ADC Calibration and Data Validation using GPIO

Tushar Tarihalkar.
Electrical Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
E-mail: tushar.tarihalkar@sjsu.edu

Abstract

This project explores the interfacing of the Raspberry Pi 3 B+ with an Analogue to digital converter for verification of nyquist criteria by comparing practical and theoretical values for the GPIO interface varied by a Potentiometer. here we design a power supply circuit and build a prototype module using an ADC to verify the data validation of the obtained results using Fast Fourier Transform (FFT) to obtain the power spectrum. This is successfully implemented by facilitating an I2C protocol for RPI to detect and write varying the 10k Potentiometer from 0V to 3.33V and analyzing range of digital outputs obtained to satisfy the Nyquist's Criteria.

1. Introduction

The main purpose of the experiment is the design and build a prototype board to interface the Raspberry P module to fetch digital output from an analogue to digital converter by varying the 10k potentiometer for fixed range of 0V to 3.3 Volts to obtain corresponding digital values. A C-code is compiled to facilitate this experimental value.

Hardware:
1.Raspberry Pi 3 B+.
2.Analogue to digital converter ADS1115.
3.Wire wrapping board.
4.Pull up resistor 540 ohm.
5.External dc voltage supply 5V.
6.Connector wires.
7. Resistor (150 ohm), 2*Capacitors (1µF)
8.LEDs, LM7805 Regulator
(7 and 8 are Required for building a Power supply circuit)

Software:
1.Raspbian Operating system on a Ssd card for Raspberry pi GUI interface.
2.VNC server for remote access of Raspberry PI.
3.Write a C-code to facilitate i2c between Rpi and ADC.

2. Methodology

Boot the ssd card into the raspberry pi to establish the raspbian OS environment. Check the GPIO pin in-out connections to the ADC and Potentiometer as follows

ADS1x15 VDD to Raspberry Pi 5V (pin 2) to Pot VCC
ADS1x15 GND to Raspberry Pi GND (pin 9) to Pot GND
ADS1x15 SCL to Raspberry Pi SCL (pin 5)
ADS1x15 SDA to Raspberry Pi SDA (pin 3) to Pot A0

Check if the I2C is enabled from the raspberry pi using " sudo raspi-config", you can also enable VNC server and other required interfacing options. write a c/python code to interface the ADC using the RPI to get varying digital values for your corresponding change in the voltage through the change in potentiometer positions.

2.1. Objectives and Technical Challenges

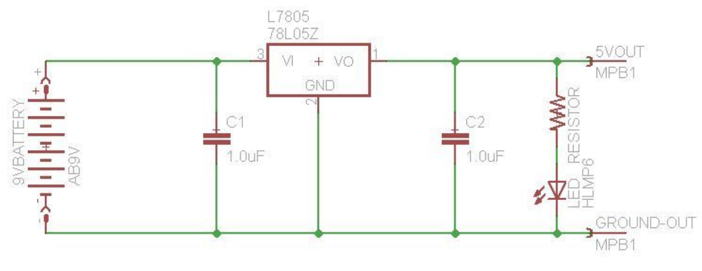
Obtain the experimental values of the ADC using RPI to check nyquist's criteria for the obtained and theoretical values. a c program has been written for the purpose.
Designing of a power circuit which will facilitate the power budget for the system is a crucial challenge. The power budget depending on the utilization as described in Table 1.

No.	Description	Consumption
U1	ARM CPU Module (RPi)	300mA
U2	Motor driver board	1200mA
U3	ADS1115	150µA
	Remaining Glue logic	200mA

Table.1. Power Consumption.

2.2. Designing a Power supply circuit.

Figure 2.2.1. Schematic for Power Distribution Circuit.



One of the important sources of DC Supply are Batteries. But using batteries in sensitive electronic circuits is not a good idea as batteries eventually drain out and lose their potential over time

The voltage provided by batteries are typically 1.2V, 3.7V, 9V and 12V. This is good for circuits whose voltage requirements are in that range. But, most of the TTL IC's work on 5V logic and hence we need a mechanism to provide a consistent 5V Supply.

