ACOUSTIC SPECTRAL ANALYZER

Gayatri Mestry

Final Project Report
ECEN 5613 Embedded System Design
December 7, 2015

Table of Contents

1. INTRODUCTION

Being bought up in a family where learning music is an important tradition, I grew up learning the piano and the violin. The melodious sounds of these instruments are because they are generated at different frequencies is what I learned in my Physics class back in school.

Today, merging my love for embedded and music, the project acoustic spectral analyzer is my attempt to present what the principle behind the electronic tuning process is, how frequencies are plotted and the underlying electronics that go into representing them.

1.1 Statement of Purpose

To demonstrate the details and working of an acoustic spectral analyzer.

1.2 Project Overview

The project aims at creating a simple spectral analyzer that recognizes frequencies in a particular range. The main concept behind the implementation of the project is the Fast Fourier Transform (FFT). It is a special case of Discrete Fourier transform (DFT) which translates the samples taken in the time domain to the frequency domain. Since FFT is much faster and less complex than computing DFT, the Fast Fourier transform is used.

The design of the analyzer incorporates the use of ATMega32A MCU as for the real-time implementation of the analyzer a faster controller than AT89C51RC2 was required. As shown in Figure 1.1 – Basic implementation, the audio input for the analyzer is taken from the basic Electret condenser microphone that is then given to the MCU which performs the FFT on the samples. The graphical representation of the FFT output can be observed on the graphical LCD. The frequency bins of the FFT output is plotted on the Y-axis of the graphical LCD while the magnitude of the FFT samples is plotted on the X axis of the LCD.

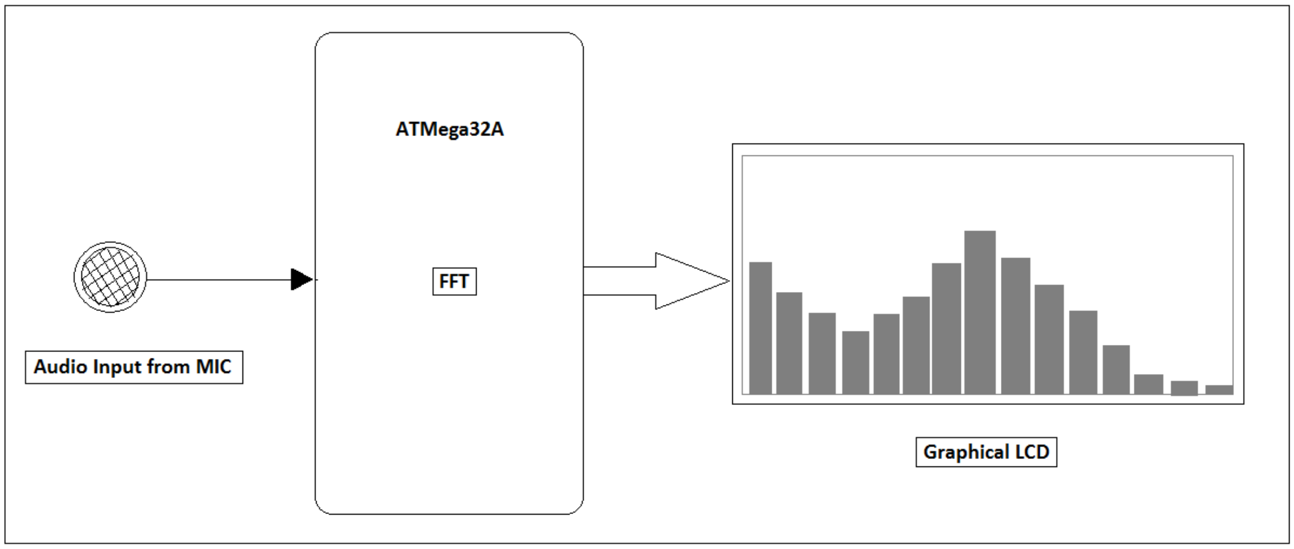


Fig 1.1 Basic Implementation

2. TECHNICAL DESCRIPTION

This section describes the detailed implementation of each individual elements that have gone into the making of spectral analyzer. The section is divided as Design Overview which describes the block diagram in detail, MCU description gives details on how to configure the ATMega for use in the spectrum analyzer, MIC Circuit describes the interfacing of the electret microphone with the MCU. The Graphical LCD interface provides information on configuring the GLCD for the analyzer. The next sections describe the firmware and the software design process.

2.1 Design Overview

- The board for spectral analyzer is powered at 5V using the LM7805 voltage regulator.
- It has ATMega32A as a potential MCU for the design.
- The MCU runs at frequency of 16MHz External Crystal.
- The Reset pin on the ATMega32A is active low, and setting it to low externally will thus result in a reset of the MCU.
- The AVR MCU is programmed using In System Programming interface.
- The audio input is taken in through the Mic circuit and pre-amplification of the audio is done in this step.
- This amplifies audio signal is then given to the ADC pin of the MCU. The ADC samples are then passed into the FFT algorithm. The FFT computes the magnitude of the samples in reference to the frequency range set by the sampling frequency with which the ADC samples are sampled.
- These magnitude samples can then be observed through the UART using the Terminal
- The magnitude samples can also be plotted in real time on the Graphical LCD.

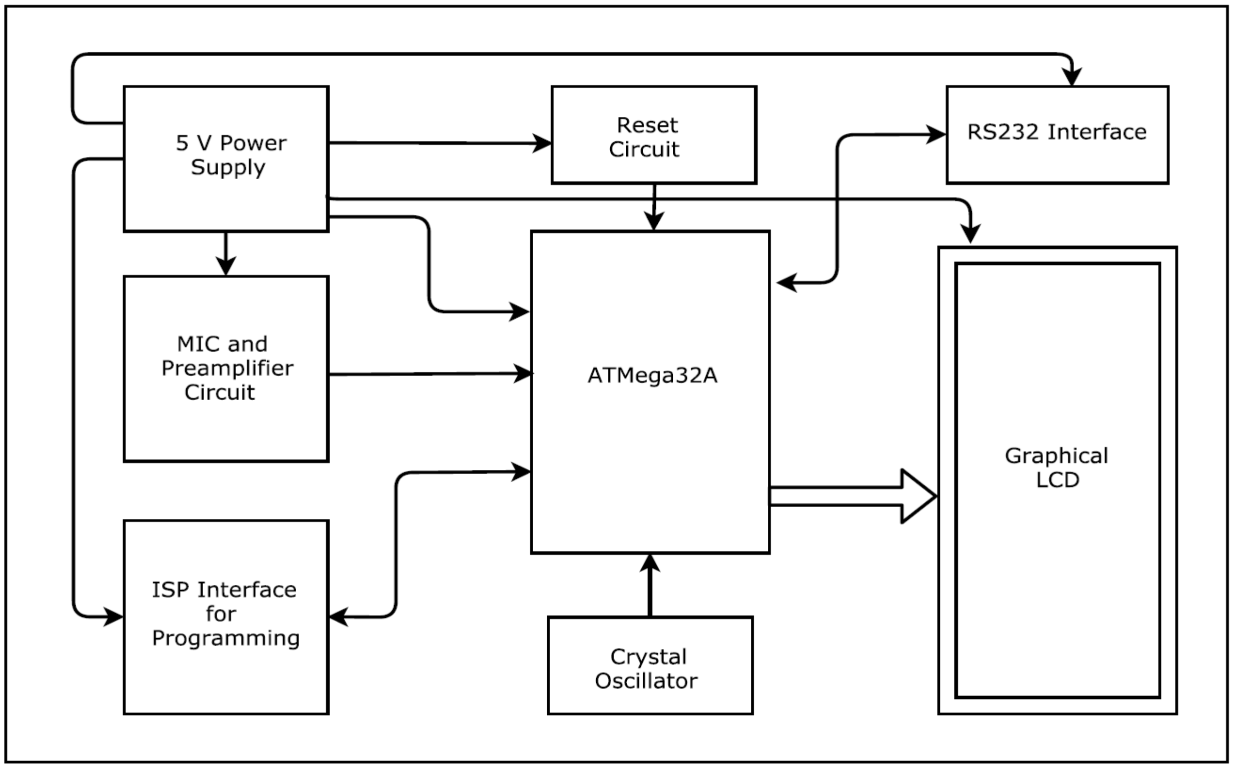


Fig 2.1 Block Diagram

2.2 MCU Description

The MCU used for spectral analyzer is ATMega32A. It is an 8-bit microcontroller having RISC architecture with most instructions being executed in a single cycle. This renders it a higher speed compared to AT89C51RC2 used in the lab. The MCU is configured to work at 5V power supply. It runs at 16MHz oscillator frequency at which the maximum throughput of 16MIPS is obtained for the controller.

2.2.1 Reset Circuit for AVR

The reset of ATMega32A is active low. Though the reset line has an internal pull up resistor, rest can occur sporadically. To prevent this a proper reset circuit needs to be devised. The details of which are:

1. Using a 10K pull up resistor to Vcc
2. A diode IN4148 in parallel with the circuit in step1 connected between Vcc and the reset line to clamp the voltage on reset to around Vcc voltage.
3. A 100ohms resistor between the reset line and the reset push button to prevent a surge of current if the capacitor unloads.
4. The push button connected between the 100 ohms resistor and ground.
5. A capacitor of 10uF in parallel to the circuit described in steps 3 and 4 connected in between the reset line and the ground. It removes the noise on the wire which resets the 44ATMega32A

2.2.2 Programming the AVR

The ATMega32A is programmed using the In System Programming (ISP) interface for programming Flash and fuse bits. The ATMega32A MCU uses the SPI lines to configure the ISP interface. The AVRISP mkll programmer was used to program the MCU. The in-system programming was used with Atmel AVR Studio 7, the current version as available on Atmel website. Figure 2.2 shows the pins of 6 pin ISP header used with mkll programmer.

