University of Central Florida
Department of Electrical Engineering and Computer Science

3D Persistence of Vision Display

Group 8

Senior Design II Documentation

Aaron Burlison
Patrick Srofe
Antonio Ortiz
Timothy Egan
Summer 2012 - Fall 2012

Table of Contents

Table of Figures

Table of Tables

1 Executive Summary:

Persistence of vision is a phenomenon that has motivated engineers for years to create a variety of inventions. This has not changed even to this day. There are still devices using this visual trick being constructed with a wealth of internet examples available to show for it. These spinning devices that utilize LEDs to create the illusion of one solid image come in a variety of shapes and sizes from spheres and discs, to cylinders.

2 Project Description:

This chapter encompasses the motivations for why we chose one of these devices as our project. It also touches on the objectives or goals for this project, and the specifications for the device that we implemented.

2.1 Motivation:

The construction of these devices encompass a large spectrum of computer and electrical engineering knowledge from embedded systems and electronics, to digital systems processing and even electric machinery. Which our team felt allowed us to effectively test and display our grasp of knowledge.

In the case of our group project, when determining which of our group's ideas we wanted to tackle we found that the group had a split in interests. While some of the group wanted to create something that displayed a level of creativity other members wanted something within the scope of the group's skill sets. Finally, we all desired a project that was either inexpensive enough for the group to fund on their own or a project that was capable of acquiring sponsorship to fund it for us. After some deliberation we all agreed on the persistence of vision project as the best fit for all these concepts. The following sections help elaborate on why this project was such a good fit for our group.

2.1.1 Sponsorship:

As mentioned above our team was seeking a project that was inexpensive or capable of sponsorship. Since there are a variety of groups or organizations that rely on public advertisement and these displays require attention getting gimmicks our team felt that a persistence of vision device is a perfect fit. These devices have adequate levels of scalability, visual attractiveness, and portability that make it perfect for such a use.

A persistence of vision device is incredibly visually attractive with its various colorful and active displays. They are great at pulling people's attention and keeping it and in a scenario where a group is seeking to be both noticed and remembered it is quite a useful device. In the case that we were adequately funded we could make this device extremely attractive through high resolutions of LEDs and wide ranges of colors. This would also allow us to create simplistic to complex animations for the device that would draw people's attention.

These devices are also extremely scalable. We wanted to make the device easily programmed and accessible to both the experienced and inexperienced. This would allow someone experienced with programming to make a variety of their own custom displays and animations on the device. Someone inexperienced with programming would be capable of inputting various functions such as text inputs for banners. Both of these functions are excellent for sponsorship since they allow the user to easily set the device for any advertisement they desire.

Portability is obviously a concern for organizations that are advertising at booths or displays. These devices are extremely portable and our design is to not only make it portable but outlet friendly allowing you to plug it in to any standard outlet.

2.1.2 Skill Sets:

With a group made up of two students of electrical engineering and two students of computer engineering, we wanted a project that adequately displayed all of our skill sets. This project not only has a significant level of electrical design in both advanced and intermediate levels of electrical engineering but it has a significant level of both advanced and intermediate levels of programming and computer architecture requirements. This means that all four of the team members working on the project would find adequate amount of both familiarity and challenge within the project.

2.1.3 Creativity:

To be completely honest, if you are not interested in a project it is very difficult to work on it. This is a very true statement and most of our group members wanted a project that was entertaining enough to really keep their attention in addition to test their knowledge. This project seemed entertaining to our team. It is as simple as that. Not only would we be developing our skills as engineers but we would also get to flex our creativity by programming and designing a variety of interesting displays for this device. The final product would be bright, exciting, and interesting to see once it was complete; which our team was very excited to experience.

2.2 Objectives:

While in concept a persistence of vision device is good we needed to put to words specifically what our objectives for this device were. Since we had some ideas of what we wanted when choosing this project we also had a variety of features we wanted to add besides the basic features these devices generally come with. The following sections identify and describe these features in further detail.

2.2.1 Frame Rate:

Since human vision is tricked to perceive motion around the rate of twenty-five frames per second we needed a device to spin at a rate capable of recreating this illusion. Since twenty-five was the bare minimum we decided to overshoot to thirty frames per second, this would hopefully either make a more seamless image or account for any variations that may occur within the device.

2.2.2 Computer Interfacing:

We also desired the project to interface easily with your computer so that the user could upload customized code for their custom patterns, text, and animations. We wanted this to be done during operation also so a method of sending information from the stationary side to the rotating side was needed.

2.2.3 High Resolution:

We wanted to create a relatively high definition image so we designed the project with a high pixel count. We specifically chose 32x384 as our target resolution. This meant we needed a total of 32 LEDs. This also meant each LED had to be capable of a large scale of colors in order to recreate the image being sent each frame.

2.2.4 Portability:

Since we had decided the device needed to be portable it could be no heavier than a small television and only about as bulky. This meant the materials we chose to build this device out of need to be durable and light weight.

2.2.5 Programmability:

We wanted the device to be easily programmable and capable of at least simple marquee text displays that would be implemented with our own self developed program. This would allow the user to simply input a text banner or the time, and have it displayed on the device instead of just the computer interface. In addition, we also wanted the device to be complex enough that someone with experience in programming could also program to the processor and create their own

custom images and animations. This would allow for a lot of space for user development which seemed desirable to someone looking to advertise with the device.

2.3 Specifications:

The following is a list of specifications that we have come up with based on both our research, assumptions, and components that we have chosen during the development of this product.

- ✓ 32 RGB LEDs
- ✓ 16 Green LEDs
- ✓ 256 colors per LED
- ✓ 60 Hz refresh rate for LEDs
- ✓ 15-20 rps
- ✓ 61 cm diameter (cylinder)
- ✓ 80 cm height (cylinder)
- ✓ 12 - 15 lbs
- ✓ Operates on 120V AC and a 9V DC battery
- ✓ 2Mbits/s data transmission

3 Administrative Content:

Having a strong plan for administrating the budget and making due dates is essential for completing any project successfully. Our senior design project is by no means an exception. Our goal was to layout an administrative plan to govern and guide our project through the various stages and be the foundation that supported our work. Our administrative plan was laid out in three sections - budget, finance and schedule and milestones.

3.1 Budget:

Understanding the cost associated with any project helps separate what is feasible from what is unrealistic. As stated in our specifications section, the POV display required 32 LED's each with a 256 color range. This required us to procure 32 RGB capable LED's which came out to be about $1.51 each. With one additional array of 16 LED’s for prototyping this came out to a total of $72.48. The text display will require 16 green LEDs each at $0.27 for a total of $4.32. In addition to the LED's, we needed to procure some way to control the LED's. We used seven controllers total each costing $2.52 per controller with one extra for prototyping. This came out to be about $20.16.

The POV display must be capable of rotating at 25-30 RPS, though we only achieved 15-20 RPS. This required a sizable motor to insure we can operate at the required toque values. As well, we required a chassis or frame to support the LED array and on board controllers. The motor came out to be $35.00, while the Chassis was donated to us.

In order to processes the incoming signals and display the image on the LEDs we needed an onboard microcontroller. In addition, we needed a way to communicate with this microcontroller during operation so a wireless chip was also needed. The microcontroller came out to be $34.99, and the wireless chip for the controller came out to be $49.99.

In order to tie the on-board controller to the LED array we needed to procure PCB boards. We also needed to procure additional wire and cable to make all the miscellaneous connections required. The PCB boards came out to about $339.99 while the wires and cables came out to be around $89.98.

The final piece of the budget was the motor control and sensor circuit. The motor control chip turned out to be $106.00. The sensor circuit included 2 High-Output Infrared LEDs for $2.19 each. It also required 1 LM399 at $2.29. We purchased a large pack of varied resistors to handle any possible application we would need them for during the project at $9.99

The total cost associated with this project was about $1031.59. As seen in Table 3.1, a detailed summary of the budget and distributed cost can be complied.

Description	Qty	Cost (Each)	Price
RGB LEDs	48	$1.51	$72.48
Green LEDs	16	$0.27	$4.32
LED Controllers	8	$2.52	$20.16
Motor	1	$35.00	$35.00
On Board Controller	1	$34.99	$34.99
Wireless Chip	1	$49.99	$49.99
Chassis	1	-	Donated
IR LEDs	2	$2.19	$4.38
9V Battery	1	$4.99	$4.99

LM339	1	$2.29	$2.29
Copper Pipe [Slip Ring]	1	$6.99	$6.99
Misc. Equipment	-	-	$150.00
Prototyping	-	-	$200.00
PCB	4	-	$339.99
Motor Driver Chip	1	$106.00	$106.00
		Total:	$1031.59

Table 3.1 Project Budget

3.2 Finance:

The second portion of our administrative plan is to determine where the financial backing will come from to support the design and development of our POV display. Our goal was to be sponsored and we luckily did gain one sponsor during the course of our work. Kemco fabricated and donated to us our Chassis for the project. As for the rest of the costs associated with this project the cost was split among the group members.

3.3 Schedule and Milestones:

The final portion of our administrative plan was to develop and follow a schedule, which included major and minor milestones, to be a guide for keeping the production of the project on time and finished by the due date.

Major milestones will be defined as events or task that must be completed before the project can continue. A complete list of major milestones for the project is below.

- ✓ Senior Design I Documentation Due
- ✓ Prototyping Completed
- ✓ Project Design Finalized
- ✓ Project Fabrication Completed
- ✓ Testing
- ✓ Senior Design II Documentation and Final Project Due

Minor milestones will be defined as events or task that are less critical on an individual basis but must be completed before a Major milestone can be completed.

- ✓ Project Research
- ✓ Project Preliminary Design Review (Prior to Senior Design I Documentation Completed)
- ✓ Senior Design I Documentation Review
- ✓ Prototype Fabrication
- ✓ Prototype Testing
- ✓ Project Design Review from Prototype Results
- ✓ Fabrication of Chassis
- ✓ Fabrication of LED Array
- ✓ Project Assembly
- ✓ Preliminary Mechanical Operational Test
- ✓ Senior Design II Documentation Review

Finally, as seen in Table 3.3, a completed schedule was put together. The schedule contains all predefined major and minor milestones as well as completion by dates.

Milestone(Major/Minor)	**Start Date**	**Duration (Days)**	**Finish Date**
Project Research (Minor):	05/27/12	46	07/12/12
Project Design Review (Minor):	07/15/12	4	07/19/12
Senior Design 1 Doc. Draft Review (Minor):	07/25/12	4	07/29/12
Senior Design 1 Documentation Printing (Minor):	07/30/12	1	07/31/12
Senior Design 1 Documentation Final (Major):	05/27/12	67	08/02/12
Prototype Fabrication (Minor):	08/19/12	14	09/02/12
Prototype Testing (Minor):	09/02/12	7	09/09/12
Project Design Review from Proto Results (Minor):	09/09/12	7	09/16/12
Project Design Finalized (Major):	09/16/12	7	09/23/12
Procurement of Equipment (Minor):	09/23/12	56	11/18/12
Fabrication of Chassis (Minor):	10/29/12	18	11/16/12
Fabrication of LED Array (Minor):	11/16/12	14	11/30/12
Programming of Processors (Minor):	11/16/12	14	11/30/12
Assembly of POV Display (Minor):	11/18/12	12	11/30/12
Preliminary Mechanical Operational Test (Minor):	11/16/12	14	11/30/12
Project Fabrication Completed (Major):	09/23/12	68	11/30/12
Complete Functional and Operational Testing (Major):	11/30/12	7	12/07/12

Senior Design 1 Doc. Draft Review (Minor):	09/23/12	75	12/07/12
Senior Design II Doc. and Final Project Due (Major):	12/07/12	3	12/10/12

Table 3.3 Project Schedule

4 Research:

Designing is both a matter of applying the best known solution for a problem and creating new methods when the problem's solution isn't well known. In addition, many times a solution has multiple methods that fit well for solving a problem. In these cases we need to effectively narrow down the list and determine the solution our group feels will work best for us. In the case of our project there were eight key issues that we needed solutions to for our project that kept appearing in our discussions of this project.

The first problem was supplying power to this device. We needed to know whether we were going to use AC or DC power or some combination of both. Did we need to do some sort of AC to DC conversion? Which one was best for the purposes of our project? Section 4.1 discusses this topic and which one best suits our needs.

The second issue was signal processing. Our group new we wanted to allow for some way for this device to communicate with a computer. The question was which medium was best for our purposes? Since none of us had any experience in video processing this also meant we needed to figure out which format was best suited for our project. Would it be better to process an HDMI signal, VGA signal, or just do some form of file transfer through USB? Section 4.2 discusses this topic and compares each of these signals and the processing method needed to implement them for our project.

The third issue was LED implementation and control. Since we needed to blink these LEDs at a rapid speed we needed to know how this would affect the LED. What LED is best suited for this application? Will using pulse with modulation effect our display rate? How do we effectively control over four hundred LEDs? Section 4.3 will discuss these questions and determine the best fitting solution for each of these problems.

The forth issue was communications. Since this device has two sides to it, a stationary side and a rotating side, we need to determine how we are going to send the above signal across these kinetic state changes. Is there a wired solution for this problem? Would wireless be an effective solution to this problem? Are there issues with wireless when dealing with a rapidly rotating

receiver? Section 4.4 discusses these issues and compares each of these communication solutions.

The fifth issue is the motor itself. None of us had much experience with motors so we needed to research specifically which motor would work best for our purposes. Would a DC motor bet best or an AC motor? What is the most effective way of controlling the motor for our purpose? How can we minimize the noise commonly associated with motors? Section 4.5 discusses these topics and compares both motor types, and which method of controlling the device is best for our purposes.

The sixth issue is the actual structure of this project. This device is going to rotate at a very fast rpm value and that means it needs to be both very stable and balanced. What material is best suited for this project then? How do we balance it? What will be the torque requirements of this device? Section 4.6 discusses these questions and determines the best solution to each of them.

The seventh and final issue is our GUI. Since we want to develop a user interface for communicating with our device we need to know the best way of going about creating it. Would it be better to create it in C language or Java? What classes, functions, and variables will we need to implement the project? Section 4.7 will further discuss these concepts answering these questions and more.

4.1 Power Supply:

Just like any machine, the POV display required a source of power to operate. As discussed in the motivation, the POV display needed to be portable to require movement between events and shows. However, due to the size of the POV display and the power requirements of the motor, to operate the POV display from a battery supply would require a significantly large battery. A large battery deters from the portability of the POV display. As such, the power supply research was focused on utilizing power from an AC outlet.

4.1.1 AC Input:

As previously stated, the POV display would draw all of its power from a standard AC outlet. In the United States, the standard power for an outlet is 120 Vac at 60 Hz. In addition, the standard wiring practices for AC power in the United States for wiring of a 120V system is for the black wire to be the hot or line, the white wire to be the neutral and the green or bare cooper wire to be the ground.

4.1.1.1 Circuit Protection:

One additional design requirement for the AC input to the POV display that would require research is circuit protection. Since we would be accepting 120V AC from

a wall outlet which is most likely rated for 15 to 20 amps into the POV display, a good design criterion would be to protect the POV display from potential damage caused by surge in current. Over current can occur anytime there is a short circuit and since we will be most likely working with a metal chassis, adequate protection against short circuits should be taken.

Currently two commonly used forms of over current protection are available, fuses and circuit breakers. One disadvantage fuses have compared to circuit breakers is once fuses are used or blown, they must be replaced with a new fuse. In the case of circuit breakers, the breaker only needs to be reset and not completely replaced. However, the upfront cost of circuit breakers generally is greater than the initial cost of fuses. Two additional advantage fuses have over circuit breakers is their size tends to be smaller than circuit breakers and the flexibility to easily change a fuse to a higher or lower current rating without the need to re-wire any equipment. Therefore, we would focus our research on available fuse blocks or holders and fuses.

4.1.1.1.1 Fuse Blocks and Fuses for Circuit Protection:

Cooper Bussmann is a well-known and commonly used manufacturer of fuse blocks. The Bussmann Type BC and BCCM Series Class CC fuse blocks offer a compact but reliable solution for fused circuit requirements. The BC and BCCM series fuse blocks accept Class CC size fuses. As well, the fuse blocks are rated for operations at 600 Volts and up to 30 Amps. Since we would be protecting the incoming AC power, only the positive or line side of the AC power supply needs protection. This means we would only require a single pole fuse block. The part number for a single pole Bussmann type BC fuse block with screw connections is BC6031S. As well, Table 4.1.1.1.1 shows some of the available Type CC fuses offered by Bussmann and their corresponding current rating.

Part Number	Current Ratings
LP-CC-1	1 Amps
LP-CC-2	2 Amps
LP-CC-3	3 Amps
LP-CC-4	4 Amps
LP-CC-5	5 Amps
LP-CC-10	10 Amps
LP-CC-15	15 Amps
LP-CC-20	20 Amps

Table 4.1.1.1.1 Type LP-CC Fuses and Current Ratings

4.1.2 AC to DC Converter:

The POV display required conversion of the AC power coming from the wall outlet to DC in order to power the motor, the LED array and the microprocessors. A simple full wave rectifier circuit as seen in Figure 4.2, would be used.

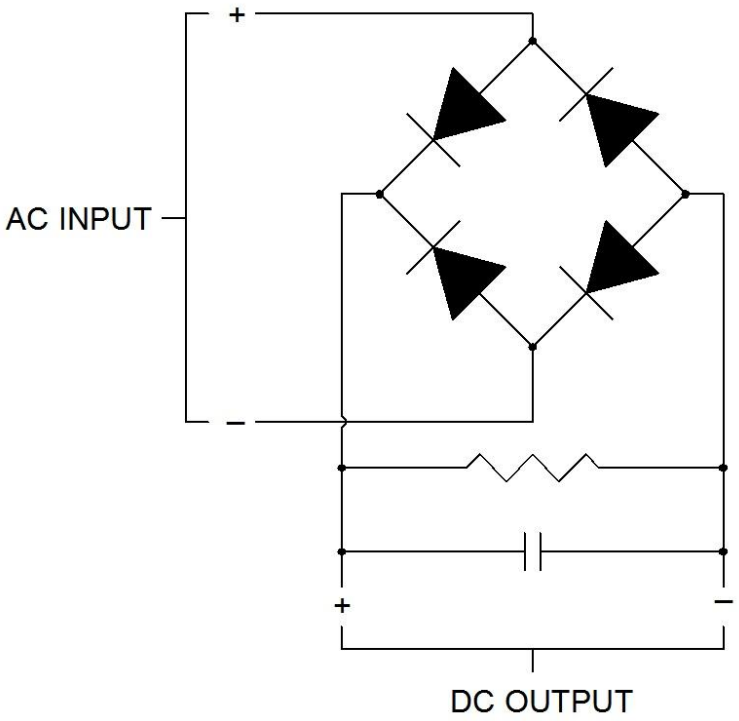


Figure 4.1.2 Full Wave Rectifier Circuit

Although the exact voltage required for the motor, LED array and microprocessors is not known at this time, we do know that we would most likely require the functionality to change the voltage output of the DC converter based on the requirements. In order to change the DC output voltage of the converter, we would vary the AC input by using a simple voltage divider circuit with a potentiometer or variable resister. Therefore we focused our research on determining what varieties of parts are available and their characteristics. In particular, we researched diodes, resisters, variable resisters and capacitors that have a maximum operating voltage of at least 150 volts and for the diodes, a power rating of at least 1500 to watts. The equation below, where Vr equals the ripple voltage and Vm equals the maximum voltage output, was used to determine the ripple voltage of the rectifier circuit and help to determine the correct combination of resistance and capacitance.

$$Vr = \frac{Vm}{2fRC}$$

4.1.2.1 Diodes:

As previously stated, the diodes required for the AC to DC converter was needed to operate at a maximum of 150 volts and 1500 watts. This design criterion was allowed for a maximum of 10 amps to flow through the diodes and provide adequate power to the motor and other circuits. One such diode is the MUR Series diode manufactured by Multicomp. The diode was designed with the purpose to be used in inverting and rectifying circuits. Part number MUR1560 has the maximum ratings of 420 Vrms and 15 A forward current. The diode comes in TO-220A case allowing for easy integration into bread boards or PCB boards. As well, the MUR1560 is readily available with over 3,000 available to ship at a cost of less than $1.00 each.

4.1.2.2 Resistors:

The voltage requirements of the converter do not necessarily directly apply to the resistor. The most important characteristic of the resistor will be the power rating. Although the power rating for the resistors is less critical than the diodes, we still required resistors with a power rating of at least 5 watts to allow for proper heat dissipation. Vishay, a well known resistor manufacture, provides a type RS resistor that is wirewound with axial leads that would work well with the bread boards and PCB boards. One example of a complete resistor part number is RS00510K00FE12, which is a resistor rated for 10 kohm, 5 watts and a tolerance of +/- 1 percent.

4.1.2.3 Potentiometers and Variable Resistors:

After during some initial research, it was discovered that potentiometers and variable resistors do not come readily available at the power ratings required for the converter. Therefore, we used fixed valued resistors similar to the type RS resistor previously discussed.

4.1.2.4 Capacitors:

One available capacitor that meets the required specifications is manufactured by Vishay. Vishay offers an aluminum electrolytic type 53D capacitor that can operate at 200 Volts. Although the tolerance is only +/- 10%, the capacitor is available at rated capacitance range of 15 uF to 220,000 uF. Just like the resistor, the capacitor has axial leads to allow for easy integration into bread boards and PCB boards. Once again, the exact capacitance requirements are not know at this time, but an example of a completed capacitor part number rated for 350 uF is 53D351F200JL6.

4.2 Video and Signal Processing:

We intended to receive a live video stream from a laptop and display this video on our LED array. There are two primary formats that computers output video data in, VGA and HDMI. The research into these two different formats was to be

used to determine which format would be most appropriate for our needs and what would be required to use that format. This section will look at various means of video data compression and alteration.

4.2.1 VGA

We considered using the VGA output available from a computer as the video source for our display. This section will focus on the VGA signal format and will describe how video data is transmitted via VGA.

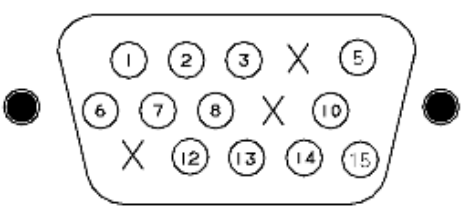
4.2.1.1 VGA Signal Standards:

In order for a computer to know what types of signal a display can handle, the computer communicates with the display through the Data Display Channel. The protocol used most commonly today is E-EDID, which has been defined by the organization VESA. With the E-EDID protocol, the computer reads a binary file in the display to determine what signal to send. It seems possible that we would have needed to write or edit our own E-EDID or file.

The EDID file is 128 bytes and contains basic information such as the vendor ID, serial number, manufacturing date of the display, and which EDID version is being used. It also contains a Video Input Definition, which specifies analog or digital. In the case of analog it contains several bits that specify which types of syncing the display supports, as there are several ways of doing this. A section of bits specify which of 16 predefined standard modes the display supports. Detailed timing information is contained within the last section. The second to last bit is a flag indicating whether or not there are any extensions to the file.

4.2.1.2 .Signal Sampling:

The video frames to be transmitted via VGA first start in a digital format on the PC and are converted to analog though the use of DAC's. Figure 4.2.1.2.a shows the pin configuration for the VGA DB15 connector and a summary of each pins function. The pins for Red, Green, and Blue (1 2 and 3) each carry a signal that ranges between 0V and 7V referenced from their respective ground pins (6 7 and 8).



1 - Red	6 - GND-R	11 - NC
2 - Green	7 - GND-G	12 - DDC Data
3 - Blue	8 - GND-B	13 - H-SYNC
4 - NC	9 - NC	14 - V-SYNC
5 - DDC Return	10 - GND-SYNC	15 - DDC CLK

Figure 4.2.1.2.a VGA DB15 connector and pin assignment

Figure 4.2.1.2.b shows how the red voltage value could be generated from 4 bits, allowing for 16 distinct voltages and therefore 16 colors of red. Combined with Blue and Green, this allows for the representation of 2^{12} different colors. There are many color modes, each with varying amounts of bits defining red, green and blue. The voltage range does not change, and when each RGB pin is read at the same time, a single pixel's color is defined.

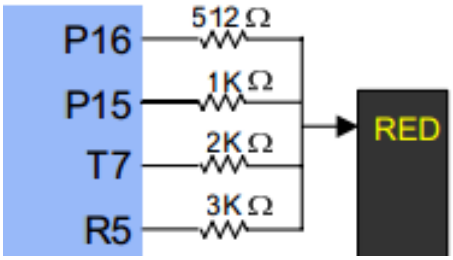


Figure 4.2.1.2.b Resistor circuit providing 16 colors from 4 inputs

The VGA signal transmits pixels one by one, starting in the top left of the frame, going from left to right, and then down. This process is timed using two synchronization pulses, HSYNC and VSYNC. The HSYNC pulse indicates the start and end of a row of pixels being transmitted, and the VSYNC indicates the start and end of a frame.

In addition to the VSYNC and HSYNC pulses, there are periods of time in which no pixel data is transmitted, which are known as the blinking and blanking intervals. As can be seen in Figure 4.2.1.2.c, these occur starting just before the VSYNC and HSYNC signals and last longer, making them a little wider. The period of blinking/blanking time before the SYNC signals is referred to as the front door, and the period after the back door.

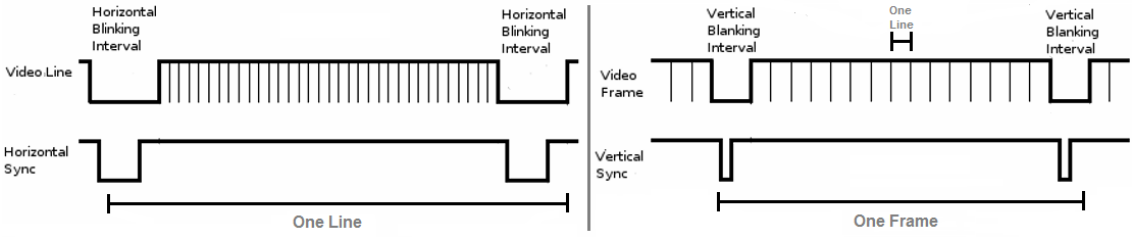


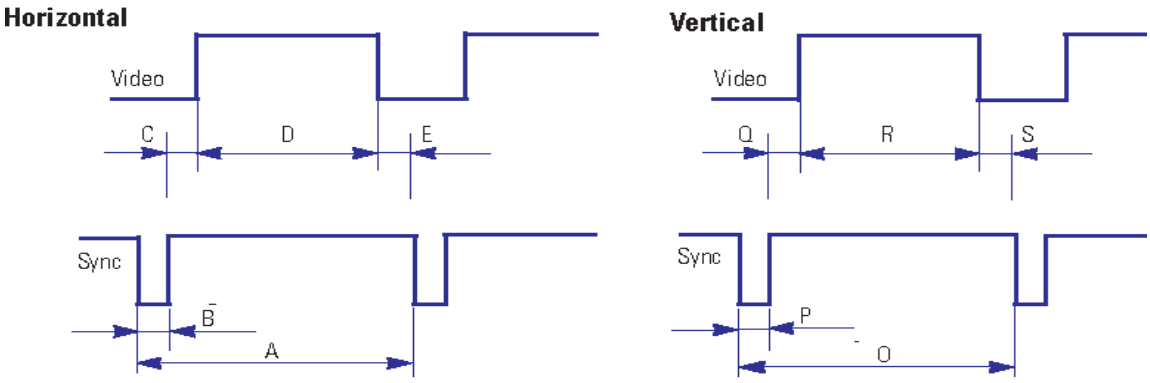
Figure 4.2.1.2.c VGA timing for V-SYNC and H-SYNC windows

The VGA signal was designed to be displayed on CRT monitors, which is the reason the blinking and blanking intervals exist, giving the monitor time for its electron gun to realign itself. Additionally, because RGB values transmitted

through VGA are a continuous waveform after the initial DAC from the PC, the number of horizontal pixels displayed by the CRT must be determined by a pixel clock. The clock timing is determined based on which video display mode is currently being used.

There are 3 other important VGA pins, the DDC clock, DDC data, and DDC return, which allows the display to comminute with the PC and determine which display mode will be used to transmit the data. Figure 4.2.1.2.d shows the timing specifications of various video modes defined by the original IMB standard and VESA standards.

As seen in Figure 4.2.1.2.c, a row of pixels is transferred in the time specified by length A, which is the distance between the front edge of each HSYNC pulse. B specifies the width of the HSYNC pulse. C and E are the front door and back door times, respectively, which surround the HSYNC pulse signal. D is the time during which actual pixel data is transmitted. The vertical timings can be interpreted similarly to the horizontal timings, O being the time for a full frame, P the VSYNC width, Q and S the front and back door times, and R the actual time it takes to transmit the frame.



	IBM		VESA						
Measure	Unit	640x480 60Hz	720x400 70Hz	640x480 75Hz	640x480 85Hz	800x600 75Hz	800x600 85Hz	1024x768 75Hz	1024x768 85Hz
F_HSYNC	kHz	31.469	31.469	37.5	43.269	46.875	53.674	60.023	68.677
A	us	31.778	31.777	26.667	23.111	21.333	18.631	16.66	14.561
B	us	3.813	3.813	2.032	1.556	1.616	1.138	1.219	1.016
C	us	1.907	1.907	3.81	2.222	3.232	2.702	2.235	2.201
D	us	25.422	25.422	20.317	17.778	16.162	14.222	13.003	10.836

E	us	0.636	0.636	0.508	1.558	0.323	0.589	0.203	0.508
F_VSYNC	Hz	59.94	70.087	75	85.008	75	85.061	75.029	84.997
O	ms	16.683	14.268	13.333	11.764	13.333	11.758	13.328	11.765
P	ms	0.064	0.064	0.08	0.671	0.064	0.056	0.05	0.044
Q	ms	1.048	1.08	0.427	0.578	0.448	0.503	0.466	0.524
R	ms	15.253	12.711	12.8	11.093	12.8	11.179	12.795	11.183
S	ms	0.318	0.413	0.027	0.023	0.021	0.019	0.017	0.015
Pixel Clock	MHz	25.175	28.322	31.5	36	49.5	56.25	78.75	94.5
HSYNC +/-		Neg	Neg	Neg	Neg	Pos	Pos	Pos	Pos
VSYNC +/-		Neg	Pos	Neg	Neg	Pos	Pos	Pos	Pos

Figure 4.2.1.2.d Precise Timing Specifications for VGA Display Modes

4.2.1.3 Analog to Digital Conversion:

In order to display frames transmitted through VGA on our LED array, we would have needed to first obtain the signal in a digital format and build each frame. This is because the VGA format transmits data in horizontal lines and our display needs the data in vertical lines. After each frame is constructed, the data must then be retransmitted a single column at a time. Additionally, this would have allowed us to perform processing on each frame, which might include cropping and resizing. Pre-buffered data can also be accommodated easier if we convert the signal to digital because the VGA stream and pre-buffered frames would be able to use the same output to the LED array.

For these reasons, ADCs would have been required. From timing diagrams in Section 4.2.1.2, we can see that the pixel clock runs at 25.175 MHz at a resolution of 640x48. At each of these pulses the analog RGB lines need to be read so 3 ADCs would be needed in total. The voltage on each pin ranges from 0 to 7 volts. With these factors considered, the ADC0801S040 seems to be a good choice for an ADC. The ADC0801S040, has an 8 bit output and operates between 2.7 V and 5.5 V, so the input signal would have needed to be scaled before going into the ADC It also has a maximum speed of 40MHz, and a clock input which could be tied to the pixel clock. This ADC costs around $4, however, it seems likely that enough could be obtained with free samples.

4.2.2 HDMI:

HDMI or High Definition Multimedia Interface is one of the possible inputs we had considered supporting in our POV display project. HDMI input would have allowed us to receive a signal in a format that is quickly gaining popularity and is currently available on many devices. The main reason we considered HDMI support is because Digilent has a Xilinx FPGA based board available with built in HDMI support, and most modern DVD players and laptop computers have HDMI outputs. This section is mainly focused on how we would have gone about receiving the HDMI signal on the Digilent Atlys board and then translated that signal into a format we can use to display it on our LED array.

4.2.2.1 HDMI Signal Standards:

The HDMI standard indicates that the term used to describe HDMI inputs is “HDMI sink”, and the term used to describe HDMI outputs is “HDMI source”. Our POV display would therefore have been the HDMI sink and any device connected to our display would have been the HDMI source. HDMI has two separate communication channel protocols that we must become familiar with: DDC, and TMDS. Another important signal that must be considered is the TMDS clock signal. HDMI provides content protection capabilities through HDCP or High-bandwidth Digital Content Protection. HDCP will not be necessary for our project so we will not consider it in our research. HDMI is also capable of sending control signals in both directions, allowing the connected devices to send commands to each other. We would have most likely not taken advantage of HDMI control signals. Our main focus for HDMI signal standards will be on the DDC and TMDS communication channels. The pin configuration for an HDMI cable is shown in the following Table 4.2.2.1.a.

PIN	Signal Assignment	PIN	Signal Assignment
1	TMDS Data2+	2	TMDS Data2 Shield
3	TMDS Data2-	4	TMDS Data1+
5	TMDS Data1 Shield	6	TMDS Data1-
7	TMDS Data0+	8	TMDS Data0 Shield
9	TMDS Data0-	10	TMDS Clock+
11	TMDS Clock Shield	12	TMDS Clock-
13	CEC	14	Reserved
15	SCL	16	SDA
17	DDC/CEC Ground	18	+5V Power
19	Hot Plug Detect		

Table 4.2.2.1.a HDMI Pin Configuration

DDC or Display Data Channel provides a way for the display to communicate which resolutions are supported to the graphics output device. HDMI uses a DDC protocol named Enhanced Extended Display Identification Data or E-EDID. This is represented by a 256 byte binary file stored in ROM on the display. Since

we would have created the display we may have to create our own EDID data file in order to properly have a device such as a DVD player send the correct resolution picture. Creating a compatible EDID file may have proven beneficial to us since it may eliminate the need for down scaling the resolution of the input since the file would communicate to the HDMI source which resolution it should be sending to the sink. Table 4.2.2.1.b below shows the structure and requirements of EDID information.

Description	**Required**
Block “0” Header	Yes
ID Manufacturer	Yes
ID Product Code	Yes
ID Serial Number	No
Week of Manufacture	No
Year of Manufacture or Model Year	Yes
EDID Version	Yes
EDID Revision	Yes
Basic Display Parameters and Features	Yes
Display x, y Chromaticity Coordinates	Yes
Established Timings	No
Standard Timing Identifications	No
Preferred Timing Descriptor Block	Yes
Range Limits Descriptor Block	No
Monitor Name Descriptor Block	No
Other Descriptor Blocks	No
Extension flag	Yes
Checksum	Yes

Table 4.2.2.1.b EDID Information and Requirements

TMDS or Transition Minimized Differential Signaling is an encoding protocol that takes place for the HDMI audio and video data. “Transition Minimized” means that the number of transitions in the digital signal is reduced as low as possible. This means that the transition from 0 to 1 or vice versa will happen as few times as possible in the transmitted signal. The reason for this is to minimize the chance of the signal degrading along the transmission line. “Differential Signaling” means that there are two different signals being sent, one on each cable in a twisted pair. One of the signals is the audio and video data, and the other signal is the inverse of the first. The receiving end compares the first signal with the second and calculates the difference between the two; this data is then used to make corrections when possible. There are three TMDS channels in an HDMI cable; each channel has its own twisted pair. There is also a TMDS clock

signal, which itself is not a TMDS signal, but simply a digital signal to help synchronize the TMDS signals and allow for the differential calculations needed for error correction. The following Figure 4.2.2.1 shows a simple flowchart of how we will be handling the HDMI TMDS signal with the HDMI input on an FPGA.

