Lab: MSP432P401R FreeRTOS Queue Introduction

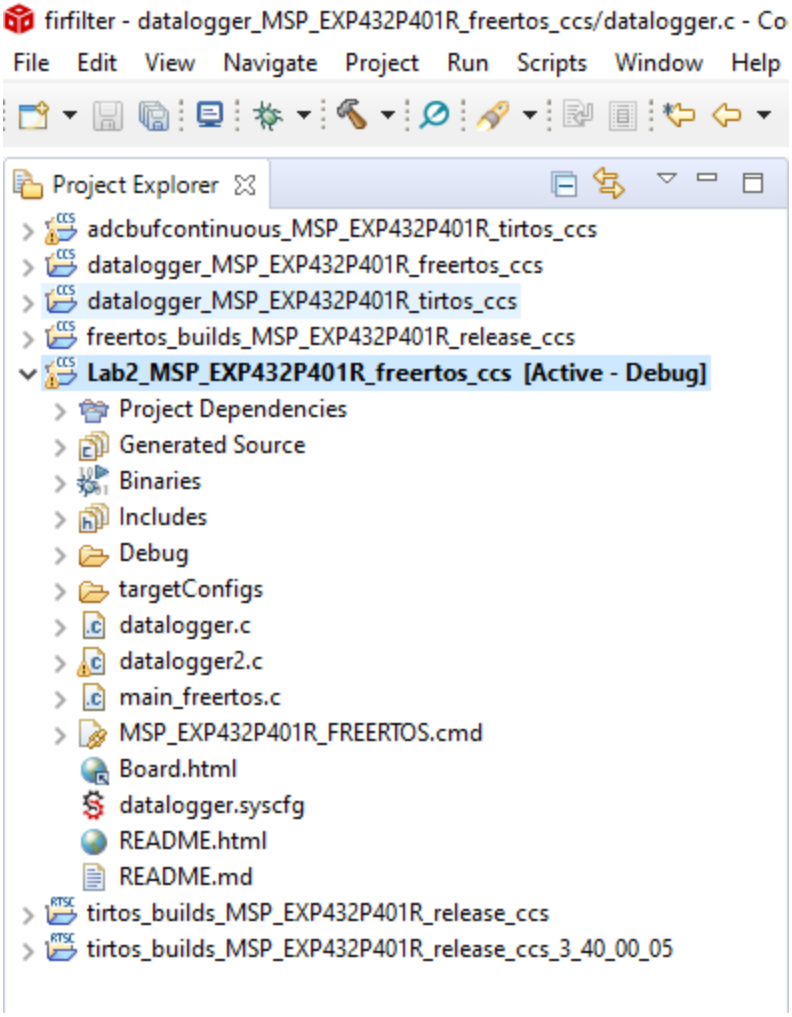
Introduction

This lab assumes you are at least familiar with the MSP432P401R and Code Composer. It will also assume that you have created a simple FreeRTOS project inside of the Code Composer studio, and have two tasks, one that simply blinks an LED at a certain rate, the other that changes the LED based on the pressing of a button.

In this lab you'll learn how to add a queue to communicate between the two processes.

Initial Setup

For this lab you will start with the project from the last lab. Copy this project and give it a new name, for example Lab2_MSP_EXP432P401R_freertos_ccs, like this:

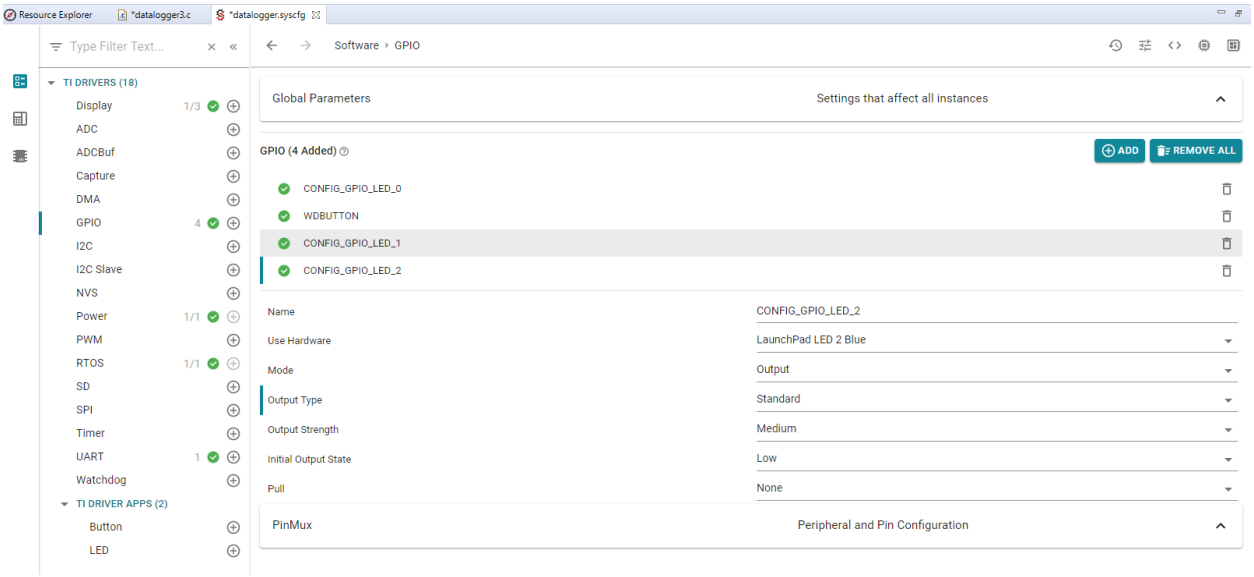


You shouldn't need to change any file names, but to make sure it happened correctly Build the Project and run it on your MSP432. When you run it for the first time it may ask you for the Debug Configuration to use, you can use the one from the last lab's project name.

Adding another Task to the Code

In this lab you will need to add one more task, in this case that will blink yet another LED but at a variable rate that is passed through a queue from the task that is synchronized to the Button press. The first step to make this happen is to add another LED to the system.

- Use the SysConfig tool to do this by selecting the .syscfg file.
- Now GPIO selection, then fill out the details to use the Blue LED:



Now you can add the CONFIG_GPI_LED_2 to your code as the Blue LED.

Now you'll need to add yet another task to the system. To do this go ahead and add another task by copying the datalogger2.c file. You can call it datalogger3.c. Edit this file to change the LED to the new LED you just created:

```
/* For usleep() */
#include <unistd.h>
#include <stdint.h>
#include <stddef.h>

/* Driver Header files */
#include <ti/drivers/GPIO.h>
#include <ti/display/Display.h>
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/Watchdog.h>
```

```c
/* Driver configuration */
#include "ti_drivers_config.h"

/*
 *  ======== mainThread ========
 */
void *mainThread3(void *arg0)
{

    /* Call driver init functions */
    GPIO_init();


    // I2C_init();
    // SPI_init();
    // UART_init();
    // Watchdog_init();

    /* Configure the LED pin */
    GPIO_setConfig(CONFIG_GPIO_LED_2, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);


    /* Turn on user LED */
    GPIO_write(CONFIG_GPIO_LED_2, CONFIG_GPIO_LED_ON);

    while (1) {
        vTaskDelay(2000);
        GPIO_toggle(CONFIG_GPIO_LED_2);
    }
}
```

