Calibration and Validation of ADC data using Compensation Function & FFT

Rishabh Sheth - 013708916
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
E-mail: rishabh.sheth@sjsu.edu

Abstract

This paper describes the design and integration of ADC with Embedded Linux Device. It also describes method used to calibrate ADC. To avoid aliasing, power spectrum is calculated to find correct sampling frequency. For the demonstration Raspberry Pi is interfaced with separate ADC IC. The objective of this lab is to get familiar with embedded Linux, make compensation function to calibrate ADC output and validate ADC value using FFT and power spectrum which successfully achieved.

1. Introduction

The lab uses Raspberry Pi 3 model B v1.2 as Embedded Linux Device. It has Broadcom's ARM Cortex A53 processor, 1GB RAM and Micro SD port for loading operating system and storing data. Raspberry has rich peripherals interface e.g. 40-pin extended GPIO, 4 USB2.0 ports, full size HDMI, Ethernet etc. It can run full Linux operating system. Board used in this lab runs Raspbian OS with the kernel version "Linux raspberrypi 4.14.98-v7+."

Raspberry don't have inbuilt ADC. So, to measure analog values microchip's MCP3008 IC is used in this lab. It is 8 channel 10-bit ADC with SPI interface. Raspberry Pi has 2 SPI interfaces, for the project ADC is connected to SPI0 of Raspberry Pi. To simulate analog input 10K potentiometer is used.

The final goal of the project is to measure analog value and process it on board. Also calibrate ADC using compensation formula to eliminate any error present in hardware and validate sampling frequency using FFT and power spectrum calculation of stored data. The technical challenges faced during the project were setting up Raspberry, making appropriate connections of the Raspberry Pi module to ADC chip, finding proper driver for SPI and GPIO. These were eventually solved by careful testing and debugging of the circuits and the code.

---

2. Methodology

This section gives a detailed description of the steps in designing the circuit and the challenges which were encountered during the same.

2.1. Objectives and Technical Challenges

The objective of the lab is to get familiar with embedded Linux board, connect analog input to board. Monitor ADC value at correct frequency and validate it.

The objective of the lab is as follows:

1. To select and setup appropriate embedded board, preferred Raspberry Pi or ARM 11
2. Build and flash Linux OS and test board by implementing basic function like LED and Switch.
3. Establishing a serial communication between board and the ADC chip via SPI communication protocol.
4. Store 10 different ADC value used it to find compensation function by comparing it with manual readings.
5. Validate sampling frequency by calculating FFT and plotting power spectrum.

The challenges encountered were as follows:

1. Setting up Raspberry Pi with Ethernet and setting up serial communication with laptop
2. Establishing communication between Raspberry Pi and ADC chip using SPI.
3. Clean soldering of the components on the board.

2.2. Problem Formulation and Design

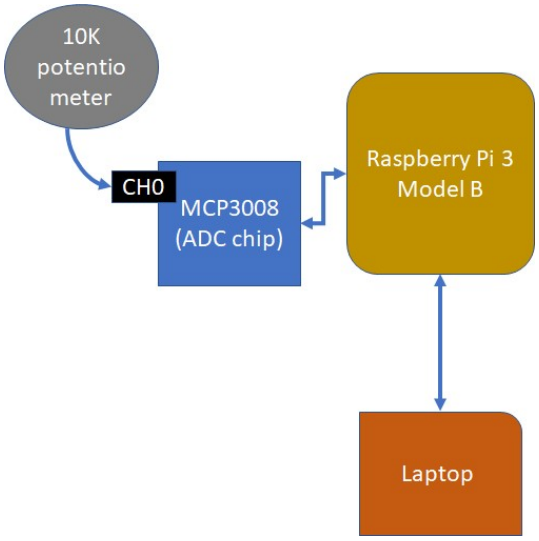


Figure 2.1: System Layout

First, official image OS available on Raspberry Pi website is flash on SD card. The Raspberry Pi is connected to laptop using Ethernet cable. Ethernet can be used to communicate with Raspberry via serial terminal or using GUI. For this project MobaXterm application is used for communication purpose. Simple GPIO-Switch program is used to test basic setup.

