Senior Design II Document
April 28, 2014

GameQube: An Interactive LED Cube

Group 33
Omar Alami
Stephen Monn
Matthew Dworkin

University of Central Florida
Dr. Samuel Richie
Senior Design I

Table of Contents

1.0 Executive Summary

The GameQube consists of an LED volume display cube that a person can play simple games on using a GameCube controller. Past LED cube project creations have mainly focused on showing pre-defined animations. This project expands on those ideas so that a user will be able to control what is shown on the cube in the same way that a video game player has control over his character. Other than basic animations, playable games such as Pong and Snake have been implemented.

This project was chosen by a group of three computer engineers at UCF to be a good mix of hardware and software in which the whole group could experience a full engineering development project. Many project ideas were considered, all were somehow linked around some sort of interactive game. LED cube projects are common and have been done by UCF groups in the past, this project looks to stand out by becoming an interactive game system which a user has direct control over. Not only does an LED Cube require significant hardware design but the project also goes beyond traditional LED Cube projects to contain a large software design portion.

The main goal of the project was to allow a player to play a classic game in a brand new way by playing it in a 3D environment on an LED cube. In order to accomplish this, the cube needed to be large enough to house the number of LEDs necessary to replicate the intended game but small enough so that it was portable. Each LED in the display is able to be one of 256 different color combinations. With respect to the system required to make the cube function, it includes a microcontroller, LED control software, and an input device that acts as a controller for the game's player. Several different control possibilities were considered including the utilization of a small infrared camera of a Wii Remote to track the positioning of the player's controller so he/she could point to a specific location in a three dimensional array of LEDs. The final decision for input was a traditional GameCube controller allowing for both wireless and wired options. Finally, the system is powered by a standard AC wall outlet and the cube is connected to a control device utilizing LEDs with a common cathode. The LED cube system is portable and easy to use for anyone who might want to use it regardless of knowing the internal workings. The final design is encapsulated in an acrylic casing to protect the cube which lays on top of a wood containment holding the electronic components. All that is needed to play is an outlet to plug into and the controllers.

This project required knowledge of both hardware and software design. In order to complete the physical LED cube display, the physical structure of the LEDs, the logical control over them, and the interface to the display were designed from the ground up. The software includes unique games designed to work in a 3D space as well as low level software interfaces for input purposes and a small menu system to choose between the different applications and games. The software design also involves a custom rendering pipeline, as well as the software interface to output to the custom LED display. Other considerations for this project included integration of LCD screens rather than LEDs. Using a physical cube with at least three LCDs and head tracking software could emulate 3D objects using perspective and allow interaction and games to be made. However, the main constraint for this idea was expense of the LCD screens and the need of an external computer to provide the rendering capabilities.

To reach the goals of this project, the team has completed research on all the necessary components. Being a three member group, utilization of time and resources was a major emphasis. Depending on each member's school and work schedules, some parts progressed faster than others. To ensure completion of all aspects of the project, modularity was a very large part of our design. This goal included having the hardware, software, and control aspects of the design to be as independent as possible before integration is required. After gathering the basic requirements of each phase's integration with the others, the development progressed unimpeded for each individual phase. This allowed the project to be divided into three development phases with an admin in charge of one. To reach such a goal, the research and communication of each phase was very thorough, as developing one phase without proper research on its integration might have proved detrimental.

The contents of this document in detail all the research of all aspects of the project that lead into the project's design that allowed the team to successfully construct and develop the LED cube.

2.0 Project Description

2.1 Project Motivation

Consisting of three computer engineers, the team looked at many options that utilized a good balance of hardware and software design. Rather than trying to find an idea that would allow sponsorship, the main motivation was a project that the team would all enjoy doing. Research for project ideas included looking at past UCF senior design projects, looking at other universities' senior design projects, and other popular projects online. After discovering the LED cube project on instructables, the project's interest was drawn. From there the team evaluated the pros and cons of the project, alternative ideas, and ways to separate this project from other similar projects.

Many other ideas which involved a cube were considered, such as the LCD cube idea discussed earlier. The LCD cube would allow for full graphical rendering using LCD screens. Rather than being physical 3D screen like the LED Cube, the LCD cube would create the 3D illusion using multiple screens. The minimum need would be three screens, each would display the perspective of the same scene creating a 3D illusion. To create this, headtracking would be needed. This project would allow for many sorts of games and effects to display based on movement. While this project was very interesting the constraints proved too great. At least three five inch LCD displays would be needed, which are expensive, along with LCD drivers. To render the scene the project would require a full GPU inside a PC, which would not be a convenient project to demonstrate. The LED cube proved to be the best choice. Not only did it seem to have significant hardware, consisting of 1000 LEDs that need to be wired and configured in a custom designed electrical circuit, but also required high level programming to provide interactive features. The software and input side also allowed the project to become extendable. Depending on time constraints, more animations, games, and software features can be continually added to the project. The input, which at the minimum will be GameCube controllers, could also be expanded to a more interactive IR movement controller and/or motion controls using accelerometers and gyroscopes.

Although the team decided a project like this would be enjoyable and motivating to work on, other considerations such as financing had to be considered. Without any significant energy or AI features, an LED cube project would most likely need to be self financed. When considering all project ideas, the budget played a major factor. The goal was to find a project that could be financed with each member paying around $100. Based on rough estimates, the LED cube seemed to fit into this budget. Given all considerations the team was willing to spend some more if the project exceeded this budget. In the end, the LED cube was determined to be the best fit for the team in terms of scope, budget, and preference.

2.2 Goals and Objectives

The main goal of the project was to have a fully portable and encased 10x10x10 LED cube display that can be plugged into a wall and utilize a controller to turn on several animations and/or effects along with playing interactive games. The hardware goals included having the 10x10x10 LED array standing straight and stable. Not only did we want the the LEDs to be aligned and straight, but the connecting wires should be as discreet as possible. The space between each LED was also very important, as it plays a main factor into viewing depth and LED ghosting. This goal also included having a good design in which a minimum amount of wires were used in order to reduce space. Another hardware goal was to have as few circuit boards as possible, ideally one integrated circuit that could fit inside the casing and includes the LED cube circuitry, the microcontrollers, and any additional features. The encasing is sturdy enough to carry and protect the cube, along with being transparent to see the cube and cause as little light reflection interference as possible.

The main processor connects the hardware and software. Our goal for the main processor included a simple interface to control the LED Cube using only input and output pins. This is also true for the GameCube controllers, which are controlled using the bidirectional pin interface that they utilize. The main processor hosts all the software, thus the goal for the main processor included support for a good IDE, preferably with C/C++ support. The goal for the input was to develop a working control system that could interact with main processor easily. This goal allowed the project to be expanded into more complex and interactive ways to control a 3D display.

The project design was split into three general blocks. The first block is LED cube, which includes the 10x10x10 LED build and the integrated circuit that controls the columns and layers of the cube. The second block is the hardware/software integration which includes the main processor and the high level software. The final block is the control, which includes interfacing the GameCube controllers to the main processor. Additionally, other aspects of the design included the power and/or power supply, along with the acrylic and wood housing unit for the cube.

The goal was to allow each member to be in charge of each block, allowing each block of the project to develop at each members comfortable pace, without impacting the other blocks. The reasoning behind this goal was to allow each member to utilize their skills most efficiently, and to allow the project to be expanded/changed in the future without requiring a complete overhaul. Knowing that the initial design would most likely need to be heavily modified, each block was kept modular to allow changes to have a smaller impact. To accomplish this goal, the integration and communication of a block with one another was a high priority. Getting the integration design details correct in the initial design proved to benefit the overall project during the development phase.

2.3 Project Requirements and Specifications

This project consisted of a 10x10x10 hardware cube with corresponding circuitry, acrylic casing to cover and protect the cube, and an encased base to the host the electronics. The goal was to have the final product easily moveable and to be usable anywhere near a power outlet. Hardware requirements and specifications include:

Hardware requirements include:

- Pixel Resolution: 10 x 10 x 10 = 1000 LEDs
- LED Volume Dimension: 10’’x10’’x10’’
- Base Dimension: 14’’x14’’x5’’
- Visible Sides of Cube: Top, All Sides (not Bottom)
- Cube Material: Acrylic, Wood
- LED Color: Multiple colors supported
- LED Refresh Rate: 60Hz
- Microcontroller: minimum 8 pin I/O ports, serial communication, 16MHz clock speed, 512 KBytes Flash memory, C/C++ software support
- Color Depth: 8bit
- Operating Temperature: 10-30 Degrees Celsius
- Input Voltage: 120V AC at 60Hz (Standard Wall Outlet)
- Input: GameCube controller

Software requirements include:

- Developed in C/C++
- Final code should be less than 512 KBytes
- Games and animations code should hardware and environment independent
- A accurate and reliable simulation software environment should be developed to allow for immediate testing and simulating of the games and animations
- Have a minimum of three choosable classic games to play
- Have at least one custom 3D specific game developed
- Have multiple animations choosable, interactive animations as well
- Embedded code to support GameCube remote control

2.4 Added Features

Beyond the initial specifications designated for this project, additional features were added to the project due to the group exceeding expectations in the initial development plan. The team decided to challenge themselves by adding the additional feature of sound to help enhance the overall experience of the interactive LED Cube.

Several different ideas were thrown around as to what should be added, but sound effects and music were an easy choice due to the level of immersion they add. Other ideas for additional features included creating custom controllers with basic motion controls, developing better input for a potential drawing like application. There was also the possibility of allowing the user to create and store animations on an SD card which they could plug into the cube and watch. But again, sound effects and music were chosen as a higher priority.

The reason why sound effects and music were chosen as an additional feature is due to many reasons. The biggest reason is that sound effects would provide a level of feedback to the user to help create the feeling that they really are interacting with the cube. Sound has also always played a very large role in video games and without it, a user can easily feel disconnected since sound is almost an expected part of video games. The idea of using music in the project also opened the realm of music visualizers and other interesting software applications beyond just basic playback during gameplay.

When this additional feature was chosen the the group was sure to provide a list of high quality specifications as to not let the music be undermined by poor implementation.

Sound requirements include:

- Allowing user to input their own music
- Must be able to play sound effects and music at the same time
- Must include ROM storage for large audio files
- Both stereo input and output
- Speakers must be internal to the housing
- 32kHz sample rate for music
- 16kHz sample rate for sound effects

3.0 Research Related to Project Definition

3.1 Existing Similar Projects and Products

The LED cube project is a very popular do-it-yourself project. With the growth of communities like instructables, teams and individuals around the world have taken up fun projects that can be done with minimum resources. The LED cube projects high popularity is due to mix of both hardware and software design, along with being a cool expansion to the average engineers first circuit: powering a single LED. Ranging from 3x3x3 to 16x16x16 and beyond, LED cubes are fun but challenging project that allows a lot of design freedom. Hundreds of different hardware and software designs are possible, yet they all come down to the same basic concepts which are detailed in the initial research section. Several different colored/uncolored, square/round, diffused/undiffused choices are available for LEDs alone. Circuits can utilizing endless combinations of multiplexing, shifting, and LED drivers to control the LEDs. Software can range from simple animations to anything you can think of within a 10x10x10 display. Controls can range from none, to accelerometers and controllers. Designs from UCF include several LED cube projects each with its own twist of functionality and design. The GameQube separates itself utilizing the interaction of a user playing a game. A basic template of how an LED cube works and looks can seen in the very popular Instructables 8x8x8 cube. This a standard implementation that includes a single color LED 8x8x8, microcontroller powered cube that plays animations. Different implementations of this standard idea are discussed in the following sections.

3.1.1 Multi-Functional Hexahedron

The Multi-Functional Hexahedron was an LED cube project by UCF Group 5 in Spring/Summer 2013. Their goal was also to provide an interactive LED Cube. Instead of having a controller or interactive games, they implemented features such as accelerometer control and VU meter control. Utilizing RGB LEDs the cube not only plays standard animations but can also be used in accelerometer mode which allows the LEDs to change with the cubes movement. The VU meter mode allows the cube to function as a 3D music visualizer, changing the LEDs based on the changes of music.

The cube would run in three different modes: animation mode, VU mode, and accelerometer mode. In animation mode, animations cycle through various pre-programmed animations. The animations include different effects such as lighting the cube in a diffused manner and quickly cycling through each of the layers. VU mode was a simulation of a music equalizer seen on computers, being implemented on the 3D LED cube.The VU meter was able to read different types of signals received from the input and communicate them to the microcontroller, which would then light the cube in a corresponding pattern. The VU meter was controlled by regular 8mm headphone jack, which could have a smartphone or other audio device play music and show the effects on the cube. In accelerometer mode the LED cube was able to light up and react depending on how it was moved. This included a water feature which had the cube working as if was filled with water and change the lighting of the cube depending on how it was tilted. To allow for this interaction, the housing had to be of a good size and weight.

The hardware design included LED drivers and an ATxmega64 microcontroller which used an SPI converter to connect to a PC for programming. To support the various modes, a VU meter chip and +-5g accelerometer chip was used. To power the cube a combination of wall and USB power was used. They had a USB connector mount to the PCB which allowed them to power the cube from a PC or wall wart with a USB port. The final housing was encased in acrylic panels, allowing the protection and viewing of the LED cube and hardware underneath show in Figure 3.1 below.

The software was written in C/C++ using the Atmel Studio 6 development platform for the ATxmega64 programming. While the cube is powered on, it runs in a loop checking the which of the three modes to run in. The main function would choose the appropriate mode and switch between them. Overall, the final product looks very nice as shown below in Figure 3.1. The main takeaway from this project was the good presentation which the group would like to emulate with a similar looking presentation for the final project. Acrylic will likely be used as well, but with a larger base made of wood or a similar material.

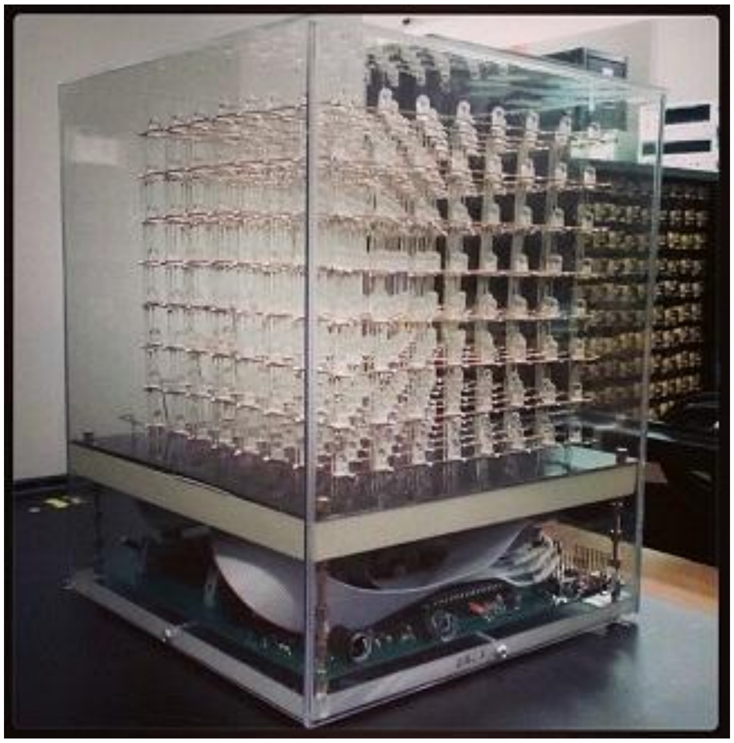


Figure 3.1: The Multi-Functional Hexahedron
Printed with permission from UCF Senior Design Group 5 Spr-Sum 2013[1]

3.1.2 How Not to Engineer: RGB LED Cube

Another interesting reference design was the Hot To Not Engineer RGB LED Cube. Many of the designs reviewed take advantage of the very useful LED drivers with integrated PWM. This cube decided to have the PWM handled by the software rather than the hardware and thus used STP16 LED drivers rather than PWM enabled ones. Using these drivers caused the end product to require more electronic components. The cube is an 8x8x8 RGB so instead of using either LED drivers, it uses twelve STP16 drivers, four for each RGB color. This difference was an option considered however the group decided PWM drivers would fit our needs better. The cube is powered using an internal power supply and held on a base with no acylic casing.. This main difference led to the hardware component of the project becoming more complicated, which is somewhere the team wishes to avoid. PWM LED drivers will be used instead.

3.1.3 Dynamic Animation Cube II

The Dynamic Animation Cube II is a UCF LED Cube project by Fall 2012 Spring 2013 Group 5. This project is continuation of the Dynamic Animation Cube, another UCF LED cube project. The first Dynamic Animation Cube a 16x16x16 LED Cubes was sponsored by the department of Electrical Engineering and Computer Science. The goal of this project was to build a large LED Cube capable of animations to display at UCF.The Dynamic Animation Cube II had the goal to improve the existing cube to include user interactive games similar to our goal.

The Dynamic Animation Cube II had taken into account the amount of current the cube had the potential to consume in which the previous group was unsuccessful with their design. Containing 4096 LEDs, over four times, controlling and providing power with a good refresh rate was the main obstacle. The new design for the Dynamic Animation Cube II included placing the LED drivers in parallel, which allowed the group to address specific drivers rather than having to shift data across an array of them. This design gave the group more control over the cube and helped increase the refresh rate of the entire cube as well.

The hardware for the Dynamic Animation Cube II was controlled by a AT32UC3C2512C microcontroller featuring 66 MHz clock speed and 512 KBytes of Flash memory. The group programmed and debugged the AT32UC3C2512C using Atmel’s AVR Dragon and a PC. Redesigning and improving upon the previous design included using a decoder to individually address all of the LED drivers. The previous group design had daisy chained them in sequence. This design change allowed the group to select one driver to communicate with instead of having to shift data into all of the drivers. Many features were added as well including the control of the animations through the use of the Wii Nunchuk. The cube was powered using a standard PC power supply to power their boards.

The software used for the cube was written in C utilizing Atmel development tools and software framework. The group had programmed the software to run in a state machine. The states including one state for each of four games, a lobby, an ambient state, and an instance of Conway’s Game of Life. To control the states the user input from a Nunchuck controller was used. The software was configured to have a refresh rate of around 90 Hz. To communicate with the cube, Serial Peripheral Interface was implemented.

Due to the shear size of the LED cube, the construction and housing of the whole project was a much bigger factor than our 10x10x10 cube will be. A full wooden base, desk sized, was needed to hold the cube and host the wiring and electronics. The full LED cube sits on top while all the wiring and electronics are hidden. To allow sturdy construction of the giant cube,the group used extruded acrylic rods.

3.1.4 Kevin Darrah 8x8x8 RGB LED Cube

Through research of other people building their own LED cubes, Kevin Darrah came up as a great example. Kevin Darrah not only built an 8x8x8 RGB LED cube, but also provided multiple videos of his process throughout the build. His goal was to build the cube in such a way that he could provide others with pre-made kits that would allow them to build their own LED cubes.

In the assembly of the cube, Darrah first needed to solder the RGB LEDs together by their three cathodes into columns of 10. This would make all the LEDs in a column share a common cathode for the red, green, and blue diode. Darrah used a simple wooden jig to hold the LEDs and wire in place during soldering. Darrah next soldered the columns into slices by soldering all anodes in a layer together along a piece of wire. After that, he soldered the slices together to make the cube. The wires used to hold the cube together were copper colored, which is something the group decided to try to avoid by using silver colored wire.

The hardware of the LED cube was controlled with the Atmel ATmega328P-PU microcontroller with Arduino Bootloader which has 32 KBytes of flash memory and 20 MHz clock speed. It was programmed using Arduino code on a PC. Darrah used the Arduino code to control the LEDs and run animations on his cube. Through this, he could create new animations from scratch or light up just a single LED if needed. The cube was powered by a 120V AC to 5V DC at 10A power supply. Although Darrah created an 8x8x8 cube, it is scalable so that it could be made to be 10x10x10 like this project intends to be. Overall this project serve as a great reference for the assembly technique in which Kevin Darrah demonstrates several useful methods in his video series.

3.2 Relevant Technologies

3.2.1 Pulse Width Modulation (PWM)

One of the major difficulties this project involved was how to adjust the brightness of each LED. The brightness of each LED is determined by current passing through it. The direct approach would be to vary the resistance or voltage of the circuit to get a current resulting in the desired brightness. However, this is very impractical for this project since we needed the brightness to be able to be controlled and varied over time. A better solution to the problem for this project was a method called Pulse Width Modulation.

The basic concept of Pulse Width Modulation involves providing a constant current to the LED, but then pulsing the current through time to create the illusion of a lower brightness. For example, if the LED is pulsed with a constant current so that there is only current flowing at an average of half the time, then the LED will only appear to be half as bright as if there was the same current level flowing at all time. Of course, this pulsing of the current must occur at a very high frequency or else the blinking will become noticeable to the human eye, effectively ruining the illusion of a lower brightness and instead provide a simple blinking effect. The Duty Cycle of the pulse is defined as the percentage of time there is current flowing through the LED. By varying the Duty Cycle on the LED, we can effectively vary how bright the LED will be.

In summary, the reason this technology is utilized in the final overall design is it provides a way of adjusting the brightness of an LED without having to modulate the circuits resistance or input voltage, which would require a significantly more complex circuit design.

3.2.2 Serial Peripheral Interface Bus (SPI)

This project includes many different kinds of devices doing their own individual tasks in order to split down the work into manageable pieces. Because there are several unique devices like this, this project involves the need for these devices to communicate with each other. One of the standards in device communication for problems such as this is the Serial Peripheral Interface, also referred to as SPI.

The Serial Peripheral Interface was first developed by Motorola and is a synchronous serial communication between a master device and a slave device. The interface is also commonly known as the Four Wire Serial Bus, since it uses mainly just four wires. The wires, or connections, are the Serial Clock, Master Out/ Slave In, Master In/ Slave Out, and Slave Select. Multiple Slave Select lines on the master also allow for communication between more than one slave device as shown in Figure 3.2 below. The communication process starts with the master setting the Serial Clock to have a frequency less than or equal to the max frequency the slave device will support. Data is then sent in both directions on the Master Out/ Slave In (MOSI) and Master In/ Slave Out (MISO) data lines. This project does not always require data to be sent in both directions, but SPI still supports it.

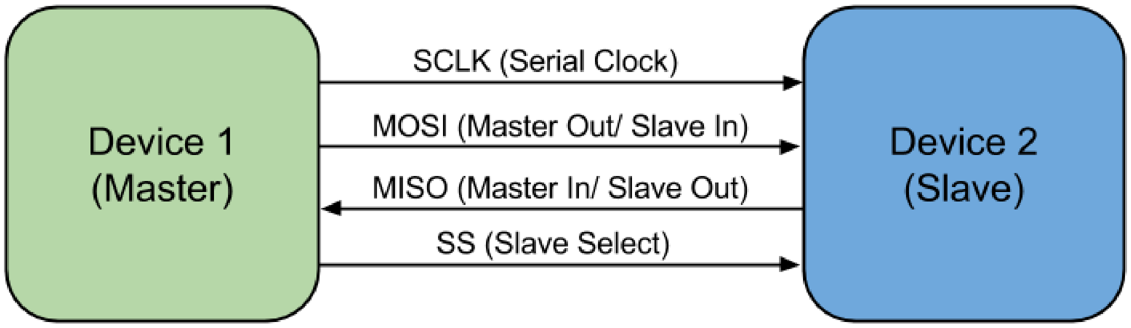


Figure 3.2: SPI Master and Slave diagram

Most of the microcontrollers researched for use in this project provide special functionality for this interface, like the MSP430. This technology provides standard for device communication that most ICs have already been designed around and optimized for. This technology is used for programming of microcontrollers used in this project.

3.2.3 Universal asynchronous receiver/transmitter (UART)

Another standard in device communication is Universal asynchronous receiver/transmitter, referred to as UART. Similarly, Universal synchronous/asynchronous receiver/transmitter is a UART that can communicate synchronously, referred to as USART. UART uses is a type of hardware that allows for parallel and serial data translation. Utilizing a serial port such as RS-232, UART allows for bytes of data to be transmitted and received. This requires two piece of UART hardware and that each contain a transmitter and a receiver. Unlike SPI, UART contains no designated slave or master. A disadvantage of UART is being asynchronous meaning the clocks of each hardware need to synced. Without a clock line, data can be corrupted due to mismatching clocks or mismatching baud rates. SPI has a dedicated serial clock to prevent this issue. While using UART will allow for less wires, the implementation would be harder for the clock requirements. SPI will use more wires but the implementation will be easier after master and slave setup.

Most microcontrollers support both of these technologies. UART is used in places where SPI is not available, however many devices will require an SPI setup. For programming and communication with a USB Host, SPI is required, thus this technology became familiar to the group. Overall, SPI technology was favored over UART.

3.2.4 Persistence of Vision Display

One of the specifications of this project was a cube display with pixel dimensions of 10x10x10. This brings many challenges, but perhaps one of the biggest was how to wire up all one-thousand LEDs so that they could be individually controlled. This is not even mentioning the fact the RGB LEDs require 3 lines to appropriately control the color. The straight forward approach would be to connect individual wires to each LED. However, this approach is not very practical since that many wires will become very messy and reduce the visibility of LEDs towards the back of the cube that may be blocked by wires. This would also require the states of all leds to be controlled directly by the I/O of some micro controller. To reduce this overwhelming load of LED control, the phenomenon of Persistence of Vision can be used. Not only will the visibility be affected by physical LEDs and wires, but the reflection and ghosting of the LEDs onto each other will also affect the visibility.

Persistence of Vision directly refers to the way in which the human eye works. The human eye can easily recognize an object blinking if it’s at a small frequency (around 5-10Hz). However when the blinking is sped up to much higher frequency, the human eye can not keep up, and the perceived image is a constant sort of mixture of the on and off state. The reason for this mostly lies in the phenomenon that images stick around on the human retina for a few fractions of a second before the image is lost. So when something flickers off for periods of time smaller than this decay, the image never appears, to the human eye, to have flickered at all.

Despite all the good, there is also a very important factor that should not be overlooked when determining the number of LEDs to be on at any given time. Although there are many benefits from multiplexing the LEDs, like a reduction in needed I/O, overall brightness of the LEDs will be reduced. If an LED is only on one tenth of the time, it will appear to be ten times less bright. This is not an unsolvable problem though, as the current can simply be increased until that one tenth brightness is suitable for the display. The problem however, comes into play when this current starts to break the max current allowed by the LED driver. By multiplexing the cube into layers of ten, the max current (in terms of the appearance of brightness) is effectively cut down by a factor of ten. Many options are available to adjust the brightness of the LEDs

Overall, the advantage of this technology, is that we cut the control load down by a factor of ten by only lighting up one tenth of the cube at a time. By cycling through and lighting up all ten sections individually, but at a very high frequency, the human eye perceives the entire cube to be on at the same time. Now the load of all the control is split up over time, which makes the problem much easier to tackle. This also allows for a reduction in the total amount of wires needed, since all ten sections can be tied together with a separate control piece sending power to only one of the sections.

3.2.5 Bluetooth

One of the challenges of creating this playable LED cube is in determining how it will be controlled. Options considered included wired controllers, wireless controllers, built-in arcade style controls, motion controls, and camera viewing controls. The team wanted to have the user be able to control the cube wirelessly, where the input device communicates directly with the microcontroller. The team initially considered that the best way to solve this problem was to utilize Bluetooth in the design.

The basic idea of Bluetooth is that data can be transferred wirelessly between devices within a certain range through the use of wave radio technology. The range can reach up to 100 meters depending on the specific device. The range desired is at least five feet away from the cube in all directions. Each device needs to have its own Bluetooth chip in order for communication between them to occur. For the purposes of this project, the input device that will be attempted to use, a Nintendo Wii remote, already has built-in Bluetooth capabilities. Other bluetooth enabled devices such as mice, keyboards, and other generic game controllers can be used in the future.

Many bluetooth forms were considered for this project. Simple bluetooth modules were the first to be considered. In choosing a Bluetooth chip to use with the microcontroller, it is necessary to understand what is compatible with the Bluetooth type found in a Wii remote. Wii remotes use the HID profile (Human Interface Device) of Bluetooth, which allows for user input and is common in many input devices such as controllers and keyboards. Therefore, if the project utilizes bluetooth chips it would need the microcontroller’s Bluetooth chip to support the HID profile in order to receive signals from the Wii remote.

Many microcontrollers have hardware that allows for simple bluetooth connections, however this will not be sufficient for the needs of the project. Also this would add another constraint in picking the microcontroller which would not be necessary. The final consideration for establishing a bluetooth connections was a bluetooth dongle and a USB host. A USB bluetooth dongle can be used to connect bluetooth devices to the USB host. The USB host will allow the dongle to communicate an active bluetooth connection to the microcontroller. A USB host module would use SPI to communicate with the microcontroller, and the bluetooth module would communicate with the bluetooth controller.

In the end, this technology is not included in the final design since we decided to use Nintendo Gamecube controllers as the user’s input device. Nintendo GameCube controllers offer both wired and wireless options. The main benefit is the wireless options are handled internally by the controller’s wireless receiver, and no bluetooth or wireless code is needed.

3.2.6 SD Card

As part of the added sound effects and music feature, there needed to be a way to store large amounts of audio samples onto some sort of ROM that would not be erased when the system does not have power. Of course the actual microcontroller used to process the sound has ROM that could be used. Most microcontrollers would provide more than enough ROM for program space so then the remainder could be used to store sound samples. However, the problem with this method is that music requires hundreds of thousands of samples to be able to play just a few minutes of audio. This is even more a problem since our specifications required that the music be sampled at a high rate of 32,000 samples per second.

The easiest solution to this problem was to utilize an external chip to store data. An SD was an easy choice since its data can be easily manipulated through any PC with an SD card slot. This would allow for the music data to be altered and programmed quickly and efficiently. An SD card also does not even require a file system to hold different files. Although this makes it difficult to transfer files to the SD card, image writing software can still be used to directly manipulate the data on the SD card at any specific address needed to be edited. A lack of a file system also helps keep things simple on the programming side of the sound processor. All the processor has to do is start reading data at a starting address and linearly follow along for the next audio sample. An SD also adds plenty of memory for storing of the audio files. This means the microcontrollers storing the software can focus their memory on the code.

Interfacing with the SD card is also very easy for a microcontroller. All of the microcontrollers used in this project run at a logic level of 3.3V. This is exactly the same logic level that an SD card runs at. SD cards also support a simple SPI interfacing mode for microcontrollers to easily access the data on the card. This means that only two pins are necessary for data communication between an SD card and a microcontroller. Most microcontrollers also include hardware SPI features which makes the communication even easier.

