Z-Goggles

Multifunction vision system for user sight enhancement

Senior Design Group 1

C.J. Silver
Will Petty
Geoffrey Jean
Cody Baxley

EEL 4915 – Fall 2010

List of Figures

List of Tables

Chapter 1 – Introduction

The following adventure is a Senior Design project brought to you by "Group 1" of the Summer-Fall 2010 EEL 4914-4915 Senior Design course at the University of Central Florida. The project demonstrates that the members of Group 1 have knowledge of computer and electrical engineering sciences necessary to analyze and design complex electrical and electronic devices, software, and systems containing hardware and software components, as required by the 2010-2011 ABET Engineering Accreditation Commission.

1.1 Executive Summary

Z-Goggles doesn't really stand for anything, it is just a name that makes you wonder what it could possibly be. The Z-Goggles system is a vision enhancement system, though not in the normal sense. It allows the user to modify what they do see in a number of ways. Mainly, this is done for the enjoyment of the user. Who doesn't want to see the world upside down? What does the world look like when it is a photo negative?

 Essentially, this is a video/image processing capability, allowing the user to modify the display with 3 functions. Z-Goggles can flip the display, create color effects, and simulate heat vision. Another twist is that the Z-Goggles are portable and capable of being worn outside. The Z-Goggles are designed to go where the user goes. It is battery-powered, and the display is integrated with a helmet for ease of use. The required power supply is located in a backpack that is worn with the helmet, so the complete system is wearable in two sections. Overall, the system isn't heavy, so the user should be capable of wearing it for extended periods. All this is done with a small budget, which entails cost-conscious part selection, and perhaps repurposing of parts for the system.

Thus, the general goals for the Z-Goggles project are cost-effective engineering, portability, effective video data gathering, processing and display techniques, and user friendly design. Each goal is accomplished by meeting a set of objectives, and all the objectives are met by meeting requirements and specifications for each sub-system. The sub-systems include the Optical System, Video Processor, VGA card, User Interface, Display, Power Supply, and Helmet/Backpack.

Regarding the technical implementation, the entire system consists of relatively few parts. A digital camera connects to an FPGA for processing and display control. A VGA output card interfaces with the FPGA processor to output to a helmet-mounted display unit. The user can select functions via a controller, which is built upon a microcontroller which interfaces with the processing FPGA. All power is provided by the backpack power supply. Due to the segmentation of the design, each sub-system has its own design and requirements. This paper is organized by the research, design and testing of these individual sections. This will lead to a complete system once each sub-system is integrated. The design group each has sections they control, so every sub-system gets personal attention. This foresight should create a well designed, fully-functional system.

1.2 Motivation

A unique concept, combined with technical skills and cost-effective component sourcing, will become a project that makes people take notice. That is the defining idea of the Z-Goggles system, and it defines our motivations, as well. Very few projects like this have been attempted before, professionally, educationally, or otherwise. While doing something unique is important, the design is also rooted in the technical skills our group has learned throughout their college careers, work experiences, and personal endeavors. This is important for a Senior Design project to be successful. The final aspect concerns elements of cost engineering; it is up to the design team to make a system perform well under a strict, small budget. There are many other reasons why we have chosen this project, but these are the core ideas.

Initially, our group was brought together by how interesting and novel the Z-Goggles concept sounded. We all wanted to enjoy our project, while learning more technical skills and gaining something to add to a resume. After some research, only one student vision system was found that is similar to our idea. However, it did not have multi-spectrum viewing, instead focusing solely on the FPGA/embedded implementation of various image processing algorithms and techniques [2]. Certainly, this is closely related, but we have more aspects to design, so our focus is only partly on that subject. Other than that, our system appears to be quite unique, and should provide discussion and ideas for future projects. Our first motivation is uniqueness.

As mentioned earlier, the Z-Goggles provide many aspects to design and integrate together. Each design team member has different interests, so having several areas to focus allows each person to get some of what they expected from a Senior Design project. For instance, the Optical system requires knowledge of spectrum filtering, camera operation, and interface with processing circuitry. Every group member has an important role. Also, the project requires integration of several sub-systems, adding crucial project planning experience. Our second motivation is professional and technical development.

Finally, we wanted to focus on making a technically proficient product that keeps costs down. It should come as no surprise in this current work climate that cost-effective engineering is at the forefront of all concerns. Large, successful corporations are all stressing cost competitiveness throughout the design and procurement process. As such, the Z-Goggles system attempts to do a lot with very little. Performance and quality are important, but in a time when that may not be affordable, many customers will look to other options. This project can be done with higher end components, which also makes the design much easier. However, a single camera would be our entire budget. Instead of throwing money at the problem, we will be replacing it with effort and skill. This is what will be expected from many employers for years to come, given shrinking budgets and less consumer spending. Our final motivation is cost-effective engineering.

1.3 Goals and Objectives

1.3.1 General Objectives

The goals of the system are cost-effective engineering, portability, user-friendliness, and effective video input, processing, and display techniques. Objectives are derived from these general ideas. The Z-Goggle system will be judged as a whole based upon this set of objectives. Described here are the components of the system, as well as expectations for the function of these components. They must be met for the system to be considered complete and fully functional. After the General Objectives section is a specific Software Objectives section, which details the image processing functions of the system. The Testing chapter provides information to determine whether these requirements, and the specifications that accompany them, are met in a satisfactory manner.

There will be at least three modification functions, which include: inverting the video output (upside down and left-right switch), color effects, and simulated heat vision. Also, other functions may be added when development is underway. This will allow creativity and exploration of system capabilities further in the design process. Video output will be at least 320 x 240, up to 640 x 480. Video input will be fast enough to allow the user to interact with the environment. A display screen will be provided that displays the input video. The user interface will allow the choice of both viewing mode and function. Only one mode need be selectable at any one time, but two viewing modes may be accessible if functions demand it. If a particular function does not work with a viewing mode or another function, the user should be informed of this on the display or via the interface control itself. Control mechanisms should be easy to use and not interfere with the system operation. After these are accomplished, the video input, processing, and display goal will be met.

To further create ease of use, the system shall be integrated with a helmet and backpack, so as to allow free movement. It shall be light enough so that a user can use it for at least 30 minutes without exhaustion. Due to this, the system should be capable of powered operation for at least 2 hours. The video display shall be placed so that eye strain isn't readily noticeable. The unit shall be distributed so that minimum strain is put on the user's neck. The system shall be able to withstand normal movement forces without failure, including turning and nodding of the head, and walking. Any systems unsafe to the user shall be covered or placed so as to not be accessed accidentally. Each component shall be well secured, and loose parts and wires kept to a minimum. All of these aspects should lead to a user-friendly experience and portability.

Finally, cost will be the determining factor in most design choices. Engineering in a cost-effective manner should be a major concern while fulfilling these objectives. Designers should use components that are feasible for required system performance, and all alternatives should be explored to find the proper part. The project requirements are low for a reason; systems are available that easily suit the requirements, but they are cost-

prohibitive and likely exceed the requirements by a great deal. Balance of performance to cost is crucial.

1.3.2 Software Objectives

Outlined in this section are the specific objectives for each vision modification function that will be developed for the Z-Goggles system. This may not contain every function that is eventually included in the image processing package. During further design and development, several other functions may be added. This depends on processing capability, time, and reusability of function blocks. There are three set functions that will be included at this time. They include reversal of video output, alteration of the color spectrum, and simulation of heat vision.

To implement reversal of the video output, the video must display the current scene upside down, as well as flipped right to left. The effect should be such that moving the head to the left will seemingly move the display to the right. When moving the head downward, the user should see the display move upwards, and so on.

Creating a photo negative effect will invert the intensity of each pixel in the image. In other words, black will become white, and vice versa. This effect is generally used on black and white pictures, but the effect should be capable of reversing color as well. There may be a modification of this function needed for it to work on the visible camera, which is also required.

To implement the simulated heat vision we will directly map the intensity values of the YUV input to the color values for RGB. This will make the color spectrum seem like everything is bright and intense with a clear contrast between objects. This function will be integrated with other functions to be able to act in parallel for the user.

It should be noted that most of these modification functions should be possible in any combination, at the user's discretion. There are some notable exceptions to this rule. It does not apply if it is determined that a certain combination of functions is found to be detrimental to using the device, or nullifies the purpose or intent of the viewing mode or functions. Also, this applies if the designers cannot combine certain functions due to performance issues. In this case, an option must be in place for the designer to halt certain function combinations from executing; the alerting system mentioned earlier will inform the user this combination of functions is not allowed.

1.4 Requirements and Specifications

The list of requirements shown in Tables 1-5 are composed of specification sections derived from the General Objectives, and combined in groups due to similarity. Each section has its own set of specifications, so that smaller subsections of the system can be evaluated objectively to determine root cause of any issues, which will lead to a better overall system.

Table 1: Optical and Display Systems Requirements

ID#	REQUIREMENT
S1.	The unit shall contain one camera to capture live video data.
S2.	The camera shall allow for color video output.
S3.	Cameras resolution will be at least 320 x 240, up to 640 x 480.
S4.	The Optical System will output RGB or YUV data in a usable or convertible size (8bit is the target for the Video Processor).
S5.	The unit's camera system shall be capable of capturing at least 15 fps (30 preferred).
S6.	The unit's display system shall be capable of displaying at least 15 fps (30 preferred).
S7.	The unit's display system shall be VGA compatible for easy interfacing.
S8.	The unit's display system shall be mounted approx 6" from the user's face, and enclosed in a manner that allows the user to focus on the screen.
S9.	The unit's display system will be capable of outputting a resolution of 640x480.
S10.	The unit's display system shall receive valid frame data via a VGA controller on a FPGA board.

Table 2: Video Processor Requirements

ID#	REQUIREMENTS
S11.	The Video Processor will be an FPGA.
S12.	Total process time will be designed for 30 fps of video output, to optimize camera performance to the highest level possible from each camera.

S13.	Any output mode must be capable of processing the input data within the time it takes to grab a new input pixel from the source.

Table 3: User Interface Requirements

ID#	REQUIREMENTS
S16.	An ATMega168 microcontroller will be uses to control the user interface.
S17.	The system's user interface shall be contained in a switch box remote
S18.	The system's user interface shall contain one switch for each individual function
S19.	The user interface software shall allow for multiple functions when feasible, and deal with multiple function switches when not.

Table 4: Power Supply Requirements

ID#	REQUIREMENTS
S20.	The unit shall have a portable power supply, such as a type of battery.
S21.	The unit shall provide multiple power outputs for the display, FPGA, and UI microcontroller.
S22.	The power source will be safely placed in the backpack, and any power lines that are run outside will be covered safely.
S23.	The power source will have storage capacity capable of running the system for 2 hours.

Table 5: Physical Distribution Requirements

ID#	REQUIREMENTS
S24.	The unit shall have a weight of no more than 50 lbs.
S25.	No more than 10 lbs. can be attached to the user's head; the remaining weight shall be distributed elsewhere.
S26.	Any electrical components attached to the helmet will be covered properly.
S27.	When project is completed, each component will be secured in some manner so that movement has no effect on operation.

Chapter 2 – Research

2.1 Optical System

2.1.1 Cameras

There were several options available to use as the Optical System cameras. Initially, due to the fact that we required three different types of video output, the idea was three separate camera systems. Other options needed to be explored, as three cameras could cause the price of the system to become unfeasible. The other option is a high quality single camera, which could be simpler to integrate and use, but disallows the use of multiple camera functions, such as image fusion. However, the choice of a camera system is not so simple. Availability of interface devices for the camera's data output is a major deciding factor, as well. All of these factors were taken into account during the decision making process.

Initially, the research focused on the base CMOS and CCD sensors that make up the 'heart' of every digital camera. These sensors are relatively cheap, and can be designed around any specification. Unfortunately, it soon became apparent that the design of a good camera system was a fully-fleshed Senior Design project in itself, and was out of the scope of our needs. This is due to the fact that an additional control system must be employed around the lens and sensor system to control the raw output of the sensor and correct dropped pixels and timing issues. The flexibility this affords would have been of great use, but it requires more work and testing time than we have available [3].

A second option was explored after our initial meeting with Dr. Richie. He gave us the idea of using the CMUCAM, which is a camera developed at Carnegie Mellon University for Machine Vision applications. At first glance, this camera was developed with our project in mind, and was within budget. It already has a suite of simple image processing algorithms programmed in, which would take some of the burden off our Video Processor and eliminate some design requirements. Unfortunately, careful inspection of the data sheet and the discussion boards found that the video output was B/W, and modifying the camera for color output would yield a very low frame rate. Our requirements for the project are to create a color camera system, at least for the visible light spectrum, and CMUCAM doesn't allow it [4]. In addition, the price only allowed for a single camera, the hazards of which are covered in more detail later.

Our third option was to purchase an already completed board-level camera of the type we initially explored. There is a niche industry that creates these cameras for other companies that produce cameras, or hobbyist camera makers, which includes us in this case. One roadblock with this type of product is that the majority of the companies do not produce single units, which eliminated multiple good options. A second problem presented itself once we requested pricing from the most promising board camera, the Dragonfly II from Point Grey. The camera and required development kit totaled $695 or

roughly 7/8ths of our total budget [5]. That fact dissuaded our use of the Dragonfly II. There were several other issues with a one camera system. Using one camera requires a switchable filtering system to even achieve the specification of three view modes. This is an additional cost; or, it would require additional design time to create the switching system. Ironically, most of these pre-made cameras were too high quality for our needs; because, the video output could not be higher resolution than 640x480, unless we spent far more than our total budget on the Video Processor components. The only positive for this option turned out to be simplicity of use, as only one camera need be controlled. This was not enough reason to utilize a single camera system for our project.

Ideas tend to come around full-circle after every other option is exhausted, and that was the case here. The final system our group decided upon was a three camera setup. Some of this decision was based upon the lack of other viable options, as detailed previously. The final support for this idea came from Dr. Richie, when he introduced us to the idea of using an interface device. These devices allow us to get varying video outputs from most types of commonly available digital cameras. In light of this information, using three simple webcams or other cameras paired with interface devices would enable us to create a low cost solution to the need for multiple spectrum viewing.

Finally, a camera was found that meets design requirements and cost requirements. The original plan was to use a Logitech C200 webcam. It captures 640 x 480 video at up to 30 fps. This meets the specifications if the interface device and Video Processor can get the best possible speed out of the cameras. We did not know how the camera is internally configured, so filtering posed a potential challenge. It supports USB high speed protocol, which complements a USB Controller host device. At $30 or three for $90, the price is very reasonable, and allows more money to be spent on other components if needed [6]. Unfortunately, when Logitech was contacted regarding their USB data format and advice regarding our needs for the device, we came across several issues. First, the camera required Windows native drivers to operate in any capacity. We had some idea this would be required, but Logitech support couldn't release any information regarding this process. Second, Logitech also could not provide us with any information regarding the data formats we could expect to be receiving once we got the USB connection running properly. Without this information, we could not determine what our data would actually look like until everything was acquired and in the prototyping stage. Not only is this a hindrance for properly designing the system, it would cause massive delays during the final semester. This, combined with the inordinate effort of reverse engineering Windows drivers to get the webcam to operate, caused us to rethink our Optical System choices and go back to the drawing board.

A new plan was formed that allows flexibility and foregoes the problems encountered with proprietary formatting and USB devices. SparkFun Electronics offers two camera solutions that are usable for the Z-Goggles. We chose two options, so that we have a fallback option if one fails to work as expected. First is the TCM8230MD, which is a tiny camera module that outputs 640 x 480 RGB or YUV data, at up to 30 fps. This module avoids the prior mentioned pitfalls of other board cameras, because it provides automatic white balance, shutter control, and gain control. It is extremely cheap, at roughly $10.

Another positive is that it would not require an interface device at all. It isn't all great, however, as it has several issues. It requires a control circuit to control power, having a relatively odd 2.8 V requirement. Also, it uses I2C to connect, which is a mystery to us currently. Its small size makes soldering very difficult. The size also causes a possible project failing issue, as the filter removal/change needed could be impossible for the IR and UV camera. The documentation isn't very helpful regarding these issues, as it is a tentative document from Toshiba. All that being said, it is a good option to explore during the design phase, and is our first choice [7].

Our second choice is related in a way to the first camera module, but is a completely designed camera system, the CM-26N/P. It outputs a NTSC analog video signal, which can utilize a video decoder to output the digital data we need (RGB or YUV). Given that the device is complete, it requires no additional control circuitry, other than the video decoder. We can immediately test it, as the camera can be plugged directly into a television or other NTSC display device. Perhaps the most important distinction in this camera choice is that the optics seems to be more easily accessible, giving our group a higher chance of removing/changing the filter. It does have some negatives, which is that the display is likely to be smaller than our maximum 640 x 480 display, which may require clipping and lower the resolution of the output video. For these reasons, it is our second choice for the Optical System camera, and will be the main design focus for the project [8].

The CM-26N NTSC camera module was found to use an "Interlaced" scanning format. This scanning format divides the screen into even and odd lines and then refreshes them alternately at 30 frames per second. This creates a slight delay because while one half of the lines keep up with the moving image with the other half waits to be refreshed. An interlaced scanning system is used commonly among analog cameras, and CRT televisions. Newer technologies like digital cameras and LCD monitors on the other hand use a "progressive" scanning format. The NTSC interlaced scanning format for a 640x480 resolution at 60Hz scans in 525 lines, at 60 fields or 30 frames per second. Each frame is scanned in two fields of 262 lines and is then combined to display a frame of video with 252 lines [9]. In order to use this camera data we must use a "Video Scan Doubler" to covert interlaced data into non-interlaced data. We first would have to use a video decoder to take the composite analog interlaced video signal and convert it to digital interlaced data. Then the "Video Scan Doubler" can take the data and convert it to a non-interlaced digital format.

2.1.2 Filters

Light incident on the cameras contains photons of many different wavelengths, including visible light, ultraviolet, and infrared spectral content. Naturally, humans cannot see in the UV and IR spectrums; however, the CCD or CMOS arrays in digital cameras may pick up content from these wavelengths. For the Optical System, one camera will utilize a filter that isolates the IR content, and another uses a filter that isolates the UV content. There is no need to change the visible spectrum webcam, as it is optimized for viewing in that spectrum already.

It is important when designing a camera system that the proper filtering is used. The sensor array used while designing a camera determines the total spectrum that will cause a reaction, i.e. create image pixels. However, applying filtering allows one to remove any unneeded or unwanted wavelength spectrums. As such, to create the IR or UV camera, a material is needed that will isolate the IR or UV spectrum and remove the visible spectrum. There are many types of material that will work for IR, including some cheap options. The black portion of a photo-negative is one possibility, as well as the media inside an old computer floppy disk. These are the materials that we propose to use in the design of the IR camera [10][11][12].

Unfortunately, the UV transmission filter is more difficult to obtain. Ideally, we need a material that eliminates everything but some portion of the UV spectrum. According to Figure 1, that would require us to filter as much as possible above 400nm. Preferably, we want nothing but this area of the spectrum to come through to the sensor. The issue here is that materials that do this tend to be quite expensive. Professional filters cost over 200 dollars on average, with more precision in the passband leading to even more expensive filters. The cheapest commercial option that is still manufactured is the UG 1 from Schott, at roughly $63. There is also a filter from B+W denoted by the number '403' for $88.50. Figure 2 and Figure 3 are graphs of the transmission spectra for the UG 1 and 403 filters, respectively.
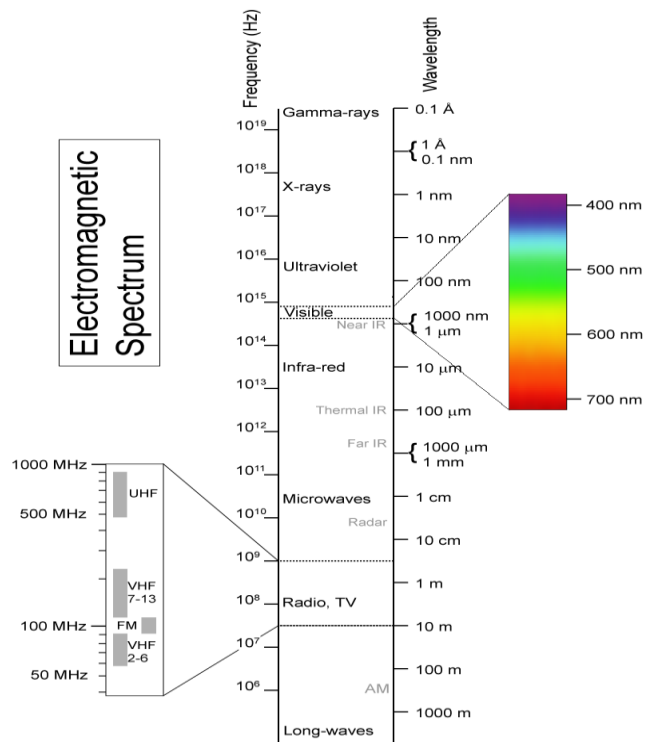
Figure 1: Electromagnetic Spectrum
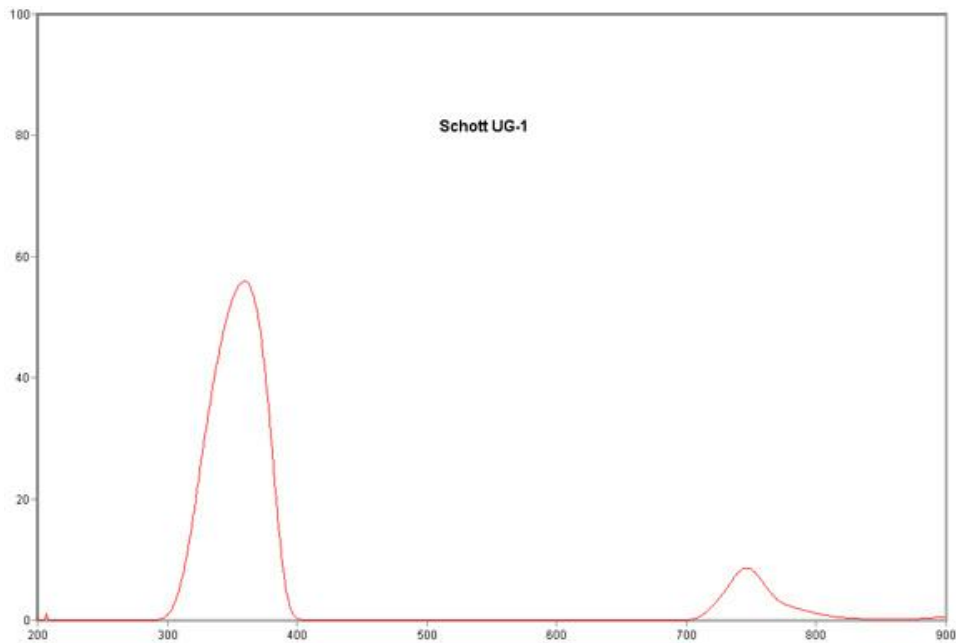
Figure 2: Schott UG-1 spectrum

(Printed with permission, copyright: Shane Elen. http://www.beyondvisible.com/BV3-filter.html)
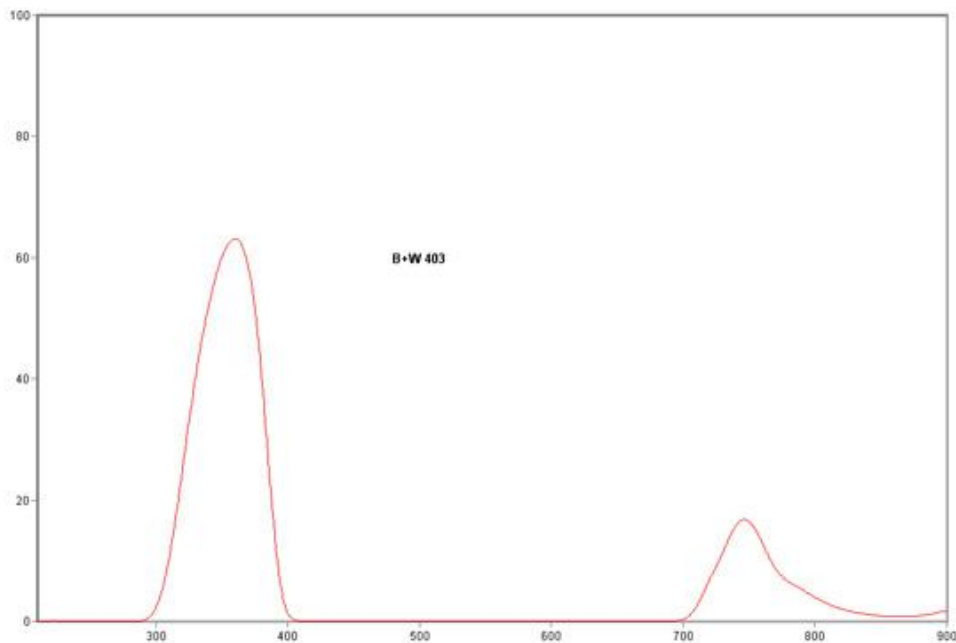
Figure 3: B+W 403 spectrum

(Printed with permission, copyright: Shane Elen. http://www.beyondvisible.com/BV3-filter.html)

From the graphs above, another issue arises that is cause for concern. The IR spectrum shows through in each of these examples! These solutions must be paired with an IR 'cut' filter to remove the IR content; thus, leaving behind only UV light. This additional cost may lead to a filter system that is twice the cost of the camera, or more. One option to reduce this cost is to use the type of glass that is used to make black-light bulbs, or "Wood's glass." This glass can be ordered for 15 dollars, and is a similar material to the above mentioned UV filters. Alternatively, a cheap black-light bulb could be broken and fashioned into a makeshift filter [13].

IR cutoff filters tend to be already present in most digital cameras, even webcams. This is due to the fact that IR light causes color change issues in images that contain both IR and Visible light. Depending on the filter included in the webcams we use, the filter could work perfectly with the UV transmission filter and create an image containing mostly UV light. However, the IR cut filter could also filter in the UV spectrum. This will have to be tested during the design phase. If the existing filter works, the solution is any of the above UV transmission filters alone. If the existing filter is inadequate, another cutoff filter will be paired with the UV transmission filter. Purchasing an IR cutoff filter isn't a terrible choice, as they can be found at Edmund Optics for roughly 40-70 dollars [13]. This option will be explored only if necessary.

2.1.3 Interface Devices

Several types of output are possible from a camera system or board-level camera. Board cameras would output raw sensor data, which requires preprocessing to output what we would normally expect from regular images, so we avoided this type of camera. It is possible, though it seems to be rare, for a camera to output usable digital data. Generally though, camera systems tend to output in NTSC or PAL color formats (composite video), or through USB connections. Each type of interface requires a device to convert the data to usable formats for image processing in our Video Processor. These usable formats are covered in section 2.1.3 Digital Image Formats. Producing data in a usable format is the main goal of using an interface device for the Optical System.

There are a few ways for video data to be encoded, and multiple devices to do data conversion. If a camera were chosen that used NTSC, we would have to get a video decoder that would take input of this type and transform it into basic image data formats. There is a set of different video decoders available from Analog Devices for just such a purpose, which allow decoding into multiple formats. The first video decoder which is viable is the AVD7180, which is capable of decoding analog Composite, S-Video, and YPbPr input signals in NTSC, PAL, or SECAM color formats. It outputs the sampled information in 8 bit or 16 bit YCbCr (4/2/2 format). It is programmable via a 2 wire serial bidirectional port, which is I2C compatible. It requires a flexible 1.8V to 3.3V power supply, which can be supplied by most FPGAs [14]. Another model, the ADV7401, is capable of converting the same types of input and more, as well as output in 12 bit RGB format. This is a valuable ability, as it would allow us to utilize minimal conversion math to have the 8 bit RGB we want [15]. A third option is the AL242 made by Averlogic.

Much like the other two, this video decoder will take in a number of formats including S-video and Composite, and will output multiple formats such as 8 bit, 16 bit and 22 bit YCbCr. The AL242 can also run off a 1.8V or 3.3V option, and is priced at $21.50 [16]. The main reason the Averlogic part would be chosen is if the analog camera chosen for the final design uses an "Interlaced" scanning system. In comparing the three, the first chip is the cheapest at roughly $10, while the second and third are roughly $22. The second chip would only be used to allow ease of use and cut down on processing requirements. They are all good choices if a video decoder is required.

If an analog camera is chosen that uses a NTSC Interlaced scanning system, a "Video Scan Doubler" will be needed along with a video decoder. As discussed in the cameras section of this research, this device is used to convert interlaced digital data to a non-interlaced format. The Averlogic AL251 is an ideal part for this job and will work perfectly with the AL242 video decoder mentioned above. The AL251 Video Scan Doubler is capable of taking in the digital 4:2:2 YCbCr formats that the video decoder outputs and output in 5:6:5 RGB digital formats. The AL251 is programmable via an I^2C interface and is capable of doing some simple image processing techniques such as overlaying an image on the video output. The AL251 is priced at $10 and will be used only if an analog camera is chosen and the camera uses an interlaced scanning system [17].

These devices aren't needed, however if the cameras chosen use the USB protocol. USB connections are generally used to interface with computers, but in our case, they need to interact with the FPGA-based Video Processing system. To interface with any cameras via USB, we need a link layer and controller. Implementing a full controller on an FPGA would be very difficult, so we require some form of USB host interface card. Relatively early in our research, a USB host Daughter Card that should work with our Video Processor was located. The advantage of this card is that we should be able to isolate the image data being output by the webcam. The Video Processor will take over from there. Digilent Inc. offers the SMSC EVB-USB3300-XLX USB Daughter Card, which provides a digital layer and buffer for our design. Instead of having to create an actual serial decoder and full USB host, we only need to create a digital interface. It supports up to USB 2.0 in all speed configurations and can function as a host or device, allowing us to test and debug all of our USB data transmission. Each camera can be fitted with a separate host controller and all of them can be sampled in parallel, guaranteeing that we get the rate of data throughput that we need [18].

While the USB Controller could output the data we need, the type of data that we will be acquiring may be dependent on the camera itself. Most digital cameras have their own raw image formats [19]. A USB webcam may be providing this proprietary image format, general format data like RGB, or perhaps even raw sensor data. Obviously, we would prefer the general format data. We will not know what the exact status of the image data from a USB camera until it is connected and providing USB packet information. The biggest issue with raw sensor data is that it isn't white balanced and color corrected. We attempted to avoid this extra processing step by not designing our own board camera, but it may be unavoidable, given raw sensor output from the USB camera.

2.1.4 Digital Image Formats

The purpose of this section is to explore the common types of color formats the system may need to utilize during image processing. While there are many different kinds of color models, most were designed for certain applications. For instance, the CMYK color model is used for color printing [20]. Obviously, this isn't a format that the Z-Goggles system will be encountering. In our judgment, our system will likely need to use either the RGB or YUV color models. Both of these color models consist of different bit number versions, however, and may require conversion or indexing to work properly with our Video Processor.

RGB color is composed of Red, Green, and Blue components for each pixel. Each of these can be considered separate pictures, which makes it simpler to use in building our image processing functions. Most RGB used today is made up of 16 or 24 bits of information per pixel. Memory size for our processing is going to be 8-bit wide, so higher quality RGB format isn't going to work. The VGA input and display that we are using will need lower quality a RGB format. SVGA format yields 256 colors using 8 bits, so this is ideal for the Video Processor requirements. Also, there is a 16 color option that only takes 4 bits, but we would rather design toward maximum usage of our resources [20]. Another question must be answered before moving on: If the camera chosen only gives 32 or 16 bit RGB data, how can a designer convert to a lower bit format? Logically, one can just pick a range of colors and represent all of those colors by one color designation in the lower-bit model. This is the idea behind the conversion, which just uses mathematical calculations to compress the color range. The formula for conversion from a 16 bit to an 8 bit number is shown below [21]:

$$256 \text{ range color} = \frac{127.5(x+1) + 16384}{32768}$$

Where, x is the input color value from $0 - 65535$. This equation should yield a value between 0 and 255. The problem here is that this gives us widely rounded numbers for the higher bit colors, and only applies to one color. Instead we will need to convert each color individually. Taking in a 16 bit RGB number as a 5-6-5 number we will need to covert 5 red bits to 3 red bits, 6 green bits to 3 green bits, and 5 blue bits to 2 blue bits. Starting with red, 5 bits gives us 2^5 or 32 shades of red. With 3 bits however we have 2^3 or 8 shades of red. To covert these 5 bits to 3, we will simply divide 32 by 8 and get 4. Every 4 shades of red will be converted to one. This same logic will be applied with green having 2^6 or 64 shades divided by 8 (8 colors per 1 color) and with blue having 2^5 or 32 shades divided by 8 (4 colors per color). Table 6 shows the conversion of each color, and which ranges of higher resolution color correspond to the lower resolution colors. The conversion from 32 bit RGB will be similar, as are any other possible bit conversions. Converting from 16 bit to 8 bit is the only RGB conversion that our system will likely need, given the quality of the data output seen during camera research.

Table 6: Conversion table for 16 bit RGB to 8 bit RGB

Red 5 bit	Red 3 bit	Green 6 bit	Green 3 bit	Blue 5 bit	Blue 2 bit
00000-00011	000	000000-000111	000	00000-00111	00
00100-00111	001	001000-001111	001	01000-01110	01
01000-01011	010	010000-010111	010	10000-10111	10
01100-01111	011	011000-011111	011	11000-1111	11
10000-10011	100	100000-100111	100		
10100-10111	101	101000-101111	101		
11000-11011	110	110000-110111	110		
11100-11111	111	111000-111111	111		

YUV color models are based on luminance and chrominance instead of the three-color combination used in RGB. Y stands for the luminance, or the brightness of the signal pixel, while U and V are the two chrominance (color) numbers. Originally, this format was used for television systems, so NTSC and PAL signals are based on this color model. Due to this, there are different types for analog and digital signals, named YPbPr and YCbCr, respectively. YCbCr is the most likely format for digital camera output. Since we would rather have RGB, a conversion would be used to go from YCbCr to RGB. This conversion is based on a set of equations and constants, where the constants are set by different standard documentation. Shown below are the equations to convert from YCbCr to RBG [22].

$$R = Y + Cr$$
$$G = Y - \left(\frac{Kby}{Kgy}\right) * Cb - \left(\frac{Kry}{Kgy}\right) * Cr$$
$$B = Y + Cb$$

Where,

$$Kgy = 1$$

The constants denoted by Kby and Kry are shown below in Table 7, along with the reference documentation by which they are set forth [22]. They are all similar, so the choice of which one to use will simply be a matter of preference, if the conversion formulas are necessary to the system.

Table 7: Constants by standard for YCbCr to RGB conversion [22]

Standard	Kby value	Kry value
ITU601 / ITU-T 709 1250/50/2:1	0.299	0.114
ITU709 / ITU-T 709 1250/60/2:1	0.2126	0.0722
SMPTE 240M (1999)	0.212	0.087

Finally, both of these color models may need to be converted to gray-scale on the UV and IR filtered cameras. The information coming through will probably already be grey-scale once filters are applied, but to be safe, conversion factors and methods were found for RGB and YUV. Converting RGB to gray-scale is very easy, as it takes a few

multiplication factors and adding the Red, Green, and Blue components [23]. This results in a single intensity value for the pixel.

$$Intensity = 0.2989 * Red + 0.5870 * Green + 0.1140 * Blue$$

YUV is technically easier, as the Y value is the luminance, which directly corresponds to the brightness or intensity. In other words, Y is the gray-scale image [22]. Unfortunately, in a typical 8 bit 4/2/2 YUV setup, this leaves very little information to determine different scales of gray. The Y component is the 4 bit component, which yields only 16 levels of gray. As such, it would be more advantageous to convert YUV to RGB with the conversion mentioned above, and then convert the RGB data to gray-scale. This would yield more than 16 levels of gray.

2.1.5 White Balance

Balancing the colors and keeping the images looking realistic is the point of white balance. Depending on what type of camera is chosen, there may or may not be white balance automatically performed prior to data output. Even if it is performed, the balance that the camera chooses to apply may lack greatly in comparison to the scene the video is attempting to represent. For these reasons, white balance will be explored for possible use in initial processing of the video data output from the Optical System.

