University of Central Florida

Senior Design 1 Project Documentation

Group #15

3D LED Cube

Authors:
Luke Ausley
Joshua Moyerman
Andrew Smith

December 2, 2013

Contents

Chapter 1

Executive Summary

This senior design project documentation will outline in detail the process for the completion of Group #15's project - a 3D LED cube. This paper will describe the goals and objectives, specification and requirements, research, design, prototype creation, as well as test and evaluation of the 3D LED cube. Coming from a collective interest in the hobby/industry of lighting, the completed 3D LED cube will act as an aesthetically pleasing display, capable of creating a variety of dynamic and unique visual animations in three-dimensional space. Furthermore, this project will serve as a proof-of-concept, providing a broad-based exploration into an exciting technology: true 3D displays.

The culmination of this project - a functioning prototype - will be built to a specific set of standards. Operating at a resolution of 10x10x10 voxels (three-dimensional pixels), the cube will utilize 1000 LED's and operate based upon the concept of persistence of vision (rapid blinking to give the appearance of a solid light source). By maintaining a minimum LED refresh rate of 100Hz, the cube will be able to display seemingly continuous graphics to the observer at a minimum animation refresh rate of 25Hz, while operating in full 24bit color space, each LED capable of producing 16.8 million individual colors. The cube size will be within 2x2x2ft, contained in an outer casing which will be slightly larger. The casing will consist of acrylic sheets to protect the LED cube without diminishing the visibility of the cube. The entire system will be powered entirely from a common AC wall outlet through the use of a commercial grade high current power supply.

Although there are multiple avenues to accomplish the primary tasks in the design of an LED cube: software, control hardware, and wiring hardware, detailed research directed the group to the utilization of specific technologies. The control of the cube from the software side will be handled by a custom software suite which will push instructions to two custom printed circuit boards (PCBs) containing an embedded processor as well as an FPGA on one, and LED drivers and MOSFETs on the other. By multiplexing, the control structure will be able to individually control each LED by manipulating the current supplied to each individual LED with the utilization of LED drivers and MOSFETs. This control structure and constant-current modulation, combined with the concept of persistence of vision, is what will allow the individual control of 1000 LED's with complicated and fast-paced animation rendering while sparing a more complicated control design.

Chapter 2

Project Description

The 3D LED display is born from the desire to design and build a high quality RGB LED cube. A cube comprised of RGB LEDs will allow users to display a broader color palette. The 3D LED display not only encompasses the physical cube itself, but the creation of a user friendly interface for animating and modifying the content displayed on the cube. The user interface will consist of a computer communicating to a firmware controller within the LED cube via an Ethernet connection. The communication will allow the display of both static and interactive graphics and animations.

In addition to presenting static and dynamic graphics, the 3D LED display will also be capable to respond to external stimuli such as audio signals, accelerometer input, etc. The operator will have dynamic control of the 3D LED Display by utilizing multiple control mechanisms; from either a laptop computer or selection of preprogrammed animations. These unique methods of communicating with the display will allow the user to accomplish a variety of tasks with the 3D LED cube such as presenting text, interactive graphics, or even video games in 3D.

2.1 Motivation and Goals

The 3D LED display is of particular interest to the group due to the desire to experiment with novel ways of driving and interacting with large RGB LED cube displays. Given the fascination of the group members with the topics of both LED matrices and LED cubes, we chose our Senior Design project vectors to explore the concepts related to the operation of 3D LED cubes, improving upon methods used in the past while incorporating new and unexplored elements to the technology. The 3D LED display includes a variety of design aspects that appeal specifically to each group member. The electrical design, embedded system development, and software development provide well-defined areas of interest to each group member.

2.2 Objectives

Aside from the general direction of the project as defined by the description and goals, specific targets have been listed for the successful completion of the project. The primary intention of the project is to complete a 3D LED cube capable of displaying visual animations. This will be accomplished by designing a custom software suite to handle the input from the user and relay animation information to a custom PCB containing an FPGA and embedded processor. The FPGA must be able to decode the information from the processor and translate it to signals that specifically control both the frequency, color, and intensity of light on each of the 1000 LEDs that comprise the cube.

The physical structure of the cube will be sturdy, with a professional appearance while maintaining both mechanical and electrical integrity. The cube will be housed inside an acrylic case, allowing for the protection of the sensitive LED's while maintaining high visibility from any angle outside of the cube. Each RGB LED will be correctly soldered, operating in full 24 bit color. The cube will be powered from a common wall outlet, operating at 120V DC with a maximum current of 6A. While the project will be completed with a budget of $750, a flexible maximum ceiling of up to $1,200 will be in place, providing for any unseen costs or further capabilities the group may choose to add. The entire cube will function as one physical unit, with all PCB's and circuitry contained within the cube's structure.

While each group member will be assigned responsibility for the various aspects of design, each member will acquire proficiency in the cross-disciplinary skills required for the design and creation of the 3D LED cube. These skills include software design (programming), hardware design (PCB/FPGA design), and circuitry hardware (soldering/power). In addition, a familiarity with all relevant technology such as LEDs, microcontrollers, and LED drivers will be attained by each member.

2.3 Requirements and Specifications

The specifications listed below will organize and describe more succinctly the requirements that have already been broadly mentioned in the objectives of the project. Furthermore, how these specifications were arrived at will be expounded upon.

- Cube Resolution of 10x10x10
- Cube Size of 2x2x2ft
- Overall Size of 2.5x2.5x3ft
- Minimum Refresh Rate of 100Hz
- Minimum Animation Frame Rate of 25Hz
- 24 bit RGB Color Space (16.8 million colors)
- Nominal Operating Voltage: 120V
- Maximum Operating Current: 6A
- Operating Temperature: 50-100F

- Operating Humidity: 10%-80%

The size of the cube was determined based on several factors. The number of LED's in each direction (10), the spacing between each LED (pitch), the size of the housing for the physical control hardware at the base, and also the spacing between the cube and the outer casing.

In order to operate by persistence of vision, research determined that a refresh rate of 100Hz would meet the requirements for blinking an LED quickly enough so that it appears to be constantly on by a viewer. This is the basis for the LED cube, as if each LED were actually on 100% of the time, the power requirements would be much too great, and each of the 1000 LED's would require an individual control line. By multiplexing, made possible only through persistence of vision, the design can be accomplished with only 310 control lines. An animation refresh rate of 25Hz allows each animation to be shown at a standard frame rate used in most TV and cinema applications. Below this frame rate, the animations will not appear smooth, whereas above this frame rate, a more robust embedded processor will be required - with the increase in frame rate providing only marginal benefits.

This design opted to use high-quality commercially packaged LED's, containing both red, green, and blue diodes. While this increases the electrical demands and complicated the control and design structure, the benefit is substantial when compared to a simple single-color LED cube. By utilizing these diodes, each LED in the cube will be able to generate 24bit color, while also varying individually in intensity - accomplished via pulse width modulation (PWM). Furthermore, the quality of the LED's allows a high level of intensity to be generated with only moderate levels of current - allowing the 3D display to have an impactful presence even in a brightly lit room.

The purpose of the LED cube - to display animations to an audience - implies that the cube should be operable in common situations. This drove the group to design the cube to be powered from a wall outlet. An alternating current (AC) to direct current (DC) converter, much like one used for a laptop, will act as a power supply for the system. This aspect of the design is also closely related to the LED selection. Although it is expected that this 3D display will be operated in climate-controlled environments, its operating temperature and humidity should include a buffer. For this reason, the LED cube was designed to have an operating temperature of 50-100F and 10%-80%. This will allow the cube to operate with electrical integrity in more challenging environments, if necessary.

Chapter 3

Research Related to Projects and Products

The 3D LED cube is a popular project in both the hobby and commercial markets. A proper analysis of existing projects will allow for successful research and development. Looking at similar products allows for comparisons of feature sets for consideration, as well as potential limitations and opportunties for improvement. Research into related products and the components they utilize is vital to beginning the design process. Some parts may be acceptable in one application but may not scale well to larger projects. Reviewing projects from multiple vendors that span a large scope will provide a rich set of components to select between or to begin research further into. With an adequate base of components and feature sets for selection, design decisions can begin to be made with selected components that fit the project requirements.

3.1 Existing Projects and Products

There are a number of projects - both commercial, academic, and hobbyist - that have developed a 3D LED cube design. These projects range from small cubes (4x4x4) to much larger ones (16x16x16). By examining existing products and projects, a foundation of research can be established to allow improved design.

3.1.1 Jameco Electronics LED Cube Kits

Jameco Electronics (jameco.com) offers both a DIY 4x4x4 and 8x8x8 LED cube kit for \$69.95 and \$149.95 respectively. The kit includes the required PCB, Atmel (4x4x4) or Arduino (8x8x8) microcontroller, single-color blue LED's, transistors, resistors, capacitors, wire, batteries, and other assorted parts. Jameco cites the difficulty of the 8x8x8 project as "Advanced" with a build time of several days as opposed to "Intermediate" and 6 hours for the 4x4x4 cube. The kit instructions provide several useful tips for creating the lattice structure out of wire. Jameco also provides sample code for programming the Arduino. Both of these resources will be used for reference in the prototyping stage of this project. Figures 3.1 and 3.2 below show the final result of the 4x4x4 and 8x8x8 blue LED cubes.

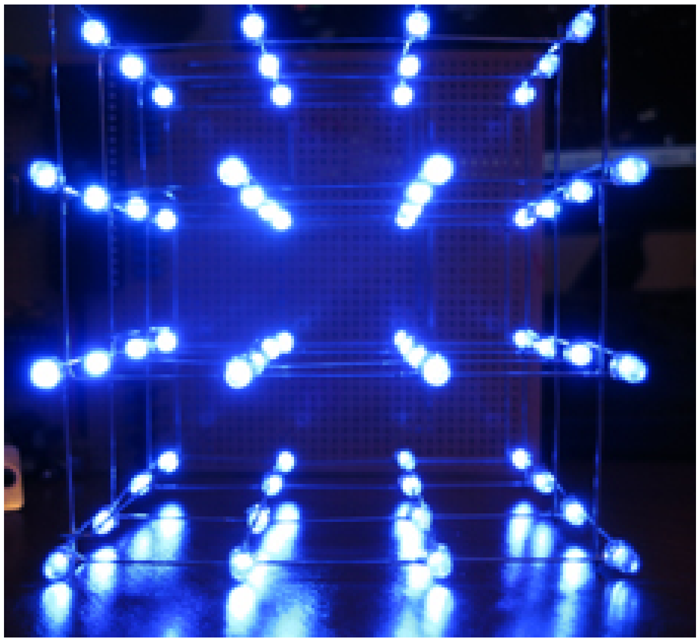


Figure 3.1: Jameco Electronics 4x4x4 DIY Blue LED Cube
Printed with Permission, Courtesy of Jameco.com [1]

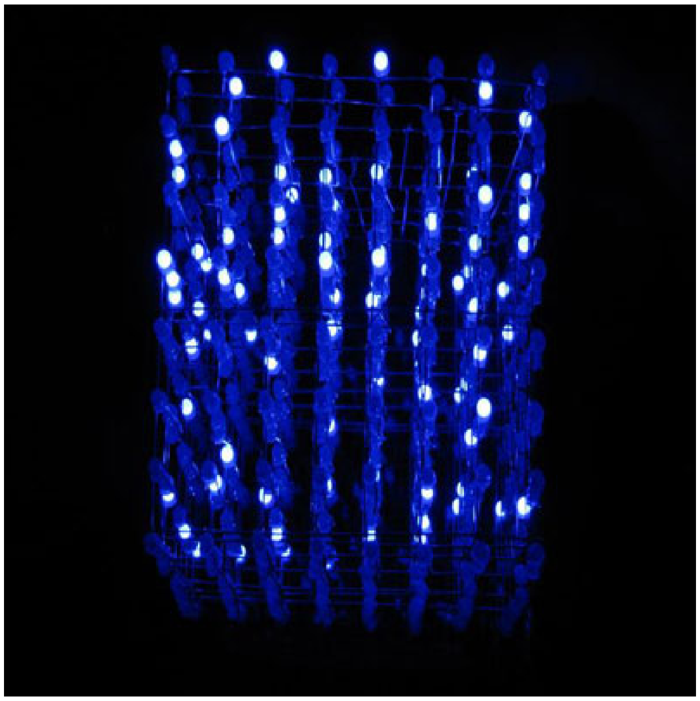


Figure 3.2: Jameco Electronics 8x8x8 DIY Blue LED Cube
Printed with Permission, Courtesy of Jameco.com [1]

3.1.2 All Spark 16x16x16 RGB LED Cube

The All Spark cube is an example of a cube with a more intricate design than the one proposed in this project. The cube's construction is relatively standard, using thick 16 gauge wire both as the structure for the cube and the electrical connection. The interesting part of the All Spark team's hardware design was their choice to use 16 individual custom microcontrollers, one for each panel. The microcontrollers are managed by the "power plant" of the system, an Arduino Mega 2560. This design utilizes an RS232 interface from computer and RS485 for cross-panel communication. The All Spark team's project documentation doesn't include information about their software development - although they mention development of an open source LED cube software suite. Figure 3.3 below shows the encased 16x16x16 cube that was completed.

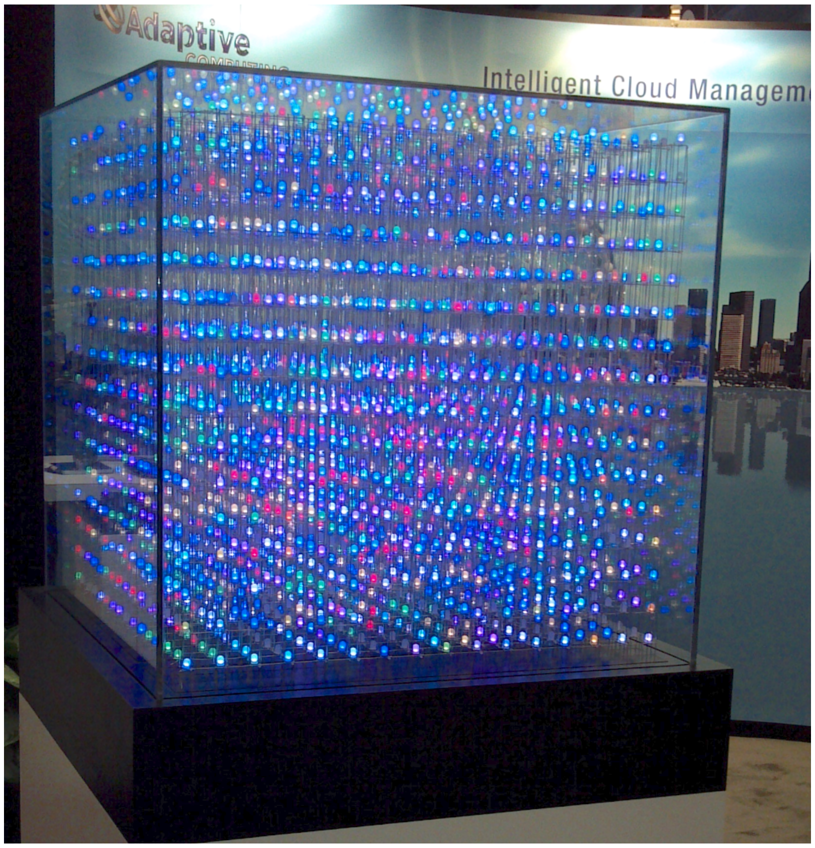


Figure 3.3: All Spark 16x16x16 3D RGB LED Cube
Printed with Permission, Courtesy of AllSparkCube.com [2]

The same conceptual process for controlling the LED's in the 16x16x16 cube design is utilized in our project. The cube uses multiplexing to generate a 3D image by displaying 2D panels in rapid succession - too quickly for the eye to notice. Pulse Width Modulation is used to control the color of each diode. This is done by varying the duty cycle of each individual red, green, and blue diode. A higher duty cycle (diode is on longer than it is off) corresponds to a brighter LED of that specific color. By varying the duty cycles, and correspondingly the brightness of each LED color, the microcontroller is able to display any color from the LED.

3.1.3 Instructables 8x8x8 Blue LED Cube

Instructables lists a very detailed design process for their single-color LED cube. Although the control diagram of their project differs from the one presented in this paper, the steps to their design will serve as a useful resource for smaller problems encountered that other projects may not have mentioned. Instructables attributes 50% of the workload to the hardware design and 50% to software design. Figure 3.4 below shows the result of the Instructables blue LED cube.

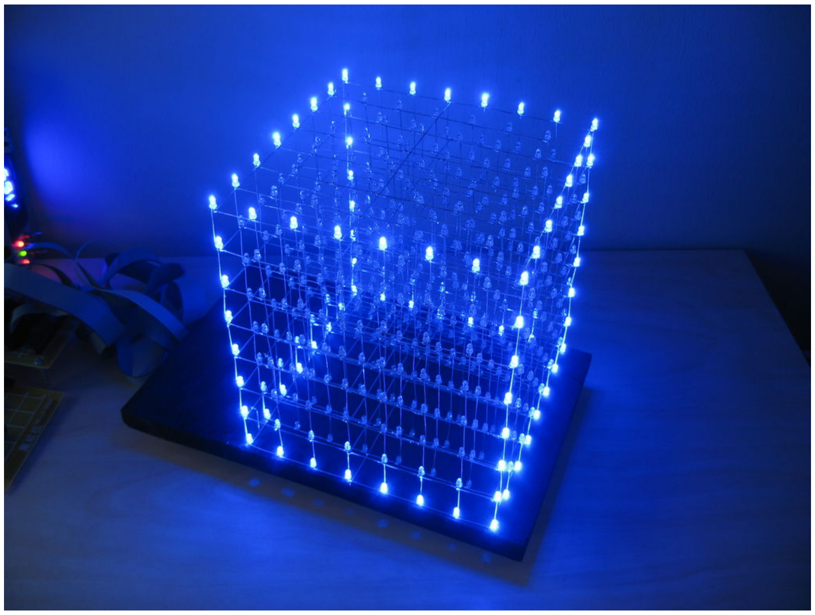


Figure 3.4: Instructables 8x8x8 3D Blue LED Cube
Printed with Permission, Courtesy of Instructables.com [3]

The Instructables design uses an Arduino ATmega32 microcontroller with a 14.7456MHz clock rate, as it is divisible by all popular RS232 popular baud rates. They chose to use a 74HC138 3 to 8 decoder and 8x 74HC574 8 bit latch's for multiplexing.

They use a 5V computer power supply in order to maintain a high current. The 8x8x8 design will draw 64 LEDs * individual diode current, if all diodes are on. The 10x10x10 design in this paper will have more stringent power supply requirements with 100 LEDs * 3 colors * individual diode current, or 470% of the maximum current required by the 8x8x8.

Instructables provides some basic open-source code in C as well as pseudocode and explanations for accomplishing the control of simple, specific animations. This may prove a useful starting points for building the animation playbook to choose from in the 10x10x10 design that will be presented.

3.1.4 HNTE 8x8x8 RGB LED Cube

The RGB LED cube featured by How Not To Engineer shares some similar aspects to the proposed design for the project in this paper. Their design utilizes 12 STP16 LED drivers, 4 for each color: red, green, and blue. Each set of 4 drivers provides 64 bits, for each LED in the 8x8 panel. Multiplexing allows this control architecture to work. A 32bit chipKIT UNO is used to control the STP16 LED drivers by providing a serial output and using bit angle modulation (BAM) to fade each LED color, providing a full color range for each LED. Bit angle modulation (BAM works by assigning a different bit to each area of the LED driver output waveform. The on or off status of each bit determines the length of the pulse to be send to the LED, in essence controlling the LED color with a value. This method is similar in concept to pulse width modulation. Figure 3.5 gives an illustration of 8 bit BAM in operation.

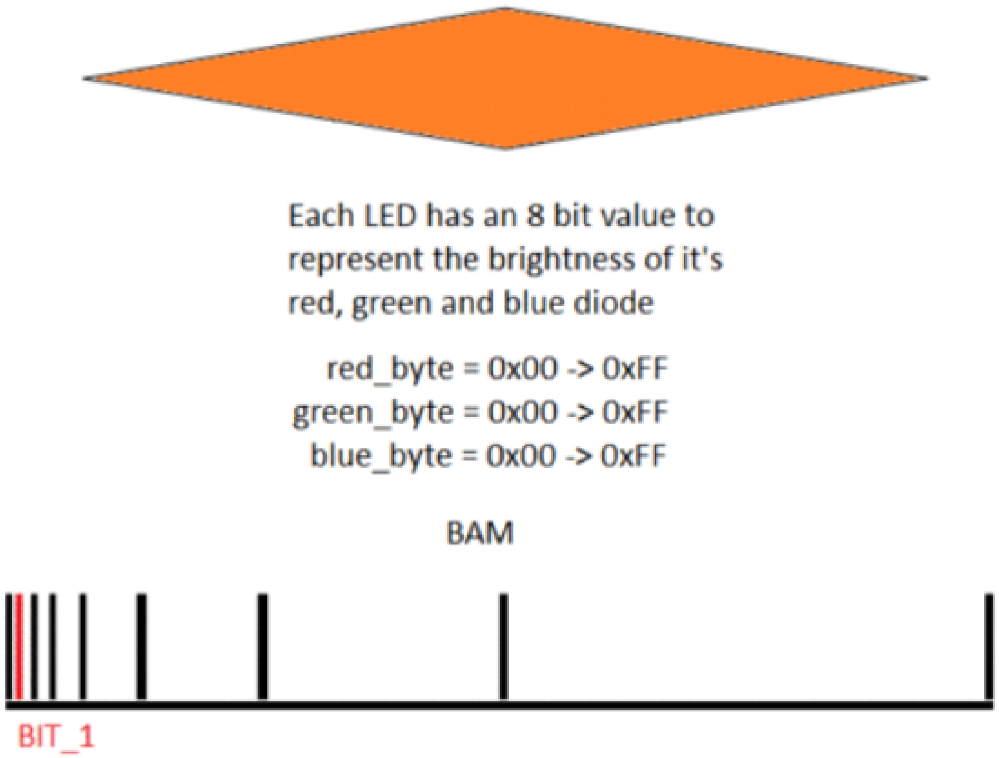

Each LED has an 8 bit value to represent the brightness of it's red, green and blue diode
red_byte = 0x00 -> 0xFF
green_byte = 0x00 -> 0xFF
blue_byte = 0x00 -> 0xFF
BAM
BIT_1

Figure 3.5: Bit Angle Modulation
Printed with Permission, Courtesy of HowNotToEngineer.com [4]

One impressive aspect of the HTNE project was the fact that their cube was built using minimal processing power, the design this group will utilize will likely involve more robust computing. Figure 3.6 shows the completed 8x8x8 RGB LED cube built by HTNE.

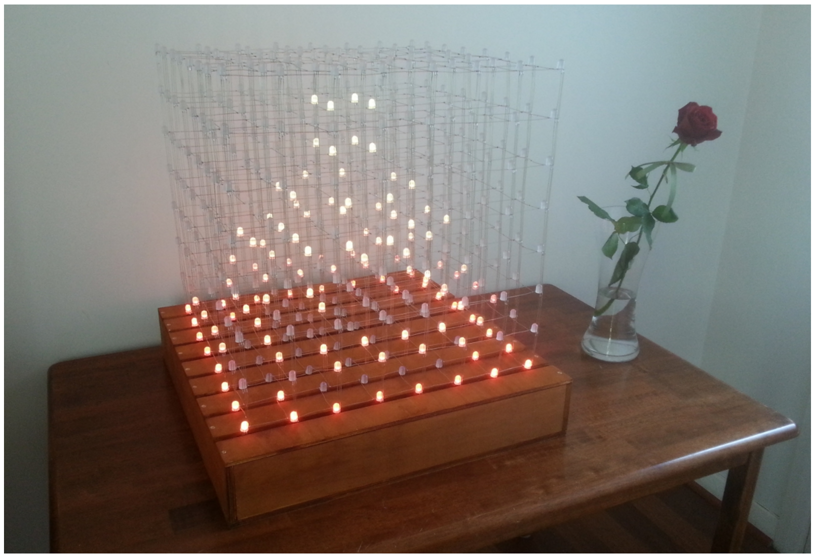


Figure 3.6: HNTE 8x8x8 3D RGB LED Cube
Printed with Permission, Courtesy of HowNotToEngineer.com [4]

3.2 Component Research

The selection of individual components for the 3D LED cube is the most crucial step in the hardware design process. All further design decisions are governed by the specifications and limitations of the components chosen. Comparisons are made between different components with decisions made based on strengths, weaknesses, and the ability to contribute and conform to the project's perfomance goals and standards.

3.2.1 Light Emitting Diode

The choice of LEDs will affect all other design decisions, and will therefore be considered first and foremost. The market for LED components varies widely in cost, quality, and branding. LEDs are composed of a PN junction that produces light when forward biased, packaged in a translucent plastic casing with metal connection points for the anode and cathode. One LED PN junction is capable of producing only one color of light. In order to create a display with more than one color, multiple different colors are necessary. Due to the additive nature of of how human vision perceives colored light, the colors necessary to produce a full color spectrum are red, green, and blue.

To design a multi-color display, it is necessary to include the above three colors into each voxel. This is commonly done using two different methods. The first method is to use three separate LED packages, each emitting a specific color. The second method is to use a single LED package containing three separate PN junctions, one for each color. Each method has benefits and disadvantages which will be discussed below. Figure 3.7 shows three separate red, green and blue LEDs whereas figure 3.8 illustrates the single LED package containing red, green and blue diodes.

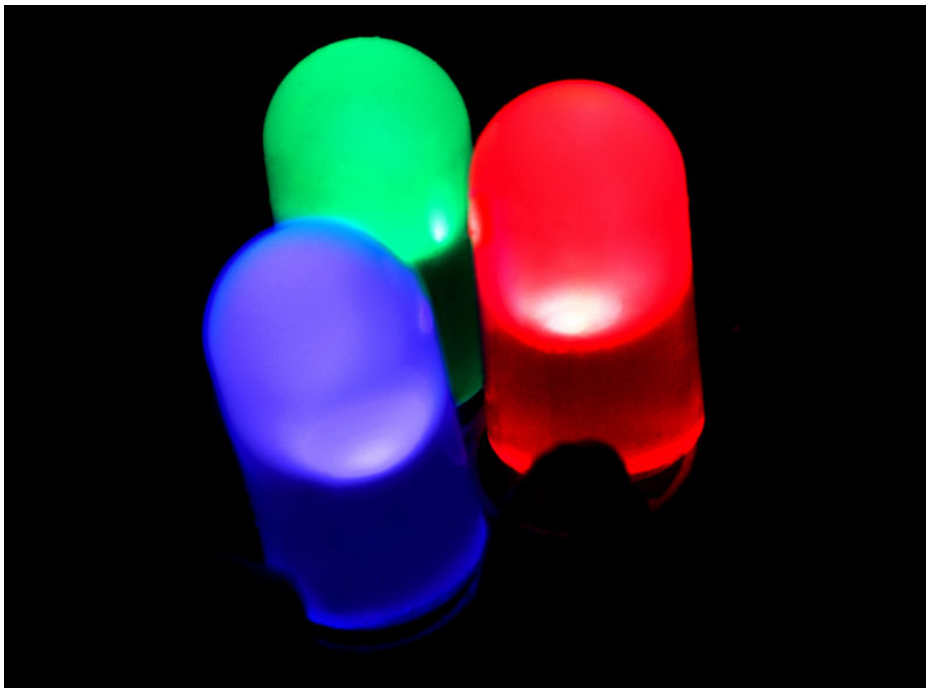


Figure 3.7: Discrete RGB LEDs
Printed with Permission, Courtesy of Wikipedia.org [5]

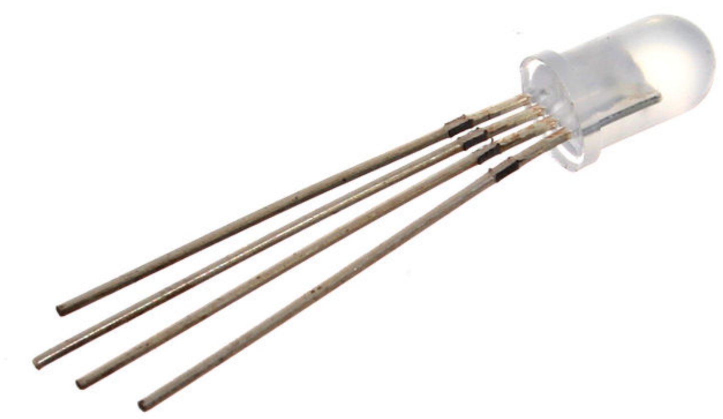


Figure 3.8: Single Package RGB LED
Printed with Permission, Courtesy of Wikipedia.org [5]

Three Discrete LEDs:

Advantages	Disadvantages
• Better Thermal Characteristics	• Extra Wiring
• Greater Reliability	• Increased Cost
• Wide Selection of Packaging	• Decreased Color Blending

Single Package LEDs:

Advantages	Disadvantages
• Simplified Wiring	• Poor Thermal Dissipation
• Decreased Cost	• Lower Reliability
• Better Color Blending	• Decreased Package Availability

Comparing the two options, using a single LED package with three individual PN junctions will be used for the construction of this project. The choice to use a single package will greatly reduce the construction cost and time, the complexity of the project, and will increase the visual appeal of the cube by allowing for much better color blending and saturation.

Commercially available LEDs are offered in many various package sizes, and formats. As with most electronic components, surface mount and through hole (leaded) package variants are available. Surface mount device LEDs, commonly abbreviated SMD, are flat chips that emit light in approximately a 180° field of view. These LEDs commonly have no lens, and are intended to provide illumination, rather then illuminate itself. Through hole LEDs are commonly shaped like a small plastic bullet, and contain two or more leads that extend out of the bottom of the package and traditionally go through a printed circuit board. This plastic package while designed to obscure as little light as possible, actually will illuminate with the junction, this package can be seen from a greater than 180° angle is what the prototype cube will be constructed with.

LED quality can be measured in two very separate but important factors, efficiency, and color wavelength. During construction of LEDs, every effort is made to produce the highest quality product. During quality control, a process called binning occurs where each LED is tested and measured to ensure that it matches the quality standards. The higher the efficiency, and the more accurate the color wavelength, the higher a price the LED will command. Also, not all suppliers list the actual specifications on their product when retailing them. Due to these concerns, the group will choose to go through an authorized distributor of a the chosen LED manufacturer's products, as opposed to a sale or bidding site like eBay and Amazon.

