

The University of Akron

IdeaExchange@UAkron

---

Williams Honors College, Honors Research  
Projects

The Dr. Gary B. and Pamela S. Williams Honors  
College

---

Spring 2021

## Hard Hat Ambient Liability Observer (HALO)

Hunter Hykes

*The University of Akron*

Nathan Kish

*The University of Akron*

Brian Thomson

*The University of Akron*

Follow this and additional works at: [https://ideaexchange.uakron.edu/honors\\_research\\_projects](https://ideaexchange.uakron.edu/honors_research_projects)



Part of the [Electrical and Electronics Commons](#), [Power and Energy Commons](#), [Signal Processing Commons](#), [Systems and Communications Commons](#), and the [VLSI and Circuits, Embedded and Hardware Systems Commons](#)

Please take a moment to share how this work helps you [through this survey](#). Your feedback will be important as we plan further development of our repository.

---

### Recommended Citation

Hykes, Hunter; Kish, Nathan; and Thomson, Brian, "Hard Hat Ambient Liability Observer (HALO)" (2021). *Williams Honors College, Honors Research Projects*. 1329.

[https://ideaexchange.uakron.edu/honors\\_research\\_projects/1329](https://ideaexchange.uakron.edu/honors_research_projects/1329)

This Dissertation/Thesis is brought to you for free and open access by The Dr. Gary B. and Pamela S. Williams Honors College at IdeaExchange@UAkron, the institutional repository of The University of Akron in Akron, Ohio, USA. It has been accepted for inclusion in Williams Honors College, Honors Research Projects by an authorized administrator of IdeaExchange@UAkron. For more information, please contact [mjon@uakron.edu](mailto:mjon@uakron.edu), [uapress@uakron.edu](mailto:uapress@uakron.edu).

# Hard Hat Ambient Liability Observer (H.A.L.O.)

## Senior Project Final Report

DT04

Hunter Hykes

Nathan Kish

Brian Thomson

Nicklaus Walsh

Dr. French

4/23/2021

Table of Contents	
<b>Section 0 Abstract</b>	5
<b>Section 1 Problem statement</b>	5
1.1 Needs	5
1.2 Objective	6
1.3 Background	6
1.4 Marketing Requirements	12
<b>Section 2 Engineering Analysis</b>	13
2.1 Circuit	13
2.2 Electronics	13
<b>Section 3 Engineering Requirements Specification</b>	18
<b>Section 4 Engineering Standards Specification</b>	20
4.1 Safety	20
4.2 Communication - I2C and SPI	20
4.3 Programming Languages	21
4.4 Connector Standards	21
<b>Section 5 Accepted Technical Design</b>	21
5.1 Hardware Design	21
5.2 Software Design	41
5.3 Final Design Images	57
5.4 Testing and Demonstrations	59
<b>Section 6 Mechanical Sketch of System</b>	67
<b>Section 7 Team Information</b>	69
<b>Section 8 Parts List</b>	70
<b>Section 9 Project Schedule</b>	71
<b>Section 10 Conclusions and Recommendations</b>	82
<b>Section 11 References</b>	84
<b>Section 12 Appendices and Data Sheets</b>	86
Appendix A: System Code	86
Appendix B: Datasheets	132
Appendix C: Parts Order Forms	147

List of Figures

<b>Figure 1: Power System Circuit Schematic</b> .....	13
<b>Figure 2: Block Diagram Level 0</b> .....	22
<b>Figure 3: Block Diagram Level 1: Time-of-Flight Subsystem</b> .....	23
<b>Figure 4: Block Diagram Level 1: Accelerometer Subsystem</b> .....	25
<b>Figure 5: Block Diagram Level 1: Processor Subsystem</b> .....	26
<b>Figure 6: Block Diagram Level 1: LED Subsystem</b> .....	27
<b>Figure 7: Block Diagram Level 1: SD Card Subsystem</b> .....	28
<b>Figure 8: Block Diagram Level 1: Power Subsystem</b> .....	29
<b>Figure 9: Block Diagram Level 2: Overall System</b> .....	31
<b>Figure 10: LTspice Schematic of Full Power Circuit</b> .....	32
<b>Figure 11: LTspice Schematic of Charging Circuit</b> .....	33
<b>Figure 12: LTspice Schematic of Buck-Boost Converter</b> .....	33
<b>Figure 13: LTspice Waveform of Power Circuit</b> .....	34
<b>Figure 14: Eagle Accelerometer Schematic</b> .....	35
<b>Figure 15: Eagle SD Card Schematic</b> .....	36
<b>Figure 16: Eagle RGB LED Schematic</b> .....	37
<b>Figure 17: Eagle Processor Schematic</b> .....	38
<b>Figure 18: Eagle Charging Circuit Schematic</b> .....	39
<b>Figure 19: Eagle Converter Circuit Schematic</b> .....	40
<b>Figure 20: Eagle Time-of-Flight Schematic</b> .....	41
<b>Figure 21: Micro-SD Card Write Functionality</b> .....	43
<b>Figure 22: HALO Main Board Schematic-Processor and SD</b> .....	44
<b>Figure 23: HALO Main Board Schematic-Peripherals</b> .....	45
<b>Figure 24: HALO Time-of-Flight Board Schematic</b> .....	46
<b>Figure 25: HALO RGB LED Board Schematic</b> .....	46
<b>Figure 26: Accelerometer Configuration</b> .....	47
<b>Figure 27: Accelerometer Data Acquisition</b> .....	48
<b>Figure 28: Time-of-Flight Functions</b> .....	49
<b>Figure 29: Time-of-Flight Configuration</b> .....	49
<b>Figure 30: Time-of-Flight Data Acquisition</b> .....	50

<b>Figure 31: Time-of-Flight Data Acquisition</b> .....	51
<b>Figure 32: getNearestObstacleIndex()</b> .....	52
<b>Figure 33: showDistanceRGB()</b> .....	53
<b>Figure 34: Software Design Flowchart</b> .....	54
<b>Figure 35: SD Card State Diagram</b> .....	56
<b>Figure 36: LED State Diagram</b> .....	56
<b>Figure 37: Frontal View of Housing Unit, ToF Sensors and LEDs</b> .....	57
<b>Figure 38: Detached Housing Unit</b> .....	57
<b>Figure 39: Main Board Front</b> .....	58
<b>Figure 40: Main Board Back</b> .....	58
<b>Figure 41: Time-of-Flight and LED Boards</b> .....	58
<b>Figure 42: LED and ToF Right Side Green</b> .....	59
<b>Figure 43: LED and ToF Right Side Red</b> .....	60
<b>Figure 44: LED and ToF Left Side Green</b> .....	60
<b>Figure 45: LED and ToF Left Side Red</b> .....	61
<b>Figure 46: LED and ToF Front Red</b> .....	61
<b>Figure 47: LED and ToF Front Blue</b> .....	61
<b>Figure 48: LED and ToF Max Distance Test</b> .....	62
<b>Figure 49: LED and ToF Max Distance 49 inches (~1245mm)</b> .....	62
<b>Figure 50: Initial Voltage Before Discharge</b> .....	63
<b>Figure 51: Final Voltage After 8 Hours</b> .....	64
<b>Figure 52: Mechanical Sketch of System (Overall)</b> .....	67
<b>Figure 53: Mechanical Sketch of System (Rear View)</b> .....	68
<b>Figure 54: Mechanical Sketch of System (Front View)</b> .....	69

List of Tables

<b>Table 1: Engineering Requirements</b> .....	18
<b>Table 2: Safety Standards</b> .....	20
<b>Table 3: Communication Protocols</b> .....	20
<b>Table 4: Connector Standards</b> .....	21
<b>Table 5: Functional Requirements</b> .....	22

<b>Table 6: Processor Functionality</b> .....	24
<b>Table 7: Accelerometer Functionality</b> .....	25
<b>Table 8: Time-of-Flight Functionality</b> .....	27
<b>Table 9: LED Functionality</b> .....	28
<b>Table 10: SD Card Functionality</b> .....	29
<b>Table 11: Power System Functionality</b> .....	30
<b>Table 12: Overall System Functionality</b> .....	31
<b>Table 13: SPI Functionality</b> .....	55
<b>Table 14: I2C Functionality</b> .....	55
<b>Table 15: Discharge Testing Results</b> .....	64
<b>Table 16: Recharge Testing Results</b> .....	65
<b>Table 17: Main Board Bill of Materials</b> .....	70

## **0.) Abstract.**

Capturing workplace incident information is a growing area of concern for most companies. To assist with this, the design team proposed the H.A.L.O. This design uses time-of-flight sensors connected to LEDs to create a proximity-based hazard warning system. It also records incident data using an accelerometer and micro-SD card. This helps workers avoid some of the most common workplace injuries, slips, trips, and falls and accidental collisions.

Students have created a design with engineering, and marketing requirements that accomplish this task. The proposed design allows for this monitoring and mitigation systems to be attached to hard hats. Team members developed software and hardware subsystems to fit on any hardhat without hindering worker safety.

The completed design uses the systems listed above register hazardous objects within 1.5m and color shifts depending on distance. Within the 150-degree FOV, any objects approaching the device are registered. In case of a possible concussive event, collision data writes to a SD card for use during an incident investigation. After a semester of development and integration, the H.A.L.O. system met the engineering requirements to assist with preventing workplace injury in a cost-effective manner. -NK

## **1.) Problem Statement.**

### **1.1 Need.**

Incident information in the workplace is difficult to capture due to the unpredictable nature of accidents. TapRoot, which is a method for identifying and correcting the root causes of an accident, and investigation analysis can only eliminate causal factors when understood and documented. Presently, reports come from firsthand accounts without numerical backing. With slips trips and falls, being 26.6% of workplace injuries, and another 23.3% being accidental

collision with objects in 2018, prevention and data collection are the best methods to create a safe workspace.

-NK

### **1.2 Objective.**

A device that can assist with incident prevention and document occurrences provides a safer working environment for everyone. The team's objective is to create a device attachable to a standard hardhat to provide enhanced sensory feedback, log sensor data related to the working environment, and record incident data. This provides a means to both prevent and more accurately document incidents.

-HH, NW

### **1.3 Background.**

The basic theory behind the concept of a smart, hard hat attachment is to increase the amount of information gleaned from an incident or accident in the workplace, specifically those in construction. Worker safety is an essential aspect of any industry, especially in dangerous fields such as construction and manufacturing. In most manufacturing facilities and construction sites, reporting an incident or an accident is rudimentary nonexistent at worst. Most reporting systems rely on the memory of the worker or a witness to describe the incident, leading to an often unreliable and unhelpful report. These reports will often not include crucial information such as the conditions when the accident took place, the exact location at which the accident occurred, or whether workers followed OSHA regulations. Without this crucial data, these accidents are likely to happen again, putting more workers in harm's way unnecessarily.

-BT



The team plans to resolve this issue using data collection and real-time monitoring of the user's surroundings using various sensors. Time-of-flight sensors will provide the individual wearing the helmet with a visual indication of hazards at the head level within some pre-specified range to help prevent accidental collisions to the head. A data logging system will use a micro SD card via an SPI interface to record relevant data, with assistance by a real-time clock module (Veeramani Kandasamy) to provide relevant timestamps when the incident occurred. These readings could then locate and address the source of these abnormalities, making it much easier to correct any infraction or spill.

In the unfortunate scenario in which the wearer suffers injury, specifically a collision to the head because of an unseen hazard or from a fall, an accelerometer measures the severity of the impact. This will provide relevant information regarding what medical treatment the victim should undergo, specifically focusing on the possibility of a concussion. One study conducted on concussions using helmeted devices in various sports shows that multiple accelerometers would provide more accurate data (O'Connor). Because of the shape of the team's initially proposed design, more accelerometers are available.

-HH

Time-of-flight sensors work "by illuminating the scene with a modulated light source and observing the reflected light. The phase shift between the illumination and the reflection is measured and translated to distance" (Li). This means that these sensors can sense the distance of the nearest object in their line of sight. By positioning three sensors on a helmet, it will be possible to detect any hazards that may come close to the individual's head and provide visual sign via a set of RGB LEDs. These LEDs are a commonplace component in modern electronics

and can produce a variety of different colors from a 'single' RGB LED, which is three LEDs (one red, one green, and one blue) within a close distance to each other on one component. It will use this feature to show how close a hazard is to the wearer's head by changing color as obstacles get closer.

-HH

Accelerometers are sensors that “measure the physical acceleration experienced by an object because of inertial forces or mechanical excitation” (Accelerometer Theory & Design). In aerospace applications, these sensors with gyroscopes for navigational and flight control purposes are widespread. In short, accelerometers act as a damped object of mass on a spring, therefore, when there is acceleration mass displaces, and it measures the displacement. Adding one to the helmet allows investigators to view the acceleration data of workers wearing the helmet after an incident. This will allow for a better determination of medical treatment.

-NW

Ambient Light Sensors (ALS) monitors and measures the level of ambient lighting surrounding the sensor. These devices primarily detect the amount of visible light surrounding the sensor. However, infrared and ultraviolet light can skew the readings of the sensor, as it does not limit the sensor to the bandwidth of the human eye. To remedy this, this sensor can also filter out infrared and ultraviolet light as to give a more accurate reading (*Ambient Light Sensor (ALS) Applications in Portable Electronics*). This sensor allows a safety inspector to see the lighting conditions at the scene of an incident or accident. This will help safety inspectors to find the root cause of an accident and will help them prevent a similar incident or accident from occurring

again. After considerations of cost, complexity, and the lack of necessity, the team removed ALS sensors from the device design.

-BT

The goals of this safety device are to record all the pertinent data relating to an incident or accident without impeding the worker, and to be an affordable alternative to other solutions currently on the market. This device design fits around the top of any hardhat with a few built in clips that fit onto the rim of the hardhat. This device will be low-profile and lightweight, as to perform all its tasks without impeding the worker. Along with that, this device will contain a battery cell that will last an entire workday on a single charge, removing the need for the worker to worry about battery levels during the workday. This device is rather simple as far as components go, as it only uses low-cost components such as a battery, a microprocessor, an accelerometer, a micro SD card slot, a micro USB port, a plastic band, a light sensor, and a few RGB LEDs, so the cost per unit will be low (less than \$80). This device will also allow supervisors to check for adherence to OSHA guidelines. Failing to comply with OSHA guidelines is a driving factor in many accidents. According to the *Commonly used Statistics* page on the Occupational Safety and Hazard Administration's webpage, the top causes of worksite fatalities are falls and being struck by an object, both of which are preventable, following proper safety standards. This device uses a combination of monitoring and recording to ensure workers create more safe working environments.

-BT

There are a few variations of “smart hard hats” in existence, many of which focus on monitoring the wearer of the device with features such as heart-rate sensors, GPS, Wi-Fi reporting of data, and map navigation of a site. Of these existing designs, many are permanent fixtures on the hardhat themselves, limiting the number of devices used to a specific set of individuals. Although the features of these designs are convenient, the modules are quite costly for the information they provide. Something else to consider is that GPS and Wi-Fi features have limited usefulness to specific types of worksites. Power plants and similar facilities are typically multiple story indoor facilities constructed out of metal—yielding any GPS features useless. Wireless communications at such sites are typically restricted or prohibited for security reasons. For these reasons, the team elected to log data via a micro-SD card to avoid these unnecessary and power-hungry modules.

-HH

Issues with limitations for the existing designs available arise since there are no similar designs. Other designs to the hard hat and non-removable. This strays from the team’s design and is costly to implement. Once a hard hat is no longer usable, workers must replace the complete system. (Temperature Measurement Inside Protective Headgear). One device that seems like our design, being an in-hat temperature sensor to prevent heat-related illnesses for workers in the field, has the limitations of needing to be inside the hat. Other designs like that of the in-helmet temperature sensor require the devices fixed to the helmet making repair or even powering the system difficult (Temperature Measurement Inside Protective Headgear). The team’s design however is an attachment to the outside of the helmet and can detach from the

helmet. This design being easily detachable from the helmet makes the repair of the system and the recharging for batteries of the device a smoother, easier task to perform.

-NW

The present designs found and listed in the patent references; current systems focus more on passive data tracking rather than incident prevention. The passive data tracking that most current designs are tracking is location, whether the user is wearing the helmet, and the activity of the worker based on their movement. This is good information, but costly, and overbearing for every worker on a site. Rather than a focus on the activity of the worker, this design is based around the safety of said worker. Also, other designs focus on the use of augmented reality assistance with communications capabilities (*Hard Hat with Additional Technical Features*). The design concept instead focuses on recording or eliminating the possibility of accidents occurring by using time-of-flight sensors, light sensors, and accelerometers. Present designs integrate the system onto the helmet for each device. As mentioned previously, a fundamental difference is that that design being created is removable. Discarding compromised helmets is possible with this design, making it better than current models.

-NK

The recording of data is the greatest design similarity to modern iterations of smart helmets. As mentioned, current systems also use removable SD cards that record the information desired. Many existing designs use real-time readings for location and other sensory details centered on worker output. Although both designs share the similarity of utilizing this technology, the designed system has focus on incident mitigation. The real time signals being

used in this design are time-of-flight readings that connect to LEDs. Alongside the time-of-flight sensors, accelerometer data assists with safety precaution and reporting.

There is a lot of research around smart hard hats. The usage of several real time sensors is already in existence, which is like the concept being designed. However, the systems already in use and patented center on the addition of augmented reality and monitoring of worker effectiveness. Some other systems are focusing on the usage of closed-circuit camera monitoring and communication systems/ subsystems for real-time interactions and direction of workers (*Smart Helmet*). These are relevant as they use similar technology to monitor these data points, but the overall system has different goals. The addition of these systems and subsystems also increases complexity and affordability of the incident monitoring system being proposed. In preliminary research, there were no designs that focused on incident prevention and risk management in a cost-effective manner like the design being proposed.

-NK

#### **1.4 Marketing Requirements.**

- 1.) The device assists with prevention of collision with objects in the working environment.
- 2.) The system will record impacts above a determined threshold for incident monitoring.
- 3.) The System is portable and rechargeable.
- 4.) The system will have a workday battery life of 8 hours.
- 5.) The system will detect objects in the user's field of vision using sensors.
- 6.) Does not change a standard hard hat's integrity.

- NK

## 2.) Engineering Analysis.

### 2.1 Circuit.

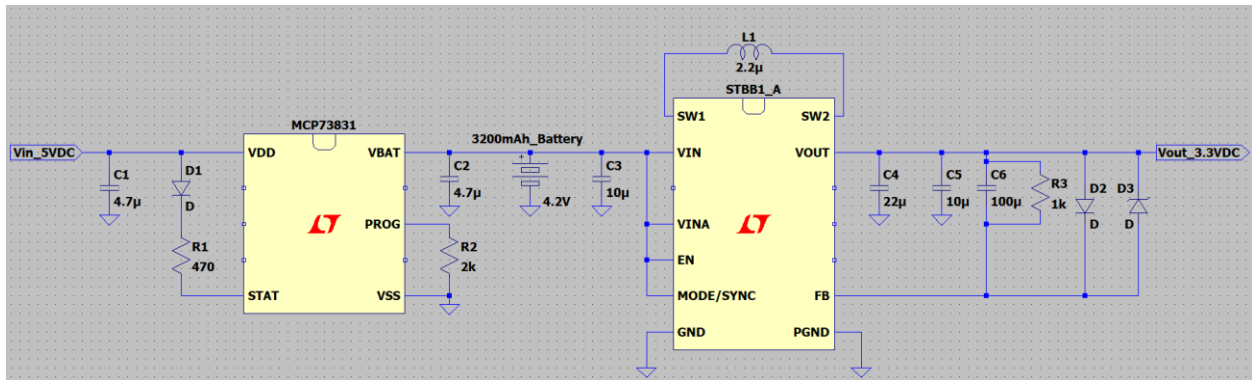


Figure 1.) Power System Circuit Schematic.

## 2.2 Electronics.

### Time-of-Flight Sensors:

For monitoring, there were two possibilities for distance sensors. IR triangulation was the first method researched for potential sensors. IR triangulation has benefits such as being compact, lightweight, and capable of detecting objects in real-time with a wide range of view. However, despite their ability to detect objects in real time, and a wide field of view on most models, they do not capture objects moving at a high speed. Their refresh rate is lower than other possibilities, making them less accurate at shorter distances. Also, part of the issue is the inability to use them in unison with other sensors. They provide less reliable measurements when used in conjunction. The intention of this design is not to detect high-speed objects moving towards the sensors in this aspect. The design requirements focus on detecting overhead objects in a predetermined FOV around the design, assisting with hazard detection.

The second sensors examined for this project were IR time-of-flight sensors. They use IR LEDs like the previously explored option. However, these use the sensors with a higher

transmission and reception time, a better distance detection limit, and better refresh rate. Some models explored were capable of refresh rates around 160 frames per second. A refresh rate that high is perfect for real time situations and with a fixed FOV, time-of-flight sensors were by far the best avenue for detections based on design requirements. -NK

A time-of-flight sensor uses infrared light to determine depth information. By emitting a pulse of infrared light and observing the reflected light pulse, the sensor can make a conclusion about the distance to the object light reflected from. The speed of light being approximately 300,000,000 m/s allows for readings taken in very little time. For reference, detecting an object approximately 100m away from the sensor would take 660ns. -HH

### **Accelerometer:**

The team will use an accelerometer to measure incident data. This is because of the component's ability to track tilt and angular changes, force and acceleration over time, and collisions. It does this via a combination of a spring, damper, and a mass. A displacement sensor is in the system to measure the mass movement relative to its attachment point. This tracks the inertial forces on the mass. It compensates these forces with the springs and damper. The tension from the combination of these two attachments keeps the mass in stasis to record.

With this system in place, when a collision occurs, a sensor on the accelerometer then calculates the displacement of the mass. This converts into measurements for acceleration ( $m/s^2$ ). With this data from the accelerometer, the second order response forms regarding displacement over time.

The system designed for incident recording uses an accelerometer that can track this response for plotting. As for how the data is captured by the accelerometer, capacitance is the



main method. In explored designs, the accelerometer has capacitor plates that attach to the mass. When the mass moves, and the capacitor plates, the capacitance does. Modern accelerometers have circuitry that takes this capacitance difference into account and presents an output voltage that represents the desired data.

-NK

### **Concussion Analysis:**

The most common form of head injury is a concussion. A concussion occurs when the head sustains a significant impact or blow that creates a chemical imbalance or physical damage within the brain. According to research by Kevin Guskiewicz, director of the Carolina's Sports Medicine Research Laboratory at UNC, concussions can result from impacts of 60g of force or more. As concussions are the most common form of workplace head injuries, it is important that workers are adequately protected from falling objects and falls. In many industries, the solution for this is to wear a hard hat. They design hard hats to mitigate the impact of falling objects or contact with the ground during a fall. The Occupational Safety and Health Administration (OSHA) requires employers with ten or more employees to keep a record of serious work-related injuries, concussions included. As stated above, the current methods in many workplaces around the country are rudimentary and non-existent at worst. -BT

### **LEDs Component Analysis:**

The need for a lighting-based warning system on the helmet led to the choice to use LEDs. A simple setup of three LEDs, that are receiving signals from the time-of-flight sensor and processor, is sufficient. The team decided this using two methods of reasoning, operation hours, and power efficiency. Despite more upfront cost for the bulbs, on average LEDs have an average rated life of roughly 50,000 hours which can be up to 50 times longer than incandescent

bulbs. LEDs also have a longer lifespan than CFLs and Halogens. The second reasoning behind picking LEDs is because of their efficiency with power consumption compared to other bulbs. On average, based on information from the department of energy, to produce a similar luminosity to other designs, LEDs consume 75% less power. This decision weighed heavily on the choice of component because of the powering requirements of the system.

The main reason that LEDs are so efficient with energy is because of their composition. An LED emits light when it is forward biased, and it applies a voltage across the junction. In the proposed design, the 3.3V signal from the processor comes from the time-of-flight sensor. As LEDs are a PN junction, the light forms when energy releases when holes and electrons combine at the junction, allowing current to flow. This is a much more energy efficient process than using filaments or halogen bulbs that burn out after a thousand hours of operation. Another important aspect of the design of the diode is their ability to change color based on the input signal. An orange or yellow light emits at a wavelength of 590 to 610 nanometers when the LED receives a signal of 2 (V) at 20 (mA). A red light radiates from most LEDs at a frequency of 610 to 760 nanometers when applied with a 1.6 to 2 (V) signal at (20 mA). The efficiency and ability for the input signal to change color of the diode will assist in the goal of avoiding hazards in proximity of the time-of-flight sensors.

-NK

### **Power Analysis:**

The system should have a 3.7V DC lithium battery and hold a charge for a minimum 8-hour workday. The battery charges using a 5V DC Micro-USB charger. Based on spec sheets for potential processors, their consumption comes at an average of 150 to 200 (mA). The (ToF) time-of-flight sensors has an average consumption of 18 (mA) per sensor. Six ToF sensors

combine for the expected average consumption of 108 (mA). The consumption of accelerometer specs is roughly 140 ( $\mu$ A). For the LEDs, each individual LED consumes on average 45 (mA) when activated with either the red or green diodes. A max consumption of power for the LEDs would be 135 (mA) with all three LEDs on. However, all three LEDs will probably not be on, since the ToF sensors spread apart at equal angles and don't link to each light. At most, expected consumption for the LEDs is 90 (mA). Taking the analysis for power consumption of all the components of the device, a total mA requirement is just below 400 (mA).

With an expected operation time of 8 hours, the battery required is  $(8hrs) * (400mA) = 3200 (mAh)$ . Since the system is being converted from a 3.7V battery to a 3.3V system, the power consumption steps down to  $(3.3V/3.7V) * 3200mAh$ . This consumption is then 2854 (mAh). With the 168 (mAh) loss from the DC to DC converter over an eight-hour period and other minor losses in mind, a battery of 3200 (mAh) is necessary. The power losses of the charging circuit are irrelevant, as that circuit does not draw any power from the battery. In practicality, the consumption of these components is much lower, so these calculations are max estimates of the power consumption in the absolute worst-case scenario.

The power system charges with a 5VDC Micro-USB cable. This will allow for a simpler design for the charging circuit, as Micro-USB charging circuits are already on the market and are cheap and easy to use. These charging circuits are also simple to integrate into a larger circuit, so they will work perfectly for our intended application. This will also allow for easy charging. If the charging process is easy and self-explanatory for the user, they will be more likely to charge the device. If the charging process is cumbersome or complicated, workers will be less likely to charge the devices. The team selected a design to run for a minimum of 8 hours. This accounts for a normal workday time frame. Using a 5VDC Micro-USB cable will also allow for cheap and

easy installation for the employers setting up charging stations for these devices. Micro-USB cables are easy to install. Therefore, designing the charging circuit to use a 5VDC Micro-USB cable will benefit us as the designers, the workers, and the employers alike.

One concern with using a 3200mAh battery is its physical size. 3200mAh batteries are 3.19” by 2.48” by 0.18”, which could present some difficulties with physical design. The weight of a 3200mAh battery is 56 grams, which is insignificant as far as the weight of the device goes. If the team can get a more accurate power analysis or reduce the number of components in this device, the size of the battery would decrease. For the time-being, however, a 3200mAh battery is acceptable.

- BT / -NK

### 3.) Engineering Requirements Specification.

Below is a table showing the engineering requirements for the design. A justification why these requirements are needed for the design are also present. Finally, the associated marketing requirement is shown to depict the reasoning why the engineering requirement was chosen. Attached to the bottom of the engineering requirement are the marketing requirements for reference.

*Table 1.) Engineering Requirements Table.*

<b>Engineering Requirements:</b>	<b>Justification</b>	<b>Marketing Requirement:</b>
1.) Must be able to last an entire 8-hour workday without needing to charge.	Allows for the worker to use the helmet for an entire shift without having to worry about battery life.	4
2.) The battery charges via a standard charging cable within 8 hours.	This should be enough time to charge the system using a standard 500mA charge rate.	3

3.) Will write suspected impact/injury data to an on-board storage system for data retrieval.	Information regarding injury is easily accessible and can be assessed after an incident.	2
4.) Provides time-stamp data relating to impact/injury.	Allows for waveform generation and data pertaining to when an incident happened.	2
5.) Does not interfere with existing standard hardhat hardware or functionality.	Makes it so companies do not need to buy special helmets to use this product while allowing for simple installation.	6
6.) Communication busses remain idle until sensors detect relevant data, keeping communication lines available.	When relevant data is not being detected, the communication busses are available. This allows relevant data transmission in a timely manner.	1,2,5
7.) LEDs illuminate with respect to a detected hazard's position relative to the user.	The system provides an intuitive display to assist users in locating a hazard.	1,5
8.) LEDs change color (yellow to red) as hazards approach between 1.5m and the user.	This will allow the worker to assess the proximity of an existing hazard.	1,5
9.) 150-degree FOV deploys to detect hazards.	Provides field of coverage relevant to where the user travels.	5
10.) Impact data above a predetermined threshold will indicate an incident.	Data above a threshold of (45g) will show an incident is within a margin of error of force that suggests a potential concussion.	2
<p>1.) The device assists with prevention of collision with objects in the working environment.  2.) The system will record impacts above a determined threshold for incident monitoring.  3.) The System is portable and rechargeable.  4.) The system will have a workday battery life of 8 hours.  5.) The system will detect objects in the user's field of vision using sensors.  6.) Does not change a standard hard hat's integrity.</p>		

#### 4.) Engineering Standards Specification.

##### 4.1. Safety.

The table below depicts the safety standards relevant to this design. Both standards listed in this table are provided and created by OSHA. -BT

*Table 2.) Safety Standards.*

<b>Safety</b>	<b>Title</b>
OSHA Part Number: 1926	Safety and Health Regulations for Construction
Standard Number: 1926.100	Personal Protective and Life Saving Equipment

Since the project is dealing with electrical equipment, students must ensure that standard 1926.100 (b) (2) applies. The team must implement electrical insulation of the device around the hard hat to keep within guidelines of OSHA requirements. For other safety standard considerations, the safety standards of selected electrical components will also apply. -NK

##### 4.2. Communication- I2C and SPI.

The table below depicts the communications protocols relevant to this design. These protocols include I<sup>2</sup>C and SPI. -BT

*Table 3.) Communication Protocols.*

<b>Communication Protocol</b>	<b>Implementation</b>
I <sup>2</sup> C	- Time-of-Flight Sensors - Accelerometer - I2C Multiplexers
SPI	- Micro-SD Card

Because of the difference in needs for communication, there are two separate communication methods. For the Time-of-Flight sensors, since there are multiple of them, an

addressed based system is required. For SPI, the standard means of embedded projects to interface with an SD card is using this method. Following this standard makes communication with the Micro-SD much more reliable and simpler.

### **4.3. Programming Languages.**

This project uses embedded C because of its commonplace implementation among Microchip devices. This language also offers a simple interface for the communication protocols by which it will communicate to the sensors, I2C and SPI. Team members have experience in C, making this the most “universal” language for interpretation for the team. - HH

### **4.4. Connector Standards.**

The table below depicts the connector standards relevant to this design. These connectors include the SD card receiver, a micro-USB connector for the charging cable, and a ICSP connector for the PICkit programming interface. -BT

*Table 4.) Connector standards.*

<b>SD-Card</b>	-SD Card Receiver
<b>USB</b>	-Charging cables
<b>ICSP</b>	<b>-PICkit programing interface</b>

## **5.) Accepted Technical Design.**

### **5.1. Hardware Design.**

#### **Block Diagram Level 0:**

The figure below shows the level 0 block diagram. This is a generalized overview of the H.A.L.O design. The design features six time-of-flight sensors, an accelerometer, a micro-SD car, and three RGB LEDs. The inputs to the processor are the time-of-flight sensors for multi-

directional distance measurements, and an accelerometer for measuring force to the helmet. The outputs of the LEDs and SD card assist with hazard recognition or incident recording.

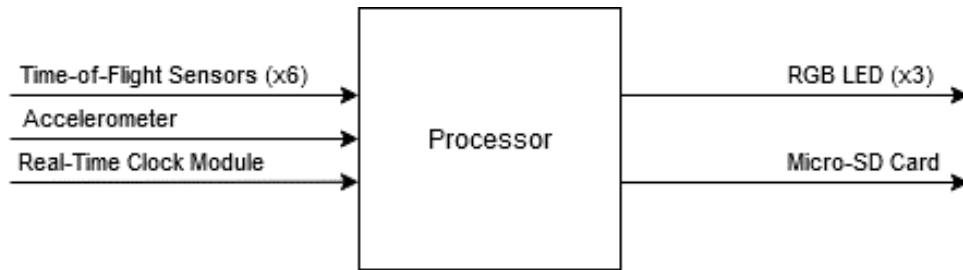


Figure 2.) Block Diagram Level 0.

- HH, NW

**Functional Requirement Table:**

The processor functional requirements table is shown below. Provided is a listing of inputs, outputs and a parts description. The inputs of the time-of-flight sensor, accelerometer, connect to the processor and go to the outputs of the LED and Micro SD card. The processor takes the data from the sensors and sends it to the proper system to assist with incident monitoring and mitigation. -NK

Table 5.) Functional Requirements Table.

Module	Processor
Designer	Hunter Hykes, Nathan Kish, Brian Thomson, Nicklaus Walsh
Inputs	<p><b>Time-of-Flight Sensors:</b> detect head-level hazards</p> <p><b>Accelerometer:</b> collect collision data if the wearer suffers from an impact to the head</p> <p><b>Real-Time Clock:</b> provide relevant timestamps for data logging</p>
Outputs	<p><b>RGB LED:</b> one light is present for each Time-of-Flight sensor and the color will show the proximity of the hazard</p> <p><b>Micro SD Card:</b> will log sensor data as for review.</p>



Description	Time-of-flight (TOF) sensors will operate when the light sensors determine the user is indoors and in a low-light area. Two RGB LEDs respond to the TOF sensor readings to show a hazard detected within some proximity to the wearer's head. LEDs from each side to help determine where the hazard is present. The LEDs will change color (likely yellow to red) as the hazard approaches. An accelerometer will be present on the device to record data if the wearer suffers an impact to the head. A volatile organic compound sensor will be present to take consistent readings as the wearer walks around the work environment, noting any abnormal conditions. A real-time clock assists a micro SD card to log all this data with timestamps.
-------------	---

-HH, NW, BT

**Block Diagram Level 1 | Processor Subsystem:**

A preliminary processor subsystem diagram is seen below. The power system's 3V3 inputs, the I2C and SPI communication lines are all visible. The interrupts signal to the processor that some event happened. This is done for both the accelerometer and time-of-flight sensor, allowing the processor to know when either a hazard is approaching, or an incident occurred. -NK

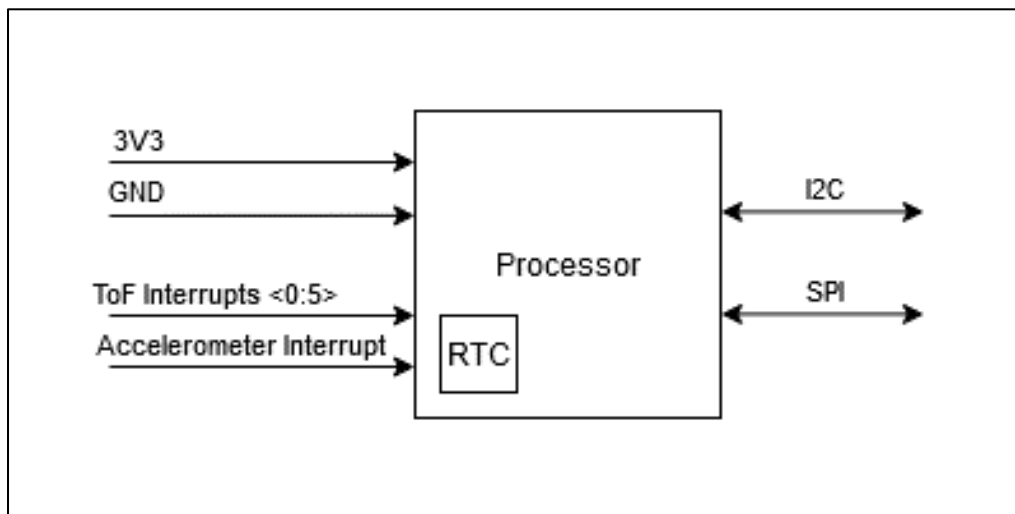


Figure 3.) Block Diagram Level 1: Time-of-Flight Subsystem.

The table below depicts the inputs, outputs, and functionality of the processor used in this design. The inputs for this processor are 3.3V power and a ground wire, and the outputs from this processor are communication lines that run to the other subsystems, and I/O interrupt pin, and the SD card communication lines. -BT

*Table 6.) Processor Functionality Table.*

Module	Subsystem Processor
Designer	Nicklaus Walsh
Inputs	3.3V input Ground wire
Outputs	Communication lines between each other subsystem I/O interrupt pin SD card communication lines
Functionality	The processor controls each subsystem and gathers information from each of the ToF sensors and the accelerometer. Based on the data relayed back to the processor, it will alert the user and record the data from the accelerometer to the SD card.

**Block Diagram Level 1 | Accelerometer Subsystem:**

A closer look at the connections between the accelerometer and the processor is below.

The outputs from the accelerometer are the serial data line, the serial clock line, and the interrupts. These allow the data from an incident where the accelerometer is used to be interpreted and used by the processor. -NK

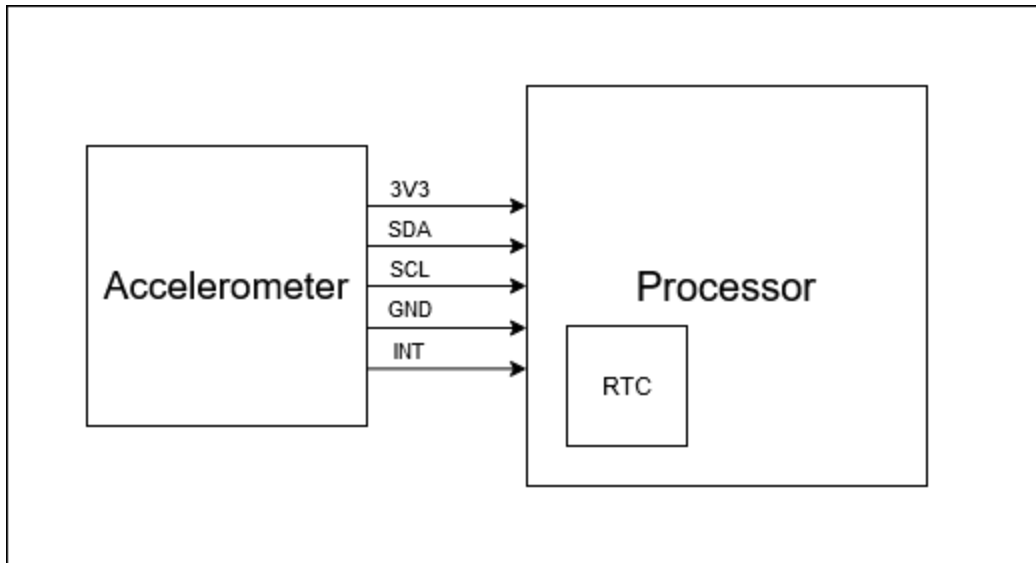


Figure 4.) Block Diagram Level 1: Accelerometer Subsystem.

