Traffic Stop Watchdog

University of Central Florida
Department of Electrical and Computer Engineering

Dr. Lei Wei & Dr. Samuel Richie
EEL 4915: Senior Design 2
Final Project Document

Group 26 Authors:
Felipe Solanet *(Computer Engineering)*
Michael Gendreau *(Electrical Engineering)*
Jordi Niebla *(Electrical Engineering)*

Mentor: Dr. Chung Yong Chan

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Executive Summary

Due to recent events in the Year 2020, it has been noted that having a law enforcement job can be quite stressful. Many police officers have encountered many scenarios where they have had to react quickly and be aware of their surroundings. Whether it is a traffic stop gone wrong or getting a call about a situation where not much information is presented, a lot can occur during an interaction between a police officer and the public. There have been many cases where police officers have been accused of using excessive force to deal with a civilian, but not enough proof was presented to back up the accusations. This can lead to very expensive and long trails to investigate the situation and ensure that all the details collected are correct. With many eyewitnesses' present, sometimes incorrect information is mentioned and mixed up to give false information. Having well recorded and documented footage can be used to help both parties (police officer and civilian associated with the police officer interaction) to confirm their side of the story.

By observing and looking at recorded footage, it is easier to determine what is going on during a highly stressful situation involving police. Police officers carry body-cams that can record from a first-person view to show the field of view of the officer. Police cruisers also have dash-cams that can record in front of their vehicle. These recordings sometimes do not show everything that is going on and leave a lot of useful information out of the loop. This is where we believe the Traffic Stop Watchdog (TSW) can be put to good use to ensure that viable recorded footage is available to deeper understand what is going on during police interactions.

The TSW camera system will be elevated higher than a normal dash-cam, while mounted on the roof of a police cruiser to give a better field of view recording. With the implementation of AI, we can use image recognition to track and follow a police officer with 180-degree rotations. This will allow for high resolution and high frames per second recorded footage with the police officer always in frame. The system will also include a handheld wireless microphone device that the police officer will have with speech recognition technology. The handheld device will be responsible for enabling and disabling the system, and detecting keywords said by the officer. This will allow the police officer to quickly call for backup or execute any other command by simply repeating a keyword into the microphone.

In this report, topics will be discussed related to the implementation of the Traffic Stop Watchdog project. Hardware and software design will be heavily demonstrated to show how the project will be designed. The report will also discuss relevant technologies researched, part selection, standards, constraints, integration, testing, and administrative contents such as cost and budget.

2. Project Description

2.1 Project Narrative/Motivation

Law enforcement personnel work in a highly reactive environment that demands focus and attention to avoid life-threatening situations. The ability to rapidly respond to changes in law enforcement scenarios is imperative, and any lack of situational awareness and information capture in these critical moments inevitably leads to obfuscation of events in the ensuing judicial review. Recent events have shown that for the safety of both law enforcement personnel and the public, higher precision information is needed in this line of work.

The "Traffic Stop Watchdog" project is a design proposal to fill a current gap in the information recorded during interactions between police officers and citizens during traffic stops or other exchanges within the vicinity of a police cruiser. Current systems like body-mounted cameras and dashcams often leave much of the scene out of view or out of focus while lacking support for live data communication. This project aims to develop a vehicle-mounted information system that utilizes AI to track a moving target on camera while providing support for a wireless speech recognition device that can be carried by an officer.

Driving the overall requirements for this system are the factors associated with its use. To be vehicle mounted, the design must be lightweight and compact with a strong enough chassis to withstand vibration and acceleration. The camera electronics must be powered off a standard car battery, consuming as little current as possible when the vehicle is off so as to not drain it. The speech recognition device should be a low-power device embedded in a small form factor design so as to avoid weighing down the user. Wireless communication between the devices in the system should maintain a minimum range of at least 500 ft, preferably greater.

By utilizing machine vision algorithms, the system should be capable of resolving more important information in a scene than current cameras. The onboard camera will track the law enforcement officer so as to record only the information vital to the scenario. 180 degrees of rotation and high placement on the vehicle will allow tracking in all directions. To maintain track of a fast moving target, the system will need high responsiveness and an accurate control loop keeping the camera centered. This will necessitate a blend of precision electromechanical design as well as robust software development.

2.2 Project Objectives

As mentioned previously in the project narrative, the objective of this project is to ensure that the police officer has a watchdog system that can track their movements and listen to the keywords said by the officer. To accomplish this, we must make sure the system is easy to interface with and allow a quick and effective AI algorithm to track the police

officer. We can meet these objectives and requirements outlined in Section 2.3 by applying the knowledge and technical skills we have learned throughout the many courses at UCF. By working together as a group of three, we can combine the skills learned throughout our ECE program to come up with a working prototype.

2.3 Project Specifications

The requirements and specifications for the project can be broken down into the software requirements and hardware requirements. These must be met in order for the project to be deemed satisfactory. All design goals and development efforts will be aimed at meeting and exceeding these requirements. This outline will also serve as a checklist for the development process.

2.3.1 Software Requirements

- Trained object detection model for police officers
- Low response times on image inferences (< 500ms)
- Camera footage recorded (32 GB max storage)
- Keyword recognition for microphone
- ZigBee Radio communication system (500 feet range)
- Accurate camera response movements
- Quick and effective officer-system interface

2.3.2 Hardware Requirements

- 180 degree pan angle, 180 degree tilt angle
- <1 degree angular resolution on both axes
- Weight of camera & mount < 7 lb
- 30-60 FPS at 720p
- Complete camera system with pan and tilt mount should be powered through the cars 12V battery
- Low Power Consumption (< 1.5 A @ 12V)
- Fast & accurate target acquisition
- Environmental feedback (Thermal Monitoring; Over-Voltage, Over-Current Circuitry; Telemetry Logging)
- Hand held microphone device should not exceed a X x X inches (Length x Width)
- Wireless communication between Xbee on microphone and Xbee on Odroid
- Microphone device should be powered by rechargeable Li-Po batteries
- Low Battery Power Consumption (<200mA on 3.6V LiPo)
- Microphone device should include an LCD screen & push buttons
- Microphone device should activate/deactivate Traffic Stop Watchdog system
- The cost to design and create the project should be less than $1150

2.4 House of Quality Analysis

In all engineering designs there are always trade-offs. It is important to understand what the main focus of your project is, and who is being targeted for use of the finished design. In our case, the Traffic Stop Watchdog design is implemented for the use of police officers. This means that the features implemented in the project should be features that will better the experience of a police officer while using our product. While we want to ensure that our design reaches every requirement set, we understand that there will be some trade-offs involved in our design.

Business or marketing requirements can be seen as requirements that any future client or consumer (police officer in our case) would like to have. Some obvious requirements that a client would like for an engineering product are low cost, long battery life for an electronic device, durability, light weight, or small size design if handheld or portable, and easy to use. These are requirements that most successful designs strive to reach for. As mentioned previously, there are many trade-offs that occur in the process of designing a product. Designs that are more efficient and usable for all consumers seem to be higher in price due to their better functionality. The size of the device can also affect how many components you can have on your board, and also the battery life. This section illustrates a house of quality table (Table 2.1) that details the direct correlation and tradeoffs between engineering requirements and business or marketing requirements.

Engineering requirements are the requirements needed for the Traffic Stop Watchdog system to be marketable. These requirements have targets that are set by us, the creators of the design. One of the toughest challenges coming from the customer requirements is getting a fast-enough image detection model to both accurately and quickly pinpoint the officer’s location on each image. Lower quality images being fed to the model will lead to faster inference times, but lessen the effectiveness of the recorded footage. The camera motion must also be swift enough to not lose sight of the officer, but smooth enough to not blur the incoming images too much.

Table 2.1 House of Quality Diagram

			Engineering Requirements						
			Power Supply	Camera Response Time	Time To Implement	Device Communication	Dimensions	Weight	Cost
			+	-	-	+	-	-	-
Business Requirements	Officer Footage	+		↓↓	↑↑				↑
	User Friendly	+			↑	↑	↓↓	↓↓	
	Durable	+	↓		↑↑		↑	↑	↑↑
	Install Ease	+	↓			↓	↓	↓	
	System Integration	+	↑		↑	↑↑	↓	↓	↑
	Cost	-	↑		↓	↑	↓	↓	↑↑
			12V Car Battery	< 500 ms	< Two Weeks	> 500 feet range	Within 12 x 24 x 12 (LWH)	< 7 Lbs	< $1200
			Engineering Targets						

Legend	
+	Positive Polarity
-	Negative Polarity
↑	Positive Correlation
↓	Negative Correlation
↑↑	Strong Positive Correlation
↓↓	Strong Negative Correlation

3. Research

3.1 Relevant Technologies

In this subsection, relevant technologies that play a big role in the TSW project development will be discussed. Speech and image recognition are both artificial intelligence driven technologies that can be quite complex. They are becoming more common in the embedded world, and in the implementation of modern technology. We will have to be familiar with both of these technologies. Image detection is one of the most important features of the camera system, and voice recognition for the handheld device. There will have to be a lot of programming involved to ensure that these systems can communicate and do their job efficiently; therefore, embedded computation via a microcontroller, programmable logic via FPGA, and object-oriented programming must also be familiar topics.

3.1.1 Image Recognition

In the field of robotics and embedded applications, cameras act as the eyes for the system a majority of the time. With the use of image recognition, embedded systems can respond to visual events that occur in their environment in real-time. Computer vision (the name given to the field of research that studies this problem) has been trying for decades to optimize image recognition and make it more adaptable to different kinds of applications. There are many different ways to go about recognizing different shapes and patterns in an image. The most common is deep learning using neural networks. Both deep learning and neural networks are each large fields of research on their own, but mold very well to the image recognition problem.

A large number of problems that computers are being used to solve today involve some kind of neural network. Their flexibility and ability to vaguely resemble how the human brain operates allows them to solve problems once believed were only solvable by humans. Neural networks can be thought of as a screening for a particular role in a film or job. They involve taking an input (or series of inputs) and asking them various different questions. Based on the answers to those questions, the input will travel along with its answers to another series of questions. The idea is that after repeating this process a few times, a prediction can be made about the inputs. Depending on the desired type of prediction, certain questions would be asked and phrased in specific ways to extract key pieces of information relevant to the prediction. In the context of a neural networks computer science problem, these questions would be represented by equations, filters, or combinations of the two. The answers would simply be the input evaluated against a particular equation/filter. Each equation and filter aims at highlighting key features in the input that would suggest a certain output. Image recognition takes advantage of a special type of neural network called a convolutional neural network, which takes in images and performs a convolution on these images with specific filters aimed at highlighting different features in the image that would point towards what the image is describing.

Neural networks alone lack a key element required in order to solve most problems. While neural networks are great at extracting features from inputs, they have no sense of how important and relevant each feature is when predicting an output. This is where deep learning comes in. The goal of deep learning is to replicate the way in which humans learn things inside a computer. In some cases, people can be told exactly what to make of a situation based on a few rules/observations. These cases are easily programmable in a computer, and all that computers historically were able to accomplish. Not all cases are this simple, however. In fact, a good number of cases cannot be easily defined by a small set of rules/observations, and most people have to observe the situation multiple times, each time learning more about what may or may not have caused the outcome. Each time the person observes the situation, he/she will have an idea about what may cause the outcome. Most of the time, especially if the situation has never/rarely been seen before, the person will be wrong about the outcome. When a person is wrong, he/she can learn from the situation and adapt so that next time the same or a very similar situation is posed, the person will be able to make a more accurate prediction than the last time.

Going back to deep learning, a neural network can use this idea to adapt to situations caused by certain inputs. This is achieved by applying weights to each feature that the neural network extracts. The more weight a feature has, the more it will affect the output. Determining the correct weights to apply to each feature is the job of deep learning. Initially, random weights are applied and the neural network is tested against cases with known outputs. Based on how close the predicted output was to the actual output, the weights will be adjusted. This is repeated multiple times in a process known as “training” until the weights converge and the neural network can make an accurate prediction in most cases. There are tons of different algorithms that determine how each weight should be adjusted, but they all revolve around a loss function and gradient descent. Every deep learning algorithm defines its own loss function, which is a function that takes in the predicted output and actual output and determines how wrong the network was. This function can be as simple as subtracting the two values and as complex as involving intense calculus and differential equations, and it is up to the algorithm to decide which function works best with it. After a loss is determined, gradient is formed as the combination of the magnitude of the loss and the direction. This gradient then propagates backwards and adjusts each of the weights.

Within image recognition lie two categories: image classification and object detection. Image classification aims at classifying an image given a finite set of possibilities. Object detection, on the other hand, is concerned with finding predefined objects within an image (e.g. cat, person, house, etc). This can be used to specify a location within the image where the object is predicted to be. Multiple objects of the same or different type can be detected in the same image. Image classification and object detection each have many different algorithms that specify how the neural network is shaped and how the network should change over time to create a model that can perform image recognition on new images with as much accuracy and in as little time as possible.

3.1.2 Embedded Computation

Embedded systems have become quite popular in the electronic world. They can be seen everywhere in the technology industry, whether it is industrial, consumer, medical, commercial, and many other applications. An embedded system can be defined as a microprocessor based computer hardware system that can be controlled through software. The hardware and software design can be designed simultaneously. They rely on microcontrollers, microprocessors, digital signal processors, and other designed processors to do the heavy computations. Embedded systems firmware is stored in ROM (read only memory) or flash memory chips and run with computer hardware resources. Embedded systems have many peripherals that can be used to connect and interact with other devices. Some of the common peripherals in a microcontroller can include GPIO's, timers, USB, ADC/DAC and other serial communication methods such as I2C, UART, and SPI. The GPIO (general purpose input/output) pins can be configured as either input or outputs and is up to the programmer to decide.

A microcontroller can be programmed through a compiler that will convert your C code, or whatever language you decide you use, into machine code. By using an IDE such as CCS (Code Composer Studio), an optimized C compiler can be used along with a source code editor, debugger, and other features. The code written will be compiled and flashed to the microcontroller by the help of JTAG (Joint Task Action Group) interface. JTAG is the hardware on the microcontroller's launchpad that allows programming of the chip and allows the compiler to debug. CCS is just an example of one IDE, as it is usually used for TI's microcontrollers.

3.1.3 Speech Recognition

Speech recognition is one key part of the Traffic Stop Watchdog implementation. Speech recognition allows a machine or program to receive and understand verbal commands coming from a user. It is a technology that has been heavily implemented in many devices. Nowadays we can examine how popular devices like smartphones, Alexa, and Google Home can incorporate speech recognition to execute commands. If you have an IPhone, you can simply send a voice command "Hey Siri", and your smartphone will instantly detect it. It will recognize the communication that you are sending its way, and can do simple tasks like set a reminder on your phone or look something up on the Internet when told to. Google Home and Alexa can even receive voice commands to communicate with other devices connected to them. With a simple command, you can tell your device to turn on a smart-lightbulb, or play music on a different device.

The way that the typical speech recognition module works is that analog audio from the user is converted into digital signals through an analog to digital converter. With the use of Fast Fourier Transform (FFT), the digital data can then be analyzed and converted into a spectrogram. The spectrogram is a graph that represents the signal strength over time as different frequencies. The digital data then is broken down into acoustic frames which are then compared to a list of known words/phrases. By comparing the acoustic frames

to the saved words/phrases, then the speech recognition module can determine and compare the words being announced [3.2].

Speech recognition modules or programs are usually evaluated due to their accuracy and their speed. A high accuracy speech recognition module is usually at the 90% or more success rate. Having high accuracy is important because you would not like the wrong command or spoken words to be picked up by the module. In the case of the police officer using the embedded microphone device it would not be ideal for a K9 unit to be called if that command was not said. This could lead to many issues and miscommunication, so it is important to deal with a high accuracy speech recognition module. The speed of the voice recognition module is also something very crucial. If the police officer is in a sticky situation and needs urgent backup, the amount of time the whole process takes to listen and follow commands is crucial.

Voice recognition is a topic that usually gets confused with speech recognition. They are usually terms that are interchanged, but both are used for different functionalities. Speech recognition is used mostly to detect spoken words and language, to then convert it into text. Voice recognition is more focused upon verifying the identity of the speaker. Voice recognition can identify your voice by the pitch, accent, or speaking style. If multiple people are speaking, the voice recognition module would be in charge of detecting the voice of a specific person. So, for the application of our project we can see how the speech recognition module is more ideal. The words repeated by the officer should be heard and acted upon quickly.

3.1.4 Programmable Logic

For many embedded applications, a microcontroller is a perfect solution for implementing programmable logic using sequential code. The standard peripherals available for many microcontrollers make them easy to interface with and control. Microcontrollers are small in footprint and require few external devices to boot up. However, for applications involving many parallel logical operations, large numbers of inputs and outputs, and the need for complete control over the logic fabric of the design, a Field-Programmable Gate Array (FPGA) is the obvious choice.

FPGAs are integrated circuits that allow the engineer to completely reorganize their internal logic elements. This freedom permits almost any conceivable logic implementation that will fit on a given piece of hardware. For high-speed interfaces that constantly update low-level memory elements for movement of precision mechanical components, an FPGA is a smart choice. That’s why they’re frequently found in high-end servo motor drivers used in Computer Numerical Control (CNC) machines. The ability to shuffle and manipulate data rapidly and synchronously in the digital domain is a major advantage of FPGA technology.

3.1.5 Object-Oriented Programming

At the core of many industrial applications and systems lies object oriented programming. Its modularity and intuitive ideas allow it to be easily implemented in projects across many different fields. Object-Oriented programming (OOP) is a software design philosophy that puts forth a set of guidelines for designing the architecture of an application. It specifies that everything can be modeled as an object that has its own state and exhibits a specific behavior. Objects can extend one another to further expand the capabilities of the base object. This idea allows sections of the code to be abstracted from one another, making the code more organized and easier to use and maintain across multiple projects. As a whole, OOP allows one piece of code to fully interact with another piece of code without much knowledge of how the code works internally.

Generally, an object is designed through the use of a class, and its state and behavior is modeled through using variables within the class called properties and behavior is modeled with functions in the class called methods. Many different languages have built in support for OOP. One of the classic OOP languages is Java. Java was created to provide a cross platform industry standard language to be used for creating object-oriented applications. This language is still used widely today, however Microsoft's C# has taken over a large portion of the object-oriented world. It's compatibility with Windows and immense support made it the language of choice for many business and engineering applications.

While Java and C# are very useful and powerful object-oriented languages, this project will utilize C++ for the bulk of its programming. C++ has object-oriented features like classes built-in to the language, but it takes a different approach to how its runtime is handled. C# and Java create virtual environments on top of the system that provide convenient features such as managed memory that make programmer's lives a lot easier. C++ runs directly on the system and it doesn't have as many features. Technically, C++ does have some small forms of these features that are implemented at compile time instead of runtime, but in general, the language is more bare bones. This makes the language very portable and can run on most devices. The downfalls of not having a virtual environment between the code and the system can also be seen as benefits in some cases. Because the memory is managed by the programer, access to direct memory addresses and manual memory allocations are allowed, which when used correctly, can allow efficient loading of big data but can put the system at risk if used incorrectly.

Each component of the software can be broken down into a model (or set of models) to abstract its internal operations from the other sections that don't necessarily need to know all of the details about how everything works. This makes the code more modular and easier to read and integrate into other parts of the project. Abstraction also opens the door for code reusability in sections that need to perform the same/similar task as others. Testing and tweaking the main application will be much more manageable, since every logical/functional block of code will be organized in a way that makes it easy to hone down spots that can be adjusted to better the product.

3.2 Existing Solutions

Through the use of pre-existing technologies and solutions, the project can be developed in a much shorter time frame. Many of the concepts and ideas that this project requires have already been designed and implemented. There are some cases where recreating something allows for project specific optimization, but should be considered carefully before doing. If there is no real benefit from redesigning and recreating what is already out there, it should be avoided to minimize errors that could come up from the added work. Proper usage of these technologies will provide a strong base for design and development and allow more focus to be dedicated to the core features of the project.

An example of a pre-existing solution is the Arducam bundled with the Nvidia Jetson development board. These devices are an off-the-shelf solution (when bundled with the right SDK) for machine vision object tracking using a small pan and tilt mount. Other examples of this pairing, a small but powerful single-board computer (SBC) and a lightweight servo-based pan and tilt mount, are numerous and inexpensive, frequently associated with Raspberry Pi hardware. However, these off-the-shelf solutions don't satisfy the requirements of our design. In these examples, image quality and frame rate are not priorities, so the camera designs can be small but low quality. They're not immediately capable of receiving 12VDC power, and none of these devices are mechanically suited for mounting to vehicles. Moreover, no interfaces are provided for speech recognition by a microphone.

3.3 Parts Selection

This section will detail various components that are compared and chosen for the use of the TSW project. The reason for picking certain components will be discussed, along with their functionality and how they will affect the project design.

3.3.1 Electromechanical

The electromechanical components of this design include motors for movement in two axes as well as the gearing and associated driver circuitry in the camera assembly. The motor technology choice is at the heart of this section of the design - the solution needed must be compact, precisely actuated, and provide enough torque without drawing too much power. To achieve this, our design weighed the options of brushed DC gear motors and stepper motors.

Brushed DC gear motors achieve high torque with a simple, robust power/control interface. Driving them typically involves a pulse-width modulated control signal applied to a power transistor, allowing speed and torque control in proportion to the pulse width. However, in order to precisely control these types of motors, angle or rate feedback in the form of an encoder is required. An accurate encoder adds substantial cost and complexity, and the overall footprint of many brushed DC motors with an encoder attached is far too large for the dimensional constraints of the camera assembly.

Stepper motors were the most cost effective solution, winning over the gear motor solution due to their precision control and compact size. Because stepper motors can be controlled “open-loop” without additional sensors, their size and cost are kept small. Additionally, the rise of mass-market 3D printer technology has pushed the price of stepper motors lower. The only downsides of stepper motors are that they require power to “hold” a position and the control logic to commutate them is more complex. To drive a multi-phase motor, a commutation sequence is required on the inputs to advance the shaft angle. Luckily, integrated circuits exist which perform the commutation from internal look-up tables, directly powering the motors and reducing the complexity of the required controller. For this purpose, the Texas Instruments DRV8846 motor drivers were chosen.

3.3.2 Microcontroller & Field-Programmable Gate Array (FPGA)

Embedded logic in our design takes the form of microcontrollers and FPGAs. The primary communication bus, using UART, needs multi-slave capacity, which is implemented via a small FPGA. The camera assembly utilizes an FPGA as the motor controller, converting angular inputs over RS232 into driver signals for the motors. The choice for a Xilinx product was made for the camera motor logic module. Their development environment and debug/simulation interface is free for hobbyist use, and their products are equipped with more than enough logic elements for the hardware design needed for motor control. Additionally, if processing/filtering is needed on the angular error inputs received from the SBC, the Xilinx chips have DSP slices that boost digital signal processing performance. This allows implementation of PID control in the logic fabric of the FPGA.

The factors involved with choosing the specific Xilinx FPGA chip for our design included cost, package type, and peripherals. FPGAs can be very expensive devices, so the lower-end Spartan 7 lineup was our first choice. The 7 Series Spartan FPGA chosen is the XC7S6-1FTGB196C, which includes 100 I/O pins and 6000 logic elements in a 1.0 mm pitch Ball Grid Array (BGA) package. Routing a BGA package IC on a PCB is typically quite a challenge, but this is mitigated by the use of a 4-layer board and the minimum feature size for vias and traces provided by JLCPCB, our PCB vendor. To store configuration data, a Xilinx-compatible 32kb flash SPI chip is also needed, and an external 12 MHz oscillator provides the system clock for the Spartan 7.

For the UART bus manager FPGA, a Lattice Semiconductor device was chosen. Lattice specializes in small-scale FPGAs for single-purpose embedded logic. The design calls for a simple hardware configuration for buffering, receiving, and sending UART messages from different devices depending on message priority and address. The specific device chosen was a Lattice MachXO2-1200, which packs 1280 logic elements in a 32 pin quad-flatpack no-leads (QFNS) package. In this role, the Lattice is superior to Xilinx and Altera products because it requires few external components and remains cheap and small-size. The MachXO2 has an onboard clock and nonvolatile configuration memory, which shrinks its overall footprint on the space-critical PCB layout.
On the microcontroller end, our requirements are for a low-power, multi-peripheral MCU that controls various functions of a handheld microphone device. This will include a display, a microphone module, and power monitoring. Our options included the STM32

lineup of 32-bit MCUs, and the Texas Instruments MSP430 family. While the other options are much more powerful ARM architecture chips, the MSP430 wins out on cost-effectiveness and low power consumption. The peripherals needed are found on most MSP430 MCUs. Specifically, we looked for hardware UART, multi-channel ADCs, and enough IO to drive a display's parallel interface. Table 3.1 lists and compares some of the MCU's features.

Table 3.1: Comparison of MCU's

	MSP430FR6989	STM32F100R4
Max Clock Frequency	16MHz	24MHz
Program Memory Size	128KB FRAM	16KB Flash
RAM Size	2KB	4KB
Core Size	16-bits	32-bits
Core Processor	CPUXV2	ARM Cortex-M3
16-bit Timers	5	Up to 12
UART Interfaces	2 UART	3 USART
Number of I/O	83	51
Voltage Supply	1.8V – 3.6V	2V – 3.6V
Price	$3.30	$4.15

MSP430FR6989

One of the main reasons the MSP430FR6989 is our leading candidate is because of the cost-effectiveness and low power consumption that the MSP430FR6989 has. It is simple to use, and throughout the UCF curriculum it is a MCU that was popular for use of embedded programming. It runs off a 16-bit RISC architecture with a clock of up to 16-MHz. The MSP430FR6989 uses 128KB of FRAM, which is a non-volatile memory that uses ferroelectric thin film capacitors to store data. This allows an increase in performance at lowered energy budgets.

The microphone device will communicate via UART, and we believe that the UART peripherals that are offered from this MCU are great for the use of our project. The eUSCI (enhanced Universal Serial Communications Interface) module on the MSP can include the two channels, (channel A & B), which channel A supports UART and SPI communication and channel B supports I2C and SPI. There are multiple eUSCI modules that allow independent communication channels which are USCI_A0 and USCI_A1. USCI_A1 is the backchannel UART that allows UART communication between the USB host, which is very useful during development. This allows a communication channel to the PC host side and allows us to communicate with a serial port terminal. The USCI_A0 UART channel uses the 20-booster pack [3.1].

The STM32F100R4 is another great MCU that stood out while researching MCU options to use for the implementation of the microphone device. It has a 32-bit ARM Cortex-M3 processor with a 24MHz maximum frequency. It has about 16 Kbytes of Flash memory, and 4Kbytes of SRAM. When comparing this MCU to the MSP430FR6989, it can be seen that it outperforms it in some categories. With a 32-bit core size, and 24MHz frequency, it is much more powerful than the MSP430's 16-bit core size and 16MHz frequency. However, for the implementation of the handheld microphone device, we decided that the specs and performance of the MSP is all that was necessary. It is low power and easy to use, and the group has experience with this MCU.

3.3.3 Single Board Computer

In charge of running the main logic that controls the entire system, the single-board computer needed to be capable of communicating to multiple devices simultaneously and processing/recording frames at a respectable rate. The operating system requirement set forth by the use of the Flir Firefly DL limited the choice of CPUs that could be used. Offloading the actual object detection work to the camera allowed for the single-board computer to be more dedicated to recording and communicating with the devices. The Odroid XU4 was chosen to take on this responsibility. Its 5V DC power requirement could easily be supplied from the 12V car battery. It comes with a small fan over the processor to help cool the chip. Running Ubuntu 16.04.3 Mate, the scheduling and file IO was already set by the OS and could be utilized in the overall logic of the system.

With two USB 3.0 ports and one USB 2.0 port, the Odroid met the requirements for communicating with multiple devices. Unfortunately, the enclosure built for the Odroid

blocked off the USB 2.0 port, so an adapter was created to allow two USB connections in one of the ports that will be discussed in a later section. Equipped with a quad-core ARM Cortex-A15 2GHz and a quad-core ARM Cortex-A7 1.3GHz CPU, the Odroid has the speed and multicore architecture required for fast response times and multi-device communications. A 64GB micro-SD card was used to hold the operating system and store a sufficient amount of footage. A convenient HDMI port and Ethernet port on the Odroid was used for debugging and setup purposes.

3.3.4 Communication Protocols

UART: Universal Asynchronous Receiver/Transmitter, also known as UART, is an interface that allows transmitting bytes between two devices. The transmitting UART is responsible for converting parallel data from a device into serial form, then transmit the serial data to another receiving UART device which will then read the serial data and convert it into parallel data. UART can be implemented with one wire so that data is transmitted in one direction (half duplex) or with two wires so that two-way simultaneous transmission will be occurring (full duplex). Two devices communicating with UART will have their Tx pin (Transmit) connected to Rx pin (Receive) [3.4].

Since UART is asynchronous, the transmitter and the receiver each have their own clock signal. This means that the rising and falling edges will not always occur at the same time. The UART transmission uses a least significant bit (LSB) and most significant bit (MSB), along with the Start, Stop, and parity bit. These bits are used to define the beginning and end of the data and allows the receiving UART to know when to start and stop reading bits. The transmission ,as shown in Figure 3.1, is as follows: The signal is idle at high, and then signal will drop to low for a one-bit duration to signal the start bit. The receiver is waiting for the start bit occurrence, and when it occurs data will begin to transmit bit by bit starting with the least significant bit. The transmission of the bits will continue until the stop bit, that has a value of high, is reached. When the receiver detects the start bit, the incoming bits are being transmitted at a frequency defined as the baud rate. This baud rate is the bit duration and is the transmitter's clock rate. A standard baud rate is 9600 baud, which corresponds to a bit lasting about 1/9600 seconds [3.5].