The name white balance stems from our perception of visual correctness relying on the whites having balanced color content, since white is made up of all colors. White balance depends on the relative color temperature of the light, which changes greatly depending on the light source(s) that are present in a scene. For example, the sun has a color temperature of 5000-6500 K in clear daylight. The reason this is important is due to the fact that 'white light' contains different amounts of each color depending on the blackbody temperature of the source. Figure 4 denotes the differing color spectra in certain color temperatures of 3000, 5000, and 9000 K [24].
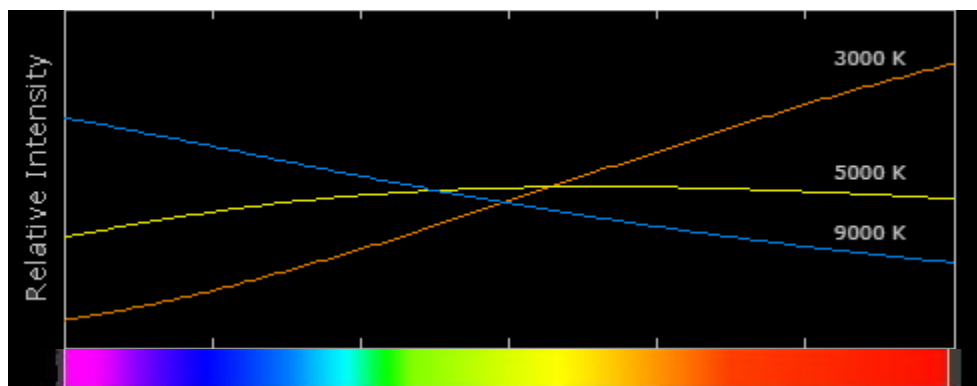
Figure 4: Intensity of color spectra for different color temperatures

(Permission pending, copyright: Sean McHugh cambridgeincolour.com)

Why is this important to the Z-Goggles system? The system will operate in sunlight, as well as indoors. These are both different light sources, and thus, could cause white balance issues with the output of the visible camera. The IR and UV cameras will likely be changed as well, but it is nearly impossible to judge what is 'right' or 'wrong' with these spectrums, because we do not see them for ourselves. From the graph in Figure 4, it is obvious that more blue or orange light content can be present, depending on the source. This would warp our visible camera output in ways we don't want. We want to modify the original scene with other content and functions, and additional color warping would detract from these effects. As an example, Figure 5 shows an image with too much blue content in comparison to the white balanced image next to it. The picture on the right is our goal [24].

Figure 5: Incorrectly balanced photo, Right: Correctly White Balanced photo

(Permission pending, copyright: Sean McHugh cambridgeincolour.com)

To correct the white balance, it must be done in the Video Processor before the data is run through the area where the functions are applied. This preprocessing is done so the input data is correct. The simplest way to adjust the white balance is to examine the color channels of an RGB image separately. Whichever channel has the highest mean is left alone, while the remaining two channels are multiplied by whatever scale factor is necessary to bring their means equal to the target mean. The scale factors should be simple ratios to equate the means together. In the following example, MeanRed is the red mean, which is highest. MeanBlue is the blue mean, and MeanGreen is the green mean. Red, Blue, and Green are the pixel values of each channel [25].

$$Red = Red$$
$$Green = \left(\frac{MeanRed}{MeanGreen}\right) * Green$$
$$Blue = \left(\frac{MeanRed}{MeanBlue}\right) * Blue$$

When this process is complete, the image channels will be more closely balanced, and the output image should be a much better representation of the scene. This simple balancing technique is automatically done by many cameras, but if the output is still not an accurate representation, we must do our own balancing by modifying the image manually with the FPGA. This is not a perfect solution, but it should allow us to get a good general setup for the types of scenes regularly being displayed.

2.1.6 Video Sync

Sync is an important issue with using multiple input cameras. The scenery incident on one camera is not necessarily the same on the other two, due to being placed in different locations. In a normal audio-visual application, there could also be an issue with Audio-Visual sync, i.e. the action shown not aligning with the sound signal. In our case, this is not an issue because the Optical System does not require sound input/output. The user will be in the environment shown at all times, and the sounds naturally sync together if the video output is quick enough. That is a design goal for video quality, so the audio sync should be inherent in that requirement. Spatial sync is the only issue that we must focus on.

Obviously, we must handle multiple input cameras in such a way as to minimize the distraction to the Z-Goggles user. The most efficient way to do this is to keep the three cameras as close as possible to one another during use. With this in mind, the each camera must be adjusted as they are mounted, and should be collocated on whatever frame they are mounted upon. If this is done, the video sync should be negligible, unless the user is looking at something very close. This requirement is a must to keep costs down, as the only other solution is manual or automatic physical adjustment of the cameras on the frame, which is neither feasible nor user-friendly.

There are other ways to deal with the remaining visual differences between outputs of each camera. The first method we have devised is to show a black screen between transitions between video modes (visible, IR, or UV). This should minimize the ability of the user to distinguish the small sync differences between the cameras. If the transition were instantaneous, the user would almost certainly detect a shift on-screen. Given a break in stimulus, along with the differences inherent in each viewing mode, the user will likely not observe the shift as easily.

Secondly, there is an issue with the fusion of two video signals, given this setup. Only a portion of the image from each camera is also present on the image from either of the other cameras. When fusing two images, the only part of the image that can realistically be fused is that which is in both cameras, which is diagramed in Figure 6. As such, during the fusion process, we will display only the overlapping image, which will eliminate the issue of visual sync while using this function. It is easy enough to leave off certain areas each image during the processing stage.
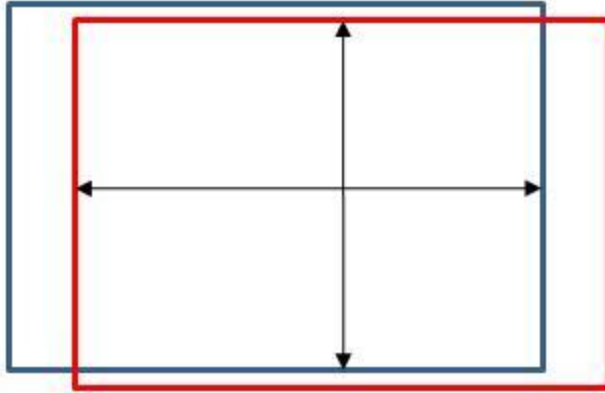
Figure 6: Video Sync Window

2.1.7 Image Fusion

One of the advantages of having multiple cameras is the capability to either combine image content, or adjust the image data of one camera based on the data from another camera. This process is called Image Fusion, and it is used for many applications, though mostly for military use. For example, thermally sensitive Long-Wave Infrared can be combined with Very-Near Infrared information to create a sensor system that works in low light, and also detects heat sources, such as hidden combatants. For the Z-Goggles system, fusion may be possible in a simple form, and a modification on traditional fusion will be explored for the spectrum shift image modification function. This capability introduces extra processing requirements to modify two full sets of data, instead of one. As a result, this type of functionality isn't a required capability, but it is still of use to understand the process. It will be enabled if possible for added functionality to the system.

The basic process of Image Fusion entails taking certain amounts of pixel data from each image and combining them together to create a new fused image. In our system, the output data from each camera flows to the Video Processor, which selects the input(s), dependent on the user's choices. Enabling fusion would be quite easy, as the user need only select two viewing modes, such as Visual + UV, or IR + UV. Corresponding pixels from each image will be weighted and added together. The weighting depends on how much of each image is desired to be in the fused result. Figure 7 demonstrates the general algorithm for combining two images of RGB format. The weighting factor used for both images having equal strength is 0.5, or half the pixel values from each image are added together. This simple method is much preferable to other methods that do Image Fusion, at least with regard to the Z-Goggles' capability. In other methods, the entire image is reduced in an iterative process, combined, and then the entire process is reversed [26]. Obviously, this would be a heavy processing burden; the FPGA that is going to be the core of the Video Processor cannot handle so many processing steps and still meet the stated objectives. As such, the weighting method is the go-to algorithm for Image Fusion, if implemented.
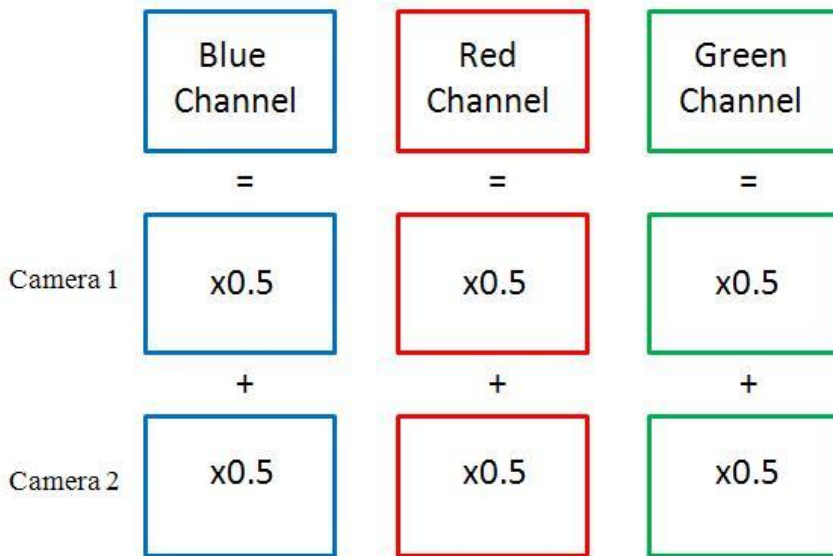
Figure 7: Image Fusion weighting algorithm example

2.2 Video Processor

Several considerations must be made before designing the Video Processor system. The aim for Z-Goggles is to achieve at least TV-quality throughput, which means pushing thirty frames per second at a resolution of 320x240@30fps with an 8-bit color depth. At maximum, we would like to push VGA-quality video (640x480@60fps 8bpp). The pixel data should be taken in parallel, so it will not factor in to sampling time. At QVGA (TV resolution), sampling time between pixel samples is approximately 434ns, which is requires a sampling frequency of 2.3MHz. At VGA the sampling time is 54ns, requiring a sampling rate of around 18.4MHz. Since we want to output in real-time, the selected technology must be able to get a pixel from input, change it, and get it back to the output before the next pixel is sampled. Failing to do so would require cascading pipelines and a dramatic increase in complexity.

To do any kind of convolutions, at least 3 lines of data need to be stored at a time, and a single output frame must be held for the video display. The output frame must be 640x480 8bpp, which is 307.2KB or storage space, and the video storage buffer must hold 640x3 24bpp, using an additional 5.76KB, making the total memory area required ~313KB. Originally we wanted to be able to do object tracking, but that would require at least 3 frames of data storage and our pipeline would have to access those frames and support frame copying or rolling as well as convolution. The overhead to do this is large and the concept greatly complicates the rest of the project. To do any kind of object detection using 3 frames of data, about 1MB of memory is required. Using convolutions to extract a 3x3 grid of pixels requires 9 memory reads and 1 write plus latency to actually do the convolution, and this must happen in under 54ns for VGA quality. Each read/write must occur in under roughly 4-5ns for VGA and roughly 40ns for QVGA. The

primary work for the convolutions depends on fast linear multipliers, as well as bitwise splitting, so any processor architecture must support both. We'll be working with simple intensity values for digital video so floating point support is unnecessary.

2.2.1 DSP

We elected to research DSP processors because they are used in a lot of video processing and accelerated rendering scenarios. They have fast linear multiplies and are ideally suited to video processing.

We were attracted to Texas Instrument DSP chips initially because of their toolchain support. We hoped to reduce our development time by implementing our algorithms in MATLAB, compiling them into C and using TI's C to DSP library to directly program the chip. It would allow for flexibility during the design and implementation of our algorithms. We reviewed the C5000 series processors. Only the fastest of the C5000 processor line had high enough clock rates to do any of the sampling work. Even with the available memory interfaces, running a DSP program fast enough to do the work required would not be possible [27].

We investigated the faster C6000 series of processors, but most of them were designed to be used in floating point or fixed point applications. The TMS320C6452 packages supported what we needed in terms of data processing, but the only available development board costs $1300 which far exceeds our budget [28]. The chips themselves are cheap, but the support hardware investment is not, and creating our own board for DSP development is a senior design project in itself. Any productivity gains in the development phase will be far outweighed by the task of constructing the required support hardware.

Having abandoned TI's line of generic DSP chips, we looked into their DaVinci video processing DSPs. Their development boards are more moderately priced and offer several different types of inputs. Unfortunately, selecting any one of them would restrict our camera input options. The TVP5158 Evaluation Board supports four CVBS inputs, but costs $500 [29]. Our other options for any analog video in VGA format only supported a single input at a time and cost around $200. These chips allow us to sample our video input at a high enough rate, but only for a single VGA camera or 3 QVGA cameras. None of them can guarantee sufficiently fast program execution for our project.

The Blackfin series of processors from Analog Devices support 16-bit or 32-bit integer math. Their internal clock speeds range from 300-600MHz. We can program them using VisualDSP++, but our team does not have experience with it. All of the development boards range from $100-$1000, but none of them support direct VGA output or enough fast on-board RAM to make them viable candidates [30].

After researching various DSP chips, we elected not to use them. The primary reason was cost of development and testing systems. The chips also offered clock rates much too low

to sample our data, no real way to deal with multiple streams of input, and not enough on-board memory to store any of the requisite data. These things, combined with our team's lack of familiarity with DSP programming and lack of time to build a custom DSP programming board, caused us to reject DSP as a viable processing candidate.

2.2.2 FPGA

FPGAs seemed to be the only other viable candidate in terms of processing speed. If we could design a hardware convolution path and guarantee though-put, the rest of our project would just be reduced to testing different coefficients and blending pathways. It would also reduce our cost dramatically, allow incorporation of the rest of the user interface, and eliminate the need for additional PCB design, testing, and debugging. Our team has experience working with Xilinx FPGA products, and the development board costs are much lower than any of the available DSP boards, in the range of $100-$200.

We considered the Nexys2 board made by Digilent, Inc. because we already had one immediately available. It has 1.2 million gates available as well as a large 16mb RAM module for data and program storage. The internal 50 MHz clock is good enough for QVGA display and can be scaled up to 100MHz. It has a built-in VGA port for video output and serial port or PMOD connectors for the user interface. The Hirose connector would give us plenty of high-speed GPIO lines for data coming from the cameras and going to the display. It also supports a battery connection which is ideal for our application [31].

The internal RAM module cannot scale far in terms of speed, requiring 20-40ns to be accessed, making this board only suitable for QVGA quality at best. We would also have to implement a very large multiplier or series of multipliers to do the kind of math we need, which would take up a very large number of gates. Even using booth's algorithm and carry-look-ahead adders, there is no guarantee that we could meet our specification criteria. However, if we use only 3 lines of internal buffer, it will fit in block RAM and we can meet our performance criteria using a fifo-oriented system that works one pixel at a time.

The S3BOARD made by Digilent, Inc. was a promising candidate for our FPGA. It supports internal speeds up to 500MHz, giving us a 2ns latch time for intermediate data storage and enabling us to sample and process fast enough to support VGA video. It also comes with 1MB of asynchronous static RAM with a 10ns access time, which is fast enough for us to use as a video output buffer. It has 12 18-bit multipliers, giving us the ability to do all of the convolutions and transforms in near-parallel. The expansion connectors are tied to internal lines to allow for debugging of the video data and transforms. It has a built-in VGA port for video output and serial port or GPIO lines for user input. This board meets our performance criteria, but only has enough IO pins to support 8-bit video, and the VGA port only supports 3-bit video. It could be used in conjunction with the Nexy2 board to support memory-mapped video but only just barely, and at extreme development cost. For this reason, the nexys2 board will be used for the project [32].

2.2.3 Data Storage/Retrieval

Application of both storing and retrieving image data is an important factor of the Z-goggle design. The design team must take into consideration methods for storing data in a fast and efficient way, while researching how to build a controller to communicate to the FPGA board's SRAM.

2.2.3.1 Data Storage Methods

To store our video data for manipulation, there are several options. The simplest option is to just place everything into on-board SRAM and do normal reads and writes. The SRAM is single channel, so it would require a memory controller to prevent conflicts between the video output and input modules. Assuming the raw camera data is stored there as well as the output frame, all of the memory would need to be available to the video input buffer logic, the convolution logic, and the video output logic. We would need to store 3 separate frames of data for the input and 1 frame for output, pushing us into a 1.2MB data requirement with only 1MB available, making it infeasible.

The next option is to only store 3 lines of input video data as well as an output frame in memory. This option requires logic to modify incoming addresses for the convolution logic as well as the video input logic to implement a rolling buffer. While the output video system would not need to access the same area of memory, the convolution logic still needs to be able to write into the area that the video system reads from. Synchronizing access to RAM is not a concern, but preventing read/write hazards is, requiring a similar memory controller as our first option. With a 10ns cycle time to read/write to the on-board SRAM, the convolution logic would take at least 100ns per pixel, making this option similarly infeasible by violating our processing time constraints.

The best option is to implement a dual-port memory in the logic for the video input logic and convolution logic to work with. Xilinx already has built-in support for this functionality. This solution alleviates the need for a complicated memory controller that must interface with all of the system. Instead a simple read-before-write single cell buffer memory controller can be used to ensure that all outgoing video data is as recent as possible, and avoid failed reads or writes between the convolution logic output and the video output logic input. An address modification system may be used to implement a rolling buffer for the video so that the temporary video buffer does not require any copy cycles. This could be removed by making the stored video lines shift registers and simply running a single cycle copy across them as the image was scanned down. Doing so would complicate the memory design greatly and offer negligible speed increase (estimating maybe 4-5 epsilon in that the translator would just need to be a MUX with a 3-state machine that would take the same time to cycle).

2.2.3.2 Spartan3/Nexys2 Memory Controller

In order to get a frame from memory and output it to the VGA controller, we must design a memory controller. After each image is processed and needs to be outputted, the frame will be stored in the FPGA's SRAM. We must use a memory controller because it is difficult for a synchronous system to access an SRAM directly. The memory controller must take commands from the main system, and generate the proper timed signals to access the SRAM. The performance of a memory controller is very important for the Z-goggle system, because we need everything to seemly happen in real time.

The FPGA board will be dealing with the camera system, image processing, and storing of a frame. Then the board will immediately output the frame data to the VGA controller. The S3 board has two 256K-by-16 asynchronous SRAM devices, which total 1M bytes. The Nexys2 board has less SRAM memory, but works much the same way. This SRAM device has an 18-but address bus (ad), a bidirectional 16-but data bus (dio), and five control signals. The data bus is divided into upper and lower bytes, which can be accessed individually. Table 8 shows the five control signals of the 256k-by-16 asynchronous SRAM devices.

Table 8: SRAM Control Signals [33]

Control Signal	Purpose	Uses
ce_n (chip enable)	disables or enables the chip	To accommodate memory expansion
we_n (write enable)	disables or enables the write operation	Write operations
oe_n (output enable)	disables or enables the output	Read operations
lb_n (lower byte enable)	disables or enables the lower byte of the data bus	To facilitate the byte-oriented configuration
ub_n (upper byte enable)	disables or enables the upper byte of the data bus	To facilitate the byte-oriented configuration

The underscore "n" in the signal names references the fact that these signals are all active low. The main system FPGA needs to communicate with the memory controller, which will communicate with the SRAM. Figure 8 shows the role of a memory controller.
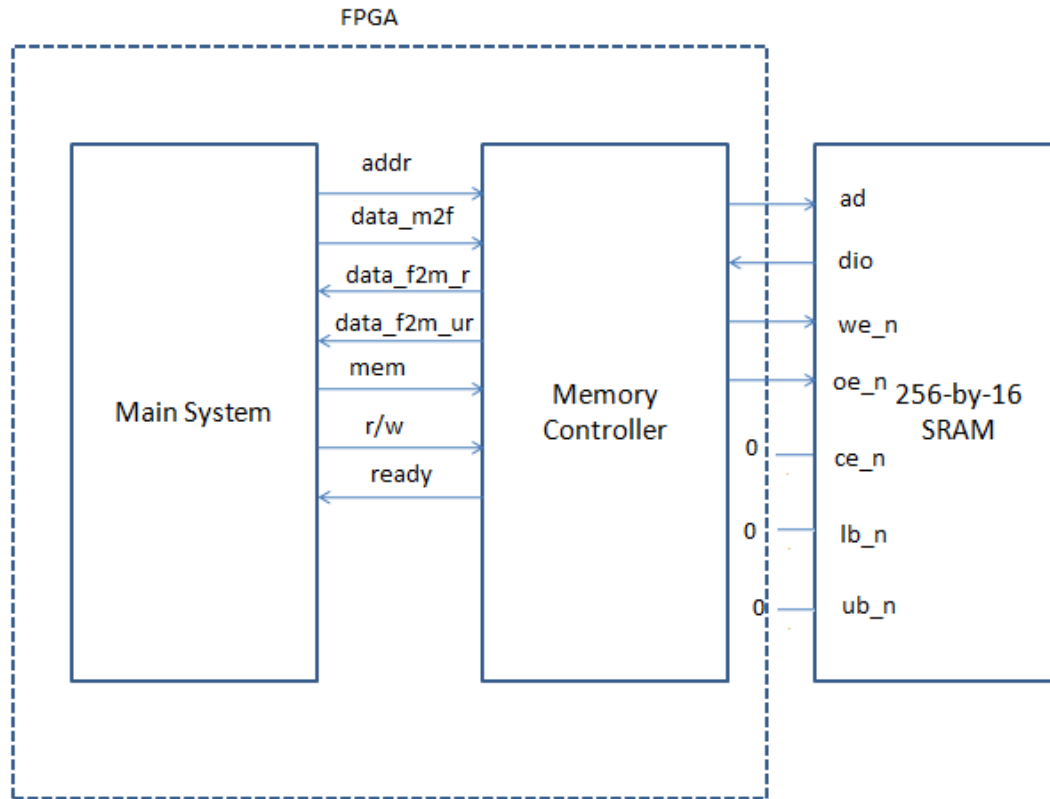
Figure 8: Role of a Memory Controller [33]

We have already discussed the signals from the memory controller to the SRAM; now, let's introduce the signals needed to communicate between the main system and memory controller. Table 9 shows details these control signals for the FPGA memory controller.

Table 9: FPGA memory controller signals [33]

Signal	Purpose
addr	The 18-bit address
data-m2f	The 16-bit data to be written to the SRAM
data-f2m_r	Is the 16-bit registered data retrieved from the SRAM
data-f2fm_ur	The 16-bit unregistered data retrieved from SRAM
mem	Is asserted to 1 to initiate a memory operation
rw	Specifies whether the operation is a read (1) or write (0) operation
ready	Status signal indicating whether the controller is ready to accept a new command. This signal is needed since a memory operation may take more than one clock cycle.

Now that we know all the signals to be sent between the devices, we are left only to design a controller, taking into consideration timing issues based on the time for read/write cycles and the clock frequency. While designing a simple microcontroller with the information above may be simple enough, cutting down on cycles and improving performance to achieve "real time" video output may prove to be significantly more difficult.

2.2.3.3 Experimental Memory Controllers

For using the spartan3 board, we needed to experimentally design some memory controllers. The output video frame is buffered in on-board SRAM as a single contiguous block. This block must be written to and read from different modules on the board, necessitating some type of memory controller.

To ensure that there are no missed reads or writes, a read-before-write memory access controller must be constructed. To maintain timing synchronization to ensure valid data gets to and from RAM, a small shift register must be used. It is constructed from a series of D-flip-flops chained together in a shift register. The first flip-flop's input will be the initiating input, and the final output will be the ready signal. Having old pixel data is not as much of a concern for the rare cases that a read is happening in the exact same location as a write. Even if there was overlap for every pixel, the frame delay would not be noticeable by a human. Writes to video memory will only come from the processing logic, and reads will only come from the video output logic. Guaranteeing the latency of a write is less important than guaranteeing the latency of a read. There should be head room for both to take place in the given time constraints, but one operation must be preferred over another, and it makes more sense to prefer the read operation because the video output needs data right now, and the processor doesn't really care when a write happens, as long as it happens before the next write.

The memory controller has inputs for read request, write request, read address, write address, and clock. It has outputs for memory address and chip enable. When a request for read or write occurs, the logic determines if there's a conflict and if there is, it will buffer the write address, and start a timer to ensure that a valid read occurs before the write takes place. When there is no conflict, the read or write address is selected from a multiplexer and passed to memory. When a conflict has passed and the buffered write is ready to be pushed, the buffered address is selected from the multiplexer. An initial design for this type of memory controller is shown in Figure 9.
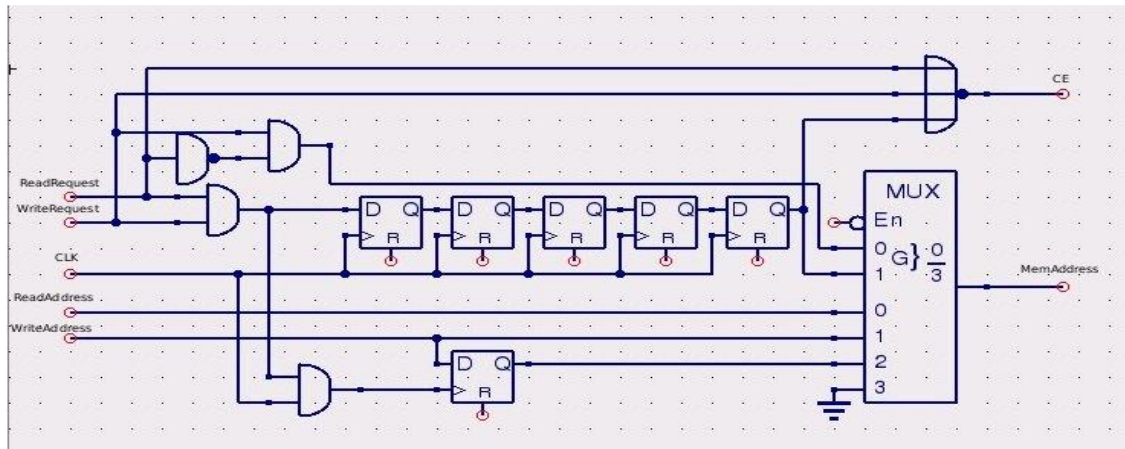
Figure 9: Experimental Memory Controller Design 1

This configuration works so long as there are no reads/writes in the time window in which the buffered write is waiting. If that assumption is not true, a cascade of invalid data can result, requiring a more complex memory request buffering system. In most cases this will be true. The video system will only attempt to read from memory every pixel cycle, so the chance of there being a timing conflict is minimal given that a read and write will take 20ns, and the next read will not occur for at least 54ns. Each write will only take place after the video processor has finished working on new pixel data, which can only be done when new pixel data is available every pixel clock cycle. The only concern is that writes will tend to happen near the end of the cycle while reads will tend to happen towards the beginning of the cycle. Writes do not need to occur within cycle boundaries, so it is possible that for more complex operations, a read will happen while a write is cached. Switching the preference for writes could overcome this problem which is a simple matter of reversing some of the logic. This preference reversal would require additional logic to signal the video output system that valid data was available, or the video output system would need to wait until the middle of the cycle to sample from RAM.

To solve this problem, memory synchronization and buffering of both read and write requests may take place. Separate read and write buffers allow caching of either operation, at the expense of being unable to guarantee read or write times, except guaranteeing they will take at most 20ns. By using an internal wait counter state machine and additional control logic, we can make sure that no reads or writes are missed. A memory controller using these components is shown in Figure 10.
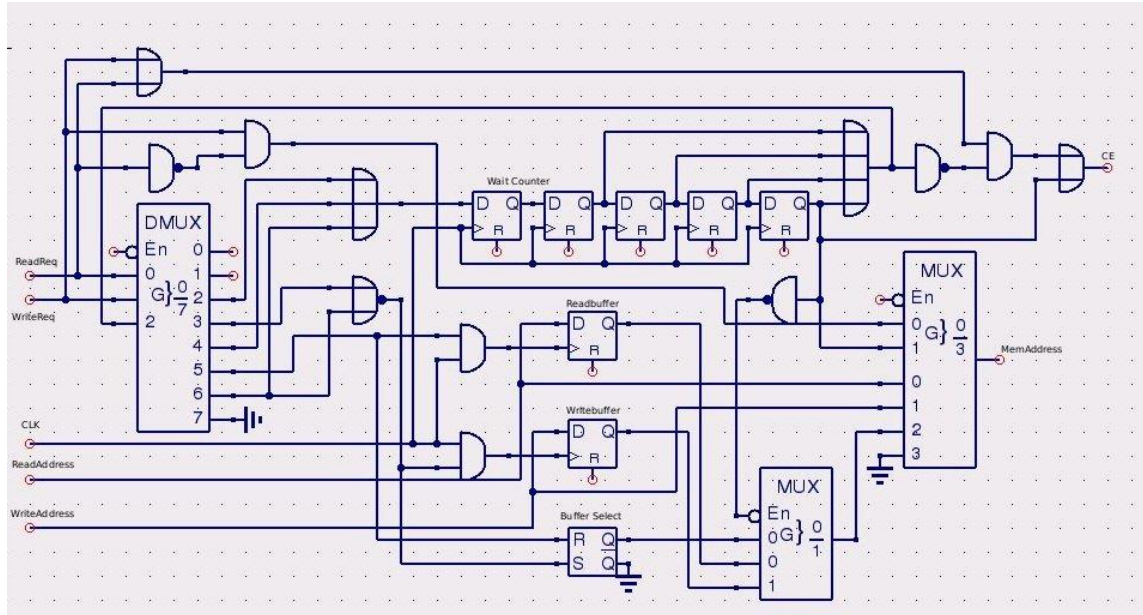
Figure 10: Experimental Memory Controller Design 2

This controller will allow caching of any different types of memory requests. The first stage determines whether there is a memory operation in queue and then buffers the incoming operation if it cannot take place yet. The second stage initiates a counter for any memory operation and controls which buffers are used and checked in the future for use in memory address calls. The third stage controls buffer selection for any buffered memory requests. The final stage determines if there's a waiting memory request, pushes the address to memory, and enables the memory chip. This design is significantly more complicated than the previous one and sacrifices any kind of latency guarantee for either side of the operation.

It would be much easier for the video output system to simply wait 10ns before making a memory request and use a simpler controller in the first place because none of the operations are going to require a write cycle in under 10ns (it'll take longer to sample and latch the new pixel data anyways before issuing a write). We elected to use this type of memory controller because it operates on assumptions that the rest of the system depends on and is significantly simpler than any of the alternative designs. It simply delays a request for memory read by 5 cycles to avoid some kind of data collision. The chosen type of memory controller is shown below in Figure 11.
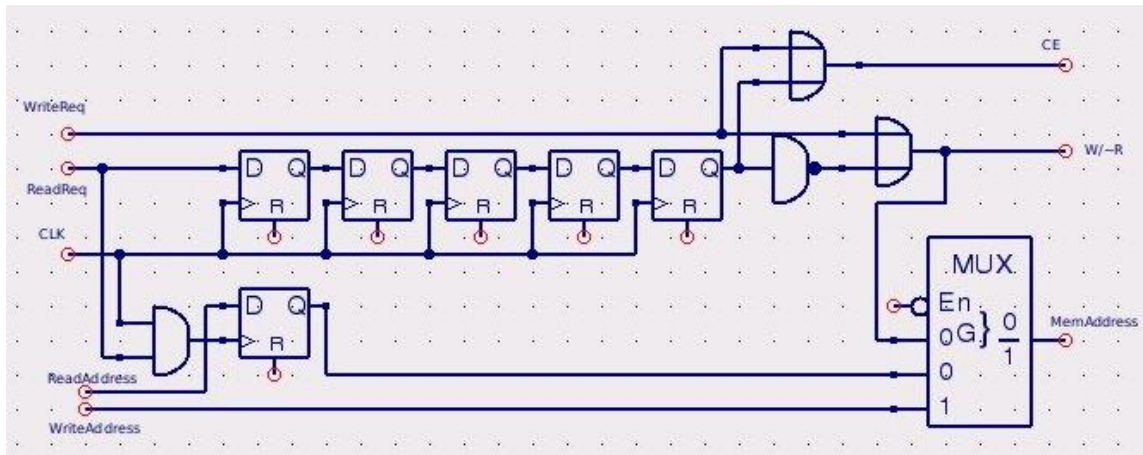
Figure 11: Experimental Memory Controller Design 3

2.2.4 Processing Methods

2.2.4.1 Homogeneous Coordinate Systems:

Homogeneous coordinates for a given XY system can be constructed as (p,q,r) coordinates, where x=p/r and y=q/r. Using a 1x3 matrix [p q r] and multiplying it by a 3x3 matrix:

A represents the horizontal scaling factor. D represents the vertical scaling factor. B represents the vertical shearing factor. C represents the horizontal shearing factor. H represents the horizontal translation. K represents the vertical translation. After multiplication by this matrix, the resultant matrix must be renormalized according to r. The result of this normalization is a new coordinate for the transformed pixel data. This system can be used to rotate images similarly. In the case of A=cos(Z), B=sin(Z), C=-sin(Z),D=cos(Z), the image will be rotated Z degrees counter clockwise. To accomplish these manipulations, multipliers and shifters would need to be used to implement a 2's complement fixed point math unit. Any rotation coefficients would need to be pre-computed and the shifting system adjusted for reasonable limits on accuracy and magnitude. With 18-bit multipliers on the S3BOARD achieving acceptable accuracy should not be a problem.

2.2.4.2 Sobel Operator

The sobel operator is a 3x3 mask applied to a matrix to generate an output value representing the gradient vertically and horizontally of the original matrix. The Sobel Operators for X and Y are shown in the information below.

$$\begin{array}{ccc} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{array} \qquad \begin{array}{ccc} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{array}$$

The result of this weighting and summing are labeled as Gx and Gy. The overall gradient is |Gx|+|Gy|. Beyond a given cutoff to be experimentally determined, an edge will be marked as detected and an output pixel will be produced whose color is dependent upon the gradient results for each color channel. Below the gradient cutoff, a black pixel will be output.

To implement this operator on the FPGA, we need to break down the resulting equations from the application of the Sobel Operator. Given an initial matrix of pixels, we can show that the Sobel Operator can be implemented using a handful of shifts, adds, and subtracts [34].

$$\begin{array}{ccc} A & B & C \\ D & & E \\ F & G & H \end{array}$$

$$Gx = -A + C - 2D + 2E - F + H$$

$$Gy = -A - 2B - C + F + 2G + H$$

$$Gx = (H - A) + (C - F) + (E - D) \ll 1$$

$$Gy = (H - A) + (F - C) + (G - B) \ll 1$$

From this it's clear that there are only a handful of actual computations to be done. The solution can be realized with 8 add-subtract circuits and 2 shifters. These values would then need to be made absolute and summed one final time before reaching a comparator. Each of these would be computed separately for the red, green, and blue channels of the image.

2.2.4.3 Blur Without Multiply:

Blur Implementation without multiply- Normally to blur an image, a weighting map is applied via convolution assuming a pixel matrix from P0 to P8 and a weighting matrix W0 to W8 using this reduction formula:

$$\frac{\sum Wi*Pi}{\sum Wi}.$$

This equation simply reweights all the points in the matrix, sums them, and then divides by the sum of the weights, effectively normalizing the output.

For our uses, to remove the need for a multiplier or floating point formats, we elected to use exclusively powers of 2, enabling the use of shifting. Essentially, this creates an ad-hoc fixed point system. To balance the surrounding pixels, a single shift is used for each and they are summed together. Because there are 8 of them, the shift amount should take an effective 3 bit shift into account. The center pixel which is being operated on is weighted separately so we can pick some kind of balance between the center pixel and surrounding pixels to determine how much the output pixel should look like the center pixel versus its surrounding pixels. The surrounding pixel weight will be known as X, and the center pixel weight will be known as Y.

$$
\begin{array}{ccc}
X & X & X \\
X & Y & X \\
X & X & X
\end{array}
$$

To renormalize this output another shift must be used so that we can be sure we don't overflow and we keep the most significant bits of the computation. We will call this normalization shift Z. All integer solutions for this type of shifting and summing must satisfy the following equation

$$2^Z(2^{X+3} + 2^Y) = 1$$

We wrote a short brute-forcing program to determine valid solutions given the constraint that a shift to the left or right must be now greater than the smallest bitfields number of bits minus 1, because otherwise we would shift all the data out of the register. The options for valid shifts are listed in the table below, referenced as a shift to the left assuming the right side of the pixel data is the lsb.

X	Y	Z
-4	-1	0
-3	0	-1
-2	1	-2
-1	2	-3
0	3	-4

With this knowledge we can create an image blurring pathway using only a few fixed parallel shifters and one accumulator. The solutions were determined by testing to be isomorphic, meaning there is only a single unique solution described by all the solutions. This reduces the complexity of the hardware if the shifting and adding solution is used, but the output will only ever be the average of the pixels in the 3x3 grid rounded down.

2.3 Display

The Z-goggle's display system is the end point for the video data taken in by the cameras. The display system must be working properly for the system to be of any use. While a close-to-eye video glasses display would be idea, such a design would be too complicated and expensive for the scope of this project.

2.3.1 LCD Panels

There are multiple technologies to consider when building a display system. The Z-goggle video goggle system needs to be lightweight, cheap, and power efficient, so using a LCD panel for the display system was a clear and easy choice for the design team. Choosing the type of LCD panel required further research. While different types of LCD displays are available for different uses, displaying video would need a graphical display. The Z-goggle team is also working within a budget, so touch panels were not researched. While the Z-goggle team focused on many factors such as: video output format, display resolution, camera system resolution, price, size, background color, and weight, we were less interested in factors such as: panel fluid type, operating temperature, and packing.

