Emanuel Donie Ypon
CMPE 146

Lab Report 7

Exercise 1

Task functions:

Task 1

```
Void task1(UArg arg0, UArg arg1)
{
    // setup red LED
    /* Configuring P2.0 outputs */
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0); // red

    /* Bring LED to low */
    GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0); // red

    while (1) {

        // blink LED
        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);

        // delay for 1000ms = 1 Hz
        Task_sleep(1000);
    }
}
```

Task 2

```
Void task2(UArg arg0, UArg arg1)
{
    // setup green LED
    /* Configuring P2.0 outputs */
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1); // green

    /* Bring LED to low */
    GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1); // green

    while (1) {

        // blink LED
        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P2, GPIO_PIN1);

        // delay for 4000ms = 0.25 Hz
        Task_sleep(4000);
    }
}
```

To control the blinking, I used the DriverLib function `Task_sleep()` to act as a delay. This function takes in an integer value in ms. To compute the arguments, I used the function T = 1 / F to get the appropriate value in ms. To blink the red LED at a rate of 1 Hz, you must make the task sleep for **1000 ms**. To blink the green LED at a rate of 0.25 Hz, you must make the task sleep for **4000 ms**.

The entire program is in the appendix.

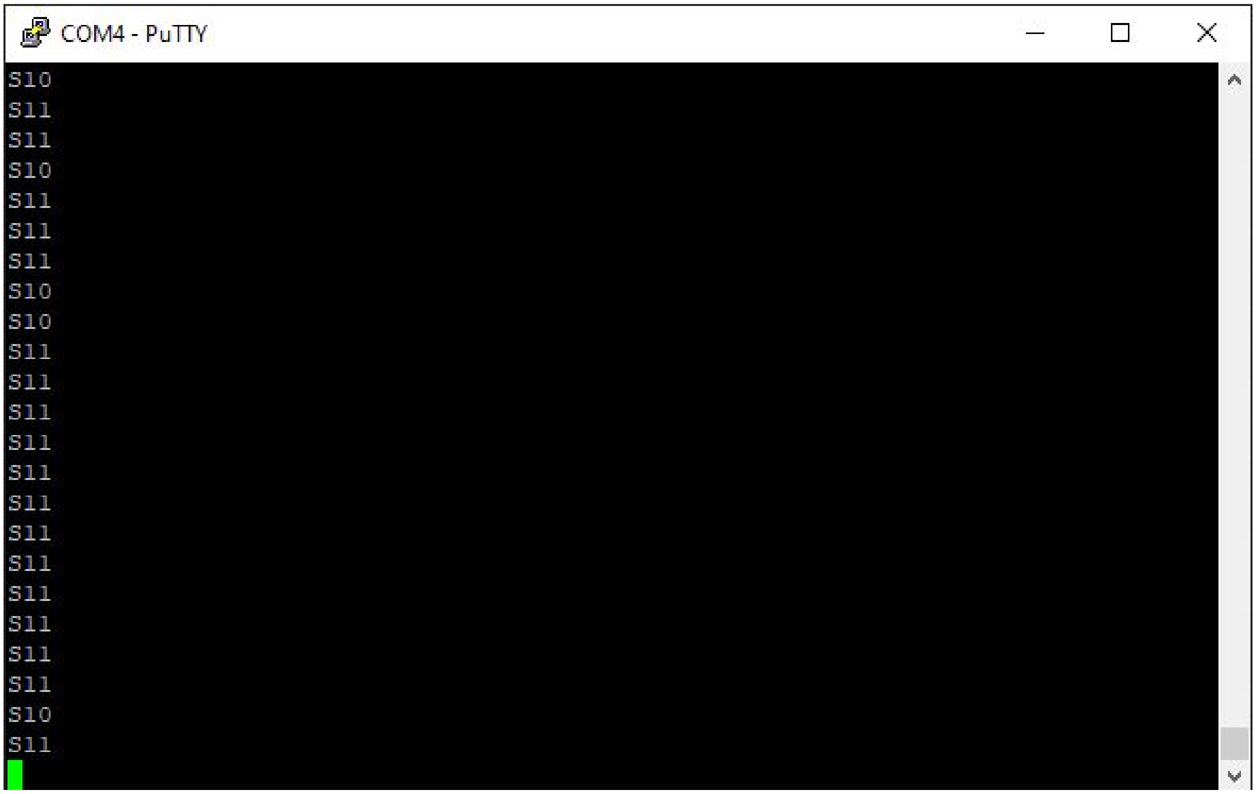
Exercise 2.1

Task function:

```
Void task1(UArg arg0, UArg arg1)
{
    printf("Task1\n");

    UART_Handle uart;
    UART_Params uartParams;

    UART_init();    // Driver init

    // Set up communication parameters and open the device
    UART_Params_init(&uartParams);
    uartParams.readEcho = UART_ECHO_OFF;
    uart = UART_open(CONFIG_UART_0, &uartParams);

    if (uart == NULL) {
        printf("Failed to open UART.\n");
        while (1);
    }

    // setup button
    GPIO_setAsInputPinWithPullUpResistor (GPIO_PORT_P1, GPIO_PIN1);

    uint32_t status;
    char buffer[10];

    while (1) {
        status = GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1);
        sprintf(buffer, "S1%u\n", status);
        UART_write(uart, buffer, strlen(buffer));
        Task_sleep(100);
    }
}
```

I referenced a sample program from the Resource Explorer which uses the button switches as an input to help set up and access the buttons on the LaunchPad. I set up the button at the beginning of the task, and within the forever while loop I read the status of the button. When the button is pressed, I print out "**S10**" in the terminal. When the button is not pressed, I print out "**S11**" in the terminal.

Screenshot of PuTTY terminal:



```
S10
S11
S11
S10
S11
S11
S11
S10
S10
S11
S11
S11
S11
S11
S11
S11
S11
S11
S11
S11
S11
S10
S11
```

The **"S10"** outputs represent when the button is pressed. The **"S11"** outputs represent when the button is not pressed.

The entire program is in the appendix.