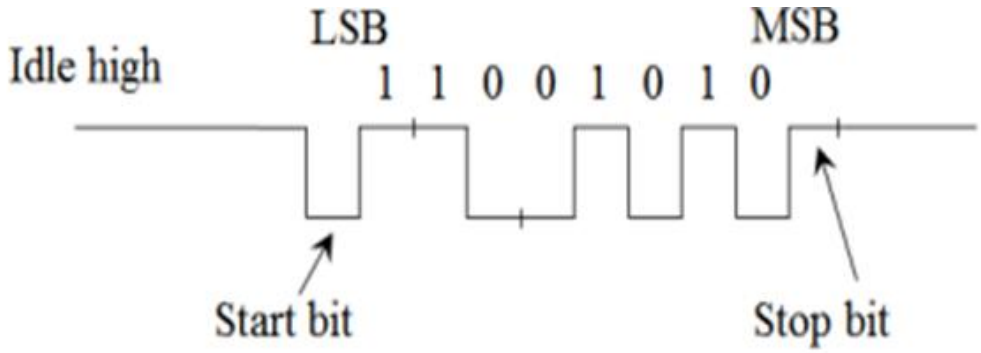


Figure 3.1: UART Transmission for a data byte. Permission obtained from Dr. Abichar

While everything has its advantages and disadvantages, let's discuss UARTs. The simplicity of UART can be very beneficial, since only two wires are used to connect the MSP430 and the microphone device. Parity bits can be used to detect if there was an error during transmission. Since the transmitter is operating on the baud rate, we do not have to worry about setting up other clocks for our receiver, this makes it very simple. The disadvantage to UART is that both devices must operate at the same baud rate (or very close), and the data bits are limited to a maximum of 9 bits. However, we believe that the advantages outweigh the disadvantages, and have picked UART to be the preferred method of communication. The speech recognition module used to communicate with the MSP430FR6989 has UART hardware implemented in it and will allow smooth communication between the two devices.

SPI: Serial Peripheral Interface (SPI) is a communication protocol that allows a main or master device to communicate with one or multiple slaves. An example can be a microcontroller communicating with a liquid crystal display or a sensor. The protocol is synchronous since the master generates a clock signal to transmit to the slave. There can only be one master device, but there can be one or more slave devices. Two data wires are used between the master and slave devices; therefore, SPI is considered to be a full-duplex protocol [3.5].

Typically, in the SPI serial protocol there are four wires used to communicate between devices. The transmission is illustrated below in Figure 3.2. The first wire is for the SCLK (serial clock) which is responsible to carry the clock signal from the master device to one or more devices on the serial bus. Next is the MOSI (master output, slave input), which carries the data from the output of the master device to the input of the slave device. After MOSI, there is the MISO (master input, slave output) wire that carries the data from the output of the slave device to the input of the master device. If multiple slave devices are being used in the design, then a SS (slave select) must be used. The slave select wire, as the name gives away, is used to allow the master device to select which slave device it will be communicating with. The master can exchange data with all the slaves, and vice versa, but the slaves cannot exchange data to each other [3.6].

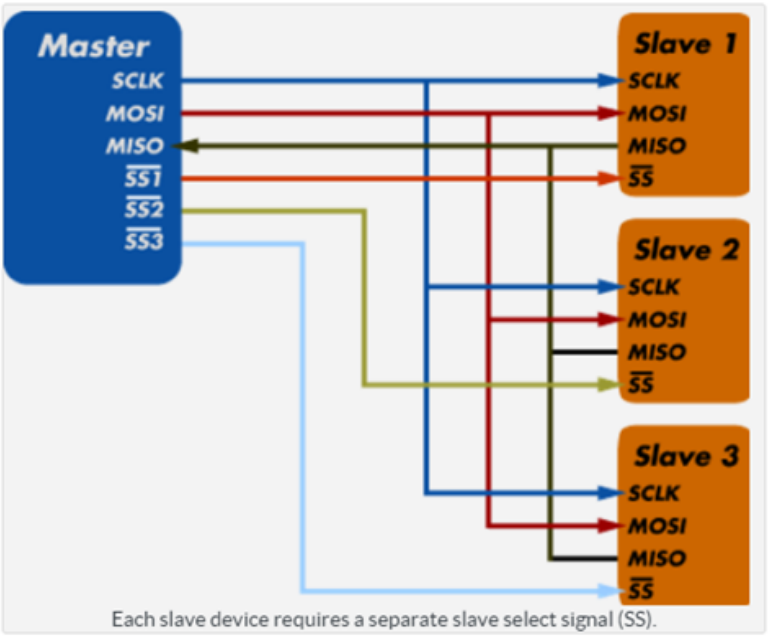


Figure 3.2 SPI communication with multiple slaves, permission requested and granted from www.totalphase.com

Table 3.2: Four modes of SPI operation

Mode	Clock Polarity	Trigger Latch	Communication
0	Idle at low	Latch at trailing edge	Communicate at leading edge
1	Idle at low	Latch at leading edge	Communicate at trailing edge
2	Idle at high	Latch at trailing edge	Communicate at leading edge
3	Idle at high	Latch at leading edge	Communicate at trailing edge

There are four modes of operations for SPI which is determined by the clock signal or clock polarity and also what edge the trigger is latched on to allow the communication action (clock phase) [3.5]. These four modes of operation are shown in Table 3.2.

When reading about SPI, we notice that there are a lot of advantages and some disadvantages. There are not many limitations on speed, making it faster than I^2C and UART, other than the limitations from the clock frequency and rise/fall time. There is full-duplex communication, similarly to UART. SPI can also be useful in applications where the master needs to communicate with multiple slaves.

Some disadvantages can include more pins and wires are required to communicate between devices, with four wires needed if there are more than one slave devices. SPI is also used only for short distance applications. While UART has parity bits to detect errors, there are no error detection protocols in SPI. Taking all these advantages and disadvantages, UART seems to be a more effective method to use for our embedded microphone to MCU communication. While the embedded microphone device only communicates to the MSP430FR6989, we do not need multiple slave connections to the master device.

I^2C: I^2C is another great serial communication protocol that is widely used across different applications. I^2C allows you to connect one or multiple slaves to a master device, with the ability to all be controlled by the master. Not only can you have multiple slaves connected on the bus line, but you can also have multiple masters. Although the ability to have multiple slave devices connected with I^2C is great, you must ensure that each slave device has a separate address (typically 7 bits) so that the address is being sent and received by the correct devices.

I^2C is synchronous, which means that a clock signal is generated by the master to transmit data to the slave. I^2C consists of the following two lines/wires: serial data (SDA) and the serial clock (SCL). I^2C is a half-duplex protocol; therefore, the SDA wire is used for sending and receiving data, bit by bit, between the master and slave. The SCL line carries the clock signal driven by the master device. Transmission on both lines are initiated by the master device. Like UART, the beginning and end of transmission is determined by start and stop signals. Figure 3.3 shows how data is transferred in messages. The start condition or signal is the first step in the I^2C communication protocol. This happens when the SDA line is switched from high voltage to low voltage right before the SCL line is switched from high to low. The stop condition is met when the SDA line is switched from low to high voltage right after the SCL line is switched from low to high. Along with the start and stop signals, there is an address frame, read/write bit, and ack/nack bit. The read/write bit is a single bit used to determine whether SDA line voltage is high or low. If the SDA voltage line is high, then the master is requesting data from the slave, and the line is low if the master is sending data to the slave. [3.7].

The address frame is the first frame after the start bit in a new message. The address frames are responsible for storing the 7-bit or 10-bit sequence for each slave, which allows the master to directly identify the proper slave and communicate with the slave. This is important because it ensures that if the proper data is being sent to the correct

address, then there will not be another slave intercepting the data on the SDA bus wire. The method of addressing is that the master will send the address of a specific slave that the master is interested in communicating with, and the slaves will read the address and compare it to their own. If the address is recognized to be of the slave, the slave will send a low voltage acknowledge bit (ACK) back to the master, signaling that the address frame was properly received. If the address is not recognized, then the slave will not communicate, and the SDA line voltage will remain high. When the slave sends an ACK bit back to the master device, the data frame will be ready to be sent. The length of a data frame is 8 bits, and when sent there should be an ACK/NACK bit followed. Whoever sends the data frame should receive the ACK/NACK bit, and the next data frame can only be sent after an ACK bit is received. The stop bit is the final step in the message [3.7]. I^2C will be used to program the LCD screen for the microphone device.

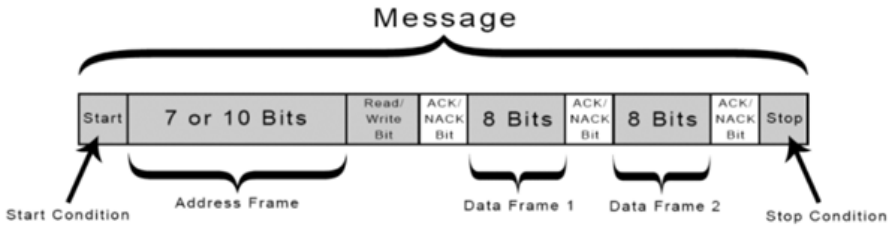


Figure 3.3: I^2C data transferred as a message. Permission for reproduction was received from circuitbasics.com

3.3.5 Embedded Microphone

One of the unique features of the “Traffic Stop Watchdog” design is the use of an embedded microphone. The embedded microphone will not only work as a microphone, but it will be the device in charge of activating and deactivating the whole watchdog system. The microphone device will have a user-interface, that can include an LED screen and a couple of push buttons. The police officer can start/stop the cameras from the microphone device to start its image classification and detection. The LED screen can be helpful for feedback and identifying the state of the system to ensure that the recording of the cameras is on.

The microphone unit should consist of an MCU chip communicating directly with a speech recognition module to execute commands when a keyword is recognized. An MCU that is low cost, low power, and can carry out UART communication is key for the design. Since the MCU will communicate with the speech recognition module, we will need to find one that can both communicate with UART and ensure that the communication is at a similar baud rate. As mentioned in section 3.3.2, the MSP430FR6989 will be the MCU in charge of communicating with the speech recognition module to create an embedded microphone.

The speech recognition module utilized for our design can store five words in a group. There are three groups that can be recorded, allowing us to store 15 recognizable words in the speech recognition module. The speech recognition module communicates

through UART and can be configured by sending commands through serial port interface. The words can be recorded through a serial port terminal application, such as CoolTerm. To begin recording we can send hex strings to the module to send a command. A hex string in the command format of “Head+Key” where the head is "0xAA” must be sent to communicate. The key is in hex format and changes depending on the command you would like to send. For example, if we would like to send an instruction to begin recording the keywords for group 1, the key is “0x11”. Therefore, to begin the process of recording the keywords for group 1, we must send a string in the hex format of “0xAA11”. After recording our keywords for group 1, we must then import group 1 so that the words are saved. This can be done by sending the string “0xAA21” where the new key is now “00x21”.

To test if the keywords are recognizable, the words can be repeated to the microphone that is connected to the speech recognition module after importing group 1. The five words that were recorded would yield a “Result: x” when the microphone detects the word. The first word will yield “Result: 11”, second word “Result: 12” and until the fifth word that will be “Result: 15”. Figure 3.4 demonstrates the simple communication that occurs between the MSP430, and the speech recognition module. The MSP430 is sending the “Head+Key” which is a hex string in the form of “0xAA21”. The speech recognition module will then receive the audio and determine if a keyword is recognized. In this case, the fourth key word is mentioned, therefore “Result:14” will be sent to the MSP430, and received.

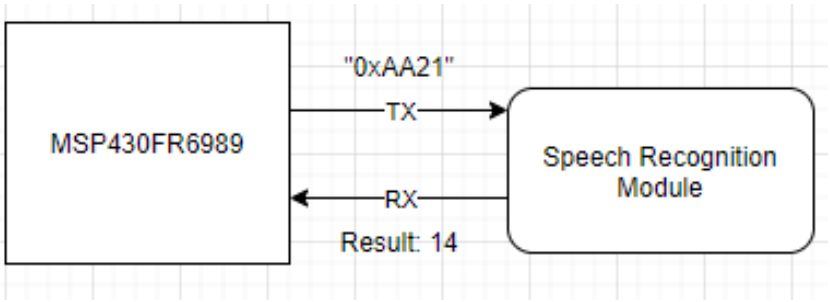


Figure 3.4: UART communication between MSP & speech recognition module when keyword #4 is repeated

By using the Analog Discovery 2, given to us by the UCF ECE department, we can read the data being sent from the transmit pin of the speech recognition module to the RX pin of the MSP430. When sending the string “0xAA21” to import group 1, the speech recognition module will begin to listen for the key words. As mentioned previously, if the word is recognized a result will be sent back. To analyze and ensure that there is data being spit out from the speech recognition module, we can read the TX line of the speech recognition module. By setting a trigger on the falling edge of the TX pin for the speech recognition module, the start bit will be signaled when the signal drops to low for a one-bit duration. This will signal the beginning of the transmission. In Figure 3.5, we can see

how “Result:14” is being transmitted by the speech recognition module. This is due to the fourth word in the group being sent to the microphone.

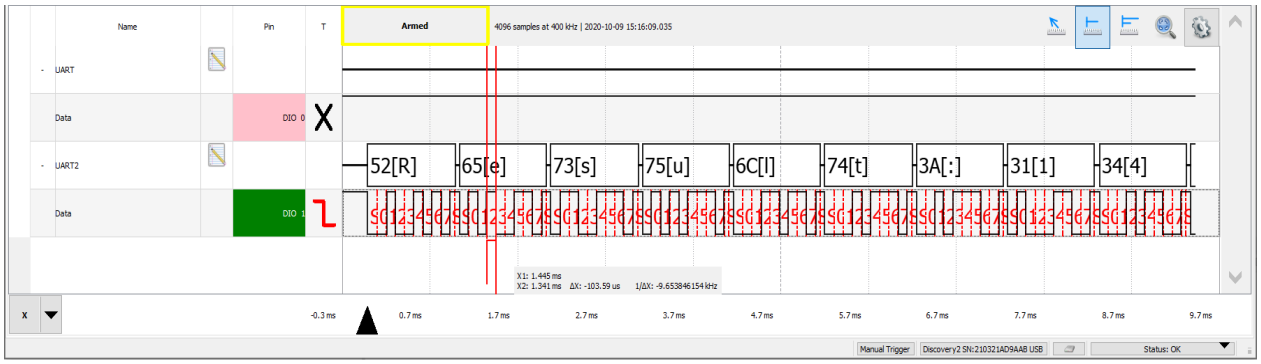


Figure 3.5. Embedded microphone transmitting bytes and recognizing word four in group 1. Illustration by J.Niebla

From Fig 3.5, it can be observed that the displacement in time (Δx) in each bit is about 103.59us. As mentioned in section 3.3.4, the baud rate is the rate at which data is being transferred bit by bit in the UART channel. The baud rate can also be found in terms of frequency, which can be found from 1/Δx. The logic analyzer has identified the baud rate to be about 9654 baud or 9.654kHz. This is a standard baud rate and is set by the MCU and the speech recognition module.

3.3.6 Camera

There are a few big requirements that needed to be met on the camera that was selected for this project. It needed to be capable of recording at a decent quality and framerate in order to provide satisfactory footage of the scene. We set the initial benchmark at 720p resolution at 30 FPS. This would put the minimum bandwidth requirement at:

$$720\ X\ 1280\ pixels\ *\ 30\ frames/second\ *\ 3\ B/pixel = 79.1\ MB/second$$

A standard USB 2.0 connection runs too slow at 60 MB/s, leaving the only options for transport protocol at Gigabit Ethernet (120 MB/s) and USB 3.0 (625 MB/s). A lightweight and compact camera would be ideal for minimizing power consumption in the stepper motors when panning and tilting as well as being able to fit in a small enclosure. Cameras that required fewer cables plugged in would also give more flexibility to the slip-ring connector. Below is a table summarizing the cameras that were investigated:

Table 3.3: Camera Options Specifications

Camera	Transport Protocol	Resolution	FPS	Cost
MER-500-14GC-P	GigE	2592 x 1944	14	$196
MER-133-54GC	GigE	1280 x 960	54	$218
Flir Firefly DL	USB 3.0	1440x1080	60	$300

The first line of cameras that were considered were the GigE cameras. These cameras transmit frames over a gigabit ethernet connection, removing the need to worry about the bandwidth. They also implement PoE (Power over Ethernet), allowing the data transfer cable to double as a power cable. Daheng Imaging provides a few lightweight, industrial cameras for embedded applications at a cheap price. Their MER-500-14GC-P provided the highest image quality in the set at 1944p, but its slow frame rate of 14 FPS was too low to capture quick movements, which could be critical pieces of evidence in a case. They had a more balanced option in the MER-133-54GC, which could run 960p at 54 FPS. However, despite meeting all the requirements set for our camera, the shipping time for the camera was significantly high, and so other lines of cameras were considered.

A big revolution in the camera search came from a Flir Systems' Firefly DL camera. Producing 1080p images at 60 FPS over USB 3.0 and a compact, light design, this camera definitely met our projects requirement. With greater specifications than the GigE cameras came a larger price however, costing about 1.5x more. What we believe justified the significant increase in price and ultimately led to the selection of the Flir Firefly DL was not in fact the impressive specifications, but the camera's built-in image processing. The camera is the first in the industry to implement the Intel Movidius Chip, a vision processing unit (VPU) built to process images through a deep-learning model at competitive speeds. This chip would allow the object detection to be done on the camera and significantly increase camera response rates while slightly lowering the programming effort. A lens was not included with the camera, and so a bundle containing a 6mm, 8mm, and 12mm lens was purchased. The different focal lengths would allow for more flexibility and customization when transitioning from a test environment (a small room) to the production environment (outside).

3.3.7 Xbee RF Module

The Xbee Series 1 RF (radio frequency) modules will be used as part of our design to account for the wireless communication between the MSP430FR6989 and the ODROID-XU4. It was a great choice because of how simple, low-cost, and low-power the modules are. The RF modules can support up to 90 meters outdoor line-of-sight use, which can be useful for applications where the police officer will be at a distance from the ODROID.

Through its serial port, the module can communicate with UART compatible devices. The serial communication is dependent on the UARTs from the microcontroller device and the XBee module. The settings for the baud rate, start bits, stop bits, parity bits, and data bits must be configured. In Figure 3.6, we can see how two devices can communicate between each other with the use of the XBee Module. Data from microcontroller #1 will be received by the XBee module via the DI (data in line), also known as the transmit line. Xbee module #1 will then wirelessly transmit the data to XBee module #2, so that It can then send the data to microcontroller #2 via the DO (data out) line, also known as the receiving line. When there is no data being sent, the signal should be idle at high.

To configure and manage the RF modules, Digi's X-CTU Software and a serial connection to a PC must be used. Through the XCTU, the baud rate and mode can be modified.

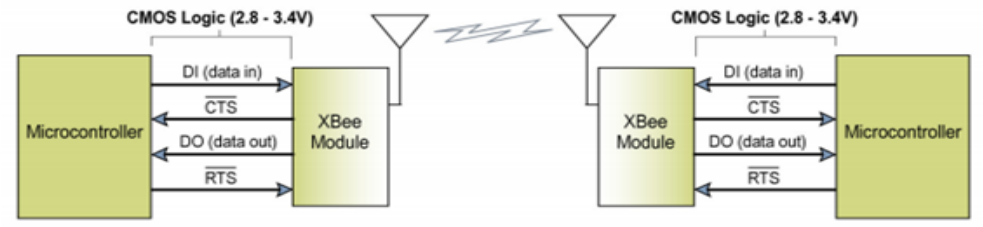


Figure 3.6: UART Interfaced System Data Flow

3.3.8 Boost Converter

The embedded microphone unit will be powered by a 4.2V lithium polymer rechargeable battery. This does not meet the required 4.5 - 5.5V that the speech recognition module requires to operate, therefore; a boost converter will be essential for our PCB. The boost converter allows an input voltage to be boosted up, while decreasing the output current. Using an IC, a boost converter can be designed such that we can achieve a desired boost in our input voltage to satisfy the voltage condition for the speech recognition module. The TPS61322 IC is a great option for our design, as it is highly efficient and carries a low 6.5-uA quiescent current. The IC can be used with a Schottky diode for applications where the load current needed is higher than 500-mA, at around 3-5V output voltage conversion. For our application, the Schottky diode design is not needed since the output current for the speech recognition module should be less than 40mA.

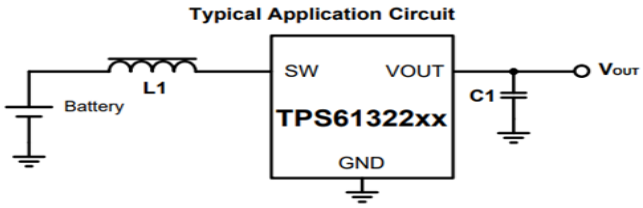


Figure 3.7: TPS6132xx Circuit from TI datasheet

The typical application circuit can be shown above in Figure 3.7, where an inductor is in series with the input voltage, while a capacitor is in parallel with the output voltage. The inductor is one of the most important components for a power regulator design. It affects the steady state and transient response behavior, due to the inductor value, saturation current and the dc resistance. When choosing a value for the inductor, the capacitor value must also be considered. As your inductor value increases, so should the capacitor value to ensure a stable loop. The capacitor value is essential for output voltage filtering; therefore, a low ESR (series resistance) ceramic capacitor is ideal for the design. ESR can affect the time constant, RC, for the charging and discharging of the capacitor, which can easily affect the capacitor and how the voltage changes or ripples. In practice it is a good idea to combine some capacitors in parallel. By utilizing this design with the proper inductor and capacitor values we can get our desired output voltage with minimum quiescent current.

3.3.9 Lithium Polymer Battery Charger

Since the embedded microphone unit will all be powered through a lithium polymer battery, it was in the best interest to make it rechargeable. It would be ideal for scenarios where the microphone unit is not being used and can be charged. This way we can ensure that the battery will not be drained in between traffic stops and allow there to be interface between the microphone device and the Odroid. A great candidate for this is the BQ21040DBVR IC chip from Texas Instrument. The IC device is a Li-Po linear charger device for space-limited portable applications. The device can be operated by a USB port or an AC adapter. For our application, we believe that the USB port is the better option. This allows for the officer to directly charge the embedded microphone unit through a USB port connected on their laptop while in their police car. Conveniently, the device should ideally always be fully charged if the police officer is correctly charging the embedded microphone while driving around and not in use.

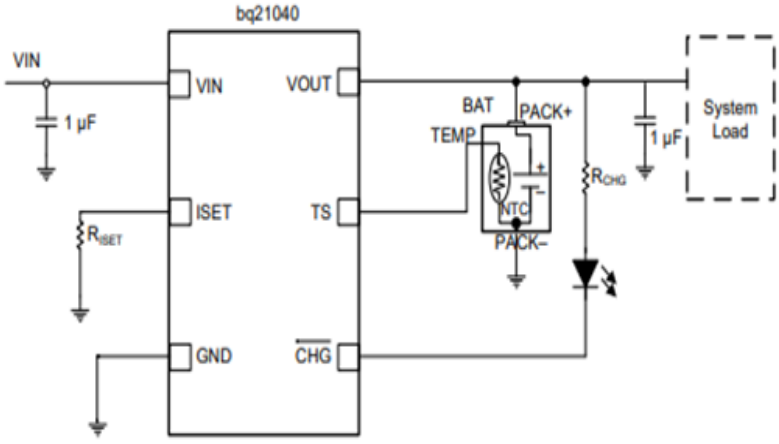


Figure 3.8: Battery Charger schematic taken from Texas Instrument bq21040 datasheet with permission

From Figure 3.8 above, it can be seen that the IC is a 6-Pin small outline transistor (SOT) containing the following pins:

1) TS – Temperature sense terminal that is responsible for following the temperature sensing standard for the Li-Pol batteries. This is done by monitoring whether the terminal is low or high. When pulling high, termination and timer disable mode (TTDM) is enabled, and the TS monitoring is disabled. Pulling Low disables the IC and disables the charging. When TTDM is enabled, a 10-hour safety timer is held in reset and termination is disabled. By removing the battery pack, the TS terminal can go high, which then a battery detect routine is run to determine if the battery was removed.

2) Vout – This is the pin that the battery connection occurs at and provides current to the battery and system. This pin can also include a 1uF to 10uF bypass capacitor in parallel with the system load and battery. The system load, in this case the embedded microphone, can also be attached while charging if the average system load does not keep the battery from charging fully during the 10-hour safety timer. The Vout terminal is a current limited source and protected against shorts. Therefore, if the system load exceeds the output current threshold, the output will be discharged.

3) CHG – A resistor in series with an LED is connected between Vout and the CHG pin so that a visual indication can allow the user to determine whether the battery is charging. There is an internal FET which when it is on, the CHG pin is low. This indicates that the charging is occurring. When the FET is off, and there is an open drain, then this indicates that the charging is complete or that there is no charging occurring.

4) ISET – This pin is used to program the fast-charge current settings. There is an external resistor connected between the ISET and ground. The resistor value can be changed to determine the current value. The range is 10.8kΩ for 50mA to 675Ω for 800mA.

5) GND – Ground terminal

6) Vin – This is the pin where the input is connected via an external DC supply such as a USB port or AC adapter. Bypass capacitors are connected from the input to the ground to reduce unwanted noise coming from the power supply. The bypass capacitors can range from 1uF to 10uF.

4. Relevant Standards

Engineering standards followed throughout our project design will be discussed in this section. These standards can include any hardware or software standards, for any programming language or any electrical device used in our design. IEEE Standard Association defines standards as “published documents that establish specifications and procedures designed to maximize the reliability of the materials, products, methods, and/or services people use every day. Standards address a range of issues, including but not limited to various protocols to help maximize product functionality and compatibility, facilitate interoperability and support consumer safety and public health.” By following the specified standards, technologies can be measured by their consistency, effectiveness, and how safe they are.

4.1 C Standard

ISO/IEC 9899:2018 is the most recent standard for the programming language C. ISO (the International Organization for Standardization) and IEC (the International Electromechanical Commission) are responsible for forming the specialized system for worldwide standardization. The ISO/IEC 9899 Standard document establishes the interpretation of programs written in C language. Some of the topics specified in the document can include representation of C programs, syntax and constraints, representation of input data to be processed and output data produced by C programs, and finally restrictions and limits imposed by an implementation of C [4.1].

Under the *Chapter 4: Conformance* section of the 550-page C Standard report, the term “shall” and “shall not” are defined. In the International Standard, “shall” is to be understood as something necessary or required on an implementation or on a program, while “Shall not” are prohibited. A behavior is undefined if a “shall” or “shall not” requirement that appears outside of a constraint is not followed. An example of one “shall not” requirement is the following:

“The implementation shall not successfully translate a preprocessing translation unit containing a #error preprocessing directive unless it is part of a group skipped by conditional inclusion” [4.1]. An example of a “shall” requirement is the following: “A strictly conforming program shall use only those features of the language and library specified in this International Standard” [4.1].

Under 5.1.1.1 Program structure section of the C standard report, we can see how the text of the program is kept in units called source files in the International Standard. Through the preprocessing directive #include, a source file and all the headers and source files included are known as a preprocessing translation unit. Translation units of a program communicate through calls to functions, through manipulation of data files, or through manipulation of objects whose identifiers have external linkage. The “Main” is the function called at program startup, and it shall be defined with a return type of int in the form of “int main (void) or with two parameters [4.1].

Chapter 6 talks about the language and syntax notation. Syntactic categories (or nonterminals) are indicated by *italic type,* while literal words and character set members (or terminals) are indicated by bold type. To define a nonterminal, a colon ":" must be used. In Chapter 7, libraries are discussed and how a library function is declared, by using the proper standard header. The header is responsible for declaring a set of functions, along with additional macros. Any declaration of a library function shall have external linkage [4.1]. While there is a lot more discussed on the 550-page document, these were the main standards that will be followed throughout the implementation of our software.

4.2 C++ Standard

The Internal Organization of Standardization standardized C++ (ISO/IEC 14882:2017), with the latest standard being C++17, standardized in December of 2017. There are two main features to the C++ standard. The first of these features is its hardware-mapping capabilities. The C language allows developers to easily communicate directly to the hardware, and these capabilities were kept in C++. In addition, abstraction in its lightest form, a feature not present in C, was added to drastically simplify development and increase code usability at little to no cost in code efficiency and optimization. Most other programming languages that provide the abstraction feature cut a lot of efficiency in doing so. The standard also specifies how the language works and how it interacts with the system.

C++ supports objects and classes along with the four many features that object-oriented programming provides: abstraction, encapsulation, inheritance, and polymorphism. Templates are also built-in, allowing for classes and functions to work with multiple different data types. Object memory can live in static storage, thread storage, automatic storage, and dynamic storage. This gives developers full flexibility on when their objects' memory is deallocated while providing some built-in memory management. There are many operators that can be used with primitive/built-in types and can be overloaded in custom classes to add/modify/provide functionality. Polymorphism allows classes to expand on one another, adding more state and behavior while potentially changing the pre-existing state and behavior. C++ has integrated error handling through the use of try-catch blocks, which automatically recursively exit the scope that the error occurred in until the error (known as an exception) is handled.

The other aspect of the C++ standard is the standard library. This library defines all of the primitive types that serve as the building blocks for custom types. It also provides built-in support for input/output operations, multithreaded programming, common search algorithms, and much more. To access the different components of the standard library, the compiler recognizes the #include tag to attach previously defined types/functions from either the standard library or custom static libraries before compiling. The standard library is partially based on the Standard Template Library, which allows its algorithms and types to be generic. The library is vital for application development and comes with most major compilers.

4.3 IEEE 802.15.4 (low-rate wireless personal area networks)

In the IEEE 802.15.4 standard, the protocol and interconnection for data communication devices using low data rate, low power, and low complexity short range radio frequency transmissions, (such as a Xbee RF module), in a wireless personal area network (WPAN) are defined. The IEEE 802.15.4 standard is responsible for defining the physical layer (PHY) and medium access control (MAC) sublayer specifications for low-data-rate wireless connectivity. The physical features of the physical layer are activation and deactivation of the radio transceiver and transmitting/receiving packets across a physical medium. The medium access control (MAC) sublayer enables the transmission and reception of MAC protocol data units across the physical layer data service [4.2]. The document also mentions Peer-to-peer network formations, which in peer-to-peer topology each device is capable of communicating with any other device within its radio communications range. Various of frequency bands are supported including:

- 868-868.6 MHz
- 902-928 MHz
- 2400 – 2483.5 MHz (2.4GHz to 2.4835GHz)

The Xbee RF modules we are using for wireless communication between the microphone device and the ODROID follow the IEEE 802.15.4 standard. It supports the needs for low cost, low power, and reliable communication of data between devices. The Xbee device will operate within the ISM 2.4 GHz frequency band and operate in a peer-to-peer network topology.

4.4 IEEE 1364 (Verilog)

IEEE 1364 describes the standardized syntax and structure of Verilog, a hardware description language (HDL) for use in FPGA and integrated circuit design. The document describes the specific details of language constructs like register and wire instantiation, allowing designers to physically layout digital circuitry in an easy to manipulate software-code format. Despite serving a different purpose than software written for a microprocessor, Verilog is designed to follow similar syntax rules and formatting as the C language. IEEE 1364.3 specifies these as “Lexical Conventions,” which are followed in order to produce synthesizable logic designs in our project.