SD cards are also very cost effective for the amount of data storage they provide. This is mostly due to how common they have become in other consumer electronics. SD cards come in different package sizes and specifications as well. For this project both a micro SD card and standard SD card will work. However, due to a standard SD card being easier to connect to a PC, this form factor was ultimately chosen. SD cards also have different classes which refers to the data speeds it can handle. For this project, most of these details do not matter as they require the use of a more streamlined interface supported by the card other than the convenient SPI interface which will be utilized.

3.3 Strategic Components

3.3.1 Light Emitting Diodes (LEDs)

The volumetric display described in the specifications of this project required some sort of light technology to create images. The most obvious solution to this was to use Light Emitting Diodes, since they consume much less power than other lighting technologies. By definition, they are just as the name indicates (a diode which emits light when a current is passed through it.) However, one possible downside was the cost, since the project specifications will require around one-thousand individual LEDs.

One important design aspect of all LEDs is clarity. Most LEDs are simply clear to allow light to pass through without being disrupted or scattered. However, for an LED providing multiple colors by combining a red, green and blue LED, it may be desirable to have the light scattered a bit in order to mix the three base colors in single noticeable color. Because of this, LEDs typically come in a clear or diffused design. The diffused design looks a little cloudy, but scatters and mixes light better, whereas the clear design simply lets the light pass straight through. The difference can be seen in Figure 3.3 below.

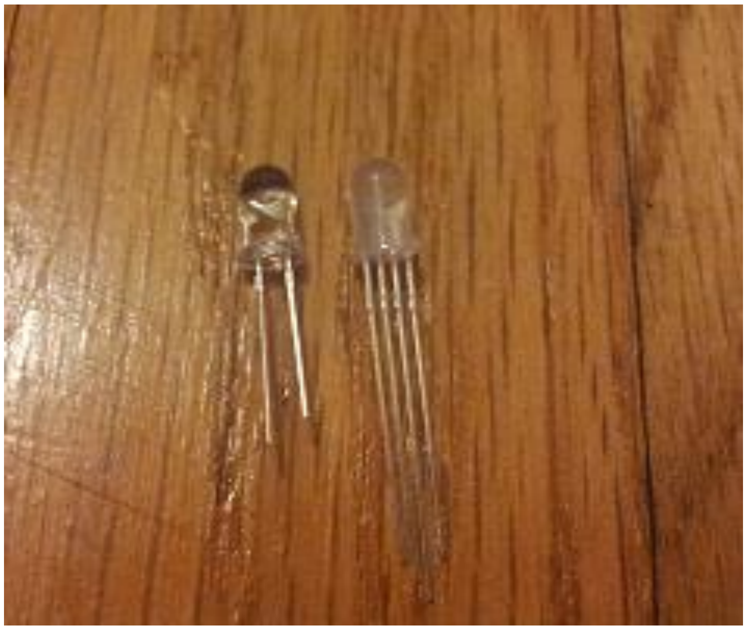


Figure 3.3: Clear (left) vs Diffused RGB (right) Light Emitted Diodes

It was important to make sure that, when ordering any LEDs, that they were rated at the current level we needed and could provide the brightness level outlined in the project's specifications.

3.3.1.1 Round 5mm RGB LEDs

One of the more common shapes of LEDs is the round design as shown above in Figure 3.3. The advantage of this is that it provides a decent and consistent viewing angle from nearly all directions except the bottom view. However, for this project, a viewing angle from the bottom was not needed due to overall construction of the display mounted on a solid base. An RGB LED like this also provides capability to provide a full range of colors by mixing the light from a red, blue and green LED. Being able to provide multiple colors was also one of the specifications for this project.

Because this type of LED is actually composed of three separate LEDs in one, there are four leads sticking out at the base, instead of the typical two. The longest lead is a common anode or a common cathode. The remaining three leads consist of the anode/cathode of the individual red, green and blue colors. providing different brightnesses to each of the color leads allow for the creating of a wide array of colors. The additional leads also account for why the LED is mostly only found with a 5mm diameter and not a 3mm diameter, which is also common among round LEDs. The additional leads require more space to fit, so a 3mm can almost never be found. In summary, this type of LED provides multiple colors that can be controlled by logic devices and provides a viewing angle well suited for a volumetric display that can be walked around as outlined in the project's specifications.

3.3.1.2 Square RGB LEDs

Another common design for RGB LEDs is a square shape which looks very different than the round LEDs. The base of the LEDs are square with a round dome on the top. Although the top is round, just like any other round LED, the sides are more square and not designed to emit light as much as the top. These LEDs are most useful for projects with some sort of two dimensional display since they can be easily tiled together in a grid pattern. The square design also fits very nicely into the RGB design discussed in the round 5mm RGB LED section, in that the four leads can easily fit into one of the four corners. It is important to note that the leads on these LEDs are much shorter than the leads found on the round LEDs. This is an important factor to consider when designing the wiring of the large volumetric display. A lot of additional wire would need to be used to make up for the short leads reach. However, these LEDs are also very friendly to the prototyping environment, as their shape can be easily popped into place on a breadboard.

This type of LED provides multiple colors that can be controlled by logic devices just like the round RGB LED, but the viewing angle is not quite as good from the sides. The small, spaced out leads on the LED would also make it hard to construct a large cube design. Because of these reasons, this type of LED was not the best choice for this project.

3.3.1.3 Multi-Color Flashing LEDs

One of the requirements specified for this project was to be able to display multiple colors at any given location in the volumetric display. In order to do that, LEDs with multiple color possibilities needed to be used. One option that met this requirement was a round LED that flashes between the primary colors of red, green and blue. The main difference was the ability to flash multiple colors at once, rather than one RGB color. Unlike the round 5mm RGB LED option discussed above, this LED only has two leads and the colors are changed in a flashing pattern. These LEDs can be found in both a fast flashing design and a slow flashing design. This however, causes a problem when it comes to the controlling of the colors displayed.

The only way to really control the color would be to design a logic device that ran in sync with the flashing of the LED. The logic device would then use PWM to change the brightness of the LED. If the logic device uses a higher duty cycle when the LED is currently on red and then drops the duty cycle when the LED is any other color, the LED would appear to be only red. Through this technique a wide array of colors could be created. The downside to this is the major challenge of getting the logic device in sync and then staying in sync with the LEDs flashing. Each LED may have a slightly different flashing pattern and just the slightest variant can cause the illusion to fail entirely. However, with only two leads, the wiring of the actual cube would be much simpler, as well as less I/O pins will ultimately be needed.

Despite the easier wiring, the control over apparent color of the LED is just much too difficult to try and implement. This is especially so in this project since the final design needed one thousand LEDs to all be perfectly controlled and in sync.

3.3.1.4 Single Color LEDs

The most common LEDs are probably the simple one color round LED which look nearly identical to the round RGB LEDs. They can be found in both 3mm diameter and 5mm diameter. These simple diodes have only two leads (one cathode and one anode). The beneficial part of this simple design, is that it does not require as much I/O to control as one of the RGB LEDs mentioned earlier, however there are a few things lost as a result of this simplification.

It’s important to note that in order to meet the color requirements outlined in the specifications for this project, multiple single color LEDs would need to exists at a single spot in order to generate a range of different color possibilities. While this is not impossible to do it is very impractical, since it would result in a very bulky display design. There would also be a problem in getting the colors to mesh together since they are not together in any sort of diffused enclosure. Having to use multiple single color LEDs in one spot would also negate the benefit gained from the simple to lead interface. There would now be two leads per color desired to mix.

Despite the initial conceived simplicity, these simple LEDs do provide any practical way to display multiple colors even when trying to group them together, thus failing to meet the specifications for this project.

3.3.2 Micro-Controllers

For this project, there were many specifications that required complex logic computation and I/O control at high speeds. In order to meet the specifications, several microcontrollers were implemented into the design. This includes microcontrollers for both the logic involved in running the volumetric display and running more complex computations for game logic and artificial intelligence. Important things to keep in mind while choosing an appropriate microcontroller are the power requirements, I/O capabilities and processing speed. It's also important to keep in mind the fact that these devices will mostly like need to communicate with each other. Separate microcontrollers were used for the LED logic circuit and main microcontroller board.

3.3.2.1 MSP430

For some of the less complicated computing and logic tasks in this project, a less powerful microcontroller can be utilized. This allows for less power consumption compared to using a more powerful microcontroller for the smaller remedial tasks. A smaller microcontroller also usually means a smaller price tag which will lower the overall costs involved in this project. The MSP430 line of microcontrollers by Texas Instruments can accomplish just this. TI provides a convenient and affordable way to quickly use and program a simple microcontroller with the MSP430 LaunchPad. This microcontroller is one the group was very familiar with and comfortable utilizing.

In particular, the logic and LED control of the volumetric display benefited greatly from this small device. The MSP430 value line utilizes a very low power consumption while running and also supports a sleep state that allows the chip to consume virtually no power when the chip is not in use. It's because of this sleep state that the gains in power consumption will really be evident since the logic for the display will only need to be computed in small bursts at a time.

There are many different variations on the MSP430, but they all stem from the same 16 bit RISC type architecture. Factors that vary include ram size, flash size and number of I/O ports. These variations can be seen in Figure 3.4 below. All configurations will use the general software but the hardware will support different number of pins, different amount of memory, and different clock speeds.

Model	RAM (kB)	Flash (kB)	Clock (MHz)	I/O Pins
MSP430g2553	0.5	16	16	24
MSP430g2203	0.25	2	16	24
MSP430g2303	0.25	4	16	24
MSP430f5529	8	128	25	63

Figure 3.4: MSP430 Model Differences

A few things to keep in mind while selecting between the different MSP430 varieties include amount of RAM, clock frequency and number of I/O pins. A large amount of RAM is required especially if the MSP430 will need to store the states of all LEDs in memory. The minimum amount of RAM in the case of this project would be 1000 bytes, or one byte per LED. The clock rate should also be closely looked at. A faster clock rate, means the serial data transmission to the LED drivers can be performed at a faster rate. It's also important to note the limitations of the LED drivers, since for example the TLC5940 has a max serial data input of 30 MHz. This means that the clock frequency of the chosen MSP430 should not exceed this maximum. Finally, the number of I/O ports is important when dealing with control over lots of devices. For example, any sort of external memory buffer will need I/O pins for the address and the data making a 16kB memory IC require a total of 22 pins. Then additionally even more pins would be required to control the LED Drivers, thus to be safe more pins would be good to have.

Another factor to consider with the MSP430 is the development tools that can easily be found for it and easily used. The group had experience using these tools which helped in the development. Texas Instruments provides an inexpensive development board called the launchpad, which is aimed at helping small projects get off the ground and running is a very short amount of time. There are also several free compilers and integrated development environments available to aid in programming these particular micro controllers (and also others like it). A compiler can be found in both assembly and C. The C compiler is a huge benefit due to a much higher level logic concept programming capability (more similar to human concepts of logic and not just a broken down set of instructions). TI Code Composer Studio allows for immediate and convenient C programming to the microcontroller. The MSP430 also has a very large and helpful support ecosystem, with resources to utilize if any questions should arise while trying to integrate the microcontroller into this project. In conclusion, the MSP430 provides a cheap, low power consumption and easy way to control some of the more smaller logic aspects of this project. In particular, the MSP430f5529 covers all the needs of the project in terms of RAM, clock frequency and amount of I/O.

3.3.2.2 Main Microcontroller

The main microcontroller decision was very important for this project as it brings the hardware LED cube, the controller input, and the software together. The main factors into deciding on a microcontroller included the available memory, the input/output pins, and the speed. One of the goals for this project included being able to play multiple animations and games. To allow the cube to play multiple animations and chose between several games, along with supporting a input controller, our microcontroller choice needed to support enough flash memory to store all the code.

Based on the research of other LED cube projects, most projects with 64 kilobytes of memory are able to store sufficient memory for several basic animations. Some projects like the Multi-Functional Hexahedron seemed to run into memory restrictions when adding additional features. To support all the animations this project runs on the cube, the decision was made to have at least 128 kilobytes of memory for animations alone. Based on our initial software design and emulation in our virtual cube environment, the code size to make an animation can be very efficient and redundant. Code size for games however, requires a lot more code to support AI and game logic. Thus the memory requirements for full game and control support was estimated to be at least 256 kilobytes, giving a total memory estimation of 384 kilobytes. To be safe, all memory estimations were very generous, and because the most common next available increment from 384 kilobytes was 512, 512 kilobytes was deemed to the best available option. A large contributing factor to the memory requirements was the fact that most microcontroller specifications are only available at certain logic increments, while 384 kilobytes is available, it is much less common than 256 and 512. This trend also meant that microcontrollers with a low clock speed and low pin count, also had relatively low memory. In the end, after evaluation each microcontroller requirement, the logical increment that fit our requirements the best was chosen.

The input and output requirements included the main output pins to the LED cube, and the input pins from the controller interface if a UART design was chosen. Initial design deemed our output format from the microcontroller would be 10 bits per clock cycle, 1 bit per pin. For a possible controller choice, 1 transmit pin and 1 receive pin could be required along with power and ground. For controllers using SPI and independent power, four SPI pins will be needed. This made the initial estimation to be 14 output pins assuming the microcontroller will only power the controller interface directly at most, forcing the cube to get power directly from the power supply.

The clock cycle and bit size ended up being less important than were expected. Similar projects used 32-64 Mhz speeds for their microcontrollers, which based on the our hardware would be more than enough as our cube is believed to a refresh rate of about 16 Mhz. Based on the microcontrollers logical increments, any microcontroller with 256 kilobytes will usually have a clock cycle speed of at least 32 Mhz. The bit size was heavily tied to the type of architecture the processor was. The group has experience using the TI MSP430, a 16-bit architecture, and although the team is comfortable and familiar developing with it, the team would also like to have experience using other hardware and tools. The manufacturer of the microcontroller is the main factor in the development tools that will be used for the project software. The MSP430 uses TI's Code Composer, which the group has experience using and will be using to program our MSP430 controllers for the LED cube logic circuit. Thus, other manufacturers besides TI were favored for the main microcontroller decision. Some of the most prominent manufactures include Atmel, Cypress, and Microchip Technology. The main requirement when evaluating different manufacturers included the development environment. A goal for this project was to be able to directly program our microcontroller using a C/C++ IDE. Atmel microcontrollers seemed to be a very popular choice among our projects due to their C/C++ compiler, community support, and architecture choices. Atmel produces microcontrollers from two main architectures, ARM and AVR. Because the final software will be written in C/C++ the architecture was not a very important factor. Instead each architecture was evaluated in their ability to store memory and save space. AVR architecture comes in both 8-bit and 32-bit, yet the 8-bit microcontrollers are only available with up to 384 KBytes of memory. ARM architecture is comes in 32-bit and 64-bit. The benefit of having less bits per instruction directly contributes to the how much space the code takes in memory, but at a cost of performance. For our purpose, the performance of a simple 8-bit processor would be sufficient; however the memory options are sparse.

Ultimately the decision came down to comparing the available options of Atmel microcontrollers of both AVR and ARM. Atmel produces the device family of 32-bit AVR UC3 with up to 512 KBytes of flash memory, 48-144 pins, and up to 66 Mhz. These specifications meet all of the initial requirements and led to the first candidate, the AT32UC3B0512. Atmel's ARM choices included the ARM Cortex family and the ARM7TDMI. Although both of the these are 32-bit, ARM features Thumb and Thumb2 instruction sets to save space. The thumb is a 16-bit instruction set based on ARM32, allowing certain processors to run in this mode and save code size, but also results in reduced functionality at times. Thumb2 is variable length instruction set that came execute both 16 and 32-bit size instructions, allowing the best of both worlds of Thumb and ARM32. The ARM7 line of processors features Thumb mode along with ARM32, while ARM Cortex is available with Thumb2. Due to performance speed not being as important as the memory requirement, the ARM7 line seemed to be more appropriate due to reduced code size. After further research, the ARM Cortex line being more current has a stronger support community and easier development environment. Many of the ARM7 processors still require assembly code for initialization unless directly implemented by the IDE. This lead to the two ARM candidates for our microcontroller choice: AT91SAM7S512, an ARM7, and ATSAM3S8B, an ARM Cortex-M3. The features of all three initial microcontrollers possibilities are shown in Figure 3.5 below.

Model	**Family**	**Architecture**	**Flash (kB)**	**Clock**	**I/O Pins**
AT32UC3B0512	AVR UC3	32-bit AVR	512	64	60
AT91SAM7S512	ARM7	ARM32/Thumb	512	64	55
ATSAM3S8B	Cortex-M3	Thumb2	512	64	64

Figure 3.5: Atmel Microcontroller Candidates

The final factor in choosing a microcontroller was the programming interface. To program the microcontroller it needs to be mounted on a board with the pins mapped and a serial port programmer that can connect to USB. Research led to the determination that AVR microcontrollers allowed the most freedom for programming. A very popular utility to program AVR microcontrollers is the AVRDude software. It allows for easy downloading of programs to the AVR ROM using in-system programming. The options for ARM microcontrollers are much smaller and require more expensive programmers. To utilize AVRDude, a compatible microcontroller and ISP programmer is all that is needed. Based on this information, an Atmel AVR microcontroller was favored. Based on the AVRDude supported microcontrollers, the AT32UC3A0512 was the only 512kB choice. The AT32UC3B0512 is not directly supported by AVRDude, but shares nearly identical specifications as the B0512. The main differences include the AT32UC3A0512 having 144 pins rather than 64 pins, meaning it still fits the requirements of our microcontroller. While it contains a lot more pins than are necessary, this microcontroller allows the project to complete the required tasks and be easily programmed.

The final board containing the microcontroller would require the pins to be mapped on a printed circuit board. This would lead to having the testing of the LED cube being delayed until the LED cube, LED logic circuit, and microcontroller board were all built. To allow the software and bluetooth to begin development and testing, more convenient development tools will be used while the final design is being built. Development boards that replicate our microcontroller design will be used for initial software testing. Boards like this available include Arduino, Raspberry Pi, and Beagleboard. Beagleboards and Raspberry Pi both include very powerful hardware with features such as GPU hardware and HDMI output which extend outside the scope of our project. Arduino on the other hand, has many embedded microcontroller options. Arduino boards are open hardware embedded microcontroller boards with input/output pins mapped along with easily programmable interfaces of micro/mini USB. The boards are programmed using the Arduino IDE in C/C++ and include a large community of support. A popular choice for smaller cubes is the Arduino Uno, a ATmega328 powered 32 Kbyte flash processor, which falls short of our initial requirements. This led to the team to the Arduino Due. The Arduino Due is powered by an Atmel SAM3X8E ARM Cortex-M3 CPU. Comparing this to the initial microcontroller choices, it is a very strong match and almost identical to the ATSAM3S8B. The Due implementation of the SAM3X8E has 54 input/output and a 84 Mhz clock versus the ATSAM3S8B's 64 input/output and 64 Mhz clock, both being more than enough. Other features of the Due included micro and mini USB port for programming and a power jack.

Overall, the Arduino Due, is almost an exact match to the project's microcontroller design. The Arduino Due has 512 kB memory and an integrated USB host to allow for a USB bluetooth dongle. Thus, the Arduino Due proves to a very useful development tool to use while designing and building our microcontroller. A main theme that the team wanted to implement was to keep the project modular, allowing the hardware cube, microcontroller generated software, and controller input to be developed independently. The Arduino Due allows the team to immediately start developing the software with the help of the virtual environment software the team is utilizing.

After further research, a better alternative was to go to the ARM microcontroller and Arduino for testing. Comparing the TI development support to ARM, it became clear TI provided more tools for building a custom microcontroller design. TI provides many documents on design decisions when using their microcontrollers. Additionally, TI provides launchpad as cheap development boards for their microcontroller. This provided a cheaper and more appropriate testing tool than an Arduino. Thus the AT32UC3A0512 was discarded in favor of the TI Tiva TM4C123GH6PM. This chip features a Cortex-M4 processor that runs the ARMv7-M Thumb2 architecture. This provides the same benefits as the Atmel ARM chip considered. The Tiva chip has 256 kilobytes of flash memory, 80Mhz clock speed, and 43 available GPIO pins. While this chip has less memory than the Atmel choices, it has a faster clock speed, and comes in a smaller package of 64-pins. This allows for easier configuration while still having enough pins for the project.

The TM4C123GH6PM was used as the project's main microcontroller. The board to host this microcontroller was a separate board to allow the project to remain modular. The board maps the necessary pins to both the LED circuit and the controller hardware. The TM4C123GH6PM fits all of the initial microcontroller requirements: memory, input/output, speed, and C/C++ development environment, making it a good choice as the main microcontroller.

3.3.3 Registers

This project involved many different types of logic devices and small logic circuits. One of the necessities that comes out of designing such circuits and interfaces is a way to remember certain logic states. A register allows for small memory storage that is useful in many logic circuits. This helped tremendously in this project for many aspects such as keeping track of any large buffers that may be used for the volumetric display.

There are many different types of registers with different interfaces, despite the fact that they are all just simple memory storage devices. Things to consider when picking out what types of registers to use are how many I/O pins are required to control it, how much memory can be stored and the read/write rate.

3.3.3.1 74HC595 8-bit Shift Register/Latch

A more common type of register is what is known as a shift register. This type of register works by writing a series of bits serially, but then allows access to all stored bits at once, once they are written. The main advantage of a design like this, is the reduction of necessary I/O pins to control the memory values. The memory write is simply time shared bit by bit. This was very useful in the control of the volumetric display, as it can act as a way to gain more I/O capability. The volumetric display requires a lot of data to be sent to the display all at once. However, it is important to note that the max currents for the output of these registers may not be high enough to power an LED directly, but this can be worked around by having the register effect a transistor instead of the LED directly.

One additional feature that can be very useful for shift registers is the use of a latch. This comes in handy when the output of the registers should not change until all of the new bits have been shifted into place. An additional I/O pin is used to latch the new output bits, whose values will remain unchanged until the pin is triggered again. Specifically, the chip being described here is the 74HC595. This design is shown in Figure 3.6 below.

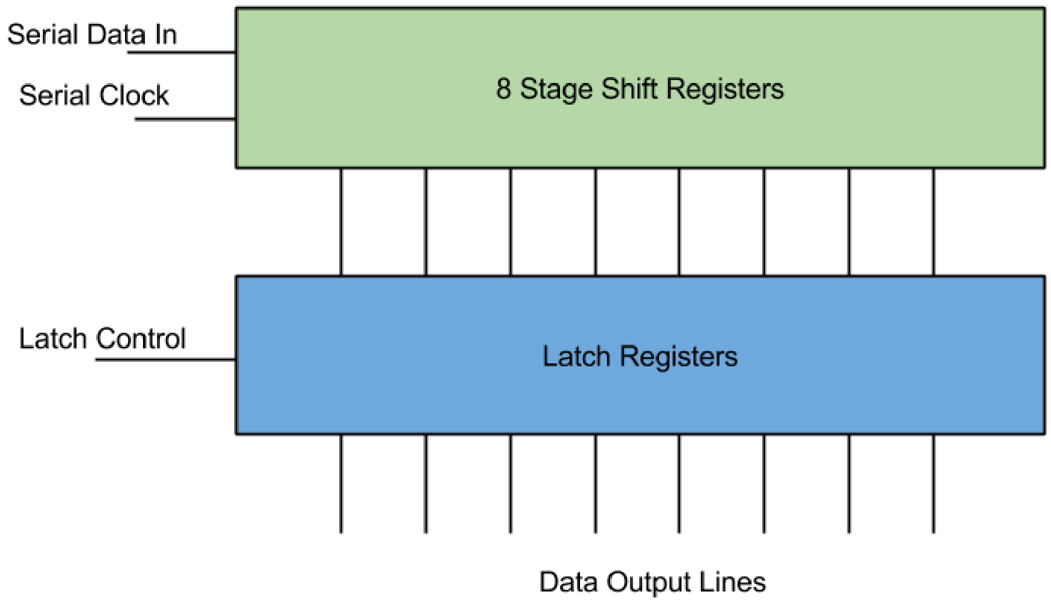


Figure 3.6: Shift Register with Latch Diagram

The use for this particular type of register for this project was to enhance I/O capabilities in a micro controller. However the current limits on the output made it very hard to utilize for powering display elements. There are better options discussed later in the research.

3.3.3.2 FIFO Registers

Another common type of register is what is known as a FIFO register (First In First Out). This is very much like the shift register, in that bits get shifted into memory. However, unlike the shift register, the bits are not then output all at once. Instead, the bits get output serially just like they were input. The advantage to this, is that there is a greater reduction in the number of pins to output data. Because of this, small FIFO register chips can hold a large amount of data without getting ridiculous in how many pins there are. These registers also differ, in that the data may be input and output in quantities more than just a bit. For example, the input and output could be whole bytes.

What made these registers desirable for this project is their ability to hold lots of data that can be accessed with minimal I/O. Particularly, the volumetric display will most likely require a large amount of data, which holds the state of each element of the display, to be stored in a buffer. This can also be used as a way for large chunks of data to be communicated between two different microcontrollers with little synchronization needed between the them.

Overall, although it would come in handy for buffer storage between devices, they are not completely necessary. The RAM found in the micro controllers themselves can easily be used as buffer storage instead, which would reduce overall cost.

3.3.4 Integrated Circuits (ICs)

A lot of the requirements needed to meet the specifications of this project involved complex circuits. To help remove the weight of designing every little element down to transistors, resistors, capacitors, diodes, etc, Integrated Circuits were utilized.

Integrated Circuits reduced a rather complex circuit into a simple chip with stress and use specifications outlined in a data sheet. Some of the advantages of using chips like these is a reduction in the overall cost, simplification in the final assembly of the project and reduction in the size of the final circuit layout. With lots of IC to search through, it was relatively easy to find helpful chips with specifications that meet this project’s needs.

3.3.4.1 TLC5940 LED Driver with PWM

One of the problems that came from designing the volumetric display in this project was the control of the LEDs. This can be implemented simply by attaching each of the LEDs anodes to a transistor with a high enough current rating to support the power required to obtain a desired brightness. However, this is very impractical since the number of LEDs required will be at least one thousand. A shift register, like the one explored in research above, could help with this, but it definitely will not be able to provide enough current to the LEDs. Another IC that solved all these dilemmas was the TLC5940 LED Driver.

This chip allows for up to 16 outputs rated at an impressive max current level, which will provide the LEDs with more than enough power. One of the things that makes this chip a little more unique is the fact that it actually does not provide the power directly, but rather acts as a switch to each output, connecting it to ground. This means that if LEDs are hooked up to this chip, then they must be attached by their anodes. The chip also utilizes PWM to control brightness of each output and internal memory to remember the states of each output. The memory is input serially, which reduces the amount of I/O required to interface with the chip. Finally, the chip provides support for something known as dot correction. Dot correction is another brightness control feature that can limit the current passing through specified LEDs in case some LEDs end up being naturally brighter than others. This allows for all LEDs to be kept at the same relative brightness level, despite any manufacturing discontinuities.

In summary, this chip is capable of supporting more than enough current to power individual LEDs, and has lots of features to help with maintaining appropriate brightness in each LED. All of this and it comes in a small and inexpensive package.

3.3.4.2 STP16CP05 LED Driver

Another useful Integrated Circuit for driving the LEDs in the volumetric display is the STP16CP05. This chip behaves much like the TLC5940 described above, but lacks the PWM brightness control. The advantages of this include the ability to implement PWM manually for full control over brightness. Also, the chip only needs one bit per LED which simplifies and speeds up the communication between the controlling device for the chip.

An important thing to note about this chip, however, is the fact that it is not as common as the TLC5940 described above. This also means that the prices for these chips are slightly higher despite the fact that they are not as complex. When picking out components for use in this project, aspects like price are very important to keep an eye on; especially since the quantity of the LED drivers for this project will be fairly high. This is due to the fact that the volumetric display specifications require a large amount of LEDs to be controlled at once.

In summary, this chip is very useful for providing control for the LEDs and enough power to run them. However, because they do not directly handle brightness, the project benefited in price by utilizing the more common Integrated Circuit described earlier in the research, which handled many more aspects needed for this project.

3.3.4.3 AS7C3256A-10TCN SRAM

An important part to the design of the LED control was to keep it separate from the main processor. In order to do this, a system needed to be put in place for the main processor to interface with the LED control board. The direct way is to link up the two processors and have them communicate data back and forth. Some of the difficulties involved with this, however, include the fact that the two processors may not be running at the same clock speed. This becomes a very difficult balancing act in which one processor may need to slow down in order to send data at a rate that the other processor can comprehend. To avoid this slowdown, a better approach to interface between the two devices would be to use a shared memory buffer. When the main processor wants to send data to the LED controller, it simply writes out its data to a memory buffer where it is stored for the LED controller to read later at its own pace. This asynchronous interface design allows for the most optimal data transfer.

For a memory buffer, a simple SRAM (static random access memory) will work very well. A chip like this works in two different modes. The first mode is a data write mode. There are a certain amount of pins for providing a word of data. This number of pins depends on the chips word size but is typically one byte or 8 bits. There are then several other pins for specifying the address of where this byte should be stored. The number of address pins is equal to the logarithm, base 2, of the total number of words that can be stored. For example, a 16k word memory chip would have 14 address lines. To store a word, the main processor only has to provide the word and then set the address of where the word should be stored. The second mode of the chip is data read mode. This mode works almost exactly like data write mode, except the word pins are output instead of input. So to recall a word from memory, the LED control simply has to provide the address and then read the word.

A few things to keep in mind when choosing the appropriate memory chip is access speed, operating voltage and memory size. The memory access speed of the chip needs to be just as fast as the fastest processor in running at. If the access speed is slower, then one of the processors may have to slow itself down in order to use the memory buffer interface. Another concern is operating voltage. Most SRAM chips operate at 5V, but this may not be compatible for microcontrollers that run at a different voltage such as 3.3V. If there is a voltage difference between the two devices one of the devices could end up being damaged due to an I/O input voltage higher than what it is rated for. After some research online, the AS7C3256A-10TCN SRAM chip was selected. It has an access time of 10ns, which equates to a usage frequency of 100MHz. It’s word size is one byte and it has a total storage space of 32,000 words. This is plenty of data and met all the specifications for this project.

3.3.5 Transistors/Resistors/Capacitors

This project was of course, not able to be designed without using some of the basic building blocks found in modern circuitry. Resistors were needed to keep control over currents running to different parts of the final design. Transistors were used to route power and perform simple logic. Capacitors were used to decouple some of the ICs from each other. Decoupling refers to the issue that the power supply will most likely be slow in adjusting to provide a constant voltage. Because a lot of the IC chips often change output, and thus powering requirements, a capacitor can be placed at the power input of the chip to act as a temporary source of power while the power supply catches up to the demands.

