Lab RTOS 1

Before you can start you need to complete installing Keil IDE version 5 and TeXaS graders for RTOS labs. Installation instructions are here - go directly to "*Only for RTOS labs*" part.
[edit]

About Lab RTOS 1[edit]

Lab preparation[edit]

You will need a TM4C123 LaunchPad and an MK-II BoosterPack.

*****Reminder*** If you are using the TM4C123/LM4F120 you will need to remove R9 and R10 to use the TM4C123/LM4F120 with the MK-II booster.**
You might ask "what is the purpose of R9 and R10?" The MSP430 is a 16-bit low power microcontroller. It has a 20-pin LaunchPad header. A large number of BoosterPacks were made for the MSP430. When the LM4F120/TM4C123 LaunchPad came out, they added R9 R10 so some of these old MSP430 BoosterPacks could be used for the new LaunchPad.

MK-II BoosterPack Introduction[edit]

Play Video

The purpose of using a BoosterPack in this course is to provide a rich set of input/output while at the same time allowing students to focus on the writing of the operating system. This way all students have the same hardware configuration as each other, and more importantly the automated lab graders are programed to understand this hardware. Figure 1.31 shows the lab kit, which comprises of a LaunchPad and the MKII BoosterPack. Later in Chapter 6 we will add a second BoosterPack to provide Bluetooth communication.

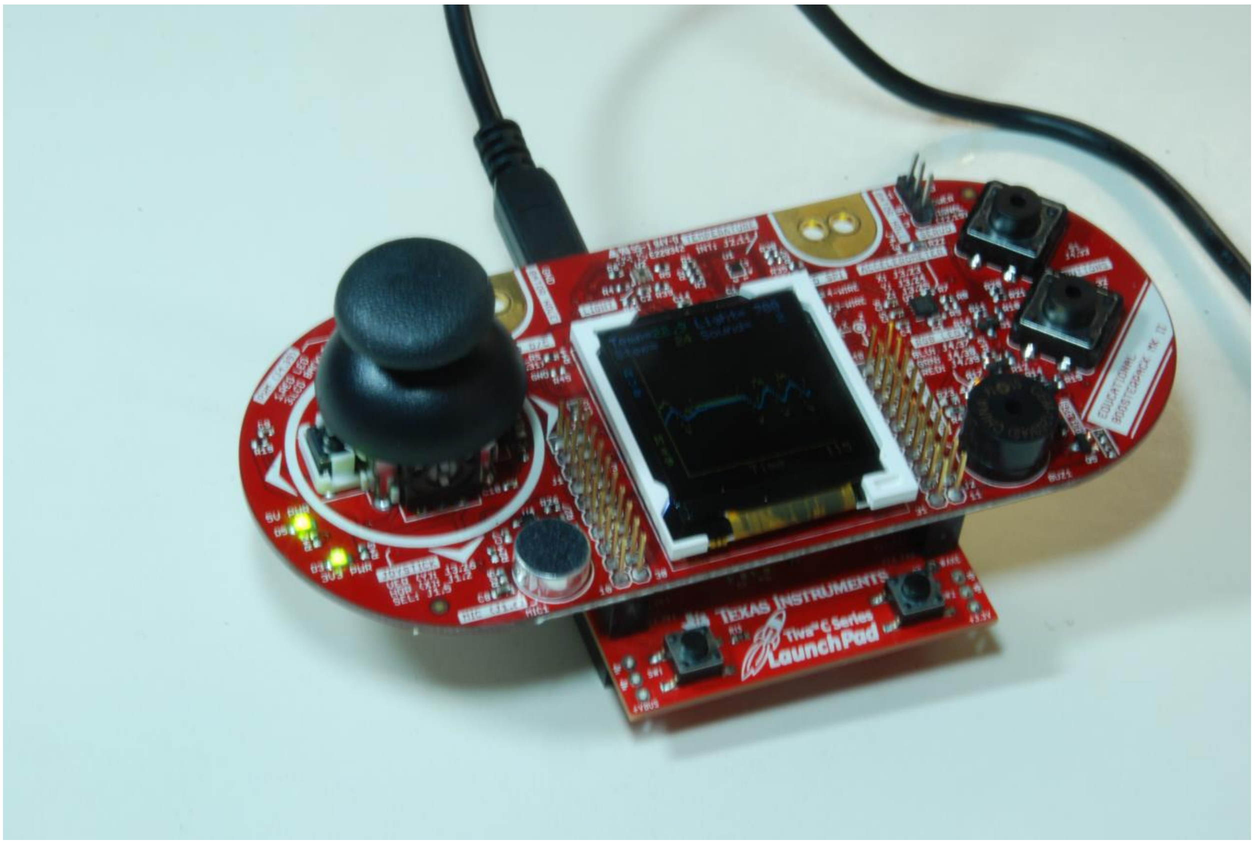


Figure 1.31. The lab hardware includes a LaunchPad, either a TM4C123 (EK-TM4C123GXL) or an MSP432 (MSP-EXP432P401R), together with an Educational BoosterPack MKII (BOOSTXL-EDUMKII).

The MKII provides a number of sensors that necessitate real-time processing: microphone for sound, temperature, 3-axis acceleration, and a light sensor. Furthermore, it has some I/O devices to interact with a human. For input it has buttons and a joystick, and for output it has LEDs, a buzzer and a color LCD display.