From the start price was one of the design team's major concerns. While buying a LCD panel directly from a supplier would seem to be the obvious choice, recycling a panel out of an old device could be more cost efficient. After some research into viable devices that contained LCD panels with a minimum resolution of 320x240 deemed usable by the design team, a few choices were chosen to look into. Although when a purchasing a LCD screen there is more than just the screen to think about. Building a LCD display system involves a screen, controller board, output ports, and drivers. LCD "modules" or "panels" that may be purchased may be very different things. They could be just a glass slide that will need drivers and a controller, or have a PCB that includes the row and column drivers, or have drivers and a on-board controller.

Since the Z-goggle project focuses on image processing in real time, and the project time constraint is limited, the design team will focus only on LCD display modules that contain a PCB with column/row drivers, and a controller.

The Crystalfontz CFAF320240F-T is a 320xRGBx240 3.5 inch diagonal full color QVGA graphic TFT display module. This display includes several interfaces such as SPI, and 18/16/9/8 bit parallel, so it can be used with many different microcontroller configurations. The display also has an integrated SSD2119 or compatible display

controller, and white LED backlight, with the anode and cathode pins brought out on the FPC connector. The TFT LCD module costs $51.16 for a single unit [35].

While 320x240 is acceptable for the Z-goggle system, the design team's goal was to output the same size image as they were taking in from the camera system. To do this we would need to have a 640x480 sized display.

The Sharp Microelectronics LS037V7DW01 is a 3.7" VGA/QVGA LED Backlit TP 640x480 display module sold by Mouser Electronics. Finding a display module in this size is much harder than 320x240 and results in a much higher price. The LS037V7DW01 display while meeting the designs specifications in size, output format, and backlighting, is sold only in packages and cost over $150 even in a package deal [36].

In a search of a 640x480 TFT LCD display module with drivers and controllers included, the design team turned from the supplier market to the consumer market. In the world we live in, electronic portable devices are in mass demand, resulting in mass supply. Graphical LCD modules are present in a wide range of devices including: smart phones, PDAs, Netbooks, Laptops, Tablet PCs, Handheld Video game systems, and more. To refine the search even more and make the creating the display system easier, we looked into external display devices. A lot of external consumer display devices (like a TFT LCD) feature a analog interface such as VGA or DVI. These devices also have built in internal controller boards that will convert these video formats into a digital RGB data signal at the native resolution of the display as well as control the backlighting of the display [37]. So basically we were looking for a display system already built for us to use. That's where the Sony PlayStation model SCPH-131 LCD screen comes in. While LCD panels straight from the supplier cost anywhere from $51.26 for the Hantronix HDG320240 (a cheap 320x240 non back-light, gray graphical display panel), to $181.23 for the Hantronix HDM6448-1-9JCF (640x480 display panel) [1w], The SCPH-131 Sony model could be found online for about 50$. This 5'' screen was used as a TV screen for the Sony PlayStation released in 2000. The screen can easily be modified into a LCD monitor with AV or S-video inputs, but more importantly because of the system's age in the marketplace, the screen can be found for a cheaper price than buying a LCD panel directly from a supplier.

The downside to using a recycled LCD screen from an older device is the lack of specifications readily available to the design team. The device's manual gives no LCD screen specifications but thanks to a community of device modifiers online the important information was able to be found on this 5 inch LCD color TFT screen [38].

Table 10 below gives reference information for the PSone LCD screen.

Essentially it came down to buying and building a custom display system for a project that calls for a fairly basic LCD monitor, or buying a Display system that needs a small amount of modifications for a lot less money.

Table 10: Reference information for LCD screen [38]

Screen Dimensions	Screen Resolution	Input/Outputs
Width: 4" (10.16cm) Height: 3" (7.62cm) Diagonal: 5" (12.70cm)	640x480 pixels	AV IN jack, headphones jack AV MULTI OUT connector

2.3.2 Display Resolution/Format

The display resolution for and LCD screen is the number of distinct fixed pixels in each dimension that can be displayed. Having a fixed grid display means that, to display different formats of video, the display would need a "scaling engine". A scaling engine is a digital video processor that includes a memory array and is used to match the incoming picture format to the display [39]. Most computer monitors have a scaling engine built into the control logic. Common formats considered for the Z-goggle system included the 320x240 QVGA and the 640x480 VGA. These formats are compared in Figure 12 below.

Figure 12: Comparison of VGA and QVGA resolutions

(Printed with permission. http://en.wikipedia.org/wiki/File:Qvga.svg)

While both of these formats are in the correct 4:3 aspect ratio the design team is looking for, the Z-goggle design team's goal is to match the display system's resolution to the camera systems input resolution to minimize the manipulation of data. The camera system is taking in 640x480 frames, and the FPGA board will be outputting the same size frames to the display through a VGA interface.

The VGA interface uses an analog component video signal. This is a type of signal that is split into multiple components. Reproducing a video signal on a display device is a straightforward process complicated by the multitude of signal sources. Computers process, store and transmit these video signals using a number of different methods, and may provide more than one signal option. To maintain signal clarity the components of

the video signal are separated to prevent them from interfering with each other. This type of separated signal is referred to as "component video", and is used for most consumer-level applications. The RGBHV (red, green, blue, horizontal sync, vertical sync) analog component video standard uses no compression standard and does not limit color depth or resolution, but does require a large amount of bandwidth to carry the signal and contains a lot of redundant data since each channel usually includes the same black and white image. A majority of computers in this day and age offer this signal via the VGA port. The VGA format uses separate sync for vertical and horizontal synchronizing of the video display. The horizontal and vertical synchronization pulses are sent in two separate channels [40].

2.3.3 Display Connections

The VGA interface will be the connection between the LCD display system and the FPGA board. VGA originally stood for Video Graphic Array but over time has evolved to mean the 15-pin VGA connector or the 640x480 resolution itself.

The VGA connector is a three-row 15-pin DE-15 connector. The 15-pin VGA connector is a basic component of many video cards and computer monitors. VGA connectors and cables carry analog component RGBHV (red, green, blue, horizontal sync, vertical sync) video signals. In the original version of DE-15 pin out, there was a mixture of pins that were rarely used and VESA DDC redefined some of these pins adding a +5 V DC power supply pin [41].

To obtain the VGA output the Z-goggle system requires, the design team will be soldering a VGA cable to the pins on the screens attached circuit board. While there are no standards defining the quality required for each resolution a VGA cable is capable of handling, the higher-quality a cable is usually thicker due to coaxial wiring and insulation. A higher quality cable is less likely to have signal crosstalk, which causes signal degradation effects. Shorter VGA cables are less likely to have signal degradation.

Interfacing the LCD display to the VGA port on the FPGA board involves more than just plugging in the VGA cable. While the drivers and controller for the display are built in, getting the right data at the right rate and time to the VGA port on the FPGA board takes some effort. To begin with we need to define some pins to output for the connection to the VGA connector on the FPGA board. The first set of pins required in any VGA system is the clock and sync pins.

The VGA clock pin is important because it has to be set to the same specific frequency the LCD monitor uses, and this must be derived from the system clock on the FPGA board. The VGA clock pin is called the pixel clock [42]. This will be discussed in detail further on.

To understand why we need the sync pins we must first understand how data is sent through a VGA interface. As stated before VGA has three color signals, which set one of these colors on or off on the screen. The intensity of those colors sets the color seen on the display. While the RGB data numbers are digital, they go through a DAC (digital to

analog converter) before leaving the FPGA board's VGA port. The analog intensity is defined by a two bit digital word for each color that goes through the DAC to obtain the correct analog signal. The image sent over the VGA cable is controlled by the horizontal and vertical sync signals that have been mentioned before. The horizontal sync is used to define the end of a line of pixels, while the vertical sync is used to define the end of each whole frame. There is also a composite sync signal that is actually a XOR of the horizontal and vertical signals [42].

After defining the clock and sync pins, we move onto the color data sets. Each color (red, green, blue) is defined with 8 bits, or a total of 24. We will define these pins as 8 bit vectors. Knowing the pins and signals we need is the first step in creating a interface between the image frame in memory, and the VGA port we want to output it to. The key to a correct VGA output is the accurate definition of timing and data by the VHDL. We have defined the pins we would need to use, but we need to understand the timing to continue. The pixel clock we introduced earlier marks the start of a new pixel with every rising edge. Now as we already discussed, at the end of each line of pixels there is a horizontal sync, and at the end of each frame there is a vertical sync. These syncs take time to move to the begging of the next line, or back to the top of the pixel array. The active video signal is 640 pixels, and then there is a time in-between each active video signal where the horizontal sync pulse occurs and the image is defined as a blank space. The signal in-between the active video sections are broken into three sections (the front porch, sync pulse, and back porch), and take up a total of 162 pixels. So all together the signal is for a line of a frame is 640+16+98+48 = 802 pixels. The active video component of that is 640/802 = .79, or 79%. At the end of each entire frame there is another non-active video time interval for the vertical sync. This total frame time is made up of 480 lines of pixels during the active video signal and a total of 44 lines of pixels during the virtual sync signal. The active video component of this is 480/(480+44) = .91, or 91%. Table 11 shows pixel clock rates, and sync pulse lengths (in pixels) for popular resolution formats.

Table 11: Popular Resolution Pixel Clock and Sync Pluses

(Printed with permission. http://www-mtl.mit.edu/Courses/6.111/labkit/vga.shtml)

Format	Pixel Clock (MHz)	Horizontal (in Pixels)				Vertical (in Lines)			
		Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640x480, 60Hz	25.175	640	16	96	48	480	11	2	31
640x480, 72Hz	31.500	640	24	40	128	480	9	3	28
640x480, 75Hz	31.500	640	16	96	48	480	11	2	32
640x480, 85Hz	36.000	640	32	48	112	480	1	3	25
800x600, 56Hz	38.100	800	32	128	128	600	1	4	14
800x600, 60Hz	40.000	800	40	128	88	600	1	4	23
800x600, 72Hz	50.000	800	56	120	64	600	37	6	23
800x600, 75Hz	49.500	800	16	80	160	600	1	2	21
800x600, 85Hz	56.250	800	32	64	152	600	1	3	27
1024x768, 60Hz	65.000	1024	24	136	160	768	3	6	29
1024x768, 70Hz	75.000	1024	24	136	144	768	3	6	29
1024x768, 75Hz	78.750	1024	16	96	176	768	1	3	28
1024x768, 85Hz	94.500	1024	48	96	208	768	1	3	36

Source: Rick Ballantyne, Xilinx Inc.

Based on this information, and what we calculated above, with a pixel clock of 25.17 MHz(for the resolution of 640x480 at 60Hz) this means that the actual image data or pixels for a line are sent in a 25.171µs window in a 31.77µs space between the sync pulses, and each frame as a whole takes place in a 15.25 ms window in the space between pulses, which is16.784 ms. Now that we know how long we have to transfer a frame, we need to make sure we will be able to do it. If we know we need to transfer 640 pixels in 25.17 micro seconds, that gives us 25.17 µs/640 = 39.328 ns per pixel. Assuming a 8bit per pixel length (for 256 colors), that gives us 4.916 ns per bit. With a 50MHz frequency, 20ns clock period, this should not be a problem.

One issue we came across during the VGA interface research had to deal with the Spartan 3 FPGA board we purchased to do the image processing for our project. The VGA port on the Spartan 3 board has 5 active signals. These signals are converted to analog signals. A video signal is represented by a 1-bit word on the Spartan-3 board, and it can be converted to 2^1 analog levels. The three video signals can generate 2^3*1 different colors. We cannot properly display the user's surrounds with only 8 colors. To solve this issue we will be running a parallel FPGA board with the Spartan-3 [43].

Since the Nexys2 board can output only 256 colors, we may want to consider using a separate digital to analog converter if we want to use 16 bit RGB data and have 65536 different colors, or 12 bit RGB data and have 4096 colors. Since the analog signals are still sent out as three voltage levels and a monitor can accept any of these voltage levels, there is theoretical no end to the amount of colors that a monitor can display. While 256 colors may be enough to display images, and make design easier, image processing on a

256 color image may make the image useless. One of the camera modules we may chose uses 16 bit RGB color or 422 YUV color formats, the second camera module uses a NTSC format that we will need to convert to RGB. If we decided to use the first module it makes sense to output the data in the highest color format available, otherwise we will be down converting the RGB data and then significantly reducing out color output. Converting 16 bit RGB digital data to 3 RGB analog signals requires hardware that we do not yet have accessible on the FPGA boards. There are however simple DACs (digital to analog converters) that can be used to do this. The problem is we have data on the FPGA board and if we have it leave the board to run through a DAC, we cannot run the analog signals back onto the board and bypass the DAC on the board to enter the D15 VGA pin connector. So we need to mount a DAC on a PCB and have the analog signals enter a D15 VGA connector mounted on the same PCB. While companies like Analog Devices have a number of DAC video converters available, none of them specifically take in the 565 RGB format that the first camera module outputs. Analog Devices does however have a encoder that will take in 10 bit YCbCr in a 422 format and output analog signals for RGB. This could be used but would require using YUV data for image processing and is not ideal. Averlogic Technologies on the other hand, has an ideal IC for the job. The AL251A-T-0-PBF VGA Video Converter is a video conversion chip for LCD VGA display. It can covert 565 RGB data to analog signals but also has I^2C programmable interface, built in RGB video lookup tables to provide gamma correction, plug and play capability and can run off 3.3 or 5 volts. I^2C interface control allows us to program the chip to work with a number of different resolutions and crop any noise that may be present. It also lets us control the LUT (lookup table) which can help calibrate the color accuracy with gamma correction. The IC also has built in RAM that can store bitmap images we can use to overlay on to any output video via I^2C interface [44]. However this IC was found to only be useful if the 5:6:5 RGB data is interlaced. The design team does not expect this to be the case.

If we choose to use the NTSC camera module we have more choices. Since we have to covert the analog NTSC signal to RGB data, we can use different decoders and covert the signals to either 12 bit or 16 bit RGB data. The nice thing about converting NSTC to 16 bit RGB data is that we can use the same DAC we talked about using for the RGB camera module. We just need to use a video decoder to covert the analog signal to 565 RGB. Due to a limit on the total amount of pins available on the Nexsys 2 board, and the fact that 4096 colors has been deemed acceptable by the design team, we will be looking at decoders and encoders for 12 bit RGB data as well. Assuming we use a NTSC camera and decode the analog signals to a 444 RGB format we could then use a part like Analog Device's ADV7340/7341, which is a multi-format video encoder that can take in 444 RGB data and output RGB analog signals for the VGA controller [45].

2.4 User Interface

The Z-goggle Video goggle system needs is composed of multiple cameras and image processing functions. In order for the user to access these cameras and functions a User Interface must be created. Before deciding on a particular method of interacting with the user, we need to think about everything a user needs to do with the system. The following

use-case diagram is Figure 13 shows that the User has the choice of three cameras and four functions:
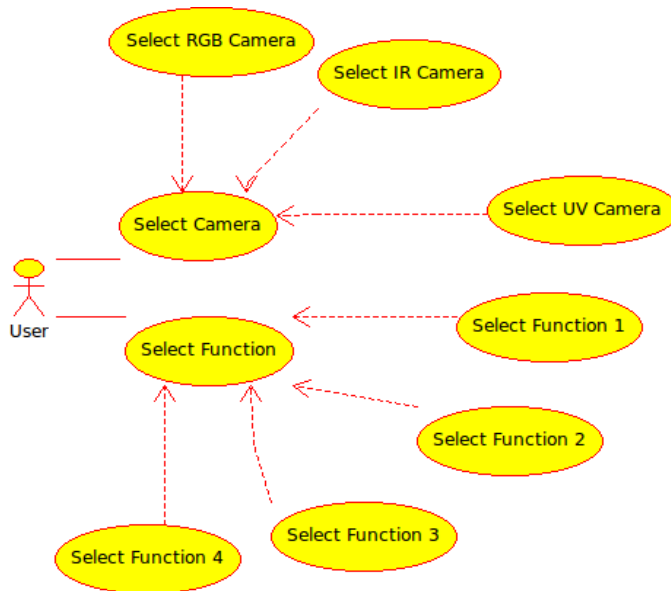
Figure 13: User Interface Use Case diagram

Some functions may be able to be used in parallel with others. For example a function that flips the image upside-down or right-to-left may be used at the same time as a more heavy processing function like shaking vision. This is because while the image processing algorithms will be done in the processor and make be complicated or strenuous on the processor, the simple flip methods will be done in addressing and require very little effort.

2.4.1 UI Choices

If only one camera and one function may be chosen at any one given time, this we prohibit us from "fusing" multiple camera inputs together. Although the design team is worried we will not be able to implement this feature due the small time delay of outputting processed images in real time, and the amount of memory to hold frames from multiple cameras at a time, The UI should be designed to allow for this feature in the future. This removes the option of a camera-knob, function-knob interface.

Possibly the easiest user interface for the Z-goggle system would be a set of 7 separate switches, one for each camera or function. This option would force the design team to design the software of the UI in a way that would handle unwanted input such as multiple functions or cameras switched on at one time. The software could be designed in a manner that would allow multiple cameras or functions, but only in cases that would work. If a particular camera or function is only able to work by itself, this would be fairly simple as it would a series of simple if-else statements that set what is allowed. With the same logic if multiple functions or camera are able to be used together, this could be done.

Now there are endless possibilities to choose from when creating a UI, but knowing what we need to do is a good start. We could create an interface with anything from buttons to voice commands, from knobs to lasers, but the Z-goggle project is about reliability and price. Simple switches may not be flashy but they will get the job done. Switch de-bouncing is another issue the design team will have to address. If you want to input a manual switch signal into a digital circuit you'll need to de-bounce the signal so a single press doesn't appear like multiple presses [46].

2.4.2 Interface Devices

Why use a microcontroller? The switches have to be connected to something to do anything. There are many different ways to interface the switches to the rest of the system. We could use the switches built into the Digilent FPGA board the team is using, but to keep from integrating the UI into the image processing code, the team decided to separate the UI from the FPGA. Microcontrollers are used all the time in automatically controlled products and devices that are too small or simple to be an entire computer, but still need to be controlled electronically. A microcontroller is a small computer on a single integrated circuit [47]. While it may be difficult to program a tiny integrated circuit, we can use a single-board microcontroller (which is a microcontroller built on a board that may have a connection that can be used to connect to a pc) to make it easier to develop an application. This way we do not need to spend time and effort in developing a controller board.

There are a number of design considerations that may help reduce your number of choices to a select few. Some of the important things to consider are: Programmability and Re-programmability, Peripherals, Physical packaging, and Memory. Let's start with the first one.

Programmability and Re-programmability- As research and design team it is important to be able to program and reprogram a microcontroller. Writing code from scratch, we are sure to deal with errors and debugging. This means using a microcontroller that is a one-time programmable chip is not sensible until we have a working design. One-time programmable chips are cheap and may be used in the final design on a PCB, but will not be used for prototyping and testing the system. UV erasable chips tend to be expensive due to the special packaging and will not be further researched for this project. This leaves the options of both microcontrollers that have internal flash or eeprom program memory and microcontrollers that can be used with external memory. Microcontrollers that can be used with external memory will not be further researched because of the small amount of memory needed for the user interface and because the complexity and expense the external memory adds to the system. So with all the other choices out of the picture the design team will focus on microcontrollers that have internal flash or eeprom program memory and can easily be erased and reprogrammed a number of times [48].

Microcontroller peripherals are used for many means of controlling input or output. Some basic useful peripherals are: UARTs (universal asynchronous receiver/transmitter), SPI (Serial Peripheral Interface Bus) or I2C controllers (Inter-Integrated Circuit), PWM (Pulse-width modulation) controllers, and EEPROM data memory. The Z-Goggle video

system user interface remote requires only I/O pins for the switches and power LED, and a small amount of memory [48].

Physical packaging-The physical package of the on-board microcontroller that the design team will use is not very important. The final design will include a custom made PCB with the microcontroller chip placed on it. The package does however make a difference during the prototyping and testing of the project. Ideally controllers with a .1in DIP package is easier to deal with than something with a smaller pin space, but this will not be an important factor in choosing the ideal microcontroller.

Memory-Memory is an important thing to consider when choosing a microcontroller. Some chips may not have enough memory to do the work intended for them. Since the UI for the system at hand is intended to be simple, straightforward, and small, memory is not an issue to worry about. We need only to store the state of at most 9 bits. Three camera switches, four function switches, a power switch, and a power LED.

What the design team needs is an easy to use single-board microcontroller for prototyping and testing, with a simple high level programming IDE(Integrated Development Environments) platform for sensing input and controlling output, so that's what we went looking for. After some initial research two companies products were chosen for further research.

Microchip PIC- PICs are a very popular brand of microcontrollers made by Microchip. They are popular because of their large user base, extensive collection of application tutorials, high availability of free development tools, and serial programming (and re-programming with flash memory) capability [49]. The design team will be using a development board before placing the microcontroller itself on a PCB. The PIC microcontrollers are sold as an on-board microcontroller by the name of PICAXE. The PICAXE boards come in a number of different pin and memory sizes and are available for an affordable price. The programming language for the PICAXE boards is "Basic-like" and support is widely available.

Atmel AVR- Atmel also produces very popular microcontrollers. An Arduino is a single-board microcontroller and a software suite for programming it. The hardware consists of a simple open hardware design for the controller with an Atmel AVR processor and on-board I/O support. The software consists of a standard programming language and the boot loader that runs on the board [50]. Arduino offers a wide variety of boards that could be used for the user interface of the Z-goggle system. Their newest board the Arduino Duemilanove which is a microcontroller board based on the ATmega168 or ATmega32. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button [51]. This board is recommended for people new to microcontrollers but also has more than enough power to do what we need. Arduino also has smaller cheaper boards like the Pro mini, but this board requires more work and is designed for advanced users. Also because we are removing the microcontroller from the board and setting it on a PCB for the final design, the Duemilanove is the preferred product.

2.4.3 Interface Format

Arduino makes formatting the user interface easy. Using a free open source IDE Arduino lets the user program the Atmel ARV microcontroller in C/C++. This is highly desired because it is a language previously used by the entire design team. The code will allow the team to specify variables for the seven switch inputs, and seven outputs. Writing in C, the team will simply write up a loop based event-driven controller program that will continue checking for a change in the input values. When a change occurs, the program will check if the change is valid, adjust the output, and send a digital output stream to the FPGA board. The manner in which the data is sent from the microcontroller to the FPGA board is another decision the design team must make.

2.4.3.1 Interface Protocol Choices

Signals can be sent from one device to another in various different ways, and deciding on a protocol for a particular application requires research into the choices of interface protocols. The user interface switch box microcontroller needs to send a stream of bits to indicate the choices of the seven switches. This stream of bits does not need to be sent in parallel because it timing is not an issue for a simple circuit like this. Sending a series of bits can be done in different manners.

RS232 is a protocol that is widely used for this application. It involves a handshake and data transfer and is fairly simple. The Spartan3 FPGA board the UI needs to communicate with already has a serial port connector on the board, so we just need to make the microcontroller connect and communicate back. The standard that the RS232 protocol was built on uses a -12V to +12V bit range. The Atmega168 microcontroller uses a 0V to 5V standard. To get our 5V controller to talk to a -12/+12V RS232, we will need an integrated circuit that coverts the voltage levels. By setting up a simple circuit involving the Max232 IC, a few capacitors and resistors and the ATmega168 we can establish a serial connection to the FPGA board. The following circuit diagram in Figure 14 shows the Max232 IC connected to a JP2 (a DP9 connector) with the capacitors it needs to operate, and the connections needed for transferring (Tx) and receiving (Rx) data on the Atmega168 microcontroller [52].
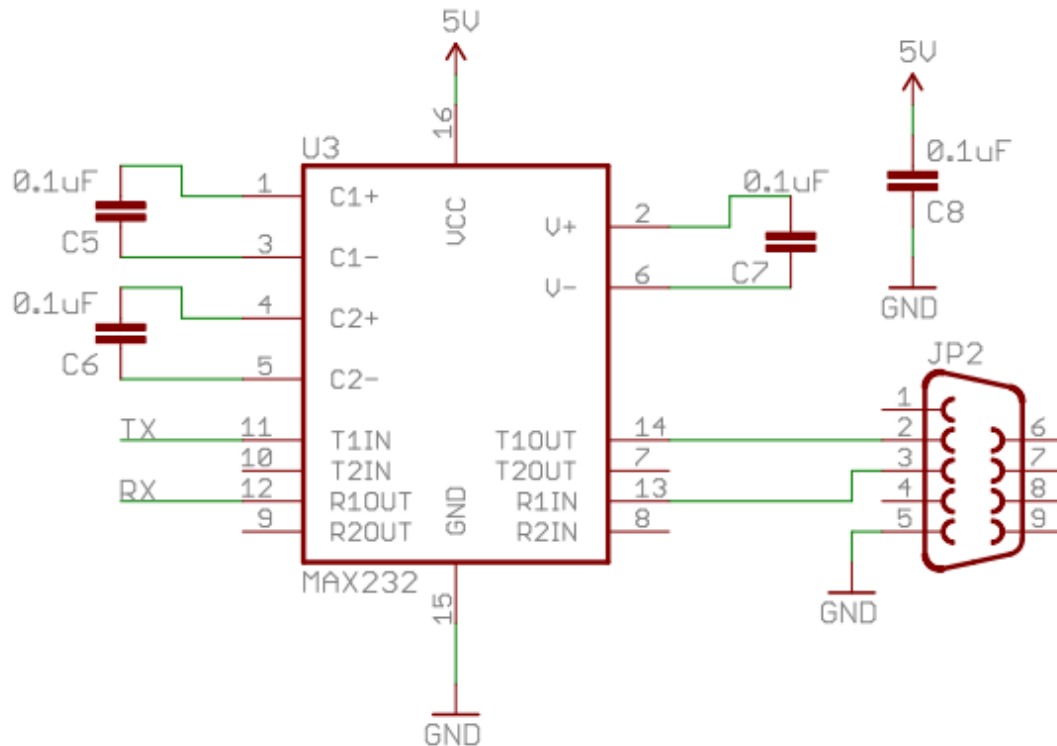
Figure 14: RS232 Circuit

(Printed with permission. SparkFun)

While RS232 protocol is an established secure choice, it may be too complicated for the simple needs of the project. Establishing our own data protocol is another option. The user interface controller only needs to send the status of seven switches, and this can be done with a single stream of bits via one digital I/O pin, or sending seven separate I/O pins (one for each value) in parallel. A series of bits on one wire will take up less pins, but will involve setting up a start/stop bit protocol, or a clock synchronization between the components. Since we are using two FPGA boards and have an abundance of extra I/O pins, setting up seven parallel wires makes more sense. By simply programming the Atmega168 microcontroller on the Arduino development board in their "wiring" C language to check the values of the switches and allow for correct input while dealing with bad input, we can just output the seven I/O pin values as "High" or "Low" and let the FPGA deal with the rest.

2.4.3.2 Interface Algorithm Choices

As for how to actually program the user interface there are various ways. Here we will look into two choices.

1. Using a if-else based program that basically just checks what cameras and switches are selected based on what is allowed, is one way of setting up the code. The following activity diagram shows one way the user interface's software could

be designed. It may need to be edited to allow for multifunction use when the design team is able to test image processing techniques and speeds. This current design allows for only one function and one camera to be selected, but also makes sure functions that cannot be used with certain cameras or vice versa are accounted for. By taking each function column in the above figure and making it into its own "function" in the code, we can then have it check if any other functions that could be possible with the first selected camera and function, are selected. An example of this type of algorithm implementation is shown in Figure 15.

2. Our second option takes a little more planning out, but significantly reduces the complication and clutter caused by the above solution. By looking at the seven switches as a seven bit binary number, we can see we have a total of 2^7, or 128 choices. We can assign the three most significant bits to be the three cameras and the bottom four to be functions like demonstrated in Figure 16. This stream of bits would mean we want to use camera three and function four. Now by using this method we can simply create a database of all 128 choices and set if each particular choice is allowed or not. We can define each number as a variable and simply set it to a "1" if that selection is allowed or a "0" if it is not. Based on the way we set it up, the numbers 0-15 are not allowed because this would mean no camera was selected. The numbers 48 through 63, 80 through 95, 96 through 111, and 112 through 128 are reserved for camera fusion choices and will initially all be set to 0. This means if at a later point in time, the design team decided to include camera fusion only the database will need to be updated. Without fusion in mind we are left with only 45 choices to set. With 4 bits allocated to functions, we have 15 choices of function possibilities per camera. All that is left to do is assign each camera and function to a C #or F # and set the database numbers to 1 if the selection of switches is allowed. This choice is heavily favored because it allows for the easiest modification possible. By simply changing a value in a 128 sized array we can enable or disable a combination of switches
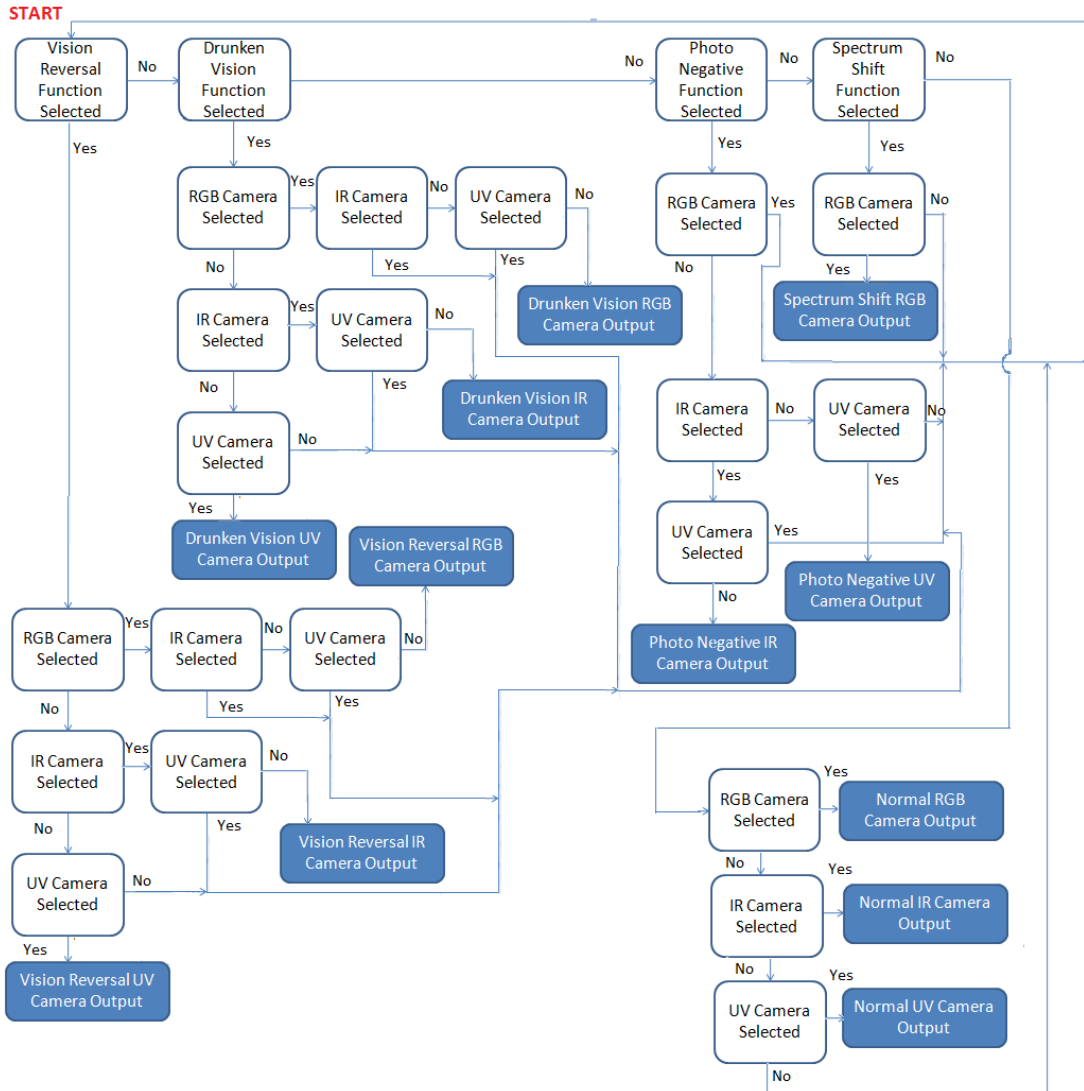
Figure 15: UI Algorithm Choice
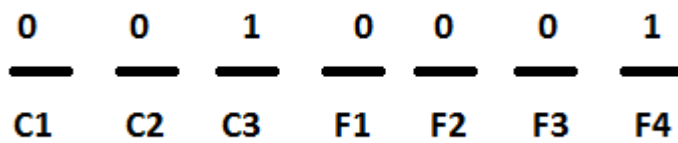
Figure 16: UI Algorithm Binary Format

2.4.4 Use Time

Our goal is to have a use time of roughly 2 hours or less. Studies have been conducted of the long term exposure of the human eye to images used on a screen and some of the results show that if a person were to spend more than two hours staring at a computer screen they will begin to experience Computer Vision Syndrome (CVS). The human eye has been shown to respond very well to images with defined edges with good contrast between the background and any letters or symbols. This helps explain why many feel that reading from regular print of black letters on white paper is much easier to read for long periods than black print on white background on a PDF document for long periods of time. Characters on a computer screen are not made of clear black lines but instead multiple pixels that appear to be clear black lines. Eyes have a very hard time focusing on pixel characters, instead they focus on the plane of the computer screen but they cannot sustain that. Instead they focus on the screen then relax to a point behind the screen, called the Resting Point of Accommodation (RPA) or dark focus as illustrated in Figure 17. This creates a rhythm of your eyes focusing to the RPA then back to the computer screen repeatedly, adding strain to your eyes. Since our screen will be much closer than the average computer screen we feel that this may accelerate this rate and introduce eyestrain earlier than normal. Thus we want the use time to be under this two hour threshold to prevent any damage to the user [53].
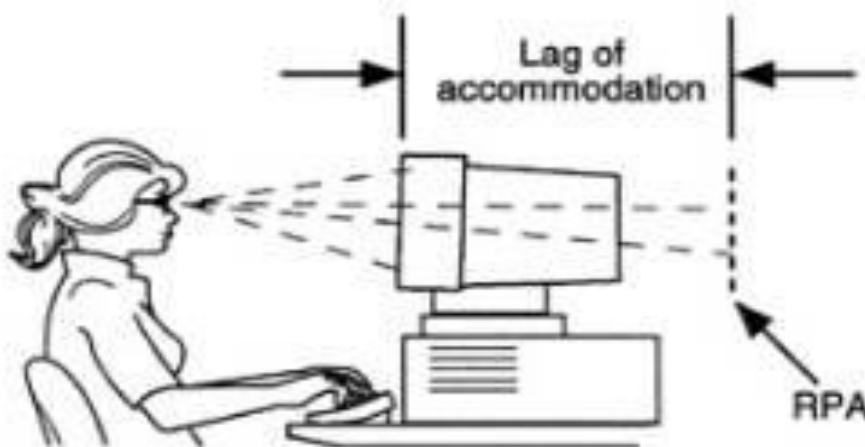
Figure 17: Visual reference of what is Resting Point of Accommodation

 (Printed with Permission. http://www.mdsupport.org/library/cvs.html)

2.5 Control Logic

2.5.1 Coding Options

For controls design of the system, we need to look into what hardware description language would be appropriate. It is possible to design parts of the control system using logic gates on board of the chip, but upon inspection the use of gates would be cumbersome and over complicated. The best choice would be to use one of the two major languages available for FPGA design, Verilog and VHDL. There are two aspects to modeling hardware that any hardware description language facilitates; true abstract behavior and hardware structure. This means modeled hardware behavior is not prejudiced by structural or design aspects of hardware intent and that hardware structure is capable of being modeled irrespective of the design's behavior [54].

2.5.1.1 Verilog

Verilog was first established since 1983 by Gateway, and became a publicly used language in 1990 after Gateway was purchased by Cadence. In December of 1995 Verilog finally became IEEE standard 1364. One of the main talking points of using Verilog is its similarity to the programming language C. This simplicity allows any user associated with a C type language to actively pick up and become experienced with Verilog. Like C, Verilog is a very procedural language that is weakly typed (less restrictive rules on the intermixing of data types) that allows the user to do low level gate design for simple logic circuits. This would be very beneficial for someone who wanted to code some sort of simple user interface that would interact with a hardware switching device on the FPGA itself or on a switch with a digital output. From a beginner's standpoint, Verilog is very beneficial in that there are a multitude of examples existing to assist in a rapid mastering of the language [54][2].

