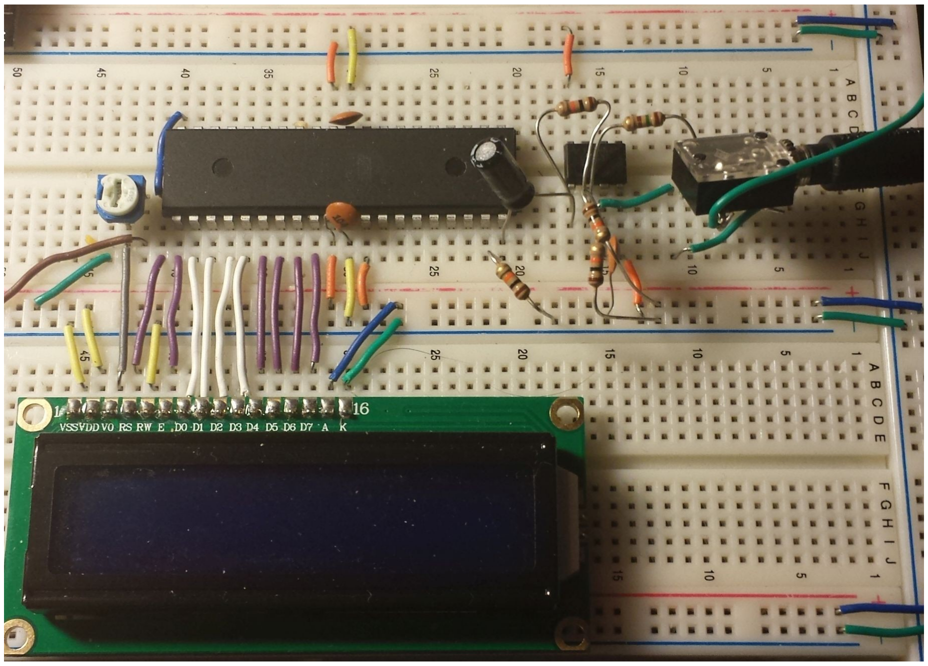
Matthew Lumantas

861166392 - mluma001@ucr.edu

EE120B - 022

Custom Lab Project - Frequency Visualizer

Wiring Setup



Picture 1: Breadboard Setup

Seen in Picture 1 is the ATMega1284 in the upper left, the LCD at the bottom left, and the amplification circuit along with the 3.5 jack at the upper right.

Component Explanation

The heart of the project is the Fast Fourier Transformation (FFT) algorithm. The FFT takes the Discrete Fourier Transformation (DFT) and speeds up runtime from N^2 to Nlog, making computation of the DFT practical. This project's implementation of the FFT is based off the Cooley-Tukey algorithm, using psudocode off wikipedia. It is a radix-2, decimation-in-time, out-of-place, depth-first, explicitly recursive implementation. The code uses a struct representing a real and imaginary component for each value in the transform.

The DFT of the entire array is split into its even and odd elements. If the resulting split array is of size 1, the DFT is trivial: the frequency representation is exactly the same as the time domain. This forms the base case of the recursion. Otherwise, the code recursively splits the split array into further even and odd element arrays until the entire array is transformed. It then combines each even-odd pair of DFTs into a single DFT. This is accomplished by taking the value of the even element and adding or subtracting the odd element muliplied by a so-called "twiddle factor". This combination operation is called a butterfly.