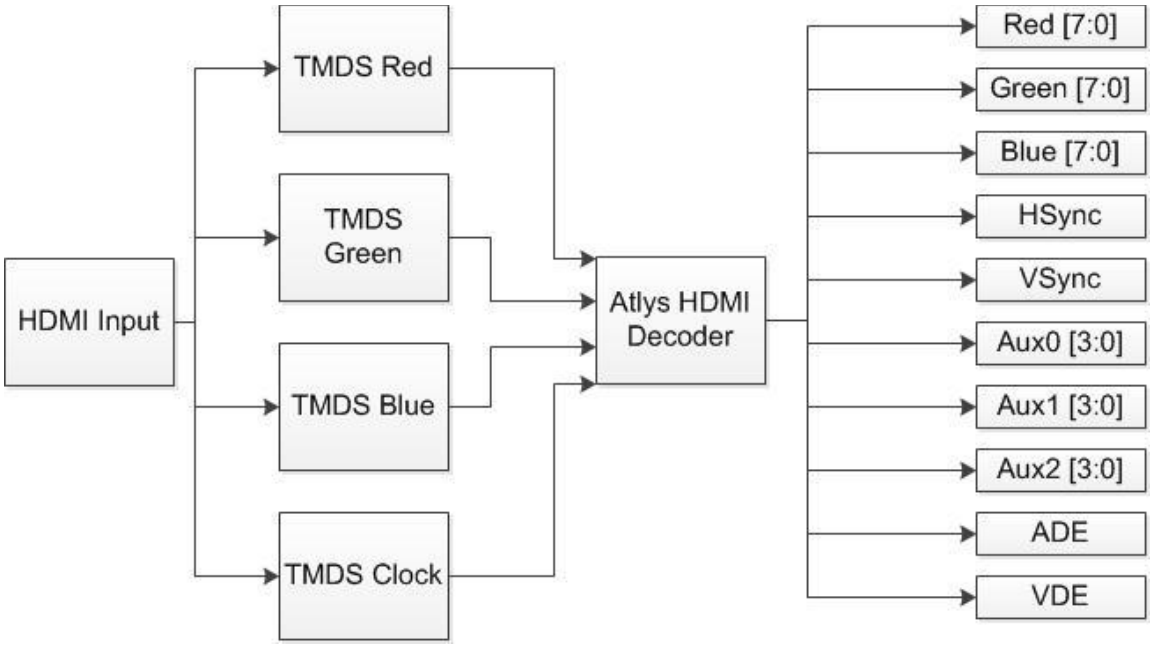


Figure 4.2.2.1 TMDS Input Flowchart

4.2.2.2 Signal Sampling:

If we were to use an HDMI input we would have used the Atlys board by Digilent. The Atlys board is based on the Xilinx Spartan 6 FPGA, and has built in HDMI inputs and outputs. The HDMI inputs and outputs on the Atlys board automatically encode or decode the TMDS signals for input or output. There is a given reference design available which uses the onboard switches to choose which video mode to use (resolution and refresh rate). We would have used the Atlys board exclusively as an HDMI sink. All of the data received from the HDMI port would have then been sent by some communication method to the secondary spinning microcontroller which would have organize the data into the appropriate latches for display on the LED array.

4.2.3 Video Processing (Stationary Controller):

Various forms video processing may be required depending upon the required format of the frames we build in the stationary controller, and how these frames are obtained. The format in which we need the frame data is dependent on the specifications for the LED array, including its size and how precise it can

represent RGB colors. This was determined by our choice of LED controllers, which in turn determined what types of image processing was required.

4.2.3.1 Color Depth Reduction:

When building each frame, there was an RGB value for each pixel in that frame. It is quite likely that these RGB values have a much higher color depth than our display is capable of handling. In code, we needed to convert these RGB values into a lower color depth. The simplest way of doing this is to truncate off the least significant bits. If we expect that the RGB color data we obtain will be in 'true color', or 24 bit color, then to reduce it to 8 bit color we would truncate the Red and Green data to their 3 most significant bits each, and for the Blue data, to its 2 most significant bits. Since we only want 8 bits for the color, Blue is picked to be the color with fewer bits because the human eye is less sensitive to changes in blues when compared to red and green. Figure 4.2.3.1 shows the same picture in various color resolutions.

Figure 4.2.3.1 Example of image shown in 4 bit and 8 bit color depth

4.2.3.2 Frame Resizing:

Since the possibility exists that we may not be able to receive the exact resolution we desire for our display, we may need to resize the frames as they are buffered. This can be accomplished most simply by truncating sections of the frame and displaying a cropped version. In the most ideal scenario, we will receive frames at a resolution of 640 x 480, which can then be easily cut in half to a resolution of 320 x 240. It may also be possible to employ algorithms on the entire 640 x 480 frame which would reduce it to 320 x 240 by using blurring techniques, but this could have an effect on how nicely the images look on the display, and they also come with a heavy processing cost.

4.2.3.3 Frame Skipping:

Assuming there is a certain amount of image buffering and that we are receiving frames into this buffer at a particular rate, it is possible that we may receive more frames than we need and might need. Our display is intended to show 30 frames per second, and most video modes provide frames at around 60 Hz. In this simple case we receive frames at twice the frequency we need them, we could simply use every other frame. A more complicated frame skipping algorithm may be needed frequency at which frames are buffered can't simply be cut in half.

There is also the possibility of increasing or decreasing the rotation speed of the display, which determines our number frames per second, to a value such that that it even divides evenly with the frequency of frames being received. As an example, if the video mode we are in is providing frames at 70 frames per second, we could display this nicely if we changed our rotation speed to 35 frames per second and then simply used every other frame. It seems likely however that we can receive 60 frames per second and display at the desired 30 frames per second.

4.2.3.4 Video Compression:

The real time requirement of transferring the frames between the stationary board and the rotating board is of some concern. Calculations for the required data rate seem to suggest that the amount of data we are transferring is small compared to the bandwidth, if it does become an issue due to overhead from various transfer protocols it would be good to have an efficient solution for minimizing the amount of data that needs to be transferred.

Video compression is possible because within each frame exists redundant data that could be described more efficiently, with or without loss of information. Redundant data can exist in two forms, spatial and temporal. Spatial redundancy occurs when there are repeated pixels in a single frame. Temporal redundancy occurs when pixels values do not change from frame to frame.

One of the simplest forms of compression involves simply throwing away the least significant bits of each RGB color, which would allow each pixel to be represented by fewer bits. Since our display is a 256 color display, this form of compression will almost certainly occur, and is discussed in more detail in section 4.2.3.1 dealing with Color Depth Reduction.

Run length encoding is a very simple compression method that deals with spatial redundancy. With run length compression, when a pixel color value C is identical for some sequence of length L, it can be represented by (C, L). The type of compression works best on computer generated images because of the increased likelihood of unvarying pixels. Combined with the color depth reduction that is to occur, the likelihood of identical pixels in sequence is increased and could greatly reduce the size of the data.

Often in transferring the signal between the source and display, composite formats are used instead of having 3 separate outputs for RGB. In the composite format, instead of RGB values, a Luminance "Y" value and a Chrominance is used to represent each pixel. Chrominance is represented by two signals, I and Q if using NTSC video, or U and V if using PAL video. Figure 4.2.3.4.a shows how the luminance and chrominance are calculating using the NTSC and PAL video standards. This in it of itself does not compress the video, it merely combines the RGB values into a single stream and it also allows compression algorithms to take advantage of the properties of Luminance and Chrominance. A simpler composite would involve concatenating the individual RGB values into a single byte, since we are using 8-bit color.

NTSC video	PAL video/Digital recorders
Y = 0.30R + 0.59G + 0.11B	Y = 0.3R + 0.6G + 0.1B
I = 0.60R – 0.28G – 0.32B	U = (B – Y) x 0.493
Q = 0.21R – 0.52G + 0.31B	V = (R – Y) x 0.877

Figure 4.2.3.4.a NTSC and PAL Calc. for Luminance and Chrominance

One form of compression relies on the premise that the human eye has poor detection of changes in chrominance values, with heavier importance placed on Luminance. Based on this nature, we could use a compression technique that involves throwing away much of the chrominance data and uses interpolation to determine the chrominance value at each pixel location instead. This method of compression is referred to as an Interpolative compression scheme. As an example of this method, we will throw out 3 out of 4 columns of chrominance values and 3 out of 4 rows of chrominance values, reducing the total amount of values by a factor of 4. Figure 4.2.3.4.b shows a matrix of chrominance values, the blue dots representing values thrown out, and black dots representing values to remain in the matrix. Shown also is a sample calculation using interpolation to approximate the missing chrominance values.

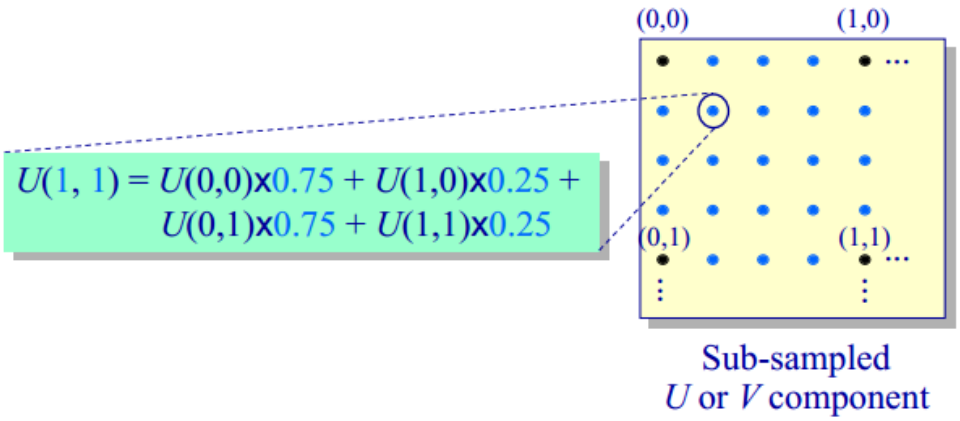


Figure 4.2.3.4.b Method for Interpolating Chrominance Values

Using only spatial compression methods, the amount of data that is required to be transferred can be vastly reduced, as has been seen. This helps to reduce any issues involving the data transfer rate between the two microcontrollers being too slow. It is important to note that using video compression involves a significant tradeoff between the required processing time and data size. Some forms of compression require additional processing power at the transmitting and receiving side because of the mathematical calculations that would need to be performed.

In our real time application, the right balance between compression and data rate is critical. On the stationary processor, where we have a faster clock speed, we can perform color depth reduction and combine the RGB values into a single byte before transmitting. Using those two techniques alone, the rotating processor would not need to perform any calculations to decode the video. The rotating processor avoids any extra processing because it will receive the data in the format that it ultimately needs. The rotating processor would be able to dedicate its cycles fully to reading the frame buffer and writing to the LED array.

4.3 LED Array:

This section of research will cover the exploring the different possibilities of not only what type of LEDs to use but different possibilities to control the LEDs as well.

4.3.1 LEDs:

There are a several available RGB LEDs all with different characteristics and specifications. Some important unique characteristics required by the POV display include size and mounting options. In order to reduce the appearance of streaking when the POV display is running, the distance between each vertical LED needs to be minimized. This eliminates the most common and popular case style of LEDs, T-1 3/4 package. The T-1 3/4 style LEDs have a width (as viewed from the top) of 5.9mm. Therefore, we turned to researching available surface mount LEDs. In general, surface mount devices or SMDs offer a much smaller package and are design for use and easy integration into printed circuit boards. One available surface mount type LED is manufactured by Multicomp. Multicomp's OVS-33 Series SMD Super Bright LED is only 2.8mm wide as viewed from the top. This is about half the size of the T-1 3/4 style LEDs. This will allow us to group the LEDs on the array much closer reducing the appearance of streaking as the POV display spins. Figure 4.3.1 shows the pin information for the OVS-33 Series SMD Super Bright LED.

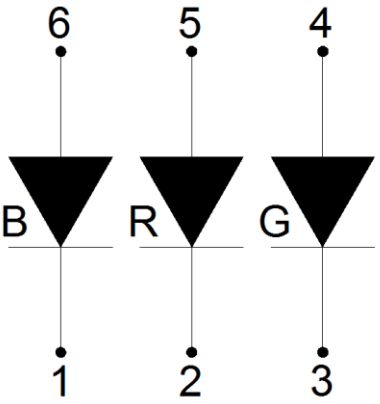


Figure 4.3.1 OVS-33 Pin Information

4.3.2 LED RGB Control:

There are several different methods for controlling RGB LEDs. Two methods that we focused our research on were Pulse Width Modulated Controllers and using latches with a resistor network.

4.3.2.1 Pulse Width Modulation:

The brightness of an LED is determined based on the amount of current the LED receives during a sample period. Pulse Width Modulation is a form of controlling the brightness of an LED by controlling the average current a LED receives during one cycle or period by varying the width of a pulse. Several manufactures offer a variety of LED controllers but during some preliminary research, it was discovered that Texas Instruments offered the best selection and supporting material for their line of LED controllers. Two LED controllers we focused our research on will be the TLC5971 and the TLC5940.

4.3.2.1.1 TLC5971 LED Controller:

Texas Instruments TLC5971 LED Driver offers 12 Channel, 16 Bit pulse width modulated control of LEDs. TI defines the design application of the TLC5971 is for RGB LED cluster lamp displays. The TLC5971 allows control of up to 12 LEDs broken into groups of (4). Each group containing controls for (3) LEDs or the RGB values of each LED. Each LED has individually adjustable output with 65,536 steps. As well, the TLC5971 allows for serial data communications and cascading of an n number of controllers together with a maximum data rate transfer of 20 MHz.

4.3.2.1.2 TLC5940 LED Controller:

Texas Instruments TLC5940 LED Driver offers 16 Channel, 12 Bit pulse width modulated control of LEDs. TI defines the design application of the TLC5940 is

for full-color LED displays, LED signboards and a general high current LED driver. The TLC5940 allows control of up to 16 LEDs but unlike the TLC5971, the outputs are not broken into RGB groups. With the TLC5940, the 12 bit pulse width allows for each LED to be individually adjusted with 4,096 steps. Like the TLC5971, the TLC5940 allows for serial data communications and cascading of an n number of controllers together with a maximum data rate transfer of 30 MHz. One additional useful feature of the TLC5940 is its XERR output. The XERR output allows for notification if an LED goes out through its LED Open Detection. As well, XERR also allows for notification of an over temperature. Both features that may benefit the functionality of the POV display.

4.3.2.2 Latch Control:

Each LED in our LED array needs to flash its appropriate color at the exact same time as all of the others, so the colors that each LED is to display must be stored before outputting to that LED. One way of accomplishing this would involve using latches. Each LED has 4 inputs, RGB colors and ground. One LED that we considered using had a color depth of up to 256 colors. Each LED would then require 8 bits of color data to determine which color it should output. If we are displaying at 320 x 240 resolution, our LED array will have 240 individual LEDs, and a latch will be required for each one of them.

Each latch would need to be able to contain 8 bits. We can use a resistor scheme the VGA signal was generated in section 3.2.1.2 on VGA Signal Sampling, which would reduce the 8 bits of information down 3 lines which would connect directly to the LEDs. In order to address each of the 240 latches, we could have used an 8 to 256 decoder, or combination of decoders. This approach requires 8 addressing lines from the rotating processor, and 8 data lines, as well as a line that would be used to update the output of the latches all at the same time, requiring 17 output lines in total. One possible way to reduce the number of required outputs from the rotating microcontroller would have been to use a counter to address the decoders, as seen in Figure 4.3.1.2. An 8 bit counter would require 2 output lines from the controller, one to increment it and one to reset it. Its output would be used to address each decoder one by one. The 8 data lines will still be required, plus the two counter lines, and one final line used to activate the latch output, so in total 11 outputs would be required from the processor.

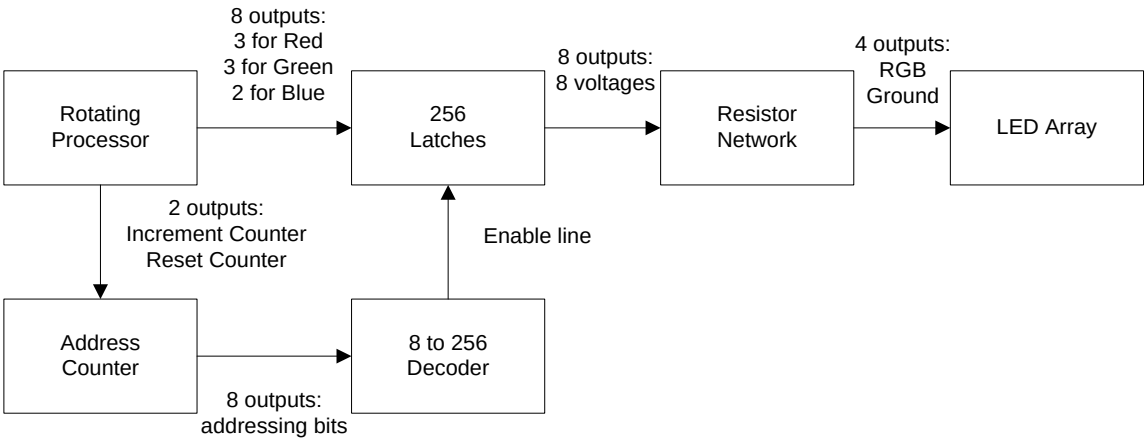


Figure 4.3.1.2 Latch control Implementation

The resistor network would have used the 8 outputs from each of the latches, which are 8 voltages, and would have converted the 8 bits of data into 4 lines with specific voltage and current for the RGB and Ground connections LEDs have. Overall this scheme involves the huge dilemma of wiring all of these connections. In total there are 240 resistor networks which convert the 8 bits down to 4, and then 4*240 (960) connections to the LEDs. Additionally, there are 240 connections from the decoder to the latches, and an 8 bit data bus which must connect to each of the 240 latches.

4.4 Communications:

Since one of our objectives with this POV device was to send a signal encoded with an image or frame of a video, we needed a way to transfer data from the stationary side of the device to the rotating side of the device. For obvious reasons the simple solution of a wire was not applicable without some special configurations. There are two options we came up with to solve this issue: a co-axel wire that is strung through the point of rotation with a rotatable joint or wireless transmission via a medium like Wi-Fi or Bluetooth.

The following sections will cover our findings for both methods, and a comparison of both methods and their pros and cons for our specifications. The final section will sum up our eventual decision and explore the reasons for our choice.

4.4.1 Requirements

Before we can even discuss either method of communication to our rotating device we need to discuss the data requirements that we needed in order to implement the system. This is so we can effectively decide the best fit for our POV display.

First we needed to determine the number of bits required for one LED to display a single color. Since we decided we wanted two-hundred-fifty-six colors we knew that we needed

about eight bits of information to display a specific color on a single LED. However, we don't want to display on just a single LED so we need to be able to determine which LED we want to send this color to. Since we planned on having four-hundred-eighty LEDs as our vertical dimension we knew we needed nine bits of information to tell the processor which LED we were addressing. That is seventeen bits total that is needed to turn a single LED in the array a specific color, the eight bits needed for the color plus the nine bits needed for the specific LED in the array. Figure 3.4.1 gives a visual representation of this concept.

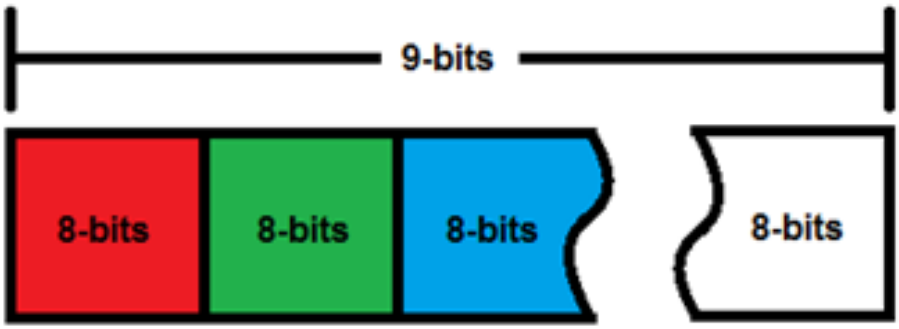


Figure 4.4.1 Data Array

This only turns a single LED in the array a single specific color. We need to turn all four-hundred-eighty LEDs a variety of colors that means we need to send a seventeen bit word to each LED at once for a single vertical line of our frame. We also need to tell the device when a new line should be displayed, so we should add two bits for an end of line message and a beginning of line message. For the sake of discussion and since it is better generally to overestimate then under estimate we will say two bits. This means we need to multiply the seventeen bit word by four-hundred-eighty LEDs and add two bits to the end of that to get the total bits needed for a single vertical line. In other words we need 8162 bits to display a single vertical line of our frame. Now to display the full frame we need to multiply this word by six-hundred-forty, since this is our horizontal dimension. This brings the data we need to transfer up to about 5.3 megabytes. We aren't done yet since we also need end and beginning of frame bits for this word, which brings us up two more bits. This is just a single frame and we need to display these frames reliably at thirty frames per second. This means we need to send the above frame data thirty times per second. This means in one second we are sending a little over a hundred-fifty-six megabytes, or more specifically: 156,710,460 megabytes.

This means no matter what form of communication we choose to use it has to be at minimum capable of sending this much information reliably. That being the case would have probably wanted a data transfer rate a little higher than this, maybe even twice as high as this to make it reasonable that with errors we would have still been able to maintain a steady transmission.

4.4.2 Wired Communications:

There are many forms of wired communications currently being implemented on a daily basis in today's high speed world. There are several design criteria which restricted some of the available forms of wired communications. From our specifications and project design criteria, we knew that our platform would be spinning at a rate of 1800 rotations per minute. Through some preliminary research, it was found that the larger number of conductors being transmitted to a rotating platform resulted in a smaller maximum allowed RPM's. In other words, any conductor larger than four strands would have been unpractical for this application. Therefore, the researched was focused on two types of wired communications, Fiber Optics and Cooper Coaxial Cable. In both situations, the wired communications needed to convert existing Ethernet communications ports on the microprocessors to a form that can be transmitted over their respective medium. With the idea of using the existing Ethernet ports and protocols of the microprocessors one additional criterion of the wired communications would have been transmission rates. Currently the standard threshold requirements for Ethernet communications are 10 Mbs, 100 Mbs, and 1000 Mbs or 1 Gbs. An additional design criterion for wired communications would have been to implement the communications with inducing the minimal amount of interference to the signal. The last design criteria for wired communications would have been to evaluate cost benefits between coaxial cable and fiber optic cable. Below a summary of the design criteria is listed and was a guide for determining the vitality of each type of wired communications.

- ✓ Rotating Speed: 1800 RPM's
- ✓ Transmission Rates: 10/100/1000 Mbs
- ✓ Little to no induced interference
- ✓ Cost

4.4.2.1 Fiber Optic Communications:

In the following section we researched the requirements for using fiber optic communications to transfer the data from the stationary side of the POV display to the rotating side of the POV display.

4.4.2.1.1 Fiber to Ethernet Conversion:

The first portion of research on fiber optic communications was to determine the requirements for converting Ethernet communications to fiber communications. Fiber optic communications use either single mode or multimode fiber cable. Therefore, in addition to determining how to convert Ethernet to fiber, a review of single mode verse multimode was required to determine which is preferred for Ethernet communications.

Single mode fiber optic communications have a smaller core size than multimode fiber cables and, as the name implies, single mode fiber cables only operate with one optical light. Generally, most single mode fiber systems operate at 1300 nm or 1550 nm wavelengths. As well, single mode fiber systems require very strict mechanical connections due to the smaller core size. Multimode fiber systems operate at 850 nm or 1300 nm wavelengths and have a larger core size than single mode fiber cables. However, due to the larger size of the multimode core, multimode systems suffer from high attenuation can therefore cannot operate at the same distance as single mode systems. One advantage the larger core size of multimode systems is the high capacity and transmission data rates. Multimode systems can transmit data at rates of 10 Mbs to 10 Gbs. As well, in general, the cost of multimode fiber systems is less than the cost of single mode fiber systems.

Upon reviewing the differences between multimode and singe mode fiber cable, the research on fiber to Ethernet conversions will focus in multimode fiber communications only. The difference in transmission length between multimode and single mode is negligible for the application of the POV display as the maximum transmission distance will not exceed more than ten feet. As well, the higher cost and lower transmission rates of single mode fiber cable make multimode fiber a clear choice for the application and use of the POV display.

Various Ethernet to fiber solutions exist on the market today. Ethernet to fiber converters or media converters are used in various industries from substation communications to bringing internet to homes across the nation. Several manufactures provide fiber to Ethernet solutions all within the design criteria of the POV display. Table 4.4.2.1.1 below list a few available solutions including product specifications and cost.

Part Number	Mfr.	Supported Data Rates	Fiber Connector	Ethernet Connector	Cost
EIR102-MT	B&B Electronics	10/100 Mbps	MM ST	RJ-45	$199.00
FCU-100SC	Aaxeon	200 Mbps	MM SC	RJ-45	$62.00
ME-1600-MM2-ST	Support Systems Int.	10/100 Mbps	MM ST	RJ-45	$69.50

Table 4.4.2.1.1 Fiber to Ethernet Converters

4.4.2.1.2 Fiber Optic Rotary Joints:

Fiber optic rotary joints or FORJs are used to make the junction between a stationary fiber cable and a rotating fiber cable. As discussed in the main section,

the fiber optic rotary joints must be capable of rotating at speeds of 1800 RPM's while not inducing a significant amount of inference. Several fiber optic rotary joints are available on the market. One company providing a wide range of fiber optic rotary joints is the Moog Components Group. Almost all available rotary joints can support both multimode or single mode fiber cable and a wide wavelength range. Therefore, the research on fiber optic rotary joints was focused on the maximum rotating speed and minimum induced noise into the signal.

Although Moog provides a variety of fiber optic rotary joints, the manufacture however does not provide any FORJs that have a maximum rotating of 1800 RPMs or higher. Fortunately, other manufactures do provided FORJs that can operate at the rotating speed required for the POV display. One alternative to Moog is Princetel and their line-up of available FORJs. In particular, Princetel offers the MJX series product line. The MJX series fiber optic rotary joints are capable of rotating at speeds up to 2000 RPMs. In addition to a maximum rotating speed of 2000 RPMs, Princetel's MJX series fiber optic rotary joints have an insertion loss of less than 2 dB (less than 0.5 dB typical) with an insertion loss ripple of less than plus/minus 0.25 dB.

It is evident that the MJX series fiber optic rotating joint met and exceeded all design criteria for the POV display. Depending on what fiber connector and wavelength is required to connect to the Ethernet convert, Table 4.4.2.1.2 below shows available MJX rotating joints and their respective part number.

Part Number	Fiber Connector	Wavelength
MJX-850-ST	ST	850
MJX-850-SC	SC	850
MJX-131-ST	ST	1310
MJX-131-SC	SC	1310

Table 4.4.2.1.2 MJX Part Numbers

4.4.2.2 Coaxial Copper Communications:

In this section we researched the requirements for using a copper coaxial cable to transfer the data from the stationary side of the POV display to the rotating side of the POV display.

4.4.2.2.1 Coaxial to Ethernet Conversion:

Coaxial to Ethernet conversion is the back bone to modern cable modem internet. Coaxial communications relay on a single copper core that is shielded by an equal but opposite current. This provides one fundamental advantage over

fiber communications, the ability to conduct power over the same line as the data signal. This allows the conversion of signals to coaxial using simple in-line converters that do not require any additional power supply. One such in-line convert is provided by EnConn. The EnConn EOC-IN-B Ethernet over Coax allows for the transmission of Ethernet of coaxial cable at transmission rates up to 10 Mbs. As stated early, the EOC-IN-B is an in-line or passive device. This means the EOC-IN-B does not require any additional power. In addition, the EOC-IN-B is a compact design allowing the device to be installed using less space not only on the stationary platform but the rotating chassis of the POV display. However, the EnConn EOC-IN-B only supports Ethernet communications up to 10 Mbs. In the case that the communications to the LED array will require a higher bandwidth additional research is required to determine the best alternative.

One alternative from EnConn is their EOC-AN and EOC-IN Ethernet over Coax extender allows for transmission of Ethernet at rates of 10 Mbps up to 100 Mbps. The EOC-AN converter requires a DC power input of 12V but the EOC-IN does not require any power input. This means we could use the EOC-AN converter on the stationary side of the POV display and power the converter from the power supply. We would then install the more compact EOC-IN on the rotating side of the POV display. Another alternative would be Pulse Link's PL3302 Ethernet over Coax bridge. The PL3302 allows Ethernet communications of 10 Mbps, 100 Mpbs and 1000 Mbps. Although the PL3302 allows for Ethernet communications up to 1000 Mbps, the Ethernet bridge will require DC power on both the stationary and the rotating side of the POV display. Another downside to the PL3302 is its size. The PL3302 dimensions are 6" wide x 1.75" high x 4.75" deep. Table 4.4.1.2.1 compares the differences between all converters.

Part Number	Mfr.	Supported Data Rates	Coax Connector	Ethernet Connector
EIR102-MT	EnConn	10 Mbps	BNC	RJ-45
EOC-AN/IN	EnConn	10/100 Mbps	MM SC	RJ-45
PL3302	Pulse Link	10/100/1000 Mbps	MM ST	RJ-45

Table 4.4.2.2.1 Coax to Ethernet Converters

4.4.2.2.2 Coax Rotating Joint:

Once the Ethernet is converted to Coax, just like with fiber, the coax will require a rotating joint to make the bridge between the stationary side and the rotating side. Although extensive research was done, only one practical solution was found. Mercotac manufactures a variety of rotating joints and slip rings. Included Mercotac's product line is a two conductor Model 205 high speed, low torque

rotating joint. The joint is not explicitly design for coaxial communications but due to the extremely low electrical noise induced by the joint and the fact that a coaxial cable can be simplified to two conductor cable makes the Model 205 a feasible solution for transmitting the coax cable from the stationary side to the rotating side. Some other advantages of Mercotac's rotary joints are life expectancy and maintenance requirements. The Model 205 rotary joint is manufactured with a life expectancy of several hundred million revolutions. If a rotary joint is installed and operated under all specified conditions, Mercotac claims the joint can even last for over a billion revolutions. As well, the joints are manufactured for to be maintenance free, meaning they will not deteriorate the signal over the lifetime of the joint. Figure 4.4.1.2.2 below shows a typical mounting and wiring of a Model 205 joint. As well, Table 4.4.1.2.2 list all models and their corresponding specifications for the 205 joint. All Model 205 joints have two terminals, operate at a voltage range of 0-250 V AC/DC and a current rating of 4 Amps at 240 V AC.

Figure 4.4.2.2.2 Model 205 Rotary Joint for Rotary Interfaces

Part Number	Max. Freq	Max RPM	Ball Bearing	Cost
205	200 MHz	2000	Steel	$28.52
205-SS	200 MHz	2000	Stainless Steel	$37.68
205-H	200 MHz	3600	Steel	$29.62
205-HS	200 MHz	3600	Stainless Steel	$38.37

Table 4.4.1.2.2: Coax to Ethernet Converters

4.4.2.3 Ethernet Protocols:

In order to determine which protocol is most appropriate for our purposes we looked at the protocols TCP, UDP, and using our own. TCP is protocol that is designed to reliability transmit a stream of bytes between two programs running on separate systems. TCP allows a program to request the transmission of data with a single request and then takes care of segmenting it into IP sized packets, which contain a sequence of bytes and a header. TCP handles the scenarios such as out of order transmission, duplicate packets, and lost packets. Out of order packets are rearranged and lost packets and be requested to be resent. Reassembly of the stream of bytes is handled by the TCP receiver, which then passes the data to the program. The TCP protocol favors the accuracy of the data over timely delivery, and uses positive acknowledgement to guarantee reliability. In positive acknowledgement method, the receiver sends an acknowledgement for each packet it receives, and the sender expects to receive the acknowledgement within a certain amount of time, or it will resend the packet because it may have been lost or corrupt. The favoring of accuracy over transmission speed makes TCP generally a poor choice for a real time application.

Another protocol option is to consider is UDP. UDP doesn't use any handshaking and does not guarantee that data is in order and not missing. Any reliability and accuracy checking, as well as error handling must be performed at the application level if it is a concern. In our case we could probably implement these checks at the application level. For instance after each frame is transmitted to the rotating board we could send a UDP datagram back to the stationary one confirming its receipt. UDP is often used for real time systems where losing a packet is preferable to waiting on it, which might make it lend itself better to our application. This would require that we handle the scenario losing a packet appropriately at the application level however, although ideally there will not be any packet loss. Packet loss is unlikely because our two systems are connected back to back via cross-over Ethernet cable, and the communication is limited to those two systems. A UDP packet consists of a header which contains the source port number, destination port number, length, and a checksum, all of which is followed by the actual data.

4.4.2.3.1 Ethernet Software Library:

The stationary FPGA would have communicated with the rotating FPGA using Ethernet wired communications. In this section we will be considering how the Ethernet communications work. This includes software library identification, and protocol selection. Software library identification for the FPGA was more challenging than expected. The Atlys board was expected to come with built in Ethernet functionality but it seems that this is not the case. Xilinx offers software

in the form of Intellectual Property (IP) cores to support Ethernet communications but this core is not free. Licensing fees would cost us over $1,000. To keep the costs of this project low we searched for alternate solutions. There is a website opencores.org which has open source “cores” available for FPGA’s. Cores are FPGA software packages that program the FPGA to function like a certain hardware design. We were able to find a core which implements a 10/100 Ethernet MAC on the FPGA. Using this core we would have been able to use the Ethernet ports on the FPGA’s for communication. If we used the Ethernet core then we would have used the UDP protocol because flow control and acknowledgements are unnecessary for our application. A live video feed cannot afford to retransmit packets. It makes more sense to simply drop any lost packets and continue transmitting the next frames.

Another alternative may have been to use the Ethernet ports in a non-standard way. We could have used the pins on the RJ-45 connector to send the data using our own design. If we picked that route we would not have been using any Ethernet protocols but simply sending raw data through a wire. This would have been the simplest method to design and implement because it would not have required any complicated software library or IP cores. After looking at example code using the Xilinx Ethernet MAC core it was obvious that many hours would be required just to understand the example. The core available through the opencores.org website was even more complex because it lacked documentation and examples. Another fact worth mentioning is that the cores do not work in a straightforward way like C programming. They are actual hardware implementations and should be viewed as such. If we create our own method of using the output pins for the RJ-45 connector we may have been able to simplify communication greatly. We would have created our own header for the data being sent to identify what is being sent. We would have most likely used a clock speed of 200MHz for a 100Mbit/s data sampling rate.

4.4.2.4 Microprocessor Ethernet Hardware:

A possible component to implement Ethernet communication on our boards could have been the Arduino Ethernet Shield, which would have required that we use Arduino boards for the rotating and stationary controllers. The board has a 16 kilobyte buffer and has a connection speed of 10/100 Mb. The board supports both TCP and UDP connections as well as simply transferring single bytes at a time without any protocol. The board contains a library of functions including a server class, client class, and an EthernetUDP class, as well as the main Ethernet class and IPAddress class which allows you to assign the board an IP address.

4.4.3 Wireless Communications:

We considered wireless communications in order to send information from the stationary FPGA to the rotating microcontroller. The wireless communication

must support a high enough bit rate to send a 320x240 color video signal. The color video signal would have had 256 possible colors per pixel, so 8 bits per pixel would have been needed. We would have also liked to transmit 30 frames per second. The minimum required bitrate that we would have needed in order to achieve the desired frame rate would have been 320x240x8x30 which is 18.432Mbps. We considered a 480 LED array supporting a 640x480 resolution. If we had used the higher resolution then our bandwidth requirements would be 640x480x8x30 which is 73.728 Mbps. Both WiFi and Bluetooth are capable of these speeds so we considered both technologies. Generic RF communication was not considered because we did not believe that it would support the bandwidth that would be required for real time video. We also researched if the rotation of the microcontroller would hinder wireless communications.

4.4.3.1 WiFi:

WiFi is the common name for the IEEE 802.11 wireless communication standard. This technology most often uses a 2.4GHz frequency. A large advantage to using WiFi for our wireless communications is that all modern laptop computers have built in WiFi communication capabilities. It was possible for us to write software for a PC that allowed direct WiFi communication between a PC and the rotating microcontroller in order to send text messages or images to be displayed.

4.4.3.1.1 WiFi Protocols:

The specific WiFi protocol we considered is 802.11g. Devices that use this protocol are commonly available and are capable of up to 54Mbps data transfer rates, which is more than enough for our application. WiFi has two possible modes of operation: infrastructure and ad-hoc. Infrastructure is the most commonly used mode, but it requires an existing infrastructure including wireless routers and/or wireless access points. We considered the ad-hoc mode for this project since it does not require any other external hardware. Ad-hoc would allow us to set up a direct wireless connection between the FPGA and the microcontroller for bi-directional communication. Although bi-directional communication would be supported we would only have to communicate in one direction. The following Figure 4.4.3.1.1 shows a comparison between infrastructure and ad-hoc modes of operation.

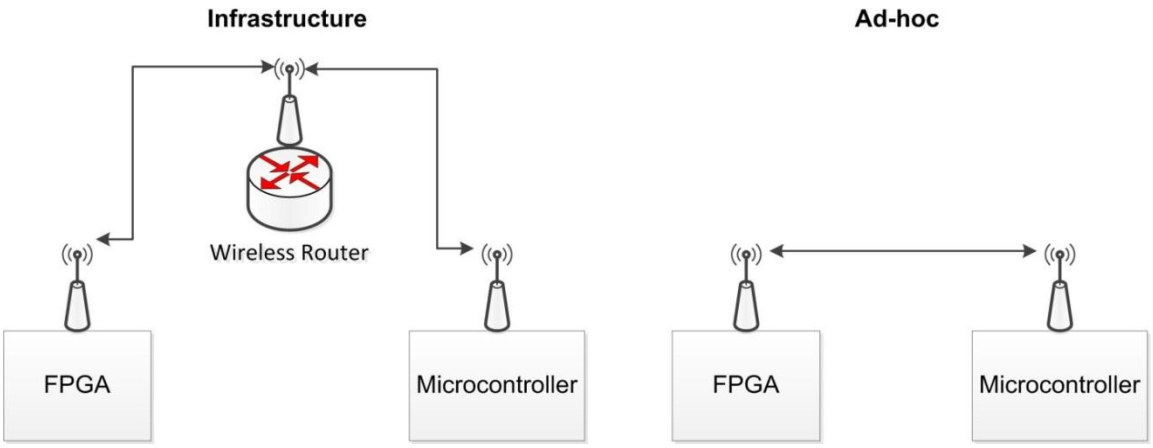


Figure 4.4.3.1.1 Infrastructure/Ad-hoc Comparison.

