

Designing compensation function and validating data through FFT

Prashant Shushilbhai Gandhi-013712361

Computer Engineering Department, College of Engineering

San Jose State University, San Jose, CA 94303

E-mail: prashantshushilbhai.gandhi@sjsu.edu

Abstract

This paper gives detail explanation of how to design a circuit to interface ADC module with raspberry pi 3 b+ and program raspberrypi such that it takes input from that ADC module and process that data so that final output will be without any errors. This paper explains how to convert digital output of ADC module to voltage and take samples of that data at sampling rate to build compensation function for ADC module so that we can reduce error which occurs at lower and higher voltage. Then we use FFT computation to validate our ADC data by computing power spectrum and based on that result we can find if there is any aliasing occurs on current sampling rate. The objective of this project to design compensation function and validate ADC output using FFT and power spectrum which is successfully achieved.

1. Introduction

This project uses Raspberrypi 3 b+ board which contains CPU ARM-Cortex A53 and it has clock cycle of 1.4 GHz. Raspberry pi 3 b+ does not have inbuilt ADC module. Due to that, to integrate sensors which give analog output with Raspberry Pi 3 b+, we need to connect ADC module externally. This project uses Adafruit MCP3008 10-bit ADC module with 8 channels to convert potentiometer output. Communication between ADC module and Raspberry Pi 3 b+ is done through SPI. At software side ADC's digital output is converted to appropriate voltage range using ADC resolution formula which this paper converts later. Due to random noise, measured voltage from multimeter at Potentiometer output and voltage after conversion in software will reflect some fractional error. To overcome this problem, compensation function is used to compensate that error which is done by taking samples of output at fixed frequency which is called sampling frequency. To validate that data and to check if aliasing is occurring or not we use FFT and plot power spectrum graph.

2. Methodology

Following section of this paper will explain objective of this paper and what technical challenges were occurred will implementing this project how to overcome from that problems.

2.1. Objectives and Technical Challenges

Objective of this paper is as below:

1. Build a ADC circuit which interface with Raspberry Pi 3b+ through SPI.
2. Convert digital output of ADC circuit to appropriate voltage range using formula.
3. Build compensation function from samples takes at sampling frequency to reduce error.
4. Validate data using FFT program and plot power spectrum to see if aliasing is occurring or not.

Technical Challenges:

1. Find SPI driver to interface with ADC module.
2. Measure difference between multimeter reading and software calculated reading and calculate compensation function.
3. Plot graph of power spectrum from FFT process.

2.2. Problem Formulation and Design

Design block diagram:

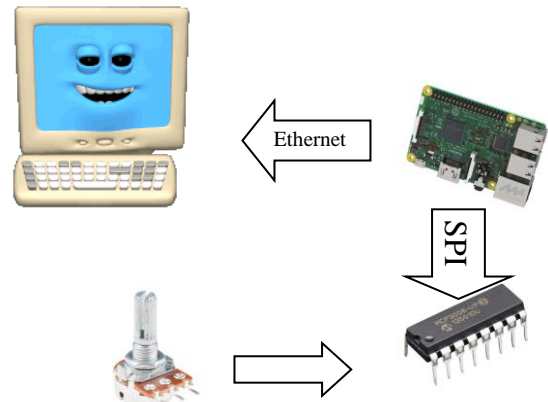


Figure 2.2.1 Design block diagram
Analog -to-Digital Conversion formula:

$$\frac{\text{Resolution of the ADC}}{\text{System Voltage}} = \frac{\text{ADC Reading}}{\text{Analog Voltage Measured}}$$

Compensation Formulation steps:

1. First take 10 readings from multimeter and corresponding digital converted values in software.
2. Do interpolation:

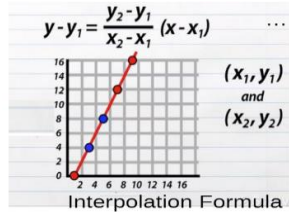


Figure 2.2.2 Interpolation formula and graph

Here, x-axis will be our voltage measured from multimeter and y-axis will be our digital raw reading. Result will be in terms of y and x. Then we apply following formula to calculate compensation function.

$$F(x) = p \cdot x + q \quad \dots\dots(1)$$

Here, from the result we can extract p and q of above formula:

Using that p and q compensation function is calculated by following formula.

$$g(x) = a \cdot g + b \cdot g, \quad \dots\dots(2)$$

where, $a \cdot g = -q$ and $b \cdot g = (a \cdot p)$.