3.3.5.1 High Current NPN Transistor

One important thing to consider when choosing transistors for switching power between the pieces of the display, is that they need support a high current load. As discussed further down in the power supply section of research, the LED display needed to be able to support at least 1.5A. This means that there will be a worst case of about 1.5A running through the transistors that route power to specified sections of the display. Transistors that a rated at this kind of current level, often have a small heat sink on them and look a little different from the standard transistor. The NTE2566 High-current Silicon NPN Transistor, as shown on Figure 3.15, can be used for this purpose.

3.3.6 Controllers

In terms of possibly using Wii controllers, communication between the user's remote and the microcontroller will need to happen, therefore both devices need to be Bluetooth compatible. The first option considered was to use a bluetooth module to add bluetooth functionality to our microcontroller. Since the Wii remote already has built-in Bluetooth, all that is left is to connect a Bluetooth chip manually to the microcontroller we chose. To accomplish this, a bluetooth module such as RN-42 HID Bluetooth module would need to be attached to the microcontroller. A viable option is the RN-42 model because it operates on 3.3V which is the voltage that the microcontroller the team chose runs on. The module can powered by simply connecting 3.3V and GND to the RN-42.

To connect the RN-42 to the microcontroller, the correct transmit and receive pins would need to be mapped to each other. During the initial development phase, the Arduino can be used to test the software functionality quickly. The Arduino's 3.3V pin needs to be connected to the RN-42's VDD pin to provide it power. Then the Arduino's GND needs to be attached to the RN-42's GND. Finally, the Arduino's RX and TX need to connect to the RN-42's UART_TX and UART_RX, respectively.

Once the RN-42 is connected to the microcontroller, the microcontroller will need to be programmed so it knows what to do with the LEDs when certain buttons are pressed on the Wii remote. For example, if the user presses ‘A’ on the remote, the LEDs could change color if that is what needs to happen. This will allow the user to have control over the game being emulated on the LED cube.

Another option was to utilize USB Hosting, allowing a USB dongle to be used for bluetooth. This option can be implemented both in the final design with the AT32 microcontroller and the Arduino Due for testing the final design. Using this method would allow the group to test the bluetooth software by taking advantage of the Arduino’s on Micro A USB port and the Arduino USBHost software. To use a USB device with a microcontroller, a USB Host must handle the connection. This functionality is built-in to the Arduino Due, thus allowing the board to appear as a USB host, enabling it to communicate with peripherals like USB mice and keyboards using a bluetooth dongle. While the USBHost library only directly supports mice and keyboards, the software can be configured to communicate with other bluetooth devices. Using the USBHost would simplify the device by allowing a bluetooth USB dongle to be directly plugged into the Micro A USB port of the Arduino. With appropriate software written for a controller like the Wii remote, the USBHost can work as the a reference for the final microcontroller bluetooth. For the microcontroller separate hardware is needed to support USB hosting.

To utilize an AVR microcontroller and use a USB bluetooth dongle, a USB host must be used. While the Arduino has USB host built-in, the microcontroller that will be used does not. The USB Host Shield Mini can serve as the USB host for the bluetooth dongle. This would perform the same function as the Arduino’s on Micro A USB port and Arduino USBHost software but instead utilizing an AVR microcontroller and AVRDude, which is a utility for programming AVR microcontrollers. This option would allow the same code made for testing in the Arduino environment to be easily used in the project’s final design. In the end, the AVR microcontroller with a USB host option was decided on since it allows for a constant connection with a bluetooth device while a bluetooth module does not host connections, only transfers bluetooth signals.

In order to control the game being shown on the LED cube, a controller that could communicate with our microcontroller is required. Using bluetooth, many types of controllers can be implemented, including mice, keyboards, or game controllers. The first option was wired controllers such as standard USB keyboard. While wired controllers would allow for an easier design, the controller choices would be limited, and the range a player can stand would be limited. Without using wireless bluetooth, the LED cube would be limited by a wire. This would not allow players to take advantage of the 3D display and move around for the best view. The best option for a controller would a wireless controller using bluetooth. By taking advantage of the USB Host Shield Mini, a controller and the AVR microcontroller can communicate with each other through the use of a bluetooth dongle. For a wireless keyboard, the amount of input would be endless, however the usability of holding a keyboard would be low. The arrow keys on the keyboard would be used for movement in the emulated game and certain keys could be mapped to game commands. The advantages of using a keyboard for control are user familiarity and better compatibility with programming software. The disadvantage, however, was that the user would need to be sitting in front of the keyboard which would limit the experience. An option that would allow the user to sit or stand to use the cube would be better.

Another option that was taken into consideration was to create a custom-built controller from scratch that would be made fully compatible with the LED cube. This option would allow for a large amount of creativity in designing our own controller, but would ultimately lead to many unnecessary issues. While creating our own controller would be a great experience and could add new possibilities to the project, the amount of time it would take to make it work is too great and valuable time would be taken away from the other components of the project.

The third option that was looked at was using a pre-built controller and taking advantage of its built-in functionality to control the cube. Many bluetooth controllers exist such as the PS3 remote, the Wii Remote, Xbox 360 remote and other generic bluetooth remotes. The Sony PS3 remote, also known as the DualShock 3 features an ergonomic game controller form. The controller features two analog sticks, a D-pad, six generic buttons, and four triggers. The controller can be connected using both a USB cable and through bluetooth. Other features include “Sixaxis” which means the sensing of rotation and translation of the remote in all three dimensional axes. The controller also has rumble and haptic feedback. A large constraint of the PS3 is the need to charge the controller through USB.

The Microsoft Xbox 360 controller features nearly the same form as the PS3 controller. The Xbox 360 controller has slightly different form factor for grip. The button layout is nearly identical, with the main difference being the placement of the D-pad. The controller is available in both a USB wired version and a radio wireless version. The main constraint of this controller is the lack of bluetooth availability.

The Nintendo Wii Remote, also known as the Wiimote, features a completely different form factor than the Xbox 360 and PS3 remotes. The Wiimote is shaped like a standard TV remote. The Wiimote remote also features a possible Nunchuk extension which plugs directly into the Wiimote and features a analog stick and two buttons. The Wiimote has no analog sticks and one D-pad. Additionally, the Wiimote has 6 generic buttons and a trigger button. Similar to the PS3 remote, the Wiimote features motion controls. The Wiimote contains an accelerometer and an Infrared sensor. Motion controls be a very cool feature to add to our input, thus a PS3 or Wiimote would be good choices. Another useful feature of the remote is the ability to use the control both vertically like a TV remote or horizontally. This will allow the project to either hard code both possibilities of control, or utilize the motion sensors to actively switch between controller modes. The Wiimote only allows for wireless using bluetooth. Unlike the PS3 controller, the Wiimote is powered using two AA batteries, not needing to be charged.

Many other generic bluetooth controllers are available, however the very popular game controllers are favored because of the large community support and documentation. The PS3 and Wiimote are very popular choices for hobbyist projects. The last option for a controller was Android. The project could be controlled using a bluetooth Android phone and custom app. This would allow endless possibilities and custom features built just for our project. Most android phones also feature accelerometers and gyroscopes for motion controllers. While Android would allow for all features to be used, most work would be required to program an app to support the project's needs. Another constraint for Android would be the requirement of the user to have an Android phone to use our LED Cube, while a traditional controller will belong to the Cube.

As the project evolved, the theme revolving around the Nintendo GameCube became clear. This led the team towards considering Nintendo GameCube controllers as the project input. GameCube controllers offer both a wired and wireless option. The controllers have six buttons, two triggers, two joysticks, and a D-pad. They are also more traditionally and user friendly in comparison to Nintendo Wii remote controllers. Additionally, the connection for using a GameCube controller is relatively simple and eliminates the need for Bluetooth.

Through research, the group determined that using a GameCube controller would be best due to fitting into the project's theme and its simple design The GameCube has the best balance and simple usability for an average user, along with plenty of buttons. Other remotes such as the PS3 remote could be supported in the future.

The final decision was to use a GameCube remote for the GameQube controller and avoid using bluetooth. This allowed for simple usability plus the option for the user to sit or stand while controlling the cube. Our goal was to utilize the built-in controller pins functionality in order to send input signals to the microcontroller which will change the LEDs on the cube.

3.3.7 Power Supply

Power was required for the LED Cube, the microcontroller board, and the sound module.

The power required for the LED cube can be calculated by the sum of the individual components, but ultimately depends on the design of the display. The chosen design for the display will require one tenth of the LEDs of the display to be on at one time. Since there are 1000 LEDs in the cube, and each LED consists of a red, blue and green LED, the total amount of LEDs to keep on at one time is three hundred. The current to run through each LED has to be carefully chosen, since the LEDs overall brightness will be reduced due to the multiplexed design. In small trial tests, a current of about 2mA multiplexed over time provides a relatively decent brightness. As an extra safety measure, this value should be doubled to give some room to increase the brightness, if so desired, later on. Assuming the worst case scenario, the power supply needs to be able to support providing current to all LEDs that could potentially be on, even though they will most likely not all be on all the time. This math then brings the total to 1.2 amps. Again, as another safety measure, this value can then be rounded up to about 1.5 amps to account for any logic circuits, ICs, or unexpected power surges. The voltage provided to the display does not matter as much, since the resistance value of the circuit can always be adjusted to achieve the same desired current. However, some ICs may require a specific voltage to run correctly without putting any stress on the chip. For example, the MSP430, discussed earlier in the research, would be very handy to use for the display and requires a voltage no greater than 3.3V. The LED drivers, discussed earlier in the research, will also adequately run on 3.3V. The only voltage requirement for the LED display then, is a voltage of at least 3.3V. If a higher voltage is input, a simple voltage divider can be used to adjust to the smaller 3.3V.

The TM4C123GH6PM and the launchpad that was used for testing both run at 3.3V and the maximum voltage that the input/output pins can tolerate is 3.3V. This is a very important detail to take into account when transferring data between each phase of the project because providing higher voltages than 3.3V to an input/output pin could damage the microcontroller. The launchpad can be powered using micro-usb or through the VIN/GND pins. The TM4C123GH6PM had to be powered from the VCC/GND pins with 3.0V - 3.6V.

Thus the final power considerations for the microcontroller included one single 3.3V line to the VDD and VDDA pins, along with GND. Other power pins include VDDA which are used for analog power, which wasn't used. An initial option considered was a PC power supply, ATX or Micro ATX. A PC power supply provides several voltage options that could be used to provide power to our project. Both ATX and Micro ATX provide more than enough voltage and current for all our needs, so a Micro ATX would be preferred to save space. The power requirements are at least 3.3V for the Cube, 3.3V for the microcontroller, and 3.3V for the GameCube controllers. An ATX power supply is made up of several different power connectors including a main ATX power connector, Molex, and SATA. The main ATX connector provides all the options needed, and because it also controls the power status of the power supply, it will be the only connector needed. Depending on the power supply, the connector can be 24 pin or 20 pin. The wires in the 24/20 pin connector include +3.3V, +5V, +12V, -5V, -12V and several ground connections, as shown in Figure 3.7 below. To use these for the project, the wires can be cut and connected directly to where needed. The PS On green wire needs to be connected to ground before any output voltage is provided. This allows the Cube to use either 3.3V or 5V independent of the other components. The Arduino has a recommended voltage of 7-12V. This means the +12V would work, however being at the maximum recommended is risky. Several options are available to combat this. To drop the 12V line, a voltage regulator or a rectifier diode can be used. Another option is instead of using the ATX ground wire, use a negative voltage wire. The possibilities include: +8.3V (+3.3V and -5V as ground), +10V (5V and -5V as ground), +15.3V (+3.3V and -12V as ground),+17V (+12V and -5V as ground), or +24V (+12V and -12V as ground). The best options to power the Arduino would either the 8.3V or the 10V configuration. The AT32UC3A0512 and bluetooth module could be simply powered by a 3.3V line.

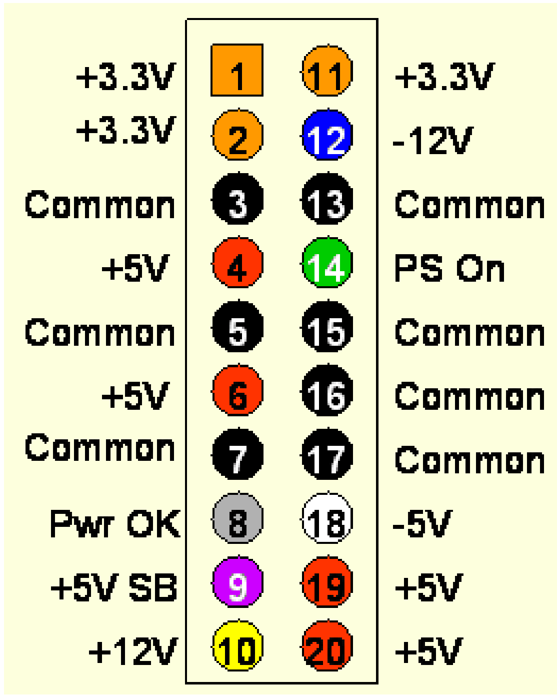


Figure 3.7: 20 Pin ATX Connector Wiring Diagram Printed with permission from Instructables[2]

The main concern with using an ATX power supply is the stability of the output. Many ATX supplies need a minimum amount of load current to regulate properly, meaning some ATX power supplies may provide more or less voltage than expected. The acquired power supply should be tested thoroughly using voltage meter to test the output during several load current values. Best case scenario is a good ATX power supply that provides stable output under all load conditions. To be safe some sort of load current should be connected at all times. Part of the housing unit of the project will be either a power LED or power LED button. A large enough LED may be able to provide a load the ATX needs, otherwise some sort of dummy load would be needed to connect to one of the 5V wires. Decoupling capacitors may also be used for stable voltage. An ATX power supply gives the option of powering the Cube with either 3.3V or 5V, the microcontroller with 3.3v, and the controllers with 3.3V from the power supply. A micro ATX can be used to take up little space in the casing, along with being conveniently plugged in straight to the wall.

Further research showed a simpler power supply would be better for the project. While an AC adapter doesn't provided several voltage lines, it is a lot smaller and convenient. Most AC adapters supply 7-12V with 500ma-2A. For the GameQube, 3.3V, 5V with at least 2A is needed. To accomplish this, the Nintendo GameCube AC adapter was considered. The official GameCube AC adapter provides 12V at 3.25A from the wall. This is better than most AC adapters because of the large current it provides, and it fits the theme well. To use this AC adapter, the voltage was be dropped to both 3.3V and 5V. A design using linear or switching regulators can be used to accomplish this.

3.4 Possible Architectures and Related Diagrams

3.4.1 8-bit Color Encoding Schemes

As described in the specifications of this project, the main processor is sending the color for each pixel in the volumetric display in a single byte of data. This 8-bit color depth allows for the encoding of 256 different colors. However, the problem that stems from this is how to represent a wide array of useful colors from just a single byte of data. One thing to keep in mind is that while the solution chosen plays a big role in the final design of the cube, it can be fairly easily modified without having to change any physical circuitry, since this is mostly a software problem.

3.4.1.1 RGB Encoding

The straight forward approach would be to assign certain bits of the 8 bit byte to the red, blue, and green component of the RGB LEDs. This is how common file types mostly encode colors, but there’s now a problem from the fact 8 bits of data does not divide up evenly between the three primary colors. One of the colors is going to need to only use 2 bits while the other two colors will use 3 bits.

Figure 3.8 shown below shows a basic color spectrum first broken down with all three primary colors getting a depth of 3 bits (top image). The bottom three color spectrums show the loss in color when one of the primary colors is reduced to a bit depth of 2 bits. Note that the middle of these spectrums, the one with blue reduced to 2-bits, most closely matches the original spectrum with an even 3 bit color depth. The reason for this illusion is based on the phenomenon that the human eye does not notice changes in shades of blue as much as it does with green and red. This is also noticeable just on the original spectrum, where jumps in the color shades going down the image seem more profound in all the areas except for blue.

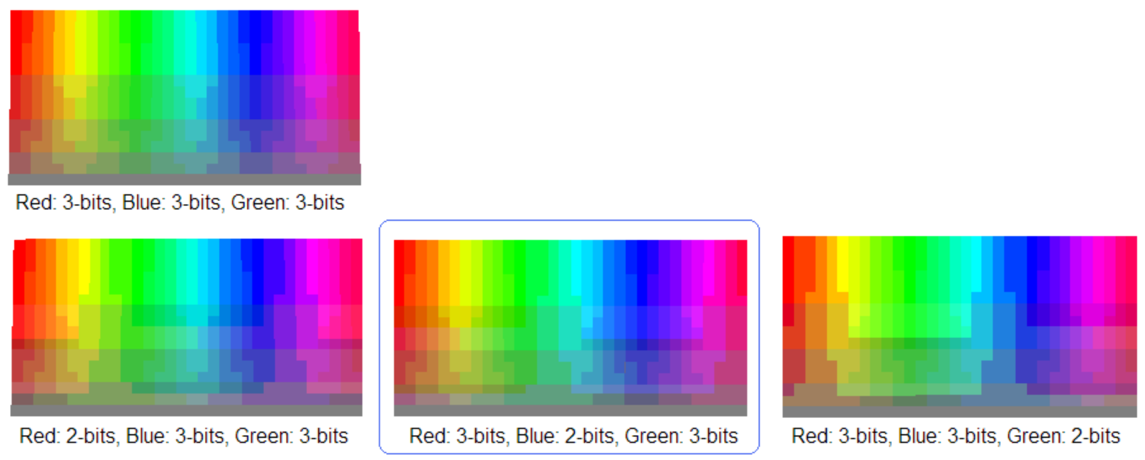

Red: 3-bits, Blue: 3-bits, Green: 3-bits
Red: 2-bits, Blue: 3-bits, Green: 3-bits
Red: 3-bits, Blue: 2-bits, Green: 3-bits
Red: 3-bits, Blue: 3-bits, Green: 2-bits

Figure 3.8: RGB Encoding Diagram

In the end, this means that the best RGB color encoding would be 2 bits blue, 3 bits red, and 3 bits green. This is exactly the same encoding process for the BMP file format when the color depth is set to only 256 colors (or one byte).

3.4.1.2 HSL and HSV Encoding

Another approach to encoding the colors for use in the display would be a hue, saturation, lightness encoding (or hue, saturation, value). Both of these are only slight variations of each other, in terms of the underlying math, but both result in a more polar representation of colors (like a wheel) as opposed to a grid like representation. The advantages to using this over a straightforward RGB encoding stems from the fact that the LEDs being used may not be able to display darker colors or washed out colors as much as the general color spectrum. Only colors that look the best on the display should be encoded in the byte, otherwise some of the bits are essentially wasted.

Figure 3.9 shown below first shows a basic encoding with 3 bits for hue, 3 bits for lightness and 2 bits for saturation. However, this direct approach has several flaws. There are a lot of repeated colors, since no matter the hue value, if saturation is turned down all the way, all the colors will appear a shade of gray. Also, a lightness value of 0 will always result in a black and a value of 1 will always result in a white. The next color encoding, moving down from the top of the figure below, shows a better range of colors by varying saturation from only 20 percent to 100 percent and lightness from only 10 percent to 90 percent. Then, to get an all-white or all black representation of color from this, only requires two colors to be hardcoded to black or white. The color encoding shown at the bottom shows how a better spectrum of colors can be gained by reducing the bits for saturation and increasing the bits for hue. These colors may be more likely to be utilized when designing the software.

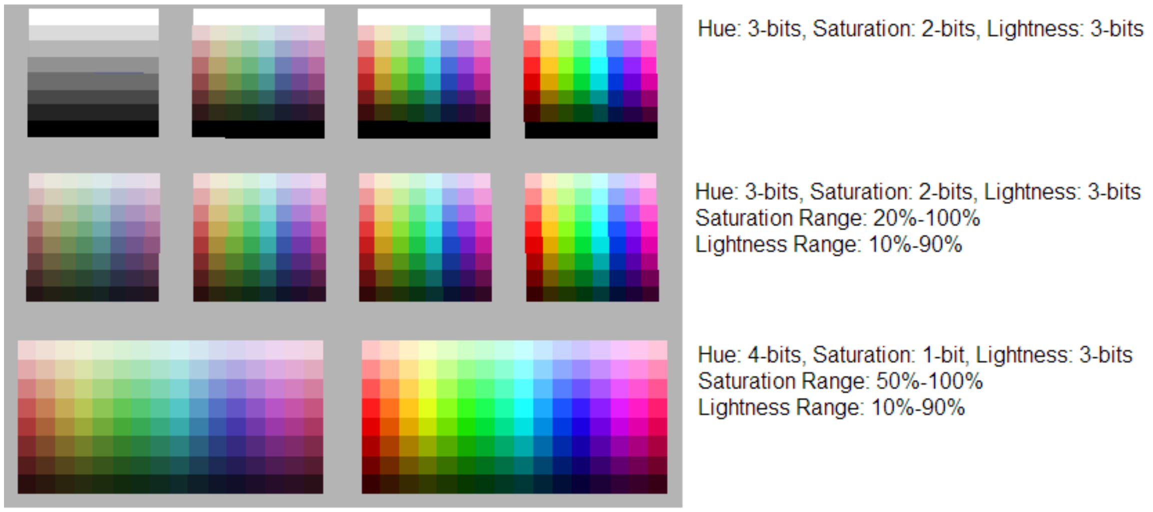

Hue: 3-bits, Saturation: 2-bits, Lightness: 3-bits
Hue: 3-bits, Saturation: 2-bits, Lightness: 3-bits
Saturation Range: 20%-100%
Lightness Range: 10%-90%
Hue: 4-bits, Saturation: 1-bit, Lightness: 3-bits
Saturation Range: 50%-100%
Lightness Range: 10%-90%

Figure 3.9: HSL Encoding Diagram

A downside to mention for this color encoding scheme, is the fact that it takes more processing to compute. In the end, each byte will have to be decoded into value for the primary colors, red, blue and green. RGB encoding means that the data is already in this format, where as HSL or HSV will require fairly advanced math to convert to RGB values. This may prove to be too costly when developing the logical control for the volumetric display.

3.4.2 Volumetric Display Construction

Of course one of the biggest decisions to be made when designing the display, was how it would be constructed. Some designs may take less time to solder together, but may also not be as structurally sound. One important thing to keep in mind for a volumetric display is the fact that there should not be too many wires in the way that might obstruct the view of the other LEDs behind them.

3.4.2.1 Common Anode per Layer

One of the more clever ways of constructing the display involves tying all the anodes of a layer together. The benefit of this design goes beyond the benefits of multiplexing. It also adds a reduction in the total amount of wires. All LEDs in a column can now share wires, instead of wires feeding to every single LED. This makes the cube appear more transparent with less to block the view of LEDs in the back. Since there are going to be 10 layers in the cube, this means the amount of wires needed is reduced by a factor of 10.

A reduction in the number of wires also allows for a much easier method of setting the LED states. Instead of addressing all one thousand LEDs individually, the hardware can simply provide only voltage to a desired layer and then ground the column the LED sits in. This allows for LEDs to be turned on from two different factors (ground, Vcc) which reduces overall LED addressing overhead by the hardware. The layers simply need to be multiplexed with their appropriate column LED states at a high speed to give the illusion that all layers are being powered at the same time. This relies on the phenomenon known as persistence of vision, discussed earlier in this paper. The construction of the common anodes is shown in Figure 3.10 below.

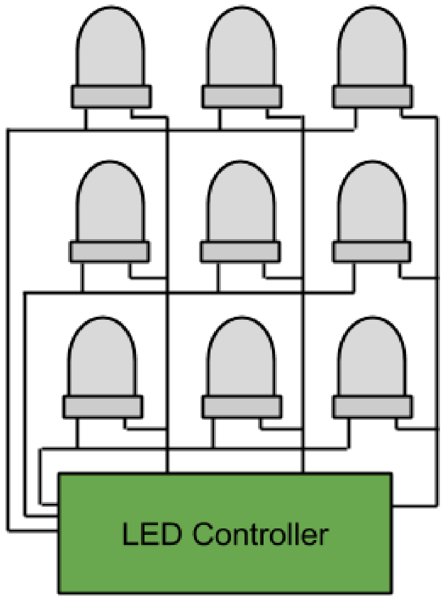


Figure 3.10: Common Anode Construction

3.4.2.2 Complete LED Addressing

The more straight forward approach to controlling the LEDs involves feeding wires up through the cube to each individual LED. This allows for complete control over all LEDs at once. The downside to this is the visibility of some of the further LEDs due to the amount of wires in the cube. This design would also require significantly more solder points, resulting in a much greater build time. A large build time is not very desirable, since most of the software cannot be fully debugged until the display is finished. This solution may not be the best due to time constraints.

Another negative of this approach is the complication of the LED control. For example, all LED states would need to be set at any given time resulting in a mass amount of LED drivers required for the display. In the case of the TLC5940 LED driver, discussed earlier, the final design would require a total of 188 ICs. This not only would result in a very massive control board to hold all these ICs, but it would bring the total cost of the project way up. The design for complete LED addressing is shown in Figure 3.11 below.

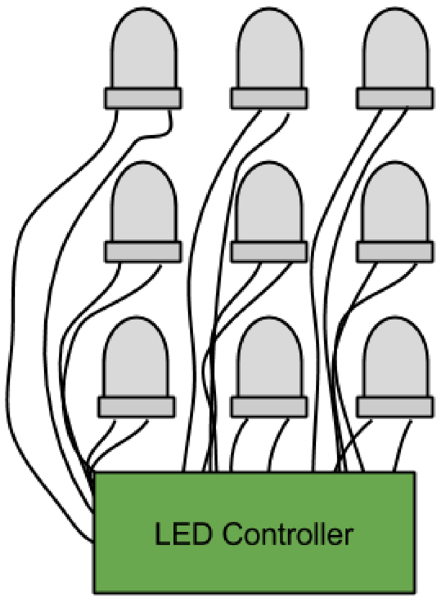


Figure 3.11: Complete LED Addressing

3.4.3 LED Control

One of the biggest tasks that needed to be accomplished for this project is how to control the volumetric display and get it to show the desired images. There are about three thousand different LEDs when factoring in RGB to control which can easily get out of hand if not controlled in a very refined manner. This brings up many different options for possible architectures in the logic control of the display, but they all at least have a few things in common.

First of all, as described above in the construction of the display, all LEDs in a layer are connected by a common anode and all LEDs in a column share common cathodes. This design means that only 300 LEDs are controlled at any one time. This reduces the necessary states to keep in the LED driving component to 300 as well. This also requires high current transistors to be controlled to route the current through the correct layer at the correct time.

As part of the specifications of this project, the LED display also needs to be completely independant from the processing in the main processor. This means that a simple communication interface needed to be created where the main processor essentially writes out a buffer of data representing what the display should look like. Because of this, the control for the LED display needed to be able to store this buffer in memory so it could be used later when multiplexing through the different layers of the cube. While this requires a little more initial hardware, in the end this design decision not only caused debugging to be easier but also allowed the development of the project to become smoother.

3.4.3.1 Shift Registers with Latch

One of the more basic ways to handle the driving of the individual LEDs would be to string together shift registers. This would allow for absolute control over the LEDs and the driving of each LEDs instead of using specific ICs. This was the method first researched and is not necessarily a bad one. Using several 74HC595 strung together would allow for control over any number of LEDs. The downside to this approach was realized when the max ratings of the 74HC595 did not meet the current requirements needed. In itself, this was not a bad problem, as simple transistors could be used to indirectly control the current flow through each LED, but the real problem shows up when prices are addressed. More components, means more money to buy the components.

Another downside to this simple approach is the lack of control over brightness. Shift registers really only allow for a simple on or off state. To get around this, a PWM method could be implemented in software. However, this would be a very difficult task to achieve. For example, a 4096 step PWM would require 300 bits to be shifted in 4096 times for one frame of brightness. Multiply this by the 10 layer multiplexing and a desired 60Hz display refresh rate, and the bits would need to be shifted in at a frequency of over 700 MHz. This is much too unreasonable and would require a microcontroller much too powerful for simple LED control. The design for shift registers with latch is shown in Figure 3.12 below.

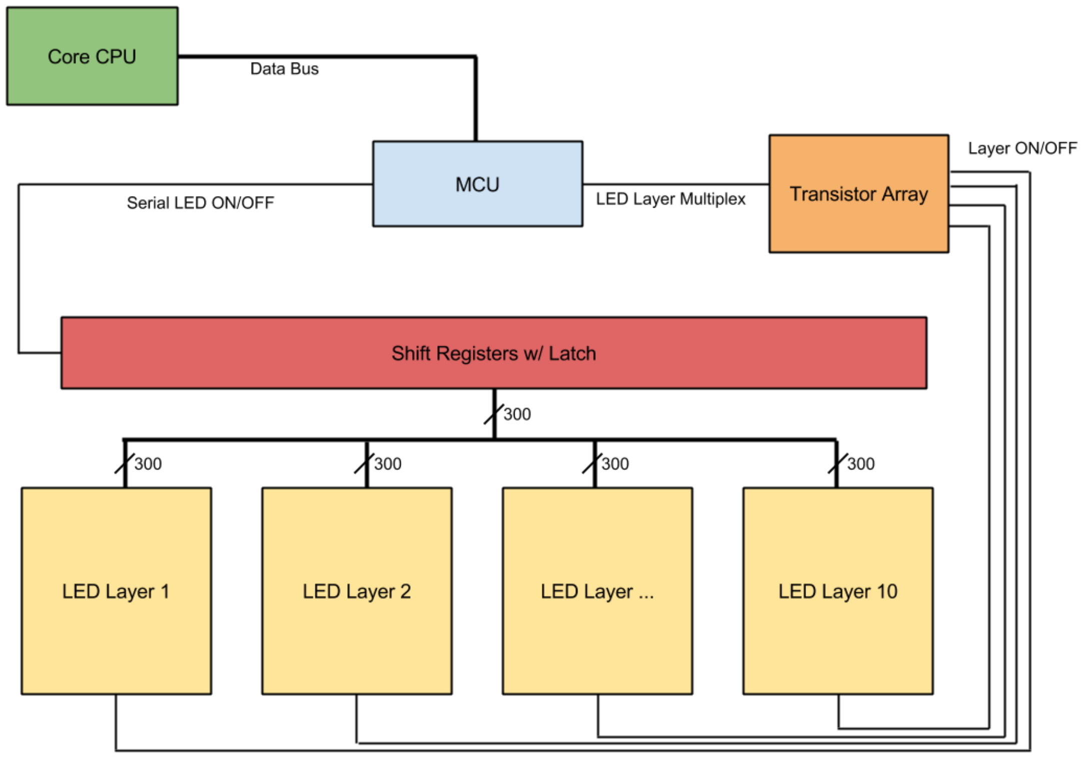


Figure 3.12: Shift Registers with Latch

When compared to the LED drivers discussed earlier in the research, it costs much less to buy the driver than the many parts of the simpler options. This is even despite the fact that the LED drivers also include more advanced features like PWM. Because of this, ICs made specifically for driving LEDs were chosen to be used in the final design instead.

