I DSP,Xilinx zynq FPGA,MCU 및

Xilinx

zynq FPGA 프로그래밍 전문가 과정

강사-INNOVA LEE(이상훈)

Gccompil3r@gmail.com

학생-윤지완

Yoonjw789 @naver.com

FreeRTOS

2. Real Time System

   A. 시스템의 처리 결과가 논리적인 처리 결과와 시간의 정확성(제한된 시간 내)에 모두 의존적인 시스템으로 특히 시간의 정확성이 우선시 되는 시스템

   B. Hard Real Time System

      i. 이벤트의 응답이 Deadline(Time-limit) 내에 처리 되어야 함.

      ii. 예: 모터제어 ; 일정한 시간 내에 속도와 토크 제어

   C. Soft Real Time System

      i. Deadline을 벗어난 응답이 결정적인 문제를 발생 시키지는 않음.

      ii. 예: 영상재상; 일정한 시간 내에 처리하지 못하면 일부 영상의 손실이 있으나 계속 재생 가능

3. RTOS의 필요성

   A. 제품의 개발 주기 및 라이프 사이클이 빨라짐

      i. 개발 기간을 단축 시키기 위하여

      ii. 검증된 Code를 재 사용 하거나,

      iii. 동시에 여러 명이 하나의 과제를 진행 할 필요

   B. 제품 기능의 복잡도 증가

      i. Real Time Control 필요,

      ii. Multi-Tasking의 기능이 필요한 제품 증가

   C. Microcontroller의 성능은 좋아지고 가격은 하락

      i. 대부분의 제품에서 RTOS에 의한 오버헤드 가 문제되지 않음

   D. 파일 시스템과 네트워크를 필요로 하는 제품 증가

4. Creating Task

A. Task는 xTaskCreate() API Function을 사용 하여 Create 된다.

xTaskCreate() API Function Prototype

```
portBASE_TYPE xTaskCreate( pdTASK_CODE pvTaskCode,
                           const signed char * const pcName,
                           unsigned short usStackDepth,
                           void *pvParameters,
                           unsigned portBASE_TYPE uxPriority,
                           xTaskHandle *pxCreatedTask
                         );
```

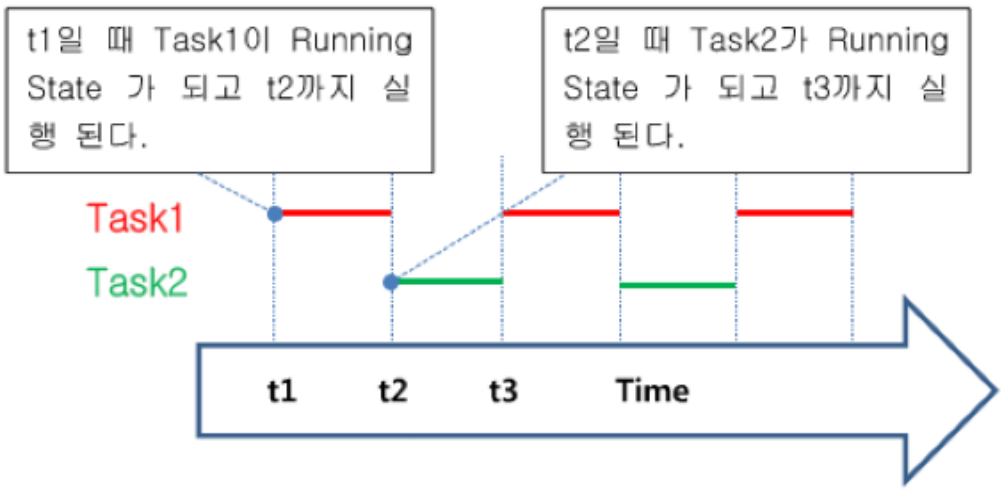
Parameter Name/ Returned value	Description
pvTaskCode	Task Function의 Pointer(Function Name)
pcName	Task Name, 개발자의 Debugging 목적(읽기 쉽게 하기 위한 목적)으로 만 이용 된다.
usStackDepth	각 Task는 자신의 Stack를 갖는다. usStackDepth는 이 Task가 갖는 Stack의 크기(Word)를 지정 한다.
pvParameter	pvParameter에 Assigned Value 가 Task에 Pass 된다.
uxPriority	Task의 Priority
pxCreatedTask	Task Handle, Task의 Priority를 바꾸거나 Task를 Delete 하는 등 Task를 참조 할 필요가 있는 경우 사용 된다. Application에서 사용하지 않는 경우에는 NULL로 설정 한다.
Returned value	2가지 가능한 값을 갖는다. pdTRUE: Task가 성공적으로 Create 된 경우 pdFALSE: Task가 성공적으로 Create 되지 못한 경우

task 를 사용하기 위해 task 를 만드는 함수 부분이다. 위에 설정은 아래 사진을 보면은 알수있다.

```c
#include "HL_sys_common.h"
#include "HL_gio.h"
#include "HL_sys_core.h"
#include "FreeRTOS.h"
#include "os_task.h"
#include "os_semphr.h"
#include "FreeRTOSConfig.h"
#include"stdio.h"
#include "HL_can.h"
#include "HL_esm.h"
#include "HL_sci.h"
#include "string.h"
xTaskHandle xTask1Handle;
xTaskHandle xTask2Handle;
xTaskHandle xTask3Handle;
QueueHandle_t mutex;
void XTask1(void *pvparameters)
{

    gioSetBit(gioPORTA,0,1);
}
void XTask2(void *pvparameters)
{
   gioSetBit(gioPORTA,1,1);
   vTaskDelay(1000);

}
void XTask2(void *pvparameters)
{
   gioSetBit(gioPORTA,2,1);
   vTaskDelay(1500);
}
void gioNotification(gioPORT_t *port, uint32 bit)
{
    gioToggleBit(gioPORTA,3);
}

void
int main(void)
{
gioInit();
gioSetDirection(gioPORTA,0xffffffff);
gioEnableNotification(gioPORTA,3);
sciInit();
xTaskCreate(XTask1,"LED",configMINIMAL_STACK_SIZE,NULL,2,&xTaskHandle)=pdTRUE);
xTaskCreate(XTask2,"LED",configMINIMAL_STACK_SIZE,NULL,1,&xTaskHandle)=pdTRUE);
xTaskCreate(XTask3,"LED",configMINIMAL_STACK_SIZE,NULL,1,&xTaskHandle)=pdTRUE);
```

```c
#define configUSE_PREEMPTION          1
#define configUSE_PORT_OPTIMISED_TASK_SELECTION 1
#define configUSE_FPU                           1
#define configUSE_IDLE_HOOK           0
#define configUSE_TICK_HOOK           0
#define configUSE_TRACE_FACILITY      0
#define configUSE_16_BIT_TICKS        0
#define configCPU_CLOCK_HZ            ( ( unsigned portLONG ) 75000000 ) /* Timer clock. */
#define configTICK_RATE_HZ            ( ( TickType_t ) 1000 )
#define configMAX_PRIORITIES          ( 5 )
#define configMINIMAL_STACK_SIZE      ( ( unsigned portSHORT ) 128 )
#define configTOTAL_HEAP_SIZE         ( ( size_t ) 8192 )
#define configMAX_TASK_NAME_LEN       ( 16 )
#define configIDLE_SHOULD_YIELD       1
#define configGENERATE_RUN_TIME_STATS 0
#define configUSE_MALLOC_FAILED_HOOK  0

/* USER CODE BEGIN (1) */
/* USER CODE END */
```

위에 값들은 HALCOGEN 에서 설정한 값들이고 FREERTOSconfig.h 에서 찾을수 있다.



위에 사진을 보면 알듯이 RTOS 의 MULTITASKING 은 TASK 가 한번에 동작하는게 아니라 한 TASK 가 끝나고 아주 작은 시간의 텀ㅇ을 두고 하지만 사람에 눈에는 동시에 동작하는 것처럼 보인다.