Play Video

This course deals with creating a real-time operating system for embedded systems. One of the important resources the OS must manage is I/O. It is good design practice to provide an abstraction for the I/O layer. Three equivalent names for this abstraction are **hardware abstraction layer** (HAL), device driver, and **board support package** (BSP). From an operating system perspective, the goal is the make it easier to port the system from one hardware platform to another. The system becomes more portable if we create a BSP for our hardware devices. We provide a BSP for the MKII BoosterPack that encapsulates the following:

-
    -
        - Button input

-

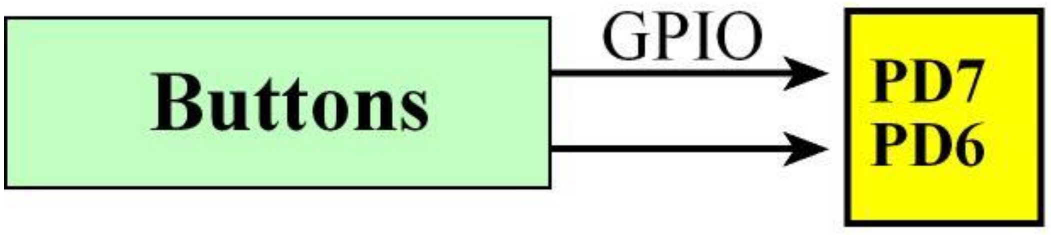
- ○
  - Joystick input
  - LED output
  - Buzzer output
  - Acceleration input
  - Microphone input
  - LCD graphics output
  - Light sensor input
  - Temperature sensor input
  - The processor clock

One of the advantages of this BSP is the operating systems we create together in class will actually run on either the MSP432 or the TM4C123. This class provides just enough information to understand what the I/O does, so that you can focus on the operating system.

If you want to understand how the I/O devices work, you should print its circuit diagram on pages 17 and 18 of the MKII Users Guide (MK-II_CircuitDiagram.pdf). You can also review the MK-II_usersGuide.pdf.

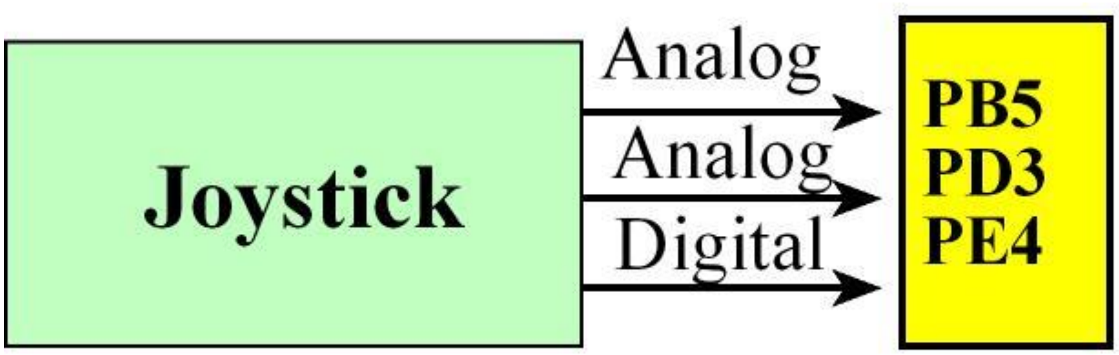
Buttons

[edit]



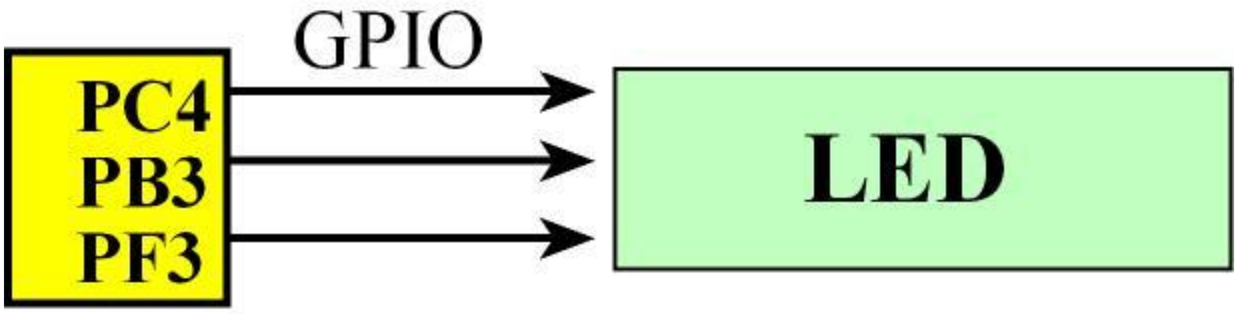
The initialization functions configure the I/O port for the two buttons (MJTP1212A_button.pdf). The input functions return the current status of the buttons. For information on how to use the functions, see the BSP.h file and look for functions that begin with **BSP_Button**. For information on how the interface operates, see the BSP.c file and the data sheet for your microcontroller.

