */
    /* Infinite loop */
    for(;;)
    {
        xEventGroupSetBits(evtGrp, evt_flag_2);
        osDelay(3000);
    }
    /* USER CODE END StartTask03 */
}
```

Event Group 실습

28

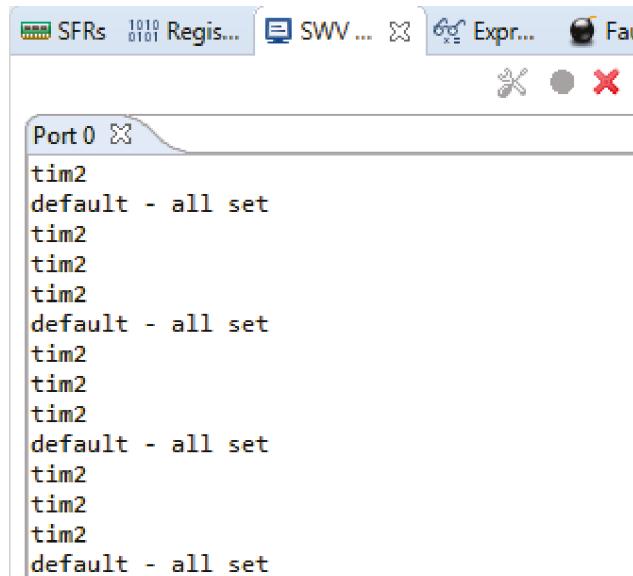
- default 태스크는 Event Group 의 Flag 1 과 2가 동시에 세팅되는것을 5초 동안 기다리고 결과를 itm_print 로 출력 한다

```
void StartDefaultTask(void const * argument)
{
    /* USER CODE BEGIN 5 */
    uint32_t result;
    /* Infinite loop */
    for(;;){
        result = xEventGroupWaitBits(
            evtGrp,
            (evt_flag_1 | evt_flag_2),
            pdTRUE, //xClearOnExit
            pdTRUE, //xWaitForAllBits
            5000); //xTicksToWait
        if((result&(evt_flag_1 | evt_flag_2)) == (evt_flag_1 | evt_flag_2)){
            sprintf(buff[2], "default - all set\n");
            itm_print(buff[2], strlen(buff[2]));
        }
        else{
            if(result & evt_flag_1){sprintf(buff[2], "default - flag1 set\n");}
            else if(result & evt_flag_2){sprintf(buff[2], "default - flag2 set\n");}
            else {sprintf(buff[2], "default - none set\n");}
            itm_print(buff[2], strlen(buff[2]));
        }
        osDelay(500);
    }
    /* USER CODE END 5 */
}
```

Event Group 실습

29

- 즉, 태스크2 은 1초 간격으로 Flag 1 을 세팅하고
- 태스크3 은 3 초 간격으로 Flag 2 를 세팅하도록 테스트 했을때
- 타이머2는 단순히 1초 간격으로 print 출력을 하고 있으며, default 태스크는 Flag 1 과 2 가 모두 세팅되는 3초마다 print 출력이 되는것을 확인할 수 있다



```
Port 0
tim2
default - all set
tim2
tim2
tim2
default - all set
tim2
tim2
tim2
default - all set
tim2
tim2
tim2
default - all set
```

- xEventGroupWaitBits 함수의 xClearOnExit 또는 xWaitForAllBits 인자를 pdFALSE 로 바꾸면 결과가 어떻게 바뀌는지 실습해본다

Event Group 실습

30

- xEventGroupWaitBits 함수의 xClearOnExit 또는 xWaitForAllBits 인자를 모두 pdTRUE 로 복구하고 타이머 2 를 아래와 같이 수정해서 다시 실습해본다

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    /* USER CODE BEGIN Callback 0 */

    /* USER CODE END Callback 0 */
    if (htim->Instance == TIM1) {
        HAL_IncTick();
    }
    /* USER CODE BEGIN Callback 1 */
    BaseType_t higher_task_waken=pdFALSE;

    if (htim->Instance == TIM2) {
        if (evtGrp!=NULL){
            xEventGroupSetBitsFromISR(evtGrp, evt_flag_2,&higher_task_waken);
            portYIELD_FROM_ISR(higher_task_waken);
        }
        sprintf(buff[0], "%s", "tim2\r\n");
        itm_print(buff[0], strlen(buff[0]));
    }
    if (htim->Instance == TIM5) {
        sprintf(buff[1], "%s", "tim5\r\n");
        itm_print(buff[1], strlen(buff[1]));
    }
    /* USER CODE END Callback 1 */
}
```

