Xilinx Zynq FPGA, TI DSP, MCU 기반의 회로 설계 및 임베디드 전문가 과정

강사 - Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 - TaeYoung Eun(은태영)

zero_bird@naver.com

1.10 메시지 버퍼 API

1.10.1 메시지 버퍼 생성 및 제거

1.10.1.1 xMessageBufferCreate : 메시지 버퍼 동적 생성

```
#include “FreeRTOS.h”
#include “message_buffer.h”

MessageBufferHandle_t xMessageBufferCreate ( size_t xBufferSizeBytes );
```

동적으로 할당된 메모리를 사용하여 새로운 메시지 버퍼를 만든다.

정적으로 할당된 메모리(컴파일 타임에 할당된 메모리)를 사용하는 버전은 xMessageBufferCreateStatic()를 참조한다.

xMessageBufferCreate()를 사용하려면 FreeRTOSConfig.h 의 configSUPPORT_DYNAMIC_ALLOCATION 를 1 로 설정하거나 정의되지 않은 상태로 지정해야 한다.

메시지 버퍼 기능은 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함시켜야 활성화된다.

(메시지 버퍼는 스트림 버퍼를 사용한다.).

1.10.1.1.1 매개 변수

xBufferSizeBytes : 메시지 버퍼가 한번에 보유할 수 있는 총 바이트 수(메시지가 아니다.)다. 메시지가 메시지 버퍼에 기록되면, 메시지 길이를 저장하기 위해서 sizeof(size_t)바이트가 추가로 기입된다. sizeof(size_t)는 32 비트 아키텍처에서 일반적으로 4 바이트이며, 대부분의 32 비트 아키텍처에서 10 바이트 메시지는 14 바이트의 메시지 버퍼 공간을 차지한다.

1.10.1.1.2 반환 값

NULL 이 반환되면 FreeRTOS 가 메시지 버퍼 데이터 구조 및 저장 영역을 할당하기에 사용할수 있는 힙 메모리가 부족하기 때문에 메시지 버퍼를 생성할 수 없다. 반환되는 값이 NULL 이 아닌 값은 메시지 버퍼가 성공적으로 생성되었음을 나타낸다. 반환 값은 생성된 메시지 버퍼에 대한 핸들로 저장되야 한다.

1.10.1.1.3 기타

```
void vAFunction( void ) {
    MessageBufferHandle_t xMessageBuffer;
    const size_t xMessageBufferSizeBytes = 100;

    // 100 바이트를 저장할수 있는 메시지 버퍼를 만든다.
    // 메시지 버퍼 구조와 데이터를 메시지 버퍼에 보관하는데 사용되는 메모리는 동적으로 할당된다.
    xMessageBuffer = xMessageBufferCreate( xMessageBufferSizeBytes );

    if( xMessageBuffer == NULL ) {
      // 힙 메모리 공간이 부족하여 메시지 버퍼를 만들 수 없다.
    } else {
      // 메시지 버퍼가 성공적으로 만들어 졌으므로 이제 사용할 수 있다.
    }
}
```

1.10.1.2 xMessageBufferCreateStatic : 메시지 버퍼 정적 생성

```
#include “FreeRTOS.h”
#include “message_buffer.h”

MessageBufferHandle_t xMessageBufferCreateStatic ( size_t xBufferSizeBytes, uint8_t * pucMessageBufferStorageArea,
                                                   StaticMessageBuffer_t * pxStaticMessageBuffer );
```

정적으로 할당된 메모리를 사용하여 새로운 메시지 버퍼를 만든다. 동적으로 할당된 메모리를 사용하는 버전은 xMessageBufferCreate()를 참조한다.

xMessageBufferCreateStatic()를 사용하려면 FreeRTOSConfig.h 의 configSUPPORT_STATIC_ALLOCATION 을 1 로 설정한다.

메시지 버퍼 기능은 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함시켜야 활성화된다.

(메시지 버퍼는 스트림 버퍼를 사용한다.).