The table below depicts the inputs, outputs, and functionality of the accelerometer used in this design. The inputs for this accelerometer are 3.3V power and a ground wire, and the outputs from this accelerometer are 3.3V power, SDA communications to the processor, SCL communications to the processor, and a ground wire. -BT

Table 7.) Accelerometer Functionality Table.

Module	Accelerometer
Designer	Brian Thomson
Inputs	3.3V power from the power system Ground Wire
Outputs	3.3V power SDA communications to the processor SCL communications to the processor Ground Wire
Functionality	To measure and quantify the acceleration of the device, and to communicate and relay that information to the main processor.

### Block Diagram Level 1 | Time-of-Flight Sensor Subsystem:

Below is a representation of the connections between the processor and the time-of-flight sensors. The same communication scheme used for the accelerometer is present here. Communication lines from the SCL and SDA allow the time-of-flight sensors to send information to the processor. This information is then sent to the LEDs, creating the proximity warning system for the H.A.L.O. design. -NK

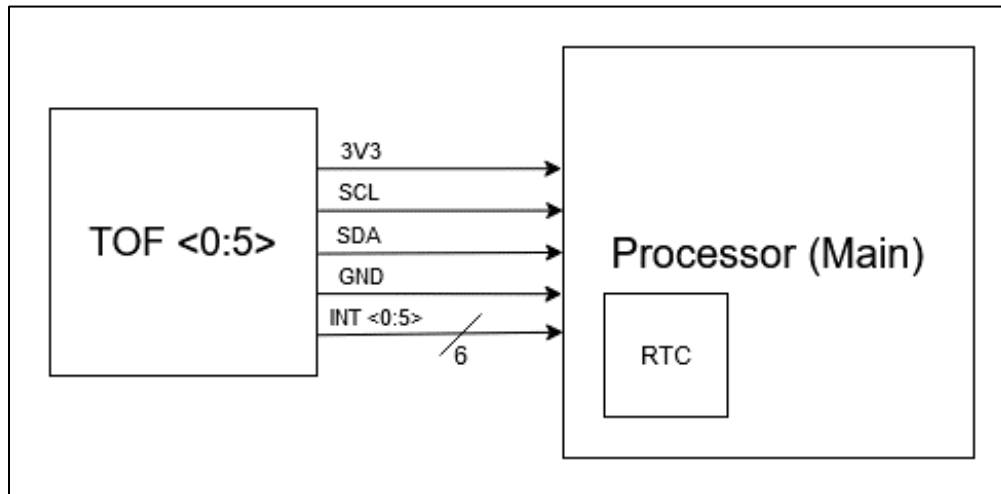


Figure 5.) Block Diagram Level 1: Processor Subsystem.

The table below depicts the inputs, outputs, and functionality of the time-of-flight sensor used in this design. The inputs for this time-of-flight sensor reflect light from a surface, 3.3V power, and a ground wire, and the outputs from this time-of-flight sensor are serial data lines to the processor. -BT

Table 8.) Time-of-Flight Functionality Table.

Module	Time-of-Flight Sensors
Designer	Nate Kish
Inputs	3.3V power Ground Wire Reflected Light from a Surface
Outputs	Serial data pertaining to the distance between the sensor and object in view.
Functionality	Acts as a measuring tool for distances from the sensors to potential hazards. Delivers data to the registers in the processors upon request.

**Block Diagram Level 1 | RGB LED Subsystem:**

The following block diagram is a representation of the LED subsystem. The H.A.L.O. uses three LEDs at equal spacing on the brim of the hard hat, all powered by the power subsystem at 3.3V. The IO pins allow for communication with the processor and time-of-flight sensors to vary the intensity of the light, acting as a warning device. -NK

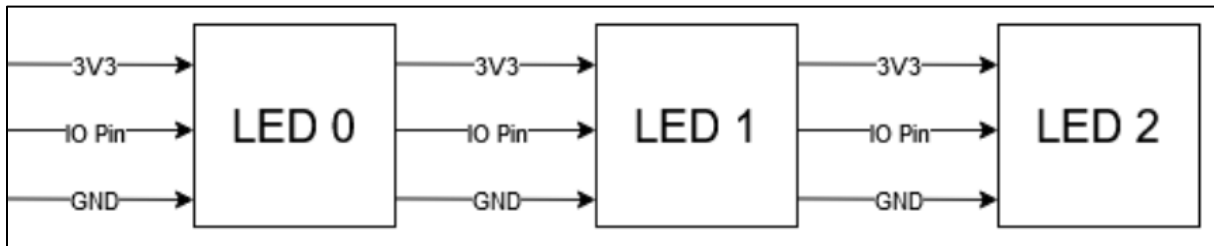


Figure 6.) Block Diagram Level 1: LED Subsystem.

The table below depicts the inputs, outputs, and functionality of the LEDs used in this design. The inputs for the LEDs are 3.3V power, an I/O pin, and a ground wire, and the outputs from the LEDs are red, orange, and yellow light. -BT

Table 9.) LED Functionality Table.

Module	LED
Designer	Nathan Kish
Inputs	3.3 V input IO Pin Ground cable
Outputs	Orange or yellow light emits at a wavelength of 590 to 610 nanometers when the LED receives a signal of 2 (V) at 20 (mA).  Red lights at a frequency of 610 to 760 nanometers
Functionality	The functionality of the LEDs is to act as a warning system for hazards in the immediate area around the user. Input distance data from the time-of-flight sensor goes to the LED and produces one of two colors depending on distance.

**Block Diagram Level 1 | Micro-SD Card Subsystem:**

The diagram below shows the interface of the processor and SD card. As seen in the figure, the SD card and processor communicate using MISO and MOSI communication scheme using SPI. The SD card connects to the processor using a serial clock line, following SPI standards to assist with data capturing. -NK

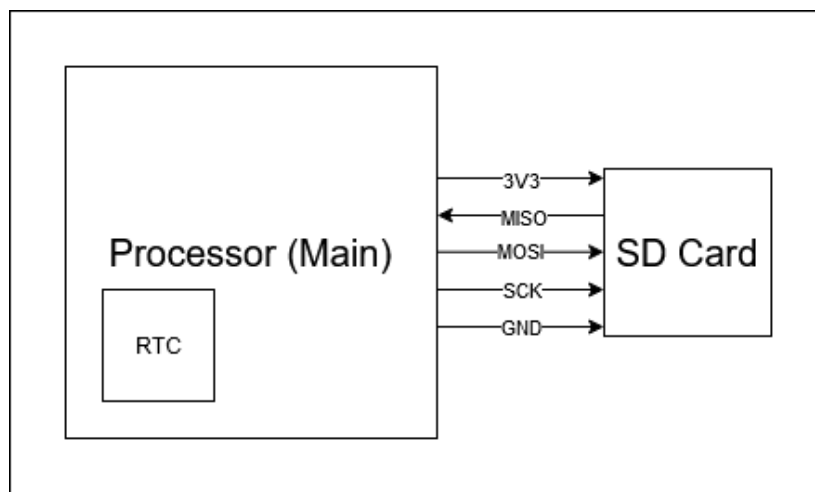


Figure 7.) Block Diagram Level 1: SD Card Subsystem.

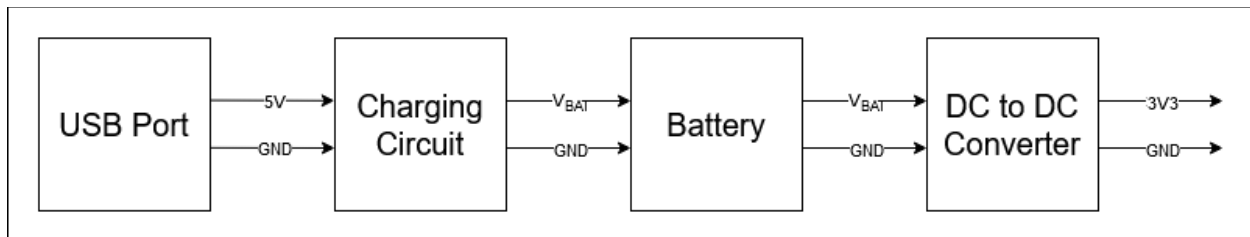
The table below depicts the inputs, outputs, and functionality of the SD card used in this design. The inputs for the SD card are 3.3V power, a MOSI communication line, and a ground wire, and the output from the SD card is a MISO communication line. -BT

*Table 10.) SD Card Functionality Table.*

Module	SD Card
Designer	Nicklaus Walsh
Inputs	3.3V, GND power input from battery slave select line from processor Master Out Slave In (MOSI) communication line
Outputs	Master In Slave Out (MISO) communication line
Functionality	Collect data from accelerometer if it is substantial contact and store on SD card in a CSV file for analysis. The CSV file generates a waveform to capture the severity of collisions should an incident occur.

**Block Diagram Level 1 | Power Subsystem:**

The figure below depicts a level 1 block diagram of the power circuit. Starting with a 5V micro-usb, the power circuit uses a charging circuit to step down the voltage to 4.2 and charge a battery of equal voltage. From said battery, voltage is then set to 3.3V using a DC to DC buck-boost converter. This subsystem powers the rest of the subsystems on the H.A.L.O. design. -NK



*Figure 8.) Block Diagram Level 1: Power Subsystem.*

The table below depicts the inputs, outputs, and functionality of the power system used in this design. The inputs for the power system are 5.0V power and a ground wire, and the outputs from the power system are 3.3V power and a ground wire. -BT

*Table 11.) Power System Functionality Table.*

Module	Power System
Designer	Brian Thomson
Inputs	5V DC input from Micro USB Charging Cable Ground Wire
Outputs	3.3V DC Power distributed throughout the system Ground Wire
Functionality	To store and distribute power of a uniform voltage throughout the circuit. Power comes into the charging circuit at 5V DC, then steps down to 4.2V to charge the battery. The battery itself has an average voltage of 3.7V. From the battery, power flows through a DC/DC converter to regulate the voltage to 3.3V. This system will also provide a universal ground for the entire circuit.

**Block Diagram Level 2:**

The following figure depicts an implementation of the whole H.A.L.O system. The level 2 diagram shows connections between the subsystems designed previously and the processor. It also depicts connection pins and lines in this diagram. Though a simplified version of the system using blocks, the flow of power, information, and data is now visible between all subsystems. -NK



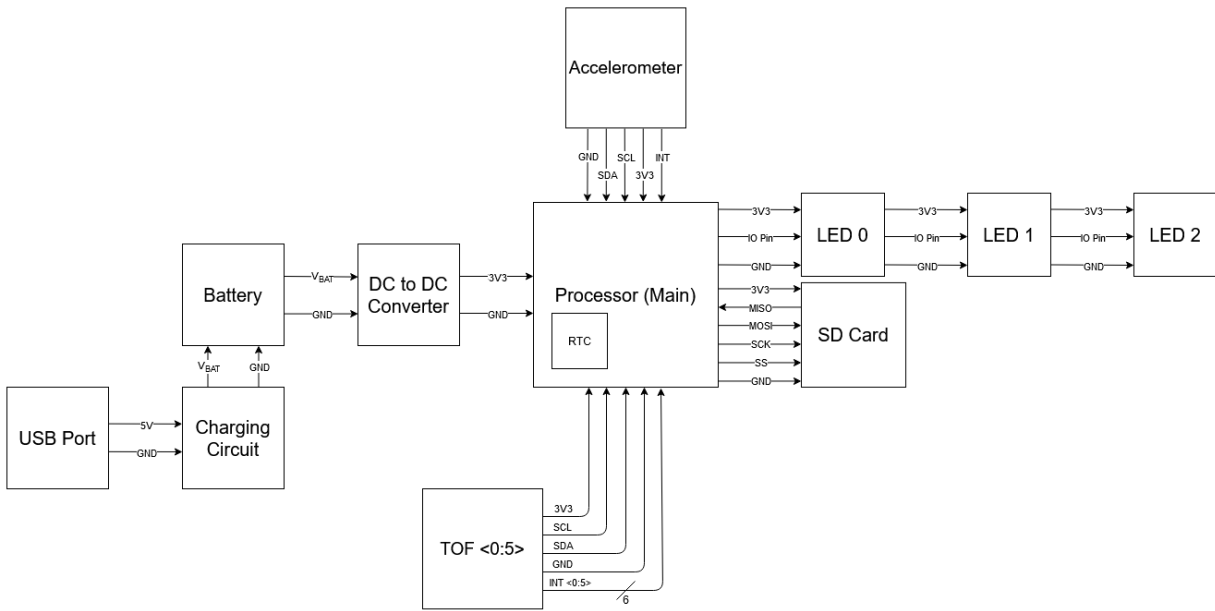


Figure 9.) Block Diagram Level 2: Overall System.

-HH

The table below describes the input, output, and functionality of the overall system.

Table 12.) Overall System Functionality Table.

Module	System Overview
Designer	Hunter Hykes
Inputs	5V DC input from Micro-USB Charging Cable Time-of-Flight Sensors (x6) Real-Time Clock Accelerometer
Outputs	RGB LEDs (x3) Micro-SD Card
Functionality	The system's battery charges via the Micro-USB cable's 5V and GND inputs. Time-of-Flight sensors will sense hazards within a predetermined spatial radius and send back detailed distance readings to the processor. The processor will then drive the respective RGB LEDs based on how close the hazard is.  The accelerometer will log data determined to be around the threshold for a concussion to the Micro-SD card. This assists with recording incident data.

## Hardware Design-Power System:

The power circuit is complete using the LTspice schematic shown in Figure 10. This circuit is three parts: the charging circuit, the battery, and the buck-boost converter. This circuit takes an input of 5V from a universal charging cable. This input goes into the charging circuit, which reduces the voltage from 5V down to 4.2V. This circuit also regulates the charging current to 500mA. The current then flows from the charging circuit into a 3200mAh lithium Ion battery. This battery stores a nominal voltage of 4.2V.

The charging circuit is in Figure 11. The charging circuit implemented in the physical circuit is a model TP4056 made by MCIgIcM. From there, power flows through the buck-boost converter. The input voltage from the battery will vary, depending on power stored within it. If the battery is fully charged, it will have a voltage of 4.2V. But as the battery drains, the voltage will drop, eventually dropping to around 0V. The average voltage of the battery while in use is 3.7V.

The buck-boost converter will yield a 3.3V output, regardless of the input voltage (so long as the battery voltage does not drop below 2V, as this is the minimum required voltage for the buck-boost converter to operate). The buck-boost converter is in Figure 12. The buck-boost converter in the physical circuit is a model STBB1-AXX made by STMicroelectronics. From there, a constant 3.3V output goes to the rest of the device. -BT

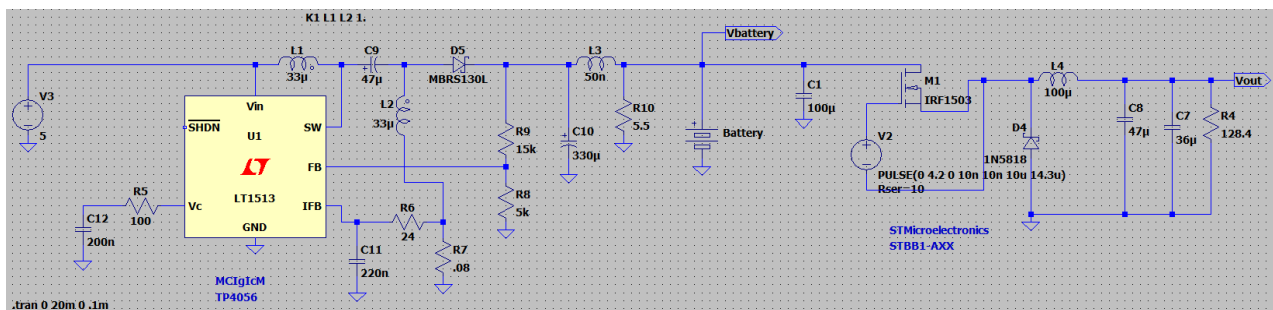


Figure 10.) LTspice Schematic of Full Power Circuit.

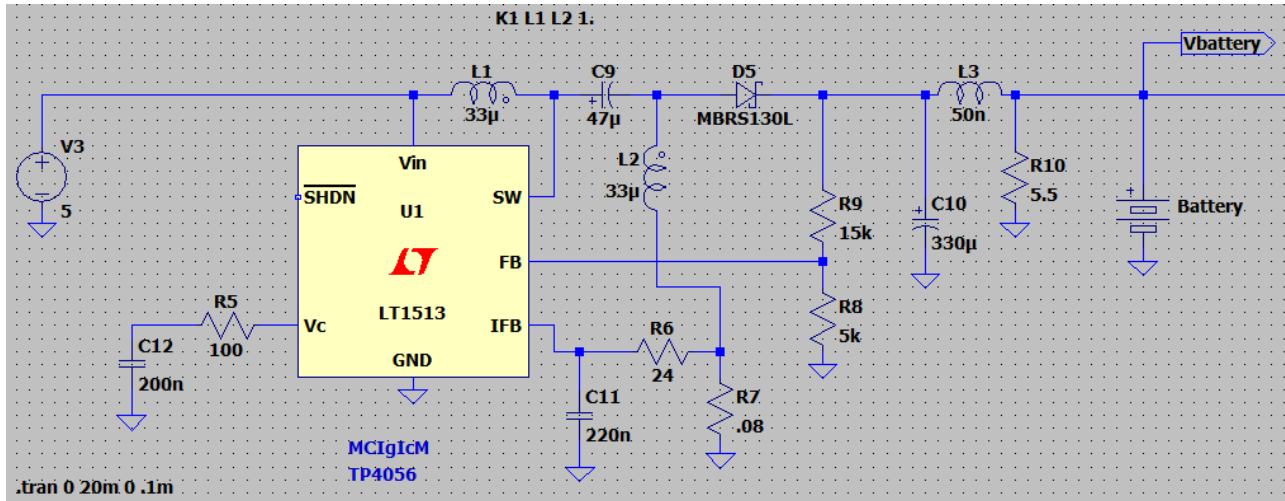


Figure 11.) LTspice Schematic of Charging Circuit.

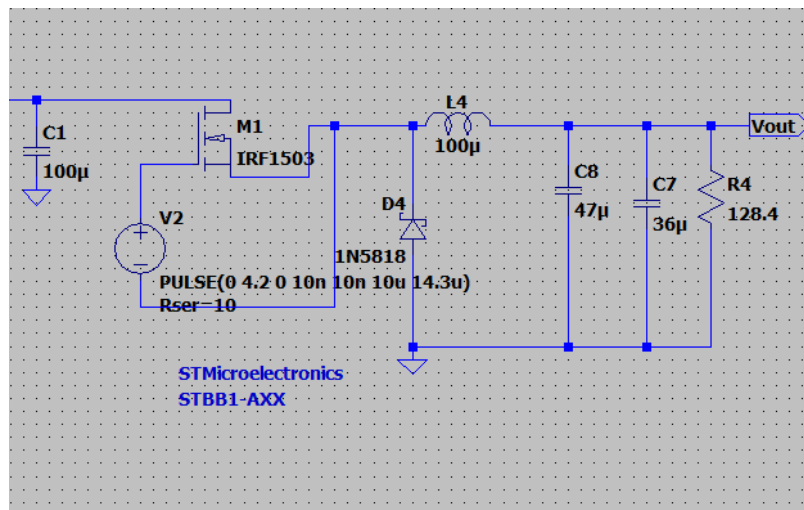


Figure 12.) LTspice Schematic of Buck-Boost Converter.

The waveform for this circuit is in Figure 13. In this waveform figure, the teal line is the input voltage, which is a constant 5V. The red line is the voltage of the battery, which is around 4.2V after a brief transient phase. For demonstrations, the battery gains a full charge nearly instantaneously. In reality, this process would have taken around six and a half hours, which would have taken days to simulate, considering it took nearly a minute to simulate 20ms. The

blue line is the current flowing into the battery, which is 500mA after a brief transient phase. The green line represents the output voltage, which is 3.3V, after the same transient phase. For proof of concept, any student educated in Spice schematic creation can follow these schematics to create a similar power circuit. -BT

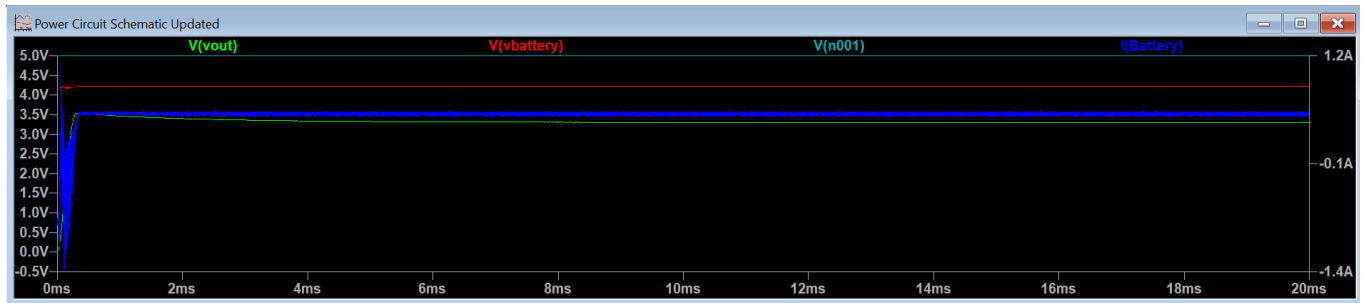


Figure 13.) LTspice Waveform of Power Circuit.

## Hardware Software Eagle Schematic:

### Accelerometer:

The accelerometer in Figure 14 is the H3LIS200DLTR. It can operate in either I<sup>2</sup>C or SPI but for this design it's using I<sup>2</sup>C. The I<sup>2</sup>C is used to write data into registers in which the content also writes back. There are two signals associated with this I<sup>2</sup>C set up, the serial clock line SCL2 and the serial data line SDA2. Both lines connect to the processor to transmit and receive data using pull-up resistors connecting the 3V3 VDD of the processor. The accelerometer acts as the slave device following the initial instruction of the processor which starts transfer, generates clock signals, and then terminates the transfer as needed. It does this through the SDA2 and SCL2 lines, as mentioned. Like the Time-of-Flight sensors, the accelerometer complies with a 400kHz fast mode and a normal mode of operation. -NK

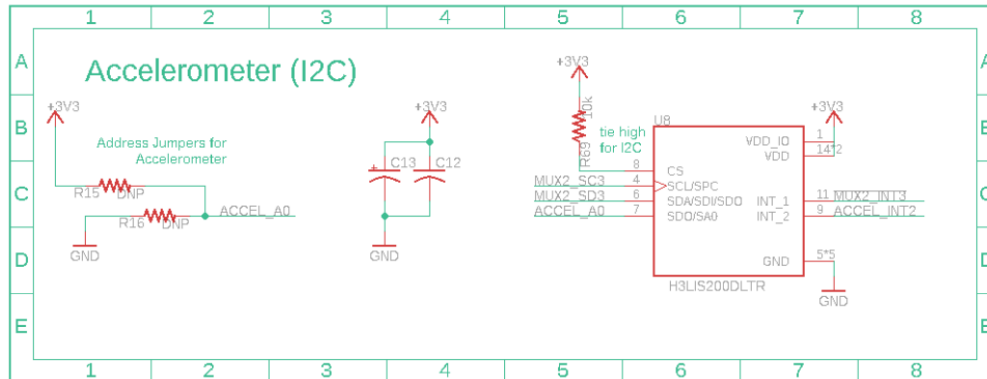


Figure 14.) Eagle Accelerometer Schematic.

### SD Card:

The SD card interface has been connected according to standard SD Card to SPI pin mappings. As is shown in Figure 15, there are four pins utilized by the SPI interface: Slave Select (SS), Serial Clock (SCK), Master Out Slave In (MOSI), and Master In Slave Out (MISO). These connections are made between the processor's SPI1 interface and a level shifter. This level shifter ensures connections going to the SD card do not exceed 3.3V. As the level shifter is shown now, VCCB is the same as VCCA; however, if the final design uses a VDD greater than 3.3V, then VCCB will be connected to this VDD and VCCA, the output voltage of data lines going into the SD card, will remain at 3.3V.

-HH

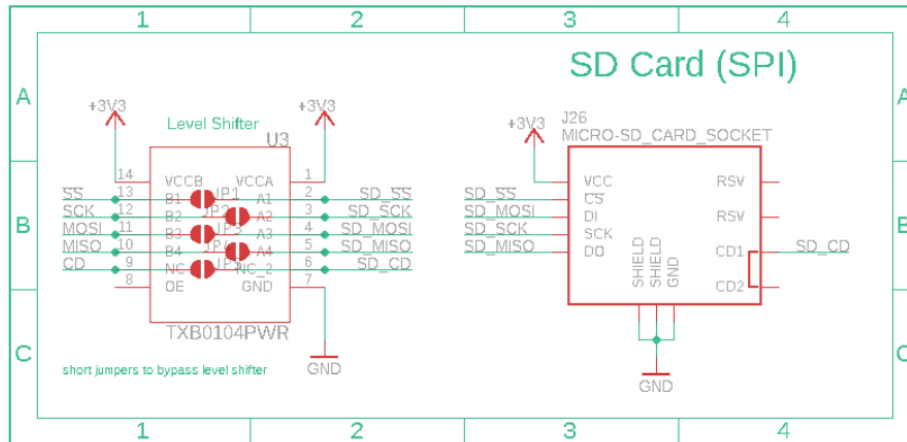
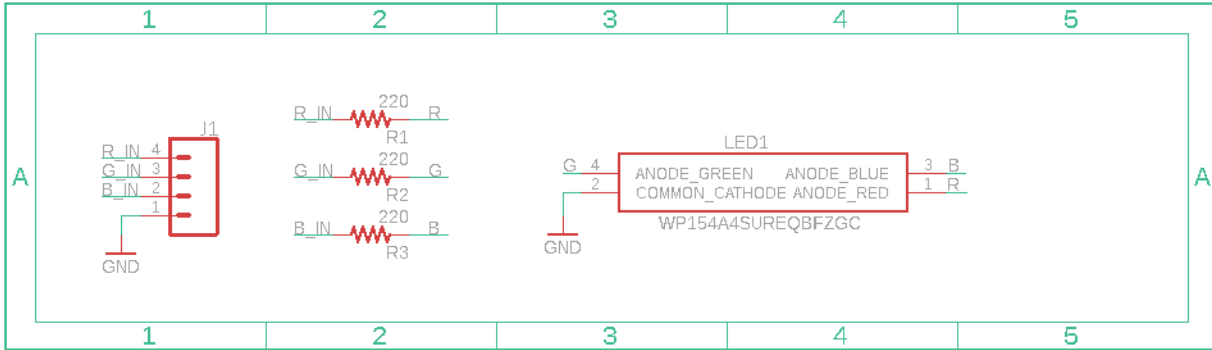


Figure 15.) Eagle SD Card Schematic.

**LED:**

The RGB LED schematic shown in Figure 16 depicts the basic connections needed to interface with an RGB LED. One board will be used per RGB LED. Each board has a connector for power, ground, and data in, and a connector for power, ground, and data out. This will allow for the boards to be connected in series, meaning only one connector will be needed on the main board to connect to the entire series of RGB LEDs. A 0.1uF capacitor is used to reduce noise on the power lines. -HH

## DT04: H.A.L.O. - RGB LED Board



### Mounting Hole



*Figure 16.) Eagle RGB LED Schematic.*

### Processor:

The processor used is the dsPIC33EP512GM706. Figure 17 illustrates the base schematic for the peripherals and features used by the H.A.L.O.. On the left of the processor are the basic components recommended for basic usage: a programming header for a PICKit device, a reset button circuit, and the power capacitors specified by the recommended minimum connection figure in the datasheet (Appendix B). Several extra capacitor footprints will be added to the board in case noise proves to be an issue later on. The schematic also includes the peripherals used by the product: I<sup>2</sup>C and SPI. A UART connection is also included, hoping to use a USB interface for debugging purposes. One RGB\_LED pin is used to send the data to the external RGB LEDs. Lastly, eight debug LEDs are also present for development and will not have to be installed on the final product.

-HH

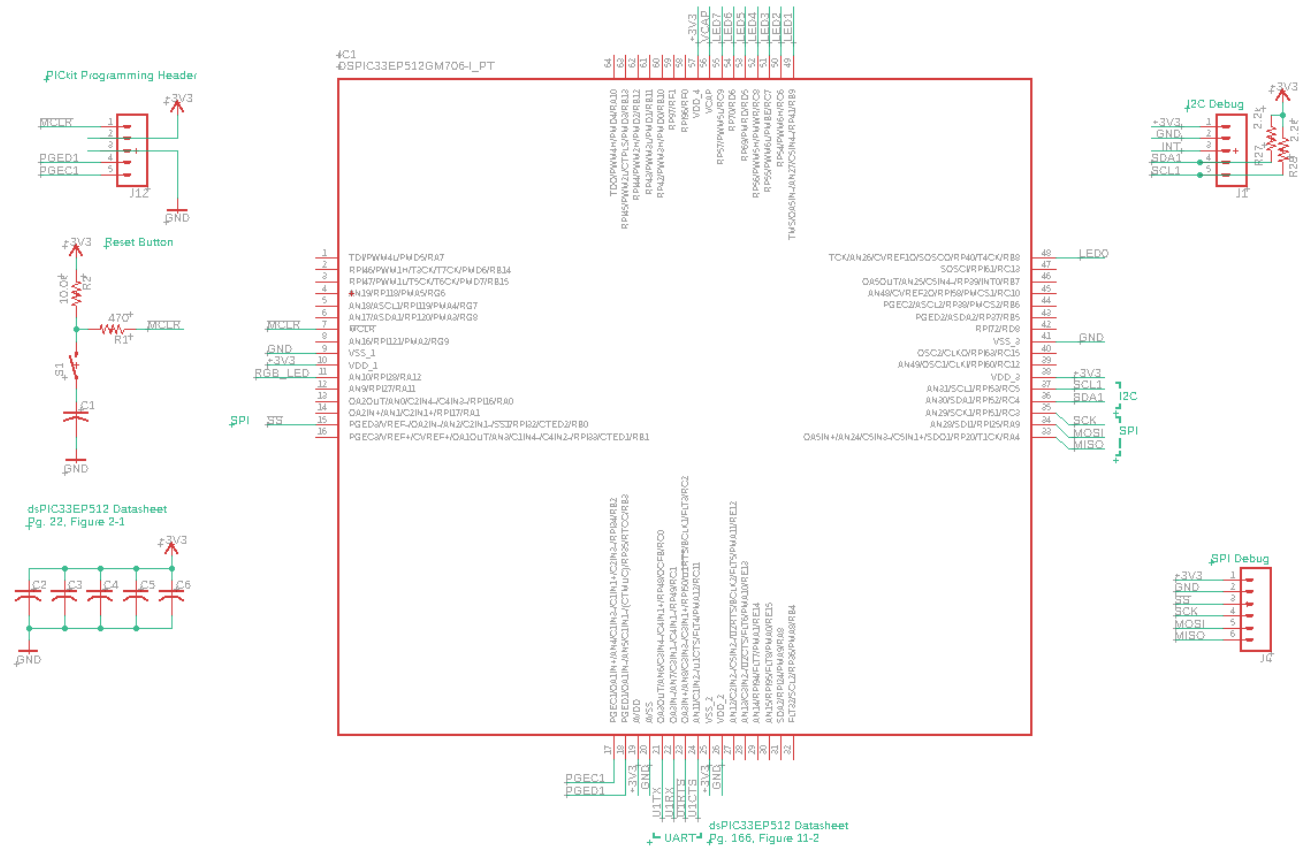


Figure 17.) Eagle Processor Schematic.

### Charging Circuit:

The charging circuit in Figure 18. shows the 5V. input to charging 4.2V circuit from the hardware design power circuit. This is the first section of the power schematic designed in eagle, accounting for the circuit elements required by datasheet specs. The resistors labeled RA, RB, and RPROG all have varying values depending on the charging requirements. They are presently blank since the physical circuit design may require different values depending on current use to meet design needs. The charging device is the TP4056. The input of the 5V. input comes from a micro-usb port. It goes through the series of resistors and capacitors, into the TP4056 device, and connects to two LEDs. These LEDs and their resistors show the circuit is charging or charged



with a RED and Green LED. The output of this circuit outputs 4.2V. directly into a Lithium-Ion Battery that is 3200 (mAH.) With the design below, the charging circuit meets the requirement of charging the battery within 8 hours. The outputs of BAT\_NEG and BAT\_POS are the inputs directly to the battery used in the Battery and Converter Circuit Below. -NK

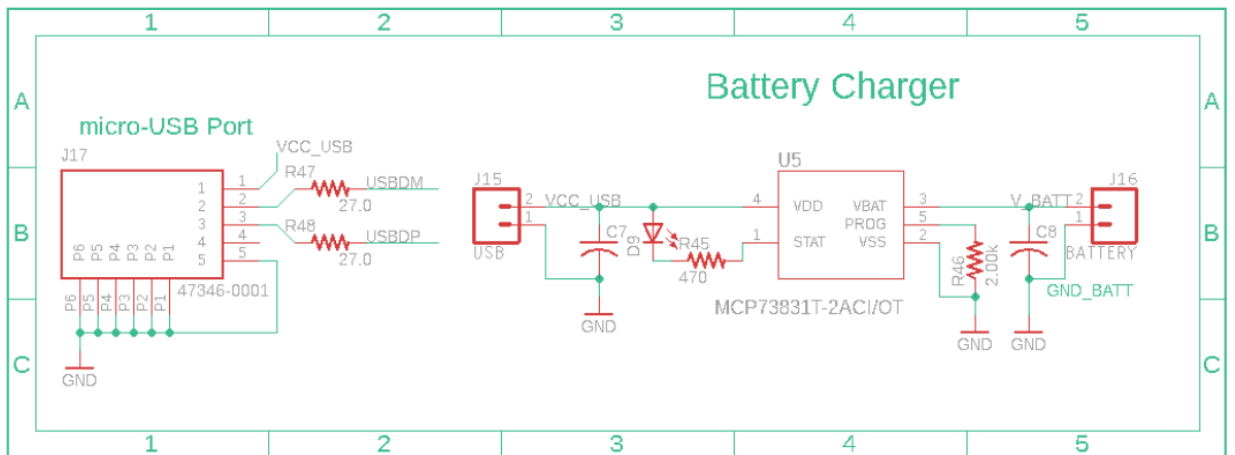


Figure 18.) Eagle Charging Circuit Schematic.

**Battery and Converter:**

The battery and converter circuit in Figure 19. follow directly after the charging circuit. The tags from the output of the charging circuit, BAT\_POS and BAT\_NEG, connect to the battery and buck boost circuit. This schematic has a placeholder 5V battery but will operate at 4.2 in the final design. Circuit components shown assist with operation of converter as described in datasheets. The most important elements besides the converter STBB1-AXX are the resistors acting as a voltage regulator. With 100k and 560k the output voltage V\_OUT, the voltage will be roughly 3.3V to power the rest of the circuits. -NK

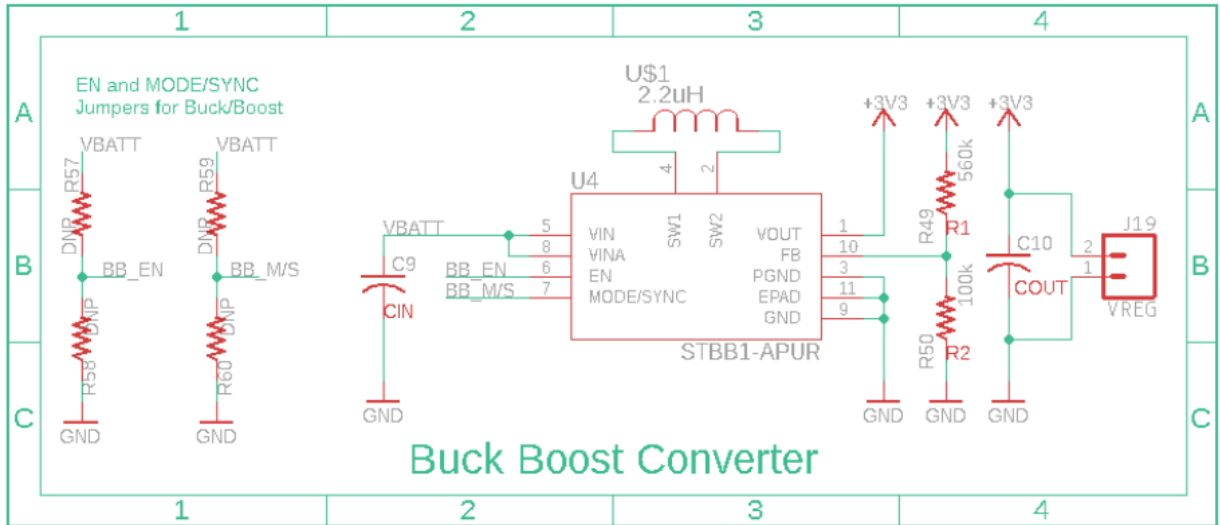


Figure 19.) Eagle Converter Circuit Schematic.

### Time-of-Flight:

The time-of-flight schematic in Figure 20. Is the VL53L0X. The layout of the time-of-flight sensor includes a serial clock line and serial data line that connects to the processor A5/SCL - I<sup>2</sup>C Serial Clock line, SCL1 and A4/SDA - I<sup>2</sup>C Serial Data line SDA1. The 3.3V inputs connect to these lines using two resistors of 10K as pull-ups. The pins of AVDD and AVDDVSEL. connect to the 3V3 tag, which connects to the processors VDD2 output. The GND pins on the processor connect to the GND and the AVSSVCSEL of the time-of-flight. The time-of-flight sensor below can measure up to 2 meters with a potential sampling rate of 400kHz. Finally, the GPIO. connects to the Interrupt with a 1.0K resistor buffer. -NK

## DT04: H.A.L.O. - Time-of-Flight Board

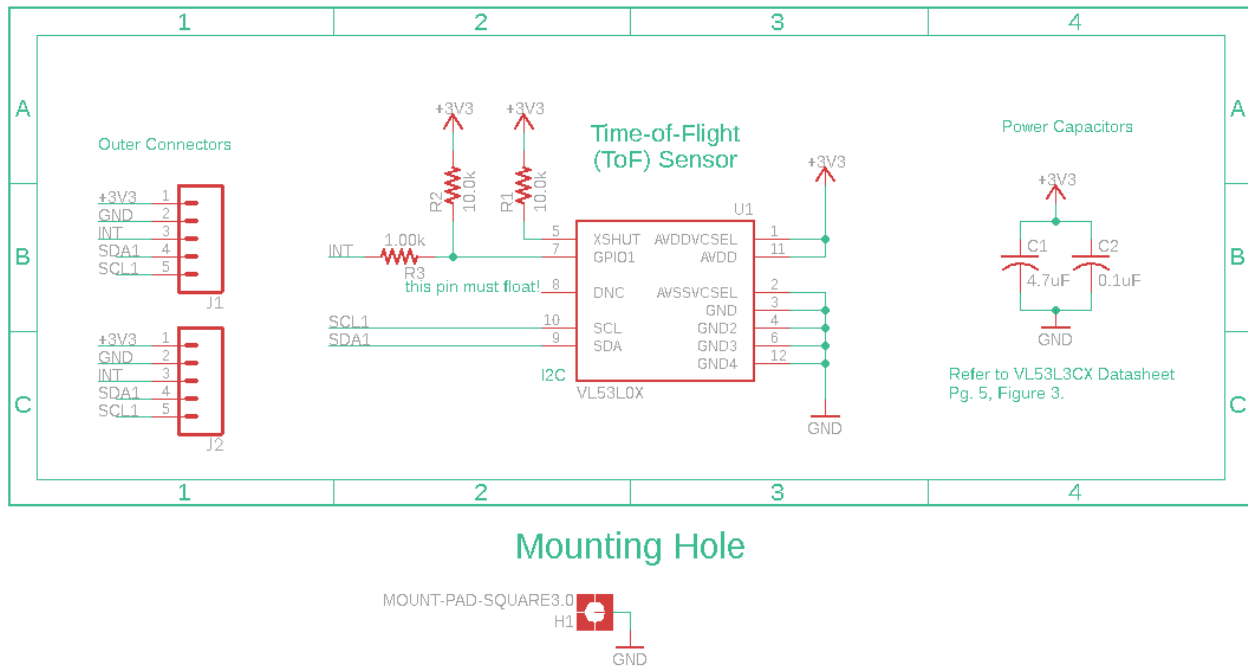


Figure 20.) Eagle Time-of-Flight Schematic.

