# ECE 3849 D2021
# Real-Time Embedded Systems
# Lab 2: RTOS, Spectrum Analyzer

## Learning 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.
- Run a computationally intensive task without slowing down the user interface.

## Deliverables

There are three deliverables for this lab:

- Lab signoff completed: 65 points
- Archived project submitted to canvas: 20 points
- Lab report submitted to canvas: 15 points

Each deliverable has a rubric at the end of this report.

To receive credit for the code and report submissions you must sign off.

Any deliverable submitted after the deadline in canvas, will receive a 20% late penalty and can be submitted up to a week late. No submissions after one week late will receive credit.

## Restrictions

- You may use TivaWare, TI Graphics Library and other libraries. External / non-TI provided libraries should not be needed.
- Code should be written in C.
- If the lab specifically calls out the use of a specific RTOS Object or calls out certain implementation details, the instructions must be followed to receive full credit.

## Required External Code / Documents

- ece3849_lab2_starter_d21.zip
- kiss_fft130.zip
- The code being ported from Lab1 must be your own Lab1 submission.

## Lab Overview

In this lab you will port the Lab 1 application, 1 Msps digital oscilloscope to run using an RTOS (TI-RTOS). You will also add an FFT mode that turns your system into a simple spectrum analyzer.
Your existing Lab1 project functionality will be broken up up into multiple Task, Hwi and Clock threads. You do not need to port the Lab 1 CPU load or extra-credit code. Required differences in implementation from Lab 1:

- All code must run in TI-RTOS threads: Task, Swi or Hwi.
- Convert the ADC ISR into an 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. Selectable rising and falling edge triggering is still required.
- 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 scan 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.
    - Selectable rising and falling edge triggering via button presses is required. Amplitude and time control from lab1 are not required and are not extra credit in lab 2 but can be preserved.

An additional feature compared to Lab 1 is **spectrum (FFT) mode**. Switch between the FFT and oscilloscope modes using one of the buttons. In FFT mode, some of the existing Task functionality should change:

- The Waveform Task copies the 1024 newest ADC samples without searching for trigger. The newest samples are the 1024 samples taken prior to the current gADCBufferIndex. Using these will avoid shared data issues.

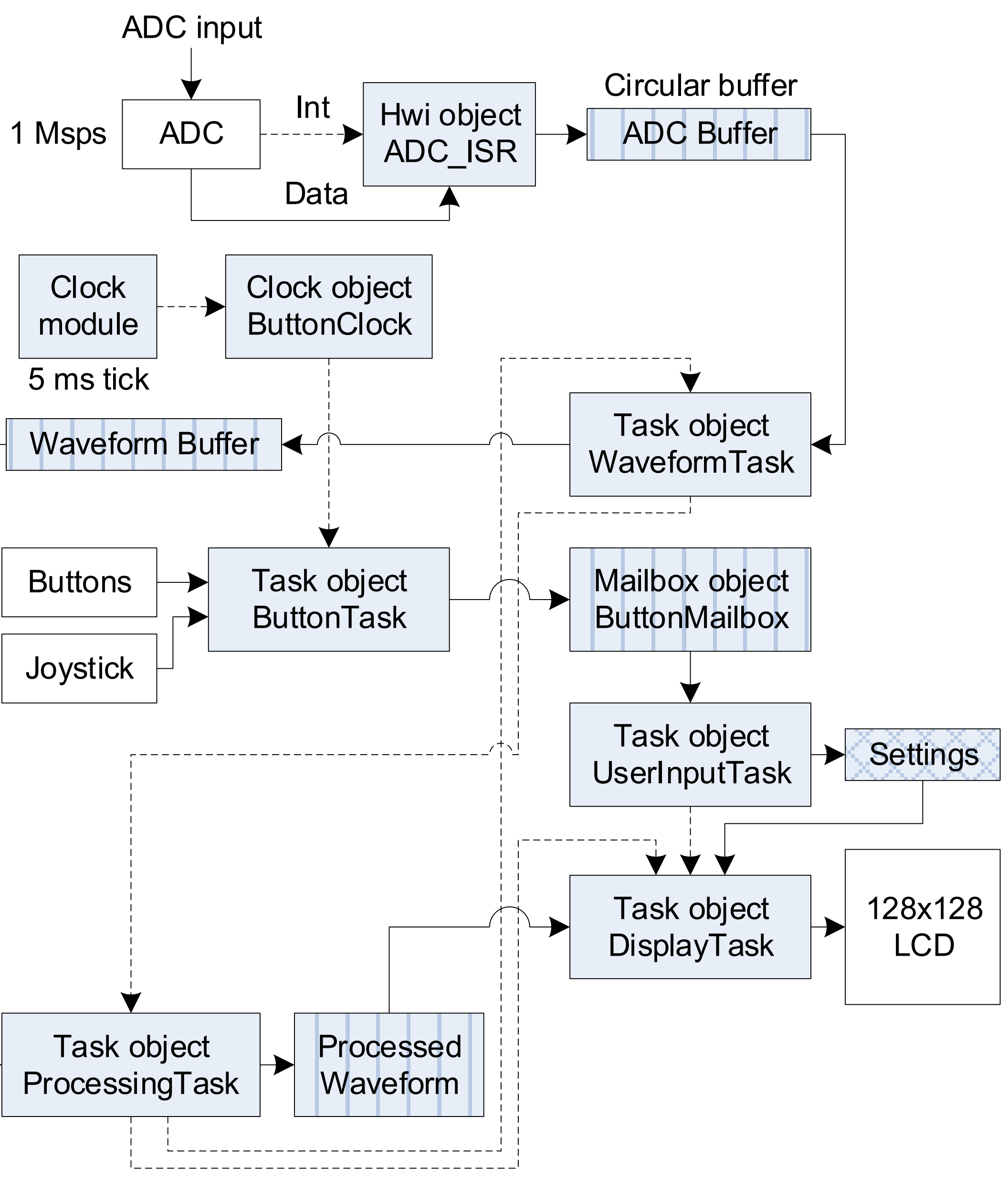
- The Processing Task performs a 1024-point FFT on the captured samples and converts the lowest 128 frequency bins to an integer 1 dB/pixel scale for display. A floating point FFT library is provided to you together with the assignment. This Task is meant to take significant time to execute to test the prioritization and preemption of RTOS Tasks.
- The Display Task shows the frequency and dB scales in FFT mode (instead of time and voltage). These scales do not need to be adjustable. The frequency grid should start at x = 0 (zero frequency).