Pin 1 is the INPUT Pin. A positive unregulated voltage is given as input to this pin. Pin 2 is the GROUND Pin. It is common to both Input and Output. Pin 3 is the OUTPUT Pin. The output regulated 5V is taken at this pin of the IC.

2.3. Pin diagrams for the components used:

A powerful feature of the Raspberry Pi is the row of GPIO (general-purpose input/output) pins along the top edge of the board. A 40-pin GPIO header is found on all current Raspberry Pi boards (unpopulated on Pi Zero and Pi Zero W). Prior to the Pi 1 Model B+ (2014), boards comprised a shorter 26-pin header. the numbering of the GPIO pins is not in numerical order; GPIO pins 0 and 1 are present on the board (physical pins 27 and 28) but are reserved for advanced use (see below). It also includes features like I2C bus protocols, SPI, PWM and Serial transmit and Receive.

Figure 2.3.1. Pin configuration for GPIO Interface:

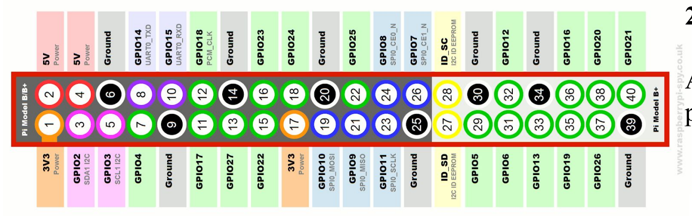
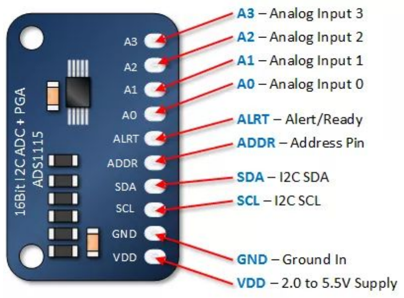


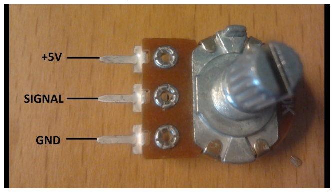
Figure 2.3.2. Pin Configuration of ADC ADS1115:

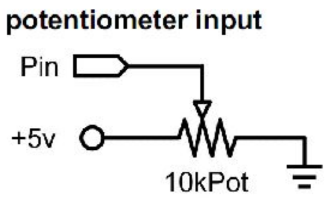


The ADS1115 are precision analog-to-digital converters (ADCs) with 16 bits of resolution offered in an ultra-small, an MSOP-10 package. Data are transferred via an I2C-compatible serial interface, four I2C slave addresses can be selected, it operate from a single power supply at 3.3V. It can be used to detect analog signal and convert it to digital signal. You can attach a joystick or other analog sensor such as NTC,temperature, dust sensor and more.

A potentiometer is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a variable resistor or rheostat described in figure 2.3.3.

Figure 2.3.3. Pin configuration of 10k Potentiometer:



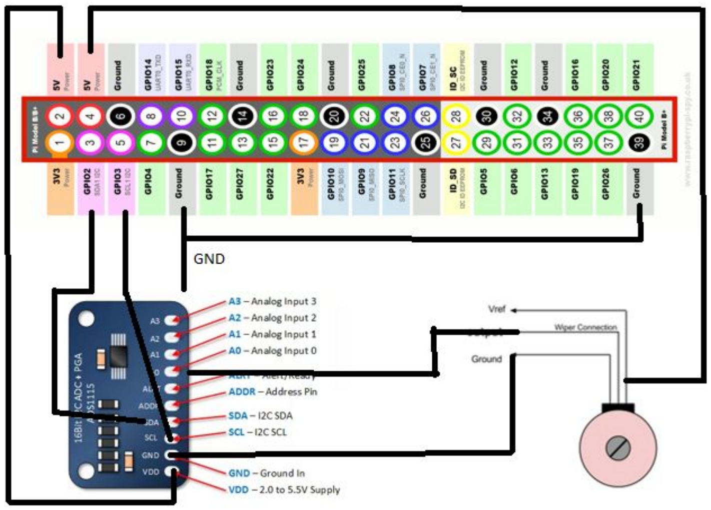


Figures 2.3.1 to figure 2.3.3 utilized depending on the required configuration for validation of system ADC.

