

TI DSP, MCU 및

Xilinx Zynq FPGA 프로그램 전문가 과정

zero_bird Portfolio

최초 작성일: 2018. 06. 13

최종 작성일: 2018. 06. 00

작성자: 은태영 (zero_bird@naver.com)

1.3 Queue API

1.3.1 Queue 생성 및 제거

1.3.1.1 vQueueCreate() : 메모리 동적 할당 Queue 생성

```
#include "FreeRTOS.h"
#include "queue.h"

QueueHandle_t xQueueCreate ( UBaseType_t uxQueueLength, UBaseType_t uxItemSize );
```

새로운 Queue 를 만들고 Queue 를 참조 할 수 있는 핸들을 반환한다.

각 Queue 는 Queue 의 상태를 유지하고, Queue 내부 항목(Queue 저장 영역)을 유지하는데 RAM 이 필요하다.

xQueueCreate()를 사용하여 Queue 를 생성하면, RAM 이 FreeRTOS 힙에서 자동으로 할당한다. xQueueCreateStatic()는 Queue 를 만들때 RAM 은 응용 프로그램 작성자가 제공한다. 더 많은 수의 매개 변수를 가져오지만, 컴파일 시간에 RAM 을 정적으로 할당할 수 있다. Queue 는 task 간에, 그리고 task 와 인터럽트 간에 데이터를 전달하는데 사용된다. Queue 는 스케줄러가 시작되기 전이나 후에 생성할 수 있다. 이 함수를 사용하려면 FreeRTOSConfig.h 의 configSUPPORT_DYNAMIC_ALLOCATION 를 1 로 설정하거나 정의되지 않은 상태로 두어야한다.

1.3.1.1.1 매개 변수

uxQueueLength : 생성되는 Queue 가 한 번에 보유할 수 있는 최대 항목 수다.

uxItemSize : Queue 에 저장할 수 있는 각 데이터 항목의 크기다.

1.3.1.1.2 반환 값

NULL : Queue 데이터 구조 및 저장 영역을 할당하기 위한 FreeRTOS 의 힙 메모리가 충분하지 않을 때 반환한다.

다른 값은 Queue 가 성공적으로 작성된 경우이다. 반환 값은 만들어진 Queue 를 참조할 수 있는 핸들이다.

1.3.1.1.3 기타

```
// Queue 의 데이터 유형을 정의한다.
typedef struct A_Message {
    char ucMessageID;
    char ucData[ 20 ];
} AMessage;

// Queue 의 매개 변수를 정의한다.
#define QUEUE_LENGTH 5
#define QUEUE_ITEM_SIZE sizeof( AMessage )

int main( void ) {
    QueueHandle_t xQueue;
    // Queue 를 생성하고 반환된 핸들을 xQueue 변수에 저장한다.
    xQueue = xQueueCreate( QUEUE_LENGTH, QUEUE_ITEM_SIZE );

    if( xQueue == NULL ) {
        // Queue 를 만들 수 없다.
    }
    // 나머지 코드가 위치한다.
}
```

1.3.1.2 vQueueCreateStatic() : 메모리 정적 할당 Queue 생성

```
#include "FreeRTOS.h"
#include "queue.h"

QueueHandle_t xQueueCreateStatic ( UBaseType_t uxQueueLength, UBaseType_t uxItemSize
                                   uint8_t * pucQueueStorageBuffer, StaticQueue_t * pxQueueBuffer );
```

Queue 를 만들고 Queue 를 참조할 수 있는 핸들을 반환한다. 각 Queue 에는 Queue 상태를 유지하고 Queue 에 포함된 항목(Queue 저장 영역)을 보유하는데 사용되는 RAM 이 필요하다.

xQueueCreateStatic()을 사용하여 Queue 를 생성하면 응용 프로그램 작성자가 RAM 을 제공하므로, 더 많은 수의 매개 변수가 생성되지만 컴파일 시간에 RAM 을 정적으로 할당할 수 있다.

Queue 는 task 사이에서나 task 인터럽트 데이터를 전달하는데 사용된다. Queue 는 스케줄러가 시작되기 전이나 후에 생성할 수 있다.

이 함수를 사용하려면, FreeRTOSConfig.h 의 configSUPPORT_STATIC_ALLOCATION 를 1 로 설정해야 한다.

1.3.1.2.1 매개 변수

uxQueueLength : 생성된 Queue 가 한번에 보유할 수 있는 최대 항목 수다.

uxItemSize : Queue 에 저장할 수 있는 각 데이터의 항목 크기(바이트)다.

pucQueueStorageBuffer : uxItemSize 가 0 이 아닐 경우, pucQueueStorageBuffer 는 Queue 에 있을 수 있는 최대 항목 수 (uxQueueLength * uxItemSize 바이트) 만큼 큰 uint8_t 배열을 가리켜야 한다. uxItemSize 가 0 이면 데이터가 Queue 저장 영역으로 저장되지 않으므로 pucQueueStorageBuffer 는 NULL 일 수 있다.

pxQueueBuffer : StaticQueue_t 유형의 변수를 가리켜야 하며, Queue 의 데이터 구조를 유지하는데 사용된다.

1.3.1.2.2 반환 값

NULL : pxQueueBuffer 가 NULL 이기 때문에 Queue 생성에 실패하였다.

다른 모든 값은 Queue 가 정상적으로 생성되었으며, 반환된 값은 생성된 Queue 의 핸들이다.

1.3.1.2.3 기타

```
// Queue 는 최대 10 개의 uint64_t 변수를 보유하도록 작성된다.
#define QUEUE_LENGTH 10
#define ITEM_SIZE sizeof( uint64_t )

// Queue 의 데이터 구조를 유지하는데 사용되는 변수다.
static StaticQueue_t xStaticQueue;
// Queue 저장 영역으로 사용되는 배열의 최소 바이트다.
uint8_t ucQueueStorageArea[ QUEUE_LENGTH * ITEM_SIZE ];

void vATask( void *pvParameters ) {
    QueueHandle_t xQueue;

    // 10 개의 uint64_t 값을 포함할 수 있는 Queue 를 만든다.
    xQueue = xQueueCreateStatic( QUEUE_LENGTH, ITEM_SIZE, ucQueueStorageArea, &xStaticQueue );

    // pxQueueBuffer 는 NULL 이 아니므로 xQueue 는 NULL 이 아니어야 한다.
    configASSERT( xQueue );
}
```

1.3.1.3 vQueueDelete() : Queue 삭제

```
#include "FreeRTOS.h"
#include "queue.h"

void vQueueDelete ( TaskHandle_t pxQueueToDelete );
```

xQueueCreate() 또는 xQueueCreateStatic()로 생성된 Queue 를 삭제한다. vQueueDelete()를 이용하여 세마포어를 삭제할 수도 있다.

Queue 는 task 사이에서나 task 인터럽트 데이터를 전달하는데 사용된다. task 는 Queue / 세마포어에 데이터를 보내려고 시도할 때, Queue / 세마포어가 이미 꽉 찼거나 Queue / 세마포어에서 데이터를 수신하려고 시도하는 경우 이를 차단(선택적 시간 제한 사용)하도록 선택할 수 있다. Queue / 세마포어가 현재 차단된 task 에 있으면 삭제를 하지 않아야 한다.

1.3.1.3.1 매개 변수

pxQueueToDelete : 삭제하고자 하는 Queue 나 세마포어의 핸들이다.

1.3.1.3.2 기타

```
// Queue 의 데이터 유형을 정의한다.
typedef struct A_Message {
    char ucMessageID;
    char ucData[ 20 ];
} AMessage;

// Queue 의 매개 변수를 정의한다.
#define QUEUE_LENGTH 5
#define QUEUE_ITEM_SIZE sizeof( AMessage )

int main( void ) {
    QueueHandle_t xQueue;

    // Queue 를 생성하고 반환된 핸들을 xQueue 변수에 저장한다.
    xQueue = xQueueCreate( QUEUE_LENGTH, QUEUE_ITEM_SIZE );

    if( xQueue == NULL ) {
        // Queue 생성을 실패하였다.
    } else {
        // xQueue 를 전달하여 Queue 를 삭제한다.
        vQueueDelete( xQueue );
    }
}
```

1.3.2 Queue 읽기 / 쓰기

1.3.2.1 xQueueSend...() : Queue 에 쓰기

```
#include "FreeRTOS.h"
#include "queue.h"

BaseType_t xQueueSend ( QueueHandle_t xQueue, const void * pvItemToQueue, TickType_t xTicksToWait );

BaseType_t xQueueSendToFront ( QueueHandle_t xQueue, const void * pvItemToQueue, TickType_t xTicksToWait );

BaseType_t xQueueSendToBack ( QueueHandle_t xQueue, const void * pvItemToQueue, TickType_t xTicksToWait );
```

항목을 Queue 의 앞 또는 뒤로 보내거나 쓴다.

xQueueSend()와 xQueueSendToBack()은 동일한 연산을 수행한다. 둘 Queue 의 뒤쪽으로 데이터를 보낸다. xQueueSend()가 원래 버전이었으므로, xQueueSendToBack() 대신 사용하는 것이 좋다.

