University of Central Florida

Department of Electrical Engineering and Computer Science

3D Persistence of Vision Display

Group 8
Senior Design I 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 phenomena 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 planned on implementing.

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 groups 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 advisement and these displays require attention getting gimmicks out 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:

One of our group members brought up the thought that if we can make this device not only display images and do simple animations then why can't we make the device also display a video such as from a DVD or CD played from your computer. Since we wanted the project to be sponsored allowing the eventual sponsor to display a prerecorded advertisement on this device from such a media, seemed like a great idea. So our group set out to implement it through computer interfacing. We chose an HDMI connection between the computer and the POV device since this would create an almost monitor like relationship with the POV device and the computer. While this was the largest challenge for this project it also seemed like its biggest feature.

2.2.3 High Resolution:

Since we wanted to create the above mention monitor relationship between the computer and the POV device we needed to design the project with a high pixel count in order to effectively display the computer's desktop or HUD. We specifically chose 640x480 as our target resolution since it seemed to be a relatively universal one. This meant we needed a total of 480 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.

480 LEDs
256 colors per LED
60 Hz refresh rate for LEDs
30 rps or 1800 rpm
61 cm diameter (cylinder)
80 cm height (cylinder)
12 - 15 lbs
Operates on 120V AC or 90V DC
HDMI input via computer connection
USB input via computer connection
2Mbits/s data transmission
Data Rates up to 800 Mb/s (12.8 GB/s peak bandwidth)
Up to 1,080 Mb/s data transfer rate per differential I/O
128 megabits of storage space for images and animations.

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 will be to layout an administrative plan to govern and guide our project through the varies stages and will be the foundation that will support our work. Our administrative plan will be 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. To better understand the cost associated with our senior design project we will need to estimate our total material cost and any cost associated with prototyping and testing.

As stated in our specifications section, the POV display will require 480 LED's each with a 256 color range. This will require us to procure 480 RGB capably LED's which we can estimate to be about $1.25 each. In addition to the LED's, we will also need to procure some way to control the LED's. If we estimated an additional $1.00 per LED to control it we have a total cost per LED of $2.25 or $1,080.

The POV display must be capable of rotating at 30 RPS or 1800 RPM. This will require a sizable motor to insure we can operate at the required toque values. As well, we will require a chassis or frame to support the LED array and on board controllers. We can estimate the cost of the motor and chassis frame at $80.00

In order to process the incoming video signals and control the LED's we will require two microcontrollers. One microcontroller, which we can refer to as the on board controller, will process the video signal coming from the main microcontroller. The on board controller will require less functionality and can be estimated at a cost of $250. The main controller will require additional functionality to interface with the user. We will estimate the cost of the main controller at $300. Therefore, the overall cost for both the on board and main controller will be $550.

In order to tie the on board controller to the LED array we will need to procure PCB boards. We will also need to procure additional wire and cable to make all the miscellaneous connections required. For estimating purposes we will lump all the cost associated with procuring PCB boards and wire as required will cost $100 but not to exceed to $200.

The final piece of the budget will be any cost associated with prototyping and testing. Currently we are anticipating we will require a scaled LED array, motor control and slip ring prototypes. If we estimated approximately 1:60 scaled LED array or about 8 LED's and using the previous estimate of $2.25 per LED the cost for the LED array prototype is about $18. The slip ring and motor control we can estimate about $25 each in cost for prototyping. That brings the total estimated cost for prototyping to $68 but not to exceed $150.

The total cost associated with this project is estimated to be $1,878 but not to exceed a total cost of $2,060. As seen in Table 3.1, a detailed summary of the budget and distributed cost can be complied. Based on this estimated, we have determined that the cost associated with this project is realistic and feasible.

Description	Est. Qty Req.	Cost (Each)	Ext. Price
LED's and Controller	480	$2.25	$1,080.00
Motor and Frame	1	$80.00	$80.00
On Board Controller	1	$250.00	$250.00
Main Controller	1	$300.00	$300.00
Misc. Equipment	1	$100.00 - $200.00	$100.00 - $200.00
Prototyping	1	$68.00 - $150.00	$68.00 - $150.00
		Total:	$1,878 to 2,060

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. As discussed in our motivation for working on this project, we are interested in finding sponsorship to support our work. Currently we have two potential sponsors. The first and preferred sponsorship will come from the United States Navy. The idea of the POV display grew from the Navy's desire for a device to aid in their recruitment of new cadets. As our sponsors, the Navy will acquire the POV display after the completion of fabrication, testing and final review by the University of Central Florida to take between various recruitment opportunities.

If the U.S. Navy elects to sponsor another project, the second potential sponsorship will come from the University of Central Florida's Department of Electrical Engineering and Computer Science or Department of EECS. If the Department of EECS sponsors the design and development of the POV display, the application of the display will remain vastly the same. The Department would use the POV display to showcase the Department of EECS and the capabilities their students possess.

The final, and least preferred option, would be to split the cost of the project equally between all four group members. In the event that we do not acquire any sponsorship, the project design maybe be scaled back slightly to account for the smaller budget we will be working worth but the overall functionality and operation of the POV display will remain the same. The ownership of the POV display after completion of the project will be up to the group. One potential outcome would be to donate the POV display to the University of Central Florida's Department of Electrical Engineering and Computer Science. Another potential outcome of ownership could be one ground member takes ownership of the POV display by means of voting or by means of additional finical responsibility beyond the other ground members.

3.3 Schedule and Milestones:

The final portion of our administrative plan will be to develop a schedule, which will include major and minor milestones, to be a guide for keeping the production of the project on time and finished by the due date. Before a schedule can be developed, the major and minor milestones of the project must be defined.

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 can be 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 will require a source of power to operate. As discussed in the motivation, the POV display will need 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 will focus on utilizing power from an AC outlet.

4.1.1 AC Input:

As previously stated, the POV display will 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 will require research is circuit protection. Since we will 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 criteria will be to protect the POV display from potential damaged cause 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 will 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 will be protecting the incoming AC power, only the positive or line side of the AC power supply needs protection. This means we will 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 will require 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, will be used.

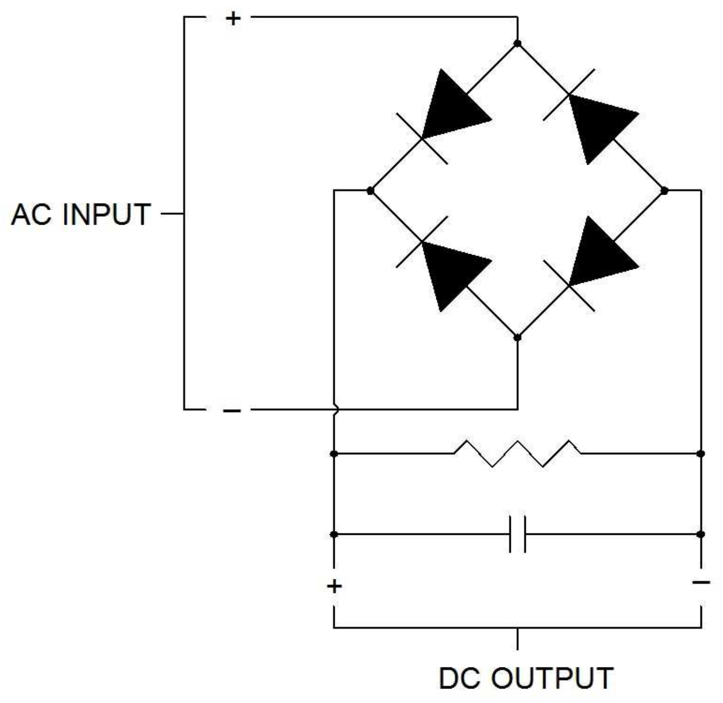


Figure 4.1.2 Full Wave Rectifier Circuit

Although the exact voltage required for the motor, LED array and microprocessors is not known at this time, we do know that we will 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 will vary the AC input by using a simple voltage divider circuit with a potentiometer or variable resister. Therefore we will focus our research on determining what variety of parts are available and their characteristics. In particular, we will be researching for 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, will be 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 will need to operate at a maximum of 150 volts and 1500 watts. This design criteria will allow 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 design 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 will still require 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 will work well the bread boards and PCB boards. Although the exact resistor requirements are not known at this time, 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 will use 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 intend 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 will be used to determine which format will be most appropriate for our needs and what would be required to use that format. This section will also look at various means of video data compression and alteration.

4.2.1 VGA

We are considering 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.

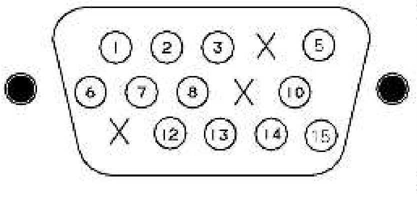
4.2.1.1 VGA Signal Standards:

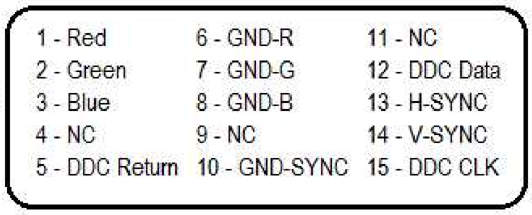
In order for a computer to know what types of signal a display can handle, the computer communicates with the display through the Data Display Channel. The protocol used most commonly today is E-EDID, which has been defined by the organization VESA. With the E-EDID protocol, the computer reads a binary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

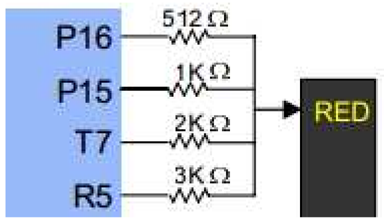


Figure 4.2.1.2.b Resistor circuit providing 16 colors from 4 inputs

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.

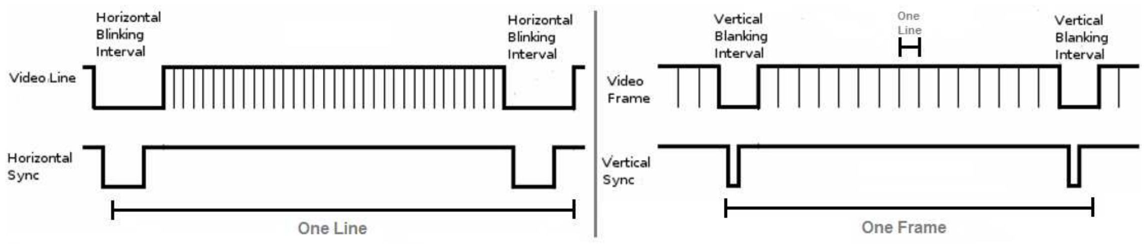
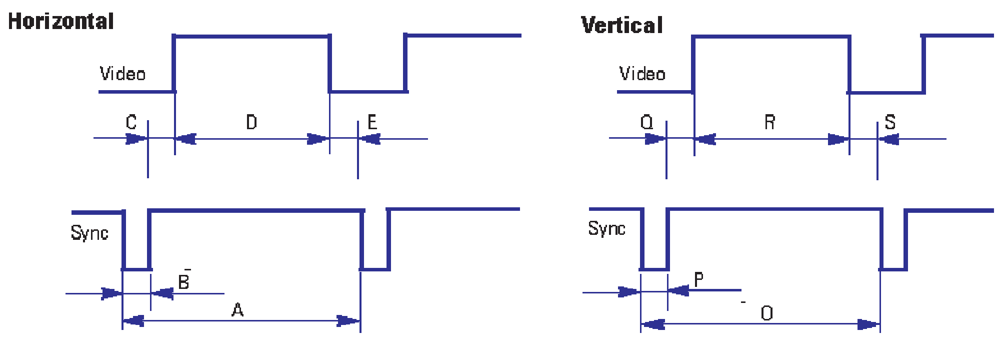

Horizontal Blinking Interval
Horizontal Blinking Interval
Video Line
Horizontal Sync
One Line
Vertical Blanking Interval
One Line
Vertical Blanking Interval
Video Frame
Vertical Sync
One Frame

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	M	25.17	28.32	31.5	36	49.5	56.25	78.75	94.5

Clock	Hz	5	2						
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 will need 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 will allow 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 will be 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 will need 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 are considering supporting in our POV display project. HDMI input will allow us to receive a signal in a format that is quickly gaining popularity and is currently available on many devices. The main reason we are considering 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 research is mainly focused on how we would go about receiving the HDMI signal on the Digilent Atlys board and then translate 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 is therefore going to be the HDMI sink and any device connected to our display will be 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 will most likely not be taking 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 are creating 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 prove beneficial to use since it may eliminate the need for downscaling the resolution of the input since the file will 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.

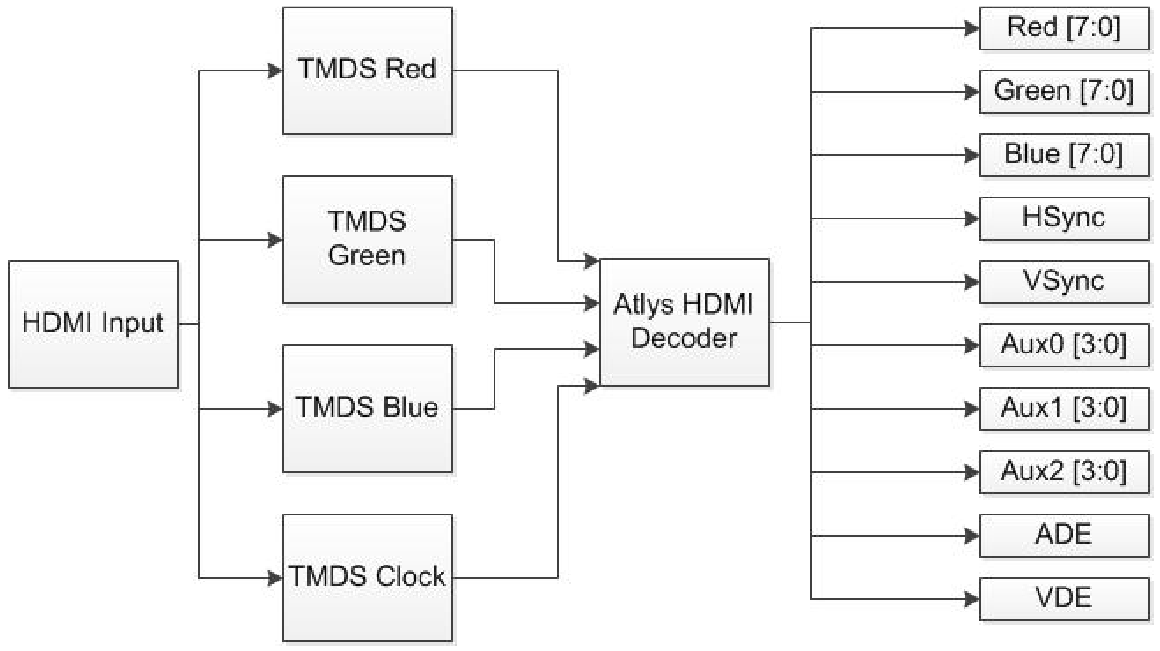


Figure 4.2.2.1 TMDS Input Flowchart

4.2.2.2 Signal Sampling:

If we are to use HDMI input we will be using 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 will be using the Atlys board exclusively as an HDMI sink. All of the data received from the HDMI port will then be sent by some communication method to the secondary spinning microcontroller which will 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 will likely be determined by our choice of LED controllers, which will in turn determine what types of image processing will be required.

4.2.3.1 Color Depth Reduction:

When building each frame, there will be an RGB value for each pixel in that frame. It is quite likely that these RGB values will have a much higher color depth than our display is capable of handling. In code, we will need 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.

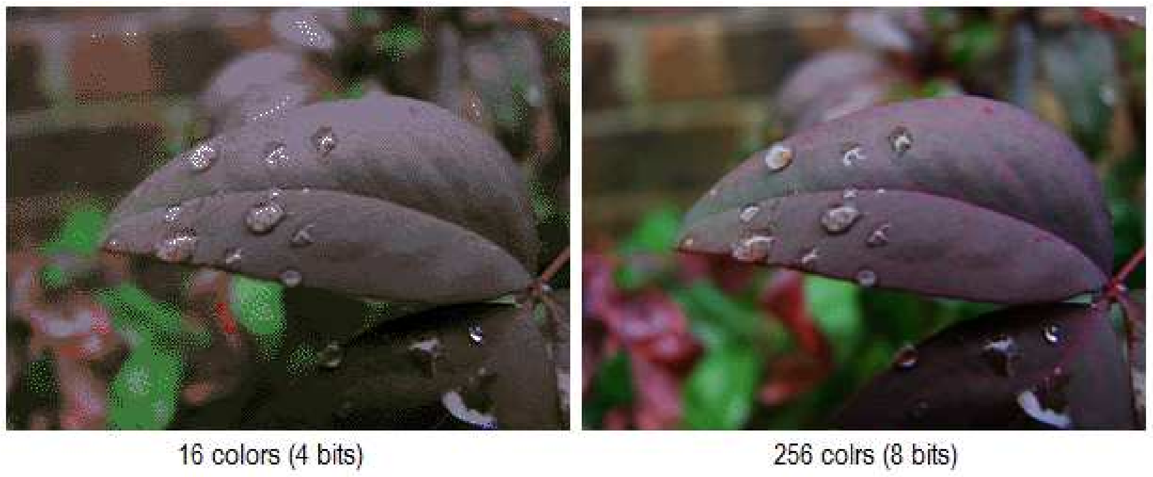


Figure 4.2.3.1 Example of image shown in 4 bit and 8 bit color depth

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.

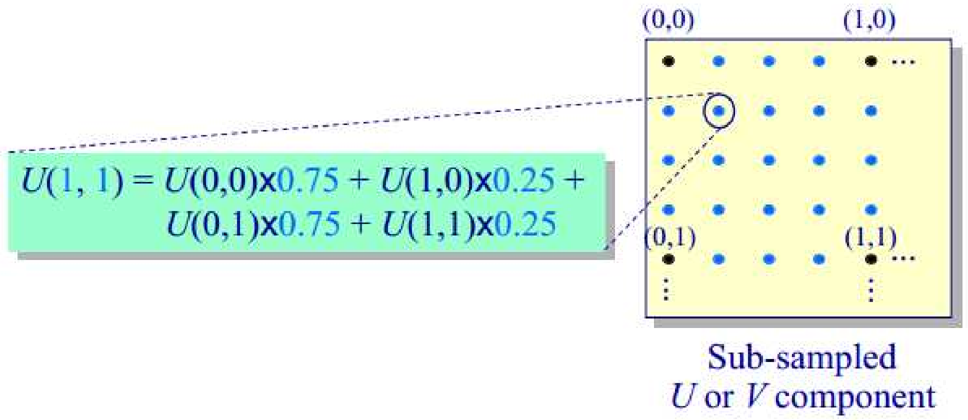

(0,0)
(1,0)
(0,1)
(1,1)
U(1, 1) = U(0,0)x0.75 + U(1,0)x0.25 + U(0,1)x0.75 + U(1,1)x0.25
Sub-sampled
U or V component

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 will turn 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.

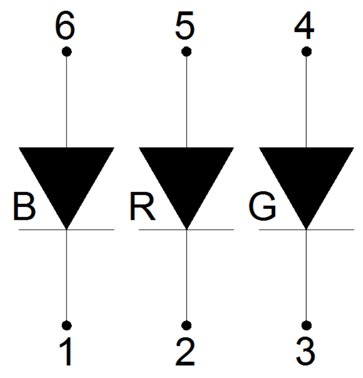


Figure 4.3.1 OVS-33 Pin Information

4.3.2 LED RGB Control:

There are several different methods for controlling RGB LEDs. Two methods that we will be focusing our research on will be 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 will focus 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 are considering using has 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 use 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 be 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.

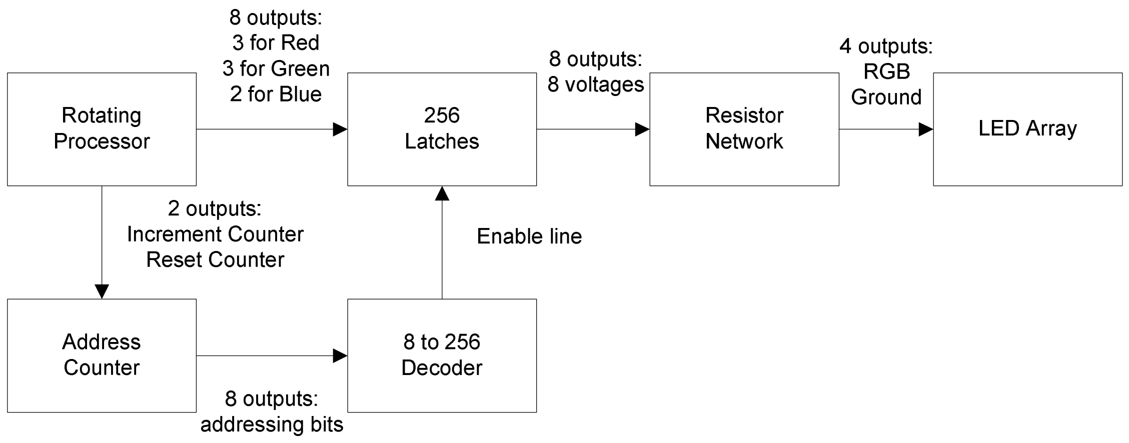


Figure 4.3.1.2 Latch control Implementation

The resistor network would use the 8 outputs from each of the latches, which are 8 voltages, and would be convert 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 is to send a signal encoded with an image or frame of a video, we need 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 is 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 will need in order to implement the system. This is so we can effectively decide the best fit for our POV display.

