Project Necromancer
Design Document

Sponsored By: The Rainforest Connection (RFCx)

Submitted By:

Joe Gibson	gibsjose@mail.gvsu.edu
David Adlof	adlofd@mail.gvsu.edu
Kalee Stutzman	stutzmak@mail.gvsu.edu
Jesse Millwood	millwooj@mail.gvsu.edu

Date Submitted: July 2015

Executive Summary

Software, electrical, and hardware enhancements to Rainforest Connection's currently used device are proposed in this paper. The electrical enhancements include the addition of a new PCB that contains power regulation and battery charging circuitry, along with current, voltage, temperature, and humidity sensors. The software enhancements include sending sensor measurements to an Android cellphone. Sending measurements from the PCB to the phone will allow for diagnostic feedback that can be used for laboratory testing and verification as well as field statistics. The point of this project is not to entirely redesign the existing device but rather to provide modular enhancements and improvements to specific components.

Power consumption is a major concern and a key area for improvement on the existing device. We will use *Max Point Power Tracking*(MPPT) ICs in the design of the power regulation and solar panel interface to ensure maximum efficiency from the solar panels. In addition we will use Lithium-Ion battery management ICs to ensure that the batteries are properly charged, properly discharged, and to prevent overheating. A switching power regulator will boost the voltage from either the batteries or the solar panel to charge the phone and a linear dropout regulator will provide 3.3V to the regulation and diagnostic circuitry on the PCB.

Temperature is another major concern. To mitigate the heat produced by the phone, the power circuitry, and the environment, a heatsink will be attached to the PCB with direct thermal contact with the phone. The heatsink will extend outside of the enclosure, transferring heat outside of the device enclosure.

Sending diagnostics will be achieved using an 8-bit Atmel ATmega328P microcontroller. Power, temperature, and humidity data will be sent from the microcontroller to the cell phone via USB, where it will be received and displayed by an Android application.

Keywords

Rain forest, Recycling, RFCx, Rainforest Connection, cellphone, Android, logging

Grand Valley State University

Contents

1 RFCx Background

Rainforest Connection (RFCx) is a non-profit organization that is dedicated to stopping illegal logging and deforestation in rainforests throughout the world. The destruction of tropical rainforests is a leading cause of carbon dioxide emission, and much of it is caused by illegal activities. RFCx combats illegal logging with devices using re-purposed cell-phones strategically placed in trees in remote areas. A picture of the entire device is shown below in Figure 1.1.

Figure 1.1: RFCx President Topher White installing an existing device in the rainforests of Borneo

These devices record and analyze audio, using their data connection to periodically send audio data to the RFCx server. Digital signal processing algorithms are used to detect the sound of chainsaws and engines at a distance of up to $\frac{2}{3}$ of a mile. If a chainsaw is detected, an alert is sent to pre-existing ground patrols, who can react to the situation and combat the threat. In addition to man-made sounds, the RFCx platform is also used to record the sounds of animals in the forest. These sounds can be used to gain information about certain species of animals and are open to anyone to use and analyze. Because of the sheer size of the area, it is impractical, if not impossible, for foot patrols to properly monitor the forests for illegal activities without the aid of technology. The flow of data, from audio collection to human intervention, is shown below in Figure 1.2.

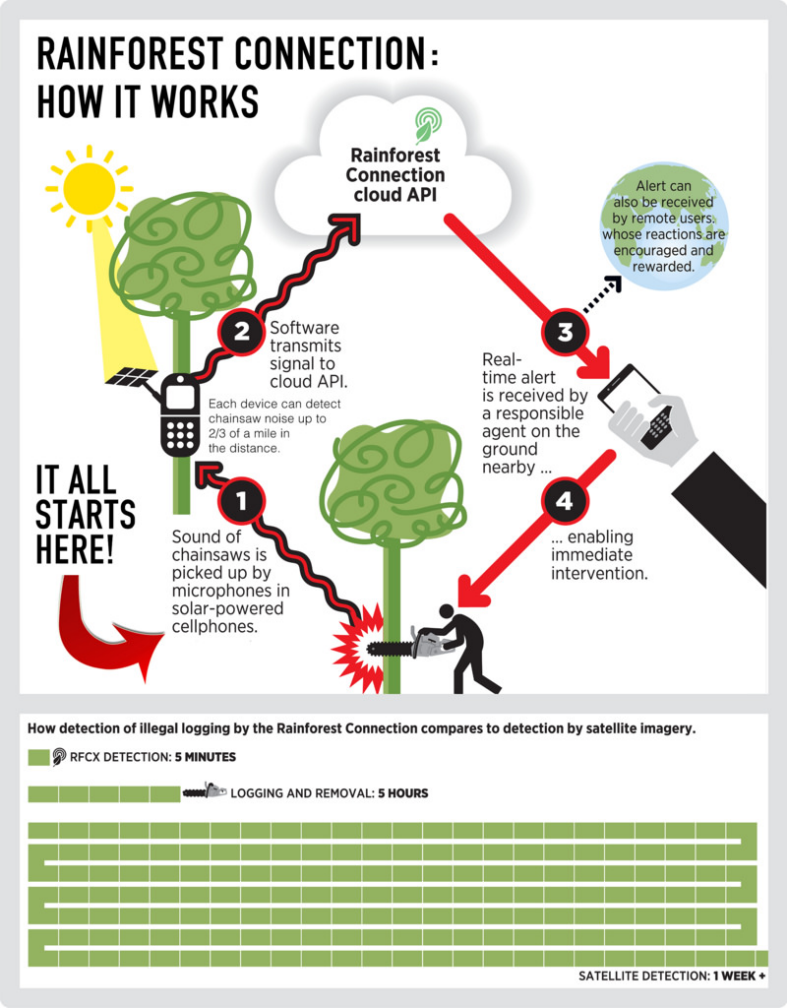


Figure 1.2: Infographic from RFCx website (www.rfcx.org) showing the flow of data

2 Project Introduction

RFCx Project Necromancer is primarily comprised of enhancements to the existing system in the areas of power consumption, power efficiency, temperature concerns, and overall modularity of components. The devices currently consist of an Android phone, Radioshack enclosure, off-the-shelf external microhpone and antenna, RFCx solar petals, two dual-cell Lithium-Ion batteries, and various off-the-shelf PCBs and individual electrical components comprising the power regulation and solar and battery management control. The external microphone, antenna, solar petals, batteries (Appendix E.6), and Android phone will remain the same. A view inside of the device is shown in Figure 2.1.

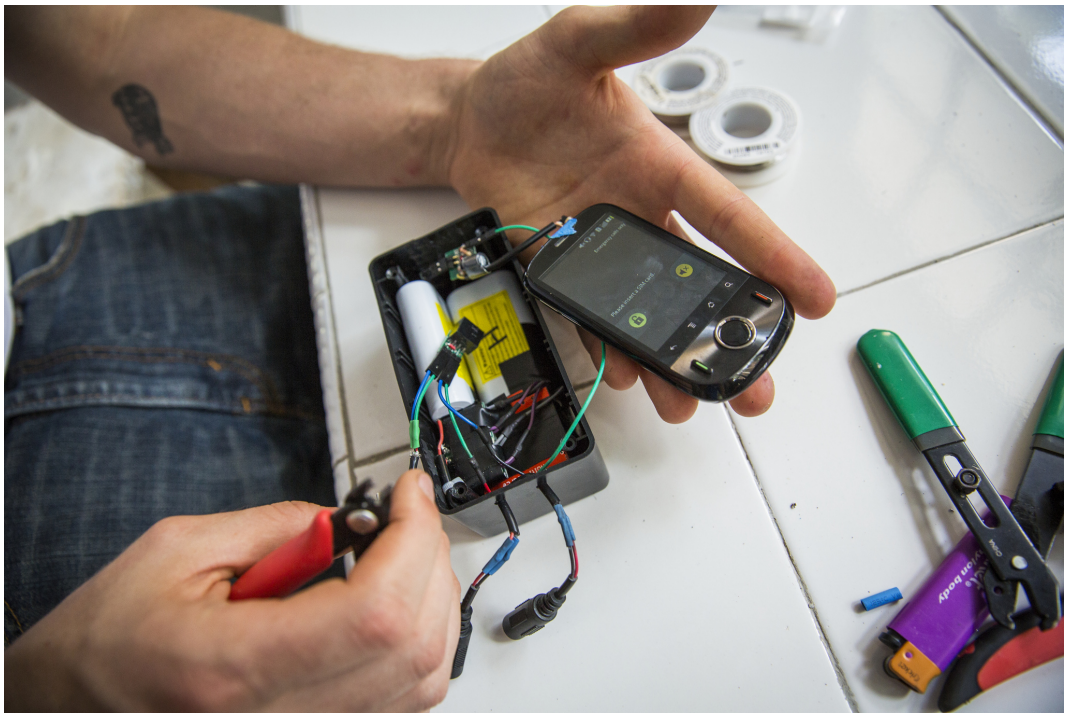


Figure 2.1: Inside of a device: Android phone, batteries, and various electrical components. Some solar charge controller PCBs can be seen in red at the bottom of the enclosure

The solar panel array consists of four petals, each of which has two solar panels on it. The panels are designed and patented by RFCx to work well under low light conditions such as those under the tree canopy. A single petal produces 1.5W of power under laboratory conditions. An individual petal can be seen in Figure 2.2

Figure 2.2: A single solar petal consisting of two panels. Each panel is made from three strips of panel scraps left over from manufacturing at major solar panel companies, connected in series to produce a 1.5V output on each panel

In regards to device enhancements, the principal objective of Project Necromancer is to merge all of the external electronics into a custom printed circuit board. This involves integrating the regulation circuitry and photovoltaic battery charge controller into a single board, and also introducing sensors for temperature, humidity, and power, a microcontroller, and other supporting electronics. The power, temperature, and humidity diagnostics will be sent from the microcontroller to the Android phone via a USB connection. Relaying the collected diagnostic data to the server is possible with the existing framework but is outside of the scope of this project. Instead, the collected data will be displayed for testing and verification purposes. A block diagram of the designed system is shown below in Figure 2.3.

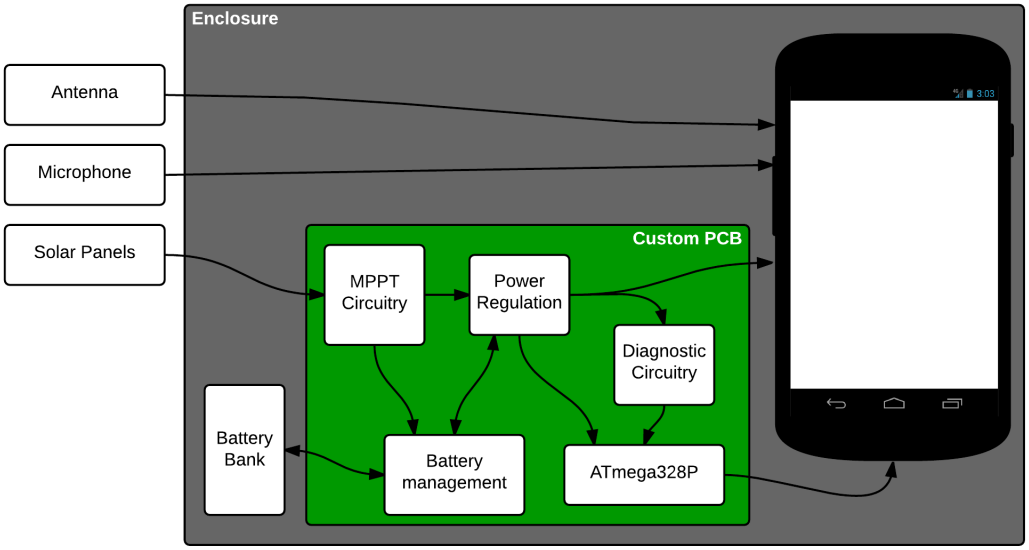


Figure 2.3: High Level Functional Diagram of the Major Components

On the phone, an app called RFCx Sentinel monitors the incoming diagnostics, displaying them to a screen, logging them to a file, and perhaps in the future transmitting them along with the audio data to the server. The RFCx Sentinel Application was developed by our team for the purpose of this project. An actual screenshot of the Sentinel application receiving data from a microcontroller is shown below in Figure 2.4. In addition, although outside of the scope of this project, different audio compression schemes may be investigated if time allows.

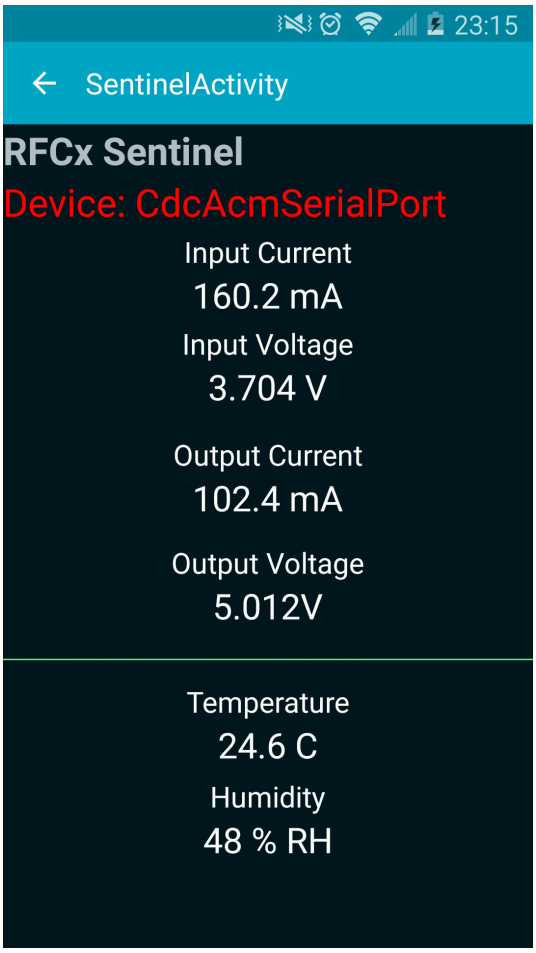


Figure 2.4: Actual RFCx Sentinel App Screenshot

3 Requirements and Functional Specifications

This section is organized by external and internal interfaces, with subsections in each for both hardware and software requirements.

3.1 External Interfaces Requirements

This section addresses the requirements of the enhanced device pertaining to its external interfaces. External interfaces are defined as components of the device interacting with something outside of the device enclosure, including the antenna, microphone, solar panels, and GSM data communication. The possibility of compression is included under external interfaces because it directly relates to the sending of data outside of the device to the server.

3.1.1 Hardware

Requirement 3.1.1.1. The device shall not exceed the existing power consumption and should reduce power consumption by at least 10%.

Requirement 3.1.1.2. The interface with the phone shall consist of a USB connection for power and data, a 3.5mm microphone connection via the standard audio jack, and a connection to the external antenna via the existing antenna cable.

Requirement 3.1.1.3. The device shall use the existing external microphone and antennae

Requirement 3.1.1.4. Shielding techniques should be investigated to reduce the GSM audio interference noted by Topher White in the audio recordings.

Requirement 3.1.1.5. Camouflage color scheme may be used to conceal the device from loggers. Additional camouflage features may be added so long as those features do not cover the solar panels.

Requirement 3.1.1.6. A simple assembly process shall be well documented with illustrations.

To verify **Requirement 3.1.1.6.**. a sample group of people with varying skill levels will be asked to provide feedback on the assembly process and will be timed. The assembly times will be compared with current assembly times.

Requirement 3.1.1.7. The assembly and installation documentation may be translated into Spanish, French, and/or Portuguese.

3.1.2 Software

Requirement 3.1.2.1. The algorithm used to compress the audio signal may be compared to other existing algorithms and may be changed as such depending on the outcomes of comparative testing.

To verify **Requirement 3.1.2.1.**, audio will be recorded and compressed using multiple algorithms including the current algorithm. An infographic hosted on RFCx's website, shown in Figure 2 illustrates the flow of data that is collected from the phone and analyzed on the server.

3.2 Internal Device Requirements

This section addresses the requirements of the enhanced device pertaining to its internal interfaces and software. Internal here is defined as anything inside of the device enclosure, including but not limited to the Android phone, batteries, custom PCB, and supporting electronics.

3.2.1 Hardware

Requirement 3.2.1.1. The PCB shall consist of battery management circuitry, the power regulation circuitry, power interface to the phone, power monitoring circuitry, as well as the microphone interface.

To verify **Requirement 3.2.1.1.**, SPICE will be used to simulate the power charging circuitry to obtain power consumption and efficiency calculations. The manufactured and assembled board will then be subjected to a load test in a lab setting.

Requirement 3.2.1.2. There should be a low power microcontroller included on the PCB that communicates with the Android phone, sending it diagnostic information about components on the PCB. This will be used to monitor power consumption and provide external control of peripherals through the Android Debug Bridge protocol over USB. Extra general purpose pins of the microcontroller should be easily accessible for interfacing with future hardware.

To verify **Requirement 3.2.1.2.**, simple packets will be sent from the microcontroller to the phone. These packets can be verified with debug tools or simple programs on the microcontroller and the phone. A debug serial port may be included as a peripheral to the microcontroller to aid in debugging. Any analog values read by the microcontroller will be verified with shop equipment.

Requirement 3.2.1.3. The monetary cost of the device, not including the solar panels, shall not exceed 25% more than the current device cost, also not including the solar panels. The current cost of the solar panels has been quoted at $200 and the rest of the device has been quoted at $200. Increases in monetary cost above the current cost and below 125% of the current cost are for enhancements in functionality (i.e. the PCB, etc.) and reductions in assembly time.

3.2.2 Software

Requirement 3.2.2.1. The microcontroller software should be capable of bi-directional communication between itself and the Android phone.

Requirement 3.2.2.2. The microcontroller software should be capable of communicating power usage diagnostics to the phone over the ADB protocol.

Requirement 3.2.2.3. An Android companion application may be capable of reporting power diagnostics from the microcontroller for testing and debugging.

Requirement 3.2.2.4. There shall be a header on the PCB to program the microcontroller.

4 Component Justification

This section describes each component of the PCB in detail and gives justifications for their selection and a discussion of alternative choices that were considered. Along with the individual requirements of each device, a driving factor in device selection was the physical package of the IC; there should be no component that the end user cannot afix to the PCB without common soldering equipment.

The following is a list of the main ICs that comprise the PCB and are outlined in this section:

•**Atmel ATMega328P** : 8-bit AVR microcontroller (Appendix E.2)

•**SPV1040 Max Point Power Tracker** : Solar power controller and lithium-ion battery charger (Appendix E.12)

•**ADS1015** : External 4-input 12-bit Analog-to-Digital Converter, I^2C communication (Appendix E.1)

•**LM75BD** : Ultra Low Power Temperature Sensor,$\pm 2°C$, I^2C communication (Appendix E.7)

•**(Optional) HIH6130** : 14-bit resolution Humidity and Temperature Sensor with accuracy of ±5% Relative Humidity and ±1°C, I^2C communication (Appendix E.5)

•**FT230X** : FTDI USB-UART interface for USB serial communication (Appendix E.13)

•**BQ2057CTS** : Linear Battery Charge Management IC (Appendix E.3)

•**LM61428** : Simple Switcher Boost Controller IC (Appendix E.8)

•**SM72238** : Micropower Fixed 3.3V LDO (Appendix E.11)

•**LTC6800** : Rail-To-Rail Input and Output Instrumentation Amplifier (Appendix E.10)

•**LTC4412** : Low loss powerpath controller (Appendix E.9)

4.1 Microcontroller

The ATmega328P microcontroller (Appendix E.2) was chosen as the microcontroller for the PCB. Many aspects were considered when choosing a microcontroller, including power consumption, available interfaces, availability of code samples, community support, ease of programming, and package size.

To this end, the ATmega328P is a logical choice; it is the microcontroller used by the popular Arduino platform, which means there are a multitude of code samples to reference for most any project. In addition, the 328P has a *picopower* deep sleep mode, which consumes under 0.1μA of current. Unlike many options, the 328P has an easy to solder TQFP package. It is also is easily programmed by the use of an ICSP interface and does not require expensive hardware programmers or debuggers, as is the case for the Freescale KL25Z, which was also considered but requires a JTAG programmer.

Another microcontroller considered was the Texas Instruments 16-bit RISC MSP430. This microcontroller is the most expensive, draws the highest current in active and sleep modes, and requires an external tool to program.

USB communication was another concern when choosing a microcontroller. Similar MCUs that included on-board USB interfaces were also considered, but none could be found that offered the simple package, deep sleep mode, and I^2C communication. I^2C communication is needed to communicate with the analog-to-digital converter and temperature/humidity sensors.

4.2 MPPT

The SPV1040 Max Point Power Tracker (MPPT) (Appendix E.12) was chosen to perform the solar charge management. A driving factor in choosing the SPV1040 was that RFCx had already established a supplier and had many on-hand to ship us for prototyping. In addition, it has the advantage of being simpler to use than similar chips, like the Solar Magic SM72442, in that no I^2C communication is necessary to set and configure registers to perform the same function. Finally, the currently used MCP73871 chip (currently on an off-the-shelf Adafruit board) fails in both the physical package being difficult to assemble by hand (QFN package) when used alone, and being meant for only single-cell Lithium-Ion batteries. In addition, the MCP73871 does not use a true MPPT algorithm, but rather an approximation.

4.3 MPPT Configuration

The MPPT configuration makes a large difference in many aspects of the selection of other components. First, each MPPT can only handle 1.8A of current. Also, the input voltage of each MPPT must be much less than the desired output voltage for them to function properly. Depending on the configuration, the number of MPPT ICs can vary from 1 to 8, increasing cost, and the voltage and current characterstics vary depending on whether they are connected in series or in parallel.

The following figure (Figure 4.1) describes the chosen MPPT configuration. In this configuration, each petal, which consists of two 1.5V panels, has its panels connected in series, giving an output of 3.0V, at 1.5W. This results in a

current of 500mA for each MPPT, which is well below the maximum. Each petal (four total) is then connected to its own MPPT, which is set to boost to 5.2V. The MPPTs are then connected in parallel.

Connecting multiple regulators in parallel can sometimes cause problems if one source is outputting a slightly smaller or larger voltage than the others. In most cases, the use of ballast resistors solves this problem. The footprint for a ballast resistor was chosen to be included in series with the output of each MPPT. The footprint will be first populated with a 0Ω resistor, and the current will be monitored from each MPPT to ensure they are sharing the load correctly. If they are not, the footprint will be populated with a $25m\Omega$ ballast resistor, and the same test will be performed.

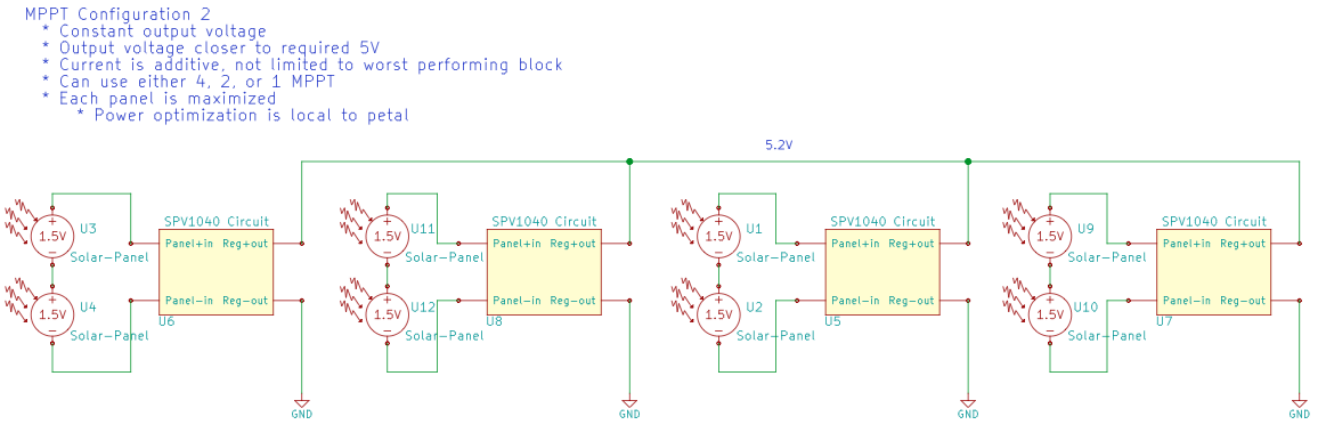


Figure 4.1: Four MPPTs are connected in parallel with the input being the series connection of each panel within a single petal

This configuration offers many benefits. First, the panels on a single petal can be either connected in series or in parallel. Since the SPV1040 can handle up to 1.8A, even when connected in parallel the petal will only produce a maximum of 1A. The input voltage to the MPPT can be either 1.5V (panels in parallel) or 3.0V (series), and the MPPT's internal boost regulator will still produce 5.2V. This allows for testing configurations where all petals have their panels connected in parallel, all in series, or a combination of both under different lighting conditions to see which performs the best without modifying the PCB in any way.

In addition, this configuration localizes the power tracking to an individual petal. Rather than maximizing the entire array, localizing to each petal allows for some petals to be in direct sunlight while others are in shade and each MPPT will react accoringly.

Furthermore, this configuration could be easily adapted to using less MPPTs if needed (to reduce cost, for example). To use two MPPTs, for example, instead of four, two of the ICs would simply not be populated and pairs of petals would be connected in parallel externally and connected to the two populated MPPTs. This is shown in Figure 4.3.

Finally, as required by the SPV1040, the input voltage will always be much less than the output in this configuration. The input voltage will be either 1.5V (parallel panels) or 3.0V (series), well below the 5.2V output.

Two other configurations that were considered are shown below. In the first alterate configuration, in Figure 4.2, two pairs of MPPTs connected in series are then connected in parallel. This has many drawbacks, including requiring a higher output voltage when it is not needed, and requiring the use of diodes to prevent current from backfeeding into the other MPPTs.

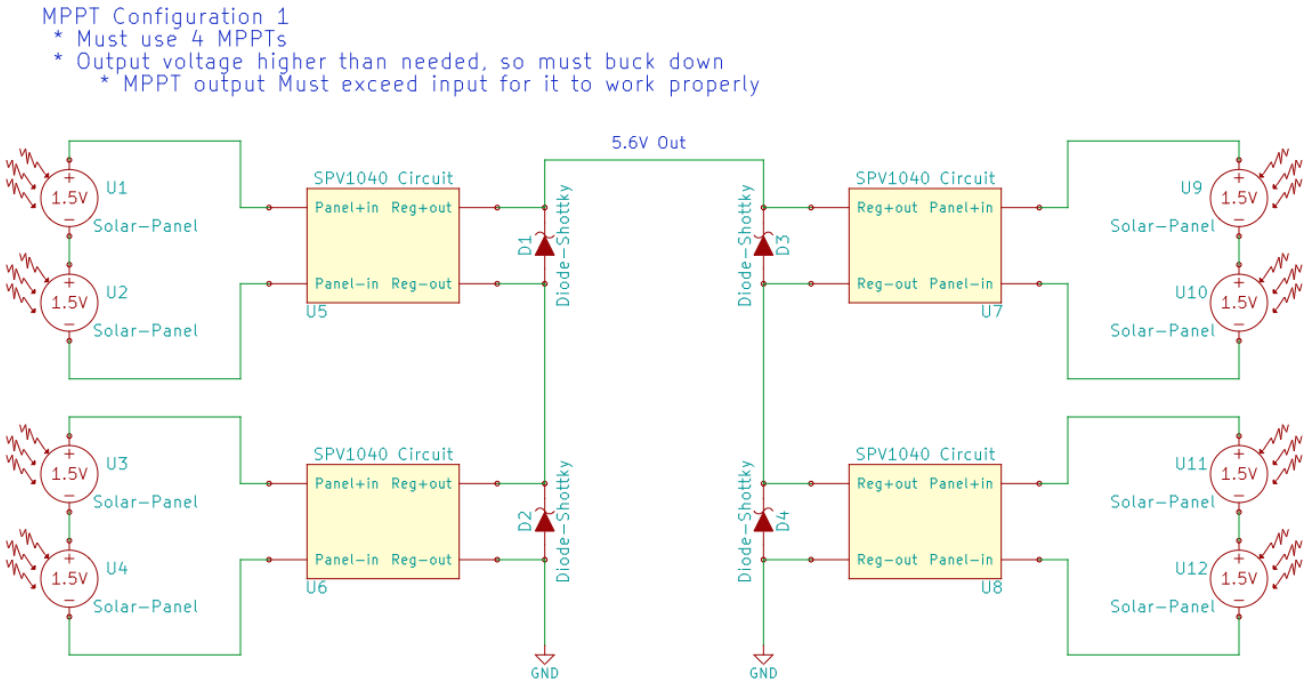


Figure 4.2: Two pairs of MPPTs are first connected in series, then in parallel, requiring the use of feedback diode protection and resulting in a higher than needed output voltage which must then be bucked down

The second alternate configuration, in Figure 4.3, is really the same as the chosen one, but using two MPPTs instead of four as described above.

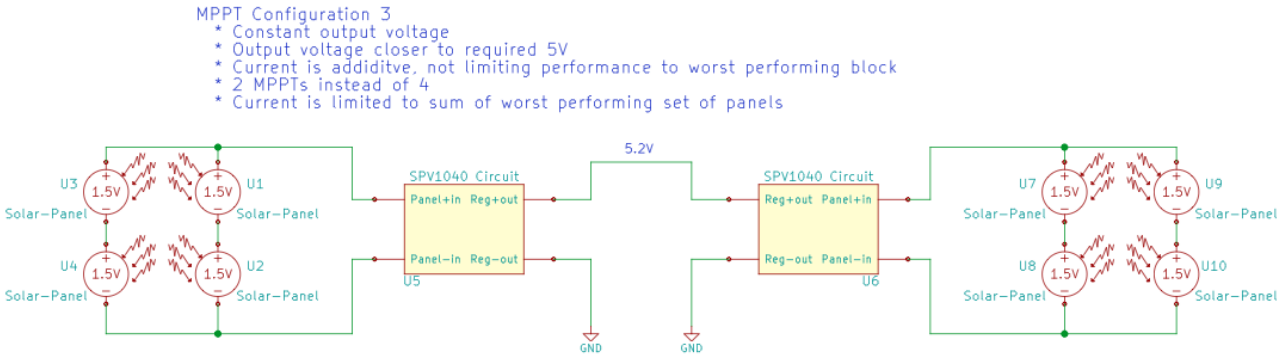


Figure 4.3: Chosen configuration but with two MPPTs instead of four. Results in less control but saves the cost of two ICs and the supporting circuitry of each

4.4 ADC

The ADS1015 external ADC (Appendix E.1) was chosen to perform analog-to-digital conversions for the power sensing. The ATmega328P does not have a strong internal ADC, being only 10-bit resolution, and the ADS1015 is easy to communicate with via I^2C. The ADS1015 consumes less than 150μA, has 12-bit resolution, and supports

four single-ended analog inputs, which will be used to monitor input current, input voltage, output current, and output voltage on the PCB.

I^2C supports multiple devices in parallel, so only two of the ATmega328P's six analog pins are used. This frees the other four for future use, which is a major advantage of using an external ADC.

4.5 Temperature and Humidity Sensor

The LM75BD (Appendix E.7) was chosen as the temperature sensor. The LM75 is a super low power temperature sensor with an accuracy of $\pm 2°C$. It uses less than 300uA during peak transmission, and costs under $1.

An alternative sensor prodiving both temperature and humidity measurements, the HIH6130 (Appendix E.5), was also considered. Also using I^2C and also being low-power, the only real drawback is the price, at nearly $14. For this reason the LM75BD was chosen, but the footprint for the HIH6130 will be included but unpopulated on the board as an option to the end user.

Other lower-cost humidity and temperature sensors were investigated, such as the DHT11 and DHT22. The DHT11 is affordable, at around $5, but has a limited range of 0-50$°C$. The DHT22 has the range required and is accurate, but costs almost as much as as the HIH6130 at around $9. Both the DHT11 and DHT22 are not I^2C, but rather use the one-wire-interface which requires elaborate timing to get the correct data, and take up another pin on the microcontroller. In addition, Adafruit and Amazon were the only places that stock the DHT22, neither of which offer the reliability or quantity of sources like Digikey or Mouser.

In addition to other I^2C temperature sensors, traditional analog sensors were also considered. The popular LM35 analog sensor, despite having less features than the LM75BD, costs over twice as much, at $1.75.

The LM75BD is accurate within $\pm 3°C$ from -25 to $+100°C$ and within $\pm 2°C$ from -55 to $+125°C$. The HIH6130 can operate in the temperature range of $-25°C$ to $85°C$ and from 10% to 90% Relative Humidity (RH) with an accuracy of $\pm 1°C$ and $\pm 5\%$ RH.

4.6 FTDI

The FT230X FTDI chip (Appendix E.13) was chosen for the UART to USB interface between the microcontroller and the Android phone. FTDI is widely used and supported, and, unlike similar devices like the Arduino's CDCACM, does not require another microcontroller to be added, configured, and flashed with software. In addition, the RFCx Sentinel application, which uses the open source usb-serial-for-android library, readily supports FTDI. Serial USB communication has already been achieved using an ATmega328P MCU, a similar FTDI chip, and the Sentinel app.

4.7 Battery Management

Battery management is important for the ability to reclaim and potentially fix a damaged battery, and also to prevent potential fire hazards. Being in a hot climate and surrounded by ample fuel for a large-scale forest fire, battery management was chosen to be included in the design. The bq2057CTS battery charge management IC (Appendix E.3) was thus chosen to control the charging of the Lithium-Ion batteries. It is designed for both single and dual-cell battery packs, and combines current and voltage regulation, charge status indication, and charge rate auto compensation.

Although battery management *can* be done in software with the microcontroller, the amount of work required to do so is disproportional to any benefits you may gain. In fact, by performing battery management on the microcontroller, those writing the software take responsibility for any damage to the batteries, or even substantial damage such as a fire. This is in contrast to using an off-the-shelf, proven solution such as the bq2057CTS.

4.8 Power Path Management

To manage the power selection between the solar panels and the batteries, a power path management solution was required. This was required because while the batteries are being charged the rest of the circuit should not draw from them. If the rest of the circuit draws from them while they are being charged, the battery charge management IC could get confused and damage the batteries by over charging or under charging. A diode-ORed configuration, in which power sources are ORed together via a near ideal diode($0.02V_f$), selects the source with the higher voltage. The LTC4412 (Appendix E.9) was chosen to perform this power path management. The ideal diodes are implemented externally with a FZT788B P-Channel MOSFET (Appendix E.4). This IC also has a very low quiescent current, below $1.15\mu A$.

4.9 Power Regulation

The LMR61428(Appendix E.8) Step-Up Voltage regulator was chosen because of its price, package, and performance. It is important for all of the ICs on this board to be in a package that the end user can assemble with without the aid of more advanced PCB assembly equipment such as solder paste/stencils and ovens. Also, some of the other buck/boost controllers required a much higher input voltage when the load current was around 500mA. The LMR61428 requires less than 2V, which is possible even when drawing from the batteries. Since this is design is for a remote device powered from batteries and solar power, quiecent current is a concern as well. The LM61428 only draws 80mA during operation.

The Texas Instruments SM72238 Micropower fixed 3.3V linear voltage regulator was chosen to regulate the 3.3V line because it has a very low dropout voltage of around 380mV at the max current draw of 100mA.

4.10 Current Sensing

The LTC6800 precision instrumentation amplifier (Appendix E.10) was chosen to perform current sensing. This instrumentation amplifier operates rail-to-rail on the input and output, which provides a larger window to sense the voltage across the sense resistor. A sense resistor value of 0.010Ω was chosen to provide minimal power dissipation. Although very small, since the leakage current into the opamp is also extremely small (on the order of 1nA), the trace resistance, which is approximately 0.035Ω, is not a concern. The LTC6800 also draws very little current. Under no load the LTC6800 draws a maximum of 1.3mA.

4.11 PCB Size

Two options were considered for the physical form of the PCB. The first was only large enough for the components on the board, and would be mounted in the enclosure alongside the batteries and phone. The second design, which was chosen, uses a PCB slightly larger than the size of the phone itself. This allows for the entire phone and PCB to be attached and easily removed/inserted into the enclosure as one piece. The PCB design is shown in Section 5.1.8.