1.3.2.1.1 매개 변수

xQueue : 데이터를 보내고자 하는(쓰여지는) Queue 의 핸들이다. Queue 의 핸들은 xQueueCreate() 또는 xQueueCreateStatic()의 호출에서 반환된다.

pvItemToQueue : Queue 에 복사할 데이터의 포인터다. Queue 가 보관할 수 있는 각 항목의 크기는 Queue 가 생성될 때 설정되며, pvItemToQueue 에서 Queue 저장소 영역으로 복사된다.

xTicksToWait : Queue 가 이미 가득 차 있으면 Queue 에서 공간을 사용할 수 있을 때까지 task 가 차단 상태로 대기하는 최대 시간이다. xTicksToWait 이 0 이고 Queue 가 가득 찬 경우, xQueueSend(), xQueueSendToFront(), xQueueSendToBack()은 즉시 반환한다.

차단 시간은 tick 기간으로 지정되므로 tick 의 절대 시간은 tick 의 주파수에 따라 달라진다. pdMS_TO_TICKS() 매크로는 tick 을 지정된 시간의 밀리 초 단위로 변환하는데 사용할 수 있다. xTicksToWait 을 portMAX_DELAY 로 설정하면 FreeRTOSConfig.h 의 INCLUDE_vTaskSuspend 가 1 로 설정되었을 때 task 가 시간 초과 없이 무기한 대기한다.

1.3.2.1.2 반환 값

pdPASS : 데이터가 Queue 에 성공적으로 전송된 경우 반환한다. 차단 시간이 지정된 경우(xTicksToWait 이 0 이 아닐 때) 함수를 반환하기 전에 Queue 를 사용할 수 있을 때까지 기다릴 수 있도록 호출한 task 가 차단 상태에 있을 수 있지만, 데이터가 성공적으로 반환되었다.

errQUEUE_FULL : Queue 가 이미 가득 차 있어서 데이터를 Queue 에 쓸 수 없을 때 반환된다. 차단 시간이 지정된 경우(xTicksToWait 이 0 이 아닐 때) 호출 task 는 차단된 상태로 전환되어 다른 task 를 기다리거나 Queue 에 공간을 만들기 위해 인터럽트 되었지만, 그 일이 있기 전에 지정된 차단 시간이 만료되었다.

1.3.2.1.3 기타

```
// Queue 의 데이터 유형을 정의한다.
typedef struct A_Message {
    char ucMessageID;
    char ucData[ 20 ];
} AMessage;

// Queue 의 매개 변수를 정의한다.
#define QUEUE_LENGTH 5
#define QUEUE_ITEM_SIZE sizeof( AMessage )

int main( void ) {
    QueueHandle_t xQueue;
```

```

// Queue 를 생성하고 반환된 핸들을 xQueue 변수에 저장한다.
xQueue = xQueueCreate( QUEUE_LENGTH, QUEUE_ITEM_SIZE );

if( xQueue == NULL ) {
    // Queue 생성에 실패하였다.
}

// Queue 핸들을 매개 변수로 전달하는 task 를 만든다.
xTaskCreate( vAnotherTask, "Task", STACK_SIZE, ( void * ) xQueue, TASK_PRIORITY, NULL );

// 실행 상태인 task 를 실행한다.
vTaskStartScheduler();

// Idle task 를 생성하기에 충분한 FreeRTOS 힙 메모리가 없을 경우에만 실행이 여기에 도달한다.
for( ;; );
}

void vATask( void *pvParameters ) {
    QueueHandle_t xQueue;
    AMessage xMessage;

    // Queue 핸들이 task 의 매개 변수로 전달된다. 매개 변수를 다시 Queue 핸들로 캐스팅한다.
    xQueue = ( QueueHandle_t ) pvParameters;

    for( ;; ) {
        // Queue 에 보낼 메시지를 만든다.
        xMessage.ucMessageID = SEND_EXAMPLE;

        // Queue 에 메시지를 보내고 Queue 가 이미 가득차면 사용할 수 있는 공간을 10 tick 동안 기다린다.
        if( xQueueSendToBack( xQueue, &xMessage, 10 ) != pdPASS ) {
            // 10tick 을 기다린 후에도 데이터를 Queue 에 보낼수 없었다.
        }
    }
}
}

```

1.3.2.2 xQueueSend...FromISR() : ISR 에서 Queue 에 쓰기

```

#include "FreeRTOS.h"
#include "queue.h"

BaseType_t xQueueSendFromISR ( QueueHandle_t xQueue, const void * pvItemToQueue,
                               BaseType_t * pxHigherPriorityTaskWoken );

BaseType_t xQueueSendToBackFromISR ( QueueHandle_t xQueue, const void * pvItemToQueue,
                                     BaseType_t * pxHigherPriorityTaskWoken );

BaseType_t xQueueSendToFrontFromISR ( QueueHandle_t xQueue, const void * pvItemToQueue,
                                     BaseType_t * pxHigherPriorityTaskWoken );

```

ISR 에서 호출할 수 있는 xQueueSend(), xQueueSendToFront(), xQueueSendToBack()이다. ISR 버전에서는 차단 시간을 지정할 수 없다.

xQueueSendFromISR() 및 xQueueSendToBackFromISR()은 동일한 연산을 수행한다. 둘 다 Queue 의 뒤쪽으로 데이터를 보낸다. xQueueSendFromISR()이 원래 버전이기 때문에, xQueueSendToBackFromISR() 대신 사용하는 것이 좋다.

인터럽트 서비스 루틴 내에서 `xQueueSendFromISR()`, `xQueueSendToBackFromISR()`, `xQueueSendToFrontFromISR()`을 호출하면 task 가 차단 상태 해제가 될 수 있다. 차단 해제된 task 의 우선 순위가 현재 실행중인 task(중단된 task)보다 높거나 같으면 컨텍스트 스위치를 수행해야 한다. 컨텍스트 스위치를 사용하면 인터럽트가 가장 높은 우선 순위의 준비 상태로 직접 반환된다.

`xQueueSend()`, `xQueueSendToBack()`, `xQueueSendToFront()` 함수와 달리 `xQueueSendFromISR()`, `xQueueSendToBackFromISR()`, `xQueueSendToFrontFromISR()`은 컨텍스트 스위치를 수행하지 않는다. 대신 컨텍스트 스위치가 필요한지 여부만 알려준다.

`xQueueSendFromISR()`, `xQueueSendToBackFromISR()`, `xQueueSendToFrontFromISR()`은 스케줄러가 시작되기 전에 호출되어서는 안된다.

1.3.2.2.1 매개 변수

`xQueue` : 데이터가 보내지는(쓰기를 하는) Queue 의 핸들이다. Queue 핸들은 `xQueueCreate()` 또는 `xQueueCreateStatic()`에 대한 호출에서 반환된다.

`pvItemToQueue` : Queue 에 복사할 데이터의 포인터다. Queue 가 보관할 수 있는 각 항목의 크기는 Queue 가 만들어 질 때 설정되며, `pvItemToQueue` 에서 Queue 저장소 영역으로 복사된다.

`pxHigherPriorityTaskWoken` : 단일 Queue 에 데이터가 사용 가능할 때까지 하나 이상의 task 가 차단 상태로 대기할 수 있다. `xQueueSendFromISR()`, `xQueueSendToFrontFromISR()`또는 `xQueueSendToBackFromISR()`을 호출하면 데이터를 사용할 수 있을 때까지 task 가 차단 상태가 될 수 있다. API 함수를 호출하면 task 가 차단 상태를 벗어나고 차단되지 않은 task 의 우선 순위가 현재 실행중인 task(중단된 task)보다 크거나 같으면 내부적으로 API 함수의 `pxHigherPriorityTaskWoken` 가 `pdTRUE` 로 설정된다. 이 값이 `pdTRUE` 로 설정되면 인터럽트가 종료되기 전에 컨텍스트 스위치가 수행되어야 한다. 이렇게 하면 인터럽트가 가장 높은 우선 순위의 준비 상태 task 로 직접 반환된다.