For **extra credit**, measure the real-time parameters of the Button Task and the trigger search code in Waveform Task, and fill the table below. For trigger search, measure the max time your Waveform Task takes to search for the trigger and copy the waveform from the ADC Buffer. Determine the relative deadline for each of these Tasks.

| Task | Latency | Response Time | Relative Deadline |
|---|---|---|---|
| Button Task | | | |
| Trigger search and waveform copy in Waveform Task | N/A | | |

## Recommended Implementation

The following figure outlines the recommended structure of the Lab 1 digital oscilloscope ported to TI-RTOS.



Software threads and modules are represented by smooth shaded boxes. If an OS object is to be used, it is indicated in the box. Important shared data and inter-task communication objects use textured shading. Clear boxes indicate hardware. Two types of connections are distinguished: solid lines = data flow, dotted lines = signaling/synchronization. **Threads are arranged vertically by priority**.

## Step 1: TI-RTOS Project

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.

Do not start from any of the TI-RTOS examples, **download the ece3849_lab2_starter project from the Canvas Lab 2 area**. Rename your project to **"ece3849d21_lab2_username,"** substituting your username.

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 in the lower left-hand corner below the project explorer (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.

## Step 2: ADC Hwi

The ADC ISR and setup code should remain nearly unchanged from Lab 1. The only difference is that the ADC ISR must be configured as a TI-RTOS Hwi object. Use the **M3 specific Hwi module (ti.sysbios.family.arm.m3)**. The Hwi module sets up the interrupt vector table and priorities, so **none of the interrupt controller setup code should remain** from Lab 1 (no driver functions starting with **Int**, except for IntMasterDisable() and IntMasterEnable()).

In the Hwi Module settings tab, specify "Priority threshold for Hwi_disable()" = 32. This leaves all higher priority ISRs enabled even while TI-RTOS runs the scheduler. Make the ADC Hwi priority 0 (highest priority on the interrupt controller). This should permit this ISR to execute every 1 μs without interference from the critical sections in the OS, at the expense of forbidding any OS calls from the ISR.

In the ADC Hwi object Instance settings tab, 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).

