FFT Display

Tom Faulconer

Due: 10 April 2020

1 Introduction

The goal of this project was to display a 256 point Fast Fourier Transform (FFT) on the ezDSPC5502's on-board LCD screen using Texas Instruments' (TI) Real Time Operating System (RTOS) called DSP BIOS. The data is collected by a hardware interrupt (HWI) and placed into a mailbox. Once the mailbox is full, it "posts" (sends) the audio samples to the designated mailbox slot. The data is then picked up from the mailbox and either processed with my Finite Impulse Response (FIR) filter. The default setting is to let the audio through, unfiltered, or, on a Switch 1 (SW1) press, filtered. Then the data is output with another HWI. This must be in real-time.

The audio samples are also placed into an FFT task that computes the 256 point FFT. Once the FFT is computed the magnitude of the FFT is sent to the LCD task which sends commands over the I^2C bus to fill pixels on the LCD. The button task implements uses the I^2C bus to detect a button press (SW2) and toggles an LED.

2 FFT Performance

The first step in the FFT was to compare the theoretical FFT (computed in Matlab) to the FFT implemented on the microcontroller (ezDSP) from TI.

I used a test vector generated by our professor that was the combination of two sinusoids, a 1000 Hz and a 4000 Hz sine wave, added together, see Figure 1.

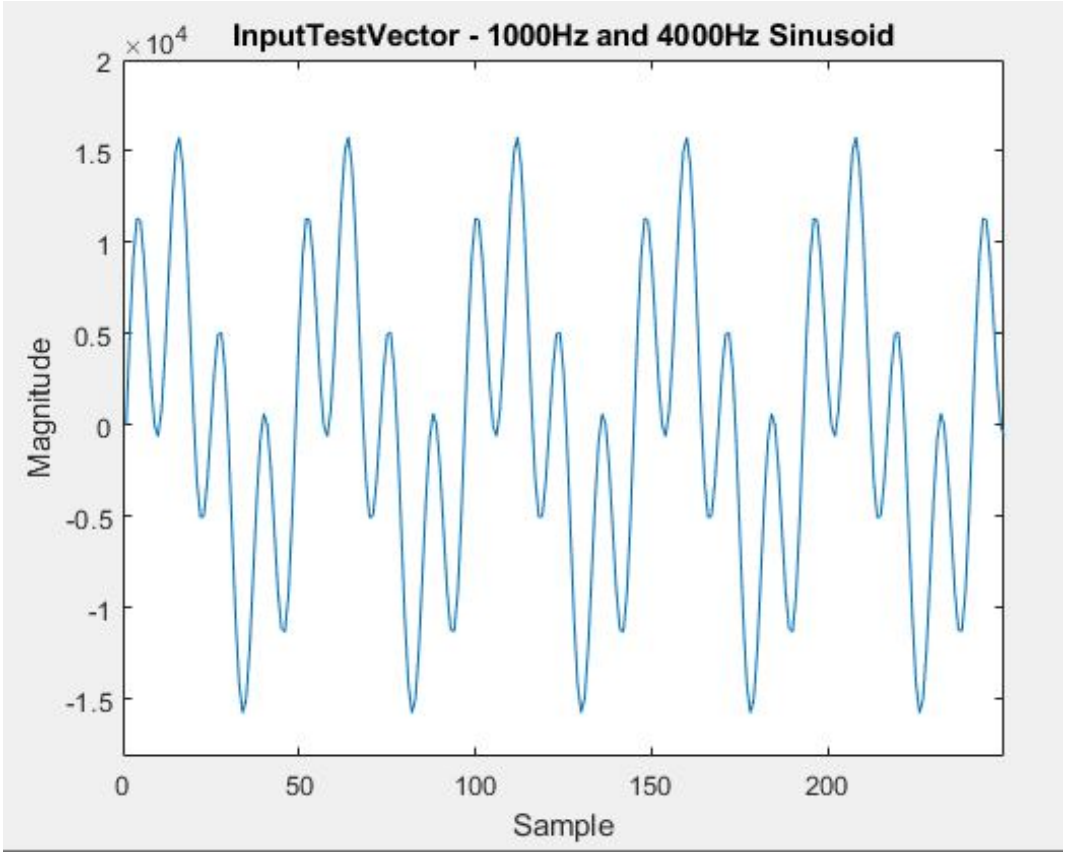

Figure 1: Input Test Vector

Now that the test vector is is place, I'm going to run the FFT using Matlab, this will be considered the theoretical output, there should be a spike at 1000 Hz and a 4000 Hz frequency. Shown in Figures 2 and 3