Event Group 실습

31

- xEventGroupSetBitsFromISR 을 사용하려면 CubeMX 에서 software 타이머와 trace facility 를 enable 하고

Software timer definitions

USE_TIMERS	Enabled
TIMER_TASK_PRIORITY	2
TIMER_QUEUE_LENGTH	10
TIMER_TASK_STACK_DEPTH	256 Words

Run time and task stats gathering related defini...

GENERATE_RUN_TIME_STATS	Disabled
USE_TRACE_FACILITY	Enabled
USE_STATS_FORMATTING_FUNCTION	Disabled

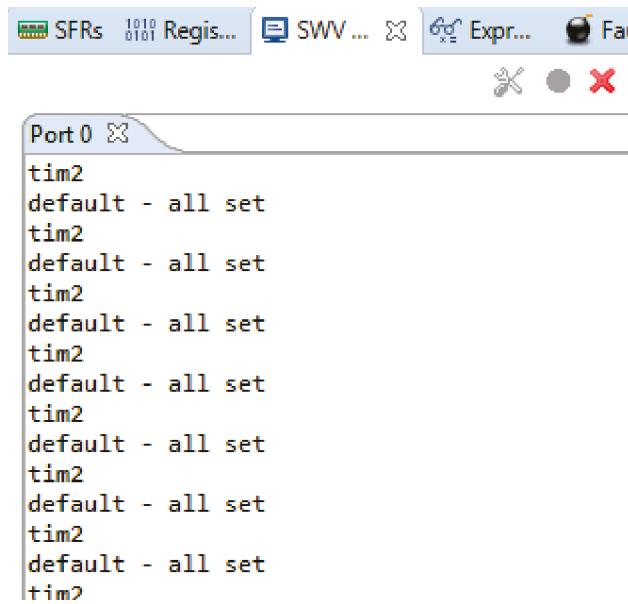
- 아래 두가지를 enable 시켜주고 코드를 다시 생성한다

xTaskGetCurrentTaskHandle	Disabled
eTaskGetState	Disabled
xEventGroupSetBitFromISR	Enabled
xTimerPendFunctionCall	Enabled
xTaskAbortDelay	Disabled