## Step 3: Waveform Processing

Break the functionality of the Lab 1 main() into several Tasks. Semaphore used in the tasks should be configured to select Semaphore type Binary FIFO. The Semaphore_pend() function is used for blocking / waiting operations and the Semaphore_post() functions are used for signaling.

The **Waveform Task** (highest priority) should block on a Semaphore. When signaled, it should search for trigger in the ADC buffer, copy the triggered waveform into the waveform buffer, signal the Processing Task, and block again. This is the highest priority Task to make sure the trigger search completes before the ADC overwrites the buffer. Protect access to the shared waveform buffer (not the ADC buffer), if necessary.

The **Processing Task** (lowest priority) should block on a Semaphore. When signaled, it should scale the captured waveform for display, but not draw it. Instead, store the processed waveform in another global buffer. Then, signal the Display Task to actually draw this waveform. Subsequently, signal the Waveform Task to capture another waveform. Finally, go back to waiting on its Semaphore. The Processing Task may seem redundant, but it does a lot more processing in spectrum mode. This time-consuming processing should be done at a lower priority than the user interface (User Input Task & Display Task).

The **Display Task** (low priority) should block on a Semaphore waiting for other Tasks to signal a screen update. When signaled, it should draw one frame including the settings and the waveform (output of the Processing Task) to the LCD, then block again. Be careful accessing the settings and the processed waveform buffer, as they are shared data.

You may need to increase the **stack** size for the Processing and Display Tasks. You can monitor the Task max stack usage through RTOS Object View (see the end of this document).

## Step 4: Button Scanning

The **Clock object** instance should be configured to schedule periodic button scanning. In the Clock **Module** Settings tab, set the tick period to a 5 ms in the Timer Control section and select "Internally configure a Timer to periodically call Clock_tick()" in the Time Base section. Specify a different "Timer Id" in the Timer Control section, if you would like to use Timer0 for something else. Note that you should remove your Lab 0/1 button timer initialization code. The RTOS handles this now.

Add a Clock Instance and the **Button Task** (high priority). The Clock function should signal the Button Task using a **Semaphore**. This Task should scan the buttons, and if a press is detected, posts 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). In the Clock Instance Settings tab, the initial timeout should be set to least 1 tick. The Period setting should also be 1 to configure the button scanning period to 5 msec (200 Hz) as it was in Lab1. Check the box next to "Start the boot time when instance is created".

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.

## Step 5: Spectrum (FFT) Mode

Pressing one of the user buttons should switch your oscilloscope into **spectrum mode**. Pressing the same button again should switch back oscilloscope mode. In spectrum mode, some of the Task functionality should change:

- The Waveform Task should capture the newest 1024 samples without searching for trigger. The newest samples are the 1024 samples taken prior to the current gADCBufferIndex. Using these will avoid shared data issues.
- The Processing Task should use the Kiss FFT package to compute the 1024-point FFT of the captured waveform in single precision floating point. It should then convert the lowest 128 frequency bins of the complex spectrum to magnitude in dB and save them into the processed waveform buffer, scaled at 1 dB/pixel for display.
- The Display Task should print the frequency and dB scales rather than the time and voltage scales. It should also modify the grid, such that the first vertical grid line is at x = 0, corresponding to zero frequency. Keep the grid spacing 20 pixels in both directions. The waveform drawing code should remain unchanged (rely on the Processing Task to prepare the waveform for display), aside from the color.

The Kiss FFT package is relatively easy to run. You only need to copy into your project the .c and .h files from the **root kiss_fft130 folder**, not subfolders. Read the README and the comments in kiss_fft.h. The only annoying issue is how to allocate the Kiss FFT cfg/state buffer. The following example initializes the Kiss FFT library for 1024 samples and computes one FFT of a preprogrammed sine wave.

```
#include <math.h>
#include "kiss_fft.h"
#include "_kiss_fft_guts.h"

#define PI 3.14159265358979f
#define NFFT 1024         // FFT length
#define KISS_FFT_CFG_SIZE (sizeof(struct kiss_fft_state)+sizeof(kiss_fft_cpx)*(NFFT-1))
```

```
static char kiss_fft_cfg_buffer[KISS_FFT_CFG_SIZE]; // Kiss FFT config memory
size_t buffer_size = KISS_FFT_CFG_SIZE;
kiss_fft_cfg cfg;                       // Kiss FFT config
static kiss_fft_cpx in[NFFT], out[NFFT]; // complex waveform and spectrum buffers
int i;

cfg = kiss_fft_alloc(NFFT, 0, kiss_fft_cfg_buffer, &buffer_size); // init Kiss FFT
```

```
for (i = 0; i < NFFT; i++) {    // generate an input waveform
  in[i].r = sinf(20*PI*i/NFFT); // real part of waveform
  in[i].i = 0;                  // imaginary part of waveform
}

kiss_fft(cfg, in, out);         // compute FFT

// convert first 128 bins of out[] to dB for display
```