2.4. ADC Connections for Calibration.

Connections are made for the calibration of the ADC and the Data validation via variations in potentiometer.

Figure 2.4 Circuit connections.



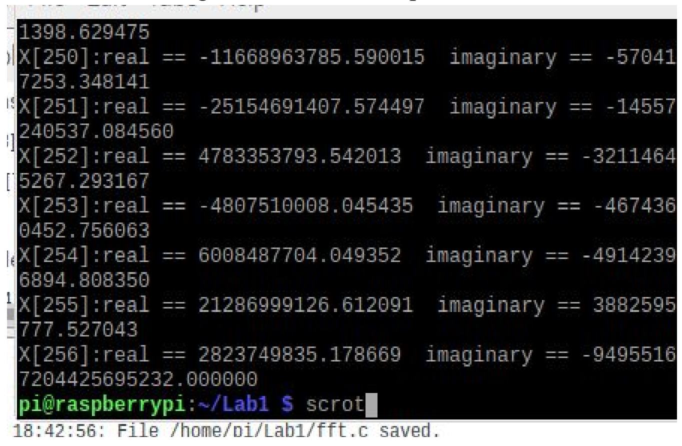
2.4.1 Data validation (Output obtained from the ADC)
Pseudo code for ADC Calibration:

```
##include <sys/ioctl.h> //include this library for further
//calibration
int main() {
   FILE *fp1, *fp2;
   fp1 = fopen("/tmp/volt.txt","w+");
   fp2 = fopen("/tmp/digital.txt","w+");

   // open device on /dev/i2c-1 the default on Raspberry Pi
B
   if ((fd = open("/dev/i2c-1", O_RDWR)) < 0) {
      printf("Error: Couldn't open device! %d\n", fd);
      exit (1);
   }
//a temporary file is created to store the voltage and digital
values obtained from the ADC. these values are separately
stored for further formulation of the FFT and Power
spectrum from the data obtained.

const float VPS = 4.096 / 32768.0; // volts per step

// power down ASD1115
   writeBuf[0] = 1;    // config register is 1
   writeBuf[1] = 0b11000011; // bit 15-8 0xC3 single shot
//on
   writeBuf[2] = 0b11100101; // bits 7-0  0x85
   if (write(fd, writeBuf, 3) != 3) {
perror("Write to register 1");
      exit (1);
```

2.4.2 FFT Calibration.

The following outputs are obtained by considering the values of the ADC obtained previously to get a Fourier transform of the ADC values as defined in the pseudo code.

Figure 2.4.2. FFT Output.



```
1398.629475
X[250]:real == -11668963785.590015  imaginary == -57041
7253.348141
X[251]:real == -25154691407.574497  imaginary == -14557
240537.084560
X[252]:real == 4783353793.542013  imaginary == -3211464
5267.293167
X[253]:real == -4807510008.045435  imaginary == -467436
0452.756063
X[254]:real == 6008487704.049352  imaginary == -4914239
6894.808350
X[255]:real == 21286999126.612091  imaginary == 3882595
777.527043
X[256]:real == 2823749835.178669  imaginary == -9495516
7204425695232.000000
pi@raspberrypi:~/Lab1 $ scrot
18:42:56: File /home/pi/Lab1/fft.c saved.
```

Pseudo code for FFT Calibration:

```
int main(void)
{
   FILE *fp1, *fp2;
   fp1 = fopen("/tmp/volt.txt","r+");
   fp2 = fopen("/tmp/fft.txt","w+");

   FILE *gnuplot = popen("gnuplot","w");


//use the output obtained from the ADC values that are
stored in the file and derive the FFT using the following
Pseudo code://

//To display the inputs considered:
float arr[5] = {1.3, 2.8, 3.6, 2.1, 3.3};
   int i;
   for (i = 0; i < 5; i++)
   {
      X[i].a = arr[i];
      X[i].b = 0.0;
   }

   printf ("*********Before*********\n");
   for (i = 1; i <= 4; i++)
         printf ("X[%d]:real == %f  imaginary == %f\n", i,
X[i].a, X[i].b);
   FFT();

   fprintf(gnuplot, "plot '-'\n");
//to obtain the considered values from the opened file//

//to evaluate and display the FFT output.//
printf ("\n\n**********After*********\n");
   for (i = 1; i <= 4; i++)
   {
         printf ("X[%d]:real == %f  imaginary == %f\n", i,
X[i].a, X[i].b);
      fprintf(gnuplot, "%g %g\n", X[i].a,X[i].b);
   }
   fprintf(gnuplot,"e\n");
   fflush(gnuplot);
   return 0;
```