1.10.1.2.1 매개 변수

xBufferSizeBytes : pucMessageBufferStorageArea 매개 변수가 가리키는 버퍼의 크기(바이트)다. 메시지가 메시지 버퍼에 기록되면, 메시지 길이를 저장하기 위해서 sizeof(size_t)바이트가 추가로 기록된다. sizeof(size_t)는 32 비트 아키텍처에서 일반적으로 4 바이트이며, 대부분의 32 비트 아키텍처에서 10 바이트 메시지는 14 바이트의 메시지 버퍼 공간을 차지한다. 저장될 수 있는 최대 바이트 수 메시지 버퍼는 실제로 xBufferSizeBytes - 1 이다.

pucMessageBufferStorageArea : 최소한 xBufferSizeBytes + 1 만큼 큰 uint8_t 배열을 가리켜야 한다. 이것은 메시지 버퍼에 메시지가 복사될 때 복사되는 배열이다.

pxStaticMessageBuffer : 최소한 xBufferSizeBytes + 1 만큼 큰 uint8_t 배열을 가리켜야 한다. 이것은 메시지 버퍼에 메시지가 복사될 때 복사되는 배열이다.

1.10.1.2.2 반환 값

메시지 버퍼가 성공적으로 생성되면 생성된 메시지 버퍼에 대한 핸들이 반환된다.

pucMessageBufferStorageArea 또는 pxStaticMessageBuffer 가 NULL 이면 NULL 이 반환된다.

1.10.1.2.3 기타

```
// 메시지를 보관 유지하기 위해 사용되는 배열의 치수를 나타내기 위해 사용된다.
// 사용 가능한 공간은 실제로 이보다 작은 공간으로 999 다.
#define STORAGE_SIZE_BYTES 1000

// 메시지 버퍼 내에서 실제로 메시지를 보유할 메모리를 정의한다. xBufferSizeBytes 매개 변수에 전달된 값보다 하나 많아야 한다.
static uint8_t ucStorageBuffer[ STORAGE_SIZE_BYTES ];

// 메시지 버퍼 구조를 보유하기 위해 사용되는 변수다.
StaticMessageBuffer_t xMessageBufferStruct;

void MyFunction( void ) {
    MessageBufferHandle_t xMessageBuffer;
    xMessageBuffer = xMessageBufferCreateStatic( sizeof( ucStoragegBuffer ), ucBufferStorage, &xMessageBufferStruct );

    /* pucMessageBufferStorageArea 또는 pxStaticMessageBuffer 매개 변수가 NULL 이 아니므로,
    xMessageBuffer 는 NULL 이 아니며 다른 메시지 버퍼 API 호출에서 생성된 메시지 버퍼를 참조하는 데 사용할 수 있다. * /
    // 메시지 버퍼를 사용하는 다른 코드는 여기에 위치할 수 있다.
}
```

1.10.1.3 vMessageBufferDelete : 메시지 버퍼 삭제

```
#include “FreeRTOS.h”
#include “message_buffer.h”

void vMessageBufferDelete ( MessageBufferHandle_t xMessageBuffer );
```

xMessageBufferCreate()또는 xMessageBufferCreateStatic()에 대한 호출을 사용하여 이전에 작성된 메시지 버퍼를 삭제한다.

메시지 버퍼가 동적 메모리를 사용하여(xMessageBufferCreate()) 생성된 경우, 할당된 메모리가 해제된다.

메세지 버퍼가 삭제 된 후에 메세지 버퍼 핸들을 사용해서는 안된다.

메시지 버퍼 기능은 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함해야 활성화된다.

(메시지 버퍼는 스트림 버퍼를 사용한다.).

1.10.1.3.1 매개 변수

xMessageBuffer : 삭제할 메시지 버퍼의 핸들이다.

1.10.2 메시지 버퍼 기능 함수

1.10.2.1 xMessageBufferReceive : 메시지 버퍼 수신

```
#include “FreeRTOS.h”
#include “message_buffer.h”

size_t xMessageBufferReceive ( MessageBufferHandle_t xMessageBuffer, void * pvRxData,
                               size_t xBufferLengthBytes, TickType_t xTicksToWait );
```

RTOS 메시지 버퍼로부터 개별 메시지를 수신한다. 메시지는 가변 길이가 될 수 있으며, 버퍼에서 복사된다.

