# Xilinx Zynq FPGA, TI DSP, MCU 기반의 회로설계 및 임베디드 전문가 과정

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 - 김형주
mihaelkel@naver.com

# FreeRTOS 분석 및 사용법

- What is FreeRTOS?

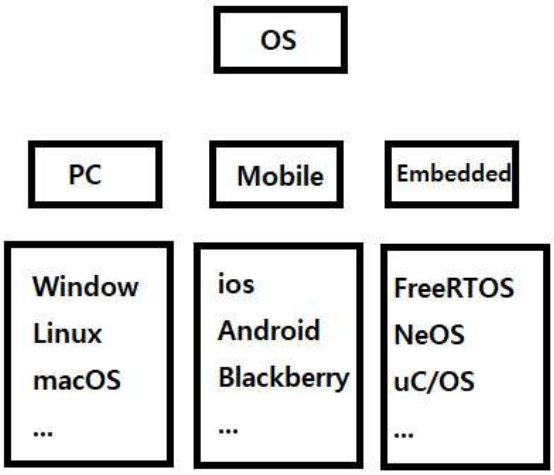
- FreeRTOS 올리기(TI Hercules MCU – TMS570LC43xx Cortex R5F)

- FreeRTOS Task 동기화 하기(TI Hercules MCU – TMS570LC43xx Cortex R5F)

## What Is FreeRTOS?

### 1.멀티 태스킹

FreeRTOS는 Amazon사에서 개발한 RTOS의 한 종류이고, RTOS는 OS중 특수한 OS를 말한다. OS라는 것은 우리가 흔히 사용하는 윈도우, 리눅스, 안드로이드, IOS 등을 말한다.




OS는 크게 5가지 기능을 가진다. 파일시스템 관리, 메모리 관리, 프로세스 관리, 입출력 장치 관리, 네트워크 관리를 담당한다. 임베디드 시스템에 일반 OS를 올리기에는 너무 무겁고, 비효율적이며 MMU를 탑재하고 있지 않아 독립적인 가상 메모리 공간을 가질 수 없어 멀티프로세싱이 불가능하다. 이런 종류의 제약이 너무 많기 때문에, RTOS는 OS의 기능 중 태스크 관리(프로세스 관리)만을 가져와 사용하는 OS라고 볼 수 있다.

아래와 같은 펌웨어 프로그램을 작성한다고 하자.

```
#include "HL_sys_common.h"
#include "HL_gio.h"

int main(void)
{
    gioInit();

    while(1)
    {
        delay(8000000);
        gioToggleBit(gioPORTA, 0);
    }
}
```

아주 간단한, portA0핀 하나를 8,000,000 delay를 기준으로 toggle하는 코드이다.
다음으로 아래의 코드를 보자.

```
#include "HL_sys_common.h"
#include "HL_etpwm.h"

int main(void)
{
    etpwmInit();
    etpwmStartTBCLK();

    while(1)
    {
        delay(10000);
        etpwmStartTBCLK();
        delay(10000);
        etpwmStopTBCLK();
    }
    return 0;
}
```

좋은 코드는 아니지만, 10,000 delay를 기준으로 pwm을 생성했다가, 멈췄다가 반복하는 코드이다.
위 두 가지 프로그램을 1개의 MCU에서 돌리기 위해서는 어떻게 해야 할까?

```
#include "HL_sys_common.h"
#include "HL_gio.h"
#include "HL_etpwm.h"

int main(void)
{
    int cnt;
    gioInit();
    etpwmInit();

    while(1)
    {
        cnt++;
        delay(10000);
        etpwmStartTBCLK();
        delay(10000);
        etpwmStopTBCLK();
        if(cnt == 400)
            gioToggleBit(gioPORTA, 0);
        cnt %= 400;
    }
}
```

두 프로그램 다 아주 간단한 형태이기 때문에, 위와 같이 쓸 수도 있을 것이다.