FreeRTOS V7.3.0 부터 `pxHigherPriorityTaskWoken` 은 선택적 매개 변수가 되어 NULL 로 설정할 수 있다.

1.3.2.2.2 반환 값

`pdTRUE` : 데이터가 Queue 에 성공적으로 전송되었다.

`errQUEUE_FULL` : Queue 가 이미 가득 차서 데이터를 Queue 에 보낼 수 없다.

1.3.2.2.3 기타

데모의 명확성을 위해 다음 예제에서는 여러개의 작은 데이터 항목을 보내도록 `xQueueSendToBackFromISR()`을 여러 번 호출한다. 이는 비효율적이므로 권장하지 않는다. 바람직한 방법은 다음과 같다.

1. 여러 데이터 항목을 구조체로 패킹 후, `xQueueSendToBackFromISR()`에 대한 단일 호출을 사용하여 전체 구조를 Queue 로 보낸다. 이 접근법은 데이터 항목의 수가 적은 경우에만 적합하다.
2. 순환 RAM 버퍼에 데이터 항목을 쓴 다음, `xQueueSendToBackFromISR()`에 대한 단일 호출을 사용하여 버퍼에 포함된 새 데이터 항목 수를 task 에 알린다.

```
// vBufferISR()은 값의 버퍼를 비우고 각 값을 Queue 에 쓰는 인터럽트 서비스 루틴이다. 데이터를 기다리는 Queue 에 차단된 여러 task 가 있을 수 있다.
void vBufferISR( void ) {
    char cIn;
    BaseType_t xHigherPriorityTaskWoken;

    // task 가 아직 차단 해제되지 않았다.
    xHigherPriorityTaskWoken = pdFALSE;

    // 버퍼가 비워질 때까지 반복한다.
    do {
        // 버퍼로부터 1 바이트를 얻는다.
        cIn = INPUT_BYTE( RX_REGISTER_ADDRESS );
```

```

    /* 바이트를 Queue 에 쓴다. Queue 에 쓰면 task 가 차단 상태를 유지하고, 차단 상태를 벗어나는 task 의 우선 순위가 현재
    실행중인 task(중단된 task)보다 높으면 xHigherPriorityTaskWoken 이 pdTRUE 로 설정된다. */
    xQueueSendToBackFromISR( xRxQueue, &cln, &xHigherPriorityTaskWoken );
} while( INPUT_BYTE( BUFFER_COUNT ) );

// 여기서 인터럽트 소스를 지운다.
/* xHigherPriorityTaskWoken 가 pdTRUE 와 같으면 버퍼가 비어 있고, 인터럽트 소스가 지워졌으므로 컨텍스트 스위치를
수행해야 한다. 참고 : ISR 에서 컨텍스트 스위치를 수행하는데 필요한 구문은 포트마다 다르다. 응용 프로그램에 필요한 구문을
찾는데 사용되는 포트에 대해 웹 설명서 및 예제를 확인하도록 한다. */
portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}

```

1.3.2.3 xQueueOverwrite() : Queue 값 덮어쓰기

```

#include "FreeRTOS.h"
#include "queue.h"

BaseType_t xQueueOverwrite ( QueueHandle_t xQueue, const void * pvItemToQueue );

```

Queue 가 가득 찬 경우에도 Queue 에 쓸 수 있는 xQueueSendToBack()에서는 큐에 이미 보관된 데이터를 덮어쓴다.

xQueueOverwrite()는 길이가 1 인 Queue 와 함께 사용하기위한 것으로 Queue 가 비어 있거나 가득 찬 것을 의미한다.

이 함수는 인터럽트 서비스 루틴에서 호출하면 안된다. 인터럽트 서비스 루틴에서는 xQueueOverwriteFromISR()을 사용한다.

1.3.2.3.1 매개 변수

xQueue : 데이터를 보낼 Queue 의 핸들이다.

pvItemToQueue : Queue 에 배치할 항목에 대한 포인터이다. Queue 가 보관할 수 있는 각 항목의 크기는 Queue 가 생성될 때 설정되며, 이 값은 pvItemToQueue 에서 Queue 저장소 영역으로 복사된다.

1.3.2.3.2 반환 값

xQueueOverwrite()는 xQueueGenericSend()를 호출하는 매크로 함수로서, xQueueSendToFront()와 동일한 반환 값을 갖는다. 그러나 pdQASS 는 xQueueOverwrite()의 Queue 가 이미 꽉 찬 경우에도 Queue 에 쓰기를 하기 때문에 반환할 수 있는 유일한 값이다.

1.3.2.3.3 기타

```

void vFunction( void *pvParameters ) {
    QueueHandle_t xQueue;
    unsigned long ulVarToSend, ulValReceived;

    /* 부호가 없는 long 값을 멤버로 하는 Queue 를 생성한다. 둘 이상의 값을 포함할 수 있는 Queue 에 대해서는
    xQueueOverwrite ()를 사용하지 않도록 권장한다. 이렇게 하면 configASSERT()가 정의된 경우 트리거 된다. */
    xQueue = xQueueCreate( 1, sizeof( unsigned long ) );

    // xQueueOverwrite()를 사용하여 값 10 을 Queue 에 쓴다.
    ulVarToSend = 10;
    xQueueOverwrite( xQueue, &ulVarToSend );

    // Queue 를 Peeking 하면 10 을 반환하지만, 값 10 은 Queue 에 남겨둔다. 블록 시간이 0 이면 Queue 가 값을 보유하고 있다.
    ulValReceived = 0;
}

```



```

xQueuePeek( xQueue, &ulValReceived, 0 );

if( ulValReceived != 10 ) {
    // 다른 task 가 값을 제거하면 오류가 발생한다.
}

// Queue 가 가득 찼다. xQueueOverwrite()를 사용하여 대기열에 있는 값을 100 으로 덮어쓴다.
ulVarToSend = 100;
xQueueOverwrite( xQueue, &ulVarToSend );

// 이번에는 Queue 를 읽어서 Queue 를 다시 비운다. 차단 시간으로 0 을 다시 사용한다.
xQueueReceive( xQueue, &ulValReceived, 0 );

// 값이 쓰여질 때 이미 Queue 가 가득 차 있더라도, 읽은 값은 마지막으로 쓰여진 값이다.
if( ulValReceived != 100 ) {
    // 다른 task 가 같은 Queue 를 사용하면 에러이다.
}
/* ... */
}

```

1.3.2.4 xQueueOverwriteFromISR() : ISR 에서 Queue 값 덮어쓰기

```

#include "FreeRTOS.h"
#include "queue.h"

BaseType_t xQueueOverwriteFromISR ( QueueHandle_t xQueue, const void * pvItemToQueue,
                                     BaseType_t * pxHigherPriorityTaskWoken );

```

ISR 에서 사용할 수 있는 xQueueOverwrite() 버전이다.

xQueueOverwriteFromISR()은 xQueueSendToBackFromISR()과 유사하지만 Queue 가 가득 차더라도 Queue 에 이미 쓰기를 한 데이터를 덮어쓰면서 Queue 에 쓰기를 한다. xQueueOverwriteFromISR()은 길이가 1 인 Queue 와 함께 사용하기 위한 것으로 Queue 가 비어 있거나 가득 찬 것이다.

1.3.2.4.1 매개 변수

xQueue : 데이터를 보낼 Queue 의 핸들이다.

pvItemToQueue : Queue 에 배치할 항목에 대한 포인터이다. Queue 가 보관할 수 있는 각 항목의 크기는 Queue 가 만들어질 때 설정되며, 바이트가 pvItemToQueue 에서 Queue 저장소 영역으로 복사된다.

pxHigherPriorityTaskWoken : xQueueOverwriteFromISR()을 Queue 로 전송하면 task 의 차단이 해제되고 차단 해제된 task 의 우선순위가 현재 실행중인 task 보다 높으면 * pxHigherPriorityTaskWoken 을 pdTRUE 로 설정한다.

xQueueOverwriteFromISR()이 값을 pdTRUE 로 설정하면 인터럽트가 종료되기 전에 컨텍스트 스위치를 요청해야 한다. 사용되는 방법은 사용하는 포트에 대한 문서의 인터럽트 서비스 루틴 섹션을 참조한다.

1.3.2.4.2 반환 값

xQueueOverwriteFromISR()은 xQueueGenericSendFromISR()을 호출하는 매크로 함수이므로, xQueueSendToFrontFromISR()과 동일한 반환 값을 갖는다. 그러나 pdQASS 는 xQueueOverwriteFromISR()이 Queue 가 이미 꽉 찬 경우에도, Queue 에 쓰기 때문에 반환할 수 있는 유일한 값이다.