First we need to determine the number of bits it will require for one LED to display a single color. Since we decided we wanted two-hundred-fifty-six colors we know that we will need 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 know we will need nine bits of information to tell the processor which LED we are 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.

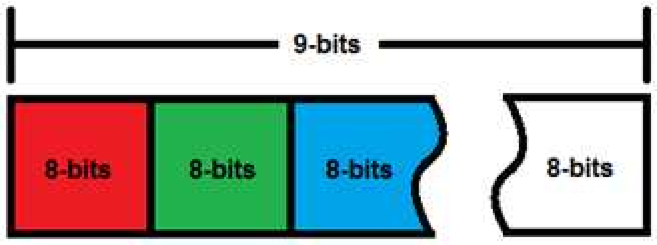


Figure 4.4.1 Data Array

This only turns a single LED in the array a specific color though, 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 we will probably want a data transfer rate a little higher than this, maybe even twice as high as this to make it reasonable that with errors we will still be 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 will restrict some of the available forms of wired communications. From our specifications and project design criteria, we know that our platform will 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 short, any conductor larger for four strands will be unpractical for this application. Therefore, the researched will be focused on two types of wired communications, Fiber Optics and Cooper Coaxial Cable. In both situations, the wired communications will need 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 criteria of the wired communications will be transmission rates. Currently the standard threshold requirements for Ethernet communications is 10 Mbs, 100 Mbs, and 1000 Mbs or 1 Gbs. An additional design criteria for wired communications will be to implement the communications with inducing the minimal amount of interference to the signal. The last design criteria for wired communications will be to evaluate cost benefits between coaxial cable and fiber optic cable. Below a summary of the design criteria is listed and will be 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 will research 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 will be 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 is 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 cable 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 either multimode or single mode fiber cable and a wide wavelength range. Therefore, the research on fiber optic rotary joints will focus 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 meets and exceeds all design criteria for the POV display. Depending on what fiber connector and wavelength is required to connected 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 will research 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 shield 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.

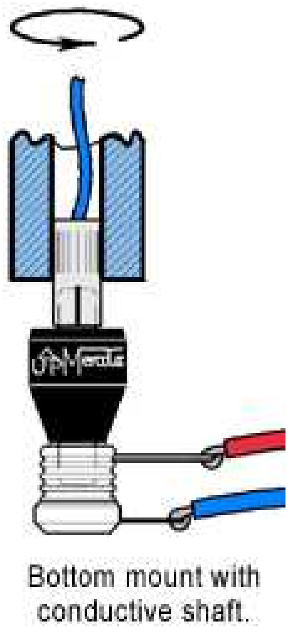


Figure 4.4.2.2.2 Model 205 Rotary Joint for Rotary Interfaces

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 will look 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 does 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 will communicate 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 should be able to use the

Ethernet ports on the FPGA's for communication. If we use the Ethernet core then we will use 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 possibly be to use the Ethernet ports in a non-standard way. We can use the pins on the RJ-45 connector to send the data using our own design. If we choose this route we will not be using any Ethernet protocols but simply sending raw data through a wire. This will be the simplest method to design and implement because it will not require 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 be able to simplify communication greatly. We will create our own header for the data being sent to identify what is being sent. We will most likely use a clock speed of 100MHz for a 100Mbit/s data sampling rate.

4.4.2.4 Microprocessor Ethernet Hardware:

A possible component to implement Ethernet communication on our boards is the Arduino Ethernet Shield, which would require 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 will be considering 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 will have 256 possible colors per pixel, so 8 bits per pixel will be needed. We would also like to transmit 30 frames per second. The minimum required bitrate that we will need in order to achieve the desired frame rate will be 320x240x8x30 which is 18.432Mbps. We are also possibly considering a 480 LED array supporting a 640x480 resolution. If we were to use 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 will be considering both technologies. Generic RF communication will not be considered because we do not believe that it will support the bandwidth that will be required for real time video. We will also be researching if the rotation of the microcontroller will 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 may be possible for us to write software for a PC that will allow 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 will be considering 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 will be considering the ad-hoc mode for this project since it does not require any other external hardware. Ad-hoc will allow us to set up a direct wireless connection between the FPGA and the microcontroller for bi-directional communication. Although bi-directional communication will be supported we will probably 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.

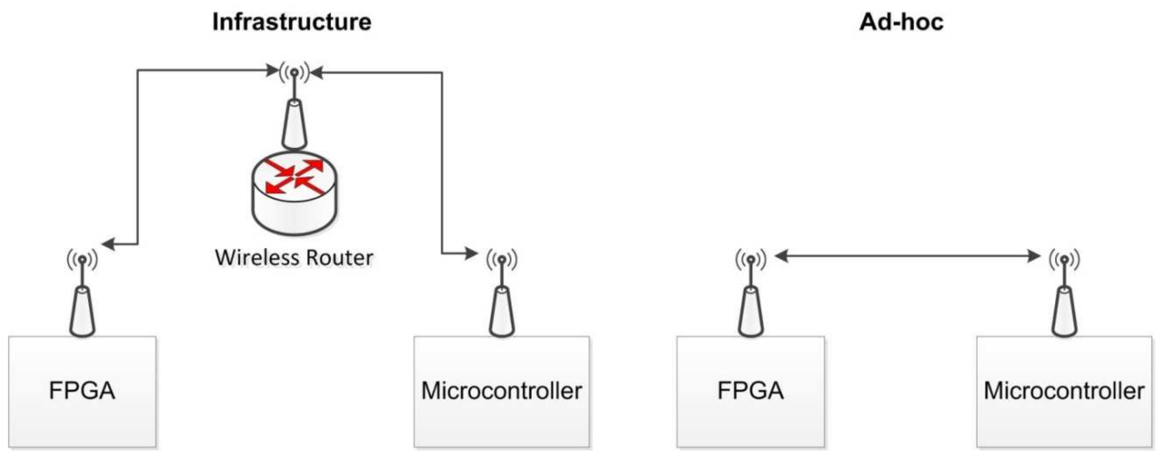


Figure 4.4.3.1.1 Infrastructure/Ad-hoc Comparison.

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 possibly provide an alternative to WiFi. A possible advantage that Bluetooth may have is that it is a low power, short range method of communication. Short range for our application may be 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 can find suitable Bluetooth hardware compatible with our FPGA and microcontroller then this will most likely be 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 will be looking 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 will consider 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 will most likely be 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 may have 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 will 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 will 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 will be using either WiFi or 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 design that houses both the LEDs, processor, and any other circuit elements we need to implement the system. It also needs to be able to rotate at the rpm needed to ‘trick’ the brain into seeing motion. In addition, under the considerations that this project is being 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 choose 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 will show that we would need about 0.5 to 1 horse power 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 are seemingly difficult to acquire, with none at the local Skycraft store available for purchase. Which meant our desire to purchase a cheaper alternative if we decide to use a DC motor would be difficult to accomplish.

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 need a motor that can handle twenty-five rotations per second. This is the bare minimum. We decided that we want 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 need 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 can 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 leaves us with some options and both are relatively inexpensive solutions.

4.5.3 Sound Requirements:

Considering this product is for in-person advertisement uses we want minimal obstructive noise. Especially if we are planning on playing videos off this 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 chasse. However, we still have 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 is the best fit. An AC motor, while capable of reaching the rpm values we need would 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 want to limit within our device. As for the torque requirements it seems that both would pass 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 is 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. 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 has 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 the PWM was capable of doing far more then we needed the controller to do 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 was 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 decided a gearbox was required to rotate it, such as in the case of using a wired transmission process, we would lose 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 is 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 becomes 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 seem 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 is the accuracy. Considering we are using an infrared sensor there is a possibility for some fall trips or even failed trips. This means we need 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 trips 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 use 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 seems to also have a very small error rate. There are however, a few issues with this method based on how we choose to use it.

The first issue with this sensing method is it’s 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, that leaves 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 is 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 will research the requirements for the chassis of the POV display. Two topics that will require research include 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 will be where a majority of the weight is located. In order to maintain the portability of the POV display some materials we will eliminate 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 is an important factor as the chassis will but 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 is desired for portable device. Wood and plastics would reduce the weight of the display but do not offer the flexibility to make a customer design that the display most likely would require. As well, wood and plastics may be more susceptible to twisting and moving while the display is running. Therefore, we will focus 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 will be 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 will be determining the best solution to rotate the POV display. At this time we do not know if we will use wired or wireless communications to connect to the rotating side of the POV display. We will need to research feasible solutions that will work with both forms of communications. If we use wired communications, the center point of rotation must be left available to allow for the mounting of either the fiber rotary joint or the coaxial rotary joint. Wireless communications will not require the center of rotation to be left available but will not be hindered from operation if the center was left available. Therefore, we will be researching 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 will turn 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, is 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 downsides is the cost of the turntable is about $215. The second downside of the turntable is there are no posted maximum rotating speeds. These means 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, is 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 does not have any inner or outer mounting holes, it does 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 extending the requirements of the POV display. As well, the cost of the bearings start 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 will be developing a GUI for use on a PC and possibly also an android device which will allow us to send either a text message or image to be displayed on the POV display. If sending an image to be displayed then the image will have to be in the correct resolution and format. If time permits we may be 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 is 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 showing the simplified waterfall model we will be using for developing the GUI.

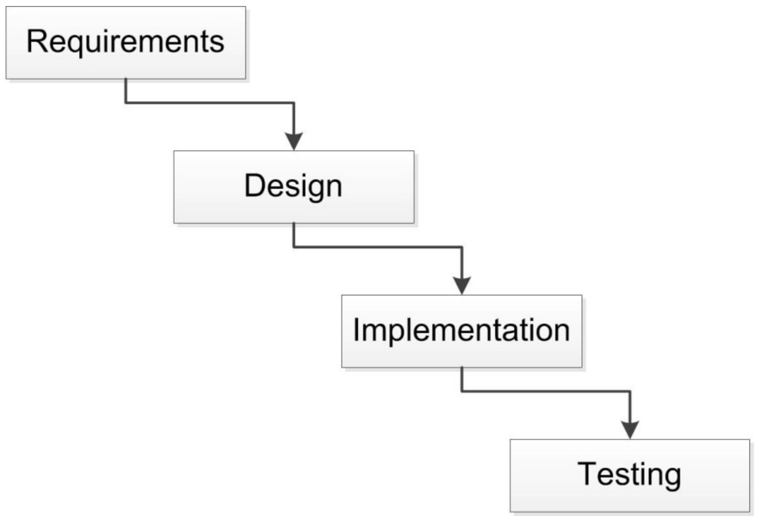


Figure 4.7.1.a Software Development Life Cycle – Waterfall Model

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 are 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 possibly of individual letters. Depending on our time constraints there are certain features that may not be necessary such a changing the color 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 permits we may 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.

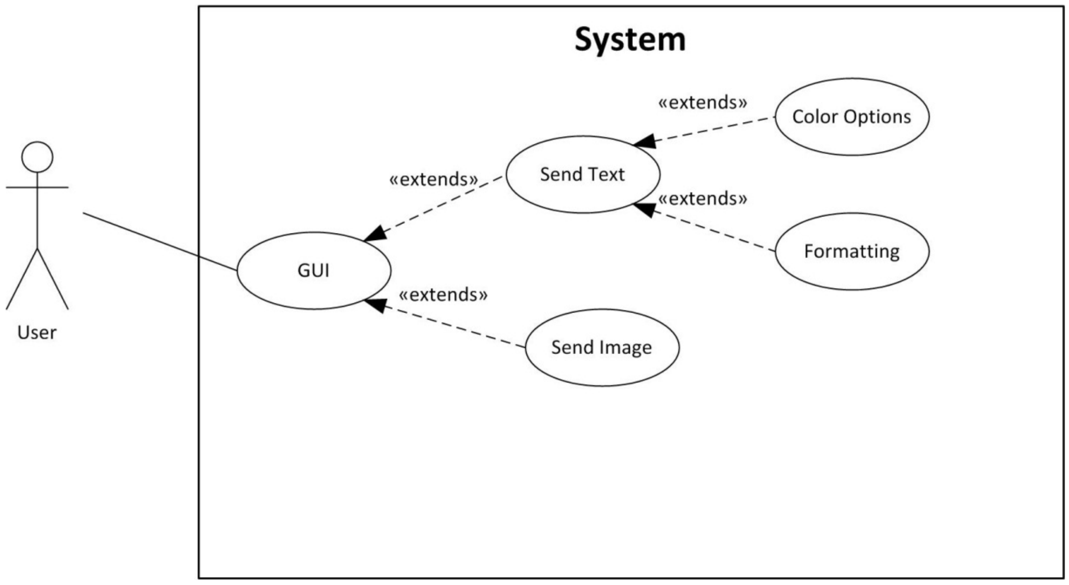


Figure 4.7.1.b Use case diagram for GUI

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 will be considering 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 have 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 are 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 complicates things since further research must be conducted in order to determine which would be the best library to use. There does 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 believe it is safe to say that C++ should not be the language we use to build our communications GUI application. Even though C++ is not the best choice for developing the GUI, C++ may be better suited to interface with the hardware for USB communications. We will be searching for libraries to solve this problem once a language has been decided on.

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 be 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 will 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 will probably be the best language for both developing a GUI for a PC and for an android device if we are able. The Netbeans IDE has a built in visual GUI editor for Java which greatly simplifies GUI design and implementation. The Netbeans GUI editor will allow 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 are also already familiar with the Java programming language. Java is the obvious choice for a high level programming language that we already possess enough knowledge to code in and has 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 is the limited functionality when it comes to accessing connected hardware. This can become an issue for us since we are planning on using communications through USB, or a connected wireless adapter. In order for us to implement USB communications using Java we are going to have to find a suitable driver for our desired operating system and find a Java library that is capable of interfacing with that driver. Assuming we can find such a driver and software library combination it can be safe to say that we will be using java for our GUI application development.

4.7.2.1 Image Format Conversion and Resizing:

Our GUI will allow users to select an image to be displayed and load it onto the rotating processor. Without being too restrictive on the user, we want our program to accept virtually any image file format that is common. The primary information we need 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 have determined that functions within the java library will be able to handle this. Several java classes will 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 involves 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 is used then Bluetooth will 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 Hardware Design:

The hardware of this device is broken into four major sections. That is the input section of the device which encompasses both the computer input such as HDMI and USB. It also includes the power supply of the entire device. Then on the stationary side of the device is the stationary control system. This section includes the motor control system, such as the pulse width modulation circuit and the tachometer. It also includes the stationary FPGA board. Then the next section is the data transfer section. This portion of the device includes both the slip ring design, Ethernet and coaxial rotational joint conversion. Finally the last section of the device is the rotational control section. This section of the device includes the rotating FPGA board and the LED array including the LED controllers. Figure 4 gives a good visual representation of the flow of the hardware and how they will be connected together.

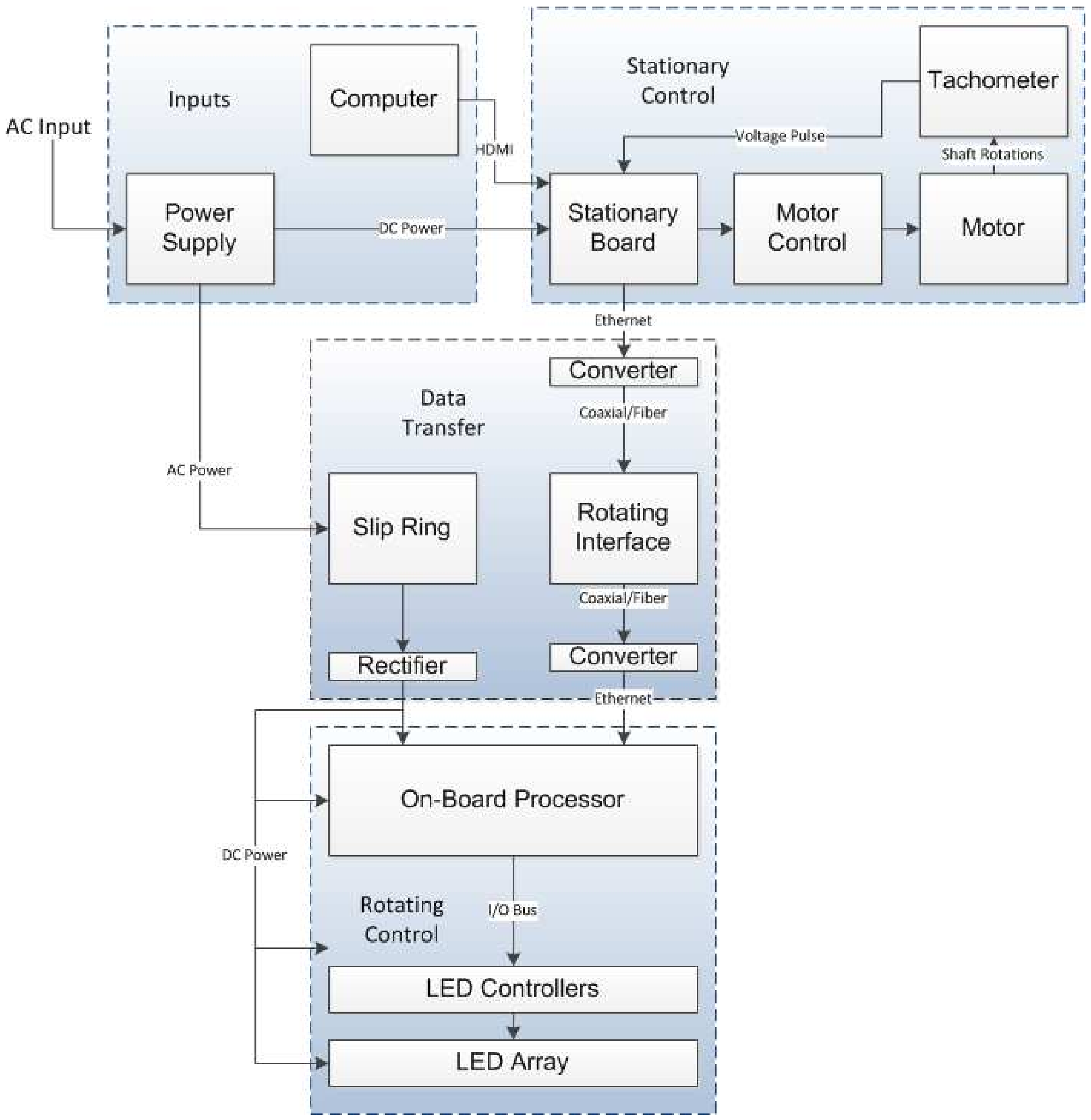


Figure 5 Hardware Flow Chart

Each of these sections of the device have 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 intend to use to implement each of these designs. Also within this section will be a layout of the structural construction of the chassis which will house all of the electrical hardware for this device.

5.1 Chassis Hardware Design:

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

5.1.1 Chassis Dimensions:

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

5.1.1.1 Dimensions of LED Array:

We will be using 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 will be 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 will therefore be ignored for the chassis design. Assuming we will mount the LEDs with a spacing of 1mm between each LED, we can determine that the spacing between each LED, as measured from center to center, will be 2.85mm. Therefore, the total vertical length of the LED array will be 2.85mm x 480 LEDs or 1,368mm. Converting the total vertical length to inches gives a final dimension of 53.858 inches.

Next, we will need 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 will want the horizontal spacing of the LEDs to be the same as the vertical spacing of the LEDs, we can use the known pixel ratio of 480 to 640 to determine the length required. Dividing 640 by 480 gives the ratio of horizontal pixels to vertical pixels, which equals 1.333 or 5:3. For accuracy, we will initially calculate the required horizontal length in millimeters. Taking the ratio of 5:3 and multiplying by the known vertical length of 1,368mm we get a required horizontal length of 1,824mm. Since the LED array can be simplified to a cylinder, we now know the circumference of the LED array. Using the formula of $C = 2\pi r$, we can calculate the radius of the LED array which is equal to 290.299mm. Converting the radius to inches, we get a final dimension of 11.429 inches.

In summary, the required dimensions of the LED array when the POV display is running will be equal to a cylinder with a radius of 11.429 inches and a height of 53.858 inches. If we convert the circumference to inches, as seen below, we can verify that our results due match the desired aspect ratio of 480 x 640.

$$\frac{53.858}{480} = 0.112 = \frac{71.811}{640}$$

5.1.1.2 Dimensions of Chassis Base:

Now that we know the size of the LED array when the POV display is spinning, we can determine an appropriate size for the chassis base. The chassis base will serve two purpose 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, will be to provide a visually marker to signify where the limit is to approach the display while it is running. This will be especially important if, for example, the display is running but displaying an image. Therefore, we will construct the base of the POV display to extend two inches past the fast rotating LED array. Since we now know the radius of the spinning LED array, we can determine that the base will need to be at least a 27 inch by 27 inch square. To simplify the fabrication process and for extra precaution, we will construct the base to be an even 28 inch square. Since we will be constructing the chassis out of 1/4 inch aluminum plate, a 28 inch square base should provide plenty of weight and strength to fully support the POV display while it is running.

