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

1.9 스트림 버퍼 API

1.9.1 스트림 버퍼 생성 및 제거

1.9.1.1 xStreamBufferCreate : 스트림 버퍼 동적 생성

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

StreamBufferHandle_t xStreamBufferCreate ( size_t xBufferSizeBytes, size_t xTriggerLevelBytes );
```

동적으로 할당된 메모리를 사용하여 새로운 스트림 버퍼를 만든다.

정적으로 할당된 메모리(컴파일 타임에 할당되는 메모리)를 사용하는 버전은 xStreamBufferCreateStatic()을 참조한다.

xStreamBufferCreate()를 사용하려면 FreeRTOSConfig.h 의 configSUPPORT_DYNAMIC_ALLOCATION 를 1 로 설정하거나 정의되지 않은 상태로 지정해야 한다. 승트림 버퍼 기능은 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함해야 활성화 된다.

1.9.1.1.1 매개 변수

xBufferSizeBytes : 스트림 버퍼가 한번에 유지할 수 있는 총 바이트 수다.

xTriggerLevelBytes : 데이터를 기다리기 위해 스트림 버퍼에서 차단된 task 가 차단 상태에서 벗어나기 전에 스트림 버퍼에 있어야 하는 바이트 수다. 예를 들어, 트리거 레벨이 1 인 빈 스트림 버퍼를 읽을때 task 가 차단된 경우, 단일 바이트가 버퍼에 기록되거나 task 차단 시간이 만료될때까지 task 가 차단된다. 다른 예로, 트리거 레벨이 10 인 빈 스트림 버퍼의 읽기에서 task 가 차단되면 스트림 버퍼에 적어도 10 바이트가 포함되어 있거나, task 차단 시간이 만료될때까지 차단이 해제되지 않는다.

트리거 레벨에 도달하기 전에 읽기 작업의 블록 시간이 만료되더라도 작업은 실제로 사용 가능한 바이트 수를 계속 수신한다. 트리거 레벨을 0 으로 설정하면 트리거 레벨 1 이 사용된다. 버퍼 크기보다 큰 트리거 레벨을 지정하는 것은 안된다.

1.9.1.1.2 반환 값

NULL : 스트림 버퍼 데이터 구조와 저장 영역을 할당하는데 사용되는 FreeRTOS 힙 메모리가 충분하지 않기 때문에 스트림 버퍼를 생성할 수 없다.

다른 값 : 스트림 버퍼가 성공적으로 생성되었다. 반환된 값은 스트림 버퍼에 대한 핸들이다.

1.9.1.1.3 기타

```
void vAFunction( void ) {
    StreamBufferHandle_t xStreamBuffer;
    const size_t xStreamBufferSizeBytes = 100, xTriggerLevel = 10;

    // 100 바이트를 저장할 수 있는 스트림 버퍼를 만든다.
    // 스트림 버퍼 구조와 데이터를 스트림 버퍼에 보관하는데 사용되는 메모리는 동적으로 할당한다.
    xStreamBuffer = xStreamBufferCreate( xStreamBufferSizeBytes, xTriggerLevel );

    if( xStreamBuffer == NULL ) {
      // 스트림 버퍼를 생성하기에 충분한 힙 메모리 공간이 없다.
    } else {
      // 스트림 버퍼가 성공적으로 만들어 졌으므로 사용할 수 있다.
    }
}
```

1.9.1.2 xStreamBufferCreateStatic : 스트림 버퍼 정적 생성

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

StreamBufferHandle_t xStreamBufferCreateStatic ( size_t xBufferSizeBytes, size_t xTriggerLevelBytes,
                                                 uint8_t * pucStreamBufferStorageArea,
                                                 StaticStreamBuffer_t * pxStaticStreamBuffer );
```

정적 할당된 메모리를 사용하여 새로운 스트림 버퍼를 만든다. 동적으로 할당당된 메모리는 xStreamBufferCreate()를 참조한다.

xStreamBufferCreateStatic()를 사용하려면 FreeRTOSConfig .h 의 configSUPPORT_STATIC_ALLOCATION 를 1 로 설정해야 한다.
스트림 버퍼의 기능을 사용하려면 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함해야 한다.