Task Priorities(우선 순위)

xTaskCreate() AI 함수에 의하여 Task 가 Creation 될 때 uxPriority

Parameter 값에 의하여 Priority 값이 초기화 된다.

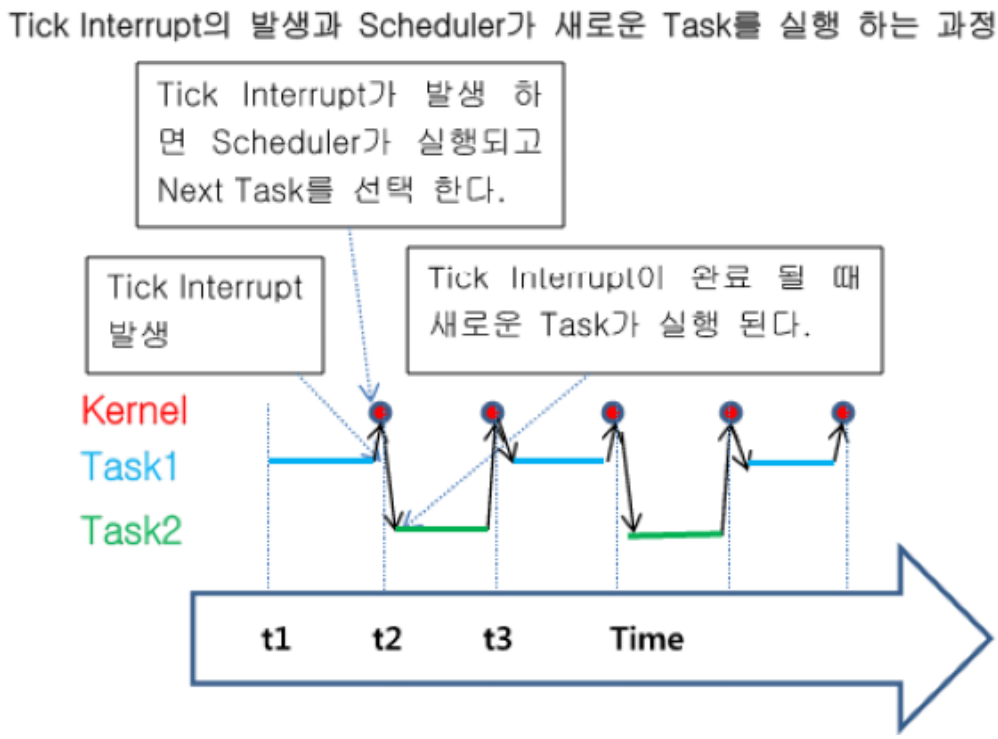
Scheduler 가 Start 된 다음에 cTaskPrioritySet() API 함수에 의하여

Priority 를 변경 할 수 있다.

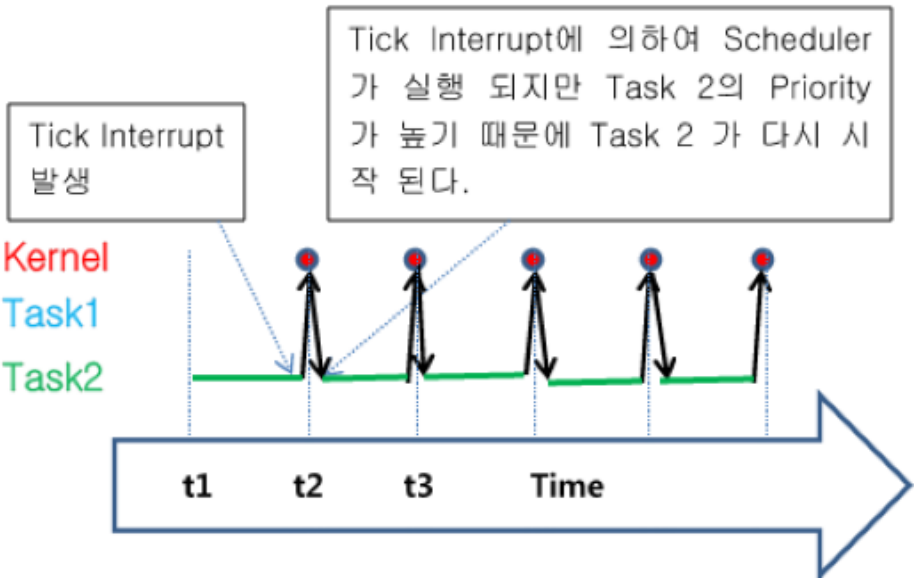
Priority 의 최대값은 FreeRTOSConfig.h 파일의 configMAX_PRIORITIES 값에 의하여 설정 된다. 그러나, 너무 큰 configMAX_PRIORITIES 값은 보다 큰 RAM 용량을 필요로 하기 때문에 Task 에 필요한 Priority 값을 설정할 수 있는 정도 값으로 설정 하여야 한다.

```
/* Map standard tasks.h API functions to the MPU equivalents. */
#define xTaskCreate                             MPU_xTaskCreate
#define xTaskCreateStatic                       MPU_xTaskCreateStatic
#define xTaskCreateRestricted                   MPU_xTaskCreateRestricted
#define vTaskAllocateMPURegions                 MPU_vTaskAllocateMPURegions
#define vTaskDelete                             MPU_vTaskDelete
#define vTaskDelay                              MPU_vTaskDelay
#define vTaskDelayUntil                         MPU_vTaskDelayUntil
#define xTaskAbortDelay                         MPU_xTaskAbortDelay
#define uxTaskPriorityGet                       MPU_uxTaskPriorityGet
#define eTaskGetState                           MPU_eTaskGetState
#define vTaskGetInfo                            MPU_vTaskGetInfo
#define vTaskPrioritySet                        MPU_vTaskPrioritySet
#define vTaskSuspend                            MPU_vTaskSuspend
#define vTaskResume                             MPU_vTaskResume
#define vTaskSuspendAll                         MPU_vTaskSuspendAll
#define xTaskResumeAll                          MPU_xTaskResumeAll
#define xTaskGetTickCount                       MPU_xTaskGetTickCount
#define uxTaskGetNumberOfTasks                  MPU_uxTaskGetNumberOfTasks
#define pcTaskGetName                           MPU_pcTaskGetName
#define xTaskGetHandle                          MPU_xTaskGetHandle
#define uxTaskGetStackHighWaterMark             MPU_uxTaskGetStackHighWaterMark
#define vTaskSetApplicationTaskTag              MPU_vTaskSetApplicationTaskTag
#define xTaskGetApplicationTaskTag              MPU_xTaskGetApplicationTaskTag
#define vTaskSetThreadLocalStoragePointer       MPU_vTaskSetThreadLocalStoragePointer
#define pvTaskGetThreadLocalStoragePointer      MPU_pvTaskGetThreadLocalStoragePointer
#define xTaskCallApplicationTaskHook            MPU_xTaskCallApplicationTaskHook
#define xTaskGetIdleTaskHandle                  MPU_xTaskGetIdleTaskHandle
#define uxTaskGetSystemState                    MPU_uxTaskGetSystemState
#define vTaskList                               MPU_vTaskList
#define vTaskGetRunTimeStats                    MPU_vTaskGetRunTimeStats
#define xTaskGenericNotify                      MPU_xTaskGenericNotify
#define xTaskNotifyWait                         MPU_xTaskNotifyWait
#define ulTaskNotifyTake                        MPU_ulTaskNotifyTake
#define xTaskNotifyStateClear                   MPU_xTaskNotifyStateClear
```