The last dimension required before we can determine the final design of the chassis will be to know the physical size of the motor. Size the motor will be mounted on the base, it will determine the height of the base. The motor we will be using is a wound field DC motor manufactured by Prestolite Motors. The overall length of the motor is 6.8 inches. To allow room to mount and secure the motor to the base, we will design the base with an internal height of 8 inches. Taking into account the thickness of the aluminum plate, the total height for the base will be 8.5 inches. Therefore, the total size of the chassis base will be 27 inches x 27 inches x 8.5 inches.

5.1.2 Chassis Assembly:

Now that we know the required dimensions of the chassis we can begin to design the assembly of the chassis. A complete chassis model can be seen in Figure 5.1.2.c below.

The first step to putting together our final design of the chassis will be 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 have two options. The first option of the turntable provides 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 will choose to use the extended-ring bearing. In order to provide the most space for feeding the power supply cable and communications cable through the rotating interface, we will choose to use the extended-ring bearing with a one inch shaft diameter, part number 8090T13. We will secure 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 will provide a strong and secure method that will hold the bearing in place during operation of the POV display. In order to secure the LED array to the bearing, we will insert a aluminum pipe through the inner ring of the bearing. The pipe will be secured to the bearing using the two set screws that come installed on the bearing. This will allow for the POV display to be easily disassembled when moving between locations. Using this design allows the pipe to be used to mount a slip ring for electrical power transfer as well as a pulley system from transferring the rotationally power from the motor. Lastly, the pipe will be notched on the top to allow the LED array support bar to be secured to the pipe. Figure 5.1.2.a below shows a model of the bearing and pipe assembly. The chassis base and LED array support frame are removed for clarity.

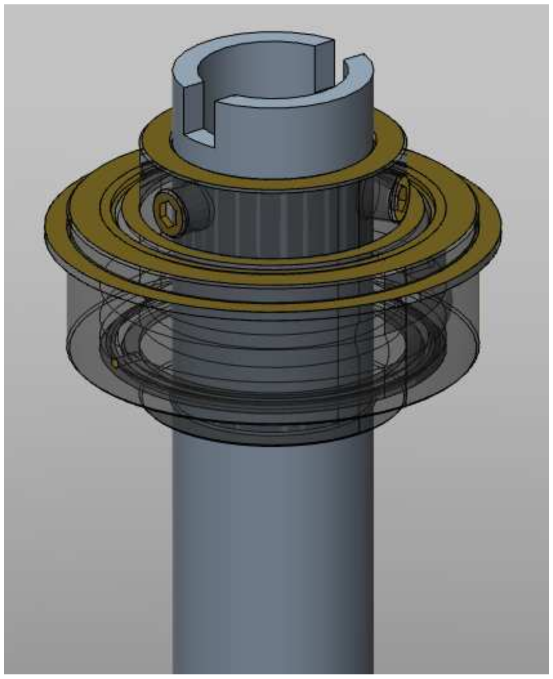


Figure 5.1.2.a Bearing Assembly

Next we will design the chassis base. As discussed, the chassis base will be 28 inches x 28 inches x 8.5 inches. The base will be constructed out of two 1/4 inch pieces of aluminum plate creating a top plate and a bottom plate. The two plates will be secured together by four solid aluminum rods, one in each corner, cut to 8 inches lengths. The plates will have counter sunk holes drilled in each corner, three inches from each side. The rods will be drilled and tapped in the center to accept a 1/4-20 screw. The rods and plates will be assembled by screwing the plates and rods together. The counter sunk holes on the plates will allow for the screws to be flush with the surface. In order to mount the bearing, a hole will be cut out from the center of the top plate. The diameter of the hole will be larger than the diameter of the inner ring of the bearing but smaller than the extended flange of the bearing. This will allow for the bearing to rest on the top plate and provide a surface for the bearing to be welded to the plate. Figure 5.1.2.b below shows a model of the base assembly, including the cut out on the top plate for the bearing.

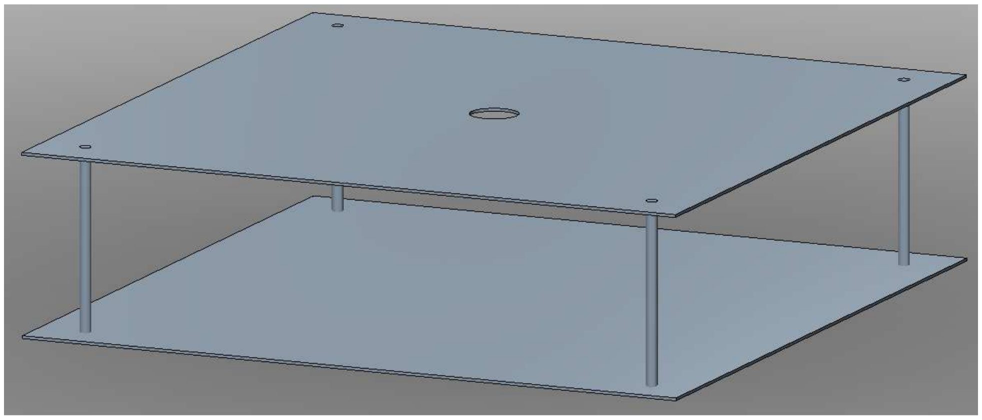


Figure 5.1.2.b Chassis Base Assembly

The LED array support frame will be constructed from 1/8 inch square tubing. From the calculations for dimension requirements of the LED array, we know that the horizontal LED array support bar, the piece that will be connected to the notched pipe, needs to be 22.585 inches long. This dimension needs 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 will be welded. We know from the LED array dimension requirements that the vertical support bars must be at least 53.585 inches long. However, because the actual length of the vertical bar does not determine the aspect ratio of the display, to simplify the fabrication process we will construct the vertical support bars to be 55 inches long. The vertical support bars will be drilled and tapped to accept 10-32 screws every two inches. This will provide enough mounting holes to mount the printed circuit boards for the LED array. The number of mounting holes will help reduce vibration and securely fasten the LED array to the chassis while the display is running. We will determine if diagonal support bars are required after testing the chassis.

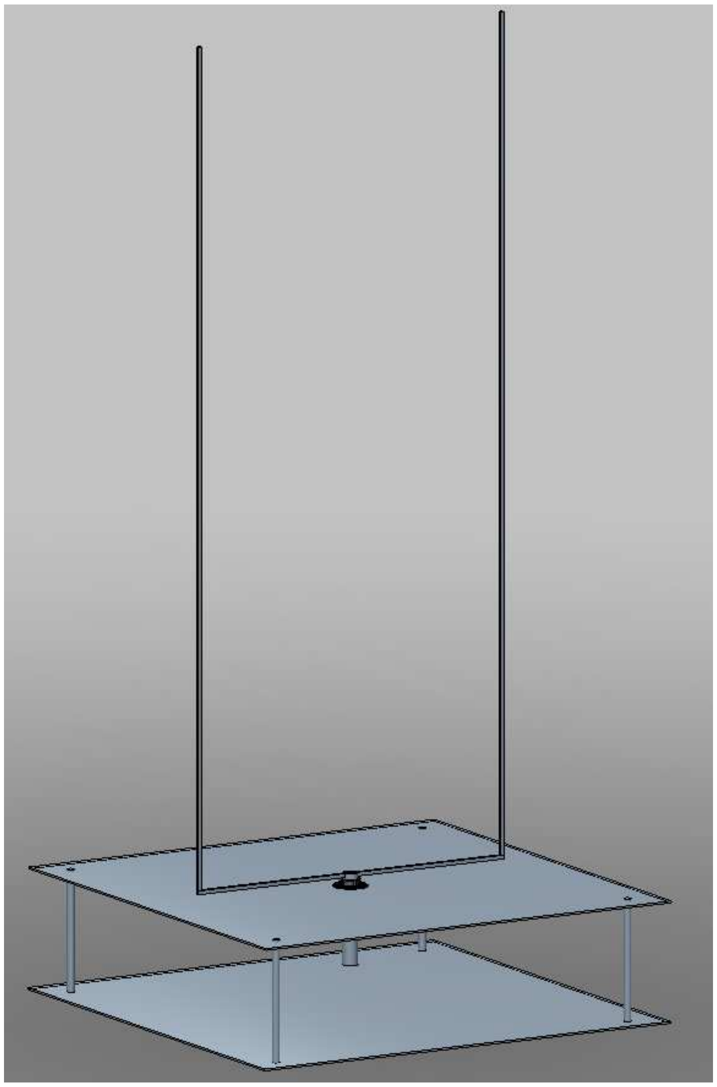


Figure 5.1.2.c Chassis Base Assembly

5.1.3 Motor Interface:

As briefly mentioned during the chassis assembly section, to transfer the power from the motor to the chassis, we will purchase a pulley to be mounted to the shaft of the motor. Since the outer diameter of the pipe is one inch, to maintain a 1:1 rotational transfer, we will use a pulley with a one inch outer diameter and a bore size appropriate to fit on the motor shaft. If we use a flat belt pulley system, for the chassis side we would be able to attached the belt directly to the pipe. To minimize the belt from slipping we would increase the friction on the section of pipe where the belt would touch.

5.1.4 Chassis Torque Calculations:

Now that we have a finalized design for the POV display chassis, we will need to estimate the torque requirements. This will only be an estimate to help decide the size of motor we will require and not an exact calculation. To help simplify the problem we broke the chassis into several pieces for which we can calculated the moment of inertia for each piece. Adding together the moment of inertia for each piece gave use a total inertia of 0.757 kg-m2. Using a frequency of rotation of 44 Hz, we calculated the angular acceleration. Multiplying the total moment of inertia by the angular acceleration gave us an estimated torque value of 0.876 N-m. Therefore, the motor must be capable of providing 1 N-m of torque at 2,640 RPMs.

5.2 LED Array Hardware Design:

As discussed during the research section for the LED array, there are 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 will choose to control the LED array using the PWM controllers. In particular, we will be using 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, the TLC5940 allows for dot correction of individual LEDs if we find, during testing of the LED array, that one or more LEDs appear dimmer or brighter than all other LEDs. However, there is one very important design restriction that we must overcome in order to use the TLC5940. If we maintain our current design criteria of 30 fps and 640 horizontal pixels, our frequency of updating the LED array will be 30 x 640 or 19,200 times per second. As well, we will cascade all the red LEDs together, all the green LEDs together and all the blue LEDs together. This will mean we will require (3) groups of (30) cascaded TLC5940 controllers. By applying the characteristic equations below to our requirement of an updating frequency of 19.2 KHz and cascaded controllers, we will find that we will exceed the max serial data transfer rate of 30 MHz.

$$f(GSCLK) = 4096\ x\ f(update) = 4096x19200 = 78{,}643{,}200\ Hz = 78.64\ MHz$$
$$f(SCLK) = 193\ x\ f(update)x\ n = 193x19200x30 = 111{,}168{,}000\ Hz = 111.2\ MHz$$

$$Where:$$
$$f(GSCLK)\ equals\ the\ minimum\ frequency\ needed\ for\ the\ gray\ scale\ clock.$$
$$f(SCLK)\ equals\ the\ minimum\ frequency\ needed\ for\ SCLK\ and\ SIN$$
$$n\ eqauls\ the\ number\ of\ cascaded\ controllers$$

Initially, some additional research was done to determine if another controller was available that could operate at a maximum data transfer rate of 111 MHz. It was quickly discovered that no other option, at least at the cost we require, that could handle this high data rate transfer. The data rate transfer restriction was due to the shift register inside the TLC5940 controller. Therefore, to overcome the restrictions of the TLC5940 controller we will make two changes to the POV display. The first change will be the frame rate. Essentially, we will use a variant of vertical interlacing. Instead of operating at 30 fps and 640 horizontal pixels, we will have two groups of LED controllers operating at 22 fps and 320 horizontal pixels. The two groups of LED controllers will be called Group A and Group B. Group A will operate while Group B is being addressed. Then, while Group B is operating, Group A will be addressed. Both groups of LED controllers will still output to the same vertical array of LEDs (so no additional LEDs are required) but now each group can be individually addressed at a much slower rate. The total frames per second as seen by a user will now be 44 fps but still maintaining the 640 pixels. Therefore, the new frequency of updating for each group of LED controllers will be 22 x 320 or 7,040 times per second. The only disadvantage to adding a secondary group of LED controllers is we now require double the amount of controllers or a total of (180).

However, we still exceed the maximum transfer rate of the SCLK and SIN. To overcome this restriction we will change the number of cascaded controllers. Similar to the solution of vertical interlacing, we will break the vertical LED into two groups, a top group and a bottom group. The top group will be called Group A and the bottom group will be called Group B. Each group will have a total of (15) cascaded LED controllers. However, unlike the vertical interlacing, both of these groups will operate at the same time. Therefore, we will now have four groups of LEDs. As seen in Figure 5.2, we will have a Group AA, AB, BA and BB. As well, if we recalculate the minimum frequency requirements of the GSCLK, SCL and SIN signals, we will find that we now operate well within the maximum data rate transfer of 30 MHz.

$$f(GSCLK) = 4096\ x\ f(update) = 4096x7040 = 28{,}835{,}840\ Hz = 28.83\ \mathrm{MHz}$$
$$f(SCLK) = 193\ x\ f(update) x\ n = 193x7040x15 = 20{,}380{,}800\ Hz = 20.38\ MHz$$

	Vertical Group A Pixel Column 1	Vertical Group B Pixel Column 2	Vertical Group A Pixel Column 3	Vertical Group B Pixel Column 4	…	Vertical Group A Pixel Column 637	Vertical Group B Pixel Column 638	Vertical Group A Pixel Column 639	Vertical Group B Pixel Column 640
Horizontal Group A	**Group AA**	**Group BA**	**Group AA**	**Group BA**	**…**	**Group AA**	**Group BA**	**Group AA**	**Group BA**
Horizontal Group B	**Group AB**	**Group BB**	**Group AB**	**Group BB**	**…**	**Group AB**	**Group BB**	**Group AB**	**Group BB**

Figure 5.2 LED Array Group Layout

5.2.1 TLC5940 Pin Out and Wiring:

Now that we have a handle on the number of LED controllers required and their configuration, we can determine the pin out and wiring of the LED controllers. As previous discussed, we will have four groups of LEDs. However, to simplify the design we can consider Group AA to be indicial in design to BA and AB to BB. This is because essentially the two vertical interlaced groups will be the same. The pin out information for a TLC5940 in a NT case can be seen in Figure 5.2.1.a below. Table 5.2.1 below shows all pins and their functions.

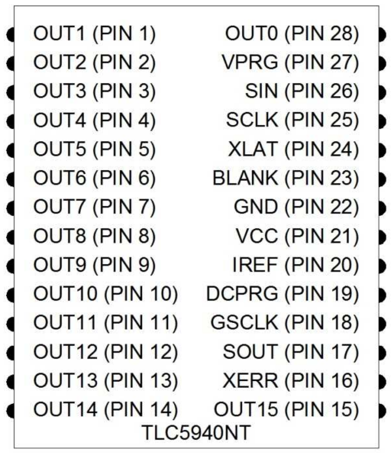


Figure 5.2.1.a TLC5940 LED Controller Pin Out

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 5.2.1 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 will be wired to the SIN pin on another controller. The wiring required for the controllers of Group AA and BA can be seen in Figure 5.2.1.b. Wiring of Out0 thru Out15 was left off for clarity but it should be noted these will be the pins connecting to the LEDs. Wiring for Groups AB and BB will be the as Figure 5.2.1.b.

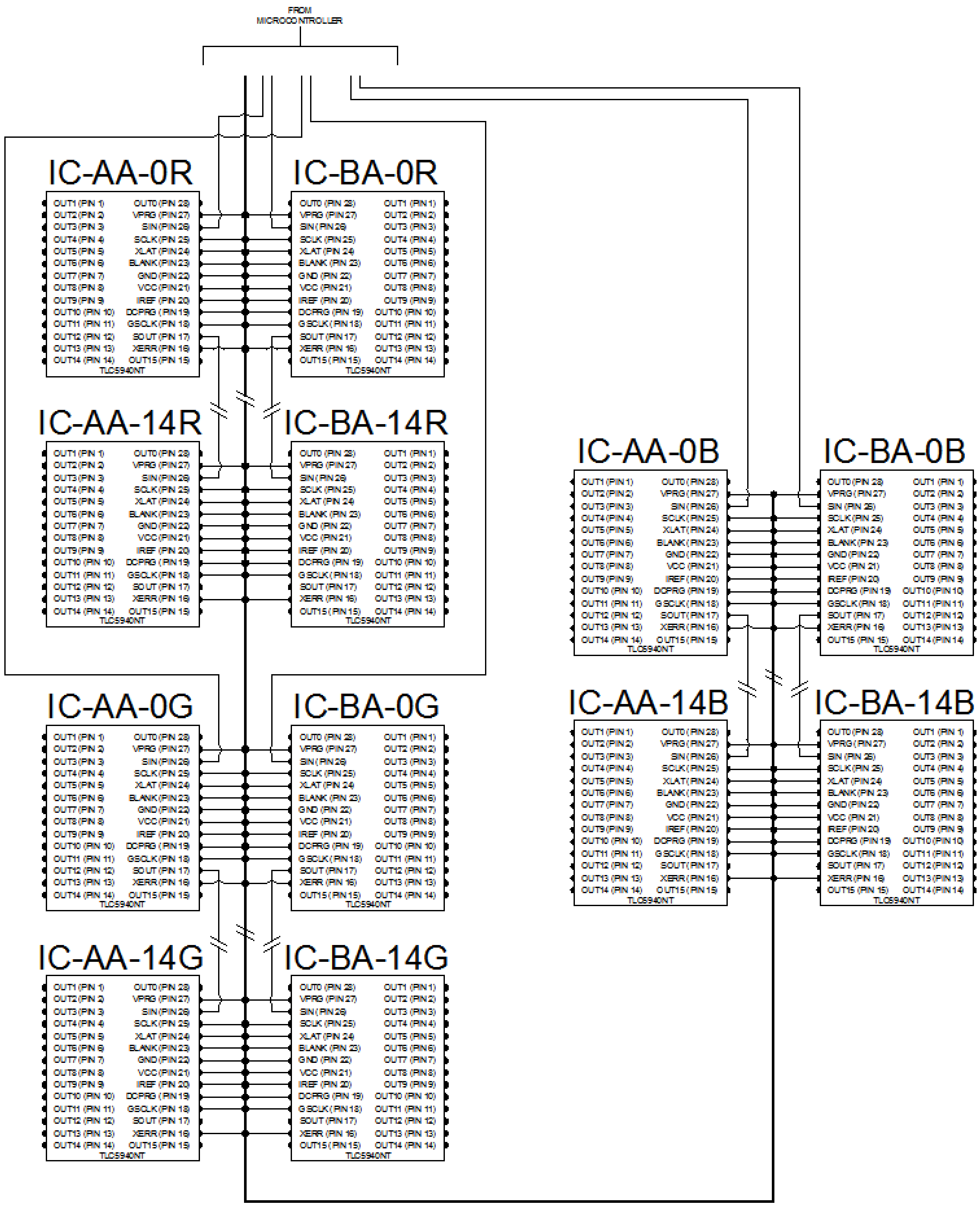


Figure 5.2.1.b LED Controller Wiring

A wiring scheme for the LEDs of array Group A can be seen in Figure 5.2.1.c below. The wiring for Group B will be the same but the first LED will start at number 240.

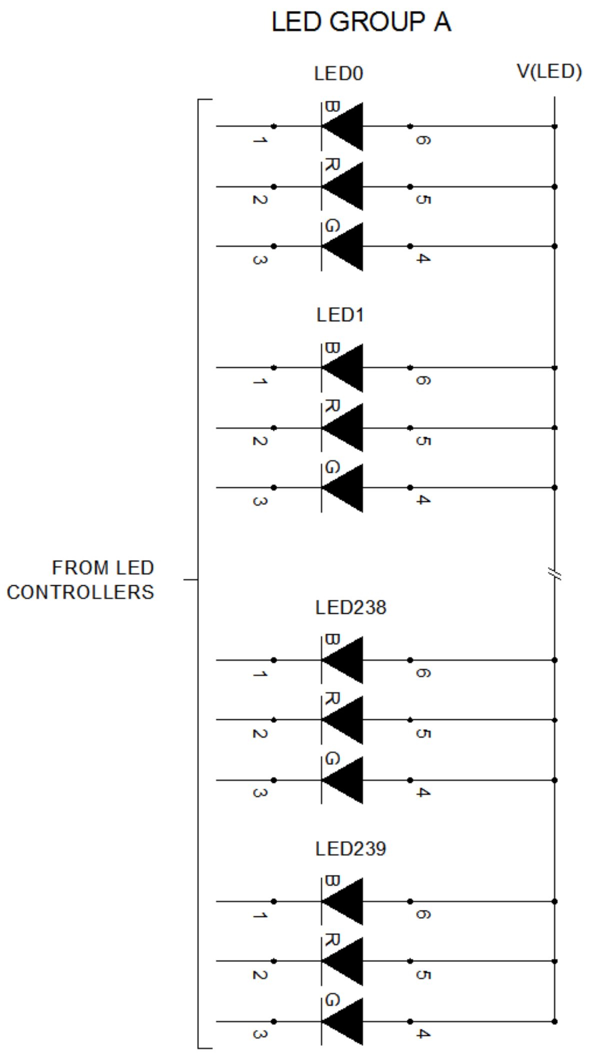


Figure 5.2.1.c LED Wiring

5.2.2 LED Array for Text Display:

The design for the LED array required for displaying text will use the same LED controller but we will only be using mono-color LEDs. The text display will contain 16 LEDs so only one controller will be required. The wiring of the controller and LEDs will be similar to Figures 5.2.1.b and 5.2.1.c.

