

6-13-2019

WASP - Wireless Analog Sensor Platform

Tyler Hack

Cole Hunter

Daniel Webber

Follow this and additional works at: https://scholarcommons.scu.edu/elec_senior

 Part of the [Electrical and Computer Engineering Commons](#)

SANTA CLARA UNIVERSITY

Department of Electrical Engineering

I HEREBY RECOMMEND THAT THE THESIS PREPARED
UNDER MY SUPERVISION BY

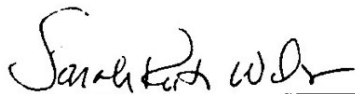
Tyler Hack, Cole Hunter, and Daniel Webber

ENTITLED

WASP - Wireless Analog Sensor Platform

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

**BACHELOR OF SCIENCE
IN
ELECTRICAL ENGINEERING**



Thesis Advisor

6/11/19

date



Thesis Advisor

6/11/19

date



Department Chair

6/11/19

date

WASP - Wireless Analog Sensor Platform

By

Tyler Hack
Cole Hunter
Daniel Webber

Submitted in partial fulfillment of the requirements
for the degrees of
Engineering Bachelor of Science in Electrical Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 13, 2019

WASP - Wireless Analog Sensor Platform

Tyler Hack
Cole Hunter
Daniel Webber

Department of Electrical Engineering
Santa Clara University
June 13, 2018

ABSTRACT

WASP's goal is to augment and eventually replace the bulky, costly, and complex data acquisition systems used for vibrational reliability tests on satellites. As a mechanism to guarantee that a spacecraft is mechanically durable and strong enough to withstand the acceleration forces experienced on the vessel during launch, companies conduct vibrational experiments on their spacecrafts by subjecting them to high G-force events. Using wired accelerometers connected to obstructive cables, the mounting process and test setups required to perform such experiments are expensive, laborious, and have the potential to generate measurement inaccuracies. We developed a low-cost, battery-powered module, designed for engineers, to replace the current sensors and data acquisition systems with a wireless solution. This will enable precise testing of conditions on a smaller time frame and at a lower cost and help eliminate the disadvantages of a wired system. A custom circuit board has been fabricated containing the critical measurement and processing components required to realize this objective, as well as a complete software solution to facilitate data transmission to a wireless router over WiFi.

ABSTRACT	2
List of Figures	5
List of Tables	6
Introduction	7
Problem Statement	7
Benefit	9
Existing Solutions	9
Wired	9
Wireless	9
Proposed Solution	10
Requirements	10
Background	10
System Specifications	11
Design Rationale	13
Accelerometer	13
Wireless System-On-Chip	15
Battery Technology	16
Power & Power Management	17
Voltage Regulators	17
Battery Supervisor	18
Analog to Digital Conversion	19
PCB Design Software	20
Firmware and Software	22
Overview	22
System Multi-threading	23
Drivers	25
Wireless Module Design	26
Architectural Block Diagrams	26
CYW43907	28
ADXL1004	29
AD4008	32
Power Electronics	33
Buck-Boost	33

Precision Voltage Reference	34
Linear Regulators	34
Gas Gauge	35
In Circuit Power Measurement	36
PCB Design	37
Schematic Design	37
PCB Layout	37
Assembly	40
Testing and Results	42
Power	42
Sensor Performance	45
Software	53
Overview	53
Drivers	54
Additional Accelerometer Functions	57
Host Side Networking	58
Server Side Networking	61
WASP SDP	66
OTA Subsystem	67
Final Cost	67
Social Analysis	68
Ethics	68
Ethical Justification	68
Engineering Virtues	69
Safety	70
Risk	70
Sustainability	71
Social	71
Environmental	71
Economic	72
Future Work	73
Vibration Comparison Testing	73
Condensed Form Factor	73
Multi-Axis Accelerometer	74
Large Node Testing And Evaluation	74
Software Optimization and Improved Feature Set	74

Lessons Learned	75
Timeline & Design Challenges	75
Hardware and Software Integration	76
Acknowledgments	76
References	76
Appendix A: WASP Module Schematic	79
Appendix B: Software	86
Appendix C: Project Timeline	86

List of Figures

Figure 1: A typical mounting setup for spacecraft vibrational testing.	8
Figure 2: A tangled array of sensor cables mounted on a spacecraft.	8
Figure 3: Functional level block diagram of our proposed solution.	10
Figure 4: WASP Server work division implementation.	25
Figure 5: High level block diagram separated by major subsystem.	27
Figure 6: Low level block diagram showing all major components and their connections.	27
Figure 7: An annotated image of the Quicksilver module.	29
Figure 8: A physical and circuit model representation of a MEMS accelerometer.	30
Figure 9: RC filter Bode plot and cursor information from LT-Spice Simulation.	31
Figure 10: A top view floorplan of our PCB.	38
Figure 11: 2D current density simulation results for PP3V3 and PP3V7 voltage rails.	39
Figure 12: 2D voltage drop simulation results for PP3V3, PP3V7 and PP3V3_ADXL voltage rails.	40

Figure 13: Draftsman drawings provided to the vendor for assembly.	41
Figure 14: Image of the assembled prototype.	42
Figure 15: Histogram showing the distribution of the 3.7 voltage rail under light load.	43
Figure 16: A distribution of PP3V3_ADXL under hibernate conditions.	46
Figure 17: A distribution of PP3V3_ADXL under data sampling conditions.	47
Figure 18: A graph from the ADXL1004 datasheet showing the normalized offset vs temperature.	48
Figure 19: Histogram of calibrated idle (0g) testing of the system.	49
Figure 20: Histogram of uncalibrated idle (0g) testing of the system.	50
Figure 21: Histogram of calibrated static +1g testing of the system.	51
Figure 22: Histogram of calibrated static -1g testing of the system.	52
Figure 23: Time domain plot of calibrated +1g/-1g testing of the system.	53
Figure 24: WASP board driver overview.	54
Figure 25: WASP data packet.	59
Figure 26: WASP server CLI.	62
Figure 27: Output of the print thread for 1 device.	63

List of Tables

Table 1: Technical specifications governing WASP system design.	12
Table 2: Accelerometer Specifications.	13
Table 3: Accelerometer Decision Matrix.	14
Table 4: Comparison of various rechargeable battery chemistries.	17
Table 5: Battery Supervisor Decision Matrix.	19

Table 6: ADC Decision Matrix.	20
Table 7: PCB Software Requirements.	21
Table 8: PCB Software Decision Matrix.	22
Table 9: System Power Measurements.	44
Table 10: Precision Reference Testing Results.	46
Table 11: Project Costs Paid By Grants.	68

Introduction

Problem Statement

The application and area of focus for this project is the space industry. Space and aerospace companies send satellites and other spacecrafts into space for any number of purposes, whether they be academic or research-based, for military or defense purposes, or simply for exploratory missions. However, successfully launching a spacecraft is a complex procedure that can take years to plan and hundreds of millions of dollars to fund. As a result, it is the company or organization's duty to perform extensive tests on a spacecraft prior to launch to ensure the success and durability of the vessel. While there are numerous ways a spacecraft can fail during a launch, the target focus of this paper is vibrational analysis and testing of spacefaring vehicles. These tests are performed to analyze the mechanical durability of the spacecraft against the extremely strong vibrational forces experienced during launch. Companies simulate these experiments by conducting non-destructive vibrational experiments on their vehicles to ensure that they will survive the high G-forces required to leave the atmosphere.[1] Figure 1 displays a typical test setup featuring a spacecraft placed on a shake table and mounted with sensors that connect, via long cables, back to data acquisition modules (DAQs) in a control room.

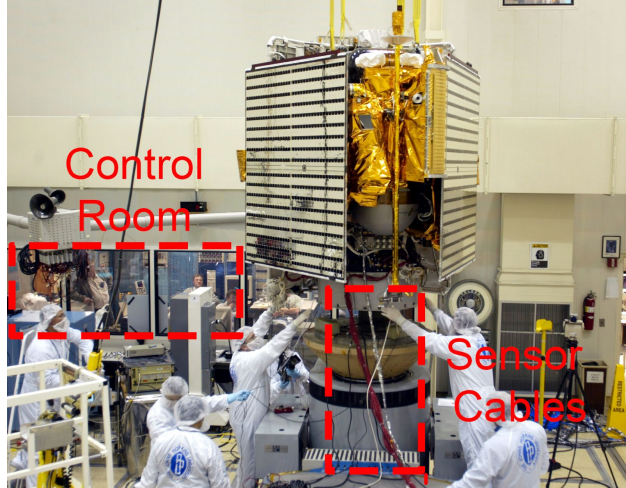


Figure 1: A typical mounting setup for spacecraft vibrational testing.

These tests are currently conducted with upwards of 200 wired accelerometers simultaneously connected to very expensive high performance DAQs.[2] It is not uncommon for a DAQ to cost upwards of one hundred thousand dollars while any individual sensor can cost up to one thousand dollars. [3] The result is a disarray of wired connections that make installing and debugging these sensors an excessively laborious and time-consuming process. However, aside from the difficulties in physically setting up the vibrational tests, the cables themselves contribute to the load of the system under test and by adding extra weight, can skew the vibrational measurement results. [4]

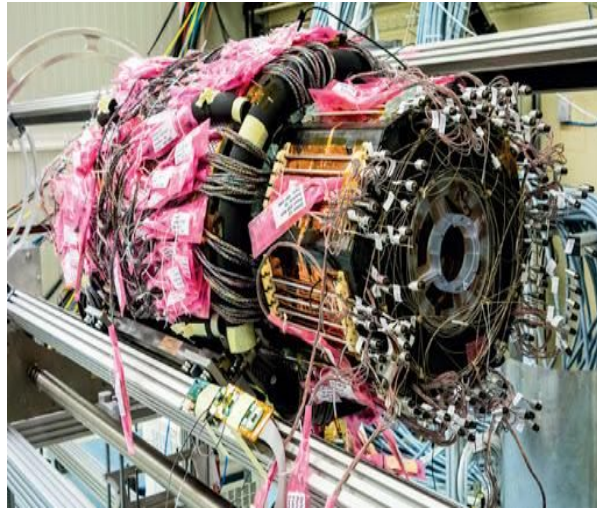


Figure 2: A tangled array of sensor cables mounted on a spacecraft.

We propose a wireless solution featuring a low-cost battery powered module to augment and eventually replace the wired testing and data acquisition systems. By reinventing the current system around an existing high resolution sensor, we can effectively eliminate the pitfalls of a wired system while maintaining the current standard of testing.

Benefit

Our device will greatly benefit any company interested in doing vibrational tests as well as academic or smaller scale space projects that cannot afford to spend hundreds of thousands of dollars on test equipment. By making the modules wireless, we are able to eliminate all of the wires in the system, removing any possible effects due to weight. The devices will also cost as little as \$100, saving valuable resources that can be allocated elsewhere. Finally, the devices will be small enough to be easily mounted in hard to access locations, and given that they can be managed remotely, the test operators will be able to easily install our system and run tests at a later date while other preparations are made.

Existing Solutions

Wired

Traditionally, the approach for acquiring data is to attach all of the wired sensors to the spacecraft. Each sensor has a short amount of wire with a connector attached at the end. From there, longer cables can be run from the sensor to the data acquisition system (DAQ). The cables have to be carefully mounted to the spacecraft for two primary reasons: to not damage the spacecraft during the test, and to not significantly change the load properties of the device under test (DUT). The second issue stands out to our team because if one would want to add more sensors to better study the system, the added weight will alter the load properties of the system, which is in direct opposition to the goal. This is because the more sensors that are added, the more likely the results are not representative of the actual system. [4]

As mentioned earlier all of the sensors will be connected to DAQs. Typical industrial DAQs are rack mounted and allow for several connections at once. Each rack can easily cost ten thousand to one hundred thousand dollars. [3] This cost prohibitor makes it difficult to add channels or reconfigure the technology. Typically larger control rooms are built and designed to house the DAQs from which cables protrude to connect the sensors to the modules. Given that each cable will perform slightly differently and have a unique transfer function, an individual “profile” needs to be generated so that these differences can be calibrated out. Although the classic wired solution is robust and tested, it is cost-prohibitive, and lacks versatility.

Wireless

In our preliminary research, we found that no wireless solution exists for vibrational tests in commercial applications. We did however find researchers at Georgia Tech who developed a platform with similar attributes. Their papers commonly cite applications such as civil engineering monitoring projects with no mention of space systems applications. Their system is called *Martlet* and runs on the IEEE 802.15.4 standard, with a typical bit rate of 100kb/s. The system footprint is 60x60mm and uses two large AA alkaline batteries. The hardware is run with a TI SOC and a TI baseband radio, which take up a large portion of the board.[5] Upon further

analysis of their hardware architecture, the SOC could likely be substituted for a smaller SOC with less features. This architecture allows for many sensors to be integrated and contains some critical signal conditioning components.

Although this system is fit to measure acceleration, its scalability is limited by the communication protocol. Research shows that even 5-15 devices transmitting simultaneously (as a satellite test would expect) increase the bit error rate multiple magnitudes.[6] This limiting ability is most likely why the board contains a SD card on the device and data is streamed post test. Although this system is similar to ours, it is not suited for the scale and expected data rates of satellite testing.

Proposed Solution

In order to improve upon the current method of conducting vibrational experiments on spacefaring vehicles, we propose a wireless alternative. Our objective is to design a custom circuit board that, when mounted on a spacecraft, could measure the acceleration experienced during the test, digitize the analog signal, and then transmit the data wirelessly to a base station. Figure 3 below outlines the high level functionality of our proposed system. In order to have this solution mitigate the pitfalls of the bulky, heavy, and expensive wired measurement systems, this custom circuit board will need to possess a small footprint and be both low power and low cost. The base station receiving the data will not be designed during this project - rather, it will consist of an off-the-shelf piece of hardware, such as a WiFi router, that can be programmed appropriately to communicate with several of our custom boards. The base station and the custom board will run software designed by us that will control and facilitate the data acquisition. The whole solution will be robust and easy to use.

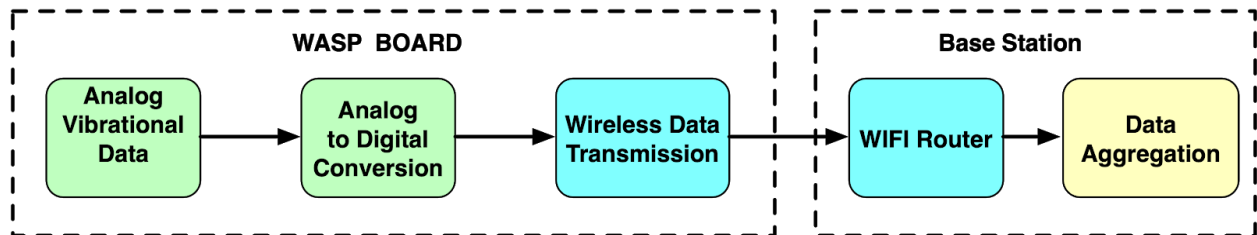


Figure 3: Functional level block diagram of our proposed solution .

Requirements

Background

To have a clear idea of what our system requires, one needs to better understand the tests that need to be run and what tests our system targets. In our research we identified three main types

of vibration tests that are run on spacecrafts: random vibration, sinusoidal vibration, and shock. [2] Random vibration is typically a test where the spacecraft is subjected to quasi-random vibrations within some specification limit. This is done to ensure complete test coverage of the spacecraft against unanticipated vibrational events. Sinusoidal vibration is when the spacecraft is acceleration a constant frequency with varying amplitudes. Test designers can then sweep frequency to help identify natural resonances of the spacecraft. Shock testing is when the spacecraft is subjected to high impulse vibrations, and these are typically done to simulate pyrotechnics exploding, which can be used to separate stages of the spacecraft. The vibration is typically at much larger magnitude than the two previous tests. [2] Working with our collaborators at SSL we determined that although we could pursue a prototype to accommodate all three tests, just focusing on random vibration and sinusoidal vibration makes sense. This is because not only would we have to design a proof of concept but to accommodate shock testing we would have to “harden” the solution. This means we would have to design the system to not destroy itself under the extreme acceleration conditions.

The setup for both the sinusoidal and random vibration tests are exactly alike. The spacecraft or device under test (DUT) is attached to a large shake table. From there accelerometers are attached. This process is the most laborious and time consuming step in the process. Our contacts at SSL have informed us depending on sensor placement, partial disassembly of the satellite may occur to facilitate placement of the sensors in the exact location needed to take data. The process of mounting can take upwards of a week of attaching the devices before the test is actually run. This small detail is critical to our design since our device is battery powered and needs to be powered on at the time of sensor attachment since it may become inaccessible due to reassembly. After sensor attachment, all the sensors are then wired into the control room which houses the data acquisition modules. Each sensor has to be manually verified at this point. Verification is critical because the engineers want to ensure that all sensors are functioning properly before the test is conducted. After verification the actual testing process begins. These tests can run up to 30-40 minutes, where the sensors are typically being sampled continuously during the entirety of the test. Once the test is complete, the data is then processed for examination. Having this clear outline of the test conditions and the procedures played a critical role in correctly architecting our system.

System Specifications

As this project is intended to be an eventual replacement for wired vibrational analysis testing of spacefaring vehicles, the primary goal of the designed system is to match the current standards of measurement in sensitivity and precision. In accordance with that, the system specifications concerning the accelerometer and also the Analog to Digital Conversion were mostly set by Space Systems Loral (SSL). However, the remaining specifications regarding power management and that of our wireless subsystem were available to us to determine. Table 1 below breaks down each of the categories of specifications around which our hardware was designed.

Table 1: Technical specifications governing WASP system design.

<u>Category</u>	<u>Specification</u>	<u>Determined by</u>
Accelerometer (single axis)	+/- 500g measurement range 10kHz frequency response	SSL
ADC (single channel)	16 bits resolution 40 kbps minimum SPI interface	SSL + Team WASP
Battery (single cell LIPO)	3.7V nominal 3.0V - 4.2V range Low battery detection <3.0V and % charge remaining	Team WASP
Chipset - CYW43907	3.3V ± 10% VDDIO and VBAT Support 2.4GHz and 5GHz WiFi	Cypress
Board Footprint	< 100x100 mm	SSL + Team Wasp

Beginning with the accelerometer, SSL stated a desire to be able to detect upwards of +/- 500g's of acceleration with vibrational frequency content up to 10kHz. Since they also wanted 16 bits of resolution on the digitized samples, we reasoned that sampling at at least twice the Nyquist rate for this application (40 kbps) would be an appropriate target. The fastest compatible digital interface available to us on the CYW platform is the Serial Peripheral Interface (SPI). SPI is used to facilitate communication between our ADC and Wireless Chipset.

Moving on to the power management specifications, nothing specific was set in place by SSL other than the statement that the modules ought to be battery powered. We selected to go with a single cell Lithium-Polymer (LIPO) battery to power our system for their very high energy densities and capacities. Given that LIPO batteries are easily rechargeable and recyclable, this choice is a sustainable one that avoids having to throw away batteries after every use. As stated above, they commonly exhibit a 3.7V nominal voltage with a usable range between 3.0V and 4.2V. However, LIPO batteries can enter a state of deep discharge which is harmful to the cell should the voltage go below 3.0V. [7] As a result, our system needs a way to track the battery capacity over time to ensure that such a scenario does not occur. By selecting a 3.0V to 4.2V range battery cell, the power electronics architecture will be designed around that specification to generate the different voltage rails necessary for operating the various integrated circuits included in the design.

The final specifications category concerns the selection of a wireless MCU component. The wireless system-on-a-chip from Cypress Semiconductor, the CYW43907, supports both 2.4GHz and 5GHz WiFi transmission and due to its high levels of integration, condensed form factor, and reduced active and idle power consumption, was selected as the MCU of choice for this

application. As our design will feature the CYW43907 integrated onto the Quiksilver Module (described later in detail), the chip requires a 10% tolerance 3.3V rail to power both the VDDIO and VBAT rails as they are shorted together. [8, 9]

Design Rationale

When designing a piece of custom hardware, the initial objective is to understand exactly what tasks and functionality the board must be capable of performing. This includes determining the specifications for the system, whether they be quantitative or qualitative specifications. Once the technical specifications are well understood, it is then possible to begin selecting what kinds of circuit components will be required in order to accomplish the design goals. Selecting the components for the system responsibly is critical as the complexity and performance/requirements of the individual components will dictate the architecture for the entire system. If not done carefully, integrating everything together can become a challenging process. With an understanding of our technical specifications, we began to narrow down specific components for our system.

Accelerometer

The accelerometer is the analog sensor on our platform which will be used to measure the acceleration at a specific location on the spacecraft. These sensors come in many different form factors and have many different methods of translating acceleration into an electric signal. Our team was generally unfamiliar with the associated specifications of these sensors, so we reached out to one of our industry collaborators on the project, Space Systems Loral (SSL). They provided us with a base set of specifications that they use to select sensors as well as actual model numbers of sensors they currently use. Table 2 enumerates the key specifications of the Accelerometer set by SSL.

Table 2: Accelerometer Specifications.

Measurement Range	Shock Durability	Bandwidth	Sensitivity
+/- 500G	10 kG	10,000 kHz	>0.02 mV/g

Measurement Range: This is the range of the accelerometer.

Shock Durability: This is the amount of acceleration the sensor should be able to handle. While it may not be able to measure up to that range, this value ensures that the sensor will not be destroyed during shock testing.

Bandwidth: This is the frequency response range of the sensor. Post processing frequency analysis is typically done up to this frequency.

Sensitivity: This is how much voltage is driven from the sensor for each G exhibited on the sensor. The lower the sensitivity, the less voltage that is given off from the sensor.

Other:

Additionally, other specifications that are important to the sensor selection are Axis Count, Output stage, built-in analog to digital conversion and mounting style.

Axis Count: This is how many axes can be measured simultaneously from the sensor. Sensors on a cell phone for example measure in all three of the spatial directions, but high precision sensors are typically limited to a single axis/direction.

Output stage: This is a feature on many integrated sensors, where they have some type of analog front end to amplify and condition the signal for input into a digital converter.

Built-in Analog to Digital Conversion: Producers of acceleration sensors are integrating built-in analog to digital conversion block in their sensor. Doing this allows for very easy and versatile integration, only requiring a digital interface to communicate with the sensor.

Unfortunately this is only common to lower precision sensors and is not common with the sensors in the required range of our platform.

Having the required specifications and other specifications in mind, our team searched for sensors that could meet our requirements. Unfortunately, we were unable to find any sensors that had a digital interface that met the required specifications. That being said, many purely analog sensors were found, including many 3 axis sensors. Knowing that each analog sensor will need its' own analog to digital converter and support circuitry to support simultaneous sampling, we opted to focus our design efforts on a single axis design. We found many sensors from that same vendor that SSL uses (PCB Piezoelectronics) as well as sensors from Analog Devices. We compiled our top sensor choices into a matrix for comparison in table 3.

Table 3: Accelerometer Decision Matrix.

Sensor	Measurement Technology	Measurement Range	Shock Durability	Bandwidth	Sensitivity	Voltage Range	Mounting Style	Estimated Cost
3501B122KG	MEMS	+/- 2 kG	10 kG	10,000 Hz	0.20 mV/g	5-10V	Off board	\$588.00
3501A2060K G	MEMS	+/-60 kG	100 kG	20,000 Hz	0.03mV/G	5-10V	Surface Mount	\$1000.00+
ADXL1004	MEMS	+/-500G	10 kG	20,000 Hz	2.63mV/G	3.3-5V	Surface Mount	\$60.00

All three sensors in the decision matrix meet SSL specifications. Further discussions determined that either surface mount or external mount are satisfactory. The main difference between the PCB Piezoelectronics sensors (3501B122KG & 3501A2060KG) and the Analog Design sensor is that the ADXL1004 has an integrated signal conditioning block. [10-12] This conditioning block includes an output stage, meaning that we will not need another stage between the sensor and the ADC. Due to the design advantages of that sensor, as well as the reduced cost, we decided to go with the ADXL1004 sensor. One other key feature of the ADXL1004 is that it includes a self test mode, which we can use to verify that the sensor is functioning as expected before we attach it to the spacecraft. Since the sensor is surface mount, the physical layout (PCB design) will be driven around the requirements of the ADXL1004.

Wireless System-On-Chip

The wireless system on chip (SOC) in our system is analogous to the brain in a human: it manages all of the subsystems on our board while providing the main function of wireless communication. The choice of the wireless SOC therefore is critical - when selecting a SOC, we need decide based on our systems needs. These needs are: low idle power, as well as the SOC supporting a wireless communication protocol that meets our data rate, node count and operating range requirements.

Since our solution intends on sending data real-time to the server, we need to first select a wireless protocol that can support our expected bit rate. We have estimated a worse case bit-rate of 20Mb/s per node. Because of this protocols that are built on IEEE 802.15.4, which supports maximum bit rates of 100kb/s like Zigbee are not suitable.[6] While Bluetooth can support our data rate, the protocol was not designed for having hundreds of clients connected to a single access point. In our research we did not find any implementations of a network topology similar to what our expected needs are. The one protocol that catered to our needs was WiFi. WiFi is an evolving standard that has been designed to be a scalable solution, supporting bit rates well above our needs. WiFi offers tens of meters of range, multiple channels, as well as control schemes to allow scaling. One feature that appealed to us is called Modulation and Coding Scheme or MCS. This is a feature that WiFi uses to manage bit-rate and bit error rate depending on network congestion. [13] This is something that the protocol itself manages, providing a seamless experience. This feature was important to us since it provides provisions to support scaling this project.

Using WiFi as the protocol significantly narrowed our search, but we were still left with more than a few options. We still had a number of parameters to choose from, the most significant being the choice between a distinct MCU/RFSOC and an all in one chip. Two separate ICs offer more flexibility in MCU selection and selection of advanced features, but an integrated solution is more likely to save power, reduce design time, reduce design complexity and prevent extra software development.

Our advisor, Dr. Dezfouli has previously collaborated with Cypress Semiconductor before on WiFi IOT related projects. In the last few years Cypress acquired a WiFi and MCU combination part line from Broadcom. The part he was most familiar from that line is the CYW43907. Upon investigation, this part had many appealing features including: Ultra low power non-networked sleep (at the time one of the lowest power consumption parts on the market), needed peripherals (I2C, SPI, etc) and documented implementations of reference designs that we can derive our design from.[14] The combination of hardware met with software support through their forum solidified our decision to pursue this part.

