ADC Interfacing and Validation with Raspberry Pi 4B

Amiraj Nigam, 01451202
Electrical Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
Email: amiraj.nigam@sjsu.edu

Abstract

This objective of this project is to study different modules in Raspberry Pi 4. Educate students about Analog to digital converter (ADC) by interfacing a Raspberry Pi 4 and an analog to digital convertor (ADC) to understand the working of a ADC. This is done by connnecting a potentiometer to the ADC module and interface that with a Raspberry Pi 4 and vary values. In this project we also get an opportunity to design a power supply circuit and build a prototype module on a general-purpose board. Using ADS1015 we will calculate the delta error, accumulative error and compensation function. Here we get an opportunity to use the I2C protocol to interface ADC with Raspberry Pi 4 and analyze the range of digital output.

1. Introduction

The idea of this project is to design a prototype board and fetch digital output from an analog to digital converter by varying the 10K potentiometer for fixed range of 0V to 3.3V to obtain corresponding digital values. A python code is compiled to facilitate this experimental value.

Hardware:

Component	Quantity
Raspberry Pi 4B	1
ADS1015(ADC)	1
Wire wrapping board	1
Potentiometer	1
Connecting wire	8-9

Software:

Tools	Purpose
Raspbian OS	To run Raspberry pi 4 board
VNC and Putty	To connect remotely to Rpi
Python 3	Implement the design
I2C	Communication between
	ADC and Rpi

2. Methodology

Boot the SD 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:

Source	Destination
ADS1015 VDD	Raspberry Pi 3.3V (pin 1)
ADS1015 GND	Raspberry Pi GND (pin 9)
ADS1015 SCL	Raspberry Pi SCL (pin 5)
ADS1015 SDA	Raspberry Pi SDA (pin 3)
Potentiometer VCC	3.3V of RPI
Potentiometer VCC	GND of RPI
Potentiometer VCC	Channel 0 (ADS1015)

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.

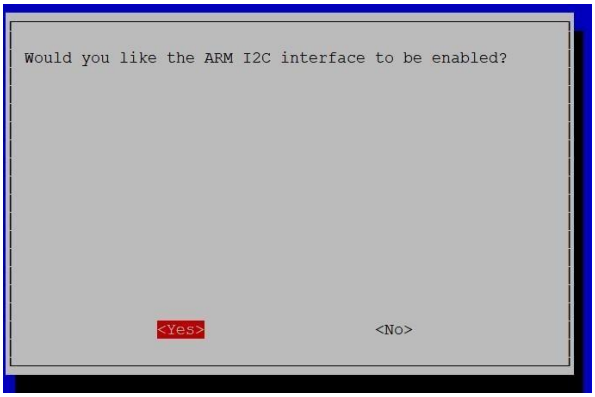


Figure 1. I2C configuration

2.1. Inter Integrated Circuit (I2C)

The below figure shows an example of devices connected on a I2C bus.

There are two pins on a I2C bus, they are as follows:

- SCL: Serial clock pin
- SDA: Serial data pin

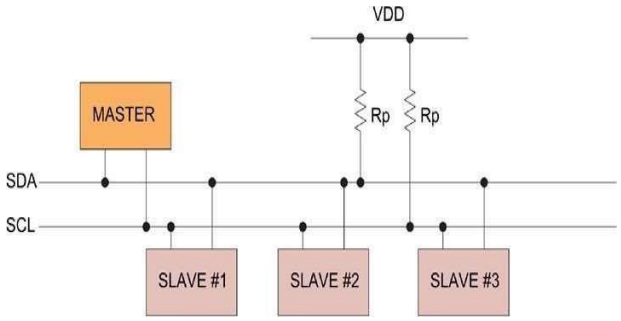


Figure 2. I2C working principle

2.4 Pin Diagram for the components

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.

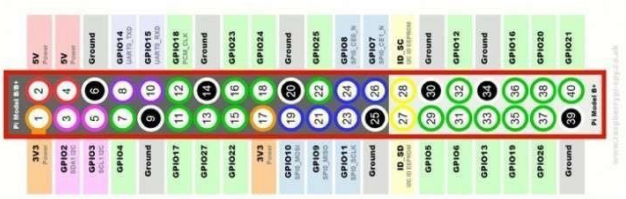


Figure3. Pin configuration for GPIO Interface

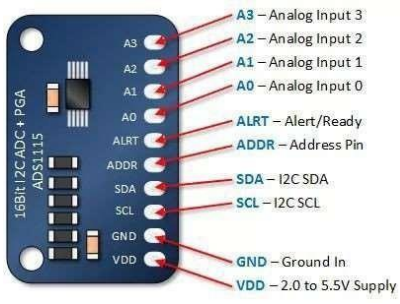


Figure 4. Pin configuration of ADC ADS1015

The ADS1015 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 operates 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 4.

Figure 5. Pin configuration of 10K Potentiometer

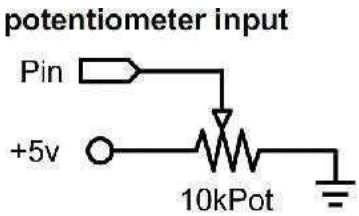


Figure 6. Pin- out 10K Potentiometer

2.3 ADC Connections for Calibrations

Connections are made for the calibration of the ADC and the Data validation via variations in potentiometer. The clock line is usually controlled by the Master with the exception that the slave may pull it low to indicate to the master that it is not ready to send data.

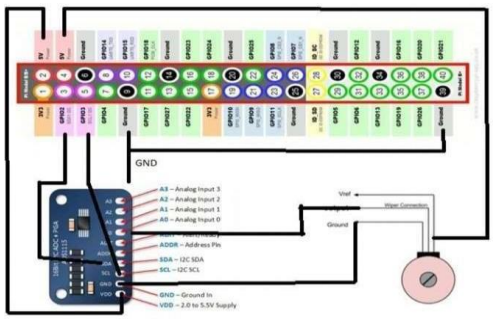


Figure 7. Circuit Connection

The data line is bi-directional and is controlled by the Master while sending data, and by the slave when it sends data back after a repeat-start condition described below.

2.4 Data Collection

Fig 7 shows three registers in the ADS1015. The first is an 8-bit "pointer" register that directs data to either the 16-bit read-only conversion register (0) or the 12-bit read/write config register (1). While the program below goes into all the config register bits. Bits D11-D9 determine the internal amp gain - it sets the upper voltage limit of the ADCs. If one is operating the ADS1015 at 3.3 volts one must select 2.048V or "010". With a Vmax of 3.3 V the voltage per step (VPS) at 12 bit (2^12) is: VPS = 3.3 /2048 = 161 uV per step.

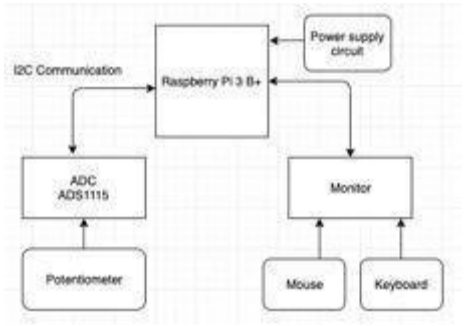


Figure 8. Circuit Schematic

D8 is the mode select bit. A "0" activates the continuous mode where once written to one could simply read the conversion register with no further writes to the config register.

A "1" (default) in D8 is the one-shot mode. In that mode the ADS1015 does a conversion after the write then powers down to low current mode. Bit Dx` serves as a ready flag and when set data is ready.
The following code snippet illustrates this:

```
value = adc.get_last_result()
```

3. 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 its characterization.

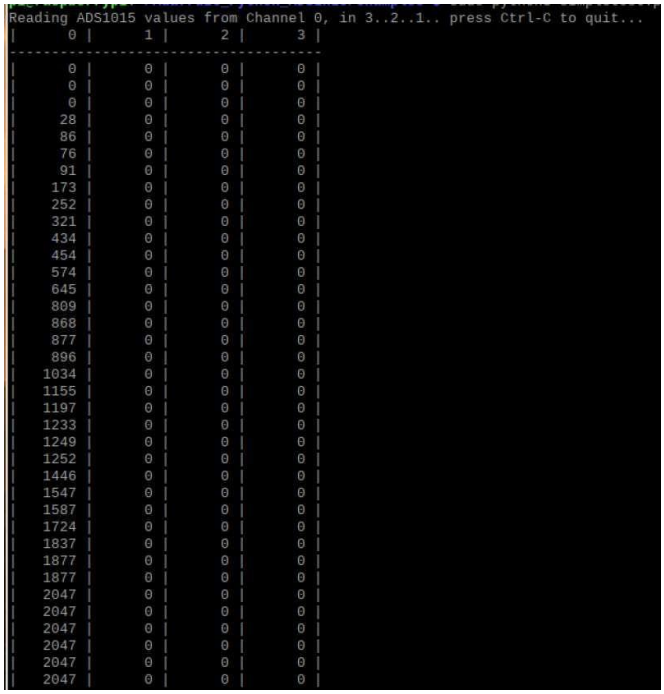


Figure 9. Output

Here I have shown the prototype board for this project implementation.

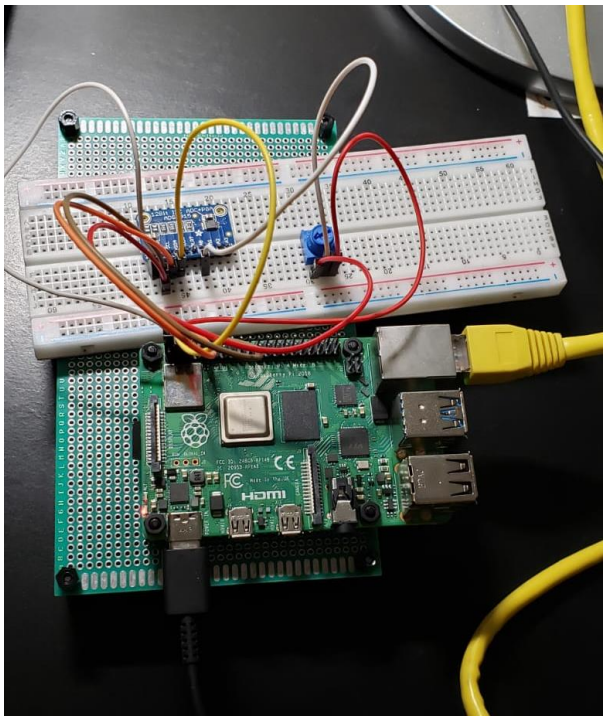


Figure 10. Prototype Board

General Flow

1. Install Raspbian OS as the environment for utilization. this can be done by booting a SD card by downloading noobs image from the official website on the SD in a bootable format.
2. Initiate the raspberry pi with the required power supply designed.
3. Make connections as per the circuit diagram.
4. 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.
5. Check the potentiometer to get zero voltage as the initial settings.
6. Run the Python-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.
7. Note down at least 10 readings between 0v to 3.3v with the corresponding readings in a tabular column and plot them
8. Calculate the delta error and accumulative error and plot the compensation function

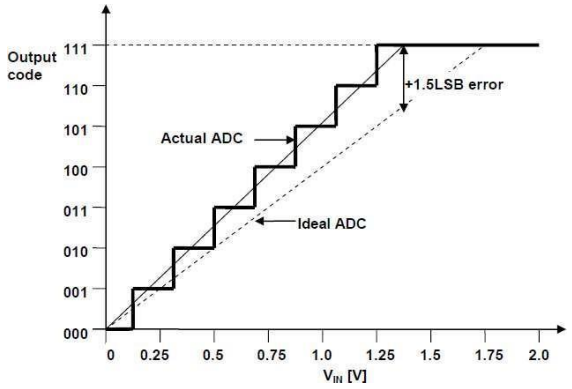


Figure 10. ADC Gain Error

4. Testing and Verification

Output of the ADC: The following data is obtained for a different potentiometer value. The ADC value corresponds to certain analog value measured at the variable terminal of potentiometer.

5. Conclusion

This project explored the interfacing of the Raspberry Pi 4 with an Analog to digital converter to observe digital values for respective analog voltage. We calculated the delta and cumulative error.

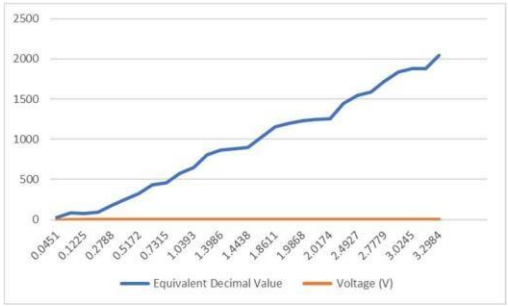
6. References

[1] H. Li, "Author Guidelines for CMPE 146/242 Project Report", Lecture Notes of CMPE 146/242, Computer Engineering Department, College of Engineering, San Jose State University, March 6, 2006, pp. 1.

[2]Adafruit Github for sample code to interface ADS1015 https://github.com/adafruit/Adafruit_Python_ADS1x15/ bl ob/master/examples/simpletest.py

[3]Adafruit Github https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/ads1015-slash-ads101

[4] Raspbian Link https://www.raspberrypi.org/downloads/raspbian/



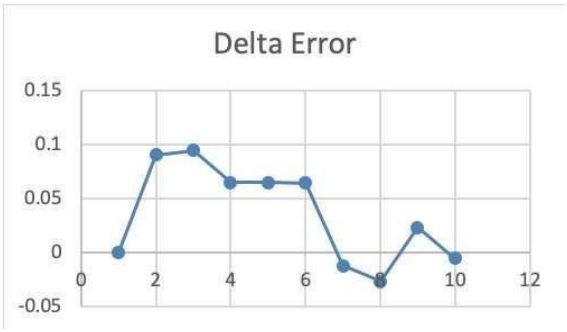


Figure 11. Delta Error

7. Appendix B

```
import time

import Adafruit_ADS1x15

# create an ADS1015 ADC (12-bit) instance. adc =
Adafruit_ADS1x15.ADS1015()


# Choose a gain of 1 for reading voltages from 0 to
4.09V.
# Or pick a different gain to change the
range of voltages that are read:
#    - 2/3  =  +/-6.144V
#    -      1 = +/-4.096V
#    -      2 = +/-2.048V
#    -      4 = +/-1.024V
#    -      8 = +/-0.512V
#    -     16  = +/-0.256V
# See table 3 in the ADS1015/ADS1115 datasheet
for more info on gain.

GAIN = 1

print('Reading ADS1x15 values, press Ctrl- C to quit...')

print('| {0:>6} | {1:>6} | {2:>6} | {3:>6}
|'.format(*range(4)))
print('-' * 37)

# Main Program starts from here while
True:

        # Read all the ADC channel values in a
list.
        values =  [0]*4
        for i in range(1):


                # Read the specified ADC channel using the
previously set gain value.
                        values[i] = adc.read_adc(i,
gain=GAIN)
        # Print the ADC values.
        print('| {0:>6} | {1:>6} | {2:>6} |
{3:>6} |'.format(*values))
        # Pause for half a second.
        time.sleep(0.5)
```

Appendix B

List Of Tables

List Of Figures