You may want to rebuild this, just to make sure you didn't make any mistakes. Now that you have this function, you can add the task to the main_freertos.c. Here is that code (note that this is not the entire file, just the changed part):

```c
/* RTOS header files */
#include <FreeRTOS.h>
#include <task.h>

/* Driver configuration */
#include <ti/drivers/Board.h>

extern void *mainThread(void *arg0);
extern void *mainThread2(void *arg0);
extern void *mainThread3(void *arg0);
/* Stack size in bytes */
#define THREADSTACKSIZE   1024

/*
 *  ======== main ========
 */
int main(void)
{
```

```
    pthread_t           thread;
    pthread_attr_t      attrs;
    struct sched_param  priParam;
    int                 retc;

    pthread_t           thread1;
    pthread_attr_t      attrs1;
    struct sched_param  priParam1;

    pthread_t           thread2;
    pthread_attr_t      attrs2;
    struct sched_param  priParam2;

    /* initialize the system locks */
#ifdef __ICCARM__
    __iar_Initlocks();
#endif

    /* Call driver init functions */
    Board_init();

    /* Initialize the attributes structure with default values */
    pthread_attr_init(&attrs);


    /* Set priority, detach state, and stack size attributes */
    priParam.sched_priority = 1;
    retc = pthread_attr_setschedparam(&attrs, &priParam);
    retc |= pthread_attr_setdetachstate(&attrs, PTHREAD_CREATE_DETACHED);
    retc |= pthread_attr_setstacksize(&attrs, THREADSTACKSIZE);
    if (retc != 0) {
        /* failed to set attributes */
        while (1) {}
    }

    retc = pthread_create(&thread, &attrs, mainThread, NULL);
    if (retc != 0) {
        /* pthread_create() failed */
        while (1) {}
    }

    /* Initialize the attributes structure with default values */
    pthread_attr_init(&attrs1);

    /* Set priority, detach state, and stack size attributes */
    priParam1.sched_priority = 1;
    retc = pthread_attr_setschedparam(&attrs1, &priParam1);
    retc |= pthread_attr_setdetachstate(&attrs1, PTHREAD_CREATE_DETACHED);
    retc |= pthread_attr_setstacksize(&attrs1, THREADSTACKSIZE);
    if (retc != 0) {
        /* failed to set attributes */
        while (1) {}
    }

    retc = pthread_create(&thread1, &attrs1, mainThread2, NULL);
```

```
    if (retc != 0) {
        /* pthread_create() failed */
        while (1) {}
    }

    /* Initialize the attributes structure with default values */
    pthread_attr_init(&attrs2);

    /* Set priority, detach state, and stack size attributes */
    priParam2.sched_priority = 1;
    retc = pthread_attr_setschedparam(&attrs2, &priParam2);
    retc |= pthread_attr_setdetachstate(&attrs2, PTHREAD_CREATE_DETACHED);
    retc |= pthread_attr_setstacksize(&attrs2, THREADSTACKSIZE);
    if (retc != 0) {
        /* failed to set attributes */
        while (1) {}
    }

    retc = pthread_create(&thread2, &attrs2, mainThread3, NULL);
    if (retc != 0) {
        /* pthread_create() failed */
        while (1) {}
    }
    /* Start the FreeRTOS scheduler */
    vTaskStartScheduler();

    return (0);
}
```

Now you can build and run the code. Unfortunately the Green and Blue LED are shared, but you should see your system running correctly.

But let's do this. Let's change the code so that Process 3 flashes the RED LED for a prescribed time, and Process 1 doesn't flash the LED but sends data over a queue to Process 3 to change the time that the RED LED flashes.

First, let's remove the flashing LED from Process 1. Change datalogger.c like this:

```
/* For usleep() */
#include <unistd.h>
#include <stdint.h>
#include <stddef.h>
#include <FreeRTOS.h>
#include <semphr.h>
/* Driver Header files */
#include <ti/drivers/GPIO.h>
#include <ti/display/Display.h>
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/Watchdog.h>

/* Driver configuration */
```

```
#include "ti_drivers_config.h"
SemaphoreHandle_t xMutex;



void sw2callback(uint_least8_t index)
{
    xSemaphoreGiveFromISR( xMutex, NULL);

 }

/*
 *  ======== mainThread ========
 */
void *mainThread(void *arg0)
{

    /* Call driver init functions */
    GPIO_init();
    xMutex = xSemaphoreCreateBinary();

    // I2C_init();
    // SPI_init();
    // UART_init();
    // Watchdog_init();

    /* Configure the LED pin */
  //  GPIO_setConfig(CONFIG_GPIO_LED_0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);

  //  GPIO_enableInt(WDBUTTON);

    /* Turn on user LED */
    GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_ON);

    while (1) {
        xSemaphoreTake( xMutex, portMAX_DELAY );
        vTaskDelay(400);
     //   GPIO_toggle(CONFIG_GPIO_LED_0);

    }
}
```