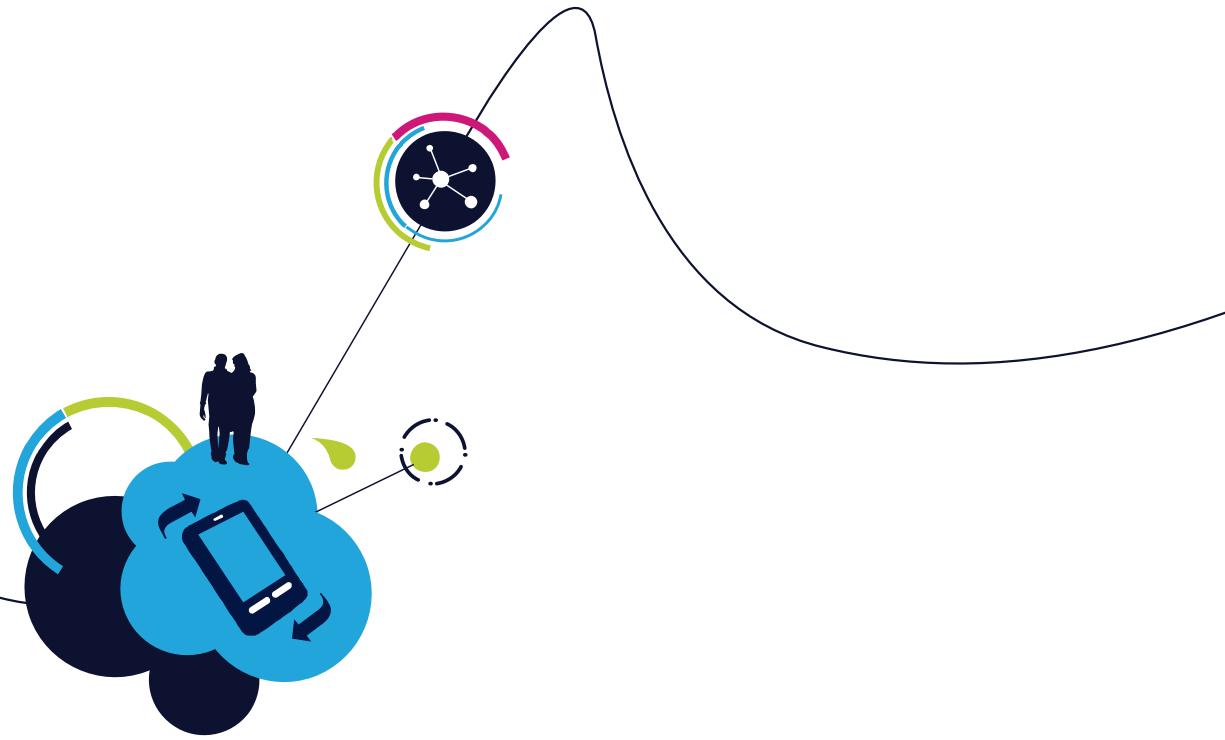
Event Group 실습

32

- 즉, 태스크2 은 1초 간격으로 Flag 1 을 세팅하고
- 태스크3 은 3 초 간격으로 Flag 2 를 세팅하고
- 타이머2 는 1초 간격으로 Flag 2 를 세팅하도록 테스트 했을때
- default 태스크는 Flag 1 과 2 가 모두 세팅되는 1초마다 print 출력이 되는것을 확인할 수 있다



```
Port 0
tim2
default - all set
tim2
```



FreeRTOS Co-Routine

Co-Routine

34

- 태스크는 태스크별로 **stack** 을 할당해서 사용하는 반면 Co-Routine 들은 하나의 **stack** 을 공유해서 사용한다
- 램 용량이 많이 부족한 시스템에서 태스크 대신에 Co-Routine 이 사용되며 태스크의 TCB (Task Control Block)보다 작은 사이즈의 CRCB (Co-Routine Control Block) 가 사용된다
- 태스크는 Systick (PendSV) 을 통해서 스케줄링이 이루어지는 반면 Co-Routine 은 Idle 태스크 hook 에서 vCoRoutineSchedule() 를 호출하는 코드를 넣어 줌으로서 스케줄링이 이루어진다
- Idle 태스크는 Running(Ready) 상태의 태스크가 없을때만 수행되기 때문에 태스크와 Co-Routine 을 동시에 함께 사용할수 있지만 태스크의 우선순위가 항상 Co-Routine 보다 높다

- Co-Routine 의 priority 는 0 부터 configMAX_CO_ROUTINE_PRIORITIES-1 값으로 설정하며 태스크 priority와 무관한 Co-Routine 들 사이의 priority 값으로 사용된다
- Co-Routine 은 함수 시작부분에 crSTART(), 마지막 부분에 crEND() 함수를 호출해야 하며 함수가 리턴 되면 안되기 때문에 무한 루프로 처리한다
- Co-Routine 용 API 함수를 사용해야 한다 (예를 들어, crDELAY(), crQUEUE_SEND() 등)
- Stack 을 공유해서 사용하기 때문에 Co-Routine 함수 내부에는 전역변수 또는 static 변수만 사용해야 하며 Co-Routine 용 API 함수는 분기된 다른 함수에서 호출되면 안되고 Co-Routine 내부에서 직접 호출되어야 한다
- Switch case 문 안에서 crDELAY() 와 같은 blocking 함수를 사용하면 안된다
- Co-Routine 은 사용법에 제약사항을 유의해야 하므로 램 용량이 부족하지 않다면 사용을 권장하지 않는다

- 제공되는 API 는 아래와 같다

Co-routine specific [API]

Modules

- CoRoutineHandle_t
- xCoRoutineCreate
- crDELAY
- crQUEUE_SEND
- crQUEUE_RECEIVE
- crQUEUE_SEND_FROM_ISR
- crQUEUE_RECEIVE_FROM_ISR
- vCoRoutineSchedule