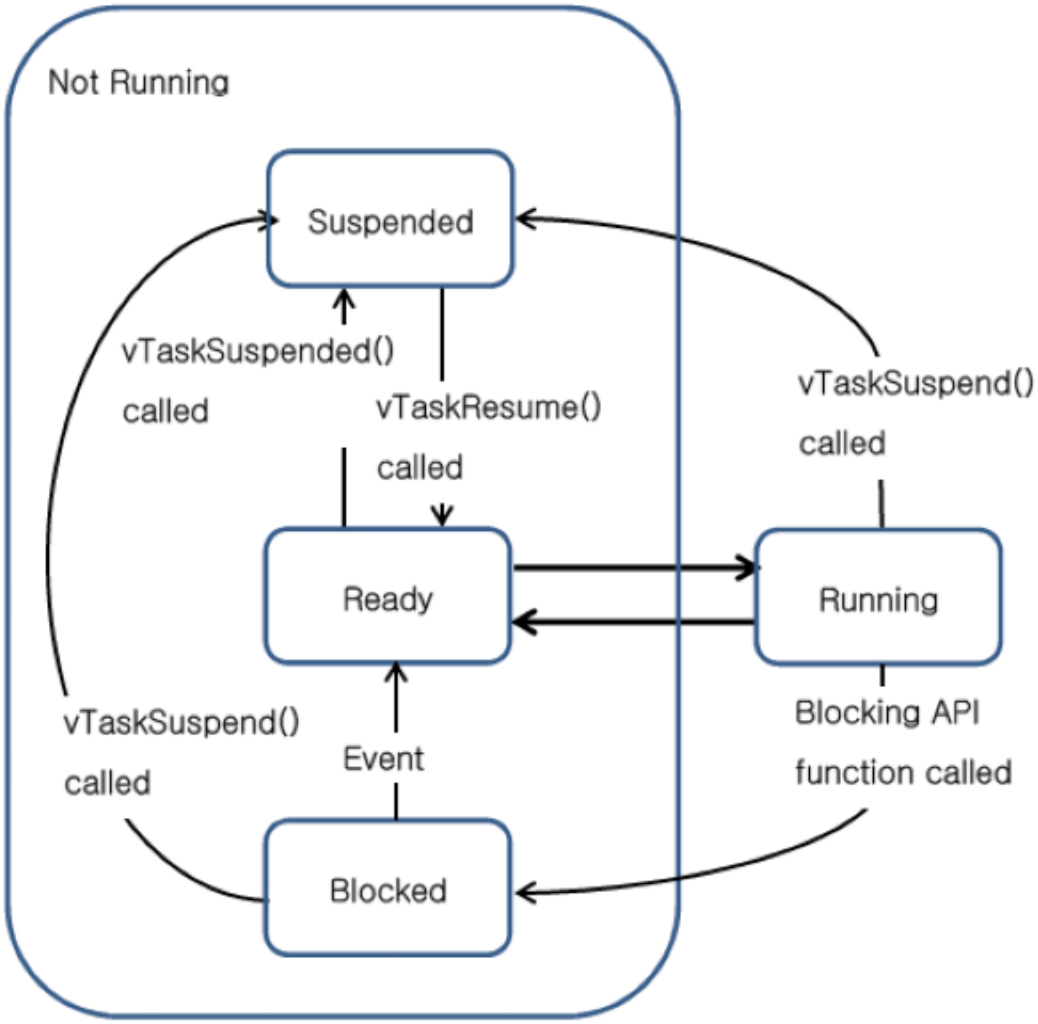
API 를 사용할라면 os_mpu_wrappers.h 에서 찾아서 쓸수 있다.



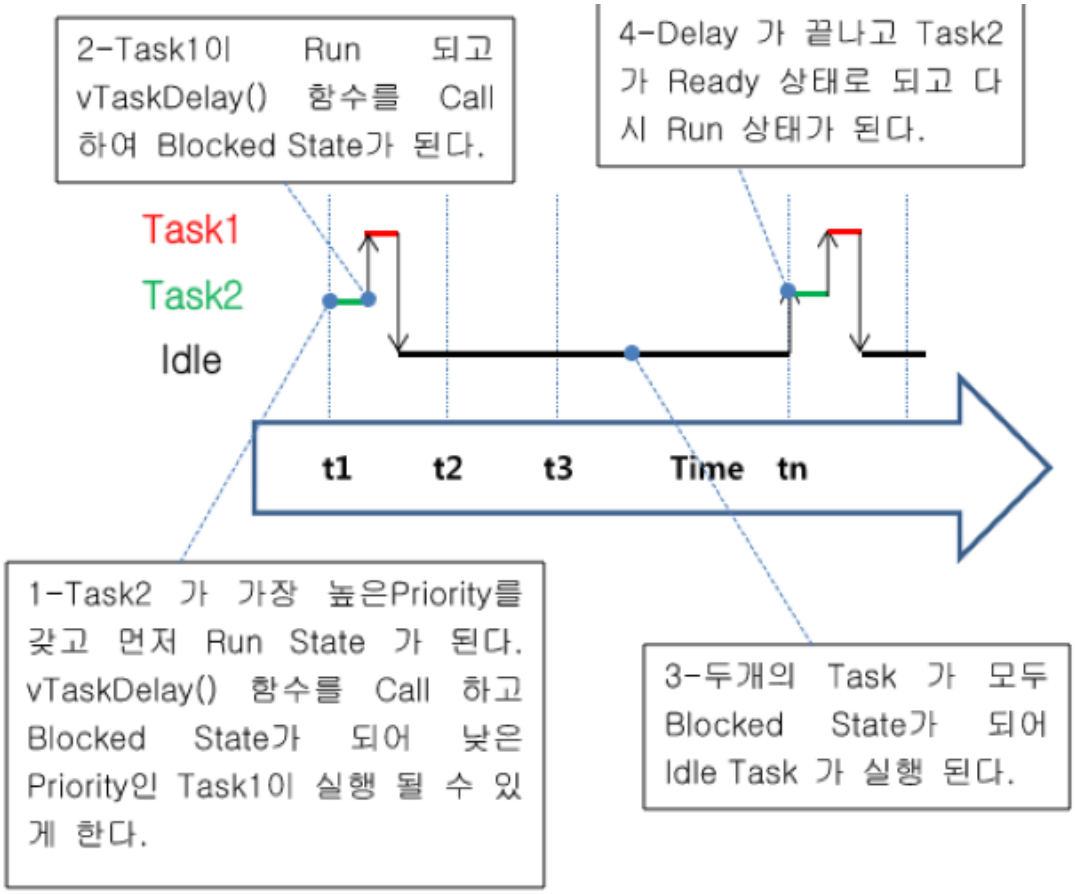
위에 사진은 각 task 마다 우선순위가 같을 때의 interrupt 동작



우선순위를 한 task 만 너무 높으면 우선순위가 낮은 task 는 동작을 하지 않는다.



Task 는 두 가지 state 로 정리할 수 있는데 no Running 상태는 task 가 실행을 하지않는 상태이면서 다음 Task 가 동작을 하기 위한 대기상태이라고 한다.



7. Idle Task와 Idle Task Hook

하나의 Task를 생성하여 실행 하고 있는 경우 이 Task가 Blocked State에 남아 있다면 이 Task는 Scheduler에 의하여 선택 될 수 없다.
그러나 Processor는 항상 어떠한 일이든 실행 하고 있어야 하기 때문에 최소한 하나의 Task는 언제나 실행 상태에 있어야 한다.
그러므로 vTaskStartScheduler() API가 Call 될 때 자동적으로 Idle Task가 Create 된다.
만약 다른 Task가 Ready State가 되면 바로 Ready State에 있는 Task가 실행 되어야 하기 때문에 Idle Task는 가장 낮은 Priority를 갖는다.