4.12 Heat Dissipation

Multiple solutions were explored for dissipating heat inside the enclosure. These include drilling holes into the enclosure, putting fans in the enclosure to control airflow in and out of the enclosure, and using a heat sink.

Using a heat sink was chosen as the best design alternative for heat dissipation. Drilling holes in the enclosure can allow water, insects and dust to enter the enclosure. A wire mesh over the holes could be used to keep insects out of the enclosure, but water and dust could still get into the enclosure and damage the electronics. Running fans in the device creates a sound in a similar frequency range to the chainsaw noise the device is trying to detect. This can cause false detection of a chainsaw by the RFCx server.

A heat sink also requires a hole to be drilled into the enclosure for the heat sink to reach outside of the enclosure. This opening will be sealed with caulk or by other methods to prevent water, insects and dust from entering the enclosure. The battery cover on the phone will be removed and thermal paste will be used between the phone

battery and the PBC, and between the PCB and the heat sink. This will allow heat to dissipate out of the enclosure and help keep the phone attached to the PCB and the PCB attached to the heat sink. Vias in the PCB will help heat to transfer through from the phone to the heatsink. The heatsink implementation is shown in Section 5.2.1.

5 System Design

5.1 Hardware Design

5.1.1 Power Sources

The system is powered from eight solar cells that have been manufactured by RFCx. These solar panels have been reported to have an open circuit voltage of 1.6V and produce an average of 0.75W each under laboratory conditions. The other power sources on board are two dual-cell 4400mA/Hr lithium-ion (Li-ion) battery packs. As discussed in the MPPT configuration justification, the solar panels will be configured in a way that a pair on an individual petal may be wired in either series or parallel. The pairs will be wired how the end user would like them and the remaining wires will be connected to a terminal block with screw/leaf spring wire termination on the PCB, which will be powering the MPPT controllers. This is illustrated in Figure 5.1.

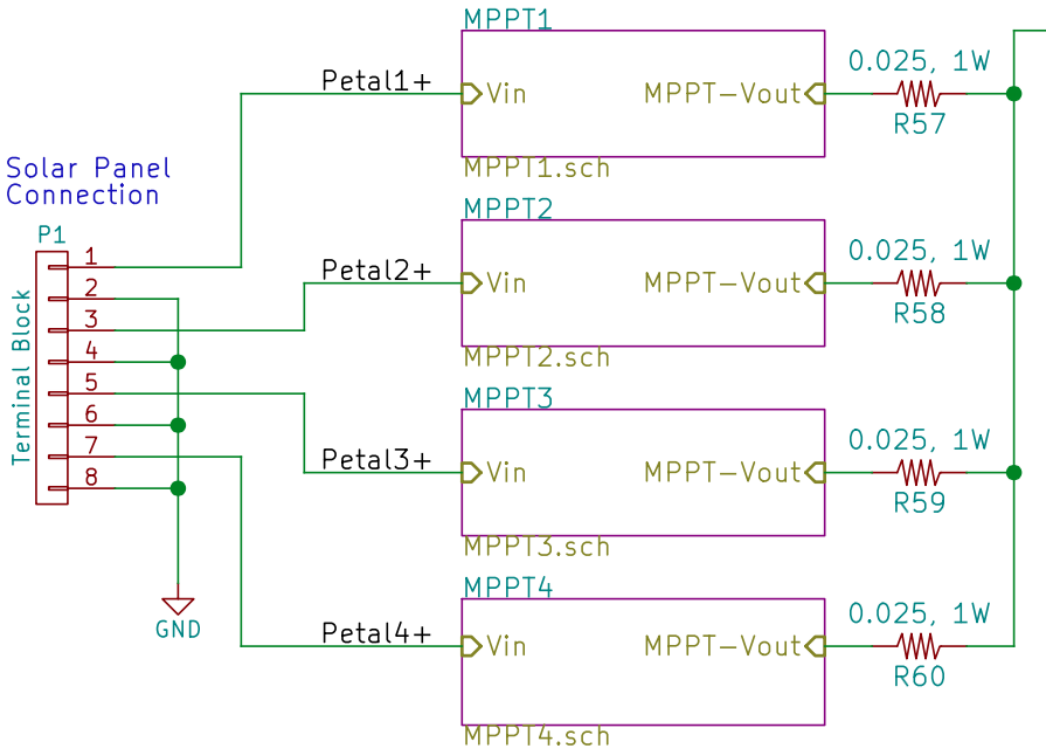


Figure 5.1: MPPT and Solar Panel Interface

With this configuration a configurable output voltage of 5.2V, the maximum of the SPV1040, will be obtained at an average of about 1.15A, yeilding an input power of about 6W.

5.1.2 Solar Charging

A Max Point Power Tracker (MPPT) increases efficiency by adjusting the current and voltage of the output to achieve the maximum power. The SPV1040 will regulate the solar cell voltage up to 5.2V. From the MPPT the battery management circuitry, the 3.3V line, and the 5V line is powered. The MPPT circuit is shown in Figure 5.2.

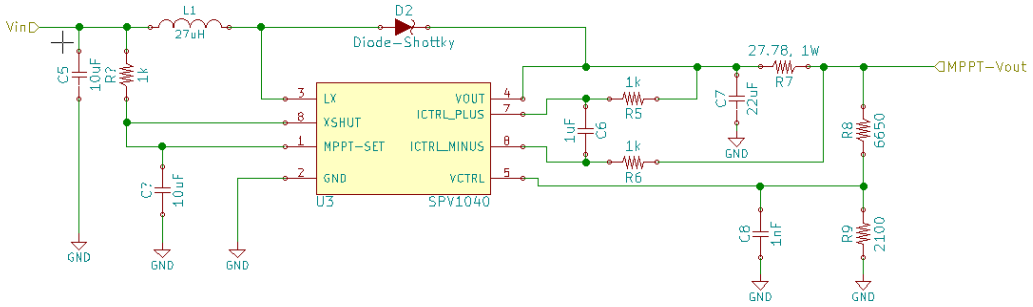


Figure 5.2: Single SPV1040 MPPT Circuit

The input capacitor's purpose is to reduce voltage ripple. It was calculated that this capacitor must be greater than $14.3\mu F$. The time constant of the resistor and capacitor on the *MPPT SET* pin of the SPV1040 IC must be less than or equal to the maximum power point tracking time ($T_{MPPT} \cong 1ms$). It is recommended that the resistor be set to $1k\Omega$. This means that the capacitor must be less than $10\mu F$. The inductor selected was based on the maximum input voltage and current produced by the solar petals. The inductor value was calculated to being greater than or equal to $8.5\mu H$. The saturation current for the inductor must also be greater than 1.8A, which is the limit of the SPV1040. The output voltage capacitor was chosen based on the short circuit current of the solar petals and the maximum voltage ripple. The output voltage capacitor must be greater than $142\mu F$ so $150\mu F$ was chosen. This was calculated with a maximum output ripple voltage of 0.01V. There are two resistors that divide the output of the SPV1040 and provide feedback to the controller. These resistors were calculated to be $76.8k\Omega$ and $24.3k\Omega$. This divides the desired maximum output voltage of 5.2V to 1.25 at the voltage sensing pin. The capacitor on the V_{CTRL} pin of the SPV1040 is needed to reject noise sensed by the pin. This was calculated to be around $45.2\mu F$. The voltage drop across the sense resistor is compared to the 50mV internal reference. The sense resistor is calculated to be around $27m\Omega$. Since there may be some noise from the switching, there is a filter on the sense pins, *Ictrl_+* and *Ictrl_-*. It is recomended that the inline resistors are $1k\Omega$ and the capacitor between the two is $1\mu F$. A Schottky diode is placed between the *Lx* and *Vout* pins of the SPV1040. The *Diodes Incorporated* 1N5819HW Schottky diode meets the suggested criteria. It has a forward voltage drop of 450mV and a non repetitive peak current of 25A. The calculations for these components can be found in Appendix D.3.

5.1.3 Battery Management

The two dual-cell Lithium-Ion batteries are each managed by a bq2057CTS. Figure 5.3 shows the battery management circuitry for one of the 2-cell battery packs. This IC provides voltage and current regulation, battery conditioning, temperature monitoring, charge termination, charge status indication, as well as charge-rate compensation. The charging is started in the constant voltage phase and stops when the current falls bellow the current termination threshold. The bq2057 inhibits charging until the battery temperature is in the user defined range. This range is defined in this circuit with the $49.9k\Omega$ resistor *R24* as an upper bound of $50°C$. The PNP BJT pass transistor controls voltage and current regulation as well as providing protection from backfed current. Only the upper temperature bound is set because the device is not intended for cold environments. The charge rate compensation is set by resistors *R18* and *R22*. These values are outlined in the datasheet (Appendix E.3) to set a proprietary charge compensation algorithm. The automatic charge compensation helps to charge the batteries faster and compensate for the battery pack's internal impedance.

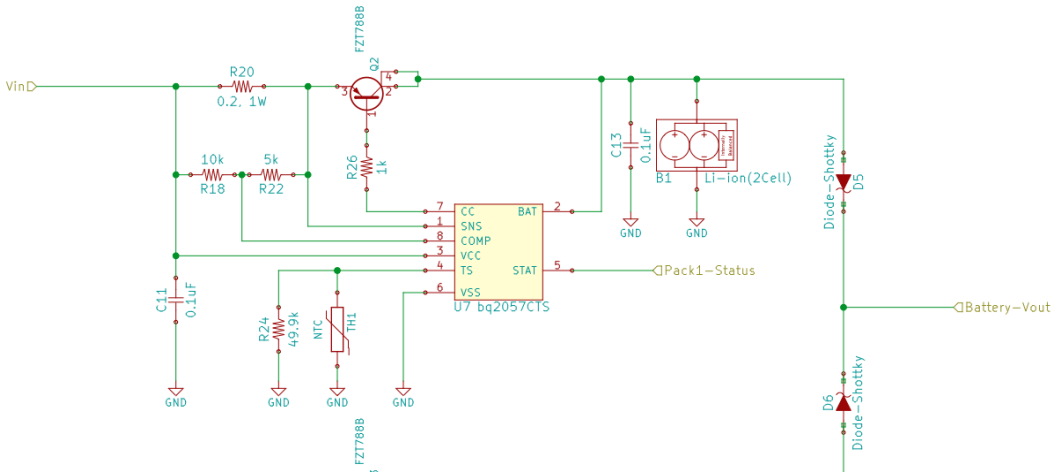


Figure 5.3: Battery Management Circuitry

The calculations for the component selection for the battery management circuitry is included in Appendix D.1.

5.1.4 Voltage Regulation

A diode-ORed power path management circuit was used to switch the power source that feeds the 3.3V and 5V regulation circuitry. This circuit is shown in figure 5.4.

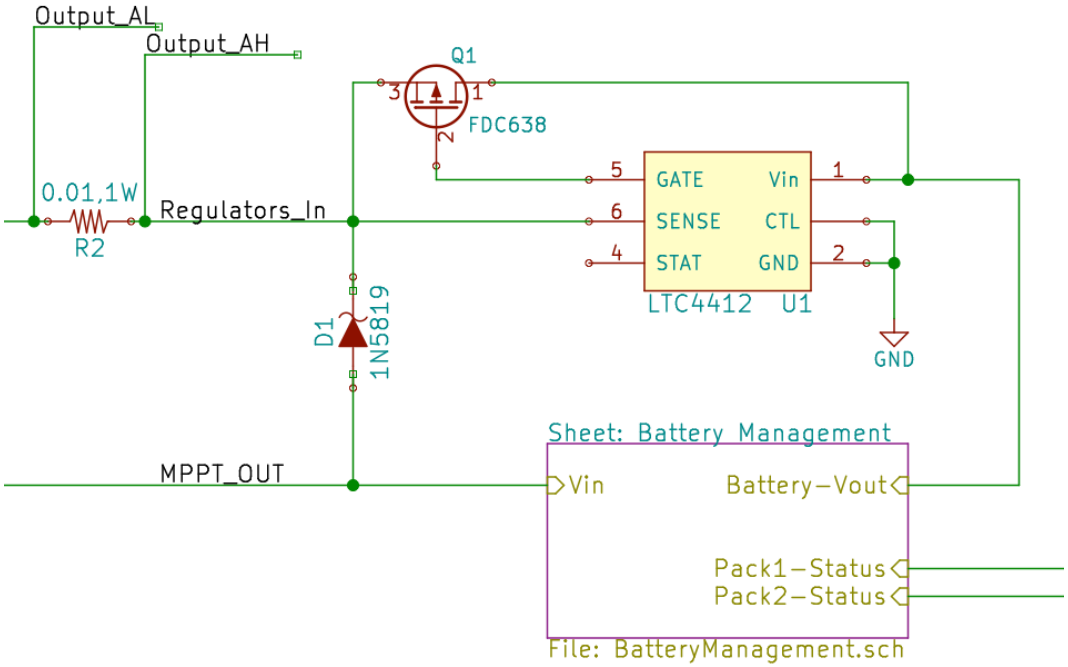


Figure 5.4: Diode Ored Circuit

This circuit provides power path switching in under 100 μs, and there is virtually no dip in the output voltage during the switch. A dip of approximately 500mV was observed when using a transistor alone to simply switch between the two sources, as shown in the simulation figure in Appendix A.10. With the diode-ORed topology, the switch is clean, as seen in the simulation figure in Appendix A.11.

The LM61428 is a step-up voltage regulator that is configured to boost the input voltage to 5V at an output current of up to 1A. The output voltage of 5V is used to power the Android phone.

The layout of the 5V boost regulator can be seen in figure 5.5.

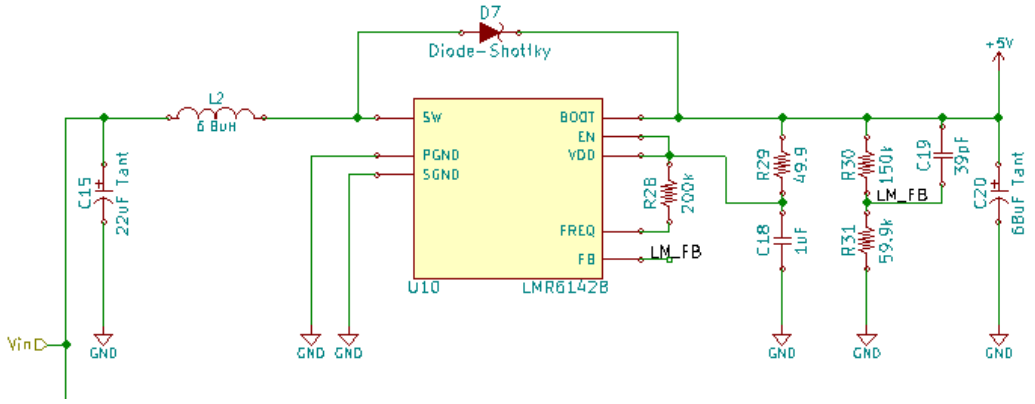


Figure 5.5: 5V Voltage Regulation

A Texas Instruments SM72238 Micropower fixed 3.3V LDO is used to regulate the 3.3V line. The SM72238 has very low dropout, which is in range for the need of this application. The only external components required by the LDO are two low ESR 1uF capacitors on the input and output. The circuit can be seen in figure 5.6.

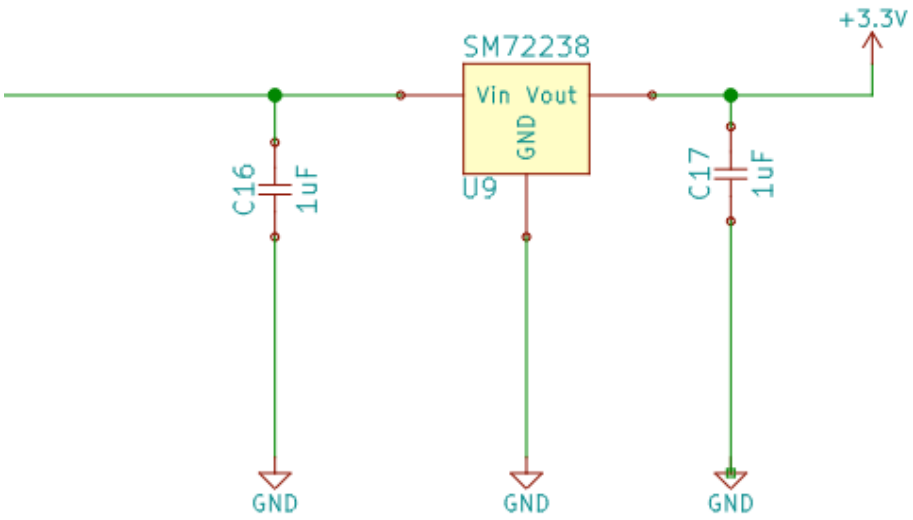


Figure 5.6: 3.3V Voltage Regulation

5.1.5 Input and Output Current and Voltage Sensing

The ADS1015 is a 12-bit ADC that can communicate over I^2C with the microcontroller. This ADC will operate in single shot mode and then go back to sleep. The ADS1015 uses a $\delta\sigma$ ADC and can perform conversions at rates

up to 3300 samples per second. The microcontroller will initiate a conversion over I^2C and be able to send the diagnostic to the phone for further processing. A code snippet can be seen below in Section 5.3.
The LTC6800 instrumentation amplifier is used to sense the voltage across a known sense resistor. The input current sense circuit is configured to have a gain of around 80.5. The sense resistor has been selected to be 0.01Ω. With this configuration the ADC will be able to sense the current with a resolution of 1mA/bit. The calculations for this section can be seen in Appendix D.2. The configuration of the current sense amplifier can be seen in Figure 5.7. The input and output voltage are sensed with simple resistor dividers that are fed to the ADC.

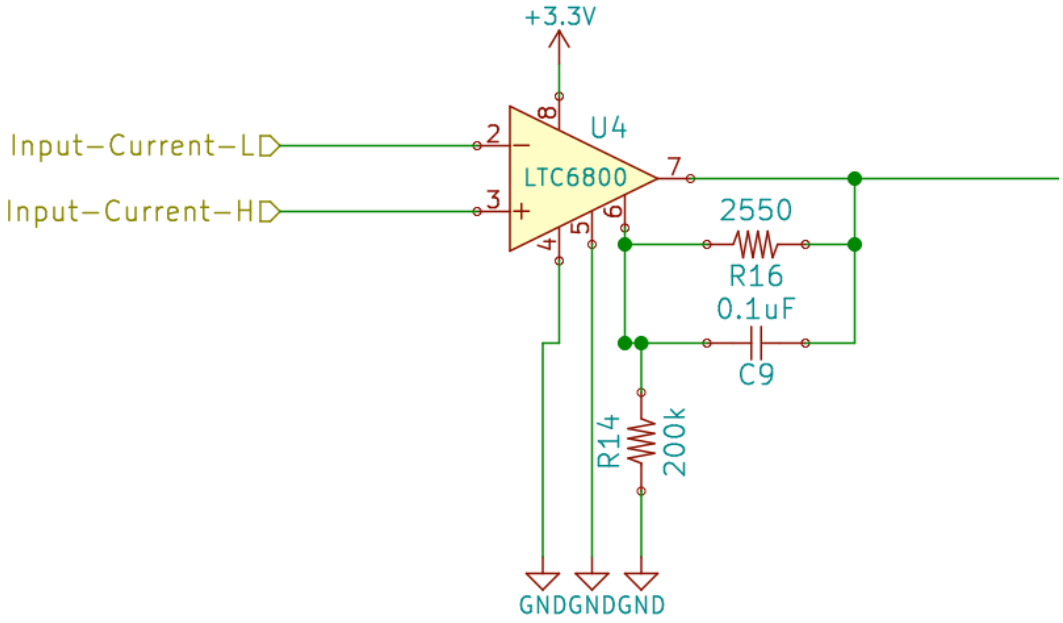


Figure 5.7: Current Sense Amplifier

5.1.6 Microcontroller

The only supporting circuitry for the ATmega328P is the 6-pin in-circuit serial programming (ICSP) port, a 32kHz external crystal, and external pull up resistors on the I^2C communication lines. The 32kHz crystal is included for the real time clock. The real time clock is used so that the ATmega328P microcontroller can go into a sleep mode and wake up later to conserve power usage. The ATmega328P also requires an FTDI IC to communicate with the phone over USB. The microcontroller and surrounding circuitry is shown below in Figure 5.8.

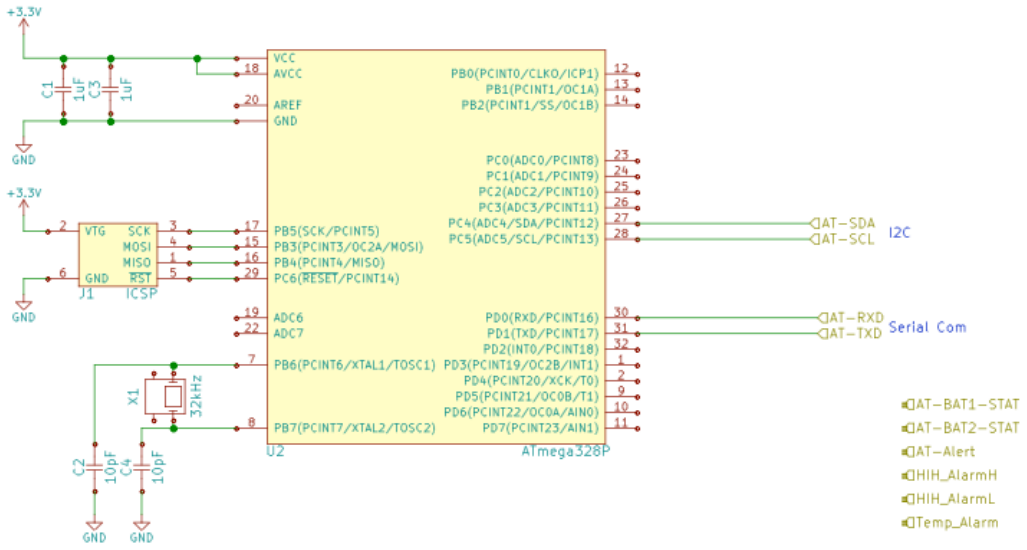


Figure 5.8: Microcontroller

5.1.7 Temperature and Humidity

Both the temperature sensor (LM75BD, Appendix E.7) and humidity/temperature sensor(HIH6130, Appendix E.5) require minimum supporting circuitry to operate due to their internal clocks, internal analog-to-digital converters, and internal memory. The LM75, however, still requires a resistor network to set the I^2C address bits associated with the chip. As seen below in Figure 5.9, resistors R51 to R56 are used to set the address bits. For each input, either the pull-up or pull-down resistor will be populated and the other will be populated with a 0Ω resistor to set the bit to either a logic 1 or 0.

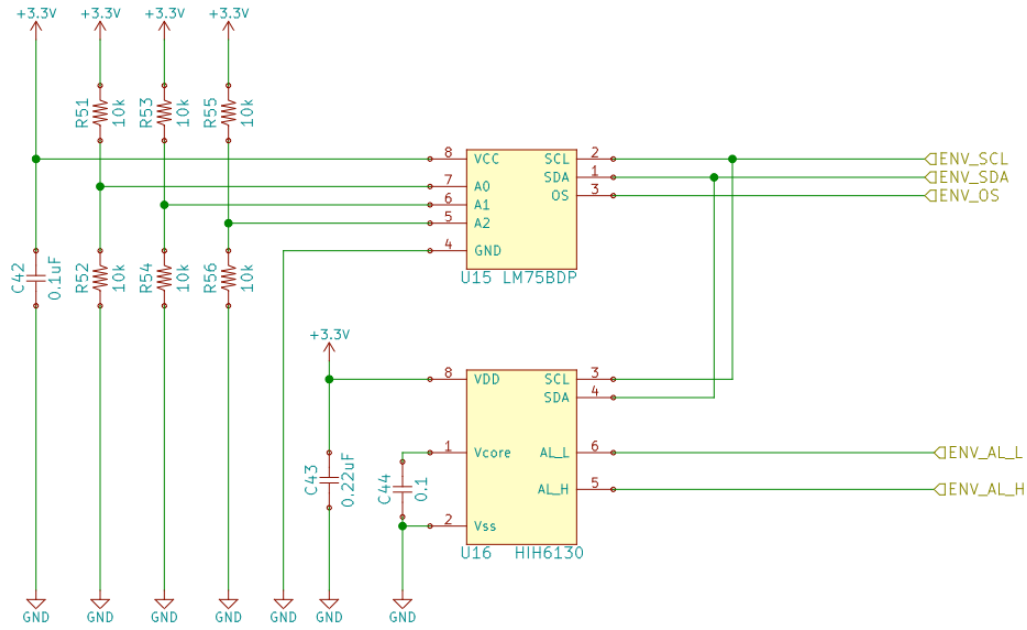


Figure 5.9: Temperature and Humidity Sensors

5.1.8 PCB

The dimensions of the chosen PCB size is shown in Figure 5.10. This size accomodates both the phone currently used, the Huawei Ideos U8180, with dimensions of *4.09"* x *2.16"* x *0.54"*, as well as larger Android phones up to *5"* x *2.8"* x *0.625"*. The dimensions of the phone relative to the PCB is shown in Figure 5.11.

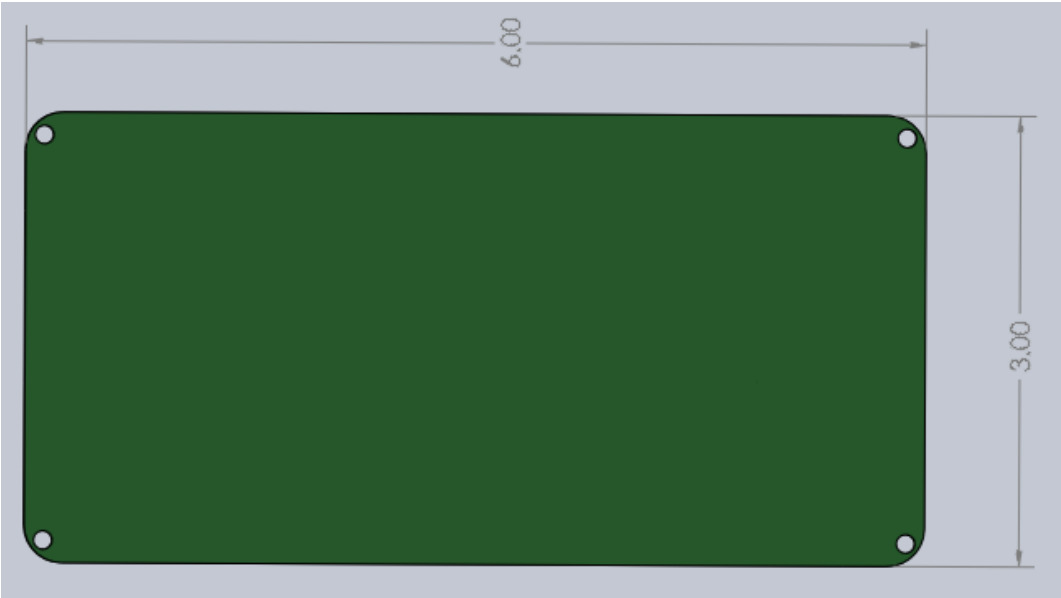


Figure 5.10: PCB Dimensions

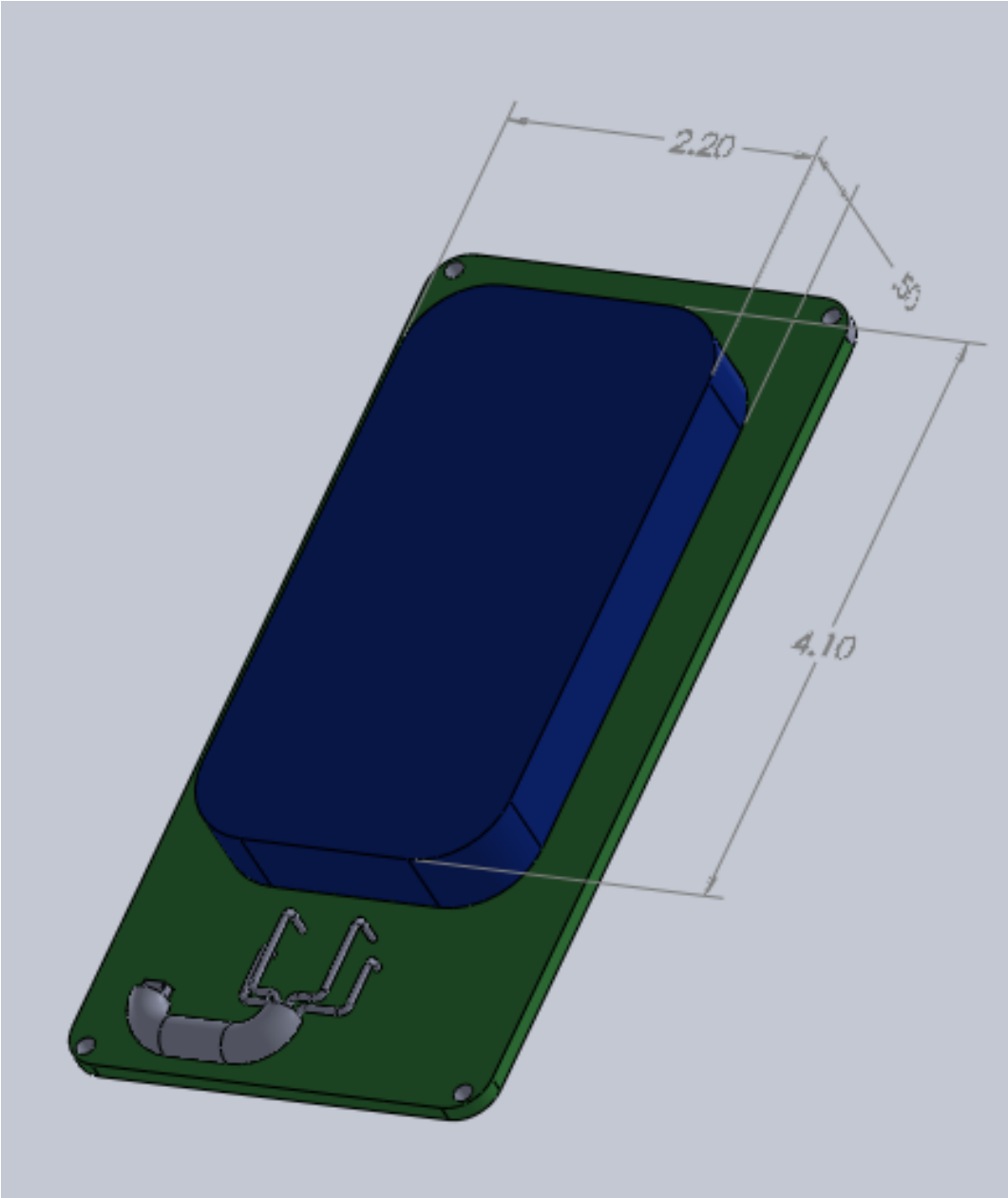


Figure 5.11: PCB Dimensions

5.1.9 Connectors

The USB connection consists of a standard USB Micro A connector, soldered directly onto the board and is secured to the board as a strain-relief mechanism, which is shown in Figure 5.12. A standard 3.5mm audio cable is used to connect the microphone. The existing standard GSM antenna adapter is used to interface with the phone's onboard GSM module.

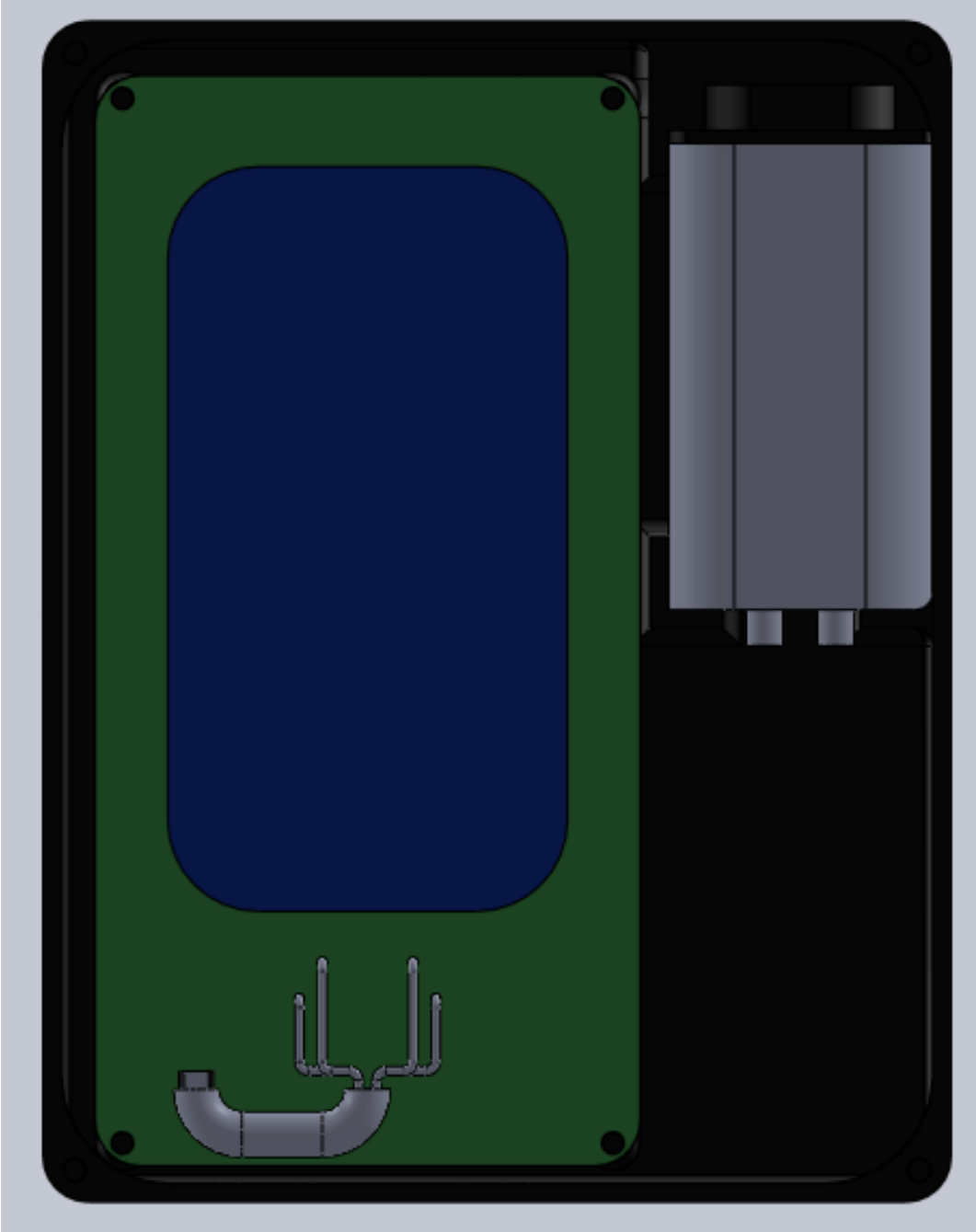


Figure 5.12: USB Micro A Connector attached to the PCB with strain relief: A small flexible cable

5.1.10 GSM Interference

The use of X2Y TDMA filtering capacitors will be tested to reduce the noise produced by the stopping and starting of GSM packets, which is in the audible range at 217Hz. These capacitors are essentially a three node capacitive circuit that has two closely matched, low inductance capacitors in a single package. Manufacturer tests show a nearly flat response for a GSM handset close to an audio circuit. A pass-through 3.5mm microphone interface will be included to test the X2Y capacitors. The pass through circuit can be seen in figure 5.13.

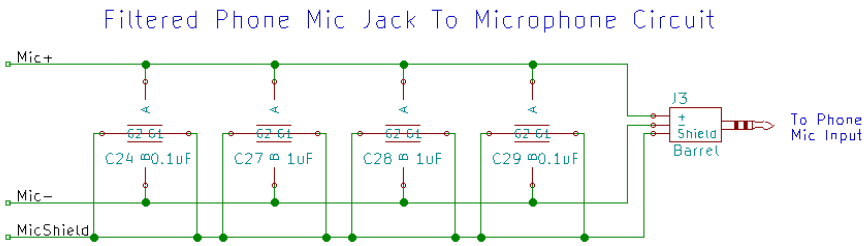


Figure 5.13: Pass Through Microphone Filter Circuit

In addition, a heatsink will be attached to a large ground plane on one side of the board, and the phone will be attached to the same ground plane, to provide some shielding.

