About RTOS Lab 2
[edit]

Play Video

Objectives[edit]

- Develop debugging skills using the Keil debugger and TExaS logic analyzer
- Understand the two existing round robin preemptive schedulers
- Appreciate the distinction between real-time and non-real time tasks
- Develop a scheduler that runs two periodic event threads and four main threads
- Implement spin-lock semaphores and a mailbox

Overview[edit]

We want you to understand how an RTOS works and demonstrate your understanding by completing a set of activities. The Lab 2 starter project using the LaunchPad and the Educational BoosterPack MKII (BOOSTXL-EDUMKII) is again a fitness device. However, the starter project will not execute until you implement a very simple RTOS. The user code inputs from the microphone, accelerometer, temperature sensor and switches. It performs some simple measurements and calculations of steps, sound intensity, and temperature. It outputs data to the LCD and it generates simple beeping sounds. Figure Lab2.1 shows the data flow graph of Lab 2. Your assignment is to first understand the concepts of the chapter in general and the projects **RTOS_xxx** and **RoundRobin_xxx** in specific. Your RTOS will run two periodic threads and four main threads. Sections 2.3.1 – 2.3.6 develops an RTOS that runs three main threads and your system must run four main threads. Section 2.3.7 explains how to extend the Scheduler() function so that it also runs periodic tasks. Section 2.4.2 explains how to implement spinlock semaphores.

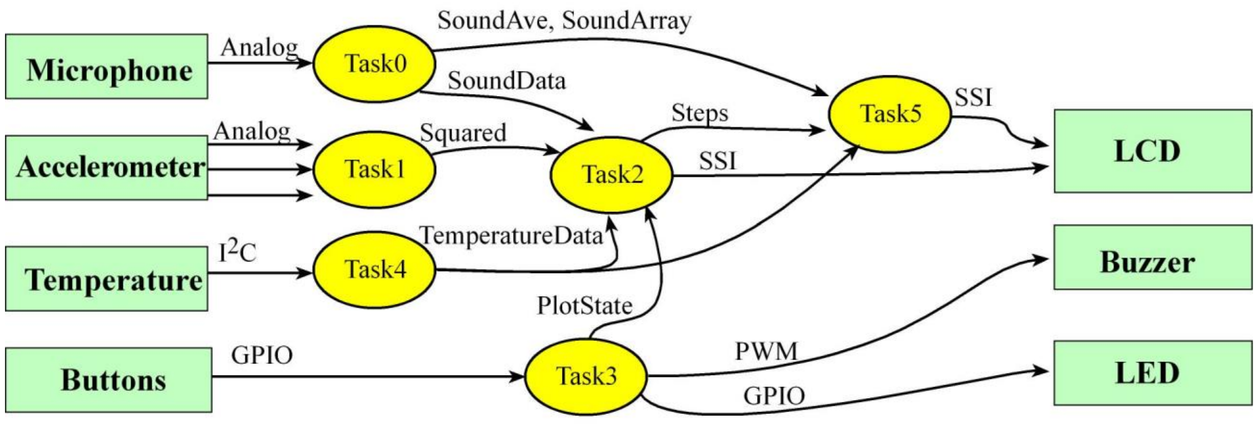


Figure 2.1. Data flow graph of Lab 2.

This simple fitness device has six tasks: two periodic and four main threads. Since we have two periodic threads to schedule, we could have used interrupts on two hardware timers to run the real-time periodic threads. However, Lab 2 will run with just SysTick interrupts to run two the periodic threads and to switch between the four main threads. These are the six tasks:

- Task0: event thread samples microphone input at 1000 Hz
- Task1: event thread samples acceleration input at 10 Hz
- Task2: main thread detecting steps and plotting at on LCD, runs about 10 Hz
- Task3: main thread inputs from switches, outputs to buzzer
- Task4: main thread measures temperature, runs about 1 Hz
- Task5: main thread output numerical data to LCD, runs about 1 Hz