1.9.1.2.1 매개 변수

xBufferSizeBytes : pucStreamBufferStorageArea 매개 변수가 가리키는 버퍼의 크기(바이트)다.

xTriggerLevelBytes : 데이터를 기다리기 위해 스트림 버퍼에서 차단된 task 가 차단 상태에서 벗어나기 전에 스트림 버퍼에 있어야 하는 바이트 수다. 예를 들어, 트리거 레벨이 1 인 빈 스트림 버퍼를 읽을때 task 가 차단된 경우, 단일 바이트가 버퍼에 기록되거나 task 차단 시간이 만료될때까지 task 가 차단된다. 다른 예로, 트리거 레벨이 10 인 빈 스트림 버퍼의 읽기에서 task 가 차단되면 스트림 버퍼에 적어도 10 바이트가 포함되어 있거나, task 차단 시간이 만료될때까지 차단이 해제되지 않는다.

트리거 레벨에 도달하기 전에 읽기 작업의 블록 시간이 만료되더라도 작업은 실제로 사용 가능한 바이트 수를 계속 수신한다. 트리거 레벨을 0 으로 설정하면 트리거 레벨 1 이 사용된다. 버퍼 크기보다 큰 트리거 레벨을 지정하는 것은 안된다.

pucStreamBufferStorageArea : 최소한 xBufferSizeBytes + 1 만큼 큰 uint8_t 배열을 가리켜야 한다. 이것은 스트림 버퍼에 기입될 때, 스트림이 복사되는 배열이다.

pxStaticStreamBuffer : 스트림 버퍼의 데이터 구조를 유지하는데 사용되는 StaticStreamBuffer_t 유형의 변수를 가리켜야 한다.

1.9.1.2.2 반환 값

스트림 버퍼가 성공적으로 생성되면 생성된 스트림 버퍼에 대한 핸들이 반환된다.

pucStreamBufferStorageArea 또는 pxStaticstreamBuffer 가 NULL 일 경우, NULL 이 반환된다.

1.9.1.2.3 기타

```
// 스트림을 보관하기 위해서 사용되는 배열의 치수를 지정한다. 사용 가능한 공간은 실제로 이보다 작은 999 다.
#define STORAGE_SIZE_BYTES 1000

// 실제 스트림 버퍼 내에 스트림을 보유할 메모리를 정의한다.
static uint8_t ucStorageBuffer[ STORAGE_SIZE_BYTES ];

// 스트림 버퍼 구조를 유지하기 위해 사용된 변수다.
StaticStreamBuffer_t xStreamBufferStruct;

void MyFunction( void ) {
    StreamBufferHandle_t xStreamBuffer;
    const size_t xTriggerLevel = 1;
    xStreamBuffer = xStreamBufferCreateStatic( sizeof( ucBufferStorage ), xTriggerLevel,
                                               ucBufferStorage, &xStreamBufferStruct );
```

```
    /* pucStreamBufferStorageArea 또는 pxStaticStreamBuffer 매개 변수가 NULL 이 아니므로 xStreamBuffer 는 NULL 이 아니며,
    다른 스트림 버퍼 API 호출에서 생성된 스트림 버퍼를 참조하는 데 사용할 수 있다. */

    // 스트림 버퍼를 사용하는 다른 코드는 여기에 위치할 수 있다.
}
```

1.9.1.3 vStreamBufferDelete : 스트림 버퍼 삭제

```
#include "FreeRTOS.h"
#include "stream_buffer.h"

void vStreamBufferDelete ( StreamBufferHandle_t xStreamBuffer );
```

xStreamBufferCreate()또는 xStreamBufferCreateStatic()에 대한 호출을 사용하여 이전에 작성한 스트림 버퍼를 삭제한다.

스트림 버퍼가 동적 메모리를 사용하여(xStreamBufferCreate()) 생성 된 경우 할당 된 메모리가 해제된다. 스트림 버퍼가 삭제 된 후에는 스트림 버퍼 핸들을 사용하면 안된다. 스트림 버퍼 기능은 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함해야 활성화된다.

1.9.1.3.1 매개 변수

xStreamBuffer : 삭제되는 스트림 버퍼의 핸들이다.