It should be noted that since our project has been fabricated, more wireless combo chips in the same class as the CYW43907 have been released to the market, offering the same or better features. Not only are there new features and lower power, but also silicon level issues on the

CYW43907 that have forced Cypress to deprecate HW blocks have been since been resolved. Fortunately we were able to work around these deprecated HW blocks.

Battery Technology

For WASP, the selecting the correct battery technology was critical. This is because it directly influences design in many ways, including power management and system size. The battery needs to be small enough that it can be paired with a board and mounted easily, while also supplying a large range of current for a significant duration. To select a battery, we looked at all the different battery packages and chemistries and selected based on the characteristics of size, energy density, and rechargeability.

Common Coin Cells

Perhaps the most natural choice of battery from a small form factor is the coin cell battery. These batteries are the smallest of any consumer battery, and would easily fit on a WASP board while not contributing significantly to the overall board size. One benefit of these batteries is due to their small size, some chemistries like Zinc-Air can have tremendous energy densities, meaning they have the most energy in the smallest space. [15] However, despite high energy density, coin cells have many attributes that make using them unsuitable for WASP. The average capacity of a coin cell battery is around ~300 milliamp hours, making available test length a concern. In addition to this, even the largest coin cells from a size and capacity standpoint are only able to supply around 50 milliamps continuously, much less than would be needed for typical transmit power for our WiFi chipset.

Alkaline batteries

Another common option for batteries is Alkaline. These are the most common battery type you would see in a supermarket. Alkaline batteries come in a range of sizes, but many are too large and heavy to be used on a WASP board (a common 9V battery for example). Many batteries are moderately sized, like the AA and AAA varieties. Although the correct sizes exist, Alkaline batteries have undesirable characteristics for our application. Though most have effective capacities of around 3000 mAh, that can drop to below 1000 mAh for higher current draw applications, such as when we are continuously transmitting data. Alkaline batteries are also not rechargeable, meaning that over the lifetime of our system thousands and thousands of batteries would be bought and thrown away, contributing heavily to both the overall cost of our system, as well as producing a large amount of waste which would be easily avoided with a rechargeable battery chemistry. [16]

Modern Variants

The best choices for our application are the more modern battery chemistries that combine rechargeability and high energy density into a small package. We did not consider Lead-Acid,

which isn't produced in a form factor suitable for our platform, that leaves us with three options, as seen in table 4. [7, 17, 18]

Table 4: Comparison of various rechargeable battery chemistries.

Battery	Voltage (nominal)	Energy Density	Charge lifetime
Nickel-Cadmium (NiCd)	1.2	50-150 W-h/L	~2000 Cycles
Nickel-Metal Hydride (Ni-MH)	1.2	140-300 Wh/L	180-2000 Cycles
Lithium Polymer (LiPo)	3.7	693 W-h/L	400-1200 Cycles

Each of the three battery types listed above presented above would be a decent choice for a system like ours - they all allow for relatively small packages and high capacities, while also being rechargeable. However, the best choice was clear: Lithium Polymer.

Lithium Polymer

Upon concluding our battery research, LiPo batteries were determined to be the top choice for our application. LiPo batteries have the highest energy density among rechargeable batteries, meaning that they pack more energy into a smaller space, allowing us to support longer test durations using the same amount of space. That also means that we are able to charge the batteries less frequently which results in a longer total battery lifetime. LiPo batteries also come in a variety of form factors which allows the engineers using the system to choose a slightly larger or smaller battery size as needed, while maintaining the capacity to draw large amounts of current during a test period regardless of size. In addition to features benefiting users, lithium polymer batteries are a good choice from a design perspective. The nominal voltage is much closer to the voltages required by our chipset meaning less regulation and better efficiencies, and LiPo's are very common in a variety of devices, giving us a wider selection of mature, well designed parts for a more cohesive design overall.[7]

Power & Power Management

Voltage Regulators

As our system is battery-powered, many intermediate voltage rails are required to power the various integrated circuits on the module, all of which will have to be derived from the battery. Voltage regulators are a class of circuits designed to take an unregulated variable voltage (in this case a single cell LiPo battery) and produce a constant regulated output voltage over various load conditions. Given that our system will be operating in modes ranging from hibernate states, in which there is very little current draw, to wireless transmission states, where hundreds of milliamps may be consumed, it is critical that the selected voltage regulators be capable of operating over a wide load profile. When selecting an appropriate voltage regulator, the first question to ask is whether or not a linear regulator or a switching regulator should be used for the

application. Generally speaking, switching regulators generate DC output voltages by modulating the duty cycle at which currents are transferred to a load. They typically offer high efficiencies at the expense of noise, PCB area, and complexity.[19] Linear regulators, on the other hand, generate DC outputs by modulating the resistance of a pass element that transfers current to a load. These converters are the simplest to implement and have low noise profiles, but can suffer from lower efficiencies depending on the application.[20] There are numerous considerations to take into account when analyzing a voltage regulators performance, however, some of the most important ones include efficiency, stability, available output current limits, required dropout voltages, and noise performance.[20]

Battery Supervisor

Due to the use case of our system, a battery will predominantly be used to power the system. These batteries need to be operated within their operating specifications. This includes but is not limited to voltage and current limits. For our system, we are using a Lithium Polymer Battery (LIPO) as mentioned in the previous section. LIPO batteries have a nominal cutoff voltage of 3.0V. Beyond that voltage, the battery will be considered in an under-voltage state. This can lead to the battery not being able to hold a charge in the future or even becoming a safety risk. That being said, we will need a subsystem on our platform to monitor the battery voltage and cut off the battery from the rest of the system once that under-voltage condition is met. Another key requirement is to know the remaining charge in the battery. Since our system will run idle for hours and then begin a test lasting 30-40 minutes, we need to know how much capacity our battery has before starting a test. This information can be fed back to the server, which will make the decision if a module has enough remaining battery to complete the test. One common method to monitor the battery capacity is to integrate the system current over a time interval to get the battery charge. More complex methods like battery chemistry curve tracing and impedance tracking exist as well.

While our base requirements could be implemented with discrete components, many integrated circuit companies have built application specific integrated circuits (ASIC) to monitor the battery and report information back to the host device. This information can be cell voltage, cell temperature, instantaneous current and milliamp hours. Since many parts exist with different features, different footprint, and associated costs, we created a decision matrix with each part where we enumerated all the features, costs and footprint to better aid us in making an informed decision in table 5. [21-23]

Table 5: Battery Supervisor Decision Matrix.

Component Name	Methodology	Comm. Method	Features	Footprint & Package	Est Cost.	Comments
LTC2941-1	Gas Gauge: dQ/dt integration	I2C	Internal Sense Resistor, Undervoltage Protection. Low current (<70uA)	6mm ² QFN Package	\$2.80	1% Charge Accuracy. Very simple to interface with.
BQ27Z561	Impedance track	I2C	Charge Management, Thermistor management, Undervoltage Protection. Low Current (<93uA)	3mm ² BGA Package 8	\$2.50	Very complex to interface with. Need external sense resistor
MAX17260	Gas Gauge: dQ/dt integration	I2C	Undervoltage Protection. Time to empty estimator. Low current (<5.1uA)	9mm ² QFN Package	\$2.21	Need external sense resistor.

After taking many factors into consideration (bringup time, added cost, added power consumption, area required) we chose the LTC2941-1. It can be noted from the table that this is the most expensive part, but the other two parts required a high precision sense resistor and at the expected volume of our boards can add up to \$3 per board. It is also worth mentioning that this part had all the features that we required and very few unneeded ones. In comparison, both the TI and Maxim parts required a substantial amount of firmware simply to bootstrap the part.

Analog to Digital Conversion

When considering what type of ADC to use for our system, the primary issues we took into consideration were resolution and sampling rate, power consumption, required signal conditioning, and digital interface. However, the first task was to select an ADC architecture, as they come in a diverse range of forms. Our project requires 16 bits of resolution with a minimum sampling rate of 40 ksp/s. Successive approximation register (SAR) converters permit high speed conversions with moderate to high resolutions and thus represented an excellent choice for this application. This topology is an excellent middle ground between delta sigma converters, which offer high resolution at the expense of slower speeds, and pipeline converters that provide very high speeds at the cost of lower resolutions. As we were working with an Analog Devices accelerometer and were in contact with one of their Field Applications Engineers (FAE) to help us in designing this critical analog stage of our module, we opted to also select an ADC from Analog Devices in order to aid in the process of interfacing the accelerometer with the ADC. When narrowing down their selection of ADCs that met our specifications, we settled on a couple of options and again adopted a matrix of criteria to select a specific part. Those criteria are outlined below in Table 6.

Table 6: ADC Decision Matrix.

Part Number	Resolution and Sampling Rate	Power Consumption	Integrated Features	Digital Interface
ADAQ7980 (SAR)	16 bits 500 ksps	16.5 mW	ADC Driver LDO for Vdd Vref Buffer	SPI
AD4008 (SAR)	16 bits 500 ksps	4mW	High-Z Mode	SPI

Looking first at the ADAQ7980, this part is a 16 bit SAR ADC featuring extremely high levels of integration. As a system-in-package (SiP), the device features critical signal conditioning and power management components integrated inside the package. The part includes an integrated LDO to power the system, high impedance ADC driver on the analog inputs, and a voltage buffer on the reference rail. These components, in addition to several integrated passive components, dramatically reduce the complexity of external components needed to make the ADC function. For our system architecture at the time, only an external filter on the analog input and a voltage reference would need to have been provided. However, after investigating the part further, it was discovered that the integrated ADC driver amplifier required at least 1.3V of headroom on the common mode input. This means that in order to achieve the full scale operating range of 0 to 3.3V (and therefore the maximum g-range on the accelerometer), we would have had to generate an additional voltage rail for our system thus adding more area and power consumption, which counteracts the benefits of choosing this component in the first place.[24]

For this reason, we selected the AD4008 for our design. While the part requires an external LDO to be powered, such an addition is very simple and consumes minimal current. In addition, it has high-Z mode functionality which is an integrated high impedance input stage that accomplishes the same goal as an integrated ADC driver, but mitigates the voltage headroom problem of the ADAQ7980. [25]

PCB Design Software

Since our design requires us to build a custom printed circuit board (PCB), there is the need to determine what software suite our team would use to design and manage the board. PCB software allows for an electrical schematic to be translated into a physical design. This physical design is what will be given to a manufacturer to produce and assemble a PCB. Our design is relatively complex and has several subsystems and necessary support architecture. Due to those complexities, we cannot select just use any PCB design software. A criteria was needed to

determine what our designed requirements were for the project. We have listed all requirements in table 7.

Table 7: PCB Software Requirements.

Feature	Required/Optional	Comments
Component Libraries	Required	Having components libraries will allow our team to focus on our design rather than designing and adding components to a library. Having verified libraries will allow for further confidence in our design.
Multi-Layer Support	Required	We anticipate our design will require at least 4 layers for signals and power routing.
Parametric Rules	Required	Allow us to set rules for the design (spacing, trace thickness, keep out).
Built-In Signal & Power Integrity Simulation	Optional	
Manufacturing Friendly Export	Required	
Free to Use	Optional	We are okay paying for low-cost education license of the software.

Once the base requirements were established, we looked for software packages that could meet our requirements. Table 8 was compiled to compare our 3 main options: Altium Designer, Cadence OrCad, and Eagle. [26-28]

Table 8: PCB Software Decision Matrix.

Feature	Altium Circuit Studio	Cadence OrCAD	Eagle
Component Libraries	✓	X	X
Multi-Layer Support (>4)	✓	✓	X
Parametric Rules	✓	✓	X
Built-In Signal & Power Integrity Simulation	✓	✓	X
Manufacturing Friendly Export	✓	✓	X
Free to Use	X	X	✓
Comments	Student License \$150 per year. No difference from Commercial License	No student version available upon contact	Only supports 2 layers, can only make a PCB 80cm ²

Given the features outlined in Table 8, it was agreed upon that Altium Designer was the optimum choice for this project. Although the software costs money, having a verified component library maintained by Altium is the primary motive why we decided to go with this PCB design software. Although only one license was purchased, the license can be shared across several computers with the only requirement being that only one person can use the license at a time.

Firmware and Software

Overview

To be a complete solution, the WASP system needs to provide all of the device software necessary to make each board functional as well as server software to govern the system as a whole. This means that our system needs to include board software (client) and server software, each running on a completely different hardware platform with completely distinct set of capabilities and tools to leverage them. These two pieces, though separate, need to communicate seamlessly which requires the design of each component to happen in tandem with all the others to create more cohesive system.

In addition to requiring multiple pieces of software, the WASP system was also required to be highly configurable and scalable, which represented a unique challenge for us. The system needed to be able to handle a variable number of boards and data rates while ensuring the same reliability as a fixed system. This meant that we needed to make large protocol design decisions early on in our timeline, and design the software around our obstacles.

We break down the description of the software into the three categories of System Multi-threading, Drivers, and Networking to elaborate on each piece in more detail.

System Multi-threading

Client

On the client side (the side of the WASP boards), our hardware determined our software ecosystem. The CYW43907 chips come with WICED Studio, a free set of APIs developed for use on that chipset to provide the base WiFi capabilities, as well as other convenience functions to make app development easier. Besides creating a custom TCP/IP stack from scratch, WICED is the only software solution available on the 43907 platform, making the choice an easy one.

Every build on the WICED system that makes use of WiFi must use an RTOS (Real Time Operating System) [29], of which there are two choices available in WICED: FreeRTOS, a mature open source RTOS, and ThreadX, a proprietary RTOS used in many commercial applications which comes free as pre-compiled binaries with WICED. Both offer similar features, and in our case, the WICED system of APIs abstracts the underlying RTOS details away from us, which makes both options good choices. In the end we settled on ThreadX, as it is the standard RTOS for most WICED builds, and it supports over the air updates, which we wanted to integrate as a part of our system, and FreeRTOS did not.

Each application built on WICED has a base system of threads. The lowest level contains the networking thread, which handles the movement of data to the WLAN core of the 43907 from the application core, and the hardware IO thread, which handles the calls to hardware to do IO operations. These two threads are transparent to the application in user space, and only run when called via an API function. On top of the lower level is the main thread, which runs all of the application code, and is where our WASP application codes lives. The main thread can spawn additional threads which we make use of to run our main data manipulation function, which combines sampling data from the ADC and sending data over wifi. This function makes use of the lower level threads to do the raw data movement, but when combined as necessary by our application, the stack needs become too great for the main thread, requiring we spawn one with a larger stack, which in our case is 10240 bytes, or slightly less than 2x that of the main thread.

In addition to the main level of thread, there exists a supervisor thread called the “watchdog” which monitors all of the other threads running, managing a timer resetting the system if one

thread holds the processor for too long (at timer expiration), potentially signalling an error. In our case, the watchdog thread offers little utility, even becoming a source of error. Each thread has a priority associated with it at creation which signals to the RTOS scheduler which threads should be scheduled to run. In our case, the hardware IO thread that is handling the sampling is a higher priority than the watchdog, and because it runs so frequently, the watchdog is unable to run and interprets this as a reset-worthy event. Unfortunately we are unable to disable the watchdog, so instead we must manage the timer manually from user space, resetting the reset timer at each loop iteration.

Server

On the server side, we were less limited by hardware, and as such we had free reign to choose from any number of hardware and software environments to develop and test the server side software. One important thing to note is that the WASP server isn't involved in any post-test data analysis and processing. Because of the amount of data generated, a large server or cluster of servers wouldn't be unreasonable for the most intensive test plans, but due to the fact that our server would not participate in that process, we were free to select a more appropriate embedded environment.

Our requirements for a server device were then easily defined: we would need a reasonably fast device with a few cores to support multiple threads of execution, with support for a well documented network stack and programming environment. In an ideal scenario, our server software would be deployed on the router that serves as the communication hub for our system. This configuration would allow us to have finer grained control over the low level wireless operation of the router from the server software, while also eliminating any extra configuration of the system necessary from an engineer with an external server setup. Unfortunately, this setup has its problems from a development standpoint. First, router firmware is very lean and as such does not come with a build environment, meaning rapid prototyping is not possible. For each change that gets made to the server, we would need to compile an entire firmware image from scratch, and then reflash the image on to the router, wasting valuable time. In addition to that, worrying about integrating fine wireless control into the control scheme of our application really only makes sense as a fine tuning step once the application is built and proven, not before. For these reasons, we decided to opt for an external system for development.

Fortunately, our router of choice (a Linksys-WRT3200ACM) has very standard hardware configuration consisting a 1.8 GHz ARM core CPU made by Marvell with 512 MB of RAM which made it very easy to pick a platform to emulate the processing power if we ever pursued running the server on the router. We ended up picking a Raspberry Pi 3 B+ system to run our server, for a few reasons: the 4 core ARM CPU running at 1.2 GHz possesses a very similar amount of processing power and similar amounts of RAM, and a 300 Mb Ethernet port meant we could link the server and router using a single cable and support the same levels of network throughput. The most attractive feature however, was the Linux operating system and included GCC C compiler, allowing us to leverage our existing expertise of the Linux network stack, POSIX compliant pthread threading interface, while also being able to make use of the wealth of documentation and examples whenever we needed.

The server side threading choices were perhaps one of the most important design decisions we made from a software perspective. While each client manages its own data transmission, the server is responsible for capturing the data stream for every device, up to the allowed maximum of 300. To make the server as efficient as possible, the server divides up the work of receiving transmission to four threads, one running on each of the processors four cores, assigning which board goes to which thread dynamically to effectively balance the load.

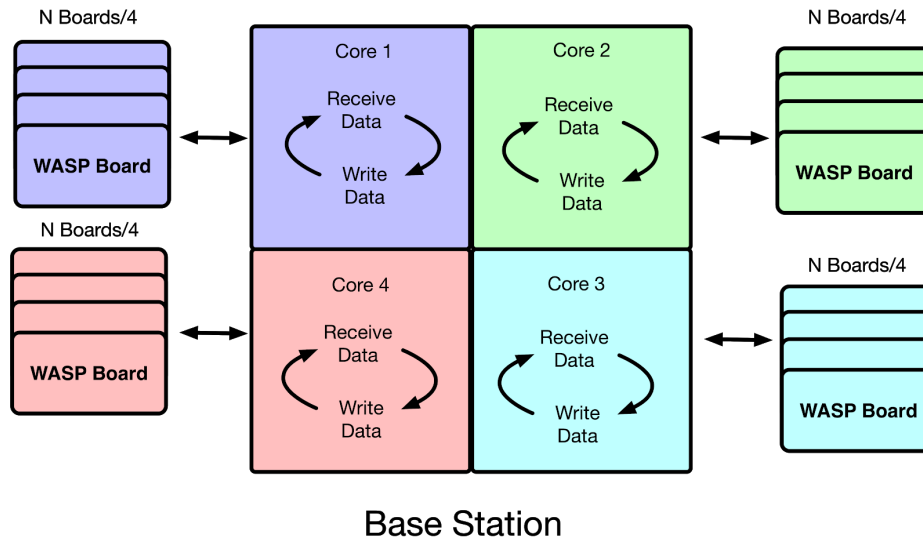


Figure 4: WASP Server work division implementation.

In addition to the packet reception threads, the server employs various other threading techniques to manage some of the servers communications needs. A single thread runs in the background to send a sleep command to any WASP board asynchronously when a test is terminated by the engineer. The server will also spawn a thread each time a board connects to receive a calibration value from the connecting board. Spawning and then returning from threads allows the server to stay in communication with the board as it takes a few minutes to calibrate itself, while not interfering with the main loop handling board registrations that needs to run on a finer timescale.

Drivers

The WASP module has a number of different hardware components on board, each using a different protocol for communication back and forth with the main chip. For each of the devices, a driver needed to be designed to implement the communication so we could use the devices individually for our purposes. For each of the protocols we needed a driver for (SPI, I2C), we designed a generic low level driver, capable of reading from and writing to arbitrary addresses in standard one byte and two byte quantities. In addition to the low level driver, we implemented a higher level driver API for each device that wraps the low level driver and additional logic into a few convenience functions, abstracting away the details of the device communication into a simple function call. This method allows for easier app level control of the devices, while also

allowing for future device integrations to be easily integrated by re-using our generic low level driver and wrapping device specific functionality into an API as we did.

In addition to our core device drivers, we also implemented a set of APIs around common tasks for convenience. APIs exist for simple tasks, such as toggling and blinking LEDs at custom intervals, as well as more complex tasks, like controlling OTA updates, setting power modes, and switching WiFi networks. This ideology of providing high level functions to the application allows for easy implementation of features in the future, as well as support for other applications of our hardware and software by other teams.

Wireless Module Design

Having performed extensive research into the necessary components and various integrated circuits required to achieve our project goals, we began the process of formally designing the system architecture. This process requires analyzing how the different parts can be integrated together. Designing the wireless module began with a simple high level block diagram and progressed into a detailed diagram of the system functionality, replacing black box sub-systems with actual integrated circuit part numbers, which allows a designer to then smoothly transition into producing an electrical schematic and PCB layout.

Architectural Block Diagrams

To understand our system better we have created two block diagrams that describe our system in varying level of detail. Figure 5 shows three blocks: *Power Electronics & Battery*, *Wireless SOC*, and *Accelerometer and Analog to Digital Conversion*. That subdivision explains the three main portions of our system and we used arrows to imply the general connectivity.

In Figure 6 we elaborate from that first diagram and explode each of those three blocks into all of the major components. We used the block color to denote what subsystem it belongs to. The figure also contains a wiring key to show the reader what type of connection is made between block. For example, the connection between the Quicksilver and the Gas Gauge is a digital interface (I2C) whereas the connection from the ADXL1004 to the ADC 4008 is an analog one. It should be noted that while this shows all of the major components, it does neglect passive components, bootstrapping components, and the diagram does not reflect the physical layout of the system.

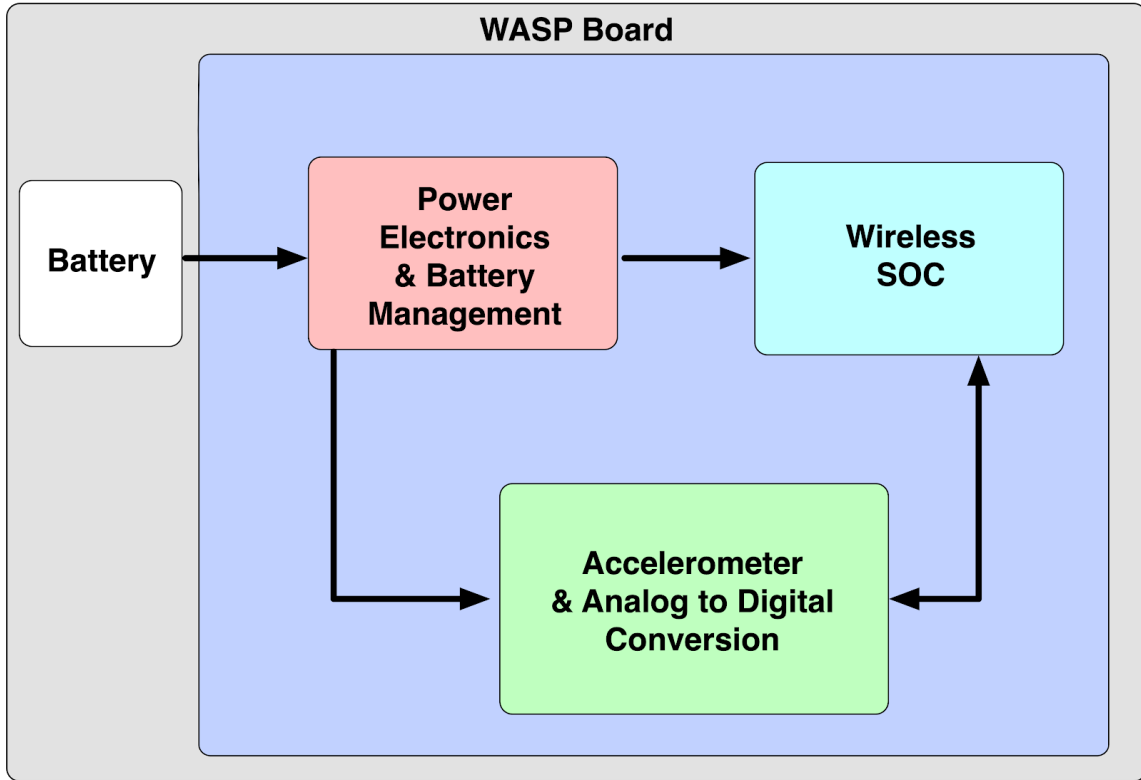


Figure 5: High level block diagram separated by major subsystem.

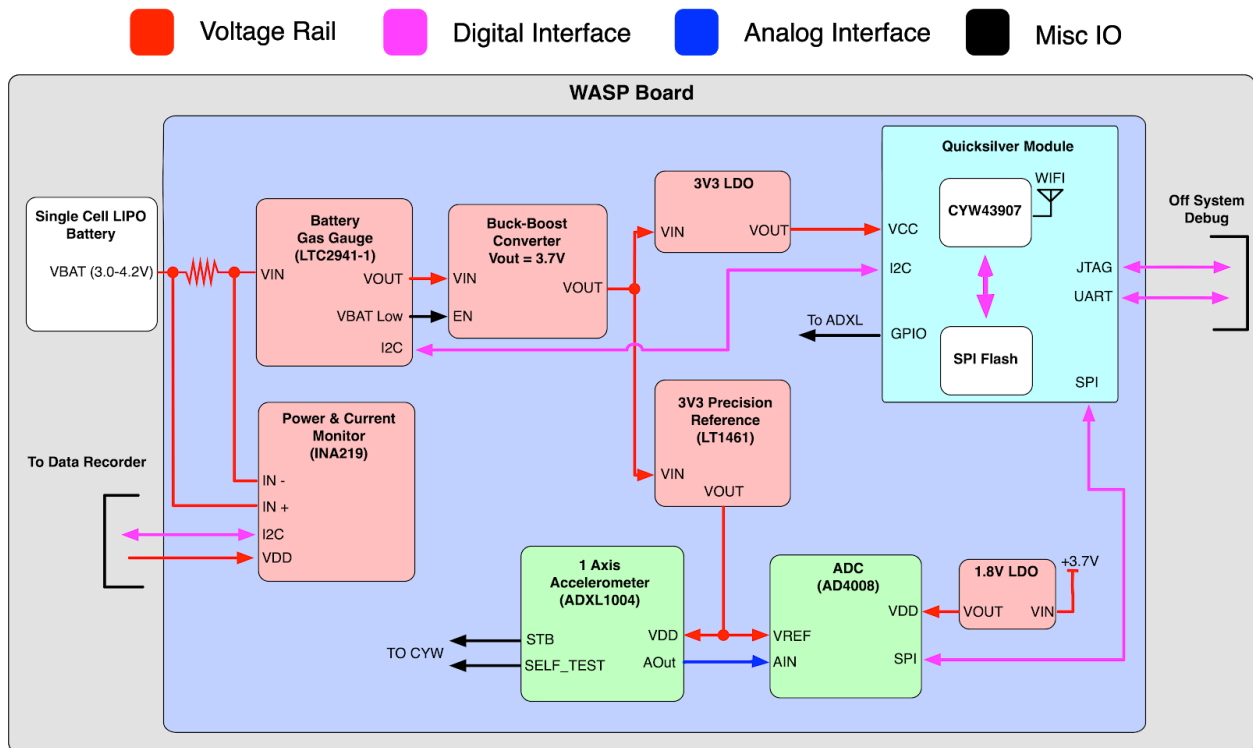


Figure 6: Low level block diagram showing all major components and their connections.