아래의 두 프로그램을 살펴보자.

```
int main(void)
{
    uint8 msg[32] = {'M','S','G','\r','\n'};
    /*this gets ADC id and value*/
    adcData_t recv_adc;
    /*restore "id" , "value" in "recv_adc"*/
    uint32 id;
    uint32 value;

    /*peripheral Init*/
    etpwmInit();
    sciInit();
    adcInit();
    gioInit();

    /*Init sound*/
    set_freq(None);

    /*set PINMUX(B5) to etPWM*/
    etpwmStartTBCLK();
    /*adc conversion starts*/
    adcStartConversion(adcREG1, adcGROUP1);

    while(1)
    {
        /*set trigger high for adc conversion*/
        gioSetBit(gioPORTB, 0, 1);

        /*waiting for completing adc conversion*/
        while(adcIsConversionComplete(adcREG1,adcGROUP1)
== 0)
            ;
        /*get the converted data*/
        adcGetData(adcREG1, adcGROUP1, &recv_adc);
        /*save the converted data*/
        id = recv_adc.id;
        value = recv_adc.value;
        /*data type casting to string*/
        sprintf(msg,"id : %d\tvalue : %d\r\n",id,value);

        /*trigger low*/
        gioSetBit(gioPORTB, 0, 0);

        /*send adc date by UART1(sci1)*/
        SendData(msg,strlen(msg)+1);

        /*set PWM period by value, which fixed by what
button is pushed*/
        sel_freq(value);
    }
    return 0;
}
```

```
int main(void)
{

    _enable_IRQ_interrupt_();

    canInit();

    printf("start\n");

    canEnableErrorNotification(canREG1);
    while(1)
    {

        delay(80000000);
        canTransmit(canREG1, canMESSAGE_BOX1, (const uint8
*)&tx_data[0]);

        printf("rx_data : %x\n",*rx_data);
    }
    return 0;
}
```

왼쪽은 전자 오르간 프로그램의 일부이고, 오른쪽은 CAN 통신 프로그램이다.

두 프로그램을 하나의 MCU에서 동시에 동작하도록 한다면, 가능은 하겠지만 꽤나 복잡할 것이다.

또한, 여러 가지 기능을 하나의 프로그램에 작성하면, 무슨 프로그램인지 한 눈에 알기 매우 어렵다.

이런 여러 개의 TASK를 동시에 동작할 수 있도록 하는 것이 RTOS의 목적이다.

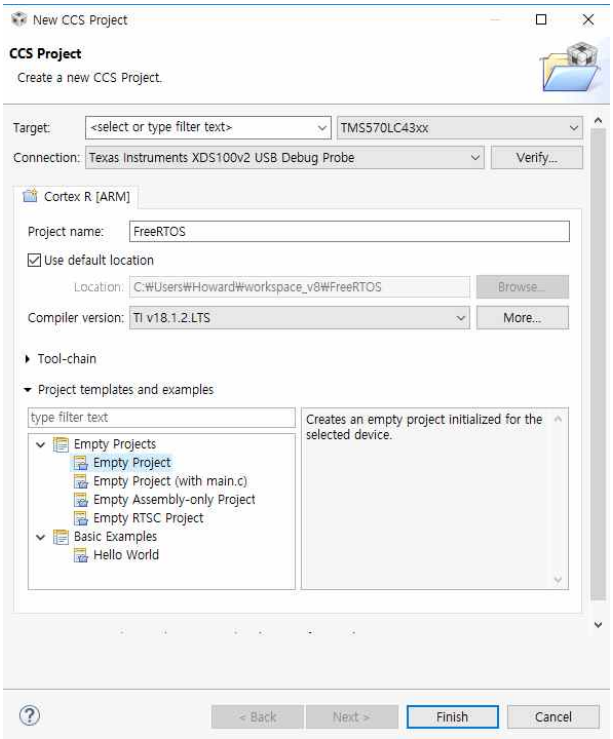
RTOS를 사용하면, 아래와 같이 몇 줄 추가하는 것으로 간단히 해결할 수 있고, Task별로 함수를 작성하기 때문에 코드를 파악하기에 매우 좋다.

```
	if(xTaskCreate(vTask1, "Piano" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask1Handle) != pdTRUE)
	{
		while(1)
			;
	}
	if(xTaskCreate(vTask2, "CAN" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask2Handle) != pdTRUE)
	{
		while(1)
			;
	}
	vTaskStartScheduler();
	while(1);
```