Because of WiFi's popularity there are many options for WiFi hardware. Digilent offers a WiFi adapter for their boards although it only supports 2Mbps data rates. Arduino shields are also available to add WiFi support. All modern laptop computers and cell phones have WiFi built in. With WiFi supported by so many devices it would be a convenient communication method for us to choose.

4.4.3.2 Bluetooth

Bluetooth may have also been possible as an alternative to WiFi. A possible advantage that Bluetooth may have had is that it is a low power, short range method of communication. Short range for our application would have been desirable for security purposes. Anyone communicating with our display would have to be within about 30 feet of the device. Bluetooth also works on the 2.4GHz frequency, and with the v3.0 specification can achieve data rates of up to 24Mbps. All modern cell phones have built in Bluetooth communication capabilities and allow us the option of creating a mobile application to interface with our POV display. If we could have found suitable Bluetooth hardware compatible with our FPGA and microcontroller then this would most likely have been our preferred method of wireless communication.

4.4.3.2.1 Bluetooth Protocols:

Bluetooth protocols are divided into two categories: controller stack and host stack. The controller stack protocols are protocols built into the Bluetooth module. The host stack protocols are what we will use to deal with our video data to be sent. We looked at both the controller and the host stack protocols relevant to our project in order to help facilitate communication programming during the design phase. First we considered the relevant controller stack protocols which are: Link Management Protocol (LMP), and Asynchronous Connection-oriented Logical transport (ACL). The LMP protocol's function is related to the name of the protocol, it manages the links. More specifically the LMP protocol deals with how

Bluetooth devices can scan and discover each other and set up a link in order to exchange data. Once a link has been set up, a new protocol can take over communications between the devices, in our case this would have been most likely ACL. The ACL protocol is designed to transmit general data packets on a previously set up Bluetooth link. ACL supports Enhanced Data Rate or EDR for increased bandwidth by changing the modulation technique. Theoretically Bluetooth is capable of achieving 24Mbps data rates using EDR. As far as hardware availability, the Digilent boards have a Bluetooth adapter available. There are also shields available for Arduino boards to add Bluetooth support. Adapters for PC's are easy to find and affordable, if the PC doesn't already have a built in solution. All modern cell phones have built in Bluetooth support.

4.4.3.3 Effects of Rotational Speed:

According to a research paper concerning wireless sensor networks, rotational speed is a factor in wireless signal quality. Some of the possible effects that we had to consider are path loss, multipath fading, the Doppler effect, and electromagnetic noise. Path loss is when the path may become interrupted due to line of sight differences along the path that our rotating microcontroller would travel through. Multipath fading could happen if the microcontroller receives the same signal from different paths at the same time. The Doppler Effect is most known for the frequency distortion of sound waves, but would have the same effect on electromagnetic waves as well. Electromagnetic noise could be caused by our mechanical components such as our motor, we probably do not need to consider electromagnetic noise for our project. According to the research paper, electromagnetic noise generated by mechanical components is usually in frequency ranges less than 1.5GHz. We used WiFi and could have used Bluetooth, both of which operated at the 2.4GHz frequency. It is safe to conclude that any electromagnetic noise introduced to our system from the mechanical components should not interfere with our wireless communications.

4.5 Motor:

In order to create the illusion of motion through the phenomena known as persistence of vision it comes to no surprise that we need some sort of motor. This motor needs to be able to rotate whatever apparatus we designed that housed the LEDs, processor, and any other circuit elements we needed to implement the system. It also needed to be able to rotate at the rpm needed to 'trick' the brain into seeing motion. In addition, under the considerations that this project was designed for the use of advertisement we would also like to find a motor that is as silent as possible so as to not be discomforting to those who either work around it or potential customers whom are attracted to it

There are a variety of motors available for such a use but for the most part the motors fall into two categories AC and DC motors. In the following sections we

will not only discuss the above design considerations but also discuss the pros and cons for both the AC and DC motors for each consideration. This discussion will eventually lead to which motor type we picked and the reasoning for the choice. Finally, in the last two sections we will discuss the process of controlling the motor we chose.

4.5.1 Torque Requirements:

As will be discussed further in the chassis design section it showed that we needed about 0.4 Nm to get the motor to just initially spin the LED apparatus. After that the torque requirements were much lower. This however, is actually a pretty large requirement for motor standards considering most cheap motors are rated for far lower ranges, somewhere in the 1/35 to 1/9 horse power range. This proved to be a bit of an issue since that means we needed a high torque motor that also could maintain our revolutions per second value.

4.5.1.1 AC Motor Application for Torque Requirements:

AC motors are perfect for this sort of activity. Our research showed that AC motors tend to be used for high torque requirements and specifically maintained high torque requirements.

4.5.1.2 DC Motor Application for Torque Requirements:

DC motors however, capable of getting high torque at start up but did not maintain them as effectively as AC motors. This did show though that both motors could be used for the application we desired it just seems that the DC motors needed for this application were rather costly. These motors can range from anywhere between two hundred dollars to a few thousand dollars. Used motors that reach these requirements were difficult to acquire, with none at the local Skycraft store available for purchase until we made a lucky break and found one.

4.5.2 RPM Requirements:

Under the consideration that the human eye is tricked into seeing motion at a rate of about twenty-five frames per second and a single rotation of the device is a frame we know we needed a motor that can handle twenty-five rotations per second. This is the bare minimum. We decided that we wanted to overshoot this value by five frames or rotations in order to create a smoother image. Our group thus decided that thirty rotations or frames per second would be adequate.

Thirty frames per second is equivalent to one-thousand-eight-hundred frames or rotations per minute. This means we needed a motor that can make one-thousand-eight-hundred rotations per minute to accomplish the desired frame rate. This rpm value cannot vary much and must be maintained at a constant rate

otherwise there may be distortions in the image due to the increasing and decreasing of the delay between each flash of an LED.

4.5.2.1 AC Motor Application for RPM Requirements:

Through some research it became apparent that AC motors were quite capable of reaching these rpm values required. However, the real problem came in the control aspect of the motor, or more specifically the ability to keep the motor at a constant rpm value. The rpm value of an AC motor can only be varied through either the number of poles the motor is built with or through the electric frequency of the voltage being applied to it. This can be done through an inverter also known as a variable-frequency drive, and this is a plausible solution. It is however, an expensive solution with some inverters ranging from two-hundred to two-thousand dollars. According to further research it also seemed like this problem could be solve by just buying a DC motor since many DC motors are actually AC motors with these variable-frequency drives pre-built within them.

4.5.2.2 DC Motor Application for RPM Requirements:

In the case of DC motors our research revealed that DC motors are generally used for our purposes, and the rpm requirements could be met easily with these motors. Considering DC motors are highly controllable and designed for constant rpm output it makes for a perfect fit with our application since our device would be running under one speed consistently and that speed must have minimal variations.

Controlling a DC motor is actually a relatively simple process. We could either achieve it through a variable resistor albeit this can generate a lot of heat, or we can use some form of PWM circuit to control the rpm of the motor. This left us with some options and both were relatively inexpensive solutions.

4.5.3 Sound Requirements:

Considering this product is for in-person advertisement uses we want minimal obstructive noise. Especially since we planned on playing videos off the device that include sound effects or music. This being the case there are two things that can cause large amounts of obstructive sound and that is either the motor or improper weighting. In the case of improper weighting the torque created by the motor alone causes rattling since the device is not properly balanced or fashioned down. This can be solved through the design of the chassis. However, we still had to watch out for our motor being rather loud. Our research showed that in this case DC motors trumped AC motors. AC motors tend to be much louder than DC motors of all makes and models.

4.5.4 AC and DC Motor Comparison:

With the above considerations reviewed it seems that a DC motor was the best fit. An AC motor, while capable of reaching the rpm values we need would have drastically increase our costs in order to control the speed of the motor. A DC motor is much easier and cheaper to control requiring only a simple variable resistor or PWM circuit. An AC motor also leans to the noisy side of the spectrum of motors, which is something we wanted to limit within our device. As for the torque requirements it seems that both would have passed the needs of our device, but with two thirds of the issues being solved either cheaper or better it comes down to a DC motor being a better choice for our application.

4.5.5 Motor Control:

Since we decided that a DC motor was the best fit as a solution to our mechanical needs, we needed to look further into the methods of controlling the motor's rpm value. Luckily our needs for the motor were relatively simple. The only thing we needed the motor to do was reach our desired rpm value and maintain that value until the device was shut down. We did not need the POV device to vary its speed which would have required more elaborate methods of control.

There turned out to be two methods that were commonly used for DC motor control and that was either using a variable resistor or potentiometer to control the speed or to use a pulse with modulation circuit to control the speed of the motor through the duty cycle. Both methods were found to be inexpensive but the question was which one was better suited for our purposes.

4.5.5.1 Variable Resistance Method to Motor Control:

In the case of the variable resistance method we came to learn through our research that it is the least liked method among motor users. There are quite a few problems with this method, especially if you are looking to constantly vary the speed of your motor or need to get a small speed but still turn on the motor. Lucky for us we didn't want to do either of these so it was still a viable solution.

The main concept that turned us away from this solution though was the heating issues that were common with it. In many cases the resistor had a chance of burning out because of the high power strain on the resistor.

4.5.5.2 Pulse Width Modulation Method for Motor Control:

PWM turned out to be a little bit of an overkill for our project's design since we did not need the motor to be highly controllable just stainable. The device only needed to spin at a constant speed so there was no very high or very low speed requirements for the device, just the ability to spin our apparatus and to spin it consistently at the desired rpm value.

PWM was quite capable of doing this and had very low heating effects on the system as long as you find the right components for the circuit. The biggest drawback to this method however was the noise. When using PWM there is a chance of causing mechanical noise within the system, or a humming sound.

4.5.5.3 Variable Resistance and Pulse Width Modulation Motor Control Method Comparison:

In the end, though we wanted to limit the noise, and while the PWM was capable of doing far more then we needed the controller to do. It did cut back on heat dissipation and we decided that this was the best method. With the motor spinning at a high rpm value heat dissipation was a concern and this would help minimize any additional heat factors within the circuit. In addition, for the sake of scalability having the motor more controllable then our original purpose would leave the device open to any future upgrades to the system that might desire a stricter control system.

4.5.5.4 Sensor Reading Applications for Motor Control:

The final concept we needed to think about for motor control though was actually tracking the rpm values of the motor so we could send a signal back to our controller to vary the input and adjust the speed of the motor to keep it constant. This was very important and had to be particularly accurate so that there were no distortions created within the image due to an increase or decrease in the rpm value and the predicted display rate. We had a couple of things to consider when deciding what form of sensing we were going to do to keep track of any changes in the rotation speed.

The first being the structure of the device itself or in other words the chasse. If the actual apparatus that we rotated was directly pivoting on the motors shaft then the rpm value would sync closely with the motor and there would likely be minimal lose in rpm value. However, if we had decided a gearbox was required to rotate it, such as in the case of using a wired transmission process, we would have lost some rpm value in the translation between the motor and the gearbox. If this was the case then our sensing side would have to be able to measure the rpm value of the apparatus and not the actual motor itself.

The second consideration was our sampling rate. Sense we are taking in a snapshot of this device’s motion we are going to want to know how frequently we want to take that snapshot. This is very important because we need to measure the rpm value rather frequently so that within one second we don’t lose or gain information. To put it in perspective one second is thirty frames and if we are losing even one percent of those we are losing point three frames. That doesn’t seem large but point three frames can become eighteen frames in one minute and ninety frames in five minutes. And each of those is a distortion in the

animation or image. This shows how important our sampling rate is to keep the integrity of the image.

There are two options that seemed to be rather common for rotational speed sensing when it comes to motors. These methods are the Hall effect and infrared sensing methods. Both have their pros and cons so we will look into those and whether they fit well for our application.

4.5.5.4.1 Infrared Sensor:

In the case of using infrared as a method for sensing and controlling the motor the process seemed relatively straight forward. We would have an infrared emitter on the rotating side of the device and an infrared receiver on the stationary side. When the emitter crossed the receiver we would get a “hit” which we could then use to calculate an rpm value. We could then send this value to a micro-controller where we would then determine whether to increase or decrease our motor's rpm value.

This method is very effective for any design we decide to go with. It can work for any motor type and is unaffected by the use of a gear-box, and in fact ideal for such a use. The only concern for this method was the accuracy. Considering we are using an infrared sensor there is a possibility for some failed trips. This means we needed to have a substantial number of samples in order to prevent too many of these errors. This may mean we need more than one infrared sensor in order to prevent these errors such as two or four sets of them.

As an added bonus, the use of this method is great for these projects for other reasons. Since we would be using infrared sensors to create trip points along the devices rotation we can use these trip points for other things besides just calculating the apparatus' rpm value. We can also use this method to predict points within its rotation and create finite starting points to our image, allowing us to split the image where ever we want. This means we aren't just floating the image anywhere the LED happens to start turning on in its trajectory. Uses of this include splitting the “screen” of the device into two separate sides, or drifting images or text in the opposite direction of our rotation.

4.5.5.4.2 Hall Effect Sensor:

The Hall Effect method for measuring and calculating the rpm value of the motor is very efficient for this application. This process is both relatively inexpensive and easy to implement and from our research seemed to also have a very small error rate. There are however, a few issues with this method based on how we decided to use it.

The first issue with this sensing method is its motor limitations. If we decide to use this on the motor side, such as in the case of direct motor-apparatus rotation,

it is limited to motors that have a rotating magnetic pole within. This means AC motors or certain DC motors are a better fit for this sensing method. Since we have already ruled out AC motors because of the expense associated with controlling them among other issues, this left us with a limited number of DC motor types that can be applied to this sensing method. The most obvious type is a brush-less DC motor. This however, is not actually that hindering to our design since brush-less DC motors are actually good for this application and are generally very silent running motors.

The second Issue with these sensors was kind of alluded to with the above paragraph in that they require a moving magnetic pole to measure. This means it would be difficult to implement this sensing method in the case of a gearbox design. We would have to create some form of moving magnetic pole on the rotating apparatus side that would cross the Hall Effect sensor in its rotation. This is possible but there are some unforeseen issues that could occur with the introduction of a moving magnetic field on the rotating side that is not being produced naturally by the components that are there.

4.5.5.4.3 Motor Sensor Comparison:

After looking at both sensing methods and our over-arching design it seemed like the most effective form of controlling our motor would be through infrared sensing. While the accuracy of this method could prove to be an issue, with enough sample points we would be able to make up for any errors that could appear in our measurements. Considering we had already determined we were going to do a wired design it seemed like the best method for solving the issue of tracking the apparatus' rpm value instead of just the motor's rpm value. In addition, it gave us some additional control over our display and flexibility in what we could do with it.

4.6 Chassis:

In the following sections we researched the requirements for the chassis of the POV display. Two topics that required research included what types of materials will be best suited to construct the POV display and how best to transfer the rotational power from the motor to the POV display.

4.6.1 Chassis Materials:

The chassis of the POV display was where a majority of the weight is located. In order to maintain the portability of the POV display some materials we eliminated from our research simply because their excessive weight. However, some weight from the chassis is required and preferred as the chassis must not twist or move while the POV display is running. As well, strength was an important factor as the chassis would be put through a range of forces as the display goes from stationary to full rotation speed. Steel and stainless steel both have high strength

values but weigh more than was desired for a portable device. Wood and plastics would have reduced the weight of the display but did not offer the flexibility to make a custom design that the display most likely would require. As well, wood and plastics may have been more susceptible to twisting and moving while the display is running. Therefore, we focused our research on aluminum as it provides the best mix characteristics to meet the requirements of the chassis. Table 4.6.1 below list several commonly used aluminum pieces, the type of aluminum and their weight. The information in Table 4.6.1 was used to calculate the torque requirements of the POV display during the design phase of the chassis.

Type of Aluminum	Weight
1/4" Plate (Type 6061-T6)	1.764 lbs per square ft.
1/4" x 1/4" Square Tubing (Type 6061 EXT)	0.294 lbs per lineal ft.
1" Solid Rounds (Type 6061 EXT)	0.924 lbs per lineal ft.

Table 4.6.1 Typical Aluminum Pieces and Weight

4.6.2 Chassis Rotating Interface:

The most challenging portion of the chassis design was determining the best solution to rotate the POV display. If we had used wired communications, the center point of rotation would have been left available to allow for the mounting of either the fiber rotary joint or the coaxial rotary joint. Wireless communications did not require the center of rotation to be left available but was not hindered from operation with the center left available. Therefore, we researched options to allow for high speed rotation using some form of a bearing allowing free access to the center of rotation.

To research possible solutions for rotating the POV display with the center of rotation left free, we turned to an online distributor, McMaster-Carr. McMaster-Carr offers a wide range of industrial products at reasonable prices. One such product, and the first feasible solution for rotating the POV display, was a plain bearing turntable. Turntables allow for the rotation of devices mounted on top while working on the device. One particular turntable, part number 8700K1, rotates with the center free and available to be used by the wired communications joint. The 8700K1 turntable can support loads up to 337 pounds, well above the weight requirements of the POV display. As well, there are 8 inner ring mounting holes and 8 outer ring mounting holes providing an adequate surface to mount not only the rotating display but also station supports. However, the turntable has two downsides that make it a less than desirable solution. The first downside was the cost of the turntable at about $215. The second downside of the turntable was there are no posted maximum rotating speeds. This meant that the turntable may be capable of rotating at the required speed of the POV display but no document exists to support it either way.

The second feasible solution from McMaster-Carr, and more promising than the turntable, was an extended-ring steel ball bearing, Type ER. The extended-ring ball bearings have an extended inner ring making installing the bearings easier. Although the extended-ring ball bearing did not have any inner or outer mounting holes, it did have two knurled cup set screws on the extended inner ring that could be used to secure the rotating side of the POV display to the bearing. As well, the bearings have a dynamic load capacity of 2,860 pounds and more depending on the part number selected. All Type ER extended-ring bearings have a max operating speed of 5,000 rpm, far exceeding the requirements of the POV display. As well, the cost of the bearings started at about $30 and go up to about $80 depending on the part number and size. Table 4.6.2 below shows some available extended-ring ball bearings, there size and cost.

Bearing No.	Shaft Dia.	OD	Wd.	Load	Part #	Cost
ER10	5/8"	1.85"	1 7/32"	2,860 lbs	8090T11	$29.67
ER12	3/4"	1.85"	1 7/32"	2,860 lbs	8090T12	$32.61
ER16	1"	2.05"	1 3/8"	3,145 lbs	8090T13	$33.79
ER24	1 1/2"	3.15"	1 15/16"	6,535 lbs	8090T17	$59.40

Table 4.6.2 Extended-Ring Ball Bearings

4.7 Graphical User Interface:

We developed a GUI for use on a PC and researched the possibility of also having one for an android device which would allow us to send either a text message or image to be displayed on the POV display. When sending an image to be displayed the image had to be in the correct resolution and format. If time permitted we would have been able to have the software handle some basic image formatting. First we will discuss the requirements of the application and the method of communication. Last we will consider multiple programming languages that will allow us to create the application effectively and efficiently.

4.7.1 Required Functions:

Part of the research for the GUI was the identification of the requirements. The requirements must be identified before the design can begin. We are going to use a simplified waterfall model for our software development life cycle. We are going to list the requirements, design the software, and then finally implement and test the software. In this section we will focus on the requirements identification only. The design and testing portion will be discussed in the corresponding section later in the paper. The following Figure 4.7.1.a shows a diagram of the simplified waterfall model we researched using for developing the GUI.

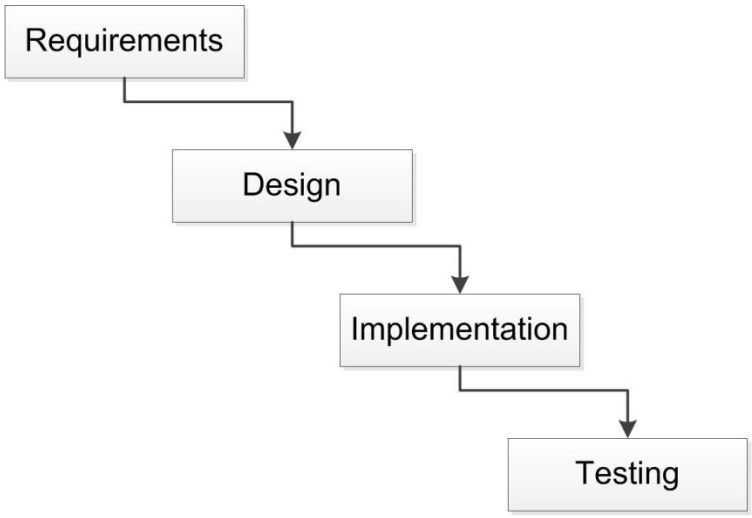


Figure 4.7.1.a Software Development Life Cycle – Waterfall Model

The GUI must provide an easy to understand and user friendly interface. The interface should have very few elements to avoid confusion. The GUI should be operable by anyone and not require any technical knowledge of our display. No training should be necessary, and everything in the GUI should be properly labeled and intuitive. The only functions necessary were to allow the user to enter a text message to display, and to allow the user to select an image file to display. The text field should support multiple lines of text and offer the user multiple color choices. There should also be color options that the user can use to select the color of the entire message and possibility of individual letters. The text message input is discussed in more detail in the design section for the GUI.

The image input will only accept the correct format and resolution images to display. If the selected image file is smaller than the maximum size than it will still be accepted and the image will display centered in the LED display. This can possibly be done by analyzing the size of the input image and calculating where to put the image so that the space to the left and right of the image is equal, and the same for the space above and below the image. If time permitted we may have been able to further increase the functionality of our software to properly scale images that are too large to be fully displayed. This would be a simple algorithm that simply picks and chooses every other pixel to display or something similar. The image input, like the text input, is also discussed in more detail in the design section for the GUI. A simple use case diagram is shown in Figure 4.7.1.b to highlight the main requirements of the GUI.

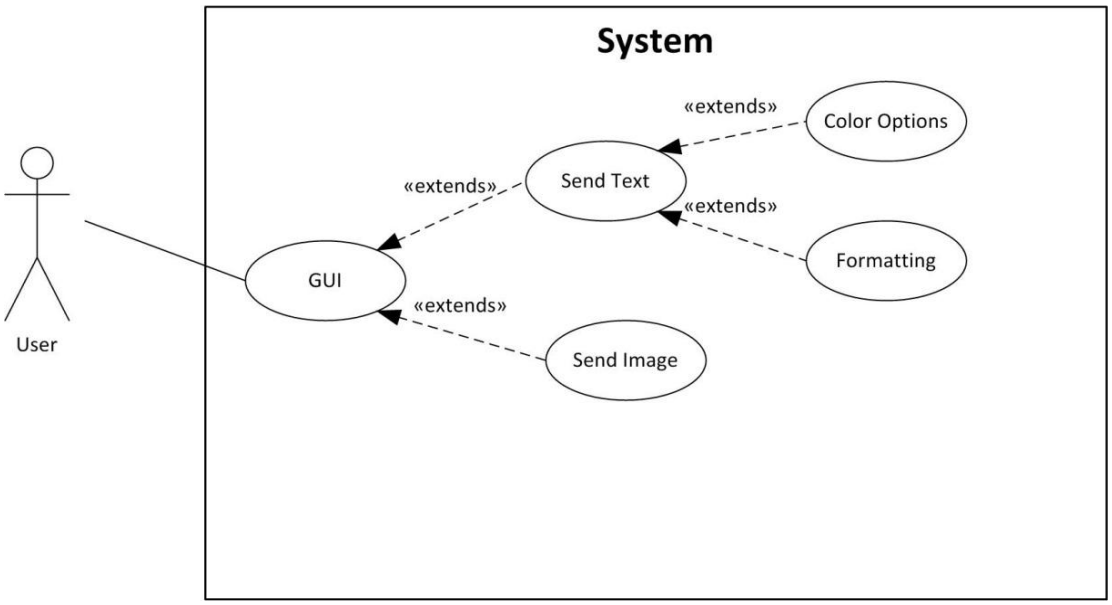


Figure 4.7.1.b Use case diagram for GUI

4.7.2 Programming Language:

We must choose a programming language to build the communication application. We should consider multiple programming languages and choose the best one suited to our task and also choose one which we are familiar with. This application will have a user friendly GUI and allow simple serial communications. All of the requirements listed above must be considered when choosing the appropriate language. In order to efficiently create a GUI the language will be required to have built in libraries that support agile GUI development. The IDE should provide tools that will allow most of the development to focus on coding the core functions of the application and not on the GUI's appearance and layout. We considered C++, Visual Basic, and Java.

The C++ programming language is something that we are all familiar with. C programming is where we started our programming education and is where C++ is derived. This is an object oriented language with wide support and plenty of documentation. We had no experience creating a GUI in C++ so further research was needed in order to determine whether or not C++ would be worth considering for the user friendly application that we were striving for. After some research it was found that there are GUI libraries available to assist in developing a GUI in C++ but there are multiple GUI libraries to choose from. Multiple choices for a GUI library further complicated things since further research would have be conducted in order to determine which would be the best library to use. There did not seem to be a visual GUI editor for C++ available, and it seems that for most GUI applications, C++ is not the language of choice. We believed it was safe to say that C++ should not be the language we use to build our communications GUI application. Even though C++ was not the best choice for developing the

GUI, C++ may be better suited to interface with the hardware for USB communications.

Visual Basic is a programming language that is specifically designed to allow agile development of GUI applications. The IDE, Visual Studio, has a visual GUI editor for programs using Visual Basic. It is very easy to drag and drop text boxes, labels, buttons, etc. onto each form of the application. Programming the functions of the elements placed in the form becomes as simple as double clicking that item and the IDE will jump to the code that controls it. This could have been a good choice for quickly developing a GUI based application, but only one member in our group is familiar with this programming language which may not be adequate. It would most likely be more efficient to have more than one group member to assist in the development of this application and having to learn a new language may decrease productivity.

The Java programming language was the best language for both developing a GUI for a PC and for an android device if we had enough time to implement it. The Netbeans IDE has a built in visual GUI editor for Java which greatly simplifies GUI design and implementation. The Netbeans GUI editor allowed us to develop a GUI application in a similar way that Visual Basic would have allowed us to. Java is a high level object oriented language and has many built in classes to support agile development. There are also many open source Java libraries available for download to provide further features and functionality. We were also already familiar with the Java programming language. Java was the obvious choice for a high level programming language that we already possess enough knowledge to code in and had enough built in features and tools to allow us to rapidly build the tools we need for our project. The only drawback to using Java was the limited functionality when it comes to accessing connected hardware. This could have been an issue for us since we were planning on using communications through USB, or a connected wireless adapter. In order for us to implement USB communications using Java we would have had to find a suitable driver for our desired operating system and find a Java library that is capable of interfacing with that driver. Since we found such a driver and software library combination it was safe to say that we would be using java for our GUI application development.

4.7.2.1 Image Format Conversion and Resizing:

Our GUI would allow users to select an image to be displayed and load it onto the rotating processor. Without being too restrictive on the user, we wanted our program to accept virtually any image file format that is common. The primary information we needed from the image are the RGB values contained within it so that we can format an output file that our device will understand. Each image format is different and must be decoded via some method in order for us to obtain this data.

In order to handle the various image types that the user could select, we had to determine that functions within the java library would be able to handle this. Several java classes would be used in order to do this, ImageIO, BufferedImage, and indirectly ImageReader. Using the java ImageIO class, we open an image file by using ImageIO.read() and supply an argument of a name/path. The ImageIO class on its own will then search for an ImageReader that claims to be able to read that type of image, and decode it. ImageIO.read() will return a java BufferedImage, from which we can easily obtain the RGB values by calling the function BufferedImage.getRGB() and supplying an x and y coordinate. Using this library the user will not need to be concerned with the image file format, and we will not need to code the tedious functions that would be required to decode the many image format possibilities.

Another concern involved image resizing. Using the simplest solution we would require that the user resize the image manually using image editing programs before trying to upload it. However, because the BufferedImage class will tell us the size of the image that has been selected by called BufferedImage.getHeight() and BufferedImage.getWidth(), we could handle the scenarios where the image is too small or too large in specific ways. In the case where it's too large, we could simply truncate the image, or offer various methods of cropping the image. If the image is too small, it could be padded and centered, depending on user specifications.

4.7.3 GUI Communications to Microcontroller:

The program will have to communicate with the microcontroller in order to get the correct image to display on the LED array. The communication should either be the wireless communication that we choose to use (WiFi or Bluetooth) or it should be through a USB cable. The preferred method of communication would be through WiFi or Bluetooth since this is also supported by android devices and would allow us to send images to be displayed on the LED's with our mobile phones. It would be very convenient as well if we did not have to connect a laptop to our display with any wires. If we use WiFi we will have to use the ad-hoc mode of networking since it would not be very practical for this project to require a wireless router as well. If Bluetooth had been used then Bluetooth would be the communication method when using the mobile application, but when using a PC a cable will be required. This is because most PC's do not have Bluetooth built in so it would be counter-productive to develop a PC application that utilizes Bluetooth communications. If we have additional time we may be able to include Bluetooth communication support for the PC application as well.

4.7.3.1 Serial Communication Software Library:

At first it seemed that Java would not have a way to access USB devices. There are no built in methods to allow Java hardware access for serial communications.

There is a library created by Sun which allows serial communications, but it is only supported on the Linux operating system. Further research allowed us to find a community created Java library called RxTx which supports serial communications on multiple platforms including Windows. In order for the RxTx library to work however, we need to find a valid USB driver that will allow windows to recognize the connected device for serial communications. If the Digilent Atlys board does not include USB drivers for this purpose, we have found a driver download as well. The driver is for the Universal Asynchronous Receiver/Transmitter or UART chip that is on the Atlys board. The UART chip allows the USB to function as a serial communication interface. With the proper drivers installed communicating with the Atlys board using USB should be no different than using the older RS-232 method. Once the RxTx Library is properly added to the JDK we can than import the methods and use them for our project. There are methods in the library to handle listing the available serial communication ports. The library will then allow us to choose an available port and use it for communication. Input and output streams will need to be declared in order to send and receive data. Overall the library seems to make it rather easy to send and receive serial communications. More details on how the serial programming works are provided in the design section.

4.8 Microcontrollers:

There are many microcontrollers available with many different feature sets. This research will focus on the different microcontrollers available and which ones we should use in our POV display. We are going to need two microcontrollers, one is going to have to deal with the video input and remain stationary in order to be able to plug in a device such as a laptop or DVD player. The other microcontroller will rotate along with the LED's and provide all of the information to the LED controllers so that they can send the PWM signals to each LED.

The stationary microcontroller is most likely going to be an FPGA since this has been the only solution we have been able to find regarding a board that accepts HDMI input. The cost of the FPGA is going to be considerable since it is a board designed to take HDMI input and possibly process that video signal. HDMI is most likely a high definition signal and therefore would require a powerful board in order to effectively process that amount of data efficiently. We all have academic experience programming an FPGA using Verilog so our biggest challenge is going to be figuring out how to process the video input.

Our rotating microcontroller will be considerably cheaper; this microcontroller does not have any special requirements other than having enough outputs to service the latches and LED's. For our rotating microcontroller we will focus on a combination of cost, and ease of use. Ease of use is a factor because we do not have the same experience working with microcontrollers that we do with FPGA devices. We would want a microcontroller that will be easy to learn and easy to

work with. Because of the large number of LED's we plan on using, we may also have to consider the number of outputs that each microcontroller is able to support.

4.8.1 Digilent Atlys (Stationary FPGA):

The Xilinx Spartan 6 FPGA available on the Digilent Atlys board. The Atlys board has onboard HDMI input. The main reason for choosing this board is for the HDMI input which will allow us to receive a video input in order to display it on the LED array. The HDMI input on the Atlys board will automatically take care of the TMDS decoding for us. We will have to figure out how to represent the video data in such a way that our secondary microcontroller will be able to split up the data and send it to the proper latches to control the LED's. The Atlys board does not seem to have built in pins in order to connect directly to the FPGA's I/O's. There is a VMOD peripheral that would take care of this problem and allow us to connect wires to any of the I/O's, but this will increase the cost of an already expensive board.

4.8.2 TI Launchpad (Rotating Microcontroller):

TI offers a very cheap microcontroller that we may be able to take advantage of. The MSP-EXP430G2 or Launchpad is a development board for the MSP430G2XXX series of microcontrollers. The board only costs $4.30 and includes two MSP430 microcontrollers, and a USB cable. The board will allow us to program the microcontrollers using the USB interface. This microcontroller has very widespread support, documentation, and example projects. Possible limitations include the limited number of I/O ports, 2KB of program memory, and 128B of SRAM. The microcontrollers that come with the Launchpad board only have 10 available I/O pins. If we were to purchase a separate higher end compatible microcontroller we can increase the number of outputs to 16. The low number of I/O pins may require us to use more than 1 microcontroller, but as stated earlier the Launchpad comes with 2 of them already, and the higher end MSP430 controllers with 16 I/O ports are less than $2 each.

4.8.3 Arduino Uno REV 3 (Rotating Microcontroller):

The Arduino Uno board is another alternative to the TI Launchpad. This board comes with an ATmega328 microcontroller on it. The Arduino Uno board takes care of the USB interfacing and programming. This board is more expensive than the TI Launchpad at $35. The higher price may be justified by the increased performance and memory of the microcontroller included. The ATmega328 has 31.5KB available for program memory (0.5KB is used by the boot loader), 2KB of SRAM, and 1KB of EEPROM. The ATmega328 also has 14 I/O pins, 6 of which can be used for PWM. Another feature that may be useful is I^2C support. I^2C will allow us to have serial communications to possibly another IC that will expand the number of I/O's available to us. This board is also widely available and

supported. There are many hobbyist projects with open source documentation and examples for helping us get familiar with programming this board. The additional program memory and RAM may not be necessary, but the additional outputs that this board provides may make a difference. Another thing to consider is the programming language. The Arduino Uno board allows the use of a C-like language to program the ATmega328 microprocessor. If we were to use the TI Launchpad we would have to use assembly. It may be easier and more time efficient to use the Arduino Uno board.

4.8.4 Digilent Cerebot MX7cK (Rotating Microcontroller):

The Digilent Cerebot MX7cK development board has a 32-bit PIC32 microprocessor. This is an expensive choice for the rotating microcontroller but it has a much higher clock speed of 80MHz. This higher clock speed may be required for our project if we are to process full motion video in real time. This board also has a built in Ethernet interface which we can possibly use for communications between the stationary FPGA and the rotating microcontroller. Programming the Cerebot board should be similar to programming the Arduino. Digilent advertises the fact that Arduino projects and code should be compatible with their Cerebot boards. Although the Cerebot board seems to outperform the other boards in every category it is much more expensive at $99. It may also be necessary for us to buy additional Pmod accessories in order to access some of the I/O pins further increasing the cost. We hope to find a microcontroller for the rotating part of our project that can keep costs to a minimum while having the required performance needed for a live video feed. The following Table 4.8.4 shows a simple comparison between all of the previously discussed microcontrollers being considered for the rotating part of our project.

Microcontroller Comparison				
	Digilent Atlys	TI Launchpad	Arduino Uno	Cerebot MX7cK
Program Memory	64MB	2KB	31.5KB	512KB
SRAM	128MB	128B	2KB	128KB
EEPROM	0B	0B	1KB	0B
I/O	48	10	14	85
Frequency	500MHz	16MHz	16MHz	80MHz
Programming	Verilog HDL	Assembly	High-level	High-level
Cost	$199	$4.30	$35	$99

Table 4.8.4 Microcontroller Comparison

4.8.5 Additional Microcontroller Concerns:

This project is highly dependent on sponsorship funding in order to include all of our intended features. Video input is not normally a feature found in a POV display. Our research has indicated that the reason for this may be the costs involved. The Atlys board described above is absolutely necessary for us to consider live video input for our POV display but there are other considerations that must be addressed as well. At first we decided that our secondary microcontroller which will spin along with all of the LED's need not be as complex and expensive as the Atlys board. After much research it became apparent that although we do not need an HDMI input on the rotating board, we do need a substantial clock frequency in order to properly sample the large amounts of data required for a live video feed. In previous sections we have mentioned possible data rates that would be required to be sent through communications between the two microcontroller boards. Regardless of the communication method we choose, we must not consider if these microcontrollers can properly sample the data at the required speeds to display a live video feed. Table 4.8.5 shows possible resolutions we may consider for our display and the required data bit rate necessary. The values in the table assume that the video data is not compressed.

