THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF ELECTRICAL ENGINEERING

MODULAR SUN SENSOR SYSTEM FOR ORBITAL ATTITUDE DETERMINATION

RAVENDER VIRK
FALL 2012

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Electrical Engineering
with honors in Electrical Engineering

Reviewed and approved* by the following:

Sven G. Bilén
Associate Professor of Engineering Design, Electrical Engineering, and Aerospace Engineering
Thesis Supervisor

Tim Kane
Professor of Electrical Engineering
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

Satellites launched into orbit often require a subsystem to determine the attitude of the satellite for use with communication with a ground station, calculating the spin speed of the system, timing experiments, orbital calculations, and other uses that depend on the orientation and position of the satellite. One version of this subsystem is a set of sun sensors that use the location of the Sun with respect to the satellite to determine attitude. Sun sensor systems are used extensively on a class of satellites known as CubeSats. These smaller satellites, often used in academia, follow a standard developed by California Polytechnic State University and Stanford University that greatly increases accessibility to orbital experimentation.

This project was to develop a fully-working set of sun sensors for the OSIRIS CubeSat being developed at The Pennsylvania State University. Once in orbit, OSIRIS will measure space weather in the ionosphere when stimulated by high-energy ground-based radio transmitters to simulate natural phenomena characteristic of the region. The OSIRIS sun sensors are needed to measure the attitude of the satellite and its science instruments when it is inside one of the stimulated regions and taking measurements. Though these sun sensors were designed specifically for OSIRIS, the circuit can easily be adopted for use aboard another CubeSat looking for a low-cost, high-performance subsystem for attitude determination.

The design cycle of this effort started with developing a theory of operation, running simulations, building and testing a prototype, and finishing with a final, modular, near flight-ready sensor. All design work was done with Altium Designer and all software for the sun sensors was written in C++. The resultant sensors successfully passed accuracy, precision, linearity, speed, power, and environmental tests and now stand as a viable solution for CubeSats to use for attitude determination or to derive customized versions of the sensor for their own needs.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank everyone in the Student Space Programs Laboratory who has helped me grow as an engineer. I would not be anywhere near as knowledgeable as I am if it had not been for the opportunities and guidance the Lab has given me since I started as a freshman. To anyone who ever answered any question I asked or helped me do something, know that I could not have done this without you.

I would especially like to thank Allen Kummer for mentoring me through my work at the Lab and for suggesting the idea for this thesis. I would also like to thank Dr. Sven Bilén for his patience, for keeping me on track, and for always getting me to recheck any estimation, simplification, or assumption I made throughout my work. Finally, I would like to thank Dr. Tim Kane for his continued advice about the honors program and my curriculum.

Chapter 1 | INTRODUCTION

For millennia, humans have used the sun as a basis for direction and guidance. With such a consistent track path across the sky—rising in the east and setting in the west with a slight shift each day used to track the progress of a year—the sun provided a highly-reliable reference point for navigating across land, then sea, then air, and now space.

With the advent of high-speed flight systems came the need for automated sun sensors to track the sun much more accurately and quickly than the human eye could. Sun sensors were first used on rocket flights in the 1950s in the form of phototubes before migrating to space systems as solid-state photoresistive devices [1]. As space systems continued to advance, so did sun sensors. Many satellites require accurate positioning information to conduct communications, determine spatial information, schedule experimentation times, etc. This importance was not lost on the CubeSat community whose developers are always looking for more efficient ways for their space system to navigate in a manner similar to ancestral humans long ago.

1.1 A BRIEF OVERVIEW OF THE OSIRIS MISSION

The OSIRIS (Orbital System for Investigating the Response of the Ionosphere to Stimulation and space weather) satellite will study the space weather that commercial and government spacecraft must regularly deal with as well as the not-so-regular aspects of space weather. Of particular interest are the ionospheric irregularities that occur unpredictably in both time and space, making them difficult to study. However, with the advent of powerful ionospheric heaters in the form of high power radio sites such as HAARP, SURA, and EISCAT, these ionosphere irregularities can now be replicated for study by an orbiting satellite such as OSIRIS. By heating the ionosphere to produce an irregularity as the satellite passes through the

region, OSIRIS can measure the plasma levels of the region before, during, and after stimulation to provide a more accurate account of what space systems experience during these rough phenomena.

OSIRIS falls in a class of satellites known as CubeSats (detailed in Section 2.1). These tiny satellites allow smaller-budget teams, such as universities, to send experiments into orbit at a greatly-reduced cost. A single-unit ("1U") CubeSat is 10×10×10 cm in size with mass of no more than 1.33 kg. OSIRIS is a three-unit ("3U") CubeSat: three single-units stacked vertically. The research and design team behind OSIRIS is the Student Space Programs Laboratory (SSPL) at The Pennsylvania State University. This research lab directed by faculty and run by students gives young scientists and engineers a chance to participate on real-world projects much larger in scope than any encountered during schooling. In the process, students gain invaluable experience working on an interdisciplinary design team to solve tough problems.

1.2 The Need to Have Sun Sensors

A sun sensor is a general term covering a class of solar radiation sensing devices used for attitude determination, attitude control, spin speed determination, and for timing purposes [1]. The OSIRIS satellite's attitude determination system requires sun sensors for detumbling when initially released into orbit and for attitude determination during normal operation.

Several methods exist for attitude determination, such as star trackers, horizon scanners, gyroscopes, magnetometers, analog sun sensors, and digital sun sensors. Of these, the first three are too large to put on OSIRIS. Additionally, analog sun sensors were rejected for their low precision and their need to have calibrated solar panels [2]. The remaining subsystems were thus chosen for attitude determination. This thesis discusses the digital sun sensors.

When at orbital altitude, CubeSats are ejected from their launch vehicle into orbit with velocity and some tip off spin rate. The satellite then must stabilize itself. OSIRIS's onboard sensors work together during this phase to determine how the system is spinning and what steps should be taken to detumble the satellite. The sun sensors are one such subsystem that enables attitude stabilization by determining the rate at which the sun vector is changing.

Once in stable attitude, OSIRIS uses the sun vector and the local magnetic vector as inputs for the optimized TRIAD algorithm to generate the direction-cosine matrix that describes orientation of the satellite's body frame relative to a topocentric frame where $+X$ points east, $+Y$ points north, and $+Z$ points along the zenith [3]. (A detailed explanation of the TRIAD algorithm can be found in [4]). This information is vital for any satellite trying to communicate with its ground station because such communication has a limited downlink window in orbit. Furthermore, OSIRIS absolutely needs this information in order to be sure it is in the correct region of the ionosphere when taking measurements.

1.3 CONTRIBUTIONS

This work contributes to the CubeSat community a new circuit for determining attitude for CubeSats. The sensors are small, modular, and easy to interface so that teams outside The Pennsylvania State University may base their sun sensor subsystems off this design. By doing so, they can save heavily on costs by avoiding pricy commercial sensors and have greater freedom in overall satellite design by shaping the sun sensors to fit their unique satellite structure.

The contribution to the OSIRIS mission is a fully-working subsystem of sun sensors. Specifically, the thesis will contribute the hardware of five sun sensors to the Guidance, Navigation, and Control (GNC) functional group and the sun sensor software class to the

Command and Data Handling (CDH) functional group. These sensors will be accurate, precise, easy to use, simple to attach, and minimal in power consumption.

1.4 THESIS OVERVIEW

Chapter 2 starts with an examination of the CubeSat platform and other sun sensors designed by those in the small satellite community or available commercially. This is followed by a theory of operation for the sun sensors used in this thesis. Chapter 3 begins by converting the theory of operation into a set of design requirements and by explaining the electrical components used to build the theory. These components are at the very center of the following three sections detailing the simulation, breakout board, and final sun sensor circuit board. The chapter ends with an explanation of the software class used to interface with the sun sensors. Chapter 4 overviews all the experimentation performed over the course of this work and documents the results of the tests. Sections 4.1–4.4 document tests of the breakout board, and Section 4.5 documents a test of the final board. Chapter 5 closes by discussing the overall accomplishments, shortcomings, and extensions of the project. Finally, Appendix A contains the board layouts involved in this thesis, and Appendix B contains the sun sensor software class's C++ files.

Chapter 2 | BACKGROUND

This chapter overviews the CubeSat standard, followed with a discussion of other sun sensor systems available to the community. The chapter then closes with a detailed look at the theory of operation behind this sun sensor design.

2.1 CUBESAT BACKGROUND

The CubeSat Project started in 1999 as a collaboration between California Polytechnic State University and Stanford University in an effort to create a standardized picosatellite model. By creating such a model, the project hoped to increase accessibility to space by reducing development time and cost and by making launches more readily available outside the commercial and government aerospace providers. Thirteen years later, the CubeSat Project has grown into a thriving community of over 100 universities, high schools, and private firms developing picosatellites with scientific, private, and government payloads [5].

The success of CubeSats can be attributed to the standardization of satellites and deployment systems. A single-unit (“1U”) CubeSat is a 10×10×10 cm cube with mass less than 1.33 kg. Any satellite meeting this (and other) criteria is authorized to launch into orbit inside a Poly Picosatellite Orbital Deployer (P-POD). P-PODs are very compact spring-loaded deployment mechanisms that affix easily onto launch vehicles and isolate the primary payload from the secondary (or tertiary) CubeSat payload. Thanks to this launch system and the relatively tiny CubeSat size, the cost to launch a CubeSat into orbit is on the order of $40,000 per 1U cube [6].

2.2 Other Sun Sensors

Over time, the CubeSat community has explored many sun sensor designs ranging from dedicated devices to using data from other systems within software. A popular implementation of the latter is to monitor the power output from each solar panel on the faces of the CubeSat and calculate the direction of the sun accordingly. This and other software methods are favorable for CubeSats because they require no hardware addition to the satellite and allow the entire exterior of the satellite to be covered by solar panels. Dedicated sun sensors sacrifice solar panel area for much greater precision and field of view. The following reviews a few sensors in this category.

2.2.1 Commercial Sun Sensors

As the CubeSat community started to expand, companies saw the opportunities available in this market and started designing subsystems for CubeSats. The sun sensor was particularly popular due to its modular nature. A few figures of merit for sensors are accuracy, field of view, and power consumption. Two of the more popular CubeSat sun sensors are the SSBV CubeSat Sun Sensor and the Solar MEMS SSOC-D60.

2.2.1.1 SSBV CubeSat Sun Sensor

This sensor features a CMOS position-sensitive device located under a small pinhole. The device outputs four analog voltages that correspond to the incident angle of the sunlight and includes a conversion formula to derive the sun vector. The sensor features an accuracy of ±0.5°, a field of view of 114°, and power consumption of 50 mW. The unit is housed in a narrow metal casing that brings the total mass to <5 g. At about US$3200, this sensor is one of the least

expensive available on the market, but features no digital communication [7]. Based on the description in its datasheet, this sensor is simplified version of the one developed for this project (Section 2.3). However, this sensor consumes too much power for use aboard OSIRIS.

2.2.1.2 Solar MEMS SSOC-D60

The SSOC-D60 is Solar MEMS digital version of their "sun sensor on a chip." This sensor uses MEMS fabrication processes to create a highly accurate and reliable sensor in a sturdy, long-lasting package. The system outputs four analog voltages relating to the light angle, but also interfaces through the Serial Peripheral Interface Bus (SPI) digital communication protocol to give the elevation and azimuth angles directly. The system also features an onboard temperature sensor to compensate for drift and an attitude failure alarm. The SSOC-D60 is accurate to ±0.3°, has a field of view of 120°, and consumes 12 mW in darkness and 60 mW in sunlight. Though advertised as a low-cost solution, each sensor sells for just over US$10,000 [8]. This sensor also consumes too much power for use aboard OSIRIS.

2.2.2 Previous OSIRIS Sun Sensor

The last revision of the OSIRIS satellite bus was that of OLite-2, a single-unit CubeSat built for a high-altitude balloon test in the summer of 2011. The system had six sun sensors, one on each face of the cube. The sensors were one-inch-diameter circular PCBs with four mounting holes for attaching an aluminum plug with a pinhole and for mounting to the inside of the solar panel boards on the CubeSat's faces. Behind the aluminum plug sat the S5990-01, a CMOS position sensitive detector (PSD), which outputs four currents corresponding to the position of sunlight on the sensor surface. These currents then flowed into current-to-voltage operational

amplifiers (the LTC6079) that sent the voltages to the ADS7682, an SPI analog-to-digital converter (ADC). Each sun sensor communicated with the motherboard through a ten-pin ribbon cable that connected to the respective solar panel board.

Unfortunately, these sensors were never actually tested. Several tests were made on the PSD to determine accuracy, but the circuit as a whole remained unverified. This was of no fault of the designers of the sensor: the SPI system on the satellite was inoperable that summer because the satellite's clock line was experiencing crippling amounts of noise from crosstalk. Combined with issues with the ADS7682 across the satellite, the OLite-2 sun sensor did not get a chance to be tested. Many of the design choices used for these sensors were adapted to the new revision described in Chapter 3.

2.3 Theory of Operation

At the broadest level, the sun sensor subsystem takes in sunlight and outputs a three-dimensional unit vector indicating the direction of the sun relative to the satellite. The subsystem is composed of five independent sun sensors with a common software controller. On the hardware side, each sensor contains three main components: the S5990-01 PSD, the LTC6079 op-amp, and the ADS1115 ADC (see Section 3.2). The flow of data along the system is shown in Figure 2-1.

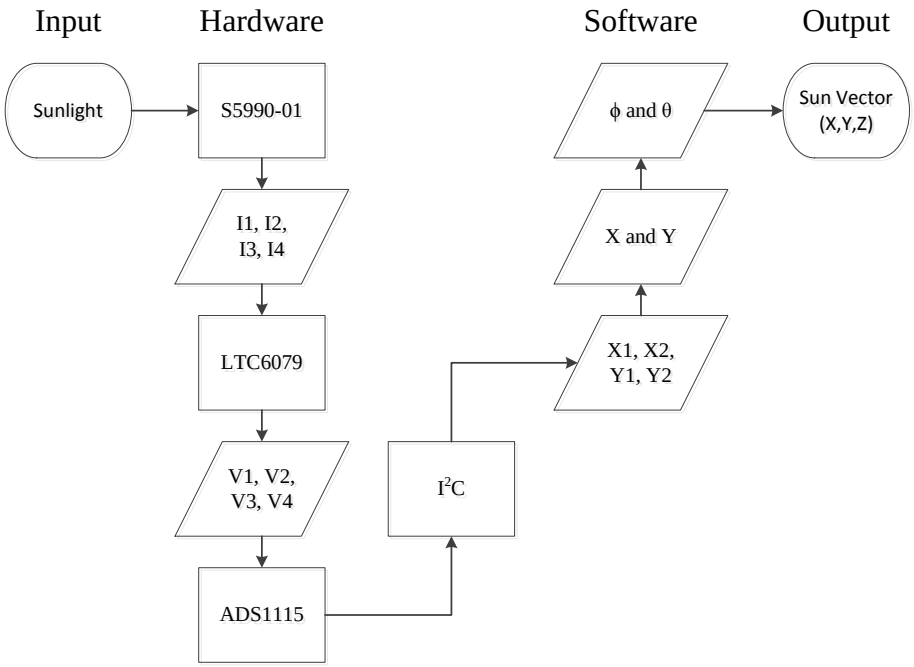

Figure 2-1: Flow of data along a single sun sensor

The S5990-01 (detailed in Section 3.2.1) receives sunlight on its photosensitive surface, but, as Figure 2-2 illustrates, cannot simply be faced towards the Sun because the sensor will return the location of the incident light as the center. By placing a pinhole in front of the sensor, the Sun projects a spot of light onto the sensor surface. Because the Sun is about 150 million kilometers from the satellite, the Sun behaves as a point source. Thus, the size of the illuminated spot can assumed to be the size of pinhole.

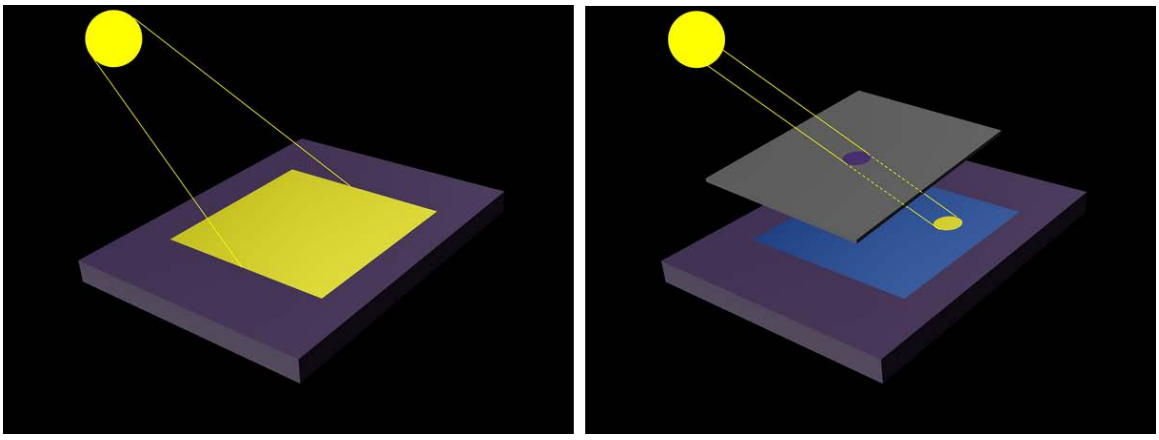

Figure 2-2: Why the S5990-01 needs a pinhole

Upon receiving this spot illumination (“spotlight”), illumination the S5990-01 outputs four currents corresponding to the *+X*, *–X*, *+Y*, and *–Y* sides of the sensor that then go to the LTC6079’s four op amps configured for current-to-voltage conversion. At this step, the currents are converted from the μA range into the mV range for the ADC of the ADS1115. Once through the ADC, the now-digital data is ready for digital processing.

The first step toward generating the sun vector is to calculate the *X* and *Y* offsets in meters at which the sunlight hits the PSD (detailed in Section 3.2.1). The next step in generating the sun vector is to calculate the angle at which the sunlight came in using a series of geometric calculations derived from [9]. The azimuth angle θ is calculated directly from *X* and *Y*; however, the elevation angle ϕ requires *Z*. Because the PSD is only a 2-D sensor, information for this third axis must come from elsewhere. Figure 2-3 is a visual representation of the coordinate geometry relations between *X*, *Y*, and *Z* versus ϕ and θ. The silver ring along the tip of the *Z*-axis is the pinhole through which light enters; the height of the pinhole is the constant *Z* value needed for ϕ. Combining these relations yields the equations

$$\phi = tan^{-1}\left(\frac{\sqrt{X^2 + Y^2}}{d}\right) \tag{2-1}$$

and

$$\theta = \begin{cases} tan^{-1}\left(\frac{Y}{X}\right) - \pi, & if\ X < 0\ and\ Y < 0 \\ tan^{-1}\left(\frac{Y}{X}\right) + \pi, & if\ X < 0 \\ tan^{-1}\left(\frac{Y}{X}\right), & otherwise \end{cases} \tag{2-2}$$

for calculating the elevation and azimuth angles. Note that *Z* is denoted as *d* to represent the constant distance from the sensor surface to the pinhole.

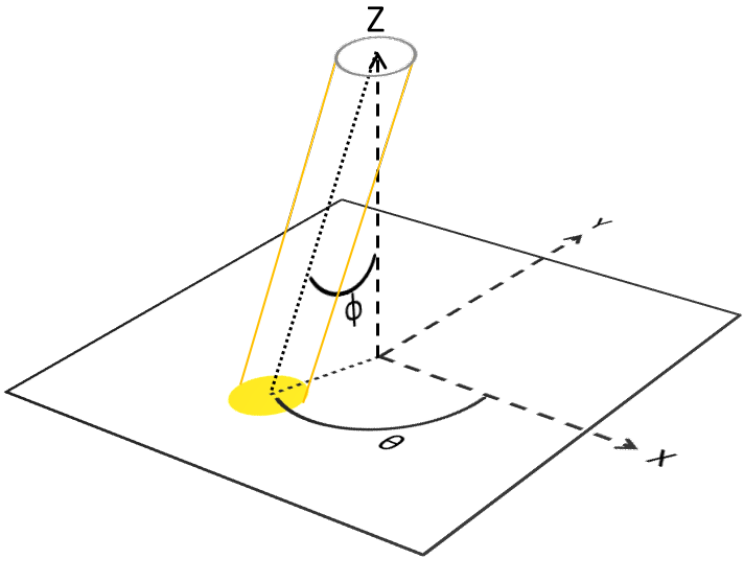

Figure 2-3: The coordinate geometry projected onto the PSD sensor surface

In this case, θ goes from −180° to 180° with the +X axis being 0° and increasing counterclockwise. Mathematically, ϕ goes from 0° to 180°; however, most of this range represents invalid inputs that will never occur. Instead, ϕ goes from 0° to the point where spotlight leaves the sensor's surface, as illustrated by Figure 3-2. The figure also shows that this limit is reduced by the size of the spotlight to avoid edge issues. Figure 2-4 shows that the actual ϕ_{max} is horizontally reduced by the radius of the pinhole (r). A simple right-triangle trigonometric calculation shows $\phi_{theoretical\_max}$ to be 56.3° if the spotlight were a point source and ϕ_{max} to be 49.5°. Thus, the sun sensor will have a reliable field of view of 91°.

