Development and Implementation of a Low-Cost Data Acquisition System Using Single-Board Computers to Measure Volumetric Water Content

by

Mohamad Idaly Bin Ali

B.Eng. (Hons.), National University of Singapore (2013)

Submitted to the Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the degree of

Master of Engineering in Civil and Environmental Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

© Massachusetts Institute of Technology 2018. All rights reserved.

Signature redacted

Author .....
Department of Civil and Environmental Engineering
May 19, 2018

Signature redacted

Certified by ....
Charles F. Harvey
Professor of Civil and Environmental Engineering
Thesis Supervisor

Signature redacted

Accepted by ....
Jesse Kroll
Professor of Civil and Environmental Engineering
Chair, Graduate Program Committee

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUL 26 2018
LIBRARIES
ARCHIVES

Development and Implementation of a Low-Cost Data Acquisition System Using Single-Board Computers to Measure Volumetric Water Content

by

Mohamad Idaly Bin Ali

Submitted to the Department of Civil and Environmental Engineering
on May 19, 2018, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Civil and Environmental Engineering

Abstract

The work described in this thesis is aimed at exploring the use of single-board computers, specifically the Raspberry Pi platform, to measure volumetric water content of soils or other porous media. We first investigated the different methods to condition and measure signal frequencies. Subsequently, we designed and fabricated a Hardware Attached on Top frequency counter add-on board for the Raspberry Pi based on the reciprocal frequency counting method, and found that this accurately and precisely measures frequency signals from water content reflectometers. We then built a dormant deploy-and-forget sensor system around this hardware. The system will be used at our research field site in Brunei Darussalam to measure volumetric water content of peat. Further work focuses on improvements to the encapsulation of the electronic hardware, and designing a multiplexer-controlled relay board to acquire signals from multiple reflectometers simultaneously.

Thesis Supervisor: Charles F. Harvey
Title: Professor of Civil and Environmental Engineering

Acknowledgements

This thesis forms a small part of my research work for the last five years, beginning at the Center of Environmental Sensing and Modeling (CENSAM) in Singapore, and the Parsons Laboratory of Environmental Science and Engineering in Cambridge, Massachusetts. I would first like to thank my advisor, Professor Charles Harvey, for his guidance, support and trust throughout the course of this research.

In Singapore, I am especially indebted to the brilliant Dr Alexander Cobb, my *other* advisor who became one of my dearest friends. His enthusiasm for the tremendously difficult research in the peatlands has made a hugely positive influence to how I live my life today, and for that I am forever grateful. I would also like to thank the good peatniks at CENSAM: Fuu-Ming Kai, Kang Min Ngo, Siew Chin Chua, Alison Hoyt, Surin Kumar, Haleem Syafiyullah, Stefan Lionar, Suhailah Isnin, Muzafar Rasheed, Ezra Hu, Robin Yeo, Jerry Wu, Ivan Lee, Hayley Win, Kenneth Sin, Rachel Lam, Loo Lian Kiat, Toh Yan Han, Rayner Ng, Joeie Lim, Goh Hung Soon, Krishna Vivek and Tee Ming Hui for their support in this endeavor.

My work in the peatlands could not have progressed to what it is today if not for the fantastic team in Brunei. Pudek Ramasamy, Jeffrey Boy, Haji Bohari Idi, Khalish Hafizhah and Jangarun Eri have been my literal pillars of support in the field. I would have crushed under the immense weight of my ridiculously heavy backpack if it were not for them.

I thank the people at Parsons Laboratory for their support, particularly Ishita Shrivastava, Cindy Wang and Irene Hu for keeping me sane throughout my hectic MIT life.

To Fernanda Basso, Benjamin Arenas, Rodrigo Urruela, Anne Gehre, Thomas Kopfmuller, Teo Jun Hao, Nick Yeo, Enzo Peroni and Alexis Charalampopoulus, thanks for scrapping me off the ground and putting me back together again after a difficult start to life in Cambridge.

And lastly, I would like to dedicate my work to Mama and Papa, and my siblings, Idly and Idalia. Growing up in a tough environment, never have I expected to go to college,

let alone be here at MIT. I owe it all to their warm and unconditional love.

Contents

Abstract **iii**

Acknowledgements **v**

1 Introduction **1**

1.1 Background 1

1.2 Current Research and Project Motivation 1

1.3 Objectives 3

2 System Architecture **5**

2.1 Introduction 5

2.2 Hardware Architecture 5

2.2.1 Campbell Scientific CS616 Water Content Reflectometer 5

Operation 6

Installation 6

Electrical Specifications and Wiring Configuration 7

2.2.2 Raspberry Pi 7

2.2.3 CD74HC4067 Multiplexer 10

Operation 10

Pin Configuration, Functional Diagram and Package 12

2.2.4 SN74HC14N Hex Schmitt Trigger Inverter 13

Operation 13

Pin Configuration, Functional Diagram and Package 15

2.2.5 SN74LV8154 Dual 16-Bit Binary Counters 15

Frequency Counting Methods 16
SN74LV8154 Binary Counter 20
Pin Configuration, Functional Diagram and Package 22
2.2.6 MCP23017 16-Bit I/O Expander with Serial Interface 23
Operation 23
2.3 Software Platform 25
2.3.1 Raspbian Operating System for Raspberry Pi 25
2.3.2 Inter-Integrated Circuit (I2C) Interface 26
Operation 27

3 Design **29**
3.1 Introduction 29
3.2 System Design 29
3.2.1 Signal Processing Strategy 29
3.2.2 Multiplexer Program Algorithm and Implementation 31
3.2.3 Signal Conditioning 34
3.2.4 Power Line Conditioning 40
3.2.5 Generating a Clock Signal with Pulse-width Modulation 41
3.2.6 Data Logging 44
3.2.7 Design and Fabrication of HAT Printed Circuit Board (PCB) . . . 45
Computer-aided Representation of Design, and Selection of Electronic Components 45
3.2.8 Board Zoning and PCB Design Guidelines 46
3.2.9 Power Management 48
3.2.10 Real-Time Clock 51
3.2.11 Prototype Packaging 52

4 Evaluation **55**
4.1 Stability of Sensor Readings 55
4.2 Linearity of sensor output data 60

4.3 Performance Evaluation in a Controlled Environment 62

5 Conclusion and Future Work **67**

5.1 Conclusion 67

5.2 Future Work 67

A CS616 Hardware Attached on Top (HAT) **69**

A.1 Schematic of CS616 HAT 69

A.2 Image of CS616 Board 71

Bibliography **73**

List of Figures

2.1 Raspberry Pi 3 B+ 8

2.2 Raspberry Pi Zero 8

2.3 EAGLE Representation of CD74HC4067 12

2.4 Schmitt Functional Diagram 13

2.5 Schmitt Circuit Diagram 14

2.6 Pulse from Square Wave Signal 16

2.7 Block Diagram of a Direct Counter 17

2.8 Block Diagram of a Reciprocal Counter 18

2.9 EAGLE Representation of SN74LV8154 23

2.10 EAGLE Representation of MCP23017 25

2.11 Block Diagram of a System in I2C Configuration 26

2.12 Block Diagram of an I2C Message 27

3.1 Signal Flow of CS616 Volumetric Water Content Data 30

3.2 Flowchart of *int2bin* function 33

3.3 Flowchart of $mux_channel$ Function 34

3.4 Time-Domain Representation of the CS616 Signal (Unfiltered) 35

3.5 Schematic of the connections between the CS616 Volumetric Water Sensor and SN74HC14N Schmitt Trigger 37

3.6 Time-Domain Representation of Unfiltered and Filtered Output Signals from CS616 38

3.7 Frequency-Domain Representation of Unfiltered and Filtered Output Signals from CS616 using Cooley-Tukey Fast Fourier Transform (FFT) 39

3.8 Electronic Device with Bypass Capacitor 40
3.9 Effects of adding a bypass capacitor on a power line 41
3.10 Clock Output from Pulse-Width Modulation 43
3.11 Clock signal input into SN74LV8154 44
3.12 Raspberry Pi HAT PCB Board Layout 47
3.13 Power Management and Supply 49
3.14 Flowchart of Power Management Daemon Service 51
3.15 Internal layout of Soil Moisture Sensing System Prototype 52
3.16 Soil Moisture Sensing System Prototype in Protective Case 53

4.1 Laboratory Set-up 56
4.2 Output signal of CS616 sensor as seen from an oscilloscope 57
4.3 Stability of CS616 Output Signal Readings from HAT (Probes in Air) . . 58
4.4 Output signal of CS616 sensor as seen from an oscilloscope (Probes Fully Submerged in Water) 59
4.5 Stability of CS616 Output Signal Readings from HAT (Probes in Water) . 59
4.6 Linearity: Output signals of CS616 sensor as seen from an oscilloscope and HAT 61
4.7 Linearity: Relationship between true value and measured value of the frequency of CS616 output signal 62
4.8 Laboratory Set-up 63
4.9 Volumetric Content Over Time 65
4.10 States of soil medium throughout experiment 66

A.1 Semi-populated Hardware-Attached-On-Top to measure frequency signals from CS616 sensor 71

List of Tables

2.1 CS616 Volumetric Water Content Reflectometer 7

2.2 Wiring Configuration of CS616 Volumetric Water Content Reflectometer . 7

2.3 Raspberry Pi 3 Specifications 9

2.4 Raspberry Pi Zero Specifications 10

2.5 CD74HC4067 Input/Output Truth Table 11

2.6 Pin Configuration of CD74HC4067 12

2.7 SN74HC14N Pin Configuration 15

2.8 Summary of Storage Register Control Pins 20

2.9 Output Bus Logic of SN74LV8154 21

2.10 Pin Configuration of SN74LV8154 22

2.11 Pin Configuration of MCP23017 24

3.1 Integer to Binary Number Converter 31

3.2 Controlling the Multiplexer 32

3.3 PCB Design Rules 48

3.4 Power consumption of Raspberry Pi B+ and Zero (Geerling, 2015) 50

List of Abbreviations

I2C	**I**nter **I**ntegrated **C**ircuit
CSV	**C**omma **S**eparated **V**alues
MUX	**M**ultiplexer
MSB	**M**ost **S**ignificant **B**it
LSB	**L**east **S**ignificant **B**it
CMOS	**C**omplimentary **M**etal **O**xide **S**emiconductor
VWC	**V**olumetric **W**ater **C**ontent
FFT	**F**ast **F**ourier **T**ransform
DC	**D**irect **C**urrent
GPS	**G**lobal **P**ositioning **S**ystem
PPS	**P**ulse **P**er **S**econd
GPIO	**G**eneral **P**urpose **I**nput **O**utput
SMBus	**S**ystem **M**anagement **Bus**
CAD	**C**omputer **A**ided **D**esign
SMD	**S**urface **M**ounted **D**evice
TH	**T**hrough **H**ole
RF	**R**adio **F**requency

Chapter 1

Introduction

1.1 Background

Tropical peatlands play an important role in the global carbon cycle. They act as long-term carbon sinks for atmospheric carbon dioxide, and are also a significant natural source of methane. At an area of about 25 MHa with an estimated carbon stock of 69 Gt, the island of Borneo, located in Southeast Asia, holds the majority of the world's peatland resource.

In the last few decades, however, these peatlands have undergone extensive land cover change brought upon by clearing, draining and burning in response to energy, agricultural and forestry demand(Cobb et al., 2017). Many areas have been converted to oil palm and pulpwood plantations, bringing about a huge detrimental impact on the global ecosystem. One of the immediate consequences of such activities is that the peatlands have become drier and more vulnerable to fire.

1.2 Current Research and Project Motivation

Soil moisture content in peat could be used as an indicator of fire risk. In Kuala Balai, Brunei, we have deployed two dataloggers at separate sites, each with four and eight

Campbell Scientific CS616 volumetric water content sensors respectively, to measure soil moisture content.

The current setup is centered on either the Campbell Scientific CR6 or CR300 dataloggers, with a reading made every 10 minutes. Depending on the model of the datalogger, the data is stored either in its internal memory of about 8MB (CR300) or an external expandable microSD memory card (CR6). The datalogger and the sensors are powered from a 12V 7Ah lead acid battery. Running at this configuration, the expected battery life is about 2 months.

We note that the peat properties are similar on the 50 to 100-m scale to the extent they were formed by the same forest type and have experienced the same land-use history. Within each environment, the variability maxes out at the 3 to 10-m scale. – this can be reached with one datalogger with eight CS616 sensors. However, because the parameters may be so variable around a single site, we cannot resolve for the difference between sites with fewer number of sensors. Therefore, it is important that we deploy as many sensors as the datalogger can handle at a particular site so that we can capture and increase the value of the data we obtained.

On the other hand, we know that the CR6 datalogger can log for up to twelve CS616 sensors while the CR300 datalogger cannot log for more than six CS616 sensors. Clearly, the deployment strategy at any site now depends on the number of CS616 sensors deployed and their reach in terms of their cable lengths, and the number and type of the dataloggers used.

While there are no better alternatives (in terms of cost and functionality) to the CS616 sensor, we find that the specifications of both the CR6 and CR300 are not well suited for our requirements. For example, if we need to deploy ten sensors using the CR300 datalogger configuration, we will need two dataloggers to log four and six sensors separately. This is obviously not a cost-effective solution, considering the cost of the dataloggers and the relatively simple task of cascading sensors together.

1.3 Objectives

The aim of this project is to develop a functional and self-contained Hardware-Attached-on-Top (HAT) electronic add-on board for the Raspberry Pi platform to measure volumetric water content using multiple Campbell Scientific CS616 sensors. Interfacing between this HAT and similar single-board computers or microcontrollers should be easy and user-friendly to aid in the deployment and/or integration of the sensors into new or existing sensor systems. The final product is expected to have the following features:

- Multiple sensor input channels (16 Channels)
- Signal noise reduction filter for accurate and precise reading of measurement
- Stack compatible with other HATs
- Electronic board with small form factor
- I2C address selector for HAT cascade

We will also build a prototype of a sensing system that is capable of measuring, collecting and storing soil moisture data. This system will have the following features:

- System encapsulated in a weatherproof case
- Long-term "deploy-and-forget" capability with low power consumption
- Solid-state relays to control multiple CS616 Sensors
- Large data storage capability
- Solar-powered with backup 12V 7Ah battery pack

Chapter 2

System Architecture

2.1 Introduction

This chapter describes the main aspects of the hardware and software architecture of the project. We will first discuss about the features, operations and applications of the hardware (sensors and electronic chips), and the role they play in our design. Subsequently, we will describe in detail the software tools and methods used, specifically the operating system of the Raspberry Pi, and signal communication protocols for data acquisition.

2.2 Hardware Architecture

2.2.1 Campbell Scientific CS616 Water Content Reflectometer

Water content of soil or similar porous media is measured using a CS616 reflectometer developed by Campbell Scientific. This high-fidelity and high-resolution sensor consists of two stainless steel rods attached to a multi-vibrator circuit encapsulated within a waterproof case (*CS616 and CS625 Water Content Reflectometers* 2016). Since the dielectric permittivity of the water content in the porous media is directly proportional to the time it takes for the signal to travel along the rods to the multi-vibrator circuit, the oscillation frequency of this multi-vibrator circuit and therefore, information regarding the water content can be easily derived. This is can be explained by the fact that water is the only

soil component that has a significantly large dielectric permittivity, and is the only other media that is susceptible to change (the other being air).

Operation

An electromagnetic pulse is emitted from the multi-vibrator, travelling along the rods at a velocity dependent on the dielectric permittivity of the surrounding media. A low velocity implies that the water content is high since more time is required for water molecules to be polarized. The output signal of the sensor is theoretically in the form of a square wave, with its period being the time it takes for each electromagnetic pulse to travel twice the length of the rod. The period of the signal is about 14 microseconds under dry air conditions, and 42 microseconds when the rods are completely immersed in water.

Using the calibration equations described below, the volumetric water content (VWC) can be obtained:

$$VWC[quadratic] = -0.0663 - 0.0063 * T + 0.0007 * T^2$$

$$VWC[linear] = -0.4677 + 0.0283 * T$$

where T is the period of the square wave signal emitted from the sensor.

For this project, we assume that the CS616 reflectometers we used are sufficiently calibrated, and are capable of producing accurate and precise volumetric water content.

Installation

The probe rods are inserted vertically into the soil. In this orientation, the volumetric water content up to 30 cm from the surface of the soil can be determined. Alternatively, the rods can be inserted horizontally to measure vertical water fluxes, or at an angle of 30 degrees from the surface of the soil – this will measure the volumetric water content of the upper 15 cm from the surface.

Electrical Specifications and Wiring Configuration

TABLE 2.1: CS616 Volumetric Water Content Reflectometer

Component	Description
Power Supply	5-18 VDC
Power Consumption	65mA @ 12VDC when enabled, 45 uA when idle
Enable Voltage	Logic High: 4-18VDC
Output Signal	0.7V Square Wave with frequency dependent on volumetric water content

TABLE 2.2: Wiring Configuration of CS616 Volumetric Water Content Reflectometer

Color	Description
Red	+12V Power
Black	Signal Ground
Clear	Power Ground
Green	Signal Output
Orange	Enable

Note: To activate a CS616 sensor, apply a logic high signal to the enable wire.

2.2.2 Raspberry Pi

The Raspberry Pi is a credit card sized single board computer that has enough computational power to run a multitude of tasks. A series of general-purpose input/output (GPIO) pins on the board allows connectivity and control of external interfaces, making the Raspberry Pi the ideal solution for this project.

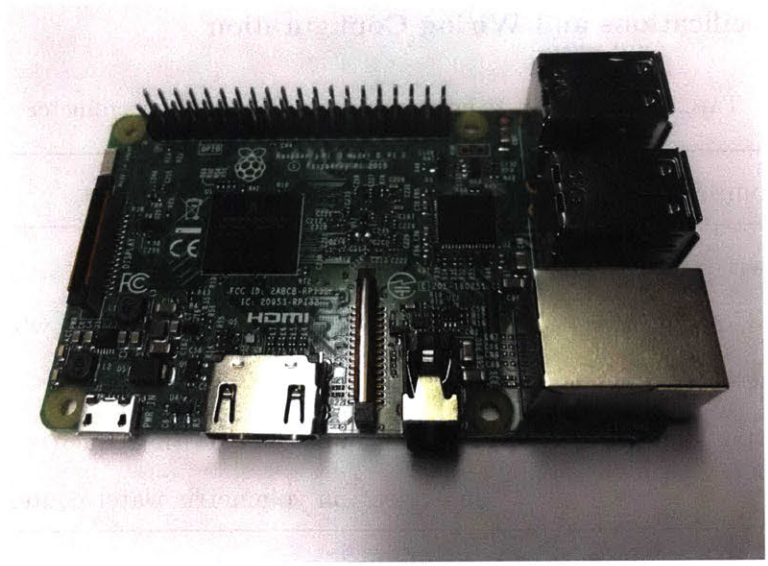


FIGURE 2.1: Raspberry Pi 3 B+

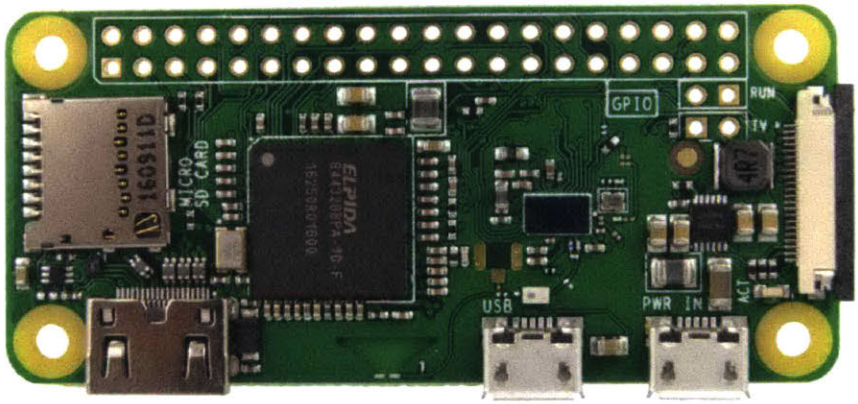


FIGURE 2.2: Raspberry Pi Zero

The following tables detail the specifications of the B+ 3 and Zero variants that will be used extensively throughout this project.

TABLE 2.3: Raspberry Pi 3 Specifications

Component	Description
System-on-Chip	Broadcom BCM2857
CPU	4x ARM Cortex-A53, 1.2GHz
GPU	1GB LPDDR2 (900 MHz)
RAM	1 GB with GPI
Networking	10/100 Ethernet, 2.4GHZ 802.11n wireless
Bluetooth	Bluetooth 4.1 Classic, Bluetooth Low Energy
Storage	MicroSD
GPIO	Populated 40-pin header
Peripherals (Low-Level)	Power (+5V, +3.3V, GND), I2C, SPI
	HDMI, 3.5mm analog jack, 4 x USB2.0
Ports	Ethernet, Camera Serial Interface,
	Display Serial Interface
Power	5V MicroUSB, 1.5W (Idle), 6.7W (Maximum Stress)

TABLE 2.4: Raspberry Pi Zero Specifications

Component	Description
System-on-Chip	Broadcom BCM2835
CPU	1 GHz single-core ARM1176JZF-S
GPU	1GB LPDDR2 (900 MHz)
RAM	512MB shared with GPU
Networking	None
Bluetooth	Bluetooth 4.1 Classic, Bluetooth Low Energy
Storage	MicroSD
GPIO	Populated 40-pin header
Peripherals (Low-Level)	Power (+5V, +3.3V, GND), I2C, SPI
Ports	mini-HDMI, micro USB2.0
Power	5V MicroUSB, 0.4W (Idle), 0.7W (Maximum Stress)

2.2.3 CD74HC4067 Multiplexer

Operation

The CD74HC4067 is a high-speed CMOS 16-Channel Analog Multiplexer that switches one of several input lines through to a single common output. The multiplexer will internally route the common pin to any of the 16 input/output channel pins based on the signal sent to the four digital switch inputs (S3, S2, S1, S0). The CD74HC4067 is also bidirectional - any of the 16 analog input pins can be used as an output and vice versa.

The following truth table describes how the multiplexer responds to various combinations of input/output signals. A '1' represents a logic HIGH (i.e. +3.3V). Conversely, '0' represents a logic LOW (GND).

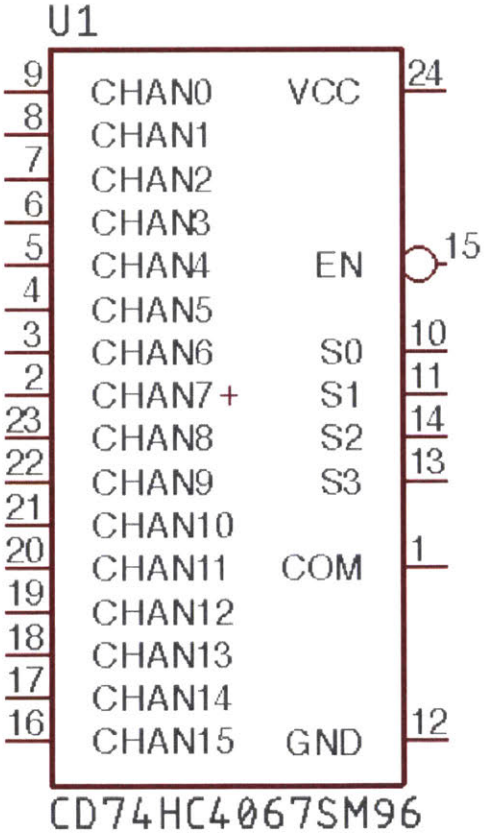
TABLE 2.5: CD74HC4067 Input/Output Truth Table

Input					Output
EN	Select				
	S3	S2	S1	S0	
0	0	0	0	0	0
1	0	0	0	1	1
1	0	0	1	0	2
1	0	0	1	1	3
1	0	1	0	0	4
1	0	1	0	1	5
1	0	1	1	0	6
1	1	1	1	1	7
1	1	0	0	0	8
1	1	0	0	1	9
1	1	0	1	0	10
1	1	0	1	1	11
1	1	1	0	0	12
1	1	1	0	1	13
1	1	1	1	0	14
1	1	1	1	1	15
0	X	X	X	X	X

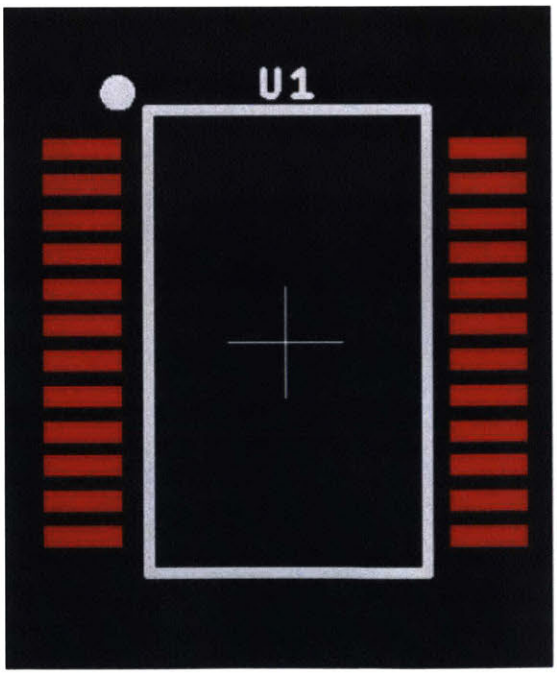
Pin Configuration, Functional Diagram and Package

TABLE 2.6: Pin Configuration of CD74HC4067

Symbol	Pin	Function
CHAN0-CHAN7	9-2 (Decreasing order)	Output/Input Pins
CHAN8-CHAN15	23-16 (Decreasing Order)	Output/Input Pins
GND	12	Signal Ground
VCC	24	Supply Voltage (+3.3V)
S3-S0	13-10 (Decreasing Order)	Digital Switch Inputs



(A) Functional Diagram of CD74HC4067



(B) PCB Footprint of CD74HC4067

FIGURE 2.3: EAGLE Representation of CD74HC4067

2.2.4 SN74HC14N Hex Schmitt Trigger Inverter

The SN74HC14N is an electronic chip containing six inverting Schmitt Triggers. In later sections of this thesis, we will discuss about the output signal from the CS616 sensor and assert that it requires conditioning before it can be digitized and processed by the Raspberry Pi. This subsection mainly discusses about the operation of a Schmitt trigger circuit, and how it can be useful for removing inherent noise from a signal.

Operation

A Schmitt trigger is essentially a comparator circuit which incorporates hysteresis into the transition threshold voltage. This reduces delay when the output signal transitions from logic low to logic high. It performs the following Boolean function (in positive logic):

$$Y = \bar{A}$$

where Y and A are the input and output signals respectively.

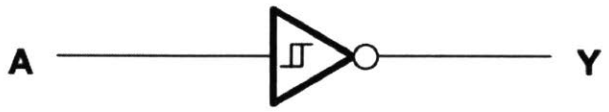


FIGURE 2.4: Functional Diagram of a Schmitt Trigger

Figure 2.4 shows a basic CMOS-based inverting Schmitt trigger circuit:

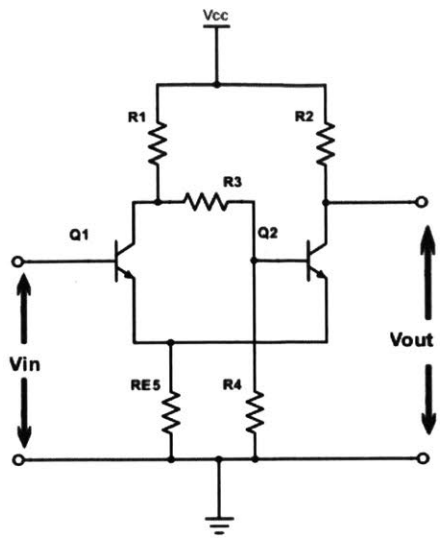


FIGURE 2.5: A Simplified Diagram of a Basic CMOS Schmitt Trigger

There are many variants of the Schmitt trigger, however, we chose the complementary metal–oxide–semiconductor (CMOS) implementation because of the following advantages:

- Input and output are balanced with threshold typically symmetrical to the supply voltage
- Wide range of compatible supply voltage
- Consumes lower power when compared with the operational amplifier configuration
- High immunity to noise
- Schmitt trigger integrated circuits are usually "all-in" i.e. do not require additional components to function

In the inverting configuration shown in Figure 2.4, the output signal remains the same until the input signal deviates significantly from the threshold, thus triggering a change. When the input is lower than the assigned threshold, the output will be high. Conversely, if the input is lower than the threshold, the output retains the value. This is known as hysteresis, which allows the Schmitt trigger to condition noisy signals, and is particularly

effective when the output signal is originally meant to be a square wave. We will describe the performance of the Schmitt trigger given a noisy signal response from a CS616 in later sections of this thesis.

Pin Configuration, Functional Diagram and Package

TABLE 2.7: SN74HC14N Pin Configuration

Symbol	Pin	Function
1A	1	Channel 1 Input
1Y	2	Channel 1 Output
2A	3	Channel 2 Input
2Y	4	Channel 2 Output
3A	5	Channel 3 Input
3Y	6	Channel 3 Output
GND	7	Ground
VCC	7	Supply Voltage (+3.3V)

Note: Only pin configuration of channels 1, 2 and 3 are shown in this table.

2.2.5 SN74LV8154 Dual 16-Bit Binary Counters

In Section 2.2.1, we find that the volumetric water content is represented as a square wave signal with a period ranging from 14 to 42 microseconds (or a frequency between 23 khz and 71 kHz). Therefore, to quantify the frequency of this square wave signal accurately, we will need to design a circuit with a sample rate high enough to precisely count the rising edges of the square wave. In this section, we will discuss the two methods to count high frequency signals and selecting the optimum sample rate. We will also elaborate the reasons we chose the SN74LV8154 as the preferred binary counter.

Frequency Counting Methods

Direct Counting

The most intuitive method to determine the frequency of a signal is to count the number of rising edges of that signal over a fixed period of time (sampling period) (*Fundamentals of the Electronic Counters* 1997). We first define a *rising edge* as the transition of a signal from logic low to logic high. Conversely, a *falling edge* is defined as the transition of a signal from logic high to logic low. We will focus only on counting these rising edges to determine the frequency of a signal in our design.

Figure 2.6 below shows a "positive-going" pulse signal, with a positive-going leading rising edge.

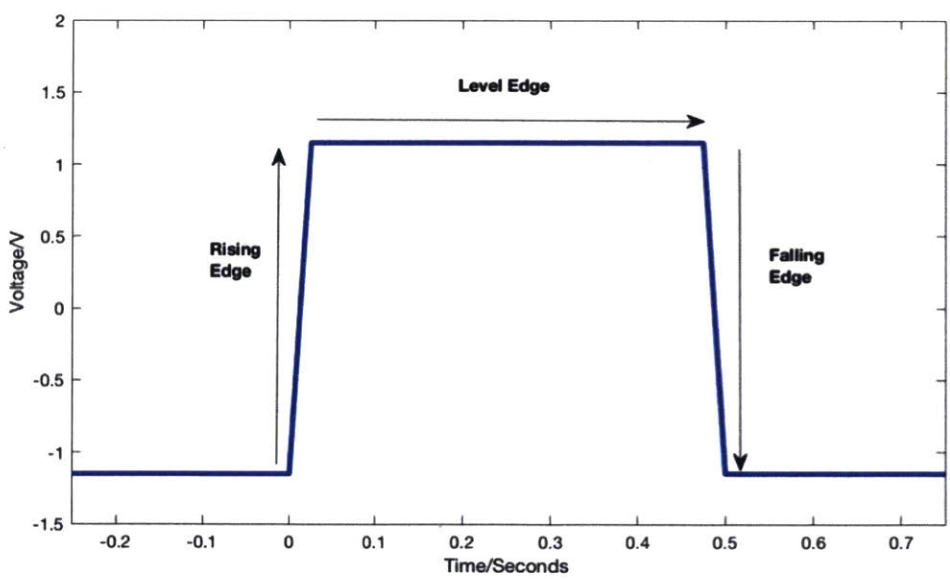


FIGURE 2.6: A pulse from a square wave signal

Since the frequency of the signal is congruent to the total number of rising edges over a precise time window, we can feed the signal of interest to a direct counter chip, such as the Renesas ICM721B or Texas Instrument TPL5100, and set the sampling period to

a pre-determined interval (1 second, for example). A system block diagram of a direct counter is shown in Figure 2.7 (Kirianaki, Yurish, and Shpak, 2001).

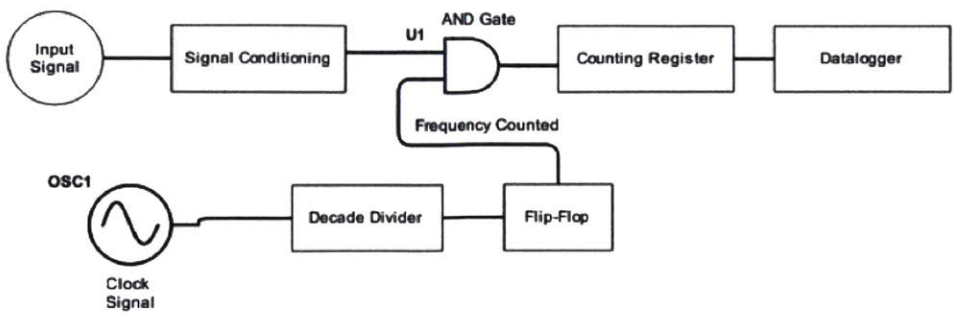


FIGURE 2.7: Block Diagram of a Direct Counter

The following equation describes how the frequency of a signal can be determined (*Frequency Measurement Guidelines for Oscillators* 2013):

$$f_{measured} = N/\tau$$

where τ is the open gate time of the chip, N is the number of rising edges counted, and $f_{measured}$ is the frequency of the signal as measured by the chip.

From this equation, we see that the resolution of $f_{measured}$ is dependent on how large τ is. If τ is significantly smaller than the actual frequency of the signal, f_{signal}, there is a chance that the chip will not capture the rising edges of the signal. Also, the accuracy of the frequency measurement is dependent on how the open gate signal represents a 1 Hz pulse. To understand how this would affect the resolution of the counter, we can look at the following example.

First, we set the open gate time τ to 1 second. Therefore, the lowest detectable frequency of the counter will be 1 Hz. Subsequently, if we feed a 10Hz signal to the counter, we expect the resolution to be 1 Hz with a 4-digit display since 1000 pulses will be let through.

Similarly, a 100 kHz signal being fed to a counter with a 1 second open gate time will have a 6-digit resolution.

We can then show that the limiting error of the direct counting method is:

$$\delta_g f_{measured} = \delta_g f_{oscillator} + +\frac{1}{N} = \delta_g f_{oscillator} + \frac{1}{f_{measured} * \tau}$$

where $\delta_g f_{oscillator}$ is the limiting error of the frequency of the quartz oscillator in the counter chip. As such, we see that the direct counting method is highly susceptible to quantization error(Kirianaki, Yurish, and Shpak, 2001).

Reciprocal Frequency Counting

The reciprocal frequency counting method, unlike the direct counting method, determines the frequency of an input signal by making a period measurement, rather than the number of counts given a fixed sampling time (Kirianaki, Yurish, and Shpak, 2001).

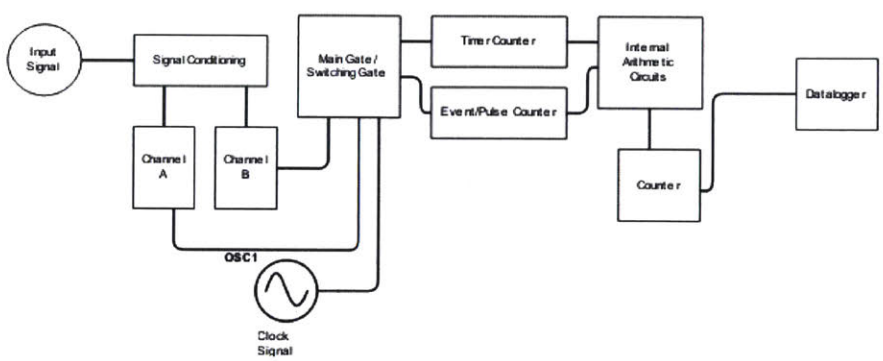


FIGURE 2.8: Block Diagram of a Reciprocal Counter

Figure 2.8 shows the block diagram of a reciprocal counter (Kirianaki, Yurish, and Shpak, 2001). While it is similar to the block diagram of the direct counter, it has two distinct internal counters that count time and event separately. The time counter accumulates the

number of rising edges from the clock signal as long as the main gate stays open, while the event counter accumulates the rising edges from the input signal. The main gate is controlled by the input signal, and the duration that the gate will stay open is exactly one period of the input signal (i.e. the time between the first and second rising edges of the input signal). Within this time period, the time counter will have accumulated a count of the rising edges of the clock signal, while the event will have accumulated one count from the input signal. Hence, the actual period of the input signal can be calculated by multiplying the total number of clock pulse counts from the time counter with the known clock period. The following example illustrates how the frequency of an input signal can be calculated.

We first define the following parameters:

$$f_{clock} = 1MHz$$

$$f_{input} = 52.5kHz$$

where f_{clock} is the frequency of the clock signal, and f_{input} is the actual frequency of the input signal.

The open gate time, τ, will be the inverse of f_{input}:

$$\tau = \frac{1}{f_{input}} = \frac{1}{50kHz} = 19.05\mu s$$

We find the number of 1 MHz periods, N, in τ to be:

$$N = \tau * f_{clock} = 19.05\mu s * 1MHz = 19.05 \approx 19$$

Since N is discrete, we drop the value of its decimal.

To find the measured frequency of the input signal, $f_{measured}$, we solve for the equation:

$$f_{measured} = \frac{f_{clock}}{N} = \frac{1MHz}{19} = 52.63kHz$$

Thus we see that in the reciprocal method, the resolution of the measured frequency is now dependent on the input clock signal at ± 1 cycle. This is a tremendous improvement on the direct counting method, where the resolution is ± 1 Hz. In both cases, however, the accuracy of the measurement is dependent on the stability, and subsequently, the accuracy of the input clock signal(Kirianaki, Yurish, and Shpak, 2001). We will discuss about the choice of input clock source in Chapter 3.2.4.

SN74LV8154 Binary Counter

The SN74LV8154 is a dual 16-bit reciprocal counter designed for low-voltage microcontroller projects. It has two dedicated counters (A and B) each with its own clock input. To access the counters, four bus reads are required.

Each counter feeds a 16-bit storage register, which is then mapped to an 8-bit output bus Y. Y is then further divided into two bytes: upper and lower bytes. Output control of these bytes dependent on a set of pins, GAL, GAU (for counter A) and GBL, GBU (for counter B). Table 2.8 summarizes the output bus logic.

TABLE 2.8: Summary of Storage Register Control Pins

Symbol	Pin
$\overline{GAL}$	Gate A LB, active-low puts LB of stored counter A on the Y bus.
$\overline{GAU}$	Gate A UB, active-low puts UB of stored counter A on the Y bus.
$\overline{GBL}$	Gate B LB, active-low puts LB of stored counter B on the Y bus.
$\overline{GBL}$	Gate B UB, active-low puts UB of stored counter B on the Y bus.

LB: Lower Byte UB: Upper Byte

TABLE 2.9: Output Bus Logic of SN74LV8154

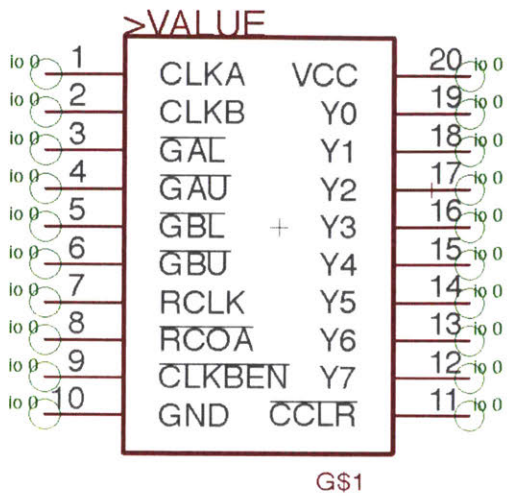
Input				Output Yn
GAL	GAU	0GAL	GAB	
0	1	1	1	Lower Byte in A Storage Register
1	0	1	1	Upper Byte in A Storage Register
1	1	0	1	Lower Byte in B Storage Register
1	1	1	0	Upper Byte in B Storage Register
1	1	1	1	X

Our design only requires one counter. We will be chaining CCKBEN to RCOA to build a single 32-bit counter and consequently, improving the design's resolution. In this configuration, the maximum number of counts for every single event is $2^{32} - 1$ (4294967295) per storage register read.

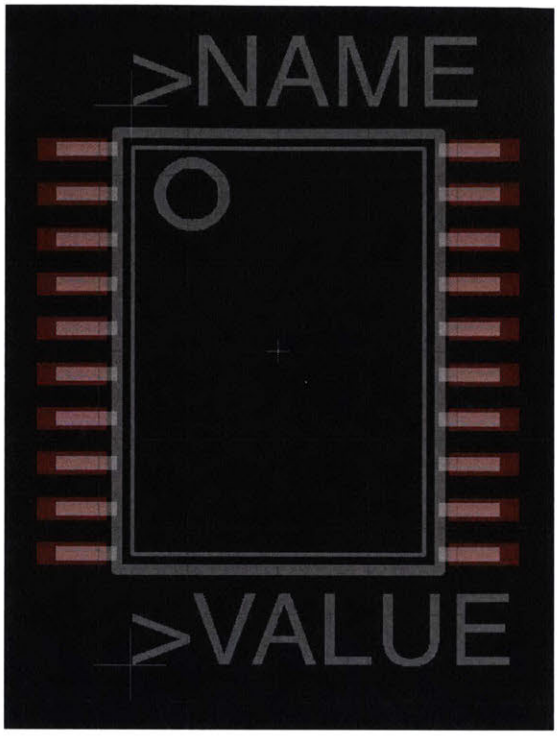
Pin Configuration, Functional Diagram and Package

TABLE 2.10: Pin Configuration of SN74LV8154

Input		Description
Name	No	
CCLR	11	Clear Clock
CLKA	1	Clock Input for Counter A
CLKB	2	Clock Input for Counter B
CLKBEN	9	Clock B Enable
GAL	3	Gate A Lower Byte
GAU	4	Gate A Upper Byte
GBL	5	Gate B Lower Byte
GBU	6	Gate B Upper Byte
GND	10	GND
RCLK	7	Register Clock
RCOA	8	Ready case overflow A
Vcc	20	Power Supply
Y0-Y7	19-12 (Descending Order)	Data Output Bin 0-7



(A) Functional Diagram of SN74LV8154



(B) PCB Footprint of SN74LV8154

FIGURE 2.9: EAGLE Representation of SN74LV8154

2.2.6 MCP23017 16-Bit I/O Expander with Serial Interface

In Section 2.2.5, we note that the SN74LV8154 requires eight and four pins to read and control the storage registers respectively. These pins will be driven via connections to the Raspberry Pi's GPIO pins. However, because the Raspberry Pi only has 26 GPIO pins, dedicating 12 pins specifically for read and control of a single chip is not the most efficient way to do a 8-bit data transfer. Furthermore, there might also be the possibility that other tasks/HATs would require the use of the same set of GPIO pins. Therefore it is good practice to reduce the total pin count requirements in our design. This can be accomplished by using an In/Out or I/O Expander.

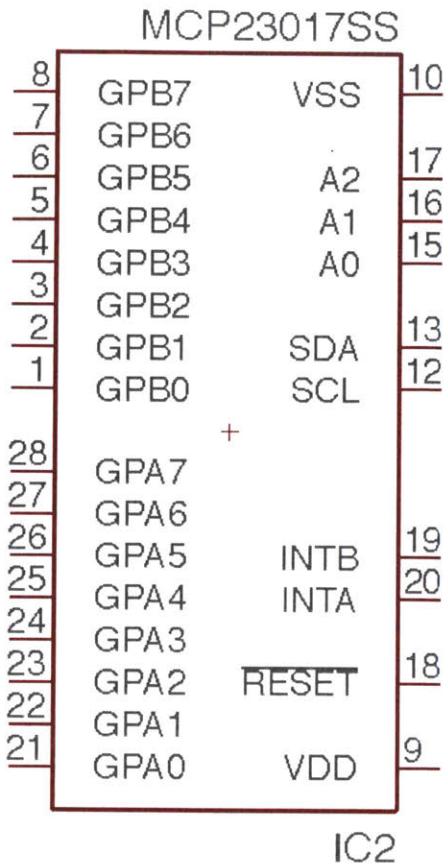
Operation

The MCP23017 is 16-Bit I/O Expander that allows control of 16 input/output channels through a serial communication via the Inter-Integrated Circuit (I2C) protocol. We will

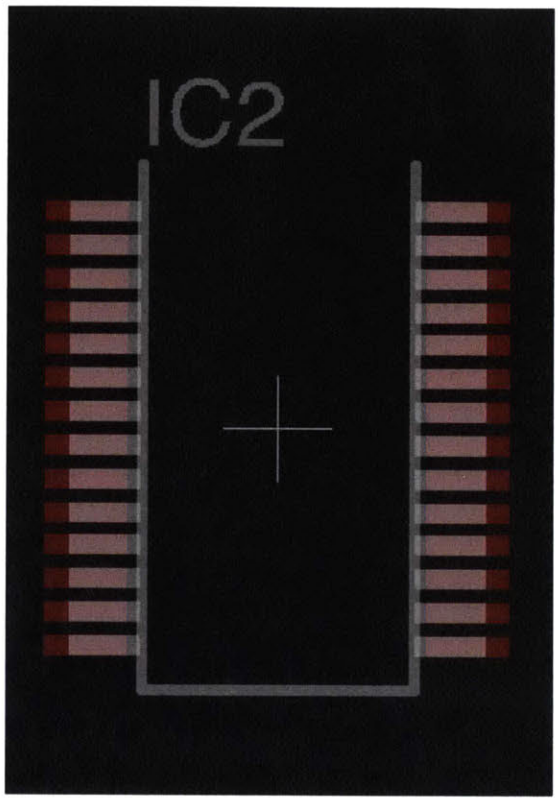
elaborate on the I2C protocol in Section 2.3.2 of this chapter. When connected to the SN74LV8154, the MCP23017 chip can reduce the pin count of the design to 2 ports from an initial 16, thus freeing the remaining GPIO pins for other tasks. We simply connect the Raspberry Pi's SCL and SDA ports (pins 3 and 5 respectively) to the corresponding SCL and SDA ports on the MCP23017, and set the desired I2C address by sending logic highs or lows to pins A2-A0.

TABLE 2.11: Pin Configuration of MCP23017

Input		Description
Name	No	
GPB0 - GPB7	1 - 8	Bi-directional I/O Pins
VDD	9	Power
Vss	10	Ground
SCL	12	Serial Clock Input
SDA	13	Serial Data I/O
A0 - A3	15-17	Hardware Address Pins
RESET	18	Hardware Reset
INTB, INTA	19, 20	Interrupt A, Interrupt B
GPA0 - GPA7	21-28	Bi-directional I/O Pins



(A) Functional Diagram of MCP23017



(B) PCB Footprint of MCP23017

FIGURE 2.10: EAGLE Representation of MCP23017

2.3 Software Platform

2.3.1 Raspbian Operating System for Raspberry Pi

The Raspberry Pi is capable of running a variety of Linux-derived operating systems (OS). For this project, we will be using the minimalistic Raspbian OS, developed specially and specifically for use with the Raspberry Pi. The current iteration of the Raspbian OS at the time of writing is based on a Debian GNU/Linux 8 system codenamed "Stretch" with an ARMv7 or armhf 32-bit hard-float architecture. We chose Debian Stretch Lite over

other Linux derivatives because we find that the technical and community support are unparalleled, with a vast quantity of repositories available online.

Accessing the OS initially requires IO peripherals like the keyboard, a display screen and a mouse. Ultimately, we would want to configure the system into a headless setup where access to the Raspberry Pi will be done remotely via SSH or a serial connection.

2.3.2 Inter-Integrated Circuit (I2C) Interface

The Inter-Integrated Circuit (I2C) is a communication protocol developed in 1982 by Philips Semiconductors. It allows multiple master chips to communicate with one or more slave chips, and like similar Asynchronous Serial Interfaces, each communication bus only consists of two wires, SCL and SDA. SDA is the wire along which data signals are conducted between chips, while SCL is the clock signal generated from the current master chip.

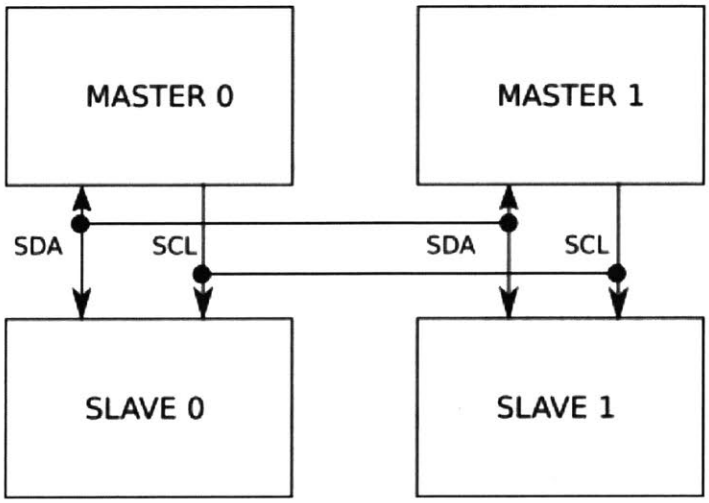


FIGURE 2.11: Block Diagram of a System in I2C Configuration

Operation

Data between master and slave chips are exchanged via packets known as messages, which consist of two frame types; address and data frames. The address frame identifies the slave that will be receiving information from the master. In any new sequence of communication, the 7-bit address frame will be clocked out from the Most Significant Bit (MSB) first, and the next bit (8th bit) after the last address determines if the message is a read or write operation.

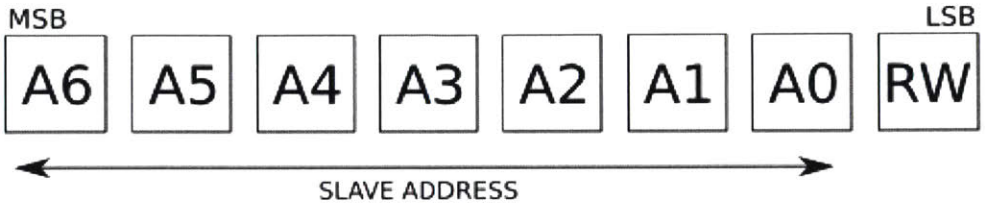


FIGURE 2.12: Block Diagram of an I2C Message

The data frame will then be transmitted through SDA immediately after the address frame has been sent to and acknowledged by the master chip at a rate dictated by the clock pulses on SCL. The length of the data frame is arbitrary i.e. it depends on the how much data is transmitted from the master to the slave, and whether it is a read or write operation.

The end of the data frame is bookended by a stop-bit, signalling to the slave chip that the transmission has reached its end. This is done by sending "a logic low to logic high" signal to the SDA immediately after a logic low to logic high on the SCL. The message is then interpreted by the Raspberry Pi, and this process will be elaborated in the next chapter.

Chapter 3

Design

3.1 Introduction

In this chapter, we will describe how the hardware and software architectures are integrated together to form a coherent system. We will also elaborate about the strategy we took to process noisy analog pseudo-square wave signals from the CS616 sensors into digital signals. Finally, we will discuss about the design and fabrication processes of the HAT printed circuit boards, and how we can build a sensor system based around these boards.

3.2 System Design

3.2.1 Signal Processing Strategy

The following steps describe the signal processing strategy to process analog pseudo-square wave signals from the CS616 sensors into digital signals, and to store this data, in the form of comma-separated values file, into the Raspberry Pi.

Step 1: Select CS616 sensor

A set of signals from the Raspberry Pi (S3, S2, S1, S0) is sent to the CD74HC4067 Multiplexer to enable the CS616 sensor of interest.

Step 2: Condition signal from CS616

Once the target CS616 sensor is enabled, its output signal is conditioned so that its signal characteristics are compatible with the SN74LV8154 Binary Counter. This is done by passing the output signal from the sensor into the SN74HC14N Hex Schmitt Trigger Inverter.

Step 3: Sample and digitize count signal from SN74LV8154 Binary Counter

We note earlier that the output from the SN74LV8154 Binary Counter is in the form of a 32-bit binary signal, of which it is represented by 32 pins. Therefore, to reduce the pin count, this signal is fed to a MCP23017 and subsequently, digitized and reduced to a 2-pin Input/Output I2C wiring format.

Step 3: Store frequency data into CSV file

Raspberry Pi reads the signals from the MCP23017, converts them into frequency domain, and store the data in the form of a comma-separated value (CSV) file.

Figure 3.1 shows the flow of data signal from a set of CS616 sensors to the Raspberry Pi

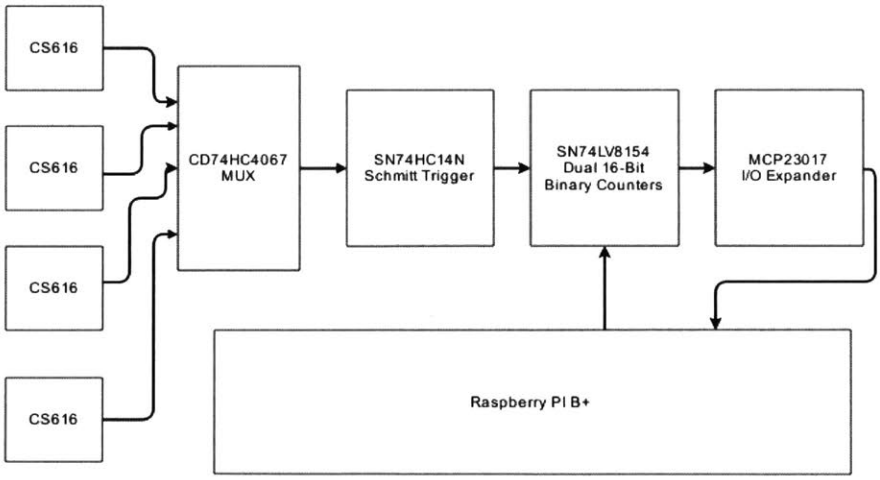


FIGURE 3.1: Signal Flow of CS616 Volumetric Water Content Data

3.2.2 Multiplexer Program Algorithm and Implementation

The Raspberry Pi selects the target CS616 sensor to be read using its four GPIO pins connected to the CD74HC4067 multiplexer. From Chapter 2, we find that access to a particular pin on a multiplexer depends on the input selection signals, S3-S0. In our configuration, we define the bus of GPIO pins to be pins 31, 33, 35 and 37, and we connect these to S3, S2, S1 and S0 respectively. A 4-bit signal from the Raspberry Pi, which corresponds to the multiplexer's input channel, will decide which input signal goes to the multiplexer output pin, COM (or SIG). For example, to feed the input signal from Channel 4 to the output pin, a 4-bit signal of Boolean values 0100 must be applied to the selection bus of the CD74HC4067 multiplexer.

To synthesize this 4-bit signal, we have written a simple Python program that controls the output of the GPIO pins based on the channel chosen by the user. The two functions in this program are:

TABLE 3.1: Integer to Binary Number Converter

int2bin *(n, count = 4)* Converts integer number to a binary representation	
Parameters	n: integer count: integer, optional
Returns	4 bit representation of n

TABLE 3.2: Controlling the Multiplexer

mux_channel *(channel, chan_list)* Select a channel on a multiplexer given an array of selection signal pin numbers	
Parameters	channel: integer Channel of multiplexer chan_list: vector of integers [1x4] Selection bus S3,S2,S1, S0
Returns	4 bit representation of n

The first function *int2bin* involves synthesizing a 4-bit binary number of base 2 from a given decimal number of base 10. We note that since there are only 16 input channels on the multiplexer, we assign each channel a number from 0 to 15. We then select the channel that we want its input signal to be fed to the output pin of the multiplexer, and assign its corresponding channel number to a variable n. If n is not a number between 0 and 15, we raise an error because the corresponding channel does not exist. Else, we continue with the next process in the algorithm.

We want the output binary number to be expressed in 4 significant bits starting from the most significant bit (MSB). Thus, we set the initial counter variable of the *for* loop sequence, *counter*, to be 4. We then shift the bits of the binary form of n to the right by y bits, and decrement *counter* by 1. This algorithm continues recursively until a 4-bit binary representation of the channel number is obtained. We return this binary number as an output to our function.

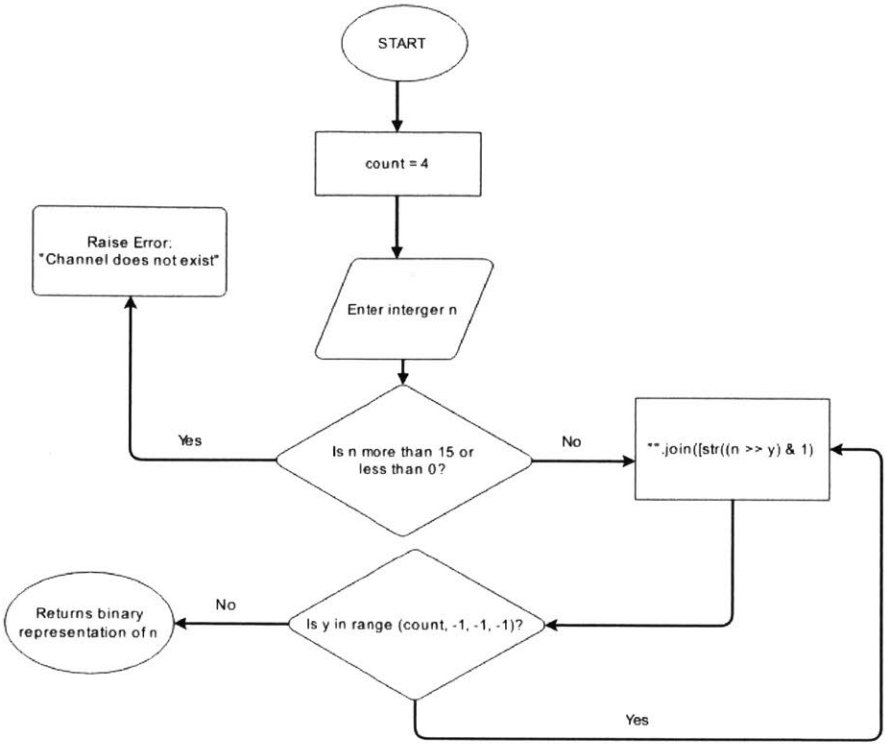


FIGURE 3.2: Flowchart of *int2bin* function

The next function, *mux_ channel*, divides the binary number obtained from *int2bin* into four individual bits. Each of these bits is assigned to S3, S2, S1 or S0, according to its position in the binary number. For example, if we input an integer value of 4 into the *mux_ channel* function, pins corresponding to S3, S2, S1 and S0 are assigned values of 0, 1, 0 and 0. A value of 1 implies that the Raspberry Pi will write and drive a logic high to the matching pin. Likewise, the signal pin will be set to a logic low if the value is 0.

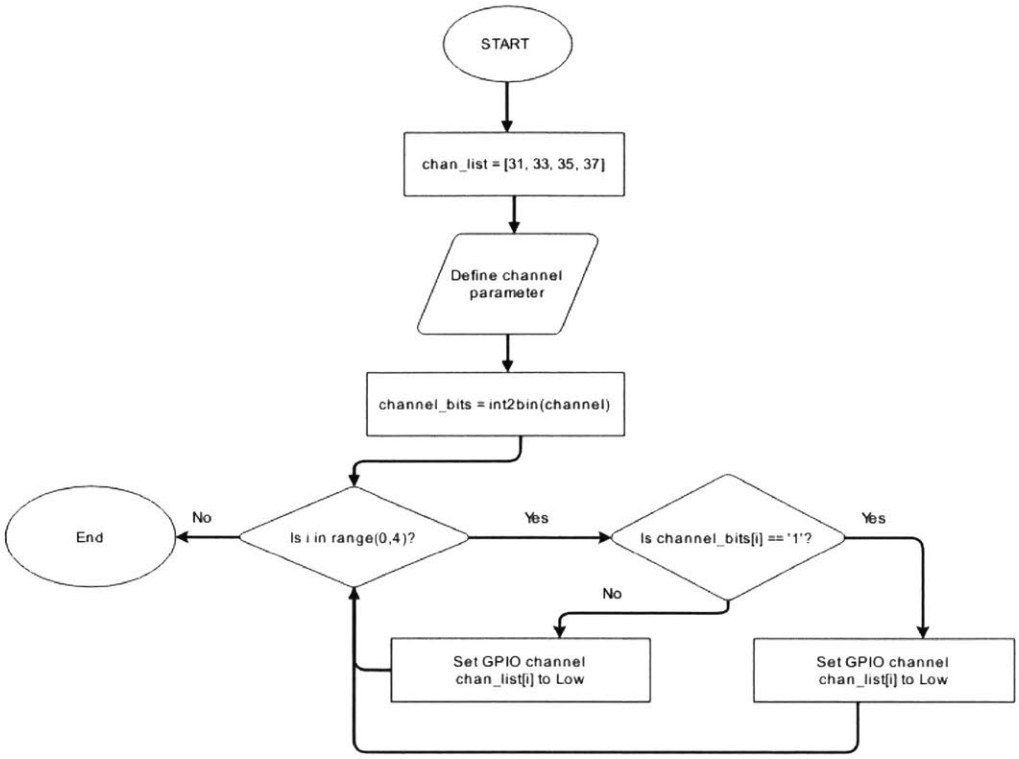


FIGURE 3.3: Flowchart of $mux_channel$ Function

The flowchart above graphically represents the steps of this Python function.

3.2.3 Signal Conditioning

In many applications, signals from sensors have to be conditioned before an acquisition device (in our case, the Raspberry Pi) can effectively measure them accurately. The techniques of signal conditioning greatly depend on the requirements of the acquisition device, but in general, the approach usually involves noise reduction, and converting the signal measurement to a form compatible with the digital acquisition hardware. In the context of this project, we define noise as any unwanted electrical signal that interferes or degrades with a communication signal.

The CS616 sensor, in theory, outputs a square wave signal, with its time period being proportional to the volumetric water content of the media in which the probe is immersed. However, we find that the presence of noise greatly distorts the shape of the square wave signal and as a result, it might be affect the accuracy of the measurement. The figure below shows the unfiltered signal from the CS616 obtained using a 100MHz 1GSa/S oscilloscope.

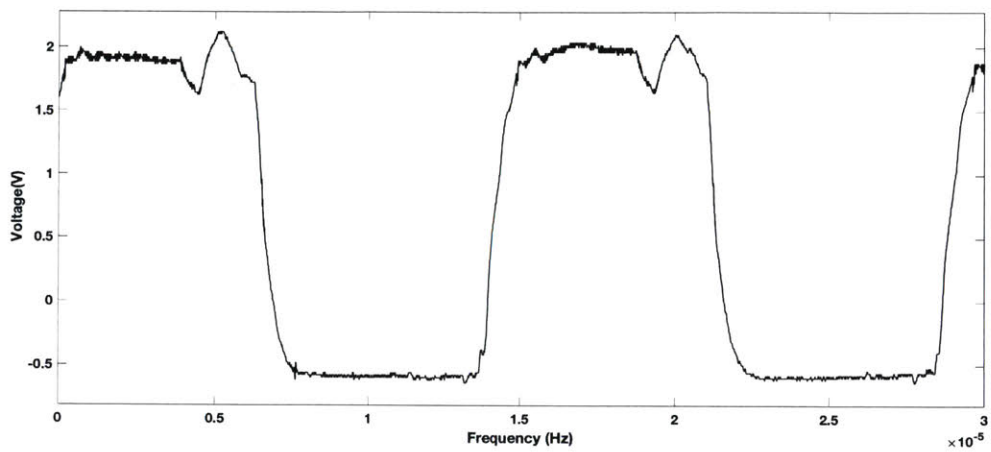


FIGURE 3.4: Time-Domain Representation of the CS616 Signal (Unfiltered)

From this graph in Figure 3.4, we see that the waveform of the unfiltered signal from the CS616 does not take the square waveform shape that we expect – and assuming that the sensor works as described in Section 2.2.1, we can conclude that there is a constant interference in the signal. There are two likely causes for this noise (*PCB Design Guidelines For Reduced EMI* 1999):

1. Capacitive coupling, or cross coupling, occurs when cable or wires carrying power or communication signals are carried in parallel over a long distance. When one signal couples capacitively with another, an electrostatic force is produced, causing what appears to be noise.

2. Inductive coupling between magnetic and electrical fields generates electromagnetic noise. This noise can be characterized by its occurrence at low frequencies. Unlike

capacitive coupling, inductive coupling usually occurs in a magnetic field dominant environment.

The next observation that we can make from the waveform is that the unfiltered signal ranges from about -0.5V to 2V. In Section 2.2.5, we define the rising edge of a pulse when a signal jumps instantaneously from logic low (0V) to logic high (3.3V). Since neither -0.5V and 2V can be considered to be logic low or logic high respectively, there will be a communications mismatch; our hardware will not be able to distinguish between the two logic levels as the architecture we have designed is built around the Raspberry Pi, which runs at a 3.3V CMOS logic level.

We need to condition the output signal of the CS616 so that:

1. The peaks and throughs are at least above 3V
2. Noise is removed from the signal, resulting in a square waveform recognizable visually and through electronic hardware and software

In Section 2.2.4, we discussed about the use of a SN74HC14N Hex Schmitt Trigger Inverter for noise reduction. Based on the operation of this device, we expect the output signal from the CS616 to satisfy the above requirements. The wiring schematic to connect the CS616 to the SN74HC14N is seen in Figure 3.5, and Figure 3.6 shows both the filtered and unfiltered signal responses from a CS616 sensor with its probe rods in free air.

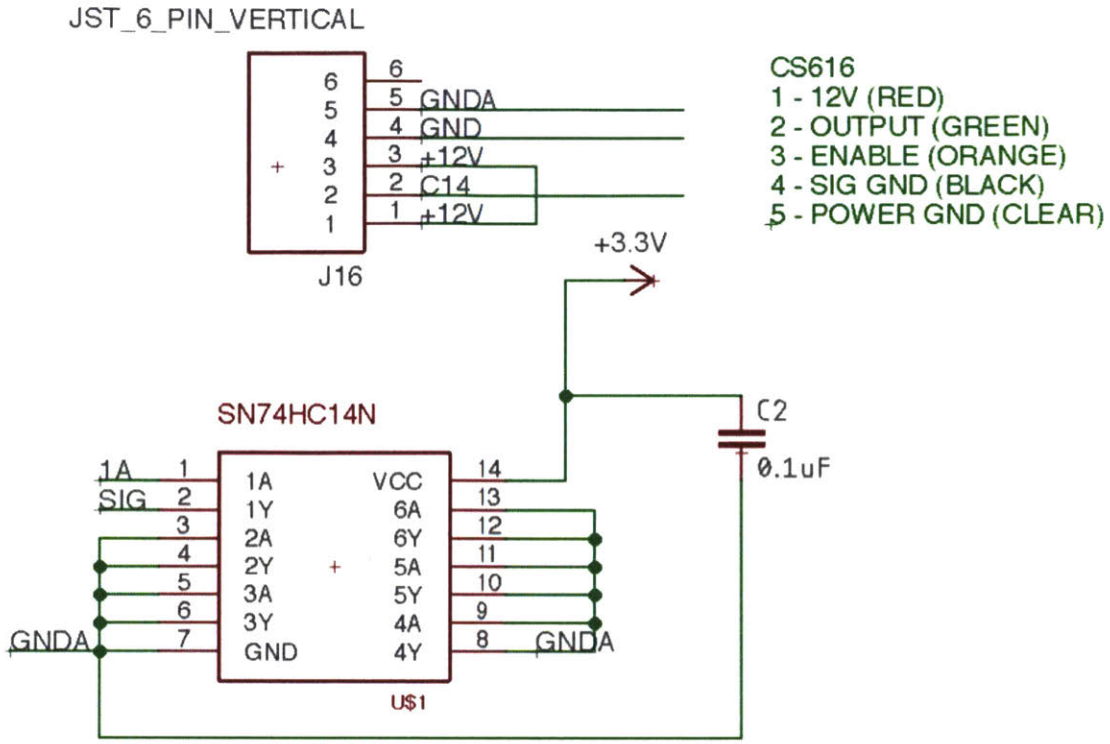


FIGURE 3.5: Schematic of the connections between the CS616 Volumetric Water Sensor and SN74HC14N Schmitt Trigger (Unfiltered)

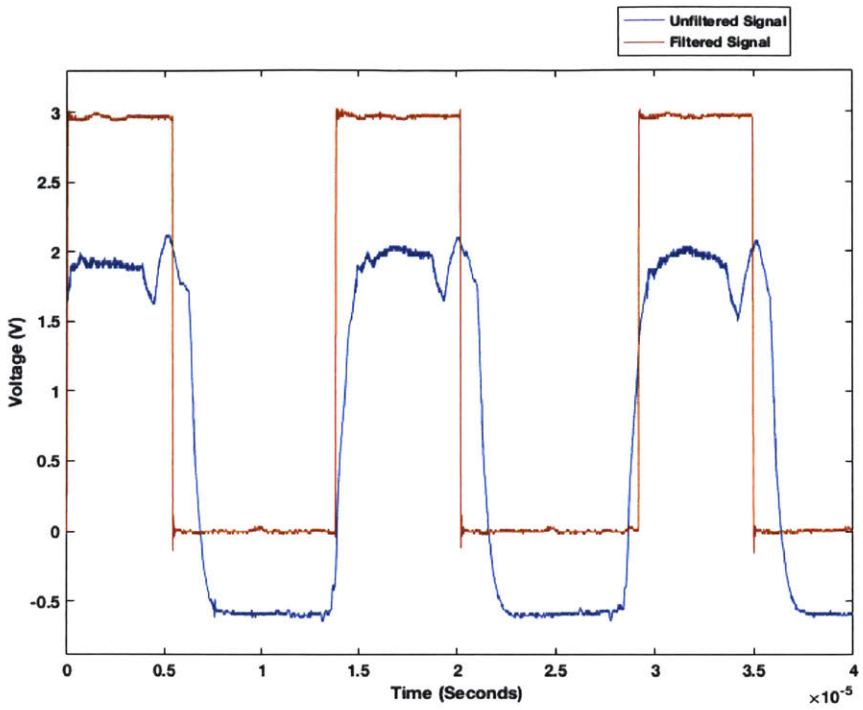


FIGURE 3.6: Time-Domain Representation of Unfiltered and Filtered Output Signals from CS616

The filtered signal now has a distinctly square wave pattern, with a fixed maximum and minimum values of about 0V and 3V, As we had expected, the maximum voltage of the square wave depends on the input VDD pin of the SN74HC14N, which in turn, depends on the voltage from the Raspberry Pi's 3.3V pin. In this case, the voltage on the Raspberry Pi's 3.3V pin is about 3V, and this value is mirrored in the waveform's maximum voltage.

The transition between its minimum and maximum points is also instantaneous, and consequently, the rising and falling edges can now be easily discerned. This is an important feature because the SN74LV8154 binary counter is triggered based on the leading edge of the input signal.

Signal conditioning sometimes results in loss of important features. As such, we would want the frequency of the filtered signal to match with the unfiltered signal even after

it has pass through the SN74HC14N Schmitt Trigger. We perform a Cooley-Tukey Fast Fourier Transform (FFT) to convert both the unfiltered and filtered signal from a time domain representation to a frequency spectrum, and the results are seen in Figure 3.7.

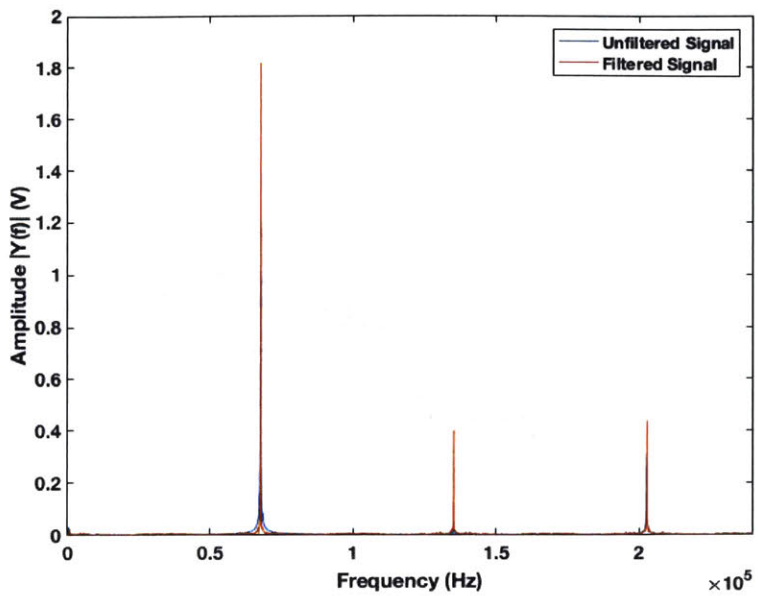


FIGURE 3.7: Frequency-Domain Representation of Unfiltered and Filtered Output Signals from CS616 Cooley-Tukey Fast Fourier Transform (FFT)

We see that the frequency spectra of both the unfiltered and filtered signal match, and this implies that there is no significant loss in the features of the frequencies after signal conditioning. For both signals, their first harmonics of the spectrum are about 67.62kHz. This is close to the frequency of pulsing with the probe rods in free air, which is about 70 kHz with a period of 14 microseconds. The difference in the values can be attributed to the humidity levels of the environment in which the trials were conducted. With this result, we can conclude that the signal response from the CS616 sensor has been sufficiently conditioned for the next stage of the process.

3.2.4 Power Line Conditioning

Delivering noise-free power supply to electronic circuits is important for optimum operation. These circuits are usually designed as direct-current (DC) circuits and as such, any fluctuation in the voltage supply may result in a device malfunction, failure and/or damage.

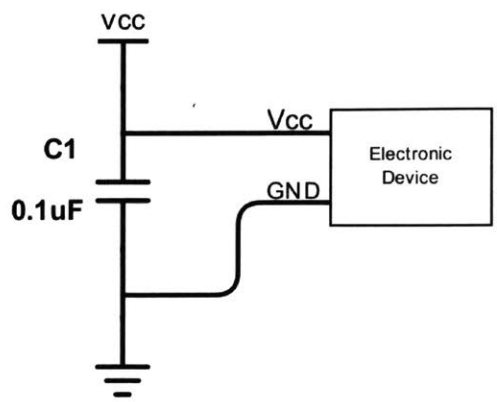


FIGURE 3.8: Electronic Device with Bypass Capacitor

We can mitigate the effects of noise on the power line by connecting a capacitor in a parallel to the voltage input and ground pins of an electronic device as illustrated in Figure 3.8. A capacitor connected in this configuration is known as a bypass capacitor. When the capacitor is charged, it basically becomes a current reserve that will compensate any drop in the voltage supply by releasing its charges. The capacitance determines how much charge (or energy) the capacitor can hold – the higher the value, the more charge the capacitor can release to compensate the drop in voltage. In our design, we chose a 0.1uF 0403 surface-mounted film capacitor connected in parallel as per Figure 3.8.

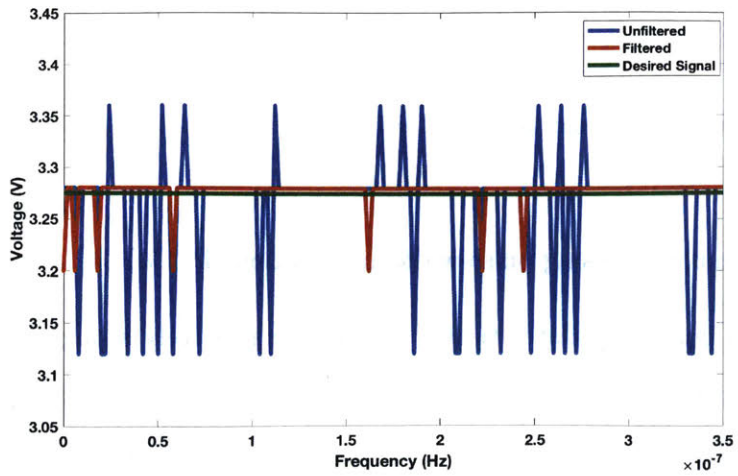


FIGURE 3.9: Effects of Adding a Bypass Capacitor on a Power Line

In Figure 3.9, we see that there is an immediate noise reduction in the power signal when a 0.1uF bypass capacitor is added to the circuit.

To measure the level of the desired voltage level to the background noise, we calculate the signal-to-noise (SNR) ratio using the following equation:

$$SNR_{dB} = 10log_{dB}(\frac{P_{signal}}{P_{noise}})$$

We find that the SNR of the power line signal in a circuit with and without the 0.1uF bypass capacitor to be 44.1766 dB and 34.4748 dB respectively. This implies that the signal strength is stronger in relation to the noise levels when a bypass capacitor is added as compared to a circuit without it.

3.2.5 Generating a Clock Signal with Pulse-width Modulation

In section 2.2.5, we note that the SN74LV8154 Dual 16-Bit Binary Counter requires a precise clock signal to count the frequency of the input signal reciprocally. In general, there are three methods to generate this signal – they are:

1. Building a timing circuit around a quartz oscillator (such as a 32.768 kHz clock oscillator).
2. Obtaining a pulse-per-second signal from a satellite using a Global Positioning System (GPS) receiver.
3. Software simulation using pulse-width modulation (PWM)

While building a quartz oscillator timing circuit is a relatively straightforward task, we find that the ability of the oscillator to generate a precise 1 pulse-per-second signal greatly depends on the ambient temperature. A temperature above or below room temperature will significantly alter the frequency of which the oscillator resonates. Since we are intending to deploy our system in environments where the temperatures are expected to be above room temperature ($>$ 25 degrees Celsius), this approach is generally not suitable for our requirements.

The second method requires an additional GPS receiver to obtain a pulse-per-second signal from a satellite. Though the PPS signal received is highly precise and stable, the amount of power the GPS receiver consumes negates this advantage. For example, the Ublox NEO-6M, a popular GPS receiver module, draws at least 40 mA of current in idle-mode after it has fixed its position. Our design is meant to be a dormant "deploy-and-forget" system and thus, it is not feasible if the set-up requires a frequent change of batteries.

A pulse-width modulation approach is essentially using a digital source to produce an analog signal. In this method, a digital signal is switched on and off at a rate fast enough that the output signal appears to behave like a constant voltage source. The constant voltage source is then dependent on two parameters:

1. Duty Cycle: The ratio of time the signal will stay at logic high to the time it takes to complete a full cycle .
2. Frequency: The time it takes for a PWM signal to complete a full cycle
3. Software simulation using pulse-width modulation (PWM)

The GPIO pins on the Raspberry Pi can be programmed to achieve this task by using the module RPi.GPIO. This module contains a class that allows the Raspberry Pi to control the GPIO pins to perform a series of tasks, including PWM modulation.

Since the SN74LV8154 Dual 16-Bit Binary Counter requires a clock signal input, we can write a Python script and define the PWM parameters in such a way that a complete square wave PPS output signal can be obtained. The parameters to generate this signal are:

1. Duty Cycle: 50 percent
2. Frequency: 1 Hz

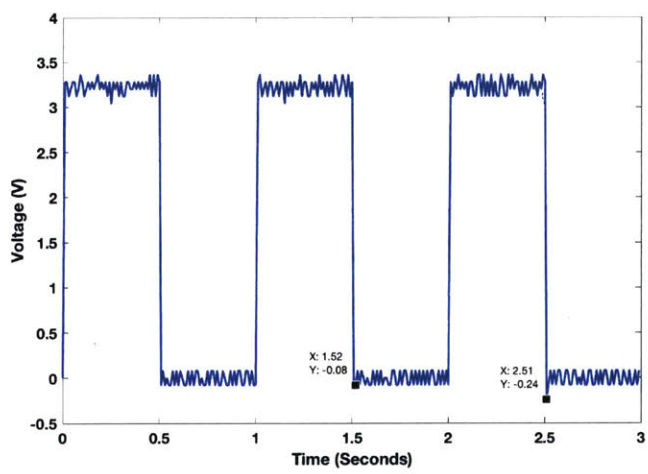


FIGURE 3.10: Clock Output from Pulse-Width Modulation

Figure 3.10 shows the output pulse signal, and we find that the frequency is very close to the desired 1 Hz rate.

We then apply this clock signal to the SN74LV8154 Dual 16-Bit Binary Counter as shown in the schematic in Figure 3.11.

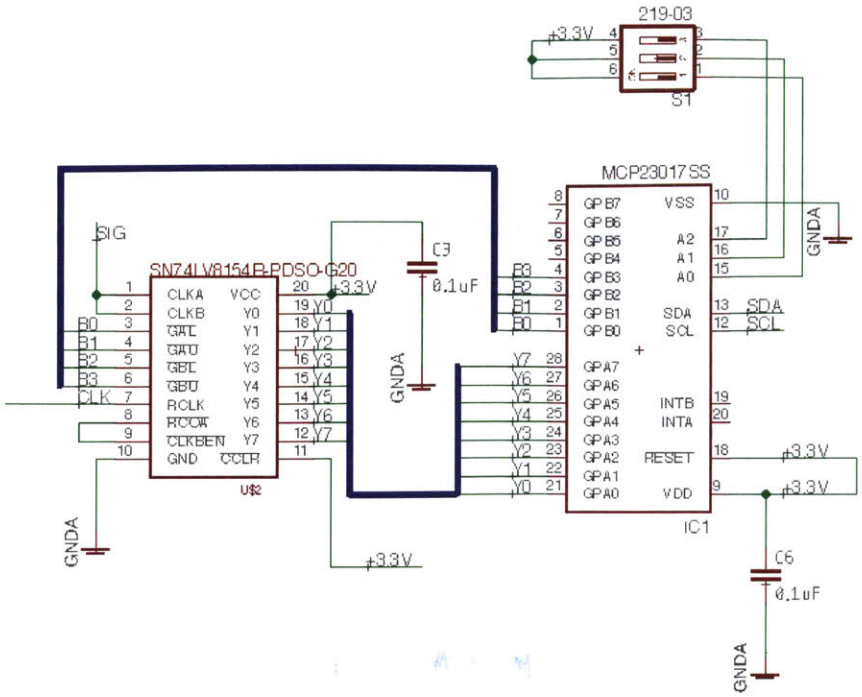


FIGURE 3.11: Clock signal input into SN74LV8154

3.2.6 Data Logging

The frequency of the CS616 signal is sensed by the SN74LV8154 Dual 16-Bit Binary Counter, and is subsequently acquired by the Raspberry Pi via the I2C protocol. This process is controlled using commands from the SMBus Python library, and the data obtained is then analyzed, processed and stored in a comma-separated values (CSV) file together with two other parameters: date and time in ISO 8601 format, and the unique identification number that corresponds to the sensor where the data originates from. Before deploying the Raspberry Pi and HAT with the CS616 sensors, the user must first define the sampling frequency, and the identification numbers of the sensors of interest. We set a sampling period of 10 min in our application.

3.2.7 Design and Fabrication of HAT Printed Circuit Board (PCB)

A printed circuit board or PCB is a board base that is used to mechanically and physically support electronic components, with each being connected via conductive pads or tracks etched on copper layers. A non-conductive material usually made of plastic resin separates each of these layers. This section describes the process of designing and fabricating a HAT PCB, particularly the design rules and guidelines that are specific to our design.

Computer-aided Representation of Design, and Selection of Electronic Components

The design of a PCB board usually begins with preparing a schematic using a computer-aided software. For this project, we use EAGLE, a design software package consisting of a schematic editor, a board builder and an auto router. Each of the electronic components is represented as a library consisting of a symbol and a footprint. A symbol is a representation of an electronic component in the schematic, while a footprint is the arrangement of pads on the PCB to physically attach and electronically connect an electronic component to the rest of the circuit.

We begin by using a schematic editor to draw a schematic of our design, and selecting the appropriate packages for the electronic components. The general rule that we follow when selecting the type of electronic components are:

1. **Surface-mount device (SMD) over through-hole devices (TH) when possible/available:** Since SMDs are usually smaller than THs, the reduction in size means that their inductance is generally lower than the latter. As such, SMDs are able to mitigate radio frequency (RF) noise better than THs. This is particularly important since our HAT PCBs are fabricated in a two-layer configuration and thus, signals conducted along the copper tracks will be prone to electromagnetic interference.

2. **Ease of assembly:** While smaller devices are better at mitigating RF noise, assembling them on the board could be difficult, considering time and resource constraints. We will restrict ourselves to selecting components with 0403 packaging.

Sketching a schematic is rather intuitive – Appendix A shows the complete schematic of our design.

3.2.8 Board Zoning and PCB Design Guidelines

A PCB board layout representation of the schematic is generated on EAGLE PCB Design software. At this stage, it is important that we arrange the electronic components according to the type of signals they generate and/or receive. This process is known as board zoning (*PCB Design Guidelines For Reduced EMI* 1999)

The features available on EAGLE student version limit our design to a two-layer board. In an ideal situation, we would want to isolate copper tracks running different voltages and signal types to separate layers. For example, there should be separate layers for +12V power lines, +3.3V power lines, digital and analog signals and ground. This is important because having copper tracks with different voltages running at opposite directions will induce electromagnetic disturbance or interference, and therefore, affecting the quality of the electrical signals conducted in the HAT. This effect will be greatly amplified in our application, since we are measuring electrical signals with relatively high frequencies.

As such, for this prototype, we have created two separate boards. The first board conforms to Raspberry PI's HAT specification, and it measures the frequency of the input signal. The voltage convention for the board is +3.3V and this matches the logic level system of the Raspberry Pi. The other board, which controls the relays, multiplexers and supplies power to the CS616 sensors, has a +12V (to power the sensors) and +5V (to power the multiplexers and relays) convention system. While ideally we would want these voltages to run on separate layers each, we find that for the prototype, the quality of these power lines will not drastically affect the performance of our system as long as this board isolated from the HAT board.

The first step to board zoning is to account for all the power requirements of the electronic components in our design. In our case, the DC voltages required are +12V and +3.3V, with a common ground GND. We note that our signalling convention is based on the Raspberry PI's, which runs on a +3.3 logic level, while +12V power is only required to power the CS616 sensor. As such, we would want to place the power jack that supplies +12V as close as possible to the 15 x JST connectors.

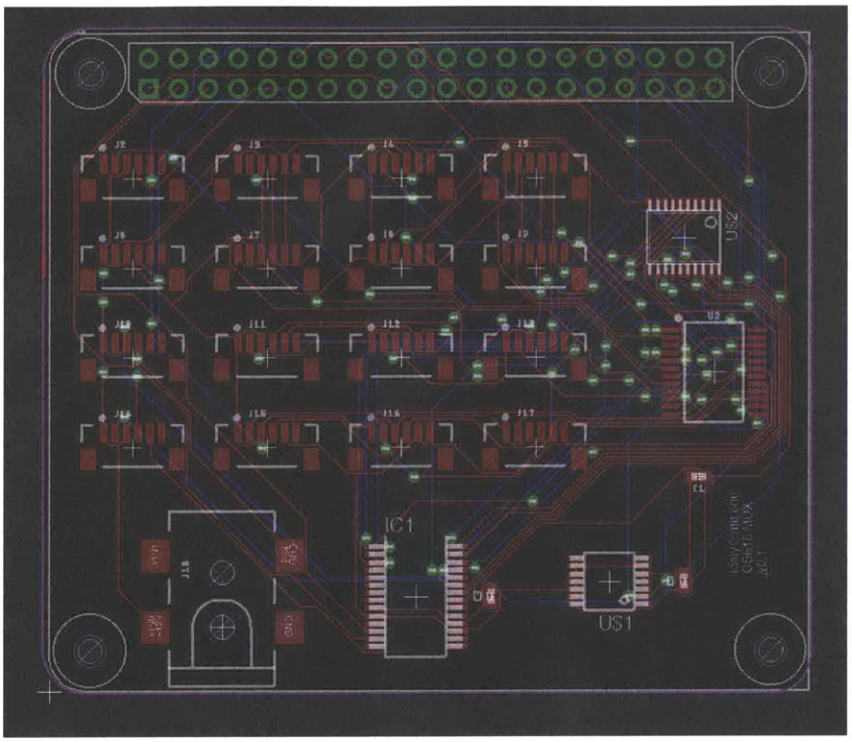


FIGURE 3.12: Raspberry Pi HAT PCB Board Layout

We place the remaining electronic components (SN74HC14N, CD74HC4067, SN74LV8154 and MCP23017) and their bypass capacitors to the bottom right of the board – this is a compromise between getting the components as close as possible to the +3.3V pin while restricting ourselves to a small work area. The SN74HC14N Schmitt Trigger is situated at the mid point between the multiplexer, binary counter and the JST connectors to reduce the effect of RF noise on the input CS616 signal. We note that the output signal coming from the CD74HC4067 multiplexer, and the power supplies and ground are susceptible to RF noise interference. As a result, we will lay the copper tracks for these routing manually

by hand. The rest of the routing is done automatically using EAGLE's autorouter, with the routing and track rules set according to the following table:

TABLE 3.3: PCB Design Rules

Description	Value
Dimensions	65 x 56 mm
Top Layer	0.035mm
Bottom Layer	0.035mm
Isolation	1.5mm
Wire, Pad and Via Clearance	8 mil
Minimum Distance between Copper Tracks	8 mil
Minimum clearance between Drill/Hole	8 mil
Minimum track width	8 mil
Thermal Isolation Width	10 mil
SMD Stop Mask	4 mil at 100 percent
SMD Cream Mask	0 mil

3.2.9 Power Management

The two major power supplies required by the devices and instruments of the system are 12V and 5V, powering the CS616 sensors and Raspberry Pi respectively.

Figure 3.13 shows the power distribution in the system. Power is sourced primarily from a 12V 7Ah lead-acid battery. To power the Raspberry Pi and its peripherals, we use an isolated DC/DC buck converter module supplying 5V at maximum current draw of 1A. 12V power from the battery is directly connected to the multiplexer breakout board.

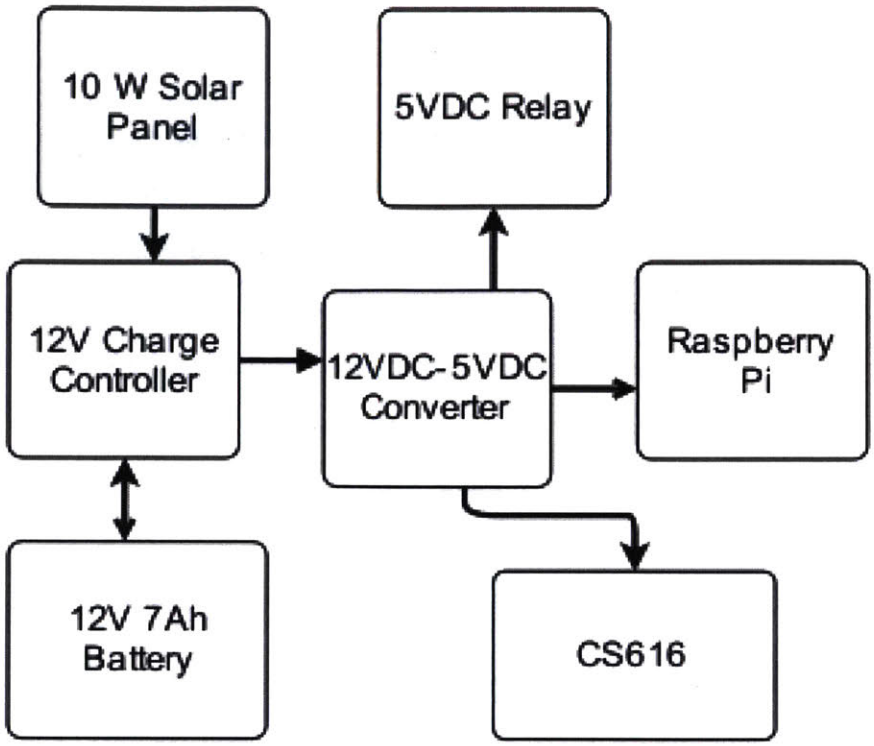


FIGURE 3.13: Flowchart of Power Management Daemon

We also connect a 10W solar panel to the 12V battery via a solar charge controller – this ensures that the battery will be sufficiently charged during sun hours so that it will have enough stored power to run the devices and instruments for an extended period of time.

The Raspberry Pi 3 B+ consumes a large amount of power even when it is in idle-state and not running any process-intensive programs (Geerling, 2015). In view of this, we substituted the Raspberry Pi B+ with a Raspberry Pi Zero (Zero). The Zero is less than half the size of a B+, and because it has fewer ports and peripherals than the B+, it consumes a significantly lower amount of power. Table 3.4 shows the current draw of both models of the Raspberry Pi (Geerling, 2015). Using the Zero, therefore, reduces the overall power consumption by about 60%.

In most soil moisture sensing applications, we find that the sample frequency is relatively low at about 2 mHz, or one reading per 10 minutes. To further reduce the power consumption of the system, we can turn off, i.e. put to sleep, the Zero as soon as soil moisture readings are made. When it is time to take readings again, we use a TPL5110 Nano Timer to wake the Raspberry Pi. In this application, the TPL5110 acts as a periodic timer, where as soon as it receives a signal indicating that a task is complete, it will disconnect power to the Raspberry Pi, and turn it back on after a pre-defined period of time. To set the wake-up timer, a resistor is connected to the delay pin of the TPL5110. For example to set a 10 min sleep time, we connect a 57.44 kOhm resistor to the delay pin. In this configuration, the battery life is expected to last up to 4 months on a full charge.

TABLE 3.4: Power consumption of Raspberry Pi B+ and Zero (Geerling, 2015)

Model	State	Current Draw
B+	Idle, HDMI, LED Disabled	220 mA
Zero	Idle, HDMI, LED Disabled	80 mA

Figure 3.14 shows the flow of the TPL5110 Nano Timer power management cycle in our system.

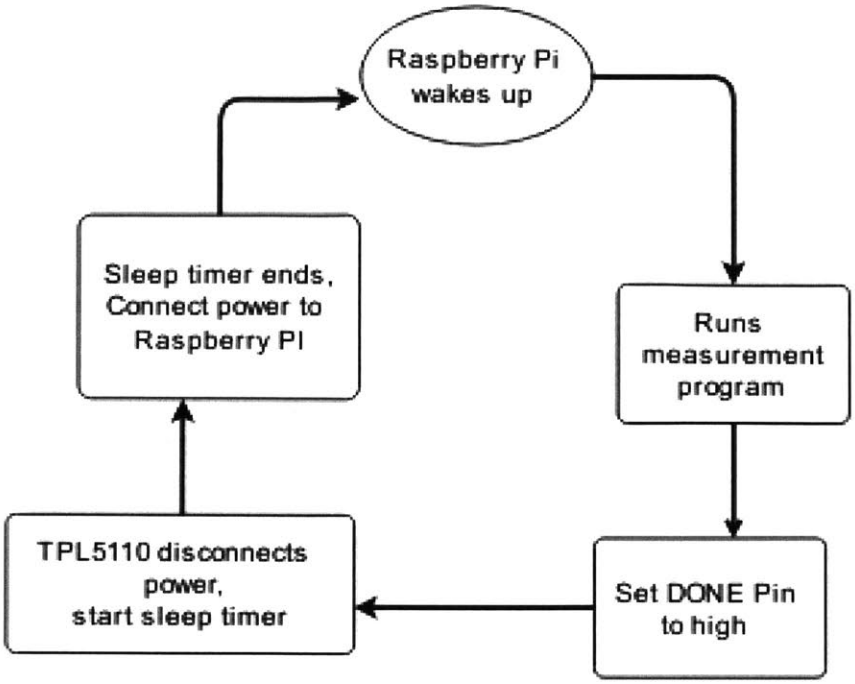


FIGURE 3.14: Flowchart of Power Management Daemon Service

3.2.10 Real-Time Clock

The Raspberry Pi does not have a built-in hardware clock to keep track of time when power is not available. This means that whenever we turn off the Raspberry Pi, it will instantly forget the time, and we will have to manually set the time again the next time we turn the Raspberry Pi on. As such, we added a Real-time Clock (RTC) module using a DS1307 RTC from ABElectronics. This module comes in a HAT form-factor, and therefore, it is compatible with the HAT that we have designed. Installing the RTC is a relatively easy endeavour – we stack the RTC HAT on top of our HAT, and we send commands to the DS1307 to store and maintain the current date and time. A CR2032 battery provides power to the RTC HAT when power is not available.

3.2.11 Prototype Packaging

We encapsulate the entire system in a Nanuk 904 Protective Case of dimensions L213mm x W152mm x H94mm. Three holes are drilled at the side of the case to feed sensor and solar panel cables into the system. For this prototype, we will connect two CS616 reflectometers and as thus, we will only need two holes fitted with cable glands.

Figure 3.15 shows the internal layout of the system in its protective case, while Figure 3.16 is an image of a fully assembled system that is ready for deployment.

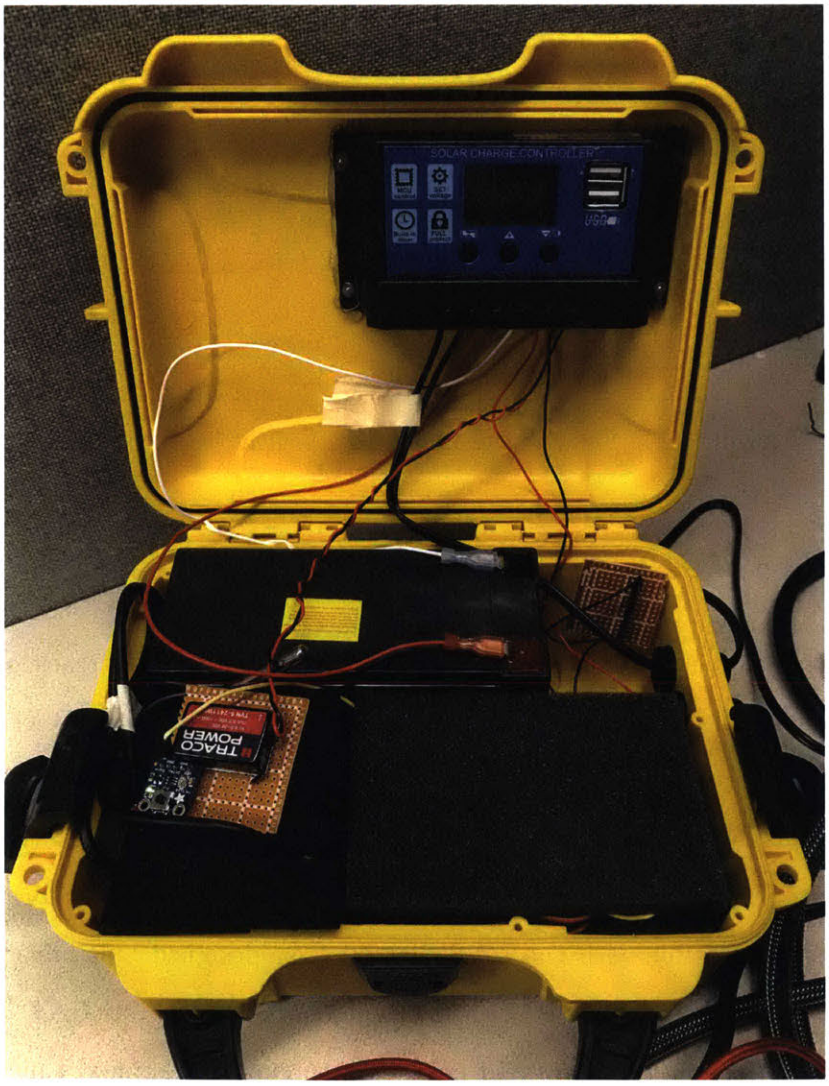


FIGURE 3.15: Internal layout of Soil Moisture Sensing System

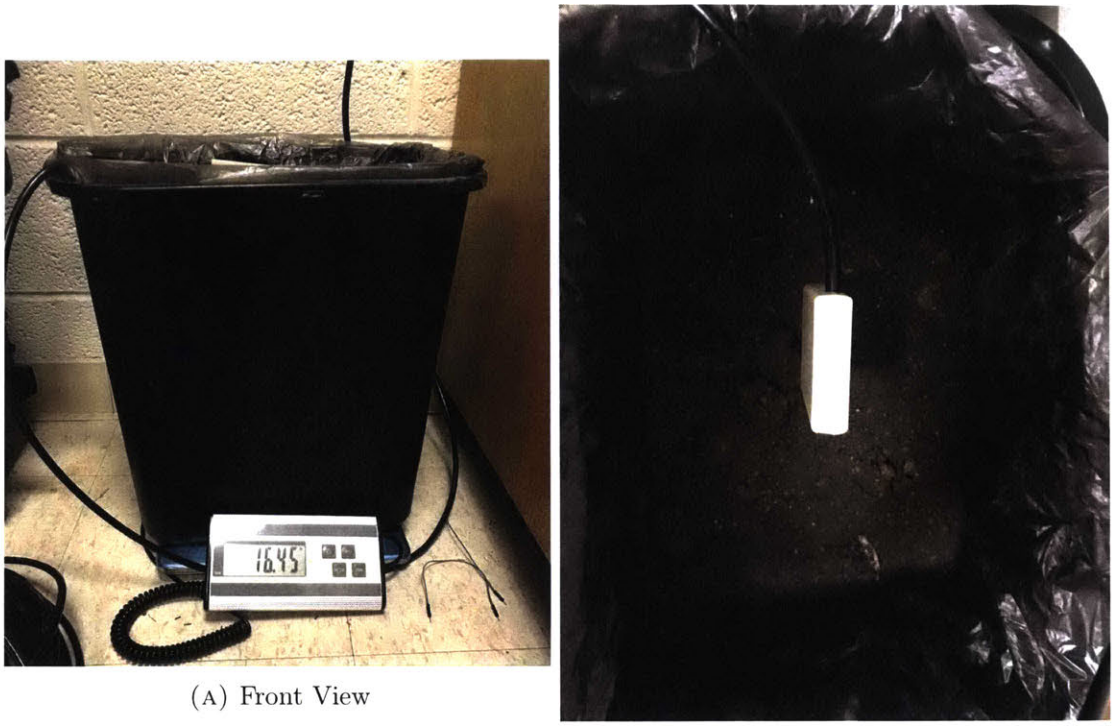
FIGURE 3.16: Soil Moisture Sensing System Prototype in Protective Case

Chapter 4

Evaluation

4.1 Stability of Sensor Readings

In Section 3.2.3, we find that the frequency of the output signal from the CS616 sensor is about 68 kHz when its rods are in the air, and 24 kHz when the rods are fully immersed in water. We, therefore, expect to get similar results when we acquire the output signal from the CS616 using our HAT under the same conditions.



(A) Front View

(B) Top View

FIGURE 4.1: Laboratory Set-up

We set up the experiment as shown in Figure 4.1. Throughout the course of this experiment, it is essential that the probe rods are kept as perpendicular to the water or ground level as possible – any deviation from the perpendicular will result in a lower frequency reading than the normal, thus affecting the accuracy of the water content measurement. This set-up involves two data acquisition devices, a Rigol DS1102B Oscilloscope and a Raspberry Pi attached with the HAT that we have designed. The signal emitted by the CS616 sensor is sent to both these devices, and we compare the difference in the readings.

We first measure the frequency of the CS6161 output signal, and subsequently, the water volumetric content when its probes are in the air. Figure 4.2 shows the CS616 output signal as acquired by the oscilloscope

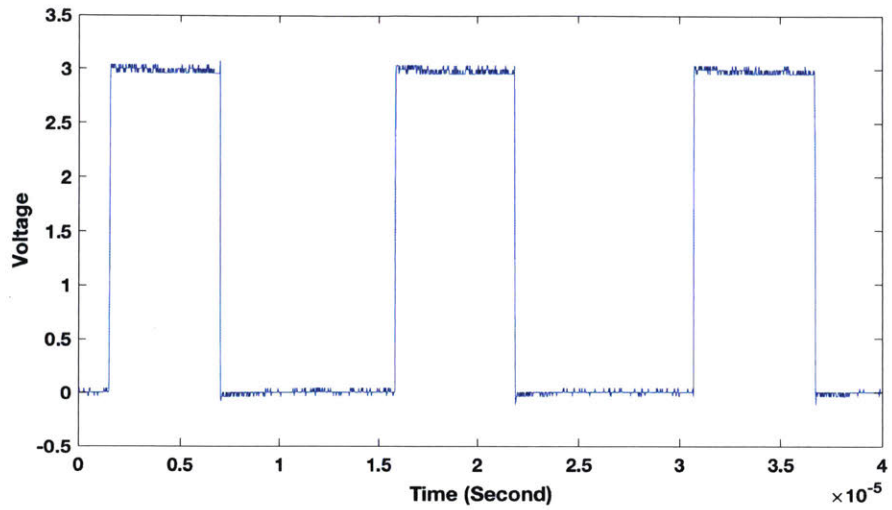


FIGURE 4.2: Output signal of CS616 sensor as seen from an oscilloscope

The period and the frequency of this signal are 14.92 microseconds and 67.024 kHz respectively.

We repeat the same experiment, this time, acquiring and storing data using the Raspberry Pi with the HAT attached. We note that the data obtained from the Raspberry Pi will be in the form of discrete frequency readings over a period of time, stored in the CSV format. The chart in Figure 4.3 below shows a plot of the results from this trial, and a straight-line plot of the average signal frequency as acquired from the oscilloscope.

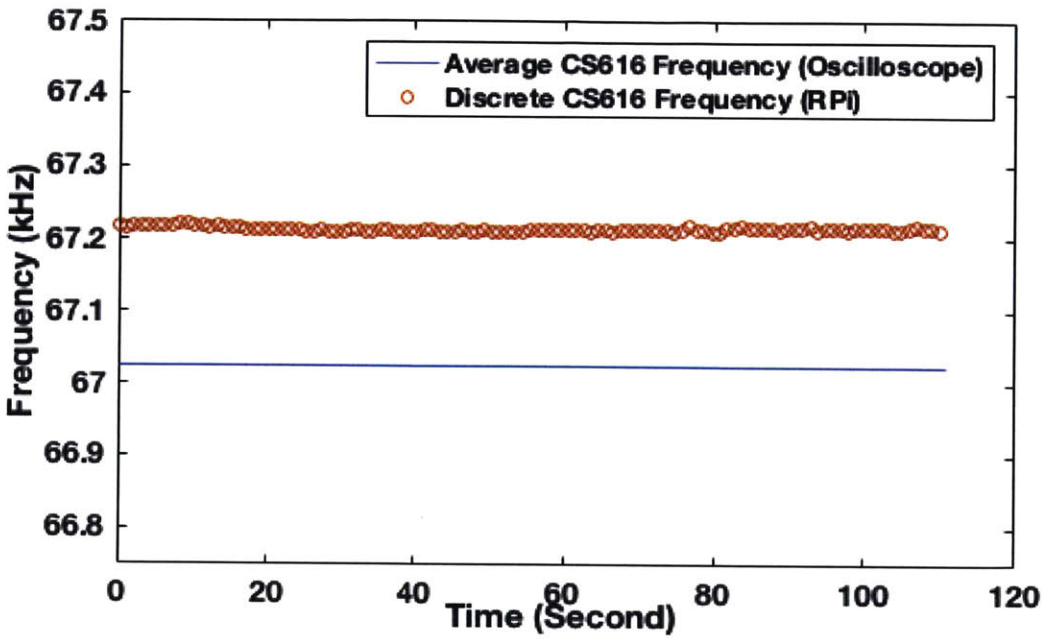


FIGURE 4.3: Stability of CS616 Output Signal Readings from HAT

The relative error is defined as:

$$RelativeError = \frac{AbsoluteError}{TrueValue}$$

$$= \frac{|MeanFrequency_{HAT} - MeanFrequency_{Oscilloscope}|}{MeanFrequency_{Oscilloscope}}$$

Similarly, the relative error percentage is defined as:

$$RelativeErrorPercentage = \frac{AbsoluteError}{TrueValue} * 100\%$$

$$= \frac{|MeanFrequency_{HAT} - MeanFrequency_{Oscilloscope}|}{MeanFrequency_{Oscilloscope}} * 100\%$$

From the data we have obtained, we find that the mean of the discrete CS616 readings from the Raspberry Pi (67.213 kHz) has a relative error percentage of 0.28% when compared with the average CS616 readings from the oscilloscope (67.024kHz).

Next, we submerge the probe in a container of water. Again, it is important that the probe is as perpendicular to the water level as possible, and that the rods are fully submerged in the water. Figure 4.5 shows the CS616 output signal as acquired by the oscilloscope.

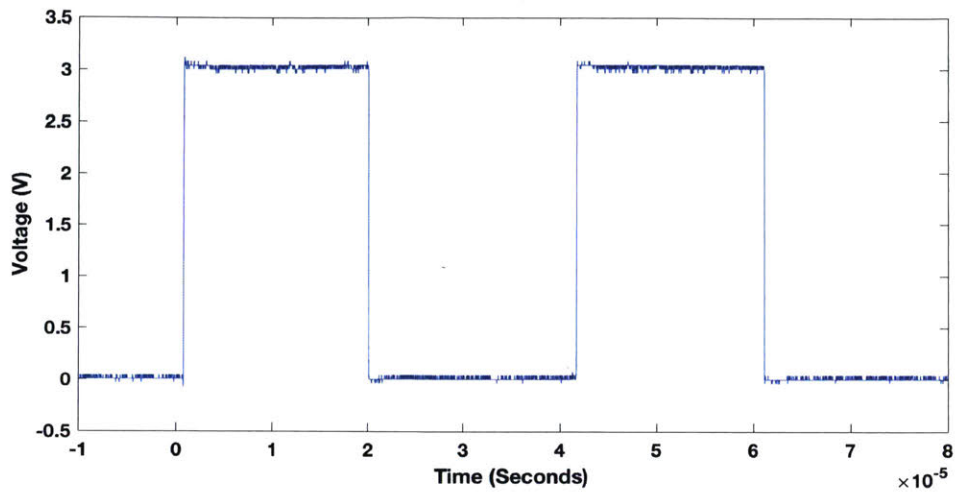


FIGURE 4.4: Output signal of CS616 sensor as seen from an oscilloscope (Probes Fully Submerged in Water)

Figure 4.4 shows a plot of the discrete frequency readings over a period of time, and a straight-line plot of the average signal frequency acquired from the oscilloscope.

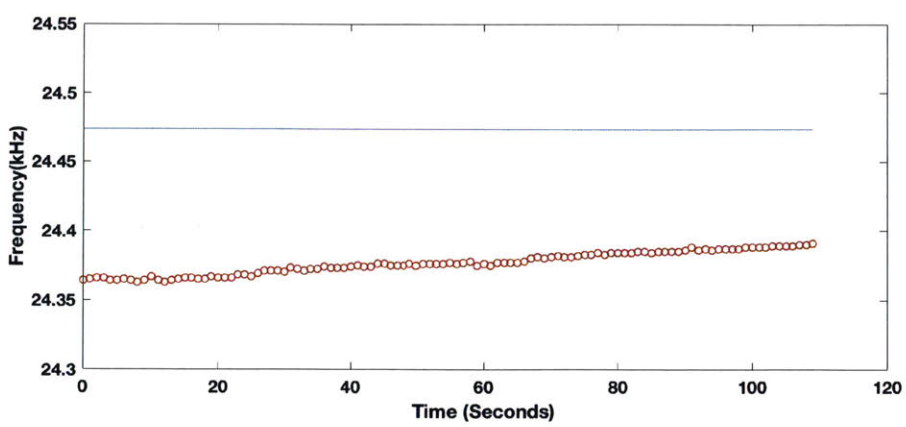


FIGURE 4.5: Stability of CS616 Output Signal Readings from HAT (Probes in Water)

We find that the mean of the discrete CS616 readings from the Raspberry Pi (24.376kHz) has a relative error percentage of 0.400% deviation when compared to the average CS616 readings from the oscilloscope (24.474kHz) when the sensor is fully submerged in water.

The low relative error percentages for readings taken under contrasting experimental conditions show that our design works well. The relative error can be attributed to the precision of the input clock signal into the SN74LV8154 binary counter. As mention in Section 2.2.5, the reciprocal counting method's precision depends on how precise the input pulse is. The relative error percentage for the clock signal is found to be:

$$RelativeErrorPercentage = \frac{AbsoluteError}{TrueValue} * 100\%$$

$$= \frac{|Period_{ClockInput} - 1second|}{1second} * 100\% = 0.02\%$$

This 0.02% error in the input clock signal is reciprocated in the error in counting of the pulses from the CS616 output signal by the SN74LV8154 binary counter. Having said that, this error is insignificant when compared to the magnitude of the frequency reading and as such, it can be ignored.

4.2 Linearity of sensor output data

The linearity of a sensor output data is defined as "the difference of bias throughout the expected operating (measurement) range of the equipment" (Raytheon, 2007). In this experiment, we will analyse the linearity of the data we obtained using two different data acquisition equipment, the oscilloscope and the HAT. This is done by taking 10 sets of readings from the CS616 spanning the range of the measurement using the HAT, and comparing them with the reference data obtained using the oscilloscope.

From Section 2.2.1, we find that the depth of which the CS616 probe rods are inserted is directly proportional to the frequency of its output signal. Therefore, by submerging the CS616 probe at various depths, and comparing the resulting frequency of the output signal

with those obtained using an oscilloscope, we can investigate the linearity characteristics of our HAT. We expect the results obtained using the HAT to be similar, with minimal deviation, to the oscilloscope.

Figure 4.6 shows the frequency of the CS616 output signal as the depth of which the probe rods are submerged is varied, while Figure 4.7 shows a plot of the measured frequency value obtained from the HAT against the true frequency value from the oscilloscope.

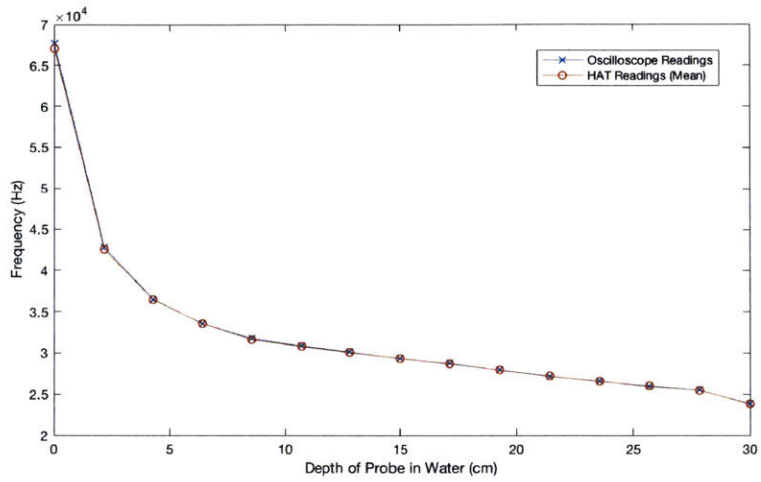


FIGURE 4.6: Linearity: Output signals of CS616 sensor as seen from an oscilloscope and HAT

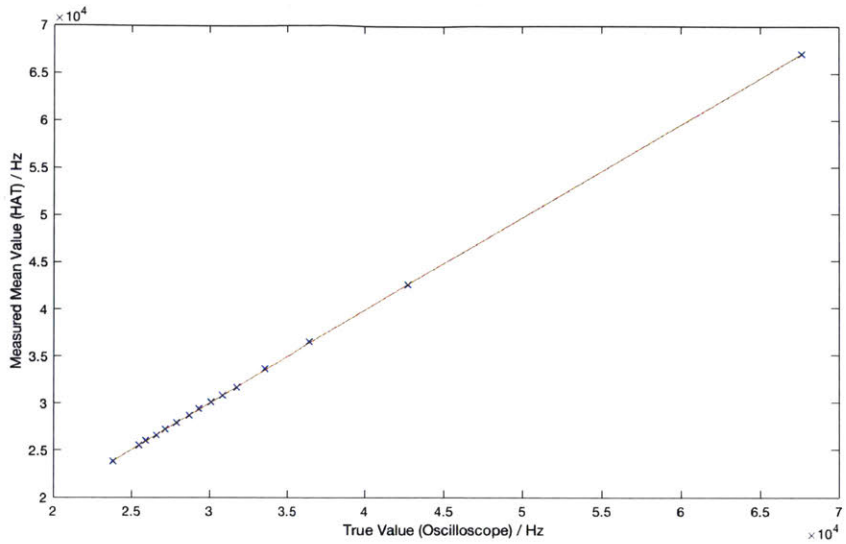


FIGURE 4.7: Linearity: Relationship between true value and measured value of the frequency of CS616 output signal

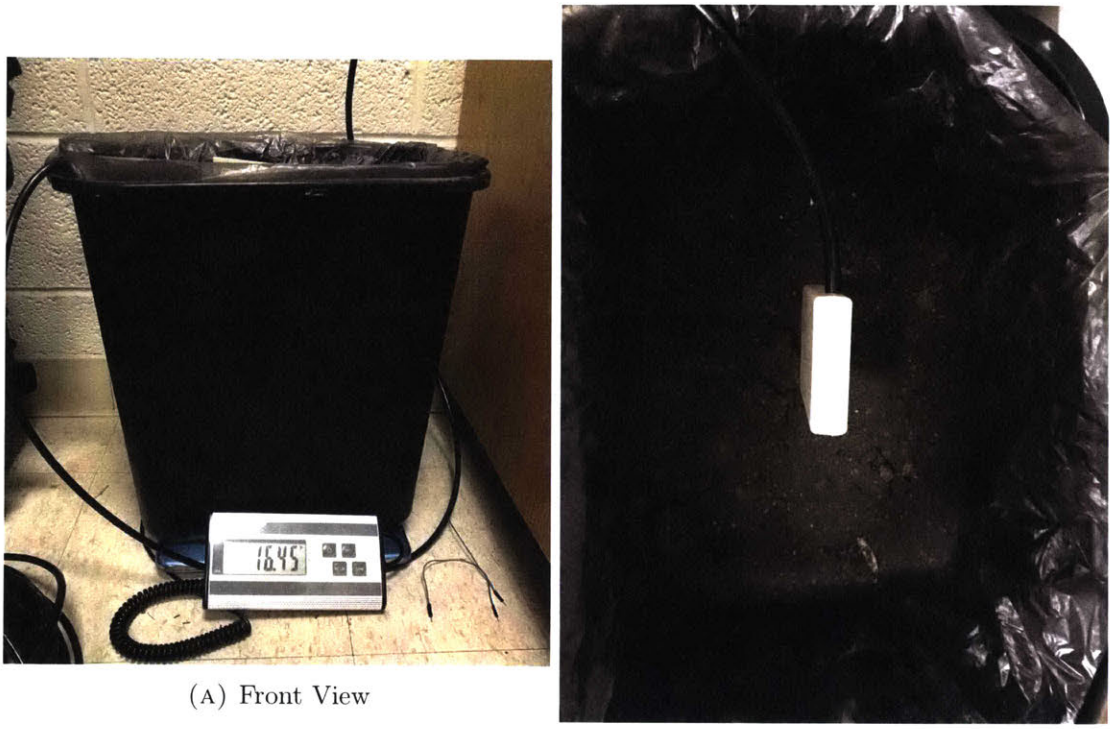
As seen in Figure 4.7, we find that the values acquired by both equipment match, and we plot a linearity graph to deduce the relationship between the measured value from the HAT to the reference value from the oscilloscope. Using polynomial curve fitting of order 1, the best fit line that illustrates the relationship between these two parameters has a gradient of 0.9855 with a constant bias of 398.4166. Since the gradient is very close to unity, we can conclude that the linearity between the HAT and the oscilloscope is almost perfect, and fits our requirement.

We note that this experiment is a test of the linearity of the HAT, rather than the CS616 sensor. Instructions to test the linearity of the CS616 sensor can be found in the product manual listed in the bibliography at the end of this thesis.

4.3 Performance Evaluation in a Controlled Environment

In previous sections, we find that the soil moisture data acquired by our HAT matches those from an oscilloscope, thus indicating that our readings are precise. For our next

experiment, we would want to test if the data that we received is an accurate representation of the actual volumetric water content in the soil. We do this by adding a known amount of water to a soil medium, and account for the loss of water over a period of time.



(A) Front View

(B) Top View

FIGURE 4.8: Laboratory Set-up

We insert a single CS616 sensor into this medium, in this case top soil, as shown in Figure 4.8. Note that the probe has to be as close to the perpendicular with respect to the ground as possible. We then take volumetric water content readings, in the form of frequencies, from the sensor at every minute and store these values in a comma-separated-values file in the Raspberry Pi's SD card.

We define the results obtained from this experiment as the *analytical volumetric water content.*

Next, we take daily spot readings of the volumetric water content of the same experiment set-up. We place the container filled with the soil medium on a digital weighing scale, and note its weight and define this as m_{dry}. In this case, the weight of the soil medium

and the container, m_{dry}, is found to be 14.7 kg. We then add 2.5 liters of water, and we find that the weight is now 17.2 kg. We define this weight as m_{wet}.

We then calculate the gravimetric water content using the equation:

$$\theta_g = \frac{m_{wet} - m_{dry}}{m_{dry}}$$

To derive the volumetric water content we find the the bulk density, ρ_{bulk}:

$$\rho_{bulk} = \frac{m_{dry}}{volume_{cylinder}}$$

where $volume_{cylinder}$ is found to be 30.8 liters, or 0.0308 m^3.

Thus, the volumetric water content, θ_v, can be calculated from the product of ρ_{bulk} and θ_g:

$$\theta_v = \rho_{bulk} * \theta_g$$

We define the results obtained from this experiment as the *experimental volumetric water content.*

Figure 4.9 shows the volumetric water contents obtained using the three methods – spot readings (experimental), and using the quadratic and the linear formulae (analytical) with the inverse of the CS616 signal frequencies, or period T as the input variable.

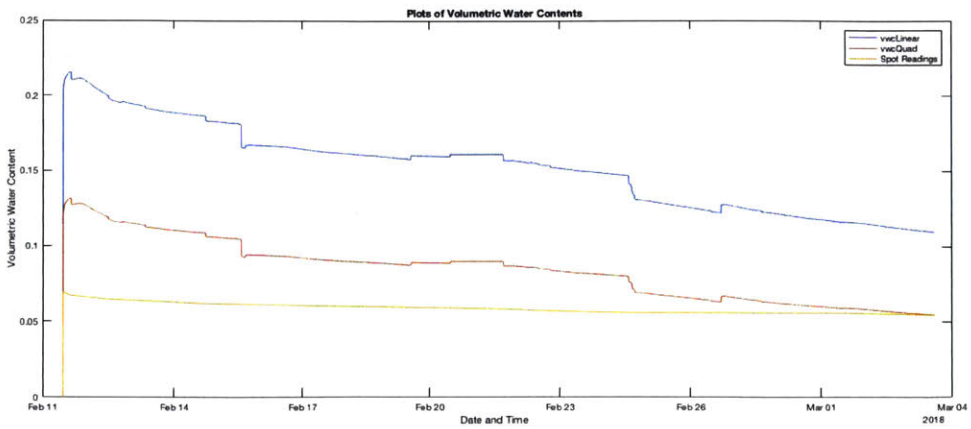


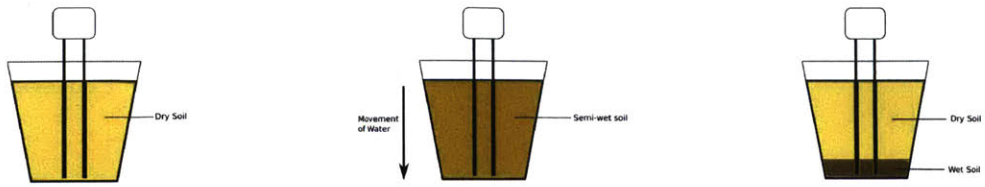
FIGURE 4.9: Volumetric Content Over Tim

We see that all three graphs show a decreasing trend where the volumetric water content decreases as a function of time. This is caused by the gradual evaporation of water from the soil medium, which in turn lowers the weight of the experimental set up and subsequently, θ_g.

In the CS616 product manual, Campbell Scientific states that the response described by using the linear equation is less accurate than the quadratic equation (*CS616 and CS625 Water Content Reflectometers* 2016). Therefore, we will analyse the results of this experiment by comparing only the response obtained using the quadratic equation with the volumetric water content obtained experimentally.

From the plots, we note that there is a large, however decreasing, discrepancy between the experimental and analytical volumetric water contents, θ_v. We can attribute this to the design of our experiment.

Immediately after pouring 2.5 liters of water into the medium (Feb 11), we find that the experimental θ_v (vwcQuad) is significantly higher than the experimental θ_v. Since the soil medium is relatively compacted, we expect the infiltration rate to be low and that water will percolate to the bottom of the container at a slower rate. The CS616 sensor



(A) Initial state of medium before adding water

(B) Transient state of medium after adding water

(C) Steady state of medium after adding water

FIGURE 4.10: States of soil medium throughout experiment

will therefore sense a higher θ_v since most of its probes are exposed to wet soil during this time period.

After three weeks, we find that the experimental and analytical θ_v matches. The soil medium and water have reached steady state, where there is pooling of water at the base of the container since there is no drainage in our experimental setup. In this state, the response from the CS616 sensor is found to be an accurate representation of the actual θ_v of the soil.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we have described the development of a self-contained Hardware-Attached-on-Top electronic add-on board for the Raspberry Pi platform that acquires volumetric water content data from Campbell Scientific CS616 reflectometers. We conducted a series of experiments to assess the linearity and stability of the response obtained using the HAT, and found that the results are as precise and accurate as the sensor permits.

Using this HAT, we have built a prototype of a fully-functional multiple-channel soil moisture sensor system in a deploy-and-forget configuration. This prototype is expected to withstand the rigours of outdoor use or harsh environments, and is able to continually run and collect data for long periods of time.

5.2 Future Work

Major improvements could be made on the design of the HAT and the sensor data acquisition system that is based around it. The design of the HAT is mainly limited by the fact that we were not able to fabricate PCB with more than 2 layers due to the version of EAGLE that we are using. As such, we had to fabricate a separate electronic board for relay control to minimize electromagnetic interference caused by mixed-signal crosstalk.

Multilayering the PCBs means that we could isolate different signals to separate planes since there is an increase in the copper area available for wiring. We could therefore integrate relay control into our design, making our board more compact and robust.

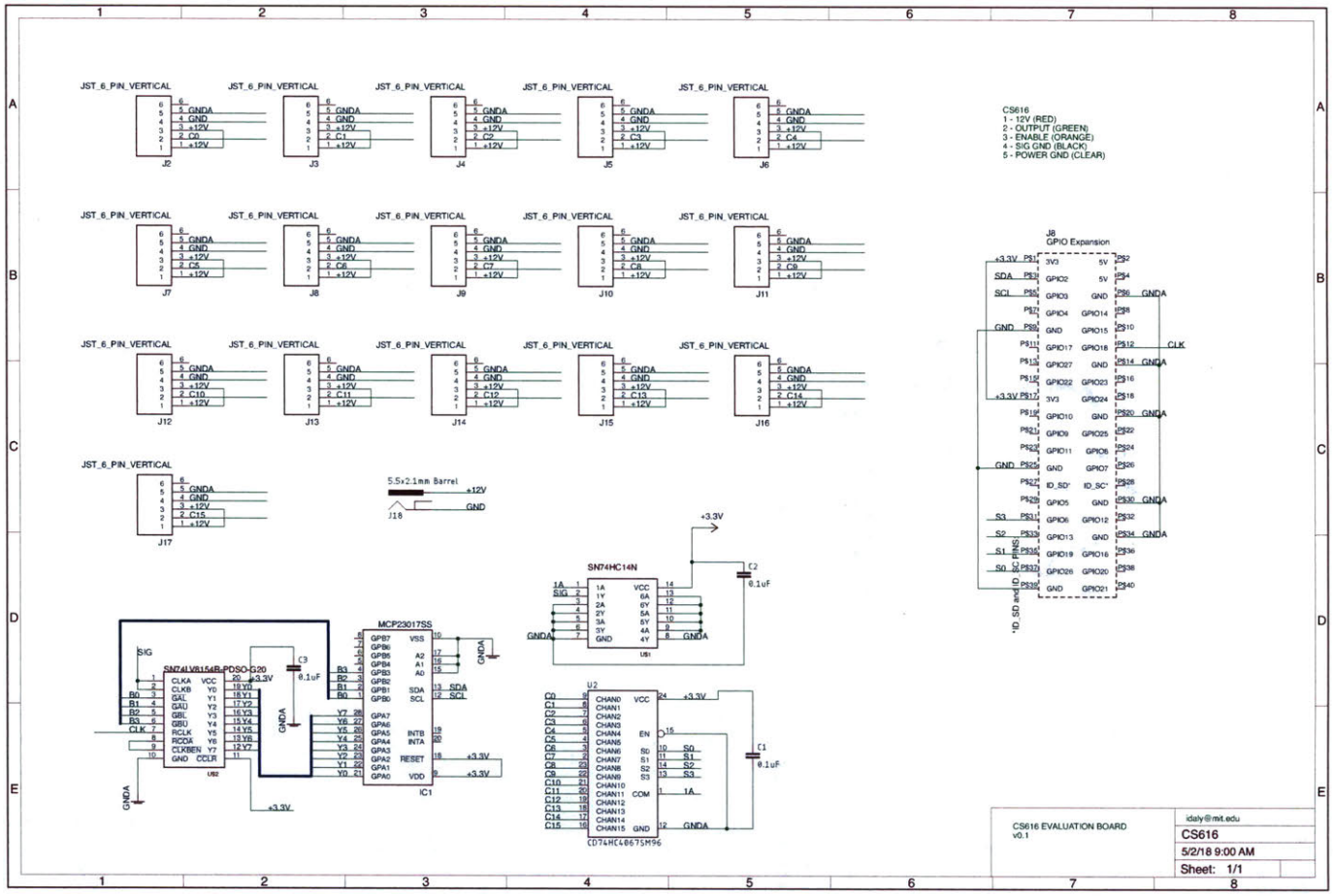
We could also improve on the hardware that we use to encapsulate our sensor system. For example, we used cable glands to feed CS616 sensor cables into the waterproof enclosure. The gap between each cable and its gland, and the glands and the enclosure itself, introduce the possibility of water seeping into the internal space of the enclosure. A better alternative is to use circular bayonet connectors; these connectors provide a complete seal to withstand moisture, condensation, vibration and flashover while at the same time, feeding electrical power and signals to the electronics encapsulated in the enclosure.

Lastly, we could replace the Raspberry Pi Zero with a custom-built data logger that fits specifically to our requirements. For this project, we simply needed a machine that could log frequencies over long periods of time and store these in a file on an SD card, and in view of the limited time that we had, we chose the Raspberry Pi because of our familiarity with this platform. The Raspberry Pi Zero, while being small in terms of its size and energy consumption, is capable of doing complex operations for a variety of tasks far beyond the scope of this project and as such, it has many electronics embedded in its design that are otherwise redundant for our requirements. By building our own data logger circuit and using only essential IC chips and electronic components, we can further reduce the power consumption of our setup and consequently, extending the battery life of our sensor system.

Appendix A

CS616 Hardware Attached on Top (HAT)

A.1 Schematic of CS616 HAT



30/3/18 3:33 PM f=0.66 /Users/idaly666/Documents/eagle/cs616/CS616.sch (Sheet: 1/1)

A.2 Image of CS616 Board

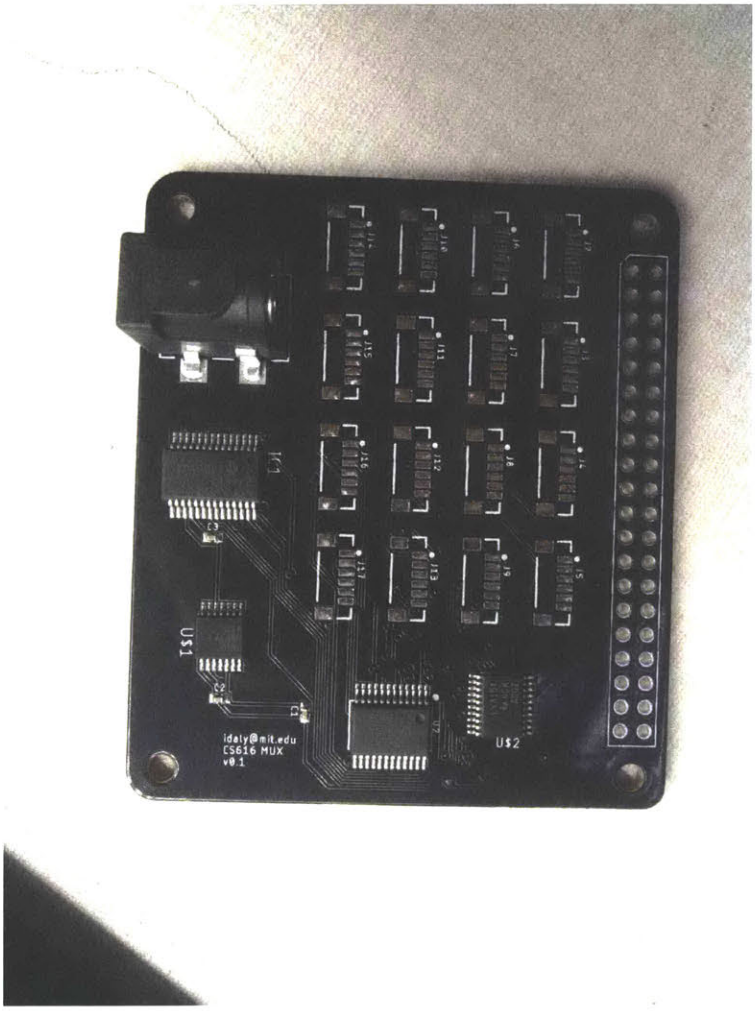


FIGURE A.1: Semi-populated Hardware-Attached-On-Top to measure frequency signals from CS616 sensor

Bibliography

Cobb, Alexander R. et al. (2017). "How temporal patterns in rainfall determine the geomorphology and carbon fluxes of tropical peatlands". In: *Proceedings of the National Academy of Sciences* 114.26, E5187. DOI: 10.1073/pnas.1701090114. URL: http://www.pnas.org/content/114/26/E5187.abstract.

CS616 and CS625 Water Content Reflectometers (2016). URL: https://s.campbellsci.com/documents/af/manuals/cs616.pdf.

Frequency Measurement Guidelines for Oscillators (2013). URL: https://www.sitime.com/api/gated/AN10033-Frequency-Measurement-Guidelines-for-Oscillators.pdf.

Fundamentals of the Electronic Counters (1997). URL: http://leapsecond.com/hpan/an200.pdf.

Geerling, Neff (2015). *Raspberry Pi Zero - Power Consumption Comparison.* URL: https://www.jeffgeerling.com/blogs/jeff-geerling/raspberry-pi-zero-power.

Kirianaki, Nikolay V., Sergey Y. Yurish, and Nestor O. Shpak (2001). "Methods of dependent count for frequency measurements". In: *Measurement* 29.1, pp. 31–50. ISSN: 0263-2241. DOI: 10.1016/S0263-2241(00)00026-9. URL: http://www.sciencedirect.com/science/article/pii/S0263224100000269.

PCB Design Guidelines For Reduced EMI (1999).

Raytheon (2007). *Measurement Systems Analysis.*