FreeRTOS 객체 중 스트림 버퍼 구현(또는 메시지 버퍼 구현, 메시지 버퍼가 스트림 버퍼의 맨 위에 구축된다.)은 버퍼에 쓰는 task 나 인터럽트가 하나뿐이며, 하나의 task 또는 인터럽트로 버퍼에서 읽는다. 읽고 쓰는 것은 다른 task 나 인터럽트를 사용하는 것이 안전하며, 다른 FreeRTOS 개체와 달리 여러개의 읽고 쓰기는 안전하지 않다. 여러개의 다른 쓰기가 있어야 하는 경우, 응용 프로그램 작성자는 각 호출을 크리티컬 섹션 내부에 쓰는 API 함수(예를 들어 xMessageBufferSend())에 배치하고 송신 차단 시간을 0 으로 사용해야 한다. 마찬가지로 여러 읽기가 있어야 하는 경우 응용 프로그램 작성자는 각 호출을 크리티컬 섹션 내부에 있는 읽기 API 함수(예를 들어 xMessageBufferRead())에 배치하고 수신 차단 시간을 0 으로 사용해야 한다.

xMessageBufferReceive()를 사용하면 task 에서 메시지 버퍼를 읽는다. 인터럽트 서비스 루틴에서 메시지 버퍼를 읽으려면 xMessageBufferReceiveFromISR()를 사용한다.

메시지 버퍼 기능은 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함해야 활성화된다.

(메시지 버퍼는 스트림 버퍼를 사용한다.).

1.10.2.1.1 매개 변수

xMessageBuffer : 메시지가 수신되는 메시지 버퍼의 핸들이다.

pvRxData : 수신된 메시지가 복사될 버퍼에 대한 포인터다.

xBufferLengthBytes : pvRxData 매개 변수가 가리키는 버퍼의 길이다. 이것은 수신할 수 있는 메시지의 최대 길이를 설정한다. xBufferLengthBytes 가 너무 작아서 다음 메시지를 보관할 수 없으면 메시지는 메시지 버퍼에 남게되고 0 을 반환한다.

xTicksToWait : xMessageBufferReceive()가 호출되었을 때 메시지 버퍼가 비어 있는 경우, 메시지를 기다리는 task 가 차단 상태로 대기하여 유지되는 최대 시간이다. xMessageBufferReceive()는 xTicksToWait 가 0 이고 메시지 버퍼가 비어있으면 즉시 반환된다. 차단 시간은 tick 으로 지정되며, tick 의 절대 시간은 tick 주파수에 따라 달라진다.

pdMS_TO_TICKS()매크로는 밀리 초 단위로 지정된 시간을 틱으로 변환하는데 사용할 수 있다. FreeRTOSConfig.h 의 INCLUDE_vTaskSuspend 가 1 로 설정된 경우, xTicksToWait 을 portMAX_DELAY 로 설정하면 task 가 시간 초과없이 무기한 대기한다. task 는 차단 상태일때, CPU 시간을 사용하지 않는다.

1.10.2.1.2 반환 값

메시지 버퍼에서 읽은 메시지의 길이(바이트)다. (있을 경우)

메시지를 사용 가능하게 되기 전 xMessageBufferReceive()가 시간 초과되면 0 을 반환한다. 메시지 길이가 xBufferLengthBytes 보다 큰 경우 메시지는 메시지 버퍼에 남아 있고 0 을 리턴한다.

1.10.2.1.3 기타

```
void vAFunction( MessageBuffer_t xMessageBuffer ) {
    uint8_t ucRxData[ 20 ];
    size_t xReceivedBytes;
    const TickType_t xBlockTime = pdMS_TO_TICKS( 20 );
```

```
    // 메시지 버퍼로부터 다음 메시지를 수신한다.
    // 차단 상태(CPU 처리 시간을 사용하지 않는다.)에서 최대 100ms 동안 메시지를 사용할 수 있을때까지 대기한다.
    xReceivedBytes = xMessageBufferReceive( xMessageBuffer, ( void * ) ucRxData, sizeof( ucRxData ), xBlockTime );

    if( xReceivedBytes > 0 ) {
      // ucRxData 는 xReceivedBytes 길이의 메시지를 포함한다. 여기에서 메시지를 처리한다.
    }
}
```