3.4.3.2 LED Driver with PWM

By utilizing the TLC5940 LED driver discussed previously, the problems that arose from using shift registers can be solved much more effectively. The main benefit being a more effective control over brightness with PWM. Instead of shifting in the 300 LED states 4096 times in a brightness cycle, only 12 bits per LED are needed to be shifted in once during the same time frame. These 12 bits represents the duty cycle from 0 to 4095 and the LED driver then implements the PWM from this data. This then reduces the data shifting frequency from over 700 MHz, as calculated before, to a much more reasonable 2.16 MHz. However, to control the LED drivers themselves, additional control lines will be needed from the microcontroller. In the case of the TLC5940, this means 5 additional I/O pins are needed, but the cost is not that great compared to the benefit that is gained. The design for an LED driver with PWM is shown in Figure 3.13 below.

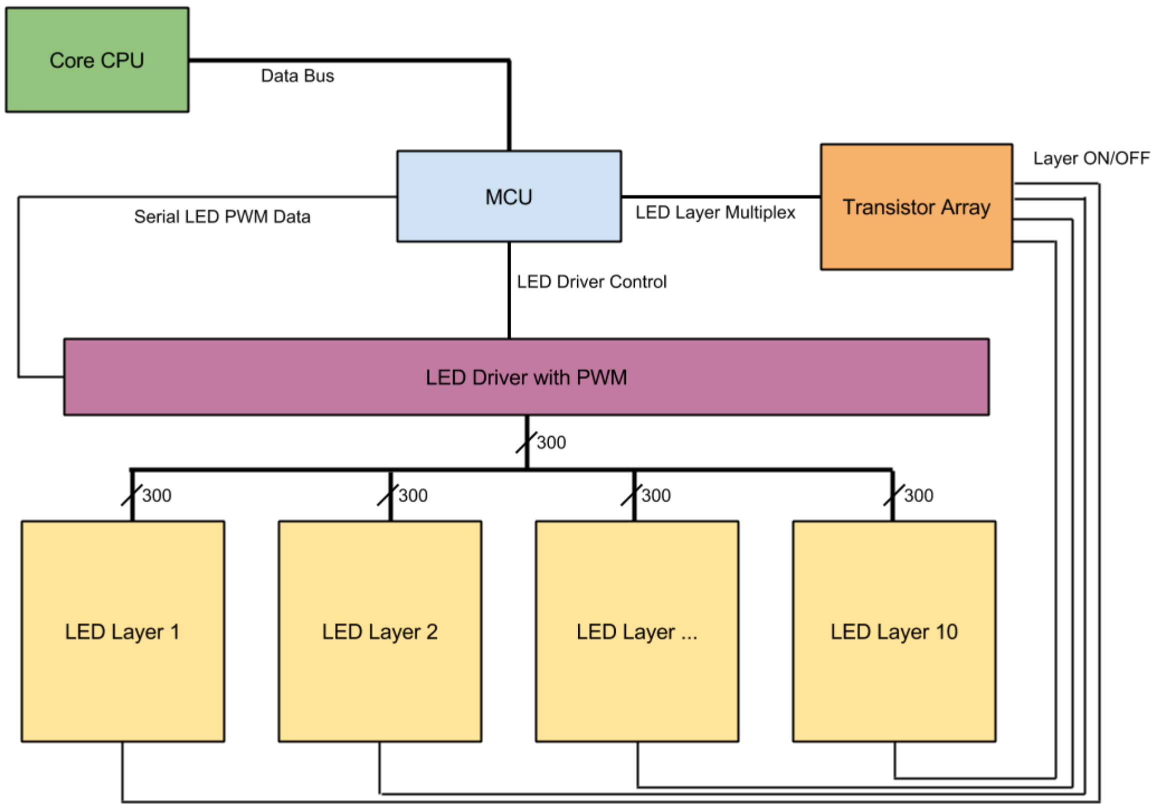


Figure 3.13: LED Driver with PWM

3.4.3.3 Shared Memory Buffer

One option for keeping track of the communicated buffer of data from the main processor and the display is to simply have the main processor interface with another microcontroller. The microcontroller would then store the data in RAM where it can recall it when it's needed. The only concerns for this approach is that the micro controller would have to stop with its LED control processes to briefly pay attention to the communication with the main processor. There is also the concern that the microcontroller may not have enough RAM to even store the entire data buffer. This may also not be the best option given the concern that the main processor may be running at a much faster clock speed, forcing it to slow down when communicating data to the display microcontroller.

Another option for allowing the main processor to communicate the display data buffer to the LED control would be a shared memory access approach. The main concept for this is that the main processor would write the data buffer to a separate memory location and then the LED controlling logic device would read the data as its needed. The benefit from this would be that the LED control and main processor would never need to directly communicate and therefore work completely independent from each other. A FIFO register could even be used in order to reduce the amount of I/O pins needed for reading and writing. The design for a shared memory buffer is shown in Figure 3.14 below.

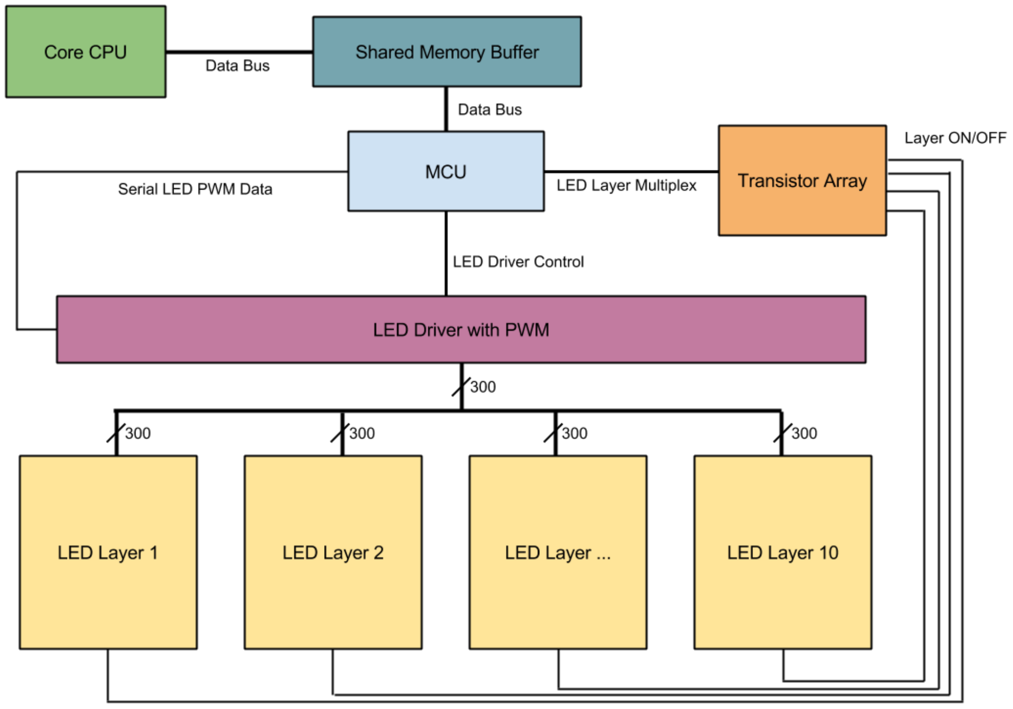


Figure 3.14: Shared Memory Buffer

One conflict that could arise from this, however, is when the LED logic tries to read the memory at the same time the main processor tries to write. Data can only be written or read at any one time, not both. To solve this problem, two memory devices would have to be used. One memory buffer would be dedicated for the LED logic to read from and the other for the main processor to write to. Once data has been completely written to one buffer, the buffers switch ownership between the main processor and LED control. So now the LED control would be reading from the newly updated data buffer provided by the main processor while the main processor writes the most recent data into the other.

3.4.3.4 RGB Color Division

One more clever approach to designing the architecture of the LED control is the concept of keeping the different colors in the RGB LEDs hardware separate. The first obvious benefit is being able to shift data to the LED drivers more efficiently. Instead of just one serial data line there would be three, allowing for three bits of data to be sent per clock cycle. This would effectively triple the maximum possible refresh rate the microcontroller can handle, since less time is wasted communicating data to the LED Drivers. Another advantage of this method is the ability to correct any sort of color errors on a hardware level. For example, if the blue diode in each LED is significantly dimmer than the other two, a simple resistor change can adjust the current running through all blue diodes, since they're already conveniently grouped together. This will come in handy if high currents are needed to achieve the desired brightness levels, since software color correction becomes less reliable and predictable with higher currents. The design for RGB color division is shown in Figure 3.15 below.

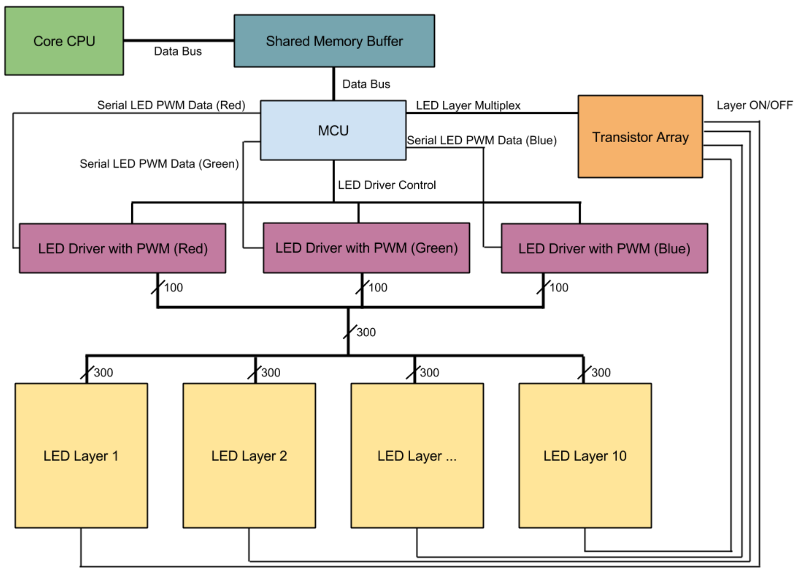


Figure 3.15: RGB Color Hardware Separation

4.0 Project Hardware and Software Design Details

4.1 Overall Design Architectures and Related Diagrams

The overall block diagram in Figure 4.1 shows the three main components of the hardware and the components of the software detailed in the following sections.

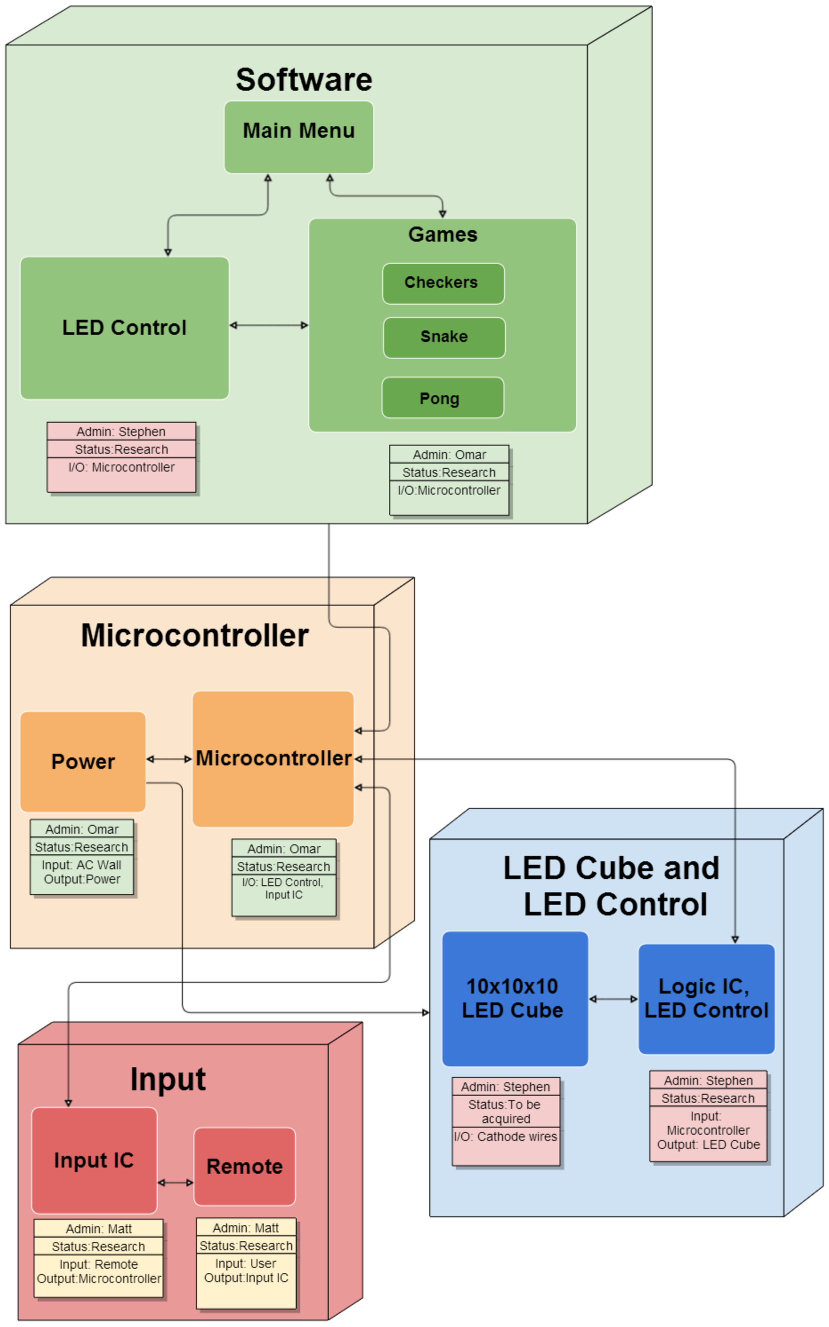


Figure 4.1: Overall Block Diagram

4.2 Input Design

The research for choosing an appropriate input interface included many possible controllers. The goal for the design was to have direct communication with a controller and the LED Cube. The final design of the input includes a Nintendo GameCube controller.

The software is setup for a standard set of inputs for the user interface and similar inputs for each game. The basic interface includes the joystick being used for 2D directions, including menu navigation. A is used for select. Start is used for pausing a runnable, where pressing Start/B will return back to the resumed runnable. Pressing A during pause will return to the main menu. To support 3D movement in games like Snake, the Joystick is used in combination with the C-stick.

Possible choices for the controller included any generic bluetooth remote, mice and keyboard, to full game controllers such as the PS3 remote or GameCube controller. The PS3 controller has a more traditional game controller shape and more buttons, while the Wiimote has a traditional TV remote shape and less buttons. Other useful features in the GameCube controller include a rumble motor. These additional features that aren't available on standard remotes such as keyboard and allows for future expansion of features. Adding rumble feedback during gameplay can add a lot of value. The GameCube controller was chosen due its convenient size and ability to be used in different ways. The GameCube controller also contains more than enough buttons to provide all the desired features, but also keeps the interface simple for any average user to pick and use. The GameCube controller allows for many different variants. The other Nintendo GameCube controllers are shown below in Figure 4.2. This includes a wired and wireless controller. They both function exactly the same where the wireless receiver handles all wireless communications. The only difference is the wired controller supports the rumble feature, which was implemented in the software to indicate death during a game. Other variants include wired and wireless options from third-party manufactures such as Mad Catz. After testing, it was concluded that all four controller variants worked on the GameQube.

To connect this controllers, the hub from a broken Nintendo GameCube was used for four GameCube controller sockets. This allowed for four wired or wireless controllers to be plugged in and power from the single hub. The hub provides connections for the power and data lines of each controller.

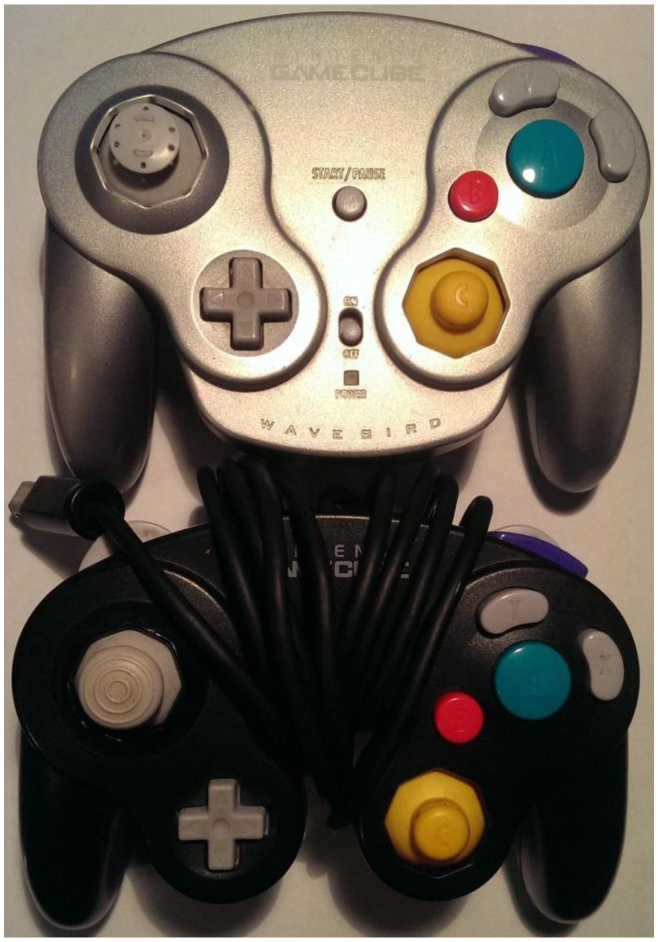


Figure 4.2: GameCube Input Buttons

For development, the Tiva C not only allowed for convenient programming through a dedicated Micro USB programming port, but it also contains a native USB port for any additional controller testing. To interface with the GameCube controller, a lot of care needed to be taken into the timing of the data communication signals. The GameCube controller uses a single bi-directional data line. This means that the main processor needed to be able to send a perfectly timed message as well as be able to interpret a perfectly timed response. To send a 0, the microcontroller must pull the data line low for 3 microseconds and then pull it high for 1 microsecond. To send a 1, the microcontroller must pull the data line low for 1 microsecond and then pull it high for 3 microseconds. In the reverse direction the microcontroller must be able to detect how long the response signal is high and low and interpret it as the responded data. To communicate with the GameCube controllers an initialization signal must first be sent and then a button data signal must be sent anytime the main microcontroller wants to know the current button states. It’s this polling structure that the communication will occur in. Figure 4.3 shows the timing precision that the data line requires.

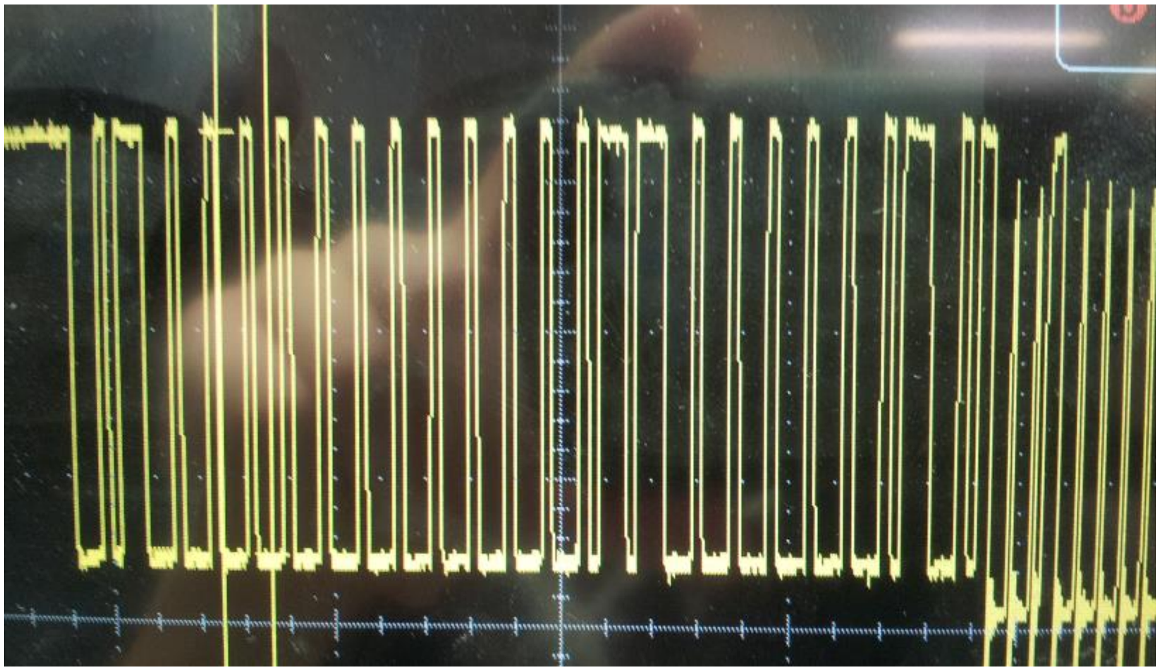


Figure 4.3: GameCube Data Line

The initial design of the input allowed for several input choices. Having several choices allowed the design to be flexible for cases in which certain controls or other peripheral devices failed to perform correctly. Having a wide variety of GameCube controllers to choose from helped with this, having both wired and wireless options. The actual connection of the controller has six main pins, but only five are actually used. As mentioned earlier there is only one data line and the remaining pins are for power and ground. Figure 4.4 shows the pinout for a GameCube controller.

Pin Function	Pin Use
Data	Bi-Directional Data Line
Gnd	Voltage Ground
5 Volts	Voltage for Rumble Motor
3.3 Volts	Voltage for Logic Circuits
Gnd	Voltage Ground

Figure 4.4: Relevant GameCube Controller Pin Mapping

4.3 LED Cube Design

The LED Cube is composed of 5mm RGB LEDs arranged in a common anode per layer design. There is a total of 300 leads, which are routed from the cube to a PCB board via ribbon cables. The PCB board contains all the logic and power for the display, including the LED drivers. The microcontroller used in the LED logic is the MSP430. A single 10 pin connector is also on the PCB board for the main processor to interface and write to the display. The final array of LEDs is encased in acrylic and stands on top of the base of the final project.

4.3.1 LED Assembly

The assembly of the cube involved a three step process similar to the same process described by Kevin Darrah in his 8x8x8 cube. The first step is to solder the RGB LEDs into columns of 10 LEDs. A column consists of LEDs soldered together by their three cathodes. This makes all LEDs in a column share a common cathode for both the red, green and blue diode. A simple rig, made out of wood, will be constructed to hold the LEDs and wire in place while they are soldered together. The rig is constructed perfectly so that there is exactly 10 inches between the first and last LED, shown in Figure 4.5 below.

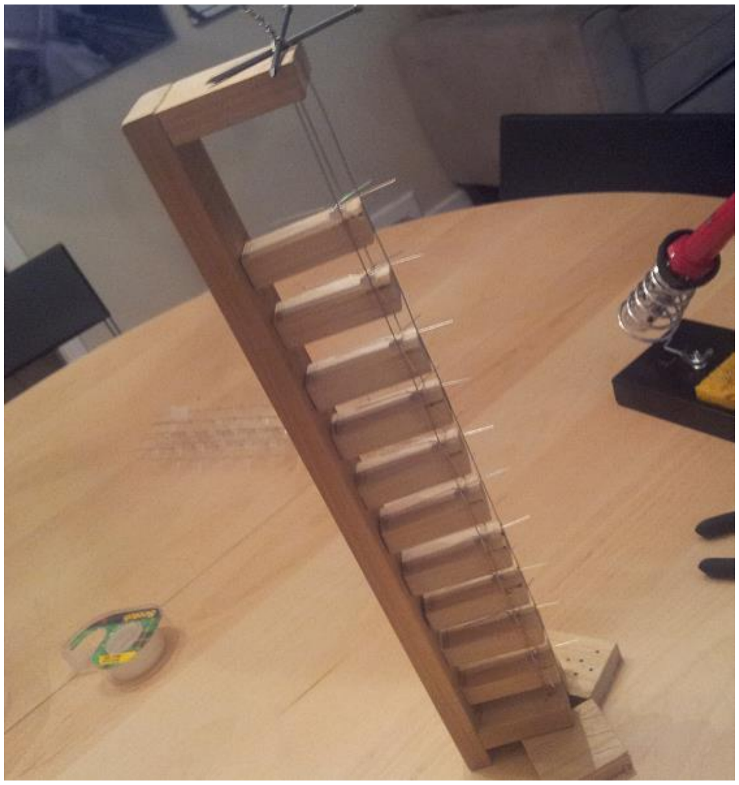


Figure 4.5: Column Soldering Rig

The second step in the LED cube assembly involves soldering the columns into sheets or slices. This is done by soldering all anodes in a layer together along a piece of wire. A separate rig will be constructed to hold the wire straight and provide markings to ensure spacing between the columns is precise, shown in Figure 4.6 below.

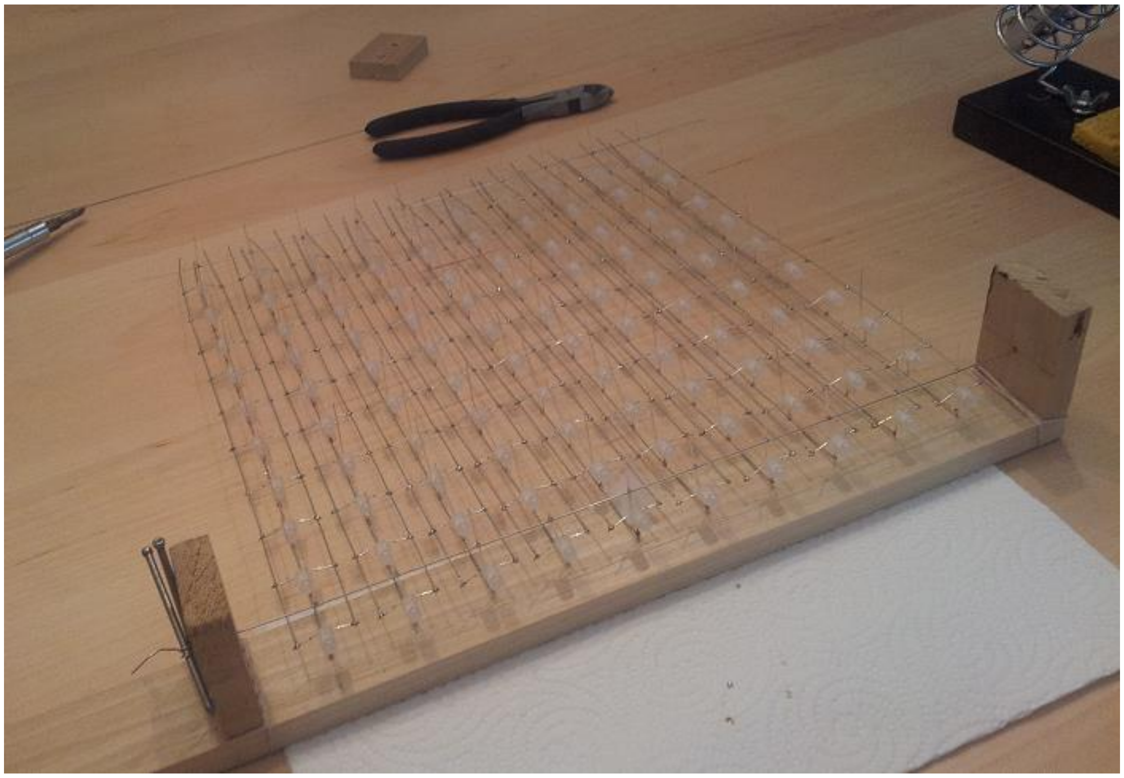


Figure 4.6: Sheet Soldering Rig

The final step in the assembly of the LED cube involved soldering the sheets together into the final cube. To accomplish this, a solid wood base had holes drilled into at precisely measured positions. This acted as a sort of rig and ensured that the sheets remained straight when being soldered together. The cathodes of the LED columns fit through the holes to hold the sheets in place. Each sheet was then soldered together with a single wire along the side for each layer. This ensured that all anodes in a layer were tied together. Once all the sheets were soldered together into the final cube, the cathode and anode leads were tied to ribbon cables, which allowed for easy hook-up to a control board with sockets for the cables. The assembled cube is shown in Figure 4.7 below.

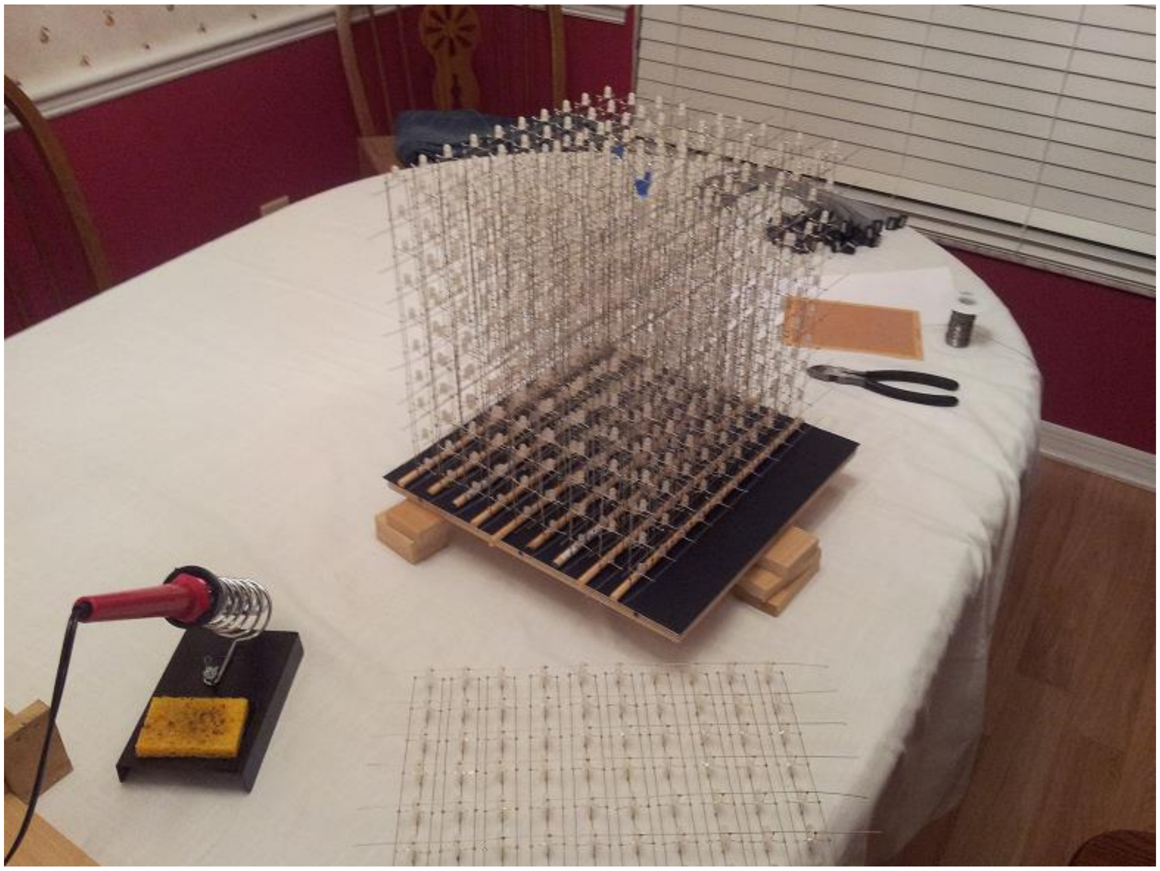


Figure 4.7: Final Cube Assembly

4.3.2 LED Control

To control the LEDs and interface with the main processor, LED drivers are used to control the brightness of each LED. In particular, the TLC5940 is used to provide PWM and effectively help to produce a wide array of different color combinations. The TLC5940 requires at least five lines to control it. The first two are SCLK and SIN, which is used to send serial data to the chip for grayscale levels. XLAT is then used to latch the input data to the chip once it has all been sent. Finally, GSCLK and BLANK are then used to first tick the PWM steps and then refresh the PWM cycle.

