Lab 4[edit]

The objectives of Lab 4 are to

- Rework scheduler to implement priority
- Add edge triggered interrupts to signal semaphores
- Signal semaphores from periodic interrupts
- Run event tasks using the regular scheduler

Play Video

Lab 4 is an incremental improvement over Lab 3. In particular, you will add priority to the TCB. If all threads have equal priority, then the system runs as round robin. There will be two types of interrupts: periodic and edge-triggered. On the occurrence of the interrupt, the OS will simply signal one or more semaphores, and then run the scheduler. All threads, including event threads and main threads, are run by the scheduler. Event threads will be assigned high priority to assure low jitter and low latency.

Just like Labs 2 and 3, the starter project will not execute until you implement the necessary RTOS functions. We encourage you to reuse code from Lab 3. Identical to Lab 3, the user code inputs from the microphone, accelerometer, light sensor, temperature sensor and switches. It performs some simple measurements and calculations of steps, sound intensity, light intensity, and temperature. It outputs data to the LCD and it generates simple beeping sounds. Figure Lab4.1 shows the data flow graph of Lab 4. Your RTOS will run eight main threads. Tasks 0, 1, 3 are event threads. The periodic interrupt will signal semaphores for Task0 and Task1. An edge triggered interrupt will signal a semaphore for Task 3.

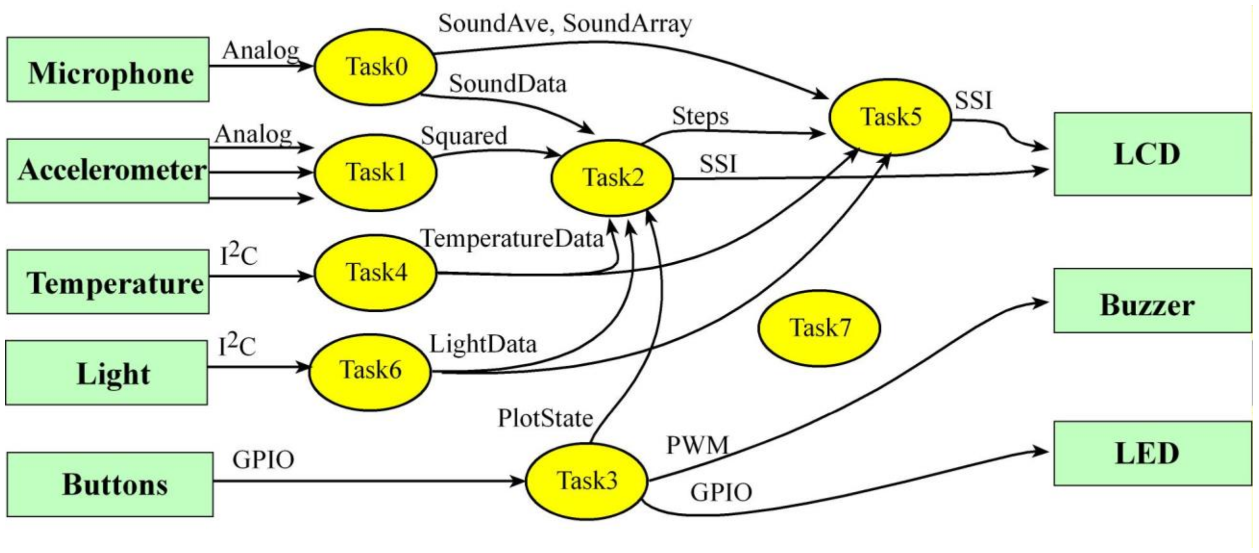

Figure Lab4.1. Data flow graph of Lab 4 (same as Lab 3).

This simple fitness device has eight tasks: eight main threads. Since you have two periodic threads to schedule, you could use one hardware timer to run both tasks, or you could use two hardware timers, one for each task. You will continue to use SysTick interrupts to switch between the six main threads. These are the eight tasks:

-
    - Task0: event thread samples microphone input at 1000 Hz
    - Task1: event thread samples acceleration input at 10 Hz (calls Put)
    - Task2: main thread detecting steps and plotting at on LCD, runs about 10 Hz (calls Get)
    - Task3: event thread inputs from switches, outputs to buzzer (calls Sleep)
    - Task4: main thread measures temperature, runs about 1 Hz (calls Sleep)
    - Task5: main thread output numerical data to LCD, runs about 1 Hz
    - Task6: main thread measures light, runs about 1.25 Hz (calls Sleep)
    - Task7: main thread that does no work

Thread 7, which doesn't do any useful task, will never sleep or block. Just like Lab 3, this thread will make your RTOS easier to implement because you do not need to handle the case where all main threads are sleeping or blocked.

