Cross-Needle Analog Meter to Monitor Charge Time for Lithium-Ion Single Cell

A thesis

submitted in fulfilment

of the requirements for the Degree

of

Master of Engineering (Electronics)

at

The University of Waikato

by

Hiroshi Mohri

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2018

Abstract

The Cross-Needle panel meter is an analog Standing wave ratio meter with a modified panel. It consists of two meters which are mirrored horizontally and 38mm apart. One needle indicates the forward travelling wave on a transmission line, while the other needle indicates reverse travelling wave. A third scale, occupying the area swept by both needles, shows Standing Wave Ratio (SWR), essentially reflection coefficient, typically of an antenna connected to a radio transmitter. This thesis reports on an adaption of this idea. The first needle shows current into a battery, the second needle indicates the voltage of the battery. There is also a third scale using contour lines which show the amount of time left until the battery is fully charged.

The contour lines are calculated and drawn using analytic geometry by predetermining the rotation of the two needles. Hence the amount of time left for the battery to fully charge can be read by simply looking at where the the two needles cross and estimating from the closest contour line that displays charge time. The input current scale is expanded logarithmically with a wide range from 1mA to 10A. This scale also evenly spreads the contour lines from 10 minutes to 1 week as opposed to a linear current scale, which causes the contour lines beyond 6 hours to cluster making it difficult to observe the charge time at low charge current inputs. The battery voltage scale is offset to suit the battery voltage range.

The meter has been implemented using an embedded microprocessor on a PCB that mounts at the rear of the meter housing. The prototype has been calibrated for a single 18650 Lithium-Ion battery which is a popular scenario for portable equipment. This project provides a manufacturing solution for a low cost, low power, portable Lithium-ion battery monitor which can be easily replicated.

Acknowledgements

The author would like to thank:

Professor Jonathan Scott for supervising, guidance, suggesting ideas, reviewing and provision of solutions, literature documents and hardware.

Benson Chang for provisioning electronic parts, various hardware and assistance in designing a PCB footprint.

Rahat Hasan for assistance with using the measuring instruments for the State of charge for the battery and the Schottkey diode I-V curves.

Vicki MacDonell for proof reading and reviewing.

Department of Science and Engineering for provisioning Post graduate student desk, internet and laboratory equipment.

The University of Waikato library for general resources, software resources and internet resources.

Contents

Chapter 1

Introduction

Rechargeable batteries are becoming more common in modern electronic devices, ranging from smartphones, laptops, drones and even electric vehicles. One of the commonly used rechargeable batteries is the 18650 Lithium-Ion single cell. It has proven to be safe, reliable, commercially available, affordable and is able to endure many discharge and charge cycles.

There are many applications for monitoring battery charge, such as mobile apps, mains powered wall chargers or simply a small LCD screen connected to a micro-controller. There may be users who prefer to visualise readings through analog meters rather than to simply view numbers shown on a LCD screen or a 7 segment display. An example of this would be the speedometers in most modern cars which still use analog meters. Also there are just as many analog SWR meters available in the market as well as digital SWR meters.

1.1 Expanded scale Analog Voltmeter

An analog voltmeter to monitor single Lithium-ion cell had been developed in 2016 which uses a LM285 2.5V band gap reference to measure the cell terminal voltage. This voltmeter displays onto a single needle pointer analog meter and an expanded scale from 3V to 4.3V [1].

1.2 Determining State of Charge

There are various State of Charge(SoC) estimation methods for Lithium ion batteries such as coulomb counting, terminal voltage, impedance spectroscopy, Kalman filtering, temperature dependence or a combination of these methods.

Coulomb counting requires regular re-calibrations and good current measurements. Impedance spectroscopy is costly and temperature dependent. Kalman filtering can be complicated with algorithms that considers all the aspects [5]. Reference [2] states that to measure Independence only to determine the SoC for a Lithium-Ion cell would be inaccurate at lower SoC. This was also shown by measuring AC impedance analysis in [3] with errors of 2.6% at 50% SoC and 8.6% at 0% SoC. Change in ambient temperatures has small effect on battery EMF. A small increase in EMF difference of about 42mV between 5°C to 25°C and small decrease of about 33mV between 25°C to 45°C [5].

1.3 Photo-voltaic cell output power

Ideally a photo-voltaic cell will be used in this project to provide charge current to the battery which can output a current that varies significantly with lighting conditions. A SunPower A300 solar cell was tested in locations with different lighting conditions and found that outputs about $20.9mW/cm^2$ outdoors when the sun is approximately at zenith, $3.03mW/cm^2$ outside the sun on a car dashboard, $0.42mW/cm^2$ Indoors under a lamp and $0.04mW/cm^2$ next to a window indoors [6]. An expanded current scale is required to monitor this large variation in input current.

1.4 Determining Schottkey Diode I-V characteristics

The I-V characteristics of a Schottkey diode are approximately linear with a semi-log scale. There are many parameters which affect the current characteristics of Silicon type Shottkey diode namely being, series resistance, temperature, ideality factor and saturation current. However, in practice the actual model is linear (semi-log scale) between a certain current range and much research has been done to model a function which accommodates the non-linear regions of the actual model. Outside these linear regions (at lower currents and higher currents) of the model other parameters begin to affect the I-V characteristics, such as the change in ideality factor, series resistance, diode area, barrier height and tunneling energy. Different methods are used to determine these parameters [7] which compares Cheung's method and Norde's method and it was found that Cheung's method could only be applied to the non-linear regions.

1.5 Outline

One approach to monitor battery charge time is to modify a pre-existing cross-needle analog SWR, where the needles cross at a point that indicates standing wave ratio. This idea can be useful for a battery charge monitoring application, where one needle displays battery charge by using the analog voltmeter features from [1] and the other displays input charge current, the two needles cross at a point to show a third scale of the time remaining until battery reaches full charge. The purpose of this is for quick referencing battery charge time monitoring. The process of implementing this idea and the accomplishments made from this project are described in the following chapters.

Chapter 2 discusses the process of designing a custom cross-needle meter front panel with charge time contour lines. The dimensions of the SWR meter

were measured to produce linear equations to simulate the deflection of the two needles, these linear equations for each of the needles were then used to calculate intersection points in a C# program to produce and visualise contour lines by joining these points. The current scale was changed to a non-linear scale to expand the current range because the meter would ideally read both small current as well as large current. Different expanded current scales and their corresponding contour lines were compared and assessed in order to select the current scale that forms the most elegant set of contour lines. The custom meter panel was then drawn to scale in Solid Works, printed then applied to the front panel of the SWR meter.

Chapter 3 discusses the tests and measurements of the hardware used for the project. The state of charge was modeled by pulse charging and discharging to approximate internal impedance and to learn the characteristics of a 18650 Lithium-Ion cell. Deflection of the analog meter was tested by changing the pulse width of a square wave signal using a function generator to confirm linearity. A digital multi-meter was used to measure full scale deflection current and meter resistance. Measurements were done to model the Schottkey diode forward voltage and current characteristics from 1mA to 10A at temperatures between 5°C to 45°C so that the firmware could read in temperature and diode voltage, and calculate log of current.

Chapter 4 discusses the details of the design such as the LTspice simulation of the opamp to compare gain and select an appropriate input resistor. The Circuit diagram shows the micro-controller pin allocation and the components used. A Custom foot print for the pads was created in order to have the PCB mount on the back on the panel with enough strength. The purpose of each component is also described.

Chapter 5 discusses the firmware used to program the micro-controller. The programming is described with a list of functions and flow charts. Measurements and calculations done by the micro-controller are explained with formulas. The log of charge current formula was linearised to simplify calculations

and conserve memory. The Deflection of the two needles were accomplished using pulse width modulation. Two timers were used in the firmware, Timer 2 for the calculations and Timer 0 was used to produce the second independent signal since the PIC16F684 has a single independent PWM by default.

Chapter 6 discusses the results of the project. Both the voltage meter and the logarithmic current meter have proven to have an exceptional accuracy despite changes in temperature conditions. The testing was done inside a incubator to test consistency with ambient temperature variation.

Chapter 2

Cross-Needle meter panel design

2.1 Overview

The SZ-70 Standing Wave Ratio(SWR) is a dual needle analog meter Manufactured by Flash Star Industrial, was Purchased online. This cross needle meter was disassembled and scanned and measurements of the dimensions for the panel were taken with Vernier calipers, ruler and a protractor as some of the required dimensions were not provided by the manufacturer. The dimensions of the SZ-70 are fundamental in creating an expanded scale custom panel and for the PCB mounting which will be discussed in chapter 4. In the custom panel, the two needles each display the battery charge and the input charge current and between these scales there are contour lines which indicate the time remaining for the battery to completely charge. It is essential that the contour lines can be easily read because the user would ideally tell the remaining charge time with out looking at the meter for too long.

2.1.1 Measuring the dimensions

The scanned image of the SZ-70 with the front panel removed then the measurements were recorded. The scan was expanded and scaled to twice as large

to make measurements easier.

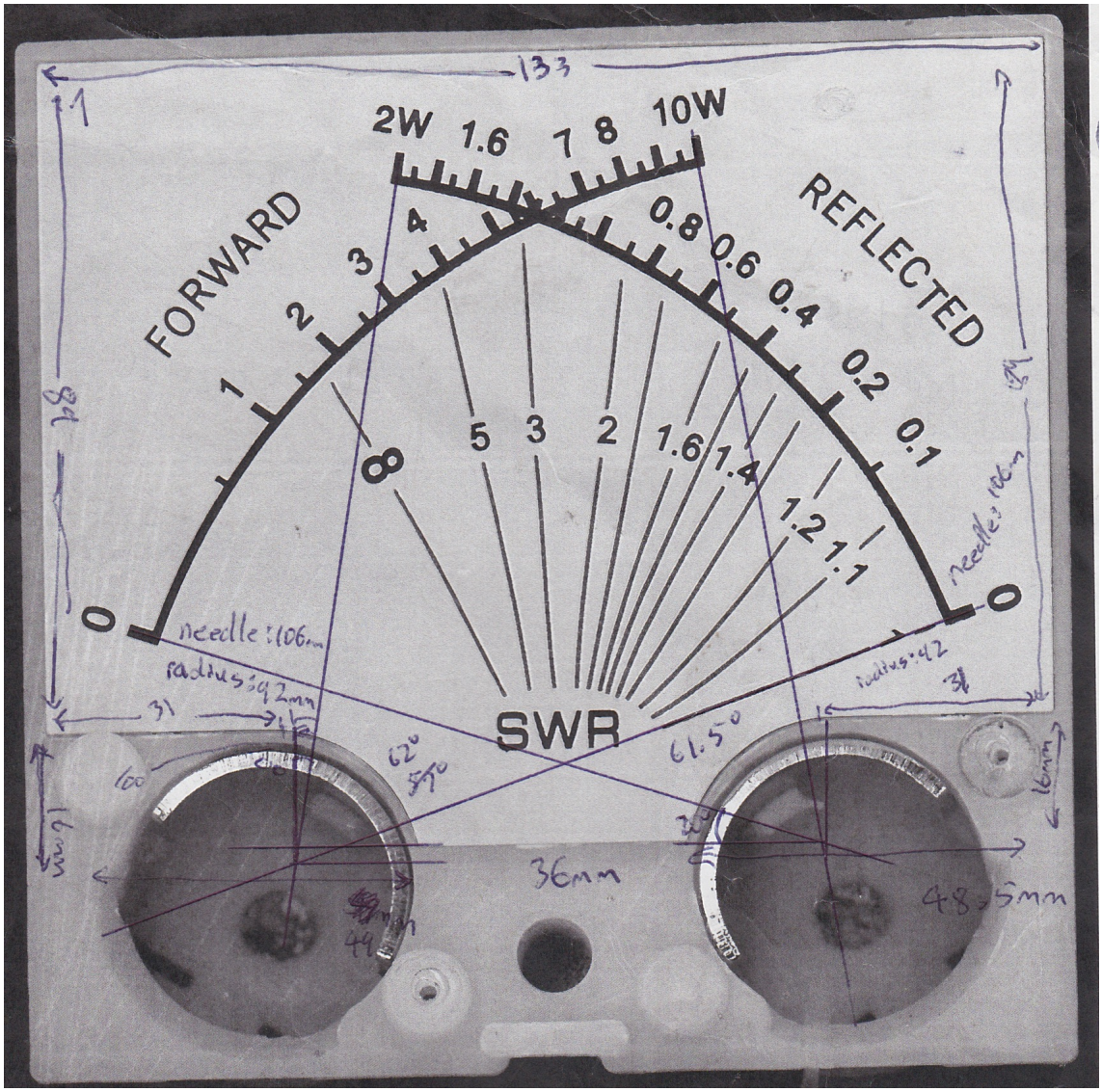


Figure 2.1: Scanned image of the SZ-70 SWR with the needles removed[1].

The position of the needle axis of rotation was estimated by drawing a straight line from the 0W scale and the Full-scale deflection (FSD) then seeing where the lines cross. The angle on which the needle rotates, was estimated by measuring the drawn straight line of the needle movement with a protractor. The two needle scales are assumed to be circular arcs drawn from 20° to 80° with a radius of 50mm. The right needle scale is the same arc except it is drawn in the clockwise direction with the axis of rotation being 38mm apart horizontally from the left needle scale.

[1]The distance between needles provided by the manufacturer was 38mm.

2.2 Contour Design

2.2.1 Intersect Calculation

Firstly, to understand and draw the custom meter scales and the contour lines, geometric analysis was conducted using the Cartesian coordinate system which allows the prediction of the needle movement and where the two needles intersect. This also allows the cross-needle meter to be simulated.

$$y = xtan\theta_L \tag{2.1}$$

$$y = -xtan\theta_R + 38tan\theta_R \tag{2.2}$$

Equation (2.1) represents the movement of the Left needle and (2.2) for the right needle. In this Cartesian coordinate system, the axis of rotation of the left needle will be the origin for simplicity. The x and y values are simply the horizontal and vertical distances from the origin and θ_L and θ_R are the angles of the corresponding left and right needles where angle θ_L is the angle of rotation in anti-clockwise and vice versa. (2.1) and (2.2) can be used to calculate the points where the needle moves. The gradient of the line varies as the needle rotates. The gradient is represented as $tan\theta$. The y intercept in (2.2), $38tan\theta_R$ also varies as the right needle rotates.

2.2.2 Contour Calculation

The angles θ_L and θ_R must be determined to use (2.1) and (2.2). In this case, the right needle will represent the state of charge of the battery in terms of percentage where 0% is flat and 100% is fully charged. The right needle angle, θ_R can be calculated from percentage battery charge, Q by using the following equation:

$$\theta_R = 160 - 0.6Q \tag{2.3}$$

The left needle angle θ_L can be calculated from charge current, I_{ch}(Amperes) using the following equation:

$$\theta_L = 6(I_{ch}) + 20 \tag{2.4}$$

However, (2.4) is for a linear scale and a set of nonlinear equations. An expanded current scale must be calculated to compare and investigate which of the non-linear expanded current scales produces a larger range of contour lines that are distributed evenly. The following set of nonlinear equations were used for comparison:

Natural Log:

$$\theta_L = 6.514ln(I_{ch}) + 65 \tag{2.5}$$

Log Base 100:

$$\theta_L = 15log_{100}(I_{ch}) + 65 \tag{2.6}$$

Square root:

$$\theta_L = \sqrt{(}360) \times \sqrt{(}I_{ch}) + 20 \tag{2.7}$$

Cube root:

$$\theta_L = \sqrt[3]{(}21600) \times \sqrt[3]{(}I_{ch}) + 20 \tag{2.8}$$

Forth root:

$$\theta_L = \sqrt[4]{(}12960000) \times \sqrt[4]{(}I_{ch}) + 20 \tag{2.9}$$

Squared:

$$\theta_L = \frac{6(I_{ch})^2}{10} + 20 \tag{2.10}$$

The charge current, I_{ch} can be calculated from charge time, t (seconds) with battery charge percentage, Q (%) for 2500mAh battery using the following equation:

$$I_{ch} = 3600t(2.5 - \frac{2.5Q}{100}) \tag{2.11}$$

Using (2.1) and (2.2) with the given angles θ_L and θ_R, and by solving simultaneous equations, the point where the two needles cross can be found. By finding the set of points where the needles intersect, the contour can be drawn for each time, t simply by joining the points where the needle cross.

2.2.3 Contour Selection

A Method was needed to efficiently process, visualize and compare the contour lines at different expanded current scales. The solution was to program a Windows Presentation Foundation (WPF) application using Visual Studio 2016 written in C# language to quickly calculate and produce the contour lines using the equations, (2.1) to (2.10). Comparing the different contours allows one to select the most easily readable and measurable contour lines.

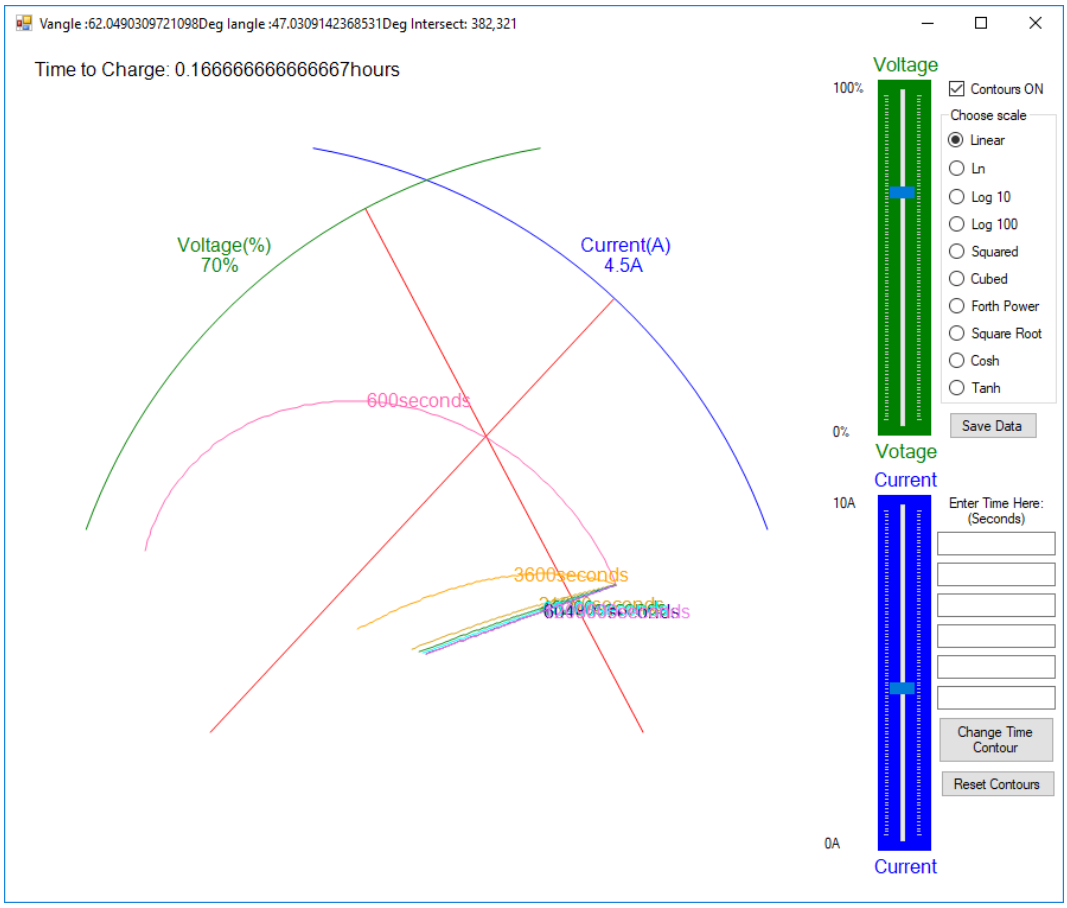


Figure 2.2: METER.exe UI displaying linear contour.

Meter.exe was written in C# with the interface shown in Figure 2.2 in order to simulate the cross needle meter with different contour lines and a selectable current scale. It can be seen in Figure 2.2 that the time calculated using 2.12 is displayed as $0.1\dot{6}$ hours which is approximately 600 seconds and it lies on the 600 second contour line. "Time to charge" label displays the amount of time left to fully charge the battery to 100% which was calculated using the

following formula:

$$t = \frac{(2.5 - \frac{2.5Q}{100})}{I} \tag{2.12}$$

Where t in (2.12) is in hours, Q is the state of charge for the battery in percentage and I is the charge current flowing to the battery in Amps.

Meter.exe has the following functions:

- The "Vangle" label at the top displays the clock wise angle of the left side charge needle, "Iangle" displays the anti-clockwise angle for the right side current needle and intersection point in Cartesian coordinates (center of left needle being the origin).

- The user can set and control the charge and current parameters by sliding the appropriate scroll bars which will also rotate and position the needle and display the values on the voltage and current label.

- The contour checkbox toggles the display of the contour lines and the scale radio buttons changes the current scale and displays the contour lines for the corresponding scale.

- The save button saves the image of the meter to a portable network graphics (png) file and the present voltage, current, time and intersection point to a text file.

- The user can choose and set up to 6 different contour time lines in seconds and display by filling the 6 empty text boxes and pressing the change time contour button and the reset contour button simple resets the contours to their default values of 10 minutes, 1 hour, 6 hours, 1 day and 1 week.

Each image with contours at different expanded current scales was saved then compared shown in Figure 2.3

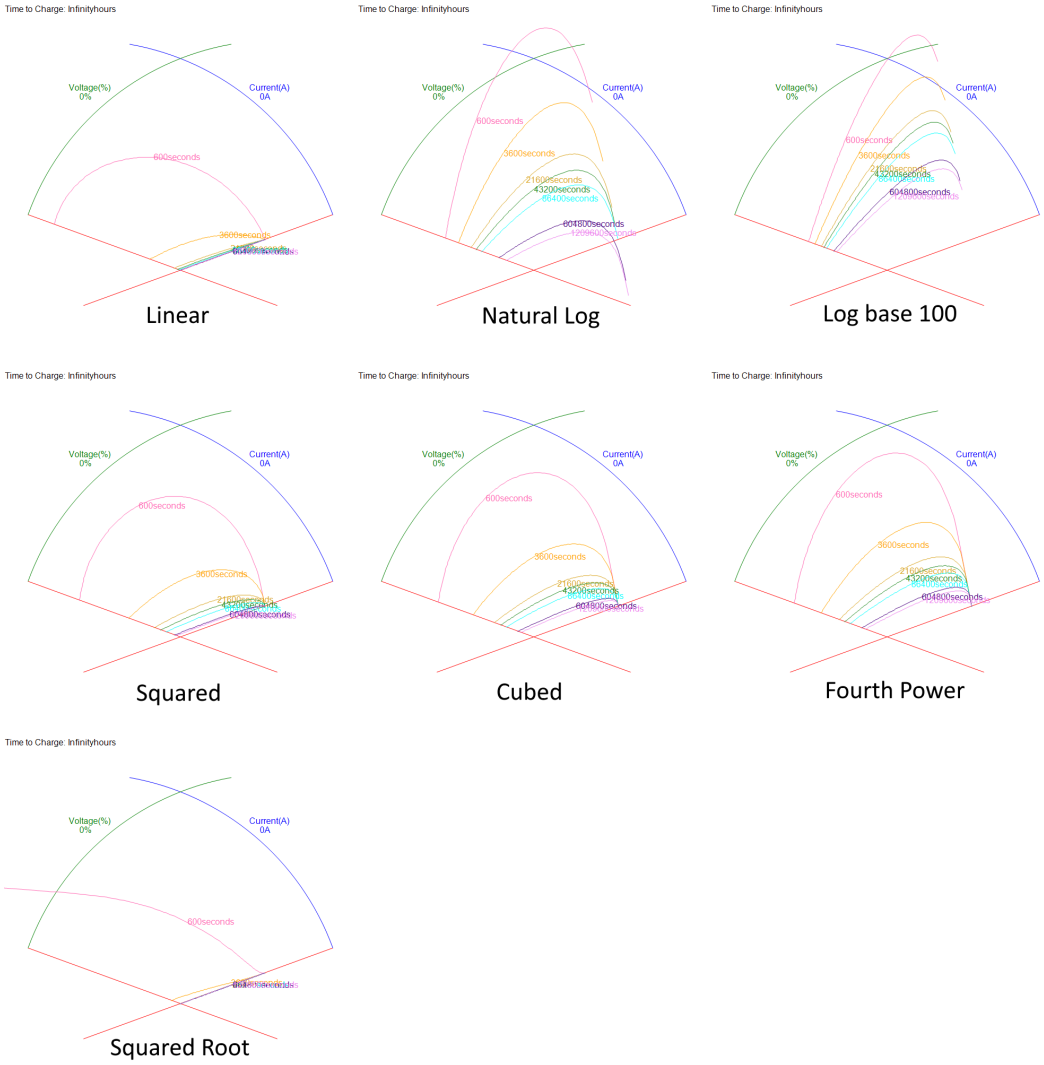


Figure 2.3: Comparison of the contour lines with a different current scale.

It is found that some of the current scales such as the linear scale, the contour lines for 6 hours and above are clustered together and difficult to read.

By studying Figure 2.3 where the charge time contour lines range from 10 minutes to 2 weeks, the contour lines with the logarithmic current scale had the most evenly spread, easy to interpret and practical. Therefore a natural log current scale was chosen for the product.

2.3 Panel Design

The drawings were done using SolidWorks 2016 SP0 from the dimensions taken from section 2.1.1. SolidWorks was capable of producing an accurate 1:1 drawing that can be printed and applied to the cross-needle meter.

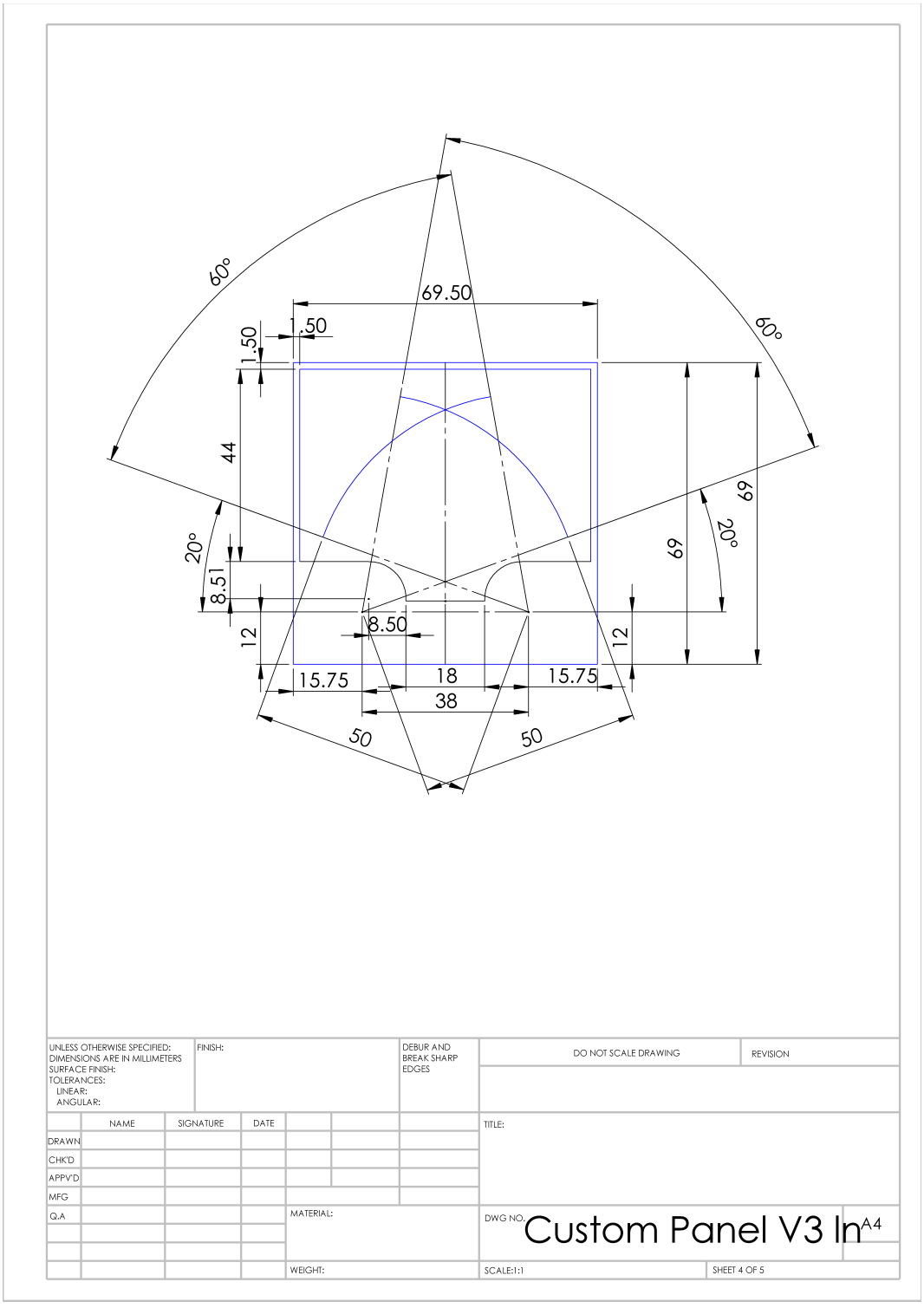


Figure 2.4: Dimensions of the front panel of the meter.

The dimensions of the needle pins were needed for the custom foot prints for the PCB to mount on the back of the panel.

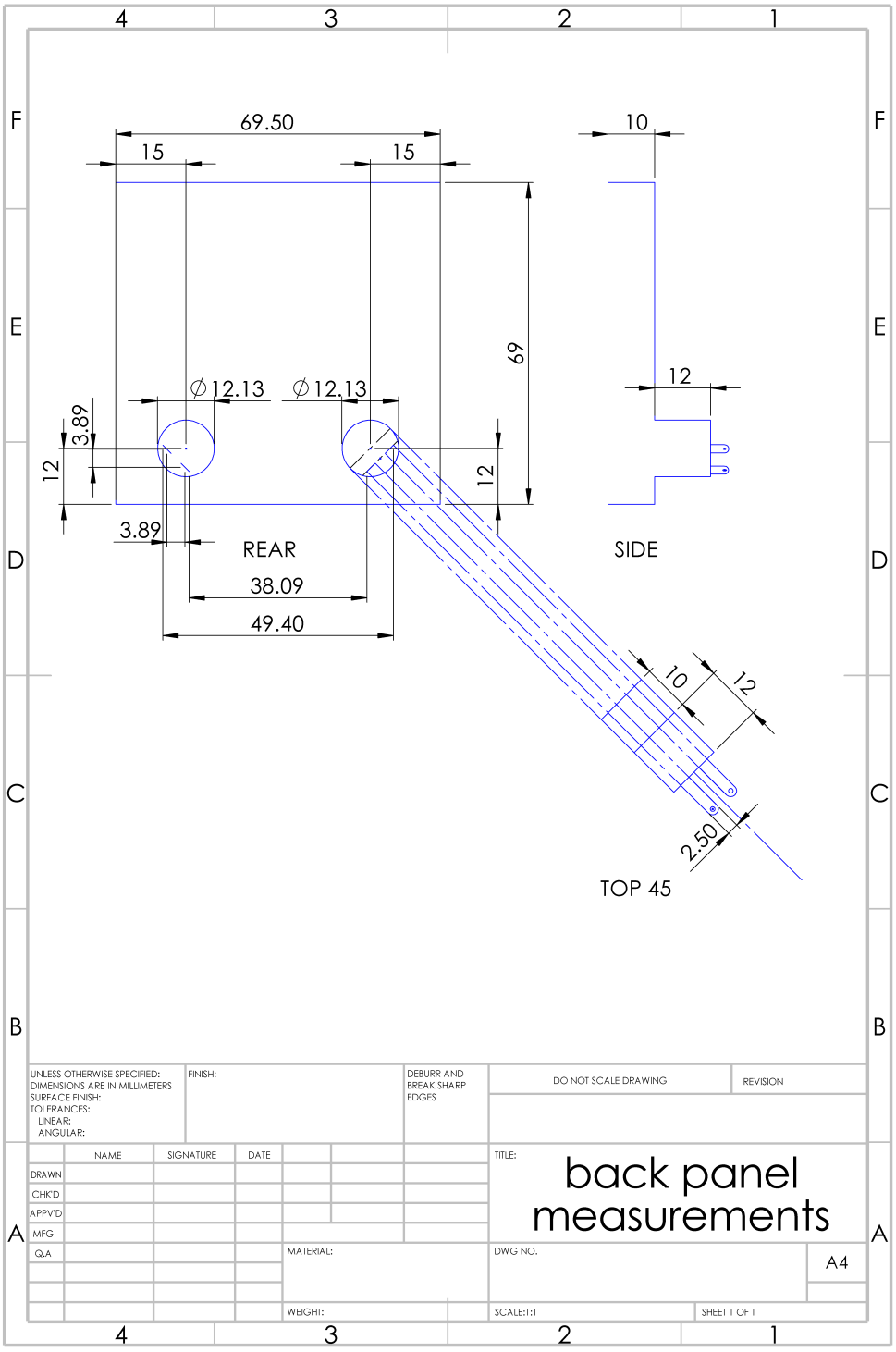


Figure 2.5: Dimensions of the back panel of the meter.

The calculations done in the WPF application in section 2.2.3 are measured in pixels and the output image of the contour lines can not be directly applied

into the SolidWorks drawing therefore a API macro was required to automate the contour lines drawings in mm were needed.

Fortunately in SolidWorks 2016 API is compatible with C# and the API can be directly used in Visual Studio. The code to calculate the points and to join the contour lines in the application section 2.2.3 were reused and drawn as a spline in SolidWorks.

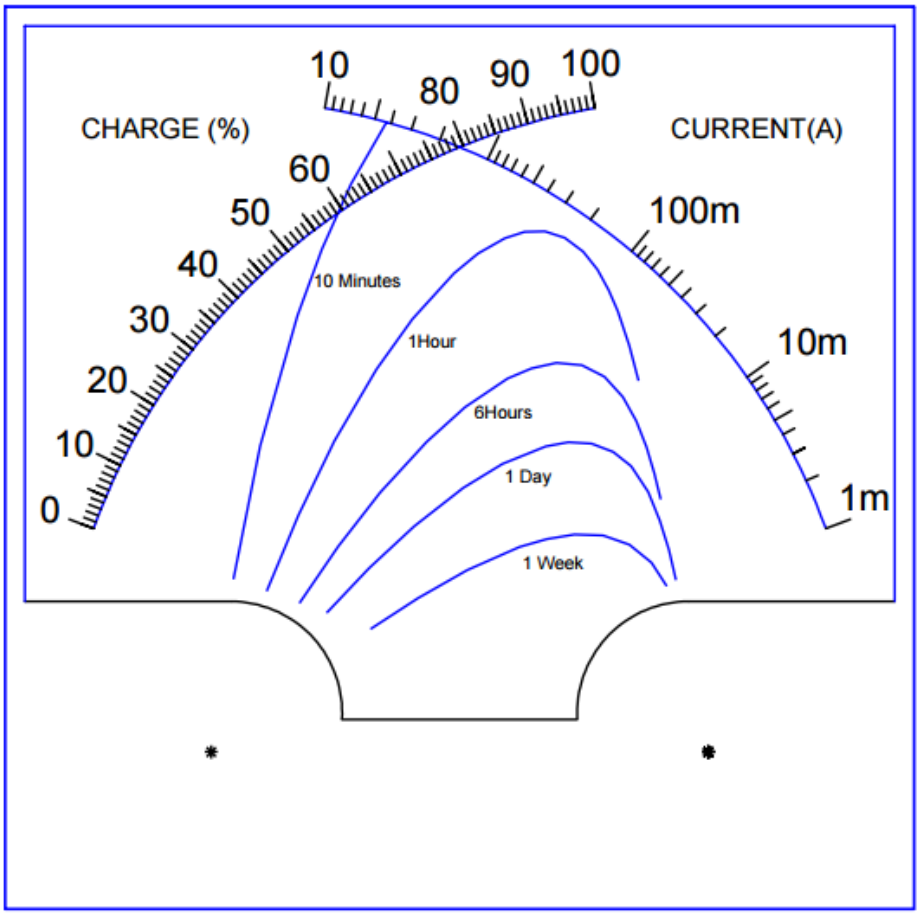


Figure 2.6: Final Drawing of the custom meter panel with a logarithmic current scale and contour lines[3].

[3]Not to scale. Values for 70% charge and 1A are removed as it would be difficult to read with too many numbers cluttered in one place

Chapter 3

Hardware Measurements

3.1 Overview

This chapter discusses the methods, instruments and the results for measuring the hardware in order to understand and compare or confirm the properties before assembly of the final product.

3.2 Modeling cell state of charge

The 18650 Lithium-Ion single cell rechargeable battery is commonly used in laptop computer batteries and some of the Tesla electric car models. 18650 refers to the external dimensions of the battery, 18mm diameter and 65mm in height.

The charge and discharge tests of the 18650 Lithium-Ion cell were conducted using the Agilient 5270B precision IV analyzer to supply 60s 0.26A current pulses while measuring open circuit and closed circuit voltage with the Agilient 34401A digital multi-meter(DMM). The purpose of this experiment is to estimate the internal impedance and determine the state of charge and terminal voltage relationship. The following figures shows the plot of the results at room temperatures (23° C - 25° C).

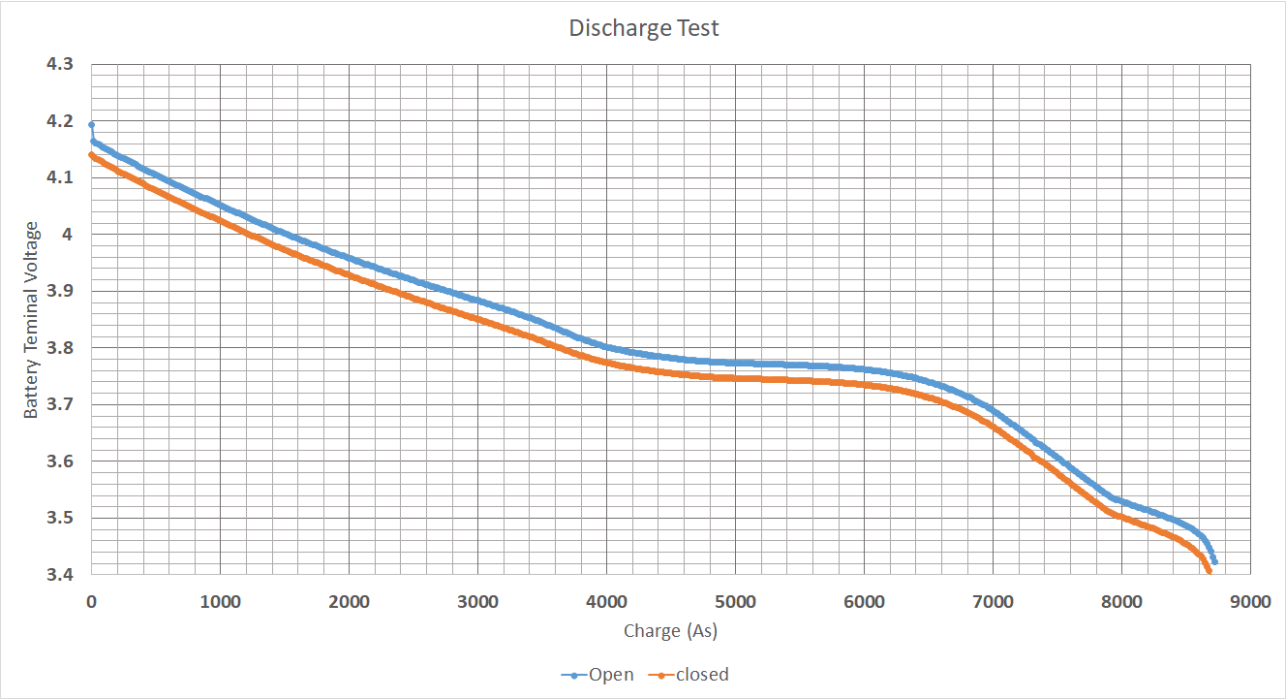


Figure 3.1: 18650 discharge test

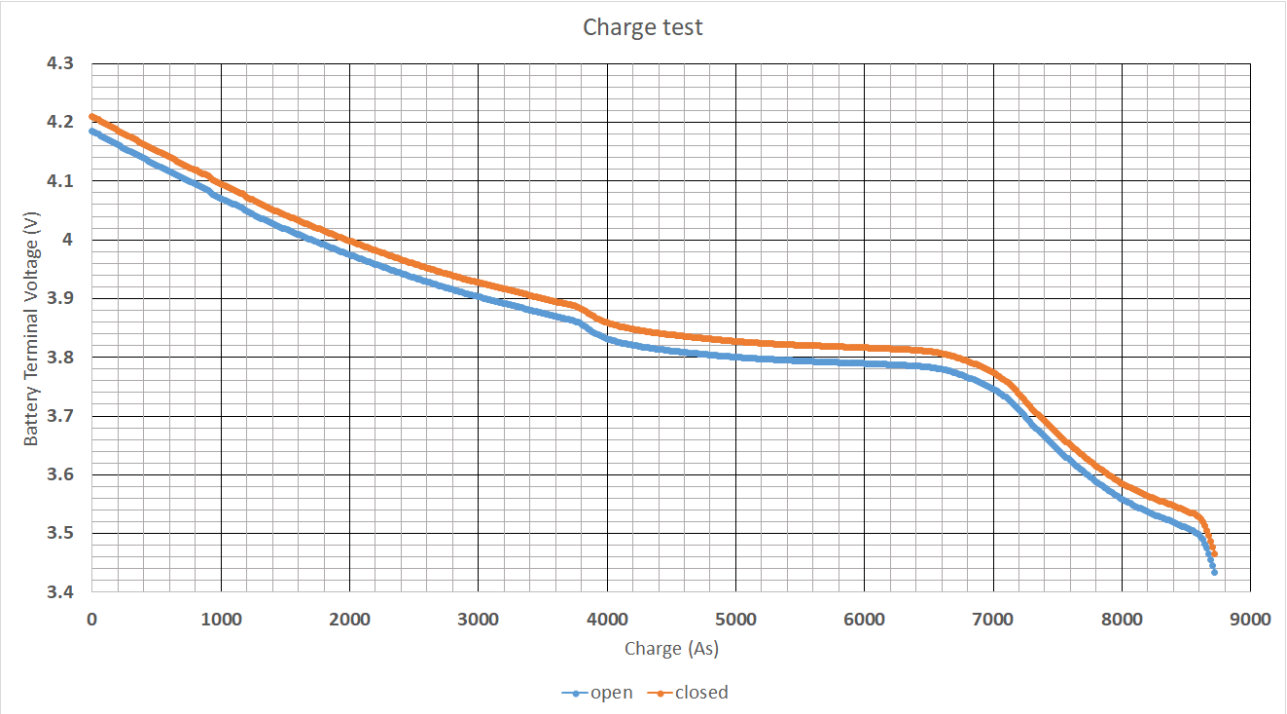


Figure 3.2: 18650 charge test

Figure 3.1 and Figure 3.2 show almost an identical trend with alternated open circuit (blue line) and closed circuit (orange line). Cell voltages at about 3.4V can assumed to be flat. It can be seen in the charge test in Figure 3.2

that cell reaches full capacity between voltages of 4.18V to 4.2V. The internal impedance ranged between 176mΩ to about 103mΩ for the discharge test and 122mΩ to about 90mΩ for the discharge test. It can be seen in both plots that voltage falls at a steep incline between 0As to 4000-4500As then falls at steadily between 4500As to 6500As then rapidly declines below 6600-7000As.

3.3 Band gap reference voltage

The LM285 band gap reference was tested with a DMM and a bench top power supply. The voltage was changed from 3.0V to 4.2V and the reference voltage stayed consistent at 2.500V. Hence this device provides a stable and accurate reference voltage which is used to calculate supply voltage in chapter 5.

3.4 Measuring Cross-Needle meter FSD

The Cross-Needle meter is simply an SWR analog meter with two galvanometers which deflects at opposite angles with a range about 60°. A small current is drawn from the input terminals through a suspended moving coil surrounded by two permanent magnets and produces a Lorentz force to deflect the needle pointer. The upper and lower balance springs restores the zero-position of the needle when no current is flowing through the coil.

560Ω resistor and 39KΩ resistors were soldered in series with the input terminals of the meter then it was tested by connecting a GFG-8015G function generator and an TBS1052-EDU oscilloscope to the input terminals. A square wave PWM with 250KHz was produced from the function generator to simulate a PIC PWM with duty cycles tuned from 10% to 90%. It was seen on the meter that it is almost linear with a 2% error from 10% to 40% and flawless linearity was observed from 50% to 90%. It is proven that this cross-needle meter can be deflected linearly using PWM.

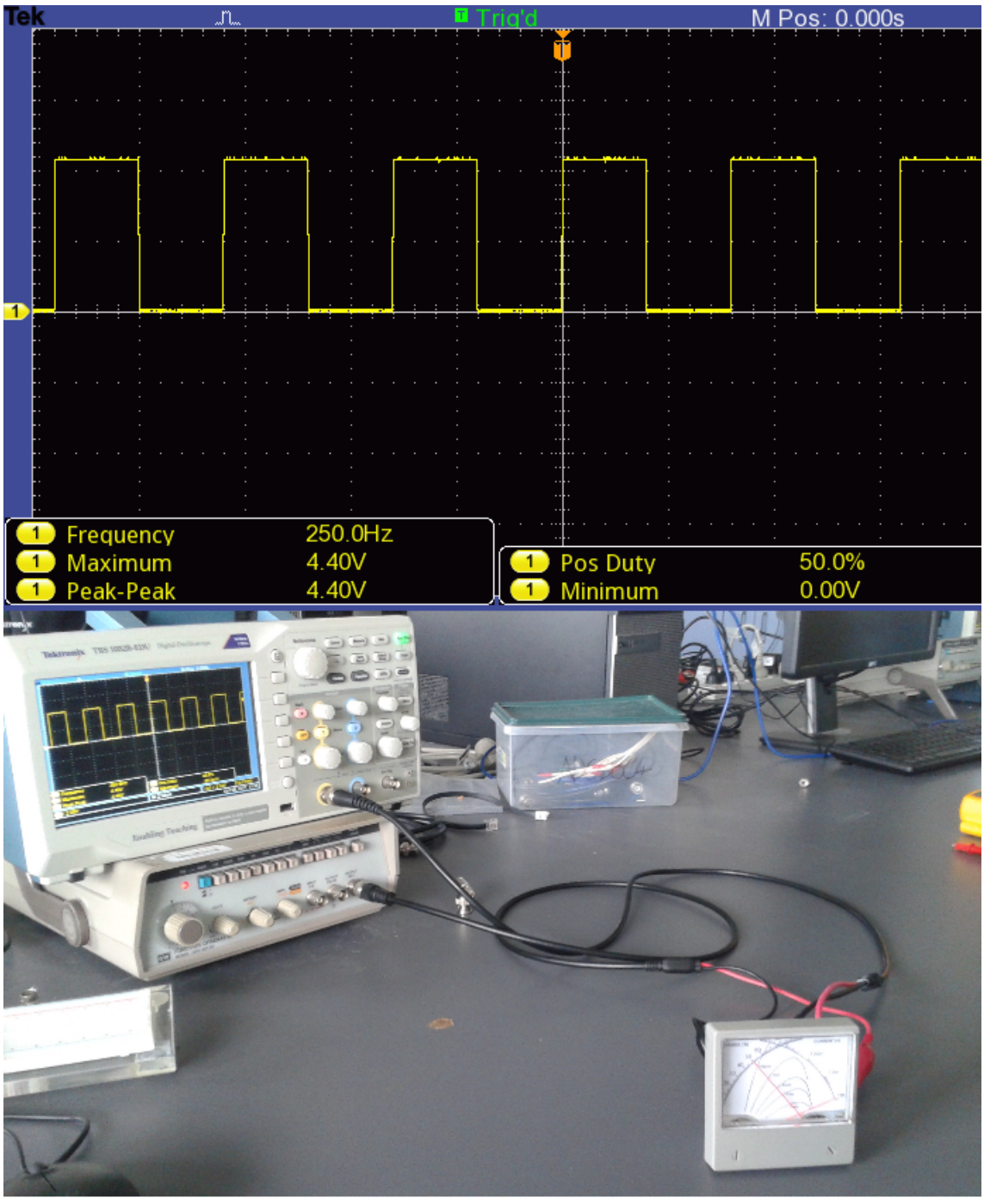


Figure 3.3: Cross-Needle meter connected to the function generator

The results also confirmed that the each meter requires 100μA for a full scale deflection (FSD) and has a total resistance of 1.54KΩ measured with a DMM. The measured values agree with the specifications which the manufacturer had provided.

3.5 Modeling Schottkey diode characteristics

The forward voltage and current characteristics needed to be modeled at different ambient temperatures for temperature correcting the log of current. Ideally the cross-needle meter will work in all sorts of climates. The Voltage across the Schottkey diode was measured with the 34401A DMM while placed into the incubator with various ambient temperatures. Current set from 0.5mA to 1A with 50 steps and supplied by the 5270B IV analyzer.

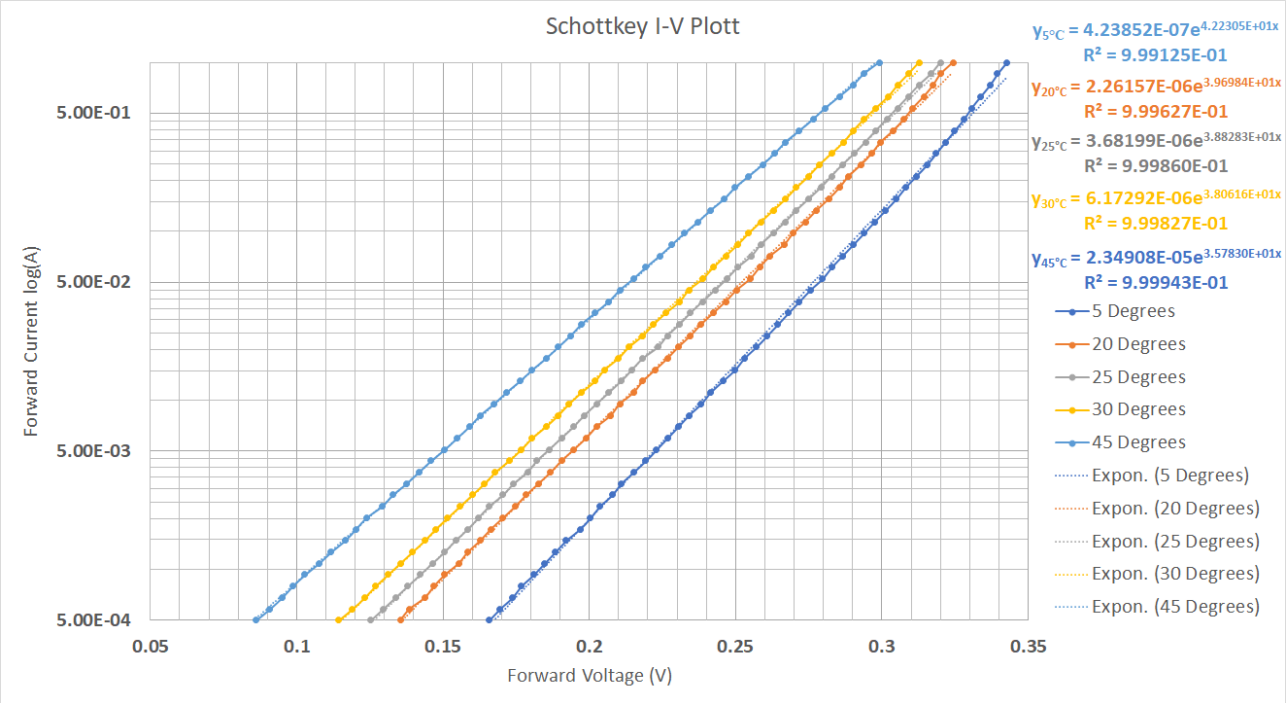


Figure 3.4: Schottkey diode IV characteristics at different ambient temperatures

The plots in Figure 3.4 are in different colours to indicate readings at a different ambient temperatures: 45°C in light blue, 30°C in yellow, 25°C in grey, 20°C in orange and 5°C in blue. The Schottkey diode current is displayed in a logarithmic scale to form a linear graph and the Schottkey diode voltage on the x axis remains in a linear scale. Trend line equations were used to approximate saturation current, thermal voltage and n factor and these values were used in the firmware for calculation.

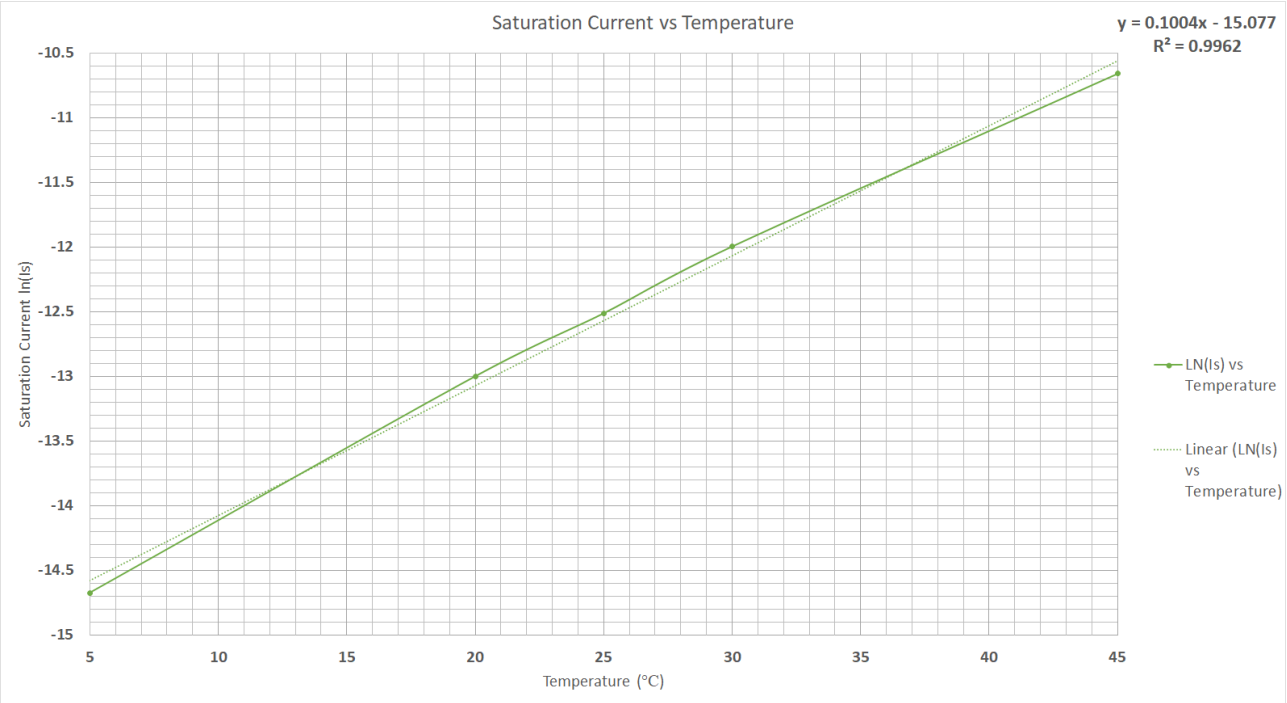


Figure 3.5: Plot of the log of saturation current against ambient temperature

The natural log of the saturation currents at different ambient temperatures were taken from the y intercepts of the trend line equations from Figure 3.4 and was plotted against the ambient temperature. This graph forms a linear plot and from the trend line equation, the log of saturation current can be estimated as a function of temperature and the gradient and y intercept were rounded to form the following equation:

$$ln(Is) = \frac{T_{ambient}}{10} - 15 \tag{3.1}$$

Equation (3.1) was used in the firmware to calculate the log of saturation current.

As there is a 1A output limit for the IV analyzer, a separate test was conducted using a TRIO PD35-20 power supply capable of delivering up to 20A. This separate test was done similarly to the previous test with the diode placed inside the incubator. Current was measured with a DMM in series and the voltage measured in parallel.

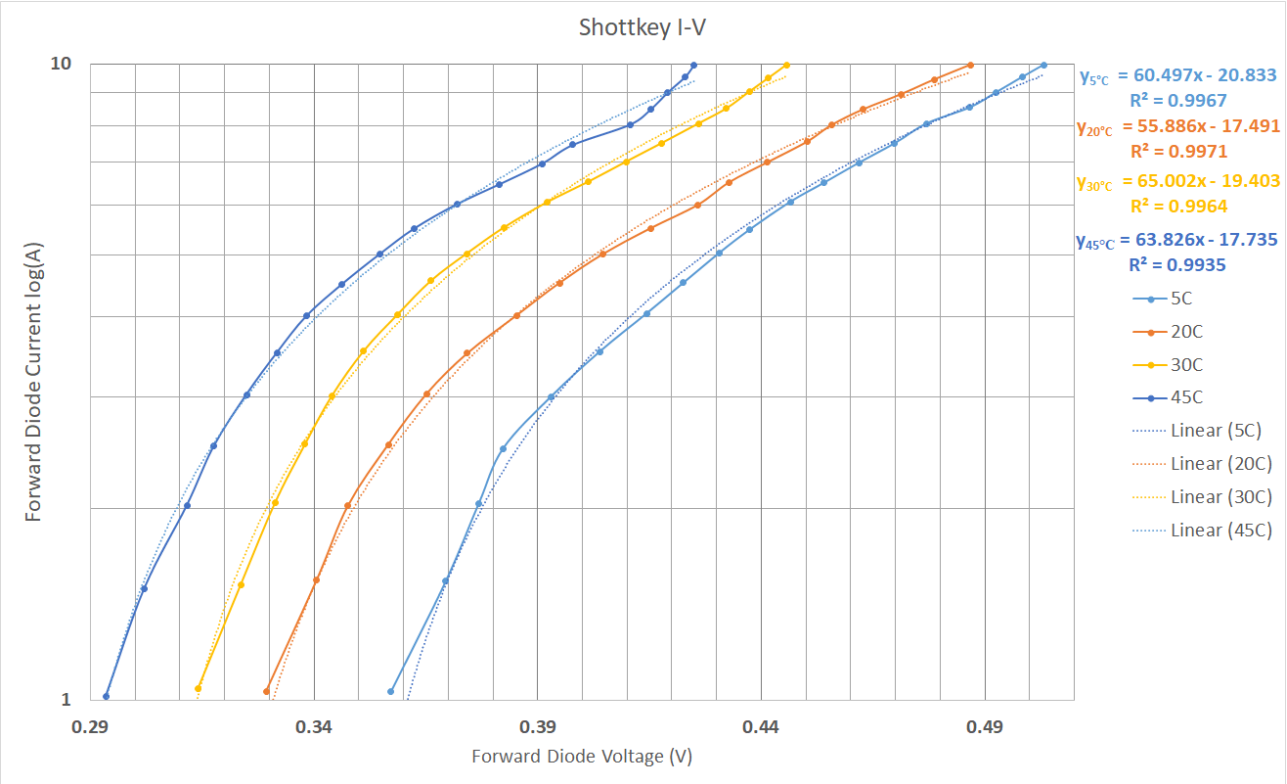


Figure 3.6: Schottkey diode IV characteristics at different ambient temperatures between 1A to 10A

It can be seen in Figure 3.6 that at high current the diode no longer follows the conventional I-V characteristic exponential function. This is due to the junction temperature rising and dissipating heat. The Schottkey diode begins show Ohmic characteristics.

3.6 Measuring temperature sensor voltage

The temperature sensor voltage was measured inside the incubator voltage of 0.4V at 0°C and 0.84V at 25°C as opposed to the 0.5V at 0°C and 0.75V at 25°C which the data sheet suggests. Checking the sensor enabled a greater accuracy of reading the temperature.

Each temperature sensor seems to give a slightly different voltage and therefore each temperatures sensor has different gradient, $V_{temp}/T(°C)$. This was tested by attaching a temperature probe to the temperature sensor and measuring the output voltage while raising the board temperature with a hot air

gun. The linear equations from the following figure was to used to formulate the relationship between the PCB temperature and the sensor voltage.

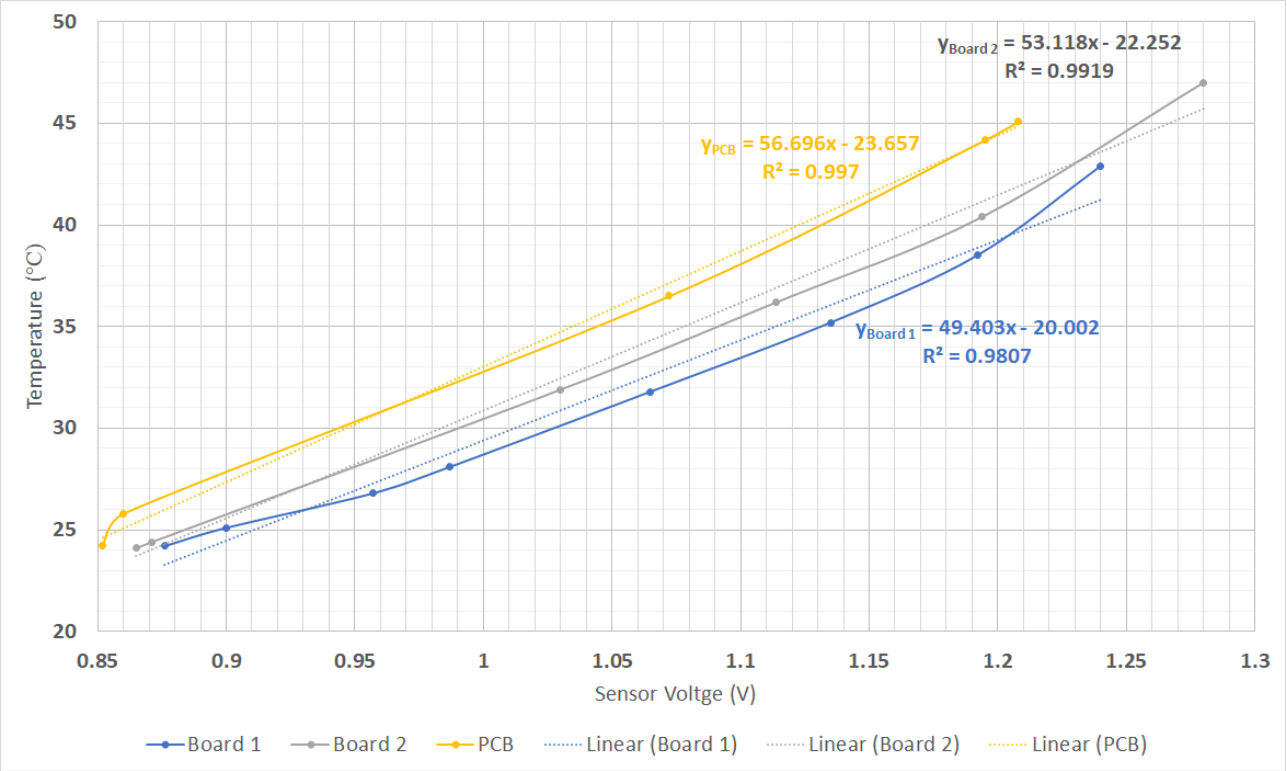


Figure 3.7: Plot of different temperature sensors being tested

Chapter 4

PCB design

4.1 Overview

The circuit is designed to mount directly onto the back of the meter. The circuit design is further developed from [1] with features such as using a LM285 2.5V band gap reference and using a Shottkey diode shunt to charge the battery, and driving an analog meter using PWM. A PIC16F684 micro-controller is used instead of 12F683 which has 14 pins (capable of 8channel 10bit resolution ADC) instead of 8 pins which is necessary to add more features.

4.2 LT Spice Simulation

The simulation was done using LTspice XVII in order to predict and reference the characteristics of the opamp that amplifies the voltage across the P-N junction of the diode while connected to the battery. Changing the battery voltage results in change in opamp offset voltage. The purpose of this simulation is to lower the the opamp gain as much as possible to prevent saturation.

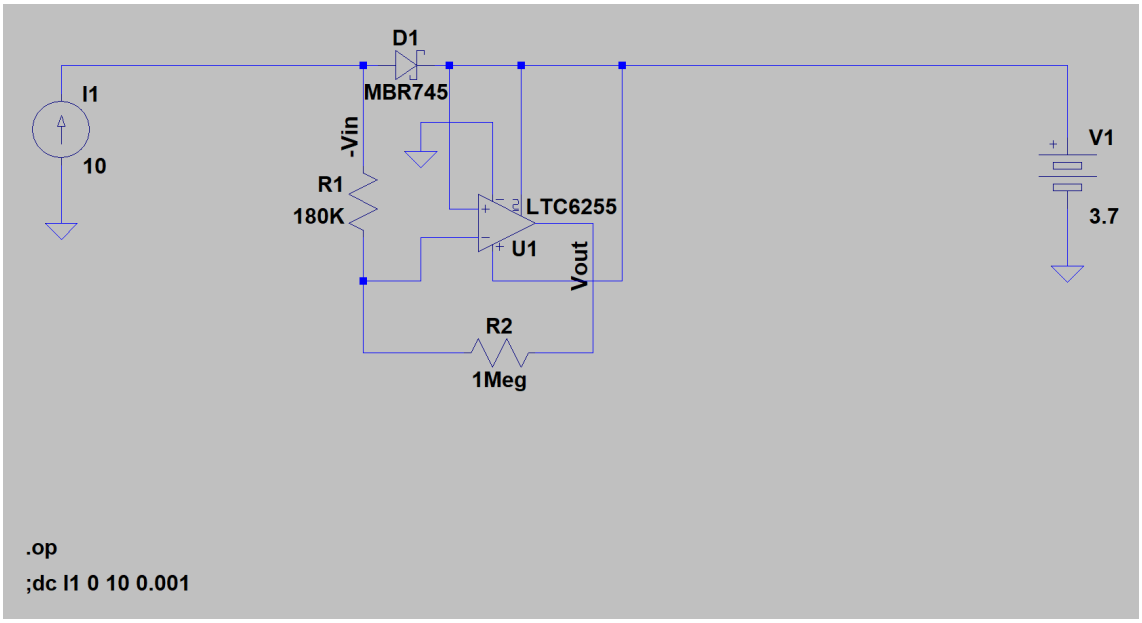


Figure 4.1: LTspice simulation circuit diagram[1].

Multiple simulations were done by doing a DC sweep of the current source and varying the gain of the opamp from 10, $8.\dot{3}$ and $5.\dot{5}$ which is done by altering the value of the resistor, R1.

[1]The simulation was done using ideal components and does not take into account parameters such as ambient temperature change, heat dissipation, internal capacitance and does not simulate the charging of the battery.

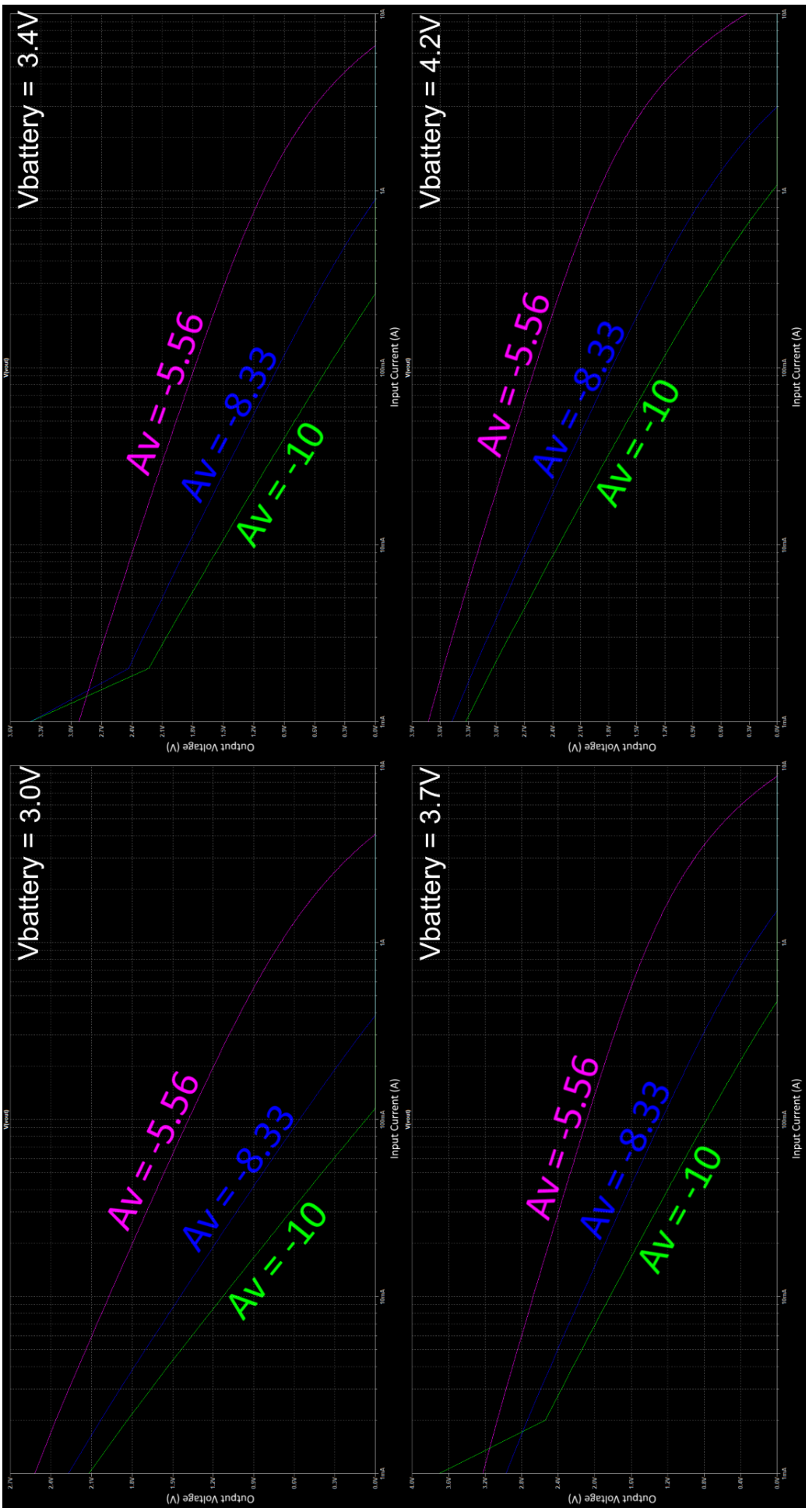


Figure 4.2: Simulation results of DC current sweep.

Figure 4.2 compares the results of different opamp gains with various supply battery voltages. The x axis is the forward current of the diode, which is swept from 0A to 10A with 1mA steps. The y axis is the op-amp output voltage. The Pink line is when the gain Av = -5.$\dot{5}$ and input resistor R1 = 180KΩ. The Blue line is when Av = -8.$\dot{3}$ and R1 = 120KΩ. The Green line is when Av = -10 and R1 = 100KΩ. It can be seen that by reducing the gain, saturation occurs at a much higher current and therefore the resistor value for R1 was chosen to be 180KΩ.

4.3 Circuit Design

This section describes the overall function of the circuit. The schematic is shown in the following figure:

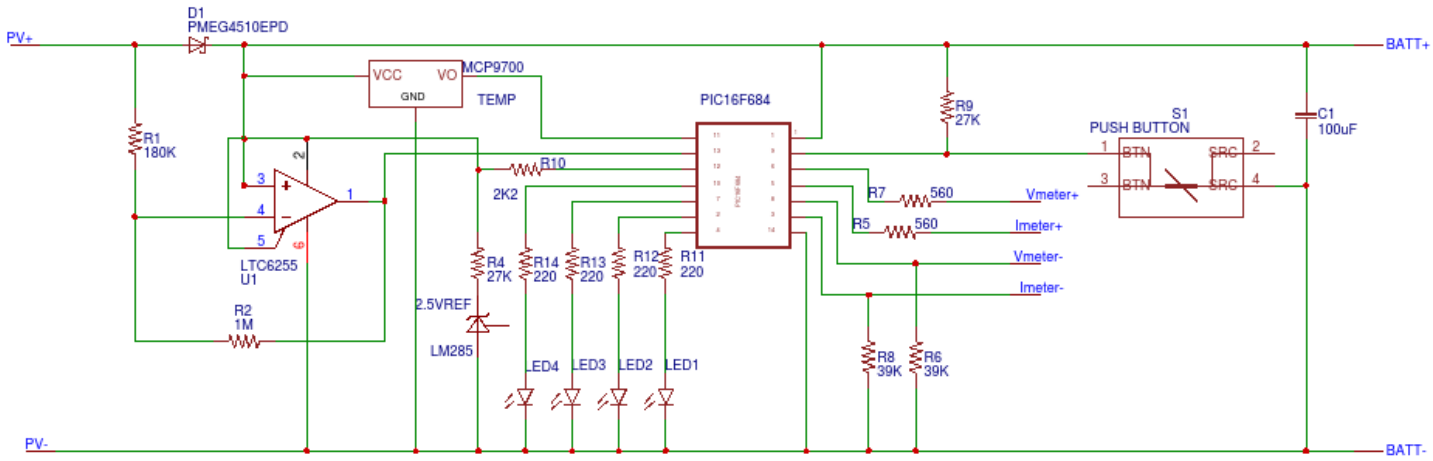


Figure 4.3: Circuit diagram of the Cross-Needle meter.

Looking at Figure 4.3, the PMEG4510 Schottkey diode with a low forward voltage acts as a shunt to allow current flow from a input power source to charge the battery. The forward voltage across the Schottkey diode increases logarithmically with current. The voltage is measured across the anode and cathode of the Schottkey diode. Diode voltage is then amplified by the opamp and the output is connected to an ADC pin of the micro-controller in order

calculate the charge current. The LTC6255 opamp has a low supply voltage ranging between 1.8V to 5.5V, which ideal for this circuit because it is powered by the 18650 Li-ion single cell battery. The opamp gain is set to $5.\dot{5}$ by R1 and R2 (discussed in section 4.2). The MCP9700 temperature sensor outputs a set voltage regardless of varying supply voltages. The temperature sensor is placed close to the Schottkey as the IV characteristics of the Schottkey diode vary significantly with ambient temperature, change in temperature is constantly monitored. Two independent PWM drive the two needles of the dual analog meter with one needle being driven by the built in CCP1 module and the other needle being driven through timer 0 software PWM (discussed in chapter 5 section 5.4). The two negative terminals of the dual needle meter are pulled down to the ground and the current through each meters are sensed by two ADC channels of the PIC. The LM285 is a reliable band gap 2.5V reference that works to as low as 3V (testing discussed in chapter 3 section 3.3) which is suitable for this low power application. The four LEDs indicate a different status such as complete charge, overcharge, flat battery etc. The push button is simply a reset button. The battery terminals are connected to a protection circuit which shuts down when overcharging occurs. The circuit was tested between voltages from 3V to 4V and only drew current from about 1.32mA to 1.6mA

4.4 PCB Manufacturing

The PCB design was done using Eagle CAD version 7.70, the Gerber files were reviewed using GC-Prevue 25.2.8 then was sent to DFRobot for manufacturing.

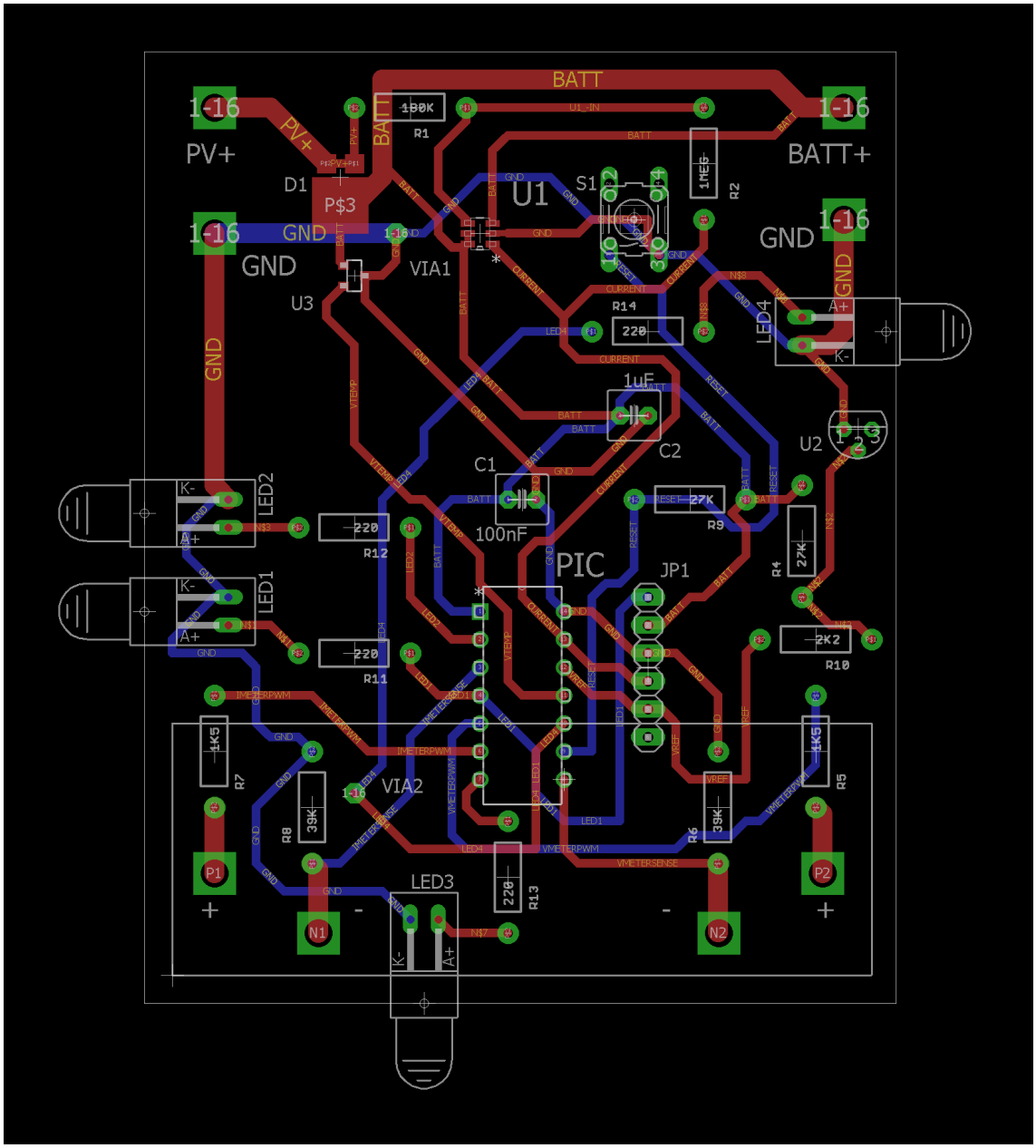


Figure 4.4: Two layer PCB layout of the circuit.

The overall dimension of the PCB was 68.25mm wide and 86.35mm long which exceeds the height of the Meter slightly when mounted. The top layer copper is in red, bottom layer in blue, pads are in green and silkscreen are in white. The tracks are 30mil thick which is the minimum thickness that the manufacturer can provide and the ground, power and meter output tracks are 70mil to provide a strong enough connection to the larger pads to the battery, ground, input power and meter. Most of the components are through-hole as this PCB is a prototype and only the Schottkey diode, opamp and

temperature are surface-mount devices (SMDs) as there were no through hole parts available.

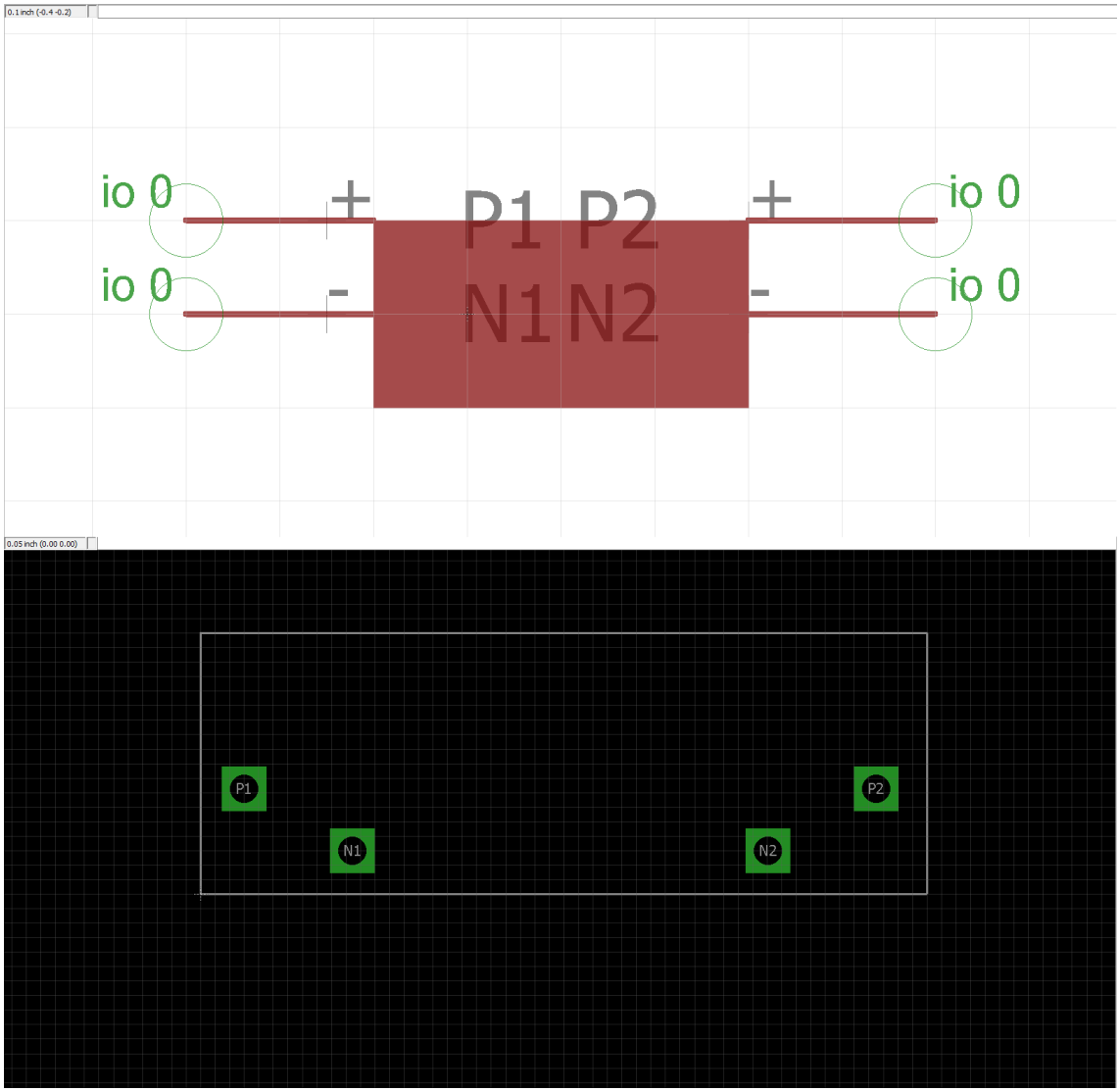


Figure 4.5: Custom symbol and footprint for the SZ-70 meter back mounting.

Dimensions from Figure 2.5 (positive meter pins being 49.4mm apart horizontally, negative pins being 38.09mm apart and 3.89mm between the positive and negative pins vertically) were used to create a custom foot print with larger pads to provide enough strength to prevent the pads from coming off. Each grid in the footprint (bottom half of Figure 4.5) are 0.5 inches long. Pin P1 and P2 connect to the positive pins of the meter and pins N1 and N2 to the negative. The pads are 0.15 $\times$ 0.15 inches and have a drill size of 0.1 inch (2.54mm) which is wide enough for the 2.5mm wide meter pins.

Figure 4.6: PCB mounted onto the back of the panel.

In Figure 4.6 it can be seen that the Pads perfectly align to the back of the meter therefore can be easily be mounted.

Chapter 5

Firmware

This chapter discusses the functions in the firmware, the limitations and how the firmware is devised.

The Firmware has the following functions:

- Measures 2.5V band gap voltages as reference
- Measures and converts temperature sensor voltage to degrees Celsius to obtain board temperature
- Calculates the saturation current of the Schottkey diode from ambient temperature
- Estimates input battery charge current from Schottkey shunt voltage
- Uses 8 bit Timer 0 software PWM signal to drive charge meter and 8 bit single output Enhanced PWM to drive the current meter
- Calculates required PWM duty cycle to drive the regardless of varying battery voltages
- Green LED turns on to indicate battery is fully charged
- Red LED turns on to indicate dead battery

The following figures show the flow charts for the overall firmware process and they are separated into main, timer 0 interrupt service routine (ISR) and timer 2 ISR.

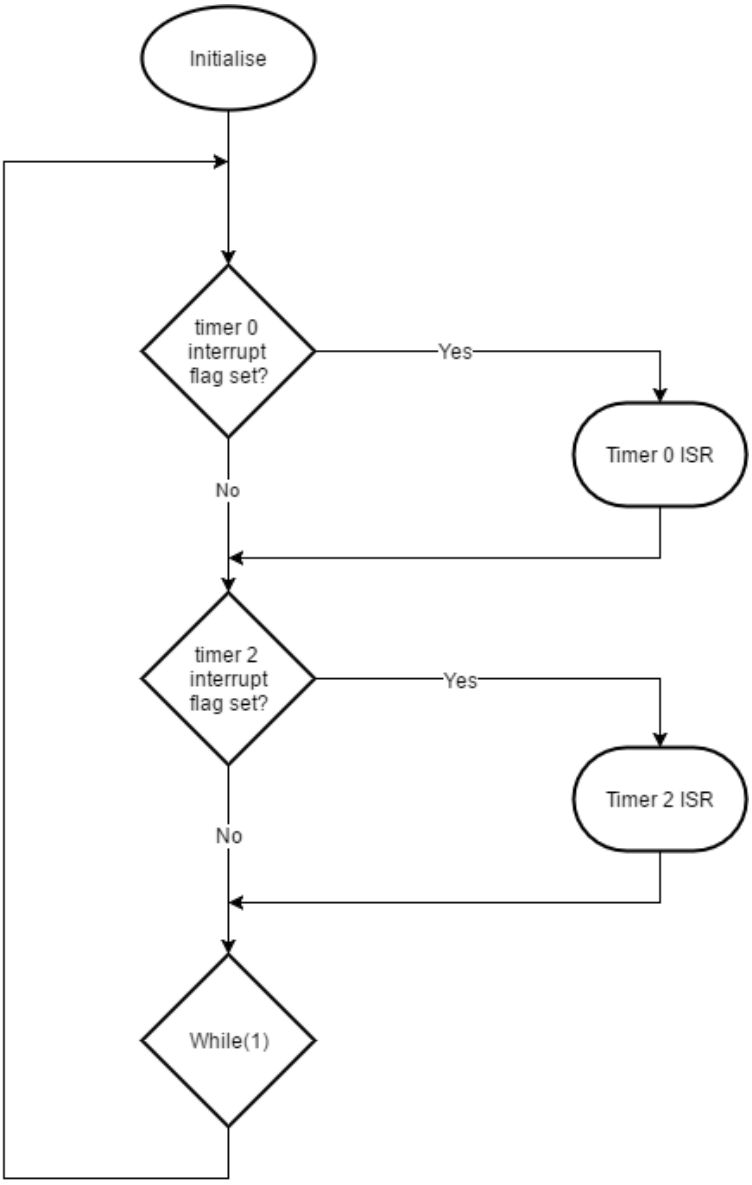


Figure 5.1: Flow chart of the main loop

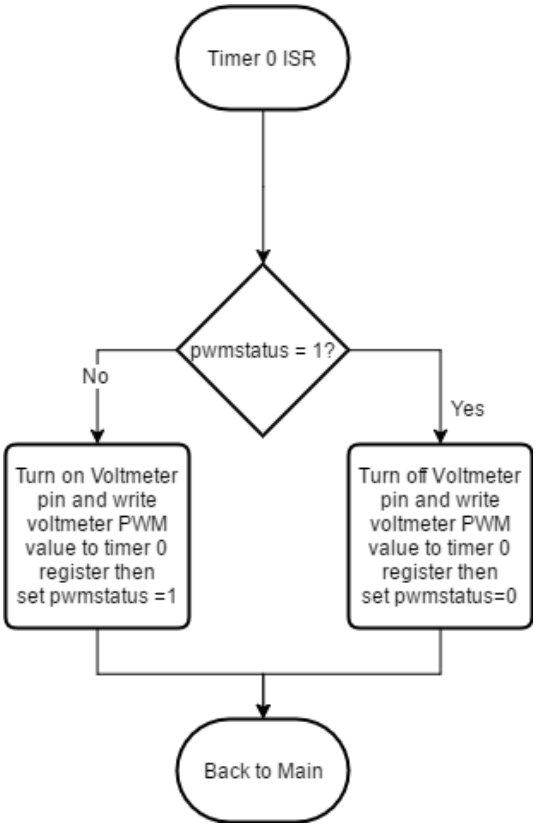


Figure 5.2: Flow chart of timer 0

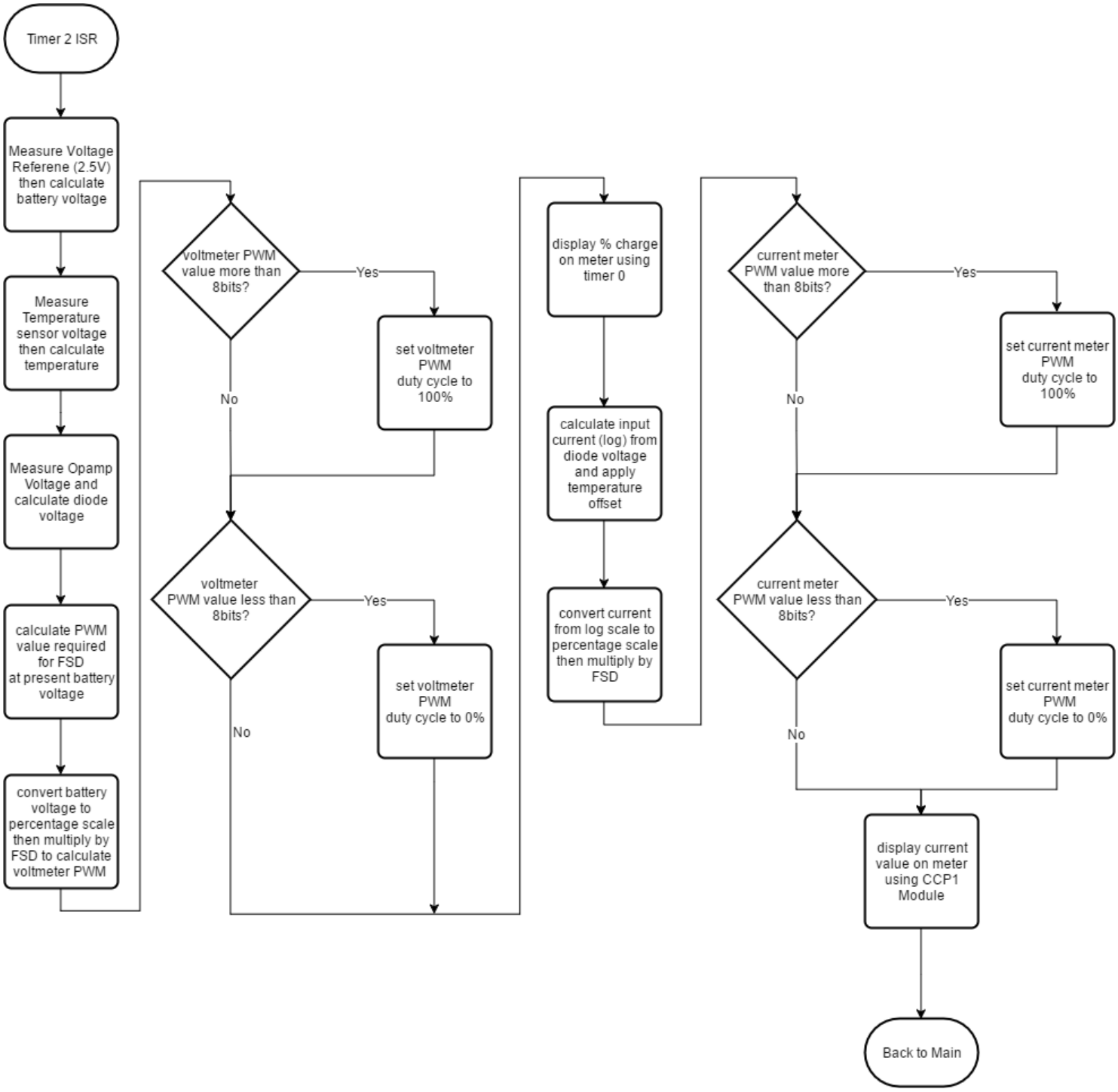


Figure 5.3: Flow chart of timer 2

Timer 2 prescaler and postscaler are set to 1:16 so the timer 2 ISR occurs every 65.3ms. This gives enough time to perform all the necessary measurements and calculations which are discussed in the upcoming sections.

69.7% program and 50.8% of the data memory was used. Compiled using Hi-Tech C standard version 9.8

5.1 Measuring Battery Voltage

The band gap voltage reference gives a constant known voltage of 2.5V and the PIC reads ADC in 10 bits hence the V_{DD} can be calculated using the following formula:

$$V_{\mathrm{DD}} = \frac{1024}{\mathrm{Value}_{\mathrm{ADC}}} \times 2.5V \tag{5.1}$$

5.2 Measuring Diode Voltage

The log of charge current can be a function of temperature using the linearised formula:

$$ln(I_d) = ln(I_s) + \frac{V_d}{V_{TH}} \tag{5.2}$$

Where $ln(I_s)$ is log of the saturation current calculated by using the linearised formula from Figure 3.5. The thermal Voltage, V_{TH} which determines the gradient in (5.2) is also calculated as a function of temperature:

$$V_{TH} = \frac{nKT}{q_{electron}} \tag{5.3}$$

Where K is the Boltzman constant $1.38064852 \times 10^{-23} m^2 kg s^{-2} K^{-1}$, q is the electrical charge of an electron $1.60217662 \times 10^{-19}$C and T is the temperature in Kelvins. The ideality factor, n in (5.3) is assumed to be 1. Changes in n with temperature is neglected as it is found by analysing the Schottkey diode I-V characteristic plot in Figure 3.4 that changes with temperature are very small ranging from -1.2% to +1.9% of n.

5.3 Measuring Temperature

As seen in Figure (3.7) that the temperature sensor has a slight variance in voltage for each board and code has to be written to adjust to this variance. In the board tested the equation to determine temperature was:

$$T_{(^\circ C)} = 57 \times V_{Temp} - 23 \tag{5.4}$$

5.4 Outputting two independent PWM signals

The PIC16F684 micro-controller only has a single PWM module and a second independent PWM is required to drive to two needles. The solution to having a second PWM module is to use a software PWM using a timer module. This is done by setting the desired output pin high for the duration of the assigned duty cycle which is simply the ratio between how long the pin will be high for and how long the pin will be low. An 8-bit timer, Timer 0 was chosen to output the software PWM as the CCP1 module is also set with 8-bits and the duty cycle values can be interchangeable which is useful for testing purposes. The flowchart in Figure 5.2 shows that the timer 0 counts until the assigned duty cycle value, then the timer reloads. The bit variable *pwmstatus* alternates every time 0 reloads and an if statement checks the *pwmstatus*. When the *pwmstatus* bit is equal to 0, the voltmeter pin is set high for the duration and the duty cycle value and the timer counts until this duty cycle value then overflows. When *pwmstatus* is equal to 1, the voltmeter pin is set low for the duration of 255 - duty cycle value. Currently for this application, using the timer 0 PWM performs well enough, however there are flaws such as not being able to set the timer 0 prescaler too low as there will not be enough time for the ISR to execute and because of this, the PWM has a low frequency of 15Hz which causes the needle to vibrate. Another flaw is that the timer 0 always sets voltmeter pin high for a very short period that can be seen on the oscilloscope even if the duty cycle is set to 0. This does not have influence on the voltmeter needle however, and no needle movement was observed.

Chapter 6

Results

6.1 Overview

This chapter discusses the methods for measuring accuracy and performance of the Cross-Needle meter and how the environment conditions were set up.

6.2 Voltage meter

The voltmeter accuracy is shown in the following figure which compares the actual DMM reading to the voltmeter reading:

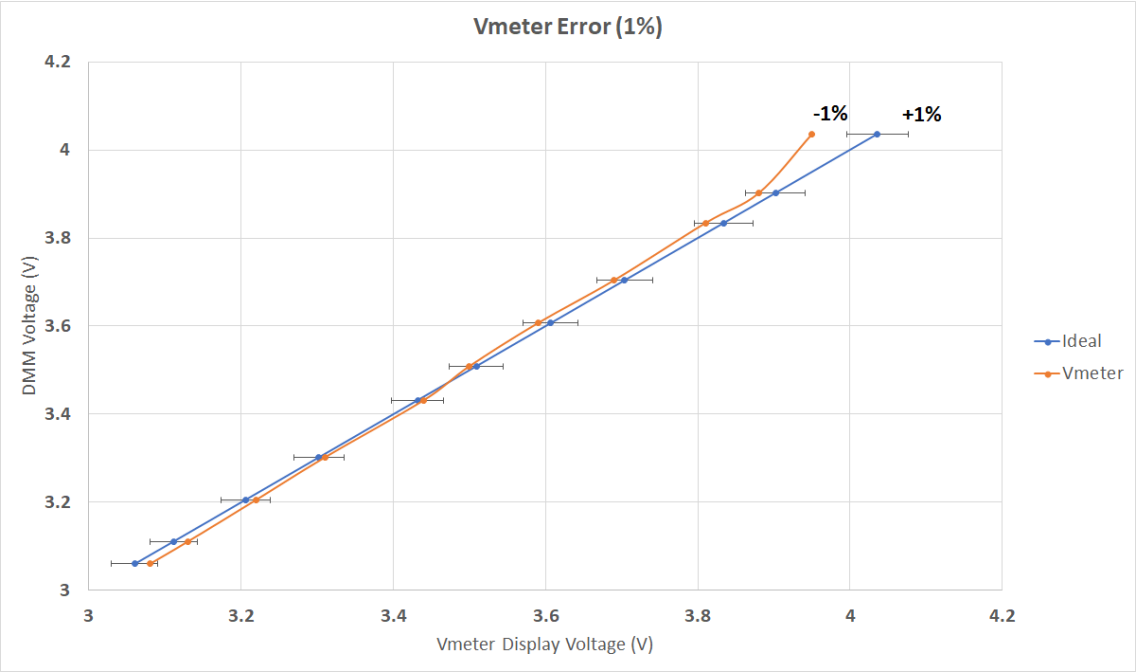


Figure 6.1: Plot of the voltage meter accuracy

In Figure 6.1, the circuit was tested by connecting a bench top power supply to the battery input terminals while the voltage was being measured with a DMM. The blue plot line shows the ideal reading with 0% error and the orange plot line is the readings displayed by the voltmeter, which are with in the boundaries of the black 1% error bars accuracy remains constant regardless of temperature.

6.3 Log Current meter

Current meter testing was done with a bench top supply which was increased logarithmically from 1mA to 1A and the current meter accuracy was compared with actual DMM readings. In Figure 6.2, the current meter was tested at different ambient temperatures. The blue line represents the ideal readings and the orange line represents the readings observed from the Analog Cross-Needle meter. It can be seen that it performs well at lower temperatures and the accuracy falls at higher temperatures.

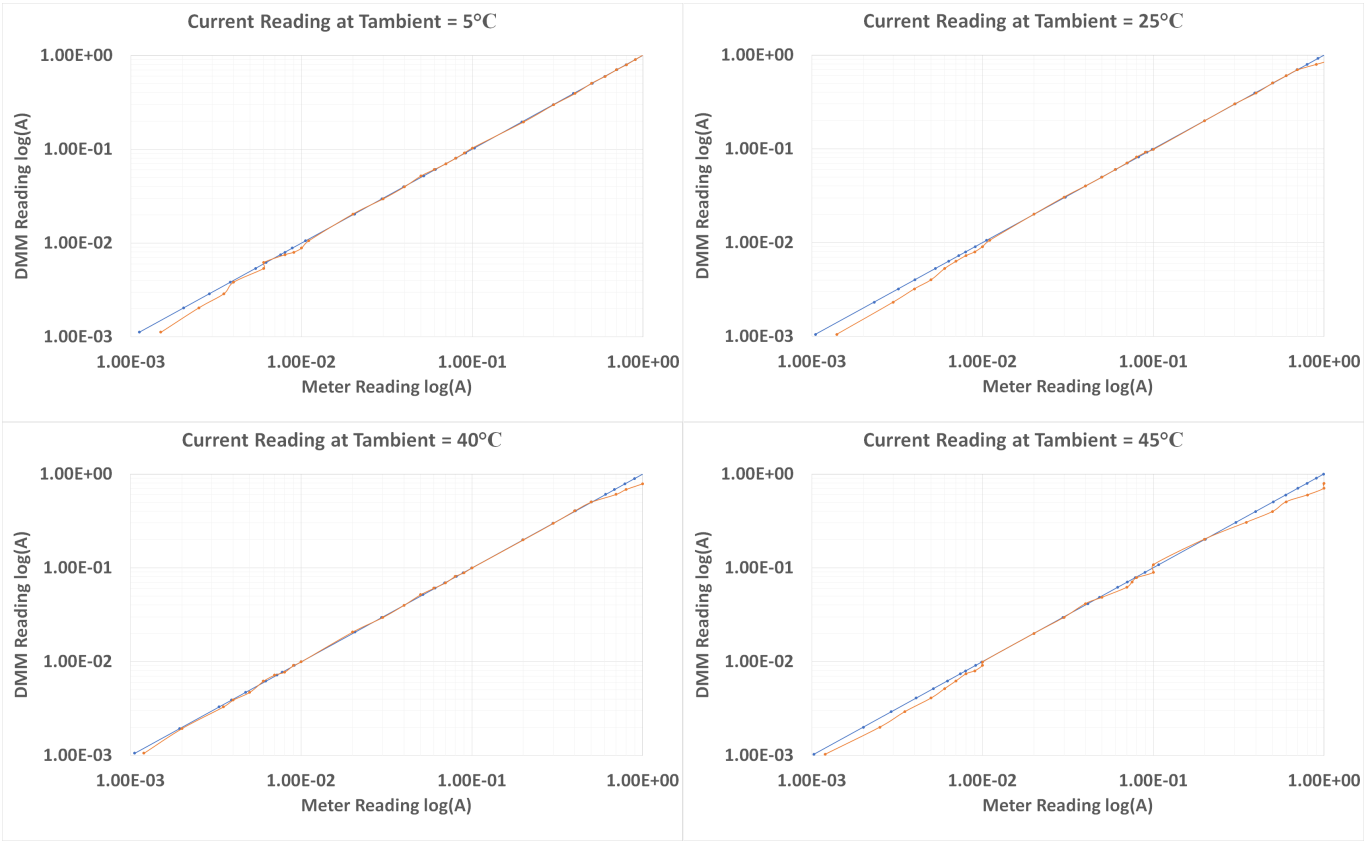


Figure 6.2: Plot of the current meter accuracy at different temperatures

It can be seen that the log current meter performs well especially at 5°C and accuracy begins to drop above the 100mA earlier at higher temperatures.

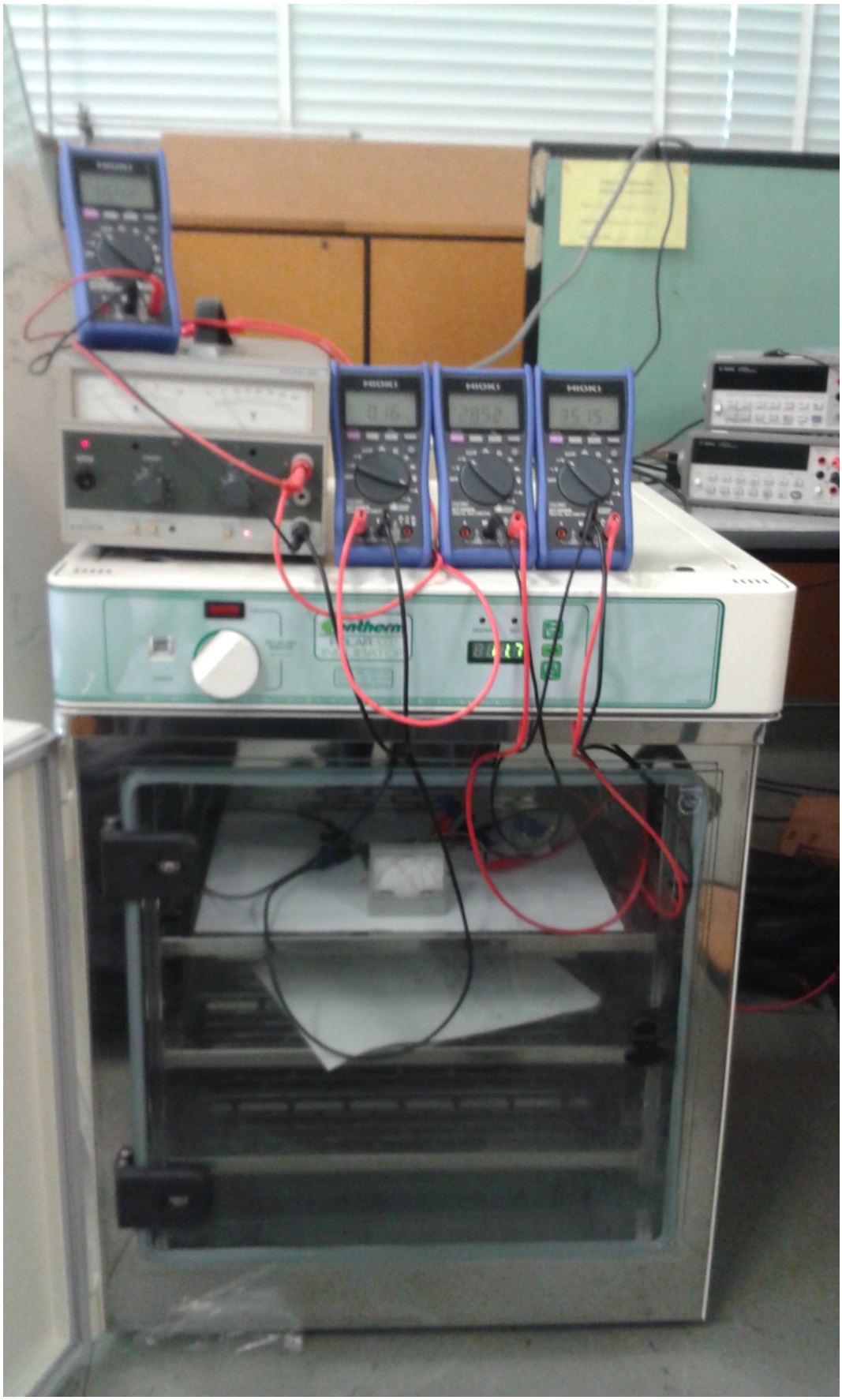


Figure 6.3: Photograph of the product being tested inside an incubator

Chapter 7

Conclusions

The Cross-Needle meter measures battery voltages and charge currents and is surprisingly accurate over a large range of current and temperature. There is a loss in accuracy as the current scale must span four decades with a sweep of 60 degrees. Quick referencing of estimating time remaining to charge can easily be done at a wide range from the evenly spread contour lines. The Current reading can give increased uncertainty at the high end of the scale and this is due to an increase in junction temperature of the diode and effects of series resistance. Estimating the current through the Shottkey diode shunt at high current and at higher ambient temperatures proved to be challenging.

This project concludes that using the cross-needle analog meter is indeed useful for quick referencing battery charge monitoring with a wide range of input supply current. I hope that this novel tool for gauging the time required to charge a battery will find wide application.

7.1 Future Work and Recommendations

A lot can be further developed from this project. One of the main regretful decision made was not choosing to use the PIC16F1825 micro-controller instead of the 16F684. The 16F1825 with two independent CCP modules would have improved the performance of the Cross-Needle meter by eliminating the issue of the vibrating needle pointer, caused by using the timer 0 as the second

PWM and this would offer another available timer. The PCB size could be significantly reduced by using SMD resistor and LED components. This would make the mounting aesthetic much more pleasing. The use of Fuel gauge IC for Lithium-Ion Cell might enhance the accuracy of SoC such as the LTC2941-1 which offers dedicated coulomb counting as well as temperature compensation of the cell, which was not implemented in this project [8].

Appendix A

A.1 METER.exe source code listing

Listing 1: METER.exe

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
using System.Linq;
using System.Text;
using System.IO;
//using System.Threading.Tasks;
using System.Windows.Forms;

namespace METER
{
    public partial class Form1 : Form
    {
        //timer
        Timer t = new Timer();
        //directory path
        string directoryPath =
            Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory);
        string fileName =
            String.Format("{0:yyyy-MM-dd_hh-mm-ss-tt}",
            DateTime.Now);

        int WIDTH = 665, HEIGHT = 690, leftNeedle = 500,
            rightNeedle = 500;

        //center
        int cxLEFT = 150;//LEFT Horizontal
        int cyLEFT = 570;//LEFT Vertical
        int cxRIGHT = 515;//RIGHT Horizontal
        int cyRIGHT = 570;//RIGHT Vertical

        double voltage;
        double current;
        double currentLog;
        double currentLog100;
        double currentSQ;
        double currentQB;
        double current4TH;
        double currentSQRT;
        double currentCosh;
```

```
        double currentTanh;

        double calc;

        double Vangle;
        double Iangle;

        double deltaX;
        double deltaY;

        //4 points of the needle
        int[] A = new int[2];//Start point of needle 1(left)
        int[] B = new int[2];//end point of needle 1(left)
        int[] C = new int[2];//start point of needle 2(right)
        int[] D = new int[2];//end point of needle 2(right)

        //find values for Voltage Ax + By = C
        //int[] Vabc = new int[3];
        //find values for current Ax + By = C
        //int[] Iabc = new int[3];

        //intersection
        int[] Inter = new int[2];

        Bitmap bmp;
        Graphics g;//drawing graphics object
        List<Color> colors = new List<Color> //list of colors
            to change when drawing
        {
            //yellow is hard to see in a white background
            Color.HotPink,Color.Orange,Color.Goldenrod,Color.ForestGreen,
            Color.Cyan,Color.Indigo,Color.Violet
        };

        List<List<Point>> Points = new List<List<Point>>();

        List<List<Point>> PointsLinear = new
            List<List<Point>>();
        //List<List<Point>> PointsLog = new
            List<List<Point>>();
        //List<List<Point>> PointsSQ = new List<List<Point>>();
        //List<List<Point>> PointsQB = new List<List<Point>>();
        //List<List<Point>> Points4TH = new
            List<List<Point>>();
        //List<List<Point>> PointsSQRT = new
            List<List<Point>>();
```

```
//constant list variables
List<double> time = new List<double>() { 600, 3600,
    21600, 43200, 86400, 604800, 1209600};//in seconds
    10min 1hour 6hours 24 hours and 1week
List<double> savedTime;//saves time values
List<int> Charge = Enumerable.Range(0,
    101).ToList();//battery charge interms of %

//2d lists each row is seperated by the time index
List<List<double>> Current = new
    List<List<double>>();//current 2d list
List<List<double>> ThetaL = new
    List<List<double>>();//Theta R 2d list
List<List<double>> ThetaR = new
    List<List<double>>();//Theta L 2d list
List<List<double>> xt = new List<List<double>>();//X
    coordinates 2d list
List<List<double>> yt = new List<List<double>>();//Y
    coordinates 2d list

public Form1()
{
    InitializeComponent();
}

private void Form1_Load(object sender, EventArgs e)
{
    // Enable double buffering.
    //this.DoubleBuffered = true;
    //create bitmap
    bmp = new Bitmap(WIDTH + 1, HEIGHT + 1);

    //backcolor
    this.BackColor = Color.White;

    //voltage label
    min1Label.Text = trackBar1.Minimum + "%";
    max1Label.Text = trackBar1.Maximum + "%";

    //current label
    min2Label.Text = trackBar2.Minimum / 10 + "A";
    max2Label.Text = trackBar2.Maximum / 10 + "A";

    calculateContour();
```

```
            //timer
            t.Interval = 1;//in millisecond
            t.Tick += new EventHandler(this.t_Tick);//event
                occurs every 1s
            t.Start();//start timer
        }

        private void t_Tick(object sender, EventArgs e)
        {
            // coodinates for end of the lines
            int[] handCoord = new int[2];//reset hand
                coordinates
            if (linearRadioButton.Checked)
            {
                calc = current;
                //Points = PointsLinear;
            }
            if (logRadioButton.Checked)
            {
                calc = currentLog;
                //Points = PointsLog;
            }
            if (log100RadioButton.Checked)
            {
                calc = currentLog100;
            }
            if (sqRadioButton.Checked)
            {
                calc = currentSQ;
                //Points = PointsSQ;
            }
            if (qbRadioButton.Checked)
            {
                calc = currentQB;
                //Points = PointsQB;
            }
            if (forthRadioButton.Checked)
            {
                calc = current4TH;
                //Points = Points4TH;
            }
            if (sqrtRadioButton.Checked)
            {
                calc = currentSQRT;
                //Points = PointsSQRT;
            }
```

```
if (coshRadioButton.Checked)
{
    calc = currentCosh;
}
if (tanhRadioButton.Checked)
{
    calc = currentTanh;
}

//draw labels
//display timer to charge
//create graphics
g = Graphics.FromImage(bmp);
g.SmoothingMode =
    System.Drawing.Drawing2D.SmoothingMode.AntiAlias;
g.PixelOffsetMode = PixelOffsetMode.HighQuality;

//rectangle
Rectangle rect = new Rectangle(0, 0, 665, 570);
//clear
g.Clear(Color.White);

//Draw Arc
g.DrawArc((new Pen(Color.Blue, 1f)), new
    Rectangle((-500 + 150), (HEIGHT - 500 - 120),
    1000, 1000), 280.0F, 60.0F);
g.DrawArc((new Pen(Color.Green, 1f)), new
    Rectangle((WIDTH - 500 - 150), (HEIGHT - 500 -
    120), 1000, 1000), 200.0F, 60.0F);
g.DrawString("Voltage(%)", new Font("Arial", 12),
    Brushes.Green, new PointF(120, 150));
g.DrawString("Current(A)", new Font("Arial", 12),
    Brushes.Blue, new PointF(460, 150));


    g.DrawString("Time to Charge: " +
        calculateTime(voltage, calc) + "hours ",
        new Font("Arial", 12), Brushes.Black, new
        PointF(0, 2));
    //display % voltage
    g.DrawString(voltage + "% ", new Font("Arial",
        12), Brushes.Green, new PointF(140, 167));
    //display current
    //change offset for display current
    g.DrawString(calc + "A", new Font("Arial",
        12), Brushes.Blue, new PointF(480, 167));
```

```
                ////////////////////////////////////////////////////////////
                //leftNeedle change when changing current
                    scale i.e convert to 0 to 10 scale
                //current must be in 0 to 10 scale
                handCoord = leftCoord(6 * current + 20,
                    leftNeedle);
                g.DrawLine(new Pen(Color.Red, 1f), new
                    Point(cxLEFT, cyLEFT), new
                    Point(handCoord[0], handCoord[1]));
                //calculate current needle angle
                deltaX = handCoord[0] - cxLEFT;
                deltaY = handCoord[1] - cyLEFT;
                //angle relative
                Iangle = -Math.Atan2(deltaY, deltaX) * 180 /
                    Math.PI;
                //get A,B
                A[0] = cxLEFT;
                A[1] = cyLEFT;
                B = handCoord;//left
                              ////////////////////////////////////////////////
                              //find Ax + By = C for current
                              //Iabc[0] = (int)deltaY;
                              //Iabc[1] = (int)deltaX;
                              //C = x1y2 - x2y1
                              //Iabc[2] = cxLEFT *
                                  handCoord[1] - cyLEFT *
                                  handCoord[0];
                             ////////////////////////////////////////////////
                              //rightNeedle change when
                                  changing voltage scale i.e
                                  convert to 0 to 10 scale
                handCoord = rightCoord((6 * voltage / 10 +
                    20), rightNeedle);
                g.DrawLine(new Pen(Color.Red, 1f), new
                    Point(cxRIGHT, cyRIGHT), new
                    Point(handCoord[0], handCoord[1]));
                //calculate voltage needle angle
                deltaX = handCoord[0] - cxRIGHT;
                deltaY = handCoord[1] - cyRIGHT;
                //angle relative
                Vangle = 180 + Math.Atan2(deltaY, deltaX) *
                    180 / Math.PI;
                //get C,D
                C[0] = cxRIGHT;
                C[1] = cyRIGHT;
                D = handCoord;//right
```

```
                    ////////////////////////////////////////////////
                    //find Ax + By = C for voltage
                    //Vabc[0] = (int)deltaY;
                    //Vabc[1] = (int)deltaX;
                    //C = x1y2 - x2y1
                    //Vabc[2] = cxRIGHT *
                        handCoord[1] - cyRIGHT *
                        handCoord[0];

                //calculate intersection angle must be in
                    radians
                Inter = Intersect(Iangle * Math.PI / 180,
                    Vangle * Math.PI / 180);

                for (int t = 0; t < time.Count();
                    t++)//calculate and fill current required
                    to charge and ThetaL values.
                {
                    try
                    {
                        //draw contour lines
                        GraphicsPath graphicsPath = new
                            GraphicsPath();
                        //path.AddLines(Points[t].ToArray());
                        graphicsPath.AddCurve(Points[t].ToArray());
                        g.DrawPath(new Pen(colors[t %
                            colors.Count()], 1f), graphicsPath);
                        g.DrawString(time[t] + "seconds ", new
                            Font("Arial", 12), new
                            SolidBrush(colors[t %
                            colors.Count()]), new
                            Point(Points[t][Points[t].Count()/2].X,
                            Points[t][Points[t].Count/2].Y-10));
                    }
                    catch
                    {
                    }
                }
                pictureBox1.Image = bmp;
                //dispose
                g.Dispose();
                //disp time
                this.Text = "Vangle :" + Vangle + "Deg " +
                    "Iangle :" + Iangle + "Deg " + "Intersect: "
                + Inter[0] + "," + Inter[1];
            }
```

```
        ////////////////////////////////////////////////////////////
        private void trackBar1_Scroll(object sender, EventArgs
            e)
        {
            //trackBar change Voltage
            voltage = trackBar1.Value;
        }

        private void trackBar2_Scroll(object sender, EventArgs
            e)
        {
            //trackbar change current
            current = trackBar2.Value;
            //0 to 10 scale
            current = current / 10;
            //log scale
            currentLog = Math.Pow(10, (current / 2.5 -
                3));//logrithmic scale
            currentLog100 = Math.Pow(100, (7*current / 20 -
                3));//logrithmic base 100 scale
            currentSQ = Math.Pow(current, 2) / 10;//squared
                scale
            currentQB = Math.Pow(current, 3) / 100;//cubed
                scale
            current4TH = Math.Pow(current, 4) / 1000;//forth
                power scale
            currentSQRT = Math.Sqrt(current) *
                Math.Sqrt(10);//square root scale
            currentCosh = 10*Math.Cosh(current) /
                Math.Cosh(10);//cosh scale
            currentTanh = 10*(Math.Tanh(current-10)+1);//tanh
                scale

            //currentSQ = Math.Pow(current, 2)/10;
            //current = 0.09545484566618341 *
                Math.Exp(0.46516870565536284 * current);
            //convert current to log scale
            //current = Math.Pow(10, current);
            //change scale to 0~10A with 1/100 divisions
            //current = current /10;
            //current = current;
            //current = 10 * Math.Log10(current) / 8;
            //current = 0.09545484566618341 *
                Math.Log10(0.46516870565536284 * current);
        }
```

```
        ////////////////////////////////////////////////////////////
        //coord for left Needle
        private int[] leftCoord(double value, double
            needleLength)
        {
            int[] coord = new int[2];
            //value *= 3.75;   //3.75 degree intervals

            if (value >= 0 && value <= 180)
            {
                coord[0] = cxLEFT + (int)(needleLength *
                    Math.Cos(Math.PI * value / 180));
                coord[1] = cyLEFT - (int)(needleLength *
                    Math.Sin(Math.PI * value / 180));
            }
            else
            {
                coord[0] = cxLEFT - (int)(needleLength *
                    -Math.Cos(Math.PI * value / 180));
                coord[1] = cyLEFT - (int)(needleLength *
                    Math.Sin(Math.PI * value / 180));
            }
            return coord;
        }

        //coord for right Needle
        private int[] rightCoord(double value, double
            needleLength)
        {
            int[] coord = new int[2];
            //value *= 3.75;   //3.75 degree intervals

            if (value >= 0 && value <= 180)
            {
                coord[0] = cxRIGHT - (int)(needleLength *
                    Math.Cos(Math.PI * value / 180));
                coord[1] = cyRIGHT - (int)(needleLength *
                    Math.Sin(Math.PI * value / 180));
            }
            else
            {
                coord[0] = cxRIGHT + (int)(needleLength *
                    -Math.Cos(Math.PI * value / 180));
                coord[1] = cyRIGHT - (int)(needleLength *
                    Math.Sin(Math.PI * value / 180));
            }
```

```
        return coord;
    }

    //calculate needle intersections
    private int[] Intersect(double angle1, double angle2)
    {
        int[] intcpt = new int[2];
        double d = (365 * Math.Tan(angle2)) /
            (Math.Tan(angle1) + Math.Tan(angle2));
        double y = d * Math.Tan(angle1);
        intcpt[0] = cxLEFT + (int)d;
        intcpt[1] = cyLEFT - (int)y;

        return intcpt;
    }

    //calculate time
    private double calculateTime(double v, double i)
    {
        double time = 0;
        if (v == 100)
        {
            time = 0;
        }
        else
        {
            //change to decimal
            v = v / 100;
            //Li-Ion capacity: 2.5Ah change for different
                battery
            time = (2.5 - 2.5 * v) / i;
        }

        return time;
    }

    //contour calculations performed here
    private void calculateContour()
    {
        if (contourCheckBox.Checked)
        {
            //reset the lists
            Current.Clear();
            ThetaR.Clear();
            ThetaL.Clear();
            xt.Clear();
```

```
            yt.Clear();
            Points.Clear();
            /*----------------------calculations----------------------*/
            for (int t = 0; t < time.Count();
                t++)//calculate and fill current required
                to charge and ThetaL values.
            {
                double I;//current
                double TR;//theta R
                double TL;//theta L
                double X;//X intersection
                double Y;//Y intersection
                int[] tempIntersect;
                var currentTemp = new List<double>();
                var ThetaRTemp = new List<double>();
                var ThetaLTemp = new List<double>();
                var XTemp = new List<double>();
                var YTemp = new List<double>();
                var PointsTemp = new List<Point>();
                for (int q = 0; q < Charge.Count(); q++)
                {
                    I = 3600.00 * (2.500 - 2.500 *
                        Charge[q] / 100.00) /
                        time[t];//calculate current
                        required to charge
                    TR = 160.00 - Charge[q] * 60.00 /
                        100.00;//calculate theta R
                    TL = 6.00 * I + 20.00;//Linear Scale
                    if (lnRadioButton.Checked) { TL =
                        6.514 * (Math.Log(I)) + 65.00;
                        }//calculate theta L Log scale
                    if (logRadioButton.Checked) { TL =
                        15.00 * (Math.Log10(I)) + 65.00;
                        }//calculate theta L Log scale
                    if (log100RadioButton.Checked) { TL =
                        15.00 * (LogBase(I, 100)) + 65.00;
                        }//Logbase 100 using LogBase Method
                    if (sqRadioButton.Checked) { TL =
                        Math.Sqrt(I) * 60.00 /
                        Math.Sqrt(10) + 20.00; }//Square
                        root Scale
                    if (qbRadioButton.Checked) { TL =
                        Math.Pow(I, (1.00 / 3.00)) * 60.00
                        / Math.Pow(10.00, (1.00 / 3.00)) +
                        20.00; }//cube root
                    if (forthRadioButton.Checked) { TL =
```

```
    Math.Pow(I, (1.00 / 4.00)) * 60.00
    / Math.Pow(10.00, (1.00 / 4.00)) +
    20.00; }///quad root
if (coshRadioButton.Checked) { TL =
    60.00 * Math.Cosh(I) /
    Math.Cosh(10.00); }//cosh scale
if (tanhRadioButton.Checked) { TL =
    60.00 * Math.Tanh(I - 10.00) +
    80.00; }//cosh scale
if (sqrtRadioButton.Checked) { TL =
    Math.Pow(I, 2.00) * 60.00 /
    Math.Pow(10.00, 2.00) + 20.00;
    }//Squared Scale


//intersect calculation using line
    formula method
//X = (cxRIGHT-cxLEFT) * Math.Tan(TR *
    Math.PI / 180) / (Math.Tan(TR *
    Math.PI / 180) - Math.Tan(TL *
    Math.PI / 180));//365 is the
    distance between the needle center
    and angle is converted to radians
//Y = X * Math.Tan(TL * Math.PI /
    180);//calculate Y intersection

//intersect using intersection method
tempIntersect = Intersect(TL * Math.PI
    / 180.00, Math.PI - TR * Math.PI /
    180.00);//must convert angles to
    radians
X = tempIntersect[0];
Y = tempIntersect[1];

currentTemp.Add(I);
ThetaRTemp.Add(TR);
ThetaLTemp.Add(TL);
XTemp.Add(X);
YTemp.Add(Y);
if (cxLEFT + (int)X > 0 && cyLEFT -
    (int)Y > 0)
{
    //PointsTemp.Add(new Point(cxLEFT
        + (int)X, cyLEFT - (int)Y));
    PointsTemp.Add(new Point((int)X,
        (int)Y));
```

```
                }
            }
            //fill in 2d lists
            Current.Add(currentTemp);//add to current
                list. Index is the same as time
            ThetaR.Add(ThetaRTemp);
            ThetaL.Add(ThetaLTemp);
            xt.Add(XTemp);
            yt.Add(YTemp);
            Points.Add(PointsTemp);
        }
    }
    else
    {
        time.Clear();
    }
}

// Calculate log(number) in the indicated log base.
private double LogBase(double number, double log_base)
{
    return Math.Log(number) / Math.Log(log_base);
}

//saves voltage and current at this instant to a text
    file with date and time
private void saveButton_Click(object sender, EventArgs
    e)
{
    try
    {
        using (StreamWriter writer = new
            StreamWriter(fileName+".txt", true))
        {
            writer.WriteLine("Voltage:{0}\%
                Current:{1}A Time to Charge:{2}hours
                Intersection point in C#
                coordinates:{3},{4}", voltage, calc,
                calculateTime(voltage, calc), Inter[0],
                Inter[1]);
            pictureBox1.Image.Save(fileName+".png",
                ImageFormat.Png);
        }
    }
    catch (Exception exception)
    {
```

```
            MessageBox.Show(exception.Message);
        }
    }

    //Radio Button Methods
    private void linearRadioButton_CheckedChanged(object
        sender, EventArgs e)
    {
        if (linearRadioButton.Checked)
            {calculateContour(); }
    }

    private void logRadioButton_CheckedChanged(object
        sender, EventArgs e)
    {
        if (logRadioButton.Checked) { calculateContour(); }
    }

    private void log100RadioButton_CheckedChanged(object
        sender, EventArgs e)
    {
        if (log100RadioButton.Checked) {
            calculateContour(); }
    }

    private void sqRadioButton_CheckedChanged(object
        sender, EventArgs e)
    {
        if (sqRadioButton.Checked) { calculateContour(); }
    }

    private void qbRadioButton_CheckedChanged(object
        sender, EventArgs e)
    {
        if (qbRadioButton.Checked) { calculateContour(); }
    }

    private void forthRadioButton_CheckedChanged(object
        sender, EventArgs e)
    {
        if (forthRadioButton.Checked) {
            calculateContour(); }
    }

    private void changeTimeButton_Click(object sender,
        EventArgs e)
```

```
        {
            if (time1TextBox.Text != " " || time2TextBox.Text
                != " " || time3TextBox.Text != " " ||
                time4TextBox.Text != " " || time5TextBox.Text
                != " " || time6TextBox.Text != " ")
            {
                time.Clear();
                try
                {
                    if (time1TextBox.Text != " ") {
                        time.Add(double.Parse(time1TextBox.Text));
                        }
                    if (time2TextBox.Text != " ") {
                        time.Add(double.Parse(time2TextBox.Text));
                        }
                    if (time3TextBox.Text != " ") {
                        time.Add(double.Parse(time3TextBox.Text));
                        }
                    if (time4TextBox.Text != " ") {
                        time.Add(double.Parse(time4TextBox.Text));
                        }
                    if (time5TextBox.Text != " ") {
                        time.Add(double.Parse(time5TextBox.Text));
                        }
                    if (time6TextBox.Text != " ") {
                        time.Add(double.Parse(time6TextBox.Text));
                        }
                    else
                    {
                        time = new List<double>() { 600, 3600,
                            21600, 43200, 86400, 604800,
                            1209600 } ;
                    }
                    time1TextBox.Clear();
                    time2TextBox.Clear();
                    time3TextBox.Clear();
                    time4TextBox.Clear();
                    time5TextBox.Clear();
                    time6TextBox.Clear();

                    savedTime = time;
                }
                catch
                {
                    time1TextBox.Clear();
                    time2TextBox.Clear();
```

```
                time3TextBox.Clear();
                time4TextBox.Clear();
                time5TextBox.Clear();
                time6TextBox.Clear();

                time.Clear();
                MessageBox.Show("You Must enter 6 numbers
                    into the boxes","Error");
                time = new List<double>() { 600, 3600,
                    21600, 43200, 86400, 604800, 1209600 };
            }
        }
        else
        {
            time.Clear();
            MessageBox.Show("Please Insert a
                Number","Error");
            time = new List<double>() { 600, 3600, 21600,
                43200, 86400, 604800, 1209600 };
        }
        calculateContour();
    }

    private void resetButton_Click(object sender,
        EventArgs e)
    {
        time.Clear();
        time = new List<double>() { 600, 3600, 21600,
            43200, 86400, 604800, 1209600 };
        calculateContour();
    }

    private void sqrtRadioButton_CheckedChanged(object
        sender, EventArgs e)
    {
        if (sqrtRadioButton.Checked) { calculateContour();
            }
    }

    private void lnRadioButton_CheckedChanged(object
        sender, EventArgs e)
    {
        if (lnRadioButton.Checked) { calculateContour(); }
    }

    private void coshRadioButton_CheckedChanged(object
```

```
            sender, EventArgs e)
        {
            if (coshRadioButton.Checked) { calculateContour();
                }
        }

        private void tanhRadioButton_CheckedChanged(object
            sender, EventArgs e)
        {
            if (tanhRadioButton.Checked) { calculateContour();
                }
        }

        private void contourCheckBox_CheckedChanged(object
            sender, EventArgs e)
        {
            if (contourCheckBox.Checked)
            {
                time = savedTime;
                scaleBox.Enabled = true;
            }
            else
            {
                scaleBox.Enabled = false;
                savedTime = time;
                time.Clear();
            }
            calculateContour();
        }
    }
}
```

End of Listing 1 METER.exe

Appendix B

B.1 Solidworks macro code listing

Listing 2: SolidWorks Contour Automation

```
using System.Collections.Generic;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

//SOLID WORKS API
using SldWorks;
using SwConst;

namespace SolidWorks_Contour_Automation
{
    public partial class Form1 : Form
    {
        //origin
        const double drawingOriginX = 66.45334818;
        const double drawingOriginY = 146.83344753;
        //dimentions
        const double WIDTH = 69.50;//width of rectangle
        const double HEIGHT = 69.00;//height of rectangle
        /*----------------global variables----------------*/
        //arc dimentions Y = 12mm X = 38mm  apart i.e 69.50 +
            15.75 and 69.5 - 15.75
        double leftNeedle = 189, rightNeedle = 189;
        double cxLEFT = (drawingOriginX + 15.75) / 1000;//LEFT
            Horizontal
        double cyLEFT = (drawingOriginY + 12.00) / 1000;//LEFT
            Vertical
        double cxRIGHT = (drawingOriginX + WIDTH - 15.75) /
            1000;//RIGHT Horizontal
        double cyRIGHT = (drawingOriginY + 12.00) /
            1000;//RIGHT Vertical

        //rectangle dimentions
        double startPointX = drawingOriginX / 1000;
        double startPointY = drawingOriginY / 1000;
        double endPointX = (drawingOriginX + WIDTH) / 1000;
        double endPointY = (drawingOriginY + HEIGHT) / 1000;
```

```
        //List<List<Point>> Points = new List<List<Point>>();
        List<List<double>> Dpoints = new List<List<double>>();
        List<double> time = new List<double>() { 600, 3600,
            21600, 86400, 604800 };//in seconds 10min 1hour
            6hours 24 hours and 1week
        List<int> Charge = Enumerable.Range(0,
            101).ToList();//battery charge interms of %

        //2d lists each row is seperated by the time index
        List<List<double>> Current = new
            List<List<double>>();//current 2d list
        List<List<double>> ThetaL = new
            List<List<double>>();//Theta R 2d list
        List<List<double>> ThetaR = new
            List<List<double>>();//Theta L 2d list
        List<List<double>> xt = new List<List<double>>();//X
            coordinates 2d list
        List<List<double>> yt = new List<List<double>>();//Y
            coordinates 2d list

        //solidworks constructors
        SldWorks.SldWorks swApp = new SldWorks.SldWorks();
        bool boolstatus = false;
        int longstatus = 0;
        ModelDoc2 swDoc = null;

        /*-------------------------------------------------*/
        public Form1()
        {
            InitializeComponent();
        }

        private void createButton_Click(object sender,
            EventArgs e)
        {
            calculateContour();
            //create object from solidworks class

            swApp.Visible = true;

            swDoc = ((ModelDoc2)(swApp.NewDocument(
            "C:\\ProgramData\\SolidWorks\\SOLIDWORKS
                2016\\templates\\Drawing.drwdot", 2, 0.2794,
                0.43179999999999996)));
            swApp.ActivateDoc2("Draw3 - Sheet1", false, ref
                longstatus);
```

```
swDoc = ((ModelDoc2)(swApp.ActiveDoc));
boolstatus =
    swDoc.Extension.SetUserPreferenceToggle(
    ((int)(swUserPreferenceToggle_e.swSketchAddConstToRectEntity)),
    ((int)(swUserPreferenceOption_e.swDetailingNoOptionSpecified)),
        false);
boolstatus =
    swDoc.Extension.SetUserPreferenceToggle(
    ((int)(swUserPreferenceToggle_e.swSketchAddConstLineDiagonalType)),
    ((int)(swUserPreferenceOption_e.swDetailingNoOptionSpecified)),
        true);
Array vSkLines = null;
vSkLines =
    ((Array)(swDoc.SketchManager.CreateCornerRectangle(startPointX,
    endPointY, 0, endPointX, startPointY, 0)));
swDoc.ViewZoomtofit2();
swDoc.ViewZoomtofit2();
swDoc.ViewZoomtofit2();

//create arc
SketchSegment skSegment = null;
//must get start and end points from drawing
skSegment =
    ((SketchSegment)(swDoc.SketchManager.CreateArc(cxLEFT,
    cyLEFT, 0,
    cxLEFT+50*Math.Cos(20*Math.PI/180)/1000, cyLEFT
    + 50 * Math.Sin(20 * Math.PI / 180)/1000, 0,
    cxLEFT + 50 * Math.Cos(80 * Math.PI / 180) /
    1000, cyLEFT + 50 * Math.Sin(80 * Math.PI /
    180) / 1000, 0, 1)));
skSegment =
    ((SketchSegment)(swDoc.SketchManager.CreateArc(cxRIGHT,
    cyRIGHT, 0, cxRIGHT + 50 * Math.Cos((180-80) *
    Math.PI / 180) / 1000, cyRIGHT + 50 *
    Math.Sin((180-80) * Math.PI / 180) / 1000, 0,
    cxRIGHT + 50 * Math.Cos((180-20) * Math.PI /
    180) / 1000, cyRIGHT + 50 * Math.Sin((180-20) *
    Math.PI / 180) / 1000, 0, 1)));
//skSegment =
    ((SketchSegment)(swDoc.SketchManager.CreateArc(cxLEFT,
    cyLEFT, 0, 0.12843797922, 0.17593445477, 0,
    0.09013575706, 0.20807383518, 0, 1)));
//skSegment =
    ((SketchSegment)(swDoc.SketchManager.CreateArc(cxRIGHT,cyRIGHT,0,
    0.11227093930, 0.20807383518, 0,
    0.07396871714, 0.17593445470, 0, 1)));
```

```
        //create arc lines
        /*for (int i = 0; i < 37; i++)
        {
            double LogI = (i+1)*(0.001 * i * Math.Pow(10,
                i));
            double angle = 15 * Math.Log10(LogI);
            skSegment =
                ((SketchSegment)(swDoc.SketchManager.CreateLine(cxLEFT
                + 50 * Math.Cos(angle * Math.PI / 180) /
                1000, cyLEFT + 50 * Math.Sin(angle *
                Math.PI / 180) / 1000,0, cxLEFT + 50 *
                Math.Cos(angle * Math.PI / 180)/1000,
                cyLEFT + 50 * Math.Sin(angle * Math.PI /
                180) / 1000,0)));

        }*/

        //create contours
        Array pointArray = null;

        //create annotations

        //draw contours and labels
        for (int t = 0; t < time.Count(); t++)
        {
            //draw contour lines
            pointArray = Dpoints[t].ToArray();
            skSegment =
                ((SketchSegment)(swDoc.SketchManager.CreateSpline(pointArray)));

            //insert Letters
            writestring((Math.Round(time[t]/3600,3)).ToString()+"Hours",
                Dpoints[t][150], Dpoints[t][151], 1.00);
        }


        string path =
            System.Environment.GetFolderPath(
            System.Environment.SpecialFolder.Desktop);
        longstatus =
            swDoc.SaveAs3(path+"\\contours.SLDDRW", 0, 2);
    }

    private void calculateContour()
    {
```

```
/*----------------calculations-----------------*/
for (int t = 0; t < time.Count(); t++)//calculate
    and fill current required to charge and ThetaL
    values.
{
    double I;//current
    double TR;//theta R
    double TL;//theta L
    double X;//X intersection
    double Y;//Y intersection
    var currentTemp = new List<double>();
    var ThetaRTemp = new List<double>();
    var ThetaLTemp = new List<double>();
    var XTemp = new List<double>();
    var YTemp = new List<double>();
    var XYZTemp = new List<double>();
    var PointsTemp = new List<Point>();

    for (int q = 0; q < Charge.Count(); q++)
    {
        I = 3600.00 * (2.500 - 2.500 * Charge[q] /
            100.00) / time[t];//calculate current
            required to charge
        TR = 160.00 - Charge[q] * 60.00 /
            100.00;//calculate theta R
        //TL = 6.00 * I + 20.00;//Linear Scale
        //TL = 15.00 * (Math.Log10(I)) +
            65.00;//calculate theta L Log base 10
            scale
        //TL = Math.Sqrt(I) *
            60.00/Math.Sqrt(10.00);//Square root
            Scale
        //TL = Math.Pow(I,2.0) * 60.00 /
            Math.Pow(10.00, 2.00);//cube root
        //TL = Math.Pow(I,(1.0/3.0)) * 60 /
            Math.Pow(10, (1.0 / 3.0));//cube root
        //TL = Math.Pow(I, (1.0 / 4.0)) * 60 /
            Math.Pow(10, (1.0 / 4.0));//quad root
        TL = 6.514 * (Math.Log(I)) +
            65.00;//calculate theta L Log base e
            scale
        X = (cxRIGHT - cxLEFT) * Math.Tan(TR *
            Math.PI / 180.00) / (Math.Tan(TR *
            Math.PI / 180.00) - Math.Tan(TL *
            Math.PI / 180.00));//38 is the distance
            between the needle center and angle is
```

```
                    converted to radians
                Y = X * Math.Tan(TL * Math.PI /
                    180.00);//calculate Y intersection

                X = X + cxLEFT;//adjust for offset
                Y = Y + cyLEFT;//adjust for offset

                currentTemp.Add(I);
                ThetaRTemp.Add(TR);
                ThetaLTemp.Add(TL);
                XTemp.Add(X);
                YTemp.Add(Y);

                if (cxLEFT + X > 0 && Y > 0)
                {
                    //PointsTemp.Add(new Point(cxLEFT +
                        (int)X, cyLEFT - (int)Y));
                    XYZTemp.Add(X);//offset for solidworks
                        rectangle
                    XYZTemp.Add(Y);
                    XYZTemp.Add(0);
                }
            }
            //fill in 2d lists
            Current.Add(currentTemp);//add to current
                list. Index is the same as time
            ThetaR.Add(ThetaRTemp);
            ThetaL.Add(ThetaLTemp);
            xt.Add(XTemp);
            yt.Add(YTemp);
            //Points.Add(PointsTemp);
            Dpoints.Add(XYZTemp);
        }
    }

    private void writestring(string input, double xPos,
        double yPos, double height)
    {
        //
        Annotation myAnnotation = null;
        TextFormat myTextFormat = null;
        Note myNote = null;

        //xPos = xPos / 1000;//convert to m
        //yPos = yPos / 1000;
        height = height / 1000;
```

```
            myNote = ((Note)(swDoc.InsertNote(input)));
            if ((myNote != null))
            {
                myNote.LockPosition = false;
                myNote.Angle = 0;
                boolstatus = myNote.SetBalloon(0, 0);
                myAnnotation =
                    ((Annotation)(myNote.GetAnnotation()));
                if ((myAnnotation != null))
                {
                    longstatus =
                        myAnnotation.SetLeader3(((int)(swLeaderStyle_e.swNO_LEADER)),
                        0, true, false, false, false);
                    boolstatus =
                        myAnnotation.SetPosition(xPos, yPos, 0);
                    //
                    myTextFormat =
                        ((TextFormat)(swDoc.GetUserPreferenceTextFormat(0)));
                    myTextFormat.Italic = false;
                    myTextFormat.Underline = false;
                    myTextFormat.Strikeout = false;
                    myTextFormat.Bold = false;
                    myTextFormat.Escapement = 0;
                    myTextFormat.LineSpacing = 0.001;
                    myTextFormat.CharHeight = height;
                    myTextFormat.TypeFaceName = "Arial";
                    myTextFormat.WidthFactor = 1;
                    myTextFormat.ObliqueAngle = 0;
                    myTextFormat.LineLength = 0;
                    myTextFormat.Vertical = false;
                    myTextFormat.BackWards = false;
                    myTextFormat.UpsideDown = false;
                    myTextFormat.CharSpacingFactor = 1;
                    boolstatus = myAnnotation.SetTextFormat(0,
                        false, myTextFormat);
                }
            }
        }
    }
}
```

End of Listing 2: SolidWorks Contour Automation

Appendix C

C.1 Main.c firmware code listing

Listing 3: Firmware main.c

```
/*
 * Author: Hiroshi Mohri ID:1143192
 * Cross Needle Meter Project
 * Electronics Thesis ENEL516C
 * PIC16F684 Compiler: HITECH-C compiler 9.8
 * ************************************
 * Channels are:
 * RA5:LED1; RA3:LED2; RC3:LED3 RC0:LED4
 * RA0:battery voltage input; RA1:Reference 2.5Volt input; RA2 Temperature sense;
 * RC1:Button input; RA4 AN3:Current meter sense ; RC2 AN6:Voltage meter sense;
 * RC4:Current meter PWM;RC5 volt meter PWM
 * ************************************
 */

/********************************** header files ***************************************/
#include <htc.h>
#include <pic.h>
#include <stdlib.h>

/********************************** defines(no memory allocation) ***************************************/
#define _LEGACY_HEADERS
#define SELECT_OPAMP CHS2=0;CHS1=0;CHS0=0;     //AN0
#define SELECT_VREF CHS2=0;CHS1=0;CHS0=1;    //AN1
#define SELECT_VTEMP CHS2=0;CHS1=1;CHS0=0;     //AN2
#define SELECT_IMSENSE CHS2=0;CHS1=1;CHS0=1;   //AN3
#define SELECT_VMSENSE CHS2=1;CHS1=1;CHS0=0;   //AN6

#define LED1 RA3                //Red LED
#define LED2 RA5                //Green LED
#define LED3 RC3                //Yellow LED
#define LED4 RC0                //Blue LED
#define Imeter RC4
#define Vmeter RC5

//#define RM 39L                //value of sensing resistor 39K
#define RM 4110L                //total resistance from pic through meter. 41100/10 = 4110
#define METERSCALEUA 100            //cuurent in meter in uA
#define _XTAL_FREQ 4000000
#define Q 11605L                //q/nK electrical charge/boltzman constant*n assuming n = 1

/********************************** function prototypes ***************************************/
char lookUpVd(signed int);            //look up table function prototype
/********************************** variables(memory allocation) ***************************************/
//__CONFIG(INTIO & WDTDIS & MCLRDIS & BOREN & UNPROTECT & PWRTEN);
__CONFIG(INTIO & WDTDIS & PWRTDIS & MCLRDIS & UNPROTECT & BORDIS & IESODIS & FCMDIS);
//volatile unsigned char  timer0    @ 0x01;   //variable referring to the location of Timer0 registers
//volatile short unsigned int timer1 @0x0E;     //variable referring to the location of
```

```
    Timer1 registers
//volatile unsigned char timer2 @0x11;          //variable referring to the location of
    Timer2 registers
//volatile unsigned char OPTION_REG @0x81;        //variable referring to the location of
     OPTION registers

//double Vref = 2.5;              //2500mV constant Vref
int Vdd;                      //battery voltage Value
int Vdiode;                     //Diode voltage
int Vth;                      //thermal voltage
char IRs;                     //diode series resistance drop
signed int lnId;                  //Diode Current logrithmic ln(Id)
int T;                        //Board Temperature
short int display;                  //Meter PWM value
int Vref = 640;                 //ADC reading of reference voltage, 2.5V at Vdd=4V
int Vm = 970;                   //ADC reading of meter voltage, 3.79V at Vdd=4V
//int Im = 970;                   //meter current when PWM on in uA
short int FSD;                    //0~255 PWM value for FSD at this instant of suply
    voltage and meter resistor
short int dutyCycle;              //timer 0 dutyCycle variable
bit pwmstatus;                  //variable to swich from on off
char i;                       //8bit variable for counting

char lookUpVd(signed int x)           //look up table for diode Rs voltage
{
  if(x<=0)
  {
    return 10;
  }
  if(x<109&&x>=69)
  {
    return 20;
  }
  if(x<138&&x>=109)
  {
    return 30;
  }
  if(x<160&&x>=138)
  {
    return 40;
  }
  if(x<179&&x>=160)
  {
    return 50;
  }
  if(x<194&&x>=179)
  {
    return 60;
  }
  if(x<208&&x>=194)
  {
    return 70;
  }
  if(x<219&&x>=208)
  {
    return 80;
  }
```

```
  if(x<230&&x>=219)
  {
    return 90;
  }
  if(x>=230)
  {
    return 100;
  }
}
/********************************** initialise function
    ***************************************/
void initialise()
{
  /**** PORT CONFIGURATION ****/
  TRISA = 0b00010111;     //RA0(Battery Current),RA1(Reference 2.5v),RA2(Temperature),RA4
    (Imeter sense) as inputs and RA3(LED2),RA5(LED1)
  TRISC = 0b00000110;     //RC1(Reset Button),RC2(Vmeter sense)
  VRCON = 0;          // Turn off voltage reference
  CMCON0 = 0x07;        // Turn off Comparators
  /**** ADC ****/
  ANSEL =  0b00000000;      // Analouge select bits ANS1, ANS 2
  ADCON0 = 0b10010001;    // ADC config reg Right Justified, Vdd ref, AN4 POT 100
    selected, A/D on
  /**** INTERRUPT ****/
  RAIE=0;           // Port A interupt enabled.
  IOCA1 = 0;          //interupt on change of RA1 (pin 12)
  IOCA2 =0;         // Interrupt on change of portA pin11(RA2)
//  PIE1 = 0;
//  PIR1 = 0;         // and clear Timer Interrupt flag
  /**** TMR0 ****/
  T0CS = 0;           //internal clock
  T0SE = 0;         //rising edge
  T0IF = 0;           //clear flag
  T0IE = 1;         //enable timer0 interupt
  PSA  = 0;         //1:256 prescaler
  PS2  = 1;
  PS1  = 1;
  PS0  = 1;         //timer0 prescaler 1:256 (interupt 15Hz interupt every 65.53ms)
  TMR0 = 0;         //timer 0 preload for timer 0 register to 1MHz (0~255) (uses more
    clock)
  /**** TMR1 ****/
/*  TMR1H=0;          //clear timer before enabling
  TMR1L=0;          //
  T1CON=0x00;         //1:1 prescale interupt every 65.53ms
  TMR1IF=0;         //clear interupt flag TMR1IF=T1IF in earlier versions of hitech
    compiler
  TMR1IE=1;         //enable timer1 interupt. TMR1IE=T1IE in earlier versions of hitech
    compiler
  TMR1ON=1;         //timer1 OFF
*/  PEIE=1;           //Enable prepheral interupt
  GIE=1;            //Enable global interupt
  /**** PWM ****/
  PR2 = 0xFF;           // Set PWM period
  TMR2 = 0;         //clear timer2
  T2CON = 0x7F;       // TMR2 Prescaler 1:16, post scaler 1:16 (244Hz 4ms period)
    interupt occurs every 65ms
  TMR2IE=1;         //Timer2 interupt disabled CCPR1L doesn't change when enabled
```

```
151   CCP1CON=0b00001100;      //Single output mode, PWM mode all active high.
152   CCPR1L=0;          //Start at 50% D/C
153   /**** Initial ADC ****/
154   /*
155   ADON=1;//turn on ADC
156   GODONE=1;//start ADC
157   while(GODONE);//wait for ADC to finish
158   offset = 255*ADRESH + ADRESL;         //measure offset from initial adc reading
159   ADON=0;//turn off ADC
160   */
161 }
162
163 /********************************** interupt service routine
        ***************************************/
164 void interrupt ISR (void)
165 {
166   /********************************** Timer 0 ***************************************
        */
167   if (T0IF)                 //timer 0 interupt flag
168     {
169     /**********Timer 0 PWM**********/
170     if (pwmstatus == 1)                     //a variable to switch between pwm on off
        durations
171     {
172       //TMR0=dutyCycle;           //Timer ON period
173
174       pwmstatus = 0;            //clear pwm status flag
175       Imeter=0;             //set the pin high for pwm output
176       //T0IF=0;             //clear flag
177       TMR0=display;
178       T0IF=0;
179     }
180     else
181     {
182       //TMR0=255-dutyCycle;          //Timer OFF period
183       pwmstatus = 1;            //set pwm status flag
184       Imeter=1;             //set the pin low for pwm output
185       TMR0=255-display;
186       T0IF=0;               //clear flag
187     }
188
189   }
190   /********************************** Timer 1 ***************************************
        */
191   /*if(TMR1IF)                  //timer 1 interupt flag
192   {
193     TMR1IF=0;//reset timer 1 flag
194   }*/
195   /********************************** Timer 2 ***************************************
        */
196   if(TMR2IF)                  //timer 2 interupt flag
197   {
198     TMR2IF=0;
199
200     /********** Measure Vmeter Sense **********/
201 //    SELECT_VMSENSE;             //change ADC channel to Vmeter
202 //    SELECT_IMSENSE;
```

```
//     for(i=0;i<20;i++){asm("nop;");}        //wait for multiplexer to settle
/*     GODONE=1;                  //start ADC
     while(GODONE);                //wait for ADC to finish
//     if(display>8&&Imeter==1)
     if(CCPR1L>8&&Vmeter==1)          //if PWM on for some time more than 1bit and Vmeter
     is high
     {
       Vm=255*ADRESH + ADRESL;
       if(Vm>970){Vm+=1;}           //filter by slowing changes
       if(Vm<970){Vm-=1;}
     }
*/
     /********** Measure Imeter Sense **********/
/*     SELECT_IMSENSE;               //change ADC channel to Vmeter
     for(i=0;i<20;i++){asm("nop;");}      //wait for multiplexer to settle
     //ADON=1;                  //turn on ADC
     GODONE=1;                  //start ADC
     while(GODONE);                //wait for ADC to finish
     if(dutyCycle>8&&Imeter==1)        //if Imeter PWM on for some time more than 1bit and
      Imeter is high
     {
       Im=255*ADRESH + ADRESL;
       if(Im>970){Im+=1;}           //filter by slowing changes
       if(Im<970){Im-=1;}
     }
*/
     /********** Measure Vref **********/
     SELECT_VREF;                  //change ADC channel to Vref
     for(i=0;i<20;i++){asm("nop;");}      //wait for multiplexer to settle
     GODONE=1;                  //start ADC
     while(GODONE);                //wait for ADC to finish
     Vref = 255*ADRESH + ADRESL;
     Vdd=2500L*1024L/(Vref);          //measure Vdd from 2.5V reference using long int (in
     mV)

     /******** LED charge signals *******/
     if(Vdd>4200)                  //fully charged
     {
       LED1=0;
       LED2=1;
       LED3=0;
     }
     if(Vdd<=3400)                 //flat
     {
       LED1=1;
       LED2=0;
       LED3=0;
     }
     else                     //charging
     {
       LED1=0;
       LED2=0;
       LED3=1;
     }

     /********** Measure Vtemperature **********/
     SELECT_VTEMP;
```

```
257     for(i=0;i<20;i++){asm("nop;");}        //wait for multiplexer to settle
258     GODONE=1;
259     while(GODONE);
260     T = 255*ADRESH + ADRESL;
261     T = (long)Vdd*T/1024L;                  //convert from 10bits to millivolts
262     T = 10*T/179 - 23;                     //25C = 860mV
263 //    T = T/18 - 21;                       //25C = 840mV
264 //    T = T/19 - 21;                       //25C = 883mV
265 //    T = 10L*(T - 400L)/175;                 //840mV = 25C
266
267     /********** Measure Vopamp *********/
268     SELECT_OPAMP;                  //change ADC channel to opamp
269     for(i=0;i<20;i++){asm("nop;");}      //wait for multiplexer to settle
270     GODONE=1;                     //start ADC
271     while(GODONE);                   //wait for ADC to finish
272     Vdiode = 255*ADRESH + ADRESL;
273     Vdiode = Vdd - (long)Vdd*Vdiode/1024L;     //convert from 10bits to millivolts and
        remove offset voltage
274     Vdiode = (1822L*Vdiode + 15)/1000L;      //Vdiode in the scale of 10*1mV
275
276     /********** Output VMETER PWM *********/
277 //non Vmeter current sense
278     Vm = (Vdd-3000L)/12L;            //voltage offset to suit voltage range and calculate
        current in uA that should flow through Vmeter
279     FSD = 256L*RM/Vdd;               //PWM required to reach FSD in present Vdd
280     display = FSD*Vm/100L;             //calculate pwm duty cycle from Vmeter current
281     if(display>255){display=255;}      //put display with in CCP1RL range of 0~255.
282     if(display<0){display=0;}
283 //    CCPR1L=display;
284
285     /********** Output IMETER PWM *********/
286     Vth = 10000L*(T+273L)/Q;                  //Vth calculation 10*1mV
287     lnId = (long)10L*T - 1509L + 100L*Vdiode/Vth;      //lnId 2 decimal places (multiplied
         by 100)
288
289 //handeling high diode current at high temperatures
290 /*    if(lnId>0)//if above 1A
291     {
292       IRs = lookUpVd(lnId);
293       lnId = (long)10L*T - 1509L + 100L*(Vdiode-IRs)/Vth;      //lnId 2 decimal places (
        multiplied by 100)
294     }
295 */
296     dutyCycle = 1086L*(lnId + 691L)/1000L;          //convert lnId to 0~1000 scale
297     dutyCycle = (long)FSD*dutyCycle/1000L;          //convert lnId to FSD
298     if(dutyCycle>255){dutyCycle=255;}           //put dutyCycle with in range of 0~255.
299     if(dutyCycle<0){dutyCycle=0;}
300
301     CCPR1L=dutyCycle;
302
303     SELECT_VMSENSE;
304   }
305 }
306
307 /********************************** main function
        ***************************************/
308 void main()
```

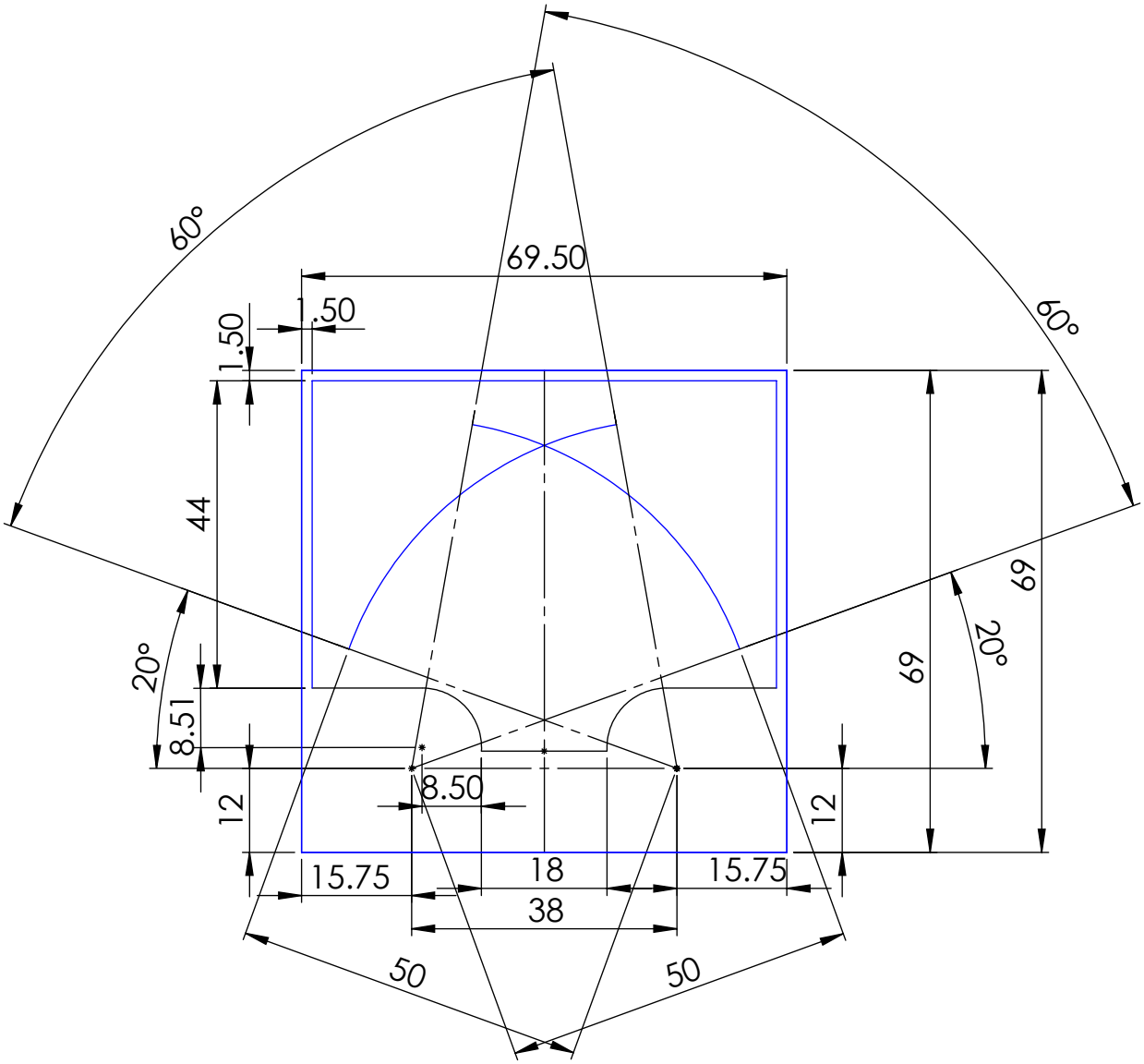
```
{
  initialise();

  while(1)
  {
  }
}
```

End of Listing 3: Firmware main.c

Appendix D

D.1 Custom meter panel Drawings



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBUR AND BREAK SHARP EDGES	All numbered dimensions in mm / REVISION

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A			

MATERIAL:

WEIGHT:

TITLE:

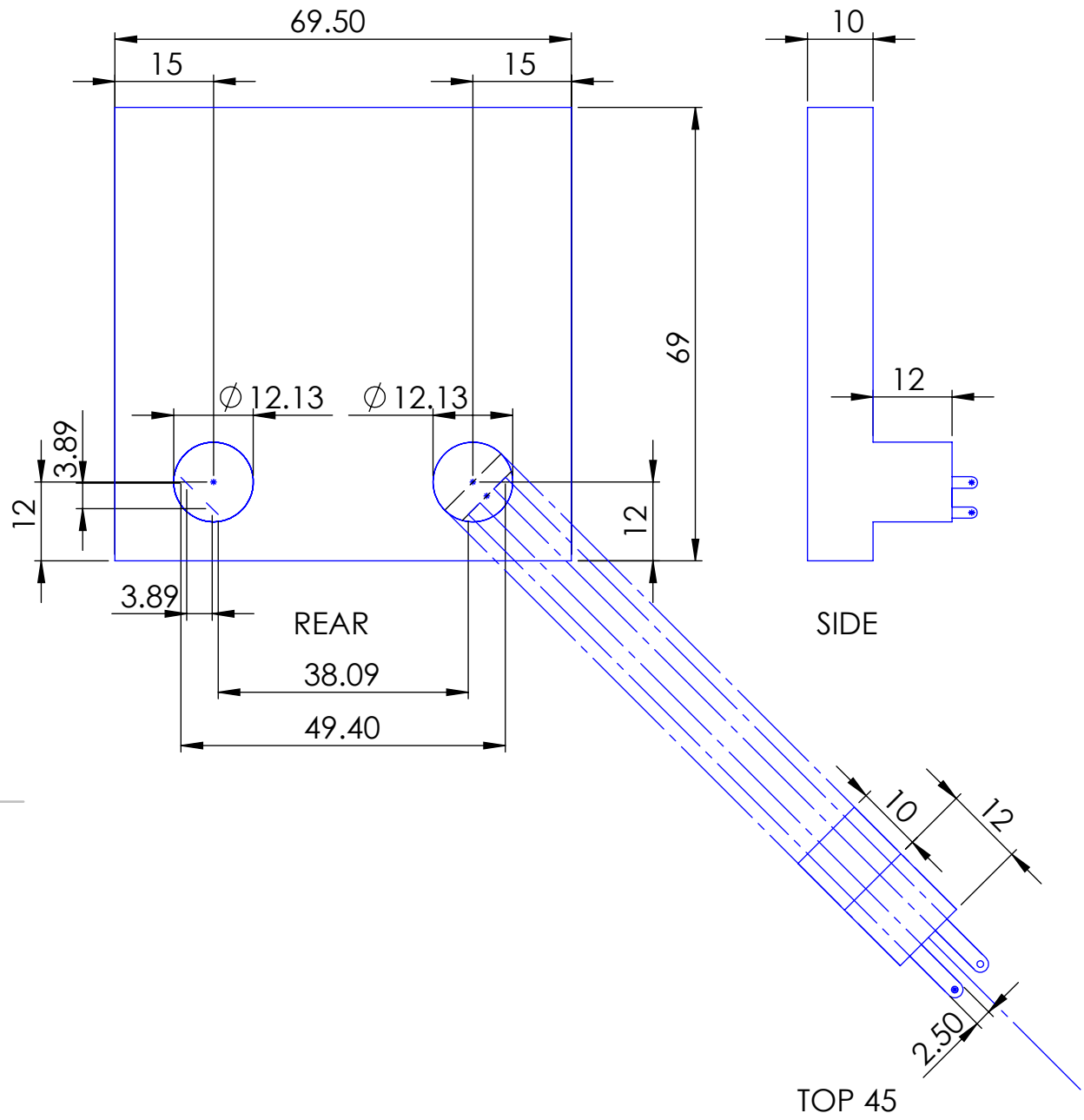
Front Meter Dimensions

DWG NO. Version 3

A4

SCALE:1:1

SHEET 4 OF 6



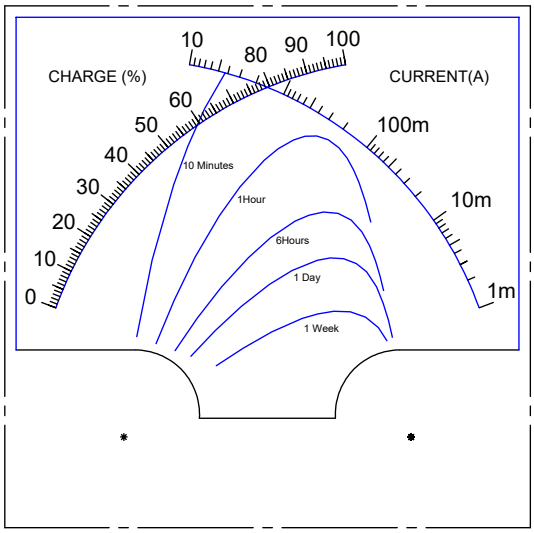
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	All numbered dimensions in mm	REVISION	

	NAME	SIGNATURE	DATE			
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A						

TITLE:

Back Of Meter Dimensions

MATERIAL:	DWG NO.	A4	
WEIGHT:	SCALE:1:1	SHEET 5 OF 6	



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBUR AND BREAK SHARP EDGES	1:1 Scale A4 Drawing	REVISION	
	NAME	SIGNATURE	DATE			
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:		
				WEIGHT:		

TITLE:

Printable Custom Panel To Scale

DWG NO. Version 3

A4

SCALE:1:1

SHEET 6 OF 6

References

[1] Jonathan Scott, "Accurate Low-Cost Expanded-Scale Analog Voltmeter with Novel Charge/Discharge Indication for Monitoring Single Lithium-Ion Cells", University of Waikato Research Commons, 2016, hdl:10289/10003.

[2] Martin Coleman, Chi Kwan Lee, Chunbo Zhu, and William Gerard Hurley, "State-of-Charge Determination From EMF Voltage Estimation: Using Impedance, Terminal Voltage, and Current for Lead-Acid and Lithium-Ion Batteries", IEEE Transactions on industrial electronics, VOL. 54, NO. 5, October 2007.

[3] Hung-Cheng Chen, Shuo-Rong Chou, Hong-Chou Chen, Shing-Lih Wu, and Liang-Rui Chen, "Fast Estimation of State of Charge for Lithium-ion Battery" , IEEE International Symposium on Computer, Consumer and Control, 2014

[4] F. Baronti, G. Fantechi, L. Fanucci, E. Leonardi, R. Roncella, R. Saletti, S. Saponara, "State of Charge Estimation Enhancing of Lithium batteries through a Temperature-Dependent Cell Model", IEEE Applied Electronics (AE) 2011 International Conference, 7-8 Sept 2011, ISSN:1803-7232

[5] Pop, V., Bergveld, H.J., Danilov, D., Regtien, P.P.L., Notten, P.H.L., "Battery Management Systems - Accurate State-of-Charge Indication for Battery-Powered Applications", Philips Research Book Series Volume 9, 2008. pp38–69

[6] Gabriela E. Bunea, Karen E. Wilson, Yevgeny Meydbray, Matthew P. Campbell and Denis M. De Ceuster SunPower Corporation, "LOW LIGHT PERFORMANCE OF MONO-CRYSTALLINE SILICON SOLAR CELLS", IEEE Conference, 7-12 May 2006, DOI: 10.1109/WCPEC.2006.279655

[7] G. Gler, . Gll, . Karata, . F. Bakkalolu,"Analysis of the series resistance and interface state densities in metal semiconductor structures ", Inter-

national Conference On Superconductivity and Magnetism (ICSM2008), 2009, DOI:10.1088/1742-6596/153/1/012054

[8] "LTC2941-1 1A I2C Battery Gas Gauge with Internal Sense Resistor", 29411f, Linear Technology Corporation, 2010.