5.3 Motor Hardware Design:

Our specific motor design encompasses a six stage process that covers how to both run and maintain the motors revolutions per second to prevent any image distortion. This process is cyclical in essence beginning and ending with the motor. Figure 5.3 will give a visual representation of exactly how this process will flow.

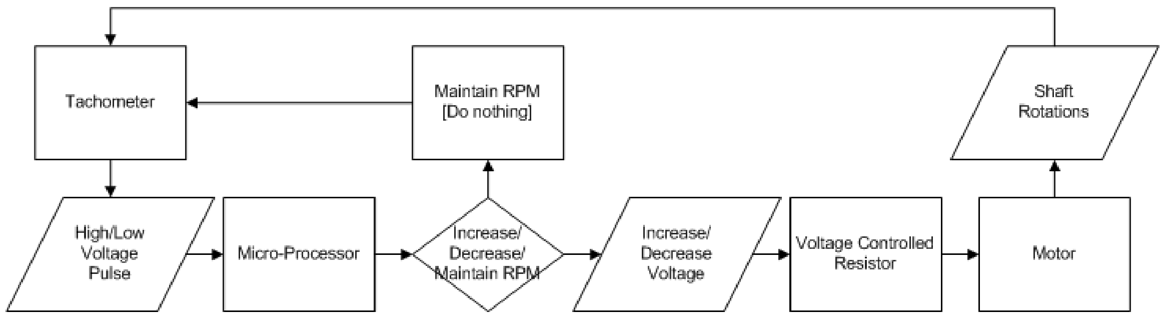

Tachometer
Maintain RPM [Do nothing]
Shaft Rotations
High/Low Voltage Pulse
Micro-Processor
Increase/ Decrease/ Maintain RPM
Increase/ Decrease Voltage
Voltage Controlled Resistor
Motor

Figure 5.3 Motor Control Flow Chart

As seen in the flow chart we will begin discussing our motor operation process with the Tachometer. While in essence the motor and shaft rotations is truly the beginning of this process, the data gathered by the tachometer is the beginning of our motor controlling process, which is the purpose of most of this hardware and software. The whole process is cyclical in nature and should repeat indefinitely until the micro-processor sends a shut down signal to the tachometer and motor. Each section of this portion of the design will discuss in further detail the hardware and software of this process, beginning with the tachometer and ending with the motor.

5.3.1 Motor Control Sensor Design:

As we discussed in the research portion of the motor control process we knew that we had to find a way of determining the revolutions per minute of the apparatus. During that discussion we determined that the best method to solve this issue was through the use of infrared. Specifically these sort of devices are called tachometers. In our case we are going to create this tachometer using a pair of infrared LEDs. Figure 5.3.1.a best shows the flow of this process.

We are going to 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 are going to use an op-amp to detect these small voltage changes. The intention is to send an infrared signal to the shaft of our display apparatus. Figure 5.3.1.b gives a visual representation of what is happening.

Mounted on the shaft will be some form of reflective surface such as white tape, that will bounce the infrared beam back to the other LED. This beam will cause a

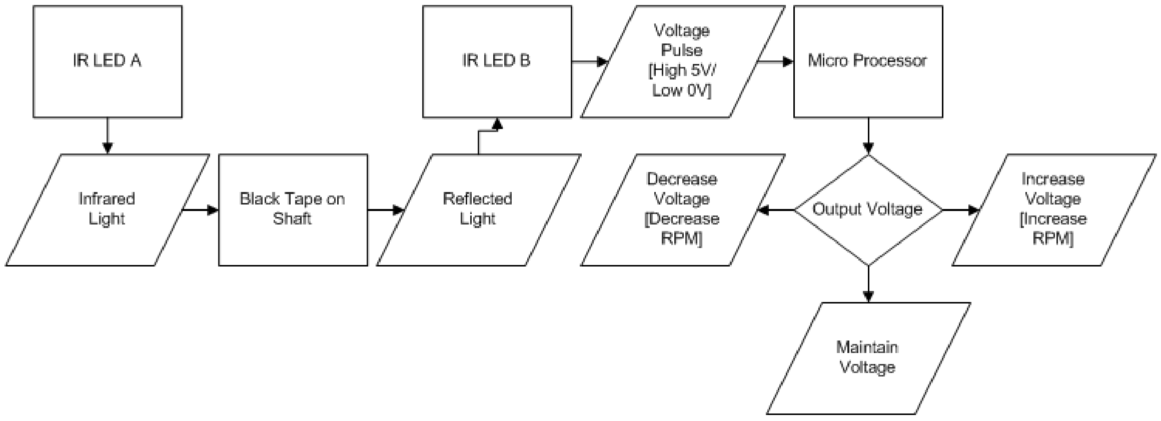


Figure 5.3.1.a Motor Control Sensor Flow Chart

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, expected to be around 5 volts. This is instead of the usual zero volts. This pulse of voltage will then be detected by the micro-processor and recorded as a one. In order to shield this process from unwanted ambient lighting we will most likely house these two LEDs in some form of cylindrical tube so that the infrared beams can escape the tubing and reflect back into it but the interior is mostly protected from unwanted lighting.

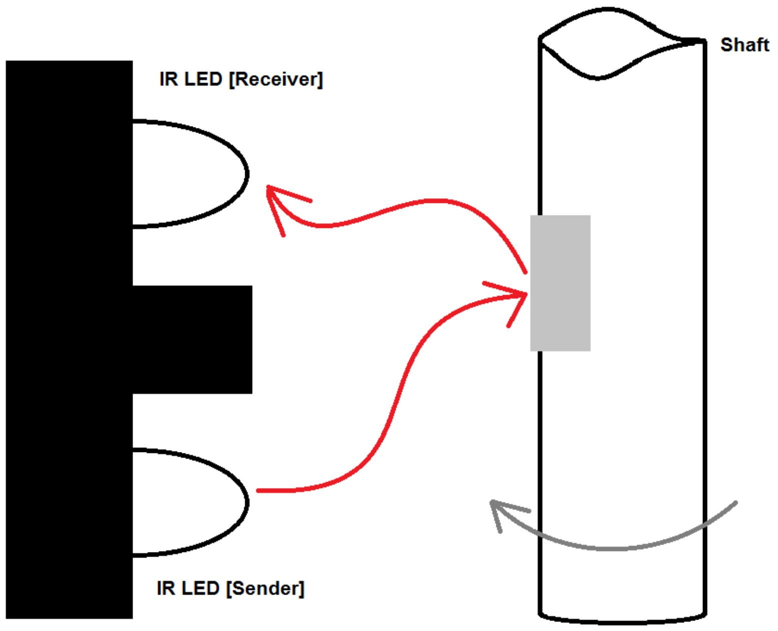


Figure 5.3.1.b Motor Control Sensor Mounting

5.3.1.1 Motor Control Sensor Hardware:

Two circuits are required to implement this tachometer. The first circuit, displayed in Figure 5.3.1.1.a, is the sending circuit. As seen in this circuit we will be using the LM358 op-am to implement this circuit. In this case a 5 volts Vin and Vcc is required to turn the circuit on. The minus terminal of the comparator will read 2.5 volts. The CTRL line will be connected to the FPGA on the rotating side and will in essence go high when the device starts spinning. This high value will be above the 2.5 volts on the minus terminal and cause the output of this op-amp to go high, 5 volts, and this will turn the infrared LED on and begin to send signals.

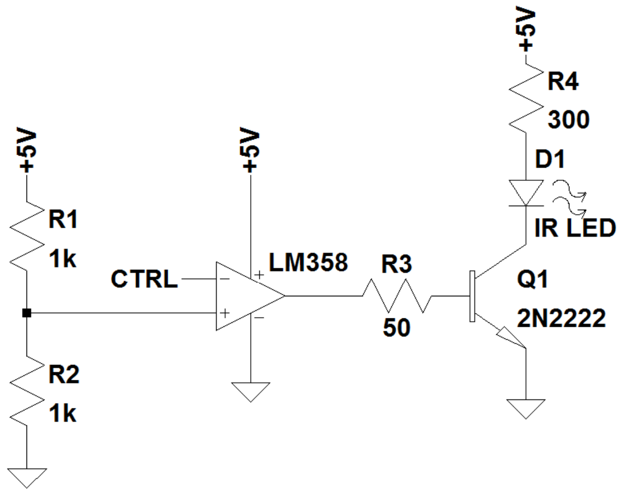


Figure 5.3.1.1.a Motor Control Sending Circuit

The second circuit required to implement this design is the receiving circuit, seen in Figure 5.3.1.1.b. This circuit will be similar in design as the first one except that the infrared LED will be feeding into the minus terminal of the op-amp. The positive terminal will have a potentiometer that will be preset to read 1.6 volts on the positive terminal of the LM358 comparator. When the infrared LED is hit by the beams of its sister LED it will create a voltage drop on the LED. This deference will cause the minus terminal to fall lower than 1.6 volts and cause the output of the op-amp to go high. When the op-amp goes high it will send a voltage drop, 5V, to the processor. Since we are using infrared lights and this whole process will be invisible to the human eye, a Red LED has been added to the circuit to blink when a “hit” is read in the receiver allowing us to see whether the tachometer is working or not.

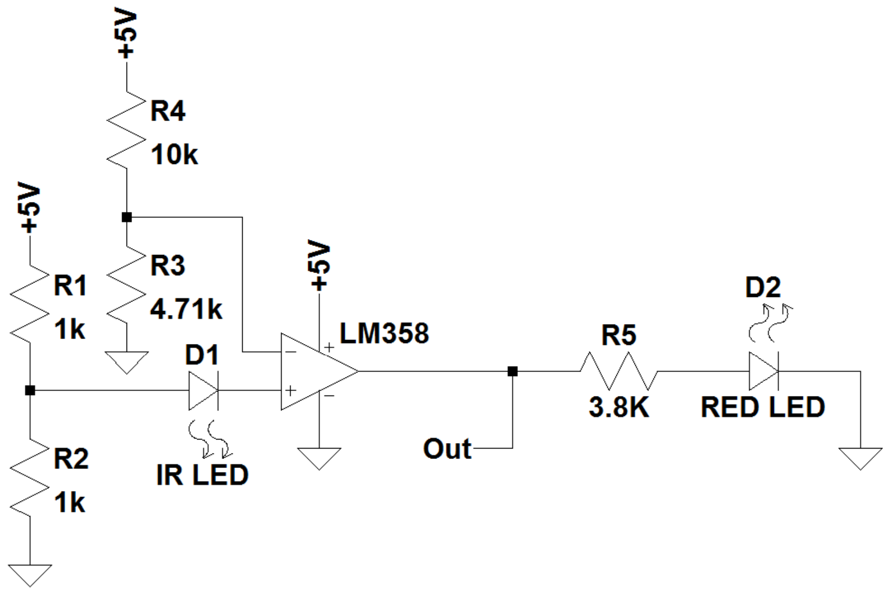


Figure 5.3.1.1.b Motor Control Receiving Circuit

5.3.2 Motor Speed Controller:

As said in section 4.5 when we discussed motor control we decided that a pulse width modulation circuit would be best for implementing a control element to our motor. It has the least amount of power consumption and is far more accurate. However, since we require a large motor to rotate our LED apparatus and to reach the revolutions per second that we desire we needed to work a little with our control circuit to get it to work effectively. Figure 5.3.2.a shows the circuit design of our pulse width modulation circuit. This circuit is based off a reference circuit Figure 10.4.3.a that can be seen in the appendix.

The circuit runs off the concept of using two LM339 comparators to create a pulse width modulation whose frequency and duty cycle can be controlled by the two voltage controlled resistors R9 and R8. R9 controls the frequency of the PWM and can range from 400 hertz to 3 kilo-hertz. R8 controls the duty cycle of the circuit which also means it controls the effective rpm value of the motor. This can range from 0 to 100% of its rated rpm value. The circuit is broken into two components the motor side and the pulse width modulation circuit logic. These two components are connected through a cascade of MOSFETs. For our purposes we needed to disconnect the power supplied to the DC motor and the MOSFET from the pulse width modulation circuit because of our high voltage requirements of the motor. In addition, the circuit will most likely need to be mounted on some form of cooling system to prevent overheating, most likely a radiator.

In order to make these resistance values change without having to physically change the circuit or a potentiometer we decided to use a component that can create the same effect of what a resistor can and that is a JFET. Since a resistor decreases current flow in the circuit as its resistance increases and a JFET's

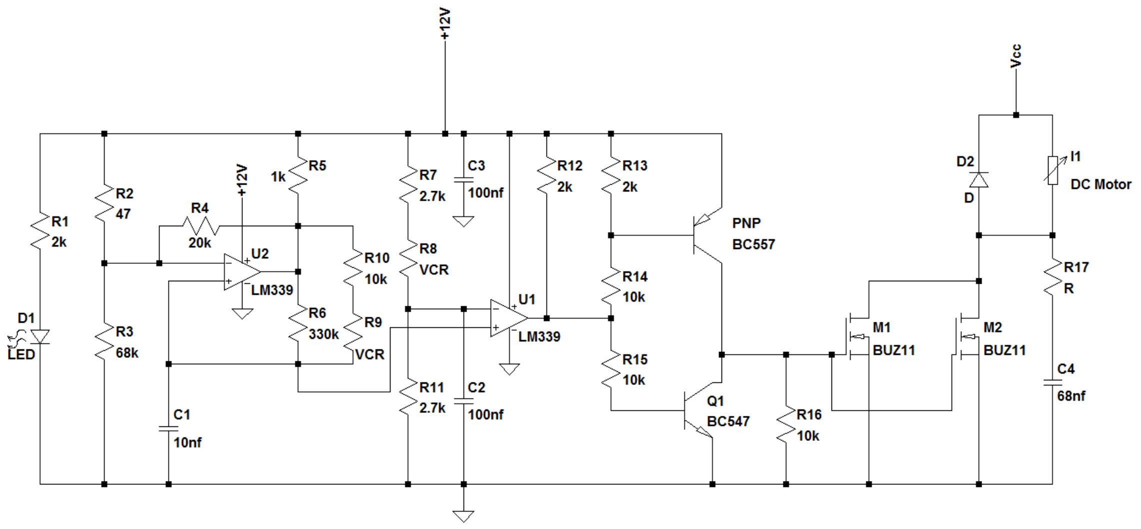


Figure 5.3.2.a PWM Circuit

channel becomes pinched as the voltage increases which essentially creates a similar effect of impeding current flow. After some quick research it turned out that this is actually a common application of JFETs. The VCR circuit we will use is shown in Figure 5.3.2.b. This circuit runs on the principle that Vout will effectively be controlled by a voltage divider using a JFET instead of a resistor. From these basic principles as we increase the voltage on this circuit our voltage controlled resistor will increase the resistance thus allowing us to control the duty cycle of the pulse width modulation circuit and in turn the revolutions per second of the motor. We can also use this same principle for the frequency control of the pulse width modulation so we will need two of these circuits, one for R9 and one for R8.

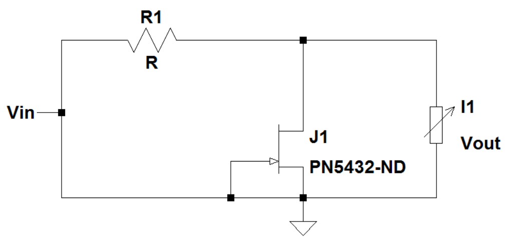


Figure 5.3.2.b Voltage Controlled Resistor Circuit

5.4 Primary Microcontroller Hardware Design (Stationary):

We will use the Atlys Spartan 6 microcontroller to process the incoming HDMI video signal. The controller will also communicate via USB port with the graphical user interface on the computer. The controller will also send and receive data from a sensor used to control the rotation speed of the POV display. The processed video stream, sensor data, and USB data, will all be forwarded to the rotating controller via Ethernet communication using the built in Ethernet port on the Atlys Spartan 6. Figure 5.4.a shows the various communication ports used by the controller.

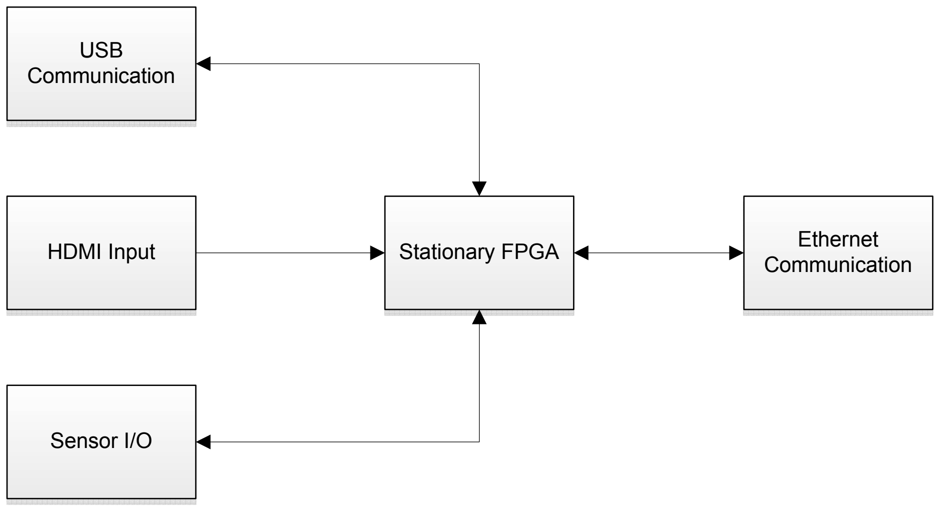


Figure 5.4.a Communication Hardware Used by Stationary Controller

A Vmod Bread Board attachment will be required in order to make I/O pins available to the microcontroller. Three I/O pins will be used to send and receive data from a sensor. The sensor will supply rotational position data and can be used by the controller to calculate the rotational speed of the POV display. The sensor will also receive a signal from the controller which will alter the rotational speed for the POV display. Figure 5.4.b shows the pin out assignments on the Vmod Bread Board.

Sensor Pin Assignment	
Pin	Assignment
I/O 1	Enable
I/O 2	S_INPUT
I/O 3	S_OUTPUT
I/O 4 - 28	Unused

Figure 5.4.b Sensor Pin Assignment on Stationary VmodBB

5.5 Secondary Microcontroller Hardware Design (Rotating):

In order to use the IO pins of the Atlys Spartan 6, we will need to connect a Vmod breadboard to the Atlys board via the VHDC connection, which will provide us with 28 I/O pins. These pins are located on two 32 pin breadboards, BB1 and BB2. Figure 5.5.a shows a summary of the pin configuration on BB1 and BB2. Each LED controller requires various signals from the rotating microcontroller, in several cases though entire groups of LED controllers can share the same signal as the other controllers in the same group. For example, on I/O pin 3, A_XLAT sends a latching signal to all controllers in the A column, which is made up of 90 LED controllers. The signal names have been appended with a prefix, either ALL_, A_, B_, AA_, AB_, BA_, or BB_, which refers to the group of LED controllers that will receive that signal.

Pin Assignments							
BB1		BB2		BB1		BB2	
Pin	Assignment	Pin	Assignment	Pin	Assignment	Pin	Assignment
I/O 1	ALL_VPRG	I/O 15	B_XLAT	I/O 8	AA_SIN_G	I/O 22	BA_XERR
I/O 2	ALL_DCPRG	I/O 16	B_GSCLK	I/O 9	AA_SIN_B	I/O 23	BB_SIN_R
I/O 3	A_XLAT	I/O 17	B_BLANK	I/O 10	AA_XERR	I/O 24	BB_SIN_G
I/O 4	A_GSCLK	I/O 18	B_SCLK	I/O 11	AB_SIN_R	I/O 25	BB_SIN_B
I/O 5	A_BLANK	I/O 19	BA_SIN_R	I/O 12	AB_SIN_G	I/O 26	BB_XERR
I/O 6	A_SCLK	I/O 20	BA_SIN_G	I/O 13	AB_SIN_B	I/O 27	Unused
I/O 7	AA_SIN_R	I/O 21	BA_SIN_B	I/O 14	AB_XERR	I/O 28	Unused

Figure 5.5.a Pin Assignments on VmodBB

One design feature that might be implemented would be a text display which would consist of 3 LED controllers outputting to 16 LEDs. This would in essence be a miniature of the full miniature LED array. If in fact we implement this text display we would need to modify the pin arrangement to make room for the outputs that would be required from the FPGA.