1.3.2.4.3 기타

```
QueueHandle_t xQueue;

void vFunction( void *pvParameters ) {
    /* 부호가 없는 long 을 보관 유지하는 Queue 를 만든다. 둘 이상의 값을 포함할 수 있는 Queue 에 대해서
    xQueueOverwriteFromISR()를 사용하지 않는 것이 좋으며, configASSERT()이 정의된 경우 트리거가 된다. */
    xQueue = xQueueCreate( 1, sizeof( unsigned long ) );
}

void vAnInterruptHandler( void ) {
    // xHigherPriorityTaskWoken 은 사용하기 전에 pdFALSE 로 설정한다.
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    unsigned long ulVarToSend, ulValReceived;

    // xQueueOverwriteFromISR()을 사용하여 10 을 Queue 에 저장한다.
    ulVarToSend = 10;
    xQueueOverwriteFromISR( xQueue, &ulVarToSend, &xHigherPriorityTaskWoken );

    // Queue 가 가득 차 있지만, Queue 에 있는 값을 새 값으로 덮어 쓰므로 xQueueOverwriteFromISR()을 다시 호출한다.
    ulVarToSend = 100;
    xQueueOverwriteFromISR( xQueue, &ulVarToSend, &xHigherPriorityTaskWoken );

    // Queue 에서 100 을 반환한다.
    if( xHigherPriorityTaskWoken == pdTRUE ) {
        /* Queue 에 쓰기 작업이 차단 해제되고, 차단 해제된 task 의 우선 순위가 현재 실행중인 task 의 우선순위보다 높거나 같다.
        (인터럽트가 중단된 task) 이 인터럽트가 차단되지 않은 task 로 직접 반환되도록 컨텍스트 스위치를 수행한다. */
        portYIELD_FROM_ISR();
    }

    // 또는 포트 END_SWITCHING_ISR()에 따른다.
}
}
```

1.3.2.5 xQueuePeek() : Queue 데이터 유지하며 읽기

```
#include "FreeRTOS.h"
#include "queue.h"

BaseType_t xQueuePeek ( QueueHandle_t xQueue, void * pvBuffer, TickType_t xTicksToWait );
```

Queue 에서 항목을 읽지만 항목을 Queue 에서 제거하지 않는다.

xQueueReceive() 또는 xQueuePeek()을 사용하여 동일한 Queue 항목을 가져올 때, 동일한 해당 항목을 반환한다.

1.3.2.5.1 매개 변수

xQueue : 데이터를 읽을 Queue 핸들이다.

pvBuffer : Queue 에서 읽은 데이터를 복사할 메모리 포인터이다. 버퍼의 길이는 최소한 Queue 항목 크기와 같아야 한다. 항목 크기는 Queue 를 만드는데 사용된 xQueueCreate() 또는 xQueueCreateStatic()의 매개 변수 uxItemSize 로 설정된다.

xTicksToWait : Queue 가 이미 비어 있는 경우 데이터가 Queue 에서 사용 가능할 때까지 task 가 차단 상태로 대기하는 최대 시간이다. xTicksToWait 가 0 이면 Queue 가 이미 비어 있는 경우, xQueuePeek()을 즉시 반환한다. 차단 시간은 tick 시간으로 지정되고, tick 의 절대 시간은 tick 주파수에 따라 달라진다.

pdMS_TO_TICKS() 매크로를 이용하여 밀리 초 단위로 지정된 시간을 tick 시간으로 변환하는데 사용할 수 있다. xTicksToWait 을 portMAX_DELAY 로 설정할 때, FreeRTOSConfig.h 의 INCLUDE_vTaskSuspend 가 1 로 설정된 경우 task 가 시간 초과없이 무기한 대기한다.

1.3.2.5.2 반환 값

pdPASS : Queue 에서 데이터를 성공적으로 읽었을 때 반환된다. 차단 시간이 지정된 경우(xTicksToWait 이 0 이 아닐 경우) 호출하는 task 가 차단 상태에서 시간이 만료되기 전에 데이터를 Queue 에서 읽었다.

errQUEUE_EMPTY : Queue 가 비어 있어서 데이터를 읽을 수 없을 경우에 반환한다. 차단 시간이 지정되면(xTicksToWait 이 0 이 아닐 경우) 호출한 task 가 차단 상태에서 시간이 만료될 때까지 Queue 의 데이터를 읽지 못하였다.

1.3.2.5.3 기타

```
struct AMessage {
    char ucMessageID;
    char ucData[ 20 ];
} xMessage;

QueueHandle_t xQueue;

// Queue 를 만들고 값을 넣는 task 다.
void vATask( void *pvParameters ) {
    struct AMessage *pxMessage;

    // AMessage 구조체에 대해 10 개를 집어넣을 수 있는 Queue 를 만든다. xQueue 변수에 생성된 Queue 의 핸들을 저장한다.
    xQueue = xQueueCreate( 10, sizeof( struct AMessage * ) );

    if( xQueue == 0 ) {
        // Queue 데이터 구조나 저장 영역을 할당하는데 사용할 수 있는 FreeRTOS 힙 메모리가 충분하지 않아, Queue 생성에 실패했다.
    } else {
        /* AMessage 객체에 대한 포인터를 xQueue 가 참조하는 Queue 에 보낸다. Queue 가 이미 가득 찬 경우 차단하지 않는다.
        xQueueSend()의 세번째 매개 변수는 0 이므로 차단 시간이 지정되지 않는다. */
        pxMessage = &xMessage;

        xQueueSend( xQueue, ( void * ) &pxMessage, 0 );
    }

    // 나머지 작업을 한다.
    for( ;; ) {}
}

// Queue 에서 데이터를 보기 위한 task 다.
void vADifferentTask( void *pvParameters ) {
    struct AMessage *pRxedMessage;

    if( xQueue != 0 ) {
        // 생성된 Queue 에 메시지를 준다. 메시지를 즉시 사용할 수 없는 경우 10tick 동안 차단한다.
        if( xQueuePeek( xQueue, & pRxedMessage ), 10 ) == pdPASS ) {
            // pRxedMessage 는 구조체 AMessage 의 변수를 가리킨다. 그러나 항목은 Queue 에 남아있다.
        }
    } else {
        // Queue 를 만들 수 없거나 만들지 못하였다.
    }

    // 나머지 작업을 한다.
    for( ;; ) {}
}
```

1.3.2.6 xQueuePeekFromISR() : ISR 에서의 xQueuePeek()

```
#include "FreeRTOS.h"
#include "queue.h"

BaseType_t xQueuePeekFromISR ( QueueHandle_t xQueue, void * pvBuffer );
```

인터럽트 서비스 루틴(ISR)에서 사용할 수 있는 xQueuePeek()이다.

Queue 에서 항목을 읽지만 항목을 Queue 에서 제거하지 않는다. xQueueReceive() 또는 xQueuePeek()을 사용하여 동일한 Queue 에서 항목을 가져올 때, 동일한 항목을 반환한다.

1.3.2.6.1 매개 변수

xQueue : 데이터를 읽을 Queue 의 핸들이다.

pvBuffer : Queue 에서 읽은 데이터를 복사할 메모리 포인터이다. 버퍼의 길이는 최소한 Queue 항목 크기와 같아야 한다. 항목 크기는 Queue 를 만드는데 사용된 xQueueCreate() 또는 xQueueCreateStatic()의 매개 변수 uxItemSize 로 설정된다.

1.3.2.6.2 반환 값

pdPASS : Queue 에서 데이터를 성공적으로 읽을 경우 반환한다.

errQUEUE_EMPTY : Queue 가 이미 비어 있어 데이터를 읽을 수 없는 경우 반환한다.

1.3.2.7 xQueueReceive() : Queue 수신(읽기)

```
#include "FreeRTOS.h"
#include "queue.h"

BaseType_t xQueueReceive ( QueueHandle_t xQueue, void * pvBuffer, TickType_t xTicksToWait );
```

Queue 에서 항목을 수신(읽기)한다.