CYW43907

Our wireless chipset, the CYW43907, is the most significant piece of hardware in our custom sensor module. This chip contains the wireless and application processor needed to sample, wrap, and then send the ADC samples to the server. This chipset is the most complex piece of silicon on our system, and required an intense design effort to ensure proper functionality. Thankfully the electronics and manufacturing industry are aware of this and have engineered a solution. To make integration easier and increase wireless performance, companies like Murata will take the wireless chipset (CYW43907) and all of the support components such as the oscillator and diplexers and solder them onto a small PCB putting it into a conformal coating and RF shielding. This completed device is typically called a *module* and this can be soldered directly onto a design. For our design we used the LBWA1UZ1GC from Muarta. The only external components needed to complete the integration of the wireless chipset are: SPI Flash (Used to store Application Firmware), 32.768kHz crystal oscillator (Used for the Hibernate Block), an inductor (1.2V buck rail on CYW) and 2.4/5GHz antenna. The SPI Flash, Oscillator and Antenna are not included in the module because they are dependent on the application where the components inside of the module are necessary for every application. The inductor is not inside of the module due to the size of the component. [9, 30]

Initially we wanted to directly place the LBWA1UZ1GC on our PCB, but due to unavoidable constraints our industry contact was unable to review our design. Our integration efforts were mostly based off of the CYW943907, which is a evaluation platform produced by Cypress using the LBWA1UZ1GC module. Since we didn't direct support we had to use the reference design schematic to make connections, but we didn't always know the purpose of connections like bootstrapping configurations. At this point we had to investigate an alternative or decide to take that risk to directly integrate the package. After some investigation, we found an evaluation board made by Arrow Electronics called the Quicksilver that had the CYW43907 in the LBWA1UZ1GC package. The Quicksilver contains a Arduino-like motherboard and a daughter board that



Figure 7: An annotated image of the Quicksilver module.

contains the LBWA1UZ1GC module, the SPI flash, the 32.768kHz crystal oscillator, 1.2V Inductor and a single 2.4/5GHz antenna. This daughter board can be soldered onto our custom PCB and will function just as intended. The daughter board simplifies many of the connections, taking care of bootstrapping and power rails configuration. The only connections it exposes are JTAG for programming, UART for communication, a single 3.3V in and all of the GPIO and all of Hardware PHYs (SPI, I2C, RMMI, USB etc). [8] Since the board was intended for use on the Quicksilver evaluation kit (which has the all the design files including the schematic public) we can ensure the connections on our board are copies of one another which allowed us to verify pin functionality on the quicksilver platform before we commit them to our own design. Having this ability allowed us to design with confidence when interfacing with our other subsystems like the ADC and Gas Gauge.

ADXL1004

The single axis accelerometer we chose is the ADXL1004 from Analog devices. This part uses Micro-Electro-Mechanical Systems (MEMS) methods to measure acceleration. The device functions like a wheatstone bridge with one of the resistor legs of the bridge being the MEMS sensor. Depending on the acceleration applied to the sensor, a voltage potential will be created across the output and the voltage is directly proportional to the acceleration. [31]

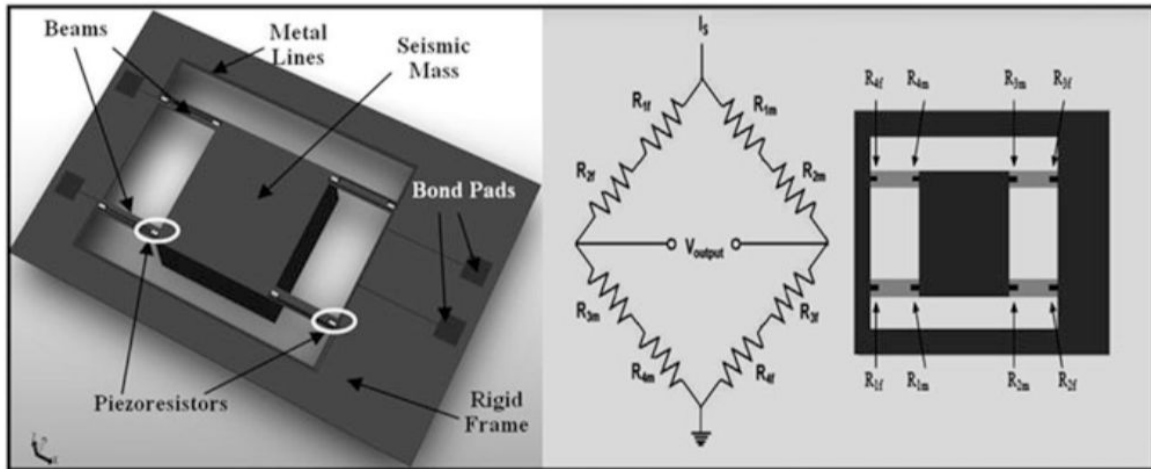


Figure 8: A physical and circuit model representation of a MEMS accelerometer.

The ADXL1004 uses that principal to build an integrated solution. The ADXL contains an output stage, amplifying the voltage from the MEMS element. Doing this not only amplifies the signals, but also decouples downstream systems like the ADC from loading on the MEMS element. The part runs on a single voltage rail; this rail not only feeds the output stage, but also is the reference directly across the MEMS element. Since the parts measures +500G to -500G the sensors outputs $VDD/2$ when no acceleration is measured. The sensitivity of the sensor measured in mV/G is directly dependent to the voltage. [10] The sensitivity is defined by the equation:

$$Sensitivity \left(\frac{mV}{G} \right) = \frac{4}{5} * VDD \quad (eq. 1)$$

Since our voltage rail is 3.3 volts our sensitivity is 2.64mV/G. In context to our system this mV step needs to be able to be detected by our ADC. Using 16 bits and considering INL and DNL we are well within that limit. All of the interpretation of acceleration is directly related to this value, that being said we decided to decouple the voltage rest of the system that requires 3.3V. To do this we used a precision voltage rail to ensure a voltage rail with a low variance (<1%) across temperature and board-to-board (Further discussed in a later section).

The frequency response of our system is specified up to 10kHz. We placed a first order RC filter at the output of the accelerometer, which filters out noise generated from the internal oscillator of the output stage as well as vibrations above 10kHz. Working with a field applications engineer who supports this line of parts, we designed a filter with practical values. We designed the filter with a $R=715$ ohm and $C=10nF$. This gives us a 3dB cutoff at 22kHz. Although this value is 10kHz greater than our desired maximum frequency, this was done to prevent minimal amplitude loss at the 10 kHz frequency. The RC filter was simulated in LT-Spice to verify the transfer function. In simulation we measured ~ 800 dB loss at 10KHz. If the roll off of the filter proves ineffective, we have provisioned on our design to support a second order RC filter by placing pads down for extra components. Currently on our design we do not populate those extra pads.

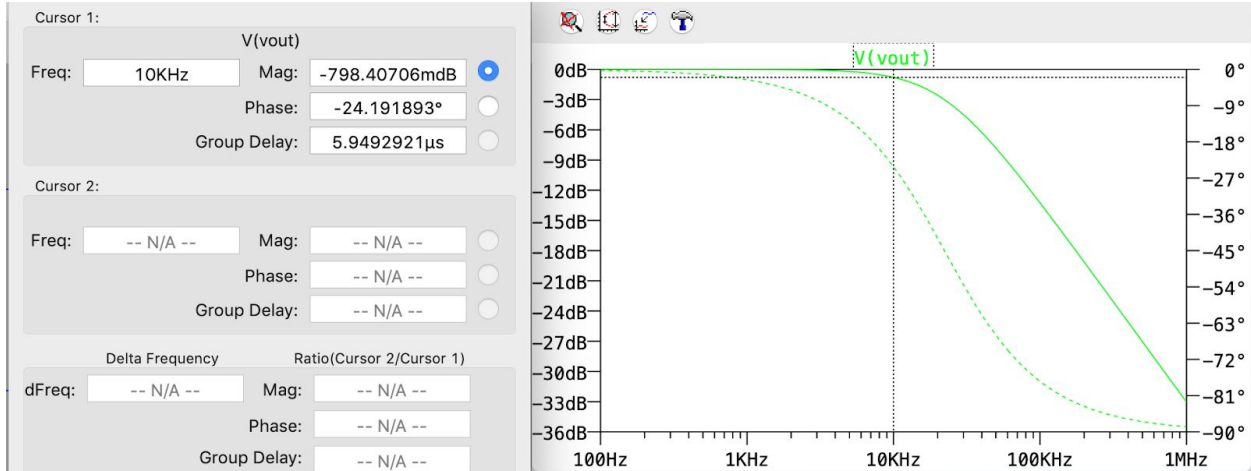


Figure 9: RC filter Bode plot and cursor information from LT-Spice Simulation.

The ADXL contains 3 auxiliary IO: Standby, Self Test, Overrange.

Standby

Driving the standby pin high will put the part into a low power mode ($I \approx 228\mu A$), doing this will shut down the oscillator block on the part. We have connected this to our wireless SOC to put the part into standby when needed.

Self Test

Driving self test high will use electrostatics to displace the MEMS element. The amount of displacement is based on the VDD of the sensor. From there you can measure the delta between the sensor output before and after self-test was asserted. If the delta is within a range specified by the data sheet the user or software can determine if the sensor is fit to be used. We use this feature on our system to verify the sensor is functional before every test. We have connected this signal to our wireless SOC to trigger the self test.

Overrange

Overrange is an output that will be asserted by the ADXL1004 in case the sensor is subjected to greater than 1kG, when that condition is met the ADXL1004 will drive over range as well as the part will be put into standby for 200us. Putting the part into standby protects the sensor and oscillator. We have this output tied to a HW interrupt on our SOC. This can be used to relay back to the server in an over range event has occurred.

Mechanical Considerations

To correctly transfer mechanical vibration to the sensor, proper mounting of the module to the device under test needs to be considered. Since we are not versed in this topic, we leveraged the datasheet and evaluations modules for the ADXL1004. The primary goal is to secure the board to device under test in a way that the board itself doesn't generate harmonic vibrations. To mitigate this, rigid mounting is needed to be placed directly around the sensor. [10] The ADXL1004 evaluation platform provide a mounting cube, which directly screws into the PCB. We designed the mounting of our system to be able to attach to the same mounting cube.

AD4008

The analog to digital converter selected for this design was the AD4008 from Analog Devices. The device is a single channel, 16 bit, Successive Approximation Register (SAR) ADC. This part features a maximum sampling rate of 500 ksp/s, which more than satisfies the 40 ksp/s specification we determined in order to satisfy the Nyquist rate. We chose this particular component due to its extremely low power consumption, high-Z mode functionality, and an integrated SPI interface.

Looking first at power consumption, the part claims to dissipate only 5mW with high-Z mode enabled sampling at 500ksp/s, which implies only a few milliamps of current draw when sampling and performing conversions. In addition, the converter powers down at the end of each conversion cycle to save power implying that power consumption is directly proportional to sampling speed. The part requires a V_{dd} of 1.8V to power the part and can accept a reference voltage of 3.3V making the ADC compatible with our accelerometer. It is also worth noting that since V_{ref} is 3.3V, this converter has an LSB size of 50.35 μ A and is therefore easily capable of resolving the accelerometer sensitivity of 2.64mV/g.

Moving on to the high-Z mode provided by the AD4008, this is a feature intended to mitigate the effects of nonlinear charge kickback at the start of each acquisition phase when the sampling capacitors connect to the input voltage. This high impedance mode allowed us to simplify the complexity of our design by not requiring an input signal buffer to drive the ADC. This means that the analog voltage output from the ADXL1004 accelerometer, following the RC low pass filter, can be directly connected to the analog inputs of the ADC.

Interfacing with the part is done over Serial Peripheral Interface (SPI). SPI is a serial interface that is run as a Full-Duplex interface. While SPI is not a standardized protocol, it typically contains 4 connections: Master Out Slave In (MOSI), Master In Slave Out (MISO), Clock and Chip Select (CS). MISO and MOSI are used for the data transfer and CS is used to tell the slave that they are selected and to expect a transaction to occur. Devices can choose to clock in data on the rising edge of the clock or the falling edge. Typically vendors will specify which mode their part supports by denoting the part as CPHA=1 or CPHA=0, where 0 means data is clocked in on the rising edge and 1 means the falling edge. [32] In some newer parts the clocking edge is potentially configurable, but in our case the AD4008 clocking is fixed. Care was taken to verify that the ADC and CYW43907 have compatible versions of SPI.

The ADC contains multiple modes that are categorized into two groups: 3 wire modes and 4 wire modes. 3 wire modes are used when the master device (CYW43907) does not need to send data (over MOSI) to the ADC. 4 wire modes are used when the master (CYW43907) does need to send data (over MOSI) to the ADC. For our application High-Z needs to be enabled and this is done via a register write, so a 4-wire mode will have to be used during the initialization of the ADC. After that is done, we switch to a three wire mode since no data needs to be sent to the ADC while we are collecting samples from the accelerometer. A signal called conversion (CNV) is used to initiate any a transaction between the CYW43907 and the ADC. This could be

analogous to chip select but has a dual role and does not have the same characteristics that one would associate with chip select. Not only does CNV tell the ADC to get ready to accept incoming data and shift data out, but also is used to turn on the acquisition stage of the part and convert the current voltage between V+ and V- and convert that to a code that can be then shifted out. To start a conversion the line transitions from low to high and stays high for a certain amount of time as specified by the datasheet (tconv). From there CNV is driven low. Once CNV is low the master can start driving the clock for 16 cycles, during each of those cycles a bit will be shifted out, the total of which are the two bytes of conversion data. This procedure is then repeated to take another sample. [25]

The ADC looks at the state of MOSI at the rising edge of CNV to determine what mode it needs to be in. For our use MOSI needs to be idle high at all times. This presented us with a problem since the SPI hardware block on the CYW43907 does not idle high and while we could have a weak pull up, we were concerned about contention with the push-pull driver of the CYW43907. Because of this, we opted to use a bit-bang SPI driver on our platform so we could configure the MOSI line to idle high via software. We have connected our ADC via pins that are multiplexed between GPIO and SPI so we can experiment with both versions. [9] The bit bang driver allows for a maximum clock rate of 1MHz, which provides a maximum bit rate well within our needs. [33] In the software section of the report we further discuss this implementation and customizations to this driver that were needed to have the ADC function as intended.

Power Electronics

Buck-Boost

As our system is being powered off of a single cell Lithium Polymer (LiPo) battery, which have a nominal voltage of around 3.7V and a usable voltage range between 3.0V and 4.2V, we wanted to ensure that our system could capture that entire range in order to maximize the battery lifetime of our system. Thus, a Buck-Boost converter was selected to provide the main voltage rail for our system, from which any other voltage domain can be derived. Since our accelerometer (ADXL1004) requires a minimum Vdd of 3.3V and was to be powered by a separate 3.3V precision voltage reference chip, we chose to select the output of our Buck-Boost to be 3.7V to provide enough room for any required dropout voltages feeding into any cascaded regulators. Note that if simply a Buck converter had been used and set to 3.7V, almost half of the LiPo voltage range would become unusable.

In regard to selecting a specific Buck-Boost component, output current, noise, quiescent current, and efficiency were among the top selection criteria. We used Texas Instrument's online software Webench Power Designer to help narrow down the selection of converters. The TPS63020 was selected for our design and features a resistively programmable output voltage capable of delivering a maximum of 4A load currents.[34] Simulating the performance of the converter across our input voltage range and with load currents up to 500mA, Webench

simulates a converter efficiency range between 88% at light loads up to 94.5% for our max load currents. The efficiency at light loads is aided by a power saving mode that occurs when the average inductor current falls below 100mA. In this state, the converter switching frequency is greatly reduced to maximize the efficiency across the entire operating region. Furthermore, the TPS63020 switches automatically between step down and step up modes, consumes less than 50uA of quiescent current, and in normal operation switches at 2.4MHz.[34]

Precision Voltage Reference

The use of a precision voltage reference was selected for this design in order to increase the measurement precision of the analog signal chain in our design. It was desired to power the accelerometer (ADXL1004) and ADC (AD4008) with the same supply so that the measurement vibrational signals and corresponding digital codes are effectively correlated to the same reference supply. Thus, if changes in Vdd affect the accelerometer readings, then those same changes will be reflected by the ADC. Since the accelerometer output and sensitivity is directly ratiometric to its supply voltage, it is desired to minimize changes on this rail. Precision voltage references provide extremely constant voltages that are resistant to temperature changes, noise on the supply rail, and process variation.[35] Powering our accelerometer or ADC directly from the Buck-Boost converter (or even an LDO) would risk unwanted switching noise or voltage ripple resulting in measurement inaccuracies.

For our design, we selected the LT1461 as our precision voltage reference making sure to consider temperature drift, power supply and noise rejection, and available output current. The device comes in a fixed 3.3V output form, which satisfies the ADXL1004's minimum supply voltage requirement. The part can deliver up to 50 mA load currents, which is more than enough for our application, and the output voltage is guaranteed to within $\pm 0.08\%$ of the nominal value with a temperature drift of only 12 ppm/deg C.[36] By selecting a 3.3V reference rail, our 16 bit ADC has an LSB size that comes out to 0.05mV, which is capable of resolving the corresponding accelerometer sensitivity of 2.64mV/g.

Linear Regulators

With a 3.7V main system voltage rail established by the Buck-Boost converter, it was necessary to then derive any other required voltages to satisfy other various ICs on the board. The CYW43907 normally requires a 3.3V $\pm 10\%$ VDDIO rail as well as a VBAT that can range from 3.0V to 4.8V. However, since we integrated the Quiksilver Module onto our board, that module has the two rails shorted together, thus we only had to supply a 3.3V rail to both pins. In choosing which type of regulator to use to generate 3.3V, two primary choices existed: a Buck Converter or a Linear Regulator. Note that the existing 3.3V precision voltage reference cannot be used for this purpose as it can only provide a maximum of 50mA of output current.[36] While a Buck converter could accomplish this task with efficiencies upwards of 90% (verified using TI Webench), we did not opt to use this regulator topology as it would contribute extra switching noise to our system that may affect the sensitive analog components. In addition, cascading a Buck converter from a Buck-Boost converter requires a second inductor on the board, which

occupies a large footprint and can likely cause coupled EMI effects. A linear regulator was chosen to supply this 3.3V rail as they are simple to implement, occupy smaller board areas, and do not contribute switching noise. In this scenario, a linear regulator stepping down a 3.7V rail to a 3.3V rail possesses a maximum efficiency of 89.19%, however, in order to better justify this decision over a Buck converter, calculations were estimated using Ti Webench tools to back-calculate the expected input current draws from the battery if either regulator was used. While a Buck was estimated to consume less current, the extra battery lifetime that this current reduction would provide was deemed insignificant compared to the additional noise performance to outweigh the benefits of using an LDO.

We selected the TPS73733 as the 3.3V LDO for our design for its appropriate dropout voltage and available output current abilities as well as its stable transient response and load regulation performance.[37] By using a fixed voltage version, we were able to reduce board area and cost by not requiring additional resistor dividers.

The second LDO required on the board was a simple 1.8V component to power the V_{dd} of our analog to digital converter (AD4008). While dropping down from 3.7V to 1.8V results in an LDO efficiency of 48.65%, since the AD4008 is expected to only consume a maximum current when sampling of around 1mA, the dissipated power losses are minimal. Thus, an LDO was once again preferable to a Buck converter for such light loads. The TPS78018 was chosen for its fixed output voltage form, low 500nA quiescent current consumption, miniaturized packaging, and strong power supply rejection from switching frequencies coming from the Buck-Boost converter.[38]

Gas Gauge

The gas gauge we chose (LTC2941-1) serves two purposes on our system. The first is total power consumption, which is role of keeping track of the remaining charge that is on the battery. This part continuously tracks the battery capacity as long as a battery is connected which is important because our SOC will periodically undergo wake and sleep cycles. Even though the SOC is asleep, other subsystems will still be idling and consuming power. In a commercial solution like an iPhone this part is typically married to the battery pack and is not on the main mother board meaning that the part is always on, and needs to be programmed at the factory with the correct settings. Since we do not have that option on our design, we needed to place the gauge as far upstream as possible. We decided not to place it at battery connector, but rather after our reverse voltage protection PMOS transistor. Although the PMOS we chose has a low R_{DS(on)} of 3mOhm and it will incur I^2R losses, it was unclear in the data sheet for the gas gauge if the device could handle the reverse voltage and to be safe we put it down stream of our reverse voltage protection. [39] Since the battery is not married to the gas gauge we need to have scheme to ensure proper tracking. We propose that an operator or engineer will take a fully charged battery with a predetermined capacity and connect it to the system; our system knows that this is the first power up and set the accumulated charge register via I2C to reflect how many coulombs this battery has. Since the system will not be powered down until after the test is complete, the coulomb counter register will correctly reflect the capacity. The SOC can read then

the accumulated charge register whenever it needs and report back to the server with that information.

The second role of the gas gauge is to protect the battery from under voltage discharge. To accomplish the low battery detection the AL/CC pin on the gas gauge is connected to the enable pin of the 3.7V regulator, so when the battery hits the 3.0V threshold the gas gauge drives AL/CC low and de-enables the 3.7V regulator, shutting down the regulator and everything downstream of it. The current draw on the battery will not be zero since the gas gauge and the 3.7V regulator will be drawing quiescent current. This draw is extremely low ($<70\mu\text{A}$) and we deemed that as not detrimental to the battery. [23]

Both the low battery detection and the correct battery capacity values are not set by default on the gas gauge, so when the system powers up for the first time it will need to correctly set the registers. More information about the Gas Gauge register configuration is in the software section of the report.

In Circuit Power Measurement

Although we have a gas gauge on board, since this design is a proof of concept we would like to have a way to measure instantaneous power on board. To do this we placed an INA219 from Texas Instruments on our system. This device combines digital logic and a current sense amplifier to take a voltage drop measurement across a sense resistor and convert that information into a power or current measurement. This measurement can be transferred over I2C to a event recorder. The INA219 has a variable gain amplifier that feeds into its' internal ADC. Having control over the gain allows for better utilization of the ADC's full scale. This also allows the use of a single sense resistor and reduces the need to have complex analog switch networks. We used this feature to our full advantage to reduce the cost and complexity. Depending on the effective number of bits set on the ADC we can achieve a maximum effective sampling rate of 11kHz, which is well within our required resolution. [40]

For our design we picked a 220 mOhm 1% sense resistor. Picking the sense resistor is a balance of how much voltage drop the system can handle and how sensitive the current sense amplifier is. Since the amplifier has a variable gain we chose our resistor to have less than 10% voltage drop of our nominal voltage at our estimated max current of 500mA. We have also placed the same sense resistor between the 3.3V LDO and the 3.3V loads, but only have test points to measure the voltage drop. We decided not to run the kelvin leads to the same INA and have a multiplexer switch between them because of the physical distance and expected noise that the lines will pick up.

To interface with the INA we placed a header that exposes VDD and I2C of the device. Since we do not want the power consumption of the INA itself to reflect in the measurements, we have a master device like an Arduino or Raspberry Pi provide the 3.3V at 1mA maximum. [40]

PCB Design

Schematic Design

Once we had an architecture and a general sense of the connections that needed to be made, we created a schematic. Multi page schematic designs can be created in two ways: flat or hierarchical. A flat design design uses multiple pages to create the circuits and uses page connectors or off-sheets to pass connections to other pages. A hierarchical design uses the idea of modules, where a user will create a block and all connections that go off page are made with ports. From there the user compiles this page, which will create a symbol which will only expose the connections intended. This technique is commonly used for designs that use modules that are intended to be reused. Since we are not concerned about reusability, we opted for a flat design for better readability. Although you do not have to label all of the connections on the schematic, we decided to explicitly label all connection on our design. This was done to make sure all connections were correctly made as well during routing we know the signal and its importance. We used a simple naming scheme where all voltage rails follow "Voltage Value_Usage", and all other signals follow "Sub-system Name_Usage". This scheme explains the purpose of the connection and what subsystem the refer to. Altium Circuit studio has a comprehensive rule checking system, which allowed us to check our design for any schematic level errors. We also used peer review to verify all signals were connected correctly and no major oversights were made.

Our design has 36 unique components and while we could have made our own symbol and footprints for each component, we didn't believe this was effective use of our time. To find symbols for all of our components we leveraged several established and reputable component libraries as well as vendor provided design files. Many of these files were used without modification, but we did end up modifying some components slightly like silk screen or via sizes. In our design we have components that we place a symbol for and route, but do not populate at the time of production. Typically these components are used for changing boot strapping or making secondary connections. These components are called "No Stuff" and Altium has a feature to not include these components in the bill of materials (BOM).