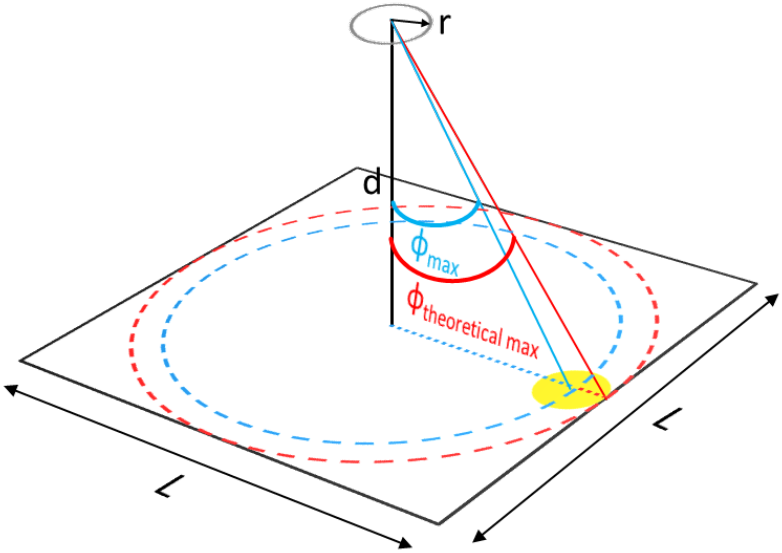

Figure 2-4: Theoretical and practical limits of ϕ

With ϕ and θ calculated, determining the sun vector in rectangular coordinates uses the equations

$$X = sin(\phi)cos(\theta) \quad (2\text{-}3)$$

$$Y = sin(\phi)sin(\theta) \quad (2\text{-}4)$$

and

$$Z = cos(\phi) \quad (2\text{-}5)$$

These equations normally have a radius value, but this was excluded because the radius is equal to unity for a unit vector.

The constant values of *d* and *r* come from the physical dimensions of the aluminum plug (the same one used with the OLite-2 sun sensors) placed on top of the sun sensors and the height of the PSD's sensor surface. For the plug, the pinhole sits 2.286 mm from the surface of the board with a radius of 0.4953 mm. The PSD's sensor surface is listed as 0.8 ± 0.15 mm from the board. Therefore, the distance from the sensor surface to the pinhole is 1.4986 ± 0.1524 mm. Ideally,

this tolerance should be measured for each sensor, but the presence of multiple sun sensor boards encouraged the use of the nominal value for all calculations.

At this point, each sun sensor outputs a sun vector relative to itself, not to the satellite. Therefore, a final system-level transformation is needed to align the selected sensor's sun vector to the body of the satellite. Figure 2-5 details the coordinate system in use by each of the five sun sensors and the desired coordinate system after transformations.

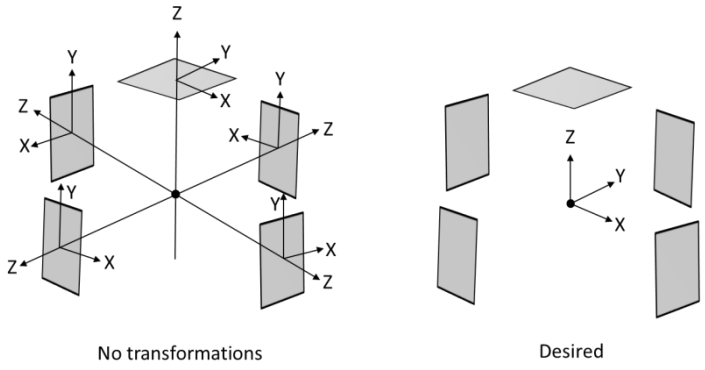

Figure 2-5: The need for coordinate transformations for each sun sensor

To achieve the desired coordinate system, two operations need to take place: a coordinate rotation followed by a projection (a simple translation would not work in this case). However, the projection is not entirely necessary. To understand why, consider how the distance of each sun sensor from the center of the satellite is a few centimeters, whereas the distance from the satellite to the sun is many millions of kilometers. The error associated with leaving the sun vector's base point at the sensor origins instead of the satellite's origin is so incredibly small that performing the calculation needed to do the transformation is not needed. The rotation, on the other hand, is definitely needed. Normally, coordinate rotations require multiplying a rotation matrix with the coordinate vector. However, because the rotations are 90° turns, the rotations can be accomplished by swapping axis values. For example, the $-X$ board's sun vector would be reassigned so $+X = -Z$, $+Y = -X$, and $+Z = +Y$.

To determine which sensor to rotate, the system has to decide which sensor is facing the Sun. At this time, the system assumes that the Sun will be the brightest light source in view. Therefore, the sensor with the largest output should be the one facing the Sun. This assumes that any albedos (the sunlight reflected off planetary bodies) from the Earth and Moon are not bright enough to overpower this assumption. Section 5.2 takes a more detailed look into this issue.

Chapter 3 | DESIGN

This chapter overviews all the design work done for the sun sensor subsystem. Sections 3.2–3.5 detail the hardware design process from simulation to final board and Section 3.6 details the evolution of the software class. Both the hardware and software were designed with a key set of requirements in mind. For reference, the PCB layout details are given in Appendix A and the full code for the software class files is given in Appendix B.

3.1 REQUIREMENTS

The vast majority of the design requirements of the sun sensor subsystem are outlined in the preliminary design review of OLite-2 [3]. Any design requirements not taken from the document were obtained from the SSPL functional leads and the OSIRIS project lead.

3.1.1 Goals

The overall goals of this project were for the final hardware to be ready-to-fly and the software to be ready-to-implement. In other words, the two deliverables of the project were a sun sensor board that could be plugged in and expected to work and a software class that could be imported into the flight software with minimal modification. The line-by-line goals that derive from these overarching objectives are as follows:

3.1.1.1 Simulation Proves Design Concept

Any proposed circuit should first pass simulation to prove it achieves the desired effect. Additionally, this simulation will help determine the values for the resistors and capacitors for the LTC6079 (Section 3.2.2).

3.1.1.2 Breakout Board Schematic and Layout

Following the simulation, a breakout board will be designed for testing. This board will bear all the testing required to design a final sun sensor unit. A schematic and PCB layout are required to order any boards through SSPL (Section 3.4).

3.1.1.3 Breakout Board Functional

SSPL generally does soldering in-house to minimize costs and provide student training. Thus, after the breakout board is soldered, it will have to be tested for any errors and to validate the schematic (Section 4.1).

3.1.1.4 A Single Sun Sensor Outputs a Sun Vector

Once functional, the data from the breakout board will be used to generate a 3-D unit vector corresponding to the direction of a light source (Section 4.2).

3.1.1.5 Sun Vector Accurate to ±2°

The output unit vector needs to be accurate for use in the TRIAD system (see Section 1.2). Whether the inaccuracy is due to calibration errors, device physics, noise, temperature drift, etc., the overall vector will need to stay within ±2° (Sections 4.2, 4.3, and 4.5).

3.1.1.6 Total Power Draw Less than 75 mW

The entire subsystem cannot exceed 75 mW of power at any given moment. This requirement is for the whole system, so each of the five sensors should not exceed 15 mW. However, because the system uses more power in sunlight than in darkness (see Section 3.2.1), each sun sensor could in theory use more than 15 mW when in sunlight provided the others use considerably less in darkness (Section 4.2.1.4).

3.1.1.7 ADC Operates at 400 kHz I^2C Speed

400 kHz is the speed that all I^2C communication will take place on OSIRIS; the sun sensors should not be an exception (Section 4.2.1.3).

3.1.1.8 Breakout Board Survives Thermal-Vacuum Testing

The final mission will take place in orbit. Thus, the sun sensors need to work at very low pressures, in the heat of the sun, and in the cold of the darkness of space (Section 4.4).

3.1.1.9 Software Function: Reset

Should any issues arise, CDH should be able to reset any individual sun sensor. This function will also be used to initialize each sun sensor (Section 3.6.1.1).

3.1.1.10 Software Function: Get Raw Data

Generally, this function will be used internally, but will also need to be available publicly for debugging (Section 3.6.1.2).

3.1.1.11 Software Function: Get Sun Vector

The main software function of the sun sensor software class. This single function call will handle all the necessary communications and algorithms to get an absolute sun vector for GNC (Section 3.6.1.5).

3.1.1.12 Software Function: Get Currently Active Sun Sensor

Used for debugging, this function will return which sun sensor was used to generate the last sun vector (Section 3.6.1.6).

3.1.1.13 Software Function: Get Configuration Register of ADC

Another debugging function; this will return the configuration register of the selected sun sensor’s ADC (Section 3.6.1.7).

3.1.1.14 Final Board Schematic and Layout

Once all testing is finished on the breakout board, the results will be used to design the final sun sensor schematic. The schematic will then be used to create the one-inch circular PCB for the OSIRIS satellite (Section 3.5).

3.1.1.15 Final Board Functional

The final board is the end goal of this project; therefore, it must work as expected (Section 4.5).

3.1.2 Constraints

The timeline for this project was to develop a fully operating system in approximately one year: starting in October of 2011 and ending in November of 2012. The OSIRIS satellite's development cycle goes through 2014 with a planned launch in 2015. Thus, the sun sensors have design requirements that are a combination of assumptions, intelligent estimates by functional leads, and extensions from previous systems. The other constraints on the system are the physical structure of the final board (Section 3.5) and the choice of electrical components (Section 3.2).

3.2 Electrical Components

The selection of the sun sensor's primary components was made in previous iterations. Each sun sensor contains three main components with the remaining being support circuitry for them. The following are summaries of relevant information about these electronics from their datasheets.

3.2.1 S5990-01

The S5990-01 is the heart of the sun sensor. (Most of the information in this section comes from the device family app note [10] provided by Hamamatsu instead of the datasheet.) This two-dimensional improved-tetra-lateral-type position sensitive detector (PSD) takes in light and outputs four currents corresponding to the location on the sensor surface the light struck. These currents are labeled I_1 through I_4 and correspond to the sides of the device labeled X1, X2, Y1, and Y2, respectively. Figure 3-1 shows how the sensor aligns with the coordinate geometry used on the sun sensors.

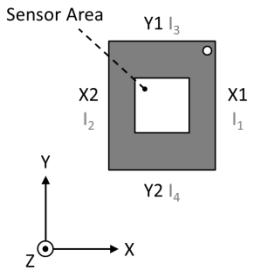

Figure 3-1: Side labels, currents, coordinate geometry, and sensor area of the S5990-01

The S5990-01 has a square sensor surface measuring 4.5 mm per side (referenced in equations as *L*). The datasheet gives the equations

$$X = \frac{L}{2} \times \frac{(I_2 + I_3) - (I_1 + I_4)}{I_1 + I_2 + I_3 + I_4} \tag{3-1}$$

and

$$Y = \frac{L}{2} \times \frac{(I_2 + I_4) - (I_1 + I_3)}{I_1 + I_2 + I_3 + I_4} \tag{3-2}$$

for determining the offset of the light in meters. Instead of dealing with currents, however, the sun sensors convert I_1 through I_4 into voltages V_1 through V_4, respectively, to read into the ADC. If all four resistors used in this conversion are the same, then the same result can be obtained from

$$X = \frac{L}{2} \times \frac{(V_2 + V_3) - (V_1 + V_4)}{V_1 + V_2 + V_3 + V_4} \quad \text{3-3}$$

and

$$Y = \frac{L}{2} \times \frac{(V_2 + V_4) - (V_1 + V_3)}{V_1 + V_2 + V_3 + V_4} \quad \text{3-4}$$

Powering the PSD requires a single, positive power supply up to 20 V (but no ground connection) referred to as the reverse voltage. As reverse voltage increases, the device's response to input quickens, and its saturation photocurrent—the output current when the entire surface area is lit—increases. However, dark current—a source of noise that flows regardless of input—also increases. Compared to other 2-D PSDs in its family, the S5990-01 boasts a smaller dark current, higher response speed, and better accuracy. Additionally, the sensor operates around the visible light spectrum with a higher photosensitivity toward infrared than toward ultraviolet but also a longer response time.

If the sensor is simply placed in front of a light source as bright as the sun, it will always return the saturation current. To get the direction of the light source instead, the sensor has to be behind a shield with a small hole to limit light input to a spotlight. When a photon strikes the highly-resistive semiconductor substrate, it generates currents via the photovoltaic effect outward to electrodes on each side of the sensor to make up the four currents. The physical location returned by these currents is the center-of-gravity of the incident spotlight. The accuracy of this center is exact when at the origin but starts to skew the further out toward the edge that the spotlight travels. When the spotlight encounters the edge of the sensor surface it drives the position detection error up very quickly because the detected center-of-gravity is no longer valid (see Figure 3-2).

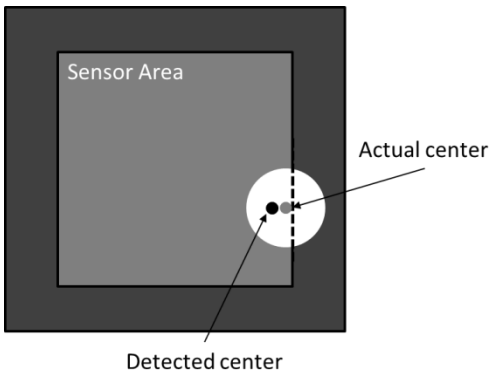

Figure 3-2: Edges effects on detection accuracy

The OLite-2 sun sensor team selected the S5990-01 because it was the best overall solution. Using two one-dimensional PSDs would be more accurate, but the board's space limitations prevented this. Moreover, other two-dimensional PSDs with higher accuracies were designed for highly-specific applications.

3.2.2 LTC6079

The simplest part of the circuit, the LTC6079 is a micropower precision CMOS quad operational amplifier. This chip makes up the four current-to-voltage op amps that convert and magnify the currents from the S5990-01 into the four voltages passed onto the ADS1115. The OLite-2 sun sensor team chose this model for its compact package, low bias current, and minimal power consumption.

3.2.3 ADS1115

The ADS1115 is a very small, low-power, 16-bit analog-to-digital converter that communicates over I^2C (Inter-IC), a digital communication protocol. The chip can either read two

differential inputs or four positive voltages referenced to GND. The device also has several other nice features that are not relevant to this project.

Though the device has four input pins, the ADS1115 only has a single ADC on board. The I^2C master must tell the device which input to connect its multiplexor to. Once selected, the input goes through a programmable gain amplifier (PGA) before reaching the Δ–Σ ADC where it is converted into a 16-bit twos-complement value. However, because the ADS1115 can only read positive inputs and the 16^{th} bit is the sign bit, the resolution is really only 15-bits in the four-input case. To determine the resolution of the input, the maximum positive input value of the PGA is divided by 2^{15}. For example, the default value of the PGA is 2.048 V, so each bit of the digital word corresponds to 62.5 μV.

For I^2C, the ADS1115 has four possible addresses via its single address pin. During any communication, a master device first sets the ADS1115's pointer register to either its configuration register (0x01), to set its configuration, or its conversion register (0x00), to later read data from. The configuration register must be set every time the master needs to read a different input and upon every power up because the ADS1115 does not save configuration settings. Overall, Figure 3-3 outlines a typical communication.

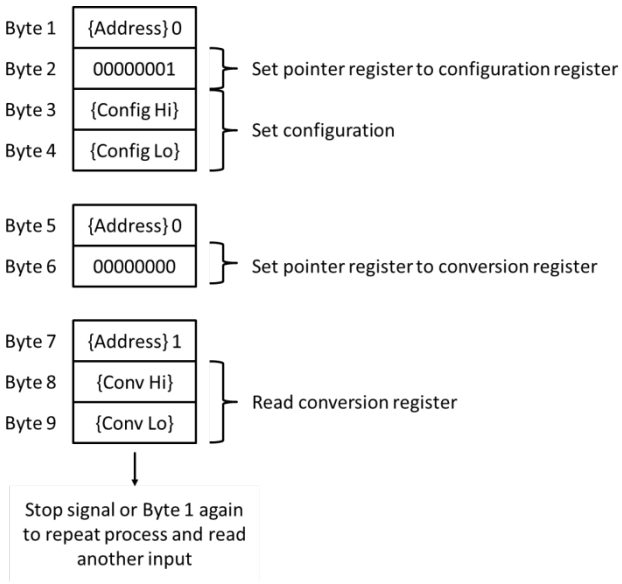

Figure 3-3: A typical communication with the ADS1115

The ADS1115 was selected by the OSIRIS team for the satellite for its low power, size, versatility, ease of use, and I^2C compatibility once the system was migrated over to I^2C from SPI.

3.3 SIMULATION

The sun sensor simulation was not very extensive. Because the ADC could not be simulated without code, this left the PSD and the op-amps. The model used for the PSD is the equivalent circuit of each of the current outputs for the S5990-01 [10]. For the LTC6079, the simulation component model was available for download from Linear Technology. Figure 3-4 shows the simulation schematic used to test the circuit and

$$V_{out} = -I_{in}R \tag{3-5}$$

gives the expected output for each of the four channels based on the input and the resistors. In the end, the resistor value was selected to magnify the PSD currents by 10^3.

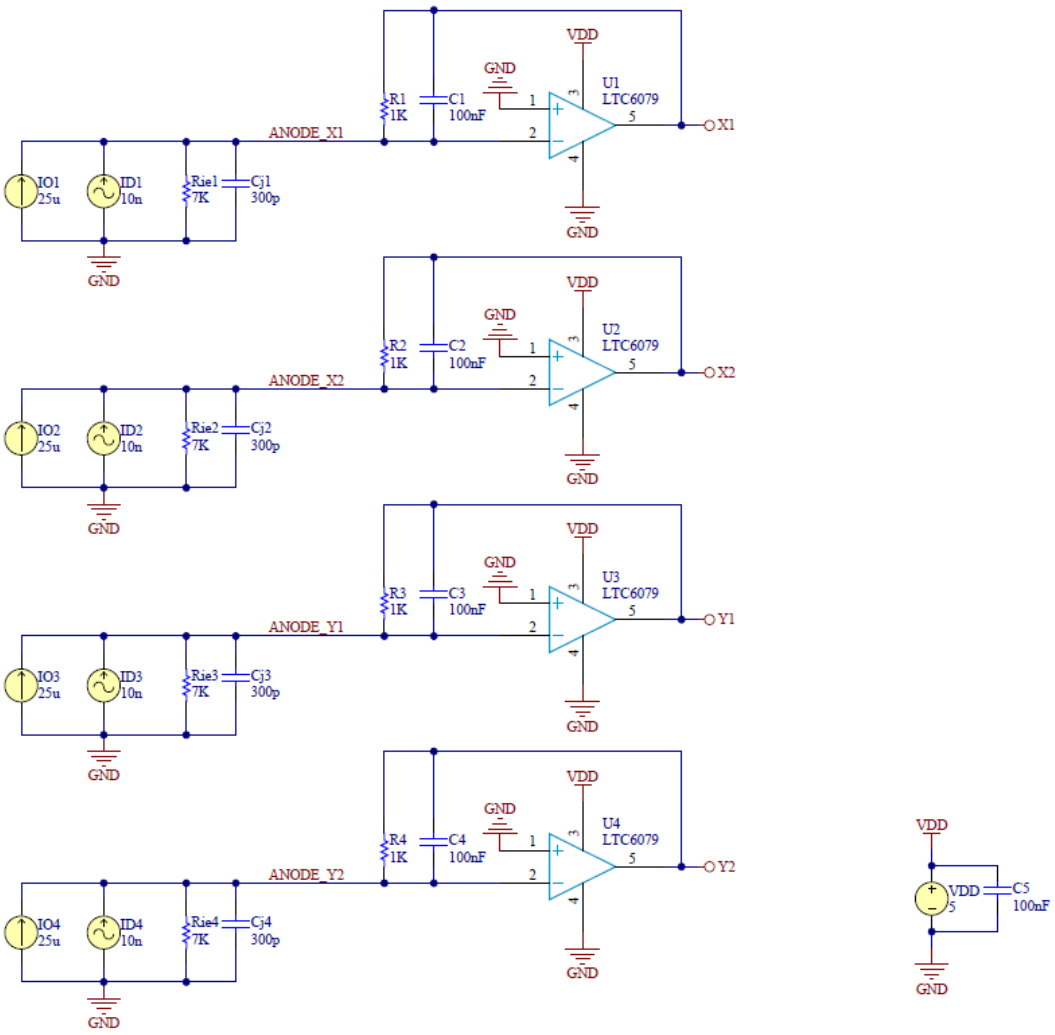

Figure 3-4: Simulation schematic

Other than showing that the circuit worked as expected, the simulation was more useful for selecting the value for the lowpass RC filter for the channels. With a combination of calculations and empirical testing, the value of 1 kΩ was selected for the resistor and 0.1 µF for the capacitor. This gives a lowpass filter with a cutoff frequency at around 3.2 kHz. This value is high enough to sample the PSD quickly while also low enough to eliminate a large amount of high-frequency noise. Finally, the simulation showed that V_{REF} for the op amps could be at GND, greatly simplifying the circuit.