2.5.1.2 VHDL

VHDL (Very high speed integrated circuit Hardware Description Language) became IEEE standard 1076 in 1987. VHDL is a strongly typed language (greater restrictions on interactions with data types) with a relatively verbose syntax and is case insensitive. Low level design in VHDL is locked to simple the two input logic operators NOT, AND, OR, NAND, NOR, XOR and XNOR. Because VHDL is strongly typed, it allows for easier detection and corrections of errors that would otherwise be missed in Verilog. The big advantage of VHDL over Verilog is its ease in doing high level constructs over Verilog such as adding generality to the language to make the code more robust to future uses. VHDL also has the capability of using libraries for future use or libraries of other projects to assist in programming. A graphical comparison of the capabilities of Verilog and VHDL is shown below in Figure 18 [54][2].
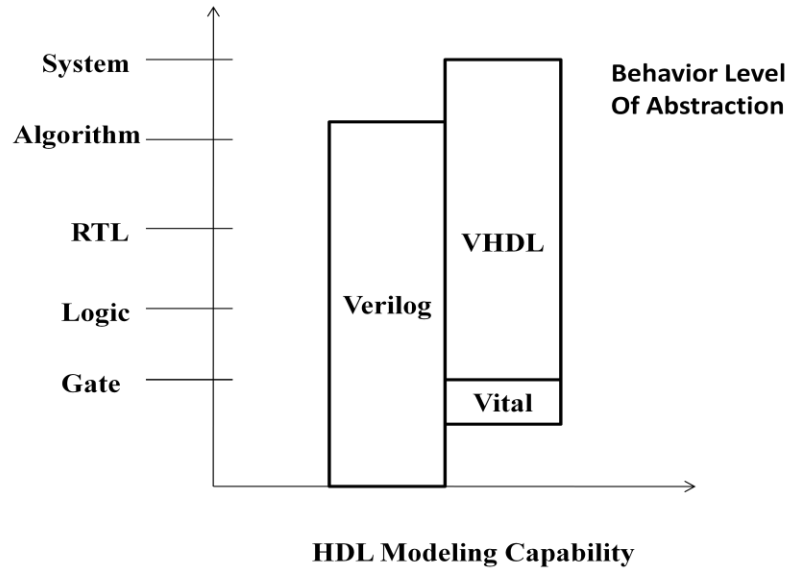
Figure 18: Graphical comparison of the HDL languages

2.5.1.3 C to Verilog Complier

Because of our familiarity with the C programming language we also consider using a C to Verilog complier. This would help out greatly if there are plenty of libraries available to assist in VGA output and interfacing with USB cameras. Because of the lack of experience of using this compiler we feel that this would be the wrong direction to go in as the time available for this project is not forgiving of errors that may arise because of using an untested method.

2.5.1.4 MATLAB Xilinx System Generator

For the DSP design the easiest method to use is a Xilinx package for the Simulink software contained in MATLAB. The system generator contains Xilinx libraries that include blocks for communications, control logic, signal processing, mathematics and memory. System generator for DSP also lets you integrate legacy HDL code, embedded IP, MATLAB functions and hardware models. This would allow for easy and rapid simulation of our DSP and control software without worrying about the possibility of coding mistakes that would take place because of our lack of familiarity with Verilog or VHDL. Use of Simulink would also allow us to create more complex DSP algorithms without the added burden of translating the mathematical functions into Verilog/VHDL. With this software we can also reduce the time to production caused by bugs within the actual Xilinx ISE environment and prevent loss of data [55].

2.5.1.5 I2C Protocol

Because many microcontrollers and interface devices are programmed using I2C protocol we need to explore how that protocol actually works. The protocol was originally created by Philips Semiconductors and allows the communication of data between two I2C devices over two wires. The connection is made serially with a clocking line (SCL) and a serial data line (SDA). For single master mode there is a master device which is in charge of the other devices as it generates the START and STOP signals for the slaves to interpret and act on the received data, this can be seen in the Figure 19 below [56].
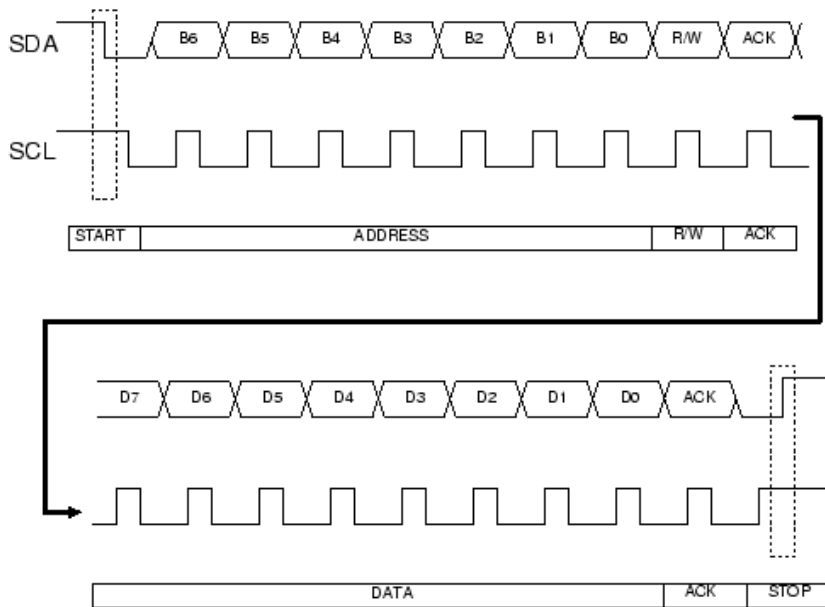
Figure 19: Typical SDA and SCL signals in I2C Protocol

(Printed with permission. Copyright: John Main. http://www.best-microcontroller-projects.com/i2c-tutorial.html)

The only data transfer that occurs is between a master to slave circuit or slave to master circuit, but never between other slave circuits. Because of the robustness of I2C you can have different masters on the same controlling line. This method can get rather complicated because there is only one data line the multiple master circuits would need to fight for control of the data bus as well as the clock synchronization. This can be minimized if everyone runs off of one single clock. The lines that are running form the master to the slave(s) are bidirectional in nature meaning that any device can drive the line. In the case of the slave systems the master is in charge of the clock rate but the slaves can influence the clock to slow down. The wires must be driven as open collector/drain outputs and need to be pulled high by a resistor on each. This is a wired AND function meaning that any device pulling the wire low causes all devices to see a low logic value, while a high value will cause all other devices to stop driving the wire. Because of the nature of this, you cannot use any other I2C devices on these lines because both are being used as a clock at some point. The data transfer from a master

device to a slave device in I2C is fairly straight forward. The figure above also demonstrates how a master to slave transfer protocol will work out. First the master sends out a start signal to tell the slave that data transfer will begin. Then there is a 7 bit address of the slave device to be written to or read from, followed by a read/write bit. Each device must have a unique address, especially if you want to have multiple devices on the same data bus. Immediately following this the slave sends back an acknowledge bit signifying that the data sent matches the data received. After the acknowledge bit the master begins to transfer an 8 bit data word to be stored at the address received, followed by another acknowledge bit. This sending of data followed by an acknowledge bit continues continuously until the master sends a stop bit signifying that the transfer is complete. In the programming of the circuit there will be an algorithm that will compute error correction if the acknowledge bit comes back wrong. Because this is such a slow protocol for controls this will likely be used for non high speed controls such as camera control or user interface control [56].

2.5.2 Microcontrollers

For our project we will need some method of ensuring that the data coming from the camera system, from the user interface and video data being sent to the display system is available and ready to execute the function it has been determined to complete. For the data that is coming from the camera system we will need to determine the data type and from what source it is coming from. The common type of input being received from cameras today follows the USB 2.0 protocol. As for the user interface, the data being received will be some form of binary data being produced by the number of knobs or switches activated by the user. For the video output data the display will be set as a slave and the controller as master. The most important aspect that needs attention is that the out data type will match up with the data type the display is designed for. If we have a display that is designed to accept a VGA protocol input then using a QVGA output from the controller will result in a breakdown of the system. Also for this system we will need to have built in some method of error checking and possibly error correction. This will be extremely important with the camera input and DSP interface, because if the data coming into the DSP is faulty then the functions applied to the data will be outputting faulty data to the display. Under consideration are multiple controllers that could complete the tasks set forth, in either some sort of master controller that will be responsible for all three interconnects or a mix of microcontrollers and a master controller. We will need to consider the available processing speed of the controller as well as they type of memory and more importantly the memory access times available. For the system to be working in real time and at minimum of 15 frames per second the memory access will probably be our largest limiting factor. The amount of accessible memory should only be enough to store the incoming and outgoing data.

2.5.2.1 FPGA Embedded System

An FPGA is a Field Programmable Gate Array that is onboard of a digital integrated circuit which would contain programmable logic with configurable interconnects between such logic. FPGAs can be implemented to perform such tasks as ASIC

(application specific integrated circuit), digital signal processing (DSP), embedded systems, physical layer communications (interface between communication chips and high level networking), and reconfigurable computing (dedicated hardware for software algorithms). Our main consideration is to implement the control logic as an embedded system on board the FPGA that is being used to do the DSP functions. A great advantage of the use of an FPGA is its ability to be programmed to do large and complex functions with retaining its ability to be reprogrammed at the user's behest. Most FPGAs today have a hybrid of embedded microprocessor cores, high speed input/output (I/O) interfaces and expansion ports to further integrate into a larger system [57].

One form of memory used in large scale production employs an antifuse technology which once programmed becomes permanent in their function. The purpose such technology is that their configuration is non-volatile so once it is powered down the configuration stays with the system. This would be ideal for a master chip which copies are made of and for large scale consumer use so that your technology isn't being used for a purpose that it was otherwise designed [57]

Most FPGA's memory is built on SRAM technology instead of antifuse. SRAM stands for Static Random Access Memory, which specifically is a type of RAM that once a bit is loaded into memory is will stay there until it is changed by an operation or power is removed from the system. Access time differences between antifuse and SRAM are negligible so the main design factor is the period of which the programming is to be stored. Even though the memory is erased after power down it has the great advantage of being able to be reprogrammed to execute such functions as a system check and hardware check for the FPGA and be programmed with its specific function. Since the memory is lost once the device is powered down, any programming that is wished to be kept needs to be written on an external memory device or stored on an onboard microprocessor. For our purposes the memory will be used to store temporary data such as user interface choices or video data with low access times [57].

To compliment SRAM memory, many vendors use FLASH based memory to store the actual programming for the processor. FLASH is very similar to SRAM except that the data is non-volatile in nature, allowing for an "instant on" state once power is applied. The memory can be programmed using an external system (computer) or programmed within the system but the programming time is roughly 3 times that of SRAM, making its access time much slower than needed. With a hybrid system of SRAM and FLASH, the contents of the FLASH memory can be copied into the SRAM cells at startup for speedy access of the programming.

For the actual processing ability of the FPGA we can use a soft or hard microprocessor core. A hard core microprocessor is a separately made microprocessor that is either interfaced with the FPGA itself or one that is embedded on the FPGA chip itself. These are generally regarded as having the most processing ability and speed. Generally hard processor cores are implemented as dedicated predefined blocks on the FPGA. On the chip they exist as a "stripe" of the processor, RAM, I/O, multipliers and other peripherals or are actually embedded on the FPGA fabric itself. Any memory used by the core is formed from embedded RAM blocks and peripheral functions are formed from groups of general purpose programmable logic blocks. Soft processor cores are groups of

programmable logic blocks that are configured to act as a microprocessor. These are generally slower and more primitive than their hard-core counterparts. One large advantage is their use of "real estate" in the FPGA so that they can be implemented as needed and one could create as many processor cores until there are no more logic blocks to implement them. This has the inherent advantage of allowing the user to create multiple processor cores for multiple functions allowing large scale parallel processing. Because these cores are not originally designed as processors from the start but built instead they are slower in their general speed [57].

2.5.2.2 Multi FPGA System

Some processing loads are too much for a single FPGA to handle. To solve this problem we are able to use multiple FPGA systems to split up the work to be processed in parallel or have a multiple stage based system. There are multiple configuration methods such as a master/slave configuration as well as parallel or serial configurations. The most straight forward method is configuring the FPGA as master and the load device as slave. A great example is the interfacing of an FPGA with a memory device. In this situation the FPGA will use several bits to control the memory device such as a reset signal to indicate it is ready to start reading data but does not indicate a specific address that will be read from. Instead the FPGA will be simply reading from the beginning of the device and stop at the FPGA's behest. This is best used when there is a device that the FPGA does not have built onto the original board it came with and the auxiliary device is simple such as memory. There is also the configuration of an FPGA in parallel and the FPGA is set as master. For the same situation as above, the memory device will now be given control, address and configuration data. This would allow the user to configure off-the-shelf devices instead of specially designed devices that would be expensive [57].

The FPGA could also be set as slave and another peripheral would be set as the master. Generally this would be done in parallel with a separate microprocessor that is designed to load the FPGA with configuration data. The driving factor is that the microprocessor is in control and would then configure the FPGA as needed. This would allow the processor to read from a peripheral that is connect and write whatever data is needed into the FPGA and use the FPGA as another stage for the recently written data. The FPGA could still be used for processing work but most of the "thinking" would be done by the microprocessor. The serial configuration is almost identical to the parallel configuration but only a single bit could be used to load data as opposed to the parallel which generally uses a byte (8 bits) of data to write at a time. The microprocessor would still read data byte by byte but that data would then be converted to a single bit stream. It is obvious that there is an inherent speed advantage with using a parallel configuration, allowing only one cycle of computer time to write 8 bits of data instead of one cycle to write a single bit. The main advantage of a serial interface is to conserve the number of I/O pins being used on the board. This would make sense for a system that does not require high speed and where parallel would add undue burden on the designers [57].

Another method of interfacing with peripherals is to use the JTAG (Joint Test Action Group) port that is normally placed on all FPGAs. The JTAG port was originally designed to implement the boundary scan technique to test circuit boards and ICs. The point of the boundary scan is to allow data to be serially clocked into the registers associated with the JTAG port for the data to be operated on by the FPGA and the results would automatically be stored on output JTAG registers to be serially clocked out to another peripheral. The JTAG port can also be used to program the FPGA by commanding the FPGA to connect to the internal SRAM or other RAM installed on the device [57].

2.6 Power Supply

The power supply subsystem for the Z-Goggles consists of multiple devices requiring battery power, including the Nexsys2 FPGA that normally runs from an AC wall adapter that provides 5V. It will be necessary to design a multi-load power supply to run the system. To accomplish this task, a source option must be acquired that meets the voltage and current requirements of the system. In addition, software for design purposes must be explored and the best solution chosen. This was determined to be a better decision than selecting each part manually, for reasons of time and lack of power supply design experience.

2.6.1 Source Options

There are generally two types of sources powering a system, battery power and connection to a stationary source such as a general power supply or a wall outlet. Both present their own unique problems with viable solutions. The main consideration for this project will be the use time and portability we want for the device. Depending on what is being used for a processor, camera system and display the power draw will vary from device to device but remain within the same range from 1.2 V to 12 V. Most of the components of the system have to be powered by the power supply, though some can be supplied by the voltage regulators located on the FPGA.

2.6.1.1 Batteries

A battery supply will give maximum portability for the system, allowing the user to be free from any wires that would restrict his/her mobility. There are many batteries available today that can supply the specific voltages required for the Video Processor and display. To supply the different voltages required by the chips, we can use separate batteries for each or we could create a voltage regulator circuit. The circuit's responsibility would be to ensure that the appropriate amount of voltage and current from the source battery are being applied to the correct load. The choice of battery is supremely dependant on what is chosen for the other components of the system. Many FPGAs have some sort of voltage regulator circuit built in, allowing us to solely supply the FPGA and use it as the voltage regulator for some parts of the system. This would

work out best if the predetermined voltage regulator levels in the FPGA match up well with the components of the Z-Goggles system. Since all circuits being considered run off of a DC voltage, a battery choice would be the least intensive and most predictable supply being considered.

There is a large selection of battery types available to choose from with each offering different available voltages, mAh (milliamp hours; the accepted standard of measurement for the charge capacity of batteries) and size. What we need to consider in each of these options is the amount of mAh, size and more importantly the cost of each unit [58].

First to be considered are the family of alkaline batteries available such as the AA, AAA, C, etc. These each correspond to the standard size associated with the battery for general consumer use, and alkaline batteries are the ones most commonly seen for everyday use. For the price per quantity, alkaline batteries are cheapest but are lacking on the number of mA/h over a consistent period for power intensive applications. Some of these are able to be recharged by using an external charging station. We may not need to run the system very long, so the cost of the charger may be over our budget goals. Luckily the voltages that we require mostly range from 1.2 to 5 Volts are commonly available in alkaline, thus allowing us use singular batteries for each device as well as not being needed to build a voltage regulation system to get specific voltages. Regular alkaline batteries are non-rechargeable, whereas their alkaline manganese counterparts are. With this type of battery, you must take care not to overcharge the batteries for fear of electrolysis of the electrolyte solution or charge at over 1.65 volts to prevent the formation of higher oxides of manganese. Environmentally, alkaline options use no toxic metals and the recharging of them results in less being discarded. The shelf life of rechargeable alkaline batteries is roughly at 10 years with the option of being used immediately [58][59].

Nickel based batteries are rechargeable and generally have a strong development history and future. First there is Nickel-Cadmium (NiCd), which has been around since 1899. Originally these batteries were expensive because the materials were costly to produce and manufacture. As modern techniques allowed these processes to be less costly their use started to become more frequent. NiCd is the only battery type that has been seen to work under rigorous conditions, while other battery types prefer shallow discharge rates and moderate load currents. Some advantages of NiCd are that it is fast and simple to charge, even after long periods of storage. If properly maintained, NiCd should have over 1000 charge/discharge cycles, which is quite high when compared to other battery types. The batteries can also be recharged at low temperatures and good low temperature performance when compared to other batteries, which add to its robustness to adverse conditions. With cost being a factor, NiCd is the lowest priced in terms of cost per cycle as well as being available in a wide range of sizes and performance options. As for disadvantages, while being charged, NiCd needs to only be charged as long as needed and should be used for long periods instead of brief periods to prevent loss of performance due to crystal growth on the cell plates. Also, NiCd has a relatively low energy density and a high self discharge rate resulting in a need for recharging after storage. Environmentally NiCd contain toxic materials that need proper disposal [59].

The other popular derivative of the Nickel based batteries is Nickel-Metal-Hydride (NiMH) type. These batteries were first being researched in the 1970s and are now

mostly used for satellite applications. As opposed to NiCd, NiMH is less durable such as cycling from heavy load and storage at high temperatures reduces the service life. Conversely, NiMH offer up to 40% higher energy density when compared to NiCd. While higher capacities are still possible, they have their own negative effects that prevent them doing so. Lately NiMH has been replacing the NiCd markets in wireless communications and mobile computing but is running out of room to grow because of some nickel based issues common to both types. Other advantages of NiMH include that it is less prone to crystal growth unlike NiCd and much more environmentally friendly with the option of being recycled. Some problems with this type of battery include is a much more limited service life with performance being diminished after 200-300 cycles if repeatedly deeply cycled. Also the shelf life is limited to 3 years, though cold temperature and partial charge slows this down. It is also capable of high discharge currents but heavy loads begin to limit the battery's life. NiMH requires high maintenance such as a full discharge and needs to be used at least once every 3 months to be considered in proper condition. Finally, NiMH requires a complex charging algorithm; it generates more heat during charge than NiCd trickle, so charge settings are critical because the battery could become damaged from over charging [59].

Lithium-ion is perhaps the most recognizable type of lithium based battery available today. Lithium based work has been ongoing since 1912 but it wasn't until the 1970s that non-rechargeable lithium batteries became commercially available. Lithium is an extremely light metal and the highest electrochemical potential, making possible the largest energy density to weight ratio. Rechargeable lithium batteries were unavailable for a long period because of the inherent instability of lithium metal, especially during charging. That forced the industry to instead invest their research into lithium-ion, which is safe as long as certain precautions are met during charging and discharging. Lithium ion's energy density is typically twice the size of the standard NiCd based battery. Also the load characteristics are very good and comparable to NiCd with a high cell voltage of 3.6 Volts. Many mobile electronics such as phones can run off a single cell where it would take multiple NiCd cells to match the same voltage. Lithium-ion is also a low maintenance battery with no crystal formation and scheduled cycling to maintain its shelf life. The self discharge rate is roughly half that of NiCd, allowing for the charge to remain on the battery for long periods without immediate recharging after storage. Lithium-ion is also safer for the environment, having little harm being associated with it when being disposed. Sadly, there are some drawbacks to lithium-ion batteries. The battery is fragile, as it requires a protection circuit to maintain safe operation. This circuit limits the peak voltage of each cell during charge and prevents the cell voltage from dropping too low during discharge. The maximum charge and discharge is limited to 1-2 Coulombs to prevent the possibility of metallic lithium plating occurring due to overcharge. Also some capacity deterioration is apparent after roughly a year, whether through use or not and failure usually resulting after two or three years. With this, there are some that can last up to five years in specific applications, though with modern advances, this limit is constantly being pushed. Cold storage can also slow down the deterioration process as well as the battery being partially charged during storage [59].

As well as lithium ion, lithium polymer is also a viable option. It is different from conventional battery systems because of the different type of electrolyte used. The type

used more frequently is a dry solid polymer instead of a porous separator soaked in electrolyte. The dry polymer allows for a more simple manufacturing process as well as a thinner profile allowing for more unique shapes and forms to be used. Previously the dry polymer suffered from high internal resistance because of the low conductivity associated. This problem has decreased because of a gelled electrolyte, allowing the lithium-ion polymer cells to be very similar in chemistry to their liquid electrolyte counterparts. This gel also helps improve the safety of the batteries because of the smaller chance of an electrolyte leakage from overcharge. When compared to lithium-ion, the lithium polymer batteries have a lower energy density as well as a decreased cycle count. Also, because the market demand is low the cost to manufacture is greater than lithium-ion as well as a higher cost-to-energy ratio [59].

The oldest type of rechargeable battery, lead acid batteries were the first rechargeable battery to be used in commercial applications since being invented in 1859. Their robustness can be seen in how the auto industry still hasn't found a cost-effective alternative for their car batteries. The same can be said for many other applications such as wheelchairs, scooters, UPS systems, etc. The charging of lead acid is an extremely critical characteristic that needs to be kept track of. If overcharging is to occur then the possibility of thermal runaway occurring is very high as well as the destruction of the battery itself. If the battery is rated for high voltage then the battery needs to be maintained at a minimum charge to maintain good performance and the possibility of being recharged. Leaving the battery on a float charge for long periods does not cause damage to the performance or to the shelf life. With these batteries you must also be aware of deep cycling the battery; a full discharge will cause extra strain and lower the service life of the battery, thus large batteries are optimum to stay away from the lower range of the battery's voltage. For high voltage lead acid, the batteries themselves produce good performance but suffer from a shorter service life from corrosion to the positive plate [59].

The shelf life of lead acid is a big advantage because it has one of the lowest self discharge rates, almost 40% of the best rechargeable batteries on the market today. For application we will need a sealed lead acid battery instead of a flooded lead acid battery to allow the battery to operate in any position. These batteries are generally maintenance free because a gelled electrolyte solution is gelled into moistened separators, whereas the flooded lead acid batteries need to stay in an upright position to prevent leakage of the electrolyte solution onto surrounding surfaces. Sealed lead acid has safety valves to allow venting during charge, discharge and atmospheric pressure changes. Sealed lead acid batteries are designed to have a low over-voltage potential so that harmful gasses cannot be produced during charge that would destroy the battery, though because of this the batteries cannot be charged to their full capacity. Lead acid batteries would be our strongest choice because of their low cost and high discharge rates. Lead acid batteries in general are not environmentally friendly because of the high lead content, therefore if we are to dispose of the battery we will need to utilize special services to ensure proper disposal [59].

A newer variety of battery is called the Absorbent Glass Mat (AGM) that uses absorbed glass mats between the plates. These plates allow the battery to recombine 99% of the

oxygen and hydrogen that exists in the battery during discharge. This also means that there is no water loss during discharge as well. The charging voltages of the AGM batteries are the same for other lead acid types, with at worst conditions there is a hydrogen emission that is below the 4% specified for enclosed spaces. The cost of AGM is roughly twice that of flooded and enclosed batteries but because of today's manufacturing process that cost is still low enough to be considered cheaper than lithium based or nickel based [59].

We will be requiring a battery with enough voltage but more importantly we need a minimum amount of amp hours to maintain our system for our use time goal. In particular we are considering the use of Power Sonic brand batteries, one which is 12V and rated for 5 A/h and one which is 12V and rated for 7 A/h. The amount of A/h is important for our ability to have a decent run time. The 12V batteries are equal to the max voltage of any of our devices and we are confident that the amount of A/h is enough to reach our goal for the use time, regardless of which battery is chosen for the final design [59][60][61].

2.6.1.2 Wall Outlet

Our other choice for a power supply would be to use the standard US wall outlet with a 120/240V, 60Hz output. This can be turned into a desired DC input, but would require the creation of an AC/DC converter circuit or purchase of one. We would also need to step down the voltage greatly from 120/240V to prevent any unwanted destruction of the DSP and display. This could be done with resistors attached to the converter circuit or we could use a transformer. The use of such would also add increased risk of electrocution as the converter circuit would ideally be attached to the user so the only thing that would not be attached to the user is the power cord itself. Because of the inherent danger associated with using wall outlet power and the low consumption nature of our devices this will be largely an auxiliary choice. Also, there is no real portability associated with this option, as the user would always be tethered to the outlet in some way.

2.6.2 Portability

The goal of this system is to allow the user to view an alternate reality that we are producing through various DSP effects to a real time video feed. To best immerse the user into this alternate reality, the user's ability to move and interact with the world will give a heightened sense of realism to our alternate world. Maximum interaction would be achieved when the user is not impaired in any way by the device, allowing the user to travel at their preference. This would best be achieved using a battery system but only if the battery is not too heavy or oblong to impede on the user. Alternately, with a wall outlet source, the user would be free of any heavy battery that would need to be attached but grant the user only limited distance to travel. This may become economical if we find that the average use time is limited so that distances within the average extension cord length are adequate.

2.6.3 Power Supply Design Software

When designing a multiple output power system, it is most important to keep track of how every part works together. Software significantly lowers the risk of design mistakes by providing a basic launch-pad for design work. This lowers the amount of time and confusion spent selecting components and allows more time for testing and verifying the design. The design team needs a rapid design solution for the power supply. After searching various vendor sites for power supply information, the design team came across several sets of tools for complete power supply design. Each vendor offers tools that create designs with exclusively their parts, but the rapid and complete analysis afforded by these design solutions is too useful to ignore. The first set of software researched includes WEBENCH and Power Expert for Xilinx FPGA's from National Semiconductor [62]. These are both free to use, and simple enough for beginning designers. Also, Linear Technology offers a free spreadsheet planning tool called LTpowerCAD that chooses parts and components based on user specifications. In addition to this, most of their parts are available in Spice models for simulation purposes [63].

WEBENCH covers the entire power supply design process, and allows the user to select input and output characteristics. It creates both single and multi-load power supply designs. The most useful feature is that gives relatively detailed analysis for such a simple program. For instance, it allows optimization of various factors, including cost, size footprint, and efficiency. Also, users can do electrical and thermal simulation of the completed design. When finished, the design schematic and bill of materials is available, which allows immediate purchase for build and testing verification. Similar to WEBENCH, the Power Expert program is more specific to designing power supplies for Xilinx FPGAs. While the board we are using is not a Xilinx FPGA, it will help the group to understand the way an FPGA is powered [62].

As a comparison, LTpowerCAD can help a designer find parts for a complete system and leads the user through the process with suggestions. One perk of this program is the ability to export the design to LTSpice and verify designs via simulation in that environment. Also, it gives an example PCB layout, which may be a great time saving measure, given that it is fairly close to what would actually be required. Unfortunately, LTpowerCAD has the following statement in its Terms of Agreement for use:

"You may not use Licensor's name or other trademarks or refer to Licensor or Licensor's products directly or indirectly in any papers, articles, advertisements, sales presentations, news releases or releases to any third party without the prior written approval of Licensor for each such use. You may not release the results of any performance or functional evaluation of the Program to any third party without prior written approval of Licensor for each such release." [63]

Due to this restriction, and the fact that permission has already been granted by National Semiconductor for including any results from WEBENCH, the National Semiconductor offerings will be used to aid in the design of the Z-Goggles power supply. The schematics

and simulation information from this design will be included in section 3.7.2 Power Supply Design. In addition to the legal reasons, this software was easier to use, and the partner software may provide information on FPGA power design. The Z-Goggles are based on FPGAs, so any additional insight into their operation is invaluable, and could prevent a lot of headaches during building.

2.7 Helmet/Backpack

The helmet/backpack combination is the acting container of the entire Z-goggle system. They are what make the system wearable and portable.

2.7.1 Weight Issues

The Z-Goggle system is meant to be attached to the user's body in a manner that allows for easy movement. Before deciding how and where to place all the components of the system, we must first know relatively how much each component weighs. Due to the fact that the weight of each part of the system is not readily available before purchasing the system, we have to estimate the weight.

The camera sub-system and display sub-system are most likely the heaviest parts of the total system. Both of course need to be placed on, or attached to the user's head. To do this we will require a sturdy helmet that may be adjusted to fit a number of different users. The FPGA and control logic may also be placed on the helmet to keep wired connections short and data transfer fast. To keep the user from being so top heavy the design team will have to spread out the weight to both the front and back of the helmet.

The system's power supply weight is most likely to be supported by a backpack-like device. While other options include a vest or belt, the power supply will not be attached to the helmet.

2.7.2 Device Placement

Device Placement on a body-wearing system can be very important. Not only can the placement of each component affect the overall performance of the system, but can also affect the safety of the user. If connections are too far apart the delay between the camera system and the display system may increase and effect the user's perception of "real-time". If the connections are not secured safely the entire system may become nonfunctional through wire disconnection or power failure. If the more delicate electronics (like the FPGA board) are not secured to a solid flat surface, that also is designed to keep the system from over-heating, the whole system is again at risk.

After taking into consideration all of these factors multiple designs were formulated for the device placement of the system. Here we will briefly describe two general designs.

1. The first design features a helmet with a built in flat-top surface. This surface could securely hold the delicate electronics and the camera system. By keeping

the camera system and FPGA board close together, we also help to minimize any delay in getting data from one point to another. Shorter wires also help develop a less "cluttered" design and keep the connections easy to maintain. The power supply would be attached in a backpack-like vessel due to its larger weight and trivial wire distance. Figure 20 and Figure 21 show a 3D model of this design idea.
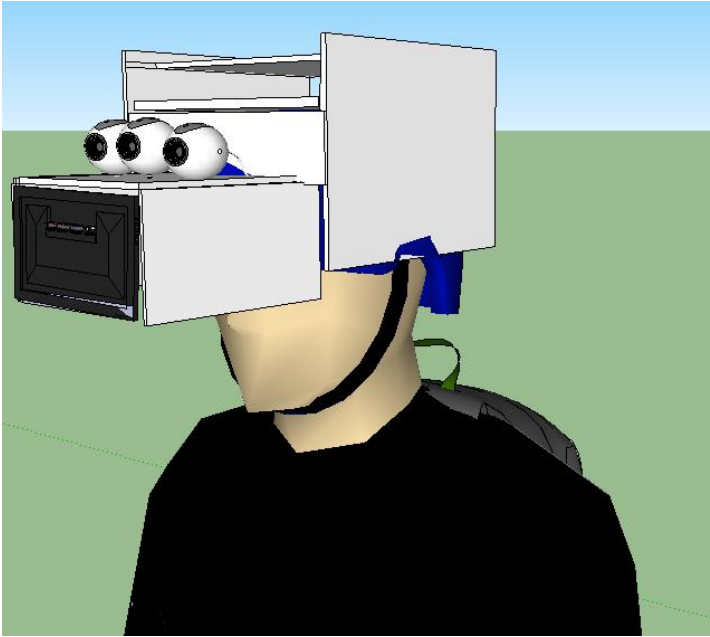
Figure 20: 3D model of Z-Goggles helmet (front view)
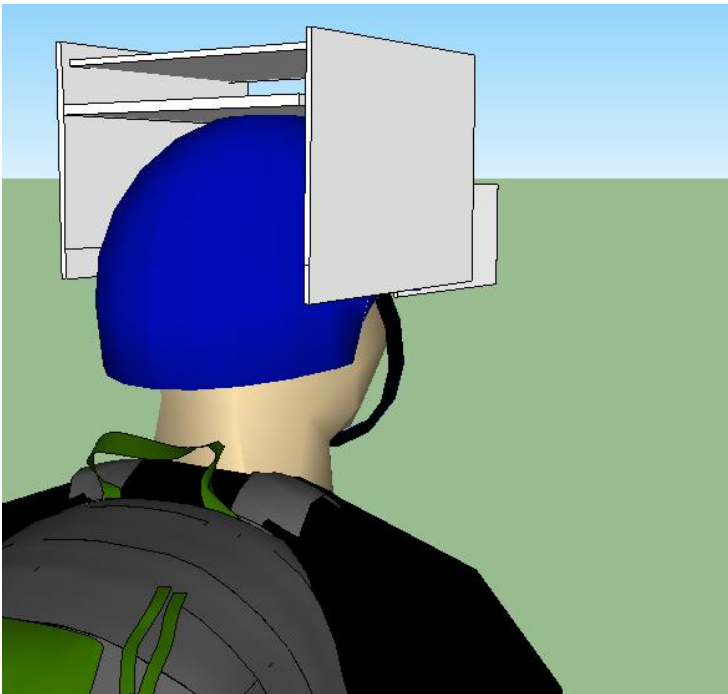
Figure 21: 3D model of Z-Goggles helmet (back view)

2. The second design also features a backpack-like vessel but would require this vessel to be more box-like and solid. With flat surfaces we could mount basically all of the electronics of the system in a box-like backpack and have only the camera system and display attached to the helmet. This would allow for a not flat-top helmet. Figure 22 and Figure 23 show a 3D model of this design idea.

Figure 22: 3D model of Z-Goggles backpack (side view)

Figure 23: 3D model of Z-Goggles backpack (back view)

2.7.3 Material Choices

Choosing the materials to support the Z-Goggle system is an important factor in the systems usability and safety. The helmet material must be strong enough to sport the weight o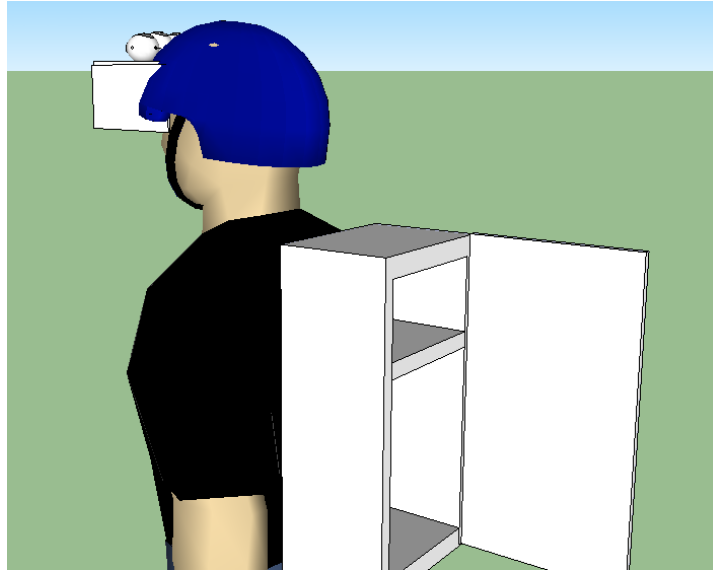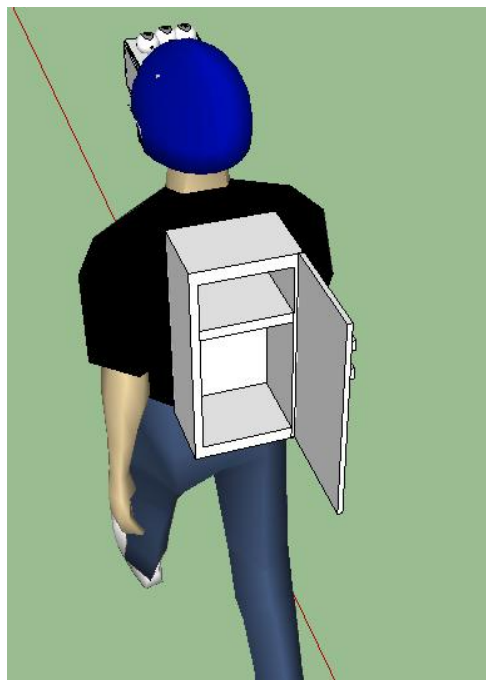f the attached display system hanging off of it, but at the same time the helmet also needs to be of a material that makes it easy to modify, allowing the design team to attach the display enclosure and maybe more to it. The backpack material is not as important as the helmet but is required to hold the power supply and may need to keep the power supply cool, or open to release heat. Price will be a major factor in deciding on materials for both the backpack and helmet. Table 12 describes the pros and cons of some average types of helmets that were considered.