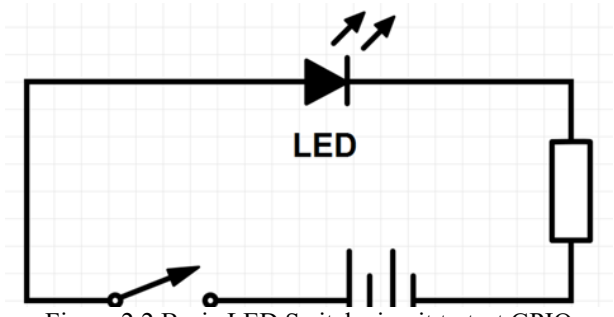


Figure 2.2 Basic LED Switch circuit to test GPIO

Compensation Function:

- Due to environment noise, hardware connection and limitation in ADC chip capability to measure exact analog value, there is high possibility of error in measured digital voltage
- The error in measuring system can be eliminated by adding offset, which can be find using compensation function
- Initially we took 10 analog input using MCP3008 ADC chip along with manual measurement using high precision Digital Multimer
- We plot measured values from the ADC onto a graph and use the formula mentioned below to calculate compensation function:

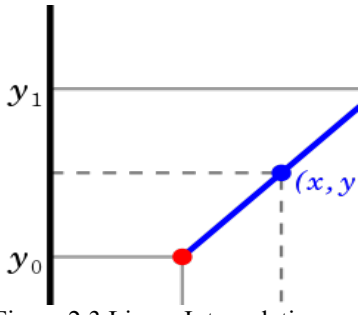


Figure 2.3 Linear Interpolation

Slop can be fined using

$$y - y0 = (y1 - y0) * (x - x0) / (x1 - x0)$$

$$\mathbf{f(x) + g(x) = ax + b;}$$
$$\mathbf{b\_g = -q; a\_g = (a - p)}$$

Here,
f(x) = Actual ADC function;
g(x) = Compensation function;
b = 0; a = Slope for the ADC data characteristic curve.

After putting actual values measured into function described above, following function obtained which is actual compensation function

g_x = (-3.485*row_digital_value) + 1.023;

Power Spectrum:

- Selecting connect sampling frequency is important part of measuring analog input. Wrong sampling frequency can lead to error in measured input.

The Discrete Fourier Transform can be expresse

$$\mathbf{F}(n) = \sum_{k=0}^{N-1} f(k)\, e^{-j2\pi nk/N} \qquad (n = 0..1 \quad :N-1$$

The relevant inverse Fourier Transform can be e

$$\frac{1}{N} \sum_{N-1}$$

- FFT of measured values can be used to plot power spectrum
- Here we measured 256 points and plot its power spectrum by using FFT program

3. Implementation

The design for this lab can be explained in two phases:

1. Hardware Design
2. Software Design

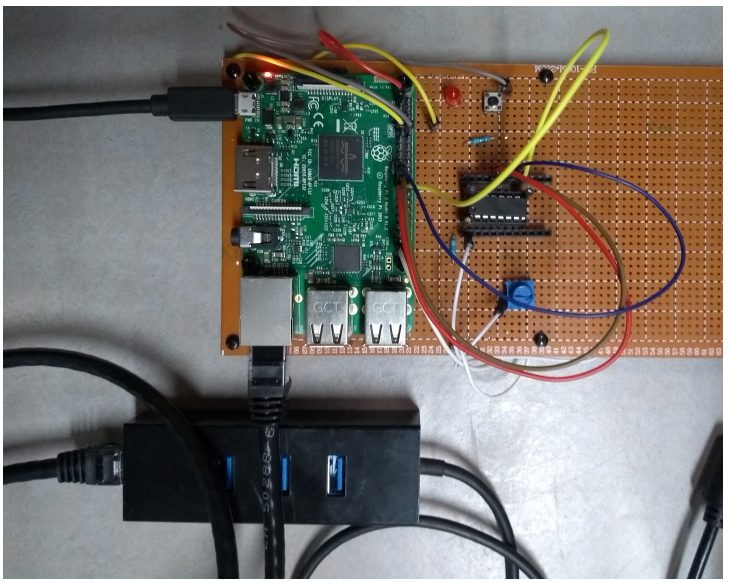


Figure 3.1 Actual Hardware implementation

3.1. Hardware Design

This project includes selecting most suitable components to make own hardware. Raspberry Pi is fixed on Prototype board with help of standoffs. All components are soldered on prototype board for better robustness.