### 5.2. Software Design:

#### SD Card Subsystem:

The micro-SD card interface uses the SPI1 interface of the dsPIC33EP512GM706. SPI1 was configured using Microchip Code Configurator (MCC) and uses the following libraries: `spi1_driver`, `sd_spi`, and `fatfs`. Setup is handled within the `SYSTEM_Initialize()` function called at startup. The code in Figure 21 achieves a write to the micro-SD card containing accelerometer data in .csv format. The file "ACCEL.CSV" is written with the content of the character arrays `data1[ ]`, `data2[ ]`, `data3[ ]`, `data4[ ]`, and `data5[ ]` which are populated with five consecutive X, Y, and Z axis readings from the accelerometer. The information in each of these arrays is followed by a carriage return and a newline, per .csv formatting. This function is called when the

getAccelPoints() function returns true, indicating that an accelerometer reading has exceeded the set threshold.

```
// write a .CSV containing accelerometer data to the SD card
// pass SD status and File Write status variables for debugging
purposes
void writeAccelToSD(void) {
    // <Nick>
    uint8_t SD_status;
    uint8_t FW_status;
    FATFS drive;          // Work area (filesystem object) for logical
drive
    FIL file;            // File to write
    UINT actualLength;  // Actual length of
    char data0[] = "X, Y, Z, t\r\n";
    char filename[] = "ACCEL.CSV";

    // write 5 data strings in .CSV format for X, Y, and Z axes
    // with millis() timestamps to plot
    sprintf(data1, "%f, %f, %f, %f \r\n", (double)x_1, (double)y_1,
(double)z_1, (double)timer1);
    sprintf(data2, "%f, %f, %f, %f \r\n", (double)x_2, (double)y_2,
(double)z_2, (double)timer2);
    sprintf(data3, "%f, %f, %f, %f \r\n", (double)x_3, (double)y_3,
(double)z_3, (double)timer3);
    sprintf(data4, "%f, %f, %f, %f \r\n", (double)x_4, (double)y_4,
(double)z_4, (double)timer4);
    sprintf(data5, "%f, %f, %f, %f \r\n", (double)x_5, (double)y_5,
(double)z_5, (double)timer5);

    // write the data strings to a file
    if( SD_SPI_IsMediaPresent() == false) {
        return;
    }
    SD_status = f_mount(&drive,"0:", 1);
    if (SD_status == FR_OK) { // mount disk
        //Open or Create <filename> file
        if (f_open(&file, filename, FA_WRITE | FA_CREATE_NEW ) ==
FR_OK) {
            // write column headers
            FW_status = f_write(&file, data0, sizeof(data0)-1,
&actualLength );

            // write each line of data
            FW_status = f_write(&file, data1, sizeof(data1)-1,
&actualLength );
            FW_status = f_write(&file, data2, sizeof(data2)-1,
&actualLength );
```

```

        FW_status = f_write(&file, data3, sizeof(data3)-1,
&actualLength );
        FW_status = f_write(&file, data4, sizeof(data4)-1,
&actualLength );
        FW_status = f_write(&file, data5, sizeof(data5)-1,
&actualLength );

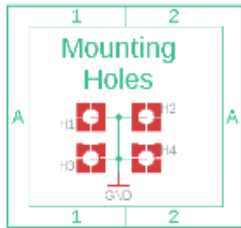
        f_close(&file); // close the file
    }
    f_mount(0,"0:",0); // unmount disk
    msTimerDelay(5);
}
// </Nick>

    showConcussion(); // 5 second LED display:
RED/PURPLE/RED/PURPLE/RED
}

```

*Figure 21.) Micro-SD Card Write Functionality.*

**-HH**



DT04: H.A.L.O. - Main Board

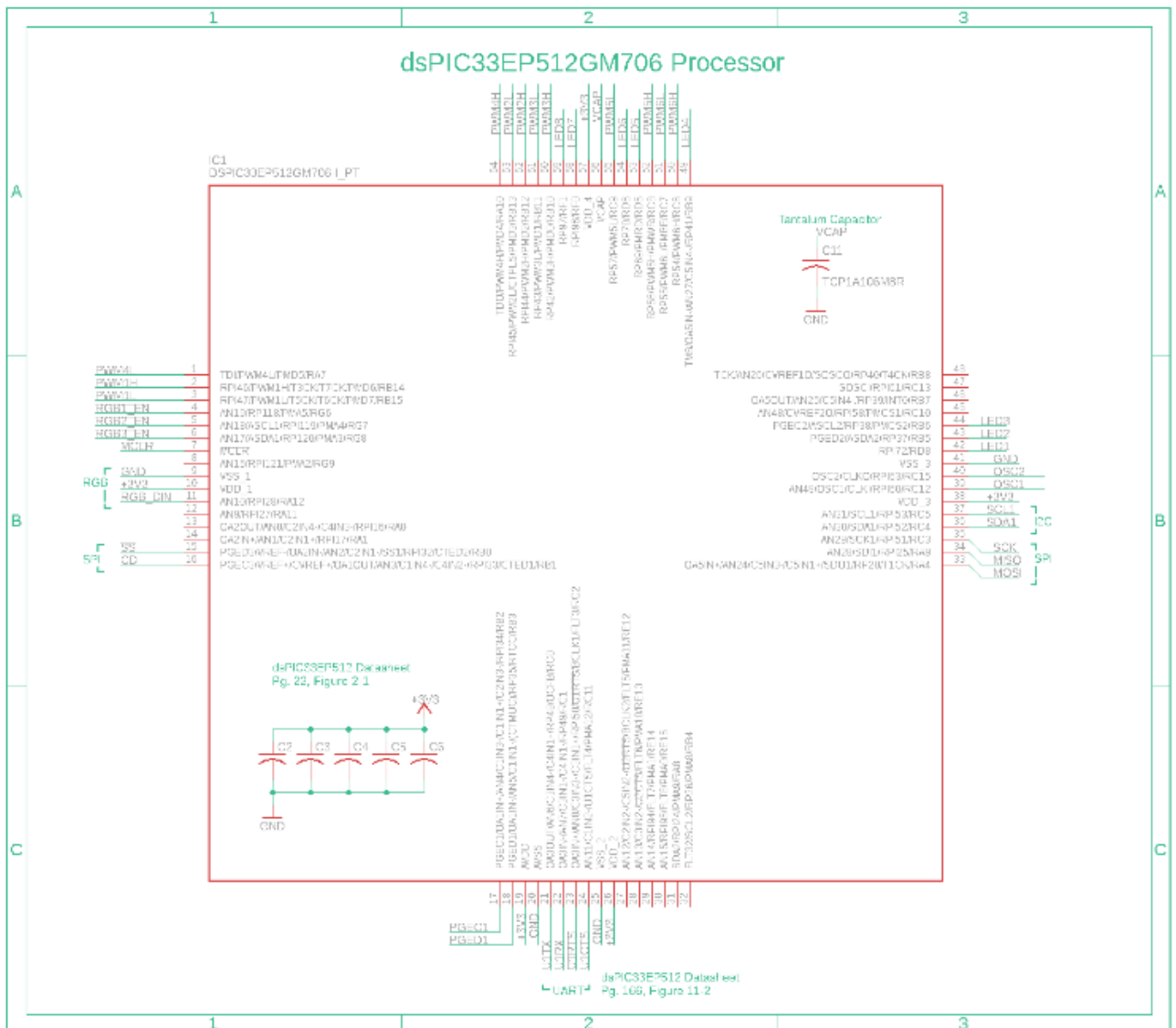
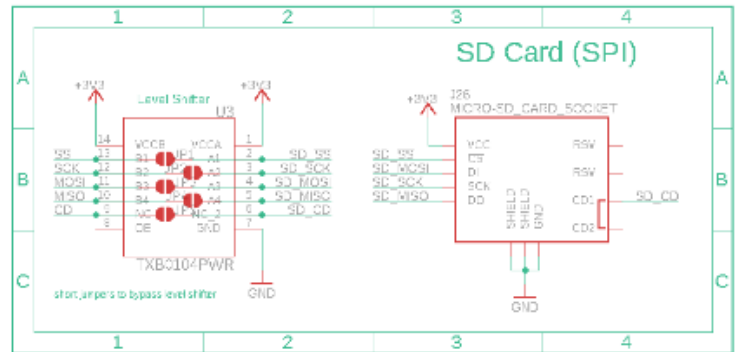


Figure 22.) HALO Main Board Schematic – Processor and SD.

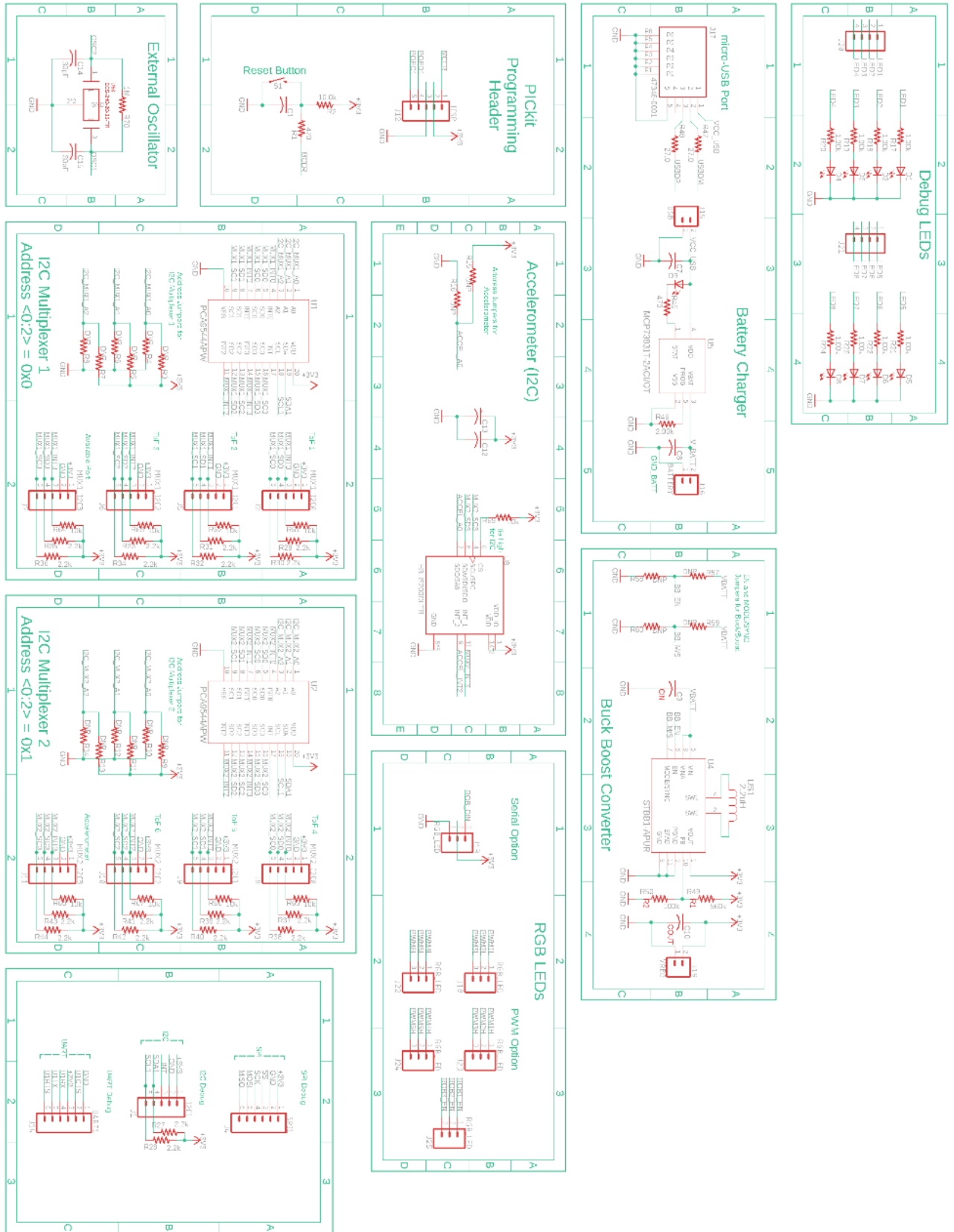


Figure 23.) HALO Main Board Schematic – Peripherals.

# DT04: H.A.L.O. - Time-of-Flight Board

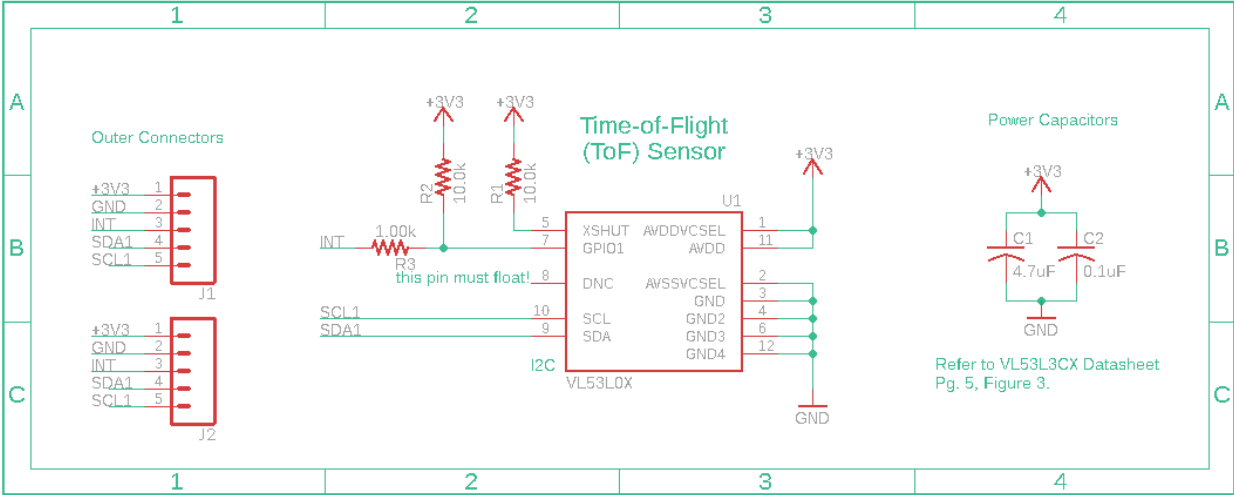
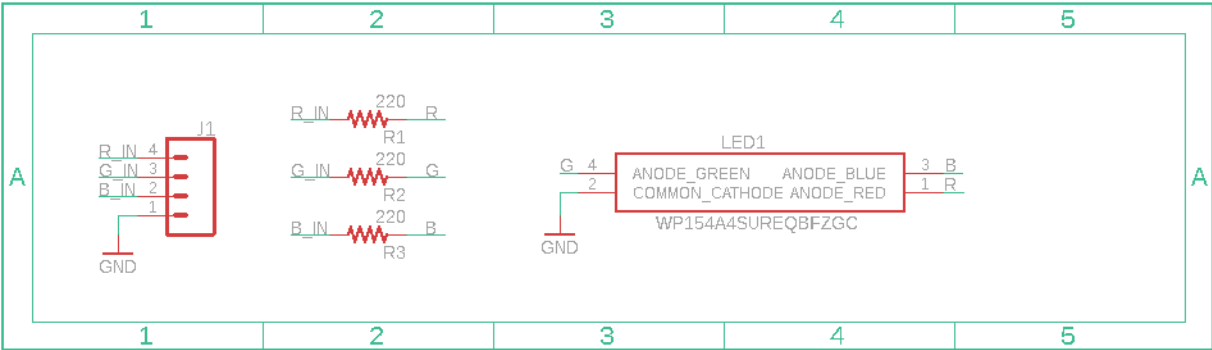


Figure 24.) HALO Time-of-Flight Board Schematic.

# DT04: H.A.L.O. - RGB LED Board



## Mounting Hole



Figure 25.) HALO RGB LED Board Schematic.

### Accelerometer Subsystem:

The accelerometer subsystem uses the I2C1 interface of the dsPIC33EP512GM706. I2C1 was configured using MCC and uses the i2c1 library. The interface also uses the I2C\_Handler



library, which was created from lightly changing the previously commented functions from MCC which deal with reading consecutive bytes of data from the I2C bus. Figure 26 shows the basic setup routine for the accelerometer, configuring two internal registers for acceleration range and sample update time following a soft reset of the device.

```
void H3LIS200DL_begin()
{
    H3LIS200DL_setPowerMode(NORMAL);
    H3LIS200DL_axesEnable(true);

    uint8_t data = 0;

    uint8_t i = 0x21;
    for (i = 0x21; i < 0x25; i++) {
        writeRegister(H3LIS200DL_I2CADDR, i, data);
    }

    uint8_t j = 0x30;
    for (j = 0x30; j < 0x37; j++) {
        writeRegister(H3LIS200DL_I2CADDR, j, data);
    }
}
```

*Figure 26.) Accelerometer Configuration.*

Figure 27 illustrates the process for acquiring data from the accelerometer. The processor moves the memory pointer in the accelerometer device to the start location for each byte of data and the accelerometer then sends each byte of data to the processor, consisting of two bytes per acceleration axis.

```

void H3LIS200DL_readAxes(int16_t* x, int16_t* y, int16_t* z)
{
    uint8_t data[6]; // create a buffer for our incoming data

    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_X_L, &data[0]);
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_X_H, &data[1]);
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_Y_L, &data[2]);
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_Y_H, &data[3]);
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_Z_L, &data[4]);
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_Z_H, &data[5]);
    // The data that comes out is 12-bit data, left justified, so the lower
    // four bits of the data are always zero. We need to right shift by four,
    // then typecase the upper data to an integer type so it does a signed
    // right shift.
    *x = data[0] | data[1] << 8;
    *y = data[2] | data[3] << 8;
    *z = data[4] | data[5] << 8;
    *x = (*x>>4) ;
    *y = (*y>>4) ;
    *z = (*z>>4) ;
}

```

*Figure 27.) Accelerometer Data Acquisition.*

The data acquired by the `H3LIS200DL_readAxes()` function is then interpreted by `getAccelPoints2()`, which will return true if any of the values in the five data sets sampled exceed the set threshold of 50Gs, indicating that a concussion may have occurred.

-HH

### **Time-of-Flight Subsystem:**

The Time-of-Flight subsystem uses a great number of configuration helper functions. These functions were developed regarding ST's application programming interface (API) for the VL53L0X Time-of-Flight sensor. They never made an official register map for the device public, so a crowd sourced register map reference. This reference held true to the definitions found in the official API. Figure 28 lists all helper functions used for configuration and interfacing with the VL53L0X Time-of-Flight sensor.

```

bool VL53L0X_config(void);
bool VL53L0X_setSignalRateLimit(float limit_Mcps);
bool VL53L0X_getSpadInfo(uint8_t * count, bool * type_is_aperture);
uint32_t VL53L0X_getMeasurementTimingBudget(void);
void VL53L0X_getSequenceStepTimeouts(SequenceStepEnables const * enables, SequenceStepTimeouts * timeouts);
uint8_t VL53L0X_getVcselPulsePeriod(vcselPeriodType type);
bool VL53L0X_setMeasurementTimingBudget(uint32_t budget_us);
void VL53L0X_getSequenceStepEnables(SequenceStepEnables * enables);
uint16_t VL53L0X_encodeTimeout(uint16_t timeout_mclks);
bool VL53L0X_performSingleRefCalibration(uint8_t vhw_init_byte);
uint32_t VL53L0X_timeoutMclksToMicroseconds(uint16_t timeout_period_mclks, uint8_t vcsel_period_pclks);
uint32_t VL53L0X_timeoutMicrosecondsToMclks(uint32_t timeout_period_us, uint8_t vcsel_period_pclks);
uint16_t VL53L0X_decodeTimeout(uint16_t reg_val);
void VL53L0X_startContinuous(uint32_t period_ms);
void VL53L0X_stopContinuous(void);
uint16_t VL53L0X_readRangeContinuousMillimeters(void);
uint16_t VL53L0X_readRangeSingleMillimeters(void);
inline void VL53L0X_setTimeout(uint16_t timeout) { io_timeout = timeout; }
inline uint16_t VL53L0X_getTimeout(void) { return io_timeout; }
bool VL53L0X_timeoutOccurred(void);

```

*Figure 28.) Time-of-Flight Functions.*

Several of the functions listed above are called by the VL53L0X\_init() function in Figure 29 via VL53L0X\_config() to do most of the configurations to use the sensor. The timeout and measurement timing budgets are then set, and the sensor can sample data.

```

uint8_t VL53L0X_init(void) {
    uint8_t success = 0;

    if(VL53L0X_config()) { // configure ToF
        success = 1;
    }

    VL53L0X_setTimeout(200); // was 500
    // Start continuous back-to-back mode (take readings as
    // fast as possible). To use continuous timed mode
    // instead, provide a desired inter-measurement period in
    // ms (e.g. sensor.startContinuous(100))

    VL53L0X_setMeasurementTimingBudget(200000); // was 200000
    //VL53L0X_startContinuous(0); // was 0
    VL53L0X_stopContinuous();

    return success;
}

```

*Figure 29.) Time-of-Flight Configuration.*

Once the device is configured as described, the sensor is prepared to provide the processor with a continuous stream of data at a rate determined in the configuration process. Figure 30 breaks down the `VL53L0X_readRangeSingleMillimeters()` function, which is to be called when the program needs to read new data from the sensor while in continuous mode. This returns a sixteen-bit value describing the distance in millimeters between the object in view of the sensor and the sensor itself. If the timeout interval expires before the sensor receives a reflected infrared beam, it is concluded that no object has been detected and the function will return a value of 65535, the maximum value for a sixteen-bit unsigned integer.

```

// Performs a single-shot range measurement and returns the reading in
// millimeters
// based on VL53L0X_PerformSingleRangingMeasurement()
uint16_t VL53L0X_readRangeSingleMillimeters(void) {
    writeRegister(VL53L0X_I2CADDR, 0x80, 0x01);
    writeRegister(VL53L0X_I2CADDR, 0xFF, 0x01);
    writeRegister(VL53L0X_I2CADDR, 0x00, 0x00);
    writeRegister(VL53L0X_I2CADDR, 0x91, stop_variable);
    writeRegister(VL53L0X_I2CADDR, 0x00, 0x01);
    writeRegister(VL53L0X_I2CADDR, 0xFF, 0x00);
    writeRegister(VL53L0X_I2CADDR, 0x80, 0x00);

    writeRegister(VL53L0X_I2CADDR, SYSRANGE_START, 0x01);

    // "Wait until start bit has been cleared"
    startTimeout();
    while (readReg(VL53L0X_I2CADDR, SYSRANGE_START) & 0x01) {
        if (checkTimeoutExpired()) {
            did_timeout = true;
            return 65535;
        }
    }

    return VL53L0X_readRangeContinuousMillimeters();
}

```

*Figure 30.) Time-of-Flight Data Acquisition.*

The process of reading each Time-of-Flight sensor is run continuously by the `getAllToF()` function, detailed in Figure 31, which populates an array of six distance values, one per sensor. This data can then be interpreted by the RGB LED subsystem to display the appropriate lighting patterns to the user.

```
void getAllToF(uint16_t *dists) {
    int i = 0;
    for(i = 0; i < 6; i++) {
        selectPort (ToF[i]);
        msTimerDelay(1);
        dists[i] = VL53L0X_readRangeSingleMillimeters();
        msTimerDelay(1);
    }
}
```

*Figure 31.) Time-of-Flight Data Acquisition.*

-HH

### **RGB LED Subsystem:**

The RGB LED subsystem responds dependent on the readings from the set of six Time-of-Flight sensors. The two ToF sensors to the left drive the left LED, the two ToF sensors in the front drive the middle LED, and the two ToF sensors to the right drive the right LED. As readings are acquired from all six Time-of-Flight sensors, the `getNearestObstacleIndex()` function shown in Figure 32 determines which sensor is detecting the nearest obstacle.

```

// given the array of distances, return the index of the closest object
uint8_t getNearestObstacleIndex(uint16_t *dists) {
    uint8_t index = 0xFF;
    uint16_t min = 0xFFFF;          // 65535

    int i = 0;
    for(i = 0; i < NUM_TOF; i++) {
        if(dists[i] < min) {        // if this is the minimum value
            min = dists[i];         // min is the minimum value
            index = i;              // update the value of index
        }
    }

    return index;                  // 255 indicates all sensors returned 65535
}

```

Figure 32.) *getNearestObstacleIndex( )*.

This index returned by `getNearestObstacleIndex()` is passed to the `showDistanceRGB()` function which will drive the appropriate LED and illuminate it with a different color depending on how far away the nearest obstacle is. If an obstacle is within 500mm (0.5m), the LED will turn red; if an obstacle is within 1000mm (1.0m), the LED will turn green; if an obstacle is within 1500mm (1.5m) the LED will turn blue; and if no obstacle is within 1900mm (1.9m), the LED will turn off. The `showDistanceRGB()` function is shown in detail in Figure 33.

```

void showDistanceRGB(uint16_t dist, LED_posn LED) {
    if(1900 < dist) {
        showBinary(0x00);
    }
    else if(LED == LED_C) {
        if(dist <= 500) {
            showBinary(MID_RED);
        } else if(500 < dist && dist <= 1000) {
            showBinary(MID_GRN);
        } else if(1000 < dist && dist <= 1500) {
            showBinary(MID_BLU);
        }
    }
    else if(LED == LED_L) {
        if(dist <= 500) {
            showBinary(LR_RED | R_OFF);
        } else if(500 < dist && dist <= 1000) {
            showBinary(LR_GRN | R_OFF);
        } else if(1000 < dist && dist <= 1500) {
            showBinary(LR_BLU | R_OFF);
        }
    }
    else if(LED == LED_R) {
        if(dist <= 500) {
            showBinary(LR_RED | L_OFF);
        } else if(500 < dist && dist <= 1000) {
            showBinary(LR_GRN | L_OFF);
        } else if(1000 < dist && dist <= 1500) {
            showBinary(LR_BLU | L_OFF);
        }
    }
}
}

```

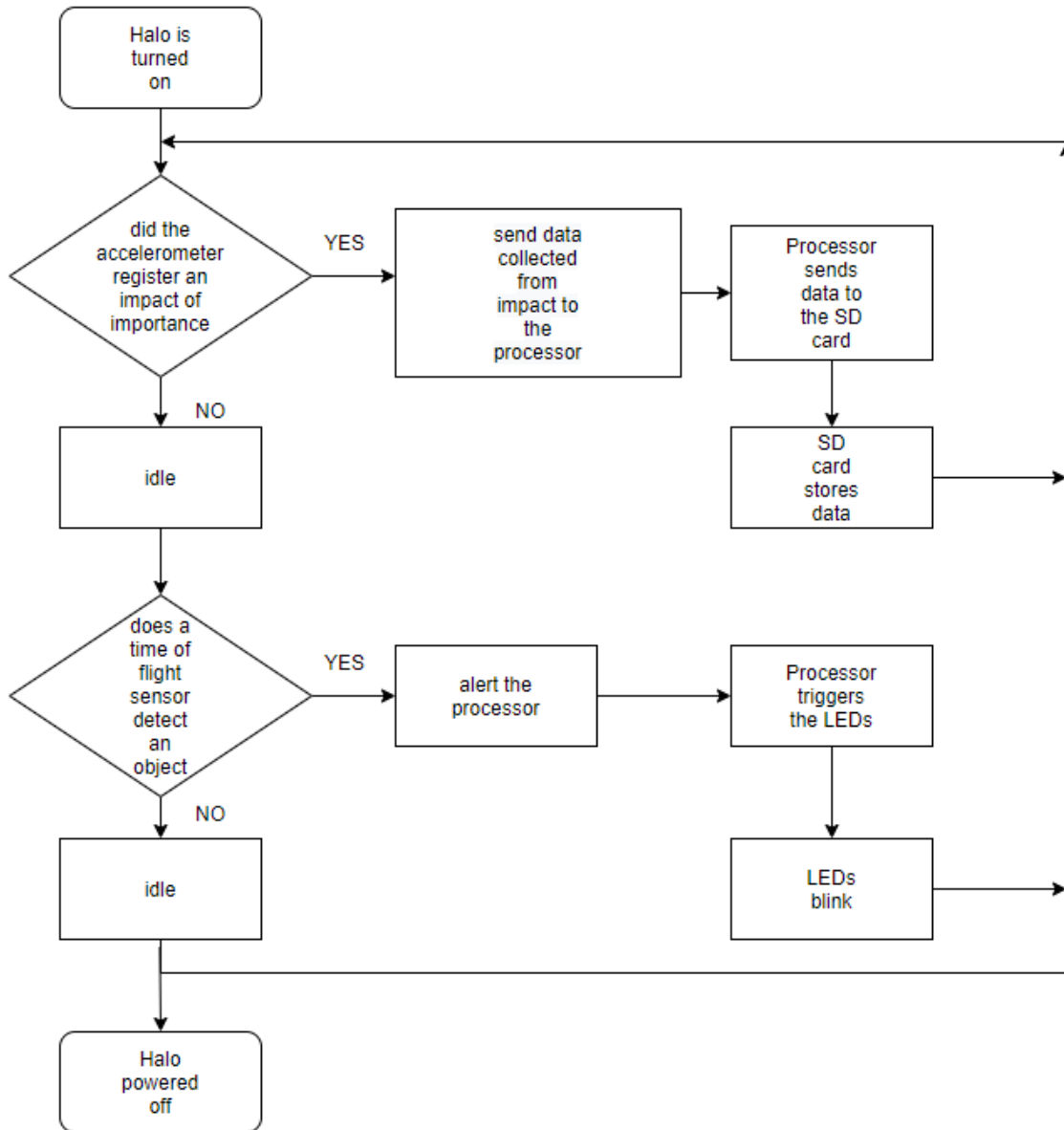
Figure 33.) *showDistanceRGB( )*.

### Software System Overview:

The flow chart in figure 34 below describes how the system will work in the field. Once the system is powered on there will be an interrupt check for the accelerometer to reach a threshold to write the data to the SD card of a potential head injury. If nothing is detected by the accelerometer, there is another check that is made by the interrupt lines of the time-of-flight sensors as to if a hazard is within range. If there is, then the interrupts trigger the LED's turning

on for the corresponding time-of-flight sensors. If not, then the accelerometer starts its checks again. This is a constant loop until the system is powered off.

-NW



-NW

Figure 34.) Software Design Flowchart.



The table below depicts the inputs, outputs, and functionality of the SPI bus used in this design. The inputs and outputs for the SPI bus communication signals with a frequency of up to 10MHz. -BT

*Table 13.) SPI Functionality Table.*

Communication Type	SPI
Designer	Hunter Hykes, Nicklaus Walsh
Inputs	Up to 10 MHz communication input
Outputs	Up to 10 MHz communication output.
Functionality	SPI is a bus style communication method used to write data to a micro-SD card.

The table below depicts the inputs, outputs, and functionality of the I2C bus used in this design. The inputs and outputs for the I2C bus communication signals with a frequency of up to 1MHz. -BT

*Table 14.) I2C Functionality Table.*

Communication Type	I2C
Designer	Hunter Hykes, Nicklaus Walsh
Inputs	Up to 1 MHz communication input
Outputs	Up to 1 MHz communication output
Functionality	I2C is a bus style communication method used to communicate with accelerometer and time-of-flight sensors.

Figures 35 and 36 below describe the different states for the accelerometer, SD card, LED's, and Time-of-Flight sensors. The first figure, the SD card and accelerometer, describes the different states of writing for the data going to the accelerometer to the SD card. If the accelerometer is writing data, then the SD card is being written to. The next diagram for the time-of-flight sensors describes the state of an LED being on or off in response to a ToF detecting a hazard.

-NW

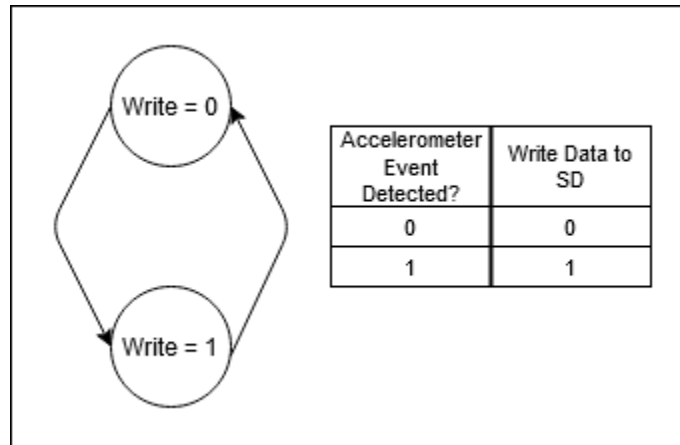


Figure 35.) SD Card State Diagram.

-HH

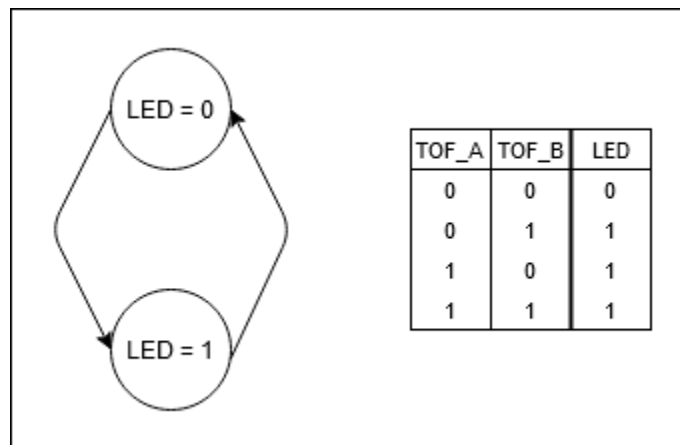


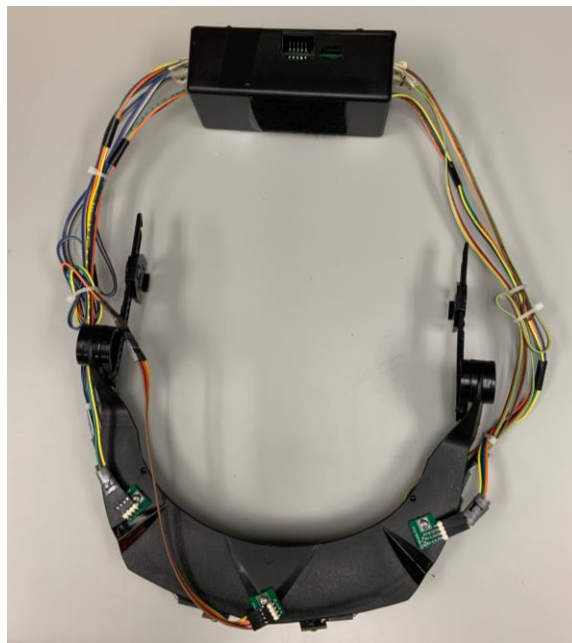
Figure 36.) LED State Diagram.

-HH

### 5.3. Final Design Images:



*Figure 37.) Frontal View of Housing Unit, ToF Sensors and LEDs.*



*Figure 38.) Detached Housing Unit.*

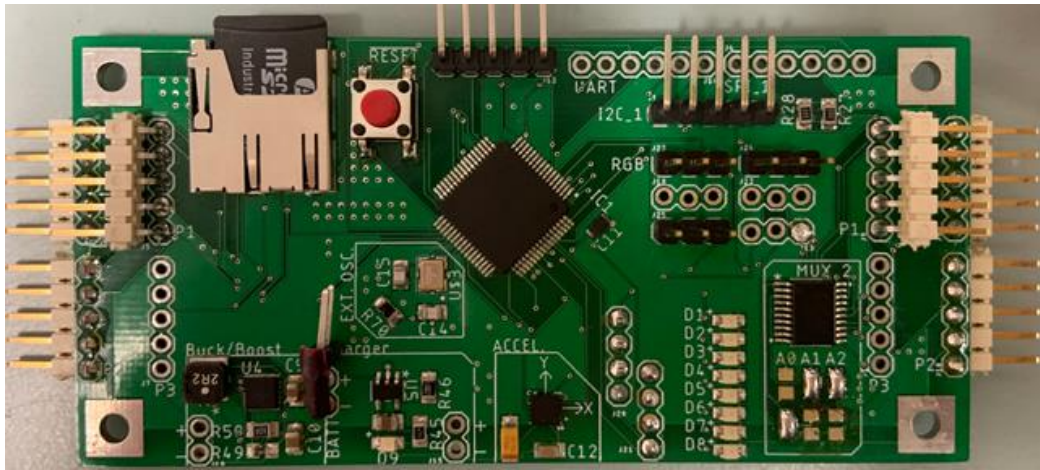


Figure 39.) Main Board Front.

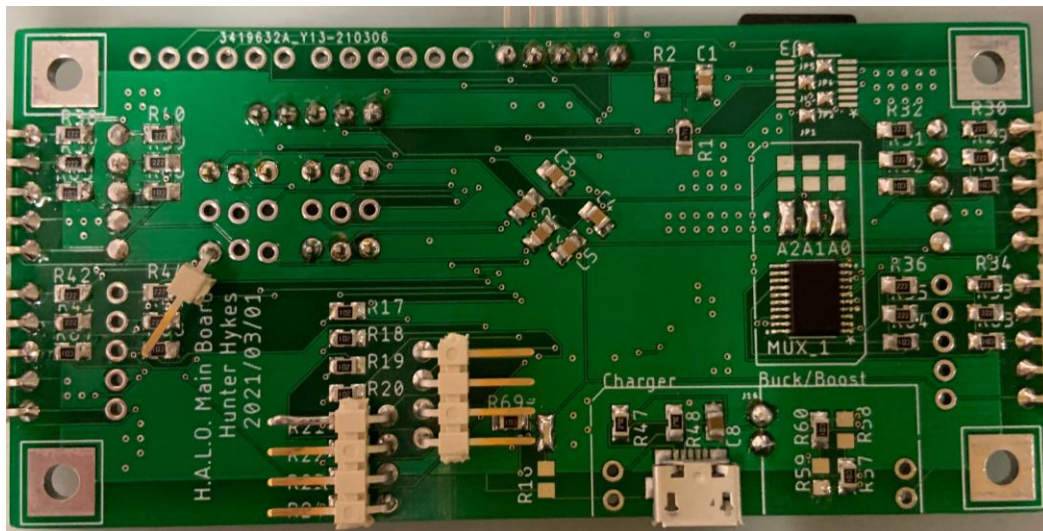


Figure 40.) Main Board Back.