LEDs are inherently temperature and voltage sensitive devices. LEDs, when biased at a higher temperature emit less light. In larger LEDs, this effect causes drastic reductions in lumen output. Due to the low power nature of the LEDs used in this design, and the operating conditions of the project, thermal efficiency should not affect the brightness of the LEDs used.

LED Choice

The decision for which LEDs to utilize in the 3D cube is absolutely the most critical choice to make. As the LEDs are really the principal component of the LED cube, the group chose a more difficult yet more rewarding path. Rather than utilizing commercial-off-the-shelf (COTS) diodes, custom LEDs were specified and ordered. This choice allowed a much higher degree of control over the rest of the design components. As the majority of design hinges upon the specifications of the LEDs, an initial custom design allowed the remainder of the design choices to be made with significantly more ease than if the LEDs presented more extensive design challenges and downsides.

20mA forward drive current
100mA peak forward rating (1/10th Duty Cycle)
-40° to 100° C Operating Temperature
5V Reverse Voltage Rating
1000V Electrostatic Discharge Threshold

Table 3.1: LED Specifications

3.2.2 LED Wiring

Several different architectures exist for wiring and powering large numbers of LEDs. A few of the most common technologies will be covered and discussed in detail, and a choice will be made as to which architecture will best suit the 3D LED cube.

Individual Wiring

The simplest, and most straight forward way to control a quantity of LEDs is to connect each LED to an individual control line of a microcontroller or similar device. The other pin of the LED will be connected to ground. Using this method, one control line is required for each LED to be controlled. To power and control the LED cube with 1,000 LEDs at three colors each, 3,000 control lines would be required. This would be extremely complicated to wire and control, and is a less than ideal choice.

In order to turn on all of one color on at once, 1,000 times the current of each LED are required to be supplied by the control circuitry. If all colors are to be turned on in order to produce white, this increases to 3,000 times the current of one LED. This would require being able to supply approximately 60 amps. This would be prohibitive for transmission at such a low voltage.

Individual wiring does provide the advantage that the full brightness of the LEDs is available. There is no scanning or refreshing of the LEDs that would detract from the brightness available, a definite plus. Another advantage of individual wiring is that without scanning through the LEDs, a lower performance processor can be used.

Multiplexing

One of the most common methods for driving large arrays of LEDs is to use a multiplexing scheme. This is the method that is used for most television and large format screens. Multiplexing controls large two dimension arrays of LEDs with less control lines than wiring each of the LEDs individually. An example of how multiplexing is wired is shown below in Figure 3.9

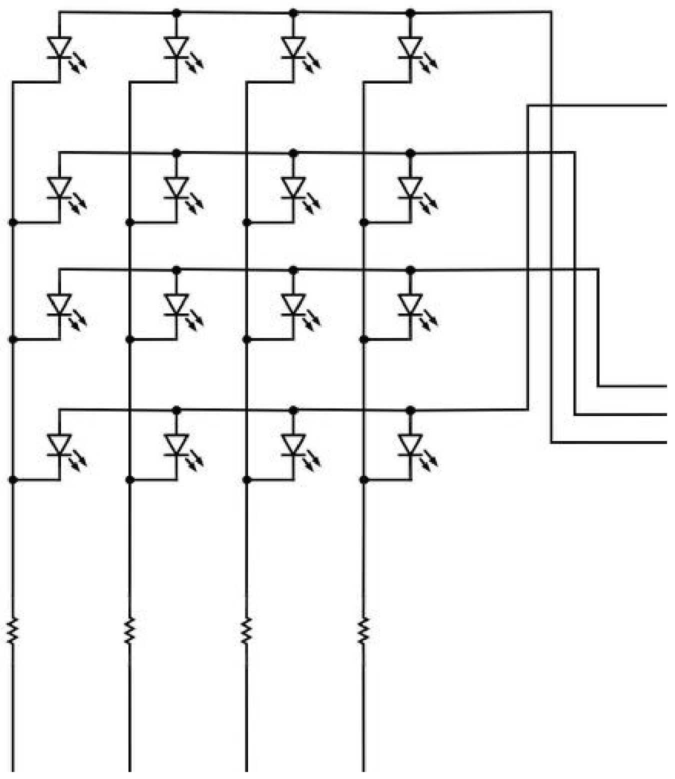


Figure 3.9: Multiplexing Diagram
Printed with Permission, Courtesy of Wikipedia.org [5]

In multiplexing, all of the anodes of a row are connected together and the cathodes of a column are connected together. By applying power to only one row, and grounding only one column, an individual LED can be turned on. By applying power to multiple rows, and grounding only one column, all of the LEDs in one column can be controlled. After this column has been controlled, the next column can be grounded, and its appropriate rows be powered.

This cycle is repeated for each of the columns in the display, rapidly enough such that the persistence of human vision perceives that each of the LEDs that was turned on at one point, is constantly on. At a rate of approximately 50Hz, most people see the LEDs as constantly on, however some detect flicker. After approximately 100Hz, the scanning appears to be smooth.

Using multiplexing, the number of control lines is drastically reduced. For an array of ten by ten LEDs only twenty control lines are required, as opposed to the one hundred control lines that would be required by wiring them individually. Wiring the cube as an array of ten LEDs by three hundred (one hundred by three colors), 310 control lines are required, a drastic reduction from the 1,000 required by using the individual wiring method. Multiplexing also allows for easier wiring of the cube, anodes can be connected by plane, and cathodes by column or row, helping to provide structural support.

Charlieplexing

Charlieplexing is a technique for driving large numbers of LEDs with a small number of control lines. This technique was first proposed by Charlie Allen at Maxim Integrated. Charlieplexing, unlike individual wiring, and multiplexing, requires three state logic, on, off, and high impedance. This high impedance mode is usually realized by setting the control line to an input. With this third state, LEDs can be hooked up in a pattern as shown by Figure 3.10.

By flashing through each of the necessary LEDs that need to be on rapidly enough, as with multiplexing, the entire display can be lit up at once. Unlike multiplexing however, only a few LEDs are on at a time when using charlieplexing, in some cases only one LED. This drastically increases the scan rate that is required to accomplish flicker free display. The current requirements of charlieplexing decrease drastically though, as only a handful of LEDs are on at the same time.

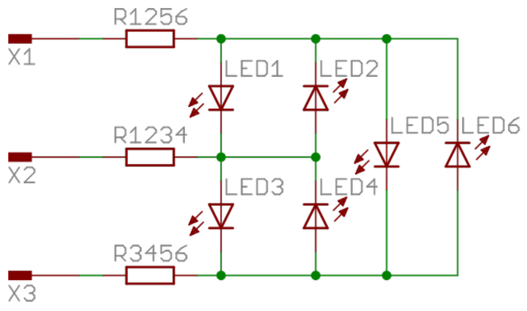


Figure 3.10: Charlieplexing Diagram
Printed with Permission, Courtesy of Wikipedia.org [5]

Using this method, a vast improvement on the number of control lines necessary is found. The number of LEDs, L, that can be driven by a number of control lines, n, is given by the equation: $L = n^2 - n$. Solving for n, it is found that $n = \frac{1+\sqrt{1+4L}}{2}$. This architecture results in the need of only 55 control lines to drive three thousand LEDs. However, the nature of the LEDs chosen, common cathode, does not allow for charlieplexing. This rules out the ability to use charlieplexing, however it would be a novel idea to consider a LED cube with this wiring method.

Wiring Choice for 3D LED cube

Given all of the advantages and disadvantages stated with individual wiring, multiplexing, and charlieplexing, the group has found that multiplexing would be the best choice for creating an LED cube. This will simplify our wiring efforts for constructing the cube, and will allow us to use the simplest, most efficient control system for powering the cube. Given that the cube is ten voxels in each direction, it makes the most sense for the group to wire each column of cathodes as one row, and each plane of anodes as the rows. From a control and programming perspective, this will make the cube appear as if it were a two dimensional display with ten rows, and three hundred columns. Viewing the cube this way will also simplify the wiring, as each LED's cathodes and anodes can be used as supporting structures.

3.2.3 LED Drivers

LEDs, being PN junctions, are non linear devices which require specific care to ensure that the device is operated within its physical limits. PN junctions have an exponential increase in current dissipation with a linear increase in bias voltage, making the current an extremely important factor for powering the LEDs. For this reason, the 3D LED Cube uses constant current technology when driving the LEDs.

This ensures that the thermal and physical characteristics of the LEDs are not passed, and breakdown does not occur.

There are numerous LED drivers available on the market, targeting specific segments and applications, some meant for driving only a small number of LEDs, some meant to power very large LEDs, and some meant to power high quantities of LEDs. For the 3D LED cube , the group is looking to find a LED driver that can handle both adequate current, and the number of outputs necessary, while being available in an easy to assemble package.

Most LED drivers have the same basic architecture. Typically, some form of a back-plane serial connection provides input data to the registers within the chip. An internal oscillator drives a counter with which each register is compared with to produce the outputs necessary. When the register equals the counter, the output is switched to a logic on. The serial nature of these LED drivers often run in the mega-hertz range, requiring a high speed link in order to be most effective. A communication of this nature may best be suited to a FPGA.

The output of LED drivers come in two major variants, open drain/collector, and standard TTL. An open drain works by connecting the output of the chip to the drain of the internal MOSFET, or collector of the internal BJT, depending on what the IC is built with. The open drain can then be used to sink current, or connect the output to ground, allowing a large range of voltages to be used with the output.

LED Driver: Texas Instruments TLC5940

The TLC5940 is one of the older devices in Texas Instrument's LED driver lineup. Operating 16 channels at constant current, it can supply either 0-60mA (Vin < 3.6V) or 0-120mA (Vin > 3.6V). The maximum current for all 16 channels is controlled with an external resistor. The maximum output voltage for the LED driver is 17V. With a 12 bit brightness control (BC), the TLC5940 is able to step brightness in 4096 increments, using pulse-width modulation (PWM) allowing the LED cube to display RGB color. 6 bit dot correction is included, allowing the LED driver to adjust the brightness variations both between different channels and also different LED drivers, so that the brightness is consistent for each LED. The data for the dot correction is stored in on-board EEPROM.

Further, the TLC5940 includes two error notification circuits: LED open detection (LOD) and thermal error flag (TEF). LOD indicates a broken or disconnected LED, and TEF indicates overheating. All communication with the driver (BC, dot correction, error retrieval) is done via serial connection. Finally, the TLC5940 is available in 28HTSSOP, 28PDIP, and 32VQFN packages. The packages a device are available in simply affect the shape of the device and power dissipation due to that shape. The performance of the device will be the same across packages. Figure 3.11

below shows the block diagram for the TLC5940 LED driver.

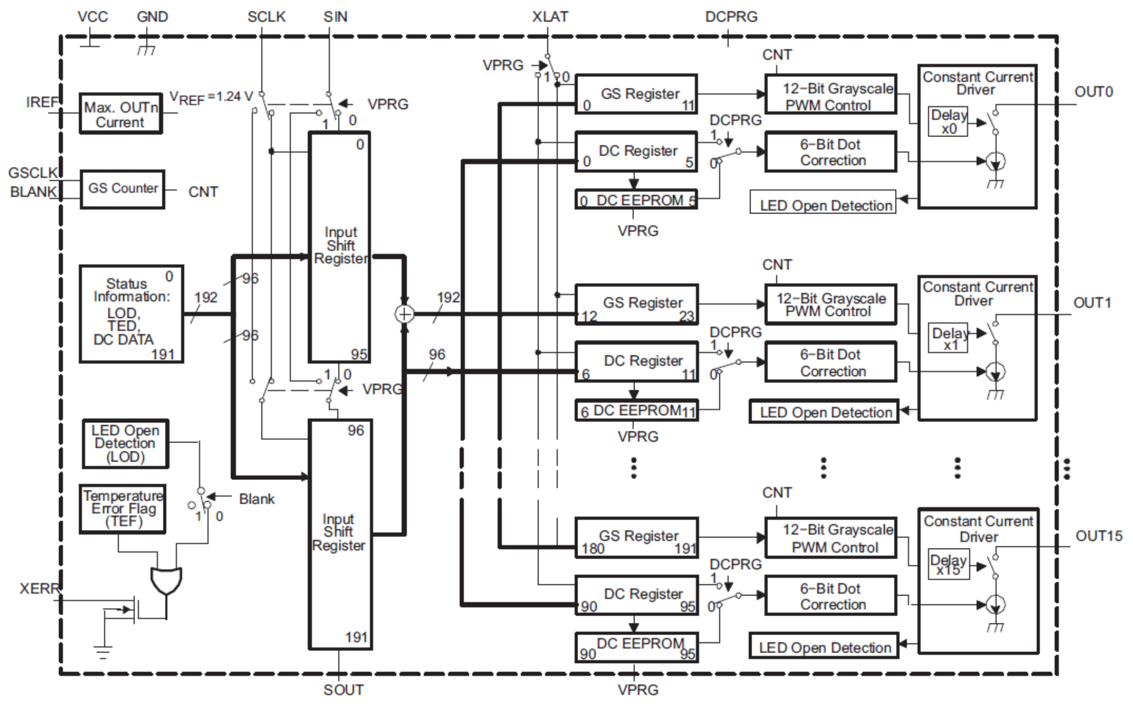


Figure 3.11: TLC5940 Block Diagram
Printed with Permission, Courtesy of Texas Instruments [6]

LED Driver: Texas Instruments TLC5948A

The TLC5948A is a newer, more robust version of the TLC5940. Similar to the TLC5940, it operates 16 channels at constant current, supplying either 2mA-45mA (Vin < 3.6V) or 2mA-60mA (Vin > 3.6V). Where it outshines its predecessor is with 16 bit brightness control (BC), stepping in 65536 increments. Further, it has 7 bits of dot correction (DC), allowing for a more precise constant-current correction between channels and separate devices. Another interesting feature of the TLC5948A is its 7 bits of global brightness control. TI cites the constant current accuracy of this driver at $\pm$.6% (typ), $\pm$2% (max) channel-to-channel and $\pm$1% (typ), $\pm$4% (max) device-to-device.

The TLC5948A has a data refresh date at 33MHz, a power saving mode to minimize Vin current, and expanded error notifications. It's six error flags are LED open detection (LOD), LED short detection (LSD), output leakage detection (OLD) reference current terminal short flag detection (ISF), pre-thermal warning (PTW), and thermal error flag (TEF). The error notifications are retrieved via a serial interface port. The operating temperature of the TLC5948A is -40 C° to +85 C°. However, its maximum LED voltage is less than the TLC5940 at only 10V. It is available in 24HTSSOP and 24SSOP packages. Figure 3.12 below shows the block diagram for the TLC5948A LED driver.

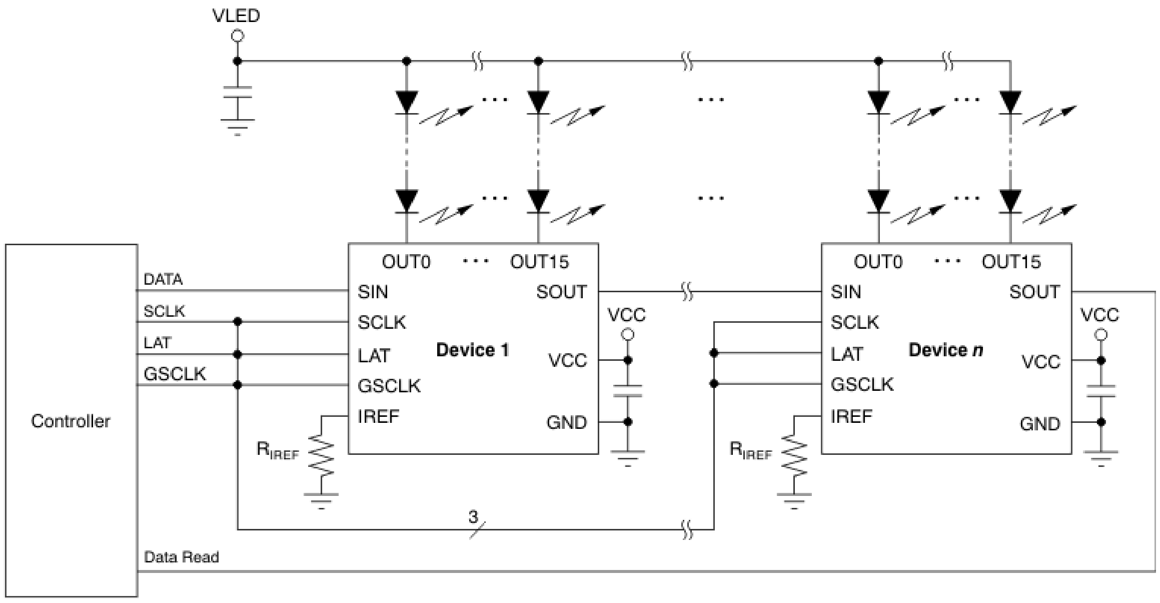


Figure 3.12: TLC5948A Block Diagram
Printed with Permission, Courtesy of Texas Instruments [6]

LED Driver: My-Semi MY9161

The MY9161 by My-Semi Incorporated is another viable LED driver. Operating 16 channels at constant current, it accepts an input voltage between 3.3V and 5V. The outputs are rated at 17V, providing 55mA of output current (at 5V input) to a string of LEDs. This high output current is required to drive the LED's at an appropriate level of brightness. The output current is further programmable with an output resistor. LED dimming (allowing RGB color) is accomplished with a 25MHz DCK. The MY9161 has a 16 bit transparent latch, 16 bit shift register, and 4-wire serial interface (DI, DCK, DO, ENB). The data out (DO) allows the drivers to be cascaded together, a requirement for an LED cube design. The MY9161 is also specified to have a constant-current accuracy of ±.7% channel-to-channel and ±.1% device-to-device, with a ±.1% output current regulation capacity and 50ns transient output response time. The driver operates over a temperature of -40C° to +85 C°, more than enough for the LED cube application. The MY9161 is available in SOP24, SSOP24, and QFN24 packages. Figure 3.13 below shows the block diagram for the MY9161 LED driver.

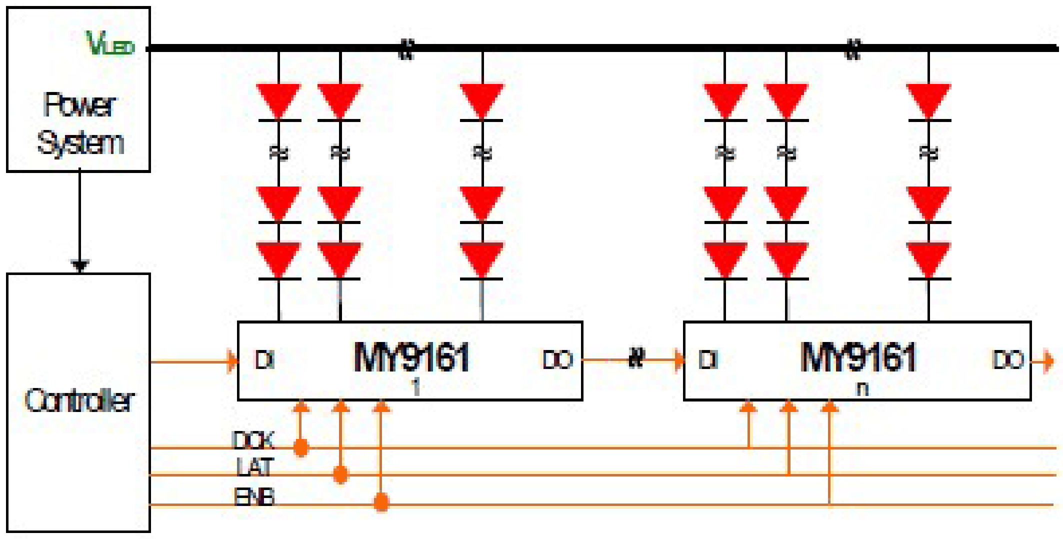


Figure 3.13: MY9161 Block Diagram
Printed with Permission, Courtesy of My-Semi Incorporated [7]

LED Driver Choice

The Texas Instruments TLC5948A was the best option for this design. The primary selling point for this driver is its 16 bit brightness control. This will provide many more color variations than a smaller-resolution dimmer and greatly enhance the visual performance of the LED cube. The shortcomings of this driver compared to the others considered for this project ended up being moot. While it only has a 10V maximum LED voltage, this exceeds our requirements. Although it cannot provide more than 60mA of current (at 5.5V), the LEDs chosen for this project only require 50mA of current. The expanded error detection and 33MHz data refresh rate are some other less important reasons that contributes to the fact that the TLC5948A rises above its competitors for this application.

3.2.4 Embedded Processors

Nearly any device made today has some sort of a processor or microcontroller embedded within it. The question is often not whether or not there will be a processor, but which processor to use, and how powerful of a processor is needed. In order to communicate with both the users computer, as well as the FPGA for the drive circuitry, a processor with multiple communication lines will be required. The group wishes to use a processor that has good documentation and reference designs available, as well as low cost development software and hardware.

The embedded processor will be responsible for managing the operation of the LED cube. It is desired that the processor have a basic web page capable of controlling and creating simple animations, with complex animations to be rendered on the host computer. The embedded processor software will go through many revisions and updates, so in field updates are a must. The processor will also need to capable of providing new firmware to the FPGA, either by loading it directly, or by reprogramming the storage device on-board. Also, given the desire to store content on the LED cube itself, the processor will need to be capable of interfacing to an external storage mechanism. A full list of the desired specifications and capabilities of the processor are shown below in Table 3.2. These requirements will be used to aid the group in their choice of an embedded processor capable of controlling the project.

In field program updates (Bootloader desired, ICSP if not)
Package Availability (easy to solder)
Ethernet connectivity
DMA Controller
External Storage (MB to GB range)
Ample RAM
SPI (atleast one)
Low Power
3v3 desired
C or C++ compiler

Table 3.2: Desired Features and Specifications

Embedded Processor: Atmel Atmega 2560

Atmel's Atemga 2560 is a popular processor among hobby projects, due to its inclusion within the Arduino line of development boards. The Atmega 2560 is one of Atmel's largest 8 bit microprocessors, providing 256Kb of on-board flash storage, and 8 KB of RAM. The Atmega 2560 satisfies most of the project's requirements, however has the lowest performance of the candidates. This lower speed would require the group to do nearly all of its processing off board, and none of it on the embedded processor. A full list of the specifications of the Atmega 2560 are in Table 3.3 below. These are all of the specifications that the group finds would be useful to the project. Almost all of the specifications meet or exceed the groups design needs, however the processor is the slowest part chosen, and does not operate at 3.3V. This would require that the group introduce level shifting circuitry between the processor and the FPGA. Level shifting is a trivial task, however it would add more design then necessary and defeat the groups desire to keep the circuitry as simple and easy as possible.

8 Bit AVR Architecture
16 MIPS Throughput at a 16MHz Clock Rate
5V Device Operation
86 User IO Pins
5 SPI Interfaces
1 I2C Interface
4 UART Interfaces
External Ethernet Options Available
8 KB Static RAM
4 KB User EEPROM
256KB Program flash memory
Available in a TQFP100 Package
C and C++ Compilers available
Assembler available
Compatible with the Arduino Development Environment

Table 3.3: Ateml Atmega 2560 Features and Specifications

Embedded Processor: Microchip PIC24HJ256GP206A

The PIC24 by Microchip is a relatively powerful, 16 bit Harvard architecture processor. The processor is capable of running at 40 MIPS, more than double that of the Atmega chip listed above. This instruction rate is more than enough to be able to perform all of the processes required. The Microchip line of processors are compatible with their line of ethernet interfaces, via SPI and parallel interfaces. When using the Microchip line of ethernet interfaces, Microchip's entire TCP/IP stack can be used, which would greatly reduce the level of effort the group would have to put in to get a working solution up and running. The PIC24 also has a large amount of user IO and a large set of features as seen in Table 3.4

16 Bit MIPS Architecture
40 MIPS Throughput
3.3V Device Operation
53 User IO Pins
2 SPI Interfaces
2 I2C Interfaces
2 UART Interfaces
16KB Static RAM
256KB Program Flash Memory
Available in TQFP64 Package
C and Assembly Supported
Ethernet Interface & Library Code available

Table 3.4: PIC24HJ256GP206A Features and Specifications

Embedded Processor: TI AM3358

TI's AM3358 is a powerful 1GHz ARM Cortex-A8 processor capable of running an operating system such as Linux. Used within the BeagleBone line of development boards, community support and reference documentation is plentiful. The AM3358 is the most powerful processor that the group is considering, allowing for a throughput of up to 2000 MIPS with a 1GHz clock rate. This amount of power would allow the group to do vast amounts of processing on-board. The AM3358 has a vast amount of user IO available, and a large quantity of pins able to be used as general output. The AM3358 is a near perfect candidate for the 3D LED cube, however it requires an extremely complex design to use. The BGA package is one that the group is unable to work with in their own capabilities. A processor like this would require at least a six-layer circuit board, and would require the group having to hire the assistance of an assembly company. A full list of the features useful to the project is listed below in Table 3.5. These are not the full feature list of the processor, but only what the group finds relevant to the project.

32 Bit ARM Cortex-A8
2000 MIPS Throughput at 1GHz Clock Rate
1.1V Core Operation, 3.3V IO Operation
128 General Purpose IO Pins
2 SPI Interfaces
3 I2C Interfaces
6 UART Interfaces
2 Onboard 1Gbit Ethernet Interfaces
Available in 298 or 324 Pin BGA Package
Linux, Android, and Bare Metal Operating Systems Available.

Table 3.5: TI AM3358 Features and Specifications

Embedded Processor Choice

The group chose to use the PIC24HJ256GP206A as the embedded processor for the project. This processor encompassed all of the features that we needed in order to provide a pleasing display, while being easy enough to work with, as well as cost effective. The PIC24 is not the fastest of the processors that the group considered, but is more than powerful enough to be able to communicate with all of the peripheral devices, and provide a low latency refresh rate. The PIC24 also is available in a much easier to work with package, TQFP, as opposed to the BGA package of the AM335X. The PIC was chosen over Atmel's Atmega device due to a higher overall clock rate, and a greater wealth of manufacturer supported development libraries and example code.

3.2.5 MOSFETs

A MOSFET, or metal-oxide-semiconductor field effect transistor, is a three-terminal (source, gate, and drain) transistor. The MOSFET is the most popular field effect transistor in current use, overtaking the once-popular bipolar junction transistor (BJT). The channel of a MOSFET can carry either electrons (NMOS) or holes (PMOS), changing the direction of current flow through the MOSFET. A MOSFET has three operations in enhancement mode (referring to the increase in conductivity with an increase in carrier-adding oxide field) which are cutoff, triode, and saturation (also known as active). These modes are controlled with a voltage Vds applied between the drain and source of the transistor. Figure 3.14 describes the voltage-current characteristic of a generic MOSFET. Notice how at cutoff, the transistor is considered to be off, with the drain current Id = 0. As Vds increases, the MOSFET enters its linear (triode) region of operation, with a constant increase in current with voltage. However, at a point, the current saturates and the MOSFET is said to be in the saturation region. Past this point, the current will be regulated at this level - regardless of the magnitude of Vds. As shown in the image, different current-voltage curves are achieved based on different values of Vgs-Vtn. Vtn is a transistor-property, based on it's construction, but Vgs is a operator-controlled value. With these characteristics in mind, the MOSFET becomes a valuable current-controlling device. This is especially useful in the application of a 3D LED cube because a number of MOSFETs can be used to control which layer (x-y plane) of LED's is activated. Essentially, the MOSFETs will act as a layer switching device in the 3D LED cube design.

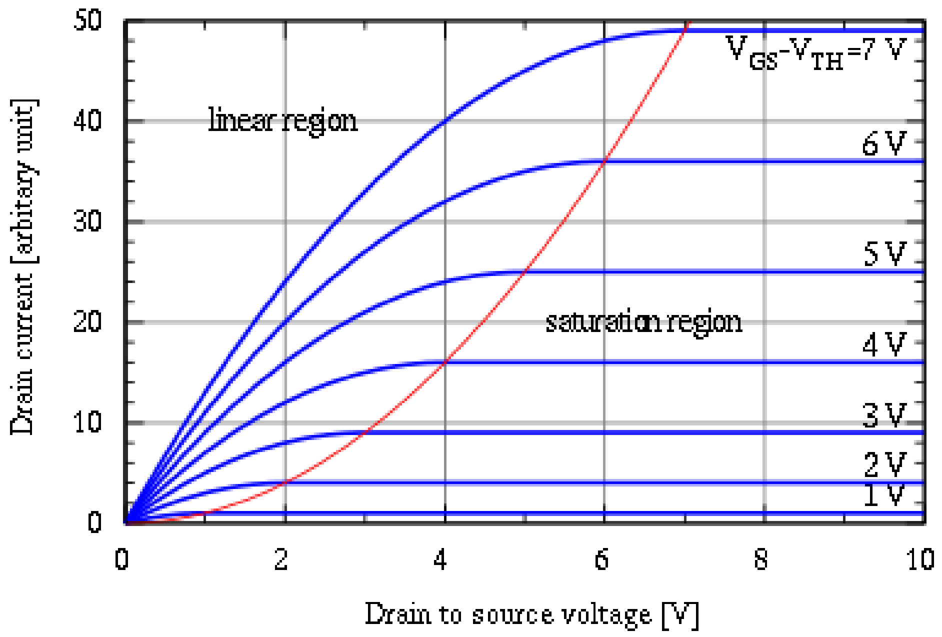


Figure 3.14: Generic MOSFET Current-Voltage Characteristics

MOSFET: Infineon Technologies SPD15P10PL G

The SPD15P10PL G by Infineon Technologies is a P-Channel MOSFET with a high continuous drain current of 15A, allowing the current regulation of the entire LED cube, even at maximum power. It has a drain-source breakdown voltage of -100V and a gate-source breakdown voltage of ± 20V. This far exceeds the requirements for this application. The drain-source resistance (on) is somewhat low at 200mΩ. The MOSFET's operating temperature ranges from -55° C to 175° C and has a typical turn off delay time of 50ns. It is available in the TO-252 package.

MOSFET: Fairchild Semiconductor FDD5614P

The FDD5614P by Fairchild Semiconductor is a P-Channel MOSFET with a high continuous drain current of 15A. The drain-source breakdown voltage is -60V, and the gate-source breakdown voltage is ± 20V. The drain-source resistance (on) is cited as 76mΩ. This MOSFET's operating temperature is also -55° C to 175° C. Its typical turn off delay time is 19ns.