5.1.11 Enclosure

A new prototype enclosure is not a hard requirement, but is an essential part of prototyping the entire device, especially given the changes being made to the components inside the enclosure. The designed enclosure is shown in Figure 5.14 and Figure 5.15 below.

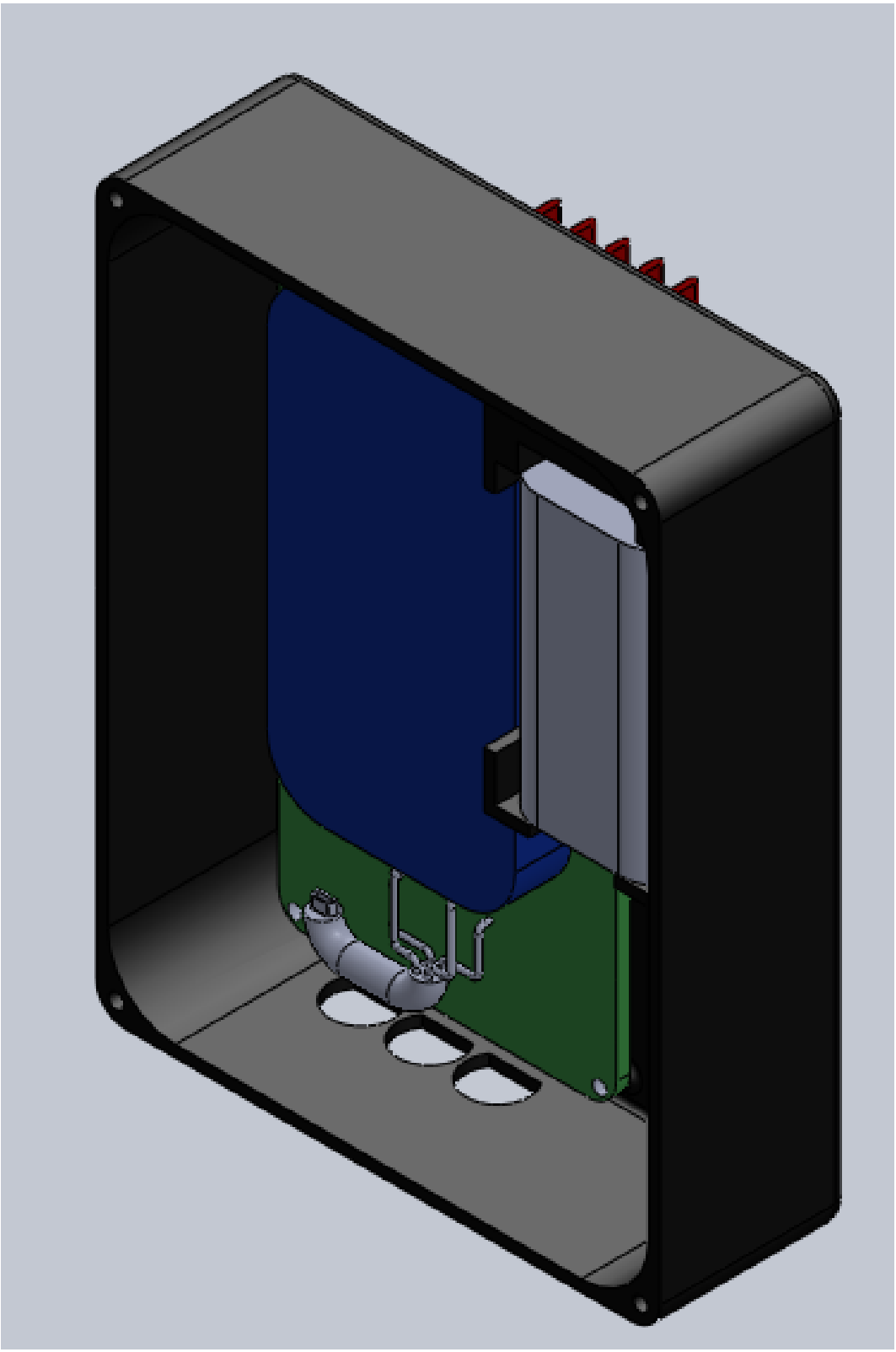


Figure 5.14: Front View of the Enclosure

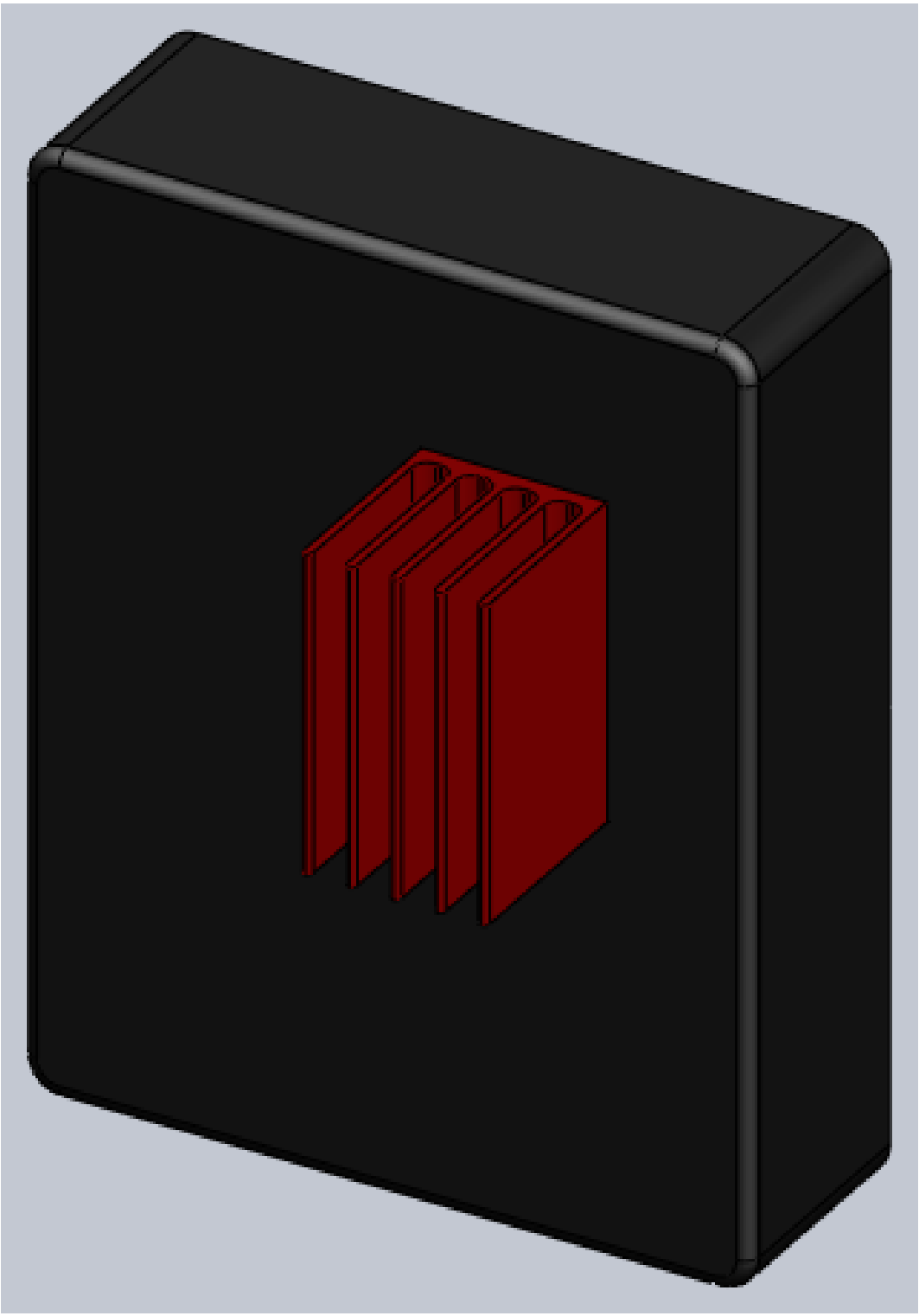


Figure 5.15: Back View of the Enclosure

5.2 Hardware Implementation

5.2.1 Heatsink

A heatsink, as described in Section 4.12 is shown in Figure 5.16. The current design calls for a rectangular opening to be cut into the front face of the enclosure so that the heatsink may pass through. The opening will be sealed with foam or rubber insulation.

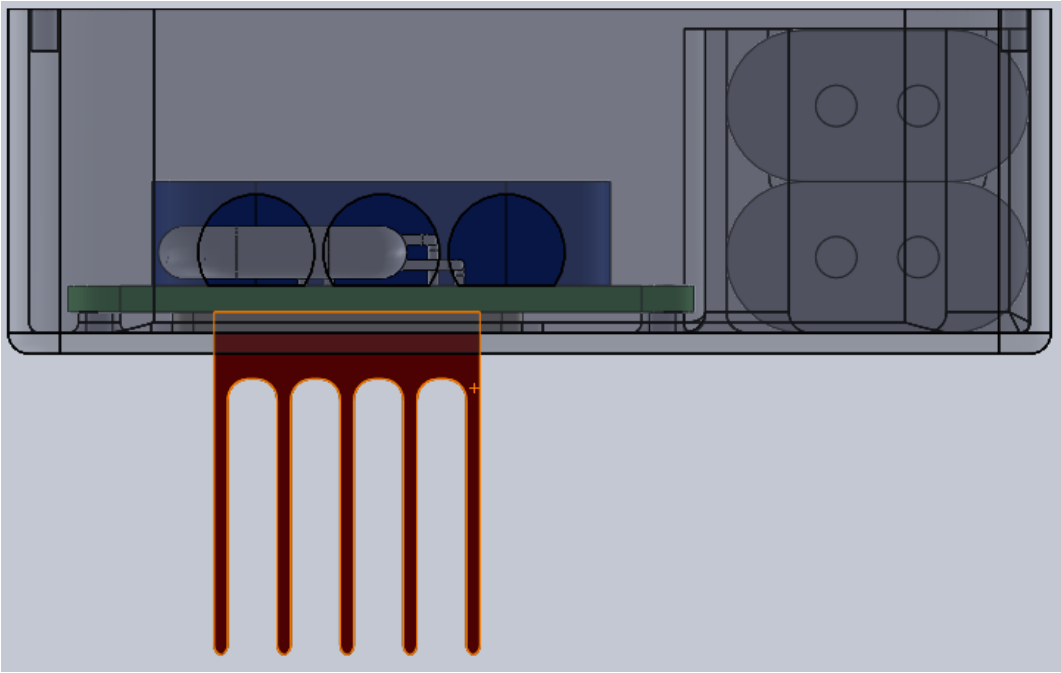


Figure 5.16: Heatsink attached to PCB and protruding through the enclosure

The open source thermal simulation tool *Energy 2D* was used to simulate the benefits of a heatsink. The following image (Figure 5.17) shows a simulation with an ambient temperature of $30°C$ and the phone (the white rectangle) starting at $125°C$ and dissipating into the enclosure (the grey box). In the simulation, the red areas are at around 30-40$°C$ while the white areas are significantly hotter, at around 100-125$°C$. The enclosure has the thermal properties of ABS plastic, while the heatsink has the thermal properties of aluminum. The simulation shows the heatsink dissipating heat outside of the enclosure, reducing the overall heat inside.

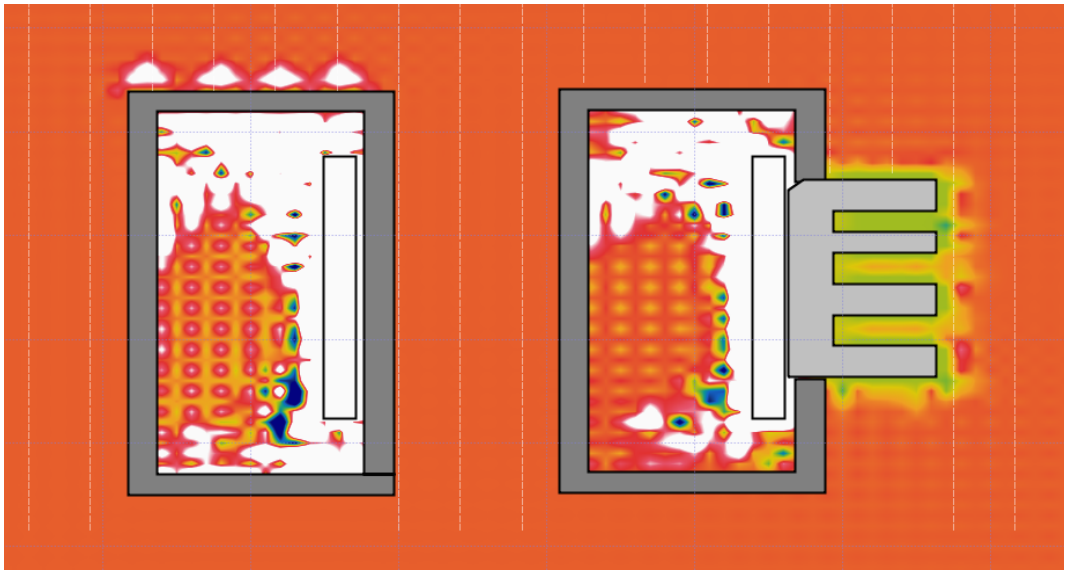


Figure 5.17: An Energy 2D simulation: On the left: Phone in enclosure with no heatsink. On the right: Phone in enclosure and attached to heatsink

5.3 Software Design

The software used in the design will consists of C and C++ code running on the ATMega328P microcontroller, as well as a companion app (RFCx Sentinel) written in Java, running on the Android phone.

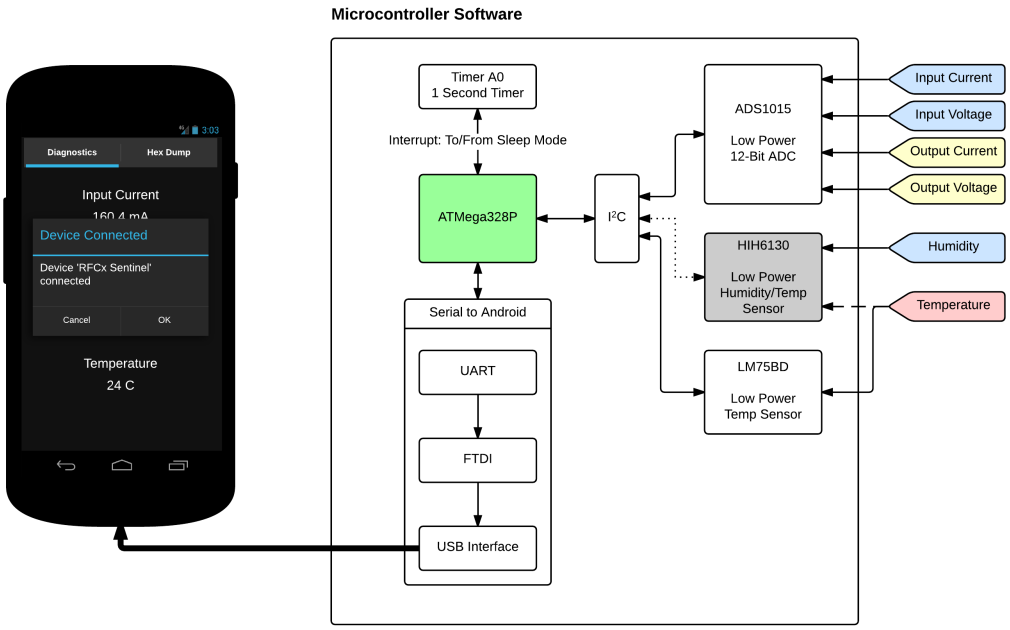


Figure 5.18: Software UML Diagram

The microcontroller software gathers data from a humidity/temperature sensor, and input/output current and voltage. This data is sent via the ATMega328P serial port to the Android phone via the FTDI chip on the PCB and the usb-serial-for-android libraries on the phone. Listing 1 describes the basic functionality of the microcontroller software.

Listing 1: MCU Flow Code Snippet

```
void wakeFromSleep() {
    //Get sensor measurements
    GetDiagnostics(&Diagnostics);
    //Send sensor measurements to phone
    SendDiagnostics(Diagnostics);
    //Go to sleep
    enterSleep();
}
```

5.4 Software Implementation

An ATMega328P microcontroller will be used to take measurements from the sensors. The temperature (and possibly humidity) measurements are sent digitally via I^2C to the microcontroller. On the same I^2C bus is the ADS1015, which has four single-ended analog inputs for reading the input current, input voltage, output current, and output voltage. This data will be sent to the phone using the standard AVR UART serial library.
The data, once sent to the phone, will be monitored by the RFCx Sentinel app. This app is written in Java in Android Studio, and uses the open-source usb-serial-for-android library. The app opens a connection upon discovering a compatible device (FTDI is compatible), and allows the user to view a streamlined interface with relevant data, as well as a tab for the raw hex dump of incoming serial data.

6 Verification

This section outlines the steps that are planned to verify that the hardware and software operate in the intended manner.

6.1 Hardware Verification

To verify the power consumption of the devices in **Requirement 3.1.1.1.**. the voltage applied and current drawn by the both the enhanced device and the current device will be measured over the course of 1 hour during a realistic usage scenario. The average power will be verified to not exceed 90% of the existing device's usage.
To verify **Requirement 3.1.1.5.**. a visual inspection by volunteers will qualify camouflage schemes. A successful inspection will be one in which more than 80% of the volunteers are unable to find the device within a 2 minute inspection window.
To verify **Requirement 3.1.1.6.**. a sample group of people with varying skill levels will be asked to provide feedback the on assembly process.

6.2 Software Verification

To verify **Requirement 3.1.2.1.**., audio will be recorded and compressed using multiple algorithms including the current algorithm. An infographic hosted on RFCx's website, shown in Figure 2 illustrates the flow of data that is collected from the phone and analyzed on the server.
To verify **Requirement 3.2.1.2.**, simple packets will be sent from the microcontroller to the phone. These packets can be verified with debug tools or simple programs on the microcontroller and the phone. A debug serial port may be included as a peripheral to the microcontroller to aid in debugging. Any analog values read by the microcontroller will be verified with shop equipment.

7 Validation

Requirement 3.2.1.2., **Requirement 3.1.1.2.**, and **Requirement 3.1.1.4.** can be validated by inspecting the components on the PCB to ensure all required components are present on the board. Requirements **Requirement 3.1.1.6.** and **Requirement 3.1.1.7.** can be validated by handing the assembly instructions out to a sample group of people with varying skill levels and receiving feedback on the set of instructions from them. People with the ability to read other languages can be used if the documents have been translated from English into other languages.
Requirement 3.2.1.3. can be validated by inspecting the total cost printed on the BOM for the enhanced device.
Requirement 3.2.1.1., **Requirement 3.1.1.1.**, and **Requirement 3.2.1.2.** can be validated by current and voltage measurements taken in the lab using a DMM. The power calculations made based on those measurements can be compared to similar calculations based on measurements taken from the current device. SPICE simulation data and values displayed on the RFCx Sentinel app will be compared to the actual power consumption of the new device.
Requirement 3.2.2.1., **Requirement 3.2.2.2.**, **Requirement 3.2.2.3.**, and **Requirement 3.2.1.2.** can be validated by comparing the values reported in the RFCx Sentinel app to the calculated power consumption of the enhanced device.
Requirement 3.1.1.3. can be validated by changing the scale factor in the microcontroller program, reprogramming the microcontroller from the PCB header and comparing the values displayed on the RFCx Sentinel app to the previous displayed values. The new values displayed on the RFCx Sentinel app should reflect the change to the scale factor in the microcontroller program.
Requirement 3.1.1.3., **Requirement 3.2.2.3.**, and **Requirement 3.1.2.1.** can be validated during a field test. The enhance device can be mounted at eye level in a tree and turned on to allow the device to start sending audio data to the RFCx server. Requirement **Requirement 3.1.1.5.** can be validated by having a volunteer outside of the project search for the device in a wooded area without knowing what it looks like to see if it sticks out. A chainsaw can be turned on 0.25 miles from the device to trigger an alert and validate that the enhanced device can communicate with the RFCx server. The size of the audio recording sent to the RFCx server after employing the audio compression algorithm can be compared to the size of audio data sent to the RFCx server by the current

device to determine the audio compression ratio of this algorithm. Both audio recordings can be played back to determine the level of audio interference created by GSM transmission.

8 Conclusion

This section describes the conclusion of the project including modifications performed on the hardware and software sides. These modifications include two revisions of the PCB, substantial additions to the microcontroller software, and changes to the enclosure, panel frame, and mounting hardware.

8.1 PCB Modifications

8.1.1 1st Revision

Design The first revision of the board was designed for a specific case in mind and was monolithic. A 3D rendering of the board can be seen in figure 8.1

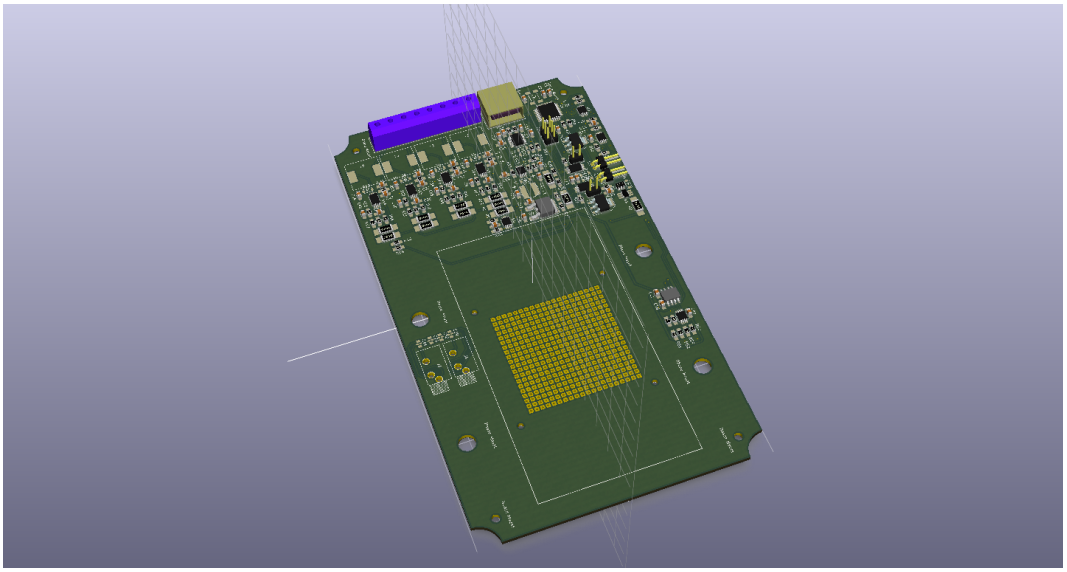


Figure 8.1: First revision of PCB design

Issues Several issues became apparent during assembly and construction of this revision of the design.

- Monolithic design meant that a problem on the board could render the whole board useless
- Position of certain components interfered physically
- An external pullup resistor on the RESET pin of the microcontroller was mistakenly omitted.
- Footprint for the HIH humidity sensor was too big
- Thermocouples were not long enough to reach the batteries
- There was no LED on the ATmega328P to use as a status LED or for debugging
- The sense resistor between the regulators and the power path management was too small
- The ADS1015 required a pullup resistor on the ALERT GPIO pin
- The capacitors on the XTAL were the wrong value, they should be around 6pF
- The output capacitor of the 3.3V regulator was not connected to GND on the schematic

Solutions The monolithic board issue was resolved by breaking the board into two separate boards. One board contained the MPPT circuitry and the other board contained the regulation circuitry for the 3.3V and 5V rails, the microcontroller, environment sensors, and the usb interface. This way, the funtionality of the main board can be retained while the power source can be changed. The two boards were made smaller so that they can fit into a multitude of enclosures and be positioned in different ways. Parts that were mistakenly omitted were included for the design of the second revision, and footprints and other issues were corrected.

8.1.2 2nd Revision

Design The second revision of the board focused on modularity and compactability so that the enclosures that it fit in are not as constrained as the first revision. A 3D rendering of the second revision of the board can be seen in figure 8.2 and 8.3. An LED was also added to the board to aid in rudimentary debugging.

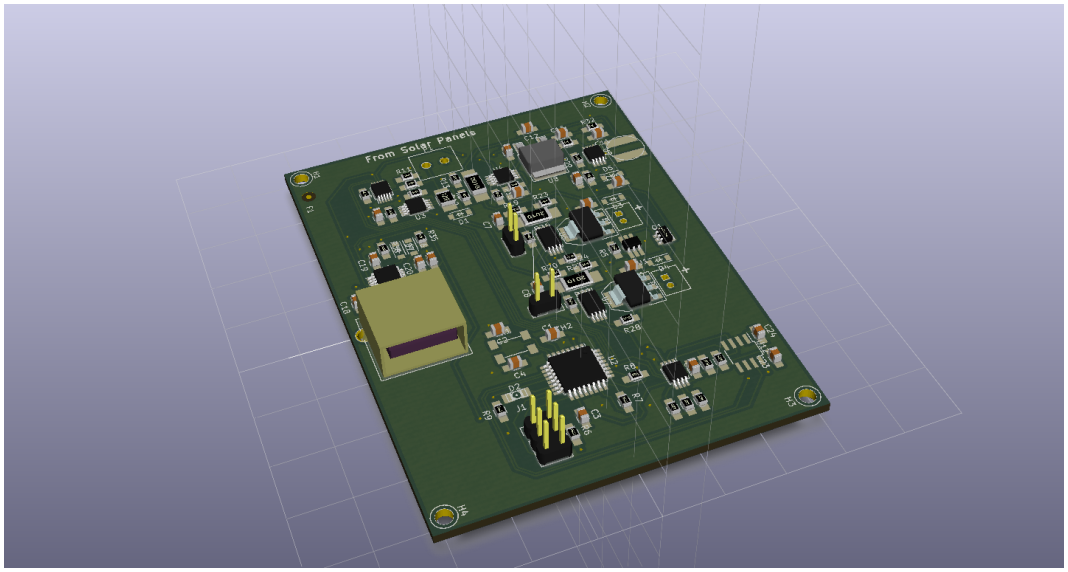


Figure 8.2: Main Board of Revision 2

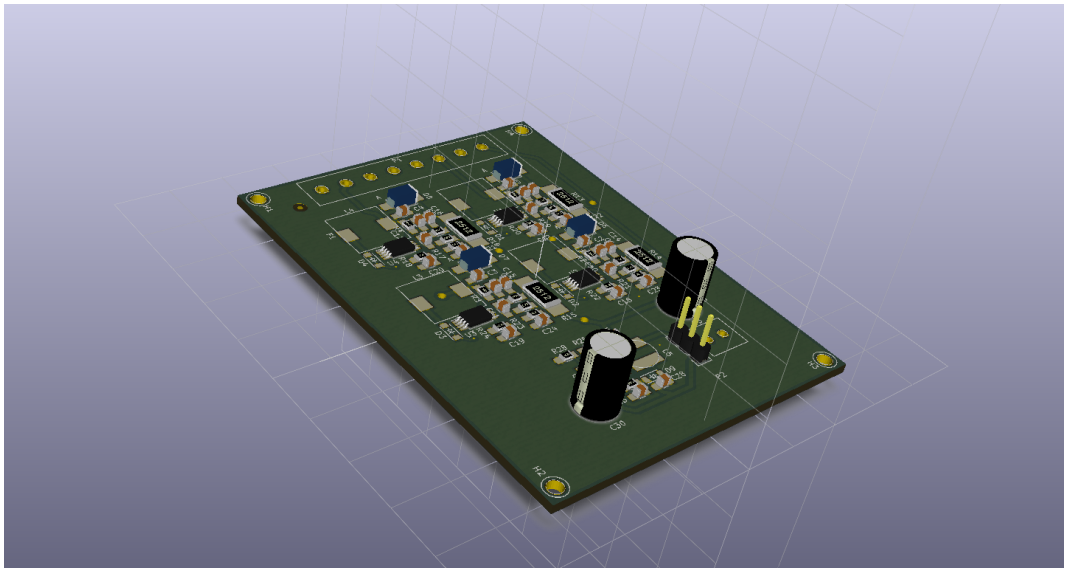


Figure 8.3: MPPT Board of Revision 2

Issues The second revision of the board was not without issues. Certain subsystems were difficult to test on the first revision of the board, leading to untested systems moving through to the second revision of the board. Some of the issues include:

- Swapped D+ and D- signals on the usb Interface
- The FT230X, the USART to USB chip, is not fully supported by Android devices. The FT232R will be used on future boards
- FTDI needs a voltage sense divider
- Footprint on external ADC is wrong, the address pin is unconnected (must be pulled up to VCC)
- The output capacitor of the 5V regulator was rated at only 6.3V, which is not enough to cover voltage spikes
- The GSM audio filtering board needs to be printed
- The X2Y capacitors have a footprint that is very difficult to solder
- The ICSP silkscreen should include a small tab where the connector has a tab so it is clear how to align it
- There are +/- signs missing on the silkscreen for the terminal header connectors
- The battery connector footprints are backwards
- R1 and R2 (the input/output sense resistors) should use the same footprint
- The main board terminal header label "From Solar Panels" should read "From MPPT Board"
- 0Ω resistors should be used instead of wires to pull up/down the address pins on the LM75BD
- Some part labels are misleading or covered up by components

Solutions For a short term fix, the signals that were routed wrong were soldered with wire on the board so that tests could be carried out. For the third revision of the board, these issues will be fixed.

8.1.3 Future 3rd Revision

Design The next revision of the board will have the routing fixes that made it to revision 2. This revision of the board design is meant to be a version that can be field tested. All issues descibed in the section for the second revision will be fixed.

8.2 Software Modifications

The software modifications were almost exclusively done to the microcontroller software. The microcontroller software was taken from a very simple piece of code that generated random values and sent them over USART on the Arduino to a functional piece of microcontroller code running on the actual board that can read sensors and transmit them to the phone.

8.2.1 Microcontroller Software

The microcontroller software underwent substanital changes, and was basically completely rewritten. The code now includes functions to read the temperature from the LM75BD, read the temperature and humidity from the HIH6130, read the input and output voltages and currents from the ADS1015, and read the battery charging status from the BQ2057. The microcontroller code can be found in section F.1.

8.3 Enclosure Modifications

The enclosure and mounting hardware were also modified. A new enclosure was chosen to be a smaller fit and also to include a weather-sealed rubber gasket on the lid. The new enclosure is shown in C under C.1. The new assemblies are shown in the same appendix under C.2 and C.3.

8.4 Results of Verification and Validation

The results of the verification and validation signoff are as below. Following each list is a description of the items in that section, where applicable.

8.4.1 Fully Completed

Item 8.4.1.1. The board has a microcontroller and a USB connection to the phone for power and data

Item 8.4.1.2. The phone can be charged from the board via the USB connector

Item 8.4.1.3. The new device does not cost more than $250

Item 8.4.1.4. The input voltage to the main board is capable of charging the batteries

Item 8.4.1.5. The Diode-ORing circuit is capable of switching between the input voltage and the batteries without an interruption in power

Item 8.4.1.1. and Item 8.4.1.2. The board does include a USB connection that supports both power and data. A phone was charged using the USB connector on the board.

Item 8.4.1.3. The new device costs $131.48, as shown in the Bill of Materials in B. The Main Board, including all components and the enclosure is $93.87, and the MPPT Board with all components is $37.61.

Item 8.4.1.4. The input voltage is capable of charging the batteries using the BQ2057 battery charge controllers. The batteries were fully charged in approximately 11 hours. The current and voltage was monitored during the charging process and it was plotted below in Figure 8.4.

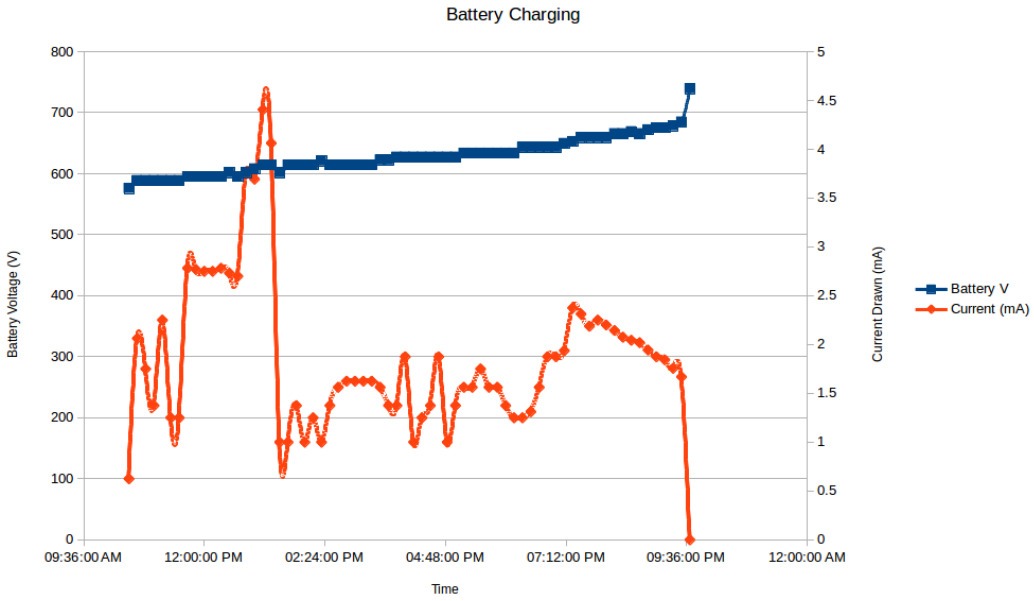


Figure 8.4: Battery Charging Curve

Although the BQ2057 *does* seem to work, an alternative IC, the MCP73871 shown here https://www.adafruit.com/datasheets/MC could be another option. This IC is used on the current revision of the device, and has been tested in the field. The MCP73871 was not chosen for the new design because the package is not solderable by hand. Since the design was completed, RFCx has shown interest in having the boards manufactured using automated pick-and-place machines, so the MCP73871 could be used again.

Item 8.4.1.5. The diode-ORing circuit was able to properly switch between the two voltage sources with no interruption. The switching can be seen below, where channel 1 (yellow) is the gate of the switching transistor for the Diode-ORing chip and channel 2 (blue) is the output voltage. It can be seen that when the input voltage exceeded the battery voltage, the output voltage followed the input. The DMM shows the battery voltage.

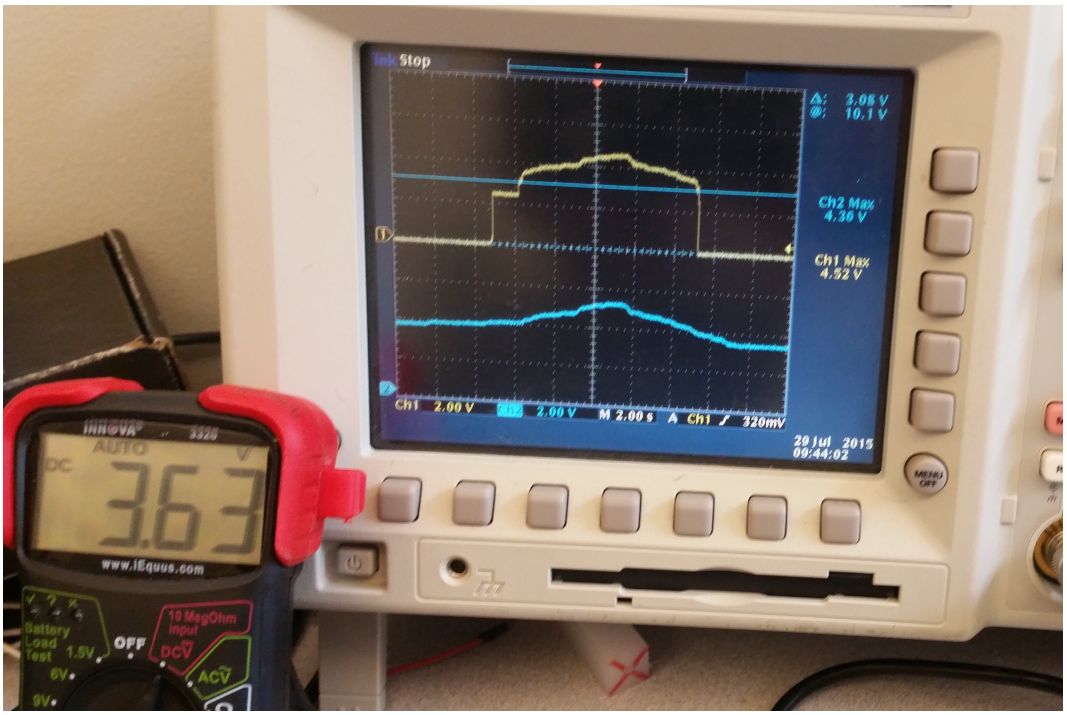


Figure 8.5: Screenshot of Diode-ORing Switching

8.4.2 Partially Completed

Item 8.4.2.1. The microcontroller is capable of sending the correct data to an Android phone

Item 8.4.2.2. The power usage of the new design is more efficient than the existing design

Item 8.4.2.1. The microcontroller is fully capable of communication with either a phone or a computer, but the chosen USART to USB chip, the FT230X FTDI chip, is not compatible with Android phones. The FT230X can communicate with a computer with no issues whatsoever, but to communicate with a phone the FT232R, another chip from FTDI must be used. The FT232R was tested on a breakout board with the microcontroller and it was able to communicate with the phone. The reason this is listed as "partial" is because the FT232R chip is currently not the chip on the board. It will be the chip for the third revision.