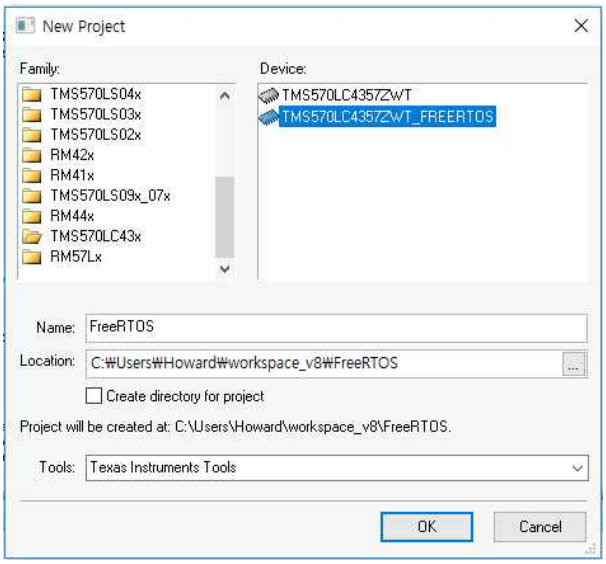
# FreeRTOS 올리기(TI Hercules MCU - TMS570LC43xx Cortex R5F)

FreeRTOS project를 만들어 보자.

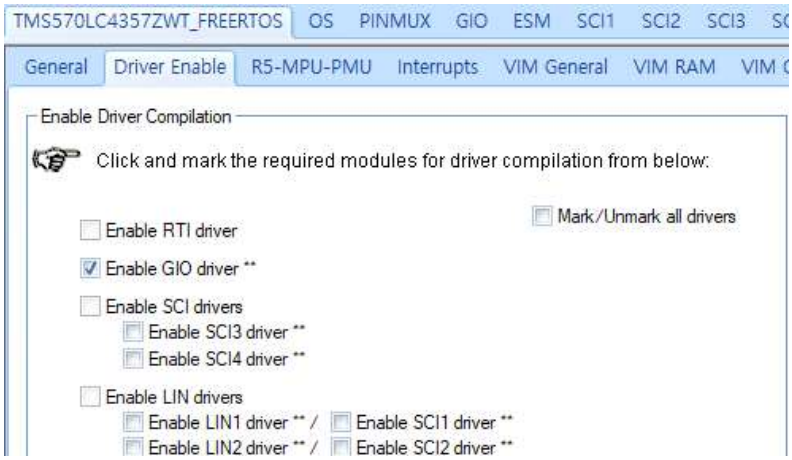
1.먼저, CCS에서 프로젝트를 생성한다.(CCS 실행 -> File -> New -> CCS Project)



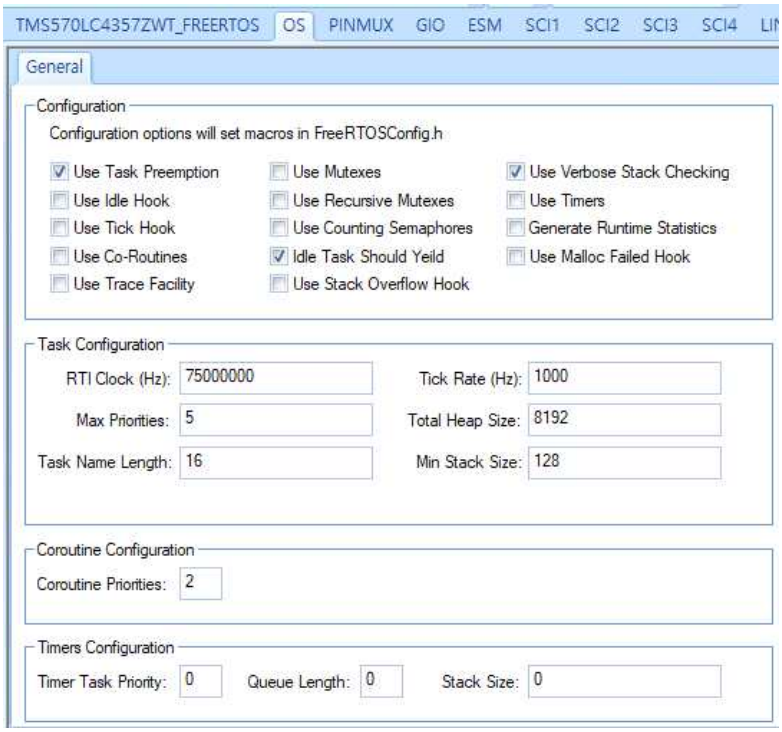

2.HALCoGen을 실행하여, 프로젝트를 만든다.(HALCoGen 실행 -> File -> New -> Project)




3.Driver Enable 탭으로 가서, 원하는 peripheral들을 추가한다.