MOSFET: Vishay/Siliconix SUD45P03-10-E3

The SUD45P03-10-E3 by Vishay/Siliconix is a P-Channel MOSFET with a high continuous drain current of 15A. The drain-source breakdown voltage is 30V with the gate-source breakdown voltage at ± 20V. The drain-source resistance (on) is a very low 18mΩ. This MOSFET's operating temperature is also -55° C to 175° C. Its typical turn off delay time is higher, at 100ns.

MOSFET Choice

The MOSFET chosen for this project is the Vishay/Siliconix SUD45P03-10-E3. The most attractive quality of this MOSFET is it's low Rds(on) resistance, at 18mΩ. This is vital, as at 15A, the power dissipated in the MOSFET will be $P = 15^2 * .018 = 4.05W$. However, if the control board is split into two separate boards, the power dissipated reduces to $P = 7.5^2 * .018 = 1.0125W$, a much more reasonable value. This low value of Rds(on) is difficult to achieve with such a high current - but clearly it is one of the most important concerns for this design. Additionally, the turn off delay time of 100ns is troublesome. Both the problem of power dissipation and higher delay time will require solutions in different aspects of the LED cube design. These solutions will be addressed in closer detail in the appropriate sections.

3.2.6 FPGAs

Due to the strict timing requirements of the LED Drivers chosen, a microcontroller is not ideal for communicating with the drivers used. The group has decided to use an FPGA to handle the strict timing requirements of the LED drivers, and to lighten the total load on the CPU, to allow it to focus on data generation and transformation. The FPGA will be used to generate all of the necessary control signals for driving the LED Drivers, and the MOSFETs in order to provide the highest refresh rate possible. FPGA devices by their very nature are ideal for strict timing due to the fact that they are hardware devices, there are no interrupts to slow execution.

FPGAs, being reconfigurable hardware devices, require some method of transferring this design from a development computer to the FPGA itself. There are many possible paths of accomplishing this process, each of which target a specific interest and need. Various methods are available to provide the easiest method of updating software, code security, fast boot time, and cheapest component cost. The group values a process that will be easy to update to a new version of firmware, due to the development nature of this project. Low component cost is also an advantage, but is not the highest priority because there will only be one control board made. The main methods of configuring an FPGA are an external EEPROM, using a CPLD, using an external microprocessor, to which the FPGA is a slave, or by means of connection to a development computer. Due to the proposed architecture of this project, it makes the most sense that the CPU be responsible for programming the FPGA. Due to the relatively large nature of FPGA configuration files, it is necessary that the CPU will have an external memory to store this configuration, which will have to be read at start up, and sent to the FPGA via the appropriate method.

When configuring an FPGA via an external processor, it is possible to use a serial, or a parallel method. Using either of these methods, it is necessary to communicate with the FPGA and tell it when to expect a program image, and how to expect this program image. There are a series of mode pins for telling the FPGA which the FPGA will read at power up to determine how it should receive it's configuration. These mode pins will be hard wired in our circuit design to only allow for external loading via a processor. The processor will be connected to the FPGA in order to signal when it should perform a reset, and when it should load new data from the input. Serial methods require only a small number of pins for communicating data, a clock line, and a data line. Parallel methods require a clock, and a series of data pins corresponding to the width of the data bus. Due to the ease of serial communication within the embedded CPU chosen, and the nature that only a few pins are required for communication, the FPGA will be loaded by the CPU in a serial nature.

The FPGA may also be used to perform more complex analysis and computation on the embedded platform, due to its ability to perform complex operations very quickly. The group may be able to perform a process such as a fast fourier transform (FFT) to analyze real-time signals received by the embedded control system to provide an interactive display experience.

An overview of the desired features and traits of the FPGA for the 3D LED cube can be found in Table 3.6 below. These traits listed are a summary of what the group is looking for in a candidate FPGA, and will be used to help the group objectively choose between researched FPGA parts.

QFP or other leaded package for assembly
Loadable via external processor
40 IO pins available
3.3V tolerant IO
Free or Low cost development tools
Familiarity with Development Environment
Low cost
Easy to obtain

Table 3.6: Desired FPGA Features and Specifications

FPGA: Xilinx Spartan 3

Xilinx Spartan 3 FPGA's are a very common FPGA for small project use, available in many different sizes, packages, and price points. The Spartan 3 line of FPGAs would be large enough to handle the multiplexing of the LED Drivers, and communication with the embedded processor. Each member of the group is familiar with the Spartan 3 line of FPGAs from classes requiring FPGA design. The familiarity with the software used for developing firmware, and the nature of Xilinx FPGAs is a large benefit to the group. A table of applicable features to the project can be seen below in Table 3.7.

QFP Package available
Loadable via SPI
173 IO Pins
3.3V tolerant IO
Free ISE Development Suite
$15 in low quantity
Available from most electronics distributors
200k System Gates
216Kb RAM

Table 3.7: Xilinx Spartan 3 Features and Specifications

FPGA: Xilinx Spartan 6

The Spartan 6 line of FPGAs is a more powerful, larger architecture of the Spartan 3 line. The Spartan 6 includes more features, such as on-board DSP processing units, larger logic blocks, and capabilities for higher clock speeds and more throughput. The Spartan 6 FPGAs are available in larger packages. A table of applicable features to the project can be seen below in Table 3.8.

QFP Package available
Loadable via SPI
200 IO Pins
3.3V tolerant IO
Free ISE Development Suite
$15 in low quantity
Available from most electronics distributors
9152 Logic Cells
576Kb RAM

Table 3.8: Desired FPGA Features and Specifications

FPGA: Altera EP1C6

Altera's EP1C6 FPGA is one of their largest FPGA's available in a TQFP package. The group considered this part in order to compare the offerings of Xilinx with other manufacturers. Without much familiarity with the Altera line of FPGA's, the group is hesitant to use it without a large quantity of benefits, which the group was unable to find. The Xilinx Spartan 6 part has more internal RAM, has more IO pins, and has a lower cost. This makes the Altera part an unwise choice for this group to use. A full list of the useful features for the Altera EP1C6 can be found below in Table 3.9

QFP Package Available
Loadable via SPI
185 IO Pins
3.3V tolerant IO
Free Quartus Development Software
$28.40 in low quantity
Available from most electronics distributors
5980 Logic Elements
80Kb RAM

Table 3.9: Desired FPGA Features and Specifications

FPGA Choice

Given the needs of the project, the group considers the best choice FPGA to be from the Xilinx Spartan 6 line. Specifically, the XC6SLX9 FPGA, the largest available in a TQFP pinout package. The group is limited to using an FPGA available in a TQFP package due to the ease of its use compared to a BGA or QFN package. This package will also simplify the design of the printed circuit board, allowing for the group to be able to route all circuit board traces with ease on a minimally complex board. The Spartan 6 also has the DSP slice, which the group will be able to utilize to do advanced computation that would otherwise overload the embedded CPU. The

Xilinx Spartan 6 meets or exceeds all of the desired features for the system FPGA. The choice of an FPGA that is larger than necessary will provide the group with room to grow, and add additional features as time allows.

3.2.7 Communication Methods and Protocols

Data must travel between multiple components throughout the systems of the project. In order to transmit this data, the use of defined protocols simplify the communication process. Multiple methods exist with various advantages and disadvantages that may benefit the different portions of the project. Communication methods must be selected for both the communication between firmware and hardware components as well as for communication of data from the software. The selection of communication protocol will impact both performance and complexity of data transmission within the components of the project.

Back-plane Communication

The nature of the 3D LED cube will require that multiple circuit board components are able to communicate with each other efficiently and quickly. A back-plane - located on the circuit board - communication method will be required. There are many different methods and protocols available to communicate between various programmable circuit chips. The design of the project will require the use of multiple communication buses, carrying information across various voltage levels and clock speeds.

Serial Peripheral Interface SPI, also known as Serial Peripheral Interface, is a common interface used between circuit chips. This interface will be used to communicate with the LED driver chips on the driver circuit board used to power the LED Cube. SPI is a loosely defined standard, capable of being implemented across various voltage levels, clock rates, and transmission methods. SPI is most commonly implemented on a single circuit board, or used to connect two circuit boards together, either via a on board mating physical connector, or a physical connector and a length of wire between the circuit boards. SPI is a bus based protocol, allowing multiple devices to share data using the same wires. The bus in controlled by a bus master, or master for short. All other devices are bus slaves, slaves for short. The bus master chip is the only chip allowed to drive the Clock, Chip Select, and Master Out pins. These pins are considered outputs for the master device, and inputs for all slaves on the bus. The master will not drive the Master In pin, and will leave this pin as an input. The master can also drive the Chip Select line either high or low, depending on the specific slave device, to alert the slave device that it is required to listen and respond to data. When the Chip Select line for a device is driven to the active state, the slave will sample data from the master on either the rising or falling edge of the clock, depending on the implementation and specification chosen. At the same time, the slave will transmit data on the Slave Out line, to be read by the master chip. This is the only pin that the slave is able to drive, and is the only output for the slave, all other pins on the slave are considered inputs. In order to have more than one chip on the bus and communicate to individual chips without collision, multiple chip select lines can be used. The master will know which chip is at which address via the chip select line, and will set this address on the chip select line before initiating communication. Using this method, it is possible for the master to communicate to a number of different chips while sharing the same data bus. This is sometimes called multiplexing the data bus and resembles the parallel architecture as shown in Figure 3.15. It is also possible to drive multiple chips in a series or daisy-chained configuration, as seen in Figure 3.16. Using this configuration, the SPI bus uses one chip select, and each chip is connected in a daisy chain manner. The Master connects to the MOSI of the first chip, the MISO of the first chip will connect to the MOSI of the second chip, and so on and so forth until the last chip's MISO connects to the MISO of the bus master. Using this method, all data is passed through each of the chips in a series manner.

SPI is a four wire protocol when fully implemented. These four wires each have a specific purpose and direction. The four signals required for a full implementation of SPI are Clock, Master In Slave Out, MISO, Master Out Slave In, MOSI, and Chip Select, CS. Implementations requiring only a one way communication may leave off the data line in the direction not used, and use SPI as a three wire communication method.

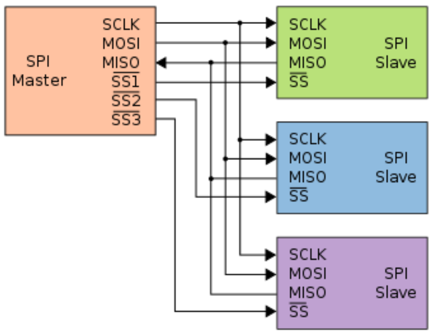


Figure 3.15: SPI Chip Select Topology
Printed with Permission, Courtesy of Wikipedia.org [5]

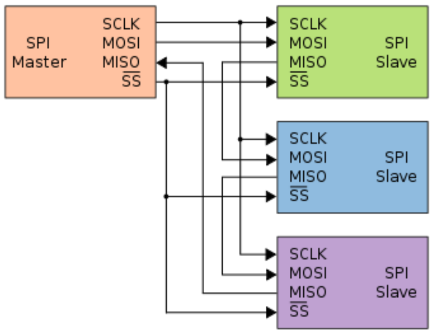


Figure 3.16: SPI Daisy Chain Topology
Printed with Permission, Courtesy of Wikipedia.org [5]

Parallel Parallel interfaces are less commonly used for inter-circuit communication in the current time with ever increasing clock rates within CPUs and FPGAs, however are worth considering. With properly implemented parallel interfaces, it is possible to transfer large quantities of data in relatively short periods of time, something that will be needed within the 3D LED cube. Parallel interfaces are useful when communicating data to discrete circuit elements, such as transistors. Parallel interfaces can be either synchronized, or unsynchronized, depending on the parameters of the circuit they are connecting. Parallel interfaces operate over a wide variety of voltages, data rates, and cabling. When using parallel interfaces, it is important that the cable lengths match for each of the data lines. If the cable lengths are unmatched, it is probable that one signal will arrive at the receiving end before another. This could cause great issue if each of the data lines were read before all of the data was present at the input. This is generally only a concern for high frequency, long distance cable runs, but it is possible that this could become a problem for the group. Every effort will be made to match cable length.

Inter Processor Communication

Inter Processor Communication refers to the communication that takes place between the laptop that will be running the software interface and the control board in the LED cube that will control the LED drivers. This communication is a vital aspect of the LED cube design, making it imperative to utilize a reliable and high speed communication protocol. To push animations from the software interface run on the laptop to the control board at the desired rate, a data transfer rate of approximately 2 megabits per second will be required. This will be the primary concern in the selection of a communication protocol.

Ethernet Ethernet is high speed published IEEE standard for communication. It's data transfer rate is on the order of a gigabit per second over standard copper cable (CAT5). Ethernet is both reliable, with error checking built in, as well as cheap to implement cheap to implement. Each of these reasons contribute to the massive popularity of ethernet as a communication protocol.

RS232 RS232 is an older communication protocol that, while once popular, is not used nearly as much as it used to be. While reliable, its speed is limited to 115 kilobytes per second. Further, utilizing this protocol would require an adapter on both the laptop and embedded processor. It also has a limited distance of approximately 50ft.

DMX512 Due to a handful of proprietary and incompatible of communication methods for controlling lighting solutions, DMX512 was developed as a standardized digital communication method. The serial protocol transmits 512 bytes of data to nodes connected along a bus. DMX512 is used throughout the lighting control industry for applications that range from small household lighting projects to professional stage lighting and special effects.

Art-Net Art-Net was developed to transmit DMX512 packets over the Internet Protocol (IP) stack. Art-Net sends the DMX512 lighting information from a server to nodes across a local area network in User Data Protocol (UDP) packets. The Art-Net protocol has the advantage that applications can still use the DMX512 protocol over existing Ethernet connections without the need for the specialized serial interfaces.

E1.31 E1.31 is a protocol within the Architecture for Control Networks (ACN) used for lighting control applications with UDP packets over the IP suite. E1.31 defines Streaming DMX over ACN is similar to Art-Net allows for DMX512 messages across existing Ethernet or wireless frameworks. The E1.31 protocol is becoming the new standard for DMX transmission over IP and will surpass protocols such as Art-Net.

Bluetooth Bluetooth has emerged as a very popular wireless data transfer protocol. Initially, this was an interesting method, but the low speeds of approximately 100 kilobytes/second disqualified it. Another concern in utilizing bluetooth is its fairly limited range of 30ft. While utilized in smart-phones, tablets, and many wireless devices, it isn't a solid option for this design

Communication Protocol Choice While initially, there was reason to investigate multiple data transfer protocols, Ethernet quickly emerged as the clear choice. The other protocols are not able to deliver the speeds required for this project, and Ethernet provides no complications or design challenges, whereas the other methods do. The Art-Net protocol operating over Ethernet will serve to transmit the lighting control information to from the software to the LED driving firmware.

3.2.8 Power Supplies

The power supply for the 3D LED cube is an important consideration. The LED cube will be drawing a significant amount of current (15A) at reasonably low voltage for LEDs (5V). Additionally, the CPU and FPGA I/O will require approximately 2A at 3.3V and the FPGA core will require approximately 1A at 1.25V. With a 5V power supply, these lower voltages will be obtained using a simple voltage regulator. It is important when considering power to overshoot the requirements, so that the power supply is not operating near 100% capacity. Utilizing 75% to a maximum of 80% is ideal.

Power Supply: Mean Well SP-200-5

The Mean Well SP-200-5 is a robust power supply, providing an output current of 40A at 5V (200W). It accepts 85-264V AC or 120-370V DC as its input voltage. Its size is relatively compact at 200mm L x 100mm W x 50mm H. Stated a commercial power supply, it is entirely enclosed - providing essentially plug-and-play operation

Power Supply: Mean Well SP-150-5

The Mean Well SP-150-5 is very similar to the SP-200-5, with 10A less current, providing an output current of 30A at 5V (150W). It accepts 90-264V AC or 254-370V DC as its input voltage. It is similarly enclosed and compact at 200mm L x 100mm W x 50mm H.

Power Supply Choice

While both the Mean Well power supplies are fantastic options, the 150W provides all the necessary power. Although these high-end power supplies will operate near 100% capacity, using a 150W power supply ensures that we will operate at an estimated 65% ($5V * 20A = 100W$) capacity with room to add any additional features or functions if necessary. The Mean Well SP-150-5 is the primary power supply choice for this project.

3.3 Software Research

Computers execute a series of binary instructions, or machine code, in order to process a computer program. Machine code while well defined is not easily programmed by humans. Programming languages were developed to create a formal higher level description of a program. A high level programming language removes fine hardware architecture level details from the program and allows the developer to focus the majority of their development on the primary intent of the program. Throughout computer software, certain routines are common between programs. To alleviate the need to rewrite the same sections of code, programmers developed sets of commonly used routines and packed them into what are called libraries. Libraries allow for a common programming interface that can be shared between multiple developers and increase the interoperability between programs. Multiple programming languages and pre-built libraries exist with which to develop the software portion of the project. Each language has advantages that bring strength to the software development process as well as potential drawbacks which might make the development more difficult to achieve. The careful choice of language and specific libraries will allow for ease of development and integration throughout the project.

3.3.1 Programming Language

The choice of programming language is the foundation of software portion of the project. The programming language dictates what feature sets are available to the developer as well as what development tools exist to support that language. The language also impacts the direction and pace of the project with certain languages favoring speed of prototyping with others favoring robustness throughout the development life cycle. Programming libraries may depend on language choice and could potentially aid or hinder the development depending on support for the language of choice.

C++

Developed to supersede and extend the C programming language, C++ allows a developer to have both high level language features such as object-oriented techniques, built-in abstract data types, and error handling as well as still giving developers low level control of the program by allowing them to make fine optimizations to the final compiled code prior to compilation. C++ has been in development since the early 1980s and due to it's features, it has grown a large user, developer, and support base. The language is one of the major professional software development languages, therefore many resources and libraries exist for further development. The fine level optimization capabilities are advantageous in time sensitive applications such as the 3D LED cube where minimum execution time is a necessity for optimum performance. Through the Microsoft Framework Class (MFC) library, Windows developers have a framework to develop Graphical User Interface (GUI) applications. The power and flexibility afforded to the developer makes this language a strong candidate for the software portion of the project.

Java

Another popular programming language is Java that was developed to feel similar to the programmer to C or C++. Java programs execute within a virtual machine called the Java Runtime Environment (JRE) which allows for programs to be platform independent and highly portable. A purely object-oriented language, Java was designed with code reuse in mind. Developers strive to define classes to describe their program's data at fine levels of abstraction in order to allow for reuse among similar problems. The mentality of abstract data description and code reuse allows for developers to speed up development once a solid framework is created. It follows that if Java packages already exist which implement features a developer needs, the developer does not need to recreate the solution. Developers can use the Swing toolkit to develop GUIs to extend there applications in much the same way as with MFC in C++. Developing the 3D LED cube in Java would allow for ease of support and continued development as the project grows as well as the ability for the program to run across multiple platforms. The fact that Java applications must run in the JRE does impose an extra layer of computation that may prove to hinder the time critical performance of the system.

Python

Python is a high level programming language designed for developers to translate their ideas clearly and concisely into readable code. Python has dynamic typing which means the language will infer variable data types through the context they are used without the need for rigid definitions. Python programs are run through an interpreter which parses through the source files to execute the program, or Python code can be entered by hand to an interpreter and be executed in real-time. This language feature lends itself to Python being primarily used as a scripting language. The ability to quickly code their ideas gives developers an opportunity to rapidly prototype concepts. This practice of making small, quick prototypes and testing as development proceeds improves the development process and allows for minor course corrections to be made during development in place of a large change needing to be made at the end of a major milestone. Python supports object-oriented programming and can be compiled into a stand alone executable to achieve similar performance to other compiled programs.

3.3.2 Libraries

With many developers working on similar complex issues, there became a demand for central application programming interface (API) collections known as libraries. The advantages of developers using a library over their home-brew solution include code reuse, reliability, standardization, and ease of development. A develop can harness the power of a programming library that handles complex routines such as networking or graphics simply by utilizing the APIs that the library makes available. The APIs are generally much more developer friendly and remove most heavy lifting from that portion of development. Through the use of appropriate libraries, a developer is able to greatly extend the capabilities of their application.

Direct3D

Microsoft developed a proprietary 3D graphics library known as Direct3D which is exclusive to the Windows platform. Direct3D was designed as an extension to the DirectX graphics API collection and now in current versions supports the 2D graphics features of DirectX. The Direct3D library allows for rendering of 3D scenes with the support of features such as texture mapping, anti-aliasing, alpha blending, and depth buffering. The framework will attempt to use hardware accelerations if available to speed up graphics computation. The Direct3D library is supported by COM-aware languages such as Visual C++, C#, or Visual Basic. This limitation constrains the choices of programming languages available to the developer.

OpenGL

The Open Graphics Library, better known as OpenGL, is a open standard 2D and 3D graphics library that directly competes with Microsoft's Direct3D. The OpenGL API implements similar and in many cases the same 3D graphics rendering features as its competitor to include. OpenGL manages the hardware resources rather than requiring the program to do so, a key difference of OpenGL over Direct3D. This allows for easier development of graphics based applications due to the extra work handled within the library itself. OpenGL was designed to be language independent and it is supported widely by a vast amount of programming languages. Similarly, OpenGL is platform independent and not limited solely to Windows operating systems.

Open Lighting Architecture

The Open Lighting Architecture provides an interface for programs to transmit and receive DMX512 messages over IP. Using this library, developers are able to control their lighting solutions over any IP protocol such as Ethernet or wireless in addition to the traditional DMX512 interfaces. The Open Lighting Architecture includes libartnet, a library that developers can use to transmit and receive with the Art-Net protocol. Currently, the Open Lighting Architecture only supports Linux, FreeBSD, and Mac OS X, but there are solutions to make the libraries work on Windows operating systems. Specifically, instructions exist to compile libartnet on a Windows computer to a Dynamically Linked Library (DLL) for use with Windows applications.

Chapter 4

Project Hardware and Software Design

The design of both the hardware and software subsystems of the project comprises the bulk of the work and effort for the first half of the project. Proper and thorough work during the design phase of the project ensures that work does not need to be repeated and that the construction, integration, and testing phases of the project will run smoothly. Design documentation includes hardware wiring diagrams, software class diagrams, data flow diagrams, power calculations, and design requirements. By following the designs laid out in this chapter, the construction and integration of portions of the project will be greatly simplified.

4.1 Architecture and Related Diagrams

This section will describe in detail the hardware and software architecture achieved during the design phase of this senior design project that is necessary for the completion of the 3D LED cube prototype. Each component of the hardware and software design will be dissected and analyzed clearly, with supporting diagrams to distinctly illustrate the operation of the 3D LED cube at a much more detailed and lower level than discussed in other areas.

4.1.1 Hardware Architecture

The hardware architecture of this project has been designed to be as simple and as straight forward as possible. The group chose an architecture that lends itself to easy design and configuration, over one that is overly complicated. The LED cube will be a fully contained system, requiring no external assembly or considerations. By simply plugging the power cord in, and connecting the cube via an Ethernet cable to either an existing network, or a direct connection to a computer, the cube will be able to light up and be controlled by the user.
The 3D LED cube will contain multiple circuit boards each functioning to fulfill a different role within the illumination of the LEDs. The final prototype will contain three circuit boards, one control board containing an FPGA, CPU, and other necessary components, and two driver circuit boards, each identical to each other, containing the LED Drivers, MOSFETS, and other necessary components to drive the LEDs as needed to illuminate the cube. The boards will connect to each other by the means of short ribbon cables. The choice to split the architecture into multiple circuit boards was due to the sheer number of components that are necessary in order

to drive an LED cube of this size. The other reason the group chose to create more than one circuit board was to attempt to half the chance of failure, in that if there is a detrimental mistake or flaw within the control circuitry, the drive circuit can still operate while a new board is designed or manufactured.

Embedded Processor

The embedded processor is one of the most essential components of the project. This single part will handle all communication with the user's computer, and the FPGA in order to display data onto the cube. The PIC24 CPU chosen has more than enough processing power to be able to react to user input in a timely fashion, and transfer this input to the FPGA, or respond to the user's request. The CPU will connect to most of the circuit components directly, and any components not connected to directly, are connected via the FPGA, and will be shared when communicating with the FPGA. This flexibility will allow the group to make design changes as they deem fit. The CPU will connect directly to the Ethernet interface, micro Secure Digital card, the FPGA, and handful of status LEDs and user buttons. The ability for the CPU to connect to such a large number of external interfaces will increase the versatility of the system. A schematic diagram of all of the CPU connections is shown below in Figure 4.1. These interfaces are described in detail below as well.

The ability to connect the PIC24 CPU to a Microchip Ethernet interface will be utilized within this project in order to provide an easy, trouble free method of interaction with the cube from the users perspective. The cube will be able to interface to any standard wired or wireless network, alleviating the need for the group to create a custom interface. Standard TCP/IP protocols will be used to alleviate the need to create a custom protocol layer, and will further simplify the architecture of the software on the embedded processor. One of the groups reasons for choosing to use ethernet was the pervasiveness of ethernet devices. Nearly every person has a wireless ethernet connected device, and would be able to interact with the cube via a wireless network, possible by plugging a wireless router into the ethernet port of the LED cube. A complete hookup diagram of the ethernet interface can be seen below in Figure 4.2

The embedded processor will be reprogrammable by an external ICSP device. These devices are often used for first programming until a boot loader is written or available. It is the groups desire that a boot loader be created such that the group will not have to use a physical circuit programmer, but be able to simply reprogram the board from any users computer, with a small software utility that would communicate with the processor via its Ethernet interface and provide a software update in the field. This desire is not a required feature, but would simplify greatly the process of software updates. In actual product release, the requirement of having a user purchase an external programming kit is not an option, and would require the need of a working boot loader.

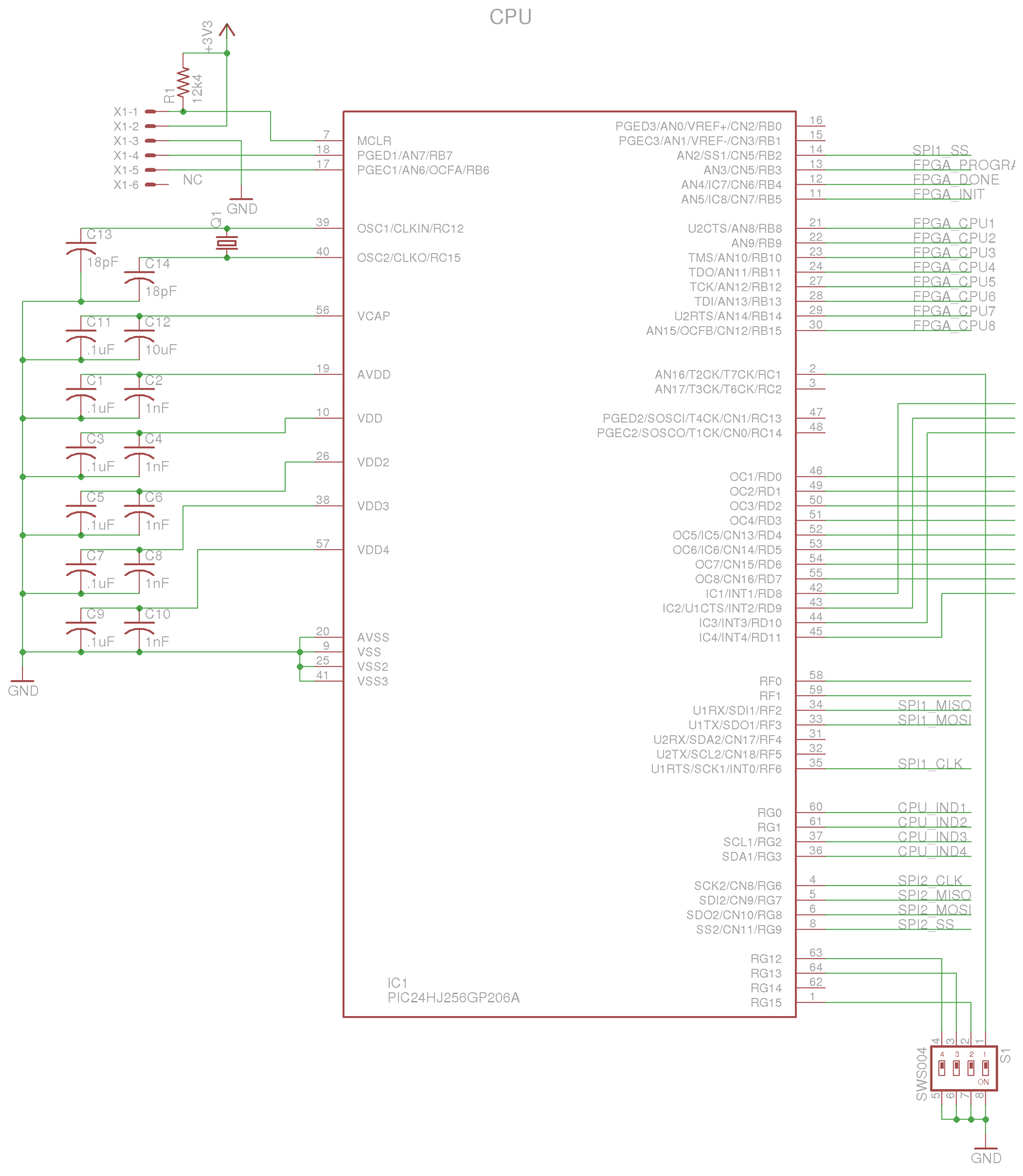


Figure 4.1: CPU Schematic

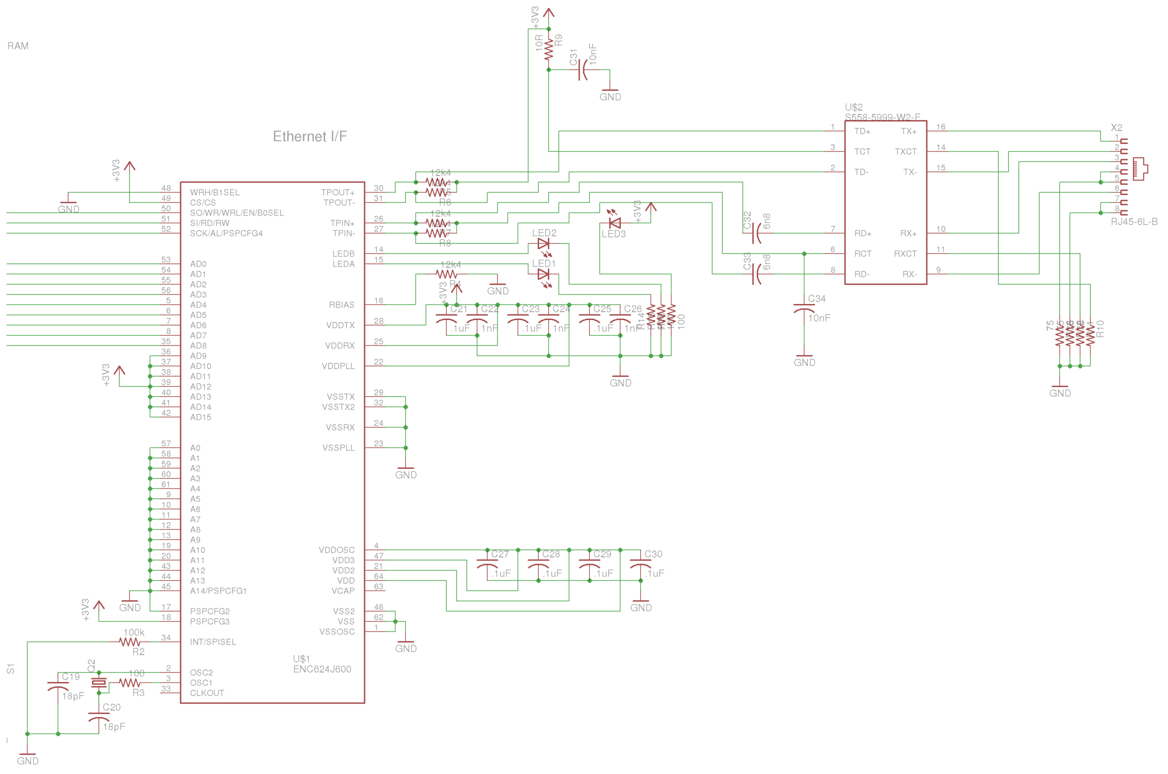


Figure 4.2: Ethernet Schematic

Communication between the embedded processor and the FPGA will occur on two buses. There will be a high-speed SPI interface between the FPGA and the embedded processor for communicating the data that is to be displayed on the LED cube. This interface will consist of single communication frames that contain an entire displays animation frame. This interface is required to be high speed due to the groups goal to reach an animation rate of twenty-five frames per second, with each frame containing eight-bit lighting data for each of the three thousand LEDs in the cube. This requirement dictates a data transfer rate of at least 600kbit per second. This data rate is easily achievable with SPI. The other interface between the CPU and FPGA will be a general-purpose parallel interface. This eight-bit wide interface will be left unimplemented, and will remain as an interface usable for expansion, and additional features that are not yet foreseen by the group.