Item 8.4.2.2. Due to the incompleted **Item 8.4.3.1.**, the power efficiency could not be measured. However, the efficiency of the regulators on the board are close to 90%. The efficiency of hte 5V and 3.3V LDO are very high, but the system efficiency cannot be measured because the efficiency of the MPPT stage is completely unknown.

8.4.3 Not Completed

Item 8.4.3.1. The MPPT board is capable of providing a 4.6V output from the solar panels

Item 8.4.3.1. The MPPT board does **not** provide the correct output. The problem is that the MPPT chip that was selected, the SPV1040, is not the correct chip for this purpose. The SPV1040 is not a real boost regulator chip, and it is far too touchy in terms of component selection. The screenshot below shows an individual MPPT

functioning, but this functionality is brief. The screenshot shows the switching of the input inductor as the light profile changes.

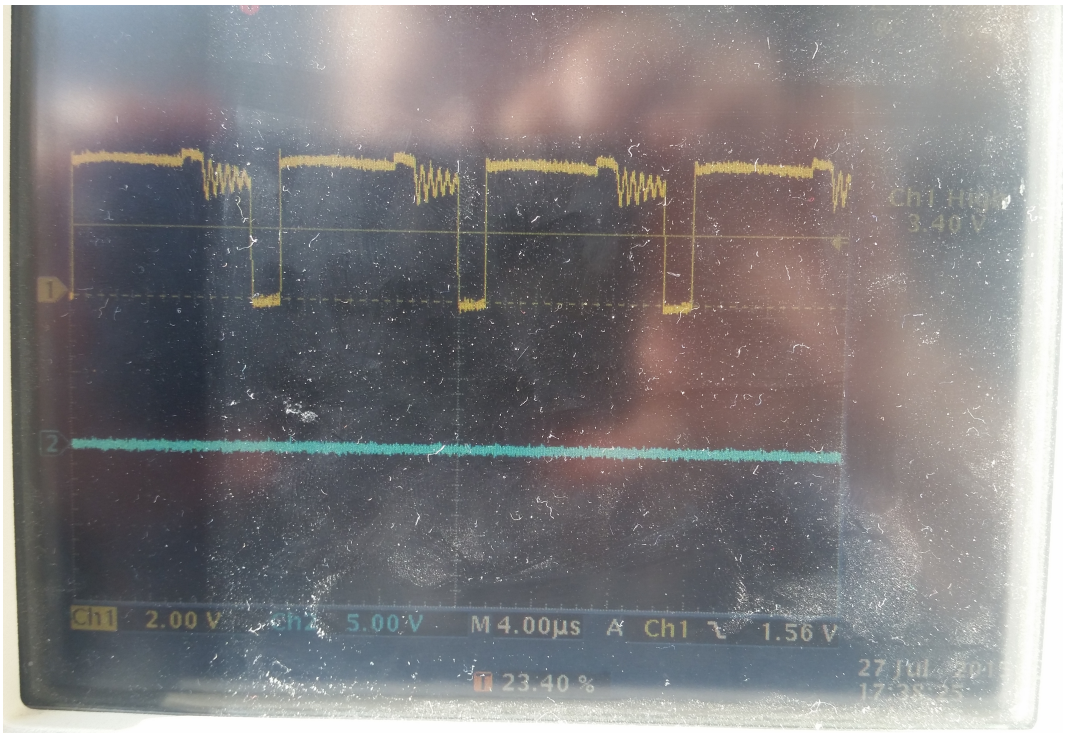


Figure 8.6: MPPT Functionality Screenshot

Although the MPPT circuit is sometimes functional, the device requires a rock-solid MPPT chip which will work every time. To that end, the following MPPT chips may present a better alternative to the SPV1040:

1. MPT612: http://www.mouser.com/pdfdocs/Solar-NXP-MPT612.pdf, which is a programmable MPPT with flexible input and output voltages and minimal components required
2. SPV1020: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00275733.pdf, which is a similar chip to the SPV1040 but is more advanced in that it allows for distributed MPPT amongst panels, although it does require more external components
3. A microcontroller-based design, as described in http://ww1.microchip.com/downloads/en/AppNotes/00001521A.pdf, which is far more flexible and reliable but requires software and some more components.

9 Project Budget and Schedule

9.1 Budget

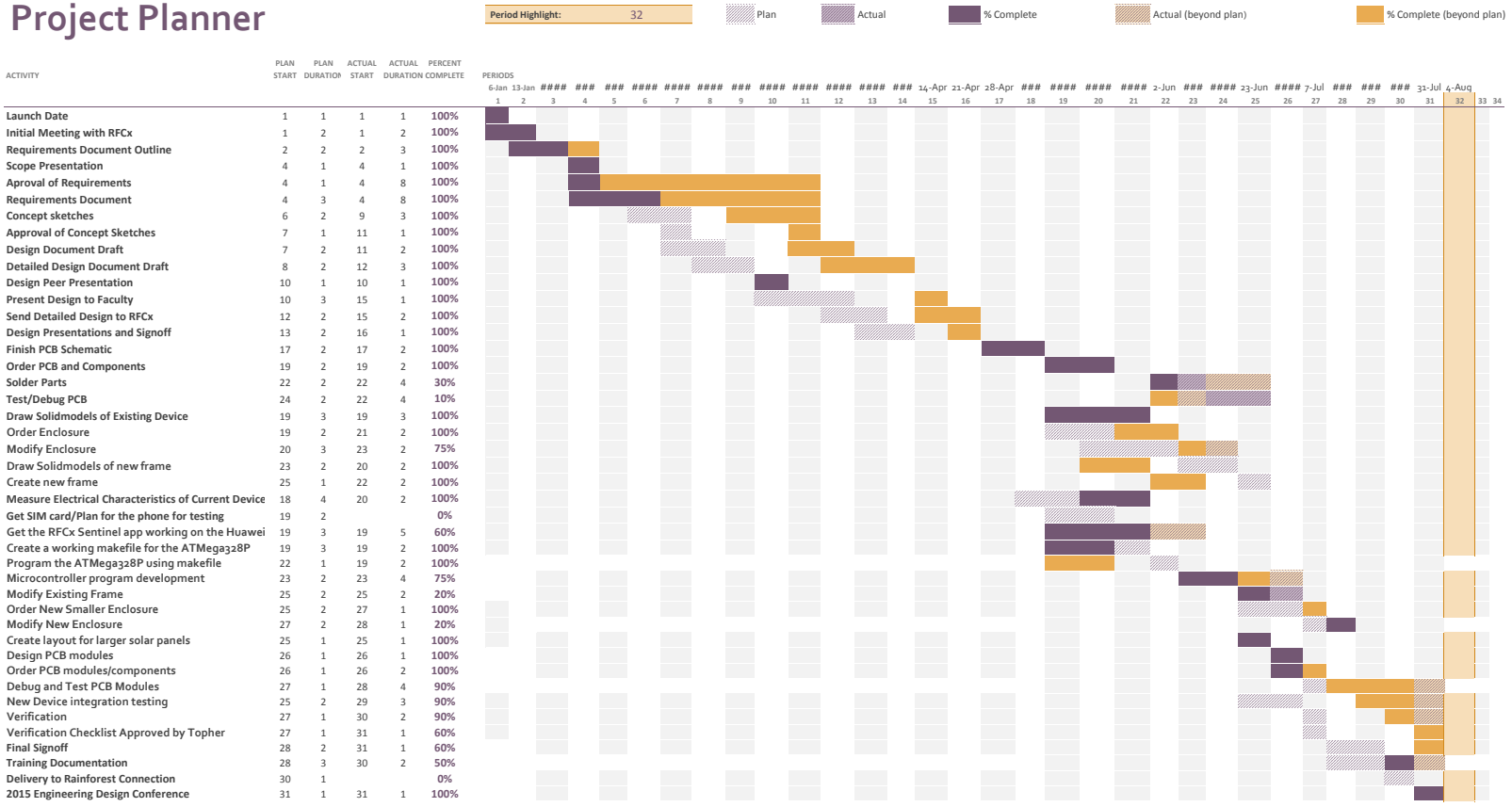
This project will require minor funding for manufacturing and populating the PCB, and the materials to construct a new enclosure, as well as general testing and component costs. As RFCx is a non-profit organization working solely on grant money, this budget is not expected to be covered by them.

- $400 = 4 x $100 for PCB manufacturing
- $320 = 4 x $80 for PCB components
- $200 = 1 x $200 for general needs
- Estimated Total: $920

RFCx

9.2 Schedule

The schedule in Figure 9.1 is the proposed schedule for all phases of the project. This will be used as a general timeline for guidance throughout the project.



ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Launch Date	1	1	1	1	100%
Initial Meeting with RFCx	1	2	1	2	100%
Requirements Document Outline	2	2	2	3	100%
Scope Presentation	4	1	4	1	100%
Aproval of Requirements	4	1	4	8	100%
Requirements Document	4	3	4	8	100%
Concept sketches	6	2	9	3	100%
Approval of Concept Sketches	7	1	11	1	100%
Design Document Draft	7	2	11	2	100%
Detailed Design Document Draft	8	2	12	3	100%
Design Peer Presentation	10	1	10	1	100%
Present Design to Faculty	10	3	15	1	100%
Send Detailed Design to RFCx	12	2	15	2	100%
Design Presentations and Signoff	13	2	16	1	100%
Finish PCB Schematic	17	2	17	2	100%
Order PCB and Components	19	2	19	2	100%
Solder Parts	22	2	22	4	30%
Test/Debug PCB	24	2	22	4	10%
Draw Solidmodels of Existing Device	19	3	19	3	100%
Order Enclosure	19	2	21	2	100%
Modify Enclosure	20	3	23	2	75%
Draw Solidmodels of new frame	23	2	20	2	100%
Create new frame	25	1	22	2	100%
Measure Electrical Characteristics of Current Device	18	4	20	2	100%
Get SIM card/Plan for the phone for testing	19	2			0%
Get the RFCx Sentinel app working on the Huawei	19	3	19	5	60%
Create a working makefile for the ATMega328P	19	3	19	2	100%
Program the ATMega328P using makefile	22	1	19	2	100%
Microcontroller program development	23	2	23	4	75%
Modify Existing Frame	25	2	25	2	20%
Order New Smaller Enclosure	25	2	27	1	100%
Modify New Enclosure	27	2	28	1	20%
Create layout for larger solar panels	25	1	25	1	100%
Design PCB modules	26	1	26	1	100%
Order PCB modules/components	26	1	26	2	100%
Debug and Test PCB Modules	27	1	28	4	90%
New Device integration testing	25	2	29	3	90%
Verification	27	1	30	2	90%
Verification Checklist Approved by Topher	27	1	31	1	60%
Final Signoff	28	2	31	1	60%
Training Documentation	28	3	30	2	50%
Delivery to Rainforest Connection	30	1			0%
2015 Engineering Design Conference	31	1	31	1	100%

Figure 9.1: Gantt Chart for Schedule

10 Sponsor Sign-off

On behalf of Rainforest Connection(RFCx), the undersigned approves the functional requirements and design contained in this document that specify the project deliverables and functionality.

Topher White

_____

Signature

_____

Date

Dave Grenell

_____

Signature

_____

Date

Appendices

Appendix A Electrical Schematics and Simulations

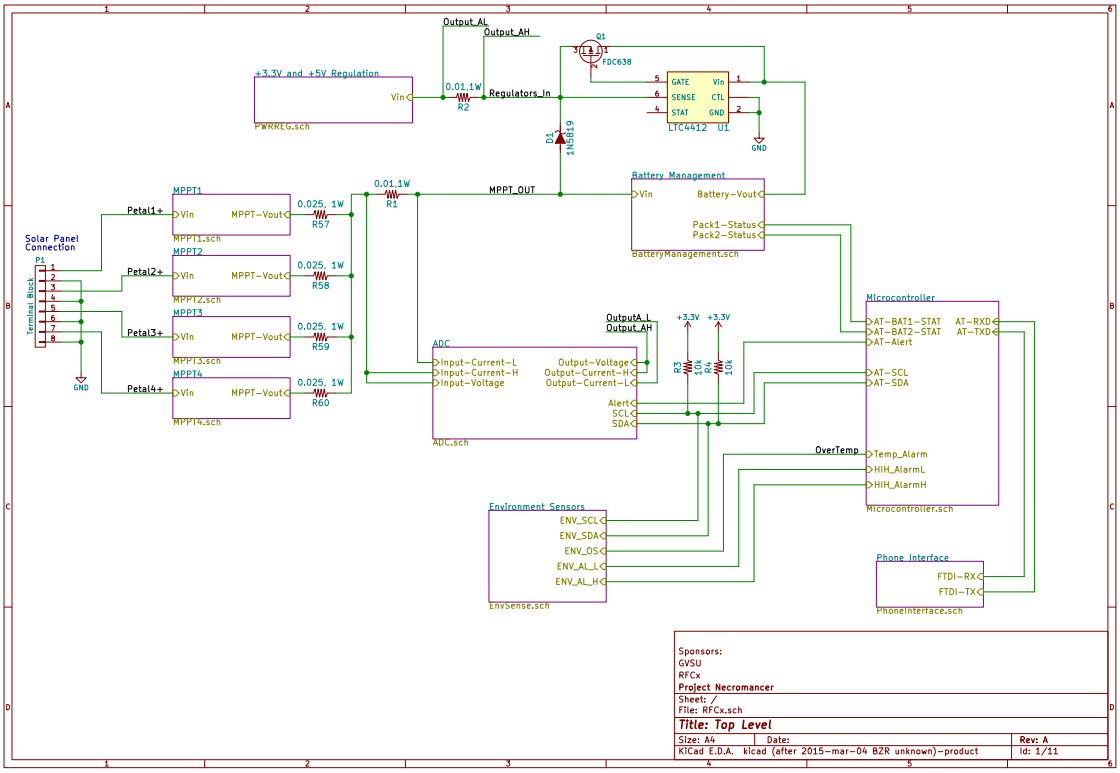


Figure A.1: Top Level Schematic

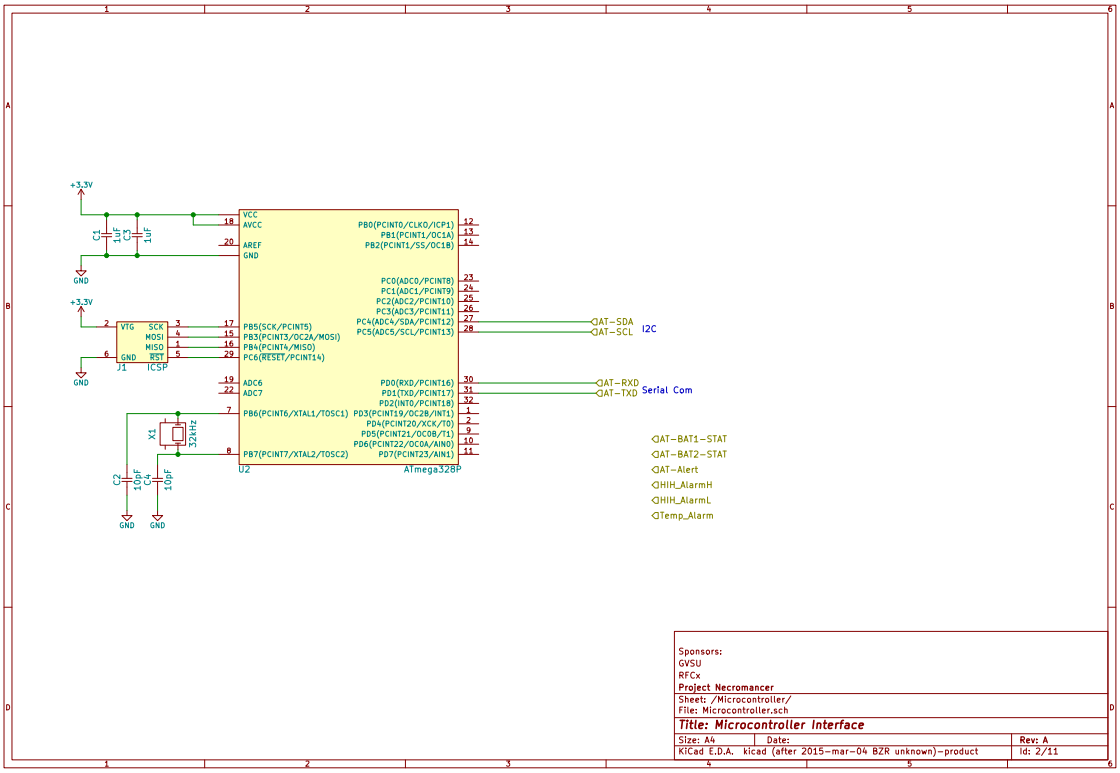


Figure A.2: ATmega328P Microcontroller

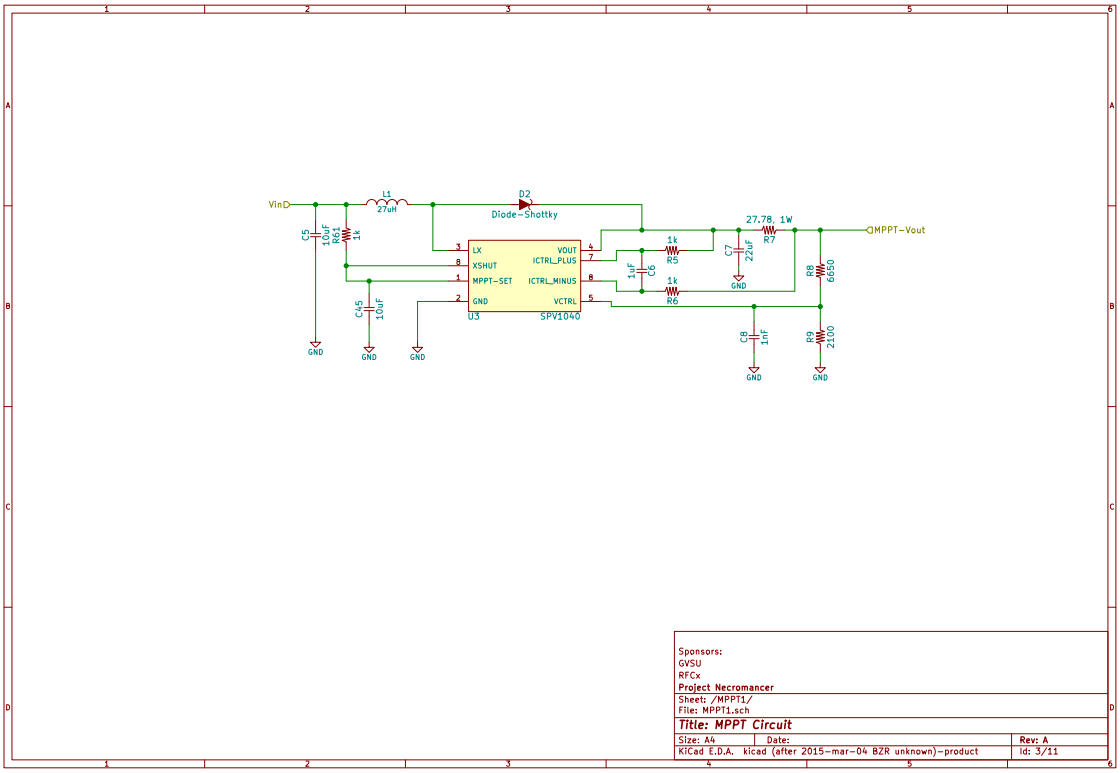


Figure A.3: SPV1040 Max Point Power Tracker (MPPT)

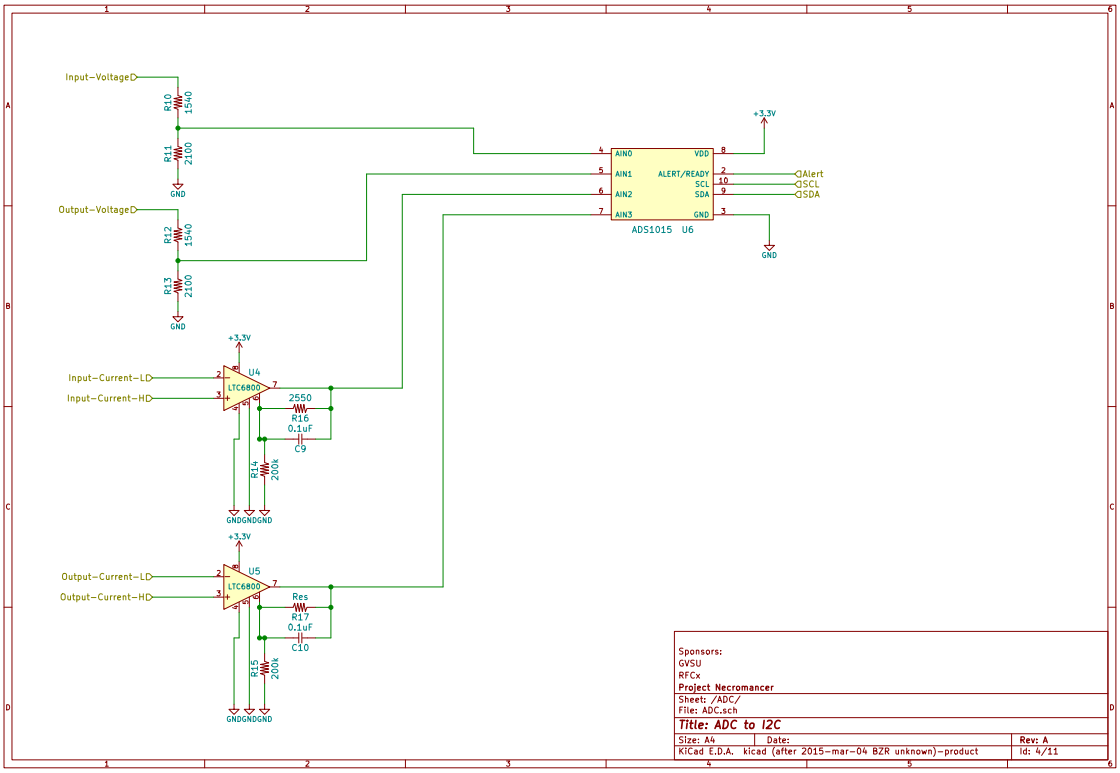


Figure A.4: ADS1015 External Analog-to-Digital Converter and Instrumentation Amplifiers

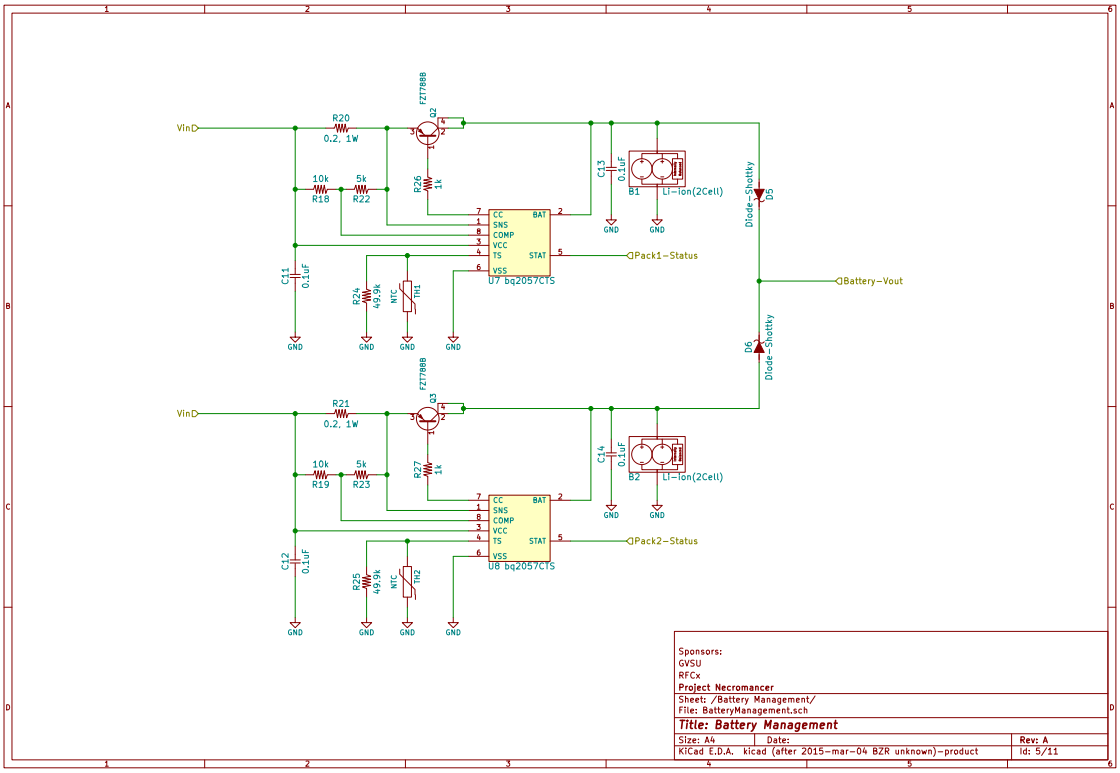


Figure A.5: BQ2057CTS Battery Management Controllers

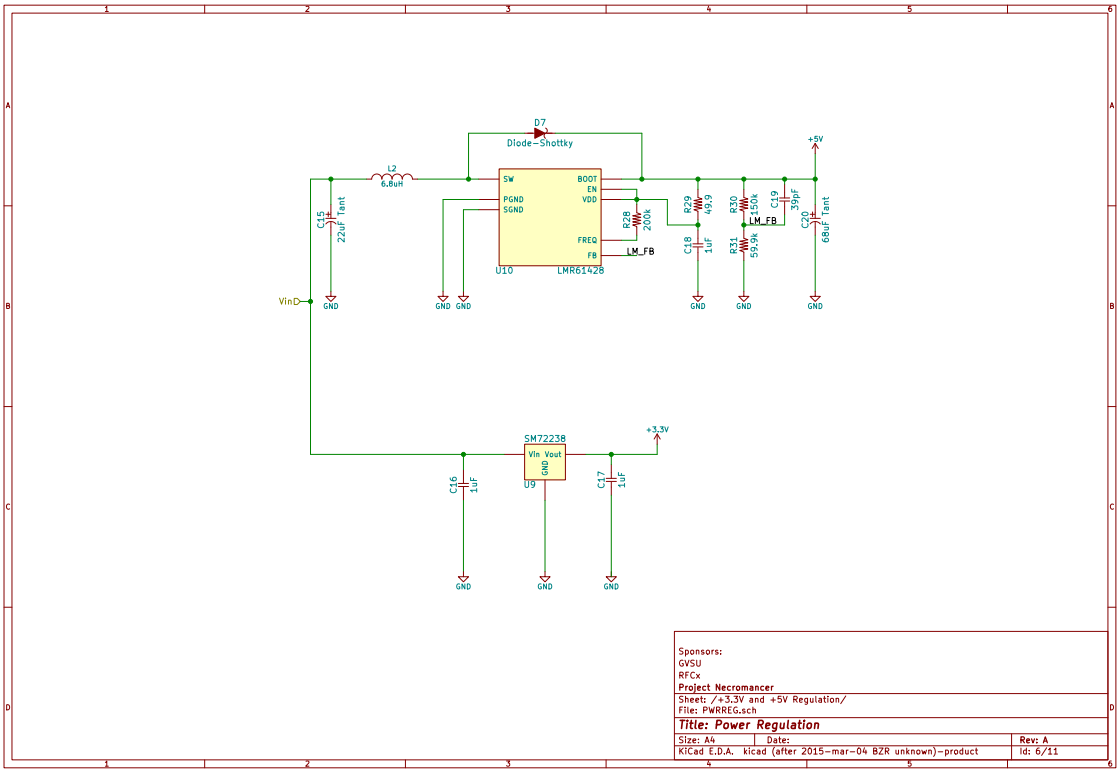


Figure A.6: LMR61428 5V Regulator and SM72442 3.3V Regulator

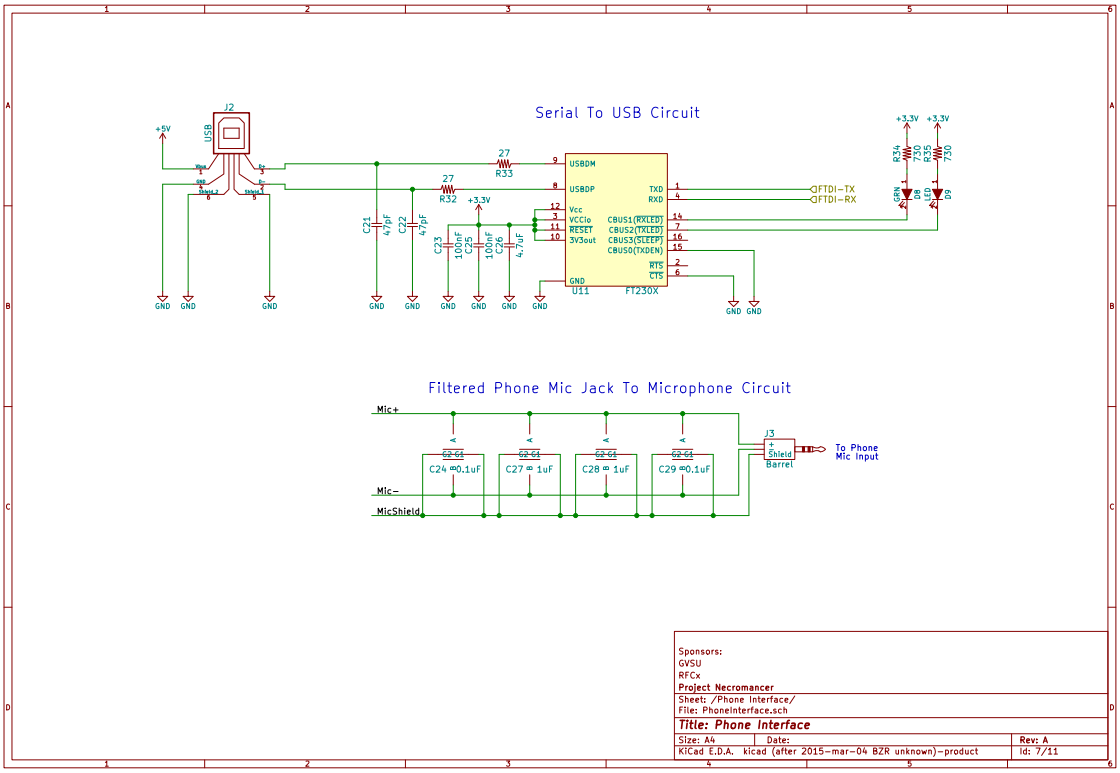


Figure A.7: Android Phone Interface: FT230X FTDI Serial to USB and Filtered Microphone Pass-through

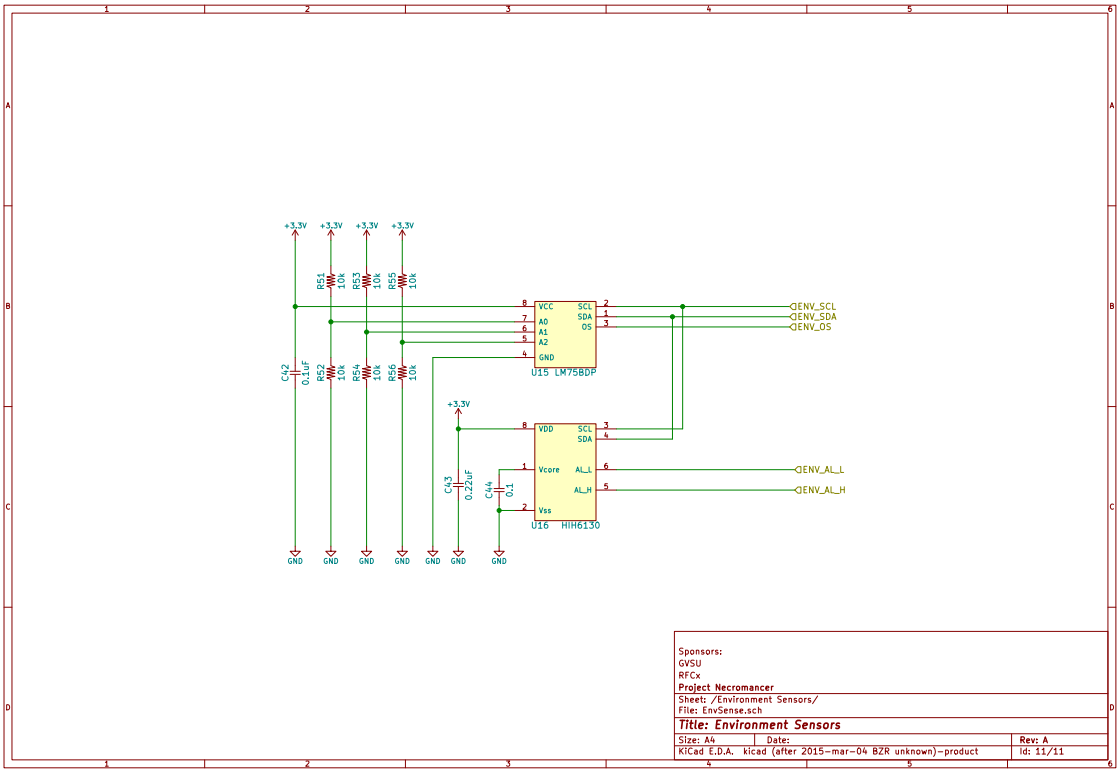


Figure A.8: LM75BD Temperature Sensor and HIH6130 Temperature and Humidity Sensor

Power Path Simulations

The two different power path management circuits were simulated with LTSpice. The two different circuits can be seen below in Figure A.9. On the left is the FET and diode topology. On the right is the chosen Diode-ORing topology. The same diode and transistor were used in each simulation. A load of 550mA was used because this is more than the calculated worst case current draw from the circuit.

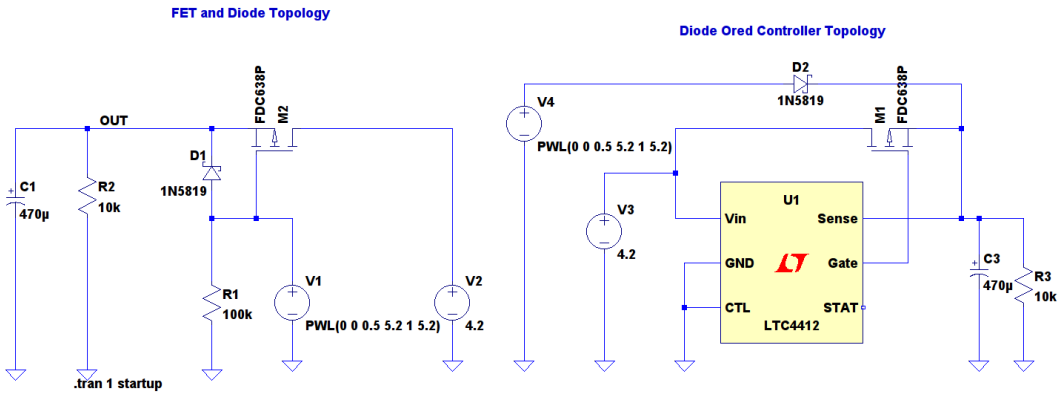


Figure A.9: Two Power Path Management Circuits. Left: FET amd diode topology. Right: Diode-ORed topology

In the simulation for the FET and diode topology, in Figure A.10, it can be seen that as the panel voltage is rising, there is a considerable 500mV dip from 3.5V to around 3.0V. If this occured, the 3.3V linear drop out regulator would not be able to regulate properly and the 3.3V line would be deactivated.

