Unmanned Weighbridge

Major Project Report

Submitted in partial fulfillment of the requirement
for the degree of

Master of Technology
in
Electronics and Communication Engineering
(Embedded Systems)

By
Vyas Ankitkumar R
12MECE28

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382481
May 2014

Unmanned Weighbridge

Major Project Report

Submitted in partial fulfillment of the requirement
for the degree of

Master of Technology
in
Electronics and Communication Engineering
(Embedded Systems)

By
Vyas Ankitkumar R
12MECE28

Under the guidance of

External Project Guide:
Mr. H.A.Shabhai
General Manager, Cement Grinding Unit,
JK Lakshmi Cement,
Kalol, Dist: Gandhinagar.

Internal Project Guide:
Prof. Dhaval Shah
Assistant Professor, (EC Dept.),
Institute of Technology,
Nirma University, Ahmedabad.

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382481

May 2014

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems, at Nirma University and has not been submitted elsewhere for a degree.

b. Due acknowledgment has been made in the text to all other material used.

Vyas Ankitkumar R

Certificate

This is to certify that the Major Project entitled **"Unmanned Weighbridge"** submitted by **Vyas Ankitkumar R (12MECE28)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems of Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Prof. Dhaval Shah
Guide

Dr. N.P. Gajjar
Program Coordinator

Dr. P.N.Tekwani
Head of EE Dept

Dr. K. Kotecha
Director, IT

Acknowledgments

With immense pleasure, I would like to present this report on the dissertation work related to "Unmanned Weighbridge". I would like to express my gratitude and sincere thanks to **Dr. P.N.Tekwani**, Head of Electrical Engineering Department, **Dr. D. K. Kothari**, Section Head of Electronics and Communication Engineering program and **Dr. Nagendra Gajjar**, PG Coordinate of M.Tech, Embedded System for allowing me to undertake this thesis work and for his guidelines during the review process.

I am deeply indebted to my thesis supervisors **Prof. Dhaval Shah,** Assistant professor, Institute of technology, Nirma University, and **Mr. H.A.Shabhai**, General Manager, Kalol grinding unit, JK Lakshmi Cement, for their constant guidance and motivation. I also wish to thank **Mr. Naveen Sharma** (VP, Kalol grinding unit, JKLCL), **Mr. Deepak Sharma** (HOD IT, JKLCL), and **Mr. Arvind Rajdan** (sr. engg. JKLCL). Without their experience and insights, it would have been very difficult to do quality work.

I wish to thank my friends of my class for their delightful company which kept me in good humor throughout the year.

Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to complete the degree program successfully.

- VYAS ANKITKUMAR R
12MECE28

Abstract

Unmanned Weighbridge is a conception of a system for measuring weight of dispatched or purchased material in industries, without human intervention. An empty truck is being weighted on this weighbridge, and the measurements are called tare weight. When it is loaded and again being weighted, it gives a gross weight. The difference of gross weight and tare weight is net weight, which is the amount of the material loaded onto the truck. The process of weighment includes human intervention for entry into data base. As if the human intervention will be removed, the speed up of weighment process can be made up, no longer be bounded with human intervention.

The measurement must be done in real time, and accurate billing information must be generated from the weight difference. And same phenomenon must be used in unloading also. To implement this system, authorization of truck, first and second weight with alignment of Centre of Gravity (CG), transaction with data base, and intermediate data conversion are the expects to be covered. Raspberry Pi as a target board is used to make this data communication and conversion possible. Light weight Linux kernel wheezy, is used to accomplish this functionality, with python libraries to implement SPI, I2C and UART communication with peripheral devices.

Abbreviation Notation and Nomenclature

ADC ..... Analog-to-Digital Converter
ARM ..... Advanced Risk Machine
CG ..... Center of Gravity
ERP ..... Enterprise Resource Planning
GPIO ..... General Purpose Input Output
I2C ..... Inter-Integrated Circuit Communication
MCU ..... Micro-Controller Unit
SAP ..... System Application and Product
SPI ..... Serial Peripheral Interface
OS ..... Operating System
RPi ..... Raspberry Pi
UART ..... Universal Asynchronous Receive Transmit
USB ..... Universal Synchronous Bus

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The demand of automation in each field is on a mark to reduce the mistakes, made by humans. In order to establish an error prone environment for any system, a high degree of precision in automation is required. One can not expect such higher precision from human and hence sometime an unmanned system is being employed. The same phenomenon is used to generate a data base entry with no errors, for loaded and unloaded truck, is called Unmanned Weighbridge

Unmanned Weighbridge is a system developed on SOC as a replacement of a conventional weighment system which is a system with a personnel computer operated by a human. Authorization of truck, alignment of truck on weighbridge, weight difference, billing and many other issues must be addressed to make a complete and meaningful weighment. And for all of them, high degree of integration is indeed needed. But the ultimate goal is to achieve simplicity at an end user side, with higher degree of complexity at designer side, to make the system more user friendly.

1.1 Motivation

The system, being used till these days for weighment, is a combination of the devices working on RS232 UART protocol with COM PORT given at the I/O panel of a personnel computer and a PC as well. All transactions are under the control of an operator. Here an operator is the one, who takes care of authorization, alignment of truck, as well as an entry in database as the data available on COM PORT. A cabin is provided to accommodate this personnel computer for total transaction with data base. If a desire to work with higher speed is there, at that time the speed is bounded by the skill level of the operator. To achieve higher degree of speed with best precisions, implementation of something where the speed of operation does not depend on the human by any means must be done, and that is the motivation to automate a weighbridge.
Another matters of consideration are, the access to the Enterprise Resource Planning

(ERP) and System Application and Product (SAP) and the space utilization in an industry. ERP and SAP are highly confidential programs and have to be operated with proper authorization. A security is employed to the use of this database by accessing them with particular System On Chip (SOC) to reduce the timing requirement of authentication and granting access permission from server. As well with that by achieving good speed, reduction in the queue of the truck over the weighbridge obtained, and that may result in good area utilization within the plant. These are the main areas as causes of a motivation to develop such unmanned system.

1.2 Project Definition and Overview

Project definition is , **To implement an automatic weighing system for trucks, so the human intervention can be reduced, and a need of a dedicated operator can be eliminated.** Automation of weighbridge consists of main four actions.

a. Authentication

b. Alignment

c. Weighment

d. Billing information generation

1.2.1 Authentication

Authentication of a truck is done by providing a Radio Frequency (RF) ID card. A document for this truck is generated in SAP against its RFID number and onwards all the entries regarding to the weighment will be entered into it. Whenever a truck comes on the weighbridge, a pre-generated document is being selected from the database entry according to the number generated from the RF ID card reader. If document is not found, it means the truck is unauthorized and no weighment will be allowed for it.

1.2.2 Alignment

In law, the weighbridge operator is responsible for ensuring that, the weights obtained from the weighbridge are correct. In practice, he has to check that the vehicle is correctly positioned on the platform. The wagon driver will have no way of knowing whether or not the truck is properly alligned, without stopping and getting down from his cab and hence, it is a requirement that the allignment informatin is generated automatically, for the driver in order to an ease of placing truck on

platform. If truck is not aligned on the weighbridge properly, wrong weight may be entered into the database. To avoid this scenario, we prefer a CG based concept. For every movement of truck on the weighbridge, readings of the weight from all the load cells are taken and from those weights, current CG of the truck is found out by the SOC.

$$CG(x) = \frac{\Sigma(wi * xi)}{\Sigma wi} \tag{1.1}$$

$$CG(y) = \frac{\Sigma(wi * yi)}{\Sigma wi} \tag{1.2}$$

Where
C.G. (x) = x coordinate of Center of Gravity of truck with respect to a common reference point.
C.G. (y) = y coordinate of Center of Gravity of truck with respect to a common reference point.
Xi = x coordinate of ith load cell with respect to the common reference point.
Yi = y coordinate of ith load cell with respect to the common reference point.
wi = weight as an output of ith load cell.

For every movement of the truck on weighbridge, weight from all load cells are taken as input parameter to this function and calculation of the CG of that truck is calculated on SOC. A tolerance range is pre defined in program, as if whenever CG of truck lies in that range, the truck is allowed for weighment.

As if CG of truck does not lie in the range of tolerance, lighting and audio information are provided to driver, to move the truck accordingly on to weighbridge so it can lie in the tolerance limit with the CG of weighbridge. After proper alignment of truck, system is ready to capture the weight of the truck and make an entry in data base for the same.

1.2.3 Weighment

Here weighment is the process of measuring weight of truck. As truck comes on weighbridge load cells start to show the weight applied on them by the truck. These signals are converted into digital signals, and given to the SOC for various calculations. As if all the parameters are proper then the weight of the truck is being measured as per the equation below.

$$W = \Sigma wi \tag{1.3}$$

Where
W = weight of the truck
Wi = weight measured on ith load cell

Once an authorization and an alignment is done, weights on the load cells are given as input parameter of this function and the output is stored as total weight of the truck. This computation is also done in SOC. Then, this weight is being uploaded into the data base of the document of that particular truck, which has been chosen at the time of authorization.

1.2.4 Billing information generation

Billing information is generated from the amount of material being loaded or unloaded from the truck and hence a second weighment is indeed necessary. From the difference of weight as gross weight, entry is added to the document and billing information is generated. An absolute function is used, so we can use the same process for loading as well as unloading of the truck. In both cases weight difference will be observed positive due to absolute function. After billing and payment information generation all data as a package, transmitted to SAP server and then, RF ID card is made free incase to assign to another truck.

1.3 Objectives

The general objective of this research work is to design and construct a Raspberry pi SOC based electronic weigh bridge. The specific objectives are:

a. Mount the encapsulated 8 double ended shear beam load cells.

b. Design and build a signal conditioning and amplification circuitry for the load cell output.

c. Develop a program for interfacing the 12-bit ADC to the Raspberry Pi for implementing the analog-to-digital conversion.

d. Develop a program for Seven Segment Display, and Lighting Control.

e. Develop a header file, for the system using Python for calculation of C.G.

f. Calibrate the system.

g. Interfacing with SAP

h. Take measurements and C.G. records using the balance.

1.4 Key Features

a. Data accuracy with pre-programmed vehicle data using RFID card readers.

b. Improved facility control and security with traffic barriers, traffic lights, surveillance cameras etc.

c. Information such us operational analysis, billing and inventory control.

d. Weigh Terminal with RFID card reader for accurate and quick data input.

e. Exit & entry barriers - for security and arranging vehicle traffic.

1.5 Significance

The Raspberry Pi based unmanned weighbridge, in this research work, is for measuring and displaying mass of loaded or unloaded truck on screen. This research work utilizes the technique of interfacing a 12-bit ADC with SPI (Serial Peripheral Interface) protocol. The system is able to sense, measure and display mass ranging from 0 to 32.76 metric ton (32760 Kg). Actual capacity for the mass weighment on this platform is of 36.28 metric ton, but we range the limits to get a sensitivity of 1Kg/lsb from ADC.

1.6 Contributions and Progress

To interface all peripherals with an SOC to obtain speedy I/O operations, communication protocols between them have to be developed. A Raspberry Pi SOC board is used to develop the project with I/O peripherals. ADC converter is used to convert a 0-20 mV analog signals into digital bit streams, and then it is given as an input to the Raspberry Pi (RPi) module. Here the communication protocol between RPi and ADC is Serial Peripheral Interface (SPI) protocol, developed with SPIDEV0.0 device driver available in Raspbian Wheeze, (A debian distribution of Linux), and Wiring Pi library is installed to get GPIO operations. Calculation of CG and weight of truck is being performed with RPi, using files as resources from the data base and input from the ADC. A header file for C.G. calculation is created, and by varying dimensional parameter in it, one can implement the C.G. calculation for any length of the weighbridge. According to the calculation either alignment or weighment signals will be generated or according operation has been performed.

For authentication, RF ID card reader is connected with RPi using RS232 protocol and a document from the database being extracted to make other entries in it. A composite video input from cameras is interfaced with RPi and a photo of truck is taken after its weighment, and added to the document within database. After

feeling up the document with details, entry is given to the SAP and database will be updated and a transaction is pronounced to be completed. To capture a composite image signal and convert as well as compress it to the format like .JPEG or .BMP we have to develop software base conversion algorithm too. It will also run on RPi with Linux.

A higher level of dependency between these functions can be observed, as RPi module has to deal with ADC, camera and RF ID reader as input peripherals; Sound file, light controller, and digitizer as output peripherals and SAP documents as input/output interface. It generates the billing information according to the weight difference from two different weighment of the same truck. To increase accuracy, a 12 bit ADC is used with SPI, to reduce pin-count as communication bus. Here 8 load cells are used so in order to perform analog to digital conversion if a facility of channel selectivity is available, then this operation can be done by a single IC and no separate IC (ADC) is required for each load cell. Hence a choice ends up with MCP3208 an 8 channel 12 bit ADC. As current weighment system is having an accuracy of 20 KG with an IC of 12 bit ADC converter.

At the end after completion of all I/O, document generated will be sent to the server in a particular package format. But in case if server is down, the weighment system must not be halt, and hence storage of document is there within SD card with which an Operating System (OS) is running on RPi SOC. To avoid conflicts within user accessible area and OS, and memory management policies should be used.

1.7 Thesis Organization

The rest of the thesis is organized as follows.

Basic flow of the project with an observation of its feasibility is discussed in **Chapter 2**. In this chapter the whole project is being represented with its subpart and the component for these subparts are discussed with their features in this chapter. This chapter gives a glance of getting started with Raspberry Pi System on chip. It is a review of the literature for the thesis work.

Chapter 3 is the description of hardware and software related issues, and very basic configuration of Raspberry Pi SOC to develop algorithms and programs in it. Network diagnostic, Keyboard configuration and diagnostic, Power diagnostic and other basic issues are discussed in this chapter.

In **Chapter 4**, detailed description of protocols used for inter module communication and related work is discussed; standard library installation for the purpose of driving protocols, comparison of different protocols, and make them work simultaneously causes a complexity of system to achieve end goal easily, are the issues to be discussed in this chapter. While **Chapter 5** is consists of the results and comparison of the developed system with a standard one.

Conclusion and future scope of work for this project is given in **Chapter 6**, followed by references as a literature survey. Thus the thesis includes all the possible expect with which this project will be approached toward fulfillment, and also it is informative enough.

Chapter 2

Components of Project and Literature Survey

In this chapter a detailed discussion of the project components and project flow is accounted. As a part of implementation, and in development phase, just a concept is a scrap until a horizon of its feasibility is observed. To make a feasible implementation, beyond just documentation, a project flow is indeed necessary. This chapter consists of such components details and the flow of the project with such components that approaches to a feasible implementation of idea. A proper insight can be developed by going through the flow chart and block diagram of this project.

2.1 Block diagram

First block is load cells, from where analog outputs come out,this signals are required to be conditioned and amplified that will taken care by block 2. It is further given to the ADC circuit, which converts these analog inputs to the digital outputs, and further given to SOC, as an input, that is combined as shown in block 3. Further an SOC block is available to look after all setting all the parameters by making according calculations on digital input provided by ADC from block 3. Finally an access to the central data base and server is provided to manipulate data with SAP system as shown in block 5.

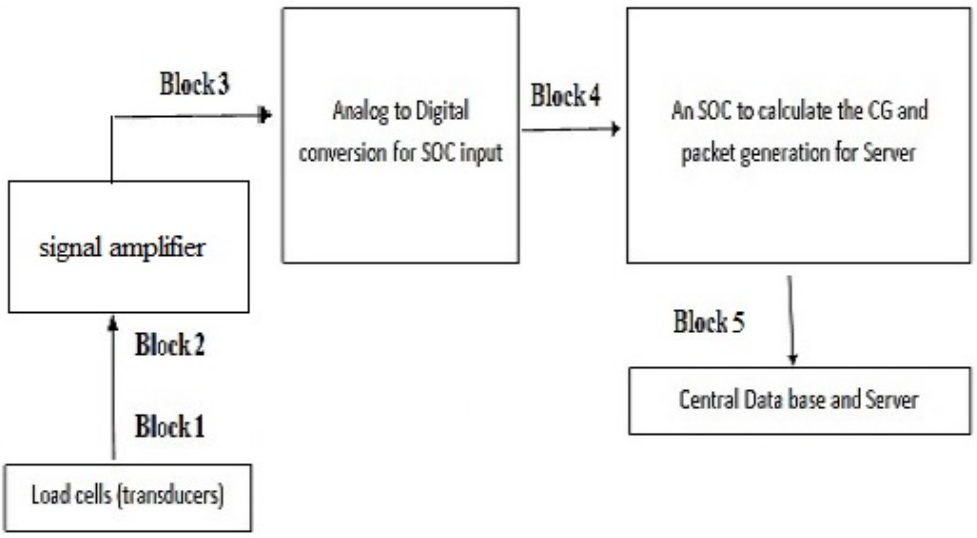

Figure 2.1: Block diagram of unmanned weighbridge

2.2 Load Cell (Block 1)

Features of the load cell 65058 are as below.

- Rated capacities of 10,000 to 100,000 pounds
- Center-link loaded
- Integral conduit adapter
- Trade certified for NTEP Class IIIL: 10000 divisions; Class III: 5000 divisions and OIML R60 3000 divisions in 20,000 to 100,000 pounds range
- Sensorgage sealed to IP67 standards
- Factory Mutual System Approved for Classes I, II, III; Divisions 1 and 2; Groups A through G. Also, non-incendive ratings (No barriers!).

The 65058 is a mid to high capacity, nickel-plated alloy steel, double ended Shear beam load cell. This product is designed for use in certified truck and rail scales and is available in capacities ranging from 10k to 100k lbs. This load cell is rated intrinsically safe by the Factory Mutual System (FM); making it suitable for use in potentially explosive environment. This load cell is certified for legal for trade applications by both American NTEP and International OIML standards.

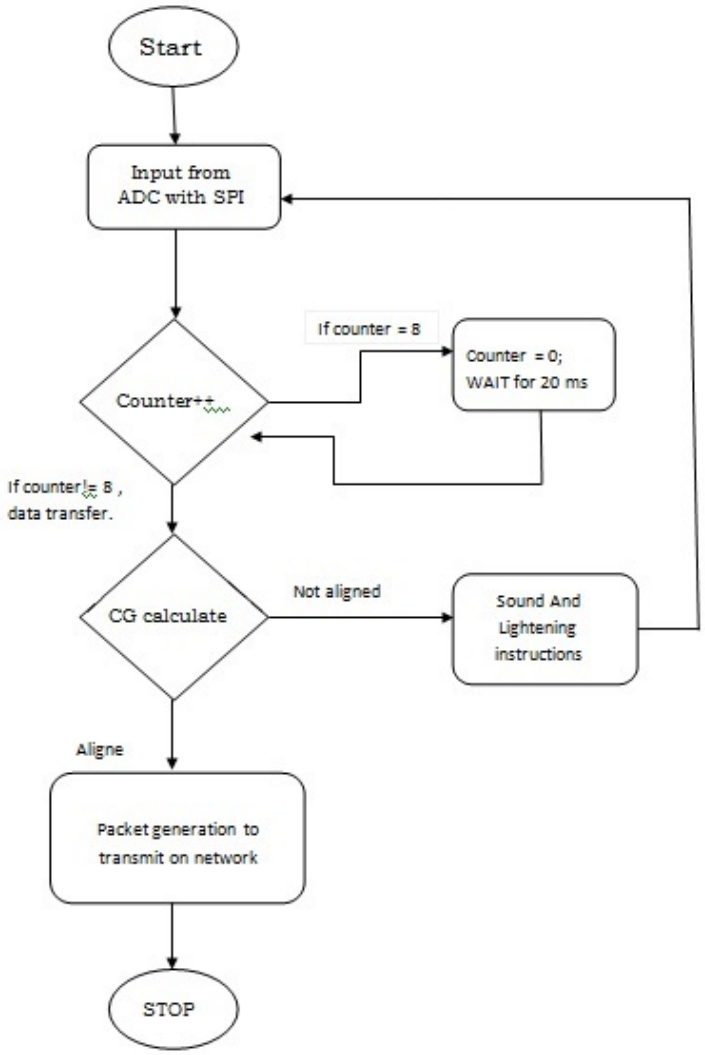

Figure 2.2: Flow chart of unmanned weighbridge

2.2.1 How to select a load cell ?

From the loading capacity and self weight, the wagons or trucks can be classified as shown in the figure 2.4. All the weights are measured in pounds. Trucks used for cement industries lies in class 7 and class 8 and they have weights around 33001 lbs (15000 Kg) to 80000 lbs(36000 Kg) as gross weight.

2.3 Strain Gage Amplifier(Block 2)

An op-amp with no feedback is already a differential amplifier, used as a strain gage amplifier for amplifying the voltage difference between the two inputs. However, its

Figure 2.3: Load Cell 65058

gain cannot be controlled, and it is generally too high to be of any practical use. So far, the application of negative feedback to op-amps has resulting in the practical loss of one of the inputs, the resulting amplifier only good for amplifying a single voltage signal input. With a little ingenuity, however, construction of an op-amp circuit maintaining both voltage inputs is required, yet with a controlled gain set by external resistors. If all the resistor values are equal, this amplifier will have a differential voltage gain of 1. The analysis of this circuit is essentially the same as that of an inverting amplifier, except that the non inverting input (+) of the op-amp is at a voltage equal to a fraction of V2, rather than being connected directly to ground. As would stand to reason, V2 functions as the non inverting input and V1 functions as the inverting input of the final amplifier circuit

If a differential gain of anything other than 1 is required, the resistances in both upper and lower voltage dividers must be adjusted accordingly , necessitating multiple resistor changes and balancing between the two dividers for symmetrical operation. This is not always practical, for obvious reasons.

2.4 ADC MCP3208 (Block 3)

Features of this IC are below

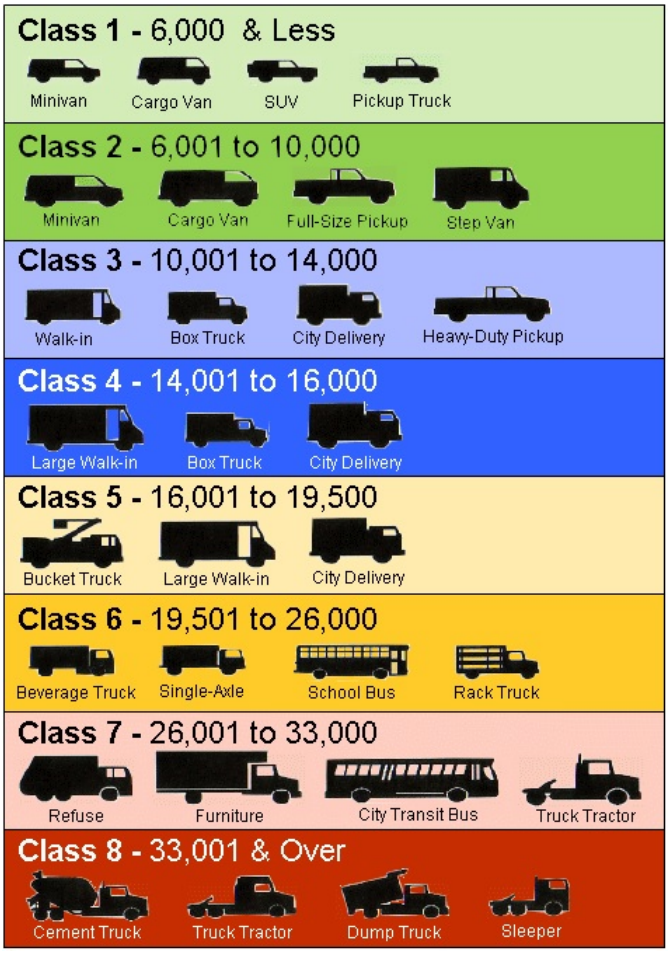

Figure 2.4: Example of truck in each truck class.
Source: Oak Ridge National Laboratory, Center for Transportation Analysis, Oak Ridge, TN. Weight category definitions from 49CFR565.6 (2000)

- 12-bit resolution
- ± 1 LSB max DNL
- ± 1 LSB max INL (MCP3204/3208-B)
- ± LSB max INL (MCP3204/3208-C)
- 4 (MCP3204) or 8 (MCP3208) input channels
- Analog inputs programmable as single-ended or pseudo-differential pairs
- On-chip sample and hold
- SPI serial interface (modes 0,0 and 1,1)

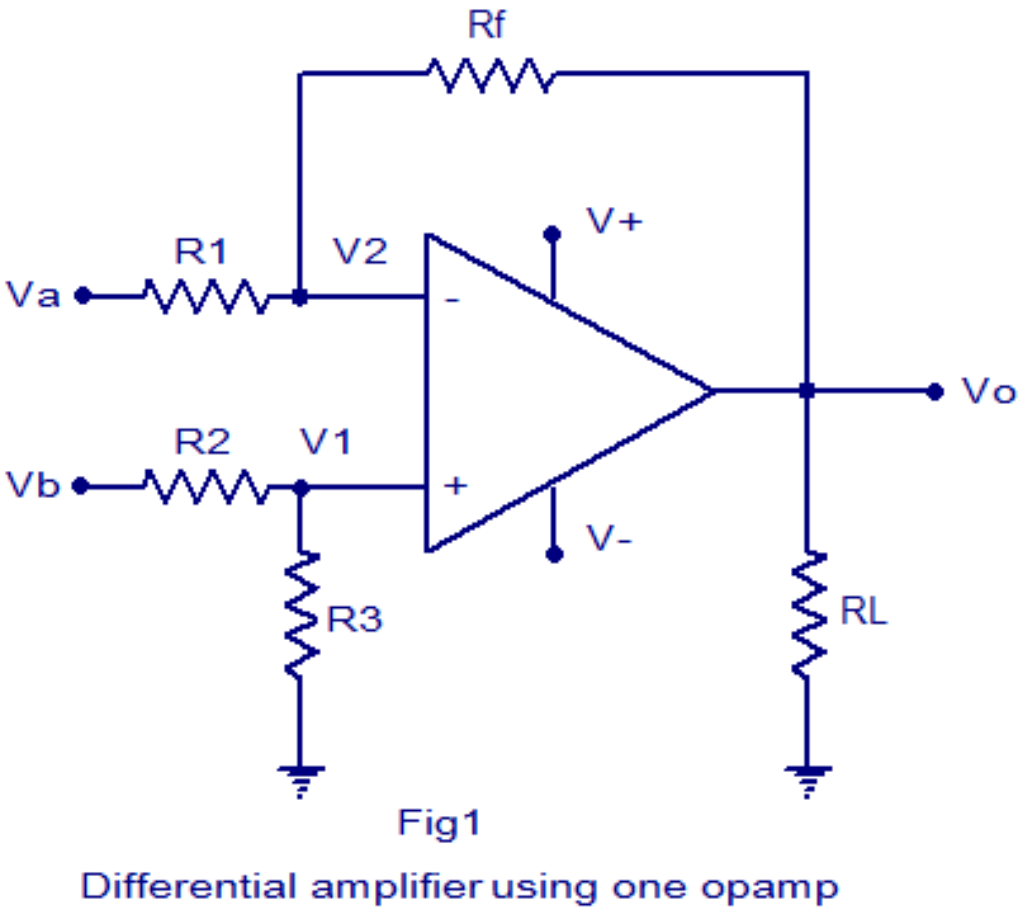

Figure 2.5: Differential Amplifier with Op-amp

- Single supply operation: 2.7V - 5.5V
- 100 ksps max. sampling rate at VDD = 5V
- 50 ksps max. sampling rate at VDD = 2.7V
- Low power CMOS technology:
  - 500 nA typical standby current, 2 μ A max.
  - 400 μ A max. active current at 5V.
- Industrial temp range: -40C to +85C
- Available in PDIP, SOIC and TSSOP packages

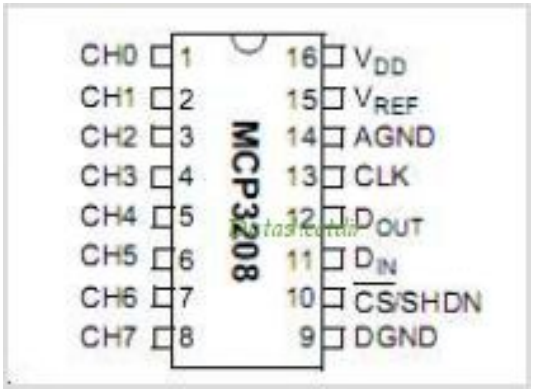

Figure 2.6: ADC MCP3208

2.4.1 ADC operation

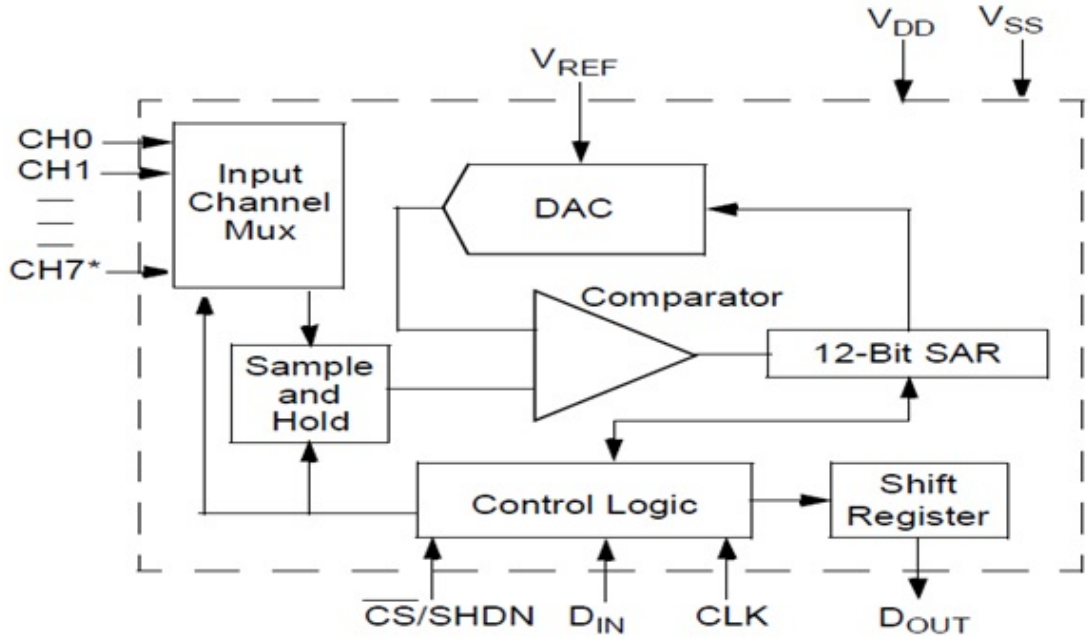

Figure 2.7: Functional block diagram of ADC3208

The MCP3208 A/D converters employ a conventional SAR architecture. With this architecture, a sample is acquired on an internal sample/hold capacitor for 1.5 clock cycles starting on the fourth rising edge of the serial clock after the start bit has been received. Following this sample time, the device uses the collected charge on the internal sample/hold capacitor to produce a serial 12-bit digital output code. Conversion rates of 100 ksps are possible on the MCP3208. Communication with the device is accomplished using a 4-wire SPIcompatible interface.

2.4.2 Analog Input

The MCP3208 devices offer the choice of using the analog input channels configured as single-ended inputs or pseudo-differential pairs. MCP3208 can be configured to provide four pseudo-differential input pairs or eight single-ended inputs. Configuration is done as part of the serial command before each conversion begins. When used in the pseudo-differential mode, each channel pair (i.e., CH0 and CH1, CH2 and CH3 etc.) is programmed to be the IN+ and IN- inputs as part of the command string transmitted to the device. The IN+ input can range from IN- to (VREF + IN-). The IN- input is limited to 100 mV from the VSS rail. The IN- input can be used to cancel small signal common- mode noise which is present on both the IN+ and IN- inputs. When operating in the pseudo-differential mode, if the voltage level of IN+ is equal to or less than IN-, the resultant code will be 000h. If the voltage at IN+ is equal to or greater than [VREF + (IN-)] - 1 LSB, then the output code will be FFFh. If the voltage level at INis more than 1 LSB below VSS, the voltage level at the IN+ input will have to go below VSS to see the 000h output code. Conversely, if IN- is more than 1 LSB above VSS, then the FFFh code will not be seen unless the IN+ input level goes above VREF level. For the A/D converter to meet specification, the charge holding capacitor (CSAMPLE) must be given enough time to acquire a 12-bit accurate voltage level during the 1.5 clock cycle sampling period.

2.4.3 Reference Input

For each device in the family, the reference input (VREF) determines the analog input voltage range. As the reference input is reduced, the LSB size is reduced accordingly. The theoretical digital output code produced by the A/D converter is a function of the analog input signal and the reference input.

$$DigitalOutputCode = \frac{4096XV_I}{V_R}$$

V_I = Analog input voltage
V_R= Reference voltage

2.4.4 Pin Descriptions

DGND: Digital ground connection to internal digital circuitry.

AGND: Analog ground connection to internal analog circuitry.

CH0 - CH7: Analog inputs for channels 0 - 7 for the multiplexed inputs. Each pair

of channels can be programmed to be used as two independent channels in single-ended mode or as a single pseudo-differential input, where one channel is IN+ and one channel is IN.

Serial Clock (CLK): The SPI clock pin is used to initiate a conversion and clock out each bit of the conversion as it takes place.

Serial Data Input (DIN): The SPI port serial data input pin is used to load channel configuration data into the device.

Serial Data Output (DOUT): The SPI serial data output pin is used to shift out the results of the A/D conversion. Data will always change on the falling edge of each clock as the conversion takes place.

Chip Select/Shutdown (CS/SHDN): The CS/SHDN pin is used to initiate communication with the device when pulled low and will end a conversion and put the device in low power standby when pulled high. The CS/SHDN pin must be pulled high between conversions.

2.5 Raspberry Pi (System on Chip) (Block 4)

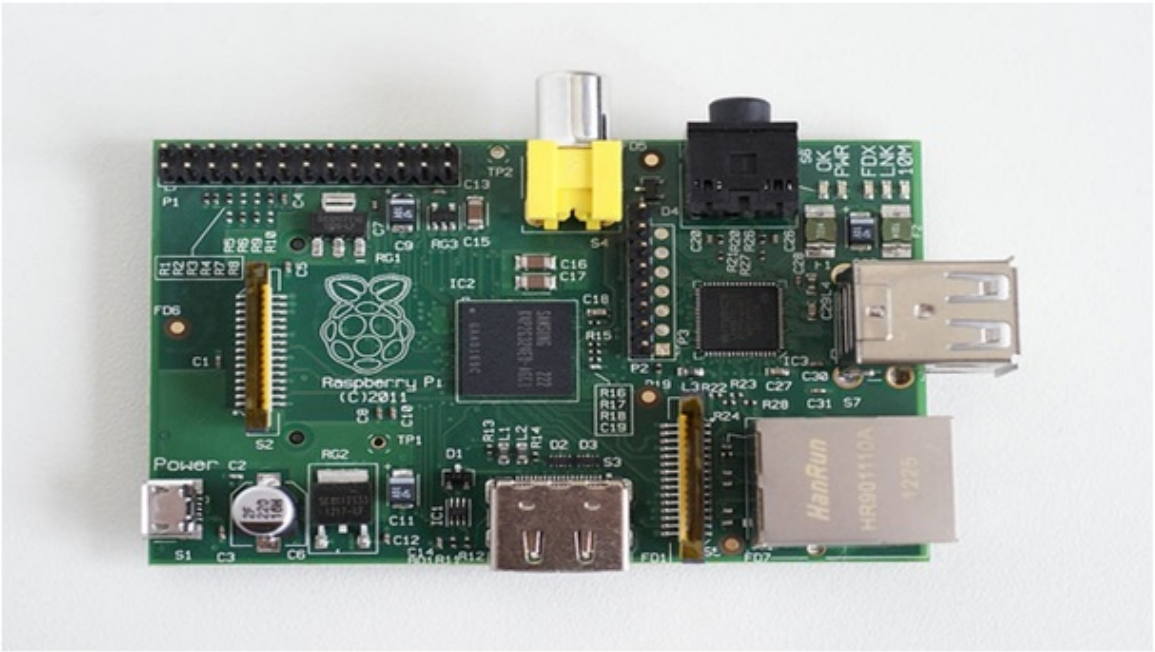

Figure 2.8: Raspberry Pi SOC

The Raspberry Pi board contains a processor and graphics chip, program memory (RAM) and various interfaces and connectors for external devices. Some of these devices are essential, others are optional. Raspberry Pi (RPi) operates in the same way as a standard PC, requiring a keyboard for command entry, a display unit and a power supply. It also requires mass-storage, but a hard disk drive of the type found in a typical PC is not really in keeping with the miniature size of Raspberry Pi. Instead we will use an SD Flash memory card normally used in digital cameras, configured in such a way to look like a hard drive to RPis processor. RPi will boot (load the Operating System into RAM) from this card in the same way as a PC boots up into Windows from its hard disk.

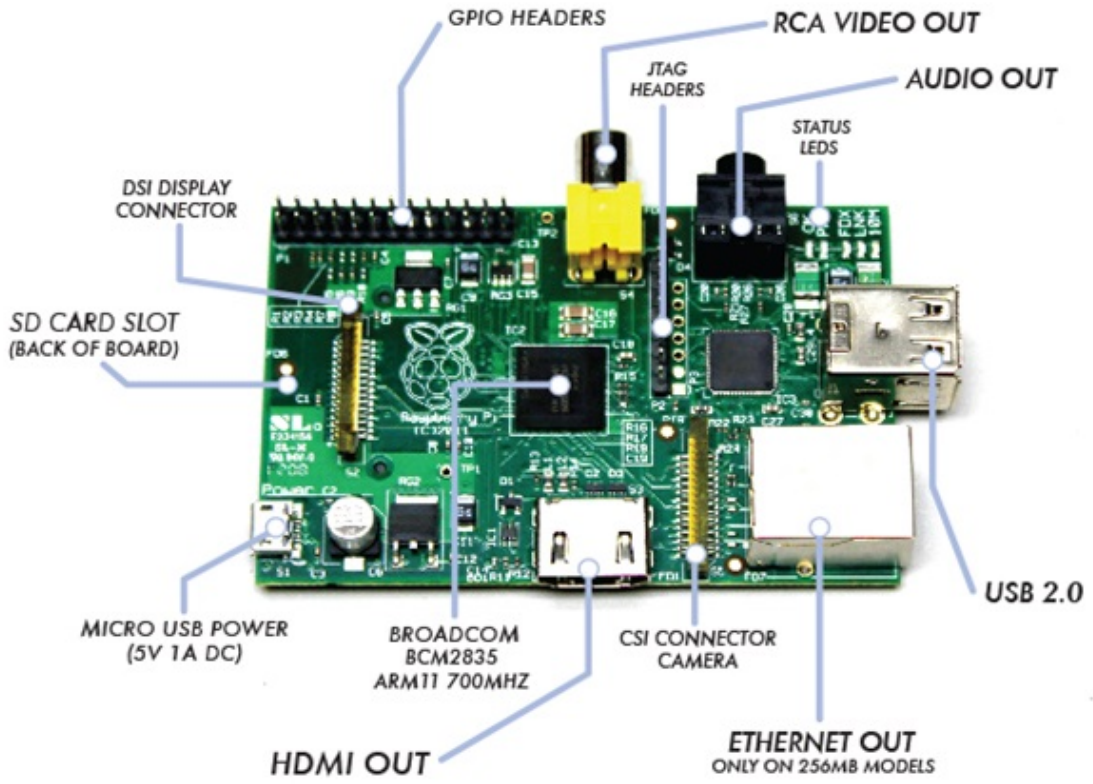

Figure 2.9: Raspberry Pi with onboard peripherals

2.5.1 Getting started with Raspberry Pi

The following are essential to get started

- SD card containing Linux Operating system
- USB keyboard
- TV or monitor (with HDMI, DVI, Composite or SCART input)
- Power supply
- Video cable to suit the TV or monitor used

Connecting everything together as following for getting started

a. Plug the preloaded SD Card into the RPi.

b. Plug the USB keyboard and mouse into the RPi, perhaps via a USB hub. Connect the Hub to power, if necessary.

c. Plug a video cable into the screen (TV or monitor) and into the RPi.

d. Plug your extras into the RPi (USB WiFi, Ethernet cable, external hard drive etc.). This is where you may really need a USB hub.

e. Ensure that your USB hub (if any) and screen are working.

f. Plug the power supply into the mains socket.

g. With your screen on, plug the power supply into the RPi microUSB socket.

h. The RPi should boot up and display messages on the screen.

Operating System SD card: As the RPi has no internal mass storage or built-in operating system it requires an SD card preloaded with a version of the Linux Operating System. Own preloaded card can be created using any suitable SD card (4GBytes or above) you have to hand.

Keyboard and Mouse: Most standard USB keyboards and mice will work with the RPi. Wireless keyboard/mice should also function, and only require a single USB port for an RF dongle. In order to use a Bluetooth keyboard or mouse, a Bluetooth USB dongle is required, which again uses a single port. Model A has a single USB port and the Model B has two (typically a keyboard and mouse will use a USB port each).

Display: There are two main connection options for the RPi display, HDMI (High Definition) and Composite (Standard Definition). HD TVs and many LCD monitors

can be connected using a full-size 'male' HDMI cable, and with an inexpensive adaptor if DVI is used. HDMI versions 1.3 and 1.4 are supported and a version 1.4 cable is recommended. The RPi outputs audio and video via HMDI, but does not support HDMI input. Older TVs can be connected using Composite video (a yellow-to-yellow RCA cable) or via SCART (using a Composite video to SCART adaptor). Both PAL and NTSC format TVs are supported. When using a composite video connection, audio is available from the 3.5mm jack socket, and can be sent to TV, headphones or an amplifier. To send audio to TV, a cable which adapts from 3.5mm to double (red and white) RCA connectors is required.

There is no analogue VGA output available. This is the connection required by many computer monitors, apart from the latest ones. If there is a monitor with only a D-shaped plug containing 15 pins, then it is unsuitable.

Power Supply: The unit is powered via the microUSB connector (only the power pins are connected, so it will not transfer data over this connection). A standard modern phone charger with a microUSB connector will do, providing it can supply at least 700mA at +5Vdc.

2.5.2 Preparing SD card for the Raspberry Pi

To prepare a blank SD card for use with the Raspberry Pi, is needed to flash an operating system onto the card. While this is slightly more complicated than simply dragging and dropping files onto the card, it shouldn't take more than a few minutes to complete.

Firstly, it is needed to decide which Linux distribution would be preferred to use with Raspberry Pi. Each has its advantages and disadvantages. The most up-to-date list of Linux releases compatible with the Pi is available from the Raspberry Pi website at

http://www.raspberrypi.org/downloads.

The Foundation provides BitTorrent links for each distribution. These are small files that can be used with BitTorrent software to download the files from other users. Using these links is an efficient and fast way to distribute large files, and keeps the Foundation's download servers from becoming overloaded. To use a BitTorrent link, it is recommended to have a compatible client installed. If no BitTorrent client is installed on a PC, download one and install it before trying to download the Raspberry Pi Linux distribution. One client for Windows, OS X and Linux is Torrent, can be downloaded from link below.

http://www.utorrent.com/downloads

Which distribution is chosen to download is up to you. Instructions are given for the Debian Raspberry Pi distribution, a good choice for beginners. Linux distributions for the Raspberry Pi are provided as a single image file, compressed to make it faster to download. Once Zip archive is being downloaded the (a compressed file, which takes less time to download than the uncompressed files would) for chosen distribution, it is needed to decompress it somewhere in system. In most operating systems, simply double-click the file to open it, and then choose Extract or Unzip to retrieve the contents. After decompression the archive, you'll end up with two separate files. The file ending in sha1 is a hash, which can be used to verify that the download hasn't been corrupted in transit. The file ending in img contains an exact copy of an SD card set up by the distributions creators in a way that the Raspberry Pi understands. This is the file that needs to be flashed to the SD card.

2.5.3 Flashing from Linux

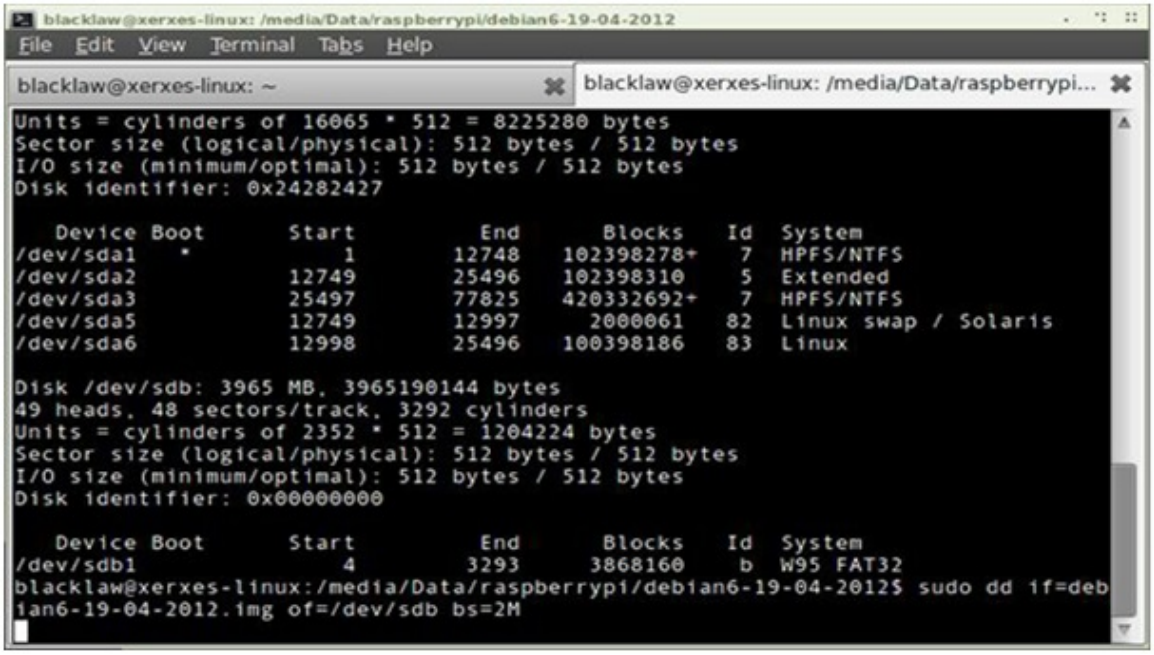

Figure 2.10: Flashing SD card with dd tools

With a PC , running a variant of Linux already, one can use the dd command to write the contents of the image file out to the SD card. This is a text-interface program operated from the command prompt, known as a terminal in Linux parlance. Follow these steps to flash the SD card:

- Open a terminal from your distributions applications menu.

- Plug a blank SD card into a card reader connected to the PC.

- Type sudo fdisk -l to see a list of disks. Find the SD card by its size, and note the device address (/dev/sdX, where X is a letter identifying the storage device. Some systems with integrated SD card readers may use the alternative format/dev/mmcblkXif this is the case, remember to change the target in the following instructions accordingly).

- Use cd to change to the directory with the .img file you extracted from the Zip archive.

- Type sudo dd if=imagefilename.img of=/dev/sdX bs=2M to write the file imagefilename.img to the SD card connected to the device address from step 3. Replace imagefilename.img with the actual name of the file extracted from the Zip archive. During flashing, nothing will be shown on the screen until the process is fully complete (see Figure 2.10).

2.6 Summary

Feasibility can be observed in this project as it is being divided into multiple sub part and then by commencing step by step, as show in flow chart. Block diagram representation shows different subparts of the project to be executed, and with all those basic blocks, the component related to them are discussed here. Raspberry Pi as a new SOC is being studied here with a bird eye view to getting started with it. Flashing an SD card with dd tool is very primary thing to dump an operating system on an SOC, and how to do this, is shown here within this chapter. This basics, helps to do further programming and application development with Raspberry Pi system. An overview of ADC and load cell is taken in account to understand the I/O device functionality and an overview of intermediate data conversion. A detailed study is shown in following chapters.

Chapter 3

Diagnosis of System Components

Basic configuration issues, related to hardware and software, that have to be addressed while working with Raspberry Pi, are discussed in this chapter. Many packages are pre-installed in Raspbian Wheezy to support different hardware interfaces, but some extra libraries or packages have to be installed. In order to get a proper access to the Raspberry Pi board, creation of configuration file is essential, as per application; otherwise Raspberry Pi will work with default configuration. In addition with that, power and performance related issues may cause a breakdown in working Raspberry Pi system, and hence to be addressed very carefully. Issues documented here are keyboard diagnostic, network diagnostic, power diagnostic and basic configurations for Raspberry Pi SOC. An ADC interface approach and C.G. calculation is shown here with detailed description. The way of calibrating a load cell with the standard balances, is also an approach to discuss here.

3.1 Raspberry Pi Configuration

Topic listed below are discussed in detail to configure Raspberry Pi SOC.

- Keyboard Diagnostic
- Network Diagnostic
- Power Diagnostic
- Config.txt

3.1.1 Keyboard Diagnostic

A common problem observed with Raspberry Pi board is, a keyboard repeats certain characters a number of times. For example, instead of startx if stttttttttartxxxxxxxxxx appears on the terminal, the execution of this command is failed, in terminal after pressing enter key. This may a cause due to either of two reasons.

a. Keyboard drawing too much power,

b. Keyboard configuration gets a conflict with chipset.

Keyboard drawing too much power

Going through the documentation given with the keyboard, or the label on its underside are the ways to get information about power requirement of keyboard. It has a current rating in milliamp (mA), that it tries to draw from USB port of Raspberry Pi system, when connected. Raspberry Pi has a component called polyfuse, having current capacity of 150 mA, causes the USB to shutoff when it draws a current beyond 150 mA.

If a higher power requirement is observed in USB keyboard, beyond the polyfuse current limit, a powered USB hub should be used to protect the board rather than direct connection with USB port of Raspberry Pi. Keyboard with inbuilt LED utilizes more power than a standard keyboard. The place of polyfuse on the Raspberry Pi board is shown in figure 3.1.

Keyboard configuration gets a conflict with chipset

Configuration of keyboard is done to minimal the chances of conflicts with Raspberry Pi chipset. Raspberry Pi gives various options to choose among available keyboard configurations in order to make a compatibility with keyboard being interfaced. Some packages are there and are given as a command line to the terminal to change the configuration.

sudo dpkg-reconfigure keyboard-configuration

This package causes to open a consol shown in figure 3.2. It has a verity of available keyboards, like wise Dell, Accer, Logitech, Intex, TVS and many more. Here Logitech Media Elite is chosen to interface with Raspberry Pi. Then it will ask for the layout; it may be any of available standard layouts as Dvorak, US, QWERTY, Macintosh, Left Handed Dvorak and many more. A keyboard must be chosen by maximum compatibility with the keyboard being interfaced.

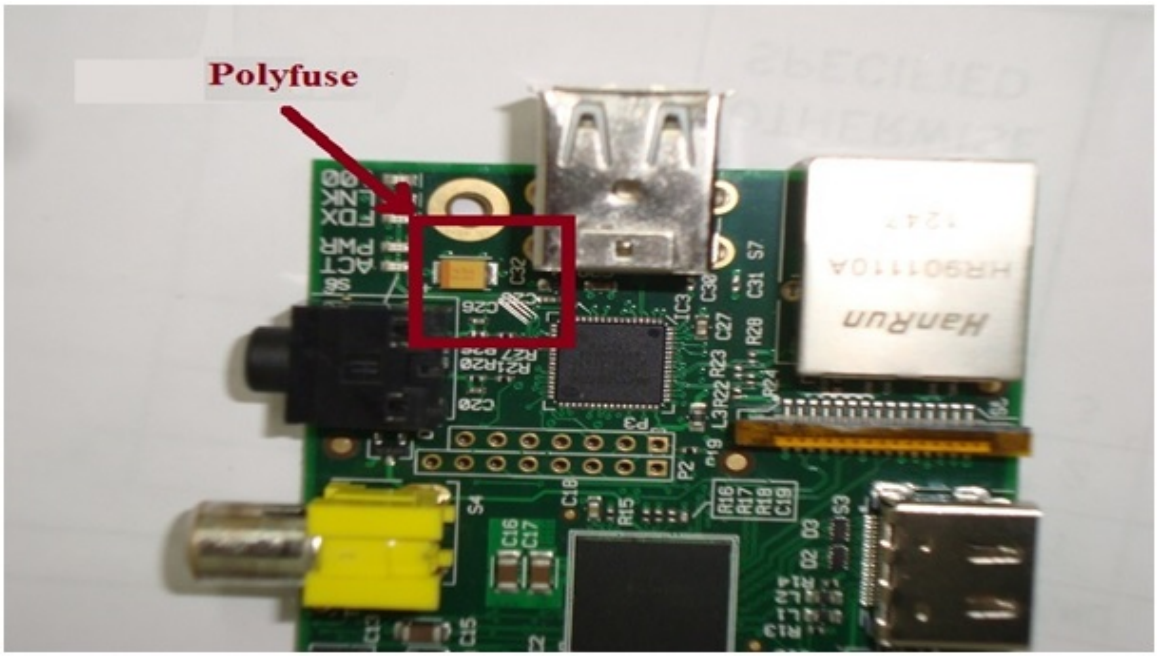

Figure 3.1: Polyfuse in Raspberry Pi board for USB protection

Its analyzed that, by going through this all consoles, a configuration file for keyboard is being modified and being loaded at boot time. If this configuration is not done properly, repetitive character syndrome, or wrong ASCII input will be considered. Errors like error:/207, error:/313 are being observed due to mismatch of keyboard, while compiling a program with gcc compiler. It shows, ASCII not matched or undefined UTF character, not supported by ANSI C format.

3.1.2 Network Configuration and diagnostic

Without configuring a static IP in Raspberry Pi board, we can use the module with Direct Host Configuration Protocol (DHCP) to connect it with a network through Ethernet. But if we want to use it in a limited access network, than we have to assign a static IP address as well as gateway and other networking information for the Raspberry Pi to be connected. A configuration file is generated and being loaded at the time of booting, assign an IP address to the Raspberry Pi module. An inbuilt text editor is used to edit this file.

Assignment of Static IP Address

To assign a static IP address to a Raspberry Pi module, it is necessary to make changes into network interface files as shown below with the command,

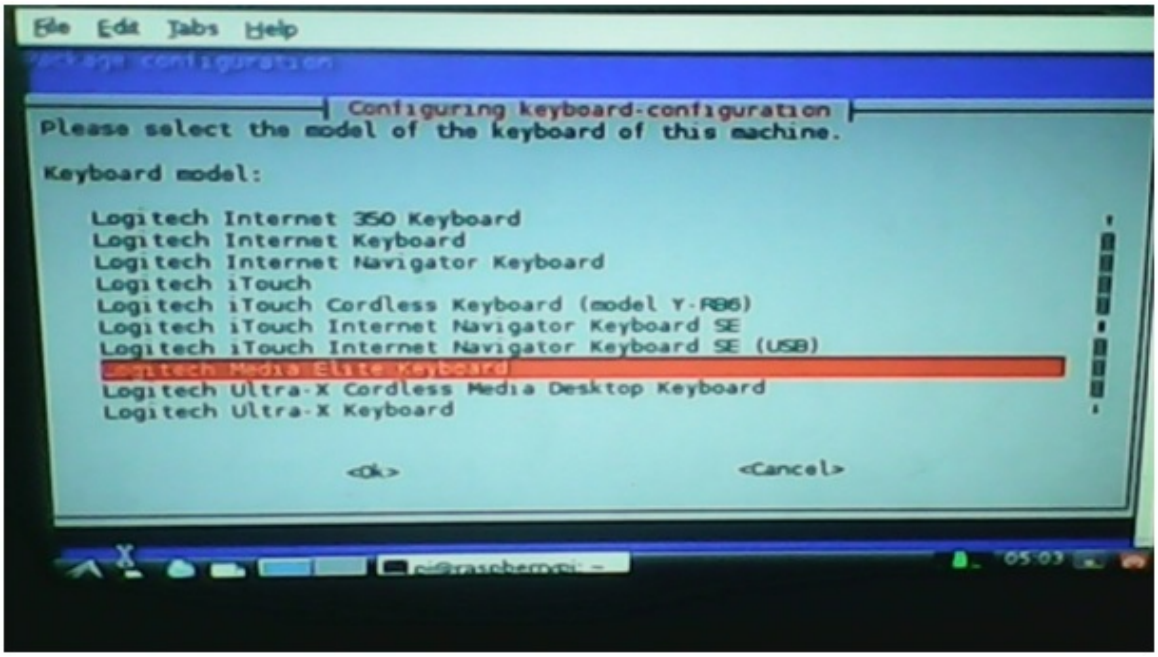

Figure 3.2: consol for keyboard configuration 1/5

sudo nano /etc/network/interfaces

By applying this command line in terminal, an editor screen as below is shown. Additions of address, netmask and gateway have to be done, with the lines starting with a tab as shown in the figure 3.7. By making this change, whenever the module is being rebooted, it will automatically load IP address written in this file, and same with gateway and netmask too.

Nano is an inbuilt text editor and it only permit a write operation to the sudoers, otherwise it only open the document in read only access, and hence to make any changes within the document, sudo must be written before nano command line.

DNS server configuration

DNS server converts an alphanumeric web address into IP address from where the data can be originated to propagate into the network. Without setting proper DNS server an error called temporary name resolution failure is generated causes, the sudo apt-get update command not to work. To avoid this problem we have to execute the command line in terminal given below.

sudo nano /etc/network/interfaces

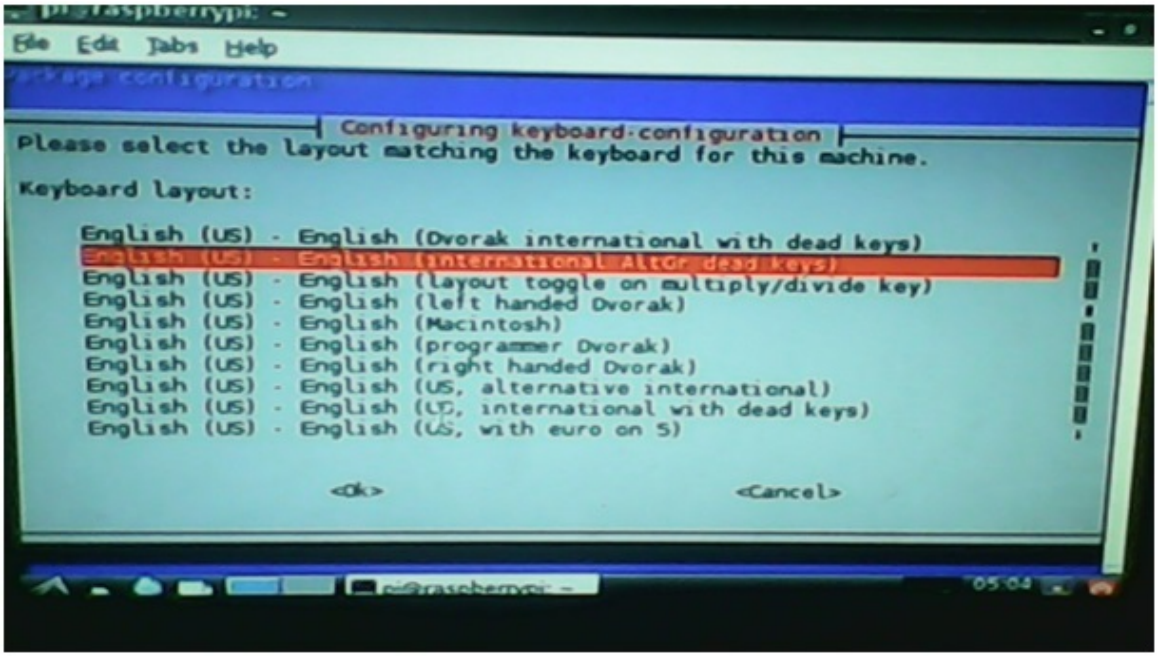

Figure 3.3: consol for keyboard configuration 2/5

It will open a file, and two lines have to be edited within it to get a proper access to the net. If instead of IP address of a DNS server, a name of DNS server is given, then the system tries to find the resolution for this server name from this file only, and hence it ties up into an infinite loop, causes the system to be hanged up. Figure 3.8 shows how the resolv.conf file is look like.

Network diagnostic

The most powerful tool for network diagnostic is ifconfig. It is the most powerful tool for controlling and configuring the Pis network port. To see the output, ifconfig should be written into the terminal. Output is as given in figure 3.9.

The output of ifconfig is split into the following sections:

Link encap - The type of encapsulation used by the network, which on the Model B will either read Ethernet for the physical network port or Local Loop back for the virtual loop back adapter.

Hwaddr - The Media Access Control (MAC) address of the network interface, written in hexadecimal. This is unique for every device on the network, and each Pi has its own MAC address, which is set at the factory. Here it is b0:27:eb:c1:ce:4b.

inet addr - The Internet protocol (IP) address of the network interface. This is how one find the Pi on the network, using it to run a network-accessible service, such

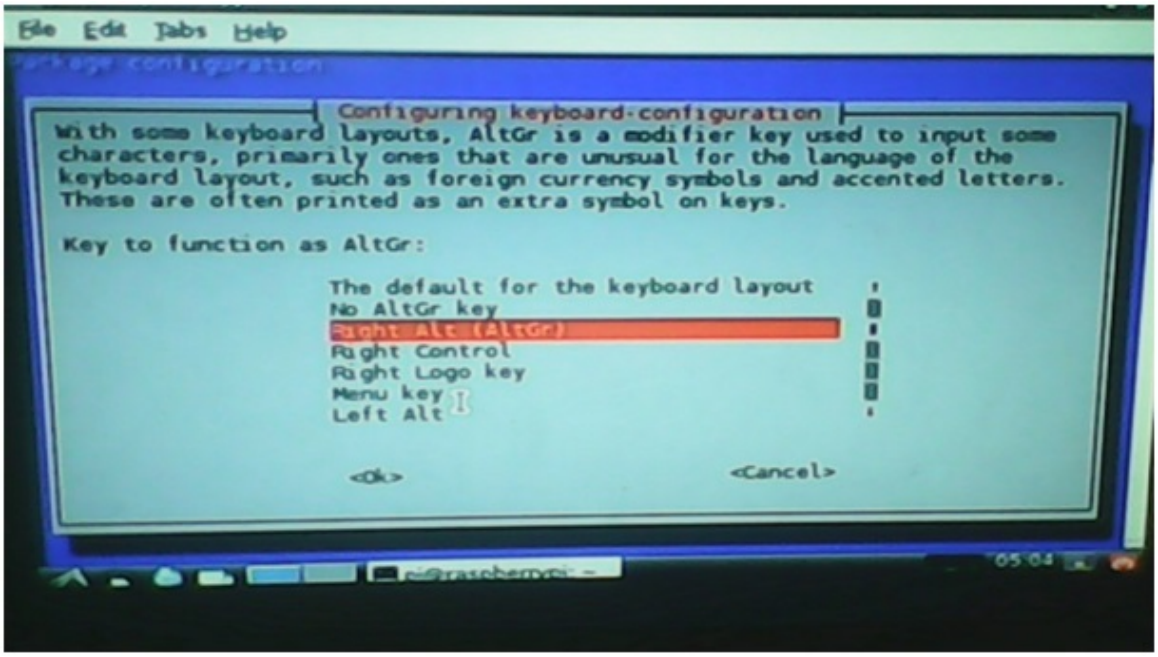

Figure 3.4: consol for keyboard configuration 3/5

as a web server or file server. Here it is 101.0.0.126.

Bcast - The broadcast address for the network to which the Pi is connected. Any traffic sent to this address will be received by every device on the network. Here it is 11.255.255.255.

Mask - The network mask, which controls the maximum size of the network to which the Pi is connected. Here it is 255.0.0.0.

MTU - The maximum transmission unit size, which is how big a single packet of data, can be before the system needs to split it into multiple packets. Here it is 1500 bytes.

RX - This section provides feedback on the received network traffic, including the number of errors and dropped packets recorded. If you start to see errors appearing in this section, there's something wrong with the network.

TX - This provides the same information as the RX section, but for transmitted packets. Again, any errors recorded here indicate a problem with the network.

collisions - If two systems on the network try to talk at the same time, you get a collision which requires them to retransmit their packets. Small numbers of collisions aren't a problem, but a large number here indicates a network issue.

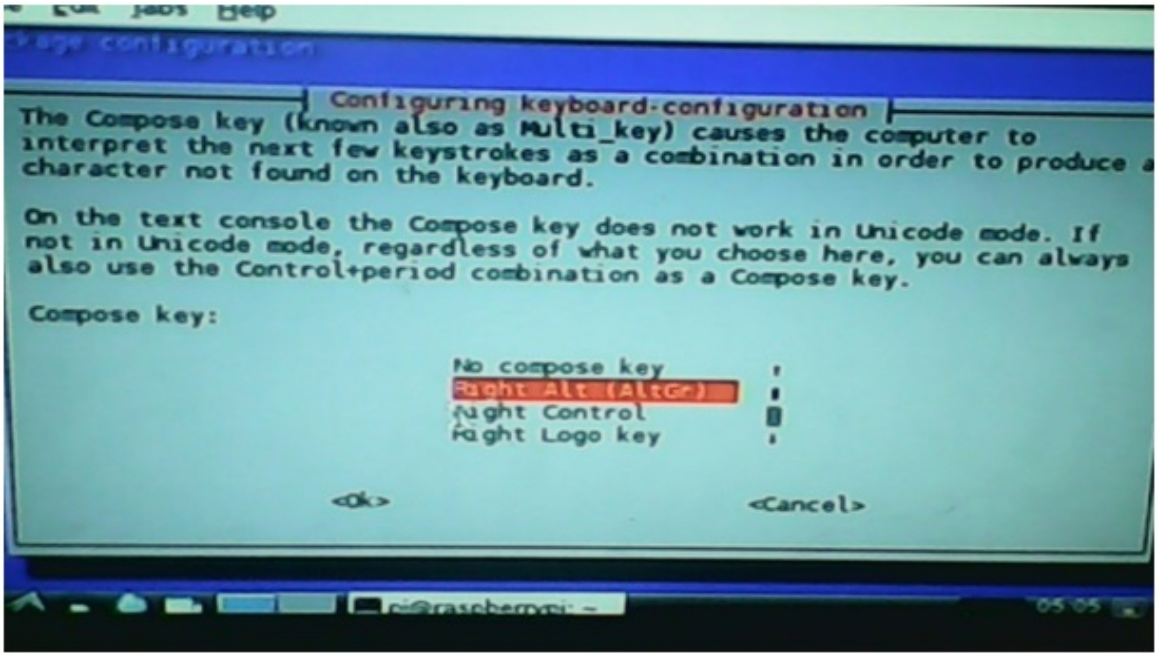

Figure 3.5: consol for keyboard configuration 4/5

txqueuelen - The length of the transmission queue, which will usually be set to 1000 and rarely needs changing.

RX bytes, TX bytes - A summary of the amount of traffic the network interface has passed.

3.1.3 Power Diagnostic

Many problems with the Raspberry Pi can be traced to an inadequate power supply. The Model A requires a 5 V supply capable of providing a 500 mA current, while the Model Bs extra components bump up the current requirement to 700 mA. Not all USB power adapters are designed to offer this much power, even if their labeling claims otherwise. The Pi provides a relatively easy way to check if this is the case in the form of two voltage test points. To use the voltage test points, a voltmeter or multimeter with direct current (DC) voltage measuring capabilities is required. If a meter has multiple inputs for different voltages, then its better to use an appropriate setting.

Two test points are small, copper clad holes known as vias, which are connected to the Pis 5 V and ground circuits. Put the positive (red) meter probe on TP1, located to the left of the board just above a small black component called a regulator

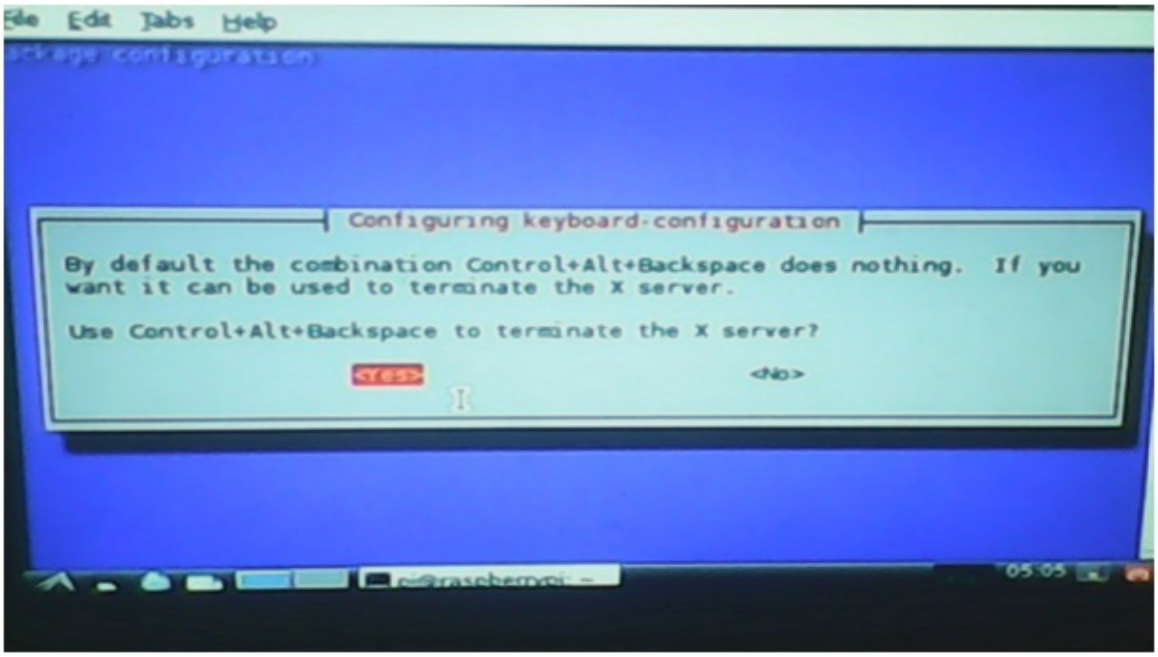

Figure 3.6: consol for keyboard configuration 5/5

labelled RG2. Connect the black (negative) meter probe to TP2, located between the copper GPIO pins and the yellow-and-silver RCA phono connector at the top-left of the board (see Figure 3.10).

The reading on the voltmeter should be somewhere between 4.8 V and 5 V. If its lower than 4.8 V, this indicates that the Pi is not being provided with enough power. Try swapping the USB adapter for a different model, and check that the label says it can supply 700 mA or more. A model rated at 1A is recommended, but beware of cheap models, they sometimes have Inaccurate labelling, and fail to supply the promised current. Genuine branded mobile phone chargers rarely have this problem, but cheap unbranded devices often sold as compatible adapters should be avoided.

3.1.4 Installation of wiring PI

Wiring Pi is a library to access GPIO available on Raspberry Pi SoC. It is a part of git-core and hence installation of GIT-CORE is done prior to set up wiring pi. The command that has to be run in terminal with active Internet connection is as shown below.

sudo apt-get install git-core

Then access the URL shown below with proper IP Address settings that is shown in next topic. Its having latest version of wiring Pi for GPIO operation of Raspberry

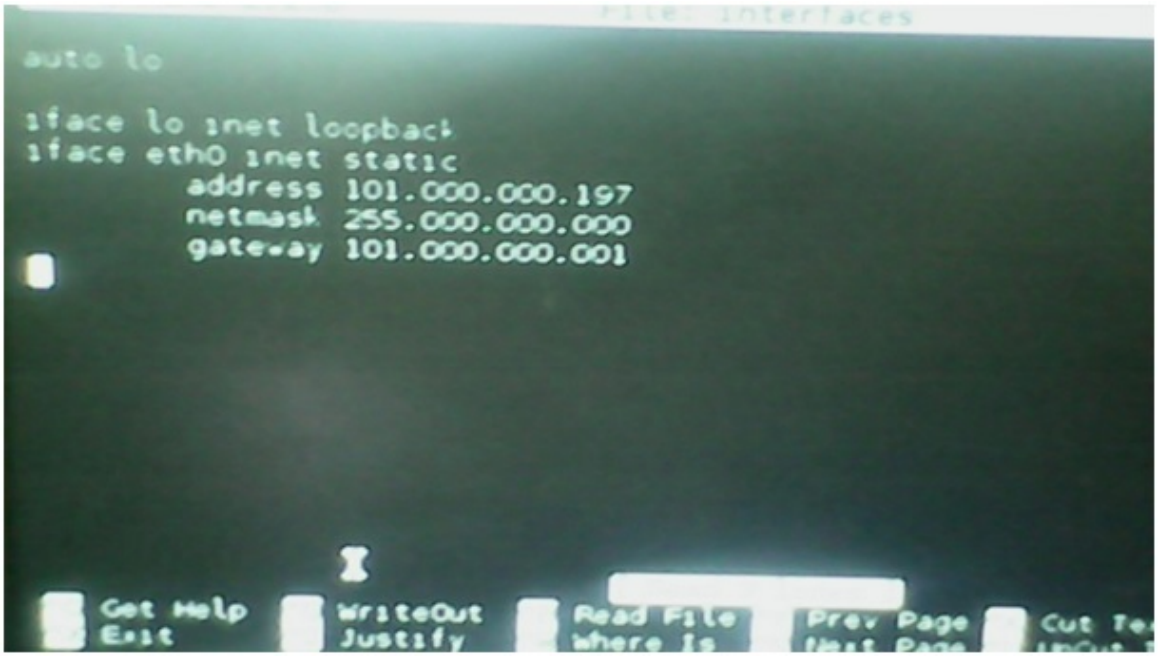

Figure 3.7: Nano Editor for interface of network

Pi.

https://git.drogon.net/?p=wiringPi;a=summary

It will download a tar.gz file with a name like wiringPi-98bcb20.tar.gz. Note that the numbers and letters after wiringPi (98bcb20 in this case) will probably be different theyre a unique identifier for each release. Following commands should be used to install wiring Pi for RPi, to get easy GPIO operations with simple command lines.

tar xfz wiringPi-98bcb20.tar.gz
cd wiringPi-98bcb20
./build

These command lines causes to install wiring Pi library in Raspbian Wheezy and now GPIO are enabled with simple command lines. Test of the proper installation of Wiring Pi can be done with following commands.

gpio -v gpio readall

If Wiring Pi is being installed properly, it will give us a version detail, installed on board, and hence a conformation of a successful installation of WiringPi on RPi can be obtained.

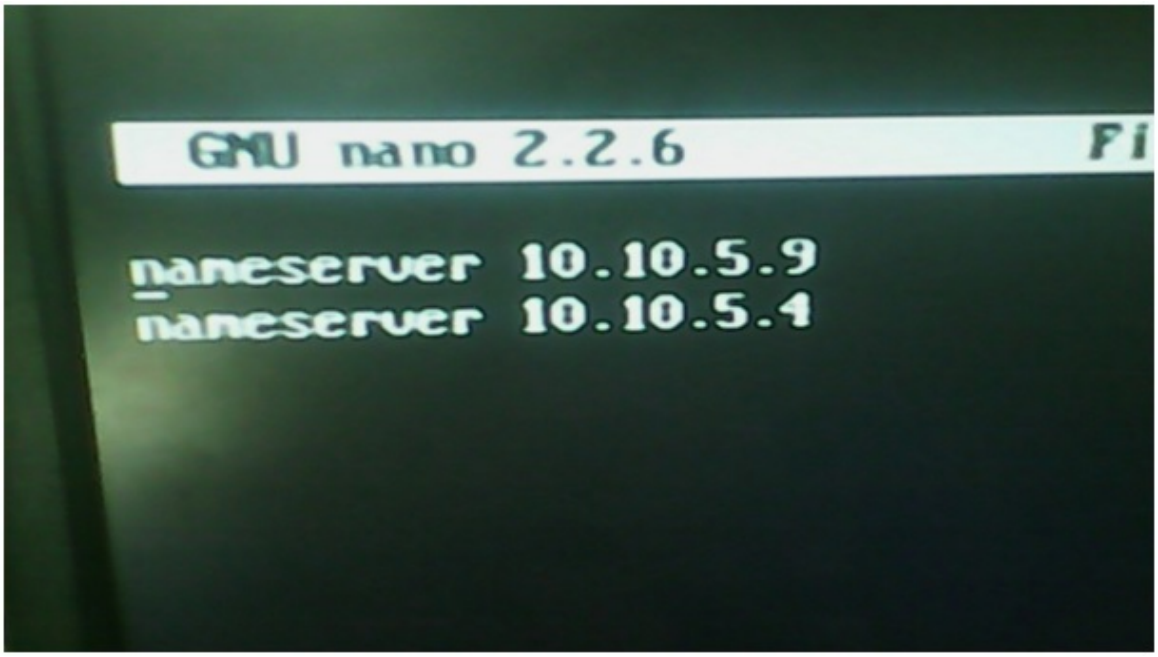

Figure 3.8: setting for resolution checking with DNS server

3.1.5 Loading SPI device driver

On many Raspberry Pi distributions, including debian based Raspbian wheezy; the SPI device drivers are not loaded by default on boot. The file named '/etc/modprobe.d/raspi-blacklist.conf' indicates, which kernel modules are not being loaded. Since SPI, isn't needed by most users, it need to be explicitly removed from that list by placing a mark (comment) in front of the 'blacklist' command. Edit and save '/etc/modprobe.d/raspi-blacklist.conf' as shown below.

nano /etc/modprobe.d/raspi-blacklist.conf
blacklist spi and i2c by default (many users don't need them)
blacklist spi-bcm2708
blacklist i2c-bcm2708

Then, to gain access to SPI and 1-Wire devices, their respective device drivers must be loaded, either by manually using the 'modprobe' command or on boot.

3.1.6 GPIO Overview

It is basic GPIO layout of RPi, having 5 pins for SPI interface (pins 19, 21, 23 for signals, and 24 and 25 for device selection), 8 pins for GPIO, 2 pins for UART, 2 pins for I2C, 2 pins for 3.3V supply, 2 pins for 5 V supply and 5 ground pins, so total number of pins are 5+8+2+2+2+2+5 = 26 pins, as shown in figure 3.11.

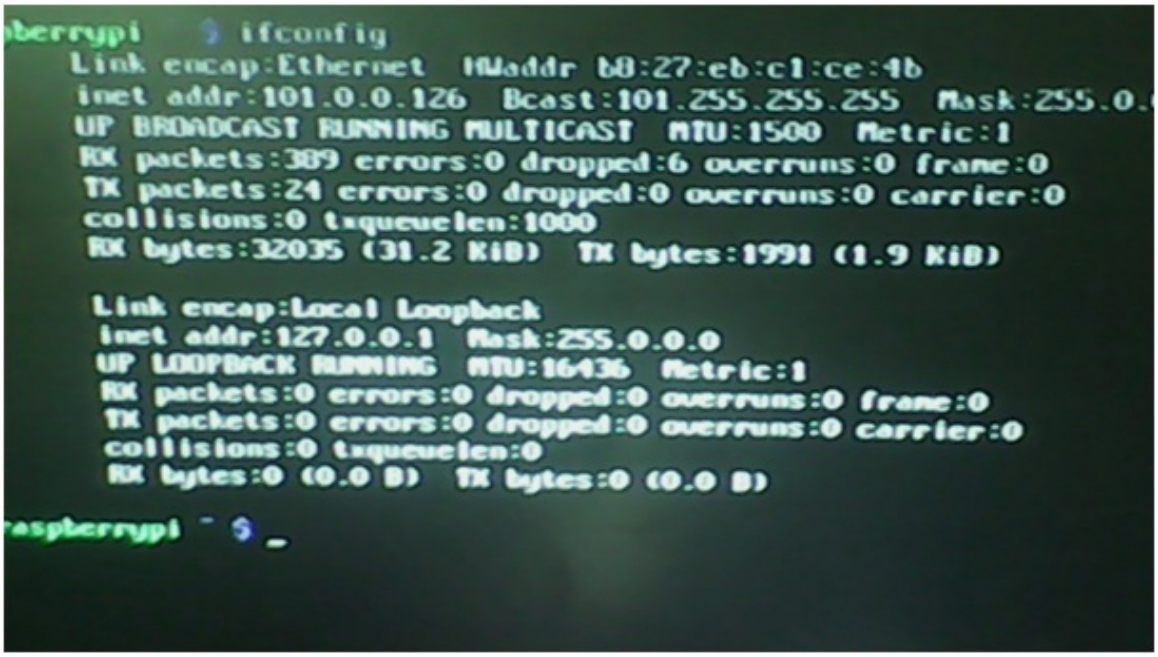

Figure 3.9: Output of ifconfig

3.1.7 Config.txt

The config.txt file can control almost all aspects of the Pis hardware, with the exception of the way the central processing unit (CPU) and graphics processing unit (GPU) sections of the BCM2835 apportion the memory. The config.txt file is only read when the system first starts up. Any changes made while the Pi is running wont take effect until the system is restarted, or switched off and back on again. In the event that the changes are unwanted, simply deleting the file from the /boot directory should be enough to restore the defaults once more. If the Pi wont boot with new settings, just remove the SD card and delete config.txt from the boot partition on another PC, and then reinsert it into the Pi and try again. Without making any edition in a new Raspberry Pi board, a blank configuration file is observed. One has to configure it as per application requirement. Parameters can be handled with the help of config.txt are listed below.

- Enabling and disabling test mode
- Enabling and disabling L2 cache (128kb)
- Overvoltage setting
- Over clocking setting
- Boot option
- Display configuration

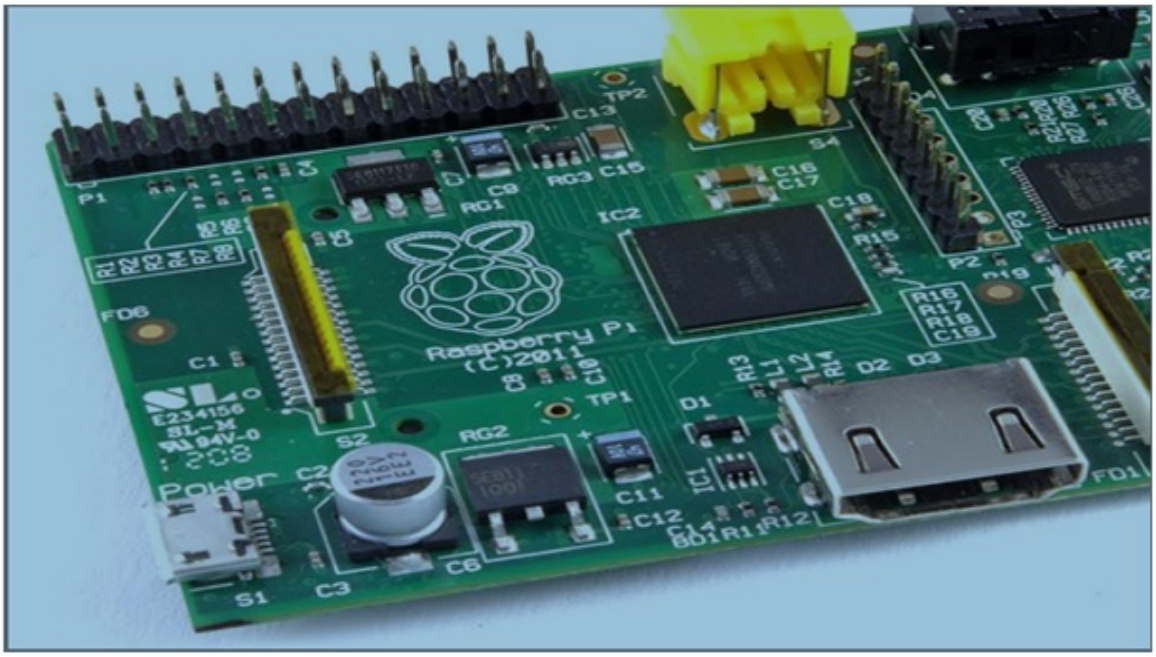

Figure 3.10: Two voltage test points, labeled TP1 and TP2

Over clocking, or overvoltage setting may be a cause of break down in current working of the Raspberry Pi system.

3.2 ADC interface and Programming

ADC MCP3208 works with SPI protocol for data communication.It has 8 input channel and inbuilt sample and hold circuit. The V_cc and ground are connected to the V_cc and ground of the raspberry Pi to reduce supply rail noise.

Table I: Connection between MCP3208 and Raspberry Pi

Pin no. of MCP3208	Pin no. of GPIO of RPi
13	23
12	21
11	19
10	24

3.2.1 Header file for CG calculation

Amazing thing about CG is that, it needs the total weight of the truck on the platform to calculate exact place of CG even after that it is free of the weight of that truck.

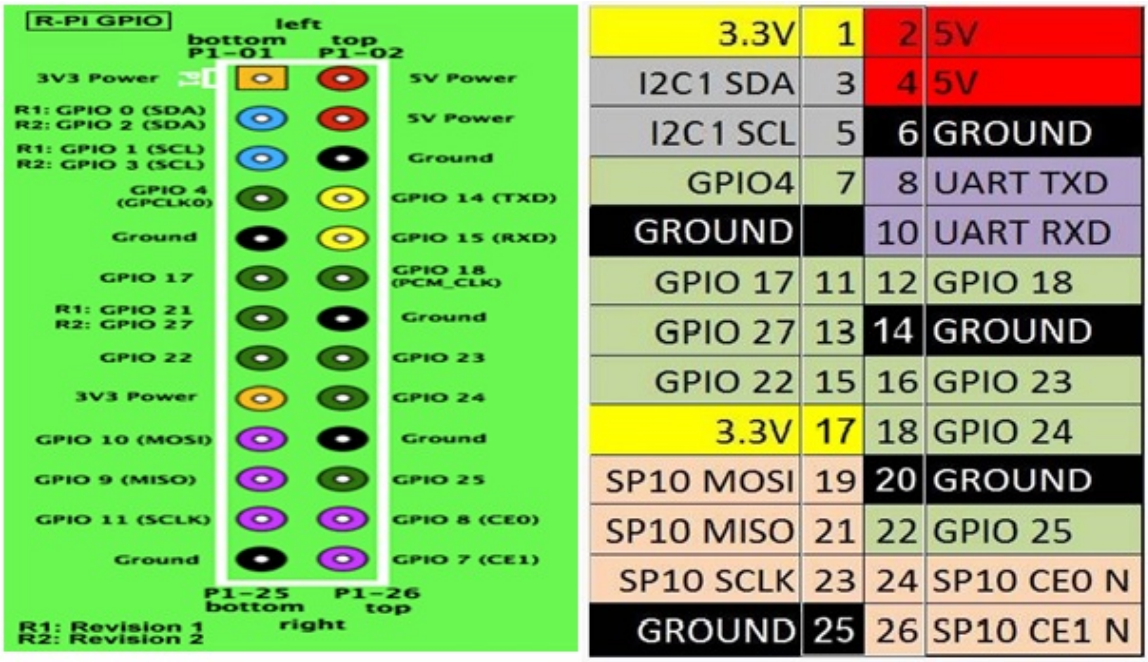

Figure 3.11: GPIO layout for Raspberry Pi

Distance of each load cells from a reference point is taken in consideration. As well with it load measured on each individual load cell is also considered.

$$CG = \frac{\Sigma wi * xi}{\Sigma wi}$$

where,
wi = weight measured on the i'th load cell.
xi = distance of load cell i, from the reference point.

Python is used to setup communication. Readings from different channels are taken as input to the CG calculation function and output will be the x-coordinate of the CG from reference point.

3.3 System Integrity

Reference voltages for MCP3208 are set to 5V causes lsb $= \frac{5}{4096}v = 1.2mv$. Maximum load capacity of the load cell is 4535 kg = 10000 lbs. Hence it shows 30mV output at 4535 kg weight and 10 V excitation. So it shows $\frac{30mv}{4535Kg} = 0.0066\frac{mv}{Kg}$. Thus gain of the amplifier stage should be greater than $\frac{1.2mv}{0.0066mv} = 180$ for amplifier unit. This calculation is good enough for the system with one load cell. But weight of truck

is being distributed on the total platform, causes all the load cells to share an amount of weight and not the whole. Consider a balance of 1 Kg. If it is placed on the platform, the total weight is shared by eight load cell, with no load cell having exact of 1 Kg. of weight causes no lsb increment for ADC and weight will not be calculated. To come out of such trouble, gain of amplifier must be higher, so for even a little amount of weight on platform, a significance change in lsb of ADC can be observed. That creates an ease to measure the weight of a truck, with proper programming.

3.4 Summary

By properly configured keyboard, one can write a c program for GPIO using wiringPi library and can compile it without errors. To make network accessibility easy, a provision of a static IP address is done to the RPi module and it will be loaded within module at the time of booting, from /etc/network/interfaces file. To enable SPI device driver, according changes should be done into /modprobe.d/raspi-blacklist.conf' file. This is how the configuration of RPi for very basic operations and required functionalities can be done.System integrity shows the factors to select the gain of differential amplifier and calibration parameter. A header file for CG calculation in python is made and used by import command in the main file. Details of interconnections, timing of communication and protocols are discussed in the following chapters.

Chapter 4

Interconnection of peripherals

Input to the Raspberry Pi and output from it, are expects to be handled, and for that, different protocols are used. GPIO of Raspberry Pi having 26 pins among them, 5 pins are dedicated to Serial Peripheral Interface (SPI) protocol, 2 pins are dedicated to Universal Asynchronous Receive Transmit, (UART) , and 2 pins are dedicated to I2C protocol. Handling of the data transmission and reception on this pins need some drivers or packages to be installed in Raspberry Pi. As the device changes, the type of interface getting changed, likewise RF ID card reader is being interfaced with UART protocol, while ADC MCP3208 is being interfaced with SPI protocol, and camera switcher and lighting controller are being interfaced with I2C protocol. Hence a higher complexity arises at the hardware side, while drivers configuration should be at booting time, from software. This chapter consists these protocols in detail, with respect to Raspberry Pi module.

4.1 Serial Peripheral Interface (SPI) protocol

The Serial Peripheral Interface bus or SPI bus is a serial data link used for synchronous data communication, for digital data and it is named by Motorola. It operates in full duplex mode. In SPI protocol, devices communicate in master/slave mode, where the data frame is initiated by the master device. Multiple slave devices are allowed without addresses provided to them, as individual slave select lines provided to each slave device. Sometimes, SPI is called a four-wire serial bus, contrasting with three, two, and one wire serial buses. SPI is often referred to as SSI (Synchronous Serial Interface) also.

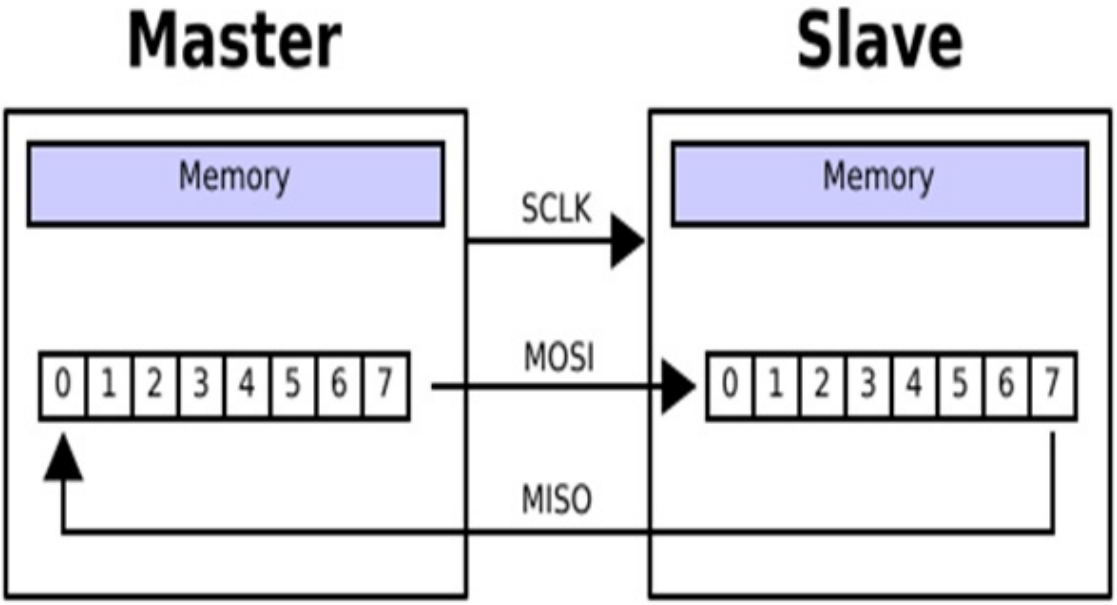

Figure 4.1: Master Slave interconnection in SPI protocol

4.1.1 SPI with multiple slaves

Common MOSI, MISO, and SCLK line is shared among all the slaves, while an individual SS (slave select) line is taken out from master for each slave. If n slaves are attached to a master then total 3+n lines are drawn out of the master as shown in figure 4.3. Some time clock skew may be observed as the distance or device increases, and hence with limited number of slaves it works better, but no any rigid boundary for upper limit of slave number.

4.1.2 SPI signaling

SPI interface uses four connections for one to one device (master to a single slave) connection, these four connection unable a two way communication between master and slave.

- MOSI master out slave in
- MISO master in slave out
- SCK serial clock

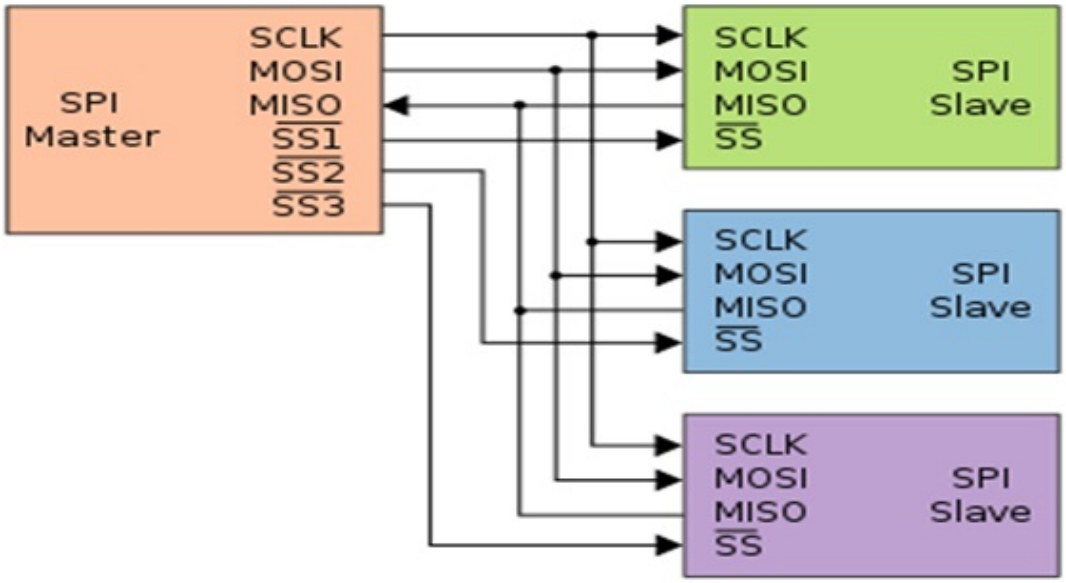

Figure 4.2: Master Slave interconnection in SPI protocol with multiple slaves

- SSn slave select

Communication by SPI protocol can take place with assertion of SSn, or say a chip select signal. (refer to figure 4.2). It generally transits from high to low. When chip-select (CS) or SSn set to low, the SPI device, connected to it is being selected, for

communication, till the time that it remains in high impedance state. SPI has a clock signal named SCLK, sent from the bus master to all slaves; all the SPI signals are synchronous to this clock signal; while it samples the MISO line As well with it, it has a data line from the master to the slaves, named MOSI (Master Out-Slave In) and another data line from the slaves to the master, named MISO (Master In-Slave Out). SPI is a single-master communication protocol. This means that one central device initiates all the communications with the slaves. When the SPI master wishes to send data to a slave and/or request information from it, it selects slave by pulling the corresponding SS line low and it activates the clock signal at a clock frequency usable by the master and the slave. The master generates information onto MOSI line.

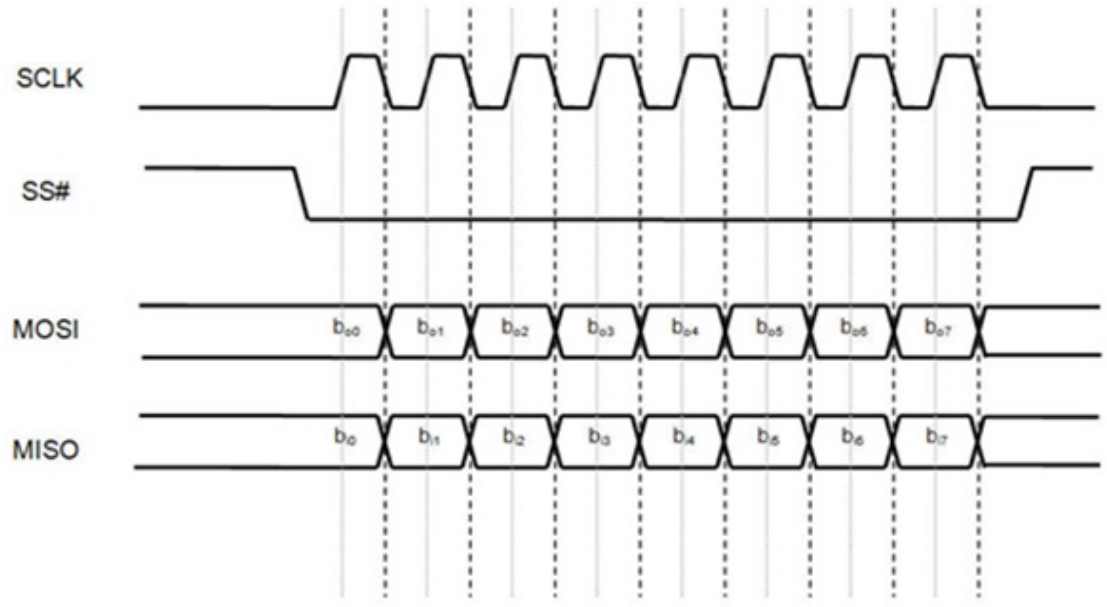

Figure 4.3: signaling in SPI protocol

4.1.3 Modes of communication in SPI

Four communication modes are available (MODE 0, 1, 2, 3) that basically define the SCLK edge on which the MOSI line toggles, the SCLK edge on which the master samples the MISO line and the SCLK signal steady level (that is the clock level, high or low, when the clock is not active). Each mode is formally defined with a pair of parameters called clock polarity (CPOL) and clock phase (CPHA).

A master/slave pair must use the same set of parameters SCLK frequency, CPOL, and CPHA for a communication to be possible. If multiple slaves are used, that are fixed in different configurations, the master will have to reconfigure itself each time it needs to communicate with a different slave. This is basically all what is defined for the SPI protocol. SPI does not define any maximum data rate, not any particular addressing scheme; it does not have an acknowledgment mechanism to confirm receipt of data and does not offer any flow control. Actually, the SPI master has no knowledge of whether a slave exists, unless something additional is done outside the SPI protocol. SPI does not care about the physical interface characteristics like the I/O voltages and standard used between the devices. Initially, most SPI implementation used a non-continuous clock and byte-by-byte scheme. But many variants of the protocol now exist, that use a continuous clock signal and an arbitrary transfer length.

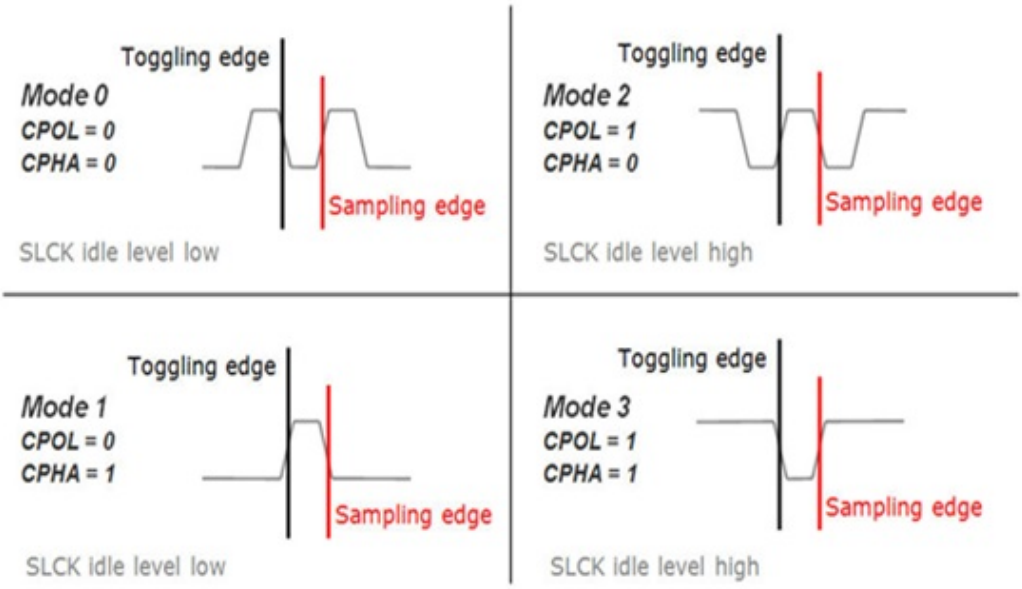

Figure 4.4: modes in SPI protocol

4.1.4 Enabling Spidev (SPI device driver) on the Raspberry Pi

Following steps are followed to enable SPI driver, (Spidev) on the Raspberry Pi

- Turn on the Raspberry Pi and connect it to the network
- Log in to the Raspberry Pi remotely over SSH.
- Open the raspi-black-list.conf file using the following command : sudo nano /etc/modprobe.d/raspi-blacklist.conf
- Comment out the blacklist spi-bcm2708 entry by putting a hash sign in front of it. So it looks like blacklist spi-bcm2708.
- Type sudo reboot. This should cause the Raspberry Pi to reboot. Which will terminate the SSH session.

Start a new SSH session and type ls /dev/spidev* If you see the output displayed in Figure 4.5, then the SPI driver (spidev) has been successfully enabled on the Raspberry Pi.

Figure 4.5: Display of Spidev

4.1.5 Communication with MCP3208

Communication with the MCP3208 device is accomplished using a standard SPI-compatible serial interface. Initiating communication with either device is done by bringing the CS line low. If the device was powered up with the CS pin low, it must be brought high and back low to initiate communication.This will be taken care by Spidev0.0 driver.

The first clock received with CS low and DIN high will constitute a start bit. The SGL/DIFF bit follows the start bit and will determine if the conversion will be done using single-ended or differential input mode. The next three bits (D0, D1 and D2) are used to select the input channel configuration. The device will begin to sample the analog input on the fourth rising edge of the clock after the start bit has been received. The sample period will end on the falling edge of the fifth clock following the start bit.

Once the D0 bit is input, one more clock is required to complete the sample and hold period (DIN is a don't care for this clock). On the falling edge of the next clock, the device will output a low null bit. The next 12 clocks will output the result of the conversion with MSB first. Data is always output from the device on the falling edge of the clock. If all 12 data bits have been transmitted and the device continues to receive clocks while the CS is held low, the device will output the conversion result LSB first. If more clocks are provided to the device while CS is still low (after the LSB first data has been transmitted), the device will clock out zeros indefinitely. If

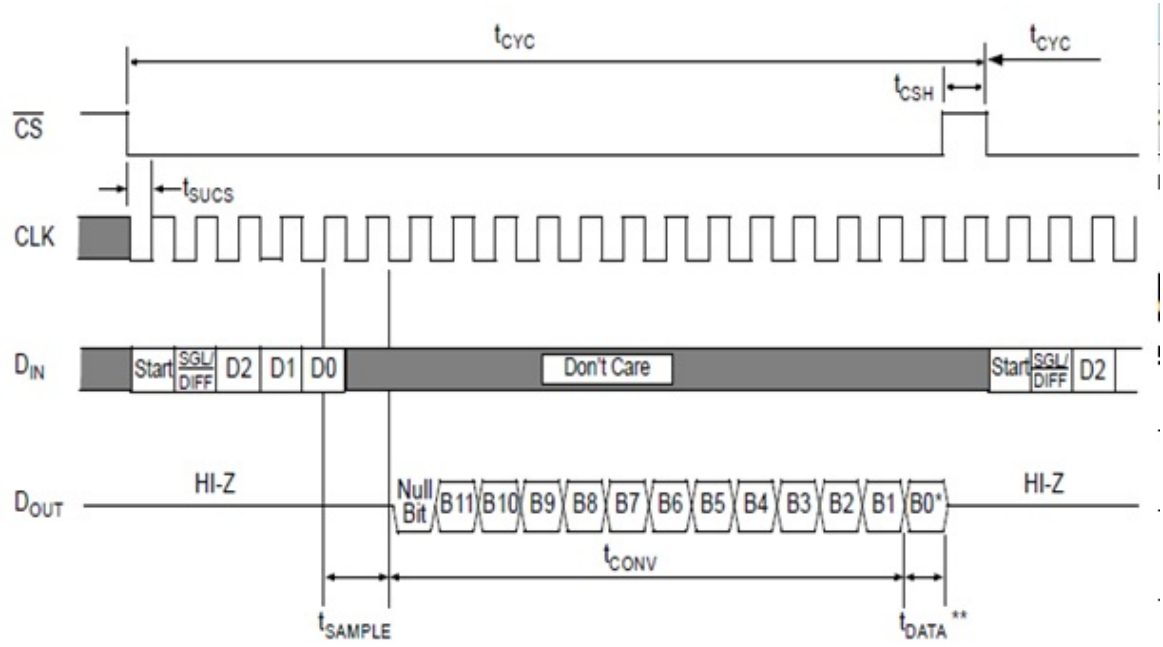

Figure 4.6: Communication with MCP3208

necessary, it is possible to bring CS low and clock in leading zeros on the DIN line before the start bit. This is often done when dealing with micro-controller-based SPI ports that must send 8 bits at a time.

4.1.6 Advantages of SPI

- Full duplex communication
- Higher throughput than IC or SMBus
- Complete protocol flexibility for the bits transferred
  - Not limited to 8-bit words
  - Arbitrary choice of message size, content, and purpose
- Extremely simple hardware interfacing
  - Typically lower power requirements than IC or SMBus due to less circuitry (including pull up resistors)
  - No arbitration or associated failure modes
  - Slaves use the master's clock, and don't need precision oscillators
  - Slaves don't need a unique address unlike IC or GPIB or SCSI

  - Transceivers are not needed

- Uses only four pins on IC packages, and wires in board layouts or connectors, much fewer than parallel interfaces
- At most one unique bus signal per device (chip select); all others are shared
- Signals are unidirectional allowing for easy Galvanic isolation
- Not limited to any maximum clock speed, enabling potentially high throughput

4.1.7 Disadvantage of SPI

- Requires more pins on IC packages than IC, even in the three-wire variant
- No in-band addressing; out-of-band chip select signals are required on shared buses
- No hardware flow control by the slave (but the master can delay the next clock edge to slow the transfer rate)
- No hardware slave acknowledgment (the master could be transmitting to nowhere and not know it)
- Supports only one master device
- No error-checking protocol is defined
- Generally prone to noise spikes causing faulty communication
- Without a formal standard, validating conformance is not possible
- Only handles short distances compared to RS-232, RS-485, or CAN-bus
- Many existing variations, making it difficult to find development tools like host adapters that support those variations
- SPI does not support hot plugging (dynamically adding nodes).

4.2 Inter- integrated circuit Protocol (I2C)

IC (Inter-Integrated Circuit, referred to as I-squared-C, I-two-C, or IIC) is a multi-master serial single-ended computer bus invented by Philips used for attaching low-speed peripherals to a motherboard, embedded system, cell phone, or other electronic device. Not to be confused with the term Two Wire Interface which only describes a compatible hardware interface? Since the mid-1990s, several competitors (e.g.,

Siemens AG (later Infineon Technologies AG, now Intel mobile communications), NEC, Texas Instruments, STMicroelectronics (formerly SGS-Thomson), Motorola (later Freescale), Intersil, etc.) brought IC products on the market, which are fully compatible with the NXP (formerly Philips' semiconductor division) IC-system. Since October 10, 2006, no licensing fees are required to implement the IC protocol. However, fees are still required to obtain IC slave addresses allocated by NXP. SMBus, defined by Intel in 1995, is a subset of IC that defines the protocols more strictly. One purpose of SMBus is to promote robustness and interoperability. Accordingly, modern IC systems incorporate policies and rules from SMBus, sometimes supporting both IC and SMBus with minimal reconfiguration required.

4.2.1 Revision in I2C

- In 1982, the original 100-kHz IC system was created as a simple internal bus system for building control electronics with various Philips chips.
- In 1992, Version 1.0 (the first standardized version) added 400-kHz Fast-mode (Fm) and a 10-bit addressing mode to increase capacity to 1008 nodes.
- 1998, Version 2.0 added 3.4-MHz High-speed mode (Hs) with power-saving requirements for electric voltage and current.
- In 2000, Version 2.1 introduced a minor cleanup of version 2.0.
- In 2007, Version 3.0 added 1-MHz Fast-mode plus (Fm+), and a device ID mechanism.
- In 2012, Version 4.0 added 5-MHz Ultra Fast-mode (UFm) for new USDA and USCL lines using push-pull logic without pull-up resistors, and added assigned manufacturer ID table. This is the most recent standard.

4.2.2 Design of I2C and signaling

A sample schematic with one master (a micro-controller), three slave nodes (an ADC, a DAC, and a micro-controller), and pull-up resistors Rp. IC uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock (SCL), pulled up with resistors. Typical voltages used are +5 V or +3.3 V although systems with other voltages are permitted.

The IC reference design has a 7-bit or a 10-bit (depending on the device used) address space. Common IC bus speeds are the 100 kbit/s standard mode and the 10 kbit/s low-speed mode, but arbitrarily low clock frequencies are also allowed. Recent revisions of IC can host more nodes and run at faster speeds (400 kbit/s Fast mode, 1 Mbit/s Fast mode plus or Fm+, and 3.4 Mbit/s High Speed mode). These speeds are

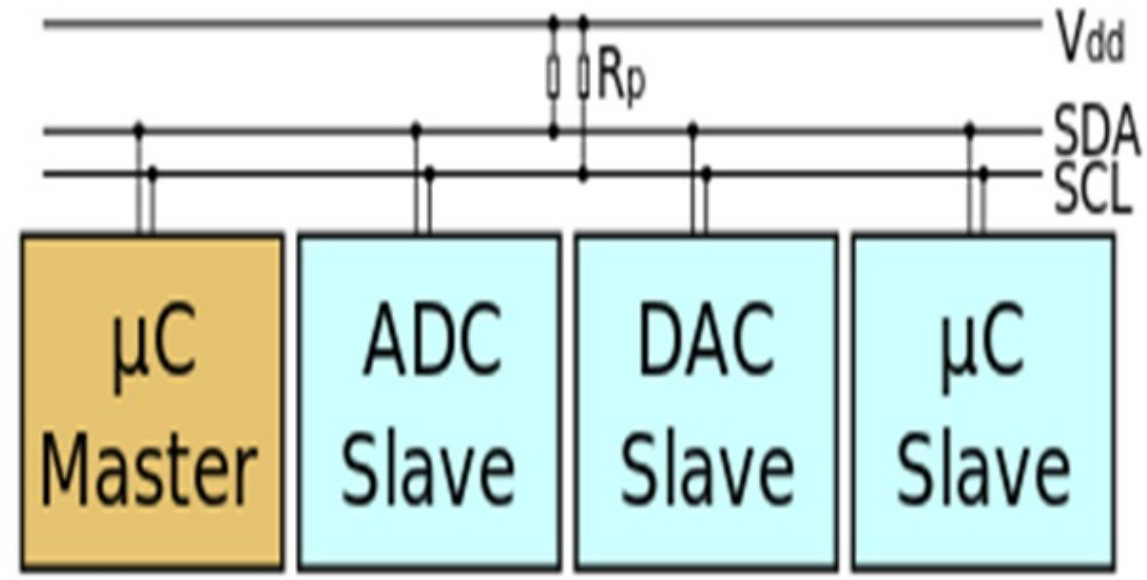

Figure 4.7: Design of I2C protocol

more widely used on embedded systems than on PCs. There are also other features, such as 16-bit addressing. The bit rates are quoted for the transactions between master and slave without clock stretching or other hardware overhead. Protocol overheads include a slave address and perhaps a register address within the slave device as well as per-byte ACK/NACK bits. Thus the actual transfer rate of user data is lower than those peak bit rates alone would imply. For example, if each interaction with a slave inefficiently allows only 1 byte of data to be transferred, the data rate will be less than half the peak bit rate. The maximum number of nodes is limited by the address space, and also by the total bus capacitance of 400 pF, which restricts practical communication distances to a few meters.

4.2.3 Reference design

The before mentioned reference design is a bus with a clock (SCL) and data (SDA) lines with 7-bit addressing. The bus has two roles for nodes: master and slave:

- Master node node that generates the clock and initiates communication with slaves
- Slave node node that receives the clock and responds when addressed by the master

The bus is a multi-master bus which means any number of master nodes can be present. Additionally, master and slave roles may be changed between messages (after

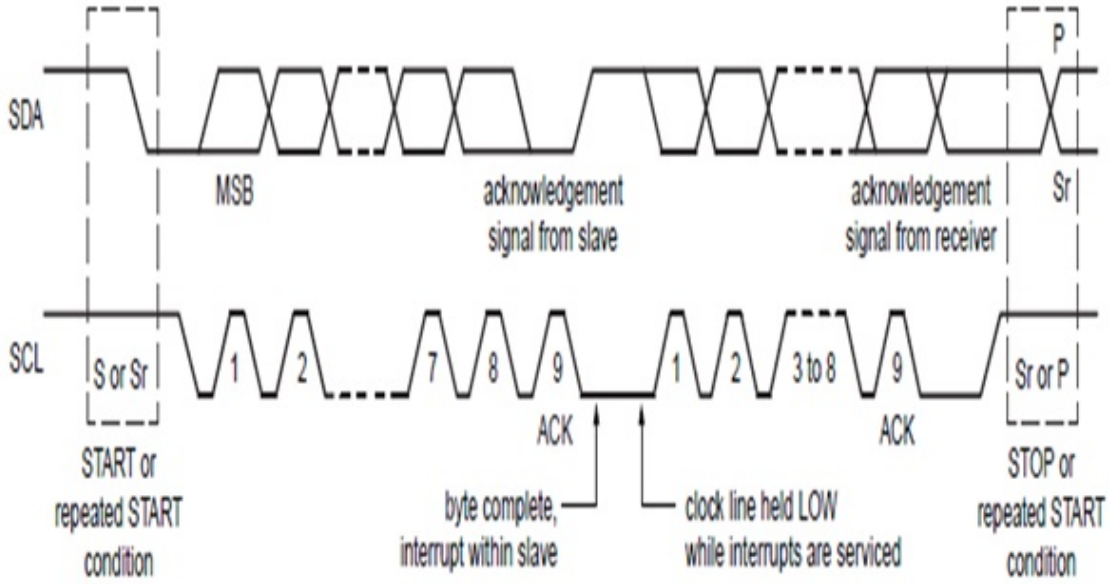

Figure 4.8: Data communication in I2C protocol

a STOP is sent). There are four potential modes of operation for a given bus device, although most devices only use a single role and its two modes:

a. master transmit master node is sending data to a slave

b. master receive master node is receiving data from a slave

c. slave transmit slave node is sending data to the master

d. slave receive slave node is receiving data from the master

The master is initially in master transmit mode by sending a start bit followed by the 7-bit address of the slave it wishes to communicate with, which is finally followed by a single bit representing whether it wishes to write(0) to or read(1) from the slave. If the slave exists on the bus then it will respond with an ACK bit (active low for acknowledged) for that address. The master then continues in either transmit or receive mode (according to the read/write bit it sent), and the slave continues in its complementary mode (receive or transmit, respectively). The address and the data bytes are sent most significant bit first. The start bit is indicated by a high-to-low transition of SDA with SCL high; the stop bit is indicated by a low-to-high transition of SDA with SCL high. All other transitions of SDA take place with SCL low. If the master wishes to write to the slave then it repeatedly sends a byte with the slave sending an ACK bit. (In this situation, the master is in master transmit mode and the slave is in slave receive mode.) If the master wishes to read from the slave then it

repeatedly receives a byte from the slave, the master sending an ACK bit after every byte but the last one. (In this situation, the master is in master receive mode and the slave is in slave transmit mode.) The master then either ends transmission with a stop bit, or it may send another START bit if it wishes to retain control of the bus for another transfer (a "combined message").

4.2.4 Message protocol

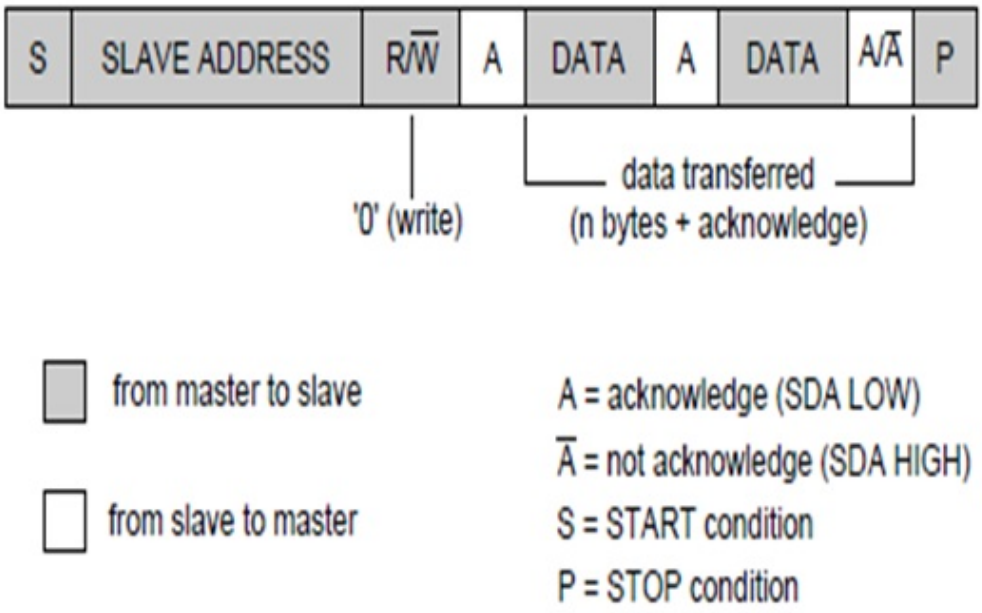

Figure 4.9: I2C Message protocol

IC defines basic types of messages, each of which begins with a START and ends with a STOP:

- Single message where a master writes data to a slave;
- Single message where a master reads data from a slave;
- Combined messages, where a master issues at least two reads and/or writes to one or more slaves. In a combined message, each read or write begins with a START and the slave address. After the first START in a combined message these are also called repeated START bits. Repeated START bits are not preceded by STOP bits, which is how slaves know the next transfer is part of the same message. Any given slave will only respond to particular messages, as defined by its product documentation. Pure IC systems support arbitrary

message structures. SMBus is restricted to nine of those structures, such as read word N and write word N, involving a single slave. PMBus extends SMBus with a Group protocol, allowing multiple such SMBus transactions to be sent in one combined message. The terminating STOP indicates when those grouped actions should take effect. For example, one PMBus operation might reconfigure three power supplies (using three different I2C slave addresses), and their new configurations would take effect at the same time: when they receive that STOP. With only a few exceptions, neither IC nor SMBus define message semantics, such as the meaning of data bytes in messages. Message semantics are otherwise product-specific. Those exceptions include messages addressed to the IC general call address (0x00) or to the SMBus Alert Response Address; and messages involved in the SMBus Address Resolution Protocol (ARP) for dynamic address allocation and management. In practice, most slaves adopt request/response control models, where one or more bytes following a write command are treated as a command or address. Those bytes determine how subsequent written bytes are treated and/or how the slave responds on subsequent reads. Most SMBus operations involve single byte commands.

4.2.5 Configure Raspberry Pi for I2C protocol

Raspberry Pi is ready to go with I2C as far as enabling the hardware goes. However, while using Raspbian, open LXTerminal and enter the following command

sudo nano /etc/modules

and add these two lines to the end of the file:

- i2c-bcm2708
- i2c-dev

After editing the file, reboot for the changes to take effect is necessary. If there are problems with I2C on Raspbian, then it is well worth updating to the latest version.The I2C bus allows multiple devices to be connected to Raspberry Pi, each with a unique address, which can often be set by changing jumper settings on the module. It is very useful to be able to see which devices are connected to the Raspberry Pi as a way of making sure everything is working. To do this, it is worth running the following commands in the Terminal to install the i2c-tools utility.

sudo apt-get install python-smbus
sudo apt-get install i2c-tools

Figure 4.10: I2C configuration

Depending on Linux distribution, system has a file called /etc/modprobe.d/raspi-blacklist.conf. If systems do not have this file then there is nothing to do, however, if systems do have this file, it is needed to edit it and comment out the lines below:

blacklist spi-bcm2708
blacklist i2c-bcm2708

by putting a hash before it, as same as done in SPI enabling. Once this is all done, following command is used to see all the connected devices (for 512MB Raspberry Pi Model B)

sudo i2cdetect -y 1

This shows that two I2C addresses are in use 0x40 and 0x70.

4.3 Universal Asynchronous Receive Transmit Protocol (UART)

The UART block provides serial communication capability with external devices through an RS-232 cable or through use of external circuitry that converts infrared

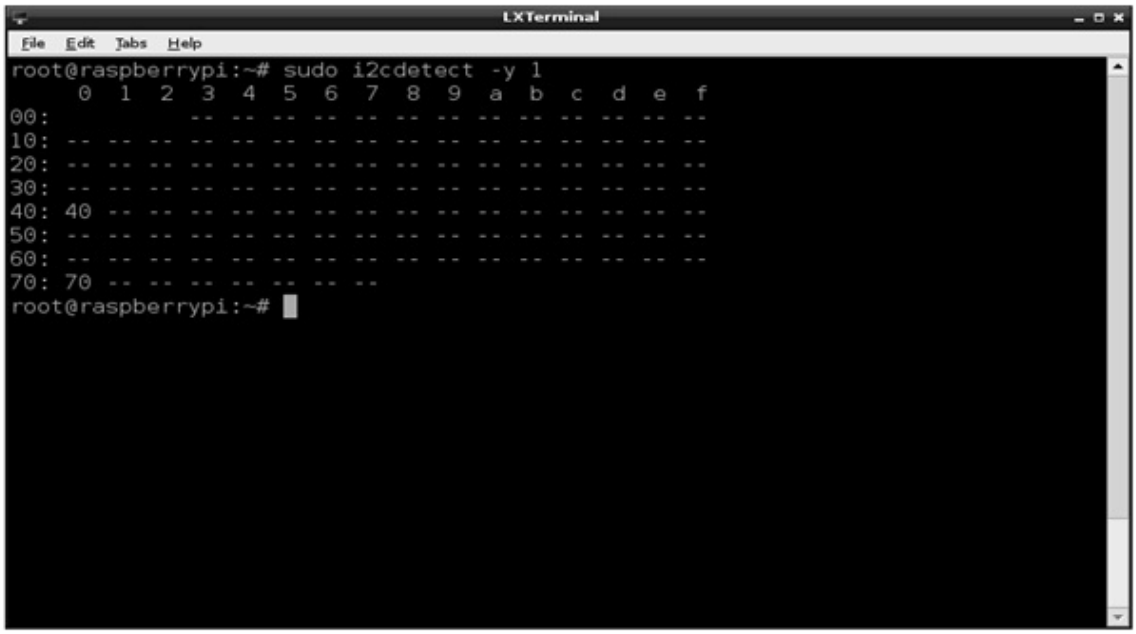

Figure 4.11: I2C addresses

signals to electrical signals (for reception) or transforms electrical signals to signals that drive an LED (for transmission) to provide low speed IrDA compatibility. The UART module supports NRZ encoding format.

4.3.1 Implementation of UART on Raspberry Pil

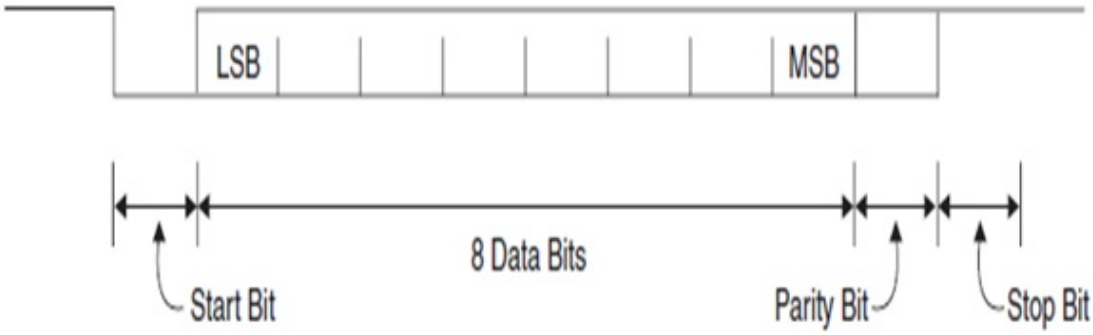

Figure 4.12: UART Protocol

In order to use the dedicated UART pins on the raspberry pi, first they have to be removed from their default application which is debugging. To do this edit "/boot/cmdline.txt" and "/etc/inittab". Backup this files if you want to return to the default configuration

cp /boot/cmdline.txt /boot/cmdline.bak
cp /etc/inittab /etc/inittab.bak

Remove console=ttyAMA0, 115200" and "kgdboc=ttyAMA0, 115200" configuration parameters from the "/boot/cmdline.txt" configuration file using nano editor.

sudo nano /boot/cmdline.txt

Comment the last line on the "/etc/inittab" file. Put a hash before "T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100.

sudo nano /etc/inittab

Now the RXD (GPIO15) and TXD (GPIO14) pins are available for general UART use. This is an example application to use the UART pins using Python and the pyserial library. To install Pyserial download the latest version from

http://sourceforge.net/projects/pyserial/files/pyserial/

Install by running the setup.py

python setup.py install

Open a Python terminal (remember to always be running under SuperUser by using sudo or su Python. To test it without the need of another device simply connect raspberry's TXD and RXD pins to each other and run the following commands on the python terminal:

textbfimport serial textbfser = serial.Serial ("/dev/ttyAMA0") textbfser.write("UART the Font") textbfread = ser.read () textbfprint read textbfser.close()

Install "minicom" for terminal emulation:

apt-get install minicom

minicom can be used by the next line command

minicom -b 9600 -o -D /dev/ttyAMA0

4.3.2 Troubleshooting UART Problems

The above code works (we've used it for TX and RX) nicely. But if it does not work for any reason even if all the steps are taken properly to release the UART from being used for the console try the following.

A. Permissions: This command will set read and write access permissions for all users on the UART it shouldn't be needed but can be used just to be sure there is not a permissions problem:

sudo chmod a+rw /dev/ttyAMA0

B. Baud Rate Error: Try using a slower BAUD rate (or a single 0xFF byte which only has the start bit low) and see if it works. It may have a problem using 115k2 baud rate where our micro-controller communicating with the RPi could hit 113636baud or 119047baud. 113636baud had the lowest error margin. However when transmitting to the Raspberry Pi nothing was ever received. Changing the micro-controller to use 119047baud caused RX to work.

4.4 Errors in Communication

Accuracy is one of the most critical factors to consider when specifying an ADC for test and measurement applications. Unfortunately, it's often confused with resolution, and although related, they are distinctly different. The ADC accuracy is the percentage error in the converted value while resolution is the smallest value into which the full range ADC reading is divided. Every ADC measurement contains a variety of unavoidable, independent errors that influence its accuracy. When,σ represents each independent error, the total error can be shown as,

$$\sigma_{total} = \sqrt{\Sigma\sigma^2} \tag{4.1}$$

This equation includes a variety of errors such as sensor anomalies, noise amplifier gain and offset, ADC quantization (resolution) error, and other factors. Accuracy of a data acquisition (DA) device e.g. an ADC is defined as the sum of three components stated in terms of the reading, range, and the least significant bit (LSB). It is a measure of the extend to which the device is error-free. Generally, accuracy is not constant over an entire measurement range; it varies with the reading magnitude. On the other hand, resolution is defined as the smallest incremental change the ADC can recognize. For example, a 12-bit ADC has a resolution of 1 part in 4096.

4.4.1 Solutions for Errors

- Rail voltages must be stable. A capacitor across V_{CC} and ground should be connected.

- V_{REF} must be stable.
- According to operation frequency, R-C filter should be applied at I/O of ADC.
- Software based triple module system should be implement to get error free output, bt it comprises the time.

4.5 Summary

By implementing these protocols for Raspberry Pi system, an easy interface for data communication can be established with the inbuilt libraries as well with their functions. A circuit level complexity increases as all the protocols have to be developed for the same board and have to work simultaneously, without affecting one another's performance. By reducing the hardware complexity with I2C and to built an easy communication way with SPI, inbuilt functions works tremendously good.Methods described in this chapter to stabilize the ADC output are required to be implemented to get proper output, else false weight will be considered. In next chapter, all results of implemented systems in comparison with a standard system is shown.

Chapter 5

Analysis and Results

As far as the implementation of this flow concern, results obtained are discussed here. A Savior standard weighbridge is considered as a reference product and the results will be observed in comparison with that. Analysis is in term of CG deviation from standard and weight difference in real and measured weight.

5.1 Analysis of CG

Unit of the CG of the system is Meter, and calculated from the first edge of the weighbridge as shown here. A Savior standard weighbridge and a system developed with Raspberry Pi is compared for observing the deviation of CG

Table I: Analysis of CG

Original	Savior System	Deviation in %	RPi based System	Deviation in %
5	5.01	2	5.05	10.0
10	10.02	2.0	10.08	8.0
15	15.04	2.0	15.06	4.0
20	20.06	3.0	20.06	3.0
25	25.08	3.2	25.04	1.6
30	30.10	3.3	30.04	1.3
35	35.11	3.1	35.02	0.5
40	40.12	3.0	40.01	0.25

All dimensions are in Meter

So the analysis shows, that as the distance from reference increases, the RPi based system shows more accurate CG than the standard one. Total length of weigh bridge is 45 meter.

5.2 System Parameters

System is being considered with the parameters like Space, flexibility, Range resolution and many more. This will be considered as a part of comparison with standard system.

5.2.1 Number of IC

Savior weighbridge system uses 1 voltage regulator, and 1 ADC for each load cell. Here MCU is not considered, as is it using Pentium-4 processor in built with PC. For data communication. Rather than MCU there are 4 ICs used per load cell, hence total 32 ICs are used in standard product. In Raspberry Pi based system developed here, a common ADC is used for all load cell, hence average 3 ICs are used per load cell, hence total ICs (except MCU) are 24 in this system.

5.2.2 Space

In terms of space, the developed system required least space for both the digital circuitry and the analog circuitry. This is attributed to the fact that minimal number of chips was used and to crown it all, the system will be implemented on PCB boards. In addition, data is serially sent from the 12-bit ADC to the Raspberry Pi, only for lines are used to interface, making the circuit design easier. Owing to the many external chips used, in standard Savior weighbridge system, a lot of space is required for circuit implementation.

5.2.3 Portability

From the portability perspective, the developed system has unlimited range as all that is needed is a single external power supply. It has an in-built regulated power supply and works well with the mains power supply. Moreover, it has been designed to make it as compact as possible. Standard Savior weighbridge is bulky and therefore has limited portability.

5.2.4 Range and resolution

The developed electronic weighbridge has tried to address both resolution and range. It is the best in as far as striking a balance between range and resolution goes. This is a modest system with a reasonable range while at the same time being sensitive enough. This is quite rare in virtually all-electronic weighbridge systems.

5.3 Summary

Here a comparison of the raspberry Pi based developed system and standard Savior weighbridge system is done. Observation says, as distance from reference increases, the Raspberry Pi based developed system is more accurate than the standard Savior system, shows the accuracy of header file for CG calculation and accurate working of ADC. As well with that, the developed system is more better in term of space utilization, flexibility, resolution and range.

Chapter 6

Conclusion and future work

6.1 conclusion

- With the help of SOC board like Raspberry Pi, integration of multiple peripherals is easy, more reliable system can be established for such huge scale operation, without human intervention.
- Real time operation required transmitting image on the network, compression techniques must be used but if time taken to compress image is higher than the transmission time, then row image transmission should be done but it may cause more memory utilization.
- Implementation with SOC integration causes the system more efficient.

6.2 Future work

- Automatic weighment even of a moving truck.
- Reducing the queue of trucks at weighbridge.
- Optimized use of space in plant.
- Remote access will be provided for fault recovery, with networking.

Appendix A

Double-Ended Shear Beam Load Cell

This section describes the basic features of load cell module65058.

Features

- Center-link loaded
- Integral conduit adaptor
- Trade certified for NTEP Class IIIL: 10000 divisions; Class III: 5000 divisions and OIML R60 3000 divisions in 20,000 to 100,000 pounds range
- Sensorgage sealed to IP67 standards
- Factory Mutual System Approved for Classes I, II, III; Divisions 1 and 2; Groups A through G. Also, non-incendive ratings (No barriers!).

- Optional
  - 65058S stainless steel, welded seal version available
  - 65058-TSA companion assemblies for vehicle scales
  - 65069-TWA companion assemblies for vessel weighing
  - Capacities up to 500,000 consult factory

A.1 Application

Application Truck scales

- Railroad track scales
- Precision tank, bin and silo weighing
- evel and inventory monitoring

A.2 Discription

steel, double ended Shear beam load cell. This product is designed for use in certified truck and rail scales and is available in capacities ranging from 10k to 100k lbs. This load cell is rated intrinsically safe by the Factory Mutual System (FM); making it suitable for use in potentially explosive environment. This load cell is certified for legal for trade applications by both American NTEP and International OIML standards.

References

[1] BROADCOM, "BMC2835 ARM PERIPHERALS GUIDE", 06 February 2012.

[2] RS components, "Raspberry Pi getting started guide Vsn 1.0", March 2012.

[3] Eben Upton, "Raspberry Pi user guide", March 2012.

[4] Vishay Precision Group, Document 11602, load cell model 65058 , 14 June 2012.

[5] Microchip, Data sheet MCP3208, DS21298C, 2012.

[6] Munyao Kivati,"Design AND fabrication of a micro-controller based electronic weighing machine in high mass regime.", School of Pure And Applied Sciences of Kenyatta University.August 2009.

[7] hysics, standard-12 Gujarat State Board of higher Secondary Board, "Chapter-6 Center of Gravity", April 2007.

[8] Vehicle Technology Market Report" Oak Ridge National Laboratory, Center for Transportation Analysis, Oak Ridge, TN. Weight category definitions from 49CFR565.6 (2000), 2013.

Index