Table 12: Helmet Choices

Helmet / Price	Pros	Cons
Bike Helmets / -20$	Cheap, lightweight, More airflow for less head heat	Odd shapes create difficulty in mounting camera, display, and FPGA systems. Looks unappealing
Multi-Sport (Skateboard) Helmet / - 13$	Cheap, Lightweight, Smooth surfaces for mounting, High tension nylon strap	Thin ABS High-Impact plastic may crack/shatter when drilled into
Motor Cycle (Half) Helmet /- 30$	Top quality ABS thermoplastic resin shell, Smooth surface for mounting, high quality,	More expensive

Any of these helmet options will most likely be able to work with the Z-goggle design, but some may take more modification than others. Choosing the type of Backpack and what the backpack itself will hold and will not hold is not yet been decided by the design team. While we may build our own backpack as a box that will hold electronics and power supply with straps attached, we may also use a pre-made backpack and simply place a box inside.

Chapter 3 – Design and Prototyping

After researching all the possible techniques, methods, architectures, integrated circuits, parts, and development tools, the system must be designed. Some sub-systems may have multiple choices deemed adequate, and therefore will have designs laid out for these choices. Design sections will assume logic acquired in the research section and will convey how the actual sub-systems we be used, built, programmed, laid out, and so on. This section will start with a summary of the final design chosen to be built and then go into each sub-system section explaining the designs for the choices of that system.

3.1 Design Summary

While research and design entailed a number of different options for components or even entire sub-system's of the Z-goggles, one selection must be made for the final design. The final design choices will be built for testing and prototyping. The Z-goggle system as a whole is shown in a simplistic manner in Figure 24.
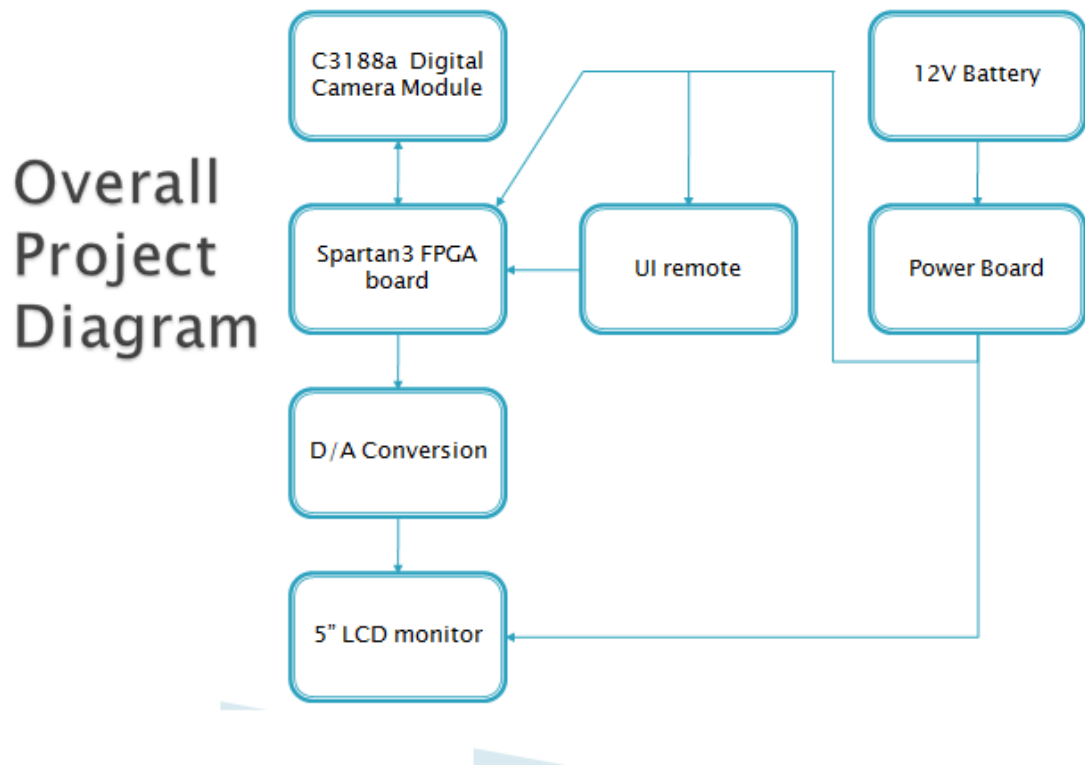

Figure 24: Z-goggle System Design

The digital camera used in the Z-Goggles design was chosen for its digital output and ease of interface with the FPGA Video Processor. Our group chose the C3188A Color Camera module based on the OV7620 image sensor from OmniVision. This camera has a pinout header that mates well with a modified cable to the FPGA, and also provides digital 8 or 16-bit RGB progressive video data output. This is the type of video we wished to use for processing and VGA output, so it was the ideal choice. Another reason this camera was chosen was its proven capability in microsatellite projects. This allowed some confidence in its ability to perform when compared to other possible camera options.

Communication with the camera consists of I2C control of the state of its data output and other commands. Due to difficulties with the camera's I2C function, our group decided to deal with the camera data as it is given to the FPGA. Essentially, the Video Processor takes in YUV digital data from the C3188A, and converts it into RGB. The rest of the information, including clock, horizontal and vertical sync signals, and data, are simply output from the camera into the FPGA.

The purpose of the FPGA is to act as a Video Processor by taking the data output from the camera and reformatting it into a properly synced series of images. These images can then be modified with the Z-Goggles image modification functions, and run into the VGA card. This may sound simple, but the main processes of image framing and memory-mapping the data output from the C3188A digital camera is a large portion of the design requirement for the entire Z-Goggles system. Also, each image modification function requires a module be designed to enact it.

A Digilent Spartan-3 FPGA board was chosen as the Video Processor. The main reason this board was chosen was its low cost and its available SRAM. This allows the memory-mapped video setup necessary for our processing engine. Doing without this capability turned out to be unrealistic, especially considering the image modifications we wish to accomplish that would flip data relative to a constant position. This is impossible without memory to hold a single frame.

To begin with, we must frame the image data coming from the C3188A digital camera. Once properly framed, the data is stored in memory. If necessary to output properly using the VGA card, we then convert the YUV data into RGB format. Next, the data is modified according to the user's preference made on the remote. Finally, the data is output to the VGA card using the VGA output controller. The image from the camera comes in along two clock pulses in YU and YV 16-bit words on a 27 MHz bus. The onboard system uses a 25 MHz clock for video output synchronization. This means that the system cannot operate synchronously and needs some form of buffering to support error-free video throughput. Because of the mismatched clocks, ideally some form of internal dual-ported RAM would be the most simple and reliable method of buffering. The board however contains 2 SRAM chips both holding 256k 16-bit words, with a shared address bus. Most boards with dual-ported RAM were too expensive for us to consider.

Image input data is only latched in within the camera's active region, denoted by a low signal on VSYNC and a high signal on HSYNC, with clock pulses marking valid pixel

input within each line. Internal state variables track these 3 signals to generate the input pixel coordinates. Each VSYNC high signal resets all the values, each HSYNC transition from high to low marks the next input line, and each clock pulse during HSYNC high denotes a half pixel of input. With these internal states, tracking can be accomplished to frame video input into memory in an asynchronous manner. Figure 25 shows the timing signals and system states/behavior.
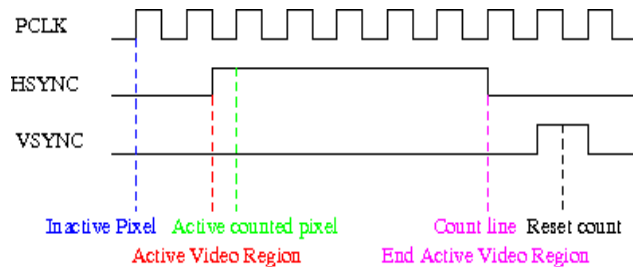

Figure 25: Timing signals and system behavior during the image framing process

To map the input and output data into RAM with the lowest possible latency, no conversion is done and the x and y coordinates are used directly for the address space. The x coordinate uses the lowest 9 bits to fit all the pixels of a given line into memory. This is enough bits to cover the entire 480 possible locations, since 2^9 equals 512. The y coordinate uses the lowest 9 bits as the upper 9 bits in the memory address. The MSB of the y coordinate controls the chip enable (CE) lines of the two memory chips to alternate them for the higher and lower parts of the frame. This is the solution to the constrained memory resources we mentioned earlier.

Because the input and output clocks are not synchronized in any way, including phase adjustment, characteristic diagonal banding will occur along an error region. In this region, contention for the address and memory busses will produce incorrect data on the output side of the system. To reduce this contention, a memory controller must be used which accepts input and output requests and buffers them accordingly.

To reduce the logic involved in the design phase, we favored reads over writes. Although writes occur more frequently, the time taken to execute the write is less important than time taken to execute a read. A delayed read operation will result in some errant pixel data. This type of failure will happen at regular intervals forming observable patterns in the output video even if the distortions themselves are minimal. A delayed write operation can only fail in the event that the delay takes longer than the frame takes to write. In most cases it is likely that the write operation will happen successfully before the next write operation. As the phase of the clocks grow apart, it is possible that a write may not be successful because of a neighboring read request very nearly follows the end of a failed write request. In our current design, it is unlikely that more than 1 error will accumulate, and if consecutive errors occur they will be more dispersed and thus less noticeable within the video window.

The current design is composed of the memory interface, internal buffers, and controller logic. The logic accepts read and write requests and on their positive edge. If the current

request is a write request and there is a read or write in progress it will be buffered internally until those requests have been processed. The memory device I/O latency is 10ns, and the clocks driving the read and write requests are spaced by about 40ns, so the chance of having an error persist for more than 1 read/write cycle is low. When there is no active request, the buffered write request will be pushed as a normal write request in the system. If there is a read request during an existing write request, the current write will be aborted and buffered. If there is a write request during a read request it will be buffered until the read request finishes. If a write request occurs during a write, it will be buffered. Reads can only take place every 40ns; and, they take precedence over everything. However, if a read request occurs during a read, something is severely broken. This case is not accounted for because the rest of the system will very obviously be malfunctioning. This memory controller is sufficient to remove contention for the address and data lines and remove the entire class of errors related to those shared busses.

There are many different color standards that may be considered and it's not precisely clear which color map the camera uses. The documentation doesn't lead to a direct answer, other than the fact that it is known to be a form of YUV data. The standard we were using specified three equations for converting YUV format data to RGB format data. They are

$$R = Y + 1.398 * V \qquad (1)$$
$$\text{and}$$
$$B = Y + 2.032 * U \qquad (2)$$
$$\text{and}$$
$$G = Y - 0.395 * U - 0.561 * V. \qquad (3)$$

Without floating point availability or the latency to even do large fixed-point multiplication, near-approximations are used. Our actual system implements these three equations instead:

$$R = Y + 1.5 * V \qquad (4)$$
$$\text{and}$$
$$B = Y + 2 * U \qquad (5)$$
$$\text{and}$$
$$G = Y - 0.5 * U - 0.5 * V. \qquad (6)$$

This change is made for the sake of time and the aforementioned accuracy issues. While this change in order of accuracy is dramatic, the color bias remaining the same across the image is more important than the color bias being very true to life. This removes implementation complexity and reduces latency. Also, quality of color is not one of the specifications we are required to meet at this time.

The simpler image modification functions directly operate on a single input pixel supplied from RAM. These functions include color inversion and palette adjustment. In the case of color inversion, the current red, green, and blue components of each pixel are subtracted from the maximum color value of that component (64 for green, 32 for blue and red, given a 5-6-5 16 bit RGB format). This results in an image with inverted color components, with white becoming black, etc. An example of this type of operation is shown in Figure 38. A palette shift is not a user modification, but it is used by the designers to shift colors to correct any perceived color glitches. Essentially, it is simply a quick color change to make the output video look better.

The second capability of the Z-Goggles is to flip the video upside down, or do a Vertical Flip. This capability is accomplished by saving the pixels in memory in a reverse manner. Instead of the 'top left' pixel being saved in the 'top left' spot in memory, it is saved in the 'bottom right'. This is only a little more complicated than the single pixel operations. An example of these types of operations is shown in Figure 35 and Figure 36.

The third capability of the Z-Goggles is to apply a simulated heat map to the color spectrum of the output. This can be done by mapping the luminance channel of the YUV camera output to the RGB values of the FPGA output. This example is shown in Figure 39.

The VGA output section of the Video Processor tracks the current and future pixels to be used and has a pixel buffer for the next processed pixel. It generates its own horizontal and vertical synchronization signals from timing offsets defined in the system through experimentation.

Many monitors will support various timing adjustments, and modern LCDs will attempt to do active region determination from color output sections, but seem to get it wrong more than right. Without being able to use any form of EDID queries for the specific monitors, each test monitor has its own hand-tuned set of constants. To assist in this tuning a special test pattern generator was developed which supplies 60x60 pixel blocks at each corner further divided into 20x20 pixel blocks. This gives us information about framing, edge behavior, color skew, and sub-pixel appearance. The next part looks at 8 colors in different sized bands to see other effects of color bleed and provide large vertical sections in which any skew issues will be more apparent and measurable. Below this are independent color gradients, and full color range gradients to look at the dynamic range of the monitor and resistor ladder in use. A small set of gray colors was selected in various sized bars for more timing and brightness analysis. This pattern can be seen in Figure 26.
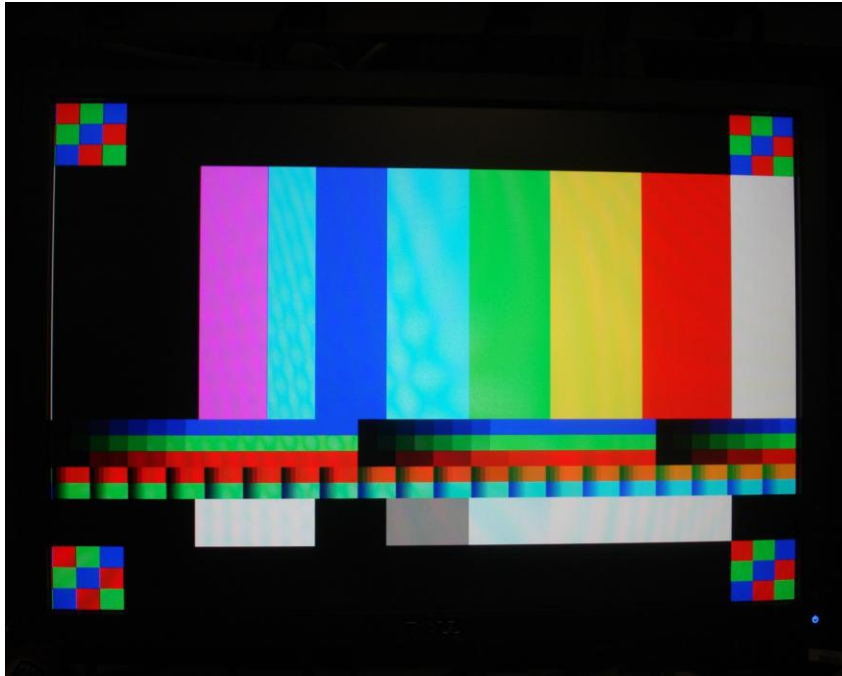
Figure 26: Test pattern for output timing estimation

The User Interface for the system is necessary to allow the user to control the image modification functions that the Video Processor provides. In this way, single or multiple functions may be selected to modify the video output shown on the Display screen in real time, as the user wears the Z-Goggles system. Shown in Figure 46 is a simple flow diagram of the User Interface.

To accomplish this, we decided to use a simple set of switches and the Atmel ATMEGA328 microcontroller. The Arduino Duemilanove board was used to develop functional code for the microcontroller. In this case, the microcontroller simply checks for the signal change from each switch, checks to see if the signals are a valid choice, and informs the FPGA of the change. There is no set switch position, so it is a direct on or off command, depending on what state the function is currently under. In addition to the switches, there is also an LED that is included to inform the user if an invalid combination of image modification functions is chosen. This UI circuit will be completed on a PCB; Figure 45 is the schematic that the PCB design is based upon.

One disadvantage to the chosen FPGA platform is its 8-color VGA output, which is insufficient to output 8 or 16-bit video. This is the reason a separate VGA card is required; the C3188A camera data output has much higher color depth than the Spartan-3 board can inherently handle. To solve this problem our group resolved to create a VGA output card that would handle the 8 or 16 bit color output data. To solve our output problems we first thought to use a digital to analog converter. After having issues with its output as well as overheating issues we decided to go with a 4 bit resistor ladder for each color to give us a 12 bit output. To give us a 75 ohm output resistance, we used a

combination of 75 ohm resistors in parallel with 150 ohm resistors. The design schematic is shown in Figure 42.

For the display unit we decided to go with a commercial off the shelf (COTS) LCD screen, the Accelevision LCDP5VGA. The screen has a pixel resolution of 640x480 which matches the ideal pixel resolution output that was specified during the design phase. The decision to use this display came after we realized that output to a computer monitor was more directly achievable with our FPGA than output in a composite format. In essence, this screen is a small computer monitor. The screen came with a DB-15 VGA connection so nothing more was needed to get it to function properly other than give it proper VGA signals. Also, this display unit was chosen because it could be run directly from the 12 Volt battery power source without the use of DC-to-DC converters. As shown in Figure 27, the backlight on the screen is acceptably bright when it is being run off directly from the battery at roughly 12V. It has a current draw of 300 mA, which is quite a bit less than we had originally assumed a display would draw. This should allow much longer system use-time before recharging the battery.

Figure 27: Accelevision LCDP5VGA Display Unit

To power the entire system, as well as create a portable system, we must use a battery-powered power supply. Our chosen design consists of one voltage regulator that converts the 12VDC power source into a 5V source. The 5V source is for the FPGA Video Processor and the UI microcontroller. The FPGA has an onboard 5V supply that will run the C3188A digital camera. The Display unit runs from the 12V battery directly. The general power supply design is shown in Figure 49. The 5V regulator we have chosen is the LM323K. It is a simple 5V regulator, which is easy to use and tough to break. The power source we have chosen is a PS1270 battery, which is a 12V output, 7Ah battery. The maximum current that the LM323K regulator can output is 3A. Our system is well under this mark, with expected draw under 1.5A. At this level, the battery will operate for over 4 hours without recharge. Our goal for system battery life is 2 hours, so the battery capacity is double what we need. The battery is easily left in a charger until use, due to the fact that it is a Lead Acid battery which prefers to be left under a maintenance charge.

The overall system is held in two physical places on the user's body. The most complex aspect of distribution is the helmet, where the FPGA Video Processor, camera, display, UI controller, and most of the power supply is located. A backpack is included, which holds only the battery. This is done to isolate the battery from other aspects of the system for safety reasons. The connections between these two parts of the system are the power lines that connect the battery to the rest of the power supply.

The helmet setup consists of a thin sheet of acrylic material attached to an easily adjustable construction helmet. The system On/Off switch is on the power supply board, which is located on the acrylic sheet. The FPGA Video Processor is located on the helmet above the Display. The UI controller is attached via cable to the FPGA in such a way as to allow access to the switches for user input. The display LCD is hung out in front of the user's face with metal bars attached to the side of the helmet. On top of the display fixture, the camera is mounted to allow clear view in front of the user. The helmet is weighted to balance the torque these components exert on the helmet. The backpack setup is simply the battery placed in a backpack. It is covered for safety and electrical tape is used to cover the power wire connection to the battery.

3.2 Optical System

The entire Z-Goggles system is based on the manipulation of input video data provided by the Optical System. The chosen camera components are three similar cameras, allowing us to use the data output from each as interchangeable, parallel inputs to our Video Processor (FPGA). One camera is unchanged; it gathers visual color content, filters out the rest, and outputs RGB or YUV data to the FPGA. The remaining two are filtered with a suitable optical filter, which allows us to see only IR or UV content. During research, two options were determined for exploration during the Design phase. As such, each camera option requires a preliminary design.

In the first Optical System setup, TCM8230MD cameras are used and they are connected to the FPGA Video Processor. They are tiny, so they will require a breakout board, but essentially they are directly connected. No interface device is used, as the camera outputs RGB or YUV directly from its data pins. As shown in Figure F10b, this design is very simple on its face. However, making this camera module operate requires more work than the second option, as you will notice the power circuitry in the block diagram. Because of the extra difficulty being added with this circuit it is being explored as a backup option. Also the lack of solid documentation to go with this camera module makes reverse engineering it particularly time consuming, which we cannot afford with the short timeline being impressed with this project. We hope with enough preliminary testing we can make this a more viable option for the future of the project.

In the second Optical System setup, CM-26N/P camera modules are connected to a video decoder device. We are going to be using the AL251A as a decoding module which will be outputting 8 bit or 16 bit YUV. The second stage will be used to turn the interlaced video output from the camera into a progressive scanned video. We need this conversion because we have our imaging processing software designed to work only with

progressive scanned data as well as our VGA output controller designed for progressive scanned. The output from the AL251 outputs 422 YUV or 444 RGB. Both of these outputs are acceptable. As compared to the first option, extra interface devices are required, but very little extra power control circuitry. Figure F11b shows the block diagram of the second option. Both camera setup options output RGB or YUV data, and within each option, all of the cameras are the same make and model. The end goal of the Optical System is to output proper video data, and these designs should accomplish that.

These original plans were not feasible in the final project setup, so we went with the C3188A digital camera. This camera allowed direct interfacing with the FPGA Video Processor, which was much more efficient for design work than the previously considered options. This interface is described more thoroughly in section 3.2.2.

3.2.2 Optical System Interface

The C3188a digital camera chosen for the final design directly interfaces with the Spartan3 FPGA board. It has 32 pins, and 26 of them are used on the FPGA. Figure 28 depicts the interface between the FPGA and camera. Since there were issues with the I2C control of the camera, everything is done in the Video Processor. The system takes in whatever the camera gives us, which in this case is an interlaced YUV data stream.
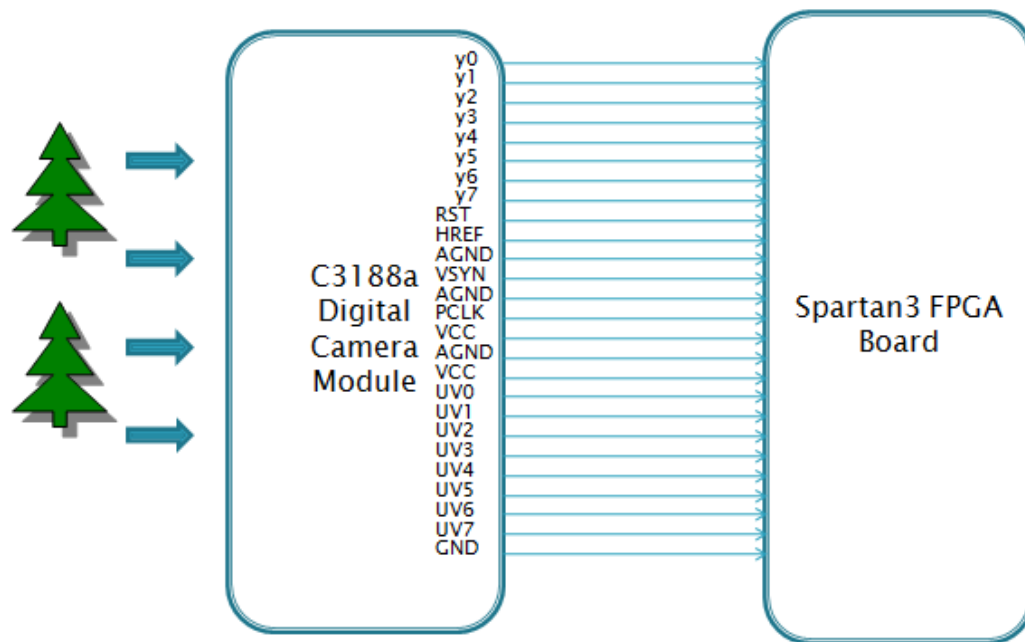

Figure 28: C3188a Digital Camera FPGA interface

3.3 Video Processor

3.3.1 Randomization Technique

Randomization- Various functions and subsystems will require some kind of random number input. For the sake of our use, it would be useful to have a parallel readable 8-bit register of random data. We don't need any kind of cryptographic-quality randomness, or even very good statistical randomness, so a simple 8-bit linear-feedback shift register will be sufficient for our uses. Taps will be placed at the 8, 6, 5, and 4 positions of the register because these taps positions will give us a run cycle of 255 before repetition occurs. It can be implemented with an 8-bit shift register and 3 XOR gates. It will supply 8 bits of pseudorandom data at any time and the data will change every clock cycle. This random number generator is shown in Figure 29 below.
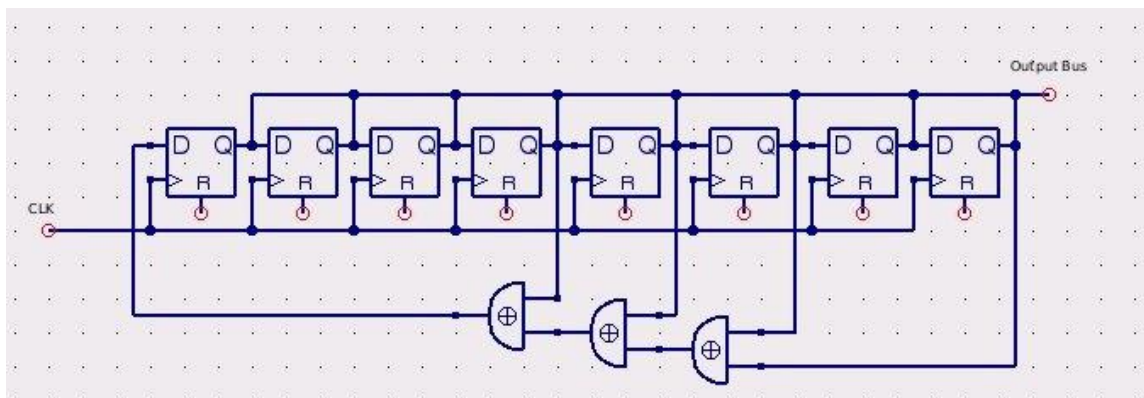

Figure 29: Random number generator

3.3.2 Video Storage

The video storage system is used to hold temporary video data that must be processed before it is pushed into the output frame. The video will be stored in 3 lines of distributed RAM on the board using the distributed SRAM between the LUT elements. It could be stored using a series of flip-flops but that would dramatically increase the gate count required. Using flip-flops has a decided advantage of taking a single clock cycle to update the most recent lines in parallel and simplify the memory management, but is too costly in gate count to consider.

To solve this problem, we'll be using a rolling buffer where each line is updated in turn and the addresses for lookup and writing are modified as they come in. The addresses for writing will be modified to move to the appropriate line, and the addresses for reading will require an additional 2 bits to specify which line is to be selected.

The first part of the memory system to modify addresses uses a state machine and input multiplexer to manage the addresses. There are 3 address storage cells, each 2 bits wide that store the current rolling address state. Each state transition follows the same 0, 2, 1, 0

pattern, making each of the cells independent. Implementing this transition is relatively simple. If the current state Q is given to be a 2-bit number, $Q1* = Q1'Q0'$, $Q0* = Q1$. These 3 values are then fed into a 2 multiplexers, which are switched by the same input. When a read occurs, the line address is the selector input for these multiplexers and the correct line address for the video line is generated. When a write occurs, the selector input is set to 2 because we're writing to the most recent line of data. The RAM will be implemented as dual-port RAM, making an advanced memory controller unnecessary. The design schematic for the video storage system is shown below in Figure 30.
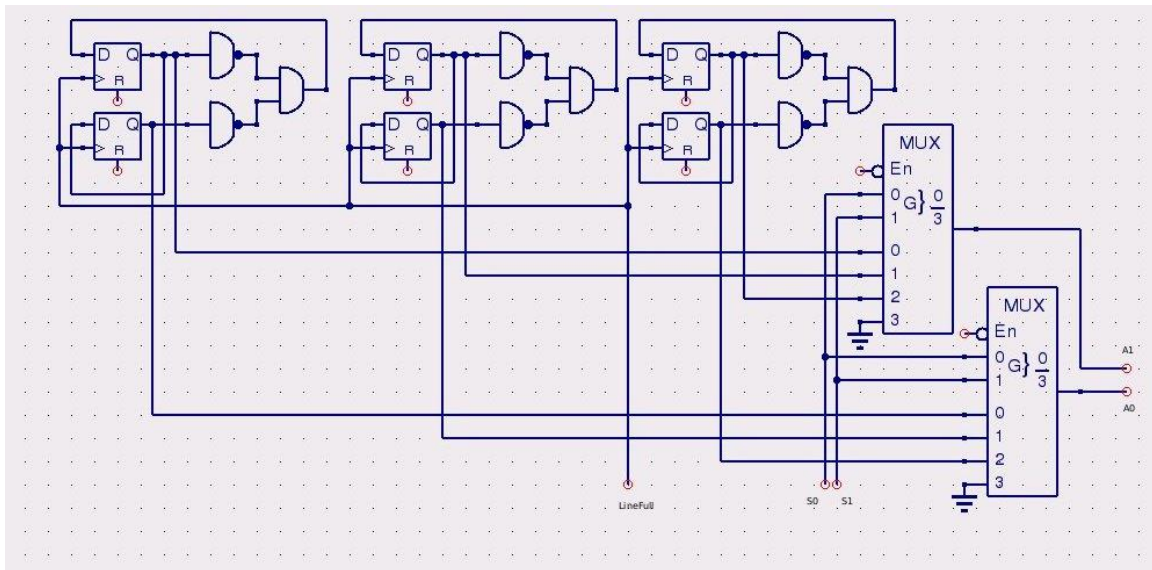
Figure 30: Video storage system

3.3.3 Video Display

The video display system simply reads pixel data in from RAM and outputs it to the board's VGA port. A VGA controller module will generate the Hsync, Vsync, blanking, and color channel outputs from a given clock input and color inputs. Part of the module will interface with RAM to gather the correct pixel data for the output stream. Since the S3BOARD only supports 3-bit color, we had to create our own resistor ladder system to support 12-bit output. This is covered later.

The VGA system requires a 25MHz input clock to do all of the synchronization timing. It divides the internal 50MHz clock on the Spartan3 board and converts the signal into horizontal and vertical sync pulses after adjusting for front and back porch timing. The color input is simply passed through during time where the video signal is active (meaning not in a blanking region on either porch).

3.3.4 Processor Architecture

The heart of the Processor is shown below in Figure 31. This is the overall schematic that will be used within the FPGA Video Processor to perform all view modification functions and other necessary processes. Each part will be detailed later in this section.
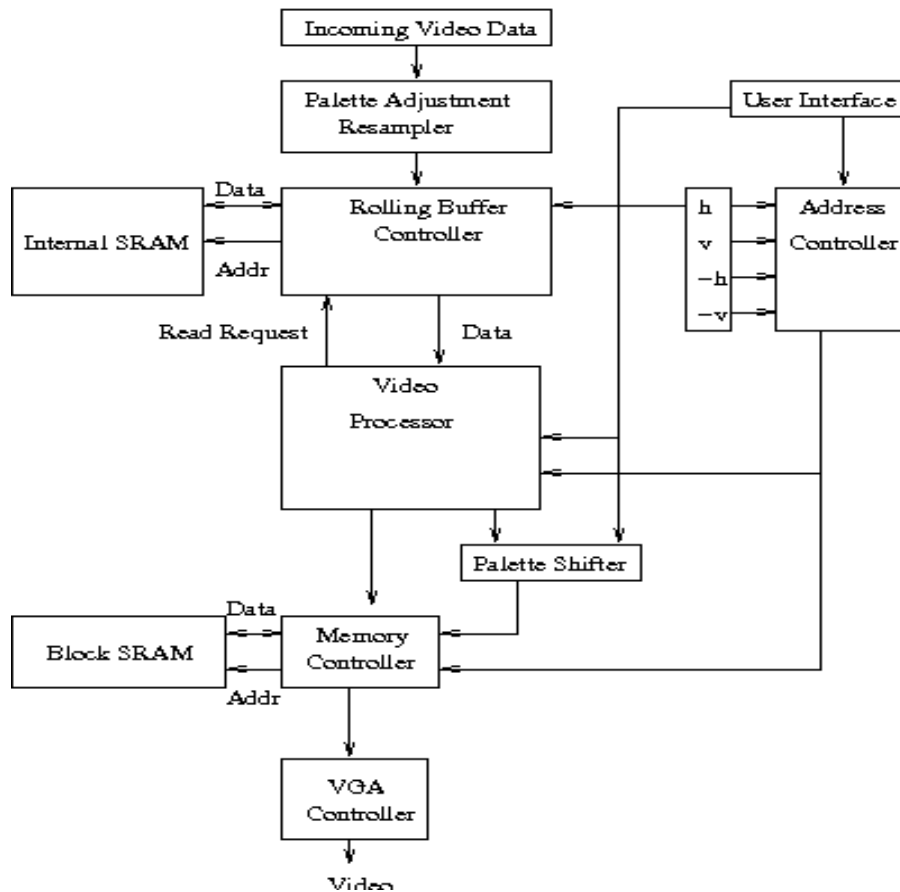

Figure 31: Processor Architecture

For video quality reasons, regardless of the input, having a correctable color table is valuable, so all inputs must be converted into their actual color or grayscale representations. The data coming in is interlaced YUV data. For this reason, pixels must be down-converted into an RGB, 8-bit, 3:3:2 format so the video processor can work with them correctly. This job is fulfilled by the Palette Adjustment Resampler. This segment also contains the logic for weighted adjustment of camera input data, allowing the input stream to the video processor to be modified. In this way something like an IR heat map can be overlaid onto the color spectrum. To accomplish this, two's complement fixed-point multiplications will be used, requiring multipliers and shifters. This corrected video output is pushed to the rolling buffer memory controller which interfaces with onboard SRAM in a dual-ported RAM configuration, because speed for reads and writes in this section is very important, and managing any access conflicts would be difficult. All writes are pushed to the most current line, at the location specified by the horizontal

counter. All reads from the controller are translated so the video processor can get access to all 3 lines of video data.

To support different video flipping modes, multiple horizontal and vertical counters must be used. The normal-scan horizontal counter is used by the rolling buffer controller as well as the address controller. The address controller supplies the video processor with a current pixel address for homogeneous coordinate manipulation system. The user interface block is used to select which set of counters to use to calculate a memory address for writing the current pixel data. This allows us to support video flipping modes in parallel with other pixel modification modes and remove complexity from the video processor. The video processor block will not be described here, but will be described in detail later.

The video processor outputs the current data to a palette shifter again to do any post-processing color modifications such as rebalancing, palette swapping, or color manipulation for any extra vision effects. The color manipulation could be used to arbitrarily increase the intensity of specific color ranges regardless of input intensity for reading messages written in certain colors that would normally be difficult to distinguish for the human eye (cases where there is a large difference in chrominance, but not in luminance). Such manipulations could also be used to translate chrominance differences into luminance differences for color-blind people. It gives us a lot of flexibility in overall video effect output for different users.

The data from the palette shifter is passed to a separate memory controller which communicates with the S3BOARD's SRAM modules. It makes a write request to the memory controller, which pulls the current address from the address controller and data from the processor's output, and then deals with any read/write conflicts from the video output system. The address controller's output is determined by the user interface allowing the video to be written in any orientation. At this point, the transformed data is presented to the screen for the user to view.

The palette shifter is composed of three color inputs, three weighting and shifting modules, an inverter, and a mixing system to swap color channels. The user interface is used by the control module to select a correct weighting and shifting amount from the look up table. Implementation of the palette shifter module is done with 3 multipliers and 3 multiplexers. The palette shifter is shown below in Figure 32.
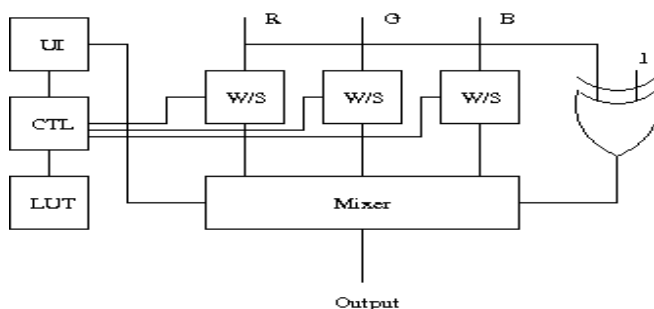

Figure 32: Palette shifter

3.3.5 Processing modes

For the image enhancement system we want to expose the user to various modes that would give the effect of a different view into everyday life. To accomplish this we wish to alter the video feed that we are receiving to the FPGA in such ways as flipping the image horizontally or vertically or both, applying an edge detection algorithm to the image and applying a photo-negative color to the image. This will be done the use of different pathways to correct color and apply convolution as well as a buffer, all to give the desired changes to the visual spectrum to the user.