The embedded processor is also responsible for the configuring of the FPGA at power up. This is necessary, as the FPGA is unable to read its configuration directly from the micro SD card before it has been programmed. The configuration file for the FPGA will be stored within the micro SD card on board, as it is unable to fit within memory of the CPU. During the initialization sequence, the CPU will read this file from the SD card and will push it to the FPGA in the necessary format. The processor will be responsible for communicating to the FPGA when it is necessary to reboot, when it is necessary to expect a new configuration, and when that configuration is to be implemented. The processor will take direct control of the FPGA's configuration lines, and monitor its output status to insure that the FPGA loads its configuration properly. Once the FPGA is up and running, the CPU will release control, and allow the FPGA to do its own work.

The embedded processor will share a bus to a micro secure digital card (SD card), capable of storing much more data than the internal memory of the CPU or FPGA. This SD card will be used to store large quantities of data, like preprogrammed sequences, and FPGA images for development and production. The micro SD interface will be implemented on a secondary SPI bus with the CPU acting as the bus master when it is communicating with the card, and releasing control and allowing the FPGA to act as the bus master when it needs to communicate with the SD card. It is anticipated that the FPGA will be able to communicate with the SD card at a much higher rate, and will be used for most data storage operations.

The embedded processor will have a series of five status indicator LEDs, used to show the current system state and processing utilization. These indicator LEDs will be surface mounted onto the control board, and will be used for diagnostics purposes only, as they will not be seen when the control circuitry is enclosed within the project structure. The processor will also interface to a series of SMD switches that will be read by the processor and used to control what mode of operation the system runs in. These switches, as with the LEDs will not be reachable from outside the project structure, and will be used mainly during the development phase of the project.

The most important part of the CPU is the firmware that will be running once the device boots. The software will be responsible for handling all of the data exchange, as well as ensuring that the user interface operates as expected. The firmware will go through a series of initialization steps, before beginning a processing loop that will receive data on the input, process the information received, and distribute the necessary outputs to the different output devices. This control loop will be run at the highest rate possible. It will be necessary to keep the program from blocking, or waiting for input, in order that all stimuli can be responded to in a timely fashion. The firmware development will be done using Microchip's MPLAB X development environment, as well as Microchip's TCP/IP stack, which is available without license for commercial and noncommercial projects. The utilization of this TCP/IP stack will save a great amount of time in developing working software.

FPGA

The FPGA on the control board is also one of the most essential components of the design. The FPGA is essential in ensuring that the drive circuitry is communicated within its strict timing requirements, and is the only way that the cube will be able to light up. The FPGA is connected to many parts of the circuit, and any parts not connected directly to the FPGA will be connected through the CPU on the board, and accessible through communication with it. The FPGA will be directly connected to the drive circuitry for the LED cube, the CPU, and the micro SD card on-board. As many of the extra FPGA's IO lines will be broken out to an external connector to allow the group to expand the capability of the system with the addition of more elements. Seen below in Figure 4.3 is the hookup for the FPGA within the control circuitry. Each of the interfaces is described in detail below as well.

The FPGA will communicate with the drive circuitry for the LED cube, and will manage all of the dimming control of the LEDs. The LED drivers communicate via a SPI interface that operates with a 5 volt SPI interface. The FPGA however, operates with an IO interface of 3.3 volts. The FPGA would be unable to reliably communicate with the LED driver, as it would be unable to send a reliable high signal to the drivers, and would be damaged due to the signals coming from the LED drivers that are received at 5 volts. In order to provide a reliable method of communication, a method of level shifting will be required. In the direction of the FPGA to the driver circuitry, a simple logic buffer will provide the necessary level shifting. Using a 74HCT241 will boost the output of the FPGA from 3.3 volts to 5 volts. This works by the fact that the 74HCT241 is a five volt part, however the HCT class of parts requires only a 2 volt level at the input pin to be considered high, an over 50% tolerance for the FPGA's output level. In the other direction, it is necessary to translate the 5 volts down to the 3.3 volts necessary for the inputs to the FPGA. A 74LVC245 can be used. This chip allows the FPGA to communicate with a voltage level of 3.3 volts, while the driver circuitry outputs a 5 volt signal. The nature of MOSFETs also dictate that they have the highest drive voltage possible, the higher the gate source voltage, the more current can pass through with less resistance. The 74HCT241 chips will provide a five volt output to drive the MOSFETs.

The Xilinx Spartan 6 FPGAs have two methods for on-board memory storage. There are a series of block ram devices, each containing eighteen kilobytes of storage space. These ram devices are spread in discrete blocks throughout the FPGA, and can be used to store large quantities of data. The other option are the storage elements within the FPGA logic slices, which can be combined to form smaller ram elements. The nature of the LED cube will be one that requires larger quantities of storage, three thousand bytes per animation frame, as well as a large quantity of information necessary for driving the cube efficiently.

The FPGA will also have five status LEDs available to show the current state of the FPGA, and the current process it is working on. The LEDs, like the CPU LEDs will be mainly for debugging and development purposes, and will be hidden from sight during the normal operation of the cube. These LEDs will be extremely useful to the development, as it is difficult to debug a design on an FPGA without some sort of output indicators and visual indication of what the current process is.

LED Drivers and MOSFETs

The LED drivers used within the 3D LED cube will be used to control each of the three hundred negative terminals of the LEDs. These will be used to sink and control the current when the desired LED is to be turned on. The LED drivers will be contained within the project's driver PCBs. These drivers will be distributed between the two boards, half on each. This will allow for the distribution of current, instead of needing to put the full current through one circuit board, the current is divided in half, with 7.5 amps on one board, and 7.5 amps on the other corresponding board. This will dramatically decrease the power dissipated by the circuit components, as power decreases by an exponential factor with a linear decrease in current.

The LED drivers receive all of their data via a SPI bus. The clock frequency of this SPI bus is dictated by both the amount of driver chips in series, and the rate at which the cube needs to be refreshed. In order to refresh the cube at the desired 100Hz rate as listed in the project specifications, it is necessary to refresh through each of the ten layers at a rate of one hundred times per second. At this frequency, the each layer will be lit for a period of one millisecond. The data must be able to be sent to the LED drivers fast enough such that it can turn on the new layer, and leave enough time for the eyes to detect the new lights turning on. Each chip, containing 16 channels, at 16 bits each, will require 256 bits of information during each refresh. With this much data, it is necessary to use multiple lines of communication, as opposed to one line of communication for all of the chips. Each of the two driver boards will contain two separate SPI lines, each connected to four LED driver chips,

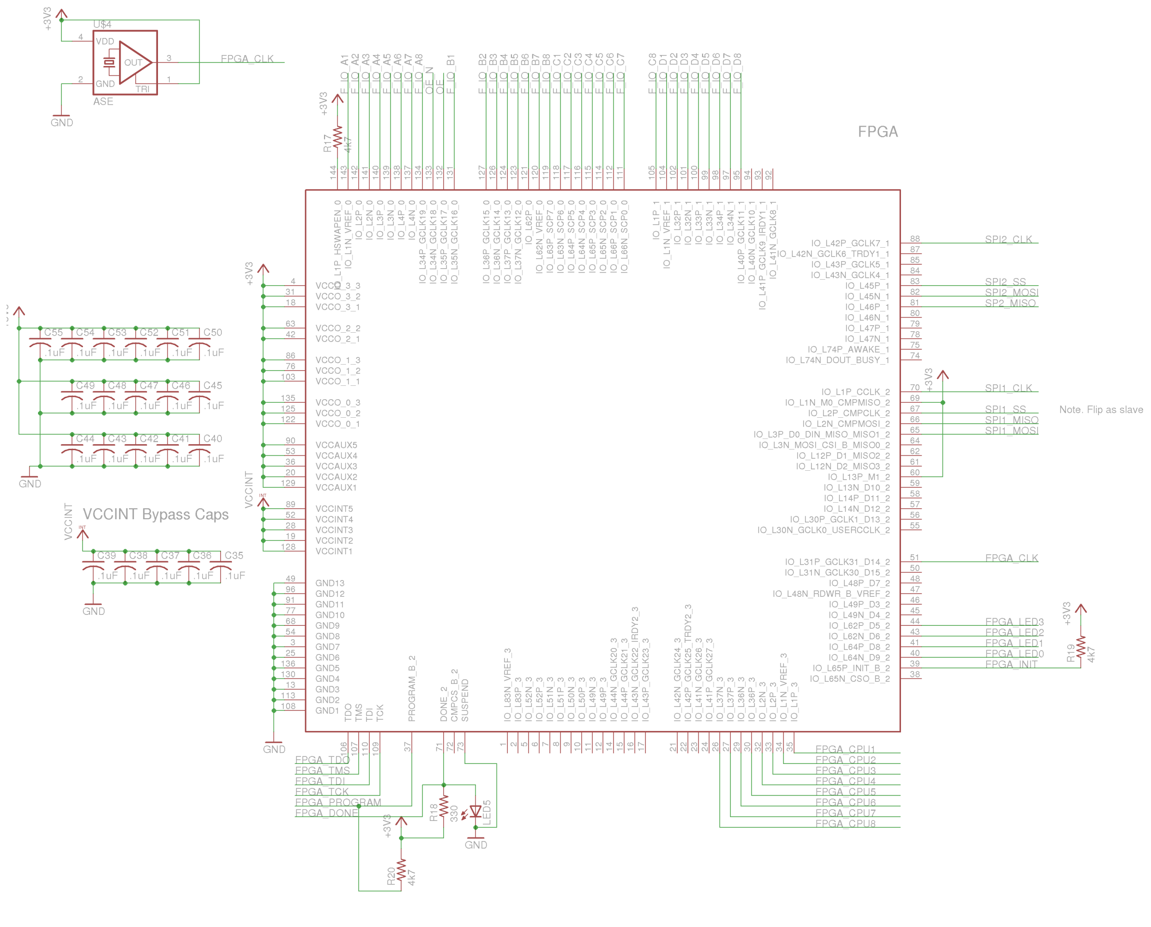


Figure 4.3: FPGA Schematic

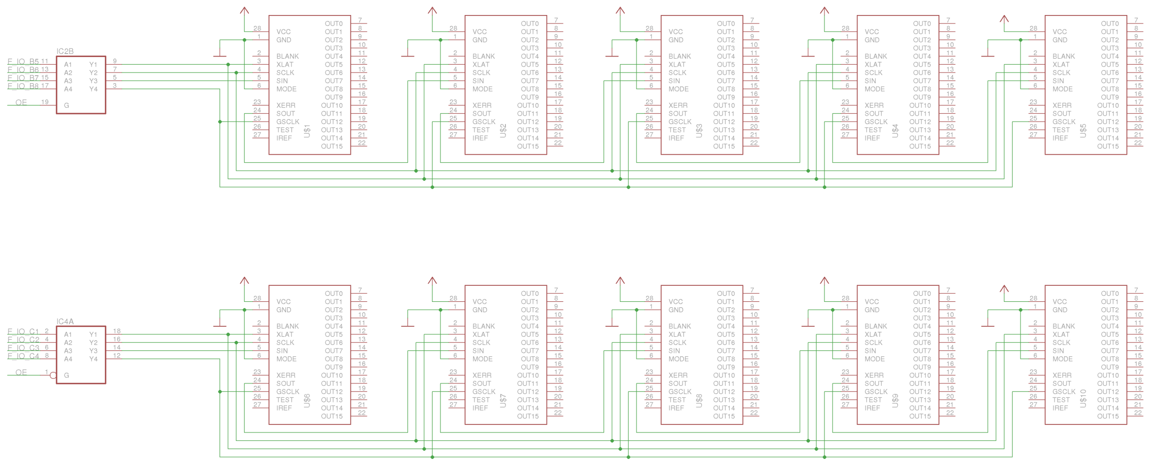


Figure 4.4: Driver Schematic

for a total of four SPI lines. This divide will allow for a higher throughput, and lower latency from data being received at the FPGA, to the data being displayed on the outputs.

The LED drivers are adjustable constant current sources. The current is set by a single resistor for each drive chip, and is desired to be set to 50mA. The equation for calculating this drive current is found in the datasheet for the TLC5948A LED Driver, and is given as follows:

$$R_{adj} = \frac{V_{ref}}{I_o} * 42.3$$
$$R_{adj} = \frac{1.20V}{50mA} * 42.3$$
$$R_{adj} = 1015\,\Omega$$

In order to keep the device current as precise as possible, the resistors used will be have a tolerance of 1 %. This will ensure that the maximum current will be 50.005 mA, and the minimum current will be 49.995 mA. This will allow for a uniform appearance of brightness across the entire display. In addition to this, it is also possible to use the brightness control feature within the LED driver, which will be investigated by the group if a uniform color distribution color is not already achieved. The resistor values chosen, as well as the method of hooking up the LED drivers can be seen in the following schematic capture, Figure 4.4 below.

The MOSFETs within the 3D LED cube will be used to turn on each of the ten positive terminals of the LED cube. Two MOSFETs will be used for each layer, each one corresponding to one hundred fifty LEDs. There will be a total of twenty MOSFETs, ten on each driver board for controlling the LEDs. This will be done to insure that power dissipation limits are not passed, and that the components used operate within their maximum ratings. Each MOSFET will at most be required to source 7.5 amps of current from the drain to the source terminal. As with any circuit element with current flowing through it, a certain amount of power will be dissipated internally, and will cause heat to be created. Using two MOSFETs per layer will cut the amount of power disappointed in each FET down by a factor of four. The calculations for the power dissipated are as follows.

$$P = I^2 * R$$
$$P = 7.5\,\mathrm{A} * 18\,\mathrm{m\Omega}$$
$$P = 1.0125\,\mathrm{W}$$

As seen, the MOSFETs will be required to be able to dissipate just over one watt each. This will be entirely within the designs of the circuit boards, as there will be ample heat-sinking available, as well as only one MOSFET will be turned on at a single time. It is the groups desire that the drive circuitry run as cool as possible. With the figures found so far, it may be possible to run the entire cube without active cooling.

Control Board Power

The control board will require active power management due to the various voltage requirements of the components on board. The CPU will require a voltage source of 3.3V as well as the ethernet interface, the FPGA will require both 3.3V and 1.25V, and the SD card will require 3.3V. The control circuit will take in 5V from the main power supply of the project, and will further filter and regulate this down to the necessary 3.3V and 1.25V that is required. This will be accomplished via the usage of two separate REG1117 regulators. These regulators each take in a power input, and will regulate the voltage down to a new level by means of dissipating the extra voltage. The dissipated voltage will be wasted as heat. A full diagram of the two regulators can be seen below in Figure 4.6.

LED Cube

The most complex wiring part of this project will be the assembly of the one thousand LEDs that the cube is composed of. Each of these LEDs has four terminals, a common positive terminal, and three negative terminals, one for red, one for green, and one for blue. Each of these LED terminals will be soldered to a bus wire, which will connect to each of the other LEDs together, and provide for structure of the LED. The anodes will all be connected together within one XY plane, and each of the colored cathodes will be connected together in a vertical Z axis column. This will be accomplished as a free standing structure, and will consist of four thousand solder joints for the group to complete. A connection of one plane of LEDs can be seen below in Figure 4.7. This shows how one XY layer will be connected together. Each of the XY layers are identical to each other, with the exception being that the anodes will each be connected together to different MOSFETs for control purposes.

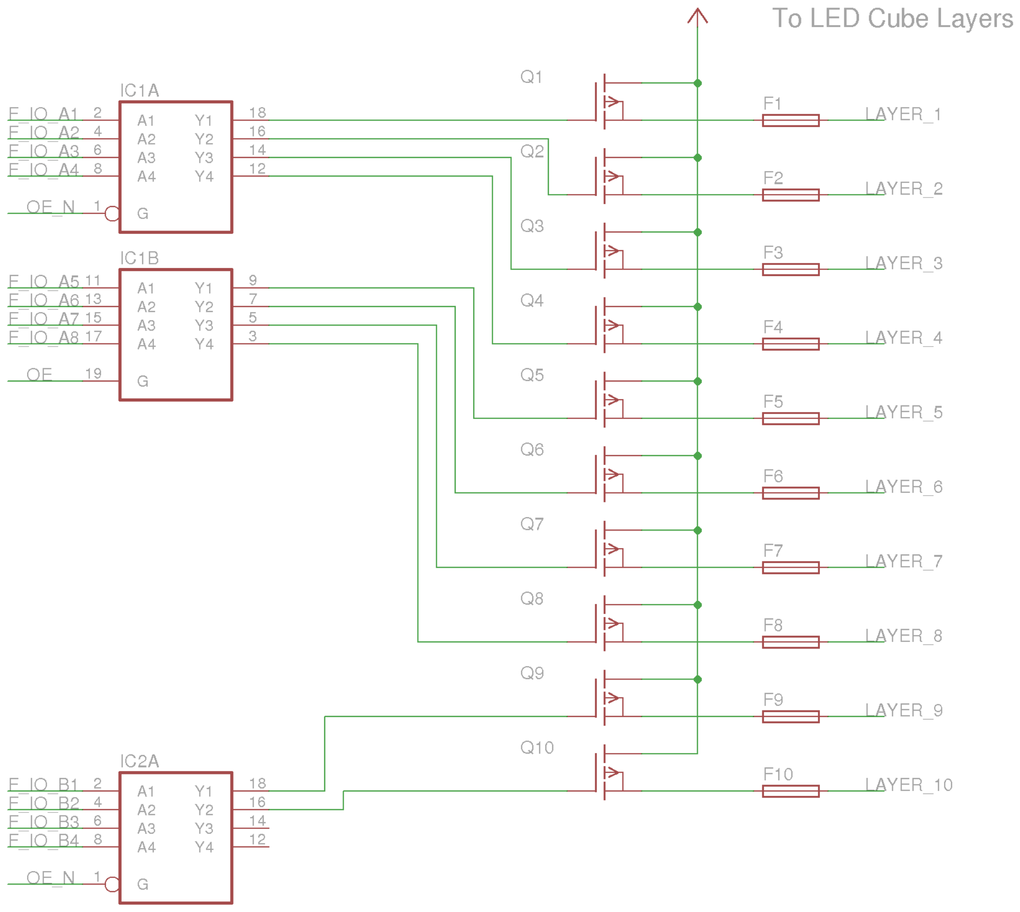


Figure 4.5: Mosfet Schematic

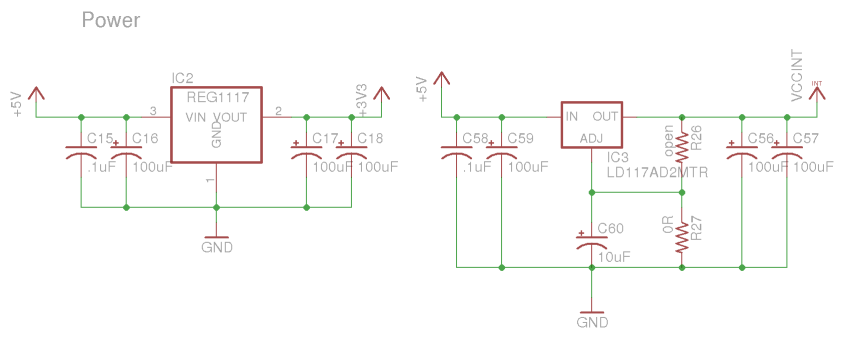


Figure 4.6: Control Power Schematic

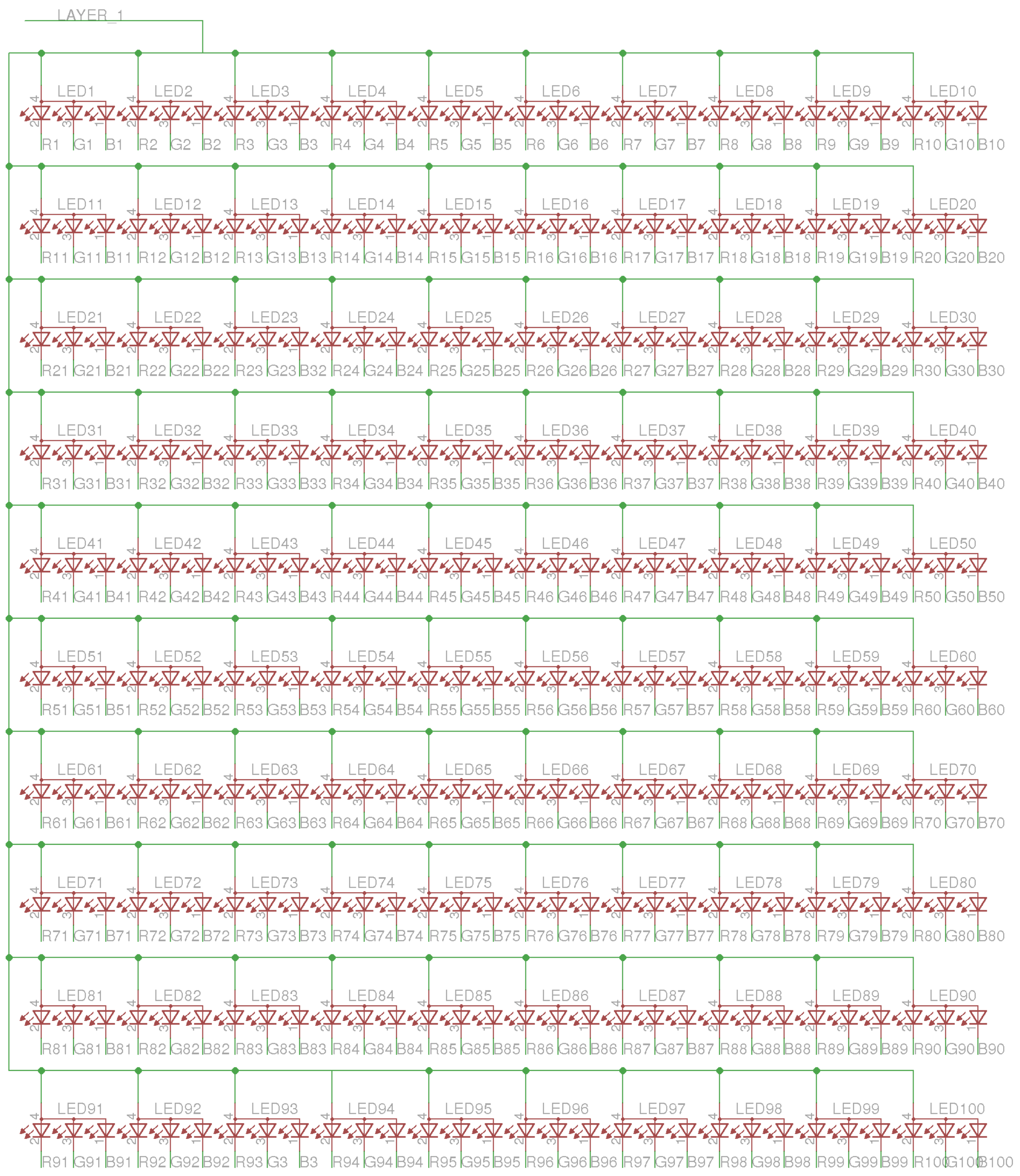


Figure 4.7: Cube Schematic

4.1.2 Software Architecture

The software architecture must consist of multiple key components shown in Figure 4.8 interacting with the rest of the 3D LED cube. While the hardware can display a matrix of LEDs in a predefined pattern with multiple colors, this in and of itself is not interesting. Therefore, the software will design multiple series of animations to display on the LED cube. The creation of animations are a key component to this project and significant emphasis will be placed on developing robust and easy to use software and development templates for use by the developer to create visuals. Once the animations are generated, this information must be relayed to the firmware in order for it to be displayed on the LED cube. A separate piece of software will handle the communication between the code that generates animations and the firmware that will drive the LED cube. The communication will occur over a wired Ethernet connection. Finally, to facilitate development alongside, but independent of, the hardware and firmware, software designed to simulate the behavior of the LED cube's firmware and hardware will stand in as replacements to allow for simplified testing while the other subsystems are still in development or not available for testing. The existence of software simulators will greatly increase the productivity of the software development process by allowing modules to be tested throughout development and consequently will reduce the number of unintended errors that will arise during the integration and testing process.

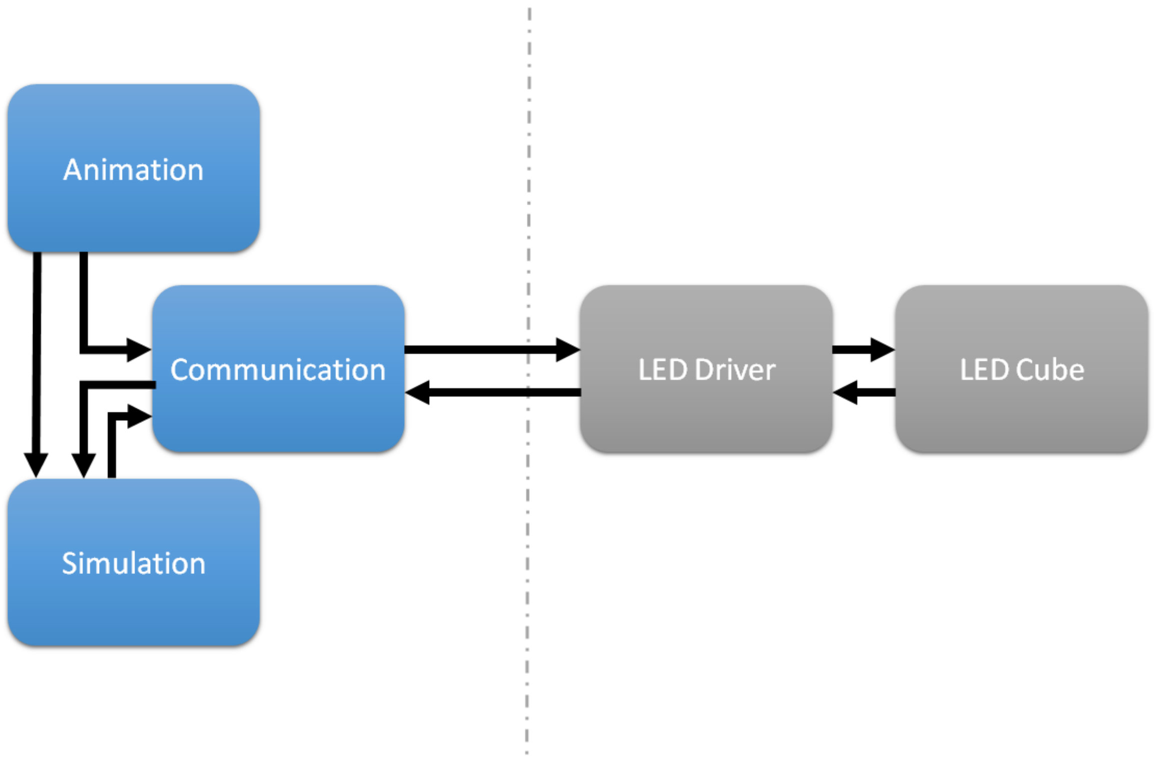


Figure 4.8: Software Architecture

Animation

The animation component of the software will be capable of generating animations through two distinct methods. Many complex animations such as manipulating geometric objects, scrolling figures, and displaying 3D mathematical patterns can be designed simply by writing code to govern the behavior each voxel during each frame of an animation. An exposed API to allow for fine control of the LED cube animations, this allows developers to have a powerful interface to create animations. Alternatively, manual entry of each voxel for each frame of animation through a GUI representation of the LED cube, shown in Figure 4.9, allows for a fine level of control to the user at the expense of time and complexity. The animation creator GUI will simply cover up the API available to the developer and create a user-friendly interface for designing animations for the LED cube. The combination of these two software elements allows developers and novice users alike to create a rich set of animations and features while permitting the simple incorporation of additional animation sets.