3. Problem Formulation and Design

The adc characteristic can be obtained by using the η equation for efficiency. where delta stands for the power spectrum of the obtained values at the considered range.

$$\eta = [\, \Sigma\Delta\ (\textit{for a fixed range}) / \Sigma\Delta\ (\textit{over the entire range})\,]$$

the discrete fourier transform to be analysed can be checked by using a fixed number of samples for a given nyquist criteria using the following equation

$$X(k) = \sum_{n=0}^{N-1} x(n) \times e^{-i\frac{2\pi nk}{N}}$$

the sampled output should satisfy the nyquist criteria depicted by

$$f_s \geq 2 f_{m(\max)}$$

where f_s is the sampling frequency and f_{max} the maximum frequency of the signal/value obtained.

Give flowchart and pseudocode description for the step-by-step discussion of your software design.

4. Implementation

in this section we will understand the general flow to be used for the design of the power supply circuit, utilization of pseudo code and steps required for ADC calibration and Validation to obtain the power spectrum.

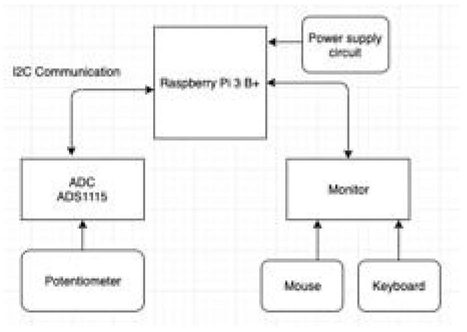


Figure 4.0 System Block Diagram

4.1.General flow:

step1. install raspbian as the environment for utilization. this can be done by booting a ssd card by downloading noobs image from the official website on the ssd in a bootable format.

step2. initiate the raspberry pi with the required power supply designed.

step3. make connections as per the circuit diagram.

step4. check the configurations settings if the I2C is enabled. if not enable them from system peripherals or by running “sudo raspi-config” in the terminal and change settings in interface configuration.

step5. check the potentiometer to get zero voltage as the initial settings.

step6. run the C-code from the terminal to receiver the ADC. turn the potentiometer knob slowly to vary your input voltage and notice the corresponding change in the digital values displayed.

step7. note down at least 10 readings between 0V to 3.3V with the corresponding digital readings in a tabular column and plot them.

step8. run the C-code with these received values to receive an FFT compilation accordingly.

step 9. check if the nyquist’s criteria is satisfied.

the following function will be followed:

$$f(x) + g(x) = ax$$

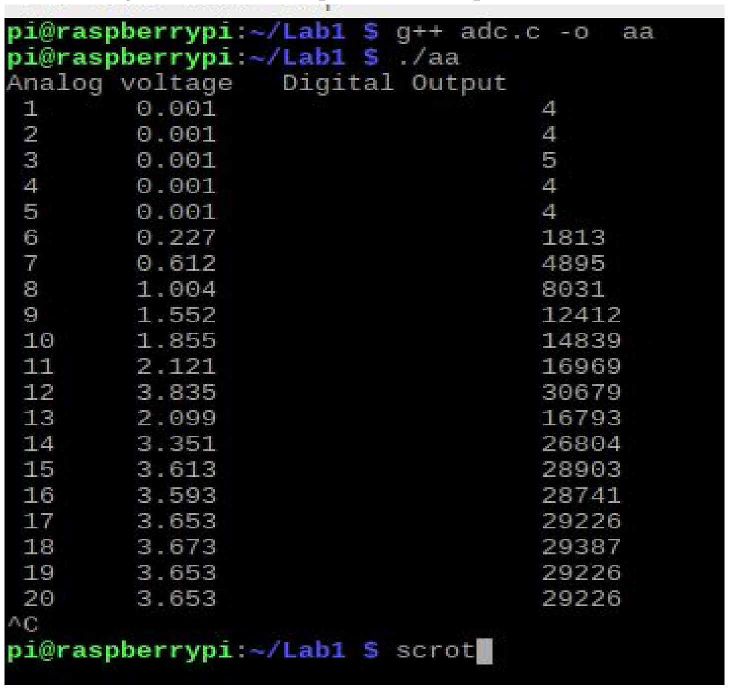
where f(x) is the function of the practically obtained values and g(x) the theoretically obtained.