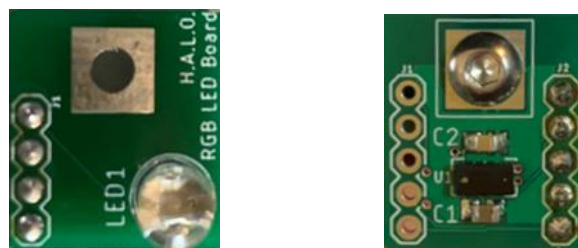


Figure 41.) Time of Flight and LED Boards.

#### 5.4. Testing and Demonstrations:

##### Time of Flight and LED Subsystems:

The following images depict the testing for the engineering requirements of 6,7,8,9. The testing procedure was to ensure that the LEDs light up with respect to the distancing of potential hazards and that they change colors between 1.5m to the user. For the furthest distance of 1.5m a blue light is used, from (1m) to (.5m) a green light is used and (.5m) to the user, a red light is illuminated. Another requirement that is shown in these testing images is that the FOV for the time-of-flight sensors is 150-degrees. For the demonstration in these images, a clipboard was used as the mock hazardous object.

-NK



*Figure 42.) LED and ToF Right Side Green.*



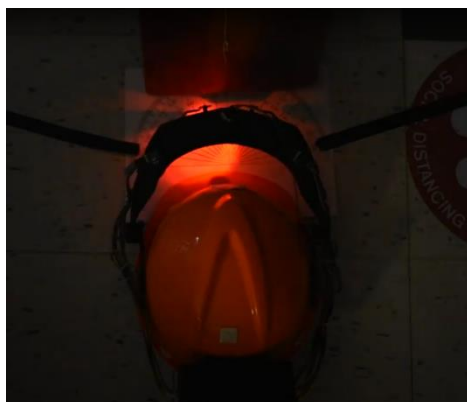
*Figure 43.) LED and ToF Right Side Red.*



*Figure 44.) LED and ToF Left Side Green.*



*Figure 45.) LED and ToF Left Side Red.*



*Figure 46.) LED and ToF Front Red.*



*Figure 47.) LED and ToF Front Blue.*



*Figure 48.) LED and ToF Max Distance Test.*



*Figure 49.) LED and ToF Max Distance 49 inches (~1245mm).*

As can be seen above in the testing images, the engineering requirements were met for the FOV and color shifting. The only one that wasn't exactly reached was the distance requirement for the ToF sensor. The range was 1.25 shy. This could be easily fixed in future designs by using



a ToF sensor with slightly larger range. With minimal changes to the code, this could be implemented with the new sensor. -NK

### Power System Testing:

For the power system engineering requirements, it was needed that the battery must last for an 8-hour workday and that the battery can recharge within 8 hours. The testing procedure for both of these was simple. First the battery was discharged over the designated time frame while voltage readings were taken every 30 minutes. After that was done, the battery was then hooked up to a standard micro-usb charging cable and recorded until it recharged to full capacity. Below are images of the recording set up and tables depicting the discharge and recharge times. -NK



*Figure 50.) Initial Voltage Before Discharge.*



*Figure 51.) Final Voltage After 8 Hours.*

*Table 15.) Discharge Testing Recordings.*

<i>Time Passed</i>	<i>Voltage</i>
<i>Initial</i>	<i>4.062 V</i>
<i>.5 Hour</i>	<i>4.058 V</i>
<i>1 Hour</i>	<i>4.053 V</i>
<i>1.5 Hours</i>	<i>4.045 V</i>
<i>2 Hours</i>	<i>4.032 V</i>
<i>2.5 Hours</i>	<i>4.013 V</i>
<i>3 Hours</i>	<i>4.005 V</i>
<i>3.5 Hours</i>	<i>3.999 V</i>
<i>4 Hours</i>	<i>3.976 V</i>
<i>4.5 Hours</i>	<i>3.960 V</i>
<i>5 Hours</i>	<i>3.943 V</i>

<i>5.5 Hours</i>	<i>3.931 V</i>
<i>6 Hours</i>	<i>3.923 V</i>
<i>6.5 Hours</i>	<i>3.914 V</i>
<i>7 Hours</i>	<i>3.900 V</i>
<i>7.5 Hours</i>	<i>3.985 V</i>
<i>8 Hours</i>	<i>3.880 V</i>

As can be seen from the slow discharge rate of the battery, the system consumes much less power than initially assumed. Conservative power estimations were done to ensure that the battery life lasts a full 8-hour workday. With the usage of a buck boost converted, the battery would be able to supply power to the system until it discharges to roughly two volts. However, recharging after an 8-hour use, it took roughly 7.5 hours to completely recharge. The battery is able to supply the system with power much longer than the 8-hour requirement. In future designs it would be possible to reduce the battery size in order to accommodate a smaller less intrusive design for the housing. -NK

*Table 16.) Recharge Testing Recordings.*

<i>Time Passed</i>	<i>Voltage</i>
<i>Initial</i>	<i>3.783 V</i>
<i>1.5 Hours</i>	<i>4.012 V</i>
<i>3 Hours</i>	<i>4.097 V</i>
<i>4.5 Hours</i>	<i>4.123 V</i>
<i>6 Hours</i>	<i>4.142</i>
<i>7.5Hours</i>	<i>4.159</i>

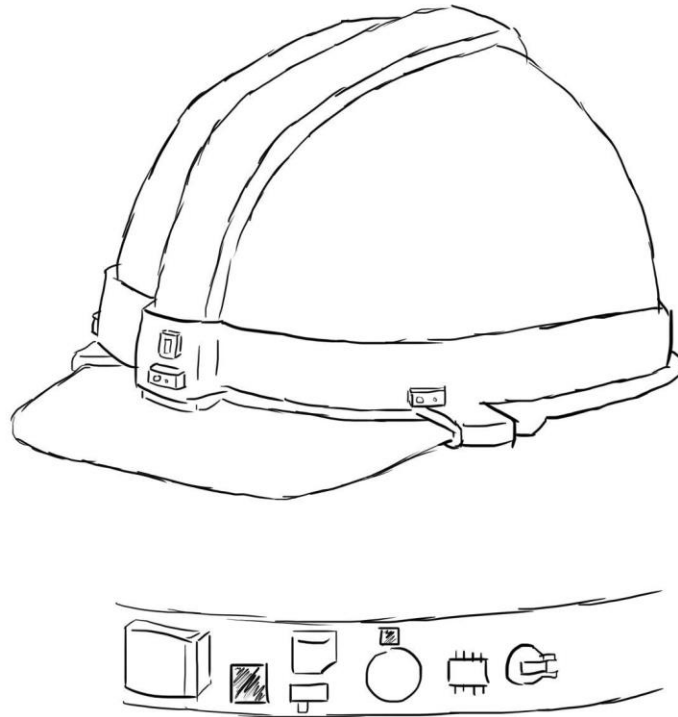
### Accelerometer and SD Card Testing:

The engineering requirements for the SD card and the Accelerometer systems we had to write suspected impact or injury data to an on-board storage system that could be retrieved later. This was in the form of the on-board SD card located on the main board. This data should also include relative timestamps relating to the impact or injury data. This was handled in the coding of the Accelerometer and SD card. Finally, the data should only be recorded if the accelerometer measures an impact at or above a predetermined threshold these reading will be the indicator for an incident.

The way these were tested was, initially, to use the onboard LEDs to get the Accelerometer data. This was done to make sure we were getting actual data from what the Accelerometer was measuring that was shown on the LEDs. We then tested to make sure the SD card could write dummy data on our board similar to how the proof of concept had shown to make sure it would work on the developed PCB rather than a development board. Once that was done we then transitioned to integrating the two sub-systems so that they interact in the way we want. This involved having to format the C-strings to be written as a CSV file. Then we wrote the data measured from the Accelerometer to the SD card only if it was reading a value at or above the threshold we chose for testing. We did not want to use the actual value for a concussion for the reasons of not wanting to actually hurt someone or potentially break another sub-system on the board. We then set that threshold and with a relatively hard smack to the helmet, the impact routine would start. The impact routine was that the LEDs for the vision detection would flash showing the Accelerometer measured a reading at or above the pre-set threshold then within the coding that took the data from the Accelerometer and formatted it and wrote it to the SD card for analysis at a later time. -NW






## 6.) Mechanical Sketch of System:

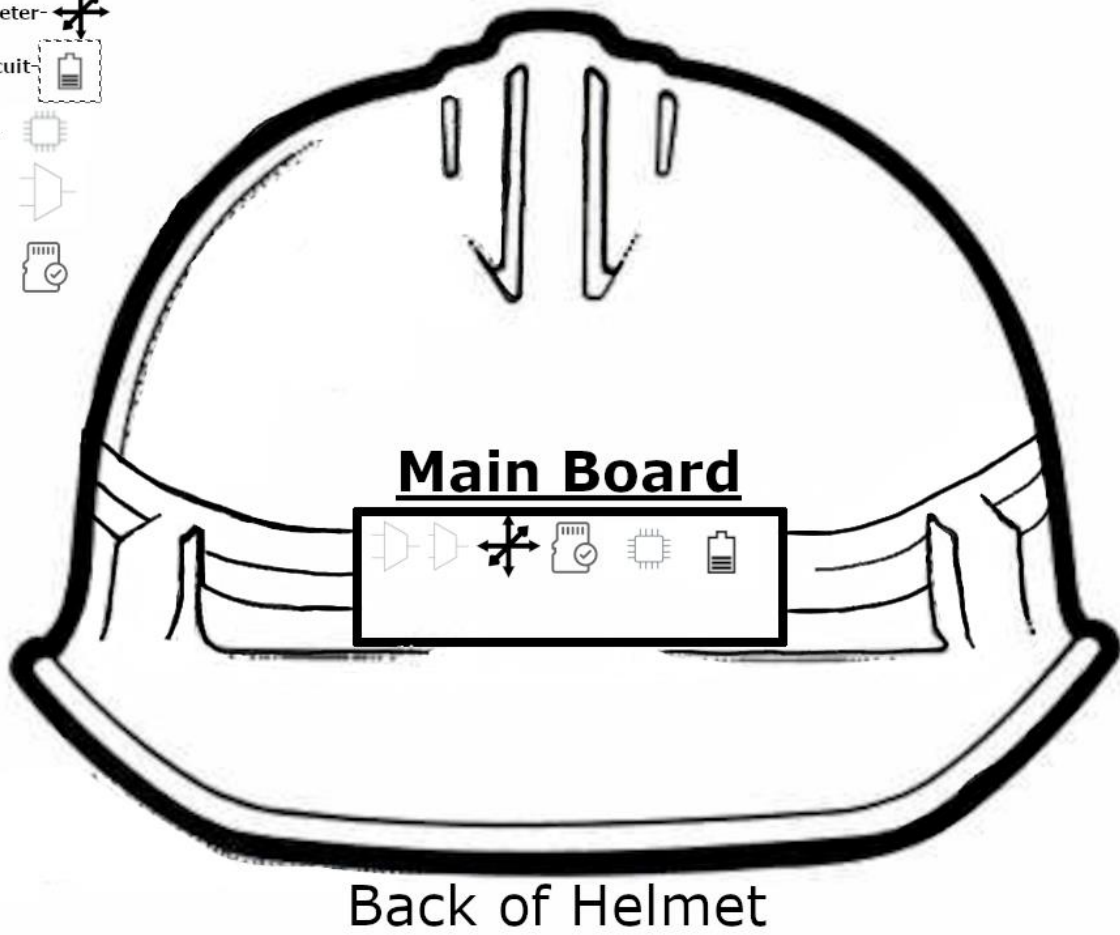
The following mechanical sketches in figure 52 through 54 show the proposed implementation of the H.A.L.O. system. The first hand-drawn sketch shows the band and placement of the sensors, LEDs, and board on the back of the system. The second figure shows a more detailed design, showing the back of the helmet and symbols representing the circuit elements that will be on the main board. Finally, the last sketch shows a frontal view of how the sensors and LEDs fit to the helmet. Note that for the final image, not every sensor or LED is shown. There are in total six ToF sensors and three LEDs in the final design. -NK



*Figure 52.) Mechanical Sketch of System (Overall).*

**Legend:**

- Accelerometer- 
- Power Circuit- 
- Processor- 
- I2C MUX- 
- SD Card- 

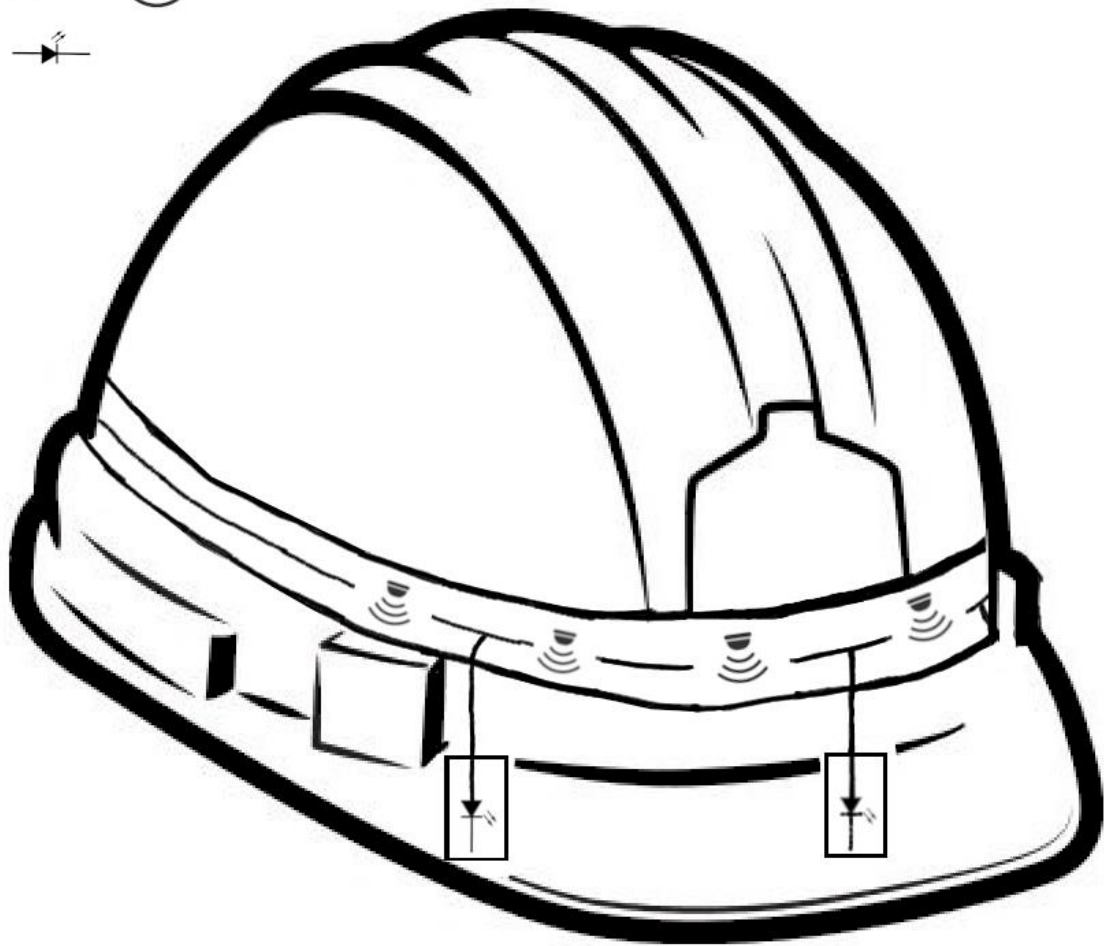


*Figure 53.) Mechanical Sketch of System (Rear View).*

## Legend:      **Front of Helmet**

ToF Sensor- 

LED- 



*Figure 54.) Mechanical Sketch of System (Front View).*

-NK

### **7.) Team Information:**

Hunter Hykes – Computer Engineer

Nathan Kish – Electrical Engineer

Brian Thomson – Electrical Engineer

Nicklaus Walsh – Computer Engineer

## 8.) Parts List:

Table 17.) Main Board Bill of Materials.

Qty.	Refdes	Part Num.	Description	Cost	Total Cost
12	C1~C6	CL21B104KBCNNN C	0.1uF Ceramic Capacitor 0805	\$0.05	\$0.56
4	C7~C8	CL21A475KPFNNN E	4.7uF Ceramic Capacitor 0805	\$0.10	\$0.40
2	C9	CL21A106MQFNNN E	10uF Ceramic Capacitor 0805	\$0.10	\$0.20
2	C10	CL21A226MAQNNN E	22uF Ceramic Capacitor 0805	\$0.34	\$0.68
16	R17~R24	RC2012J102CS	1.00kOhm Resistor 0805	0.13	\$2.08
6	R2, R25~R26	RC2012J103CS	10.0kOhm Resistor 0805	0.13	\$0.78
36	R27~R44	RC2012J222CS	2.20kOhm Resistor 0806	0.13	\$4.68
4	R1, R45	RC2012J471CS	470Ohm Resistor 0805	0.14	\$0.56
2	R46	ERA-6AEB202V	2.00kOhm Resistor 0805	0.31	\$0.62
4	R47~R48	RC2012F270CS	27.0Ohm Resistor 0805	0.14	\$0.56
2	R49	RMCF0805JT560R	560kOhm Resistor 0805	0.10	\$0.20
2	R50	ERA-6AEB104V	100kOhm Resistor 0805	0.31	\$0.62
16	D1~D8	LY R976-PS-36	LED YELLOW DIFFUSED 0805 SMD	0.27	\$4.32
2	D9	LG R971-KN-1	LED GREEN DIFFUSED 0805 SMD	0.25	\$0.50
18	J1~J2, J5~J12	2011- 1X05TSD025B	Connector Header Through Hole 5 position 0.100"	0.84	\$15.12
2	J3	DM3D-SF	CONN MICRO SD CARD PUSH- PULL	1.65	\$3.30
1	J13	2011- 1X03G00SD025B	PIN HEADER, SINGLE ROW, 3 PIN, S	0.06	\$0.06
1	J14	2011H-1X06G01SB	PIN HEADER, SINGLE ROW, 6 PIN, S	0.87	\$0.87
4	J15~J16	2011- 1X02TSH035B	PIN HEADER, SINGLE ROW, 2 PIN, T	0.51	\$2.04
2	J17	473460001	CONN RCPT USB2.0 MICRO B SMD R/A	0.95	\$1.90
3	IC1	DSPIC33EP512GM 706-I/PT	Microchip DSPIC33EP512GM706- I/PT, 16bit dsPIC Microcontroller, 60MHz, 512 kB Flash, 64-Pin TQFP	6.17	\$18.51
4	U1~U2	PCA9544APW,118	I2C Multiplexer	1.78	\$7.12
2	U3	TXB0104PWR	Voltage Level Translator Bidirectional 1 Circuit 4 Channel 100Mbps 14-TSSOP	0.92	\$1.84
2	U4	STBB1-APUR	Buck-Boost Converter	2.34	\$4.68
2	U5	MCP73831T- 2ACI/OT	Battery Charging	0.56	\$1.12



2	U8	H3LIS200DLTR	H3LIS200DL Series 3.6 V 400 Hz Low-Power 3-Axis Digital Accelerometer -TFLGA-16L	6.68	\$13.36
2	S1	CT11025.0F160	Momentary Switch	0.12	\$0.24
	(ToF Brd)				
12	U1	VL53L0CXV0DH/1	Time-of-Flight ranging sensor	3.88	\$46.56
2	C1	CL21A475KPFNNN E	4.7uF Ceramic Capacitor 0805	\$0.10	\$0.20
2	C2	CL21B104KBCNNN C	0.1uF Ceramic Capacitor 0805	\$0.05	\$0.09
4	R1~R2	RC2012J103CS	10.0kOhm Resistor 0805	0.13	\$0.52
2	R3	RC2012J102CS	1.00kOhm Resistor 0805	0.13	\$0.26
4	J1~J2	2011-1X05TSD025B	Connector Header Through Hole 5 position 0.100"	0.84	\$3.36
	(RGB Brd)				
6	LED1	1655	ADDRESS LED DISC SERIAL RGB 1=10	4.50	\$27.00
6	J1~J2	2011-1X03G00SD025B	PIN HEADER, SINGLE ROW, 3 PIN, S	0.06	\$0.36
6	C1	CL21B105KAFNFN E	CAP CER 1UF 25V X7R 0805	0.10	\$0.60
				<b>Total</b>	<b>\$165.88</b>

## 9.) Project Schedule:

### Mid-Semester Gantt Chart

This section shows the beginning of the semester to the mid-point. This part of the Gantt chart covers the initial engineering requirements research and beginning of the technical design. During this portion of the Gant chart, the split of work is visible, assigning each team member a subsystem and engineering requirements research required for mid-term presentations. -NK

ID	Task Name	Duration	Start	Finish
1	SDP I 2020			
2	<b>Project Design</b>	<b>93 days</b>	<b>Mon 8/24/20</b>	<b>Wed 11/25/20</b>
3	<b>Midterm Report</b>	40 days	Wed 8/26/20	Mon 10/5/20
4	Cover page	40 days	Wed 8/26/20	Mon 10/5/20
5	T of C, L of T, L of F	40 days	Wed 8/26/20	Mon 10/5/20
6	<b>Problem Statement</b>	<b>40 days</b>	<b>Wed 8/26/20</b>	<b>Mon 10/5/20</b>
7	Need	40 days	Wed 8/26/20	Mon 10/5/20
8	Objective	40 days	Wed 8/26/20	Mon 10/5/20
9	Background	40 days	Wed 8/26/20	Mon 10/5/20
10	Marketing Requirements	40 days	Wed 8/26/20	Mon 10/5/20
11	Engineering Requirements Specification	40 days	Wed 8/26/20	Mon 10/5/20
12	<b>Engineering Analysis</b>	<b>40.38 days</b>	<b>Wed 8/26/20</b>	<b>Mon 10/5/20</b>
13	Circuits (DC, AC, Power, ...)	40.38 days	Wed 8/26/20	Mon 10/5/20
14	Electronics (analog and digital)	40.38 days	Wed 8/26/20	Mon 10/5/20
15	Signal Processing	40 days	Wed 8/26/20	Mon 10/5/20
16	Communications (analog and digital)	40 days	Wed 8/26/20	Mon 10/5/20
17	Electromechanics	40 days	Wed 8/26/20	Mon 10/5/20
18	Computer Networks	40 days	Wed 8/26/20	Mon 10/5/20
19	Embedded Systems	40.38 days	Wed 8/26/20	Mon 10/5/20
20	<b>Accepted Technical Design</b>	<b>40.38 days</b>	<b>Wed 8/26/20</b>	<b>Mon 10/5/20</b>
21	<b>Hardware Design: Phase 1</b>	<b>32.38 days</b>	<b>Wed 8/26/20</b>	<b>Sun 9/27/20</b>
22	Hardware Block Diagrams Levels 0 thru N (w/ FR tab	32.38 days	Wed 8/26/20	Sun 9/27/20
23	<b>Software Design: Phase 1</b>	<b>32.38 days</b>	<b>Wed 8/26/20</b>	<b>Sun 9/27/20</b>
24	Software Behavior Models Levels 0 thru N (w/FR tab	32.38 days	Wed 8/26/20	Sun 9/27/20
25	<b>Mechanical Sketch</b>	40 days	Wed 8/26/20	Mon 10/5/20
26	<b>Team information</b>	40 days	Wed 8/26/20	Mon 10/5/20
27	<b>Project Schedules</b>	40 days	Mon 8/24/20	Sat 10/3/20
28	<b>Midterm Design Presentations Day 2</b>	<b>7.38 days</b>	<b>Wed 9/30/20</b>	<b>Wed 10/7/20</b>
29				
30	Midterm Design Gantt Chart	40 days	Wed 8/26/20	Mon 10/5/20
31	<b>References</b>	40 days	Wed 8/26/20	Mon 10/5/20
32	<b>Midterm Parts Request Form</b>	47 days	Wed 8/26/20	Mon 10/12/20
33	<b>Midterm Design Presentations Day 1</b>	<b>7.38 days</b>	<b>Wed 9/23/20</b>	<b>Wed 9/30/20</b>
34				
35	<b>Project Poster</b>	14 days	Wed 10/21/20	Wed 11/4/20
36	<b>Final Design Report</b>	50 days	Tue 10/6/20	Wed 11/25/20
37	<b>Abstract</b>	48 days	Tue 10/6/20	Mon 11/23/20
38	<b>Hardware Design: Phase 2</b>	<b>48 days</b>	<b>Tue 10/6/20</b>	<b>Mon 11/23/20</b>
39	<b>Modules 1...n</b>	<b>48 days</b>	<b>Tue 10/6/20</b>	<b>Mon 11/23/20</b>
40	Simulations	48 days	Tue 10/6/20	Mon 11/23/20
41	Schematics	48 days	Tue 10/6/20	Mon 11/23/20
42	<b>Software Design: Phase 2</b>	<b>48 days</b>	<b>Tue 10/6/20</b>	<b>Mon 11/23/20</b>
43	<b>Modules 1...n</b>	<b>48 days</b>	<b>Tue 10/6/20</b>	<b>Mon 11/23/20</b>

% Complete	Work Stage	Team Member
0%		
<b>69%</b>		
95%	Complete	Everyone
100%	Complete	Everyone
100%	Complete	Brian
<b>100%</b>	<b>Complete</b>	
100%	Complete	Nate
100%	Complete	Hunter, Nick
100%	Complete	Everone
100%	Complete	Nate
100%	Complete	Everyone
<b>83%</b>	<b>Preliminary Calculations</b>	
74%	Preliminary Calculations	Brian, Nate
94%	Preliminary Calculations	Everyone
0%		
0%		
0%		
0%		
79%	Preliminary Calculations	Nick, Hunter
<b>100%</b>		
<b>100%</b>		
100%		Everyone
<b>100%</b>		
100%		Nick, Hunter
50%	Rough Draft	Hunter
100%	Complete	Everyone
20%	Preliminary Schedule Complete	Nate
<b>100%</b>		
100%		Nate
30%	Background Refrences	Everyone
0%		
0%	Team 4 Presented	
0%		
0%		
0%		
5%		
5%		
0%		
10%	Power Circuit Schematic Designed	Brian
<b>0%</b>		
<b>0%</b>		

ID	Task Name	Duration	Start	Finish
44	Code (working subsystems)	48 days	Tue 10/6/20	Mon 11/23/20
45	System integration Behavior Models	48 days	Tue 10/6/20	Mon 11/23/20
46	<b>Parts Lists</b>	<b>48 days</b>	<b>Tue 10/6/20</b>	<b>Mon 11/23/20</b>
47	Parts list(s) for Schematics	48 days	Tue 10/6/20	Mon 11/23/20
48	Materials Budget list	48 days	Tue 10/6/20	Mon 11/23/20
49	<b>Proposed Implementation Gantt Chart</b>	48 days	Tue 10/6/20	Mon 11/23/20
50	<b>Conclusions and Recommendations</b>	48 days	Tue 10/6/20	Mon 11/23/20
51	Final Parts Request Form	13 days	Sun 10/11/20	Sat 10/24/20
52	Subsystems Demonstrations Day 1	0 days	Tue 11/10/20	Tue 11/10/20
53	Subsystems Demonstrations Day 2	0 days	Tue 11/17/20	Tue 11/17/20
54	Parts Request Form for Spring Semester	9 days	Mon 11/23/20	Wed 12/2/20

### Final Design Gantt Chart

This section shows the progress from the mid-point of the semester onwards.

Team members worked on the subsystems to prepare for demonstrations at this point.










The engineering hardware and software work split between the two groups, with a member of the Electrical and Computer teams working on both. Subsystem assignment dates were met and demonstrations completed within the timeframe given. -NK

ID	Task Mode	Task Name	Duration	Start	Finish	% Complete
1		SDP I 2020				0%
2		<b>Project Design</b>	<b>93 days?</b>	<b>Mon 8/24/20</b>	<b>Wed 11/25/20</b>	<b>94%</b>
3		<b>Midterm Report</b>	40 days	Wed 8/26/20	Mon 10/5/20	100%
4		Cover page	40 days	Wed 8/26/20	Mon 10/5/20	100%
5		T of C, L of T, L of F	40 days	Wed 8/26/20	Mon 10/5/20	100%
6		<b>Problem Statement</b>	<b>40 days</b>	<b>Wed 8/26/20</b>	<b>Mon 10/5/20</b>	<b>100%</b>
7		Need	40 days	Wed 8/26/20	Mon 10/5/20	100%
8		Objective	40 days	Wed 8/26/20	Mon 10/5/20	100%
9		Background	40 days	Wed 8/26/20	Mon 10/5/20	100%
10		Marketing Requirements	40 days	Wed 8/26/20	Mon 10/5/20	100%
11		Engineering Requirements Specification	40 days	Wed 8/26/20	Mon 10/5/20	100%
12		<b>Engineering Analysis</b>	<b>40.38 days</b>	<b>Wed 8/26/20</b>	<b>Mon 10/5/20</b>	<b>100%</b>
13		Circuits (DC, AC, Power, ...)	40.38 days	Wed 8/26/20	Mon 10/5/20	100%
14		Electronics (analog and digital)	40.38 days	Wed 8/26/20	Mon 10/5/20	100%
15		Signal Processing	40 days	Wed 8/26/20	Mon 10/5/20	0%
16		Communications (analog and digital)	40 days	Wed 8/26/20	Mon 10/5/20	0%
17		Electromechanics	40 days	Wed 8/26/20	Mon 10/5/20	0%
18		Computer Networks	40 days	Wed 8/26/20	Mon 10/5/20	0%
19		Working Subsystems	40.38 days	Wed 8/26/20	Mon 10/5/20	100%
20		<b>Accepted Technical Design</b>	<b>40.38 days?</b>	<b>Wed 8/26/20</b>	<b>Mon 10/5/20</b>	<b>100%</b>
21		<b>Hardware Design: Phase 1</b>	<b>32.38 days</b>	<b>Wed 8/26/20</b>	<b>Sun 9/27/20</b>	<b>100%</b>
22		Hardware Block Diagrams Levels 0 thru N (w/ FR tab	32.38 days	Wed 8/26/20	Sun 9/27/20	100%
23		<b>Software Design: Phase 1</b>	<b>32.38 days</b>	<b>Wed 8/26/20</b>	<b>Sun 9/27/20</b>	<b>100%</b>
24		Software Behavior Models Levels 0 thru N (w/FR tab	32.38 days	Wed 8/26/20	Sun 9/27/20	100%
25		<b>Mechanical Sketch</b>	40 days	Wed 8/26/20	Mon 10/5/20	100%
26		<b>Team information</b>	40 days	Wed 8/26/20	Mon 10/5/20	100%
27		<b>Project Schedules</b>	40 days	Mon 8/24/20	Sat 10/3/20	100%
28		<b>Midterm Design Presentations Day 2</b>	<b>7.38 days</b>	<b>Wed 9/30/20</b>	<b>Wed 10/7/20</b>	<b>100%</b>
29						
30		Midterm Design Gantt Chart	40 days	Wed 8/26/20	Mon 10/5/20	100%
31		<b>References</b>	40 days	Wed 8/26/20	Mon 10/5/20	100%

Work Stage	Team Member
Complete	Everyone
Complete	Everyone
Complete	Brian
<b>Complete</b>	
Complete	Nate
Complete	Hunter, Nick
Complete	Everone
Complete	Nate
Complete	Everyone
<b>Final Calculations</b>	
Working Subsystems	Brian, Nate
Working Subsystems	Everyone
Preliminary Calculations	Nick, Hunter
	Everyone
	Nick, Hunter
Final Draft	Hunter,Nate
Complete	Everyone
Fall Semester Schedule Complete	Nate
	Nate
Background Refrences / Datasheets	GEveryone

ID	Task Mode	Task Name	Duration	Start	Finish	% Complete
32		<b>Midterm Parts Request Form</b>	47 days	Wed 8/26/20	Mon 10/12/20	0%
33		<b>Midterm Design Presentations Day 1</b>	7.38 days	Wed 9/23/20	Wed 9/30/20	100%
34						
35		<b>Project Poster</b>	14 days	Wed 10/21/20	Wed 11/4/20	30%
36		<b>Final Design Report</b>	50 days	Tue 10/6/20	Wed 11/25/20	100%
37		<b>Abstract</b>	50.38 days	Tue 10/6/20	Wed 11/25/20	100%
38		<b>Hardware Design: Phase 2</b>	<b>41.38 days</b>	<b>Tue 10/6/20</b>	<b>Mon 11/16/20</b>	<b>100%</b>
39		<b>Modules 1...n</b>	<b>41.38 days</b>	<b>Tue 10/6/20</b>	<b>Mon 11/16/20</b>	<b>100%</b>
40		Simulations	41.38 days	Tue 10/6/20	Mon 11/16/20	100%
41		Hardware P-spice Design: Charging Circuit	41.38 days	Tue 10/6/20	Mon 11/16/20	100%
42		Hardware P-spice Design: Converter Circuit	41.38 days	Tue 10/6/20	Mon 11/16/20	100%
43		Hardware Eagle Cad Design: Charging Circuit	41.38 days	Tue 10/6/20	Mon 11/16/20	100%
44		Hardware Eagle Cad Design: Converter Circuit	41.38 days	Tue 10/6/20	Mon 11/16/20	100%
45		Hardware Eagle Cad Design: LED	41.38 days	Tue 10/6/20	Mon 11/16/20	100%
46		Hardware Eagle Cad Design: Processor / SD Interface	41.38 days	Tue 10/6/20	Mon 11/16/20	100%
47		Hardware Eagle Cad Design: Time of Flight	41.38 days	Tue 10/6/20	Mon 11/16/20	100%
48		Hardware Eagle Cad Design: Accelerometer	41.38 days	Tue 10/6/20	Mon 11/16/20	100%
49		Schematics	41.38 days	Tue 10/6/20	Mon 11/16/20	100%
50		<b>Software Design: Phase 2</b>	<b>41.38 days?</b>	<b>Tue 10/6/20</b>	<b>Mon 11/16/20</b>	<b>100%</b>
51		<b>Modules 1...n</b>	<b>41.38 days?</b>	<b>Tue 10/6/20</b>	<b>Mon 11/16/20</b>	<b>100%</b>
52		Code (working subsystems)	41.38 days	Tue 10/6/20	Mon 11/16/20	100%
53		Subsystem Software Code: LEDs	1 day	Sun 11/15/20	Mon 11/16/20	100%
54		Subsystem Software Code: Time of Flight Sensors	1 day	Sun 11/15/20	Mon 11/16/20	100%
55		Subsystem Software Code: Accelerometer	1 day	Sun 11/15/20	Mon 11/16/20	100%
56		Subsystem Software Code: Processor	1 day	Sun 11/15/20	Mon 11/16/20	100%
57		Subsystem Software Code: SD Card	1 day	Sun 11/15/20	Mon 11/16/20	100%
58		Subsystem Software Code:	1 day	Sun 11/15/20	Mon 11/16/20	100%
59		Board Schematic for Final Design	71.38 days	Tue 9/15/20	Wed 11/25/20	100%
60		<b>Parts Lists</b>	<b>50.38 days</b>	<b>Tue 10/6/20</b>	<b>Wed 11/25/20</b>	<b>100%</b>
61		Parts list(s) for Schematics	48 days	Tue 10/6/20	Mon 11/23/20	100%
62		Materials Budget list	50.38 days	Tue 10/6/20	Wed 11/25/20	100%

Work Stage	Team Member
Team 4 Presented	
Schematics and Sketeches on paper	Nate, Nick
	Everyone
Power Circuit spice simulation	Brian,Hunter
Working Spice Circuit	Brian
Working Spice Circuit	Brian
Eagle Schematic Completed	Nate
Eagle Schematic Completed	Nate
Eagle Schematic Completed	Hunter
Eagle Schematic Completed	Hunter
Eagle Schematic Completed	Hunter, Nate
Eagle Schematic Completed	Hunter, Nate
Power Circuit Schematic Designed	Brian
Preliminary Subsystem Code	Hunter
	Hunter
	Hunter,Nate
	Nick, Hunter
	Nick, Hunter
	Hunter,Nate
Attached to Final Report	Nick, Hunter
Attached to Final Report	Nick, Hunter

ID	Task Mode	Task Name	Duration	Start	Finish	% Complete
63	 	<b>Proposed Implementation Gantt Chart</b>	50.38 days	Tue 10/6/20	Wed 11/25/20	0%
64	 	<b>Conclusions and Recommendations</b>	50.38 days	Tue 10/6/20	Wed 11/25/20	100%
65	 	Final Parts Request Form	13.38 days	Sun 10/11/20	Sat 10/24/20	100%
66	 	Subsystems Demonstrations Day 1	6.38 days	Tue 11/10/20	Mon 11/16/20	100%
67		Parts Request Form for Spring Semester	9 days	Mon 11/23/20	Wed 12/2/20	50%

Work Stage	Team Member
Not completed till next semester	Nate
Attached to Final Report	Nate
	Nick
Presented 11/16/20	Everyone
Finalizing parts list	Everyone

