San José State University
Department of Computer Engineering
CMPE 146-03, Real-Time Embedded System Co-Design, Fall 2019
Lab Assignment 7

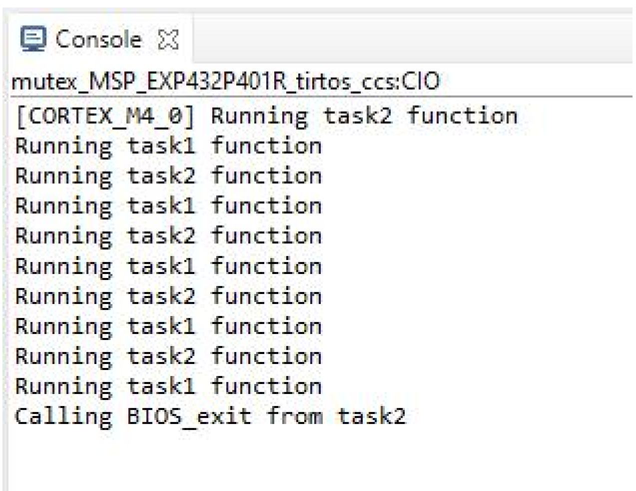
Name: Chenghui Xue

Exercise 1

In this exercise, I will practice with TI's real-time operating system, TI-RTOS. I need to create tasks to do some work. Also, we need to use TI driver to communicate with UART. In exercise 1, I will use TI-RTOS to control two LEDs.

Exercise 1.1

In the exercise 1.1, I need to build the local project and run the code. Here is the result:



Exercise 1.2

In this exercise1.2, I need to use taskFxn() blink one LED and task2Fxn() blink a different LED. And I need to use while loop to blink the LED. Turn on the LED 2s, and turn it off for 2s, and repeat.

In my mind, I choose LED1(1.0) and LED2(2.0) as my output. Here is the code:

```
/*set GPIO pin*/
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN0);
```

I create while loop for both task1Fxn and task2Fxn(). First, I set high to LED1(pin1.1), Between the blink, I use Task_sleep(2000) to suspend the task for 2s. Same as LED2(pin2.0)

Here is the code:

```
Void task1Fxn(UArg arg0, UArg arg1)
{
    while(1){
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
        Task_sleep(2000);
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
        Task_sleep(2000);
    }
}
```

Exercise 1.3

In the exercise1.3, we will use semaphore to synchronize the execution of the two tasks. In each tasks, I added Semaphore_pend(semHandle, BIOS_WAIT_FOREVER) after the blink in order to get the semaphore. After that, I use Semaphore_post(semHandle) to increase the semaphore's count so that the wairing task can be made ready again. Here is the code for task1:

```
Void task1Fxn(UArg arg0, UArg arg1)
{
        for (;;) {
            System_printf("Running task1 function\n");
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
            Task_sleep(2000);
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
            Task_sleep(2000);

            if (Semaphore_getCount(semHandle) == 0) {
                System_printf("Sem blocked in task1\n");
            }

            /* Get access to resource */
            Semaphore_pend(semHandle, BIOS_WAIT_FOREVER);


            /* Do work on locked resource */
            resource += 1;
            /* Unlock resource */

            Semaphore_post(semHandle);

        }
}
```

And source code of task2 only the GPIO pin different.

Exercise 2

Exercise 2.1

In this exercise, I will add python to control the LED turning on or off. I modified task1Fxn(), and add uart to the task1. Let Uart read the input which given by Python script. I find that Python script will send an ASCII ‘1’ and ‘0’ which is 49 and 48 to the LaunchPad. Therefore, I write a condition to check which input uart received. Here is the code of task1Fxn():

```
Void task1Fxn(UArg arg0, UArg arg1)
{
    char        input;
    UART_Handle uart;
    UART_Params uartParams;

    UART_init();
    /* Create a UART with data processing off. */
    UART_Params_init(&uartParams);
   uartParams.writeDataMode = UART_DATA_BINARY;
   uartParams.readDataMode = UART_DATA_BINARY;
   uartParams.readReturnMode = UART_RETURN_FULL;
   uartParams.readEcho = UART_ECHO_OFF;
   uartParams.baudRate = 115200;

   uart = UART_open(CONFIG_UART_0, &uartParams);

    while(1){
        UART_read(uart, &input,1);
        if(input == 49)
        {
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
        }
        else if(input == 48)
        {
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
        }
        printf("%d\n", input);
    }
}
```