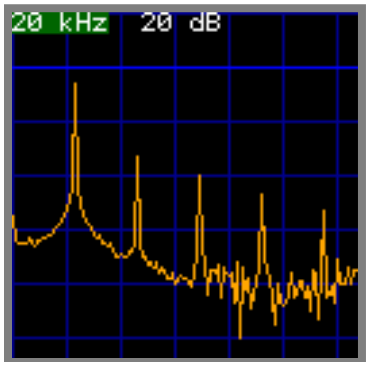
The three sections are includes/defines, initialization (place this code before the infinite loop of the Processing Task), and computation (in the Processing Task's infinite loop).

The test input sinusoid has a period of 102.4 samples (1/10 of the total number of samples). The output should be mostly zeros, except `out[10].i = -512` and `out[1014].i = 512`. When done testing, convert the waveform generator code to copying from the Waveform Buffer.

Convert the spectrum to dB one point at a time as follows:

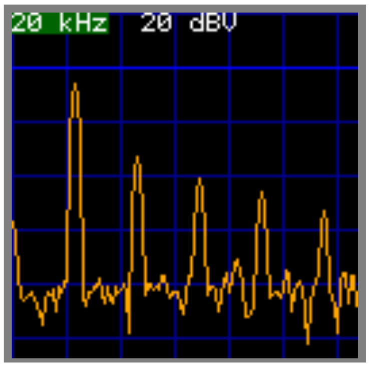
```
out_db[i] = 10 * log10f(out[i].r * out[i].r + out[i].i * out[i].i);
```

You may need to add an offset to move the spectrum into the y-coordinate range of the display. The dB scale does not need to be calibrated to any specific voltage level, but the spectrum peaks and the noise floor should be visible. See the screen capture below for an example of what the spectrum analyzer output should approximately look like. (Performance note: The Cortex-M4F supports only single-precision floating point natively. Floating-point calculations in C default to double precision. To enforce single precision, use the "f" suffix on numbers and math functions.)



Optionally, you may window your time-domain waveform before the FFT to reduce spectral leakage (the skirt-like signal drop-off around spectral peaks). Multiply your 1024 time-domain samples by the corresponding window function samples given to you below, then FFT the resulting waveform. The spectrum should look like that in the following screen capture.

```
static float w[NFFT]; // window function
for (i = 0; i < NFFT; i++) {
    // Blackman window
    w[i] = 0.42f
           - 0.5f * cosf(2*PI*i/(NFFT-1))
           + 0.08f * cosf(4*PI*i/(NFFT-1));
}
```



## Extra Credit: Real-time Measurements

We have been carefully monitoring ADC1 for missed deadlines. Now do the same for the remaining two real-time deadlines in this system: button scanning and trigger search.

For button scanning, measure the max latency and response time of the Button Task. You may use the **ece3849_rtos_latency** CCS project on the course website as a reference. The RTOS Clock module uses a hardware timer to schedule system tick interrupts. Read the timer count to determine the time since the last interrupt. Determine if a deadline was missed by checking the count of the Semaphore used to signal the Button Task. Exercise your code by pressing buttons.

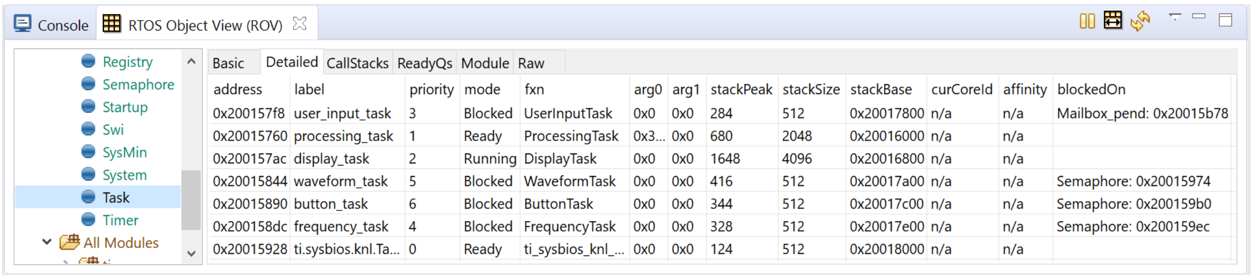
For trigger search, measure the max time your Waveform Task takes to search for the trigger and copy the waveform from the ADC Buffer in Oscilloscope mode. You may use the TI-RTOS Timestamp module (need to add it in rtos.cfg) or any free hardware timer for this measurement. Review the Lab 1 assignment for an explanation of the real-time requirements for trigger search. Note that the worst-case trigger search time is when the trigger is not found. To force this condition, disconnect your AIN3 input.