## Initial (Spring) Project Gantt Chart

ID	Task Mode	Task Name	Duration	Start	Finish	Pr
1		<b>SDP2 Implementation 2020</b>	<b>103 days</b>	<b>Mon 1/11/21</b>	<b>Fri 4/23/21</b>	
2		Revise Gantt Chart	14 days	Mon 1/11/21	Sun 1/24/21	
3		<b>Implement Project Design</b>	<b>90 days</b>	<b>Mon 1/11/21</b>	<b>Sat 4/10/21</b>	
4		<b>Hardware Implementation</b>	<b>48 days</b>	<b>Mon 1/11/21</b>	<b>Sat 2/27/21</b>	
5		<b>Breadboard Components</b>	<b>7 days</b>	<b>Mon 1/11/21</b>	<b>Sun 1/17/21</b>	
6		Breadboard RGB LED circuit	7 days	Mon 1/11/21	Sun 1/17/21	
7		Set up buck-boost converter and demonstrate proper regulation of voltage	7 days	Mon 1/11/21	Sun 1/17/21	
8		<b>Layout and Generate PCB(s)</b>	<b>7 days</b>	<b>Mon 1/11/21</b>	<b>Sun 1/17/21</b>	
9		Route and order Main Board	14 days	Mon 1/11/21	Sun 1/24/21	
10		Route and order Time of Flight Sensor Boards	14 days	Mon 1/11/21	Sun 1/24/21	
11		Route and order RGB LED Boards	14 days	Mon 1/11/21	Sun 1/24/21	
12		<b>Assemble Hardware</b>	<b>35 days</b>	<b>Mon 1/11/21</b>	<b>Sun 2/14/21</b>	
13		Populate Main Board Circuit components: SD card slot, Accelerometer, I2C Multiplexers	14 days	Mon 2/1/21	Sun 2/14/21	
14		Populate Main Board: Power Circuit, Charging Circuit, and Buck-boost Converter	35 days	Mon 1/11/21	Sun 2/14/21	
15		Populate Time of Flight Sensor Board	14 days	Mon 2/1/21	Sun 2/14/21	
16		Populate RGB LED Board	14 days	Mon 2/1/21	Sun 2/14/21	
17		Prepare wiring for subsystem integration	35 days	Mon 1/11/21	Sun 2/14/21	
18		Prototype housing unit assemblies for boards on helmet	35 days	Mon 1/11/21	Sun 2/14/21	
19		<b>Test Hardware</b>	<b>42 days</b>	<b>Mon 1/11/21</b>	<b>Sun 2/21/21</b>	
20		Verify the battery is charging within specifications	14 days	Fri 2/5/21	Thu 2/18/21	
21		Make sure buck-boost converter is outputting 3.3V given the battery's nominal voltage input	14 days	Fri 2/5/21	Thu 2/18/21	
22		RGB LED Board: Verify that onboard capacitors are reducing noise (unintended flickering does not occur)	39 days	Mon 1/11/21	Thu 2/18/21	
23		Verify accelerometer readings with orientation diagram in datasheet	39 days	Mon 1/11/21	Thu 2/18/21	
24		Main Board: Check that I2C Multiplexers can be addressed as defined by hardware address jumpers	39 days	Mon 1/11/21	Thu 2/18/21	
25		ToF Board: Check that Time of Flight sensors communications are established with main board	7 days	Fri 2/12/21	Thu 2/18/21	
26		<b>Revise Hardware</b>	<b>39 days</b>	<b>Mon 1/18/21</b>	<b>Thu 2/25/21</b>	

Page 1

ID	Task Mode	Task Name	Duration	Start	Finish	Pr
27		ToF Boards: Adjust area of coverage as needed by adjusting positioning of sensors	7 days	Fri 2/19/21	Thu 2/25/21	25
28		Main Board: Battery/Charging: Adjust battery capacity dependent on initial test results for operating hours	39 days	Mon 1/18/21	Thu 2/25/21	25
29		Main Board: SD Card: Determine if level shifter is needed (dependent on buck-boost output)	7 days	Fri 2/19/21	Thu 2/25/21	25
30		<b>MIDTERM: Demonstrate Hardware Subsystems</b>	<b>5 days</b>	<b>Thu 2/18/21</b>	<b>Mon 2/22/21</b>	
31		SDC & FA Hardware Approval	0 days	Sat 2/27/21	Sat 2/27/21	30
32		<b>Software Implementation</b>	<b>49 days</b>	<b>Mon 1/11/21</b>	<b>Sun 2/28/21</b>	
33		<b>Develop Software</b>	<b>35 days</b>	<b>Mon 1/11/21</b>	<b>Sun 2/14/21</b>	
34		Create function to acquire distance readings from Time-of-Flight sensors	28 days	Mon 1/18/21	Sun 2/14/21	
35		Trigger interrupts when ToF sensor detects object within range	28 days	Mon 1/18/21	Sun 2/14/21	
36		Create function to acquire accelerometer axis readings	28 days	Mon 1/18/21	Sun 2/14/21	
37		Configure accelerometer to send interrupt signal when significant impact is detected	28 days	Mon 1/18/21	Sun 2/14/21	
38		Create function to output specific RGB value to LED	35 days	Mon 1/11/21	Sun 2/14/21	
39		Create function to initialize real-time-clock	35 days	Mon 1/11/21	Sun 2/14/21	
40		Create function to format accelerometer data to a C-string for output to SD card	35 days	Mon 1/11/21	Sun 2/14/21	
41		Create function to format real-time clock data to a C-string for output to SD card	35 days	Mon 1/11/21	Sun 2/14/21	
42		<b>Test Software</b>	<b>39 days</b>	<b>Mon 1/11/21</b>	<b>Thu 2/18/21</b>	
43		Verify ToF distance reading is accurate	28 days	Fri 1/22/21	Thu 2/18/21	
44		Ensure ToF interrupt lines are being driven within the appropriate range	28 days	Fri 1/22/21	Thu 2/18/21	
45		Check that accelerometer readings are within reason	28 days	Fri 1/22/21	Thu 2/18/21	
46		Ensure accelerometer drives interrupt line when the specified threshold is reached	28 days	Fri 1/22/21	Thu 2/18/21	
47		Verify that the appropriate colors are produced by LEDs from the given RGB values	28 days	Fri 1/22/21	Thu 2/18/21	
48		Successfully open .csv file from SD card on a desktop computer	28 days	Fri 1/22/21	Thu 2/18/21	
49		<b>Revise Software</b>	<b>49 days</b>	<b>Mon 1/11/21</b>	<b>Sun 2/28/21</b>	
50		Acquire ToF readings dependent on the interrupt line triggered	14 days	Mon 2/15/21	Sun 2/28/21	33
51		Acquire accelerometer values when interrupt is triggered	14 days	Mon 2/15/21	Sun 2/28/21	33
52		Create full .csv file from concatenating C-strings	21 days	Mon 2/8/21	Sun 2/28/21	33
53		Write .csv C-string to SD card	21 days	Mon 2/8/21	Sun 2/28/21	33

Page 2



56		<b>MIDTERM: Demonstrate Software Subsystems</b>	5 days	Tue 2/23/21	Sat 2/27/21
57		SDC & FA Software Approval	0 days	Sat 2/27/21	Sat 2/27/21 56
58		<b>System Integration</b>	<b>43 days</b>	<b>Sat 2/27/21</b>	<b>Sat 4/10/21</b>
59		<b>Assemble Complete System Integration</b>	<b>15 days</b>	<b>Sat 2/27/21</b>	<b>Sat 3/13/21 56</b>
60		Connect ToF boards to their respective I2C Multiplexer interface header	14 days	Sun 2/28/21	Sat 3/13/21 56
61		Connect RGB LEDs in series to the RGB LED header	14 days	Sun 2/28/21	Sat 3/13/21 56
62		Provide power to the entire system from the battery	14 days	Sun 2/28/21	Sat 3/13/21 56
63		Attach ToF Boards to helmet at appropriate angles	14 days	Sat 2/27/21	Fri 3/12/21
64		<b>Test Complete System Integration</b>	<b>8 days</b>	<b>Sat 3/13/21</b>	<b>Sat 3/20/21 59</b>
65		Ensure ToF boards cover full field of view	7 days	Sun 3/14/21	Sat 3/20/21 59
66		Ensure appropriate RGB LEDs light up corresponding to the ToF sensor detecting an object	7 days	Sun 3/14/21	Sat 3/20/21 59
67		Ensure Accelerometer event causes full .csv file report to be written to SD card	7 days	Sun 3/14/21	Sat 3/20/21 59
68		Ensure battery charges at appropriate rate when connected to circuit	7 days	Sat 3/13/21	Fri 3/19/21 59
69		Ensure ToF sensors detect objects up to 1.5m	7 days	Sun 3/14/21	Sat 3/20/21 59
70		<b>Revise Complete System Integration</b>	<b>17 days</b>	<b>Sat 3/20/21</b>	<b>Mon 4/5/21 64</b>
71		Adjust ToF sensors to eliminate blind spots in field of view	16 days	Sun 3/21/21	Mon 4/5/21 64
72		Ensure RGB LEDs react to ToF object detection in a timely manner	16 days	Sun 3/21/21	Mon 4/5/21 64
73		Ensure .csv logged to SD card after Accelerometer event can be read on a desktop PC	16 days	Sun 3/21/21	Mon 4/5/21 64
74		Ensure battery lasts a full 8 hours with the system running and detecting objects	16 days	Sat 3/20/21	Sun 4/4/21 64
75		<b>Demonstration of Complete System</b>	5 days	Tue 4/6/21	Sat 4/10/21 70
76		<b>Develop Final Report</b>	<b>103 days</b>	<b>Mon 1/11/21</b>	<b>Fri 4/23/21</b>
77		Write Final Report	103 days	Mon 1/11/21	Fri 4/23/21
78		Submit Final Report	0 days	Fri 4/23/21	Fri 4/23/21 77
79		Spring Recess	7 days	Mon 4/12/21	Sun 4/18/21

Page 3

Resource Names	% Work Complete	Resource Names	% Work Complete	Resource Names	% Work Complete
	3%	All Team Members	0%	Nick W.	0%
	0%	Brian T. / Nate K.	0%		
	4%			Hunter H.	0%
	10%	Hunter H. / Nick W.	0%		0%
	0%		0%		0%
Nate K. / Brian T.	0%		0%		0%
Nate K. / Brian T.	0%		0%		0%
	100%		0%		0%
Hunter H. / Brian T.	100%	Hunter H.	0%	Nate K.	0%
Nate K. / Brian T.	100%	Nick W.	0%	Nate K.	0%
Nick W.	100%	Nick W.	0%	Brian T.	0%
	0%	Nate K.	0%	Nate K.	0%
Hunter H. / Nick W.	0%	Nate K.	0%		0%
		Nate K.	0%	Nick W.	0%
Nate K. / Brian T.	0%	Nick W.	0%	Hunter H.	0%
Nate K. / Brian T.	0%	Nick W.	0%		
Nick W.	0%	Nick W.	0%	Nick W.	0%
Brian T. / Nate K.	0%	Nick W.	0%	Brian T. / Nate K.	0%
Brian T. / Nate K.	0%		0%	Nick W.	0%
	0%	Hunter H.	0%		0%
Nate K. / Brian T.	0%	Nick W.	0%	Brian T.	0%
Nate K. / Brian T.	0%	Hunter H.	0%	Hunter H.	0%
		Nick W.	0%	Nate K.	0%
Nick W.	0%	Nate K.	0%		
Hunter H.	0%	Nate K.	0%	Brian T.	0%
Hunter H.	0%		0%		0%
		Nick W.	0%		0%
Hunter H. / Nate K.	0%	Nick W.	0%	Nate K. / Hunter H.	0%
		Hunter H.	0%	Brian T.	0%
	0%	Hunter H.	0%		0%

The charts above show the initial Gantt charts from the beginning of the semester. Though the outline and subsystem division remained relatively similar, the actual Gantt charts depicted below show the difference in scheduling and division in work.

The reasoning for the change in schedules for the Gantt chart were based upon the difficulties in implementing subsystems for the midterm presentation. Due to the inability to demonstrate the Accelerometer, LEDs, full range of time-of-flight sensors, the latter half of the workload was changed as follows:

### Spring Final Adjustments/ System Integration Design Gantt:

30		<b>System Integration</b>	<b>43 days</b>	<b>Sat 2/27/21</b>	<b>Sat 4/10/21</b>	
31		<b>Software Implementation</b>	<b>88 days</b>	<b>Mon 1/11/21</b>	<b>Thu 4/8/21</b>	
32		<b>Develop Software</b>	<b>81 days</b>	<b>Mon 1/11/21</b>	<b>Thu 4/1/21</b>	
33		Create function to acquire distance readings from Time-of-Flight sensors	28 days	Fri 3/5/21	Thu 4/1/21	
34		Trigger interrupts when ToF sensor detects object within range	28 days	Fri 3/5/21	Thu 4/1/21	
35		Create function to acquire accelerometer axis readings	28 days	Fri 3/5/21	Thu 4/1/21	
36		Configure accelerometer to send interrupt signal when significant impact is detected	28 days	Fri 3/5/21	Thu 4/1/21	
37		Create function to output specific RGB value to LED	81 days	Mon 1/11/21	Thu 4/1/21	
38		Create function to initialize real-time-clock	81 days	Mon 1/11/21	Thu 4/1/21	
39		Create function to format accelerometer data to a C-string for output to SD card	81 days	Mon 1/11/21	Thu 4/1/21	
40		Create function to format real-time clock data to a C-string for output to SD card	81 days	Mon 1/11/21	Thu 4/1/21	
41		<b>Test Software</b>	<b>85 days</b>	<b>Mon 1/11/21</b>	<b>Mon 4/5/21</b>	
42		Verify ToF distance reading is accurate	28 days	Tue 3/9/21	Mon 4/5/21	
43		Ensure ToF interrupt lines are being driven within the appropriate range	28 days	Tue 3/9/21	Mon 4/5/21	
44		Check that accelerometer readings are within reason	28 days	Tue 3/9/21	Mon 4/5/21	
45		Ensure accelerometer drives interrupt line when the specified threshold is reached	28 days	Tue 3/9/21	Mon 4/5/21	
46		Verify that the appropriate colors are produced by LEDs from the given RGB value	28 days	Tue 3/9/21	Mon 4/5/21	
47		Successfully open .csv file from SD card on a desktop computer	28 days	Tue 3/9/21	Mon 4/5/21	
48		<b>Revise Software</b>	<b>85 days</b>	<b>Mon 1/11/21</b>	<b>Mon 4/5/21</b>	<b>32</b>
49		Acquire ToF readings dependent on the interrupt line triggered	14 days	Fri 4/2/21	Thu 4/15/21	32
50		Acquire accelerometer values when interrupt is triggered	14 days	Fri 4/2/21	Thu 4/15/21	32
51		Create full .csv file from concatenating C-strings	57 days	Mon 2/8/21	Mon 4/5/21	32
52		Write .csv C-string to SD card	57 days	Mon 2/8/21	Mon 4/5/21	32

Page 2

ID	Task Mode	Task Name	Duration	Start	Finish	Pr
53		Create function to determine the appropriate RGB value dependent on distance reading	85 days	Mon 1/11/21	Mon 4/5/21	
54		Create function to drive a specific LED dependent on the ToF sensor detecting an object	85 days	Mon 1/11/21	Mon 4/5/21	
55		SDC & FA Software Approval	0 days	Sat 2/27/21	Sat 2/27/21	
56						
57		<b>Assemble Complete System Integration</b>	<b>85 days</b>	<b>Mon 1/11/21</b>	<b>Mon 4/5/21</b>	
58		Connect ToF boards to their respective I2C Multiplexer interface header	14 days	Tue 3/23/21	Mon 4/5/21	
59		Connect RGB LEDs in series to the RGB LED header	14 days	Tue 3/23/21	Mon 4/5/21	
60		Provide power to the entire system from the battery	14 days	Tue 3/23/21	Mon 4/5/21	
61		Attach ToF boards to helmet at appropriate angles	38 days	Sat 2/27/21	Mon 4/5/21	
62		<b>Test Complete System Integration</b>	<b>24 days</b>	<b>Sat 3/13/21</b>	<b>Mon 4/5/21</b>	<b>57</b>
63		Ensure ToF boards cover full field of view	21 days	Sat 3/13/21	Fri 4/2/21	
64		Ensure appropriate RGB LEDs light up corresponding to the ToF sensor detecting an object	9 days	Thu 3/25/21	Fri 4/2/21	
65		Ensure Accelerometer event causes full .csv file report to be written to SD card	24 days	Sat 3/13/21	Mon 4/5/21	
66		Ensure battery charges at appropriate rate when connected to circuit	24 days	Sat 3/13/21	Mon 4/5/21	57
67		Ensure ToF sensors detect objects up to 1.5m	24 days	Sat 3/13/21	Mon 4/5/21	
68		<b>Revise Complete System Integration</b>	<b>17 days</b>	<b>Sat 3/20/21</b>	<b>Mon 4/5/21</b>	<b>62</b>
69		Adjust ToF sensors to eliminate blind spots in field of view	5 days	Thu 4/1/21	Mon 4/5/21	
70		Ensure RGB LEDs react to ToF object detection in a timely manner	5 days	Thu 4/1/21	Mon 4/5/21	
71		Ensure .csv logged to SD card after Accelerometer event can be read on a desktop PC	5 days	Thu 4/1/21	Mon 4/5/21	
72		Ensure battery lasts a full 8 hours with the system running and detecting objects	24 days	Sat 3/20/21	Mon 4/12/21	62
73		<b>Demonstration of Complete System</b>	<b>5 days</b>	<b>Tue 4/6/21</b>	<b>Sat 4/10/21</b>	<b>68</b>
74		<b>Develop Final Report</b>	<b>103 days</b>	<b>Mon 1/11/21</b>	<b>Fri 4/23/21</b>	
75		Write Final Report	103 days	Mon 1/11/21	Fri 4/23/21	
76		Submit Final Report	0 days	Fri 4/23/21	Fri 4/23/21	75
77		Spring Recess	7 days	Mon 4/12/21	Sun 4/18/21	
78		<b>Project Demonstration and Presentation</b>				

Resource Names	% Work Complete	Resource Names	% Work Complete
All Team Members	0%	Nick W.	0%
Brian T. / Nate K.	0%	Hunter H.	0%
Hunter H. / Nick W.	0%		0%
	0%		0%
	0%		0%
	0%	Nate K.	0%
Hunter H.	0%	Nate K.	0%
Nick W.	0%	Brian T.	0%
Nick W.	0%	Nate K.	0%
Nick W.	0%		0%
Nate K./Hunter H.	0%	Nick W.	0%
Nick W.	0%	Hunter H.	0%
Nick W.	0%	Nick W.	0%
Nick W.	0%	Brian T. / Nate K.	0%
	0%	Nick W.	0%
	0%		0%
Hunter H.	0%	Brian T.	0%
Nick W.	0%	Hunter H.	0%
Hunter H.	0%	Nick W.	0%
Nick W.	0%		
Nate K.	0%	Brian T./Nate K.	0%
Nate K./Hunter H.	0%		0%
	0%		0%
Nick W.	0%	Nate K. / Hunter H.	0%
Nick W.	0%	Brian T.	0%
Hunter H.	0%		0%
Hunter H.	0%		0%

As mentioned previously, the subsystem design was not completed at the midterm goals as initially intended. The power subsystems among other systems worked but mainly the LEDs and accelerometer still needed adjustments before the final. Through reallocating the work from the Gantt chart to the figure above, the design goals were reached for the final presentation. Many of the goals not reached during the mid-term demonstration could be attributed to either poor connections for the ToF and LED system. For the accelerometer, the subsystem was reading but writing to the SD card was more difficult than anticipated. Still, with a joint effort from team members, the project ultimately got back on track and succeeded in meeting its requirements. -

NK

## **10.) Conclusions and Recommendations:**

Workplace injury is still of great consideration to most companies. The ability to track and mitigate these injuries is always a growing concern and call to improve. To assist with this need, the design team proposes the H.A.L.O. This is an affordable and simplistic design that has several benefits to other solutions created previously. Other models or designs focus on different aspects of work monitoring, such as employee location and productivity. Current designs also prioritize inclusive models that are built into the hard hat. Since OSHA standards require helmets are replaced after an incident, smart hard hats in collisions are not reusable, causing large amounts of waste.

Given these circumstances, the development of the H.A.L. O includes engineering and marketing requirements that offer a cost-effective solution. The H.A.L.O. design is replaceable and reusable, reducing waste and still offering the services of smart helmet devices. The requirements of battery life being 8 hours and rechargeable, time of flight sensors reading their designated distances, LEDs shifting colors, and accelerometers reading and writing collision data to a SD card have all been met. All of the housing, hardware, and software have been integrated, accomplishing the engineering and marketing requirements.

With the design implemented, there are some recommendations that can be made. If given more time, revisions to the housing unit and wire management would be important. Reducing the overall size of the H.A.L.O. the number of wires going to sensors would be beneficial. Redesign of the P.C.B. would allow for further reduction of the size as well. Future of the project could potentially see a gradient color shift instead of a hard shift a given intervals for the time of flight and LEDs. In future designs, better formatting and graphing of the

accelerometer data is recommended. Finally, with the subsystems working, reducing the cost of construction for a unit would also be beneficial in future designs.

The design of the H.A.L.O system ran into implementation issues mainly with the accelerometer and time-of-flight subsystems. Due to issues with hardware and coding, the communication and implementation of multiple time-of-flight sensors to the LED subsystem were not finished in time for the midterm presentation. This was also the case with the accelerometer. The time-of-flights were able to read but integration of both systems together was not completed until the final design demonstration. Causes of initial difficulties can be attributed in part to issues with the soldering of the surface mount components, PCBs needing redesigned, and code revision.

However, despite these issues, all the subsystems worked and were demonstrated for the final. The engineering requirements were met and the H.A.L.O. system assists with incident recording and mitigation as intended. The project success relied heavily on software and embedded systems work which caused the team dynamic to shift towards the computer engineering side of the project. This caused an imbalanced team dynamic that initially was unforeseen. The hardware and power systems saw less issues with implementation mainly due to the simplistic nature of their function.

As a recommendation, a better division of subsystems and better planning in regard to workload division would have benefitted the project. With a project that largely relied on embedded systems, small components on the PCBs, and programming, all students having a background in Embedded systems classes such as Embedded Systems Interfacing, and programming classes is recommended. For students that wish to work on projects that deal with

sensors and recording their inputs, such as time-of-flights and accelerometers, understanding of programming in C or C++ would be beneficial.

Another recommendation for all senior design students is to have backgrounds in soldering. Some of the components breaking due to mistakes with soldering slowed work on the project leaving gaps where little progress was made. One final recommendation to students for future students would be to order spares of components in order to avoid losing out on valuable work time.

This project taught all members the value of properly applying concepts and theory learned in academics in a realistic work environment. Dealing with deadlines, setbacks, and monetary restrictions provided an important test to the skills students have developed throughout their time in the college of engineering. Thanks to the assistance of their senior design professors and coordinator, students were able to complete their design and gain experience in the engineering process. -NK

## **11.) References:**

Occupational Injury and Illness Classification System 2.01 developed by the Bureau of Labor Statistics. Source: U.S. Bureau of Labor Statistics, U.S. Department of Labor, November 2019.

- 1.) *Commonly used statistics / occupational safety and health administration.* (17 December 2019). Retrieved March 31, 2020, from <https://www.osha.gov/data/commonstats>
- 2.) *Hard Hat with Additional Technical Features.*  
<https://patents.google.com/patent/US20140208487A1/en>. Accessed 29 Mar. 2020.
- 3.) Li, Larry. *Time-of-Flight Camera – An Introduction.* Technical White Paper, SLOA190B, Texas Instruments, May 2014, p. 10,  
<http://www.ti.com/lit/wp/sloa190b/sloa190b.pdf>.

- 4.) Mitchell, Joel. B., et al. "Temperature Measurement Inside Protective Headgear: Comparison With Core Temperatures and Indicators of Physiological Strain During Exercise in a Hot Environment." *Journal of Occupational & Environmental Hygiene*, vol. 12, no. 12, Dec. 2015, pp. 866–74. *EBSCOhost*, doi:10.1080/15459624.2015.1072631.
- 5.) O'Connor, Kathryn L., et al. "Head-Impact–Measurement Devices: A Systematic Review." *Journal of Athletic Training*, vol. 52, no. 3, Mar. 2017, pp. 206–27. PubMed Central, doi:10.4085/1062-6050.52.2.05.
- 6.) *Smart Helmet*. <https://patents.google.com/patent/US20170048496A1/en>. Accessed 29 Mar. 2020.
- 7.) US EPA, OAR. "Basic Information about Oil and Natural Gas Air Pollution Standards." U.S. EPA, 20 Sept. 2016, <https://www.epa.gov/controlling-air-pollution-oil-and-natural-gas-industry/basic-information-about-oil-and-natural-gas>.
- 8.) Veeramani Kandasamy, Dr T. (2016). Microcontroller and SD Card Based Standalone Data Logging System using SPI and I2C Protocols for Industrial Application. 5. 2208-2214. 10.5281/zenodo.3543657.
- 9.) Accelerometer Theory & Design. (2008). Retrieved April 3, 2020, from [https://shodhganga.inflibnet.ac.in/bitstream/10603/2272/8/08\\_chapter 2.pdf](https://shodhganga.inflibnet.ac.in/bitstream/10603/2272/8/08_chapter%202.pdf).
- 10.) Spinelle, Laurent, et al. "Review of Portable and Low-Cost Sensors for the Ambient Air Monitoring of Benzene and Other Volatile Organic Compounds." *Sensors (Basel, Switzerland)*, vol. 17, no. 7, June 2017. *PubMed Central*, doi:10.3390/s17071520.
- 11.) *Ambient Light Sensor (ALS) Applications in Portable Electronics*. Rohm Semiconductor. [https://www.rohm.com/documents/11308/12928/CNA09016\\_wp.pdf](https://www.rohm.com/documents/11308/12928/CNA09016_wp.pdf). Accessed 3 Apr. 2020.

## 12.) Appendices / Datasheets.

### Appendix A: System Code

```
/**
  Section: Included Files
*/
#include <xc.h>
#include <stdbool.h>
#include <stdlib.h>

#define FCY 4000000UL // clock frequency
#include <libpic30.h>

#include "mcc_generated_files/mcc.h"
//#include "mcc_generated_files/system.h"
#include "mcc_generated_files/fatfs/fatfs_demo.h"
#include "mcc_generated_files/pwm_module_features.h"
#include "I2C_Handler.h"
#include "Initialize.h"

// Address Definitions
#define MUX_0 0x70
#define MUX_1 0x71

// number of ToF sensors
#define NUM_TOF 6

// pairing of mux numbers and port numbers
typedef struct {
    uint8_t mux;
    uint8_t port;
} muxPort_t;

// store (mux, port) pairs in an array
muxPort_t ToF[NUM_TOF] = {
    {0x00, 0x00}, // Left Center --> MID LED
    {0x00, 0x01}, // Left --> LEFT LED
    {0x00, 0x02}, // Left Left --> LEFT LED
    {0x01, 0x00}, // Right Center --> MID LED
    {0x01, 0x01}, // Right --> RIGHT LED
    {0x01, 0x02} // Right Right --> RIGHT LED
};

// LEFT LED == 0, MID LED == 1, RIGHT LED == 2
int LEDs[NUM_TOF] = {
    1, 0, 0, 1, 2, 2
};

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*/
```



```

// These are utilizing the same I/O pins as the on-board LEDs
/*
* (0x01) LED1 <==> L/R RED
* (0x02) LED2 <==> L/R GRN
* (0x04) LED3 <==> L/R BLU
* (0x08) LED4 <==> R ENABLE (ENABLEs are active LOW)
* (0x10) LED5 <==> L ENABLE
* (0x20) LED6 <==> MID RED
* (0x40) LED7 <==> MID GRN
* (0x80) LED8 <==> MID BLU (MID LED has no ENABLE, is tied to GND)
*/
#define LR_RED 0x01
#define LR_GRN 0x02
#define LR_BLU 0x04
#define LR_YLW 0x03
#define LR_PRP 0x05
#define LR_ON 0x00 // both L and R on
#define R_ON 0x00
#define R_OFF 0x10
#define L_ON 0x00
#define L_OFF 0x08
#define MID_RED 0x20
#define MID_GRN 0x40
#define MID_BLU 0x80
#define MID_YLW 0x60
#define MID_PRP 0xA0

typedef enum { LED_L, LED_C, LED_R } LED_posn;

/* * * * * * * * * * * ToF Sensor Definitions * * * * * * * * * * */
# define MAX_DISTANCE 1500 // 1500mm = 1.5m, the maximum range of
interest

#define VL53L0X_I2CADDR 0x29
// Record the current time to check an upcoming timeout against
#define startTimeout() (timeout_start_ms = millis())
// Check if timeout is enabled (set to nonzero value) and has expired
#define checkTimeoutExpired() (io_timeout > 0 && ((uint16_t)millis() -
timeout_start_ms) > io_timeout)
// Decode VCSEL (vertical cavity surface emitting laser) pulse period
in PCLKs
// from register value
// based on VL53L0X_decode_vcsel_period()
#define decodeVcselPeriod(reg_val) (((reg_val) + 1) << 1)
// Encode VCSEL pulse period register value from period in PCLKs
// based on VL53L0X_encode_vcsel_period()
#define encodeVcselPeriod(period_pclks) (((period_pclks) >> 1) - 1)
// Calculate macro period in *nanoseconds* from VCSEL period in PCLKs
// based on VL53L0X_calc_macro_period_ps()
// PLL_period_ps = 1655; macro_period_vclks = 2304

```

```
#define calcMacroPeriod(vcSEL_period_pclks) (((uint32_t)2304 *
(vcSEL_period_pclks) * 1655) + 500) / 1000)
```

```
enum regAddr {
    SYSRANGE_START = 0x00,

    SYSTEM_THRESH_HIGH = 0x0C,
    SYSTEM_THRESH_LOW = 0x0E,

    SYSTEM_SEQUENCE_CONFIG = 0x01,
    SYSTEM_RANGE_CONFIG = 0x09,
    SYSTEM_INTERMEASUREMENT_PERIOD = 0x04,

    SYSTEM_INTERRUPT_CONFIG_GPIO = 0x0A,

    GPIO_HV_MUX_ACTIVE_HIGH = 0x84,

    SYSTEM_INTERRUPT_CLEAR = 0x0B,

    RESULT_INTERRUPT_STATUS = 0x13,
    RESULT_RANGE_STATUS = 0x14,

    RESULT_CORE_AMBIENT_WINDOW_EVENTS_RTN = 0xBC,
    RESULT_CORE_RANGING_TOTAL_EVENTS_RTN = 0xC0,
    RESULT_CORE_AMBIENT_WINDOW_EVENTS_REF = 0xD0,
    RESULT_CORE_RANGING_TOTAL_EVENTS_REF = 0xD4,
    RESULT_PEAK_SIGNAL_RATE_REF = 0xB6,

    ALGO_PART_TO_PART_RANGE_OFFSET_MM = 0x28,

    I2C_SLAVE_DEVICE_ADDRESS = 0x8A,

    MSRC_CONFIG_CONTROL = 0x60,

    PRE_RANGE_CONFIG_MIN_SNR = 0x27,
    PRE_RANGE_CONFIG_VALID_PHASE_LOW = 0x56,
    PRE_RANGE_CONFIG_VALID_PHASE_HIGH = 0x57,
    PRE_RANGE_MIN_COUNT_RATE_RTN_LIMIT = 0x64,

    FINAL_RANGE_CONFIG_MIN_SNR = 0x67,
    FINAL_RANGE_CONFIG_VALID_PHASE_LOW = 0x47,
    FINAL_RANGE_CONFIG_VALID_PHASE_HIGH = 0x48,
    FINAL_RANGE_CONFIG_MIN_COUNT_RATE_RTN_LIMIT = 0x44,

    PRE_RANGE_CONFIG_SIGMA_THRESH_HI = 0x61,
    PRE_RANGE_CONFIG_SIGMA_THRESH_LO = 0x62,

    PRE_RANGE_CONFIG_VCSEL_PERIOD = 0x50,
    PRE_RANGE_CONFIG_TIMEOUT_MACROP_HI = 0x51,
    PRE_RANGE_CONFIG_TIMEOUT_MACROP_LO = 0x52,

    SYSTEM_HISTOGRAM_BIN = 0x81,
```

```

HISTOGRAM_CONFIG_INITIAL_PHASE_SELECT      = 0x33,
HISTOGRAM_CONFIG_READOUT_CTRL              = 0x55,

FINAL_RANGE_CONFIG_VCSEL_PERIOD            = 0x70,
FINAL_RANGE_CONFIG_TIMEOUT_MACROP_HI      = 0x71,
FINAL_RANGE_CONFIG_TIMEOUT_MACROP_LO     = 0x72,
CROSSTALK_COMPENSATION_PEAK_RATE_MCPS    = 0x20,

MSRC_CONFIG_TIMEOUT_MACROP                = 0x46,

SOFT_RESET_GO2_SOFT_RESET_N              = 0xBF,
IDENTIFICATION_MODEL_ID                   = 0xC0,
IDENTIFICATION_REVISION_ID               = 0xC2,

OSC_CALIBRATE_VAL                        = 0xF8,

GLOBAL_CONFIG_VCSEL_WIDTH                 = 0x32,
GLOBAL_CONFIG_SPAD_ENABLES_REF_0         = 0xB0,
GLOBAL_CONFIG_SPAD_ENABLES_REF_1         = 0xB1,
GLOBAL_CONFIG_SPAD_ENABLES_REF_2         = 0xB2,
GLOBAL_CONFIG_SPAD_ENABLES_REF_3         = 0xB3,
GLOBAL_CONFIG_SPAD_ENABLES_REF_4         = 0xB4,
GLOBAL_CONFIG_SPAD_ENABLES_REF_5         = 0xB5,

GLOBAL_CONFIG_REF_EN_START_SELECT         = 0xB6,
DYNAMIC_SPAD_NUM_REQUESTED_REF_SPAD     = 0x4E,
DYNAMIC_SPAD_REF_EN_START_OFFSET         = 0x4F,
POWER_MANAGEMENT_GO1_POWER_FORCE         = 0x80,

VHV_CONFIG_PAD_SCL_SDA__EXTSUP_HV        = 0x89,

ALGO_PHASECAL_LIM                        = 0x30,
ALGO_PHASECAL_CONFIG_TIMEOUT             = 0x30,
};

typedef enum { VcSELPeriodPreRange, VcSELPeriodFinalRange }
vcSELPeriodType;

uint8_t last_status; // status of last I2C transmission

typedef struct {
    bool tcc, msrc, dss, pre_range, final_range;
    // TCC: Target CentreCheck
    // MSRC: Minimum Signal Rate Check
    // DSS: Dynamic Spad Selection
} SequenceStepEnables;

typedef struct {
    uint16_t pre_range_vcSEL_period_pclks,
    final_range_vcSEL_period_pclks;

    uint16_t msrc_dss_tcc_mclks, pre_range_mclks, final_range_mclks;

```

```

    uint32_t msrc_dss_tcc_us,    pre_range_us,    final_range_us;
} SequenceStepTimeouts;

uint8_t address;
uint16_t io_timeout;
bool did_timeout;
uint16_t timeout_start_ms;
uint8_t stop_variable; // read by init and used when starting
measurement; is StopVariable field of VL53L0X_DevData_t structure in
API
uint32_t measurement_timing_budget_us;

/* * * * * * * * * * * * * * I2C Bus Read/Write Functions * * * * * * * *
* * * */
I2C1_MESSAGE_STATUS I2C_Status = I2C1_MESSAGE_COMPLETE; // I2C Bus
Status

void selectPort(muxPort_t sensorPort);
void selectPort2(uint8_t mux, uint8_t port); // select mux port
uint8_t getMuxCtrlReg(uint8_t mux); // read interrupt values
from mux
uint8_t getInterrupts(uint8_t mux);
void writeRegister(uint8_t dev, uint8_t reg, uint8_t data);
void writeRegister_16b(uint8_t dev, uint8_t reg, uint16_t data);
void writeRegister_32b(uint8_t dev, uint8_t reg, uint32_t data);
uint8_t readReg(uint8_t dev, uint8_t reg);
void readRegister(uint8_t dev, uint8_t reg, uint8_t* data);
uint16_t readRegister_16b(uint8_t dev, uint8_t reg);

/* * * * * * * * * * * * * * ToF Helper Functions * * * * * * * * * *
* * * */
void initSingleToF(int ToF_num, uint16_t *dists);
void initAllToF(uint16_t *dists);
void initAllToF2(uint16_t *dists);
void getSingleToF(int ToF_num, uint16_t *dists);
void getAllToF(uint16_t *dists);
void getAllToF2(uint16_t *dists);
uint8_t getNearestObstacleIndex(uint16_t *dists);
uint8_t getNearestObstacleIndex2(uint16_t *dists);

/* * * * * * * * * * * * * * Accelerometer Definitions * * * * * * * * * *
*/
//<Nick>
#define H3LIS200DL_I2CADDR 0x19
#define H3LIS200DL_WHO_AM_I 0x0F
#define H3LIS200DL_CTRL_REG1 0x20
#define H3LIS200DL_CTRL_REG2 0x21
#define H3LIS200DL_CTRL_REG3 0x22
#define H3LIS200DL_CTRL_REG4 0x23

```

```

#define H3LIS200DL_CTRL_REG5 0x24
#define H3LIS200DL_HP_FILTER_RESET 0x25
#define H3LIS200DL_REFERENCE 0x26
#define H3LIS200DL_STATUS_REG 0x27
#define H3LIS200DL_OUT_X_H 0x29 // X Data
#define H3LIS200DL_OUT_X_L 0x28
#define H3LIS200DL_OUT_Y_H 0x2B // Y Data
#define H3LIS200DL_OUT_Y_L 0x2A
#define H3LIS200DL_OUT_Z_H 0x2D // Z Data
#define H3LIS200DL_OUT_Z_L 0x2C
#define H3LIS200DL_INT1_CFG 0x30 // Interrupt 1 (Pin 11,
used)
#define H3LIS200DL_INT1_SRC 0x31
#define H3LIS200DL_INT1_THS 0x32
#define H3LIS200DL_INT1_DURATION 0x33
#define H3LIS200DL_INT2_CFG 0x34 // Interrupt 2
#define H3LIS200DL_INT2_SRC 0x35
#define H3LIS200DL_INT2_THS 0x36
#define H3LIS200DL_INT2_DURATION 0x37
// POWER MODES
#define H3LIS200DL_PWR_DWN 0x00 // Power Down Mode
#define H3LIS200DL_NRML 0x01 // Normal Mode
#define H3LIS200DL_LP_0_5HZ 0x02 // Low Power 0.5Hz
#define H3LIS200DL_LP_1HZ 0x03 // Low Power 1.0Hz
#define H3LIS200DL_LP_2HZ 0x04 // Low Power 2.0Hz
#define H3LIS200DL_LP_5HZ 0x05 // Low Power 5.0Hz
#define H3LIS200DL_LP_10HZ 0x06 // Low Power 10.Hz
// OUTPUT DATA RATES
#define H3LIS200DL_DR_50HZ 0x00 // 50Hz
#define H3LIS200DL_DR_100HZ 0x01 // 100Hz
#define H3LIS200DL_DR_400HZ 0x02 // 400Hz
#define H3LIS200DL_DR_1000HZ 0x03 // 1000Hz

#define H3LIS200DL_EN_X 0x01 // Enable X Data
#define H3LIS200DL_EN_Y 0x02 // Enable Y Data
#define H3LIS200DL_EN_Z 0x04 // Enable Z Data
#define H3LIS200DL_EN_XYZ 0x07 // Enable X, Y, and Z Data

/* * * * * * * * * * * Accelerometer Function Definitions * * * * *
* * * * */
typedef enum {USE_I2C, USE_SPI} comm_mode;
typedef enum {POWER_DOWN, NORMAL, LOW_POWER_0_5HZ, LOW_POWER_1HZ,
LOW_POWER_2HZ, LOW_POWER_5HZ, LOW_POWER_10HZ}
power_mode;
typedef enum {DR_50HZ, DR_100HZ, DR_400HZ, DR_1000HZ} data_rate;
typedef enum {HPC_8, HPC_16, HPC_32, HPC_64}
high_pass_cutoff_freq_cfg;
typedef enum {PUSH_PULL, OPEN_DRAIN} pp_od;
typedef enum {INT_SRC, INT1_2_SRC, DRDY, BOOT} int_sig_src;
typedef enum {LOW_RANGE, MED_RANGE, NO_RANGE, HIGH_RANGE} fs_range;
typedef enum {X_AXIS, Y_AXIS, Z_AXIS} int_axis;