IEEE 1364 specifies two forms of Verilog system modeling: behavioral and gate level. Gate level modeling in Verilog allows a designer to directly implement logic gates and combinatorial logic in a design as exactly specified. This can be useful for streamlining designs and wringing the best performance out of many possible logic designs. Behavioral modeling in Verilog removes the requirement for closely specifying the exact usage of logic gates and resources. Behavioral modeling of a system allows for a higher abstraction of the intended function of a design. As laid out in the IEEE standard, the synthesis software that reads your Verilog code will interpret behaviorally modeled logic into efficiently generated logic gates and elements. For the most part, this project overwhelmingly utilizes behavioral modeling in our FPGA designs.

5. Design Constraints

This section will discuss some of the constraints associated with the design and implementation of our project. Constraints can include any limitations or restrictions that are brought upon during the development of our design. Most of the constraints brought challenges and a necessity for critical thinking to work around certain constraints.

5.1 Cost Constraints

Throughout the design and implementation procedure of many engineering related projects, we can realize that there is a tradeoff associated with performance and cost. For the design of the Traffic Stop Watchdog system, it is easy to realize that there are some expensive components in our design. It is important to keep the cost of the project down, and only buy components that are necessary for the design. Without a sponsor for the group project, we must all pay for this project out of pocket. This is a constraint and limitation to how effective we can make our design; therefore, extensive research was done to compare the performance and price of certain components. Extra steps were taken to ensure that many of the different PCB designs created were ordered together so that the manufacturing and shipping cost would be cut down.

5.2 Technology

Choosing to work with the Flir Camera brought a huge benefit in the built in Vision Processing Unit (Intel Movidius). The extra dedicated and specialized hardware for processing incoming frames will speed up response times significantly for the system, but also will add a few constraints involving the software that integrates with the camera. While they do restrict the tools available for development, the response speed increase justifies it.

5.2.1 Operating System

The SDK for the Flir Firefly DL (Spinnaker SDK) requires an operating system in order to use. While an operating system provides a more flexible layer to run programs off of, it adds a heft requirement on the hardware for the system. Supporting an operating system requires a fast enough cpu and large enough memory to perform all the background tasks that an operating system does while still being capable of running user programs with good performance. The operating system takes up a lot of space, and would require external storage to be used to hold it. The storage requirement is not too bad, considering the device needs to have a large storage anyway to store more footage.

The two main choices for operating systems were, as usual, Windows and Linux. Windows provides access to Microsoft's .NET Framework (C#), arguably the easiest and most flexible programming framework that exists. There is a cross platform version of the .NET Framework called .NET Core that can ruin on most Linux distros, but the .NET Spinnaker SDK requires the windows-exclusive version. Linux on the other hand can run

on an ARM processor and is free to use. Ultimately, Linux won the fight over Windows thanks to the greater availability of ARM cpus on embedded devices and much cheaper cost. The distro used was Ubuntu 18.04.5, which requires a 2 Ghz dual core processor, 2 GB of memory, and 25 GB of storage. There are two versions of Windows, Windows 7 Embedded and Windows 10 IoT, that have significantly less hardware requirements but are not supported by the Spinnaker SDK.

5.2.2 Object Detection Algorithm

CPUs and to an extent GPUs are general purpose, meaning the hardware provides little to no limitations on what the chip can process. The Intel Movidius Chip is a specialized processor, meaning it can only be used to perform a set number of operations. The benefit of the specialized processor is that the hardware can be optimized for performing those few operations, producing very high performance for a very low cost. The Movidius chip was built to process images through image classification and object detection models, but only supports a few different algorithms. For object detection, the only supported algorithm is MobileNet SSD v1, which has some benefits geared towards this project. MobileNet is very small and runs faster than other algorithms. Its decrease in accuracy can be compensated by only needing to classify one type of image (police officer). The constraint of only having one algorithm available also adds the constraint of requiring a training software that can train off of the MobileNet algorithm. Section 7.1 will further discuss the training software that was used to train the model.

5.3 Ethical Constraints

While our design did not have many ethical constraints, one can argue that having so many recording devices is not ethical or necessary. With the police officer already having a dashcam on their police car and a bodycam, some might say that this is already enough. With recent events occurring and many police officers taking the blame for handling certain situations, our group believes that the Traffic Stop Watchdog system can be very helpful in providing crucial information and monitoring what is occurring at a scene or traffic stop. Even with a FPV (first person view) from the bodycam, it is still not enough to see what is going around the officer (behind and to the side of the officer).

5.4 Time Constraints

Throughout the course of the semester for Senior Design 1, there were many assignments due to show the progress of our project. With a 45, 75, and final 90-page report deadline put in place, this forced us to stay ahead and ensure that all team members were putting in effort to reach the deadlines. Every engineering project has strict deadlines that must be met, and penalized if not met. Punishment for not reaching the page count, or even submitting in the final report late can include a letter grade reduction. To account for this, we had weekly meetings where we discussed the progress of our report, and even set goals to reach a certain number of pages by the end of the week.

Along with the design of the project, we were also responsible for taking quizzes on some of the Senior Design material, while only allowing one question to be wrong to pass the quiz. Every quiz that was failed more than five times would penalize you a deduction of a letter grade to your final grade. These quizzes were to be taken seriously in order to pass the class and took up some time to read the material from the textbook.

Working ahead of schedule is one of the methods that our group has used to ensure that these deadlines would be reached. Ensuring that we started as soon as possible with our design, while researching key components of our project, helped us get ahead and begin the implementation of our project.

5.5 Health and Safety Constraints

COVID-19 was a big health and safety constraint going into Senior Design 1. As of Spring 2020, all UCF classes were transitioned to online through Zoom. Through the Summer and Fall 2020 semester, the same rules were applied. UCF recommended students to practice social distance and not to meet up in groups on UCF campus. This limited our group on how often we could meet up in person to discuss crucial parts of our project design. To solve this issue, we decided to meet at least once a week through Zoom to discuss topics about the project. Due to campus operations being limited, we were not able to access any lab equipment to troubleshoot or test components. Thankfully, the ECE department was generous enough to send each group an Analog Discovery 2, along with some other tools and breadboards.

Another safety constraint that will be considered in the future during the development of our project is the use of a vehicle. When the project is completed and it is time to showcase the project, ideally, we would like to show the functionality of our project with the use of the 12V battery from a car as our supply voltage. This means mounting our camera system on top of the vehicle and testing it in an outside environment. Some tests might include the vehicle moving, and we must ensure that we take a look at our surroundings and do not cause any accidents while testing. We must also ensure that we are properly connecting and integrating the 12V battery to our design without causing any damage to the vehicle or the hardware used for the project.

The handheld device will operate off rechargeable LiPo batteries, which can be quite dangerous if not careful. Lithium polymer cells should not exceed 4.2V or they can become overcharged and therefore dangerous. This could lead to a potential fire started by the battery. Certain precautions can be used to ensure that maximum safety is achieved while using LiPo batteries. This can include:

- Picking up a LiPo battery by its body, and not the leads
- Wait until the battery is cooled down before charging
- Keep an eye on the battery while charging
- Do not overcharge (do not exceed 4.2V per cell)
- Avoid leaving the battery out in the sun, or in a hot car

6. Hardware Design

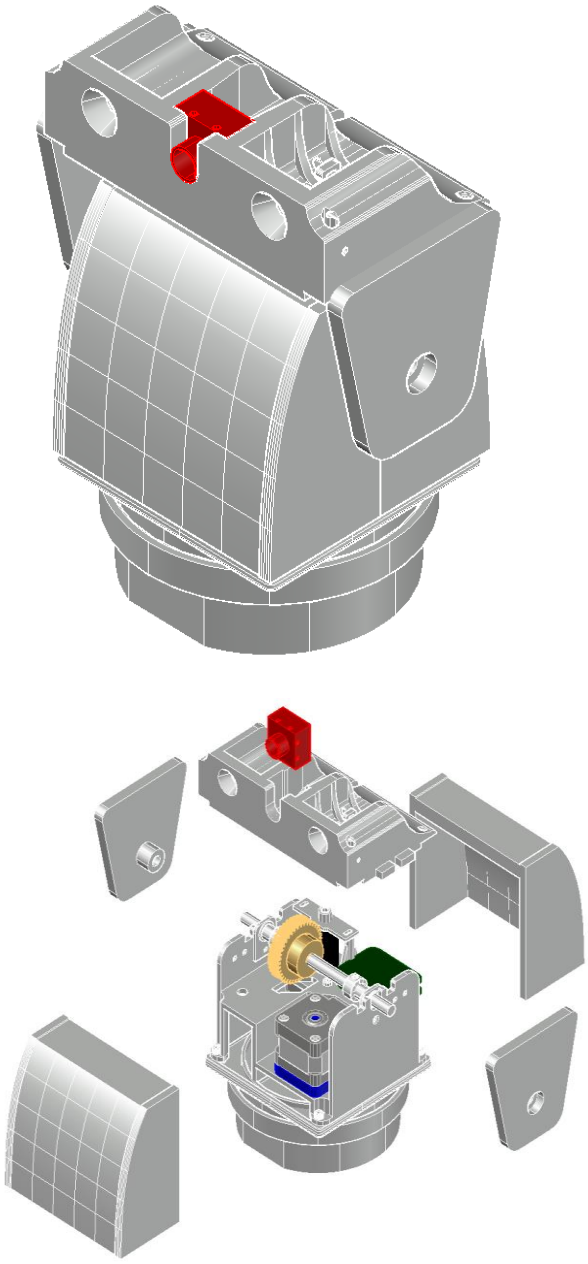


Figure 6.1: Camera Assembly, External & Internal View

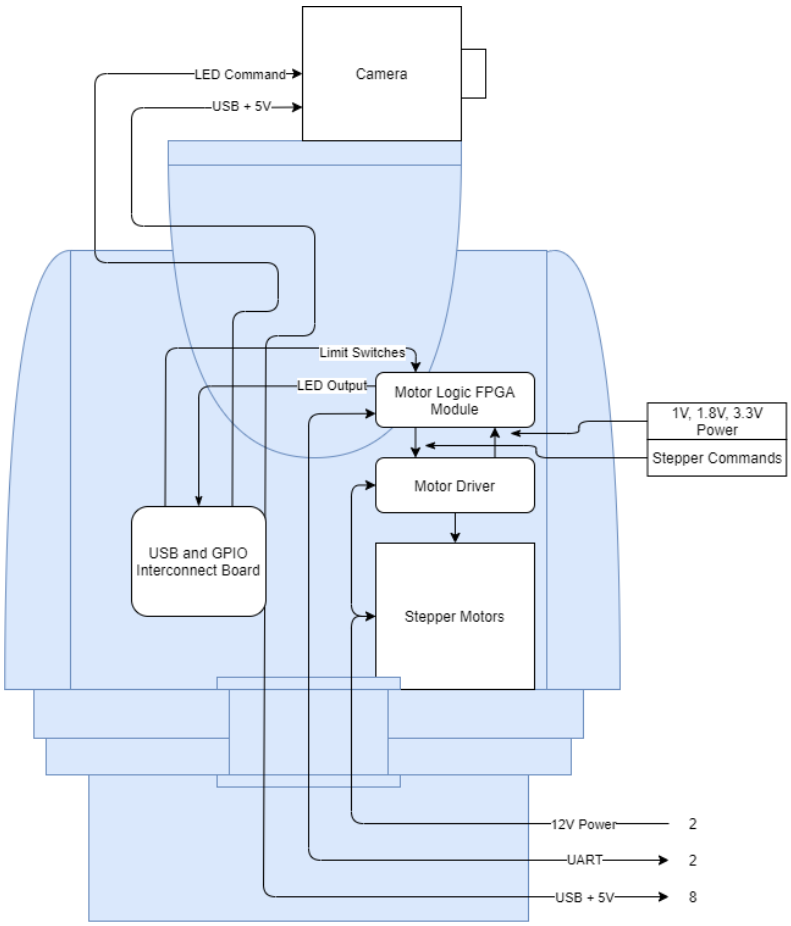


Figure 6.2: Camera Subsystem Hardware Diagram

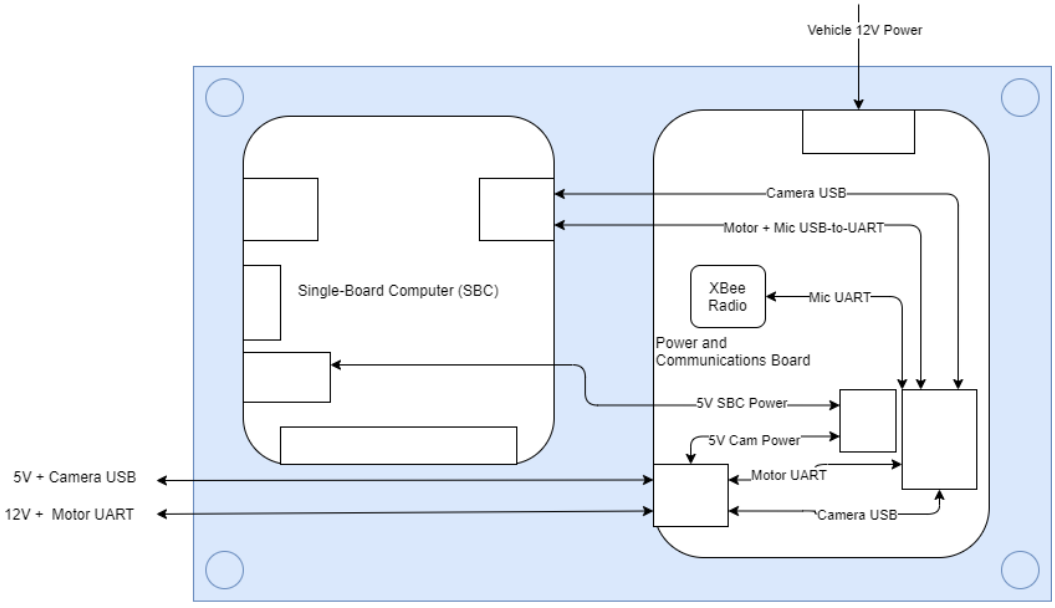


Figure 6.3: SBC Enclosure Subsystem Hardware Diagram

6.1 Mechanical Design

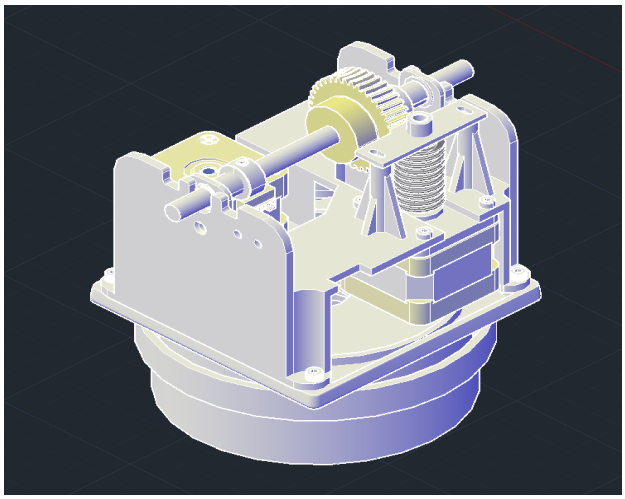


Figure 6.4: Camera Mount CAD Model

To support the fast tilt and pan rates needed for an autonomous camera, a rugged and stable two-axis platform is needed. As a foundation for this design, our team purchased a cheap pan and tilt mount for closed-circuit surveillance cameras from a surplus store. The mount, equipped with geared AC motors and held together with plastic screws, was initially unsuitable for our purpose. It lacked rigidity in both axes, and the assembly was flimsy with lots of play in the gear subassemblies. It served as a great starting location, but much of the original hardware was replaced or redesigned for our usage. This was known ahead of time, since the original mount was designed for manual control in 90 degrees of movement for both axes.

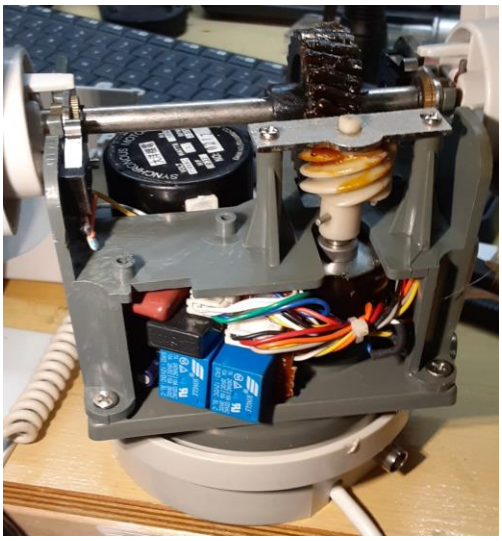


Figure 6.5: Camera Mount Internals, as Purchased

To start, the two motors were unsuitable. They required 120VAC and used extremely high gear ratios that inhibited output speed. These were replaced with high-accuracy 12VDC stepper motors. To accommodate the face profile of the stepper motors to the mounting locations for the old motors, 3D printed faceplates were designed and printed for the steppers that precisely aligned the motors to where their shafts needed to be based on grooves in the plastic housing. Instead of the self-tapping plastic screws that held the original motors, new stainless steel fasteners were installed, in 4 locations instead of 2, with metal-on-metal thread engagement to securely retain the motors.

The next challenge was the geartrains for each axis. These were not designed with precision in mind, and the gears had enough play for tenth-of-an-inch axial and radial movement in their housings. The pan-axis primary output gear had a crack that would unmesh it from the secondary gear every rotation, causing binding in the assembly and stopping any movement in that axis. To rectify this, this gear was replaced by an identical pitch and outer diameter brass gear with set-screw shaft attachment to a shaft. A custom shaft was fabricated from 6061 aluminum with set-screw attachment to the pan-axis stepper motor shaft. This gear shaft was pressed into a flanged bearing that was inserted into the plastic housing to limit radial movement. The result was a rigid, low backlash gear subassembly.

In the tilt axis, the original design used a worm gear design with plastic injection-molded parts. While the worm gear design choice was retained, the gears and shafts were unsuitable for precision because the injection-molding had left many plastic spurs in the gear profile that would unmesh the gears at some speeds. A new brass worm wheel and ground-steel worm gear replaced the plastic components, and a shaft was fabricated out of tight-tolerance 4140 steel to fit ball bearings in mounting plates instead of the existing loose-fitting brass bushings. These mounting plates were built out of carbon steel so as to add rigidity to the plastic housing. On the worm wheel shaft, spacers and belleville spring washers were inserted to add axial preload on the bearings which greatly reduced axial movement. The steel worm gear is mounted on a shaft supported by a low friction, tight fitting bushing pressed into a steel support plate. Gear metal-on-metal contact surfaces are greased or oiled to prevent premature wear.

Everywhere else in the camera housing, steps were taken to produce a high-strength assembly that would limit wear from assembling and disassembling its components. All self-tapping plastic screws were replaced with stainless steel machine screws mounted in brass heat-set threaded inserts. This allows repeated assembly and disassembly of fasteners without loss of the thread material. To facilitate 180 degrees of rotation, the cable passthrough in the center of the assembly was augmented with a slip ring connector that prevents electrical wiring from becoming twisted from the rotation of the mount.

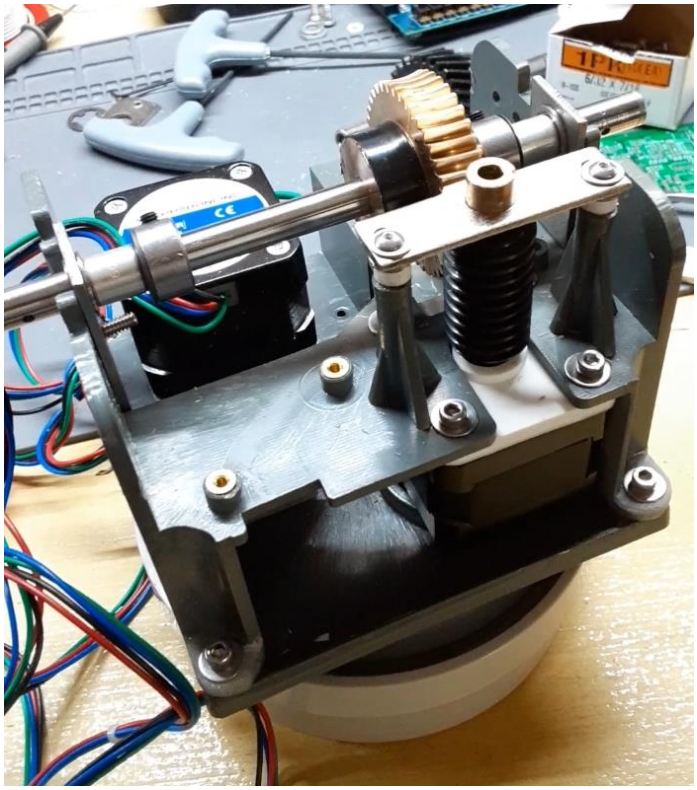


Figure 6.6: New Camera Mount Internals

6.2 Power Distribution and Device Inter-Communication

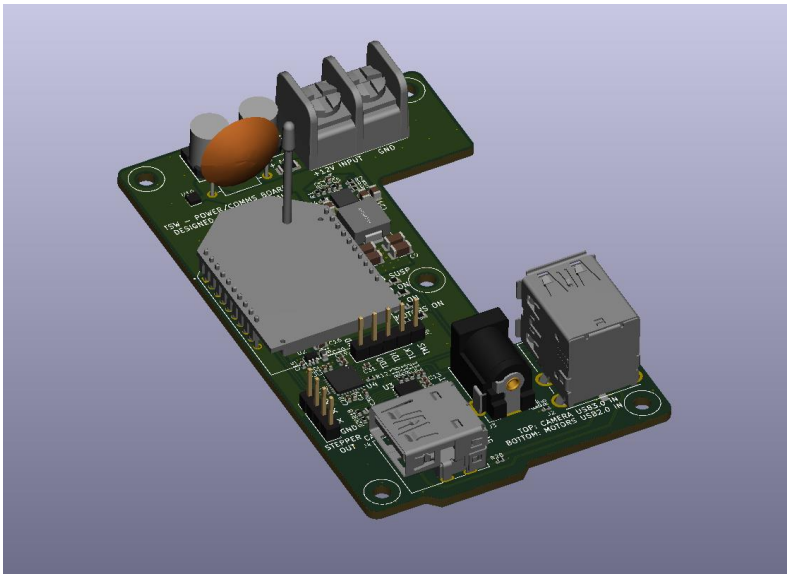


Figure 6.7: Power and Communications Board

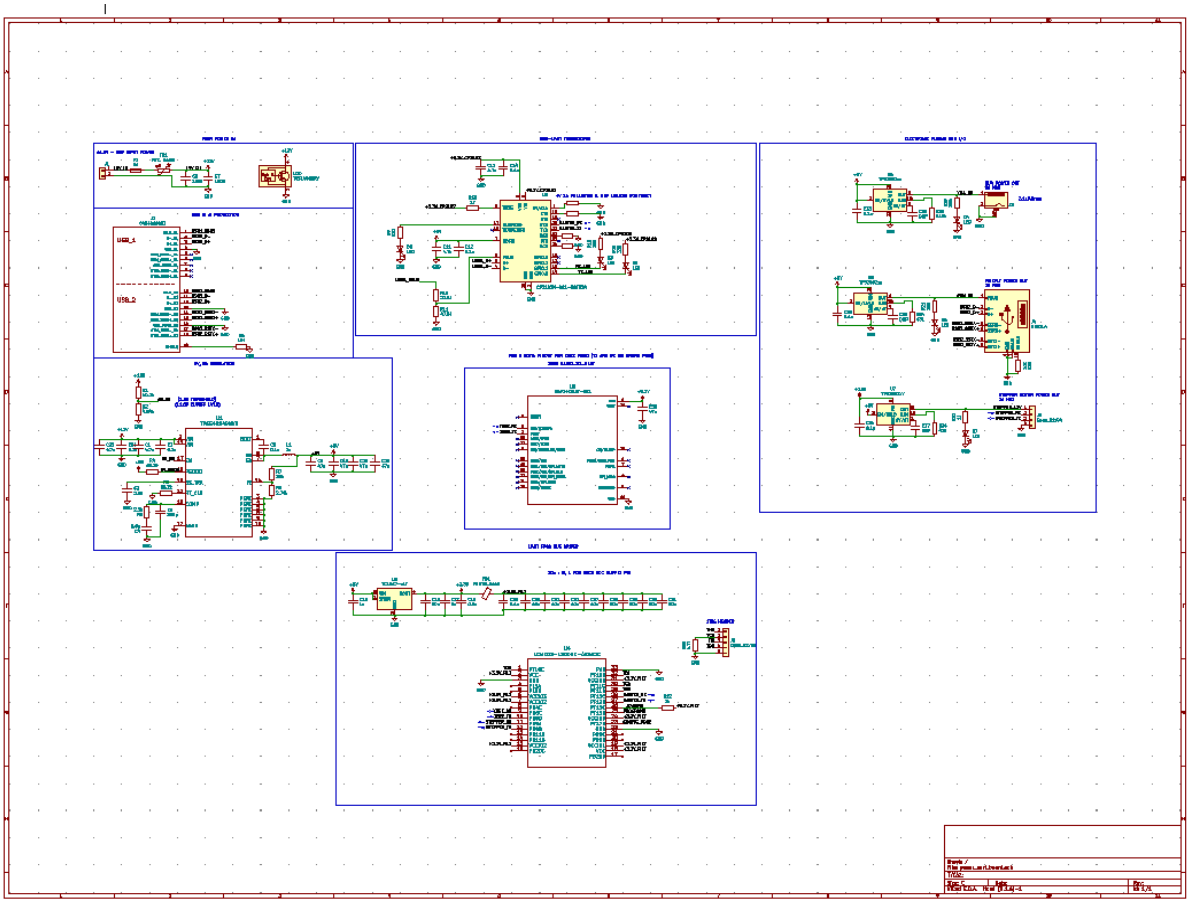


Figure 6.8: Power and Communications Board Schematic

6.2.1 Power

A number of voltage supplies are needed for the various subsystems in the camera assembly. The primary supply is a 12V lead acid car battery found in most automobiles. 11.5V - 13V is fed to the power and communications board located within the SBC enclosure. This board is a 4-layer design, with the stackup specified as JLC7628 (See section 6.3 for more details). Primary input power is filtered by a pair of 100 uF electrolytic capacitors and protected by a 6A fuse. An inrush-current limiting NTC is in series with the main fuse to prevent high capacitor charging currents from damaging the vehicle power bus or blowing the fuse. Screw terminals are used to prevent vibration from loosening connectors.

From the filtered and protected input 12V supply line, a high-current capacity 5V supply rail is generated. This is achieved by a Texas Instruments TPS54824 switch-mode voltage regulator. Rated for 8A output current capacity, it provides the primary input power for the Odroid XU4 SBC and Firefly DL Camera. The output 5V supply along with the 12V supply are passed through three separate Texas Instruments TPS2592 electronic fuses, one for each major subsystem power rail (5V camera, 5V SBC, and 12V motors). This protects each subsystem from voltage spikes on the primary 12V bus,

prevents devices from drawing power if the vehicle battery is below the undervoltage discharge threshold (11.5V), and shuts down subsystems that draw more than their rated current consumption. Status of each subsystem enabled power input is verified by an onboard surface mount LED.

6.2.2 Communications

The power and communications board additionally serves as the interface between the high-level AI computation taking place on the Odroid XU4 and the low-level processing needed to drive the camera motors and provide user input from the wireless microphone device. The primary communication bus between these different subsystems is a modified UART protocol. The host, the Odroid XU4 SBC, needs to send and receive serial data to two subsystems using UART. The microphone device communicates using a pair of XBee Series 1 digital radios, which have onboard UART interfaces for the incoming and outgoing data. The motor module has a UART transceiver implemented on the driver FPGA that translates angular error commands into stepper movement.

Natively, the UART protocol does not specify an address scheme or provide for multiple slaves on one bus like I2C or SPI. Unfortunately, the need for UART was driven by specific hardware availability, so in order to make it work in this capacity, a small Lattice Semiconductors MachXO2 FPGA was implemented as a communication bus manager. Explained in more detail under “Programmable Logic” Section 7.5, the device allows two slaves to communicate with the SBC host without any messages being missed.

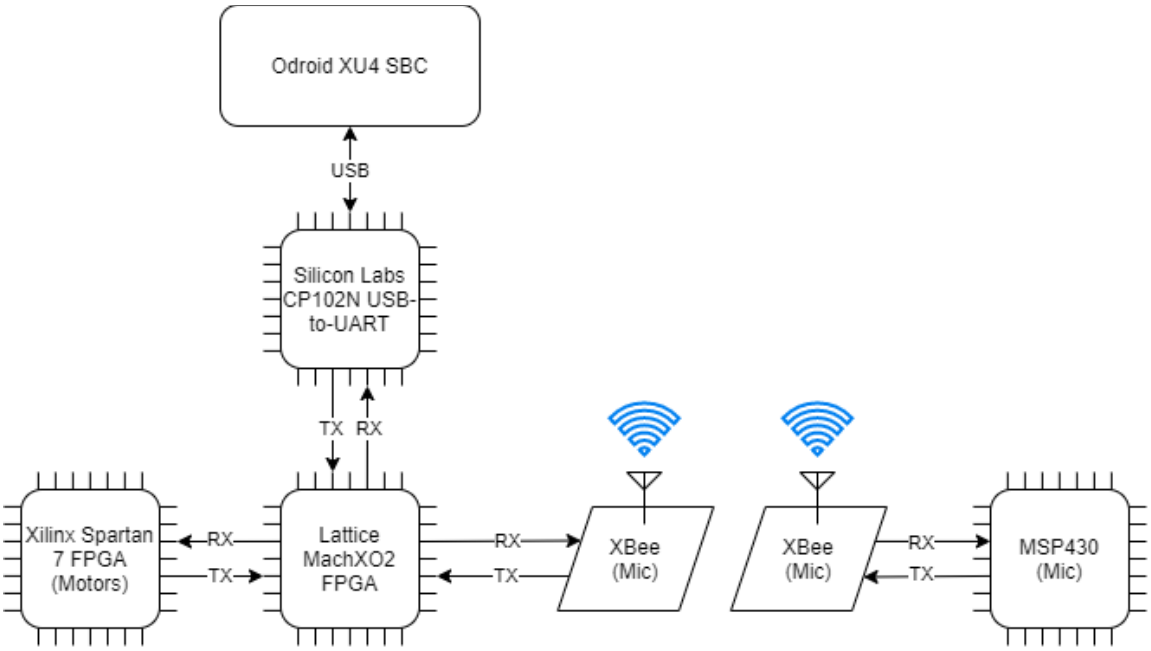


Figure 6.9: System Communications Diagram

6.3 Motor Driver and Controller Subassembly

Because the motor driver and controller circuitry are very dependent on one another, the two boards are combined as a subassembly in a stacked configuration. The separate-board, stacked configuration is also beneficial because it simplifies the hardware development, reduces the footprint of the boards, isolates delicate logic and flash memory components from the more robust driver components, and isolates high voltages and switching power components from the low voltage digital controller FPGA. The following image demonstrates the stacked configuration of the boards, and the following subsections detail the design of each component board.