A. Idle Task Hook Function

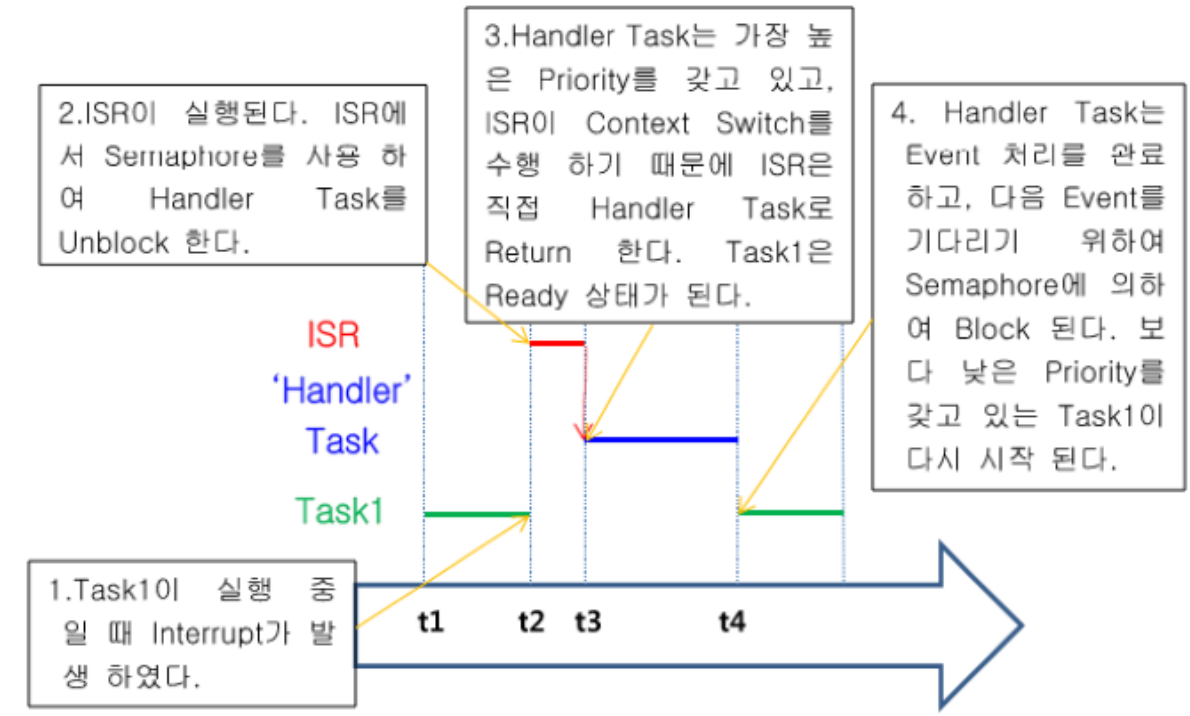
Idle Task 가 실행 될 때 Idle Task에 의하여 실행 된다.

i. Idle Task HooK의 일반적인 사용 분야

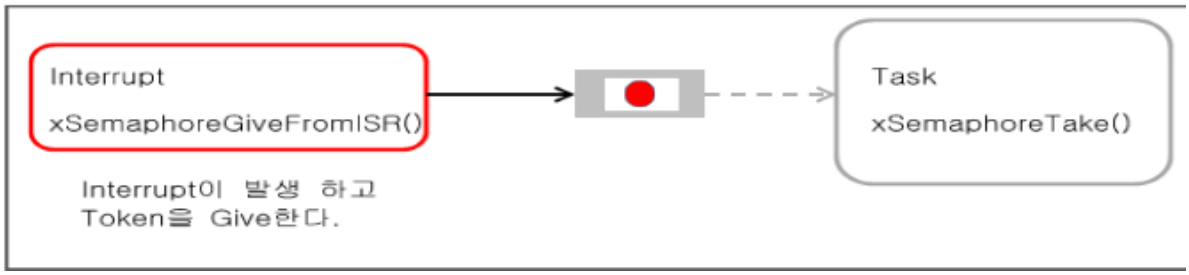
1. Low Priority로 Background 에서 계속적으로 실행 된다.
2. Idle Task의 Processing Time을 측정 하여 Processing Time이 어느 정도 여유 있는 지 알 수 있다.
3. Application Processing 이 없는 경우 자동적으로 Processor를 Low Power Mode로 놓을 수 있다.

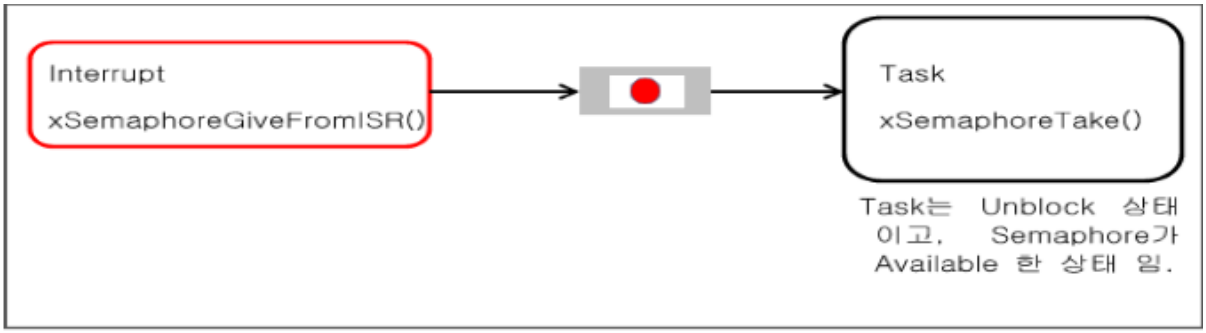
RTOS_interrupt

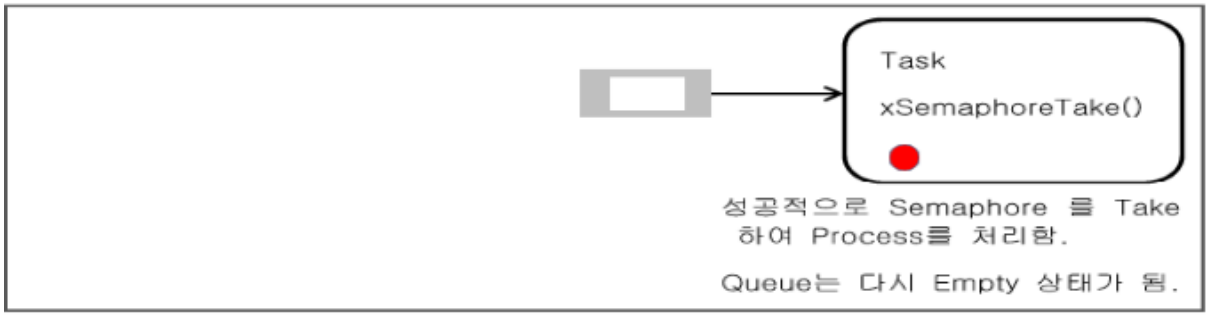
Embedded Real Time 시스템은 주변 장치로부터 발생 하는 Event 에 실시간으로 응답 하여야 하는 응용 분야에 많이 이용 된다. 응용 분야에 따라서는 여러 개의 Interrupt Source로부터 발생 하는 Event를 실시간으로 처리 하여야 하고, 각각의 Interrupt 처리는 서로 다른 처리 시간과 속도를 필요로 하기 때문에 최적의 Event 처리를 위한 전략이 필요 하다

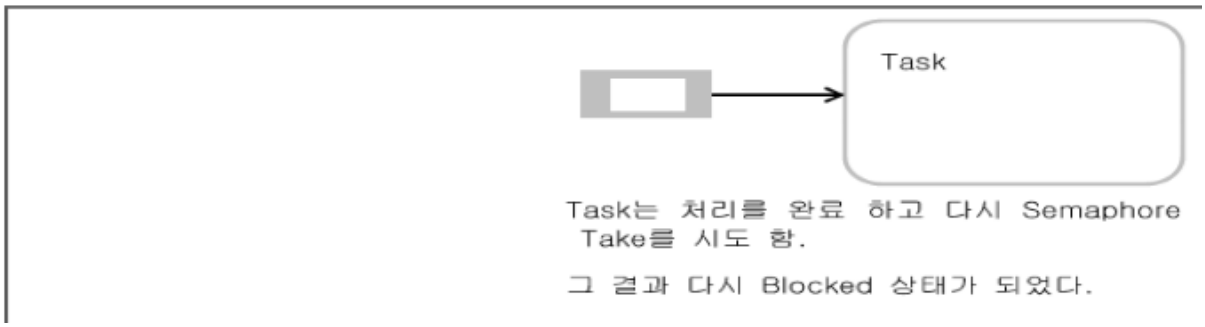


Binary Semaphore를 이용 한 Interrupt와 Task 동기









위에 사진은 binary semaphore 에 동작을 그림으로 상세히 설명하는 부분이다.

