Physics 498 – Simple Spectrum Analyzer Using Atmel atmega32

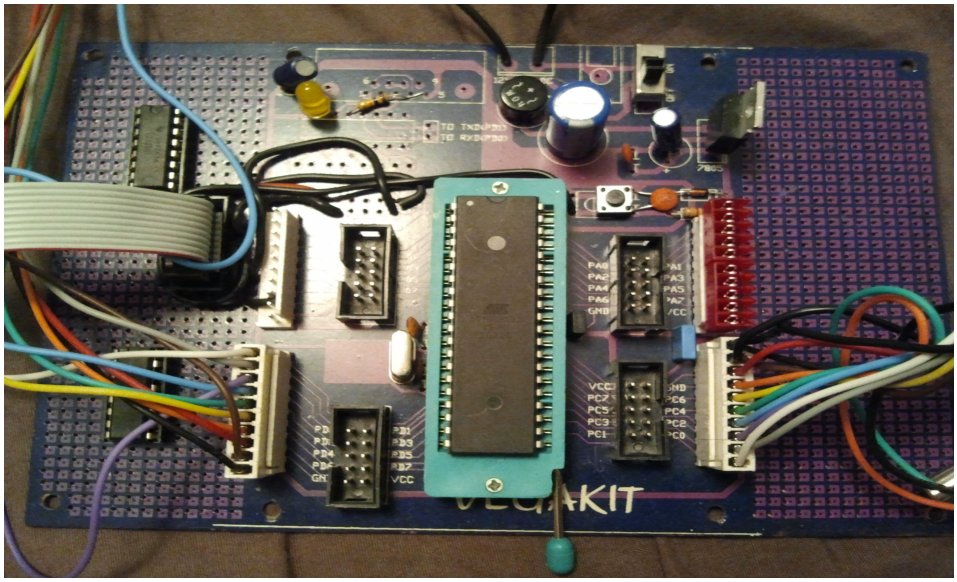
-Bilal Gabula

The initial idea/plan was to build a graphic equalizer using the atmega32 which is a simple micro-controller. As the semester progressed I realized how unrealistic my goals actually were and finally decided to stick to completing a simple spectrum analyzer.

The reason for using Atmega32 in the first place was that I already had the chip and a development board that I had soldered over the winter break, I also had all the software and had tested the chip and board to work and perform basic functions like turn on an LED and respond to button.

The Building Process:

The Micro-controller:



Atmega32 + Development Board

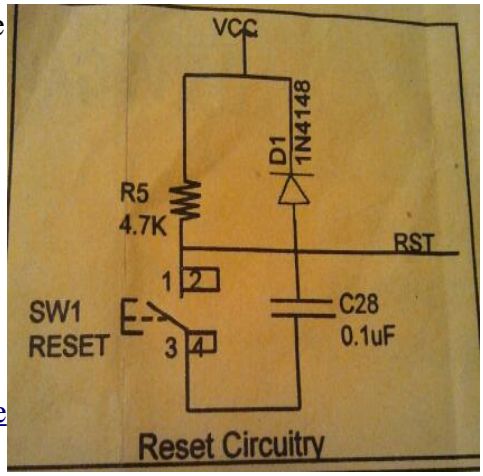
This is the micro-controller on the development board. The IC has 40 pins in total 32 of which are ip/op pins, divided into 4 ports. PORTA, PORTB, PORTC and PORTD. PORTA is on the top right, with the red-female connectors which can be used as a D/A convertor. I use this port to convert the analog input into a digital signal that can be interpreted and used by the micro-controller. PORT D is at the bottom left which can be used for external timers and interrupts but I have used it to connect to the switches and the LED which is on the same board. PORTC is on the bottom right and can be used for JTAG interfacing but I have used it to connect to the LCD screen. PORTB is on the top left and can also be used as analog comparator, and to serial interface and program. I have used it to connect to a ISP (In System Programmer) which is connected to my computer which allows me to program the chip.

Right now the micro-controller is also getting power through the programmer and my computer but it can be hooked up to an external 12V AC or DC source as it has an on-board 7805 along with a rectifier.

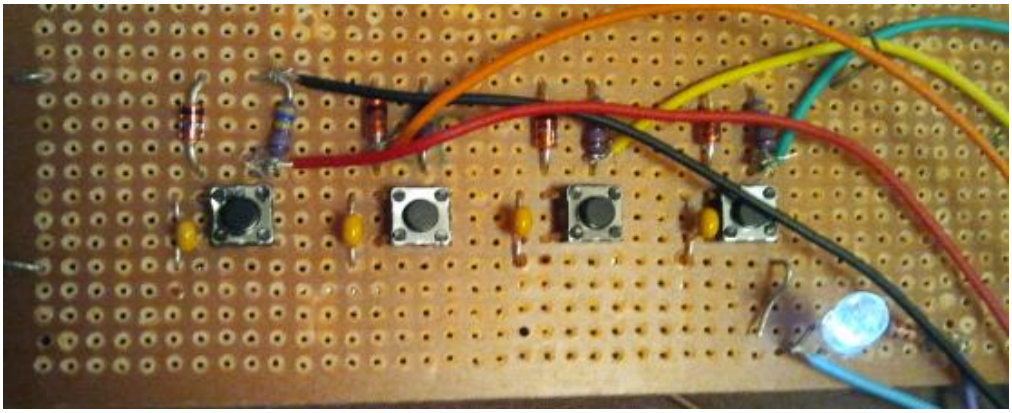
Initial Steps:

I bought four DPDT push button switches and replicated the exact same circuitry that was on the reset switch of the micro-controller already as I knew that worked and provided an almost instantaneous 0V at an otherwise high pin. The point marked reset on the adjacent diagram was where I connected the pin of the micro-controller.

The diode is a fast switching diode to prevent current from flowing in the opposite direction (from the pin to Vcc). The capacitor is to reduce the AC signal coming across due to the Pole hitting the Throw multiple times during one push. Finally the resistor prevents huge amounts of current flow when the switch is pressed. When the switch is pressed the pin (RST) is connected to ground and if the switch is not pressed it is connected to Vcc or 5V in this case. I built this circuit with an existing dot-matrix circuit board that I already had in my possession. I soldered in a LED with a small resistor (to prevent the LED from blowing out) in series to test the circuit and wrote simple code to test the switch circuitry. Here is the code that I wrote to check the switches. Simply put the LED would stay on for a different amount of time for each different switch was pressed. This is a simple header file that I wrote to make it easier to take inputs set outputs.



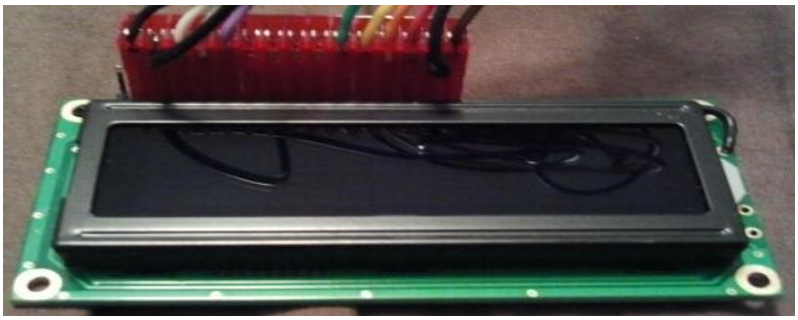
Switch Circuit


Assembled switch circuit and LED

The LCD screen:

I got a small 16x2 LCD screen from spark fun electronics and started working on writing a header file to make the micro-controller initialize and talk to the LCD driver. The reason I wanted to write my own header file was so that I could understand to the most basic level what was going on to write a character to the LCD and optimize it so that it would fit my requirements. After reading the data sheet for the LCD driver (HD44780), writing and debugging code recursively until I was finally fed up with the process which was yielding me almost no results I began an internet search to find a good library that I could use. Though I was unable to get my header file to work I did learn a lot about HD44780 and how any basic LCD screen works. After some searching I found a very good header file here which had most of the functions I needed.

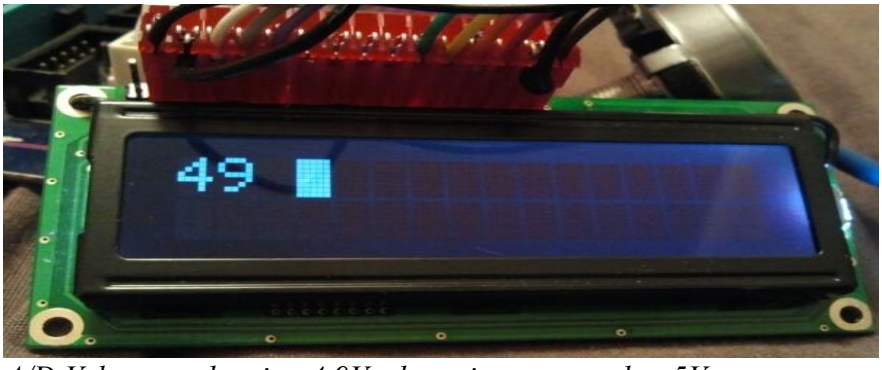
Since I wanted to be able to easily change connections on the LCD screen and have the option of using it with another device I soldered male pins to it and used female connectors to connect wires running from the micro-controller to it.



The LCD screen!

Checking ADC input:

Once I got the LCD screen which took some significant amount of trying and testing, I tested out the A/D convertor on board the atmega32. Using this code I managed to get the micro-controller along with the LCD screen act as a pretty accurate Voltmeter (up to one decimal place).