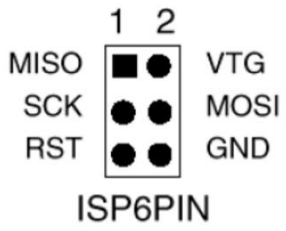


Fig 2.2 ISP 6 Pin header[12]

The pins of ISP header:

1. MISO : This is connected to the MISO pin of the AVR MCU and acts as a general purpose input output pin.
2. VTG : This the positive voltage pin of the AVR and is connected to the VCC power supply of 5V. The supply voltage can be on the range of 2.7V - 5.5V
3. SCK : The AVR uses the clock signal generated from ISP programmer. The correct speed is automatically generated when configured using the AVR studio. The default clock of 125kHz for ISP was selected. The ISP clock frequency must be lower than one fourth of frequency the device is operating on.
4. MOSI : This pin is connected to the MOSI pin of the AVR MCU and acts as a general purpose input output pin.

5. RST : To enter programming mode, ISP pulls the Reset line low. The external pull-up resistor on Reset pin should not be strong that it holds the reset pin high.
6. GND : This is connected to the ground of the target supply.

2.2.3 Setting the FUSE bits

The fuse bits are part of a separate memory in the AVR MCU. ATMega32A has two fuse bytes the high fuse byte and the low fuse byte. The default value of the fuse bytes are 0x99E1. A 0 in the fuse bit means programmed and a 1 in the fuse bit means not programmed. The fuse setting used for programming MCU through ISP interface is 0xC9FF. The figure 2.3 shows high and the low fuse bytes of ATMega32A.

OCDEN	JTAGEN	SPIEN	CKOPT	EESAVE	BOOTSZ1	BOOTSZ0	BOOTRST
BODLEVEL	BODEN	SUT1	SUT0	CKSEL3	CKSEL2	CKSEL1	CKSEL0

Fig 2.3 High and Low byte of AVR Fuse Bits

Higher Fuse Byte:

7. OCDEN : To enable or disable on-chip debugging.
6. JTAGEN : Disable it if the JTAG interface is not being used.
5. SPIEN : To use serial programming of ATmega32A.
4. CKOPT : To set the output swing of the oscillator frequency to full swing or smaller swing.
3. EESAVE : To save EEPROM from erasing during chip erase along with flash.
2. BOOTSZ0 | BOOTSZ1 : To set the bootloader size.
0. BOOTRST : To select if the device needs to be running from the bootloader block.

Lower Fuse byte:

7. BODLEVEL : To set the trigger level for the BOD. Here it is programmed so that the trigger level is 4V
6. BODEN : TO enable the Brown-out Detection circuit for monitoring the VCC level during operation.
5. SUT1 | SUT0 : This sets the setup time to delay the period from the time when the external reset is released until the internal reset is released.
3. CKSEL [3-0] : To select different clock options available. Here the value of 0xF is selected to set the frequency of crystal oscillator.

2.2.4 Analog- to- Digital Conversion

The analog audio signal is obtained from the audio preamplifier circuit is given to the ATMega32A ADC. The ADC converts the input voltage to a 10-bit value. In this project a capacitor is connected between AREF pin and the Ground. Also a capacitor is connected between AVcc and ground. ADC7 pin is taken for the analog to digital conversion purposes.

2.2.5 Universal Synchronous Asynchronous Receiver Transmitter (USART)

The ATMega32A MCU USART is used in synchronous mode for communicating to the terminal to display the FFT samples.

Transmit and receive signals from the ATMega32A MCU are connected to the T1in and R1out signals of the MAX232 IC respectively. The T1out and the R1in signals of MAX232 are connected to the TXD and the RXD pins of the connector respectively.

2.3 Microphone Circuit description

The microphone circuit acts as the main processing block for the audio input before it is given to the ADC pin for analog to digital conversion. The preamplifier circuit is biased at 2.5V for zero input. The main components of the circuit are the electret condenser microphone, the LM358P Operational amplifier, and the biasing circuit.

2.3.1 Electret Condenser microphone

The microphone used in the project is the AOM-4544P-R electret condenser microphone. It is an omni directional microphone with minimum and maximum operating voltage as 3V and 10V dc respectively. The signal to noise ratio is 60dB. The frequency response of the microphone varies from 50Hz to 16,000 Hz.

The electret microphone contains an internal Field Effect Transistor. Also it needs resistor to limit current to FET and the output is obtained from the region where the resistor is connected to the FET.

2.3.2 Circuit elements:

- The AOM-4544P-R electret condenser microphone.
- A pull up resistor of 1K ohms
- Capacitor of 1uF to block the DC components of input signal.
- The op-amp LM358P with a gain of 100 is used in inverting mode.
- The circuit is biased to get output voltage of 2.5V
- Decoupling capacitor to op-amp added to improve the stability and prevent oscillations
- Capacitor added as a current bank for output which drains when sudden surges of current occur.

2.4 Graphical LCD

The Graphical LCD used in this project is the Sparkfun Graphic LCD 128x64 STN LED Backlight. It is a dot matrix liquid crystal graphic display and has a KS0108B parallel interface chipset. The typical supply voltage required is 5V. The KS0108B LCD driver has the internal display RAM for storing the display data that is transferred to generate driving signals to represent the stored data.

The Graphical LCD is used to plot the FFT samples buy displaying it as bar graph of 8 columns each. At a time 14 such FFT samples can be represented on the graphical LCD. The bar graph are incremented one line of a page at a time according to the FFT values.

Interfacing the Graphical LCD :
The LCD is connected to the ATMega32A in the way described in table 2.1

Pin no.	Pin Name	Description	Connected to
1	Vdd	Supply voltage	Vcc
2	Vss	Ground	GND
3	Vo	Operating Voltage	Potentiometer
4-11	DB0-DB7	Data byte	PC0-PC7
12	CS2	Chip select 2	PD2
13	CS1	Chip select 1	PD3
14	/RES	Reset Signal	PD4
15	R/W	Read Write	PD5
16	DI	Data/Instruction	PD6
17	E	Enable	PD7
18	Vee	Operating Voltage	Potentiometer
19	A	Backlight	Vcc
20	K	Backlight	GND

Table 2.1 : Connections of GLCD

Also the GLCD is connected to a decoupling capacitor and a potentiometer. The wiper of the potentiometer is connected to Vo pin of the GLCD. The Vee pin of the GLCD is connected to one of the terminals of the potentiometer and the third terminal of the potentiometer is connected to ground.

2.5 Firmware Design

The firmware design of the spectral analyzer comprises of the driver for ADC, USART and GLCD.

2.5.1 Driver for Analog to Digital Converter

For the ADC to sample the input signal correctly it should be configured in the following way

- The ADC pre-scalar is set to 125 kHz that is the ADC will sample the input signal at this frequency.
- ADC reference voltage is selected as AVcc
- The ADC pin 7 is selected for conversion of audio signal
- The ADC register is left shifted so that only 8 bit value of the ADCH is used for comparison since this is faster.
- The ADC is configured in free running mode so that it constantly samples and updates the ADC values.
- The ADC register is polled to check if the conversion value has arrived.
- Also the ADC samples are collected at a sampling frequency of 8000Hz.

To configure the above settings the ADC Control and Status Register A (ADCSRA) and the AD Multiplexer Selection Register (ADMUX) in MCU are used.

2.5.2 Driver for USART:

The USART is configured as:

- The baud rate is set to 9600 using the equation to calculate the UBRR value according to the formula:

$$UBRR = \frac{Fosc}{(16 * BAUD)} -- 1$$

- This value of UBRR is entered in the high and low bytes of the UBRR registers.
- The USART Control and Status register A (UCSRA) is set to the vale 0x00 as it deals with all the error conditions and USART transmit and receive complete conditions.

- In the USART Control and Status register B (UCSRB) transmit and receive complete interrupt are enabled to generate the transmit and receive interrupts respectively.
- The USART Control and Status register C (UCSRC) sets the communication mode by setting the mode as Asynchronous operation, keeping parity mode disabled with selecting 1 stop bit and selecting the character size as 8 bits.
- For transmitting a byte on UART the USART Data Register Empty (UDRE) bit is polled in USCRA register to know if the transmit buffer(UDR) is ready to receive data.
- While receiving a byte the USART Receive Complete bit is polled in USCRA register to check if there is any unread data in the receive buffer and thus this buffer is then read to get the data byte.

