AVR FreeRTOS : Queue Management

1. 이 장의 개요

FreeRTOS의 Task는 서로 독립된 Task로 서로 데이터를 주고 받으며(통신을 하며) 일을 처리 하여야 하는 경우가 많다.

Queue(큐)는 Task 간 통신과 동기에 이용 된다.

A. 이 장의 중요 목표

i. Queue Create 에 대한 이해.

ii. Queue를 이용한 데이터 처리에 대한 이해

iii. Queue에 데이터를 보내는 방법에 대한 이해

iv. Queue로부터 데이터를 받는 방법에 대한 이해

v. Queue에 데이터가 준비될 때 까지 Task가 기다리는 Block 상태의 이해

vi. Queue로부터 데이터를 읽거나 쓸 때 priority의 영향

이 장에서는 Task 간의 통신에 대하여만 다룬다.

Task-to-Interrupt와 Interrupt-to-Task Communication은 3장에서 설명한다.

2. Queue의 특성

A. Data Storage

i. Queue는 일정한 크기의 유한한 Data를 저장 한다. Queue 가 저장할 수 있는 데이터의 최대 개수를 Queue Length라고 한다. Queue의 Length와 Size는 Queue가 생성될 때 설정 된다.

ii. 보통 Queue는 First In First Out(FIFO) 버퍼로 사용 된다. 그러나 Queue의 앞 부분에 Data를 쓰는 것도 가능 하다.

B. Queue는 어떤 특별한 Task에 소유 되지 않기 때문에 여러 개의 Task에서 같은 Queue에 데이터를 쓸 수 있다. 그리고 같은 Queue에서 여러 개의 Task가 데이터를 읽을 수 있다. 그러나 동일한 Queue에서 여러 개의 Task가 데이터를 읽는 경우는 드문 일 이다.

C. Task가 Queue로부터 데이터를 읽을 때 필요에 따라 옵션으로 Block Time을 선언 할 수 있다. 이 경우 Queue에 읽을 수 있는 데이터가 준비 될 때까지 Task는 Blocking State가 된다. Task의 Blocking 상태는 다른 Task나 Interrupt에서 Queue에 데이터를 저장 하면 자동으로 해제 되어 Ready 상태가 된다. 또한 Queue 가 비어 있는 상태에서 지정한 Block Time 이 지난 경우에도 Ready 상태가 된다.

만약 여러 개의 Task가 Queue로부터 데이터를 받기 위하여 Blocked 상태에 있는 경우에 Queue에 새로운 데이터가 하나 도달 하면 가장 우선 순위(Priority)가 높은 하나의 Task만 Unblocked 상태(Ready 상태)로 된다. 만약 동일한 Priority를 갖는 Task가 여러 개 대기 중일 경우에는 가장 오래 대기한 Task가 Unblocked 된다.

3. Queue의 사용

A. xQueueCreate() API Function

Queue는 사용되기 전에 Create되어야 한다.

Queue는 xQueueHandle 형 변수를 이용 하여 참조 한다. xQueueCreate() 함수는 Queue를 생성하고 xQueueHandle을 Return 한다.

FreeRTOS는 Queue의 Data 구조와 Data 저장 장소로 RAM을 사용한다. 만약 Queue 생성에 필요한 RAM 공간이 부족한 경우에는 NULL을 Return 한다.

xQueueCreate() API Function의 Prototype

```
xQueueHandle xQueueCreate(unsigned portBASE_TYPE uxQueueLength, unsigned portBASE_TYPE uxItemSize );
```

Parameter Name/ Returned value	Description
uxQueueLength	Queue가 저장할 수 있는 최대 Item 수
uxItemSize	Queue에 저장 되는 각 Data Item의 Byte Size
Returned value	RAM 부족으로 Queue가 Create 되지 못한 경우 NULL 이 Return 된다. Queue가 성공적으로 Create 된 경우 Created Queue의 Handle 이 Return 된다.

B. xQueueSendToBack()과 xQueueSendToFront() API 함수

xQueueSendToBack()는 Data를 Queue의 Back(Tail)에 보내고, xQueueSendToFront()는 Queue의 Front(Head)에 Data를 보낸다.

xQueueSend()는 xQueueSendToBack() 함수와 정확히 같은 함수이다.

주의: Interrupt Service Routine에서는 xQueueSendToBack()과 xQueueSendToFront() API 함수를 사용 하여서는 안 된다. 대신 Interrupt-safe version인 xQueueSendToBackFromISR()과 xQueueSendToFrontFROMISR() 함수를 사용 하여야 한다.

xQueueSendToFront() API Function Prototype

```
portBASE_TYPE xQueueSendToFront( xQueueHandle xQueue, const void * pvItemToQueue, portTickType xTicksToWait );
```

xQueueSendToBack() API Function Prototype

```
portBASE_TYPE xQueueSendToBack( xQueueHandle xQueue, const void * pvItemToQueue, portTickType xTicksToWait );
```

Parameter Name/ Returned value	Description
xQueue	Queue의 Handle, Queue Handle은 Queue가 생성될 때 Return 된다.
pvItemToQueue	Queue에 복사될 Data의 Pointer 각 Item의 Size는 Queue 가 생성될 때 결정 된다.
xTickToWait	Task가 Blocked State에서 Queue에 Data를 저장 할 수 있는 공간이 생길 때 까지 기다리는 최대 Tick 수 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: Data가 성공적으로 Queue에 저장 된 경우 errQUEUE_FULL: Queue 가 Full 상태라 Data가

	Queue에 쓰여지지 못한 경우

C. xQueueReceive()와 xQueuePeek() APL Functions

xQueueReceive() 함수는 Queue로부터 하나의 Item을 받는데 사용 한다. 받은 Item은 Queue로부터 삭제 된다.

xQueuePeek() 함수도 Queue로부터 하나의 Item을 받는데 사용 된다. 그러나 받은 Item은 Queue로부터 삭제 되지 않고 Queue에 그대로 남이 있게 된다.