Here, $a = 1024/3.3$ (1024 because 10-bit ADC)

$G(x)$ is our compensation function.

Data validation:

To validate our data, we use FFT and DFT process and plot power spectrum to check if aliasing is happening.

The Discrete Fourier Transform can be expressed as

$$F(n) = \sum_{k=0}^{N-1} f(k) e^{-j2\pi nk/N} \quad (n = 0 \dots N-1)$$

The relevant inverse Fourier Transform can be expressed as

$$f(k) = \frac{1}{N} \sum_{n=0}^{N-1} F(n) e^{j2\pi nk/N} \quad (k = 0, 1, 2, \dots, N-1)$$

3. Implementation

Implementation can be categorized in two parts,

1. Hardware part.
2. Software part.

3.1 Hardware Design

Hardware aspect of this project includes designing of ADC circuit using potentiometer. List of components are in the below table.

Table 3.1.1 Bill of Materials

SL.No	Components	Description	Notes
1.	Raspberry Pi 3 B+	CPU	Qty 1
2.	MCP3008	ADC Module	10-bit 8-Channels
3.	Potentiometer	10K Ohm	Qty 1
4.	Jumper wires	-	Qty 2

System block diagram:

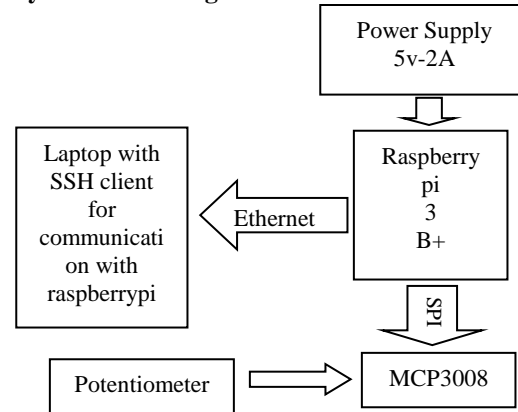


Figure 3.1.1 System block diagram

Raspberry pi 3 b+ is connected to ADC module through SPI protocol whose connection diagram is as below.

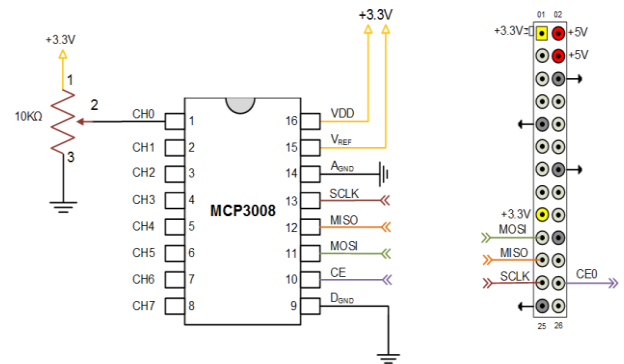


Figure 3.1.2 Raspberrypi and MCP3008 SPI connection Below is real hardware picture.

Table 3.1.2 Pin connection

Pin	Raspberry Pi	Pin	MCP3008
1	3V3	16	VDD
1	3V3	15	Vref
6	GND	14	Agnd
6	GND	9	Dgnd
23	SPI0_SCLK	13	SCLK
21	SPI0_MISO	12	MISO
19	SPI0_MOSI	11	MOSI
24	SPI0_CE0_N	10	CE
2	Potentiometer	1	CH0

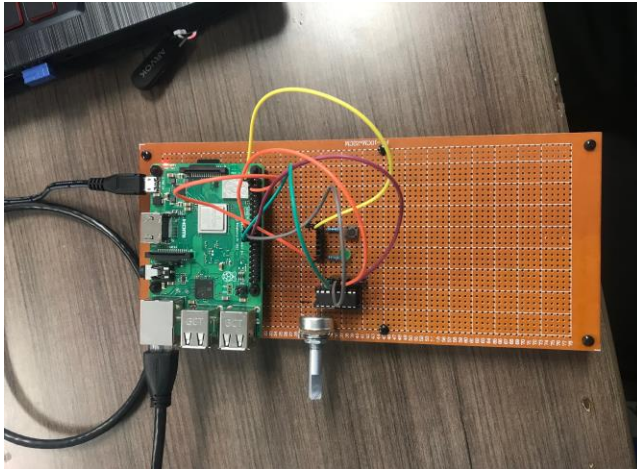


Figure 3.1.3 Real Hardware

3.2. Software Design

Below is the Algorithm, Flow chart of the software implementation.