1.9.2 스트림 버퍼 기본 기능

1.9.2.1 xStreamBufferReceive : 스트림 버퍼 수신

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

size_t xStreamBufferReceive ( StreamBufferHandle_t xStreamBuffer, void * pvRxData, size_t xBufferLengthBytes,
                              TickType_t xTicksToWait );
```

스트림 버퍼로부터 바이트를 받는다.

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 xStreamBufferSend())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 xStreamBufferRead())에 배치하고 수신 차단 시간을 0 으로 사용해야 한다.

xStreamBufferReceive()를 사용하여 task 에서 스트림 버퍼를 읽는다. 인터럽트 서비스 루틴에서 스트림 버퍼를 읽으려면 xStreamBufferReceiveFromISR()를 사용한다.

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

1.9.2.1.1 매개 변수

xStreamBuffer : 바이트를 수신하는 스트림 버퍼의 핸들이다.

pvRxData : 수신된 바이트가 복사될 버퍼에 대한 포인터다.

xBufferLengthBytes : pvRxData 매개 변수가 가리키는 버퍼의 길이다. 하나의 호출에서 수신할 수 있는 최대 바이트 수를 설정한다. xStreamBufferReceive()는 가능한 많은 바이트를 xBufferLengthBytes 에 의해 설정된 최대 값까지 반환한다.

xTicksToWait : 스트림 버퍼가 비어있는 경우 데이터를 사용할 수 있을때까지 대기할 task 가 차단 상태로 유지되는 최대 시간이다. xStreamBufferReceive()의 xTicksToWait 가 0 이면 즉시 반환된다. 차단 시간은 tick 으로 지정되며, tick 의 절대 시간은 tick 주파수에 따라 달라진다. pdMS_TO_TICKS()매크로는 밀리 초 단위로 지정된 시간을 tick 으로 변환하는데 사용된다.

FreeRTOSConfig.h 의 INCLUDE_vTaskSuspend 가 1 로 설정된 경우, xTicksToWait 을 portMAX_DELAY 로 설정하면 task 가 시간초과 없이 무기한 대기한다. task 는 차단 상태일 때 CPU 시간을 사용하지 않는다.

1.9.2.1.2 반환 값

xBufferLengthBytes 를 사용할 수 있기 전에 xStreamBufferReceive()에 대한 호출 시간이 초과됬었을 때, xBufferLengthBytes 보다 작은 값은 스트림 버퍼에서 실제로 읽은 바이트 수다.

1.9.2.1.3 기타

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

    /* 스트림 버퍼로부터 다른 sizeof(ucRxData)바이트를 수신한다. 최대 sizeof(ucRxData)바이트 수를 사용할 수 있으려면
    차단 상태(CPU 처리 시간을 사용하지 않는다.)에서 최대 100ms 동안 대기한다.. */
    xReceivedBytes = xStreamBufferReceive( xStreamBuffer, ( void * ) ucRxData, sizeof( ucRxData ), xBlockTime );
```

```
    if( xReceivedBytes > 0 ) {
      // ucRxData 에는 여기에서 처리할수 있는 xRecievedBytes 바이트의 데이터가 포함되어 있다.
    }
}
```

1.9.2.2 xStreamBufferReceiveFromISR : ISR 에서 스트림 버퍼 수신

```
#include "FreeRTOS.h"
#include "stream_buffer.h"

size_t xStreamBufferReceiveFromISR ( StreamBufferHandle_t xStreamBuffer, void * pvRxData,
                                     size_t xBufferLengthBytes, BaseType_t * pxHigherPriorityTaskWoken );
```

스트림 버퍼에서 바이트를 수신하는 API 함수의 인터럽트 서비스 루틴 버전이다.

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 xStreamBufferSend())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 xStreamBufferRead())에 배치하고 수신 차단 시간을 0 으로 사용해야 한다.

xStreamBufferReceive()를 사용하여 task 에서 스트림 버퍼를 읽는다. 인터럽트 서비스 루틴에서 스트림 버퍼를 읽으려면 xStreamBufferReceiveFromISR()를 사용한다.

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

1.9.2.2.1 매개 변수

xStreamBuffer : 바이트를 수신하는 스트림 버퍼의 핸들이다.

pvRxData : 수신된 바이트가 복사될 버퍼에 대한 포인터다.

xBufferLengthBytes : pvRxData 매개 변수가 가리키는 버퍼의 길이다. 하나의 호출에서 수신할 수 있는 최대 바이트 수를 설정한다. xStreamBufferReceive()는 가능한 많은 바이트를 xBufferLengthBytes 에 의해 설정된 최대 값까지 반환한다.

pxHigherPriorityTaskWoken : 스트림 버퍼는 공간이 사용 가능할 때가지 task 를 차단할 수 있다. xStreamBufferReceiveFromISR()을 호출하면 사용 가능한 공간이 생기므로 공관을 확보하기 위해 대기중이던 task 가 차단 해제 상태가 될 수 있다. xStreamBufferReceiveFromISR()을 호출하면 task 의 차단 상태를 벗어나고 차단 해제된 task 의 우선순위가 현재 실행중인 task(인터럽트 된 task)보다 높을 경우, 내부적으로 xStreamBufferReceiveFromISR()은 *pxHigherPriorityTaskWoken 을 pdTRUE 로 설정한다. xStreamBufferReceiveFromISR()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.