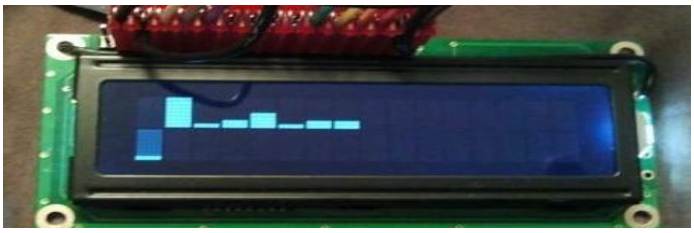
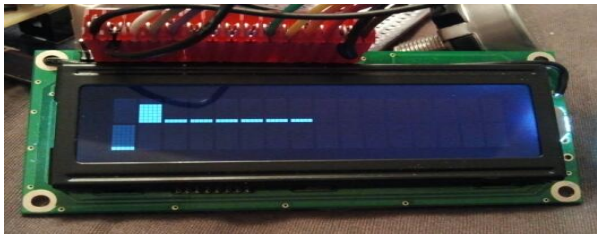


A/D Voltmeter showing 4.9V when pin connected to 5V

The DFT:

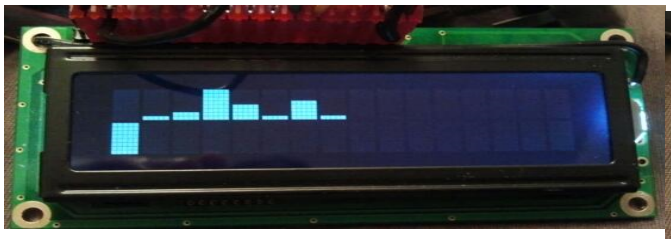
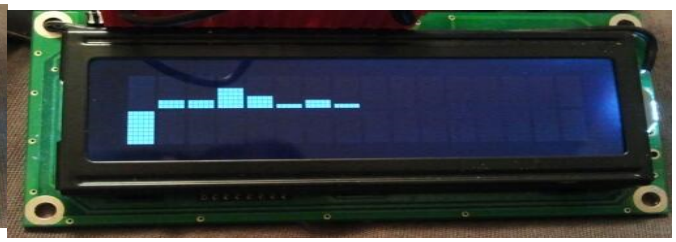
After my bad experience trying to write my own code for the LCD I started with looking for DFT code online and found this site which was very helpful and had code which was very usable after some modifications. I created this header file that I used to calculate the DFT as well as print the spectrum on the screen when the switch is pressed. The inputs are sampled with a delay of 20 micro seconds. This means the frequency of sampling is 50Khz. Therefore the top of the spectrum should be at approximately 25Khz, and the middle should be at 12.5Khz. I simulated signals using audacity (measured the wave file with Spear and it looks pretty clean and doesn't have any harmonics so it should ideally give one peak frequency band somewhere near the middle of the spectrum) and set it up as the input of the micro-controller. The results are below.

For 500Hz at two different instances.

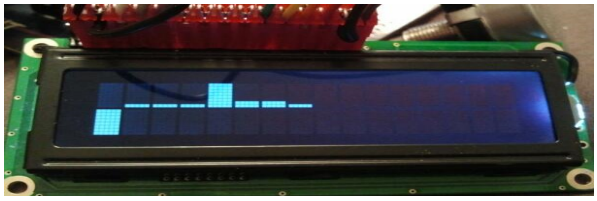



For 7Khz

For 10Khz at two different instances

For 17Khz



My audio card on my laptop cannot create tones 18Khz and above so I was unable to test that. But as I did see in lab with the frequency generator that as the frequency increases the accuracy decreases.
I did use a hamming window on the input signal but since I only have 16 different samples the resolution or error is more due to the lack of enough samples rather than the DC offsets at sides of the window. If I take more samples it takes longer time to compute the DFT and do the math making running in real time impossible.

Problems I encountered:

Parts:

The actual audio D/A chip that Analog devices were nice enough to send me for free were surface mount. I did not have the circuitry or soldering power to use surface mount chips. Though I never really got to the part of actually converting a digital signal to analog.

Time:

The chip runs at a frequency of 11.0592Mhz. So I never thought time would be a big factor as at that speed it would take around 90 nanoseconds to complete any one line of code. But I realized that while computing square roots and squares of a number (while taking absolute value of a complex number) the micro controller would take a significant amount of time.
I was unable to get a good resolution DFT in the forward direction fast enough so as to just get the frequency bands. So there was no way I would be able to take the forward transform, reverse transform , multiply a desired amplification to every band differently and output the audio signal in real time.

Conclusion:

I got the spectrum analyzer to work pretty well but was unable to get it to work in real time due to time it takes the micro-controller to do math. I was able to however get a reasonable spectrum every time I want by clicking a button. The hamming window really helped get a better spectrum. Before applying the hamming window to the input signal the frequency bands were very random even though I had it hooked up to a frequency generator. Even though it is not accurate every time, on an average it does give the correct bin.

This whole process would have been easier, faster and probably much more successful if I had used a DSP chip with inbuilt functions for DFTs and which can probably do math a lot quicker than a standard all-purpose micro-controller. But if I had followed that approach I would have never learned about LCD screens and its operation.