3.2.1 Algorithm

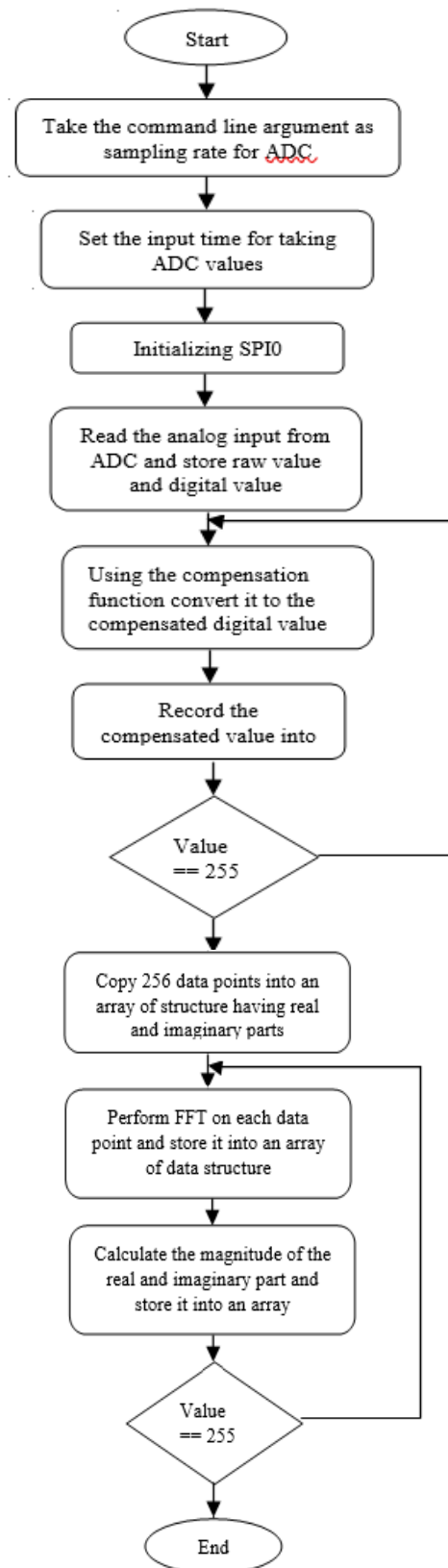
Algorithm for getting ADC output and build compensation function.

1. Take sampling rate from user to take samples of ADC output using command line argument. Using command line argument.
2. Initialize the Raspberry Pi 3 B+ SPI 0 Port.
3. Read ADC module's Channel 1 to get RAW data. Convert it such that its range become 0-3.3V.
4. Calculate compensation function using that data.
5. Repeat steps 3 and 4, introduce delay to take data in accordance of sampling rate.

Algorithm to validate above data using FFT and get power spectrum.

1. Get the 256 raw data from the file in which output is stored.
2. Feed this data to FFT.c file to calculate imaginary and real part
3. Calculate magnitude using that imaginary and real part.
4. After getting output of magnitude we store data in file and using excel we create plot of power spectrum.

3.2.2 Flow-chart



3.2.3 Pseudo Code:

ADC Compensation Function :

```
//Read channel-1 of MCP3008 to get raw data
raw_data = myAnalogRead(0,8,1);

//Convert it into voltage (0-3.3v)
uncompensated_value = ((3.3/1023)*raw_data);

//Apply compensation formula
g_x = (-3.495*uncompensated_value) + 1.032;

//Add that calculate value to raw data and compensate
value
//compensated value

new_raw_data = raw_data + g_x;
compensated_value = (3.3/1023)*new_raw_data;
```

4. Testing and Verification

Below are the pictures of Successful compilation of project, ADC 256-point output, FFT power spectrum.