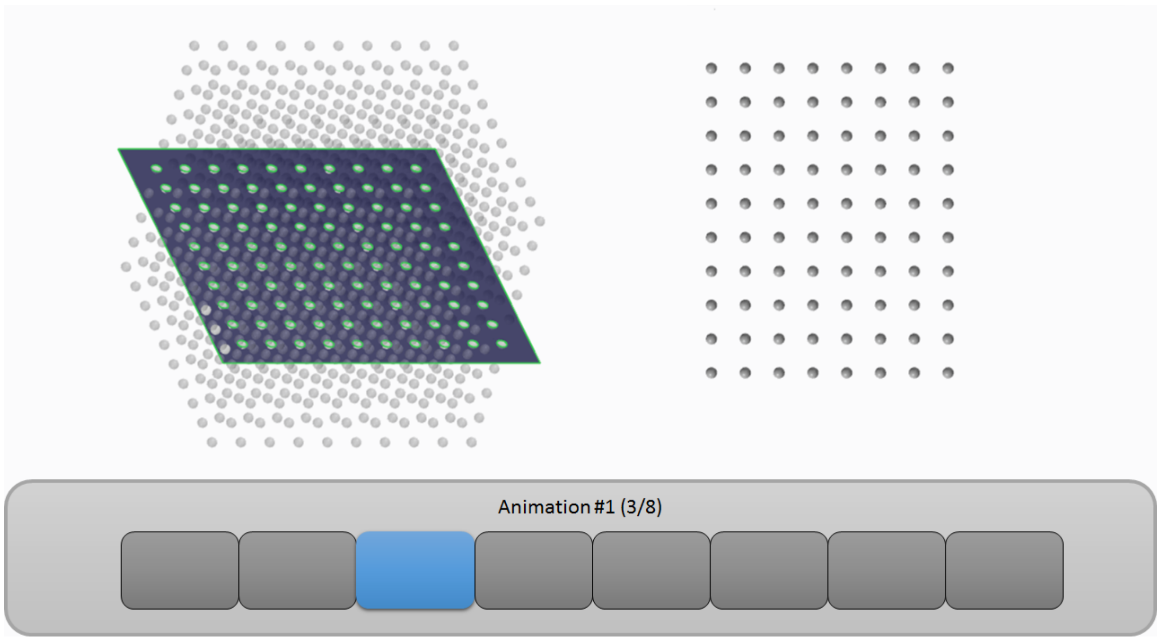


Figure 4.9: Animation Creator GUI

Communication

The communication software will facilitate the transmission of animations from the animation software to the firmware driving the LED cube. This segment of the software should act as a black box and proxy between the animations and the LED cube that displays it. The communication software will utilize multiple transmission modes shown in Table 4.1 to accomplish the goal of displaying an animation on the LED cube. Each transmission mode will allow different features to be implemented within the LED cube animations such as complex animations or high refresh rate animations. Full animations can be pushed to the firmware and stored in memory before displaying for animations of small size. Advantages of transmitting an entire animation in one long transmissions include complex animations that need long compute times, no dependence on transmission rates, and a high refresh rate. Conversely, animations can be sent frame by frame with each frame to be displayed as it is received. This mode allows large, dynamic, or complex animations to be handled by the animation software rather than simply being replayed by the LED controller at the expense of refresh rate. This method will most likely be the most utilized as it should allow animations to be compute in real-time by the animation software and does not require any additional storage resources. Two further modes that make compromises between the above-discussed methods are to alter the currently displayed frame on the LED cube. This is accomplished by either shifting the frame by a layer on any of the six sides and pushing in a new layer, or by defining a rotation or translation of the current frame. Shifting layers into an existing frame allows for the support of animations where the next frame only changes in a minimally from the frame that currently exists on the cube. The rotation and translation of a current frame produces support for various animations with only minimal data transmission. This allows for a high rate of transmission and low bandwidth at the expense of limited animation capabilities and increased use of computational power within the firmware.

Transmission Method	Advantages	Disadvantages
Full Animation	High refresh rate	Small animation sizes
Frame by Frame	Larger, more complex animations	Slower refresh rate
Shift Layer In	Higher refresh rate and lower bandwidth	Slower animation rate
Translate / Rotate Frame	High refresh rate and low bandwidth	Limited animation capabilities

Table 4.1: Transmission Methods

Simulation

Since the design of the LED cube will be completed in parallel with the software development, it is necessary to design a simulator of the physical elements of the LED cube for testing purposes. The simulator will consist of two independent sections to mirror the firmware and hardware components of the LED cube. The firmware simulator will receive the animation transmissions and handle them according to the expected behavior of firmware. In order to appropriately simulate the Ethernet connection between the software and firmware portions of the project, the firmware simulator will also receive data in the same manner as the actual firmware would. The firmware simulator will then pass the information for each frame to a graphical rendering of the LED cube. The hardware simulator will provide a visualization of the LED cube structure in a rendered 3D scene. The hardware simulator will accept data from both the firmware simulator as well as the animation component directly. The use of these simulators will allow for simple visualization of animations and facilitate the testing of both animations and transmission methods during development and throughout all phases of the integration and testing process.

4.1.3 Software Design

While the software architecture describes the high level behavior of all software components of the 3D LED cube, the software design attempts to detail each individual component at a low enough level to completely describe every system within the software. The designs include requirements for how each software component must perform, the explicitly defined behavior of each component, and class diagrams for all relevant components and their data structures.

System Requirements

The software subsystem of the project must transmit the data of the current animation to the firmware component of the LED cube at a fast enough rate in order to ensure the desired frame rate is achieved. For this to occur, the communication component must be capable of transmitting data to the firmware at a rate that is faster than the data is being consumed. Similarly, the animation component must provide new animation data to the communication component at a rate that is faster than the data is being consumed by that module. The simulation components of the software must simulate functionality and behavior of the firmware and hardware components of the LED cube as near to identically as possible. The firmware simulator must receive transmissions from the communication component in the same manner as the firmware. For testing purposes, the hardware simulator must have the ability to receive control data directly from the animation component as well as through the firmware simulator. These requirements will create the nearest solution to the final environment in order to act as an appropriate test bed for the software portion of the 3D LED cube.

Functional Requirements

The communication component will govern the transmission of animations to the other subsystems of the LED cube and will serve as the major bottleneck for the software subsystem. The transmission of lighting control data from the communication component must be capable of filling incoming data buffer of the firmware faster than that data is consumed to drive the LED cube. Therefore, in the following transmission modes, the communication component must guarantee transmission at the minimum rates defined in Table 4.2 in order to prevent a software limitation on the other physical subsystems.

Transmission Method	Data	Minimum Rate
Full Animation	10x10x10xT RGB voxels	5 Hz
Frame by Frame	10x10x10 RGB voxels	25 Hz
Shift Layer In	10x10 RGB voxels + Direction	100 Hz
Translate / Rotate Frame	Translate/Rotate Vector	250 Hz

Table 4.2: Transmission Rate Requirements

The above requirement imposes an addition requirement on the animation component of the software. In order to guarantee the minimum transmission time, the animation component must fill the transmission buffer at least the minimum rate of transmission for the current transmission mode. This means that in the given transmission modes, the animation component can spend a maximum time of computation on the next piece of animation data for transmission defined in Table 4.3. This requirement will potentially limit the complexity of animations that are available when operating in each transmission mode.

Transmission Method	Data	Maximum Time
Full Animation	10x10x10xT RGB voxels	200 ms
Frame by Frame	10x10x10 RGB voxels	40 ms
Shift Layer In	10x10 RGB voxels + Direction	10 ms
Translate / Rotate Frame	Translate/Rotate Vector	4 ms

Table 4.3: Animation Rate Requirements

The simulation component will handle at minimum the rates defined by the firmware and hardware subsystems. All internal datatypes within the simulators must conform to the hardware restraints of the physical systems they are attempting to simulate. The simulation component will also throttle itself to the maximum capabilities of the firmware and hardware subsystems to properly mimic the subsystems. These constraints will allow for the detection of bottlenecks or conditions that exceed project capabilities that may arise during the development process before the integration phase.

Animation

Internally, the LED cube, shown in Figure 4.10 and visualized in Figure 4.11, is a logical map of the 3D physical cube with each voxel representing a RGB LED on the LED cube. The internal representation of the LED cube can be modified voxel by voxel or en masse by changing an entire 2D layer at once. A frame, shown in Figure 4.12, represents the LED cube at one discrete point in time with an associated length of time for the frame to be displayed. An animation will cycle through each frame in the sequence pausing for the appropriate length of time in between frame changes.

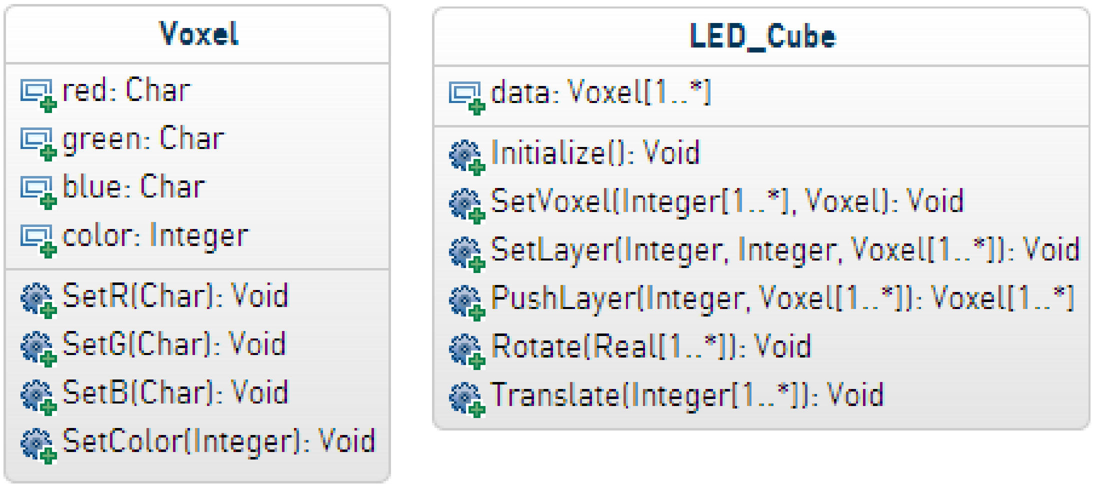


Figure 4.10: LED Cube Class Diagrams

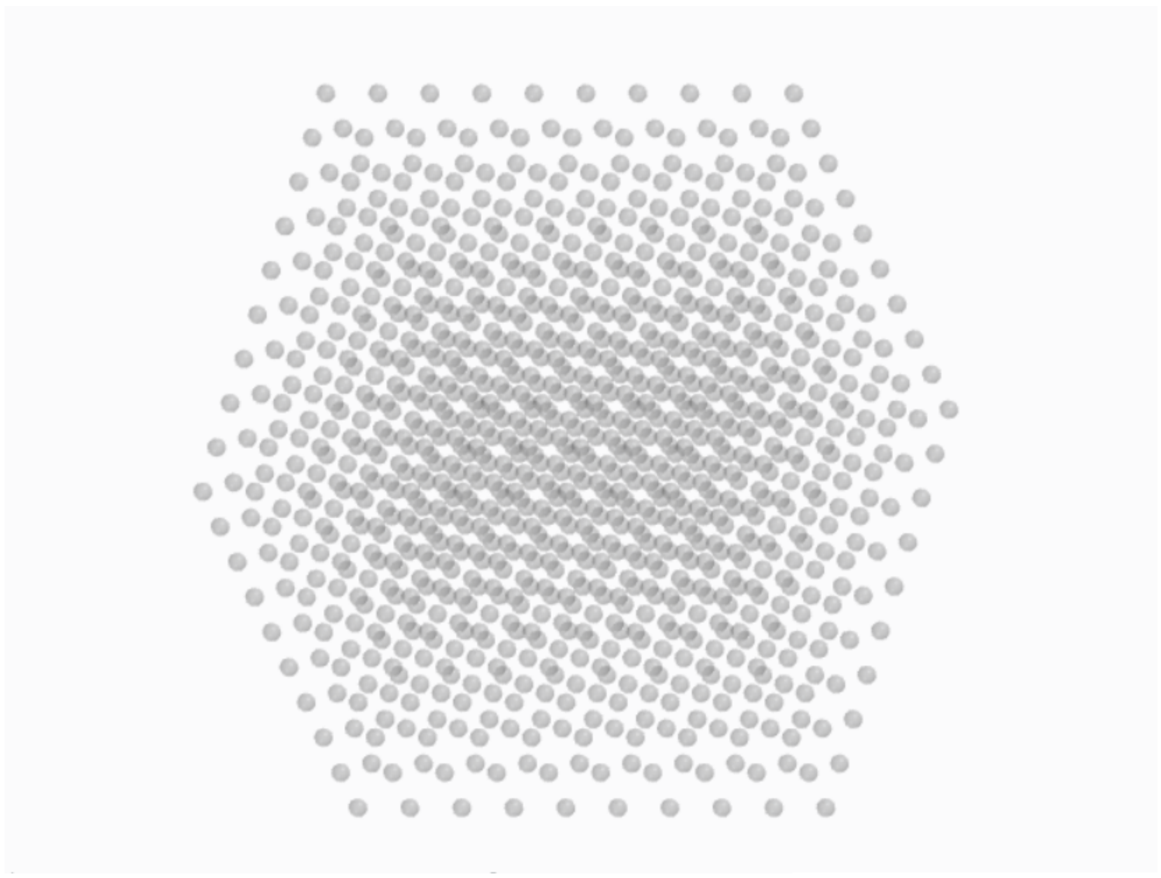


Figure 4.11: Internal Animation Frame Representation

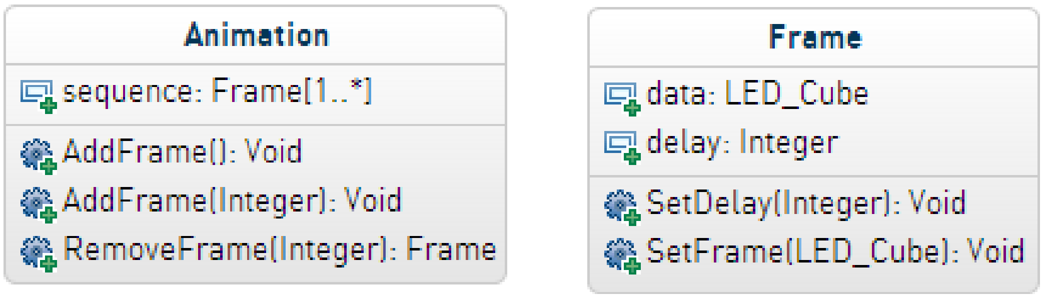


Figure 4.12: Animation Class Diagrams

Animation Creator GUI The animation creator GUI, shown earlier in Figure 4.9, will allow the developer to manually input RGB color values for each voxel within a frame. Each voxel can be addressed in 3D space by its (x, y, z) coordinate or by selecting from the 3D representation with a mouse via the animation creator GUI. To add finer control, a user can select a specific layer along any of the three primary axis, like in Figure 4.13, to edit individually as a 2D representation of the voxels, shown in Figure 4.14. Once the color values for all voxels in the frame and an appropriate delay have been assigned, the user can begin designing the next frame in the animation. The animation creator GUI will allow for frames to be created and removed from the animation at any point within the animation sequence. The rearrangement of frames currently in the animation will also be supported. All values within the animation creator GUI will default to a fixed frame rate and all voxels initially in the off state.

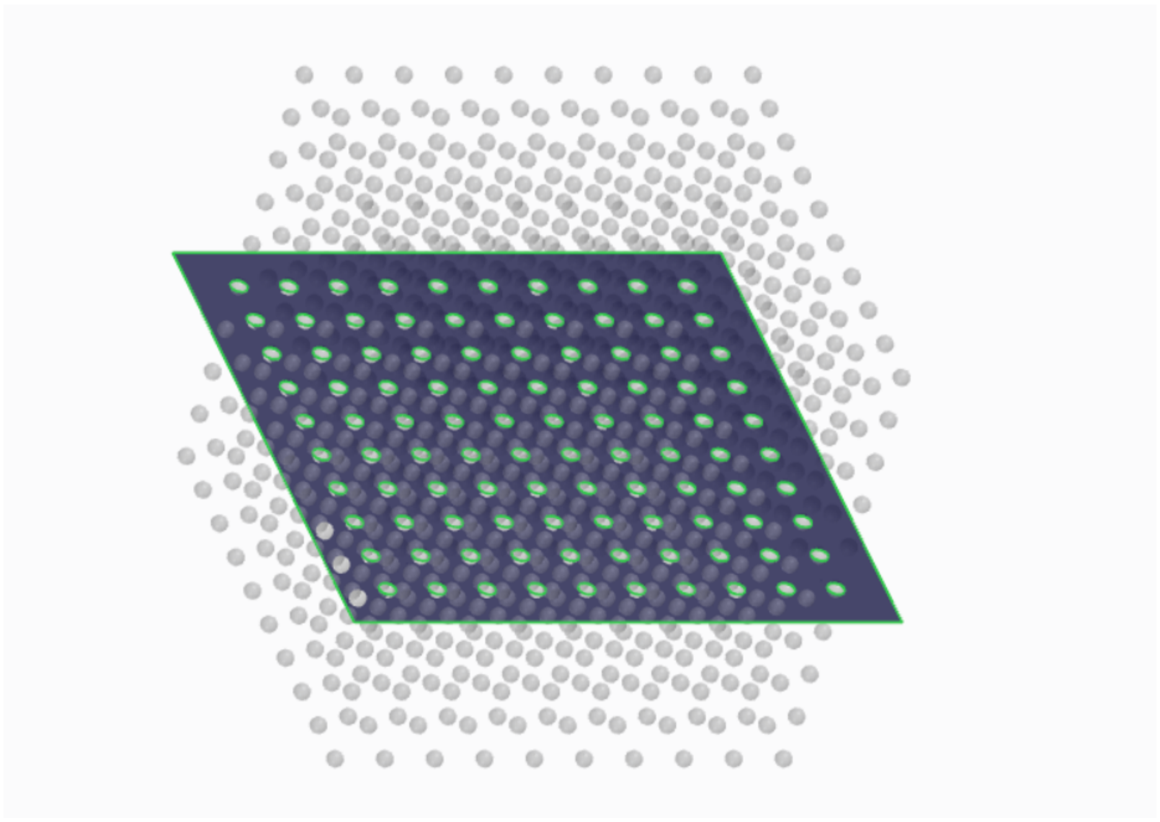


Figure 4.13: Frame Representation with Selected Layer

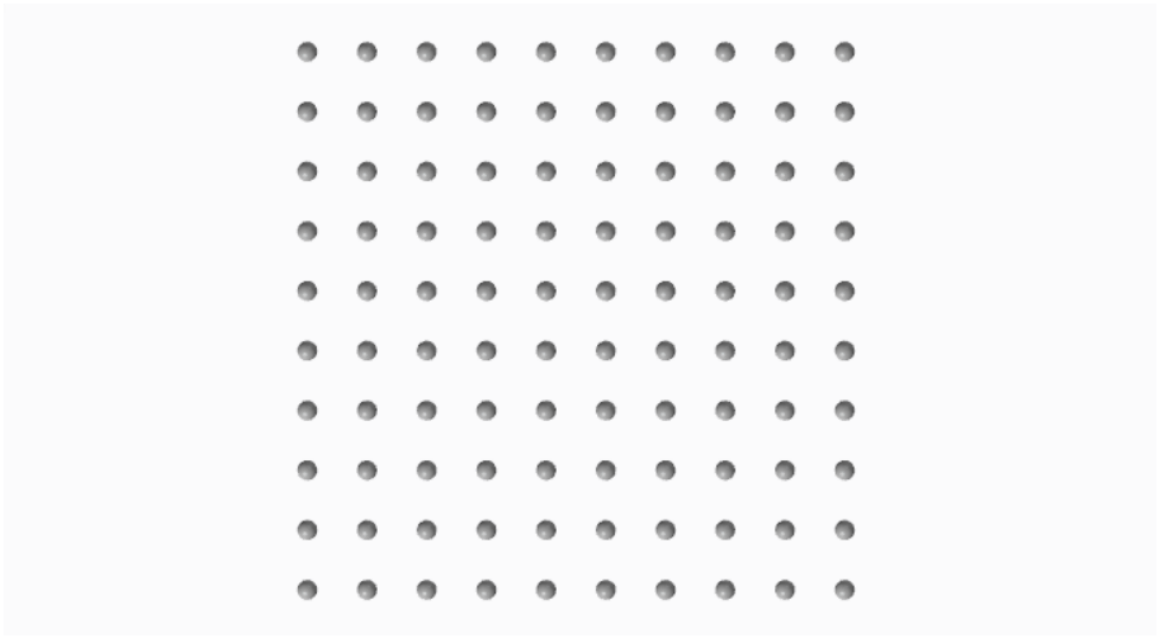


Figure 4.14: Individual Layer Representation

Coded Animation Interface The developer can replicate the animation creator GUIs functionality in a more structured and automated fashion by defining functions to modify the voxels within frames to create animations. Each voxel is addressed by it's (x,y,z,t) coordinate, with the t coordinate denoting to which frame of the animation sequence it belongs. The LED cube of each frame is directly modifiable just as it is within the animation creator GUI. The APIs that the animation creator GUI uses to generate animations is simply exposed to the programmer. The developer can now define frames en masse with an algorithm to create an animation.

Communication

The transmission of data between the software that generates the animation and the firmware that will control the display of the animation is handled by the communication component. Data transmission will occur via the IP suite of protocols over an Ethernet connection. The communication software will transmit the animation data to the firmware using ArtNet, a digital lighting control protocol shown in Figure 4.15. An ArtNet packet contains a header with magic numbers to distinguish it as an ArtNet packet, fields to designate the protocols and versions being used, sequencing information, source and destination addresses, and finally up to 512 bytes of lighting control data. Each of our RGB voxel contains three bytes of color data. This means that a single ArtNet packet can communicate the control information for up to 170 voxels. A full frame contains 1,000 voxels plus an integer to define the time of display and is completely transmitted with six ArtNet packets. The animation software will send full frames or animations to the communications software treating it as a black box. The communication software will manage the segmenting of frames to send to the firmware. The data structures and functionality of the communication component are described in Figure 4.16.

0	8	16	24 31
'A'	'r'	't'	'-'
'N'	'e'	't'	0x00
Opcode ArtDMX (0x5000)		Protocol Version (14)	
Sequence	Physical	Universe	
Length (2-512, even)		Data	
... Data ...			

Figure 4.15: ArtNet packet

The communication component can utilize four transmission methods to transmit an animation to the LED cube driver. All data will be contained in packets of reasonable size for the transmission medium. This first method is to send the entire animation to the firmware. The animation is transmitted to the LED cube as an entire block divided amongst the required number of packets. The communication component can then wait until the firmware requests the next animation. Similarly, an animation can be sent frame by frame. Each frame of an animation is transmitted as an entire block divided amongst the required number of packets. A new frame will be pushed to the firmware at the defined frame rate of the animation. For a low bandwidth solution, animations can be created by modifying the current frame and shifting a layer into the frame from one of the six sides of the cube and shifting out the opposite layer of the frame. The communication component will transmit a layer of a frame and which side the layer will shift into. Analogous to the frame-by-frame method, each new layer is pushed to the firmware at the defined frame rate. The final transmission method requires the lowest bandwidth, as it transmits no frame data. All that is transmitted to the firmware is translational and rotational value, which dictates how the current frame will be shifted within the cube. The firmware will handle the shifting of the frame.

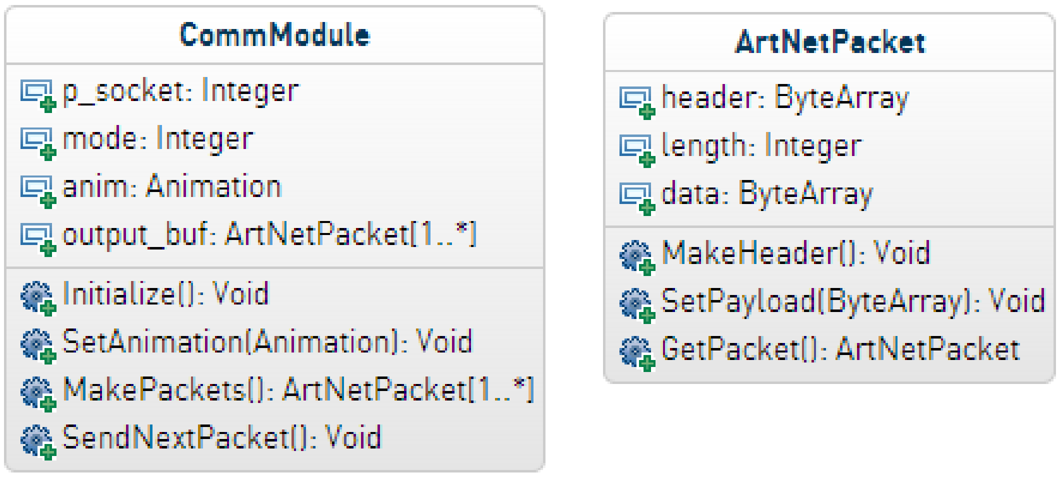


Figure 4.16: Communication Class Diagrams

Simulation

The software simulation components are critical to the testing process while the physical LED cube is either in construction or not available for testing. The simulators must behave identically to the physical components they are simulating. Therefore, the design of the simulator software's data structures and routines should be similar to the physical components. The firmware simulator will act as the firmware controller for the 3D LED cube and the hardware simulator will act as the LED cube itself. The hardware simulator will be unique and not fully representative of the actual behavior in that developers can access it either through the firmware simulator or directly through exposed API.

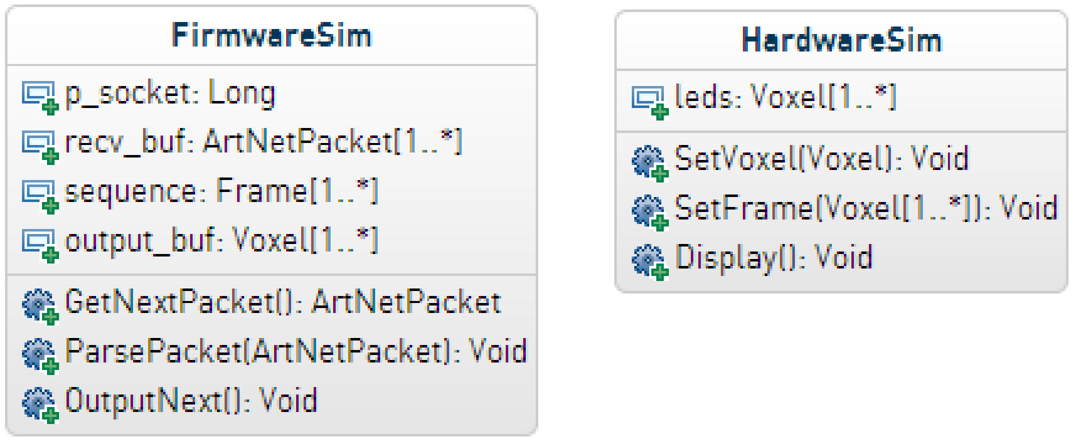


Figure 4.17: Simulation Class Diagrams

Firmware Simulator The firmware simulator is designed to simulate and test the communication between the software and firmware components. In order to properly simulate this, no data will be directly passed to the firmware simulator through API. Rather, all data will be received by the firmware simulator via ArtNet packets over an Ethernet interface. After the simulator receives a packet, it is added to an incoming data buffer where it will wait until the simulator is able to handle the packet. The simulator will parse packets from the incoming buffer and place completed frames into a buffer of frames that acts as the display sequence. As frames are scheduled to be displayed, the frame will be decomposed into a buffer of voxels to pass onto the hardware simulator. Finally, the simulator will control the output of the hardware simulator.

Hardware Simulator The hardware simulator will display a 3D graphics representation of the LED cube. The LED cube will be simulated as a 10x10x10 cube of colored spheres laid into a 3D scene. Graphics rendering will be handled with OpenGL to create a scene that a user can rotate and view from all angles. The hardware simulator will allow individual voxels to be set as well as entire frames and will refresh only when prompted to update the display. All control of the hardware simulator is through exposed API and data will be transmitted directly through them to simulate the direct links between the firmware and LED controlling hardware. Although meant to simulate the behavior of the 3D LED cube, the hardware simulator will accept data directly from the animation software in order to isolate the animation testing process from the communication method testing.

4.1.4 Printed Circuit Board

The project will require the design and assembly of two circuit boards to hold and connect all of the components necessary to illuminate and control the vast number of LEDs on the cube. Collectively, the group has some experience in designing circuit boards using Cadsoft's Eagle editor. The group will use this software again to design the circuit boards in order to save time having to learn a new editor, while designing a complex circuit board. The group hopes that with the availability of industry reference designs, and industry advisors, that their circuit board will be designed with ease, and will operate with efficiency. There are many considerations that the group will have to make when designing and fabricating a circuit board of this complexity, some of them being trace thickness, grounding planes, trace length, component placement, bypass capacitance, and radio frequency emissions. The group hopes to balance all of these factors in order to create a well designed, high quality, functional circuit board. The group hopes to be able to prototype as much of the design as possible in order to reduce the possibility of errors within the circuits, and increase the reliability and confidence in the designed circuit.

Printed circuit boards, like any other method of connecting voltage signals, will dissipate power due to the resistance of the materials used. It is necessary to use the widest possible traces for any component that will be carrying a great deal of power, and smaller traces for those that will be carrying just signals and communications. It is also necessary to make sure that the impedance of the circuit traces are within reason. Each signal will have a small amount of resistance, capacitance, and inductance, that will affect how the circuit chips are able to pass voltage along them. The longer a trace is, the more resistance it will have. This resistance will affect how much current is able to pass from one chip to another. Also, with increased trace length, the capacitance of the trace increases. This capacitance causes the voltage to rise more slowly, and decreases the slew rate of the output of whichever circuit is driving the trace. This limits the clock rate as the circuit needs more time to stabilize. The team will keep these factors in mind when they are designing the circuit board, and will make every effort to keep trace lengths as short as possible. Another important aspect of designing traces is how the traces are bent. Very few circuit boards are able to wire all traces as straight lines, some bends will be essential. Bends however can create manufacturing difficulties, due to the fact that etching used to create the PCB may pool in a corner trace. Also, corner traces create micro sized antennae that can transmit radio frequency interference, which is a very undesirable trait for any circuit. For this reason, the group will avoid using any 90 degree bends within the circuit, and will instead use other degree angles.