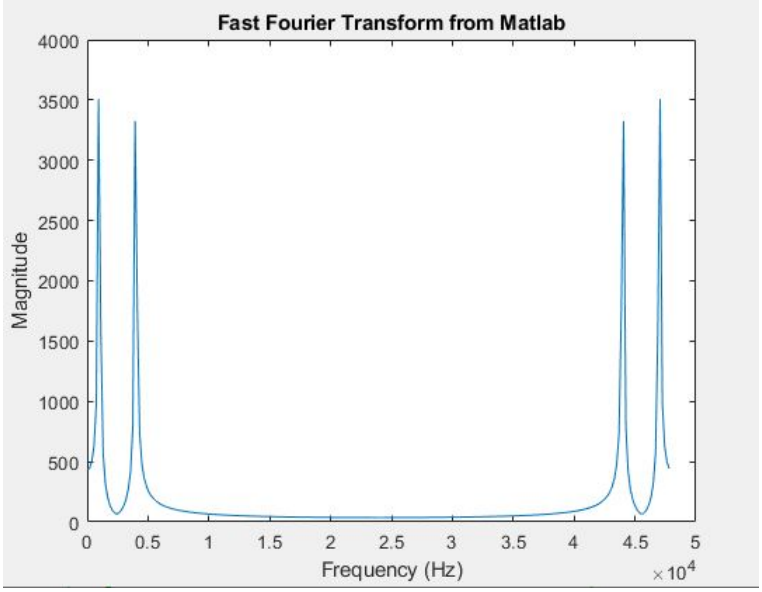

Figure 2: Matlab FFT Output

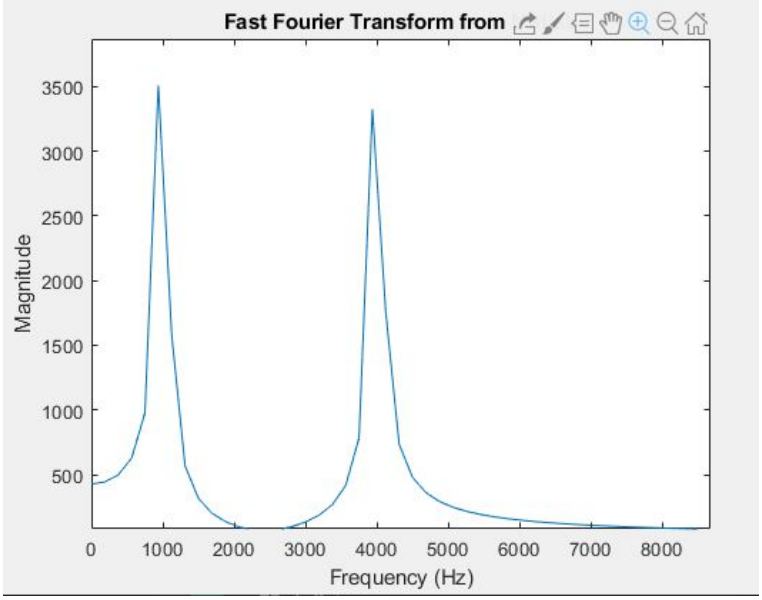

Figure 3: Matlab FFT Output zoomed in

From here, I can run the FFT library from TI to verify that it's output is

correct. I will use the same test vector and grab the output from the ezDSP and compare them to the theoretical ones. Refer to Figure 4 and 5 below