With the current configuration, we are using 26 of the 28 available I/O pins. It is possible for us to do without the XERR input from the 4 groups of LED controllers and possibly use a completely different error checking mode when we would like to receive that input. Removing the for XERR inputs frees up in total 6 pins for the text display. The text display requires at minimum 7 pins though to be implemented: XLAT, BLANK, GSCLK, SCLK, SIN_RED, SIN_GRN, and

SIN_BLU. Figure 5.5.b shows this pin configuration. In order to get around this issue, we will use the 6 available pins for all of the text displays input except for XLAT, which will be handled via other means. Figure 5.5.1.b shows this pin configuration, and notice the loss of the XERR pin inputs. I/O pins used for the text display will all be prefixed with T_ and can be seen on pins 12, 13, 25, 26, 27, and 28.

The issue of the missing pin for XLAT can be resolved by using the XLAT pulse from both the A and B columns tied together as input into the text display. In effect this will flash the Text Array at 44 frames per second, because it will receive 22 pulses per second from the A columns XLAT, and 22 pulses per second from the B columns XLAT at a 180 degree phase difference. It would also be possible to consider using the BLANK signal from A and B tied together as well, however the remaining 5 pins for GSCLK, SCLK, and RGB SOUT pins would remain necessary.

Pin Assignments with Text Display							
BB1		BB2		BB1		BB2	
Pin	Assignment	Pin	Assignment	Pin	Assignment	Pin	Assignment
I/O 1	ALL_VPRG	I/O 15	B_XLAT	I/O 8	AA_SIN_G	I/O 22	BB_SIN_R
I/O 2	ALL_DCPRG	I/O 16	B_GSCLK	I/O 9	AA_SIN_B	I/O 23	BB_SIN_G
I/O 3	A_XLAT	I/O 17	B_BLANK	I/O 10	AB_SIN_R	I/O 24	BB_SIN_B
I/O 4	A_GSCLK	I/O 18	B_SCLK	I/O 11	AB_SIN_G	I/O 25	T_BLANK
I/O 5	A_BLANK	I/O 19	BA_SIN_R	I/O 12	AB_SIN_B	I/O 26	T_SIN_R
I/O 6	A_SCLK	I/O 20	BA_SIN_G	I/O 13	T_GSCLK	I/O 27	T_SIN_G
I/O 7	AA_SIN_R	I/O 21	BA_SIN_B	I/O 14	T_SCLK	I/O 28	T_SIN_B

Figure 5.5.b Pin Assignments with Text Display Implementation

5.6 Power Supply Hardware Design:

The power for the POV display will come from a standard AC wall outlet. As shown on the hardware flow chart, we will divide the incoming AC power into two circuits. One will be used to power the stationary side of the POV display. The second circuit will be used to power the rotating side of the display. The choice to transfer power to the rotating side through the slip ring using AC instead of DC was in an effort to reduce the number of connections required. We will transfer the positive AC line through the slip ring. We will then terminate the neutral and ground wires together to the metal chassis. Since the chassis will be constructed out of aluminum, this should provide an adequate amount of conduction for the neutral. We will then pick up the neutral on the rotating side from the metal chassis. The only drawback to this will be with the bearing. However, since the bearing is constructed from metal as well, there should still be enough conductive material to allow the power to be transferred. Additional, since the slip ring will be bare conductor, we will step the incoming AC voltage down from 120 V to 24 V in an effort to reduce the potential danger associated with using exposed connections.

5.6.1 Stationary Power Supply:

The stationary power supply will be designed to power the motor and the stationary side FPGA board. We will use two rectifier circuits. One rectifier will be dedicated to power the power. The second rectifier circuit will be used to power the FPGA board and the Ethernet converter.

5.6.2 Rotating Power Supply:

The rotating power supply will be designed to power the LED array and the rotating side FPGA board. We will use one rectifier circuit dedicated to the FPGA board. We will then use two rectifier circuits to power the LED array. One rectifier for the Group A of LEDs and one for Group B LEDs.

5.6.3 Slip Ring Design:

In order to transfer power to the rotating side of the device we need a slip ring. This ring will consist of a copper washer attached to the shaft of the bottom section of the LED apparatus. Here a frayed copper wire will be mounted and just touching the washer. An insulating material will separate the washer from the shaft, and another wire will be connected to the inner side of the washer and a hole within the shaft will feed the wire from the washer up to the LED apparatus to power the system. Power will effectively be applied to the copper washer from the wire as the device rotates. There the power will travel from the washer to the rest of the device.

Figure 5.6.3 is a visual representation of this set-up created in DraftSight. Our construction of this device will be dependent on the requirements of the rotating apparatus Before we discuss exactly what is needed to create this slip ring we need to discuss these power requirements of the rotating apparatus. Since the basic element will be LEDs we need to figure out the power usage of a single LED and then the whole array itself. Since each LED requires about 100 mA to operate and will have a 5 voltage drop across them a single LED will require about 0.5 watts, multiply that by four-hundred-eighty LEDs and we have 240 watts. We will assume a high 30 watt requirement for the Xilinx Atlys Spartan 6 development board. However, it is best to give a 20% margin of error over the possibility of loss within the slip ring due to thermal dissipation. This puts us up to a total power requirement of 324 watts. Since we estimate to need about 24 volts to power the rotating apparatus we will also need around 13.5 amps. This means we need materials that can handle not only the 13.5 amps but both the 325 watts of power being transferred and whatever thermal power that will be applied to the slip ring as it is grinding against the washer. For this reason a 10 GA wire will be effective for this use. These wires are rated at a maximum current of 15 amps which is 1.5 amps above our needs, and its melting point is well above these power requirements.

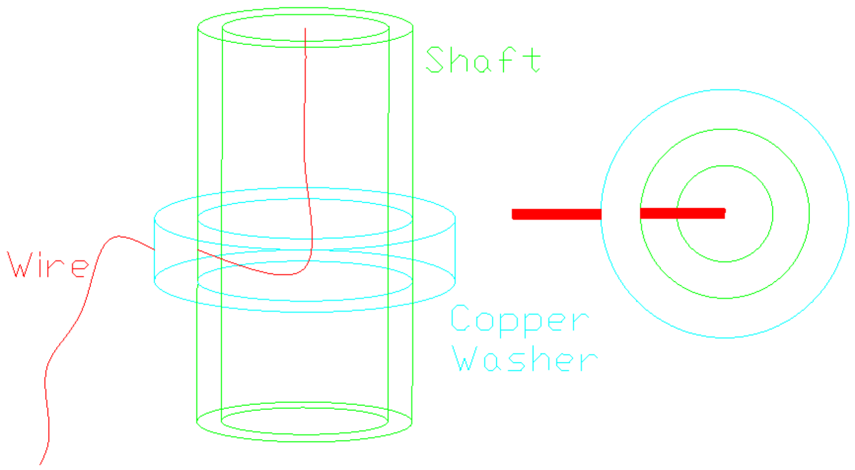


Figure 5.6.3 Slip Ring side and top view

On the opposite end of the slip ring, right before the rest of the device will be a bridge rectifier and a voltage regulator to convert from AC to DC and prevent over voltage. Figure 5.6.3 shows the circuit design of both of these.

5.7 Wired Ethernet Communications:

The POV display will require wired Ethernet communications to transfer the information from the stationary board to the rotating board. We determined that the Ethernet communications must be design to meet the standard 100 Mbps transfer rate. This is less than what was we initially thought would be required. Therefore, we will be using the Enconn EOC-AN and EOC-IN Ethernet over Coax Extender. Since only the EOC-AN requires additional power, the EOC-IN can be mounted on the rotating side of the POV display without requiring any additional inputs or power. Additional, since we will be using coaxial cable to transmit the Ethernet communications, we will be using the Model 205 rotary joint from Mercotac. Due to the increase rotation speed of the POV display, we must use the high speed models of the 205. Additional, to increase the life expectancy of the rotary joint we will be using the model with stainless steel ball bearings, part number 205-HS. To mount the rotary joint, we will attach a plate to the bottom of the support pipe on the chassis at the center of rotation. We will then run the coaxial wire coming from the rotating side of the joint up through the center of the pipe to the LED array.

6 Software Design:

There is a significant amount of software which must be designed for this project. We will be focusing on the software design for each individual part of the project here. This will include the design of the GUI application, the software functions of each of the FPGA's to receive input and display output, and the Ethernet communications between the two FPGA boards. These designs will serve as outlines for how we expect the implementation to be executed. These designs will not be a definitive method of implementation since many changes may come up during development. Figure 6 shows a simple overall flowchart of the communication between our software designs.

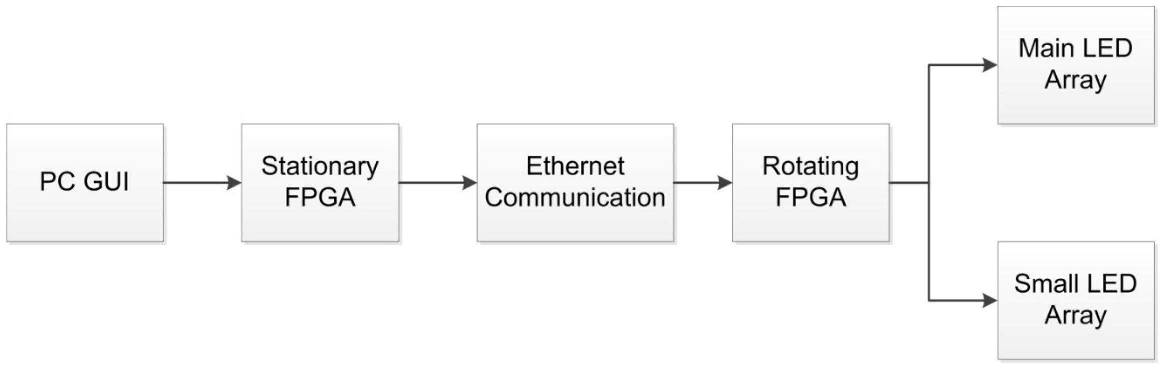


Figure 6 Software Flowchart

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.

We will be using the Atlys FPGA's for both the rotating FPGA's and stationary one. This is largely because of the real time requirements that 44 frames per second demands. The stationary FPGA must process an incoming HDMI frame buffer and convert it into a format that will be useful to the rotating FPGA. The rotating FPGA will dedicate its available clock cycles to reading this incoming frame buffer that arrived via Ethernet communication and output the data to the LED controllers on the LED array. Both FPGA's will be operating at 500 MHz.

The stationary FPGA will receive HDMI input frames at a rate of 60Hz into a frame buffer. It will also receive infrared sensor input and perform calculations to determine the rotating speed of the device, and send feedback to the sensor which will ultimately increase or decrease the rotating speed of the motor. The stationary FPGA also receives data via USB port which can include preprocessed images, processed text information to be displayed on the text display, and display commands which will then be passed to the rotating FPGA via Ethernet.

The rotating FPGA will receive the processed HDMI frames via Ethernet and begin writing the data to the LED controllers column by column and signaling for them to flash each time. There are very specific timing requirements under which the various signals from the FPGA to the Array must be sent. The software will make use of the Atlys boards 100 MHz clock from which sub clocks can be derived though the use of phase shifting and division, allowing for extremely finite timing. The rotating board must also control a small text array, and respond to various control signals telling it to start or stop displaying.

Our FPGA's will be communicating with each other using the built in Ethernet ports. This communication plays a key role in making this project feasible. We must create a design that is fast enough to transmit a live video feed. We most likely will not be using a standard Ethernet protocol because of the unnecessary overhead involved. In our software design for Ethernet communications we will decide on our communication protocols and outline our method of sending the data. We will be viewing the Ethernet communication as a hardware interface between the two boards. When programming an FPGA it is often advantageous to view the design as a hardware design instead of a software design to complement the Verilog HDL.

6.1 Primary Microcontroller Software Design (Stationary):

This section will cover the software design requirements for the stationary FPGA board. Software requirements include processing the incoming HDMI signal and formatting the data to be transmitted via Ethernet to the rotating FPGA board.

6.1.1 Processing HDMI Signals:

The Atlys board has several registers used to initialize HDMI input and view the status of the connection. Figure 6.1.1 contains detailed bit information and a description of these registers. Upon plugging a source into the HDMI port of the FPGA, the ICC core, which is dedicated to handling HDMI input and output, will begin receiving interrupts with the HDMI source looking for EDID information about the FPGA, which the source views as a display. An interrupt service routine will send EDID information, stored in DDR2 memory whenever this interrupt occurs. The status register's Bit(30) will read a 1 if the frame dimensions of the incoming video are determined.

Before the flow of HDMI input frames can begin being received, both the Frame Base Address and the Line Stride registers must be set. The Frame Base

Address is the starting location in the DDR2 memory where the frames will begin to be stored. The Line Stride is defined as the number of pixels in a single horizontal line of a frame, in our case 640.

Control Register:	0x00 : R/W
Bits (0 : 30)	Reserved
Bit(31)	Write Enable – Enables the core to begin writing video data to the frame buffer. Default is '0'.
Status Register:	0x04 : R
Bits (0 : 29)	Reserved
Bit(30)	Frame Locked – Reading a value of '1' means that the frame dimensions of the incoming video signal have been determined.
Bit(31)	PLL Locked – Reading a value of '1' means that a valid clock signal is detected on the TMDS lines.
Frame Width Register:	0x08 : R
Bits (0 : 15)	Reserved
Bits (16 : 31)	Unsigned value that represents the width of the input frame in pixels
Frame Height Register:	0x0C : R
Bits (0 : 15)	Reserved
Bits (16 : 31)	Unsigned value that represents the height of the input frame in lines, minus 1
Line Stride Register:	0x10 : R/W
Bits (0 : 15)	Reserved
Bits (16 : 31)	Unsigned value that defines the line stride of the frame in pixels (described below). This value must have bits 26 to 31 equal to zero in order to be 128 byte aligned. Default value is 0 and must be set before enabling the core.
Frame Base Add. Reg.:	0x14 : R/W
Bits (0 : 31)	Unsigned value that defines the physical address of the frame buffer. This address must fall somewhere within the DDR2

	memory space and have enough trailing memory to fit the entire frame. This value must have bits 25 to 31 equal to zero in order to be 128 byte aligned. Default value is 0 and must be set before enabling the core.

Figure 6.1.1 HDMI Registers

6.1.2 Frame 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 6.1.2.a 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 6.1.2.a 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 6.1.2.b 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 6.1.2.b 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. Figure 6.1.2.c 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)

Figure 6.1.2.c Arrangement of Pixel Coordinates in a Frame

6.1.3 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. Figure 6.1.3.a 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							
1							
.	AA	BA	AA	BA	…	AA	BA
.							
239							
240							
241							
.	AB	BB	AB	BB	…	AB	BB
.							
479							

Figure 6.1.3.a 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. Figure 6.1.3.b 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

Figure 6.1.3.b Memory locations of the 12 output Bins

6.1.4 Frame Processing:

This section will cover the various steps involved with converting a frame from the input HDMI format to the output format that we have specified. 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. Figure 6.1.4.a 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		

Figure 6.1.4.a: 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)

Figure 6.1.4.b 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. Figure 6.1.4.c 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.

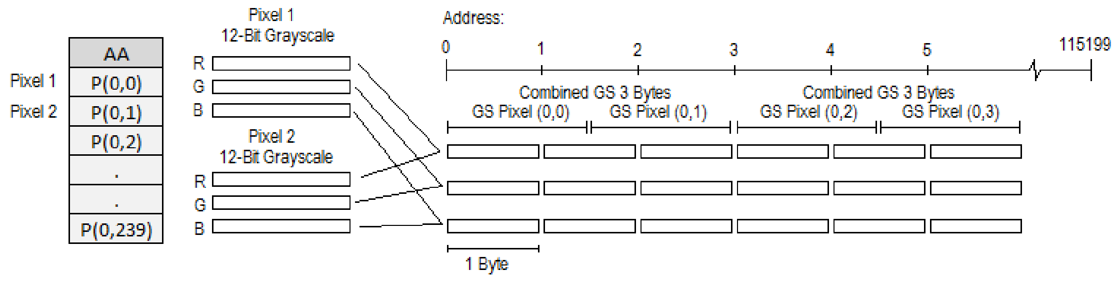


Figure 6.1.4.c Combining Grayscale values and storing in memory

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 data, Figure 6.1.2.c in the section HDMI Input and Frame Buffer Format shows the 16 bit RGB configuration. 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 6.1.4.d.

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.

Two Byte RGB color:

R5	R4	R3	R2	R1	G6	G5	G4	G3	G2	G1	B5	B4	B3	B2	B1

First Byte Obtained:

R5	R4	R3	R2	R1	G6	G5	G4

Logical Right Shift x 3:

0	0	0	R5	R4	R3	R2	R1

Figure 6.1.4.d 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 logcal 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 6.1.4.e 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.

Two Byte RGB color:

R5	R4	R3	R2	R1	G6	G5	G4	G3	G2	G1	B5	B4	B3	B2	B1

First Byte Obtained in Temp:

R5	R4	R3	R2	R1	G6	G5	G4

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 6.1.4.e 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 6.1.4.e provides a visualization of the logic for isolating the blue RGB value from the pixel data. Figure 6.1.4.f shows a visualization of this function.

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 6.1.4.f Visualization of Isolating Green 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 5.1.1.3.g shows the mapping of values for both Red/Blue and Green.

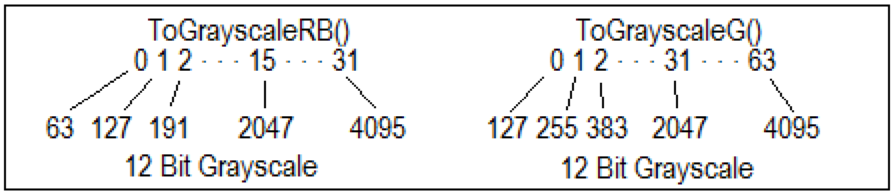


Figure 6.1.4.f Grayscale Mapping Diagram

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 6.1.4.h.

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 6.1.4.g Visualization of Combining Grayscale Values

6.1.5 Stationary FPGA Ethernet Communications:

The stationary FPGA will have to send messages to the rotating FPGA. For this we will be using the built in Ethernet ports. Although we are using the Ethernet ports we will not be using a standard Ethernet protocol. We will be designing our own simple communication protocol that it 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: video, image, text (main), text (small), command, and sensor data. With six different possibilities we will need 3 bits for the header. The first three bits of each data stream will be our header and they will be assigned as shown in Table 6.1.5

Header Bits	Data Type
000	Video
001	Image
010	Text (Small)
011	Command
100	Sensor Data

Table 6.1.5 Header Information

As seen in Table 6.1.5 there is no text data type for the main display. This is due to the fact that when text is sent, it will be sent in the same format as an image. 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 image or video. Even though the data type is labeled text, it can also be an image. The header simply decides which display to send the data. The video header will herald an incoming stream of frames. To signal the end of a video feed there will be a separate signal that we will call CommEnable. When this signal is high there will be active communication, and when this signal is low, communication will cease. The video feed will be the only data type that will require us to specify when the data ends. When the CommEnable signal goes low during a video feed then the FPGA will clear the screen image. All non-video data types will have a fixed size that is expected to be sent. These data types with fixed sizes will hold their image when the CommEnable signal goes low. The image data type will be a single frame that will be displayed constantly on each rotation. A command signal can include commands to clear the main screen image or the text screen image. Lastly sensor data is from the IR sensors that will detect each revolution of the display. This data will help control the motor speeds. A simple Ethernet transmission is shown in Figure 6.1.5. The figure assumes the appropriate header is added either from the PC application or from the HDMI processing.

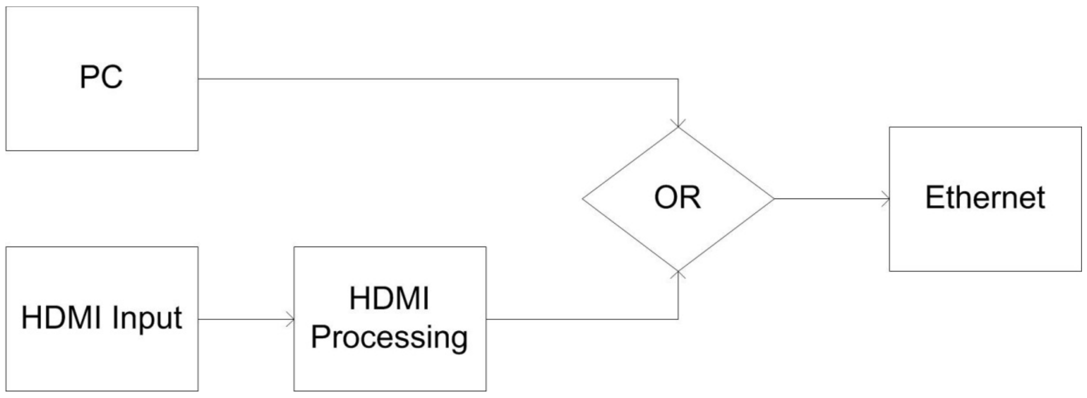


Figure 6.1.5 Ethernet Transmission Flowchart

6.1.6 Motor Control Sensor Software:

Software will be required to process the information from the motor control circuit. The software is required to read in these voltage pulses and determine the revolutions per second of the device through this data. It will also then need to make a decision of whether to maintain, decrement, or increment the speed of the motor based on this device. This will all have to be done with software within the processor. Figure 6.1.5 best shows the flow of this process. In this process when a pulse is detected the current value of time tc will be subtracted from the previously recorded clock value tp. This value will then be stored as the time between that pulse Tn. Once a second passes each of these pulse values will then be averaged out using the following equation: [T1+T2+...+Tn]/n = Ta. This will be compared with the expected Ta value with a 1% allowable error margin. Since we want to have around thirty revolutions per second we need a 0.033 period. So this Ta needs to be within 0.03267 and 0.03333. If it is above this margin then we will have to decrement the resistance in the pulse width modulator and if it is below this margin we will have to increment the resistance in the pulse width modulator. Our incremented and decremented value should be around 1%. This should be enough that we won't overshoot our margin if we increment or decrement just outside the margin.