3.4 BREAKOUT BOARD

With the simulation performed, a prototype board was developed for rigorous testing. The design idea was to isolate each of the three major components from Section 3.2, with each of their support circuitry, from one another. This way each subcircuit could be tested and debugged independently. Figure 3-5 shows the schematic of the breakout board. Note that the S5990-01 and the LTC6079 are not disconnected in this schematic. Instead, the S5990-01 was simply not connected to the board while the LTC6079 was being verified (See section 4.1). Testing the entire circuit simply involved connecting P1 and P2 together and P3 to an I^2C master. The top half of the circuit and bottom half also had separate power supplies, so each one was given a bypass capacitor to reduce noise.

The power supplies available to the circuit from the satellite were +3.3 V, +5 V, and +15 V. The circuit was powered with +5 V because the ADS1115's input limits are set by its supply voltage. Though +5 V could have been used to power the S5990-01, +15 V was used instead to maximize the reverse voltage (Section 3.2.1)

Finally, the ALERT, SDA, and SCL pins of the ADS1115 were given 10-kΩ pull-up resistors. This value was selected based on acceptable I^2C values. There was no plan to use the ALERT pin; the option was merely available should the need arise. The ADS1115's ADDR pin was tied to ground to give it the I^2C address of 0x48 (before left-shifting). This address was chosen for convenience, but was first cleared with the OSIRIS functional lead.

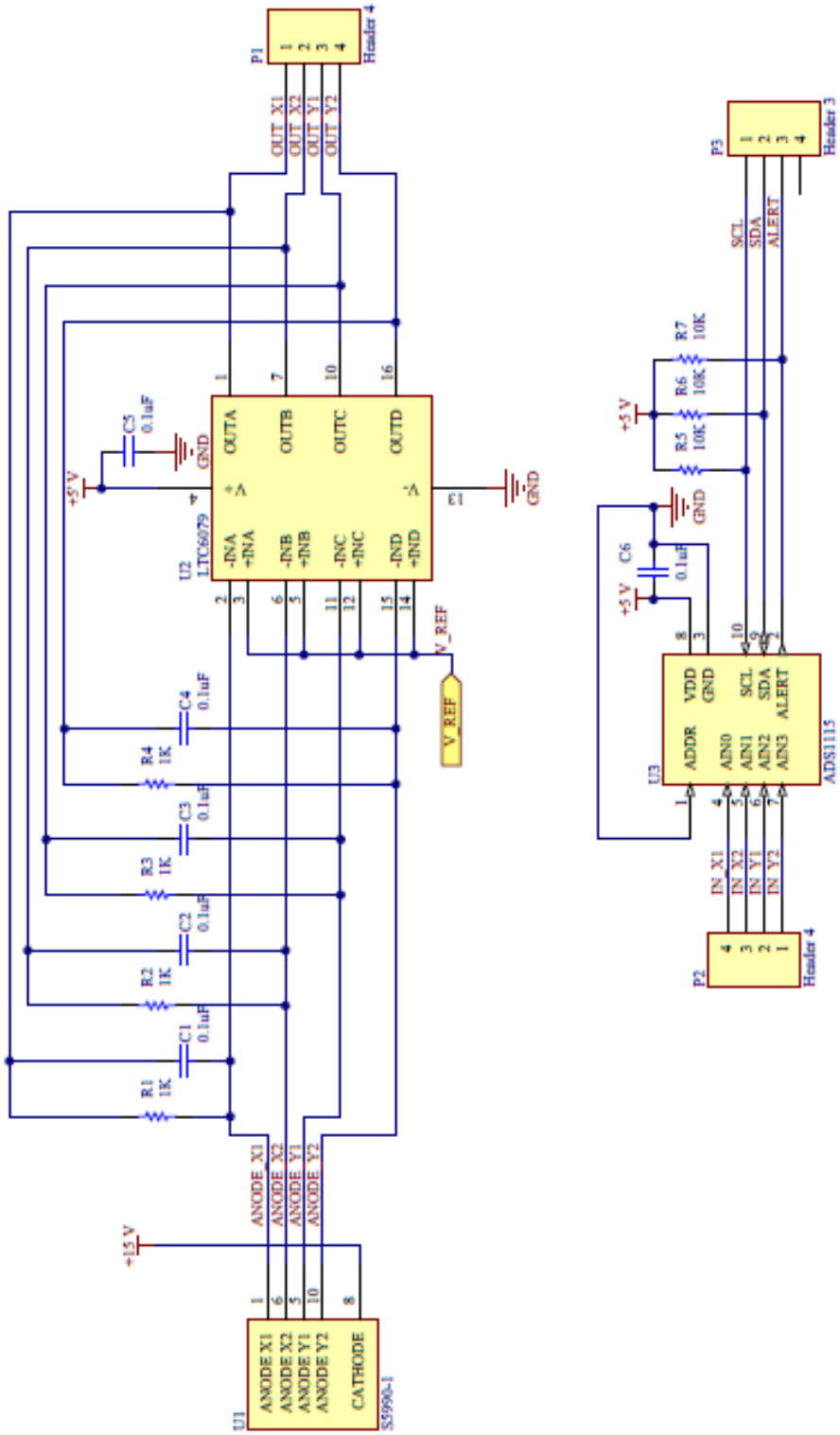

Figure 3-5: Schematic of breakout board

3.5 Final Board

The final board connected all the components together in the schematic (Figure 3-6) and included some subtle design improvements over the breakout board thanks to the experimentation of Sections 4.1–4.4. To start, a subtle change was the addition of more bypass capacitors: one for each of the three chips and two large ones at the power rails' point of entry onto the board. Another subtle change was the pull-up resisters were changed to 3.3 kΩ to sharpen the transitions of the I^2C clock signal. Moreover, the ALERT pin of the ADS1115 was no longer connected.

The major changes to the schematic were the 10-pin board connector and the thermal subcircuit (detailed in Section 3.5.1). This board connector was the same one used on OLite-2 and was reselected with conjunction with the Power functional lead because the sensors connect to solar panels through the ribbon. Though the connector and ribbon have ten pins, the sun sensors only need five: +5 V, +15 V, GND, SDA, and SCL. The excess pins were put to good use instead of left floating. As shown in Figure 3-7, the +5-V line was given two pins to account for the higher current draw of the thermal subcircuit. Additionally, the power rails and the I^2C lines were all separated by GND lines to eliminate any crosstalk.

The one big change made to the board layout was the rotation of the PSD by 45° so that the mounting holes align along the edges instead of the corners. This was done to account for a mounting change the Structures functional group made to the holes for the sun sensors (i.e., they were rotated 45°). Aside from this, the board shape followed the same format as the OLite-2 sun sensors: a one-inch-diameter circular PCB with the PSD on one side, and the rest of the circuitry on the other side. However, this revision featured a major design improvement over the previous model: this sun sensor was only a two-layer PCB whereas the OLite-2 sensor was a four-layer PCB.

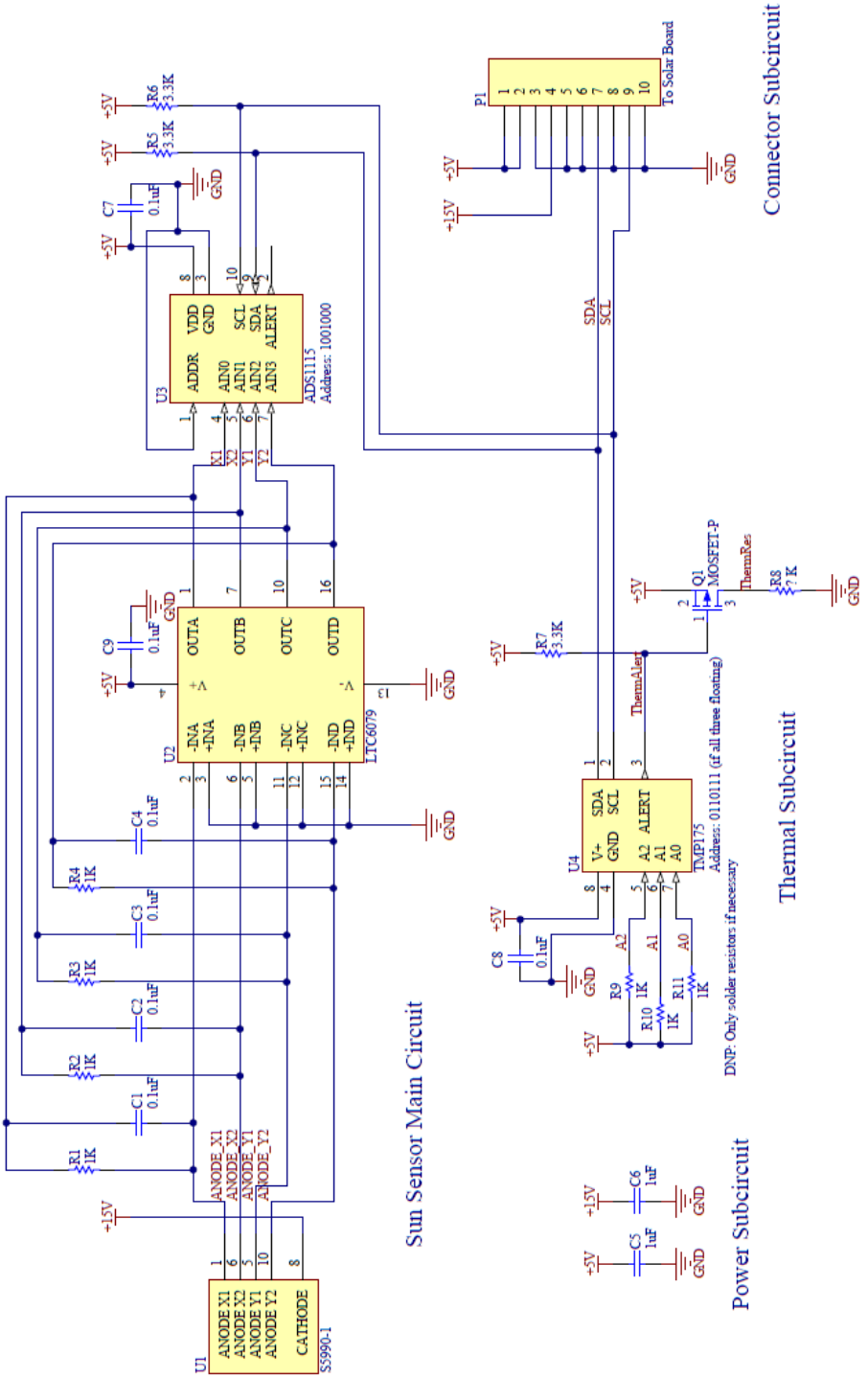

Figure 3-6: Final board schematic

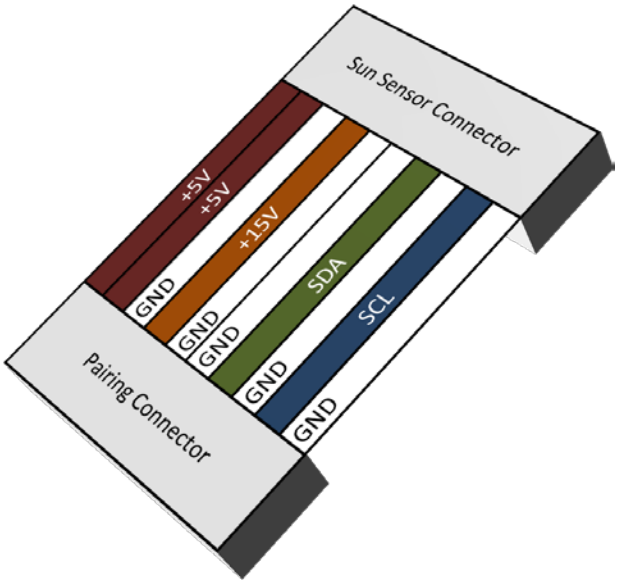

Figure 3-7: Pinout of ribbon connector for final board

3.5.1 Thermal Subcircuit

The thermal subcircuit was a necessary part of the sun sensor. Unfortunately, the circuit had to be designed much too early. As a result, the goal the thermal circuitry was to offer as much flexibility as possible for when Thermal makes its final design decisions. The one certainty in the thermal circuit was that the TMP175 temperature sensor was placed in the exact center of the board, and it triggers a heat resistor via a PMOS through its ALERT line. Beyond this, all design requirements were pending.

To start, the TMP175 needs an I^2C address separate from the ADS1115. Luckily, the TMP175 has three address pins (with floating as a recognized input) to offer 27 possible addresses. For simplicity, all three pins were left floating to simplify board layout on the already-pressed-for-space PCB. However, in case an issue with floating pins arises, the pins were given optional pull-up resistors to select another address. Due to space constraints, optional pull-down resistors could not also fit on the board. So if floating pins is an issue, the TMP175 can only be configured to use one other address corresponding a logic high on each pin.

Next, the MOSFET and heat resistor footprints were chosen for flexibility. The SOT-23 footprint is a very common package used for power MOSFETs. Therefore, the standardized footprint for PMOS SOT-23 will fit a large number of models. The PMOS connects the TMP175 with the heat resistor for which the footprint selected was just two large pads so that most any size resistor could fit.

3.6 Software Class

The software class evolved from a simple main file running on the test master I^2C device to a full software class between the end of breakout board testing and the start of final board testing. As such, the software has been shown to work. Aside from the required software functions of Section 3.1.1, the class also contains a few helper functions. The remainder of this section walks through the inputs, operation, and outputs of each of the functions defined by the sun sensor header file. Note that all functions are void and only return results via variables passed by reference or array pointers. This was done for compatibility with the cross-threading architecture that CDH is developing.

3.6.1 Functions

When a new sun sensor class is initiated, the system creates an I^2C object and readies the configuration data to send to the ADCs.

3.6.1.1 Reset

The input is the index of the sun sensor ADC to reset. The indices are defined in the Sun Sensors enumeration. It performs the reset by rewriting configuration data to the ADC. This function returns the passed-by-reference acknowledge result of the I^2C communication used to reset the sensor.

3.6.1.2 Get Raw

The input is the index of the sun sensor ADC to read from. The code flow follows that of a typical communication with the ADS1115 (Figure 3-3) four times to get the X1, X2, Y1, and Y2 voltages. The function then returns these four values through the “raw” array pointer. This is mostly used by the Get Sun Vector function.

3.6.1.3 Select Sensor

The input is the raw data from all five sun sensors. Once acquired, the function adds each inputs’ four voltages and selects the one with the greatest (brightest) input. The index of the selected sensor is returned through the “active” integer passed by reference. This is mostly used by the Get Sun Vector function.

3.6.1.4 Transform Vector

The input to this function is the sun vector to transform and the index of the active sensor to decide how to transform it. The process to do this works by reassigning the axes of the vector (see Section 2.3). If the $+Z$ sensor is used, the function simply returns because the sun vector is

already aligned along the desired orientation. The sun vector array pointer input is also the output. This is mostly used by the Get Sun Vector function.

3.6.1.5 Get Sun Vector

The only input is how many points to average; if omitted, it defaults to one (no averaging). This function starts by calling the Get Raw function for each of the five sensors and passing all the data to the Select Sensor function. Once a sensor is selected, the correct calibration offset is selected and the averaging algorithm executes at least once to get the X and Y positions. These then equate into ϕ and θ, which in turn equate into a sun vector. Finally, the vector is transformed through a Transform Vector function call and returned through the sun vector array pointer.

3.6.1.6 Get Active

This function does not take any inputs. It simply returns, through the “active” integer passed by reference, the index of the sun sensor last selected to generate the sun vector.

3.6.1.7 Read Config

The input is the index of the sun sensor ADC to read the configuration of. Via I^2C, the function sets the pointer register to the configuration register and then attempts to read the configuration register if the appropriate acknowledge is received. The returned values are the acknowledge status of the write request and the high and low bytes of the configuration passed by reference.

Chapter 4 | TESTING AND RESULTS

This chapter documents all the physical testing of the sun sensor system. The first four experiments tested the breakout board to confirm the feasibility of its design. The board was tested to see how well the components worked, how well the board worked, how reliable the data was, and how the board performed in an environmental test. Once done, the final board was designed based on the results (Section 3.5) and tested in the last experiment.

4.1 VERIFICATION OF THE BREAKOUT BOARD

This experiment tested breakout (prototype) boards for the sun sensor system (Figure 4-1). Three breakout boards—referenced by a number (1, 2, or 3) written on the back of the board—were populated and tested to account for any future board failures. A breakout board had three key components: the S5990-01 PSD, the LTC6079 quad-op-amp, and the ADS1115 ADC. In this experiment, only the LTC6079 and ADS1115 were tested to ensure no complications would arise once testing of the S5990-01 started (and because it was on backorder at the time). Testing the board was a two-step process: test the ADC first and then test the op-amps and ADC together so that any quirks in operation could easily be traced to the correct component.

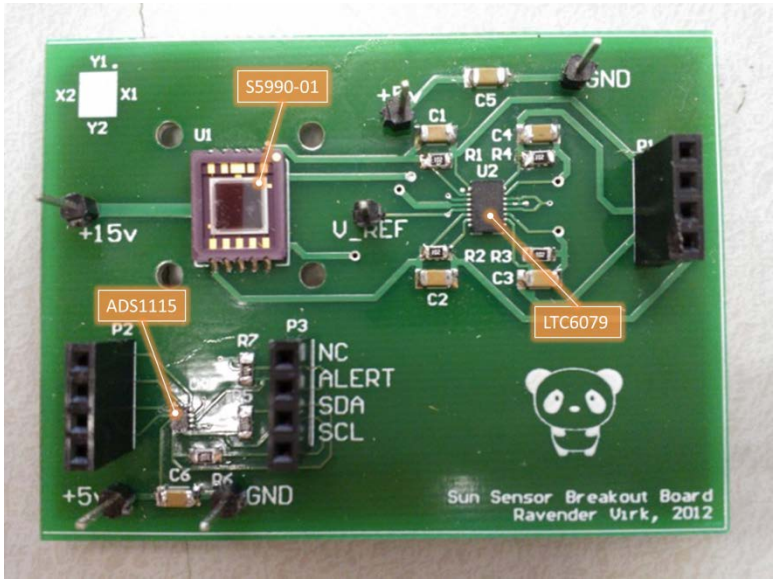

Figure 4-1: Sun sensor breakout board without aluminum plug

4.1.1 Procedure and Results

The experiment setup is shown in Figure 4-2. Testing the ADC was done by communicating with it over I^2C via an MBED prototyping unit. A power supply powered the ADC and another power supply fed test voltages to its input header to mimic input from the op amps. While this happened, the MBED was programmed to poll the ADC every one second for its four values. The MBED then processed the binary output and sent the voltage results onto the screen via PuTTY. Based on the results, the ADC was very accurately reading the input voltages. This allowed the experiment to continue to the next step.

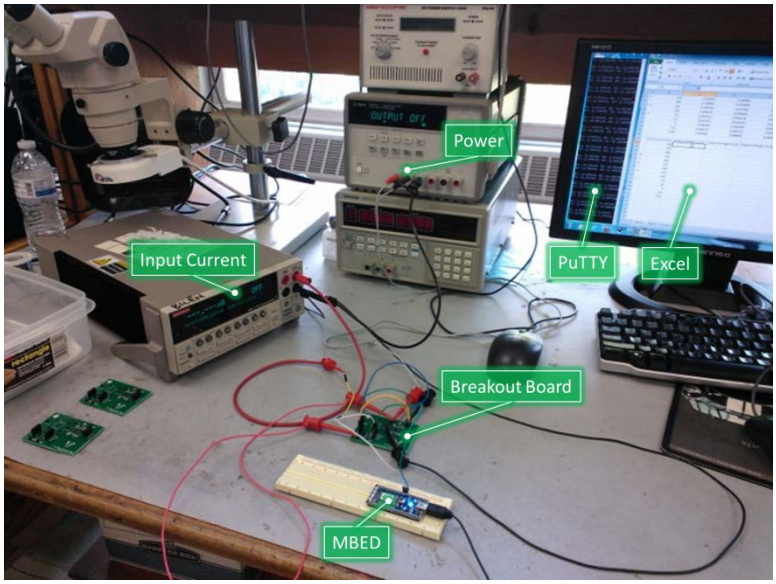

Figure 4-2: Test setup for breakout board verification

The second half of the experiment took place with the same equipment, but with the second power supply set to supply the input current. This power supply could simulate the very low currents (microamps and nanoamps) that the op amps would receive from the PSD. The remainder of the setup was the same except that now both ICs were powered.

During testing, board 1 and board 3 stopped working correctly. An investigation showed that the LTC6079s on board 1 and board 3 had burned out, and the ADS1115 burned out on board 1—likely due to a momentary short caused by the jumper cables. All further testing included safeguards to prevent this from happening again. After replacing the chips with new ones, both boards started working again. However, as highlighted in Table 4-1, the Y1 channel of the op amp still did not work correctly. Thankfully, Table 4-2 and Table 4-3 show the others boards to work flawlessly.