1.10.2.2 xMessageBufferReceiveFromISR : ISR 에서 메시지 버퍼 수신

```
#include "FreeRTOS.h"
#include "message_buffer.h"

size_t xMessageBufferReceiveFromISR ( MessageBufferHandle_t xMessageBuffer, void * pvRxData,
                                      size_t xBufferLengthBytes, BaseType_t * pxHigherPriorityTaskWoken );
```

메시지 버퍼에서 개별 메시지를 수신하는 API 함수의 인터럽트 서비스 루틴 버전이다.

메시지는 가변 길이가 될 수 있으며 버퍼에서 복사된다.

FreeRTOS 객체 중 스트림 버퍼 구현(또는 메시지 버퍼 구현, 메시지 버퍼가 스트림 버퍼의 맨 위에 구축된다.)은 버퍼에 쓰는 task 나 인터럽트가 하나뿐이며, 하나의 task 또는 인터럽트로 버퍼에서 읽는다. 읽고 쓰는 것은 다른 task 나 인터럽트를 사용하는 것이 안전하며, 다른 FreeRTOS 개체와 달리 여러개의 읽고 쓰기는 안전하지 않다. 여러개의 다른 쓰기가 있어야 하는 경우, 응용 프로그램 작성자는 각 호출을 크리티컬 섹션 내부에 쓰는 API 함수(예를 들어 xMessageBufferSend())에 배치하고 송신 차단 시간을 0 으로 사용해야 한다. 마찬가지로 여러 읽기가 있어야 하는 경우 응용 프로그램 작성자는 각 호출을 크리티컬 섹션 내부에 있는 읽기 API 함수(예를 들어 xMessageBufferRead())에 배치하고 수신 차단 시간을 0 으로 사용해야 한다.

xMessageBufferReceive()를 사용하면 task 에서 메시지 버퍼를 읽는다. 인터럽트 서비스 루틴에서 메시지 버퍼를 읽으려면 xMessageBufferReceiveFromISR()를 사용한다.

메시지 버퍼 기능은 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함해야 활성화된다.

(메시지 버퍼는 스트림 버퍼를 사용한다.).

1.10.2.2.1 매개 변수

xMessageBuffer : 메시지가 수신되는 메시지 버퍼의 핸들이다.

pvRxData : 수신된 메시지가 복사될 버퍼에 대한 포인터다.

xBufferLengthBytes : pvRxData 매개 변수가 가리키는 버퍼의 길이다. 이것은 수신할 수 있는 메시지의 최대 길이를 설정한다. xBufferLengthBytes 가 너무 작아서 다음 메시지를 보관할 수 없으면 메시지는 메시지 버퍼에 남게되고 0 을 반환한다.

pxHigherPriorityTaskWoken : 메시지 버퍼는 공간이 사용 가능할 때가지 task 를 차단할 수 있다. xMessageBufferReceiveFromISR()을 호출하면 사용 가능한 공간이 생기므로 공관을 확보하기 위해 대기중이던 task 가 차단 해제 상태가 될 수 있다. xMessageBufferReceiveFromISR()를 호출하면 task 의 차단 상태를 벗어나고 차단 해제된 task 의 우선순위가 현재 실행중인 task(인터럽트 된 task)보다 높을 경우, 내부적으로 xMessageBufferReceiveFromISR()은 *pxHigherPriorityTaskWoken 을 pdTRUE 로 설정한다. xMessageBufferReceiveFromISR()이 pdTRUE 로 설정하면 보통 인터럽트가 종료되기 전에 컨텍스트 스위치가 수행되야 한다. 이렇게 하면 인터럽트가 최상위 우선순위의 준비 상태 task 로 직접 반환된다. * pxHigherPriorityTaskWoken 은 함수로 전달되기 전에 pdFALSE 로 설정되야 한다.