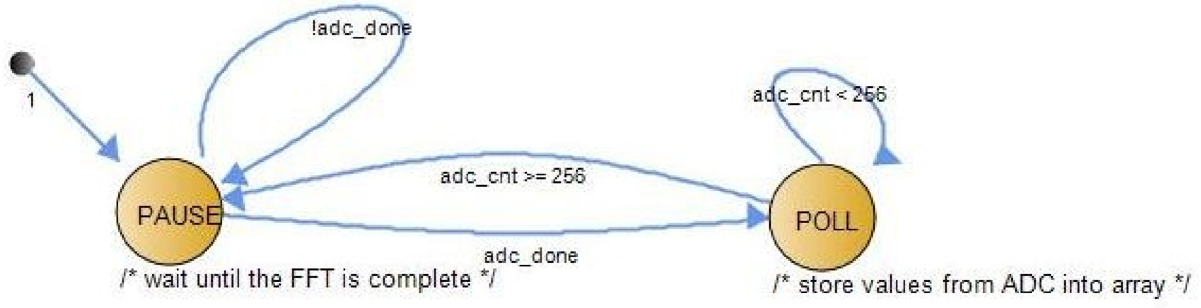
The twiddle factors are computed by using Euler's Formula to split the exponential into a real cosine and imaginary sine value. These values are computed using a form of a lookup table. An array of char values with -128 representing $\sin(-\pi / 2) = -1$ and 127 representing $\sin(\pi / 2) = 1$ is used to quickly compute sine and cosine values. The frequency-domain output of the algorithm is then converted to the absolute value of the real and imaginary components. It is then averaged into 16 different bands to be output to the LCD.

The LCD output consists of representing each of the 16 frequency bands using the 16x2 character display. By default, the LCD does not contain character data for increasing lenth vertical retangles representing a bar filling vertically. It does, however, contain functionality for the user to save eight different characters into the character memory. This functionally is used to combine each column of two 5x8 pixel characters into a bar with values of range 0-16. This, for the purposes of this project, emulates the possible band outputs of a 16x16 LED matrix while using a lot less overhead in terms of wiring, output pins, and need for the ATMega to produce and sink current.

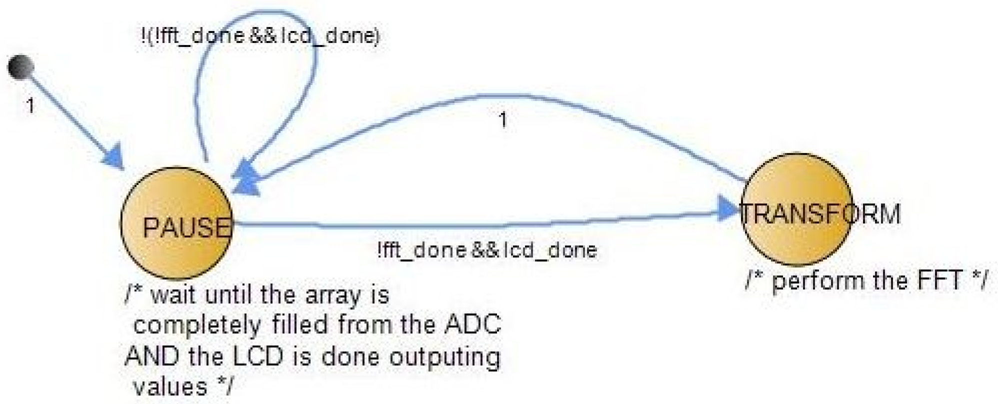
The amplification circuit is a standard inverted multiplier circuit using a voltage divider to bias the ground to $V_{cc} / 2$ to allow single supply operation: i.e. V_+ and V_- to the op-amp are 5V and 0V, with "ground" at 2.5V. Using a 130 and 10k ohm resistor in the amplifier creates a gain of about 77. This allows the ADC to more easily see the waveform of the audio input. This circuit, however, acts as a high pass filter with a cutoff frequency of around 300 Hz due to the final decoupling capacitor and the pull-down resistor at A0.

The audio jack is a standard 3.5 tip-ring-sleeve connector that can be used with almost any analog playback device. The sleeve is grounded relative to the amplifier at 2.5V, the sleeve is left floating, and the tip is connected to the input of the amplifier. The sleeve is left floating due to most audio player's stereo output. The ring is either the left or right channel with the sleeve being the opposite channel. To simplify the analog circuit I decided to only use one channel instead of mixing the stereo inputs.