Table 4-1: Data from board 1's ADC during verification test

	Input Current (uA)	Output Voltage X1	Output Voltage X2	Output Voltage Y1	Output Voltage Y2
Board 1	-500	0.498578 V	0.499703 V	6.144188 V	0.498765 V
	-450	0.448701 V	0.449639 V	6.144188 V	0.448889 V
	-400	0.398825 V	0.399762 V	6.144188 V	0.398825 V
	-350	0.348948 V	0.349698 V	6.144188 V	0.348948 V
	-300	0.299072 V	0.299634 V	11.055337 V	0.299072 V
	-250	0.249195 V	0.249570 V	12.142683 V	0.249195 V
	-200	0.199131 V	0.199694 V	12.170621 V	0.199131 V
	-150	0.149255 V	0.149630 V	12.173809 V	0.149255 V
	-100	0.099566 V	0.099566 V	12.177747 V	0.099378 V
	-50	0.049689 V	0.049689 V	12.178684 V	0.049502 V
	-10	0.009750 V	0.009938 V	11.979928 V	0.009750 V
	-5	0.004875 V	0.004500 V	12.001491 V	0.004500 V
	-0.5	0.000188 V	0.000188 V	12.086244 V	0.000188 V
	-0.01	0.000000 V	0.000000 V	0.044439 V	0.000000 V

Table 4-2: Data from board 2's ADC during verification test

	Input Current (uA)	Output Voltage X1	Output Voltage X2	Output Voltage Y1	Output Voltage Y2
Board 2	-500	0.498390 V	0.494265 V	0.500453 V	0.497265 V
	-450	0.445701 V	0.444764 V	0.450389 V	0.447389 V
	-400	0.396200 V	0.395262 V	0.400137 V	0.397512 V
	-350	0.346698 V	0.345761 V	0.350261 V	0.348011 V
	-300	0.297009 V	0.296447 V	0.300197 V	0.298134 V
	-250	0.247695 V	0.246945 V	0.250133 V	0.248445 V
	-200	0.198006 V	0.197444 V	0.199881 V	0.198569 V
	-150	0.148130 V	0.147942 V	0.149817 V	0.148692 V
	-100	0.098816 V	0.098441 V	0.099753 V	0.099003 V
	-50	0.049314 V	0.049126 V	0.049689 V	0.049502 V
	-10	0.010125 V	0.010125 V	0.010313 V	0.010313 V
	-5	0.004500 V	0.004688 V	0.004688 V	0.004688 V
	-0.5	0.000375 V	0.000375 V	0.000188 V	0.000188 V
	-0.01	0.000000 V	0.000000 V	0.000000 V	0.000000 V

Table 4-3: Data from board 3's ADC during verification test

	Input Current (uA)	Output Voltage X1	Output Voltage X2	Output Voltage Y1	Output Voltage Y2
Board 3	-500	0.497265 V	0.498765 V	0.496515 V	0.496140 V
	-450	0.447576 V	0.448889 V	0.446826 V	0.446451 V
	-400	0.397887 V	0.399012 V	0.397137 V	0.396950 V
	-350	0.348011 V	0.348948 V	0.347448 V	0.347261 V
	-300	0.298134 V	0.299072 V	0.297759 V	0.297572 V
	-250	0.248445 V	0.249195 V	0.248070 V	0.247883 V
	-200	0.198569 V	0.199319 V	0.198194 V	0.198194 V
	-150	0.148880 V	0.149255 V	0.148505 V	0.148505 V
	-100	0.099191 V	0.099378 V	0.099003 V	0.099191 V
	-50	0.049502 V	0.049689 V	0.049502 V	0.049502 V
	-10	0.010125 V	0.010125 V	0.010125 V	0.010125 V
	-5	0.004688 V	0.004688 V	0.004500 V	0.004688 V
	-0.5	0.000375 V	0.000375 V	0.000375 V	0.000188 V
	-0.01	0.000000 V	0.000000 V	0.000000 V	0.000000 V

In theory, the expected output was supposed to follow Equation *3-5* for a current-to-voltage op amp, where *R*, in this case, is equal to 1000 Ω, nominally. After accounting for non-ideal resistors and noise, the resultant voltages are indeed very close to what was expected.

One issue discovered was that the ADC had trouble reading the lower-voltage values because of quantization error. Another anomaly occurred when the input voltage was zero (i.e., at ground). The digital output voltages were sometimes in the range of over 12 V. Since the op-amp was not at fault and the ADC cannot read such an input voltage under any circumstances, the error clearly lay with the software. The theory is that while the ADC operates in two-complement, the C++ file was not, and thus mistook a very low value for a maximum. The code was updated to include a quick check that corrects any such values to zero.

As a side note, the resolution of the stored digital voltages was truncated to six digits for display. The actual resolution of the stored values depends on the ADC's configuration (refer to Section 3.2.3 for more information). For this test, the precision was 187.5 μV, as confirmed by the lower inputs.

4.1.2 Conclusions

As expected, the breakout boards worked in accordance with the simulation results (Section 3.3). However, the digital values have issues with quantization. To remedy this, the ADC's programmable gain amplifier configuration was changed from ±6.144 V to ±0.256 V. This increased the precision to ~7.8 μV. While decreasing the gain also limits the safe input to the ADC, the circuit should never produce this value in reality because of the pinhole limiting light onto the PSD. At no point in later testing was this decision an issue.

4.2 CALIBRATION TEST

This experiment was actually much more than calibration: it was a comprehensive look into getting the sun sensor board to align with the theory defined in Section 2.3. It covers every step from attaching the PSD to the board to getting a valid sun vector output to the screen while meeting the design criteria defined in Section 3.1. More specifically, this section details the sun sensor's output reliability, axis linearity, I^2C compatibility, and power consumption.

4.2.1 Procedure and Results

Once the PSD was soldered onto the board, a quick qualitative test showed that circuit would output greater X1, X2, Y1, and Y2 voltages when more light was shone onto the PSD's surface. Afterwards, the same test was done with the plug in place (Figure 4-3) with the same encouraging relationship and with logical changes to match the position of the spotlight. The light source for these tests was a simple LED flashlight. Now that the whole circuit was verified as

working, the unconnected subcircuits, their power rails, and V_{REF} and GND were soldered together to eliminate the jumpers that were obstructing the pinhole's line-of-sight (Figure 4-4).

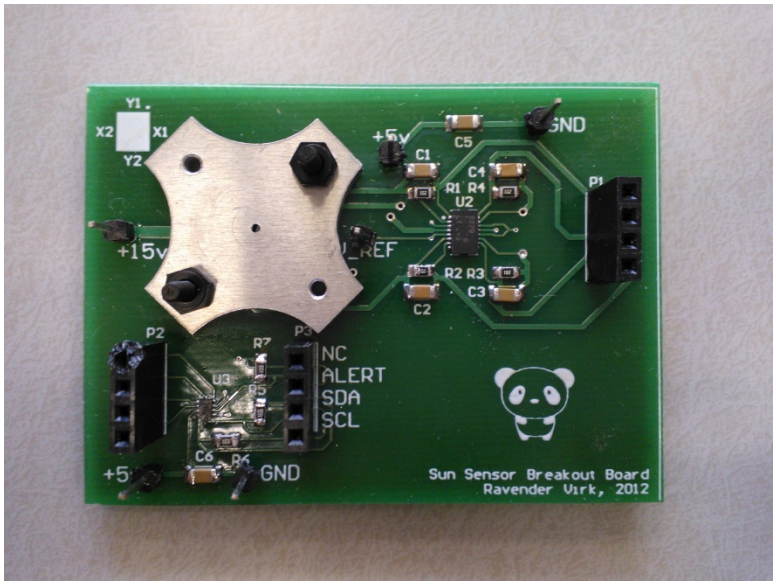

Figure 4-3: Breakout board with aluminum plug in place

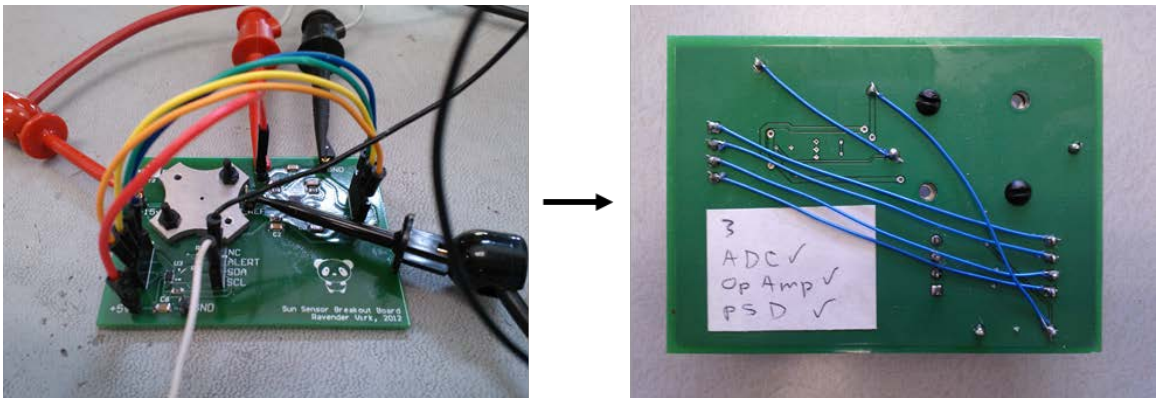

Figure 4-4: Permanent joining of subcircuits on breakout board

4.2.1.1 Reliability of Calculated Angles

The next step was a quantitative test to evaluate how reliable the returned sun vector was. (Note that this was only a partial test of accuracy. A full accuracy assessment also requires the precision results of the noise test of Section 4.3.) The test apparatus for this experiment was the "dark box" in the SSPL (Figure 4-5). Designed a few years ago to test the OLite-2 sun sensors

(Section 2.2.2), it features an array of fourteen infrared LEDs and a special mount for the sensor boards. The LEDs served two functions: First, the PSD's spectral response peaks in the 900–1000 nm wavelength range. Second, the directional nature of the LEDs mimics the Sun's point source nature when in orbit. The mount attaches to the sun sensor by the same screw holes that attach the plug to the board. This perfectly aligns both the pinhole and the PSD under it along to the center of the apparatus. For testing, the mount rotates left and right with a protractor giving the angle. Finally, the entire box is covered in black fabric, made of black plastic, and limits connections to an opening concealed by more black fabric behind the sun sensor. In short, no outside light can interfere with the PSD's reading of the LEDs. (While helpful, this last feature proved unnecessary because ambient office lighting is not bright enough to register with the PSD through the pinhole.)

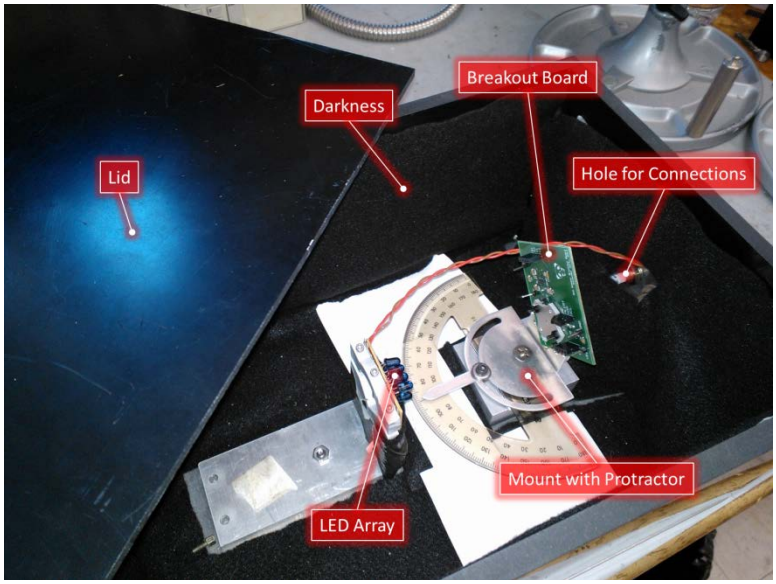

Figure 4-5: Testing apparatus for the sun sensors

Despite its usefulness, the dark box had its fair share of problems for this test. First, the mount was designed for the one-inch-diameter circular PCB of the OLite-2 sun sensor, not the larger, rectangular breakout board. Second, the mounting plate was made of aluminum, and it would short-circuit the pins on the back of the board. Third, the LED array was made by hand so

some of the directionality intent was defeated by the LEDs not being perfectly flat against the board. Finally, the positioning along the protractor is done by hand and is thus the resolution is limited to about 1°. These issues were shown to have minimal effect on the testing results.

Because of the way the mounting holes are located on the breakout board and on the mount, the sensor is mounted at a 45° angle. Therefore, when the system is rotated left and right, both the X and Y axes of the PSD rotate diagonally. This setup actually proved to be more convenient because both axes could be tested simultaneously, which worked well because the breakout board only fit onto the mount one way. Once the dark box issues were resolved, the breakout board was tested from −45° to +45° in 5° increments. Of the four sets of digital data used in the algorithm to generate the sun vector, ϕ and θ are the best figures of merit to examine for this test.

The first test provided somewhat promising results. As shown in Figure 4-6, both ϕ and θ gave results that look ideal, except they are offset from the center by about 20°, ten times the acceptable error. After a brief look to ensure the error was not due to the software, a visual inspection of the circuit found that the PSD was not soldered perfectly flat onto the board; this was quickly remedied. The experiment was then reset and redone. Unfortunately, the results were actually worse than the first attempt, namely in θ (Figure 4-7). An investigation into the source of this error eventually led to a revelation that starts with a look at the X and Y offset values. Figure 4-8 shows the computed offset values for the test. Since ϕ and θ are calculated directly from these two values, the start of the debugging back-trace led here first. Ideally, both the X and Y plots should cross the horizontal axis when the board is incident to the LED array; clearly this is not the case as X is offset by 0.532 mm and Y by 0.286 mm.

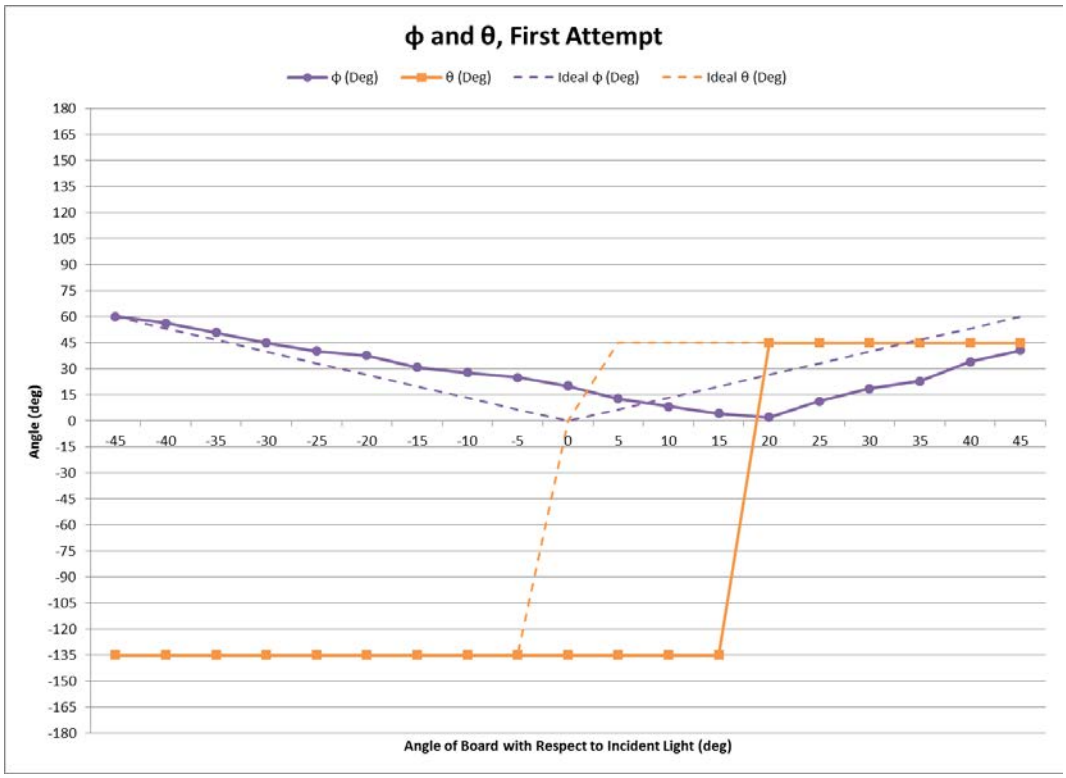

Figure 4-6: ϕ and θ as processed from the first attempt of dark box testing

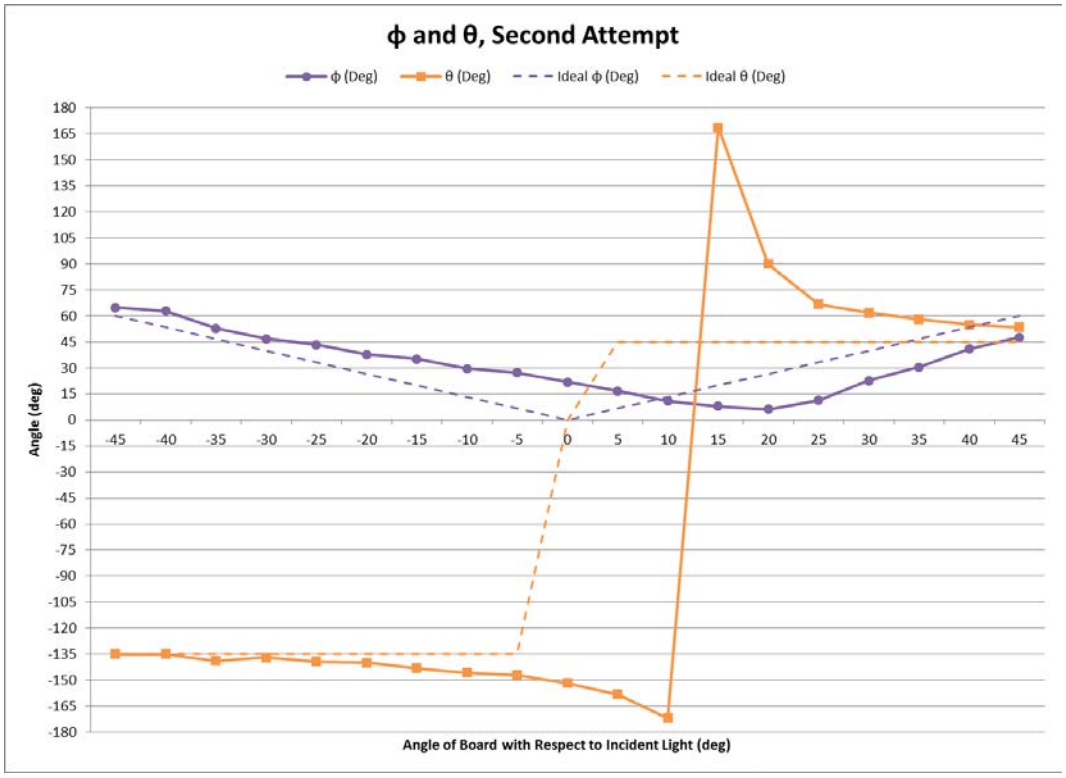

Figure 4-7: ϕ and θ as processed from the second attempt of dark box testing

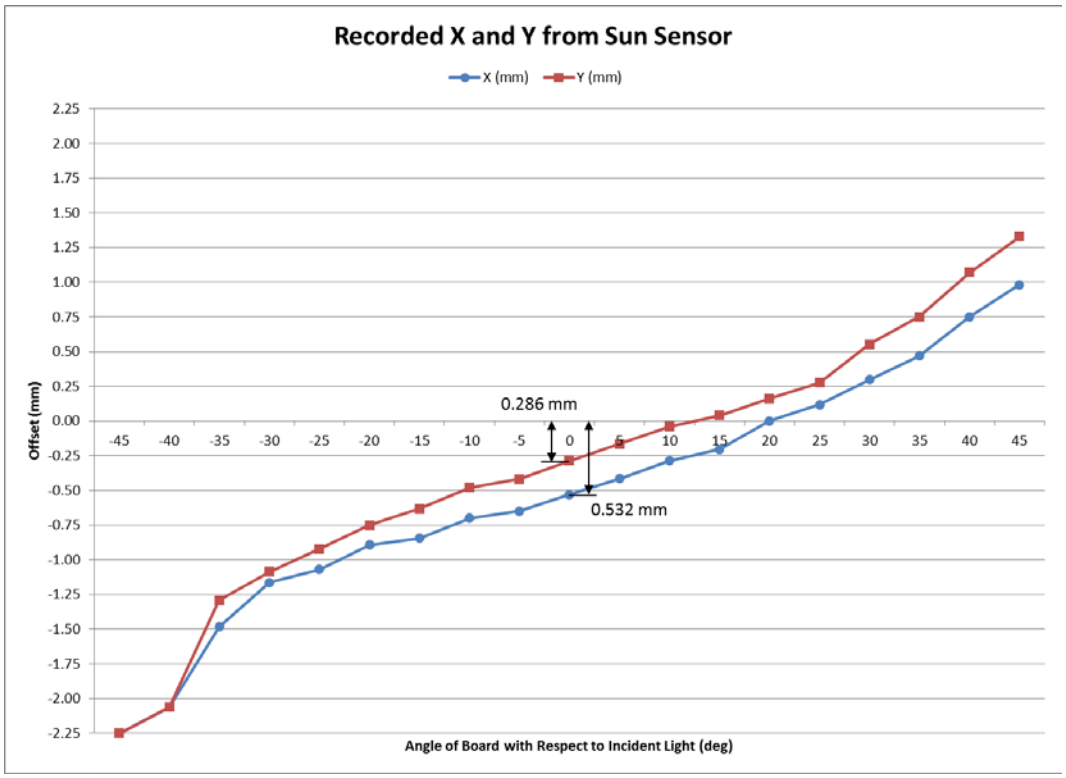

Figure 4-8: X and Y values from the second dark box test, with annotations

Before continuing the investigation, the entire *X* and *Y* data sets were offset so both were zero when at 0°. The resulting plot of ϕ and θ is shown in Figure 4-9. Without question, this plot is incredibly close to the ideal case. Thus, the resulting error appears to be caused by a linear offset in the zero position of the sensor. A close look at the PSD under a microscope confirmed this theory. Though perfectly flat, the PSD was not centered along its silkscreen in either the horizontal or vertical direction (Figure 4-10). When examined with a micrometer (Figure 4-11), the PSD was horizontally off-center by 0.11 mm. When split between a $+X$ and $-X$, the offset would be 0.055 mm. A similar approach gave a *Y* offset of 0.03 mm. Both of these are a factor of ten from being almost exactly the offset discovered by testing the sensor. Thus, this investigation showed that the source of error can be eliminated by calibration.