1.10.2.2.2 반환 값

메시지 버퍼에서 읽은 메시지의 길이(바이트)다. (있는 경우)

1.10.2.2.3 기타

```
// 이미 생성된 메시지 버퍼다.
MessageBuffer_t xMessageBuffer;

void vAnInterruptServiceRoutine( void ) {
    uint8_t ucRxData[ 20 ];
    size_t xReceivedBytes;
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    // pdFALSE 로 초기화 한다.
    // 메시지 버퍼로부터 다음 메시지를 수신한다.
    xReceivedBytes = xMessageBufferReceiveFromISR( xMessageBuffer, ( void * ) ucRxData,
                                                   sizeof( ucRxData ), &xHigherPriorityTaskWoken );
    if( xReceivedBytes > 0 ) {
      // ucRxData 는 xReceivedBytes 길이의 메시지를 포함한다. 여기에서 메시지를 처리한다.
    }

    /* xHigherPriorityTaskWoken 이 pdTRUE 로 설정되면 현재 실행중인 task 의 우선 순위보다 우선 순위가 높은 task 가
    차단 해제되었기 때문에 컨텍스트 스위치가 요청되야 한다. 사용된 매크로는 특정 포트에 따라 다르며,
    portYIELD_FROM_ISR() 또는 END_SWITCHING_ISR() 이다. 사용중인 포트의 설명서 페이지를 참조한다. */
    taskYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

1.10.2.3 xMessageBufferSend : 메시지 버퍼에 송신

```
#include “FreeRTOS.h”
#include “message_buffer.h”

size_t xMessageBufferSend ( MessageBufferHandle_t xMessageBuffer, const void * pvTxData,
                            size_t xDataLengthBytes, TickType_t xTicksToWait );
```

메시지 버퍼에 개별 메시지를 보낸다. 메시지는 버퍼의 여유 공간 내에 있는 길이일 때, 버퍼에 복사된다.

FreeRTOS 객체 중 스트림 버퍼 구현(또는 메시지 버퍼 구현, 메시지 버퍼가 스트림 버퍼의 맨 위에 구축된다.)은 버퍼에 쓰는 task 나 인터럽트가 하나뿐이며, 하나의 task 또는 인터럽트로 버퍼에서 읽는다. 읽고 쓰는 것은 다른 task 나 인터럽트를 사용하는 것이 안전하며, 다른 FreeRTOS 개체와 달리 여러개의 읽고 쓰기는 안전하지 않다. 여러개의 다른 쓰기가 있어야 하는 경우, 응용 프로그램 작성자는 각 호출을 크리티컬 섹션 내부에 쓰는 API 함수(예를 들어 xMessageBufferSend())에 배치하고 송신 차단 시간을 0 으로 사용해야 한다. 마찬가지로 여러 읽기가 있어야 하는 경우 응용 프로그램 작성자는 각 호출을 크리티컬 섹션 내부에 있는 읽기 API 함수(예를 들어 xMessageBufferRead())에 배치하고 수신 차단 시간을 0 으로 사용해야 한다.

xMessageBufferSend()를 사용하면 task 에서 메시지 버퍼에 기록한다. 인터럽트 서비스 루틴에서 메시지 버퍼에 쓰려면 xMessageBufferSendFromISR()를 사용한다.

메시지 버퍼 기능은 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함해야 활성화된다.

(메시지 버퍼는 스트림 버퍼를 사용한다.).

1.10.2.3.1 매개 변수

xMessageBuffer : 메시지가 전송되는 메시지 버퍼의 핸들이다.

pvTxData : 메시지 버퍼에 복사될 메시지에 대한 포인터다.

xDataLengthBytes : 메시지의 길이다. 즉 pvTxData 에서 메시지 버퍼로 복사할 바이트 수다. 메시지가 메시지 버퍼에 기록되면, 메시지 길이를 저장하기 위해 sizeof(size_t)바이트가 추가로 기입된다. sizeof(size_t)는 32 비트 아키텍처에서 일반적으로 4 바이트이므로 대부분의 32 비트 아키텍처에서 xDataLengthBytes 를 20 으로 설정하면 메시지 버퍼의 여유 공간이 24 바이트가 된다. (메시지 데이터 20 바이트 + 메시지 길이)

xTicksToWait : : xMessageBufferSend()가 호출되었을 때 메시지 버퍼의 공간이 충분하지 않은 경우, 호출하는 task 가 차단 상태로 대기하여 유지되는 최대 시간이다. xTicksToWait 가 0 이면 호출 task 가 차단되지 않는다. 차단 시간은 tick 으로 지정되며, tick 의 절대 시간은 tick 주파수에 따라 달라진다.

pdMS_TO_TICKS()매크로는 밀리 초 단위로 지정된 시간을 틱으로 변환하는데 사용할 수 있다. FreeRTOSConfig.h 의 INCLUDE_vTaskSuspend 가 1 로 설정된 경우, xTicksToWait 을 portMAX_DELAY 로 설정하면 task 가 시간 초과없이 무기한 대기한다. task 는 차단 상태일때, CPU 시간을 사용하지 않는다.

1.10.2.3.2 반환 값

메시지 버퍼에 기록된 바이트 수다. xMessageBufferSend()에 대한 호출이 메시지 버퍼에 메시지를 쓰는데 충분한 공간이 있기 전에 시간초과 하면 0 이 반환된다. 호출 시간을 초과하지 않으면 xDataLengthBytes 가 반환된다.

1.10.2.3.3 기타