PCB Layout

Once the schematic has been thoroughly reviewed, we moved on to taking the components and physically placing them on a circuit board and connecting them. Many consideration have to be taken into account to have a successful physical design. These included: Stackup, Design For Manufacturing, Noise Mitigation, Wireless Performance, Power Delivery Network Performance and Signal Integrity. Having a successful design comes down to the initial placement of the components. Having a good placement or floorplan makes routing logical and methodical. We spent a considerable amount of time optimizing the placement of the components. We balanced the size of the board to allow for a small enough footprint to conduct physical testing, but also allow for debugging of the system with test points and headers.

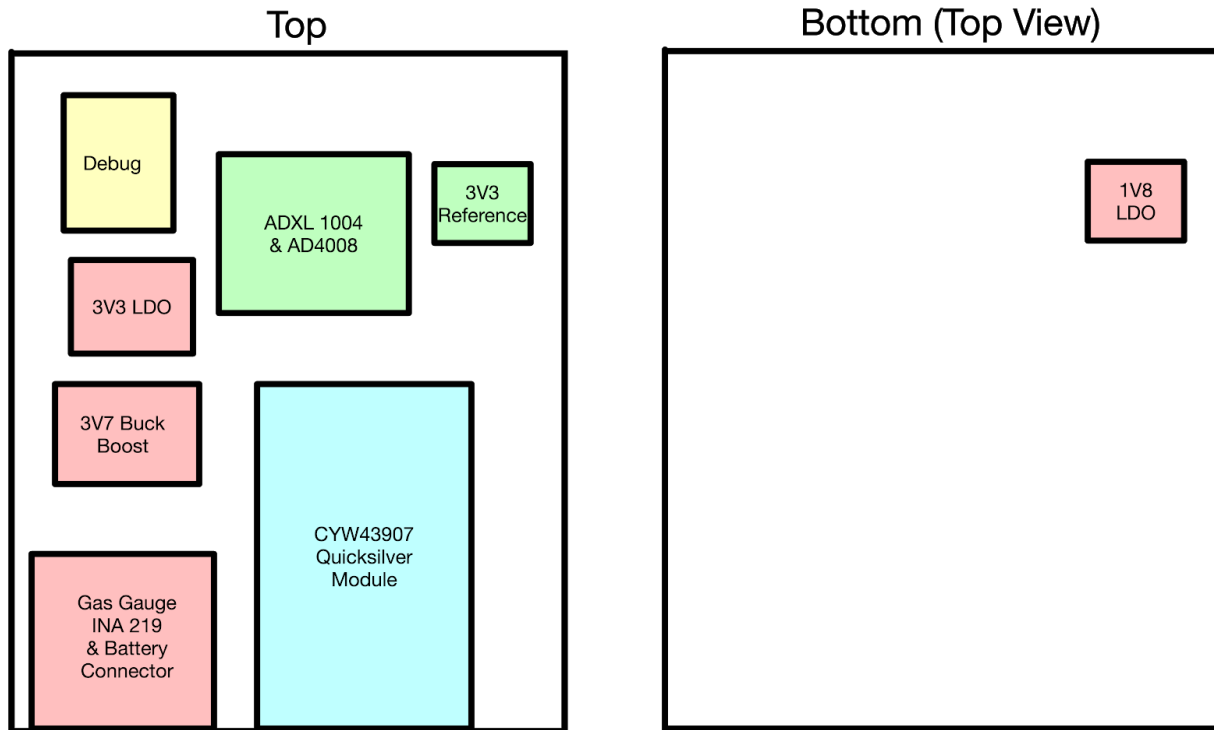


Figure 10: A top view floorplan of our PCB.

Once the components were placed, we could begin to route. We first routed all critical signals like analog signals and our SPI interface. Care was given to route with the least length and least amount of via transitions. Then we poured all power planes. Planes were used instead of traces to allow for higher current carrying capabilities. From there we routed all the rest of the IO and then finally we could pour the ground planes. We poured as much ground as possible and made all of layer 2 ground to provide isolation to our sensitive analog components. We then used vias to connect all of the ground pours together on adjacent layers.

A PCB design has many parameters that are required for design. These parameters are typically related to the manufacturing abilities of the company fabricating the design. That means that we had to have a general idea of the manufacturer that we were eventually going to work with. We extensively surveyed the capabilities of both domestic and foreign manufacturing companies to make an informed decision. Our decision was based on features that were required for our design like hole size, minimum spacing, and minimum trace size. These rules are then imported into Altium, from there we can run a Design Rule Check or DRC against our board. This is important because some vendors will not run this check themselves, and any errors not caught can cause delays as well as errors in manufacturing.

Once the design was pretty much ready for fabrication, we ran power analysis simulations on our power planes and traces in CST microwave studio. This analysis takes the PCB design including the material properties and allows one to simulate voltage source and loads. From there you can analyze the voltage drop and current density of the power planes. We designed all of our power planes using the IPC-2221 standard, which specifies equations to determine the maximum

current capacity based on the temperature rise of the copper. While our power planes are based off that specification and should be able to support our estimated maximum current, CST will tell us the voltage drop at specific pins. This is important because on our precision reference rail we want to reduce as much voltage drop as possible from the regulator to the accelerometer as well as make sure no rail droops too low during a max current transient. From the results of the simulation minor changes were made to the layout like extra return current vias and shape modification.

The physical dimensions of our completed design are 80 mm x 65 mm x 1.57 mm, which is within our original specification.

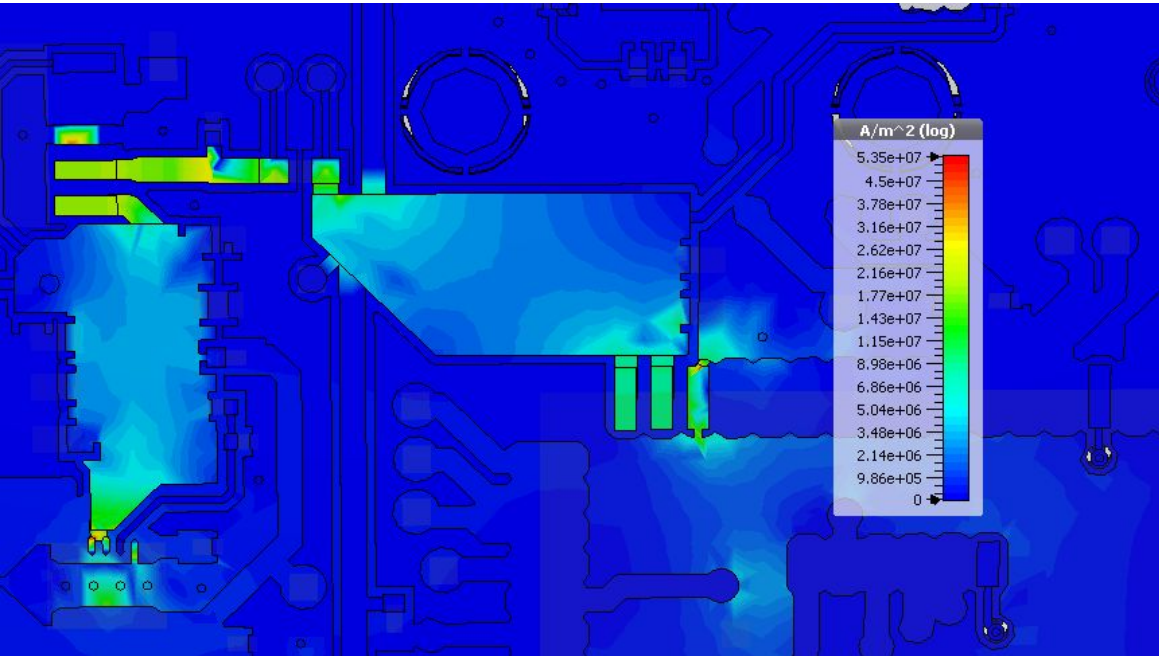


Figure 11: 2D current density simulation results for PP3V3 and PP3V7 voltage rails.

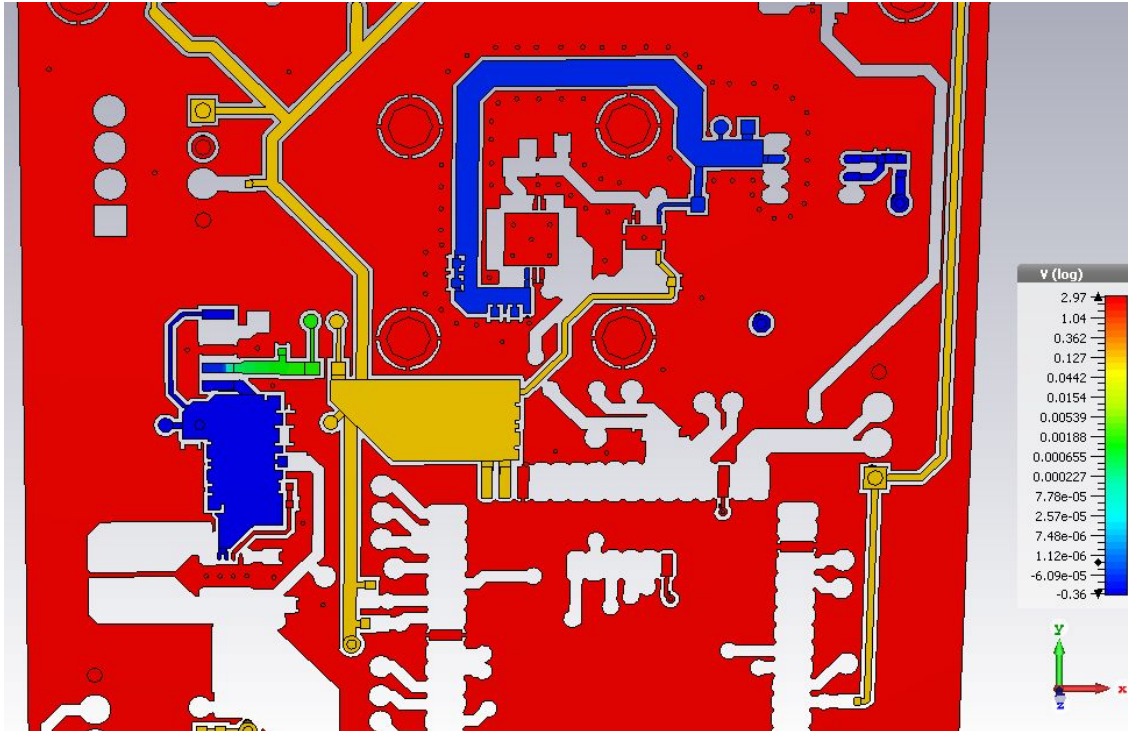


Figure 12: 2D voltage drop simulation results for PP3V3, PP3V7 and PP3V3_ADXL voltage rails.

Assembly

Once the design files were reviewed, we had to get the board fabricated and assembled. Although assembly can be done by hand, due to unexposed pins and sensitive components we decided that we would prefer to get the board assembled by a company. Since the design is immature and we are not experts in PCB manufacturing we opted to work with a vendor who would provide design for manufacturing support (DFM). Although overseas vendors will be cheaper and can provide some DFM support, we opted to work with a domestic vendor due to faster communication. We ended up choosing Sierra Circuits in Sunnyvale CA because not only can they fabricate, but they can also assemble the PCBs. To streamline the process of assembly Sierra also procured the components necessary and anything they could not order we provided (ie the Quicksilver modules).

To fabricate and assemble the design, many files are needed. Vendors work with manufacturing files compared to our schematic and board files. Thankfully, Altium has built in manufacturing output file generation, including Gerber files and many others. Gerber files are a file format that explains the board on a layer by layer basis. These layers include the physical copper layer, but also mechanical layers like silk screen, solder mask and keep-outs. The file known as a drill file provides XY coordinates of all required drill holes and denotes the size and if they need to be plated or not. These files are needed for the PCB itself, but other files are needed for the assembly. A machine called a pick-and-place will pick up the components and place them on the board in their proper location. The machine needs an explicit file to give XY coordinate of where

the part is located and the relative rotation. On top of this, a drawing needs to be created to call out all the components on the board and where their pin 1 is located. Special alignment pins were placed on our design called *fiducials* that are used by the pick and place machine to create a local coordinate system for the PCB. Altium provides a plugin called *Draftsmen* which takes your design and outputs the needed mechanical drawings. This was very useful since a manual, scale drawing could take hours of work.

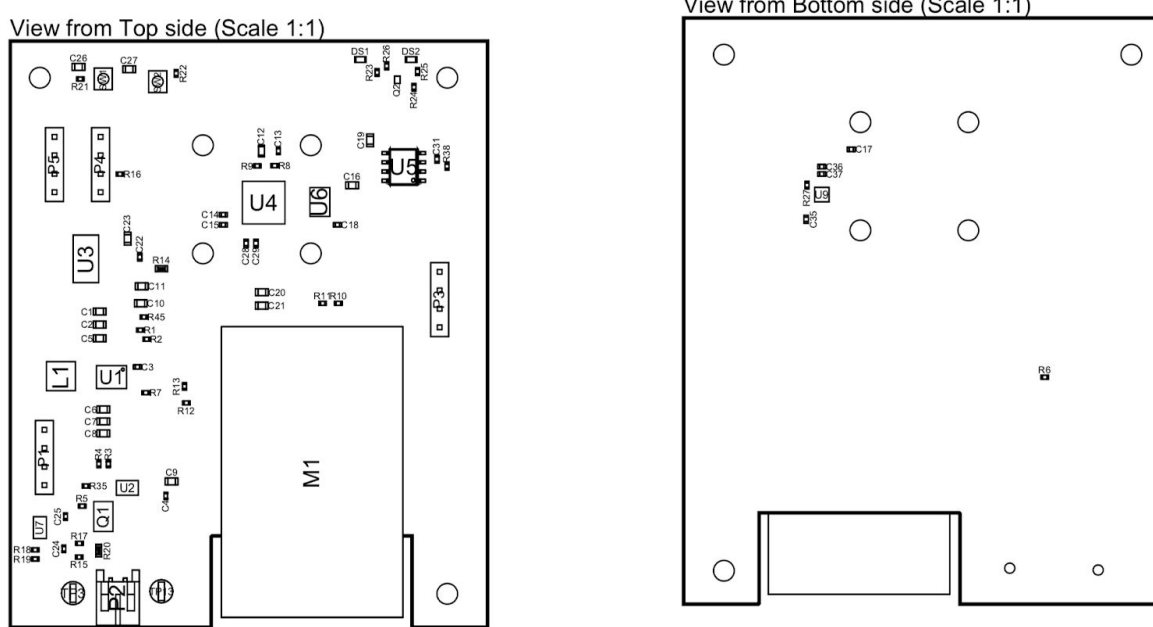


Figure 13: Draftsman drawings provided to the vendor for assembly.

Through the DFM checks provided by Sierra minor issues were found and resolved between us and Sierra engineers. Overall, working with this vendor was very straightforward and they communicated clearly what they required from us and gave us good feedback as well as being prompt. It only took five days from placing the purchase order to receiving the boards back.

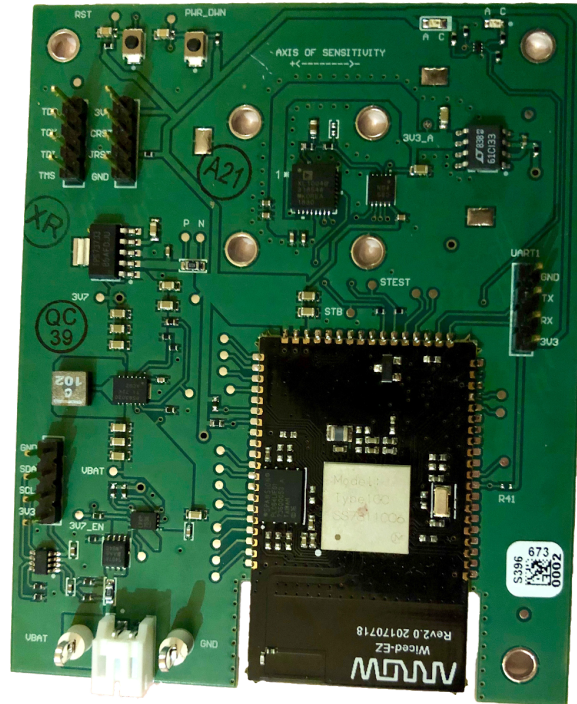


Figure 14: Image of the assembled prototype.

Testing and Results

Power

Upon receiving our boards back from the fab, our first task was to inspect their performance and verify no critical errors were created during manufacturing or assembly. This involved first checking that no unexpected short circuits or open circuits existed on critical pins of the various ICs. Upon probing the boards, we observed no obvious assembly errors, which gave us confidence to connect a voltage source and power on the board.

After applying a DC voltage to the system, all of the voltage rails appeared to be functioning stably at or close to their intended values. The 3.3V LDO, 3.3V precision reference, and 1.8V LDO were all operating nominally, however, we observed that the Buck-Boost converters were sitting a little higher than expected around 3.81V. After researching the issue further, this was confirmed to be due to the IC operating in its power save mode where the nominal voltage typically sits an average of 3% higher. A histogram taken by a high precision 6 decimal multimeter can be seen below of this rail when the system is in power save mode.

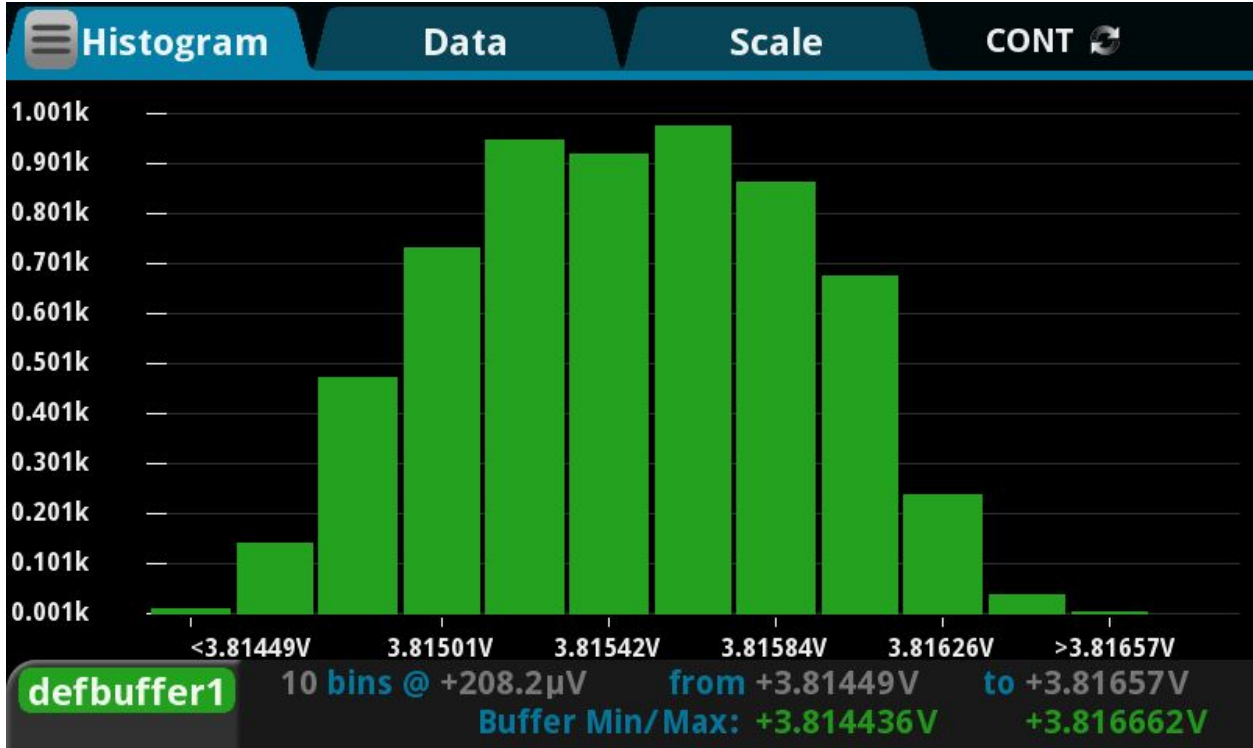


Figure 15: Histogram showing the distribution of the 3.7 voltage rail under light load.

For an expected output of 3.7V, 103% of that results in a 3.81V rail, which is confirmed by the figure. Thus, each component on the board appeared to be properly powering on, meaning that we could begin testing the functionality of different subsystems on the board.

One area of great importance that required analysis and characterization was the power consumption of our entire system. As we designed the system to be as low power and efficient as possible, it was necessary to determine exactly how much power the device uses in various modes of operation. These tests were critical given that the boards will be mounted on a spacecraft and left in a low power hibernate state for any number of days leading up to a vibrational test. To accomplish this characterization we wrote a testbench to read data from the INA219 measuring the total system current consumption in order to estimate how much battery lifetime our system would have for a given battery capacity. The INA was powered and programmed off board by an external Arduino so that the measured current values did not include the chip's own quiescent current consumption. It should be noted that all tests were performed with battery voltages closer to 3.0V as lower voltages cause the system to draw more current than higher voltages. Such behavior is expected from a boost converter. The board's power consumption was measured in three different modes, the results of which are shown below in Table 9.

Table 9: System Power Measurements.

Board Mode	WiFi	Vbat	Average Current	Average Power Consumption	Battery Lifetime**
System in Hibernate	-	3.10 V	1.428 mA	4.427 mW	24.91 days
Networked and inactive	2.4 GHz 5 GHz	3.14 V 3.07 V	154.21 mA 227.78 mA	484.2 mW 699.3 mW	20.99 hours
Networked and Communicating*	2.4 GHz 5 GHz	3.15 V 3.10 V	443.12 mA 347.97 mA	1.396 W 1.079 W	8.25 hours

* 100% duty cycle, max throughput, 20Mbps

**Assuming 2500 mAh battery

The first mode tested was when the CYW43907 is in its hibernate mode in which the device is not connected to the network and its internal power management unit is powered down. This represents the lowest power state of our system and most of the current consumption is the aggregate of all the component quiescent currents. The board was measured to consume an average of 1.428mA. However, observing the current data produced by the INA219, there were noticeable current spikes up to around 3mA occurring at seemingly periodic intervals, which exceeded the expected total quiescent currents. Investigating the issue further concluded that the current spikes were once again a product of the Buck-Boost converter's power save mode. In power save mode, the output voltage is allowed to fall before being compared against a specific reference comparator at which point the voltage pulses back up at a near instantaneous speed. Such a rapid change in the output voltage effectively looks like a step input to the system, which couples back to the input of the converter causing the current spikes observed in our measurement. They therefore represent a real current draw that must not be subtracted out from the measurements.

The second and third modes tested involved the CYW43907 being connected to the WiFi network (either 2.4GHz or 5GHz) and either idling or transmitting data. Here, idling is being used to distinguish from data transmission in which packets are being sent over the network. In this idling mode with no transmission, the system consumed an average of 154.21mA connected to 2.4GHz WiFi and 227.78mA on 5GHz WiFi. In the transmitting mode, the system consumed 443.12mA on the 2.4GHz network and 347.97mA on the 5GHz network. However, for this test a maximum transmission duty cycle of 100% was used (zero time delay between packets being sent), in order to simulate the highest possible current consumption. This meant that data was being sent at a rate of 20 Mbps, which can be compared to the rate of a functional vibrational test, where a rate of around 5 Mbps is expected. Between the maximum duty cycle and low battery input voltage applied during these tests, it is not expected to draw more current than the values provided in Table 9.

While power measurements for our system were insightful, they do not directly translate to a quantity that can determine how long our system could last after being installed on a spacecraft before requiring a recharge. Hence, a second test was created, analyzing the same three modes of board operation, that made use of the fuel gauge to keep track of the lost battery capacity during the test. Since a real 2500mAh LIPO battery was used for this test and one cannot accurately know the initial battery capacity upon plugging it in, we wrote an initial value of 2500mAh to the fuel gauge, ran the tests for a time interval of 10 minutes, and then reported the final battery capacity at the end. While the absolute battery capacity values are meaningless, the amount of capacity lost from beginning to end can be used to extrapolate how much time our system would be able to last on a fresh 2500mAh battery. Given that LIPO batteries are approximately linear over most of their operating region, this method is a reasonable approximation.

Observing the rightmost column in Table 9, we estimated an allowable battery lifetime of 24.91 days assuming the system was left entirely in hibernate and waking up every ten minutes to check if a test is ready to be run. This figure results from the very low power consumption of the module in hibernate and represents a key result from our design. On the order of 20 days should be sufficient time for engineers to mount the sensors on a spacecraft and set up any required tests.

Sensor Performance

Due to timing and logistics we were unable to complete dynamic vibrational analysis. This analysis would allow us to directly correlate the time domain and frequency response of our sensor with a reference platform like a shake table. Although we do not have a system to vibrate our board we can characterize our sensor in a stationary fashion as well as using the acceleration due to gravity.

Precision Reference Testing

As mentioned earlier the accelerometer output is directly ratiometric to the the VDD of the sensor. In conversion of the ADC samples to voltage or g value we need to use a v_{ref} value. Since we do not have a way to measure v_{ref} on each board we assume it is exactly 3.3V. This is a justified assumption due the high precision of the voltage regulator. To better justify this claim we used a 6.5 digit multimeter and plotted the distribution of voltages. We measured the rail in two conditions: hibernate and during data sampling. We decided to measure during two different modes because the current consumption on the rail is different. Figure 16 is when the system in hibernate and figure 17 is when the system is in test. We do see the rail in test mode droop slightly, but this is due the higher current consumption and is expected. Even though the delta during a test is higher than in hibernate it correlates to a worst case mV/g change of 0.1%, which we are extremely satisfied with. Note that this is a DC voltage measurement since it was done with a standard multimeter. We measured voltage ripple with an oscilloscope but ran out of vertical resolution and hit the noise floor of the device. We measured 4mV of ripple, which correlate to 1.51g of error, but we could not conclude that it was from the voltage source or the

noise floor of the scope. To better understand if this ripple can cause a measurement error we ran static testing.

Table 10: Precision Reference Testing Results.