B. vSemaphoreCreateBinary() API Function

Semaphore는 사용되기 전에 Create되어야 한다.

xSemaphoreCreateBinary() API Function의 Prototype

```
void xSemaphoreCreateBinary ( xSemaphoreHandle xSemaphore);
```

Parameter Name	Description
xSemaphore	Semaphore Handle

xSemaphoreTake() API Function

'Taking' Semaphore의 의미는 Semaphore를 'Receive' 또는 'Obtain' 의 의미 이다. 는 사용되기 전에 Create되어야 한다.

xSemaphoreTake()는 Interrupt Service Routine에서 사용 할 수 없다.

xSemaphoreTake() API Function의 Prototype

```
portBase_TYPE xSemaphoreTake ( xSemaphoreHandle
xSemaphore, portTickType xTickToWait );
```

Parameter Name/ Returned value	Description
xSemaphore	Semaphore Handle
xTickToWait	Task가 Blocked State에서 Semaphore를 사용 할 수 있을 때 까지 기다리는 최대 Tick 수 만약 xTicksToWait가 0 이면 Semaphore를 회득 할 수 없으면 xSemaphoreTake()는 즉시 Return 된다. FreeRTOSConfig.h에서 INCLUDE_vTaskSuspend 가 1로 Set되고, xTicksToWait 가 portMAX_DELAY로 설정 된 경우 Task는 Timing out 없이 무한이 기다린다.
Returned value	2가지 가능한 값을 갖는다. pdPASS: xSemaphoreTake()가 성공적으로 Semaphore를 획득 한 경우 pdFALSE: Semaphpre를 획득 할 수 없는 경우

Binary semaphore 는 interrupt 의 발생 주기가 Event 처리 속도보다 늦을때는 상관이 없지만 만약 이 반대의 상황이 되면 Event 의 정보는 상실 된다. 그래서 이 부분을 보와하기 위해 counting semaphor,mutex 가 이부분을 보완해준다.

Resouce Manigement

Multitasking System에서 한 Task가 어떤 Resource를 사용 하고 있는 도중에 Running State에서 벗어 나는 일이 생길 수 있고, 이 상태에서 다른 Task나 Interrupt가 동일한 Resource를 사용 하려고 시도 할 수 있다. 이 경우 Data 가 충돌 하거나 손상 될 수 있다.

Mutual Exclusion

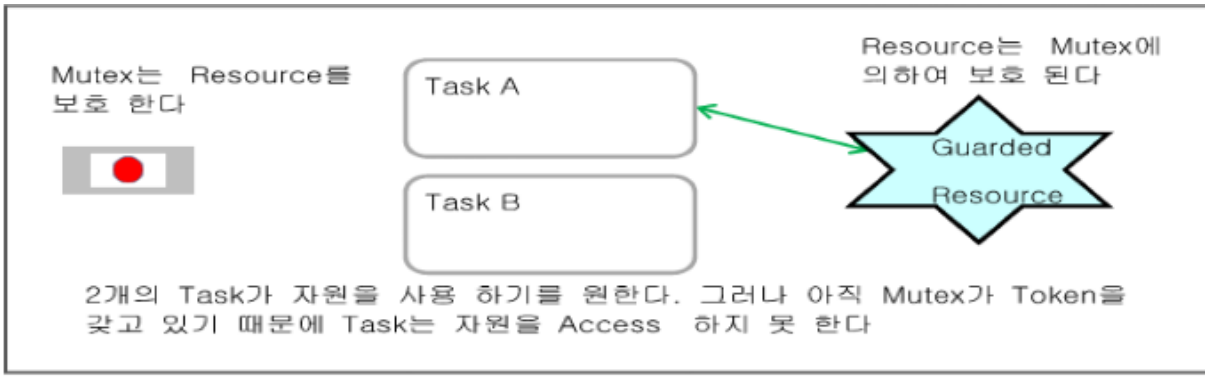
Task 사이에 공유되는 자원과 Task와 Interrupt 사이에 공유되는 자원은 'Mutual Exclusion' 기술로 관리 되어 야 한다. 한 Task가 이용 하기 시작한 자원(Resource)은 이 Task의 사용이 종료 되어 Returned 될 때까지 배타적으로 사용 되어야 한다.

FreeRTOS는 Mutual Exclusion을 구현 하는 방법을 제공 한다. 그러나 최상의 방법은 하나의 자원은 하나의 Task만 사용 하도록 설계하는 것이 최선 이다.

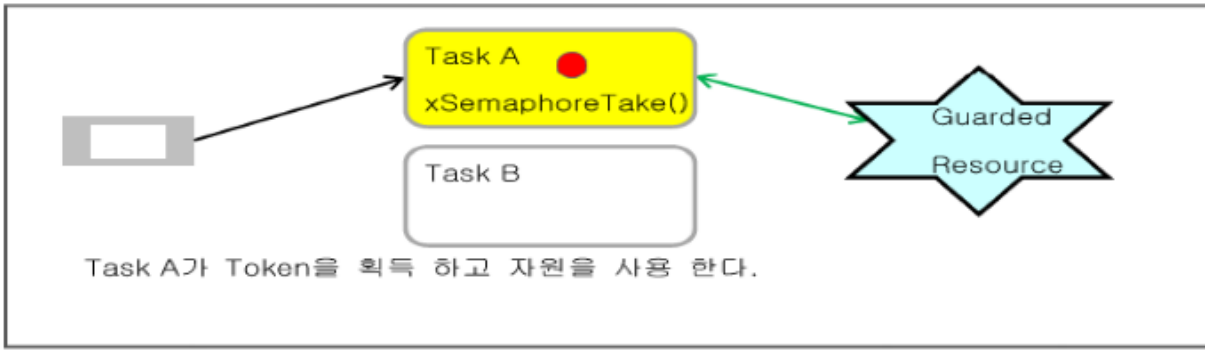
Mutual Exclusion

Task 사이에 공유되는 자원과 Task와 Interrupt 사이에 공유되는 자원은 'Mutual Exclusion' 기술로 관리 되어 야 한다. 한 Task가 이용 하기 시작한 자원(Resource)은 이 Task의 사용이 종료 되어 Returned 될 때까지 배타적으로 사용 되어야 한다.