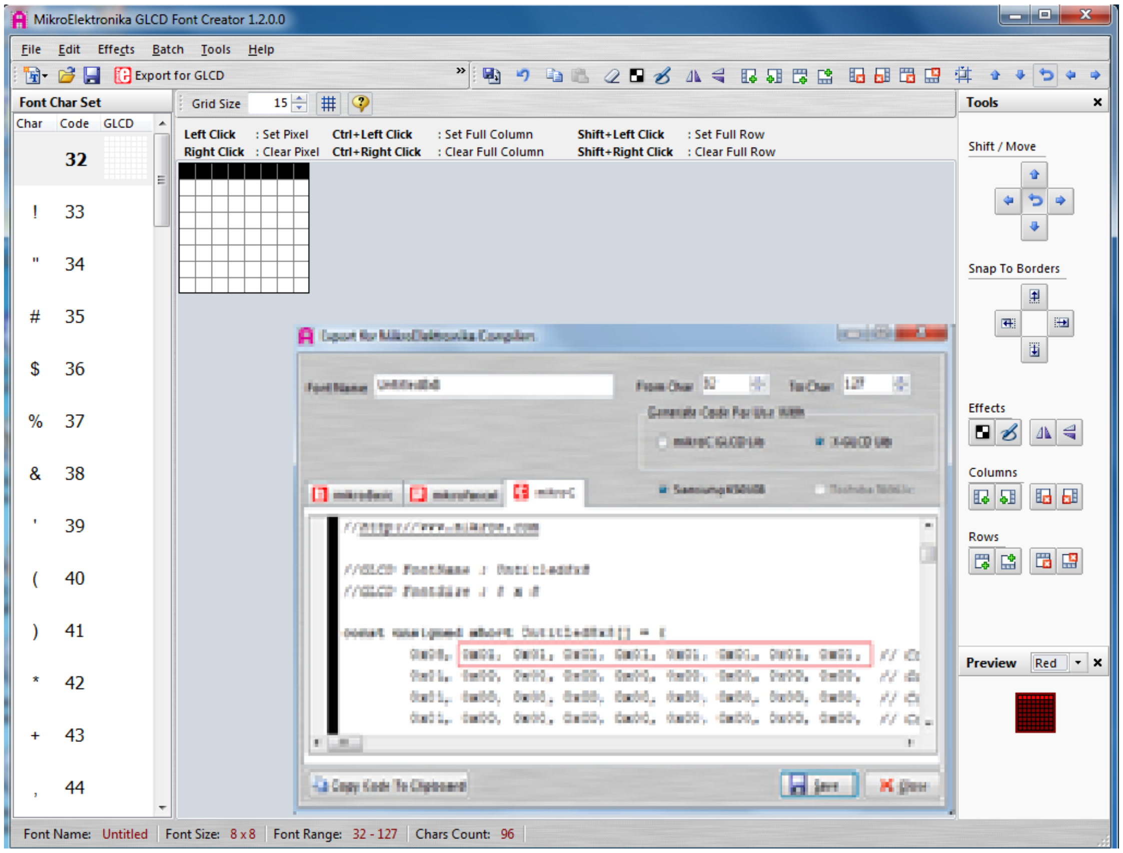
2.5.3 Driver for GLCD

The GLCD Driver has the following functionalities:

1. GLCD Initialization
   - For initializing the GLCD the following commands need to be used
     0xC0 - Setting the display data line at the top of the screen
     0xB8 - Setting the X address to page 0
     0x40 - Setting the Y address to 0
     0x3F – Turning the display on
2. Sending Command on GLCD
   To send command on the GLCD the DI and RW line has to be pulled low as well as the enable signal needs to be pulled high and then low after a delay of minimum 450ns which is the Twl time that is E low level width
3. Sending Data
   The functionality to send data is similar to sending a command except that the DI line is pulled high during this operation.
4. Selecting Page
   This functions enables a particular chip of 64 column display and disables the other using the chip select lines.
5. Clear Display
   The GLCD display is cleared page by page by setting data as 0x00 for each column in a page.
6. Go to X-Y location
   This functions sets the X and Y address to go to a particular pixel location on the GLCD screen. X and Y location of the pixel are taken as parameters. The value of X is compared to 64 to check if the pixel falls on the chip1 or chip2 and accordingly the command to set X axis data of the selected pixel is executed.

```
void GOTO_XY(unsigned int x,unsigned int y)
{
      unsigned short Col_Data;
      GLCD_CTRL_PORT &= ~(DI);
      GLCD_CTRL_PORT &= ~(RW);
      if(y<64) //left section of GLCD
      {
            Col_Data = y;
            Select_page(1);
            GLCD_Comd(0xB8 | x); //Display part of image to Page0
            GLCD_Comd(DISPLAY_SET_Y|Col_Data);
      }
      else
      {
            Col_Data = y-64;    //put column address on data port
            Select_page(0);     //Display part of image to Page1
            GLCD_Comd(0xB8 | x);
            GLCD_Comd(DISPLAY_SET_Y|Col_Data)}}
```

7. GLCD Draw line functionality
   This function helps create a bar graph by incrementing a line of pixels in 8 columns of a page. This helps in adding a new line according to the values of FFT. The values in this function are obtained by using the software GLCD Font Creator.



```
//http://www.mikroe.com

//GLCD FontName : Untitled8x8
//GLCD FontSize : 8 x 8

const unsigned short Untitled8x8[] = {
    [illegible]
```

Font Name: Untitled | Font Size: 8 x 8 | Font Range: 32 - 127 | Chars Count: 96

Fig 2.4 Font Creator to know the pixel value

```
  void GLCD_Draw_line(unsigned short x,unsigned short y, unsigned short j)
  {
  unsigned short Col_Data=0x00,i;
  for(i=0;i<8;i++)
  {
          switch(j)
          {
                  case 0: Col_Data = 0x01; break;
                  case 1: Col_Data = 0x03; break;
                  case 2: Col_Data = 0x07; break;
                  case 3: Col_Data = 0x0F; break;
                  case 4: Col_Data = 0x1F; break;
                  case 5: Col_Data = 0x3F; break;
                  case 6: Col_Data = 0x7F; break;
                  case 7: Col_Data = 0xFF; break;
          }

          GOTO_XY(x,(y));
          GLCD_Data(Col_Data);
  y--;
  }
}
```

8. GLCD Bar Graph
   This function helps to represent normalized FFT values using the GLCD Draw Line functionality

9. GLCD_Spectrum
   This function helps to get the FFT spectrum for all frequencies that can be plotted on the GLCD Y axis with the bar graph showing the amplitudes of the input signal on the Y axis. A minimum amount of delay had to added after displaying the bar graphs for all the frequencies on the GLCD so that the bar graphs for one sample can be adequately visible.

2.6 Software Design

The heart of the software design of the spectral analyzer is the Fast Fourier Transform. It is a faster version of the Discrete Fourier Transform. In this project we are using the 32 point FFT analysis of the input signal sampled at sampling frequency of 8000Hz. According to the Nyquist criteria the result of the FFT analysis will be in the frequency range of 0-4000Hz with a frequency resolution of 250Hz. The FFT process in the code can be seen in detail as follows.

Any signal, irrespective of its complexity, can be represented by a sum of individually mixed sine and cosine functions. This representation is achieved by a Fourier transform. If the sampling rate is small more sine waves of varying frequencies are needed to represent the signal. The number of sine waves required to represent the signal is equal to the number of samples taken. The lowest and the highest frequency sine waves required to reconstruct the input signal need to be considered. Along with the frequency the amplitude of the sine wave is important. Generally low frequency sine waves in a particular range of frequency have higher amplitude than high frequency sine waves. To compute the Fourier analysis the amplitude of the individual portion of the signal is necessary. A close approximation of the amplitude of the sine waves can be obtained by comparing the frequency of the sine waves and the frequency of the input signal. If the frequency of the sine wave matches with the frequency of the input signal the signal is considered to have the same magnitude as the sine wave else the frequency is considered to be absent.

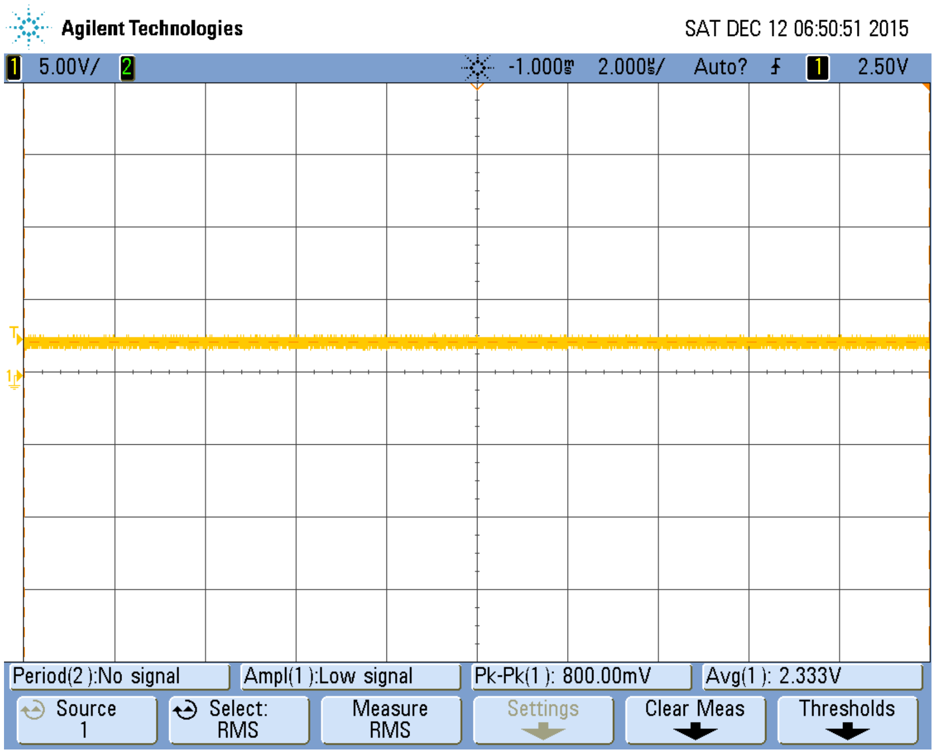
The sine waves of known frequencies and unit amplitude are multiplied with the input signal and their result is added together to get the amplitude of the sine wave at the frequency of the input signal. The frequency range in which these sine waves are obtained are known as ‘bins’. This entire operation is representative of working of a filter. Thus the functionality of band pass filters centered on the frequencies of the input signal is obtained by Fourier transform. Depending on whether the resulting signal is similar to a sine or a cosine wave, the resulting value of the signal will have a larger or smaller amplitude. Both sine and cosine signals are used to introduce phase and make it more efficient. Since this method of operation of Discrete Fourier transform is independent of the nature of input waves, it takes long time to compute the result of the transform. On the other hand the FFT computes the result by spreading the energy of the input signal over nearby frequencies. The FFT also requires the length of transform to be a power of 2. The transform is symmetric in nature and thus only the positive half of the result is taken into consideration with one additional bin with highest frequency to compensate for the symmetric nature of the transform.

Since 32 point FFT is used in this project, the first 16 values are important as the other values are symmetric. In this project due to the visual limitation of the graphical LCD, one to fourteenth frequency bins are displayed on it. The zeroth and the fifteenth frequency bin is not considered. So the range of frequencies that can be plotted on the GLCD is 250Hz to 3750Hz with a frequency resolution of 250Hz.