```
void vAFunction ( MessageBufferHandle_t xMessageBuffer ) {
    size_t xBytesSent;
    uint8_t ucArrayToSend[] = { 0, 1, 2, 3 };
    char *pcStringToSend = "String to send";
    const TickType_t x100ms = pdMS_TO_TICKS( 100 );

    // 메시지 버퍼에 배열을 보내고 메시지 버퍼에 충분한 공간이 있을 때까지 최대 100ms 동안 차단 상태로 대기한다.
    xBytesSent = xMessageBufferSend( xMessageBuffer, ( void * ) ucArrayToSend, sizeof( ucArrayToSend ), x100ms );
    if( xBytesSent != sizeof( ucArrayToSend ) ) {
      // xMessageBufferSend()에 대한 호출은 버퍼에 데이터를 쓸 공간이 충분해지기 전에 시간 초과 되었다.
    }

    // 문자열을 메시지 버퍼로 보낸다. 버퍼에 충분한 공간이 없는 경우 즉시 반환한다.
    xBytesSent = xMessageBufferSend( xMessageBuffer, ( void * ) pcStringToSend, strlen( pcStringToSend ), 0 );

    if( xBytesSent != strlen( pcStringToSend ) ) {
      // 버퍼에 충분한 여유 공간이 없으므로, 문자열을 메시지 버퍼에 추가할 수 없다.
    }
}
```

1.10.2.4 xMessageBufferSendFromISR : ISR 에서 메시지 버퍼에 송신

```
#include “FreeRTOS.h”
#include “message_buffer.h”