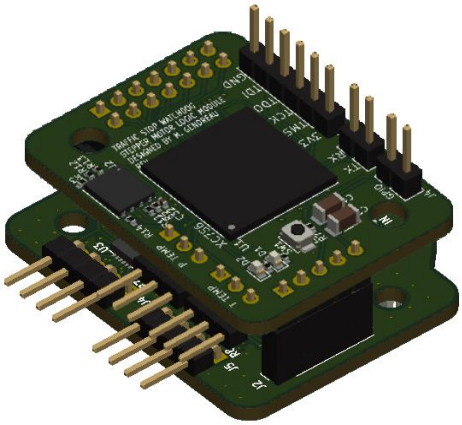


Figure 6.10: Driver-Controller Circuit Subassembly

6.3.1 Stepper Motors & Driver

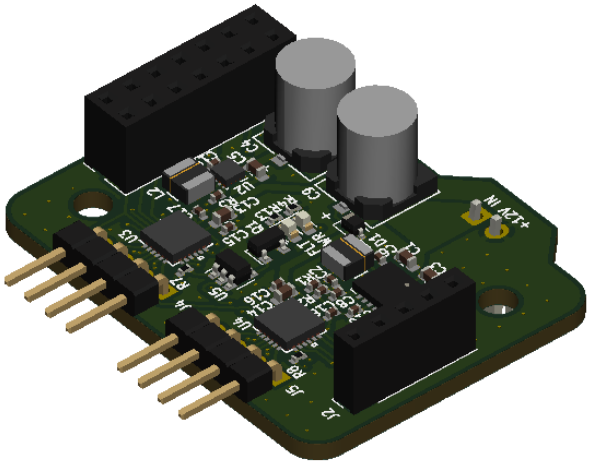


Figure 6.11: Stepper Motor Driver Board

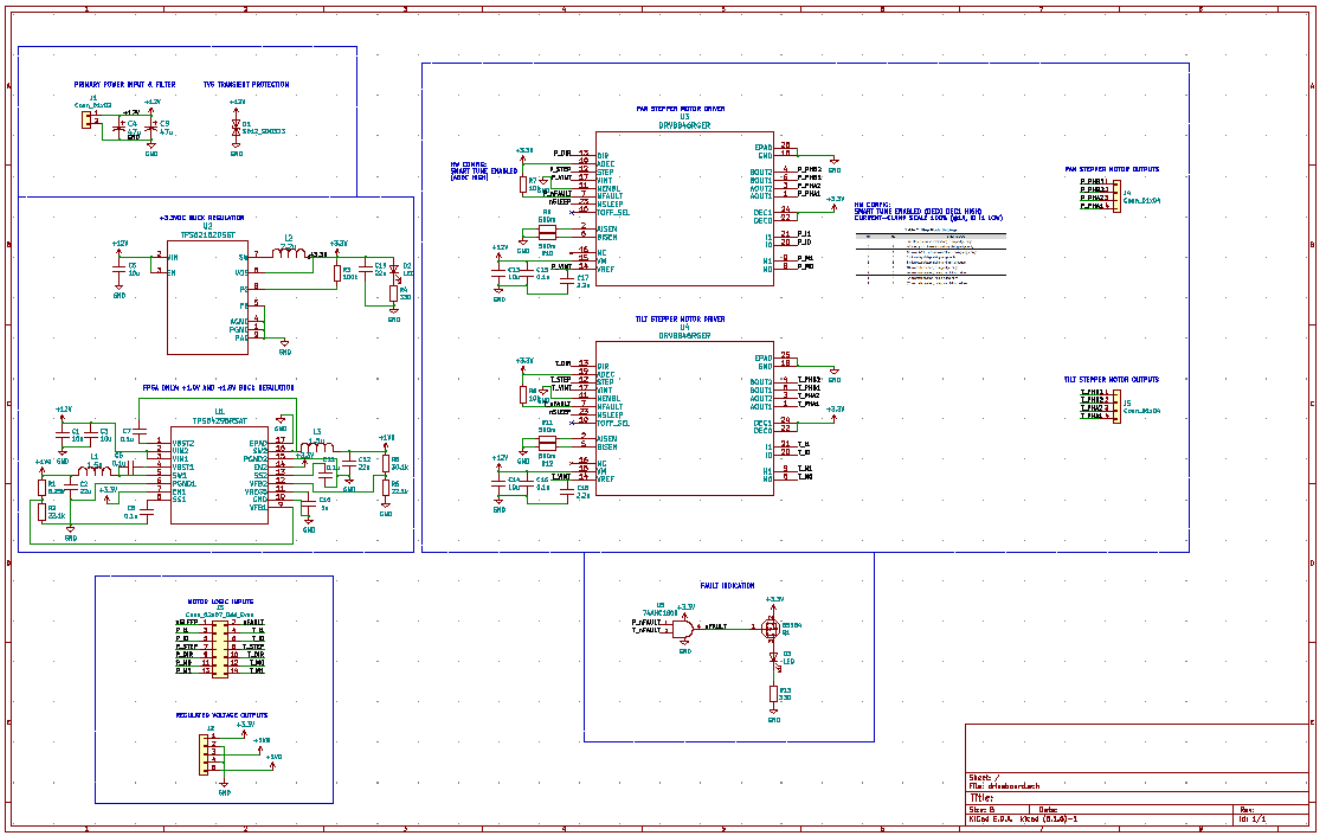


Figure 6.12: Driver Board Schematic

As mentioned, stepper motors took the place of geared AC motors. The selected motors are two-phase with 1.8 degree step resolution, which is sufficient for the angular resolution of the design given the gearing on both axes. However, for reduction of noise and vibration, the motors are micro-stepped using the commutation logic of the DRV8846 stepper driver ICs. These reside on the stepper driver and voltage regulation board, part of the main power and logic subassembly that includes the FPGA board which stacks on top in order to receive power and deliver motor inputs. Using the highest level of microstepping capable of these chips, 1/32 of a full step, we are capable of achieving angular resolution of 0.05625 degrees in each axis before gearing.

One of these chips is connected to the two phases of each motor, and they contain a MOSFET full-bridge per phase. The lookup tables for advancing each leg of the full-bridges are contained within the IC, so only external step and direction input signals are necessary for advancing the motors. Additionally, the DRV8846 provides inputs for setting the microstep resolution and current-chopping limits. These inputs are utilized for smoothing the motor movement and preventing the motors from overheating respectively.

Besides driving the stepper motors, this board also supports voltage regulation from the 12VDC input. Two switch mode voltage regulator ICs are used for this purpose, generating 3.3V, 1.8V, and 1.0V rails. These are required for the logic inputs to the motor

drivers as well as the various voltage supplies required for the Xilinx Spartan FPGA. 3.3V feeds the IO banks, 1.8V is used for the auxiliary voltage input, and 1.0V is used for the internal logic supply voltage. Switch mode regulators are used instead of linear regulators to reduce power loss during the step-down. Additionally, this section of the design lacks sensitive analog components that might be disturbed by switching noise on the voltage rails.

The board stackup is 4 layers, 1 oz. cu outer layers with 0.5 oz. cu inner layers. Two outer signal layers sandwich an inner ground plane and 3.3V power plane to improve routing and reduce loop inductance between decoupling capacitors and switching loads. The exact stackup, per JLCPCB's JLC7628, is shown in the following table.

a) JLC7628 Stackup:

Layer	Material Type	Thickness	
Top Layer1	Copper	0.035 mm	
Prepreg	7628*1	0.2 mm	
Inner Layer2	Copper	0.0175 mm	1.1mm (with copper core)
Core	Core	1.065 mm	
Inner Layer3	Copper	0.0175 mm	
Prepreg	7628*1	0.2 mm	
Bottom Layer4	Copper	0.035 mm	

Figure 6.13: JLC7628 4-Layer Stackup (from jlcpcb.com)

6.3.2 Stepper Motor Controller Board

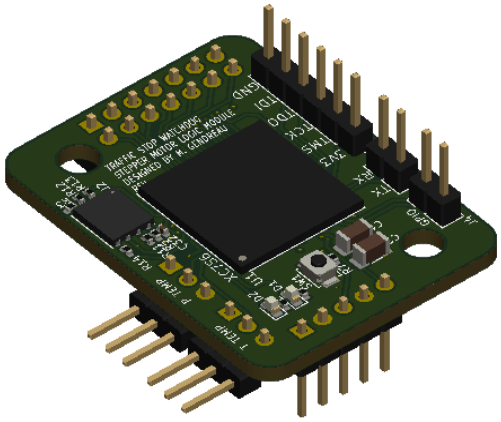


Figure 6.14: Motor Controller Board

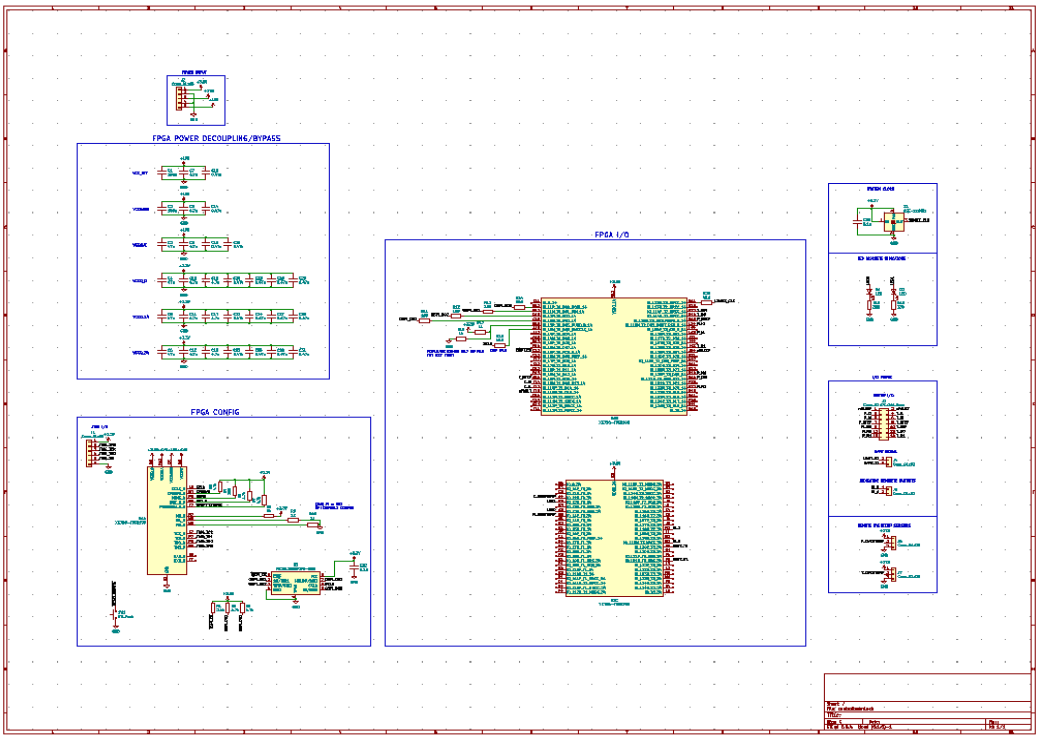


Figure 6.15: Controller Board Schematic

The logic board for the stepper motors is a small (60x40mm) 4-layer board that mounts directly on top of the aforementioned driver board. This board receives UART commands from the Odroid XU4 SBC that translate to movement of the motors in proportion to the angle of movement commanded, and it also monitors various parameters to prevent damage to the assembly. The brains of the board reside in a Xilinx XC7S6 Spartan-7 FPGA with configuration data stored in a Macronix MX25L3233 nonvolatile flash memory IC. The primary system clock is fed from a 12.00MHz crystal oscillator. This frequency was selected as the base frequency for clock dividers used throughout the Verilog design, and it meets the specs for the system clock as specified in the chip datasheet. Status LEDs are provided on a couple IO pins for the purpose of system debugging, along with a configuration reset button. The programming interface of choice is JTAG, broken out to a 6-pin header, and used to configure the SPI flash chip through the internal JTAG configuration bus of the FPGA.

The 4-layer stackup for this board is identical to the driver board's stackup. 4 layers is more important for this board because the chosen FPGA IC uses a 196-pin ball grid array (BGA) package in a 15 mm x 15 mm footprint. Signal and power routing are therefore challenging to achieve with most board houses' 2-layer capabilities. The Xilinx FPGA also requires a great deal of decoupling capacitors for the various IO banks and internal voltage supplies, called for by the chip's datasheet. These are accommodated in the design by use of small 0402 SMD ceramic capacitors placed on the underside of the board, mounted as close as possible to the respective voltage supply pins on the FPGA. Decoupling capacitors are placed in order to "decouple" the attached device from potential noise on the voltage supply bus, as well as reduce the amount of noise put

back into the bus by internal switching of the device. Closely placing these capacitors to their respective voltage supply pins reduces the inductance of the recirculation path, better suppressing high frequencies on the DC supply.

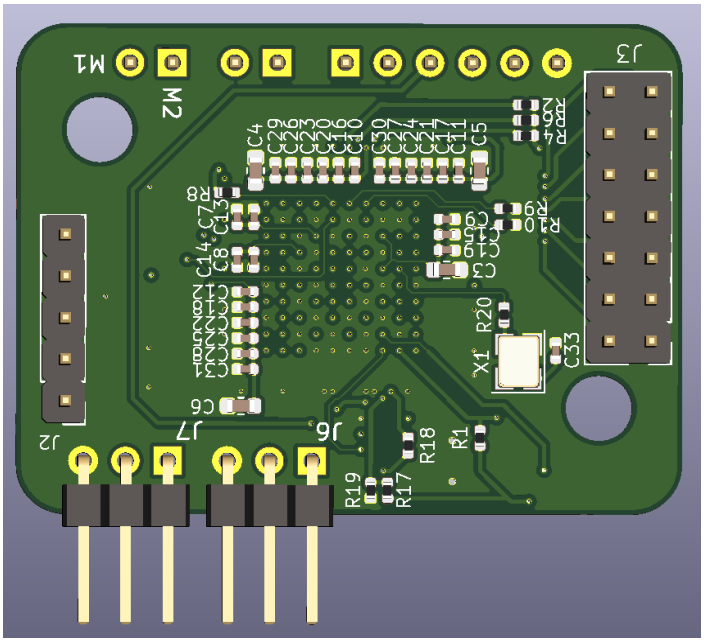


Figure 6.16: Underside of the Motor Controller Board

6.4 Camera & Optics Subassembly

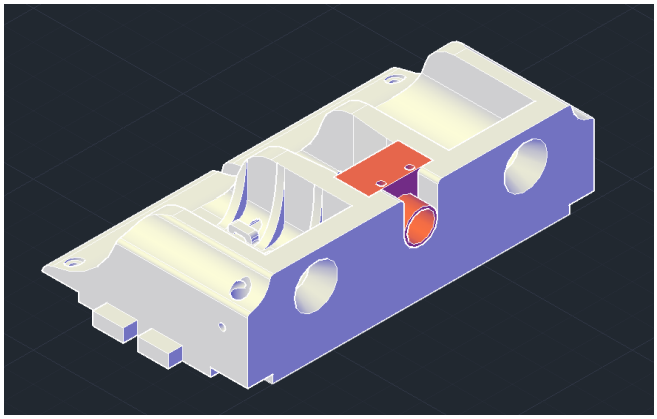


Figure 6.17: Camera Platform Subassembly

The camera itself is located on a 3D printed platform attached to the pivoting arms of the tilt and pan mount. The platform has mounting locations shouldering the camera for LED floodlights and indicators. A pair of specially-designed PCBs serve as the base for 8 LED floodlights, providing up to 2.4 watts of illumination. These help illuminate a target in low lighting conditions, automatically activated by the logic on the Odroid XU4 SBC. Asymmetric LED indicators on these PCBs also display the status of the system. The left

side of the camera contains a green LED for “power on” status, while the right side of the camera contains a red LED for “tracking” status.

6.4.1 Camera

The camera is fitted with an S-Mount, and is fitted with a 6mm 1080p 1/2.5 inch lens. The sensor on it is a Sony IMX296LLR, which has 3.4um x 3.4um pixels at 1.58M. This gives the camera a better ability to capture a scene while also resolving details needed for analysis.

6.4.2 Indicator Board

The indicator board is a simple two-layer board with LEDs for both status and target illumination. The board fits into the camera platform subassembly in a pair of conical reflectors designed into the 3D printed part. Each board is identical in PCB design, but there are jumpers that are connected in one board to enable it to have the opposite response from its counterpart for the illumination of its status LED. When the system is in track mode, a red LED is illuminated while the green LED turns off, and this logic is encoded in the same single signal line - the red LED is switched by an inverter transistor circuit on the same signal as the green LED.

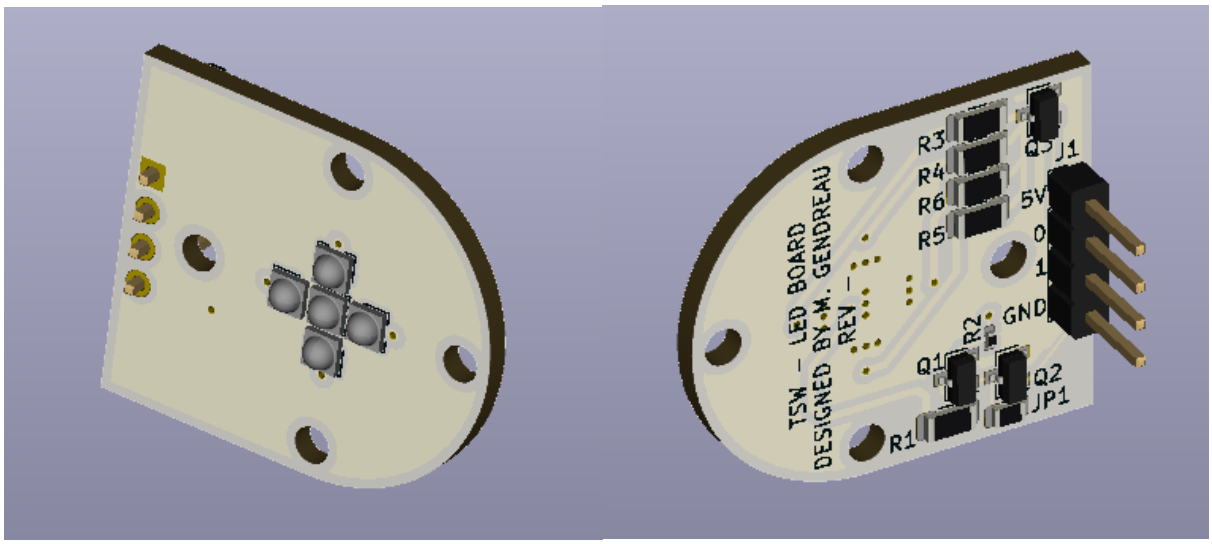


Figure 6.18: Indicator Board Front and Rear

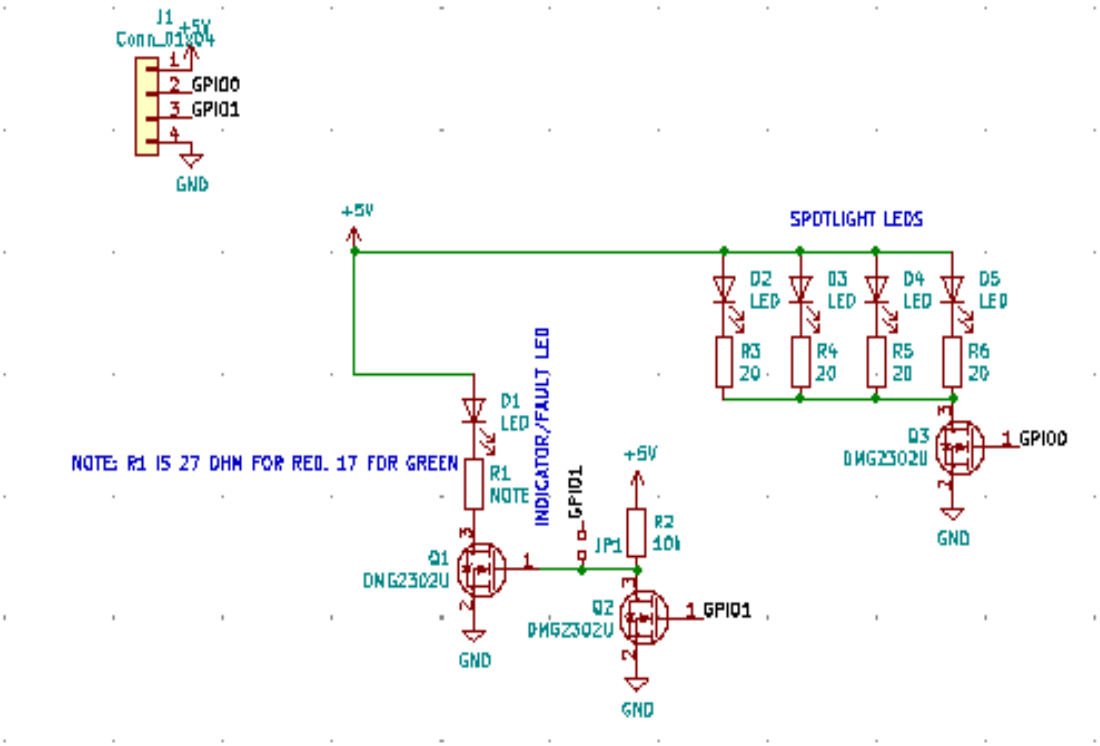


Figure 6.19: Indicator Board Schematic

Because the LEDs are closely spaced and consume approximately 1W on each board, thermal analysis was required to verify that the boards would not overheat during operation. The board was designed for thermal dissipation in mind, with large power planes on each side of the PCB and solid thermal coupling to the pads of the LEDs. To determine if external board-mounted heatsinks were needed for the design, a FLIR camera was used to measure the heat distribution on the board after five and ten minutes of continuous operation. The results, seen below, demonstrate that the board rose to a maximum of 64º after ten minutes of operation. Around 64º, the temperature rise was too gradual to accurately measure. This is within the operating point of the electronics used in these boards, and below the glass-transition temperature of the supporting 3D printed ABS plastic (105º).

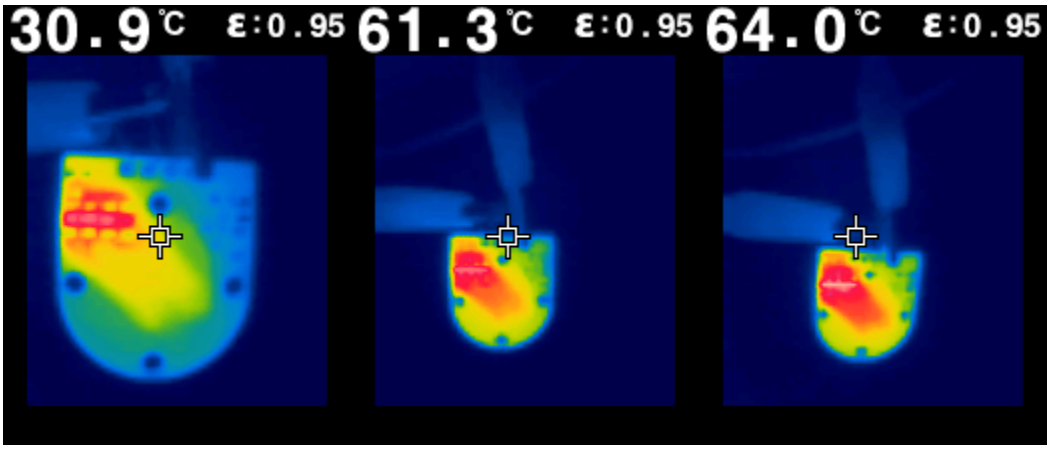


Figure 6.20: Temperature at T=0 minutes, T=5 minutes, and T=10 minutes (left-to-right)

6.5 Microphone and Speech Recognition

For the design of the embedded microphone device, the speech recognition module needs somewhere between 4.5-5.5V to operate on, while the MSP430FR6989 chip & the Xbee module need the VCC supply to be at about 3.3V. This means that a voltage regulator will be needed to ensure that the proper stepped down voltage from the LiPo battery will be distributed to the MSP chip and other components on the PCB. The major components on this PCB board can include the MSP430FR6989, an LCD screen along with some push buttons for the user interface, boost converter, voltage regulator, battery charger, and an Xbee Series 1 module that will wirelessly communicate with the Xbee module on the Odriod's communication board.

Figure 6.21 shows the actual handheld device and what the PCB will look like. Ideally, we want the device to be as small and lightweight as possible, as to be able to fit in the pocket of the officer, or the shirt pocket. The dimensions of the device shown in the figure is 4 inches x 3.5 inches (height x width) and should not exceed these dimensions. The LCD screen chosen for our design is about 3 inches in length, with a 1-inch height. There will be three push buttons, which each will have a different function as described later in this section. There is a charging port that is at the bottom of the microphone device, which will allow for USB connection from the Li-Po battery charger to the device.

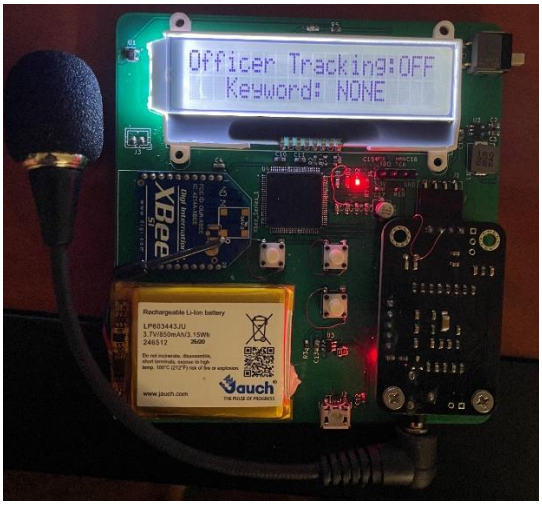


Figure 6.21: Handheld microphone device

6.5.1 Boost Converter

While the typical rechargeable LiPo battery voltage ranges between 2.7 volts while discharged to about 4.2 volts when fully charged. Typically, the nominal or average voltage of most lithium polymer batteries are 3.7V. This is still not enough voltage to meet the requirement of input voltage for the speech recognition module. Therefore, a boost converter will be used to allow the voltage of the LiPo battery to be “boosted” up to a value necessary to be the input for the speech recognition module. The boost converter that will be used is the TPS613222A, shown in Figure 6.22, which will give us an output voltage of 5V, with a current limit of 1.8A. The SW (switch) pin will be connected to the input voltage through a 2.2uH inductor, and a decoupling 4.7uF capacitor in parallel with our LiPo battery input voltage. On the output of the boost converter, there will be a 5V output, with a 0.4mA output current. Two 22uF output capacitors will be used and should be as close in proximity as possible to the VOUT and GND pins. The output capacitors are used to limit the output ripple.

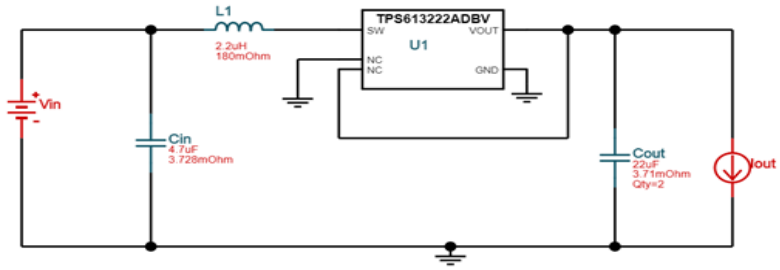


Figure 6.22: TPS613222A schematic from TI’s WEBENCH Power Designer tool

6.5.2 Voltage Regulation

A voltage regulator will be needed to step down the LiPo battery to about 3.3 volts so that many other components such as the xbee module, LCD, MSP430FR6989, and

boost converter on the PCB can be powered. The voltage regulator chosen for this design is the TPS73633 and it was chosen specifically because it is a great low-dropout regulator. The Li-Po battery input voltage can range anywhere from 2.7 to 4.2 V and needs to be regulated to about 3.3V. With a low dropout voltage regulator, we can ensure that regulation will occur when the input voltage is close to the output voltage. This is important because as previously mentioned, the Li-Po battery will eventually discharge and drop close to the 3.3V value needed to be regulated. The TPS73633 is a Texas Instrument voltage regulator that uses an NMOS pass element in a voltage-follower configuration. The regulator is stable even without the use of output capacitors, but they can still be used if desired. The voltage regulator will deliver very low dropout voltages (about 75mV to 200mV), along with a very low current consumption of 1uA when not enabled. This makes the voltage regulator a great choice since the embedded microphone device will be portable.

The 5 Pin voltage regulator has a NR (noise reduction) pin that can be connected to an external capacitor. This is optional as mentioned previously and can reduce the output noise to very low levels. The other pins and their functions can include:

- In: Input supply
- EN: Enable pin that turns on the regulator. Driving the pin low will put the regulator into shutdown mode, but EN can be connected to the input pin if not used.
- GND: Ground
- OUT: Output pin where the regulated voltage will be. No output capacitors are needed for stability

6.5.3 LCD and Push Buttons

The handheld microphone device that the police officer will carry will have an LCD and some push buttons for some interaction with the device. Ideally, we would like to implement about four push buttons on the device to navigate through the user interface on the handheld device. The user interface will include an assigned button that is in charge of turning on and off the system. This is the button that the officer will press at the beginning of the traffic stop, when they are getting ready to interact with someone they just pulled over. This push button is responsible for activating the camera system. At the end of the traffic stop, if the officer wishes to turn off the complete system, they could press the button again. Another button can be implemented to send a ping to the Odroids communication board and wait for an acknowledgment to ensure that there is a connection and communication between the two devices. This can be helpful during the testing stage, where we can ensure that over certain distances there is still clear communication between the microphone and the ODROID device. The third button will be responsible for activating the speech recognition module, and to begin the process of hearing for keywords.