Resolution	Data Rate
640x480	73.728 Mbit/s
320x240	18.432 Mbit/s
160x120	4.608 Mbit/s
80x60	1.152 Mbit/s
40x30	0.288 Mbit/s

Table 4.8.5 Possible Resolutions and Corresponding Data Rates

The data rate values in table 4.8.5.a are calculated using the simple formula HP × VP × BPP × FPS where HP is Horizontal Pixels, VP is Vertical Pixels, BPP its Bits Per Pixel, and FPS is Frames Per Second. According to the data sheet for the ATmega328 microcontroller, the maximum data rate that the microcontroller is capable of sampling with its 16MHz crystal is 2Mbit/s. This means that the Arduino Uno and TI Launchpad development boards would only be able to support a display with a resolution up to 80x60. The calculation to determine the maximum data rate given the frequency of the microcontroller is given in the data sheet for the ATmega328. The formula is shown next for reference.

$$BAUD = \frac{f_{osc}}{8}$$

Although formula above came from the ATmega328 data sheet it can still be used as an approximation for the capabilities of the other processors too. The baud rate for the ATmega328 is measured in bits per second which is why the maximum data rate for the ATmega328 mentioned previously was in the units of

Mbit/s. Using formula 3.8.4-1 for an 80MHz clock frequency it can be said that the maximum practical data rate that the Cerebot MX7cK microcontroller should be able to effectively sample should be about 10Mbit/s. The higher clock speed allows for a much higher data rate. The maximum resolution that we are considering that can be implemented with 10Mbit/s maximum data rates is 160x120. This leads us to the conclusion that if we intend to implement any resolution higher that 160x120 then we will have to use two of the Digilent Atlys boards, one which will remain stationary to receive the video input, and the second one to spin with the LED's and send all of the data to the LED controllers. Only the 500MHz clock on the Atlys board would be able to effectively sample the high amounts of data associated with uncompressed high resolution video.

5 POV Design:

The persistence of Vision device required two major areas of design to successfully create. The first section was our hardware design, which would make up the Chassis, motor control circuit, motor, power supply, and display alignment sensor. Each of these sections that we designed has subjections within them that also needed to be designed for the project. The second major section was the software design of the project. This section encompassed the Wi-Fi server and connection processes, the image processing for both the text and RGB arrays, the data structures for all the code, the handling of all the hardware inputs such as the display alignment sensor's output, and finally the GUI that was needed to integrate many of these elements into a more user friendly format. The following subsections will discuss the overall design of these sections, including design elements that were thrown away during the testing phase. Such elements that were thrown away for better methods will be indicated both within this section and the testing sections as well as why the better methods were chosen. Figure 5 is a photo graphic image of the final design of the POV display that we will be discussing in this section. Within the image you can see all the elements of the project, hardware-wise that went into the construction of the final product.

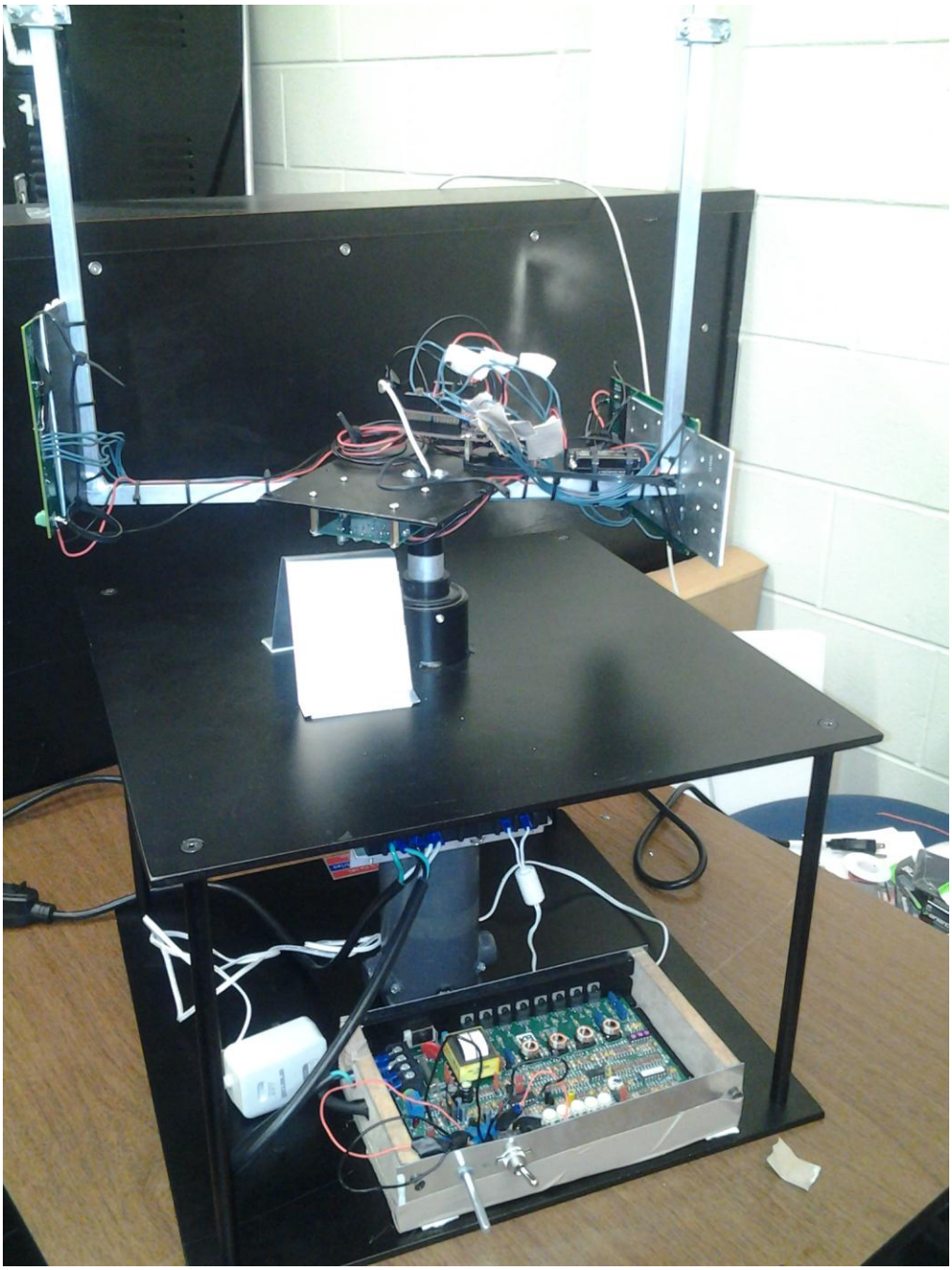


Figure 5: POV display

5.1 Hardware Design:

The hardware of this device is broken into four major sections. The first section is the stationary control section which consists entirely of the AC input, the KBRG-212D motor control chip, Dayton 9FHD7 DC motor, the computer that will connect via Wi-Fi to the rotating microcontroller, and the AC to DC variable

adapter. The second section is the power transfer section that consists entirely of the slip ring and wires that connect from it to the rotating side. The third section, the rotating control side consists of the PIC32 microcontroller, RGB and Text LED arrays, and the 9V battery that powers the microcontroller. The final portion of the hardware design is the Display Alignment Sensor which consists of the IR sending and receiving circuit and the rotating apparatus it is attached to. Figure 5.1 gives a good visual representation of the flow of the hardware and how they will be connected together.

Each of these sections of the device has a variety of different hardware components needed in order to achieve the ultimate goal of creating this persistence of vision device. The following sections will discuss more thoroughly our final decisions on the hardware design of each portion of this device and the actual hardware design themselves. This will include the specific components we used to implement each of these designs. Also within this section will be a layout of the structural design of the chassis which will house all of the electrical hardware for this device.

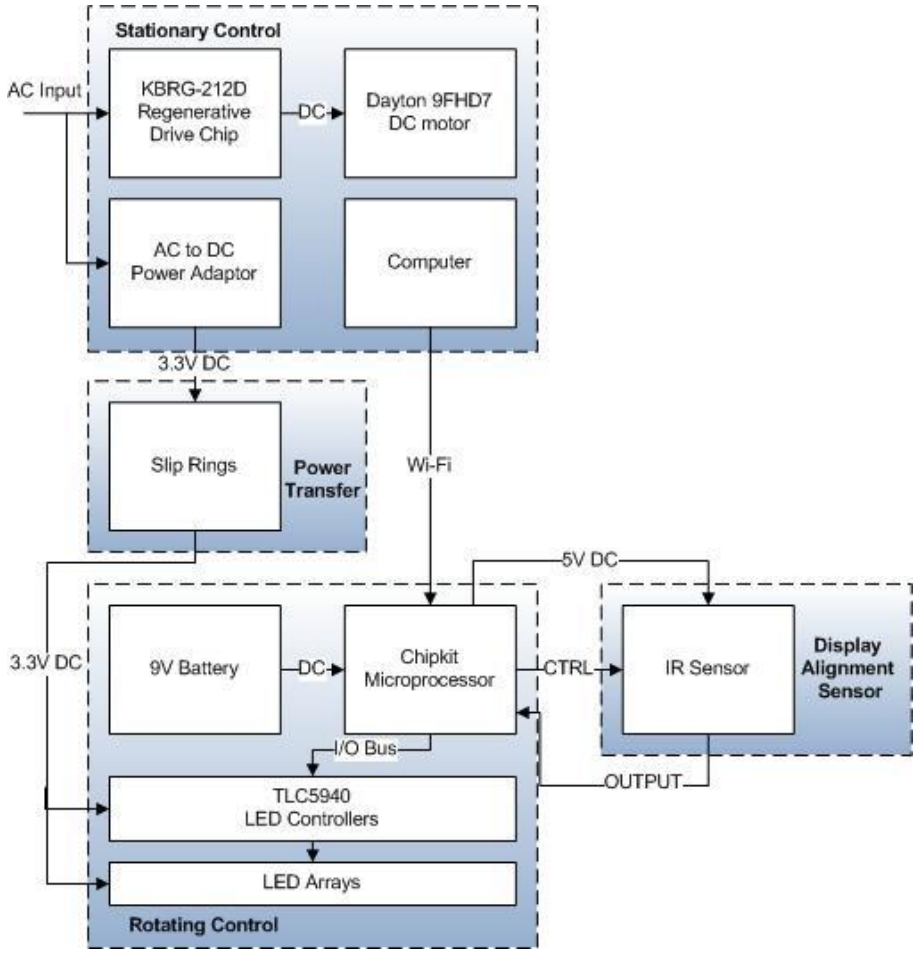


Figure 5.1 Hardware Flow Chart

5.1.1 Chassis Hardware Design:

As discussed during the research section for the chassis, we constructed the chassis from aluminum using a combination of aluminum plate, square tubing and solid round rods.

5.1.1.1 Chassis Dimensions:

Before we were able to finalize our chassis design, some basic dimension requirements had to be identified. The first, and most critical dimension requirement was for the physical size of the LED array. We then needed to determine the size of the chassis base and the space required to mount the motor.

5.1.1.2 Dimensions of LED Array:

We used the Multicomp's SMD Super Bright LED, part number OVS-3309. The LED has a vertical dimension of 2.8mm and a horizontal dimension of 3.2mm. The horizontal dimension was required to properly mount the LEDs on a printed circuit board but are not a dimension required or even necessary to determine the size of the LED array and is therefore ignored for the chassis design. We mounted the LEDs with a spacing of 1mm between each LED, allowing us to determine that the spacing between each LED, as measured from center to center, is 2.85mm. Therefore, the maximum allowed total vertical length of the LED array is 2.85mm x 128 LEDs or 365mm. Converting the total vertical length to inches gives a final dimension of approximately 15 inches to which we constructed the LED array to.

Next, we needed to determine the diameter of the LED array. When the POV display is running, we can simplify the LED array to cylinder. As well, since we wanted the horizontal spacing of the LEDs to be the same as the vertical spacing of the LEDs, we used the known pixel ratio of 128 to 384 to determine the length required. Dividing 384 by 128 gives the ratio of horizontal pixels to vertical pixels, which equals 3 or 3:1. For accuracy, we initially calculated the required horizontal length in millimeters. Taking the ratio of 3:1 and multiplying by the known vertical length of 365mm we got a required horizontal length of 121.67mm. Since the LED array can be simplified to a cylinder, we were able to find the circumference of the LED array. Using the formula of $C = 2\pi r$, we calculated the radius of the LED array which is equal to 119mm. Converting the radius to inches, we got a final dimension of approximately 6 inches.

5.1.1.3 Dimensions of Chassis Base:

Now that we knew the size of the LED array when the POV display is spinning, we were able to determine an appropriate size for the chassis base. The chassis base serves as two purposes for the POV display. The first and most obvious purpose is to provide an adequate foundation for the display. The second purpose and more important than the first, is to provide a visually marker to signify where the limit is to approach the display while it is running. This is especially important if, for example, the display is running but not displaying an image. Therefore, we constructed the base of the POV display to extend just past the fast rotating LED array. Since we knew the radius of the spinning LED array, we determined that the base will needed to be at least a 6 inch by 6 inch square. We then took into account the extend size of the secondary test LED array which will extend approximately 2 inches past the primary image LED array. Adding an extra inch between the spinning array and the base gave us a final dimension of 18 inches x 18 inches. Since constructed the chassis out of 1/4 inch aluminum plate, a 18 inch square base provided plenty of weight and strength to fully support the POV display while it is running.

The last dimension that was required before we were able to determine the final design of the chassis was the physical size of the motor. Size the motor is mounted on the base, it determined the height of the base. The overall length of the motor was 8.8 inches. To allow room to mount and secure the motor to the base, we designed the base with an internal height of 12 inches. Taking into account the thickness of the aluminum plate, the total height for the base is 12.5 inches. Therefore, the total size of the chassis base is 18 inches x 18 inches x 12.5 inches.

5.1.1.4 Chassis Assembly:

Now that we knew the required dimensions of the chassis we began to design the assembly of the chassis. A complete chassis model can be seen in Figure 5.1.1.4c below.

The first step to putting together our final design of the chassis was to determine which rotating interface we will use to transfer the rotating power of the motor to the LED array. As discussed in our research, we had two options. The first option of the turntable provided the easiest solution for mounting the LED array and base to the rotating interface. However, due to cost and no defined specification of the maximum rotating speed, we choose to use the extended-ring bearing. In order to provide the most space for feeding the power supply cable cable through the rotating interface, we choose to use the extended-ring bearing with a one inch shaft diameter, part number 8090T13. We then secured the bearing to the base of the chassis by welding the extended-ring portion of the bearing to the top of the base. Although welding does not allow for easy modifications, it provided a strong and secure method that of holding the bearing in place during operation of the POV display. In order to secure the LED array to the bearing, we inserted a aluminum pipe through the inner ring of the bearing. The pipe was then secured

to the bearing using the two set screws that come installed on the bearing. This allowed for the POV display to be easily disassembled when moving between locations. Using this designed allowed us to use the pipe to mount the slip ring for electrical power transfer. Lastly, we notched to top of the pipe to allow the LED array support bar to be secured to the pipe. Figure 5.1.1.4a below shows a model of the bearing and pipe assembly. The chassis base and LED array support frame are removed for clarity.

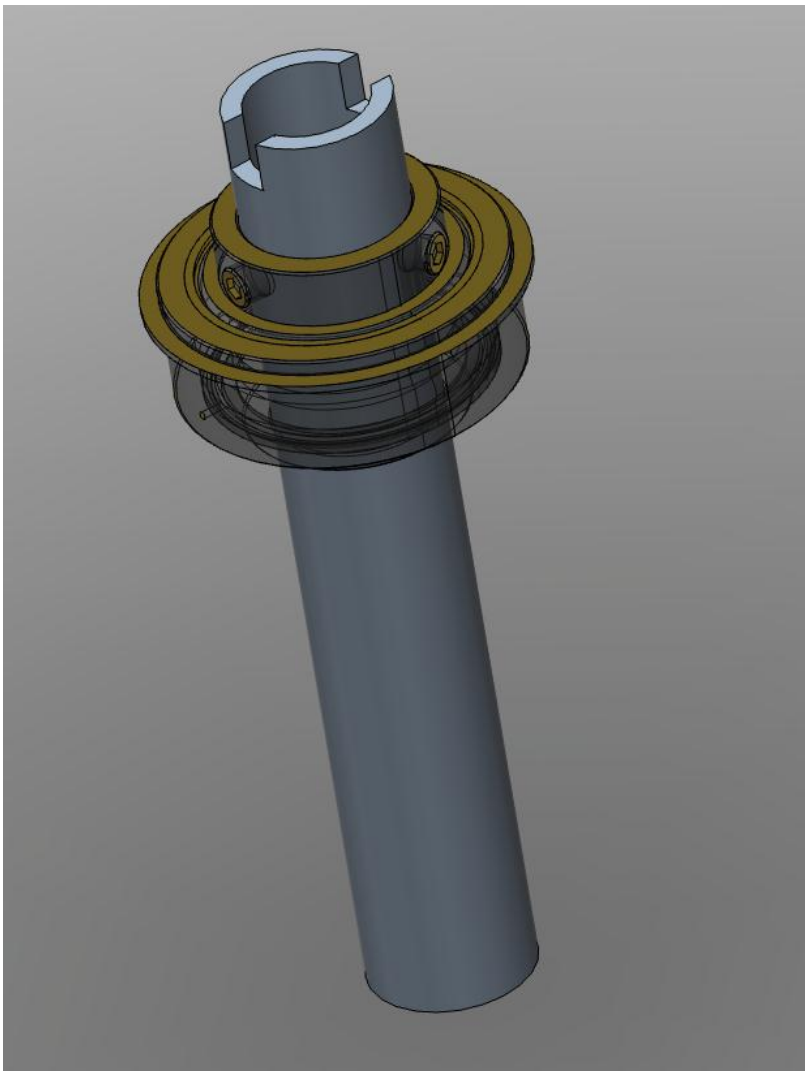


Figure 5.1.1.4a Bearing Assembly

Next we designed the chassis base. As discussed, the chassis base needed to be 19 inches x 19 inches x 12.5 inches. The base was constructed out of two 1/4 inch pieces of aluminum plate creating a top plate and a bottom plate. The two plates were secured together by four solid aluminum rods, one in each corner, cut to 12 inches lengths. The plates had counter sunk holes drilled in each corner, three inches from each side. The rods were drilled and tapped in the

center to accept a 1/4-20 screw. The rods and plates were assembled by screwing the plates and rods together. The counter sunk holes on the plates allowed for the screws to be flush with the surface. In order to mount the bearing, a hole was cut out from the center of the top plate. The diameter of the hole was made larger than the diameter of the inner ring of the bearing but smaller than the extended flange of the bearing. This allowed for the bearing to rest on the top plate and provided a surface for the bearing to be welded to the plate. Figure 5.1.1.4b below shows a model of the base assembly, including the cut out on the top plate for the bearing.

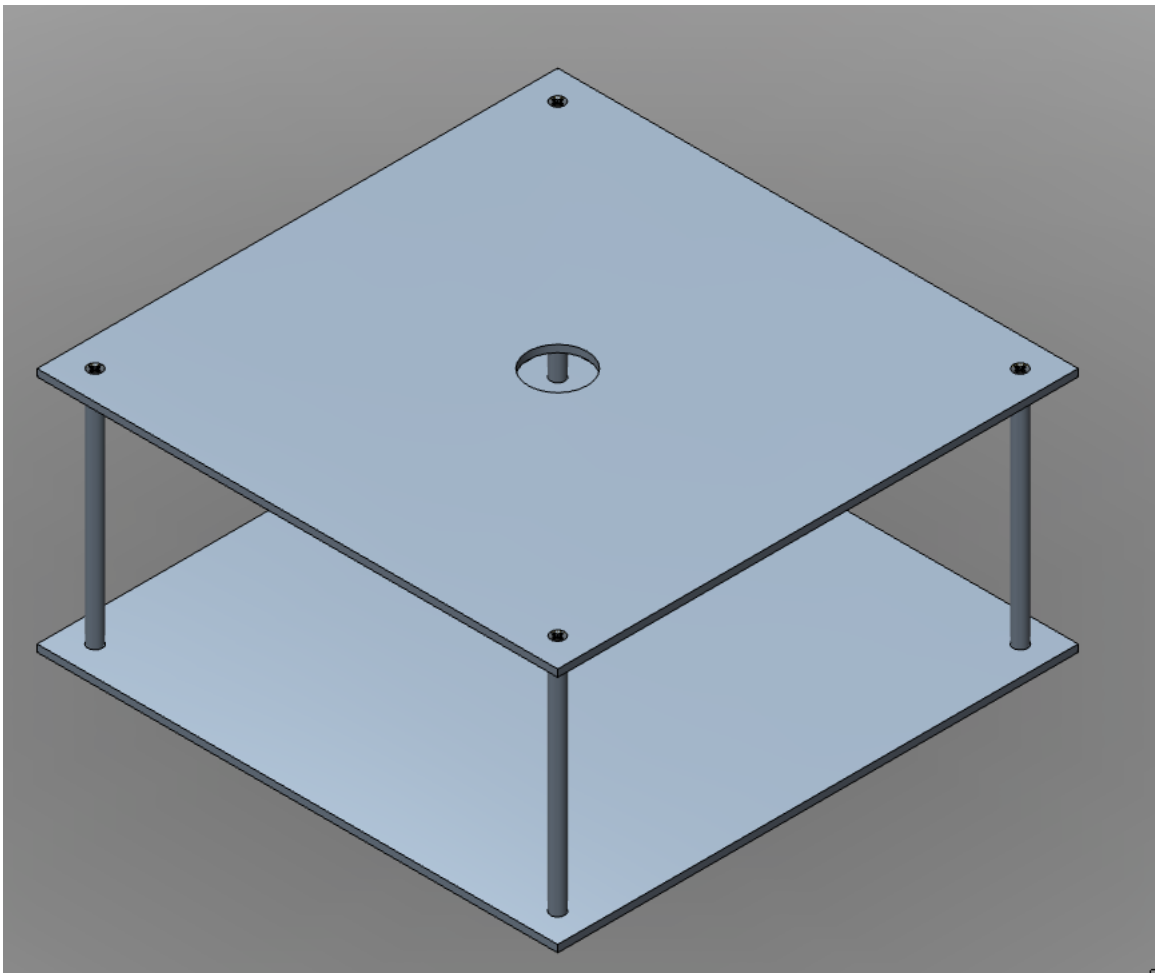


Figure 5.1.1.4b Chassis Base Assembly

The LED array support frame was constructed from 1/8 inch square tubing. From the calculations for dimension requirements of the LED array, we knew that the horizontal LED array support bar, the piece that will be connected to the notched pipe, needs to be 14 inches long. This dimension needed to be exact as it will directly affect the aspect ratio of the display. At each end of the horizontal LED array support bar, vertical LED array support bars were welded. We know from

the LED array dimension requirements that the vertical support bars must be at least 15 inches long.

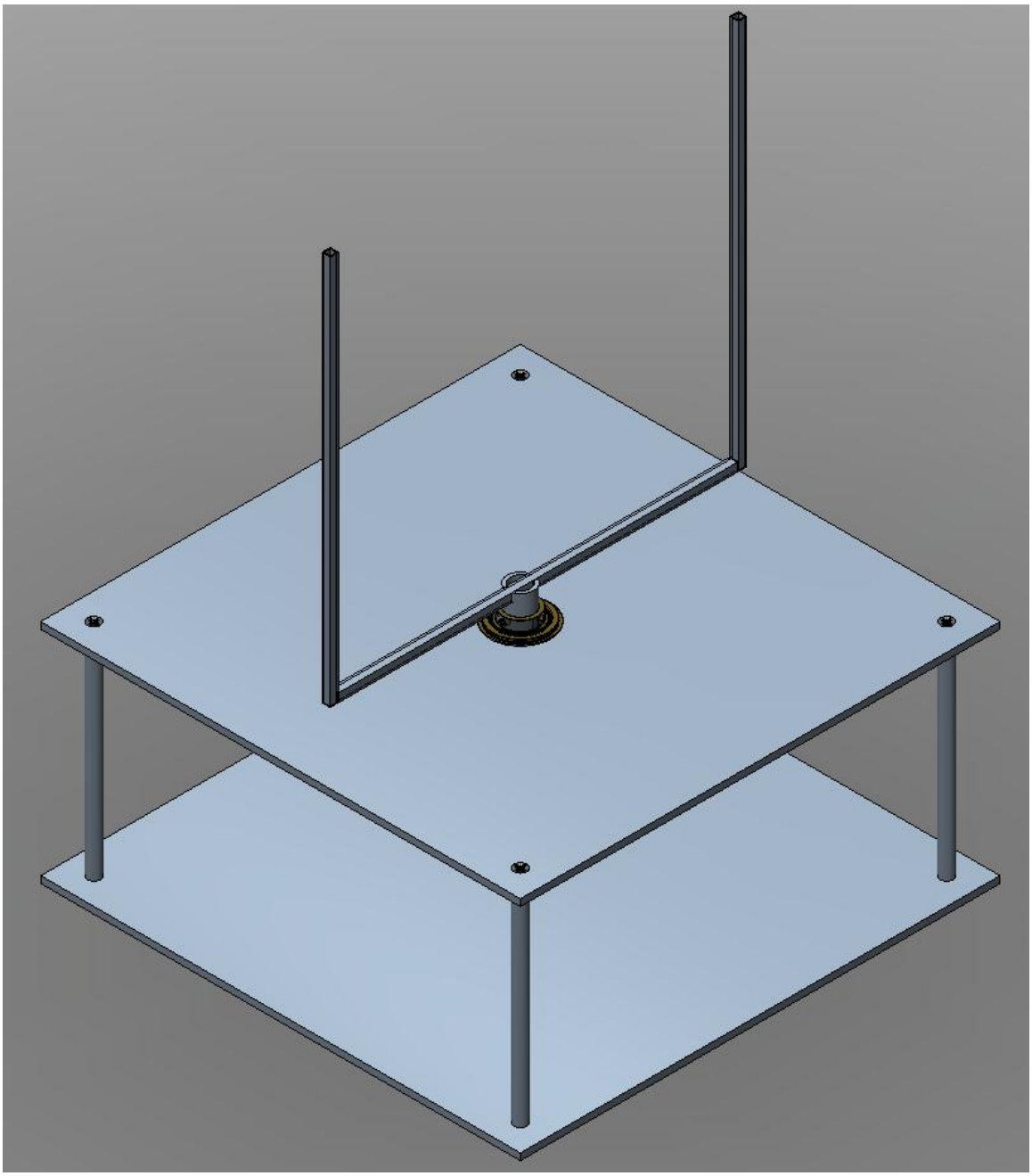


Figure 5.1.1.4c Chassis Base Assembly

5.1.1.5 Motor Interface:

To transfer the power of the motor to the LED array, we mounted the support pipe directly over the motor shaft. Then using a set screw we secured the pipe to

the shaft. This allowed for the motor to directly drive the LED array and simplified to the fabrication process.

5.1.1.6 Chassis Torque Calculations:

Now that we have finalized our design for the POV display chassis, we needed to estimate the torque requirements. To simplify the torque calculations we used the simplified LED frame shown below in Figure 5.1.1.6a.

Figure 5.1.1.6a: Simplified LED Support Frame for Torque Calculations

We then used the values and equations shown below to estimate the torque requirements of the POV display when operating at 15 RPMs. The mass M1 and M2 were derived from the linear weight per foot of the aluminum square tubing used to fabricate the LED support frame.

$$M1\ (mass\ of\ primary\ LED\ array) = 0.15\ kg$$
$$M2\ (mass\ of\ secondary\ LED\ array) = 0.15\ kg$$
$$R1 = 0.35448\ m\ and\ \ 0.0875\ kg$$
$$R2 = 0.40752\ m\ and\ 0.1125\ kg$$

$$I_{M1} = M1\ x\ R1^2 = (0.15\ kg)x(0.35448)^2 = 0.0188\ kg \cdot m^2$$
$$I_{M2} = M2\ x\ R2^2 = (0.15\ kg)x(0.40752)^2 = 0.0249\ kg \cdot m^2$$
$$I_{R1} = {}^1/_3\ x\ MR1\ x\ R1^2 = (0.333)x(0.0875kg)x(0.35448)^2 = 0.00366\ kg \cdot m^2$$
$$I_{R2} = {}^1/_3\ x\ MR2\ x\ R2^2 = (0.333)x(0.1125kg)x(0.40752)^2 = 0.00622\ kg \cdot m^2$$

$$\sum I = I_{M1} + I_{M2} + I_{R1} + I_{R2} = 0.05358\ kg\ \cdot\ m^2$$

$$\sum I = 0.05358\ kg\ \cdot\ m^2$$

$$\alpha = \frac{[(15)x(2\pi)^2]}{[(2)x(1)x(2\pi)^2]} = 7.5$$

$$\boldsymbol{T = \sum I\ x\ \alpha = 0.402\ N \cdot m}$$

Our final calculated torque requirements came out to be 0.402 N•m which is under the maximum torque of 0.49 N•m provided by the motor.

5.1.2 LED Array Hardware Design:

As discussed during the research section for the LED array, we had two options for controlling the LEDs. One option was to use a latch control system and the second option was to use pulse width modulation LED controllers manufactured by Texas Instruments. Due to the easy integration of the LED controllers into the microcontroller outputs and the built-in latch control we choose to control the LED array using the PWM controllers. In particular, we used the TLC5940 16 channel LED driver. The reason for choosing to use the TLC5940 is due to its high data transfer rate of 30 MHz as well as allowing us to individually control each LED. Additionally, the TLC5940 controllers allowed us to wire the controllers together to cascade the serial communications required to write to each controller.

5.1.2.1 TLC5940 Pin Out and Wiring:

After selecting which method we wanted to use for controlling the LEDs, the next step was for us to determine the pin out and wiring of the LED controllers. The pin out information for a TLC5940 in a NT case can be seen in Figure 5.1.2.1a below. Table 5.1.2.1 below shows all pins and their functions.

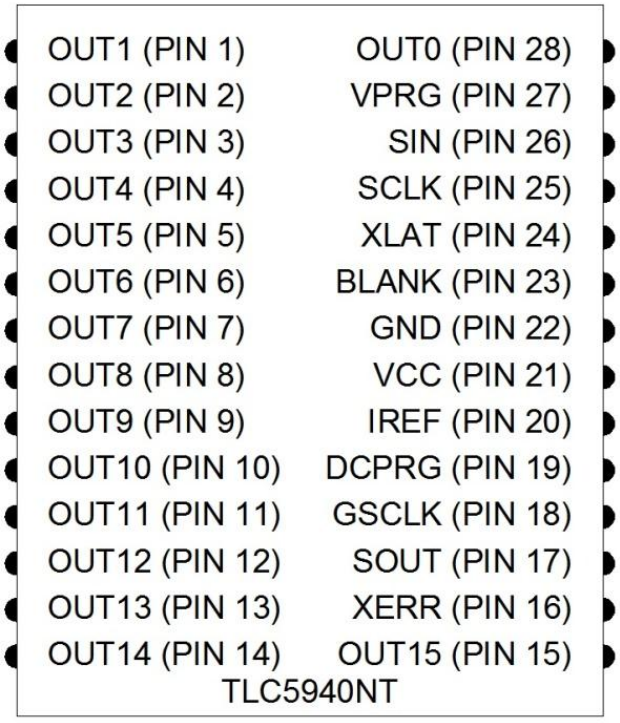


Figure 5.1.2.1a TLC5940 LED Controller Pin Out

Pin #	Name	Description
1	Out 1	Current Output to LED

2	Out 2	Current Output to LED
3	Out 3	Current Output to LED
4	Out 4	Current Output to LED
5	Out 5	Current Output to LED
6	Out 6	Current Output to LED
7	Out 7	Current Output to LED
8	Out 8	Current Output to LED
9	Out 9	Current Output to LED
10	Out 10	Current Output to LED
11	Out 11	Current Output to LED
12	Out 12	Current Output to LED
13	Out 13	Current Output to LED
14	Out 14	Current Output to LED
15	Out 16	Current Output to LED
16	XERR	Error Output Low = Error
17	SOUT	Serial Data Output
18	GSCLK	Reference Clock for PWM Control
19	DCPRG	Dot Correction Switch Low = DC Connected to EEPROM High = DC Connection to DC Register
20	IREF	Reference Current Terminal
21	VCC	Power Input Terminal
22	GND	Ground
23	BLANK	Turns all outputs on or off Low = Outputs are controlled by PWM High = All outputs forced off, GSCLK is reset
24	XLAT	Latch Signal Low = Data in registers held constant High = writes from shift register to DC or GS register
25	SCLK	Serial Data Shift Clock
26	SIN	Serial Data Input
27	VPRG	Input Pin GND = Controller is in GS Mode VCC = Controller is in DC Mode V(vprg) = DC register data can be programmed into DC EEPROM
28	Out 0	Current Output to LED

Table 5.1.2.1 TLC5940 LED Controller Pin Information

To cascade the controllers together requires the SIN and SOUT pins to be wired together in series. Meaning the SOUT from one controller was wired to the SIN

pin on another controller. The wiring required for the controllers can be seen in Figure 5.2.1.b.

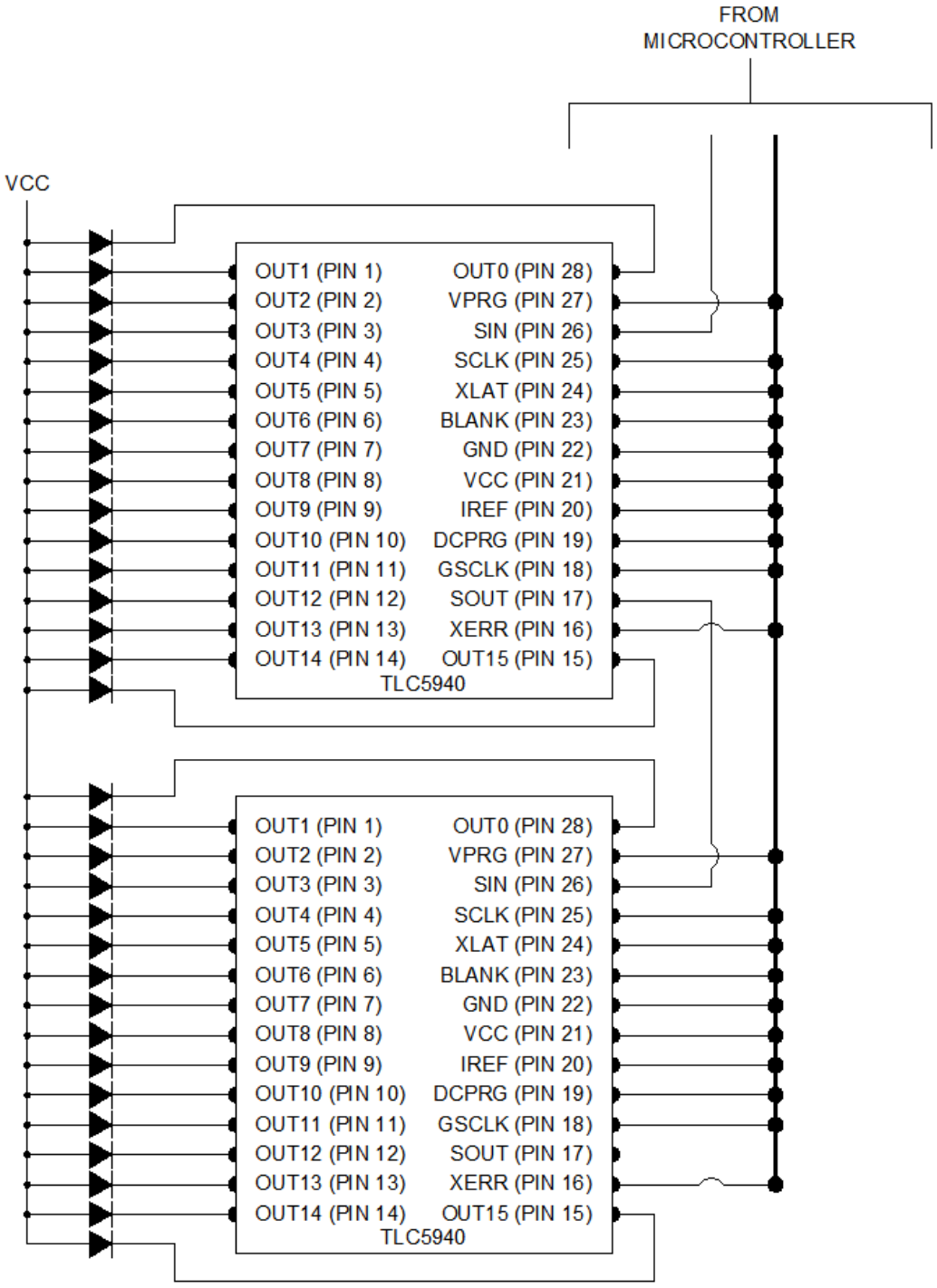


Figure 5.1.2.1b LED Controller Wiring

5.1.2.2 LED Array for Text Display:

The design for the LED array required for displaying text used the same LED controller but we only used mono-color LEDs. The text display contains 16 LEDs so only one controller was required. The wiring of the controller and LEDs was similar to Figures 5.2.1.b.

5.1.3 Motor Hardware Design:

Our specific motor design encompasses two major sections a control elements and control inputs section. The control elements section makes up the things that need to be controlled: the motor control chip and the motor, while the control inputs section encompasses the input signal controlling the speed and whether the motor is enabled or disabled.