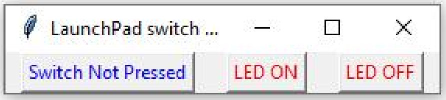
Exercise 2.2

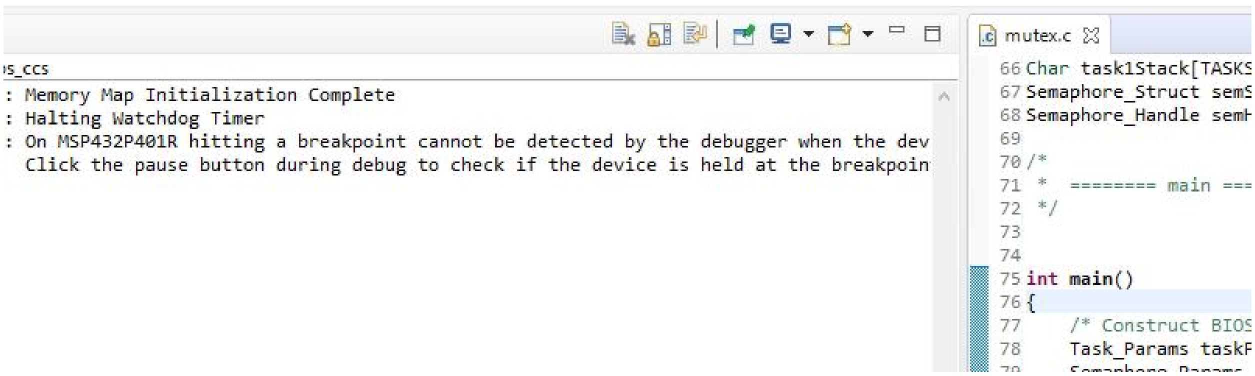
I create two status variable named bool status and bool last. therefore, we can compare the status with last as if condition. I set the default condition is false (Status = false), which means the button is not pressed. Here is the code of task2Fxn():

```
Void task2Fxn(UArg arg0, UArg arg1)
{
```

```
    char        input;
    UART_Handle uart;
    UART_Params uartParams;
    bool status = false; // set default button not pressed
    bool last = true;
    UART_init();
    /* Create a UART with data processing off. */
    UART_Params_init(&uartParams);
   uartParams.writeDataMode = UART_DATA_BINARY;
   uartParams.readDataMode = UART_DATA_BINARY;
   uartParams.readReturnMode = UART_RETURN_FULL;
   uartParams.readEcho = UART_ECHO_OFF;
   uartParams.baudRate = 115200;

   uart = UART_open(CONFIG_UART_0, &uartParams);
   const char  echoPrompt[] = "OPEN\n";
   const char  echoPrompt1[] = "CLOSE\n";
//   UART_write(uart, echoPrompt, sizeof(echoPrompt));
        while(1) //check status
        {
            Task_sleep(100);
            //UART_write(uart, echoPrompt, sizeof(echoPrompt));
            status = false;    // precondition false
            if(status == false){
                if(status == last){
                Task_sleep(2000);
                UART_write(uart, echoPrompt, sizeof(echoPrompt));
                }
                else if(status!= last)
                {
                UART_write(uart, echoPrompt, sizeof(echoPrompt));
                last = status;
                }

            }
            UART_read(uart, &input, 1);
            if(input == 49 || 48)
            {
                status = true;
                UART_write(uart, echoPrompt1, sizeof(echoPrompt1));
                last = status;

            }

        }


}
```