The LCD planned to be used for the handheld microphone device is a NewHaven Display – C0220BiZ. The LCD dimensions are 75.70mm x 27.10mm x 6.80mm (about 3inch x 1inch x 0.27inch) and would be a perfect size for the handheld device. As mentioned previously, the LCD will operate off of the stepped down voltage from the Li-Po battery,

and needs an input voltage of about 2.7V – 3.5V. The display format is 20x2, which means that 20 characters can be displayed in each row for two rows. Each character has a size of about 5.55mm by 2.45mm, and there is a white LED backlight which makes the LCD super visible to the officer. The LCD is interfaced through I^2C and comes with a built-in ST7036i controller.

6.5.4 Overall Schematic for Handheld Microphone Device

Figure 6.23 shows the schematic for the handheld microphone device, with all the components mentioned in the 6.5 section. As seen below, the MSP430FR6989 has many pins, but not all the pins will be used. On the top left of the figure, we can see the boost converter that will feed 5V to the speech recognition module. Below that we can see the voltage regulation that will reduce the 4.2V Li-Po battery input voltage to a 3.3V output that will be necessary for other components. Other components also on the schematic can include the LCD and the Xbee Module. Both will communicate with the MSP430FR6989. The MSP430FR6989 has two communication channels for UART transmission, USCI_A1 and USCI_A0. The speech recognition module will be connected through Channel A0, and the Xbee device will be connected through Channel A1. The LCD communicates with the MSP via I^2C, and therefore will be connected through the USCI_B1 channel.

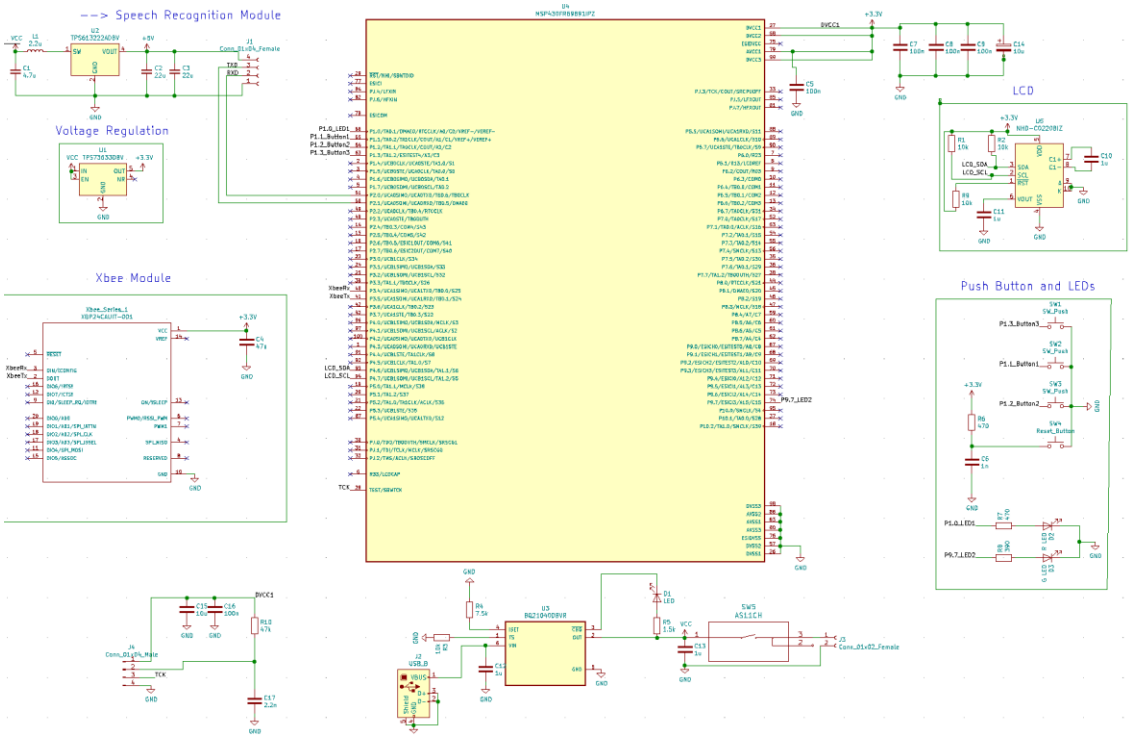


Figure 6.23: Microphone Device Schematic & PCB

7. Software Design

Acting as the brains of the operation, the software was an integral component to make the system function as desired. It is responsible for carrying out a variety of tasks in order to allow the system to capture the best footage of the officer. The software itself has a lot of components, and so it needed to be organized and efficient to minimize error between its components and maximize ease of integration with the physical hardware. To accurately track the officer, an object detection model needed to be trained that could distinguish police officers from other people. The camera needed to be integrated into the recording process and the logic portion that controls where to move the camera based on the officer's location in the frame. To give the officer control of the system, a communication protocol from the microphone device needed to be established and followed. This section will outline how each component of the software was designed and integrated into the system.

7.1 Object Detection Training Setup

The custom-trained object detection model is one of the central components to the project. It is the model's job to parse the incoming frames and determine if an officer is in the frame, as well as where in the frame that officer is. This information can be propagated through the entire system to ensure that the officer stays within frame for most if not the entire duration of the footage.

7.1.1 Training Environment

The goal of this section is to give an overview of the technological environment that was setup to do the training. It is not intended to be read as a guide on how to set up the training, but rather a benchmark that can be replicated in order to achieve similar training times. Below is an outline of the hardware specifications of the training PC as well as descriptions of the drivers and software installed in order to train the model.

7.1.1.1 Hardware Specifications

CPU:	GPU:	Memory:
Intel Core i9 10900KF 10 Cores @ 3.7 GHz	NVIDIA GeForce RTX 2080 Super 3072 CUDA Cores @ 1.65 GHz	2 x 16 GB DDR4 RAM @ 3200 MHz 8 GB GDDR6 VRAM @ 496 GB/sec

7.1.1.2 Drivers and Software

Ubuntu 18.04.5
This was the most up-to-date operating system that was compatible with all the software and drivers needed to train the model.
NVIDIA-SMI v. 450.51.06
Driver used to access the graphics card.

CUDA 11.0
Driver used to communicate with CUDA cores on the graphics card to perform computational operations.

Pycaffe w/ Python 3.6
Python wrapper for caffe (C++ deep learning library). This is used to execute training of the model.

GNU C++ Compiler v. 7.5.0
Compiler used to compile the caffe library.

Intel Movidius NCSDK
SDK for Intel Movidius chip on the camera. The SDK compiles the trained model so that it can be written to the camera.

FLIR Spinnaker SDK + SpinView
Software for communicating with the camera (will be discussed in detail in a later section). This software is used to upload the object detection model to the camera.

7.1.2 Dataset Creation

7.1.2.1 Requirements

One of the most crucial parts of any deep learning project is finding a good dataset to train the model off of. The first step was to determine the requirements that a satisfactory dataset would need to achieve. The ideal dataset would contain images both with and without police officers, some having multiple in the same image. Because the model would need to recognize officers in the middle action, it would have to be flexible enough on the orientation and pose of the officer in the image so that it could recognize officers that were not facing the camera directly and possibly moving. Lastly, multiple different environments/backgrounds would need to be present in the images to prevent the model from being too reliant on the officer's contrast to a specific background.

Finding a dataset that met the above requirements was necessary in order to achieve a model that was both flexible and accurate. Searching online for a pre-labeled dataset (a group of images that each have the training metadata already defined) did not yield much success. The closest datasets that met the requirements were for detecting people in images, however these datasets would falsely label a bystander as a police officer, and thus could not be used. In order to fix the problem of not being able to distinguish an officer from any other person, a custom dataset was created.

7.1.2.2 Data Recording

Datasets for deep learning can be very large, containing thousands of images that represent the data. To achieve such a large number of images, videos were taken and split into frames. To meet the requirements for the dataset, a script was created that would mimic a typical traffic stop and capture the officers moving towards the stopped vehicle as well as standing still when talking to the driver. Two officer costumes were used to act as visual definitions of a police officer to the model. Supplemental footage of each of the officers by themselves was taken to broaden the variety of poses and orientations of the officers in the dataset. The script was recorded with both a stationary and moving camera in multiple settings to increase the diversity in backgrounds present in the images.

7.1.2.3 Data Preparation

After recording the footage for the dataset, the videos were split up into frames, producing a dataset with just under 2000 images. The total number of frames in the video far exceeded 2000, however most frames were skipped because of the immense similarity between adjacent frames. While this decreased the size of the dataset, it made the dataset more spread and allowed for each image in the process to be processed more in the same amount of time.

With all the images extracted from the video, the next step was to label each image with the training metadata that would allow the algorithm to correctly identify police officers. A label making tool was used to generate all the labels for the dataset using the PASCAL VOC Format. Finally, the images and labels were compiled into an LMDB (Lightning Memory-Mapped Database), which was used as the dataset to train the model.

7.1.3 Training

To help decrease the overall training time, the images were resized to a resolution of 1280 x 720. While this did decrease the detail contained in each image, it allowed for more iterations to be used, which provided a larger benefit than the small extra detail that was cut out. With the training hardware specifications described in section 7.1.1.2, the model was able to run through 10 iterations in approximately 17.5 seconds. The model was trained multiple times on the initial dataset in different fashions. Below is a summary of each time the model was trained and what was observed of the model after the training.

Table 7.1 Object Detection Model Training Results

Training	Iterations	Observation	Training Time
With pre-trained weights	10000	Failed to distinguish bystander and police officer	~5 hours
From scratch	12000	Many false positives without a clear pattern	~6 Hours
From scratch	35000	Very few false positives, weak against similar-colored objects	~16 hours

Initially, a set of pretrained weights were used as a starting point for the model. These weights were trained to identify and locate over 20 different classes of images, one of which being a person. The idea behind using pre-trained weights is if the weights were trained to detect objects that are similar in nature to those that the model is currently being trained to detect, the model will be able to converge a lot faster. The problem that came up with the pretrained weights was that the model was unable to distinguish a bystander from a police officer. This can be heavily attributed to the pre-trained model's familiarity with a person and insufficient iterations to significantly overwrite previous ideas of what makes up a person (the pre-trained weights were created from 73000 iterations). The massive number of iterations needed to correct this was deemed not worth the time and effort, and so the model would be retrained from scratch.

The advantage of training the model from scratch was that it would have no prior knowledge, and thus was very sensitive to the dataset it was training off of. The problem of having no pre-trained weights became clear once the second model was tested. While it was able to identify police officers very well, it falsely identified random spots on the image as police officers that looked nothing like a person/police officer (a door for example). It was clear that the model did not have enough iterations to converge, and there was still room for random images to score well.

After many more iterations, the model improved significantly. False positives were few if any, and the model was very consistent at being able to differentiate a bystander from a police officer. A new problem came up, however, that pointed towards a flaw in the dataset (as opposed to how the model was trained). The model pretty consistently classified objects as police officers that were the same color as the uniform (e.g. blue plush toy, blue flag). This could cause the camera to follow blue cars or even the person being pulled over by the police if he/she was wearing a blue shirt. The dataset needed to be enhanced to fix this issue.

New images (and labels) were added to the dataset to provide information different from the images in the dataset up to that point. Previously, some of the images included police officers and some of them did not. An aspect that was missing, however, was a more surefire way of determining what was not an officer. All of the blue regions in the photos were police officers and so the model learned to strongly associate the color with police officers. Images were added to the dataset that contained other blue items that were not labeled as a police officer. Half of the new images had both blue items and police officers and the other half had only the new items. Because the model couldn't just rely on color anymore to point out an officer, it had to learn some of the actual features on top of the color that signify a police officer, including the human-like shape and hat. This required the model to be trained longer, but ultimately led to more accurate predictions overall.

7.2 Device Communications

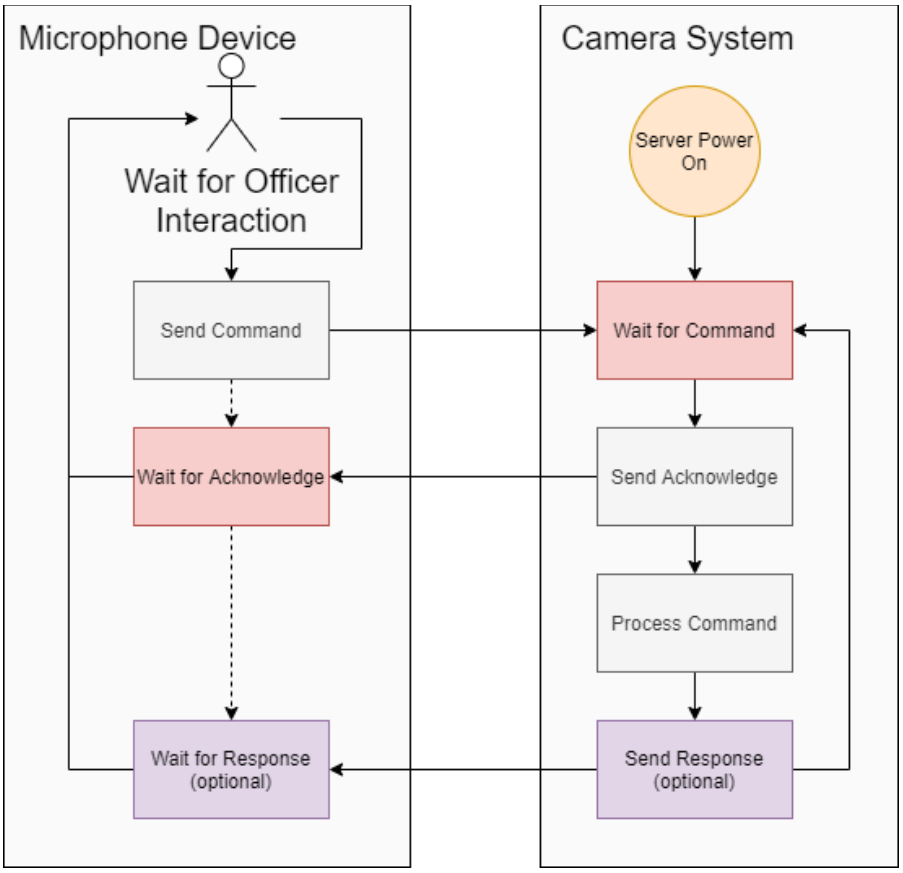
Without a proper communication setup and protocol, the different components of the system would not be able to interact with each other, and thus be unable to function. Each component provides vital information and essential functionalities to the other components and a robust standard for communication was developed to administer control between them. Bytes will be sent back and forth over serial to act as messages and conversations between the devices. The next two sections will go over the two main responsibilities that the communication system assumes.

7.2.1 Officer-System Interaction

The primary responsibility of the communication system is to administer the officer's interaction with the system. Because of the camera system's fixed location on top of the vehicle, the officer will be unable to physically interact with the camera system directly. This opens the door for the microphone device to be the bridge of communication between the officer and the system. There are a few operations within the camera system that require input from the officer to trigger. The first and most obvious is to start the camera tracking and recording of the officer. If the system was always attempting to track the officer and record footage, then the SD card would fill up quickly and a lot of power would be wasted moving the camera in search of the officer while the officer was inside the vehicle. Allowing the officer to start the system will minimize the amount of wasted power and storage space consumed by the camera. It follows naturally that the second task would be to stop tracking and recording the officer. Finally, the microphone device will need to be able to send its detected keywords over to the camera system to ensure that the keyword is recognized and processed.

To achieve these responsibilities, the camera system will act as a server that receives and processes commands from its client, the microphone device. Each operation will be tied to a command that the microphone device will send over to be processed by the camera system. The overall flow of the camera system side is shown in the figure below.

Figure 7.1



Summary of interaction between camera system and microphone device. Red blocks indicate where the software is blocked, waiting for an I/O event to occur. Purple blocks are executed depending on the command.

An important piece of the communication flowchart is the command acknowledgement sent by the server. Wireless systems always have the possibility to go down, so every system should be prepared for these cases. This system is particularly vulnerable to these random shutdowns since the client (carried by the officer) could potentially move very far away from the vehicle and can be obstructed by obstacles such as trees and other vehicles. By sending an acknowledgement response after each command, the client will be able to ensure that connection to the server was not lost and the system is processing the command. If for whatever reason an acknowledgment was never received by the client, the client can alert the officer that the message didn't go through.

7.2.1.1 Command Format

The format for the commands is very simple. The first byte sent will specify which command the client is trying to send to the server. Depending on the command, the client will send additional bytes to provide supplemental information needed to process the command. In the list below, BIT 0 represents the least significant bit and BIT 7 represents the most significant bit.

Byte 0: Command Specifier
BIT 0-3: Command Enumeration Value
BIT 4-6: Number of additional bytes (possible values: [0, 7])
BIT 7: Device Specifier

Byte 1+: Optional Command Arguments (command-specific format)

The command acknowledgement will be the following:
BIT 0-3: 1111
BIT 4-6: 000
BIT 7: 0 or 1 depending on device

7.2.1.2 Handheld Command List

Table 7.2 Handheld Commands

Name	Bytes	Description
Ping	Byte 0: 0x01	Sends a ping to the server. Aside from acknowledging the command, the server does nothing.
Start Camera Tracking	Byte 0: 0x02	Starts recording frames and tracking the officer. If the system was already started, this command does nothing.
Stop Camera Tracking	Byte 0: 0x03	Stops recording frames and tracking the officer. If the system was not running, this command does nothing.
Send Keyword	Byte 0: 0b0xxx0100 Next xxx Bytes: Keyword	Sends a keyword to the server for processing.

7.2.2 Motor Control

Moving the camera according to the officer's location within the frame is the job of the motor control. It utilizes a UART protocol to send and receive command bytes between

the ODROID and motor FPGA. Each movement involves a vertical and horizontal rotation. Two different types of movements are defined by the protocol: relative and absolute. Relative movements move the motors a specified angle from their current angle. This is useful while the camera is following the officer, since it only knows how far the officer is from its current position. An absolute movement will move the motors to a fixed position relative to a predefined limit switch on the gear. When resetting the system, this can be used to reposition the camera at its start location. In addition, each movement can happen synchronously or asynchronously. When performing a synchronous movement, the motors will send back a response when they reach the specified destination. This is especially useful when done with absolute movements, so that the ODROID can guarantee that the camera is at the specified position before further commands are sent. An asynchronous movement will not send any response when reaching the destination. The advantage to this is that the ODROID does not have to sit and wait for a response from the motor before sending another command. The table below outlines all the commands that can be sent to the motor control. The response column does not include the acknowledgement byte that is sent upon receiving each command.

7.2.2.1 Motor Command List

Table 7.3 Motor Commands

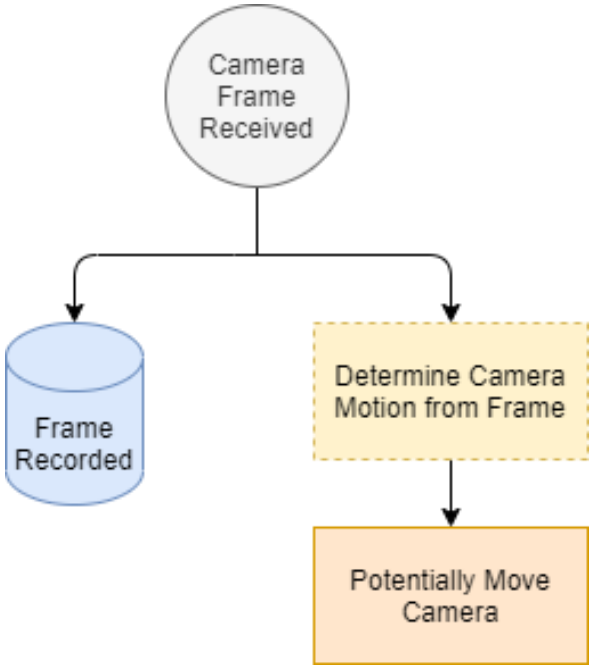
Name	Bytes	Response	Description
Ping	Byte 0: 0x81	None	Sends a ping to the motors. Aside from acknowledging the command, the motors do nothing.
Relative Move Synchronous	Byte 0: 0xE5 Bytes 1-3: Horizontal Move Bytes 4-6: Vertical Move	Success: 0x81 Fault: 0x82	Moves the motors a certain angle relative to their current location. Sends a response when the destination is reached.
Relative Move Asynchronous	Byte 0: 0xE6 Bytes 1-3: Horizontal Move Bytes 4-6: Vertical Move	None	Moves the motors a certain angle relative to their current location.
Absolute Move Synchronous	Byte 0: 0xE7 Bytes 1-3: Horizontal Move Bytes 4-6: Vertical Move	Success: 0x81 Fault: 0x82	Moves the motors a certain angle. Sends a response when the destination is reached.
Absolute Move Asynchronous	Byte 0: 0xE8 Bytes 1-3: Horizontal Move Bytes 4-6: Vertical Move	None	Moves the motors a certain angle.
Activate	Byte 0: 0x89	0x81 when activation finished	Activates the motors.
Deactivate	Byte 0: 0x8A	None	Deactivates the motors.

Set Speeds	Byte 0: 0xAB Byte 1: Pan Speed Byte 2: Tilt Speed	None	Sets the speed on the pan and tilt axis.
Headlights	Byte 0: 0x9C Byte2: 0b000000XY	None	Sets the state of the headlights to X and Y.

7.3 Camera SDK Integration

Flir provides a handful of SDKs for communicating with their cameras, the most useful for this project being the Spinnaker SDK. This library contains all the low-level code to connect to the camera, grab frames, and read the object detection results on each frame. This section will go over how the SDK was used in the project to satisfy the requirement of providing directions to move the officer within the frame. Below is a flowchart that outlines the two primary tasks that the SDK was used to complete.

Figure 7.2: SDK primary tasks



Every time a new frame was processed on the camera, it was grabbed from the buffer and then copied so that it could be used elsewhere in the program without clogging up the camera buffer. Each frame was saved to a single avi file inside a Micro-SD card to provide full footage of the scene. On top of recording the frames, the camera library occasionally had frames investigated to determine where the officer was in the image (and if the officer was even present in the images all). Not all frames were used to guide the camera. Some frames were ignored in order to reduce the number of different movements that the camera made. Less movements meant a less frantic movement pattern, leading to better quality camera footage. The number of frames to ignore was

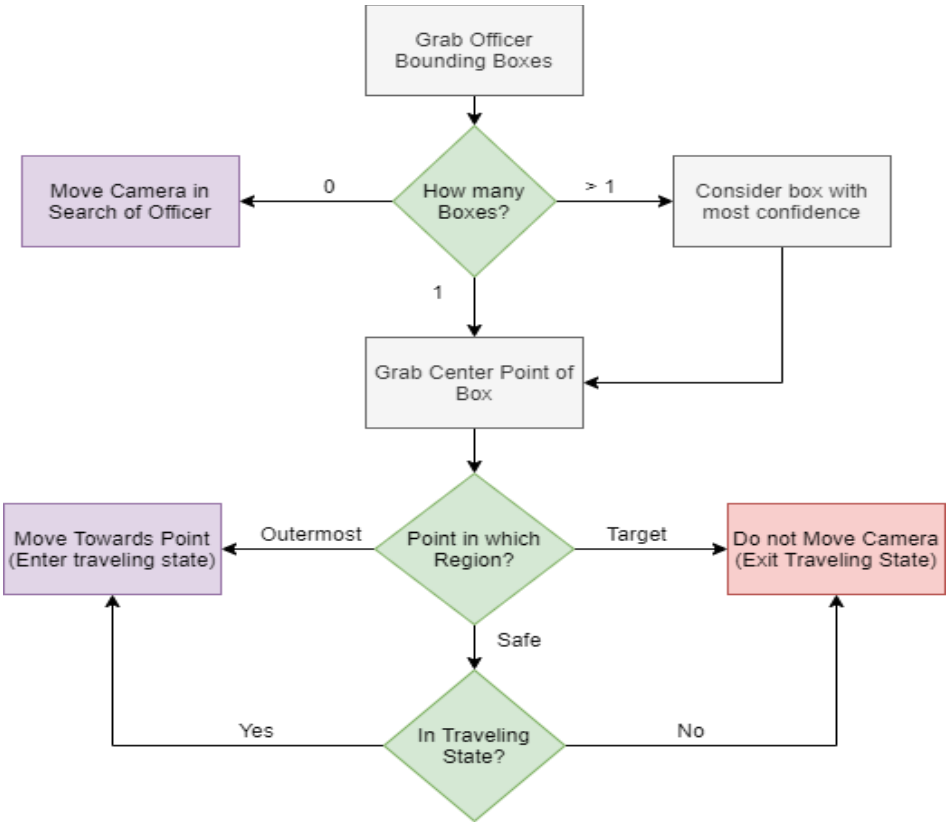
considered carefully, as ignoring more frames meant a slower response time for the camera, something which needed to be pretty low in order to not lose sight of the officer in moments of rapid movement. Additional measures and strategies were used to maximize the quality of the footage and minimize the response time of the camera that will be discussed further in the next section.

7.3.1 Frame Analysis

The purpose of the frame analysis was to determine where the officer was in the image and where the camera needed to move in order to better position the officer in the frame. While the computation-heavy work of processing the image through the object detection model was already done on the camera itself, there were still things that needed to be determined in order to make the camera movement as smooth as possible. This section will cover the cases where at least one officer was found in the frame. The camera's response to frames containing no officers will be discussed in detail in the next section.

There were too extremes to take the frame analysis part of the code. The first was the simplest and would just involve always commanding the camera to move such that the officer is dead center in the frame. While this keeps the officer centered in the frame as much as possible, it adds too many slight adjustments to the camera position that lead to very shaky and poor quality footage. As long as the officer is close to the center frame (not necessarily in the center), the camera will capture all the footage it needs while maintaining a more consistent and steady movement pattern. This was the basis for the approach used to dictate the camera motion outlined in the figure below.

Figure 7.3



Summary of frame analysis. Purple boxes indicate actions that lead towards moving the camera. Red boxes indicate no motion required.

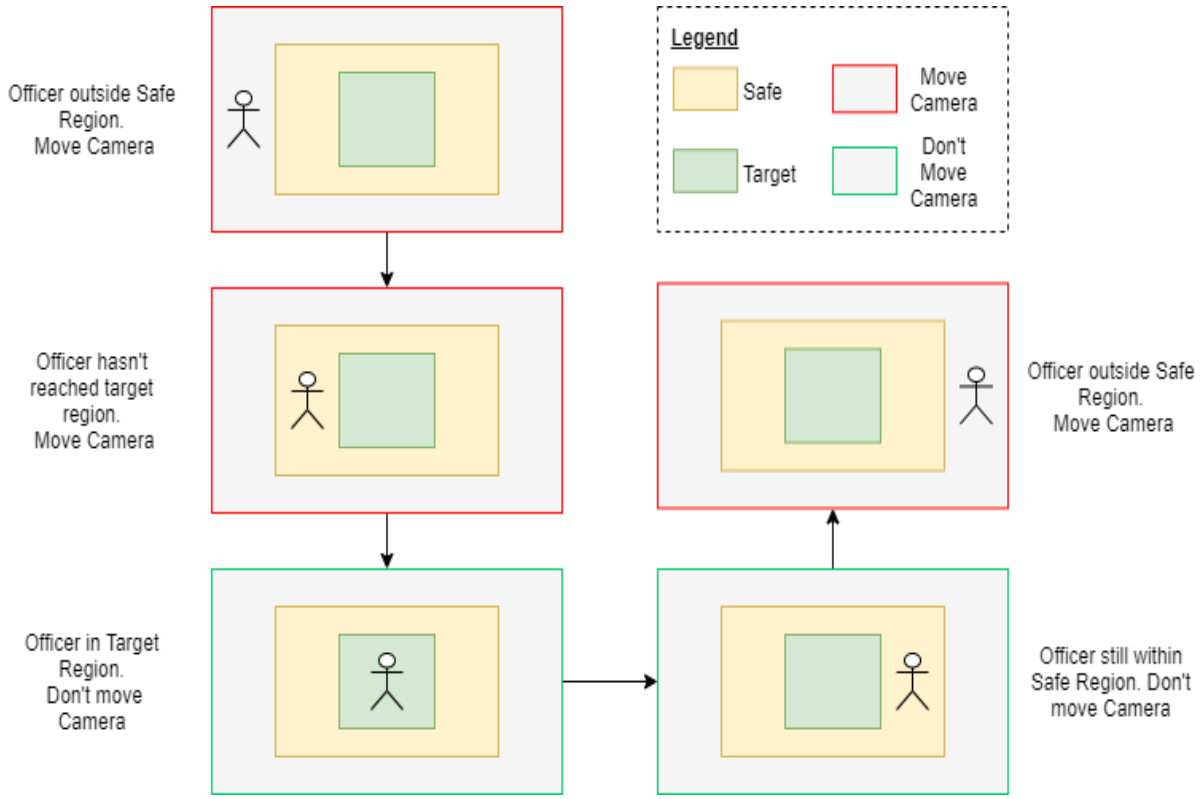
Each frame that the camera sent over contained metadata that housed the results of the object detection performed on the image. The object detection model would produce boxes around areas in the image that it thought were police officers. With each box was a confidence value that could be used to gauge how likely the prediction actually was to be correct. It is possible for the model to make mistakes and perceive miscellaneous objects as police officers. Further training the model reduces this chance, but regardless of the procedure used to train the model or the time spent, there will always be a few edge cases that the model misinterprets as a police officer. The frame analysis hurdles around this by only considering the bounding box with the highest confidence value associated with it. The police officer's location is simply the center of this box. This same strategy implicitly solves the problem of what to do when multiple officers are in frame. A possible alternative would be to average out the positions of each officer, possibly adding a minimum confidence value to be considered in the averaging. While this approach could work in a number of cases, it is very sensitive to misinterpretations by the model, especially when combined with a failed recognition of the officer.

Noise, defined in this case to be frequent slight changes of the bounding box surrounding each officer that each deviate slightly from the true position, could cause the camera to adjust frequently. This will cause the footage to be unsteady and low quality. A potential counter to the noise would be to not look at the officer's immediate position, but a moving average of the previous positions. A moving average takes the last few immediate

positions and averages them together to form what it believes is the true position. The more points that are considered in the average, the more points it will take to shift the camera's perception of the officer's position in the frame. This would minimize the effect that one small outlier would have on where the camera moves to. Unfortunately, there is a downside to this solution that lies in the very aspect that makes it so good at its resistance to noise. Because it takes more points to significantly shift the camera's perception of the officer's location, it will take longer for the camera to react to actual changes in the officer's location in the frame. Modifying the rate at which frames are interpreted and the number of positions considered in the moving average will change both how responsive the camera will be and how resistant it is to noise. The fatal issue was that the more the parameters were adjusted to bring the response time to a reasonable amount, the more slight camera adjustments would be made resulting from the noise.