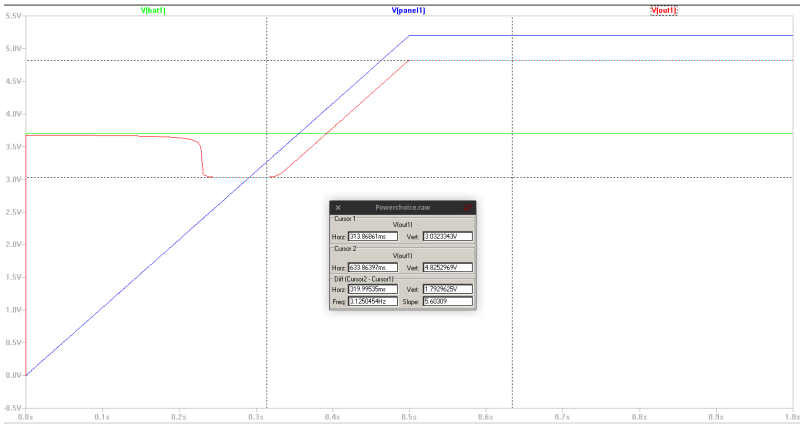


Figure A.10: Simulation Result of FET and Diode Topology

Conversely, in Figure A.11 the Diode-ORed topology simulation shows an extremely small drop in voltage when powered from only the batteries and a reasonable drop when powered from the solar panel. This is the desired behavior.

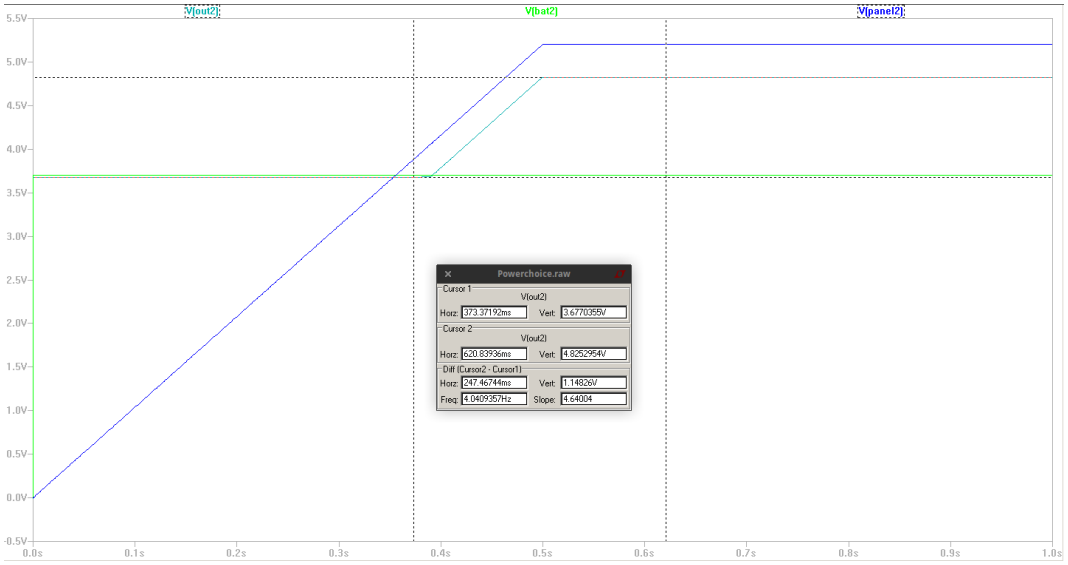


Figure A.11: Simulation Result of Current Diode-ORed Topology

Appendix B Bill Of Materials

Mainboard_BOM

Project: Mainboard
Tool: kicad
Number of Parts 97

Reference	Quantity	Value	Distributor	Distributor #	Price	Extended Price
Q1	1	FDC638	DigiKey	FDC638PCT-ND	$0.55	$0.55
Q2,Q3	2	FZT788B	DigiKey	FZT788BCT-ND	$1.01	$2.02
F1	1	Fiducial	DigiKey	NA		$0.00
C2,C4	2	6pF	DigiKey	311-1095-1-ND	$0.18	$0.36
C15	1	39pF	DigiKey	1276-1769-1-ND	$0.10	$0.10
C18,C17	2	47pF	DigiKey	1276-1832-1-ND	$0.10	$0.20
C5,C6,C7,C9,C8,C10,C19,C20,C22	9	0.1uF	DigiKey	399-1167-1-ND	$0.11	$0.99
C23	1	0.22uF	DigiKey	587-1287-1-ND	$0.12	$0.12
C3,C1,C14,C13,C12	5	1uF	DigiKey	1276-1275-1-ND	$0.10	$0.50
C21	1	4.7uF	DigiKey	1276-1065-1-ND	$0.21	$0.21
C11	1	22uF Tant	DigiKey	511-1506-1-ND	$1.03	$1.03
C16	1	68uF Tant	DigiKey	478-8414-1-ND	$1.08	$1.08
C24	1	0.1	DigiKey	399-1167-1-ND	$0.11	$0.11
R21,R22	2	0.2, 1W	DigiKey	989-1049-1-ND	$0.61	$1.22
R2	1	0.5,1W	DigiKey	RCWE.51FCT-ND	$0.79	$0.79
R1	1	0.51, 1W	DigiKey	RCWE.51FCT-ND	$0.79	$0.79
R34,R33	2	27	DigiKey	P27ACT-ND	$0.10	$0.20
R30	1	49.9	DigiKey	P49.9KCCT-ND	$0.10	$0.10
R35,R36	2	56	DigiKey	311-56JRCT-ND	$0.10	$0.20
R9	1	730	DigiKey	P732CCT-ND	$0.10	$0.10
R7,R8,R27,R28	4	1k	DigiKey	P1.0KACT-ND	$0.10	$0.40
R10,R12	2	1540	DigiKey	P1.54KCCT-ND	$0.10	$0.20
R11,R13	2	2100	DigiKey	P2.10KCCT-ND	$0.10	$0.20
R16,R17	2	2550	DigiKey	311-2.55KCRCT-ND	$0.10	$0.20
R3	1	4.7k	DigiKey	P10KACT-ND	$0.10	$0.10
R23,R24	2	5k	DigiKey	P4.99KCCT-ND	$0.10	$0.20
R4,R6,R18,R19,R20,R38,R40,R42	8	10k	DigiKey	P10KACT-ND	$0.10	$0.80
R25,R26,R32	3	49.9k	DigiKey	P49.9KCCT-ND	$0.10	$0.30
R31	1	150k	DigiKey	P150KCCT-ND	$0.10	$0.10
R14,R15,R29	3	200k	DigiKey	P200KACT-ND	$0.10	$0.30
R5	1	470k	DigiKey	311-470KCRCT-ND	$0.10	$0.10
H1,H2,H3,H4	4	Mounting_Hole	DigiKey	NA		$0.00

Page 1

Figure B.1

Mainboard_BOM

L1	1	6.8uH	DigiKey	SRN6045-6R8YCT-ND	$0.43	$0.43
D2	1	LED	DigiKey	160-1404-1-ND	$0.39	$0.39
D5	1	Diode-Shottky	DigiKey	1N5819HW-FDICT-ND	$0.52	$0.52
D6	1	GRN	DigiKey	350-2885-ND	$0.47	$0.47
D7	1	YLW	DigiKey	511-1275-1-ND	$0.62	$0.62
D1,D3,D4	3	1N5819	DigiKey	1N5819HW-FDICT-ND	$0.52	$1.56
U1	1	LTC4412	DigiKey	LTC4412ES6#TRPBFCT-ND	$3.07	$3.07
P1	1	CONN_01X02	DigiKey	ED2740-ND	$0.38	$0.38
U2	1	ATmega328P	DigiKey	ATMEGA328P-AU-ND	$3.58	$3.58
J1	1	ICSP	DigiKey	S2011EC-03-ND	$0.36	$0.36
U3,U4	2	LTC6800	DigiKey	LTC6800HMS8#PBF-ND	$3.12	$6.24
U5	1	ADS1015	DigiKey	296-25227-1-ND	$3.32	$3.32
U6,U7	2	bq2057CTS	DigiKey	296-25916-1-ND	$1.85	$3.70
TH1	1	NTC	DigiKey	BC2384-ND,455-1657-ND	$1.70	$1.70
TH2	1	NTC	DigiKey	BC2384-ND, 455-1657-ND	$1.70	$1.70
U9	1	LMR61428	DigiKey	LMR61428XMM/NOPBCT-N	$1.87	$1.87
U8	1	SM72238	DigiKey	296-39811-5-ND	$1.67	$1.67
U10	1	FT230X	DigiKey	768-1135-1-ND	$2.38	$2.38
J2	1	USB	DigiKey	UE27AC54100-ND	$0.40	$0.40
U11	1	LM75BDP	DigiKey	568-4768-1-ND	$0.74	$0.74
U12	1	HIH6130	DigiKey	480-3651-1-ND	$13.75	$13.75
X1	1	32kHz	DigiKey	300-8744-1-ND	$1.18	$1.18
	1		OSH PARK		$13.78	$13.78
	1		Polycase	WP-33*15	$16.49	$16.49
					TOTAL:	**$93.87**

Page 2

Figure B.2

MPPT_BOM

Project:	MPPT
Tool:	kicad
Number of Parts	85

Reference	Quantity	Value	Distributor	Distributor #	Price	Extended Price
F1	1	Fiducial	DigiKey	NA		$0.00
C29,C30	2	DNP	DigiKey	NA		$0.00
C27	1	39pF	DigiKey	1276-1769-1-ND	$0.10	$0.10
C17,C18,C19,C20	4	1nF	DigiKey	399-1147-1-ND	$0.10	$0.40
C5,C6,C7,C8	4	100nF	DigiKey	399-1167-1-ND	$0.11	$0.44
C9,C10,C11,C12,C26	5	1uF	DigiKey	1276-1275-1-ND	$0.10	$0.50
C13,C14,C15,C16	4	4.7uF	DigiKey	490-3338-1-ND	$0.17	$0.68
C1,C2,C3,C4	4	10uF	DigiKey	587-1312-1-ND	$0.17	$0.68
C22,C23,C21,C24	4	10uF	DigiKey	490-3886-1-ND	$0.20	$0.80
C25	1	22uF	DigiKey	511-1506-1-ND	$1.03	$1.03
C28	1	68uF Tant	DigiKey	478-8414-1-ND	$1.08	$1.08
R13,R14,R15,R16	4	0.5, 2W	DigiKey	CRM2512-FX-R510ELFCT-ND	$0.53	$2.12
R26	1	49.9	DigiKey	P49.9CCT-ND	$0.10	$0.10
R5,R6,R1,R7,R8,R2,R9,R10,R3,R11,R12,R4	12	1k	DigiKey	P1.0KACT-ND	$0.10	$1.20
R28	1	49.9K	DigiKey	P49.9KCCT-ND	$0.10	$0.10
R20,R22,R24,R18	4	121K	DigiKey	311-121KCRCT-ND	$0.10	$0.40
R27	1	150K	DigiKey	P150KCCT-ND	$0.10	$0.10
R25	1	200K	DigiKey	P200KACT-ND	$0.10	$0.10
R19,R21,R23,R17	4	330K	DigiKey	311-330KCRCT-ND	$0.10	$0.40
H1,H2,H3,H4	4	Mounting_Hole	DigiKey	NA		$0.00
L5	1	6.8uF	DigiKey	SRN6045-6R8YCT-ND	$0.43	$0.43
L1,L2,L3,L4	4	10uH	DigiKey	308-1666-1-ND	$1.35	$5.40
D5,D6,D8,D7	4	Diode-TVS	DigiKey	SMAJ5.0ALFCT-ND	$0.43	$1.72
D1,D2,D3,D4	4	1n5819	DigiKey	1N5819HW-FDICT-ND	$0.52	$2.08
D9	1	1N5819	DigiKey	1N5819HW-FDICT-ND	$0.52	$0.52
U1,U2,U3,U4	4	SPV1040	DigiKey	NA		$0.00
U5	1	LMR61428	DigiKey	LMR61428XMM/NOPBCT-ND	$1.87	$1.87
P1	1	CONN_01X08	DigiKey	ED2615-ND	$1.31	$1.31
P3	1	CONN_01X02	DigiKey	NA		$0.00
P2	1	CONN_01X03	DigiKey	952-2264-ND, S9001-ND	$0.27	$0.27
	1		OSH Park		$13.78	$13.78
					TOTAL:	**$37.61**

Page 1

Figure B.3

Appendix C Mechanical Drawings

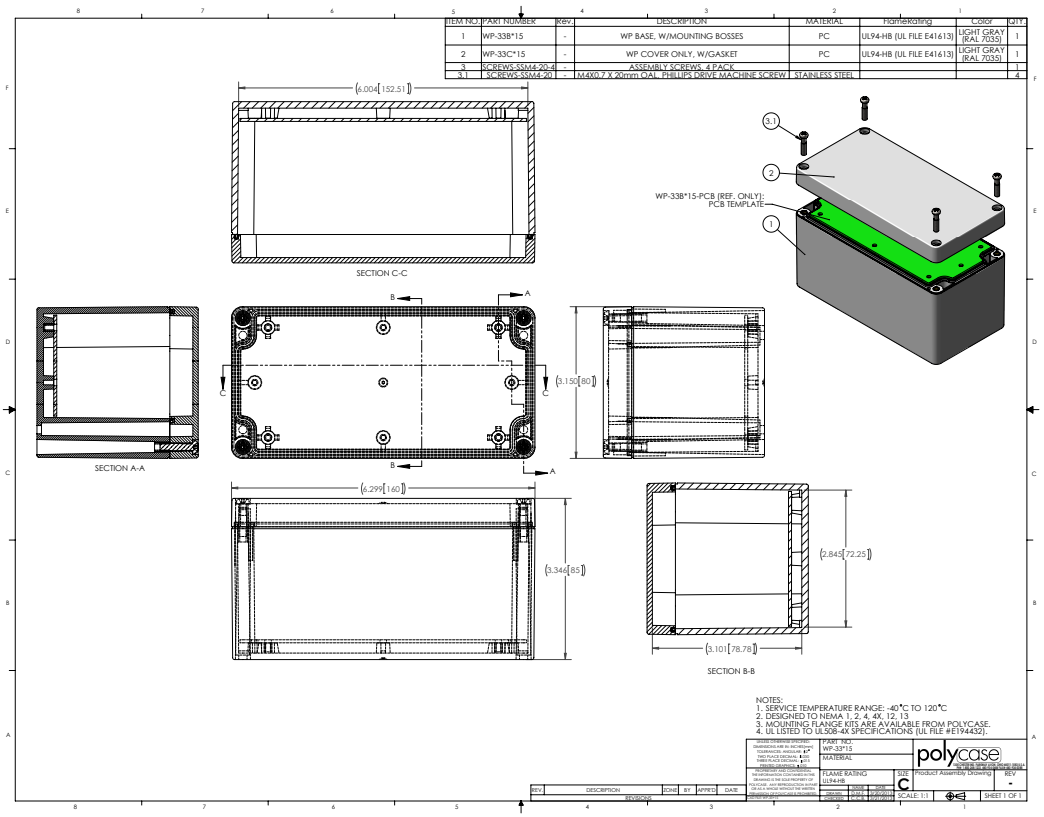


Figure C.1: New Case

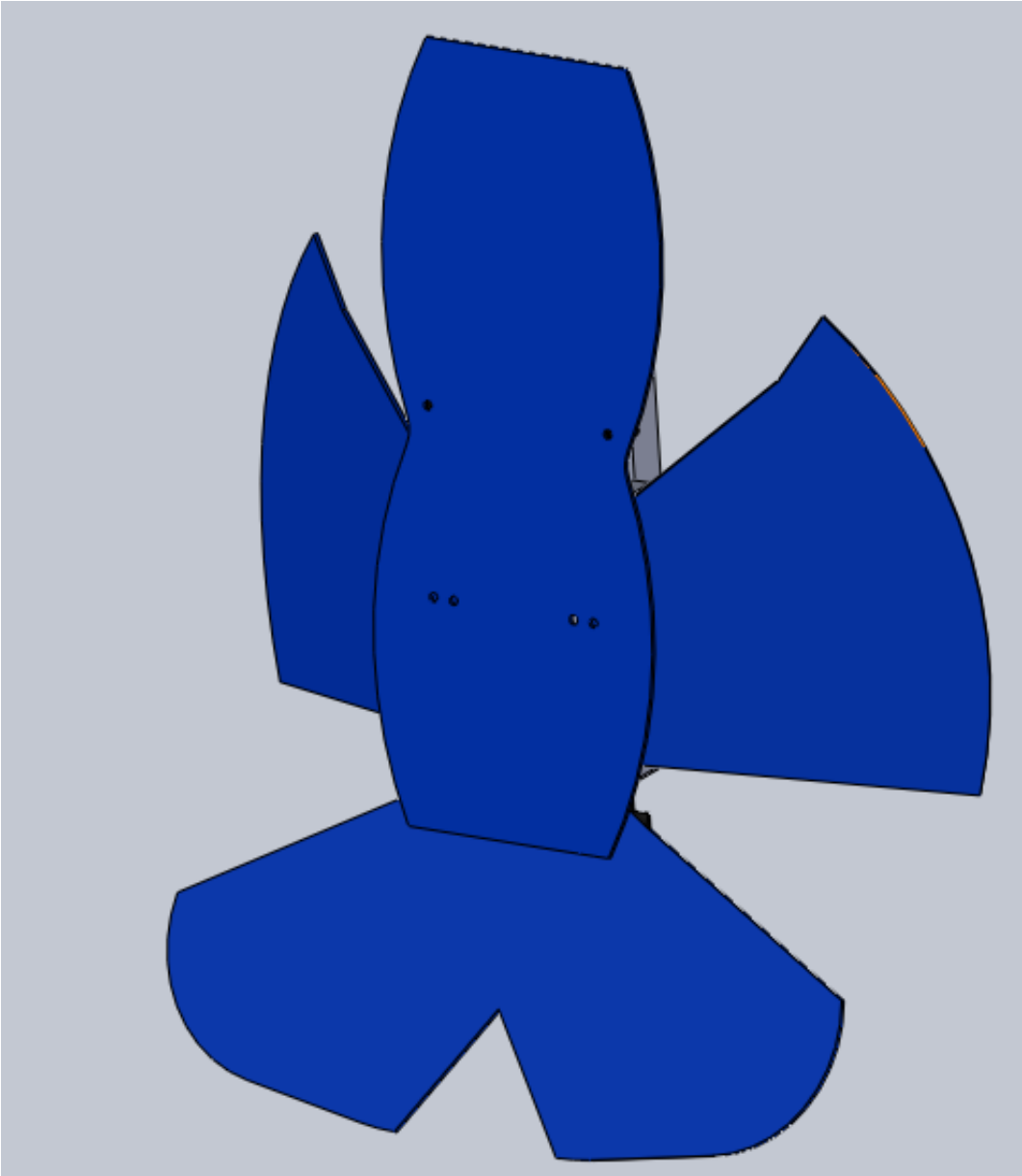


Figure C.2: Panel Assembly

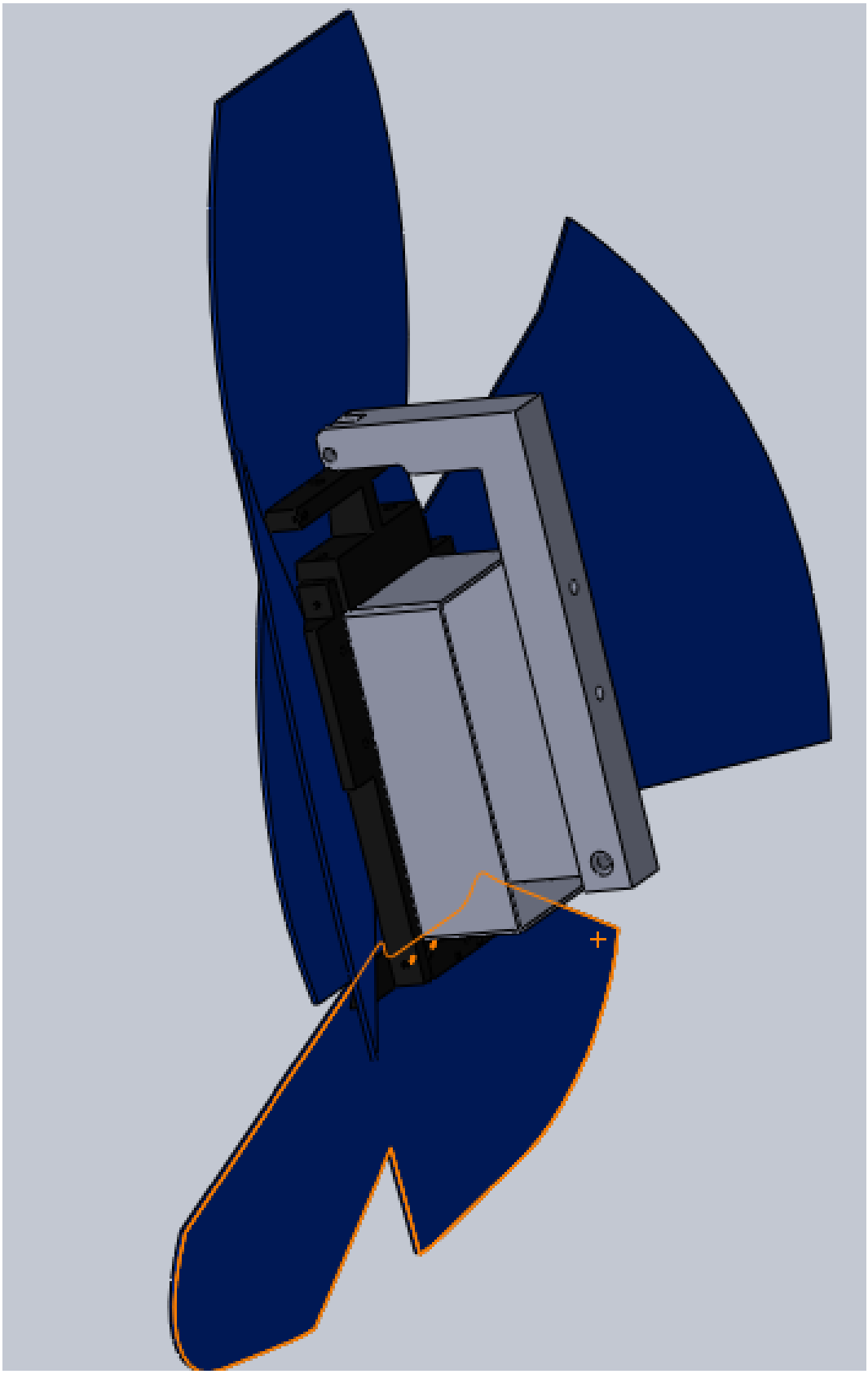


Figure C.3: Panel and Case Assembly

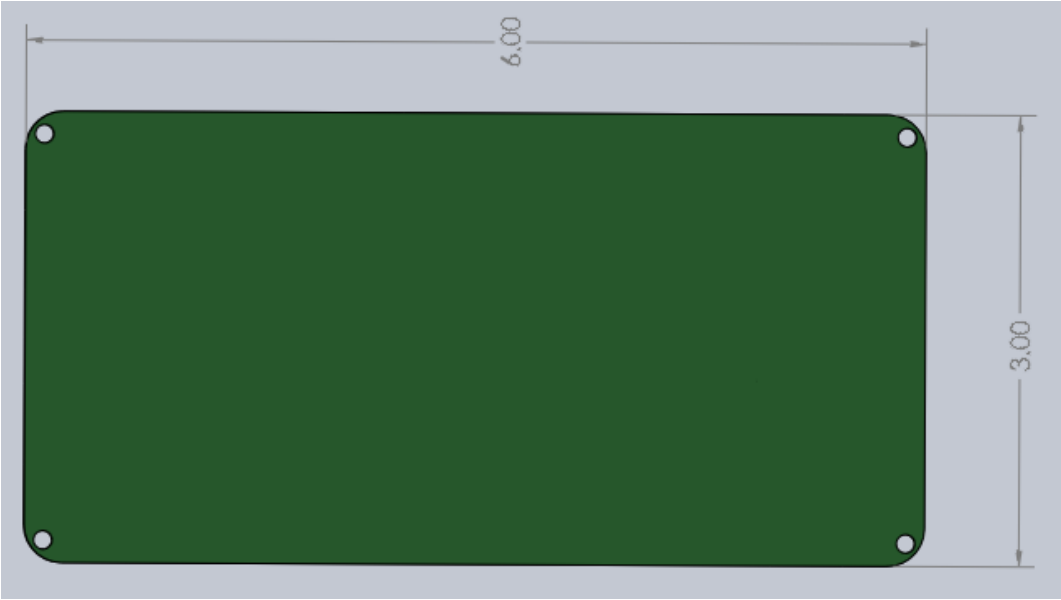


Figure C.4: Rev 1: PCB Dimensions

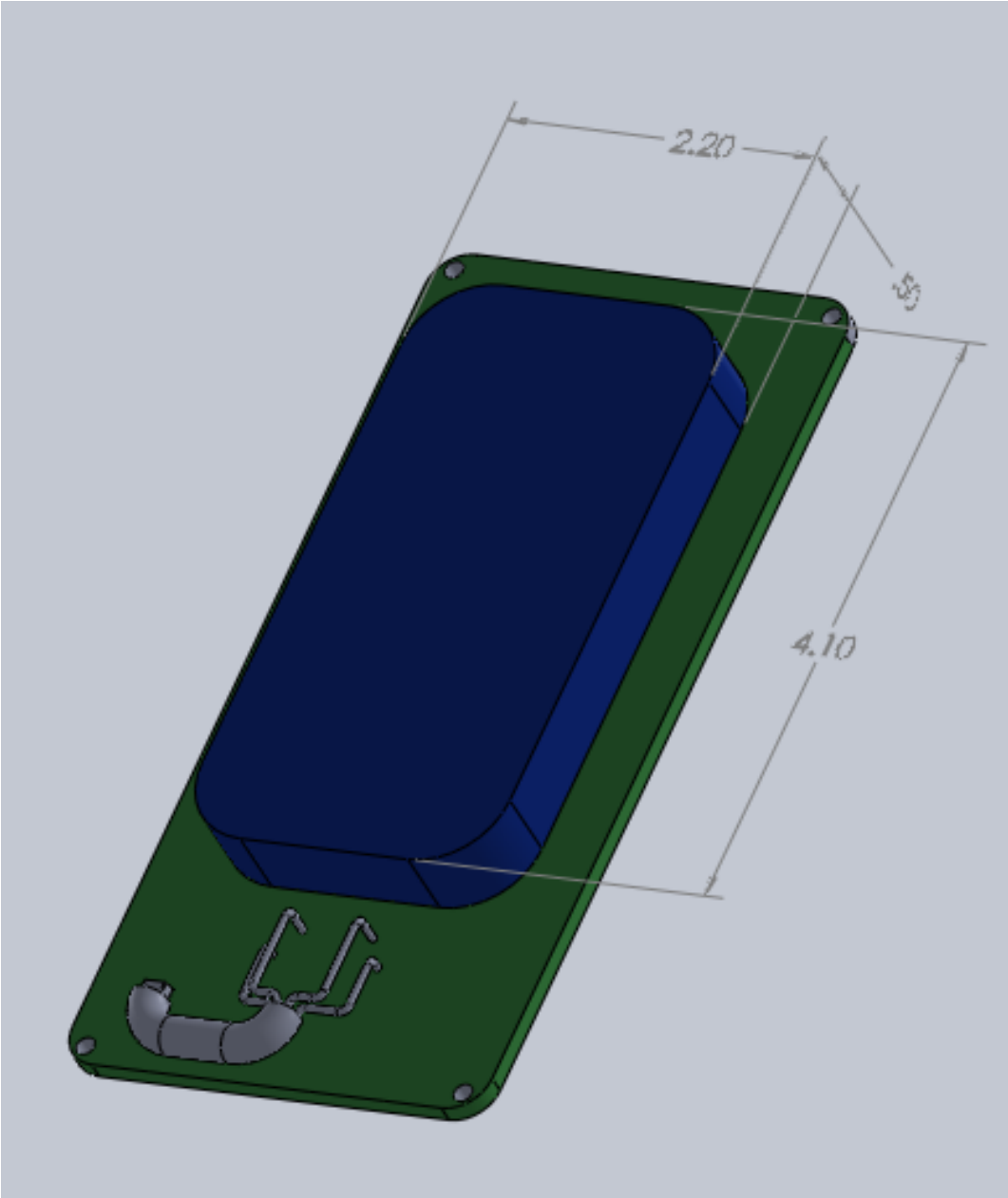


Figure C.5: Rev 1: PCB Dimensions

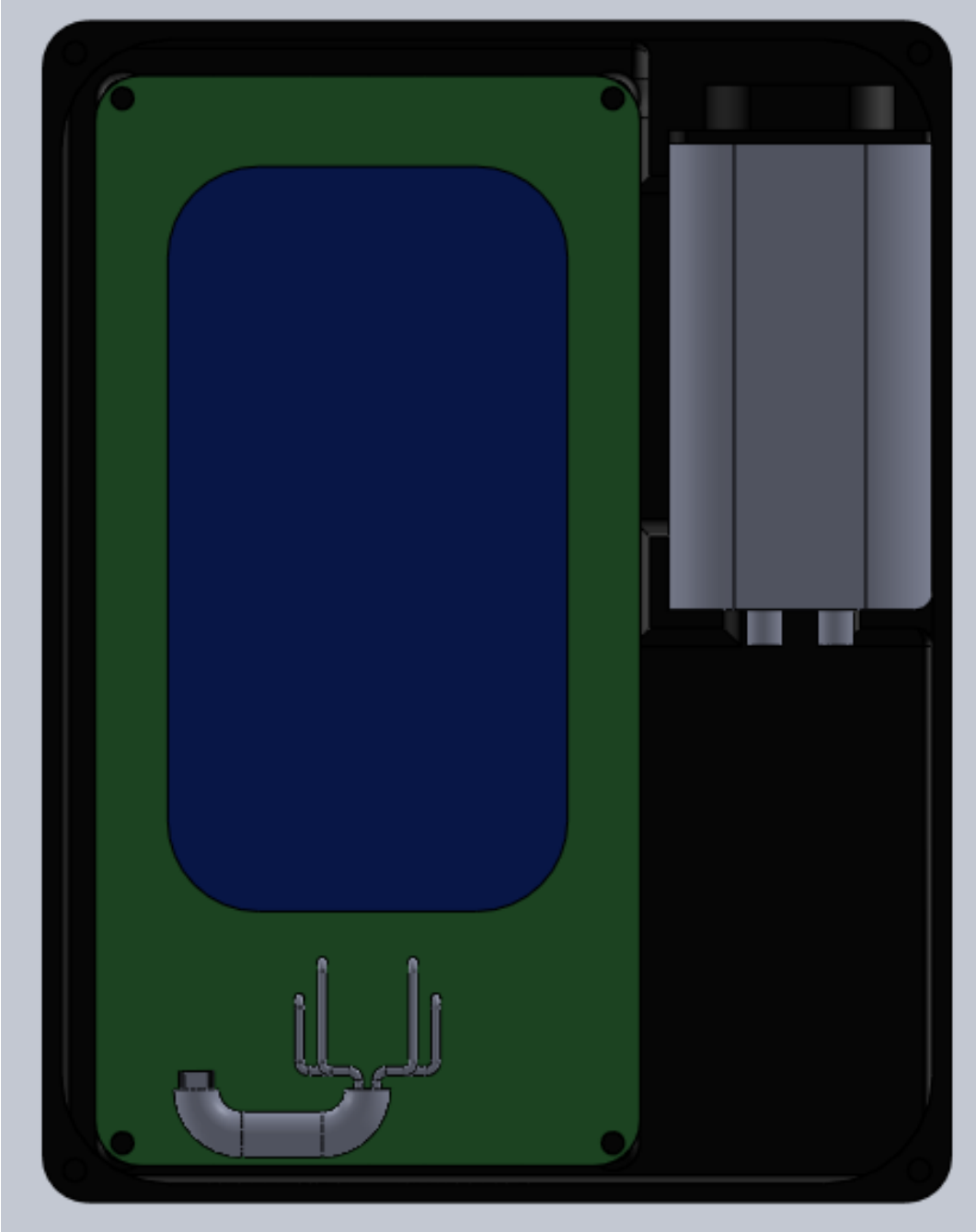


Figure C.6: Rev 1: USB Micro A Connector attached to the PCB with strain relief: A small flexible cable