1.3.2.7.1 매개 변수

xQueue : 수신(읽기)하고자 하는 Queue 의 핸들이다. Queue 핸들은 xQueueCreate() 또는 xQueueCreateStatic()호출 시 반환된다.

pvBuffer : Queue 에서 읽은 데이터를 복사할 메모리 포인터이다. 버퍼의 길이는 최소한 Queue 항목 크기와 같아야 한다. 항목 크기는 Queue 를 만드는데 사용된 xQueueCreate() 또는 xQueueCreateStatic()의 매개 변수 uxItemSize 로 설정된다.

xTicksToWait : Queue 가 이미 비어 있는 경우 데이터가 Queue 에서 사용 가능할 때까지 task 가 차단 상태로 대기하는 최대 시간이다. xTicksToWait 가 0 이면 Queue 가 이미 비어 있는 경우, xQueueReceive()를 즉시 반환한다. 차단 시간은 tick 시간으로 지정되고, tick 의 절대 시간은 tick 주파수에 따라 달라진다. pdMS_TO_TICKS()매크로를 이용하여 밀리 초 단위로 지정된 시간을 tick 시간으로 변환하는데 사용할 수 있다.

xTicksToWait 을 portMAX_DELAY 로 설정할 때, FreeRTOSConfig.h 의 INCLUDE_vTaskSuspend 가 1 로 설정된 경우 task 가 시간 초과없이 무기한 대기한다.

1.3.2.7.2 반환 값

pdPASS : Queue 에서 데이터를 성공적으로 읽었을 때 반환된다. 차단 시간이 지정된 경우(xTicksToWait 이 0 이 아닐 경우) 호출하는 task 가 차단 상태에서 시간이 만료되기 전에 데이터를 Queue 에서 읽었다.

errQUEUE_EMPTY : Queue 가 비어 있어서 데이터를 읽을 수 없을 경우에 반환한다. 차단 시간이 지정되면(xTicksToWait 이 0 이 아닐 경우) 호출한 task 가 차단 상태에서 시간이 만료될 때까지 Queue 의 데이터를 읽지 못하였다.

1.3.2.7.3 기타

```
// Queue 의 데이터 유형을 정의한다.
typedef struct A_Message {
    char ucMessageID;
    char ucData[ 20 ];
} AMessage;

// Queue 의 매개 변수를 정의한다.
#define QUEUE_LENGTH 5
#define QUEUE_ITEM_SIZE sizeof( AMessage )

int main( void ) {
    QueueHandle_t xQueue;

    // Queue 를 생성하고, 핸들을 xQueue 변수에 저장한다.
    xQueue = xQueueCreate( QUEUE_LENGTH, QUEUE_ITEM_SIZE );

    if( xQueue == NULL ) {
        // Queue 를 생성할 수 없다.
    }

    // task 를 생성하고, Queue 핸들을 task 매개 변수로 전달한다.
    xTaskCreate( vAnotherTask, "Task", STACK_SIZE, ( void * ) xQueue, TASK_PRIORITY, NULL );

    // 실행 상태인 task 를 시작한다.
    vTaskStartScheduler();

    // Idle task 를 생성하기에 충분한 FreeRTOS 힙 메모리가 남아 있지 않은 경우에만 실행이 여기에 도달한다.
    for( ;; );
}

void vAnotherTask( void *pvParameters ) {
    QueueHandle_t xQueue;
    AMessage xMessage;

    // Queue 핸들이 task 매개 변수로 전달된다. 매개 변수를 다시 Queue 핸들로 캐스팅 한다.
    xQueue = ( QueueHandle_t ) pvParameters;

    for( ;; ) {
        /* Queue 에서 데이터를 사용할 수 있게 될 때까지 기다린다. FreeRTOSConfig.h 의 INCLUDE_vTaskSuspend 가 1 일 경우,
        종료 시점이 불명확하다. */
        if( xQueueReceive( xQueue, &xMessage, portMAX_DELAY ) != pdPASS ) {
            // 차단 상태로 데이터가 도착하기를 기다리지만, Queue 에서 아무것도 수신하지 못하였다.
        } else {
            // xMessage 에 수신된 데이터가 포함된다.
        }
    }
}
```

1.3.2.8 xQueueReceiveFromISR() : ISR 에서의 xQueueReceive()

```
#include "FreeRTOS.h"
#include "queue.h"

BaseType_t xQueueReceiveFromISR ( QueueHandle_t xQueue, void * pvBuffer,
                                   BaseType_t * pxHigherPriorityTaskWoken );
```

ISR 에서 호출할 수 있는 xQueueReceive()이다. xQueueReceive()과는 달리 xQueueReceiveFromISR()는 차단 시간을 지정할 수 없다.

인터럽트 서비스 루틴 내에서 xQueueReceiveFromISR()을 호출하면 Queue 에서 차단 상태인 task 가 차단 해제될 수 있다. 이러한 차단 해제된 task 우선 순위가 현재 실행중인 task(인터럽트 된 task)보다 높거나 같으면 컨텍스트 스위치를 수행해야 한다. 컨텍스트 스위치를 사용하면 인터럽트가 가장 높은 우선순위의 준비 상태 task 로 직접 반환된다. xQueueReceive()와는 달리 xQueueReceiveFromISR()은 컨텍스트 스위치를 수행하지 않는다. 대신 컨텍스트 스위치가 필요하지 여부만 알려준다. 따라서, xQueueReceiveFromISR()은 스케줄러가 시작되기 전에 호출되어서는 안된다. xQueueReceiveFromISR()을 호출하는 인터럽트는 스케줄러가 시작되기 전에 실행하지 않아야 한다.

1.3.2.8.1 매개 변수

xQueue : 수신(읽기)하고자 하는 Queue 의 핸들이다. Queue 핸들은 xQueueCreate() 또는 xQueueCreateStatic()호출 시 반환된다.

pvBuffer : Queue 에서 읽은 데이터를 복사할 메모리 포인터이다. 버퍼의 길이는 최소한 Queue 항목 크기와 같아야 한다. 항목 크기는 Queue 를 만드는데 사용된 xQueueCreate() 또는 xQueueCreateStatic()의 매개 변수 uxItemSize 로 설정된다.

pxHigherPriorityTaskWoken : 단일 Queue 에서 하나 이상의 task 가 차단되어 Queue 에서 공간을 사용할 수 있을 때까지 기다리는 것이 가능할 수 있다. xQueueReceiveFromISR()을 호출하면 사용 가능한 공간이 만들어 지므로, 그러한 task 가 차단 상태를 벗어난다.

API 함수를 호출하면 task 가 차단 상태를 벗어나고, 차단 해제된 task 우선 순위가 현재 실행중인 task(중단된 task)보다 크거나 같으면 내부적으로 API 함수가 * pxHigherPriorityTaskWoken 을 pdTRUE 로 설정한다. xQueueReceiveFromISR()의 값이 pdTRUE 로 설정되면 인터럽트가 종료되기 전에 컨텍스트 스위치가 수행되어야 한다. 이렇게 하면 인터럽트가 직접적으로 우선 순위가 가장 높은 task 를 준비 상태로 만든다. FreeRTOS V7.3.0 부터 pxHigherPriorityTaskWoken 은 선택적 매개 변수로 바뀌었으며 NULL 로 설정할 수 있다.

1.3.2.8.2 반환 값

pdPASS : 데이터가 Queue 에서 성공적으로 수신되었다.

pdFAIL : Queue 가 이미 비어있어 Queue 에서 데이터를 받지 못하였다.

1.3.2.8.3 기타

데모의 명확성을 위해 섹션의 예제에서 여러 개의 작은 데이터 항목을 받기 위해 xQueueReceiveFromISR()를 여러 번 호출한다.

이는 비효율적이므로 대부분의 응용 프로그램에 권장되지 않는다. 한 가지 구조에서 여러 데이터 항목을 단일 게시물의 Queue 로 보내고, xQueueReceiveFromISR()은 한번만 호출한다. 이는 task 레벨로 연기될 수 있다.

```
/* vISR 값은 Queue 를 비우고, 각각을 주변 장치로 보내는 인터럽트 서비스 루틴이다. Queue 가 데이터를 공간을 기다리는 동안 해당 Queue 에 여러 task 가 차단되어 있을 수 있다. */
void vISR( void ) {
    char cByte;
    BaseType_t xHigherPriorityTaskWoken;

    // task 가 아직 차단 해제되지 않았다.
    xHigherPriorityTaskWoken = pdFALSE;

    /* Queue 가 비게 될 때까지 반복한다. xQueueReceiveFromISR()를 호출하면, task 가 차단 상태를 벗어나고,
    차단 해제된 task 의 우선 순위가 현재 실행 상태인 task(ISR 이 중단된 task)보다 크거나 같으면
    xHigherPriorityTaskWoken 이 xQueueReceiveFromISR() 내에서 pdTRUE 로 설정된다. */
    while( xQueueReceiveFromISR( xQueue, &cByte, &xHigherPriorityTaskWoken ) == pdPASS ) {
```

```

// 수신된 바이트를 주변 장치에 쓴다.
OUTPUT_BYTE( TX_REGISTER_ADDRESS, cByte );
}

// 인터럽트 소스를 지운다.
/* Queue 가 비어 있고, xHigherPriorityTaskWoken 가 pdTRUE 로 설정되었을 때, 필요한 경우 컨텍스트 스위치를
수행할 수 있는 인터럽트를 해제한다. 참고 : ISR 에서 컨텍스트 스위치를 수행하는데 필요한 구문은 포트와 컴파일러마다 다르다.
응용 프로그램에 필요한 올바른 구문을 찾으려면 웹 설명서와 사용중인 포트의 예를 확인한다. */
portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}

```

1.3.2.9 xQueueReset() : Queue 리셋

```

#include "FreeRTOS.h"
#include "queue.h"

BaseType_t xQueueReset ( QueueHandle_t xQueue );

```

Queue 를 빈 상태로 리셋 한다. 리셋 될 때 Queue 에 포함된 모든 데이터는 삭제된다.