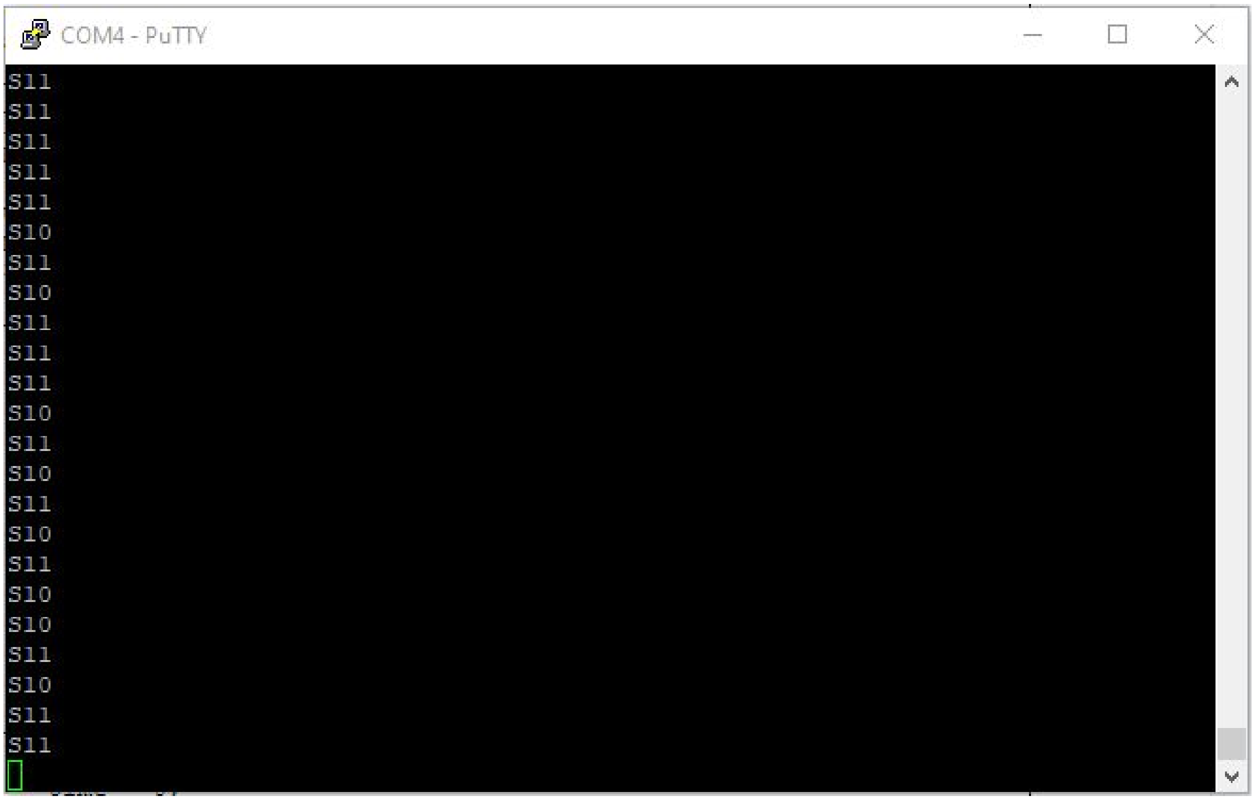
Exercise 2.2

Task function:

```
Void task1(UArg arg0, UArg arg1)
{
    printf("Task1\n");

    UART_Handle uart;
    UART_Params uartParams;

    UART_init();    // Driver init

    // Set up communication parameters and open the device
    UART_Params_init(&uartParams);
    uartParams.readEcho = UART_ECHO_OFF;
    uart = UART_open(CONFIG_UART_0, &uartParams);

    if (uart == NULL) {
        printf("Failed to open UART.\n");
        while (1);
    }

    // setup button
    GPIO_setAsInputPinWithPullUpResistor (GPIO_PORT_P1, GPIO_PIN1);

    // prevStatus keeps track of previous state
    // time used to keep track of how long since last state change
    uint32_t status, prevStatus = 1;
    int time = 0;
    char buffer[10];

    while (1) {
        status = GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1);
        sprintf(buffer, "S1%u\n", status);
        if (status != prevStatus || time >= 50) {
            UART_write(uart, buffer, strlen(buffer));
            time = 0;
        }

        // set prevStatus to current status
        prevStatus = status;

        // increment time
        time++;

        Task_sleep(100);
    }
}
```

In this updated task, to avoid sending the same message again and again I keep track of the previous state and current state. If both of these are the same, then I do not print anything. However, if nothing has been sent in > 5 seconds, I print out the current state. To do this, I keep track of the time elapsed with the variable, *time*. After each iteration, I increment time. If the calculated value is greater than 5s, I then send the current state.

Screenshot of PuTTY terminal:



```
S11
S11
S11
S11
S11
S10
S11
S10
S11
S11
S11
S10
S11
S10
S11
S10
S11
S10
S10
S11
S10
S11
S11
```

When the button is not pressed, “**S11**” is printed out. If no state change occurs in 5 seconds, “**S11**” or “**S10**” is printed out again but with an indication that it was automatically printed. When the button is pressed, we print out "**S10**”. Here, I left the button unpressed, and after 5 seconds a repeating “**S11**” is printed out. Alternatively, if I hold the button a repeated “**S10**” is printed out after 5 seconds.

The entire program is in the appendix.

Exercise 2.3

Task functions:

Task1()