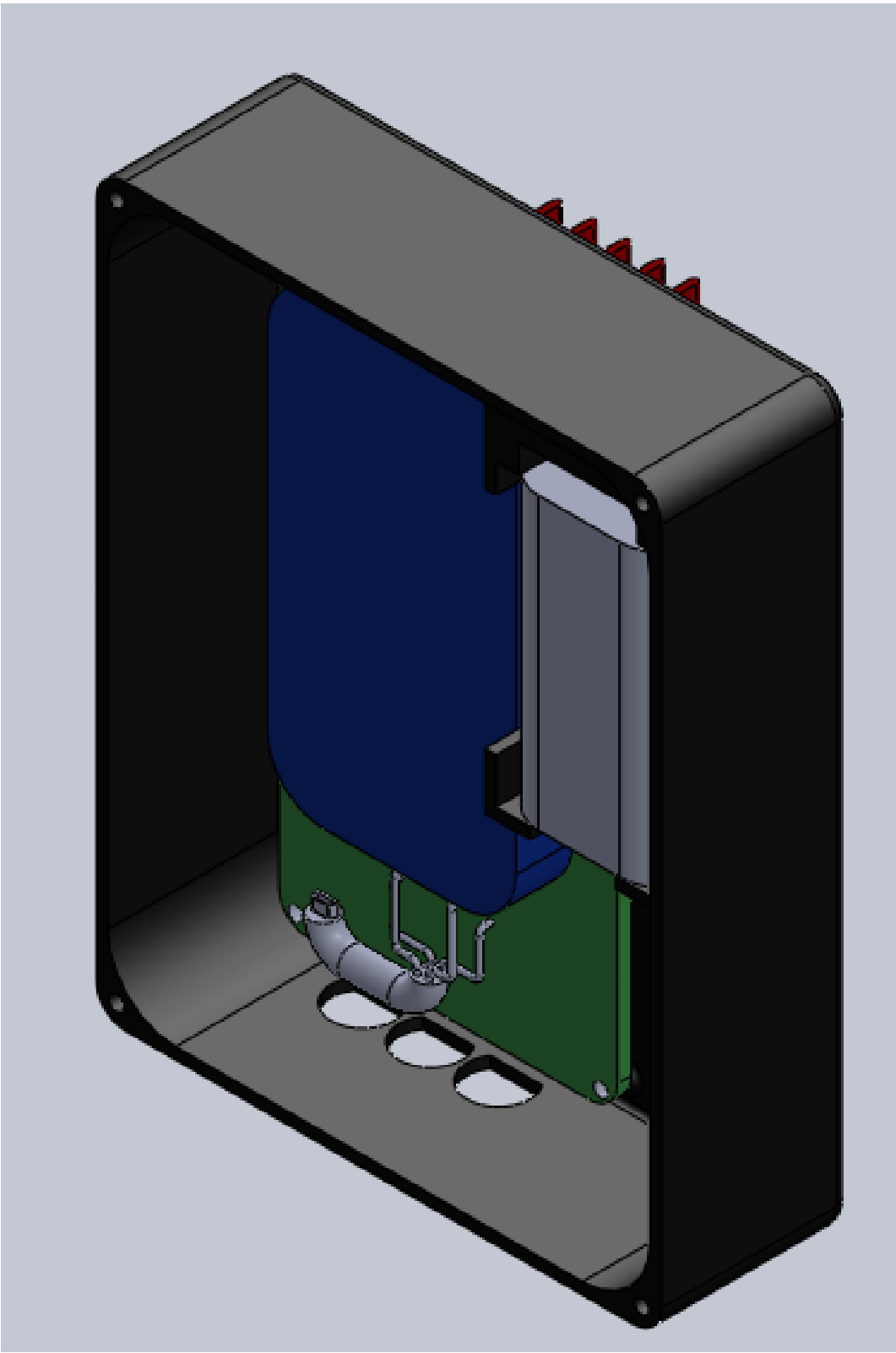


Figure C.7: Rev 1: Front View of the Enclosure

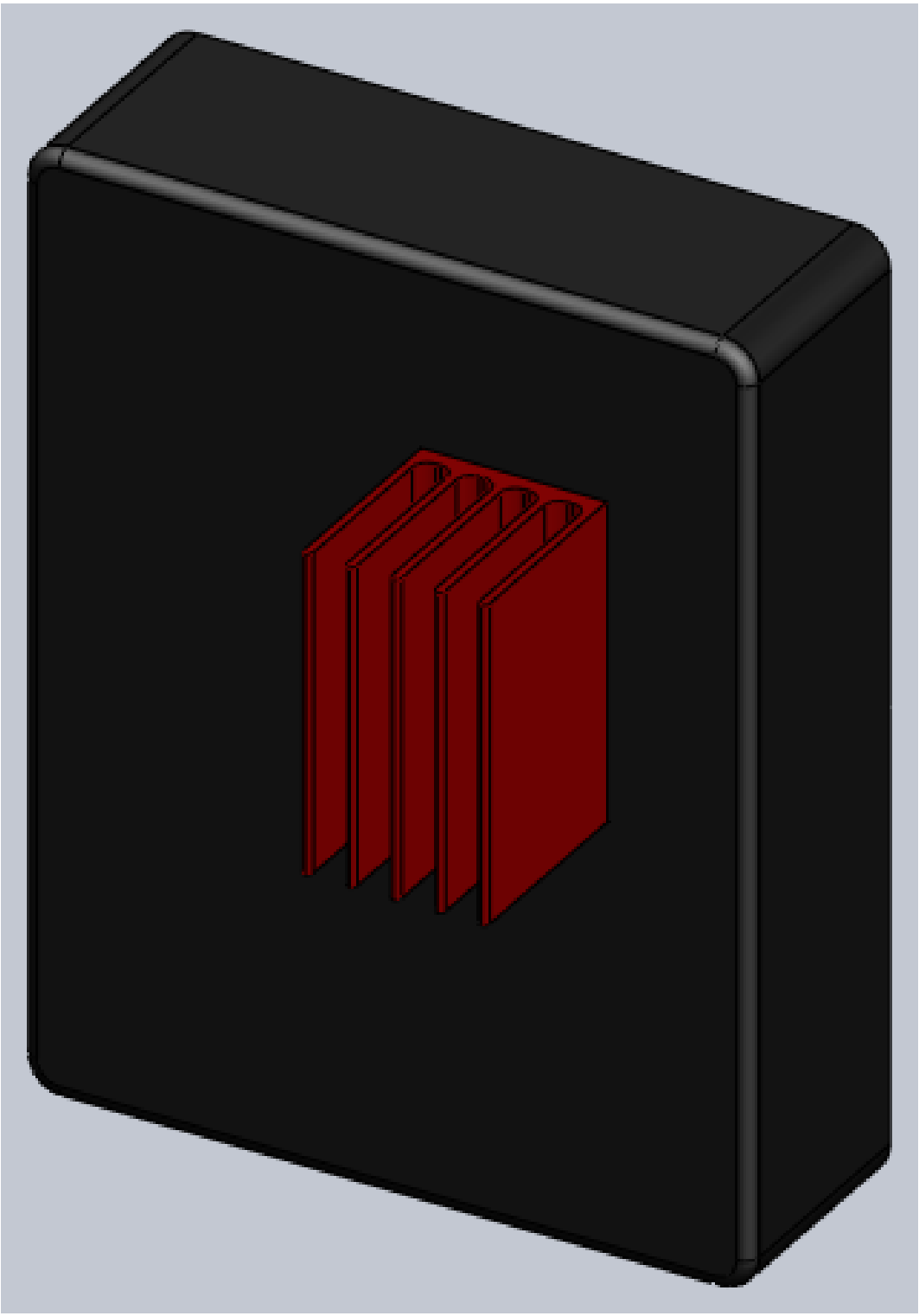


Figure C.8: Rev 1: Back View of the Enclosure

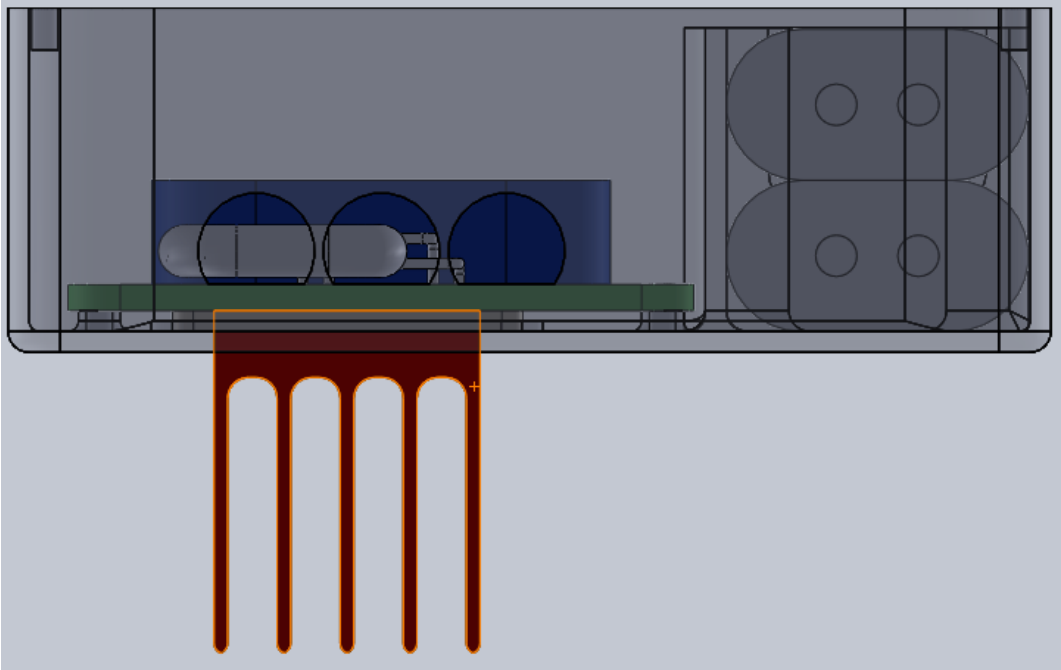


Figure C.9: Rev 1: Heatsink attached to PCB and protruding through the enclosure

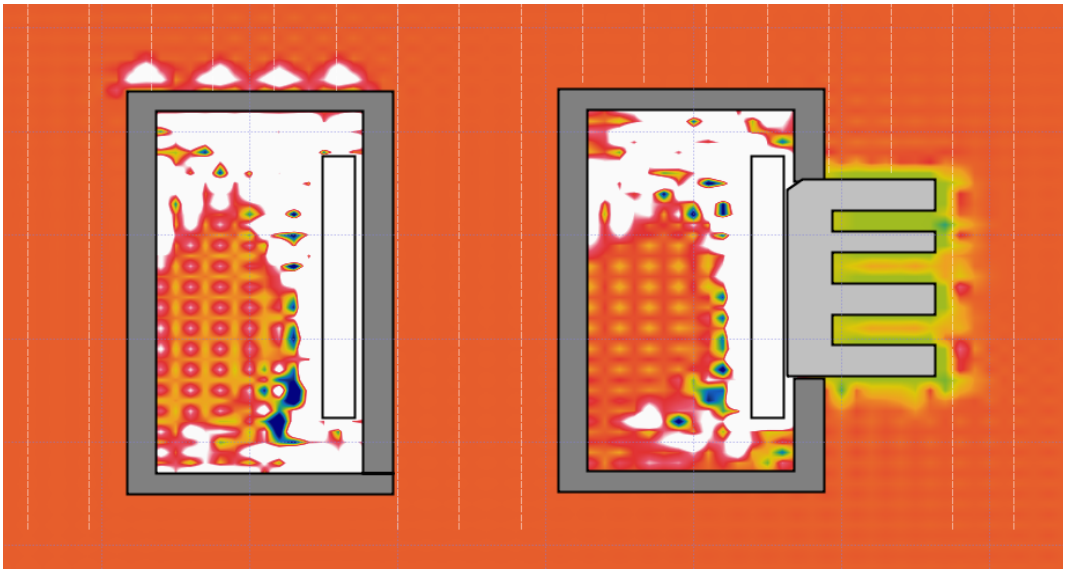


Figure C.10: An Energy 2D simulation: On the left: Phone in enclosure with no heatsink. On the right: Phone in enclosure and attached to heatsink

Appendix D Calculations

D.1 bq2057CTS Battery Management Calculations

Current Regulation

$$R_{SNS} = \frac{V_{SNS}}{IO_{REG}} \tag{D.1}$$
$$= \frac{105mV}{0.2 \cdot 4400mAHr} \tag{D.2}$$
$$= 119m\Omega \tag{D.3}$$

Voltage Regulation

$$\frac{R_{B1}}{R_{B2}} = \left(N \cdot \frac{V_{CELL}}{V_{OREG}}\right) - 1 \tag{D.4}$$

Temperature Monitoring
Using PTC thermistor
External PNP Transistor
Max Power disipation when battery is lowest (3V)

$$V_{in} = 5.2$$
$$I_{REG} = 880mA$$
$$V_{CS} = 0.176$$
$$V_D = 0.3$$

$$P_D = ((V_{in} - V_D - V_{CS}) - V_{BAT}) * I_{REG} \tag{D.5}$$
$$= (5.2 - 0.3 - 0.176 - 3) \cdot 0.88 \tag{D.6}$$
$$= 1.517W \tag{D.7}$$

Calculate β_{min}

$$\beta_{min} = \frac{I_{Cmax}}{I_B} \tag{D.8}$$
$$= \frac{0.88}{0.035} \tag{D.9}$$
$$= 25.14 \tag{D.10}$$

Input Capacitor
$0.1\mu F$ Ceramic recommended on Vcc and Vss pins
Automatic Charge Rate Compensation

$$V(z) = 0.154 \tag{D.11}$$
$$V(comp) = \frac{0.154}{2.2} \tag{D.12}$$
$$= 0.07 \tag{D.13}$$

$$V(pack) = 4.2 + (2.2 \cdot 0.07) \tag{D.14}$$
$$= 4.354 \tag{D.15}$$

$$\frac{V_{comp}}{V_{sns}} = \frac{R_{comp2}}{R_{comp1} + R_{comp2}} \tag{D.16}$$

The data sheet recommended that R_{comp2} be $10k\Omega$ and that V_{sns} be $105mV$

$$R_{comp1} = \frac{R_{comp2} \cdot (V_{sns} - V_{comp})}{V_{comp}} \tag{D.17}$$

$$R_{comp1} = \frac{10k \cdot (0.105 - 0.07)}{0.07} \tag{D.18}$$

$$R_{comp1} = 5k\Omega \tag{D.19}$$

NTC Calculations The BC2384-ND $47k\Omega$ NTC Thermistor was used RT_2 was calculated because only the upper bound is of interest The upper bound temperature was set to $50°C$

$$R_{TH} = 18k\Omega \text{ (Property of the NTC)} \tag{D.20}$$

$$R_{TC} = 673k\Omega \text{ Property of the NTC)} \tag{D.21}$$

$$R_{T2} = \frac{5 \cdot R_{TH} \cdot R_{TC}}{(2 \cdot R_{TC}) - (7 \cdot R_{TH})} \tag{D.22}$$

$$= 49.65k\Omega \tag{D.23}$$

$$\cong 49.9\Omega \tag{D.24}$$

D.2 Current and Voltage Sensing

Using the LTC6800 **Output Voltage**

$$V_{out} = \left(1 + \frac{R_2}{R_1}\right) \cdot V_{in} \tag{D.25}$$

ADC Resolution 12-bit

$$\frac{3.3V}{2^{12}} = 805\mu V \tag{D.26}$$

Choose gain of LTC6800

$$ADCres = \frac{805\mu V}{bit} \tag{D.27}$$

I want 1mA/bit resolution (D.28)

$$1mA = \frac{805\mu V}{1bit} \cdot \frac{I}{V_{AmpOut}} \tag{D.29}$$

$$V_{AmpOut} = 3.3V \text{ (Limited by ADC supplly)} \tag{D.30}$$

$$I = 4.099 \tag{D.31}$$

$$V_{AmpOut} = Gain \cdot I \cdot R_{sense} \tag{D.32}$$

$$R_{sense} = 0.01\Omega \tag{D.33}$$

$$Gain = 80.5 \tag{D.34}$$

Choose resistors for gain

$$V_{out} = \left(1 + \frac{R_2}{R_1}\right) \cdot V_{in} + V_{REF} \tag{D.35}$$

$$V_{REF} = 0V \tag{D.36}$$

$$V_{in} = 0.04V \tag{D.37}$$

$$V_{out} = 3.3V \tag{D.38}$$

$$Gain = 1 + \frac{R_2}{R_1} = 80.5 \tag{D.39}$$

Choose resistors to be $R_2 = 200k\Omega$ and $R_1 = 2.55k\Omega$.

D.3 SPV1040 Calculations

Component Values
These are constraints or known values for the calculations

$$
\begin{aligned}
F_{SW} &= 70k - 130k, typ: 100k \\
V_{IN_RP_MAX} &= 1V \\
I_{SC} &= 1A \\
T_{MPP} &= \ 1ms \text{ (tracking time)} \\
R_{MPPTset} &= 1k\Omega \\
I_{MP} &= 0.44A \text{ (current at maximum power point)} \\
I_{SC} &= 0.5A \text{ (short circuit current)} \\
V_{MP} &= 3.0V \text{ (Voltage at maximum power point)} \\
V_{OC} &= 3.4V \text{ (Open Circuit voltage)} \\
V_{out_rp_max} &= 0.01 \text{ (Maximum output voltage ripple} \\
V_{OUT_MAX} &= 5.2V \text{ (Desired Maximum output voltage)} \\
I_{OUT_MAX} &= 1.8V \text{ (Limitation of the SPV1040)}
\end{aligned}
$$

Input Capacitor

$$
\begin{aligned}
C_{IN} &\geq \frac{I_{SC}}{F_{SW} \cdot V_{IN_RP_MAX}} && \text{(D.40)} \\
&\geq \frac{1}{70k \cdot 1} && \text{(D.41)} \\
&\geq 14.3\mu F && \text{(D.42)}
\end{aligned}
$$

Input Voltage Sensing Capacitor

$$
\begin{aligned}
C_{MPPTset} &\leq T_{MPP} \cdot \frac{1}{R_{MPPTset}} && \text{(D.43)} \\
C_{MPPTset} &\leq 10\mu F && \text{(D.44)}
\end{aligned}
$$

Inductor Selection

$$
\begin{aligned}
I_{Lrms} &\cong I_{MP} \leq I_{SC} && \text{(D.45)} \\
I_{peak} &= I_{Lrms} + \frac{9e-6 \cdot V_{MP}}{2 \cdot L} && \text{(D.46)} \\
I_{Lpeak} &\leq 1.8 \text{ Limitation of SPV1040} && \text{(D.47)} \\
L &\geq \frac{1}{2}\frac{9e-6 \cdot V_{MP}}{2 - I_{Lrms}} = \frac{1}{2}\frac{9e-6 \cdot V_{MP}}{2 - I_{MP}} && \text{(D.48)} \\
L &\geq 8.5\mu H && \text{(D.49)}
\end{aligned}
$$

Output Voltage Capacitor

$$
\begin{aligned}
Cout &\geq \frac{I_{SC}}{F_{SW} \cdot V_{out_rp_max}} && \text{(D.50)} \\
Cout &\geq 142\mu F && \text{(D.51)}
\end{aligned}
$$

Output Voltage Partitioning

$$
\begin{aligned}
\frac{R_{VctrlTOP}}{R_{VctrlBOT}} &= \frac{V_{OUT_MAX}}{1.25} - 1 && \text{(D.52)} \\
R_{VctrlTOP} &= 76.8k\Omega && \text{(D.53)} \\
R_{VctrlBOT} &= 24.3k\Omega && \text{(D.54)}
\end{aligned}
$$

Output Voltage Sense Capacitor

$$C_{Vctrl} \cong 10 \cdot \frac{1}{F_{SW}} \cdot \frac{1}{R_{VctrlTOP}/R_{VctrlBOT}} \tag{D.55}$$

$$C_{Vctrl} \cong 45.2\mu F \tag{D.56}$$

Output Current sensing filter

$$R_S = \frac{50mV}{I_{OUT_MAX}} \tag{D.57}$$

$$R_S = 27m\Omega \tag{D.58}$$

Schottky Diode
Must satisfy the following:

$$V_F \leq 5.5 - V_{BATT_MAX} \text{ and } I_F \geq I_{Lmax} V_F \leq 0.8 \text{ and } I_F \geq I_{Lmax}$$

The *Diodes Incorporated* 1N5819HW has a V_F of 450mV and a non repetitive peak current of 25A.

D.4 LM61428 Calculations

The values for the components were recommended by the data sheet.

$$L = 6.8\mu H$$
$$C_{in} = 22\mu F$$
$$C_{out} = 68\mu F$$

The diode was recommended to be a Schottky with I_f higher than the load current, 500mA. The reverse voltage must be higher than the output voltage and it must be able to switch fast.

Appendix E Data Sheet Snippets

This appendix contains the 1st sheet of the data sheets for the parts used in this project.

TEXAS INSTRUMENTS

ADS1013
ADS1014
ADS1015

www.ti.com

SBAS473C – MAY 2009 – REVISED OCTOBER 2009

Ultra-Small, Low-Power, 12-Bit Analog-to-Digital Converter with Internal Reference

Check for Samples: ADS1013 ADS1014 ADS1015

FEATURES

- **ULTRA-SMALL QFN PACKAGE: 2mm × 1,5mm × 0,4mm**
- **WIDE SUPPLY RANGE: 2.0V to 5.5V**
- **LOW CURRENT CONSUMPTION: Continuous Mode: Only 150µA Single-Shot Mode: Auto Shut-Down**
- **PROGRAMMABLE DATA RATE: 128SPS to 3.3kSPS**
- **INTERNAL LOW-DRIFT VOLTAGE REFERENCE**
- **INTERNAL OSCILLATOR**
- **INTERNAL PGA**
- **I²C™ INTERFACE: Pin-Selectable Addresses**
- **FOUR SINGLE-ENDED OR TWO DIFFERENTIAL INPUTS (ADS1015)**
- **PROGRAMMABLE COMPARATOR (ADS1014 and ADS1015)**

APPLICATIONS

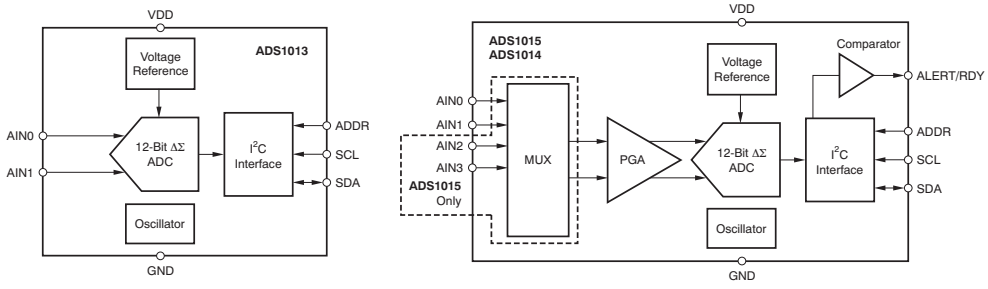
- **PORTABLE INSTRUMENTATION**
- **CONSUMER GOODS**
- **BATTERY MONITORING**
- **TEMPERATURE MEASUREMENT**
- **FACTORY AUTOMATION AND PROCESS CONTROLS**

DESCRIPTION

The ADS1013, ADS1014, and ADS1015 are precision analog-to-digital converters (ADCs) with 12 bits of resolution offered in an ultra-small, leadless QFN-10 package or an MSOP-10 package. The ADS1013/4/5 are designed with precision, power, and ease of implementation in mind. The ADS1013/4/5 feature an onboard reference and oscillator. Data are transferred via an I²C-compatible serial interface; four I²C slave addresses can be selected. The ADS1013/4/5 operate from a single power supply ranging from 2.0V to 5.5V.

The ADS1013/4/5 can perform conversions at rates up to 3300 samples per second (SPS). An onboard PGA is available on the ADS1014 and ADS1015 that offers input ranges from the supply to as low as ±256mV, allowing both large and small signals to be measured with high resolution. The ADS1015 also features an input multiplexer (MUX) that provides two differential or four single-ended inputs.

The ADS1013/4/5 operate either in continuous conversion mode or a single-shot mode that automatically powers down after a conversion and greatly reduces current consumption during idle periods. The ADS1013/4/5 are specified from –40°C to +125°C.



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

I2C is a trademark of NXP Semiconductors.
All other trademarks are the property of their respective owners.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of the Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

Figure E.1: 1st Page of ADS1013 Datasheet

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash progam memory (ATmega48PA/88PA/168PA/328P)
  - 256/512/512/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
  - 512/1K/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V for ATmega48PA/88PA/168PA/328P
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 20 MHz @ 1.8 - 5.5V
- Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:
  - Active Mode: 0.2 mA
  - Power-down Mode: 0.1 A
  - Power-save Mode: 0.75 A (Including 32 kHz RTC)

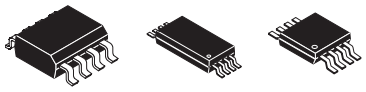
8-bit AVR® Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash

ATmega48PA
ATmega88PA
ATmega168PA
ATmega328P

Rev. 8161D–AVR–10/09

Figure E.2: 1st Page of ATmega328P Datasheet

TEXAS INSTRUMENTS



bq2057, bq2057C
bq2057T, bq2057W

SLUS025F – MAY 2001 – REVISED JULY 2002

ADVANCED LINEAR CHARGE MANAGEMENT IC FOR SINGLE- AND TWO-CELL LITHIUM-ION AND LITHIUM-POLYMER

FEATURES

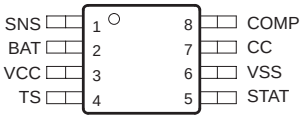
- **Ideal for Single (4.1 V or 4.2 V) and Dual-Cell (8.2 V or 8.4 V) Li-Ion or Li-Pol Packs**
- **Requires Small Number of External Components**
- **0.3 V Dropout Voltage for Minimizing Heat Dissipation**
- **Better Than ±1% Voltage Regulation Accuracy With Preset Voltages**
- **AutoComp™ Dynamic Compensation of Battery Pack's Internal Impedance to Reduce Charge Time**
- **Optional Cell-Temperature Monitoring Before and During Charge**
- **Integrated Voltage and Current Regulation With Programmable Charge-Current and High- or Low-Side Current Sensing**
- **Integrated Cell Conditioning for Reviving Deeply Discharged Cells and Minimizing Heat Dissipation During Initial Stage Of Charge**
- **Charge Status Output for Single or Dual Led or Host Processor Interface**
- **Automatic Battery-Recharge Feature**
- **Charge Termination by Minimum Current**
- **Automatic Low-Power Sleep Mode When V_{CC} Is Removed**
- **EVMs Available for Quick Evaluation**
- **Packaging: 8-Pin SOIC, 8-Pin TSSOP, 8-Pin MSOP**

DESCRIPTION

The BENCHMARQ bq2057 series advanced Lithium-Ion (Li-Ion) and Lithium-Polymer (Li-Pol) linear charge-management ICs are designed for cost-sensitive and compact portable electronics. They combine high-accuracy current and voltage regulation, battery conditioning, temperature monitoring, charge termination, charge-status indication, and AutoComp charge-rate compensation in a single 8-pin IC. MSOP, TSSOP, and SOIC package options are offered to fit a wide range of end applications.

The bq2057 continuously measures battery temperature using an external thermistor. For safety, the bq2057 inhibits charge until the battery temperature is within user-defined thresholds. The bq2057 then charges the battery in three phases: conditioning, constant current, and constant voltage. If the battery voltage is below the low-voltage threshold, $V_{(min)}$, the bq2057 precharges using a low current to condition the battery. The conditioning charge rate is approximately 10% of the regulation current. The conditioning current also minimizes heat dissipation in the external pass-element during the initial stage of the charge. After conditioning, the bq2057 applies a constant current to the battery. An external sense-resistor sets the current. The sense-resistor can be on either the high or low side of the battery without additional components. The constant-current phase continues until the battery reaches the charge-regulation voltage.

bq2057xSN or bq2057xTS
SOIC (SN) or TSSOP (TS) PACKAGE
(TOP VIEW)



bq2057xDGK
MSOP (DGK) PACKAGE
(TOP VIEW)

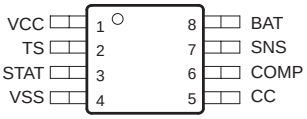


Figure E.3: 1st Page of bq2057 Datasheet

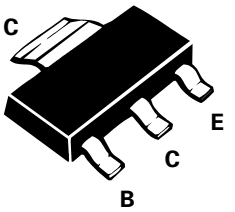
SOT223 PNP SILICON PLANAR MEDIUM POWER HIGH GAIN TRANSISTOR

FZT788B

ISSUE 3 - OCTOBER 1995

FEATURES

* Low equivalent on-resistance; **$R_{CE(sat)}$ 93mΩ at 3A**
* Gain of 300 at I_C=2 Amps and Very low saturation voltage



APPLICATIONS

* Battery powered circuits

COMPLEMENTAY TYPE – FZT688B

PARTMARKING DETAIL – FZT788B

ABSOLUTE MAXIMUM RATINGS.

PARAMETER	SYMBOL	VALUE	UNIT
Collector-Base Voltage	V_{CBO}	-15	V
Collector-Emitter Voltage	V_{CEO}	-15	V
Emitter-Base Voltage	V_{EBO}	-5	V
Peak Pulse Current	I_{CM}	-8	A
Continuous Collector Current	I_C	-3	A
Power Dissipation at T_{amb}=25°C	P_{tot}	2	W
Operating and Storage Temperature Range	T_j:T_{stg}	-55 to +150	°C

ELECTRICAL CHARACTERISTICS (at T_{amb} = 25°C)

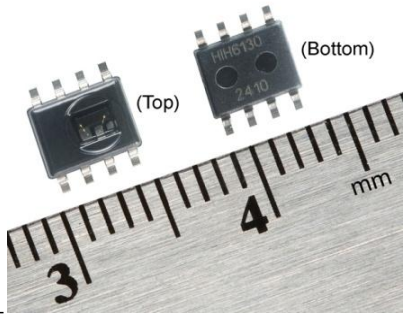
PARAMETER	SYMBOL	MIN.	TYP.	MAX.	UNIT	CONDITIONS.
Collector-Base Breakdown Voltage	$V_{(BR)CBO}$	-15			V	I_C=-100μA
Collector-Emitter Breakdown Voltage	$V_{(BR)CEO}$	-15			V	I_C=-10mA*
Emitter-Base Breakdown Voltage	$V_{(BR)EBO}$	-5			V	I_E=-100μA
Collector Cut-Off Current	I_{CBO}			-0.1	μA	V_{CB}=-10V
Emitter Cut-Off Current	I_{EBO}			-0.1	μA	V_{EB}=-4V
Collector-Emitter Saturation Voltage	$V_{CE(sat)}$			-0.15 -0.25 -0.45 -0.5	V V V	I_C=-0.5A, I_B=-2.5mA* I_C=-1A, I_B=-5mA* I_C=-2A, I_B=-10mA* I_C=-3A, I_B=-50mA*
Base-Emitter Saturation Voltage	$V_{BE(sat)}$			-0.9	V	I_C=-1A, I_B=-5mA*
Base-Emitter Turn-On Voltage	$V_{BE(on)}$		-0.75		V	I_C=-1A, V_{CE}=-2V*
Static Forward Current Transfer Ratio	h_{FE}	500 400 300 150		1500		I_C=-10mA, V_{CE}=-2V* I_C=-1A, V_{CE}=-2V* I_C=-2A, V_{CE}=-2V* I_C=-6A, V_{CE}=-2V*
Transition Frequency	f_T	100			MHz	I_C=-50mA, V_{CE}=-5V f=50MHz
Input Capacitance	C_{ibo}		225		pF	V_{EB}=-0.5V, f=1MHz
Output Capacitance	C_{obo}		25		pF	V_{CB}=-10V, f=1MHz
Switching Times	t_{on} t_{off}		35 400		ns ns	I_C=-500mA, I_{B1}=-50mA I_{B2}=-50mA, V_{CC}=-10V

*Measured under pulsed conditions. Pulse width=300μs. Duty cycle ≤2%
Spice parameter data is available upon request for this device

Figure E.4: 1st Page of FZT788B Datasheet

Honeywell

Honeywell HumidIcon™ Digital Humidity/Temperature Sensors: HIH-6130/6131 Series



DESCRIPTION (★ = *competitive differentiator)*
Honeywell HumidIcon™ Digital Humidity/Temperature Sensors: HIH-6130/6131 Series, is a digital output-type relative humidity (RH) and temperature sensor combined in the same package. These devices offer several competitive advantages, including:

- Industry-leading Total Error Band
- Industry-leading stability
- Industry-leading reliability
- Lowest total cost solution
- True temperature-compensated digital I^2C output
- Energy efficiency
- Ultra-small package

★ **Industry-leading Total Error Band (TEB) (±5 %RH):** Honeywell specifies Total Error Band—the most comprehensive, clear, and meaningful measurement –that provides the sensor's true accuracy of ±5 %RH over a compensated range of 5 °C to 50 °C [41 °F to 122 °F] and 10 %RH to 90 %RH. TEB includes all errors due to:

- Humidity non-linearity
- Humidity hysteresis
- Humidity non-repeatability
- Thermal effect on zero
- Thermal effect on span
- Thermal hysteresis

Total Error Band should not be confused with "Accuracy", which is actually a component of Total Error Band. Many competitors simply specify the accuracy of their device; however, the specification may exclude hysteresis and temperature effects, and may be calculated over a very narrow range, at only one point in the range, or at their absolute best accuracy level. It is then up to the customer to calibrate the device to make sure it has the accuracy needed for the life of the application.

Honeywell's industry-leading Total Error Band provides the following benefits to the customer:

- Eliminates individually testing and calibrating every sensor, which can increase their manufacturing time and process
- Supports system accuracy and warranty requirements
- Helps to optimize system uptime
- Provides excellent sensor interchangeability—the customer can remove one sensor from the tape, remove the next sensor from the tape, and there is no part-to-part variation in accuracy

For more information about Total Error Band, please see the related Technical Note "Explanation of the Total Error Band Specification for Honeywell's Digital Humidity/Temperature Sensors."

★ **Industry-leading long term stability (1.2 %RH over five years):** Competitive humidity sensors need to go through a 12 hour at 75 %RH rehydration process (which requires special equipment chambers) to correct reflow temperature offset. Honeywell's sensor also experiences an offset after reflow; however, it only requires a five hour rehydration under ambient conditions (>50 %RH). Honeywell's industry-leading long term stability provides the following benefits to the customer:

- Minimizes system performance issues
- Helps support system uptime by eliminating the need to service or replace the sensor during its application life
- Eliminates the need to regularly recalibrate the sensor in their application, which can be inconvenient and costly

★ **Industry-leading reliability:** Honeywell's new HIH-6130/6131 Series sensors use a laser trimmed, thermoset polymer capacitive sensing element. The element's multilayer construction provides resistance to most application hazards such as condensation, dust, dirt, oils, and common environmental chemicals which help provide industry-leading stability and reliability.

Figure E.5: 1st Page of HIH6130 Datasheet

湖南桑顿新能源有限公司
Hunan Sounddon New Energy Co.,Ltd

Specification No. APL-2013517
Edition No. 1.0

Cylindrical Li－ion Battery Pack

Product Specification

Model：18650 3.7v 4400mAh

Hunan Sounddon New Energy Co., Ltd

Address:No.98, Fuzhou Road,Jinhua Demonstration Area,Xiangtan City,Hunan Province,China.

http://www.soundnewenergy.com

Tel: (86) 731 - 5856 7126

Fax: (86) 731 - 5823 6346

E-mail:linda.ding@soundnewenergy.com

Prepared by：Liu Li

Checked by：Zhu Qiang

Approve by: Xiao Linping

All 10 Sheets

Figure E.6: 1st Page of Li-Ion Battery Pack Datasheet

LM75B

Digital temperature sensor and thermal watchdog

Rev. 6.1 — 6 February 2015 Product data sheet

1. General description

The LM75B is a temperature-to-digital converter using an on-chip band gap temperature sensor and Sigma-Delta A-to-D conversion technique with an overtemperature detection output. The LM75B contains a number of data registers: Configuration register (Conf) to store the device settings such as device operation mode, OS operation mode, OS polarity and OS fault queue as described in Section 7 “Functional description”; temperature register (Temp) to store the digital temp reading, and set-point registers (Tos and Thyst) to store programmable overtemperature shutdown and hysteresis limits, that can be communicated by a controller via the 2-wire serial I^2C-bus interface. The device also includes an open-drain output (OS) which becomes active when the temperature exceeds the programmed limits. There are three selectable logic address pins so that eight devices can be connected on the same bus without address conflict.

The LM75B can be configured for different operation conditions. It can be set in normal mode to periodically monitor the ambient temperature, or in shutdown mode to minimize power consumption. The OS output operates in either of two selectable modes: OS comparator mode or OS interrupt mode. Its active state can be selected as either HIGH or LOW. The fault queue that defines the number of consecutive faults in order to activate the OS output is programmable as well as the set-point limits.

The temperature register always stores an 11-bit two’s complement data giving a temperature resolution of 0.125 °C. This high temperature resolution is particularly useful in applications of measuring precisely the thermal drift or runaway. When the LM75B is accessed the conversion in process is not interrupted (that is, the I^2C-bus section is totally independent of the Sigma-Delta converter section) and accessing the LM75B continuously without waiting at least one conversion time between communications will not prevent the device from updating the Temp register with a new conversion result. The new conversion result will be available immediately after the Temp register is updated.

The LM75B powers up in the normal operation mode with the OS in comparator mode, temperature threshold of 80 °C and hysteresis of 75 °C, so that it can be used as a stand-alone thermostat with those pre-defined temperature set points.

2. Features and benefits

- Pin-for-pin replacement for industry standard LM75 and LM75A and offers improved temperature resolution of 0.125 °C and specification of a single part over power supply range from 2.8 V to 5.5 V
- I^2C-bus interface with up to 8 devices on the same bus
- Power supply range from 2.8 V to 5.5 V
- Temperatures range from −55 °C to +125 °C

Figure E.7: 1st Page of LM75B Datasheet

TEXAS INSTRUMENTS

LMR61428

www.ti.com SNVS815A – JUNE 2012 – REVISED APRIL 2013

LMR61428 SIMPLE SWITCHER® 14Vout, 2.85A Step-Up Voltage Regulator in VSSOP

Check for Samples: LMR61428

FEATURES

- 1.2V to 14V Input Voltage
- Adjustable Output Voltage up to 14V
- Switch Current up to 2.85A
- Up to 2 MHz Switching Frequency
- Low Shutdown Iq, <1µA
- Cycle-by-Cycle Current Limiting
- VSSOP Packaging (3.0 x 5.0 x 1.09mm)
- WEBENCH® Enabled

PERFORMANCE BENEFITS

- Extremely Easy to Use
- Tiny Overall Solution Reduces System Cost

APPLICATIONS

- Boost/SEPIC Conversions from 3.3V, 5V, and 12V
- Space Constrained Applications
- LCD Displays
- LED Applications

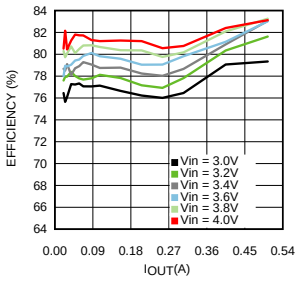
DESCRIPTION

The LMR61428 is a step-up DC-DC switching regulator for battery-powered and low input voltage systems that can achieve efficiencies up to 90%. It has a wide input voltage range from 1.2V to 14V and a possible regulated output voltage range of 1.24V to 14V. It has an internal 0.17Ω N-Channel MOSFET power switch.