5. Testing and Verification

Output of the ADC:

The following data is obtained for a sample space of 20 (pre defined in the program) to depict the change in the digital values due to corresponding change in the voltage due to the Potentiometer.

Figure 5.1 ADC Output for 20 samples.



```
pi@raspberrypi:~/Lab1 $ g++ adc.c -o  aa
pi@raspberrypi:~/Lab1 $ ./aa
Analog voltage   Digital Output
 1      0.001                  4
 2      0.001                  4
 3      0.001                  5
 4      0.001                  4
 5      0.001                  4
 6      0.227                  1813
 7      0.612                  4895
 8      1.004                  8031
 9      1.552                  12412
 10     1.855                  14839
 11     2.121                  16969
 12     3.835                  30679
 13     2.099                  16793
 14     3.351                  26804
 15     3.613                  28903
 16     3.593                  28741
 17     3.653                  29226
 18     3.673                  29387
 19     3.653                  29226
 20     3.653                  29226
^C
pi@raspberrypi:~/Lab1 $ scrot
```

ADC Output obtained for 128 readings:

x-axis: Voltage.
y-axis: digital values.

Figure 5.2 ADC Output Graphical Representation.

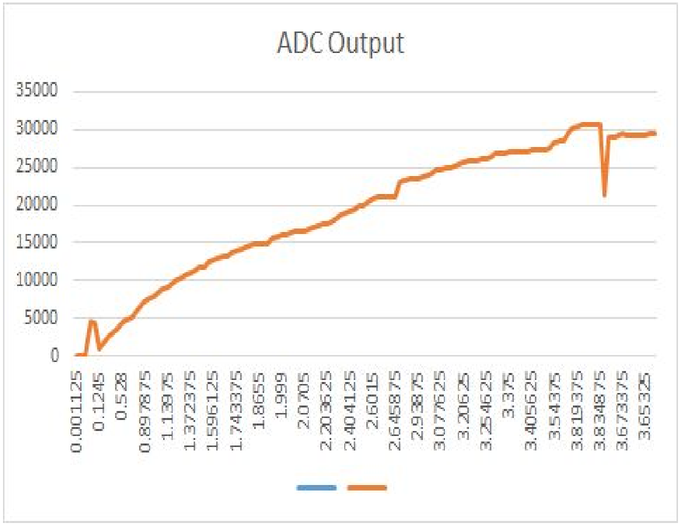
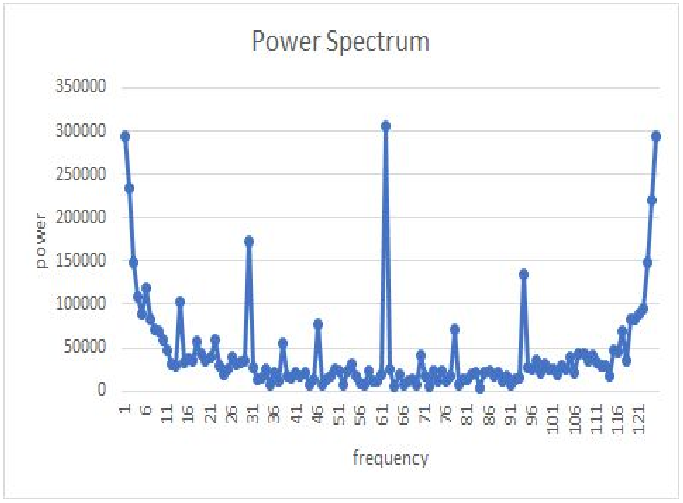


Figure 5.3 Power spectrum density (below)



6. Conclusion

This project explored the interfacing of the Raspberry Pi 3 B+ with an Analogue to digital converter for verification of nyquist criteria by comparing practical and theoretical values for the GPIO interface varied by a Potentiometer.