3.1.1 Bill of Materials

The list of components used for the project are listed below:

Sr. no.	Component	Description	Price $
1	Raspberry Pi 3 B v1.2	Embedded Linux Board	35.0
2	Prototype Circuit Board Breadboard	Prototype Circuit Board Breadboard	3.28
3	MCP3008	ADC IC	6.51
4	Potentiometer	Analog input	0.99
5	Resistors (1k Ω)	To protect GPIO	0.99
6	LEDs	Test GPIO	0.99
7	Header strips (Male and Female)	Connect IC	1.99
8	DC power adapter (5v, 2a)	To provide high power to pi	4.99
9	Ethernet Adapter	To connect with USB Type C	19.99
10	Ethernet Cable	To connect Pi with Laptop	4.49

Table 3.1 Bill of Materials

3.1.2 Pin connection

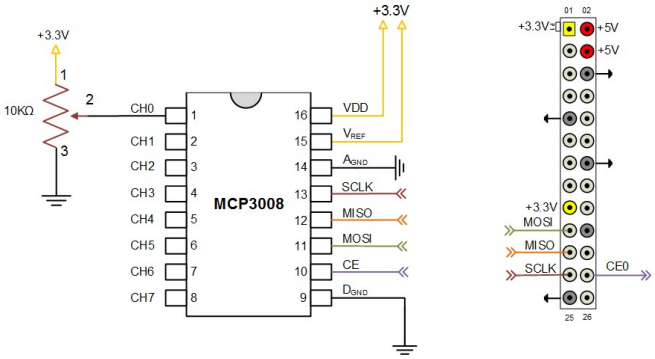


Figure 3.2 Pin connection of Raspberry Pi with ADC

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

3.2. Software Design

Wiring Pi GPIO Interface library is used for the Raspberry Pi. This library provided APIs to connect MCP3008 ADC chip with Raspberry using SPI communication protocol.

3.2.1. Algorithm and Pseudo implementation

Algorithm to build compensation function.

STEP 1: Indianize SPI0 of Raspberry Pi
STEP 2: Read row ADC data from MCP3008 chip
STEP 3: Convert row value into voltage using flowing equation

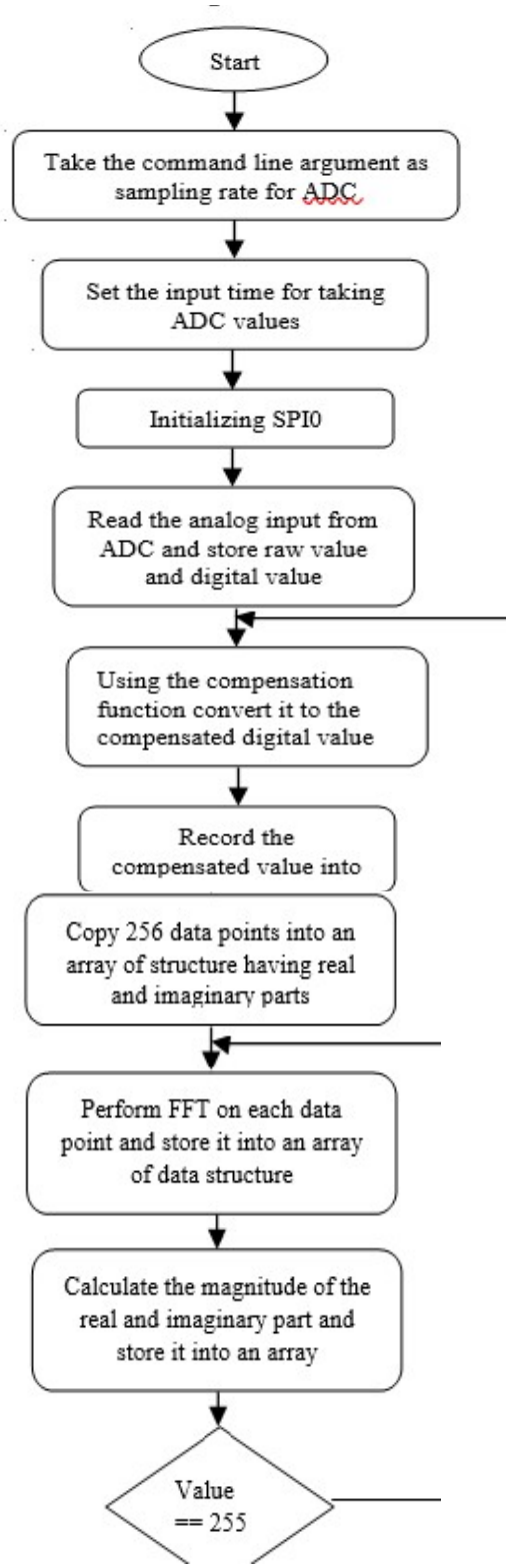
$$\frac{Resolution\ of\ the\ ADC}{System\ Voltage} = \frac{ADC\ Readi}{Analog\ Voltage\ M}$$