Your RTOS manages these six tasks. We will use the same metrics as described as used in Lab 1, except jitter and error are only relevant for the two real-time event tasks:

- Min_j = minimum ΔT_j for Task j, j=0 to 5
- Max_j = maximum ΔT_j for Task j, j=0 to 5
- $\text{Jitter}_j = \text{Max}_j - \text{Min}_j$ for Task j, j=0 to 1
- Ave_j = Average ΔT_j for Task j, j=0 to 5
- $\text{Err}_j = 100*(\text{Ave}_j - \Delta t_j)/\Delta t_j$ for Task j, j=0 to 1

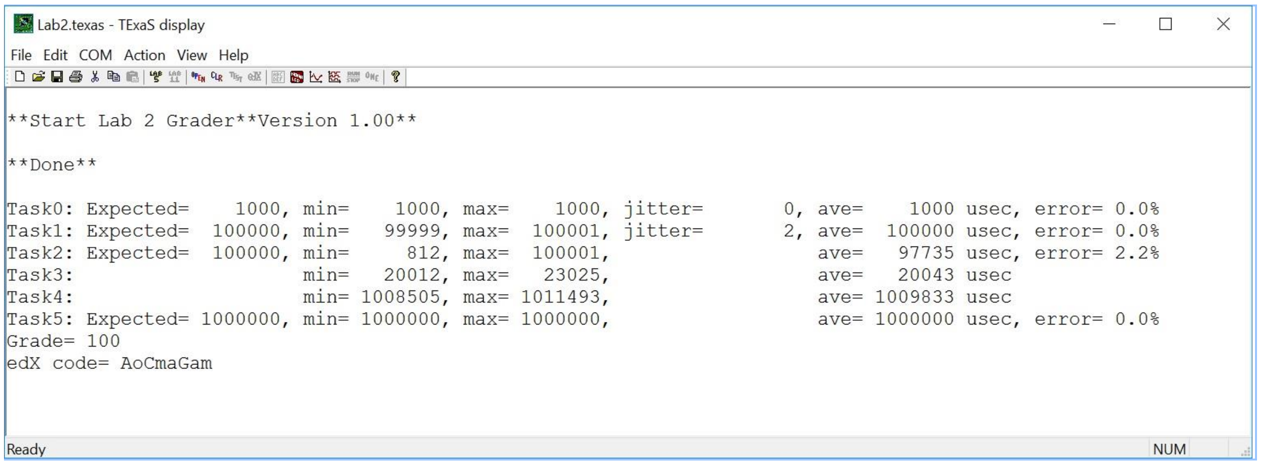
In addition to the above quantitative measures, you will be able to visualize the execution profile of the system using a logic analyzer. Each task in Lab 2 toggles both the virtual logic analyzer and a real logic analyzer when it starts. For example, Task0 calls **TExaS_Task0()**. The first parameter to the function **TExaS_Init()** will be **GRADER** or **LOGICANALYZER**. Just like Lab 1, calling **TExaS_Task0()** in grader mode performs the lab grading. However in logic analyzer mode, these calls implement the virtual logic analyzer and can be viewed with **TExaSdisplay**. The TExaS logic analyzer should be used during debugging.

Debugging Lab 2[edit]

Specifications[edit]

A real-time system is one that guarantees the jitters are less than a desired threshold, and the averages are close to desired values. Now that we are using interrupts we expect the jitter for the two event tasks to be quite low. For the four main threads, you will be graded only on minimum, maximum, and average time between execution of tasks. Your assignment is implement the OS functions in **OS.c** and write the SysTick interrupt service routine in **osasm.s**. We do not expect you to edit the user code in **Lab2.c**, the board support package in **BSP.c**, or the interface specifications in profile.h, Texas.h, BSP.h, or OS.h. More specifically, we are asking you to develop and debug a real-time operating system, such that