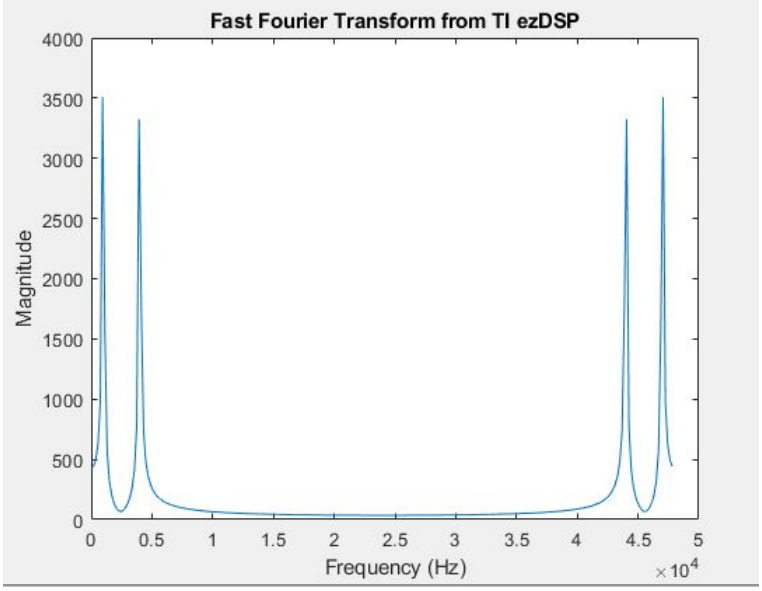

Figure 4: ezDSP FFT Output

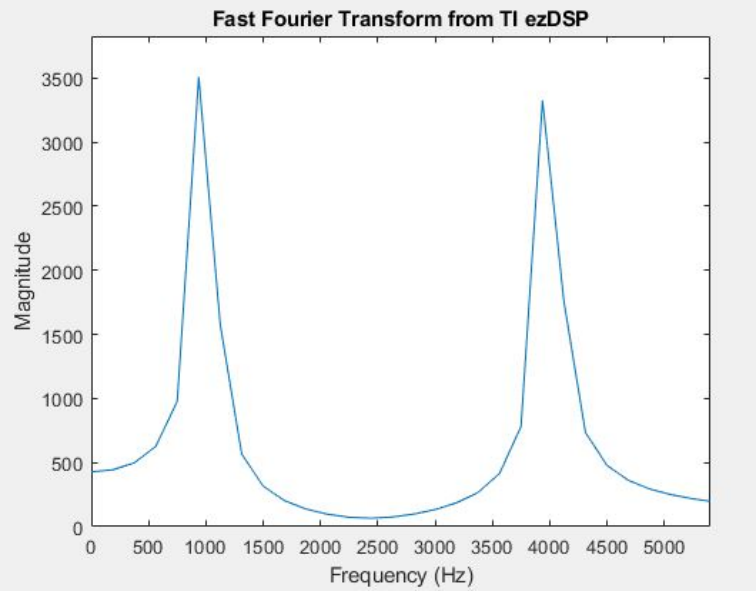

Figure 5: ezDSP FFT Output zoomed in

The output is the same! This confirms the output is working, but let's use a more qualitative approach to compare the two, that is I will use the mean-squared-error (MSE) to compare the outputs. After all, Matlab is using 64-bit floating point values and the ezDSP is using 32-bit fixed point math. The MSE was 0.3176 - the closer the MSE is to 0 the better. The MSE expresses the quality of an estimator, in this case our estimation FFT was very close to the "true" value. Lastly, the FFT library given to us by Texas Instruments is very fast and was calculate to be 21 cycles.

3 LCD Design and Performance

The 256 point FFT produces 128 frequency bins with widths of 187.5 Hz ($F_s/2$). The LCD driver is capable of driving a 128 wide x 64 tall pixel screen, but the on-board LCD is 96 wide x 16 tall, so the frequency bins between 96 and 128 will be dropped. One might assume data would be lost by dropping these frequency bins, but by dropping those bins we're dropping frequency content between 18kHz - 24kHz. The human audio spectrum is roughly 20 Hz - 20 kHz. So there is very little audible frequencies that are actually dropped here.

The final frame rate was recorded at 20 frames per second. The LCD example project sent individual I^2C commands which comes with a lot of overhead when trying to send hundred's of commands. To remedy this, one can modify the "multi-send" command. Instead of sending sending one command at a time, all 193 commands can be sent at once in one multi-send command. This eliminates the overhead of starting and stopping the I^2C bus on each send. Another technique to increase the frame rate would be to increase the I^2C bus speed from 100 kHz (standard mode) to 400 kHz (fast mode).

The next task was to plot the magnitude of the FFT on the LCD screen. The LCD only has a resolution of 16 pixels so it's important to pick a maximum value high enough to be able to have an appropriate granularity to the LCD. The height was built using trial and error. I knew a magnitude well above 60,000 could be displayed, but the resolution of our LCD is only 16 pixels. After listening to multiple songs, I determined that the magnitude never went above approximately 1500, so I chose 1700, adding an extra 200 in the event that the threshold was too low.