Palette Shifter Pathway- The palette shifter pathway is used for any given mode to transform the output color map before it goes into the output RAM for the video output system. It consists of several adders, multipliers, and shifters, which allows an arbitrary set of color transforms depending upon the input parameters. There are 12 on-board multipliers of which 3 are available for the palette shifting system, allowing any arbitrary scaling transform if a fixed-point system is used for the channels. Given that the multipliers are 18 bits wide and the 3 channels of data are only 8 bits wide, using a series of shifter could reduce the multiplier requirement from 3 to 1 or 2 multipliers.

Convolution Pathway- The convolution pathway allows nearest-neighbor matrix multiplies and reduces to generate pixel data. It consists of 9 multipliers to do a full 3x3 multiply in parallel. Some of the multiplications for different fields may be done on the same multiplier by shifting them over so that their results cannot interfere with each other. The convolution pathway has a series of shifters to accomplish this goal. After the multipliers is a large 9-way summing junction to output a final pixel value for each channel. The convolution pathway may only support 1 or 2 cameras at a time and introduces the most latency; some other video modes may not be available simultaneously.

Temporary Input Buffer- To reduce throughput latency for blur and edge detection, a storage matrix composed of 3 lines of 3 shift register elements each will be used. On every pixel clock input, the matrix will be updated by shifting all the pixel rows over and grabbing 3 new pixels from memory. The extreme edges of the screen will experience wrap-around distortion, but the memory throughput will triple. This configuration also simplifies the design for the rest of the video processor sections. Each clock cycle a new data cell will be loaded from the temporary video memory through the rolling buffer controller. The only thing that must be given is the line requested; the rolling buffer controller will grab the newest data and output it. The schematic below in Figure 33 shows how this would be implemented.
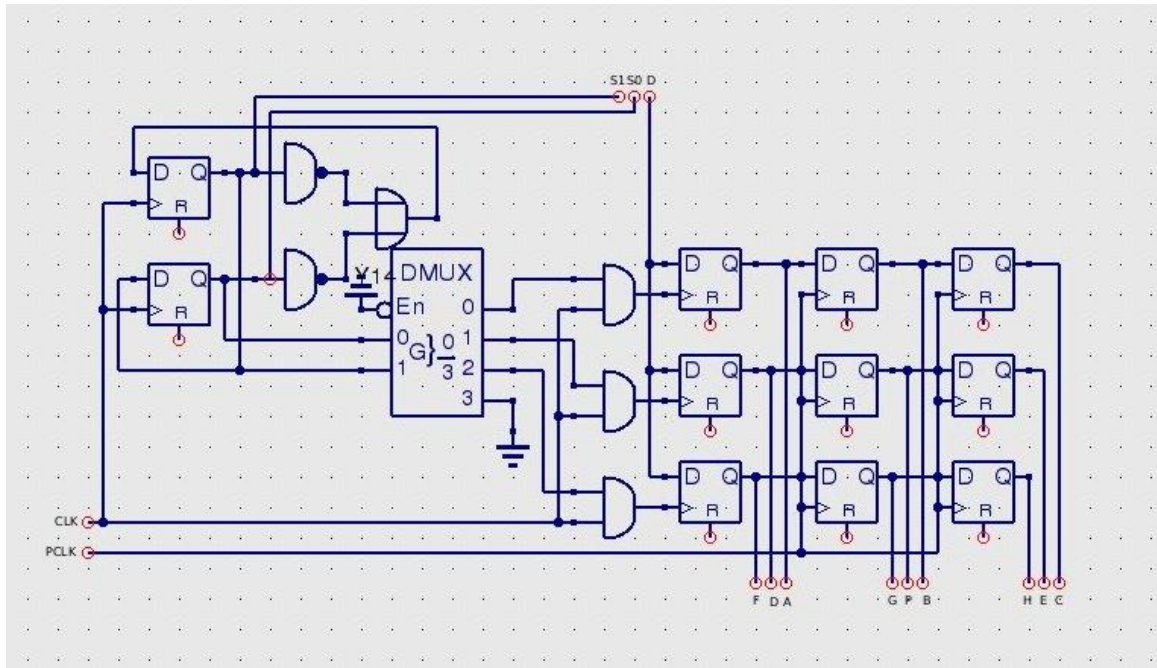
Figure 33: Rolling Buffer Controller

In this schematic, CLK and PCLK are inputs for the clock and pixel clock signals routed on the board. In the top are the request codes from the temporary system giving the line number to select and the data output line. The line select system is mirrored in this module to scan through different lines and update each respectively by changing which flip flop is enabled by the clock. This system drops the transfer requirement of data by keeping old data in the matrix and updating the matrix as new pixel data arrives. The data output lines are labelled in correspondence to the input label in the blur and edge detect modules that use this functionality.

In the images below are the examples of the different functions we plan to offer the user. Some of the different functions will be available across different modes such as a flip mode matched with the photo-negative colors, This will not be available for the blurring effect. Each mode will work with the different types of vision as well as well as the spectrum shift function.

Regular Vision- Regular vision is the default testing mode of output. The RGB camera input is pushed straight through into RAM for reading by the video output system. This mode allows us to test out frame-rate, focus, and color adjustments on the video system before trying to do anything complicated. An example of this viewing mode is shown in Figure 34.

Figure 34: Regular Vision with RGB input

Upside-Down- To flip vision upside down, the memory write requests must be altered from their normal state. This module keeps a separate, but parallel pixel location counter which counts in reverse for the video line count, but forward for the video line location. It will take a given image without any processing and modify its output location in RAM so that the picture on the video screen appears upside-down from a normal perspective. This mode can be used in conjunction with other camera modes. An example of this viewing mode is shown in Figure 35.

Figure 35: Upside down image with RGB input

Left-Right switch- To flip vision horizontally, the memory write requests will be altered from their normal state. The switch module keeps a separate, parallel pixel location counter which counts in reverse for video line position, but forward for video line count. This flips the video as if the person were looking at reality through a mirror. This mode can be used in conjunction with other camera modes. An example of this viewing mode is shown in Figure 36.

Figure 36: Horizontal flip with RGB input

Total flip- This mode uses the modified memory addresses from both the upside-down and left-right switch modules to write the data in reverse order through RAM. The output will appear as if it had been flipped along the y=x line for the video output system. This mode can be used in conjunction with other camera modes. An example of this viewing mode is shown in Figure 37.

Figure 37: Vertical and Horizontal flip with RGB input

Inversion- The inversion view works similarly to the palette swap pathway. It can be combined with other modes of operation. The inversion system inverts all of the incoming bits, creating a view that looks similar to a photo negative. This mode can be used in conjunction with other camera modes. An example of this mode is shown in Figure 38.

Figure 38: Inverted colors with RGB input

Heat Map- The heat map mode takes the YUV luminance data and places it in the RGB output memory to be output directly. This results in a video output that looks like a heat scanning device. An example of this mode can be seen in Figure 39.

Figure 39: Heat Map example

3.4 Display

For the display unit we decided to go with a commercial off the shelf (COTS) LCD screen, the Accelevision LCDP5VGA. The screen has a pixel resolution of 640x480 which matches the ideal pixel resolution output that was specified during the design phase. The decision to use this display came after we realized that output to a computer monitor was more directly achievable with our FPGA than output in a composite format. In essence, this screen is a small computer monitor. The screen came with a DB-15 VGA connection so nothing more was needed to get it to function properly other than give it proper VGA signals.

Also, this display unit was chosen because it could be run directly from the 12 Volt battery power source without the use of DC-to-DC converters. As shown in Fig. 12, the backlight on the screen is acceptably bright when it is being run off directly from the battery at roughly 12V. It has a current draw of 300 mA, which is quite a bit less than we had originally assumed a display would draw. This should allow much longer system use-time before recharging the battery.

3.4.1 Display Enclosure

The display system is much too large to put directly in-front of the face. The screen needs to be about 6" from the users face in order for the user to be able to see the entire screen without an enormous amount of stress being placed on the eye. The screen's back is covered to avoid any open wires from the circuit board, while the user's peripheral vision needs to be blocked to keep the user from seeing anything other than the video output experience desired. Figure 40 and Figure 41 show the enclosure around the LCD screen.
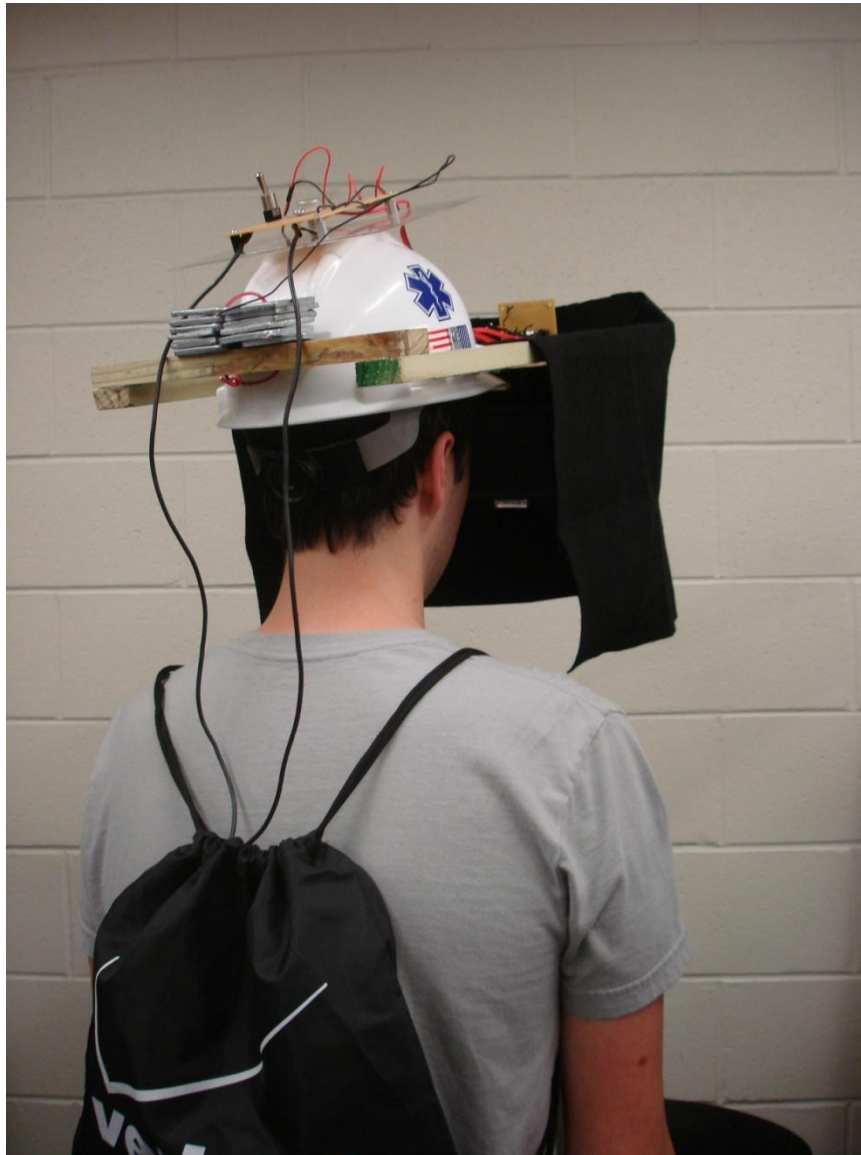

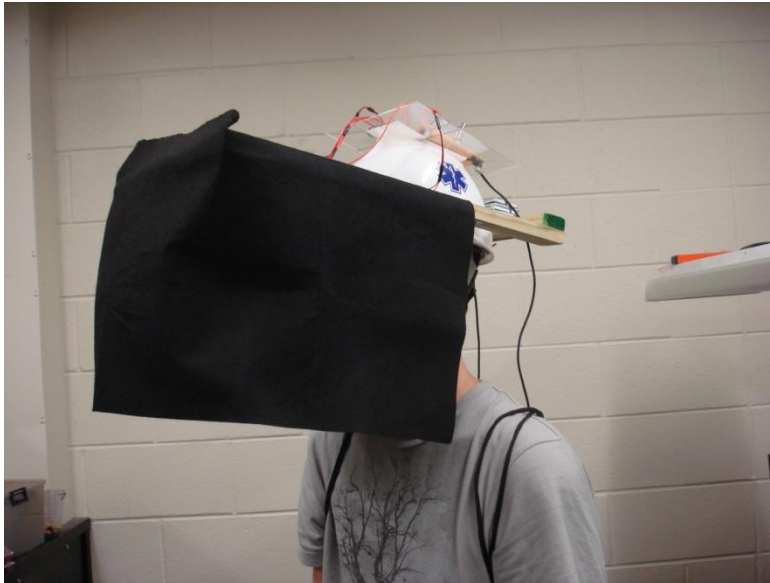*Figure 40: Display view inside helmet*

And an alternate view:

Figure 41: Display view outside helmet

3.4.2 Display Interface Design

The VGA display controller will be programmed on to the Spartan3 FPGA board. Using Verilog, we can break the VGA controller code into sections for synchronization, memory, and output.

To design the synchronization code we start with the clock. The Spartan3 board runs on a 50MHz clock, while a 640x480 resolution must be run on a 25MHz clock. To fix this issue without creating a separate 25MHz clock domain, we will create a 25MHz tick by setting a variable that switches between on and off on every clock cycle. Since the desired pixel rate is exactly half the clock rate, we can just send data when this variable (we will call pixel_tick) is set to one. After setting the pixel time we can work on the horizontal sync. After every 640 pixels are sent for a single line of active video, we must horizontally synchronize the signal back to the beginning of the next line. While the original timing was based on the time to reset the electron beam in a CRT monitor, the same timing is still used for LCD screens. We define the "Front Porch" as the time it takes to send 16 pixels, the "Back Porch" as the time it takes to send 48 pixels and the "Pulse width" as the time it takes to send 96 pixels between the front and back porch. So we need to set the output to the next horizontal line every 800(640+16+96+48) pixels. We can set the end of a horizontal line to be every 799 pixels, so we now when we need to move to the next line. Vertical synchronization is very similar to horizontal but occurs at the end of a frame. After a frame or 480 lines of pixels are sent, we must turn off active video and transfer another front porch, back porch, and pulse width. This vertical sync time is defined by the transfer time of 10 lines for the front porch, 29 lines for the back porch, and 2 lines for the pulse width. So again we can set a variable to show the end of vertical lines in a frame to 520(480 +10+ 29+2 -1). Now with this information we can

simply send data at the pixel clock rate, and properly send synchronization signals using simple counting registers and if statements [43][66]. These signals are then sent to the resistor ladder board, which holds the VGA output port.

For the design, the Spartan3 FPGA was used, due to memory mapping advantages. The Spartan3 board is capable of outputting only eight colors though it's on-board VGA port. To work around this issue the design team implemented a 3-teir R2-R resistor ladder. One four-bit resistor ladder was used for each color (Red, Green, Blue), which gave the design team 12 total bits of color to work with. On this 4096 color output option, three color adjusting knobs were also added, to allow for any color adjustment needed on the LCD screen output. Figure 42 shows the three R-2R ladders schematic, and Figure 43 shows the final result built.
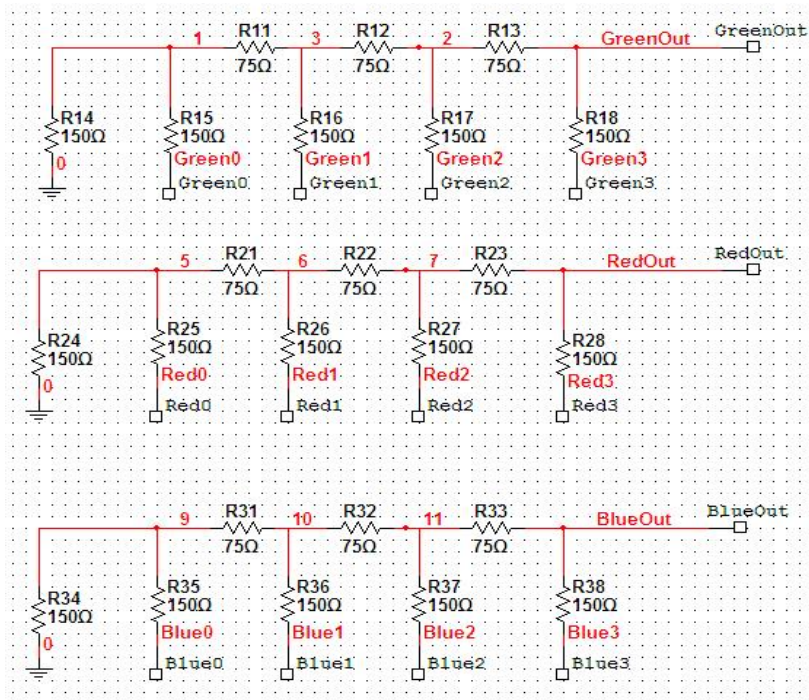

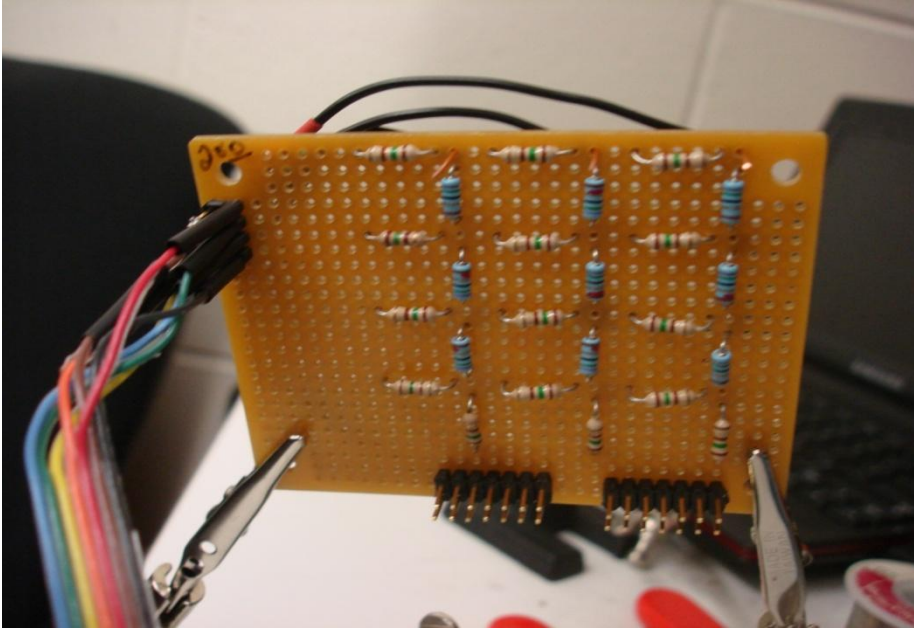*Figure 42: R-2R Resistor Ladder Schematic*

Figure 43: Three 4-bit R-2R ladders

3.5 User Interface

The user interface system is the bridge between the user and the rest of the Z-goggle system. It will let the user select cameras and functions.

3.5.1 User Interface Hardware Design

To design of the Z-Goggle User Interface the Arduino Duemilanove board will be used for development. First we need to describe how the design team intends to create the UI with the Arduino board. The Z-Goggle system is a complicated helmet system, and the user of the system will be limited in vision. To allow the user to interact with the system while wearing it, the design team will create a wired remote control. This remote holds the Arduino Duemilanove board, the seven switches for the individual functions, and a power switch to turn on/off the system.

A basic switch circuit needs to be set up for each switch/import pin on the microcontroller. Pins configured as input pins on the Atmel microcontroller are in a "high-impedance state". While this means that it takes very little current to move the input pin from one state to another, it also means that input pins with nothing connected to them, or with wires connected to them that are part of an open circuit(like a switch), will report seemingly random changes in pin state. The pins will pick up electrical noise from the environment and make the switch states unstable [67]. Pull-up resistors solve this problem by "pulling" the voltage of the non-voltage-source wire it is connected to towards its voltage source level when the switch is open. By connecting four switches to seven import pins in this manner and connecting seven output pins to the FPGA board,

we have set up a simple interface allowing the user to select a function and output the selection to the FPGA board. Figure 44 is the completed remote. Figure 45 shows the UI circuit design.
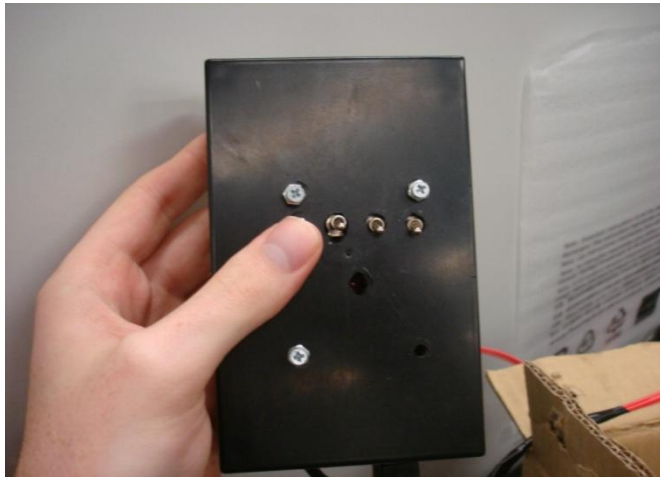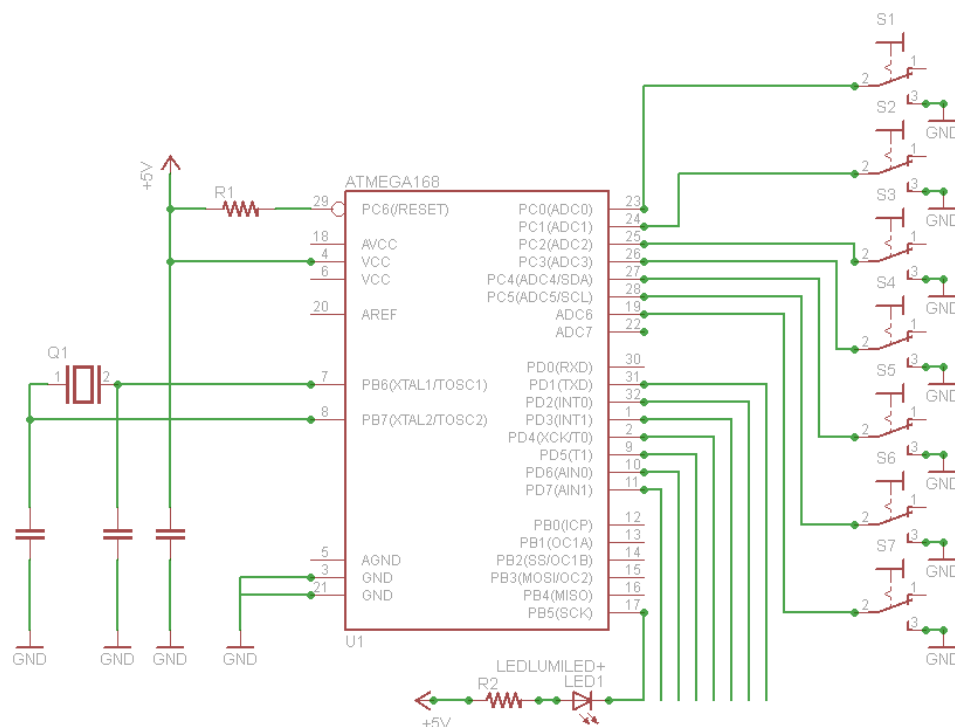

Figure 44: UI Controller

Figure 45: UI circuit schematic

3.5.2 User Interface Software Design

Now that have the hardware setup we need to program the microcontroller to take in and send out the switches input. We will be using the second algorithm choice outlined in the research section. This involves creating a database or table of 128 values. The code will first need to read in the four values from the four switches and then convert the four values into a seven bit binary number. Since we would then convert the binary number into a decimal number to make the table setup and table lookup more convenient, we will instead convert the four switch signals directly into a decimal number. To do this we will simply set a variable up and add in a specific decimal number for each signal checked. After we have checked every switch, we will then search the table for the decimal number (0-128) and return "1" if that choice is allowed or "0" if it is not. If not allowed the program should go back to the start of the code and wait for any switch values to change. If allowed the code must take the seven bit binary number and assign the corresponding output values. When checking the switches inputs in a loop waiting for a change, we must also use a sampling time in order to reduce the chance of detecting noise as an input change. This small time delay will be unnoticeable in system use but will help reduce the chance of error. For initial testing we will check all the switches every few milliseconds using the "delay()" function. The following pseudo code and comments describe the syntax of how the ATMega168 is programmed, but not the actual algorithm used. Lines that start with "***" are comments. Figure 46 explains the sequence of events that the microcontroller will be programmed to follow.
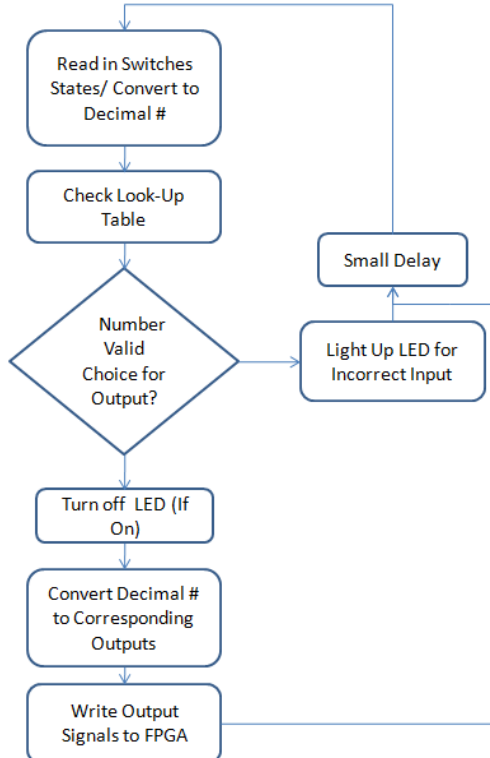

Figure 46: UI Flow Diagram

```
***first we want to declare all the pins we want to use and then ***set them as either inputs or outputs
byte switchCameraPin[3] = { 9 , 10 , 11 } ; //9,10,11 will be camera switches for INPUT
byte cameraOutputPin[3] ={ 16,17,18};// 16,17,18 will be camera pins for OUTPUT
byte switchFunctionPin[4] = { 12 , 13 , 14, 15 } ; //12,13,14,15 will be function switches for INPUT
byte functionOutputPin[4] = { 19 , 20 , 21, 22 } ; //19,20,21,22 will be function pins for OUTPUT
*** we set them as input/output pins using the pinMode() function ***like so…
*** setup is a function used just to declare all the variables we will be using
void setup() {
pinMode(switchCameraPin[i],INPUT);
pinMode(cameraOutputPin[i],OUTPUT);
*** [i] will be replaced with the particular camera pin you are ***setting, e.g(1,2,3)
*** after setting up all the pins including all not shown above, we ***start the main function
void loop(){
*** loop() is a function that will continuously run over and over ***again, checking the switches
***inside loop() we will use if-else statements to check the ***switches input wires for all the possible
combinations of ***cameras and switches, setting the output pins when the ***combination is valid, and
not changing when the switches are ***invalid. Using the digitalRead function like so:
if(digitalRead(switchCameraPin[0]==SWITCH_PRESSED){
}
*** digitalRead checks the pins voltage value and returns a 1 or 0 if the pin is High or Low.
                        *** using && (and) statements within if statements allow us to say
                        things like IF(Only camera one is selected) Then ***( do something)
```

While the design team estimates that any possible combination of the four functions is possible, the look-up table is still useful to stop multiple cameras from being chosen. Although the UI just takes in switch signals and sends them out, without checking anything, the look-up table still serves the purpose of control if not all 128 options are possible, and allows us to test combinations for future work.

3.6 Power Supply

Designing the power supply will be done in three stages. The first stage is data gathering in regards to the current draw and voltage requirements of each part. When a solid estimate of these requirements is determined, the actual design phase can begin. The design phase encompasses determining the number and type of voltage regulators needed to power the entire system, the schematics for these parts, and the overall layout of the system's power supply. Finally, a method for switching the system on or off will be determined, with several possibilities explored with regard to the chosen power supply setup.

3.6.1 Current Analysis

The first step in designing the power supply for the system is to determine what sort loads will be present to be supported. Information needed about the load consists of a voltage requirement and current draw that is expected. For the Z-Goggles system, the primary design focus includes one digital camera, one Spartan3 FPGA, 1 Display, 1 UI microcontroller. These will all be running off a single battery source. The two batteries

we found were Lead-Acid, 12 V, 5 and 7 Ah batteries. Depending on this current analysis, we will choose our battery for the design. Table 13 contains the load information. This includes the estimated typical or max current draw required. It should be noted that these draws are extremely important for later design of the power supply, as regulators have a maximum current draw they are capable of handling. Within the table, some estimation is done; some data sheets do not have complete power information. The ATmega168 microcontroller used for the UI has very low power consumption, but we are overestimating it by a great deal because the data sheet isn't clear on the maximum current draw. For each component, the data was gathered or estimated from the data sheets or reference manuals for the part.

Table 13: Current draw and voltage for each component of the Z-Goggles system

Component	Voltage required	Typical Current	Max Current
C3188a Camera	5 V	120 mA	-
Spartan3	5 V	440 mA (1A use Est.)	5.95 A
Display	12 V	-	300 mA (Est.)
UI microcontroller	5 V	-	50 mA
		Total Estimation	1.47 A

Given that several components are likely overestimated, a general estimate of 1.5A of total current needed seems fair. Clearly, the two batteries chosen can both run for 2 hours or more given this load. Due to the nature of Lead-Acid batteries requiring low discharge rates, the best choice is the highest A/h rating [68]. In this case, 7 Ah batteries should be capable of meeting our current needs with greater efficiency than 5 Ah batteries. We chose to use the PS1270 from PowerSonic, due to this fact [61].

3.6.2 Power Supply Design

As discussed in the Research section, our design software of choice is the WEBENCH Power Design tool from National Semiconductor. Specifically, we will be using their Power Architect capability for our multi-load system. This program allows the user to specify any number of sources and loads by their Voltage, Current requirement, and other characteristics. When finished, the program allows optimization and replacement of individual regulators with alternate choices. The schematic for the part is provided, and a bill of materials is provided as well [62]. Simulation of the performance of the setup is the last step. To be thorough in the design, a setup using National Semiconductors typical voltage regulators will be explored, in addition to a design using their power modules. These are also regulators, but they offer higher efficiency and thermal performance, at a higher cost [69]. In particular, the two designs will be compared by price, efficiency, and thermal results. Only the better design will be fully explained in this section, as that is the design the Z-Goggles will be using.

The designs compared differed only by one part, the main voltage regulator that will act as a 5V source for the FPGA and UI microcontroller. The other part was the same between each design, a 12V to 12V voltage regulator for the 3 cameras. Since this is the

same, and may not even be needed, the comparison is mostly based on the main 5V regulator. The first comparison is simple. The first design includes has a total Bill of Materials of $10.07, while the second design has a total Bill of Materials of $5.44. Obviously, the first design is double the cost of the second design, though the overall prices are low. This price is due to the fact that the first design uses a LMZ12003 power module, which is basically a more advanced version of the traditional voltage regulator. The second design features a LM22676MR-5.0 voltage regulator, so it is much cheaper. Our second metric for comparison is thermal performance, which is shown by a thermal simulation from WEBENCH. As shown in Figure 47, the difference is negligible, with a slight edge to Design 1 and the LMZ12003. Just to the right of center there is a blue-yellow region in the Design 2 simulation, which is a good bit hotter than any part of Design 1 [75][74].

(1)   (2)
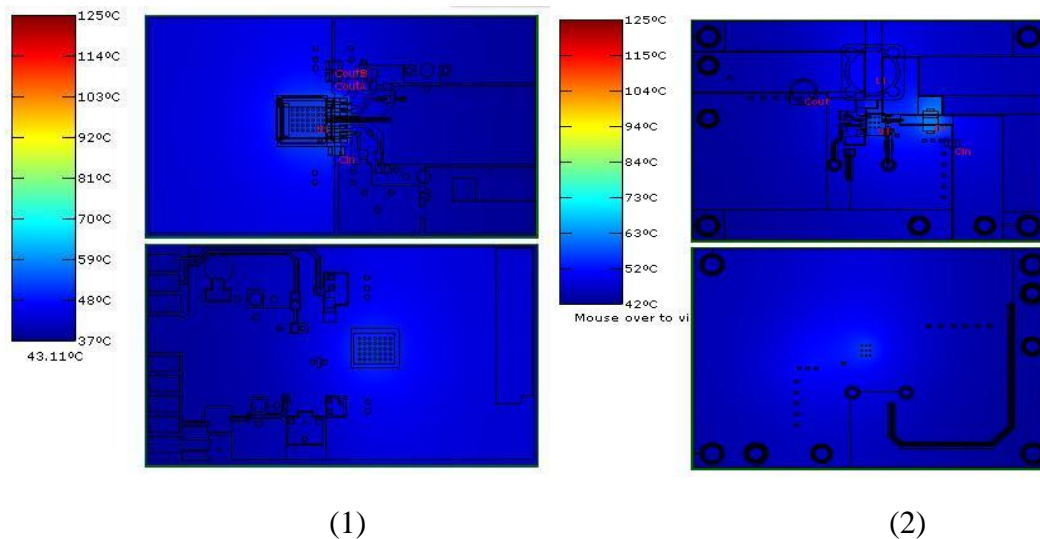
Figure 47: Thermal simulation results for (1) Design 1 and (2) Design 2

The final metric for comparison is efficiency. Design 1 boasts a total efficiency of 92.2%, compared to Design 2 with 89.3%. Similar enough, but the efficiency graphs for the two main parts, the LMZ12003 and LM22676MF-5.0 have different behaviors to changing output current. This situation is a real possibility, because the load current numbers used for the design are partly based on estimation. We must compare the two parts on their efficiency curve in respect to output current, shown in Figure 48. As can be observed, the efficiency drop-off is much sharper for Design 2. The parts are rated to 3A output current, but we shouldn't get to that point. However, if we approach anywhere near 3A, Design 2 will have an efficiency in the neighborhood of 86-87%, while Design 1 holds steady above 90%. When designing for the worst case scenario, Design 1 seems the superior choice. It seems to have little to no temperature issues, and has a steady, high efficiency. Due to the power supply parts being relatively cheap across the board, Design 1 with the LMZ12003 is our power supply design choice.