2.7 Testing Process

2.7.1 Microphone circuit testing

After the Power and the reset circuitry were tested for their functionality, the microphone and preamplifier circuit was added. The output voltage at the preamplifier circuit at zero input was observed to be 2.48V on Multimeter. Different frequencies and types of signal were given as audio input to the microphone and the signal output of preamplifier circuit was tested through oscilloscope.



Period(2):No signal
Ampl(1):Low signal
Pk-Pk(1): 800.00mV
Avg(1): 2.333V

Fig 2.4 Zero input voltage at the preamplifier circuit

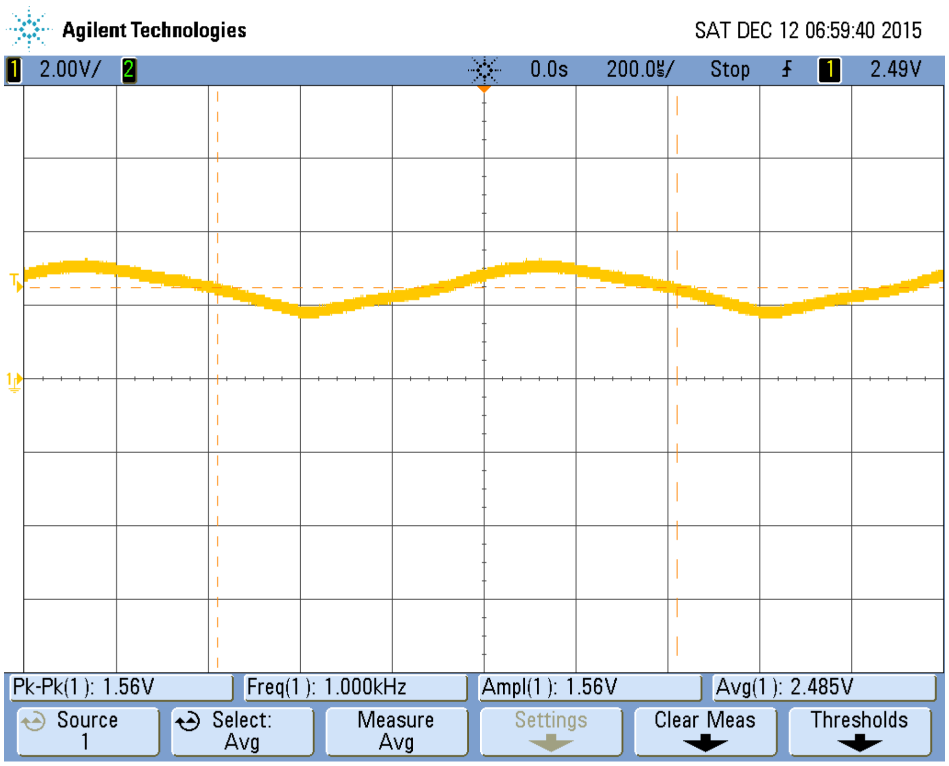


Fig 2.5 1 kHz sine wave audio input to microphone

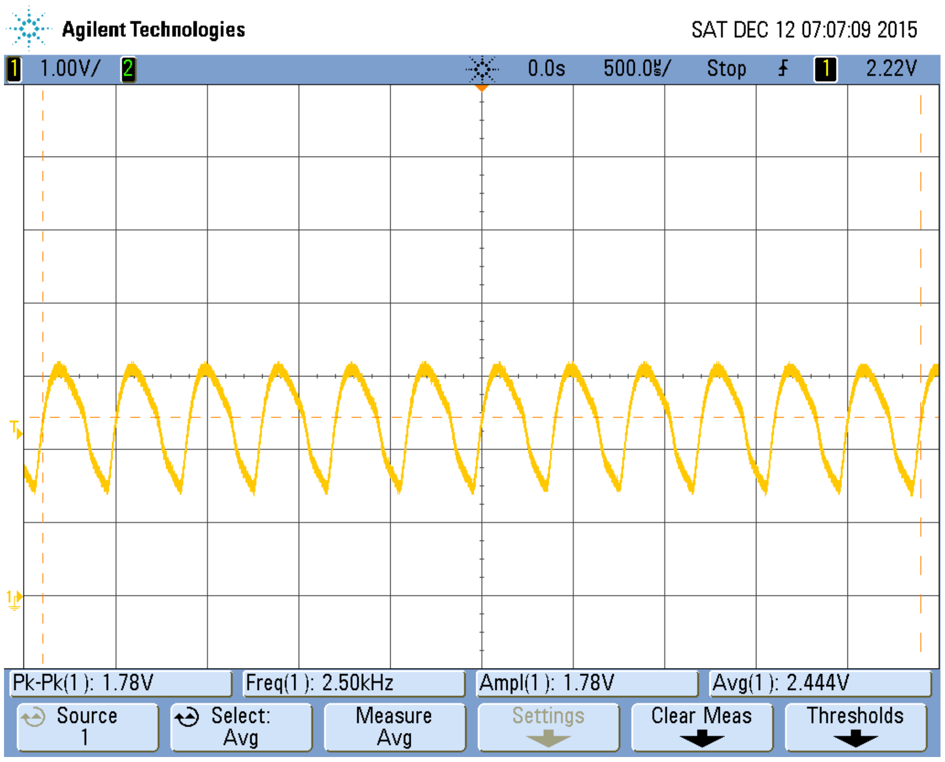


Fig 2.6 2.5 kHz sawtooth wave audio signal input to the microphone

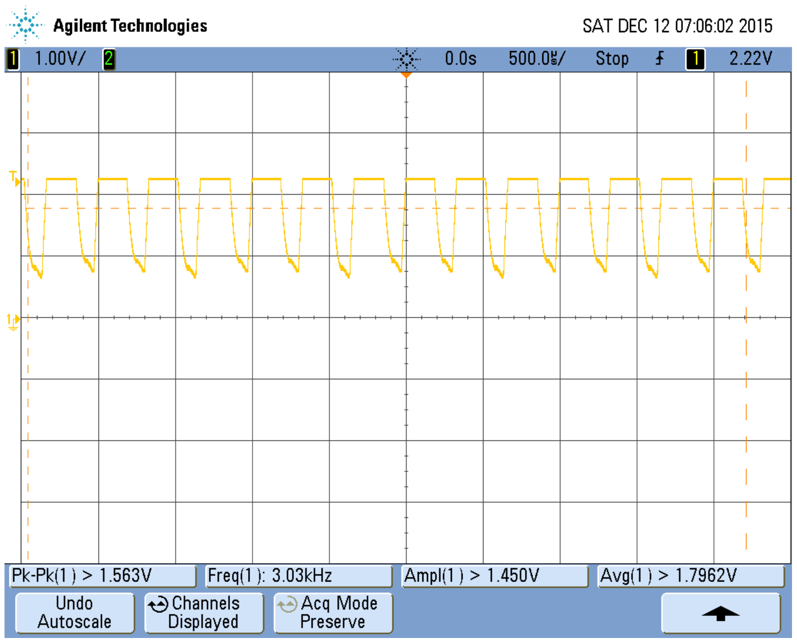


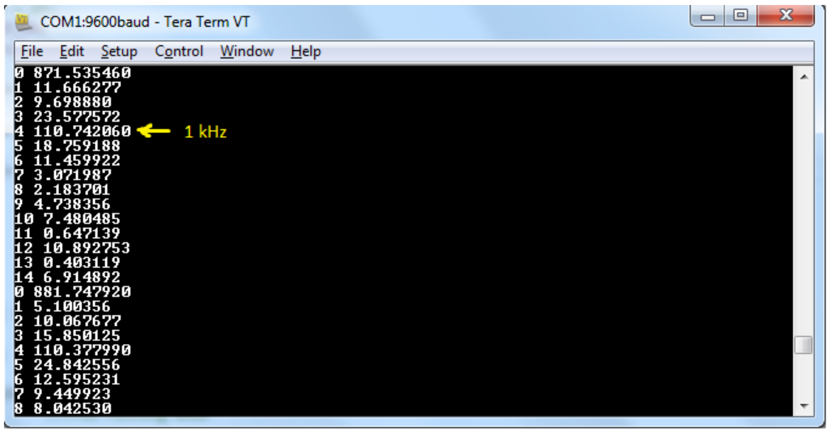
Fig 2.7 3.5 kHz square wave audio signal input to the microphone

2.7.2 ADC testing

To test the ADC, an LED circuit was hooked up and the board of the project. The ADC result of 10 bit was left shifted to obtain an 8 bit value of ADC result in ADC Data register high (ADCH). Since the input is biased at 2.5V and the reference voltage of ADC being 5V, the values of ADCH greater than 128 are considered. The LED was therefore turned on only if the value of ADCH goes beyond 128 which occurred only if there was input at the microphone circuit. Else the LED was in the off state.

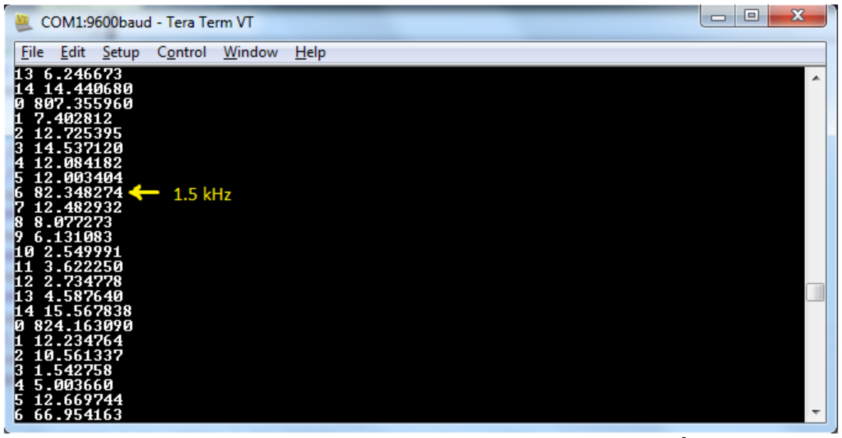
2.7.3 USART and FFT testing

Since the FFT values are in double variable type, they had to be printed to verify their authenticity. Due to this the ATMega32A had to be configured in the asynchronous mode. The FFT values were then displayed on the terminal through the MAX232 and DB9 connection circuit. Various audio input signal was given to the microphone and the output of the preamplifier circuit was given to the ADC pin7 of ATMega32A. The FFT operation was performed on the samples and its output was tested through USART.