size_t xMessageBufferSendFromISR ( MessageBufferHandle_t xMessageBuffer, const void * pvTxData,
                                   size_t xDataLengthBytes, BaseType_t * pxHigherPriorityTaskWoken );
```

메시지 버퍼에 개별 메시지를 보내는 API 함수의 인터럽트 서비스 루틴 버전이다.

메시지는 버퍼의 여유 공간 내에 있는 길이일 때, 버퍼에 복사된다.

FreeRTOS 객체 중 스트림 버퍼 구현(또는 메시지 버퍼 구현, 메시지 버퍼가 스트림 버퍼의 맨 위에 구축된다.)은 버퍼에 쓰는 task 나 인터럽트가 하나뿐이며, 하나의 task 또는 인터럽트로 버퍼에서 읽는다. 읽고 쓰는 것은 다른 task 나 인터럽트를 사용하는 것이 안전하며, 다른 FreeRTOS 개체와 달리 여러개의 읽고 쓰기는 안전하지 않다. 여러개의 다른 쓰기가 있어야 하는 경우, 응용 프로그램 작성자는 각 호출을 크리티컬 섹션 내부에 쓰는 API 함수(예를 들어 xMessageBufferSend())에 배치하고 송신 차단 시간을 0 으로 사용해야 한다. 마찬가지로 여러 읽기가 있어야 하는 경우 응용 프로그램 작성자는 각 호출을 크리티컬 섹션 내부에 있는 읽기 API 함수(예를 들어 xMessageBufferRead())에 배치하고 수신 차단 시간을 0 으로 사용해야 한다.

xMessageBufferSend()를 사용하면 task 에서 메시지 버퍼에 기록한다. 인터럽트 서비스 루틴에서 메시지 버퍼에 쓰려면 xMessageBufferSendFromISR()를 사용한다.

메시지 버퍼 기능은 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함해야 활성화된다.

(메시지 버퍼는 스트림 버퍼를 사용한다.).

1.10.2.4.1 매개 변수

xMessageBuffer : 메시지가 전송되는 메시지 버퍼의 핸들이다.

pvTxData : 메시지 버퍼에 복사될 메시지에 대한 포인터다.

xDataLengthBytes : 메시지의 길이다. 즉 pvTxData 에서 메시지 버퍼로 복사할 바이트 수다. 메시지가 메시지 버퍼에 기록되면, 메시지 길이를 저장하기 위해 sizeof(size_t)바이트가 추가로 기입된다. sizeof(size_t)는 32 비트 아키텍처에서 일반적으로 4 바이트이므로 대부분의 32 비트 아키텍처에서 xDataLengthBytes 를 20 으로 설정하면 메시지 버퍼의 여유 공간이 24 바이트가 된다. (메시지 데이터 20 바이트 + 메시지 길이)

pxHigherPriorityTaskWoken : 메시지 버퍼의 데이터를 기다리는 동안 차단된 task 가 있을 수 있다. xMessageBufferSendFromISR()을 호출하면 데이터를 사용할 수 있게 되므로 task 가 차단 해제 상태가 되어 대기한다. xMessageBufferSendFromISR()를 호출하면 task 의 차단 상태를 벗어나고 차단 해제된 task 의 우선순위가 현재 실행중인 task(인터럽트 된 task)보다 높을 경우, 내부적으로 xMessageBufferSendFromISR()은 *pxHigherPriorityTaskWoken 을 pdTRUE 로 설정한다. xMessageBufferSendFromISR()가 pdTRUE 로 설정하면 보통 인터럽트가 종료되기 전에 컨텍스트 스위치가 수행되야 한다. 이렇게 하면 인터럽트가 최상위 우선순위의 준비 상태 task 로 직접 반환된다. * pxHigherPriorityTaskWoken 은 함수로 전달되기 전에 pdFALSE 로 설정되야 한다.

1.10.2.4.2 반환 값

메시지 버퍼에 실제로 기록된 바이트 수다.

메시지 버퍼에 메시지 저장 공간이 충분하지 않을 경우 0 이 반환되고, 아닐경우 xDataLengthBytes 가 반환된다.

1.10.2.4.3 기타

```
// 이미 생성된 메시지 버퍼다.
MessageBufferHandle_t xMessageBuffer;

void vAnInterruptServiceRoutine( void ) {
```

```
    size_t xBytesSent;
    char *pcStringToSend = "String to send";
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    // pdFALSE 로 초기화한다.
    // 문자열을 메시지 버퍼에 보낸다.
    xBytesSent = xMessageBufferSendFromISR( xMessageBuffer, ( void * ) pcStringToSend,
                                            strlen( pcStringToSend ), &xHigherPriorityTaskWoken );
    if( xBytesSent != strlen( pcStringToSend ) ) {
      // 버퍼에 충분한 여유 공간이 없으므로 문자열을 메시지 버퍼에 추가할 수 없다.
    }

    /* xHigherPriorityTaskWoken 이 pdTRUE 로 설정되면 현재 실행중인 task 의 우선 순위보다 우선 순위가 높은 task 가
    차단 해제되었기 때문에 컨텍스트 스위치가 요청되야 한다. 사용된 매크로는 특정 포트에 따라 다르며,
    portYIELD_FROM_ISR() 또는 END_SWITCHING_ISR() 이다. 사용중인 포트의 설명서 페이지를 참조한다. */
    taskYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

1.10.2.5 xMessageBufferReset : 메시지 버퍼 리셋

```
#include “FreeRTOS.h”
#include “message_buffer.h”

BaseType_t xMessageBufferReset ( MessageBufferHandle_t xMessageBuffer );
```

메시지 버퍼를 초기 상태(비어있는 상태)로 리셋한다.