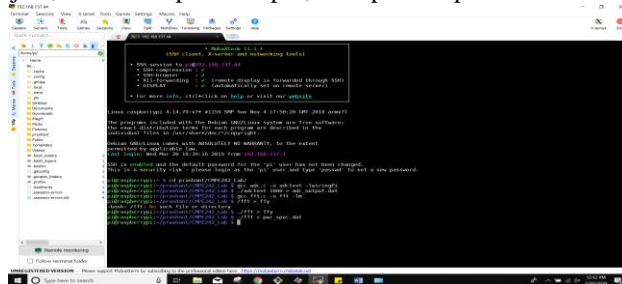


Figure 4.1 Successful compilation

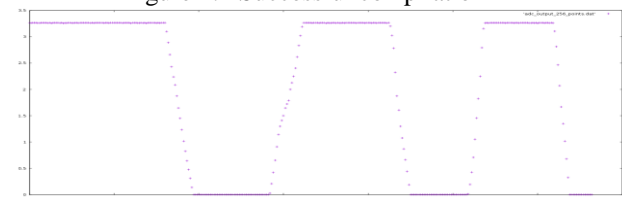


Figure 4.1 256-point ADC digital output

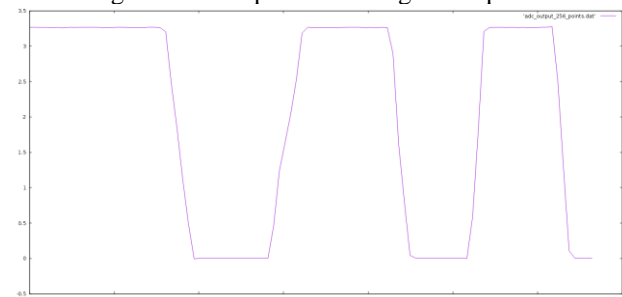


Figure 4.1 256-point ADC digital output connecting dots

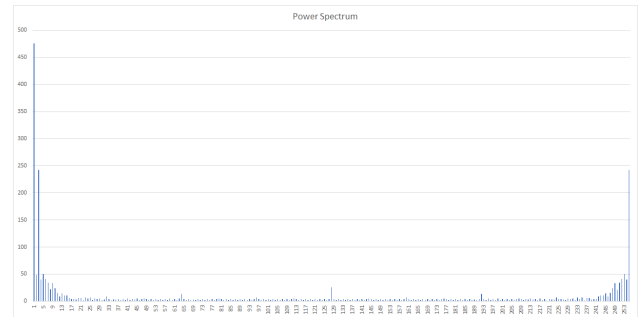


Figure 4.2 Power spectrum

5. Conclusion

Objective of this project was to design ADC circuit for Raspberry Pi 3 B+ module and implement program which can successfully convert digital output into appropriate voltage using compensation function and validate that data using FFT process and plot power spectrum graph for same which is successfully implemented, and project is getting output as expected.

6. Acknowledgement

I would like to express my gratitude and appreciation to our Professor Harry Li for his encouragement, valuable guidance and patient review. His constant suggestion on hardware design helped us to implement good hardware and write effective program to compensation function for ADC module output. Also, I would like to thank him for giving us such a valuable knowledge of how to program Raspberry pi 3 b+. At last, I would also like to thank my project partners for their valuable inputs.

7. References

- [1] <https://github.com/hualili/CMPE242-Embedded-Systems/tree/master/2019S>
- [2] <http://wiringpi.com/reference/spi-library/>
- [3] <https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/mcp3008>
- [4] <https://alvinalexander.com/technology/gnuplot-charts-graphs-examples>
- [5] <https://learn.sparkfun.com/tutorials/analog-to-digital-conversion/all>

8. Appendix

8.1 Source Code

```
adc.c
/* ////////////////////////////////////////
 * LAB-1 : ADC Compensation function and Data
Validation
 * Name : PRASHANT SHUSHILBHAI GANDHI
 * STD-ID : 013712361
 * Email-ID : prashantshushilbhai.gandhi@sjsu.edu
 * Date : 03/20/2019
```

```

* //////////////////////////////////////////////////
* //////////////////////////////////////////////////
* For GPIO and SPI driver we are using wiringPi's Driver
* Steps to compile:
* gcc adc.c -o adctest -lwiringPi
* To run file:
* ./adctest 10000 (Where 10000 is Sampling Rate)
* //////////////////////////////////////////////////
*/
#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#include <wiringPi.h>
#include <wiringPiSPI.h>

#define INPUT_ACQ_TIME    5000000
#define SPI_FREQ          1000000

static int setup ;

// Setting Up SPI
void spiSetup (int spi_Channel_Number)
{
    if ((setup = wiringPiSPISetup (spi_Channel_Number,
SPI_FREQ)) < 0)
    {
        fprintf (stderr, "Can't open the SPI bus: %s\n",
strerror (errno)) ;
        exit (EXIT_FAILURE) ;
    }
}

int myAnalogRead(int spi_Channel_Number,int
channel_Config,int analog_Channel)
{
    if(analog_Channel<0 || analog_Channel>7){
        return -1;
    }
    unsigned char buffer[3] = {1};
    buffer[1] = (channel_Config+analog_Channel) << 4;
    wiringPiSPIDataRW(spi_Channel_Number, buffer, 3);
    return ( (buffer[1] & 3) << 8 ) + buffer[2]; // Extract
last 10-bits
}

// This function is to generate Delay
void _delay(int numberOfSeconds)
{
    int milli_seconds = 1000 * numberOfSeconds;
    clock_t start_time = clock();