```

```

typedef enum {TRIG_ON_HIGH, TRIG_ON_LOW} trig_on_level;

void H3LIS200DL_begin();
void H3LIS200DL_axesEnable(bool enable);
void H3LIS200DL_setPowerMode(power_mode pmode);
void H3LIS200DL_setODR(data_rate drate);
void H3LIS200DL_readAxes(int16_t* x, int16_t* y, int16_t* z);
uint8_t H3LIS200DL_readReg(uint8_t reg_address);
int16_t H3LIS200DL_convertToG(int16_t maxScale, int16_t reading);
void H3LIS200DL_setHighPassCoeff(high_pass_cutoff_freq_cfg hpcoeff);
void H3LIS200DL_enableHPF(bool enable);
void H3LIS200DL_HPFOntPin(bool enable, uint8_t pin);
void H3LIS200DL_intActiveHigh(bool enable);
void H3LIS200DL_intPinMode(pp_od _pinMode);
void H3LIS200DL_latchInterrupt(bool enable, uint8_t intSource);
void H3LIS200DL_intSrcConfig(int_sig_src src, uint8_t pin);
void H3LIS200DL_setFullScale(fs_range range);
bool H3LIS200DL_newXData();
bool H3LIS200DL_newYData();
bool H3LIS200DL_newZData();
void H3LIS200DL_enableInterrupt(int_axis axis, trig_on_level
trigLevel, uint8_t interrupt, bool enable);
void H3LIS200DL_setIntDuration(uint8_t duration, uint8_t intSource);
void H3LIS200DL_setIntThreshold(uint8_t threshold, uint8_t intSource);
int16_t H3LIS200DL_Read_x(int16_t x);
int16_t H3LIS200DL_Read_y(int16_t y);
int16_t H3LIS200DL_Read_z(int16_t z);
//</Nick>
bool getAccelPoints(void);
bool getAccelPoints2(void);

/* * * * * * * * * * * SD Card Functions * * * * * * * * * * */
void writeTemplateToSD(void);
void writeAccelToSD(void);

/* * * * * * * * * * * Time-of-Flight Sensor Functions * * * * * * * *
* * */
uint8_t VL53L0X_init(void);
bool VL53L0X_config(void);
bool VL53L0X_setSignalRateLimit(float limit_Mcps);
bool VL53L0X_getSpadInfo(uint8_t * count, bool * type_is_aperture);
uint32_t VL53L0X_getMeasurementTimingBudget(void);
void VL53L0X_getSequenceStepTimeouts(SequenceStepEnables const *
enables, SequenceStepTimeouts * timeouts);
uint8_t VL53L0X_getVcSELPeriodType(vcSELPeriodType type);
bool VL53L0X_setMeasurementTimingBudget(uint32_t budget_us);
void VL53L0X_getSequenceStepEnables(SequenceStepEnables * enables);
uint16_t VL53L0X_encodeTimeout(uint16_t timeout_mclks);
bool VL53L0X_performSingleRefCalibration(uint8_t vhw_init_byte);

```

```

uint32_t VL53L0X_timeoutMclksToMicroseconds(uint16_t
timeout_period_mclks, uint8_t vtsel_period_pclks);
uint32_t VL53L0X_timeoutMicrosecondsToMclks(uint32_t
timeout_period_us, uint8_t vtsel_period_pclks);
uint16_t VL53L0X_decodeTimeout(uint16_t reg_val);
void VL53L0X_startContinuous(uint32_t period_ms);
void VL53L0X_stopContinuous(void);
uint16_t VL53L0X_readRangeContinuousMillimeters(void);
uint16_t VL53L0X_readRangeSingleMillimeters(void);
inline void VL53L0X_setTimeout(uint16_t timeout) { io_timeout =
timeout; }
inline uint16_t VL53L0X_getTimeout(void) { return io_timeout; }
bool VL53L0X_timeoutOccurred(void);

/* * * * * * * * * * * * * * * * LED Display Functions * * * * * * * * * * *
* * * */
void showBinary(uint8_t n);
void showStartup(void);
void showStartupRGB(void);
void showDistanceRGB(uint16_t dist, LED_posn LED);
void showInitRGB(int index);
void showConcussion(void);
void showCount(void);
void showError(void);

/* * * * * * * * * * * * * * * * Accelerometer Variables * * * * * * * * * * *
* * */
// <Nick>
int16_t x_1, y_1, z_1;
int16_t x_2, y_2, z_2;
int16_t x_3, y_3, z_3;
int16_t x_4, y_4, z_4;
int16_t x_5, y_5, z_5;
int16_t thresh = 50; // impact threshold (in Gs) // set to 50 for
actual use
int16_t max = 0x0000; // current maximum axis reading

// timestamps of readings
unsigned long timer1, timer2, timer3, timer4, timer5;

// c-strings for writing data to SD card
char data1[255];
char data2[255];
char data3[255];
char data4[255];
char data5[255];
// </Nick>

// set TRUE to automatically clear the interrupt of each ToF sensor
upon reading

```





```

    /***** I2C Time-of-Flight *****/
    getAllToF(distances); // get the distances from all ToF
sensors

    // get the distance from the ToF_to_test sensor
    // getSingleToF(ToF_to_test, distances);

    /***** SHOW READING *****/
    // dist_8 = distances[ToF_to_test] & 0xFF;
    // show reading LSBs via on-board LEDs
    // showBinary(dist_8);
    // msTimerDelay(10);
    // show RGB corresponding to reading of ToF_to_test
    // showDistanceRGB(distances[ToF_to_test], LEDs[ToF_to_test]);

    // find the sensor detecting the closest obstacle
    index = getNearestObstacleIndex(distances);
    // show RGB corresponding to reading of said sensor
    if(index == 0xFF) {
        showBinary(0x00);
    }
    else {
        showDistanceRGB(distances[index], LEDs[index]);
    }

}
return 0;
}

// displays the byte value in binary on the LEDs
void showBinary(uint8_t n) {
    LED1 = n;
    LED2 = n >> 1;
    LED3 = n >> 2;
    LED4 = n >> 3;
    LED5 = n >> 4;
    LED6 = n >> 5;
    LED7 = n >> 6;
    LED8 = n >> 7;
}

// a fun visual to do at boot
void showStartup(void) {
    uint16_t delay = 25;

    uint8_t display = 0x01;
    while(display != 0x80) { // run one LED "up"
        display = display << 1;
        showBinary(display);
        msTimerDelay(delay);
    }
}

```

```

while(display > 0x00) { // run one LED back "down"
    display = display >> 1;
    showBinary(display);
    msTimerDelay(delay);
}

display = 0x00; // reset to 0
while(display < 0xFF) { // "fill" LEDs
    display = display << 1;
    showBinary(++display);
    msTimerDelay(delay);
}
while(display > 0x00) { // "empty" LEDs
    display = display >> 1;
    showBinary(display);
    msTimerDelay(delay);
}

}

void showStartupRGB(void) {
    uint16_t delay = 200;

    showBinary(LR_RED | LR_ON | MID_RED);
    msTimerDelay(delay);

    showBinary(LR_PRP | LR_ON | MID_PRP);
    msTimerDelay(delay);

    showBinary(LR_BLU | LR_ON | MID_BLU);
    msTimerDelay(delay);

    showBinary(LR_BLU | LR_GRN | LR_ON | MID_BLU | MID_GRN);
    msTimerDelay(delay);

    showBinary(LR_GRN | LR_ON | MID_GRN);
    msTimerDelay(delay);
}

void showDistanceRGB(uint16_t dist, LED_posn LED) {
    if(1900 < dist) {
        showBinary(0x00);
    }
    else if(LED == LED_C) {
        if(dist <= 500) {
            showBinary(MID_RED);
        } else if(500 < dist && dist <= 1000) {
            showBinary(MID_GRN);
        } else if(1000 < dist && dist <= 1500) {
            showBinary(MID_BLU);
        }
    }
}

```

```

else if(LED == LED_L) {
    if(dist <= 500) {
        showBinary(LR_RED | R_OFF);
    } else if(500 < dist && dist <= 1000) {
        showBinary(LR_GRN | R_OFF);
    } else if(1000 < dist && dist <= 1500) {
        showBinary(LR_BLU | R_OFF);
    }
}
else if(LED == LED_R) {
    if(dist <= 500) {
        showBinary(LR_RED | L_OFF);
    } else if(500 < dist && dist <= 1000) {
        showBinary(LR_GRN | L_OFF);
    } else if(1000 < dist && dist <= 1500) {
        showBinary(LR_BLU | L_OFF);
    }
}
}

void showInitRGB(int index) {
    int clr = index % 3; // determines what color to turn the LED
    // 0 -> RED
    // 1 -> YELLOW
    // 2 -> WHITE

    if(index < (NUM_TOF/2)) { // left side sensors
        if(clr == 0) {
            showBinary(LR_RED | R_OFF); // red
        } else if(clr == 1) {
            showBinary(LR_RED | LR_GRN | R_OFF); // yellow
        } else if(clr == 2) {
            showBinary(LR_RED | LR_GRN | LR_BLU | R_OFF); // white
        }
    }
    else if(( (NUM_TOF/2) <= index ) && ( index < NUM_TOF )) { //
right side sensors
        if(clr == 0) {
            showBinary(LR_RED | L_OFF); // red
        } else if(clr == 1) {
            showBinary(LR_RED | LR_GRN | L_OFF); // yellow
        } else if(clr == 2) {
            showBinary(LR_RED | LR_GRN | LR_BLU | L_OFF); // white
        }
    }
}

void showConcussion(void) {
    uint16_t delay = 1000;

    // show all red
    showBinary(LR_RED | LR_ON | MID_RED);
}

```

```

msTimerDelay(delay);

// show all purple
showBinary(LR_PRP | LR_ON | MID_PRP);
msTimerDelay(delay);

// show all red
showBinary(LR_RED | LR_ON | MID_RED);
msTimerDelay(delay);

// show all purple
showBinary(LR_PRP | LR_ON | MID_PRP);
msTimerDelay(delay);

// show all red
showBinary(LR_RED | LR_ON | MID_RED);
msTimerDelay(delay);
}

// a visual for errors
void showError(void) {
    uint16_t delay = 50;

    uint8_t i = 0x00;
    while(i < 3) { // flash alternating LEDs 3 times
        showBinary(0x55);
        msTimerDelay(delay);
        showBinary(0xAA);
        msTimerDelay(delay);
        i++;
    }
}

/* * * * * * * * * * * * * * I2C Bus Read/Write Functions * * * * * * * *
* * * */

void selectPort(muxPort_t sensorPort) {
    if(sensorPort.mux == 0x00) {
        writeRegister(MUX_1, 0x00, 0x00); // disable mux 1
    }
    else if(sensorPort.mux == 0x01) {
        writeRegister(MUX_0, 0x00, 0x00); // disable mux 0
    }
    // base mux address is 0x70
    writeRegister((MUX_0 + sensorPort.mux), 0x00, (0x04 +
sensorPort.port));
}

void selectPort2(uint8_t mux, uint8_t port) {
    // base mux address is 0x70
    if(mux == 0x00) {
        writeRegister(MUX_1, 0x00, 0x00);
    }
}

```

```

        writeRegister(MUX_0, 0x00, (0x04 + port));
    }
    else if(mux == 0x01) {
        writeRegister(MUX_0, 0x00, 0x00);
        writeRegister(MUX_1, 0x00, (0x04 + port));
    }
}

// read the interrupts from mux
uint8_t getMuxCtrlReg(uint8_t mux) {
    // read the control register of mux
    return readReg((MUX_0 + mux), 0x00);
}

// writes to the device a byte of data to the register
void writeRegister(uint8_t dev, uint8_t reg, uint8_t data) {
    uint8_t config[] = {reg, data};

    while(I2C_Status != I2C1_MESSAGE_COMPLETE);
    I2C1_MasterWrite(config, sizeof(config), dev, &I2C_Status);
    while(I2C_Status != I2C1_MESSAGE_COMPLETE);
    // showBinary(I2C_Status);
    msTimerDelay(5);
}

// writes to the device a byte of data to the register
void writeRegister_16b(uint8_t dev, uint8_t reg, uint16_t data) {
    // send MSB then LSB
    uint8_t config[] = {reg, data >> 8, data & 0x00FF};

    while(I2C_Status != I2C1_MESSAGE_COMPLETE);
    I2C1_MasterWrite(config, sizeof(config), dev, &I2C_Status);
    while(I2C_Status != I2C1_MESSAGE_COMPLETE);
    msTimerDelay(5);
}

void writeRegister_32b(uint8_t dev, uint8_t reg, uint32_t data) {
    // send MSB, ..., then LSB
    uint8_t config[] = {reg, data >> 24, data >> 16 & 0x00FF, data >>
8 & 0x00FF, data & 0x00FF};

    while(I2C_Status != I2C1_MESSAGE_COMPLETE);
    I2C1_MasterWrite(config, sizeof(config), dev, &I2C_Status);
    while(I2C_Status != I2C1_MESSAGE_COMPLETE);
    msTimerDelay(5);
}

// reads from the device a byte of data from the given register
void readRegister(uint8_t dev, uint8_t reg, uint8_t* data) {
    while(I2C_Status != I2C1_MESSAGE_COMPLETE);
    I2C1_MasterWrite(&reg, 1, dev, &I2C_Status);
}

```



```

    VL53L0X_init();
    dists[i] = VL53L0X_readRangeSingleMillimeters();
    if(!auto_int_clr) {
        writeRegister(VL53L0X_I2CADDR, SYSTEM_INTERRUPT_CLEAR,
0x01); // clear interrupt
    }
    showInitRGB(i); // show RGB LED change on each side as sensors
init
}

// turn all on white for 1 second to show initialization is done
showBinary(LR_RED | LR_GRN | LR_BLU | LR_ON | MID_RED | MID_GRN |
MID_BLU);
msTimerDelay(1000);
}

void initAllToF2(uint16_t *dists) {
    int i = 1;
    // selectPort(ToF[i]);
    // msTimerDelay(1);
    // VL53L0X_init();
    // dists[i] = VL53L0X_readRangeSingleMillimeters();
    // if(!auto_int_clr) {
    //     writeRegister(VL53L0X_I2CADDR, SYSTEM_INTERRUPT_CLEAR,
0x01); // clear interrupt
    // }

    i = 3;
    selectPort(ToF[i]);
    msTimerDelay(1);
    VL53L0X_init();
    dists[i] = VL53L0X_readRangeSingleMillimeters();
    if(!auto_int_clr) {
        writeRegister(VL53L0X_I2CADDR, SYSTEM_INTERRUPT_CLEAR, 0x01);
// clear interrupt
    }

    i = 4;
    selectPort(ToF[i]);
    msTimerDelay(1);
    VL53L0X_init();
    dists[i] = VL53L0X_readRangeSingleMillimeters();
    if(!auto_int_clr) {
        writeRegister(VL53L0X_I2CADDR, SYSTEM_INTERRUPT_CLEAR, 0x01);
// clear interrupt
    }

    i = 5;
    selectPort(ToF[i]);
    msTimerDelay(1);
    VL53L0X_init();
    dists[i] = VL53L0X_readRangeSingleMillimeters();

```

```

        if(!auto_int_clr) {
            writeRegister(VL53L0X_I2CADDR, SYSTEM_INTERRUPT_CLEAR, 0x01);
// clear interrupt
        }
    }

void getSingleToF(int ToF_num, uint16_t *dists) {
    selectPort(ToF[ToF_num]); // Select Multiplexer 0, Port 0
    msTimerDelay(1);
    dists[ToF_num] = VL53L0X_readRangeSingleMillimeters();
    msTimerDelay(1);
}

void getAllToF(uint16_t *dists) {
    int i = 0;
    for(i = 0; i < 6; i++) {
        selectPort(ToF[i]);
        msTimerDelay(1);
        dists[i] = VL53L0X_readRangeSingleMillimeters();
        msTimerDelay(1);
    }
}

void getAllToF2(uint16_t *dists) {
    int i = 1;
//    selectPort(ToF[i]); // Select Multiplexer 0, Port 1
//    msTimerDelay(1);
//    VL53L0X_startContinuous(0);
//    dists[i] = VL53L0X_readRangeSingleMillimeters();
//    msTimerDelay(1);

    i = 3;
    selectPort(ToF[i]); // Select Multiplexer 1, Port 0
    msTimerDelay(1);
    dists[i] = VL53L0X_readRangeSingleMillimeters();
    msTimerDelay(1);

    i = 4;
    selectPort(ToF[i]); // Select Multiplexer 1, Port 1
    msTimerDelay(1);
    dists[i] = VL53L0X_readRangeSingleMillimeters();
    msTimerDelay(1);

    i = 5;
    selectPort(ToF[i]); // Select Multiplexer 1, Port 2
    msTimerDelay(1);
    dists[i] = VL53L0X_readRangeSingleMillimeters();
    msTimerDelay(1);
}

// given the array of distances, return the index of the closest
object

```





```

/* * * * * * * * * * * * * * * * * ToF Sensor Functions * * * * * * * * * *
* * * */
uint8_t VL53L0X_init(void) {
    uint8_t success = 0;

    if(VL53L0X_config()) { // configure ToF
        success = 1;
    }

    VL53L0X_setTimeout(200); // was 500
    // Start continuous back-to-back mode (take readings as
    // fast as possible). To use continuous timed mode
    // instead, provide a desired inter-measurement period in
    // ms (e.g. sensor.startContinuous(100))

    VL53L0X_setMeasurementTimingBudget(200000); // was 200000
    //VL53L0X_startContinuous(0); // was 0
    VL53L0X_stopContinuous();

    return success;
}

bool VL53L0X_config(void) {
    // VL53L0X_DataInit() begin
    uint8_t io_2v8 = 0;

    // sensor uses 1V8 mode for I/O by default; switch to 2V8 mode if
    necessary
    if (io_2v8) {
        writeRegister(VL53L0X_I2CADDR,
VHV_CONFIG_PAD_SCL_SDA__EXTSUP_HV,
        readReg(VL53L0X_I2CADDR, VHV_CONFIG_PAD_SCL_SDA__EXTSUP_HV) |
0x01); // set bit 0
    }

    // "Set I2C standard mode"
    writeRegister(VL53L0X_I2CADDR, 0x88, 0x00);

    writeRegister(VL53L0X_I2CADDR, 0x80, 0x01);
    writeRegister(VL53L0X_I2CADDR, 0xFF, 0x01);
    writeRegister(VL53L0X_I2CADDR, 0x00, 0x00);

    stop_variable = readReg(VL53L0X_I2CADDR, 0x91);

    writeRegister(VL53L0X_I2CADDR, 0x00, 0x01);
    writeRegister(VL53L0X_I2CADDR, 0xFF, 0x00);
    writeRegister(VL53L0X_I2CADDR, 0x80, 0x00);

    // disable SIGNAL_RATE_MSRC (bit 1) and SIGNAL_RATE_PRE_RANGE (bit
4) limit checks
    writeRegister(VL53L0X_I2CADDR, MSRC_CONFIG_CONTROL,
readReg(VL53L0X_I2CADDR, MSRC_CONFIG_CONTROL) | 0x12);

```

```

    // set final range signal rate limit to 0.25 MCPS (million counts
per second)
VL53L0X_setSignalRateLimit(0.25);

writeRegister(VL53L0X_I2CADDR, SYSTEM_SEQUENCE_CONFIG, 0xFF);

// VL53L0X_DataInit() end

// VL53L0X_StaticInit() begin

uint8_t spad_count;
bool spad_type_is_aperture;
if (!VL53L0X_getSpadInfo(&spad_count, &spad_type_is_aperture)) {
return false; }

// The SPAD map (RefGoodSpadMap) is read by
VL53L0X_get_info_from_device() in
// the API, but the same data seems to be more easily readable
from
// GLOBAL_CONFIG_SPAD_ENABLES_REF_0 through _6, so read it from
there
uint8_t ref_spad_map[6];

VL53L0X_Read(GLOBAL_CONFIG_SPAD_ENABLES_REF_0, ref_spad_map, 6);

// -- VL53L0X_set_reference_spads() begin (assume NVM values are
valid)

writeRegister(VL53L0X_I2CADDR, 0xFF, 0x01);
writeRegister(VL53L0X_I2CADDR, DYNAMIC_SPAD_REF_EN_START_OFFSET,
0x00);
writeRegister(VL53L0X_I2CADDR,
DYNAMIC_SPAD_NUM_REQUESTED_REF_SPAD, 0x2C);
writeRegister(VL53L0X_I2CADDR, 0xFF, 0x00);
writeRegister(VL53L0X_I2CADDR, GLOBAL_CONFIG_REF_EN_START_SELECT,
0xB4);

uint8_t first_spad_to_enable = spad_type_is_aperture ? 12 : 0; //
12 is the first aperture spad
uint8_t spads_enabled = 0;

uint8_t i;
for (i = 0; i < 48; i++) {
    if (i < first_spad_to_enable || spads_enabled == spad_count) {
        // This bit is lower than the first one that should be
enabled, or
        // (reference_spad_count) bits have already been enabled, so
zero this bit
        ref_spad_map[i / 8] &= ~(1 << (i % 8));
    }
    else if ((ref_spad_map[i / 8] >> (i % 8)) & 0x1) {

```

```

        spads_enabled++;
    }
}

// TODO: ?
//writeMulti(GLOBAL_CONFIG_SPAD_ENABLES_REF_0, ref_spad_map, 6);
uint8_t ad[] = {GLOBAL_CONFIG_SPAD_ENABLES_REF_0}; // this is a
dumb workaround
while(I2C_Status != I2C1_MESSAGE_COMPLETE);
I2C1_MasterWrite(ad, 1, VL53L0X_I2CADDR, &I2C_Status);
while(I2C_Status != I2C1_MESSAGE_COMPLETE);
I2C1_MasterWrite(ref_spad_map, sizeof(ref_spad_map),
VL53L0X_I2CADDR, &I2C_Status);
while(I2C_Status != I2C1_MESSAGE_COMPLETE);
// -- VL53L0X_set_reference_spads() end

// DefaultTuningSettings from vl53l0x_tuning.h
// -- VL53L0X_load_tuning_settings() begin
writeRegister(VL53L0X_I2CADDR, 0xFF, 0x01);
writeRegister(VL53L0X_I2CADDR, 0x00, 0x00);

writeRegister(VL53L0X_I2CADDR, 0xFF, 0x00);
writeRegister(VL53L0X_I2CADDR, 0x09, 0x00);
writeRegister(VL53L0X_I2CADDR, 0x10, 0x00);
writeRegister(VL53L0X_I2CADDR, 0x11, 0x00);

writeRegister(VL53L0X_I2CADDR, 0x24, 0x01);
writeRegister(VL53L0X_I2CADDR, 0x25, 0xFF);
writeRegister(VL53L0X_I2CADDR, 0x75, 0x00);

writeRegister(VL53L0X_I2CADDR, 0xFF, 0x01);
writeRegister(VL53L0X_I2CADDR, 0x4E, 0x2C);
writeRegister(VL53L0X_I2CADDR, 0x48, 0x00);
writeRegister(VL53L0X_I2CADDR, 0x30, 0x20);

writeRegister(VL53L0X_I2CADDR, 0xFF, 0x00);
writeRegister(VL53L0X_I2CADDR, 0x30, 0x09);
writeRegister(VL53L0X_I2CADDR, 0x54, 0x00);
writeRegister(VL53L0X_I2CADDR, 0x31, 0x04);
writeRegister(VL53L0X_I2CADDR, 0x32, 0x03);
writeRegister(VL53L0X_I2CADDR, 0x40, 0x83);
writeRegister(VL53L0X_I2CADDR, 0x46, 0x25);
writeRegister(VL53L0X_I2CADDR, 0x60, 0x00);
writeRegister(VL53L0X_I2CADDR, 0x27, 0x00);
writeRegister(VL53L0X_I2CADDR, 0x50, 0x06);
writeRegister(VL53L0X_I2CADDR, 0x51, 0x00);
writeRegister(VL53L0X_I2CADDR, 0x52, 0x96);
writeRegister(VL53L0X_I2CADDR, 0x56, 0x08);
writeRegister(VL53L0X_I2CADDR, 0x57, 0x30);
writeRegister(VL53L0X_I2CADDR, 0x61, 0x00);
writeRegister(VL53L0X_I2CADDR, 0x62, 0x00);
writeRegister(VL53L0X_I2CADDR, 0x64, 0x00);

```

```

writeRegister (VL53L0X_I2CADDR, 0x65, 0x00);
writeRegister (VL53L0X_I2CADDR, 0x66, 0xA0);

writeRegister (VL53L0X_I2CADDR, 0xFF, 0x01);
writeRegister (VL53L0X_I2CADDR, 0x22, 0x32);
writeRegister (VL53L0X_I2CADDR, 0x47, 0x14);
writeRegister (VL53L0X_I2CADDR, 0x49, 0xFF);
writeRegister (VL53L0X_I2CADDR, 0x4A, 0x00);

writeRegister (VL53L0X_I2CADDR, 0xFF, 0x00);
writeRegister (VL53L0X_I2CADDR, 0x7A, 0x0A);
writeRegister (VL53L0X_I2CADDR, 0x7B, 0x00);
writeRegister (VL53L0X_I2CADDR, 0x78, 0x21);

writeRegister (VL53L0X_I2CADDR, 0xFF, 0x01);
writeRegister (VL53L0X_I2CADDR, 0x23, 0x34);
writeRegister (VL53L0X_I2CADDR, 0x42, 0x00);
writeRegister (VL53L0X_I2CADDR, 0x44, 0xFF);
writeRegister (VL53L0X_I2CADDR, 0x45, 0x26);
writeRegister (VL53L0X_I2CADDR, 0x46, 0x05);
writeRegister (VL53L0X_I2CADDR, 0x40, 0x40);
writeRegister (VL53L0X_I2CADDR, 0x0E, 0x06);
writeRegister (VL53L0X_I2CADDR, 0x20, 0x1A);
writeRegister (VL53L0X_I2CADDR, 0x43, 0x40);

writeRegister (VL53L0X_I2CADDR, 0xFF, 0x00);
writeRegister (VL53L0X_I2CADDR, 0x34, 0x03);
writeRegister (VL53L0X_I2CADDR, 0x35, 0x44);

writeRegister (VL53L0X_I2CADDR, 0xFF, 0x01);
writeRegister (VL53L0X_I2CADDR, 0x31, 0x04);
writeRegister (VL53L0X_I2CADDR, 0x4B, 0x09);
writeRegister (VL53L0X_I2CADDR, 0x4C, 0x05);
writeRegister (VL53L0X_I2CADDR, 0x4D, 0x04);

writeRegister (VL53L0X_I2CADDR, 0xFF, 0x00);
writeRegister (VL53L0X_I2CADDR, 0x44, 0x00);
writeRegister (VL53L0X_I2CADDR, 0x45, 0x20);
writeRegister (VL53L0X_I2CADDR, 0x47, 0x08);
writeRegister (VL53L0X_I2CADDR, 0x48, 0x28);
writeRegister (VL53L0X_I2CADDR, 0x67, 0x00);
writeRegister (VL53L0X_I2CADDR, 0x70, 0x04);
writeRegister (VL53L0X_I2CADDR, 0x71, 0x01);
writeRegister (VL53L0X_I2CADDR, 0x72, 0xFE);
writeRegister (VL53L0X_I2CADDR, 0x76, 0x00);
writeRegister (VL53L0X_I2CADDR, 0x77, 0x00);

writeRegister (VL53L0X_I2CADDR, 0xFF, 0x01);
writeRegister (VL53L0X_I2CADDR, 0x0D, 0x01);

writeRegister (VL53L0X_I2CADDR, 0xFF, 0x00);
writeRegister (VL53L0X_I2CADDR, 0x80, 0x01);

```

```

writeRegister(VL53L0X_I2CADDR, 0x01, 0xF8);

writeRegister(VL53L0X_I2CADDR, 0xFF, 0x01);
writeRegister(VL53L0X_I2CADDR, 0x8E, 0x01);
writeRegister(VL53L0X_I2CADDR, 0x00, 0x01);
writeRegister(VL53L0X_I2CADDR, 0xFF, 0x00);
writeRegister(VL53L0X_I2CADDR, 0x80, 0x00);
// -- VL53L0X_load_tuning_settings() end

// "Set interrupt config to new sample ready"
// -- VL53L0X_SetGpioConfig() begin
writeRegister(VL53L0X_I2CADDR, SYSTEM_INTERRUPT_CONFIG_GPIO,
0x04);
writeRegister(VL53L0X_I2CADDR, GPIO_HV_MUX_ACTIVE_HIGH,
readReg(VL53L0X_I2CADDR, GPIO_HV_MUX_ACTIVE_HIGH) & ~0x10); // active
low
if(auto_int_clr) {
writeRegister(VL53L0X_I2CADDR, SYSTEM_INTERRUPT_CLEAR, 0x01);
// clear interrupt
}
// -- VL53L0X_SetGpioConfig() end

measurement_timing_budget_us =
VL53L0X_getMeasurementTimingBudget();

// "Disable MSRC and TCC by default"
// MSRC = Minimum Signal Rate Check
// TCC = Target CentreCheck
// -- VL53L0X_SetSequenceStepEnable() begin

writeRegister(VL53L0X_I2CADDR, SYSTEM_SEQUENCE_CONFIG, 0xE8);

// -- VL53L0X_SetSequenceStepEnable() end

// "Recalculate timing budget"
VL53L0X_setMeasurementTimingBudget(measurement_timing_budget_us);

// VL53L0X_StaticInit() end

// VL53L0X_PerformRefCalibration() begin
(VL53L0X_perform_ref_calibration())

// -- VL53L0X_perform_vhv_calibration() begin

writeRegister(VL53L0X_I2CADDR, SYSTEM_SEQUENCE_CONFIG, 0x01);
if (!VL53L0X_performSingleRefCalibration(0x40)) { return false; }

// -- VL53L0X_perform_vhv_calibration() end

// -- VL53L0X_perform_phase_calibration() begin

writeRegister(VL53L0X_I2CADDR, SYSTEM_SEQUENCE_CONFIG, 0x02);

```

```

    if (!VL53L0X_performSingleRefCalibration(0x00)) { return false; }

    // -- VL53L0X_perform_phase_calibration() end

    // "restore the previous Sequence Config"
    writeRegister(VL53L0X_I2CADDR, SYSTEM_SEQUENCE_CONFIG, 0xE8);

    // VL53L0X_PerformRefCalibration() end

    return true;
}

bool VL53L0X_setSignalRateLimit(float limit_Mcps) {
    if (limit_Mcps < 0 || limit_Mcps > 511.99) { return false; }

    // Q9.7 fixed point format (9 integer bits, 7 fractional bits)
    writeRegister_16b(VL53L0X_I2CADDR,
FINAL_RANGE_CONFIG_MIN_COUNT_RATE_RTN_LIMIT, limit_Mcps * (1 << 7));
    return true;
}

// Get reference SPAD (single photon avalanche diode) count and type
// based on VL53L0X_get_info_from_device(),
// but only gets reference SPAD count and type
bool VL53L0X_getSpadInfo(uint8_t * count, bool * type_is_aperture) {
    uint8_t tmp;

    writeRegister(VL53L0X_I2CADDR, 0x80, 0x01);
    writeRegister(VL53L0X_I2CADDR, 0xFF, 0x01);
    writeRegister(VL53L0X_I2CADDR, 0x00, 0x00);

    writeRegister(VL53L0X_I2CADDR, 0xFF, 0x06);
    writeRegister(VL53L0X_I2CADDR, 0x83, readReg(VL53L0X_I2CADDR, 0x83)
| 0x04);
    writeRegister(VL53L0X_I2CADDR, 0xFF, 0x07);
    writeRegister(VL53L0X_I2CADDR, 0x81, 0x01);

    writeRegister(VL53L0X_I2CADDR, 0x80, 0x01);

    writeRegister(VL53L0X_I2CADDR, 0x94, 0x6b);
    writeRegister(VL53L0X_I2CADDR, 0x83, 0x00);
    startTimeout();
    // HELP
    while (readReg(VL53L0X_I2CADDR, 0x83) == 0x00) {
        if (checkTimeoutExpired()) { return false; }
    }
    writeRegister(VL53L0X_I2CADDR, 0x83, 0x01);
    readRegister(VL53L0X_I2CADDR, 0x92, &tmp);

    *count = tmp & 0x7f;
    *type_is_aperture = (tmp >> 7) & 0x01;
}

```

```

writeRegister(VL53L0X_I2CADDR, 0x81, 0x00);
writeRegister(VL53L0X_I2CADDR, 0xFF, 0x06);
writeRegister(VL53L0X_I2CADDR, 0x83, readReg(VL53L0X_I2CADDR, 0x83)
& ~0x04); // (3) for this
writeRegister(VL53L0X_I2CADDR, 0xFF, 0x01);
writeRegister(VL53L0X_I2CADDR, 0x00, 0x01);

writeRegister(VL53L0X_I2CADDR, 0xFF, 0x00);
writeRegister(VL53L0X_I2CADDR, 0x80, 0x00);

return true;
}

// Get the measurement timing budget in microseconds
// based on VL53L0X_get_measurement_timing_budget_micro_seconds()
// in us
uint32_t VL53L0X_getMeasurementTimingBudget(void) {
    SequenceStepEnables enables;
    SequenceStepTimeouts timeouts;

    uint16_t const StartOverhead      = 1910; // note that this is
different than the value in set_
    uint16_t const EndOverhead        = 960;
    uint16_t const MsrcOverhead       = 660;
    uint16_t const TccOverhead        = 590;
    uint16_t const DssOverhead        = 690;
    uint16_t const PreRangeOverhead   = 660;
    uint16_t const FinalRangeOverhead = 550;

    // "Start and end overhead times always present"
    uint32_t budget_us = StartOverhead + EndOverhead;

    VL53L0X_getSequenceStepEnables(&enables);
    VL53L0X_getSequenceStepTimeouts(&enables, &timeouts);

    if (enables.tcc) {
        budget_us += (timeouts.msrc_dss_tcc_us + TccOverhead);
    }

    if (enables.dss) {
        budget_us += 2 * (timeouts.msrc_dss_tcc_us + DssOverhead);
    }
    else if (enables.msrc) {
        budget_us += (timeouts.msrc_dss_tcc_us + MsrcOverhead);
    }

    if (enables.pre_range) {
        budget_us += (timeouts.pre_range_us + PreRangeOverhead);
    }

    if (enables.final_range) {
        budget_us += (timeouts.final_range_us + FinalRangeOverhead);
    }
}

```



```

    }

    measurement_timing_budget_us = budget_us; // store for internal
reuse
    return budget_us;
}

// Get sequence step enables
// based on VL53L0X_GetSequenceStepEnables()
void VL53L0X_getSequenceStepEnables(SequenceStepEnables * enables) {
    uint8_t sequence_config;
    readRegister(VL53L0X_I2CADDR, SYSTEM_SEQUENCE_CONFIG,
&sequence_config);

    enables->tcc          = (sequence_config >> 4) & 0x1;
    enables->dss          = (sequence_config >> 3) & 0x1;
    enables->msrc         = (sequence_config >> 2) & 0x1;
    enables->pre_range    = (sequence_config >> 6) & 0x1;
    enables->final_range  = (sequence_config >> 7) & 0x1;
}

// Get sequence step timeouts
// based on get_sequence_step_timeout(),
// but gets all timeouts instead of just the requested one, and also
stores
// intermediate values
void VL53L0X_getSequenceStepTimeouts(SequenceStepEnables const *
enables, SequenceStepTimeouts * timeouts) {
    timeouts->pre_range_vcsel_period_pclks =
VL53L0X_getVcselPulsePeriod(VcselPeriodPreRange);

    timeouts->msrc_dss_tcc_mclks = readReg(VL53L0X_I2CADDR,
MSRC_CONFIG_TIMEOUT_MACROP) + 1;
    timeouts->msrc_dss_tcc_us =
        VL53L0X_timeoutMclksToMicroseconds(timeouts->msrc_dss_tcc_mclks,
        timeouts-
>pre_range_vcsel_period_pclks);

    timeouts->pre_range_mclks =
        VL53L0X_decodeTimeout(readRegister_16b(VL53L0X_I2CADDR,
PRE_RANGE_CONFIG_TIMEOUT_MACROP_HI));
    timeouts->pre_range_us =
        VL53L0X_timeoutMclksToMicroseconds(timeouts->pre_range_mclks,
        timeouts-
>pre_range_vcsel_period_pclks);

    timeouts->final_range_vcsel_period_pclks =
VL53L0X_getVcselPulsePeriod(VcselPeriodFinalRange);

    timeouts->final_range_mclks =
        VL53L0X_decodeTimeout(readRegister_16b(VL53L0X_I2CADDR,
FINAL_RANGE_CONFIG_TIMEOUT_MACROP_HI));
}

```

```

    if (enables->pre_range) {
        timeouts->final_range_mclks -= timeouts->pre_range_mclks;
    }

    timeouts->final_range_us =
        VL53L0X_timeoutMclksToMicroseconds(timeouts->final_range_mclks,
        timeouts->final_range_vcsel_period_pclks);
}

// Get the VCSEL pulse period in PCLKs for the given period type.
// based on VL53L0X_get_vcsel_pulse_period()
uint8_t VL53L0X_getVcselPulsePeriod(vcselPeriodType type) {
    if (type == VcselPeriodPreRange) {
        return decodeVcselPeriod(readReg(VL53L0X_I2CADDR,
        PRE_RANGE_CONFIG_VCSEL_PERIOD));
    }
    else if (type == VcselPeriodFinalRange) {
        return decodeVcselPeriod(readReg(VL53L0X_I2CADDR,
        FINAL_RANGE_CONFIG_VCSEL_PERIOD));
    }
    else { return 255; }
}

// Convert sequence step timeout from MCLKs to microseconds with given
VCSEL period in PCLKs
// based on VL53L0X_calc_timeout_us()
uint32_t VL53L0X_timeoutMclksToMicroseconds(uint16_t
timeout_period_mclks, uint8_t vcsel_period_pclks) {
    uint32_t macro_period_ns = calcMacroPeriod(vcsel_period_pclks);

    return ((timeout_period_mclks * macro_period_ns) + (macro_period_ns
/ 2)) / 1000;
}

// Convert sequence step timeout from microseconds to MCLKs with given
VCSEL period in PCLKs
// based on VL53L0X_calc_timeout_mclks()
uint32_t VL53L0X_timeoutMicrosecondsToMclks(uint32_t
timeout_period_us, uint8_t vcsel_period_pclks) {
    uint32_t macro_period_ns = calcMacroPeriod(vcsel_period_pclks);

    return (((timeout_period_us * 1000) + (macro_period_ns / 2)) /
macro_period_ns);
}

// Decode sequence step timeout in MCLKs from register value
// based on VL53L0X_decode_timeout()
// Note: the original function returned a uint32_t, but the return
value is
// always stored in a uint16_t.