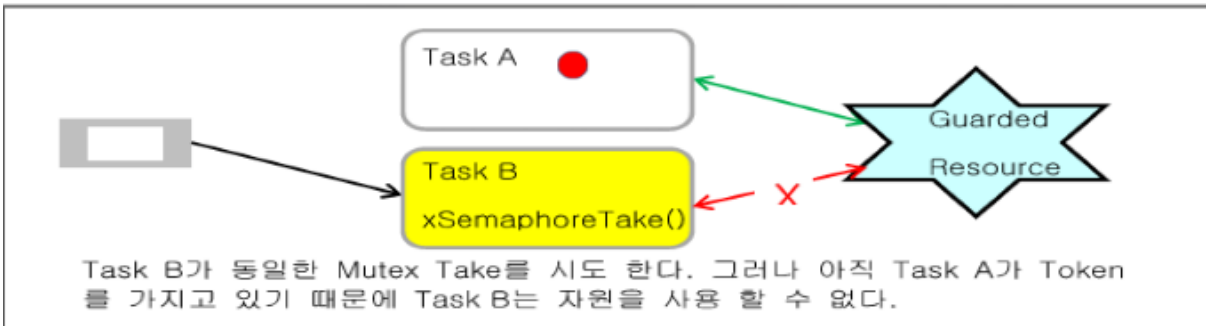
FreeRTOS는 Mutual Exclusion을 구현 하는 방법을 제공 한다. 그러나 최상의 방법은 하나의 자원은 하나의 Task만 사용 하도록 설계하는 것이 최선 이다.



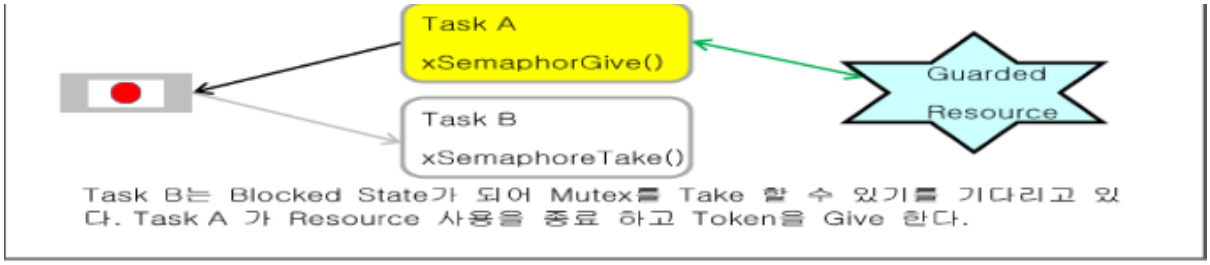
2개의 Task가 자원을 사용 하기를 원한다. 그러나 아직 Mutex가 Token을 갖고 있기 때문에 Task는 자원을 Access 하지 못 한다



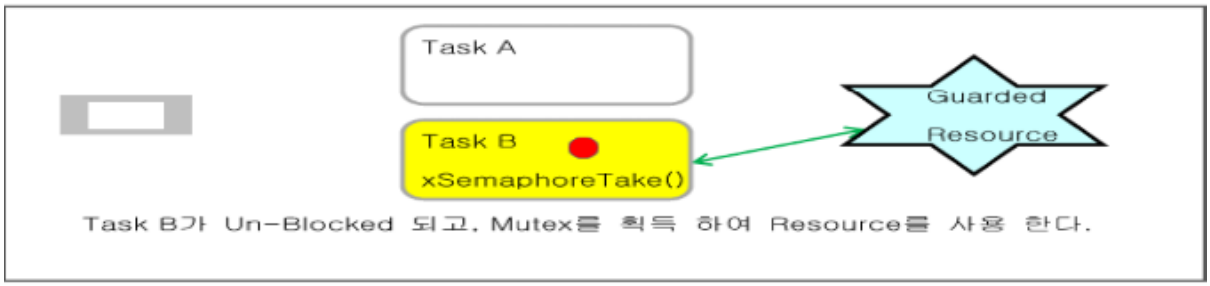
Task A가 Token을 획득 하고 자원을 사용 한다.



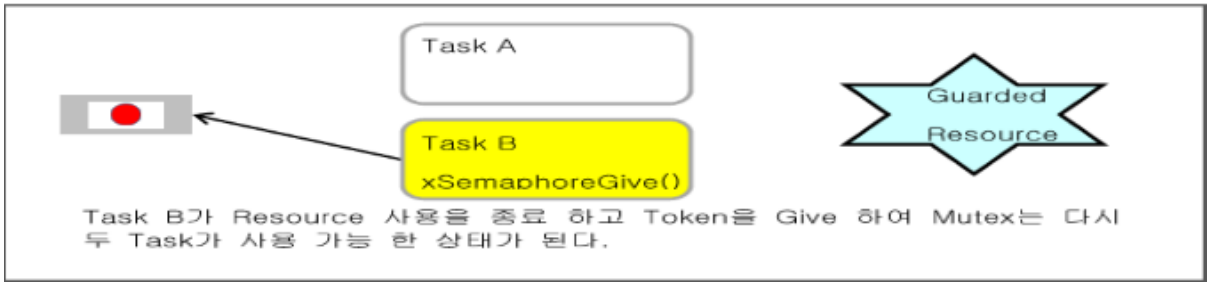
Task B가 동일한 Mutex Take를 시도 한다. 그러나 아직 Task A가 Token를 가지고 있기 때문에 Task B는 자원을 사용 할 수 없다.



Task B는 Blocked State가 되어 Mutex를 Take 할 수 있기를 기다리고 있다. Task A 가 Resource 사용을 종료 하고 Token을 Give 한다.

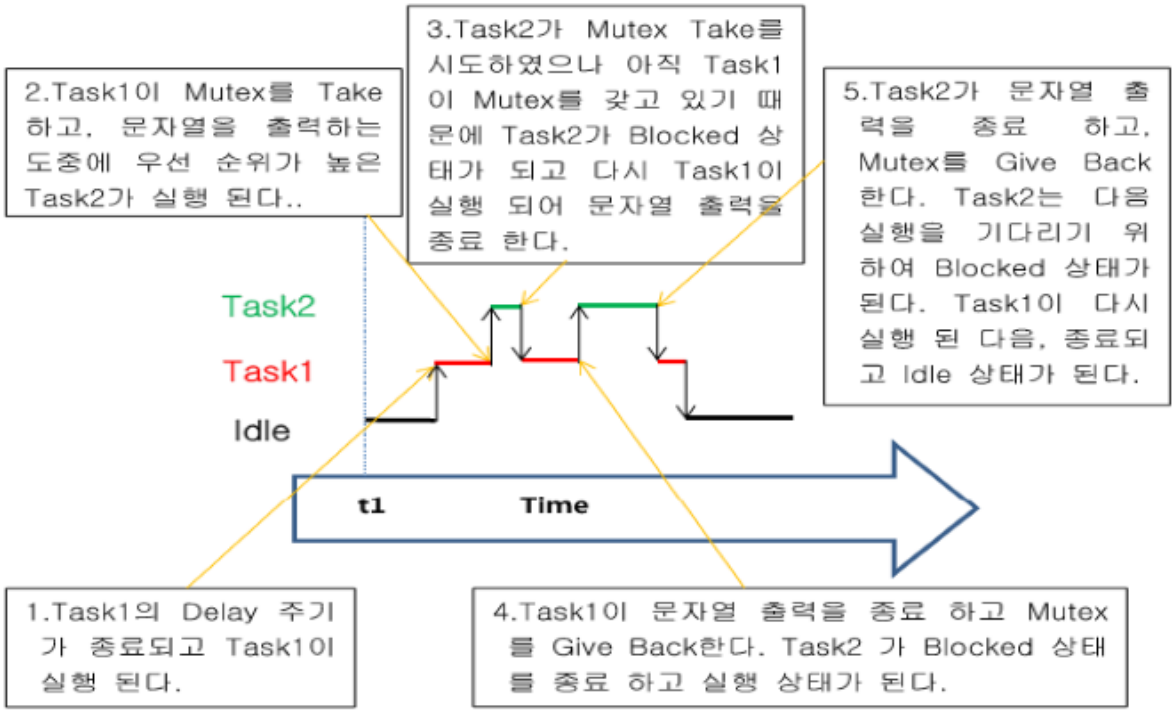


Task B가 Un-Blocked 되고, Mutex를 획득 하여 Resource를 사용 한다.