To prevent these excessive camera adjustments, regions were defined on the image that were used to dictate whether the camera should move to that point or stay stationary. Each region was defined as a rectangle centered in the image with a different height and width that were proportional to the height and width of the image. The innermost region, denoted as the target region, was the ideal region for the officer to be located in. When moving, the camera would continue to move until the officer landed in the target region of the frame. Unfortunately, this region alone does not solve the problem of minimizing the small camera adjustments. It only relocates the area of interest from the center of the image to the border of the target region, meaning instead of the camera constantly moving the camera so the officer was centered in the frame, it would move back and forth across the target region border. A middle region, denoted as the safe region, was added to fix this problem. This region stretched out farther than the target region, but not across the entire frame. When the officer was located outside the safe region, the camera would enter "travelling state" and move the officer through the safe region and into the target region. Once the officer reached the target region, the traveling state would end. If the officer then exited the target region into the safe region, the camera would recognize that it is not in a traveling state and not move. This movement process is depicted in the figure below.

Figure 7.4



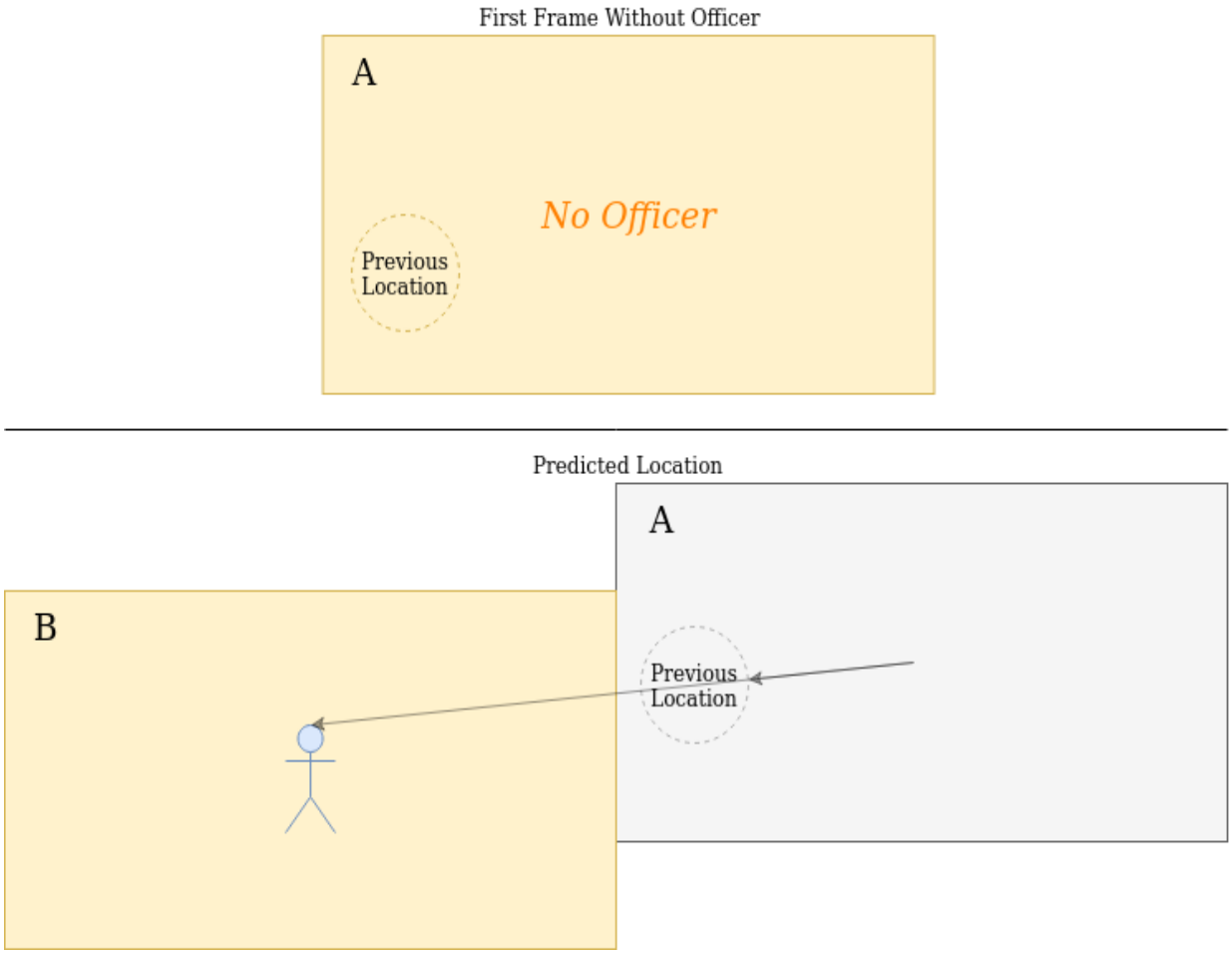
Visualization of interpretation of camera frames as it relates to officer location.

7.3.2 Officer Search

Applications that involve following an object in an image seem straightforward at first. Move so that the desired object is in the center of the frame. An oversight to the problem of object tracking in images is what to do when the desired object is not in frame. Without the object in the frame, the image provides no information as to where to move next. A well-designed system will have measures in place so that even if the object cannot be seen, it can be searched for in a logical way. The smarter the system is at reacting towards these situations, the faster the system can resume following the object.

Police officers have to sometimes react quickly, causing rapid movement changes that our system will inevitably be unable to keep up with. Our system assumes that a failure to find the officer in frame is most likely due to a rapid movement made by the officer. When the system first sees a frame without an officer, it looks at the last position the officer was located within the frame and projects the camera in the same direction. One concern with this approach was that if the officer was not in fact located along that direction, then the camera would be stuck moving along a path indefinitely that would never find the officer, unless the officer personally moved back into frame. A cap on how far the camera would look in that direction was placed to prevent this from happening. The cap allowed the camera to move to the point where the view of the camera at the cap was entirely different than that at the start of the movement (i.e. no pixels from one view mapped to a pixel in the other.

Figure 7.5



Depiction of where the camera will move when it loses sight of the officer. The letters in the top left are used to identify the frames in each picture.

The only case this does not account for is the one which an officer has yet to be found. This case will usually come up right after the officer initiates the tracking, and the prediction is that he/she will be inside or getting out of the car at this time. A home position was configured that points right outside the door of the vehicle. If the officer does not appear outside the door (potentially caused by a late start or outside of the ordinary movement pattern by the officer), then the camera will simply rotate horizontally in a circle until the officer is found. This is intended as the last resort solution, since while it does allow the camera to see all over, it may take some time before it can latch onto the officer again.

7.4 Embedded MCU Firmware

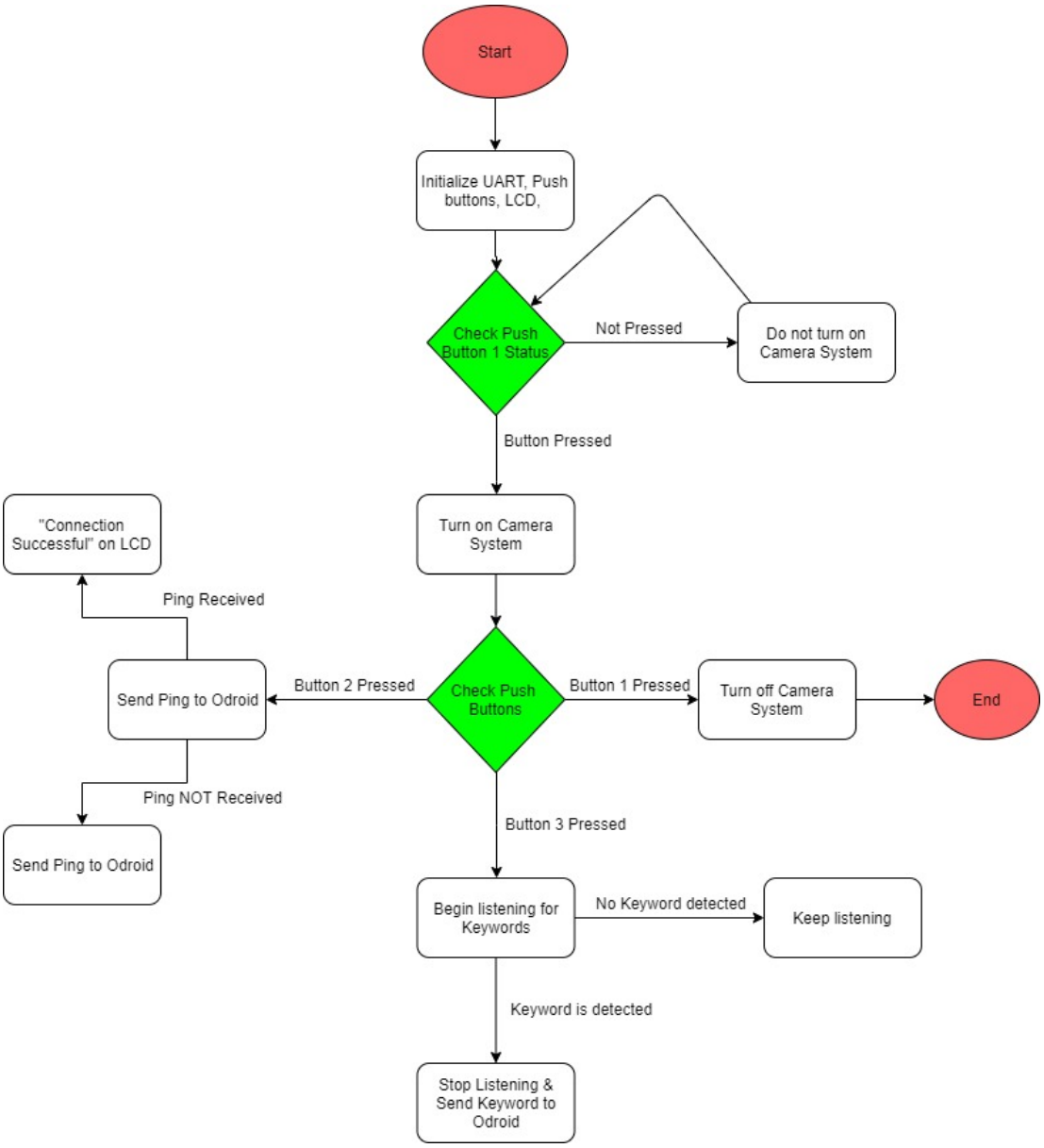


Figure 7.6: Embedded microphone software diagram

The idea for the code for the embedded microphone device can be illustrated by a sample software block diagram through Figure 7.6. The MCU will communicate with the speech recognition module through UART, and therefore we will initialize and configure the UART transmission. We will also have to initialize the use of the LCD, and three push buttons. The code will then be used to check if push button #1 has been pressed. This push button is responsible for starting and stopping the camera system when the button is pressed. When the camera system has been turned on, the MCU will wait or any other push buttons to be pressed. If push button number 1 has been pressed again, the system will turn off. If push button number 2 is pressed, a ping will be sent to the Odroid. The

Odroid is then responsible for sending an acknowledgement back to the MCU, and if received, the LCD on the handheld device will display “Connection Successful”. This will give the user an indication of whether or not they are connected to the camera system and in range for communication. If Push button 3 is pressed, this will begin the speech recognition operation of listening for keywords. The LCD will display “listening for keywords”, and if a keyword is detected, the device will stop listening and send a message to the Odriod signaling that a keyword has been detected.

7.5 Programmable Logic - Verilog

The FPGAs in the system are configured using different toolchains specific to the hardware involved. For the motor logic Xilinx Spartan 7, the Vivado Design Suite is used for programming and configuration/debugging. The Platform Cable USB II programmer is used for the USB-to-JTAG interface for system programming and debugging. The UART Bus manager utilizes the Diamond development environment from Lattice Semiconductors paired with their version of the USB-to-JTAG programmer device.

7.5.1 UART Bus Manager

UART communication is the standard interface for interconnectivity in the system. UART is used to communicate with the XBee radio that connects to the wireless microphone device, and it’s used for communication with the motor FPGA. This requires an intermediary device, a Lattice MachXO2 FPGA, to facilitate multi-slave communication with the host Odroid SBC. UART does not natively support multiple slaves on one bus in its simplest implementation. To achieve this, the Lattice FPGA uses three separate channels with logic to queue messages for the host based on slave message priority and arrival time.

In both FPGAs, at least one UART transceiver is required to be implemented. This looks like a hardware interface with a port for writing a byte at a time as well as reading a byte at a time. The inputs to the transceiver are the RX signal, the flag clear signal, and the byte-wide transmit buffer. The outputs to the transceiver are a byte-wide receive buffer and an RX flag signal. A state machine of the logic for every received byte can be seen below. The receiver is synchronous to the system clock, but counts the baud rate-specific clock pulse width of each bit by dividing the system clock by 104.

In order to reduce the error-rate of the protocol, the receiver implements oversampling on every bit reading of the state machine. This samples the RX signal line 7 times during each non-transition period, and adds the sum of each sample to a register. At the 8th period, or when the bit is supposed to transition, the sum of the samples is compared. If the 7 samples equal a value greater than 3, then the saved sample is a 1. If they’re less than 4, then the saved sample is 0. This averages the samples and reduces the chances of a transient or noisy signal from affecting the received data.

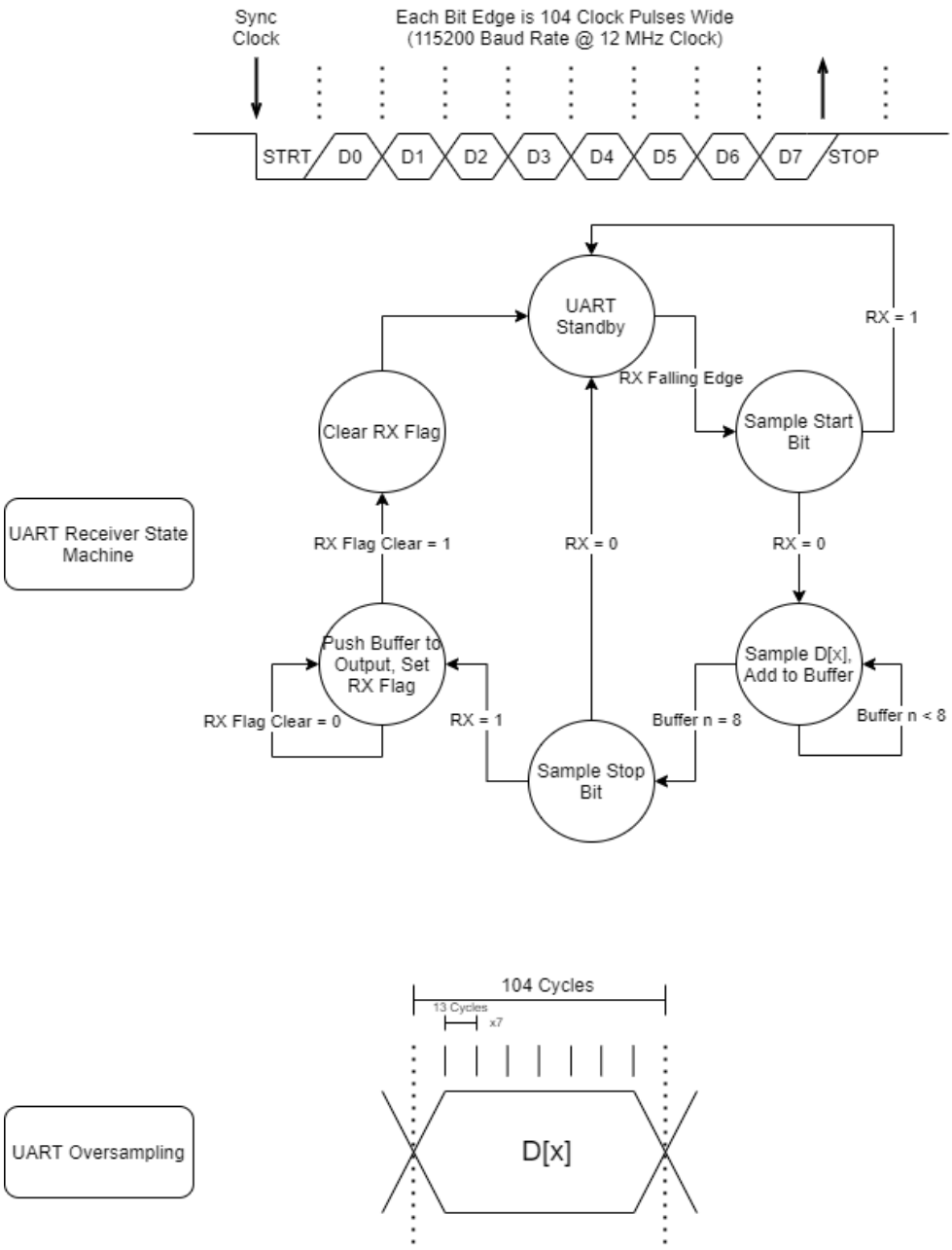


Figure 7.7: UART Protocol, Receiver State Machine, and Oversampling

7.5.2 Motor Logic

The motor controller uses a number of logical structures to move the stepper motors in a precise, accurate, and fast manner. There are three key modules embedded in the motor controller FPGA: a UART transceiver and a pair of motor movement modules, one for each axis. The interconnect for this construction is shown in the following diagram.

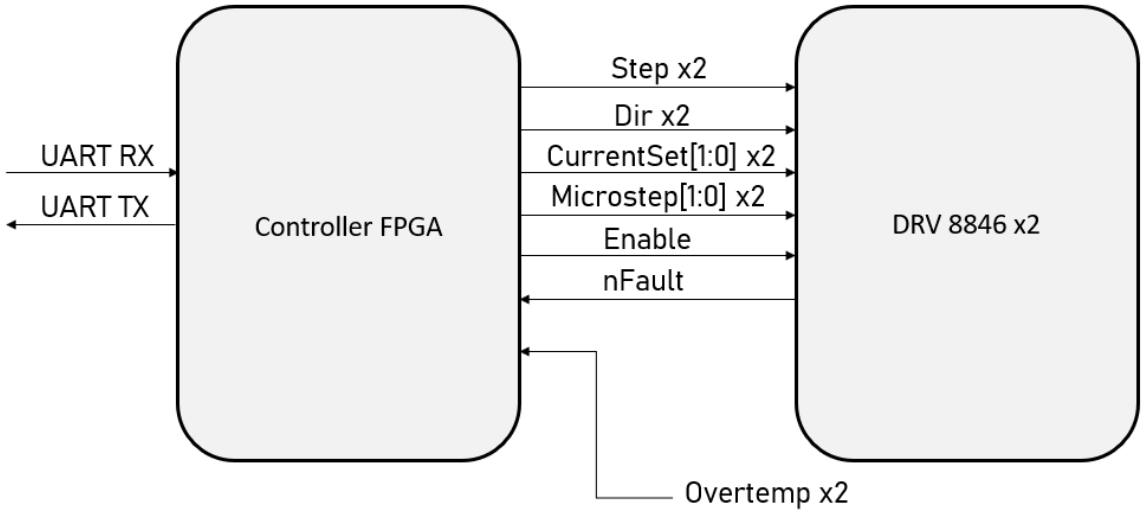


Figure 7.8: Controller-Driver IO Interconnect Diagram

The overall goal for the motor controller FPGA logic design is to enable accurate and fast movement in spite of potentially aperiodic, rapidly arriving movement commands on the UART bus. With a 12 MHz system clock, the Xilinx Spartan 7 has the ability to manipulate many data streams within the timing constraints of the system design. Paramount among the data processing that must be done on the FPGA is the generation of acceleration profiles for the stepper motors. An acceleration profile is a timing scheme for acceleration inputs to the motors that generate a smooth commutation pattern with no jittering or missed steps in the motor movement. This is important in stepper motors because their movement is dependent on discrete pulses with no feedback in place to determine whether or not the motor is in the position commanded by the driver/controller combo. A missed step is an accumulating error for subsequent movement, and must be avoided at all costs in the system design.

The most efficient solution for avoiding missed steps and jitter is the use of trapezoidal velocity profiles between commanded positions. A trapezoidal velocity profile has a period of constant acceleration (ramp up), a period of constant velocity (coast), and a period of constant deceleration (ramp down). The acceleration values and constant velocity value is determined by the physical properties of the system, to include rotational inertia and max motor speed. The timing of the acceleration inputs generates this profile, and the duration of each portion determines where the final position of the motor ends at. This is the case because the position of the motor at any point in time past T=0 is the integration of the motor velocity over time.

To keep track of where the motor is at any point in time, the digital design incorporates a number of integrators. This allows discrete-time tracking of the motor position, velocity, and acceleration in incredible precision at a per-clock-cycle basis. The integrators take the form of accumulator registers in the FPGA. For the position data, a 56-bit wide register stores fractional position data, with the upper 24 bits delineating physical position and the lower 32 bits defining fractional position, which is needed for per-clock cycle position updates at a 12 MHz clock. The clock for the accumulator is really a 1.5 MHz clock, however, explained in more detail later. A wire net connection in the digital design connects the lowest significant bit in the upper 24 position bits to the physical step output for the respective motor driver, producing a level-change (1/32 microstep movement) at every change in value of that bit. Current velocity is added to the position accumulator at every cycle of the motor dynamics clock. This data is stored as its own 26-bit wide accumulator, with a peak value defined as a parameter of the Verilog module. The velocity accumulator itself is updated with the current acceleration setpoint at every cycle of the motor dynamics clock. Acceleration is stored in an 8 bit wide register, with values constrained to max acceleration and zero acceleration. A number of 1-bit registers define the direction of the acceleration and velocity, as well as states of movement.

The acceleration profiles are generated as soon as a movement command has been received and parsed from the UART transceiver. Movement commands take the form of 7-byte transmissions, with the first byte delineating the command to move and the subsequent 6 bytes providing 3-byte-wide position data to both axes. Each axis movement module receives their respective position data, and the acceleration profile generator state machine takes over from there if the data is considered valid (within the position constraints that are defined for that axis). The reason the motor dynamics clock doesn't run at 12 MHz is based on the fact that the acceleration profile needs to be generated even if the motors are currently moving. If everything ran synchronous to the same clock, the acceleration profile would be invalid by the completion of its dynamics calculations - the numbers for position and velocity would have changed between the beginning of the algorithm and the end. This is avoided by dividing the motor dynamics clock by 8, giving the acceleration profile generator state machine 8 clock cycles between changes to the position and velocity accumulators of the motors.

In those 8 clock cycles, a number of decisions need to be made: Is the current velocity headed in the right direction? Is the motor going to overshoot the desired position even if deceleration is applied immediately? Will the peak velocity value be achieved in the distance between current position and desired position, or will a new lower peak velocity be required for the trapezoidal movement to not overshoot the target position? These calculations and conditional statements are performed in a sequence defined by a fairly complex state machine. The state machine has several loop-back conditions for situations where the acceleration profile generation will not be able to arrive at the desired location with the chosen velocity and acceleration parameters, allowing the motor to come to rest before recalculating the desired profile. In order to guarantee no change in position or velocity registers during computations, the state machine, in its initial state of 0, has a

condition that the first state transition can only occur immediately following the rising edge of the motor dynamics clock.

The state machine for acceleration profile generation is fairly complex, so a state-transition diagram is more apt for describing its operation. The following diagram is a mild abstraction of the actual state machine but is useful for describing how it works.

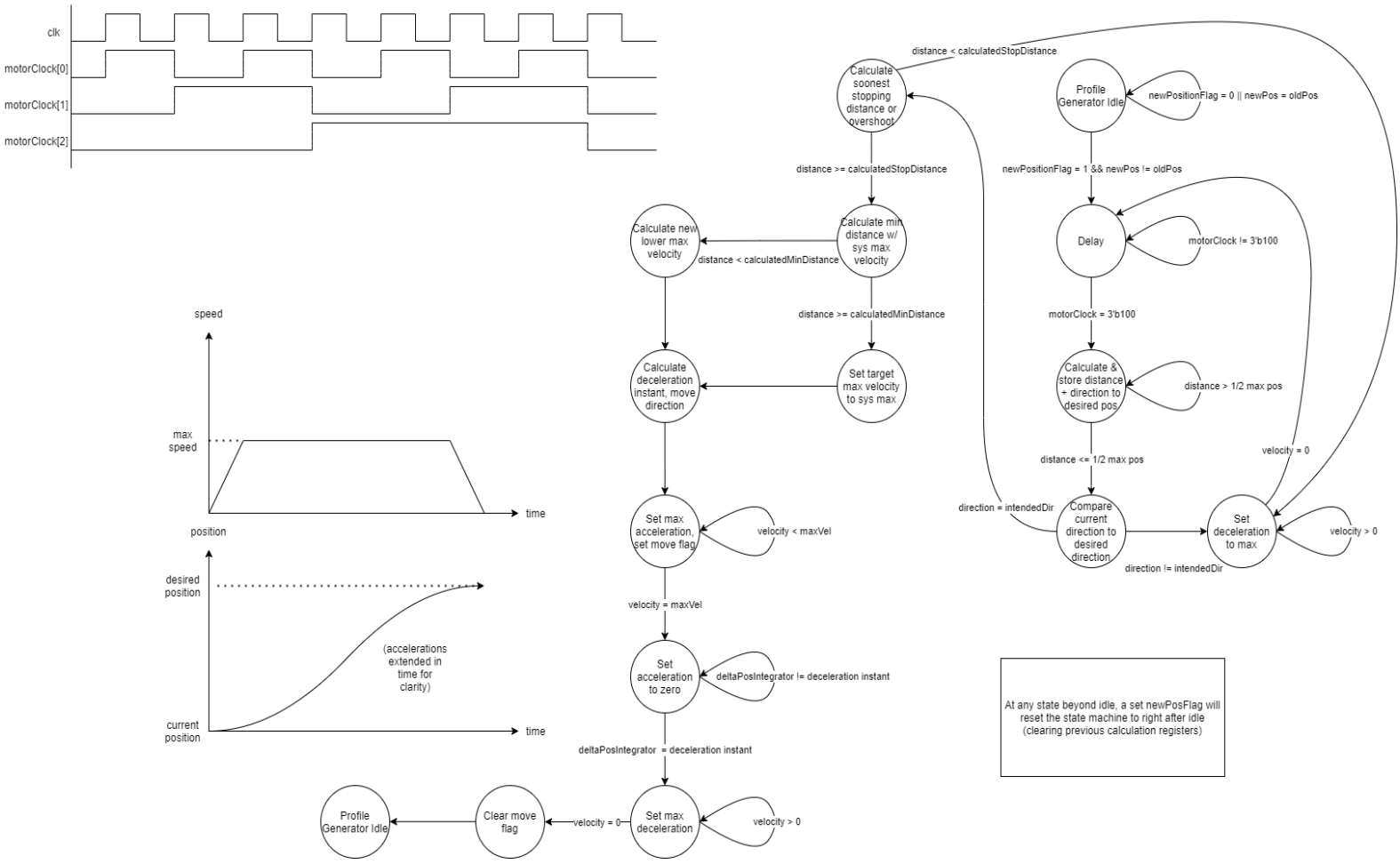


Figure 7.9: Acceleration Profile Generator State Machine

Within Vivado, a logic analyzer debug probe IP can be instantiated in the target FPGA. This is used to evaluate the internal state machine performance during testing, and a snapshot of such testing can be seen here with some of the aforementioned calculations and accumulators visible.

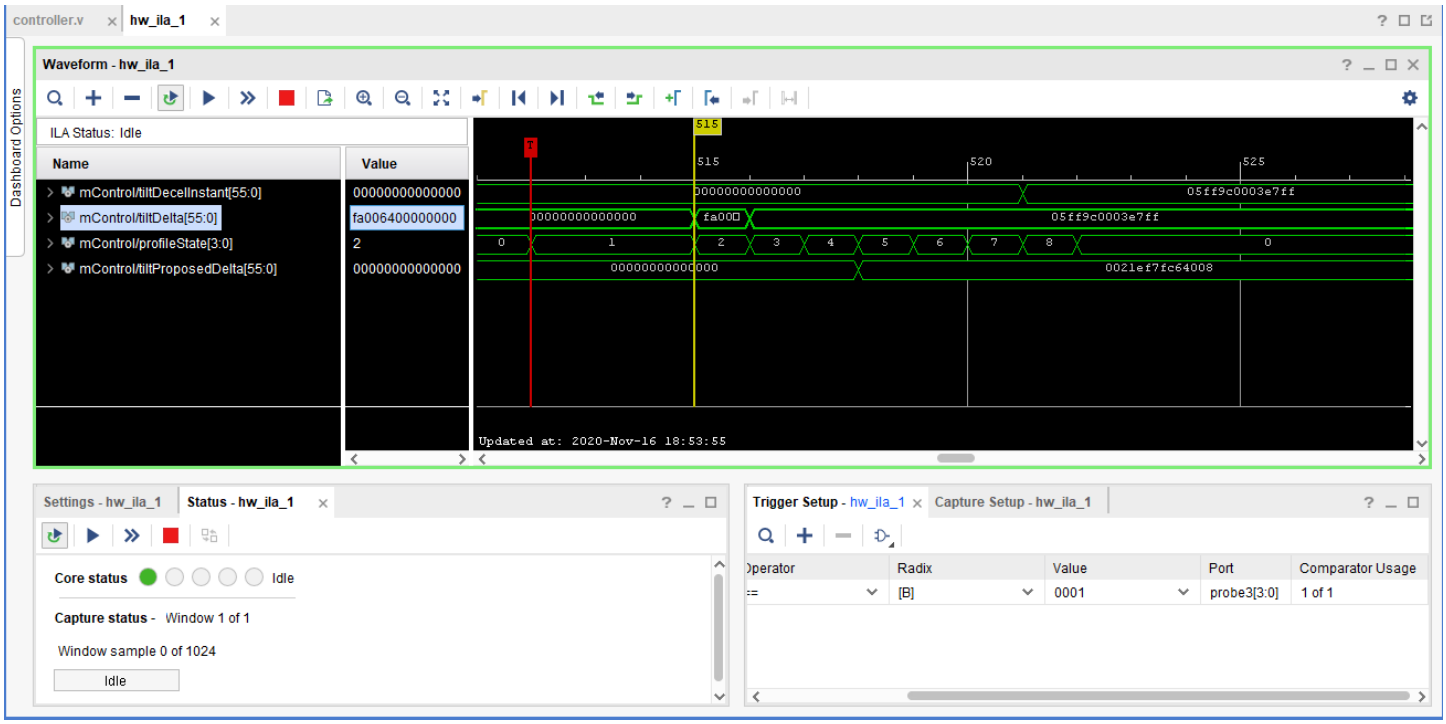


Figure 7.10: Xilinx Logic Analyzer IP Debug Window

Use of an FPGA is vital for the speed at which these calculations are taking place. The ability to parallel hardware inside the chip makes per-clock-cycle computations achievable, including the calculation of square roots necessary for some of the dynamics calculations of the profile generator. The end result, from the perspective of the motor driver board, is a pulse train of varying frequency proportion to the current velocity determined by the FPGA. On an oscilloscope, the smooth trapezoidal velocity profile is visible on the step outputs as a pulse train beginning with a low frequency, rising in frequency, maintaining constant frequency, and declining in frequency until no state change is observed.