Fill in the following table to show to the grader during your sign-off and include an analysis of your findings in the report. Use actual time units, not CPU clock cycles.

| Task | Latency | Response Time | Relative Deadline |
|---|---|---|---|
| Button Task | | | |
| Trigger search and waveform copy in Waveform Task | N/A | | |

## Debugging Features of the RTOS

When debugging your software running on an RTOS, it helps to be able to examine the state of all RTOS objects: which Tasks are blocked, Semaphore counts and waiting lists, etc. TI-RTOS provides a specialized watch window for RTOS objects in CCS. Navigate the menu to Tools → ROV Classic. To find out what is blocking your Tasks, you will need to examine the Semaphores and Mailboxes, not just the Tasks. Under Detailed information for a Task you see what it is blocked on, but the blocking object is identified only by its address, not name. You can also observe the peak stack usage for each task, and whether any stack has overflowed.



| address | label | priority | mode | fxn | arg0 | arg1 | stackPeak | stackSize | stackBase | curCoreId | affinity | blockedOn |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x200157f8 | user_input_task | 3 | Blocked | UserInputTask | 0x0 | 0x0 | 284 | 512 | 0x20017800 | n/a | n/a | Mailbox_pend: 0x20015b78 |
| 0x20015760 | processing_task | 1 | Ready | ProcessingTask | 0x3... | 0x0 | 680 | 2048 | 0x20016000 | n/a | n/a | |
| 0x200157ac | display_task | 2 | Running | DisplayTask | 0x0 | 0x0 | 1648 | 4096 | 0x20016800 | n/a | n/a | |
| 0x20015844 | waveform_task | 5 | Blocked | WaveformTask | 0x0 | 0x0 | 416 | 512 | 0x20017a00 | n/a | n/a | Semaphore: 0x20015974 |
| 0x20015890 | button_task | 6 | Blocked | ButtonTask | 0x0 | 0x0 | 344 | 512 | 0x20017c00 | n/a | n/a | Semaphore: 0x200159b0 |
| 0x200158dc | frequency_task | 4 | Blocked | FrequencyTask | 0x0 | 0x0 | 328 | 512 | 0x20017e00 | n/a | n/a | Semaphore: 0x200159ec |
| 0x20015928 | ti.sysbios.knl.Ta... | 0 | Ready | ti_sysbios_knl_... | 0x0 | 0x0 | 124 | 512 | 0x20018000 | n/a | n/a | |

# Sign-off Procedure

**Every lab MUST be signed-off to receive credit for the lab.**

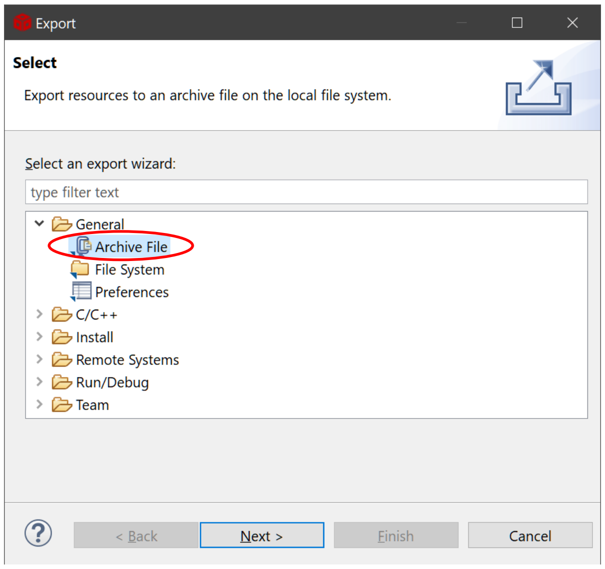
Signoff can be done in-person in AK113 during the officially scheduled lab times on Thursday 10-1:50 or 2-4:50 or via zoom during any lab time or office hour on the Weekly calendar.

