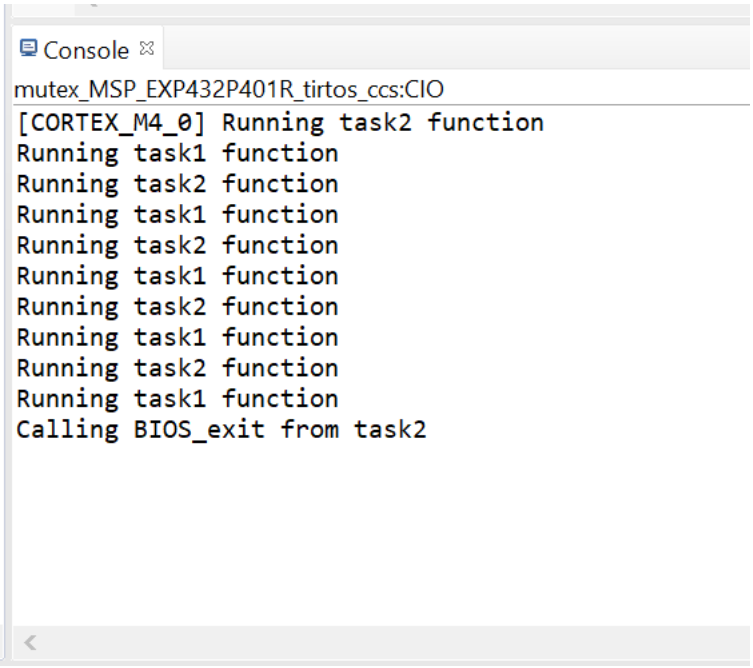
Rajbir Kahlon

Lab 7 Assignment

Exercise 1.1

The example mutex project was built and run without a problem.



Exercise 1.2

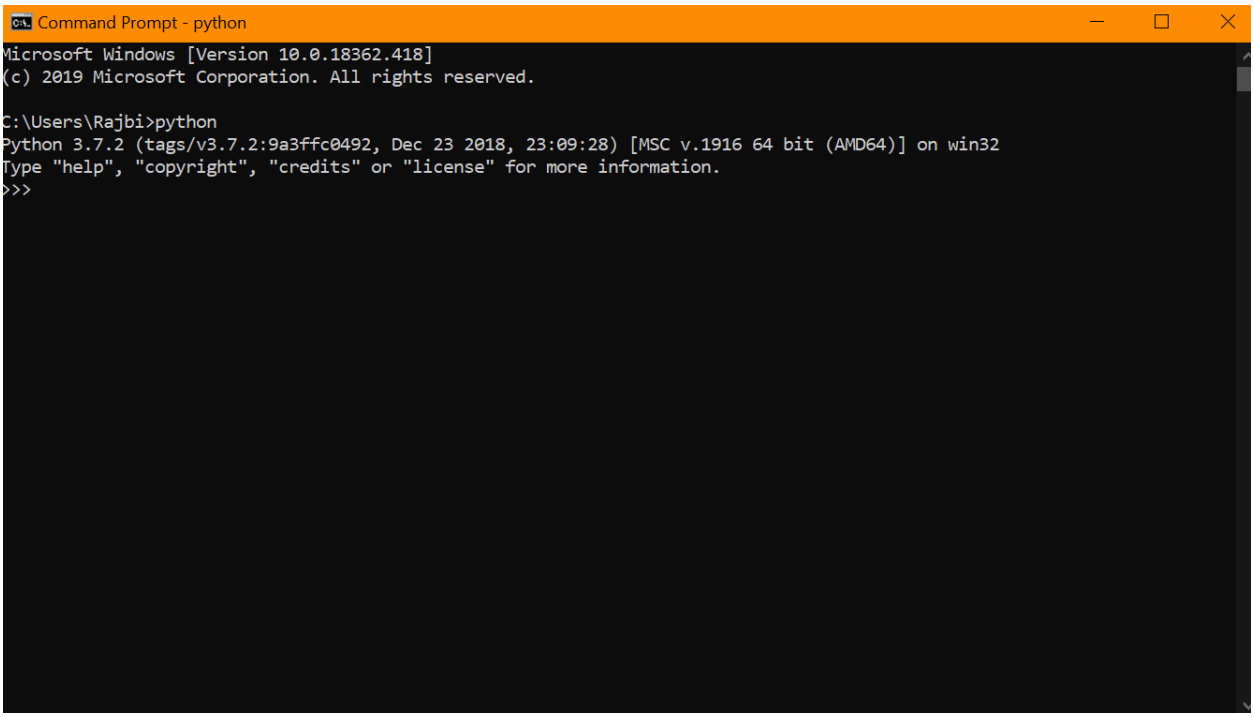
The tasks were re-written to blink the LEDs on the board. *#define __MSP432P4XX__* and *#include <ti/devices/msp432p4xx/driverlib/driverlib.h>* were added to use LED control as done in past labs. In the first task, the LED was initialized to high but in second task its LED it was set to low to avoid them being on at same time and then each toggled inside an infinite while loop. The Task_sleep(2000) was used to keep each on for 2 seconds. The source code can be found in the appendix.