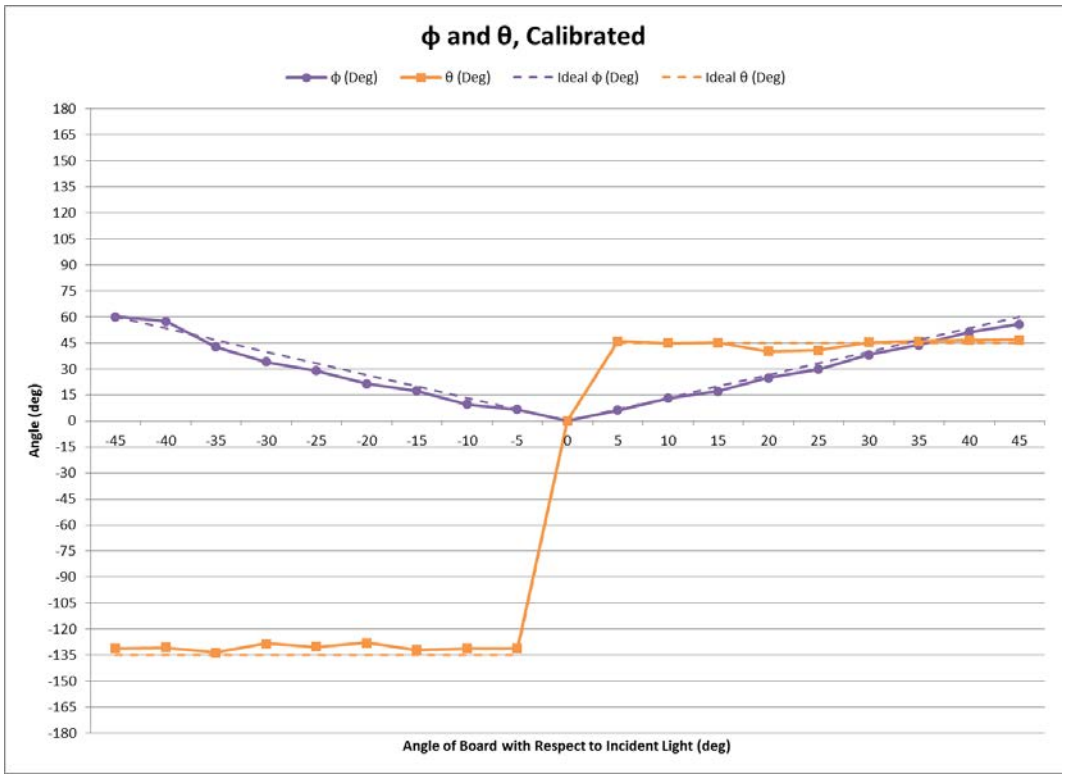

Figure 4-9: ϕ and θ when X and Y are zeroed at the incident angle

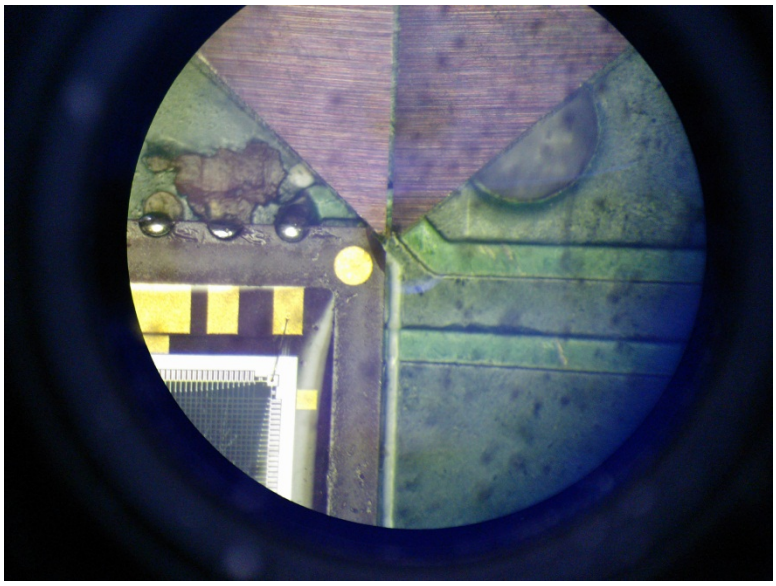

Figure 4-10: Physical offset of PSD on board as viewed from under a microscope

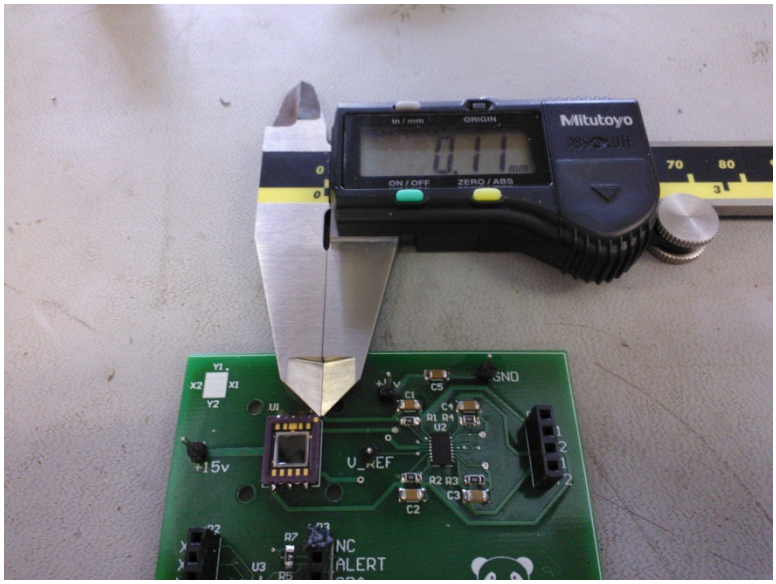

Figure 4-11: Physical X offset of PSD on board

4.2.1.2 Axis Linearity

The previous test also yielded the data for another measure: how the PSD behaves as it approaches its angular limits of operation. As discussed in Section 2.3, the sun sensor can output reliable data between ±49.5° in theory, but the PSD's position detection error slightly increases linearly outward from the center while in this valid region. Figure 4-12 shows data that confirm this. As the spotlight gets farther from the center, the reported position of the spotlight drifts increasingly farther out from its actual position. Heavily impacted by this error is ϕ because its calculation involves the sum of squares of X and Y. A mere 0.1-mm offset in just one of the values results in a 1.2° error in ϕ, and a 0.1-mm offset in both values results in a 2.5° error in ϕ.

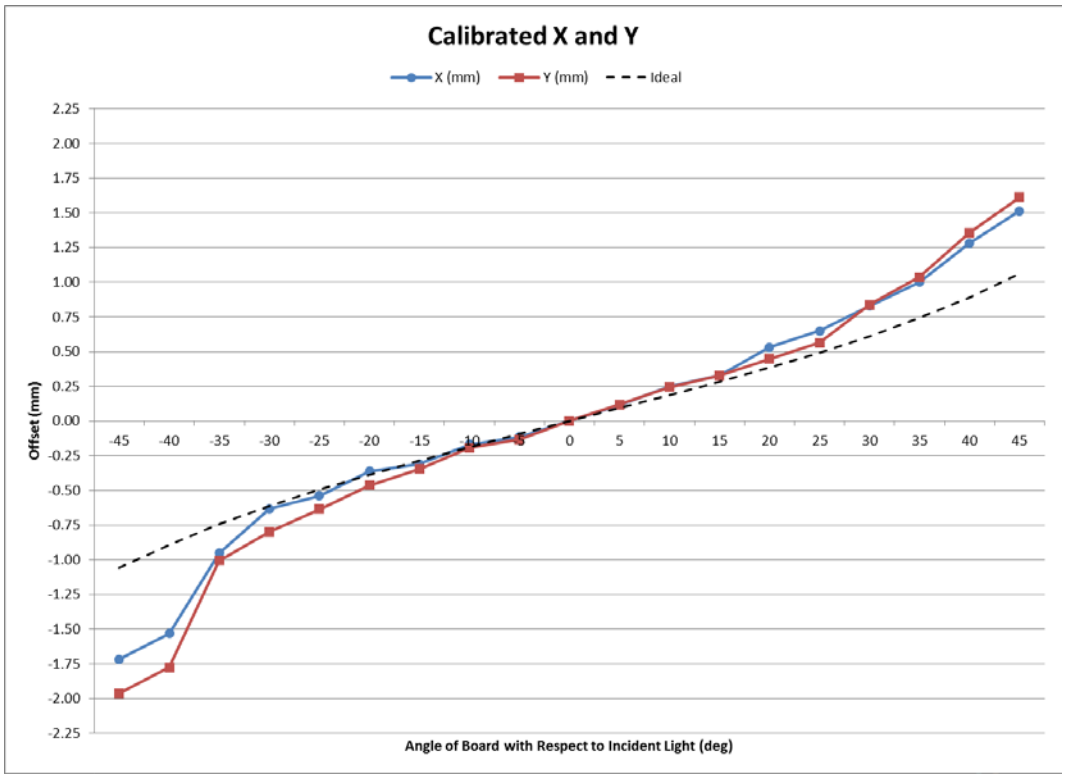

Figure 4-12: Position detection error of X and Y

Another test of linearity is how linear X and Y are with respect to each other. This is especially important in calculating θ because its calculation involves the inverse tangent of their ratio. Despite the position detection error away from the origin, θ remains largely unaffected if and only if the error is roughly the same for both X and Y. Figure 4-13 shows that this is indeed the case. Thus, position detection error is not worrisome for θ.

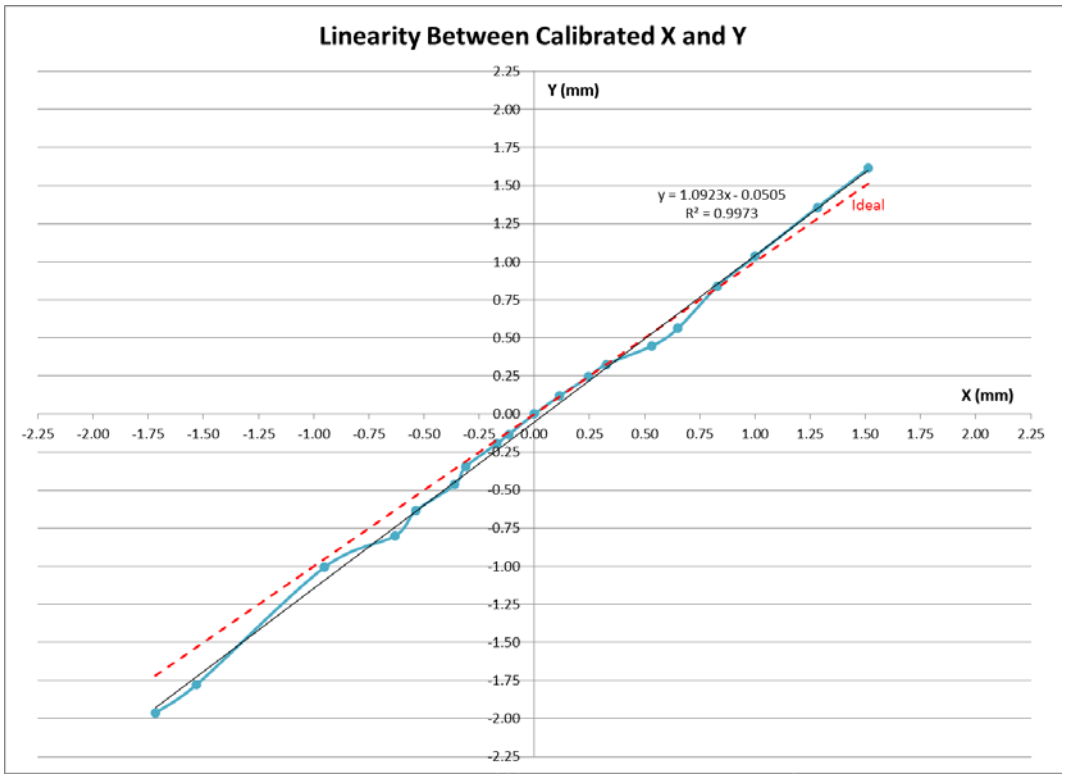

Figure 4-13: Linearity between X and Y

4.2.1.3 I^2C speed compatibility

As part of the design requirements, the sun sensors have to operate at an I^2C clock speed of 400 kHz. Figure 4-14 shows an oscilloscope capture of SCL, the I^2C clock line of the ADC, running at slightly above 400 kHz. The circuit continued to operate without issue at this speed. However, at this speed, SCL is noticeably affected by "sawtoothing." To remedy this, smaller-valued I^2C pull-up resistors were used on the final sun sensor board.

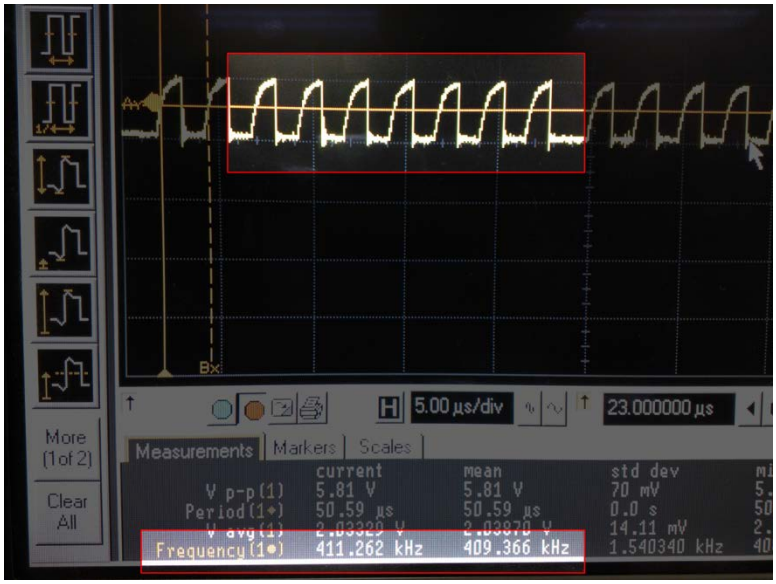

Figure 4-14: I^2C clock line at 400 kHz

4.2.1.4 Power requirements

A very important design requirement is the sun sensor power budget. The whole system is not allowed to use more than 75 mW (excluding thermal circuitry). The power draw of a single sun sensor is affected most by whether or not it is currently facing the sun because the PSD outputs more current when sensing brighter light sources. Consequently, the sun sensors were tested for maximum power draw when in darkness and when facing the sun. In darkness, the combined power draw of the PSD's dark current, the op-amp, and the ADC (including communication) was 2 mW. To mimic the sun, the sensors were placed in front of a halogen light with an irradiance mildly greater than the 1300 W/m^2 expected in orbit. The maximum power draw ever experienced in this situation was 13 mW for a brief moment every so often.

4.2.2 Conclusions

With the close of this series of experiments, the selected design was shown to be not only viable, but actually quite effective for several reasons. First, though the system gives dramatic errors if the PSD is not positioned precisely on the board (i.e., when soldered by hand), these errors can be removed by calibration. In the complete system, each sensor will have to be calibrated individually and documented so that the software can correctly match each sensor with its calibration constants (Section 3.6). Second, even though the PSD has an increasing position detection error outward from its center, this can mostly be ignored for θ and only affects ϕ considerably near the edges of detection. Finally, the system uses very little power. Theoretically, if all five sensors were facing the Sun at the same time, the system would still use 10 mW less than the maximum allowed. In reality, this value will never be reached because only one sensor can actively face the sun at a time for an extended duration because of each sensors field-of-view. The inactive ones will only output photocurrents if sensing the albedos of the Earth or Moon (refer to Section 2.3 for more information on albedos).

4.3 NOISE TEST

A noise profile is essential in assessing the precision of a sensor. Section 4.2.1.1 verified that the system produced reliable digital outputs for given analog inputs whereas this section evaluated the reliability of those analog inputs. The core question explored in this test was whether or not the sun sensor can output results within a ±2° precision.

4.3.1 Procedure and Results

The procedure for this test was very simple: place the breakout board in the dark box at an arbitrary angle and log sun vector data for 48 iterations; the results are plotted in Figure 4-15.

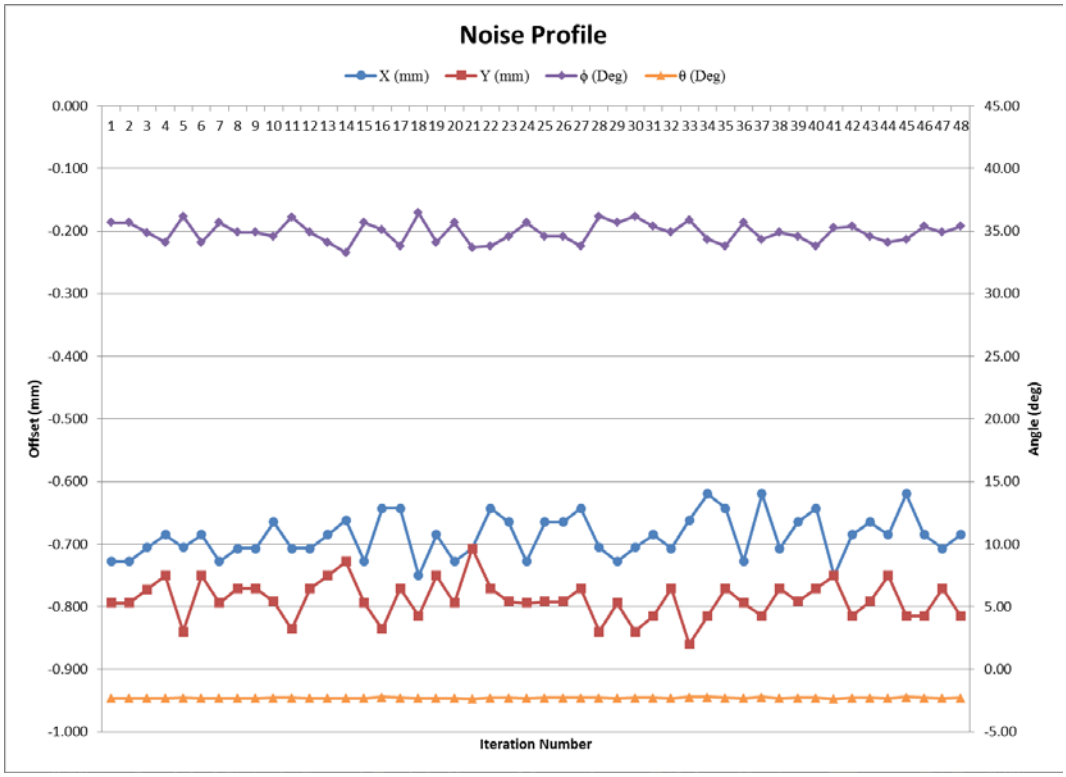

Figure 4-15: Noise Profile

Based on the test, the circuit produces a rather noisy signal even after the lowpass filters (refer to Section 3.3). Table 4-4 shows that, though θ is largely unaffected, ϕ has an error of 3.177° in this trial. The total angular error then, given by

$$Error = \sqrt{\phi_{difference}^2 + \theta_{difference}^2}, \quad 4\text{-}1$$

is 3.180°, greater than the allowed precision error. To bring this error down, a simple software mean filter was implemented (Section 3.6). For the purposes of this experiment, the filter was set to take twenty *X* and *Y* readings and average them. To clarify, this is not a moving-average filter, so every averaged sun vector requires twenty new iterations and thus takes twenty times longer to

compute. The reasoning behind this decision is that in practice, the satellite will not be requesting the sun vector too often. This fact makes a moving-average filter impractical because the inputs will vary dramatically and lets the sun sensors take more time to work because their infrequency limits their performance impact on the system. In theory, the entire operation should take no more than 1 ms based on the I^2C bus speed and the number of bytes sent along the bus to complete twenty reads (based on Section 3.2.3). The resulting statistics are shown in Table 4-4 and the iterations are plotted in Figure 4-16. This tactic brought the error down to about 1°, more than sufficient to satisfy the precision requirement.