To help organize the amount of LED driver chips, the LEDs are divided into groups according to their respective colors (red, green, blue). This not only helps with organization, but it will also help to perform any sort of color correction at a hardware level. There are three LED driver sections each with 7 TLC5940s, 10 sockets for ribbon cables, with 10 pins connectors, and 12 miscellaneous pins which could be used to drive additional LEDs. This also means that each LED sheet has three ribbon cable (one for each color). The LED driver color section is shown in Figure 4.8 below.

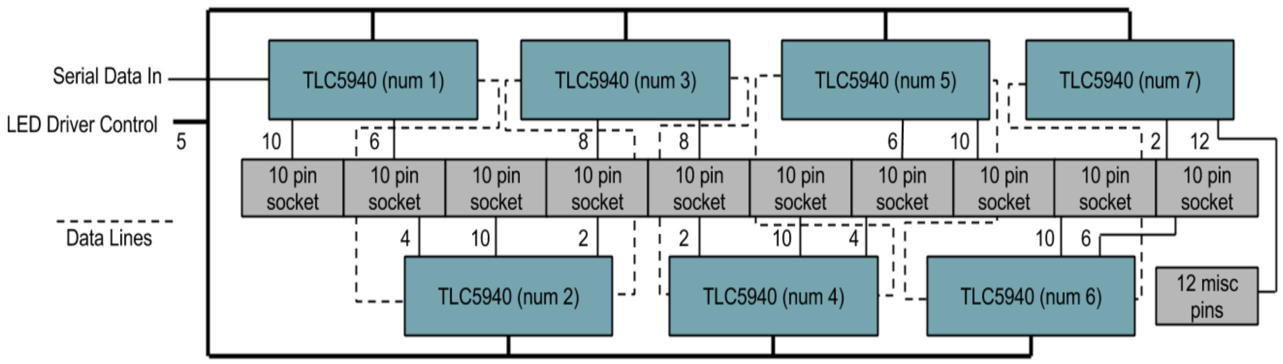


Figure 4.8: LED Driver Color Section

Since the LEDs are not all going to be on at the same time, a microcontroller was needed to handle the multiplexing of the individual layers. To do this, the MSP430F5529 was implemented in the design since it has a clock frequency of 25MHz, closely matching the TLC5940 max serial data transfer speed of 30MHz. This microcontroller was also selected due to its high amount of I/O pins. To select the columns to light, the microcontroller interfaces with the three color LED driver sections. To select a layer, the microcontroller interfaces with an array of transistors which selectively provide power to only one layer at a time. Finally, in order to receive video data from the main processor, a separate shared memory buffer is used. This allows an asynchronous data transfer to occur between the two devices even if the main processor is running at speeds much faster than the LED control microcontroller. The LED control design is shown in Figure 4.9 below.

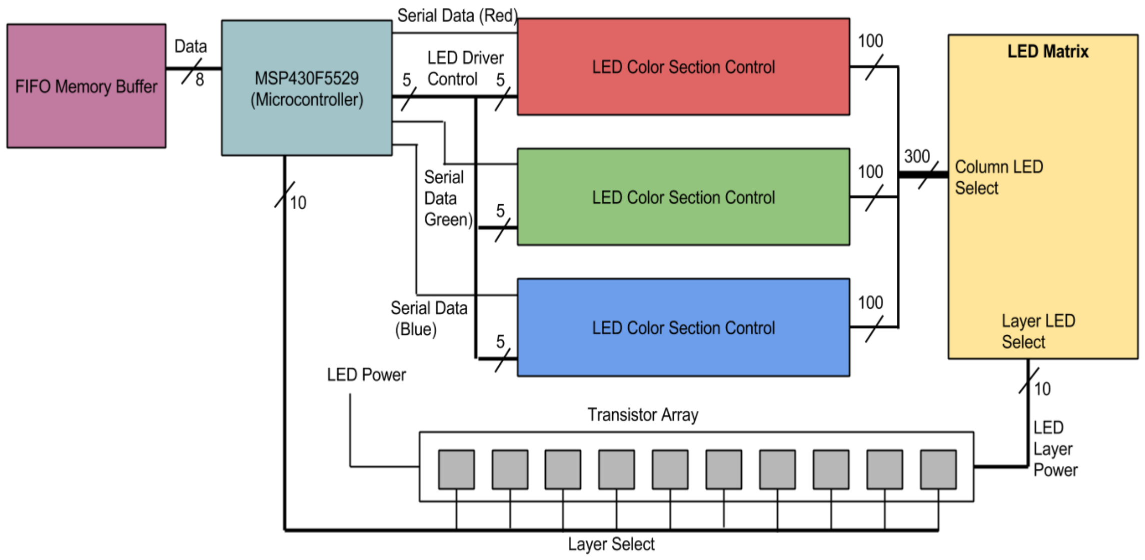


Figure 4.9: LED Control

The video data stored in the external FIFO memory chip is stored in the format of one byte per LED to denote color. The reason for this is to improve the amount of time it takes for the main processor to output a frame. The separate modular approach for the LED control will also help improve video output times for the main processor. For example, if the main processor was to control the TLC5940 LED drivers directly, it would take a considerably more amount of time to output all desired LED states. All LED data would need to be shifted in with 12 bits per color. That's 26 bits per LED for one thousand LEDs. If only one bit can be shifted in at a time it would take 26000 clock cycles. Encoding each LED into a byte and allowing a byte to be output in a single clock cycle, it only takes 1000 clock cycles to output video data. This speeds up the data transfer by a factor of 26.

4.3.3 LED Display Interface

In order to control the LED display, an interface needed to be designed to allow for the main processor to send its video buffer data to the display so that it could be represented correctly. To do this, a 10 pin interface was designed to facilitate the communication process. The interface, illustrated in Figure 4.10 below, includes 8 data pins, a clock pin and a sync pin. The 8 data pins allow for data to be transferred a byte at a time. The reason this particular number was chosen was because it is a convenient number for microcontrollers to output and each LEDs color is represented in a byte of data. So for the main microcontroller to send it's video buffer data, it sets the data pins one byte at a time and ticks the clock pin to transfer the byte once it is set. The tenth pin, the sync pin, exists for several reasons. One reason is to let the main processor know when the display driver is finished processing the last set of data. This helps prevent the main microcontroller from tripping over the slower display processor since it's running at a much higher speed. Secondly, the sync pin acts as a clock for the main microcontroller to try and maintain a constant 60Hz frame rate. This means that the main microcontroller can actually use the display to keep time rather than needing a separate external clock.

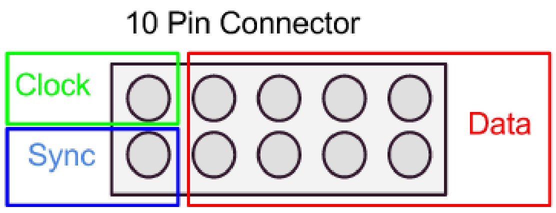


Figure 4.10: Display Interface

The LED control software is interrupt based running on a timer inside the MSP430. Using this timer the cube cycles through its entire multiplexing cycle at a rate of 300Hz. This speed is nice for the persistence of vision illusion, but is complete overkill for simply updating the video frame. Because of this, the interface is actually designed around the set refresh rate of 60Hz. This is still overkill for refresh rates, but it allows for the display processor to be used more efficiently and save power. However, because the display is still refreshing at a rate of 300Hz, it has a much higher capability to poll if new video data is available. This means that beyond the standard 60Hz target, if the main processor cannot render at that speed, the display will poll at a rate of 300Hz and is therefore capable of supporting additional refresh rates. Typically if it was running at a refresh rate of 60Hz the next lowest supported refresh rate would be 30Hz or half the targeted value. With the higher speed polling design the display interface is capable of supporting frame rates of 60Hz, 50.8Hz, 43.6Hz, 38Hz, 33.9Hz, 30Hz, etc. Figure 4.11 illustrates the timings that make this possible.

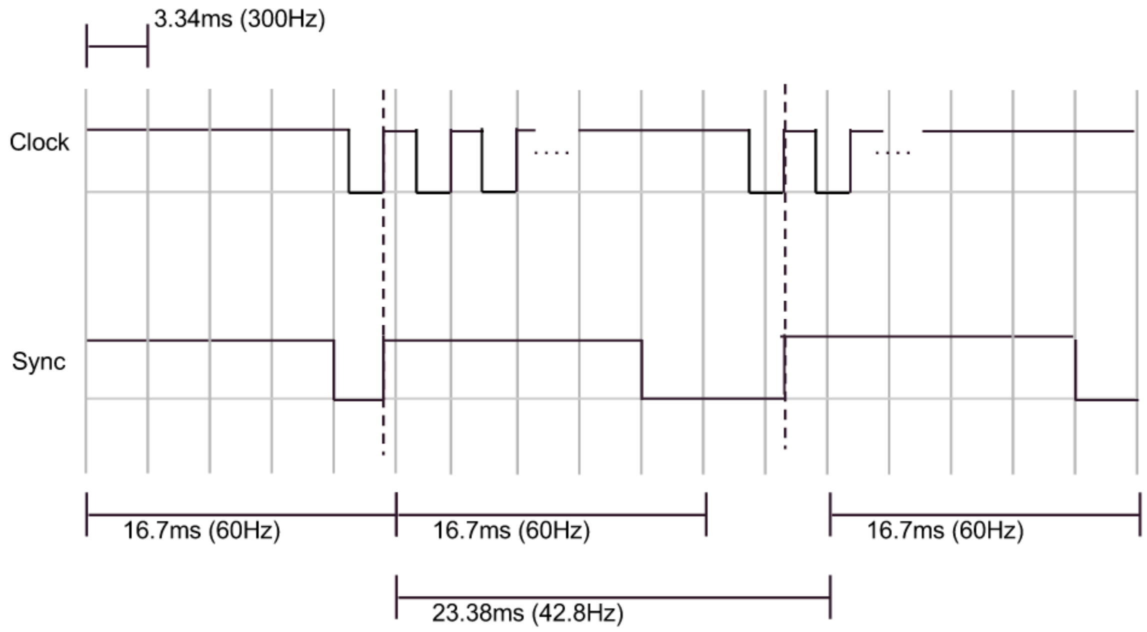


Figure 4.11: Interface Timing Diagram

4.4 Microcontroller Design

The microcontroller board is based on the TI Tiva Launchpad. The microcontroller being used is the TM4C123GH6PM. The Tiva TM4C has a 32-bit ARM Cortex-M4 80 Mhz processor and 256 KB flash memory. The package size is 64-pin LQFP and supports up to 43 GPIOs. To use this chip on a custom board, the design for the decoupling capacitors, clock crystal, and IO interfaces must be made. The Tiva supports USB, JTAG, UART, and I2C communication. Programming will be done using the JTAG interface and the launchpad. To do this TCK, TMS, TDO, and TDI JTAG pins must be connected from the microcontroller to the launchpad. Additionally, a common ground and reset must be connected.

The first part of designing the microcontroller board includes the power. The Tiva has four VDD lines and two VDDC lines. The microcontroller requires a single

+3.3V line to VDD and VDDA. VDD powers the I/O and some logic, VDDA powers the analog circuits, and VDDC powers most logic. VBAT is used for hibernation mode, which won't be used, but the pin will be pulled high as this follows the preferred practice. Other unused connections for hibernation mode such as HIB,WAKE, XOSCO0, and XOSCO1 are connected as recommended for unused connections. VDDC does not require input voltage, instead the two lines must connected together with LDO filter capacitors. Four capacitors are used that total 3.4 uF. The VDD lines require decoupling capacitors near each VDD pin. Two 0.1 uF and two 0.01 uF capacitors are connected between each pair of VDD and GND pins. Reset is pulled high with a jumper to ground allowing for manual resetting. The pin is also routed for programming. The main crystal is connected through OSC0 and OSC1 using two 10 pF capacitors. The rest of design includes using the GPIO for the features of the GameQube. Ten pins are used for sending data in sync to the LED cube. Nine pins are used for sound and four pins are used for supporting four controllers. The pin mappings for TM4C can be seen below in Figure 4.12. With all this, the Tiva chip still has more than 10 pins leftover for additional IO. The design is simple and takes up little space, proving the TM4C to be a good choice. Added on to this board is the design to support sound with directly connects to to this chip, discussed in detail in the next section.

Pin Function	Pin Use	Pin label (TM4C)
VDDA/GNDA	Analog voltage	2,3
VDD/GND	Input voltage	11,12; 26,27; 39,42; 54,55
VDDC	Logic voltage	25,56
RST,OSC0-1	Reset, Clock	38,40,41
JTAG	Programing	49,50,51,52
PA0-PA7	Display Data	17-24
PB6-PB7	Display Sync	1,4
PE2-PE4,PE0-PE1	Sound Data	6,7,59,60; 8,9
PF1-PF3	Sound Sync	29,30,31
PD0-PD3	Controllers	61-64

Figure 4.12: Relevant TM4C123GH6PM Pin Mappings

4.4.1 Sound Design

Sound design consists of 4 main components, the ADC, sound processor, SD card storage and stereo audio input. The analog to digital converter allows for the sound to be heard by the main processor to perform visualizations. The sound processor mixes the audio samples and converts the digital samples into an analog signal. The SD acts as external ROM storage for large music samples and the stereo audio input helps mix user audio with the sound the system generates.

The sound processor handles communication with the SD card for collecting music samples. The sound effect samples are stored in the extra ROM space on the microcontroller as a quick resource to pull from. However, because there is not a lot of extra ROM the sound effects are only sampled at a rate of 16kHz as opposed to the music on the SD card which is sampled at a rate of 32kHz. Since sound effects are not long in duration this convenient little storage medium was possible to utilize. To create the final audio signal the microcontroller digitally mixes the sound effect sample and music sample into one value. This value is then output using PWM to create an analog signal. The samples being mixed on the microcontroller are only for a signal channel, but it is output on two different PWM outputs in order to keep an electrical distinction between the left and right channel of the users input audio which gets mixed in later.

The SD card is configured on a PC as a simple linear chain of all the music files stitched together. There is no file system utilized on the card. The microcontroller simply has the start and end addresses of all music options hardcoded in its program and addresses data on the SD card accordingly. The SD card is easily able to interface with the microcontroller thanks to its 3.3V logic level and easy to use SPI interface.

After the analog sound signal is created by the sound processor, it gets mixed with the users input audio through some simple circuitry. Typical input voltage ranges are very small so the input is first put through an op amp to amplify the signal. A rail-to-rail op amp was used in order to ensure that no higher voltage than the tolerance level of the main processor was produced. Through a series of resistors both the left and the right channels get mixed together to produce the final audio signal that is fed to the speakers. Before the signal is sent to the speakers, however, is is routed to the analog to digital converters of the main processor. This allows the main processor to be able to record and observe the signal for use in applications like the music visualizer.

Finally, the main processor and the sound processor communicate in only one direction from the main to the sound processor. The main processor sets up an operation code on a 5 pin interface and then sends the command through a separate clock line to the sound processor. The sound processor then works on an interrupt based system where when its clock line is pulsed, the sound mixing is interrupted to process the requested operation. The operation code system designed allows for up to 16 music selections to be made, 8 sound effect selections and support for 8 additional commands like stop and pause. The final microcontroller design can be seen below in Figure 4.13 which includes the Tiva and MSP430 for sound.

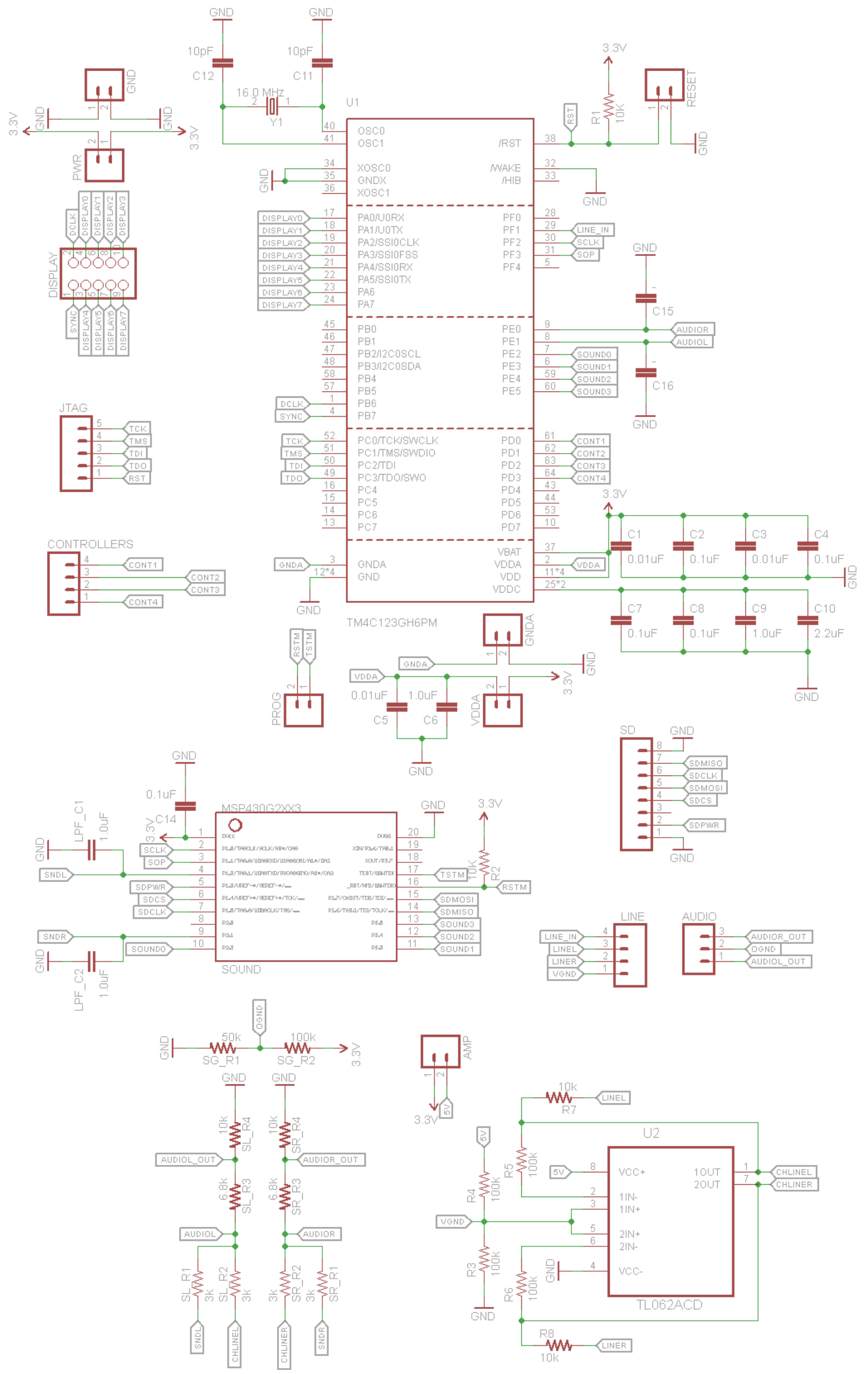


Figure 4.13: Microcontroller board schematic

4.5 Power Design

One of the main goals of the project was to make a modular design in which the three main hardware components would be as independent as possible. The three main components include the LED logic circuit, the main microcontroller, and the controller interface. The main goal for the power supply was the ability to plug the final product straight into the wall. To complete both of these goals, each component needs to be powered separately. The best way to achieve is to use official GameCube power supply. The GameCube AC adapter provides 12V DC at 3.25A with input directly from the wall. The adapter is a standard wall adapter with a 3-prong plugin to the wall and a brick that hosts the power supply. The connection is a special connector used for the GameCube. To use this connection, the socket was taken from a broken Nintendo GameCube and fused into the back of the GameQube. This allows for easy connecting, while the socket routes the voltage and ground lines to the electronics inside the housing. This power supply provides more than enough voltage, in which no components in the GameQube consume more than 5V. While a lower voltage power supply would be easier to step down and dissipate heat, the GameCube AC adapter was attractive due to the current output. With 3.25A, this allows plenty of freedom when lighting all the LEDs at varying brightness.

The MSP430 and Tiva microcontrollers run at 3.3V as well do the GameQube. The GameCube controllers take in 3.3V and 5V. The power requirements for the GameQube are shown in Figure 4.14. To allow use of both 3.3V and 5V from the 12V power supply, a DC-DC converter is used to step-down the voltage. The converter uses a LM25965 switching regulator and a PJ1084 low dropout voltage regulator to provide both 3V and 5V at 3A from a single 12V input. The 5V line goes solely to the GameCube controller hub. The GameCube controller hub consists of four sockets for four GameCube controllers. This hub was taken off a broken Nintendo GameCube and fused into the center of the GameQube housing. This provides an professional look that reminds users of the GameCube theme. This hub allows for a single 3.3V and a single 5V with ground to power all four controllers. The 3.3V line of the regulator also goes to the main microcontroller board which provides power to the LED circuit.

Component	**Power Requirements**
LED Circuit	3.3 V, 1-2 A
TM4C123GH6PM	3.3 V
GameCube Controller	3.3 V, 5 V (for rumble)

Figure 4.14: Power Requirements

4.6 Housing Design

The housing has two main parts: the electronics housing and LED housing. The electronics housing is the base of the project which holds the electronics inside and holds the LEDs above. The inside of the electronics housing can be seen below in Figure 4.15. All electronics including the LED circuit, microcontroller, power supply, power button, speakers, and controller hub are inside**.** The base is 14” x 14” x 5”.

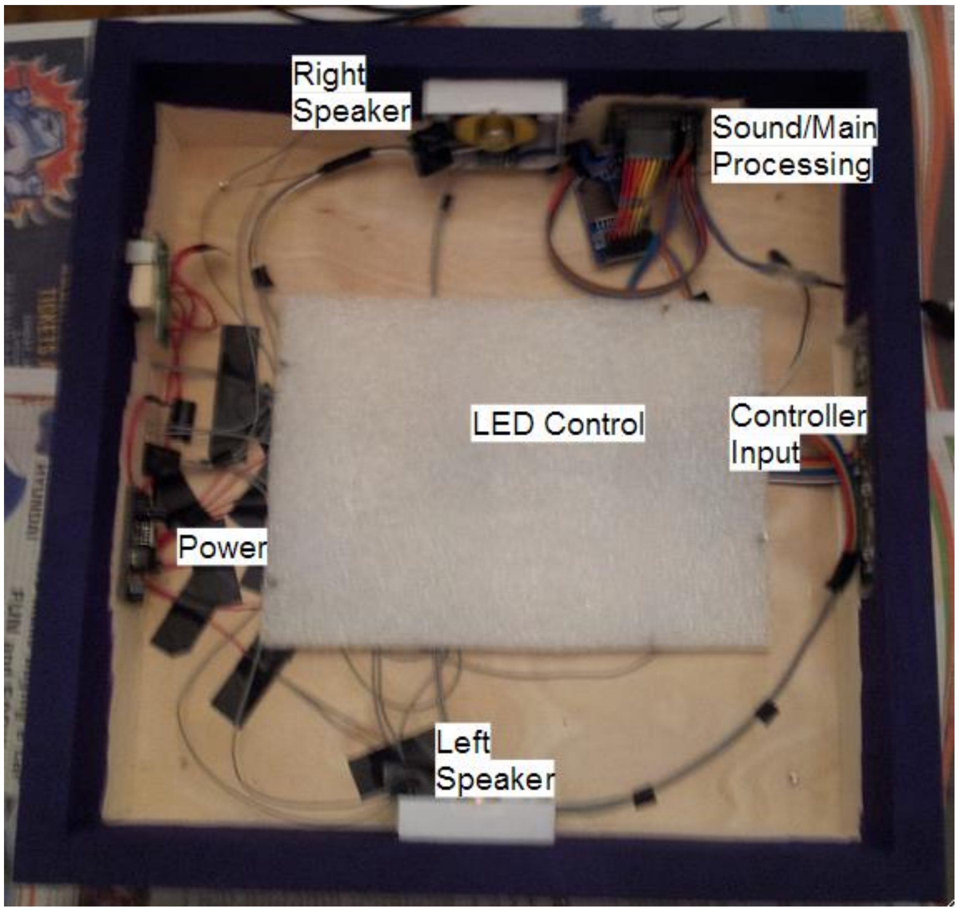


Figure 4.15: Electronics Housing Diagram

The base is a rectangular prism made with wood. The four sides and the bottom are completely solid and fixed. The top contains cutouts to wire the LEDs. To access the electronics for programming and modifications, the LED cube must be tilted to remove the top side. The LED cube cables then must be unplugged, and the top can then be removed allowing free access to the electronics. The front side contains a cutout for the controller input. The back side contains cutouts for the power input and the power button. The other two sides have holes for the speakers. The LED controller circuit is located in the middle while the main microcontroller is mounted on the side. The wood outside is painted all purple to match the original Nintendo GameCube purple theme.

The second component of the housing is the LED encasing, shown below in Figure 4.16. To protect the LEDs, five acrylic sheets will be used. Two acrylic sheets will be 12' x 10', two will be 10' x10', and the final top sheet will be 12' x12'. Different acrylic widths are being considered for optimal view, protection, and ease to cut. Most likely the width will 5mm or less. To cut the acrylic many options are available including sharp cutting blades, handsaws, or sawblades. After cutting the edges must be smoothed and chemically attached to each other, then embedded into the top of the base. Together, both parts should offer the project good protection and good presentation.

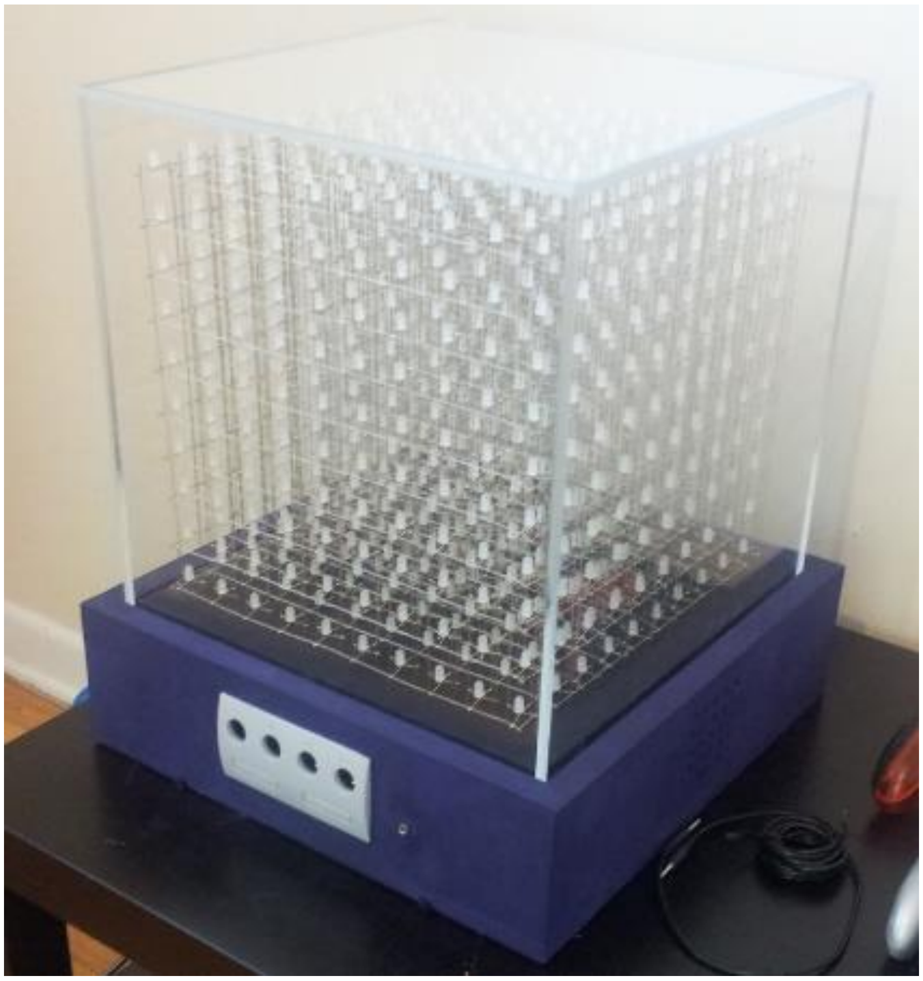


Figure 4.16: Electronics Housing Diagram

4.7 Software Design

The software was compiled and sent to the microcontroller using Code Composer Studio in C/C++. The overall software architecture shown in Figure 4.18 below is divided into three blocks which will be detailed in the following sections.