- Task0: jitter between executions should be less than or equal to 15us
- Task1: jitter between executions should be less than or equal to 30us
- Task2: average time between executions should be 100 ms within 5%
- Task3: average time between executions should be less than 50 ms
- Task4: average time between executions should be less than 1.2 s
- Task5: average time between executions should be 1.0 s within 5%



```
**Start Lab 2 Grader**Version 1.00**

**Done**

Task0: Expected=    1000, min=    1000, max=    1000, jitter=       0, ave=    1000 usec, error= 0.0%
Task1: Expected=  100000, min=   99999, max=  100001, jitter=       2, ave=  100000 usec, error= 0.0%
Task2: Expected=  100000, min=     812, max=  100001,                   ave=   97735 usec, error= 2.2%
Task3:                    min=   20012, max=   23025,                   ave=   20043 usec
Task4:                    min= 1008505, max= 1011493,                   ave= 1009833 usec
Task5: Expected= 1000000, min= 1000000, max= 1000000,                   ave= 1000000 usec, error= 0.0%
Grade= 100
edX code= AoCmaGam
```

Figure 2.2. TExaS window showing a solution to Lab 2.

Approach[edit]

Before you begin editing, downloading and debugging, we encourage you to first open up and run a couple of projects. The first project we recommend is **RTOS_xxx**. This project implements a very simple real-time operating system as described in Sections 2.3.1 – 2.3.5. Make sure you understand function pointers and each line of the SysTick ISR.

Play Video

Next, we encourage you should open up the project **RoundRobin_xxx**. This project extends the simple real time operating system so the scheduler is implemented in C as described in Section 2.3.6. Make sure you understand how the assembly code calls a C function. You will need to understand how the assembly code accesses the shared global, **RunPt**. Remember this approach will only work if the time to execute **Scheduler()** is very short compared to the time between SysTick interrupt triggers.

Running round robin

Play Video

Running Lab 2 with the TExaS Logic Analyzer

Play Video

Running Lab2 with a real logic analyzer

Play Video

Third, we encourage you should open up the project **Lab2_xxx** and fully understand the system from the user perspective by reading through **Lab2.c**. Lab2 requires both the LaunchPad and the Educational BoosterPack MKII. Next, read through **OS.c** and **OS.h** to learn how your operating system will support the user system. Since this is a class on operating systems, and not personal fitness devices, we do not envision you modifying **Lab2.c** at all. Rather you are asked to implement the RTOS by writing code in the **osasm.s** and **OS.c** files.

To activate the logic analyzer, initialize TExaS with **TExaS_Init(LOGICANALYZER,1000);** Do not worry about the number 1000; you will fill in a valid number once you are done with Lab 2.

To activate the grader, initialize TExaS with **TExaS_Init(GRADER,1000);** When you run the starter code in grading mode, you should see this output on TExaSdisplay. Note the numbers on the MSP432 running at 48 MHz will be slightly different than the numbers generated by the TM4C123 running at 80 MHz.