COM1:9600baud - Tera Term VT
File Edit Setup Control Window Help
1 kHz

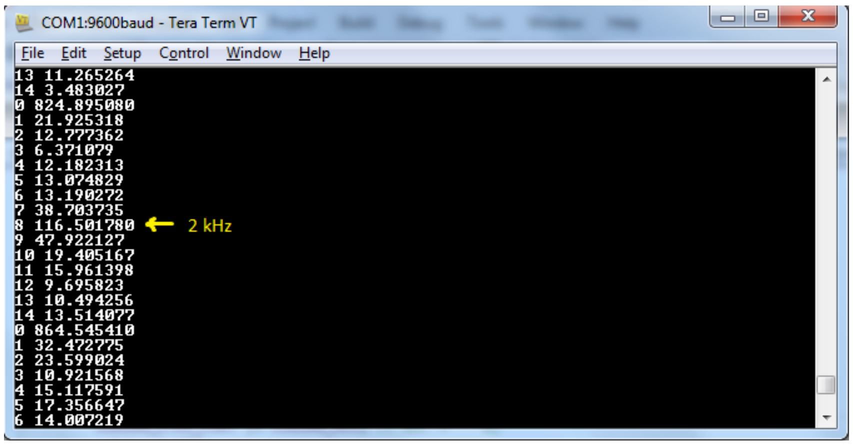
```
0 871.535460
1 11.666277
2 9.698880
3 23.577572
4 110.742060
5 18.759188
6 11.459922
7 3.071987
8 2.183701
9 4.738356
10 7.480485
11 0.647139
12 10.892753
13 0.403119
14 6.914892
0 881.747920
1 5.100356
2 10.067677
3 15.850125
4 110.377990
5 24.842556
6 12.595231
7 9.449923
8 8.042530
```

Fig 2.8 FFT output for frequency 1kHz at 5th bin


COM1:9600baud - Tera Term VT
File Edit Setup Control Window Help
1.5 kHz

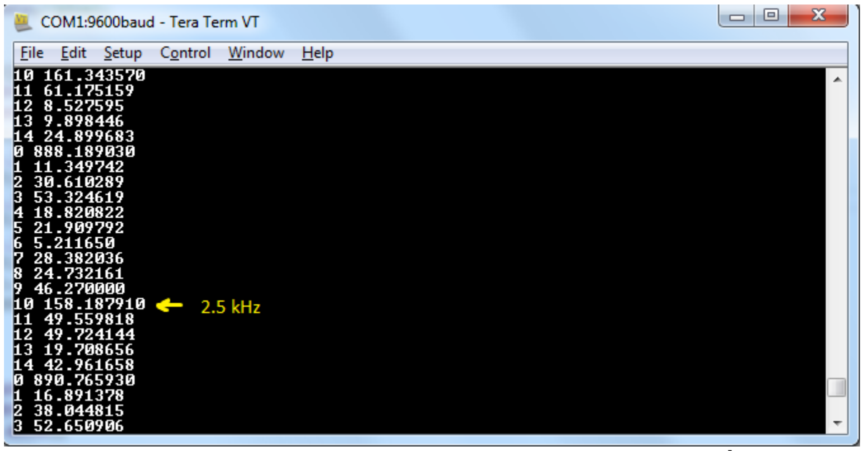
```
13 6.246673
14 14.440680
0 807.355960
1 7.402812
2 12.725395
3 14.537120
4 12.084182
5 12.003404
6 82.348274
7 12.482932
8 8.077273
9 6.131083
10 2.549991
11 3.622250
12 2.734778
13 4.587640
14 15.567838
0 824.163090
1 12.234764
2 10.561337
3 1.542758
4 5.003660
5 12.669744
6 66.954163
```

Fig 2.9 FFT output for 1.5kHz frequency at 6th bin


COM1:9600baud - Tera Term VT
File Edit Setup Control Window Help
2 kHz

```
13 11.265264
14 3.483027
0 824.895080
1 21.925318
2 12.777362
3 6.371079
4 12.182313
5 13.074829
6 13.190272
7 38.703735
8 116.501780
9 47.922127
10 19.405167
11 15.961398
12 9.695823
13 10.494256
14 13.514077
0 864.545410
1 32.472775
2 23.599024
3 10.921568
4 15.117591
5 17.356647
6 14.007219
```

Fig 2.10 FFT output for 2kHz frequency at 8th bin


COM1:9600baud - Tera Term VT
File Edit Setup Control Window Help
2.5 kHz

```
10 161.343570
11 61.175159
12 8.527595
13 9.898446
14 24.899683
0 888.189030
1 11.349742
2 30.610289
3 53.324619
4 18.820822
5 21.909792
6 5.211650
7 28.382036
8 24.732161
9 46.270000
10 158.187910
11 49.559818
12 49.724144
13 19.708656
14 42.961658
0 890.765930
1 16.891378
2 38.044815
3 52.650906
```

Fig 2.11 FFT output for 2.5 kHz frequency at 10th bin

2.7.4 Graphical LCD testing

The graphical LCD was tested for the changing 32 point FFT samples by representing them in adjacent columns. Since the representation of one line per column left the rest of the columns in the LCD to be empty, the values of the FFT samples were chosen to be represented in the form of bar graph.

3. Results and Error Analysis

3.1 Deductions about working of the project:-

- The project output turned out to be fine and as expected.
- The audio spectral analyzer could represent various frequencies and signal input of sine wave, square wave and sawtooth audio signal when played once and also in loop.
- The analyzer was also tested to represent the gradual increasing and decreasing frequencies from 1 kHz to 2 kHz in a loop.
- The analyzer also showed good performance for gradual increasing frequency signal from 0 - 2.5 kHz
- Also the real time representation of random audio music signal was achieved by the synthesizer.

3.2 Concepts Learned

The crux of this project was the FFT transform and understanding its application in this project was crucial to have things sorted out. There are a plenty of FFT codes available on the internet. But they need to be understood well and implemented properly and match the working of the MCU to get a proper FFT result. This entire process teaches one a lot. By building the microphone circuit and ATMega32A circuit from scratch helped in understanding the working of analog circuits and the concept of Fuse bits and ISP programming. Working with AVR gave an insight into the versatility of the controller. Also testing the circuits well with multimeter, oscilloscopes, signal generator, etc before proceeding to the next process helped in making progress step by step.

3.3 Challenges faced and solved:

1. The understanding of the FFT process is important to proceed with this project.
2. Due to addition of noise of surrounding noise through the microphone, the FFT samples were not high enough to be able to plot on the Graphical LCD. Although the signal is amplified before it enters the MCU, the signals are boosted programmatically as well so they can be viewed on the Graphical LCD. With this the signal of the particular frequency is amplified well enough to be plotted and viewed on the Graphical LCD.
3. The ADC is configured in a free running mode and the FFT algorithm used is fast enough for the GLCD to catch up to display the bar graphs efficiently. The refresh rate of Graphical LCD driven by KS0108B is 20 FPS which is about 6ms. So the traces of the previous bar graph were visible in the next iteration as well. To solve this problem a delay of 5ms had to be added after displaying a set of FFT values on the Graphical LCD.

4. Conclusion

As a person who has always been fascinated about different forms of music, this project provided me a different perspective to look at it. Moreover I always wanted to work on the concepts of digital signal processing which I had learned theoretically in my undergraduate class. This project gave me an opportunity to work on those concepts and understand them more clearly.

One more reason that this project was different for me because it gave me a chance to configure AVR controller from scratch. There was a lot to learn about fuse bits of AVR, programming AVR as well as learning their architecture and instruction set.

Working on the microphone circuitry helped me learn the concepts in analog circuit design. This project helped in clearing many concepts and I enjoyed the journey of learning them.

5. Future development ideas

This project can be scaled up further if possible. The features that can be added are:

1. Interfacing a switch on the board which causes the display on the GLCD to freeze when pressed.
2. Interfacing another switch for changing the patterns in which the FFT samples are displayed on the GLCD screen.
3. Interfacing rows of multicolored LED strips that display the FFT samples as they increment or decrement.
4. The project can also be configured to choose a versatile instrument like a piano or Guitar and can produce FFT transform for the frequencies produced by their keys or strings.
5. The project can also be implemented to represent human speech.

6. Acknowledgement

I would like to thank Professor Linden McClure for this well designed course which challenges one as an engineer and also as an individual on the whole.

I would thank all the TAs who have been very supportive and helpful to answer my doubts.

Also I would like to thank all the authors and followers of the websites which help in sharing useful information.