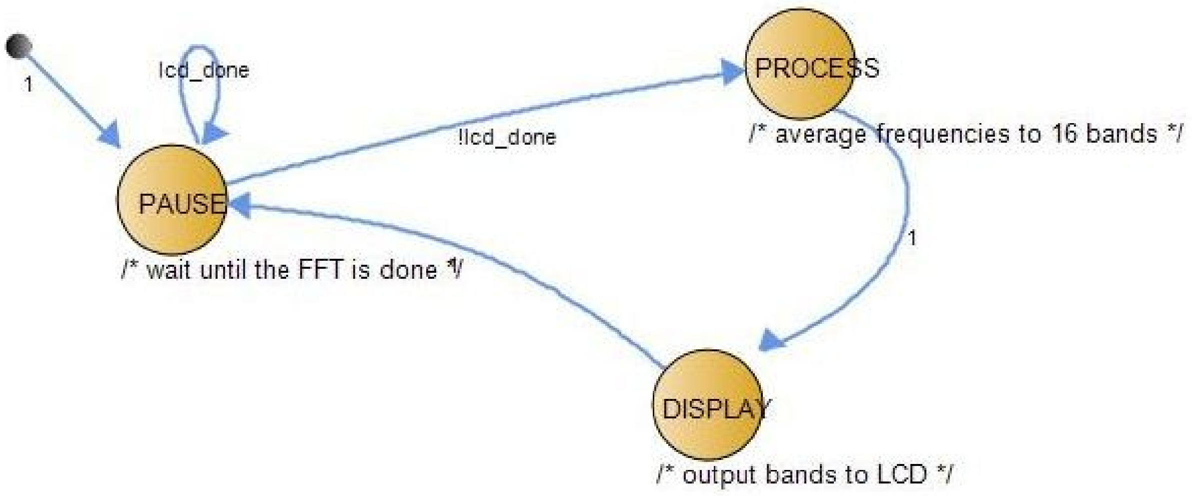
State Machines


!adc_done
1
adc_cnt < 256
adc_cnt >= 256
PAUSE
POLL
adc_done
/* wait until the FFT is complete */
/* store values from ADC into array */

SM 1: ADC Polling, period 0.1 ms


!(!fft_done && lcd_done)
1
1
PAUSE
TRANSFORM
!fft_done && lcd_done
/* wait until the array is completely filled from the ADC AND the LCD is done outputing values */
/* perform the FFT */

SM 2: Fast Fourier Transform, period 5 ms



1
lcd_done
!lcd_done
PROCESS
/* average frequencies to 16 bands */
PAUSE
/* wait until the FFT is done */
1
DISPLAY
/* output bands to LCD */

SM 3: Frequency Processing and LCD Output, period 5 ms

The state machines used in this project are fairly simple. The only thing of note is the use of flags to denote whether the different state machines are currently busy or free. Because of this, the period of the state machines are not very relevant, as most of the actions are performed based on flags set or clears by the state machines, instead of a base period. This will be touched further upon when discussing the bugs and issues.

Bugs and Known Issues.

The output of the LCD does not very accurately display the frequencies of the input signal. The FFT algorithm, however, seems to be reasonably accurate for the computational and memory shortcuts used. It seems that the free-running nature of the state machines due to the use of flags to determine if the state machines are ready is causing the error. The ADC polling is not specifically periodic, since it only polls when ready and not at regular intervals. When using test arrays, however, the frequencies output by the algorithm are acceptable. There is also a significant amount of error in the analog amplifier circuit, as the input signal is filters and attenuated to a noticeable amount.

Video Link: https://www.youtube.com/watch?v=QozpGKtpRC8

Instructions for use: Plug AUX cable into the 3.5mm jack on the project. Output will then be shown on the LCD.

Resources used: Psudocode for Cooley-Tukey FFT from Wikipedia