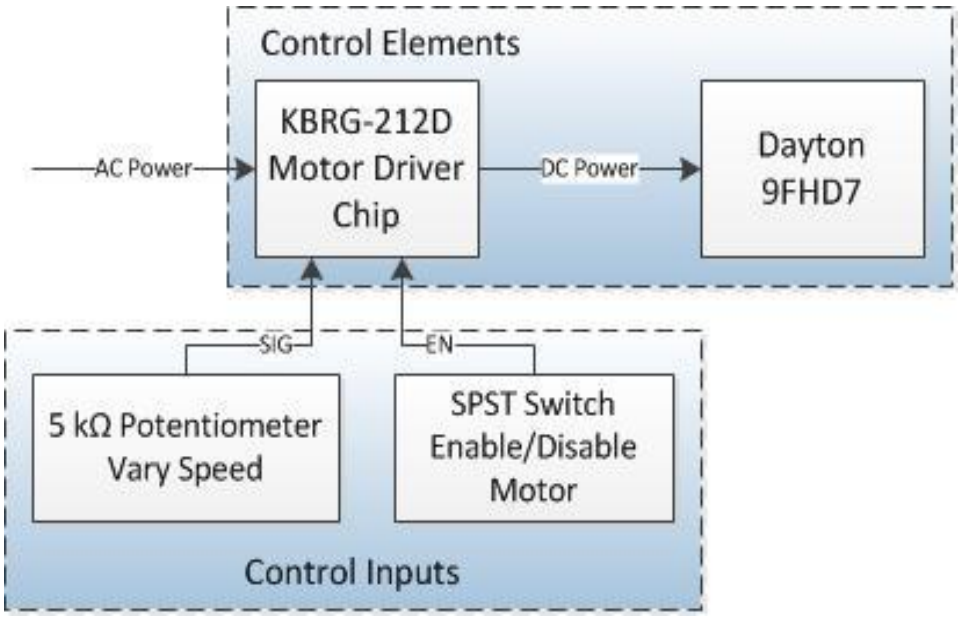


Figure 5.1.3a Motor Control Flow Chart

Figure 5.1.3 is a flow chart that gives a visual representation of how this process was configured. In the following subsections both the Control Elements and Control Inputs will be discussed.

The KBRG-212D is a regenerative driver chip for both permanent magnet and field wound motors. This chip both powers and controls the Dayton 9FHD7 motor, but requires inputs to determine specifically what speed the motor should be placed at. Figure 5.1.3b is a picture of the KBRG-212D chip while Figure 5.1.3c is the Dayton motor used for this project.

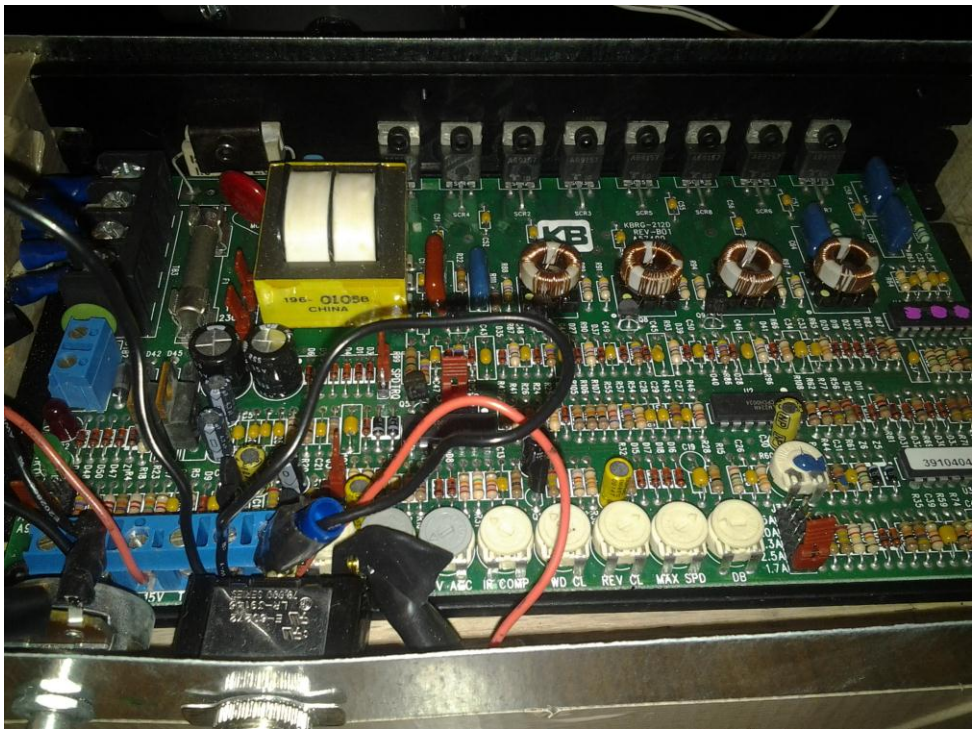


Figure 5.1.3b The KBRG-212D Regenerative Drive Chip

As seen in this image of the drive chip the input switches face out towards the user. The white circular dials towards the bottom of the picture are the electromechanical potentiometers that can be used to calibrate the KBRG-212D chip and will be discussed further in this section. The power terminals and the motor terminals can be seen on the top left of the picture where the four screws can be seen.

As for the Dayton 9FHD7 you can see both the aluminum hoops used to mount the stranded wire to the slip rings. Both of these are mounted to the motor with two screws and washers to keep them in place. The relay that is also attached to the motor is where the AC adaptors and the wire contacts for the slip ring will be attached to.

Figure 5.1.3c The Dayton 9FHD7

5.1.3.1 Motor Control Elements:

The control elements section deals with the KBRG-212D chip and the Dayton 9FHD7 DC motor. The KBRG-212D is a regenerative driver chip for both permanent magnet and field wound motors. This chip was used to both power and control the Dayton 9FHD7 motor. Figure 5.1.3.1a shows visually these components as placed within the final design.

The Dayton 9FHD7 DC motor is directly connected to the KBRG-212D via the M1 and M2 terminals on the chip. The L1 and L2 terminals are connected to the AC adapter. The M1 and M2 terminals supply 90V and 1.5A to the Dayton motor as described in the Stationary Power Supply Section. The Dayton motor was then mounted to the main Chassis' inner section.

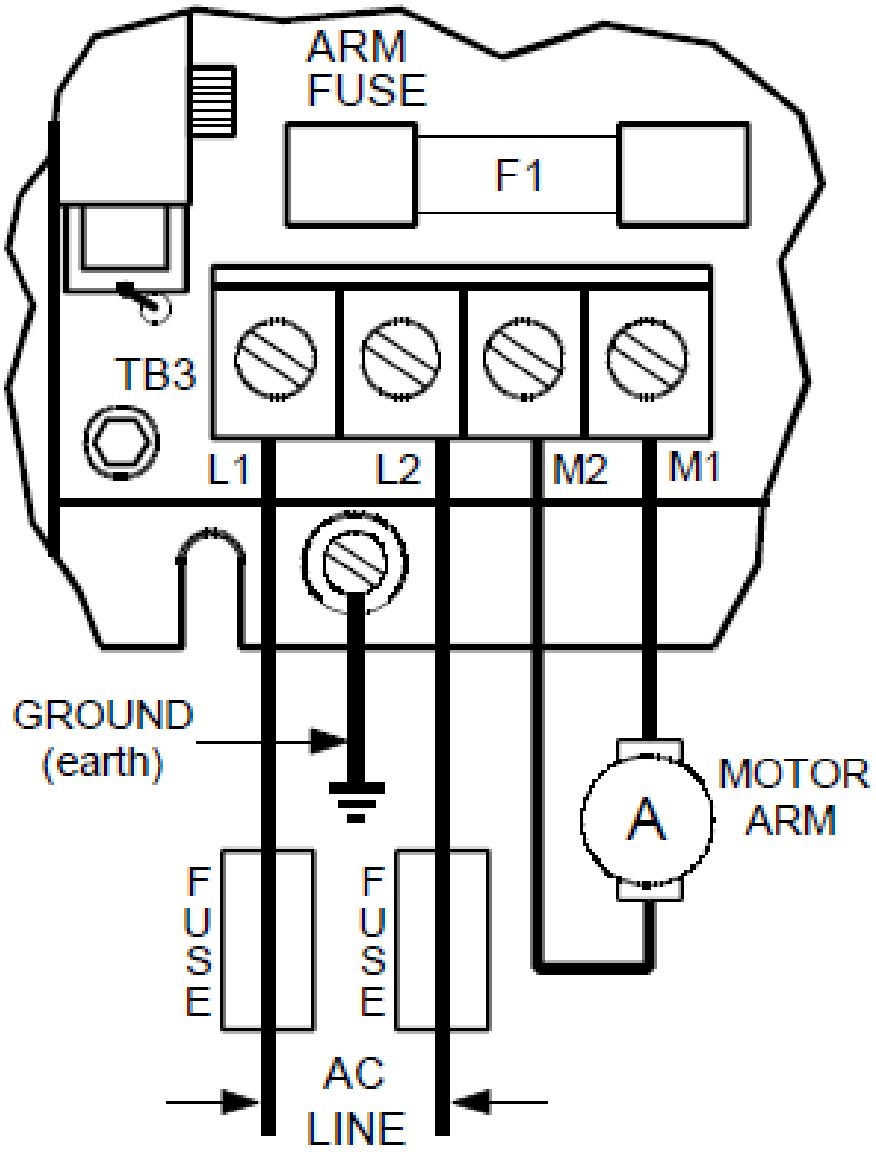


Figure 5.1.3.1a Motor and Power Connection

The KBRG-212D is partially enclosed in a wooden and steel enclosure that is also used as a mounting system for the two input switches required to control the Dayton motor. The partial enclosing was decided so that the KBRG-212D was capable of being easily reconfigured and so that the OL indicator light was capable of being observed during operation. There are two simple input circuits connected to the KBRG-212D that will be discussed further in the Control Inputs section that are connected to the SIG, +15V, COM, and EN terminals. On the KBRG-212D there are eight variable potentiometers that can be used to configure the KBRG-212D even further. For our purposes those potentiometers are all set to the factory presets as outlined in Figure 5.1.3.1b, except for the

FWD CL potentiometer which is used to set the input current into the Dayton motor down from the 1.7A preset to 1.5A. This process for calibrating the motor is outlined in the KBRG-212D section of the user manual.

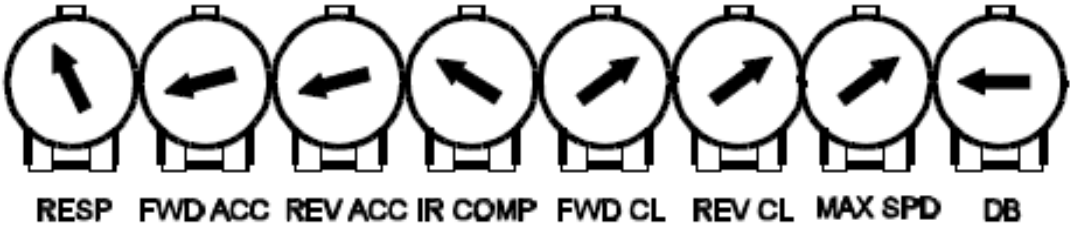


Figure 5.1.3.1b Variable Potentiometer Presets

5.1.3.2 Motor Control Inputs:

The KBRG-212D works on two simple input circuits: a 5kOhms variable potentiometer and a Single Pull Single Throw switch. The Drive Chip is capable of a variety of configurations including Unidirectional forward or reverse, bidirectional, and even an analog signal input instead of a potentiometer input. Due to time constraints an analog signal input was not used since it required isolation from the circuit due to the fact that both of these inputs are not isolated from the AC power taken in by the circuit. However, these portions of the circuit means that the device can be scaled up to a digital input allowing for much more control over the motor speed and even possible wireless control of the motor. The enable input of the circuit can also be used in this method allowing for a digital on/off signal.

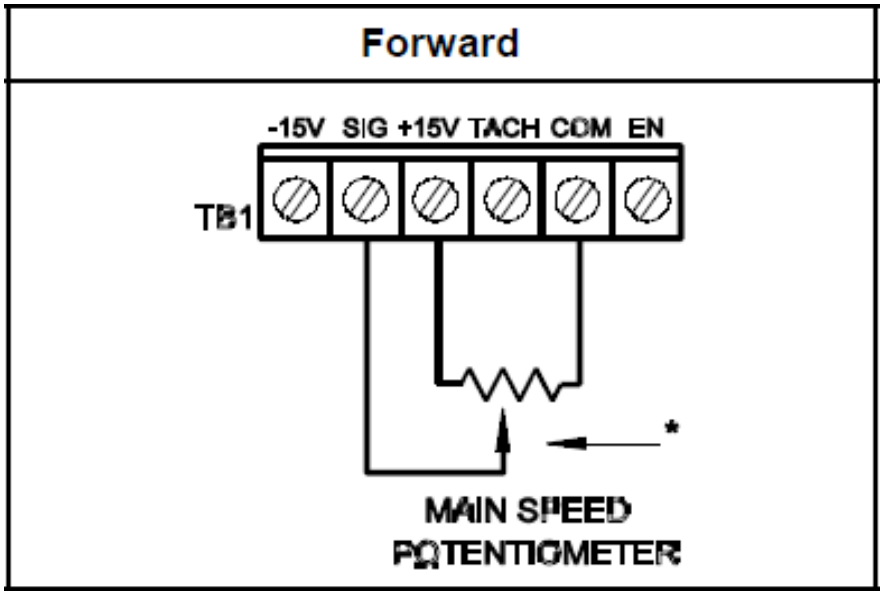


Figure 5.1.3.2a Speed Control Circuit

For our purposes we used the simplest method of a potentiometer and a SPST switch. For the potentiometer we used the forward configuration shown in Figure 5.1.3.2a. In this configuration the motor's speed can be increased or decreased by turning the electromechanical trimmer dial counter-clockwise [decrease speed] or clockwise [increase speed]. The positive terminal of the potentiometer is fed into the +15V terminal while the trimmer terminal is fed into the SIG terminal.

The neutral terminal is fed into the COM terminal along with the neutral terminal of the enable circuit. The switch as just mentioned has its neutral terminal in the COM terminal and its live terminal fed into the EN terminal, this connection is shown in Figure 5.1.3.2b. As seen in this image the J7 jumper is placed into the CTS position. The KBRG-212D is capable of both a regenerative stop and a coast to stop setting; we chose the coast to stop setting in order to have a much smoother stop of the LED apparatus when the motor is powered off.

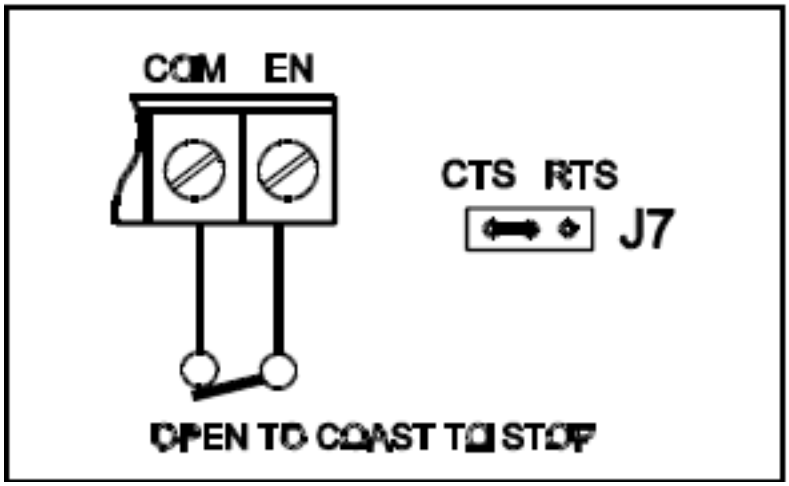


Figure 5.1.3.2b Enable Circuit

5.1.4 Display Alignment Sensor:

While our original plan was to both control the motor and align the display using a sensor in our research this changed during the construction process and ended up with the sensor only being needed for aligning the display. As discussed in the research however, the best method to solve this issue was through the use of infrared. In our case we used a pair of infrared LEDs. Figure 5.1.4a is a flowchart of this process.

The circuits we created rely on a property common to LEDs in which when subjected to light they produce a voltage deference across their leads. However, this value is very small and can barely be detected. So in order to detect it we

used an LM358 op-amp to detect these small voltage changes. The intention was to send an infrared signal to a reflective surface and receive it.

In order to create a signal spike in the sensor an aluminum plate was mounted on the top surface of the Chassis that reflected the infrared beam back to the receiving LED. This beam caused a voltage difference in the LED, this difference in voltage on the LED causes the voltage difference within the op-amp to show a voltage on the output of around 2.7-3.6V. This is instead of the usual voltage on the output which is very small and considered as zero by the microcontroller.

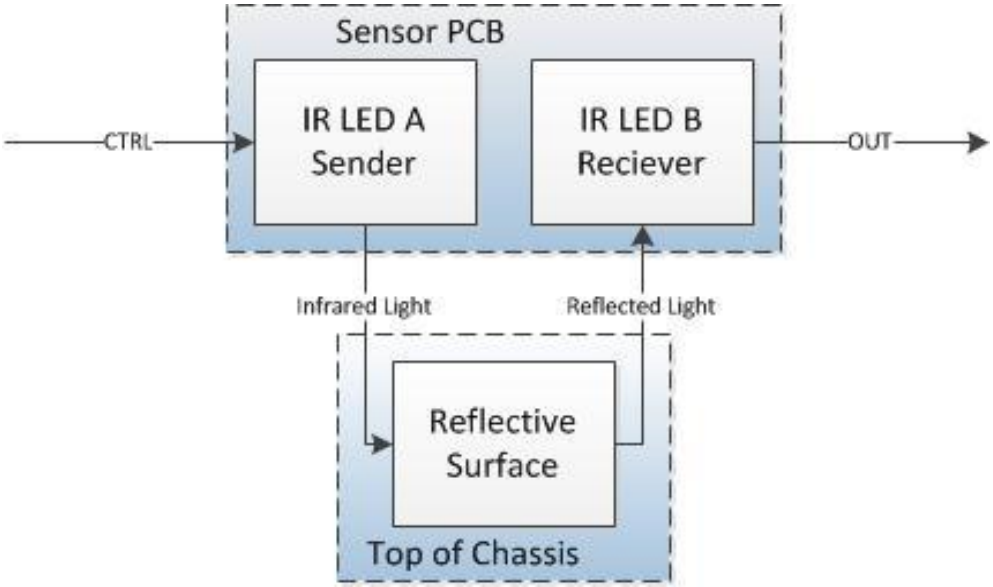


Figure 5.1.4a Display Alignment Sensor Flow Chart

The microcontroller uses this signal as a hardware interrupt which is further outlined in the software section of the project.

There are two circuits we used to implement this design. The first circuit, displayed in Figure 5.1.4b, is the sending circuit. As seen in this circuit we used the LM358 op-amp to implement this circuit. In this case a 5 volts Vin and Vcc is required to power the circuit. The minus terminal of the op-amp reads about 2.5 volts. The CTRL line was connected to the rotating microcontroller and in essence was always set to high when the device needed to align the display. This high value was around 2.5 volts or more and caused the output of this op-amp to go high, between 2.7-3.6 volts, which turned the infrared LED on and caused it to begin sending signals.

This signal as stated above was sent to an aluminum plate that the circuit passed over and reflected off the plate and into another circuit connected in parallel with the sending circuit. Both IR LEDs were place next to each other on the housing

PCB so that they could better send and receive signals. The second circuit that was implemented in our design is this receiving circuit, seen in Figure 5.1.4c.

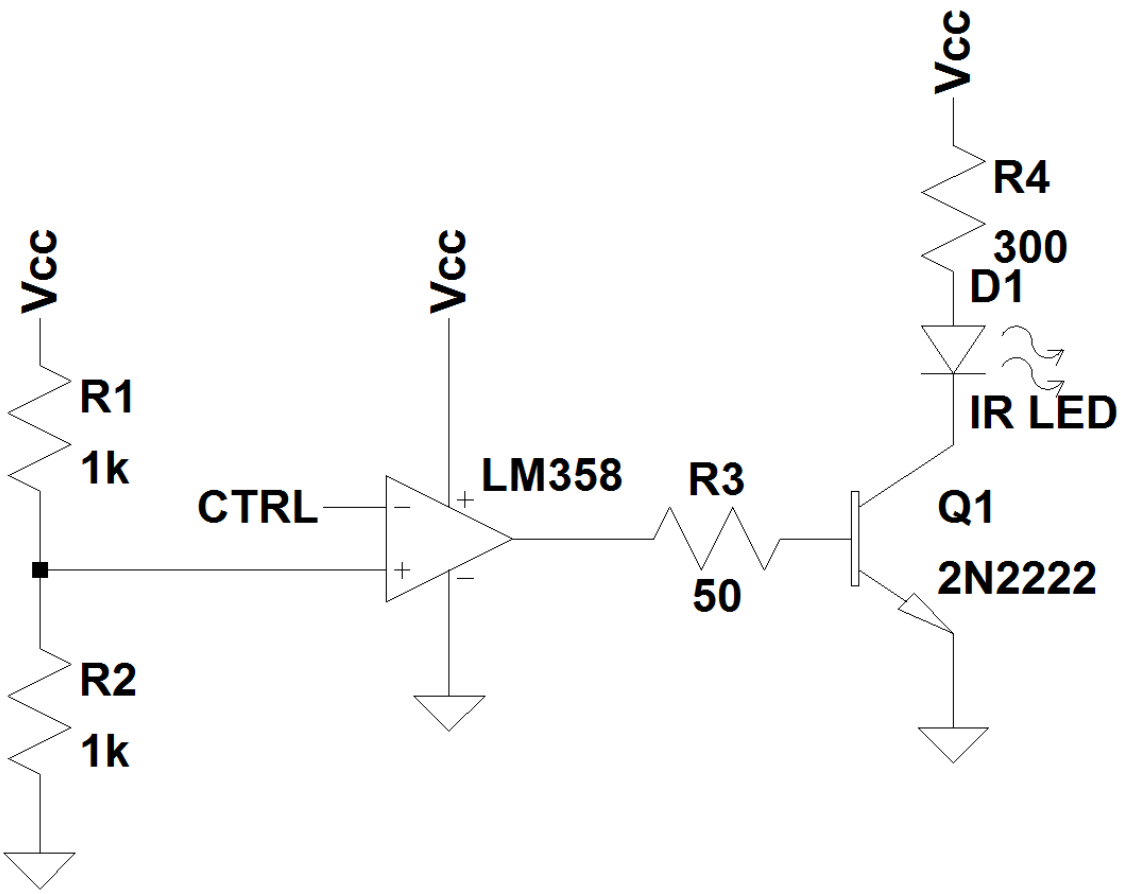


Figure 5.1.4b Display Alignment Sending Circuit

The receiving IR LED is in an off state during operation of this circuit. When the reflected infrared light hits this LED it reads a potential difference along its leads and causes the output of the LM358 to go high. The positive terminal has a potentiometer that was preset to read 2.5 volts on the positive terminal of the LM358 comparator. This potentiometer allowed for the receiving circuit's sensitivity to be either increased or decreased by changing the voltage entering the minus terminal of the LM358. When the infrared LED was hit by the beams of its sister LED it created a voltage drop on the LED. This deference caused the positive terminal to fall lower than the minus terminal's signal and forced the output of the op-amp high. When the op-amp goes high it sends a voltage drop around 2.7-3.6 volts to the microprocessor. Since we are using infrared lights and this whole process is invisible to the human eye we used a Green LED as an added indicator to the circuit. This indicator blinks when a "hit" is read in the receiver allowing us to see whether the sensor is working or not for trouble shooting purposes.

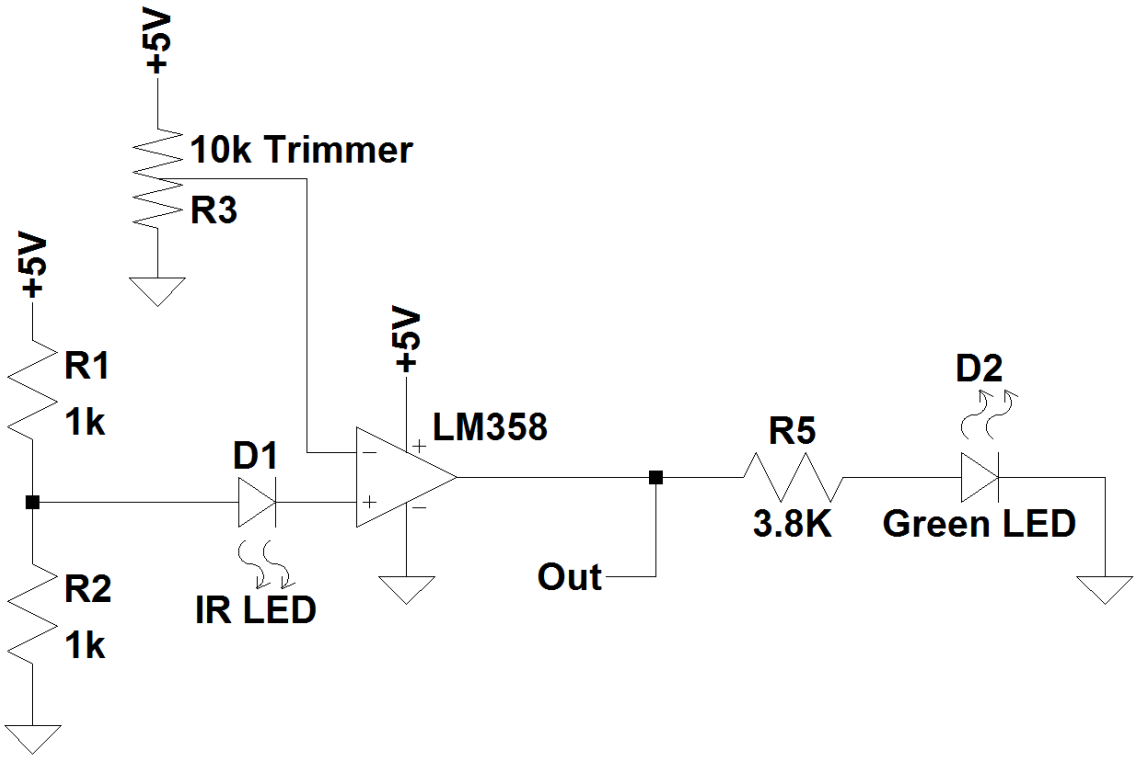


Figure 5.1.4c Display Alignment Receiving Circuit

The Eagle board layout of this entire circuit that was used for the PCB of the final circuit can be seen in Figure 5.1.4d. In this layout you can see the input and output terminals on the left side that include the two inputs Vcc, and CTRL. Just below that is the OUT terminal, and then finally the GND or common. Everything is connected directly to the microcontroller on the rotating side, though the Vcc is capable of being connected to an external power supply as long as the GNDs are common between the microcontroller and this external Vcc. The Vcc required to run the device is 5 volts. Also noticeable in this board layout are the two adjacent IR LEDs. These through hole LEDs were solder into their ports with a little length left on their leads. This allowed us to choose either to read from the z-axis [Coming out of the board layout] or the positive x-axis of the board by bending the leads to reface the LEDs.

In addition Figure 5.1.4e is a photographic image depicting the Display alignment sensor and the reflective surface used in the final design of the product. In the image you can see how the sensor would traverse over the reflective surface and trip. The surface used as stated was a sheet of aluminum metal that was attached to the surface of the chassis top with permanent double-sided tape. The sensor as seen in the image was mounted to the bottom of the aluminum bar that was rotated. It was mounted there with three screws and washers that created a tight sandwiching effect preventing the board from shifting.

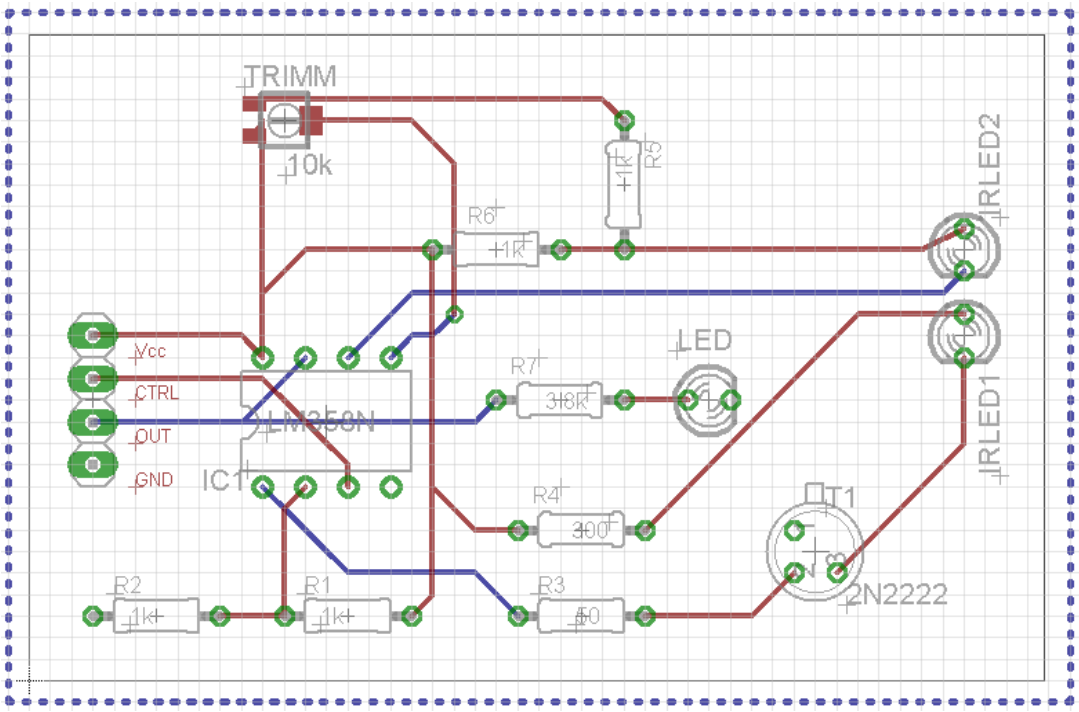


Figure 5.1.4d Display Alignment PCB Layout

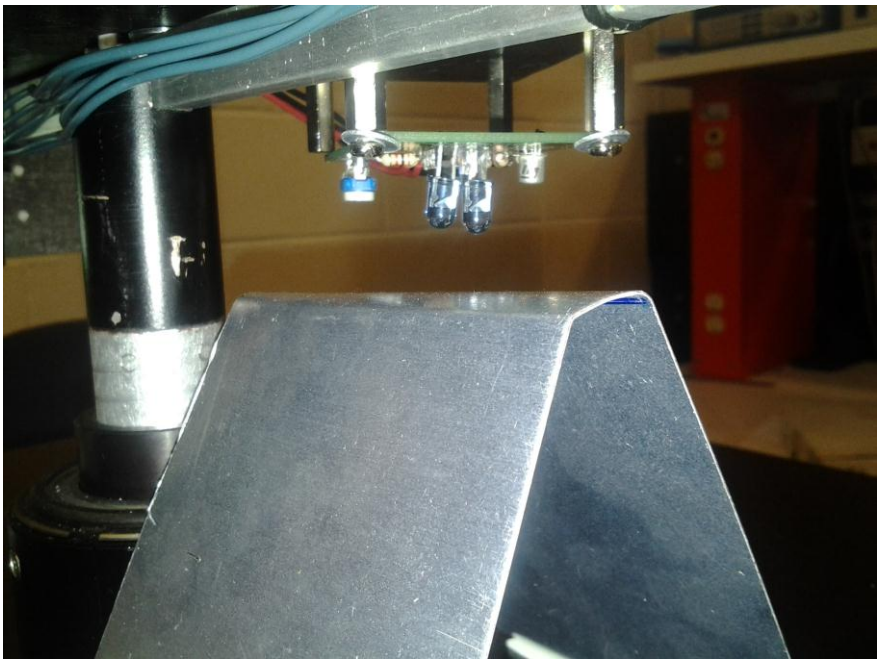


Figure 5.1.4e Sensor With Reflective Surface

5.1.5 Power Supply:

While our original plans for the device were to entirely power the motor and the rotating side via an AC outlet there were two major issues that appeared during the construction process of our original design. First was powering the motor. Our original PWM circuit effectively controlled the motor but being able to supply 90V and 1.5A without the power supply blowing due to the motor's sudden spiking during operation turned out to be a very difficult thing to do without extra expenses we were unable to commit to. The second problem that showed itself was that our slip ring had some inconsistencies in power transmission. While it effectively transferred the power with little dissipation, even during rotation, it would sometimes loose contact to the slip ring and fail to send any actual power. This would have effectively repeatedly reset the microcontroller and caused unforeseeable consequences to the lifetime of the device. Because of these issues we altered our design slightly in the power transmission department, but heavily in the motor control department. These changes are outlined in the following subsections.

5.1.5.1 Stationary Power Supply:

The stationary power supply was only needed for the Dayton motor and was completely controlled by the KBRG-212D as outlined in the motor control section. AC power was transferred to the control circuit directly via an AC outlet plug with a built in single pole single throw switch that turns the power transferred through the plug off, this is primarily needed as an emergency off switch for the motor controller.

The KBRG-212D has a built in power supply that can be used for a variety of motor types. The settings we used are specifically for our motor selection, which is on the lower end of the control circuits capabilities and could be scaled up with a far more powerful motor if desired. Using the 115AC input configuration [J1] and the 90V [J4], 1.7A [J3] output configuration we are capable of adjusting the 1.7A down to the 1.5A via the FWD CL variable potentiometer and thus power the Dayton 9FHD7 with little difficulty. In order to get an exact measurement of the current we plugged an amp meter in series with the motor and locked the motor shaft. Then turning on the Driver Chip we quickly adjusted the FWD CL potentiometer until the amp meter read around 1.5A. The KBRG-212D can be scaled up to 230AC power setting and run both permanent magnet [as we are doing] or field wound motors. It is also capable of running an 180V permanent magnet motor.

5.1.5.2 Rotating Power Supply:

The rotating power supply consisted of three things, an AC to DC variable converter plug, a slip ring, and a 9V battery. An AC to DC converter was used to plug directly into a standard wall outlet and output 3.3V DC and 2A. This DC power was then transferred directly through the slip ring to the RGB array circuits. Due to time constraints and issues with regulating the DC power

transferred over the slip ring, a 9V battery connected using a DC plug was used for the microcontroller to prevent damage to it via fluctuations caused by the slip ring. The Text display LED array was powered directly off the microcontroller while the RGB array was powered via DC transferred over the slip ring. This design was chosen because the microcontroller could be damaged when attempting to supply the 1.5-2.0A that the RGB LED array was capable of pulling during sustained operation.

5.1.5.3 Slip Ring Design:

In order to transfer power to the rotating side of the device we needed two slip rings. These rings consisted of two copper washers attached to the shaft of the bottom section of the LED apparatus. Here two lengths of stranded copper wire were mounted on the motor and wrapped around the shaft of the LED apparatus but not attached to the shaft directly. An insulating material used for cable line repair was placed between the copper washers and the shaft of the LED apparatus. Two wires were soldered to the copper washers and wired through the Apparatus' center via a hole drilled through it. These wires were then lead up through the Apparatus's center to the control side of the LED apparatus and connected to the RGB LED arrays as a live and neutral wire to complete the circuit. Power was effectively applied to the copper washers from the wires as the device rotated. There the power traveled from the washers to the RGB LED arrays to support their operation. Figure 5.1.5.3a is a visual representation of the design of a single slip ring. In this design you can get a basic idea of how we planned out the slip ring's construction. This image was created in DraftSight.

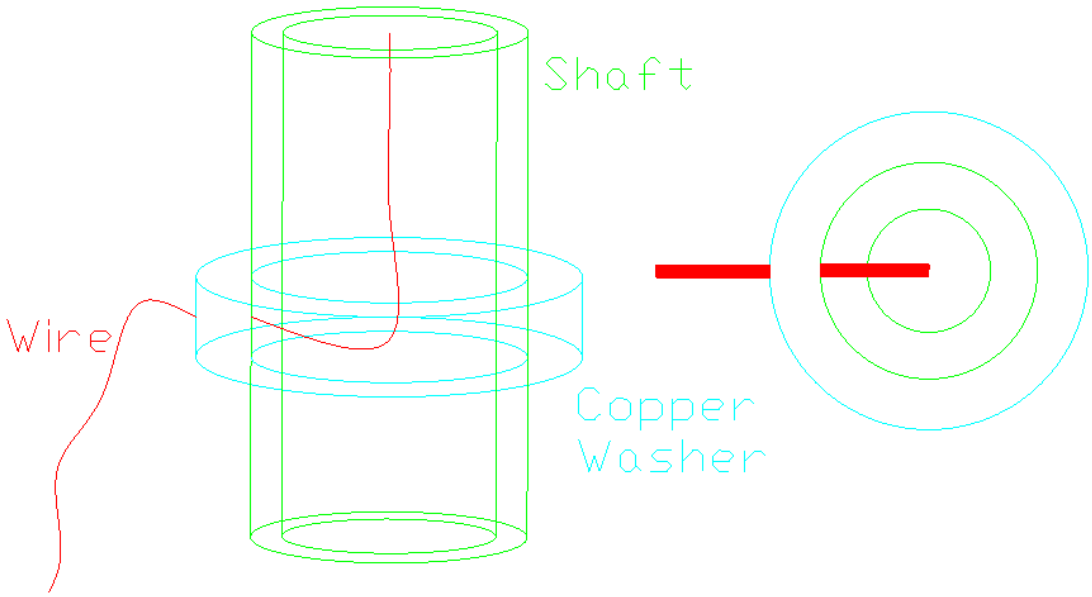


Figure 5.1.5.3a Slip Ring side and top view

The actual slip ring, depicted in Figure 5.1.5.3b had a lot more requirements to it. As seen in the picture two aluminum hoops were used to elevate both the contact wires and mounted by being threaded through a cable mounting piece and attached to the aluminum hoops. These hoops were fashioned to the motor itself to prevent movement. The wires that were threaded through the cable mounting pieces were then attached to a relay via a lug attached to the stranded wire's end. This relay was where the DC power would be transferred to via the AC to DC adapter. Between both copper washers electrical tape was wrapped to prevent the wires from contacting the pipe and causing the pipe to become live, or shorting both elements.