Test Condition	Minimum Measured	Maximum Measured	Delta
Hibernate	3.300921	3.301028	10.7uV
Test (Taking Samples)	3.296497	3.296731	234uV

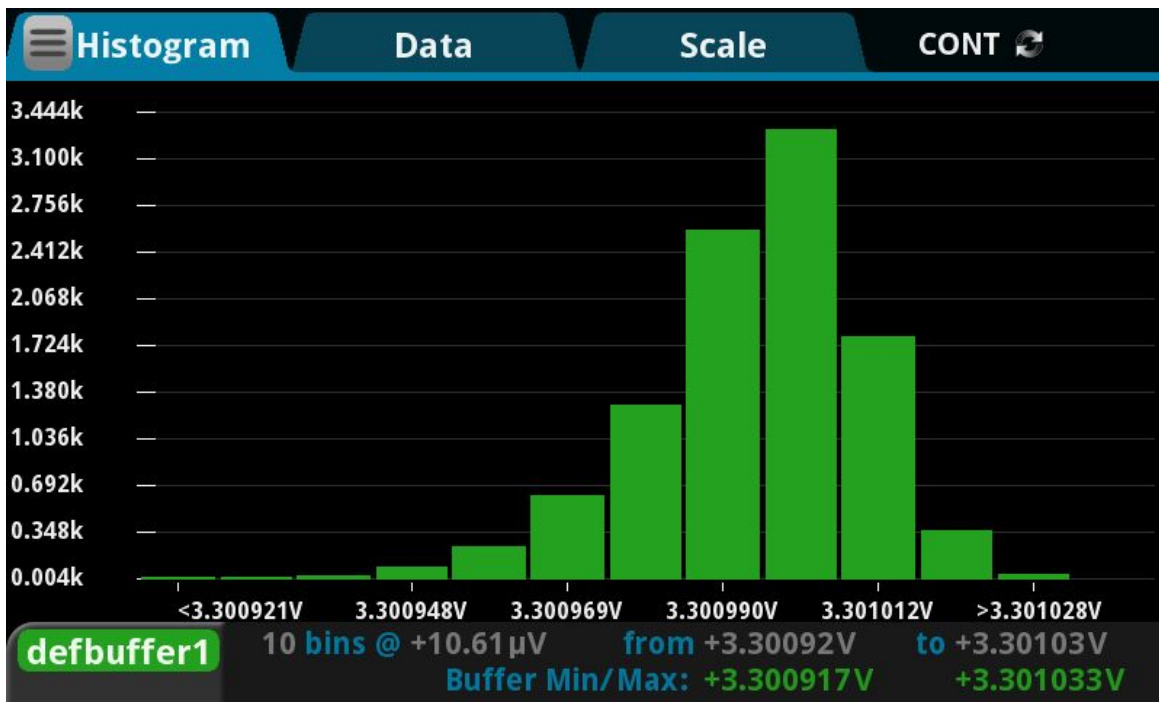


Figure 16: A distribution of PP3V3_ADXL under hibernate conditions.

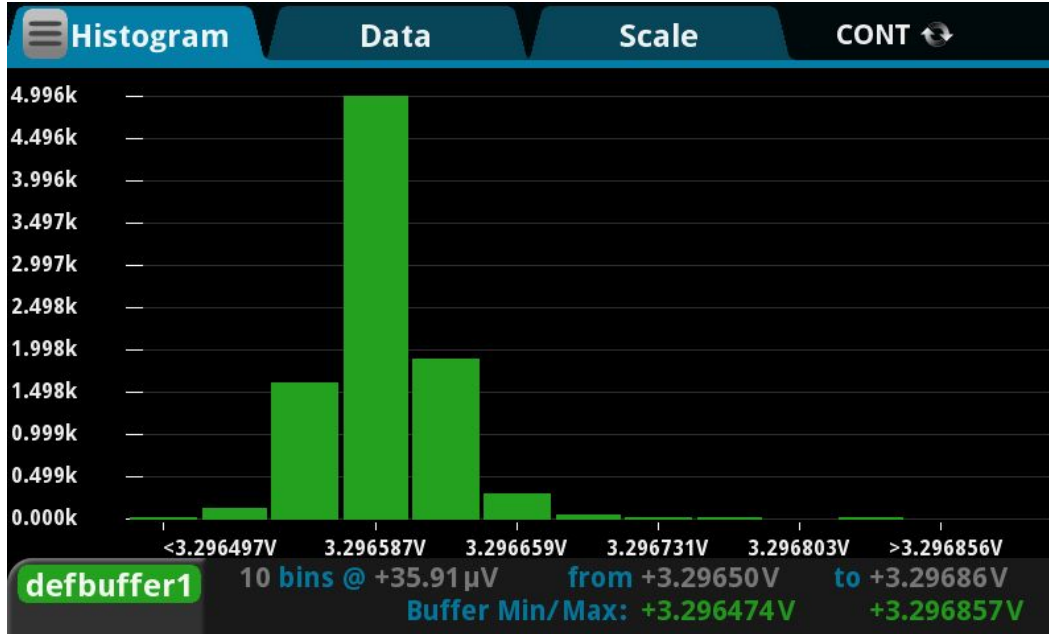


Figure 17: A distribution of PP3V3_ADXL under data sampling conditions.

Static Measurement Testing.

We ran two static tests. The first on one is to position the board off-axis and measure the acceleration, which should yield a measurement of 0g. The second test we ran is where we set the board on axis and move the device to measure +/- 1g. According to the datasheet for the ADXL1004 we can expect a normalized g offset depending on the temperature. At room temperature we can see up to +/- 2g of offset as shown in figure 18. [10] That being said we expect to see an offset upon taking measurements. Since the datasheet does not provide a transfer function for calibration we created a naive approach of calibration that will allow us to generate a calibration value for each sensor that can be then later accounted for. This calibration scheme take a series of samples for 30 seconds and then takes the average over the entire period. From there the average value is then passed to the server to be recorded. Each board has its own unique calibration value, which our post processing python script will use to correct the measurements.

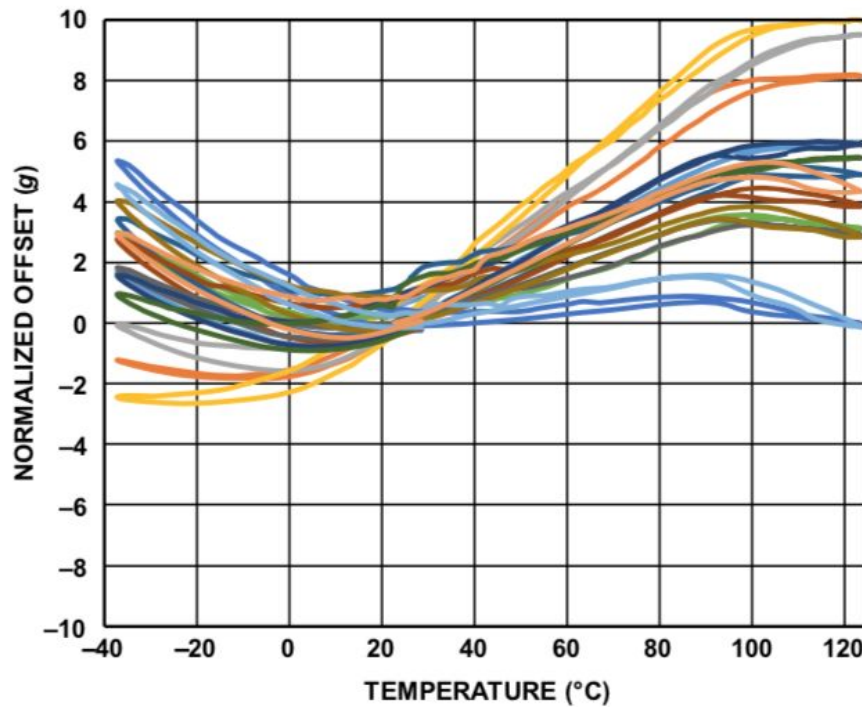


Figure 18: A graph from the ADXL1004 datasheet showing the normalized offset vs temperature.

We ran our first test for a duration 100 seconds where we have the board sitting as flat as possible on a level surface. Over those 100 seconds we take about 5 million samples, so we have plenty of samples to produce a histogram. Plotting both calibrated and uncalibrated data in figure 19 and 20 respectively, we can see that our calibration scheme is working as expected. We see the calibrated data is centered slightly off the 0g bin by 0.1g. Fitting the calibrated data to a normal distribution in Matlab we have a mean of **-0.110804** and a standard deviation of **0.102539**. We believe this deviation is due to a combination of ADC nonlinearity, cross axis coupling of the accelerometer as well as the board not being exactly level due to some through hole pins slightly propping up the board (~1-3 degrees).

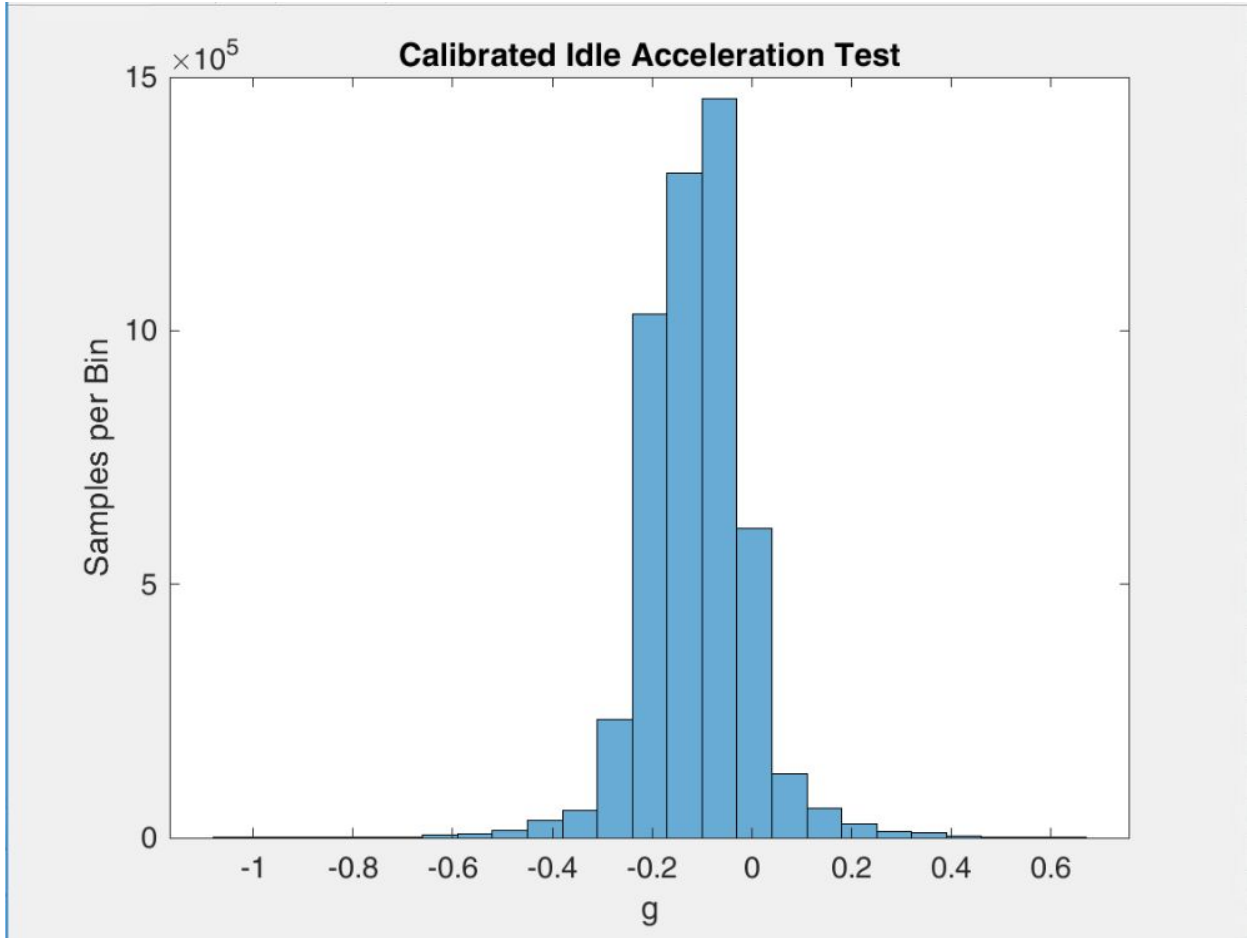


Figure 19: Histogram of calibrated idle (0g) testing of the system.

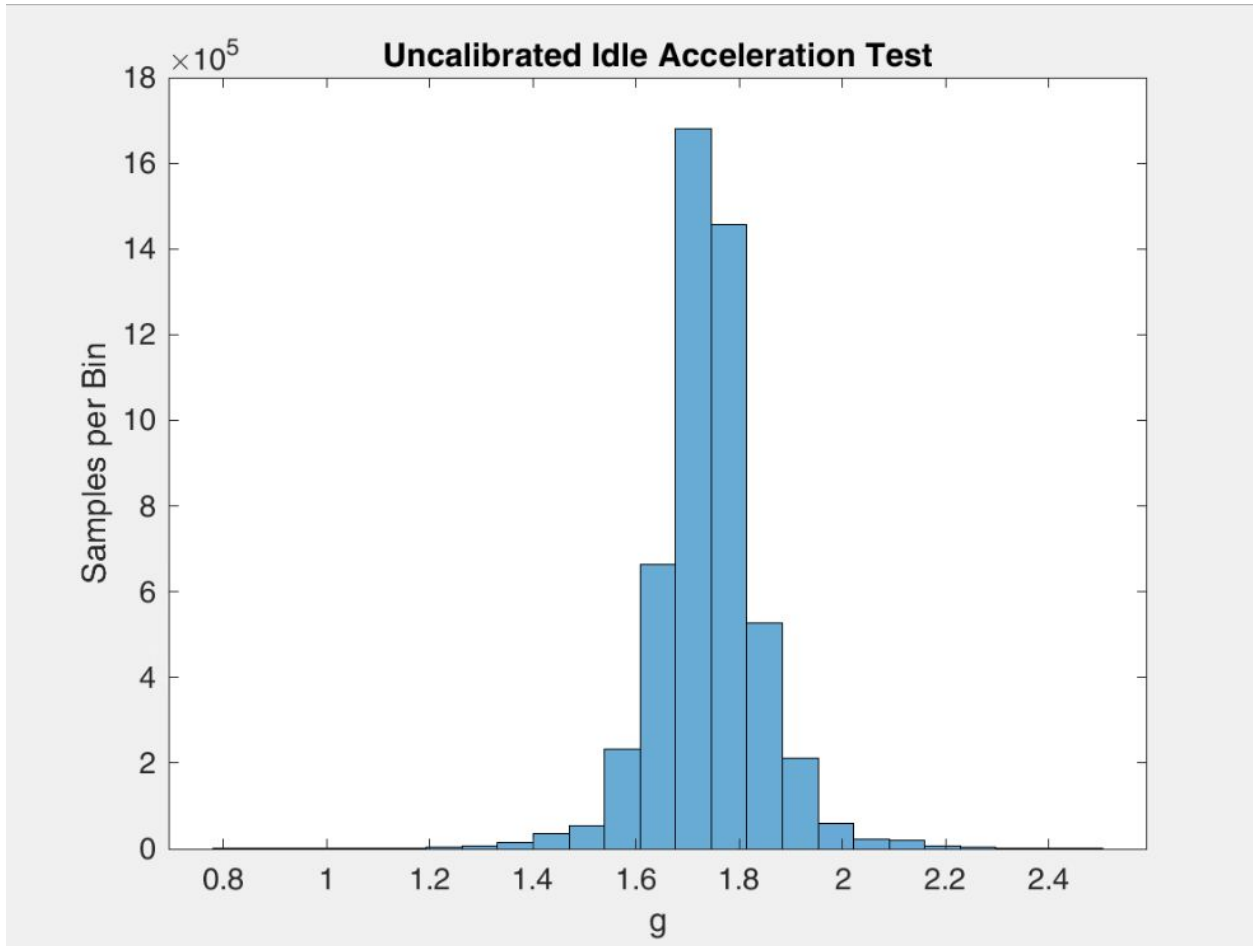


Figure 20: Histogram of uncalibrated idle (0g) testing of the system.

The second test we ran was to observe if the sensor can correctly detect the acceleration due to gravity (+/- 1g) when the board is correctly oriented. To do this we put the board flat, then orient it in the +1g direction, wait for a couple seconds, then transition the board to the -1g orientation. We plotted the time domain signals as well as individually binned the data in the +1g and -1g samples to see the distribution of measurements. Both the +1g and -1g distributions are centered to their expected value within 0.1g. We observed a -8g transient that quickly decays when rotating the board. Due to previous testing we do not believe this is due to the voltage rail noise, which can cause an incorrect measurement. We have speculated that we exerted the board to a larger g force and it correctly detected it. Unfortunately without vibrational equipment we could not further verify this transient. Overall, our verification efforts here have proved insightful and have helped verify our design efforts.

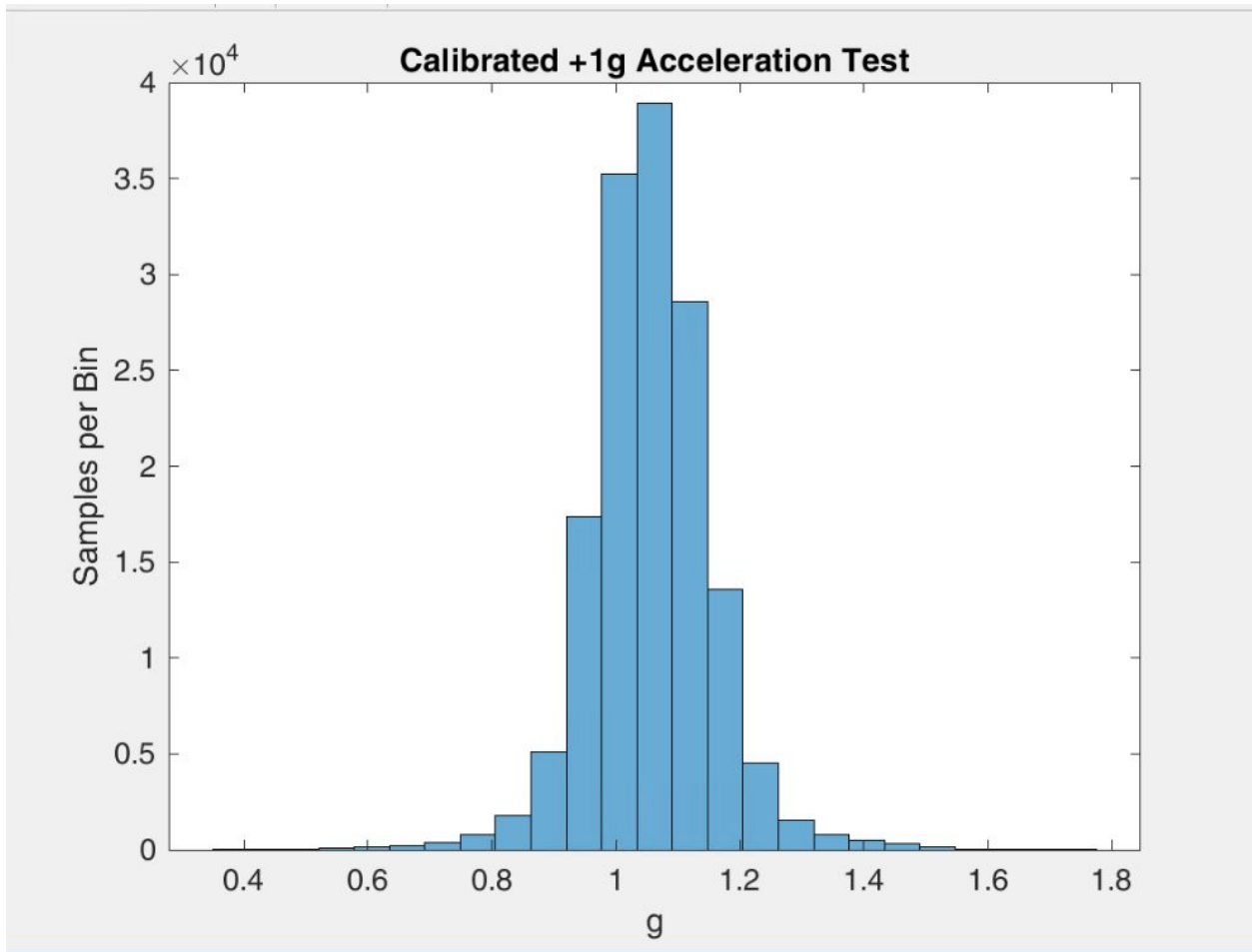


Figure 21: Histogram of calibrated static +1g testing of the system.

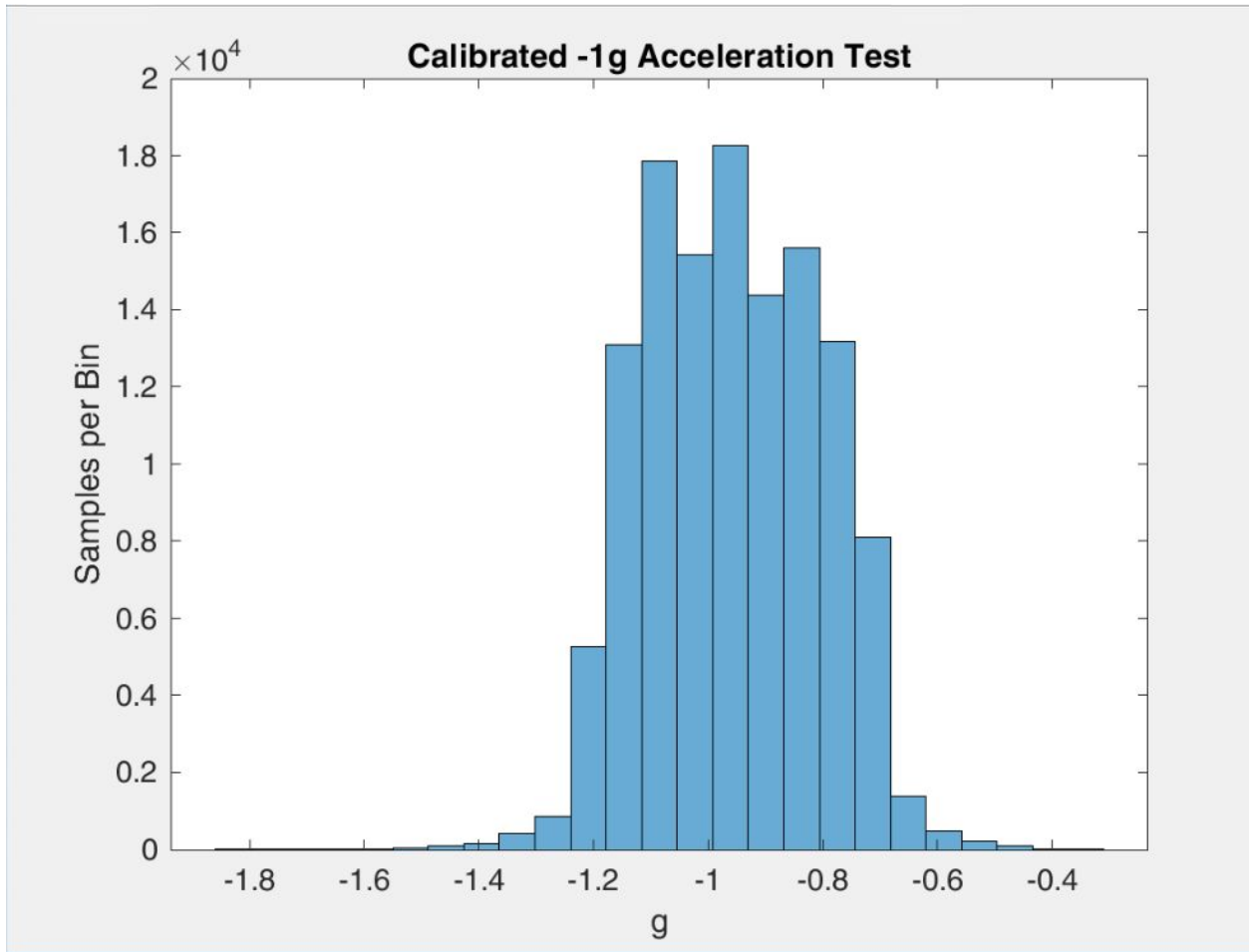


Figure 22: Histogram of calibrated static -1g testing of the system.

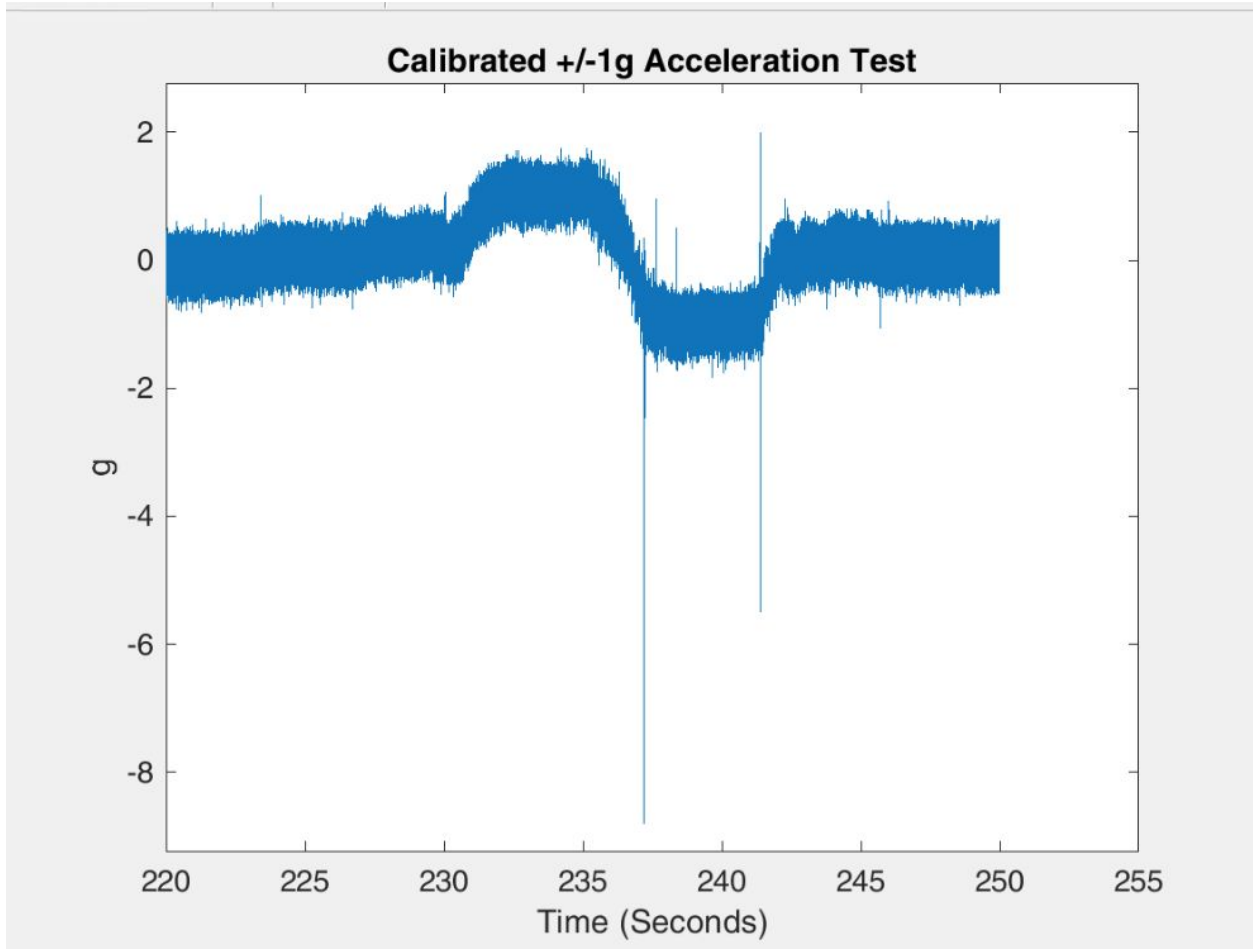


Figure 23: Time domain plot of calibrated +1g/-1g testing of the system.