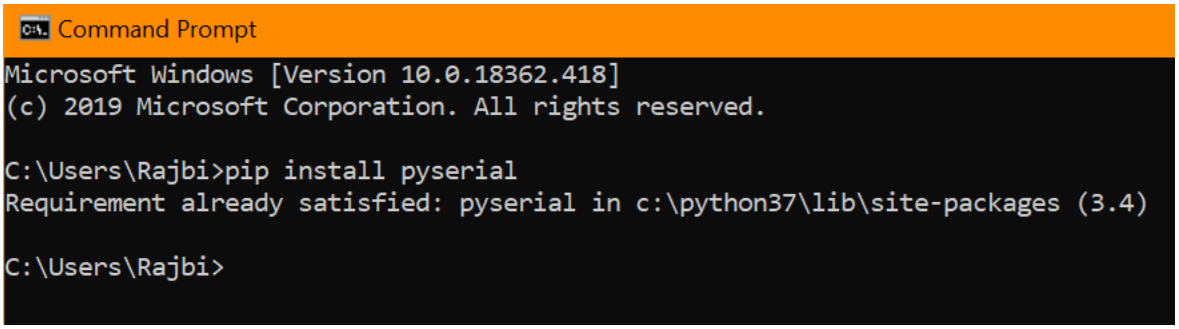
Exercise 1.3

For this portion the semaphores were used as shown in the example mutex program provided by TI. To prevent violating mutual exclusion, semaphore is implanted so that once a task is done, it can restore the semaphore value. Also, if the semaphore is 0, the task will wait until the resource is available again. The mutex example was written to only run 5 times so that part of the code was discarded. Similar to the 1.2, each Led is initialized either high or low to make sure only one is one at a time. So if a semaphore is available, the LED in a task will toggle and then increment the value of the semaphore, making it available for the other task to use. Semaphore_pend function reserves the semaphore to use for the current task and Semaphore_post function gave the resource up to be available for use again. The source code is listed in the appendix. The

TASKSTACKSIZE was incremented to 1024 to accommodate the printf statements inside the tasks as suggested.

Exercise 2





Not able to send to UART correctly

Appendix

Exercise 1.1

```
/*
 * Copyright (c) 2015-2019, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
```

```
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/*
 *  ======== 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 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();

    /* 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");

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

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

        /* Do work by waiting for 2 system ticks to pass */
        time = Clock_getTicks();
        while (Clock_getTicks() <= (time + 1)) {
```

```
            ;
        }

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

        Semaphore_post(semHandle);

        Task_sleep(sleepTickCount);
    }
}

/*
 *  ======== task2Fxn ========
 */
Void task2Fxn(UArg arg0, UArg arg1)
{
    for (;;) {
        System_printf("Running task2 function\n");

        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);

        Task_sleep(sleepTickCount);

        finishCount++;
        if (finishCount == 5) {
            System_printf("Calling BIOS_exit from task2\n");
            BIOS_exit(0);
        }
    }
}
```

Exercise 1.2

```
/*
 * Copyright (c) 2015-2019, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
```

```
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/*
 *  ======== 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>
#define __MSP432P4XX__
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

#include <ti/drivers/Board.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()
{
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1);

    /* Construct BIOS objects */
    Task_Params taskParams;
    Semaphore_Params semParams;

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

    /* 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)
{
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0); //prevent both from being
in same state

    while(true) {
        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
        Task_sleep(2000); //to give other task 2 seconds to execute
    }
}

/*
```

```
 *  ======== task2Fxn ========
 */
Void task2Fxn(UArg arg0, UArg arg1)
{
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1); //prevent both from being in
same state

    while(true) {
        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P2, GPIO_PIN1);
        Task_sleep(2000); //to give other task 2 seconds to execute
    }
}
```

Exercise 1.3

```
/*
 * Copyright (c) 2015-2019, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

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

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

/* BIOS module Headers */
```

```c
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>
#define __MSP432P4XX__
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

#include <ti/drivers/Board.h>

#define TASKSTACKSIZE   1024

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()
{
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1);
    /* Construct BIOS objects */
    Task_Params taskParams;
    Semaphore_Params semParams;

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

    /* 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");

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

        /* Get access to resource */
        Semaphore_pend(semHandle, BIOS_WAIT_FOREVER); //reserve resource to use for
this task

        /* Do work by waiting for 2 system ticks to pass
        time = Clock_getTicks();
        while (Clock_getTicks() <= (time + 1)) {
            ;
        } */

        /* Do work on locked resource */
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0); //prevent both from
being in same state
        while(true) {
            MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
            Task_sleep(2000); //to give other task 2 seconds to execute
            }
        resource += 1; //increase semaphore to let other task use it if needed
        /* Unlock resource */

        Semaphore_post(semHandle);

        Task_sleep(2000); //to give other task 2 seconds to execute
    }
}

/*
 *  ======== task2Fxn ========
 */
Void task2Fxn(UArg arg0, UArg arg1)
{
    for (;;) {
        System_printf("Running task2 function\n");
```

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

        /* Get access to resource */
        Semaphore_pend(semHandle, BIOS_WAIT_FOREVER); //reserve resource to use for
this task

        /* Do work on locked resource */
        Semaphore_post(semHandle);
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1); //prevent both from
being in same state
        while(true) {
            MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P2, GPIO_PIN1);
            Task_sleep(2000); //to give other task 2 seconds to execute
            }
        resource += 1; //increase semaphore to let other task use it if needed
        /* Unlock resource */

        Task_sleep(2000); //to give other task 2 seconds to execute

        /*finishCount++;
        if (finishCount == 5) {
            System_printf("Calling BIOS_exit from task2\n");
            BIOS_exit(0);
        } */
    }
}
```