주의: Interrupt Service Routine에서는 xQueueReceive()과 xQueuePeek() API 함수를 사용 하여서는 안 된다. 대신 Interrupt-safe version인 xQueueReceiveFromISR() 함수를 사용 하여야 한다.

xQueueReceive() API Function Prototype

```
portBASE_TYPE xQueueReceive( xQueueHandle xQueue, const void * pvBuffer, portTickType xTicksToWait );
```

xQueuePeek() API Function Prototype

```
portBASE_TYPE xQueuePeek( xQueueHandle xQueue, const void * pvBuffer, portTickType xTicksToWait );
```

Parameter Name/ Returned value	Description
xQueue	Queue의 Handle, Queue Handle은 Queue가 생성될 때 Return 된다.
pvBuffer	Queue로부터 Memory에 복사될 Data의 Pointer 각 Item의 Size는 Queue 가 생성될 때 결정 된다.
xTickToWait	Task가 Blocked State에서 Queue가 비어 있는 경우 Queue에 유용한 Data가 준비 될 때 까지 기다리는 최대 Tick 수 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: Queue로 부터 성공적으로 Data를 읽은 경우 errQUEUE_EMPTY: Queue 가 Empty 상태라 Queue로 부터 Data를 읽지 못한 경우 만약 Block Time이 설정된 경우(xTickToWait 가 0 가 아닌 경우) 이 시간 동안 다른 Task나 Interrupt로부터 Queue에 Data사 보내 지기를 기다린다.

D. uxQueueMessageWaiting() API Function

uxQueueMessageWaiting() 현재 Queue에 저장 되어 있는 Item의 수를 구하는데 이용 된다.

주의: Interrupt Service Routine에서는 uxQueueMessageWaiting() API 함수를 사용 하여서는 안 된다. 대신 Interrupt-safe version인 uxQueueMessageWaitingFromISR() 함수를 사용 하여야 한다.

xQueueMessageWaiting() API Function Prototype

unsigned portBASE_TYPE xQueueMessageWaiting(xQueueHandle xQueue);

Parameter Name/ Returned value	Description
xQueue	Queue의 Handle, Queue Handle은 Queue가 생성될 때 Return 된다.
Returned value	현재 Queue에 저장 되어 있는 Item의 수 Queue 가 비어 있는 경우 zero가 Return 된다.

E. 여러 개의 Task로부터 Data가 Queue에 보내지고 하나의 Task가 Queue로부터 Data를 Receive 하는 예.

이 예에서 Data를 보내는 Task는 Block Time을 설정 하지 않고, Priority 도 Data를 받는 Task에 비교 하여 낮게 설정 하였다.

그 결과 Queue에는 1개 이상의 Data가 저장 되지 않고 Data 가 저장 되는 즉시 받는 Task가 Data를 바로 가져 기서 이용 하게 된다

Ex. 10 프로그램

F. Queue에 Data와 Data Source함께 전송 하기

하나의 Queue가 여러 개의 Source로부터 Data를 받는 경우 Data가 어디로부터 오는 것인지 알 필요가 있는 경우가 있다. 이 경우 Data와 Data Source 정보를 포함한 구조체를 Queue를 이용 하여 전달 한다.

G. Queue를 이용 하여 Data 와 Data Source 정보를 포함한 구조체를 전송 하는 경우 발생 하는 Blocking 예제

이 예는 앞의 예와 유사 하다. 그러나 Data를 보내는 Task의 Task Priority가 높고 받는 Task 가 낮도록 하기 때문에 Task Priority는 반대이다.

보내는 Task의 Priority를 높게 하는 것은 구조체를 Queue에 보내기 위하여는 연속하여 여러 개의 Item을 보내 수 있어야 하기 때문이다.

그 결과 Queue가 Full 상태가 될 수 있고 이 경우 보내는 Task 가 Blocking State가 된다.

예제 프로그램 11

4. 크기가 큰 Data를 서로 다른 Task가 서로 이용 하여야 하는 경우

Task 간에 전송하여야 할 Data의 크기가 큰 경우 Queue를 이용 하여 직접

Data를 전송하는 것 보다 Pointer를 이용 하는 것이 처리 시간과 Queue 생성에 필요한 RAM 크기 등에서 보다 효과 적이다.

이 경우 아래와 같은 주의가 필요 하다.

A. 서로 다른 Task가 Pointer를 사용 하여 Data를 이용 하는 경우 현재 시간에 Data(RAM)를 사용 할 수 있는 Task(Owner)에 대한 정의가 명확 하여야 한다.

   두 개의 Task가 동시에 Data를 변경 하는 경우 Data의 유용성에 문제가 생길 수 있다.

   이 문제를 해결 하는 방법으로 Data Pointer를 전송하는 Task는 Pointer를 전송 하기 전 까지만 Data를 사용 할 수 있고, Data pointer를 받는 Task는 Pointer를 받은 다음에만 Data를 사용 할 수 있도록 하는 것이 한 방법 이다.

B. 만약 Pointer를 이용 하여 사용 하는 Memory 가 Dynamically Allocated Memory 인 경우 에는 오직 한 Task만이 Memory를 해제 할 수 있고, 해제된 Memory를 더 이상 참조 할 수 없다.

C. Task Stack 상에 Data Memory가 Allocated 된 경우에는 보다 주의를 필요로 한다. 이 경우 Stack Frame이 변경 되면 Data는 더 이상 유효 하지 않게 된다.