Now change datalogger3.c to flash the RED LED, like this:

```
/* For usleep() */
#include <unistd.h>
#include <stdint.h>
#include <stddef.h>

/* Driver Header files */
#include <ti/drivers/GPIO.h>
#include <ti/display/Display.h>
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SPI.h>
```

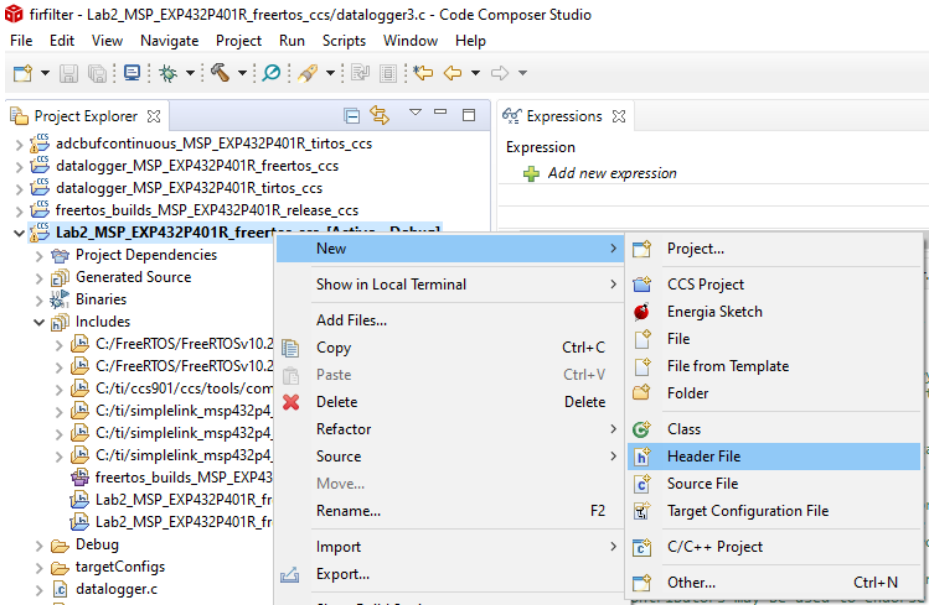
```
// #include <ti/drivers/UART.h>
// #include <ti/drivers/Watchdog.h>

/* Driver configuration */
#include "ti_drivers_config.h"

/*
 *  ======== mainThread ========
 */
void *mainThread3(void *arg0)
{

    /* Call driver init functions */
    GPIO_init();
    Display_init();
    Display_Handle hSerial = Display_open(Display_Type_UART, NULL);

    // I2C_init();
    // SPI_init();
    // UART_init();
    // Watchdog_init();

    /* Configure the LED pin */
    GPIO_setConfig(CONFIG_GPIO_LED_0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);


    /* Turn on user LED */
    GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_ON);

    while (1) {
        vTaskDelay(2000);
        GPIO_toggle(CONFIG_GPIO_LED_0);
    }
}
```

Build and Run the code, now the Green and RED leds should flash, but pressing the switch will not be indicated on the unit.