Task B가 Resource 사용을 종료 하고 Token을 Give 하여 Mutex는 다시 두 Task가 사용 가능 한 상태가 된다.

Mutex를 이용한 자원 관리 예(Task2의 Priority 가 Task1보다 높은 경우)



<CountingSemaphore>

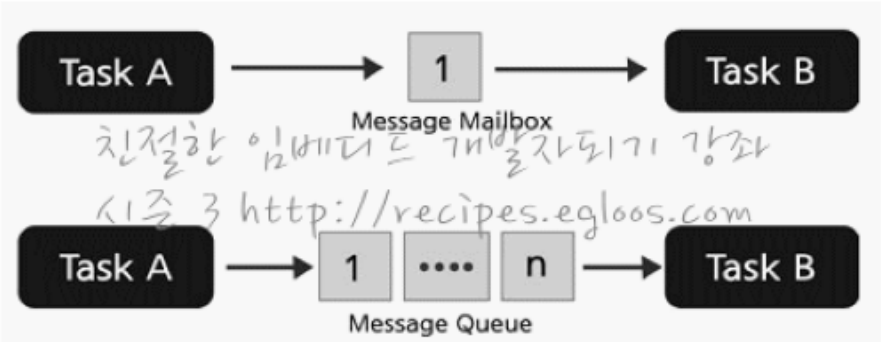
ii. Counting Semaphore를 Resource Management에 사용 하는 경우
이 경우 Semaphore의 Count Value는 사용 가능한 자원의 수를 표시 한다. Resource의 사용권을 획득 하기 위하여 Task는 먼저 Semaphore를 'Take' 하여야 한다. Task가 Semaphore를 'Take' 할 때 마다 Semaphore의 Count Value는 1씩 감소 하고 Count Value가 0가 되면 사용 할 수 있는 Resource가 없게 된다.
Task 의 Resource의 사용이 완료 될 경우 Task는 Semaphore를 'Give' 하고 Count Value는 1 증가 한다. 그러면 다른 Task가 자원을 하나 더 사용 할 수 있게 된다.
Counting Semaphore를 자원 관리에 사용 하는 경우 Count Value의 초기 값을 사용 가능한 자원의 개 수로 하여 Counting Semaphore를 Create 한다.

H. xSemaphoreCreateCounting() API 함수
FreeRTOS Semaphore의 Handle은 xSemaphoreHandle 형태의 변수에 저장 된다.
Semaphore는 사용 하기 전에 Create 되어야 한다. Counting Semaphore는 xSemaphoreCreateCounting() API 함수에 의하여 Create 된다.

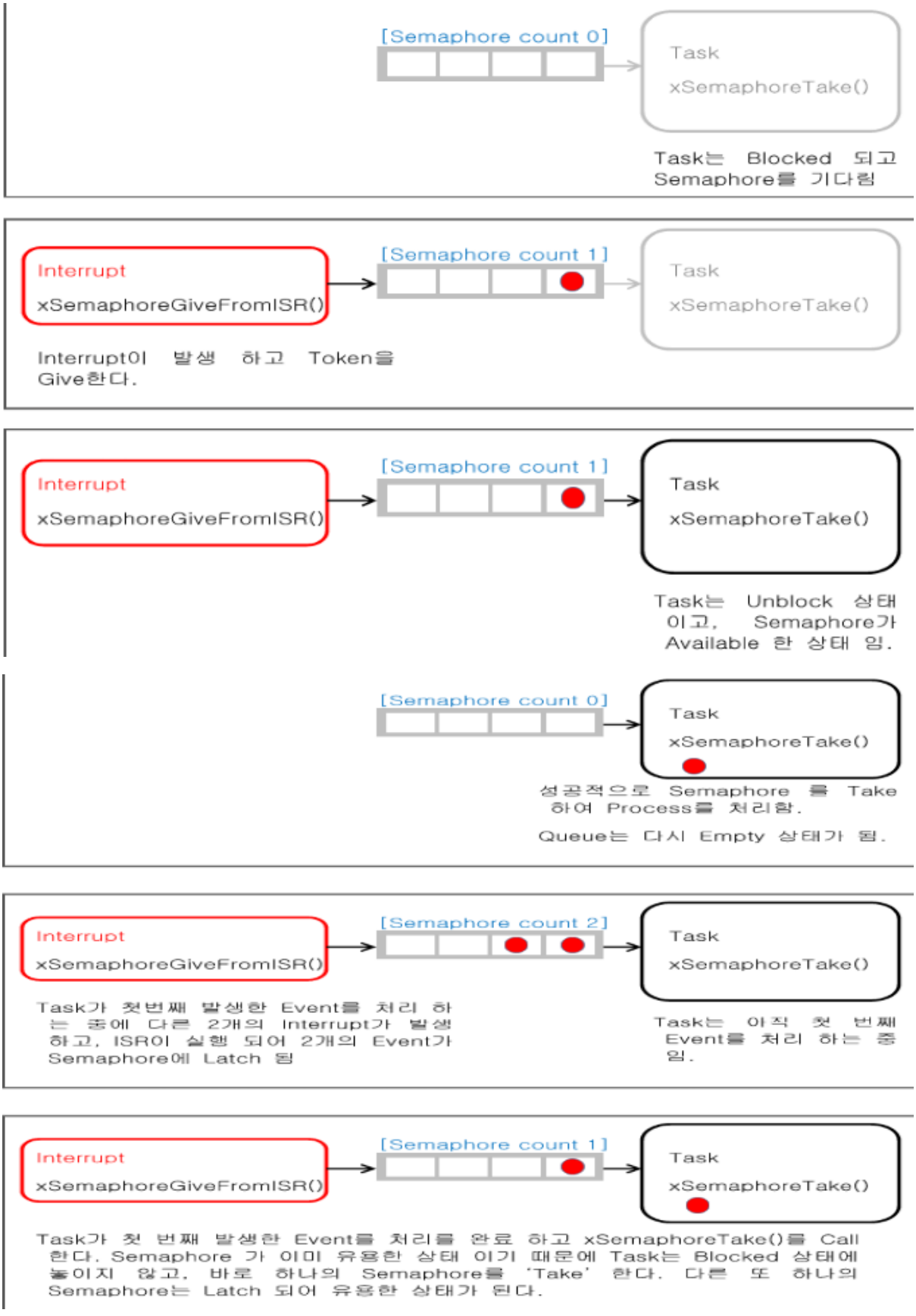
xSemaphoreCreateCounting() API 함수의 Prototype

xSemaphoreHandle xSemaphoreCreateCounting (unsigned portBASE_TYPE uxMaxCount, unsigned portBASE_TYPE uxInitialCount)