1.9.2.2.2 반환 값

스트림 버퍼로부터 읽어진 바이트의 수(존재하는 경우)다.

1.9.2.2.3 기타

```
// 이미 생성된 스트림 버퍼다.
StreamBuffer_t xStreamBuffer;
```

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

    // pdFALSE 로 초기화한다.
    // 스트림 버퍼로부터 다음 스트림을 수신한다.
    xReceivedBytes = xStreamBufferReceiveFromISR( xStreamBuffer, ( void * ) ucRxData,
                                                  sizeof( ucRxData ), &xHigherPriorityTaskWoken );

    if( xReceivedBytes > 0 ) {
      // ucRxData 는 스트림 버퍼에서 읽은 xReceivedBytes 를 포함한다. 여기서 스트림을 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 );
}
```

1.9.2.3 xStreamBufferSend : 스트림 버퍼에 송신

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

size_t xStreamBufferSend ( StreamBufferHandle_t xStreamBuffer, void * pvRxData, size_t xDataLengthBytes,
                           TickType_t xTicksToWait );
```

스트림 버퍼에 바이트를 보낸다. 바이트는 스트림 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 xStreamBufferSend())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 xStreamBufferRead())에 배치하고 수신 차단 시간을 0 으로 사용해야 한다.

xStreamBufferSend()를 사용하면 task 에서 스트림 버퍼를 쓸 수 있다. 인터럽트 서비스 루틴에서 스트림 버퍼를 읽으려면 xStreamBufferSendFromISR()를 사용한다.

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

1.9.2.3.1 매개 변수

xStreamBuffer : 스트림이 보내지는 스트림 버퍼의 핸들이다.

pvTxData : 스트림 버퍼에 복사할 바이트를 보유하고 있는 버퍼에 대한 포인터다.

xDataLengthBytes : pvTxData 로부터 스트림 버퍼에 복사 할 수있는 최대 바이트 수다.

xTicksToWait : 스트림 버퍼에 다른 xDataLengthBytes 를 보유할 공간이 너무 적으면 task 가 스트림 버퍼에서 사용 가능하게 될 때까지 task 가 차단 상태로 유지되는 최대 시간이다. 차단 시간은 tick 으로 지정되며, tick 의 절대 시간은 tick 주파수에 따라 달라진다. pdMS_TO_TICKS()매크로는 밀리 초 단위로 지정된 시간을 tick 으로 변환하는데 사용된다. FreeRTOSConfig.h 의 INCLUDE_vTaskSuspend 가 1 로 설정된 경우, xTicksToWait 을 portMAX_DELAY 로 설정하면 task 가 시간초과 없이 무기한 대기한다.

모든 xDataLengthBytes 를 버퍼에 쓸 수 있기 전에 task 가 시간 초과되면 가능한 많은 바이트를 쓴다. task 는 차단 상태에서 CPU 시간을 사용하지 않는다.

1.9.2.3.2 반환 값

스트림 버퍼에 기록된 바이트 수다. 모든 xDataLengthBytes 를 버퍼에 쓸 수 있기 전에 task 가 시간초과되면 가능한 많은 바이트를 쓴다.

1.9.2.3.3 기타

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

    // 스트림 버퍼에 배열을 보내고, 스트림 버퍼에서 충분한 공간을 사용할 수 있을때까지 기다리는 것을 최대 100ms 동안 차단한다.
    xBytesSent = xStreamBufferSend( xStreamBuffer, ( void * ) ucArrayToSend, sizeof( ucArrayToSend ), x100ms );

    if( xBytesSent != sizeof( ucArrayToSend ) ) {
        // xStreamBufferSend()호출은 버퍼에 데이터를 쓸 수 있는 충분한 공간이 있기 전에 시간 초과하지만
        // xBytesSent 바이트를 성공적으로 작성했다.
    }

    // 문자열을 스트림 버퍼로 보낸다. 버퍼에 충분한 공간이 없는 경우 즉시 반환한다.
    xBytesSent = xStreamBufferSend( xStreamBuffer, ( void * ) pcStringToSend, strlen( pcStringToSend ), 0 );

    if( xBytesSent != strlen( pcStringToSend ) ) {
      // 버퍼에 충분한 여유 공간이 없기 때문에 전체 문자열을 스트림 버퍼에 추가할 수 없지만,
      // xBytesSent 바이트가 전송되었다. 남은 바이트를 다시 보낸다.
    }
}
```

1.9.2.4 xStreamBufferSendFromISR : ISR 에서 스트림 비트에 송신

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

size_t xStreamBufferSendFromISR ( StreamBufferHandle_t xStreamBuffer, void * pvRxData, size_t xDataLengthBytes,
                                  BaseType_t * pxHigherPriorityTaskWoken );
```

스트림 버퍼에 바이트 스트림을 보내는 인터럽트 서비스 루틴에서의 API 함수다.