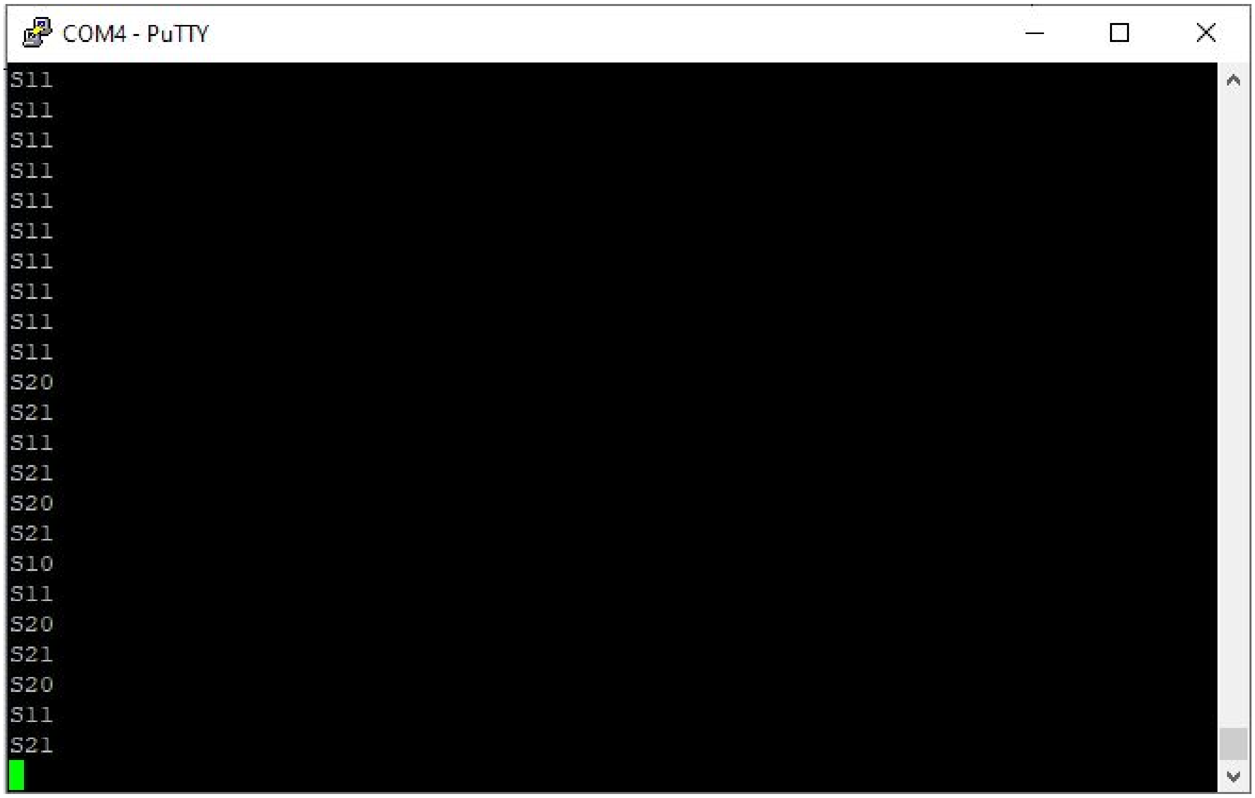
```
Void task1(UArg arg0, UArg arg1)
{
    printf("Task1\n");

    // setup button
    GPIO_setAsInputPinWithPullUpResistor (GPIO_PORT_P1, GPIO_PIN1);

    // prevStatus keeps track of previous state
    // time used to keep track of how long since last state change
    uint32_t status, prevStatus = 1;
    int time = 0;
    char buffer[10];

    while (1) {
        status = GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1);
        sprintf(buffer, "S1%u\n", status);
        if (status != prevStatus || time >= 50) {
            UART_write(uart, buffer, strlen(buffer));
            time = 0;
        }

        // set prevStatus to current status
        prevStatus = status;

        // increment time
        time++;

        Task_sleep(100);
    }
}
```

Task2()

```
Void task2(UArg arg0, UArg arg1)
{
    printf("Task2\n");

    // setup button
    GPIO_setAsInputPinWithPullUpResistor (GPIO_PORT_P1, GPIO_PIN4);

    // prevStatus keeps track of previous state
    // time used to keep track of how long since last state change
    uint32_t status, prevStatus = 1;
    int time = 0;
    char buffer[10];

    while (1) {
        status = GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN4);
        sprintf(buffer, "S2%u\n", status);
        if (status != prevStatus || time >= 50) {
            UART_write(uart, buffer, strlen(buffer));
            time = 0;
        }

        // set prevStatus to current status
        prevStatus = status;

        // increment time
        time++;

        Task_sleep(100);
    }
}
```

Since we cannot initialize or open the UART multiple times, for both tasks to have access to the same UART, I declared the UART variables and its parameters and opened the UART inside the main function. This enables us to interface both buttons with the UART.