In order to prevent the aluminum hoops from being contacted also both wires were wrapped in electrical tape from the point that they tied to create the hoop that was wrapped around the shaft. This helped reinforce the contact around the hoop and was used to thread through the cable mounting piece which helped insulate the mounted wire from the aluminum hoops.

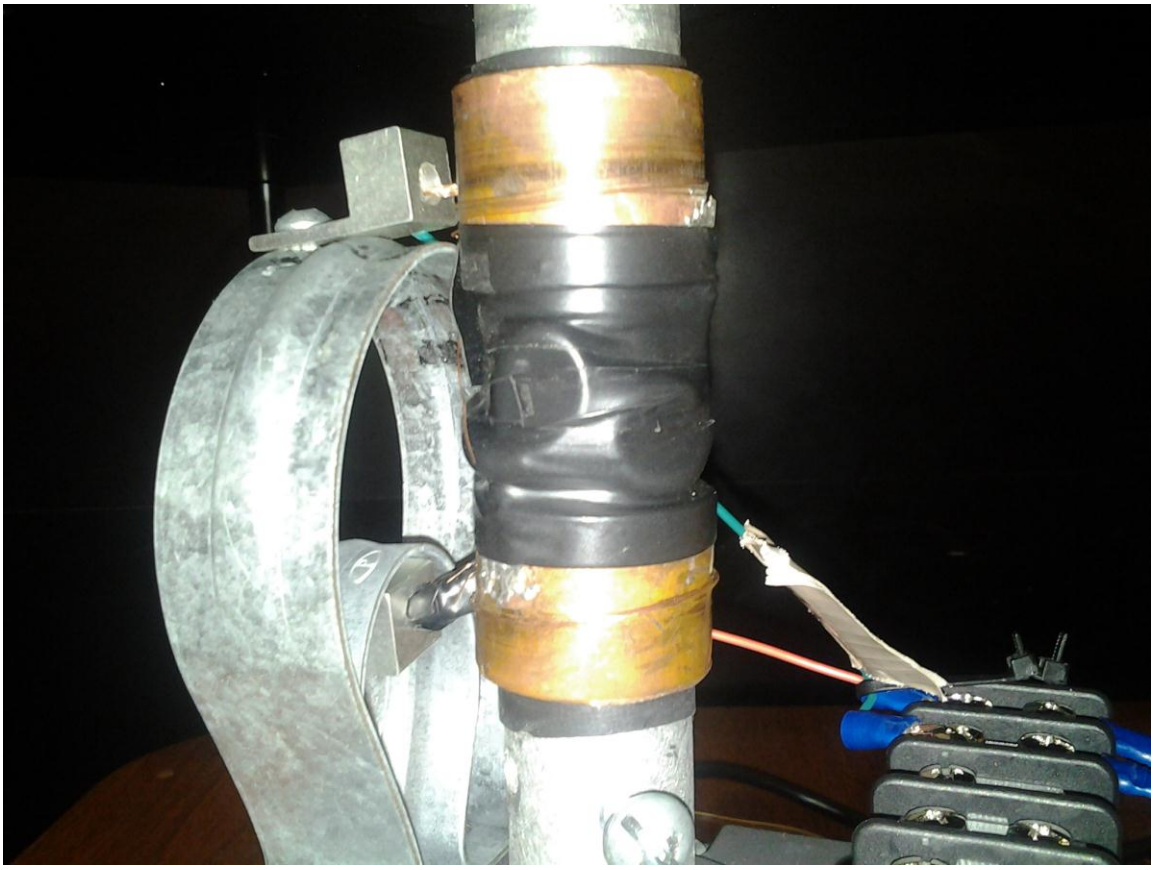


Figure 5.1.5.3b Slip Rings

5.2 Software Design:

There is a significant amount of software which had to be designed for this project. We will be focusing on the software design for each individual part of the project here. There are two primary software entities, the computer side GUI which accepts user input and the microcontroller which receives commands and data and writes to the LED controllers.

In the GUI design we will be discussing the design section of our software development lifecycle. This section will focus on organizing the set of requirements and specifications, deciding on a suitable architecture, and visually designing the GUI. No implementation will be done here. This section serves as a plan of action for the implementation and to answer all questions that may arise during development. An effective design will make all of the decisions that need to be made in order to make implementation straightforward and agile.

The microprocessor will receive data via Wi-Fi from a computer which has connected to the server running on it. This data includes ASCII values for live updating of the text display, as well as commands to turn the display on and off, change what is being displayed, and to store data in Micro SD memory. User input is received through computer side GUI and includes text messages and image input. Image input has been designed to accept almost any image file type. The GUI has been designed to be straightforward and agile, allowing for easy implementation of new features that are already supported by it.

The microcontroller will also be writing to the LED controllers for the RGB array and text array. This will require 5 signals from the microcontroller: data, data clock, blank pulse, latch pulse, and grayscale clock. Data will be written using the controllers SPI interface. The blank, latch, and grayscale clock signals are generated using pulse width modulation which relies on output compare to be generated. The output compare are set up using two hardware timers, one for blank and latch, the other for GS clock, operating at 10 MHz and1220 Hz respectively.

5.2.1 Computer Side Processing

This section will cover the various forums image processing that take place before sending image and text data to the microcontroller. We want to handle the various image formats that the user may try and use as input. To handle this, several java classes were used: ImageIO, BufferedImage, and indirectly ImageReader. Using the java ImageIO class, we open an image file by using ImageIO.read() and supply an argument of a name/path. The ImageIO class on its own will then search for an ImageReader that claims to be able to read that type of image, and decode it. ImageIO.read() will return a java BufferedImage, from which we can easily obtain the RGB values by calling the function

BufferedImage.getRGB() and supplying an x and y coordinate. Using these library the user will not need to be concerned with the image file format, and we will not need to code the tedious functions that would be required to decode the many image format possibilities. Having obtained the raw RGB data from the image, we will need to format so that the microcontroller can use it efficiently.

5.2.1.1 Image Buffer Format:

The frame buffer will begin receiving frames starting at the Frame Base Address. The frame will be stored linearly starting with the pixel in the upper left hand corner, going from left to right, and then down. Each pixel contains two bytes of color data. Figure 5.2.1.1a shows the arrangement of pixels in memory. The size of a single frame in memory can be calculated by multiplying the number of pixels by the number of bytes per pixel: 480*640 pixels/frame * 2 bytes/pixel = 614400 bytes/frame.

Pixels Stored in Memory							
Pixel	0		1		...	639	
0	0	1	2	3	…	1278	1279
1	1280	1281	1282	1283	…	2558	2559
2	2560	2561	2562	2563	…	3838	3839
.	.	.				.	.
.	.	.				.	.
479	613120	613121	.	.	...	614398	614399

Figure 5.2.1.1a Arrangement of Pixel Values in Memory

Each pixel is stored in memory as two bytes, containing the RGB color data. Red and Blue both have a color depth of 5 bits, stored in Bits(11:15) and Bits(0:4), respectively. Green has 6 bits of color depth, stored in Bits(5:10). Figure 5.2.1.1b shows the ordering of the two byte pixel data, as well as which bit is the most significant for each color.

RED MSB	RED	RED	RED	RED LSB	GRN MSB	GRN	GRN	GRN	GRN	GRN LSB	BLU MSB	BLU	BLU	BLU	BLU LSB
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 5.2.1.1b 16 Bit RGB Arrangement

Each pixel, although stored linearly in memory, can be referred to by an X and Y coordinate. X will refer to the column of a pixel and Y to the row. Table 5.2.1.1 shows the arrangement of pixels in a frame and how they will be referred to. In order to calculate the memory location of a given point P(X,Y), we will use the equation Pixel_Addr = Base_Frame_address + 2*x + 2*Line_Stride*y.

Arrangement of Pixel Values In a frame				
Pixel	0	1	...	639
0	P(0,0)	P(0,1)	…	P(639,0)
1	P(0,1)	P(1,1)		P(639,1)
2	P(0,2)	P(1,2)		P(639,2)
.	.	.		.
.	.	.		.
479	P(0,479)	P(1,479)	...	P(639,479)

Table 5.2.1.1 Arrangement of Pixel Coordinates in a Frame

5.2.1.2 Output Format Specification:

The format the RGB frames are received in must be changed into a format that will be useful to the rotating processor which will talk to the LED array controllers. The LED controllers required Grayscale information as opposed to RGB data. A Grayscale value is a 12 bit value between 0 and 4095 which will determine the duration of time, and therefore the brightness, that an LED will be on for one of its colors. Three Grayscale values can be correlated easily from the RGB color data.

There are 4 groups of 45 LED controllers, for a total of 180. Each of those 4 groups is responsible for displaying the output of various sections of the screen. Within each section of 45 controllers, 15 are dedicated to the Red outputs for that section, 15 for the Green, and 15 for the blue. Table 5.2.1.2a shows the arrangement of each of these sections, referred to as AA, AB, BA, and BB.

Arrangement of Sections							
Pixel	0	1	2	3	…	638	639
0	AA	BA	AA	BA	…	AA	BA
1							
.							
.							
239							
240	AB	BB	AB	BB	…	AB	BB
241							
.							
.							
479							

Table 5.2.1.2a Arrangement of Frame Sub-divisions

It would be ideal if each section of controllers had all of the data that it needs stored in order in memory as to minimize having to move any pointers around. Because of this, 12 output bins will be created where the processed data will be stored, which will allow the data to be accessed in a sequential manner. Table 5.2.1.2b shows these 12 bins and their starting and ending memory addresses relative to some base address. The size of each bin can be calculated by first determining the total size of a Grayscale frame. For each pixel, there are 3 Grayscale values, each of size 1.5 bytes; Size of frame in Grayscale = 480 * 640 * 3 * 1.5 = 1382400 Bytes. The size of one of the 4 sections would be 1/4th that, and then broken up into 3 subsections for each color. So the size of a single subsection or bin is: 1382400 * ¼ * 1/3 = 125200 Bytes.

Frame Output Format In Memory			
Section/Color	Starting Addr.	…	Final Addr.
AA_RED	0	…	115199
AA_GRN	115200	…	230399
AA_BLU	230400	…	345599
AB_RED	345600	…	460799
AB_GRN	460800	…	575999
AB_BLU	576000	…	691199
BA_RED	691200	…	806399
BA_GRN	806400	…	921599
BA_BLU	921600	…	1036799
BB_RED	1036800	…	1151999
BB_GRN	1152000	…	1267199
BB_BLU	1267200	…	1382399

Table 5.2.1.2b Memory locations of the 12 output Bins

5.2.1.3 Frame Processing:

This section will cover the various steps involved with converting a frame from the input format to the output format that has been specified in the previous sections. The TranslateFrame() function will translate a frame at a specified address and store the output in 12 bins as described in the output specification. TranslateFrame() begins by initializing the output pointers for each of the 12 bins. The memory address for each bin can be calculated by adding an offset value together with the base address in DDR2 memory where output is to be written. The required calculation can be seen below:

BinPointer = FRAME_OUTPUT_BASE_ADDR+i*115200 where I = 1-11

TranslateFrame() has one outer loop and two inner loops. The outer loop counter increments by two every iteration because we will be processing two columns of

the frame each pass through the loop. The columns include all for pixel sections AA, AB, BA and BB.

The first inner loop handles the translation of pixels P(X,0)-P(X,239), which are sections AA and BA. The second inner loop handles the translation of pixels P(X,240)-P(X,480) which are sections AB and BB. Within each inner loop, TranslateAndOutput is called on the current P(X,Y) pixel and P(X,Y+1) pixel. In the first inner loop P(X,Y) is always part of section AA and P(X, Y+1) is always a pixel from section BA. Table 5.2.1.3a shows what section of the frame each look handles.

Translate Frame Loop		
Pixel	0	1
	Inner Loop 1	
0	AA	BA
.		
.		
.		
239		
	Inner Loop 2	
240	AB	BB
.		
.		
.		
479		

Table 5.2.1.3a: The Translate Frame Loop

Similarly, TranslateAndOutput is twice called in the second inner loop for each in pixel, each call corresponding to a pixel in section AB and BB. The inner loops increment by 2 because each call to TranslateAndOutput will look at two pixels at a time, which will be described in more detail in the description for TranslateAndOutput(). Figure 6.1.4.b shows which inner loop L is responsible for building up the contents of the bins, and what pixel data ends up in those bins.

Each Section Written to Memory						
L		StartAddr.	Grayscale Data			
1	AA_RED	0	P(0,0)-P(0,239)	P(2,0)-P(2,239)	…	P(638,0)-P(638,239)

1	AA_GRN	115200	P(0,0)-P(0,239)	P(2,0)-P(2,239)	…	P(638,0)-P(638,239)
1	AA_BLU	230400	P(0,0)-P(0,239)	P(2,0)-P(2,239)	…	P(638,0)-P(638,239)
1	AB_RED	345600	P(0,240)-P(0,479)	P(2,240)-P(2,479)	…	P(638,240)-P(638,479)
1	AB_GRN	460800	P(0,240)-P(0,479)	P(2,240)-P(2,479)	…	P(638,240)-P(638,479)
1	AB_BLU	576000	P(0,240)-P(0,479)	P(2,240)-P(2,479)	…	P(638,240)-P(638,479)
2	BA_RED	691200	P(1,0)-P(1,239)	P(3,0)-P(3,239)	…	P(639,0)-P(639,239)
2	BA_GRN	806400	P(1,0)-P(1,239)	P(3,0)-P(3,239)	…	P(639,0)-P(639,239)
2	BA_BLU	921600	P(1,0)-P(1,239)	P(3,0)-P(3,239)	…	P(639,0)-P(639,239)
2	BB_RED	1036800	P(1,240)-P(1,479)	P(3,240)-P(3,479)	…	P(639,240)-P(639,479)
2	BB_GRN	1152000	P(1,240)-P(1,479)	P(3,240)-P(3,479)	…	P(639,240)-P(639,479)
2	BB_BLU	1267200	P(1,240)-P(1,479)	P(3,240)-P(3,479)	…	P(639,240)-P(639,479)

Table 5.2.1.3b Range of pixel data as it is stored in memory

The TranslateAndOutput() function starts by obtaining the two pixels values adjacent to each other in the same column. Table 5.2.1.3c shows pixel one and two on the left side of the image. Three combined Grayscale values are then obtained for each color, red, green, and blue from those two pixels. A combined Grayscale value is a 3 byte data structure than contains two 12 bit Grayscale values that have been melded together. The 3 byte combined Grayscale values are then written to memory, stored in their appropriate bin, and following that the output pointers are incremented by 3. Figure 6.1.4.b shows how the combined Grayscale values are stored in memory as they are output.

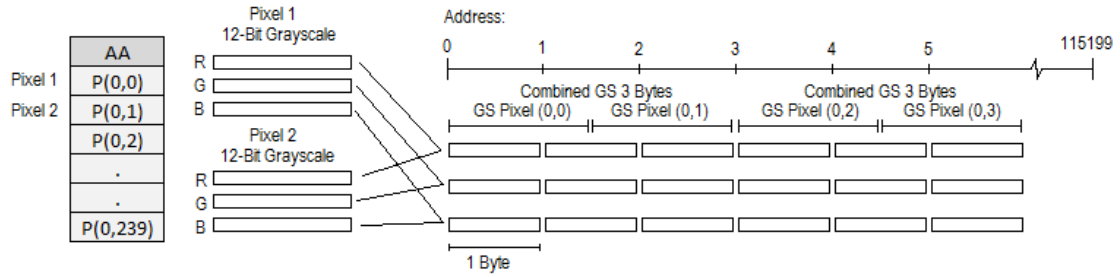


Figure 5.2.1.3c Combining Grayscale values and storing in memory

GetRedCombinedGS() is used to obtain the combined red Grayscale value for the two pixels it is called with as arguments. The red data is stored entirely in the first byte of the pixel dat. The first byte of the color data contains the red values. Using a logical shift right function with an argument of 3, the unwanted green data is pushed out. The process of isolating the red value can be seen visually in Figure 5.2.1.3a.

The red value is then converted to a grayscale value by calling a function to convert the byte. The red data isolation process is then repeated for the second pixel, and then also converted to grayscale. Following this, the two grayscale values for each pixel are combined into a three byte structure using a combine grayscale function which returns a 3 byte structure.

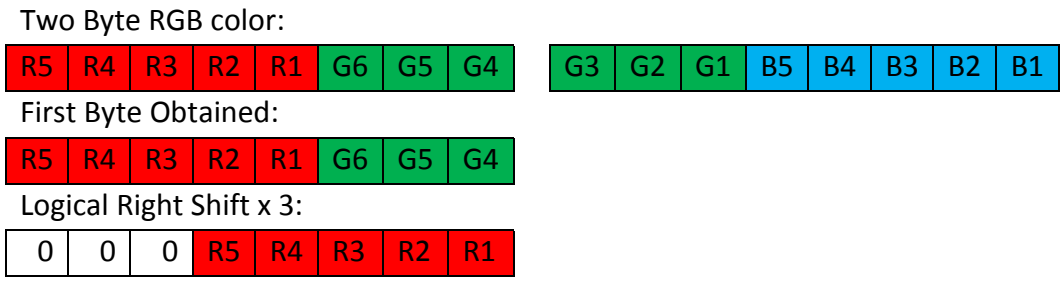
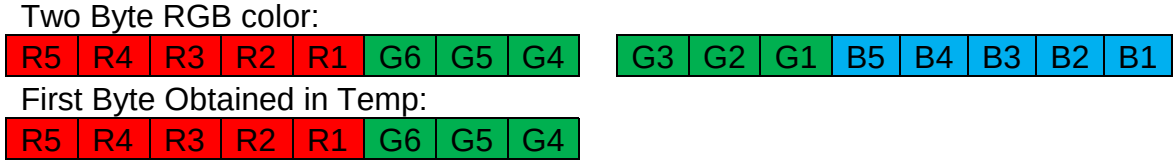


Figure 5.2.1.3a Visualization of Isolating Red RGB Value

Obtaining the green pixel values entails a little more effort since its values are spread across two bytes. The first byte of the pixel is obtained, which is then has the logical AND performed on it with 0x07, which zeros out any red data in that byte. The first byte is then shifted left 3, so that its three LSB are zero and able to be combined with the 3 bits of green data from the second byte. The second byte is obtained in a temporary variable and shifted to the right by 5. Combining the first byte and the temp variable with a logical OR operation gives the complete green data. Figure 5.2.1.3b gives a visualization of the logic used to isolate green.

The color isolation process can then be repeated for the second pixel. The two green RGB values are then converted to grayscale on lines 8 and 16. Following this, they are combined into a single 3 byte structure that is then returned on lines 18 and 19.



AND with 0xF8:

0	0	0	0	0	G6	G5	G4

Logical Left Shift x 3:

0	0	G6	G5	G4	0	0	0

Obtain Second Byte in Temp2:

G3	G2	G1	B5	B4	B3	B2	B1

Logical Right Shift x 5:

0	0	0	0	0	G3	G2	G1

Combine Temp1 and Temp2 with AND

0	0	G6	G5	G4	G3	G2	G1

Figure 5.2.1.3b Visualization of Isolating Green RGB Value

The blue color data can be obtained from the second byte of the pixel data. Because blue is already completely to the right, isolating it is as simple as performing a logical AND on it with 1F, zeroing out any green data present. The grayscale values are obtained using ToGrayscaleRB() and then combined. Figure 5.2.1.3c provides a visualization of the logic for isolating the blue RGB value from the pixel data.

Two Byte RGB color:

R5	R4	R3	R2	R1	G6	G5	G4

G3	G2	G1	B5	B4	B3	B2	B1

Obtain Second Byte

G3	G2	G1	B5	B4	B3	B2	B1

AND with 0x1F

0	0	0	B5	B4	B3	B2	B1

Figure 5.2.1.3c Visualization of Isolating Blue RGB Value

Converting pixel data to Grayscale is accomplished using ToGrayscaleRB() and ToGrayscaleG(). The need for a separate function for green is because green contains an extra bit of color depth, and can be mapped to a more precise grayscale value. A grayscale value ranges from 0 to 4095, represented by 12 bits. Figure 5.2.1.3d shows the mapping of values for both Red/Blue and Green.

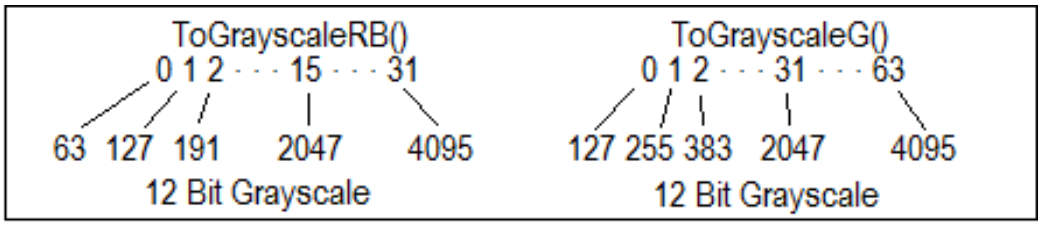


Figure 5.2.1.3d Grayscale Mapping Diagram

The CombineGrayscale() function is used to combine two grayscale values into a 3 byte data type and then return those 3 bytes. A grayscale value is stored in 2 bytes even though it only contains 12 bits worth of information. Because of this, the first 4 bits of every grayscale value are 0b0000, which is why it would be nice to compact 2 of these grayscale values together to eliminate the wasted bits and decrease the overall size of the date. The CombineGrayscale() functions behavior can be visualized in Figure 5.2.1.3e.

Grayscale Value 1

0	0	0	0	24	23	22	21
20	19	18	17	16	15	14	13

Grayscale Value 2

0	0	0	0	12	11	10	9
8	7	6	5	4	3	2	1

Logical Left Shift x 4 Grayscale Value 1 Byte 1

24	23	22	21	0	0	0	0

Logical Right Shift x 4 on a copy of Grayscale Value 1 Byte 2

0	0	0	0	20	19	18	17

AND Grayscale Value 1 Byte 1 with the copy

24	23	22	21	20	19	18	17

Logical Left Shift x 4 GS1 Byte 2

16	15	14	13	0	0	0	0

AND Byte 2 with GS2 Byte 1

16	15	14	13	12	11	10	G9

Return GS1 Byte 1 and 2, and GS2 Byte 2

24	23	22	21	20	19	18	17
16	15	14	13	12	11	10	9
8	7	6	5	4	3	2	1

Figure 5.2.1.3e Visualization of Combining Grayscale Values

5.2.1.4 PC Wi-Fi Communications

The PC will have to send messages to the microcontroller. For this we will be using Wi-Fi in ad-hoc mode. We will be using the TCP protocol over Wi-Fi in order to ensure that the packets are received without error. We will be designing our own simple communication protocol that is suitable for our purpose. It must be fast and it must have a way of differentiating between types of data. Since we will be sending different types of data we will have to add a header to each data stream. The possible types of data that we may send are: image, text (main), text (small), and command. With four different possibilities we will need 2 bits for the header. We will actually just send the first byte with our header data which will be assigned the following possible values shown in Table 5.2.1.4.

Header Bits	Data Type
000	Text(Small)
001	Text(Main)
010	Image
011	Command

Table 5.2.1.4 Header Information

The small text data is meant to appear on the secondary smaller display which will give the effect of text on top of the current main image. The header simply decides which display to send the data. Each data type has a fixed size that is expected to be sent. For the small text display the expected size is 15 bytes, on1 byte is the header and then each character is one byte for a total of 14 characters. We are currently allowing 7 characters per line. The image data type will be a single frame that will be displayed constantly on each rotation. Lastly, a command signal can include commands to clear the main screen image or the text screen image. We are also currently able to use a command signal to have the microcontroller run the cycle test code which cycles through each LED in order. A simple Wi-Fi transmission is shown in Figure 5.2.1.4. The figure assumes the appropriate header is added from the PC.

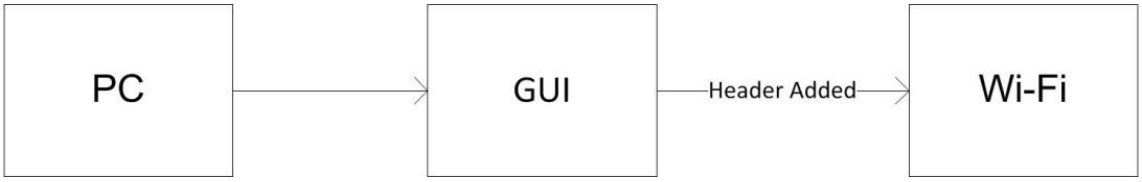


Figure 5.2.1.4 Wi-Fi Transmission Flowchart

5.2.1.5 Display Alignment Sensor Software:

An IR sensor is being used as input into the microcontroller. The sensor generates a high voltage when tripped. In software, we have set up a hardware interrupt which looks for a rising edge on the pin for sensor input. When the interrupt is generated, a flag is set to true and allows the display code to execute. The display code sets this flag back to false upon completion and then waits for the flag to be set true once again.

5.2.2 Microcontroller Software Design:

This section will cover the software design requirements for the rotating microcontroller board. Software requirements include reading the preprocessed data and image frames arriving at the board via Wi-Fi connection, and outputting this data to the LED array.

5.2.2.1 Modes of Operation:

The controller has various modes of operation and is receiving various data and commands via Wi-Fi from the stationary controller. Data includes processed frames, which are to be stored in a frame buffer, processed still images, and text data which will be displayed on a small text array. Sensor data is also received and is used to determine when to start displaying the device. Additionally, commands will be received that change the operation modes of the POV display.

The main LED array displays data that corresponds to a single image that has been stored in memory. The main array output can either be in IMAGE_MODE, or OFF_MODE. In the case of OFF_MODE the main LED array will not display anything, however the text array could still be in use.

The text array is in use, it will be operating in TEXT_MODE. While in TEXT_MODE various commands will alter the way in which text is being displayed. For instance scrolling text can be enabled and the speed at which the text scrolls can be calibrated via commands being received from the stationary controller, which received those commands via USB from the GUI interface on the computer. The text array can also be in an off mode when it is not in use which is simply OFF_MODE. Figure 6.2.1 shows the various state combinations the Main Array and Text Array can be in after receiving a single state change command.

5.2.2.2 Outputting Data to LED Array:

The rotating microcontroller will be responsible for outputting the color data the LED controllers. There are 180 LED controllers, 90 represented an A column, and 90 representing a B column, although both columns output to the same LEDs. Both the A column and B column will output to the LEDs at 22 frames per second, staggered such that the LEDs will flash at 44 frames per second. Two clock driven interrupt handlers will tell the A and B columns of LED controllers to display at the appropriate times. After a column is displayed all of the controllers in that column require a blanking pulse of 20ns in length. On lines 6 and 14 of the interrupt handler pseudocode, the Column Written flag is set to false so that a separate loop can begin writing the new pixel data to be displayed to the LED controllers.

The A column has been instructed to start displaying by calling DisplayAColumn(). A pulse is sent on the XLAT pin for the A column of duration 20 ns, which moves the data written in the controllers shift register to the grayscale register. The controllers now require the GSCLK signal to tick 4096 times at 30 MHz. The values in the grayscale register will determine how long the outputs from the LED controllers to the LEDs stay on, effectively determining the color that will be displayed. On lines 2 and 13, XLAT is pulsed, and on lines 3 and 14 a grayscale counter is initialized. A loop is then entered that the program

will remain in until pulseGS has been set to true 4096 times. A clock interrupt handler sets pulseGS to true at a rate of 30MHz, and each time it's true, the GSCLK for the column is pulsed for 16ns.

A loop will be running which writes the data to the LED controllers after each time a column is displayed, because that column now requires new data. Lines 2 and 6 of this pseudocode check the Column Written flag to see if new data has been written since the last flash of that column. If the new data has not been written yet, a function is called on lines 3 and 7 which will write the new data to the LED controllers. The Column Written flag is then set to true.

The Write Column functions handle writing data to the controllers one bit at a time. There are 90 controllers in total used in column A. This is divided into two groups, AA and AB, each with 45 controllers. AA and AB are each divided into 3 groups of 15 controllers for red, green, and blue. This makes for a total of 6 groups of 15 controllers. A single controller requires 192 bits and with each group containing 15 controllers, 2880 bits will be written to each group. In line 2, a loop will be entered that will continue until 2880 bits have been written to all 6 groups. The rate at which the data can be written to the controllers is limited to 30MHz, because of this a clock interrupt will set SCLKpulse to true at a rate of 30MHz. Whenever this pulse occurs, the bits to be written for each group will be obtained as seen on lines 4-9, and then written into memory at the addresses associated with the A columns SOUT pins on lines 11-16. An SCLK pulse is then required so that the LED controls read the new bit into their shift registers.

ObtainBit() returns either 0xFF or 0xFE depending upon whether the next bit of data was a 1 or a 0. When a logical AND is performed between that byte and the output pin address, only the last bit will be altered. The first loop lines up a 1 bit in the temp variable with the index we are interested in, and then a logical AND is performed zeroing out all other bits. In the second loop, the bit we are interested in is shifted right until it is the LSB. A logical OR is then performed on that value and 0xFE, which will guarantee the return value is either 0xFF or 0xFE.

5.2.2.3 Outputting Data to Text Array:

One design feature to be implemented is a text array which consists of 16 RGB LEDs controlled by 3 LED controllers. This would in essence be a miniature of the full miniature LED array. The addition of this display required modification of the pin I/O's available to the full array, specifically the loss or XERR input. This also requires that we use the A and B array columns latching signal which both pulse at 7040 Hz to be combined into a separate output pin that pulses at 14080. 14080 Hz allows the text display to be flashed at 22 frames per second.

In order for this implementation to work, the interrupt handlers displaying column A and column B of the primary array would effectively also be flashing the text display at the same time. In the primary loop which handles writing to each

column, the text display would also need to be written to and made ready before each of the display interrupts occur.

The text display will be capable of displaying text with various settings, such as scrolling text left or right at different speeds, and color alteration. Allowing the text to scroll involves incrementing certain pointers into memory while always keeping track of the base pointer for the text data. After a certain amount of rotations of the POV display, a pointer that points to the text data is incremented and becomes the new reference base pointer. When writing the data, the reference base pointer is incremented and a modulo operation is performed to wrap it back around to the true base address of the text data. The speed at which the text will rotate depends on how many rotations of the POV device are required before the pointer is moved.

5.2.2.4 Microcontroller Wi-Fi Communications:

The microcontroller receives communications through the attached Wi-Fi shield. This board will act as a server waiting for a client to connect. Header information will have to be deciphered on this board so that this board knows where to send the data. There is a single byte variable assigned to hold the header data. Once the header has been received it is analyzed and the microcontroller is able to decide where to send the rest of the data it receives. If a control signal header is found then the data that is sent is considered an instruction to perform. If the instruction is something the microcontroller recognizes then the proper action is taken. For example, if the cycle signal is sent the microcontroller will begin cycling through the LED's until a new message is received that requests the microcontroller to take a new action. The text messages were greatly simplified by using a font table that we hard coded in the flash memory of the microcontroller. There is a two dimensional byte array that holds all of the column data for each letter and for most symbols. This font table uses standard ASCII values less 32. For the main display the text signal is very similar, it still uses the font table but we had to skip to every third LED since each RGB LED contains three individual LED's. The image header will signal an image of a fixed size. The PC takes care of the image formatting and the microcontroller simply receives the bytes and stores them in order to be displayed. Figure 5.2.2.4 shows a simple flowchart detailing Wi-Fi communications on the receiving end.

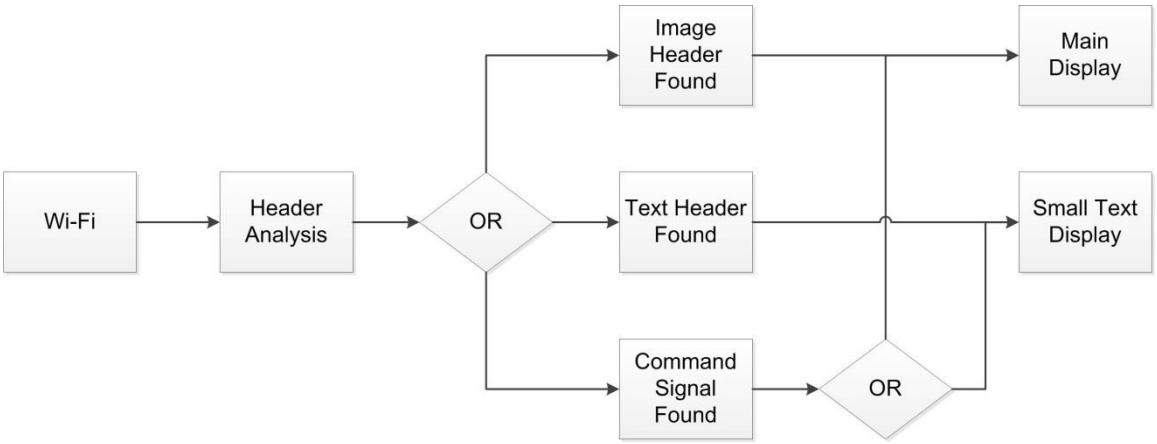


Figure 5.2.2.4 Wi-Fi Receiving Flowchart

5.2.3 GUI Design:

The GUI was implemented on a windows PC using Java. The following bulleted list will show a formal enumeration of the requirements which were implemented in the final application. Most of the items in the list are repeated from the requirements analysis in the research section. Some of the requirements are new and have been discovered while executing the design.

- Intuitive user interface
- Multiple line text message entry
- Color options for text messages
- Animation options for text messages
- Image import with simple image processing
- Image positioning
- Image cropping option
- Image clear button

A pipe and filter architecture was used for this application. The user input should be either the text message or image which will then pass through a software "filter" before being output in the proper format. The following architecture diagram in Figure 5.2.3 better illustrates the pipe and filter model we used:

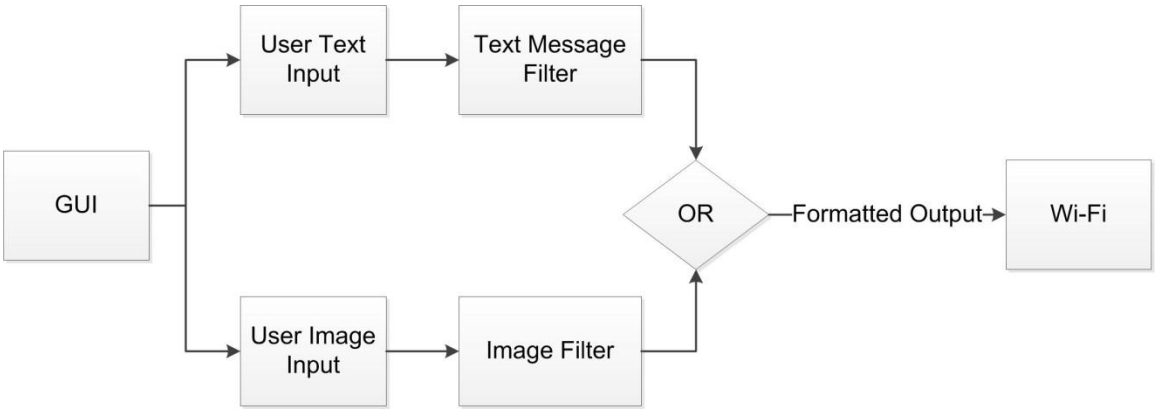


Figure 5.2.3 Pipe and Filter Software Architecture

The GUI was designed using Java in the Netbeans IDE. This IDE was chosen for its robust GUI editor which allowed us to quickly create an interface before completing any coding. Designing the interface first also helped to serve as an outline to facilitate the implementation. Creating the first visual draft of the GUI's appearance was the next step in the design. We considered the detailed designs involved in both sending a text message and sending an image in order to create a draft of the GUI's appearance.

5.2.3.1 Text Message Input:

According to the requirements there should be either a single multi-line text box, or multiple text boxes to accommodate multiple lines of text entry. We chose to have multiple text boxes. Near the text input areas we included the color options, which include preset colors as well as user defined colors. These colors will be applied to the entire text message. The current GUI design for text input is shown in Figure 5.2.3.1. This design takes all of the previously mentioned requirements for text input into consideration.

The text input will be stored in a character array. The character array will contain a pre-set length of characters. We chose to have 7 characters available for each line, which means the character array sent will be 14 characters long. This information will then be sent to the microcontroller using Wi-Fi. The information that the microcontroller needs to receive is simply the ASCII value for each character. The microcontroller is able to interpret each ASCII value and match it with a font table that is stored in flash memory in order to display the message.