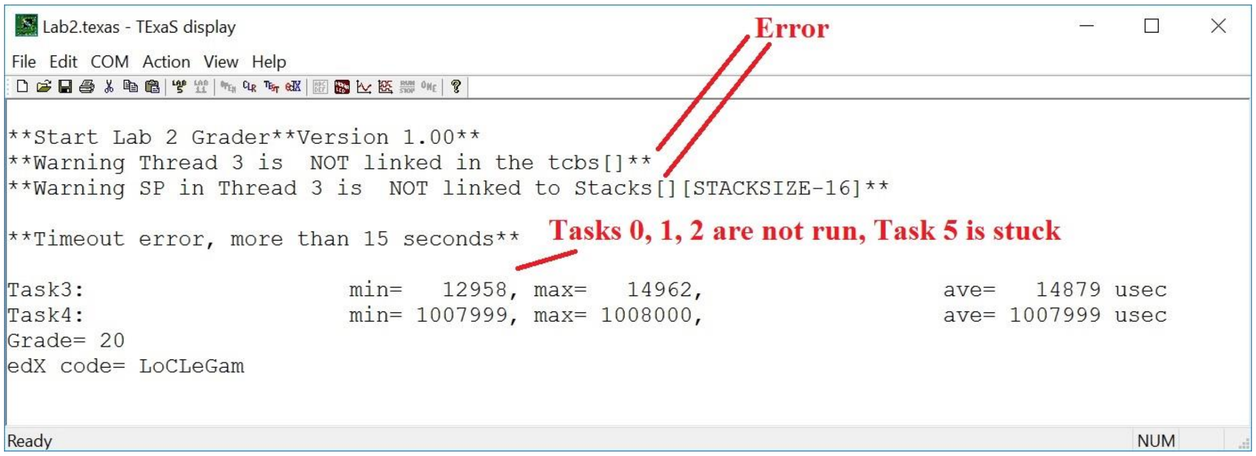
Step 1) Implement the three spin lock semaphore functions as defined in **OS.c** and **OS.h**. For more information on semaphores review Section 2.4. Create a simple main program to test the functions.

```
int32_t s1,s2;
int main(void){
  OS_InitSemaphore(&s1,0);
  OS_InitSemaphore(&s2,1);
  while(1){
    OS_Wait(&s2);   //now s1=0, s2=0
    OS_Signal(&s1); //now s1=1, s2=0
    OS_Signal(&s2); //now s1=1, s2=1
    OS_Signal(&s1); //now s1=2, s2=1
    OS_Wait(&s1);   //now s1=1, s2=1
    OS_Wait(&s1);   //now s1=0, s2=1
  }
}
```

Step 2) Implement the three mailbox functions as defined in **OS.c** and **OS.h**. Task1 is an event thread that calls **OS_MailBox_Send**. Therefore, your implementation of send cannot spin. In other words if Task1 sends data to the mailbox and the mailbox is already full that data will be lost. Create a simple main program to test the functions.

```
uint32_t Out;
int main(void){ uint32_t in=0;
  OS_MailBox_Init();
  while(1){
    OS_MailBox_Send(in);
    Out = OS_MailBox_Recv();
    in++;
  }
}
```

Step 3) The minimal set of functions you need to write to get the system running is
 SysTick_Handler (without calling the C function and without running periodic threads)
 StartOS
 OS_Init
 OS_AddThreads (with just 3 threads for now)
 OS_Launch
Use this minimum OS to run Task3 Task4 and Task5. In other words, for now we will not run Task0, Task1, and Task 2. To get it to compile you will have to change the prototype of **OS_AddThreads** to match the implementation in **OS.c** and the call in main(). In other words, this minimal OS runs three main threads and your Lab 2 will eventually run four main threads. Task5 will stall in **OS_Wait** because it has no data. However Task3 should respond to button pushes and Task4 should measure temperature. You should hear the buzzer when you press a switch. You should be able to see these three tasks running on the TExaS logic analyzer, and you should be able to see global variables **PlotState** change with buttons, and you should notice that **TemperatureData** is set by Task4.