Your RTOS manages these eight tasks. We will use the same metrics as described as used in Lab 3:

Min_j = minimum ΔT_j for Task *j*, *j*=0 to 5
Max_j = maximum ΔT_j for Task *j*, *j*=0 to 5
$Jitter_j = Max_j - Min_j$ for Task *j*, *j*=0 to 1
Ave_j = Average ΔT_j for Task *j*, *j*=0, 1, 3, 4, 5, and 6 (not 2 and 7)
$Err_j = 100*(Ave_j - \Delta t_j)/\Delta t_j$ 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. Tasks 0 to 6 toggle both the virtual logic analyzer and a real logic analyzer when they start. For example, **Task0** calls **TExaS_Task0()**. The first parameter to the function **TExaS_Init()** will be **GRADER** or **LOGICANALYZER**. Just like Labs 1-3, 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**. At the start of each task it also toggles an actual pin on the microcontroller. For example, **Task0** calls **Profile_Toggle0()**. You do not need a real logic analyzer, but if you have one, it can be used.

Lab 4 Logic Analyzer

Play Video

A real-time system is one that guarantees the jitters are less than a desired threshold, and the averages are close to desired values. We expect the jitter for the two periodic tasks to be quite low. 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 **Lab4.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 10us
- Task1: jitter between executions should be less than or equal to 30us
- Task2: average time between executions should be 100 ms within 5%
- Task3: runs at least three times during the grading period
- Task4: average time between executions should be less than 1.2 s
- Task5: average time between executions should be 1.0 s within 5%
- Task6: average time between executions should be less than 1.0 s
- Task7: no specifications

Debugging Lab 4[edit]

Step 1 First, we encourage you to open up the project **Lab4_xxx** and fully understand the system from the user perspective by reading through **Lab4.c**. Lab4 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 **Lab4.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; when using the logic analyzer this number does not matter. 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.

Before you begin editing, and debugging, we encourage you to open up **os.c** from Lab 3 and the **os.c** for Lab 4 and copy C code from Lab 3 to Lab 4 (do not move the entire file, just the insides of C functions as needed). *To not change the prototypes, just add implementations.* Similarly, copy the SysTick ISR from Lab 3 **osasm.s** to your Lab 4 **osasm.s**. The Lab 3 SysTick ISR should be identical as Lab 4.

Lab 4 Step 1

Play Video

Step 1) Increase the number of main threads from six to eight. Implement a priority scheduler that continues to support blocking and sleeping as defined in **OS.c** and **OS.h**. Extend your **OS_AddThreads** from Lab 3 to handle eight main threads, add a priority field to the TCB, and rewrite the scheduler to handle priority. The high priority threads will run frequently, while the low priority threads will run less frequently.

Step 2

Play Video

Step 2) In this step you trigger two semaphores from a timer interrupt. These semaphores will trigger real time tasks.

Step 3

Play Video

Step 3) In this step you trigger a semaphore from an edge-triggered interrupt. This semaphore will run a user function whenever you press a switch. The semaphore and sleeping will be deployed to debounce the switch.
Step 4) Debug your Lab4 using debugging windows and the TExaS logic analyzer. You should hear the buzzer when you press a switch. You should be able to see seven of the eight tasks running on the TExaS logic analyzer, and you should be able to see global variables **PlotState**change with buttons, see **TemperatureData** set by Task4, and see **LightData** set by Task6.

Grading Lab 4

Play Video

Grading your Lab 4 does require a LaunchPad development board. Your assignment is to implement a very simple RTOS that runs three event threads and five main threads. In particular, we are asking you to modify the main program, such that

- Task0 runs exactly every 1ms (jitter less than 10us)
- Task1 runs exactly every 100ms (jitter less than 30us)
- Task2 runs every 100ms (average within 5%)
- Task3 runs at least 3 times (push button 1)
- Task4 runs approximately every 1s (average less than 1.2s)
- Task5 runs approximately every 1s (average within 5%)
- Task6 runs approximately every 1s (average less than 1.0s)

The timing of Task7 is not graded. We do NOT want you to execute Tasks 0 and 1 during the execution of the SysTick ISR (inside of Scheduler). Rather, we expect you to execute Tasks 0 and 1 from a separate timer ISR (like BSP_PeriodicTask_InitB). Therefore, we expect the jitter on these two tasks to be small. Task 3 should run each time you press button 1. Because of the bounce, sometimes Task3 also runs when you release button 1.

Hand-held Video Game[edit]

There is a new ungraded Lab 4 that runs a hand held video game, called World Shapers. In the latest download you will see the **WorldShapers_xxx** project. If you finish your regular lab 4 you can import your Lab 4 OS into this project and run the video game. This version of the OS has dynamic thread creation and destruction. In particular, you can add threads one at time at run time. Furthermore, a thread can kill itself (OS_Kill), freeing its resources. Cool features

- **OS_Kill** removes a running thread using **PendSV**
- **OS_AddThread** adds one thread dynamically
- Up to 20 threads
- Converts BMP files to graphic images on the MK-II LCD
- Plays music on the 1-bit buzzer (see images folder)

To run the game.

1. Finish the Lab 4 priority scheduler.
2. Import the Lab 4 OS functions into the WorldShapers **os.c** and **osasm.s** files. Inside os.c, look for the phrase ****IMPLEMENT THIS**** and paste in your Lab4 solution.