7. References

[1] http://www.atmel.com/images/atmel-8155-8-bit-microcontroller-avr-atmega32a_datasheet.pdf

[2] https://www.sparkfun.com/datasheets/LCD/GDM12864H.pdf

[3] https://www.sparkfun.com/datasheets/LCD/ks0108b.pdf

[4] https://courses.engr.illinois.edu/ece410/documents/fft.pdf

[5] http://blogs.zynaptiq.com/bernsee/dft-a-pied/

[6] http://blog.vinu.co.in/2012/05/implementing-discrete-fourier-transform.html

[7] http://www.avrfreaks.net/forum/reset-circuit-0

[8] http://ecee.colorado.edu/~mcclurel/PCB_for_ECEN5613_9-11-2005.pdf

[9] https://www.pantechsolutions.net/mocrocontroller-boards/glcd-interfacing-with-8051-primer

[10] https://www.sparkfun.com/tutorials/105

[11] http://www.eetimes.com/document.asp?doc_id=1276974

[12] http://www.atmel.com/tools/AVRISPMKII.aspx?tab=overview

[13] http://playwithrobots.com/avr-fuse-bits/

[14] http://www.atmel.com/images/atmel-2521-avr-hardware-design-considerations_applicationnote_avr042.pdf

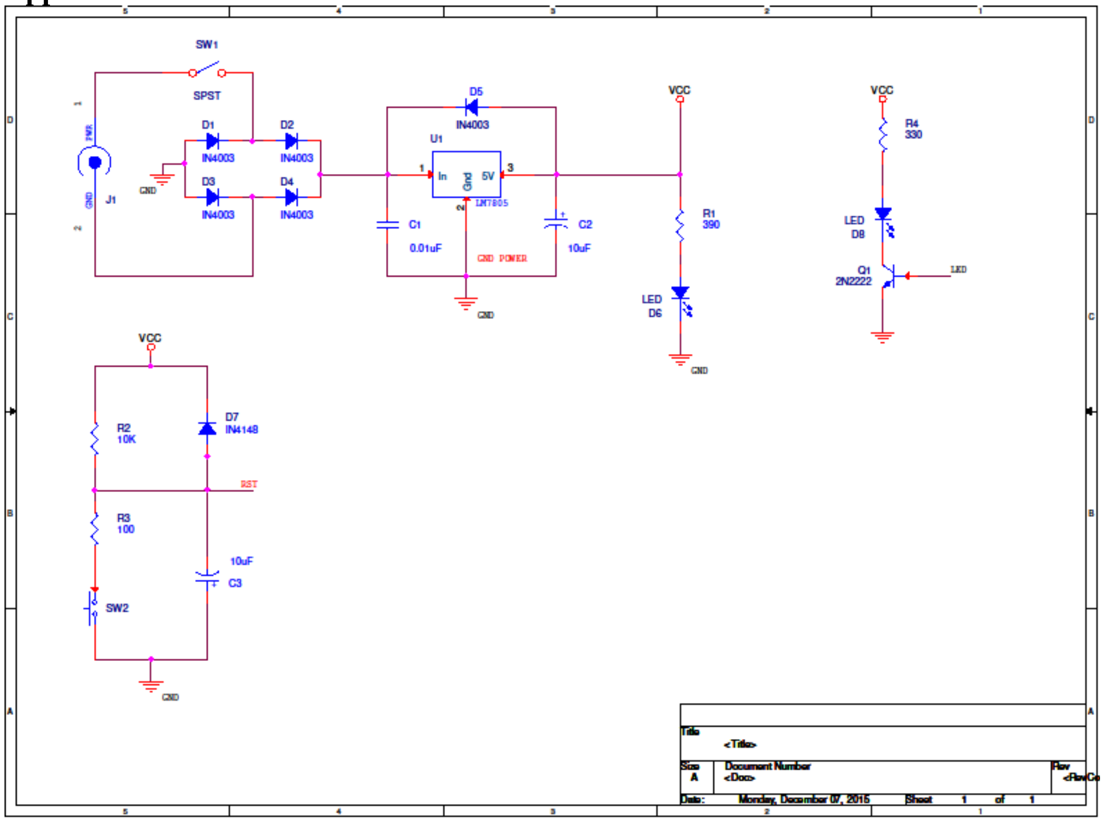
[15] http://www.atmel.com/tools/ATMELSTUDIO.aspx

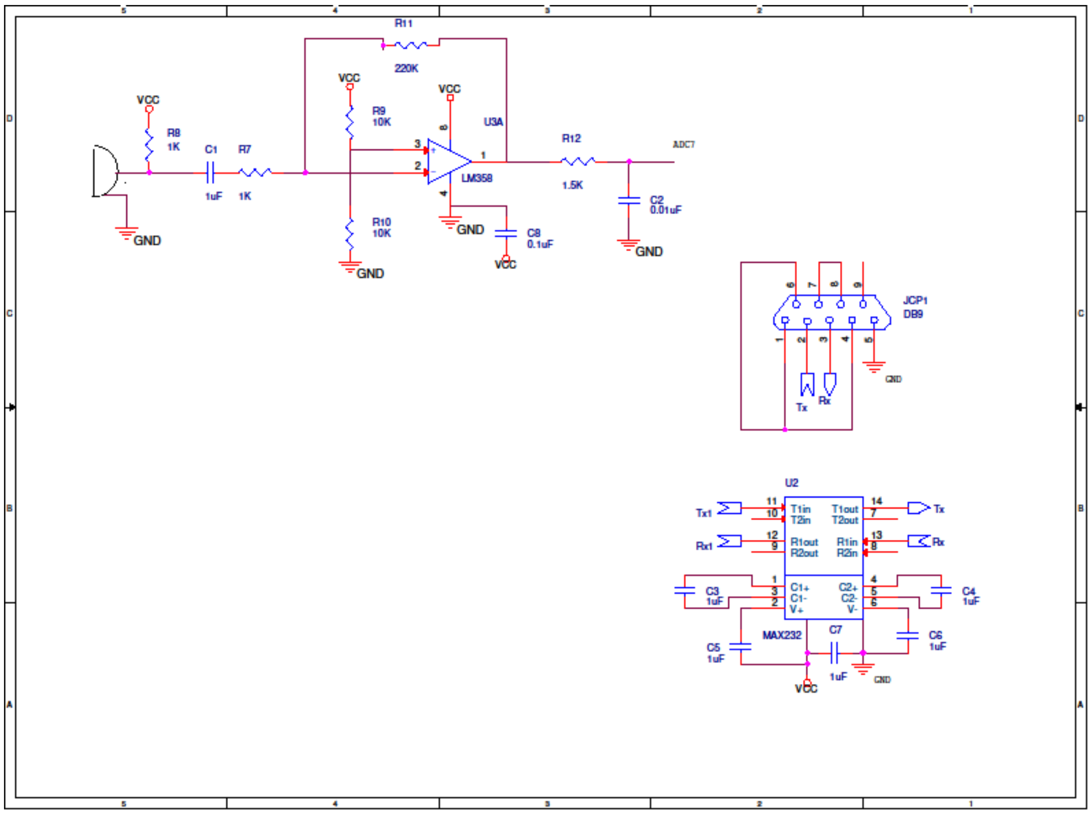
8. Appendices
The appendices used in the report are as follows:

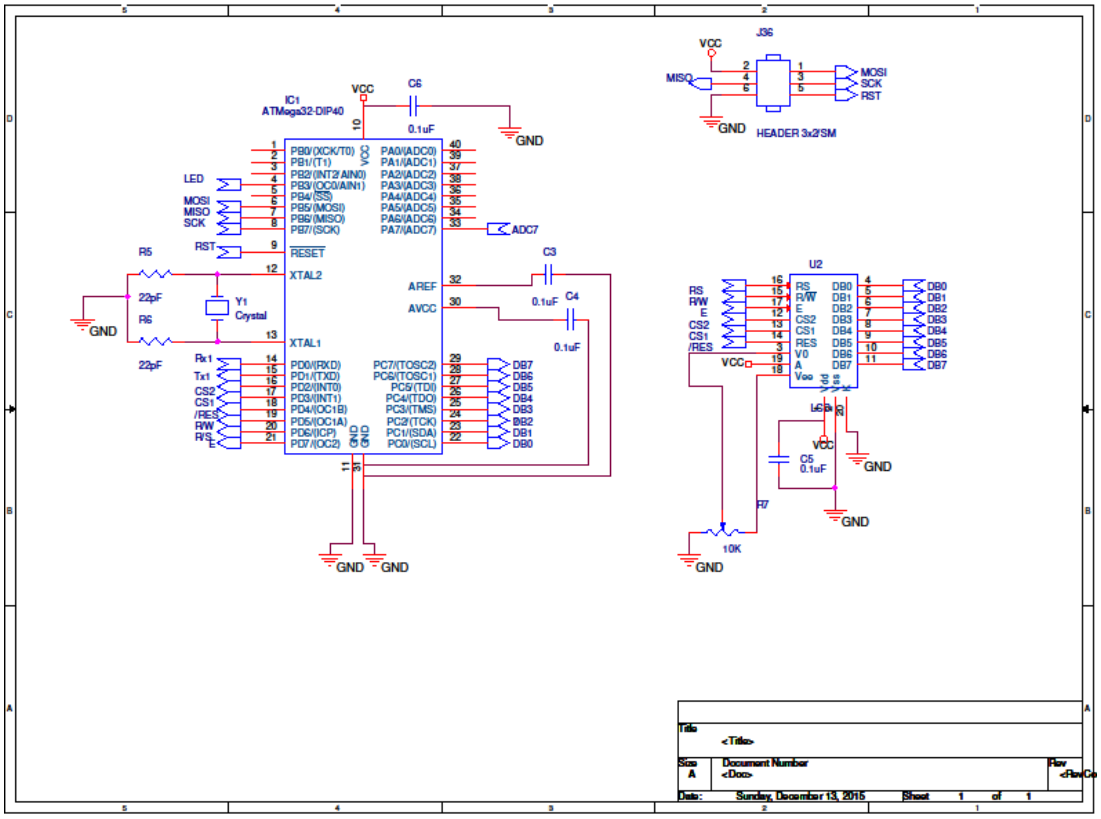
8.1 Appendix - Bill of materials

Part Description	Source	Cost
ATMega32A-PU-ND	www.digikey.com	$6.28
LM358P	www.digikey.com	$0.44
AOM-4544P-R	www.digikey.com	$0.91
LM7805	www.digikey.com	$1.66
Capacitor 0.1uF	www.digikey.com	$1.68
Crystal 16MHz	www.digikey.com	$0.36
Capacitor 1uF	JBSaunders	$0.5
Capacitor 22pF	Sparkfun	$0.5
128X64 GLCD	Sparkfun	$19.95
PCB Board	eestore	$4.30
PCB Stands	eestore	$4.66
Power Jack & Switch	JBSaunders	$3.5
Resistors (variable)	JBSaunders	$2.67
Female to female wires	eestore	$3.28
T44 Pins	JBSaunders	$0.60
Pin Headers	JBSaunders	$1.20
Total		**$52.49**