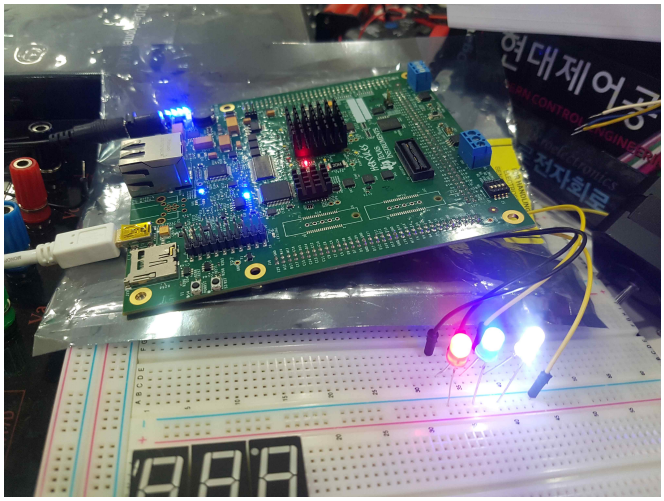
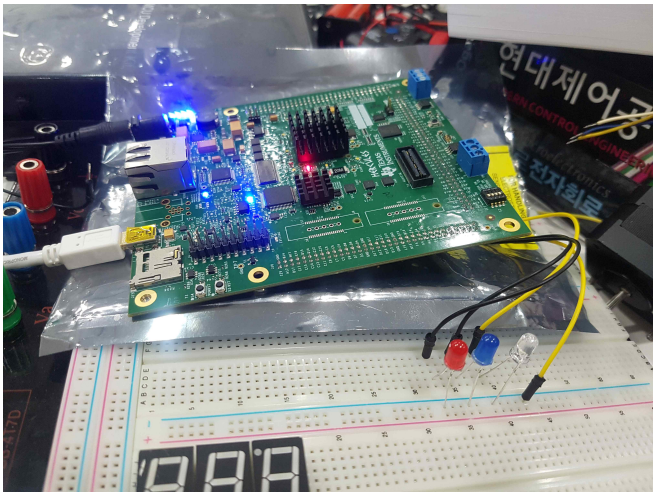
4.os탭으로 가서 설정을 해준다.




5.코드를 생성한다. (File -> Generate Code , or just press F5)

6.아래와 같이 코드를 작성한다.

```
#include "HL_sys_common.h"
#include "FreeRTOS.h"
#include "os_task.h"
#include "HL_gio.h"
#include "HL_sys_common.h"

xTaskHandle xTask1Handle;
xTaskHandle xTask2Handle;
xTaskHandle xTask3Handle;
void vTask1(void *pbParameters)
{
    for(;;)
    {
        gioSetBit(gioPORTA, 0, gioGetBit(gioPORTA, 0)^1);
        vTaskDelay(1000);
    }
}
void vTask2(void *pbParameters)
{
    for(;;)
    {
        gioSetBit(gioPORTA, 1, gioGetBit(gioPORTA, 1)^1);
        vTaskDelay(1000);
    }
}
void vTask3(void *pbParameters)
{
    for(;;)
    {
        gioSetBit(gioPORTA, 2, gioGetBit(gioPORTA, 2)^1);
        vTaskDelay(1000);
    }
}
void main(void)
{
    gioInit();
    if(xTaskCreate(vTask1, "Task1" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask1Handle) != pdTRUE)
    {
        while(1)
            ;
    }
    if(xTaskCreate(vTask2, "Task2" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask2Handle) != pdTRUE)
    {
        while(1)
            ;
    }
    if(xTaskCreate(vTask3, "Task3" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask3Handle) != pdTRUE)
    {
        while(1)
            ;
    }
    vTaskStartScheduler();

    while(1);
}
```

태스크 생성에 필요한 함수들을 알아보자.

## xTaskCreate

```
BaseType_t xTaskCreate(         TaskFunction_t pxTaskCode,
                                                const char * const pcName,
                                                const uint16_t usStackDepth,
                                                void * const pvParameters,
                                                UBaseType_t uxPriority,
                                                TaskHandle_t * const pxCreatedTask )
```

xTaskCreate에는 6개의 인자가 들어간다.

pxTaskCode에는 생성하고자 하는 태스크의 함수를 넣는다.

pxTaskCode의 데이터타입을 타고 들어가보면

```
typedef void (*TaskFunction_t)( void * );
```

즉, TaskFunction_t 은 함수포인터 변수로 프로토타입은 void (*)(void *)이 됨을 알 수 있다.