Result output:





Source Code of Exercise 1:

```
/*
 *  ======== mutex.c ========
 */

/* XDC module Headers */
#include <xdc/std.h>
#include <xdc/runtime/System.h>

/* BIOS module Headers */
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>

#include <ti/drivers/Board.h>
#define __MSP432P4XX__
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
#define TASKSTACKSIZE   512

Void task1Fxn(UArg arg0, UArg arg1);
Void task2Fxn(UArg arg0, UArg arg1);

Int resource = 0;
Int finishCount = 0;
UInt32 sleepTickCount;
```

```
Task_Struct task1Struct, task2Struct;
Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE];
Semaphore_Struct semStruct;
Semaphore_Handle semHandle;

/*
 *  ======== main ========
 */
int main()
{
    /* Construct BIOS objects */
    Task_Params taskParams;
    Semaphore_Params semParams;

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

    /*set GPIO pin*/
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN0);

    /* Construct writer/reader Task threads */
    Task_Params_init(&taskParams);
    taskParams.stackSize = TASKSTACKSIZE;
    taskParams.stack = &task1Stack;
    taskParams.priority = 1;
    Task_construct(&task1Struct, (Task_FuncPtr)task1Fxn, &taskParams, NULL);

    taskParams.stack = &task2Stack;
    taskParams.priority = 2;
    Task_construct(&task2Struct, (Task_FuncPtr)task2Fxn, &taskParams, NULL);

    /* Construct a Semaphore object to be use as a resource lock, inital count 1 */
    Semaphore_Params_init(&semParams);
    Semaphore_construct(&semStruct, 1, &semParams);

    /* Obtain instance handle */
    semHandle = Semaphore_handle(&semStruct);

    /* We want to sleep for 10000 microseconds */
    //sleepTickCount = 10000 / Clock_tickPeriod;

    BIOS_start();    /* Does not return */
    return(0);
}

/*
 *  ======== task1Fxn ========
 */
Void task1Fxn(UArg arg0, UArg arg1)
{
    //    UInt32 time;
    //
        for (;;) {
```

```
            System_printf("Running task1 function\n");
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
            Task_sleep(2000);
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
            Task_sleep(2000);

            if (Semaphore_getCount(semHandle) == 0) {
                System_printf("Sem blocked in task1\n");
            }

            /* Get access to resource */
            Semaphore_pend(semHandle, BIOS_WAIT_FOREVER);


            /* Do work on locked resource */
            resource += 1;
            /* Unlock resource */

            Semaphore_post(semHandle);

        }

//    while(1){
//        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
//        Task_sleep(2000);
//        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
//        Task_sleep(2000);
//    }


}

/*
 *  ======== task2Fxn ========
 */
Void task2Fxn(UArg arg0, UArg arg1)
{
    for (;;) {
        System_printf("Running task2 function\n");
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0);
        Task_sleep(2000);
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
        Task_sleep(2000);

        if (Semaphore_getCount(semHandle) == 0) {
            System_printf("Sem blocked in task2\n");
        }

        /* Get access to resource */
        Semaphore_pend(semHandle, BIOS_WAIT_FOREVER);

        /* Do work on locked resource */
        resource += 1;
        /* Unlock resource */
```

```
        Semaphore_post(semHandle);
    }

//    while(1){
//        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0);
//        Task_sleep(2000);
//        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
//        Task_sleep(2000);
//    }

}
```