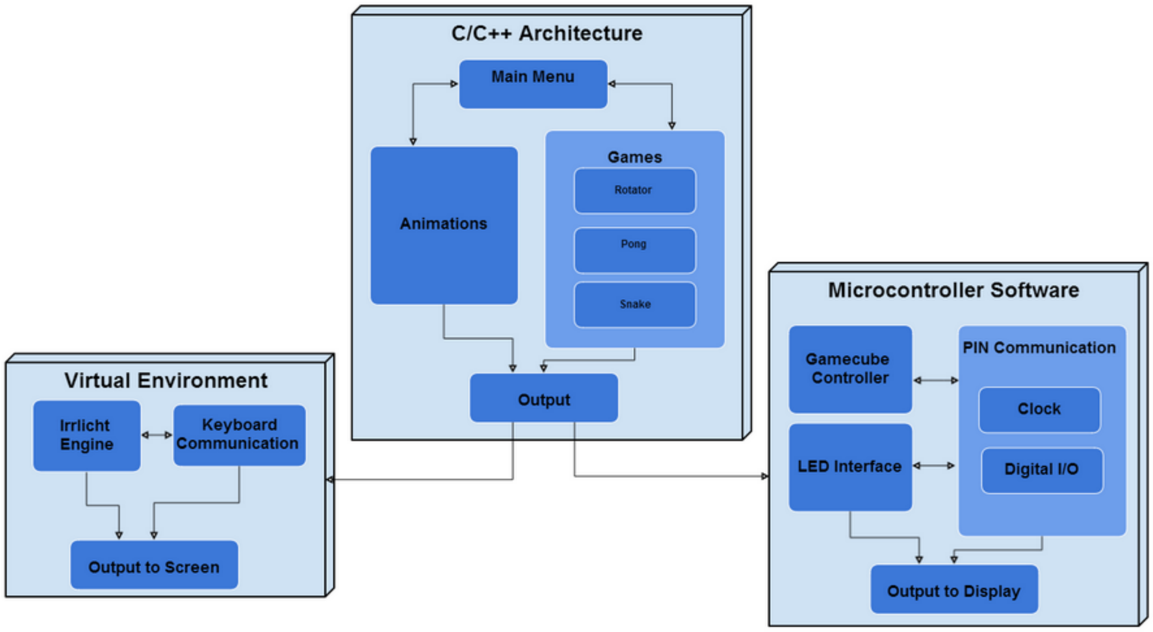


Figure 4.17: Software Architecture Block Diagram

For testing with the development boards, the software was configured for Code Composer Studio in C/C++. Initial design and implementation of the code was developed and ran in Microsoft Visual C++ 2010. The reasoning behind this was due to the virtual environment the group used as simulation is implemented in Visual C++. Code Composer Studio is also very similar to the Visual C++ IDE. With the exceptions of certain unsupported functions the code was completely portable from the virtual environment in Visual C++ to the LED Cube in Code Composer Studio.

The basic architecture of the software design was creating a 1000 byte output with the color for each LED. This was all implemented in common C/C++ code that created an array of data. Using this, the virtual environment software built the simulated cube and displayed the data. The embedded code did the same thing, however instead of initializing the virtual cube, the hardware pins/output, clock and interrupt cycles, and controller input must be managed. The main microcontroller must also implement the low level software.

4.7.1 Virtual Environment

As part of the theme of modularity, a virtual representation of the the LED Cube was needed. A virtual cube would allow the software to be developed immediately with an accurate simulation environment. As long as the final output of the code is the same, the virtual environment and the physical LED cube should function the same if both built correctly. The virtual environment allowed for immediate testing and compiling of the code which allowed the software phase of the project to evolve immediately. To create the virtual environment the team utilized the Irrlicht Engine. The Irrlicht Engine is a cross-platform high performance realtime 3D engine written in C++ in the forms of downloadable libraries. It features a powerful high level API for creating complete 3D and 2D applications, for our purpose a 3D LED cube.

Utilizing Visual Studio IDE for using the Irrlicht Engine is very simply implemented. The basics of the engine include the VideoDriver, the GUIEnvironment, and the SceneManager. The engine is available are free to download SDK. To use the engine the code must include the header file <irrlicht.h>. With this header th group was able to build the Virtual Environment.

The main goal of the software was to provide a fun and interactive experience for a user. Being a three-dimensional display, game and animations that utilize the full space were favored. Before 3D graphics, classic games were limited to what could be simulated on a 2D screen. The GameQube allows for those games that work with limited resolution to evolve into a 3D world.

The software architecture, referred to as the Virtual Environment, is an object-oriented C++ runtime. The software runs on a loop running different applications, called runnables in the software. Each runnable refers to a different application a user can select from the main menu. These include games such as Snake and Pong, Animations to watch, and a Music Visualizer. Each runnable extends IRunnable which sets up the class to render the scene using the Virtual Environments SceneManager and VideoDriver.

The SceneManager is the container class responsible for hosting all the nodes that can rendered on the cube. A node is predefined object with certain behavior that is drawn to the cube. Some common nodes used include CubeSceneNode, TextSceneNode, and LineSceneNode. Each of these nodes keeps track of size, color, and other attributes of the object and defines how to draw the node. The SceneManager is responsible for adding and removing each node to the scene. Every frame the SceneManger is called to render all nodes.

Rendering is handled through the VideoDriver. When each node is called to render, it calls a render function in the VideoDriver that defines how that node draws to the scene. The scene promises of 1000 points, or LEDs. Each node eventually breaks down into a collection of drawing points. The function used to draw a point is drawRawPoint, which takes in the encoded color and position and writes a byte of data to the buffer. This 1000 byte buffer is what is sent to the LED cube circuit.

Each runnable then uses nodes to create a game or animation on the cube. The architecture allows the runnables to be far removed for any low level software functions. This allows writing of the applications to be as simple as creating the game using the same logic as if it were used in a 2D display or 3D engine.

Additional components to the Virtual Environment include the controller class which checks for input each frame and the runnable is free to respond how it likes. The wired GameCube controller supports the rumble function, allowing the controller to rumble under certain conditions such as death. Runnables also interact with the sound class which allows a runnable to define a music track to play while running, along with certain short sound effects. The class diagram to the rendering engine can been below in Figure 4.18.

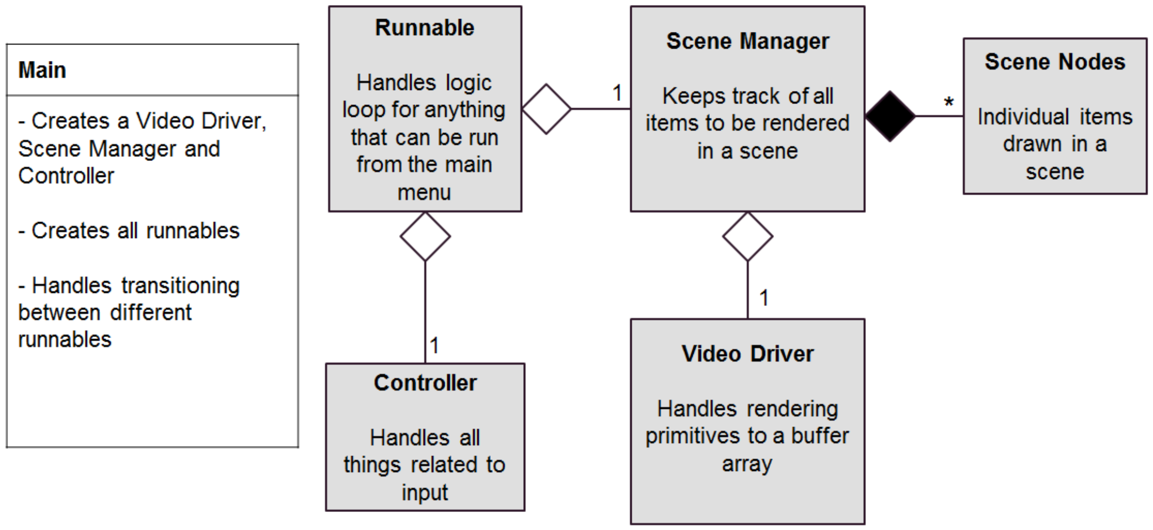

Main
- Creates a Video Driver, Scene Manager and Controller
- Creates all runnables
- Handles transitioning between different runnables
Runnable
Handles logic loop for anything that can be run from the main menu
1
Scene Manager
Keeps track of all items to be rendered in a scene
*
Scene Nodes
Individual items drawn in a scene
1
Controller
Handles all things related to input
1
Video Driver
Handles rendering primitives to a buffer array

Figure 4.18: Rendering Engine Class Diagram

The virtual environment works by using the built in scene manager of the Irrlicht Engine. This is different then the team's implemented scene manager, which extends this to make more objects specific to our games/animations such a star as seen in the next figure. The engine support main basic models. To make an LED Cube simulation, 1000 cubes is all that is needed from the Irrlicht Engine. Other functions of engine include setting and moving the camera, and complete keyboard and mouse input controls. Using simple nested for loops, 1000 cubes are drawn using the scene manager with each having the position set. Draw of objects like the cubes contains support for many familiar graphical options including diffuse, ambient, and specular colors. A simple transparent texture is used for default cubes to simulate an LED off. From here, any drawn cubes will turn on by setting the color to a buffer. The buffer is 1000 bytes, each corresponding to RGB value of one LED/cube. This allows for an 8-bit color to be displayed, 3 bits for red, 3 bits for green, and 2 bits for blue, a palette of 256 colors. This buffer code is set to be close to what the output from the microcontroller to the hardware cube will be. Thus any code that leads to generation of a 1000 byte buffer could easily ported to the final software. A scene with 1000 cube objects configured to display a large cube outline and star is shown in Figure 4.19 shown below.

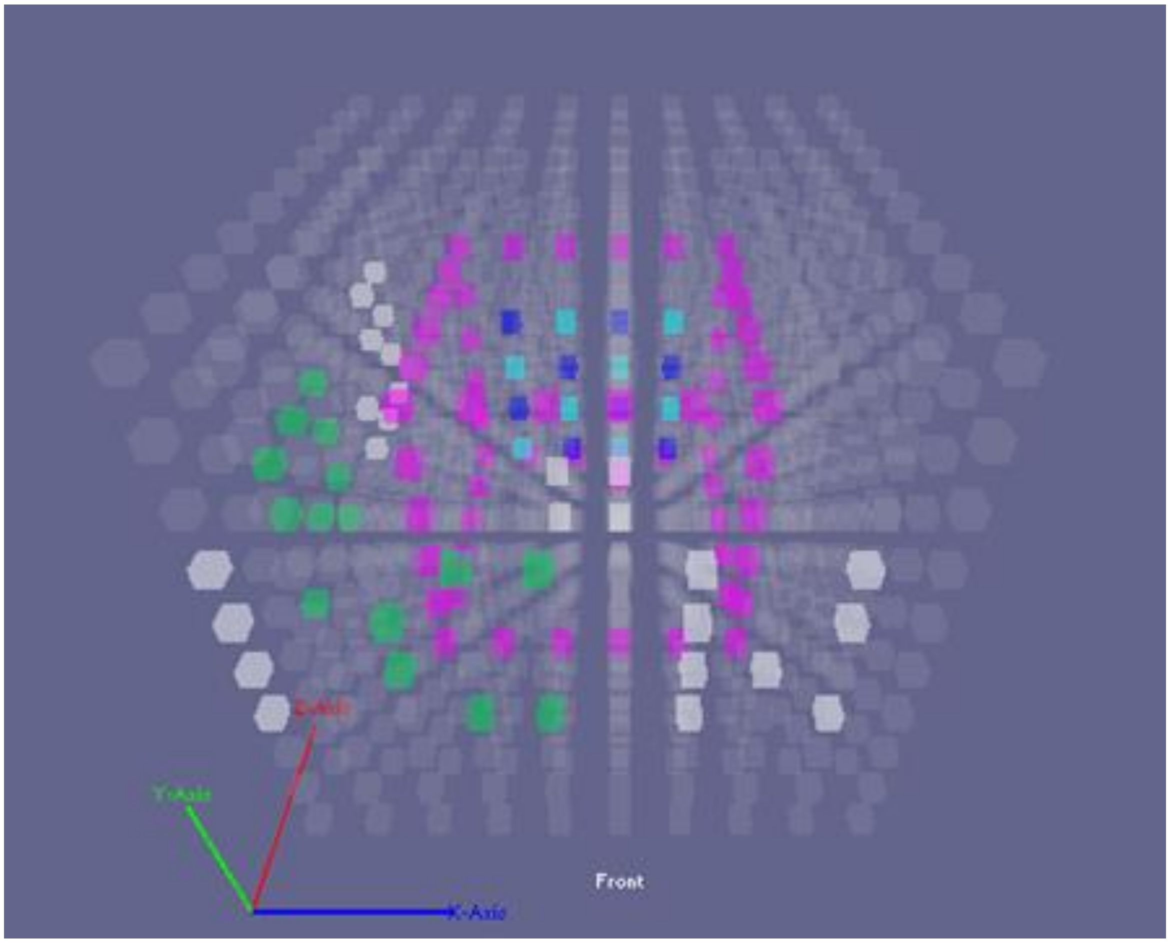


Figure 4.19: Virtual Environment

Overall, the virtual environment software requires little effort compared to the time it saved in testing the software in the final hardware. Not only did it facilitate in testing the games and animations, it also allowed for keyboard mapping of the input controls to the PC for testing the input controls and button configuration. This required very little effort because all that is needed to use a keyboard with a PC is the correct drivers. The final design required a lot of effort before it was ready to test the input configuration. However, the virtual environment allowed the games and animations software to begin development immediately without the hardware completed.

4.7.2 Tiva C Software

The Tiva C Software can be used to verify the functionality of the microcontroller component of the final design. This code is not used in the final design but portions implemented will be portable to the final code. The code should use the same concept as the virtual environment software. All C/C++ software developed that leads to an output buffer of 1000 bytes should be portable between each block. The difference lies in the lower level software that handles the hardware specific pins and timers. The virtual environment requires initialization and utilization of functions in the Irrlicht engine. The Arduino environment will require initialization of the pins and timers. The LED drivers will configure any color depth options and brightness options. The Tiva C software will implement a similar system as main from the virtual environment. After hardware specific initializations, a basic menu was implemented to choose different modes including games and animations. Main takes the output from these classes and configures it for correct output and also relays the controller input. From there, the Tiva C specific code only needed to handle controller input as the animations and games were coded hardware independent in C/C++ leading to the 1000 byte buffer. This allowed for portability to the final design.

Some important aspects of the code specific to the Tiva C included configuring things like clock speeds, data transfer speeds, enabling/disabling peripherals and peripheral configuration. The clock speed for the main controller is not just decided by the external crystal connected to it, it also relies heavily on the configuration done on the software level. The Tiva C has configuration options to utilize a phase lock loop to achieve a higher core clock speed than what the crystal may be providing. This will have to be utilized to achieve the desired 80MHz clock rate for this project. Any hardware peripherals that need to be utilized also need to be configured in the software as well. This include configuring clock speeds for hardware SPI and assigning the input/output pins.

4.7.3 Animations and Games

The overall concept of the GameQube was spawned from the idea that it could be used as a unique way to play games, especially ones that were originally created in a 2D space. By introducing a 3D LED environment, classic games can be rewritten to be played in three dimensions, making them seem like brand new games. Many of the games we chose to design for the cube were originally made to be played in 2D.

One of the games we chose to bring into the 3D world was “Pong”, a classic 2 player game. The objective is for a player to hit a ball with a paddle past the opponent’s paddle to score. Normally, the ball would be able to move in two dimensions (x and y) and a player’s paddle would move in only one dimension (y). However, with the shift to 3D the ball can move in three dimensions (x, y, and z) and the paddles can move in two (x and y). This new functionality adds an entirely new layer of gameplay depth to a previously much simpler game. The game starts when player 1 presses ‘A’ on his controller to release the ball. Each player then moves his paddle with the controller’s analog stick to attempt to put it in front of the incoming ball to hit it back towards his opponent. When a ball goes past a paddle, that player’s cube side turns red and the other player’s side turns green to signify that he scored. Once a player scores enough times, his cube side flashes green which means that he won.

Another game that is playable on the GameQube is our version of “Atari Breakout” called “Brick”. In this game, a player controls a paddle and hits a ball away from him, similar to “Pong”. The objective, though, is to have the ball hit the colored bricks on the other side of the cube to destroy them. When all the bricks are destroyed, the player wins the game. Also similar to “Pong”, the ball is released into play by pressing ‘A’ and the paddle is controlled with the analog stick.

The third game that has been implemented for the cube is “Snake”. In “Snake”, the player is in control of a moving snake (line of LEDs) and needs to navigate it to a fruit that will increase the snake’s size by one LED when collected. This continues until the player either runs the snake into itself or a wall, which ends the game. The snake can move in 3D and the fruit is located in 3D, which makes keeping the snake alive much more difficult.

The fourth game designed for the cube is called “Block”. The goal of this game is for the player to rotate a shape in such a way that it will fit into a hole in a wall. At the start of the game, the player is given an odd shape and is shown a wall of LEDs at the bottom of the cube with a hole in it. The player can then start the game by beginning to rotate the shape with the analog stick. At that moment, a timer starts to tick down on the left side of the screen. Once the timer hits the bottom of the cube, the LED wall will rise up towards the top of the cube. If the rotated shape fits through the hole, the player is awarded a point and a new shape and hole appears. If the shape does not fit, the game ends and a screen shows the player how many points were awarded to him.

In order to diversify the types of applications on the cube, we decided to design animations that a user could watch. Once the animation runnable is selected, an assortment of animations begins to play, automatically shifting from one to the next. The user can cycle through the animations manually with the ‘A’ button. When the list of animations reaches its end, it loops back to the first animation and runs through them again. With the number of games and animations that are present on the GameQube, it is unlikely that a user will not find something interesting and enjoyable to them.

The last runnable is the Music Visualizer. The music visualizer allows for a user to play with stored music tracks or input their own, to run music based animations. The animations variable includes seeing the raw waveform from either the left channel, right channel or both.

4.8 PCB Design

The PCB design for this project consists of two separate circuit boards. One board is dedicated to the display and all the necessary components to control the LEDs. The second board is used for the main processor and sound processing. Figure 4.20 shows the design for the LED display circuitry.

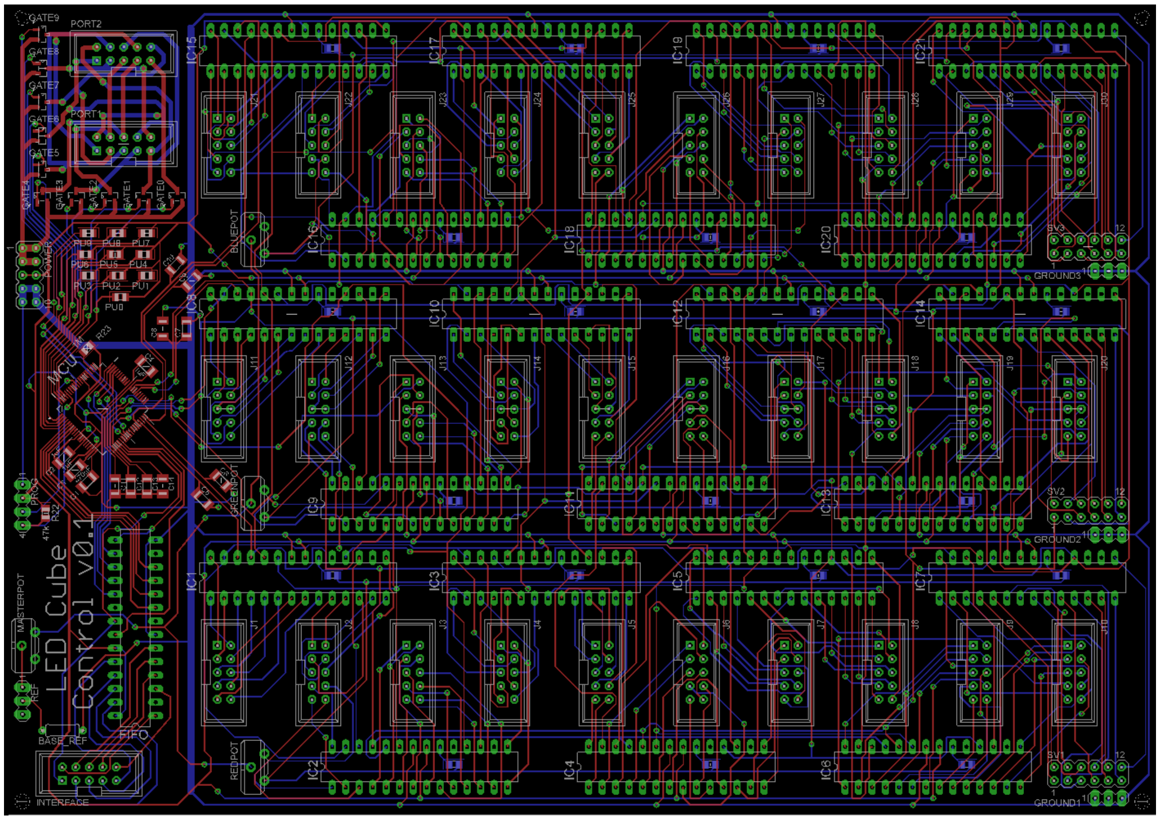


Figure 4.20 LED Display PCB

Some key features of the LED display PCB include the use of through hole decoupling capacitors not pictured in the design. By doing this, lots of space was able to be saved on the board. The board also features several capacitors that were not used in the final assembly and only served as a way to access the control lines for any additional prototyping needs that may arise. Figure 4.21 shows the design for the main processor PCB.

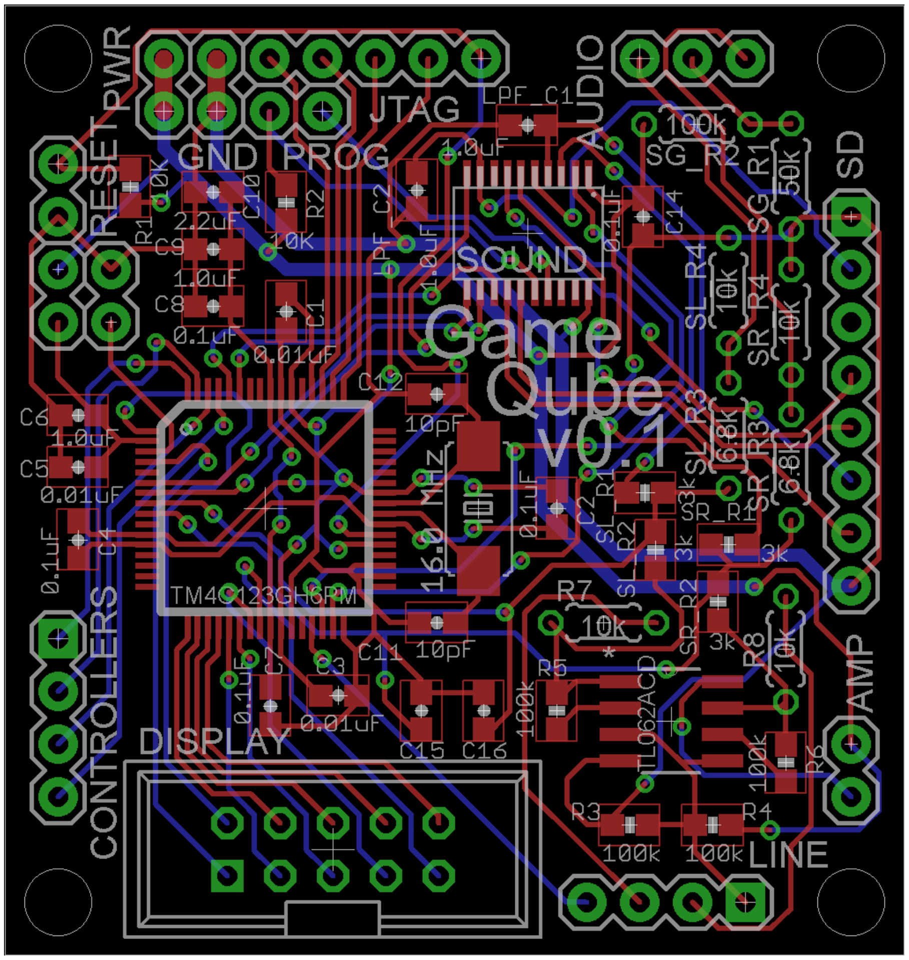


Figure 4.21 Main Processor PCB

Some key features of the main processor PCB include the use of through hole resistors for the components that affect the audio mixing. This allows for the resistance values to easily be swapped if they need to be for a little extra prototyping. The main processor PCB also makes room for the sound processor and includes the interface on the board itself.

5.0 Design Summary of Hardware and Software

The 10x10x10 LED Cube includes both hardware and software design. The hardware design begins with the physical construction and soldering of the 1000 LEDs. Controlling that includes the logic design made using a custom circuit. The main microcontroller serves as the host of the software and mediates between the output and the input. The software design includes designing a successful simulation environment and designing animations and games to function in a 3D display.

5.1 Parts List

The initial parts chosen by the group based on initial research are shown in Figures 5.1.1 and 5.1.2 below. The category and part chosen is shown with the source of the purchase and price. Most parts were purchased online. Many of electronic components such as the LEDs were cheapest on Ebay but required long shipping times. More detailed information on parts selection is available in section 6.1 and budget and shipping is available in the section 8.2.

Category:Part Chosen	**Source**	**Price**
LEDs: 5mm RGB LEDs x1000	Ebay.com	$60
LED Cables: 10pin Ribbon Cable	Ebay.com	$15
LED Cables: Standard Wire	Skycraft	$15
LED Cables: Ribbon Cable Sockets x20	Ebay.com	$10
Logic Circuit:28pin DIP IC Sockets x5	Ebay.com	$5
LED Drivers: TLC5940NT x20	Ebay.com	$17.50
LED Microcontroller: MSP430F5529	Texas Instruments	0
Logic Circuit: Resistors/Capacitors	---	0
FIFO Memory Buffer	Mouser.com	$1.50

Figure 5.1.1: LED Parts List

Main Microcontroller: TM4C123GH6PM	TI.com	0
Programmer: TI Tiva Launchpad	TI.com	$15
Controller: GameCube Controllers	--	$0
PCB: LED Circuit Board	PCB.Net	$25
PCB: Microcontroller Board	OSH Park	$15
Hardware casing: Acrylic and Wood	Skycraft	$40
Power Supply: Nintendo GameCube AC	--	$0
Speakers: Generic Spare Speakers	--	$0
Miscellaneous Expenses	--	$50
Total		**$269.00**

Figure 5.1.2: Microcontroller, Power, and Peripherals Parts List

5.2 Hardware Design Summary

The hardware can be broken down into three main pieces with sub pieces in between that helped the communication flow from the input to the display. The heart of the design is the main processor. For this, a microcontroller from TI was used under the part number TM4C123GH6PM. This board directly interfaces with LED circuit using 10-pin data connector. The software is stored and ran on this chip. Additionally this board hosts the sound design and the input design.

Once the microcontroller has the input, the game logic is computed and a frame will is rendered to a video buffer in the chips RAM. To output the video buffer to the display, the 10 pin connector is to be used to send the data eight bits at a time. The remaining two pins are to be used for control of the data transfer. In order to display the video memory to the display, an additional microcontroller was used to decode the bytes of data. Each byte will translate into a color for an LED. Using the MSP430 microcontroller and the TLC5940 LED drivers, the final desired image will be rendered on the LED matrix. This then ends the datapath for the hardware. The flow of the hardware design between each of these components can be seen in Figure 5.2 below.

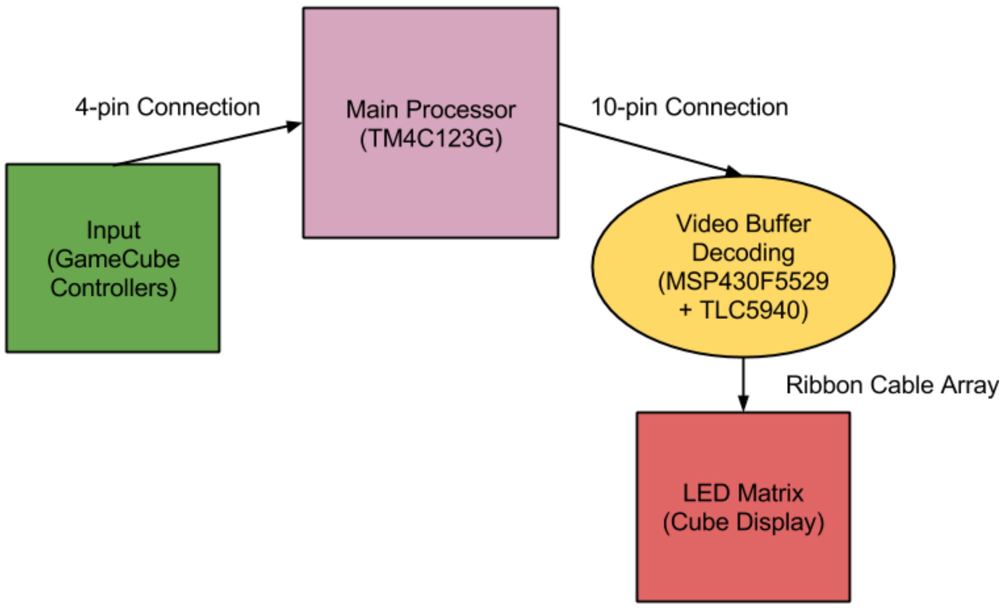

4-pin Connection
Main Processor (TM4C123G)
10-pin Connection
Input (GameCube Controllers)
Video Buffer Decoding (MSP430F5529 + TLC5940)
Ribbon Cable Array
LED Matrix (Cube Display)

Figure 5.2: Hardware Design Flow

The overall construction of the cube involved soldering together all one thousand LEDs in a common anode per layer design and include over 4000 solder points with over 500 feet of wire. Due to time constraints, construction of the cube will began very early in the design process, since it is so vital to the debugging of the additional hardware.

A clear plexiglass enclosure was made for the display to protect it from being damaged and wire shorting. This display of LEDs are mounted on a box base which houses the additional hardware components. Inside includes the microcontroller board, the speakers, the LED circuit, and the power supply. The end result exposes no internals and only has a power cord, power button, and controller hub visible. The power cord will connect directly to the power supply and when turned on will not activate the cube. Like a PC, the power supply will not start when the supply is plugged in but instead be wired to turn on with a push button on the front of the LED cube. Finally, the GameCube controller will be the last component of the hardware. Figure 5.3 below shows a picture of the final product.