따라서 태스크를 생성할 때에는 void (*)(void*) 형태의 함수로 작성해야 된다.

const char* const pcName에는 태스크의 이름을 작성한다.

태스크의 이름은 HalCoGen에서 설정한 글자수를 넘지 않도록 유의한다.


아래의 코드를 보면, 글자수가 설정한 제한보다 넘을 때 뒤의 글자는 잘리는 것을 알 수 있다.

os_task.c, line 827

```
        for( x = ( UBaseType_t ) 0; x < ( UBaseType_t ) configMAX_TASK_NAME_LEN; x++ )
        {
                pxNewTCB->pcTaskName[ x ] = pcName[ x ];

                /* Don't copy all configMAX_TASK_NAME_LEN if the string is shorter than
                configMAX_TASK_NAME_LEN characters just in case the memory after the
                string is not accessible (extremely unlikely). */
                if( pcName[ x ] == 0x00 )
                {
                        break;
                }
                else
                {
                        mtCOVERAGE_TEST_MARKER();
                }
        }
```

CCS 내에서 글자수 제한을 바꾸고 싶다면, configMAX_TASK_NAME_LEN을 수정하면 된다.

FreeRTOSConfig.h, line 105

```
#define configMAX_TASK_NAME_LEN                 ( 16 )
```

다음 인자는 const uint16_t usStackDepth이다. 말 그대로 생성되는 태스크의 스택 사이즈를 결정한다.
최소 스택 크기가 있는데, 이 또한 HalCoGen에서 설정할 수 있다.


Default 값으로 128로 되어 있음을 알 수 있다. 마찬가지로 바꾸고 싶다면, 아래 코드를 수정하면 된다.
FreeRTOSConfig.h, line 103

```
#define configMINIMAL_STACK_SIZE        ( ( unsigned portSHORT ) 128 )
```

다음 인자는 void* pvParameters 이다.
이 인자는 task를 생성할 때, main함수에서 인자를 주는 역할이라고 생각했다.
스택 초기화 함수를 살펴보면,

```
/* Function parameters are passed in R0. */
        *pxTopOfStack = ( StackType_t ) pvParameters; /* R0 */
        pxTopOfStack--;
```

주석도 그렇고 내부 코드도 태스크를 생성하면서 인자로 어떤 값을 주는, 그런 역할로 묘사되어있다.
하지만 아래와 같이 코드를 작성하면, gioPORTA1 핀에 출력이 HIGH값이 되어야 하는데 그렇지 않고,
모든 태스크가 멈춰버린다. 이 **문제를 해결하면, 나중에 수정하겠다.**

```
void vTask1(void *pbParameters)
{
    gioSetBit(gioPORTA, *((int*)pbParameters), 1);
    for(;;)
    {
    }
}
void main(void)
{
    gioInit();
    sciInit();
    int *sw = (int*)malloc(sizeof(int)*1);
    *sw = 1;
    if(xTaskCreate(vTask1, "Task1" , configMINIMAL_STACK_SIZE, (void*)sw, 1, &xTask1Handle) != pdTRUE)
    {
        while(1)
            ;
    }

    vTaskStartScheduler();

    while(1)
        ;
}
```

다음 인자는 UBaseType_t uxPriority 인데, 이름 그대로 우선순위를 나타낸다.
우선순위는 0부터 시작해서 설정한 최대값까지 존재한다. 숫자가 0이면 우선순위가 최하위이고, 높을수록 우선순위가 높다. 최대 우선순위는 HalCoGen에서 아래와 같이 설정이 가능하다.


이 또한 코드 상에서 변경이 가능하다.
FreeRTOSConfig.h, line 102

```
#define configMAX_PRIORITIES            ( 5 )
```

우선순위의 최대값이 높을수록 미세한 우선순위 조정이 가능하지만, 그만큼 성능이 떨어진다.
따라서 시스템을 생각하여 적정값을 넣어주는 것이 좋다.