1.3.2.9.1 매개 변수

xQueue : 리셋 하고자 하는 Queue 의 핸들이다.

Queue 핸들은 xQueueCreate() 또는 xQueueCreateStatic()에 대한 반환 값으로 얻을 수 있다.

1.3.2.9.2 반환 값

xQueueReset()의 원래 버전은 pdPASS 또는 pdFAIL 을 반환하였다. 하지만, FreeRTOS V7.2.0 부터 xQueueReset()은 항상 pdPASS 를 반환한다.

1.3.3 Queue Set

1.3.3.1 vQueueCreateSet() : Queue Set 생성

```
#include "FreeRTOS.h"
#include "queue.h"

cQueueSetHandle_t xQueueCreateSet ( const UBaseType_t uxEventQueueLength );
```

Queue Set 은 RTOS task 가 RTOS Queue 들이나 세마포어에서 동시에 읽는 작업을 차단(보류)할 수 있도록 하는 메커니즘을 제공한다.

Queue Set 은 xQueueCreateSet()을 사용하여 명시적으로 만들어야 사용 가능하다. 일단 생성되면 표준 FreeRTOS Queue 와 세마포어를 xQueueAddToSet()을 이용하여 Set 에 추가할 수 있다. 그런 다음 xQueueSelectFromSet()을 사용하여, Set 에 포함된 Queue 나 세마포어가 Queue 읽기 또는 세마포어 작업이 성공했는지 여부를 확인한다.

뮤텍스가 포함된 Queue Set 을 차단해도 뮤텍스 소유자는 차단된 task 의 우선 순위를 상속하지 않는다.

Queue Set 에 추가된 모든 Queue 는 각 공간에 추가로 4 바이트 RAM 이 필요하다. 따라서 최대 카운트 값이 큰 카운터 세마포어를 Queue Set 에 추가하면 안된다.

xQueueSelectFromSet() 호출을 통해 처음 Queue 집의 멤버에 대한 핸들을 반환하지 않는 한, 수신(Queue 의 경우) 또는 task(세마포어의 경우)는 Queue Set 을 수행하면 안된다.

xQueueCreateSet() API 함수를 사용하려면 FreeRTOSConfig.h 의 configUSE_QUEUE_SETS 을 1 로 설정해야 한다.

Queue Set 을 사용하는 것보다 간단한 대안이 존재한다. 이는 여러 개체의 차단 페이지를 참조한다. 자세한 정보는 FreeRTOS.org 웹사이트를 참조한다.

1.3.3.1.1 매개 변수

uxEventQueueLength : Queue Set 은 해당 Set 에 포함된 Queue 및 세마포어에서 발생하는 이벤트를 저장한다.

uxEventQueueLength 는 한번에 대기할 수 있는 최대 이벤트 수를 지정한다. 이벤트가 손실되지 않는다는 것을 확실히 하려면

uxEventQueueLength 를 이진 세마포어와 뮤텍스의 길이 1 인 Queue 에 추가된 Queue 의 길이의 합으로 설정되어야 한다. 카운터 세마포어는 최대 카운트 값에 의해 설정된 길이를 가진다.

예를 들어 아래와 같이 사용한다.

- Queue Set 에 길이 5 인 Queue, 길이 12 인 다른 Queue 및 이진 세마포어를 보유하려면 uxEventQueueLength 를 (5+12+1) 또는 18 로 설정해야 한다.

- Queue Set 에 3 개의 이진 세마포어를 보유할 경우 uxEventQueueLength 는 (1+1+1) 또는 3 으로 설정해야 한다.

- Queue Set 의 최대 개수가 5 인 카운터 세마포어 및 최대 개수가 3 인 카운터 세마포어를 보유할 경우 uxEventQueueLength 는 (5+3) 또는 8 로 설정해야 한다.

1.3.3.1.2 반환 값

NULL : Queue Set 을 작성하지 못한 상태이다.

다른 값은 Queue Set 이 성공적으로 작성되었다. 리턴 값은 작성된 Queue Set 을 참조할 수 있는 핸들이다.

1.3.3.1.3 기타

```
// Queue Set 에 추가될 Queue 길이를 정의한다.
#define QUEUE_LENGTH_1 10
#define QUEUE_LENGTH_2 10
```



```

// 이진 세마포어의 유효 길이는 1 이다.
#define BINARY_SEMAPHORE_LENGTH 1

// 각각의 Queue1 과 Queue2 가 보유할 항목의 크기를 정의한다. 사용된 값은 데모 용이다.
#define ITEM_SIZE_QUEUE_1 sizeof( uint32_t )
#define ITEM_SIZE_QUEUE_2 sizeof( something_else_t )
// Queue Set 에 추가될 2 개의 Queue 와 이진 세마포어의 결합 길이이다.
#define COMBINED_LENGTH ( QUEUE_LENGTH_1 + QUEUE_LENGTH_2 + BINARY_SEMAPHORE_LENGTH )

void vAFunction( void ) {
    static QueueSetHandle_t xQueueSet;
    QueueHandle_t xQueue1, xQueue2, xSemaphore;
    QueueSetMemberHandle_t xActivatedMember;
    uint32_t xReceivedFromQueue1;
    something_else_t xReceivedFromQueue2;

    // 모든 Queue 와 세마포어에 대해서 이벤트를 보유할 수 있을 만큼 큰 Queue Set 을 만든다.
    xQueueSet = xQueueCreateSet( COMBINED_LENGTH );

    // Set 에 포함될 Queue 를 만든다.
    xQueue1 = xQueueCreate( QUEUE_LENGTH_1, ITEM_SIZE_QUEUE_1 );
    xQueue2 = xQueueCreate( QUEUE_LENGTH_2, ITEM_SIZE_QUEUE_2 );

    // Set 에 포함될 세마포어를 만든다.
    xSemaphore = xSemaphoreCreateBinary();

    // 세마포어를 가져와서 비운다. 이 세마포어는 방금 생산되었으며, 사용할 수 있기 때문에 차단 시간으로 0 을 준다.
    xSemaphoreTake( xSemaphore, 0 );

    // Queue 와 세마포어를 Set 에 추가한다.
    // xQueueSelectFromSet()에 대한 호출은 이 시점(Queue 또는 세마포어 핸들을 반환할 때)부터 수행할 수 있다.
    xQueueAddToSet( xQueue1, xQueueSet );
    xQueueAddToSet( xQueue2, xQueueSet );
    xQueueAddToSet( xSemaphore, xQueueSet );

    for( ;; ) {
        // Set 에 추가된 Queue 나 세마포어에서 사용할 수 있는 것을 기다린다. 200ms 이상 차단하지 않는다.
        xActivatedMember = xQueueSelectFromSet( xQueueSet, pdMS_TO_TICKS( 200 ) );

        // 선택된 멤버를 확인한다.
        // xQueueSelectFromSet()의 반환 값 핸들에 사용 가능한 것이 없는 경우, 수신 / 호출은 0 의 차단 시간을 사용한다.
        if( xActivatedMember == xQueue1 ) {
            xQueueReceive( xActivatedMember, &xReceivedFromQueue1, 0 );
            vProcessValueFromQueue1( xReceivedFromQueue1 );
        } else if( xActivatedMember == xQueue2 ) {
            xQueueReceive( xActivatedMember, &xReceivedFromQueue2, 0 );
            vProcessValueFromQueue2( xReceivedFromQueue2 );
        } else if( xActivatedMember == xSemaphore ) {
            // 세마포어가 주어질 수 있는지 다시 확인한다.
            xSemaphoreTake( xActivatedMember, 0 );
            vProcessEventNotifiedBySemaphore();
            break;
        } else {
            // 처리할 준비가 된 RTOS Queue 나 세마포어 없이 200ms 차단 시간이 만료되었다.
        }
    }
}

```

1.3.3.2 vQueueAddToSet() : Queue Set 에 추가

```
#include "FreeRTOS.h"
#include "queue.h"

BaseType_t xQueueAddToSet ( QueueSetMemberHandle_t xQueueOrSemaphore, cQueueSetHandle_t xQueueSet );
```

xQueueCreateSet()호출에 의해 이전에 작성된 Queue Set 에 Queue 또는 세마포어가 추가된다.

xQueueSelectFromSet() 호출을 통해 처음 Queue Set 의 멤버에 대한 핸들을 반환하지 않는 한, 수신(Queue 의 경우) 또는 task(세마포어의 경우)는 Queue Set 을 수행하면 안된다.

xQueueAddToSet() API 함수를 사용하려면, FreeRTOSConfig.h 의 configUSE_QUEUE_SETS 을 1 로 설정해야 한다.