Joystick[edit]



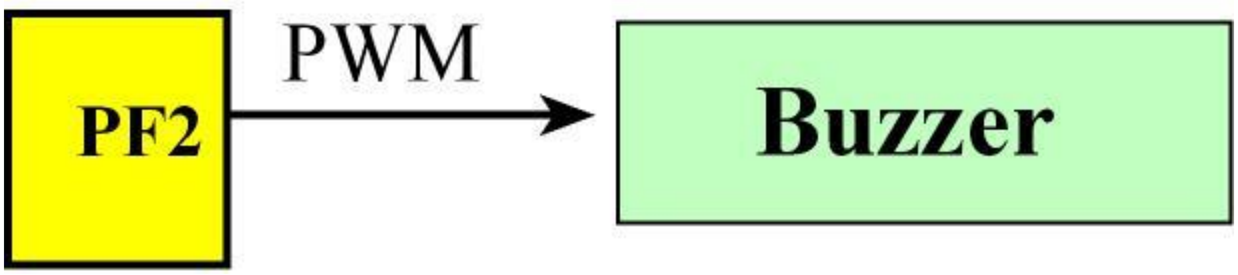
The joystick has two potentiometers and a momentary switch (IM130330001Joystick.pdf). One analog input is a function of the X-position of the joystick and another analog input is a function of the Y-position. The microcontroller uses its analog to digital converter (ADC) to measure the joystick position. The initialization functions configure the I/O ports for the joystick. The input functions return the current status of the joystick. For information on how to use the functions, see the BSP.h file and look for functions that begin with BSP_Joystick. For information on how the interface operates, see the BSP.c file and the data sheet for the joystick and for your microcontroller.

LEDs[edit]



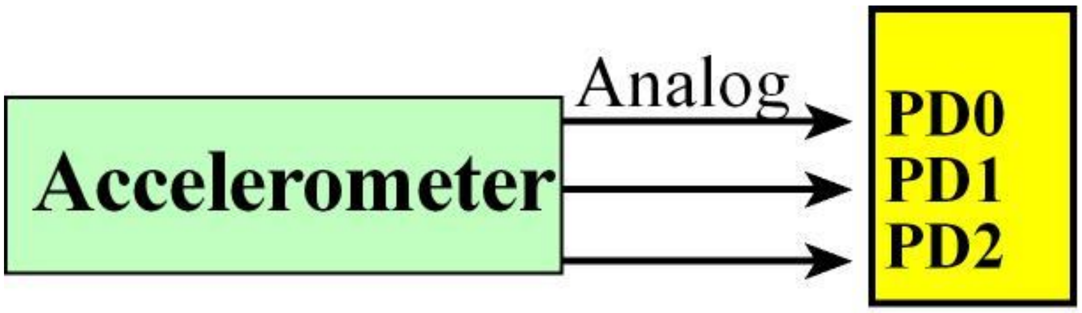
The MK-II has a 3-color LED (CLV1AFKB_LED.pdf). The initialization functions configure the I/O port for the LED. The output function sets the color of the LED. For information on how to use the functions, see the BSP.h file and look for functions that begin with BSP_RGB. For information on how the interface operates, see the BSP.c file and the data sheet for the LED and for your microcontroller.

Buzzer output[edit]



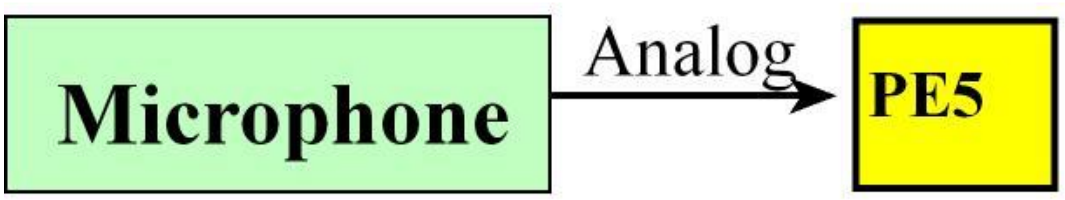
There is a buzzer on the MK-II. For more information, see the CEM1203 data sheet (cem1203buzzer.pdf). The digital control can set the loudness, but not the pitch. Outputting zero turns it off, and outputting one sets it at max loudness. Pulse width modulation (PWM) is a mechanism to adjust power to a device. In this interface the digital output is a wave with a fixed frequency of 2048 Hz (488us), but the software can set the duty cycle. For example, if the digital signal is high for 122us and low for 366us, the buzzer will be at 25% loudness. Duty cycle is defined as the time the signal is high divided by the total period of the wave. The initialization functions configure the I/O port for the buzzer. The output function sets the duty cycle of the PWM output. For information on how to use the functions, see the BSP.h file and look for functions that begin with BSP_Buzzer. For information on how the interface operates, see the BSP.c file and the data sheet for the buzzer and for your microcontroller.

Accelerometer for motion[edit]



The MK-II has a 3-axis accelerometer. For more information, see the KXTC9-2050 data sheet (KXTC9-2050Accelerometer.pdf). The X,Y,Z parameters are provided by three analog signals. The microcontroller uses its analog to digital converter (ADC) to measure acceleration. The initialization functions configure the I/O ports for the accelerometer. The input function performs an ADC conversion and returns the X,Y,Z acceleration data. For information on how to use the functions, see the BSP.h file and look for functions that begin with **BSP_Accelerometer**. For information on how the interface operates, see the BSP.c file and the data sheet for your microcontroller.

ADC Microphone for sound[edit]



The MK-II has a microphone for measuring sound. For more information, see the MK-II circuit diagram (MK-II_CircuitDiagram.pdf) and the microphone data sheet (cma-4544pf_microphone.pdf). The microcontroller uses its analog to digital converter (ADC) to measure sound. The initialization functions configure the I/O ports for the ADC. The input function performs an ADC conversion and returns the sound amplitude. Normally, we sample sound at 10 to 44 kHz and process the data to detect particular sounds. In this class, we will collect multiple sound samples at at a fast rate and use this buffer of sound data to measure the overall amplitude of the sound. Let *x(i)* be the measured sound data for *i* = 0 to *n*-1, where n=1000.