메시지 버퍼에 있는 모든 데이터는 삭제된다. 메시지 버퍼는 메시지 버퍼로 보내거나 메시지 버퍼로부터 수신 대기중인 task 가 차단되지 않은 경우에만 재설정 할 수 있다.

메시지 버퍼 기능은 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함해야 활성화된다.

(메시지 버퍼는 스트림 버퍼를 사용한다.).

1.10.2.5.1 매개 변수

xMessageBuffer : 재설정을 하고자 하는 메시지 버퍼의 핸들이다.

1.10.2.5.2 반환 값

메시지 버퍼가 재설정되면 pdPASS 가 반환된다. 메시지 버퍼로 보내거나 메시지 버퍼에서 읽는 것을 기다리는 task 가 차단 상태인 경우 메시지 버퍼는 재설정 되지 않고 pdFAIL 이 반환된다.

1.10.2.6 xMessageBufferIsEmpty : 메시지 버퍼가 비어 있는지 확인

```
#include “FreeRTOS.h”
#include “message_buffer.h”

BaseType_t xMessageBufferIsEmpty ( MessageBufferHandle_t xMessageBuffer );
```

메시지 버퍼가 비어있는지 확인한다.

메시지 버퍼가 메시지를 포함하지 않으면 비어있는 상태다.

메시지 버퍼 기능은 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함해야 활성화된다.

(메시지 버퍼는 스트림 버퍼를 사용한다.).

1.10.2.6.1 매개 변수

xMessageBuffer : 확인하고자 하는 메시지 버퍼의 핸들이다.

1.10.2.6.2 반환 값

메시지 버퍼가 비어있으면 pdTRUE 가 반환된다. 그렇지 않으면 pdFALSE 가 반환된다.

1.10.2.7 xMessageBufferIsFull : 메시지 버퍼가 가득 찼는지 확인

```
#include “FreeRTOS.h”
#include “message_buffer.h”

BaseType_t xMessageBufferIsFull ( MessageBufferHandle_t xMessageBuffer );
```

메시지 버퍼가 가득 찼는지 확인한다.

메시지 버퍼에서 메시지를 제거되어 공간이 확보될 때까지, 메시지 버퍼가 어떤 크기의 메시지도 더이상 허용할 수 없는 경우가 가득 찬 상태이다.

1.10.2.7.1 매개 변수

xMessageBuffer : 확인하고자 하는 메시지 버퍼의 핸들이다.

1.10.2.7.2 반환 값

메시지 버퍼가 가득 차면 pdTRUE 가 반환된다. 그렇지 않으면 pdFALSE 가 반환된다.

1.10.2.8 xMessageBufferSpacesAvailable : 메시지 버퍼 여유 공간 확인

```
#include “FreeRTOS.h”
#include “message_buffer.h”

size_t xMessageBufferSpacesAvailable ( MessageBufferHandle_t xMessageBuffer );
```

메시지 버퍼가 가득찰때까지 메시지 버퍼로 보낼 수 있는 데이터의 양과 같은 여유 공간이 얼마나 있는지 확인한다.

반환 값은 메시지 버퍼로 보낼 수 있는 최대 메시지 크기보다 4 바이트 더 크다.

메시지 버퍼 기능은 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함해야 활성화된다.

(메시지 버퍼는 스트림 버퍼를 사용한다.).

1.10.2.8.1 매개 변수

xMessageBuffer : 확인하고자 하는 메시지 버퍼의 핸들이다.

1.10.2.8.2 반환 값

메시지 버퍼가 가득 차기 전에 메시지 버퍼에 쓸 수 있는 바이트 수다. 메시지가 메시지 버퍼에 기록되면, 메시징의 길이를 저장하기 위해 sizeof(size_t)바이트가 추가로 기록된다. sizeof(size_t)는 32 비트 아키텍처에서 일반적으로 4 바이트이므로, xMessageBufferSpacesAvailable()가 10 을 반환하면 메시지 버퍼에 쓸 수 있는 가장 큰 메시지의 크기는 6 바이트(10 - 4 바이트)다.