이 설명은 xQueueCreateSet()함수의 예제를 참조한다.

1.3.3.2.1 매개 변수

xQueueOrSemaphore : Queue Set 에 추가되는 Queue 또는 세마포어 핸들이다.(QueueSetMemberHandle_t 유형을 사용한다.)

xQueueSet : Queue 또는 세마포어를 추가 할 Queue Set 핸들이다.

1.3.3.2.2 반환 값

pdPASS : Queue 또는 세마포어가 Queue Set 에 성공적으로 추가되었다.

pdFAIL : Queue 또는 세마포어가 이미 다른 Set 의 멤버이기 때문에 Queue Set 에 추가할 수 없다.

1.3.3.3 xQueueSelectFromSet() : Queue Set 멤버 탐색

```
#include "FreeRTOS.h"
#include "queue.h"

QueueSetMemberHandle_t xQueueSelectFromSet ( QueueSetHandle_t xQueueSet, const TickType_t xTicksToWait );
```

xQueueSelectFromSet()은 Queue Set 의 멤버에서 데이터를 포함하거나(Queue 일 경우), take(세마포어일 경우)를 이용할 수 있다.

xQueueSelectFromSet()은 효과적으로 task 를 차단(보류)하고, Queue Set 에 있는 Queue 와 세마포어들의 읽기를 동시에 처리할 수 있다. 뮤텍스가 포함된 Queue Set 을 차단해도 뮤텍스 소유자는 차단 된 task 의 우선 순위를 상속하지 않는다. xQueueSelectFromSet()의 호출이 핸들을 반환하지 않는 한, Queue Set 의 멤버들에 대한 수신(Queue 의 경우) 또는 take(세마포어의 경우)를 수행하지 않아야 한다.

xQueueSelectFromSet() 함수를 사용하려면 FreeRTOSConfig.h 의 configUSE_QUEUE_SETS 를 1 로 설정해야 한다.

Queue Set 사용에 대한 대안이 존재한다. 자세한 내용은 FreeRTOS.org 웹 사이트의 Multiple Objects 를 참조한다.

1.3.3.3.1 매개 변수

xQueueSet : 작업이 (잠재적으로) 차단할 대기열 세트입니다.

xTicksToWait : 호출 한 작업이 대기열 읽기 또는 세마포어 가져 오기 작업을 위해 대기열 세트 구성원이 준비 될 때까지 대기하기 위해 차단 된 상태 (다른 작업 실행 중)로 유지되는 최대 시간 (틱)입니다.

1.3.3.3.2 반환 값

NULL : xTicksToWait 매개 변수에 지정된 차단 시간이 만료되기 전에 Set 에 포함된 Queue 또는 세마포어를 사용할 수 없다.

다른 값 : 데이터가 들어있는 Queue Set 에 포함된 Queue 의 핸들(QueueSetMemberHandle_t 유형) 또는 사용 가능한 세마포어(QueueSetMemberHandle_t 유형)의 핸들이다.

1.3.3.3.3 기타

이 설명서의 xQueueCreateSet() 함수에 제공된 예제를 참조한다.

1.3.3.4 xQueueSelectFromSetFromISR() : ISR 에서 Queue Set 멤버 탐색

```
#include "FreeRTOS.h"
#include "queue.h"

QueueSetMemberHandle_t xQueueSelectFromSetFromISR ( QueueSetHandle_t xQueueSet );
```

인터럽트 서비스 루틴에서 사용할 수 있는 xQueueSelectFromSet()이다.

xQueueSelectFromSetFromISR()를 사용하려면 FreeRTOSConfig.h 의 configUSE_QUEUE_SETS 을 1 로 설정해야 한다.

1.3.3.4.1 매개 변수

xQueueSet : 탐색중인 Queue Set 이다. 이 함수는 인터럽트에서 사용하도록 설계되었기 때문에 읽기를 차단할 수 없다.

1.3.3.4.2 반환 값

NULL : Queue Set 의 멤버를 사용할 수 없다.

다른 값 : Queue Set 에 위치한 사용 가능한 Queue 핸들 또는 세마포어 핸들이다. (QueueSetMemberHandle_t 유형)

1.3.3.4.3 기타

```
void vReceiveFromQueueInSetFromISR( void ) {
    QueueSetMemberHandle_t xActivatedQueue;
    unsigned long ulReceived;

    // Queue Set 에 데이터가 있는지 확인한다.
    xActivatedQueue = xQueueSelectFromSetFromISR( xQueueSet );

    if( xActivatedQueue != NULL ) {
        // xQueueSelectFromSetFromISR()에 의해 반환된 Queue 를 읽는다.
        if( xQueueReceiveFromISR( xActivatedQueue, &ulReceived, NULL ) != pdPASS ) {
            // xQueueSelectFromSetFromISR()에서 핸들을 반환할 때 데이터를 사용할 수 있어야 한다.
        }
    }
}
```

1.3.3.5 xQueueRemoveFromSet() ; Queue Set 데이터 제거

```
#include "FreeRTOS.h"
#include "queue.h"

 BaseType_t xQueueRemoveFromSet ( QueueSetMemberHandle_t xQueueOrSemaphore, QueueSetHandle_t xQueueSet );
```

Queue Set 에서 Queue 또는 세마포어를 제거한다.

Queue 또는 세마포어가 비어있는 경우에만 Queue 또는 세마포어를 Queue Set 에서 제거할 수 있다.

xQueueRemoveFromSet() API 함수를 사용하려면 FreeRTOSConfig.h 의 configUSE_QUEUE_SETS 을 1 로 설정해야 한다.

1.3.3.5.1 매개 변수

xQueueOrSemaphore : Queue Set 에서 제거되는 Queue 또는 세마포어의 핸들이다. (QueueSetMemberHandle_t 유형으로 캐스팅한다.)

xQueueSet : Queue 또는 세마포어가 포함된 Queue Set 의 핸들이다.

1.3.3.5.2 반환 값

pdPASS : Queue 또는 세마포어가 Queue Set 에서 성공적으로 제거되었다.

pdFAIL : Queue 또는 세마포어가 Queue Set 에 없거나 Queue 또는 세마포어가 비어 있지 않아서 제거되지 않는다.

1.3.3.5.3 기타

이 예제는 xQueueSet 이 이미 작성된 Queue Set 이고, xQueue 가 이미 작성되어 xQueueSet 에 추가된 Queue 라고 가정한다.

```
if( xQueueRemoveFromSet( xQueue, xQueueSet ) != pdPASS ) {
    // xQueue 가 xQueueSet 의 구성원이 아니거나 xQueue 가 비어 있지 않으므로 Queue Set 에서 제거할 수 없다.
} else {
    // Queue 가 성공적으로 제거되었다.
}
```

1.3.4 Queue 레지스트리

1.3.4.1 vQueueAddToRegistry() : Queue 레지스트리에 Queue 추가

```
#include "FreeRTOS.h"
#include "queue.h"

void vQueueAddToRegistry ( QueueHandle_t xQueue, char * pcQueueName );
```

사람이 읽을 수 있는 이름을 Queue 에 할당하고, Queue 레지스트리에 Queue 를 추가한다.

Queue 레지스트리는 커널 인식 디버거에서 사용된다.