Software

Overview

To take advantage of all the advanced features the WASP board provides and to meet our requirement of being a drop in replacement for an existing data acquisition system, we needed to do some significant development on our own software platform. Each individual system is focused towards speed and reliability in an embedded environment, and as such, C was the language used for all the development, both on the server side as well as the host side. On our custom board, we made use of the software development kit (SDK) provided by Cypress, WICED. WICED (Wireless Internet Connectivity for Embedded Devices) provided a set of APIs that allowed us to take full advantage of the low power WiFi connectivity and the included

ThreadX RTOS. WICED also made it possible for us to integrate some of their turnkey solutions like over the air update functionality, and made the process of programming over JTAG with our SEGGER jlink relatively easy [41].

Testing of all our software was done on the host side using the CYW943907 development kit up until we were able to confirm functionality with our custom board. On the server side, we were targeting the software to run on the base station, but as the router did not include an on platform build system, we chose to employ a Raspberry Pi 3, which has very similar compute capability as our target platform and a linux based development environment to enable rapid prototyping and testing.

Drivers

To be able to communicate with the battery fuel gauge and the ADC, we needed to implement our own custom driver software. We built each driver to be lean and efficient while providing a public high level API to the main application to simplify the code. The drivers are split into three parts, two for each communication protocol we used: I2C and SPI, and the last being the generic GPIO driver.

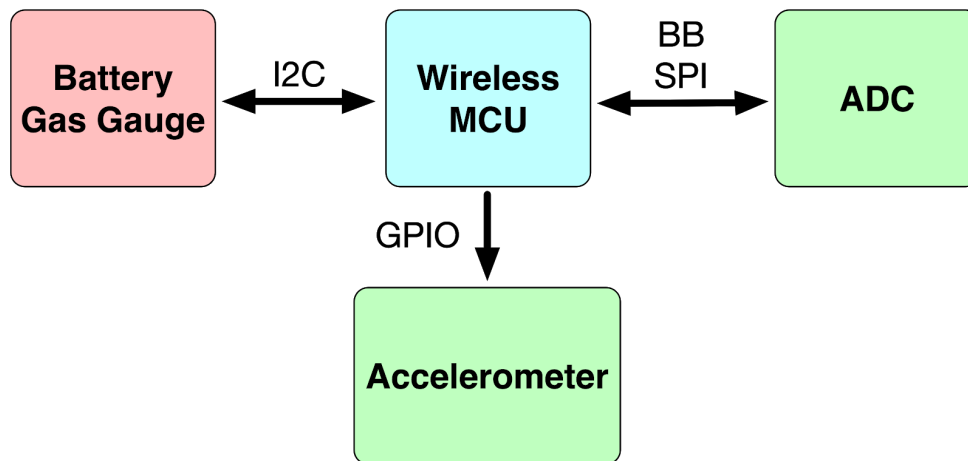


Figure 24: WASP board driver overview.

I2C

The I2C protocol is used exclusively for communication with the LTC2941-1 battery gas gauge. Much of the code is dependent on the battery being used - the battery capacity is set at build time using a #define statement.

Register Read/Write functions

The LTC2941-1 has eight 8 bit registers, the first two being status and control registers respectively, and the remaining 6 being grouped into sets of two registers to read 16 bit

quantities of battery information (accumulated charge and charge thresholds low and high). To be able to use these correctly, we provide a set of four private functions that can read and write in 8 and 16 bit quantities. Each read function takes a base register address and returns the data read, and each write function takes a base address and a data value, returning success if and only if the data read back immediately after a write is correct.

Choosing a prescaler

The first step to setting the battery information in the gas gauge is to choose a prescaler (denoted by m). The prescaler “effectively increases integration time by a factor M programmable from 1 to 128. At each underflow or overflow of the prescaler, the accumulated charge register (ACR) value is incremented or decremented one count.” [23] The prescaler is calculated by the formula in equation 2:

$$m = \frac{(128*BC*R)}{(2^{16}*0.085mAh*50mOhm)} \quad (eq. 2)$$

Where BC is the defined battery capacity, R is the resistor value (50 milliOhms in our case) and 0.085mAh is the constant LT defines as the charge (Q) LSB scaling factor. However, M can be only be one of the first 7 powers of 2 (1,2,4,8,16,32,64,128), and so to choose our prescaler we first calculate m with our values and choose the closest option from the list. Although we could use the default prescaler (128), picking a value that better suits the battery capacity will allow us to better utilize the register space. [23]

Device initialization/Setting Vbat Alert

Once the prescaler is chosen, we can then move on to initializing the device. We first initialize the I2C interface on the CYW module, followed immediately by polling LTC device. We read from the status register and compare against the default state provided by the datasheet to confirm the device has been brought up successfully. If all the checks pass, we then write the prescaler value we found earlier as well as enable 3.0V Vbat alert setting (AL/CC) to the command register. Enabling this alert will set AL/CC to drive low if the $V+$ falls below 3.0V. Finally, we calculate the proper value to place in the accumulated charge register (ACR). This is done with the following equation:

$$ACR \text{ Value} = \text{floor}\left(\frac{BC*128}{0.085mAh*m}\right) \quad (eq. 3)$$

Where BC is the defined battery capacity in mAh, m is the prescaler we determined earlier and 0.085mAh is the constant LT defines as the charge (Q) LSB scaling factor. This value is split since its MSB and LSB will be written to different registers. Once calculated, we write the values to their respective registers, completing the gas gauge initialization. [23]

Public API

The public API is simple, only consisting of 2 functions able to get the up-to-date battery coulombs or mAh respectively. Each of the functions reads from the accumulated charge registers and applies a simple unit conversion. These functions are used by the main application to monitor the battery status and report to the server or potentially disqualify itself from further activity if the battery lacks the required charge to complete a test.

SPI

The SPI protocol is used for communication with the AD4008 ADC to take samples from our integrated accelerometer. The ADC has requirements that the CYW43907 SPI hardware block was unable to support as mentioned in a previous section, so we elected to use a modified bitbang driver that emulates the SPI hardware completely from software - at the expense of some speed, added processor overhead and power.

ADC Requirements/Bitbang Driver

Cypress ships a lightweight bitbanging driver capable of running at a max speed of 1 MHz. We decided that the speed was acceptably fast based on our required bitrate, and that modifying the existing driver would be much faster than implementing our own. [33]

We had to make additions and modifications to the driver to support our ADC. The first one is to add support for conversion signal (CNV). CNV is connected to the ADC via a GPIO on the CYW43907. A simple pulse was needed, but we had to verify that the length of the pulse satisfied the minimum conversion time required by the AD4008 data sheet. [25] A logic analyzer was used to verify the pulse length. The second requirement is that in order to operate in the intended mode on the ADC we needed the Master Out Slave In (MOSI) signal to idle high. This is critical to ensure correct operation of the ADC. This change was simple once we identified where MOSI was configured in the driver initialization. A simple modification was made and verified to idle high upon power up.

Setting Mode

As mentioned earlier we need to enable High-Z mode for our ADC. This mode is enabled via a simple register write to the single register exposed on the ADC. This is done using 4 wire mode of the ADC, where CNV will pulse for the required amount of time. From there the CYW43907 passes two bytes, which is the write command (0x14) immediately followed by the data to write to the register (0xE4), which sets the high Z bit. We then wait briefly and read the register back to verify. This is done by sending the read command (0x54) and then looking at lower 8 bits that are shifted out of the ADC. We compare this value to what we would expect from our previous write, verifying that the High-Z mode has been correctly set.

Conversions

Taking a sample is very straightforward on the ADC. We pulse CNV for the time specified by the ADC and then start the SPI clock for 16 cycles, each of those being one bit of conversion data. Those 16 bits are shifted out of the ADC into the CYW43907 via the MISO connection.

Public API

The public API provided by our WASP software encapsulates all of the work of communication over SPI and setting the modes into 3 simple functions, *adc_adxl_setup*, *adc_set_high_z*, and *adc_sample*. The first function, *adc_adxl_setup*, accomplishes all of the work of initializing the SPI interface as well as setting all of the default levels for CNV and MOSI, as well as the pin modes and ISR triggers needed for the ADXL special functions, which we deal with in the next section. The other work of initialization is done by *adc_set_high_z*, where the proper mode for sampling on our board gets set. Finally, *adc_sample* does the actual sampling, returning a 16 bit sample for each call.

Additional Accelerometer Functions

Our ADXL1004 accelerometer provides a few other features that we incorporated into our driver, namely self test and standby modes, as well as a dedicated over range alert pin.

For WASP, one of the most important things to uphold is the accuracy of data. The self test mode on the ADXL accelerometer gives us the ability to check if the sensor is in working order and will provide accurate data, or if it has been damaged somehow and should be disqualified. We implement the check by taking a sample via the ADC as per usual, then setting the self test pin on the ADXL high. Setting the pin will physically displace the sensor, allowing us to take another sample and compute a voltage delta. This delta is then compared against a high and low bound which is set via a `#define` statement (for our purposes we have defined the range to be between 15 and 30 mV), and if the voltage delta falls outside of that range the part is considered not fit for use. The self test function will return the state of the part, allowing the data to be passed to the server.

The ADXL also provides a standby mode pin to send the part into a lower power mode when not in use, saving power when the WASP board would be in hibernate. Unfortunately, the CYW43907 pins tri-state when in hibernate mode meaning we can't guarantee that the sensor will stay in standby without additional hardware not present on our rev1 WASP board. Regardless, we provide the set/clear standby functions that could potentially be used in a future revision.

The last special function the sensor provides is an over range pin that is triggered when the acceleration is over 2 times the full scale acceleration output, this mechanism is used to protect the sensors internally oscillator from damage. Our driver internally attaches an interrupt to this

pin that sets a flag disabling the part until a reset occurs, allowing the device to be inspected for damage before seeing any potentially incorrect data.

Host Side Networking

For WASP, the networking code is likely the most critical function. To be a worthy replacement for a wired system, WASP needs to be able to provide high data rates with equally high reliability, as well as means to control and monitor the function and performance of individual sensors easily and efficiently. With that in mind, we went about defining our networking system from a low level.

Fundamentals

Synchronization

The first principal challenge we faced with networking was data synchronization. In a wired DAQ, all the data collection is triggered at the same time, and propagation speed of data through the wires is nearly the speed of light. In a wireless system, networking delays are introduced on top of the main challenge, that every piece of data needs a timestamp to be usable for analysis. Without a timestamp, the data received would be impossible to correlate to vibration events, and the data would then be meaningless. Knowing this, we surveyed the CYW43907 platform and found its three main timing features: a real time clock (RTC), an ISO time clock, and an internal nanosecond clock. The RTC was far too imprecise - offering accuracy only to the second made little sense with thousands of data points coming in that timespan. The nanosecond clock offered fantastic resolution, but the resolution came at the expense of it being too hard to synchronize across devices on the network. The ISO clock however, was easily synchronized across devices using SNTP (simple network time protocol). The downside however, was that the millisecond time resolution didn't offer much room for inflated sample rates if we ever wanted to increase it to support another application.

Our chosen solution ended up being a hybrid solution of the nanosecond and ISO clocks. The ISO clocks will be synced together at device startup using a local NTP server to ensure a uniform clock across all devices. The nanosecond clock value will be reported to the server as a part of device registration and stored so an accurate time delta can be calculated for all subsequent nanosecond times reported, and with each WASP packet sent both time values are packaged with the acceleration data. Finally, the clocks are frequently re-synchronized with the NTP server to correct for any drift in times possible, though we likely wouldn't see a significant amount over the course of a single testing period.

Protocol Selection

Our first and main goal with WASP was to achieve a high data rate: 40,000 acceleration data points per second per board at a minimum. TCP (Transmission Control Protocol) is the more

commonly used protocol for sending packets over the internet, but it was engineered to be reliable at the expense of faster speeds. TCP uses complex flow control algorithms which would add overhead to every packet sent, and would reduce our overall data speed. On the other hand, UDP (User Datagram Protocol) avoids any overhead and is targeted for speed and throughput, but in exchange, there is a chance packets could be lost. However, UDP is reliable enough to be the main means of sending streaming multimedia (e.g Spotify music, YouTube videos) and coupled with keeping packet sizes small, we saw experimentally that packet loss was much less than 1%, which made UDP perfect for WASP data packets.

A WASP data packet is 511 bytes in total, and consists of a 4 byte packet count (to accurately measure packet loss on the server side), 35 bytes of timing and synchronization data (27 bytes for ISO time and the rest for the nanosecond clock value) and 472 bytes of data, which are in a contiguous array of 236 separate 2 byte acceleration samples. With 472 data bytes per packet, we only incur an 8% transmission overhead.

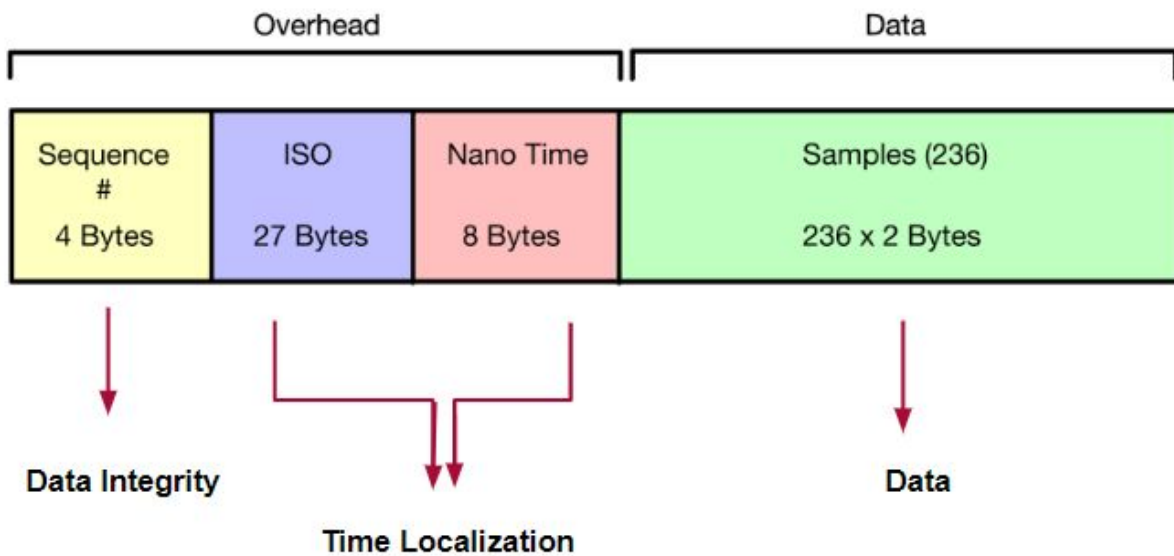


Figure 25: WASP data packet

One of our other goals for WASP was to allow for extensibility and user control through commands as well as an accurate accounting of the systems resources (battery levels, in service debug and test, OTA), and for these goals reliability of packet reception at both ends was much more important than data rate. In the end, we decided to complicate our network architecture by adding TCP communication to our in place UDP system to allow us to meet our goals.

WASP command and control packet types are more numerous - there are 3 packet types for initial device registration with the server which contain device registration data like MAC address, battery information and self test return codes, commands for further testing, and responses to specific commands, as well as 2 types for command/control requests and responses

for times of board activity. As all these packets are TCP, we could afford to be more liberal with packet sizes, though we still kept packets small for greater performance - most packet fields are 1 byte, and bit fields were avoided for simplicity of implementation.

Device Side Control description

At a high level, the WASP board will always exist in one of five possible states: Off, Powerup, Hibernate, Active and Test. Each of these modes are enumerated below.

Off

The off state is the simplest of the states: the board is powered off completely, and is without power. The only way out of this mode is through a user connecting a battery, sending the device into its powerup state.

Powerup

The powerup state is the transient state that the device enters immediately after the board is powered or through a wake from the hibernation state. At each entrance into the powerup state, the device will determine the boot type. If the boot type was a first time powerup or an exit from hibernate (non-special functionality) the device will proceed with the standard start up initialization functions: initializing all of the external ICs, performing some self test functions, setting up network functionality, and finally registering with the base station for further instruction. At the time of registration, the base station will respond to the device with commands depending on the state of the system, and either send the device into hibernate mode or active mode.

Hibernate

Hibernate is the WASP device's lowest power mode, which is critical to maintaining a long life on battery. In this mode, almost all of the external ICs will have been put into their low power or sleep modes, and the CYW43907 is off save for the small internal counter waiting to wake the device back up. Hibernate mode has a fixed duration: 10 minutes. That means that each time the device goes to sleep it will sleep for exactly 10 minutes before waking back up and reentering the powerup state. 10 minutes was chosen as it is a long enough duration to save a significant amount of power, but it is also short enough that the system can be quickly responsive given that the test command has been given - the whole system will be ready to begin in at most 10 minutes. This setting is very much related to the constraints at SSL, and as such the hibernation time can be changed at build time using a #define. It can also easily be made dynamic to fit a more dynamic setting, as we did during the initial verification phase of our board.

Active

Active mode can be thought of mostly as a transition mode - this is the mode the device is in when it is waiting for the rest of the system to come back online from hibernate to initiate an

acceleration test. In this mode, the WASP device will be performing its final pre-test checks and initializations, and listening for the test begin command from the server to send it into the Test mode.

Test

Test mode is main mode of the WASP board, as it accomplishes the primary function of our system - taking data from the integrated accelerometer, serializing it into a network ready format, and sending it off to the server, repeating the process until the test is finished. The WASP board will only leave the test mode by a command from the server signalling that the test period has ceased, and upon receiving the message it transitions back into hibernate mode to await further tests or be collected and used elsewhere.

Server Side Networking

The server is likely the most complex piece of software created for the WASP project, as it combines intricate networking code, threading, polling IO, and a complex system of management of the WASP data. Below we will walk through the server step by step.

Detailed Server Operations

Global variables

The WASP server uses global variables for the purposes of management of data and devices across threads, and having a globally visible data set allows for the thread that handles printing debug information to have data stream from each process. The first few global variables are all parallel arrays of length 300 - these serve as a global look up table for all the relevant device information like MAC address, IP address, assigned port, and file pointers associated with each device. The rest of the arrays hold data like battery levels, time of first connection, and self test information which allows us to display these data points on a per device basis to the users. The rest of the variables help the server keep track of the current mode, and the number of devices in each state.

Command Line Interface

The WASP server has a standard command line interface like that of any standard linux C program, with a set of options as well as a help page. Our CLI is implemented using the GNU standard argp set of functions. We provide four options to be used in any order:

1. Verbose (-v): This option sets a flag telling the server to enable additional printing. This option is intended mostly for debug purposes, but may be useful in other situations. The default for this argument is *false*.

2. Calibrate (-c): This option sets a flag that adds an extra step during the client registration step of each board, commanding the client to go through its built-in calibration step and reporting its result to the server for use in post processing (see *static measurement testing*). The default of this argument is *false* for the WASP team, though it should default to *true* for production use.
3. Data Directory (-d): This option takes one string argument, specifying a directory for all generated files to be placed in. If the specified directory does not exist, it will be created with 0777 permissions, meaning any user has read/write/execute access from within. The default argument is the current working directory.
4. Network Interface (-n): This option takes one string argument which is a network interface as seen in the linux program *ifconfig* (which internally uses the */net/dev* interface of the *procs*). This argument specifies the network interface that the server should listen on if WASP's service discovery protocol is enabled. The default argument is *eth0*.

```

pi@raspberrypi:~/wasp_server $ ./wasp -?
Usage: wasp [OPTION...]
WASP server -- Host code to interact with WASP modules for acceleration tests.

  -c, --calibrate           Run calibration step on first connect of each
                           board
  -d, --Data Dir=DATADIR   Custom directory for all data to be written to
  -n, --Net iface=NIFACE   Network interface being used
  -v, --verbose            Add extra debug information to print
  -?, --help               Give this help list
                           --usage           Give a short usage message
  -V, --version            Print program version

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.

Created by Cole Hunter, Tyler Hack, and Daniel Webber

Report bugs to <chunter@scu.edu>.

```

Figure 26: WASP server CLI.

Initialization

Once the arguments are parsed, the first thing the WASP server does is check the endianness of the server hardware. Because of the necessary serialization of data to be sent by the WASP boards, the interpretation of that data on the server side is partially dependent on the endianness of the server side hardware. If the server detects its running on a big endian machine, it will throw an error message indicating the problem and abort.

The WASP server then begins by creating a file called “mode” which will serve as the means by which an engineer can change the testing mode asynchronously. The mode is initialized to be hibernate, meaning all devices will be immediately sent to sleep to preserve battery life. If the

calibration argument evaluates to true, the server will also create a file called “calibration.data” which will store all of the device calibration values and associated MAC addresses.

Next, the server initializes a thread whose sole purpose is to print data. The print thread will print useful debug information and the current server state to the user, as well as a formatted table showing all of the connected devices and their information (MAC addr, battery level, self test status, and time since the last hibernate wakeup) which comes from the global variables described above. Additionally, if the verbose flag is set, the thread will also list some information about the running threads, along with some other debug information. The thread will update the data on the console once every 5 seconds so as not to consume any resources that are better allocated to data reception.

```
pi@raspberrypi:~/wasp_server $ ./wasp
listening for connections
Current Mode = Hibernate
Devices ready to test = 0
|Client |MAC Address      |Battery Level %|Self-Test  |Time Since Ping |
|0      |00:a0:50:c4:26:00|100            |PASS       |00:24            |
```

Figure 27: Output of the print thread for 1 device.

Once the server finishes creating the print thread, it zeros out the data in the lookup tables, and begins the TCP server functionality.

TCP server

The WASP TCP server has the responsibility of listening for and responding to WASP board registration messages. At each message reception, the server will decide whether or not to send the board back to sleep, to fetch an update, or optionally to initiate the calibration step if the flag is set. To do this, the server first creates a TCP socket on port 50005 (TCP registration port in the WASP protocol) and begins to listen.

As mentioned above, the WASP server needs to listen in for TCP registration packets, but it also needs to keep track of the time since the last registration for each board for display to the user. This creates a problem because accepting a connection from a WASP board is a blocking operation, meaning we cannot do anything until a packet arrives. To get around this, we use a form of polling IO provided by the Linux OS via the *select()* function. Using *select*, we can define a timeout of 1 millisecond for the packet reception, allowing us to only accept connections when one is ready to be read. At each loop iteration, we update the list of file descriptors to be monitored (which is just our TCP socket in this case) and we call *select*. If a connection becomes available for reading, we can service it, otherwise we can go about the timekeeping necessary to keep track of the boards.

If *select()* flags that there is a connection available, we accept the connection and spawn a new file descriptor for the connection to be serviced. We can then read the registration packet and

begin processing. Once the packet is read, we first look through the lookup table to see if the board has previously connected and is already in the table. If it exists in our table, we simply update its information and send it a command to sleep or get ready to test based on the current mode. If it isn't in our table the device is connecting for the first time - its information is first recorded, then it is assigned a UDP port for sending data during the test before the commands are sent. If the calibration flag is set, the server will spawn another thread tasked with asynchronously receiving the calibration value from the WASP board and writing it to the calibration file. In either case, once the commands are sent and everything is received, the file descriptor is closed (as the TCP connection is no longer relevant) and the loop will continue.

If select times out on the other hand, the server first checks its mode, immediately continuing on to the next iteration if the server is still sending boards to hibernate. However if the user has chosen to edit the mode file to signal a test cycle to begin, the server will start a timer for 12 minutes. The purpose of this timer is to allow for all the devices to come back from hibernate and from the logic above be sent into active mode to await the test. 12 minutes was chosen instead of 10 because the devices need some time to connect to the network, which may take a moment. Once the timer begins, no new devices are allowed to register with the server, so the number of devices we are waiting for is fixed. If that number is reached before the timer expires, the server will proceed on to the next stage. If the timer expires and the number of devices is less than what it should be, the user will be prompted as to whether they would like to continue.

Once we have moved past the 12 minute timer, we move to another short timer, this time only 2 minutes. The purpose of this timer is to allow the devices to start and complete their NTP sync. We have found experimentally that the time taken to sync time across all the devices can take anywhere from a few seconds to a little over a minute, but having the timer set at 2 minutes gives us some margin of error so that every device can sync up before the test begins.

Once the devices finish their sync, they will begin waiting for the test to begin, and the server will spawn the UDP data reception threads. The logic is such that only 1, 2 or 3 threads will be created if only those number of devices are connected to save resources if they aren't needed. Once the threads have been successfully created, the main thread will sleep for a few seconds to allow the reception threads to complete their initialization, then it will jump into its UDP broadcast function.

