

**ECE 3849 B2023**  
**Real-Time Embedded Systems**  
**Lab 3: Real Time Operating System**

This lab will port the application from the last two labs, a 1 Msps digital oscilloscope, to a RTOS (TI-RTOS).

To complete all labs in this course, you will need to submit your source for that lab, and demonstrate the lab in operation to one of the course staff (a TA or tutor). You will be graded on the completion of the steps in this lab document. After you have received a signoff, follow the instructions at the end of the document to submit your source code.

This lab is due on **Wednesday, November 29**. Late assignments will receive a 20% penalty.

Step	Max Points	Score
ADC ISR converted into an Hwi object, meets all deadlines.	10	
Waveform, Processing and Display Tasks implemented. Oscilloscope is drawing waveforms.	30	
User commands from the button Mailbox are processed and control the oscilloscope settings.	15	
CPU utilization measured and displays on screen	5	
Source Code & Report	25	
No shared data bugs handling options data or waveform data	10	
Total	100	

### Objectives

- Port an existing real-time application to an RTOS
- Use fundamental RTOS objects: Task, Hwi, Clock, Semaphore, Mailbox
- Deal with shared data and other inter-task communication

### Assignment

In this lab, you port all the oscilloscope functionality of the previous 2 labs to TI-RTOS, breaking up the project into multiple Task, Hwi, and Clock threads.


- All code must run in TI-RTOS threads: Task, Swi, or Hwi.
- Convert the ADC ISR into a Hwi, but preserve its low latency by configuring it as a “zero-latency interrupt”
- Create the Waveform Task (highest-priority) that searches for the trigger and copies the triggered waveform into a waveform buffer (a shared global array). It then signals the Processing Task.
- Create the Processing Task (lowest-priority) that scales the captured waveform in oscilloscope mode, or applies FFT to the waveform in spectrum mode. (Use a separate buffer for the processed waveform.) It then signals the Display Task and the Waveform Task, in that order.
- Create the Display Task (low-priority) that draws a complete frame (grid, settings and waveform) to the LCD screen.
- Button handling
  - Schedule the periodic button scanning using the Clock module. The Button Clock function should signal the Button Task using a Semaphore.
  - Create the Button Task (high-priority) that scans the buttons and places button IDs into a Mailbox object.
  - Create the User Input Task (mid-priority) that processes the user input that it receives through the Button Mailbox. If no buttons are being pressed, this Task remains blocked waiting on the Mailbox. When the user makes changes to the settings, this Task signals the Display Task to redraw the screen.

### Part 0: Import TI-RTOS Project

Here, you will import a TI-RTOS project and copy over your code from the previous labs.

While it is possible to start with one of the pre-configured TI-RTOS example projects, you would have to remove a lot of features we will not use. The main feature we will ignore is the set of device drivers that comes with TI-RTOS. For our purposes, these device drivers are just a different interface to TivaWare, and are also incomplete. Instead, we will simply use TivaWare.

You now have an almost empty project linked with TI-RTOS. The primary files you will modify are `main.c` and `rtos.cfg`. Opening `rtos.cfg` should bring up the TI-RTOS configuration GUI. If `rtos.cfg` shows up as a script instead, close it, then right-click it and select `Open With` → `XGCONF`. The basic features of `rtos.cfg` are explained as you navigate the different objects in it.

You can click the  icon while viewing any part of `rtos.cfg` GUI to bring up detailed documentation on every object you encounter. All the objects controlled by the `rtos.cfg` GUI are listed in the Outline window. You can add more TI-RTOS modules to your system through the Available Products window (if not visible, open the menu `View` → `Other...` and type in “Available Products”). The starter project features only a single Task `task0` to demonstrate its configuration in `rtos.cfg` and its code in `main.c`. You should remove or rename this Task.

Perform hardware initialization with interrupts globally disabled in `main()` before starting the OS. It is recommended to globally enable interrupts in one of the Tasks, to prevent ISRs from running before the OS is initialized. The OS does not globally enable interrupts during initialization. Note that TI-RTOS configures the FPU, clock generator, interrupt controller and (periodic/one-shot) timers on its own. You should not initialize these peripherals manually.

Copy your Lab 1 source (.c and .h) files into your new Lab 2 project. Do not copy `tm4c1294ncpdt_startup_ccs.c` or `main.c`. Copy the text of your Lab 1 `main.c` into the Lab 2 `main.c`, and comment out that old code. You will gradually move pieces of that code into their final locations and then uncomment them.

Place the PWM signal generator initialization code in `main()` before the `BIOS_start()` call.