When we compare our Ta we will have to make one of three choices based on this comparison. Our choice will be used to send out a voltage signal to a voltage controlled resistance that will raise, decrease, or maintain its value. This change in resistance will trigger the PWM circuit to change the speed of the motor by 1%.

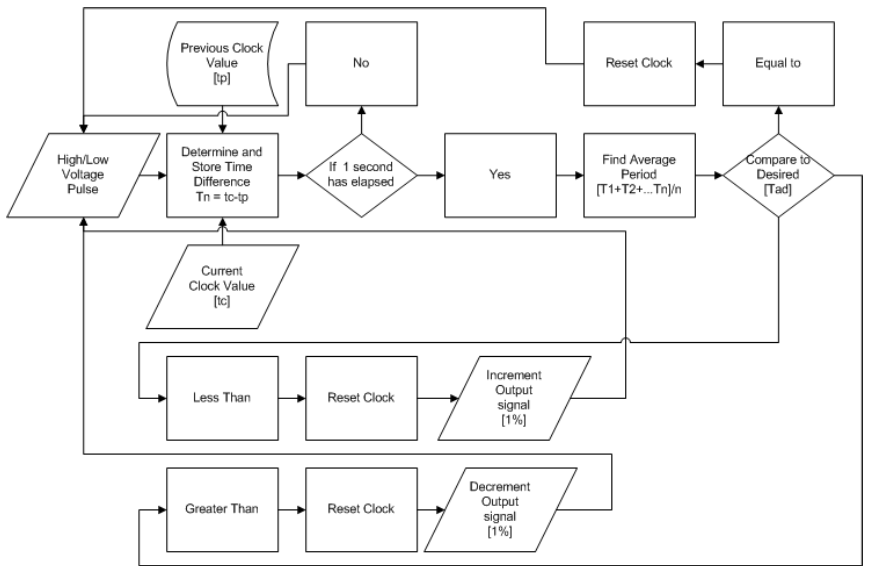

Previous Clock Value [tp]
No
Reset Clock
Equal to
High/Low Voltage Pulse
Determine and Store Time Difference Tn = tc-tp
If 1 second has elapsed
Yes
Find Average Period [T1+T2+...Tn]/n
Compare to Desired [Tad]
Current Clock Value [tc]
Less Than
Reset Clock
Increment Output signal [1%]
Greater Than
Reset Clock
Decrement Output signal [1%]

Figure 6.1.6 Control Program Flow Chart for Motor Control

6.2 Secondary Microcontroller Software Design:

This section will cover the software design requirements for the rotating FPGA board. Software requirements include reading the preprocessed data and HDMI frames arriving at the board via Ethernet connection, and outputting this data to the LED array.

6.2.1 Modes of Operation:

The secondary controller will have various modes of operation and will also be receiving various data and commands via Ethernet from the stationary controller. Data includes HDMI 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 will also be received which is used to determine when to start displaying the device, giving it a tap dead center. Additionally, commands will be received that change the operation modes of the POV display.

The main LED array can either be displaying data from the HDMI buffer or data that corresponds to a single image that has been stored in memory. The main array output can either be in HDMI_MODE, 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

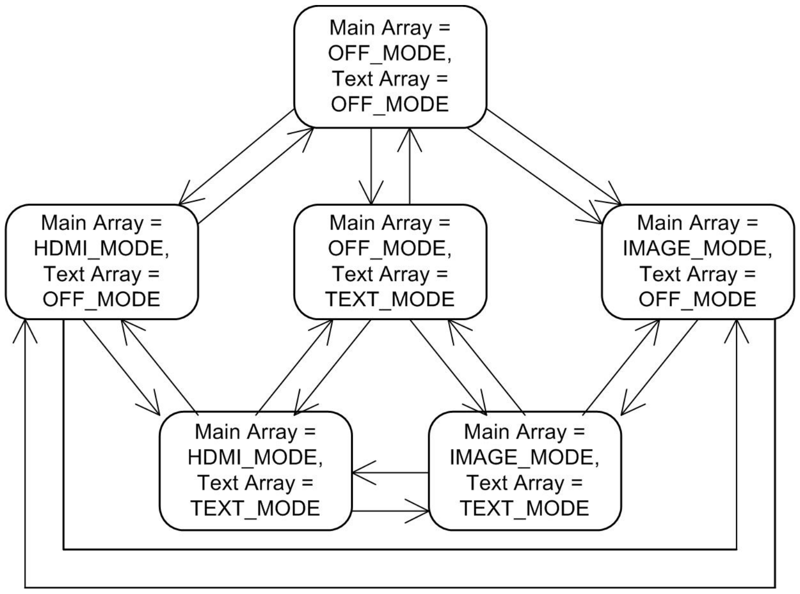

Main Array = OFF_MODE, Text Array = OFF_MODE
Main Array = HDMI_MODE, Text Array = OFF_MODE
Main Array = OFF_MODE, Text Array = TEXT_MODE
Main Array = IMAGE_MODE, Text Array = OFF_MODE
Main Array = HDMI_MODE, Text Array = TEXT_MODE
Main Array = IMAGE_MODE, Text Array = TEXT_MODE

Figure 6.2.1 State Chart for Modes of Operation

6.2.2 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.

6.2.3 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.

6.2.4 Rotating FPGA Ethernet Communications:

The rotating FPGA will receive communications through the built in Ethernet port. 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 will be a 3 bit register to hold the header data. This register will become active at every positive edge of the CommEnable signal. After the positive edge of the CommEnable signal the header register will receive the data stream for three clock cycles to obtain the header information. Once the first three data bits have been read into this register the FPGA will then know what kind of data is about to be sent. This should only happen at the beginning of a data stream which is why the positive edge of the CommEnable signal should suffice. Figure 6.2.4.a shows a simple three bit register that will obtain the header information. The actual implementation will be done using Verilog HDL.

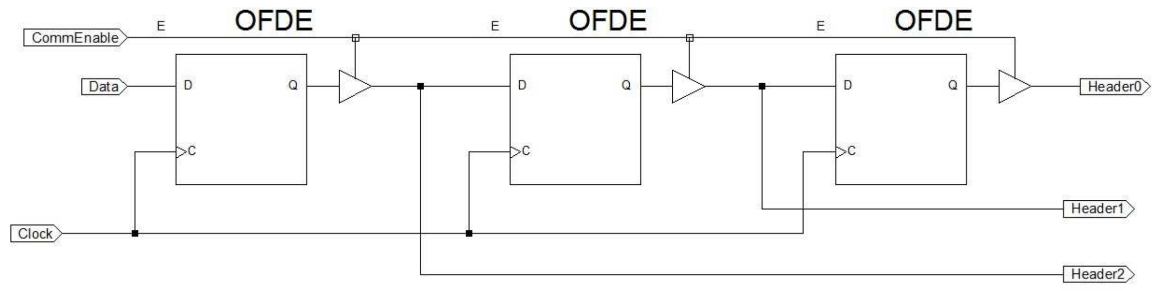


Figure 6.2.4.a Simplified Ethernet Header Register

If we use an implementation as shown in Figure 6.2.4.a then we will have to read the header information exactly on the fourth clock cycle. If this turns out not to be possible we can add multiplexers to the inputs that will have the register hold the values so that they can be read at a later time. The three header bits will then be sent to a comparator which will test the values against each of the acceptable values shown in Table 6.1.5. Once the header has been decoded the rotating FPGA will then expect a certain amount of data. Video data will be expected constantly until there is a new positive edge on the CommEnable signal. In the case of an image file being sent the FPGA will expect 640x480x8 bits of data or 2457600 bits. After the required number of bits has been read the FPGA will hold the image on the display, and will not expect any more data until the next CommEnable positive edge. The text signals will be similar to the image signal. For the main display the text signal will have the same number of bits as the image signal. This is because all of the pixel data will still be sent. For the small text display the data required will be 640x16x8 or 81920 bits. For the command signals the number of bits will be very small in comparison. The commands will be hard coded and we will assign 4 bits for commands. We do not need 4 bits for now with only two clear commands, but this will allow us to add more commands without making many changes to our design. Figure 6.2.4.b shows a simple flowchart detailing Ethernet communications on the receiving end. The header register in Figure 6.2.4.a will have its outputs fed into the header comparator shown in Figure 6.2.4.b.

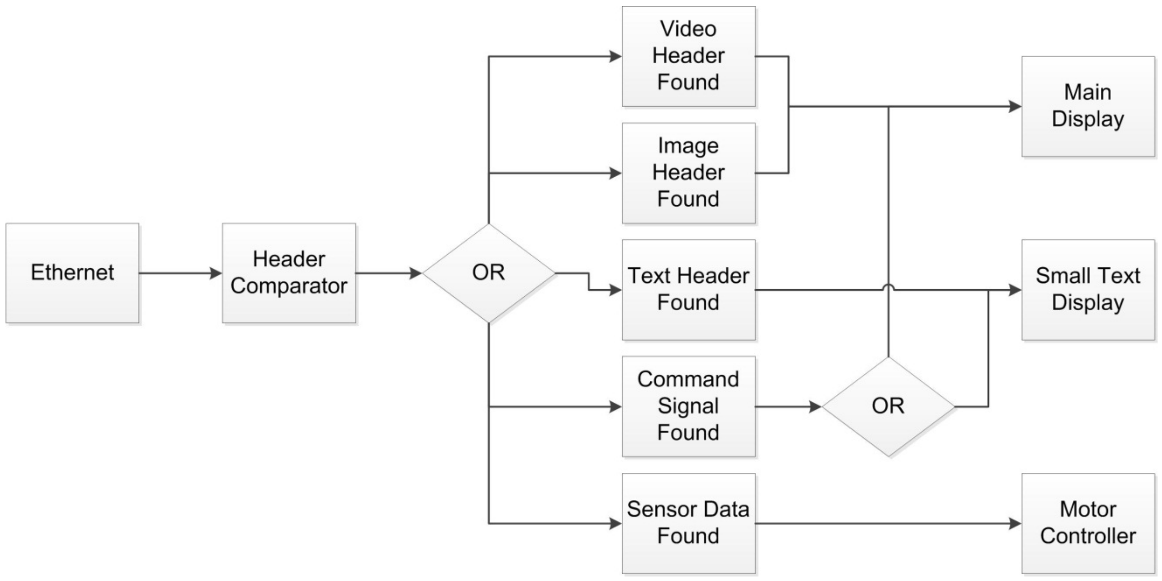


Figure 6.2.4.b Ethernet Receiving Flowchart

6.3 Computer GUI Software Design:

The GUI may be implemented on a PC, an android device, or possibly both. This design will focus on designing the GUI for a PC but regardless of the device the GUI is implemented on, the design remains the same. The following bulleted list will show a formal enumeration of the requirements which must be 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
Communications port selection
Loading bar or visual progress indicator

A pipe and filter architecture will be suitable for this application. The user input will 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 better illustrates the pipe and filter model we will be using:

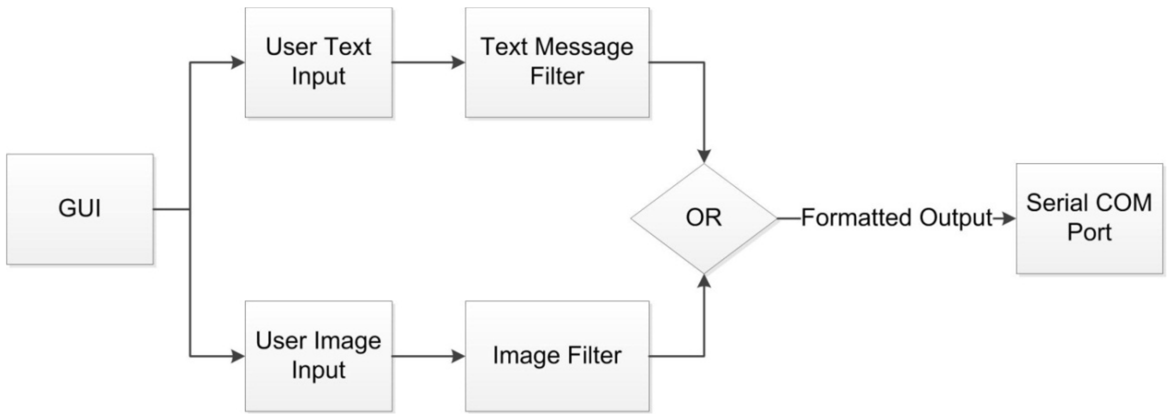
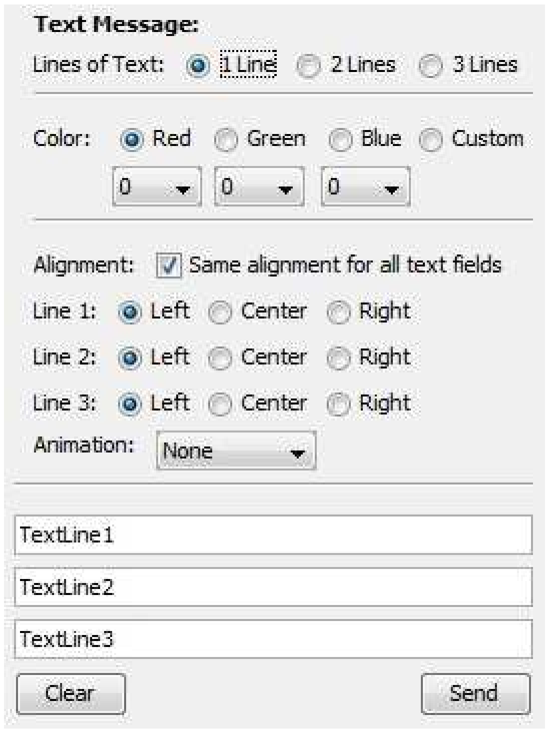


Figure 6.3 Pipe and Filter Software Architecture

The GUI will be designed using Java in the Netbeans IDE. This IDE was chosen for its robust GUI editor which will allow us to quickly create an interface before completing any coding. Designing the interface first also helps to serve as an outline to facilitate the implementation. Creating the first visual draft of the GUI's appearance will be the next step in the design. We will be considering 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.

6.3.1 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. Near the text input areas there should be obvious labeled color options, which should include preset colors as well as user defined colors. These colors will be applied to the entire text message. It should also be noted that if the user were to choose black as the text color (RGB values all zero) then we may consider having the background show up as white. This is a special case and we are not considering allowing the user to choose a background color at this time. The text should also have alignment options to determine the position of the text on the display. These options should include left, center, and right alignments. The alignment options should also be applied to all or some of the lines of text. Changes in the alignment options should also be directly visible in the GUI to help the user visualize how it will appear on the display. The initial GUI design for text input is shown in the following Figure 6.3.1. This initial design takes all of the previously mentioned requirements for text input into consideration.


Text Message:

Lines of Text: [✓ 1 Line] 2 Lines 3 Lines

Color: [✓ Red] Green Blue Custom

0 0 0

Alignment: [✓ Same alignment for all text fields]

Line 1: [✓ Left] Center Right

Line 2: [✓ Left] Center Right

Line 3: [✓ Left] Center Right

Animation: None

TextLine1

TextLine2

TextLine3

Clear Send

Figure 6.3.1 Text Input GUI Draft

The text input will be stored in a string array. The string array will have three index values from 0-2 representing the line number for that string. This information will then be sent to the FPGA using the RxTx library. The text will actually be converted into an image that the FPGA can interpret. The information will be 640x480 pixels sent one at a time, with the text message lines inserted where appropriate. The text lines will be displayed within rows of 150 pixels which is slightly less than 480/3. These rows will be 150 pixels high in order to allow a gap between the lines of text to make it look better. Each row will then be divided into a certain number of columns. We will wait to decide on how many columns to divide the pixel rows until we have the display working. We will experiment with different values until we find the greatest number of characters we can fit without having the characters look distorted. Since each row will be divided into a set number of columns, we will have a set number of characters per line. This will simplify the formatting since we can hard code each letter. There will be a method for each letter which will return an integer array representing the pixel data for that letter. There will be an array of letters for each line of text. Finally when the data is sent, the pixel data will be sent from each letter array in the proper order.

6.3.2 Image Input:

The image input option will allow the user to select an image from the hard disk to display. The Image input should appear 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 should be 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 primary focus during implementation will be to have the image selection only work for images of the proper size. Depending on the complexity of image processing, other sizes may be supported as well. 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 should be the position where the image I displayed, this is for images that are too small. There should be nine choices available in a box shape from top left to bottom right. Images will be sent to be displayed on top of each other. If there is a small image sent to the bottom left portion of the display, and then another small image is sent to the top right, both images should be visible simultaneously. Because of this a clear button will be necessary to clear the image on the display. 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 draft shown in figure 6.3.1 and the image input draft in figure 6.3.2.

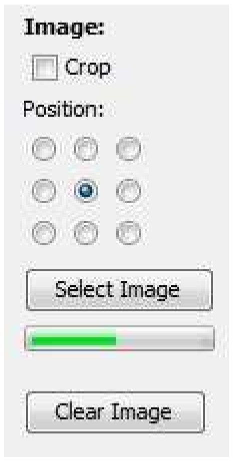
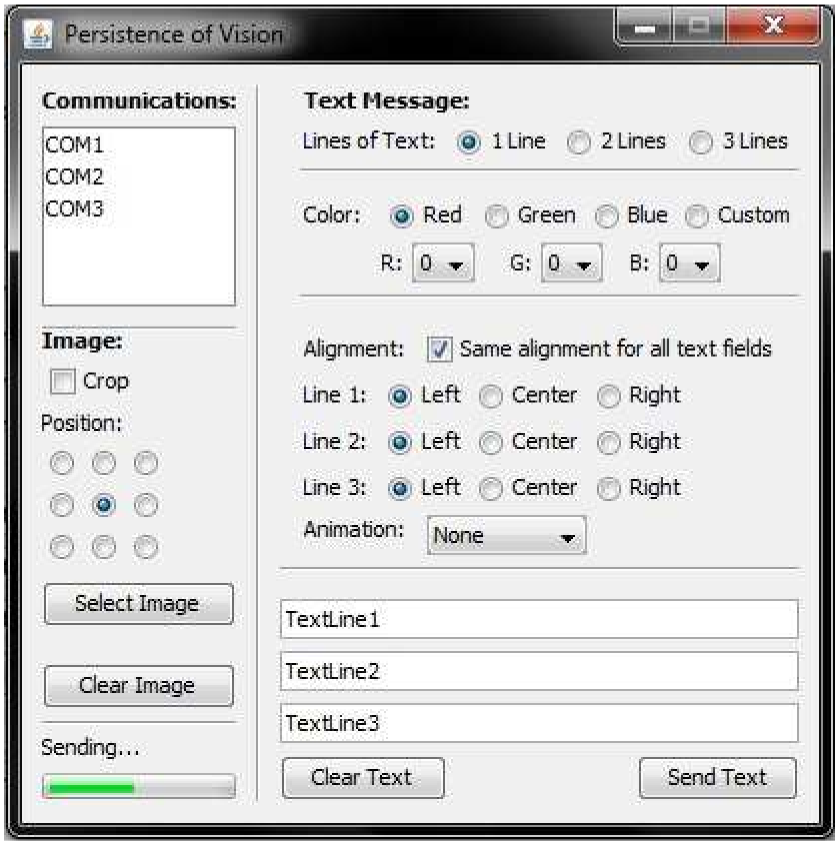


Figure 6.3.2 Image Input GUI Draft

The Image will be read using built in classes and methods for image handling. These classes include ImageIO, BufferedImage, and ImageReader. We will be using an ImageReader to interpret the image format and translate it. This will allow ImageIO to read the image into a BufferedImage. We will then use the getRGB method of the BufferedImage class to get the pixel color values. The pixel color values should be very easy to convert to our format (if any conversion is necessary at all) so that we can send the image using serial communications. Any blank pixels (no image data) will be represented with a special value that will let the FPGA know not overwrite any previous value in the FPGA memory for that pixel. This will allow 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 will be sent to the FPGA. This can be done by sending an entirely black image, or by a unique clear signal that will have the FPGA overwrite all pixels with black values. The loading bar shown in Figure 6.3.2 will most likely be used for sending text messages and images. The loading bar will be updated through the USB communications which will be discussed next.