Broadcast UDP

Once the wasp boards have all been synchronized, they will begin to wait for a test start message. However, we want all of the boards to begin to take data at the exact same time (to the best of our abilities) and sending TCP or UDP packets through regular means will introduce a slight delay as it sends each start packet one by one. A better way would be to send a single packet to all the boards simultaneously through a broadcast message. While TCP would be preferred to guarantee the packets will be received, TCP is unicast only, so instead we must use UDP.

To send the broadcast message, we first create a new UDP socket like any other. On Linux however, sockets require special permissions to broadcast. To set the correct permissions, we use the *setsockopt()* system call to set the broadcast permission bit. We can then change the mode to test active by writing the mode file created earlier, and sending the message to the broadcast address (255.255.255.255) on the specified WASP protocol port of 50006. For a normal packet the data being sent would matter, but in our case the WASP board considers the reception of a packet with any data to be a valid test start signal.

Using UDP inherently means that there is a chance that the test start signal may not reach the destination. However, in a closed system (the intended use case of the vibrational tests), the network would have no activity for seconds prior to the signal being sent, eliminating any chance of collisions. Despite that, we choose to be very cautious, and we send the test start signal 10 times repeatedly to ensure that the signal is heard by all the devices.

UDP reception and Data storage

Once the main thread spawns the UDP reception threads, they all will immediately begin to go through the initialization process to become ready to receive data. It's important to note that the main thread spawns each UDP reception thread with the same main function, meaning that they all do the exact same operations, the only difference being the argument to the function - the port that thread is responsible for.

Each thread first begins initialization by setting up local copies of the information contained in the global look up table. Because each thread is running on its own core, there is a possibility of contention between two or more threads wanting to access the same data at the same time. By creating a local copy, we can avoid implementing any complex methods to avoid that contention and also benefit from the speedup. Each thread will make a local copy of the MAC and IP addresses while also maintaining the global index of each entry so it can update the global tables when necessary. These arrays are all of length 75, as each thread will be responsible for one quarter of the total devices up to a max of 300 as set above. To make the local copies, the thread will scan through the global array and note which devices have the port assigned to the current thread, populating the local data as it finds them.

In addition to making local copies of the MAC/IP addresses, the thread will also create a file for each device assigned to it. WASP takes the approach of having a single file for all the data of a single board so that data can more easily be attributed to boards, and therefore locations on the satellite as the locations are fixed. The file names are created as the MAC address of the board to aid with that correlation. For each device found, the thread will turn the MAC address into a string, create a CSV file for that MAC, and write the CSV header. CSV was chosen for simple and fast post processing that will work in almost any software. The file pointers are then updated to a parallel array with the MACs and IPs.

Once all the required files are created, the thread can then prepare to receive data by creating/binding a new socket, and entering the receive loop. The receive loop simply receives a packet and the IP of the sender, uses the IP to find the correct file to write to from our local array

of file pointers, and writes the data to the file. The files aren't open and closed in the receive loop as the open and closing operations have a non-negligible delay that could incur packet loss. Instead, the files remain open for the duration of the test, and the buffers are flushed after each write, effectively writing the data to the file without closing it. By flushing the write buffers, we can avoid the delays of opening and closing the file, while also saving gigabytes of memory. Flushing also gives us the benefit that if the user were to quit the server unexpectedly, the data would be safe, and the files would automatically be closed by the OS when it cleans up the open file descriptors on exit. The receive loop will also check the mode the server is in at each iteration, breaking from the loop and closing all the open files if the user decides to end the test (by updating the mode file). The function will then return to the main, and once all the threads have exited the main thread will exit, terminating the program.

Once the WASP boards receive the broadcasted test start signal, they will immediately begin sampling data and sending packets. By the time the test start signal is received, all four threads would have successfully initialized and will be waiting for packets.

TCP Asynchronous Client

In addition to UDP functionality, the WASP server also provides a TCP client to send commands asynchronously to the WASP boards during tests. Currently the WASP boards listen for TCP commands but take no action on the reception - this feature was added in case the need arose to send commands mid-test, but currently we don't use it. In any case, the WASP server will spawn a thread to send TCP commands as soon as the broadcast signal is sent, but no actual sends take place. We leave the logic of when to send up to any future party interested in using this feature.

WASP SDP

As we were undergoing the lengthy process of qualifying our system, one step stood out as a great annoyance - setting the server address. In the original version, the WASP server needed to be set up before anything else so its IP address could be found, and subsequently hard coded into the WASP board firmware. This process was not ideal, and we found that changing location to do tests meant that a new IP needed to be programmed into all of the boards which was a very time consuming process. To address this, we implemented our own simple version of a service discovery protocol (SDP).

Our SDP implementation makes use of the same type of broadcast messages as the test start commands to communicate with an unknown server. On startup, the WASP board will send out a broadcast message over UDP to every device on the network on port 60007. On the server side, a thread is running a dedicated listener, which will determine its own IP address on startup, and when it receives the broadcast message from a board, it will respond with the address as a period delimited string. The WASP board can then turn this received address into a binary IP address,

and continue its startup procedure. To enable the protocol, the user must build a version of the server with SDP added.

OTA Subsystem

WASP has an interesting functional requirement in that once the boards are placed, no human will come into contact with the device for potentially many weeks. This meant that we had to design the board to operate at very low power levels, but it also presents another interesting challenge - we recognize that we may want to fix bugs or even quickly add features, and we need a complete method of doing so that will work reliably.

The WICED software provided an example implementation of an OTA (Over The Air) update application which used the CYW chipset ability to become an access point to push updates, all of which was built on top of HTTP. While some of the features provided were applicable to us, the overall flow of pushing updates wasn't desirable. Instead, we opted to build our own pull based system on top of the a stripped down version of WICED's OTA subsystem.

Our version of OTA is simple and is designed to require almost no interaction with the end user. Each build of the WASP firmware is associated with a version code consisting of a major and minor release number (ie 1.6). The most recent firmware build is placed on the WASP server and a corresponding file is updated with the latest version number. At registration, the WASP board will receive the version number of the firmware that has been placed on the server. The device will then check this number against its own number, and if the server number is greater, it will request the filename of the updated binary firmware file, reboot into the OTA update mode, and proceed to update itself. In the update mode, the device will query the HTTP endpoint it received from the server to download the file. Once downloaded, the device will reboot, where the bootloader will take over by extracting the new firmware into a staging area of nonvolatile memory, replacing the old firmware with the newly extracted version, and finally rebooting itself into the updated version. The newly updated device will then proceed like normal as described above.

Final Cost

We received \$1500 from the School of Engineering as well as \$2900 from Analog Devices. Both of these required grant proposals of why our project aligned to the ethos of both organizations. Thankfully we didn't not have to purchase any test equipment like multimeters or oscilloscopes since the senior design lab had adequate material. We did use some of our own devices like programmers, a logic analyzer, SMD rework tools, a microcontroller, and single board computers, but we didn't get reimbursed from the school since we intend to keep them. We made sure that funds were spent appropriately and effectively. For example we were able to direct Sierra Circuits, the manufacturer of our hardware to procure the components from a specific

website, which we determined was the lowest cost. In table xxxx we have rolled up all the expenses that were covered by the two grants.

Table 11: Project Costs Paid By Grants

Category	Description	Cost
WASP Custom PCB	Cost of fabrication, BOM, and design tools for 5 boards	\$3348.56
Evaluation Kits	ADC Eval Kit	\$312.32
Total	-	\$3660.88

We also considered the cost of each WASP Board at an expected volume of 1000 board. Building at a larger volume allows for cost breaks from the BOM as well as manufacturing cost. It is very common for prototype and low volume boards to cost multiple times more than the device once in volume production. We ran quotes using the expected volume and accounting for the BOM costs, manufacturing, and material costs, we can expect each board to cost \$100 or less. Considering just the cost of the sensors that are currently used which cost upwards of \$1000 each, we see a cost reduction on the order of 10x.

Social Analysis

Ethics

Ethical Justification

Space exploration is undoubtedly going to become an even more frequent endeavour for companies, organizations, and academic institutions. This is because not only are the missions mysterious and enticing from an exploratory point of view, but because it will continue to lead to the advancement of new technologies and open doors for new research. Yet, for most curious individuals or scientific groups, space exploration remains largely inaccessible due to the sheer cost of planning, testing, and executing a launch. The effect of this is a gap in knowledge that forces those without the necessary funds to make scientific hypotheses and conclusions based off of another organization's space mission, and not their own. Given that many missions are funded by governmental organizations that have their own agenda and dictate the purpose of the mission, there is so much information out there to learn and research to be conducted. This is ultimately a missed opportunity where sufficient funding simply does not exist.

As a result of this inaccessibility, the question must then be raised as to what ethical action can be taken to open up space exploration to a broader audience of both intellectual and curious

individuals. The Markkula Center for Applied Ethics claims that “ethical actions treat all human beings equally - or if unequally, then fairly based on some standard that is defensible.”[42] This argument leads to the general idea that people who work harder or contribute more value to a group or cause should be rewarded more for their efforts. However, what happens when two equally skilled scientists or engineers work on a space exploration project yet one of them works for a government-funded organization while the other for a smaller university? It quickly becomes apparent that rewarding the former employee more just because of their involvement in a group that possesses funding to actually execute a launch does not feel like the “right” or more ethical action to take.

As the engineers behind this project, it is our belief that cost, manpower, and setup complexity are the three main roadblocks in providing accessible space exploration to a larger audience. In order to justify the cost of building a spacecraft, it must obviously be tested extensively to ensure that the craft will not fail during the mission. With the current method of vibrational analysis testing the sheer complexity and time required for the setup itself is enough to give only the largest and well funded organizations a shot at succeeding. The objective of WASP is to deliver a lower cost, small form factor, and highly configurable device that can augment or replace the current wired testing equipment in a manner that is easy to install and just as accurate in regard to measurement precision. By doing so, we hope that this module will allow smaller groups of researchers to invest in spacecraft test equipment while only needing access to a computer and a WiFi router instead of a state-of-the-art control room. In addition, by adding software functionality and user-friendly interfaces, the engineers will be able to setup and conduct the tests in a quicker time frame with fewer people involved. Thus, space exploration may therefore become more accessible to all and mitigate the issue of lack of funding or regulatory bodies dictating the outcome of a exploratory or research mission.

Engineering Virtues

The ultimate goal of our project is to produce a system to assist with vibrational testing of spacefaring vehicles. As we are designing the system to replace current wired and bulky measurement systems, our sensor module must be small enough to mount of a satellite without adding to the system load, low power enough to permit long testing times without requiring a battery recharge, and use as few components as possible so that the new system is less expensive than the current wired system.

Therefore, in regard to the frugal innovation constraint of manufacturability, our system accomplishes this by trying to minimize our component count and eliminating unnecessary functionality that is not integral to the vibrational analysis. The design will be easily manufactured in large quantities and result in a final form factor that is on the order of a few square centimeters. Our decision to manufacture the boards locally in the US was chosen as a means to facilitate better communication with a PCB fab, ensuring that the manufacturing process is well understood and significant collaboration exists between us and the fab house.

In regard to usability, we want to ensure that our design is easily implemented by other engineers. The amount of labor required to both install and debug our wireless system must be less than that of the current system or else our design cannot be justified as a suitable replacement. We are ensuring that there is both physical mounting capabilities, software functionality, and long lasting battery lifetimes to try and offload some of the arduous user tasks that are required of these systems. As engineers from different technical backgrounds will use our system, it must function in parity with all of their skills giving access to different kinds of users.

Safety

Our physical system poses little to no threat to a user. Steps were taken in the design process to ensure that the system does not pose a risk to itself as well as the user. Starting from the input to the system we have reverse voltage protection. Even though our battery connector has a keyed connector we placed a MOSFET to prevent any reverse current from flowing into the board if the battery is connected in the wrong direction. The most significant safety risk in our system is the Li-Po battery itself. One issue is these batteries have to operate within the specified limits. We have properly spec'd our battery to ensure our maximum current is within its' recommended operating limit. Operating voltage needs to be maintained or the battery can damage itself as well as possibly combust. Ignition of these batteries is especially dangerous since these can trigger thermal runaway situations where the battery will burn hotter and hotter until the energy stored in the battery is depleted. The gas gauge on our system protects the battery, which will cut off the battery from the rest of the system once the voltage limit is met.

Another safety consideration we have examined is the wireless emissions from the WiFi radio. While WiFi does not transmit ionizing radiation, limits are still in place by the FCC and the FDA for transmission power. [43] In the United States all devices are evaluated and approved by the FCC to ensure the transmission power is within legal limits as well as within the specified frequency band. The Quicksilver module as well as the LBWA1UZ1GC have already been certified by the FCC. This ensures that our device does not pose a threat to humans as well as interfere with other devices.

Risk

Once our system is fully functioning, the intention is to replace the currently working system. These tests are used to qualify spacecrafts against the design specifications needed to ensure a successful deployment. The risk our project presents is that if one were to use our system to qualify their system and our platform has an issue that presents erroneous data, an incorrect decision could be made on the bad data. That could mean putting a spacecraft up in the sky that was damaged during flight and does not function or something more catastrophic like affecting the performance of the rocket itself. The scale of the risk is something we have not taken lightly. To ensure proper operation we have spent significant amount of time engineering a robust system as well as validating critical components of the system affecting measurements. Although we have not been able to complete dynamic vibrational analysis alongside a reference sensor, this test is something we have made clear to our collaborators and advisors that this is necessary to consider our system as a functioning and safe alternative.

Sustainability

Social

This device is intended to simplify the lives of engineers interested in performing vibrational tests on spacefaring vehicles. Such tests are necessary to ensure the safety of the vehicle during high force events such as takeoff and re-entry. We want the installation process of these modules to be easier, less bulky, lightweight, and easy to debug in the case of a device failure. We also hope that this system can be modified and potentially used for other applications with different types of sensors. One example could be in medical applications to capture physical motion or perhaps even vital health data from a patient to gain more information on the nature of their condition.

In addition, although it may appear abstract, our project's outcomes have strong ties to Social Equity. Satellite development even on the academic level is plagued by the high cost of not only materials to build the satellite, but also the test equipment. Test equipment costs are extremely high but are needed to ensure a successful design. As a result, this only allows well-funded universities or corporations to build and test satellites. This creates an unfair situation where only certain experiments can make it to space. With our test system, the costs are significantly lower than the conventional system. This will allow for any university regardless of major funding to test their satellites. We believe that any university regardless of size and funding should be able to build and test satellites for the further advancement of knowledge.

Environmental

In our design project, one of our main goals is to deliver a module that is completely wireless and portable, and consequently we need a battery to deliver the portable power necessary to make our design work. We chose to use a LiPo battery due to its best in class energy density, but this choice has a significant impact on the sustainability of our project as a whole. LiPo batteries use the mineral Lithium to achieve their level of performance, and lithium production can be a dirty business. The mining process involves pumping salt water deep into the ground to pick up the valuable materials, and then the resultant lithium rich water is left to evaporate leaving the mineral behind to be collected for further processing.[44] This poses two problems: the mining process is an incredibly energy intensive process, but even worse, the mineral rich water is prone to mixing with groundwater or other natural water resources, which can be very problematic for the neighboring environment. Furthermore, the main production of lithium occurs in south america, thousands of miles away from the processing plants which are primarily in China.[44] Due to the distance and nature of the material to be shipped, heavy cargo ships are the main means of transportation of the raw lithium. The shipping industry is notoriously the largest producer of greenhouse gasses, making the transport of lithium a large greenhouse gas producer with an equally large carbon footprint.

As with any other design project containing anything more than the most basic electronic circuits, our module will feature many capacitors. Projects like ours also have an additional constraint on their choices for passives: size. Picking the smallest capacitors can help us save money and space, but that choice has a downside - many of the smallest capacitors utilize the material Tantalum for their capacitive material. Tantalum is a particularly devastating material for the communities in Africa where it is predominantly found, and due to the ever increasing demand, the damage continues to get worse. From an environmental standpoint it isn't much better. Like the lithium described above, tantalum is processed great distances from where it is mined, meaning that the process of shipment incurs a great greenhouse gas penalty. The processing is much the same - due to the strength of the material, it requires heavy processing requiring a multitude of strong chemical processes. Each of these steps in the path to making a refined product like a surface mount capacitor is very harmful to the environment, both due to the greenhouse gases (that are stronger than CO₂ in many cases) and the high energy cost to sustain intense heats and pressures where the reactions take place.

Researching if there were any battery alternatives more sustainable than lithium batteries was a challenging task. In spite of its issues from a sustainability point of view, it offers an incredible value to us due to its small size and high capacity. We were reluctant to make any changes, but we decided to look up some potential alternatives. What we found was actually surprising: compared to other rechargeable batteries (obviously disposable batteries would be a worse option due to the high waste) with high energy densities, Lithium was among the best options. In place of a change though, we still have a plan for improving our sustainability situation with regards to lithium. Our main energy consumer in our project is the processor module on the board, and we have complete control over what it does at any time via our code development. By optimizing our code for better efficiency and maximizing time spent in low power states, we can effectively reduce the size of battery we need and therefore reduce our lithium consumption. We believe this to be the best means of change for our project.

The choice regarding how to minimize our impact through tantalum was much easier compared to the plan we had to devise for our lithium use. Ceramic capacitors offer many of the same benefits as tantalum caps: small size, good performance, and ceramic capacitors aren't polarized meaning less chance of error if we ever needed to hold solder any. Ceramic capacitors also don't explode if they are overvolted - this wouldn't be an issue in our design due to proper tolerance parts being in place, but it is a nice added bit of safety for a project that deals with expensive equipment. The downside to ceramic capacitors is that we could possibly pay slightly more, but passive components are by far the cheapest parts on our board, and the substitution won't cause a significant change to the overall price.

Economic

Economic development has been the biggest social factor in shaping our project. Its impact comes on a few levels - firstly, we wanted to make our physical components as low cost as possible while still ensuring maximum reliability for our users. This is because of two reasons: as students, we are on a constrained budget, and we cannot afford to be careless with our funds.

Production and assembly cost is high for us, limiting the amount of things we can do. We have to be very careful not to make any mistakes, because we likely only have a single try to produce an working design. We also have to carefully choose components, technologies, and methods that give the most performance per dollar to maximize our board output while simultaneously providing the best design. But more importantly, we want to provide a product that is cheap enough to be a good value proposition to our target market, while also broadening the application of safety critical technology. One of our original goals was to reduce the cost of the systems used for these tests - and by doing so the companies that use our designs can save money to be reinvested, hopefully back into more satellites, or other space related research and development, which we believe is to the benefit of the whole human race.

Future Work

Vibration Comparison Testing

As mentioned earlier we were only able to complete static vibration testing. Unfortunately due to logistics and timing we were unable to complete dynamic testing. This test would be run on a shake table alongside a reference sensor to be able to compare results after the test is complete. This testing is something we believe is critical to be able to compare and qualify the system alongside wired solutions. Frequency and time domain analysis is needed to verify performance of our data acquisition system. Although we were not able to complete this testing, the static testing allowed us to create all of the required server side, client side and post processing software, so in theory one should be able to attach the board to a shake table and proceed with testing with little to no development effort.

Condensed Form Factor

As our design is intended to be mounted on a spacecraft with minimal weight and size so as not to affect the accuracy of the vibrational measurements themselves, there will always be a need to reduce the form factor of our system. This would require altering the PCB design and conducting a second hardware revision. Firstly, as this was our first revision and we left ample room for extra components if needed, added headers for debugging purposes, and included numerous test points, we would begin to shrink the form factor by removing or condensing these parts after having gained confidence from the initial design. However, the biggest reductions in form factor could be achieved by integrating the components currently on the Quiksilver Module directly onto our PCB. The module contains the CYW43907 SiP, a single antenna, a SPI flash, an inductor, and a crystal oscillator, all of which could be added directly onto the WASP module without a separate solderable card. This design revision, in particular, would be very challenging due to the complexity of the CYW43907 SiP. Nevertheless, since we have confidence on the

PCB design of our other subsystems, proceeding with this level of integration could become a feasible task.

Multi-Axis Accelerometer

The WASP module currently utilizes a single axis MEMS accelerometer to acquire analog vibrational signals. However, in order to design a more robust system that better analyzes the complete acceleration profile of a spacecraft under test, the WASP module could be upgraded to a three axis system. From a hardware perspective, this would require a three axis accelerometer and a corresponding three channel ADC or three separate single channel ADCs. In regard to software, the new system, assuming the same sampling rate on each axis, would need to be capable of handling three times as much data being received from the accelerometer and eventually transmitted over WiFi. Doing so would broadly entail integrating the three data streams into a single packet structure and compensating it on the server side to de-interleave the data for post processing.

Large Node Testing And Evaluation

As was mentioned above, we were only able to fabricate a total of five of our WASP boards due to the immense cost of fabrication for a precision module on our university budget. Due to this fact, we were limited in our system tests to 5 boards, much less than our soft limit of 300 devices on a network. To fully evaluate the maximum performance of the system we designed, we would need to test the working system on larger numbers of boards on a closed network (ie no other networks within range, no devices connected besides WASP modules), ideally in a real satellite testing facility. Due to the emphasis we placed on the scalability of our system, we would not anticipate any problems or degradation of service up to numbers in the hundreds of devices, but this testing would allow us to take more data related to the working of our system and analyze the performance limitations of such a system.

Software Optimization and Improved Feature Set

As with many prototype or proof-of-concept software works released, the WASP source code was written with delivering a working product as the highest priority, while organization and optimization came after. As such, some work remains on refactoring our codebase to be more condensed, readable, and organized. Additionally, time could be well spent on performing an in depth analysis of the client and server along with the large node testing mentioned above to more heavily optimize the code and streamlining the system to consume less processing resources and provide more performant code overall. These tasks would be a high priority for us if we had slightly more time.

In addition to refactoring, we have imagined some extra features and improvements that could add functionality and ease of use to our system. Chiefly among these ideas is a graphical

interface. Although our command line interface and regularly updating ASCII table of devices is quite helpful, the gold standard for interfacing would be a GUI. This GUI could present the information in a better format, and potentially even automate any interaction the engineers would need to have with the server.

Besides the GUI, we would also like to add support for the server to run on big endian machines, like any processor in the x86 family of devices. Currently we disallow the use of the server on big endian machines to avoid any errors based on the difference of byte ordering from the WASP devices, but effort could be made to add support for all devices, significantly broadening the range of devices the WASP server can be used on.

Finally, we believe there are more features that could be added to the platform on both sides to support other tests types and applications. The features added would be heavily dependent on applications, but in general, we would imagine SD card support, variable sampling rates, and more configurability would be beneficial to almost all future stakeholders.

Lessons Learned

Timeline & Design Challenges

Our initial timeline was extremely aggressive; this was done so we could possibly create a second design revision of the board. By the end of the Fall quarter we wanted to send out the initial prototype for fabrication. While we have had experience in internships working on aggressive timelines like this, due to our engineering and core classwork load we quickly realized that this is not feasible. To reduce our total workload we just elongated our project schedule to accommodate our academic schedule. Increasing our timeline put the second design revision extremely close to the end of the school year, so our group agreed that if we did not finish the second design revision we would still consider our project in a completed state.

Integrating the CYW43907 Murata implementation is not a trivial task, it contains over 100 pins ranging from power, ground, bootstrapping and interfaces. Due to the tight pin pitch a mistake cannot be corrected after layout is completed. That being said to have confidence in our implementation we asked our partners if they would help review our schematic. Initially they committed and created an internal work order that allowed us to have foresight on the progress. Slowly, no progress was made and attempts to contact the support engineers were met with no response. Our hardware design efforts were mostly stopped waiting for a response. Eventually after two months we decided we had to commit to an unreviewed design or determine a way to integrate the module in a fashion that will reduce the risk of error. Thankfully we found the Quicksilver module and pursued that path without any issues. This experience is a good example of why engineers need to be able to think on their feet as well as be able to factor risk in decision making. Although brief, we can use this experience in the future to make better engineering decisions.

Hardware and Software Integration

Our project is a good example of why software and hardware teams cannot operate without each others input. Our system requires tight coupling of how the hardware functions to get the most out of the software as well the hardware design needs to take software limitations into account. The most salient examples are the design and implementation of our SPI and I2C drivers. We had to have a clear understanding via component datasheets how we needed to interface with our gas gauge and ADC. From there we designed software to correctly send and receive data to the device. While this sounds trivial, operating in this mindset will translate to architecting a much larger system, where intensive collaboration between hardware and software teams is necessary to ensure a proper solution is created.

Acknowledgments

The authors would like to thank Analog Devices and the Santa Clara University School of Engineering for funding this project. Without the funding from both parties, manufacturing a physical piece of hardware with a low volume count would simply not have been possible. In addition, the guidance and support of our advisors Dr. Wilson and Dr. Dezfouli helped us bring this project into fruition.

We would also like to thank Cesar Tesen, fellow student, who helped up design the silkscreen of the PCB. Without his artistic abilities we wouldn't be able to capture the true essence of a senior design project.

References

- [1] National Aeronautics and Space Administration, "Vibration Testing," 2012.
- [2] P.M. Fantasia, "Vibration and Acoustic Test Facility (VATF): User Test Planning Guide," 2011.
- [3] National Instruments, "Data Acquisition Overview," 2019.
- [4] M. Lindstrom, What's Wrong With my Piezoelectric Accelerometer, 2014, .
- [5] X. Liu, X. Dong, Y. Wang, J. Dodson and B. Joyce, High-g Shocking Testing of the Martlet

Wireless Sensing System, 2018, pp. 1-6.

[6] E.E. Petrosky, A.J. Michaels and D.B. Ridge, "Network Scalability Comparison of IEEE 802.15.4 and Receiver-Assigned CDMA," *JIoT*, pp. 1, Nov 29,. 2018.

[7] Dunn Terry, "RC Battery Guide: The Basics of Lithium-Polymer Batteries," .

[8] Arrow, "Quicksilver Datasheet," 2017.

[9] Cypress Semiconductor, "CYW43907 WICED IEEE 802.11 a/b/g/n SoC with an Embedded Applications Processor Datasheet," 2018.

[10] Analog Devices, "ADXL1004 Datasheet," 2018.

[11] PCB Piezotronics, "3501A2060KG Installation and Operating Manual," 2016.

[12] PCB Piezotronics, "3501B122KG Installation and Operating Manual," 2016.

[13] Tektronix, "Wi-Fi: Overview of the 802.11 Physical Layer and Transmitter Measurements," 2013.