마지막 인자는 TaskHandle_t * const pxCreatedTask 이다. 여기에는 main.c에서 전역변수로 xTaskHandle 타입의 변수를 생성하여 넣어준다. **이 인자의 역할도 추후 알게되면 수정하겠다.**

```
xTaskHandle xTask1Handle;
          :
          :

void main(void)
{
    if(xTaskCreate(vTask1, "Task1" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask1Handle) != pdTRUE)
    {
        while(1)
            ;
    }

    vTaskStartScheduler();

    while(1)
        ;
}
```

## vTaskDelay

```
void vTaskDelay( const TickType_t xTicksToDelay )
```

이 함수는 인자값 만큼 태스크를 delay하는 역할을 한다.
인자값은 ms단위이다. 즉, 1초를 delay하고 싶다면,

```
vTaskDelay(1000);
```

위와 같이 사용할 수 있다.

## vTaskStartScheduler

```
vTaskStartScheduler();
```

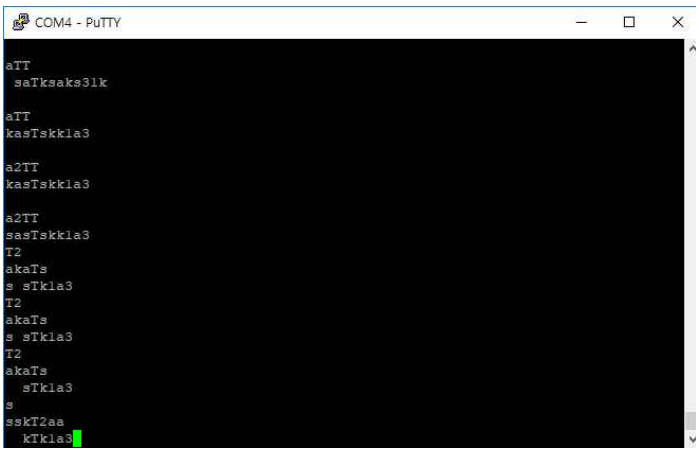
태스크의 스케줄러를 시작한다.
최초의 태스크인, Idle task를 생성하고 스케줄링을 한다.
Idle task의 priorty값은 0 이므로, 일단 스케줄링이 시작되면 CPU 점유율에 영향을 미치지 않는다.

이러한 함수들을 사용하면, 태스크를 여러 개 만들어서 멀티태스킹을 할 수 있다.
그럼 목적을 달성한 것인가?
아래의 코드를 보자.

```
#include <FreeRTOS.h>
#include <FreeRTOSConfig.h>
#include <HL_gio.h>
#include <HL_hal_stdtypes.h>
#include <HL_reg_sci.h>
#include <HL_sci.h>
#include <os_mpu_wrappers.h>
#include <os_projdefs.h>
#include <os_task.h>
#include <stdlib.h>
#include <string.h>

xTaskHandle xTask1Handle;
xTaskHandle xTask2Handle;
xTaskHandle xTask3Handle;
void send_data(sciBASE_t* sci, uint8* msg, int length);
void vTask1(void *pbParameters)
{
    uint8 msg[32] = {'T','a','s','k','1','\r','\n'};
    for(;;)
        send_data(sciREG1, msg, strlen(msg));
}
void vTask2(void *pbParameters)
{
    uint8 msg[32] = {'T','a','s','k','2','\r','\n'};
    for(;;)
        send_data(sciREG1, msg, strlen(msg));
}
void vTask3(void *pbParameters)
{
    uint8 msg[32] = {'T','a','s','k','3','\r','\n'};
    for(;;)
        send_data(sciREG1, msg, strlen(msg));
}
void main(void)
{
    sciInit();
    if(xTaskCreate(vTask1, "Task1" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask1Handle) != pdTRUE)
```

```
    {
        while(1)
            ;
    }

    if(xTaskCreate(vTask2, "Task2" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask2Handle) != pdTRUE)
    {
        while(1)
            ;
    }
    if(xTaskCreate(vTask3, "Task3" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask2Handle) != pdTRUE)
    {
        while(1)
            ;
    }

    vTaskStartScheduler();

    while(1)
        ;
}
void send_data(sciBASE_t* sci, uint8* msg, int length)
{
    int i;
    for(i=0;i<length;i++)
        sciSendByte(sci,msg[i]);
}
```

실행결과를 보면




3개의 Task가 Context Switching을 하기 때문에, 결과가 원하는 데로 나오질 않는다.

이 때 필요한 것이 동기화이다. 동기화를 할 때에는 mutex, semaphore, coroutine 등을 사용한다.