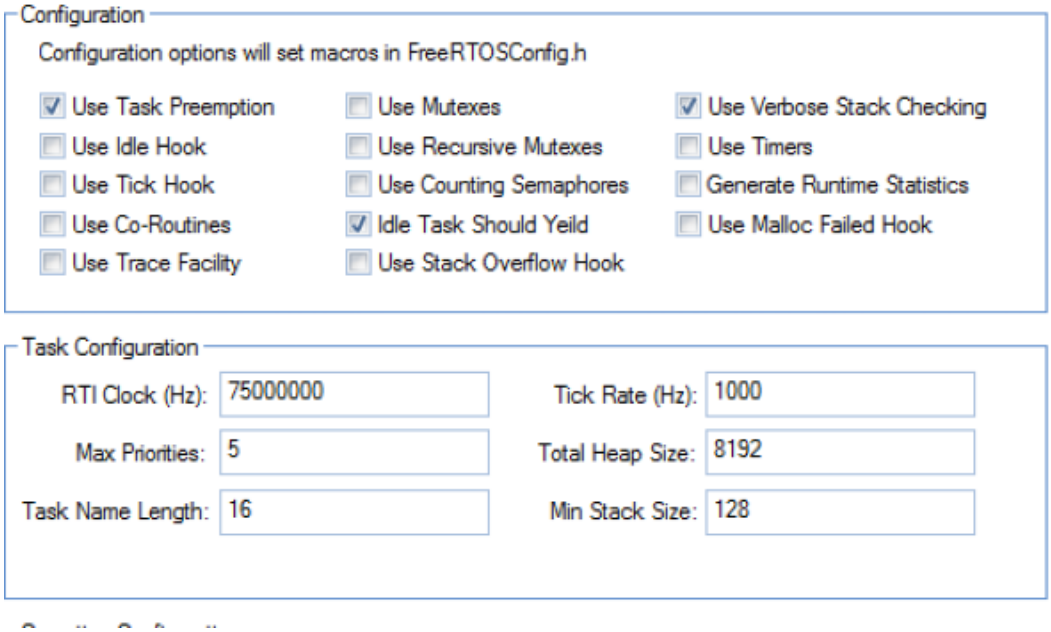
Parameter Name/ Returned value	Description
uxMaxCount	Count 할 Semaphore의 최대 값. Semaphore가 Count 나 Events Latch로 사용 될 경우 uxMaxCount는 Latch 되는 Event의 최대 값
uxInitialCount	Semaphore 가 Create 될 때 Count Value의 초기 값 Semaphore가 Event counting에 사용 될 경우 아직 Event 가 발생 하지 않은 경우에 0로 설정 한다. Semaphore가 자원 관리에 사용 될 경우 사용 가능한 자원의 최대수(uxMaxCount)로 설정 한다.
Returned value	NULL이 Return 되는 경우에는 Semaphore가 Create 되지 못 한 경우 이다. Non-NULL이 Return 된 경우 이 값이 Create 된 Semaphore의 Handle로 저장 된다.




```
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

/* Mutexes */
#define configUSE_MUTEXES                0
#define configUSE_RECURSIVE_MUTEXES      0

/* Semaphores */
#define configUSE_COUNTING_SEMAPHORES    0

/* Timers */
```



<counting semaphores 동작 과정>



<halcogen os 설정 목록>

```
#define configSUPPORT_STATIC_ALLOCATION           1
#define configSUPPORT_DYNAMIC_ALLOCATION          1

#define configNUM_THREAD_LOCAL_STORAGE_POINTERS 1
#define configUSE_TICKLESS_IDLE                   1

/* USER CODE BEGIN (2) */
/* USER CODE END */

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES             0
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

/* Mutexes */
#define configUSE_MUTEXES                 0
#define configUSE_RECURSIVE_MUTEXES       0

/* Semaphores */
#define configUSE_COUNTING_SEMAPHORES     1
```

맨 위에 보면 configSUPPORT_STATIC_ALLOCATION:정적 메모리 사용 가능

configSUPPORT_o DYNAMIC_ALLOCATION:동적 메모리 메모리 사용 가능

ConfigUSE_COUNTING_SEMAPHORE1 로 설정(HALCOGEN 에서 체크)

configSUPPORT_DYNAMIC_ALLOCATION 이 1로 설정되거나 정의되지 않은 상태에서 사용할 수있는 다음 API 함수는 동적으로 할당 된 RAM을 사용하여 RTOS 객체를 만듭니다.

- xTaskCreate ()
- xQueueCreate ()
- xTimerCreate ()
- xEventGroupCreate ()
- xSemaphoreCreateBinary ()
- xSemaphoreCreateCounting ()
- xSemaphoreCreateMutex ()
- xSemaphoreCreateRecursiveMutex ()

<동적 메모리 사용하는 API>

configSUPPORT_STATIC_ALLOCATION 이 1로 설정된 경우 사용할 수있는 다음 API 함수를 사용하면 응용 프로그램 작성자가 제공 한 메모리를 사용하여 RTOS 객체를 만들 수 있습니다. 메모리를 제공하기 위해 응용 프로그램 작성자는 적절한 객체 유형의 변수를 선언하고 변수의 주소를 RTOS API 함수에 전달하기 만하면됩니다. 함수 사용법을 보여주기 위해 StaticAllocation.c 표준 데모 / 테스트 작업이 제공됩니다.

- xTaskCreateStatic ()
- xQueueCreateStatic ()
- xTimerCreateStatic ()
- xEventGroupCreateStatic ()
- xSemaphoreCreateBinaryStatic ()
- xSemaphoreCreateCountingStatic ()
- xSemaphoreCreateMutexStatic ()
- xSemaphoreCreateRecursiveMutexStatic ()

<정적 메모리 사용하는 API>

위에 사진있는 것처럼 동적,정적,Counting_Semaphore 를 모두 1 로 설정하였지만 저글링 100 부대가 쏟아져 나오듯이 에러가 엄청 나왔다.

그 중에 unresolved symbol vApplicationGetIdleTaskMemory, first referenced in ./source/os_tasks.obj 라는 에러 문구를 보고 vApplicationGetIdleTaskMemory 를 정의를 해주어야 한다.

```
#if( configSUPPORT_STATIC_ALLOCATION == 1 )
    extern void vApplicationGetIdleTaskMemory( StaticTask_t **ppxIdleTaskTCBBuffer, StackType_t **ppxIdleTaskStackBuffer, uint32_t *pulIdleTaskStackSize );
#endif


void vApplicationGetIdleTaskMemory(StaticTask_t **ppxIdleTa
{
    *ppxIdleTaskTCBBuffer=&buf;
    *ppxIdleTaskStackBuffer=&buf1[0];
    *pulIdleTaskStackSize=size;
}
```

Static stackType_t buf;

Static staticType_t buf1[size];

#define size configMINIMAL_STACK_SIZE*100

```
SemaphoreHandle_t semaphore;
SemaphoreHandle_t counting_semaphore;
 void vApplicationGetIdleTaskMemory(StaticTask_t **ppxI
void sciDisplay(sciBASE_t *sci, uint8 *test, uint32 len
void vTask1(void *p)
{
     int i;
     char aa[256];
     printf("static_semaphore sucess\n");
     for(i=0;i<100;i++)
     {
     sprintf(aa,"increasing counter=%x\n",i);
     sciDisplay(sciREG1,(uint8 *)aa,strlen(aa));
}
     xSemaphoreTake(counting_semaphore,portMAX_DELAY);
     xSemaphoreGive(counting_semaphore);
     vTaskDelay(500);
}
void vTask2(void *p)
{
     int j;
     char aa[256];
     printf("DYNAMIC_MEEMORY\n");
     for(j=0;j<100;j++)
     {
          sprintf(aa,"increasing counter=%x\n",j+1);
          sciDisplay(sciREG1,(uint8 *)aa,strlen(aa));

     }
      xSemaphoreTake(semaphore,portMAX_DELAY);
          xSemaphoreGive(semaphore);
          vTaskDelay(1000);
}
```

```
  sciInit();
  gioInit();
  vSemaphoreCreateBinary(semaphore);
ounting_semaphore=xSemaphoreCreateCounting(10,0);
   if(xTaskCreateStatic(vTask1,"STATIC_MEMORY",configMINIMAL_STACK_SIZE,NULL,1,&buffer[0],&TCBbuffer)!=NULL)
   {
      while(1)
         ;
   }
   if(xTaskCreate(vTask2,"DYNAMIC_MEMORY",configMINIMAL_STACK_SIZE,NULL,1,&xTask1Handle)!=pdTRUE)
   {
      while(1)
         ;
   }
  return 0;
```

```
TaskHandle_t xTaskCreateStatic( TaskFunction_t pxTaskCode,
                                const char * const pcName,
                                const uint32_t ulStackDepth,
                                void * const pvParameters,
                                UBaseType_t uxPriority,
                                StackType_t * const puxStackBuffer,
                                StaticTask_t * const pxTaskBuffer ) /*li
{
TCB_t *pxNewTCB;
```