Screenshot of PuTTY terminal window:



```
S11
S11
S11
S11
S11
S11
S11
S11
S11
S11
S20
S21
S11
S21
S20
S21
S10
S11
S20
S21
S20
S11
S21
```

When button 1 or button 2 is pressed, "**S10**" and "**S20**" are printed, respectively. When there has been no state change for neither button, then it is the state is printed out automatically. No-state change updates will be checked for both pressed/not pressed and is shown for both.

The entire program is in the appendix.

Exercise 2.4

Task function:

```
Void task3(UArg arg0, UArg arg1)
{
    // setup LED
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN2);
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);

    uint8_t UARTval;
    while (1) {
        // read UART state
        UART_read(uart, &UARTval, 1);
        printf("%u", UARTval);

        // toggle LED
        if(UARTval == 49) {
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2);
        } else if (UARTval == 48) {
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);
        }

        Task_sleep(100);
    }
}
```

The entire program is in the appendix.

Appendix

Exercise 1

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

/* 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   2048

Void task1(UArg arg0, UArg arg1);
Void task2(UArg arg0, UArg arg1);

Task_Struct task1Struct, task2Struct;
Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE];

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

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

    /* Construct task threads */
    Task_Params_init(&taskParams);
    taskParams.stackSize = TASKSTACKSIZE;
    taskParams.stack = &task1Stack;
    Task_construct(&task1Struct, (Task_FuncPtr)task1, &taskParams, NULL);

    taskParams.stack = &task2Stack;
    Task_construct(&task2Struct, (Task_FuncPtr)task2, &taskParams, NULL);

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

// blink red
Void task1(UArg arg0, UArg arg1)
```

```
{
    // setup red LED
    /* Configuring P2.0 outputs */
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0); // red

    /* Bring LED to low */
    GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0); // red

    while (1) {

        // blink LED
        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);

        // delay for 1000ms = 1 Hz
        Task_sleep(1000);
    }
}

// blink green
Void task2(UArg arg0, UArg arg1)
{
    // setup green LED
    /* Configuring P2.0 outputs */
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1); // green

    /* Bring LED to low */
    GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1); // green

    while (1) {

        // blink LED
        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P2, GPIO_PIN1);

        // delay for 4000ms = 0.25 Hz
        Task_sleep(4000);
    }
}
```

Exercise 2.1

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

/* 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>

#include <ti/drivers/UART.h>
#include "ti_drivers_config.h"

#define TASKSTACKSIZE   2048

Void task1(UArg arg0, UArg arg1);
Void task2(UArg arg0, UArg arg1);

Task_Struct task1Struct, task2Struct;
Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE];

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

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

    /* Construct task threads */
    Task_Params_init(&taskParams);
    taskParams.stackSize = TASKSTACKSIZE;
    taskParams.stack = &task1Stack;
    Task_construct(&task1Struct, (Task_FuncPtr)task1, &taskParams, NULL);

    taskParams.stack = &task2Stack;
    Task_construct(&task2Struct, (Task_FuncPtr)task2, &taskParams, NULL);

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

Void task1(UArg arg0, UArg arg1)
{
    printf("Task1\n");

    UART_Handle uart;
    UART_Params uartParams;

    UART_init();    // Driver init

    // Set up communication parameters and open the device
    UART_Params_init(&uartParams);
    uartParams.readEcho = UART_ECHO_OFF;
    uart = UART_open(CONFIG_UART_0, &uartParams);

    if (uart == NULL) {
        printf("Failed to open UART.\n");
        while (1);
    }
    // setup button
    GPIO_setAsInputPinWithPullUpResistor (GPIO_PORT_P1, GPIO_PIN1);

    uint32_t status;
    char buffer[10];

    while (1) {
        status = GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1);
        sprintf(buffer, "S1%u\n", status);
        UART_write(uart, buffer, strlen(buffer));
        Task_sleep(100);
    }
}

Void task2(UArg arg0, UArg arg1)
{
    printf("Task2\n");
}
```