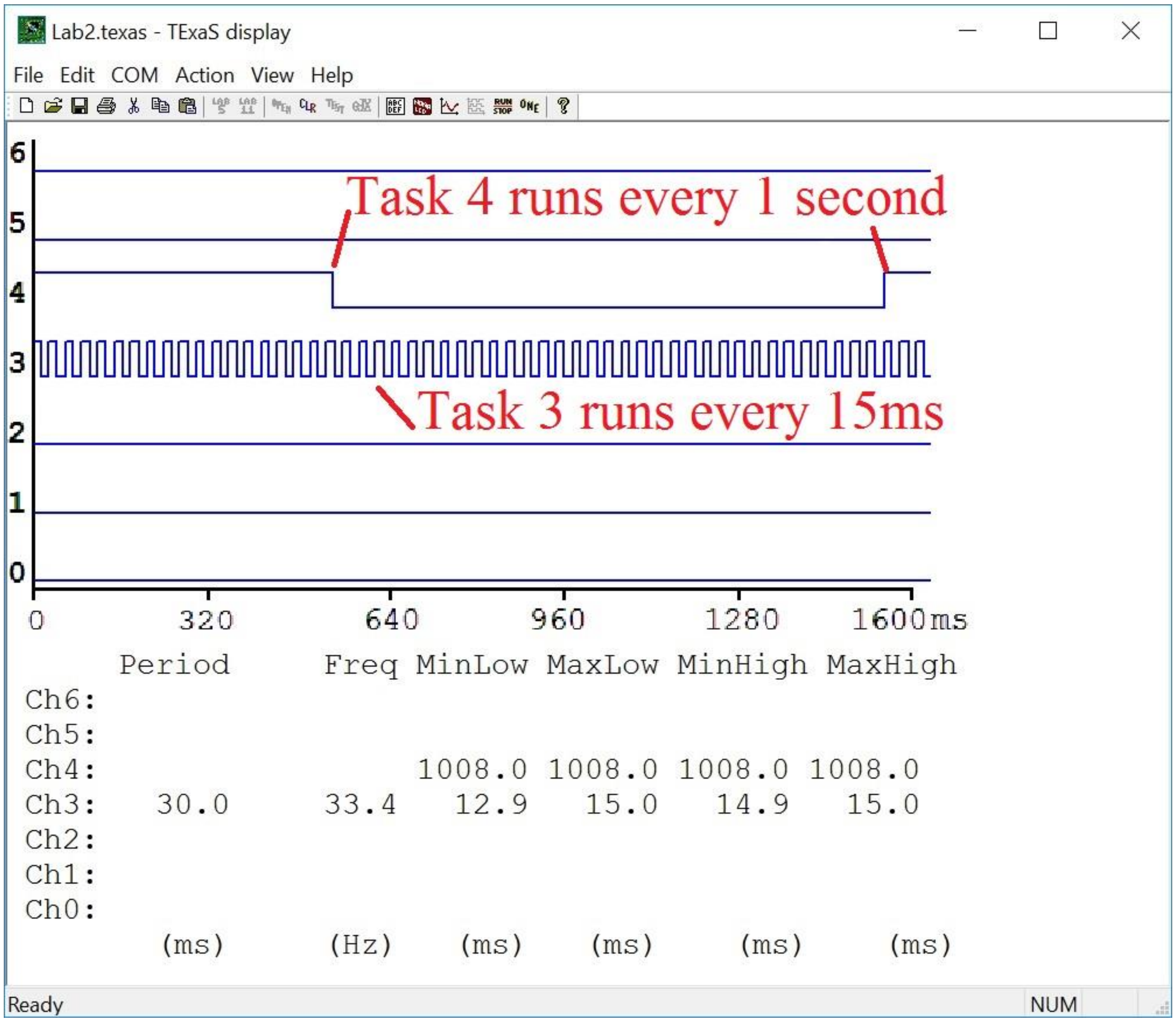

Lab2.texas - TExaS display
File Edit COM Action View Help
Error
Tasks 0, 1, 2 are not run, Task 5 is stuck
Ready
NUM

```
**Start Lab 2 Grader**Version 1.00**
**Warning Thread 3 is  NOT linked in the tcbs[]**
**Warning SP in Thread 3 is  NOT linked to Stacks[][STACKSIZE-16]**

**Timeout error, more than 15 seconds**

Task3:                    min=   12958, max=   14962,                 ave=   14879 usec
Task4:                    min= 1007999, max= 1008000,                 ave= 1007999 usec
Grade= 20
edX code= LoCLeGam
```