Here we designed an effective power supply circuit and built a prototype module using an ADC to verify the data validation of the obtained results using Fast Fourier Transform (FFT) to plot the power spectrum.

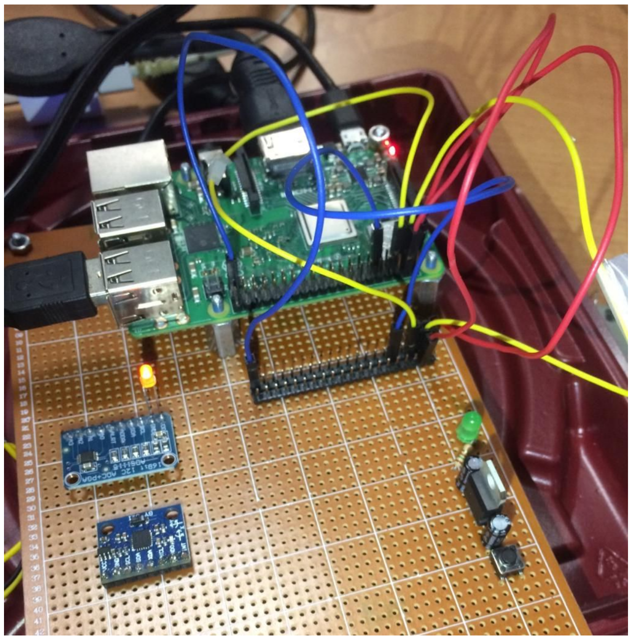


Figure 6.0 Implemented Prototype board.

7. References

[1] H. Li, "Author Guidelines for CMPE 149/242 Project Report", *Lecture Notes of CMPE 146/242*, Computer Engineering Department, College of Engineering, San Jose State University, March 18, 2019, pp. 1. https://github.com/hualili/CMPE242-Embedded-Systems-

[2]Adafruit Github for sample code to interface ADS1115 https://github.com/adafruit/Adafruit_Python_ADS1x15/blob/master/examples/simpletest.py

[3]Adafruit Github https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/ads1015-slash-ads1115

7. Appendix A

Code for ADC:

```
#include <stdio.h>
#include <unistd.h>    // read/write enable
#include <inttypes.h>  // uint8_t
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>    // exit
#include <fcntl.h>
#include <linux/i2c-dev.h> // I2C
#include <sys/ioctl.h>

// Connect ADDR to GRD.
// Setup to use ADC0 single ended

int fd;
// Note PCF8591 defaults to 0x48!
int asd_address = 0x48;
```

```
int16_t val;
uint8_t writeBuf[3];
uint8_t readBuf[2];
float myfloat;
const float VPS = 4.096 / 32768.0; // (volts/step)
int main() {
   FILE *fp1, *fp2;
   fp1 = fopen("/tmp/volt.txt","w+");
   fp2 = fopen("/tmp/digital.txt","w+");

   // open device on /dev/i2c-1 the default on Raspberry Pi B
   if ((fd = open("/dev/i2c-1", O_RDWR)) < 0) {
      printf("Error: Couldn't open device! %d\n", fd);
      exit (1);
   }

   // connect to ADS1115 as i2c slave
   if (ioctl(fd, I2C_SLAVE, asd_address) < 0) {
      printf("Error: Couldn't find device on address!\n");
      exit (1);
   }
   writeBuf[0] = 1; // config register is 1
   writeBuf[1] = 0b11000010; // 0xC2 single shot off
   // Note: bit 15=flag bit for single shot not used here

 writeBuf[2] = 0b11100101; // bits 7-0  0x85
   // Bits 7-5 data rate default to 100 for 128SPS
   // Bits 4-0  comparator functions see spec sheet.

   //to begin conversion
   if (write(fd, writeBuf, 3) != 3) {
      perror("Write to register 1");
      exit (1);
   }

   sleep(1);
     // set pointer to 0
   readBuf[0] = 0;
   if (write(fd, readBuf, 1) != 1) {
      perror("Write register select");
      exit(-1);
   }
   int count = 1;

   while (1)   {

 // read conversion register
      if (read(fd, readBuf, 2) != 2) {
         perror("Read conversion");
         exit(-1);
      }
      val = readBuf[0] << 8 | readBuf[1];
            if (val < 0)   val = 0;
      myfloat = val * VPS; // convert to voltage

      fprintf(fp1, "%d\n", (int)myfloat);
      fprintf(fp2, "%d\n", val);

      printf("Digital Value:  %d \t %4.3f volts.\n", val, myfloat);
            count++; // inc count
   }
   // power down ADC1115 ADC
   writeBuf[0] = 1;    // config register is 1
   writeBuf[1] = 0b11000011;                          // MSB
   writeBuf[2] = 0b11100101;                // LSB
   if (write(fd, writeBuf, 3) != 3) {
perror("Write to register 1");
      exit (1);
   }

   close(fd);

   return 0;
}
```