The design of the printed circuit boards will be all completed within the design software Eagle, released by Cadsoft. The group has some familiarity with the program, as well as its popularity with the hobby and hacker community due to ease of use, and low cost helped the group make their decision to use this program. The group has, at this time, completed all of the schematics for the project necessary, and will use those schematics in order to begin laying out the circuit board. Using this method, it will be sure that the group has hooked all components up to the proper supply voltages, as well as the proper components are connected to each other. The group will then use the design rules checker utility within Eagle to ensure that the circuit board is routed such that it will be able to be successfully manufactured by the circuit board house chosen.

One of the largest factors often overlooked by the novice circuit board designers, and experts alike is the form factor of the circuit board to be produced, and how it will be mounted to the enclosure that it is in. The group wishes for all of the circuit boards to fit within the LED cube base, and will mount them to the system via plastic standoffs that mate to mounting holes on the circuit board. The group will design and place componenets on the circuit board such that the circuit board will be six inches in one dimension at max. The group wishes that at this size, the circuit board will be easy enough to work with, but will not overcrowd the area beneath the LED cube.

4.1.5 Physical Structure

The structure of the cube will be comprised of 18 gauge pre-tinned copper wire. This will provide both the electrical connections for the anodes and cathodes of the LEDs as well as the structural support for the cube. The wire is sold in spools of 100s of feet, with approximately 2000ft of wire needed for this construction application. Each piece of wire will be cut into lengths of approximately 60cm each. The wire will be stretched to straighten the wire and increase its strength, preventing the wire frame from becoming bent. In the 3-dimensional coordinate system of the cube, each x-y plane will be soldered to the anodes of the LEDs, while the wires in z plane will reference the cathodes. The pitch, or how far each LED will be spaced from each other, will be 6cm. When the cube is constructed, it will be placed on a wooden base that will house the control hardware: control boards and driver boards. The outside of the LED cube will have acrylic paneling that attached to the base, protecting the delicate LEDs and semi-delicate copper wire structure.

Chapter 5

Design Summary of Hardware and Software

This chapter summarizes the hardware and software designs as well as their interactions with each other. A high level summary of all systems is described through the use of block diagrams. A summary of functionality is provided for all major software components. Data flow is defined from creation, through transmission, all the way to final display on the 3D LED cube. High level descriptions and visualizations of features, interactions, and data flow are provided to further define the inner workings of the project.

5.1 High Level Design

The highest level description of the 3D LED cube includes only it's most general components. Figure 5.1 describes this by listing the signal flow from the user input to the visual LED output. After the user inputs a command into the software, via a GUI, the software will communicate the details of the animation to the embedded processor on one of the PCBs. The embedded processor will control the on-board FPGA, which will operate the row and column circuitry to update the animation on the LED cube frame-by-frame. Figure 5.1 shows this signal flow of the 3D LED cube at the system level.

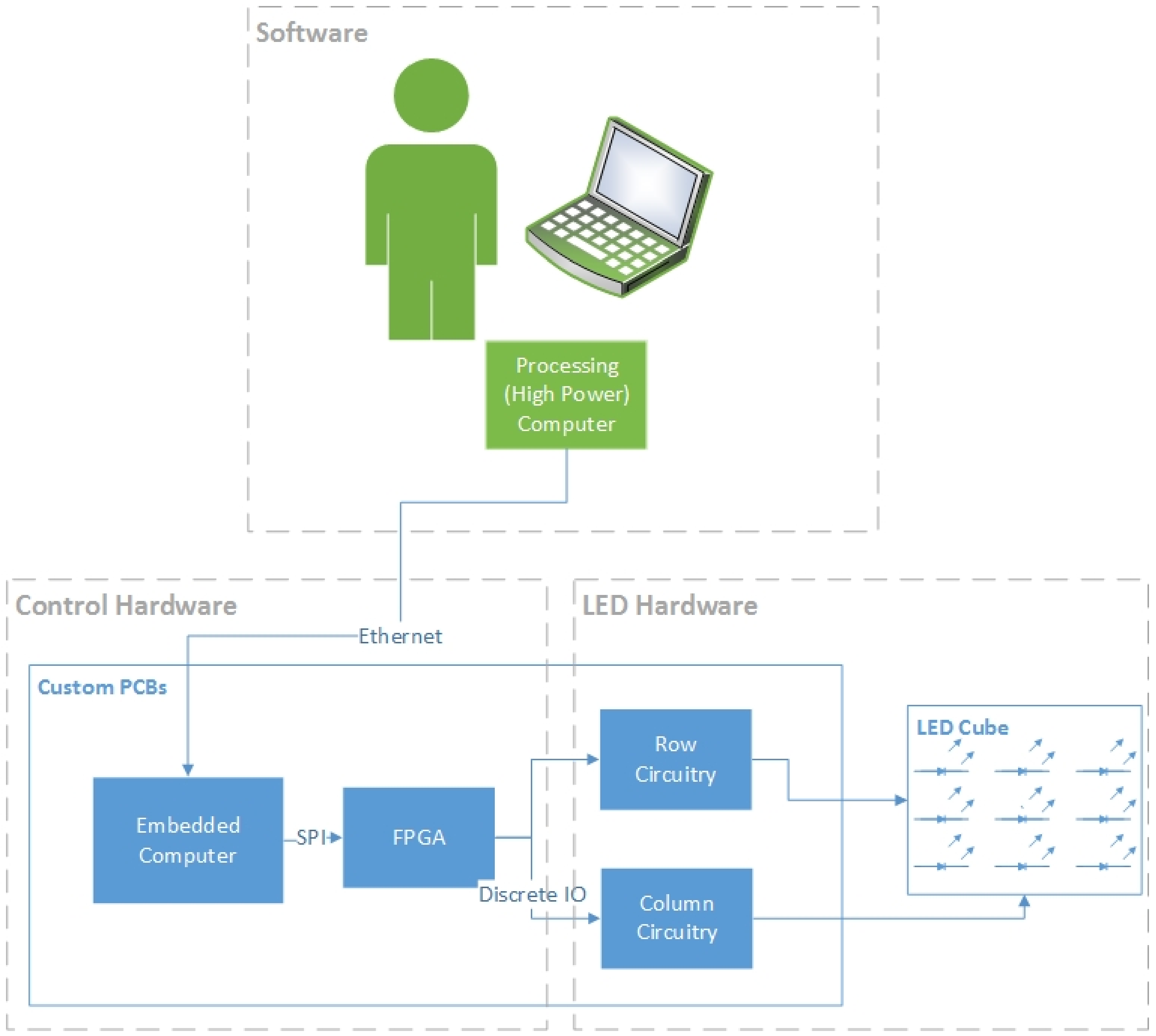


Figure 5.1: LED Cube Block Diagram

5.2 Hardware

The high-level description of hardware only includes more detail than the general description. Figure 5.2 illustrates through the use of a signal flow block diagram the operation of the hardware components of the 3D LED cube.

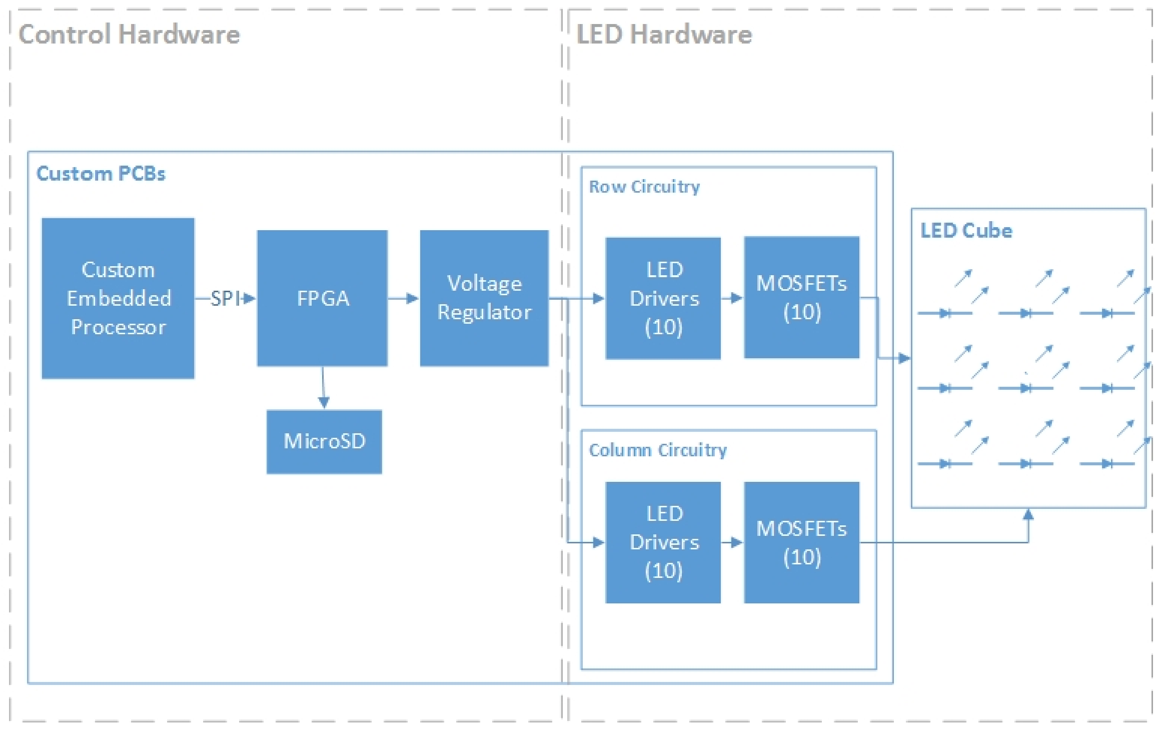


Figure 5.2: LED Cube Hardware Block Diagram

5.3 Software

The software section of the project is divided into three major components: animation, communication, and simulation. The animation component generates animation sequences for display on the LED cube. The animation component is further divided between the Animation Creator GUI and a Programmable Animation Framework. Each provides developers and users with powerful interfaces with which they can create animations for display. The communication component transmits all lighting control data necessary for driving the animations to the firmware subsystem. The transmission modes supported by the communication component allow for a vast set of animation options for utilization in the 3D LED cube. The communication component appears as a black box for the animation components to send animations to the LED cube. The simulation component uses both a firmware and a hardware simulator to simulate the behavior of the physical components of the project. Both simulators seek to simulate the behavior of the physical subsystems as accurately as possible, but they will also allow for features that do not exist in actuality for additional debugging options. The simulators will serve as a test bed for the software development process. Developers can use both simulators to test features in software before the integration and testing phase of the project. The software subsystems and their interactions with each other are shown in Figure 5.3.

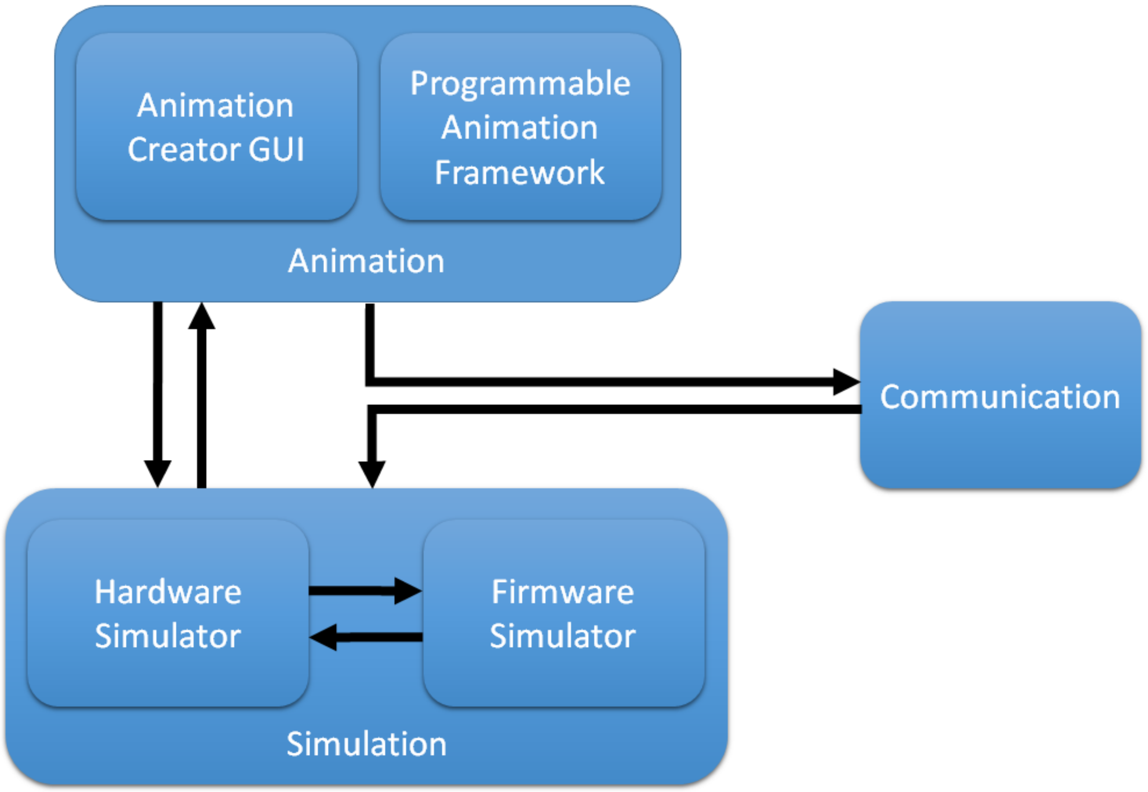


Figure 5.3: Software Summary

5.3.1 Animation

The animation component controls the generation of animations to display on the LED cube. The animations serve as the most visual representation of software portion of the 3D LED cube , and as such the developer must have power tools and interfaces to create animations. Animations are generated manually through the use of an animation creator GUI to create detailed animations frame by frame or by utilizing the exposed API to edit the voxels in the LED cube of each frame in an animation though code to automate the process.

Animation Creator GUI

The animation creator GUI provides a user friendly interface in which they are able to generate highly detailed and intricate animations. Through the animation creator GUI users are able to combine the basic building blocks of frames together to create a full animation. Through the interface, users are able to view a 3D representation of each frame as an LED cube and edit the color content of each voxel within the frame. The process of selecting individual voxels is aided by the ability to further select a layer of the frame from any of the three primary axis and edit voxels within a 2D plane which is more simply visualized. Both the 2D and 3D representations are displayed side by side to give the user a full view of the changes they are making and how they are affecting the current frame. The user is able to add additional frames to the animation and edit them in much the same manner. A final feature to the animation creator GUI is the ability to add, edit, and remove frames from the animation sequence. The frames will be displayed in the GUI in a queue that details the order and duration of each frame within the current animation sequence. Therefore, the animation creator GUI allows a user to create and modify animations through the manipulation of the LED colors within a frame, frame order, and frame duration.

Programmable Animation Framework

Each animation when broken down completely is simply a collection of voxels organized and held in memory. The animation creator GUI seeks to remove the complexity of this from the user by adding an extra layer of abstraction to the process and displaying the animation creation process in a user friendly manner. For a developer that is familiar with the API used to control the animation creation process though, many complex and interesting animations can be created simply by utilizing the same API to achieve a similar but automated animation creation process. The developer will maintain a list of frames within the animations and add frames to the animation as they are created. The voxels of each individual frame will likely be assigned according to a defined algorithm to produce a desired effect. This process can be repeated allowing for many vastly different animations to be created by simply modify the algorithm used to designate voxel colors. Through the programmable animation framework, a developer is able to quickly create a large set of animations with only a limited amount of effort expended in design.

Animation Controller

The animation controller provides a user friendly interface in which they are able to combine, edit, and display animations on the LED cube. Through the animation controller users are able to combine animations as well as edit select variables to modify the behavior of animations to create a larger or more complex sequence of animations. The animation controller allows for animations to be sequenced together to form larger sequences and behaves similarly to the animation creator GUI but using full animations instead of frames. Various implemented animations contain a set of user modifiable variables that alter the behavior of the animation. These variables, such as text to scroll across the cube, are set within the interface to generate the animation sequence for display. Just as the animation creator GUI abstracts the programmable animation framework from the user, so does the animation controller from the communication component. The animation controller allows a user to choose animations for display on the 3D LED cube without having to use the communication component directly.

5.3.2 Communication

The communication component controls the transmission of animations to the LED cube itself and acts as a proxy between the software and the firmware. This component will act as the most critical portion of the software portion of the project. Without proper communication between the animation software and the physical components, none of the animations will be properly displayed, if at all. Transmission of lighting control data is handled through multiple transmission methods which provide unique feature sets and performance tradeoffs to allow for a multitude of animation features. All lighting control data is transmitted over a wired Ethernet connection using the Art-Net protocol. Communication methods include the transfer of an entire animation, the transfer of individual frames, and transmission of data used to transform the current frame. Each method features sets of advantages and disadvantages that make them applicable to certain sets of animations. The transmission of an entire animation sequence allows for complex animations to be transmitted to the firmware without concern for transfer rate. This will allow for a high rate of playback with the limitation of memory available to the firmware component. Transmitting single frames and displaying them immediately after a full frame is received allows for dynamic animations to be displayed on the LED cube in real-time. The refresh rate of the animation is limited to the transmission speed of the communication software. Finally, in situations where the current frame remains largely the same, small modifications to the frame will be communicated to result in a low bandwidth solution. Single layers can be transmitted and pushed into the current frame from a designated axis allowing for simple scrolling of objects within frames. Similarly, an object within a frame can be rotated without the need to retransmit an entire frame. The combination of all transmission options, shown in Figure 5.4, allows for a robust communication feature set that expands the animation options for the developer greatly.

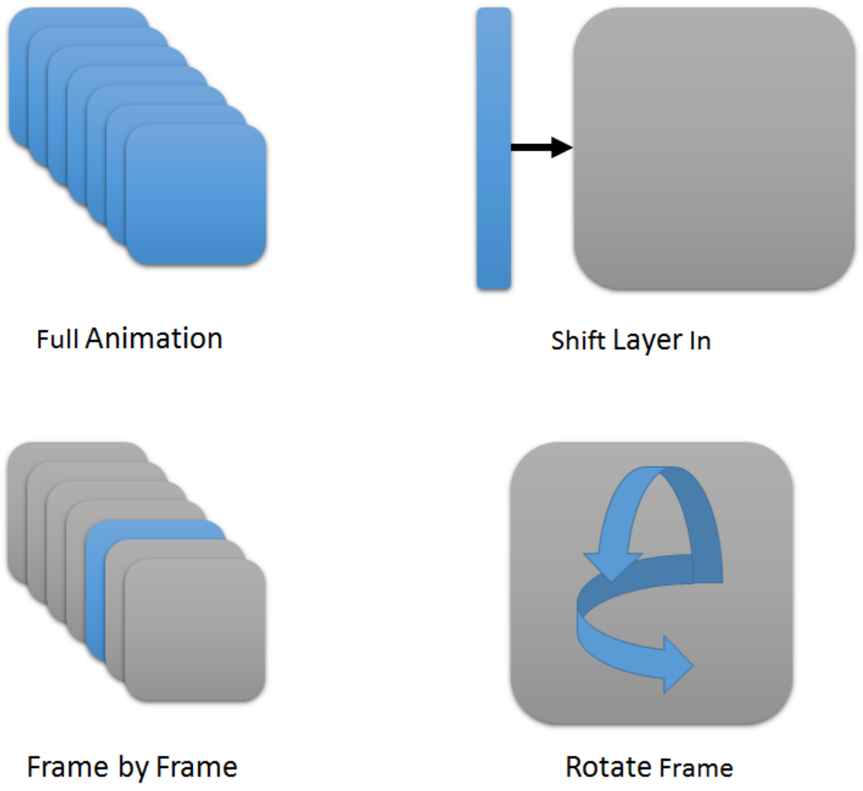


Figure 5.4: Transmission Methods

5.3.3 Simulation

The simulation component allows the developer to test all software components during development. Due to all parts of development occurring concurrently, the all components of the LED cube may not be available for testing throughout the entire development process. For this reason, a method to test the individual components of software without physically having the LED cube is extremely advantageous. The LED cube simulator consists of two components to simulate both the firmware and hardware components of the LED cube that behave identically to the physical components of the LED cube itself for testing. In addition to the tradition behavior that exists in the physical subsystems, functionality is added to incorporate additional debugging options. The flow of data between the software components and the simulators is shown in Figure 5.5. The proper use of software simulators will create an opportunity to catch bugs in the code during development prior to the integration and testing phase of development and reduce overall rework time.

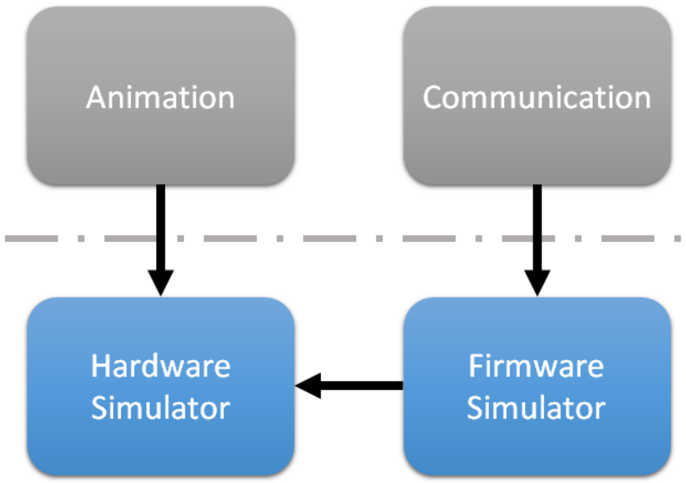


Figure 5.5: Simulator Data FLow

Firmware Simulator

The firmware simulator acts to replicate the behavior of the firmware subsystem of the project which controls the LED cube given input from the software subsystems. This specific simulator serves to test the functionality of the communication component of the software. In order to properly simulate the behavior of the firmware subsystem, the firmware simulator will receive transmissions in the same manner as the firmware component. The firmware simulator will receive all lighting control data by listening on an Ethernet interface. No lighting control data will be passed directly to the simulator through memory, as this would not accurately simulate the link between the two components. The firmware simulator will support all transmission modes of the communication component and will simulate the physical components limitations in receiving and maintaining that data in memory. After reception and decoding of the lighting control data, the firmware simulator will behave similarly to the firmware itself and control the LED cube simulated by the hardware simulator.

Hardware Simulator

The hardware simulator serves to visualize animations as they would appear on the 3D LED cube. The hardware simulator displays a 3D rendering of the LED cube that users are able to manipulate to view from all possible angles. Through the simulated display, developers are able to verify correctness of animations and individual frames. The hardware simulator will accept input from the firmware simulator to recreate the behavior that exists within the physical subsystems. In addition to this, the hardware simulator will allow for input directly from the animation software, a behavior which does not exist in the physical subsystems. This direct input gives developers an opportunity to debug animations directly. The hardware simulator will act as an engineering tool to improve the development process of animations and the rest of the software subsystem without the need for any of the physical components.

Chapter 6

Project Prototype Construction and Coding

With a project as large as the LED Cube will be, a plan for assembly and building is necessary. The following chapter will outline the acquisition and completion of the building of the LED cube. The group will explain and provide details for the hardware, firmware, and software completion of the project, and the methods used to ensure that the cube is built to specification.

6.1 Parts Acquisition

As each part for the prototype has already been chosen, it is left to describe how the parts will be obtained. Table 6.1 outlines how each item necessary for the design and prototype build will be acquired.

Component	Method Acquired
LEDs	Sponsored (Free)
Wire	Purchased
Embedded Computer	Designed/Purchased
FPGA	Purchased
PCB	Designed/Purchased
LED Drivers	Sampled (Free)
MOSFETs	Sampled (Free)
Analog to Digital Converter	Sampled (Free)
Power Supply	Purchased
Processing Computer	Owned (Free)
Frame & Case	Built/Purchased
Building Supplies (plywood, screws, etc)	Purchased
Acrylic Panels	Purchased

Table 6.1: Component Acquisition

6.2 Bill of Materials

The bill of materials lists each specific manufacturer and part number for all the primary components that will be utilized for the prototype build. Smaller items, or parts that will be obtained on a need basis throughout the project will not be included. Table 6.2 outlines how each item necessary for the design and prototype build will be acquired.

6.3 Printed Circuit Board Construction

The 3D LED cube will be compromised of two unique circuit boards, each of which will be sent off for professional manufacture, and will be assembled by the group. The group has some experience in designing and soldering circuit boards, however the completion of boards as complex as these will be a nontrivial task for the group to complete. The group will allocate a large amount of time in order to be sure that they are not rushing through the assembly and testing of the circuit boards in order to assure that none of the components are damaged during assembly. The group will follow all standard static electricity safety procedures, and will use proper grounding procudures when working with static sensitive devices. Overall, the group hopes to take every precaution necessary to ensure that the circuit boards are assembled problem free.

The assembly of each of the circuit boards will be completed by the group. The group has some experience in soldering surface mount components on the order of the size that the design will be using. The group will have to use a variety of methods for properly soldering each of the components to the board. The group desires to be able to complete the assembly in the most efficient, safest, and easiest way, ensuring that the quality of any of the components used are not destroyed, and that any and every specification put in the manufacturer provided datasheets are followed. The group will use a hotplate method for soldering all of the large surface mount chips on the first side of the circuit board, and then will use a soldering iron, and possibly a hot air rework station for soldering the other surface mount chips. The group will use a combination of solder paste, and solid solder, both unleaded for the ease of assembly and soldering. The group anticipates the construction of the circuit boards taking only approximately two days in total.

As with any project, the group understands that there will be a need to have spare parts, that are easily interchangeable. The group hopes that nothing will fail within the system, but there is always the possibility for failure. The group will order spare parts of each of the parts put on the circuit board, and will be able to provide repairs to the circuit should a component fail, and will also understand and analyze why the component failed, and if a substitution needs to be made to prevent future failures. The group also hopes to create one full spare circuit board of each of the driver, and the control designs, such that the

Component	Quantity
Custom LEDs	1200
18 Gauge Pre-Tinned Copper Wire	2000ft
PIC24HJ256GP206A	1
XC6SLX9	1
Printed Circuit Board	3
TLC5948A	20
SUD45P03-10-E3	20
REG1117	2
ADS1278I	1
8MHz Crystal Oscillator	2
SP-150-5	1
Lenovo Thinkpad W530	1
RJ45	1
MicroSD connector	1
ENC624J600	1
Status LED	10
4 Pole Dipswitch	1
100uf Capacitor	6
10uf Capacitor	1
.1uF Capacitor	56
1nF Capacitor	36
10nF Capacitor	2
6.8nF Capacitor	2
18pF Capacitor	4
100kΩ Resistor	1
12.4kΩ Resistor	6
4.7kΩ Resistor	8
330Ω Resistor	1
100Ω Resistor	4
75Ω Resistor	4
10Ω Resistor	1
1015Ω Resistor	25
Acrylic Panels	5
Building Supplies	Assorted

Table 6.2: Bill of Materials

group will be able to avoid last minute item failure, and stress within the scope of the project completion. Should a circuit board fail, the group will be able to replace it within minimal time, and demonstrate a working, easy to maintain product.

6.4 LED Cube Construction

The construction of the LED cube will be completed in several stages. The cube itself, or the actual 1000 LEDs and their wiring/structure, will be completed plane-by-plane. Each plane of LEDs will be soldered to their respective joints made up of 10x10 crossing pieces of 18 gauge pre-tinned copper wire, carefully spaced based upon the required pitch of 6mm. The spacing will be accomplished by building the plane flat, on a panel of plywood. Screws will be placed in the plywood to direct the spacing of the 18 gauge wire. This stage of construction is illustrated in Figure 6.1. The LEDs will be soldered to the wire plane structure over the plywood. Finally, upon the completion and testing of a panel, the panel will be integrated into the other panels, one by one, to form the cube. This will be accomplished by placing long, thin pieces of plywood off the side of a bench, the wood steadily secured with screws to the bench. Each plane of wire and LEDs will then be slid perpendicularly over the pieces of wood sticking out of the bench. As the planes will hang securely upon this wooden jig, the placing between planes can also be precisely controlled. It is from this position that the 10x10x10 LED cube will then be attached to the base of the cube. This stage of the design is illustrated in Figure 6.2.