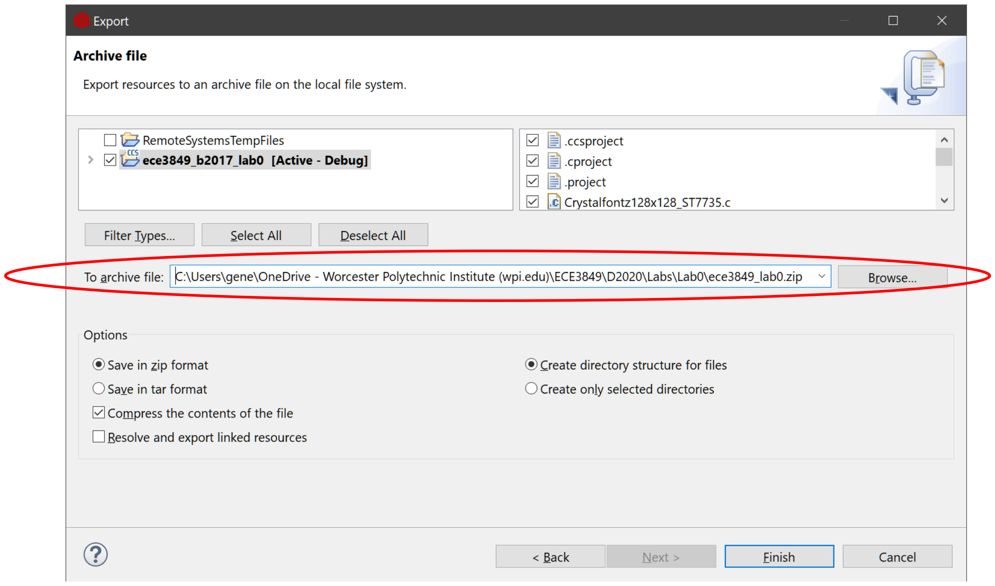
If signing-off remotely verify that the functions of your Lab kit can be clearly displayed with zoom's video capability prior to signing off. This may require a use of an external camera, phone , tripod or other equipment.

1) Have your CCS project open and ready to show the TA.
2) The TA will ask you to give a brief walk through the code that you wrote or modified and briefly talk about what was done where in your code.
3) With the TA present Build and Run your CCS project. TA will verify there are no preventable warnings in the code after building.
4) The TA will go through the sign-off check-list and ask you to demonstrate each function. The TA will record results and assign points for all functionality that is correct.
   a) You can demo as many times as needed.
   b) You will always be rewarded points for the functions that are work.
5) Once you are happy with your demonstration results.
   a) You will archive the project that you just demoed and submit the generated zip immediately to the Code Archive assignment in canvas.
   b) The TA will verify the archived project is submitted and complete the sign-off process.

## Instructions for archiving

i) Make sure your Lab 2 project is named "**ece3849d21_lab2_username**" with your username substituted.
ii) **Clean** your project (remove compiled code).
iii) Right click on your project and select "Export..." Select General → "Archive File." Click Next.
iv) Click Browse. Find a location for you archive and specify the exact same file name as your project name.
v) Click Finish.





[ ] RemoteSystemsTempFiles
[✓] **ece3849_b2017_lab0 [Active - Debug]**

[✓] .ccsproject
[✓] .cproject
[✓] .project
[✓] Crystalfontz128x128_ST7735.c

To archive file: C:\Users\gene\OneDrive - Worcester Polytechnic Institute (wpi.edu)\ECE3849\D2020\Labs\Lab0\ece3849_lab0.zip

Options
- [✓ Save in zip format]
- Save in tar format
- [✓ Compress the contents of the file]
- Resolve and export linked resources
- [✓ Create directory structure for files]
- Create only selected directories

# Sign-Off Check List (65 points)

## Project Status ( 10 points)

1. Student Walk through (5 points)
    - Student is able to give a brief walk through of their work. Showing where and how functionality are implemented specifically how the Lab1 code fit into the Lab2 tasks. If things are unclear, student should be able to answer questions about their code for clarification.
    - Review the rtos.cfg configurations. Verify Button, display, processing, user_input and waveform tasks and their corresponding semaphores are present. Verify Hwi configuration for ADC and Mailbox for button synchronization.
2. No preventable warnings (3 points)
    - Check the problems window, verify that all preventable warnings are resolved.
3. Project Builds and Runs ( 2 points)