- Co-Routine 을 사용하기 위해서는 CubeMX 의 다음 항목을 enable 한다

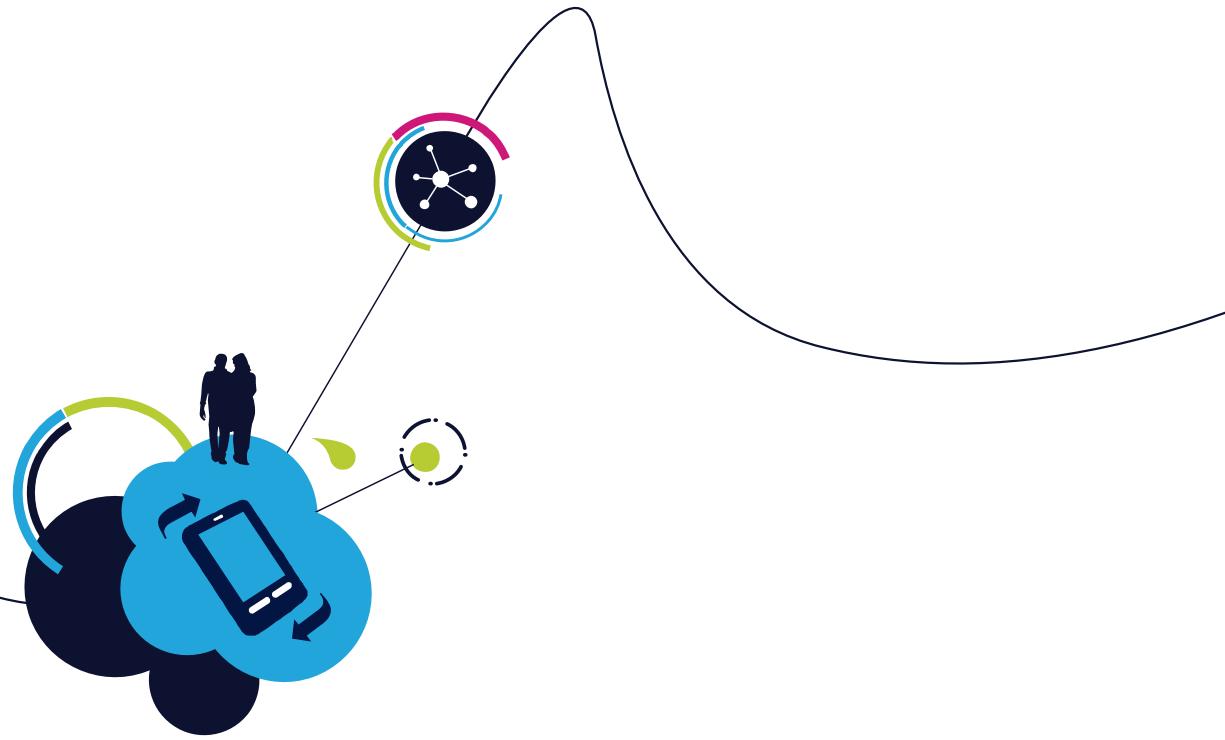
↳ Co-routine related definitions	
USE_CO_ROUTINES	Enabled
MAX_CO_ROUTINE_PRIORITIES	2
↳ Hook function related definitions	
USE_IDLE_HOOK	Enabled
USE_TICK_HOOK	Disabled
USE_MALLOC_FAILED_HOOK	Disabled
USE_DAEMON_TASK_STARTUP_HOOK	Disabled

- Idle hook 핸들러에서 Co-routine 스케줄러를 호출해야 한다

```
weak void vApplicationIdleHook( void )
{
    /* vApplicationIdleHook() will only be called if configUSE_IDLE_HOOK is set
     * to 1 in FreeRTOSConfig.h. It will be called on each iteration of the idle
     * task. It is essential that code added to this hook function never attempts
     * to block in any way (for example, call xQueueReceive() with a block time
     * specified, or call vTaskDelay()). If the application makes use of the
     * vTaskDelete() API function (as this demo application does) then it is also
     * important that vApplicationIdleHook() is permitted to return to its calling
     * function, because it is the responsibility of the idle task to clean up
     * memory allocated by the kernel to any task that has since been deleted. */
    vCoRoutineSchedule( );
}
```