$Ave = (x(0)+x(1)+x(2)+\ldots x(n-1))/n$

$Rms = \mathrm{sqrt}(\ ((x(0)-Ave)^2+(x(1)-Ave)^2+\ldots(x(n-1)-Ave)^2)/n\ )$

Fitting into the theme of safety and fitness, the parameter the sound amplitude is a measure of occupational safety. For more information on occupational safety see https://www.osha.gov/pls/oshaweb/owadisp.show_document?p_table=STANDARDS&p_id=9735 href="https://www.osha.gov/Publications/laboratory/OSHAfactsheet-laboratory-safety-noise.pdf" target="_blank" rel="noreferrer noopener">https://www.osha.gov/Publications/laboratory/OSHAfactsheet-laboratory-safety-noise.pdf

For information on how to use the functions, see the BSP.h file and look for functions that begin with **BSP_Microphone**. For information on how the interface operates, see the BSP.c file and the data sheet for your microcontroller.

LCD for graphics[edit]

The MK-II has a color LCD for plotting data, drawing images, outputting text, and displaying numbers. There are a wide range of functions. The best way to learn how the LCD works is to first briefly review the available functions in BSP.h that begin with **BSP_LCD**. Next, you can look at example main programs that use the driver for display. The BoardSupportPackage project illustrates using the LCD to output text and numbers. The Lab1 starter project uses the LCD for both text and graphics. In Lab 4 we will use the LCD to display graphics for a hand-held game. The LCD is 128 by 128 pixels. The location (0,0) is in the upper left, (127,0) is upper right, (0,127) is lower left, and (127,127) is lower right. Each color pixel is 16 bits in RGB format of 5-6-5 bits. The BSP.h defines some standard colors

```
//color constants                red grn blue
#define LCD_BLACK      0x0000 //   0,   0,   0
#define LCD_BLUE       0x001F //   0,   0, 255
#define LCD_DARKBLUE   0x34BF //  50, 150, 255
#define LCD_RED        0xF800 // 255,   0,   0
```

```
#define LCD_GREEN      0x07E0 //   0, 255,   0
#define LCD_LIGHTGREEN 0x07EF //   0, 255, 120
#define LCD_ORANGE     0xFD60 // 255, 175,   0
#define LCD_CYAN       0x07FF //   0, 255, 255
#define LCD_MAGENTA    0xF81F // 255,   0, 255
#define LCD_YELLOW     0xFFE0 // 255, 255,   0
#define LCD_WHITE      0xFFFF // 255, 255, 255
```

Light and temperature sensors[edit]

The MK-II has light and temperature sensors. For more information on the light sensor, see the OPT3001 data sheet (opt3001.pdf). For more information on the temperature sensor, see the TMP006 data sheet (tmp006.pdf). Both sensors are integrated solutions implementing the sensor and interface electronics into a single package. Both use I2C communication to interface to the microcontroller. To take a measurement, the microcontroller issues a start command. Both sensors are extremely slow. The light sensor takes about 800 ms to convert. For example, to measure light we could execute

```
   BSP_LightSensor_Start();
   done = 0;     // it will take 800 ms to finish
   while(done==0){
     done = BSP_LightSensor_End(&lightData);
   }
```

The temperature sensor takes about 1 second to convert. Similarly, to measure temperature we could execute

```
   BSP_TempSensor_Start();
   done = 0;      // it will take 1000 ms to finish
   while(done==0){
     done = BSP_TempSensor_End(&tempData);
   }
```

The initialization functions configure the I/O ports for the light and temperature sensors. For information on how to use the functions, see the BSP.h file and look for functions that begin with **BSP_LightSensor** and **BSP_TempSensor**. For information on how the interface operates, see the BSP.c file and the data sheet for your microcontroller.

The MK-II sensors pose two very interesting challenges for this class. The first problem is synchronization. Even though temperature and light are fundamentally separate and independent parameters, the two sensors reside on the same I2C bus, therefore the software must manage these two devices in a coordinated fashion so that light and temperature activities do not interact with each other. The RTOS will need a mechanism to allow mutual exclusive access to the I2C bus. In a similar manner, the accelerometer, joystick and microphone all share the same ADC. Therefore the

RTOS must coordinate access to the ADC. The second challenge is timing. The labs will have three categories of devices

- Fast, on the order of 1 to 100us: switches, LED, and microphone
- Medium, on the order of 1 to 10ms: joystick and buzzer
- Slow, on the order of 1s: light and temperature

Processor clock[edit]

In order to make the labs in this class run on either the MSP432 or the TM4C123 we did three things. First, we created the BSP described in this section so the I/O interface to the MK-II has the same set of functions. In particular, the BSP.h for the MSP432 is the same as the BSP.h for the TM4C123.

Second, we created common I/O port definitions for the core elements like SysTick, PendSV and the nested vectored interrupt controller (NVIC). These definitions can be found in CortexM.h and CortexM.c. The names of these registers do not match either the TM4C123 or MSP432 definitions found in the Texas Instruments software examples. However the operation of the registers and the meaning of each bit obviously match, because these CortexM functions are implemented by ARM and exist on every Cortex M. For example, the following table shows the register names for the SysTick registers.