### Part 1: ADC Hwi

In this part, you will port the ADC ISR to be a TI-RTOS Hwi.

The ADC ISR and setup code should remain nearly unchanged. The only difference is that the ADC ISR must be configured as a TI-RTOS Hwi object. **Use the M3 specific Hwi module.**

The Hwi module sets up the interrupt vector table and priorities, so **none of the interrupt controller setup code should remain** (no driver functions starting with `Int`, except for `IntMasterDisable()` and `IntMasterEnable()`).

The ADC ISR must be a zero-latency interrupt, enabled even when the TI-RTOS scheduler is running. This should permit this ISR to execute every 1  $\mu$ s without interference from the critical sections in the OS, at the expense of forbidding any OS calls from the ISR.

First, create a Hwi object in the `rtos.cfg` using the M3 specific Hwi module. In the Hwi Module settings, specify “Priority threshold for `Hwi_disable()`” = 32, and make the Hwi you will use for the ADC ISR have a priority of 0 (the highest priority on the interrupt controller). This will ensure that the ADC ISR is a zero-latency interrupt.

Next, you will need to specify the interrupt number. Look it up in the TM4C1294NCPDT datasheet in Table 2-9 under Vector Number (not Interrupt Number – yes, confusing).

Finally, point this HWI to your ADC ISR function.

### Part 2: Waveform Processing

In this section, you will split the display functionality of main into three TI-RTOS Tasks, the Display task, the Processing Task, and the Waveform task.

As each task will have its own stack, you may need to increase the stack size of the Display and Processing tasks, which perform memory intensive operations such as string formatting.

First, you will create the **Waveform** task. The Waveform task will block on a semaphore. When it is signaled, it will perform the trigger search, then copy the captured waveform into a buffer, and signal the Processing task. This task should be the highest priority, to ensure trigger search can complete before the ADC overwrites the buffer. If necessary, protect access to the shared waveform buffer (not the ADC buffer).

The **Processing** task will also block on a semaphore. When it is triggered, it will scale the captured samples for display, and store them in a separate buffer. Once the signal has been scaled, it will signal the Display task to draw the buffer onto the screen. After it signals the Display task, it will signal the Waveform task to capture the signal again. The processing task may seem redundant for now, but will be more useful in Lab 4 when more intensive processing will be done. This task should be lower priority than display or user interface tasks.

As with the other tasks, the **Display** task will wait on a semaphore for other tasks to signal a screen update. When triggered, it should draw one frame to the LCD, then block again. Be sure to protect accesses to shared data such as the processed buffer and settings.

### Part 3: Button Scanning

Here, you will convert user input to use TI-RTOS objects.

As the TI-RTOS will be handling the timer now, the timer initialization code will no longer be needed and should be removed.

Configure a **Clock object** to schedule button scanning. First, initialize the Clock **Module** to a 5 ms clock tick period and “Internally configure a Timer to periodically call Clock\_tick()”. Specify a different “Timer Id” if you would like to use Timer0 for something else.

Add a Clock Instance and the **Button task**. The Clock function should signal the Button task using a **Semaphore**. This Task should scan the buttons, and if a press is detected, will post the button ID to a **Mailbox** for further processing by a lower-priority Task. Make sure the button clock function and the Button task no longer access any timer hardware (this is now the RTOS’s responsibility). The Clock Instance requires an initial timeout of at least 1 tick. Check the box to start the Clock Instance at boot time. The Button task should be high priority, to ensure user input is not missed.

Add the **User Input task** (medium priority) that pends on the button mailbox (blocking call). When it receives a button ID, it should modify the oscilloscope settings, signal the Display task (using a Semaphore) to update the LCD screen, then go back to waiting on the mailbox. Be careful here as the oscilloscope settings are shared data. Make sure no serious shared data bugs arise. Although the Button task and the User Input task have similar priority, keep them separate. This separation enforces encapsulation (the Button Task is the only Task accessing the button peripherals) and allows for the possibility of inserting a Task of intermediate priority between these tasks.

### Part 4: CPU Load Measurement

Now, you will port the CPU Load Measurement to TI-RTOS.

CPU Load should be measured in the Display task, as we are primarily concerned with whether there is enough CPU time for the display to be drawn at a reasonable rate. The background load should be measured before starting the RTOS.

### Part 5: Submitting Source Code

Make sure your Lab 3 project is named “**ece3849\_lab3\_username**” with your username substituted. **Clean** your project (remove compiled code). Right click on your project and select “Export...” Select General → “Archive File.” Click Next. Click Browse. Find a location for you archive and specify the exact same file name as your project name. Click Finish. Submit the resulting .zip file to Canvas. This concludes Lab 3.