# FreeRTOS Task 동기화 하기(TI Hercules MCU - TMS570LC43xx Cortex R5F)

**뮤텍스 :**

친구 4명(A,B,C,D)이 자동차를 하나 같이 쓴다고 하자.
자동차도 1개이고, 자동차키 또한 1개 일 것이다. 만일 A가 자동차키를 가지고
차를 사용중이라고 하면, B, C, D 3명은 무슨 짓을 하든 그 자동차를 사용할 수 없을 것이다.
A가 다 사용하고 키를 반납하면, 그 때서야 4명 중 누구든 차를 운전할 수 있다.
위의 예에서 자동차를 시스템 자원, 친구(A, B, C, D)들을 태스크, 자동차키를 뮤텍스라고 생각하면 된다.

**세마포어 :**

반면, 세마포어는 위의 예시에서 완전히 동일한 자동차키와 자동차가 하나씩 더, 즉, 2개씩 있다고 생각하면 된다.
A가 자동차를 운전 중이라면, 남은 자동차와 자동차 키는 1개씩 줄어든다. B 또한 자동차를 사용한다고 하면, C와 D는 A와 B중 한 명이 자동차를 반납할 때까지 기다려야 한다.
세마포어 중 key의 개수가 1개인 세마포어를 바이너리 세마포어 라고 한다.

**차이점 :**

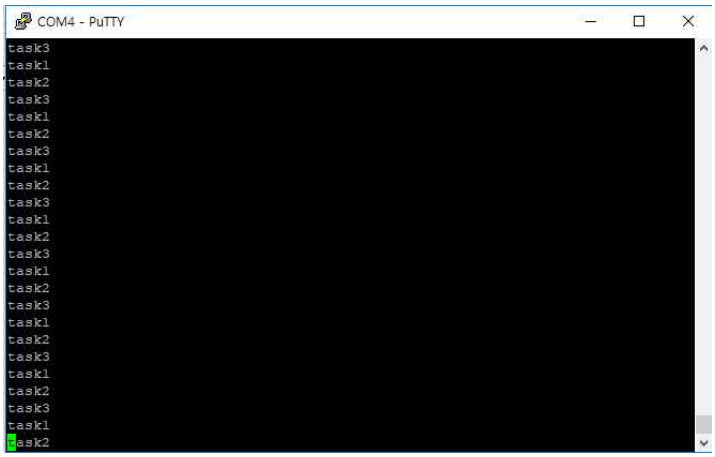
그렇다면 바이너리 세마포어 == 뮤텍스 인가?
기본적으로 그렇게 보이지만, 차이점은 있다. 뮤텍스의 경우 다른 태스크가 키를 갖기 위해서는, 키를 가지고 있는 태스크가 직접 키를 넘겨줘야 한다. 반면 세마포어는 키를 소유하는 개념이 아니기 때문에, 다른 태스크에서도 세마포어를 해제할 수 있다.

위의 3개의 sci 통신 태스크를 바이너리 세마포어를 통해 동기화하여 정상적인 문자열을 출력해보자.

```
#include <FreeRTOS.h>
#include <FreeRTOSConfig.h>
#include <HL_hal_stdtypes.h>
#include <HL_reg_sci.h>
#include <HL_sci.h>
#include <os_mpu_wrappers.h>
#include <os_projdefs.h>
#include <os_semphr.h>
#include <os_task.h>
#include <string.h>

xTaskHandle xTask1Handle;
xTaskHandle xTask2Handle;
xTaskHandle xTask3Handle;

QueueHandle_t mutex;


void vTask1(void* pvParameters);
void vTask2(void* pvParameters);
void vTask3(void* pvParameters);
void send_data(sciBASE_t* sci, uint8* msg, int length);

int main(void)
{
    sciInit();
    vSemaphoreCreateBinary(mutex)
    if(xTaskCreate(vTask1, "Task1" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask1Handle) != pdTRUE)
    {
        while(1)
            ;
    }
    if(xTaskCreate(vTask2, "Task2" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask2Handle) != pdTRUE)
    {
        while(1)
            ;
    }
    if(xTaskCreate(vTask3, "Task3" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask3Handle) != pdTRUE)
    {
        while(1)
            ;
    }
    vTaskStartScheduler();

    while(1)
        ;
    return 0;
}
void vTask1(void *pbParameters)
{
    uint8 msg[32] = {'t','a','s','k','1','\r','\n'};
    for(;;)
    {
        xSemaphoreTake(mutex, portMAX_DELAY);
        send_data(sciREG1, msg, strlen(msg));
        xSemaphoreGive(mutex);
        vTaskDelay(10);
```

```
    }
}
void vTask2(void *pbParameters)
{
    uint8 msg[32] = {'t','a','s','k','2','\r','\n'};
    for(;;)
    {
        xSemaphoreTake(mutex, portMAX_DELAY);
        send_data(sciREG1, msg, strlen(msg));
        xSemaphoreGive(mutex);
        vTaskDelay(10);
    }
}
void vTask3(void *pbParameters)
{
    uint8 msg[32] = {'t','a','s','k','3','\r','\n'};
    for(;;)
    {
        xSemaphoreTake(mutex, portMAX_DELAY);
        send_data(sciREG1, msg, strlen(msg));
        xSemaphoreGive(mutex);
        vTaskDelay(10);
    }
}

void send_data(sciBASE_t* sci, uint8* msg, int length)
{
    int i;
    for(i=0;i<length;i++)
        sciSendByte(sci,msg[i]);
}
```