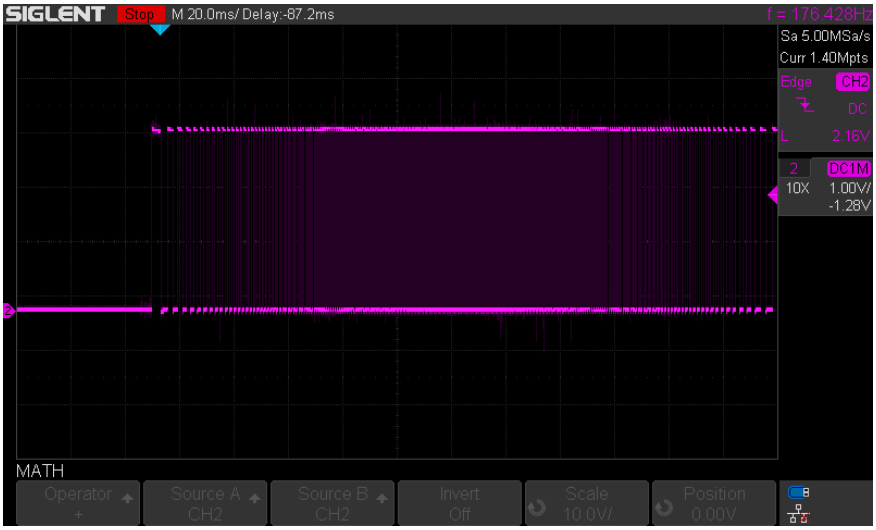


Figure 7.11: Short Duration Movement Pulse Train on Oscilloscope

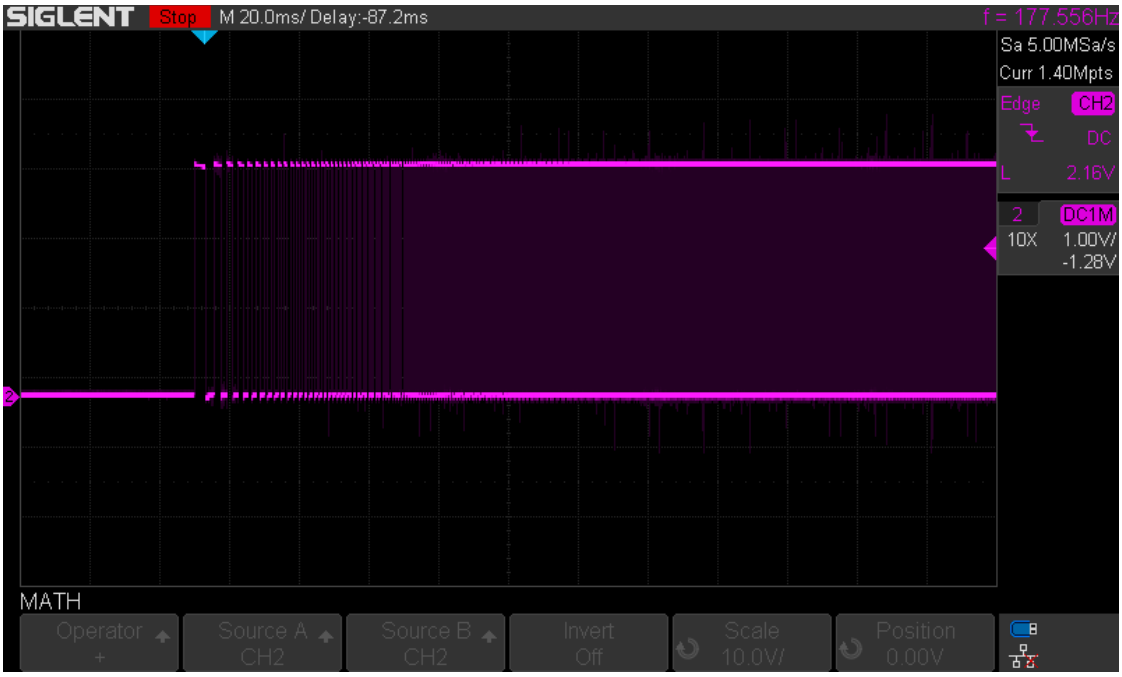


Figure 7.12: Long Duration Movement Pulse Train on Oscilloscope

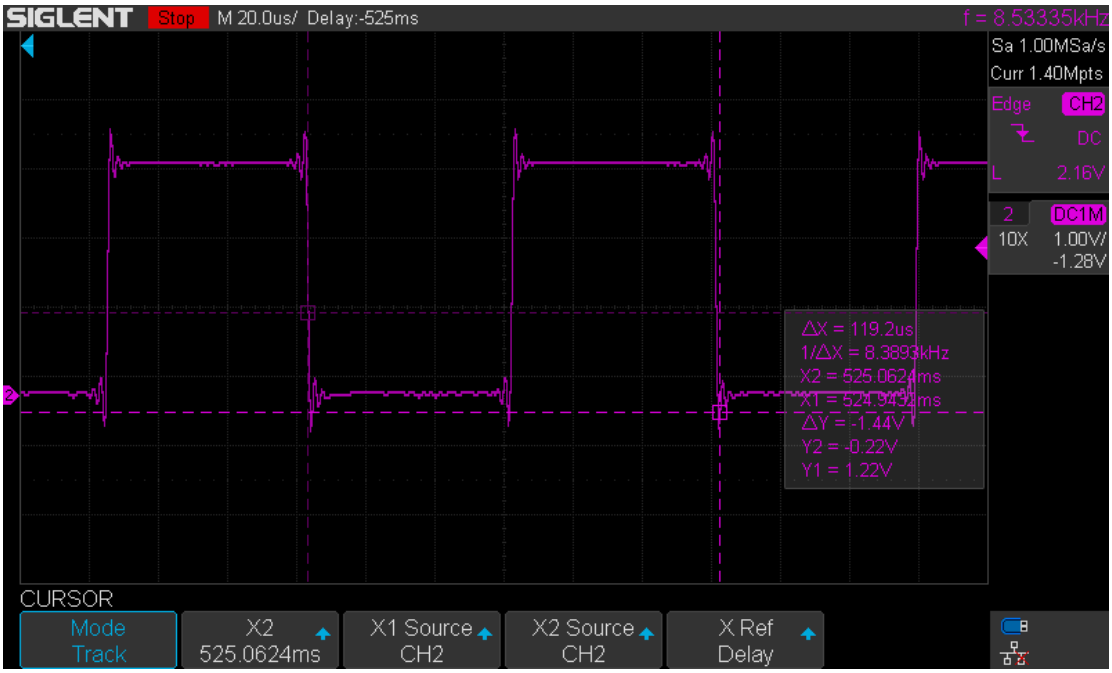


Figure 7.13: Close-Up of Pulse Train at Constant Velocity (Frequency) Coast Phase

8 Fabrication + Integration

8.1 PCB CAD

Creating a PCB is one of the necessary requirements for the Senior Design course. By designing well structured and clean PCB's, we can easily solder key components onto our PCB and ensure that the connections are made correctly. Part of designing a PCB is creating a schematic and ensuring that you pick the proper components and elements for your design. There are tradeoffs, as you will have to account for price and application of certain components.

To ensure that your PCB design will work, and all components or traces are spaced out correctly, you must attach the correct footprint to each element placed in your schematic. The footprints are used to lay out the circuit board since it can contain information about the shape and mechanical dimensions, while also helping determine spacing between components. Most components and IC's that you would use in a PCB have footprints that can be accessible through electronic search engine websites like "SnapEDA" or "Ultra Librarian". Through these websites you can look up components and download their footprints, to then assign them on your software of choice when designing your schematic.

Schematics for the hardware design were constructed through the KiCad software. KiCad is a great tool for designing electronics circuits, to then convert to PCB designs. With the use of KiCad, we can insert symbols and footprints for each component to design the schematic for our circuits. As mentioned before, SnapEDA and Ultra Librarian were both used to ensure that the proper footprints were downloaded and inserted for each component. Through KiCad, we can also create 3D views of PCB's, some of which are shown in the following section. This is a helpful feature to give a general idea of how much space is available and how components are physically connected.

The basic procedure or workflow in KiCad is the following:

1. Create project & create a schematic

2. Assign footprints to symbols and generate the netlist

3. Create the PCB, importing the netlist

4. Use the "Design Rule Check" to test the board and ensure all connections are made

8.2 Camera Assembly

The camera assembly includes several custom fabricated mechanical components necessary for the smooth, precise, and robust movement of the camera view. Because a pan-and-tilt mount was procured from a surplus store, we already had in mind that the

equipment would need some enhancements for our purpose. After disassembling the mount, it quickly became apparent that major mechanical redesigning was necessary.

The pan gear train was the first suspect. The brass input gear that connected the pan motor shaft to the output gear that turned the mount in the horizontal axis had a large crack down the width of one tooth. The crack impacted the pitch diameter to the point where the gear wouldn't mesh with the adjacent gear after half a revolution. To rectify this, a new brass gear of the same pitch diameter, teeth count, and width was found. However, this gear lacked the purpose-built shaft that was integrated into the cracked gear. The solution was to fabricate a new shaft for the new gear to attach onto. Lacking the correct equipment (a lathe), a drill press with an X-Y table on it was substituted. The part was secured in the chuck of the drill press, and the X-Y table had a brazed-carbide tip lathe tool clamped to it. Movement in the Y-axis of the table pushed the tool into the spinning part, turning the diameter down where required to fit the new gear and the new ball bearing that all but removed the previous extra play in the radial axis of the old shaft. The part, an aluminum rod, was soft enough for this unconventional equipment combination to do the job. A flat section of the upper shoulder of the shaft was milled by putting an end mill in the chuck of the drill press and clamping the part to the X-Y table. This flat section was necessary for a hole to be drilled and tapped for a 4-40 set screw, allowing the stepper motor shaft to be coupled to the rest of the gear train in a semi-permanent fashion. Similar fabrication steps were utilized to make the primary tilt-axis shaft using a 12 mm dia. 4140 steel rod. Additionally, the worm wheel in the tilt axis had a flat section and set screw milled and drilled into it to allow it to be better secured on the tilt axis shaft.

Figure 8.1: Drill Press as a Lathe (Don't Recommend Doing This)

Figure 8.2: Finished Part with Set Screw for Stepper Motor Drive Shaft

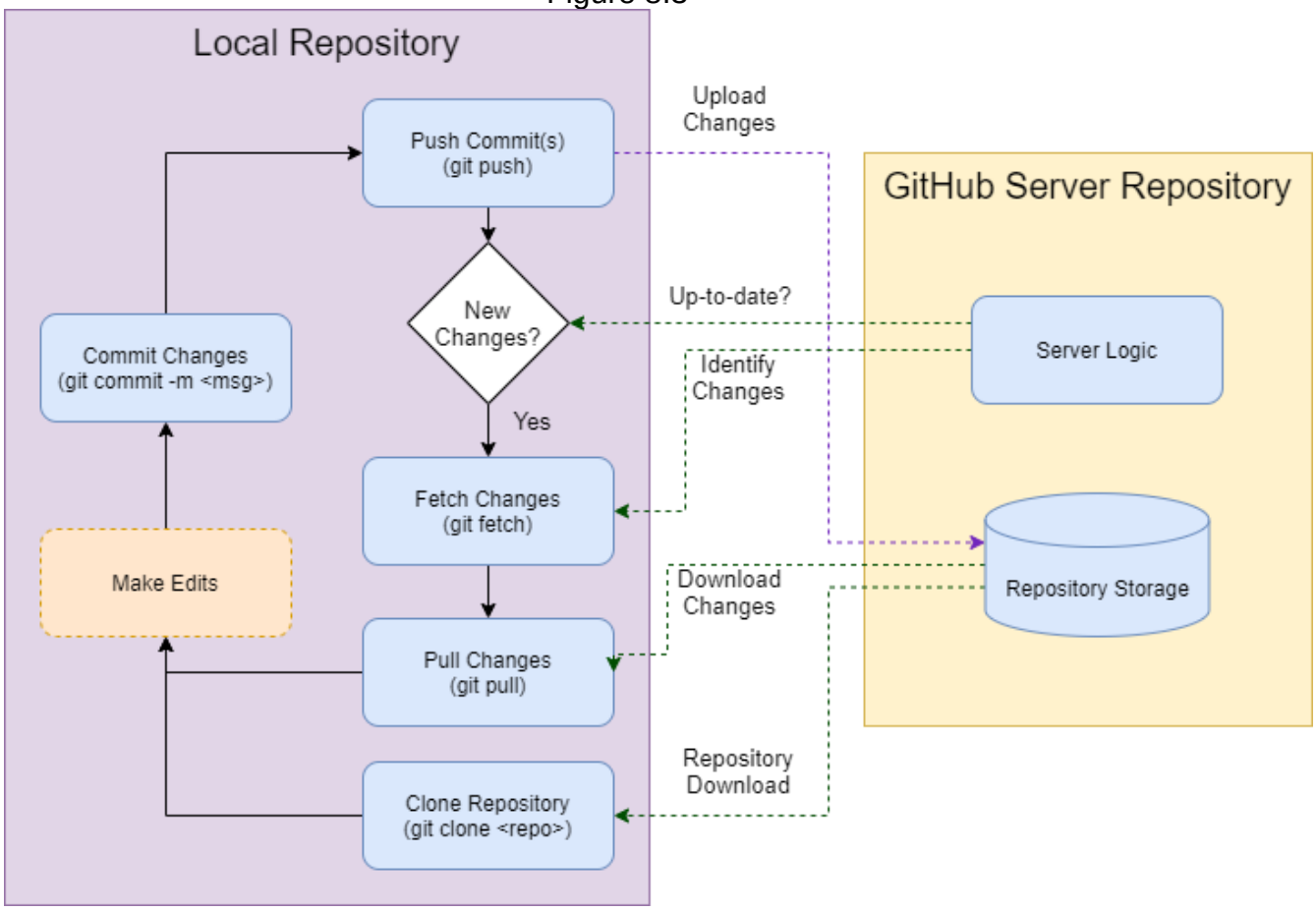
8.3 Source Control

Managing the source code for a project is an often looked over task in the beginning that proves more and more useful as the project is developed and more developers/devices are added. Without proper source control, a project is vulnerable to data loss/corruption and can become a real nightmare to manage multiple developers working simultaneously. The source control software used for this project was GitHub. This free-to-use platform provides servers for storing files and a control flow for updating files on the server and downloading the most recent changes. The client-side is a little more generic. A software called Git is run on the client machine that contains all of the logic for interacting with any server that implements it. GitHub is one of many platforms that Git can work with, but it is by far the most popular, developed, and advanced of the group. Outside of a few cases that require specialized servers (usually for dealing with very large files), GitHub is the platform of choice.

Git's outermost layer of storage for a project is the repository. On the server, the repository is the root directory of all project files and usually corresponds one-to-one with the project (one project has one repository). Anyone who wants to work on the project must download the repository by cloning it. This creates a copy of the repository on the user's machine that Git will monitor for changes. Once a user has finished edits in the repository, the edits can be committed. This forces Git to take note of the files that were added, deleted, or modified so that it can properly update the server. Usually, a commit is tied to a message created by the user that briefly summarizes the edits that were made. After committing, the edits can be uploaded to the server by pushing them. A push can only be made if there are no changes on the server that the clients edits did not account for (i.e. if someone else pushed to the server first, those changes must be downloaded before a push can happen).

To download others' changes on the server, they must first be fetched. This has Git ask the server what commits have been pushed since it last checked (last fetch/clone). After the server responds, the changes can then be pulled into the local repository, which downloads and integrates the changes with the current repository. This pull can only happen if the user has committed all edits. There are many other features available from Git and GitHub including branches and forks, which allow the project to be split up on the server and make incorporating multiple developers much more feasible. This project does not take advantage of these features, since most of the development work is done on the same machine. The flowchart below illustrates the source control process used.

Figure 8.3



Flow of data/files between user's local repository and GitHub's server repository. Blue boxes indicate Git logic and commands. Orange boxes represent user modifications to the local repository.

8.4 Software-Hardware Integration

After a long period of software and hardware design/development, integrating the two pieces together presents a daunting challenge for any project. A lot of ideas that were thought to work and be effective can prove faulty or ineffective when combined with the other piece. Engineers learn through experience what things to look for when entering this stage of the project's growth, and can leverage their experience to know how to design and develop for an easy integration period. If developed perfectly, the integration stage could be as easy as fitting two puzzle pieces together. But nothing is perfect, especially in the world of engineering. This section will outline some of the problems that came up during the integration phase of the project and how they were addressed/fixed.

8.4.1 Motor Logic Module

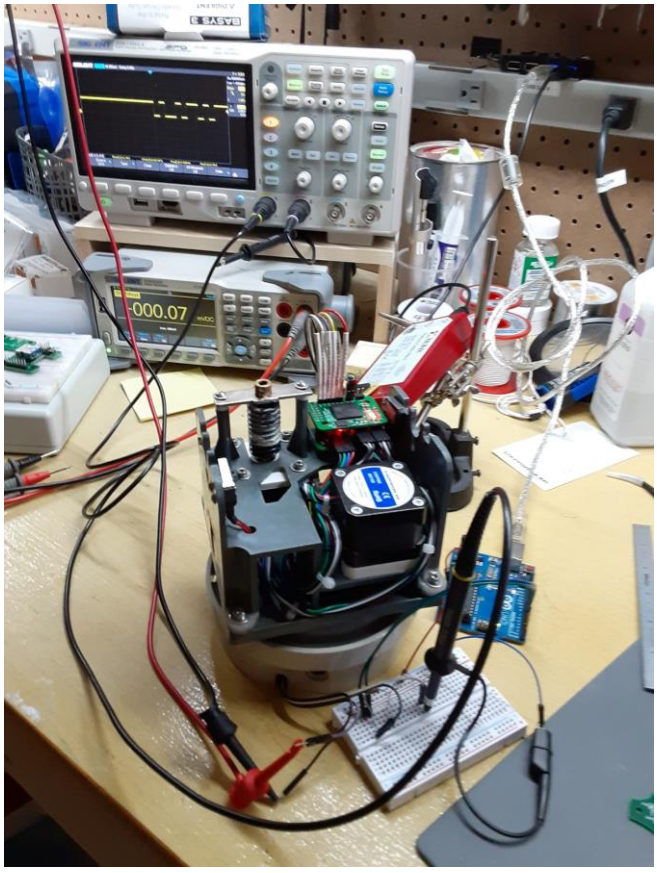


Figure 8.4: HDL Development for Motor Controller Board, Hardware-Software Integration

The motor logic module utilizes an internal hardware design that maximizes responsiveness to movement commands. To integrate the Verilog code, a benchtop oscilloscope, power supply, and USB-to-UART converter (Arduino Uno & MSP430 Launchpad) were used for stand-ins for the Odroid SBC. This configuration enabled all commands to be tested on the system without the unnecessary subsystems needing to be attached. A Xilinx Platform Cable USB II programmer/debugger was used for JTAG

communication with the FPGA, allowing uploading of the hardware configuration to the onboard SPI flash device.

8.4.2 Parallel Processing

Systems that react to external events must be capable of processing multiple things simultaneously. If only one thing can be processed at a time, the system will miss events that it should've reacted to, but didn't because it wasn't listening for them. Our system must be capable of reacting to events in real time and executing other tasks simultaneously in order to achieve low response times. Each isolated thing that our system must process happens on a different thread on the CPU. While each thread has its own program counter (meaning they can reference different instructions), they share the same memory, giving them read and write access to the same data and variables. A problem arises as a result of the sharing of memory that is present in most multi-threaded systems. Sometimes a piece of memory should only be accessed by one thread at a time, or it will not behave as expected. A working multi-threaded system recognizes this problem and fixes it where necessary.

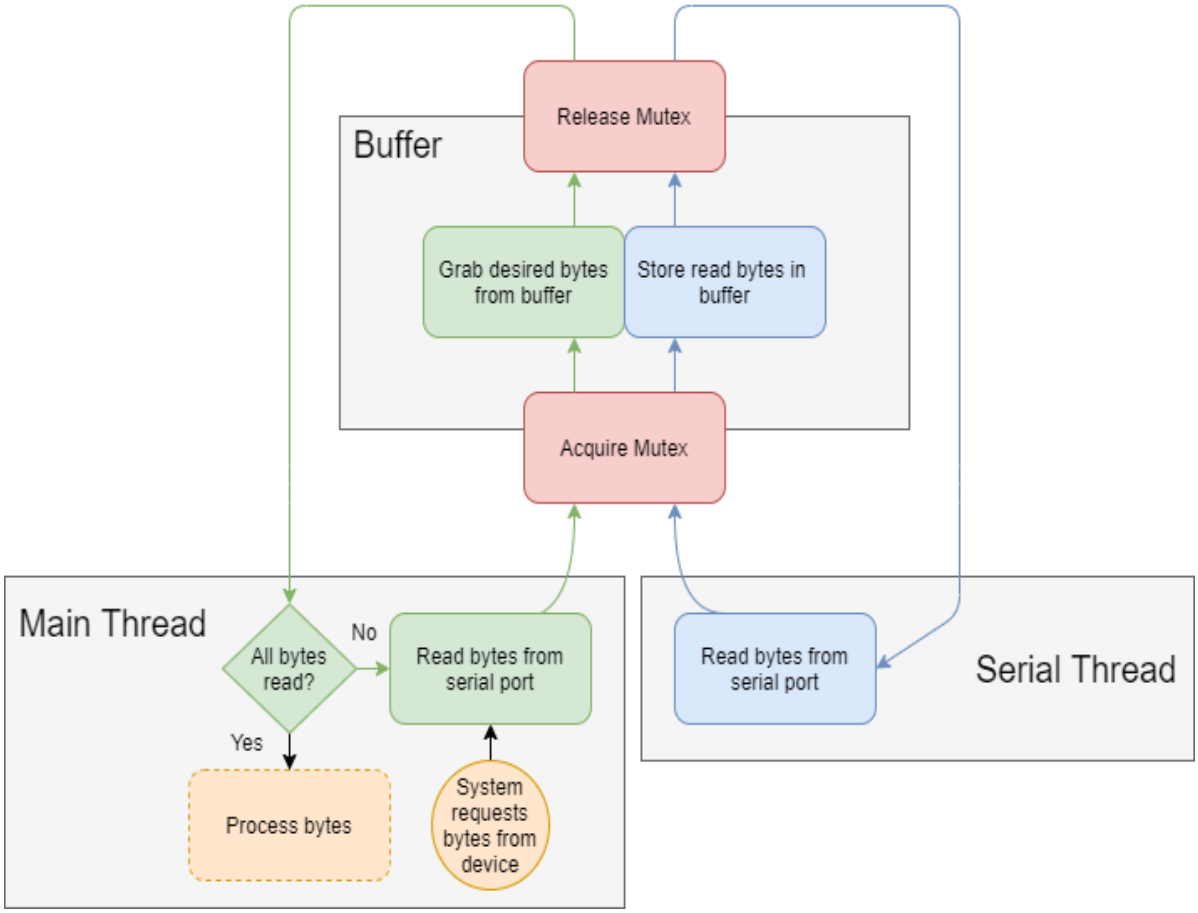
The biggest split in processing for our system involves the camera and the handheld device. The ODROID must constantly be listening to commands from the microphone device while simultaneously grabbing and processing frames from the camera. Each of these tasks requires the full attention of the CPU, and therefore cannot be achieved without multi-threading. At the start of the program, the camera kicks off a thread that retrieves frames from the camera and alerts other objects including the recording device and camera motion controller that a frame is grabbed. The frames are then processed on the thread that the camera started (not the thread that the program started on). While that thread is going, the initial thread that the program started on is waiting for commands from the handheld device to act on.

Consider the situation when the system is listening for bytes coming from one device but bytes from another device are sent. In this situation, two problems occur. The first is that the expected message will be misinterpreted, and the second is that the message that was intended by the first message won't be interpreted at all (effectively losing the message). Having an address tied to the message helps with this problem, but it would force the system to ignore the unwanted message in anticipation for the other device's message. To solve the problem, the way in which the system read from the serial port was restructured. A thread was dedicated to grabbing bytes from the actual serial buffer and storing them in another internal buffer, labeling them according to the device that they came from. When the main thread wanted to read a number of bytes from a specific device on the serial port, it would instead poll the internal buffer until that many bytes were placed in the internal buffer. Those bytes would be removed from the buffer to prevent them from crowding the memory. This allowed bytes from each device to come in simultaneously and still be correctly interpreted.

When multiple threads modify the same section of memory simultaneously, they run the risk of corrupting the memory. This usually happens when the modification requires a few

instructions to execute before the list can be accessed/modified again. The processor is capable, however, of scheduling one thread to perform some of the modification instructions and then another thread to access/modify the same memory. To prevent this, a mutex can be used. A mutex acts like a lock that is given to any thread that asks for it, so long as no other thread has it. While a thread has a lock, all other threads will be forced to wait for the lock to be released before they can continue execution. It is important that the lock be released as soon as possible to prevent the other threads from idling for too long or indefinitely. A mutex was used to prevent the corruption of the internal buffer of read bytes from the serial port. Every time the thread that was reading from the actual serial port was able to read bytes, it would have to obtain the mutex in order to store the bytes in the buffer. Once the bytes were written, the mutex was released. On the other end, a thread that wanted to read and possibly remove from the internal buffer had to acquire that same mutex, preventing concurrent access. The flowchart below provides a summary of how the threads would interact with the mutex and internal buffer.

Figure 8.5



Process flow of multi-threaded serial port operations. Blue boxes represent logic that happens on the main serial thread. Green boxes represent logic run by other threads to acquire bytes from the serial port. Red boxes indicate code run by both threads to ensure thread-safety.

Another important use of parallel processing in the system was with recording frames. With a fast enough processor, especially if a graphics processor is available, frames could be recorded synchronously everytime the camera adds a new one. Unfortunately, the Odroid does not have a built-in graphics card and the CPU speed, while good for an embedded device, is not fast enough to keep up with the frame rate of the camera. On top of grabbing frames, the same thread is responsible for sending move commands to the motors. Even though a command is not sent every frame, the few frames that do send commands can daa a delay in the upwards of 10s or even 100s of milliseconds. All of this amounts to a hefty amount of processing that needs to be done in between each frame. While frames are being recorded, a few frames in the camera are skipped, causing the resulting video to seem like it was sped up. The task of taking each frame and saving it to the micro-SD card was delegated to another thread. The Odroid's chip comes equipped with two 4-core processors allowing a frame to be grabbed while the previous one is still being written to memory.

To keep track of the frames that still needed to be recorded, a queue was implemented that was shared between the thread that grabbed frames and the thread that recorded them. Whenever a new frame arrived, it would be added to the queue to be recorded. The recording thread would be constantly polling the queue for available frames, recording any that were found. A mutex was also implemented similarly to the serial mutex to prevent the queue from getting corrupted when a frame was inserted concurrently with a frame being removed. Despite allowing the grabbing and recording to be done simultaneously, having a thread dedicated to recording the frames can be a risky option. If the camera can produce frames faster than the recording threads can record them, the recording buffer will slowly fill up, causing an increasing amount of memory to be consumed. If this buildup happens for a while, the device can run out of memory, and the entire program will crash. This was monitored carefully across the development process to ensure that the program was fast enough to keep up with the camera.

9. System Testing

9.1 Hardware

9.1.1 Bench Testing

Bench testing of the hardware was achieved using standard electronics laboratory equipment and JTAG debugger/programmers. The following is a list of the specific equipment models used:

- Siglent SDS 1104X-E 4-Channel 100MHz Digital Oscilloscope
- Siglent SDM3045X Digital Multimeter
- GW Instek GPC-3020 32V/2A 3-Channel DC Power Supply
- BK Precision 9104 84V/10A Single Channel DC Power Supply
- Texas Instruments MSP-FET320UIF JTAG Debugger/Programmer
- Xilinx USB Platform Cable II JTAG Debugger/Programmer
- Lattice Hardware USB 2 JTAG Debugger/Programmer

This list includes equipment for powering the various devices-under-test (DUT), measuring DC voltages and currents, probing signal lines, and programming embedded MCUs and FPGAs/flash configuration memories.

Bench power supplies are vital tools for powering systems on a test bench without needing in-situ power source for the system. In our project, the in-situ power source is a lead-acid 12V car battery, which makes for an inconvenient and hazardous power supply in an indoor electronics laboratory. The power supplies used in our testing also contain overload protection and overcurrent monitoring using a user-defined current limit. During testing, these features were essential insurance in the event of a short circuit or similar massive spike in current consumption. Without current limiting, the hardware could be severely damaged or destroyed.

Similarly, to catch any defects in manufacturing and assembly of the boards, all PCBs designed and assembled by us underwent an initial safe-to-turn-on test. This encompasses a number of final checks and measurements made on each board to ensure that the signal and power nets have maintained the isolation required for adjacent traces and that components are installed in proper orientation. The most important measurement made in a safe-to-turn-on test is the resistance measurements made between terminals of power rails using a digital multimeter. This includes the primary supply rail and ground net - a low resistance measurement made between these two connections will indicate a short in the board somewhere. Typically, a short found at this stage can be traced to solder bridges between adjacent pins in a large-pin-count IC. If left unresolved, a current well over the safe current capacity of the small traces and shorted pins will flow through the PCB, overheating and damaging the board in the process. Other checks related to shorted pins include visual inspection of soldered chips under a microscope and additional cleaning using alcohol to remove excess flux and un-reflowed solder paste.