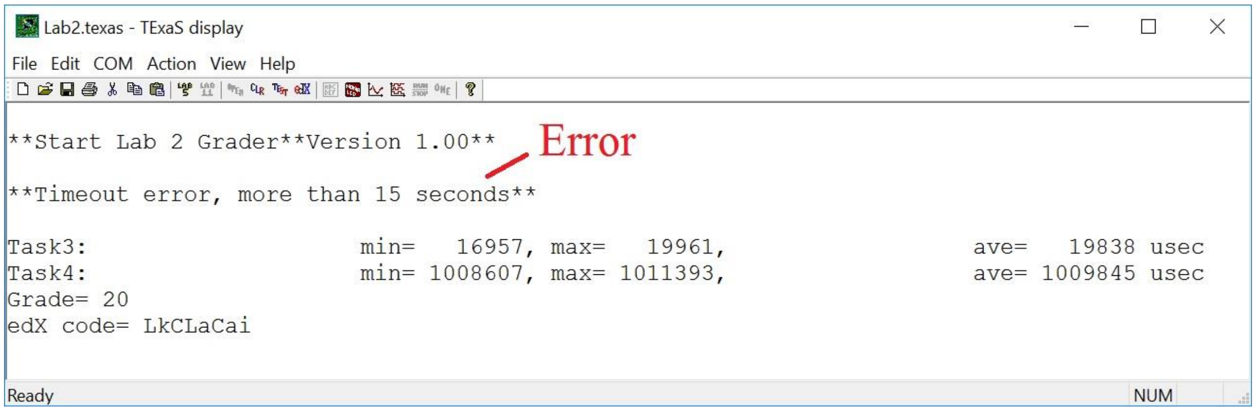
Figure 2.3. TExaS window showing a step 3 output. Notice only Tasks 3 and 4 will run.


Lab2.texas - TExaS display
File Edit COM Action View Help
6
5
4
3
2
1
0
Task 4 runs every 1 second
Task 3 runs every 15ms
0
320
640
960
1280
1600ms
Ready
NUM

```
      Period      Freq MinLow MaxLow MinHigh MaxHigh
Ch6:
Ch5:
Ch4:                   1008.0 1008.0 1008.0 1008.0
Ch3:   30.0       33.4   12.9   15.0   14.9   15.0
Ch2:
Ch1:
Ch0:
       (ms)       (Hz)   (ms)   (ms)   (ms)   (ms)
```

Figure 2.4. TExaS logic analyzer trace for a step 3 output. Notice only Tasks 3 and 4 will run. Task5 is running but it is stuck in OS_Wait.

Step 4) Modify the system so the OS takes and runs four main threads and use it to run Task2 Task3 Task4 and Task5. Modify the SysTick ISR so it calls a C function and implement the round robin scheduler in C. You will add periodic threads in the next step. The behavior of Task3 Task4 and Task3 will be similar to step 3. In addition, Task2 will stall in the **OS_Wait** inside of **OS_MailBox_Recv**. The logic analyzer trace for step 4 will be similar to Figure 2.4, except Task 3 runs a little slower because the scheduler is running tasks 2,3,4,5.



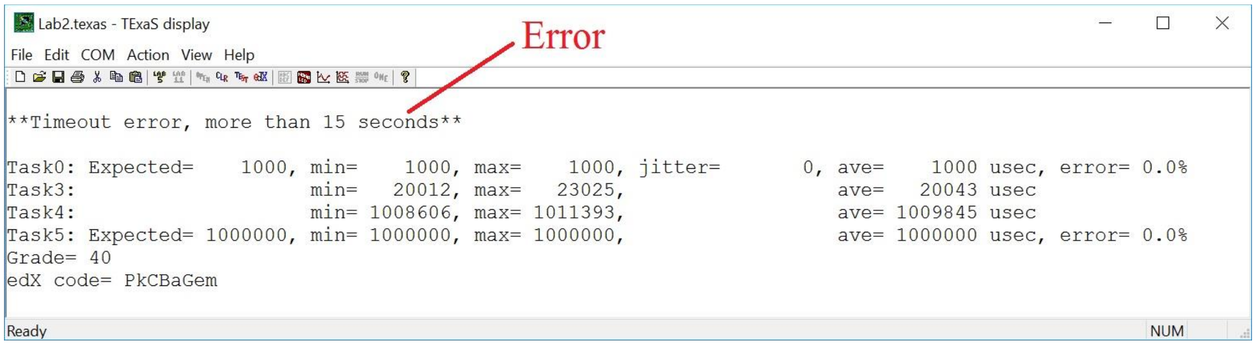
```
**Start Lab 2 Grader**Version 1.00**

**Timeout error, more than 15 seconds**

Task3:                    min=   16957, max=   19961,                ave=   19838 usec
Task4:                    min= 1008607, max= 1011393,                ave= 1009845 usec
Grade= 20
edX code= LkCLaCai
```