6.3.3 FPGA GUI Communications:

In this section we will consider how to communicate with the FPGA using serial communications through the USB port. We will also be considering how to properly format the user input so that the FPGA has all of the necessary data to update the display. We will be using the Java RxTx library to do USB serial communications between the PC and the Atlys board. The first step in serial communications is enumerating the available serial communication ports. This will add another element to our GUI, either list box or a combo box that will list all available ports. The user should be able to select which port to use when attempting to send data. The selected port will be stored in a private string named "port". This brings us to the GUI design draft that combines all requirements discussed so far. The following Figure 6.3.3 shows the first complete GUI design draft.



Lines of Text: [✓ 1 Line] 2 Lines 3 Lines

Color: [✓ Red] Green Blue Custom

R: 0 G: 0 B: 0

Alignment: [✓ Same alignment for all text fields]

Line 1: [✓ Left] Center Right

Line 2: [✓ Left] Center Right

Line 3: [✓ Left] Center Right

Animation: None

TextLine1

TextLine2

TextLine3

Figure 6.3.3 Complete GUI Design Draft

6.3.3.1 Serial Communication Thread:

Serial communications should be done in a separate thread than the rest of the program. This is due to the fact that serial communications is a type of blocking I/O. This means that if the communications took place in the same thread as the main program then when communication is taking place the entire program may hang while waiting for I/O. The most common method used to solve this problem is to handle the serial communications in a separate thread. This can be done by creating a new class that extends the built in Thread class. We will be creating a class called USBComm which will extend the Thread class. Any class that extends the Thread class must have a run method. The run method will execute within a new thread when the start method is called. We will have to put all communication code within the run method of the USBComm class and start the thread each time we want to send a data payload to the FPGA. A new instance of the USBComm class will be created each time communications are needed and the arguments passed into the constructor will include the data to be sent. We will also only allow one thread (besides the main program) to exist at a time, this will prevent multiple messages being interleaved in the serial communication stream. For this we will have to create a Boolean variable to indicate the existence of a current active communication stream.

6.3.3.2 Serial Communication I/O:

The serial communications I/O operations will take place within the USBComm class. The USBComm class will contain instances of the serial communication classes from the RxTx library. These classes from the library will handle the communications and allow the USBComm class to send a payload to the FPGA, as well as receive acknowledgements. The acknowledgements received will allow the updating of a loading bar. A simple sequence diagram shown next in Figure 6.3.3.2.a should help to illustrate the planned data flow for the USBComm class.

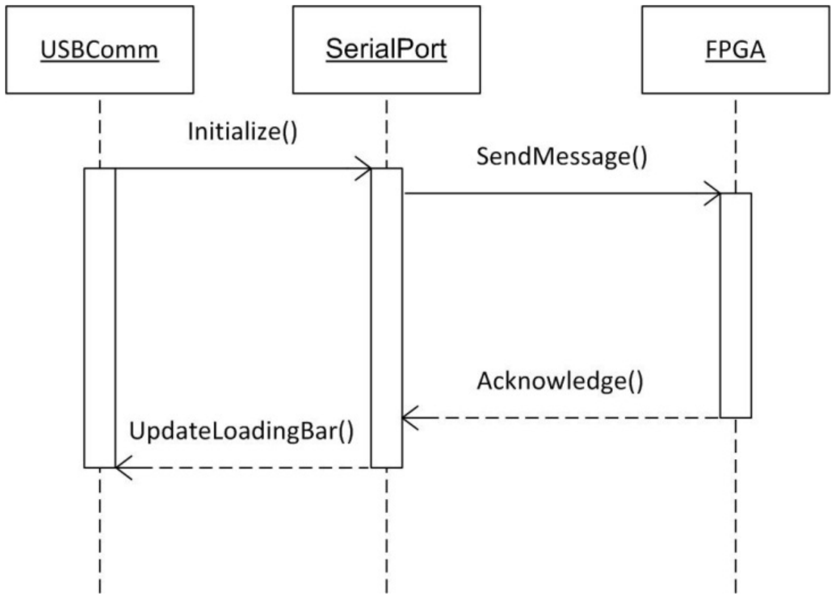


Figure 6.3.3.2.a USBComm Sequence Diagram

Before serial communication can occur the port must be acquired and initialized. In order to acquire the port, the open method must be called on the SerialPort object. The open method will throw a PortInUseException if the port is already in use by some other process. Once the port has been opened it can then initialized. Initializing a port includes setting the initial values for baud rate, data bits, stop bits, and parity. These values will most likely be hard coded to the best settings for the Atlys board. After the port initialization we will have to initialize a ReadStream and PrintStream object in order to serve as the input and output handlers of the serial communication. The ReadStream and PrintStream objects will be initialized using the SerialPort's getInputStream and getOutPutStream methods. After the streams are initialized then serial communication can finally begin. Simply print items to the PrintStream object and read items from the ReadStream object. When all reading and writing is complete, the port 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 resourced devoted to the streams will be freed. The following Figure 6.3.3.2.b will illustrate how to use each of the serial communications related classes and methods described so far.

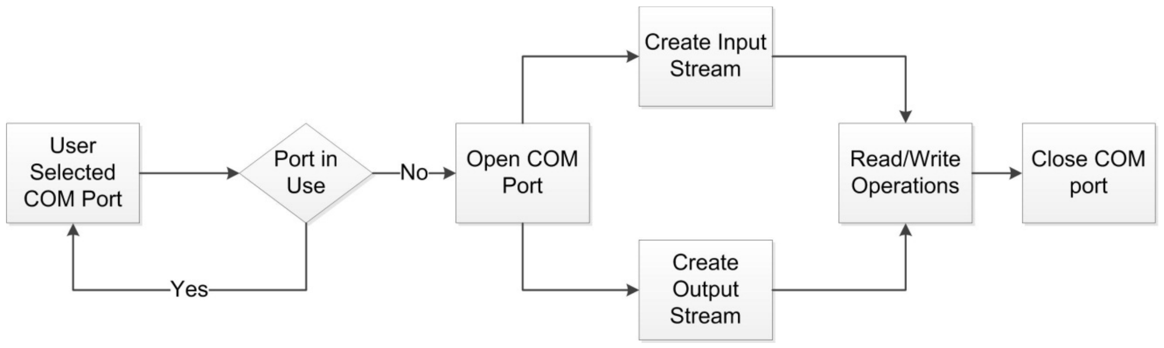


Figure 6.3.3.2.b

6.3.4 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, USBComm, 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 USBComm class will need to contain classes from the RxTx library including SerialPort and CommPortIdentifier. The SerialPort will also contain PrintStream and ReadStream classes for the I/O operations. A class diagram showing these classes and their relationships to each other is shown in the next figure. 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.

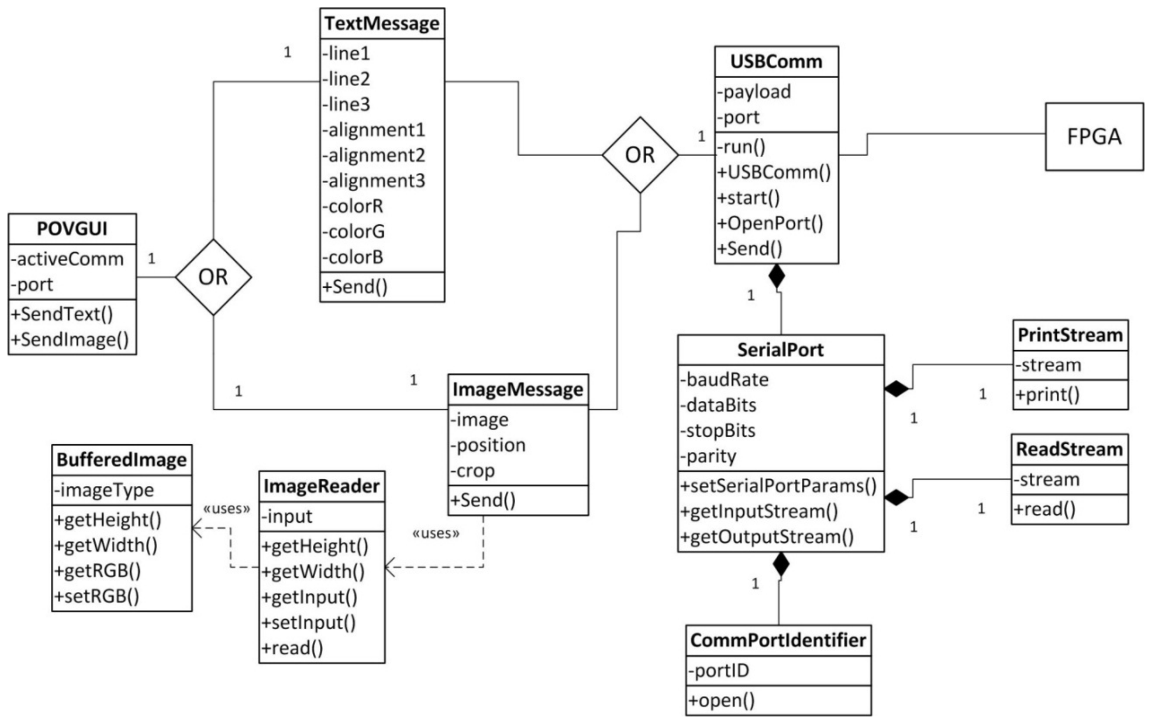


Figure 6.3.4 Class Diagram for the GUI

7 Prototyping:

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

7.1 Slip Ring Power Transmission Prototype:

While testing the slip ring we will need a prototype circuit that can be used to tell if the slip ring is properly transferring the amount of power we need 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 three 120 watt bulbs. Since the expected amount of power needed on the opposite side of the device is 324 watts of power then if the slip ring can power while in motion 360 watts worth of power then we know that we should have no problem powering the 324. In addition, we can tell what the minimum amount of power is needed to power the

360 watts so we can get an idea of the loss in the system. The circuit for this prototype is shown in Figure 7.1.

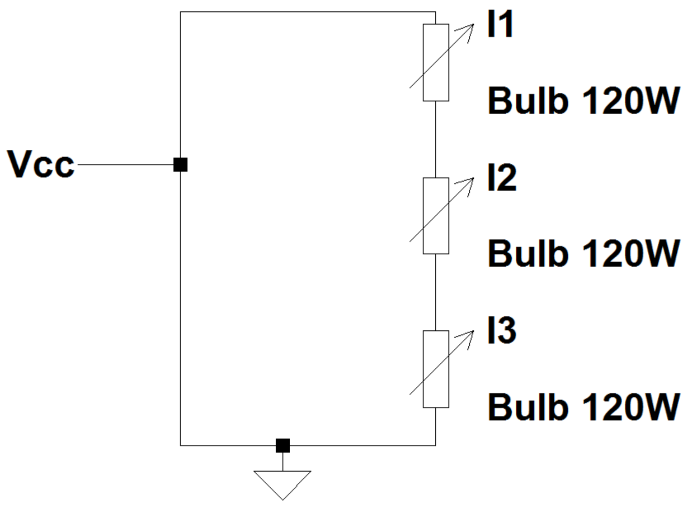


Figure 7.1 Power Transmission Prototype

7.2 Scaled LED Array Prototype:

In order to verify our LED array design works, we will build a prototype of the LED array. We will build a LED array for 16 RGB LEDs controlled by LED controllers.

7.2.1 Scaled LED Array Hardware Prototype Design:

The design for the LED array prototype will be similar to the full scale LED array. The only difference will be that number of cascaded LED controllers. For the prototype version of the LED array will not have any controllers cascaded. Although this will not test the speed at which we will be able to address the cascaded controllers, f(sclk), this will provided a test of the grayscale clock, f(gsclk).

7.2.2 Scaled LED Array Software Prototype Design:

We would like to use the Atlys board to send data to the prototype LED array and have the LED array display various different test data. One of the tests would have the LED array display its full range of color values in an endless loop. The FPGA will need to write to each of the 3 LED controllers in the prototype controlling red, blue, and green, the grayscale data that will correspond to every RGB combination. In our actual device we only have 16 bits worth of color depth since we use image frames as a source, but for this prototype we will be able to use 24 bits of color depth since the 12 bit grayscale values will not be translated from RGB data. This amount of color depth is known as true color and 16777216 unique colors will be displayed by the LEDs. We can vary the color depth and attempt to determine when the color transition loses its smoothness.

In this example Pseudocode, the colordepth can be specified and using 3 for loops every color combination can be iterated through. Determining the grayscale values would depend on the colordepth and a function that determines those values is called on lines 6, 7 and 8. A function used for writing to the LED controllers would be called with the address for the pin to write to as an argument as well as the grayscale value to write. Write() would write that grayscale value 16 times since each of the controllers is connected to 16 LEDs. Finally on line 12 the display() function is called, which will wait for a clock pulse before proceeding to flash.

```
1  while(1) {
2  int r = 0, g = 0, b = 0;
3    for(r=0; g<colordepth/3; r++) {
4      for(g=0; g<colordepth/3; g++){
5        for(b=0; b<colordepth/3; b++) {
6          redgrayval = tograyscale(r, colordepth)
7          grngrayval = tograyscale(g, colordepth)
8          blugrayval = tograyscale(b, colordepth)
9          write(red_controller, redgrayval)
10         write(grn_controller, grngrayval)
11         write(blu_controller, blugrayval)
12         display()
13       }
14     }
15   }
16}
```

Another test would involve varying the flashing speed of the LED array, and use that information to determine if the LED controllers can indeed be updated at the rate we need them to. In order to display the 640x480 image, each controller will need to be able to update 320 times at 44 frames per second, or 14080 times per second. We may be able to use this information to determine if we can display multiple copies of the same frame, which would require that the controllers flash the LEDs at a faster rate. There is also a limit as to how fast the controllers themselves can be written to and we can attempt to transfer data at this limit and even exceed.

8 Testing:

Since like any project it is unlikely to work exactly like we designed it the first time we turn the device on it is 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 will also help us calibrate certain components of the device so that they work effectively.

In the following sections of this chapter there are a series of tests that will be 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 desired results and the modifications to the design that may occur based on the results of the test.

8.1 Video Signal Processing Testing:

The primary objective of the video processing on the stationary microcontroller is to turn frames of RGB data into frames of Grayscale data as described in section 6.1.1.2 Output Format Specification. Specifically, RGB frames consisting of 640x480 pixels, each 2 bytes, are to be processed and turned into grayscale data which is then deposited into 12 bins. For each of the four sections AA, AB, BA, and BB, there are 3 bins for the RGB grayscale data. In order to test the various functions used in the video processing, we will create a test frame with known RGB values and process it. The memory contents of the processed image can then compared to a table of expected contents. Figure 8.1.a shows the layout of the test frame. The test frame contains 10 sections of 2 byte RGB colors and has been designed to test the various functions used in video processing.

Video Processing Test Frame					
Pixel	0 - 127	128 - 255	256 - 383	384 - 511	512 - 639
0 1 . . 239	Section 1: 0xFFFF	Section 3: 0x0000	Section 5: 0xF800	Section 7: 0x07E0	Section 9: 0x8010
240 . . 478 479	Section 2: 0x0000	Section 4: 0xFFFF	Section 6: 0x001F	Section 8: 0xFD00	Section 10: 0xFFE0

Figure 8.1.a Video Processing Test Frame Layout

Sections 1-4 of the test frame will test frame which are solid white and solid black will test the TranslateAndOutput() function which is given the pointers of the output bins it should write to and handles converting pixel by pixel. This function is called 4 times in each loop of the main TranslateFrame() functions, each time converting either an AA, AB, BA, or BB section of pixels. The expected memory output from converting these 4 sections can be seen in Figure 8.2.b, from memory offset addresses 0 to 46079.

Sections 5 6 and 7 of the test frame are the colors solid red, blue, and green, and verify that each of the functions GetRedCombinedGS(), GetBluCombinedGS(), and GetGrnCombinedGS() each successfully isolate their respective colors from the RGB pixel data. The memory contents for converting section 5 can be seen in the expected memory contents table in the address range 46080-69119 in the AA and BA bins. Similarly, the results for section 6 can be seen in the memory range 46080 – 69119 in the bins for AB and BB. Section 7 results are located at memory addresses 69120-92159 in bins AA and AB.

The final 3 sections, 8 9 and 10 are composite colors each containing a mix of RGB values. These sections will further ensure that RGB colors are being isolated correctly, and also test the CombineGrayscale() function which combines two 12 bit grayscale values into 3 bytes. For example, in section 8 the grayscale memory contents for AB_GRN in the 69120 – 92159 memory range is 0xA3FA3F, which are the bytes A3 FA and 3F. This sequence of 3 bytes is the result of combining 0x0A3F with 0x0A3F.

Expected Memory Contents						
		Memory Offset				
		0 - 23039	23040 - 46079	46080 - 69119	69120 - 92159	92160 - 115199
Bin	Start Add.	⬜	⬛	🟥	🟩	🟪
AA_RED	0	0xFFFFFF	0x000000	0xFFFFFF	0x000000	0x87F87F
AA_GRN	115200	0xFFFFFF	0x000000	0x000000	0xFFFFFF	0x000000
AA_BLU	230400	0xFFFFFF	0x000000	0x000000	0x000000	0x87F87F
BA_RED	691200	0xFFFFFF	0x000000	0xFFFFFF	0x000000	0x87F87F
BA_GRN	806400	0xFFFFFF	0x000000	0x000000	0xFFFFFF	0x000000
BA_BLU	921600	0xFFFFFF	0x000000	0x000000	0x000000	0x87F87F
Bin	Start Add.	⬛	⬜	🟦	🟧	🟨
AB_RED	345600	0x000000	0xFFFFFF	0x000000	0xFFFFFF	0xFFFFFF
AB_GRN	460800	0x000000	0xFFFFFF	0x000000	0xA3FA3F	0xFFFFFF
AB_BLU	576000	0x000000	0xFFFFFF	0xFFFFFF	0x000000	0x000000
BB_RED	1036800	0x000000	0xFFFFFF	0x000000	0xFFFFFF	0xFFFFFF
BB_GRN	1152000	0x000000	0xFFFFFF	0x000000	0xA3FA3F	0xFFFFFF
BB_BLU	1267200	0x000000	0xFFFFFF	0xFFFFFF	0x000000	0x000000

Figure 8.1.b Expected Memory Contents of Processed Test Frame

If the memory contents of the processed test frame exactly match the data in the Expected Memory Contents table then we can be pretty confident that the TranslateFrame() function is working as desired. If we find that the contents differ, we will be able to approximate the cause of the problem based on where in memory the differences are occurring.

8.2 LED Array Testing:

We will be testing the LED array using the same frame layout as described in the test section for Video Signal Processing. After the we complete the testing of the video signal processing, we can use the known frame information to verify that the correct LEDs are addressed and the correct sections display the correct color. As well, this test will allow us to determine if the image is shifting across the display. If we find the image is shifting, we will need to make adjustments to the motor to increase or decrease the rotation speed of the POV display.

8.3 GUI Testing:

In order to test the GUI we will be using the requirements specified in the research, and refined in the design. These requirements will serve as the basis for our testing procedures. There are three basic main requirements that we will address: text message formatting, image message formatting, and communications. The formatting of the messages will focus on verifying that the user input is properly transformed into the proper pixel image data. Communications will be tested by connecting the computer to the FPGA using a USB interface and verifying that the data is sent and acknowledged correctly. We will look at each of these three 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.

8.3.1 Test Messaging Testing:

The text message format testing will include all of the features of the text message area of the GUI. This includes the number of text lines to be sent, the alignment of the individual text lines, and the color of the text to be displayed. All of the mentioned features must then be combined and converted to the proper format to be sent to the FPGA. We will begin with the easiest part of testing and work our way to the more complex tests. First we will test the number of text lines sent. Then we will test the color of text and the alignment. Lastly we will test the animation option. All tests can be performed by either reading the data in memory using a debugger, or having the program output to a text file.

The first test will verify that the correct number of text lines is sent to the FPGA. We will start by choosing one line of text. We can then verify that the text is converted to the proper format. We should only see one line of text sent and no others. Once we verify that one line works, we will then try two and three lines. Each time we will verify that the data shows the correct number of lines, and that they are displayed in the correct order. We will also want to make sure that all of the characters and symbols that we want to support appear correctly. After the line numbers have been tested we will move on to color and alignment.

When it comes to color and alignment there isn't one that is harder to test than the other so the order does not matter. We may also be able to test them at the same time. The data should show how the text is aligned. The easiest way to test the alignment is by typing in a single letter. This will make it very obvious where the text is located and how it is aligned. The color will also be easy to test because again we can use a single letter. We can then look at the information to verify that different colors produce the desired results. Ideally we would test all 256 color combinations. We will then combine this with the previous test and verify that different number of text lines can also be shown with the correct alignments and colors. After all the tests up to this point have been verified we can then move on to animation testing.

Animation is the last thing we will test for the text messages. The animation requires a combination of the software on the FPGA and the GUI. The GUI side of testing will actually be very easy. We will simply send an animation signal telling the FPGA which animation to use. The FPGA should have pre coded animations that it will use to display the text. We will verify that the animations work as intended and we will combine this final test with all the previous tests as well. The animations should work with all of the combinations of text line numbers, alignments, and colors. Once all previous tests have been verified then the text messaging will be considered fully implemented.