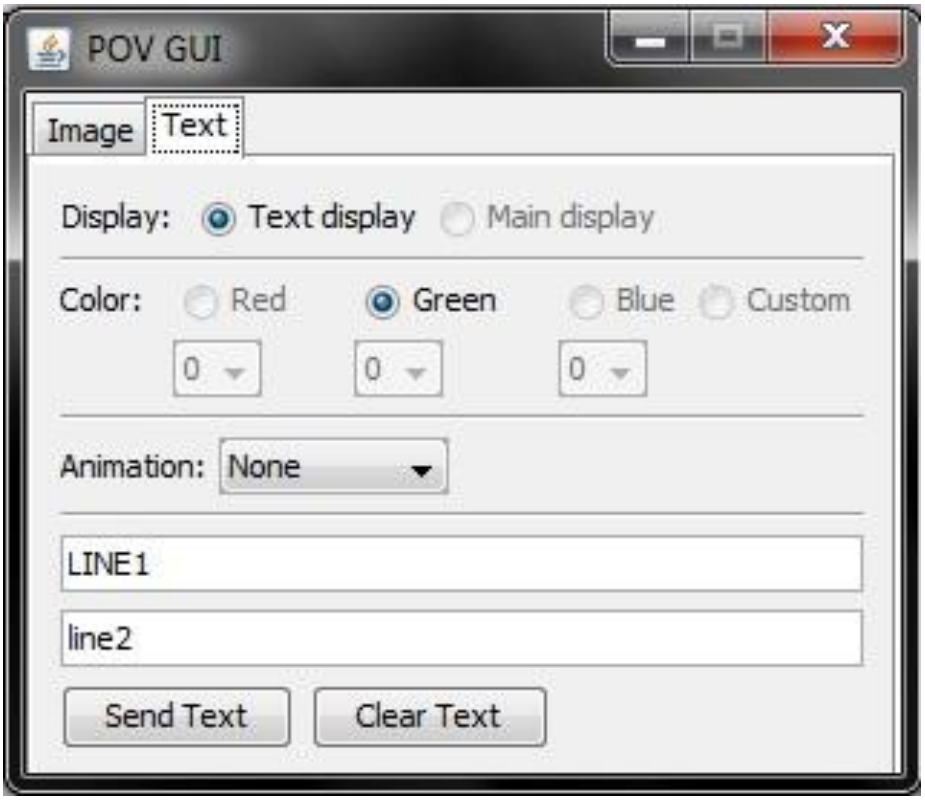


Figure 5.2.3.1 Text Input Tab

5.2.3.2 Image Input:

The image input option allows the user to select an image from the hard disk to display. The Image input appears on the GUI as a button that will then open the file chooser dialog allowing the user to select an image from the hard disk. Any common image format is acceptable. The only image formats that should be restricted are the ones that the built in ImageIO Java class is not capable of parsing. The options for images include whether or not to crop the image (when the image is too large). If the crop option is selected then only the portion of the image that can fit on the screen will be shown, otherwise the image will be shrunk to fit. Another option for image input is the position where the image is displayed; this is for images that are too small. There are nine choices available in a box shape from top left to bottom right. The current version does not fully support image messages. The file chooser is available and the user can choose an image. The selected image is then converted into a file that has the RGB data from the image organized by column to be displayed on the POV display, but the image is not sent to the microcontroller. The RGB LED array does not allow us to use Wi-Fi communications so we cannot send an image at this time. Figure 6.3.2 shows a draft of the GUI design for the image input. The final GUI design will

contain a combination of both the text input tab shown in figure 6.3.1 and the image input tab in Figure 5.2.3.2.

Figure 5.2.3.2 Image Input Tab

The Images are read using built in classes and methods for image handling. These classes include ImageIO, BufferedImage, and ImageReader. We used an ImageReader to interpret the image format and translate it. This allowed ImageIO to read the image into a BufferedImage. We then used the getRGB method of the BufferedImage class to get the pixel color values. The pixel color values were very easy to convert to our format so that we can send the image using Wi-Fi. Any blank pixels (no image data) are represented with a special value that lets the microcontroller know not to overwrite any previous value in the flash memory for that pixel. This allows us to overlay multiple images if they are small enough. This is also the reason for the clear image button in the GUI. When the clear image button is pressed, a special clear instruction is sent to the microcontroller. This is done by sending a unique clear signal that will have the microcontroller overwrite all pixels with black values.

5.2.3.3 GUI Wi-Fi Communications:

In this section we will consider how to communicate with the microcontroller using Wi-Fi. We will also be considering how to properly format the user input so that the microcontroller has all of the necessary data to update the display. We used the built in Java networking classes to perform Wi-Fi communications between the PC and the Chipkit uC32 board. The first step in Wi-Fi communications is manually connecting the PC to the “POV Display” wireless network and entering the correct security key.

5.2.3.3.1 Wi-Fi Communications Class:

Wi-Fi communications were handled in a separate WiFiConnection class. This class has methods to connect to the POV display (assuming the PC is already on the correct network), and send data. The constructor initializes a new WiFiConnection class and sets the IP address and the port number. When the connect method is called it will take the IP address and port number, and create a new socket to establish the connection. The send method was designed to for multiple input possibilities. We use a char array as an argument, which is the case for when two lines of text are being sent.

5.2.3.3.2 Wi-Fi Communications I/O:

The Wi-Fi communications I/O operations take place within the WiFiConnection class. The WiFiConnection class contains an instance of the Socket class from the built in Java network library. The Socket class handles the communications and allows the WiFiConnection class to send a payload to the microcontroller, as well as receive acknowledgements. The acknowledgements will be optional. We currently do not receive acknowledgements but it can be easily added if we need them for debugging. Another use for the optional acknowledgments may also be a loading bar which can continue to fill as the acknowledgements are received. A simple sequence diagram shown in Figure 6.3.3.2 next should help to illustrate the planned data flow for the WiFiConnection class.

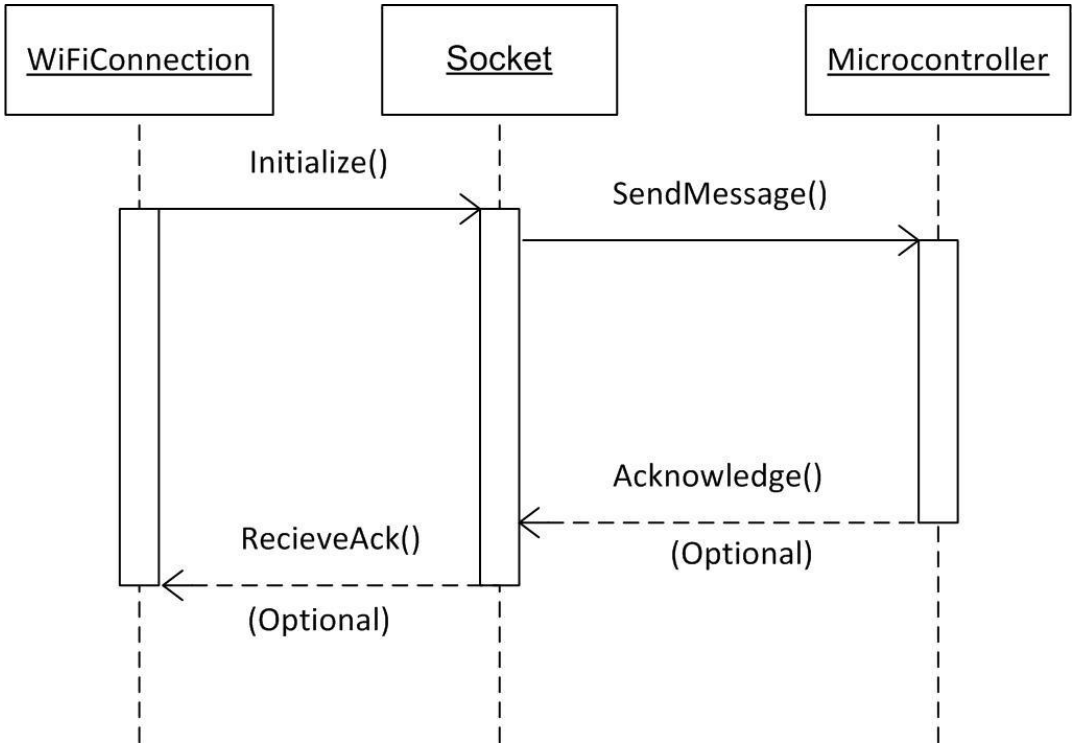


Figure 5.2.3.3.2 WiFiConnection Sequence Diagram

Before Wi-Fi communication can occur the Socket must be properly initialized. The Socket is either initialized by using the WiFiConnection class constructor or by calling the connect method with an IP address and a port number as arguments. Both the WiFiConnection constructor and the connect method throw an UnknownHostException, or an IOException. The UnknownHostException is thrown when the IP address and port number combination cannot be found on the current network. IOExceptions may be thrown when the IO for the socket does not initialize properly. After the socket initialization the input and output streams need to be initialized. The input stream is created using a BufferedReader class that is created using the getInputStream method of the Socket class. Similarly the output stream is created using a PrintWriter class that is created using the getOutputStream method of the Socket class. The creation of both the input and output stream both have the possibility of throwing the previously mentioned IOException. Now that the Socket and it's input and output streams are properly initialized the user is able to send messages to the POV display. Simply print items to the PrintWriter object and read items from the BufferedReader object. When all reading and writing is complete, the Socket and both streams must be closed. This is simple to do and simply requires a call to the close method for each object. The operating system will then have that port back in a usable state and all resources devoted to the streams will be freed.

5.2.3.4 GUI Class Summary:

In this section we will consider all of the class interactions throughout all parts of the GUI application. Classes that we will have to create include: POVGUI, WiFiConnection, TextMessage, and ImageMessage. Image reading classes include ImageReader, and BufferedImage, which will be used by the ImageMessage class. The text message class will not need helper classes since it is dealing with simple text data. The WiFiConnection class will need to contain classes from the built in Java libraries including Socket, BufferedReader, and PrintWriter. A class diagram showing these classes and their relationships to each other is shown in Figure 6.3.4. It should be noted that either a text message is sent or an image message is sent, but not both. Also the multiplicity shown for each class is one, because only one of them should exist at a time. If multiple messages are to be sent, the same class will be used with different values.

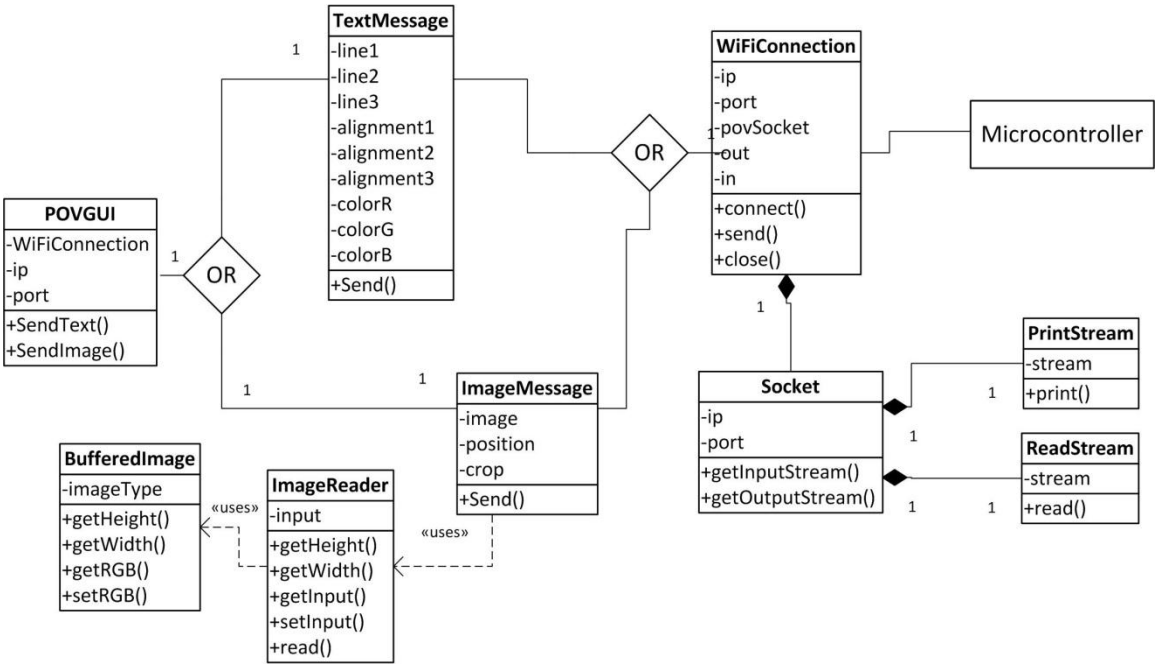


Figure 5.2.3.4 Class Diagram for the GUI

6 Prototyping:

In order to determine if our design worked under our specific conditions we needed to test them. We expected them to work theoretically but theory doesn't always work practically. This being the case we created a variety of prototypes of each section of the device that we felt may be prone to failure. These prototypes were used in the test procedure chapter to create and describe both the process and the purpose of the tests that were applied to each of these prototypes.

6.1 Slip Ring Power Transmission Prototype:

While testing the slip ring we needed a prototype circuit that could be used to tell if the slip ring was properly transferring the amount of power we needed to power the LED apparatus and microprocessor without actually connecting the processor so as to not cause any damage to either the processor or the LEDs. That being the case we created a prototype circuit using one 60 watt bulb. Since the expected amount of power needed on the opposite side of the device was around 10 watts of power then if the slip ring could power while in motion 60 watts worth of power then we know that we should have no problem powering the 10 watts. In addition, we were able to tell what the minimum amount of power needed to power the 10 watts was so we could get an idea of the loss in the system. This prototype circuit can be scaled up by adding more bulbs in series to

create larger power requirements in order to test for a scaled up version of the final device. The circuit for this prototype is shown in Figure 7.1.

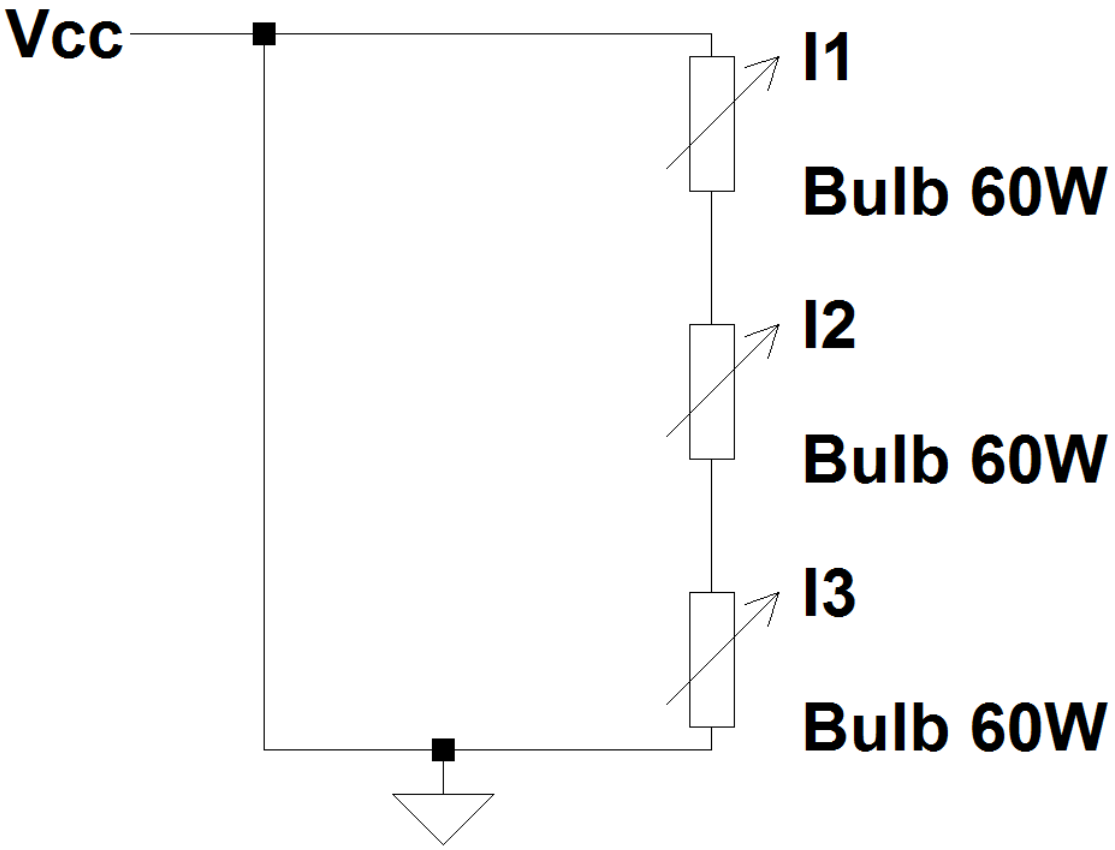


Figure 7.1 Power Transmission Prototype

6.2 Scaled LED Array Prototype:

In order to verify our LED array design works, we built two prototype of the LED array. Both prototypes used the TL5940NT models instead of the surface mount due to the difficulty of having to surface mount the IC just for testing. The first array used 4 RGB LEDs and 4 Green LEDS. This was so that we could test both the text array and RGB array LEDs with the TLC5940. The section prototype used 6 RGB LEDS, 5 of which were fully functional while the last one could only light up the red color.

6.2.1 Scaled LED Array Hardware Prototype Design:

The design for the LED array prototype was similar to the full scale LED array. The only difference was that none of the controllers were cascaded, they were single arrays. For the prototype version of the LED array did not need any

controllers cascaded. Although this did not help with testing the speed at which we could address the cascaded controllers, f(sclk), this did provide a test of the grayscale clock, f(gsclk).

7 Testing:

Since like any project it is unlikely to work exactly like we designed it the first time we turn the device on it was best to create some testing procedures to fully experiment with certain software and hardware features of the device that could prove to not model exactly like the theoretical design. These testing procedures helped us calibrate certain components of the device so that they worked more effectively.

In the following sections of this chapter there are a series of tests that were implemented with the prototypes described in the prototype chapter before. Each test will identify the objective of the test, the prototype being used for the test and a short paragraph of the results of the test.

7.1 Display Alignment Sensor Testing:

This section will cover the hardware and software test required to verify the correct and desired operation of the display alignment sensor.

7.1.1 Sensor Hardware Test:

There are a few things that we needed to determine about the sensor's hardware that required testing to insure that it would best align the LED display so that we could better control the location and spacing of the image. The objective of the first test was to determine if the sensor would correctly show a voltage pulse when it registered a movement change. The second test's objective was to determine if varying the CTRL signal to the sending circuit would have an impact on the efficiency of the sending and receiving process of the infrared LEDs. The third test's objective was to determine whether the output of the receiver circuit had a noticeable enough pulse or change in voltage. Finally the fourth test's objective was to determine what effects the potentiometer had on the sensor circuit.

For all four of these tests we needed to use both circuits of the infrared sensor outlined in the design section for the display alignment sensor, Section 5.2. The circuit was left disconnected from the microprocessor for the purposes of these tests.

7.1.1.1 Sending/Receiving Signal Hardware Test:

This test encompassed both the first and third test requirements. For this test a voltmeter was connected to the OUT location on the PCB design figure 5.2d. We then connected a voltage source to the CTRL pin. This source was used to turn on the sender circuit. The IR LEDS were placed in such a way that it was directed toward a surface with some reflectivity, an aluminum plate which was the same surface that was used as the trip in the final design. Then the CTRL pin was increased slowly until a hit was registered on the receiving circuit. This was observed by tracking whether the LED turns on or off and whether a voltage was registered on the voltmeter. Once a hit was received we then removed the reflective surface and watched to see if the LED turned off or stayed on and whether the voltage dropped or rose on the voltmeter.

The desired result of the test was to have the LED turn on when the CTRL was turned on and reached a voltage of about 2.5 volts or larger, a value that would be higher than the minus terminal of the op-amp, then to have the LED turn off immediately upon removal of the reflective surface. The test was successful and the sensor was extremely responsive to the reflective surface and most any other surface in general. The only two things that the sensor was not responsive to was a black surface or an abundance of distance between the sensor and the surface. In addition, the voltage pulse on the OUT of the circuit proved to be noticeably observable. In other words, if there was a reflective surface the OUT read a clear 2.7-3.6 volts and while there was no reflective surface the voltage was down in the low millivolts range.

7.1.1.2 CTRL Signal Calibration:

This test encompassed the second test requirements. For this test the set-up was the same as the test in section 7.1.1.1. After the circuit was set-up and the tachometer was directed away from the reflective surface, the voltage supply connected to the control pin was slowly increased. Starting at 0V the voltage from the power supply was increased by 0.5 volts up to 5V. During each increase in voltage the reflective surface was passed in front of the infrared sensor slowly. The voltage change on the voltmeter was observed during each increment.

The desired result of this test was to determine that the strength of the CTRL signal is irrelevant when it comes to the effects of the strength of the receiver “hit” signal. The result proved that below 2.5 volts the sensor did not register any value and the CTRL signal had not effect on the sensor. However, once the CTRL signal reached the 2.5 volts further increases to the CTRL line had some impact on the OUT signal increasing its value slightly. This increase however, required substantial increases in the CTRL signal in order to make an impact on the OUT signal. The difference turned out to be around a 1 volt change in the CTRL line amounting to a few millivolts change in the OUT signal.

7.1.1.3 Sensor Sensitivity Test:

The final test for the sensor was the sensor sensitivity test that used the potentiometer. This test was setup exactly as the first test. In this test the potentiometer was varied between detection tests to determine the best calibration for the sensor sensitivity.

During this test it was found that there were two sensing locations for the sensor circuit. One, which never changed with variation of the potentiometer, was right in front of the sensor. The other changed slightly within a small range at a large distance from the IR LEDs. This property was used to find the best distance at which to place the reflective surface so as to obtain an effectively consistent “hit” indication.

7.2 KBRG-212D Calibration Tests:

The motor control circuit required to control the motor required three tests to effectively use for the device. Test one was a current calibration test, which was needed to calibrate the KBRG-212D to the specific current needed for the Dayton motor. The second test was the speed control test. This test was used to determine how fast we could spin the motor before overcurrent through the motor would be an issue. The final test was the enable line. This was to determine whether the enable line would return the motor to the specific rpm value we left it at after an off/on toggle. The following subsections will discuss the process of each of these tests, the desired result, and our actual result from the test.

7.2.1 Current Calibration Test:

In this test the KBRG-212D was connected as seen in Figure 5.1.1a. The only difference is that an amp meter was connected in series with the motor. FWD CL variable potentiometer was then set to its lowest value, while all other potentiometers were set to the factory conditions shown in Figure 5.1.1b. Then the KBRG-212D was turned on and the FWD CL was quickly raised up until the amp meter read 1.5 A. This should be done as quickly and accurately as possible since damage to the motor can occur if the motor shaft is locked for too long.

The desired result of this was to test calibrating the motor for the Dayton. The actual result was close to desirable though we found that the FWD CL was very sensitive and prone to not being consistently accurate. This resulted in the FWD CL needing to be recalibrated every so often in order to prevent overcurrent in the motor.

7.2.2 Speed Test:

In this test the KBRG-212D was connected as seen in Figure 5.1.1a. In addition, the LED apparatus frame was attached to the motor so as to get an accurate speed assessment of the motor when rotating. The potentiometer was connected in the forward setting as shown in figure 5.1.2a. The KBRG-212D was then turned on and the potentiometer was raised from its lowest setting to its highest setting slowly and back down again. Then the sensor circuit was turned on and the microcontroller was set to count the number of hits in one second and store the value, repeating this three times and then averaging the rps.

The desired result was to obtain an rps value between 25-30 rps and to be able to control the motor speed from 0-100% of its rated value. This turned out to be false on both accounts. The tests revealed that we could only reach between 15-20 rps without sacrificing torque capabilities. The second issue we found is that the potentiometer actually ranged between 0%-150% of the rated value. This meant that going too high on the potentiometer caused overcurrent in the motor.

7.2.3 Enable/Disable Switch Test:

In this test the KBRG-212D was connected as seen in Figure 5.1.1a. The potentiometer was connected in the forward setting as shown in figure 5.1.2a and the enable line was connected as seen in Figure 5.1.2b. The KBRG-212D was then turned on and the potentiometer raised to its expected running speed and left to maintain this speed. The Enable line was first set to on and then flipped off and left off for a few seconds before being turned on again. Before and after these transitions the speed calculation code was run again to determine the speed of the motor.

The desired result was to ensure that the rps was maintained when the enable/disable switch was toggled. In each test the switch effectively maintained its original rps value before the transition. This meant that we could set the rps value of the motor and leave the potentiometer alone and just use the enable/disable switch to control the off/on properties of the motor unless we needed more or less rps. As per the recommendation of the KBRG-212D user manual this enable line should NOT be used as an emergency turn off switch.

7.3 Slip Ring Test:

It can't be stated enough the importance of getting the slip ring to work for this specific project. Without proper power management the rotating side would be unable to do anything we had desired it to do. This is why we came up with a few tests to insure that the slip ring design would work under our desired conditions. The first test was nothing more than a durability test of the slip ring to determine if it could handle both the electrical and physical demands of the device. The second test was the power transfer test; it was to determine that under the most ideal of conditions that power is at least properly transferred through the slip ring.

The final test was to determine if the slip ring can transfer power during rotation and how much power loss is suffered due to thermal dissipation in the junction.

7.3.1 Slip Ring Durability Test:

This test required a completed motor and motor control circuit so that the motor's speed could be varied. This meant that we had to connect the control elements and control inputs discussed in the motor control section. Thus the Dayton motor and the KBRG-212D were both used in addition to the slip ring. In essence, the slip ring was attached to just the stranded wire with no power transfer. Then the motor's revolutions per second were slowly increased and then maintained at its maximum rotations. After a number of minutes had passed the motor's revolutions per second were slowly decreased and then the motor was shut down. After the device had been powered down the slip ring was checked for damage.

The desire of this test was to have the slip ring capable of handling the maximum possible revolutions per second that the motor could obtain and any variations in this rotational speed. The test results proved just that with no damage to the slip ring from the stranded wire.

7.3.2 Ideal Power Transfer Test:

This test required only the power supply, the slip ring shown, and a voltmeter. The slip ring was connected to the power supply on the outside, and the voltmeter was connected to the wire of the slip ring that was expected to be threaded through the shaft of the LED apparatus. The power supplies voltage was varied from a low to high DC value while the amount on the voltmeter side was recorded.

The desire of this test was to see that all or a majority of the power applied to the non-rotating side was seen on the side that would be rotating. This test proved that there was almost exact power transfer through the slip ring when not being rotated with only a very small resistive loss.

7.3.3 Rotational Power Transfer and Thermal Dissipation Test:

For this test we needed the motor control circuit, the motor, the slip ring design and the prototype circuit in Figure 6.1 of the prototype section. In this test the slip ring was connected to the power supply. Since we don't have a way to directly measure the rotational side during its rotation we used this prototype circuit to get an idea of how much power was being supplied to the rotational side. To begin we started with a power input of around 55 watts DC and begin rotating the device with the prototype circuit connected to the rotational portion of the device. We then gradually increased the DC power supply until we got the light to just turn on and recorded this value. Since the bulb required 60 watts to power the

difference between the 60 watts and what it took to power the bulb should be close to the dissipation through the slip ring.

The desire of this test is to have the slip ring handle the physical and electrical demands of the device with minimal power loss. The result of the test showed that the slip ring could handle turning on the light bulb at the exact watt requirements being sent over. So in other words 60 watts sent over had no dissipation, or so minimal that it was not enough to prevent the bulb from lighting. What was learned however, from this test was that the slip ring had contacting issues and would cause the bulb to flicker on and off a little. This required us to change our wire connection design from being just a straight contact to being tied around the shaft creating a hoop around the shaft that the shaft scrapped against.

7.4 Wi-Fi Communication Testing:

The Wi-Fi communication testing verified that the PC is able to communicate with the microcontroller successfully. This testing required us to have a simple server program on the microcontroller, and a client program on the PC. These testing procedures were done with the microcontroller and the PC concurrently since in order to test one we must communicate with the other. We tested the effectiveness of our protocol and made adjustments to our design as necessary. The main focus for the Wi-Fi communications was reliability since dropped packets may mean artifacts appearing in the image.

Our first test consisted of a simple TCP echo server over Wi-Fi. The code for this test came from the example code provided with the Wi-Fi library for the Wi-Fi shield. In order to test this echo server we wrote a simple TCP echo client in Java. In the Java application we set up the IP address and port number according to the settings in the server. The first sign that the echo server was working correctly was the appearance of the “POV Display” SSID on the PC. We connected to the network before running the Java application. The Java application was designed to receive a text input from the user and then send the text through the network to be received on the server. The server receives the text from the client and sends the same text back to the client. The client then receives the text from the server and displays it. A working echo server should appear to display whatever is typed in which means that the information was sent successfully through the network and back again. The echo server and client programs worked correctly the first time we tried them which verified that our hardware was working properly.

7.5 Software Testing:

The prototype LED array has been used to test numerous aspects of the POV display while sitting stationary, and can be seen in Figure 7.4. These tests include LED manipulation, brightness levels, displaying various RGB colors, and updating the controller at the fastest frequencies possible. In addition to testing the capabilities of the LED controller itself, we also used to test programs designed to display entire images and text. The prototype was also involved in compatibility tests which were used to help integrate the use of Micro SD Memory, Wi-Fi, and Sensor Input when combined with the LED array.

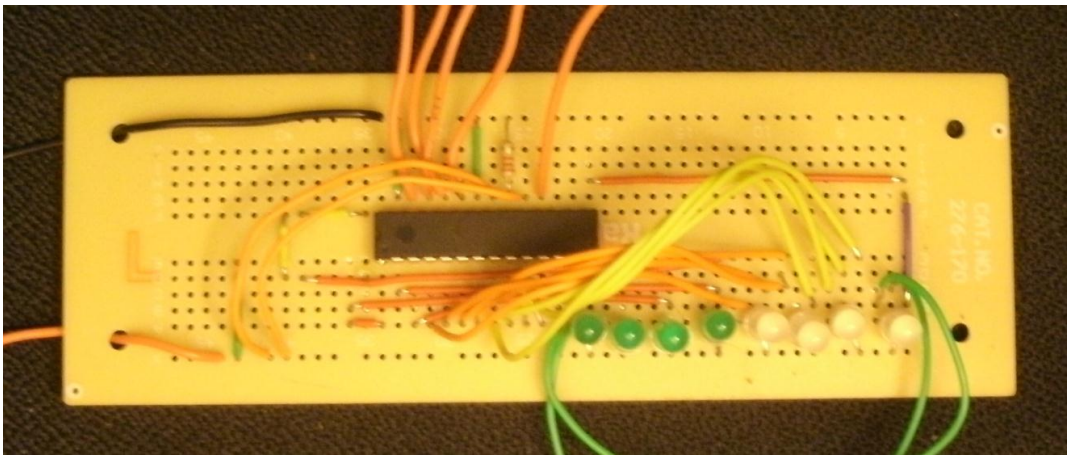


Figure 7.5 Prototype LED Array used in testing.

7.5.1 Prototype LED Array Capabilities Testing:

Basic Cycle Test: The basic cycle test was used as a baseline to determine that without any possibility of programming error, the hardware configuration was wired correctly and that the behavior of the LED array was as expected. In this test, each LED would be lit up one by one in sequence, with only one LED on at one time. After reaching the last LED, the first LED would be the next one to light and this would continue in a loop endlessly. A one second delay between each cycle was used causing the lit up LED to chance once per second.

Speed Cycle Test: This is a modification of the Basic Cycle Test. This test simply increases the speed at which the LEDs cycle by decreasing the delay. The delay value could be sent via Serial Monitor allowing for real time modification of the speed of the cycling. The Serial Monitor allows for communication with the program via USB interface.

Basic RGB Test: The prototype LED array included 4 RGB LEDs which were used to test displaying various composite colors. In this code, we manipulated only the first RGB led, which uses 3 inputs from the LED controller. The code

was designed to cycle very rapidly through 4096 colors values by using a triple for loop. Each RGB value was represented by 24 bits, which is considered true color and near the limit the human eye is capable of distinguishing two colors at. Each color with 24 bit color had 8 bits of color depth.

Timer/Output Compare Test: The purpose of this test was to determine the maximum frequencies that data could be written to the LED controllers. The PIC32 microcontroller contains 2 hardware timers that are available to be set, Timer2 and Timer3. Timer1 cannot be changed and operates at 80 MHz, the clock speed of the PIC32. In addition to the timers, 5 output compare modules are available and are used to generate the Blank, Latch, and GS Clock signals. The grayscale clock needs to operate as fast as possible so that the full brightness of each LED can be obtained very rapidly. Tests have indicated that at 20 MHz the microcontroller puts out a very clean signal with clear rising and falling edges. Additionally, tests attempting to obtain a 26.66 MHz frequency yielded inconsistent results. The signal would sometimes be useable, and sometimes become deformed, indicating that outputting this frequency is beyond the capabilities of the PIC32. The frequency used to test Blank and Latch were much smaller, requiring only a few thousand Hertz. All measurements were verified using an oscilloscope.

SPI Data Writing Speed Test: Similarly to testing the Grayscale Clock speed, we needed to obtain the fastest possible data writing speeds. The faster the data can be written the more often we would be able to update the display with a new column of data, and thus the higher frame rate we would be able to achieve. Data is being written using the SPI interface which is built into hardware on 4 pins, which include Data Out, Data In, Data Clock, and Slave Select. The SS pin can actually be any pin that you choose, but this is one which is hardwired to be one. By manipulating the SPI settings we were able to obtain a data writing frequency of 20 MHz with zero errors. Any attempt to increase this speed to 26.66 MHz failed to work properly. Frequency measurements verified with an oscilloscope

7.5.2 Micro SD Testing:

Basic SD Testing: MPIDE provides various example programs for accessing micro SD memory and the files contained within it. The built in functions allow you to open a file, read the contents within it byte by byte, as well as write bytes to the file. The library also includes functions for seeking to a specific byte location in the file. The library also provides functions for checking to see if a file exists, and when attempting to open a file that does not, it will create it. In this test, all of these functions are tested one at a time with serial print outs used to verify the success of each individual sub part of this function.

Additional SD Testing: In order to further test the behavior of each of the functions provided by the micro SD library, we performed additional tests

involving file creation and manipulation. The seeking function was given a more rigorous test by finding specific known data within a large text file. We also tested the behavior of the library when reading a non-text file, such as a binary file, which behaved exactly as expected.

7.5.3 Wi-Fi Communications Testing:

The Wi-Fi communication testing verified that the PC is able to communicate with the microcontroller successfully. This testing required us to have a simple server program on the microcontroller, and a client program on the PC. These testing procedures were done with the microcontroller and the PC concurrently since in order to test one we must communicate with the other. We tested the effectiveness of our protocol and made adjustments to our design as necessary. The main focus for the Wi-Fi communications was reliability since dropped packets may mean artifacts appearing in the image.

Our first test consisted of a simple TCP echo server over Wi-Fi. The code for this test came from the example code provided with the Wi-Fi library for the Wi-Fi shield. In order to test this echo server we wrote a simple TCP echo client in Java. In the Java application we set up the IP address and port number according to the settings in the server. The first sign that the echo server was working correctly was the appearance of the “POV Display” SSID on the PC. We connected to the network before running the Java application. The Java application was designed to receive a text input from the user and then send the text through the network to be received on the server. The server receives the text from the client and sends the same text back to the client. The client then receives the text from the server and displays it. A working echo server should appear to display whatever is typed in which means that the information was sent successfully through the network and back again. The echo server and client programs worked correctly the first time we tried them which verified that our hardware was working properly.

7.5.4 GUI Testing:

There are two basic main requirements that we will address: text message formatting, and communications. The formatting of the messages will focus on verifying that the user input is properly transformed into the proper data. Communications will be tested by connecting the computer to the microcontroller using Wi-Fi and verifying that the data is sent and received correctly. We will look at each of these features individually for testing. If each one is tested individually and verified to be working correctly, then the application as a whole will be considered complete.

The first test of the GUI involved seeing how the data would look like that is to be sent. This was done by organizing all of the data into an array and printing the contents to the console. No networking code was used for this test. The data printed to the console corresponded to the text contained in the text fields for line 1 and line 2. We performed this test for multiple text combinations and also for blank text boxes. We also verified that symbols and spaces had the correct output as well.

The final test for the GUI will be the integration test which involves sending text messages through the network. After adding the network code we use the TCP echo server on the microcontroller to act as a way to verify our data. The microcontroller receives the bytes of the text entered in lines 1 and 2, then echo's the text back. After this simple test we added more functionality by saving the text for line 1 into an array and the text for line 2 into a separate array. These arrays were then used for the echo back to verify that they contained the proper information. After this test was completed we were confident that our GUI worked properly and should send the proper text messages to the microcontroller. The next step from this point is integration testing including the microcontroller displaying the message that it receives.