(1)   (2)
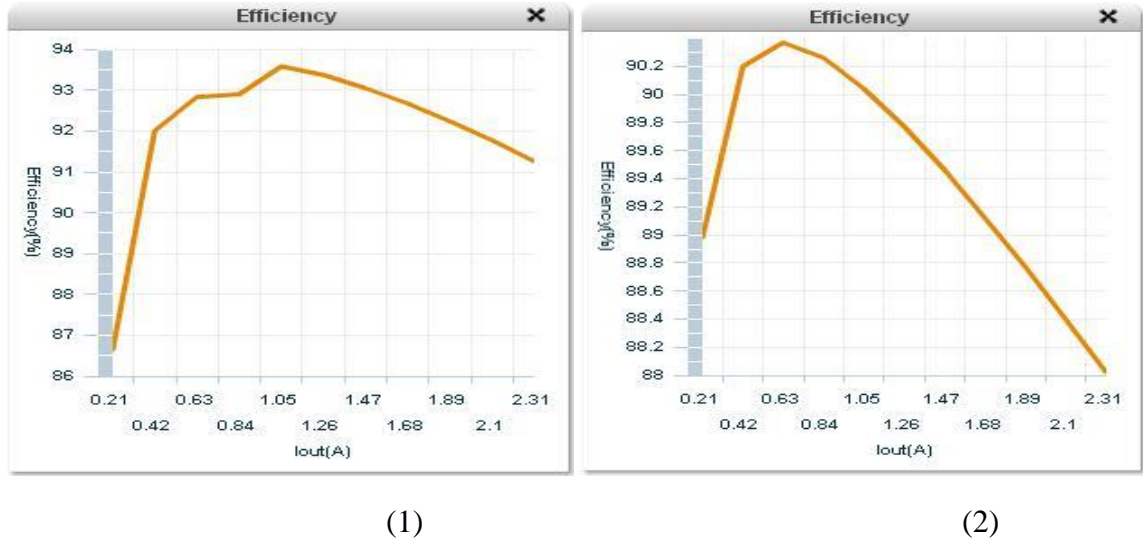
Figure 48: Efficiency vs. Output Current for (1) Design 1 and (2) Design 2

Our chosen design consists of one voltage regulator that converts the 12VDC power source into a 5V source. The 5V source is for the FPGA Video Processor and the UI microcontroller. The FPGA has an onboard 5V supply that will run the C3188A digital camera. The Display unit runs from the 12V battery directly. The general power supply design is shown in Figure 49.
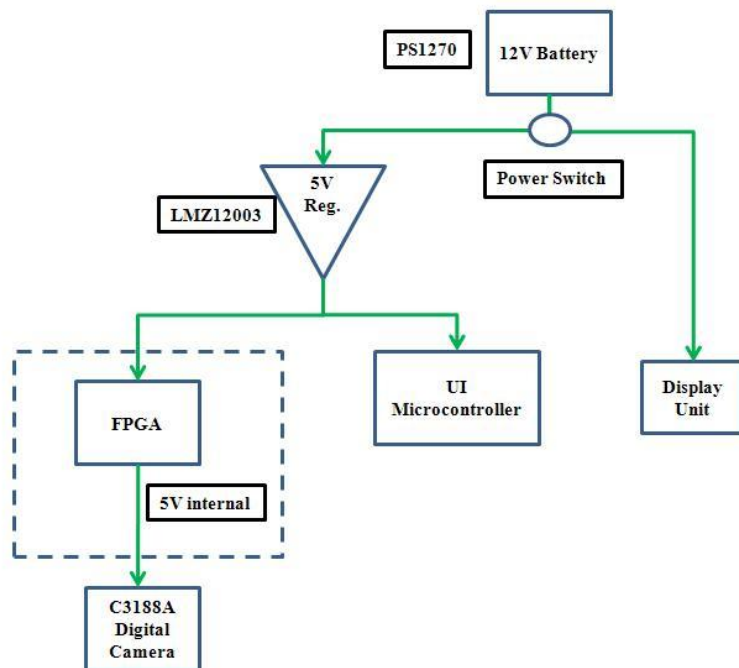
Figure 49: Overall Power Supply layout

As mentioned before, the 12 to 5 voltage regulator is the LMZ12003. WEBENCH supplies schematic diagrams that show the configuration of all external resistors and capacitors for each part. Shown in Figure 58 is the schematic for the LMZ12003 component. The essential results of the schematic are shown in Figure 50, with Vin = 12V, Vout = 5V, and Iout = 2.102A. To check that the proper output voltage is set, the data sheet calls for this equation:

$$Vout = 0.8 * \left(1 + \frac{Rfbt}{Rfbb}\right) \; ; \; DESIGN: Vout = 0.8 * \left(1 + \frac{5.62}{1.07}\right) = 5.0019 \, Volts \quad [70]$$
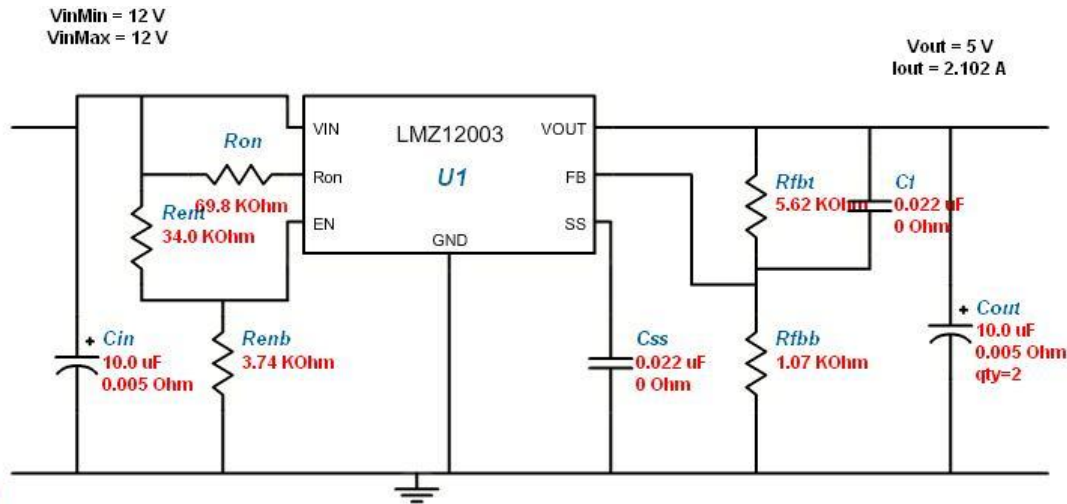
Figure 50: Design Schematic for the LMZ12003

A voltage of 5.0019 is very close to our required output voltage of 5V.

The LMZ12003 regulator worked well, except for the fact that our surface mount resistor order never came. Without these resistors, the output was stuck at 1.8V instead of 5V. We believe it was lost in the mail. As such, we had to go with a stand-by regulator for the final design. The LM323K was acquired, which has the same 5V output and 3A capacity. We made a PCB with a single LM323K DC to DC converter to step down the 12V supply to 5V to power the FPGA and UI. Figure 51 is the schematic of the design to be etched and soldered onto a PCB. Figure 52 is the final power board.
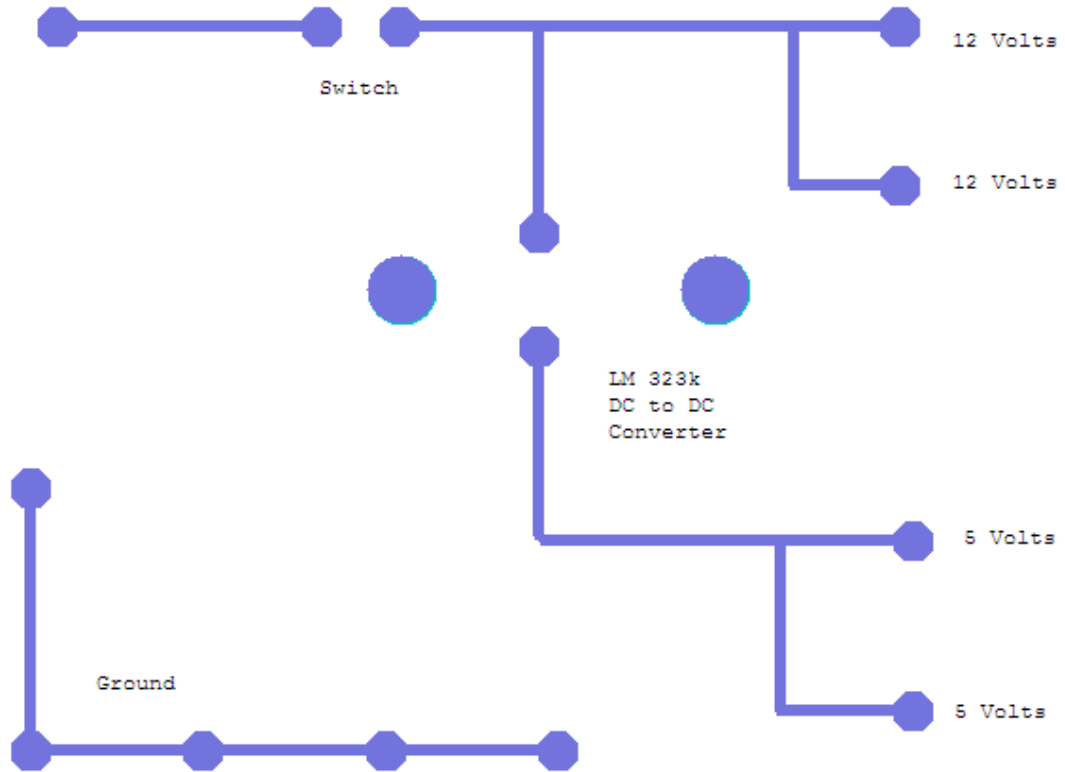
Figure 51: PCB for the power board
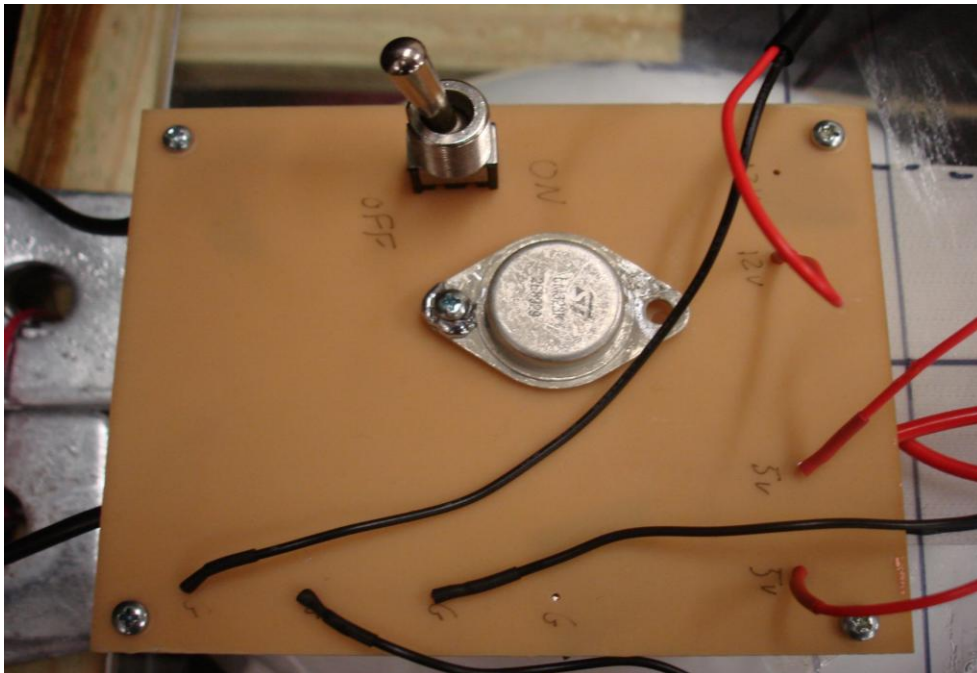
Figure 52: Power Board

3.6.3 On/Off Switch options

There are likely many ways to go about switching the power on and off for the Z-Goggles. The first thought that came to mind would be to use a simple switch between the power source and the first regulator. A second idea was to use the voltage regulators coming directly off the battery, and send them a power down signal, if possible. Safety is the main issue here, as we don't want parts to receive an overabundance of current or voltage, which may damage the parts or cause performance degradation.

In regards to the first option, simply switching off the input voltage to the regulator would probably work. Voltage regulators generally have a threshold that they shut down or enter standby mode when reached, which would remove power from all other regulators, and everything would lose power from there. In theory, this makes sense, but it would require some testing to see if any ill effects are noted when it is implemented. However, simpler can be better, so this option is the first choice for the On/Off switch design. A simple switch was located at Radio Shack, which is rated for 5A at up to 125V AC, so it should be able to handle our 12V battery input and estimated 2.2A current draw [72].

Due to the fact that we had to use the LM323K regulator, our only On/Off switch option turned out to be the first option of switching off the battery voltage directly. This option worked perfectly.

3.7 Helmet/Backpack

The Z-goggle element and backpack combination act as a container for the various components of the system, while making the system wearable for the user. Simply put, the FPGA Video Processor, resistor ladder, camera, and power board are located on the helmet. The UI controller is attached to the FPGA Video Processor by long wires, which allows the user to hold the remote. The battery is in a small satchel that is worn on the back of the user. Figure 53 shows the FPGA, camera board, and power board attached to the helmet. For other pictures showing the full system on a user, see Figure 40 and Figure 41.
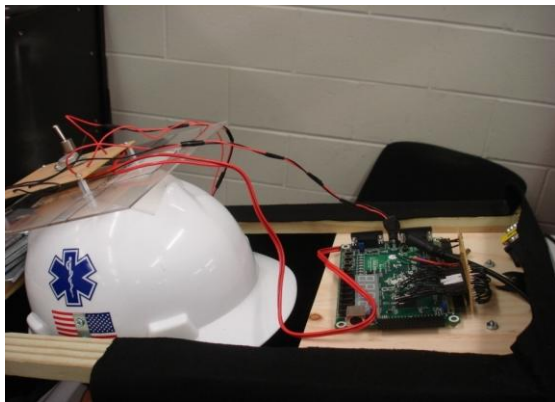

Figure 53: Components attached to helmet

Chapter 4 - Testing

As each sub-system design of the Z-goggles is built, testing will be a continuous important factor in ensuring the final system works. Therefore, coming up with how we will test each stage of design before we actually build anything will greatly reduce testing time as the system is built. The following sections describe how each component of either hardware or software will be tested before integrated together to become the Z-Goggle system.

4.1 Optical System

4.1.1 Data Output Test

Output from the Optical System is extremely important to the overall development of the Z-Goggles system. In total, two capabilities will be tested in regards to data output. Depending on the image/video data that is gathered, extra image processing steps may be required to create a proper display image. It is always possible that the color palette of the output data is off compared to the real world. Also, testing must ensure that the proper format of data is being received from the Optical System. Expectations are RGB formatted data, in 8 bit format. Bear in mind that this must be completed for the visible, IR, and UV cameras separately. The data from all three cameras should only differ in value, not format. Only after the data is correct will the IR and UV cameras be converted to gray-scale before reaching the Video Processor.

First, the data must be in 8 bit RGB format. Generally, this can be determined by devices that are used to create the data, as they are set by them. In this case, the data itself should be tested to make sure that is accurate. Once the camera system is built, including any decoder or other circuitry necessary, it should be connected to the FPGA and powered. The data coming from the device should be determined and compared to 8 bit RGB (3/3/2). If the output is YCbCr, then it should be converted to RGB format as described in section 2.1.3 Digital Image Formats. If the output is 16 bit RGB, the conversion from 16 bit to 8 bit RGB is covered in the same section.

Second, the data must be displayed through any means necessary, which will likely only be possible when the basic Video Processor system architecture is completed. It is unnecessary to complete any functions, but the input to output portion of the system must be completed. The system can then be connected to any VGA display device, preferably a VGA computer monitor, and the correctness of the data can be tested. The data should be tested inside and in sunlight, to vary the types of light sources encountered. For each camera, the separate Red, Green, and Blue channels should be shown on the screen, to compare the channel data. This will allow one to determine if a specific portion of the data is causing the problem. If the colors are an incorrect representation of the scene being imaged, color palette changes will be needed before the data reaches the Video Processor. The same type of color palette processing should be applied here as is done after the image processing functions are done in the Video Processor.

4.2 Video Processor

To manage the test subsystem, a simple circuit must be used. The board only has 8 switches, 4 buttons, 8 LEDs, and 4 seven-segment displays which operate from a fixed 16-bit register. To initiate a test, this hardware must be used, and to select test parameters, the same hardware must be shared. A de-multiplexer will be used with the inputs assigned to user switches, and the outputs latched and clock from a pushbutton, pushing a single line high as a signal that a certain test should execute and continue executing. Each test module can then use a XOR gate along with a buffer to execute any kind of test initialization. The test circuitry is shown in Figure 54.
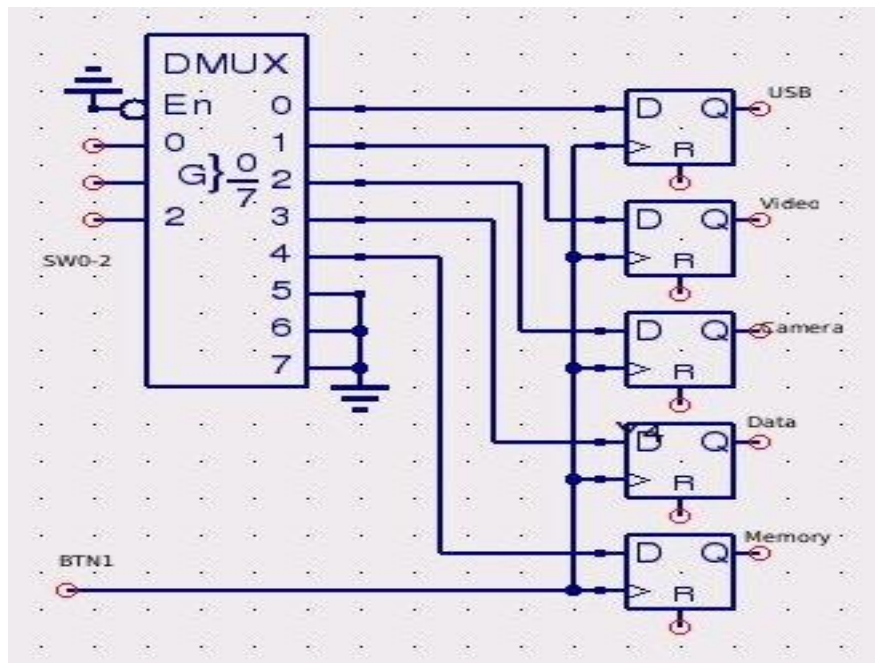
Figure 54: Testing Circuit

7-Segment Display- Many testing modes will require the use of a 7-segment display for data output. These 4-digit 7-segment displays have separate cathode lines and separate anode lines. This means that each separate digit must be pushed one at a time (otherwise all of the digits would display the same thing). The data itself is stored in a 16-bit register. This register is connected to some translation logic which converts each 4-bit value into a series of cathode pins to represent a hex number. To display the numbers themselves on the display, an adjustable length register is used, of which a few primary taps are used to signal the anode pins. By adjusting the cycling buffer length, we can adjust for clock timing. By adjusting the spacing of the taps, we can adjust display timing. Adjusting both of these is essential because otherwise the display will flicker or smearing will be visible across the display. The taps are only put in 2 places and this is used to drive a multiplexer which selects the current anode and cathode output set.

The LEDs in the segments update very quickly, so the primary reason for a clock division circuit is to just not waste power driving them. It is best to start out with a large number for the divider width, which will produce flicker on the board, and then decrease it until no flicker is observable for a constant output. After doing so, selecting the taps should be done with the taps as far apart as possible, and then making them closer together until there is no visible smearing on the display itself. If the clock division is too low, smearing will occur, if it is too high, flickering will occur. If the tap distance is too low or too high, smearing will occur. This module will be used in the output stages of many of the testing components.

To drive the display, we implemented each discrete component in Verilog to allow any 16-bit input to be displayed

```
.
module hex7seg(input [3:0] o, output [6:0] s);
assign s[0]=~((~o[2]&~o[0])|(o[3]&~o[1]&~o[0])|(o[3]&~o[2]&~o[1])|(o[1]&~o[0])|(~o[3]&o[1])|(o[2]&o[1])|(~o[3]&o[2]&o[0]));
assign s[1]=~((~o[2]&~o[0])|(~o[3]&~o[2])|(~o[3]&~o[1]&~o[0])|(~o[3]&o[1]&o[0])|(o[3]&~o[1]&o[0]));
assign s[2]=~((~o[3]&~o[1])|(~o[3]&o[0])|(o[3]&~o[2])|(~o[1]&o[0])|(~o[3]&o[2]));
assign s[3]=~((~o[2]&o[1]&o[0])|(~o[3]&~o[2]&~o[0])|(o[2]&o[1]&~o[0])|(o[2]&~o[1]&o[0])|(o[3]&~o[1]&~o[0]));
assign s[4]=~((o[3]&o[2])|(o[1]&~o[0])|(~o[2]&~o[0])|(o[3]&o[1]));
assign s[5]=~((~o[1]&~o[0])|(~o[3]&o[2]&~o[1])|(o[3]&~o[2])|(o[2]&o[1]&~o[0])|(o[3]&o[1]));
assign s[6]=~((o[1]&~o[0])|(o[3]&~o[2])|(o[3]&o[0])|(~o[3]&~o[2]&o[1])|(~o[3]&o[2]&~o[1]));
endmodule

module muxit(input [3:0] a, input [3:0] b, input [3:0] c, input [3:0] d, input [1:0] s, output [3:0] o);
assign o[0]=(~s[1]&~s[0]&a[0])|(~s[1]&s[0]&b[0])|(s[1]&~s[0]&c[0])|(s[1]&s[0]&d[0]);
assign o[1]=(~s[1]&~s[0]&a[1])|(~s[1]&s[0]&b[1])|(s[1]&~s[0]&c[1])|(s[1]&s[0]&d[1]);
assign o[2]=(~s[1]&~s[0]&a[2])|(~s[1]&s[0]&b[2])|(s[1]&~s[0]&c[2])|(s[1]&s[0]&d[2]);
assign o[3]=(~s[1]&~s[0]&a[3])|(~s[1]&s[0]&b[3])|(s[1]&~s[0]&c[3])|(s[1]&s[0]&d[3]);
endmodule
module clk2sel(input wire clk, output [1:0] s);
reg [10:0] clkdiv;
assign s = clkdiv[10:9];
always @(posedge clk) begin
clkdiv=clkdiv+1;
end
endmodule

module disp7(input wire mclk, input [15:0] data, output [6:0] cathodes, output [3:0] anodes);
wire [3:0] digit;
wire [1:0] sel;
clk2sel a(.clk(mclk),.s(sel));
muxit b(.a(data[3:0]),.b(data[7:4]),.c(data[11:8]),.d(data[15:12]),.s(sel),.o(digit));
hex7seg c(.o(digit),.s(cathodes));
assign anodes[0]=~(~sel[1]&~sel[0]);
assign anodes[1]=~(~sel[1]&sel[0]);
assign anodes[2]=~(sel[1]&~sel[0]);
assign anodes[3]=~(sel[1]&sel[0]);
endmodule
```

The hex7seg module converts an arbitrary 4-bit binary number into a series of cathode outputs. The muxit module implements a 4-1 multiplexer without any conditionals, allowing the system to update asynchronously. The clk2sel module manages clock division manually instead of using the clock modulation circuitry provided onboard. Several options were considered because a linear adder seemed like a poor choice in terms of scaling, but it worked out fine. The tap and size selections are for the Nexys2 board output system.

4.2.1 Video Output Test

To test video output the VGA controller hardware will be tied to a pattern generator instead of to RAM. The first mode of test will supply solid color data to the VGA controller. For the second test random data from the LFSR will be sampled and latched for one second while the LFSR output is placed onto the on-board LEDs to check for any color or refresh anomalies at boundaries. The third test will allow the solid color to be selected by the on-board switches to program in any specific color for verification.

4.2.2 Camera Acquisition Test

To test camera acquisition by itself, no data will be transferred from the camera to RAM. The camera will supply some form of data output which will be compared to as the camera is sampled to ensure that the camera is sending data and operating in the correct mode for our use. A match to correctly formatted data will be signaled on one of the on-board LEDs. This is run in concert with the Data Output test discussed in section 4.1.1.

4.2.3 Data Modification Test

To test the data modification system, preloaded data registers will be used. Each mode will be selected from the remote as per usual operating procedure. Pre-computed data will then be compared against the video system output and if there is a match, an indicator LED on the board will light up, signaling a pass of the test. For some modes this will be a pure data input versus data output check, for others, taps on memory address lines will be required.

4.2.4 Memory Throughput Test

Testing memory is difficult to do well. There are several things we need to test. The first and most important is checking memory to make sure that all of it actually works to begin with. To do this, a counter will be used to supply the address information, and 2 cycles will be run, the first writing a constant set of 1's, the second writing a constant set of zeroes. After each write, a read on the same memory location will take place to ensure that data reads and writes work correctly. If any of the reads and writes fail, a signal LED will be turned on to show that there is at least one error in memory. The lowest 16-bits of the address will be displayed on the on-board 7-segment display and the upper 2 bits will be displayed on LEDs and the test will halt so that manual inspection can be done. This test is essential. If the board has bad segments of RAM, it will allow us to move our video buffer offset to minimize or eliminate storage or reading errors.

The other important test for memory is data throughput. Two data samples will be taken from the LFSR for use in the memory data and the test controller will attempt to read and write from memory in several different modes. In the first mode, it will try to simulate average memory load expected from the video processing system and video output system reading at the beginning of the pixel clock and writing towards the end of the

pixel clock. In the second mode, it will deliberately attempt to make reads at inconvenient times. In the third mode, it will attempt to make simultaneous reads and writes to different locations.

Each mode will be tested using the acquired pseudorandom data and a clocked counter to determine how long the memory operations together took, giving us an average benchmark of each memory controller under different types of loads, allowing us to select the best available option when profiling our video processing system. The average memory cycle time in clock cycles will be displayed on the 7-segment display and the memory testing mode will be selectable from using input switches. When this testing is complete, we should be able to estimate the speed at which the processor is completing the viewing mode functions.

4.3 Display

Testing the display system will involve first testing for power, once we can get power to the screen will need to test the VGA soldering. To do this we will connect the display to a desktop computer. After the display is tested and working, we will need to test the display interface for getting image data from the FPGA to the screen, then we can test the memory controller to see if we can get full frames from memory.

4.3.1 Data Input Test

After getting power to the Display via the 5V power regulator, the design team will need to test the data coming into the VGA port on the FGPA board to make sure the correct images will be outputted to the LCD screen. A requirement set by the design team is that the display will be capable of displaying 30 frames per second. While it is already known that the display capable of this, we do need to test how fast we are getting frames from the FPGA board and outputting them. If all forms of video outputted look smooth and continuous then an actual test will not be needed, but since some function combinations may take more processing that others, we cannot be sure that that the frame rate will be the same for all outputted data. Each possible function combination will be looked at, and tested further if not completely smooth looking. To test the exact frame rate displayed by a particular function combination is hard to do because the system is built to display the frames as fast as it can, up to 60 frames per second.

4.3.2 Display Interface Test

To make sure the design interface works correctly we will test each level of a VGA controller we build. The first test was to output solid colored frames to a VGA monitor, by setting the RGB data for an entire frame to the switches on the Nexys3 board, and placing it into a monitor. This first test was successful showing us we could export RGB data to a LCD monitor.

Next we will test getting a frame from memory and outputting it. Once we can store a frame into memory, we can attempt to output it. After this all we need to do is test to make sure we can output at the desired speed of around 30fps.

4.4 User Interface

The UI sub-system of the Z-goggles is composed of primary just the seven switches and the signals they send to the FPGA board. So while final testing of these switches depend largely on the rest of the system as a whole working properly, we can test ahead of time that the signals sent from the switch combinations are correctly taken in and sent out. The first thing we need to do to test the UI design laid out in the design section is program the microcontroller to do something. As discussed earlier the ATMega168 microcontroller will be placed on an Arduino Duemilanove development board to be programmed. The ATMega168 will be purchased with the Arduino boot loader already on the device. Arduino's open source IDE will then be used to write up the program and transfer it to the board via a USB cable.

4.4.1 Function Test

To test the function switching of the system, we will first program the microcontroller from a PC and add in a print statement. Then the program will be run from the development board, with the switches connected on a breadboard, on the PC. The print statement will output simply the seven bit binary number indicating a "1" or "0" for each of the switches. After testing on a PC, the microcontroller, while still on the Arduino development board and connected to the switches on a breadboard, will be connected to a FPGA board. To test the hardware setup we will first program the microcontroller to simply send out seven signals for the seven switches states, disregarding the look-up table. On the FPGA board we will set these signals to light up seven separate LEDs. This will confirm that there is no problem in the hardware setup and will allow for software testing. To test fully programmed microcontroller we will first use the same LED setup to confirm that the software only allows the function/switch combinations it was built for. Then we will test the UI with the entire system, or as much as possible due to time constraints before we send out the schematic to be made into a printed circuit board. This test largely relies on the rest of the system being in working order. After receiving the PCB we will solder the UI together and perform all the tests again before placing the PCB into a closed off switchbox. The remote box will be built to allow for simple dismantling in case any circuit issues arise after initial testing.

4.5 Power Supply

The main purpose of power testing is to check the voltage assumptions and current draw estimations that the design was based on. Essentially, we must make sure the circuits can all handle what is being given at the input and output junctions of the devices. This should cause a confirmation or a reassessment of the design parameters. The tools for this testing are a voltage meter and an ammeter.

4.5.1 Battery - Power Test

The general output voltage and output current of the battery should be tested with each component attached. This is crucial, to confirm that the battery doesn't have to provide too much current and will reach the required usage time. This also has implications for the durability of the battery. The current draws from the fully loaded regulators and the voltage provided to these regulators is what is at stake here. It is important to check here that the voltage regulators aren't exceeding their maximum ratings as well. In the case of the LMZ12003, that limit is 3A, and is the only one that could realistically be reached. These initial tests are crucial to the operation of the rest of the power supply circuit.

4.5.2 Display - Power Test

To test the power being received at the display we will compare the voltages being applied to the circuit to the voltages being received in the display itself. For long term use we will run the system for twice the use time to ensure that the power supply is constantly being received into the display. Our estimation here was 500 mA current draw and 5V to the display. We may need to adjust the backlighting or other settings on the PSone display to get the draw to manageable levels.

4.5.3 Video Processor - Power Test

To test the power being received into the FPGA controller we will compare the voltages being applied to the circuit to the voltages being received in the FPGA itself. For long term use we will run the system for twice the use time to ensure that the power supply is constantly being received into the FPGA. The output of the LMZ12003 power module that connects to the FPGA battery input connection should be tested for voltage level and current draw. We hoped for at most 1.5 A of current draw from the FPGA, and 5V to the FPGA.

A second aspect of this test is to check that the FPGA is providing power properly to the 3 AL242 Line Doublers that are designed to run off the FPGA itself. As such, 3.3V and 1.8V sources from the FPGA must be tested as well, and the current draw from the Doublers. This was estimated at 92 mA from each, typically. Third, the 3 AL51s that connect to the 5V regulator will need to be tested. Again, we assumed that 92 mA will come from each part.

4.5.4 Optics - Power Test

To test the power being received at the cameras we will be comparing the voltages being applied to the circuit with the voltages being received into the optics. This basically entails checking the voltage and current input/output of the LM2698-ADJ to make sure the change in voltage is minimal; it should be roughly 12V input and output. The current

draw from all three cameras was estimated at 150 mA, so this should be somewhat close or over what is determined during the test.

4.5.5 UI - Power Test

The UI microcontroller will be tested in a similar manner. We assumed a small current drain from this device of 50 mA, though it should end up being lower. It should take 5V from the LMZ12003.

4.6 Helmet/Backpack

The way the helmet and backpack design is setup is a very important factor of the Z-goggle system. Thorough testing must be done in order to meet the safety requirements provided by the design team. Ensuring that the system's users are not at risk of any electrical shocks, fires or falls is of upmost importance. The following descriptive wear tests will explain how these safety measures will be met.

4.6.1 Wear Tests

The helmet design choice will depend on the camera modules chosen to use but either way most of testing will be the same. Here we discus general measures that will be taken in testing and will cover both designs. In both designs the display system remains the same. The LCD screen will be attached to the helmet and enclosed. While the intent of the enclosure is to block the peripheral vision of the user, the bottom of the display enclosure will be partly open to allow the user to look down. This will allow the user to see the remote as well as any change in the level of the ground. The display enclosure will be tested to make sure enough downward vision to see obstacles is allowed, while not effecting the user's experience with the display. The enclosure will be designed with weight factors in mind, but will still need to be tested for durability, and long term head use. The display must stay static with the user's head, but cannot weigh enough to affect the user's experience. These tests will be simple and will require testing different weights added to the back of the helmet to test the effect of leveling out the weight. The display enclosure must also be long term tested with multiple viewing modes on, to ensure no significant eye problems, or headaches occur. If the helmet is required to hold the FPGA boards and their incasing, then further stability and durability test will be performed.

Wear testing for the helmet and backpack will focus mainly on making sure connections are safe and reliable, while the system is in movement. If the backpack is not secure the entire system is at risk. We will test the backpack in wear tests before mounting the electronics, and again after. Wear testing will include standard stress and unusual movement tests.

Chapter 5 – Budget and Milestones

5.1 Budget

The target budget of this project is be less than or equal to $800.00. The budget of the project will be supplied by the design team out of their personal finances. We predict that each member will contribute equal amounts to the project. If there is a member that does not contribute an equal or equivalent amount to the project then he is expected to compensate the rest of the design team so that the overall cost to each member is equal. We have separated the budget into these separate areas as shown in Table 14.

Table 14: Preliminary budget breakdown of the project

Purpose	Budgeted Amount
Testing	$100.00
FPGA	$120.00
Optics	$180.00
Display	$150.00
Power Supply	$150.00
Design Software	$100.00

The final budget of our project turned out to be close to what we set out at the beginning. Shown in Table 15 is our final project budget.