[14] Cypress Semiconductor, "CYW4390x Power Consumption Measurements," 2017.

[15] Energizer, "Zinc Air Prismatic Battery Handbook," 2015.

[16] Energizer, "Alkaline Manganese Dioxide Handbook and Application Manual," 2018.

[17] Energizer, "Nickel Metal Hydride (NiMH) Handbook and Application Manual," 2018.

[18] Energizer, "Nickel Cadmium Batteries Application Manual," 2001.

[19] Texas Instruments, "Switching Regulator Fundamentals Application Report," 2019.

[20] Texas Instruments, "Low Dropout Regulators Quick Reference Guide," 2018.

[21] Texas Instruments, "BQ27z561 Impedance Track Battery Gas Gauge Datasheet," 2018.

[22] Maxim Integrated, "MAX17260 1-Cell Fuel Gauge Datasheet," 2018.

[23] Linear Technology, "LTC2941-1 1A I2C Battery Gas Gauge Datasheet," 2011.

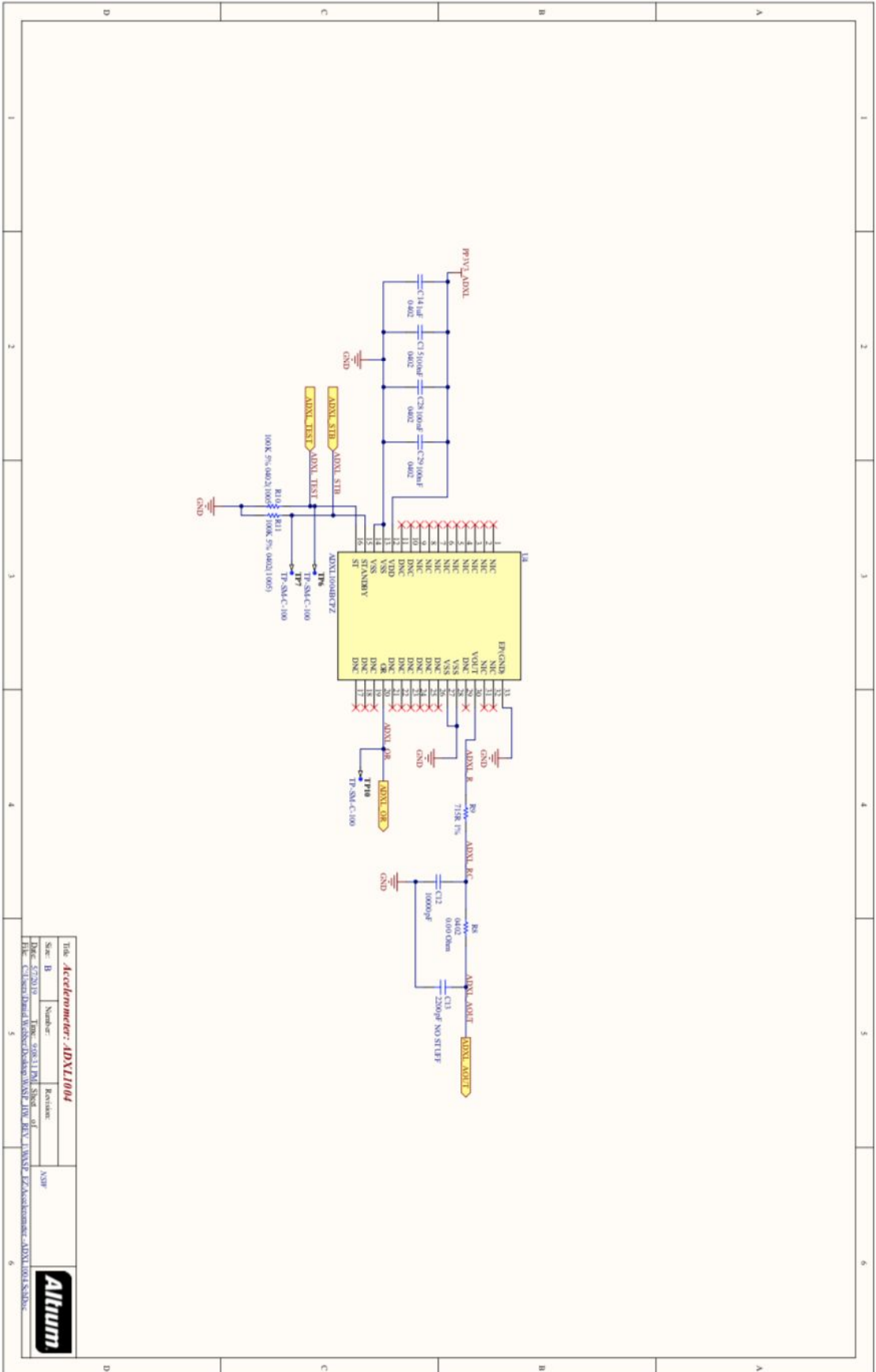
- [24] Analog Devices, "ADAQ7980 SAR ADC Datasheet," 2018.
- [25] Analog Devices, "AD4008 SAR ADC Datasheet," 2017.
- [26] Autodesk, "Eagle CAD Overview," 2019.
- [27] Altium, "Altium Circuit Studio Overview," 2019.
- [28] Cadence, "OrCAD Overview," 2019.
- [29] Cypress Semiconductor, "The WICED Software Stack," 2019.
- [30] Murata, "LBWA1UZ1GC-958 Datasheet," 2018.
- [31] A. Roy and T. Bhattacharyya, "Design, fabrication and characterization of high performance SOI MEMS piezoresistive accelerometers," *Microsyst Technol*, vol. 21, pp. 55-63, Jan. 2015.
- [32] Total Phase, "SPI Background," 2015.
- [33] Cypress Semiconductor, "SPI in CYW43907," 2017.
- [34] Texas Instruments, "TPS63020 Buck-Boost Converter Datasheet," 2017.
- [35] Texas Instruments, "Tips and Tricks for Designing with Voltage References," 2017.
- [36] Analog Devices, "LT1461 Datasheet," 2015.
- [37] Texas Instruments, "TPS73733 3.3V Linear Regulator Datasheet," 2015.
- [38] Texas Instruments, "TPS78018 1.8V Linear Regulator Datasheet," 2015.
- [39] Vishay, "SiS407ADN Datasheet," 2014.
- [40] Texas Instruments, "INA219 Current/Power Monitor Datasheet," 2011.
- [41] Cypress Semiconductor, "WICED Application Framework," 2017.
- [42] M. Velasquez et al, "Markkula Center for Applied Ethics: A Framework for Ethical Decision Making," 2015.
- [43] Federal Communications Commission, "SAR For Cell Phones: What It Means For You,"

2016.

[44] Katwala Amit, "The spiraling environmental cost of our lithium battery addiction," 2018.

Appendix A: WASP Module Schematic

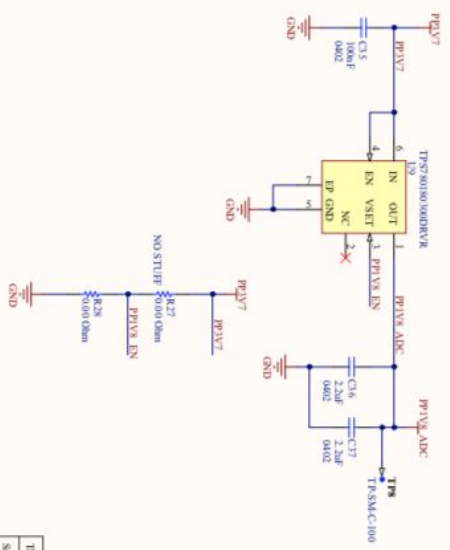
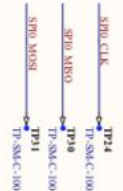
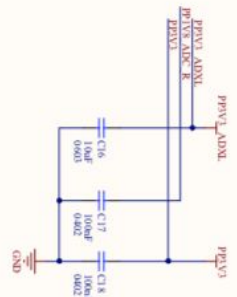
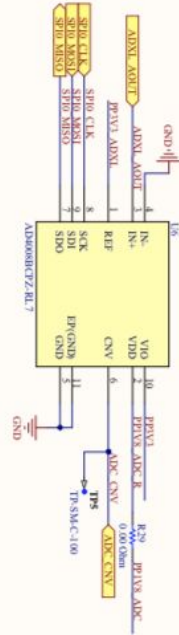
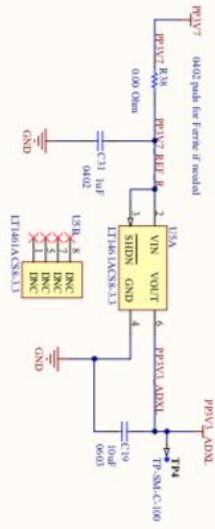
For full design files including Altium files please refer to the github repository linked below in Appendix B.



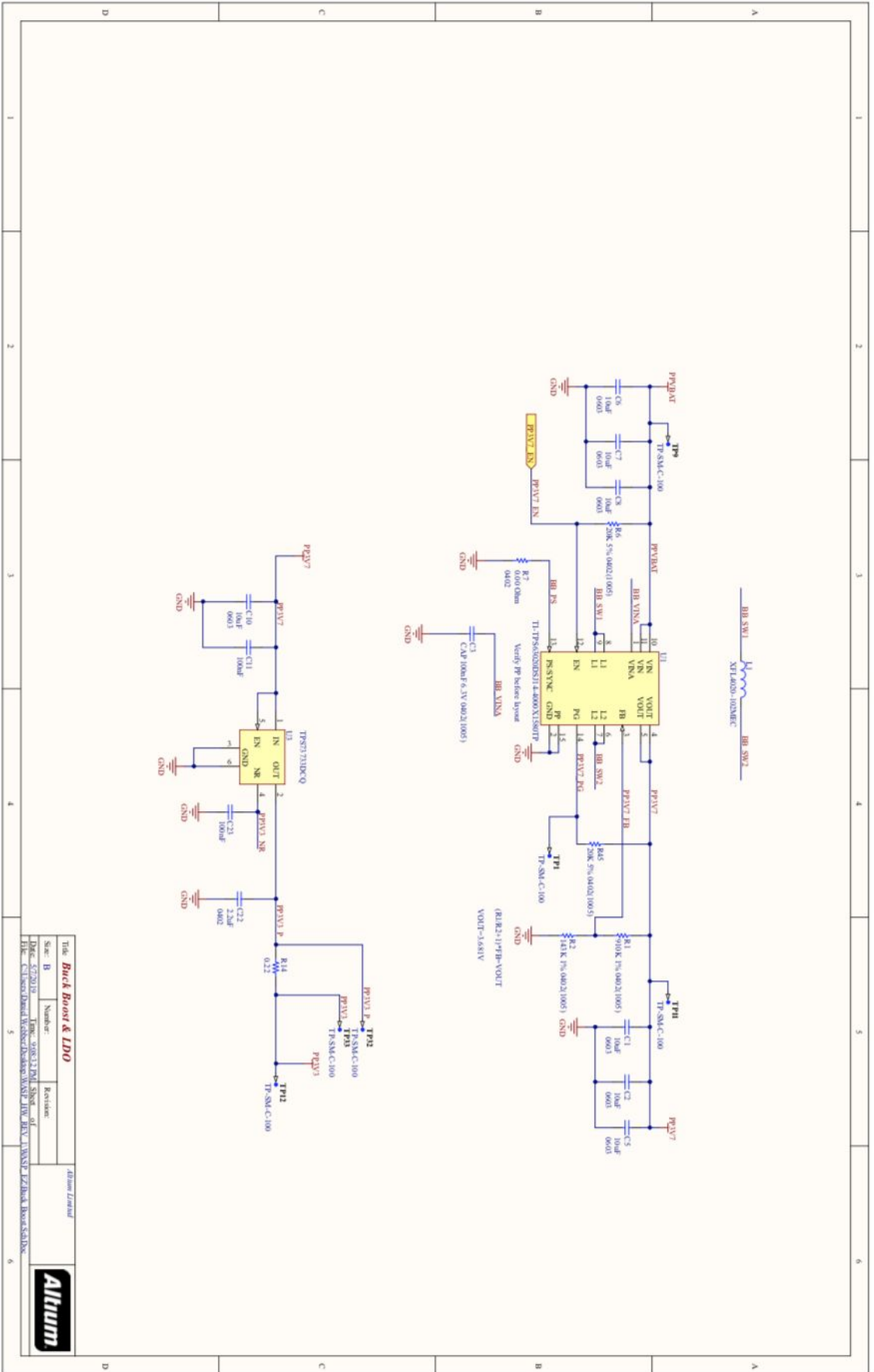
Title: Accelerometer: ADXL1004	
Rev: 4.2.0	Number: ADXL1004
File: c:\users\jand\Documents\Projects\ADXL1004\ADXL1004_Sch.Sch	Revision: ASW
Date: 4/23/10 Time: 10:11:16 Sheet: 1 of 1	



ADXL 3.3V Reference

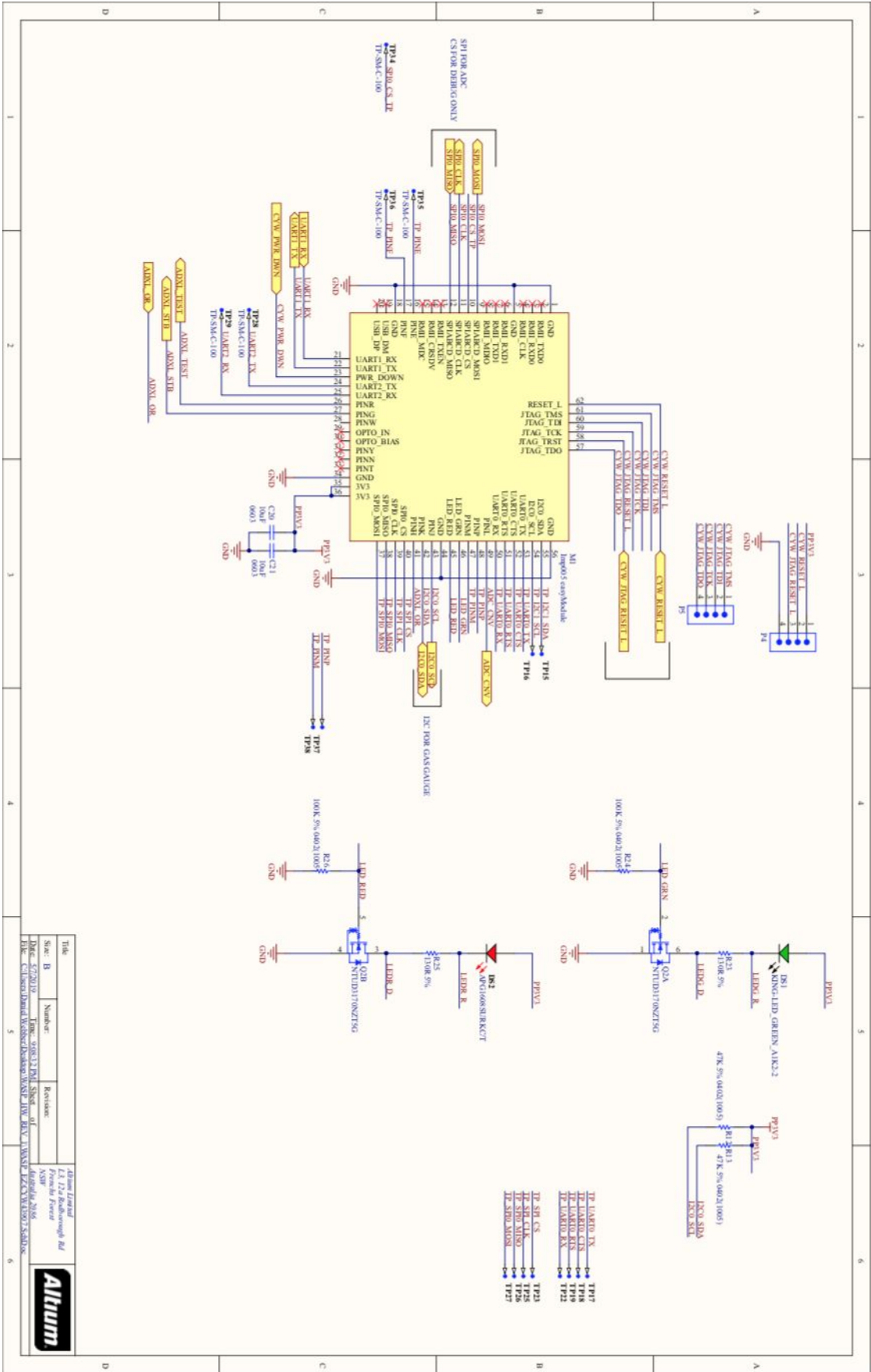


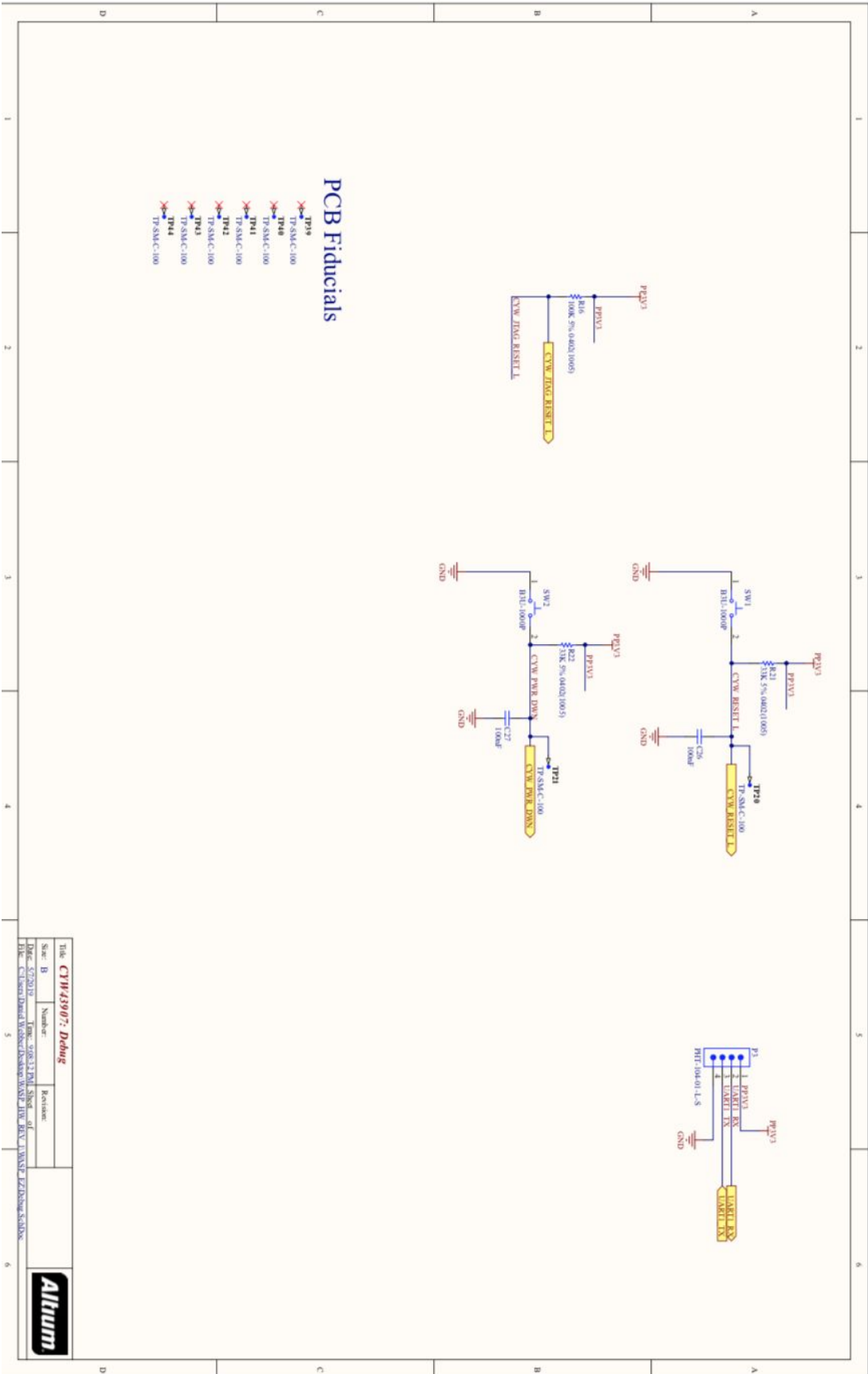
104	Rev: 1.0	1.1.124 Reference
Spec: B	Number:	French Force
Date: 12/2010	Trace: 08/11/2010	Sheet: 1 of 1
File: C:\Soc\Board\Wisc\Design\WSPF_JTW_DEV_1\WSPF_AD4880RQZ-RL7\WSPF_AD4880RQZ-RL7_SCH.DOC		
AltiUm		



Title: Buck Boost & LDO		Revision:	
Spec: B	Number:	Date: 5/7/2019	
Author: [Name]		Checked: [Name]	
Drawn: [Name]		Reviewed: [Name]	
File: C:\Users\David.West\Desktop\WSP\JHEV_JHEV_V1\WSP_L2\JHEV_BUCK_BOOST.LIB			

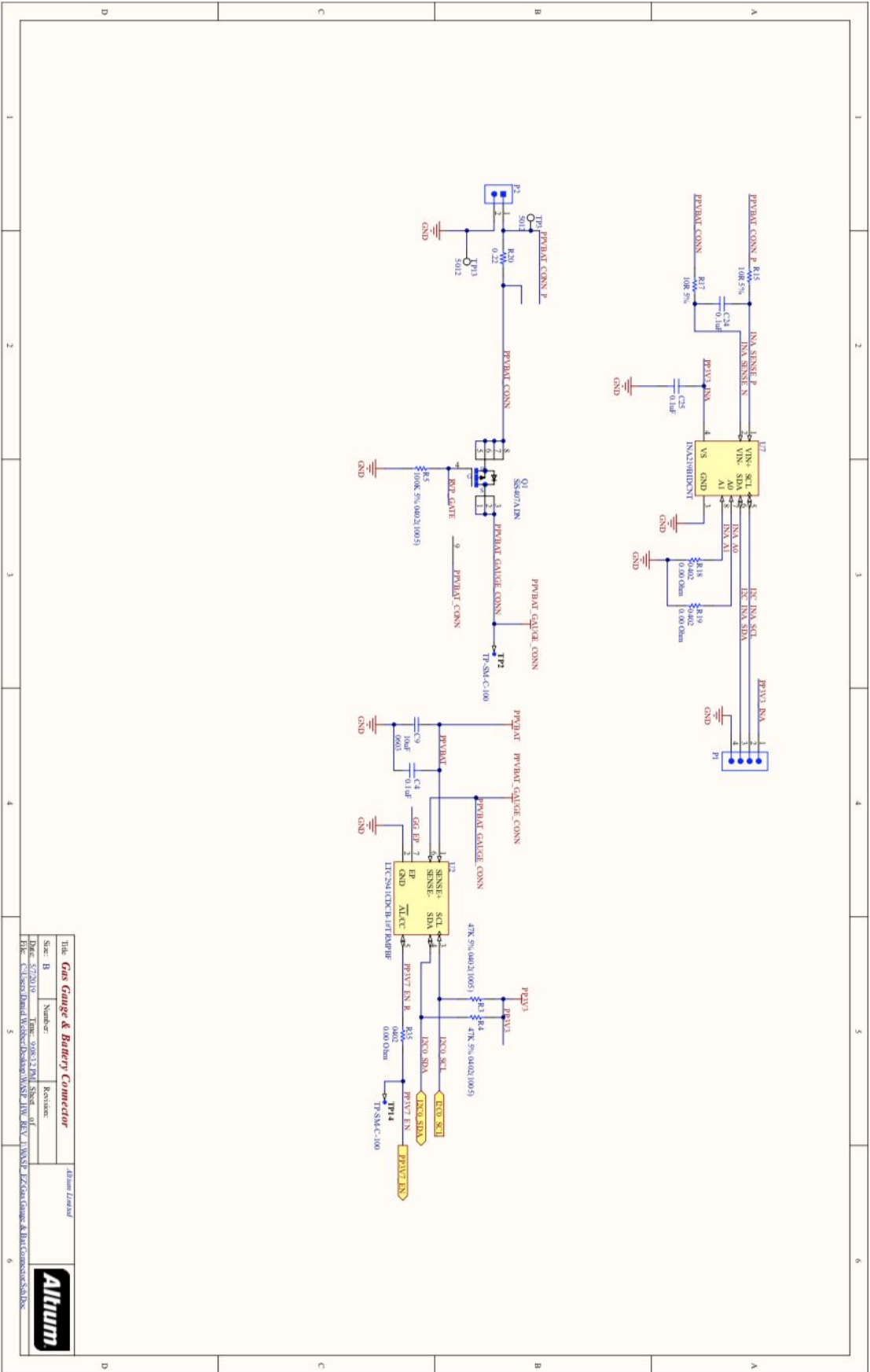






Title: CYW43907: Debug	
Spec. ID:	Number:
Date: 5/23/12	Time: 2:08:12 PM
Drawn by: C:\Users\David.Watson\Desktop\WAVE_HIVE_BV_V1.WSPZ_EZDesign.SchDoc	





Appendix B: Software

The WASP source code consists of many files and thousands of lines of code, much too many to include in this document. Instead, we have placed all of our C source and Altium design files into a Github repository for viewing. To view the files, please see the [WASP-Final](#) repository. The directory structure is listed as follows:

- PCB/ - This folder contains all of the Altium design files generated as a part of creating the printed circuit board.
- WASP_main/ - This folder contains all of the files necessary to build the WASP board firmware. The directory structure is correct for a WICED build, simply drag this folder into the WICED apps folder and follow the instructions in the readme to build the application.
- modifications/ - This folder contains all of the base WICED files that need to be modified to ensure a successful build of the firmware. For instructions on how to modify, see the readme.
- post_processing/ - This folder contains Python and MATLAB scripts with which to do some simple processing on the generated data files.
- server/ - This folder contains the source files and makefile needed to build the WASP server. Instructions to build can be found in the readme.
- README.md - This file contains all of the instructions needed to build the software and evaluate the WASP system in a step by step format.

Appendix C: Project Timeline

Hardware Design Schedule



Software Design Schedule