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 xStreamBufferSend())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 xStreamBufferRead())에 배치하고 수신 차단 시간을 0 으로 사용해야 한다.

xStreamBufferSend()를 사용하면 task 에서 스트림 버퍼를 쓸 수 있다. 인터럽트 서비스 루틴에서 스트림 버퍼를 읽으려면 xStreamBufferSendFromISR()를 사용한다.

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

1.9.2.4.1 매개 변수

xStreamBuffer : 스트림이 보내지는 스트림 버퍼의 핸들이다.

pvTxData : 스트림 버퍼에 복사할 바이트를 보유하고 있는 버퍼에 대한 포인터다.

xDataLengthBytes : pvTxData 로부터 스트림 버퍼에 복사 할 수있는 최대 바이트 수다.

pxHigherPriorityTaskWoken : 스트림 버퍼에 데이터 대기 상태의 task 가 있을 수 있다. xStreamBufferSendFromISR()을 호출하면 데이터를 사용할 수 있게되므로, 데이터가 차단 상태를 벗어나 기다리는 task 가 발생한다. xStreamBufferSendFromISR()을 호출하면 task 의 차단 상태를 벗어나고 차단 해제된 task 의 우선순위가 현재 실행중인 task(인터럽트 된 task)보다 높을 경우, 내부적으로 xStreamBufferSendFromISR()은 *pxHigherPriorityTaskWoken 을 pdTRUE 로 설정한다. xStreamBufferSendFromISR()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.

1.9.2.4.2 반환 값

스트림 버퍼에 기록된 바이트 수다. 모든 xDataLengthBytes 를 버퍼에 쓸 수 있기 전에 task 가 시간초과 되면 가능한 많은 바이트를 송신한다.

1.9.2.4.3 기타

```
// 이미 생성된 스트림 버퍼다.
StreamBufferHandle_t xStreamBuffer;

void vAnInterruptServiceRoutine( void ) {
    size_t xBytesSent;
    char *pcStringToSend = "String to send";
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    // pdFALSE 로 초기화한다.
    // 스트림을 스트림 버퍼에 보내려고 시도한다.
    xBytesSent = xStreamBufferSendFromISR( xStreamBuffer, ( void * ) pcStringToSend,
                                           strlen( pcStringToSend ), &xHigherPriorityTaskWoken );
    if( xBytesSent != strlen( pcStringToSend ) ) {
        // 스트림 버퍼 전체 문자열이 쓰여지는 충분한 여유 공간이 없으며, xBytesSent 바이트가 기록되었다.
    }

    /* 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 );
}
```

1.9.2.5 xStreamBufferReset : 스트림 버퍼 리셋

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

BaseType_t xStreamBufferReset ( StreamBufferHandle_t xStreamBuffer );
```

스트림 버퍼를 초기 상태(비어 있는 상태)로 리셋한다. 스트림 버퍼에 있는 모든 데이터는 삭제된다. 스트림 버퍼는 스트림 버퍼와의 송수신 대기 task 가 차단되지 않을 경우에만 리셋할 수 있다.

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

1.9.2.5.1 매개 변수

xStreamBuffer : 리셋하고자 하는 스트림 버퍼의 핸들이다.

1.9.2.5.2 반환 값

스트림 버퍼가 재설정 되면 pdPASS 가 반환된다. task 가 스트림 버퍼에 송신 또는 스트림 버퍼로부터 대기중인 차단 상태의 task 가 있을 경우 리셋되지 않고 pdFAIL 을 반환한다.

1.9.2.6 xStreamBufferSetTriggerLevel : 스트림 버퍼 트리거 레벨 설정

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

BaseType_t xStreamBufferSetTriggerLevel ( StreamBufferHandle_t xStreamBuffer, size_t xTriggerLevel );
```