Table 15: Final Project budget

Component	Cost
C3188A Camera	$57.00
Replacement C3188A Camera	$105.00
Spartan 3 FPGA	$200.00
LCDP5VGA Monitor	$160.00

PS1270 Battery	$19.00
Battery Charger	$40.00
Power Board creation	$25.00
Helmet/Backpack creation	$20.00
Arduino Duemilanove	$30.00
Project Box (Remote)	$5.00
Resistor Ladder components	$12.00
Testing/Broken Components	$100.00
Total Cost of Project:	$773.00

5.2 Milestones

We began our project with the research phase for each member's respected area of expertise and interest. We planned that this would continue for the majority of the summer semester and until we have gathered enough information to begin making educated design decisions. In the period that we have a research and design overlap we anticipated that this would be towards the end of the summer semester in late July. By the end of the semester we have the deadline of turning in our research gathered and design to complete the summer. In the break between the summer semester and the spring semester we will be conducting preliminary tests on separate parts for our separate areas of interest to become familiar with the hardware we will be working with. At the beginning of the fall semester we will continue with the preliminary tests as well as refine our design as per the results of our testing. After testing is completed and all group members are familiar with the designs we will begin writing the code for the FPGA interfaces and laying our circuit board designs. We predict that this will take the bulk of our time because most of our project will be concentrated on programming everything

correctly. This also allows us to do some work without necessarily needing the parts at the same time. By October we plan to have the bulk of our code for the separate systems written and will begin to integrate the separate systems into the final product. This integration phase we predict will last until a half month before the project is due. At that point we plan to begin the final testing phase of the complete system and work out any bugs that result from the integrations. This is predicted to last until the project is due at the beginning of December. Each Gant chart for the sub-systems follows in Tables 16-20.
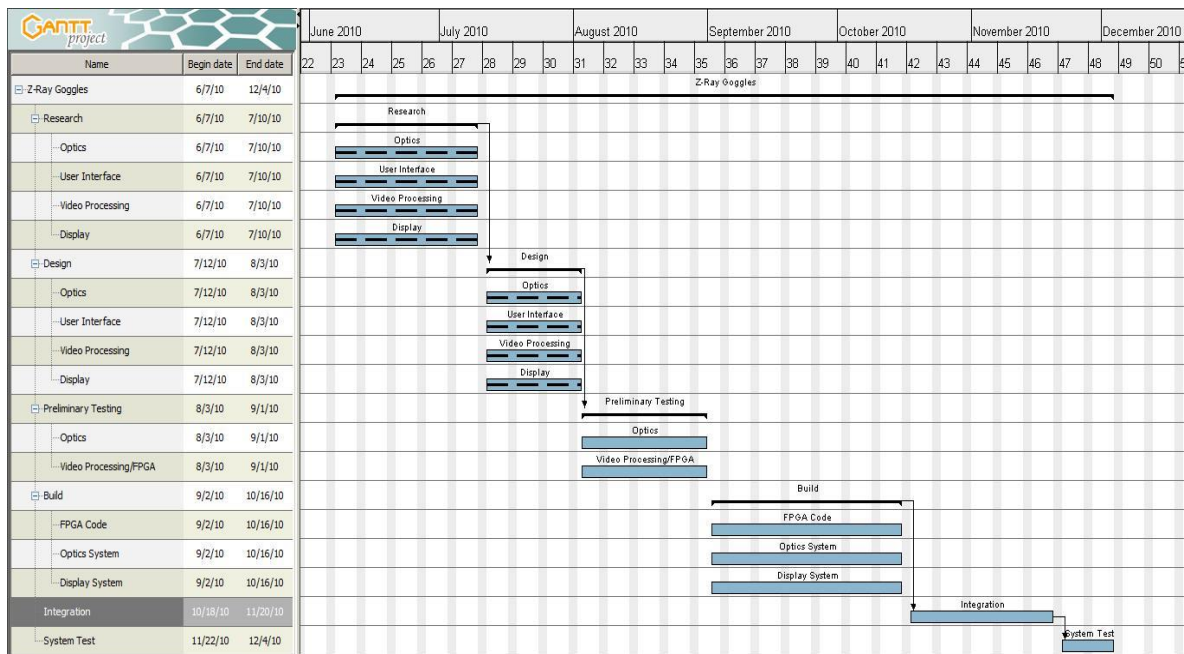
Table 16: Overall Gantt chart
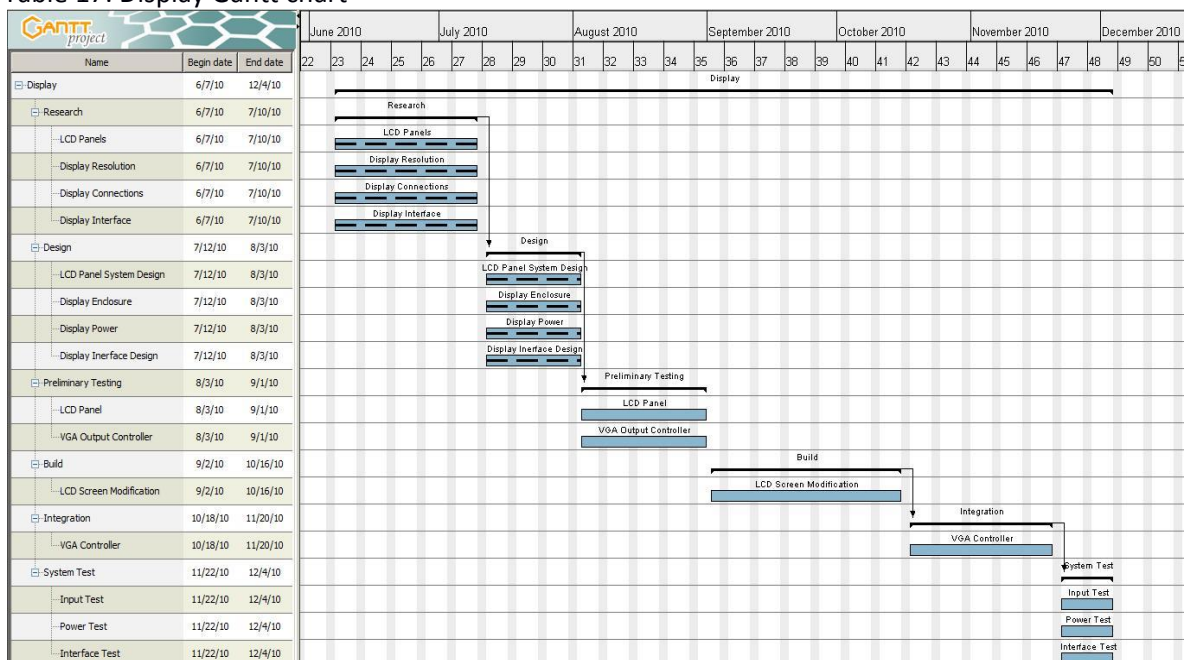

Table 17: Display Gantt chart
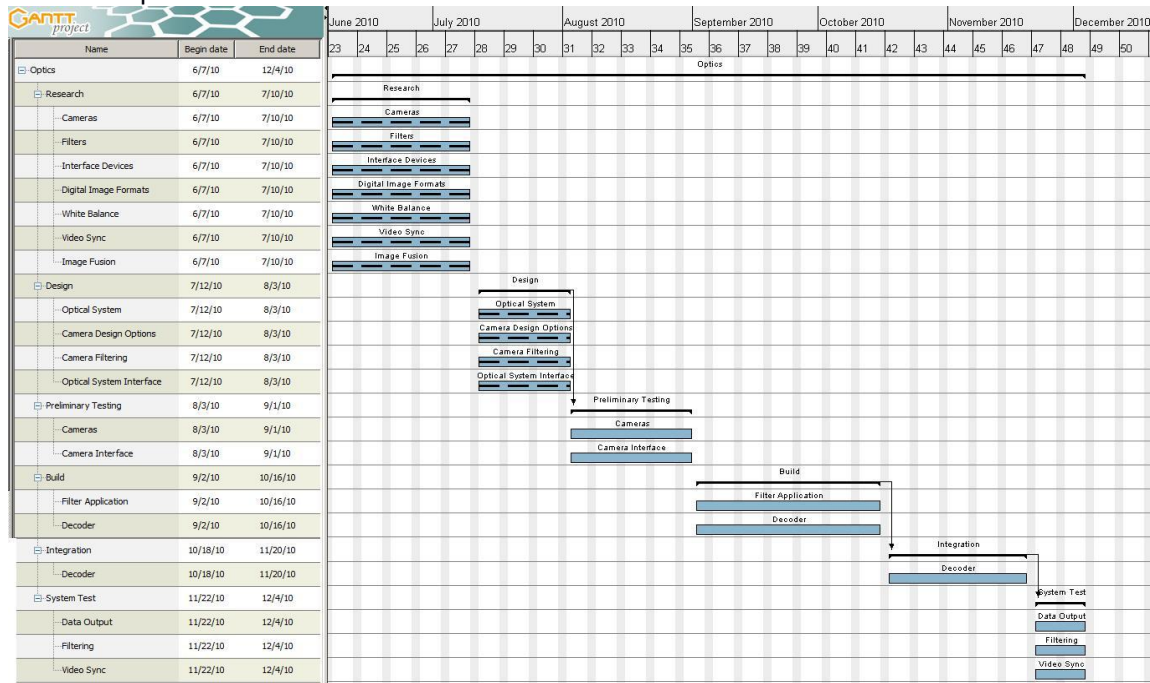
Table 18: Optics Gantt chart

Table 19: UI Gantt chart

Table 20: Video Processor Gantt chart

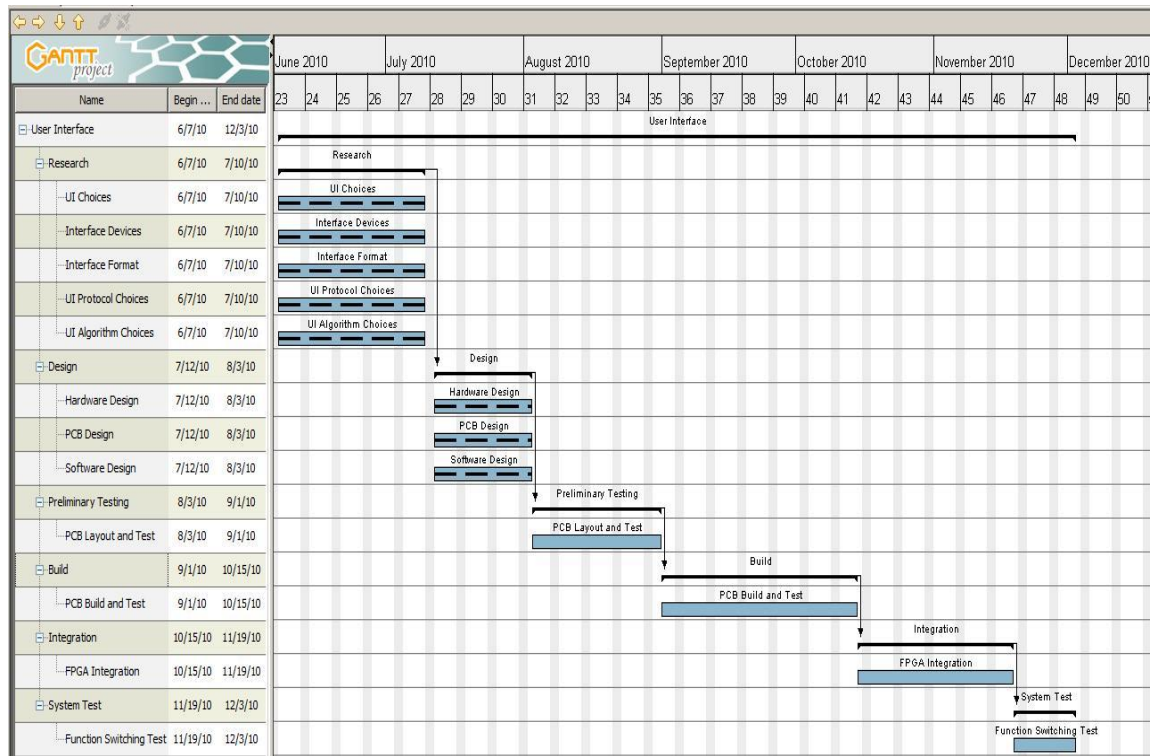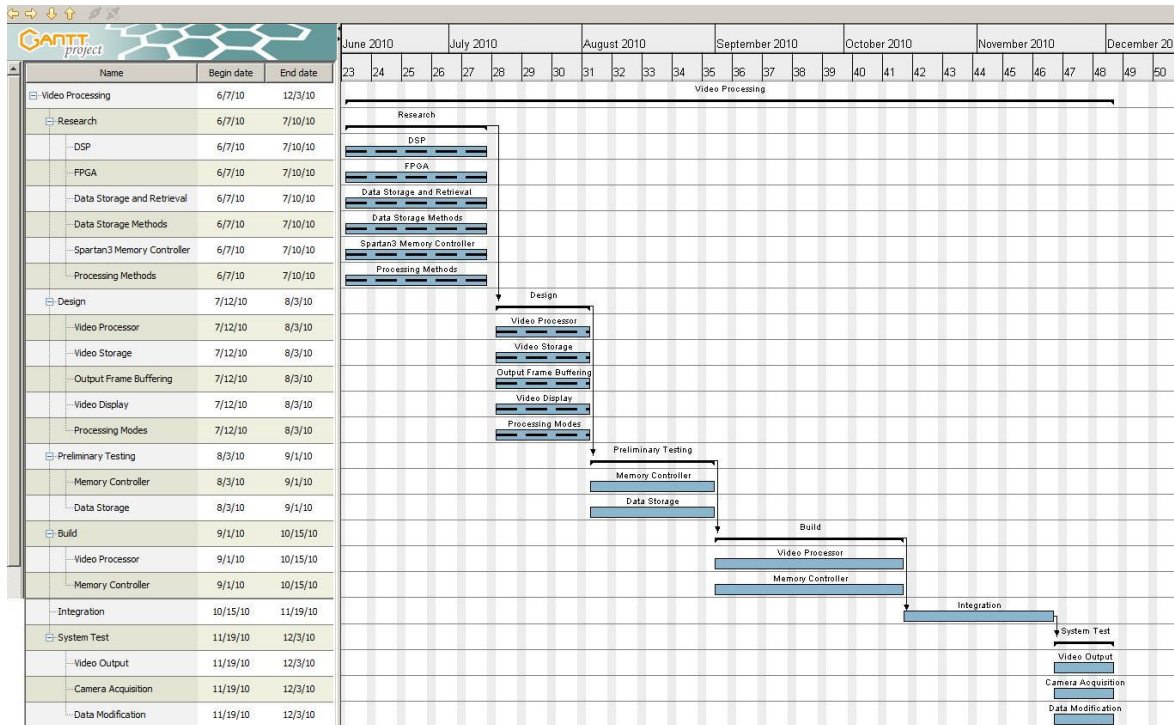| Name	Begin date	End date
Video Processing	6/7/10	12/3/10
Research	6/7/10	7/10/10
DSP	6/7/10	7/10/10
FPGA	6/7/10	7/10/10
Data Storage and Retrieval	6/7/10	7/10/10
Data Storage Methods	6/7/10	7/10/10
Spartan3 Memory Controller	6/7/10	7/10/10
Processing Methods	6/7/10	7/10/10
Design	7/12/10	8/3/10
Video Processor	7/12/10	8/3/10
Video Storage	7/12/10	8/3/10
Output Frame Buffering	7/12/10	8/3/10
Video Display	7/12/10	8/3/10
Processing Modes	7/12/10	8/3/10
Preliminary Testing	8/3/10	9/1/10
Memory Controller	8/3/10	9/1/10
Data Storage	8/3/10	9/1/10
Build	9/1/10	10/15/10
Video Processor	9/1/10	10/15/10
Memory Controller	9/1/10	10/15/10
Integration	10/15/10	11/19/10
System Test	11/19/10	12/3/10
Video Output	11/19/10	12/3/10
Camera Acquisition	11/19/10	12/3/10
Data Modification	11/19/10	12/3/10

Chapter 6 – Project Summary

It was the goal of the Z-Goggles design team to learn about a number of things. The project was chosen to further expand our knowledge in topics such as FPGA programming, hardware path design, microcontroller programming, camera filtering, image processing algorithms, power consumption, voltage regulators, PCB design, encoding and decoding data formats, and overall system design. We learned quite a bit during Senior Design 1, and then even more during Senior Design 2.

During the time period from the start of the design process to the end of Senior Design 2, many original design ideas changed. While originally the Digilent Spartan3 board was purchased and used in researching things such as a memory controller, or video processing algorithms, it was later realized that the Spartan3 board's VGA controller was only capable of outputting 8 colors. From there the design team decided to connect the Nexys2 FPGA board that was already available to the Spartan3 board, and use the Nexys2's VGA controller. After further research the design team realized this may be difficult do the lack of pins available on the Spartan3. Eventually the design team came to the decision that the Nexys2 board would be capable of doing all the work, and had enough pins. This decision was scrapped in Senior Design 2, where we decided to use the Spartan3 board for the final design, due to the fact that we needed its RAM to complete some of the functions we wished to do.

The camera system was another section that had completely changed multiple times through both semesters. When the project began, USB webcams were considered to be the cheapest, easiest option. After realizing it may require reverse engineering Windows webcam drivers, and that the formatting of the webcam data may be proprietary, webcams were discarded. The indecision between very small digital camera modules and analog cameras that require decoding and reformatting was caused by a lack of previous knowledge on the subject, and inconclusive camera module datasheets. Eventually the design team chose to use a analog camera module, due to the fact that the lens isn't too small to add or remove filters, and the pin connections are significantly more accessible. After deciding on the analog camera, the design team knew a video decoder would be required. The design team was unaware of the fact that NTSC analog cameras usually use an interlaced scanning system, as appose to a progressive format. Luckily the design team was able to find an IC that did just the job they needed, but it meant that an additional part would be needed for each camera. These decisions were then changed again at the beginning of Senior Design 2, when a digital camera module became available that would directly interface with the FPGA Video Processor. This turned out to be a blessing and a curse, because it allowed us to quickly begin work on the Video Processor architecture, but the camera's I2C functionality was buggy and caused delays in development trying to get it functional. Deciding to bypass I2C allowed work to progress better.

The final issue we dealt with was the Digital to Analog conversion necessary to get a VGA output. We originally used a 10-bit DAC that worked sporadically, and which seemed destined to fail when we needed it to work properly. As such, our decision to create a resistor ladder to handle the VGA output conversion worked phenomenally well, allowing us to achieve 12-bit output video fairly efficiently, which showed far more color depth than we had originally planned.

While many small changes have been made, the Z-Goggles project is above all an entertainment system. The main focus of the project was to create something that would be fun to use, while pushing the design team to advance key skills in electrical and engineering fields. We learned the value of designing well, and also the amount of issues that could cause design changes. Our group persisted through more system changes than any other group, and we still met or exceeded almost all of our requirements. Thus, the Z-Goggles project was a success.

Project Operation Manual

To start off using the Z-Goggles, the user must have some to assist with the attachment of the helmet apparatus and the battery backpack. First the helmet is attached with the facing forward and looking straight ahead while he/she is seated for ease of the assistant. Then the user will stand up and the assistant will help the user put the backpack on, careful not to disconnect the power wires that are running from the top of the helmet and into the backpack. The assistant will then turn the power switch from the off to the on position and allow the user to begin using the Z-Goggles. For proper implementation the user will begin in regular RGB vision with no alteration being done so the user can adjust to and new visions easily.
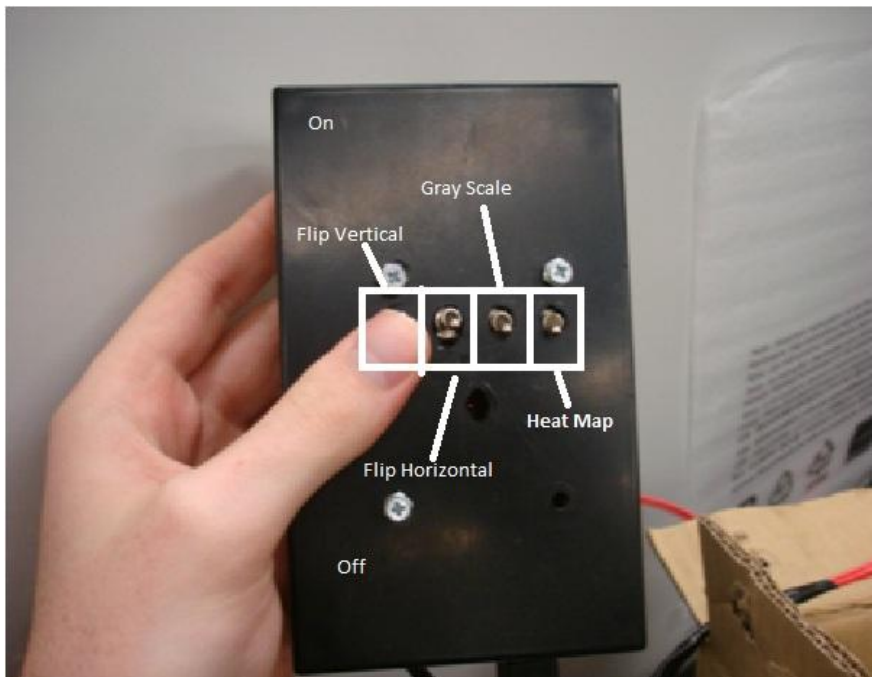

Figure 55: Remote with function indicators

As show in Figure 55 the user will control the Z-Goggles by the simple remote shown. The remote has 4 functions attached to the corresponding switches. The function combinations are such that vertical and horizontal can be implemented at all times, singularly or together. Only the gray scale or the heat map can be activated one at a time, if the user tries to activate both at the same time then a red LED will light up in the box and the current set of functions activated will remain the same.

It is suggested by the designers that the user have a partner to assist them with using the Z-Goggles so as to prevent any injury that may come from the results of an altered state of vision. The user is also advised that prolonged use can cause eye strain, neck strain and temporary vision impairment (normal vision seems flipped after removal of the goggles). The battery life will guarantee the user a minimum of 30 minutes of use time. Any exposure to liquids is to be avoided as it will result in damage to the system as well as any high impact and prolonged vibration to the helmet.

Team Information

Cody Baxley

Cody is currently a College Work Experience Program student working at Lockheed Martin. He works in Internal Research and Development on an infrared sensor system. This work provides experience in image processing algorithm development and testing, with a focus on coding in C++ and MATLAB. Also, he has taken the Optical Engineering undergraduate course offered by CREOL. This class introduced designing with optical components, and provided prior knowledge related to vision systems. In the future, Cody wants to pursue graduate study in Optics or Opto-Electronic Engineering.

Geoffrey Jean

Geoff is an electrical engineering student at the University of Central Florida. His current interests lie in electromagnetic waves, specifically microwave frequencies. He has taken Microwave Engineering and Electric Machinery as his technical electives so far and plans to take the graduate course Antenna Design in the fall. His other interests include FPGA design and integration of such into communication systems. In the future, Geoff would like to work for a company doing communication design or some related field.

Will Petty

Will Petty is a Computer Engineering major at the University of Central Florida. His current interests lie primarily with software development in C, C++, Java, Verilog, and Actionscript. He has taken Computer Architecture, and Operating Systems which teach system design of both software and hardware. In the future, Will wants to work for a company doing a wide range of software development.

C.J. Silver

C.J. Silver is an Electrical Engineering major at the University of Central Florida. His current interests lie in embedded device programming and digital hardware prototyping. He has taken Computer Architecture, Computer Science 1, and Electric Machinery as his technical electives. In the future, C.J. wants to work for a company doing embedded hardware design and programming.

Appendices

<u>Appendix A</u> – Works Cited

[1] All You Ever Wanted to Know About Digital UV and IR Photography, but Could Not Afford to Ask, Bjorn Rorslett
http://www.naturfotograf.com/UV_IR_rev00.html#top_page

[2] FPGA Based Embedded Vision Systems, Lixin Chin
http://robotics.ee.uwa.edu.au/theses/2006-Embedded-Chin.pdf

[3] CMOS Digital Image Sensors, Craig Peacock
http://www.beyondlogic.org/imaging/camera.htm

[4] CMU Cam, Carnegie Mellon University
http://www.cmucam.org/

[5] Dragonfly II, Point Grey
http://www.ptgrey.com/products/dragonfly2/index.asp

[6] Logitech C200 Webcam, Logitech
http://www.logitech.com/en-us/webcam-communications/webcams/devices/5865

[7] TCM8230MD Camera module, SparkFun Electronics
http://www.sparkfun.com/commerce/product_info.php?products_id=8667

[8] CM-26N/P Camera module, SparkFun Electronics
http://www.sparkfun.com/commerce/product_info.php?products_id=8739

[9] Interlace vs Progressive Scan
http://searchwarp.com/swa21525.htm

[10] How to make a webcam work in infrared, Geoff Johnson
http://www.hoagieshouse.com/IR/

[11] Digital Photography For What It's Worth, Jeremy McCreary
http://www.dpfwiw.com/filters.htm#ir

[12] Medical and Scientific Photography, Prof. Robin Williams and Gigi Williams
http://msp.rmit.edu.au/Article_01/06.html

[13] Photography in the Ultraviolet Spectrum
http://www.instructables.com/id/Photography-in-the-Ultraviolet-spectrum/

[14] ADV7180 Video decoder, Analog Devices

http://www.analog.com/en/audiovideo-products/video-decoders/adv7180/products/product.html

[15] ADV7401 Video decoder, Analog Devices
http://www.analog.com/en/audiovideo-products/video-decoders/adv7401/products/product.html

[16] Semiconductor Store AL242
http://www.semiconductorstore.com/cart/pc/viewPrd.asp?idproduct=42043

[17] Semiconductor Store AL251A
http://www.semiconductorstore.com/cart/pc/viewPrd.asp?idproduct=3747

[18] EVB-USB3300-XLX User Manual
http://digilentinc.com/Data/Products/EVB-USB3300-XLX/evb3300xlxuser.pdf

[19] Digital Camera RAW image formats
http://www.fileinfo.com/help/raw-image-formats.html

[20] Digital Photography - Color Models, Mark Habdas
http://www.computerschool.net/dphoto/color.html

[21] Adobe Forums, Forum post: How does Photoshop convert 16 bit RGB values to 15 bit and 8 bit?
http://forums.adobe.com/message/2596315

[22] YUV, YCbCr, YPbPr colour spaces, Discovery Scientific, LLC
http://discoverybiz.net/enu0/faq/faq_YUV_YCbCr_YPbPr.htm

[23] The MathWorks, Technical Solutions: How do I convert my RGB image to gray-scale without using the Image Processing Toolbox?
http://www.mathworks.com/support/solutions/en/data/1-1ASCU/

[24] Tutorials: White Balance, Sean McHugh
http://www.cambridgeincolour.com/tutorials/white-balance.htm

[25] RGB "Bayer" color and Micro Lenses, Silicon Imaging
http://www.siliconimaging.com/RGB%20Bayer.htm

[26] Signal Processing and Performance Analysis for Imaging Systems; S. Susan Young, Ronald G. Driggers and Eddie L. Jacobs. Artech House, 2008.

[27] C5000™ Low Power DSP
http://focus.ti.com/paramsearch/docs/parametricsearch.tsp?family=dsp&sectionId=2&tabId=50&familyId=114

[28] C6452 DSP Evaluation
Modulehttp://focus.ti.com/docs/toolsw/folders/print/tmdxevm6452.html
[29] TVP5158 Evaluation Module
http://focus.ti.com/docs/toolsw/folders/print/tvp5158evm.html

[30] Blackfin Family Selection Table
http://www.analog.com/en/embedded-processing-dsp/blackfin/processors/selection-tables/Blackfin_Selection_Table/resources/fca.html

[31] Nexys2 FPGA board
http://digilentinc.com/Products/Detail.cfm?NavPath=2,400,789&Prod=NEXYS2

[32] Spartan 3 FPGA board
http://digilentinc.com/Products/Detail.cfm?NavPath=2,400,799&Prod=S3BOARD

[33] Chu, Pong P. FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version. Hoboken, NJ: Wiley-Interscience, 2008. Print.

[34] Sobel Edge Detection Operator
http://edge.kitiyo.com/2009/intro/sobel-edge-detection-operator.html

[35] Crystalfontz LCD panel CFAF320240F-T
http://www.crystalfontz.com/product/CFAF320240F-T

[36] Mouser Electronics LCD Panel 568-LS037V7DW01
http://www.mouser.com/ProductDetail/Sharp-Microelectronics/LS037V7DW01/?qs=3FxHq6lNtC7Yyho60c%252bhew%3d%3d

[37] TFT LCD Description
http://www.articlefield.com/25923/tft-lcd/

[38] PS One LCD MOD Forum
http://www.overclock.net/other-hardware-mods/170245-official-psone-lcd-mod-thread.html

[39] A glossary of video terminology
http://serato.com/articles/video-sl/1935/a-glossary-of-video-terminology

[40] Description of Component Video
http://en.academic.ru/dic.nsf/enwiki/175428

[41] Description of a VGA connector
http://www.museumstuff.com/learn/topics/VGA_connector

[42] Wilson, Peter Robert. Design Recipes for FPGAs. Amsterdam: Newnes, 2007. Print

[43] Chu, Pong P. *FPGA Prototyping by Verilog Examples: Xilinx Spartan -3 Version*. Hoboken, NJ: J. Wiley & Sons, 2008. Print.

[44] Averlogic 251A Flier
http://www.averlogic.com/pdf/AL251A_flyer.pdf

[45] ADV 7340/7341 decoder data sheet
http://www.analog.com/static/imported-files/data_sheets/ADV7340_7341.pdf

[46] Switch Debouncing Description
http://www.labbookpages.co.uk/electronics/debounce.html

[47] Embedded systems design: By Steve Heath, Goggle Book link
http://books.google.com/books?id=cQrCEQPct0wC&dq=Embedded+systems+design:+By+Steve+Heath&printsec=frontcover&source=bn&hl=en&ei=PTFXTNHsOYzSsAPF2rHaAg&sa=X&oi=book_result&ct=result&resnum=4&ved=0CCgQ6AEwAw#v=onepage&q&f=false

[48] How to choose a Microcontroller
http://www.instructables.com/id/How-to-choose-a-MicroController

[49] PIC Microcontroller
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2551

[50] Arduino
http://www.arduino.cc/

[51] Arduino Duemilanove
http://arduino.cc/en/Main/ArduinoBoardDuemilanove

[52] SparkFun Lecture 4 - UART and Serial Communication
http://www.sparkfun.com/commerce/tutorial_info.php?tutorials_id=104

[53] Computer Vision Syndrome and Computer Glasses
by Wendy Strouse Watt, O.D.
http://www.mdsupport.org/library/cvs.html

[54] VHDL & Verilog Compared & Contrasted Plus Modeled Example Written in VHDL, Verilog and C by Douglas J. Smith
http://www.angelfire.com/in/rajesh52/verilogvhdl.html

[55] Using Simulink with Xilinx System Generator for DSP
http://www.mathworks.com/fpga-design/simulink-with-xilinx-system-generator-for-dsp.html

[56] I2C tutorial
 http://www.best-microcontroller-projects.com/i2c-tutorial.html

[57] Maxfield, Clive. The Design Warrior's Guide to FPGAs: Devices, Tools and Flows. Boston: Newnes, 2004. Print.

[58] Battery University
http://www.batteryuniversity.com/partone-4.htm

[59] PowerStream Battery Chemistry FAQ
http://www.powerstream.com/BatteryFAQ.html

[60] 12 Volt 5 Amp Hour PS1250 data sheet
http://www.power-sonic.com/site/doc/prod/86.pdf

[61] PS1270 battery, PowerSonic
http://www.batterystuff.com/batteries/upc-telecom/PS1270.html

[62] National Semiconductor software page – WEBENCH and Xilinx Power Expert
http://www.national.com/analog/xilinx#software

[63] Linear Technology – Design Support
http://www.linear.com/designtools/software/

[64] How to make a webcam work in infrared, Geoff Johnson
http://www.hoagieshouse.com/IR/

[65] Guide to VGA
http://www.laptop2tv.co.uk/guide/VGA-exp/guide-to-vga.htm

[66] Nexsys2 Reference Manual, Digilent
http://digilentinc.com/Data/Products/NEXYS2/Nexys2_rm.pdf

[67] Arduino Tutorial: Digital Pins
http://www.arduino.cc/en/Tutorial/DigitalPins

[68] Discharge Methods, Battery University
http://www.batteryuniversity.com/partone-16.htm

[69] Power Modules overview, National Semiconductor
http://www.national.com/analog/power/simple_switcher_power_modules

[70] LMZ120003 Power module, National Semiconductor
http://www.national.com/pf/LM/LMZ12003.html#Overview

[71] LM2698-ADJ, National Semiconductor
http://www.national.com/pf/LM/LM2698.html#Overview

[72] SPDT Heavy-Duty Switch, Radio Shack
http://www.radioshack.com/product/index.jsp?productId=2062495#showFullReviews

[73] Linear and Switching Voltage Regulator Fundamentals, Chester Simpson, National Semiconductor
http://www.national.com/appinfo/power/files/f4.pdf

[74] LMZ120003 Power module, National Semiconductor
http://www.national.com/pf/LM/LMZ12003.html#Overview

[75] LM22676MR-5.0, National Semiconductor
http://www.national.com/pf/LM/LM22676.html#Overview

Appendix B – Advice gained

Dr. Samuel Richie, Assistant Director and Professor, Department of EECS at UCF
Dr. Arthur Weeks, Professor, Department of EECS at UCF
Logitech Technical Support

Appendix C –Permissions
Permission for Figures 2 and 3

BeyondVisible to me   show details Jul 15   Reply ▼

Hi Cody

sorry it took so long to get back to you but there has been my daughters graduation, daughters wedding, a family tragedy and a short vacation away from home, so I have been quite busy..

You are welcome to use the two spectra you requested on the condition that they are noted in the figure caption "by permission www.beyondvisible.com". I would also appreciate receiving a copy of your final paper concerning the UV camera and its application.

regards
Shane

----- Original Message -----
From: BeyondVisible
To: Cody Baxley
Sent: Tuesday, June 29, 2010 8:21 AM
Subject: Re: Regarding using your UV filter spectra graphs.

Hi Cody

I am away right now and will get back to you soon.

Shane

----- Original Message -----
From: Cody Baxley
To: shane@beyondvisible.com
Sent: Saturday, June 26, 2010 6:29 PM
Subject: Regarding using your UV filter spectra graphs.

Mr Elen,

I'm a student at the University of Central Florida. I request to use your spectra graphs for the B+W 403 filter and Schott UG 1 filter for my Senior Design paper. I am comparing them for use in an ultraviolet camera solution for a larger project. I would appreciate the use of your work, because I do not have the funds to test the filters myself.

Your site is quite informative on the subject, though I will not use any of your other information in this particular research. Thank you for the work you put into it, and I understand if you do not want me to use the graphs.

Thanks,

Cody Baxley
codyrbaxley@gmail.com

Permission for Figure 14 - RS232 Circuit

Permission for Figure 63 – Infrared photograph of U.S. currency

☆ ● **Cody Baxley** to mariusm   show details Jul 3   ↩ Reply ▼

Mr Milner,

I request to use the infrared photograph of U.S. money that you posted to Flickr. I had the idea that money would be a good way to test an IR camera system that I am working on for a Senior Design project in Engineering. I want to include the picture in our design document, as it was your picture that gave me the idea.

Thank you, and I understand if you do not want me to use the picture.

Cody Baxley
codyrbaxley@gmail.com

↩ Reply   → Forward

☆   **Marius Milner** to me   show details Jul 5   ↩ Reply ▼

Hello, thanks for considering using my picture.

I'll grant you a free license to use the photo in your document as
long as it includes a suitable credit such as:
 Photo used with permission; Copyright 2005 Marius Milner.

Good luck with your project.

Marius
- Show quoted text -

↩ Reply   → Forward

☆ ● **Cody Baxley** to Marius   show details Jul 5   ↩ Reply ▼

Thank you, all of that information will be included.

Cody Baxley

- Show quoted text -

Permission for **Figure 46** - Wiring information for soldering

Bit Tech image   Inbox | X

☆   **Anthony White** to me   show details Jul 14   ↩ Reply ▼

Hi Will

Thanks for your email. Yes, providing that it is only for the reasons you state below, I am happy for you to use the image for your class.

Kind regards

Anthony

Anthony White
Commercial Director - Dennis DMS | Syndication | Magbooks
T: +44 (0)20 7907 6472 | M: +44 (0)7905 234788
www.dennis.co.uk | www.dennisimages.com | www.bizarrearchive.com

Dennis Publishing Ltd, 30 Cleveland Street, London, W1T 4JD

P please consider the environment before printing this e-mail

From: Will Petty [mailto:willpettyucf@gmail.com]
Posted At: 13 July 2010 19:51
Posted To: Reception public folder
Conversation: Asking permission to use an image from a http://www.bit-tech.net/ article
Subject: Asking permission to use an image from a http://www.bit-tech.net/ article

Hi, my name is Will.
  I'm building a video goggle system for a senior design class and I wanted to ask for permission to use a picture from an article on http://www.bit-tech.net/. The article is " named PSone LCD in a PC and was Published on 7th July 2004 by Dave Williams. The image I'm asking permission to use is http://images.bit-tech.net/content_images/2004/07/psone_lcd/schematic.jpg . The paper I would be using this image in is purely for academic reasons and will NOT be published. Please let me know if the image is ok to re-use.

thanks, Will Petty
NOTE: The information in this email is confidential and may be legally privileged, unless stated to the contrary. If you are not the intended recipient, you must not read, use or disseminate that information.
Any opinions or comments are personal to the writer and do not represent the official view of Dennis Publishing Ltd. If you have received this email and are not a named addressee, please contact itdirector@dennis.co.uk immediately by reply email and then delete this message from your system.
Please do not copy it or use it for any purpose, or disclose its contents to any other person.
Although this email and any attachments are believed to be free of any virus, or other defects, it is the responsibility of the recipient to ensure that they are virus free and no responsibility is accepted by Dennis Publishing Ltd for any loss or damage arising from the receipt or use thereof.
Company registered in England No. 1138891
Registered office: 30, Cleveland Street, London, W1T 4JD

Permission for Table 7 - Popular Resolution Pixel Clock and Sync Pluses Reference

asking permission to use table from MIT course website for senior design project. Inbox | X WILL | X

☆ ● **Will Petty** to anantha   show details Jul 24 (9 days ago)   ↩ Reply ▼

Hi, Im a student at the University of Central Florida and am currently working on a senior design project. I want to ask permission to use the table of " lists timing values for several popular resolutions" on the site

http://www-mtl.mit.edu/Courses/6.111/labkit/vga.shtml

The paper is not being published, and is for academic purposes only, Thank you.
-Will Petty
willpettyucf@gmail.com

↩ Reply   → Forward

☆ **Anantha Chandrakasan** to me   show details Jul 24 (9 days ago)   ↩ Reply ▼

no problem.

Will Petty wrote:
    - Show quoted text -
    willpettyucf@gmail.com <mailto:willpettyucf@gmail.com>

--
Anantha P. Chandrakasan
Joseph F. and Nancy P. Keithley Professor of Electrical Engineering
Director, MIT Microsystems Technology Laboratories

Permission for all WEBENCH data

☆ **New Feedback** to me   show details Jul 28 (6 days ago)   ↩ Reply ▼

Dear Mr. Baxley,
Feel free to use any of the data from WEBENCH Designer for your paper.

Regards,
Jeff Perry
WEBENCH Manager
National Semiconductor
(408)721-4545
Jeff.Perry@nsc.com

Please provide a detailed explanation of your technical question?
I am a student at the University of Central Florida. I would like to request to use information and charts from the WEBENCH Power Designer tool in my engineering Senior Design paper. I am unsure how National Semiconductor feels about this. No figures or graphs will be used from datasheets, only the design information will be covered for the purpose of explaining a possible power supply solution.

Permission for Figure 17

Image use permission Inbox | X | geoffrey.tjean@gmail.com | X

show details Jul 27 (7 days ago) | Reply | ▶

Geoffrey - **Resume** to director

Hello,

I am working on a senior design project for school and I am inquiring if I can use the picture located at http://www.mdsupport.org/library/cvs.html describing the Resting Point of Accommodation in my Senior Design documentation. The document will not be published and will be purely for academic use. Please let me know as soon as possible

Thank You,
Geoffrey Jean

Reply | Forward

show details Jul 27 (6 days ago) | Reply | ▶

Dan Roberts to Geoffrey

Geoffrey,

You may use the image for the purpose you described. Good luck with the project.

Dan Roberts, Director
Macular Degeneration Support
email: director@mdsupport.org
web site: http://www.mdsupport.org
- Show quoted text -

Permission for Figure 19 and 27

☆ **John.Main@best-microcontroller-projects.com** to geoffrey.t.jean   show details Jul 28 (6 days ago)   ↰ Reply ▶

Hi Geoffrey,

Yes that's fine for you to use in documentation
as long as they are annotated with thier source .

Best Regards John Main

The Only Microcontroller C Programming Course
Showing You How To Make Your Own C Projects:

http://www.best-microcontroller-projects.com/c-start

\----
Get the Microcontroller ezine for
C programming tips, projects and more.
Visit http://www.best-microcontroller-projects.com/ezine
\----

On Tue, Jul 27, 2010 at 10:00 PM, <bestmicr@best-microcontroller-projects.com> wrote:
On Tue Jul 27 16:59:04 2010, the following results were submitted from the "Contact" on best-microcontroller-projects.com:

\----------
First Name: Geoffrey
E-mail Address: geoffrey.t.jean@gmail.com
Questions or Comments:: I am writing a senior design project and would like know if I can have permission to use the image of the data transfer from master to
slave in your I2C tutorial section found at http://www.best-microcontroller-projects.com/i2c-tutorial.html in my documentation. The document will not be published
and is being used for purely academic uses.
Please let me know as soon as possible.
Thank You,
Geoffrey Jean
Can I use your Comments on this site?: No

\----------
Best regards,
John Main
best-microcontroller-projects.com