결과 :




## vSemaphoreCreateBinary()

```
#define vSemaphoreCreateBinary( xSemaphore )
                \
                {
                                                                \
                    ( xSemaphore ) = xQueueGenericCreate( ( UBaseType_t ) 1, semSEMAPHORE_QUEUE_ITEM_LENGTH,
queueQUEUE_TYPE_BINARY_SEMAPHORE );         \
                    if( ( xSemaphore ) != NULL )
                                \
                    {
                                                                \
                        ( void ) xSemaphoreGive( ( xSemaphore ) );
                    \
                    }
                                                                \
                }
```

vSemaphoreCreateBinary() 는 함수가 아닌 매크로이다.

매크로 내부에 세미콜린이 포함되어 있으므로, 뒤에 세미콜론(;)을 붙이지 않고 사용한다.

인자에는 세마포어의 키를 저장하는 변수를 넣는다.

변수의 데이터 타입은 아래와 같이 선언한다.

```
QueueHandle_t mutex;
```

xSemaphoreTake();

```
#define xSemaphoreTake( xSemaphore, xBlockTime )
xQueueGenericReceive( ( QueueHandle_t ) ( xSemaphore ), NULL, ( xBlockTime ), pdFALSE )
```

xSemaphoreTake()는 세마포어 키를 얻는 함수이다.

즉, 보호하고자 하는 영역(Critical Section)에 접근하기 전에 사용한다.

첫 번째 인자에는 vSemaphoreCreateBinary() 매크로를 통해 얻은 키 변수를 넣고,

두 번째 인자에는 xBlockTime을 넣는다. 이 변수는 키를 얻을 때까지 얼마나 대기할거냐를 결정한다.

xBlockTime만큼 대기하였지만 키를 얻지 못했다면, 키를 얻지 않고 다음 명령어로 넘어간다.

최대 대기시간은 portMAX_DELAY 로 정의되어 있다.

```
#define portMAX_DELAY (TickType_t) 0xFFFFFFFF
```

xSemaphoreGive(mutex);

```
#define xSemaphoreGive( xSemaphore )
xQueueGenericSend( ( QueueHandle_t ) ( xSemaphore ), NULL, semGIVE_BLOCK_TIME, queueSEND_TO_BACK )
```

xSemaphoreGive(); 는 얻은 세마포어 키를 반환하는 함수이다.

당연히 인자에는 얻었던 키 변수가 들어간다.

**뮤텍스, 일반 세마포어, 코루틴 등은 향후 시간되면 추가함.**

참고자료 :

MMU, MPU 차이점 :

http://materer.tistory.com/15

태스크 관리 및 메모리 공유 :

http://wiki.tuestudy.org/aosabook/materials/FreeRTOS

http://www.ubinos.org/mediawiki/index.php/FreeRTOS_%EA%B5%AC%EC%A1%B0

FreeRTOS API 함수 목록 :

http://www.ubinos.org/mediawiki/index.php?title=FreeRTOS_%EC%86%8C%EA%B0%9C&printable=yes

FreeRTOS 공식 가이드 문서 :

https://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf

https://www.freertos.org/Documentation/FreeRTOS_Reference_Manual_V10.0.0.pdf