```

```

uint16_t VL53L0X_decodeTimeout(uint16_t reg_val) {
    // format: "(LSByte * 2^MSByte) + 1"
    return (uint16_t)((reg_val & 0x00FF) <<
        (uint16_t)((reg_val & 0xFF00) >> 8)) + 1;
}

bool VL53L0X_setMeasurementTimingBudget(uint32_t budget_us) {
    SequenceStepEnables enables;
    SequenceStepTimeouts timeouts;

    uint16_t const StartOverhead      = 1320; // note that this is
different than the value in get_
    uint16_t const EndOverhead        = 960;
    uint16_t const MsrcOverhead       = 660;
    uint16_t const TccOverhead        = 590;
    uint16_t const DssOverhead        = 690;
    uint16_t const PreRangeOverhead   = 660;
    uint16_t const FinalRangeOverhead = 550;

    uint32_t const MinTimingBudget = 20000;

    if (budget_us < MinTimingBudget) { return false; }

    uint32_t used_budget_us = StartOverhead + EndOverhead;

    VL53L0X_getSequenceStepEnables(&enables);
    VL53L0X_getSequenceStepTimeouts(&enables, &timeouts);

    if (enables.tcc) {
        used_budget_us += (timeouts.msrc_dss_tcc_us + TccOverhead);
    }

    if (enables.dss) {
        used_budget_us += 2 * (timeouts.msrc_dss_tcc_us + DssOverhead);
    }
    else if (enables.msrc) {
        used_budget_us += (timeouts.msrc_dss_tcc_us + MsrcOverhead);
    }

    if (enables.pre_range) {
        used_budget_us += (timeouts.pre_range_us + PreRangeOverhead);
    }

    if (enables.final_range) {
        used_budget_us += FinalRangeOverhead;

        // "Note that the final range timeout is determined by the timing
        // budget and the sum of all other timeouts within the sequence.
        // If there is no room for the final range timeout, then an error
        // will be set. Otherwise the remaining time will be applied to
        // the final range."
    }
}

```

```

if (used_budget_us > budget_us) {
    // "Requested timeout too big."
    return false;
}

uint32_t final_range_timeout_us = budget_us - used_budget_us;

// set_sequence_step_timeout() begin
// (SequenceStepId == VL53L0X_SEQUENCESTEP_FINAL_RANGE)

// "For the final range timeout, the pre-range timeout
// must be added. To do this both final and pre-range
// timeouts must be expressed in macro periods MClks
// because they have different vcsel periods."

uint16_t final_range_timeout_mclks =
    VL53L0X_timeoutMicrosecondsToMclks(final_range_timeout_us,
timeouts.final_range_vcsel_period_pclks);

if (enables.pre_range) {
    final_range_timeout_mclks += timeouts.pre_range_mclks;
}

writeRegister_16b(VL53L0X_I2CADDR,
FINAL_RANGE_CONFIG_TIMEOUT_MACROP_HI,
    VL53L0X_encodeTimeout(final_range_timeout_mclks));

// set_sequence_step_timeout() end

measurement_timing_budget_us = budget_us; // store for internal
reuse
}
return true;
}

// Encode sequence step timeout register value from timeout in MCLKs
// based on VL53L0X_encode_timeout()
// Note: the original function took a uint16_t, but the argument
// passed to it
// is always a uint16_t.
uint16_t VL53L0X_encodeTimeout(uint16_t timeout_mclks) {
    // format: "(LSByte * 2^MSByte) + 1"

    uint32_t ls_byte = 0;
    uint16_t ms_byte = 0;

    if (timeout_mclks > 0) {
        ls_byte = timeout_mclks - 1;

        while ((ls_byte & 0xFFFFF00) > 0) {
            ls_byte >>= 1;

```

```

        ms_byte++;
    }

    return (ms_byte << 8) | (ls_byte & 0xFF);
}
else { return 0; }
}

// based on VL53L0X_perform_single_ref_calibration()
bool VL53L0X_performSingleRefCalibration(uint8_t vhw_init_byte) {
    writeRegister(VL53L0X_I2CADDR, SYSRANGE_START, 0x01 |
vhw_init_byte); // VL53L0X_REG_SYSRANGE_MODE_START_STOP

    startTimeout();
    while ((readReg(VL53L0X_I2CADDR, RESULT_INTERRUPT_STATUS) & 0x07)
== 0) {
        if (checkTimeoutExpired()) { return false; }
    }

    if(auto_int_clr) {
        writeRegister(VL53L0X_I2CADDR, SYSTEM_INTERRUPT_CLEAR, 0x01);
// clear interrupt
    }

    writeRegister(VL53L0X_I2CADDR, SYSRANGE_START, 0x00);

    return true;
}

// FOR TOF READS:

// Start continuous ranging measurements. If period_ms (optional) is 0
or not
// given, continuous back-to-back mode is used (the sensor takes
measurements as
// often as possible); otherwise, continuous timed mode is used, with
the given
// inter-measurement period in milliseconds determining how often the
sensor
// takes a measurement.
// based on VL53L0X_StartMeasurement()
void VL53L0X_startContinuous(uint32_t period_ms) {
    writeRegister(VL53L0X_I2CADDR, 0x80, 0x01);
    writeRegister(VL53L0X_I2CADDR, 0xFF, 0x01);
    writeRegister(VL53L0X_I2CADDR, 0x00, 0x00);
    writeRegister(VL53L0X_I2CADDR, 0x91, stop_variable);
    writeRegister(VL53L0X_I2CADDR, 0x00, 0x01);
    writeRegister(VL53L0X_I2CADDR, 0xFF, 0x00);
    writeRegister(VL53L0X_I2CADDR, 0x80, 0x00);

    if (period_ms != 0) {
        // continuous timed mode

```

```

// VL53L0X_SetInterMeasurementPeriodMilliseconds() begin

uint16_t osc_calibrate_val = readRegister_16b(VL53L0X_I2CADDR,
OSC_CALIBRATE_VAL);

if (osc_calibrate_val != 0) {
    period_ms *= osc_calibrate_val;
}

writeRegister_32b(VL53L0X_I2CADDR, SYSTEM_INTERMEASUREMENT_PERIOD,
period_ms);

// VL53L0X_SetInterMeasurementPeriodMilliseconds() end

writeRegister(VL53L0X_I2CADDR, SYSRANGE_START, 0x04); //
VL53L0X_REG_SYSRANGE_MODE_TIMED
}
else {
    // continuous back-to-back mode
    writeRegister(VL53L0X_I2CADDR, SYSRANGE_START, 0x02); //
VL53L0X_REG_SYSRANGE_MODE_BACKTOBACK
}
}

// Stop continuous measurements
// based on VL53L0X_StopMeasurement()
void VL53L0X_stopContinuous(void) {
    writeRegister(VL53L0X_I2CADDR, SYSRANGE_START, 0x01); //
VL53L0X_REG_SYSRANGE_MODE_SINGLESHOT

    writeRegister(VL53L0X_I2CADDR, 0xFF, 0x01);
    writeRegister(VL53L0X_I2CADDR, 0x00, 0x00);
    writeRegister(VL53L0X_I2CADDR, 0x91, 0x00);
    writeRegister(VL53L0X_I2CADDR, 0x00, 0x01);
    writeRegister(VL53L0X_I2CADDR, 0xFF, 0x00);
}

// Returns a range reading in millimeters when continuous mode is
active
// (readRangeSingleMillimeters() also calls this function after
starting a
// single-shot range measurement)
uint16_t VL53L0X_readRangeContinuousMillimeters(void) {
    startTimeout();
    while ((readReg(VL53L0X_I2CADDR, RESULT_INTERRUPT_STATUS) & 0x07)
== 0) {
        if (checkTimeoutExpired()) {
            did_timeout = true;
            return 65535;
        }
    }
}

```



```

/* * * * * * * * * * * * * * * Accelerometer Functions * * * * * * * * * *
* * */
//<Nick>
void H3LIS200DL_begin()
{
    H3LIS200DL_setPowerMode(NORMAL);
    H3LIS200DL_axesEnable(true);

    uint8_t data = 0;

    uint8_t i = 0x21;
    for (i = 0x21; i < 0x25; i++) {
        writeRegister(H3LIS200DL_I2CADDR, i, data);
    }

    uint8_t j = 0x30;
    for (j = 0x30; j < 0x37; j++) {
        writeRegister(H3LIS200DL_I2CADDR, j, data);
    }
}

void H3LIS200DL_axesEnable(bool enable)
{
    uint8_t data;
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG1, &data);
    if (enable)
    {
        data |= 0x07;
    }
    else
    {
        data &= ~0x07;
    }
    writeRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG1, data);
}

void H3LIS200DL_setPowerMode(power_mode pmode)
{
    uint8_t data;
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG1, &data);

    // The power mode is the high three bits of CTRL_REG1. The mode
    // constants are the appropriate bit values left shifted by five,
    // so we
    // need to right shift them to make them work. We also want to mask
    // off the
    // top three bits to zero, and leave the others untouched, so we
    *only*
    // affect the power mode bits.
    data &= ~0xe0; // Clear the top three bits
    data |= pmode<<5; // set the top three bits to our pmode value
}

```



```

    writeRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG1, data); //
write the new value to CTRL_REG1
}

void H3LIS200DL_setODR(data_rate drate)
{
    uint8_t data;
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG1, &data);

    // The data rate is bits 4:3 of CTRL_REG1. The data rate constants
are the
    // appropriate bit values; we need to right shift them by 3 to
align them
    // with the appropriate bits in the register. We also want to mask
off the
    // top three and bottom three bits, as those are unrelated to data
rate and
    // we want to only change the data rate.
    data &=~0x18; // Clear the two data rate bits
    data |= drate<<3; // Set the two data rate bits appropriately.
    writeRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG1, data); //
write the new value to CTRL_REG1
}

void H3LIS200DL_readAxes(int16_t* x, int16_t* y, int16_t* z)
{
    uint8_t data[6]; // create a buffer for our incoming data

    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_X_L, &data[0]);
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_X_H, &data[1]);
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_Y_L, &data[2]);
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_Y_H, &data[3]);
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_Z_L, &data[4]);
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_Z_H, &data[5]);
    // The data that comes out is 12-bit data, left justified, so the
lower
    // four bits of the data are always zero. We need to right shift by
four,
    // then typecase the upper data to an integer type so it does a
signed
    // right shift.
    *x = data[0] | data[1] << 8;
    *y = data[2] | data[3] << 8;
    *z = data[4] | data[5] << 8;
    *x = (*x>>4) ;
    *y = (*y>>4) ;
    *z = (*z>>4) ;
}

int16_t H3LIS200DL_Read_x(int16_t x)
{
    uint8_t data[6];

```

```

    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_X_L, &data[0]);
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_X_H, &data[1]);
    x = data[0] | data[1] << 8;
    x = x >> 4;
    return(x);
}

int16_t H3LIS200DL_Read_y(int16_t y)
{
    uint8_t data[6];
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_Y_L, &data[2]);
    //I2C1_MasterRead(&data[2], 2, H3LIS200DL_I2CADDR, &I2C_Status);
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_Y_H, &data[3]);
    //I2C1_MasterRead(&data[3], 2, H3LIS200DL_I2CADDR, &I2C_Status);
    y = data[2] | data[3] << 8;
    y = y >> 4;
    return(y);
}

int16_t H3LIS200DL_Read_z(int16_t z)
{
    uint8_t data[6];
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_Z_L, &data[4]);
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_OUT_Z_H, &data[5]);
    z = data[4] | data[5] << 8;
    z = z >> 4;
    return(z);
}

int16_t H3LIS200DL_convertToG(int16_t maxScale, int16_t reading)
{
    maxScale = (float)maxScale;
    reading = (float)reading;
    float result = ((maxScale * reading)/2047);
    return ((int16_t)result);
}

void H3LIS200DL_setHighPassCoeff(high_pass_cutoff_freq_cfg hpcoeff)
{
    // The HPF coeff depends on the output data rate. The cutoff
    frequency is
    // is approximately fs/(6*HPc) where HPc is 8, 16, 32 or 64,
    corresponding
    // to the various constants available for this parameter.
    uint8_t data;
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG2, &data);
    data &= ~0xfc; // Clear the two low bits of the CTRL_REG2
    data |= hpcoeff;
    writeRegister(H3LIS200DL_CTRL_REG2, data, 1);
}

void H3LIS200DL_enableHPF(bool enable)

```

```

{
    // Enable the high pass filter
    uint8_t data;
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG2, &data);
    if (enable)
    {
        data |= 1<<5;
    }
    else
    {
        data &= ~(1<<5);
    }
    writeRegister(H3LIS200DL_CTRL_REG2, data, 1);
}

```

```

void H3LIS200DL_HPFOntPin(bool enable, uint8_t pin)
{
    // Enable the hpf on signal to int pins
    uint8_t data;
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG2, &data);
    if (enable)
    {
        if (pin == 1)
        {
            data |= 1<<3;
        }
        if (pin == 2)
        {
            data |= 1<<4;
        }
    }
    else
    {
        if (pin == 1)
        {
            data &= ~1<<3;
        }
        if (pin == 2)
        {
            data &= ~1<<4;
        }
    }
    writeRegister(H3LIS200DL_CTRL_REG2, data, 1);
}

```

```

void H3LIS200DL_intActiveHigh(bool enable)
{
    // Are the int pins active high or active low?
    uint8_t data;
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG3, &data);
    // Setting bit 7 makes int pins active low
    if (!enable)

```

```

    {
        data |= 1<<7;
    }
else
    {
        data &= ~(1<<7);
    }
writeRegister(H3LIS200DL_CTRL_REG3, data, 1);
}

void H3LIS200DL_intPinMode(pp_od _pinMode)
{
    uint8_t data;
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG3, &data);
    // Setting bit 6 makes int pins open drain.
    if (_pinMode == OPEN_DRAIN)
    {
        data |= 1<<6;
    }
else
    {
        data &= ~(1<<6);
    }
    writeRegister(H3LIS200DL_CTRL_REG3, data, 1);
}

void H3LIS200DL_latchInterrupt(bool enable, uint8_t intSource)
{
    // Latch mode for interrupt. When enabled, you must read the
    INTx_SRC reg
    // to clear the interrupt and make way for another.
    uint8_t data;
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG3, &data);
    // Enable latching by setting the appropriate bit.
    if (enable)
    {
        if (intSource == 1)
        {
            data |= 1<<2;
        }
        if (intSource == 2)
        {
            data |= 1<<5;
        }
    }
else
    {
        if (intSource == 1)
        {
            data &= ~1<<2;
        }
        if (intSource == 2)

```

```

        {
            data &= ~1<<5;
        }
    }
    writeRegister(H3LIS200DL_CTRL_REG3, data, 1);
}

void H3LIS200DL_intSrcConfig(int_sig_src src, uint8_t pin)
{
    uint8_t data;
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG3, &data);
    // Enable latching by setting the appropriate bit.
    if (pin == 1)
    {
        data &= ~0xfc; // clear the low two bits of the register
        data |= src;
    }
    if (pin == 2)
    {
        data &= ~0xe7; // clear bits 4:3 of the register
        data |= src<<4;
    }
    writeRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG3, data);
}

void H3LIS200DL_setFullScale(fs_range range)
{
    uint8_t data;
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_CTRL_REG4, &data);
    data &= ~0xcf;
    data |= range<<4;
    writeRegister(H3LIS200DL_CTRL_REG4, data, 1);
}

bool H3LIS200DL_newXData()
{
    uint8_t data;
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_STATUS_REG, &data);
    if (data & 1<<0)
    {
        return true;
    }
    else
    {
        return false;
    }
}

bool H3LIS200DL_newYData()
{
    uint8_t data;

```

```

readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_STATUS_REG, &data);
if (data & 1<<1)
{
    return true;
}
else
{
    return false;
}
}

bool H3LIS200DL_newZData()
{
    uint8_t data;
    readRegister(H3LIS200DL_I2CADDR, H3LIS200DL_STATUS_REG, &data);
    if (data & 1<<2)
    {
        return true;
    }
    else
    {
        return false;
    }
}

void H3LIS200DL_enableInterrupt(int_axis axis, trig_on_level
trigLevel,
                                uint8_t interrupt, bool enable)
{
    uint8_t data, reg, mask;
    mask = 0;
    if (interrupt == 1)
    {
        reg = H3LIS200DL_INT1_CFG;
    }
    else
    {
        reg = H3LIS200DL_INT2_CFG;
    }
    readRegister(H3LIS200DL_I2CADDR, reg, &data);
    if (trigLevel == TRIG_ON_HIGH)
    {
        mask = 1<<1;
    }
    else
    {
        mask = 1;
    }
    if (axis == Z_AXIS) mask = mask<<4;
    if (axis == Y_AXIS) mask = mask<<2;
    if (enable)
    {

```

```

    data |= mask;
}
else
{
    data &= ~mask;
}
writeRegister(H3LIS200DL_I2CADDR, reg, data);
}

void H3LIS200DL_setIntDuration(uint8_t duration, uint8_t intSource)
{
    if (intSource == 1)
    {
        writeRegister(H3LIS200DL_I2CADDR, H3LIS200DL_INT1_DURATION,
duration);
    }
    else
    {
        writeRegister(H3LIS200DL_I2CADDR, H3LIS200DL_INT2_DURATION,
duration);
    }
}

void H3LIS200DL_setIntThreshold(uint8_t threshold, uint8_t intSource)
{
    if (intSource == 1)
    {
        writeRegister(H3LIS200DL_I2CADDR, H3LIS200DL_INT1_THS, threshold);
    }
    else
    {
        writeRegister(H3LIS200DL_I2CADDR, H3LIS200DL_INT2_THS, threshold);
    }
}
//</Nick>

// returns true if one of the axes exceeds the set threshold
bool getAccelPoints(void) {
    uint8_t delay = 5;
    selectPort2(0x00, 0x03); // select Accelerometer port (Mux 0, Port
3)

    // <Nick>
    H3LIS200DL_readAxes(&x_1, &y_1, &z_1);
    timer1 = millis();
    msTimerDelay(delay);

    H3LIS200DL_readAxes(&x_2, &y_2, &z_2);
    timer2 = millis();
    msTimerDelay(delay);

    H3LIS200DL_readAxes(&x_3, &y_3, &z_3);

```

```

timer3 = millis();
msTimerDelay(delay);

H3LIS200DL_readAxes(&x_4, &y_4, &z_4);
timer4 = millis();
msTimerDelay(delay);

H3LIS200DL_readAxes(&x_5, &y_5, &z_5);
timer5 = millis();
msTimerDelay(delay);

// find the maximum value of the 3 axes
if(x_3 > y_3) {
    if(x_3 > z_3) {
        max = x_3;
    }
    else {
        max = z_3;
    }
}
else {
    if(y_3 > z_3) {
        max = y_3;
    }
    else {
        max = z_3;
    }
}
// </Nick>

// if the maximum value is at or above the preset threshold,
return true
return (max >= thresh);
}

bool getAccelPoints2(void) {
uint8_t delay = 1;
selectPort2(0x00, 0x03); // select Accelerometer port (Mux 0, Port
3)

// <Nick>
int16_t max1 = 0x0000;
int16_t max2 = 0x0000;
int16_t max3 = 0x0000;
int16_t max4 = 0x0000;
int16_t max5 = 0x0000;
int16_t truemax = 0x0000;

H3LIS200DL_readAxes(&x_1, &y_1, &z_1);
timer1 = millis();
if(x_1 > y_1) {
    if(x_1 > z_1) {

```



```

        max1 = x_1;
    }
    else {
        max1 = z_1;
    }
}
else {
    if(y_1 > z_1) {
        max1 = y_1;
    }
    else {
        max1 = z_1;
    }
}
msTimerDelay(delay);

H3LIS200DL_readAxes(&x_2, &y_2, &z_2);
timer2 = millis();
if(x_2 > y_2) {
    if(x_2 > z_2) {
        max2 = x_2;
    }
    else {
        max2 = z_2;
    }
}
else {
    if(y_2 > z_2) {
        max2 = y_2;
    }
    else {
        max2 = z_2;
    }
}
msTimerDelay(delay);

H3LIS200DL_readAxes(&x_3, &y_3, &z_3);
timer3 = millis();
if(x_3 > y_3) {
    if(x_3 > z_3) {
        max3 = x_3;
    }
    else {
        max3 = z_3;
    }
}
else {
    if(y_3 > z_3) {
        max3 = y_3;
    }
    else {
        max3 = z_3;
    }
}

```

```

    }
}
msTimerDelay(delay);

H3LIS200DL_readAxes(&x_4, &y_4, &z_4);
timer4 = millis();
if(x_4 > y_4) {
    if(x_4 > z_4) {
        max4 = x_4;
    }
    else {
        max4 = z_4;
    }
}
else {
    if(y_4 > z_4) {
        max4 = y_4;
    }
    else {
        max4 = z_4;
    }
}
msTimerDelay(delay);

H3LIS200DL_readAxes(&x_5, &y_5, &z_5);
timer5 = millis();
if(x_5 > y_5) {
    if(x_5 > z_5) {
        max5 = x_5;
    }
    else {
        max5 = z_5;
    }
}
else {
    if(y_5 > z_5) {
        max5 = y_5;
    }
    else {
        max5 = z_5;
    }
}
msTimerDelay(delay);
//showBinary(y << 1);
if(max1 > max2)
{
    if(max1 > max3)
    {
        if(max1 > max4)
        {
            if(max1 > max5)
            {

```

```

        truemax = max1;
    }
    else
    {
        truemax = max5;
    }
}
else if(max4 > max5)
{
    truemax = max4;
}
else
{
    truemax = max5;
}
}
else if(max3 > max4)
{
    if(max3 > max5)
    {
        truemax = max3;
    }
    else
    {
        truemax = max5;
    }
}
else
{
    if(max4 > max5)
    {
        truemax = max4;
    }
    else
    {
        truemax = max5;
    }
}
}
else if(max2 > max3)
{
    if(max2 > max4)
    {
        if(max2 > max5)
        {
            truemax = max5;
        }
        else
        {
            truemax = max5;
        }
    }
}
}

```



```

msTimerDelay(5);
if( SD_SPI_IsMediaPresent() == false) {
    return;
}
SD_status = f_mount(&drive,"0:", 1);
if (SD_status == FR_OK) { //mount
    if (f_open(&file, filename, FA_WRITE | FA_CREATE_NEW ) ==
FR_OK) { //Open or Create TEST.TXT file
        FW_status = f_write(&file, data0, sizeof(data0)-1,
&actualLength ); //write the first line
        FW_status = f_write(&file, data1, sizeof(data1)-1,
&actualLength );
        FW_status = f_write(&file, data2, sizeof(data2)-1,
&actualLength );
        f_close(&file);
    }
    f_mount(0,"0:",0); //unmount disk
    msTimerDelay(5);
}
}

// write a .CSV containing accelerometer data to the SD card
// pass SD status and File Write status variables for debugging
purposes
void writeAccelToSD(void) {
    // <Nick>
    uint8_t SD_status;
    uint8_t FW_status;
    FATFS drive; // Work area (filesystem object) for logical
drive
    FIL file; // File to write
    UINT actualLength; // Actual length of
    char data0[] = "X, Y, Z, t\r\n";
    char filename[] = "ACCEL.CSV";

    // write 5 data strings in .CSV format for X, Y, and Z axes
    // with millis() timestamps to plot
    sprintf(data1, "%f, %f, %f, %f \r\n", (double)x_1, (double)y_1,
(double)z_1, (double)timer1);
    sprintf(data2, "%f, %f, %f, %f \r\n", (double)x_2, (double)y_2,
(double)z_2, (double)timer2);
    sprintf(data3, "%f, %f, %f, %f \r\n", (double)x_3, (double)y_3,
(double)z_3, (double)timer3);
    sprintf(data4, "%f, %f, %f, %f \r\n", (double)x_4, (double)y_4,
(double)z_4, (double)timer4);
    sprintf(data5, "%f, %f, %f, %f \r\n", (double)x_5, (double)y_5,
(double)z_5, (double)timer5);

    // write the data strings to a file
    if( SD_SPI_IsMediaPresent() == false) {
        return;
    }
}

```

```

SD_status = f_mount(&drive,"0:", 1);
if (SD_status == FR_OK) { // mount disk
    //Open or Create <filename> file
    if (f_open(&file, filename, FA_WRITE | FA_CREATE_NEW ) ==
FR_OK) {
        // write column headers
        FW_status = f_write(&file, data0, sizeof(data0)-1,
&actualLength );

        // write each line of data
        FW_status = f_write(&file, data1, sizeof(data1)-1,
&actualLength );
        FW_status = f_write(&file, data2, sizeof(data2)-1,
&actualLength );
        FW_status = f_write(&file, data3, sizeof(data3)-1,
&actualLength );
        FW_status = f_write(&file, data4, sizeof(data4)-1,
&actualLength );
        FW_status = f_write(&file, data5, sizeof(data5)-1,
&actualLength );

        f_close(&file); // close the file
    }
    f_mount(0,"0:",0); // unmount disk
    msTimerDelay(5);
}
// </Nick>

showConcussion(); // 5 second LED display:
RED/PURPLE/RED/PURPLE/RED
}

/**
End of File
*/

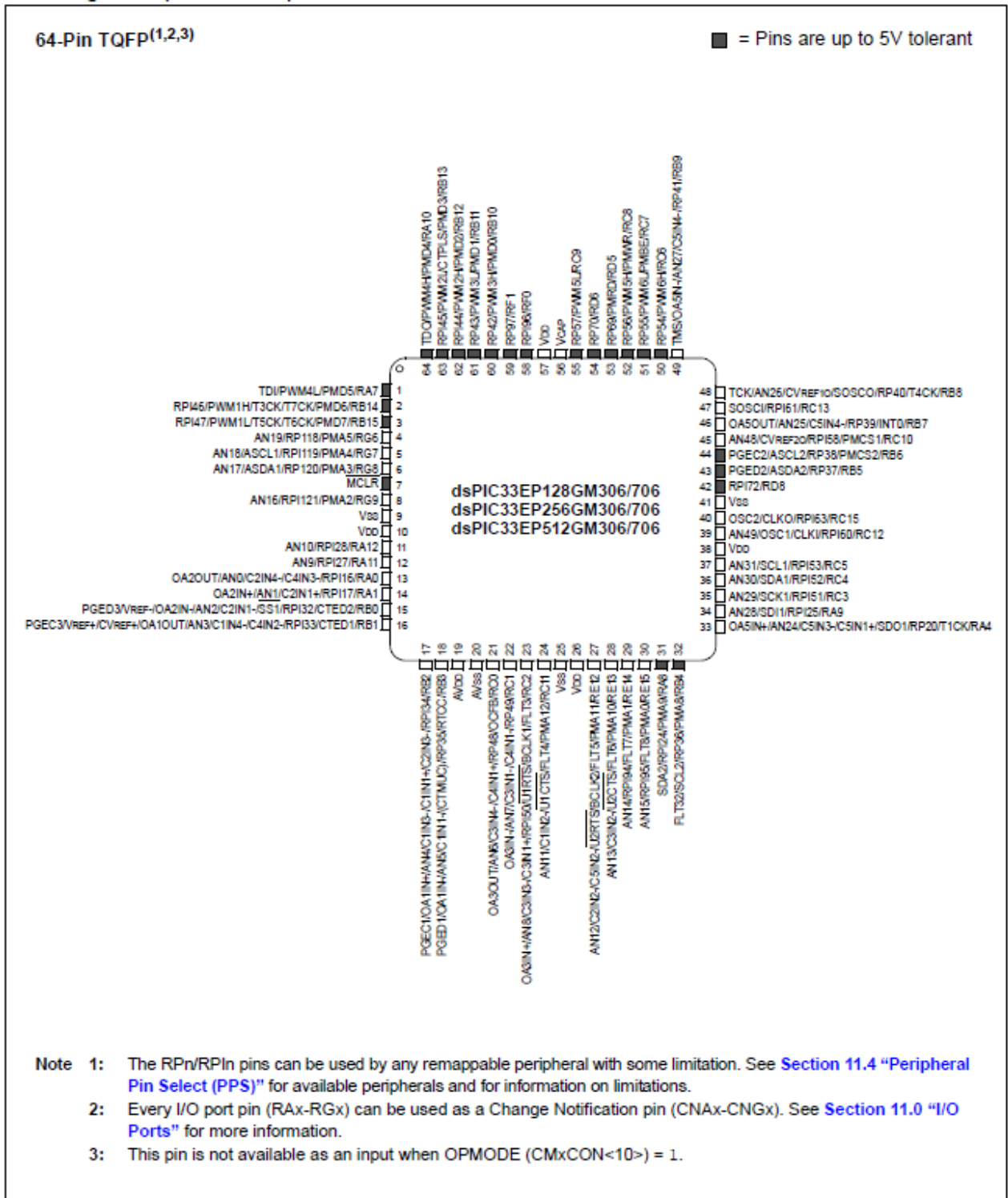
```

## Appendix B: Datasheets

The following datasheet figures are from various subsystems on the H.A.L.O. design. All subsystem datasheets and schematics have use in either board design, schematic design, or code creation. For the physical circuits of the charging, converter, processor, I2C Multiplexers, Accelerometer, and Time-of-Flight sensors, the pin connections, specifications for use and requirements are in use. The schematics in these datasheets references also reflect the creation of eagle schematics in the H.A.L.O. Though not a comprehensive list of all

datasheet components in use, the following figures convey majority of the information used in designs so far. -NK

# Processor datasheet:





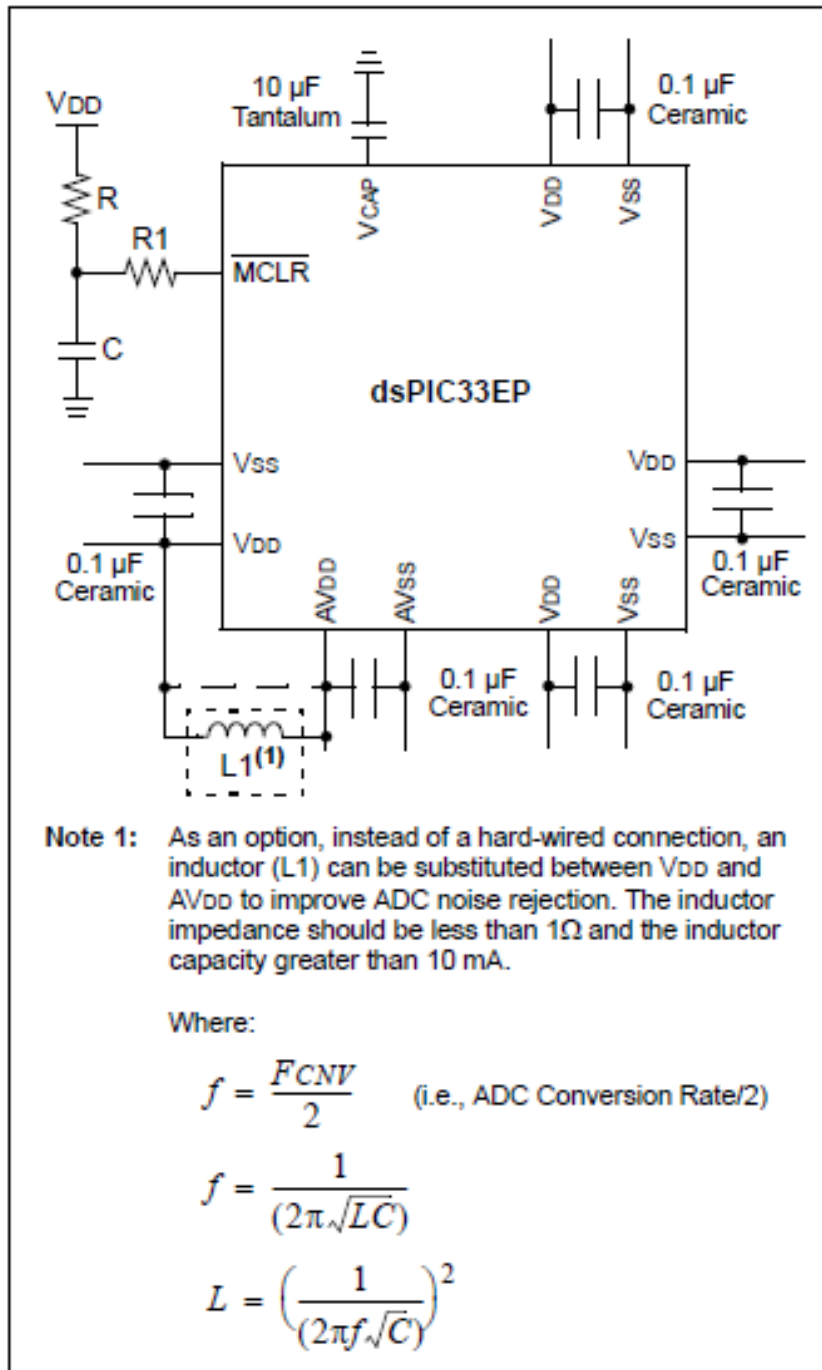
Pin #	Full Pin Name
A1	TDO/PWM4H/PMD4/RA10
A2	RPI45/PWM2L/CTPLS/PMD3/RB13
A3	RP125/RG13
A4	RP42/PWM3H/PMD0/RB10
A5	RPI112/RG0
A6	RP97/RF1
A7	Vdd
A8	No Connect
A9	RPI76/RD12
A10	RP54/RC8
A11	TMS/OA5IN-/AN27/C5IN1-/RP41/RB9
B1	No Connect
B2	AN23/RP127/RG15
B3	RPI44/PWM2H/PMD2/RB12
B4	RP43/PWM3L/PMD1/RB11
B5	RF7
B6	RPI98/RF0
B7	VcAP
B8	RP69/PMRD/RD5
B9	RP55/PMBE/RC7
B10	Vss
B11	TCK/AN28/CVREF10/SOSCO/RP40/T4CK/RB8
C1	RPI46/PWM1H/T3CK/T7CK/PMD6/RB14
C2	Vdd
C3	RPI124/RG12
C4	RP128/RG14
C5	RF6
C6	No Connect
C7	RP57/RC9
C8	RP56/PMWR/RC8
C9	No Connect
C10	SOSCI/RPI61/RC13
C11	AN48/CVREF20/RPI58/PMCS1/RC10
D1	PWM5L/RD1
D2	RPI47/PWM1L/T5CK/T6CK/PMD7/RB15
D3	TDI/PWM4L/PMD5/RA7
D4	No Connect
D5	No Connect
D6	No Connect
D7	RP70/RD6
D8	RPI77/RD13
D9	OA5OUT/AN25/C5IN4-/RP39/INT0/RB7
D10	No Connect
D11	PGEC2/ASCL2/RP38/PMCS2/RB6

Pin #	Full Pin Name
E8	AN47/INT4/RA15
E9	RPI72/RD8
E10	PGED2/ASDA2/RP37/RB5
E11	AN46/INT3/RA14
F1	MCLR
F2	AN17/ASDA1/RP120/PMA3/RG8
F3	AN16/RPI121/PMA2/RG9
F4	AN18/ASCL1/RPI119/PMA4/RG7
F5	Vss
F6	No Connect
F7	No Connect
F8	Vdd
F9	AN49/OSC1/CLKI/RPI60/RC12
F10	Vss
F11	OSC2/CLKO/RPI63/RC15
G1	AN21/RE8
G2	AN20/RE9
G3	AN22/RG10
G4	No Connect
G5	Vdd
G6	Vss
G7	Vss
G8	No Connect
G9	AN45/RF5
G10	AN43/RG3
G11	AN44/RF4
H1	AN10/RPI28/RA12
H2	AN9/RPI27/RA11
H3	No Connect
H4	No Connect
H5	No Connect
H6	Vdd
H7	No Connect
H8	AN28/SDI1/RPI25/RA9
H9	AN29/SCK1/RPI51/RC3
H10	AN31/SCL1/RPI53/RC5
H11	AN42/RG2
J1	OA2OUT/AN0/C2IN4-/C4IN3-/RPI16/RA0
J2	OA2IN+/AN1/C2IN3-/C2IN1+/RPI17/RA1
J3	PGED1/OA1IN-/AN5/C1IN1-/CTMUG/RP35/RTCC/RB3
J4	AVdd
J5	AN11/C1IN2-/U1CTS/FLT4/PMA12/RC11
J6	AN35/RG11
J7	AN12/C2IN2-/C5IN2-/U2RTS/BCLK2/FLT5/PMA11/RE12

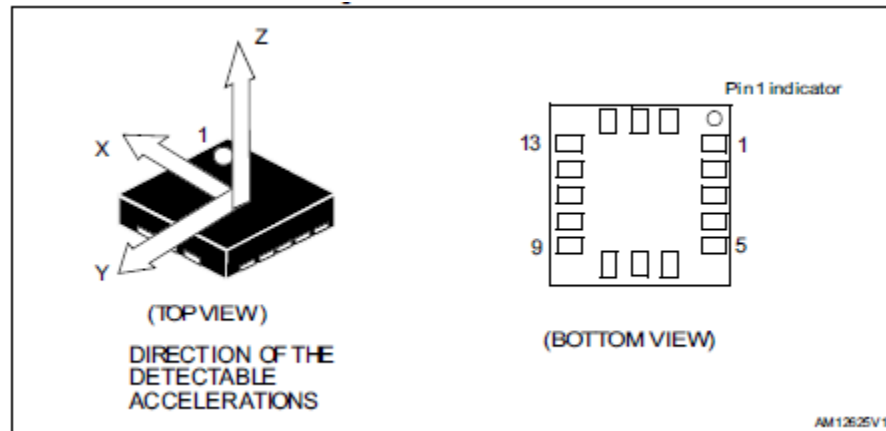
Pin #	Full Pin Name
E1	PWM6H/T8CK/RD4
E2	PWM6L/T9CK/RD3
E3	AN19/RP118/PMA5/RG6
E4	PWM5H/RD2
E5	No Connect
E6	RP113/RG1
E7	No Connect
K4	OA3OUT/AN6/C3IN4-/C4IN4-/C4IN1+/RP48/OCFB/RCD
K5	No Connect
K6	AN37/RF12
K7	AN14/RP194/FLT7/PMA1/RE14
K8	VDD
K9	AN39/RD15
K10	OA5IN+/AN24/C5IN3-/C5IN1+/SDO1/RP20/T1CK/RA4
K11	AN40/RP180/RED
L1	PGEC1/OA11N+/AN4/C1IN3-/C1IN1+/C2IN3-/RP134/RB2
L2	VREF-/AN33/PMA6/RF9

Pin #	Full Pin Name
J8	No Connect
J9	No Connect
J10	AN41/RP81/RE1
J11	AN30/SDA1/RP152/RC4
K1	PGED3/OA2IN-/AN2/C2IN1-/SS1/RP132/CTED2/RB0
K2	PGEC3/CVREF+/OA1OUT/AN3/C1IN4-/C4IN2-/RP133/CTED1/RB1
K3	VREF+/AN34/PMA7/RF10
L3	AVSS
L4	OA3IN-/AN7/C3IN1-/C4IN1-/RP49/RC1
L5	OA3IN+/AN8/C3IN3-/C3IN1+/RP150/U1RTS/BCLK1/FLT3/PMA13/RC2
L6	AN36/RF13
L7	AN13/C3IN2-/U2CTS/FLT6/PMA10/RE13
L8	AN15/RP195/FLT8/PMA0/RE15
L9	AN38/RD14
L10	SDA2/RP124/PMA9/RA8
L11	FLT32/SCL2/RP36/PMA8/RB4