STEP 4: Take multiple sample and calculate compensation function described in section 1

Algorithm plot power spectrum.

STEP 1: Get user input for custom sampling frequency
STEP 2: Indianize SPI0 of Raspberry Pi
STEP 3: Read row ADC data from MCP3008 chip
STEP 4: Convert row value into voltage
STEP 5: Repeat above step to take 256 samples
STEP 6: Store sample data in file
STEP 7: Feed this data to FFT.c file to calculate imaginary and real part
STEP 8: Calculate magnitude using that imaginary and real part.
STEP 9: After getting output of magnitude we store data in file
STEP 10: Using excel or GNU plot, plot the power spectrum

3.2.2 Flowchart



4. Testing and Verification

We rigorously tested code multiple times and output was as expected. Below are the pictures of Successful compilation of project, ADC 256-point output, FFT power spectrum.

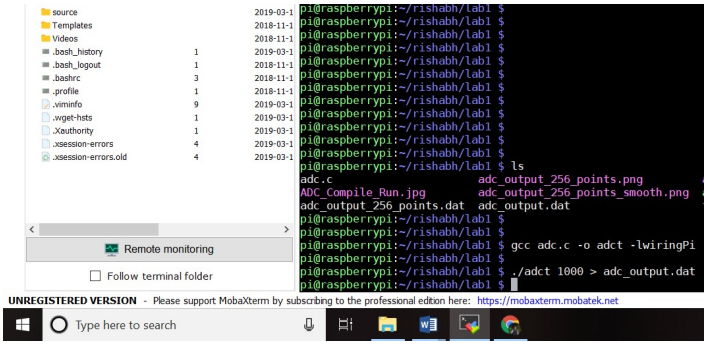


Figure 4.1 successful compilation of source code

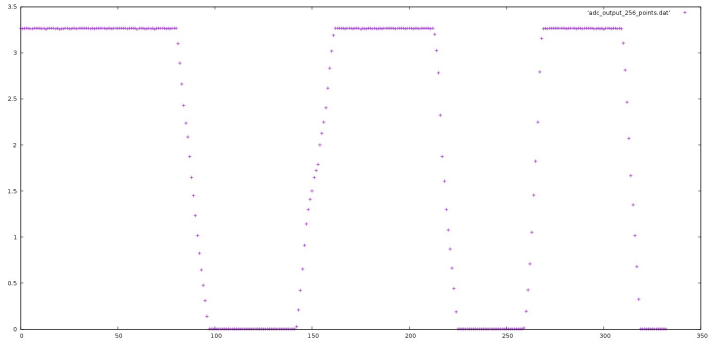


Figure 4.2 ADC digital output (256-point ADC)

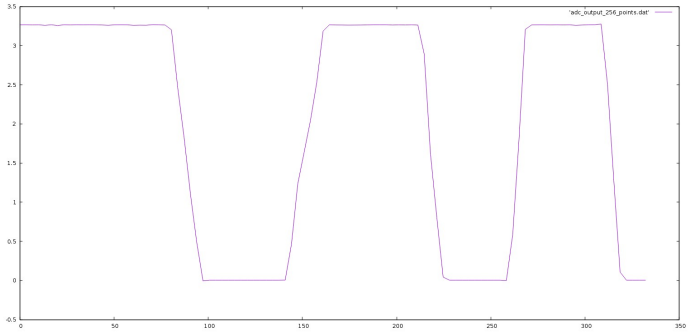


Figure 4.3 ADC output using line graph

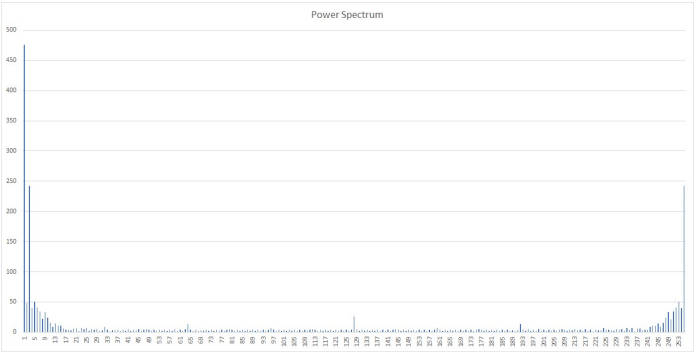


Figure 4.4 Power Spectrum of ADC data

5. Conclusion

The objective of this lab to get familiar with embedded hardware design by connecting ADC to embedded board. We successfully able to measure ADC value using raspberry pi. We calibrated ADC using compensation function and validated sampling frequency using FFT and plotting power spectrum of measured ADC value.

6. Acknowledgement

I would like to thank Dr. Harry Li for providing us detailed knowledge of embedded hardware design. Concept learned during class like ADC compensation, FFT and power spectrum is vary useful to make quality embedded hardware. Material available on Dr. Harry Li's GitHub helped me lot to make this project working. I would also like to thank my project teammate to help me during this project.

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

8. Appendix

```
/* ////////////////////////////////////////////////////////
 * LAB-1    : ADC Compensation function and Data Validation
 * Name     : Rishabh Sheth
 * Email-ID : rishabh.sheth@sjsu.edu
 * Date     : 03/20/2019
 * ////////////////////////////////////////////////////////
```

Source Code:
ADC.c

```
#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 spi0_Fd ;

void my_spiSetup (int spiChannel){
   if ((spi0_Fd = wiringPiSPISetup (spiChannel, SPI_FREQ)) < 0){
      fprintf (stderr, "Can't open the SPI bus: %s\n", strerror (errno)) ;
      exit (EXIT_FAILURE) ;
   }
}
int myAnalogRead(int spiChannel,int channelConfig,int analogChannel){
   if(analogChannel<0 || analogChannel>7){
      return -1;
         }
   unsigned char buffer[3] = {1};
   buffer[1] = (channelConfig+analogChannel) << 4;
   wiringPiSPIDataRW(spiChannel, buffer, 3);
   return ( (buffer[1] & 3 ) << 8 ) + buffer[2];
}

void _delay(int number_of_seconds) {
   int milli_seconds = 1000 * number_of_seconds;
   clock_t start_time = clock();
   while (clock() < start_time + milli_seconds);
}

float compensate_value(void){
   float uncompensated_digital_value = 0,
                  compensated_digital_value = 0,
                  g_x = 0,
                  new_raw_value = 0;
   int raw_value = 0;
   raw_value = AnalogRead(0,8,0);
   uncompensated_digital_value = ((3.3/1023)*raw_value);
   g_x = (-3.495*uncompensated_digital_value) + 1.032;
   new_raw_value = raw_value + g_x;
   compensated_digital_value = (3.3/1023)*new_raw_value;
   return compensated_digital_value;
}

int main (int agrc, char *argv[]){
   // command line argument will accept sampling rate
   char *char_sampling_rate = argv[1];
   int int_sampling_rate = atoi(char_sampling_rate);
   printf("Sampling Rate: %dHz\n", int_sampling_rate);
```

```
    printf("Input time is %d seconds\n",
(INPUT_ACQ_TIME/1000000));
    //Initialization of SPI
    wiringPiSetup();
    my_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", compensate_value());
                        _delay(1000 / int_sampling_rate);
    }
    close(spi0_Fd);
    return 0;}
```

Note: For full code and plots refer to https://github.com/rishabh00/CMPE242_Lab