1. 디버깅 인터페이스에서 쉽게 Queue 및 세마포어 식별을 위해 텍스트 이름을 Queue 또는 세마포어와 연관시킬 수 있다.
2. 디버거가 Queue 및 세마포어 구조를 찾을 수 있는 방법을 제공한다.

configQUEUE_REGISTRY_SIZE 커널 구성 상수는 한번에 등록할 수 있는 큐 및 세마포어의 최대 수를 정의한다.

커널 인식 디버깅 인터페이스에서 볼 필요가 있는 Queue 및 세마포어만 등록해야 한다. Queue 레지스트리는 커널 인식 디버거가 사용될 때만 필요하다. 그 밖의 경우에는 용도가 없으므로 configQUEUE_REGISTRY_SIZE 를 0 으로 설정하거나, configQUEUE_REGISTRY_SIZE 구성 상수 정의를 생략할 수 있다.

등록된 Queue 를 삭제하면 자동으로 레지스트리에서 제거된다.

1.3.4.1.1 매개 변수

xQueue : 레지스트리에 추가될 큐의 핸들이다. 세마포어 핸들을 사용할 수도 있다.

pcQueueName : 대기열 또는 세마포어를 설명하는 이름이다. 이것은 FreeRTOS 에서 특별히 사용되지 않고, 디버깅을 돕는 목적으로만 사용된다. 핸들로 식별하는 것보다 식별하기 쉽게 하기 위함이다.

1.3.4.1.2 기타

```
void vAFunction( void ) {
    QueueHandle_t xQueue;

    // 10 글자를 저장할 만큼의 Queue 를 생성한다.
    xQueue = xQueueCreate( 10, sizeof( char ) );

    // 생성된 Queue 는 커널 인식 디버거에서 볼 수 있어야 하기 때문에 레지스트리에 추가한다.
    vQueueAddToRegistry( xQueue, "AMeaningfulName" );
}
```

1.3.4.2 pcQueueGetName() : Queue 텍스트 이름 출력

```
#include "FreeRTOS.h"
#include "queue.h"

const char * pcQueueGetName ( QueueHandle_t xQueue );
```

사람이 읽을 수 있는 Queue 의 텍스트 이름을 출력한다. Queue 는 Queue 레지스트리에 추가된 경우에만 텍스트 이름을 갖는다.

1.3.4.2.1 매개 변수

xQueue : 출력하고자 하는 Queue 의 핸들이다.

1.3.4.2.2 반환 값

Queue 의 텍스트 이름은 NULL 로 끝나는 C 문자열이다.

반환되는 값은 출력하는 Queue 의 이름을 가리키는 포인터이다.

1.3.5 Queue 정보 출력

1.3.5.1 xQueueIsQueueEmptyFromISR() : ISR 에서 Queue 항목 출력

```
#include "FreeRTOS.h"
#include "queue.h"

BaseType_t xQueueIsQueueEmptyFromISR ( const QueueHandle_t pxQueue );
```

Queue 에 항목이 있는지 또는 이미 비어 있는지 여부를 확인한다. Queue 가 비어 있으면 항목을 Queue 에서 받을 수 없다.

이 기능은 ISR 에서만 사용되어야 한다.

1.3.5.1.1 매개 변수

pxQueue : 출력하고자 하는 Queue 다.

1.3.5.1.2 반환 값

pdFALSE : xQueueIsQueueEmptyFromISR()가 호출될 때, 조회중인 Queue 가 비어 있다. (데이터 항목을 포함하지 않는다.)

다른 모든 값은 xQueueIsQueueEmptyFromISR()이 호출될 때 비어 있지 않는다. (데이터 항목이 포함된다.)

1.3.5.2 xQueueIsQueueFullFromISR() : ISR 에서 Queue 공간 확인

```
#include "FreeRTOS.h"
#include "queue.h"

BaseType_t xQueueIsQueueFullFromISR ( const QueueHandle_t pxQueue );
```

Queue 가 가득 차 있는지 새 항목을 받을 공간이 있는지 확인한다.

Queue 가 가득 차 있지 않을 때만 새 항목을 성공적으로 수신할 수 있다.

이 기능은 ISR 에서 사용할 수 있다.

1.3.5.2.1 매개 변수

pxQueue : 확인하고자 하는 Queue 다.

1.3.5.2.2 반환 값

pdFALSE : Queue 가 가득 차 있지 않다.

다른 모든 값은 xQueueIsQueueFullFromISR()을 호출할 때 가득 차 있는 상태이다.

1.3.5.3 uxQueueMessagesWaiting() : Queue 가 보관하는 항목의 수 출력

```
#include "FreeRTOS.h"
#include "queue.h"

UBaseType_t uxQueueMessagesWaiting ( const QueueHandle_t xQueue );
```

Queue 에 보관되고 있는 항목의 수를 출력한다.

1.3.5.3.1 매개 변수

xQueue : 출력하고자 하는 Queue 의 핸들이다.

1.3.5.3.2 반환 값

uxQueueMessagesWaiting()을 호출할 때 확인하고자 하는 Queue 가 보관하고 있는 항목의 수다.

1.3.5.3.3 기타

```
void vAFunction( QueueHandle_t xQueue ) {
    UBaseType_t uxNumberOfItems;

    // xQueue 핸들에 의해 참조되는 Queue 가 보관하는 항목의 수를 받는다.
    uxNumberOfItems = uxQueueMessagesWaiting( xQueue );
}
```

1.3.5.4 uxQueueMessagesWaitingFromISR() : ISR 에서 보관하는 항목의 수 출력

```
#include "FreeRTOS.h"
#include "queue.h"

UBaseType_t uxQueueMessagesWaitingFromISR ( const QueueHandle_t xQueue );
```

ISR 에서 사용할 수 있는 uxQueueMessagesWaiting()이다.

1.3.5.4.1 매개 변수

xQueue : 출력하고자 하는 Queue 의 핸들이다.

1.3.5.4.2 반환 값

uxQueueMessagesWaitingFromISR()을 호출할 때 확인하고자 하는 Queue 가 보관하고 있는 항목의 수다.

1.3.5.4.3 기타

```
void vAnInterruptHandler( void ) {
    UBaseType_t uxNumberOfItems;
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    // Queue 의 상태를 확인한다. Queue 에 10 개 이상의 항목이 포함되어 있는 경우, Queue 를 비우는 작업을 수행한다.
    // xQueue 핸들에 참조된 Queue 가 얼마나 많은 항목을 갖고 있는지 확인한다.
    uxNumberOfItems = uxQueueMessagesWaitingFromISR( xQueue );

    if( uxNumberOfItems > 10 ) {
        // task 가 깨어 있는 상태면 xSemaphore 는 차단된다. 세마포어를 지정하면 task 가 차단 해제된다.
        xSemaphoreGiveFromISR( xSemaphore, &xHigherPriorityTaskWoken );
    }

    /* 이 시점에서 xHigherPriorityTaskWoken 가 pdTRUE 일 경우, xSemaphoreGiveFromISR()호출로 차단 해제된
    task 의 우선순위가 현재 실행중인 task 보다 크거나 같다. (인터럽트가 발생했을 때 실행 상태에 있던 task) 이 경우
    인터럽트 서비스 루틴을 떠나기 전에 우선순위가 높은 준비 상태 task(차단 해제된 task)로 인터럽트가 반환되도록
    컨텍스트 스위치를 수행한다. 인터럽트 내부에서 컨텍스트 스위치를 수행하는데 필요한 구문은 포트마다 다르다.
    사용중인 포트에 대한 웹 문서 및 예제를 확인하여 응용 프로그램에 맞는 구문을 사용한다. */
}
```

1.3.5.5 uxQueueSpacesAvailable() : Queue 여유 공간 출력

```
#include "FreeRTOS.h"
#include "queue.h"

UBaseType_t uxQueueSpacesAvailable ( const QueueHandle_t xQueue );
```

Queue 에 사용 가능한 여유 공간의 수를 반환한다. 즉, Queue 가 가득 차기 전에 Queue 에 게시할 수 있는 항목의 수다.

1.3.5.5.1 매개 변수

xQueue : 조회하고자 하는 Queue 의 핸들이다.

1.3.5.5.2 반환 값

uxQueueSpacesAvailable()가 호출될 때 질문하는 Queue 에서 사용 가능한 여유 공간의 수다.

1.3.5.5.3 기타

```
void vAFunction( QueueHandle_t xQueue ) {
    UBaseType_t uxNumberOfFreeSpaces;

    // 현재 xQueue 핸들에 의해 참조되는 Queue 의 여유 공간을 uxNumberOfFreeSpaces 에 저장한다.
    uxNumberOfFreeSpaces = uxQueueSpacesAvailable( xQueue );
}
```