9.1.2 Embedded Microphone Test

To test the UART communication between the MSP430FR6989 and the speech recognition module, a test involving the MSP430's LED's was implemented. The idea was that if the MSP430 device can recognize an individual speech command, then an LED will light up. For this test we used the word "Red" and "Green" to turn the red and green LED on correspondingly. By using Code Composer Studio, a test code was written that would send the speech recognition device the "Head+Key" (0xAA21) which would allow the speech recognition device to begin its listening procedure. A character array was implemented that would store each individual character in each index to read the result. Table 9.1 shows each word and their expected result. As mentioned previously, only two LEDs are used for this test and the words Red and Green will only be repeated. This means that "Result:12, Result:14, and Result:15" should never appear in any of our results when running this test.

Table 9.1: Test keywords & results

Keyword	Expected Result
Red	Result:11
Blue	Result:12
Green	Result:13
Purple	Result:14
Black	Result:15

As shown below in Figure 9.1 and Figure 9.2, the character array has 10 elements. The keywords "Red" and "Green" were repeated and read by the speech recognition device. The MSP then reads the character array, that stores the result values, sent from the speech recognition device. The most important element in the array is data[8]. When any keyword is mentioned or repeated, the speech recognition module will always send "Result:1x" to the MSP device. Therefore, we can check the character stored in the 8th index of the array to see which word has been repeated. Figure 9.3 on the left shows the red LED is turned on when the keyword "Red" is repeated.On the right, the green LED is turned on when the keyword "Green" is repeated. After a keyword has been recognized, a hex string "0XAA00" will be sent to the speech recognition module to enter into a waiting stage. During this waiting stage, the module will not be listening for keywords, and must be sent the "0XAA21" string to begin listening for keywords again.

data	unsigned char[10]	0x0023F2
[0]	unsigned char	R
[1]	unsigned char	e
[2]	unsigned char	s
[3]	unsigned char	u
[4]	unsigned char	l
[5]	unsigned char	t
[6]	unsigned char	:
[7]	unsigned char	1
[8]	unsigned char	1
[9]	unsigned char	.

Figure 9.1: Array of characters to read “Result:11”

data	unsigned char[10]	0x0023F2
[0]	unsigned char	R
[1]	unsigned char	e
[2]	unsigned char	s
[3]	unsigned char	u
[4]	unsigned char	l
[5]	unsigned char	t
[6]	unsigned char	:
[7]	unsigned char	1
[8]	unsigned char	3
[9]	unsigned char	.

Figure 9.2: Array of characters to read “Result:13”

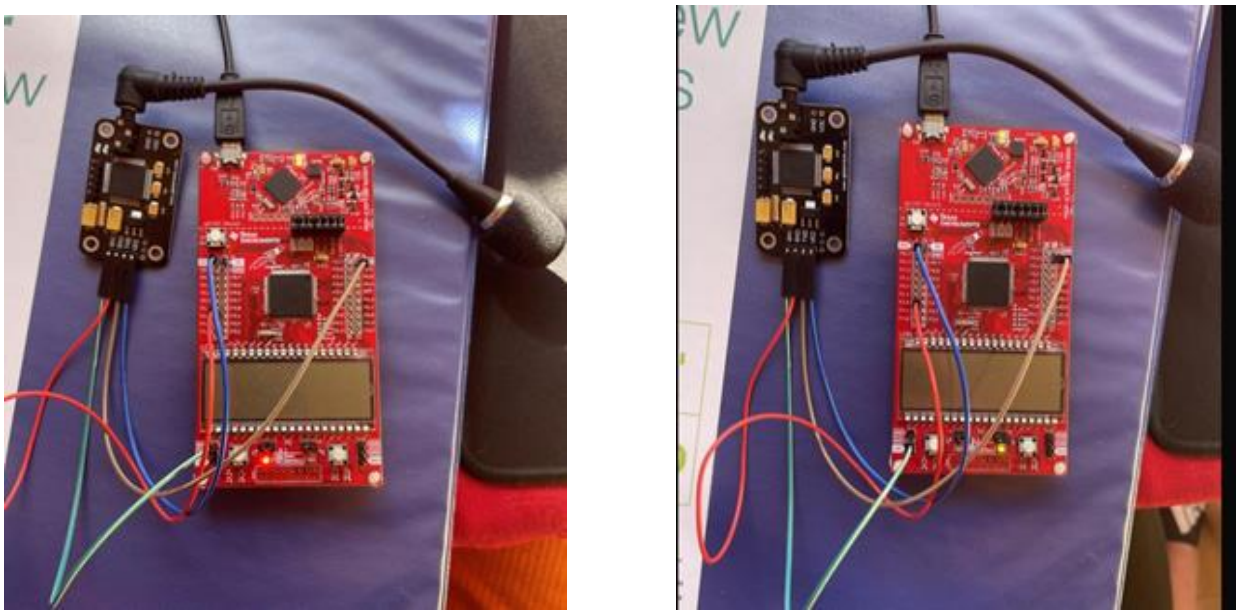


Figure 9.3: Red and Green LED test for keywords

9.2 Software

9.2.1 Functional Testing

Before integrating the software into the whole system, its core components were tested rigorously for bugs and logical errors. When creating an entire communication system spread across multiple devices, there is plenty of room for mistakes and format issues, which can be very hard to diagnose after the system is put together. There were two main tests that software went through before it was deemed ready to be integrated with the rest of the system for further testing. The first point to be reached was making sure the full communication cycle between the devices happened flawlessly and the officer tracking + recording worked as expected. Following that, the software would be ported over to the Odroid, where the processor was put to the test to see if it was capable of keeping up with the incoming camera frames and the response time was low enough. The tests required that no supplemental code be written to allow the software to pass the test, although extra code could be used to help reduce the amount of code that was being tested at once.

A huge tool throughout the development process that was vital to this stage in the testing process was a debugger. Debuggers allow developers to manual execute their code instead of it running at the CPU's GHz clock speed. There are many different debuggers out there, and most come with the editor used to write the source code to begin with. In fact, editors or IDEs that include a built-in debugger are far more popular and are used more widely by the industry, since it lowers development time. For developing this project, Visual Studio Code (VS Code) was used to both write the source and debug it afterwards. This application supports a multitude of languages and has extensions for intellisense in each of these languages. Intellisense suggests code to fill in for you in real time, so the developer does not have to memorize the exact spelling of each class or function. The main reason to use VS Code was its elaborate debugger. It allows the code to be run one line at a time, and variables could be inspected to see their current values. Not only did this made finding and fixing bugs a whole lot easier,but it helped ensure that the code was working as expected throughout the entire process.

9.2.1.1 Communications Testing

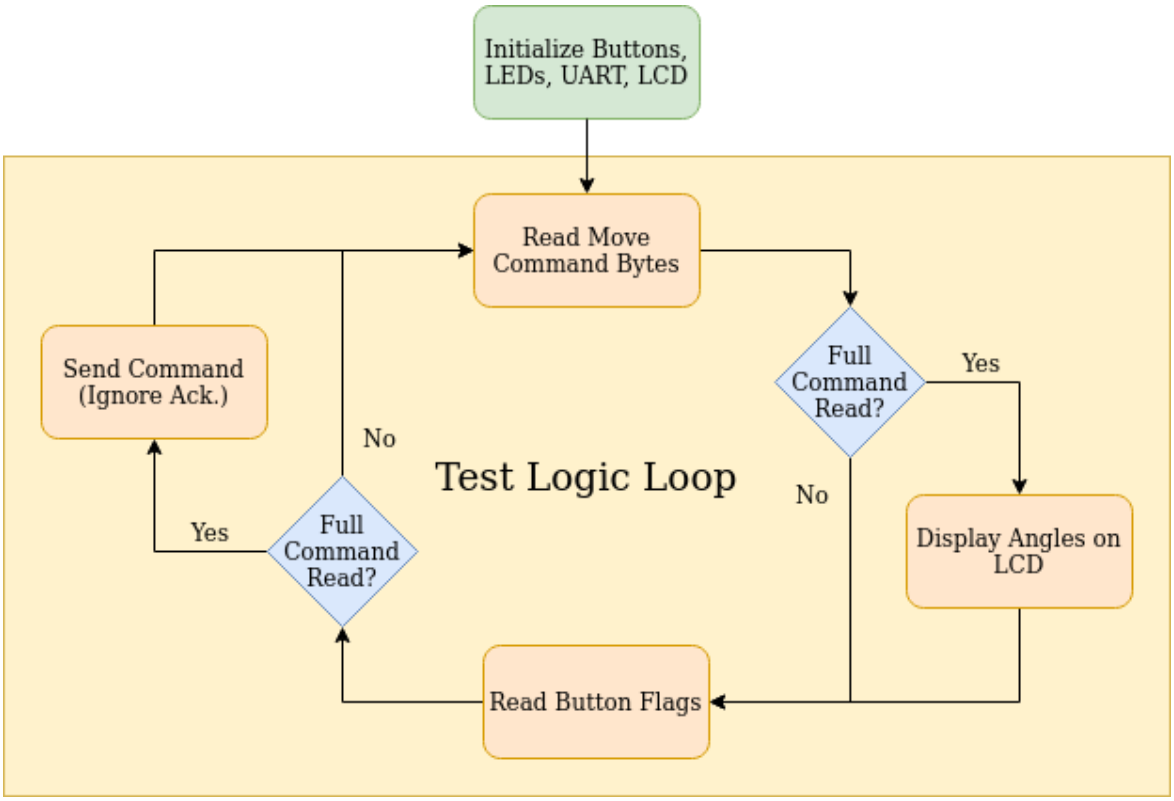
Without access to the device adapter, testing the communication and functionality of the system would require a seperate program to be written. The main system would have to receive commands from a separate device over serial and send out commands to that same device, and so an external microcontroller was used. A spare MSP430FR6989 was programmed to simultaneously act as the device adapter, handheld device, and motor controller. The goal of the program was to send commands to the software and respond to received commands in the exact same manner that the real devices would. Once achieved, the testing MSP could be replaced by the real hardware and the system would not be able to tell the difference. The slight system architecture differences between the test environment (test MSP only) and production environment (device adapter + handheld + motor controller) would add a layer of challenges and limitations for the test program to face.

The biggest obstacle to overcome was that the MSP had to be able to read bytes intended for two different processes and correctly handle them. The solution to this problem in the main system was to use multiple threads and have one dedicated to reading the bytes from the serial port and labeling the bytes by device. The MSP, however, came with a constraint that would make that impossible to achieve in the test program. The MSP's chip is simple and does not natively support multithreading. A technology in the form of RTOS (real-time operating system) is available which allows for the scheduling of tasks on the MSP chips, but the setup and development overhead for RTOS was too large to be worth implementing for the test environment. Instead, a few realizations were made that would allow small aspects of the test system to be cut out in exchange for no longer needing to worry about the multiple devices issue. The situation that brings the most trouble to the test program is when it is trying to read acknowledgements sent from the main software while waiting for a move command. The only real purpose of the acknowledgements is to determine if the system has lost connection with one of the devices, which should never happen in the test environment, and can therefore be ignored. Instead, testing of the acknowledgments can be done after the software has been integrated with the rest of the system. Part of the main system guarantees that only one message will be written to the serial at a time, and so if the header byte for a move command is received, the following bytes can be safely read as the move arguments. Without the need to read acknowledgement bytes, the test program can just disregard these bytes, and only look at bytes that correspond to a move command.

A few of the MSP's peripherals were used to mimic the user input from the handheld and the system output to the motors. There are conveniently three push buttons on the board that can each represent one of the main commands that the handheld can send (Start/Stop Officer Tracking and Send Keyword). The board's LCD screen can be used to output the motor rotation values sent by the software. To mimic the I/O of the system, a command will be sent each time its corresponding button is pressed, and the most recent horizontal and vertical angles to rotate the motors will always be displayed on the LCD screen.

After initializing the buttons, UART channel, and LCD screen, the program enters a continuous loop to handle all the command processing logic. To maximize responsiveness of the buttons, the program should never hang too long at any point, so that the button press can be registered. First, the program attempts to read a move command. If all the bytes for the move command are read, then the program will display the two angles, otherwise it will continue on to the next part so that the buttons can be read faster. To display the two values, the LCD screen has six 8-segment display nodes. This gives each angle three characters to work with. The possible range of values that the angle could take are [-180, 180], and so at most four characters would be needed. In these cases, the least significant digit is chopped off, although this case is not expected to occur, since most of the movement commands are small. After displaying the move command (or skipping the display to wait for all the bytes to be read), the program checks the button flags to see if either button was pressed and sends the corresponding command to the main software. The figure below summarizes the FLOW of the test program.

Figure 9.4



Flowchart of test program logic and I/O.

9.2.2 Scenario Testing

Running the software with the rest of the system is a misleadingly long and difficult process that ought to be prepared for extensively before integration happens. Unexperienced engineers and developers tend to believe their part works perfectly and will easily mold into the other parts of the system without fail, only to find out that the other parts of the project expose bugs and flaws that the debugger can't really test for. It is especially tricky because many times there is uncertainty on which part or parts caused the error, which can have the developers and engineers searching for hours to find a bug/problem that doesn't exist. To help pinpoint exactly where the problem is, an extra layer of code can be written to provide more information at runtime on what exactly the code is doing and/or waiting for. This comes in handy when dealing with a system that involves multiple device communication, since most issues that occur in these systems involve two devices deadlocking, or waiting indefinitely for each other to send something. Two measures were taken to save time and effort in integrating the software with the rest of the system.

9.2.2.1 Logging System

The best and most common way of providing information at runtime is through extensive use of logging. Logging involves outputting information to the console window and/or a file when certain events occur in the code. Using a console is good because it provides immediate feedback that can be more easily read in real time. Files are more useful when the system is being deployed somewhere and the problem has to be diagnosed after the fact, so the console output is unavailable. Information that gets logged could be as simple as a process has started or as complex as showing the state of certain objects when the code reaches certain key points. Knowing what to log and when to log it is important because too little logging can lead to not having enough information to diagnose a problem and too much logging can lead to not being able to put together the right pieces of information to diagnose and/or solve the problem. Plus, if exceedingly large amounts of data are logged, the log files can get very large and unopenable without special software, potentially reaching gigabytes in size. This project will mostly utilize console outputs, since it is easier to access and will make testing a lot easier.

It is almost impossible to provide a lot of detail on each aspect of the program while only logging key points in the process. Our system uses a selective logging system to combat this dilemma and provide both in-depth log information and keep the log size at a minimum. Selective logging allows users to turn on and off logging for certain areas of the code. If a problem is suspected in the camera portion of the program, the camera logging can be turned on while the others are turned off, so that the logging focuses around the problem. A few options provide higher level logging that doesn't go in depth in any area, but covers the entire program. These options work well initially to help narrow down the general area that a problem can occur. It is important to note that the multi-threaded aspect of the program can rearrange the order that log messages appear in the console. The process scheduler inside the operating system chooses which thread gets to run on each cycle, and it does not guarantee any pattern or order for choosing the next thread to run.

To implement the selective logging, a custom log function was created that looks at a bitmask to determine when to actually log information. A bitmask is an unsigned integer whose value is not as important as the individual bits that are set to 1. They are most prominently used in areas that have a lot of different settings that can be turned on and off and need to be checked in bulk. To check if a particular flag is set to true, the bit mask can be ANDed with the flag in question, a non-zero result meaning the flag is true. At the start of the program, the requested log options can be set up by ORing all of the desired options and setting the overall log flag to that result. Below is a list and short description of each log option available.

Information: General information on the overall process.

Debug: Same as "Information", but also includes some state output of the program.

Error: Any errors that occur during the program and any exceptions that get thrown. These are logged if any of the log options are selected.

Frames: Camera frame acquisition thread.

Officers: Summary of object detection results on each analyzed frame.

Movements: All movement commands sent to the motors.

Recording: Camera frame recording thread. These are also logged if the “Frames” option is selected.

Raw Serial: Any bytes that are read from or written to the serial port.

Raw Serial Continuous: Any bytes that are read from or written to the serial port as well as all attempts to read from the serial port.

Device Serial: When the system is waiting for certain messages from the devices and when the system sends messages to the devices. These are also logged if the “Raw Serial” option is selected.

Acknowledge: Notification of waiting on acknowledgements/responses from the devices. These are also logged if either of the “Device Serial” or “Raw Serial” options is selected.

Locking: Which threads attempt to lock a mutex and which mutex is being locked.

Flir: The Spinnaker SDK debug log information.

LED: Turning on/off the LED on the odroid as well as the headlight LEDs.

OpenCV: OpenCV operations involving image filtering.

9.2.2.2 Program Settings

Recompiling a program to make changes is a costly effort that should be avoided if at all possible. On the odroid, the program takes 1 - 2 minutes to compile, not including having to fetch the new source code from GitHub. Obviously, if an error is found in the code or the overall functionality of a certain module needs to change, then the code will have to be recompiled. It is smaller changes, however, that can be made without needing to recompile the program that will save a lot of time during the development, testing, and integration process of the system. Having configurable settings not only shrinks the number of recompilations, but also adds a layer of customization for after the program has been deployed. Our system has quite a few variables that can be tweaked to modify its performance and results. They are implemented using a JSON file containing all of the program’s settings that gets loaded at startup. To change the settings, one simply has to edit the JSON file and rerun the program, and all of the new settings will be applied. Any setting that is expected to change or has the potential to change should be included in a program’s settings. Settings that cannot change, or settings that can

severely harm the system if incorrectly changed should not be included, or at least carefully considered before including. Below is a list and description of each of the settings for the program.

Camera Serial Number: The serial number of the Flir Firefly DL that is used to connect to the camera in the program. While the serial number is not likely to ever change on the same system, different systems will have different cameras, each with a unique serial number.

Device Serial Configuration: The path to the device serial port and the baud rate. In Linux (the OS running on the Odroid), reading and writing to serial ports is emulated using files.

Motors Serial Configuration: The path to the motors serial port and the baud rate. Only used when the device adapter is disabled for testing.

Handheld Serial Configuration: The path to the handheld serial port and the baud rate. Only used when the device adapter is disabled for testing.

Use Device Adapter: Enables/Disables use of the device adapter. If disabled, a separate serial connection is made for the motors and handheld device.

Officer Class Id: The id of the bounding boxes that correspond to officers in the object detection model. This could change if the model used contains other classes of images.

Target Region Proportion: The percentage of the frame that the target region occupies. See section 7.3.1 for more information on the target region.

Safe Region Proportion: The percentage of the frame that the safe region occupies. See section 7.3.1 for more information on the safe region.

Camera Frames To Skip Moving: The number of camera frames to skip before performing frame analysis to determine possible movements.

Camera Frame Rate: The desired frame rate (FPS) of the camera.

Camera Frame Width: The desired width of the images produced by the camera.

Camera Frame Height: The desired height of the images produced by the camera.

Home Angles: The initial angles to move the motors to when the system starts up.

Angle X Bounds: The two angles to move between on the pan axis when searching for the officer.

Pan Configuration: The minimum and maximum angle supported on the pan axis as well as the corresponding step number. Step values were linearly interpolated from these values.

Tilt Configuration: The minimum and maximum angle supported on the pan axis as well as the corresponding step number. Step values were linearly interpolated from these values.

Image Processing Configuration: Enabled and disabled different parts of the system to aid in testing and debugging.

Frame Display Refresh Rate: Set the refresh rate for displaying the live feed display. This display was only used during testing and debugging.

Use Status LED: Enabled and disabled using the status LED on the Odroid. This was mainly used to know what the system was doing when it was not connected to a monitor.

Status LED File: The path to the file that controlled the status LED brightness. This file was used to flash the LED.

Officer Confidence Threshold: The minimum confidence that the bounding box had to have to be considered a police officer.

Camera Buffer Count: The number of frames that the camera could buffer on the Odroid at once.

Minimum Officer HSV: The minimum HSV color for the filter that was applied to the images to improve the officer detection.

Maximum Officer HSV: The maximum HSV color for the filter that was applied to the images to improve the officer detection.

Officer Threshold: The percentage of the bounding box that had to lie within the HSV threshold to be considered an officer.

Motor Speeds: The speeds of the motors.

Log Flags: All the available logging options. Each option can be individually turned on and off.

Having the ability to change the log options without recompiling the program saved a significant amount of time while testing the fully integrated system. Modifying the camera settings allowed the footage quality to be maximized while still being runnable on the Odroid. The regions could also be adjusted to optimize the system's response times while minimizing the shakiness of the footage.

10. Administrative Content

As engineers, it is our responsibility to ensure that the project will be efficient and low cost. Cost is a big priority in many engineering designs, as the tradeoff for a higher and better working product is higher cost. If your design is too cheap, you might run into an abundance of problems working with less effective hardware, and spend more time troubleshooting or designing. Staying focused and planned out is also another important responsibility to help us achieve required deadlines for our projects. Administrative planning is useful to outline and define priorities for our project. In this chapter, project budget will be discussed along with project milestones.

10.1 Project Budget

As previously mentioned, cost is a big factor in engineering designs. As COVID-19 affected our semester drastically, it was tough to find a sponsor to help pay for our project. Companies were hesitant to fund the usual amount of senior design projects for UCF students, which means that we had to decide to pay for the funds of the project out of pocket. As a group of three students, we will evenly split the cost between each of us. This can be crucial, since not having a fourth person in our group will mean the price each of us will pay will be higher. For this reason, it is important to achieve a design that will meet and exceed expectations for our project, but also be reasonable in cost. It is helpful to come up with an estimated project budget & expected price to understand the distribution of our budget throughout the project.

Below in Table 10.1, we can see the major components of our project along with their associated price. The total price will consider shipping/tax, and account for the possibility of parts having to be reordered in case of damaged equipment. The most expensive component included in our design is the FireFly DL camera. This was an important piece of the project, and we believed that the cost put into it would be worth it due to the effect it would have on the project. The FireFly camera producing 1080p images at up 60FPS over USB 3.0, and the cameras built in image processing made it a clear choice that the price was worth this upgrade in our camera section. Mechanical components are also another bulk of the price due to the pan and tilt mounting of the camera and the enclosures needed to shield our hardware and electronic devices. The PCB design price considers that we will probably have to reorder certain PCB’s, just in case small fixes or changes need to be made in Senior Design 2. A lot of groups make mistakes during the construction of their PCB’s, or in the soldering process and it is common to have to reorder a PCB. To implement our project, we have set a price goal that we hope to not exceed, which is $1150.00.

Table 10.1: Project budget

Component/Item	Expected # of units	Estimated Price/Unit	Estimated Price
Mechanical Components			$160.00
FireFly DL Camera	1	$320.00	$320.00
S Mount Lens	1	$12.00	$12.00
ODROID-XU4 (SBC)	1	$80.00	$80.00
MSP430FR6989	1	$8.00	$8.00
Lattice MachXO2-1200 FPGA	1	$10.00	$10.00
Xilinx Spartan 7 FPGA	1	$17.00	$17.00
PCB Design + Components		$120	$120
Speech Recognition Module	1	$35.00	$35.00
Zigbee Radio	2	$12.50	$25.00
Micro SD Card + Reader	1	$25.00	$25.00
LCD	1	$11.50	$11.50
Total Estimated Cost			$824.50

10.2 Project Milestones

Setting goals and milestones in a project is beneficial to everyone in the group. Having a guideline and schedule to follow will assist and allow us to complete certain tasks before the deadline is reached. In Senior Design I, we are tasked with working on the project documentation and completing the report with a minimum of 90 pages with project content. To ensure that the group is progressing throughout the semester, a 45 page, and 75 page draft are both due during the semester. In Senior Design II, we are tasked with building a functional prototype for our design. To complete these tasks, we must apply some of the knowledge gained throughout our courses in the Electrical/Computer Engineering department. We understand this project will not be easy, so we have decided to get ahead and begin the process of ordering components, along with testing them during Senior Design I. By doing this, we can expect to be well equipped and ready to take on the challenges and errors that we run into in Senior Design II.

Table 10.2: Project Goals for Documentation

Milestone	Planned Finish date	Required Due Date
45 Page Draft	10/30/20 - 11/06/20	11/13/20
75 Page Draft	11/13/20 - 11/20/20	11/27/20
Final Document	11/27/20 - 12/05/20	12/08/20

Table 10.3: Project Goals for Development

Milestone	Achieve by (SD1 & SD2)
Overall system design	Week 4 (9/14 - 9/18)
Microphone interaction with MCU	Week 5 (9/21 - 9/25)
Pan & Tilt mount design	Week 8(10/12 - 10/16)
Initial trained object detection model	Week 10 (10/26 - 10/30)
PCB Design	Week 12 (11/9 - 11/20)
Camera library integration	Week 13 (11/16 - 11/20)
Device communication implementation	Week 15 (11/30 - 12/4)
Full software integration	Week 23 (2/22 - 2/26)

11. Project Conclusion

The objective for Senior Design is to learn as much about different fields in Electrical and Computer Engineering, while coming up with an efficiently working and useful prototype. With incorporating image recognition and speech recognition, we were able to learn about technologies that are quickly taking over in high-tech devices. Incorporating FPGA's in the project was also another example of how we challenged ourselves to learn as much as possible about different fields we were not yet comfortable with.

The Traffic Stop Watchdog design is definitely an idea that can be deemed as useful for the real world. Not only would our design benefit the police force, but it can also benefit civilians who are interacting with police officers. With precise stepper motor movement allowing vertical and horizontal movement, the camera will act as a set of eyes always to be tracking the police officer during an encounter. This will provide useful footage that can be used as evidence in many cases involving a police officer and a civilian. With the use of the handheld microphone device, a police officer will be available to call for backup quickly and have other useful commands.

Throughout the Fall 2020 semester, extensive research was done to ensure that we can design and implement the best working prototype with the budget that we have set. All aspects of the project were mentioned in this report, with great detail on how we have designed our system. While COVID-19 has had a big toll on the 2020 school year, the best was done to ensure that we were meeting weekly online to discuss the project's progress. Thankfully, we have gotten far ahead on our software and hardware development and have been able to design and test many important components of the project. Many components & PCBs have already been ordered and received. This alleviated the troubles and struggles that came along in Senior Design 2, when all that is left is some minor tweaks and then integration. As more testing and integration occurred during Senior Design 2, we expected many obstacles and problems to occur. Due to our early progress in the project, more time was available to focus on these issues to ensure that all the requirements for the Traffic Stop Watchdog system were met and satisfied.

12. Appendix A - References

[3.1]: Texas Instruments, “MSP430FR698x(1), MSP430FR598x(1) Mixed-Signal Microcontrollers datasheet (Rev. D),” slas789c datasheet, June 2014 [Revised Aug. 2018].

[3.2]: Contributors, TechTarget. “What Is Voice Recognition (Speaker Recognition)? - Definition from WhatIs.com.” *SearchCustomerExperience*, TechTarget, 31 Jan. 2018, searchcustomerexperience.techtarget.com/definition/voice-recognition-speaker-recognition.

[3.3]: “How Does Speech Recognition Software Work?” *Explain That Stuff*, 3 June 2020, www.explainthatstuff.com/voicerecognition.html.

[3.4]: “Basics of UART Communication.” *Circuit Basics*, 11 Apr. 2017, www.circuitbasics.com/basics-uart-communication/.

[3.5]: https://www.ece.ucf.edu/wp-content/uploads/2019/09/EEL4742-Lab-Manual.pdf

[3.6]: “What Is the SPI Protocol and How to Debug SPI Communication?” *Total Phase Blog*, 14 July 2020, www.totalphase.com/blog/2020/07/what-is-spi-protocol-how-to-debug-spi-communication/.

[3.7]: “Basics of the I2C Communication Protocol.” *Circuit Basics*, 11 Apr. 2017, www.circuitbasics.com/basics-of-the-i2c-communication-protocol/.

[3.8]: Texas Instruments, “TPS61322 6.5-µA Quiescent current, 1.8-A switch current boost converter datasheet (Rev. D),” SLVSDY5D datasheet, Jan. 2018 [Revised Feb. 2019].

[4.1]: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf

[4.2]: IEEE Standards Association, “IEEE Standard for Local and metropolitan area networks, Sep.2018.
http://ecee.colorado.edu/~liue/teaching/comm_standards/2015S_zigbee/802.15.4-2011.pdf

13. Appendix B - Permission for Copyrighted Images

Total Phase Support <support@totalphase.com>
4:00 PM

To: Jordi Niebla

Good Day Jordi,

We have approval in place for you to use the diagram with following terms and conditions:

- There should be a reference to Total Phase Website (www.totalphase.com) in credits section OR any other relevant section in documentation.
- A copy of final draft should be sent to us. You can email it me directly at tjohar@totalphase.com.
- At the end of the Project please share the public URL to be able to access the projects final report.

Please let us know if you have nay additional concerns.

Best regards,
Tarun Johar
Manager For Support
Total Phase, Inc.
http://www.totalphase.com/

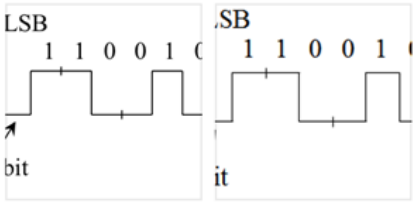
Follow Total Phase
Twitter: http://twitter.com/totalphase
Blog: http://blog.totalphase.com
YouTube: http://youtube.com/totalphase

Re: Permission to use EEL4742 Lab Manual images/content

Zakhia Abichar <Zakhia.Abichar@ucf.edu>
12:20 AM

To: Jordi Niebla

Save all attachments



Hi Jordi,

Sounds great. I attach the figure here.

Re: New Message From Circuit Basics

SC

Scott C <circuitbasics@gmail.com>
12:01 AM

To: Jordi Niebla

Hi Jordi,

Sure, go right ahead. You can use any of the images on Circuit Basics for educational purposes in a school or university setting. We just ask that you include a reference to the original web page that the image is taken from. Thanks for reading Circuit Basics and good luck with your project!

Scott
Circuit Basics
www.circuitbasics.com

JN

Jordi Niebla <Jniebla1@Knights.ucf.edu>
11/21/2020 4:21 PM

To: copyrightcounsel@list.ti.com

Good afternoon,
My name is Jordi Niebla and I am a University of Central Florida Engineering undergraduate student working on my Senior Design report with my group. We would like to request permission to use the following three images that come from your TI Workbench tool, and data sheets. We plan on using these components in our design for our project, and only plan on using each picture once on our report with credit given to Texas Instrument. Any response would be greatly appreciated, thank you.