Register	TM4C123	MSP432
Current	NVIC_ST_CURRENT_R	SYSTICK_STCVR
Control	NVIC_ST_CTRL_R	SYSTICK_STCSR
Reload	NVIC_ST_RELOAD_R	SYSTICK_STRVR

Third, we abstracted time by implementing BSP_Clock functions. The MSP432 and TM4C123 run at different speed. After executing **BSP_Clock_InitFastest**, the MSP432 will run at 48 MHz. After executing this function on the TM4C123, the processor will be running at 80 MHz. The BSP maintains a 32-bit timer with a common resolution of 1us regardless of whether you are running on a MSP432 or TM4C123. For example, to initialize the timer, execute **BSP_Clock_InitFastest** and **BSP_Time_Init**. Now to measure the current time, one calls **BSP_Time_Get**, which will return the current system time in us. This system time does rollover every 71 minutes. Another time feature that runs similarly on both the MSP432 and the TM4C123 is **BSP_Delay1ms**. You can call this function to delay the specified number of ms.

Lab starter projects[edit]

Lab1_4C123

Lab overview
[edit]

The Lab 1 starter project using the LaunchPad and the Educational BoosterPack MK-II (BOOSTXL-EDUMKII) is a fitness device. It inputs from the microphone, accelerometer, light sensor and buttons. It performs some simple measurements and calculations of steps, sound intensity, and light intensity. It outputs data to the LCD and it generates simple beeping sounds. Figure Lab1.1 shows the data flow graph of Lab 1. Your assignment in Lab 1 is to increase the rate of Task0 from 10 to 1000 Hz.

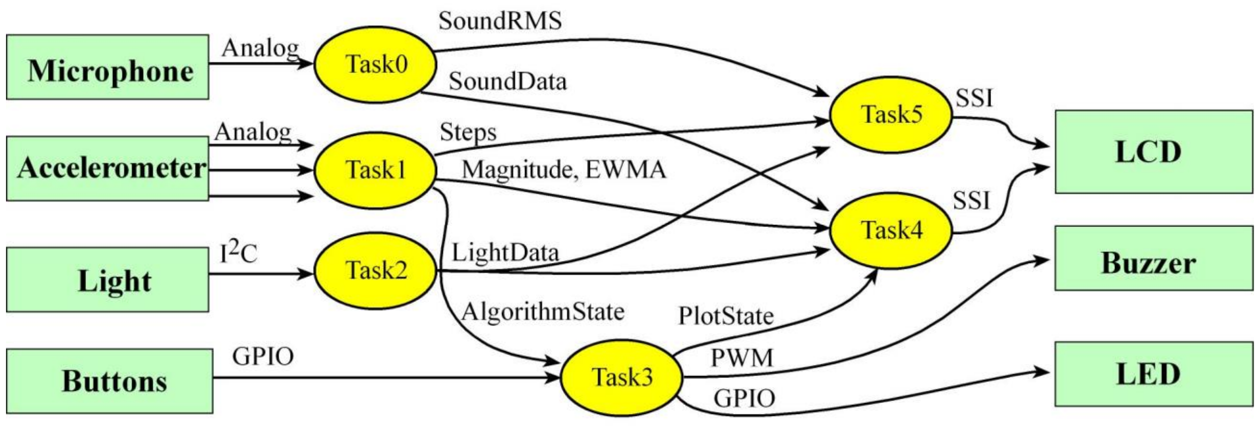


Figure Lab1.1. Data flow graph of Lab 1.

Play Video

This simple fitness device has six tasks. Normally, one would use interrupts to create real-time periodic events. However, Lab 1 will run without interrupts to illustrate the need for an operating system to manage multiple tasks that are only loosely connected. A very poorly constructed main program runs four of the tasks at about 10 times a second and the other two tasks at about once a second. One of the best ways to see how the six tasks fit together is to understand the data being passed.