Table 4-4: Noise statistics with and without filtering

		Mean	Min	Max	Difference
No Filter	X (mm)	-0.687	-0.750	-0.620	-0.130
	Y (mm)	-0.789	-0.860	-0.707	-0.153
	ϕ (Deg)	34.904	33.264	36.442	-3.177
	θ (Deg)	-2.288	-2.356	-2.221	-0.135
	Error (deg)				**3.180**
20-Point Mean Filter	X (mm)	-0.758	-0.774	-0.742	-0.032
	Y (mm)	-0.870	-0.891	-0.852	-0.039
	ϕ (Deg)	37.562	37.083	38.104	-1.020
	θ (Deg)	-2.288	-2.301	-2.276	-0.025
	Error (deg)				**1.021**

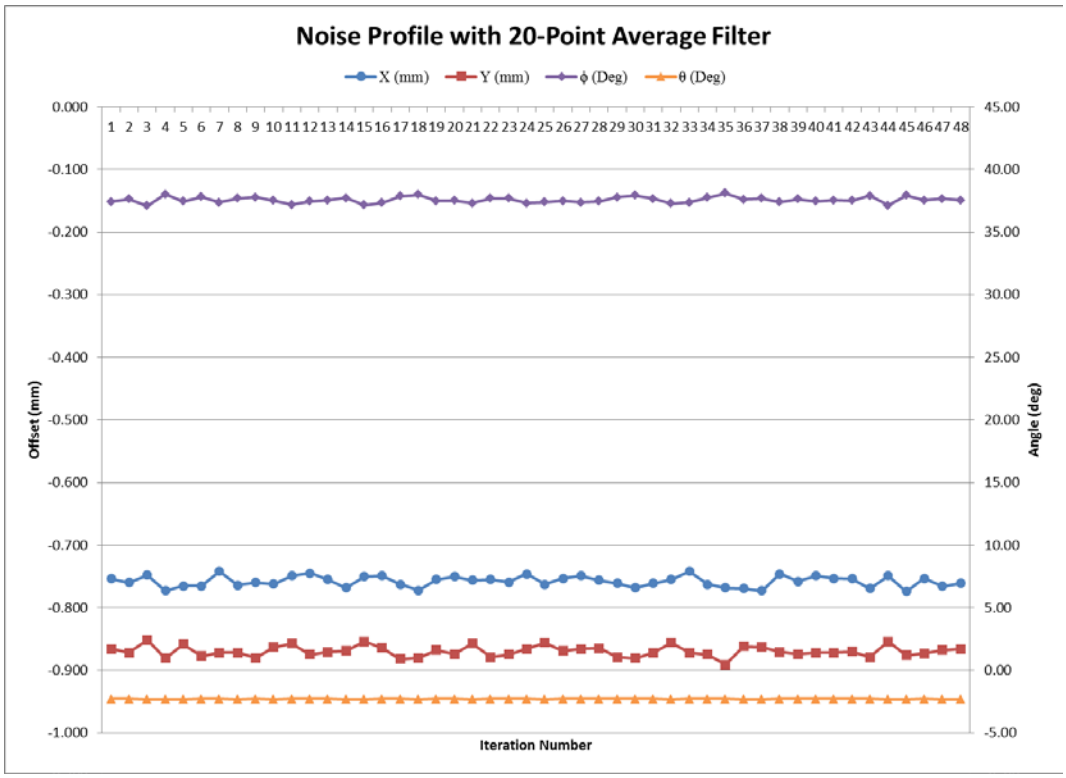

Figure 4-16: Noise profile with software mean filter

4.3.2 Conclusions

Due to the software filter, the precision of the sun sensor can be controlled. A twenty-point average gave a precision of about 1°, so a smaller filter could be used if speed becomes an issue. As of now, the software has full functionality to change the number of points to average with any request. However, averaging will probably be unnecessary in the final implementation. Section 4.5 details how the final board showed excellent noise immunity.

4.4 Thermal-Vacuum Test

The difficulty with designing space systems is the environment in which they will operate. Not only does the lack of atmosphere create a near-vacuum environment, it also lacks the

temperature regulation of convection. Consequently, a system will experience high heat input when in sunlight and extreme cold when in eclipse. To have any hope of surviving this environment, the sun sensors had to survive thermal cycling in a near vacuum.

4.4.1 Procedure and Results

Figure 4-17 shows the test setup inside the vacuum chamber used in this experiment. Essentially, the LED array from the dark box and the breakout board were mounted to the metal base plate inside the chamber. The circuitry was connected to power supplies outside the chamber via a vacuum connector (Figure 4-18). The MBED was located outside the chamber to log data coming from the sensor. With the chamber sealed, the system was pumped down from 733 Torr (the room pressure) to approximately 0.06 Torr. While the pump was not running, the chamber repressurized very slowly. If the pressure ever exceeded 1 Torr, the system was pumped back down. Interestingly, the sun sensor had difficulty outputting results during pumping, but resumed normal operation once pumping finished.

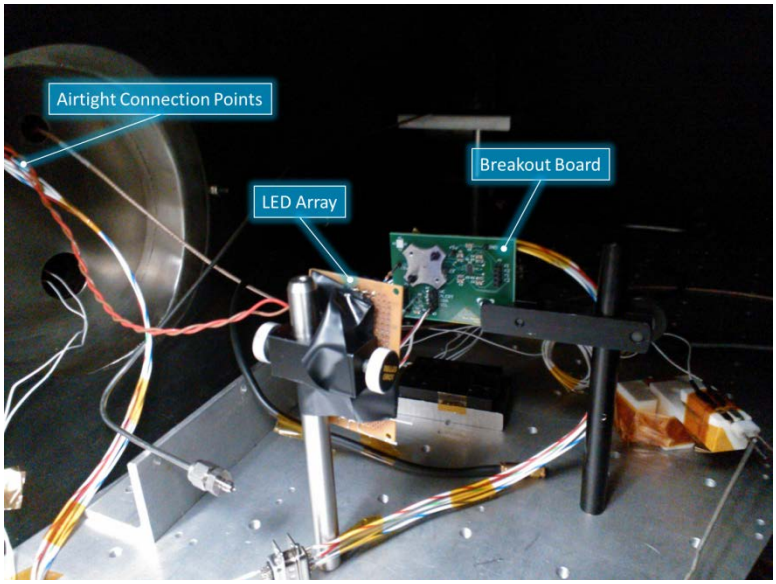

Figure 4-17: Test setup inside vacuum chamber

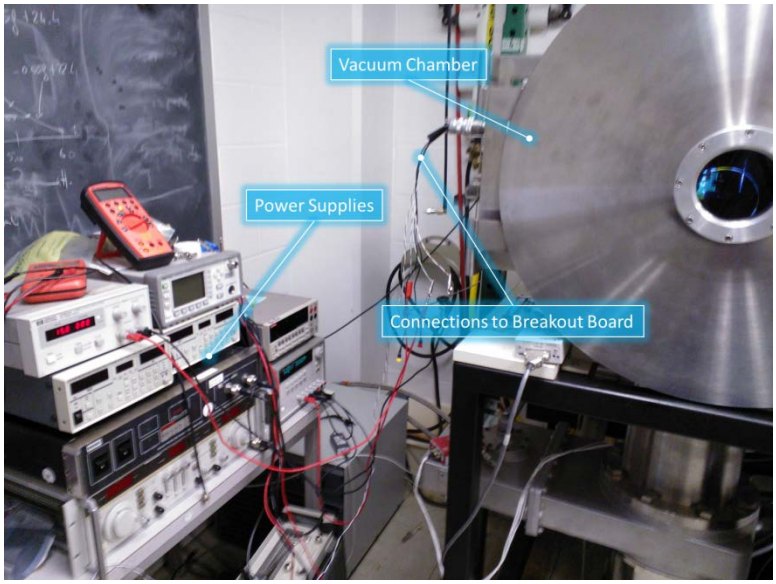

Figure 4-18: Connections to power supplies from vacuum chamber

Thermally, the design requirements, defined in Section 3.1, require that the sensors survive from −15°C to 60°C. The vacuum chamber has a temperature control unit (Figure 4-19) that could both chill and heat the chamber. Because of complications that arose from an earlier incomplete thermal test that was first chilled, the system was heated first for this trial. Starting at 22.5 °C (the room temperature), the chamber was heated up to 60 °C with measurements taken every 5 °C. When the system was set to cool, it rose up to 63 °C before starting descent. Data was collected every 1 °C above 60 °C because these temperatures were outside the rated operating temperature of the PSD. For cooling, the regulator was set to −15 °C with measurements taken every 5 °C once the system cooled down to 20 °C. Unfortunately, the unit could not chill the chamber down below 8 °C. Table 4-5 summarizes the resulting angles during this experiment.

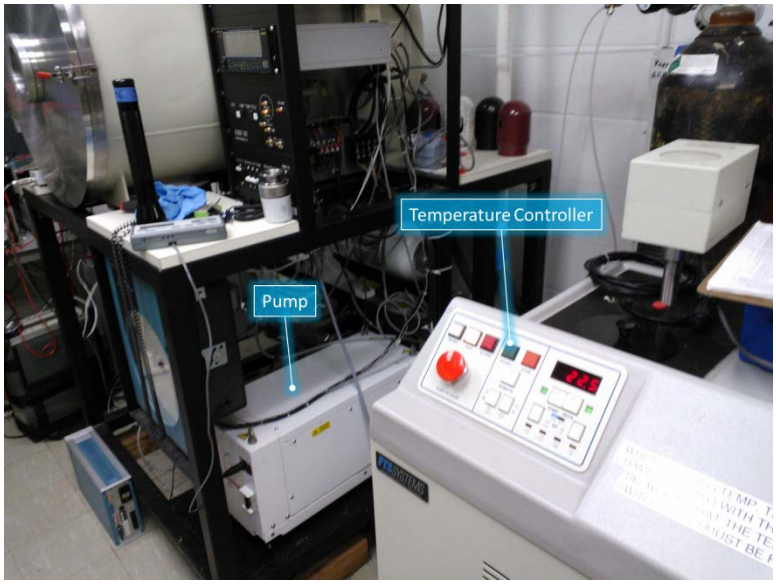

Figure 4-19: Temperature regulator of the vacuum chamber

Table 4-5: Angle outputs during thermal vacuum testing

Temperature (°C)	Pressure (Torr)	ϕ (Deg)	θ (Deg)
22.5	733	31.7	161.6
22.5	0.060	31.7	166
-8.0	**0.736**	**32**	**-171.9**
-5.0	0.638	32	-171.9
0.0	0.283	32	-171.9
5.0	0.933	29.2	171.9
10.0	0.810	29.2	171.9
15.0	0.690	29.2	171.9
20.0	0.610	29.2	171.9
25.0	0.127	32.9	171.9
30.0	0.160	32.9	171.9
35.0	0.219	29.3	166
40.0	0.290	29.3	166
45.0	0.373	29.3	166
50.0	0.472	29.6	161.6
55.0	0.591	29.6	161.6
60.0	0.733	29.6	161.6
61.0	**0.790**	**30**	**158.2**
62.0	**0.813**	**30**	**158.2**
63.0	**0.841**	**30**	**158.2**

4.4.2 Conclusions

The sun sensor successfully survived this test, indicting a much stronger case that the circuit can survive orbit, although a thermal cycling test with many more hot–cold cycles should be performed. More importantly, this test showed how changes in the environment affect the sun vector. For an unexplained reason, the sun sensor outputs several null vectors when in a rapidly depressurizing setting. Thankfully, the sensor will not undergo this in flight. When the pressure was stable, both ϕ and θ remained steady regardless of the pressure. Similarly, ϕ remained unaffected by changes in temperature. θ, on the other hand, experienced a thermal drift such that increasing temperature rotated θ clockwise and vice versa. This could present a problem in orbit because the satellite will enter sunlight chilled from eclipse and continue heating until reentering eclipse. Thankfully, the final sun sensor will have thermal circuitry to monitor and control the temperature (Section 3.5.1). Further testing could define a temperature-dependent offset value to apply to θ in software (Section 5.3).

4.5 Verification of the Final Board

All of the other tests in this chapter were research for the design of the final sun sensor board (Section 3.5). This last test was a verification that the final board worked and a noise profile study. Since the schematic of the final board was almost identical of that of the breakout board, the only reason the final board would not work correctly would be due to the layout. Therefore, if the verified breakout board and the final board give the same output under the same conditions, the final board can be trusted as operating correctly.

4.5.1 Procedure and Results

Once soldered, the final board was paired with a pinout board designed for the sun sensor's 10-pin connector (see Section 3.5). Next, the breakout board was mounted in the dark box and tested at −20°, −10°, 0°, 10°, and 20°. Afterwards, the final board was mounted and tested in the same manner. The only issue with this approach is that the final board's mounting holes are rotated 45° from those of the breakout board. As a result, whereas the breakout board tested both X and Y along the $\theta = 45°$ line, the final board tested each axis independently.

At this point, the final board was giving odd results by reporting that both axes were moving along the $\theta = 45°$ line—clearly incorrect. In an effort to isolate the cause, the board was qualitatively tested with a flashlight. This test hinted that the cause for the suspicious data was the fourteen-LED array. The LED array was replaced with a powerful single-LED flashlight (to better mimic the point-source nature of the Sun) that was mounted in the actual mounting apparatus instead of with duct tape.

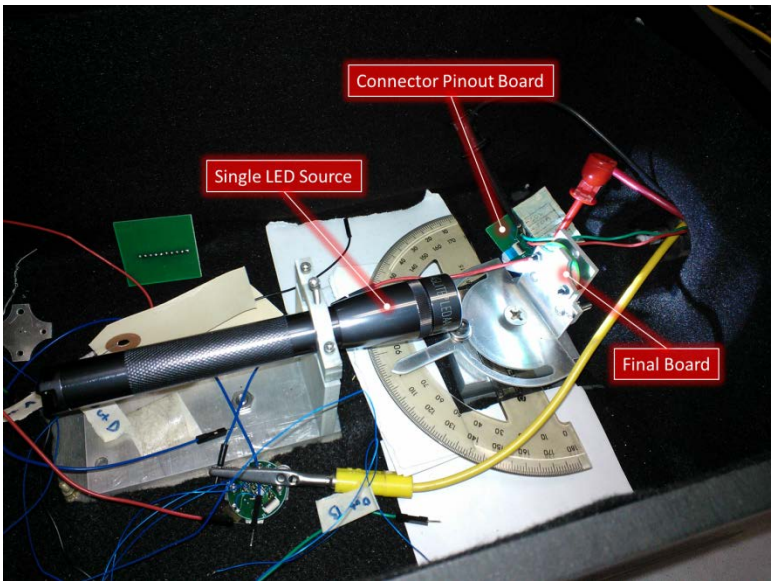

Figure 4-20: Dark box with new light source testing the final board

The resultant data were very promising. Not just for the final board, the breakout board also showed great improvement over previous testing. Because of this, a new noise profile was determined for the final board before continuing with the experiment. Figure 4-21 and Table 4-6 show how clean the outputs were with this new light source. Overall, the error (as determined by Equation 4-1) for the final board is less than ±0.3°.

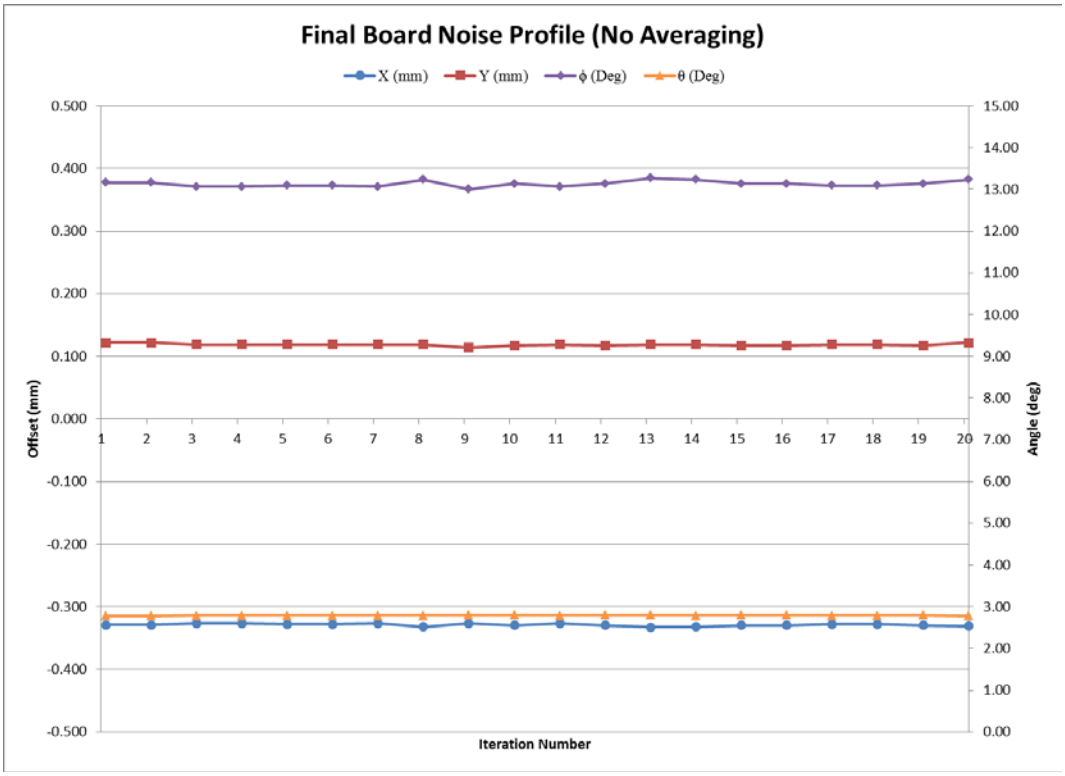

Figure 4-21: Final board noise profile without averaging

Table 4-6: Final board noise statistics

		Mean	Min	Max	Diff
Final Board (No Averaging)	X (mm)	-0.329	-0.333	-0.327	-0.006
	Y (mm)	0.119	0.114	0.122	-0.008
	ϕ (Deg)	13.131	13.000	13.265	-0.265
	θ (Deg)	2.795	2.786	2.806	-0.020
	Error (deg)				**0.266**

Table 4-7 details the results of the verification. Listed first are the results of the breakout board followed by those of the *X*-axis and *Y*-axis of the final board. The boards were mounted along the noted *X* and *Y* directions, when viewed from the flashlight, and rotated left and right. The *X* and *Y* data were then calibrated using a simple (though not perfectly accurate) scheme that shifted the data to zero them at the incident angle. The ϕ data show that the final board agrees with the breakout board, and the θ data match the line of rotation each board underwent: −135° to 45° for the breakout board, 0° to 180° for the final board's *X*-axis and −90° to 90° for the final board's *Y*-axis.

Table 4-7: Final board data compared to breakout board

	Angle	X (mm)	Y (mm)	Cal. X	Cal .Y	ϕ (Deg)	θ (Deg)
Breakout Board	-20	-0.485	-0.485	-0.314	-0.292	15.95	-137.08
	-10	-0.327	-0.343	-0.156	-0.150	8.21	-136.12
	0	-0.171	-0.193	0.000	0.000	0.00	0.00
↖X	10	-0.025	-0.060	0.146	0.133	7.50	42.33
Y↙	20	0.154	0.114	0.325	0.307	16.60	43.37
Final Board (X-Axis)	-20	0.254	0.076	0.392	-0.021	14.67	-3.07
	-10	0.041	0.082	0.179	-0.015	6.83	-4.79
	0	-0.138	0.097	0.000	0.000	0.00	0.00
Y↑	10	-0.346	0.115	-0.208	0.018	7.92	175.05
→X	20	-0.541	0.115	-0.403	0.018	15.05	177.44
Final Board (Y-Axis)	-20	0.140	-0.304	-0.014	-0.370	13.87	-92.17
	-10	0.150	-0.111	-0.004	-0.177	6.73	-91.29
	0	0.154	0.066	0.000	0.000	0.00	0.00
↑X	10	0.162	0.236	0.008	0.170	6.47	87.31
Y←	20	0.164	0.453	0.010	0.387	14.47	88.52

4.5.2 Conclusions

The final board performed better than expected. With the removal of the sporadic light source, the system's precision increased ten-fold. This test also showed the final board is working without any major issues.

Chapter 5 | CONCLUSION

In the broadest sense, the sun sensor project was a success. Every design requirement was met or exceeded, and the OSIRIS satellite can now implement a sun sensor system with confidence. However, the project was not perfect. The accelerated design timeline forced several shortcomings in the design and left plenty of room to extend the project.

5.1 ACCOMPLISHMENTS

At the outset, the goal was to develop a working set of sun sensors with a full software class that simplifies implementation to a few function calls. When the sun sensors are integrated with the satellite, they will work at the I^2C speed of the rest of the system and use only about half of their power budget. Upon calling a single function, GNC can receive a sun vector oriented along the geometry of the satellite that is accurate to less than ±0.3° for use with the TRIAD system. For software, CDH simply has to create a sun sensor object to control the sun sensors. For normal operation, after the class object is created, each sun sensor only needs to be issued a reset command to initialize it. Afterward, a single function call will return a 3-D unit vector pointing to the Sun. If needed, several debugging functions are also available should any issues arise. Finally, the sensors survived the same thermal vacuum test the OSIRIS satellite will have to pass.

