M480 FreeRTOS Queue

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This code is a basic application for FreeRTOS queue creating to communicate between tasks.
BSP Version	M480 Series BSP CMSIS V3.04.000
Hardware	NuMaker-PFM-M487

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 Function Description

1.1 Introduction

This FreeRTOS sample code is a basic application for queue. It creates two tasks that communicate over a single queue. One task acts as a producer, the other a consumer.

1.2 Principle

Queues are underlying mechanism beyond all tasks communication in a FreeRTOS environment. Queues are able to be read and written by several different tasks, and don't belong to any task in particular. When a task wants to read in a queue, it is moved to "Blocked" state and wait data has been written in the queue by another task then moved back to "Ready". A task can also specify a maximum waiting time for the queue. After this time, the task switches back to "Ready" state.

This FreeRTOS sample code creates two tasks. One task acts as a producer, the other a consumer. The producer loops for three iteration, posting an incrementing number onto the queue each cycle. It then delays for a fixed period before doing exactly the same again. The consumer loops emptying the queue. Each item removed from the queue is checked to ensure it contains the expected value. When the queue is empty it blocks for a fixed period, and then does the same again. All queue access is performed without blocking. The consumer completely empties the queue each time it runs so the producer should never find the queue full.

1.3 Demo Result

```
FreeRTOS is starting ...
Check Task is running ...

Queue Send:    0
Queue Send:    1
Queue Send:    2
Queue Receive: 0
Queue Receive: 1
Queue Receive: 2

Queue Send:    3
Queue Send:    4
Queue Send:    5
Queue Receive: 3
Queue Receive: 4
Queue Receive: 5
```

2 Code Description

Create two tasks for producer and consumer :

```
void vStartPolledQueueTasks( UBaseType_t uxPriority )
{
static QueueHandle_t xPolledQueue;

    /* Create the queue used by the producer and consumer. */
    xPolledQueue = xQueueCreate( pollqQUEUE_SIZE, ( UBaseType_t ) sizeof( uint16_t ) );

    if( xPolledQueue != NULL )
    {
        /* vQueueAddToRegistry() adds the queue to the queue registry, if one is
        in use.  The queue registry is provided as a means for kernel aware
        debuggers to locate queues and has no purpose if a kernel aware debugger
        is not being used.  The call to vQueueAddToRegistry() will be removed
        by the pre-processor if configQUEUE_REGISTRY_SIZE is not defined or is
        defined to be less than 1. */
        vQueueAddToRegistry( xPolledQueue, "Poll_Test_Queue" );

        /* Spawn the producer and consumer. */
        xTaskCreate( vPolledQueueConsumer, "QConsNB", pollqSTACK_SIZE, ( void * )
       &xPolledQueue, uxPriority-1, ( TaskHandle_t * ) NULL );
        xTaskCreate( vPolledQueueProducer, "QProdNB", pollqSTACK_SIZE, ( void * )
       &xPolledQueue, uxPriority, ( TaskHandle_t * ) NULL );
    }
}
```

Set producer in the task function :

```
static portTASK_FUNCTION(vPolledQueueProducer, pvParameters)
{
    uint16_t usValue = (uint16_t) 0;
    BaseType_t xError = pdFALSE, xLoop;

    for(;;) {
        printf("\n");
        for(xLoop = 0; xLoop < pollqVALUES_TO_PRODUCE; xLoop++) {
            /* Send an incrementing number on the queue without blocking. */
```

```
            if(xQueueSend(*((QueueHandle_t *) pvParameters), (void *) &usValue,
            pollqNO_DELAY) != pdPASS) {
                /* We should never find the queue full so if we get here there
                has been an error. */
                xError = pdTRUE;
            } else {
                if(xError == pdFALSE) {
                    /* If an error has ever been recorded we stop incrementing the
                    check variable. */
                    portENTER_CRITICAL();
                    xPollingProducerCount++;
                    printf("Queue Send:    %d\n",usValue);
                    portEXIT_CRITICAL();
                }

                /* Update the value we are going to post next time around. */
                usValue++;
            }
        }

        /* Wait before we start posting again to ensure the consumer runs and
        empties the queue. */
        vTaskDelay(pollqPRODUCER_DELAY);
    }
}
```

Set consumer in the task function：

```
static portTASK_FUNCTION(vPolledQueueConsumer, pvParameters)
{
    uint16_t usData, usExpectedValue = (uint16_t) 0;
    BaseType_t xError = pdFALSE;

    for(;;) {
        if(xQueueReceive(*((QueueHandle_t *) pvParameters), &usData, pollqCONSUMER_DELAY)
        == pdPASS) {
            if(usData != usExpectedValue) {
                /* This is not what we expected to receive so an error has
                occurred. */
                xError = pdTRUE;
```

```
                /* Catch-up to the value we received so our next expected
                value should again be correct. */
                usExpectedValue = usData;
            } else {
                if(xError == pdFALSE) {
                    /* Only increment the check variable if no errors have
                    occurred. */
                    portENTER_CRITICAL();
                    xPollingConsumerCount++;
                    printf("Queue Receive: %d\n",usData);
                    portEXIT_CRITICAL();
                }
            }

            /* Next time round we would expect the number to be one higher. */
            usExpectedValue++;
        }
        /* Now the queue is empty we block, allowing the producer to place more
        items in the queue. */
    }
}
```

3 Software and Hardware Environment

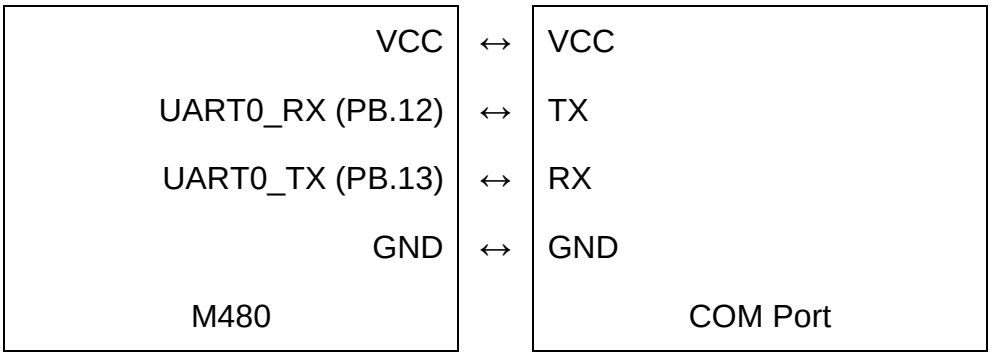
Software Environment

- BSP version
  - M480 Series BSP CMSIS V3.04.000
- IDE version
  - Keil uVersion 5.26

Hardware Environment

- Circuit components
  - NuMaker-PFM-M487 v3.0
- Diagram

Using printf function to output message and the UART0 TX RX pin are PB.13 and PB.12 respectively.



4 Directory Information

- EC_M480_FreeRTOS_Queue_V1.00
  - Library — Sample code header and source files
    - CMSIS — Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
    - Device — CMSIS compliant device header file
    - StdDriver — All peripheral driver header and source files
  - SampleCode
    - ExampleCode — Source file of example code
  - ThirdParty
    - FreeRTOS — A real time operating system available for free download. Its official website is: http://www.freertos.org/

5 How to Execute Example Code

1. Browsing into sample code folder by Directory Information (section 4) and double click M480_FreeRTOS_Queue.uvproj.
2. Enter Keil compile mode
   a. Build
   b. Download
   c. Start/Stop debug session
3. Enter debug mode
   a. Run

6 Revision History

Date	Revision	Description
Jul. 11, 2019	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.