Figure 2.5. TExaS window showing a step 4 output. Tasks 2 and 5 are running but they are stuck in OS_Wait.

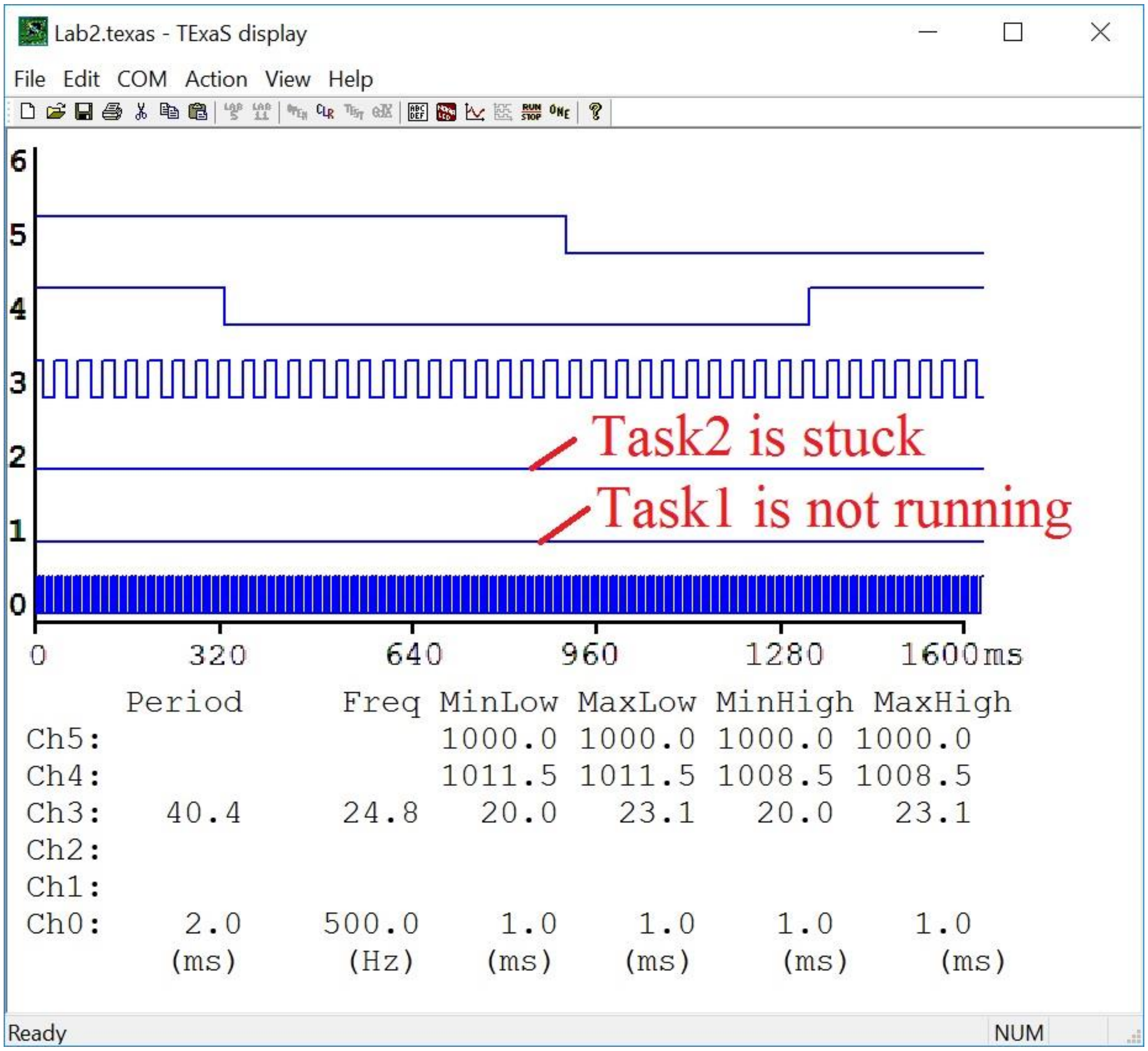
Step 5) Modify the system to execute one of the periodic tasks. If you run Task0, then Task5 will now run. If you run Task1, then Task2 will run.



```
**Timeout error, more than 15 seconds**

Task0: Expected=    1000, min=    1000, max=    1000, jitter=      0, ave=    1000 usec, error= 0.0%
Task3:                    min=   20012, max=   23025,                 ave=   20043 usec
Task4:                    min= 1008606, max= 1011393,                 ave= 1009845 usec
Task5: Expected= 1000000, min= 1000000, max= 1000000,                 ave= 1000000 usec, error= 0.0%
Grade= 40
edX code= PkCBaGem
```