Exercise 2.2

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

/* 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>

#include <ti/drivers/UART.h>
#include "ti_drivers_config.h"

#define TASKSTACKSIZE   2048

Void task1(UArg arg0, UArg arg1);
Void task2(UArg arg0, UArg arg1);

Task_Struct task1Struct, task2Struct;
Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE];

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

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

    /* Construct task threads */
    Task_Params_init(&taskParams);
    taskParams.stackSize = TASKSTACKSIZE;
    taskParams.stack = &task1Stack;
    Task_construct(&task1Struct, (Task_FuncPtr)task1, &taskParams, NULL);

    taskParams.stack = &task2Stack;
    Task_construct(&task2Struct, (Task_FuncPtr)task2, &taskParams, NULL);

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

Void task1(UArg arg0, UArg arg1)
{
    printf("Task1\n");

    UART_Handle uart;
    UART_Params uartParams;

    UART_init();    // Driver init

    // Set up communication parameters and open the device
    UART_Params_init(&uartParams);
    uartParams.readEcho = UART_ECHO_OFF;
    uart = UART_open(CONFIG_UART_0, &uartParams);

    if (uart == NULL) {
        printf("Failed to open UART.\n");
        while (1);
    }

    // setup button
    GPIO_setAsInputPinWithPullUpResistor (GPIO_PORT_P1, GPIO_PIN1);

    // prevStatus keeps track of previous state
    // time used to keep track of how long since last state change
    uint32_t status, prevStatus = 1;
    int time = 0;
    char buffer[10];

    while (1) {
        status = GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1);
        sprintf(buffer, "S1%u\n", status);
        if (status != prevStatus || time >= 50) {
            UART_write(uart, buffer, strlen(buffer));
            time = 0;
        }

        // set prevStatus to current status
        prevStatus = status;

        // increment time
        time++;

        Task_sleep(100);
    }
}

Void task2(UArg arg0, UArg arg1)
{
    printf("Task2\n");
}
```

Exercise 2.3

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

/* 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>

#include <ti/drivers/UART.h>
#include "ti_drivers_config.h"

#define TASKSTACKSIZE   2048

Void task1(UArg arg0, UArg arg1);
Void task2(UArg arg0, UArg arg1);

Task_Struct task1Struct, task2Struct;
Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE];

UART_Handle uart;
UART_Params uartParams;

int main()
{
    /* Construct BIOS objects */
    Task_Params taskParams;
    /* Call driver init functions */
    Board_init();

    UART_init();    // Driver init

    // Set up communication parameters and open the device
    UART_Params_init(&uartParams);
    uartParams.readEcho = UART_ECHO_OFF;
    uart = UART_open(CONFIG_UART_0, &uartParams);

    if (uart == NULL) {
        printf("Failed to open UART.\n");
        while (1);
    }


    /* Construct task threads */
    Task_Params_init(&taskParams);
    taskParams.stackSize = TASKSTACKSIZE;
    taskParams.stack = &task1Stack;
    Task_construct(&task1Struct, (Task_FuncPtr)task1, &taskParams, NULL);

    taskParams.stack = &task2Stack;
    Task_construct(&task2Struct, (Task_FuncPtr)task2, &taskParams, NULL);

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

Void task1(UArg arg0, UArg arg1)
{
    printf("Task1\n");

    // setup button
    GPIO_setAsInputPinWithPullUpResistor (GPIO_PORT_P1, GPIO_PIN1);

    // prevStatus keeps track of previous state
    // time used to keep track of how long since last state change
    uint32_t status, prevStatus = 1;
    int time = 0;
    char buffer[10];

    while (1) {
        status = GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1);
        sprintf(buffer, "S1%u\n", status);
        if (status != prevStatus || time >= 50) {
            UART_write(uart, buffer, strlen(buffer));
            time = 0;
        }

        // set prevStatus to current status
        prevStatus = status;

        // increment time
        time++;
        Task_sleep(100);
    }
}

Void task2(UArg arg0, UArg arg1)
{
    printf("Task2\n");

    // setup button
    GPIO_setAsInputPinWithPullUpResistor (GPIO_PORT_P1, GPIO_PIN4);

    // prevStatus keeps track of previous state
    // time used to keep track of how long since last state change
    uint32_t status, prevStatus = 1;
    int time = 0;
    char buffer[10];

    while (1) {
        status = GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN4);
        sprintf(buffer, "S2%u\n", status);
        if (status != prevStatus || time >= 50) {
            UART_write(uart, buffer, strlen(buffer));
            time = 0;
        }

        // set prevStatus to current status
        prevStatus = status;

        // increment time
        time++;

        Task_sleep(100);
    }
}
```