Code for FFT:

```
#include <stdio.h>
#include <math.h>
#include <koolplot.h>
struct Complex
{    double a;        //Real Part
   double b;        //Imaginary Part
}         X[5], U, W, T, Tmp;


void FFT(void)
{
   int M = 2;
   int N = pow(2, M);
   int i = 1, j = 1, k = 1;
   int LE = 0, LE1 = 0;
   int IP = 0;
   for (k = 1; k <= M; k++)
   {
      LE = pow(2, M + 1 - k);
      LE1 = LE / 2;

 U.a = 1.0;
      U.b = 0.0;
      W.a = cos(M_PI / (double)LE1);
      W.b = -sin(M_PI/ (double)LE1);

      for (j = 1; j <= LE1; j++)
      {
         for (i = j; i <= N; i = i + LE)
         {
            IP = i + LE1;
            T.a = X[i].a + X[IP].a;
            T.b = X[i].b + X[IP].b;
            Tmp.a = X[i].a - X[IP].a;
            Tmp.b = X[i].b - X[IP].b;
            X[IP].a = (Tmp.a * U.a) - (Tmp.b * U.b);
            X[IP].b = (Tmp.a * U.b) + (Tmp.b * U.a);
            X[i].a = T.a;
            X[i].b = T.b;
         }
         Tmp.a = (U.a * W.a) - (U.b * W.b);
         Tmp.b = (U.a * W.b) + (U.b * W.a);
         U.a = Tmp.a;
         U.b = Tmp.b;
      }
```

```
    }

    int NV2 = N / 2;
    int NM1 = N - 1;
  int K = 0;

    j = 1;
    for (i = 1; i <= NM1; i++)
    {
       if (i >= j) goto TAG25;
       T.a = X[j].a;
       T.b = X[j].b;

       X[j].a = X[i].a;
       X[j].b = X[i].b;
       X[i].a = T.a;
       X[i].b = T.b;
    TAG25:    K = NV2;
    TAG26:    if (K >= j) goto TAG30;
       j = j - K;
       K = K / 2;
       goto TAG26;
    TAG30:    j = j + K;
    }
}

int main(void)
{
    FILE *fp1, *fp2;
    fp1 = fopen("/tmp/volt.txt","r+"); //reads voltage values from
a //text file specific for voltage
    fp2 = fopen("/tmp/fft.txt","w+"); //writes/stores fft values

    FILE *gnuplot = popen("gnuplot","w");
    //library used to plot power spectrum directly after
//compilation of the code
    float arr[5] = {1.3, 2.8, 3.6, 2.1, 3.3};
    int i;
    for (i = 0; i < 5; i++)
    {
       X[i].a = arr[i];
       X[i].b = 0.0;
    }

    printf ("*********Before*********\n");
    for (i = 1; i <= 4; i++)
       printf ("X[%d]:real == %f  imaginary == %f\n", i, X[i].a,
X[i].b);
    FFT();

    fprintf(gnuplot, "plot '-'\n");

    printf ("\n\n**********After*********\n");
    for (i = 1; i <= 4; i++)
    {
       printf ("X[%d]:real == %f  imaginary == %f\n", i, X[i].a,
X[i].b);
       fprintf(gnuplot, "%g %g\n", X[i].a,X[i].b);
    }
    fprintf(gnuplot,"e\n");
    fflush(gnuplot);

    return 0;
}
```

Appendix B

List of Tables:

List of Figures: