High Voltage Conductor Identifier

EE465 – Senior Design II

Senior Design Project

Final Design Report

Cole Hennies

Cody Decker

Department of Electrical Engineering and Computer Science

South Dakota State University

May 3, 2023

1. Introduction ... 3
1.1 Background ... 3
1.2 Problem Motivation ... 3
1.3 Project Description ... 4
2. Scope ... 4
2.1 In Scope ... 4
2.2 Out of Scope ... 4
3. Design Requirements ... 4
3.1 Objectives, Requirements, Specifications ... 4
3.2 Constraints ... 5
4. Design/Technical Solution ... 6
4.1 Overview ... 6
4.2 Hardware Implementation ... 6
4.2.1 Submodule Design ... 7
4.2.1.1 Battery Charging ... 7
4.2.1.2 Voltage Regulators ... 8
4.2.1.3 Microcontroller ... 10
4.2.1.4 Signal Processing ... 10
4.2.1.4.1 Current Detection ... 10
4.2.1.4.2 Rogowski Integrator ... 11
4.2.1.4.3 RMS-DC Converter ... 11
4.2.1.4.4 Zero Crossing Detection ... 13
4.2.1.5 Cellular Module ... 14
4.2.1.6 GPS Module ... 15
4.2.2 PCB Layout ... 16
4.2.3 Case Design ... 19
4.3 Software Implementation ... 20
4.3.1 Peripheral Initialization ... 22
4.3.2 Operational Loop ... 22
5. Testing Procedures ... 23
5.1 Specification Testing ... 23
6. Design Budget and Expenditures ... 26

7. Conclusions 27
8. References 28
9. Appendix 28

1. Introduction

1.1 Background

Many power companies supply power to homes and businesses directly, which presents challenges with maintenance and new construction. Maintenance includes splicing damaged wire, replacing and testing equipment. All these aspects of power distribution system maintenance can require that a part of the distribution grid be taken offline. Field maintenance crews are unable to accurately identify phases and parallel circuits of underground conductors without disconnecting power, which costs power companies time and money. If field personnel are instead provided a way to properly identify each line while live, they can perform maintenance with reduced downtime.

Underground distribution lines may run up to 2,500 ft between above-ground access points. Because of the long distance, a wireless method of communication must be implemented. Inaccurately identifying energized lines also poses a safety hazard for technicians and engineers working in the field. As such, the method of measurement must be non-intrusive and safe.

The objective is to design a stand-alone portable measurement system that a single individual can use, in real time, to accurately identify and differentiate between energized phases and parallel feeds. A High Voltage Distribution Phase Identifier will provide accurate identification of energized lines, thus increasing safety and efficiency for the user.

The SDSU CPSS team is proposing a system that improves crew safety, reduces downtime for utility customers, increases productivity during routine or emergency maintenance, thus saving both the utility and its customers money. Several systems currently exist to determine phase while live. The main competitor devices for this project, the Phase Trakker Jr and SPI-III, both use non-intrusive measurement devices to obtain phase data of live lines. However, their prices range between $2000 and $7200. One of the goals of this project is to develop two devices that in total cost less than $1000.

The sponsor of this project is the Electric Power Research Institute GridEd (EPRI GridEd). They provided us with $5000 total to spend on the project with a $1000 budget going toward the development of the actual devices and $4000 being spent on test equipment. The customer and advisor for this project is Dr. Steven Hietpas, PE (EE).

1.2 Problem Motivation

This project is currently in its 3rd design phase. In Phase I (2019-2020 academic year), the team was comprised of three members: Zach Kirsch, Erik Ode, and Obinna Ezugu. They were challenged with the Covid-19 global pandemic, which curtailed the prototype development in March of 2019.

Although the Phase I team faced unforeseen setbacks due to Covid-19, they accomplished a design where data of phase and current magnitude could be measured non-intrusively and stored on a SD card for post analysis. In summary, the Phase I group successfully developed a:

1. Non-intrusive measuring method able to measure with relative accuracy both magnitude and phase.
2. Time-stamp method to synchronize stored data from both the master and slave units using off-the-shelf GPS receiver chip set.
3. Post analysis was implemented within a spreadsheet.

In phase II of the project, the scope was expanded to have a functional prototype, Fig. 1, which integrated the modules in Phase I, while also incorporating wireless RF communication [1]. It was capable of circuit and phase identification on three phase circuits with energized lines. However, RF system selected limited the design to line-of-sight communication.

1.3 Project Description

Phase III will focus on alternative implementations of the design from the second phase of the project. Most notably will be exploring other options for implementation of the front-end measurement circuitry as well as different methods of communicating wirelessly between the two devices. The goal of these design changes is based on increasing communication capabilities and reduce costs while improving the accuracy of phase and circuit identification. In Table 1, two devices that determine phase while the line is energized are shown. The high prices of these devices are the main motivation for this project. The two devices have been used in industry in the past, however, the previous sponsor felt as though the devices did not work very well and are looking for a better solution. The goal of this project is to make a device that can identify line current phases and parallel conductors of the same phase at a cheaper price point.

2. Scope

2.1 In Scope

1. Design a phase identifier
2. Design a device that can discern between parallel conductors
3. Integration of a wireless communication system
4. Design a mirrored user interface on both devices
5. Implementation of a portable power source.

2.2 Out of Scope

1. Storage and/or ability to download for archival purposes
2. 3-D printing of prototype case

3. Design Requirements

3.1 Objectives, Requirements, Specifications

1. Objective: Distinguish between parallel lines of same phase, while energized
   - 1.1 Requirement: Distinguish between parallel lines of the same phase
     - 1.1.1 Specification: The magnitude of the current shall be within 50% of the minimum differential between parallel lines.

2. Objective: Identify individual line current phases
   - 2.1 Requirement: Identify individual line current phases
     - 2.1.1 Specification: Phase values shall be within +/- 10 degrees

3. Objective: Display Information to user

3.1 Requirement: Display current, phase, and line-phase-match indicator to user on both devices
    3.1.1 Specification: Information display must be readable from 5 feet away

4. Objective: Device shall be portable
    4.1 Requirement: Device shall be powered by a battery
        4.1.1 Specification: Device shall operate for a minimum of 2 hours
    4.2 Requirement: Device shall be hand-held
        4.2.1 Specification: device shall be less than 210 x 100 x 60 mm
    4.3 Requirement: Device shall be hand transportable
        4.3.1 Specification: device shall weigh less than 8 pounds

5. Objective: System shall communicate wirelessly
    5.1 Requirement: Device shall communicate to cellular network
        5.1.1 Specification: Device shall connect to a cell tower from a minimum distance of 2 miles

6. Objective: Meet Applicable Safety Standards
    6.1 Requirement: FCC title 47 standards [4]
        6.1.1 Specification: Shall adhere to FCC title 47 standards regarding telecommunications
    6.2 Requirement: FAA 49 CFR175.10 standards [5]
        6.2.1 Specification: Shall adhere to FAA 49 CFR 175.10 standards regarding transportation of batteries on airlines

3.2 Constraints

The customer has stated the following constraints:

1. Design
    1.1 Measurements are non-intrusive.
    1.2 Galvanic connection to conductors cannot be made.
2. All components must be available from a minimum of two OEM vendors.

4. Design/Technical Solution

4.1 Overview

To solve the problem, two identical devices that are portable, handheld, and communicate wirelessly were designed. *Figure 1* shows the general overview of how both devices will be communicating with each other. Both devices will be synchronized with each other using a GPS chipset. This is necessary to measure the phase as well as getting the precise timestamp as to when each measurement is taken. The devices will communicate with each other using an application peripheral interface (API) over the cellular network. Each device will be equipped with a flexible current transducer known as a Rogowski coil which outputs a signal proportional to the differential current flowing through the conductor it is wrapped around. This sinusoid is then passed to a zero-crossing detector circuit for phase identification. This signal is also used to determine the current magnitude using an RMS-DC converter by converting the sinusoid into a DC value that the microcontroller can use.

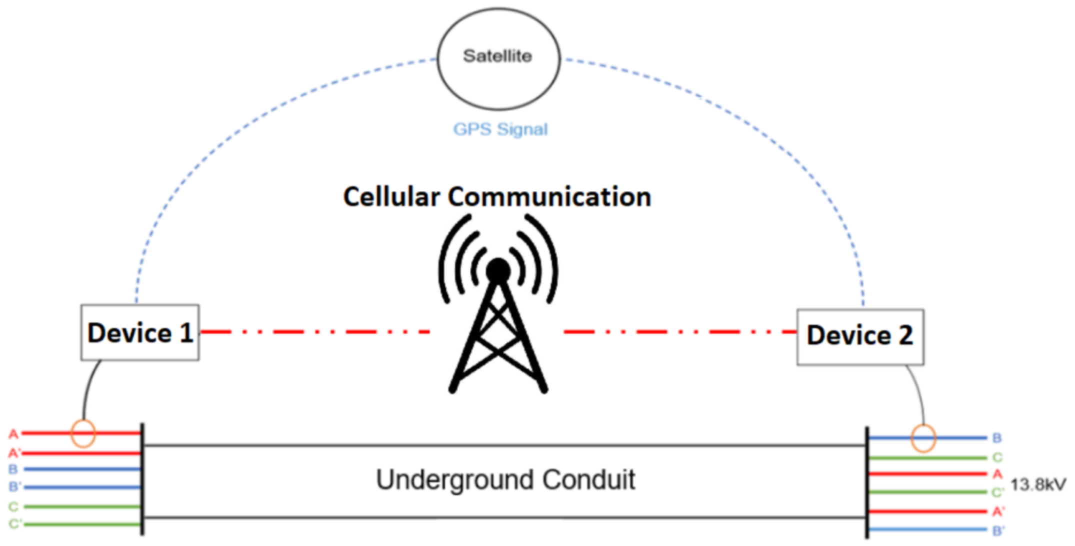


Figure 1: Overview of Proposed Solution

4.2 Hardware Implementation

The primary design consists of 6 main subsystems per device. These subsystems include battery charging, LCD display, microcontroller, signal processing, cellular module, GPS (global positioning system) module, and power supplies. The high-level block diagram can be seen in Figure *2*. Each individual device makes a non-galvanic connection to a prospective conductor to be identified via a current measuring device, which outputs a scaled sinusoid of the waveform running through the conductor. This waveform is processed in the zero crossing and current magnitude (signal processing subsystem). The GPS module, microcontroller, and cellular module all work together to process the zero crossing timings and magnitude and communicate the information between the devices before it is finally displayed to the user via the LCD display.

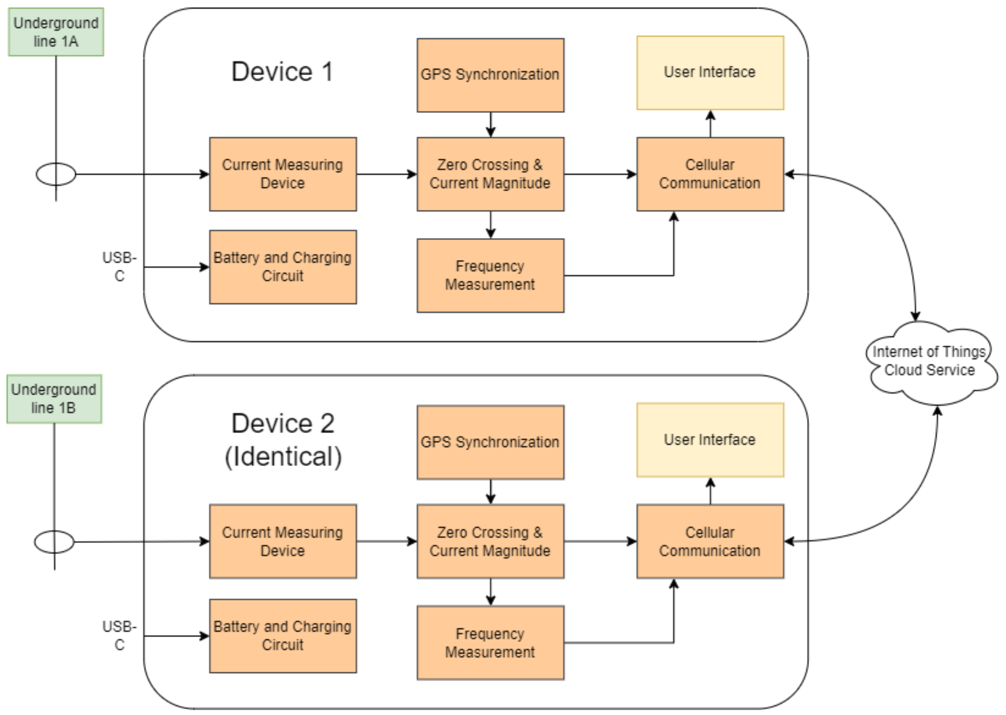


Figure 2: High Level Block Diagram

4.2.1 Submodule Design

The PCB layout for both devices is completely identical. The board contains all subsystems except the cellular module soldered directly onto the board. The cellular module circuitry is instead connected to the other subsystems through a development board and Arduino header.

4.2.1.1 Battery Charging

The entire device is powered by a single 3.7V lithium-ion battery. A single lithium-ion battery was chosen as opposed to a proprietary power-tool type battery or standard alkaline batteries to cut down on weight and size of the system (increased portability) while still including the ability for charging. For battery charging, it was decided that a USB-C female connector would be the best choice because of the widespread use of USB-C charging. In addition to the USB-C port, a battery charging IC was also integrated into the PCB design with two LED indicators. The charging IC that was used was the BQ24072. This IC was chosen after doing some research on charging IC's because it can charge at 0.5A and has the feature of having USB power passthrough meaning the device can be powered straight off of USB power. External resistors control the current flow into the battery. For the LED indicators, one LED indicates if the battery is charging and the other indicates if USB power is detected. *Figure 3* is the entire design for the battery charging circuitry and USB-C connector. The approximate power budget for the entire device can be seen in *Table 1*. This includes the approximate theoretical duration of the battery to meet the 2-hour specification with the components that draw the most power.

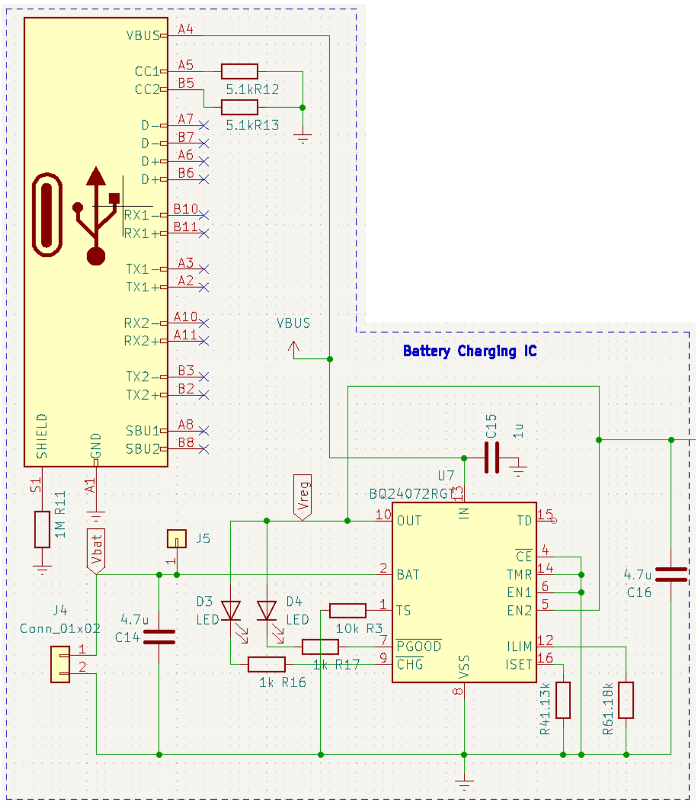


Figure 3: Battery Charging Circuit

Table 1: Power Budget Approximation

Component	Description	Power (W)
TL084	Quad Op-Amp	0.0186
LTC1968	RMS-DC Converter	0.015
Ublox MAX-8C	GPS	0.0594
TAOGLAS Antenna	GPS Antenna	0.0396
20x4 LCD	LCD Screen	3.75
Botletics SIM7000	Cellular Module	0.3828
Total		4.2654
Battery Pack	Capacity (Wh)	Duration (h)
Adafruit 2.5Ah battery	9.25	2.17

4.2.1.2 Voltage Regulators

There are three voltage regulators that are needed for the PCB. The first is the +5V regulator to power the LCD Display as well as the 3V regulator and the -5V regulator. The +5V regulator that was chosen was the TPS613222ADBVR boost regulator. This was to increase the voltage of the battery to a regulated +5V. This IC was chosen because it has a higher than 90% efficiency, has thermal shutdown protection, and comes in a small package. It also only requires only 3 external components consisting of one inductor and two capacitors. The +5V regulator design can be seen in *Figure 4*.

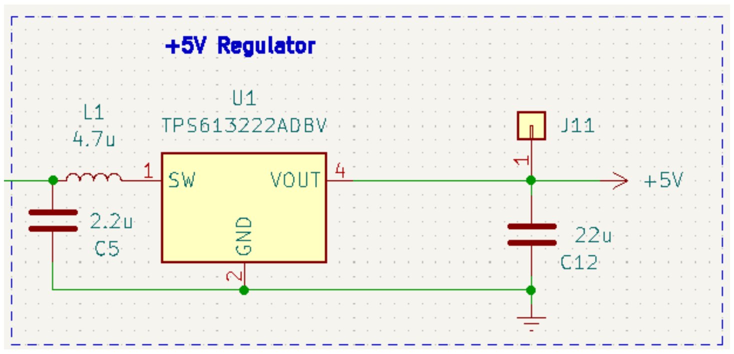


Figure 4: +5V Regulator

The next voltage regulator that is needed is the 3.3V regulator to power the microcontroller and GPS. The 3.3V regulator that was chosen was the AP7361C-33ER linear regulator. The input to the regulator was coming from the output of the +5V boost converter. A linear regulator was chosen for this module because there will not be significant power losses going from 5V to 3.3V. Any linear regulator that outputs 3.3V would suffice for this power supply, but this one was chosen because it comes in a small footprint and only requires 2 capacitors. *Figure 5* shows the 3.3V regulator design.

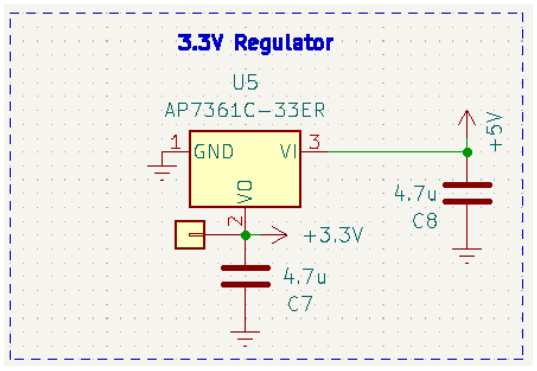


Figure 5: 3.3V Regulator

The final voltage regulator that was needed was the -5V regulator to power the op-amp as well as the RMS-DC converter. Because the op-amp and RMS-DC converter do not draw a lot of power, an inverting charge pump IC was used. The IC that was used was the MAX1673esa inverting charge pump. The inverting charge pump was chosen for the -5V regulators because there would not be large amount of current draw and the inverting charge pump is a good choice for this application. The output of the charge pump is controlled by two external resistance using feedback. The equation for the output voltage can be seen in (1). When supplying +5V to the inverting charge pump, two 100kΩ resistors can be used to achieve -5V at the output. The -5V regulator design can be seen in Figure 6.

$$V_{out} = -V_{REF}\left(\frac{R2}{R1}\right) \tag{1}$$

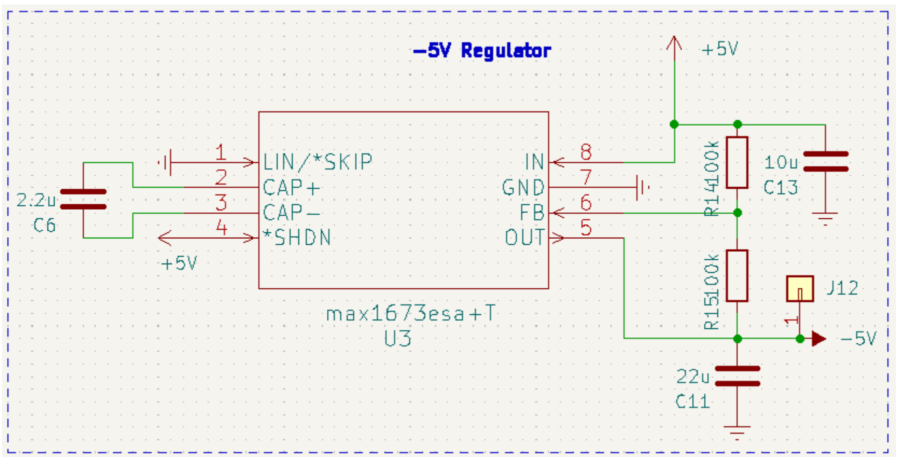


Figure 6: -5V Regulator

4.2.1.3 Microcontroller

The hub for all the communications and calculations is the microcontroller. The microcontroller chosen for this project was the Texas Instruments TM4C1233H6PM. This microcontroller was chosen because we were familiar with using other TI microcontrollers as well as using the programming software called Keil. It was also chosen because it has enough storage for the expected size of the program as well as enough GPIO's, UART's, I2C, and ADC's.

4.2.1.4 Signal Processing

The signal processing section consists of everything from connection of the current transducer to the signal inputs to the microcontroller. This section consists of 4 main subsections including current detection, integration, RMS-DC converter, and zero-crossing detection.

4.2.1.4.1 Current Detection

The first component of the signal processing is the Rogowski Coil. The Rogowski coil is a device that is used for measuring AC currents and is also known as a flexible current transducer (CT). The Rogowski coil physically is a toroid of wire that has a voltage output that is proportional to the rate of change (derivative) of the current of the conductor that the CT is wrapped around. To get the magnitude and the current waveform of the conductor, an integrator circuit is required. *Figure 7* depicts how the Rogowski coil works in conjunction with a conductor. For ease of installing the Rogowski coil, a female BNC jack is connected to the end of the wire to connect to the device.

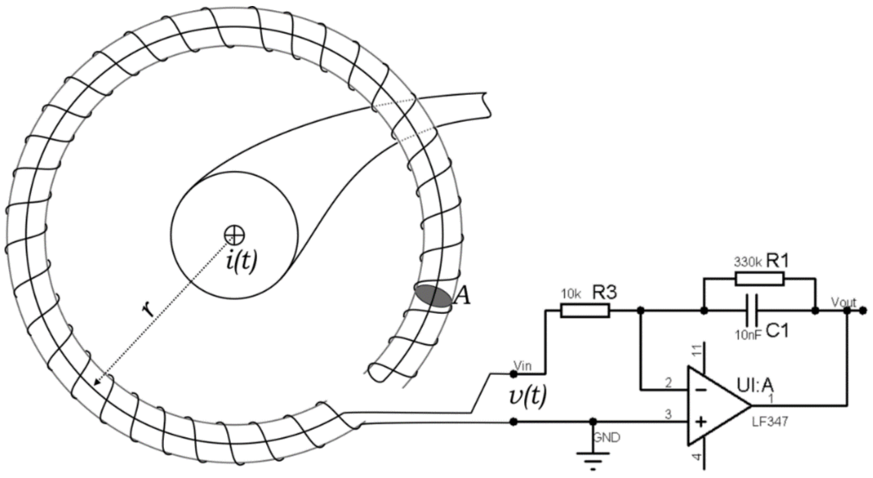


Figure 7: Rogowski Coil Schematic

4.2.1.4.2 Rogowski Integrator

To get the current waveform from the Rogowski coil, an integrator circuit is required. Texas Instruments has a 47-page document detailing how to create the best integrator circuit for the Rogowski coil depending on the application and this document was used for the basis of the integrator design [2]. For this design, emphasis on precision measurement was essential. The integrator that was used was the inverting integrator that consists of two resistors and a capacitor. The component values and design of the integrator can be seen in *Figure 8*. The component values were chosen to achieve a gain of 9V/V. This was chosen specifically because the RMS-DC converter has a very limited input voltage range, and the gain of this front stage could not be any higher without exceeding the limits of the RMS-DC converter. During initial testing, it was found that there would be a DC offset at the output of the op-amp due to leaky capacitors in the integrator. Because of this, an AC coupling capacitor was placed on the output of the integrator along with a large resistor.

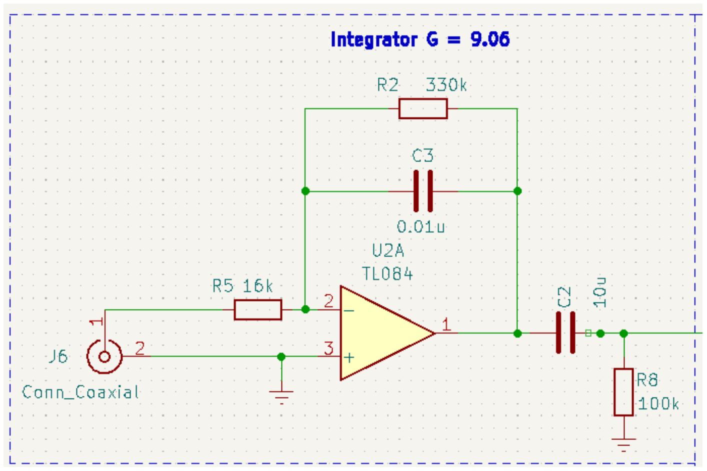


Figure 8: Inverting Integrator

4.2.1.4.3 RMS-DC Converter

The RMS-DC converter is used to get the RMS value of the current flowing through the high voltage conductor. The RMS-DC converter used was the LTC1968 because it is one of the only options for an RMS-DC converter that was available. This device has a limited input voltage of 1V and supply voltage of a maximum of 6V. Two Zener diodes were used to regulate the device at +/- 2.4V. This was done to improve the linearity of the device and followed TI's datasheet. The output of the RMS-DC converter was an amplifier with a gain of 3V/V. This was done to get the maximum resolution out of the ADC of the microcontroller. *Figure 9* shows the design of the RMS-DC converter and gain stage.

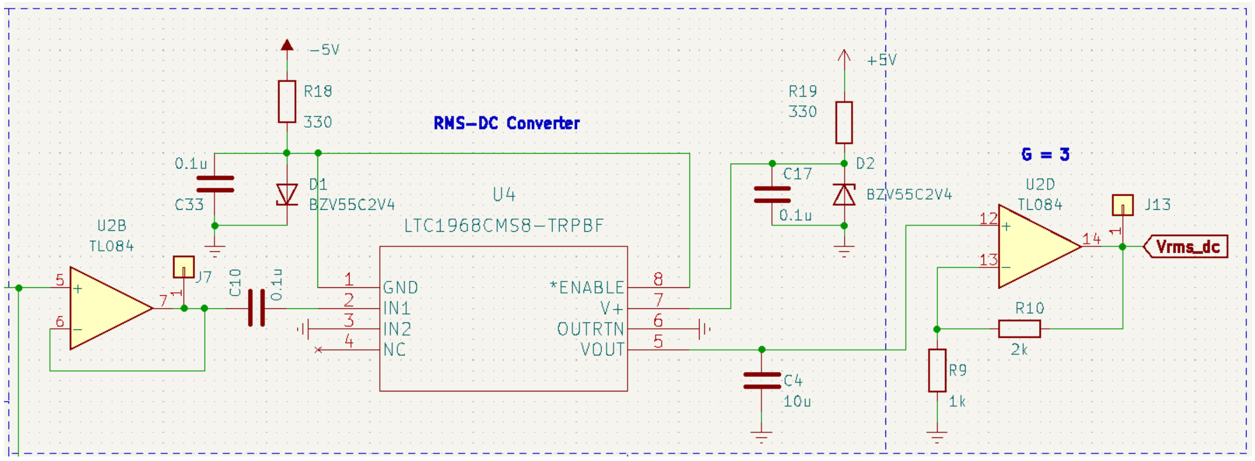


Figure 9: RMS-DC Converter Design

After designing the RMS-DC converter, the linearity of the converter was characterized by connecting the Rogowski coil to a conductor on the Automatic Load Bank (ALB) and increasing the load slowly and recording the output voltage as well as measuring the line current. *Table 2* shows the comparison of the RMS-DC voltage to the line current. After this, *Figure 10* was created using the data and the R^2 value was found along with the slope of the line. This would later be used by the microcontroller to display the actual line current using the DC voltage of the RMS-DC converter. The R^2 value was found to be 99.82% meaning that 99.82% of the variation in the y values are accounted for by the x values. The data is also very linear throughout the entire test.

Table 2: RMS-DC Converter Characterization Results

RMS-DC Voltage (V)	Line Current (A)
0.17	35
0.195	40
0.222	46.5
0.248	52.5
0.272	58
0.3	65
0.33	70
0.356	77
0.388	82.5
0.41	89
0.443	95
0.475	103.5
0.509	110
0.545	115
0.57	120

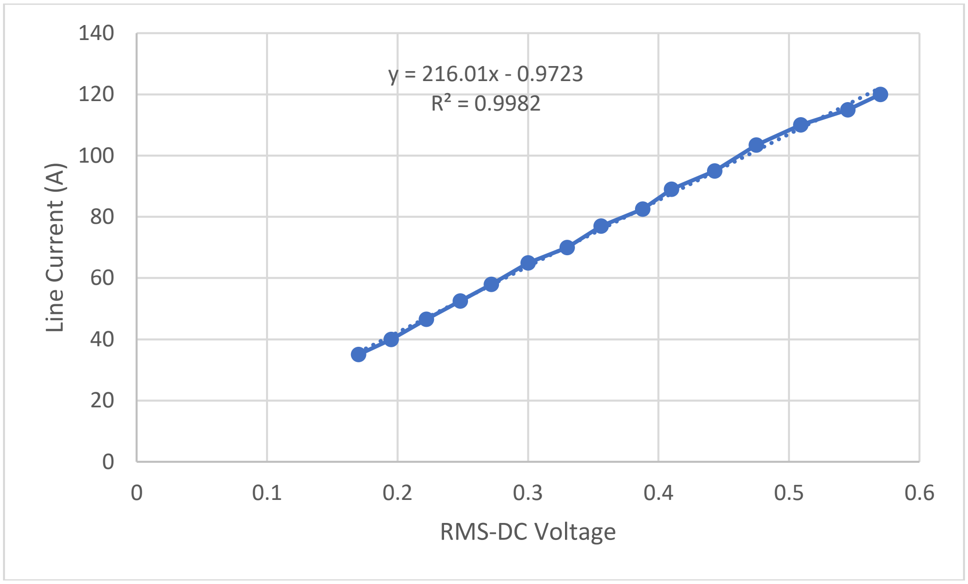


Figure 10: RMS-DC converter characterization plot

4.2.1.4.4 Zero Crossing Detection

The zero-crossing circuit is used to measure the phase of each conductor. This was done by taking the output from the integrator to an op-amp biased at 0V. This means that when the input goes above 0V, the output will go high and when it goes below 0V, the output will go low. To limit the output to 0-5V, a Schottky diode was used along with a current limiting resistor. The zero-crossing circuitry design can be seen in *Figure 11*. *Figure 12* shows the comparison of the input vs the output of the zero-crossing detection circuit. *Figure 13* shows the comparison of two different Rogowski coils on different phases and comparing the zero crossing results. Looking at the time difference between the two rising edges, the time is 5.56ms which is equivalent to a 120° phase shift.

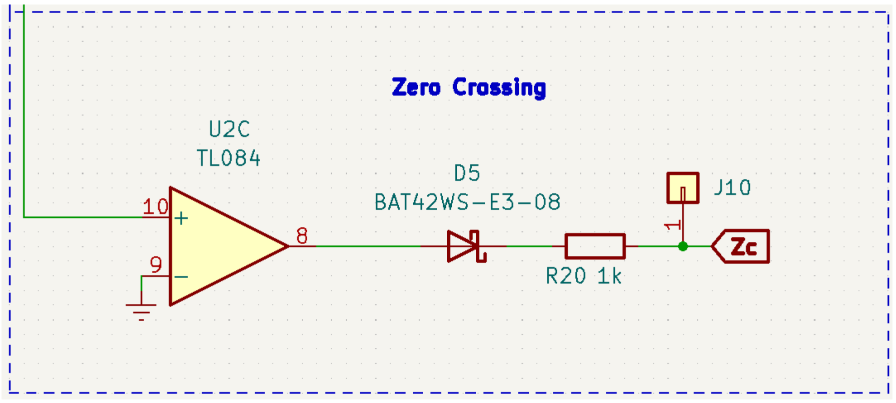


Figure 11: Zero Crossing Design

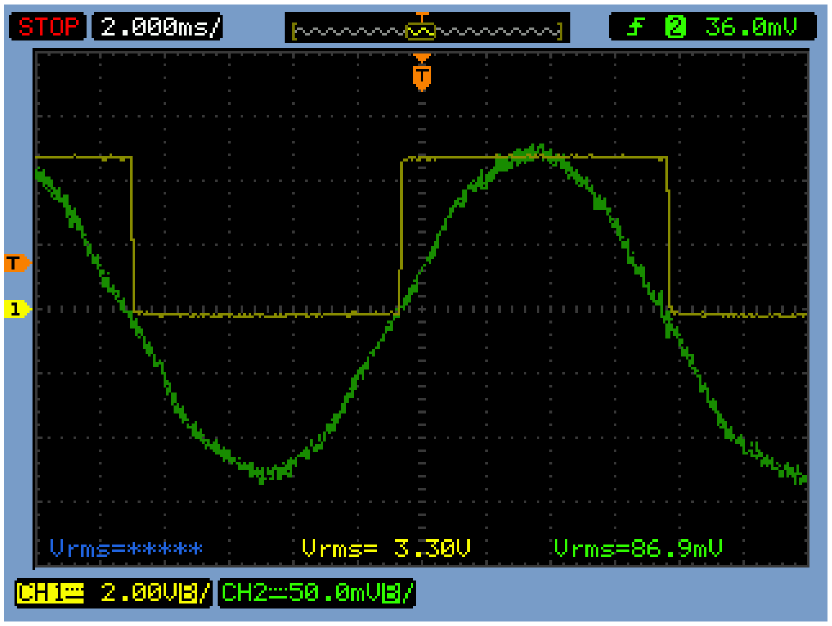


Figure 12: Zero crossing detection results for output from integrator (green) and output of zero crossing detection circuit (yellow)

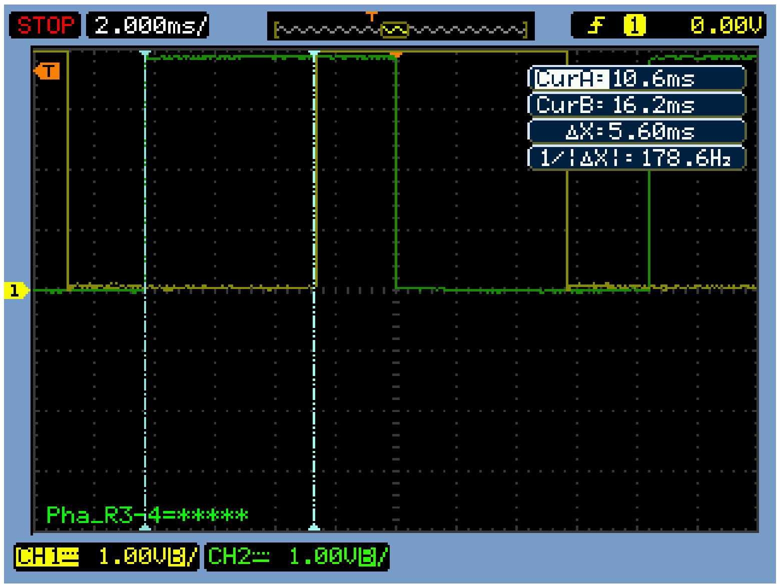


Figure 13: Comparison of two zero crossing devices on different phases shown

4.2.1.5 Cellular Module

For both devices to communicate with each other, it was decided that cellular communication would be one of the best options for this project. Each device has its own cellular communication module to send and receive data between one another. The cellular module that was chosen was the Botletics SIM7000. This module is based on the SIMCom SIM7000G chip set and the development board is shown in *Figure 14*. This module sits right on top of the PCB and connects using headers. Along with the module, a special SIM card is needed as well to connect to the network. The card chosen for this was the Hologram Global IOT SIM card. This allows the module to communicate and send and receive data.

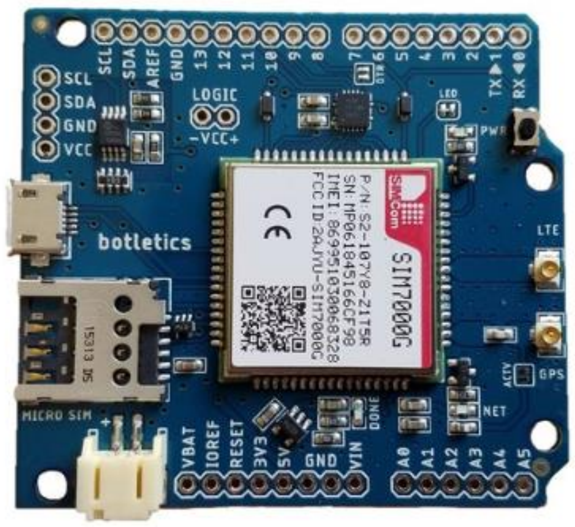


Figure 14: Cellular Communication Module

For both devices to communicate, an Application Peripheral Interface (API) was used. In this case, the API used was called Dweet.io and was free to use and easy to debug and analyze data. An API is a way to send and retrieve data from a server and is a very good way of sharing data between devices. This is a very common way in industry to efficiently share data between devices and to look at the status of the device on a network. Each device has its own specific identification number and posts its data to the API. The opposite device will then be able to get the other devices data that it had posted and perform the calculations to determine if the devices are in or out of phase and if they are on the same line.

4.2.1.6 GPS Module

Both devices are equipped with a Global Positioning System module (GPS) that can provide essential information such as Coordinated Universal Time (UTC) as well as a pulse per second (PPS). The PPS signal is the most important part of the entire project. The GPS chosen for this project was the U-blox Max-8C as it can provide UTC time as well as the PPS signal without being super expensive. The PPS signal is used to synchronize both devices to each other and this signal was found to be accurate to around 20ns. The GPS module layout can be seen in *Figure 15*. The GPS antenna is connected using a U.FL connector.

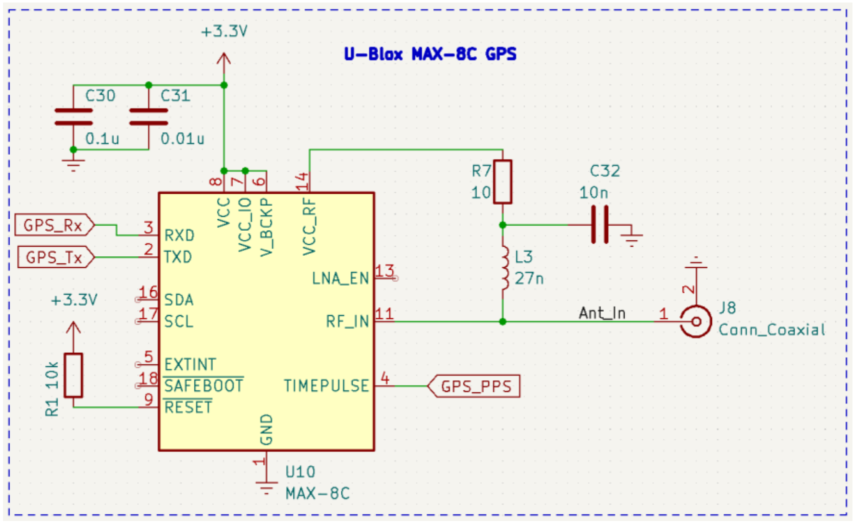


Figure 15: GPS Module Design

Along with the GPS module, a special active antenna was also purchased and attached to the U.FL connector. The GPS antenna that was chosen was the TAOGLAS AP.25F.07.0078A and can be seen in *Figure 16*. This is a two stage GPS active patch antenna with front-end Saw filter. With the two-stage low noise amplifier (LNA), the gain is 28dB which was the highest gain patch antenna readily available. The LNA's are what amplify the small GPS signal and the SAW filter is often used when Global System for Mobile Communication (GSM) transmitters are close to the GPS antenna. In this case, the GSM transmitter is the cellular communication module. Without a SAW filter, the active antenna can be oversaturated with the GSM signals. The RL filter connected to the coaxial connector is to supply the GPS antenna with 3.3V and the design followed the datasheet for the GPS module.

Figure 16: GPS Antenna

4.2.2 PCB Layout

The entire PCB layout and design was accomplished via KiCad. All the IC's were either imported or found using the KiCad library and the appropriate package and sizes for all the resistors, capacitors, and inductors can be selected. All resistors, capacitors, and inductors were chosen to be 0805 sizes for ease of soldering. *Figure 17* shows the entire schematic layout and wiring diagram for the project. Also included in this, that was not mentioned before, is the power switch represented by SW1. A SPST switch was soldered onto the board to turn it on and off.

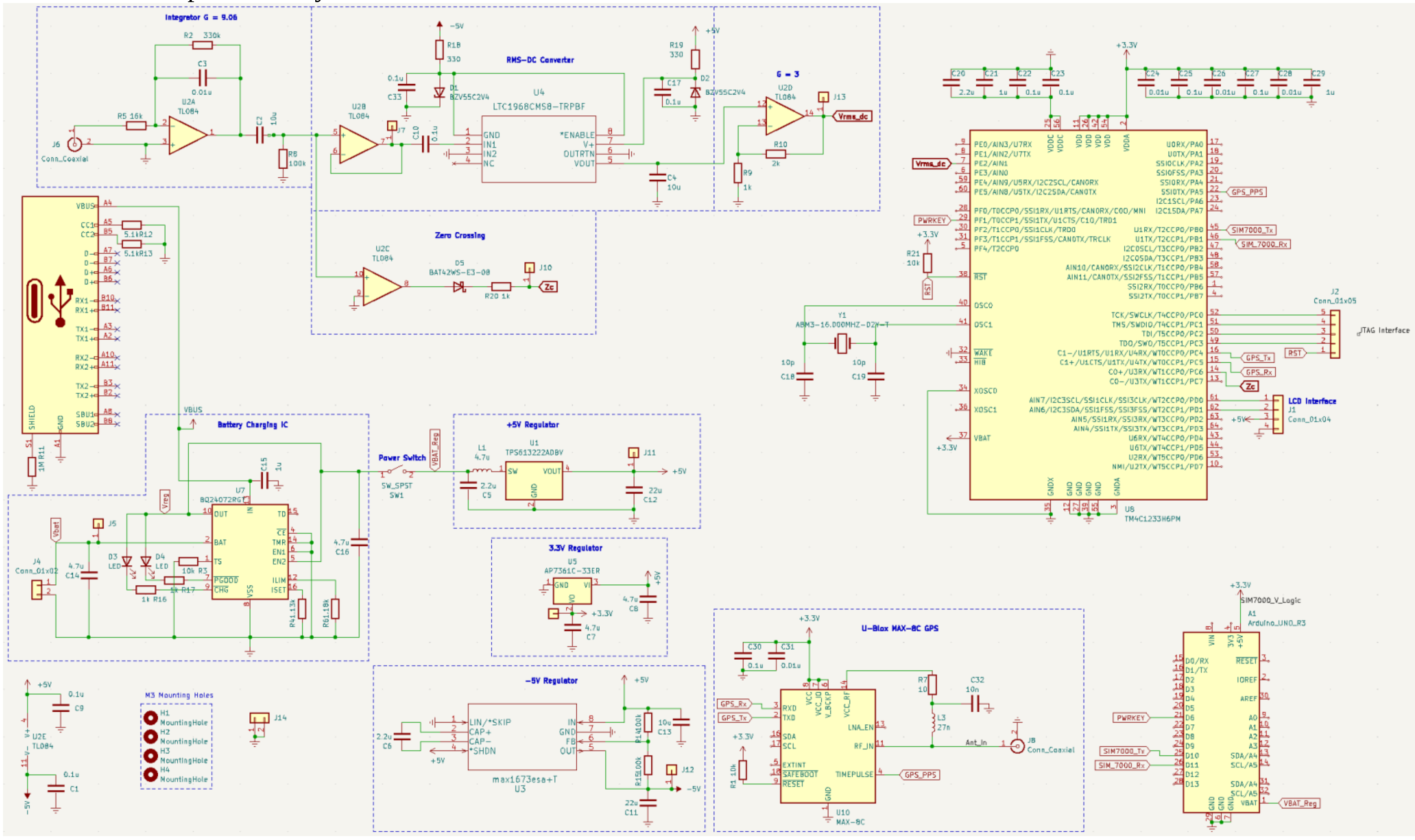


Figure 17: Overall Schematic Layout and Wiring Diagram

For the PCB layout and design of the actual board, all the parts and packages were imported and laid out grouping components accordingly. High frequency devices such as the microcontroller, signal processing, battery charging, and power regulation were kept far away from the GPS modules to reduce the interference. *Figure 18* shows the entire layout including all traces for the PCB. The board design consists of a two-layer board with a ground plane on bottom (blue) and ground/signal plane on top. All the components were laid out to reduce the length of the traces as much as possible. On of the most important parts of the PCB design was the layout for the GPS module. In the initial stages of testing, the signal strength of the GPS was very low. This problem was solved by following the GPS's datasheet for PCB layout design by adding an entire ground plane on top and creating lots of vias going to the bottom ground plane. In addition to this, a coplanar wave guide was also designed for the antenna coming from the U.FL connector. This is to ensure a 50Ω line to match the impedance of the coaxial line on the GPS antenna. KiCad has a built in tool used to calculate impedance of transmission lines and can be seen in *Figure 19*. This change alone made the GPS signal strength increase by around 10dB and was a huge improvement to the initial design. The LCD display is connected to the board using long wires and communicates using the I2C protocol. The LCD display chosen was a generic 20x4 display to display all the necessary information such as phase, frequency, current magnitude, and line match.

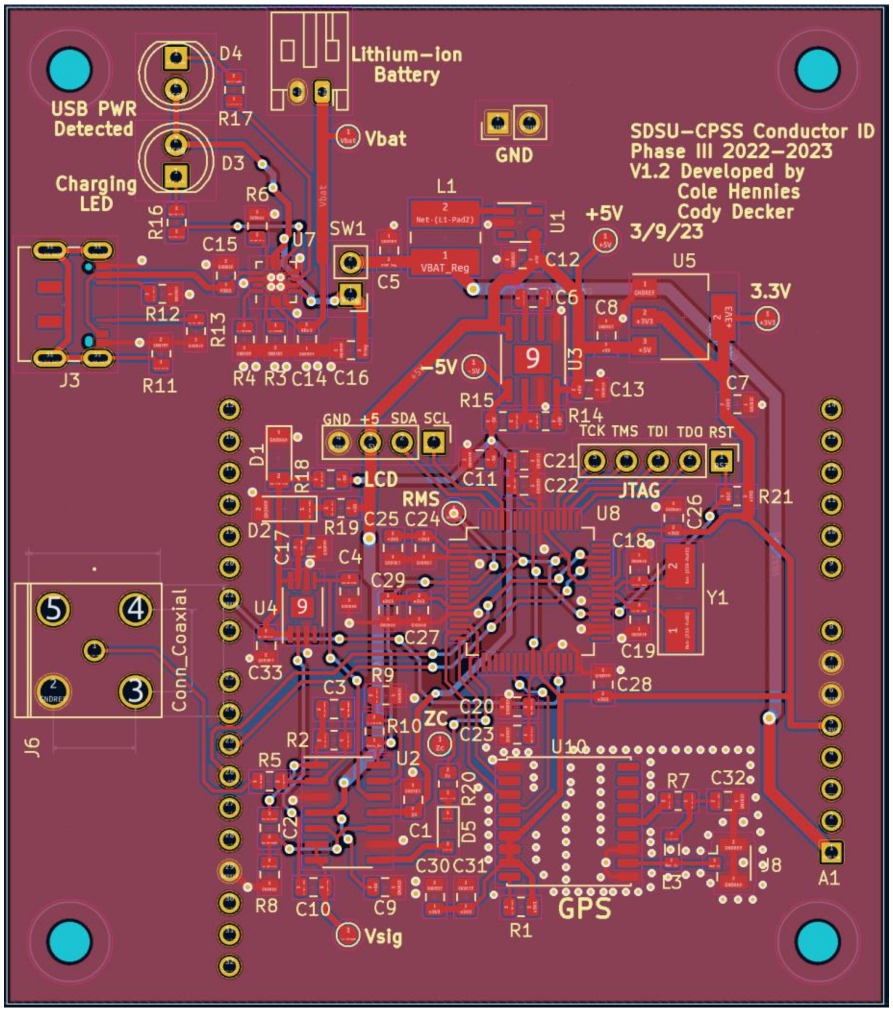
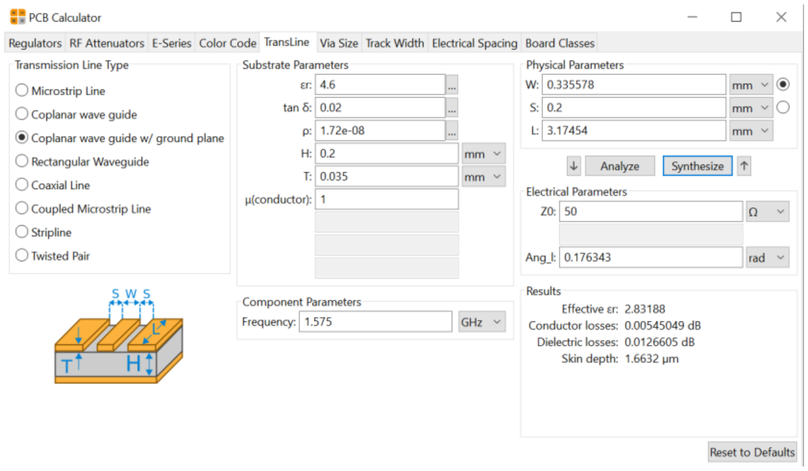


Figure 18: PCB Design and Layout



Transmission Line Type
- Microstrip Line
- Coplanar wave guide
- [✓ Coplanar wave guide w/ ground plane]
- Rectangular Waveguide
- Coaxial Line
- Coupled Microstrip Line
- Stripline
- Twisted Pair

Substrate Parameters

εr:	4.6	
tan δ:	0.02	
ρ:	1.72e-08	
H:	0.2	mm
T:	0.035	mm
μ(conductor):	1	

Component Parameters

Frequency:	1.575	GHz

Physical Parameters

W:	0.335578	mm
S:	0.2	mm
L:	3.17454	mm

Electrical Parameters

Z0:	50	Ω
Ang_l:	0.176343	rad

Results

Effective εr:	2.83188
Conductor losses:	0.00545049 dB
Dielectric losses:	0.0126605 dB
Skin depth:	1.6632 μm

Figure 19: GPS Antenna line calculation/design

4.2.3 Case Design

The entire PCB, cellular module, GPS, and LCD screen was designed to fit snugly inside a handheld plastic case designed by Hammond Mfg. (Part No. HM 3998-ND). The PCB was mounted using screws and standoffs inside the case. Holes were drilled for the USB-C charging, BNC connector for the Rogowski coil connection, as well as the power switch. The LCD display was mounted using cyanoacrylate and hot glue. The GPS antenna and cellular antenna were mounted to the case using double sided tape. One of the best improvements to GPS signal quality was moving the GPS antenna to the top of the cellular module. This increased our signal strength on average by around 20dB and improved the time to first fix to satellites down to 2 minutes maximum indoors compared to sometimes the tens of minutes that were required before. *Figure 20* shows the inside design for the case including mounting position for the battery, GPS antenna, PCB mounting, and cellular antenna. *Figure 21* shows the entire product assembled.

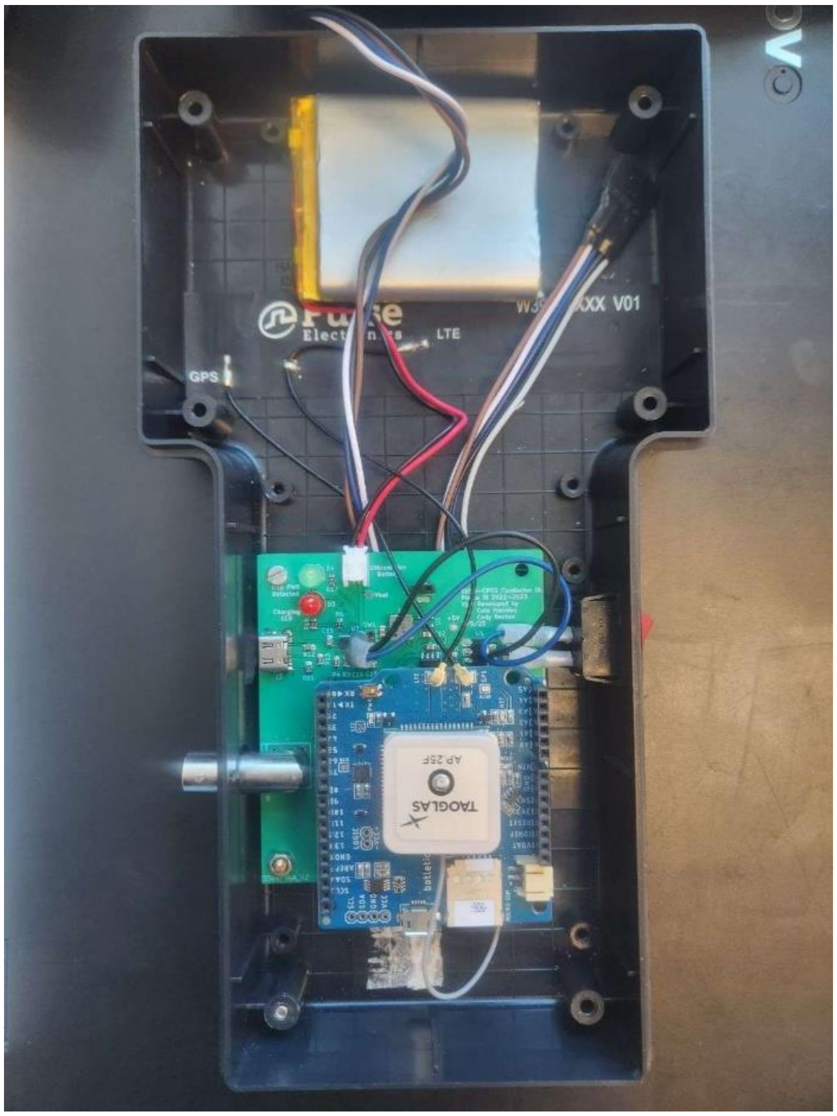


Figure 20: Inner Case Design

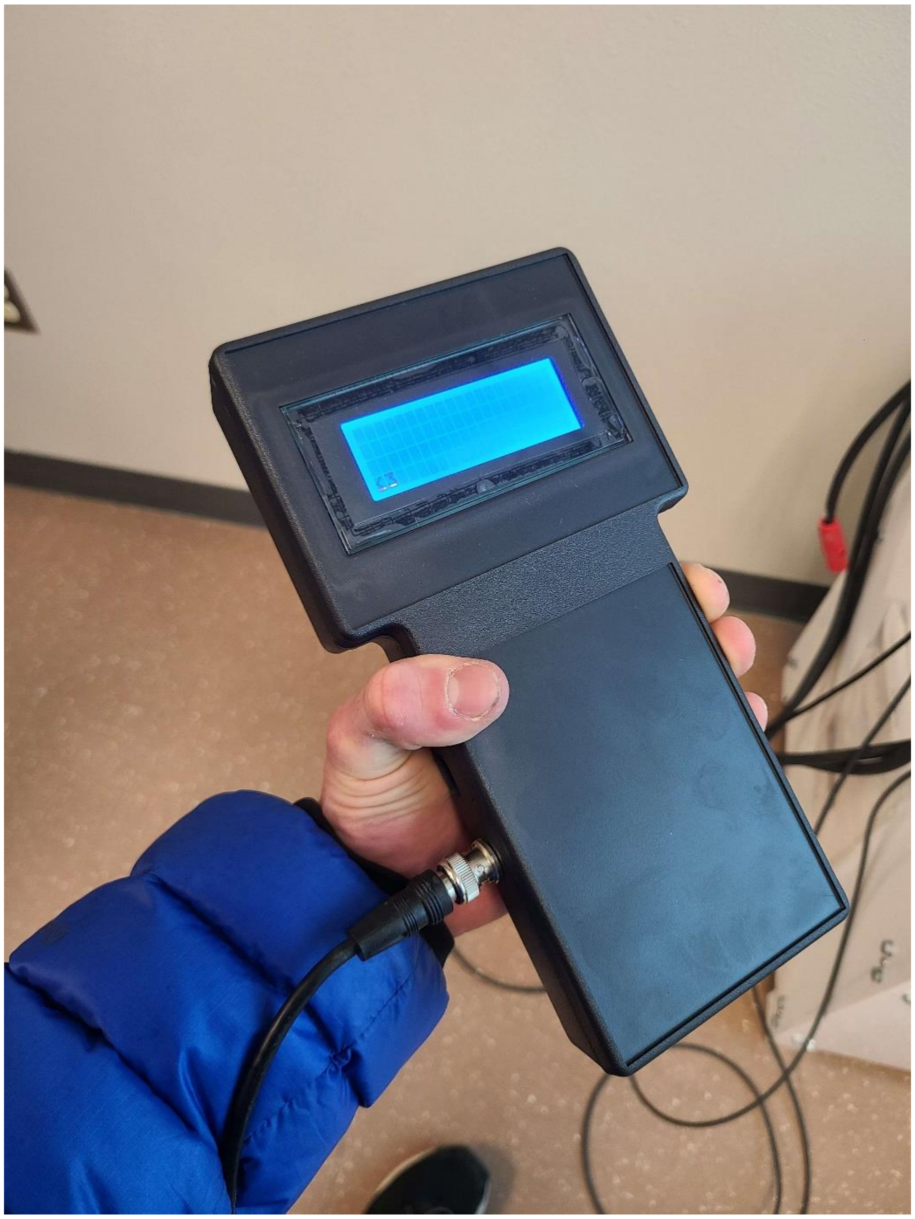


Figure 21: Assembled Product

4.3 Software Implementation

Software implementation for each individual device used code which follows a two-step process of first initializing all peripherals and then entering an infinite operational loop for gathering and displaying data. The flow diagram for the software implementation is shown below in *Figure 22*. A more top-level outline for how the software loop calculates and displays to the user shown in *Figure 23*.

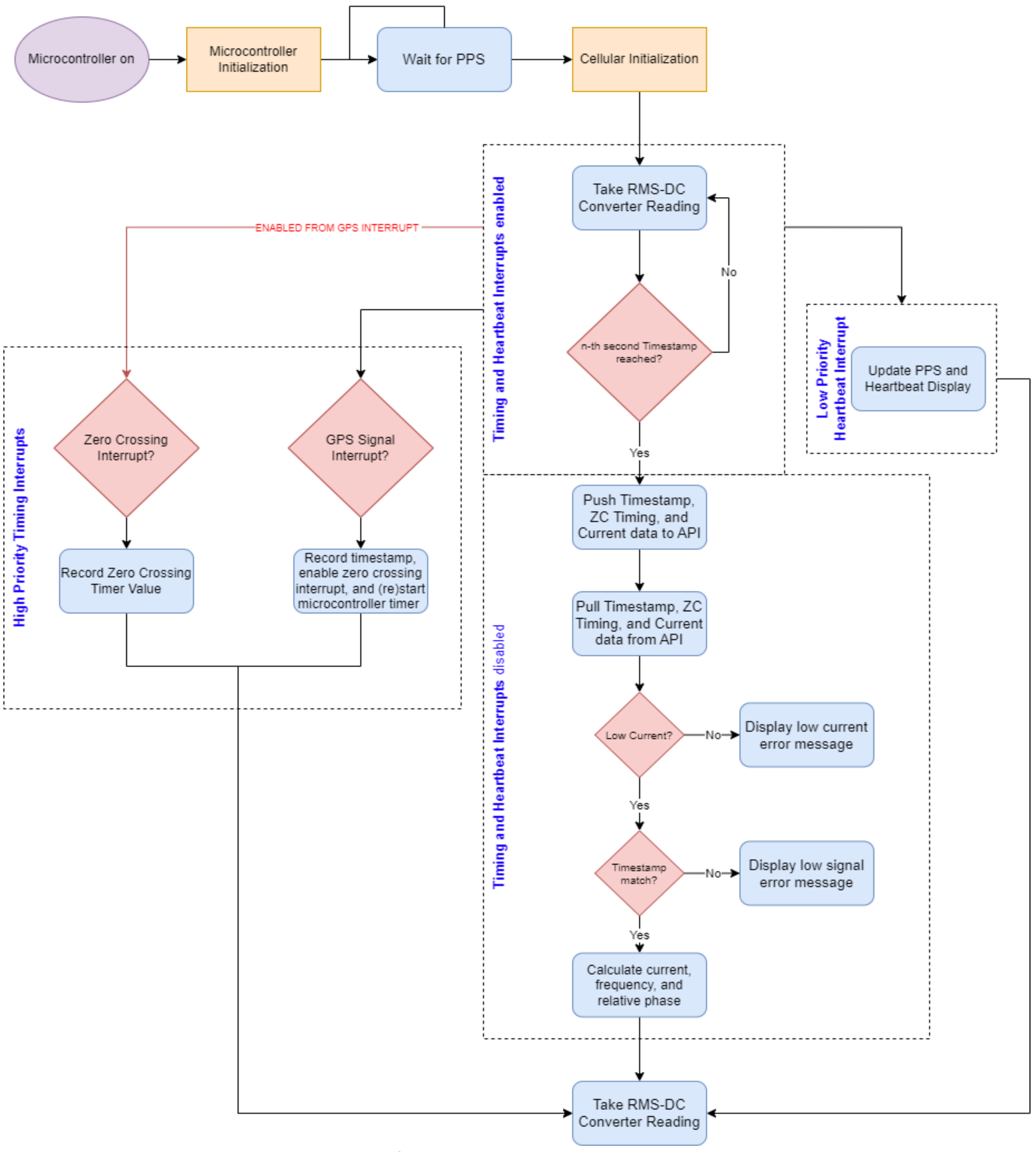

Microcontroller on
Microcontroller Initialization
Wait for PPS
Cellular Initialization
Timing and Heartbeat Interrupts enabled
Take RMS-DC Converter Reading
n-th second Timestamp reached?
No
Yes
ENABLED FROM GPS INTERRUPT
High Priority Timing Interrupts
Zero Crossing Interrupt?
GPS Signal Interrupt?
Record Zero Crossing Timer Value
Record timestamp, enable zero crossing interrupt, and (re)start microcontroller timer
Low Priority Heartbeat Interrupt
Update PPS and Heartbeat Display
Timing and Heartbeat Interrupts disabled
Push Timestamp, ZC Timing, and Current data to API
Pull Timestamp, ZC Timing, and Current data from API
Low Current?
No
Display low current error message
Yes
Timestamp match?
No
Display low signal error message
Yes
Calculate current, frequency, and relative phase
Take RMS-DC Converter Reading

Figure 22: Software Implementation Flow Diagram

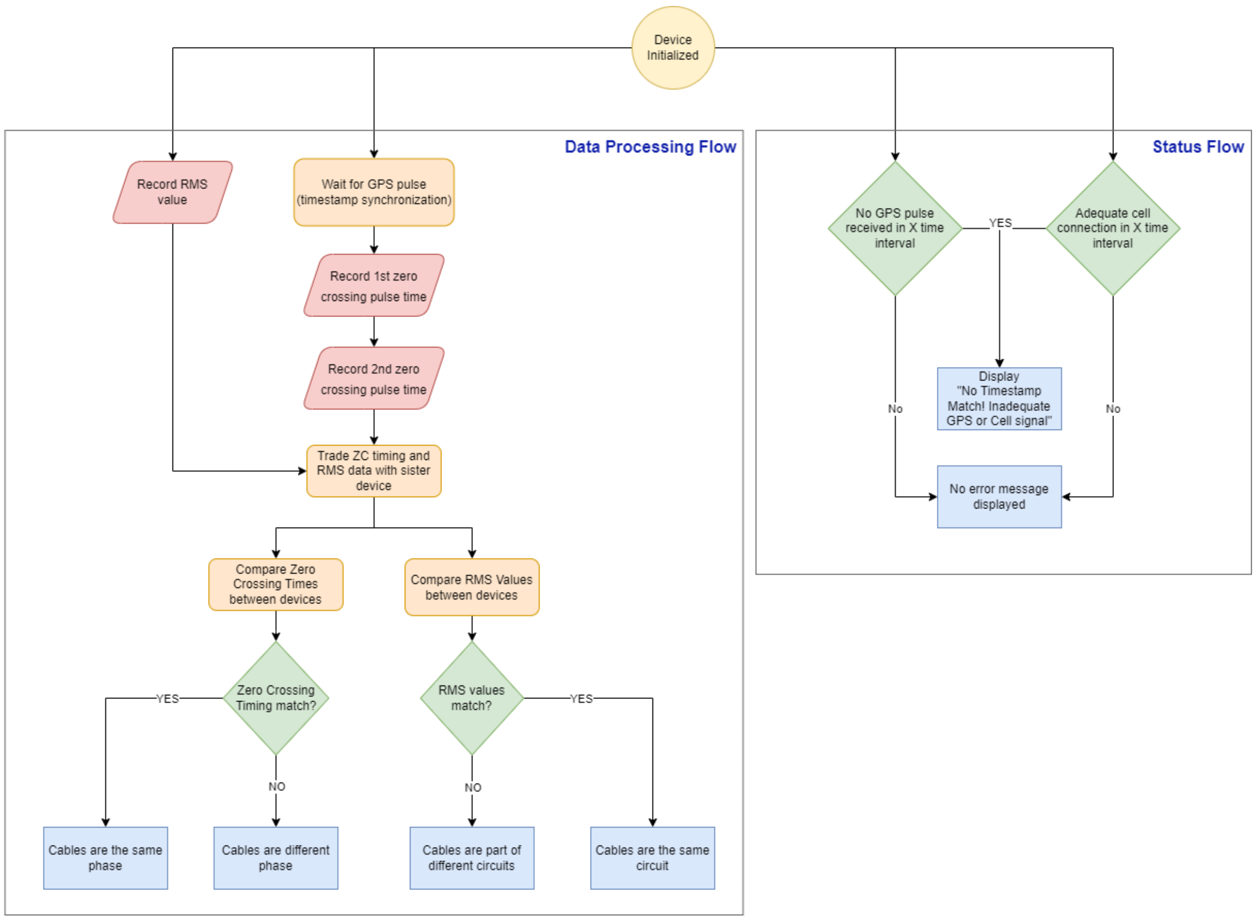

Device Initialized
Data Processing Flow
Record RMS value
Wait for GPS pulse (timestamp synchronization)
Record 1st zero crossing pulse time
Record 2nd zero crossing pulse time
Trade ZC timing and RMS data with sister device
Compare Zero Crossing Times between devices
Compare RMS Values between devices
Zero Crossing Timing match?
RMS values match?
YES
NO
Cables are the same phase
Cables are different phase
Cables are part of different circuits
Cables are the same circuit
Status Flow
No GPS pulse received in X time interval
Adequate cell connection in X time interval
Display "No Timestamp Match! Inadequate GPS or Cell signal"
No error message displayed
No

Figure 23: High Level Flow Diagram

4.3.1 Peripheral Initialization

The software begins with a simple initialization process that turns on and/or configures all peripherals of the microcontroller (LCD display, GPS, Cellular Module). For the LCD display, it is powered up and configured for 4-bit command mode in which commands must be sent in nibbles (4-bits at a time). The GPS is configured to disable all default messages while also enabling the ZDA GNSS sentence to be produced every second, which provides information only pertaining to the date and time. Finally, the cellular module is turned on by pulsing the PWRKEY pin of the development board to a LOW voltage and connecting the module to the DWEET.io API.

4.3.2 Operational Loop

Following peripheral initialization, each microcontroller enters an operational loop in which it switches between taking data and updating information to the cloud and its own display. The data polling loop consists of periodically checking the values read by the ADC, receiving timestamps from the GPS, and timing zero crossings. This process continues until a UTC time divisible by 10 is reached (ie 12:30.40PM). This triggers both microcontrollers to simultaneously enter their informational updating state. First the cloud data for each respective device is updated, after which the data for the other device is pulled (with an appropriate delay to ensure that the information has been received by the cloud). Afterwards, the microcontrollers compare the local and pulled data to calculate phase differences, line current matches, and frequency.

5. Testing Procedures

5.1 Specification Testing

Each specification outlined in the objectives, requirements, and specifications part of this report were tested against a set of procedures proving that the standard has been met. For specifications regarding regulations of communications electronics, proof that the specification standards have been met is shown with the manufacturer testing procedures for each specification are outlined in *Table 3* and *Table 4* shown below.

Table 3: Specification Testing Procedures and Results

Specification	The magnitude of the current shall be within 50% of the minimum differential between parallel lines.	Phase values shall be within +/- 10 degrees	Information display must be readable from 5 feet away	device shall operate for a minimum of 2 hours
Equipment (Devices incl.)	ALB power bench Current probe	ALB power bench	Measuring Tape	Timer
Procedure	1. Analyze current draw data for nearby substations for current draw differences between parallel lines. 2. Initialize device(s). 3. Initialize ALB(s). 4. Initialize current probe(s). 5. Attach Rogowski coils to any phase. Attach current probe(s) to the same phase. 6. Check that difference between device reading and current probe reading is within the tolerance value of the minimum current difference between parallel lines.	1. Initialize device(s). 2. Initialize ALB(s) 3. Attach Rogowski coils to the same phase of ALB. 4. Check that phase reading is within the phase tolerance threshold of the nominal phase difference value. 5. Attach Rogowski coils to differing phases of ALB and repeat previous step (4).	1. Initialize device(s). 2. Set device on a level surface at an elevation between 3-6ft from ground in the hours of 8-5PM. 3. Check readability from a marked position directly in front of device at 5 feet away.	1. Initialize device(s). 2. Ensure cellular and GPS connection has been established. 3. Check if device(s) are on every 15 minutes until minimum operating threshold has been surpassed.
Pass/Fail	Pass	Pass	Pass	Pass
Comments	Typical error <3 Amps, minimum differential from substation data 6 Amps [2]	• Error at low current (<100A) ~6 degrees • Error at high current (>100A) ~1 degree	Readable from up to 10 feet	255 mA typical operating current (1.05W at 4.2V) 200 mA when idling without cellular connection
Date of Test	3/30/2023	3/30/2023	3/30/2023	4/12/2023
Tester(s) signature	CD	CD	CD	CD

Table 4: Specification Testing Procedures and Results Continued

Specification	Device dimensions shall be less than 210 x 100 x 60 mm	device shall weigh less than 8 pounds	device shall connect to a cell tower from a minimum distance of 2 miles	Shall adhere to FCC title 47 standards regarding telecommunications	Shall adhere to FAA 49 CFR 175.10 standards regarding transportation of batteries on airlines
Equipment (Devices incl.)	Measuring Tape	Scale	Map	None-Rated by third party	None-Rated by third party
Procedure	1. Measure X,Y, and Z axis of devices. 2. Check if respective dimension thresholds are surpassed.	1. Weigh device(s). 2. Check if weight threshold exceeded.	1. Find location(s) of cellphone tower in FCC database: (https://wireless2.fcc.gov/UlsApp/AsrSearch/asrRegistrationSearch.jsp) 2. Travel to a location ~2 miles away from a tower. 3. Initialize device(s) and check that API has been updated.	1. Check that GPS and Cellular modules comply with FCC title 47 according to manufacturer specs.	2. Check that Li-ion batteries used by device(s) do not exceed 100Wh rating according to manufacturer specs.
Pass/Fail	Pass	Pass	Pass	Pass	Pass
Comments	Device dimensions: 235x177x38 mm	Device+coil: 1lb, 6.3 oz	No comment	FCC Approval IDs SIM7000:2AYJU NEO6: N/A, module not subject to regulation	3.8V, 2.5Ah batteries 9.5 Wh
Date of Test	4/14/2023	4/14/2023	3/28/2023	4/12/2023	4/4/2023
Tester(s) signature	CD	CD	CD	CD	CD

Field Testing

In addition to testing each of the specifications, a field test was also performed. This consisted of testing the system in various configurations (parallel lines, varying distance, differing phases, etc.). *Table 5* outlines the displayed results for each of the conditions that the system was operating under.

Field testing was conducted at a MidAmerican 7.4kV substation in Dakota Dunes, South Dakota. The first stage of testing consisted of a tour of the facility, including a tour of the onsite SCADA equipment and potential conductors that the system could be testing on. Eventually it was decided that the best plan of action would be to first conduct short range testing by connecting the system in various configurations to the secondary side of the transformer as seen in Figure *24*. This would be followed by longer range testing in which one device would be connected to the transformer, while the other would be connected to a feeder approximately 50 feet away. These conductors were chosen because they provided an open view of the sky, and therefore better cellular and GPS connection, as well as drawing a current of about 200 amps which guarantees accurate readings.

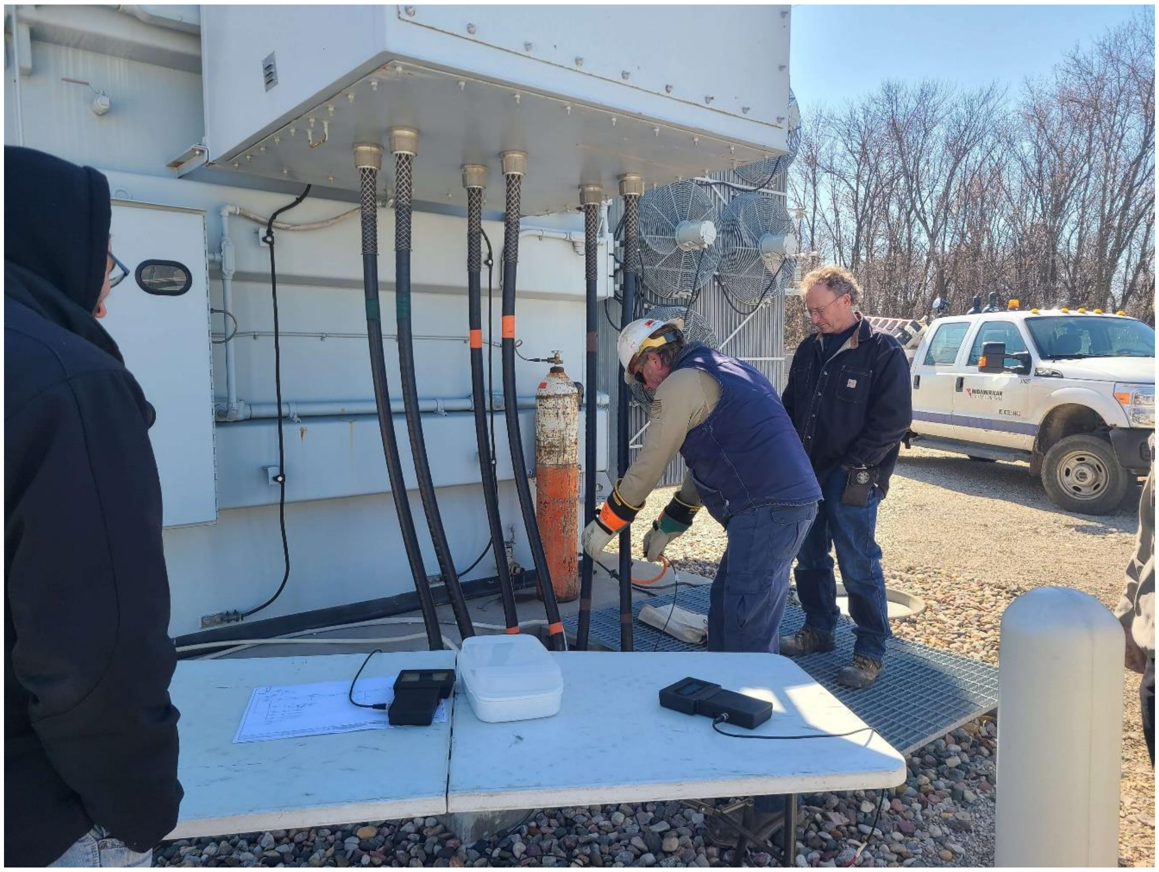


Figure 24: Live field testing on secondary transformer

Table 5: Field Testing Data

Xfrmer->Xfrmer	Phase (deg)	Actual Phase	Current (A)	Xfrmer->Feeder	Phase (deg)	Actual Phase	Current (A)
A->C	120	120	198/190	A->C	113	114	165/77
A->B	111	113	193/194	A->A	7	6	169/87
B->C	129	127	178/171				
A->A	0	0	199/178				
B->B	0	0	198/197				
A->B'	110	113	199/178				
B->C'	129	127	196/177				

The system was able to identify whether lines were of the same phase and discern the difference between parallel conductors. One major observation that was different from expected was the phase being offset from a multiple of 120 degrees. For example, when measuring between phases B and C, a phase difference of 110 degrees was shown instead of 120. Similar offsets (127 degrees) were seen when measuring over a long distance from phase A to C. While at first this was thought to be due to some sort of magnetic coupling between conductors, further analysis of data from monitoring equipment at the substation show that the current phases were in fact roughly 7-10 degrees away from a balanced value of 120 degrees. Other issues that arose during testing were a periodic lack of adequate signal for the GPS to identify conductors, as well as one of the devices shutting off roughly ~2 hours into testing. This first issue was attributed to the feeder device operating near a large metal building with shielding occurring, while the device powering down was due to the wall charger that was used to charge the device the night before not having a good connection.

6. Design Budget and Expenditures

The total budget given for this project was $1000 on the design for the two devices and $4000 for test equipment that may need to be purchased. The overall budget that was spent can be seen in *Table 6*. The total expenditure on all components was $879.68. This falls under the $1000 budget that was given to us for the development of both devices. The overall cost could be brought down even more because some components were purchased in extras such as the GPS, resistors, and capacitors.

Table 6: Project Expenditures

Product	Quantity	Price/Unit	Subtotal	Total (Taxes+Shipping)
Cellular Module	2	65.00	130.00	139.56
LCD Display x2	1	16.99	16.99	16.99
SIM Card	2	5.50	11.00	21.41
LT1116	2	10.40	20.80	20.80
LTC1968	2	11.91	23.82	31.81
Demo Board V1.0	5			17.14
Demo Board V1.0 Parts				128.31
Mouser Order				72.68
Batteries	2			26.64
PCB V1.0				55.06
Final PCB Parts				127.70
PCB V1.1				12.14
Max 7C GPS	2			73.47
PCB V1.2				22.14
Digikey Order 6				73.75
Digikey Order 5				40.08
Total				879.68

In addition to the overall project expenditures, the BOM and price per component can be seen in **Error! Reference source not found.**. As seen in the table below, the overall price of each unit is only around $200. This meets the goal of developing two devices under $1000. In addition to the overall price per unit, a two-channel function generator was also purchased through Mr. Jason Sternhagen to test the devices during the Senior Design Expo. This came out of the $4000 budget to purchase test equipment and would also prove useful in testing the devices without having to run the ALB's in the power lab. Not included in the budget expenditure this year, the same exact Rogowski coils that were purchased 3 years ago were used again for this year. The price per Rogowski coil is $118.29 which adds on to the cost of the BOM. This brings the overall cost per device up to around $310.

Table 7: BOM and Individual Component Cost

Item	Qty	Reference(s)	Value	LibPart	Price ($)
1	1	A1	Botletics SIM7000	MCU_Module:Arduino_UNO_R3	65.00
2	10	C1, C9, C10, C17, C22, C23, C25, C27, C30, C33	0.1u	Device:C	0.44
3	3	C2, C4, C13	10u	Device:C	0.35
4	5	C3, C24, C26, C28, C31	0.01u	Device:C	0.31
5	3	C5, C6, C20	2.2u	Device:C	0.30
6	4	C7, C8, C14, C16	4.7u	Device:C	0.66
7	2	C11, C12	22u	Device:C	0.37
8	3	C15, C21, C29	1u	Device:C	0.31
9	2	C18, C19	10p	Device:C	0.22
10	1	C32	10n	Device:C	0.18
11	2	D1, D2	BZV55C2V4	Diode:BZV55C2V4	0.39
12	2	D3, D4	LED	Device:LED	0.36
13	1	D5	BAT42WS-E3-08	Diode:BAT42W-V	0.25
14	4	H1, H2, H3, H4	MountingHole	Mechanical:MountingHole	0.00
15	1	J1	Conn_01x04	Connector_Generic:Conn_01x04	0.00
16	1	J2	Conn_01x05	Connector_Generic:Conn_01x05	0.00
17	1	J3	USB_C_Receptacle	Connector:USB_C_Receptacle	1.72
18	1	J4	Conn_01x02	Connector_Generic:Conn_01x02	0.00
19	6	J5, J7, J10, J11, J12, J13	Conn_01x01	Connector_Generic:Conn_01x01	0.00
20	1	J6	Conn_Coaxial	Connector:Conn_Coaxial	10.72
21	1	J8	Conn_Coaxial	Connector:Conn_Coaxial	1.29
22	1	J9	Conn_01x01	:Conn_01x01_1	0.00
23	1	J14	Conn_01x02	Connector_Generic:Conn_01x02	0.00
24	1	L1	4.7u	Device:L	2.05
25	1	L3	27n	Device:L	0.23
26	3	R1, R3, R21	10k	Device:R	1.74
27	1	R2	330k	Device:R	0.34
28	1	R4	1.13k	Device:R	0.03
29	1	R5	16k	Device:R	0.03
30	1	R6	1.18k	Device:R	0.02
31	1	R7	10	Device:R	0.21
32	3	R8, R14, R15	100k	Device:R	0.06
33	4	R9, R16, R17, R20	1k	Device:R	0.10
34	1	R10	2k	Device:R	0.10
35	1	R11	1M	Device:R	0.03
36	2	R12, R13	5.1k	Device:R	0.05
37	2	R18, R19	330	Device:R	0.04
38	1	SW1	SW_SPST	Switch:SW_SPST	0.94
39	1	U1	TPS613222ADBV	Regulator_Switching:TPS613222ADBV	1.16
40	1	U2	TL084	Amplifier_Operational:TL084	0.52
41	1	U3	max1673esa+T	PhaseID:max1673esa+T	5.64
42	1	U4	LTC1968CMS8-TRPBF	PhaseID:LTC1968CMS8-TRPBF	12.05
43	1	U5	AP7361C-33ER	Regulator_Linear:AP7361C-33E	0.63
44	1	U7	BQ24072RGT	Battery_Management:BQ24072RGT	3.61
45	1	U8	TM4C1233H6PM	MCU_Texas:TM4C1231H6PM	13.65
46	1	U10	MAX-8C	RF_GPS:MAX-8C	21.00
47	1	Y1	ABM3-16.000MHZ-D2Y-T	Device:Crystal	0.58
48			2.5Ah Battery	Li-ion Battery	14.95
49			931-1144-ND	RF Antenna	15.32
50			HM3998-ND	Case	16.18
				TOTAL	194.12

7. Conclusions

The system fulfilled not only the objectives set out to solve the problem of being able to identify high voltage conductors, but also to the standards of industry workers that would be prospective end-users of the devices. With the inclusion of cellular communication, working distance for the devices is essentially infinite so long as an adequate signal can be obtained. High accuracy both with current magnitudes (<3A error) and phase (<0.1 degrees error) can be obtained when large currents are being measured (>100A). While unbalanced current phases will cause displayed phases to be slightly off from 120 degrees, this is not enough to prevent the more important danger of conductors being improperly identified. Finally, the GPS outdoors took around 35 seconds from starting up to fully lock on to enough satellites to provide the PPS. The biggest downfall of this device however is that if it is used indoors, it can sometimes take up to 15 minutes to fully lock on to the satellites even when near a window. This is clearly a limitation of the current design.

Future Work

The newest improvement to the system, the inclusion of cellular communication instead of RF (Radio Frequency), extends the range of the devices to essentially infinite. As such, no future improvements are intended to the devices themselves, however a future update with a proprietary API instead of DWEET.io could be done. Perhaps researching a better way to get the GPS PPS faster indoors could be done for future work.

8. References

[1] M. Belisario, N. Mahowald and J. Winter, "High Voltage Distribution Phase Identifier Phase II," 2021.

[2] Texas Instruments, "Active Integrator for Rogowski Coil Reference Design With Improved Accuracy for Relays and Breakers," 2016.

9. Appendix

Operation Instructions

1. The first step in operating the Conductor Identifier is to make sure that the SIM cards have data loaded onto them. Go to Hologram.io and login and select the organization “Conductor Identifier.”
2. Next step is to make sure that both devices have data and are activated. If they do not have data, you will need to go to the “billing” tab and add balance then activate the cards.
3. The cards should be activated within around 10 minutes and the devices can be used.
4. For best operating conditions, the devices should be used outside. If this is not possible, place each device close to a window.
5. For charging the devices, plug in any USB-C connector to charge.
6. Once devices are charged, turn them on and wait for the GPS and wrap the Rogowski coil around the conductor.

Microcontroller Code:

```
#include "TM4C123.h"

#include <math.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#define pin0 0x01
#define pin1 0x02
#define pin2 0x04
#define pin3 0x08
#define pin4 0x10
#define pin5 0x20
#define pin6 0x40
#define pin7 0x80

#define CLOCK_GPIOA  0x01
#define CLOCK_GPIOB  0x02
#define CLOCK_GPIOC  0x04
#define CLOCK_GPIOD  0x08
#define CLOCK_GPIOE  0x10
#define CLOCK_GPIOF  0x20

#define Row1 0x00
#define Row2 0x40
#define Row3 0x14
#define Row4 0x54

//#define GPSTESTING
#define INITDELAYS
//#define SINGLEDEVICETESTING
//#define PCBTESTING

void I2C3_init(void);
void I2C_Send(char byte);
void Receive_LCD_Address(void);
void LCD_Send(char byte);
void SendLCDString(char string[]);
void Send_LCD_Command(char command);
void Write_Character(char character);
void InitializeLCD(void);
void PrintPhaseData(void);
void PrintCurrentData(void);
void PrintFrequencyData(void);
void PrintLCDData(void);
void PrintLowSignalErrorMessage(void);
void PrintProbeErrorMessage(void);


void InitializationSequence(void);
void GPIO_PWRKEY_init(void);
void ADC0_init(void);
void UART0_init(void);
void UART1_init(void);
void UART4_init(void);
void GPIO_inits(void);
void PPS_GPIO_init(void);
void ZC_GPIO_init(void);
void ZC_Timer_init(void);
void HeartBeatTimerInit(void);

void MaskMeasurementInterrupts(void);
void UnmaskMeasurementInterrupts(void);
void ConvertADCReadingtoCurrent(void);
void ADC0_TakeSamples(void);
void ProcessZCTimingData(void);
void TurnOnSIM7000(void);
void delayMs(unsigned long counter);
void TestLocalComs(void);
void echoATcom(void);
void sendATcom(void);
void rsfPutChar (char c);
void ReceiveFromGPS(void);
void ReceiveFromSIM7000(void);
void UpdateTime(void);

void WaitTilUART1ReceiveNotBusy(void);
void WaitTilUART1NotBusy(void);
void WaitTilUART4ReceiveNotBusy(void);
void WaitTilUART4NotBusy(void);
void DesyncDeviceDWEETing(void);

void UpdatePulledFromAPIVars(char APIName[]);
void PullAPIData(char DweetThingName[]);
void SendDWEET(int DWEETData);
void SendToSIM7000(char SendBuf[]);
void SendNumDeviceOnDWEET(char DweetThingName[],int NumOn);
void HTTP_init(void);
```

```
int FindGPSMessage(char MessageName[],int MessageNameSize);
void PrintGPSREADING(int n);

void CompareData(void);
void ConfigureGPSMessages(void);
void UpdateCloudData(char DweetThingName[]);
int ParseSIM7000ForString(char SearchString[],int SearchStringSize);
int ReturnVariableData(char APIVariableName[], int APIVariableNameLength);
int FindNumDevicesInitialized(char DweetThingName[]);
void AddVariableToSend(char SendBuffer[], char VariableName[], int VariableData);

void PopulateHeader(void);
void PopulateClassID(int CommandClass, int ID);
void PopulatePayloadLength(int PaySize);
void PopulateTargetMSG(int MSGClass, int MSGID);
void PopulateDisableUARTPorts(bool Disable);
void PopulateRatePayload(void);
void PopulateChecksum(int PayloadEnd);
void SendToNEO6GPS(char SendBuf[]);

char PPSResetDelayAPIVar[]="ResetDelay";
char TimeStampAPIVar[]="TimeStamp";
char ZeroCrossingAPIVar[]="ZCTime";
char CurrentAPIVar[]="CurrentReading";
char DesyncVariable[]="NumDevicesInitialized";
char Device1DWEETname[] = "phaseid1";
char Device2DWEETname[]= "phaseid2";

int OOFThreshold = 15;                                          //1.0 degree of phase
shift
int ParallelCircuitThreshold = 100;    //max current difference for parallel lines
int CurrentErrorThreshold = 400;                     //min current for definitive reading

char GPSCommand[40];
int TESTSTRING[60];
char READING[120];
char SIM7000Reading[400];
int GPSIndex = 0;
int SIM7000Index = 0;

unsigned volatile int test_time = 233012;
unsigned volatile int test_zc_time = 50000000;

unsigned volatile int LOCAL_TIME = 0;
unsigned volatile int PULLED_TIME = 0;
unsigned volatile int LOCAL_CURRENT = 0;
unsigned volatile int PULLED_CURRENT = 0;

unsigned volatile int LOCAL_ZC_TIMING[60] = {0};
int ZCTimingIndex = 0;unsigned volatile int PULLED_ZC_TIMING = 0;

int MeasurementsPerSec = 5;
int PPSResetDelay[60]= {0}; //Stores Difference between expected and actual time PPS resets
int MeasurementDelay[40]={0};
int TestInt = 0;
int DeviceIdentifier = 0;
bool TimeReadyToUpdate = false;
bool NewTimeStampReceived = false;
bool FIRST_PPSRECEIVED = false;

float Frequency = 0;
int Phasematch = 2;
float PhaseDeg = 0;
bool CurrentMatch = false;
//int ErrorState = 1;           //0 = Normal Operation, 1=Current probe Error,2=GPS/Cellular
error,3 or 4=Error message printed

bool LOWCURRENTERROR = false;
bool LOWSIGNALERROR  = false;
bool HEARTBEAT = false;
bool NEW_PPSRECEIVED = false;

unsigned volatile int ADC0_data = 0;

// LCD IO EXPANDER (MSB->LSB)
// 0000 0 0 0  0
// DDDD X E RW RS

// LCD ROW ADDRESSES
// Row 1 = 0x00
// Row 2 = 0x40
// Row 3 = 0x14
// Row 4 = 0x54

//INPUTS
//PE2 = Current RMS (ADC1)
//PA5 = PPS waveform (GPIOA Interrupt)
//PC6 = ZC waveform (WT1CCP0)

//OUTPUTS
//PF1 = PWRKEY
```

```
//UARTS
//UART1 for SIM7000
//PB0 = Rx, PB1 = Tx
//UART4 for NEO6x GPS
//PC4 = Rx, PC5 = Tx

//I2C
//I2C3 for LCD Display
//PD0 = SCL, PD1 = SDA

//Peripherals//
//SIM7000
//PF1->6
//PB0->10,PB1->11
//5V and VBAT->3.3V, GND->GND

void UART4_Handler(void){
int i=0;
ReceiveFromGPS();

if(NewTimeStampReceived == true){
UpdateTime();
for(i=0;i<119;i++)
READING[i]=0;
NewTimeStampReceived=false;

if(LOCAL_TIME%10==0){
//GPIOA->IM |= 0x20;
//UART4->IM &= ~0x10;
}
}
UART4->ICR |= 0x10;
}

void UART1_Handler(void){
ReceiveFromSIM7000();

UART1->ICR |= 0x10;
}

void GPIOA_Handler(void){
PPSResetDelay[TestInt] = WTIMER1->TAV-50000000;
WTIMER1->TAV |= 0xFFFFFFFF; //Reset TimerA0 to 0 whenever PPS occurs
TestInt++;
if(TestInt>60){
```

```
for(TestInt=0;TestInt<60;TestInt++)
PPSResetDelay[TestInt]=0;
TestInt=0;
}
delayMs(1);

if(LOCAL_TIME%10==0){
ZCTimingIndex=0;

WTIMER1->ICR |= 0x4; //Clear Capture interrupt from timer 0
WTIMER1->IMR |= 0x4; //Unmask Capture Interrupt from timer0A

GPIOA->IM   &= ~0x20;
}
if(FIRST_PPSRECEIVED == false)
FIRST_PPSRECEIVED = true;

NEW_PPSRECEIVED=true;
//NVIC_ClearPendingIRQ(GPIOA_IRQn);
GPIOA->ICR |= 0x20;          //Clear GPIOA interrupt
}

void WTIMER1A_Handler(void){
MeasurementDelay[ZCTimingIndex]= WTIMER1->TAV;
LOCAL_ZC_TIMING[ZCTimingIndex] = WTIMER1->TAR;
ZCTimingIndex++;

if(ZCTimingIndex > 20){
//UART4->IM |= 0x10;
WTIMER1->IMR &= ~0x4;
TimeReadyToUpdate = true;
}

WTIMER1->ICR|= 0x4;       //Clear Capture interrupt from timer0A
}

void WTIMER2A_Handler(void){
//        if(FIRST_PPSRECEIVED == false)
//                Send_LCD_Command(0x01);

if(HEARTBEAT){
Send_LCD_Command(0x80|Row4+18);
SendLCDString("<3");
HEARTBEAT = false;
}
else{
```

```
Send_LCD_Command(0x80|Row4+18);
SendLCDString("__");
HEARTBEAT = true;
}

if(NEW_PPSRECEIVED || GPIOA->RIS == 0x20){
Send_LCD_Command(0x80|Row4+17);
SendLCDString(".");
NEW_PPSRECEIVED=false;
}
else{
Send_LCD_Command(0x80|Row4+17);
SendLCDString("_");
}

WTIMER2->TAV=0x0;
WTIMER2->ICR|=0x10;
}
int main(void){
InitializationSequence();

ZCTimingIndex=0;

while(1){
ADC0_TakeSamples();
ConvertADCReadingtoCurrent();
delayMs(250);

if(TimeReadyToUpdate==true){
MaskMeasurementInterrupts();

if(DeviceIdentifier==1){
UpdateCloudData(Device1DWEETname);
delayMs(4000);
UpdatePulledFromAPIVars(Device2DWEETname);
}
else{
UpdateCloudData(Device2DWEETname);
delayMs(4000);
UpdatePulledFromAPIVars(Device1DWEETname);
}

CompareData();
PrintLCDData();
LOCAL_TIME=1;
TimeReadyToUpdate=false;
UnmaskMeasurementInterrupts();
}
}
}



void PopulateHeader(void){
GPSCommand[0] = 0xB5; GPSCommand[1] = 0x62;                    //Standard UBX
Header
}
void PopulateClassID(int CommandClass, int ID){
GPSCommand[2] = CommandClass; GPSCommand[3] = ID;                    //Class,ID for
CFG command
}
void PopulatePayloadLength(int PaySize){                    //Payload Length for
enabling/disabling GPS ports
GPSCommand[4] = PaySize; GPSCommand[5] = 0x0;
}
void PopulateTargetMSG(int MSGClass, int MSGID){
GPSCommand[6] = MSGClass; GPSCommand[7] = MSGID; //Target NMEA message Class
and ID
}
void PopulateDisableUARTPorts(bool Disable){
if(Disable == true){
GPSCommand[8] = 0x1;          //DDC (I2C) port enabled
GPSCommand[9] = 0x0;          //UART1 port disabled
GPSCommand[10] = 0x0;         //UART2 port disabled
GPSCommand[11] = 0x0;         //USB port disabled
GPSCommand[12] = 0x1;         //SPI port enabled
GPSCommand[13] = 0x1;         //6th port enabled?
}
else{
GPSCommand[8] = 0x1;          //DDC (I2C) port enabled
GPSCommand[9] = 0x1;          //UART1 port enabled
GPSCommand[10] = 0x1;         //UART2 port enabled
GPSCommand[11] = 0x1;         //USB port enabled
GPSCommand[12] = 0x1;         //SPI port enabled
GPSCommand[13] = 0x1;         //6th port enabled?
}
}
void PopulateChecksum(int CheckSumStart){
int CK_A=0;
int CK_B=0;
int i;
for(i=2;i<CheckSumStart;i++){
```

```
CK_A = CK_A+GPSCommand[i];
CK_B = CK_B+CK_A;
}
GPSCommand[CheckSumStart] = CK_A;
GPSCommand[CheckSumStart+1] = CK_B;
GPSCommand[CheckSumStart+2] = '\r';
GPSCommand[CheckSumStart+3] = '\n';
}


void PopulateRatePayload(void){
GPSCommand[6] = 0x10;
GPSCommand[7] = 0x27;
GPSCommand[8] = 0x01;
GPSCommand[9] = 0x00;
GPSCommand[10]= 0x01;
GPSCommand[11]= 0x00;
}


void TurnOnSIM7000(void){
GPIOF->DATA &= (~pin1);
delayMs(20000);
GPIOF->DATA |= (pin1);
delayMs(7000);
}

void delayMs(unsigned long counter){
unsigned long i = 0;
unsigned long j = 0;
for(i = 0; i < counter; i++){
for(j=0;j<3180;j++);
}
}

void ADC0_TakeSamples(void){
ADC0->PSSI  |= 0x08;          //Start A Sampling on SS3
while((ADC0->RIS & 0x08   )==0){};                    //Wait on status of SS3
  ADC0_data = (ADC0->SSFIFO3 & 0xFFF); //Load FIFO into data variable;
ADC0->ISC    = 0x08;                               // clear status flag & go again

}
void ConvertADCReadingtoCurrent(void){
LOCAL_CURRENT = ADC0_data*(3.3/4096)*(2160); //3.3V/4096 bits ADC, 1000Amps/0.6V
rogowski output, 2160A/V observed
if(LOCAL_CURRENT < CurrentErrorThreshold)
```

```
LOWCURRENTERROR = true;
else
LOWCURRENTERROR = false;
}
void TestLocalComs(void){
int i;
char com[] = "Test\n";
int size = sizeof(com);
for(i = 0; i<size; i++){
while((UART0->FR & 0x8) != 0); // wait until UART is not busy
UART0->DR = com[i];
UART1->DR = com[i];
}
}

void echoATcom(void){
while((UART0->FR & 0x8) != 0); // wait until UART is not busy
UART0->DR = UART1->DR;
//char payload = UART1->DR;
//SendLCDString(payload);
}

void sendATcom(void){
int i;
char com[] = "AT";
int size = sizeof(com);
for(i = 0; i<size; i++){
WaitTilUART1NotBusy(); // wait until UART is not busy
UART1->DR = com[i];
UART0->DR = com[i];
//UART0->DR = UART4->DR;
}
}
void ReceiveFromGPS(void){
WaitTilUART4NotBusy();

READING[GPSIndex] = UART4->DR;

if(READING[GPSIndex]==0x0A){
GPSIndex=0;
NewTimeStampReceived = true;
}
else
GPSIndex++;
}
```

```
void ReceiveFromSIM7000(void){
WaitTilUART1ReceiveNotBusy();

SIM7000Reading[SIM7000Index] = UART1->DR;

if(SIM7000Reading[SIM7000Index]==0x0A){
SIM7000Index=0;
}
else
SIM7000Index++;
}

void UpdateTime(void){
int i,j,GPSpos;
LOCAL_TIME = 0;

GPSpos=FindGPSMessage("GPZDA",5);
j=3;

for(i=0;i<4;i++){
LOCAL_TIME+=(READING[GPSpos+3]-48)*pow(10,j);
GPSpos++;j--;
}
}

void PrintGPSREADING(int n){
int i;
char GPSSTRING[80];

for(i=0; i<sizeof(GPSSTRING); i++){
GPSSTRING[i] = READING[n];
n++;
}
Send_LCD_Command(0x01); //Clear LCD
Send_LCD_Command(0x80|Row1);

SendLCDString(GPSSTRING);
delayMs(3000);
}



int FindGPSMessage(char MessageName[],int MessageNameSize){
int i;
int j=0;
```

```
for(i=0; i<sizeof(READING); i++){
if(j == (MessageNameSize))
return i;
else if(READING[i]==MessageName[j]){
j++;
}
else
j=0;
}
return -1;
}

void SendToSIM7000(char SendBuf[]){
int i = -1;
strcat(SendBuf,"\r\n");
do{
i++;
WaitTilUART1NotBusy();
UART1->DR = SendBuf[i];
}while(SendBuf[i] != '\n');
delayMs(500);
}

void UpdatePulledFromAPIVars(char APIName[]){
PullAPIData(APIName);
PULLED_TIME                    =       ReturnVariableData(TimeStampAPIVar,
sizeof(TimeStampAPIVar));
PULLED_ZC_TIMING       =       ReturnVariableData(ZeroCrossingAPIVar,
sizeof(ZeroCrossingAPIVar));
PULLED_CURRENT                 =       ReturnVariableData(CurrentAPIVar,
sizeof(CurrentAPIVar));
}

void SendToNEO6GPS(char SendBuf[]){
int i = -1;
do{
i++;
WaitTilUART4NotBusy();
UART4->DR = SendBuf[i];
}while(SendBuf[i] != '\n');
delayMs(50);
}

void SendDWEET(int DWEETData){
char DWEETDataBuf[5];
char DweetBuf[65]    = "AT+HTTPPARA=\"url\",dweet.io/dweet/quietly/for/phaseid?data=";
char DweetAction[]    = "AT+HTTPACTION=1";

sprintf(DWEETDataBuf, "%d", DWEETData);
strcat(DweetBuf,DWEETDataBuf);

SendToSIM7000(DweetBuf);
SendToSIM7000(DweetAction);
}

void UpdateCloudData(char DweetThingName[]){
char DweetBuf[120]   = "AT+HTTPPARA=\"url\",dweet.io/dweet/quietly/for/";
char DweetAction[]    = "AT+HTTPACTION=1";

strcat(DweetBuf, DweetThingName);
strcat(DweetBuf, "?");

AddVariableToSend(DweetBuf,TimeStampAPIVar,LOCAL_TIME);
AddVariableToSend(DweetBuf,ZeroCrossingAPIVar,LOCAL_ZC_TIMING[1]);
AddVariableToSend(DweetBuf,CurrentAPIVar,LOCAL_CURRENT);
//AddVariableToSend(DweetBuf,PPSResetDelayAPIVar,PPSResetDelay);

SendToSIM7000(DweetBuf);
SendToSIM7000(DweetAction);
delayMs(500);
}

void PullAPIData(char DweetThingName[]){
char DweetBuf[65]     = "AT+HTTPPARA=\"url\",dweet.io/get/latest/dweet/for/";
char DweetAction[]    = "AT+HTTPACTION=0";
char CheckDweetData[]         = "AT+HTTPREAD";

strcat(DweetBuf, DweetThingName);

SendToSIM7000(DweetBuf);
SendToSIM7000(DweetAction);
delayMs(4000);                    //Extra Delay for server to respond
SendToSIM7000(CheckDweetData);
delayMs(200);                     //Extra Delay to load response
}

int ParseSIM7000ForString(char SearchString[],int SearchStringSize){
int i;
int j=0;

for(i=0; i<(sizeof(SIM7000Reading)/sizeof(char)); i++){
if(j == (SearchStringSize)-2)
return i;
else if(SIM7000Reading[i]==SearchString[j]){
j++;
}
else
j=0;
}

return -1;
}
int ReturnVariableData(char APIVariableName[], int APIVariableNameLength){
char DataBuffer[15];
int VariableValue, i = 0;
int VariableDataIndex = ParseSIM7000ForString(APIVariableName,
APIVariableNameLength)+3;

if(VariableDataIndex != -1){
while(SIM7000Reading[VariableDataIndex] != ',' && SIM7000Reading[VariableDataIndex] !=
'}'){
DataBuffer[i] = SIM7000Reading[VariableDataIndex];
VariableDataIndex++;
i++;
};
VariableValue = atoi(DataBuffer);
}
else VariableValue = -1;
return VariableValue;
}

void AddVariableToSend(char SendBuffer[], char VariableName[], int VariableData){
char ConcatDataBuff[20];

strcat(SendBuffer,"&");
strcat(SendBuffer,VariableName);
strcat(SendBuffer,"=");
sprintf(ConcatDataBuff,"%d",VariableData);
strcat(SendBuffer,ConcatDataBuff);
}
void CompareData(void){
int timedifferenceRaw,currentdifference,i,FrequencySum;
Phasematch = 2;
CurrentMatch = false;
PhaseDeg=0;Frequency=0;FrequencySum=0;


for(i=0;i<(ZCTimingIndex-1);i++){
```

```
FrequencySum += (LOCAL_ZC_TIMING[i+1]-LOCAL_ZC_TIMING[i]);
}
//Frequency = 50000000/(Frequency/ValidDifferences);
Frequency = 50000000/((float)FrequencySum/(ZCTimingIndex-1));
//Timestamps match
#ifndef SINGLEDEVICETESTING
if(LOCAL_TIME-PULLED_TIME == 0){
#endif
LOWSIGNALERROR = false;
timedifferenceRaw = LOCAL_ZC_TIMING[1] - PULLED_ZC_TIMING;
currentdifference = LOCAL_CURRENT - PULLED_CURRENT;
//PhaseDeg = (((float)timedifferenceRaw/50000000)*(Frequency*360));
        //(frequency*360)/50000000)

//(frequency*360)/Clockrate
PhaseDeg = (float)timedifferenceRaw/2314;
//if(abs((int)PhaseDeg) > 180)                //Ensures 120 increments
//      PhaseDeg = PhaseDeg-360;

if(abs((int)PhaseDeg) < OOFThreshold){
Phasematch = 0;
}
else if(PhaseDeg < OOFThreshold){
Phasematch = -1;
}
else if(PhaseDeg > OOFThreshold){
Phasematch = 1;
}

if(abs(currentdifference) < ParallelCircuitThreshold)
CurrentMatch = true;
#ifndef SINGLEDEVICETESTING
}
//Timestamps dont match
else
LOWSIGNALERROR = true;
#endif

}
void WaitTilUART1ReceiveNotBusy(void){
while((UART1->FR & 0x20) != 0);
}
void WaitTilUART4ReceiveNotBusy(void){
while((UART4->FR & 0x20) != 0);
}
void WaitTilUART1NotBusy(void){
```

```
while((UART1->FR & 0x8) != 0);
}
void WaitTilUART4NotBusy(void){
while((UART4->FR & 0x8) != 0);
}

//Bite-sized commands sent in nibble pairs
void Send_LCD_Command(char command){
char UpperNibble = command & 0xF0;
char LowerNibble = (command & 0x0F)<<4;

LCD_Send(UpperNibble);
LCD_Send(LowerNibble);
delayMs(8);
}
void Write_Character(char character){
char UpperNibble = (character & 0xF0) | 0x1;
char LowerNibble = ((character & 0x0F)<<4) | 0x1;

LCD_Send(UpperNibble);
LCD_Send(LowerNibble);
delayMs(1);
}

void LCD_Send(char byte){
I2C_Send(byte);
I2C_Send(byte | 0x0C);
delayMs(1);
I2C_Send(byte | 0x08);
delayMs(1);
}
void I2C_Send(char byte){
I2C3->MDR = byte;
I2C3->MCS = 0x7;
while(I2C3->MCS &0x1000000);
}
void SendLCDString(char string[]){
int i=0;
while(string[i]!='\0'){
Write_Character(string[i]);
i++;
}
}
void PrintPhaseData(void){
char PhaseDegBuf[5];
char PhaseDisplay[10] = "Deg:";
```

```
sprintf(PhaseDegBuf,"%0.2f", PhaseDeg);
strcat(PhaseDisplay,PhaseDegBuf);

SendLCDString(PhaseDisplay);

Send_LCD_Command(0x80|Row2);
switch(Phasematch){
case 0:
SendLCDString("Phase:MATCH"); break;
case -1:
SendLCDString("Phase:LAGGING"); break;
case 1:
SendLCDString("Phase:LEADING"); break;
default:
SendLCDString("Phase:ERROR"); break;
}
}
void PrintCurrentData(void){
char CurrentDataBuf[5];
char CurrentDisplay[10]="LocA:";
char CurrentDisplay2[10]="PulA:";
sprintf(CurrentDataBuf, "%0.1f", (float)LOCAL_CURRENT/10);
strcat(CurrentDisplay,CurrentDataBuf);

sprintf(CurrentDataBuf, "%0.1f", (float)PULLED_CURRENT/10);
strcat(CurrentDisplay2,CurrentDataBuf);

Send_LCD_Command(0x80|Row3);
SendLCDString(CurrentDisplay);

Send_LCD_Command(0x80|Row3+10);
SendLCDString(CurrentDisplay2);

Send_LCD_Command(0x80|Row4);
if(CurrentMatch)
SendLCDString("Line Match:TRUE");
else
SendLCDString("Line Match:FALSE");
}
void PrintFrequencyData(void){
char FreqDataBuf[4];
char FreqDisplay[10]="Freq:";

sprintf(FreqDataBuf, "%0.2f", Frequency);
strcat(FreqDisplay,FreqDataBuf);
```

```
Send_LCD_Command(0x80|Row1+10);
SendLCDString(FreqDisplay);
}
void PrintLCDData(void){
Send_LCD_Command(0x01); //Clear LCD
Send_LCD_Command(0x80|Row1);

if(LOWSIGNALERROR == false && LOWCURRENTERROR == false){
PrintPhaseData();
PrintCurrentData();
PrintFrequencyData();
}
else if(LOWCURRENTERROR==true)
PrintProbeErrorMessage();
else if(LOWSIGNALERROR ==true)
PrintLowSignalErrorMessage();
else
SendLCDString("UNKNOWN ERROR");
}
void PrintLowSignalErrorMessage(void){
Send_LCD_Command(0x01);           //Clear LCD
Send_LCD_Command(0x80|Row1);
SendLCDString("No Timestamp Match!");
Send_LCD_Command(0x80|Row2);
SendLCDString("Low GPS/Cell Signal!");
}
void PrintProbeErrorMessage(void){
Send_LCD_Command(0x01); //Clear LCD
Send_LCD_Command(0x80|Row1);
SendLCDString("No Current Reading!");
Send_LCD_Command(0x80|Row2);
SendLCDString("Adjust Probes!");
}

void UART0_init(void){

SYSCTL->RCGCUART |= 0x1;    // enable clock for UART Module 0
SYSCTL->RCGCGPIO |= CLOCK_GPIOA;    // set clock for GPIO Port A
delayMs(1);

GPIOA->DEN |= (pin0 + pin1);   // digitally enable PA0 and PA1 (A0=Rx, A1=Tx)
GPIOA->AFSEL |= (pin0 + pin1); // enable PA0 and PA1 in alternate function mode
GPIOA->PCTL |= 0x00000011;  // set PA0 and PA1 to alternate UART function

UART0->CTL &= ~0x1;                           // disable UART before making changes
```

```
UART0->IBRD = 325;                    // calculated for 9600 baud --- 50MHz/(16*9600) =
325.5208
UART0->FBRD = 33;                     // calculated with int(0.5208*64+0.5) = 33.8312 --- round
down, gives 33
UART0->LCRH |= 0x60;          // set configuration
UART0->CC &= ~0xF;                              // set main clock as UART0 clock
UART0->CTL |= 0x301;          // enable UART0

delayMs(1);
}

void InitializationSequence(void){
//For LCD Display
I2C3_init();
InitializeLCD();
//For USB, SIM7000, GPS UARTS
UART0_init();
UART1_init();
UART4_init();

//Current RMS-DC value
ADC0_init();
//PPS and Zero Crossing
PPS_GPIO_init();
ZC_GPIO_init();
ZC_Timer_init();
//GPS NMEA Messages
ConfigureGPSMessages();
delayMs(10000);
//SIM7000 HTTP connection

HeartBeatTimerInit();

Send_LCD_Command(0x01);
GPIOA->IM   |= pin5;
while(FIRST_PPSRECEIVED==false){
Send_LCD_Command(0x80|Row1);
SendLCDString("WaitingforPPS...");
delayMs(5000);
}
GPIOA->IM   &= ~pin5;
//Turn on SIM7000
GPIO_PWRKEY_init();
TurnOnSIM7000();
HTTP_init();
```

```
//SendLCDString(SIM7000Reading);
DeviceIdentifier=2;

//Differentiate devices based on startup order
//DesyncDeviceDWEETing();
delayMs(2000);
//UART4->IM |= 0x00;
GPIOA->IM   |= pin5;                                         //Unmask interrupt for
PA5
WTIMER1->IMR |= 0x4;                                    //Unmask ZC interrupt
UART1->IM |= 0x10;                                      //Unmask Cellular interrupt
UART4->IM |= 0x10;                                      //Unmask GPS RX interrupt
}

void I2C3_init(void){
SYSCTL->RCGCGPIO |= CLOCK_GPIOD;                        //Clock to Port D
   SYSCTL->RCGCI2C |= 0x8;                       //I2C3 Clock
delayMs(1);

GPIOD->DEN |= (pin0+pin1);
GPIOD->AFSEL |= (pin0+pin1);      //PD0(SCL) PD1(SDA)
GPIOD->PCTL |= 0x00000033;    //I2C3 on Pins 0 and 1
GPIOD->ODR |= pin1;

I2C3->MCR  |= 0x10;
I2C3->MTPR |= 0x9;
I2C3->MSA = 0x3F << 1;
}

void InitializeLCD(void){
delayMs(1);
LCD_Send(0x30);
LCD_Send(0x30);
LCD_Send(0x30);      //3x Special Startup Commands
LCD_Send(0x20);

Send_LCD_Command(0x28); //2 Line, 5x7 font
Send_LCD_Command(0x08); //On
Send_LCD_Command(0x01); //Clear Display
Send_LCD_Command(0x06); //Increment Entry Mode
Send_LCD_Command(0x0C);
Send_LCD_Command(0x02);

Send_LCD_Command(0x80|Row1);
SendLCDString("LCD Initialized!");
#ifndef INITDELAYS
```

```
delayMs(400);
#endif
}

//SIM7000
void UART1_init(void){

SYSCTL->RCGCUART |= 0x2;    // enable clock for UART Module 1
SYSCTL->RCGCGPIO |= CLOCK_GPIOB;    // set clock for GPIO Port B

GPIOB->DEN |= (pin0 + pin1);   // digitally enable PB0 and PB1 (B0=Rx, B1=Tx)
GPIOB->AFSEL |= (pin0 + pin1); // enable PB0 and PB1 in alternate function mode
GPIOB->PCTL |= 0x00000011;  // set PB0 and PB1 to alternate UART function

UART1->CTL &= ~0x1;                            // disable UART before making changes
UART1->IBRD = 325;                    // calculated for 9600 baud --- 50MHz/(16*9600) =
325.5208
UART1->FBRD = 33;                     // calculated with int(0.5208*64+0.5) = 33.8312 --- round
down, gives 33
UART1->LCRH |= 0x60;          // set configuration
UART1->CC &= ~0xF;                             // set main clock as UART1 clock
UART1->CTL |= 0x301;          // enable UART1

delayMs(1);

//UART1->IM |= 0x10;                                                   //enable RX interrupt
UART1->IM &= ~0x10;                                                  //enable RX interrupt
NVIC_EnableIRQ(UART1_IRQn);                       //enable UART4 intterupt vector
NVIC_SetPriority(UART1_IRQn,1);

Send_LCD_Command(0x80|Row2);
SendLCDString("Cell UART Started!");
#ifndef INITDELAYS
delayMs(1000);
#endif
}

//GPS
void UART4_init(void){

SYSCTL->RCGCUART |= 0x10;    // enable clock for UART Module 4
SYSCTL->RCGCGPIO |= CLOCK_GPIOC;    // set clock for GPIO Port C

GPIOC->DEN |= (pin4 + pin5);   // digitally enable PC4 and PC5 (C4=Rx, C5=Tx)
GPIOC->AFSEL |= (pin4 + pin5); // enable PC4 and PC5 in alternate function mode
GPIOC->PCTL |= 0x00110000;  // set PC4 and PC5 to alternate UART function
```

```
UART4->CTL &= ~0x1;                              // disable UART before making changes
UART4->IBRD = 325;                  // calculated for 9600 baud --- 50MHz/(16*9600) =
325.5208
UART4->FBRD = 33;                   // calculated with int(0.5208*64+0.5) = 33.8312 --- round
down, gives 33
UART4->LCRH |= 0x60;        // set configuration
UART4->CC &= ~0xF;                               // set main clock as UART4 clock
//UART4->LCRH |= 0x70;
UART4->CTL |= 0x301;        // enable UART4

delayMs(1);

//UART4->IM |= 0x10;                                                    //enable RX interrup
UART4->IM &= ~0x10;

NVIC_EnableIRQ(UART4_IRQn);                     //enable UART4 intterupt vector
NVIC_SetPriority(UART4_IRQn,2);


Send_LCD_Command(0x80|Row3);
SendLCDString("GPS UART Initialized");
#ifndef INITDELAYS
delayMs(1000);
#endif
}

void ADC0_init(void){
SYSCTL->RCGCADC         |= 0x01;                            // clock to Port E
SYSCTL->RCGCGPIO        |= CLOCK_GPIOE;   // ADC0

delayMs(3);
GPIOE->AFSEL        |= pin2;                // PE2 Alternate function
GPIOE->DEN          &=~pin2;                // Clearing, do NOT want digital for A2D
GPIOE->AMSEL        |= pin2;                // select analog mode
ADC0->ISC           |= 0x08;                            // clear status flag-- can now access ADC
regs w/out fault
ADC0->ACTSS         &=~0x0F;                            // disable all sample sequencesrs (#3 should
be enough-p 821)

while(ADC0->ACTSS &= 0x10000){};       // wait for not busy?  -- should fall through right
away

ADC0->EMUX                  &=(~0xF000);// EM3 = 0000 -> software trigger
ADC0->SSMUX3        = 0x01;                             // Select AN1 (PE2) as the analog input
```

```
ADC0->SSCTL3       |= 0x06;                   // 1st sample is end of sequence (so we're done after
1)
ADC0->PC                    = 0x03;                               // 0x03 = 250kS/s
ADC0->SAC                   |= 0x04;                 // 16x oversampling and then averaged
ADC0->ACTSS                 = 0x08;                               // Configure (re-anable) ADC0
module for sequencer 3

Send_LCD_Command(0x80|Row4);
SendLCDString("ADC Initialized!");
#ifndef INITDELAYS
delayMs(1000);
#endif
}


//TurnOnSIM7000 pulls PWRKEY pin on SIM7000 to GND for >64mS to wake module
void PPS_GPIO_init(void){
SYSCTL->RCGCGPIO |= CLOCK_GPIOA;          // set clock for GPIO Port A
delayMs(1);

GPIOA->DIR &= ~pin5;                 // set PA5 to IN
GPIOA->DEN |= pin5;                           // digitally enable PA5
GPIOA->PDR |= pin5;                           // enable pulldown resistor for PA5

GPIOA->IS     &= ~pin5;                        // PA5 is edge-sensitive
GPIOA->IBE&= ~pin5;                           // PA5 is not both edges
GPIOA->IEV|= pin5;                   // PA5 falling edge event

GPIOA->ICR = pin5;                                                  //Clear flag for pin 5
GPIOA->IM   |= pin5;                                                         //Unmask interrupt for
PA5
NVIC_EnableIRQ(GPIOA_IRQn);                    //Enable GPIOA interrupts
NVIC_SetPriority(GPIOA_IRQn,1); //Priority is 2

Send_LCD_Command(0x01);
Send_LCD_Command(0x80|Row1);
SendLCDString("PPS Initialized!");
#ifndef INITDELAYS
delayMs(1000);
#endif
}

void ZC_GPIO_init(void){
SYSCTL->RCGCGPIO |= CLOCK_GPIOC;          // set clock for GPIO Port C
delayMs(1);
```

```
GPIOC->DIR &= ~pin6;                                    //PC6 to input
GPIOC->DEN |= pin6;                                              //Digital enable PC6
GPIOC->AFSEL |= (pin6);                        //set PC6 to alternate function
GPIOC->PCTL |= 0x07000000;        //set PC6 to timer function
}
void ZC_Timer_init(void){


SYSCTL->RCGCWTIMER |=0x2;// Enable clock for timer0;
delayMs(1);
WTIMER1->CTL &=~0x1;                           // Ensure TimerA1 is disabled
WTIMER1->CFG |= 0x4;                           // Timer1 on
WTIMER1->TAMR|= 0x17;             // TimerA1 to capture, edge-time, and count up
WTIMER1->CTL &= ~0xC;             // TimerA1 to rising edge trigger event
//WTIMER1->CTL |= 0x4;              // TimerA1 to FALLING EDGE FO TESTING
WTIMER1->TBILR|=0x0000FFFF; //TimerB1 register must be loaded first to avoid interrupt
thrown
WTIMER1->TAILR|=0xFFFFFFFF; //TimerA1 loaded with 0x0;

//WTIMER1->IMR |= 0x4;                         // Unmask Capture interrupt
WTIMER1->IMR &= ~0x4;
NVIC_EnableIRQ(WTIMER1A_IRQn);                                   //Enable TimerA1 interrupt
NVIC_SetPriority(WTIMER1A_IRQn, 4);   //Priority 1

WTIMER1->CTL |= 0x1;                           // Enable TimerA0;

Send_LCD_Command(0x80|Row2);
SendLCDString("ZC Initialized!");
#ifndef INITDELAYS
delayMs(1000);
#endif
}

void HeartBeatTimerInit(void){
SYSCTL->RCGCWTIMER |= 0x4;
delayMs(1);
WTIMER2->CFG &= 0x0;
WTIMER2->CTL &= ~0x1;
WTIMER2->CFG |= 0x4;
WTIMER2->TAMR|= 0x32;

WTIMER2->TAILR |= 0xFFFFFFFF;
WTIMER2->TAMATCHR = 0x08000000;

WTIMER2->IMR |= 0x10;
NVIC_EnableIRQ(WTIMER2A_IRQn);
```

```
NVIC_SetPriority(WTIMER2A_IRQn,5);
WTIMER2->CTL |=0x1;
}
void ConfigureGPSMessages(void){
int MSGCheckSumStart = 14;
int DefaultMessageID;

//Populates GPS Send buffer with:
//Header,CFG, and Payload and Length fields for disabling NMEA messages
PopulateHeader();
PopulateClassID(0x06,0x01);
PopulatePayloadLength(0x08);
PopulateDisableUARTPorts(true);

//Disables GGA,GLL,GSA,GSV,RMC, and VTG in that order
for(DefaultMessageID=0;DefaultMessageID<6;DefaultMessageID++){
PopulateTargetMSG(0xF0,DefaultMessageID);       //Disables GGA
PopulateChecksum(MSGCheckSumStart);
SendToNEO6GPS(GPSCommand);
}

//Enables ZDA (time NMEA sentence)
PopulateTargetMSG(0xF0,0x8);
PopulateDisableUARTPorts(false);
PopulateChecksum(MSGCheckSumStart);
SendToNEO6GPS(GPSCommand);

/*
//Reenable GGA (Satellite Strenght sentence)
PopulateTargetMSG(0xF0,0x00);
PopulateDisableUARTPorts(false);
PopulateChecksum(MSGCheckSumStart);
SendToNEO6GPS(GPSCommand);

//Reenable GSV (Satellite Strenght sentence)
PopulateTargetMSG(0xF0,0x03);
PopulateDisableUARTPorts(false);
PopulateChecksum(MSGCheckSumStart);
SendToNEO6GPS(GPSCommand);
*/

/*
//Set Messages to once per 5 sec
PopulateClassID(0x06,0x08);
PopulatePayloadLength(0x06);
PopulateRatePayload();
```

```
PopulateChecksum(MSGCheckSumStart-2);
SendToNEO6GPS(GPSCommand);
*/

Send_LCD_Command(0x80|Row3);
SendLCDString("GPS Messages Set!");
delayMs(400);
}

void HTTP_init(void){
int i;
char SendBuf[40] = "AT";
SendToSIM7000(SendBuf);

#ifdef PCBTESTING
while(1){
SendLCDString("Checking");
delayMs(5000);
}
#endif

for(i=0;i<3;i++){
strcpy(SendBuf,"AT+SAPBR=3,1,\"APN\",\"hologram\"");
SendToSIM7000(SendBuf);
strcpy(SendBuf,"AT+CSTT=\"hologram\"");
SendToSIM7000(SendBuf);
strcpy(SendBuf,"AT+SAPBR=1,1");
SendToSIM7000(SendBuf);

strcpy(SendBuf,"AT+CGNSPWR=0");
SendToSIM7000(SendBuf);

strcpy(SendBuf,"AT+HTTPINIT");
SendToSIM7000(SendBuf);
}

Send_LCD_Command(0x80|Row4);
SendLCDString("HTTP Sequence Sent!");
#ifndef INITDELAYS
delayMs(400);
#endif
}

void DesyncDeviceDWEETing(void){
/*
int NumDeviceInitialized;
```

```
NumDeviceInitialized = FindNumDevicesInitialized(Device1DWEETname);

if(NumDeviceInitialized == 0 || NumDeviceInitialized == -1){
SendNumDeviceOnDWEET(Device1DWEETname,1);
DeviceIdentifier = 1;
}
else{
SendNumDeviceOnDWEET(Device2DWEETname,2);
DeviceIdentifier = 2;
}
Send_LCD_Command(0x01);
Send_LCD_Command(0x80|Row1);
SendLCDString("Unit Initialized!");
Send_LCD_Command(0x80|Row2);
SendLCDString("Waiting for other");
Send_LCD_Command(0x80|Row3);
SendLCDString("unit...");

while(NumDeviceInitialized!=2){
NumDeviceInitialized = FindNumDevicesInitialized(Device2DWEETname);
delayMs(2000);
Send_LCD_Command(0x80|Row4);
SendLCDString("<3");
delayMs(2000);
Send_LCD_Command(0x80|Row4);
SendLCDString("__");
}
*/
DeviceIdentifier = 1;
if(DeviceIdentifier == 1)
UpdateCloudData(Device1DWEETname);
if(DeviceIdentifier == 2)
UpdateCloudData(Device2DWEETname);
}

int FindNumDevicesInitialized(char DweetThingName[]){
int NumDevicesInitialized;
PullAPIData(DweetThingName);
NumDevicesInitialized = ReturnVariableData(DesyncVariable, sizeof(DesyncVariable));
return NumDevicesInitialized;
}

void SendNumDeviceOnDWEET(char DweetThingName[],int NumOn){
char DWEETDataBuf[5];
char DweetBuf[100]   = "AT+HTTPPARA=\"url\",dweet.io/dweet/quietly/for/";
char DweetAction[]    = "AT+HTTPACTION=1";
```

```
strcat(DweetBuf, DweetThingName);
strcat(DweetBuf, "?");
strcat(DweetBuf, "NumDevicesInitialized=");

sprintf(DWEETDataBuf, "%d", NumOn);
strcat(DweetBuf,DWEETDataBuf);

SendToSIM7000(DweetBuf);
SendToSIM7000(DweetAction);
}

void GPIO_PWRKEY_init(void){
SYSCTL->RCGCGPIO |= CLOCK_GPIOF;
delayMs(1);
GPIOF->DIR |= pin1;
GPIOF->DEN |= pin1;
}
void MaskMeasurementInterrupts(void){
GPIOA->IM   &= ~0x20;
WTIMER1->IMR &= ~0x4;
UART4->IM &= ~0x10;
WTIMER2->IMR &= 0x10;
}
void UnmaskMeasurementInterrupts(void){
GPIOA->IM   |= 0x20;
//WTIMER1->IMR |= 0x4;
UART4->IM |= 0x10;
WTIMER2->IMR |= 0x1;
}
```