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

Lab Assignment 7

Due date: Friday, 11/29/2019

1. Description

In this assignment, we will be familiarized with TI's real-time operating system, TI-RTOS. We will create tasks to do some work. We will use semaphore to do task synchronization. We will also use the TI driver to do communication with UART.

Details of the TI-RTOS can be found in the wiki page: http://processors.wiki.ti.com/index.php/TI-RTOS.
TI-RTOS User's Guide: http://www.ti.com/lit/ug/spruhd4m/spruhd4m.pdf.
TI-RTOS Kernel User's Guide: http://www.ti.com/lit/ug/spruex3u/spruex3u.pdf.

2. Exercise 1

We will do some multitasking with TI-RTOS to control two LEDs. We will use the local CCS. Again, it will be easier to duplicate an example project and modify it for our need. On CCS, go to Resource Explorer. Import the example project, <Software><SimpleLink MSP432P4 SDK – v:3.30.00.13><Examples><Development Tools><MSP432P401R LaunchPad – Red 2.x (Red)><TI-RTOS Kernel (SYS_BIOS)><mutex><TI-RTOS><CCS Compiler><mutex>.

Exercise 1.1

Build the project and run it. You should see the debug console showing something like this:

```
Running task1 function
Running task2 function
Running task1 function
Running task2 function
Running task1 function
Calling BIOS_exit from task2
```

Lab Report Submission

Show a screenshot of the debug console.

Exercise 1.2

We are going to modify the example project to create two tasks. One task blinks one LED and the other task blink another LED.

The following are excerpts from the example project's main file, *mutex.c*, to illustrate the steps to create a new task.

```
1   Void task1Fxn(UArg arg0, UArg arg1);
2
3   Task_Struct task1Struct;
4   Char task1Stack[TASKSTACKSIZE];
5
6   int main()
7   {
8       Task_Params taskParams;
9
10      Task_Params_init(&taskParams);
11      taskParams.stackSize = TASKSTACKSIZE;
12      taskParams.stack = &task1Stack;
13      taskParams.priority = 1;
14      Task_construct(&task1Struct, (Task_FuncPtr)task1Fxn, &taskParams, NULL);
15
16      BIOS_start();     /* Does not return */
17      return(0);
18  }
19
20  Void task1Fxn(UArg arg0, UArg arg1)
21  {
22      :
23      :
24  }
```

Line 3 declares a data structure for the task.
Line 4 defines a stack space to run the task. In order to use *printf()*, you may need to increase the stack size, to something like 1024 bytes.
Line 8 declares a parameter data structure for the task. Line 10 initializes the data structure to contain default values.
Lines 11-13 set up the parameters for the task we are going to create.
Line 14 creates the task in the operating system. The task's program instructions are contained in the *function task1Fxn()*.
Line 16, *BIOS_start()* jumps to start the operating system and it will not return to *main()*. The OS will schedule the tasks created and run them in a time-slice basis.

For this exercise, you can simply replace the entire contents of *task1Fxn()* and *task2Fxn()* with your own instructions to blink the LEDs. You can use codes from previous projects that use DriverLib to control the LEDs. To use DriverLib, include the following two lines at the beginning of the main file:

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

When you import codes from previous (non-RTOS) projects, make sure you don't include anything that is unrelated to the GPIO port that drives an LED, like stopping the watchdog timer or setting up new clock frequencies. Those things are taken care of by the OS. All we need to do is to focus on the tasks.

Have *task1Fxn()* blink one LED and *task2Fxn()* blink a different one. Use the system function *Task_sleep(time_ms)* to control the timing. For example, *Task_sleep(1000)* will allow the OS to suspend the task for 1000 ms. During those times, the OS will use them to run other tasks. You can use a forever *while* loop to blink the LED. Turn on the LED for 2 s, turn it off for 2 s, and repeat.

Lab Report Submission

Explain how you implemented the tasks. List the program.