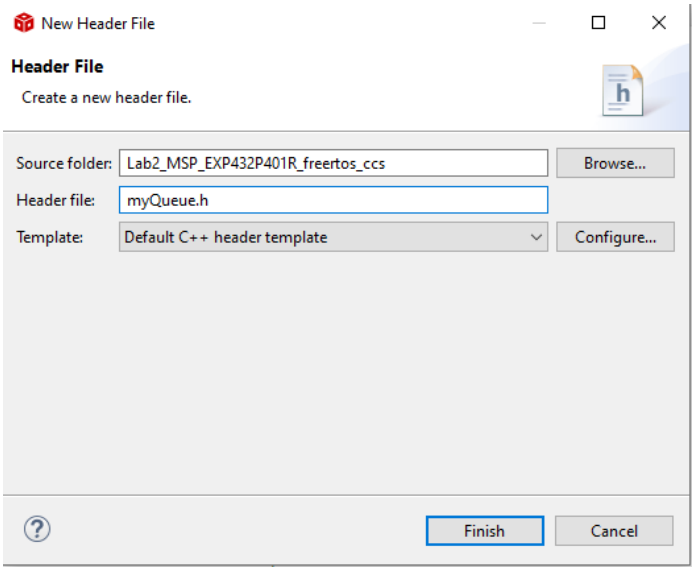
Adding a Queue

Now you will add a queue between Process 1 and Process 3 to pass some data (in this case the length of time to flash the LED).

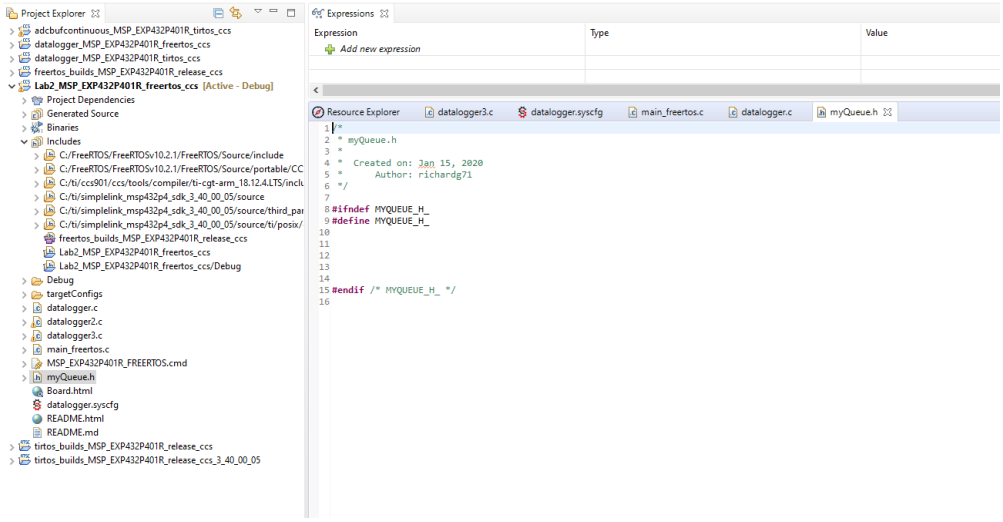
To do this you'll first add a header file to add the proper header stuff. Right Click on your project, then select New->Header File:



Now fill out the information:



And you should now see your new file in the project, and it will be opened for editing.



You'll need to add some simple includes and a global so you can define and use your queue. Modify the code like this:

```
/*
 * myQueue.h
 *
 *  Created on: Jan 15, 2020
 *      Author: richardg71
 */

#ifndef MYQUEUE_H_
#define MYQUEUE_H_

#include <FreeRTOS.h>
#include <semphr.h>

QueueHandle_t myLEDQueue;


#endif /* MYQUEUE_H_ */
```

Here you have created a data structure so that you can create and access a queue.