7.5.5 Sensor Input Testing:

Hardware Interrupt Test: The IR sensor provides a high signal when it is tripped, and we wanted to use this signal to generate an interrupt in our program so that a flag could be set indicating that it is time to display. The PIC32 has several pins which can be used as hardware interrupt inputs, and ISRs can be created which monitor the status of that pin and allow for various modes of triggering. The supported modes by the PIC32 libraries are rising edge and falling edge triggering. We set up an ISR with rising edge triggering and simply printed out a statement to the serial monitor each time the sensor was triggered. We also tested that using a volatile int as a flag would work properly without creating any errors. The sensor worked flawlessly and provided hardware interrupts without any need for sensitivity calibration to the sensor.

Program Flow Test: In order to see the sensor influencing the program flow in some way, we created a test program where triggering the sensor would change the behavior of the LED array prototype. This test demonstrated that the sensor hardware interrupt was working correctly, and that the behavior of the display could be properly influenced by the triggering signal. The test code involved a onetime cycle through all of the LEDs with very short delay. Each time the sensor was triggered, the LED array would cycle rapidly through each of its 16 outputs.

7.5.6 Integration Testing:

LED Array and Micro SD Testing: This test was to determine if writing to the LED array and also reading from Micro SD Memory at the same time would cause any conflicts. Both devices are using the SPI interface for communications and share data lines, and data clock signals. This test involved simply reading and writing various strings to micro SD memory while executing the basic cycle code intermittently. The SD library properly handled sharing the SPI interface with the LED array. The SD library accomplishes this by storing the current SPI settings on a stack, loading in its own, and then restoring the previous settings once compete.

Wi-Fi and Micro SD Testing: We had many problems during the Wi-Fi server testing when including the SD card. At first the Wi-Fi server would work correctly, but when it came time to write to the SD card, it was unable to open a file. We were able to verify that the SD card initialized properly but for some reason a file could not be opened. At first we suspected that the order that the libraries were included may make a difference, so we tried including the SD library first. The order of the library includes did not seem to make a difference so after much online searching and troubleshooting we found some advice to initialize the SD card before initializing the Wi-Fi connection. Again this did not seem to make a difference. Our next guess was that the SPI bus was not being shared properly. Our searches for information online seemed to point out that the SD and Wi-Fi libraries are supposed to handle sharing the SPI bus on their own, but since we are unable to open a file while using the Wi-Fi echo server, we suspected otherwise. Our next attempt at fixing the problem included us finding out exactly which registers store the SPI bus settings so that we could make sure they did not get corrupted. We wrote functions to save and restore the SPI state which allowed us to ensure that the SPI settings were not being corrupted at any point. Unfortunately this also did not work. We were running out of ideas regarding why we were unable to open a file, but we knew that according to multiple forum postings the libraries were successfully working together without any problems. Since the SD card example program worked by writing to a file named “test.txt” we decided to match the file name for this integration test. Previously we were using the filename “smallImage.bin”, after switching the filename to “test.txt” the problem seemed to have been fixed. We were now somehow able to write and read from the SD card in conjunction with running a TCP echo server via Wi-Fi. This test, although extremely time consuming and frustrating showed us that the SD card library only supports a small file name length. We are now using SI.bin for the small image and LI.bin for the large image.

Wi-Fi and LED Array Testing: This was a test designed to find any compatibility issues in using the Wi-Fi library and LED array library simultaneously. Both libraries are using the SPI interface in order to communicate with the microcontroller. The test consisted of running the basic cycle code at the same time as a server is running and echoing the text sent to it. It was discovered during this test that the libraries conflicted and were not sharing the SPI interface properly. One cause of the problem was the use of Pin 10 by the LED array

library, as that pin is also used as the slave select line for the Wi-Fi chip. Because of this issue, we needed to move the blank signal required by the LED array off of pin 10 and onto a different pulse width modulation pin. This involved simply using a differed output compare module since each one is associated with a particular pin. Having moved the blank signal from pin 10 to pin 5, we resumed compatibility testing. A second conflict involving the SPI interface also existed, and this problem was related to the SPI settings themselves. Wi-Fi uses its own data clock setting, and our array uses a different data clock setting. Because of this, we implemented in the LED array library a way to restore its own SPI state in a similar way that the Micro SD library maintains its own state. Having added this restore SPI settings function to the LED array library, all issues were resolved and we moved onto implementing live update features via Wi-Fi.

Wi-Fi/SD/Array Testing: This integration test includes all devices that are sharing the SPI interface for communication. In this test we wanted to see that we could display on the LED array, receive data via Wi-Fi, and store it on the Micro SD Memory chip without any interference. In a previous test we added a restore SPI settings function to the LED array code, which allowed it to use the SPI when it wanted to, and let the Micro SD and Wi-Fi libraries handle restoring their own settings. We determined that data was being received accurately and was being written to the Micro SD Memory accurately, all while displaying the basic cycle code in addition to other LED control programs. We did find that while using the Wi-Fi communication junk data would occasionally be written into the LED array and cause a slight flickering. This issue was reduced somewhat by restricting the amount of Wi-Fi communication. In addition, the visual artifacts were minor and uncommon, which did not necessitate further debugging. I speculate that if we were to hold the blank signal high, driving all LED outputs off, during Wi-Fi updates, all visual artifacts would be eliminated.

Wi-Fi/SD/Array/Sensor Testing: This was the final integration test, which used Wi-Fi, Micro SD, LED array, and sensor input all simultaneously. The only additional feature in this test as opposed to the previous was the addition of the sensor. The sensor uses two pins from the microcontroller, once which provides a Vcc, and another on a pin which can generate a hardware interrupt when receiving input from an external device. Strangely, we found that the sensor was causing difficulties with the Wi-Fi initialization, and after a bit of research we discovered that the Wi-Fi module uses an assortment of pins in addition to the pins used for SPI communication for various reasons, and one that we had selected to use for the sensor was conflicting. By simply switching the sensor to pins that the Wi-Fi chip does not interact with, we were able to resolve this compatibility issue. At this point we were able to display test code which included sensor influenced code, while reading and writing from SD and receiving Wi-Fi updates, as well as displaying the basic cycle code on the LED array.

7.5.7 Feature Implementation Testing:

RGB Data from SD Test: The goal of this test was to determine whether or not an image which has been preprocessed on the computer and stored in SD is lighting the LEDs as expected. To determine this, a test image was created that was 16 LEDs tall and 384 wide. This test image is seen in figure 7.4.2. It contains a rainbow of colors as the start and end of the image, seconds of solid red, and sections of red and blue. In addition to this, the very last row of the image contains a rainbow of pixels going horizontally for a short section. This image was processed on the computer and loaded into SD memory. In the microcontroller, we then read the data from memory and output the data to the prototype LED array. This verified that the computer generated processed image was formatted correctly, and that our code for displaying an image from micro SD was also behaving correctly.

Figure 7.5.7a Test Image used for reading from Micro SD memory

Letter A Test: - This test was designed to test the code related to displaying a single letter of text. It was also a test of the persistence of vision effect on our display since this was the first shape displayed that had a predefined and expected form. This test did not include the sensor, Wi-Fi, or micro SD code and was a separate program. The letter A displayed in a verity of positions and moved around because of the lack of the sensor. This test verified that we would in fact need the sensor to act as a trip for when to start displaying. A second version of the letter A code was designed which implemented the sensor and it was at this point that we could see the first letter of text being clearly displayed and with minimal shifting from side to side. It was noticed that the spacing between columns of the letter seemed wider than desirable. We deduced that this was because of the processing time involved with updating the grayscale data array that is written out to the controllers. Because we were using a set function designed to set a specific 12 bit value at a specified index, a lot of calculations were taking place that were not necessary. We could instead just transverse the array one time, setting bytes to either the values 0xFF, 0xF0, 0x0F, and 0x00. This optimized code was designed specifically for being able to display text quicker and cause the displayed text to appear in higher resolution. After implemented the optimized code we could see improved clarity in the displayed text.

Column Test: In this test we have edited the ‘Letter A’ program to print two separate letters, one on top of the other, so that a column of letters would appear. This was a step towards being able to display multiple lines of text. The sensor was not initially implemented for this test and was added after the logic appeared to be correct. Having then implemented the sensor, we could see clearly two letters displayed, one below the other, and holding their position without moving side to side.

Word Test: - After successfully displaying a single letter on each line we decided to try displaying a word on each line. We stored the values corresponding to the font table entries in an array for each line. For line one the array contained data for the word “HELLO” and line 2 contained data for the word “WORLD”. When we ran this test the words successfully appeared but they were appearing in inconsistent locations and the letters were lowercase instead of the intended uppercase. After double checking the values in the font table and comparing them with the values we were reading, it was determined that we were sending the unmodified ASCII values. We changed to code to send modified ASCII values which consist of the original ASCII value less 32 to index the proper part of the font table. The letters also seemed to have a large gap between them but it was hard to see with the words displaying in a different position every time. We decided to include the hardware driven interrupt from the sensor in order to ensure that the words would display in the same location each rotation.

Word Test with Sensor Test: When we added the functionality of the sensor circuit the word test became much more legible. It also became clear that there were large spaces in between each letter which did not look very good. We tried optimizing the way that the microcontroller communicates with the LED controllers by directly manipulating that data that is sent to them. This new method of sending the data to the LED controllers replaced the functions we were using from the TLC library. We knew that the optimized code should run faster than the previous code but for some reason there were still large gaps between each letter. We eventually figured out that the reason for the gaps was due to serial communication being attempted. We had some serial print statements to help with debugging which would send serial data to the PC over USB. This allowed us to read messages and see where in the code the board was currently executing, but it also slowed down the speed of communication between the microcontroller and the LED controller. After removing the serial print outs we saw that the spaces between the letters was now gone and the image looked as intended.

Live Updating of Text: For our next test we wanted to be able to update the message that is displayed in real time while the device is turned on and spinning. For this we used Wi-Fi server code that was very similar to the echo server. The Wi-Fi code for this test does not echo back any sort of response, it simply receives data and stores that data into two byte arrays. We did not use the SD card for this test in an effort to keep it simpler. The message that the display shows defaults to “HELLO WORLD” with each word appearing on its own line. There are two byte arrays, one for line 1 and the other for line 2. These arrays are where the data for the default “HELLO WORLD” is read from. This is done the same way as described in the word test. The main reason for this test was to change what the display says by receiving data via Wi-Fi and saving that new data into each of the byte arrays for line 1 and line 2. Since the software is constantly displaying the message stored in the byte arrays, the message should

update if new data is stored in those arrays. After solving the problems of the previous test we were able to successfully complete this test quickly.

We had one major problem while performing this test which involved the Wi-Fi initialization to fail. The microcontroller would inconsistently initialize the Wi-Fi server, or not. If the Wi-Fi server didn't initialize we would sometimes be able to reset the board until it initializes. This behavior was unexpected and extremely hard to understand. We tried multiple different variations of initializing the TLC library before Wi-Fi, then after Wi-Fi. We also including the SD card libraries even though we weren't using the SD card in an effort to force the libraries to properly manage a shared SPI bus. We thought that without the SD card library included the state of the SPI bus may not be properly managed because the software may not think that the bus is shared. None of these things seemed to help and the reliability of the Wi-Fi initialization remained unpredictable. We eventually noticed that the server seemed to work better when we physically disconnected the RGB LED array from the microcontroller. It seemed that having the RGB LED's connected introduced some sort of interference which caused the microcontroller to behave unpredictably. This test finally became a success after we solved this problem, but this made a new problem obvious: we will not be able to send Wi-Fi updates to the RGB LED array if we cannot solve this problem. The result of our testing can be seen in Figure 7.5.7b. The display originally said "HELLO WORLD" and we were able to update it via Wi-Fi to read "UCF KNIGHTS".

Figure 7.5.7b: Text Display

Basic RGB Cycle Test: The basic RGB cycle test is a test very similar to the Basic Cycle Test used on the prototype display. It is a base line for determining if the RGB array has been correctly wired up and Is functioning properly. In this test, each color in each LED Is flashed sequentially until reaching the last LED, and then starting back over with the first LED.

Cascaded RGB Array Test: This test was intended to determine if any issues arise when cascading several RGB PCB modules together. Although each module already contains 3 LED controllers cascaded, this test verifies that the data correctly passes through the entire module, and that the share controlled signals also pass through. Additionally, this test was used to determine whether or not any problems would arise from having too many controllers cascaded and how it would affect the update rate of the display. With two modules attached together, data is being written to 6 LED controllers. Initially upon testing this configuration the grayscale clock signal and SPI data writing speed was not set high enough to accommodate 6 controllers. For the text array, operating at 5 MHz for each was sufficient, however after testing we determined that at least 10 MHz is required when updating 6 LED controllers. The code for this test is the basic RGB cycle code with the NUM_TLCS variable set to 6. After having fixed the timing issues, we successfully cycled through all of the LEDs in the combined array.

RGB Text: This test was designed to determine the capabilities of our RGB display and to show the functionality of the RGB array when operating at real time speeds. The test program is a modified version of the Word Test with Sensor, with some convenient array indexing multipliers and a counter to change the color between each letter. From this test we determined that the red LEDs are the most visually appealing and that though the blue and green LEDs are fainter, the words can still be clearly read by an observer.

Figure 7.5.7c: RGB Text Array displaying alternating color text.

Based on the previous results, we modified the test code to display the entire word phrase in all Red, Green, and Blue, and to cycle between each color for displaying the text. Each color displayed clearly, blue being the faintest and red the brightest. Figure X.X.X shows the word test displayed in solid red.

Figure 7.5.7d: RGB Text Array red letters

8 User Manual:

The following section will discuss the process of setting up and using the various hardware pieces outlined in the above design section along with the proper use of the GUI and how to program the microcontroller directly. Each sub section will deal with a specific hardware or software component of the device and outline how to effectively use it alone with safety warnings to prevent injury.

8.1 KBRG-212D Manual:

The KBRG-212D has a very thorough user manual that describes the various settings and features and how to use them. This section will discuss only the features and settings used for the purposes of this project. It is recommended that before use of the KBRG-212D you follow the current calibration process outlined in the testing section and the user manual of the KBRG-212D chip, especially after prolonged disuse of the chip and Dayton motor.

There are two major control inputs for the KBRG-212D for the purposes of this project. The first is the speed control dial featured in Figure 8.1 on the left. This dial controls the speed at which the motor is running at. Turning the dial counter-clockwise decreases the speed of the motor, while turning the dial clockwise increases the speed of the motor. The motor speed can be raised to the maximum point on the potentiometer, though this is not recommended due to overcurrent in the motor. It is recommended that if the OL light indicator turns on while increasing the motor that the speed of the motor be decreased until the OL indicator turns off so as to prevent damage to the motor.

Figure 8.1 KBRG-212D Input Switches

The second control input for the motor is the ENABLE switch featured in Figure 8.1 on the Right. The ENABLE switch is used to turn the motor off during reprogramming of the onboard microcontroller or start-up. It is recommended that the user turns the ENABLE switch off before turning AC power off and not re-activate the ENABLE switch until ready for the motor to actually start running. It is highly recommended that after setting the speed to the desired level the user does not excessively mess with the speed dial and only uses the Enable switch to control the operation of the Dayton motor. It should be noted that the KBRG-212D should be handled with care during operation since no portion of the KBRG-212D is isolated from the AC power.

8.2 Display Alignment Manual:

The display alignment sensor has four terminals used for operation. The Vcc requires a constant 5V DC that is connected directly to the Chipkit microprocessor's 5V pin. The CTRL line can be plugged into any of the output terminals of the Chipkit microprocessor and requires a 2.5V constant signal that is used to turn on the sensor, for our purposes we used the pin 8 terminal. The OUT terminal can be plugged into any input terminal on the Chipkit

microprocessor and will read a high value when the sensor receives a hit from passing over a reflective surface, for our purposes we used the pin 7 terminal. The GND terminal should be plugged into any of the GND terminals of the Chipkit microprocessor.

8.3 Power Supply Manual:

There are two AC adaptors and a 9 volt battery required for operation of the POV display. The First AC adaptor controls the KBRG-212D drive chip and also has a built in single pole single throw switch for purposes of turning off the motor and control chip, feature in Figure 8.3. This switch can be used to turn off and on power to the motor without pulling the AC adaptor, this switch should also be used as an emergency off switch instead of the ENABLE switch outlined in the KBRG-212D manual.

The second adaptor is an AC to DC variable adaptor and should be set to its lowest setting 3.3V. The variable adaptor is designed such that if the user decided to increase the number of RGB array modules they could also increase the power required to power them. The final power supplier is a 9 volt battery with a DC adaptor for plugging into the regulated port of the Chipkit microprocessor. This is stationed on the rotating side and when replace should be fashioned to the armature with the supplied zip ties. The 9 volt battery should be replaced after about an hour to two hours use.

Figure 8.2 KBRG-212D Power Switch

8.4 GUI User Manual:

When executing the GUI the first screen that appears is the text message tab. This is the screen that will allow you to send text messages to the POV display in real time. The text message tab can be seen in Figure 8.4.

The first option is “Display”, this option lets you choose which LED array to display the text on. The text display is the only option that is currently implemented in this version. The next two options are “Color”, and “Animation”. Color cannot be changed when the text display is selected since the text display only uses green mono color LED’s.

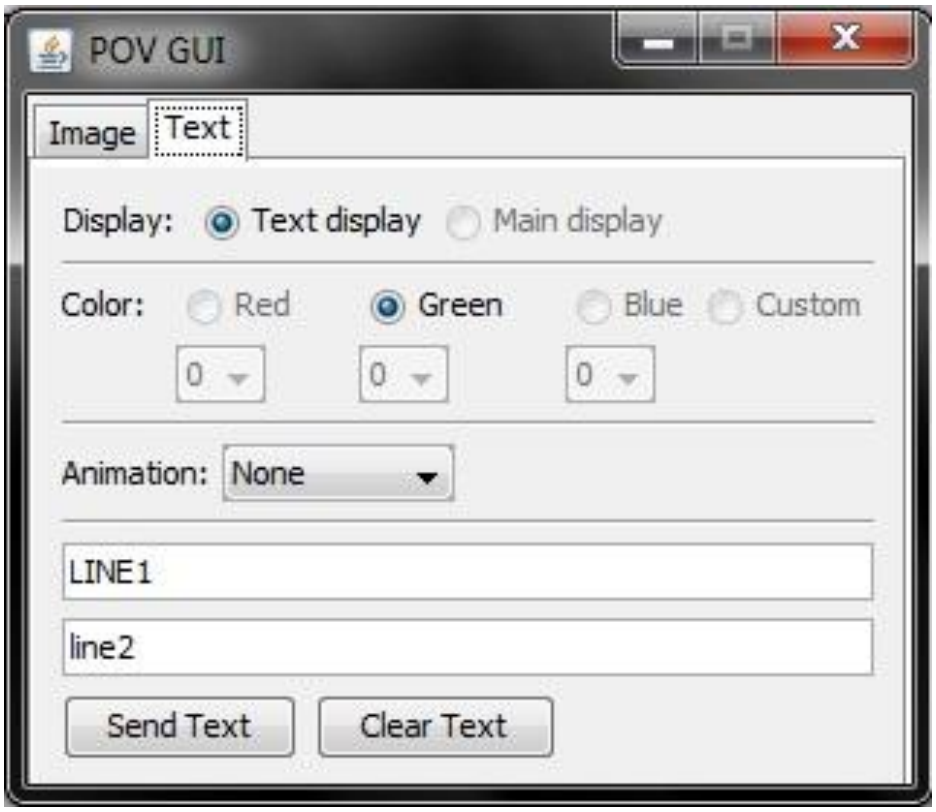


Figure 8.4 POV GUI Text Message Tab

The Animation option is not fully implemented and may not work as intended. The text boxes which say “LINE1” and “line2” are where you should enter the text that you wish to send to the display. If you wish there to be a blank line, simply delete the text from the text box before sending the message. After you are done editing the contents of the two text boxes, pressing the “Send Text” button will wirelessly send the data to the POV display. The last button available is the

“Clear Text” button, when this button is pressed it is the same as sending two blank lines of text to the POV display, which means nothing will be displayed. If sending a text message does not seem to be received by the POV display, ensure that the PC running the GUI application is connected to the “POV Display” wireless network, and that you are using the correct WEP key. The “Image” tab was designed for sending an image to be displayed on the color main display, but this has not been implemented since the main display cannot support Wi-Fi communications.

8.5 Switching Between RG and Text Array:

Our POV display has two hardware configurations for displaying either from the Text array or from the RGB array. In order to switch between these there are 5 lines coming from the microcontroller that will need to be changed. The pin configuration from the controller is as follows:

Pin 3 – Grayscale Clock

Pin 5 – Blank signal

Pin 9 – Latch signal

Pin 11 – Sin (Data out)

Pin 13 – Data Clock

To operate in RGB mode, the text array should be entirely disconnected from the controller. This includes removing its ground and Vcc lines. The RGB display should then be wired to the controller for the 5 pinouts that have been specified. Additionally, the ground line for the RGB display will need to be connected to the microcontroller. The wall AC adapter must then be plugged in so that power can be transferred through the slip ring.

When operating in Text display mode, the RGB display should be completely disconnected from the microcontroller. This includes insuring that its ground line is also removed, as it will affect the operation of the device. The 5 lines coming from Text array should then be wired into the microcontroller based on the pin out specified above. The Vcc and ground for the text array must also be wired into the microcontroller.

9 Conclusion:

The process of designing a persistence of vision device turned out to be a far more complicated endeavor then our team expected. While we had already expected some complications in the power transmission process of this device a whole slew of issues revealed themselves in other areas of the device that we had initially thought to be simplistic. The process of choosing a motor and controlling it seemed at first to be a simple idea but when we began to research further into the process it turned out to be far more complex than expected, specifically for the high rotational and torque requirements of our system. Eventually we came to the design presented which accomplishes our goals for this. However, this design did not hold over well since, while the pulse width modulation circuit did effectively control the motor, powering the motor and controlling it together became a substantial problem. The 90 volts and 1.5 A was difficult to power on a low budget and ended up forcing us to scrap our original design for a more integrated approach which resulted in the KBRG-212D chip being the method to power and control the motor. This design change had drawl backs as we were forced into a more electromechanical method to control the motor instead of the original desire of having the motor controlled digitally with your computer.

However, motor control was not the only unexpected challenge. The design of the LED array turned into a rigorous design challenge when it turned out that trying to address each and every LED would send our data transfer rates into the nine digit figures. Which the Chipkit board was capable of handling only a fraction of the LEDs we originally intended to have so we ended up having to significantly scale back on our LED array design. This was not the only issue, though because soon after acquiring the arrays we ended up with another issue with the Chipkit, the frequency it could obtain. While the Chipkit theoretically could obtain the frequency we needed to run our pixel count, it ended up only being able to run 32 instead of the first attempt at scaling back which was 128.

These design challenges discussed above were overcome but at a substantial increase in our first projected costs. This meant that the need for sponsorship has tremendously increased. The entire design was under the expectation of an almost limitless budget. However the loss of the expected sponsorship required some rather extreme reductions in scale of the design. Specifically our team had to drop the HDMI instantaneous streaming of the display device. This was primarily for two reasons. The first reason was that the loss of a sponsor required us to drop the LED count and thus drop the resolution to a level that would not be cohesive with the idea of displaying a computer screen for video playback. The second reason for this design cut was the ability to purchase less powerful and thus less expensive micro processing boards for image processing. Without the demands of the high data transfer associated with the instantaneous streaming of the display device our display ended up having to project much more simplistic

animations and text, thus needed much less data transfer and processing. We were unable to cut our motor demands however, but we were still able to find a far less expensive motor then the one outlined in the design. However, with these cuts it did not spell the end for scalability of the device. Since even with the reduction in hardware features there are still a vast amount of room for software features to more than make up for the loss of the instantaneous streaming of the display device.

For the future of this device there are a variety of possible additions that can be added to it. First the Driver chip can be modified to include a digital way to control it, allowing for integration of a Wi-Fi chip on the stationary side also. This would allow the user to turn the device on and off, and even vary the speed of the motor all from the same program that is controlling the image processing. The LED arrays are designed such that with a more powerful power supply and microprocessor more arrays can be added to create a larger device with better resolution. The processor could also be programed with additional functionality.

10 Bill of Materials:

As seen in Table 10, is a list of major items required to build the POV display. Some components with a "-" mark for part number were bought in bulk variety packs and thus did not have any sort of Part number. In addition, some materials were marked with "-" quantity did not have unit prices and instead were bought in bulk packaging, thus a quantity used could not be established.

Bill of Materials				
Item Number	Part Number	Mfr.	Description	Qty
Microprocessor				
101	uc32	Chipkit	Microprocessor, 80 MHz, 5V, 512K Flash, 32K SRAM	1
Motor and Chassis				
201	9FHD7	Dayton	Motor, DC, Permanent Magnet, 90 Volts, 1.5A, 0.49 N·m Torque, 1800 RPMs.	1
202	KBRG-212D	KB	Regenerative Drive Chip, 115/230 AC, Permanent Magnet or Field Wound Motors	1
203	8090T13	McMaster-Carr	Bearing, Extended-Ring Type ER, rated for 3,145 dynamic load pounds and 5,000 RPMs	1
204	Custom Metal Work	KEMCO Industries	Custom Aluminum Metal to include top plate, base plate	1

			and support rods.	
LED Array				
301	OVS-3309	Multicomp	LED, Type OVS, RGB, SMD	48
302	TLC5940	TI	LED Controller, 16-Channel	7
303	TLC5940NT	TI	LED Controller, 16-Channel	2
304	-	Multicomp	LED, Green	20
Components				
401	-	Radio Shack	Resistor, 1 kΩ	4
402	-	Radio Shack	Resistor, 300 Ω	1
403	-	Radio Shack	Resistor, 3.8 kΩ	1
404	-	Radio Shack	Resistor, 50 Ω	1
405	-	Radio Shack	Trimmer, 10 kΩ	1
406	-	Radio Shack	LED, Green	1
407	-	Radio Shack	Resistor, 2.2 kΩ	6
408	-	Vishay	Resistor, 100 Ω	144
409	-	Radio Shack	Trimmer, 5 kΩ	1
410	-	Radio Shack	Switch, SPST	1
411	-	Radio Shack	LED, Infrared	2
Miscellaneous Equipment				
501	-	enercell	Adapter, AC to DC, Variable: 3.3-7.5V	1
502	-	Kintron	Adaptor, AC, SPST Switch on/off	1
503	-	enercell	Battery, 9V	1
504	-	Radio Shack	DC Plug	1
505	-	Home Depot	Zip Ties	-
506	-	Home Depot	Metal Clamps, medium	2
507	-	Home Depot	Aluminum Plate, Small	1
508	-	Home	Aluminum Plate, Medium	1

		Depot		
509	-	Radio Shack	Electrical Tape	-
510	-	Radio Shack	Wire, Stranded, 18 gauge	-
511	-	Radio Shack	Wire, Solid, 18-22 gauge	-
512	-	Radio Shack	Aluminum Sheet Metal	-
513	-	Home Depot	Permanent Double-sided tape	-
514	-	Home Depot	Wood Plank	-
515	-	Home Depot	Lugs	-
516	-	Home Depot	Copper Pipe 1.5” Diameter	1

Table 10: Bill of Materials

11 Appendix:

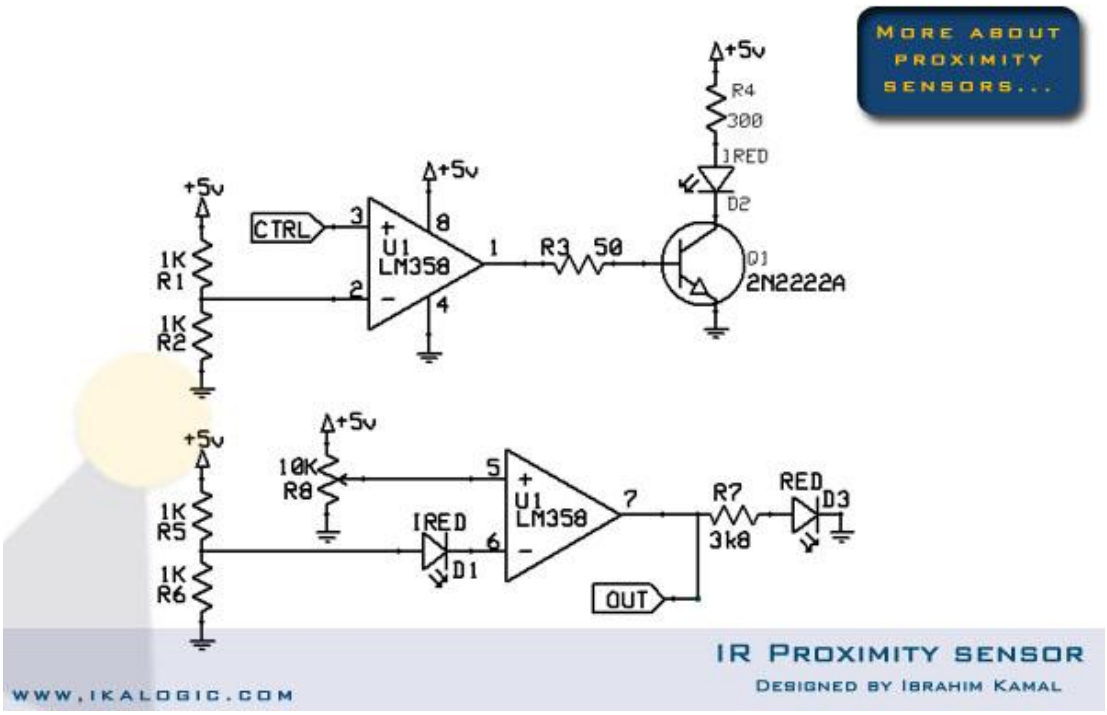


Figure 10.a Infrared Sensor Reference Circuit

12 Bibliography:

"802.11 Wireless Standards." About.com. Web. <http://compnetworking.about.com/od/wireless80211/80211_Wireless_Standards.htm>.

"Arduino - Ethernet." Arduino.cc. Web. <http://arduino.cc/en/Reference/Ethernet>.

"BIT DEPTH TUTORIAL." Cambridgeincolour.com. Web. <http://www.cambridgeincolour.com/tutorials/bit-depth.htm>.

"Bluetooth." Wikipedia. Wikimedia Foundation, 30 July 2012. Web. <http://en.wikipedia.org/wiki/Bluetooth>.

"BufferedImage (Java 2 Platform SE V1.4.2)." Docs.oracle.com. Web. <http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/image/BufferedImage.html>.

"Color Depth." Wikipedia. Wikimedia Foundation, 08 Jan. 2012. Web. <http://en.wikipedia.org/wiki/Color_depth>.

"A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi." Http://eee.guc.edu.eg. Web. <http://eee.guc.edu.eg/Announcements/Comparaitive_Wireless_Standards.pdf>.

"Digilent Inc. - Atlys Spartan 6." Digilent Inc. Web. <http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,836>.

"Digilent Inc. - VMOD-BB." Digilent Inc. Web.
<http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,648,847>.

"Effects of Mobile Rotational Movements in Wireless Propagation Channels." Http://ieeexplore.ieee.org. Oct. 2008. Web.
<http://ieeexplore.ieee.org/xpl/login.jsp?reload=true&tp=&arnumber=4635902&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Fisnumber%3D4635901%26arnumber%3D4635902>.

"HANDBOOK OF WIRELESS NETWORKS AND MOBILE COMPUTING." Http://www.nettech.in. Web.
<http://www.easybib.com/cite/edit/134383789490cd7ed2-3cfb-42ec-8f8c-d156a560a67a>.

"How Bluetooth Works." HowStuffWorks. Web.
<http://www.howstuffworks.com/bluetooth.htm>.

"How WiFi Works." HowStuffWorks. Web.
<http://computer.howstuffworks.com/wireless-network.htm>.

"IEEE 802.11." Wikipedia. Wikimedia Foundation, 31 July 2012. Web.
<http://en.wikipedia.org/wiki/IEEE_802.11>.

"ImageIO (Java 2 Platform SE V1.4.2)." Docs.oracle.com. Web.
<http://docs.oracle.com/javase/1.4.2/docs/api/javax/imageio/ImageIO.html>.

"ImageReader (Java 2 Platform SE V1.4.2)." Docs.oracle.com. Web.
<http://docs.oracle.com/javase/1.4.2/docs/api/javax/imageio/ImageReader.html>.

Jeffay, Kevin. "Coding and Compression Basics." Http://www.cs.odu.edu. Web. <http://www.cs.odu.edu/~cs778/jeffay/Lecture3.pdf>.

"The New Wi-Fi Protocol." Suite101.com. Web. <http://suite101.com/article/wifi-protocols-a42024>.

"NXVGA." Digilent Inc. 9 Nov. 2006. Web. <http://www.digilentinc.com/Data/Products/NXVGA/NXVGA_rm.pdf>.

"RGB Video Out." Eecg.toronto.edu. Web. <http://www.eecg.toronto.edu/~tm4/rgbout.html>.

"A Robotic Wireless and Sensor Network Testbed." Cs.utah.edu. Web. <http://www.cs.utah.edu/flux/papers/robots-infocom06.pdf>.

"Serial Port on Atlys." Danielbit.com. Web. <http://www.danielbit.com/blog/microblaze/serial-port-on-atlys>.

"Serial Programming/Serial Java." En.wikibooks.org. Web. <https://en.wikibooks.org/wiki/Serial_Programming/Serial_Java>.

"VGA Video." MIT.edu. Web. <http://web.mit.edu/6.111/www/s2004/NEWKIT/vga.shtml>.

"VGA Video Signal Format and Timing Specifications." Javier Valcarce's Personal Website. Web. <http://www.javiervalcarce.eu/wiki/VGA_Video_Signal_Format_and_Timing_Specifications>.

"VGA." Wikipedia. Wikimedia Foundation, 24 July 2012. Web. <http://en.wikipedia.org/wiki/VGA>.

Westrelin, Roland. "TCP and Real-time." Blogs.oracle.com. Web. <https://blogs.oracle.com/roland/entry/tcp_and_real_time>.

"Wi-Fi: The Most Commonly Used Wireless Technology." About.com. Web. <http://voip.about.com/od/mobilevoip/p/wifi.htm>.

"Wi-Fi." Wikipedia. Wikimedia Foundation, 08 Jan. 2012. Web. <http://en.wikipedia.org/wiki/Wi-Fi>.

"WIRELESS NETWORK COMMUNICATIONS OVERVIEW FOR SPACE MISSION OPERATIONS." Public.ccsds.org. Web. <http://public.ccsds.org/publications/archive/880x0g1.pdf>.

"Wireless Sensors on Rotating Structures: Performance Evaluation and Radio Link Characterization." Http://dl.acm.org. Web. <http://dl.acm.org/citation.cfm?id=1287770>.

"AC Motor." *Wikipedia*. Wikimedia Foundation, 30 July 2012. Web. 01 Aug. 2012. <http://en.wikipedia.org/wiki/AC_motor>.

"MICROMO." *Micro Drive Systems- Brushless, Coreless & Linear DC Motors*. N.p., n.d. Web. 01 Aug. 2012. <http://www.micromo.com/>.

"DC Motor." *Wikipedia*. Wikimedia Foundation, 24 July 2012. Web. 01 Aug. 2012. <http://en.wikipedia.org/wiki/DC_motor>.

"DC Motor Calculations, Part 1." - *Developer Zone*. N.p., n.d. Web. 01 Aug. 2012. <http://zone.ni.com/devzone/cda/ph/p/id/46>.

"Go Green, Go Electric." *DC Motor Speed Controller PWM 0-100% 400Hz-3khz Freq*., N.p., n.d. Web. 01 Aug. 2012.

<http://www.masinaelectrica.com/dc-motor-speed-controller-pwm-0-100-400hz-3khz-freq/>.

"Passive Infrared Sensor." Wikipedia. Wikimedia Foundation, 30 July 2012. Web. 01 Aug. 2012. <http://en.wikipedia.org/wiki/Passive_infrared_sensor>.

"IKA-TACH." *IKALOGIC*. N.p., n.d. Web. 01 Aug. 2012. <http://www.ikalogic.com/ika-tach/>.

"99 000 RPM Contact-Less Digital Tachometer." *IKALOGIC*. N.p., n.d. Web. 01 Aug. 2012. <http://www.ikalogic.com/99-000-rpm-contact-less-digital-tachometer/>.

"Infra-Red Proximity Sensor Part 1." *IKALOGIC*. N.p., n.d. Web. 01 Aug. 2012. <http://ikalogic.cluster006.ovh.net/infra-red-proximity-sensor-part-1/>.

"Carl Pisaturo - Electrical Notes: Slip Rings." *Carl Pisaturo - Electrical Notes: Slip Rings*. N.p., n.d. Web. 01 Aug. 2012. <http://www.carlpisaturo.com/_ElNo_SLIP.html>.

"Lighting and Display Solutions." TLC5940. *Texas Instruments.* N.p., n.d. Web. 01 Aug. 2012. <http://www.ti.com/product/tlc5940>.

"Lighting and Display Solutions." TLC5971. *Texas Instruments.* N.p., n.d. Web. 01 Aug. 2012. <http://www.ti.com/product/tlc5971>.

"Low Cost Slip Ring." *Model 205. Mercotac.*

N.p., n.d. Web. 01 Aug. 2012.
<http://www.mercotac.com/html/205.html>.