The high switching frequency of up to 2MHz of the LMR61428 allows for tiny surface mount inductors and capacitors. Because of the unique constant-duty-cycle gated oscillator topology very high efficiencies are realized over a wide load range. The supply current is reduced to 80µA because of the BiCMOS process technology. In the shutdown mode, the supply current is less than 2.5µA. The LMR61428 is available in a VSSOP-8 package.

System Performance

Efficiency vs Load Current, V_{OUT} = 5V



Efficiency vs Load Current, V_{OUT} = 3.3V

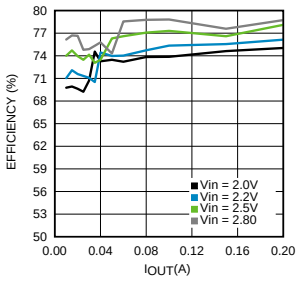


Figure E.8: 1st Page of LMR61428 Datasheet

LINEAR TECHNOLOGY

LTC4412

Low Loss PowerPath™ Controller in ThinSOT

FEATURES

- **Very Low Loss Replacement for Power Supply OR'ing Diodes**
- **Minimal External Components**
- **Automatic Switching Between DC Sources**
- **Simplifies Load Sharing with Multiple Batteries**
- **Low Quiescent Current: 11µA**
- 3V to 28V AC/DC Adapter Voltage Range
- 2.5V to 28V Battery Voltage Range
- Reverse Battery Protection
- Drives Almost Any Size MOSFET for Wide Range of Current Requirements
- MOSFET Gate Protection Clamp
- Manual Control Input
- Low Profile (1mm) ThinSOT™ Package

APPLICATIONS

- Cellular Phones
- Notebook and Handheld Computers
- Digital Cameras
- USB-Powered Peripherals
- Uninterruptible Power Supplies
- Logic Controlled Power Switch

DESCRIPTION

The LTC®4412 controls an external P-channel MOSFET to create a near ideal diode function for power switchover or load sharing. This permits highly efficient OR'ing of multiple power sources for extended battery life and low self-heating. When conducting, the voltage drop across the MOSFET is typically 20mV. For applications with a wall adapter or other auxiliary power source, the load is automatically disconnected from the battery when the auxiliary source is connected. Two or more LTC4412s may be interconnected to allow load sharing between multiple batteries or charging of multiple batteries from a single charger.

The wide supply operating range supports operation from one to six Li-Ion cells in series. The low quiescent current (11µA typical) is independent of the load current. The gate driver includes an internal voltage clamp for MOSFET protection.

The STAT pin can be used to enable an auxiliary P-channel MOSFET power switch when an auxiliary supply is detected. This pin may also be used to indicate to a microcontroller that an auxiliary supply is connected. The control (CTL) input enables the user to force the primary MOSFET off and the STAT pin low.

The LTC4412 is available in a low profile (1mm) ThinSOT package.

TYPICAL APPLICATION

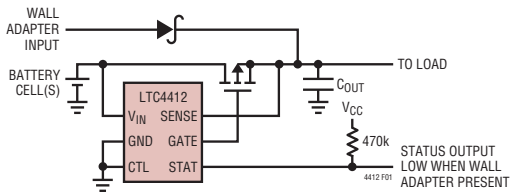


Figure 1. Automatic Switchover of Load Between a Battery and a Wall Adapter

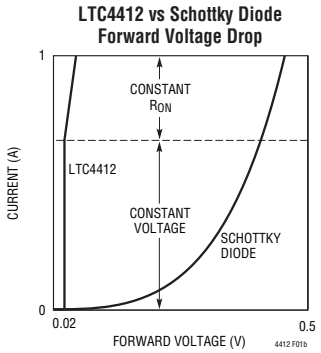


Figure E.9: 1st Page of LTC4412 Datasheet

LINEAR TECHNOLOGY

LTC6800

Rail-to-Rail, Input and Output, Instrumentation Amplifier

FEATURES

- **116dB CMRR Independent of Gain**
- **Maximum Offset Voltage: 100µV**
- **Maximum Offset Voltage Drift: 250nV/°C**
- **–40°C to 125°C Operation**
- Rail-to-Rail Input Range
- Rail-to-Rail Output Swing
- Supply Operation: 2.7V to 5.5V
- Available in MS8 and 3mm × 3mm × 0.8mm DFN Packages

APPLICATIONS

- Thermocouple Amplifiers
- Electronic Scales
- Medical Instrumentation
- Strain Gauge Amplifiers
- High Resolution Data Acquisition

DESCRIPTION

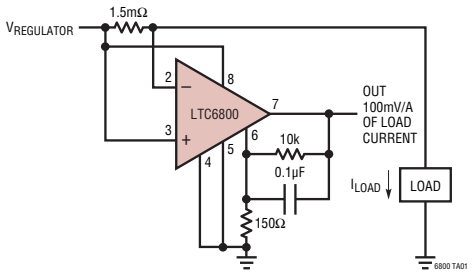
The LTC®6800 is a precision instrumentation amplifier. The CMRR is typically 116dB with a single 5V supply and is independent of gain. The input offset voltage is guaranteed below 100µV with a temperature drift of less than 250nV/°C. The LTC6800 is easy to use; the gain is adjustable with two external resistors, like a traditional op amp.

The LTC6800 uses charge balanced sampled data techniques to convert a differential input voltage into a single ended signal that is in turn amplified by a zero-drift operational amplifier.

The differential inputs operate from rail-to-rail and the single ended output swings from rail-to-rail. The LTC6800 is available in an MS8 surface mount package. For space limited applications, the LTC6800 is available in a 3mm × 3mm × 0.8mm dual fine pitch leadless package (DFN).

TYPICAL APPLICATION

High Side Power Supply Current Sense



Typical Input Referred Offset vs Input Common Mode Voltage (V_S = 3V)

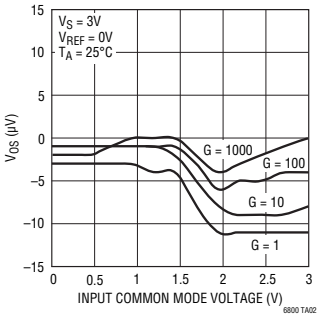


Figure E.10: 1st Page of LTC6800 Datasheet

TEXAS INSTRUMENTS

SM72238

www.ti.com SNVS694C – JANUARY 2011 – REVISED APRIL 2013

Micropower Voltage Regulator

Check for Samples: SM72238

FEATURES

- Renewable Energy Grade
- High-Accuracy Output Voltage
- Ensured 100mA Output Current
- Extremely Low Quiescent Current
- Low Dropout Voltage
- Extremely Tight Load and Line Regulation
- Very Low Temperature Coefficient
- Use as Regulator or Reference
- Needs Minimum Capacitance for Stability
- Current and Thermal Limiting
- Stable With Low-ESR Output Capacitors (10mΩ to 6Ω)

DESCRIPTION

The SM72238 is a micropower voltage regulator with very low quiescent current (75µA typ.) and very low dropout voltage (typ. 40mV at light loads and 380mV at 100mA). It is ideally suited for use in battery-powered systems. Furthermore, the quiescent current of the SM72238 increases only slightly in dropout, prolonging battery life.

The SM72238 is available in the surface-mount D-Pak package.

Careful design of the SM72238 has minimized all contributions to the error budget. This includes a tight initial tolerance (.5% typ.), extremely good load and line regulation (.05% typ.) and a very low output voltage temperature coefficient, making the part useful as a low-power voltage reference.

Block Diagram and Typical Applications

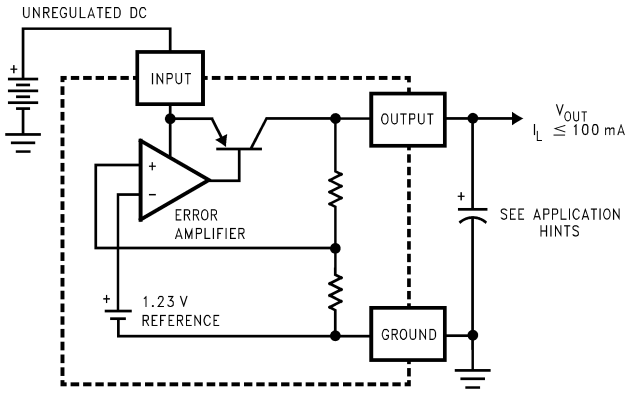


Figure 1. SM72238

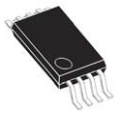
Figure E.11: 1st Page of SM72238 Datasheet

life.augmented

SPV1040

High efficiency solar battery charger with embedded MPPT

Datasheet - production data



TSSOP8

Features

- 0.3 V to 5.5 V operating input voltage
- 140 mΩ internal synchronous rectifier
- 120 mΩ internal power active switch
- 100 kHz fixed PWM frequency
- Duty cycle controlled by MPPT algorithm
- Output voltage regulation, overcurrent and overtemperature protection
- Input source reverse polarity protection
- Built-in soft-start
- Up to 95% efficiency
- 3 mm x 4.4 mm TSSOP8 package

Applications

- Smart phones and GPS systems
- Wireless headsets
- Small appliances, sensors
- Portable media players
- Digital still cameras
- Toys and portable healthcare

Description

The SPV1040 device is a low power, low voltage, monolithic step-up converter with an input voltage range from 0.3 V to 5.5 V, and is capable of maximizing the energy generated by even a single solar cell (or fuel cell), where low input voltage handling capability is extremely important.

Thanks to the embedded MPPT algorithm, even under varying environmental conditions (such as irradiation, dirt, temperature) the SPV1040 offers maximum efficiency in terms of power harvested from the cells and transferred to the output.

The device employs an input voltage regulation loop, which fixes the charging battery voltage via a resistor divider. The maximum output current is set with a current sense resistor according to charging current requirements.

The SPV1040 protects itself and other application devices by stopping the PWM switching if either the maximum current threshold (up to 1.8 A) is reached or the maximum temperature limit (up to 155 °C) is exceeded.

An additional built-in feature of the SPV1040 is the input source reverse polarity protection, which prevents damage in case of reverse connection of the solar panel at the input.

Table 1. Device summary

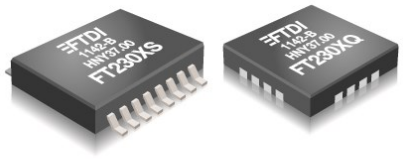
Order code	Package	Packaging
SPV1040T	TSSOP8	Tube
SPV1040TTR		Tape and reel

Figure E.12: 1st Page of SPV1040 Datasheet

Future Technology Devices International Ltd.

FT230X

(USB to BASIC UART IC)



The FT230X is a USB to serial UART interface with optimised pin count for smaller PCB designs and the following advanced features:

- Single chip USB to asynchronous serial data transfer interface.
- Entire USB protocol handled on the chip. No USB specific firmware programming required.
- Fully integrated 2048 byte multi-time-programmable (MTP) memory, storing device descriptors and CBUS I/O configuration.
- Fully integrated clock generation with no external crystal required plus optional clock output selection enabling a glue-less interface to external MCU or FPGA.
- Data transfer rates from 300 baud to 3 Mbaud (RS422, RS485, and RS232) at TTL levels.
- 512 byte receive buffer and 512 byte transmit buffer utilising buffer smoothing technology to allow for high data throughput.
- FTDI's royalty-free Virtual Com Port (VCP) and Direct (D2XX) drivers eliminate the requirement for USB driver development in most cases.
- Configurable CBUS I/O pins.
- Transmit and receive LED drive signals.
- UART interface support for 7 or 8 data bits, 1 or 2 stop bits and odd / even / mark / space / no parity
- Synchronous and asynchronous bit bang interface options with RD# and WR# strobes.
- USB Battery Charger Detection. Allows for USB peripheral devices to detect the presence of a higher power source to enable improved charging.
- Device supplied pre-programmed with unique USB serial number.
- USB Power Configurations; supports bus- powered, self-powered and bus-powered with power switching
- Integrated +3.3V level converter for USB I/O.
- True 3.3V CMOS drive output and TTL input; operates down to 1V8 with external pull ups. Tolerant of 5V input
- Configurable I/O pin output drive strength; 4 mA (min) and 16 mA (max).
- Integrated power-on-reset circuit.
- Fully integrated AVCC supply filtering - no external filtering required.
- UART signal inversion option.
- + 5V Single Supply Operation.
- Internal 3V3/1V8 LDO regulators
- Low operating and USB suspend current; 8mA (active-typ) and 70uA (suspend-typ).
- UHCI/OHCI/EHCI host controller compatible.
- USB 2.0 Full Speed compatible.
- Extended operating temperature range; -40 to 85ºC.
- Available in compact Pb-free 16 pin SSOP and 16 pin QFN packages (both RoHS compliant).

Figure E.13: 1st Page of FTDI Datasheet

Appendix F Source Code

F.1 Microcontroller Source Code

The microcontroller code is all open source and can be found at https://github.com/rfcx/rfcx-microcontroller.

F.1.1 rfcx-mcu.c

Listing 2: rfcx-mcu.c

```
/*************************************************************
* RFCx Microcontroller Software - Main
*
* Kalee Stutzman    (stutzmak@mail.gvsu.edu)
* Joe Gibson     (gibsjose@mail.gvsu.edu)
*
* 08 July 2015
*
*   www.rfcx.org
*************************************************************/

#include "rfcx-mcu.h"

//Don't forget `volatile`!
volatile bool sensors = false;

//Timer 1 Compare Interrupt Vector (1s CTC Timer)
ISR(TIMER1_COMPA_vect) {
  //Blink LED
  PORTB ^= _BV(LED_PIN);

  //Initiate a sensor reading
  sensors = true;
}

int main(void) {
  //Sensor/battery structures
  batteries_t batteries;
  temp_data_t lm75;
  adc_data_t ads1015;
  humid_data_t hih6130;

  //Android Serial Structure
  android_serial_t android;

  char message[128];
#ifndef ARDUINO
  char humid_status[32];
  char battery_1_status[32];
  char battery_2_status[32];
  char tmp_str[6];
#endif//ARDUINO

  int ret = 0;

  memset(message, 0, 128);

  //Initialize USART at 9600 baud (UBRR defined in rfcx-mcu.h)
  usart_init(UBRR);
```

```
50
      //Initialization
    #ifndef ARDUINO
      usart_send_string("Initializing...\r\n");
    #endif
55    ret = init();
    #ifndef ARDUINO
      if(ret) {
         usart_send_string("<-- ERROR: Initialization failed -->\r\n");
      } else {
60       usart_send_string("Initialization successful\r\n");
      }
    #endif

      rfcx_temp_data_init(&lm75);
65    rfcx_humid_data_init(&hih6130);
      rfcx_adc_data_init(&ads1015);
      rfcx_batteries_data_init(&batteries);

      //Main Loop
70    while(true) {
         //Sensor Loop
         if(sensors) {
    #ifdef ARDUINO
           rfcx_read_temp(&lm75);
75         //rfcx_read_humid(&hih6130);
           //rfcx_read_adc(&ads1015);
           //rfcx_batteries_status(&batteries);

           rfcx_android_package(&android, &lm75, &hih6130, &ads1015, &batteries);
80         rfcx_android_serialize(message, &android);

           usart_send_string(message);
    #else//MAIN BOARD
           usart_send_string("\r\n-----------------------------\r\n");
85         //Temperature Sensor
           rfcx_read_temp(&lm75);

           usart_send_string("LM75BD:\r\n");
           dtostrf((double)lm75.temperature, 5, 2, tmp_str);
90         sprintf(message, "\tTemperature: %sC\r\n", tmp_str);
           usart_send_string(message);

           //Humidity Sensor
           rfcx_read_humid(&hih6130);
95
           usart_send_string("HIH6130:\r\n");
           dtostrf((double)hih6130.humidity, 5, 2, tmp_str);
           sprintf(message, "\tHumidity: %s%%\r\n", tmp_str);
           usart_send_string(message);
100
           dtostrf((double)hih6130.temperature, 5, 2, tmp_str);
           sprintf(message, "\tTemperature: %sC\r\n", tmp_str);
           usart_send_string(message);

105        rfcx_humid_status_string(humid_status, hih6130.status);
           sprintf(message, "\tStatus: %s\r\n", humid_status);
           usart_send_string(message);
```

```
            //Voltage/Current ADC
            // rfcx_read_adc(&ads1015);
            //
            // dtostrf((double)ads1015.input_voltage, 5, 2, tmp_str);
            // sprintf(message, "Input Voltage: %sV\r\n", tmp_str);
            // usart_send_string(message);
            //
            // dtostrf((double)ads1015.output_voltage, 5, 2, tmp_str);
            // sprintf(message, "Output Voltage: %sV\r\n", tmp_str);
            // usart_send_string(message);
            //
            // dtostrf((double)ads1015.input_current, 5, 2, tmp_str);
            // sprintf(message, "Input Current: %smA\r\n", tmp_str);
            // usart_send_string(message);
            //
            // dtostrf((double)ads1015.output_current, 5, 2, tmp_str);
            // sprintf(message, "Output Current: %smA\r\n", tmp_str);
            // usart_send_string(message);

            //Battery Status
            rfcx_batteries_status(&batteries);

            usart_send_string("Batteries:\r\n");
            rfcx_battery_status_string(battery_1_status, batteries.battery_1.status);
            rfcx_battery_status_string(battery_2_status, batteries.battery_2.status);

            sprintf(message,  "\tBattery 1 Status: %s\r\n"
                              "\tBattery 2 Status: %s\r\n",
                              battery_1_status,
                              battery_2_status);
            usart_send_string(message);

            usart_send_string("-----------------------------\r\n");
    #endif//ARDUINO
            sensors = false;
        }
      }

      return 0;
    }

    int init(void) {
      int ret = 0;

      //Clear interrupts
      cli();

      //Initialize ports
      port_init();

      //Initialize peripherals
      peripheral_init();

      //Initialize devices
      ret = device_init();

      //Enable interrupts
      sei();
```

```
  return ret;
}

int port_init(void) {
  //Initialize LED as output
  DDRB |= _BV(LED_DD);

  //Initialize battery inputs/outputs
  rfcx_batteries_init();

  return 0;
}

int timer1_init(void) {
  //Initialize Timer 1
  TCCR1A = 0;
  TCCR1B = 0;

  //Set CTC compare value (1 second)
  OCR1A = TIMER1_COUNT;

  //Enable CTC mode
  TCCR1B |= (1 << WGM12);

  //Enable 1024 prescaler
  TCCR1B |= (1 << CS10);
  TCCR1B |= (1 << CS12);

  //Enable Timer 1 output compare interrupt
  TIMSK1 |= (1 << OCIE1A);

  return 0;
}

int peripheral_init(void) {
  //Initialize Timer 1
  timer1_init();

  //Initialize I2C (TWI) peripheral as a whole
  rfcx_i2c_init();

  return 0;
}

int device_init(void) {
  int ret = 0;

  //Initialize external I2C temp sensor (LM75BD)
  ret = rfcx_temp_init();
  if(ret) {
#ifndef ARDUINO
    usart_send_string("<-- ERROR: Error initializing temp sensor -->\r\n");
#endif
    return ret;
  } else {
#ifndef ARDUINO
    usart_send_string("Successfully initialized temp sensor\r\n");
#endif
  }
```

```
  //Initialize external I2C ADC (ADS1015)
  // ret = rfcx_adc_init();
  // if(ret) {
  //     usart_send_string("<-- ERROR: Error initializing ADC -->\r\n");
  // } else {
  //     usart_send_string("Successfully initialized ADC\r\n");
  // }

  //Initialize external I2C humidity sensor (HIH6130)
  ret = rfcx_humid_init();
  if(ret) {
#ifndef ARDUINO
      usart_send_string("<-- ERROR: Error initializing humidity sensor -->\r\n");
#endif
      return ret;
  } else {
#ifndef ARDUINO
      usart_send_string("Successfully initialized humidity sensor\r\n");
#endif
  }

  return ret;
}
```

F.1.2 rfcx-mcu.h

Listing 3: rfcx-mcu.h

```
/*************************************************************
* RFCx Microcontroller Software - Main
*
* Kalee Stutzman    (stutzmak@mail.gvsu.edu)
* Joe Gibson     (gibsjose@mail.gvsu.edu)
*
* 08 July 2015
*
*   www.rfcx.org
*************************************************************/

#ifndef RFCX_MCU_H
#define RFCX_MCU_H

//ARDUINO: DEBUG FOR USART ONLY: Comment out for normal operation
//#define ARDUINO

#include "rfcx-globals.h"   //Global definitions

#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <string.h>
#include <stdio.h>
#include <stdbool.h>

#include "rfcx-i2c.h"
#include "rfcx-battery.h"
#include "rfcx-android.h"
```

```
#include "utilities/delay.h"
#include "utilities/usart.h"

//USART Settings
#define FOSC F_CPU          //Clock Speed (Hz)
#define BAUD 9600           //Baud Rate
#define UBRR (((((FOSC * 10) / (16L * BAUD)) + 5) / 10) - 1)

//Timer Definitions
#define TIMER1_COUNT    ((FOSC / 1024) - 1)     //Timer 1 count value for CTC mode:
    1 second, at 1024 prescaler

//Pin Definitions
#ifdef ARDUINO
    #define LED_PIN     PB5
    #define LED_DD      DDB5
#else
    #define LED_PIN     PB2     //PB2 is the board LED, PB5 is the Arduino LED
    #define LED_DD      DDB2
#endif

int init(void);
int port_init(void);
int timer1_init(void);
int peripheral_init(void);
int device_init(void);

#endif//RFCX_MCU_H
```

F.1.3 rfcx-i2c.c

Listing 4: rfcx-i2c.c

```
/*************************************************************
* RFCx Microcontroller Software - I2C
*
* Kalee Stutzman    (stutzmak@mail.gvsu.edu)
* Joe Gibson     (gibsjose@mail.gvsu.edu)
*
* 08 July 2015
*
*   www.rfcx.org
*************************************************************/

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

#include "rfcx-i2c.h"

//Initialize I2C peripheral
void rfcx_i2c_init(void) {
    i2c_init();
}

//Initialize temperature sensor
int rfcx_temp_init() {
    unsigned char value, value2;
```

```
        value = 0x01;
        value2 = 0x04;

        int ret = 0;

        char str[512];
        memset(str, 0, 512);

        unsigned char address;

        address = TEMP_ADDR;

        //sprintf(str, "Device address: 0x%02X\r\n", address);
        //usart_send_string(str);

        //Begin TWI communication
        ret = i2c_start(address + I2C_WRITE);
        if(ret) {
            i2c_stop();
            usart_send_string("<-- ERROR: Unable to start communication-->\r\n");
            return ERROR;
        }

        //Set the pointer to the configuration register
        ret = i2c_write(value);
        if(ret) {
            i2c_stop();
            usart_send_string("<-- ERROR: Could not set pointer to LM75 config register
                -->\r\n");
            return ERROR;
        }

        //Enable the temp sensor
        ret = i2c_write(value2);
        if(ret) {
            i2c_stop();
            usart_send_string("<-- ERROR: Could not enable LM75 -->\r\n");
            return ERROR;
        }

        //Release bus
        i2c_stop();

        return OK;
    }

    //Initialize ADC
    int rfcx_adc_init() {
        unsigned char value, value2, value3;
        value = 0x01;
        value2 = 0x41;
        value3 = 0xEF;

        i2c_start_wait(ADC_ADDR);

        //Set the pointer to the configuration register
        i2c_write(value);

        //Put the ADC in single conversion mode
```

```
        i2c_write(value2);

        //Set the data rate to 3300, disable the comparator
        i2c_write(value3);

        return OK;
    }

    //Initialize humidity sensor
    int rfcx_humid_init(void) {
        return OK;
    }

    //Shutdown (stop) I2C peripheral
    //@TODO Does this function even make sense?
    void rfcx_i2c_shutdown(void) {
        i2c_stop();
    }

    void rfcx_temp_data_init(temp_data_t * data) {
        memset(&(data->raw), 0, sizeof(temp_raw_t));
        data->temperature = 0.0;
    }

    void rfcx_humid_data_init(humid_data_t * data) {
        memset(&(data->raw), 0, sizeof(humid_raw_t));
        data->humidity = 0.0;
        data->temperature = 0.0;
        data->status = HUMID_STATUS_NORMAL;
    }

    void rfcx_adc_data_init(adc_data_t * data) {
        memset(&(data->raw), 0, sizeof(adc_raw_t));
        data->input_voltage = 0.0;
        data->input_current = 0.0;
        data->output_voltage = 0.0;
        data->output_current = 0.0;
    }

    //Shutdown temperature sensor:
    // Set Temp Sensor control address to
    // active high comparator mode, shutdown mode
    void rfcx_temp_shutdown() {
        unsigned char value, value2;
        value = 0x01;
        value2 = 0x05;

        i2c_start_wait(TEMP_ADDR);

        //Set the pointer to the configuration register
        i2c_write(value);

        //Put the temp sensor in shutdown mode
        i2c_write(value2);
        i2c_stop();
    }

    void rfcx_adc_shutdown(void) {
        return;
```

```
}

void rfcx_humid_shutdown(void) {
    return;
}

//Read Temperature data from the LM75B
int rfcx_read_temp(temp_data_t * data) {
    int ret = 0;

    //Write the pointer register in the sensor to point to the temp register
    i2c_start_wait(TEMP_ADDR + I2C_WRITE);
    ret = i2c_write(0x00);
    if(ret) {
        i2c_stop();
        usart_send_string("<-- ERROR: Could not set pointer register to temp (0x00)
            -->\r\n");
        return ERROR;
    }

    //Read both bytes for the temperature (msb first, then lsb)
    ret = i2c_rep_start(TEMP_ADDR + I2C_READ);
    if(ret) {
        i2c_stop();
        usart_send_string("<-- ERROR: Could not repeat start temp sensor-->\r\n");
        return ERROR;
    }

    data->raw.msb = i2c_readAck();
    data->raw.lsb = i2c_readNak();
    i2c_stop();

    //Convert data
    convert_temp_data(data);

    return OK;
}

int rfcx_read_adc(adc_data_t * data) {
    //Perform all reads
    rfcx_read_adc_pin(data, ADC_INPUT_VOLTAGE_PIN);
    rfcx_read_adc_pin(data, ADC_OUTPUT_VOLTAGE_PIN);
    rfcx_read_adc_pin(data, ADC_INPUT_CURRENT_PIN);
    rfcx_read_adc_pin(data, ADC_OUTPUT_CURRENT_PIN);

    //Convert data
    convert_adc_data(data);

    return OK;
}

int rfcx_read_adc_pin(adc_data_t * data, int pin) {
    unsigned char value, value2, value3;
    value = 0x01;

    switch(pin) {
        case ADC_INPUT_VOLTAGE_PIN:     //Pin AIN0 - Input voltage
            value2 = 0xC1;
            break;
```

```
            case ADC_OUTPUT_VOLTAGE_PIN:    //Pin AIN1 - Output voltage
                value2 = 0xD1;
                break;
            case ADC_INPUT_CURRENT_PIN:     //Pin AIN2 - Input current
                value2 = 0xE1;
                break;
            case ADC_OUTPUT_CURRENT_PIN:    //Pin AIN3 - Output current
                value2 = 0xF1;
                break;
            default:                        //Pin AIN0 - Input voltage
                value2 = 0xC1;
                break;
        }

        value3 = 0xEF;
        i2c_start_wait(ADC_ADDR + I2C_WRITE);

        //Set the pointer to the configuration register
        i2c_write(value);

        //Put the ADC in single conversion mode, read from AIN0
        i2c_write(value2);

        //Set the data rate to 3300, disable the comparator
        i2c_write(value3);

        value = 0x00;
        i2c_rep_start(ADC_ADDR + I2C_WRITE);
        //Set the pointer to the conversion register
        i2c_write(value);
        i2c_stop();

        unsigned char msb, lsb;

        //Read from the conversion register
        i2c_rep_start(ADC_ADDR + I2C_READ);
        msb = i2c_readAck();
        lsb = i2c_readNak();

        //Store correctly based on pin
        switch(pin) {
            case ADC_INPUT_VOLTAGE_PIN:
                data->raw.input_voltage_msb = msb;
                data->raw.input_voltage_lsb = lsb;
                break;
            case ADC_OUTPUT_VOLTAGE_PIN:
                data->raw.output_voltage_msb = msb;
                data->raw.output_voltage_lsb = lsb;
                break;
            case ADC_INPUT_CURRENT_PIN:
                data->raw.input_current_msb = msb;
                data->raw.input_current_lsb = lsb;
                break;
            case ADC_OUTPUT_CURRENT_PIN:
                data->raw.output_current_msb = msb;
                data->raw.output_current_lsb = lsb;
                break;
            default:
                data->raw.input_voltage_msb = msb;
```

```
260                 data->raw.input_voltage_lsb = lsb;
                    break;
            }

            return OK;
265     }

        int rfcx_read_humid(humid_data_t * data) {
            int ret= 0;

270         //Issue a 'Measurement Request' command
            i2c_start_wait(HUMID_ADDR + I2C_WRITE);
            i2c_stop();

            //Delay ~37ms (datasheet specifies 36.65ms)
275         // NOTE:    This could be avoided if necessary, and we
            //          will just always get the data from the
            //          previous conversion.
            delay_us(HUMID_CONV_TIME);

280         //Fetch humidity + temp data
            ret = i2c_rep_start(HUMID_ADDR + I2C_READ);
            if(ret) {
                i2c_stop();
                usart_send_string("<-- ERROR: Could not repeat start humidity sensor-->\r\n
                    ");
285             return ERROR;
            }

            data->raw.humid_msb = i2c_readAck();
            data->raw.humid_lsb = i2c_readAck();
290         data->raw.temp_msb = i2c_readAck();
            data->raw.temp_lsb = i2c_readNak();
            i2c_stop();

            //Status bits are two msb's of humid_msb
295         data->status = (data->raw.humid_msb & 0xC0) >> 6;

            //Perform conversion
            convert_humid_data(data);

300         return OK;
        }


        void convert_temp_data(temp_data_t * data) {
305         int tmp = 0;
            float result = 0.0;

            int msb = (int)data->raw.msb;
            int lsb = (int)data->raw.lsb;
310
            //Shift 'dem bits around
            tmp = ((msb << 8) | (lsb & ~0x1F)) >> 5;

            //Check sign of data
315         if((msb & 0x80) == 0x80) {
                result = (float)(tmp) * 0.125;
            }
```

```
        else {
            result = -1.0 * (float)(~tmp + 1) * 0.125;
        }

        data->temperature = result;
    }

    void convert_adc_data(adc_data_t * data) {
        data->input_voltage = convert_adc_data_pin(data, ADC_INPUT_VOLTAGE_PIN);
        //...
    }

    float convert_adc_data_pin(adc_data_t * data, int pin) {
        //Perform conversion for individual pin
        //Should be able to use pin nubmer as offset into the raw struct to make this
            easier

        //2-byte conversion register
        int tmp = 0;
        int msb = 0; //Set to correct msb
        int lsb = 0; //Set to correct lsb
        float voltage = 0.0;

        tmp = ((msb << 8) | lsb) >> 4;

        voltage = ((float)tmp / (0x01 << 12)) * 3.3;

        //Rerturn voltage
        return voltage;

        //@TODO RETURN VOLTAGE IF VOLTAGE, ELSE CONVERT TO CURRENT VIA ANOTHER FUNCTION
    }

    void convert_humid_data(humid_data_t * data) {
        uint16_t tmp_humid = 0;
        uint16_t tmp_temp = 0;

        uint16_t humid_msb = (uint16_t)data->raw.humid_msb;
        uint16_t humid_lsb = (uint16_t)data->raw.humid_lsb;
        uint16_t temp_msb = (uint16_t)data->raw.temp_msb;
        uint16_t temp_lsb = (uint16_t)data->raw.temp_lsb;

        //Bit shifting
        tmp_humid = ((humid_msb & ~0xC0) << 8) | humid_lsb;
        tmp_temp = ((temp_msb << 8) | (temp_lsb & ~0x03)) >> 2;

        data->humidity = ((float)tmp_humid / HUMID_COUNTS) * 100.0;
        data->temperature = (((float)tmp_temp / TEMP_COUNTS) * 165.0) - 40.0;

        return;
    }

    void rfcx_humid_status_string(char * str, unsigned char status) {
        switch(status) {
            case HUMID_STATUS_NORMAL:
                sprintf(str, "Normal");
                break;
            case HUMID_STATUS_STALE:
                sprintf(str, "STALE DATA");
```

```
                break;
            case HUMID_STATUS_COMMAND:
                sprintf(str, "Command Mode");
                break;
            case HUMID_STATUS_DIAG:
                sprintf(str, "Diagnostic Condition");
                break;
            default:
                sprintf(str, "UNKNOWN");
                break;
        }
    }
```

F.1.4 rfcx-i2c.h

Listing 5: rfcx-i2c.h

```
    /*************************************************************
    * RFCx Microcontroller Software - I2C
    *
    * Kalee Stutzman    (stutzmak@mail.gvsu.edu)
    * Joe Gibson      (gibsjose@mail.gvsu.edu)
    *
    * 08 July 2015
    *
    *   www.rfcx.org
    *************************************************************/

    #ifndef RFCX_I2C_H
    #define RFCX_I2C_H

    #include <avr/io.h>
    #include <string.h>
    #include "i2cmaster/i2cmaster.h"

    #include "rfcx-globals.h"
    #include "utilities/delay.h"
    #include "utilities/usart.h"

    //Simple Error Representation
    #define OK      0
    #define ERROR   1

    //I2C Addresses
    #define TEMP_ADDR   0x90    //1001CBA0, A = B = C = 0 (pulled up/down in hardware)
    #define ADC_ADDR    0x92    //1001XX1X, ADDR connected to VCC
    #define HUMID_ADDR  0x4E    //01001110, Pre-defined address (0x27 << 1)

    //ADC Pins
    #define ADC_INPUT_VOLTAGE_PIN   0x00
    #define ADC_OUTPUT_VOLTAGE_PIN  0x01
    #define ADC_INPUT_CURRENT_PIN   0x02
    #define ADC_OUTPUT_CURRENT_PIN  0x03

    //Humidity Sensor Status
    #define HUMID_STATUS_NORMAL     0x00    //Normal operation
    #define HUMID_STATUS_STALE      0x01    //Stale data
    #define HUMID_STATUS_COMMAND    0x02    //Command mode
```

```
#define HUMID_STATUS_DIAG       0x03    //Diagnostic condition

//Humidity Sensor Conversion
#define HUMID_COUNTS            0x3FFF  //2^14 - 1 = 16383
#define TEMP_COUNTS             0x3FFF
#define HUMID_CONV_TIME         0x9088  //37000us -> 37ms (HIH6130 datasheet
    specifies 36.65ms conversion time)

//Data structure for LM75BD temp sensor
typedef struct temp_raw_t {
    unsigned char msb;
    unsigned char lsb;
}temp_raw_t;

typedef struct temp_data_t {
    temp_raw_t raw;         //Raw bytes
    float temperature;      //Temperature
}temp_data_t;

//Data structure for ADS1015 external ADC
typedef struct adc_raw_t {
    unsigned char input_voltage_msb;
    unsigned char input_voltage_lsb;
    unsigned char output_voltage_msb;
    unsigned char output_voltage_lsb;
    unsigned char input_current_msb;
    unsigned char input_current_lsb;
    unsigned char output_current_msb;
    unsigned char output_current_lsb;
}adc_raw_t;

typedef struct adc_data_t {
    adc_raw_t raw;          //Raw bytes
    float input_voltage;    //Input voltage
    float output_voltage;   //Output voltage
    float input_current;    //Input current
    float output_current;   //Output current
}adc_data_t;

//Data structures for HIH6130 humidity + temp sensor
typedef struct humid_raw_t {
    unsigned char humid_msb;
    unsigned char humid_lsb;
    unsigned char temp_msb;
    unsigned char temp_lsb;
}humid_raw_t;

typedef struct humid_data_t {
    humid_raw_t raw;        //Raw bytes
    float humidity;         //Relative humidity
    float temperature;      //Temperature
    unsigned char status;   //Status
}humid_data_t;

//Initialization
void rfcx_i2c_init(void);
int rfcx_temp_init(void);
int rfcx_adc_init(void);
int rfcx_humid_init(void);
```

```
100
    //Data Initialization
    void rfcx_temp_data_init(temp_data_t *);
    void rfcx_humid_data_init(humid_data_t *);
    void rfcx_adc_data_init(adc_data_t *);
105
    //Shutdown
    void rfcx_i2c_shutdown(void);
    void rfcx_temp_shutdown(void);
    void rfcx_adc_shutdown(void);
110 void rfcx_humid_shutdown(void);

    //Read
    int rfcx_read_temp(temp_data_t *);
    int rfcx_read_adc_pin(adc_data_t *, int);
115 int rfcx_read_adc(adc_data_t *);
    int rfcx_read_humid(humid_data_t *);

    //Static Conversion Helpers
    void convert_temp_data(temp_data_t *);
120 void convert_adc_data(adc_data_t *);
    float convert_adc_data_pin(adc_data_t *, int);
    void convert_humid_data(humid_data_t *);

    //String Conversion Helpers
125 void rfcx_humid_status_string(char *, unsigned char);

    #endif//RFCX_I2C_H
```