The next step is to create the Queue, you'll do that in main_freertos.c. Make these changes:

First, add the myQueue.h to the includes:

```
/* RTOS header files */
#include <FreeRTOS.h>
#include <task.h>
#include "myQueue.h"
/* Driver configuration */
#include <ti/drivers/Board.h>
```

Now add some code to create the queue:

```
    /* Call driver init functions */
```

```
    Board_init();

    myLEDQueue = xQueueCreate( mainQUEUE_LENGTH, sizeof( int) );

    /* Initialize the attributes structure with default values */
    pthread_attr_init(&attrs);
```

You may want to build and run the code, just to make sure it works correctly.

Now let's add the queue to the two processes. First let's edit process 1 to send the data (the file datalogger.c). First add the header file:

```
/* For usleep() */
#include <unistd.h>
#include <stdint.h>
#include <stddef.h>
#include <FreeRTOS.h>
#include <semphr.h>
#include "myQueue.h"
/* Driver Header files */
#include <ti/drivers/GPIO.h>
#include <ti/display/Display.h>
```

Add an integer variable count to the process, initialize it to 1000. Then each time the switch is pressed we'll add 1000 to the value, and then send it through the queue:

```
void *mainThread(void *arg0)
{
    int count = 1000;
    /* Call driver init functions */
    GPIO_init();
    xMutex = xSemaphoreCreateBinary();

    // I2C_init();
    // SPI_init();
    // UART_init();
    // Watchdog_init();


    GPIO_enableInt(WDBUTTON);


    while (1) {
        xSemaphoreTake( xMutex, portMAX_DELAY );

        count = count += 1000;
        if (count > 5000)
            count = 1000;
        configASSERT(myLEDQueue);
        xQueueSend( myLEDQueue, &count, portMAX_DELAY );
        vTaskDelay(400);
    }
}
```

Now you'll need to add some code to task 3 to receive the data from the queue. So first add the queue.h file to datalogger3.c:

```
/* Driver Header files */
#include <ti/drivers/GPIO.h>
#include <ti/display/Display.h>
#include "myQueue.h"
```

And now update the process to take data from the queue:

```
*  ======== mainThread ========
 */
void *mainThread3(void *arg0)
{
    int count = 100;
    /* Call driver init functions */
    GPIO_init();
    Display_init();
    Display_Handle hSerial = Display_open(Display_Type_UART, NULL);

    // I2C_init();
    // SPI_init();
    // UART_init();
    // Watchdog_init();

    /* Configure the LED pin */
    GPIO_setConfig(CONFIG_GPIO_LED_0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);


    /* Turn on user LED */
    GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_OFF);

    while (1) {
        configASSERT(myLEDQueue);
        xQueueReceive( myLEDQueue, &count,  portMAX_DELAY);
        GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_ON);
        vTaskDelay(count);
        GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_OFF);
    }
}
```

When you build and run the code the green light should flash thanks to task 2. When you press the right switch you should see the RED LED flash and then go off. Each time you press the switch you should see it stay on for a longer amount of time.

Debugging FREERTOS

Perhaps it would be useful to show how the FREERTOS system is set up to help you resolve problems. Most of this is done by the source code having specific ASSERTS in the code to catch when you are using the code incorrectly. Let's look at an example.

Let's say that instead of using the`xMutex = xSemaphoreCreateBinary();` command to create a Binary semaphore for your synchronization process, you use `xMutex = xSemaphoreCreateMutex();` instead. If you edit datalogger.c and make this change, then build and run the program and then press the right hand switch the program should stop.

When it is stopped select the

Run->Suspend and you will see the queue.c code displayed, with this line marked:

```
	/* Normally a mutex would not be given from an interrupt, especially if
	there is a mutex holder, as priority inheritance makes no sense for an
	interrupts, only tasks. */
	configASSERT( !( ( pxQueue->uxQueueType == queueQUEUE_IS_MUTEX ) && ( pxQueue->u.xSemaphore.xMutexHolder != NULL ) ) );
```

This warns you that you should not be using a MUTEX semaphore from an ISR, and hopefully leads you to make the change from a Mutex to a Binary sephamore.

Lab Submission:

Submit a short video of your lab working.