Source code for Exercise 2:

```
/*
 *  ======== mutex.c ========
 */

/* XDC module Headers */
#include <xdc/std.h>
#include <xdc/runtime/System.h>
#include <ti/drivers/UART.h>

#include "ti_drivers_config.h"
#include<stdio.h>
/* BIOS module Headers */
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/drivers/GPIO.h>
#include <ti/drivers/Board.h>
#include <stdint.h>
#include <stddef.h>
#define __MSP432P4XX__
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
#define TASKSTACKSIZE   10240
#include <stdbool.h>

Void task1Fxn(UArg arg0, UArg arg1);
Void task2Fxn(UArg arg0, UArg arg1);

Int resource = 0;
Int finishCount = 0;
UInt32 sleepTickCount;

Task_Struct task1Struct, task2Struct;
Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE];
Semaphore_Struct semStruct;
Semaphore_Handle semHandle;

/*
```

```
 *  ======== main ========
 */

int main()
{
    /* Construct BIOS objects */
    Task_Params taskParams;
    Semaphore_Params semParams;

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

    /*set GPIO pin*/
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);

    /* Construct writer/reader Task threads */
    Task_Params_init(&taskParams);
    taskParams.stackSize = TASKSTACKSIZE;
 //  taskParams.stack = &task1Stack;
//   taskParams.priority = 1;
//    Task_construct(&task1Struct, (Task_FuncPtr)task1Fxn, &taskParams, NULL);

    taskParams.stack = &task2Stack;
    taskParams.priority = 2;
    Task_construct(&task2Struct, (Task_FuncPtr)task2Fxn, &taskParams, NULL);

    /* Construct a Semaphore object to be use as a resource lock, inital count 1 */
    Semaphore_Params_init(&semParams);
    Semaphore_construct(&semStruct, 1, &semParams);

    /* Obtain instance handle */
    semHandle = Semaphore_handle(&semStruct);

    /* We want to sleep for 10000 microseconds */
    sleepTickCount = 10000 / Clock_tickPeriod;

    BIOS_start();    /* Does not return */
    return(0);


}

/*
 *  ======== task1Fxn ========
 */
Void task1Fxn(UArg arg0, UArg arg1)
{
//    char        input;
//    UART_Handle uart;
//    UART_Params uartParams;
//
//    UART_init();
//    /* Create a UART with data processing off. */
//    UART_Params_init(&uartParams);
```

```
//   uartParams.writeDataMode = UART_DATA_BINARY;
//   uartParams.readDataMode = UART_DATA_BINARY;
//   uartParams.readReturnMode = UART_RETURN_FULL;
//   uartParams.readEcho = UART_ECHO_OFF;
//   uartParams.baudRate = 115200;
//
//   uart = UART_open(CONFIG_UART_0, &uartParams);
//
//    while(1){
//        UART_read(uart, &input,1);
//        if(input == 49)
//        {
//            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
//        }
//        else if(input == 48)
//        {
//            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
//        }
//        printf("%d\n", input);
//    }
}
Void task2Fxn(UArg arg0, UArg arg1)
{
    char        input;
    UART_Handle uart;
    UART_Params uartParams;
    bool status = false; // set default
    bool last = true;
    UART_init();
    /* Create a UART with data processing off. */
    UART_Params_init(&uartParams);
   uartParams.writeDataMode = UART_DATA_BINARY;
   uartParams.readDataMode = UART_DATA_BINARY;
   uartParams.readReturnMode = UART_RETURN_FULL;
   uartParams.readEcho = UART_ECHO_OFF;
   uartParams.baudRate = 115200;

   uart = UART_open(CONFIG_UART_0, &uartParams);
   const char  echoPrompt[] = "OPEN\n";
   const char  echoPrompt1[] = "CLOSE\n";
//   UART_write(uart, echoPrompt, sizeof(echoPrompt));
        while(1) //check status
        {
            Task_sleep(100);
            //UART_write(uart, echoPrompt, sizeof(echoPrompt));
            status = false;    // precondition false
            if(status == false){
                if(status == last){
                Task_sleep(2000);
                UART_write(uart, echoPrompt, sizeof(echoPrompt));
                }
                else if(status!= last)
                {
                UART_write(uart, echoPrompt, sizeof(echoPrompt));
                last = status;
```

```
            }

        }
        UART_read(uart, &input, 1);
        if(input == 49 || 48)
        {
            status = true;
            UART_write(uart, echoPrompt1, sizeof(echoPrompt1));
            last = status;

        }

    }


}
```