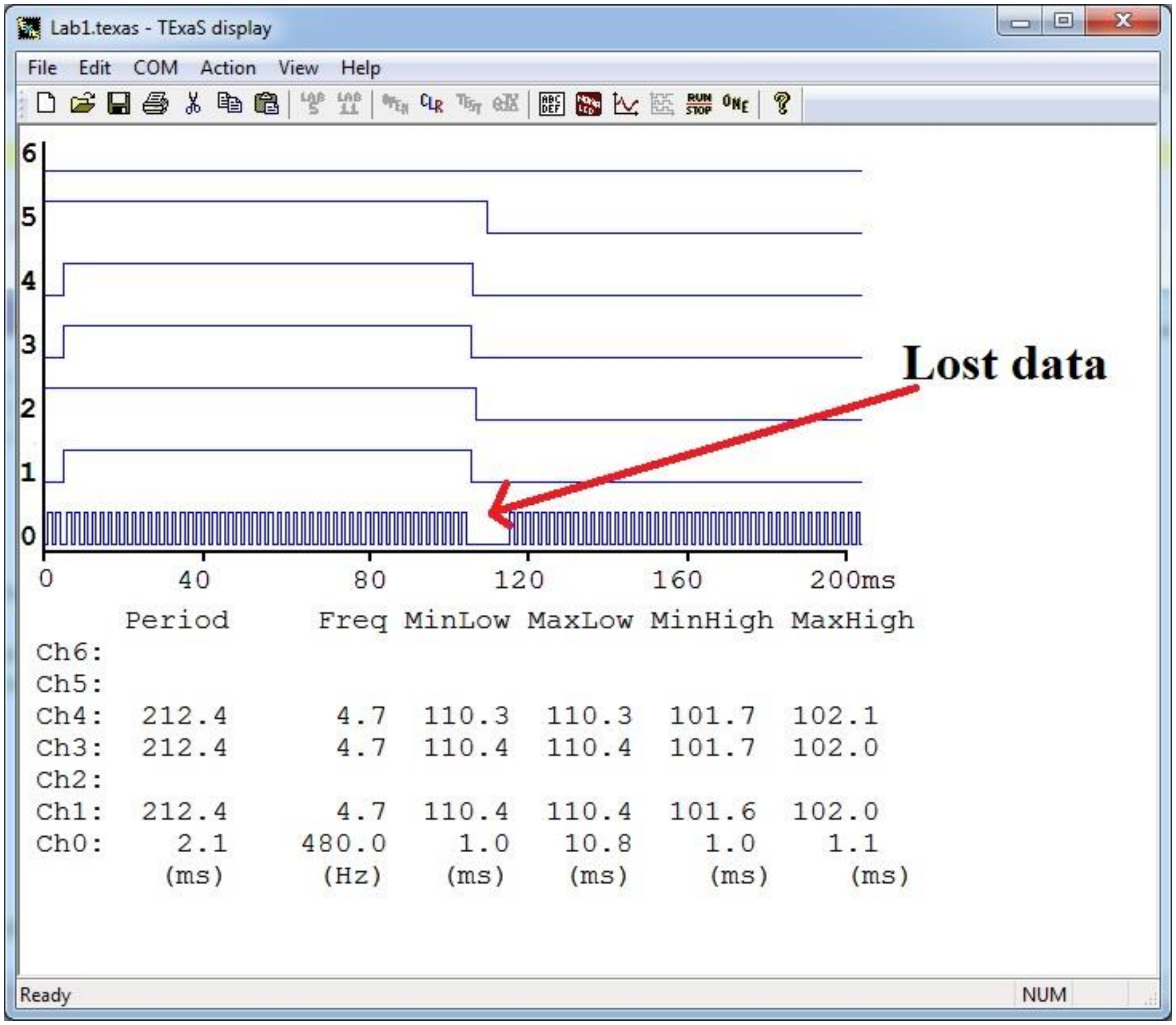
-
  - Task0: microphone input measuring RMS sound amplitude running at 10 Hz
    1. Reads sound from microphone (ADC)
    2. Sends **SoundData** to Task4
    3. Sends **SoundRMS** to Task5
  - Task1: acceleration input measuring steps running at 10 Hz
    1. Reads x,y,z acceleration (ADC)
    2. Sends **AlgorithmState** to Task3
    3. Sends **Magnitude**, **EWMA** to Task4
    4. Sends Steps to Task5
  - Task2: light input measure average light intensity running at 1 Hz
    1. Reads light from sensor (I2C)

    2. Sends **LightData** to both Task4 and Task5
- Task3: input from switches, output to buzzer running at 10 Hz
    1. Inputs from Buttons (GPIO)
    2. Sends **PlotState** to Task4
    3. Outputs to Buzzer (PWM)
    4. Outputs to LED (GPIO)
- Task4: plotting output to LCD running at 10 Hz
    1. Receives **SoundData**, **Magnitude**, **EWMA**, **LightData**, **PlotState**
    2. Outputs to LCD (SSI)
- Task5: numerical output to LCD running at 1 Hz
    1. Receives **SoundRMS**, **Steps**, **LightData**
    2. Outputs to LCD (SSI)

The main program manages these six tasks. We can define the real-time performance of this manager by measuring the time between execution of tasks. For example, the grader will measure when Task0 is started for the first *n* times it is run, $T_{0,i}$, for *i*=0 to *n*-1. From these measurements we calculate the time difference between starts. $\Delta T_{0,i} = T_{0,i} - T_{0,i-1}$, for *i* = 1 to *n*-1. Each task has a desired rate. Let Δt_j be the desired time between executions for task *j*. The grader will generate these performance measures for each for Task *j*, *j*=0 to 5:

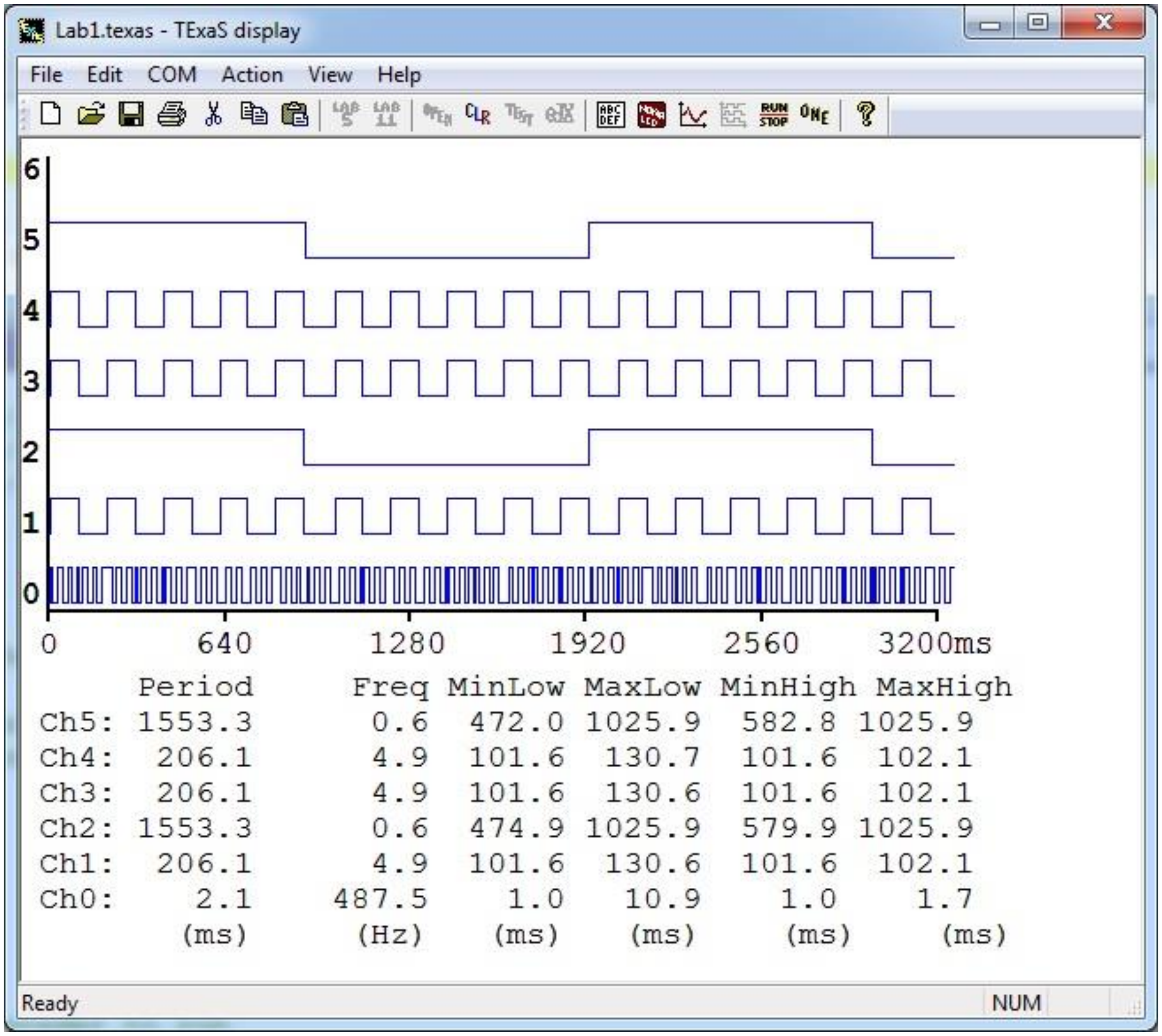
Min_j = minimum ΔT_j
Max_j = maximum ΔT_j
$Jitter_j = Max_j - Min_j$
Ave_j = Average ΔT_j
$Err_j = 100*(Ave_j - \Delta t_j)/ \Delta t_j$

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 1 toggles both a 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**. Calling **TExaS_Task0()** in grader mode performs the lab grading. However in logic analyzer mode, these calls implement the virtual logic analyzer and viewed with **TExaSdisplay**. Figure Lab1.2 shows a typical profile measured with the TExaS logic analyzer.



```
       Period      Freq MinLow MaxLow MinHigh MaxHigh
Ch6:
Ch5:
Ch4:  212.4       4.7  110.3  110.3  101.7  102.1
Ch3:  212.4       4.7  110.4  110.4  101.7  102.0
Ch2:
Ch1:  212.4       4.7  110.4  110.4  101.6  102.0
Ch0:    2.1     480.0    1.0   10.8    1.0    1.1
        (ms)      (Hz)   (ms)   (ms)    (ms)    (ms)
```

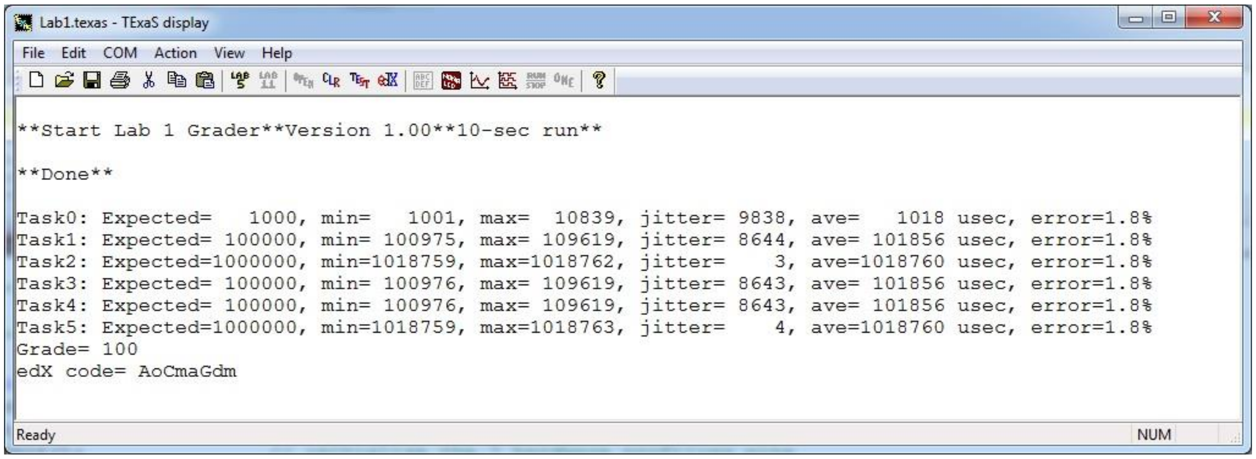
Figure Lab1.2. Task profile measured on the Lab 1 solution using TExaS. Notice Task0 is running about every 1ms. Notice Tasks 1, 3, and 4 are running about every 100ms. Notice that Task 0 looses samples whenever Tasks 2 and 5 run.