Exercise 1.3

In this exercise, we are going to blink the LEDs in order. After we turn off one LED, we immediately turn on the other. We will use semaphore to synchronize the execution of the two tasks we created in the previous exercise. The following are excerpts from the example project's main file, mutex.c, to illustrate the steps to create a semaphore.

```
1  Semaphore_Struct semStruct1;
2  Semaphore_Handle semHandle1;
3  Semaphore_Params semParams;
4
5  Semaphore_Params_init(&semParams);   // Initialize structure with default parameters
6  Semaphore_construct(&semStruct1, 1, &semParams);   // Create an instance of semaphore object
7  semHandle1 = Semaphore_handle(&semStruct1);
```

Line 1 declares a data structure for the semaphore.
Line 2 declares a handle for the semaphore. We use the handle when we operate on the semaphore. For example, we use it to lock or unlock the semaphore.
Line 3 declares a parameter data structure for the semaphore.
Line 5 initializes the parameter data structure with default values.
Line 6 creates the semaphore with maximum count value of 1.
Line 7 gets the handle of the semaphore for future operations.

We can use the system function *Semaphore_pend(semHandle1, BIOS_WAIT_FOREVER)* to get the semaphore. If it is not available (count ≤ 0), the OS will suspend the task until the semaphore is available. We can use *Semaphore_post(semHandle1)* to increment the semaphore's count so that the waiting task can be made ready again.

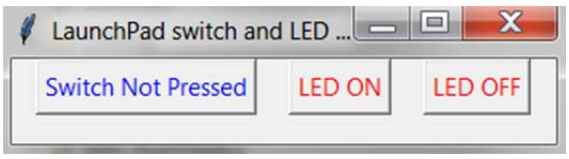
Similar to the previous exercise, you can use a forever *while* loop to blink the LED. Turn on the LED for 2 s, turn it off for 2 s, and repeat. Make sure they are not on or off at the same time.

Lab Report Submission

Explain how you made use of semaphore to synchronize the blinking of the two LEDs. List the program.

3. Exercise 2

In this exercise, we are going to use the UART device driver in TI-RTOS to communicate with a Python script running on a laptop. The Python script produces a small GUI (see below) that shows the status of a push button on LaunchPad. It also allows the user to turn on or off an LED. The Python script *led_sw_control.py* comes with this assignment. To run it: `python led_sw_control.py --port com_port`.



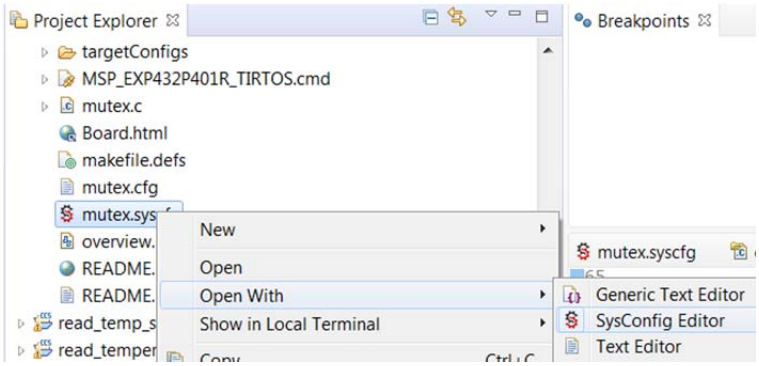
We will use the local CCS. On CCS, go to Resource Explorer. Import the example project, <Software><SimpleLink MSP432P4 SDK – v:3.30.00.13><Examples><Development Tools><MSP432P401R LaunchPad – Red 2.x (Red)><TI Drivers><uartecho><TI-RTOS><CCS Compiler><uartecho>. Build and run the project with your favorite terminal program to make sure the

communication between LauchPad and your laptop is working. Whatever you type on the terminal should be echoed back.