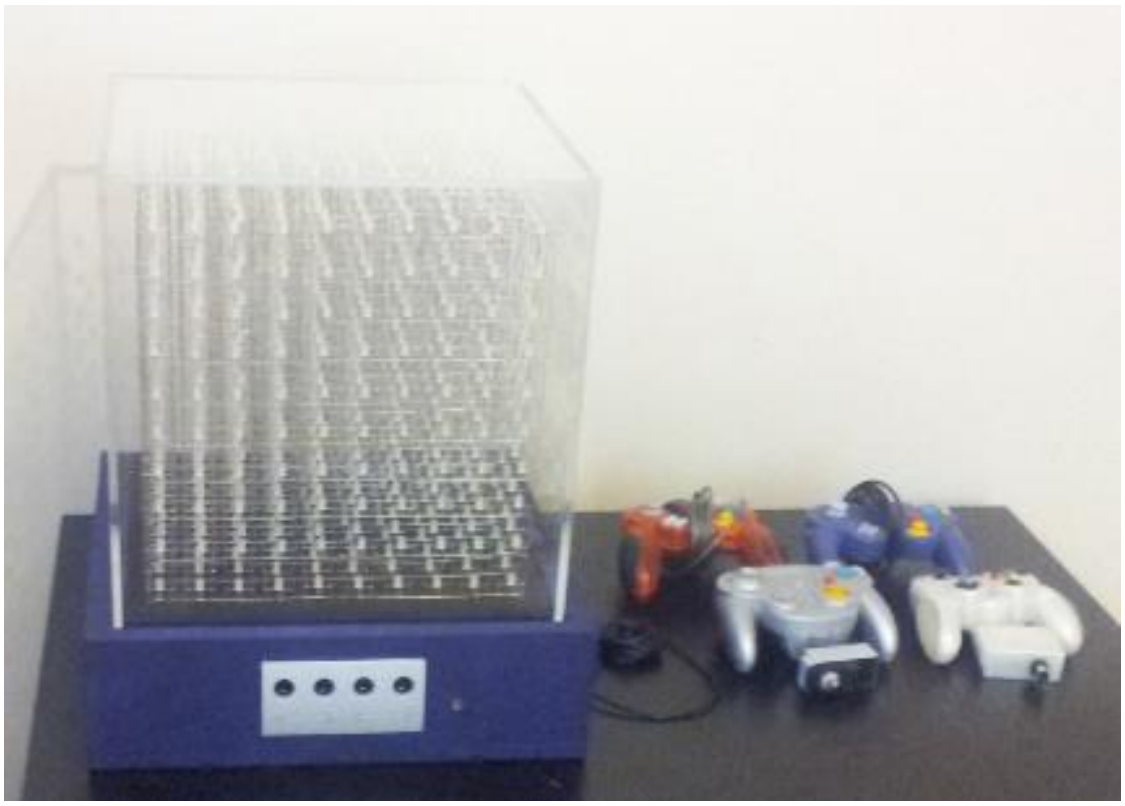


Figure 5.3: Final Design

5.3 Software Design Summary

The LED Cube software is hosted in the TM4C123GH6PM flash memory. The software is well under 256 KBytes in size. The basic software user interface is shown in Figure 5.4 below.

The user interface for the project was designed with simplicity and usability as major factors. It needed to be easy for any user to navigate, regardless of technical history. With that in mind, the interface was designed with a main menu showing the runnable applications, the running application screens, and a pause screen for each running application.

When the user first turns the GameQube on, a quick animation is played to show that the cube is starting up. This animation mimics the original Nintendo GameCube boot animation. After the animation completes, the user is brought to the main menu. The main menu of the cube was created as a vertical scrolling menu, controlled with up and down pulls on the controller’s analog stick or D-pad. This allowed for quick transitions between selectable games and animations, as well as the ability to bring focus to a single runnable at a time. To take advantage of this focus, each runnable has its name written in LED light scrolling around the sides of the cube. Also, each runnable has a small animation running on the main menu that shows its functionality. The user is able to then press the ‘A’ button on the controller to start the runnable that is currently being shown.

At this point, the user starts to play or watch whichever runnable has been chosen. During the execution of the runnable, it is possible for it to be paused if the user decides to take a break. By pressing the 'START/PAUSE' button on the controller, the user is brought to a pause screen that suspends the current action the runnable is taking. At this screen, the user sees a shrunken version of the paused runnable inside of a cube along with a scrolling pause icon. The options for the user are to press 'START/PAUSE' again to resume the runnable or to press 'A' to be brought back to the main menu.

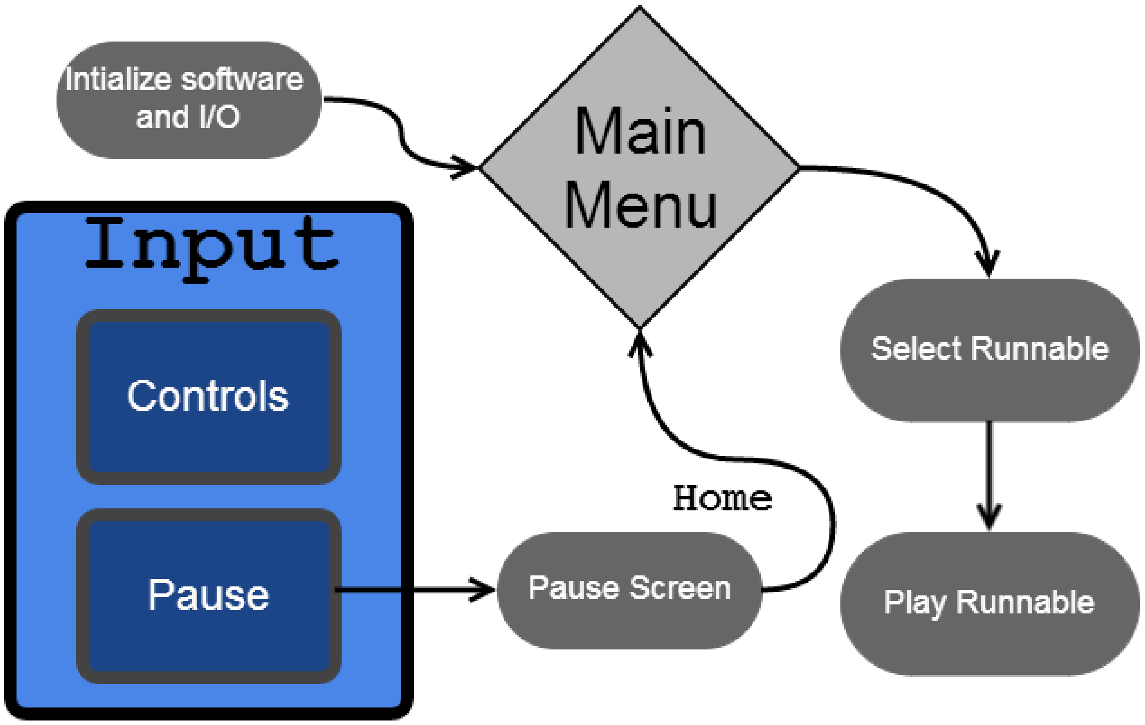


Figure 5.4: Software Design Flowchart

The software runs on the TM4C123GH6PM and starts with main() function. Here the software initiates any global variables and configures the board for the input and output. The output is configured for the 10 digital output pins connected to the LED circuit. The input is configured for two GameCube controllers. The input connection works by sending in a data signal to each controller during each frame. The controllers then respond with the button data telling us with buttons are being pressed. With the button data known, the rendering engine software can access this data through the controller class where the data is stored.

The animations and games are coded using a fundemental CSceneManger class that supports drawing of simple objects like cubes. The class also keeps track of the xyz position of the object and allowing movement. The main function of this class is converting the object into the correct format for output, a 1000 byte buffer. This 1000 byte buffer contains the color of each 1000 LEDs in an RGB 8bit color encoding. Each bit will be sent one by one per clock cycle, until the

1000 bytes are received by the cube to draw. An additional two bits are needed as output for communication synchronization between the microcontroller and LED cube. All games and animations class keep track of all current objects drawn on the scene, from there the scene manger must combine all objects into one useable output buffer. After a predetermined amount of time, the microcontroller timer will request an output from main to CSceneManger which must return the current output buffer. Collisions of objects must be checked for in the games class. Given the scene objects, the animations and games will utilize nested for loops for repeat draw calls and create visual effects. The scene manager will also delete the objects when needed. Animations include all types of basic and complex visual effects, including interactive animations. Games include Pong, Snake, Brick, and Block.

The Virtual Environment code works directly with the embedded code on Tiva. Main sets up the timing for the controllers and sends the data to the classes. The embedded code handles getting the sound data and synchronizing it to the games and effects. All the pins and analog features of the microcontroller are initialized and configured to be used appropriately Lastly the buffer gets sent to the display each frame.

Overall the software involves the data and syncing setup on the embedded level, along with the rendering engine. The rendering engine allows for high level code that allows for making games and animations.

5.4 Design Issues

In terms of the LED assembly, there were several issues that needed to be kept in mind. The first and biggest issue was determining which method to drive the LEDs. This was a delicate balancing act of raw conceptual control and price effectiveness. The immediate design chosen with simple shift register, provided a complete and raw way to control the LEDs. In effect, it would have been like designing the logic inside the IC LED drivers, but allowing for complete control over changing any aspect of the architecture. The downside to this, however, was in keeping the design cost effective. All the little components add up as well as take up a lot of space, which would in turn raise PCB board costs. The challenge now, was to find an LED driver chip that was compact and price effective, but also was capable of providing the amount of control that was desired for this project. Through many hours of research and datasheet reading, the TLC5940 LED driver was finally decided upon over the initial raw control concept.

Another issue that needed to be overcome was the actual construction of the LEDs into a cube matrix. There are plenty of examples of the methods that other people to be found online, but this does not necessarily make for the best solution. For example, the final assembly was decided to be carried out very similarly to Kevin Darrah’s method (discussed in the research section), but the wires would be spaced out evenly so that the density of wires in any one location, from any perspective of the cube, would look fairly consistent. The base of the assembly also strays from most of the assembly methods discovered in research. Instead of using a piece of foam to simply push the cathodes through, a solid piece of wood would be used. Holes would be precisely measure and drilled into the wood to ensure correct spacing and the cathodes would then be tacked in place once to ensure their positions will not move up and down within the holes.

6.0 Project Prototype Construction and Coding

6.1 Parts Selections and Acquisition

The LED driver TLC5940 was selected based on it's relatively inexpensive price and inclusion of PWM for brightness control. This makes the whole LED control circuit much more compact in size, which resulted in a smaller cost for PCB design. The TLC5940 was found fairly cheaply priced on Ebay, especially when purchasing in the large amount we needed (around 21 chips). The microcontroller for the LED control was selected more out of convenience, due to the fact that free samples of the MSP430 could be acquired from Texas Instruments under a student account on their website. The MSP430 line offers a wide selection of specifications for the microcontroller, so one was simply chosen that met the projects specifications. Texas Instruments website really helped to tune into the MSP430 with exactly the specifications needed.

The microcontroller was selected based on being the best fit on requirements and programmer. The microcontroller was ordered from TI as a pair of samples, many were ordered for prototyping. The Tiva Launchpad was purchased from TI for prototyping, testing, and programming.

For a lot of the less research intensive aspects of the project, but still vital, the question of where to acquire these materials lingered through Senior Design 2. In order to keep cost down, the first place to go to when in need of such supplies would be a surplus store. A store known as Skycraft, which sells surplus supplies for many areas of electrical and mechanical engineering. For example, wire is sold there at a very reasonable price and the variety offered allows for the exact type of wire needed to be adequately selected. Going to other mainstream stores for such materials is also an option, but will most likely result in a higher overall cost, since they do not have the best prices. However, some things may still need to be purchased at such places, due to the fact that an item is not guaranteed to be found in a surplus store.

6.2 PCB Vendor and Assembly

Upon completion of the circuit design and PCB layout, the final PCB board must be ordered from a vendor. To allow the LED circuit to be simplified and allow the microcontroller section to be separate, two PCB boards must be made. Initial testing and developing was done using breadboards when possible. The PCB boards were designed and completed using Eagle PCB Design Software. Once testing is complete the PCB boards can be ordered and assembled. Many websites allow for uploading Eagle files of PCB and ordered. They usually require an accompany gerber file which incorporates more than the PCB schematic. The gerber files include copper layers, solder masks, and drill holes. Drill holes will be used to attach the final boards to the case. When choosing the PCB vendor the most important factors are price and quality. Early completion of verified designs will be necessary as shipping may take a few weeks. Since our design contains two boards, it was smarter and cheaper to design the boards on the same PCB, and cut away the separate designs. This allowed the two boards to stay separate, yet have only one PCB board ordered. This technique required the design of the board to contain each space to separate the two designs successfully. Many PCB vendors have student deals that the team should take advantage of. Ultimately, two separate PCB manufacturers were used. PCB.net was used to fabricate the larger LED display driver board as they did not charge by size and had a simple flat rate of $25. For the main processor board, OSH Park was used due to the small size of the board and their per square inch pricing. The main processor board only cost around $15 for three copies.

Once the PCB boards have been acquired, simple soldering was done by the group. Any advanced soldering required help from some of the many resources available at UCF. Once the functionality of the PCB boards were verified and fully tested they were mounted into the final casing.

6.3 Final Coding Plan

The code was kept in a shared google drive to allow for access to all team members. To begin coding of the final animations and games, the virtual environment was the first priority. Setting up a successful virtual environment allowed the team to code the final animations and games while the LED cube was being assembled. Once the infrastructure of the virtual environment was deemed to be an accurate and reliable simulation of the hardware LED cube, the coding of hardware independent animations and games in C/C++ begun. The virtual environment was developed in Microsoft Visual C++ 2010, as the Irrlicht Engine utilizes many of its tools. Upon completion of the virtual environment, the animations and games were IDE and hardware independent, with as few external dependencies as possible. This allowed for the animation and game files to be easily included in either the virtual environment or the microcontroller software with little to no modification. The final code was developed and programmed to the microcontroller using Code Composer Studio. The animations and games can be developed in any C++ editor, however because of the virtual environment, they were developed in Visual C++ for convenience and immediate testing in the virtual cube.

The MSP430 code was written in C using TI Code Composer Studio. Initial prototyping with the TLC5940 LED drivers proved that several methods will work in interfacing with these chips. The TLC5940 requires several inputs to control the LEDs properly. Firstly, the LED driver needs the clock for PWM to be input manually to the chip. There are several ways of going about this, however. The first involves also taking care of the serial data transmission. So the software would first check to see what the LED states should be and then begin a loop where in each iteration the PWM clock it ticked and one bit of data is also sent serially to the chip. This occurs until there is not more data and then the PWM clocked is ticked again and again by itself until there have been 4096 ticks. once the full PWM cycle has been finished, the serial data is then latched, PWM data is blanked and the process starts all over again for the next frame. This method was proven successful in a small scale test with the TLC5940, but has several downfalls. The first downfall is that the software must waste time to tick the PWM clock manually, which means that the final PWM speed will be less that the actual microcontroller clock speed. The other downfall is that no more than 4096 bits can be sent during a single frame without complicating the code to undesirable levels. However, there are several methods to optimize this code.

Another coding method for the LED control would be to use the built in hardware timers to control the PWM clock. This means that the PWM clock can now run at exactly the same frequency as the microcontroller or even slower, if the microcontroller needs more time to send the serial data. An interrupt is then programmed to be run every time the timer ticks 4096 times, or the full PWM cycle completes. When the interrupt is triggered, the PWM cycle is reset and the next frames data is then serially sent to the LED drivers. To assist in the serial data transfer, the hardware SPI capabilities of the MSP430 are used to send the data at a rate almost matching the microcontrollers own clock speed. The reason this could not be achieved before was because to tick an I/O port manually like a clock, it takes several clock cycles for the microcontroller to first set the pin high and then set the pin low. Without using hardware SPI, it is guaranteed that the data transfer will always occur rate than the core clock frequency. Through this interrupt method, the low voltage state can also then be utilized only to return to full power when the interrupt is triggered again. This allows the chip to go into sort of a sleep state when the data transfer is complete and only the PWM clock output is needed. This will save the LED control from consuming as much power as the method mentioned earlier.

The animations and games software follow the architecture of utilizing the CSceneManager to draw common objects and animations. This class creates the objects in RGB format for each individual cube/led. This class also creates, deletes, and moves the position of the object based on the calls from the animation and game logic. A strong and reliable scene manager was developed first, following simple animations which led to more complex animations. Simple games were developed next, followed by custom games that make use of a 3D display. Future features, such as more complex controls were then integrated. Adding support for additional controllers or even more advanced controls using the Wiimote was also considered. Utilizing the motion controls of the Wiimote such as the accelerometer and gyroscope sensor would only be added if convenient and useful value is actually added. A constraint of this would be the memory space available.

7.0 Project Prototype Testing

7.1 Hardware Test Environment

The hardware testing environment consisted of a laptop computer, microcontroller development boards, oscilloscope, multimeter and breadboard with a surplus of electrical components (wires, resistors, etc). Using this equipment, prototyping designs and small scale testing phases were completed and tested to ensure specifications were met for the project as well as ICs and electrical components. This was important in order to make sure the final design was not stressed and unstable. Figure 7.1 shows the basic hardware test environment with a laptop used to program the processors through the TI launchpads.

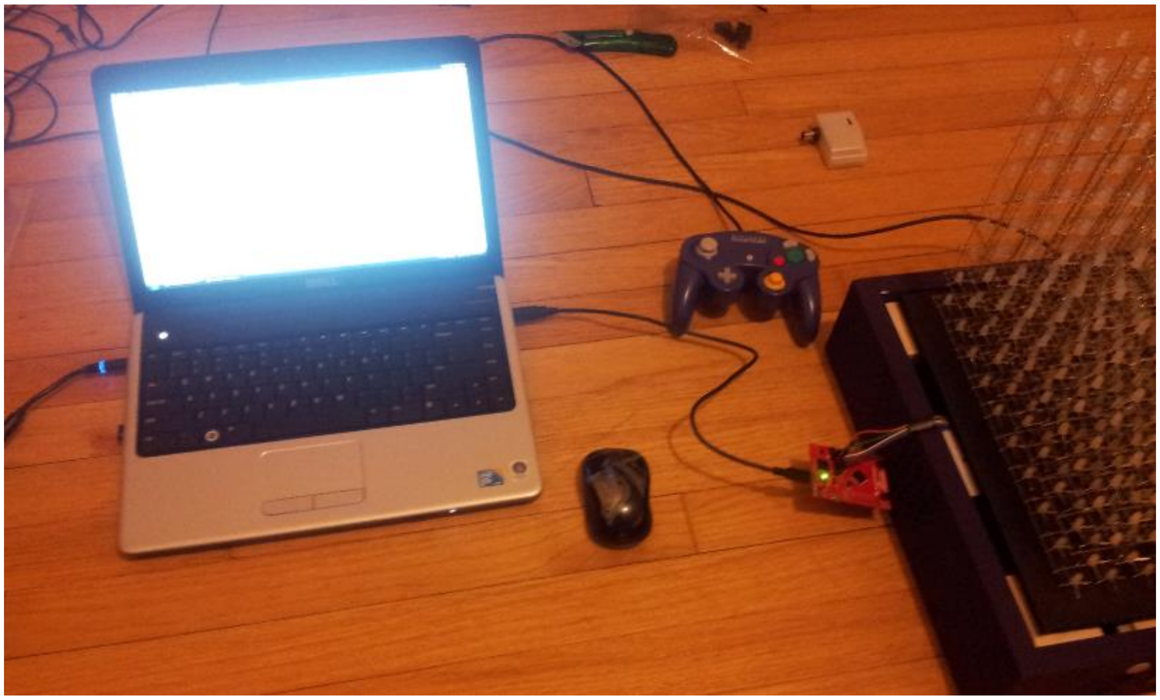


Figure 7.1: Hardware Test Environment

7.2 Hardware Specific Testing

LED cube design was split into three phases. The first phase consisted of a breadboard prototype test to get the MSP430 microcontroller to interface correctly with the TLC5940 LED drivers, shown in Figure 7.2 below. This involved getting the control pins established and the overall control implementation working. An oscilloscope was used to measure how fast the data transfers were occurring and what limits were there, if any, if the design was scaled. Important factors are current draw from the LEDs at the desired brightness, timing checks for serial data communication, and achievable PWM frequency/ refresh rates.

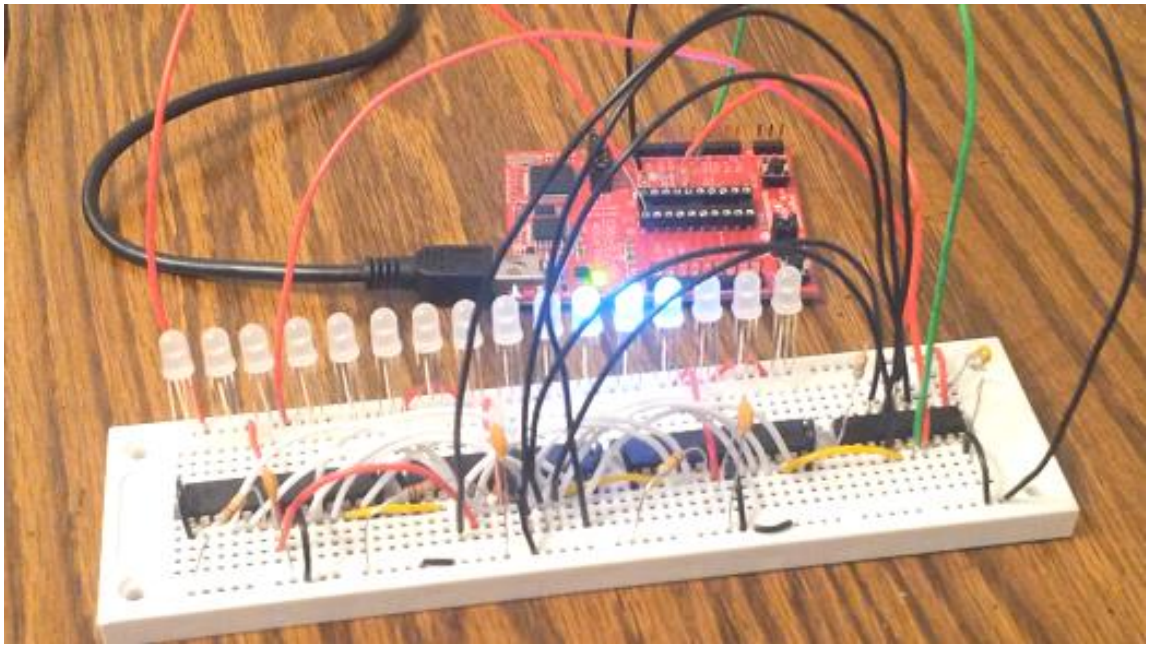


Figure 7.2: MSP430 and TLC5940 Testing Environment

Construction of the LED matrix also involved small testing steps to ensure that the LEDs were behaving as expected. This was done so that bad LEDs could be caught early on and events like having to replace an LED in a hard to reach place could be avoided. Once 100 LEDs were soldered together in a grid before the final cube assembly, the LEDs were placed under a stress environment where they were run at a moderate 10 mA for at least three hours time. This ensured the LEDs were dependable for long hours of continuous operation. Again, catching any LEDs that do not meet the requirements helped to avoid a much more difficult situation later on. The LED time stress testing is shown in Figure 7.3 below.

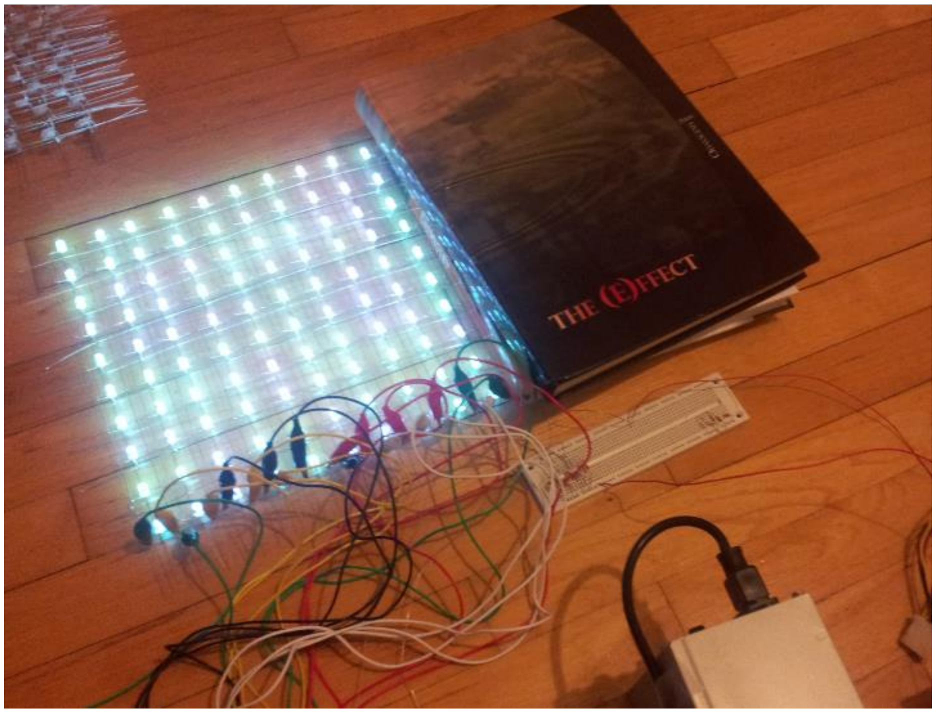


Figure 7.3: LED Time Stress Testing Environment

The second phase consisted of a completely constructed LED cube and all logic devices mapped out on a breadboard. At least one layer of the cube was able to be completely controlled by the logic with brightness, refresh rates and power requirements fully calculated.

The third phase consisted of appropriate multiplexing of all ten layers as well as a proven interfacing technique between the main processor and control devices in the LED display. This was the last buffer phase before designing the PCB board for the final permanent LED control layout.

There was also a final phase in which the final construction, including the final enclosure, was tested for any sorts of stress in the electrical components or bugs in the interfacing of the final package.

To help with the testing of the LED control, there were several tools used to program and debug the small microcontroller. Texas Instruments provides a line of development tools specifically centered around the MSP430. Known as the Launchpad, this development tool aided in the programming and debugging. Along with the Launchpad, Texas Instruments provides a free software development environment known as Code Composer Studio. The Launchpad and Code Composer Studio together, provide a vital environmental that helped with the software development and provided features like stepping through the program with interrupts as well as monitoring factors like register values along the way.

7.3 Final Hardware Prototypes

A large factor in the success of the hardware designs is owed to the making of accurate prototypes of every design made. Many simple and common mistakes were found that would have cost time and money if not tested. Missing design errors and making the PCB would lead to a long delay. Each design was build a either a breadboard or perfboard. The first prototype was the LED circuit prototype, which involved the wiring from all the LED drivers to the LED cube, another prototype board for the MSP430. These two can be seen below in Figure 7.4. Any adjustments and mistakes were found at this point before the PCB design was order. FIgure 7.5 shows the two prototypes juxtaposed to their equivalent PCB board.

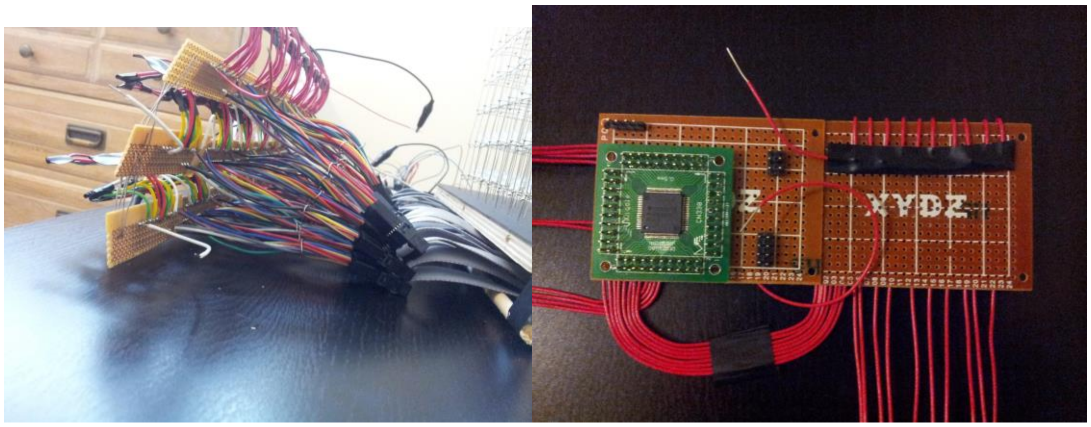


Figure 7.4: LED Circuit Prototype Boards

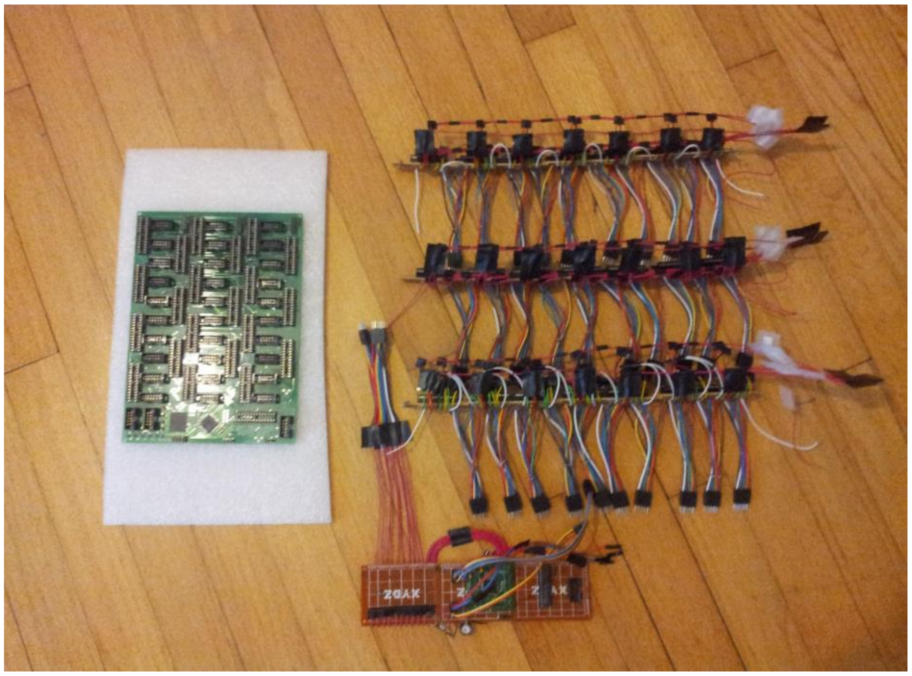


Figure 7.5: LED Circuit PCB(left) and LED Circuit Prototype Boards(right)

The prototype board for Tiva microcontroller was made next. This design was interfaced and tested with other prototypes. After this was verified to be working, the sound design began. The sound design was also prototyped and tested. The designs were then combined into one PCB Board. The prototype can be seen below in Figure 7.6, and the equivalent PCB in Figure 7.7.