```

```

while (clock() < start_time + milli_seconds)
    ;
}

float compensation_function(void){
    float uncompensated_value = 0, compensated_value =
0, g_x = 0,
        new_raw_data = 0;
    int raw_data = 0;
    raw_data = myAnalogRead(0,8,1); // Initialize all
chaner, Take output from channel 1
    uncompensated_value = ((3.3/1023)*raw_data);
    g_x = (-3.495*uncompensated_value) + 1.032;
    new_raw_data = raw_data + g_x;
    compensated_value = (3.3/1023)*new_raw_data;
    return compensated_value;
}

int main (int argc, char *argv[])
{
    int int_sampling_rate=0;
    // command line argument will accept sampling rate
    if(argc < 2)
    {
        int_sampling_rate = 1000;
    }else{
        char *sampling_rate = argv[1];
        int_sampling_rate = atoi(sampling_rate);
    }
    printf("Sampling Rate given: %dHz\n",
int_sampling_rate);
    printf("Input acquisition time is %d seconds\n",
(INPUT_ACQ_TIME/1000000));
    //Initialization of SPI
    wiringPiSetup();
    spiSetup(0);

    int seconds = INPUT_ACQ_TIME;
    //Take input for 5 seconds
    clock_t start_time = clock();
    while(clock() < start_time + seconds){
        printf("%f\n", compensation_function());
        _delay(1000 / int_sampling_rate);
    }

    close(setup);
    return 0;
}

```

FFT.c

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define SIZE_OF_INPUT 256
#define TWO_TO_THE_POWER 8

```

```

struct Complex
{
    double a;    //Real Part
    double b;    //Imaginary Part
} X[SIZE_OF_INPUT], U, W, T, Tmp;

void FFT(void)
{
    int M = TWO_TO_THE_POWER;
    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].b;
                T.b = X[i].b +
                    X[IP].a -
                    Tmp.a -
                    Tmp.b;
                X[IP].a = (Tmp.a *
                    X[IP].b = (Tmp.a *
                        X[i].a = T.a;
                        X[i].b = T.b;
                    }
                Tmp.a = (U.a * W.a) - (U.b *
                    Tmp.b = (U.a * W.b) + (U.b *
                        W.b);
                        Tmp.b = (U.a * W.b) + (U.b *
                            W.a);
                            Tmp.a = Tmp.a;
                            Tmp.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 *in_file =
fopen("adc_output_256_points.dat", "r"); // read only

    // test for files not existing.
    if (in_file == NULL)
    {
        printf("Error! Could not open file\n");
        exit(-1); // must include stdlib.h
    }
    float ab[256];
    // write to file vs write to screen
    for(int j = 0; j < SIZE_OF_INPUT; j++){
        fscanf(in_file, "%f", &ab[j]); // write to file
    }

    int i;
    for (i = 0; i < SIZE_OF_INPUT; i++)
    {
        X[i].a = ab[i];
        X[i].b = 0.0;
    }

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

    printf ("\n\n*****After*****\n");
    for (i = 1; i <= SIZE_OF_INPUT - 1; i++)

```

```

        printf ("X[%d]:real == %f imaginary
== %f\n", i, X[i].a, X[i].b);

    for(i = 1; i <= SIZE_OF_INPUT - 1; i++){
        X[i].a = pow(X[i].a, 2);
        X[i].b = pow(X[i].b, 2);
        ab[i] = X[i].a + X[i].b;
        ab[i] = sqrt(ab[i]);
        printf("Squared value [%d] = %f\n", i,
ab[i]);
//        printf("%f\n", ab[i]);
    }
    return 0;
}

```