- Co-Routine 의 사용 예제는 아래와 같다

```
/* USER CODE BEGIN Includes */
#include "croutine.h"
/* USER CODE END Includes */
void CoRoutine1(xCoRoutineHandle xHandle, unsigned portBASE_TYPE uxIndex)
{
    static uint32_t no_local_variable = 0x11;
    crSTART( xHandle );
    for( ; ; ){
        no_local_variable++;
        crDELAY( xHandle, 10 );
    }
    crEND();
}
void CoRoutine2(xCoRoutineHandle xHandle, unsigned portBASE_TYPE uxIndex)
{
    crSTART( xHandle );
    for( ; ; ){
        crDELAY( xHandle, 20 );
    }
    crEND();
}
void main(void)
{
...
    xCoRoutineCreate( CoRoutine1, configMAX_CO_ROUTINE_PRIORITIES, 0 );
    xCoRoutineCreate( CoRoutine2, configMAX_CO_ROUTINE_PRIORITIES, 0 );
}
```



FreeRTOS Task Notification

Task Notification

40

- 태스크 생성시 32bit 의 notification value 가 태스크마다 하나씩 할당된다. Notification value 를 통해서 태스크들 및 인터럽트 핸들러들 사이에 동기화 신호를 주거나 및 32bit 데이터 전송을 할수 있는 용도로 사용된다
- Queue, semaphore, event group 과 같이 송수신 중간에 객체를 사용해 전달하는 방법이 아닌 태스크로의 직접 전달 방법이기 때문에 binary semaphore 보다 속도와 램 사용량에 이점이 있다
- Queue, semaphore, event group 은 핸들을 가지고 있는 여러개의 태스크에서 수신을 할수 있지만 Task Notification 은 하나의 수신 태스크만 사용 가능하다
- Task Notification 은 인터럽트 (또는 태스크)에서 송신하고 태스크에서 수신이 가능하다
- Task Notification 은 태스크에서 송신하고 인터럽트에서 수신은 불가능하다
- Queue, semaphore, event group 의 송신은 blocked 상태에서 송신 완료를 기다릴수 있지만 Task Notification 의 송신은 단순히 overwrite 된다

- 제공되는 API 는 아래와 같다

RTOS Task Notifications

[API]

RTOS task notification API functions:

- `xTaskNotifyGive()`
- `vTaskNotifyGiveFromISR()`
- `ulTaskNotifyTake()`
- `xTaskNotify()`
- `xTaskNotifyAndQuery()`
- `xTaskNotifyAndQueryFromISR()`
- `xTaskNotifyFromISR()`
- `xTaskNotifyWait()`
- `xTaskNotifyStateClear()`

Task Notification

42

- Task Notification 을 이용하기 위해서는 CubeMX 에서 다음 항목을 enable 한다

ENABLE_BACKWARD_COMPATIBILITY	Enabled
USE_PORT_OPTIMISED_TASK_SELECTION	Enabled
USE_TICKLESS_IDLE	Disabled
USE_TASK_NOTIFICATIONS	Enabled
RECORD_STACK_HIGH_ADDRESS	Disabled

Task Notification

43

- Task Notification 을 binary semaphore 처럼 사용하는 예제는 아래와 같다

```
void StartDefaultTask(void const * argument)
{
    /* USER CODE BEGIN 5 */
    uint32_t noti_value;
    /* Infinite loop */
    for(;;){
        noti_value = ulTaskNotifyTake( pdTRUE, 3000 ); //xClearCountOnExit, xTicksToWait
        if(noti_value != 0){sprintf(buff[2], "something rcvd\n");}
        else {sprintf(buff[2], "nothing rcvd\n");}
        itm_print(buff[2], strlen(buff[2]));
    }
    /* USER CODE END 5 */
}

void StartTask02(void const * argument)
{
    /* USER CODE BEGIN StartTask02 */
    /* Infinite loop */
    for(;;)
    {
        xTaskNotifyGive( defaultTaskHandle);
        osDelay(1000);
    }
    /* USER CODE END StartTask02 */
}
```