F.1.5 rfcx-battery.c

Listing 6: rfcx-battery.c

```
    /************************************************************
    * RFCx Microcontroller Software - Battery
    *
    * Kalee Stutzman    (stutzmak@mail.gvsu.edu)
5   * Joe Gibson     (gibsjose@mail.gvsu.edu)
    *
    * 08 July 2015
    *
    *   www.rfcx.org
10  ************************************************************/

    #include "rfcx-battery.h"

    //Declare pins as inputs/outputs
15  void rfcx_batteries_init(void) {
        //Inputs
        DDRC &= ~_BV(BAT_1_INPUT_PIN);
        DDRC &= ~_BV(BAT_2_INPUT_PIN);

20      //Outputs
        DDRC |= _BV(BAT_1_OUTPUT_PIN);
        DDRC |= _BV(BAT_2_OUTPUT_PIN);

        //Initialize outputs low
25      PORTC &= ~_BV(BAT_1_OUTPUT_PIN);
```

```
    PORTC &= ~_BV(BAT_2_OUTPUT_PIN);
}

void rfcx_batteries_data_init(batteries_t * batteries) {
    batteries->battery_1.status = BAT_STATUS_ERROR;
    batteries->battery_2.status = BAT_STATUS_ERROR;
}

void rfcx_batteries_status(batteries_t * batteries) {
    batteries->battery_1.status = rfcx_battery_status(BATTERY_1);
    batteries->battery_2.status = rfcx_battery_status(BATTERY_2);
}

unsigned char rfcx_battery_status(unsigned char id) {
    bool first = 0x00;
    bool second = 0x00;

    unsigned char input_pin = 0x00;
    unsigned char output_pin = 0x00;

    //Battery 1
    if(id == BATTERY_1) {
        input_pin = BAT_1_INPUT_PIN;
        output_pin = BAT_1_OUTPUT_PIN;
    }

    //Battery 2
    else if(id == BATTERY_2){
        input_pin = BAT_2_INPUT_PIN;
        output_pin = BAT_2_OUTPUT_PIN;
    }

    //This should never happen...
    else {
        return BAT_STATUS_ERROR;
    }

    //Clear output pin LOW
    PORTC &= ~_BV(output_pin);

    //Read input pin initially
    first = (bool)(PINC & _BV(input_pin));

    //Set output pin HIGH
    PORTC |= _BV(output_pin);

    //Read again
    second = (bool)(PINC & _BV(input_pin));

    //Clear output again
    PORTC &= ~_BV(output_pin);

    //If input pin followed output (low -> high) it is in High Z mode (sleep mode/
       temp fault)
    if((!first) && second) {
        return SLEEP_MODE;
    }

    //Remained high both times (charging)
```

```
        else if(first && second) {
                return CHARGING;
        }

        //Remained low both times (charged)
        else if(!(first && second)) {
            return CHARGE_COMPLETE;
        }

        //Shouldn't ever get here
        return BAT_STATUS_ERROR;
    }

    void rfcx_battery_status_string(char * str, unsigned char status) {
      switch(status) {
         case CHARGING:
            sprintf(str, "Charging");
            break;
         case CHARGE_COMPLETE:
            sprintf(str, "Charge Complete");
            break;
         case SLEEP_MODE:
            sprintf(str, "Sleep Mode");
            break;
         //@TODO How to identify between Sleep Mode and Temp Fault?
         // case TEMPERATURE_FAULT:
         //    sprintf(str, "Temperature Fault");
         //    break;
         case BAT_STATUS_ERROR:
            sprintf(str, "ERROR");
         default:
            sprintf(str, "UNKNOWN");
            break;
      }
    }
```

F.1.6 rfcx-battery.h

Listing 7: rfcx-battery.h

```
    /*************************************************************
    * RFCx Microcontroller Software - Battery
    *
    * Kalee Stutzman    (stutzmak@mail.gvsu.edu)
    * Joe Gibson     (gibsjose@mail.gvsu.edu)
    *
    * 08 July 2015
    *
    *   www.rfcx.org
    *************************************************************/

    #ifndef RFCX_BATTERY_H
    #define RFCX_BATTERY_H

    #include <avr/io.h> //Pin definitions
    #include <stdbool.h>
    #include <stdio.h>
```

```
    //Battery Identifiers
20  #define BATTERY_1           0x01
    #define BATTERY_2           0x02

    //Battery Status Pins
    #define BAT_1_INPUT_PIN     PC0
25  #define BAT_1_OUTPUT_PIN    PC1
    #define BAT_2_INPUT_PIN     PC2
    #define BAT_2_OUTPUT_PIN    PC3

    //Battery Charging Status
30  #define CHARGING            0x01    //HIGH
    #define CHARGE_COMPLETE     0x00    //LOW
    #define SLEEP_MODE          0x02    //HIGH_Z (Input = Output)
    #define TEMPERATURE_FAULT   SLEEP_MODE    //Temperature fault and sleep mode share
        the same value for some reason...
    #define BAT_STATUS_ERROR    0xE0    //Error indicator
35
    typedef struct battery_t {
        unsigned char status;
    }battery_t;

40  typedef struct batteries_t {
        battery_t battery_1;
        battery_t battery_2;
    }batteries_t;

45  void rfcx_batteries_init(void);
    void rfcx_batteries_data_init(batteries_t *);
    void rfcx_batteries_status(batteries_t *);
    unsigned char rfcx_battery_status(unsigned char);
    void rfcx_battery_status_string(char *, unsigned char);
50
    #endif//RFCX_BATTERY_H
```

F.1.7 rfcx-android.c

Listing 8: rfcx-android.c

```
    /************************************************************
    * RFCx Microcontroller Software - Android Serial Comm.
    *
    * Kalee Stutzman    (stutzmak@mail.gvsu.edu)
5   * Joe Gibson     (gibsjose@mail.gvsu.edu)
    *
    * 08 July 2015
    *
    *   www.rfcx.org
10  ************************************************************/

    #include "rfcx-android.h"

    //Package all sensor structure pointers in a single android_serial_t structure
15  void rfcx_android_package(android_serial_t * android, temp_data_t * lm75,
        humid_data_t * hih6130, adc_data_t * ads1015, batteries_t * batteries) {
        android->lm75 = lm75;
        android->hih6130 = hih6130;
        android->ads1015 = ads1015;
```

```
        android->batteries = batteries;
    }

    //Serialize the data to the correct format expected by the Android application:
    //
    //The format is as follows (all comma separated)
    //  ANDROID_BEGIN_FLAG (0x7B)
    //  Input Voltage
    //  Input Current
    //  Output Voltage
    //  Output Current
    //  Temperature
    //  Humidity
    //  Humidity Sensor Status
    //  Battery 1 Status
    //  Battery 2 Status
    //  ANDROID_END_FLAG (0x7E)
    //
    void rfcx_android_serialize(char * buffer, android_serial_t * android) {
        char iv_str[6];
        char ic_str[6];
        char ov_str[6];
        char oc_str[6];
        char temp_str[6];
        char humid_str[6];

        //Convert from float to strings
        dtostrf((double)android->ads1015->input_voltage, 5, 2, iv_str);
        dtostrf((double)android->ads1015->input_current, 5, 2, ic_str);
        dtostrf((double)android->ads1015->output_voltage, 5, 2, ov_str);
        dtostrf((double)android->ads1015->output_current, 5, 2, oc_str);
        dtostrf((double)android->lm75->temperature, 5, 2, temp_str);
        dtostrf((double)android->hih6130->humidity, 5, 2, humid_str);

        //Serialize in buffer
        sprintf(buffer, "%lu,%s,%s,%s,%s,%s,%s,%lu,%lu,%lu,%lu\r\n",
            (unsigned long)ANDROID_BEGIN_FLAG,
            iv_str,
            ic_str,
            ov_str,
            oc_str,
            temp_str,
            humid_str,
            (unsigned long)android->hih6130->status,
            (unsigned long)android->batteries->battery_1.status,
            (unsigned long)android->batteries->battery_2.status,
            (unsigned long)ANDROID_END_FLAG);
    }
```

F.1.8 rfcx-android.h

Listing 9: rfcx-android.h

```
    /*****************************************************
    * RFCx Microcontroller Software - Android Serial Comm.
    *
    * Kalee Stutzman    (stutzmak@mail.gvsu.edu)
    * Joe Gibson     (gibsjose@mail.gvsu.edu)
```

```
 *
 * 08 July 2015
 *
 *   www.rfcx.org
 *************************************************************/

#ifndef RFCX_ANDROID_H
#define RFCX_ANDROID_H

#include <stdio.h>
#include <string.h>

//Include for struct definitions
#include "rfcx-i2c.h"
#include "rfcx-battery.h"

//Packet Framing Bytes
#define ANDROID_BEGIN_FLAG  0x7B  //Marks the beginning of a packet  //0111 1011
#define ANDROID_END_FLAG    0x7E  //Marks the end of a packet        //0111 1110

typedef struct android_serial_t {
    temp_data_t * lm75;
    humid_data_t * hih6130;
    adc_data_t * ads1015;
    batteries_t * batteries;
}android_serial_t;

void rfcx_android_package(android_serial_t *, temp_data_t *, humid_data_t *,
   adc_data_t *, batteries_t *);
void rfcx_android_serialize(char *, android_serial_t *);

#endif//RFCX_ANDROID_H
```

F.1.9 rfcx-globals.h

Listing 10: rfcx-globals.h

```
/*************************************************************
* RFCx Microcontroller Software - Global Definitions
*
* Kalee Stutzman    (stutzmak@mail.gvsu.edu)
* Joe Gibson     (gibsjose@mail.gvsu.edu)
*
* 08 July 2015
*
*   www.rfcx.org
*************************************************************/

#ifndef RFCX_GLOBALS_H
#define RFCX_GLOBALS_H

#ifdef ARDUINO
    #define F_CPU 16000000UL
#else
    #define F_CPU 8000000UL     //Clock Speed (Hz)
#endif

#endif//RFCX_GLOBALS_H
```

F.1.10 delay.c

Listing 11: delay.c

```
#include "delay.h"

int delay_us(unsigned long int microseconds) {
  volatile unsigned cycles = microseconds/64;
  TCCR2A  = 0x00;
  TCCR2B =_BV(CS22) | _BV(CS21) | _BV(CS20); // set timer to use internal clock
      with 1:1024 pre-scale
  if(microseconds < 16321) {
     TCNT2 = 0;
     OCR2A = cycles;
     TIFR2 = _BV(OCF2A); // Set to clear bit 1
     while ((TIFR2 & _BV(OCF2A)) == 0); // NULL
     return(0);
  }
  else {
     TCNT2 = 0;
     OCR2A = 255;
     TIFR2 = _BV(OCF2A); // Set to clear bit 1
     while ((TIFR2 & _BV(OCF2A)) == 0); // NULL
     return(delay_us(microseconds - 16320));
  }
}
```

F.1.11 delay.h

Listing 12: delay.h

```
#ifndef DELAY_H
#define DELAY_H

#include "../rfcx-globals.h"    //F_CPU definition

#include <util/delay.h>
#include <avr/interrupt.h>

int delay_us(unsigned long int);

#endif//DELAY_H
```

F.1.12 usart.c

Listing 13: usart.c

```
/*
 * usart.cpp
 *
 * Created: 07/12/2011 15:17:35
 *  Author: Boomber
 * Modified: 09/15/2013 by R. Bossemeyer to usart.c
 * changed one line to placate GCC
 *
 */
#include "usart.h"
```

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <stdio.h>        // Conversions

void usart_init( unsigned int ubrr)
{
/*Set baud rate */
UBRR0H = (unsigned char)(ubrr>>8);
UBRR0L = (unsigned char)ubrr;
//Enable receiver and transmitter */
UCSR0B = (1<<RXEN0)|(1<<TXEN0);
/* Set frame format: 8data, 2stop bit */
UCSR0C = (1<<USBS0)|(3<<UCSZ00);
}




void usart_send_byte( unsigned char data )
{
/* Wait for empty transmit buffer */
while ( !( UCSR0A & (1<<UDRE0)) )
;
/* Put data into buffer, sends the data */
//UDR0 = char(data);
UDR0 = data;
}

void usart_send_string(const char *str)
{

    while (*str)
      usart_send_byte(*str++);

}

void usart_send_int(unsigned int d )
{
  char str[10];
  sprintf(str,"%u",d);
  usart_send_string(str);

}



unsigned char usart_receive( void )
{
/* Wait for data to be received */
while ( !(UCSR0A & (1<<RXC0)) )
;
/* Get and return received data from buffer */
return UDR0;
}
```

F.1.13 usart.h

Listing 14: usart.h

```
/*
 * usart.h
 *
 * Created: 07/12/2011 15:16:27
 *  Author: Boomber
 */


#ifndef USART_H_
#define USART_H_

void usart_init( unsigned int ubrr);

void usart_send_byte( unsigned char data );
void usart_send_string(const char *str);
void usart_send_int(unsigned int d);

unsigned char usart_receive( void );


#endif /* USART_H_ */
```

F.2 Peter Fleury's I2C Library

Peter Fleury's open source AVR I2C library was used for the I2C/TWI communication. The library and documentation can be found at http://homepage.hispeed.ch/peterfleury/avr-software.html.

F.2.1 twimaster.c

Listing 15: twimaster.c

```
/*************************************************************************
* Title:    I2C master library using hardware TWI interface
* Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
* File:     $Id: twimaster.c,v 1.3 2005/07/02 11:14:21 Peter Exp $
* Software: AVR-GCC 3.4.3 / avr-libc 1.2.3
* Target:   any AVR device with hardware TWI
* Usage:    API compatible with I2C Software Library i2cmaster.h
*************************************************************************/
#include <inttypes.h>
#include <compat/twi.h>

#include <i2cmaster.h>

#include "../utilities/usart.h"

/* define CPU frequency in Mhz here if not defined in Makefile */
#ifndef F_CPU
#define F_CPU 8000000UL
#endif

/* I2C clock in Hz */
#define SCL_CLOCK  100000L

/*************************************************************************
 Initialization of the I2C bus interface. Need to be called only once
```

```
*************************************************************************/
void i2c_init(void)
{
  /* initialize TWI clock: 100 kHz clock, TWPS = 0 => prescaler = 1 */

  TWSR = 0;                         /* no prescaler */
  TWBR = ((F_CPU/SCL_CLOCK)-16)/2;  /* must be > 10 for stable operation */

}/* i2c_init */


/*************************************************************************
  Issues a start condition and sends address and transfer direction.
  return 0 = device accessible, 1= failed to access device
*************************************************************************/
unsigned char i2c_start(unsigned char address)
{
    uint8_t   twst;

  // send START condition
  TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

  // wait until transmission completed
  while(!(TWCR & (1<<TWINT)));

  // check value of TWI Status Register. Mask prescaler bits.
  twst = TW_STATUS & 0xF8;
  if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;

  // send device address
  TWDR = address;
  TWCR = (1<<TWINT) | (1<<TWEN);

  // wail until transmission completed and ACK/NACK has been received
  while(!(TWCR & (1<<TWINT)));

  // check value of TWI Status Register. Mask prescaler bits.
  twst = TW_STATUS & 0xF8;
  if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;

  return 0;

}/* i2c_start */


/*************************************************************************
 Issues a start condition and sends address and transfer direction.
 If device is busy, use ack polling to wait until device is ready

 Input:   address and transfer direction of I2C device
*************************************************************************/
void i2c_start_wait(unsigned char address)
{
    uint8_t   twst;


    while ( 1 )
    {
      // send START condition
```

```
85            TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

          // wait until transmission completed
          while(!(TWCR & (1<<TWINT)));

90        // check value of TWI Status Register. Mask prescaler bits.
          twst = TW_STATUS & 0xF8;
          if ( (twst != TW_START) && (twst != TW_REP_START)) continue;

          // send device address
95        TWDR = address;
          TWCR = (1<<TWINT) | (1<<TWEN);

          // wail until transmission completed
          while(!(TWCR & (1<<TWINT)));
100
          // check value of TWI Status Register. Mask prescaler bits.
          twst = TW_STATUS & 0xF8;
          if ( (twst == TW_MT_SLA_NACK )||(twst ==TW_MR_DATA_NACK) )
          {
105           /* device busy, send stop condition to terminate write operation */
              TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

              // wait until stop condition is executed and bus released
              while(TWCR & (1<<TWSTO));
110
              continue;
          }
          //if( twst != TW_MT_SLA_ACK) return 1;
          break;
115       }

      }/* i2c_start_wait */


120   /*************************************************************************
       Issues a repeated start condition and sends address and transfer direction

       Input:   address and transfer direction of I2C device

125    Return:  0 device accessible
                1 failed to access device
      *************************************************************************/
      unsigned char i2c_rep_start(unsigned char address)
      {
130       return i2c_start( address );

      }/* i2c_rep_start */


135   /*************************************************************************
       Terminates the data transfer and releases the I2C bus
      *************************************************************************/
      void i2c_stop(void)
      {
140       /* send stop condition */
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

        // wait until stop condition is executed and bus released
```

```
  while(TWCR & (1<<TWSTO));

}/* i2c_stop */


/*************************************************************************
  Send one byte to I2C device

  Input:    byte to be transfered
  Return:   0 write successful
            1 write failed
*************************************************************************/
unsigned char i2c_write( unsigned char data )
{
    uint8_t   twst;

  // send data to the previously addressed device
  TWDR = data;
  TWCR = (1<<TWINT) | (1<<TWEN);

  // wait until transmission completed
  while(!(TWCR & (1<<TWINT)));

  // check value of TWI Status Register. Mask prescaler bits
  twst = TW_STATUS & 0xF8;
  if( twst != TW_MT_DATA_ACK) return 1;
  return 0;

}/* i2c_write */


/*************************************************************************
 Read one byte from the I2C device, request more data from device

 Return:  byte read from I2C device
*************************************************************************/
unsigned char i2c_readAck(void)
{
  TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
  while(!(TWCR & (1<<TWINT)));

    return TWDR;

}/* i2c_readAck */


/*************************************************************************
 Read one byte from the I2C device, read is followed by a stop condition

 Return:  byte read from I2C device
*************************************************************************/
unsigned char i2c_readNak(void)
{
  TWCR = (1<<TWINT) | (1<<TWEN);
  while(!(TWCR & (1<<TWINT)));

    return TWDR;

}/* i2c_readNak */
```

F.2.2 i2cmaster.h

Listing 16: i2cmaster.h

```
#ifndef _I2CMASTER_H
#define _I2CMASTER_H   1
/************************************************************************
* Title:    C include file for the I2C master interface
*           (i2cmaster.S or twimaster.c)
* Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
* File:     $Id: i2cmaster.h,v 1.10 2005/03/06 22:39:57 Peter Exp $
* Software: AVR-GCC 3.4.3 / avr-libc 1.2.3
* Target:   any AVR device
* Usage:    see Doxygen manual
************************************************************************/

#ifdef DOXYGEN
/**
 @defgroup pfleury_ic2master I2C Master library
 @code #include <i2cmaster.h> @endcode

 @brief I2C (TWI) Master Software Library

 Basic routines for communicating with I2C slave devices. This single master
 implementation is limited to one bus master on the I2C bus.

 This I2c library is implemented as a compact assembler software implementation of
    the I2C protocol
 which runs on any AVR (i2cmaster.S) and as a TWI hardware interface for all AVR
    with built-in TWI hardware (twimaster.c).
 Since the API for these two implementations is exactly the same, an application
    can be linked either against the
 software I2C implementation or the hardware I2C implementation.

 Use 4.7k pull-up resistor on the SDA and SCL pin.

 Adapt the SCL and SDA port and pin definitions and eventually the delay routine in
     the module
 i2cmaster.S to your target when using the software I2C implementation !

 Adjust the  CPU clock frequence F_CPU in twimaster.c or in the Makfile when using
    the TWI hardware implementaion.

 @note
    The module i2cmaster.S is based on the Atmel Application Note AVR300, corrected
        and adapted
    to GNU assembler and AVR-GCC C call interface.
    Replaced the incorrect quarter period delays found in AVR300 with
    half period delays.

 @author Peter Fleury pfleury@gmx.ch  http://jump.to/fleury

 @par API Usage Example
  The following code shows typical usage of this library, see example
     test_i2cmaster.c

```

```
 @code

 #include <i2cmaster.h>


 #define Dev24C02  0xA2      // device address of EEPROM 24C02, see datasheet

 int main(void)
 {
     unsigned char ret;

     i2c_init();                             // initialize I2C library

     // write 0x75 to EEPROM address 5 (Byte Write)
     i2c_start_wait(Dev24C02+I2C_WRITE);     // set device address and write mode
     i2c_write(0x05);                        // write address = 5
     i2c_write(0x75);                        // write value 0x75 to EEPROM
     i2c_stop();                             // set stop conditon = release bus


     // read previously written value back from EEPROM address 5
     i2c_start_wait(Dev24C02+I2C_WRITE);     // set device address and write mode

     i2c_write(0x05);                        // write address = 5
     i2c_rep_start(Dev24C02+I2C_READ);       // set device address and read mode

     ret = i2c_readNak();                    // read one byte from EEPROM
     i2c_stop();

     for(;;);
 }
 @endcode

*/
#endif /* DOXYGEN */

/**@{*/

#if (__GNUC__ * 100 + __GNUC_MINOR__) < 304
#error "This library requires AVR-GCC 3.4 or later, update to newer AVR-GCC
    compiler !"
#endif

#include <avr/io.h>

/** defines the data direction (reading from I2C device) in i2c_start(),
    i2c_rep_start() */
#define I2C_READ    1

/** defines the data direction (writing to I2C device) in i2c_start(),i2c_rep_start
    () */
#define I2C_WRITE   0


/**
 @brief initialize the I2C master interace. Need to be called only once
 @param  void
 @return none
 */
```

```
    extern void i2c_init(void);


105 /**
     @brief Terminates the data transfer and releases the I2C bus
     @param void
     @return none
     */
110 extern void i2c_stop(void);


    /**
     @brief Issues a start condition and sends address and transfer direction
115
     @param    addr address and transfer direction of I2C device
     @retval   0   device accessible
     @retval   1   failed to access device
     */
120 extern unsigned char i2c_start(unsigned char addr);


    /**
     @brief Issues a repeated start condition and sends address and transfer direction
125
     @param   addr address and transfer direction of I2C device
     @retval  0 device accessible
     @retval  1 failed to access device
     */
130 extern unsigned char i2c_rep_start(unsigned char addr);


    /**
     @brief Issues a start condition and sends address and transfer direction
135
     If device is busy, use ack polling to wait until device ready
     @param    addr address and transfer direction of I2C device
     @return   none
     */
140 extern void i2c_start_wait(unsigned char addr);


    /**
     @brief Send one byte to I2C device
145  @param    data  byte to be transfered
     @retval   0 write successful
     @retval   1 write failed
     */
    extern unsigned char i2c_write(unsigned char data);
150


    /**
     @brief    read one byte from the I2C device, request more data from device
     @return   byte read from I2C device
155  */
    extern unsigned char i2c_readAck(void);

    /**
     @brief    read one byte from the I2C device, read is followed by a stop condition
160  @return   byte read from I2C device
```

```
 */
extern unsigned char i2c_readNak(void);

/**
 @brief    read one byte from the I2C device

 Implemented as a macro, which calls either i2c_readAck or i2c_readNak

 @param    ack 1 send ack, request more data from device<br>
               0 send nak, read is followed by a stop condition
 @return   byte read from I2C device
 */
extern unsigned char i2c_read(unsigned char ack);
#define i2c_read(ack)  (ack) ? i2c_readAck() : i2c_readNak();


/**@}*/
#endif
```

F.3 Android Source Code

Listing 17: DeviceListActivity.java

```
/**
 * Copyright 2015 RFCx <www.rfcx.org>
 * Copyright 2015 Grand Valley State University <www.gvsu.edu>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
 * USA.
 *
 * Project Home Page: www.github.com/gibsjose/RFCxSentinel
 */

package org.rfcx.sentinel;

import android.app.Activity;
import android.content.Context;
import android.hardware.usb.UsbDevice;
import android.hardware.usb.UsbManager;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.os.SystemClock;
import android.util.Log;
import android.view.LayoutInflater;
```

```
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.TwoLineListItem;

import com.hoho.android.usbserial.driver.UsbSerialDriver;
import com.hoho.android.usbserial.driver.UsbSerialPort;
import com.hoho.android.usbserial.driver.UsbSerialProber;
import com.hoho.android.usbserial.util.HexDump;

import java.util.ArrayList;
import java.util.List;

/**
 * Shows a {@link ListView} of available USB devices.
 *
 * @author Joe Gibson (gibsjose@mail.gvsu.edu)
 * @author Mike Wakerly (opensource@hoho.com)
 */
public class DeviceListActivity extends Activity {

    private final String TAG = DeviceListActivity.class.getSimpleName();

    private UsbManager mUsbManager;
    private ListView mListView;
    private TextView mProgressBarTitle;
    private ProgressBar mProgressBar;

    private static final int MESSAGE_REFRESH = 101;
    private static final long REFRESH_TIMEOUT_MILLIS = 5000;

    private final Handler mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case MESSAGE_REFRESH:
                    refreshDeviceList();
                    mHandler.sendEmptyMessageDelayed(MESSAGE_REFRESH,
                        REFRESH_TIMEOUT_MILLIS);
                    break;
                default:
                    super.handleMessage(msg);
                    break;
            }
        }

    };

    private List<UsbSerialPort> mEntries = new ArrayList<UsbSerialPort>();
    private ArrayAdapter<UsbSerialPort> mAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

```
        mUsbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
        mListView = (ListView) findViewById(R.id.deviceList);
        mProgressBar = (ProgressBar) findViewById(R.id.progressBar);
        mProgressBarTitle = (TextView) findViewById(R.id.progressBarTitle);

        mAdapter = new ArrayAdapter<UsbSerialPort>(this,
                android.R.layout.simple_expandable_list_item_2, mEntries) {
            @Override
            public View getView(int position, View convertView, ViewGroup parent) {
                final TwoLineListItem row;
                if (convertView == null){
                    final LayoutInflater inflater =
                            (LayoutInflater) getSystemService(Context.
                                LAYOUT_INFLATER_SERVICE);
                    row = (TwoLineListItem) inflater.inflate(android.R.layout.
                        simple_list_item_2, null);
                } else {
                    row = (TwoLineListItem) convertView;
                }

                final UsbSerialPort port = mEntries.get(position);
                final UsbSerialDriver driver = port.getDriver();
                final UsbDevice device = driver.getDevice();

                final String title = String.format("Vendor %s Product %s",
                        HexDump.toHexString((short) device.getVendorId()),
                        HexDump.toHexString((short) device.getProductId()));
                row.getText1().setText(title);

                final String subtitle = driver.getClass().getSimpleName();
                row.getText2().setText(subtitle);

                return row;
            }

        };
        mListView.setAdapter(mAdapter);

        mListView.setOnItemClickListener(new ListView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
                 long id) {
                Log.d(TAG, "Pressed item " + position);
                if (position >= mEntries.size()) {
                    Log.w(TAG, "Illegal position.");
                    return;
                }

                final UsbSerialPort port = mEntries.get(position);
                showConsoleActivity(port);
            }
        });
    }

    @Override
    protected void onResume() {
        super.onResume();
        mHandler.sendEmptyMessage(MESSAGE_REFRESH);
```

```
150         }

            @Override
            protected void onPause() {
                super.onPause();
155             mHandler.removeMessages(MESSAGE_REFRESH);
            }

            private void refreshDeviceList() {
                showProgressBar();
160
                new AsyncTask<Void, Void, List<UsbSerialPort>>() {
                    @Override
                    protected List<UsbSerialPort> doInBackground(Void... params) {
                        Log.d(TAG, "Refreshing device list ...");
165                     SystemClock.sleep(1000);

                        final List<UsbSerialDriver> drivers =
                                UsbSerialProber.getDefaultProber().findAllDrivers(
                                    mUsbManager);

170                     final List<UsbSerialPort> result = new ArrayList<UsbSerialPort>();
                        for (final UsbSerialDriver driver : drivers) {
                            final List<UsbSerialPort> ports = driver.getPorts();
                            Log.d(TAG, String.format("+ %s: %s port%s",
                                    driver, Integer.valueOf(ports.size()), ports.size() ==
                                        1 ? "" : "s"));
175                         result.addAll(ports);
                        }

                        return result;
                    }
180
                    @Override
                    protected void onPostExecute(List<UsbSerialPort> result) {
                        mEntries.clear();
                        mEntries.addAll(result);
185                     mAdapter.notifyDataSetChanged();
                        mProgressBarTitle.setText(
                                String.format("%s device(s) found",Integer.valueOf(mEntries
                                    .size())));
                        hideProgressBar();
                        Log.d(TAG, "Done refreshing, " + mEntries.size() + " entries found.
                            ");
190                 }

                }.execute((Void) null);
            }

195         private void showProgressBar() {
                mProgressBar.setVisibility(View.VISIBLE);
                mProgressBarTitle.setText(R.string.refreshing);
            }

200         private void hideProgressBar() {
                mProgressBar.setVisibility(View.INVISIBLE);
            }

            private void showConsoleActivity(UsbSerialPort port) {
```

```
            //Uncomment this to use the Serial Activity for Raw Serial Data
            //SerialConsoleActivity.show(this, port);
            SentinelActivity.show(this, port);
        }

    }
```

Listing 18: SentinelActivity.java

```
/**
 * Copyright 2015 RFCx <www.rfcx.org>
 * Copyright 2015 Grand Valley State University <www.gvsu.edu>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
 * USA.
 *
 * Project Home Page: www.github.com/gibsjose/RFCxSentinel
 */

package org.rfcx.sentinel;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.hardware.usb.UsbDeviceConnection;
import android.hardware.usb.UsbManager;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

import com.hoho.android.usbserial.driver.UsbSerialPort;
import com.hoho.android.usbserial.util.SerialInputOutputManager;

import java.io.IOException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

/**
 * Shows a {@link android.widget.TextView} of relevant data (power, temperature,
    humidity)
 *
 * @author Joe Gibson (gibsjose@mail.gvsu.edu)
 */
public class SentinelActivity extends Activity {
    private final String TAG = SentinelActivity.class.getSimpleName();
    //private final long BEGIN_FLAG = 0xBB;
```

```
50          //private final long END_FLAG = 0xEE;

            private static UsbSerialPort sPort = null;

            private TextView mStatus;
55          private TextView mInputCurrentValue;
            private TextView mInputVoltageValue;
            private TextView mOutputCurrentValue;
            private TextView mOutputVoltageValue;
            private TextView mTemperatureValue;
60          private TextView mHumidityValue;

            private final ExecutorService mExecutor = Executors.newSingleThreadExecutor();

            private SerialInputOutputManager mSerialIoManager;
65
            private final SerialInputOutputManager.Listener mListener = new
                SerialInputOutputManager.Listener() {

                @Override
                public void onRunError(Exception e) {
70                  Log.d(TAG, "Runner stopped.");
                }

                @Override
                public void onNewData(final byte[] data) {
75                  SentinelActivity.this.runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            SentinelActivity.this.updateReceivedData(data);
                        }
80                  });
                }
            };

            @Override
85          protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.sentinel);
                mStatus = (TextView) findViewById(R.id.status);
                mInputCurrentValue = (TextView)findViewById(R.id.inputCurrentValue);
90              mInputVoltageValue = (TextView)findViewById(R.id.inputVoltageValue);
                mOutputCurrentValue = (TextView)findViewById(R.id.outputCurrentValue);
                mOutputVoltageValue = (TextView)findViewById(R.id.outputVoltageValue);
                mTemperatureValue = (TextView)findViewById(R.id.temperatureValue);
                mHumidityValue = (TextView)findViewById(R.id.humidityValue);
95          }

            @Override
            protected void onPause() {
                super.onPause();
100             stopIoManager();
                if (sPort != null) {
                    try {
                        sPort.close();
                    } catch (IOException e) {
105                     // Ignore.
                    }
                    sPort = null;
```

```
            }
            finish();
        }

        @Override
        protected void onResume() {
            super.onResume();
            Log.d(TAG, "Resumed, port=" + sPort);
            if (sPort == null) {
                mStatus.setText("No serial device.");
                mStatus.setTextColor(0xFFFF5F69);
            } else {
                final UsbManager usbManager = (UsbManager) getSystemService(Context.
                    USB_SERVICE);

                UsbDeviceConnection connection = usbManager.openDevice(sPort.getDriver
                    ().getDevice());
                if (connection == null) {
                    mStatus.setText("Opening device failed");
                    mStatus.setTextColor(0xFFFF5F69);
                    return;
                }

                try {
                    sPort.open(connection);

                    //115200 Baud rate
                    //8 data bits
                    //1 stop bit
                    //No parity bit
                    sPort.setParameters(115200, 8, UsbSerialPort.STOPBITS_1,
                        UsbSerialPort.PARITY_NONE);
                } catch (IOException e) {
                    Log.e(TAG, "Error setting up device: " + e.getMessage(), e);
                    mStatus.setText("Error opening device: " + e.getMessage());
                    mStatus.setTextColor(0xFFFF5F69);
                    try {
                        sPort.close();
                    } catch (IOException e2) {
                        // Ignore.
                    }
                    sPort = null;
                    return;
                }
                mStatus.setText("Device: " + sPort.getClass().getSimpleName());
                mStatus.setTextColor(0xFFFFFFFF);
            }
            onDeviceStateChange();
        }

        private void stopIoManager() {
            if (mSerialIoManager != null) {
                Log.i(TAG, "Stopping io manager ..");
                mSerialIoManager.stop();
                mSerialIoManager = null;
            }
        }

        private void startIoManager() {
```

```
            if (sPort != null) {
                Log.i(TAG, "Starting io manager ..");
                mSerialIoManager = new SerialInputOutputManager(sPort, mListener);
                mExecutor.submit(mSerialIoManager);
            }
        }

        private void onDeviceStateChange() {
            stopIoManager();
            startIoManager();
        }

        private void updateReceivedData(byte[] data) {
            final String message = new String(data);

            /**
             *  values[0] = Input Current (mA)
             *  values[1] = Input Voltage (V)
             *  values[2] = Output Current (mA)
             *  values[3] = Output Voltage (V)
             *  values[4] = Temperature (C)
             *  values[5] = Humidity (RH)
             */
            String[] values = message.split(",");

            //Ensure the length is correct (8), and use framing
            //if((values.length >= 8) && (Long.parseLong(values[0]) == BEGIN_FLAG) && (
                Long.parseLong(values[7]) == END_FLAG)) {
            if(values.length >= 8) {
                    mStatus.setTextColor(Color.GREEN);
                    mInputCurrentValue.setText(values[1] + " mA");
                    mInputVoltageValue.setText(values[2] + " V");
                    mOutputCurrentValue.setText(values[3] + " mA");
                    mOutputVoltageValue.setText(values[4] + "V");
                    mTemperatureValue.setText(values[5] + " C");
                    mHumidityValue.setText(values[6] + " % RH");
            } else {
                mStatus.setTextColor(Color.RED);
                //mStatus.setText(message);
                //mInputCurrentValue.setText("Length: " + values.length);
            }
        }

        /**
         * Starts the activity, using the supplied driver instance.
         *
         * @param context the Context
         * @param port the USB Serial port
         */
        static void show(Context context, UsbSerialPort port) {
            sPort = port;
            final Intent intent = new Intent(context, SentinelActivity.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP | Intent.
                FLAG_ACTIVITY_NO_HISTORY);
            context.startActivity(intent);
        }
    }
```