Lab1.texas - TExaS display
File Edit COM Action View Help
6
5
4
3
2
1
0
0 640 1280 1920 2560 3200ms
Ready
NUM

```
      Period      Freq MinLow MaxLow MinHigh MaxHigh
Ch5: 1553.3       0.6  472.0 1025.9  582.8 1025.9
Ch4:  206.1       4.9  101.6  130.7  101.6  102.1
Ch3:  206.1       4.9  101.6  130.6  101.6  102.1
Ch2: 1553.3       0.6  474.9 1025.9  579.9 1025.9
Ch1:  206.1       4.9  101.6  130.6  101.6  102.1
Ch0:    2.1     487.5    1.0   10.9    1.0    1.7
         (ms)      (Hz)   (ms)   (ms)    (ms)    (ms)
```

Figure Lab1.3. Task profile measured on the Lab 1 solution using TExaS. Notice Tasks 2 and 5 are running about every 1s.

Figure Lab1.4 shows the grader output of a typical solution to Lab 1.



```
**Start Lab 1 Grader**Version 1.00**10-sec run**

**Done**

Task0: Expected=   1000, min=   1001, max=  10839, jitter= 9838, ave=   1018 usec, error=1.8%
Task1: Expected= 100000, min= 100975, max= 109619, jitter= 8644, ave= 101856 usec, error=1.8%
Task2: Expected=1000000, min=1018759, max=1018762, jitter=    3, ave=1018760 usec, error=1.8%
Task3: Expected= 100000, min= 100976, max= 109619, jitter= 8643, ave= 101856 usec, error=1.8%
Task4: Expected= 100000, min= 100976, max= 109619, jitter= 8643, ave= 101856 usec, error=1.8%
Task5: Expected=1000000, min=1018759, max=1018763, jitter=    4, ave=1018760 usec, error=1.8%
Grade= 100
edX code= AoCmaGdm
```

Figure Lab1.4. Grader output for the Lab 1 solution.

Notice the software toggles the logic analyzer pin every time the thread runs. This results in a logic analyzer frequency that is 1/2 the thread frequency. I.e., a 500 Hz squarewave means the signal is toggled at 1000 Hz.

If you want to switch between text and graph mode TExaS display application as shown on Figure Lab 1.3 and Figure Lab 1.4 then select from View menu **Text I/O** or **Logic Analyzer**. If you want to trigger logic analyzer then select **View**->**Logic Analyzer Configuration** and select for example **Channel 3** -> **Falling**.

Debugging Lab 1[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. Without interrupts it will be quite difficult to bring the jitter down to acceptable values. Consequently, you will be graded only on average time between execution of tasks. Your assignment is to increase the rate of Task0 from 10 to 1000 Hz, while maintaining the existing rates of the other five tasks. More specifically, we are asking you to modify the main program, such that

- Task0: desired time between executions is 1000μs (starter code runs at 100,000μs)
- Task1: desired time between executions is 100,000μs
- Task2: desired time between executions is 1,000,000μs
- Task3: desired time between executions is 100,000μs
- Task4: desired time between executions is 100,000μs
- Task5: desired time between executions is 1,000,000μs

You can obtain full score if your system runs within 5% of these specifications.

Before you begin editing, downloading and debugging, we encourage you to first open up and run a couple of projects. The first two projects we recommend are **InputOutput_xxx** and **SysTickint_xxx**.

These projects interact with the switches and LEDs on the LaunchPad, and they will run with just the LaunchPad. Running these two projects will verify Keil, TExaS, and the windows drivers are properly installed.

Next, we encourage you should open up the project **BoardSupportPackage_xxx** and run it as described in Section 1.6. This project requires both the LaunchPad and the Educational BoosterPack MKII. This project illustrates many of the features on the MKII. Running this system will establish that all the lab components of this class are properly connected.

Third, we encourage you should open up the project **Lab1_xxx** and run it without editing. Lab1 requires both the LaunchPad and the Educational BoosterPack MKII. If the main program includes the call **TExaS_Init(LOGICANALYZER,1000);** logic analyzer data will be passed to the PC and will be visible in **TExaSdisplay** like Figure Lab1.2. We encourage you to run the starter code for Lab 1 in logic analyzer mode visualizing the task profile.

Play Video

Running the starter code

Next, we encourage you to activate the grader, changing the call to **TExaS_Init** to
TExaS_Init(GRADER,1000);
When you run the starter code in grading mode, you should see this output on **TExaSdisplay**.

Debugging with a real Logic Analyzer

Play Video

Lab 1 Grader[edit]

Grading your lab solution does require a LaunchPad development board. Your assignment is to increase the execution rate of Task0 from 10 to 1000 Hz, while maintaining the existing execution rates of the other five tasks. In particular, we are asking you to modify the main program, such that

- Task0 runs approximately every 1ms
- Task1 runs approximately every 100ms
- Task2 runs approximately every 1s
- Task3 runs approximately every 100ms
- Task4 runs approximately every 100ms
- Task5 runs approximately every 1s

You can obtain full score if your system runs within 5% of these specifications. Because we did not intend you to solve this with interrupts, we grade Lab 1 on average rates and not on jitter.

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.