Overall, the sun sensor hardware is ready-to-fly and the software is ready-to-implement. Thanks to the modular design, the system can be debugged easily, malfunctioning units can be replaced easily, and most functionally can be changed in software easily. This project also generated extensive documentation for future users to quickly learn how the sun sensors operate.

5.2 SHORTCOMINGS

Unfortunately, the system is not without its shortcomings. Whether the reason was another system not being ready to make a design decision or the deadline for the sun sensors approaching, a host of issues remain unresolved.

To start, the software class has no method to differentiate which sun sensor to act upon. Several functions have an input parameter to operate on a specific sun sensor (+*X*, –*X*, +*Y*, –*Y*, or +*Z*), but as of the time of this writing, the class simply acts on the +*X* sensor. This is because CDH is responsible for interfacing with the sun sensors. This same reason is why the I^2C function calls refer to a dummy class that does virtually nothing. Once this system of communication is designed, someone will have to edit the sun sensor class to implement it.

Similarly, determining which sun sensor is facing the Sun is lacking. Currently, the system simply picks which sensor is receiving the brightest light input. In an environment with only one light source or with one light source much brighter than the others, this check will work. However, there is no guarantee that the albedos of the Earth and Moon will always be dimmer than the Sun until further research is done to either confirm or deny this assumption. Even if the research does confirm it, the possibility of an unexpected bright reflection (dazzling) or a shadow off another system in orbit would still render this selection process inaccurate.

On the hardware side, the thermal circuitry was not tested at all. The Thermal functional group does not currently have the specifics on what to expect thermally. The only absolute is the TMP175. As a result, the optional address resistors for the TMP175 take up board space but only allow for one other I^2C address should floating pins prove troublesome. Moreover, the footprint for the power MOSFET is a generic footprint of a PMOS in a SOT-23 package. Though this gives the most flexibility for when Thermal finishes calculating the power requirements of the heat resistor, SOT-23 packages rated for more than 1 W will prove very difficult to find if that is what

is required. As for the heat resistor, the footprint is merely two large pads on the side of the board. This limits the heating device to two pins and to a particular space on the board. Consequently, if any of these thermal assumptions are eventually proven insufficient, someone will have to figure out a way to white-wire new thermal support circuitry onto the small board.

On a related note, thermal testing could not reach the desired lower temperature range. Since Thermal will nominally let the system cool to −15°C before the heater is activated [3], −8°C is not close enough say the system will survive. Similarly, though 10^{-2} Torr is small, satellites in orbit experience pressures far below this in the range of 10^{-9} to 10^{-11} Torr [11]. To extend beyond this, no testing or research was done to determine how well the sensors will hold up when exposed to space weather or the conditions present in the ionosphere when it is stimulated for testing. However, this was not a goal for this project, so more information about this topic is in Section 5.3.

5.3 EXTENSIONS

Extensions for the sun sensor project go beyond simply fixing all the shortcomings of Section 5.2. Thanks to the software-heavy nature of the system, several new features could be added anytime in the future. One such feature would take advantage of the TMP175 on the board to generate a temperature-dependent calibration to fix the drift observed in θ during thermal testing. This offset equation would have to be determined empirically because the S5990-01 documentation provides no information on the matter. Similarly, the software could also add a position-dependent calibration to supplement the calibration constants. As noted in [10] and as verified in Section 4.2.1.2, the PSD's accuracy decreases farther from the origin. The best method to fix this would to use the error graph presented in [10] and making adjustments as necessary via

experimentation. The only catch to this approach is that the error graph is for single-dimensional PSDs, so this method hypothesizes each axis can be independently fixed in this way.

As noted at the end of Section 5.2, environmental testing of the sun sensors can continue beyond the simple test of Section 4.4. To start, the thermal chamber has an additional cryogenic pump to test at even lower pressures. This may be the best option because pumping down to orbital pressures on Earth is incredibly difficult. On the other hand, controlled radiation testing is possible at Penn State's Breazeale Nuclear Facility. However, this would take considerable time to research and set up. A better approach is to test the entire OSIRIS system for space-weather compatibility when a final unit is ready. Another small but still needed test is a vibration or shock test. As noted in [1], the biggest concern with vibration or shock is the potential to alter sensor alignment and throw off calibration.

Another possible extension to research is the "single-shot" operating mode of the ADC. Assuming that, under normal circumstances the sun vector is only requested intermittently, the ADCs of the sun sensors do not need to be constantly reading data. This is even truer in eclipse where the sun sensors are useless. Moreover, GNC can eventually determine the satellite's position and orientation using mathematical models based on previous sun sensor readings; at this point, the sun sensors' job is done. In all of these cases, the ADCs would continue to operate continuously, however. In fact, since reading a different PSD output requires reconfiguring the ADC, in idle times the ADCs will all be reading the Y2 values off their respective PSDs. By switching to single-shot mode, this waste of resources can be prevented by having the ADCs only read inputs when requesting a sun vector and then powering down. To save even more power, the entire system can simply be shut off when idle. Testing has shown that the sun sensors go from no power to full operation very quickly.

Finally, a future sun sensor revision should focus on field of view. This revision has a maximum field of view of 91°. This is the only figure of merit commercial sun sensors are

generally better at than the one produced in this thesis. Overall though, this sensor is highly competitive with those on the market. Especially with power consumption because no other sensor even comes close to the maximum of 13 mW consumed by a single sensor.

BIBLIOGRAPHY

[1] Hall, J. M., *et al.*, "Space Sun Sensors," NASA, Washington D.C., NASA SP-8047, 1970.

[2] Winetraub, Y., S. Bitan, Y. Nativ, and A. Heller, "Attitude Determination – Advanced Sun Sensors for Pico-Satellites," Handasaim School, Tel-Aviv Univ., Israel.

[3] Kummar, A., *et al.*, "Preliminary Design Review: OSIRIS LITE 2 Balloon Payload," Penn State SSPL, University Park, PA, 5006-00-1003, 2010.

[4] Hall, C. D., "Spacecraft Attitude Dynamics and Control," Virginia Tech., Blacksburg, VA, 2003.

[5] "CubeSat Design Specification," Cal. Poly., San Luis Obispo, CA, 2009.

[6] *About Us* [Online]. Available: http://www.cubesat.org/index.php/about-us.

[7] (2012) *CubeSat Sun Sensor* [Online]. Available: http:// http://www.cubesatshop.com/.

[8] (2011) *Aerospace* [Online]. Available: http://www.solar-mems.com/en/products/aerospace.

[9] Malone, A., "Sun Sensor Brief Overview," Penn State SSPL, University Park, PA, 2009.

[10] "PSD (Position Sensitive Detector)," Hamamatsu, Bridgewater, NJ, KPSD0001E01, 2002.

[11] *Vacuum Reference Chart* [Online]. Available: http://www.orbitec.com/documents/Orbitec_Vacuum_Reference.pdf.

Appendix A | BOARD LAYOUTS

The following are technical details of the PCB layout behind the breakout board and final board.

A.1 BREAKOUT BOARD

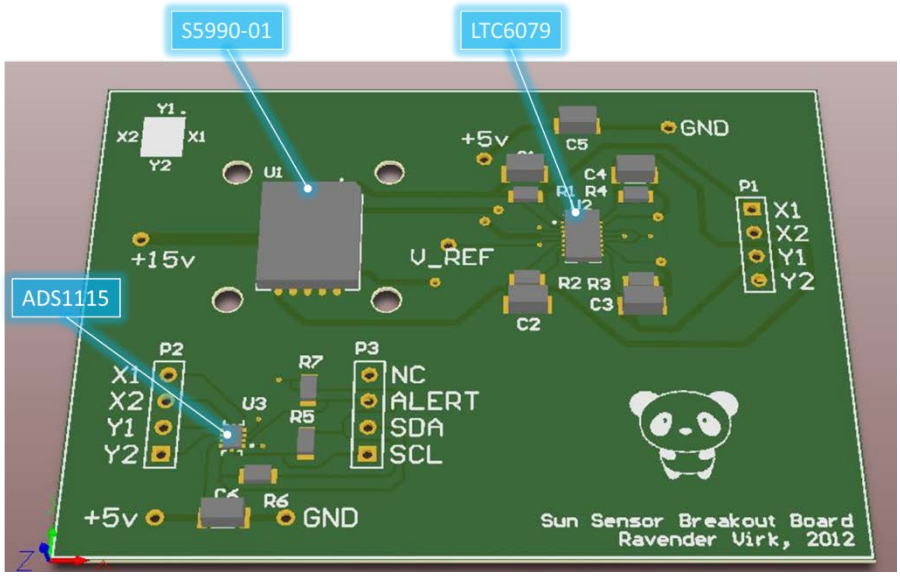

Figure A-1: Breakout board render with annotations

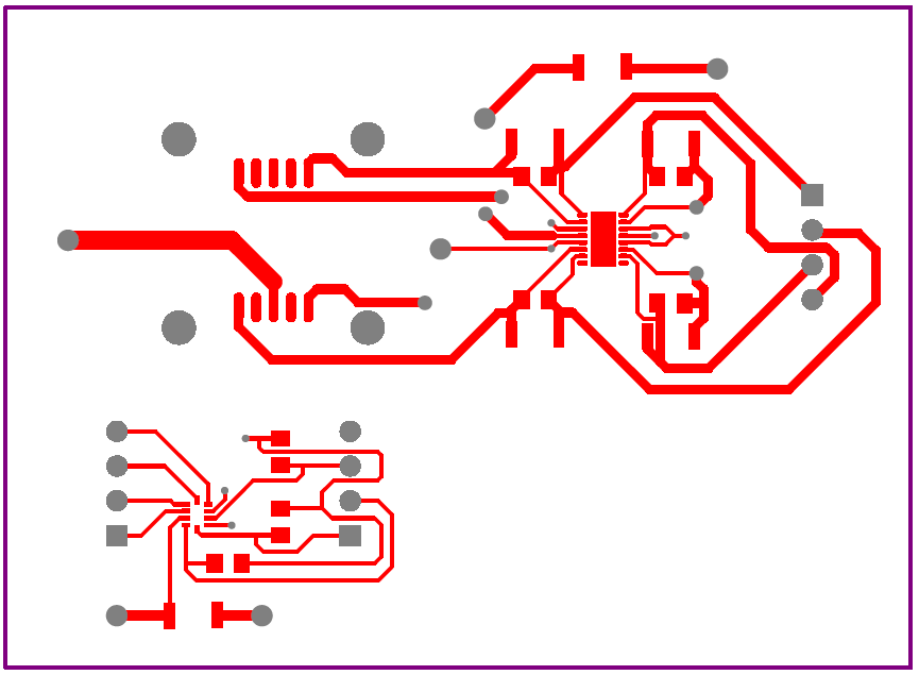

Figure A-2: Breakout board - top layer

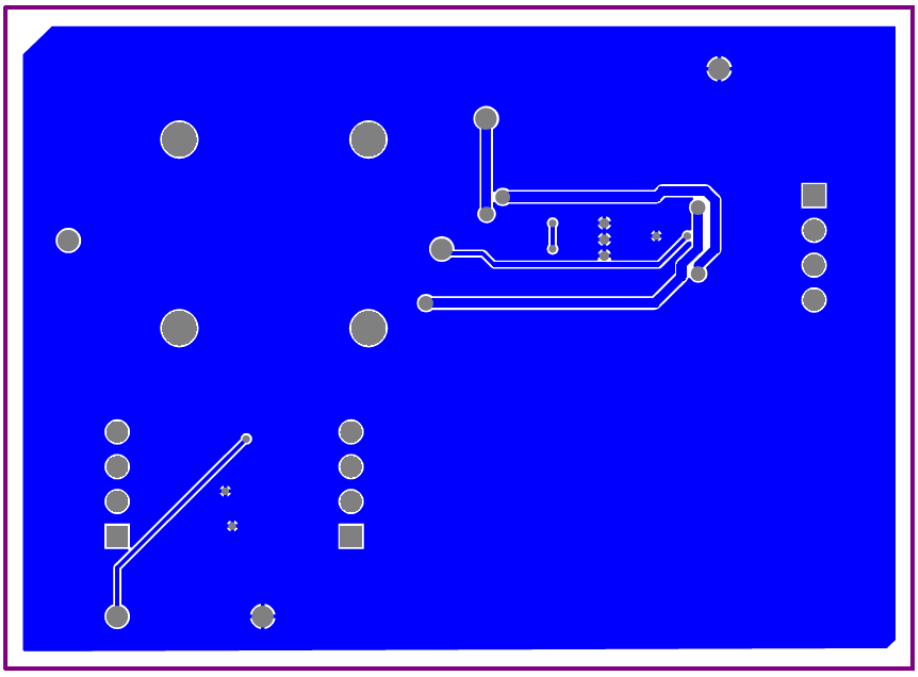

Figure A-3: Breakout board - bottom layer

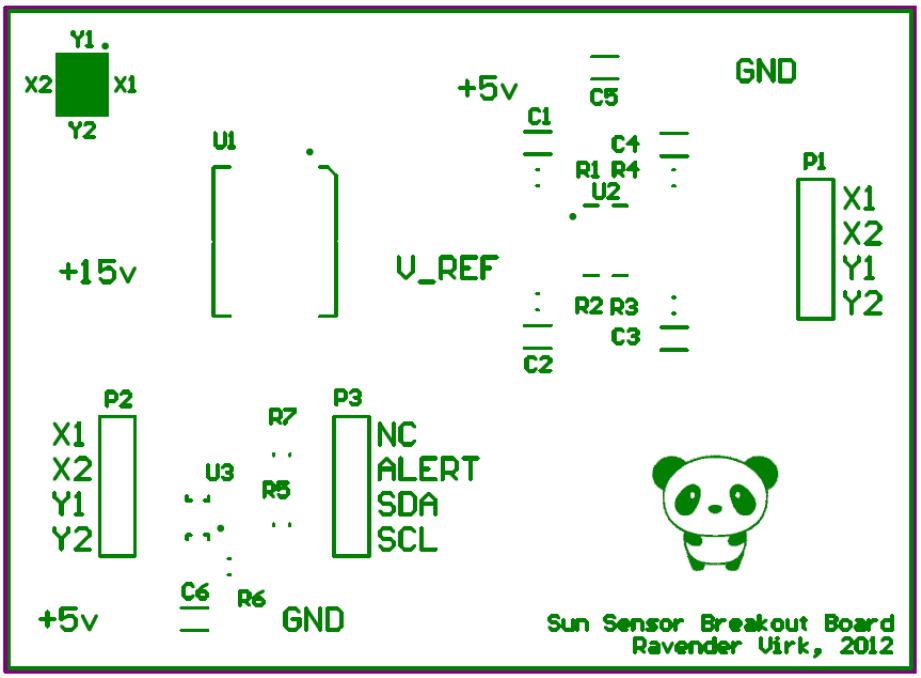

Figure A-4: Breakout board - top silkscreen

A.2 Final Board

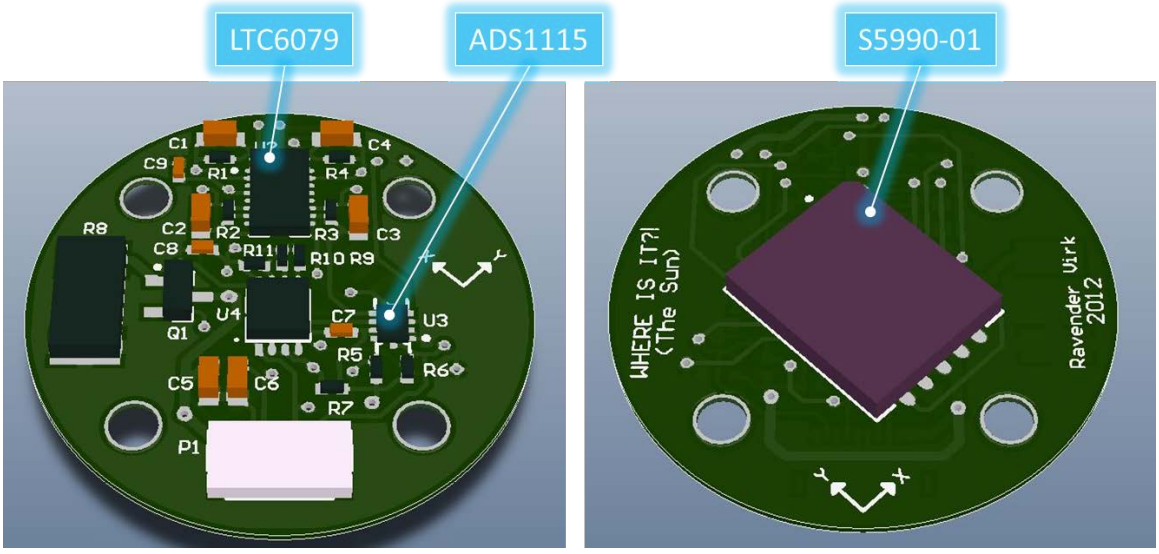

Figure A-5: Final board render, front and back, with annotations

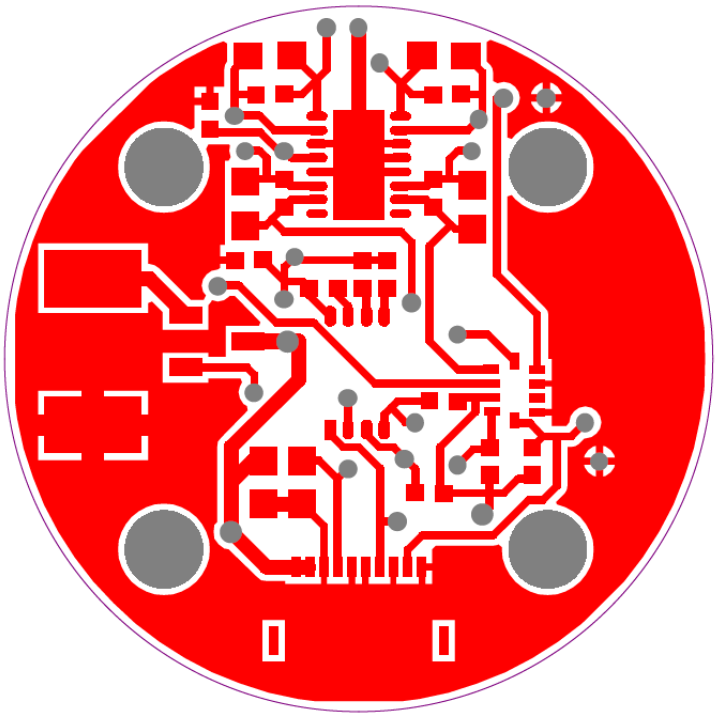

Figure A-6: Final board - top layer

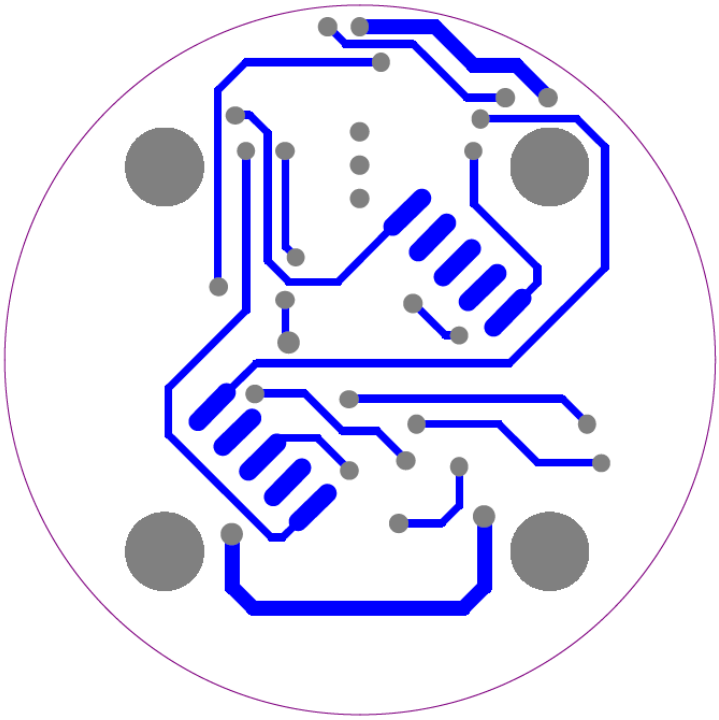

Figure A-7: Final board - bottom layer

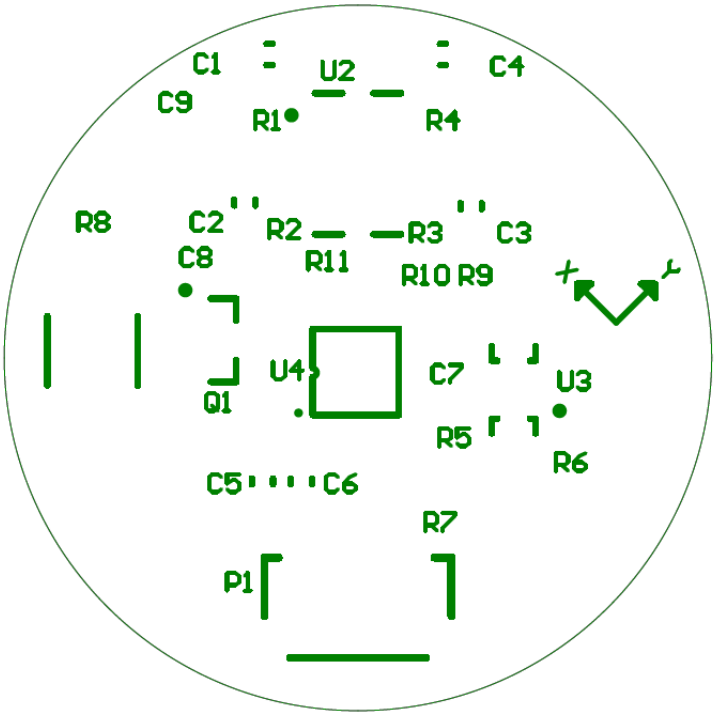

Figure A-8: Final board - top silkscreen

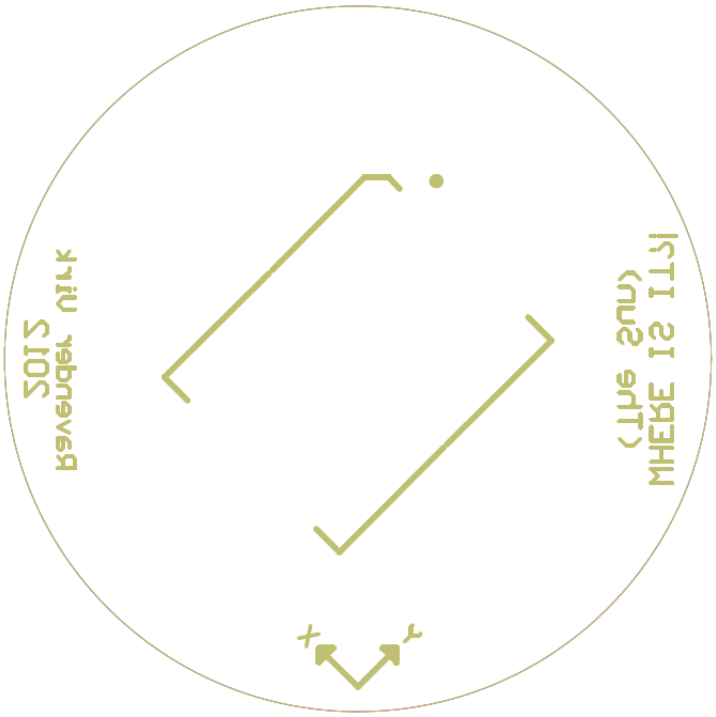

Figure A-9: Final board - bottom silkscreen

Appendix B | SUN SENSOR SOFTWARE FILES

The following are the C++ header and sources files that make up the sun sensor software class.