**Minimum recommended connections:**



## Accelerometer datasheet:

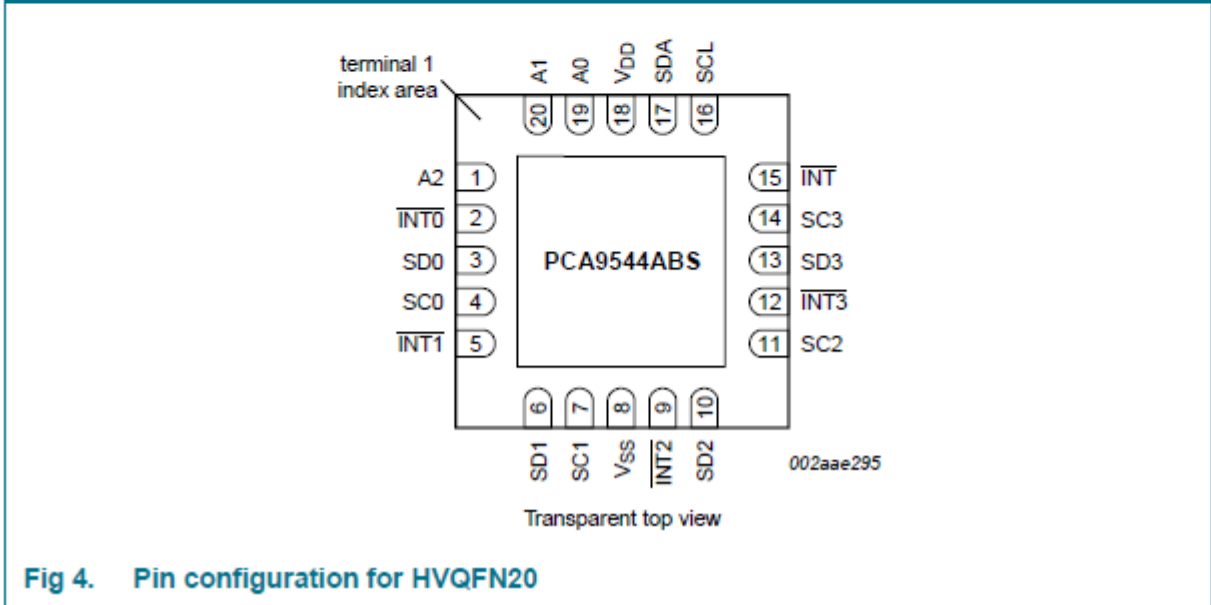
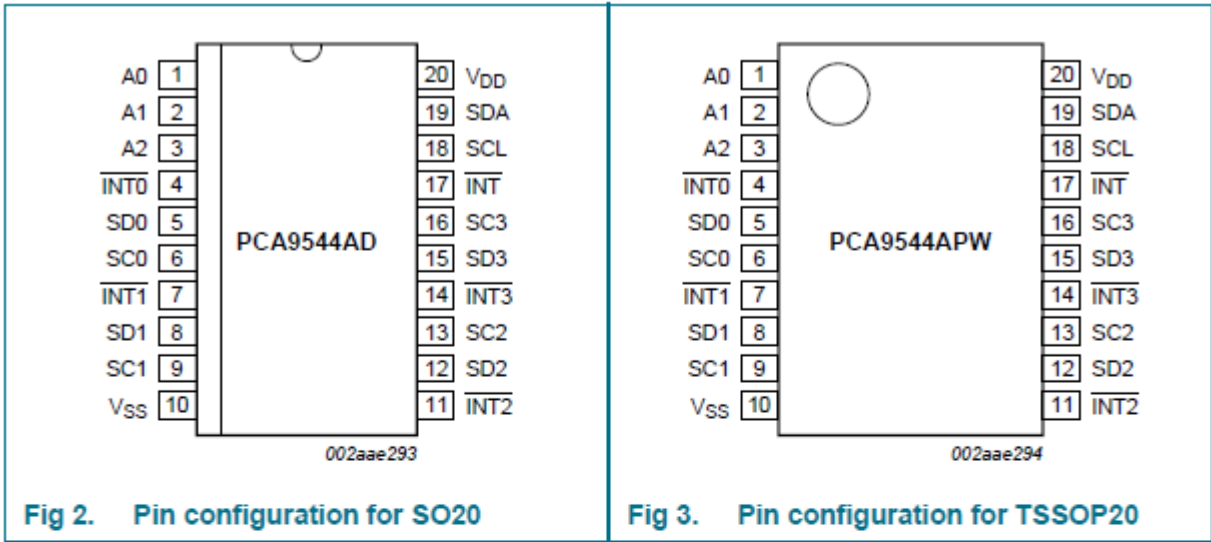


Pin#	Name	Function
1	Vdd_IO	Power supply for I/O pins
2	NC	Not connected
3	NC	Not connected
4	SCL SPC	I <sup>2</sup> C serial clock (SCL) SPI serial port clock (SPC)
5	GND	0 V supply
6	SDA SDI SDO	I <sup>2</sup> C serial data (SDA) SPI serial data input (SDI) 3-wire interface serial data output (SDO)
7	SDO SA0	SPI serial data output (SDO) I <sup>2</sup> C less significant bit of the device address (SA0)
8	CS	SPI enable I <sup>2</sup> C/SPI mode selection (1: I <sup>2</sup> C mode; 0: SPI enabled)
9	INT 2	Inertial interrupt 2
10	Reserved	Connect to GND
11	INT 1	Inertial interrupt 1
12	GND	0 V supply
13	GND	0 V supply
14	Vdd	Power supply
15	Reserved	Connect to Vdd
16	GND	0 V supply

**Max ratings:**

Symbol	Ratings	Maximum value	Unit
V <sub>DD</sub>	Supply voltage	-0.3 to 4.8	V
V <sub>DD_IO</sub>	I/O pins supply voltage	-0.3 to 4.8	V
V <sub>IN</sub>	Input voltage on any control pin (CS, SCL/SPC, SDA/SDI/SDO, SDO/SA0)	-0.3 to V <sub>DD_IO</sub> +0.3	V
A <sub>POW</sub>	Acceleration (any axis, powered, V <sub>DD</sub> = 2.5 V)	3000 g for 0.5 ms	
		10000 g for 0.1 ms	
A <sub>UNP</sub>	Acceleration (any axis, unpowered)	3000 g for 0.5 ms	
		10000 g for 0.1 ms	
T <sub>OP</sub>	Operating temperature range	-40 to +85	°C
T <sub>STG</sub>	Storage temperature range	-40 to +125	°C
ESD	Electrostatic discharge protection	4 (HBM)	kV
		1.5 (CDM)	kV
		200 (MM)	V

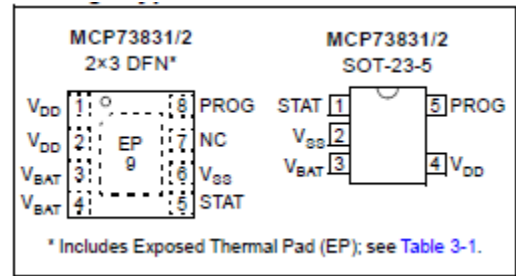
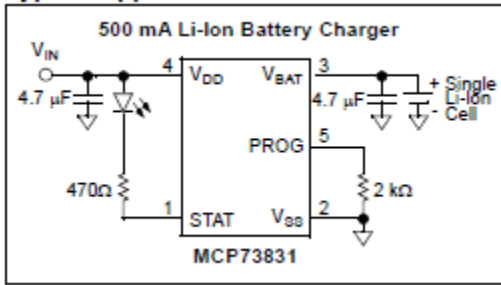
**I2C Multiplexer datasheet:**



Symbol	Pin		Description
	SO20, TSSOP20	HVQFN20	
A0	1	19	address input 0
A1	2	20	address input 1
A2	3	1	address input 2
$\overline{\text{INT0}}$	4	2	active LOW interrupt input 0
SD0	5	3	serial data 0
SC0	6	4	serial clock 0
$\overline{\text{INT1}}$	7	5	active LOW interrupt input 1
SD1	8	6	serial data 1
SC1	9	7	serial clock 1
V <sub>SS</sub>	10	8 <sup>(1)</sup>	supply ground
$\overline{\text{INT2}}$	11	9	active LOW interrupt input 2
SD2	12	10	serial data 2
SC2	13	11	serial clock 2
$\overline{\text{INT3}}$	14	12	active LOW interrupt input 3
SD3	15	13	serial data 3
SC3	16	14	serial clock 3
$\overline{\text{INT}}$	17	15	active LOW interrupt output
SCL	18	16	serial clock line
SDA	19	17	serial data line
V <sub>DD</sub>	20	18	supply voltage

Symbol	Parameter	Conditions	Min	Max	Unit
V <sub>DD</sub>	supply voltage		-0.5	+7.0	V
V <sub>I</sub>	input voltage		-0.5	+7.0	V
I <sub>I</sub>	input current		-	±20	mA
I <sub>O</sub>	output current		-	±25	mA
I <sub>DD</sub>	supply current		-	±100	mA
I <sub>SS</sub>	ground supply current		-	±100	mA
P <sub>tot</sub>	total power dissipation		-	400	mW
T <sub>j(max)</sub>	maximum junction temperature	<sup>(1)</sup>	-	125	°C
T <sub>stg</sub>	storage temperature		-60	+150	°C
T <sub>amb</sub>	ambient temperature	operating	-40	+85	°C

## Charging circuit datasheet:



## DC CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, all limits apply for  $V_{DD} = [V_{REG}(\text{typical}) + 1.0V]$  to 6V,  $T_A = -40^\circ\text{C}$  to  $+85^\circ\text{C}$ . Typical values are at  $+25^\circ\text{C}$ .

Parameters	Sym.	Min.	Typ.	Max.	Units	Conditions
<b>Supply Input</b>						
Supply Voltage	$V_{DD}$	3.75	—	6	V	
Supply Current	$I_{SS}$	—	510	1500	$\mu\text{A}$	Charging
		—	53	200	$\mu\text{A}$	Charge Complete, No Battery
		—	25	50	$\mu\text{A}$	PROG Floating
		—	1	5	$\mu\text{A}$	$V_{DD} \leq (V_{BAT} - 50 \text{ mV})$
		—	0.1	2	$\mu\text{A}$	$V_{DD} < V_{STOP}$
UVLO Start Threshold	$V_{START}$	3.3	3.45	3.6	V	$V_{DD}$ Low-to-High
UVLO Stop Threshold	$V_{STOP}$	3.2	3.38	3.5	V	$V_{DD}$ High-to-Low
UVLO Hysteresis	$V_{HYS}$	—	70	—	mV	
<b>Voltage Regulation (Constant-Voltage Mode)</b>						
Regulated Output Voltage	$V_{REG}$	4.168	4.20	4.232	V	MCP7383X-2
		4.317	4.35	4.383	V	MCP7383X-3
		4.367	4.40	4.433	V	MCP7383X-4
		4.466	4.50	4.534	V	MCP7383X-5
Line Regulation	$ \frac{\Delta V_{BAT}}{V_{BAT}} / \frac{\Delta V_{DD}}{V_{DD}} $	—	0.09	0.30	%/V	$V_{DD} = [V_{REG}(\text{typical}) + 1V]$ to 6V, $I_{OUT} = 10 \text{ mA}$
Load Regulation	$ \frac{\Delta V_{BAT}}{V_{BAT}} $	—	0.05	0.30	%	$I_{OUT} = 10 \text{ mA}$ to 50 mA $V_{DD} = [V_{REG}(\text{typical}) + 1V]$
Supply Ripple Attenuation	PSRR	—	52	—	dB	$I_{OUT} = 10 \text{ mA}$ , 10Hz to 1 kHz
		—	47	—	dB	$I_{OUT} = 10 \text{ mA}$ , 10Hz to 10 kHz
		—	22	—	dB	$I_{OUT} = 10 \text{ mA}$ , 10Hz to 1 MHz
<b>Current Regulation (Fast Charge Constant-Current Mode)</b>						
Fast Charge Current Regulation	$I_{REG}$	90	100	110	mA	PROG = 10 k $\Omega$
		450	505	550	mA	PROG = 2.0 k $\Omega$ , <b>Note 1</b>
		12.5	14.5	16.5	mA	PROG = 67 k $\Omega$
						$T_A = -5^\circ\text{C}$ to $+55^\circ\text{C}$



Electrical Specifications: Unless otherwise indicated, all limits apply for $V_{DD} = [V_{REG}(\text{typical}) + 1.0V]$ to 6V, $T_A = -40^\circ\text{C}$ to $+85^\circ\text{C}$ . Typical values are at $+25^\circ\text{C}$ .						
Parameters	Sym.	Min.	Typ.	Max.	Units	Conditions
<b>Preconditioning Current Regulation (Trickle Charge Constant-Current Mode)</b>						
Precondition Current Ratio	$I_{PREG} / I_{REG}$	7.5	10	12.5	%	PROG = 2.0 k $\Omega$ to 10 k $\Omega$
		15	20	25	%	PROG = 2.0 k $\Omega$ to 10 k $\Omega$
		30	40	50	%	PROG = 2.0 k $\Omega$ to 10 k $\Omega$
		—	100	—	%	No Preconditioning $T_A = -5^\circ\text{C}$ to $+55^\circ\text{C}$
Precondition Voltage Threshold Ratio	$V_{PTH} / V_{REG}$	64	66.5	69	%	$V_{BAT}$ Low-to-High
		69	71.5	74	%	$V_{BAT}$ Low-to-High
Precondition Hysteresis	$V_{PHYS}$	—	110	—	mV	$V_{BAT}$ High-to-Low
<b>Charge Termination</b>						
Charge Termination Current Ratio	$I_{TERM} / I_{REG}$	3.75	5	6.25	%	PROG = 2.0 k $\Omega$ to 10 k $\Omega$
		5.6	7.5	9.4	%	PROG = 2.0 k $\Omega$ to 10 k $\Omega$
		8.5	10	11.5	%	PROG = 2.0 k $\Omega$ to 10 k $\Omega$
		15	20	25	%	PROG = 2.0 k $\Omega$ to 10 k $\Omega$ $T_A = -5^\circ\text{C}$ to $+55^\circ\text{C}$
<b>Automatic Recharge</b>						
Recharge Voltage Threshold Ratio	$V_{RTH} / V_{REG}$	91.5	94.0	96.5	%	$V_{BAT}$ High-to-Low
		94	96.5	99	%	$V_{BAT}$ High-to-Low
<b>Pass Transistor ON-Resistance</b>						
ON-Resistance	$R_{DSON}$	—	350	—	m $\Omega$	$V_{DD} = 3.75V$ , $T_J = 105^\circ\text{C}$
<b>Battery Detection</b>						
Battery Detection Current	$I_{BAT\_DET}$	0.6	6	—	$\mu\text{A}$	$V_{BAT}$ Source Current
No-Battery-Present Threshold	$V_{NO\_BAT}$	—	$V_{REG} + 100\text{ mV}$	—	V	$V_{BAT}$ Voltage $\geq V_{NO\_BAT}$ for No Battery condition
No-Battery-Present Impedance	$Z_{NO\_BAT}$	7	—	—	M $\Omega$	$V_{BAT}$ Impedance $\geq Z_{NO\_BAT}$ for No Battery condition, <a href="#">Note 1</a>
<b>Battery Discharge Current</b>						
Output Reverse Leakage Current	$I_{DISCHARGE}$	—	0.15	2	$\mu\text{A}$	PROG Floating
		—	0.25	2	$\mu\text{A}$	$V_{DD}$ Floating
		—	0.15	2	$\mu\text{A}$	$V_{DD} < V_{STOP}$
		—	-5.5	-15	$\mu\text{A}$	Charge Complete
<b>Status Indicator – STAT</b>						
Sink Current	$I_{SINK}$	—	—	25	mA	
Low Output Voltage	$V_{OL}$	—	0.4	1	V	$I_{SINK} = 4\text{ mA}$
Source Current	$I_{SOURCE}$	—	—	35	mA	
High Output Voltage	$V_{OH}$	—	$V_{DD} - 0.4$	$V_{DD} - 1$	V	$I_{SOURCE} = 4\text{ mA}$ (MCP73831)
Input Leakage Current	$I_{LK}$	—	0.03	1	$\mu\text{A}$	High-Impedance
<b>PROG Input</b>						
Charge Impedance Range	$R_{PROG}$	2	—	67	k $\Omega$	
Minimum Shutdown Impedance	$R_{PROG}$	70	—	200	k $\Omega$	
<b>Automatic Power Down</b>						
Automatic Power Down Entry Threshold	$V_{PDENTER}$	$V_{DD} < (V_{BAT} + 20\text{ mV})$	$V_{DD} < (V_{BAT} + 50\text{ mV})$	—		$3.5V \leq V_{BAT} \leq V_{REG}$ $V_{DD}$ Falling

Electrical Specifications: Unless otherwise indicated, all limits apply for  $V_{DD} = [V_{REG}(\text{typical}) + 1.0V]$  to 6V,  $T_A = -40^\circ\text{C}$  to  $+85^\circ\text{C}$ . Typical values are at  $+25^\circ\text{C}$ .

Parameters	Sym.	Min.	Typ.	Max.	Units	Conditions
Automatic Power Down Exit Threshold	$V_{PDEXIT}$	—	$V_{DD} < (V_{BAT} + 150 \text{ mV})$	$V_{DD} < (V_{BAT} + 200 \text{ mV})$		$3.5V \leq V_{BAT} \leq V_{REG}$ $V_{DD}$ Rising
<b>Thermal Shutdown</b>						
Die Temperature	$T_{SD}$	—	150	—	$^\circ\text{C}$	
Die Temperature Hysteresis	$T_{SDHYS}$	—	10	—	$^\circ\text{C}$	

## AC CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, all limits apply for  $V_{DD} = [V_{REG}(\text{typical}) + 1.0V]$  to 6V,  $T_A = -40^\circ\text{C}$  to  $+85^\circ\text{C}$ . Typical values are at  $+25^\circ\text{C}$ .

Parameters	Sym.	Min.	Typ.	Max.	Units	Conditions
UVLO Start Delay	$t_{START}$	—	—	5	ms	$V_{DD}$ Low-to-High
<b>Constant-Current Regulation</b>						
Transition Time Out of Preconditioning	$t_{DELAY}$	—	—	1	ms	$V_{BAT} < V_{PTH}$ to $V_{BAT} > V_{PTH}$
Current Rise Time Out of Preconditioning	$t_{RISE}$	—	—	1	ms	$I_{OUT}$ Rising to 90% of $I_{REG}$
Termination Comparator Filter	$t_{TERM}$	0.4	1.3	3.2	ms	Average $I_{OUT}$ Falling
Charge Comparator Filter	$t_{CHARGE}$	0.4	1.3	3.2	ms	Average $V_{BAT}$
<b>Status Indicator</b>						
Status Output Turn-Off	$t_{OFF}$	—	—	200	$\mu\text{s}$	$I_{SINK} = 1 \text{ mA to } 0 \text{ mA}$
Status Output Turn-On	$t_{ON}$	—	—	200	$\mu\text{s}$	$I_{SINK} = 0 \text{ mA to } 1 \text{ mA}$

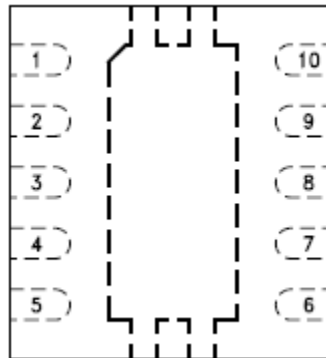
## Pin functions:

Pin No.		Symbol	Function
DFN	SOT-23-5		
1	4	$V_{DD}$	Battery Management Input Supply
2	—	$V_{DD}$	Battery Management Input Supply
3	3	$V_{BAT}$	Battery Charge Control Output
4	—	$V_{BAT}$	Battery Charge Control Output
5	1	STAT	Charge Status Output
6	2	$V_{SS}$	Battery Management 0V Reference
7	—	NC	No Connection
8	5	PROG	Current Regulation Set and Charge Control Enable
9	—	EP	Exposed Thermal Pad (EP); must be connected to $V_{SS}$

## Buck boost datasheet:

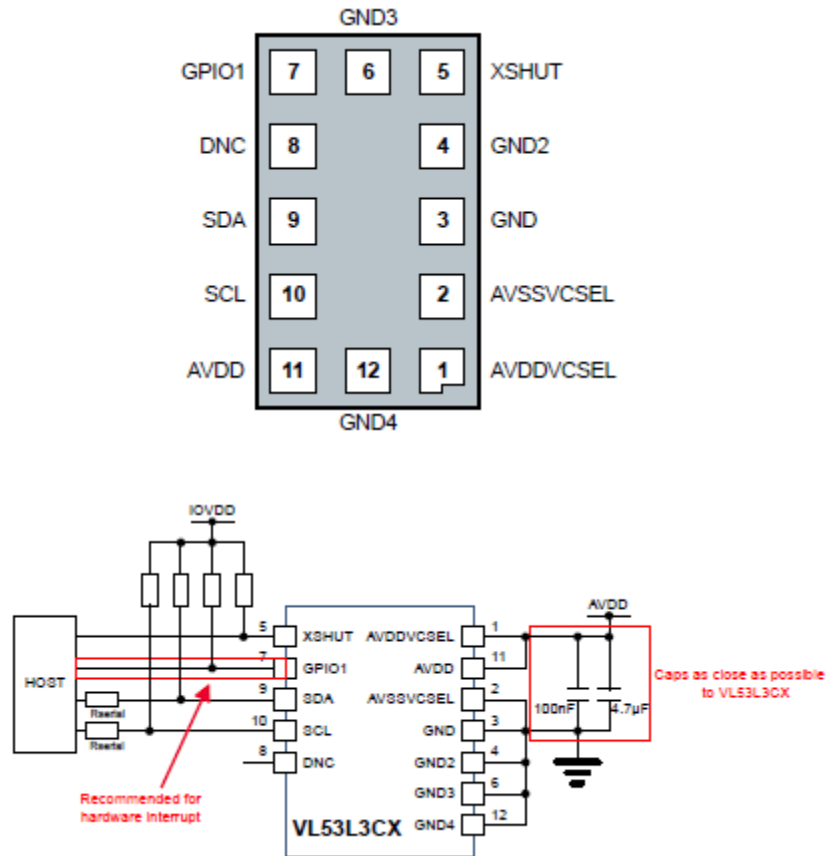
### Max ratings

Symbol	Parameter	Value	Unit
VINA, VIN	Input voltage	-0.3 to 7	V
VOUT	Output voltage	-0.3 to 7	V
SW1, SW2	DC voltage	-0.3 to 7	V
FB	DC voltage	-0.3 to 1.5	V
MODE/SYNC, EN	DC voltage	-0.3 to 7	V
T <sub>J</sub>	Maximum junction temperature	150	°C
T <sub>STG</sub>	Storage temperature range	-65 to +150	°C
T <sub>JOP</sub>	Operating junction temperature range	-40 to +85	°C
ESD	Human body model	2	kV



Pin	Symbol	Name and function
1	VOUT	Output voltage
2	SW2	Switch pin. Internal switches are connected to this pin. Connect inductor between SW1 to SW2
3	PGND	Power ground
4	SW1	Switch pin. Internal switches are connected to this pin. Connect inductor between SW1 and SW2
5	VIN	Power input voltage. Connect a ceramic bypass capacitor (10 $\mu$ F minimum) between this pin and PGND
6	EN	Enable pin. Connect this pin to GND or a voltage lower than 0.4 V to shut down the IC. A voltage higher than 1.2 V is required to enable the IC
7	MODE / SYNC	Operation mode selection. If MODE pin is low, the STBB1-AXX automatically switches between pulse skipping and fixed frequency PWM according to the load level. If MODE pin is pulled high, the STBB1-AXX works in PWM mode. When a square waveform is applied, this pin provides the clock signal for oscillator synchronization
8	VINA	Supply voltage for control stage
9	GND	Signal ground
10	FB	Feedback voltage
	Exposed pad	Power ground

## Time-of-Flight datasheet:



Pin number	Signal name	Signal type	Signal description
1	AVDDVCSEL	Supply	VCSEL supply, to be connected to main supply
2	AVSSVCSEL	Ground	VCSEL ground, to be connected to main ground
3	GND		To be connected to main ground
4	GND2		To be connected to main ground
5	XSHUT	Digital input	Xshutdown pin, active low
6	GND3	Ground	To be connected to main ground
7	GPIO1	Digital output	Interrupt output. Open drain output
8	DNC	Digital input	Do not connect, must be left floating
9	SDA	Digital input/output	I2C serial data
10	SCL	Digital input	I2C serial clock input
11	AVDD	Supply	Supply, to be connected to main supply
12	GND4	Ground	To be connected to main ground

## Appendix C: Parts Order Forms

The following spreadsheets depict which parts were ordered on what day, the amount ordered, and the cost per unit. -BT

### 21 January 2021 Parts Order Form:

DT #		4 Project Leader: Brian Thomson			Email: <a href="mailto:bat61@uakron.edu">bat61@uakron.edu</a>				
Qty.	Refdes	Part Num.	Description	Suggested Vendor	Vendor Part Num.	Catalog #/Page #/Hyperlink	Cost	Total Cost	
12	C1-C8	CL21B104KBCNNNC	0.1uF Ceramic Capacitor 0805	Digi-Key	1276-1003-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.05	\$0.56	
4	C7-C8	CL21A475KACLRLNC	4.7uF Ceramic Capacitor 0805	Digi-Key	1276-2415-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.24	\$0.96	
2	C9	CL21A106MCFNNNE	10uF Ceramic Capacitor 0805	Digi-Key	1276-1288-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.10	\$0.20	
2	C10	CL21A226MAQNNNE	22uF Ceramic Capacitor 0805	Digi-Key	1276-2908-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.34	\$0.68	
3	C11	TCP1A106M8R	CAP TANT 10uF 20% 10V 0805	Digi-Key	511-1685-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.56	\$1.68	
16	R17-R24	RC2012J102CS	1.00kOhm Resistor 0805	Digi-Key	1276-5531-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.13	\$2.08	
6	R2, R25-R26, R61	RC2012J103CS	10.0kOhm Resistor 0805	Digi-Key	1276-5552-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.13	\$0.78	
36	R27-R44	RC2012J222CS	2.20kOhm Resistor 0806	Digi-Key	1276-5537-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.13	\$4.68	
4	R1, R45	RC2012J471CS	470Ohm Resistor 0805	Digi-Key	1276-5523-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.14	\$0.56	
2	R46	ERA-6AEB202V	2.00kOhm Resistor 0805	Digi-Key	P2.0KDACT-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.31	\$0.62	
4	R47-R48	RC2012F270CS	27.0Ohm Resistor 0805	Digi-Key	1276-5198-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.14	\$0.56	
2	R49	RMCF0805JT560R	560kOhm Resistor 0805	Digi-Key	RMCF0805JT560RCT-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.10	\$0.20	
2	R50	ERA-6AEB104V	100kOhm Resistor 0805	Digi-Key	P100KDACT-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.31	\$0.62	
16	D1-D8	LY R376-PS-36	LED YELLOW DIFFUSED 0805 SMD	Digi-Key	475-2560-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.27	\$4.32	
2	D9	LG R971-KN-1	LED GREEN DIFFUSED 0805 SMD	Digi-Key	475-1410-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.25	\$0.50	
16	J1-J2, J5-J12	2011-1X05TSD025B	Connector Header Through Hole 5 position 0.100"	Digi-Key	2553-2011-1X05TSD025B-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.64	\$15.12	
2	J3	DM3D-SF	CONN MICRO SD CARD PUSH-PULL	Digi-Key	HR1941CT-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$1.65	\$3.30	
1	J4	2011-1X03G00SD025B	PIN HEADER, SINGLE ROW, 3 PIN, S	Digi-Key	2553-2011-1X03G00SD025B-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.06	\$0.06	
1	J14	2011H-1X06G015B	PIN HEADER, SINGLE ROW, 6 PIN, S	Digi-Key	2553-2011H-1X06G015B-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.87	\$0.87	
4	J15-J16	2011-1X02TSH035B	PIN HEADER, SINGLE ROW, 2 PIN, T	Digi-Key	2553-2011-1X02TSH035B-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.51	\$2.04	
2	J17	473460001	CONN RCPT USB2.0 MICRO B SMD R/A	Digi-Key	WM117141CT-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.95	\$1.90	
3	IC1	DSPIC33EP512GM706-I/PT	Microchip DSPIC33EP512GM706-I/PT, 16bit dsPIC Microcontroller, 60MHz, 512 kB Flash, 64-Pin TQFP	Digi-Key	DSPIC33EP512GM706-I/PT-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$6.17	\$18.51	
3	US1	VLFC4020T-2R2N1R7	FIXED IND 2.2uH 1.72A 59 MOHM	Digi-Key	445-VLFC4020T-2R2N1R7CT-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.29	\$0.87	
4	U1-U2	PCA9544APW,118	I2C Multiplexer	Digi-Key	568-1861-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$1.78	\$7.12	
2	U3	TXB0104PWR	Voltage Level Translator Bidirectional 1 Circuit 4 Channel 100Mbps 14-TSSOP	Digi-Key	296-21929-2-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.92	\$1.84	
2	U4	STBB1-APUR	Buck Boost Converter	Digi-Key	497-11971-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$2.34	\$4.68	
2	U5	STC4054GR	Battery Charging	Digi-Key	497-5809-2-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.63	\$1.26	
2	U6	H3LS200DLTR	H3LS200DL Series 3.6 V 400 Hz Low-Power 3-Axis Digital Accelerometer -TFLGA-16L	Digi-Key	497-15698-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$6.68	\$13.36	
2	S1	CT11025.0F160	Momentary Switch	Digi-Key	2449-CT11025.0F160-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.12	\$0.24	
12	U1 (ToF Brd)	VL53LOCXV0DH1	Time-of-Flight ranging sensor	ST	VL53LOCXV0DH1	<a href="https://estore.st.com/en/v530c">https://estore.st.com/en/v530c</a>	\$3.88	\$46.56	
2	C1	CL21A475KACLRLNC	4.7uF Ceramic Capacitor 0805	Digi-Key	1276-2415-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.24	\$0.48	
2	C2	CL21B104KBCNNNC	0.1uF Ceramic Capacitor 0805	Digi-Key	1276-1003-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.05	\$0.09	
4	R1-R2	RC2012J102CS	1.0kOhm Resistor 0805	Digi-Key	1276-5552-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.13	\$0.52	
2	R3	RC2012J102CS	1.0kOhm Resistor 0805	Digi-Key	1276-5531-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.13	\$0.26	
4	J1-J2	2011-1X05TSD025B	Connector Header Through Hole 5 position 0.100"	Digi-Key	2553-2011-1X05TSD025B-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.84	\$3.36	
6	LED1 (RGB Brd)	1655	ADDRESS LED DISC SERIAL RGB 1=10	Digi-Key	1528-1104-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$4.50	\$27.00	
6	J1-J2	2011-1X03G00SD025B	PIN HEADER, SINGLE ROW, 3 PIN, S	Digi-Key	2553-2011-1X03G00SD025B-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.06	\$0.36	
6	C1	CL21B105KAFNFNE	CAP CER 1uF 25V X7R 0805	Digi-Key	1276-2926-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.10	\$0.60	
							<b>Total</b>	<b>\$169.41</b>	

### 2 February 2021 Parts Order Form:

DT #		4 Project Leader: Brian Thomson			Email: <a href="mailto:bat61@uakron.edu">bat61@uakron.edu</a>				
Qty.	Refdes	Part Num.	Description	Suggested Vendor	Vendor Part Num.	Catalog #/Page #/Hyperlink	Cost	Total Cost	
5	IC1	DSPIC33EP512GM706-I/PT	Microchip DSPIC33EP512GM706-I/PT, 16bit dsPIC Microcontroller	Digi-Key	DSPIC33EP512GM706-I/PT-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$6.17	\$30.85	
5	US1	VLFC4020T-2R2N1R7	FIXED IND 2.2uH 1.72A 59 MOHM	Digi-Key	445-VLFC4020T-2R2N1R7CT-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.29	\$1.45	
36	R27-R44	RC2012J222CS	2.20kOhm Resistor 0806	Digi-Key	1276-5537-1-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.12	\$4.32	
2	R49	RMCF0805JT560R	560kOhm Resistor 0805	Digi-Key	RMCF0805JT560RCT-ND	<a href="https://www.digikey.com/en/prd">https://www.digikey.com/en/prd</a>	\$0.10	\$0.20	
							<b>Total</b>	<b>\$36.82</b>	



## 23 February 2021 Parts Order Form:

DT #		4 Project Leader: Brian Thomson			Email: <a href="mailto:bat61@uakron.edu">bat61@uakron.edu</a>			
Qty.	Refdes	Part Num.	Description	Suggested Vendor	Vendor Part Num.	Catalog #/Page #/Hyperlink	Cost	Total Cost
2		HTSW-150-08-L-S-LA-006	CONN HEADER R/A 50POS 2.54MM	Digi-Key	HTSW-150-08-L-S-LA-	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$4.48	\$8.96
10		VL53L3CXV0DH1	Optical Sensor 196.85" (5m) IFC Output	Digi-Key	497-VL53L3CXV0DH1	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$3.78	\$37.84
10		WP15A4SUREQBFZGC	LED RGB CLEAR T-1 3/4 T/H	Digi-Key	754-1615-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$1.25	\$12.51
50	C1-C6	CL21B104KBCNINC	0.1uF Ceramic Capacitor 0805	Digi-Key	1276-1003-1-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.03	\$1.28
20	C7-C8	CL21A475KACLRNC	4.7uF Ceramic Capacitor 0805	Digi-Key	1276-2415-1-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.16	\$3.20
2	C9	CL21A106MQFNNE	10uF Ceramic Capacitor 0805	Digi-Key	1276-1298-1-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.11	\$0.22
2	C10	CL21A226MAQNNNE	22uF Ceramic Capacitor 0805	Digi-Key	1276-2908-1-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.35	\$0.70
2	C11	TCP1A106M8R	CAP TANT 10UF 20% 10V 0805	Digi-Key	511-1685-1-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.59	\$1.18
2	J3	DM3D-SF	CONN MICRO SD CARD PUSH-PULL	Digi-Key	HR1941CT-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$1.65	\$3.30
2	J17	473460001	CONN RCPT USB2.0 MICRO B SMD R/A	Digi-Key	WM17141CT-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.95	\$1.90
5	R1, R45	RC2012J471CS	4700hm Resistor 0805	Digi-Key	1276-5523-1-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.13	\$0.65
100	R2, R25-R26, R6	RC2012J103CS	10.0kOhm Resistor 0805	Digi-Key	1276-5552-1-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.04	\$4.41
30	R17-R24	RC2012J102CS	1.00kOhm Resistor 0805	Digi-Key	1276-5531-1-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.11	\$3.33
30	R27-R44	RC2012J222CS	2.20kOhm Resistor 0806	Digi-Key	1276-5537-1-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.12	\$3.45
3	R46	RT0805BRD072KL	2.00kOhm Resistor 0805	Digi-Key	YAG1859TR-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.36	\$1.08
5	R47-R48	RC2012F270CS	27.0Ohm Resistor 0805	Digi-Key	1276-5198-1-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.13	\$0.65
3	R50	ERA-6ABE104V	100kOhm Resistor 0805	Digi-Key	P100KDACT-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.31	\$0.93
3	S1	CT11025.0F160	Momentary Switch	Digi-Key	2449-CT11025.0F160-N	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.12	\$0.36
1	U3	TXB0104PWR	Voltage Level Translator Bidirectional 1 Circuit 4 Channel 100M	Digi-Key	296-21929-2-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.98	\$0.98
8	U1-U2	PCA9544APWR,118	I2C Multiplexer	Digi-Key	568-1861-1-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$1.89	\$15.12
5		TSW-101-08-T-D-RA	CONN HEADER R/A 2POS	Digi-Key	SAM1049-01-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.18	\$0.90
3		A700V106M006ATE055	CAP ALUM POLY 10UF 20% 6.3V SMD	Digi-Key	399-5494-2-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$2.07	\$6.21
							<b>Total</b>	<b>\$109.16</b>

## 1 March 2021 Parts Order Form:

DT #		4 Project Leader: Brian Thomson			Email: <a href="mailto:bat61@uakron.edu">bat61@uakron.edu</a>			
Qty.	Refdes	Part Num.	Description	Suggested Vendor	Vendor Part Num.	Catalog #/Page #/Hyperlink	Cost	Total Cost
4	N/A	TAJA108K010RNJ	CAP TANT 10UF 10% 10V 1206	Digi-Key	478-1654-2-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.35	\$1.40
10	R49	RT0805BRD07560KL	RES SMD 560K OHM 0.1% 1/8W 0805	Digi-Key	YAG1930TR-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.26	\$2.59
3	N/A	ECS-240-20-33-TR	CRYSTAL 24.0000MHZ 20PF SMD	Digi-Key	XC1141TR-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$1.07	\$3.21
6	N/A	C0805C300K5GAC7800	CAP CER 30PF 50V NPO 0805	Digi-Key	399-17443-2-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$0.27	\$1.62
5	N/A	M55342K06B1F00RS3	RES SMD 1M OHM 1% 0.15W 0705	Digi-Key	1135-1009-2-ND	<a href="https://www.digikey.com/en/prod">https://www.digikey.com/en/prod</a>	\$1.54	\$7.70
3	N/A	PRT-00127	MicroSD Socket	Sparkfun	PRT-00127	<a href="https://www.sparkfun.com/product">https://www.sparkfun.com/product</a>	\$3.95	\$11.85
1	N/A	Y11-3419632A	H.A.L.O. Main Board r3	JLPCPCB	Y11-3419632A	<a href="https://cart.ilpcb.com/cart">https://cart.ilpcb.com/cart</a>	\$2.00	\$2.00
1	N/A	SO2103026022-3419632	H.A.L.O. Main Board r3 Stencil	JLPCPCB	SO2103026022-3419632	<a href="https://cart.ilpcb.com/cart">https://cart.ilpcb.com/cart</a>	\$7.05	\$7.05
1	N/A	Y12-3419632A	H.A.L.O. RGB Board r3	JLPCPCB	Y12-3419632A	<a href="https://cart.ilpcb.com/cart">https://cart.ilpcb.com/cart</a>	\$4.00	\$4.00
1	N/A	SO2103026026-3419632	H.A.L.O. ToF Board r2 Stencil	JLPCPCB	SO2103026026-3419632	<a href="https://cart.ilpcb.com/cart">https://cart.ilpcb.com/cart</a>	\$7.05	\$7.05
							<b>Total</b>	<b>\$48.47</b>