8.2 Appendix- Schematics




R11
220K
VCC
R8
1K
C1
1uF
R7
1K
R9
10K
U3A
LM358
R10
10K
GND
C8
0.1uF
R12
1.5K
ADC7
C2
0.01uF
JCP1
DB9
Tx
Rx
U2
MAX232
T1in
T2in
T1out
T2out
R1out
R2out
R1in
R2in
C1+
C1-
V+
C2+
C2-
V-
Tx1
Rx1
C3
1uF
C4
1uF
C5
1uF
C6
1uF
C7
1uF


IC1
ATMega32-DIP40
VCC
C6
0.1uF
GND
PB0/(XCK/T0)
PB1/(T1)
PB2/(INT2/AIN0)
PB3/(OC0/AIN1)
PB4/(SS)
PB5/(MOSI)
PB6/(MISO)
PB7/(SCK)
RESET
XTAL2
XTAL1
PA0/(ADC0)
PA1/(ADC1)
PA2/(ADC2)
PA3/(ADC3)
PA4/(ADC4)
PA5/(ADC5)
PA6/(ADC6)
PA7/(ADC7)
AREF
AVCC
PD0/(RXD)
PD1/(TXD)
PD2/(INT0)
PD3/(INT1)
PD4/(OC1B)
PD5/(OC1A)
PD6/(ICP)
PD7/(OC2)
PC7/(TOSC2)
PC6/(TOSC1)
PC5/(TDI)
PC4/(TDO)
PC3/(TMS)
PC2/(TCK)
PC1/(SDA)
PC0/(SCL)
LED
MOSI
MISO
SCK
RST
R5
R6
22pF
Y1
Crystal
C3
C4
0.1uF
DB7
DB6
DB5
DB4
DB3
DB2
DB1
DB0
J36
HEADER 3x2/SM
U2
RS
R/W
E
CS2
CS1
/RES
V0
A
Voo
R7
10K
C5
0.1uF
Title
<Title>
Size
A
Document Number
<Doc>
Rev
<RevCo
Date:
Sunday, December 13, 2015
Sheet
1
of
1