Exercise 2.4

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

/* 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>

#include <ti/drivers/UART.h>
#include "ti_drivers_config.h"

#define TASKSTACKSIZE   2048

Void task1(UArg arg0, UArg arg1);
Void task2(UArg arg0, UArg arg1);
Void task3(UArg arg0, UArg arg1);

Task_Struct task1Struct, task2Struct, task3Struct;
Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE],
task3Stack[TASKSTACKSIZE];

UART_Handle uart;
UART_Params uartParams;

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

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

    UART_init();    // Driver init

    // Set up communication parameters and open the device
    UART_Params_init(&uartParams);
    uartParams.readEcho = UART_ECHO_ON;
    uart = UART_open(CONFIG_UART_0, &uartParams);

    if (uart == NULL) {
        printf("Failed to open UART.\n");
        while(1);
    }

    /* Construct task threads */
    Task_Params_init(&taskParams);
    taskParams.stackSize = TASKSTACKSIZE;

    taskParams.stack = &task1Stack;
    Task_construct(&task1Struct, (Task_FuncPtr)task1, &taskParams, NULL);

    taskParams.stack = &task2Stack;
    Task_construct(&task2Struct, (Task_FuncPtr)task2, &taskParams, NULL);

    taskParams.stack = &task3Stack;
    Task_construct(&task3Struct, (Task_FuncPtr)task3, &taskParams, NULL);

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

Void task1(UArg arg0, UArg arg1)
```

```
{
    printf("Task1\n");

    // setup button
    GPIO_setAsInputPinWithPullUpResistor (GPIO_PORT_P1, GPIO_PIN1);

    // prevStatus keeps track of previous state
    // time used to keep track of how long since last state change
    uint32_t status, prevStatus = 1;
    int time = 0;
    char buffer[10];

    while (1) {
        status = GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1);
        sprintf(buffer, "S1%u\n", status);
        if (status != prevStatus || time >= 50) {
            UART_write(uart, buffer, strlen(buffer));
            time = 0;
        }

        // set prevStatus to current status
        prevStatus = status;

        // increment time
        time++;

        Task_sleep(100);
    }
}

Void task2(UArg arg0, UArg arg1)
{
    printf("Task2\n");

    // setup button
    GPIO_setAsInputPinWithPullUpResistor (GPIO_PORT_P1, GPIO_PIN4);

    // prevStatus keeps track of previous state
    // time used to keep track of how long since last state change
    uint32_t status, prevStatus = 1;
    int time = 0;
    char buffer[10];

    while (1) {
        status = GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN4);
        sprintf(buffer, "S2%u\n", status);
        if (status != prevStatus || time >= 50) {
            UART_write(uart, buffer, strlen(buffer));
            time = 0;
        }

        // set prevStatus to current status
        prevStatus = status;

        // increment time
        time++;
        Task_sleep(100);
    }
}

Void task3(UArg arg0, UArg arg1)
{
    // setup LED
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN2);
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);

    uint8_t UARTval;

    while (1) {
        // read UART state
        UART_read(uart, &UARTval, 1);
        printf("%u", UARTval);

        // toggle LED
        if(UARTval == 49) {
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2);
        } else if (UARTval == 48) {
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);
        }

        Task_sleep(100);
    }
}
```