Figure 2.6. TExaS window showing a step 5 output. Task 2 is running but it is stuck in OS_Wait.



```
      Period      Freq  MinLow  MaxLow  MinHigh MaxHigh
Ch5:                    1000.0  1000.0  1000.0  1000.0
Ch4:                    1011.5  1011.5  1008.5  1008.5
Ch3:    40.4      24.8    20.0    23.1    20.0    23.1
Ch2:
Ch1:
Ch0:     2.0     500.0     1.0     1.0     1.0     1.0
        (ms)      (Hz)    (ms)    (ms)    (ms)    (ms)
```

Figure 2.7. TExaS logic analyzer trace for a step 5 output. Only one periodic task is running (just Task 0 and not Task1)

Hint: If I wished to run **Myfunction()** every 49ms, I could do this

```
uint32_t TheTime=0;
void Scheduler(void){ // every 1ms time slice
  TheTime++;
  if(TheTime == 49){
    MyFunction(); // every 49ms
    TheTime = 0;
  }
  RunPt = RunPt->next; // Round Robin
}
```

Step 6) Modify the system to execute both periodic tasks.

Lab 2 Grader[edit]

Grading your lab solution does require a LaunchPad development board. Your assignment is to implement a very simple RTOS that runs two event threads and four main threads. In particular, we are asking you to modify the main program, such that

- Task0 runs exactly every 1ms (jitter less than 15us)
- Task1 runs exactly every 100ms (jitter less than 30us)
- Task2 runs every 100ms (average within 5%)
- Task3 runs at least every 50ms (average less than 50ms)
- Task4 runs approximately every 1s (average less than 1.2s)
- Task5 runs every 1.0s (average within 5%)

We do expect you to execute Tasks 0 and 1 during the execution of the SysTick ISR (inside of Scheduler), so we expect the jitter on these two tasks to be small.

Step 1 Compile (build) your project in Keil, and download the code.

Step 2 Start **TExaSdisplay** and open the COM port.

Step 3 Start your software with the debugger or by hitting the reset on the LaunchPad. It takes about 10 seconds to collect the task profile data the grader needs. The logic analyzer is not available during grading. Wait until grading is finished. Any score above 70 will be considered a passing grade. If you are not satisfied with your score you are allowed multiple submissions.