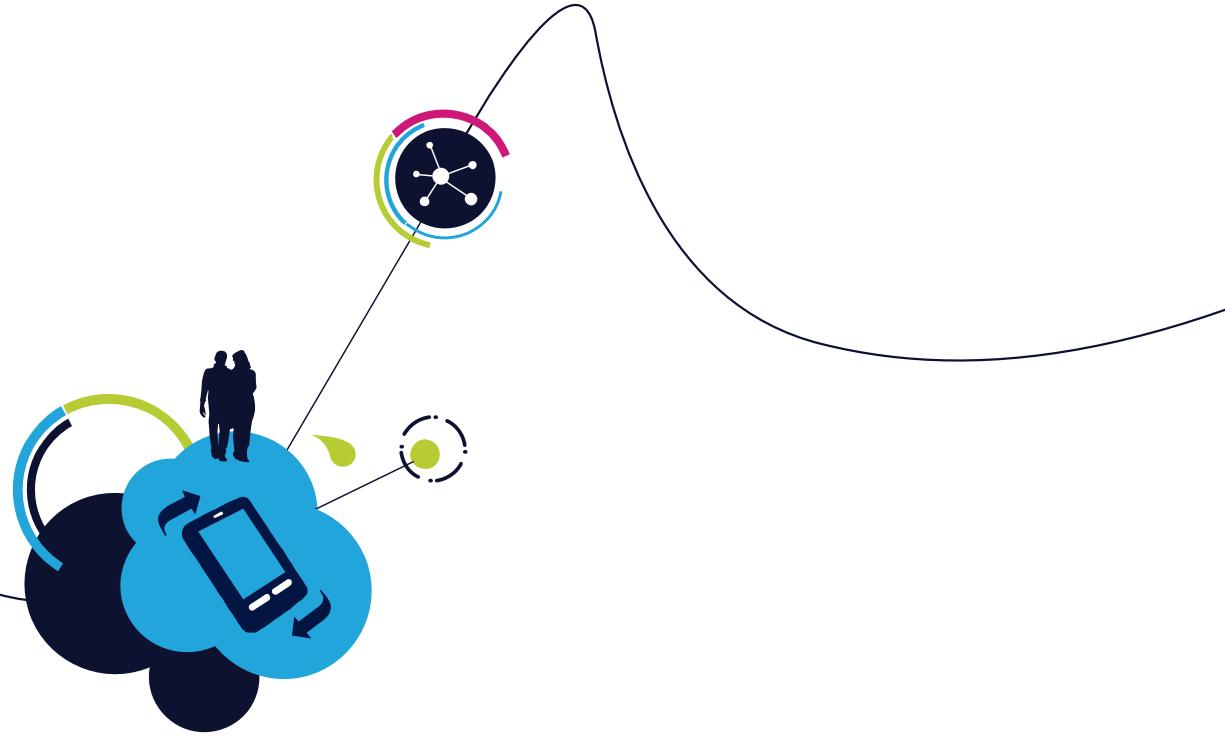
Task Notification

- Task Notification 을 동기화 및 데이터 전송으로 사용하는 예제는 아래와 같다

```

void StartDefaultTask(void const * argument)
{
    /* USER CODE BEGIN 5 */
    uint32_t noti_value;
    /* Infinite loop */
    for(;;){
        xTaskNotifyWait( 0x00,          /* Don't clear any notification bits on entry. */
                        0xffffffff, /* Reset the notification value to 0 on exit. */
                        &noti_value, /* Notified value pass out in noti_value. */
                        portMAX_DELAY ); /* Block indefinitely. */
        if(noti_value & 0x01){sprintf(buff[2], "bit0 set\n");}
        else {sprintf(buff[2], "bit0 else\n");}
        item_print(buff[2], strlen(buff[2]));
    }
    /* USER CODE END 5 */
}
void StartTask02(void const * argument)
{
    /* USER CODE BEGIN StartTask02 */
    /* Infinite loop */
    for(;;){
        xTaskNotify( defaultTaskHandle, 0x01, eSetBits );
        osDelay(1000);
        xTaskNotify( defaultTaskHandle, 0x02, eSetBits );
        osDelay(1000);
    }
    /* USER CODE END StartTask02 */
}