B.1 SUNSENSOR.H

```
/*-------------------------------------------------------------------
File:          SunSensor.h
Author:        Ravender Virk
               Student Space Programs Laboratory
               The Pennsylvania State University
Date:          October 2012

This is the header file for the Sun Sensor class to be used on the
OSIRIS Satellite. Normal use of this class is as follows:
1. Create a SunSensor class
2. Reset all five Sun Sensors
3. Call "GetSunSensor" as needed.
The remaining functions are helper functions or for debugging. Please
refer to the Sun Sensor documentation for more information.

Revisions made to either "SunSensor.h" or "SunSensor.cpp"
MM/DD/YY   [Name] Changes
========================
11/13/12   [Ravender Virk] Completed the TransformVector function
-------------------------------------------------------------------*/

#pragma once

#include <iostream>
#include "i2cdummy.h" // I2C dummy

// Absolute Constants (do not change)
#define L 0.0045
#define d 0.0015
#define pi 3.14159265359

// ADC Constants
#define FS 6.144
#define MAX 0x7fff

// Calibration
// Refer to Sun Sensor documentation for calibration instructions
#define XplusXOffset -0.000000
#define XplusYOffset -0.000000

#define XminusXOffset -0.000000
#define XminusYOffset -0.000000

#define YplusXOffset -0.000000
```

```
#define YplusYOffset -0.000000

#define YminusXOffset -0.000000
#define YminusYOffset -0.000000

#define ZplusXOffset -0.000000
#define ZplusYOffset -0.000000

using namespace std;

enum SunSensors { Xplus, Xminus, Yplus, Yminus, Zplus };
enum SunSensorRaw { raw_x1, raw_x2, raw_y1, raw_y2 };
enum SunVector { sv_X, sv_Y, sv_Z };

class SunSensor
{
public:
	// Class constructor
	SunSensor();

	/*=== Sun Sensor: Reset ========================================
	Resets the configuration register of the specified Sun Sensor.

	Parameters:
	    sensor: the index of the Sun Sensor to reset
	    ack: [returned] the acknowledge from the I2C communication
	         with the ADC
	==============================================================*/
	void Reset(const int sensor, int& ack);

	/*=== Sun Sensor: Get Raw Data ================================
	Returns the raw x1, x2, y1, and y2 data from the specified Sun
	Sensor.

	Parameters:
	    sensor: the index of the Sun Sensor to read from
	    raw: [returned] { x1, x2, y1, y2 } such that raw[0] = x1,
	         raw[2] = x2, etc.
	==============================================================*/
	void GetRaw(const int sensor, double* raw);

	/*=== Sun Sensor: Select Appropriate Sun Sensor ================
	Looks at the output of all the sun sensors and decides which one
	to use to calculate the Sun Vector. Currently, this is based on
	which sensor is giving the brightest input.

	Parameters:
	    rawXplus: x1, x2, y1, and y2 of Xplus sun sensor
	    rawXminus: x1, x2, y1, and y2 of Xminus sun sensor
	    rawYplus: x1, x2, y1, and y2 of Yplus sun sensor
	    rawYminus: x1, x2, y1, and y2 of Yminus sun sensor
	    rawZplus: x1, x2, y1, and y2 of Zplus sun sensor
	    active: [returned] The index of the appropriate Sun Sensor
	==============================================================*/
	void SelectSensor(const double* rawXplus, const double* rawXminus,
	                  const double* rawYplus, const double* rawYminus,
	                  const double* rawZplus, int& active);

	/*=== Sun Sensor: Transform Sun Vector =========================
	Takes the relative sun vector generated by the active Sun Sensor
	and    transforms it to the absolute sun vector relative to the
```

```
        satellite.

        Parameters:
            active: the Sun Sensor that generated the sun vector
            sunVector: [input and returned] the generated sun vector to
                       transform
        ================================================================*/
        void TransformVector(int active, double* sunVector);

        /*=== Sun Sensor: Get Sun Vector ==================================
        Pools all the Sun Sensors, decides which one to use, averages the
        desired number of readings, and generates the absolute sun vector.
        When averaging, any points which return NaN are not averaged, so
        points2avg is the maximum number of points, not the actual.

        Parameters:
            sunVector: [returned] { X, Y, Z }, unit vector
            point2avg: how many times to average the sun vector, defaults
                       to 20 for reliability
        ================================================================*/
        void GetSunVector(double* sunVector, const int points2avg = 1);

        /*=== Sun Sensor: Get Active Sun Sensor ==========================
        Gets the currently active Sun Sensor, i.e. the last one read from.

        Parameters:
            active: [returned] the currently active Sun Sensor
        ================================================================*/
        void GetActive(int& active);

        /*=== Sun Sensor: Read Configuration Registers of ADC ============
        Reads the two byte configuration register of the ADC via I2C.

        Parameters:
            sensor: the index of the Sun Sensor to read from
            ConfigHi: [returned] bits 15:8 of the ADC config register
            ConfigLo: [returned] bits 7:0 of the ADC config register
        ================================================================*/
        void ReadConfig(const int sensor, unsigned char& configHiRead,
                        unsigned char& configLoRead, int& ack);

    private:
        static const int ADDR = 0x48 << 1;
        static const char CONV_REG = 0x00;
        static const char CONFIG_REG = 0x01;

        //                                      MUX Input[14:12]
        static const unsigned char configHiX1 = (1<<6)|(0<<5)|(0<<4) |
               Gain[11:9]             Mode[8]
               (1<<3)|(0<<2)|(1<<1) | (0<<0);
        static const unsigned char configHiX2 = (1<<6)|(0<<5)|(1<<4) |
               (1<<3)|(0<<2)|(1<<1) | (0<<0);
        static const unsigned char configHiY1 = (1<<6)|(1<<5)|(0<<4) |
               (1<<3)|(0<<2)|(1<<1) | (0<<0);
        static const unsigned char configHiY2 = (1<<6)|(1<<5)|(1<<4) |
               (1<<3)|(0<<2)|(1<<1) | (0<<0);
        //                                    Data Rate[7:5]
        static const unsigned char configLo = (1<<7)|(0<<6)|(0<<5);

        char config[3];
        double raw[4], rawXplus[4], rawXminus[4], rawYplus[4],
```

```
                rawYminus[4], rawZplus[4];
        int active;

        I2C i2c; // I2C dummy
};
```

B.2 SUNSENSOR.CPP

```
/*----------------------------------------------------------------------
File:           SunSensor.cpp
Author:         Ravender Virk
                Student Space Programs Laboratory
                The Pennsylvania State University
Date:           October 2012

This is the body file for the Sun Sensor class to be used on the
OSIRIS Satellite. Please refer to "SunSensor.h" for further
documentation and usage instructions.
----------------------------------------------------------------------*/

#include "SunSensor.h"

SunSensor::SunSensor()
{
        i2c = I2C();
        config[0] = CONFIG_REG;
        config[1] = configHiX1;
        config[2] = configLo;
        active = -1;
}

void SunSensor::Reset(const int sensor, int& ack)
{
        // TODO (CDH): Implement methed to differentiate sensors

        ack = i2c.write(ADDR, config, 3); // I2C dummy

        // Debugging, optional
        if (ack == 0)
        {
                printf("[SS] Sun Sensor %d, Reset: Success\r\n", sensor);
        }
        else
        {
                printf("[SS] Sun Sensor %d, Reset: Fail\r\n", sensor);
        }
}

void SunSensor::GetRaw(const int sensor, double* raw)
{
        // TODO (CDH): Implement methed to differentiate sensors

        char conv[2];

        // Failsafe if Reset has not been called, optional
        config[0] = CONFIG_REG;
```

```
        config[2] = configLo;

        // X1
        config[1] = configHiX1;
        i2c.write(ADDR, config, 3);   // I2C dummy
        i2c.write(ADDR, &CONV_REG, 1); // I2C dummy
        i2c.read(ADDR, conv, 2);      // I2C dummy
        raw[raw_x1] = ((conv[0]<<8)|(conv[1])) * FS / MAX;

        // X2
        config[1] = configHiX2;
        i2c.write(ADDR, config, 3);   // I2C dummy
        i2c.write(ADDR, &CONV_REG, 1); // I2C dummy
        i2c.read(ADDR, conv, 2);      // I2C dummy
        raw[raw_x2] = ((conv[0]<<8)|(conv[1])) * FS / MAX;

        // Y1
        config[1] = configHiY1;
        i2c.write(ADDR, config, 3);   // I2C dummy
        i2c.write(ADDR, &CONV_REG, 1); // I2C dummy
        i2c.read(ADDR, conv, 2);      // I2C dummy
        raw[raw_y1] = ((conv[0]<<8)|(conv[1])) * FS / MAX;

        // Y2
        config[1] = configHiY2;
        i2c.write(ADDR, config, 3);   // I2C dummy
        i2c.write(ADDR, &CONV_REG, 1); // I2C dummy
        i2c.read(ADDR, conv, 2);      // I2C dummy
        raw[raw_y2] = ((conv[0]<<8)|(conv[1])) * FS / MAX;

        // Account for overflow
        raw[raw_x1] = raw[raw_x1] > 12 ? 0 : raw[raw_x1];
        raw[raw_x2] = raw[raw_x2] > 12 ? 0 : raw[raw_x2];
        raw[raw_y1] = raw[raw_y1] > 12 ? 0 : raw[raw_y1];
        raw[raw_y2] = raw[raw_y2] > 12 ? 0 : raw[raw_y2];

        // Debugging, optional
        printf("[SS%d] X1: %f, X2: %f, Y1: %f, Y2: %f\r\n", sensor, raw[raw_x1],
              raw[raw_x2], raw[raw_y1], raw[raw_y2]);
}

void SunSensor::SelectSensor(const double* rawXplus, const double* rawXminus,
        const double* rawYplus, const double* rawYminus,
        const double* rawZplus, int& active)
{
        // TODO (GNC): Implement a better methed to pick correct Sun Sensor if
              desired

        double XplusTotal = rawXplus[raw_x1] + rawXplus[raw_x2] +
              rawXplus[raw_y1] + rawXplus[raw_y2];
        double XminusTotal = rawXminus[raw_x1] + rawXminus[raw_x2] +
              rawXminus[raw_y1] + rawXminus[raw_y2];
        double YplusTotal = rawYplus[raw_x1] + rawYplus[raw_x2] +
              rawYplus[raw_y1] + rawYplus[raw_y2];
        double YminusTotal = rawYminus[raw_x1] + rawYminus[raw_x2] +
              rawYminus[raw_y1] + rawYminus[raw_y2];
        double ZplusTotal = rawZplus[raw_x1] + rawZplus[raw_x2] +
              rawZplus[raw_y1] + rawZplus[raw_y2];

        if ((ZplusTotal >= XplusTotal) && (ZplusTotal >= XminusTotal) &&
              (ZplusTotal >= YplusTotal) && (ZplusTotal >= YminusTotal))
```

```
	{
		active = Zplus;
	}
	else if ((XplusTotal >= XminusTotal) && (XplusTotal >= YplusTotal) &&
		(XplusTotal >= YminusTotal))
	{
		active = Xplus;
	}
	else if ((XminusTotal >= YplusTotal) && (XminusTotal >= YminusTotal))
	{
		active = Xminus;
	}
	else if (YplusTotal >= YminusTotal)
	{
		active = Yplus;
	}
	else
	{
		active = Yminus;
	}
}

void SunSensor::TransformVector(int active, double* sunVector)
{
	if (active == Zplus)
	{
		// No transform needed for +Z sensor
		return;
	}

	double x = sunVector[sv_X];
	double y = sunVector[sv_Y];
	double z = sunVector[sv_Z];

	switch(active)
	{
	case (Xplus):
		sunVector[sv_X] = z;
		sunVector[sv_Y] = x;
		sunVector[sv_Z] = y;
		break;
	case (Xminus):
		sunVector[sv_X] = -z;
		sunVector[sv_Y] = -x;
		sunVector[sv_Z] = y;
		break;
	case (Yplus):
		sunVector[sv_X] = -x;
		sunVector[sv_Y] = z;
		sunVector[sv_Z] = y;
		break;
	case (Yminus):
		sunVector[sv_X] = x;
		sunVector[sv_Y] = -z;
		sunVector[sv_Z] = y;
		break;
	}
}

void SunSensor::GetSunVector(double* sunVector, const int points2avg)
{
```

```
double X, Y, Z, p, t, mag, XOffset, YOffset;
double Xavg = 0;
double Yavg = 0;
int i = 0;
int pointsAveraged = 0;

// x1, x2, y1, and y2 for all Sun Sensors
GetRaw(Xplus, rawXplus);
GetRaw(Xminus, rawXminus);
GetRaw(Yplus, rawYplus);
GetRaw(Yminus, rawYminus);
GetRaw(Zplus, rawZplus);

// active, XOffset, and YOffset
SelectSensor(rawXplus, rawXminus, rawYplus, rawYminus, rawZplus, active);
switch(active)
{
case (Xplus):
	XOffset = XplusXOffset;
	YOffset = XplusYOffset;
	break;
case (Xminus):
	XOffset = XminusXOffset;
	YOffset = XminusYOffset;
	break;
case (Yplus):
	XOffset = YplusXOffset;
	YOffset = YplusYOffset;
	break;
case (Yminus):
	XOffset = YminusXOffset;
	YOffset = YminusYOffset;
	break;
case (Zplus):
	XOffset = ZplusXOffset;
	YOffset = ZplusYOffset;
	break;
}
do
{
	// x1, x2, y1, and y2
	GetRaw(active, raw);

	// X&Y (also a Coldplay album)
	X = (L / 2) * ((raw[raw_x2] + raw[raw_y1]) - (raw[raw_x1] +
		raw[raw_y2])) / (raw[raw_x1] + raw[raw_x2] + raw[raw_y1] +
		raw[raw_y2]) - XOffset;
	Y = (L / 2) * ((raw[raw_x2] + raw[raw_y2]) - (raw[raw_x1] +
		raw[raw_y1])) / (raw[raw_x1] + raw[raw_x2] + raw[raw_y1] +
		raw[raw_y2]) - YOffset;

	// Debugging, optional
	printf("[SS%d] --> Point %d: X: %f, Y: %f\r\n", active, i + 1, X,
		Y);

	// Xavg and Yavg
	if (!(X != X || Y != Y))
	{
		Xavg += X;
		Yavg += Y;
		pointsAveraged++;
```

```
		}
	} while (++i < points2avg);

	Xavg /= pointsAveraged;
	Yavg /= pointsAveraged;

	// Debugging, optional
	printf("[SS%d] Xavg: %1.3f mm, Yavg: %1.3f mm\r\n", active, Xavg*1000,
		Yavg*1000);

	// Phi and Theta
	p = atan(sqrt(pow(Xavg,2) + pow(Yavg,2)) / d);
	if (Xavg < 0 && Yavg < 0)
	{
		t = atan(Yavg / Xavg) - pi;
	}
	else if (Xavg < 0)
	{
		t = atan(Yavg / Xavg) + pi;
	}
	else
	{
		t = atan(Yavg / Xavg);
	}

	// Debugging, optional
	printf("[SS%d] p: %1.3f(%3.1f), t: %1.3f(%3.1f)\r\n", active, p,
		p*180/pi, t, t*180/pi);

	// Sun Vector
	sunVector[sv_X] = sin(p)*cos(t);
	sunVector[sv_Y] = sin(p)*sin(t);
	sunVector[sv_Z] = cos(p);
	TransformVector(active, sunVector);

	// Debugging, optional
	printf("[SS] Sun Vector: [%1.3f, %1.3f, %1.3f]\r\n", sunVector[sv_X],
		sunVector[sv_Y], sunVector[sv_Z]);
}

void SunSensor::GetActive(int& active)
{
	active = SunSensor::active;
}

void SunSensor::ReadConfig(const int sensor, unsigned char& configHiRead,
	unsigned char& configLoRead, int& ack)
{
	// TODO (CDH): Implement methed to differentiate sensors

	char configBytes[2];
	ack = i2c.write(ADDR, &CONFIG_REG, 1);
	if (ack == 0)
	{
		i2c.read(ADDR, configBytes, 2);
		configHiRead = configBytes[0];
		configLoRead = configBytes[1];
	}
}
```

ACADEMIC VITA

Ravender Virk

College Address
825 S. Allen St. Apt #5
State College, PA 16801

ravender@psu.edu

Permanent Address
2195 Porter Way
Lancaster, PA 17601-5948

EDUCATION

Bachelor of Science in Electrical Engineering

The Pennsylvania State University, Schreyer Honors College
Expected Graduation: December 2012 (3.5 years)

WORK EXPERIENCE

Student Space Programs Laboratory (2009 – Present): University Park, PA
(Specialized in the design, analysis, and testing of high-altitude and space systems)

- Developed an array of sun sensors to determine satellite's orientation in space relative to the sun from concept to prototype to flight-ready sensor
- Circuit design and production
  - Participated as a team to develop and layout the satellite's motherboard
  - Utilized SharePoint and Subversion (SVN) to share documents and designs amongst the design team and facilitate communication
  - Engineered printed circuit boards with Altium Designer
  - Soldered SMD components accurately and frequently without error
- Circuit testing and debugging
  - Debugged errors in operation using continuity checks, custom breakout boards, controlled inputs, multimeters, and oscilloscopes
  - White-wired PCBs to fix these errors by studying designers' schematics

TECHNICAL SKILLS

- MATLAB
- Design software
  - Altium Designer, National Instruments Multisim and Ultiboard, Autodesk AutoCAD and Inventor, Microsoft Visual Studio, and Visio
- Advanced programming knowledge
  - C, C++, Assembly, C#, ASPX, Silverlight, WP7, SQL, XML (DTD, XSLT), HTML/CSS, Java SE/ME, Linux command shell, and many others

LEADERSHIP ROLES

- Two-year Treasurer and current Corporate Relations Chair of the Penn State IEEE
  - Collaborated as an officer team to bring the club from almost non-existence to 400+ members, a $22,000 balance, and several corporate and academic partnerships in just two years
  - Hosted a series of weekly professional, educational, and social activities
  - Initiated outreach programs to foster interest in engineering in children
- Two-year Treasurer of the Penn State Thespians
  - Reversed nearly a decade of annual net loss into a stable financial model: from –$6000 a year to +$500 a year for a $35,000 balance
  - Single-handedly managed an annual exchange of $60,000
- Two-year President of the Penn State Performing Magicians