4 Structure & Priority

Task priority was an important part of development. The LCD task as the lowest priority, but noticed that it never was allowed to run by the scheduler. To alleviate this problem, the LCD task priority needed to match the FFT task. So the priority hierarchy is as follows:

1. Audio Processing **(Highest Priority)**
2. User Interface (buttons)
3. FFT / LCD **(Lowest Priority)**

Because the FFT and LCD tasks are at the same level, they are placed into a Queue. Whichever task is placed first into the scheduler will always run. For example if the FFT task was placed into the queue first then the LCD task would never run (even though it's at the same priority level. The solution O chose to mitigate this issue was to use the API "TSK_sleep()" which causes the tasks to be blocked for the number of ticks passed in as an argument. I chose 5 ticks (5 ms for our system) to be adequate for both tasks to get time on the CPU.

The LCD and the button tasks both utilize the I^2C bus to send and receive data. To ensure that the bus was used by one task at a time, a semaphore was required, otherwise the button task would read the bus and disrupt the audio every 50 ms (the time I decided to check the button task). The semaphore protected the data being transferred over the I^2C bus so it would not get corrupted, nor be used by another task.

A block diagram of the structure can be seen in Figure 1 below:

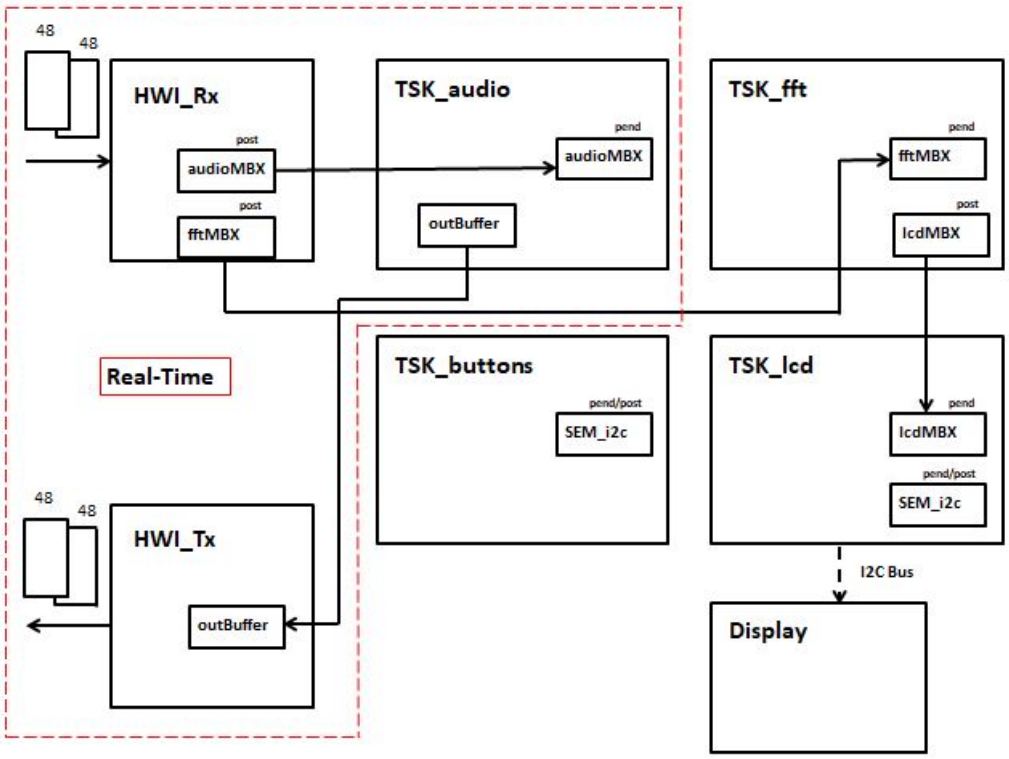

Figure 6: Program Block Diagram

5 Conclusion

This project allowed me to display the frequency content of audio signals over an LCD and utilize semaphores to protect important resources. Additionally task priority was an important concept to understand. Adding a mailbox to pass data to the LCD from the calculated FFT was required for real-time behavior.