스트림 버퍼의 트리거 레벨은 데이터를 기다리기 위해 스트림 버퍼에서 차단된 task 가 차단 상태에서 벗어나기 전에 스트림 버퍼에 있어야 하는 바이트 수다. 예를 들어 트리거 레벨이 1 인 비어있는 스트림 버퍼를 읽을 때, task 가 차단된 상태일 경우 단일 바이트가 버퍼에 쓰여지거나, task 차단 시간이 만료되면 task 의 차단이 해제된다. 다른 예로 트리거 레벨이 10 인 빈 스트림 버퍼의 읽기에서 task 가 차단 상태면 스트림 버퍼에 적어도 10 바이트가 포함되거나, task 차단 시간이 만료될 때까지 task 가 차단해제되지 않는다. 트리거 수준에 도달하기 전에 읽기 task 의 차단 시간이 만료되면 task 는 존재하는 바이트를 실제로 받을수 있다. 트리거 레벨을 0 으로 설정하면 트리거 레벨 1 이 사용된다. 버퍼 크기보다 큰 트리거 레벨을 지정하는 것은 유효하지 않는다.

트리거 레벨은 스트림 버퍼가 생성될 때 설정되며 xStreamBufferSetTriggerLevel()을 사용하여 수정할 수 있다.

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

1.9.2.6.1 매개 변수

xStreamBuffer : 갱신하고자 하는 스트림 버퍼의 핸들이다.

xTriggerLevel : 스트림 버퍼의 새로운 트리거 레벨이다.

1.9.2.6.2 반환 값

xTriggerLevel 이 스트림 버퍼의 길이보다 작거나 같으면 트리거 레벨이 업데이트 되고 pdTRUE 가 반환된다.

아닐 경우 pdFALSE 가 반환된다.

1.9.2.7 xStreamBufferIsEmpty : 스트림 버퍼가 비어 있는지 확인

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

BaseType_t xStreamBufferIsEmpty ( StreamBufferHandle_t xStreamBuffer );
```

스트림 버퍼가 비어있는지 확인한다. 스트림 버퍼에 데이터가 포함되어 있지 않으면 비어있다.

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

1.9.2.7.1 매개 변수

xStreamBuffer : 확인하는 스트림 버퍼의 핸들이다.

1.9.2.7.2 반환 값

스트림 버퍼가 비어있으면 pdTRUE 를 반환한다.

그렇지 않으면 pdFALSE 가 반환된다.

1.9.2.8 xStreamBufferIsFull : 스트림 버퍼가 가득 찼는지 확인

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

BaseType_t xStreamBufferIsFull ( StreamBufferHandle_t xStreamBuffer );
```

스트림 버퍼가 가득 찼는지 확인한다. 스트림 버퍼에 여유 공간이 없으면 더이상 데이터를 받을 수 없다.

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

1.9.2.8.1 매개 변수

xStreamBuffer : 확인하는 스트림 버퍼의 핸들이다.

1.9.2.8.2 반환 값

스트림 버퍼가 가득 차 있으면 pdTRUE 가 리턴된다.

아닐 경우 pdFALSE 가 리턴된다.

1.9.2.9 xStreamBufferBytesAvailable : 스트림 버퍼 읽을 공간 확인

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

size_t xStreamBufferBytesAvailable ( StreamBufferHandle_t xStreamBuffer );
```

스트림 버퍼를 확인하여 스트림 버퍼가 비워지기 전에 스트림 버퍼에서 읽을 수 있는 바이트 수와 동일한 양의 데이터를 확인한다.

스트림 버퍼 기능은 FreeRTOS / source / stream_buffer.c 소스 파일을 빌드에 포함해야 활성화 된다.

1.9.2.9.1 매개 변수

xStreamBuffer : 확인하는 스트림 버퍼의 핸들이다.

1.9.2.9.2 반환 값

스트림 버퍼를 읽기 전에 스트림 버퍼에서 읽을 수 있는 바이트 수다.

1.9.2.10 xStreamBufferSpacesAvailable : 스트림 버퍼 여유 공간 확인

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

size_t xStreamBufferSpacesAvailable ( StreamBufferHandle_t xStreamBuffer );
```

스트림 버퍼가 가득 차기 전에 스트림 버퍼로 보낼수 있는 데이터의 양과 같은 여유 공간이 얼마나 남아있는 확인한다.

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

1.9.2.10.1 매개 변수

xStreamBuffer : 확인하는 스트림 버퍼의 핸들이다.

1.9.2.10.2 반환 값

스트림 버퍼가 가득 차기 전에 스트림 버퍼에 쓸 수 있는 바이트 수다.