```



FreeRTOS Run Time Stack Checking

Run Time Stack Checking

- 태스크에 할당된 stack 의 overflow 체크를 위해 아래 상수를 1 또는 2로 설정한다 (FreeRTOSConfig.h)
 - configCHECK_FOR_STACK_OVERFLOW 가 1 인 경우, 스케줄러 vTaskSwitchContext() 함수가 호출될 때마다 TCB 에 있는 stack top 주소 변수와 현재 stack 주소 변수를 비교해서 stack overflow 를 검사한다
 - configCHECK_FOR_STACK_OVERFLOW 가 2 인 경우, 태스크가 생성될때 stack 전체를 특정 패턴의 데이터 (0xa5) 로 채우고 스케줄러 vTaskSwitchContext() 함수가 호출될 때마다 현재 stack 주소에서 20 byte 를 읽어서 특정패턴(0xa5) 이 다른 데이터로 overwrite 되지 않았는지 검사한다
- Stack overflow 후킹 함수는 위의 상수가 1 또는 2로 설정된 경우, stack overflow 검사시 발생한 태스크의 핸들과 이름을 인자로 리턴한다. Stack overflow 가 심할 경우 잘못된 태스크 이름이 전달될 수도 있고 후킹 함수가 호출 조차 안될수 있기 때문에 신뢰도는 제한적이다
 - vApplicationStackOverflowHook(xTaskHandle *pxTask, signed portCHAR *pcTaskName)
- 태스크 생성 이후부터 기록된 남아 있는 최소 stack 크기 조회를 매뉴얼 하게 하려면 INCLUDE_uxTaskGetStackHighWaterMark 를 enable 하고 아래 함수를 호출한다. 리턴값 1 은 1 word 를 나타낸다.
 - unsigned portBASE_TYPE uxTaskGetStackHighWaterMark(xTaskHandle xTask);

Task Notification & Run Time Stack Checking 실습

- 남아 있는 stack 크기를 조회하려면 CubeMX에서 아래 옵션을 enable 한다

xTaskResumeFromISR	Enabled
xQueueGetMutexHolder	Disabled
xSemaphoreGetMutexHolder	Disabled
pcTaskGetName	Disabled
uxTaskGetStackHighWaterMark	Enabled
xTaskGetCurrentTaskHandle	Disabled
eTaskGetState	Disabled

- 앞장에서 사용한 예제의 타이머2 시작도 주석처리 한다

```
/* USER CODE BEGIN 2 */
//HAL_TIM_Base_Start_IT(&htim2);
//HAL_TIM_Base_Start_IT(&htim5);
/* USER CODE END 2 */
```

Task Notification & Run Time Stack Checking 실습

48

- 태스크2에서 default 태스크로 1초 간격으로 tick_count 를 notification value로 보내도록 코드를 아래와 같이 수정한다

```
void StartTask02(void const * argument)
{
    /* USER CODE BEGIN StartTask02 */
static uint32_t tick_count;
    /* Infinite loop */
    for(;;)
    {
        xTaskNotify( defaultTaskHandle, ++tick_count, eSetValueWithOverwrite );
        osDelay(1000);
    }
    /* USER CODE END StartTask02 */
}
```

Task Notification & Run Time Stack Checking 실습

49

- default 태스크를 아래와 같이 수정한다

```
void StartDefaultTask(void const * argument)
{
    /* USER CODE BEGIN 5 */
    uint32_t noti_value;
    /* Infinite loop */
    for(;;){
        xTaskNotifyWait( 0x00,      /* Don't clear any notification bits on entry. */
                        0xffffffff, /* Reset the notification value to 0 on exit. */
                        &noti_value, /* Notified value pass out in noti_value. */
                        portMAX_DELAY ); /* Block indefinitely. */
        sprintf(buff[2], "noti:%d, stack:%d\n", noti_value,
uxTaskGetStackHighWaterMark(defaultTaskHandle));
        itm_print(buff[2], strlen(buff[2]));
        StartDefaultTask(NULL); //스택 overflow 를 만들기 위한 Recursive call
    }
    /* USER CODE END 5 */
}
```

- Recursive 함수로 인해 stack 이 줄어드는것을 확인할 수 있다



Port 0	
noti:1,	stack:99
noti:2,	stack:22
noti:3,	stack:18
noti:4,	stack:14
noti:5,	stack:10
noti:6,	stack:6

Task Notification & Run Time Stack Checking 실습

50

- stack overflow 검사를 하려면 CubeMX에서 아래 옵션을 1이나 2로 선택한다.
실습은 2로 선택한다.