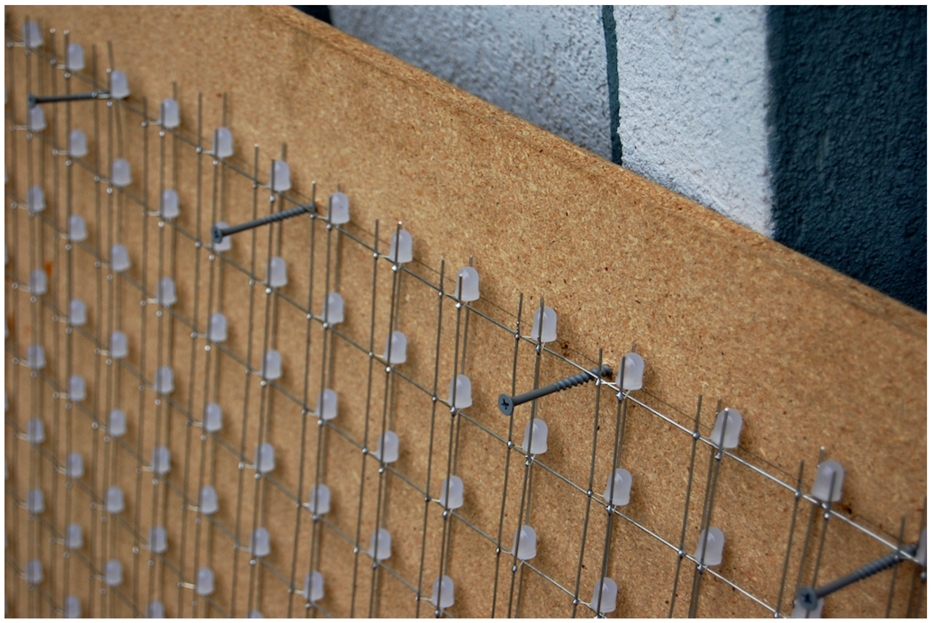


Figure 6.1: LED Plane Construction

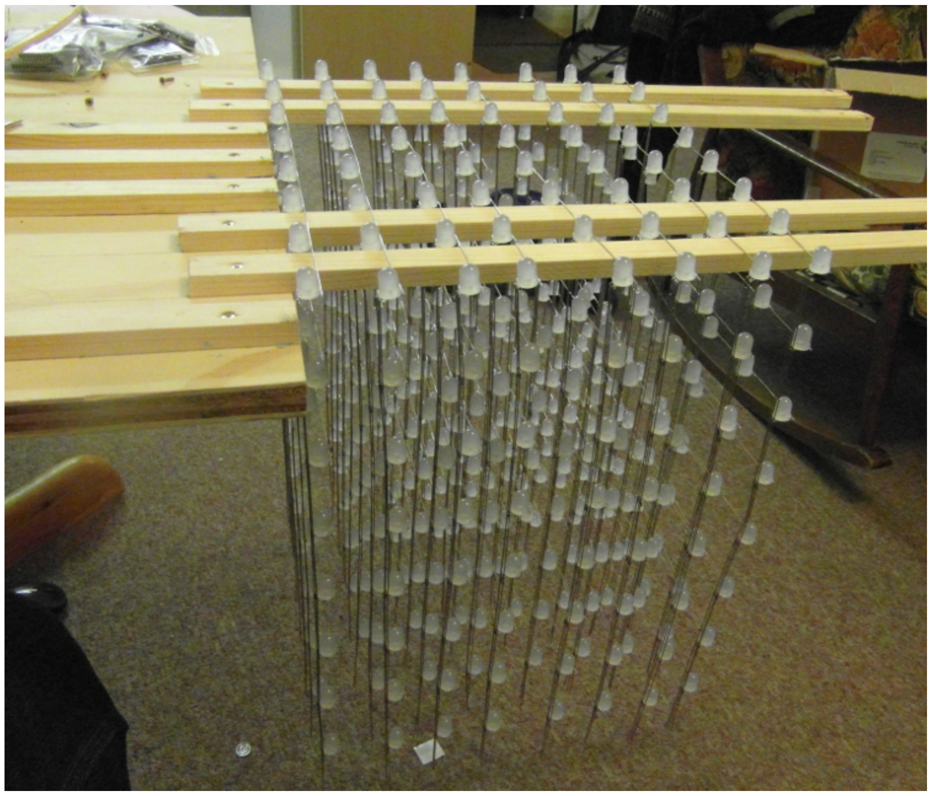


Figure 6.2: LED Plane Integration

The base of the cube will be built separately, approximately 2.5ft W x 2.5ftL x 1ft H. It will also be built with plywood, secured together with screws. The control hardware will be stored inside the base of the cube, and connected through the base to the LED cube structure.

6.5 Software Implementation

The software subsystem of this project provides critical functionality to the 3D LED cube. The software development process will begin by defining the most basic data types and functions as building blocks. Development will continue with the implementation of software components and features according to the software development schedule described below. The software development schedule prioritizes components and features by criticality, dependencies, and a measure of investment versus reward. The development of core functionality is of the highest priority, followed closely test bed environments for integration testing, and finally after all major software development milestones are met, the development of animations and additional features can begin. The software development process includes integration testing between software components and through simulation at all major milestones in development. This will serve to reveal bugs and errors in the software at early stages of development and hopefully reduce the impact of errors and integration issues late in the development cycle when time left for development will become critical.

6.5.1 Components

The software portion of this project is divided among the distinct components. All of which are critical to the development process, though only some of which are necessary for the completed project. The development process can be seen in Figure 6.3. The software classes defining the LED cube itself are the most critical as they are present in all software components. After the basic building blocks are constructed, the framework to generate animations can begin. The hardware simulator development will follow to serve as a test bed and visualizer for animations. The hardware simulator will also serve as a critical component to test and utilize the firmware simulator, so its early development serves a dual purpose. The communication component and firmware simulators will be developed concurrently next. Both components are fundamentally linked and no other components depend on their development, so they do not suffer from late development. Finally, once all other critical components and their respective test suites are developed, development of the animation creation GUI can begin as the last major objective. Integration between developed components will be tested at all points throughout the development process to ensure that all components function properly together.

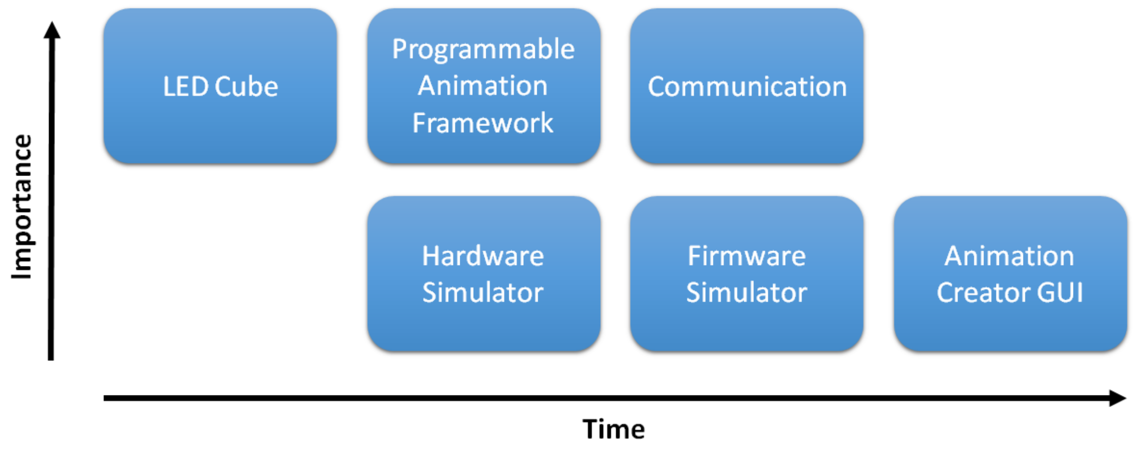


Figure 6.3: Software Development Process

LED Cube

The most essential software components to the entire software subsystem are the LED cube and voxel implementations. The functionality of both classes will conform with the class diagrams described in Figure 4.10. A voxel is simply implemented as a structured data type with functions to operate over the fields. All three 8-bit color channels of the voxel are implemented as a single 32-bit integer as shown in Figure 6.4. A value of 0x00 in the channel signifies no color is being produced from that channel, while a value of 0xFF signifies full strength color provided from that channel. The functions SetR(), SetG(), and SetB() will assign a single byte to the respective color channel location in the color value. The SetColor() function will assign all three color channels at once. GetR(), GetG(), GetB, and GetColor() functions are additionally implemented to access specific values from the structure. The use of macros can enable all these functions to be implemented as inline functions with bitmasks.

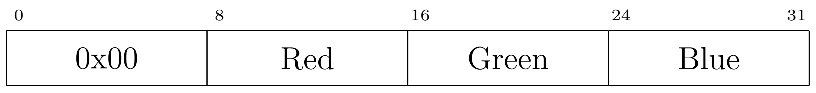


Figure 6.4: Voxel Datatype

The LED cube is represented as a 3-dimensional array of voxels. The array is implemented with the x-coordinate being the slow scanning index, the y-coordinate falling in between, and finally the z-coordinate as the fast scanning index. Each voxel will correspond to a unit coordinate along the positive xyz axes based at the origin. When initialized, all voxels will have color values of black, or simply equal to 0x00. A developer can set the color value of a voxel directly with the SetVoxel() function by referring to its 3D coordinate and the voxel color. The color values for an entire layer of the cube can be similarly assigned with the SetLayer() function by passing a layer of voxels, to which axis the layer belongs, and specifying a coordinate to hold constant. The PushLayer() function treats the voxel array as a 3-dimensional double ended queue where a layer can be pushed in from any cube face and the rest of the layers will shift to accommodate, ultimately resulting in the layer on the opposing face to be shifted out of the voxel array. The Rotate() function will take a 3-dimensional vector specifying the cube's rotations about each of the three primary axes centered at the middle of the cube. All voxels will be mapped to the closet unit coordinate within the cube after rotation and unmapped values will be assigned the color value of black. The Translate() function is implemented similarly to the Rotate() function with the exception that the vector specifies translation along, not rotation about, each of the three primary axes.

Animation

The animation component implementation is based on the classes documented in Figure 4.12 and largely depends on the data types defined in Figure 4.10. An animation is comprised as a sequence of frames. A frame contains the color data of all voxels in the LED cube and a delay describing the length of display. The LED cube data in frame is only set to a frame with the SetFrame() function. All manipulation of the contents of the frame is handled through the LED cube class. The SetDelay() assigns the duration for the frame to be displayed. Animations are held as a list of frames. The animation classes implements two methods for inserting a frame into the animation sequence. The AddFrame() function implicitly inserts the frame at the tail of the animation sequence, while specifying an index to the same function will insert the frame at that index of the sequence, pushing the rest of the sequence back. The RemoveFrame() function will remove a frame from the sequence and return that frame. Using the RemoveFrame() and AddFrame() functions, a developer can reorder frames in an animation.

Communication

The communication component lies between the animation component and the physical systems to transmit animations as lighting control data through an implementation described in Figure 4.16. Data is transmitted within an ArtNet packet defined in Figure 4.15. The MakeHeader() function assigns the default values of the packet header. The SetPayload() function assigns the lighting control data to the packet. The function will also ensure that the data is bounded between 2 and 512 bytes and padded with a zero byte to ensure an even length if necessary. Once both functions have been called, a completed packet ready for transmission is returned with the GetPacket() function. The communication class must maintain all necessary information to communicate animations. A network socket attached to the Ethernet interface and the transmission mode are both initialized during the Initialize() function. The animation component passes the animation data through the SetAnimation() function. Depending on the transmission mode, the MakePackets() function outputs the next set of ArtNet packets to an output buffer for transmission. Transmission of lighting control data is handled via the SendNextPacket() function. The functionality of the class is handled within a main thread that accepts animation data as it appears from the animation component and transmits ArtNet packets as necessary to the physical subsystems.

Simulation

The software simulators will be implemented as two separate subsystems described in Figure 4.17. The firmware simulation will stand alone from the rest of the software, while the hardware simulation will allow for direct interfacing from both the firmware simulation and the software components. Development will stress conformity to the real limitations of the systems they simulate. While accuracy to the actual systems is necessary, the simulators will implement additional features to aid in debugging. The simulator implementations will accept input from the software components just as the physical components would and their functionality will appear transparent to the software components.

Firmware Simulation The firmware simulator receives all lighting control data through ArtNet packets over an Ethernet interface. Once initialized, the firmware simulator will maintain a socket connection to the interface where it will listen for incoming data with GetNextPacket() function. All incoming packets are placed into a received packets buffer until the main simulator thread can handle them. The ParsePacket() function will take the lighting control data from the received ArtNet packets and reconstruct the data into animation sequence frames. Once reconstructed the frames are placed into an animation sequence similar to the one maintained by the communication component. The main simulator thread will control the output of the animation frames to the hardware simulator based on the delays associated with each frame. At the appropriate intervals, the OutputNext() function will pass the frame's voxel data to the hardware simulator for display and update the display of the hardware simulator.

Hardware Simulation The hardware simulator implements the LED cube as a 3D OpenGL scene of a 10x10x10 equally spaced spheres. The hardware simulator will maintain a 3D array of voxels similar to that of the LED cube class. The scene maintained by the hardware simulator can be altered by setting the entire cube of voxels through the SetFrame() function or by setting the color data for individual voxels with the SetVoxel() function. Voxels can be set either by the firmware simulator to simulate the actual behavior of the system or by the animation component in order to isolate the animations during the testing process. The rendered 3D scene displayed to the user is only updated when the Display() function is called. The hardware simulator implementation will allow for rotation of the scene by dragging the mouse across the screen to change the camera location within the scene.

6.5.2 Software Features

The software features available to the user significantly improve the utility, functionality, and ease of use when creating and displaying animations with the 3D LED cube. While a developer that has been working on the project for an entire semester might have an intimate knowledge of the inner workings of the software. It would be naive to extend the same assumption to any other user. Through the added software features, the same utility available to developers is made available with an added layer of abstraction from the code itself and an increase in simplicity for the user. These features also serve to amplify the power of the underlying code by allowing the user to modify, combine, and create more animations easily within the given framework.

Animation Creator GUI

The animation creator GUI allows a user to create animations through a user friendly GUI that abstracts the programmable animation framework. The animation creator GUI will contain three main animation editing window segments, similarly to Figure 4.9, and a color palette toolbar. The two main window segments to edit the colored voxels within an animation frame will be a display a 3D rendering of the cube, and a second display of a single layer from the cube. The single layer window segment maps to a layer within the cube and is selected through a series of dropdown boxes to specify the primary axis to hold constant and the index of that particular layer. The selected layer will be visualized in the 3D window segment to allow the user to quickly see which layer they are editing. Colors are selected from a color palette similar to that implemented in the program Microsoft Paint, and all voxels in the 2D window segment will obtain the selected color when clicked. The final window segment serves as an editor for the animation sequence. Through this section, additional animation frames can be added, removed, reordered, and edited. The behavior of the animation creator GUI for creating an animation sequence will be quite similar to how Microsoft PowerPoint implements the creation of slideshows. This framework removes the need to code an animation by hand through the programmable animation framework.

Object Rasterization

A beneficial feature for the user would be to aid in the process of creating a 2D or 3D figure. Without this feature, a user that wished to recreate an image on the LED cube would have to manually map out and input appropriate colors for each voxel in the cube. Through the process of rasterization, a vector figure can be reduced to a dot matrix representation that is immediately ready for display on the LED cube. All coordinates within the figure are sampled and mapped to voxel on the LED cube with an appropriate color. The object rasterization feature allows the newly created LED cube figure to be exported for use in the animation creator GUI. With this feature, a user can now input or figure and have it recreated for display on the LED cube with minimal effort.

Animation Controller

Similarly, to how the animation creator GUI allows a user to quickly and easily create animations, the animation controller allows a user to quickly and easily display sets of animations on the LED cube. Through a GUI interface, users are able to create a sequence of animations for playback. The animation controller maintains a list of animations that can be selected to add to the animation sequence or for immediate display on the LED cube. The animation list will include the animation features discussed below and will allow for various variables within the animation to be manipulated to edit the behavior of the animation. The animation controller will create a sequence of animations by appending each animation to each other to create a longer animation that can be saved or passed along seamlessly to other software components. The animation controller will abstract the communication component and perform the communication of selected animations to the physical components of the LED cube. The animation controller will control the distribution of animations to the communication component for playback.

6.5.3 Animation Features

The animation feature set will act as a visual representation of the power and utility of the animation component of the software. A rich animation feature set will be directly correlated with an audience's immediate perception of project success, and for this reason, the feature set must be robust and vast. The following animations are all generated through the Programmable Animation Framework API. The animation features will be available for selection from the Animation Controller with animation specific variables exposed to change the behavior and appearance of the animation. Each specific animation feature may require additional data structures to be maintained to support each individual animation in order to produce the intended features. The features will share common functions as components to achieve a larger and more complex animation. A few animation features will depend on input from other peripheral components such as audio sources and accelerometers.

Mathematical Patterns Seemingly complex patterns and visually appealing animations can be created by defining math functions in 3-dimensional space and using the cube to visualize the result. The functions will be computed on the xy-plane in the grid from $(0, 0)$ to $(9, 9)$ with the z coordinate defining the resultant voxel on the LED cube. All function solutions will be snapped to the nearest unit coordinate for display on a voxel that exists. Any values that exceed the limits of the voxels will be snapped to the largest coordinate the cube is capable of displaying. Mathematical functions can also define color values for each voxel.

A paraboloid centered on the cube that is vertically shifted from far below the cube to far above the cube gives the illusion of the voxels snapping from the bottom layer to the top layer of the cube. The only visible section of voxel transition will be in voxels within the z range visible on the cube, because all voxels below or above the range will snap to the lowest and highest voxels respectively. The negation of the function can achieve the effect of the animation running in the other direction and the voxels returning to their original positions. The use of sinusoidal functions creates periodic animations with interesting patterns that can appear. The generation of sinusoidal functions with variable amplitude, frequency, and phase across dependent on both x and y and possibly time can create a vast set of animations. Additional animations can be created by expanding or contracting a sphere or other volumes from various coordinates within the LED cube. The numbers of possible animations that utilize combinations of mathematical functions are limited only by the unit display resolution of the LED cube.

Raining Voxels A raining effect can be achieved from the LED cube by having fully lit voxels appear to drop from the top to the bottom of the cube. This effect is implemented rather simply by pushing new layers onto the top of the cube, which results in the entire cube shifting down at each iteration. The pushing down of layers creates the effect of gravity, with new droplets beginning only when a new layer is inserted. With this method of inserting and shifting droplet layers down, the animation only needs to compute new droplet locations and does not need to keep track of rain droplets within the animation. In addition to this, creating a fading trail behind each falling droplet creates a further illusion of motion. This is implemented simply by decreasing the color intensity of the previous layer by a constant factor before adding in the new droplets. With an appropriate fade rate, the improved raining voxel animation can mimic the famous falling characters from the movie The Matrix.

Fireworks The illusion of fireworks can be created by combining some functions of previously described animation features. A firework animation consists of a firework shot rising up a column of the LED cube, exploding to create a hollow sphere of particles centered at the peak of the rising shot, and finally all the exploded particles falling down the cube. The rising shot portion of the animation is implemented similarly to the raining voxel animation by shifting layers into the bottom of the cube until the shot has reached its peak. The firework explosion is created by generating an expanding hollow sphere of voxels centered at the peak of the shot. After the explosion has completed, gravity pulling the exploded particles to the ground is implemented similarly to how falling raindrops are implemented. The firework animations will feature multiple explosion colors, and the falling exploded particles will turn to resemble an orange ember that fades at it falls to the ground.

Scrolling Figures The LED cube can display scrolling figures similar to a scrolling LED banner in multiple manners that complement as well as extend the common features of the traditional 2D LED banner. The traditional 2D scrolling animation from one face to another is implemented in 3D to supports both 2D figures across a single layer along the axis of motion or 3D figures spread across layer perpendicular to the axis of motion. The scrolling figure is represented internally as an array of the same dimensions as the LED cube with the one exception that the axis of motion is extended to support the length of the scrolling animation. A moving window across the scrolling figure data structure will determine the current display on the LED cube. As new layers are pushed into the LED cube, the old layers will shift out, achieving the scrolling effect across a single axis of motion. By treating the LED cube as a hollow cube and only considering the outermost voxels, the effect of a 2D banner scrolling across a 3D volume is achieved. Again, internally the 2D scrolling animation will be represented by a 2D array of voxels the height of the cube and the width of the scrolling figure. The number of faces chosen as display surfaces determines how many layers must be sampled from the scrolling figure for display. With an example of only utilizing three faces and neglecting the rear face, a left-to-right scrolling animation would begin on the leftmost face, travel across the front face, and disappear between the rightmost and rear faces. Three moving windows that overlap by one voxel column at their borders will be shifted along the 2D figure and the entire layer will be set on each appropriate cube face for display. The implemented scrolling figures features recreate but also extend the capabilities of similar 2D LED displays, highlighting the capabilities of the 3D LED cube.

Rotating Figures A solid figure can be rotated within the LED cube to provide an additional feature as well as to possibly add support for other features. With a figure placed in the center of the cube, the figure must be limited to a cylinder of radius 5 for rotation about a single primary axis or a sphere of radius 5 for rotations about all axes in order to prevent clipping of the animation at the edges. A series of defined rotations to the base animation results in rotated set of voxels in the coordinate space for display. After rotation, not all voxels from the base animation will line up perfectly to a voxel for display. The voxels of the rotated figure will be snapped to the voxel represented by the nearest unit coordinate. An internal representation of the base figure will never be modified to prevent figure degradation. Rather, the rotating figure animation will maintain a vector that represents the current orientation of the base figure. Incremental changes of rotation to the figure are made directly to the orientation vector and the displayed figure is recomputed from this orientation vector at every iteration to prevent the unintended walking or degradation of voxels from the original figure.

Gravity Simulation As an extension of the rotating figures rotating figures animation, an interesting feature can be added by adding an accelerometer on the LED cube to provide orientation data. With the orientation of the cube known at all times, it is possible to orient the displayed figure on the cube such that it appear to be affected by gravity and always rests at the bottom of the cube with respect to the ground. This gravity effect is created by applying the accelerometer output as the input to the rotating figure animation. The input from the accelerometer must be sampled frequently and the update figure must be displayed at a high refresh rate in order to achieve a realistic effect. A novel example of this animation feature is to consider the LED being half-full of water and bounded by the edges of the cube. A layer of voxels rests in the middle of the cube representing the water surface and is rotated based on the orientation of the cube to remain level with respect to the ground. All voxels below the water surface will be filled as water. An additional effect would be to allow for an open upper face, and to allow the water level to decrease as water poured from the top of the cube. For this feature to be implemented, it requires the addition of an extra physical component to the LED cube and a channel for communication back to the animation software.

Audio Visualization A neat and interactive animation feature to include is an audio visualizer. By taking an audio sample as input and applying a Fast Fourier Transform to the waveform, the waveform will decompose to a spectrum of frequencies and their associated amplitudes. A visualization of the audio signal is created by plotting the spectrum on the xz-plane of the cube with the z axis representing amplitude and displayed with respect to a shifting time window along the y-axis. The colors of voxels will be determined by both frequency and amplitude to provide added visual separation across the audio signal spectrum. This animation gives a viewer the ability to visualize the audio signal they are hearing and identify interesting features of the audio signal that appear in the spectrum across time. The audio signals can be input in both real-time through a microphone peripheral or from a saved audio file input to the animation software.

Chapter 7

Project Prototype Testing

An important aspect of the design process is testing the prototype to establish that it meets all design criteria and functionality in the required environments. The completion of the prototype testing stage will confirm the specifications of the LED cube as well as its consistent functional operation. Further, the tests will be conducted both modularity and chronologically, meaning that each specific stage of the design will be tested individually throughout the build process. This will allow cascading errors (an error early in the build process causing multiple errors later on) to be entirely avoided as well as provide the identification of any errors to a specific hardware or software functionality concern. For example, rather than stating "one row of LEDs isn't turning on." This test procedure will allow us to specifically state the cause of the problem; which could be "our power supply isn't operating correctly", or "we soldered a row incorrectly", or "an error was made in the software development." Clearly, this type of test procedure is vital to the completion of any project - especially one with a number of intricate components. Rarely are tasks completed perfectly the first time - we do not expect this design to be without error. By utilizing a proper test protocol, we expect to discover our errors and fix them in the most efficient method possible.

7.1 Hardware Test Environment

All tests will be conducted within the humidity and temperature operating ranges of the 3D LED cube, and in a well-lit area to test the brightness of the LEDs. The majority of tests will be conducted indoors, with a temperature near 75°F and a humidity near 50%. The lighting of the room will be equivalent to a standard indoor setting.

7.2 Hardware Tests

The three primary hardware components to be tested will be the LEDs, LED drivers, and control board. Each of these components will be tested separately to quarantine any errors and aid in the finding of solutions via process of elimination.

LEDs The LEDs will be tested in a simple manner, confirming their functionality. Each LED test will involve an "on/off" check with each color: red, green, and blue. Each LED will be tested individually before being soldered to the cube, then also individually after soldering to the cube. This will ensure that both the LED is in operating condition, as well as identify any problems in the wiring of the LED cube.

As the cube will be soldered one plane at a time, each plane will be tested before its addition to the cube structure. Finally, upon the addition of each plane to the cube, the entire cube existing at that point in time will be tested.

LED Drivers The LED driver board is a simple component of the LED cube, controlling only current, brightness, and color. Each of these factors will be tested individually from each channel. The channels will be calibrated so that the same color/brightness appears for all LED's across each LED driver channel and device.

Control Board The control board, a complicated design component, will undergo rigorous testing. At the simplest level, the firmware will be updated and read/write access to the on-board SD card will be tested. The communication between the laptop and control board CPU will be tested by hosting a basic website on the control board, a simple procedure in an Ethernet based communication protocol. The communication between the control board and the driver board will be tested by blinking LEDs on and off.

7.3 Software Test Environment

The software simulator of both the firmware and hardware will enable testing of much of the software components. The firmware simulator will serve to test the communication component of the software. While, the hardware simulator will serve to test the animation component of the software.

The firmware simulator will measure transmission rate from the communication node. Additionally, it will test that all animation data being transmitted to the firmware can be properly reconstructed given the information received by the firmware simulator. The firmware simulator will mirror the physical firmware design. As such, the firmware simulator will detect cases when the firmware's capabilities are exceeded, including but not limited to processing speed, transmission rates, and buffer overflows.

The hardware simulator will provide a visual representation of the LED cube. In addition to visualizing the LEDs, the hardware simulator will measure the current, voltage, and power requirements of each individual frame and full animation. This allows developers to view animations, as well as measure the physical demands each animation will take on the LED cube.

7.4 Software Tests

Each piece of software that enters the project must be thoroughly tested before integration to the final release project. All features and animations should be tested first on the simulator test bed then tested with the physical LED cube itself.

7.4.1 Animation Tests

Any animations that are developed will pass through the software test bed prior to verify the animation renders as intended and that it does not exceed the power requirements of the LED cube. If the animation exceeds any physical limitations of the LED cube, the animation must be redesigned. Likewise, if the animation does not render as intended, the animation should be corrected and retested.

Animation Creator GUI The functionality of the Animation Creator GUI must be tested. The tester must verify that all voxels from the Animation Creation GUI map exactly to the corresponding LED on the LED cube. The tester must verify that the Animation Creator GUI is able to produce the appropriate colors when displayed on the LED cube. This will include testing all the gradients of each RGB channel.

Coded Animations The coded animations must be tested in a similar manner. The tester must verify that all voxels indexed within the code map exactly to the corresponding LED on the LED cube. The tester must verify that the code is able to produce the appropriate colors when displayed on the LED cube. This will include testing all the gradients of each RGB channel.

7.4.2 Communication Tests

The communication tests will consist of sending a verified animation into the communication component and verifying the output after the data is transmitted to the firmware simulator. The communication tests will also verify that the communication methods are able to produce throughput that conforms to the function requirements.

Full Animation The full animation transmission mode will be verified by inputting a full animation to the communication component. Once transmission of the animation from the communication component to the firmware simulator is completed, the animation will be verified both by playback of the animation through the hardware simulator and a memory comparison. The entire animation must be transmitted perfectly. The throughput of the full animation transmission method must conform to the functional requirements.

Frame by Frame The frame-by-frame transmission mode will be verified by repeatedly inputting a single frame to the communication component. Once transmission of the frame from the communication component to the firmware simulator is completed, the frame will be verified both by display of the frame through the hardware simulator and a memory comparison. The entire frame must be transmitted perfectly. The throughput of the frame-by-frame transmission method must conform to the functional requirements.

Shift Layer In The shift layer in transmission mode will be verified by repeatedly inputting a single layer from every side of the cube to the communication component. Once transmission of the layer and side from the communication component to the firmware simulator is completed, the updated frame will be verified both by display of the frame through the hardware simulator and a memory comparison. The entire layer and side information must be transmitted perfectly. The throughput of the of the shift layer in transmission method must conform to the functional requirements.

Translate / Rotate Frame The translate / rotate frame transmission mode will be verified by repeatedly sending translation and rotation vectors to the communication component. Once transmission of the translation and rotation vector from the communication component to the firmware simulator is completed, the updated frame will be verified both by display of the frame through the hardware simulator and a memory comparison. The entire translation and rotation vector must be transmitted perfectly and the desired effect must be produced in the displayed frame. The throughput of the translate / rotate frame transmission method must conform to the functional requirements.

7.5 Final Integrated Tests

Upon completion of each individual test, a final integrated test will be performed. The purpose of this inclusive, rigorous test is to ensure that no additional errors arise from the integration of each separate design component. By putting the final design through a comprehensive test, the integrity of the system will be confirmed. This intensive test will consist of 2 hours of run-time. The software will loop different types of animations over this time period and each design component will be carefully supervised to confirm proper operating condition. The software will be observed, with any performance-detracting bugs noted. Each hardware component will be periodically measured: the voltage/current output of the power supply, the current output of the LED drivers and the current output of the MOSFETs. The temperature of each component will be measured after a significant amount of run-time to confirm all hardware parts remain within an appropriate temperature range for operation. Visually, the LED cube will be observed to confirm that the brightness and color across the LEDs as a whole is consistent with the software instructions. Each animation will be shown sequentially on the cube to verify that the animations appear as designed, with the correct timing, color, brightness and with minimal signal delay. Finally, several separate viewers will observe the LED cube to confirm the refresh rate exceeds the threshold necessary for the perception of continuous operation, as this factor can be relative to the viewer. Any errors will be noted and iteratively fixed. The successful completion of this extended operation test will conclude the design and testing of the 3D LED cube.