8.3 Source code

```
/*
 * fft1.c
 *
 * Created: 12/1/2015 3:48:04 PM
 * Author : Gayatri Mestry
 * References to leveraged code given in the function headers
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define F_CPU 16000000UL          // Frequency of oscillator
#include <util/delay.h>
#define BAUD 9600                      //baud rate
volatile double buff1[32],getmax; //buffer for FFT
void fft(int n,int m);

#define FFT_SIZE      32            // 32 point FFT implementation
#define log2FFT              5                      //2^5 = 32
#define NUM                  (2*FFT_SIZE)
#define log2N      (log2FFT + 1)

#define UBRR_Value ((F_CPU/(16UL*BAUD))-1)     //UBRR value

#define LED_CTRL_PORT PORTB     //For LED working
#define LED_CTRL_DIR  DDRB
#define LED_ON               (1 << 3)
#define UART_CTRL_PORT       PORTD     //Port for USART
#define UART_CTRL_DIR DDRD
#define UART_BEGIN           (1 << 0|1 << 1)

#define DATA_PORT     PORTC           //LCD Data port
#define DATA_DIR      DDRC
#define DATA_PIN      PINC

#define CTRL_PORT     PORTD          //LCD Control signal port
#define CTRL_DIR             DDRD

#define DI                   (1 << 6)     //Data/Instruction
#define RW                   (1 << 5)        // Read/Write
#define EN                   (1 << 7)     //Enable
#define RES                  (1 << 4)     //Reset
#define CS1                  (1 << 3)     //Chip Select1
#define CS2                  (1 << 2)     //Chip select2

#define SCREEN_WIDTH  128      //no of columns
#define SCREEN_HEIGHT 64       //number of sections of pages

//Commands for GLCD
#define DISPLAY_SET_Y       0x40
#define DISPLAY_SET_X       0xB8
#define DISPLAY_START_LINE  0xC0
#define DISPLAY_ON_CMD               0x3E

#define ON    0x01
#define OFF   0x00
#define DISPLAY_STATUS_BUSY  0x80
unsigned char screen_x;
unsigned char screen_y;
unsigned int show = 0,sample = 0,j=0;

/*      Function Declarations         */
static int usart_putchar1(char c, FILE *stream);
static FILE print = FDEV_SETUP_STREAM(usart_putchar1, NULL, _FDEV_SETUP_WRITE);

void USART_Init(); // configure USART
void USART_tx(unsigned char send_data); // Transmit
void USART_putstring(char* StringPtr);
unsigned char USART_rx(); // Receive
void GLCD_InitalizePorts(void);
void GLCD_Init();
void Select_page(unsigned char );
void GLCD_Comd(unsigned char );
void GLCD_Data(unsigned char dat);
void GLCD_ClearScreen(void);
void GOTO_XY(unsigned int x,unsigned int y);
void GLCD_Draw_line(unsigned short x,unsigned short y, unsigned short j);
void GLCD_Spectrum(double *value);
void GLCD_Bar_Graph(unsigned short x,unsigned short y, unsigned short j);


unsigned char data;

int main(void)
{

       /* Replace with your application code */
       unsigned char data;//pixel;
       double max=0;
       int i;
       double buff2[32];//min = 15;
              LED_CTRL_DIR |= LED_ON;
              LED_CTRL_PORT &= ~LED_ON;
              _delay_ms(2);
              CTRL_PORT |= RES; // RST = 1;             Resetting the AVR
              _delay_ms(5);
              CTRL_PORT &= RES; // RST =   0;
              _delay_ms(5);
              CTRL_PORT |= RES; // RST =    1;
              _delay_ms(5);
              GLCD_Init();             //Initialize GLCD
              _delay_ms(15);
       //USART_Init();


       ADCSRA |= (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);   //prescaler --125kHz
       ADMUX |=  (1<<REFS0);                                                 //ADC
reference to AVcc
       ADMUX |= (1 << ADLAR);                                               //Left shifting
ADC Data register

       ADMUX |= (1<<MUX2) | (1<<MUX1) |(1<<MUX0);   //selecting adc7
       ADCSRA |= (1<<ADATE);                                                //free  running
mode

       ADCSRA |= (1<<ADEN);                                                  //enable adc

       sei();
       ADCSRA |= (1<<ADSC);                                                 //start     adc
conversion

       //data = USART_rx();
       while (1)
       {

              for (i=0; i<NUM; i++)
              {
                     while(!(ADCSRA & (1<<ADIF)));        //checking  idf  value  arrived
in ADC Data Register High
                     buff1[i] = ADCH;                                //placing   the
values of ADCH in a buffer
                     _delay_us(125);                      //sampling at 8000Hz

              }

              fft(32,5);                            // Performing Fast Fourier Transform

              for(j=0;j<15;j++)
              {
                     if(buff1[j]>max)                      //finding the maximum value in
the buffer array of FFT values
                     max = buff1[j];
              }

              for(j=0;j<15;j++)
              {
                     (buff2[j]) = ((buff1[j])/max)*900;     //normalizing the array
                     //printf("%d %f\n",j,buff2[j]);
              }

              GLCD_Spectrum(buff2);                                  //Drawing   the   GLCD
spectrum to show all bar graphs
       }
       return 0;
}
```

```
/*  void USART_Init()
---This is used for initializing the USART by setting the baud rate through the UBRR value
---The parity bit and the data length of 8 bits is configured and the number of stop bits
is 1
---The function also implement to use printf functionality to print values to the terminal
*/
void USART_Init()
{
       UBRRH=(1<<7)|(unsigned char)(UBRR_Value>>8);
       UBRRL=(unsigned char)(UBRR_Value);
       UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
       UCSRB = (1<<RXEN)|(1<<TXEN);
       UCSRA &= 0x00;
       stdout = &print;
}

/*     void USART_tx(unsigned char send_data)
---In this function the UDRE bit is polled to know if the transmit buffer
---is empty and ready to send data                          */
void USART_tx(unsigned char send_data)
{
       while(!(UCSRA & (1<<UDRE)));
       UDR = send_data;
}

/*   unsigned char USART_rx()
---The RXC bit in USCRA register is polled to check if the data
---is arrived in the UDR buffer after reception of a byte    */
unsigned char USART_rx()
{
       while(!(UCSRA & (1<<RXC)));
       return UDR;
}

/*   void USART_putstring(char* StringPtr)
--This function is used to print a string of characters by using the USART_tx function  */
void USART_putstring(char* StringPtr){

       while(*StringPtr != 0x00){
              USART_tx(*StringPtr);
       StringPtr++;}

}

/* static int usart_putchar1(char c, FILE *stream)
This is a standard function for using printf()
--Referred  to  user  manual  http://www.gnu.org/savannah-checkouts/non-gnu/avr-libc/user-
manual/group__avr__stdio.html  */
static int usart_putchar1(char c, FILE *stream)
{
       if (c == '\n') usart_putchar1('\r', stream);

       while(!(UCSRA & (1<<UDRE)));
       UDR = c;

       return 0;
}

/***********************************************************/
/* fft.c                                                   */
/* (c) Douglas L. Jones                                    */
/* University of Illinois at Urbana-Champaign              */
/* January 19, 1992                                        */
/*                                                         */
/*   fft: in-place radix-2 DIT DFT of a complex input      */
/*                                                         */
/*   input:                                                */
/* n: length of FFT: must be a power of two                */
/* m: n = 2**m                                             */
/*   input/output                                          */
/* x: double array of length n with real part of data      */
/* y: double array of length n with imag part of data      */
/*                                                         */
/*   Permission to copy and use this program is granted    */
/*   under a Creative Commons "Attribution" license        */
/*   http://creativecommons.org/licenses/by/1.0/           */
/***********************************************************/
void fft(int n,int m)
{
```

```
int i,j,k,n1,n2;
       double c,s,e,a,t1,t2;
       double y[32]={0.0};


       j = 0; /* bit-reverse */
       n2 = n/2;
       for (i=1; i < n - 1; i++)
       {
              n1 = n2;
              while ( j >= n1 )
              {
                     j = j - n1;
                     n1 = n1/2;
              }
              j = j + n1;

              if (i < j)
              {
                     t1 = buff1[i];
                     buff1[i] = buff1[j];
                     buff1[j] = t1;
                     t1 = y[i];
                     y[i] = y[j];
                     y[j] = t1;
              }
       }


       n1 = 0; /* FFT */
       n2 = 1;

       for (i=0; i < m; i++)
       {
              n1 = n2;
              n2 = n2 + n2;
              e = -6.283185307179586/n2;
              a = 0.0;

              for (j=0; j < n1; j++)
              {
                     c = cos(a);
                     s = sin(a);
                     a = a + e;

                     for (k=j; k < n; k=k+n2)
                     {
                            t1 = c*buff1[k+n1] - s*y[k+n1];
                            t2 = s*buff1[k+n1] + c*y[k+n1];
                            buff1[k+n1] = buff1[k] - t1;
                            y[k+n1] = y[k] - t2;
                            buff1[k] = buff1[k] + t1;
                            y[k] = y[k] + t2;

                     }
              }
       }
       for(k=0;k<128;k++)
       {

              buff1[k]=sqrt(buff1[k]*buff1[k]+y[k]*y[k]);
       }

}

/*     void GLCD_InitalizePorts(void)
---The GLCD control signal ports are initialized for output   */
void GLCD_InitalizePorts(void)
{
       CTRL_DIR |= (CS1 | CS2 | DI | RW | EN | RES);
       CTRL_PORT &= ~(CS1 | CS2| DI | RW | EN );
}

/*      void GLCD_Delay(void)
---Delay function to introduce delay for proper functioning of the GLCD  */
void GLCD_Delay(void)
{
       _delay_us(9);
}
```

```
/*     void GLCD_Init()
---This is the initialzation function for GLCD
---The initialization commands are sent to setup the display screen to both the chips
Reference  https://www.pantechsolutions.net/microcontroller-boards/glcd-interfacing-with-
8051-slicker */
void GLCD_Init()
{
       int i;
       unsigned char Comd[5]={0xC0,0xB8,0x40,0x3F};//LCD Command list
       GLCD_InitalizePorts();
       Select_page(1);           //send commands to page1
       for(i=0;i<4;i++)
       GLCD_Comd(Comd[i]);
       Select_page(0);           //send commands to page0
       for(i=0;i<4;i++)
       GLCD_Comd(Comd[i]);
       GLCD_ClearScreen();
}

/*  void Select_page(unsigned char Page)
--This function enables a chip and disables the other to get or set the characters on GLCD
*/
void Select_page(unsigned char Page)
{
       if(Page)
       {
              CTRL_PORT &= ~CS1;//CS1=0;       //Page 0 LCD IC1
              CTRL_PORT |= CS2; //CS2=1;

       }
       else
       {
              CTRL_PORT &= ~CS2;
              CTRL_PORT |= CS1;

       }
}

/*   void GLCD_Comd(unsigned char cmnd)
---This function initializes the GLCD control signal to send a command byte from the
ATMega32A to GLCD  */
void GLCD_Comd(unsigned char cmnd)
{
       DATA_DIR = 0xFF;
       DATA_PORT = cmnd;
       CTRL_PORT &= ~(DI);
       CTRL_PORT &= ~(RW);
       CTRL_PORT |= EN;           //Enable high
       GLCD_Delay();
       CTRL_PORT &= ~EN;          //Enable low
}

/*   void GLCD_Data(unsigned char dat)
--- The functions sets the GLCD control signals to send a data byte from ATMega32A to the
GLCD */
void GLCD_Data(unsigned char dat)
{
       DATA_DIR = 0xFF;
       DATA_PORT = dat;
       CTRL_PORT |= (DI);
       CTRL_PORT &= ~(RW);
       CTRL_PORT |= EN;
       GLCD_Delay();
       CTRL_PORT &= ~EN;
}

/*    void GLCD_ClearScreen(void)
---The function configures the columns of eac---h page as it iterates on the GLCD screen
---and writes the dat 0x00 to the screen to clear it                         */
void GLCD_ClearScreen(void)
{
       int Page=0;
       int Column=0;

       for (Page = 0; Page < 8; Page++)
       {
              Select_page(1);                        //Display part of image to Page1
              GLCD_Comd(0xB8 | Page);
```

```
	GLCD_Comd(0x40);

		for (Column = 0; Column < 128; Column++)
		{
			if (Column == 64)
			{
				Select_page(0);			//Display part of image 
to Page0
				GLCD_Comd(0xB8 | Page);
				GLCD_Comd(0x40);
			}
			GLCD_Data(0x00);
		}
	}
}

/*    void GOTO_XY(unsigned int x,unsigned int y)
---This functions takes the parameters as the x and y location on the screen of GLCD 
display
---and sets he x and y location address to go to that particular location on GLCD	*/
void GOTO_XY(unsigned int x,unsigned int y)
{
	unsigned short Col_Data;
	CTRL_PORT &= ~(DI);
	CTRL_PORT &= ~(RW);
	if(y<64) //left section of GLCD
	{
		Col_Data = y;
		Select_page(1);
		GLCD_Comd(0xB8 | x);			//set x-address

		//Display part of image to Page0
		GLCD_Comd(DISPLAY_SET_Y|Col_Data);	//set y address
	}
	else
	{
		Col_Data = y-64; //select the next chip area to display
		Select_page(0);				//Display part of image to Page1
		GLCD_Comd(0xB8 | x);
		GLCD_Comd(DISPLAY_SET_Y|Col_Data);   //set Y address
	}

}

/*       void GLCD_Bar_Graph(unsigned short x,unsigned short y, unsigned short j)
---This function uses the  GLCD_Draw_line() function to display a line and increment it
---depending upon the value of FFT and plots a graph		*/
void GLCD_Bar_Graph(unsigned short x,unsigned short y, unsigned short j)
{
	unsigned short xaxis =x;//yaxis=y;
	unsigned short k,l,m,n;
	l=j/8;				//number of all dark columns
	m=j%8;				// number of columns to be filled extra to 
represent FFt value
	xaxis=0;
	for(n=0;n<l;n++)
	{
		k=7;
		GLCD_Draw_line(xaxis,y,7);
		xaxis++;
	}
	for(n=0;n<m;n++)
	{
		k=m;
		GLCD_Draw_line(xaxis,y,k);
	}

}

/*    void GLCD_Draw_line(unsigned short x,unsigned short y, unsigned short j)
---This function takes in the x and y location and the value to be fille in th ecolumns to
---show the bar graphs properly. The GLCD Font Crator tool is used to get the values
*/
void GLCD_Draw_line(unsigned short x,unsigned short y, unsigned short j)
{
	unsigned short Col_Data=0x00,i;
```

```
                for(i=0;i<8;i++)
                {
                        switch(j)
                        {
                                case 0: Col_Data = 0x01; break;
                                case 1: Col_Data = 0x03; break;
                                case 2: Col_Data = 0x07; break;
                                case 3: Col_Data = 0x0F; break;
                                case 4: Col_Data = 0x1F; break;
                                case 5: Col_Data = 0x3F; break;
                                case 6: Col_Data = 0x7F; break;
                                case 7: Col_Data = 0xFF; break;
                        }

                        GOTO_XY(x,(y));
                        GLCD_Data(Col_Data);
                y--;
                }

        }

        /*   void GLCD_Spectrum(double *value)
        --The function takes in the value of fft buffer as a parameter
        ---and plots the graph for frequency bins from 1 to 14 on the GLCDvdisplay screen */
        void GLCD_Spectrum(double *value)
        {
                GLCD_ClearScreen();
                GLCD_Bar_Graph(0,127,value[1]);
                GLCD_Bar_Graph(0,118,value[2]);
                GLCD_Bar_Graph(0,109,value[3]);
                GLCD_Bar_Graph(0,100,value[4]);
                GLCD_Bar_Graph(0,91,value[5]);
                GLCD_Bar_Graph(0,82,value[6]);
                GLCD_Bar_Graph(0,73,value[7]);
                GLCD_Bar_Graph(0,64,value[8]);
                GLCD_Bar_Graph(0,55,value[9]);
                GLCD_Bar_Graph(0,46,value[10]);
                GLCD_Bar_Graph(0,37,value[11]);
                GLCD_Bar_Graph(0,28,value[12]);
                GLCD_Bar_Graph(0,19,value[13]);
                GLCD_Bar_Graph(0,10,value[14]);
            _delay_ms(12);

        }
```

8.4 Appendix - Datasheet and Application Notes

1. ATMega32A Datasheet
   http://www.atmel.com/images/atmel-8155-8-bit-microcontroller-avr-atmega32a_datasheet.pdf
2. Graphical LCD Datasheet
   https://www.sparkfun.com/datasheets/LCD/GDM12864H.pdf
3. KS0108B Driver Datasheet
   https://www.sparkfun.com/datasheets/LCD/ks0108b.pdf
4. Electret Microphone Datasheet
   http://www.puiaudio.com/pdf/AOM-4544P-R.pdf
5. Operational Amplifier Datasheet
   http://datasheet.octopart.com/LM358P-Texas-Instruments-datasheet-8220297.pdf
6. ATMEL Application Note
   http://www.atmel.com/images/atmel-2521-avr-hardware-design-considerations_applicationnote_avr042.pdf