✓ Hook function related definitions	
USE_IDLE_HOOK	Enabled
USE_TICK_HOOK	Disabled
USE_MALLOC_FAILED_HOOK	Disabled
USE_DAEMON_TASK_STARTUP_HOOK	Disabled
CHECK_FOR_STACK_OVERFLOW	Option2
✓ Run time and task stats gathering related definitions	
GENERATE_RUN_TIME_STATS	Disabled

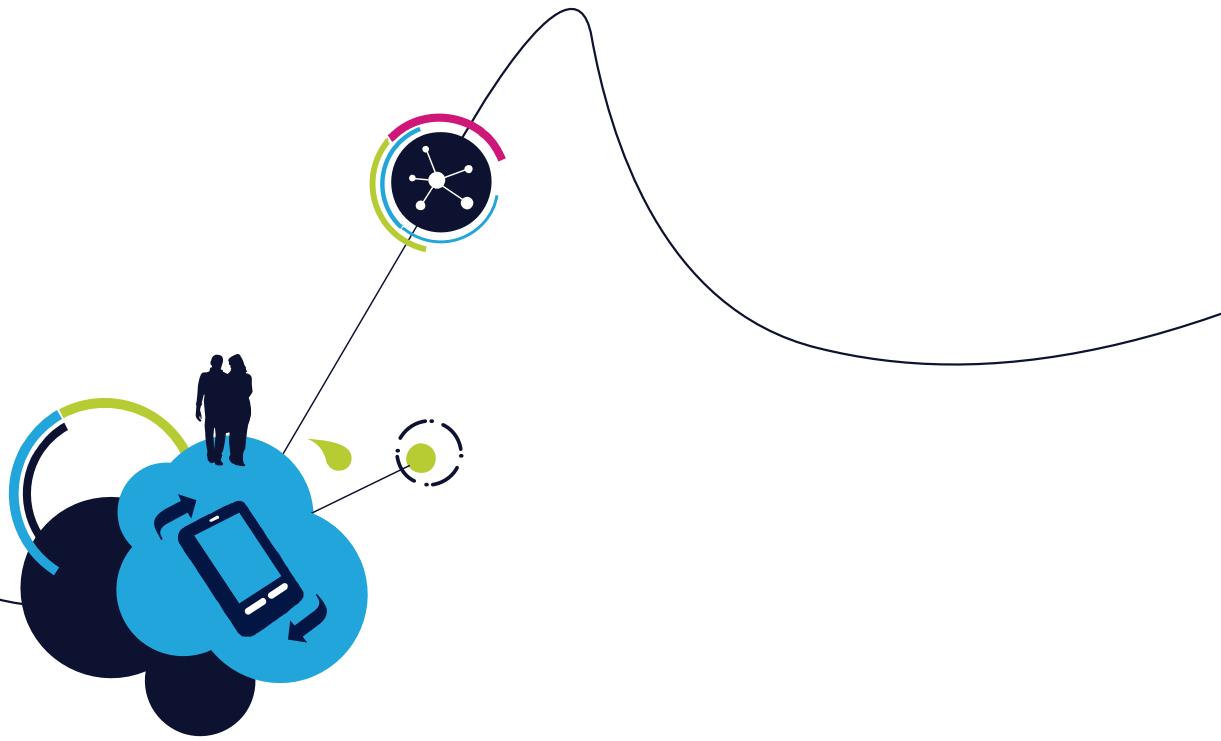
- Stack overflow 후킹 함수를 아래와 같이 추가해준다

```
/* USER CODE BEGIN 4 */
__weak void vApplicationStackOverflowHook(xTaskHandle xTask, signed char *pcTaskName)
{
    /* Run time stack overflow checking is performed if
    configCHECK_FOR_STACK_OVERFLOW is defined to 1 or 2. This hook function is
    called if a stack overflow is detected. */
    extern char buff[5][64];
    extern void itm_print(char *p, int len);
    sprintf(buff[2], "overflow: %s\n", pcTaskName);
    itm_print(buff[2], strlen(buff[2]));
}
/* USER CODE END 4 */
```

Task Notification & Run Time Stack Checking 실습

- 아래와 같이 4 word 크기의 stack 이 남은 지점부터 특정패턴(0xa5) 20 byte 검사에서 stack overflow 가 감지되면서 후킹 함수의 print 문이 호출되는 것을 확인 할 수 있다

```
Port 0
noti:1, stack:99
noti:2, stack:22
noti:3, stack:18
noti:4, stack:14
noti:5, stack:10
noti:6, stack:6
overflow: defaultTask
noti:7, stack:2
overflow: defaultTask
noti:8, stack:0
overflow: defaultTask
noti:9, stack:0
overflow: defaultTask
```



Thank you