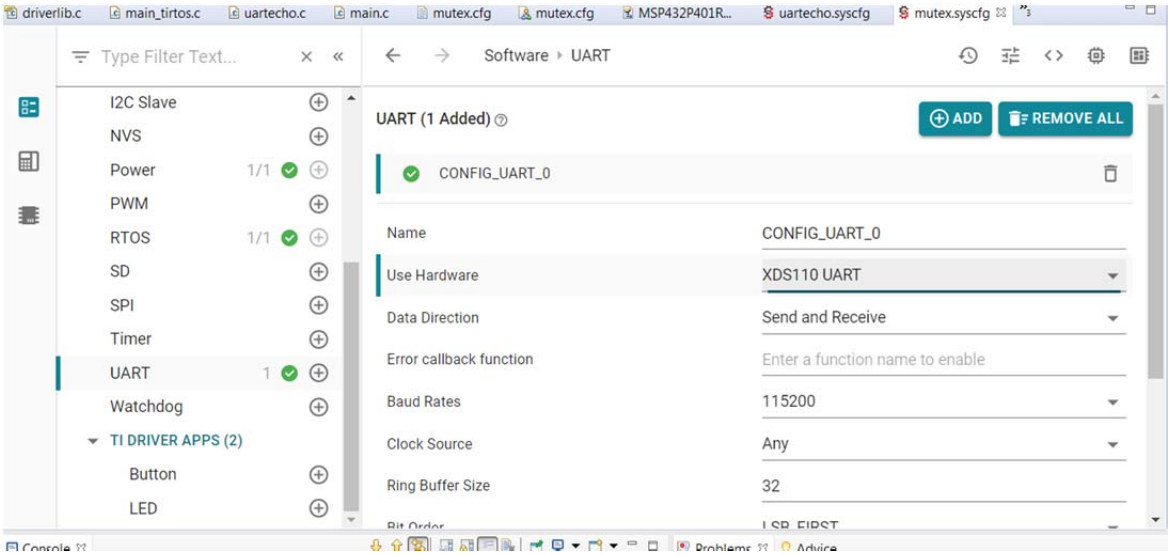
Study the *uartecho.c* file in the project. It contains a task that sets up the UART and does the echoing. Details of the UART APIs can be found in the TI-RTOS User's Guide, Section 5.12 UART Driver.

We will use the project we built in Exercise 1. You can duplicate the project or re-import the *mutex* example project. There seems to be no direct way on the local CCS to duplicate a project. One way is do it is to export an existing project to a zip file in the local hard disk, rename the project on CCS, and re-import the exported project. Another way is to rename the existing project on CCS and re-import the example project from Resource Explorer.

We need to add the UART driver to the project. On the project pane, locate the *mutex.syscfg* file. Right-click on it and open it with SysConfig Editor.



Scroll to the UART section on the left pane. Click on the circled + symbol. Set "Use Hardware" to XDS110 UART.



In the main file, include the following two lines:

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

Exercise 2.1

We are going to replace the two tasks with new instructions. In this exercise, we will build the task to control the LED. So, let's modify *task1Fxn()* first. Comment out the line to create *task2* for the time being. Model the same way it is done in *uartecho.c* but do not echo the character received. Also add codes to set up a GPIO pin to drive an LED. When the user presses the "LED ON" button, the Python script will send an ASCII '1' to LaunchPad; it will send a '0' when the "LED OFF" button is pressed. Therefore, modify *task1Fxn()* so that it turn on or off the LED accordingly.

Lab Report Submission

Explain how you implemented the task. List the program.

Exercise 2.2

In this exercise, we are going to show a push button's status on the Python GUI. We are going to implement the functionality in *task2Fxn()* to accomplish that. Here are the requirements:

1. The task shall read the switch status once every 100 ms. If the switch is not pressed, it will send an ASCII string "OPEN\n" to the UART; if the switch is pressed, it will send "CLOSE\n" instead.
2. In order to avoid sending the same message again and again (wasting communication bandwidth), if the new status is same as the old one, do not send the message.
3. Make an exception to #2 above. It must send the status message if nothing has been sent in 2 seconds. This is to make sure that the board is still alive and the communication channel is still okay.

Lab Report Submission

Explain your implementation. List your program.

4. Grading

	Points
Exercise 1.1	1
Exercise 1.2	1
Exercise 1.3	3
Exercise 2.1	1
Exercise 2.2	3
Report Contents	1
Total	10

Grading Policy:

The lab's grade is primarily determined by the report. We will grade the report first. Based on what we see in the demo, we may adjust the grade by either deducting or adding points.

Demo is mandatory. If you don't show up for demo, half of the points will be deducted.

If you miss the submission deadline for the report, you will get zero point. However, you can still come to

the demo to show what you have done and hopefully learn something as well. And you may get a maximum of 30% of the full grade for the assignment. I will drop the lowest score of the lab assignments when I determine the final grade.

As for the report contents, do not use screenshots to show your codes. In the report body, list the relevant codes only for the exercise you are discussing. Put the entire program listing in appendix. Put the contents of one exercise in one section. Do not separate them to two different sections. This will make the grading easier.