8.3.2 Image Messaging Testing:

Next we will test sending an image message to the FPGA. Testing the image message formatting is going to be more complicated than testing the text messages will be. The complexity is due to the fact that we have to analyze specific pixels to make sure that the data matches. This can be very tedious if we test with large images. I think most of the testing will be done with a simple low resolution image. When testing the image messaging we must consider the crop function, the position choices, and the clear image button. As in the text messaging testing we will be testing the easiest functions first before we move on to the more complex scenarios.

The clear image button will obviously be the easiest test to perform. We will simply press the button and verify that the information sent is the clear signal. On the FPGA side we will have to test and verify that this clear signal is received and properly clears the screen. Our next image test will be the position chooser. This will most likely be easier to test than the crop function. We will use a small simple image, possibly even just one color. We will see if the pixel data matches the location that we choose from the position radio buttons. After we verify that the basic position function is working we can then verify it with more complex images as well. This will also be a good time for us to test multiple image formats. Java's built in functions are supposed to handle all common image formats, and we will put that to the test. If we find certain formats cause trouble we may change the design to not allow the faulty formats.

Lastly we will test the crop function. We expect this to be the hardest part of the image messaging testing. It will be fairly easy to determine what the correct output should be when the crop option is checked. When the crop option is not checked it will be much more challenging to properly determine what the output should be. When the crop option is unchecked the image will be resized and the easiest way to see if this is done properly is going to be when the entire system is working and we will be able to see the image displayed. Before then we will have to estimate the correctness. When the crop option is checked we will be able to compare sections of the image with the data being sent. This should be similar to verifying our position tests, except now the image will take up the entire display.

8.4 Tachometer Testing:

This section will cover the hardware and software test required to verify the correct and desired operation of the motor control circuit.

8.4.1 Tachometer Hardware Test:

There are a few things that we must determine about the tachometer's hardware that we must determine through testing to insure that it will best track the revolutions per second of the LED apparatus so that we can effectively control the motor. The objective of the first test is to determine if the tachometer will correctly show a voltage pulse when it registers a movement change. The second test's objective is to determine if varying the CTRL signal to the sending circuit will have an impact on the efficiency of the sending and receiving process of the infrared LEDs. Finally the third test's objective is to determine whether the output of the receiver circuit has a noticeable enough pulse or change in voltage.

For all three of these tests we will need to use both circuits of the infrared sensor outlined in the design section for the tachometer, Section 5.3.1. The circuit will be left disconnected from the microprocessor for the purposes of this test.

8.4.1.1 Sending/Receiving Signal Hardware Test:

For the first test a voltmeter should be connected to the out location on the circuit design figure 4.3.1.1b also known as the receiver circuit. We will then connect a voltage source to the CTRL pin. This source will be used to turn on the sender circuit. The tachometer will be placed such that it will be directed toward a surface with some reflectivity, preferably the same type of surface that will be used on the LED apparatus in the final design. Then the CTRL pin will be increased slowly until a hit is registered on the receiving circuit. This will be noticeable by tracking whether the LED turns on or off and whether a voltage is registered on the voltmeter. Once a hit is received we will then remove the reflective surface and watch to see if the LED turns off or stays on and whether the voltage drops or raises on the voltmeter. If it stays on, we will slowly decrease the CTRL until it turns off.

The desired result of this test is to have the LED turn on when the CTRL is turned on and reaches a voltage above 2.5 volts, 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. Failure on both of these accounts could mean that the infrared circuit is too sensitive to ambient light, or the op-amp connections need to be reworked.

8.4.1.2 CTRL Signal Calibration:

For this test the set-up will be the same as the test in section 8.6.1. After the circuit is set up and the tachometer is directed away from the reflective surface, the voltage supply connected to the control pin will be slowly increased. Starting at 0V the voltage from the power supply should be increased by 0.5 volts up to 5V. During each increase in voltage the reflective surface should be passed in front of the infrared sensor slowly. The voltage change on the voltmeter should be recorded also during each increment.

The desired result of this test is 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. If this is not the case then we would like to determine with this test what CTRL signal creates the strongest and most noticeable receiver “hit” signal.

8.4.1.3 Tachometer Hardware Test Conclusion:

Both test results can be used to determine the third final objective and that is whether the signal is strong enough to determine when a hit is received or when ambient light is scrambling the signal some. The results of these tests as stated in each test will allow us to calibrate the CTRL signal and determine if an analog to digital conversion will be needed to determine when a true voltage pulse is received instead of ambient light interference.

8.4.2 Tachometer Software Test:

While a lot of debugging will obviously go into making of the software for the processor the tachometer debugging may take a little work in order to determine that it is effectively maintaining the revolutions per second of the motor. There will be two stages to this test. The first part is to determine whether the processor is effectively increasing or decreasing the voltage correctly with respect to an increase or decrease in the revolutions per second of the motor. The second part is to see if the motor's speed is maintained when connected to the processor. In both cases the full control circuit as outlined in the motor control section 4.3 will be needed.

In the first step of the test the microprocessor should remain disconnected from the pulse width modulation control circuit. The tachometer should be connected to the processor and set to record the revolutions per second of the motor shaft. A voltmeter should be connected to the out pin that will be connected to the voltage controlled resistors on the control circuit. A power supply should be connected to the voltage controlled resistors R8 and R9. In this portion of the test the motor's speed should be varied from high to low revolutions per second values and the voltage on the out pin of the microprocessor recorded.

The second step of this test is then to connect the microprocessor to the control circuit and to watch the revolutions per second of the motor and see if the motor settles at a speed and maintains that speed until stopped.

The desire of this test is to show that not only does the program react correctly to an increase or decrease in revolutions per second in the motor but will also maintain the speed of the motor.

If the program cannot maintain the speed then a new algorithm may be needed to effectively control the motor better, or an error in the programs logic may be causing some form of interference with the control circuit.

8.5 Pulse Width Modulation Circuit Test:

The pulse width modulation circuit is integral to sustain a motor speed in order to prevent image distortion. This means that it is very important that we test it thoroughly. There are also some calibrations that will need to be done with this circuit to determine what best operates with the motor. The first test's objective is to determine the best resistance to have in series with the JFET for the voltage controlled resistors. The second test will also be used to determine the actual relationship between the perceived resistance value and both the frequency and duty cycle of the circuit. The third test's objective is to determine the best frequency that this circuit will run on in order to best control the motor. The final test will be to determine if the pulse width modulation circuit can effectively control the motor's revolutions per second from 0 to 100% of its rated value. The circuit that will be used for this test is the circuit outlined in Figure 5.3.2.a. For the purposes of these tests the circuit will not be integrated yet with the microprocessor.

8.5.1 Voltage Controlled Resistance Calibration:

In this test we will need a potentiometer to take place of the R resistor in the design of the voltage controlled resistor figure 4.3.2b. A voltage source should be connected to the Vin while a voltmeter should be connected to the Vout. Starting with 1 kilo-ohm the voltage in the Vin should be increased incrementally to 12 volts. The Vout voltage should be recorded during each incremental step. Then using the voltage divider principles the effective resistance should be determined. This should be repeated for multiple resistances until a resistance on the potentiometer creates the perceived range of resistances along the JFET.

The desired outcome of this test is to either determine that this R resistance is irrelevant with respects to the perceived resistance ranges of the voltage controlled resistance or what is the best resistance to gain the largest range of resistance.

8.5.2 VCR Frequency/Duty Cycle Relationship:

For this test the pulse width modulation circuit from Figure 4.3.2a will be connected to an oscilloscope at the M2 MOSFET. A power supply will be connected to both the voltage controlled resistances. The first portion of this test is to incrementally increase the voltage of the R8 voltage controlled resistor and track the change in the duty cycle. The second stage of the test will be to then increase the voltage of the R9 voltage controlled resistor incrementally, during each increment varying the voltage of R9 up and down. The frequency should be tracked during each change in R8 and R9's voltage.

The desired result of this test is to determine how much voltage in R8 changes the duty cycle of the waveform, and how much voltage in R9 changes the frequency of the waveform. This can be used to accurately assume the relationship of the voltage controlled resistance and the change in these two variables of the circuit.

8.5.3 Frequency Calibrations:

This test will be completed with the Figure 4.3.2a with an oscilloscope connected to the M2 MOSFET. The motor will be connected to the circuit for this test. The voltage controlled resistors will be connected to a power supply instead of the microprocessor. The first step of this test is to set the voltage of the R8 voltage controlled resistor to a value that noticeably rotates the motor. Then to incrementally increase the R9 voltage controlled resistor's voltage and watch for any disturbance in the motor speed. The desire of this test is to determine whether the frequency even has a factor in the revolutions per second of our motor, and if they do what is the best frequency to control our motor with.

8.5.4 Motor Control Test:

This test will be conducted with the Figure 4.3.2a pulse width modulation circuit. The motor will be connected to the circuit for this test. The voltage controlled resistors will be connected to a power supply instead of the microprocessor. In this test the frequency of the pulse width modulation circuit will be maintained at a steady value as the R8 voltage controlled resistance is varied. Starting with a very small perceived resistance and increasing to the voltage controlled resistance's maximum value the revolutions per second of the motor will be observed.

The desire of this test is to have the motor start from a stationary position and raise all the way up to its maximum revolutions per second and then back down again all through just the variation of the R8 voltage controlled resistance.

8.5.5 PWM Test Conclusions:

The hopeful results of these test is to have the best frequency for the pulse width modulation circuit to run on and to have a mathematical model of the voltage controlled resistors and both the duty cycle and the revolutions per second of the motor. If one or all of these are not accomplished then an alternate motor control method may be needed. Such as a variable resistance method or a pulse width modulation circuit using a 555 timer circuit design that we have prepared as an alternate possibility.

8.6 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 with be unable to do anything we desire it to. This is why we have come up with a few tests to insure that the slip ring design we came up with will work under our desired conditions. The first test is nothing more than a durability test of the slip ring to determine if it can handle both the electrical and physical demands of the device. The second test is the power transfer test, it is to determine that under the most ideal of conditions that power is at least properly transferred through the slip ring. The final test is 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.

8.6.1 Slip Ring Durability Test:

This test will require a completed motor and motor control circuit so that the motor's speed can be varied. This means that most likely the Figure 4.3.2a pulse width modulation circuit and the motor will be used for this test in addition to the slip ring as shown in Figure 5.6.2.a. In essence, the slip ring, attached to the motor and LED apparatus with the LEDs not installed yet will be at first begin stationary. Then the motor's revolutions per second will slowly be increased and then maintained at its maximum rotations. After a number of minutes have passed the motor's revolutions per second will be slowly decreased and then the motor will be shut down. After the device has been powered down the slip ring will be checked for damage.

The desire of this test is to have the slip ring capable of handling the maximum possible revolutions per second that the motor can obtain and any variations in this rotational speed. While it would be ideal to not have to worry about any structural damage to the slip ring we want to at least see minimal damage.

8.6.2 Ideal Power Transfer Test:

This test will require only the power supply, the slip ring shown in Figure 4.6.2a, and a voltmeter. The slip ring will be connected to the power supply on the outside, and the voltmeter will be connected to the wire of the slip ring that is expected to be threaded through the shaft of the LED apparatus. The power supplies voltage will be varied from a low to high AC value while the amount on the voltmeter side will be recorded.

The desire of this test is to see that all or a majority of the power applied to the non-rotating side is seen on the side that will be rotating. Since this would be optimal conditions, neither rotating, for obvious reasons if the power is not seen on the other side of the slip ring or there is a large amount of loss then this design for the slip ring is no good.

8.6.3 Rotational Power Transfer and Thermal Dissipation Test:

For this test we will need 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 will be connected to the power supply. Since we don't have a way to directly measure the rotational side during its rotation we will use this prototype circuit to get an idea of how much power is being supplied to the rotational side. To begin we will start with a power input of around 300 watts and begin rotating the device with the prototype circuit connected to the rotational portion of the device.. We will then gradual increase the AC power supply until we get the light to just turn on and record this value. Since the bulbs require 360 watts to power whatever the difference in what is being inputted into the slip ring and the expected 360 watts will be a portion of our dissipation loss in the circuit during rotational conditions.

The desire of this test is to have the slip ring handle the physical and electrical demands of the device with minimal power loss.

8.6.4 Slip Ring Test Conclusions:

If the wire is unable to handle the physical demands of the device then we may have to add more wire taps into the ring or choose a more durable material for the wire. If the slip ring itself can't handle the demands of the device, which we believe is less likely, then we will have to come up with a more durable design or we may only need to choose a better material. In the case of high thermal loss we may need to then adjust our input power to the slip ring and alter our rectifier circuit on the opposite side so that it can handle these changes.

9 Conclusion:

The process of designing a persistence of vision device turned out to be a far more complicated endeavor then our team expect. 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. The design presented hopefully should accomplish our goals for this. 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 FPGA board was capable of handling, the problem is the LED controllers available to us were not. This meant having to come up with an ingenious method for refreshing our display that our controllers could handle. Which lead into an increase in our revolutions per second for the whole device, which in turn raised the requirements of our motor again. Power transmission as expected gave us many issue, since the process of transferring power over a rotating wire is difficult. Since we didn't want to just transfer power but also wanted to transfer data from our computer interface, we also needed a process that would not lose information or very little at least. In the end, this meant sending two different signals, our power signal through a slip ring and our data through a coaxial rotational joint. However, in order to do this we wanted a medium that was common for our FPGA so it would be easy to transfer and receive data. This turned out to be Ethernet since both FPGAs would have an Ethernet input. However, needed the Ethernet connection meant we needed to convert to coaxial with a converter in order to then use the rotational joint.

These design challenges discussed above or overcome but at a substantial increase in our first projected costs. This means that the need for sponsorship has tremendously increased. The entire design is under the expectation of an almost limitless budget, however the loss of sponsorship would require some rather extreme reductions in scale of the design. Specifically our team has discuss that the HDMI instantaneous streaming of the display device would most likely have to be cut. This is primarily for two reasons. The first reason is that the loss of a sponsor would most likely require us to drop the LED count and thus dropping 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 is the ability to purchase less powerful and thus less expensive FPGA boards for image processing. Without the demands of the high data transfer associated with the instantaneous streaming of the display device our display would most likely project much more simplistic animations and text, thus needed much less data transfer and processing. Hopefully the cutting of both the HDMI and the more expansive FPGA options we would also be able to cut our motor demands which would allow us to purchase a far less expensive motor then the one outlined in the design. This would be because we could probably redesign our controller and LED array for a less demanding refresh rate. However, with these cuts it would not spell the end for scalability for the device. Since even with a reduction in hardware features there would still be 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 our entire group is looking forward to the construction of it in Senior Design II. Within the first two weeks of class we are hoping to learn whether sponsorship will be a definite possibility or whether rescaling of the project should begin. While we are still planning to order many of the parts associated with the portions of the project that we do not expect to change with a scaling change of the project, such as some of the LEDs for the prototypes, the pulse width modulation circuit, and the tachometer components. We do need to know whether we are changing gears of this project so ideally we do not purchase any components that will not be used since this has turned into an expensive endeavor. This will all be based on whether we obtain sponsorship from either the navy as originally desired or whether the University of Central Florida takes on our sponsorship needs as we had discussed with our supervising professor for the Senior Design I. Hopefully we will get sponsored and we can begin the process of creating this device which our entire group is very excited about seeing realized.

9.1 Bill of Materials:

As seen in Table 9.1, is a list of major items required to build the POV display.

Bill of Materials				
Item Number	Part Number	Mfr.	Description	Qty
Microprocessor				
101	Spartan-6	Atlys	FPGA Development Board	2
102	VMODBB	Atlys	VHDC Breadboard I/O Extender	2
Motor and Chassis				
201	MUV-6301S	Prestolite	Motor, DC, Wound Field, 12 Volts, 1.6 HP, 2800 RPMs.	1
202	8090T13	McMaster-Carr	Bearing, Extended-Ring Type ER, rated for 3,145 dynamic load pounds and 5,000 RPMs	1
203	Custom Metal Work	KEMCO Industries	Custom Aluminum Metal to include top plate, base plate and support rods.	1
LED Array				
301	OVS-3309	Multicomp	LED, Type OVS, RGB, SMD	480
302	TLC5940	TI	LED Controller, 16-Channel	180
Ethernet Communications				
401	EOC-AN/IN	EnConn	Ethernet of Coax Converter, 100 Mbps	1
402	205-HS	Mercotac	Rotary Joint, 2 Conductor	1

Table 9.1 Bill of Materials

10 Appendix:

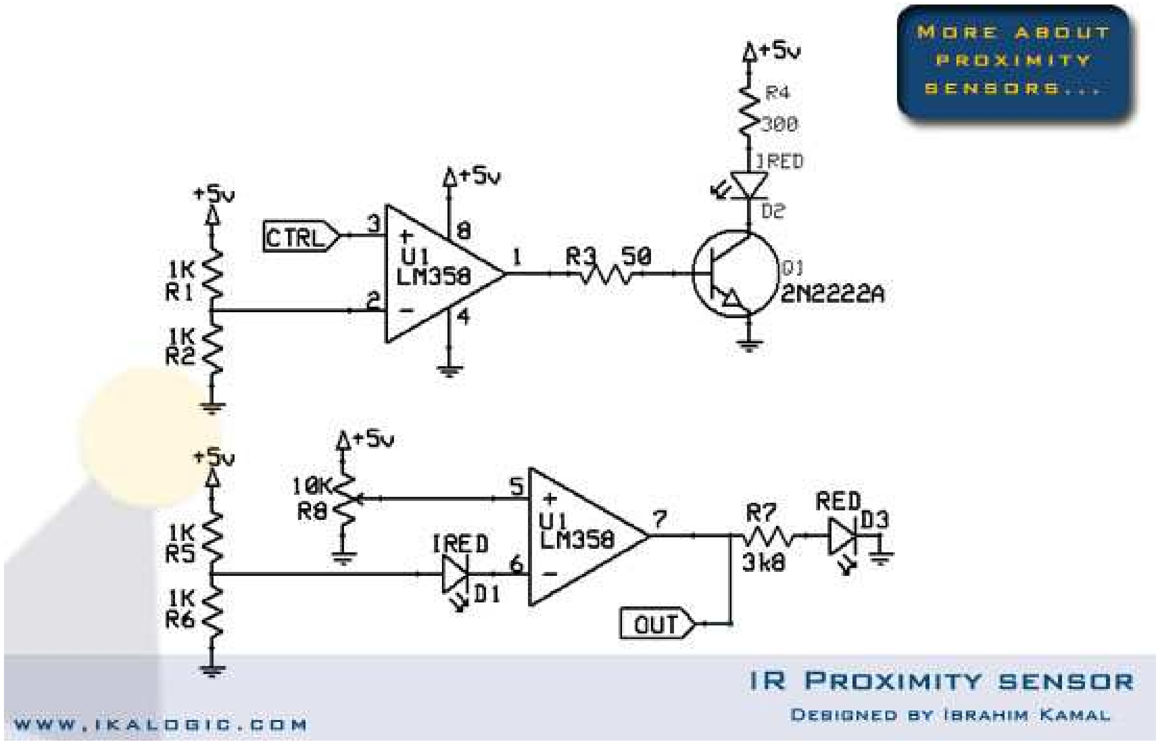


Figure 10.a Infrared Sensor Reference Circuit

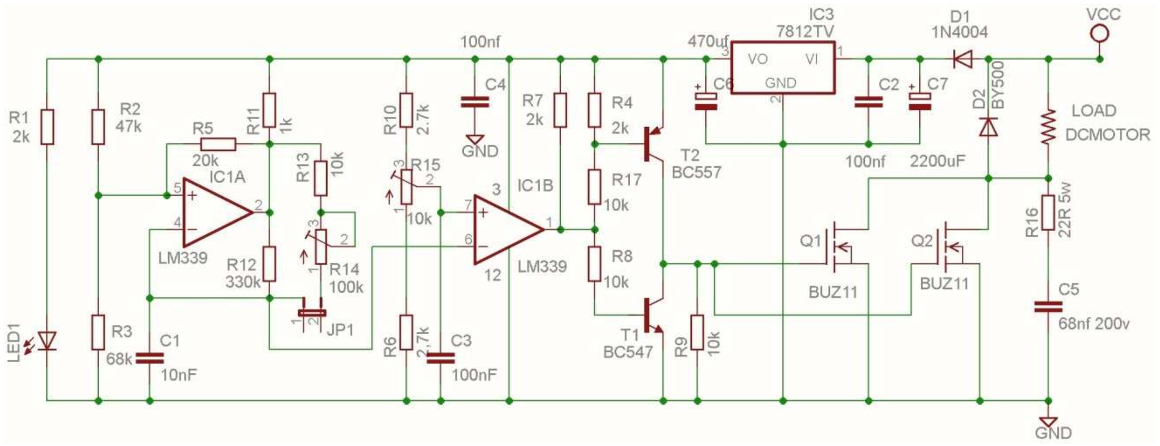


Figure 10.b Pulse Width Modulation Reference Circuit

11 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_Tim ing_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>.