## Functionality Check List

1. Verify gADCErrors (5 points)
    - Run code with gADCErrors displayed continuously updating should stay constant when running and increment by one when paused.
2. Waveform display has proper formatting (5 points)
    - Gridlines are in correct locations.
    - Text at top of screen shows time/division, volts/division and rising / falling trigger text or icon.
    - Waveform is properly centered with trigger in the middle of the screen.
3. Waveform is properly displayed (15 points)
    - Disconnect input and verify flat line.
    - When PWM connected signal has expected swing and frequency at 1V/division and 20 usec/division.
    - Drawn using lines with smooth look.
4. User commands from the button Mailbox are processed and control the oscilloscope. (8 points)
    - When a button is pushed the trigger slope toggles between rising and falling and the trigger text / icon is updated.
    - When button is pushed switches between FFT and waveform modes.
5. Spectrum mode (FFT) Grid and Scale are to specification (5 points)
    - Vertical grid lines start at x=0 with 20 pixel spacing.
    - Horizontal grid lines are centered with 20 pixel spacing.
    - Text at top of screen showing frequency/division and dB/division.

(continued on next page)

6. FFT is properly displayed (15 points)
   - FFT waveform is shown clearly and centered vertically such that values are not clipped at top or bottom of screen.
   - FFT looks appropriate for square wave with frequency components at multiples of 20k.
   - When the signal is removed FFT shows large DC value with no peaks for the rest of the plot.
7. Extra Credit: Button Task timing measurements (3 points)
   - Explain the measurement taken and what the relative deadline is. Show the results in the debugger that are used to measure the response time.
8. Extra Credit: Trigger search and waveform copy in Waveform Task (3 points)
   - Explain the measurement taken and what the relative deadline is. Show the results in the debugger that are used to measure the response time.

## Canvas Submissions Completed (2 points)

1. Archived project submitted in canvas at time of sign-off

## Source Code Rubric (20 points)

- [8 pts] Tasks and Synchronization Objects written to lab specification
  - Verify Waveform, Processing and Display Tasks are implemented using the required synchronization tasks.
  - Review the rtos.cfg settings, verify Button, display, processing, user_input and waveform tasks are present and their corresponding semaphores. Verify Hwi configuration for ADC and Mailbox for button synchronization.
  - If student has implemented the extra credit review how it was done.
- [3 pts] Performance where needed
  - Do not perform extraneous computations in performance-sensitive code, such as ISRs, critical sections, etc.
  - Use direct register access in the shortest-period ISRs.
  - Avoid excessive optimization where performance is not critical, and readability is more important.
- [3 pts] Structure
  - Control flow should be concise and clear:
    - No excessive nested loops.
    - No long switch statements that could be replaced by a simple computation.
  - Use global variables only if you intend to share them between functions (or for debugging purposes).
  - Separate software into modules for each subsystem, e.g. buttons and sampling. A module is a separate .c file, usually with an accompanying .h file.
- [3 pts] Readability
  - Use driver function calls for hardware initialization.
  - Initialize each subsystem (e.g. buttons and sampling) in a separate function.
  - Do not use “magic numbers.” Define named constants or use existing constants.
  - Use clear variable and function names that reflect their functionality.
- [3 pts] Comments
  - Code should be well-commented: explain the high-level functionality.
  - Each function should have an explanation of its arguments, return value and functionality.

## Report Rubric (15 points)

- [2 points] Introduction
    - State objectives of the lab and what you actually accomplished. This section is brief and informs the reader of what they should expect to find in the rest of the document
- [10 points] Discussion and Results
    - Explain the system requirements and real-time goals. Comment on specific things your  implementation did to meet these goals.
        - How were tasks prioritized? What was high , middle, low priority?
    - For each major lab component or step explain your implementation to demonstrate your understanding.
        - What functions did you create? How did they work together to implement the required functionality?
        - How did various parts of the code pass information or communicate?
        - If code was provided or given as an example, how did was it incorporated in the design? What modifications were needed to make it work?
- [3 points] Conclusion
    - List significant problems that occurred and how / if they were solved.
    - What you learned from the lab.
    - What you found valuable / interesting about the lab and how it might be improved in the future.