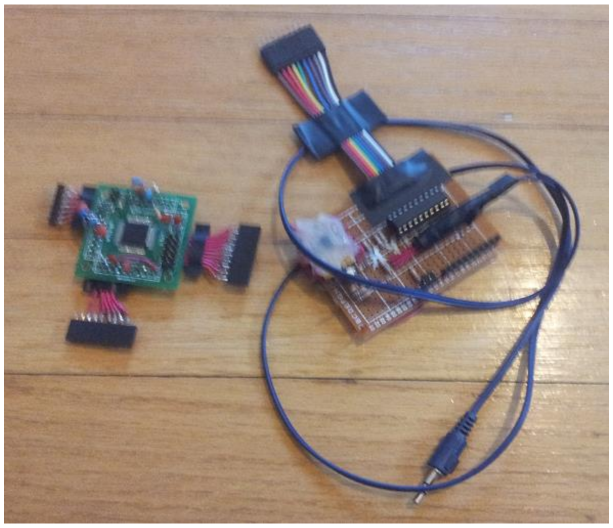


Figure 7.6: Microcontroller Prototype Board

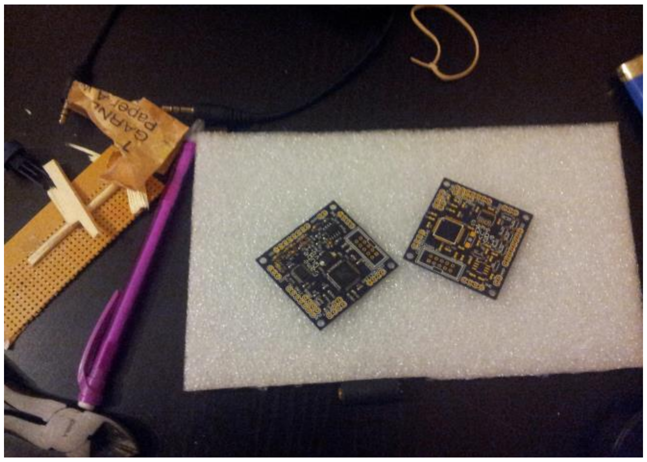


Figure 7.7: Microcontroller PCB Board

7.4 Software Test Environment

The software testing took place in two environments. The first environment was the custom built simulation virtual environment detailed earlier. Not only did it serve as an immediate simulation of the cube software, it also was the host for the preliminary testing of the software. Once it was set-up, the high level software that generates the animations and games had a platform to be viewed and tested. The second environment was the actual hardware cube, the final testing environment. To test this, the final hardware board was used along with the main microcontroller to isolate bugs. Successful testing of the first environment led to the software game and animation logic to be finalized. The second environment testing led to microcontroller software and hardware logic to be finalized.

As mentioned earlier, for the LED control, Code Composer Studio was used to develop the software for the LED control. Code Composer Studio allows for code to be written for any variety of the MSP430 in both assembly and the C programming language. Either of these programming options also provides powerful debugging tools.

7.5 Software Specific Testing

The software contains four main components: general C/C++ code, microcontroller initialization and setup, and controller communication. The general C/C++ hosts the logic for the games and animations using standard high level programming. The main source of this phase of testing was the virtual environment.

Given the completion of the virtual environment, animations and games were then tested. Basic C++ code testing was ran to check for memory leaks for all allocated memory. Testing of unexpected inputs were tested during all phases. Testing of a random and excessive input was ran. Inputs were used during all modes of the software, this input testing was very important. This phase of testing should be the easiest and most familiar to the group as it is isolated in high level programming. The testing of hardware specific code proved to be the most difficult.

The successful testing of the animations were easily tested and verified by running the code and viewing the virtual cube. The software was made to output at a rate similar to the LED cube's hardware refresh rate. Initial tests included making sure with this refresh rate, the animations looked smooth and correct. A large constraint of the virtual environment simulation is the clarity. The virtual environment provides an ideal world situation in which each LED lights up and doesn't interfere with the surroundings. The actual hardware cube had to deal with the LED effect ghosting, in which surrounding LEDs will reflect some of the light. This can also be used as a benefit, in which after effects or trails can be seen. Physical constraints such as the actual blocking from the LEDs and wires played a large role in visibility. Software testing in this area was primarily done in the actual cube environment. The virtual environment testing had the main purpose of testing the logic of the cube. The hardware cube environment was used to see the actual implementation of this cube and how it looks. Based on the visual feedback, it was decided whether or not adjustments needed to be made.

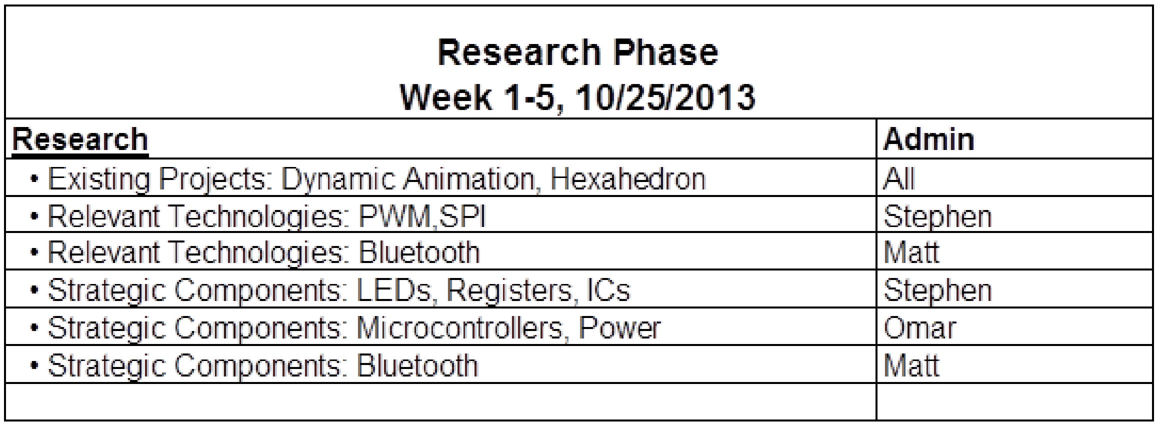
The testing of the games followed the same path as the animations. While the main point of emphasis for the animations testing was visual feedback, the games needed to focus more on the logic and input. Animations use up more of the cube and more lights at once, causing visual adjustments to be made, while the games are relatively more simple visually. Input for the games needed to be exact. Negative testing of unexpected inputs, along with inputs between refresh phases, was heavily tested.

With the completion of testing of each software component, integration testing of all code working together was completed. This was completed by integrating the USB bluetooth and the microcontroller and having the GameCube controller perform basic inputs. The final integration was with the hardware cube.

8.0 Administrative Content

8.1 Milestones

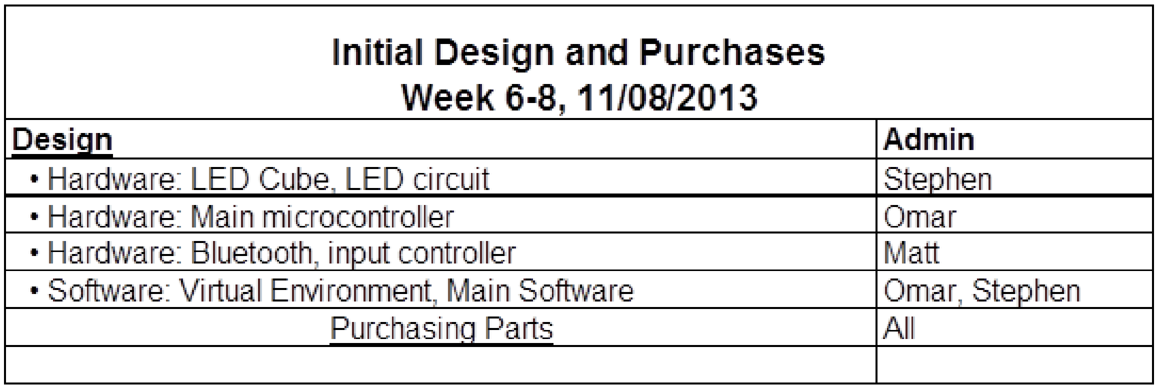
The project milestones for Senior Design 1 and 2 are shown below in all figures labeled Figure 8.1.x. The milestones are divided into separate phases for a certain period of time. Each phase is divided into separate categories with an assigned admin to lead the development of that section and to make sure the group reached the milestones successfully.



Research Phase
Week 1-5, 10/25/2013

Research	Admin
• Existing Projects: Dynamic Animation, Hexahedron	All
• Relevant Technologies: PWM,SPI	Stephen
• Relevant Technologies: Bluetooth	Matt
• Strategic Components: LEDs, Registers, ICs	Stephen
• Strategic Components: Microcontrollers, Power	Omar
• Strategic Components: Bluetooth	Matt

Figure 8.1.1: Research Phase

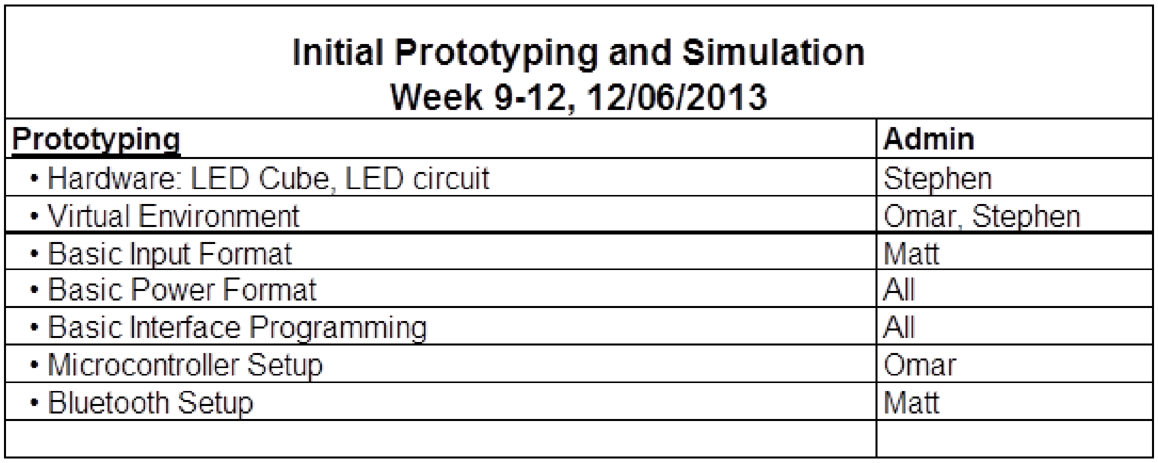


Initial Design and Purchases
Week 6-8, 11/08/2013

Design	Admin
• Hardware: LED Cube, LED circuit	Stephen
• Hardware: Main microcontroller	Omar
• Hardware: Bluetooth, input controller	Matt
• Software: Virtual Environment, Main Software	Omar, Stephen
Purchasing Parts	All

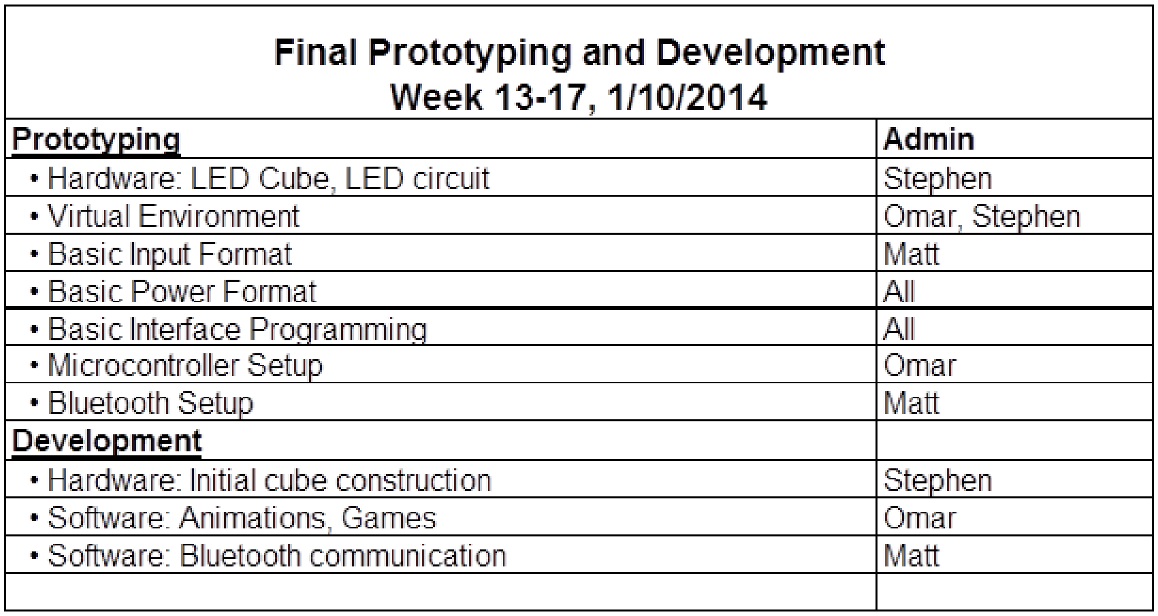
Figure 8.1.2: Initial Design and Purchases

The research phase was completed in time with the initial design as well during Senior Design 1. The initial design of the LED cube withstood for the most part as the construction began almost immediately. Initial design for the microcontroller was redone almost completely during Senior Design 2.



Initial Prototyping and Simulation Week 9-12, 12/06/2013	
Prototyping	**Admin**
• Hardware: LED Cube, LED circuit	Stephen
• Virtual Environment	Omar, Stephen
• Basic Input Format	Matt
• Basic Power Format	All
• Basic Interface Programming	All
• Microcontroller Setup	Omar
• Bluetooth Setup	Matt

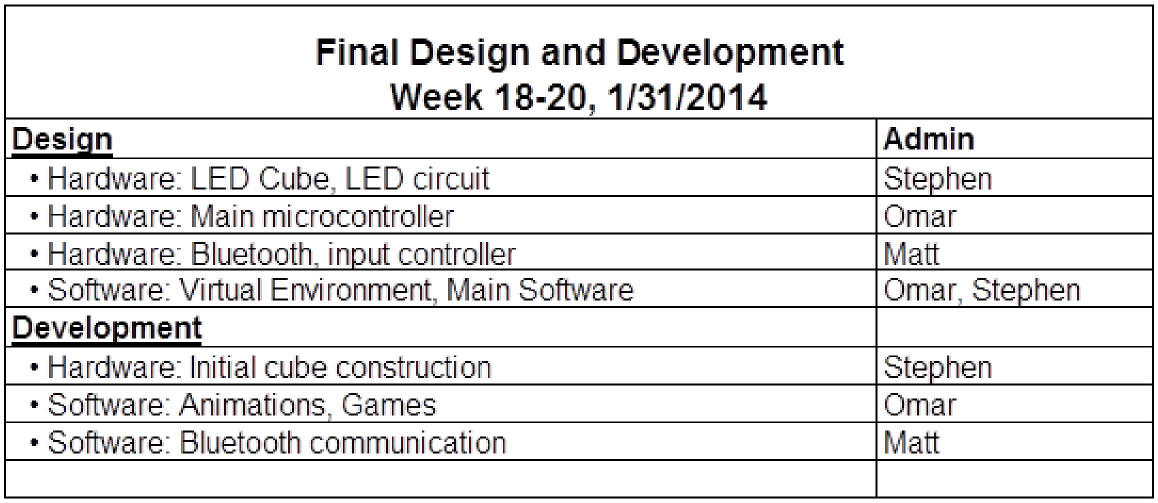
Figure 8.1.3: Initial Prototyping and Simulation



Final Prototyping and Development Week 13-17, 1/10/2014	
Prototyping	**Admin**
• Hardware: LED Cube, LED circuit	Stephen
• Virtual Environment	Omar, Stephen
• Basic Input Format	Matt
• Basic Power Format	All
• Basic Interface Programming	All
• Microcontroller Setup	Omar
• Bluetooth Setup	Matt
Development	
• Hardware: Initial cube construction	Stephen
• Software: Animations, Games	Omar
• Software: Bluetooth communication	Matt

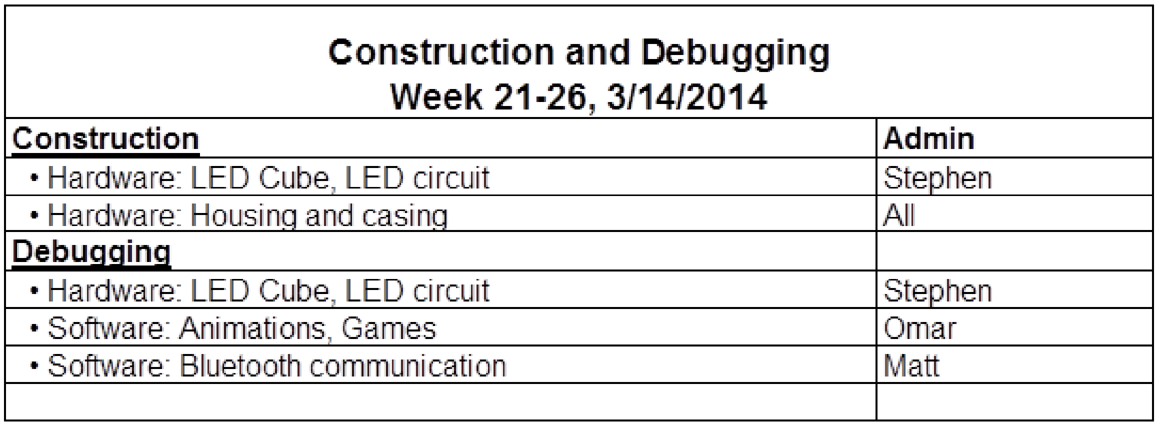
Figure 8.1.4: Final Prototyping and Development

The next two phases of milestones were on schedule for the LED cube. The LED Cube had been completed while the prototypes began development. The LED circuit design went through minor adjustments as the prototypes were being developed. The microcontroller board had been reevaluated and designed to be a new design.



Final Design and Development Week 18-20, 1/31/2014	
Design	**Admin**
• Hardware: LED Cube, LED circuit	Stephen
• Hardware: Main microcontroller	Omar
• Hardware: Bluetooth, input controller	Matt
• Software: Virtual Environment, Main Software	Omar, Stephen
Development	
• Hardware: Initial cube construction	Stephen
• Software: Animations, Games	Omar
• Software: Bluetooth communication	Matt

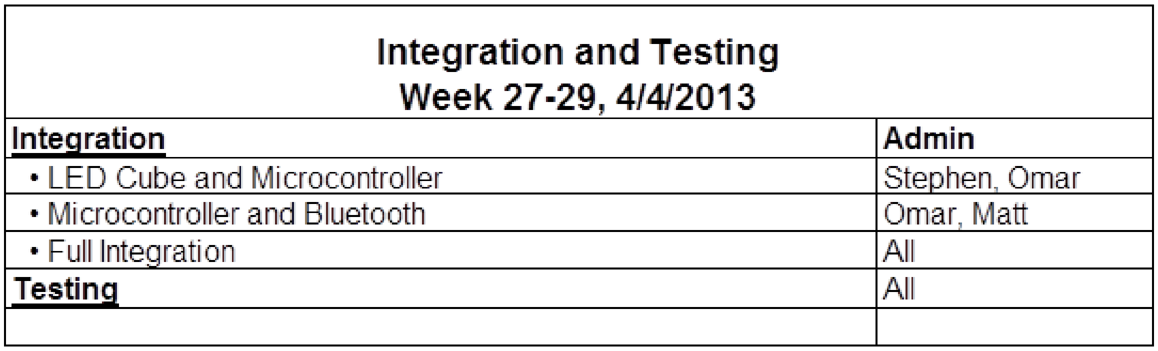
Figure 8.1.5: Final Design and Development



Construction and Debugging Week 21-26, 3/14/2014	
Construction	**Admin**
• Hardware: LED Cube, LED circuit	Stephen
• Hardware: Housing and casing	All
Debugging	
• Hardware: LED Cube, LED circuit	Stephen
• Software: Animations, Games	Omar
• Software: Bluetooth communication	Matt

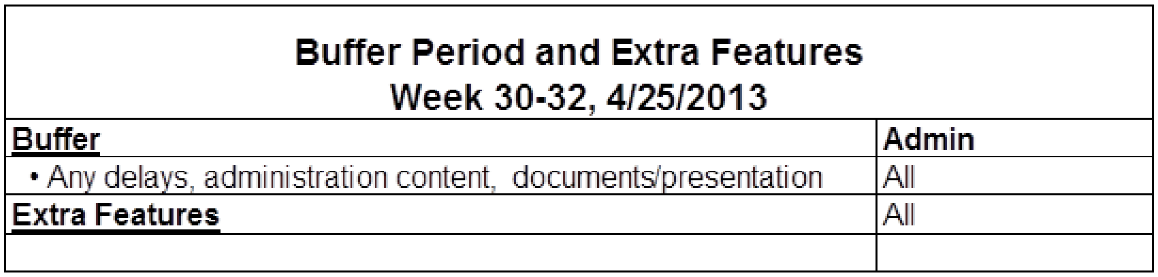
Figure 8.1.6: Construction and Debugging

The final for the hardware had fell back into sync during the next phase. Because the LED circuit required more testing, the microcontroller design had been completed and testing had also began. After testing of both of these components together, the final designs begun. Because the team was ahead of schedule, the microcontroller design fell behind the LED circuit again due to sound being added to the microcontroller. The LED circuit was finalized and ordered, and a couple weeks later the microcontroller was finalized and ordered. From here the construction of the PCB and the housing began ahead of schedule.



Integration and Testing Week 27-29, 4/4/2013	
Integration	**Admin**
• LED Cube and Microcontroller	Stephen, Omar
• Microcontroller and Bluetooth	Omar, Matt
• Full Integration	All
Testing	All

Figure 8.1.7: Integration and Testing



Buffer Period and Extra Features Week 30-32, 4/25/2013	
Buffer	**Admin**
• Any delays, administration content, documents/presentation	All
Extra Features	All

Figure 8.1.8: Buffer Period and Extra Features

The buffer period implemented in initial scheduling proved to be valuable, as it allowed the team time to add sound to the design. As the final construction was being completed, the focus shifted to the software. The vast majority of the user interface, games, and animations were developed at this time. Finally, the software was finalized with a week before the presentation for final testing. The housing, PCB, and software were all completed and integrated for final testing. In the end, the team was able to complete the project fully on time.

8.2 Budget and Finance Discussion

The team decided to find a project that the team would find enjoyable to complete rather than trying to find an idea that would allow sponsorship. The LED cube was a project the team was motivated to complete and came at a reasonable cost. The goal was to find a project that could be self financed with each member paying around $100, although the team was willing to exceed this. Based on rough estimates and purchases so far, the LED cube seemed to fit into this budget. Research on almost all components has been complemented fit the budget. The main concern with the budget is the housing unit. The acrylic casing could end up being a larger cost than anticipated, however early research in local surplus stores shows possible purchases that would fit the project budget.

All purchases were made by the team members, mostly online, and recipes were saved to add to the final budget and divided upon completion of the project. Additional purchases for the project include development tools used to built and complete the project. This includes the wood and soldering iron used to build the cube. Breadboards and wires were also purchased for testing of the electronics.

These tools are all very useful tools for the project and for the group members in the future. Each of these tools were used and kept by the group member who purchases them and will not be factored into the budget. The items to be bought and the total prices are shown in Figure 8.2 below.

Item	**Price**	**Qty**	**Total**	**Shipping**
10pin Ribbon Cable x1	$0.99	16	$15.84	$1.00
Ribbon Cable Sockets x20	$4.99	2	$9.98	$0.00
Ribbon Cable Sockets x4	$1.01	1	$1.01	$0.00
28pin DIP IC Sockets x5	$0.99	5	$4.95	$2.51
TLC5940NT x20	$17.50	1	$17.50	$2.00
5mm RGB LEDs x1000	$60.60	1	$60.60	$22.00
MSP430F5529	$0.00	1	$0.00	-
Resistors/Capacitors	$0.00	0	$0.00	-
Wire (price/ft)	$0.03	500	$15.00	-
2kx9 FIFO CY7C429-25PC x5	$4.99	1	$4.99	$2.00
P-Channel MOSFET x20	$3.19	1	$3.19	$0.00
LED Matrix Wood Base	$4.99	1	$4.99	-
LED Matrix Black Paper	$0.33	1	$0.33	-
LED Matrix Nails	$1.30	1	$1.30	-
LED Display PCB	$25.00	1	$25.00	$0.00
Main Processor PCB	$15.00	1	$15.00	$0.00
PAM8403 Audio Amplifier	$2.49	1	$2.49	$0.00
SD Card Pinout	$0.89	1	$0.89	$0.00
Acrylic	$26.40	1	$26.40	-
Wood Filler	$6.48	1	$6.48	-
Wood	$21.24	1	$21.24	-
Super Glue	$1.97	1	$1.97	-
LED Display Components	$7.85	1	$7.85	$2.86
Main Processor Components	7.14	1	$7.14	$3.4
MSP430G2553	$0.00	1	$0.00	-
TL 792	$0.00	1	$0.00	-
TM4C123GH6PM	$0.00	1	$0.00	-
Paint	$0.69	2	$1.38	-
		Sub Total:	$255.52	
		Shipping:	$35.77	
		Grand Total:	$291.29	

Figure 8.2: Budget Analysis

In conclusion, the team was able to complete the project within the projected budget.

9.0 Conclusion

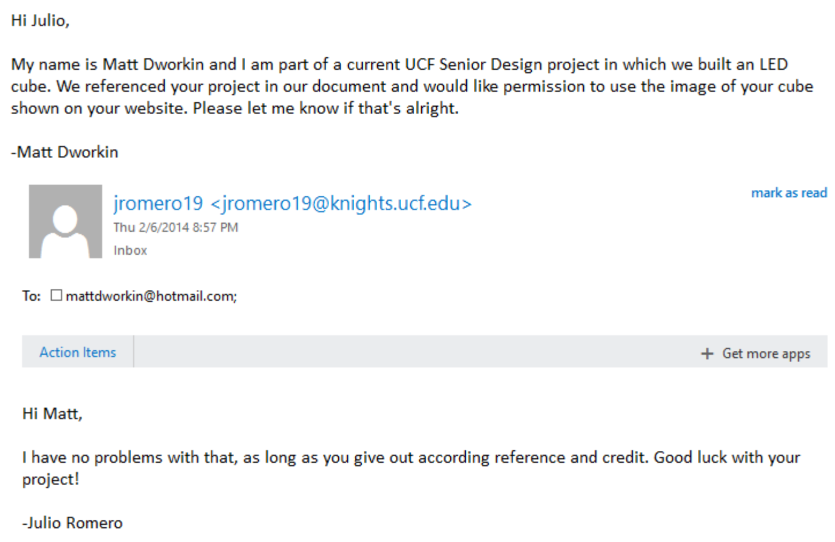
The goal of the GameQube was to create a new and unique take on LED cubes. No longer limited to static animations, the GameQube allows users to play games on an LED cube. The LED cube was constructed first, taking many hours to complete. The LED circuit was designed, prototyping, tested and finalized. The microcontroller board was designed, redesigned, prototyped, tested, and finalized. Ahead of schedule, the team aimed at creating a more complete entertainment device, by adding sound. Finally, the hardware design was completed and integrated into the the final product. The housing was constructed to follow the theme of the Nintendo GameCube. The software design was made in the form of a rendering engine, leading to the development of many games and animations. The final product exceeded the team’s expectations for an interactive LED cube.

Through completing this project, we have gained valuable experience and knowledge that will no doubt assist us in the future. Our original vision of an LED cube for games grew into an entertainment system used for games, animations, and sound visualizing. With the help of our education and resources from UCF we were challenged to complete a full engineering project development. We were able to incorporate a number of ideas that were originally thought to be stretch goals, such as extra games and sound. The GameQube is completely functional and can do everything that we hoped for.

Appendices

Appendix A: Copyright Permissions

1. Figure 3.1: The Multi-Functional Hexahedron
Image featured on eecs.ucf.edu:
http://www.eecs.ucf.edu/seniordesign/sp2013su2013/g05/
Permission Below:



Hi Julio,

My name is Matt Dworkin and I am part of a current UCF Senior Design project in which we built an LED cube. We referenced your project in our document and would like permission to use the image of your cube shown on your website. Please let me know if that's alright.

-Matt Dworkin

jromero19 <jromero19@knights.ucf.edu>
Thu 2/6/2014 8:57 PM
Inbox

mark as read

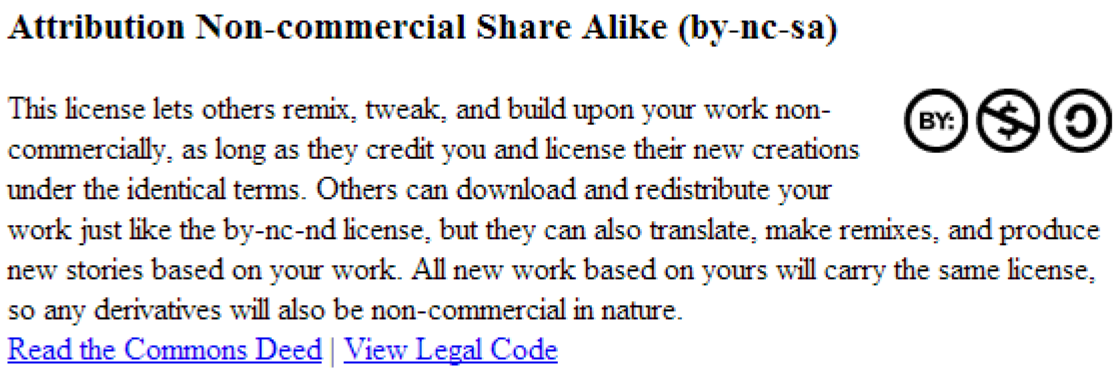
To: ☐ mattdworkin@hotmail.com;

Hi Matt,

I have no problems with that, as long as you give out according reference and credit. Good luck with your project!

-Julio Romero

2. Figure 3.7: 20 Pin ATX Connector Wiring Diagram
Image featured on Instructables.com:
http://www.instructables.com/id/Power-Supply-For-Arduino-power-and-breadboard/?ALLSTEPS
Permission via Instructables.com, Attribution Non-commercial Share Alike:



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.
Read the Commons Deed | View Legal Code