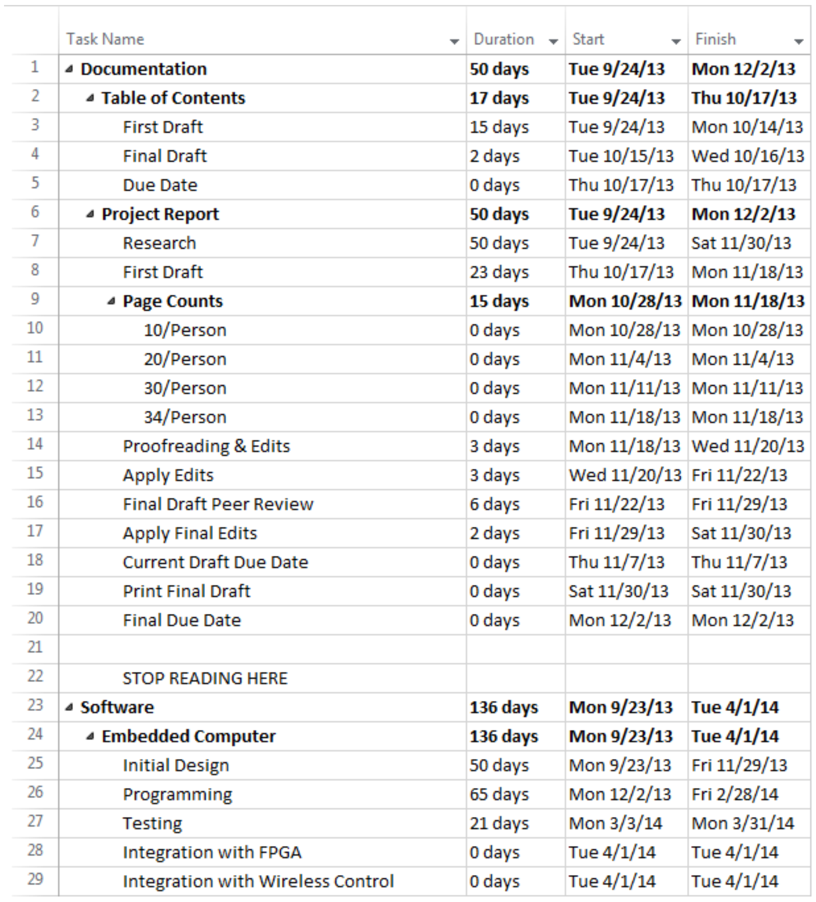
Chapter 8

Administrative Content

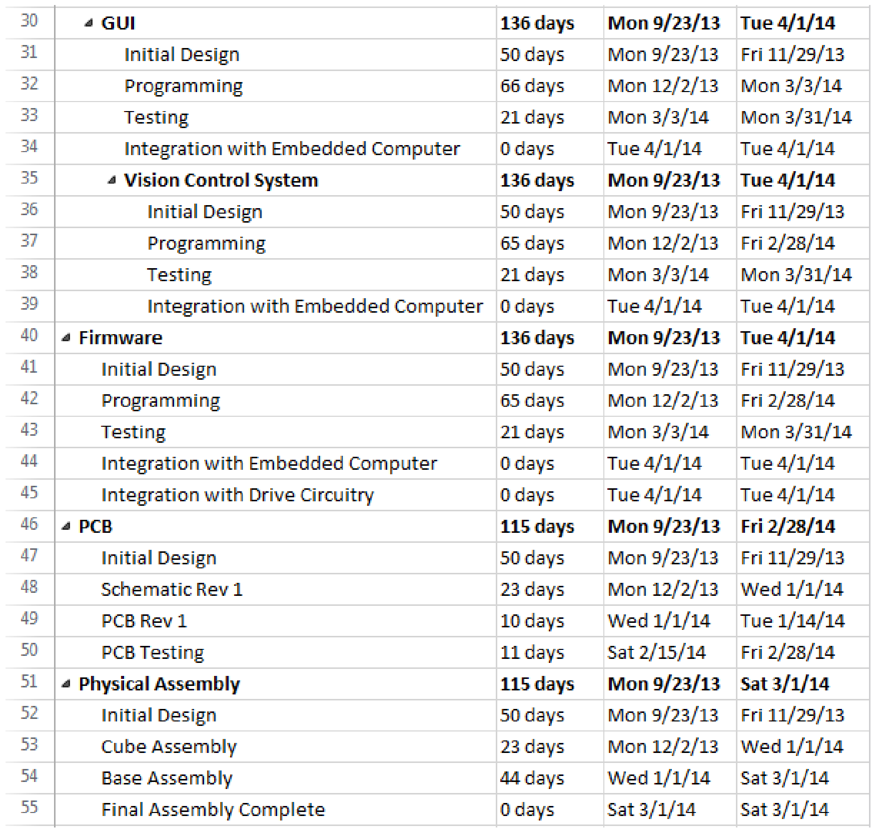
As with any project, it is essential to create and ensure that a schedule and budget are followed carefully. It is also important to get to know the people working on the project, and their goals and aspirations. The following chapter will cover the administrative material of the project, and will provide a brief overview into who the group is made of, and what we enjoy doing.

8.1 Milestones

A challenge of the Senior Design documentation and design process was creating and following a schedule. Adopting a milestone chart, with specific time requirements for the completion each component of the research and designed allowed for a systematic completion guideline. Figure 8.1 describes the time allotted as well as the deadline for each stage of development.



	Task Name	Duration	Start	Finish
1	**Documentation**	**50 days**	**Tue 9/24/13**	**Mon 12/2/13**
2	**Table of Contents**	**17 days**	**Tue 9/24/13**	**Thu 10/17/13**
3	First Draft	15 days	Tue 9/24/13	Mon 10/14/13
4	Final Draft	2 days	Tue 10/15/13	Wed 10/16/13
5	Due Date	0 days	Thu 10/17/13	Thu 10/17/13
6	**Project Report**	**50 days**	**Tue 9/24/13**	**Mon 12/2/13**
7	Research	50 days	Tue 9/24/13	Sat 11/30/13
8	First Draft	23 days	Thu 10/17/13	Mon 11/18/13
9	**Page Counts**	**15 days**	**Mon 10/28/13**	**Mon 11/18/13**
10	10/Person	0 days	Mon 10/28/13	Mon 10/28/13
11	20/Person	0 days	Mon 11/4/13	Mon 11/4/13
12	30/Person	0 days	Mon 11/11/13	Mon 11/11/13
13	34/Person	0 days	Mon 11/18/13	Mon 11/18/13
14	Proofreading & Edits	3 days	Mon 11/18/13	Wed 11/20/13
15	Apply Edits	3 days	Wed 11/20/13	Fri 11/22/13
16	Final Draft Peer Review	6 days	Fri 11/22/13	Fri 11/29/13
17	Apply Final Edits	2 days	Fri 11/29/13	Sat 11/30/13
18	Current Draft Due Date	0 days	Thu 11/7/13	Thu 11/7/13
19	Print Final Draft	0 days	Sat 11/30/13	Sat 11/30/13
20	Final Due Date	0 days	Mon 12/2/13	Mon 12/2/13
21				
22	STOP READING HERE			
23	**Software**	**136 days**	**Mon 9/23/13**	**Tue 4/1/14**
24	**Embedded Computer**	**136 days**	**Mon 9/23/13**	**Tue 4/1/14**
25	Initial Design	50 days	Mon 9/23/13	Fri 11/29/13
26	Programming	65 days	Mon 12/2/13	Fri 2/28/14
27	Testing	21 days	Mon 3/3/14	Mon 3/31/14
28	Integration with FPGA	0 days	Tue 4/1/14	Tue 4/1/14
29	Integration with Wireless Control	0 days	Tue 4/1/14	Tue 4/1/14



30	**◢ GUI**	**136 days**	**Mon 9/23/13**	**Tue 4/1/14**
31	Initial Design	50 days	Mon 9/23/13	Fri 11/29/13
32	Programming	66 days	Mon 12/2/13	Mon 3/3/14
33	Testing	21 days	Mon 3/3/14	Mon 3/31/14
34	Integration with Embedded Computer	0 days	Tue 4/1/14	Tue 4/1/14
35	**◢ Vision Control System**	**136 days**	**Mon 9/23/13**	**Tue 4/1/14**
36	Initial Design	50 days	Mon 9/23/13	Fri 11/29/13
37	Programming	65 days	Mon 12/2/13	Fri 2/28/14
38	Testing	21 days	Mon 3/3/14	Mon 3/31/14
39	Integration with Embedded Computer	0 days	Tue 4/1/14	Tue 4/1/14
40	**◢ Firmware**	**136 days**	**Mon 9/23/13**	**Tue 4/1/14**
41	Initial Design	50 days	Mon 9/23/13	Fri 11/29/13
42	Programming	65 days	Mon 12/2/13	Fri 2/28/14
43	Testing	21 days	Mon 3/3/14	Mon 3/31/14
44	Integration with Embedded Computer	0 days	Tue 4/1/14	Tue 4/1/14
45	Integration with Drive Circuitry	0 days	Tue 4/1/14	Tue 4/1/14
46	**◢ PCB**	**115 days**	**Mon 9/23/13**	**Fri 2/28/14**
47	Initial Design	50 days	Mon 9/23/13	Fri 11/29/13
48	Schematic Rev 1	23 days	Mon 12/2/13	Wed 1/1/14
49	PCB Rev 1	10 days	Wed 1/1/14	Tue 1/14/14
50	PCB Testing	11 days	Sat 2/15/14	Fri 2/28/14
51	**◢ Physical Assembly**	**115 days**	**Mon 9/23/13**	**Sat 3/1/14**
52	Initial Design	50 days	Mon 9/23/13	Fri 11/29/13
53	Cube Assembly	23 days	Mon 12/2/13	Wed 1/1/14
54	Base Assembly	44 days	Wed 1/1/14	Sat 3/1/14
55	Final Assembly Complete	0 days	Sat 3/1/14	Sat 3/1/14

Figure 8.1: Project Milestones

8.2 Workload Distribution

The workload for the completion of this senior design project was divided as evenly as possible amongst the group members. Each member was delegated responsibility for a major component of the project. All control hardware and circuitry was given to Josh, all circuitry for LED operation was given to Luke, and Andrew took responsibility of all software design. This responsibility separation is illustrated in Figure 8.2, which shows the block diagram of the 3D LED cube, with the responsibilities of each team member distinguished. Although responsibility was separated, each group member played a key role in every aspect of the design. Research vectors were decided upon as a team, and parts were also researched together. Josh, the group member funding the project, made final purchasing decisions. Each group member claimed specific sections in this documentation paper at his own discretion, however, past the initial rough draft stage, each group member contributed to the content and editing of many separate sections. This paper and this project truly is a fully collaborative effort, and while each group member emerged with expertise in their specific areas, a high level of competency was formed and maintained across all design structures by each group member.

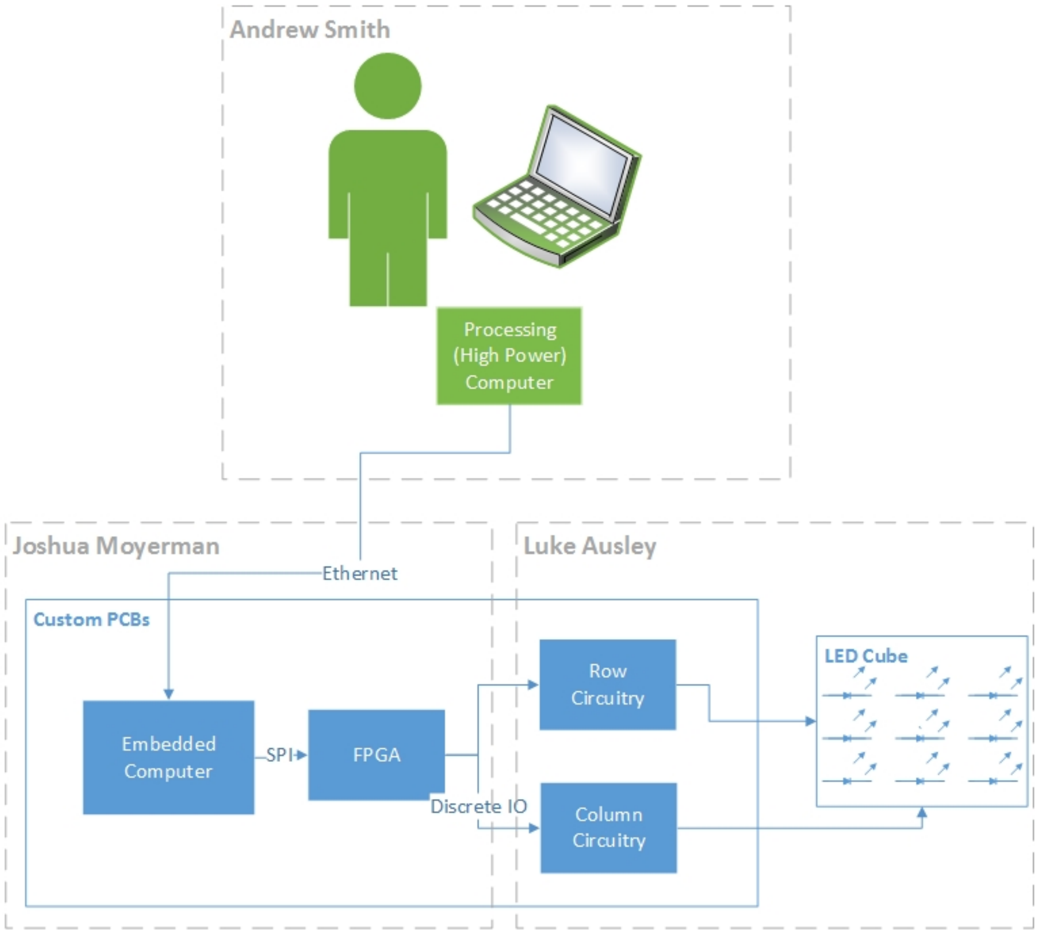


Figure 8.2: Workload Distribution

8.3 Budget and Finances

The 3D LED display is approximated to cost $900. A sponsorship by Stellascapes will cover the cost of LEDs and PCB, which will reduce the out of pocket cost to the group. Stellascapes has a great interest in the operation of the embedded processor and FPGA working in conjunction for lighting control. In exchange for a]the sponsorship, the group will share all designs valuable information learned in the design/prototype process. Group member Joshua Moyerman will provide the balance of the project expenditures, up to an amount of an additional $1,000, due to interest in owning the project after its completion. The group members will evenly distribute any additional costs necessary for the completion of the project amongst themselves. Table 8.1 outlines the group budget by item.

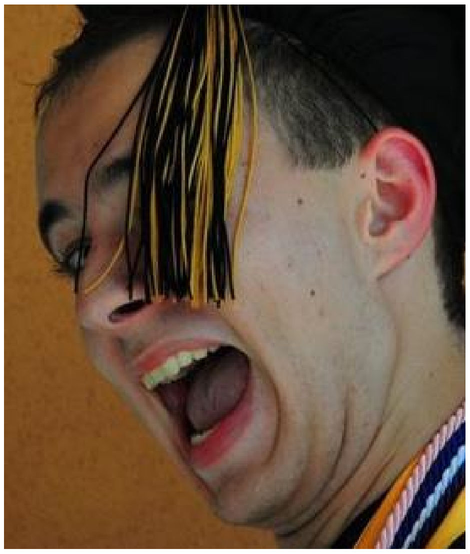
Item	Cost
LEDs	$350
Wire	$150
Embedded Computer	$10
PCBs	$100
LED Drivers	$60
MOSFETs	$10
Analog to Digital Converter	$40
Passive Components	$5
Power Supply	$70
Processing Computer	Group Owned
Frame & Case	$115
Subtotal	$900
Sponsored	($450)
Sampled	($100)
Total	$350

Table 8.1: Project Budget

8.4 Group Member Information

Luke Ausley was born April 2, 1993 in Pensacola, FL and raised in the panhandle of Florida. Attending the Collegiate High School at Northwest Florida State College, Luke graduated in May 2011 with his high school diploma and Associate of Arts degree. Luke will graduate with his Bachelor of Science in Electrical Engineering (BSEE) in May 2014 to pursue a career in his field of interest: Optics. In his free time, Luke enjoys several hobbies with notable interests related to cars, landscape/nature photography and high-end audio equipment. Enjoying traveling, one his life goals is to set foot on each continent. A recipient of the Department of Defense SMART scholarship, Luke has worked full-time as an engineering intern for the past 4 summers with the Air Force Research Lab's (AFRL) Munitions Directorate at Eglin AFB, FL. He has accepted post-graduation employment with the AFRL as an electronics engineer.

Joshua Moyerman was born March 14, 1992 in Philadelphia, PA, and raised in both Pennsylvania, and Florida. Having attended East Ridge High School, and Lake Sumter Community College as a high school student, Josh graduated in May 2010 with his high school diploma, and a year of college already completed. Josh anticipates graduating with his Bachelor of Science in Computer Engineering (BSPE) in May 2014 to pursue a career in the field of embedded development. Josh has volunteered and worked throughout his college career to expand his engineering experience. Joshua is currently working for Stellascapes, a company largely responsible for the sponsorship of this project, and hopes to continue working there after he graduates. He hopes to also spend more enjoying his hobbies of photography, music, and reading.



Andrew Smith was born December 11, 1990 in Boynton Beach, FL. Andrew lived briefly at Maxwell AFB in Montgomery, AL before relocating to MacDill AFB in Tampa, FL. He attended H. B. Plant High School where he was a State Champion football player and graduated in May 2009. Andrew joined Air Force ROTC his junior year of college and will commission into the United States Air Force (USAF) as a 2Lt when he completes his Bachelor of Science in Computer Engineering (BSPE) in May 2014. Andrew interned with the AFRL Information Directorate's Information Assurance Internship in the summer of 2013. He currently works as an intern in the College Work Experience Program with Lockheed Martin developing and supporting engineering tools for the Arrowhead fire control system. After graduation, Andrew will begin work with the USAF as a Cyberspace Operations Officer pursuing his passion for cyber security and defending the nation's network assets.

Chapter 9

Conclusion

The completion of this project documentation culminated in a clear and precise direction moving toward the completion of a final prototype of the 3D LED cube. Throughout the research phase of development, each group member built a high level of proficiency in each of their respective areas of concentration, while maintaining competency across all the areas of the project design. The design phase was where the individual skills of the group members were put to the test - creating a highly detailed and formulated design plan to achieve the required specifications for the completed prototype. Each stage of this design process was heavily documented and outlined in this paper, adding accountability to each major decision made - as each step had to have specifications and facts supporting that particular directive.

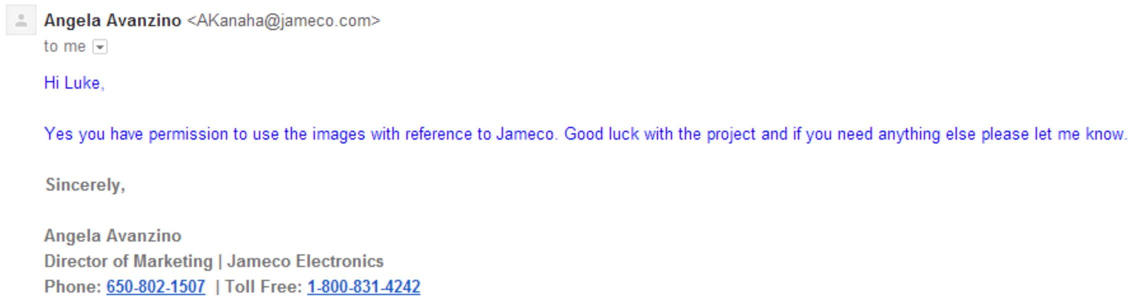
The research stage of the senior design project led to a number of important decisions - choosing parts as well as noticing and planning ahead for potential design concerns. When the design was completed and documented, we had a clear and functional outline: a software interface to send instructions to an on-board control PCB, containing an embedded processor and FPGA operating in unison to control two separate driver boards containing LED drivers and MOSFETs which in turn modulated the current to each individual LED with a high level of speed and precision. A detailed testing procedure confirms the operation of the 3D LED cube - leaving the group with a clear and finalized path moving forward.

The most substantial result of this documentation was the learning experience of the individual group members. This senior design project took it's members well beyond their formal classroom electrical and computer engineering experience. Although the skill development was broad-based, including time management, team building, communication, and technical writing, the primary growth came from achieving the goal of this project: establishing a functional 3D LED cube prototype design. This success accompanied the new skills and knowledge obtained such as embedded processor development, discrete component integration, printed circuit board construction, communication protocols, embedded software development, and software design. Clearly this project spanned several topics covering the topics of both electrical and computer engineering. The group members look forward past the successful completion of this stage of development, to the completion of a physical functional prototype to be built in the secondary phase of this senior design project.

Appendix A

Copyright Permissions

[1]



Angela Avanzino <AKanaha@jameco.com>
to me

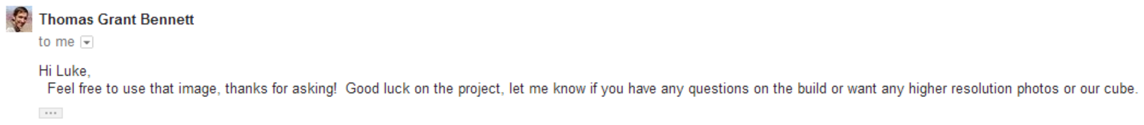
Hi Luke,

Yes you have permission to use the images with reference to Jameco. Good luck with the project and if you need anything else please let me know.

Sincerely,

Angela Avanzino
Director of Marketing | Jameco Electronics
Phone: 650-802-1507 | Toll Free: 1-800-831-4242

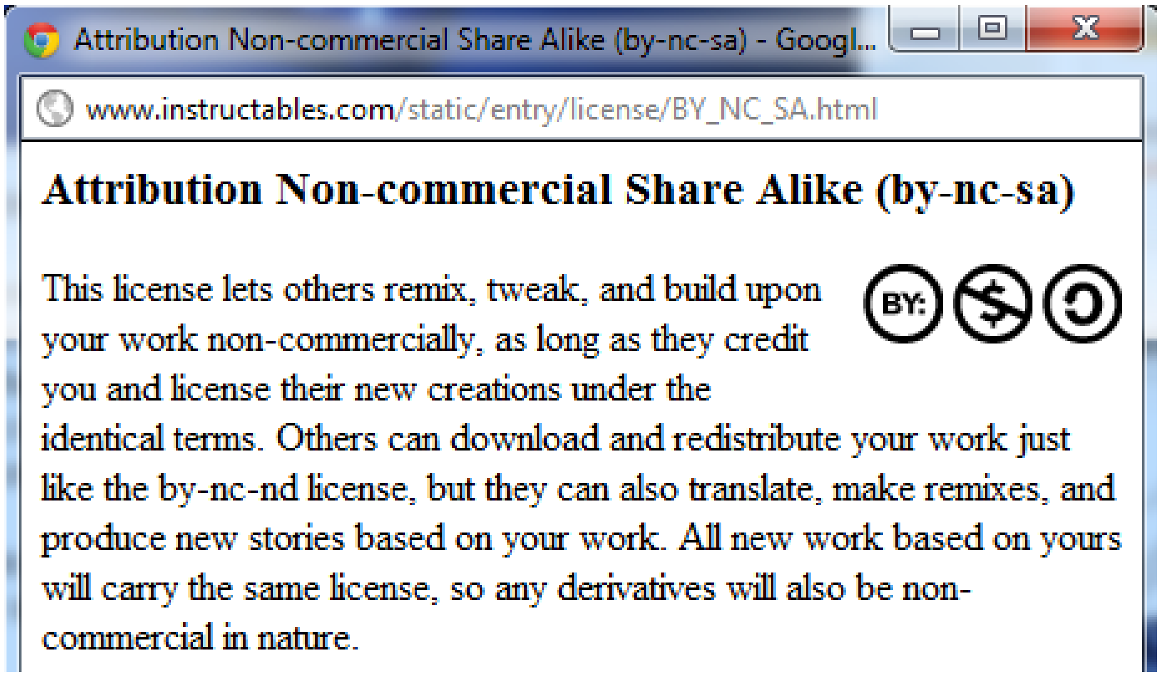
[2]



Thomas Grant Bennett
to me

Hi Luke,
Feel free to use that image, thanks for asking! Good luck on the project, let me know if you have any questions on the build or want any higher resolution photos or our cube.

[3]



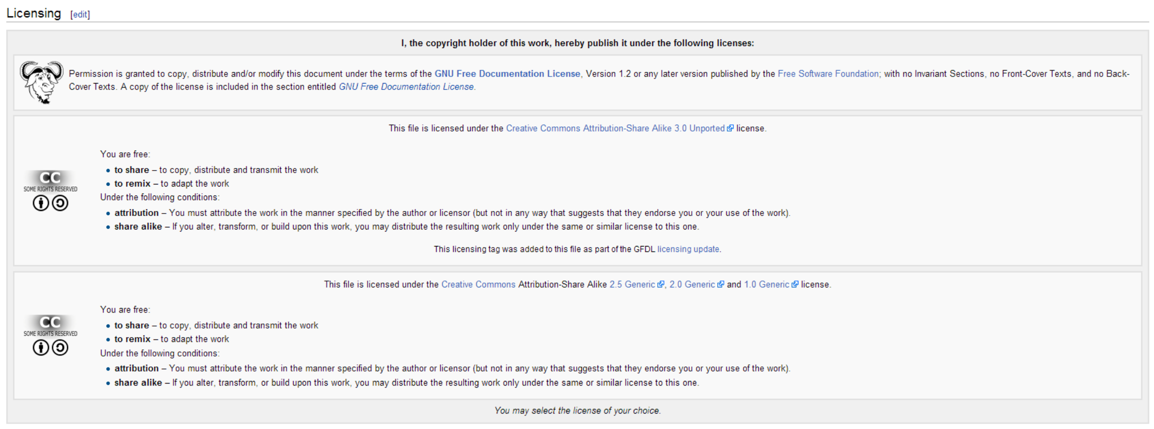
Attribution Non-commercial Share Alike (by-nc-sa) - Googl...

www.instructables.com/static/entry/license/BY_NC_SA.html

Attribution Non-commercial Share Alike (by-nc-sa)

This license lets others remix, tweak, and build upon your work non-commercially, as long as they credit you and license their new creations under the identical terms. Others can download and redistribute your work just like the by-nc-nd license, but they can also translate, make remixes, and produce new stories based on your work. All new work based on yours will carry the same license, so any derivatives will also be non-commercial in nature.

[4] Permission pending from HowNotToEngineer.com
[5]


Licensing [edit]

I, the copyright holder of this work, hereby publish it under the following licenses:

Permission is granted to copy, distribute and/or modify this document under the terms of the **GNU Free Documentation License**, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled *GNU Free Documentation License*.

This file is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.

You are free:
- **to share** – to copy, distribute and transmit the work
- **to remix** – to adapt the work

Under the following conditions:
- **attribution** – You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **share alike** – If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

This licensing tag was added to this file as part of the GFDL licensing update.

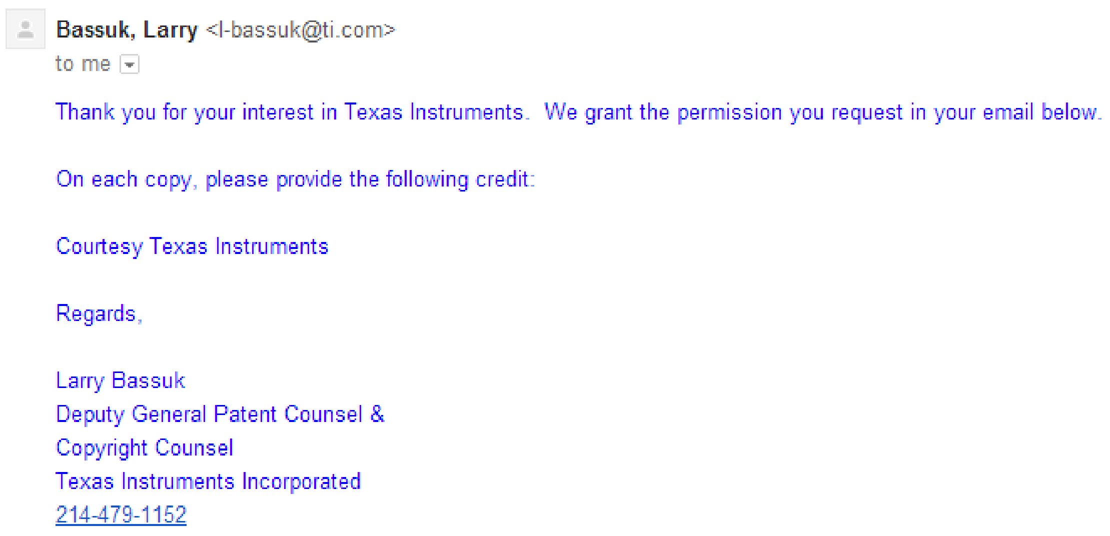
This file is licensed under the Creative Commons Attribution-Share Alike 2.5 Generic, 2.0 Generic and 1.0 Generic license.

You are free:
- **to share** – to copy, distribute and transmit the work
- **to remix** – to adapt the work

Under the following conditions:
- **attribution** – You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **share alike** – If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

You may select the license of your choice.

[6]


Bassuk, Larry <l-bassuk@ti.com>
to me

Thank you for your interest in Texas Instruments. We grant the permission you request in your email below.

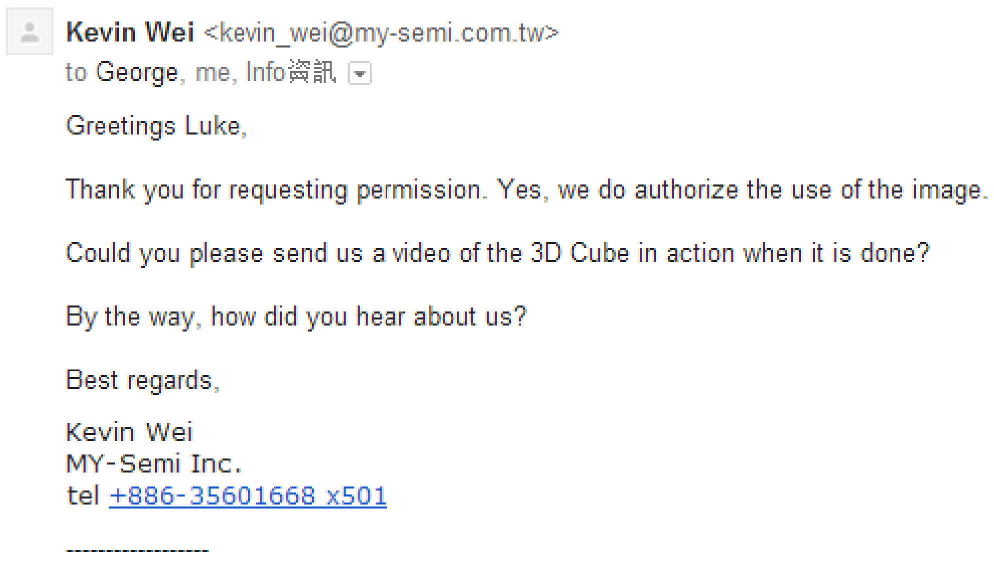
On each copy, please provide the following credit:

Courtesy Texas Instruments

Regards,

Larry Bassuk
Deputy General Patent Counsel &
Copyright Counsel
Texas Instruments Incorporated
214-479-1152

[7]



Kevin Wei <kevin_wei@my-semi.com.tw>
to George, me, Info資訊

Greetings Luke,

Thank you for requesting permission. Yes, we do authorize the use of the image.

Could you please send us a video of the 3D Cube in action when it is done?

By the way, how did you hear about us?

Best regards,

Kevin Wei
MY-Semi Inc.
tel +886-35601668 x501
