The GameQube

Omar Alami, Stephen Monn, Matthew Dworkin

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — The GameQube consists of an LED volume display cube that a person can play simple games on using a GameCube controller. The GameQube uses 1000 RGB LEDs in a 10x10x10 arrangement and embedded circuitry to form a completely enclosed gaming system. The GameQube takes on the classic LED cube project and expands it to be able to play classic games in a 3D environment. The software and hardware design details that create the completely embedded LED game cube are presented.**

***Index Terms* — Circuits, Microcontrollers, Multiplexing.**

I. Introduction

An LED cube is a group of Light Emitting Diodes configured to form a cube shaped display for animations in 3D. Past LED cube project creations have mainly focused on showing pre-defined animations. This project furthers those ideas so that a user is 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 game such as Pong, Brick, Snake, and Space Invaders have been implemented. A custom made game has also been implemented, Block, designed specifically with 3D in mind. Additional features such as sound have been added to increase the user experience. Altogether, the GameQube strives to go beyond traditional LED cube projects and become a fully interactive entertainment system.

The project consists of the physical LED arrangement, the internal circuitry that controls the LEDs, a housing unit that hosts the hardware, and the software that controls what is being shown on the cube. A main concept used in creating an LED cube is Persistence of Vision. Persistence of Vision directly refers to the way in which the human eye works. When an object is blinking at a high frequency, the human eye cannot keep up, and the perceived image is a constant sort of mixture of the on and off state. This effect combined with multiplexing layers of the cube allows for all LEDs to appear to be on. This concept serves as the main criteria in designing an LED cube.

II. System Components

The design of an LED cube allows for flexibility in creating a functional multiplexing scheme. Power and brightness tradeoffs serve as major guideline when choosing components.

A. Light Emitting Diodes (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.

For the purpose of a display, diffused LEDs are more desirable due to the combinations of colors required to make from mixing the primary colors red, green and blue. Another important factor in choosing diffused LEDs is the fact that they scatter light in all directions. This is important for the Cube display, since it will be viewed from many different angles. Finally, if the display is to use some sort of LED drivers, then common anode RGB LEDs should be used as opposed to common cathode. The LED Cube is composed of 5mm RGB LEDs arranged in a common anode per layer design.

B. Microcontrollers

To create an LED cube there are many specifications that require complex logic computation and I/O control at high speeds. In order to meet the specifications, several microcontrollers have been 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 have been used for the LED logic circuit and main microcontroller board and sound driver.

For some of the less complicated computing and logic tasks in this project, such as the LED display control, 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. The MSP430 value line utilizes 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. Similarly, all sound processing is handled by an MSP430 microcontroller. When no sound is being played, the chip can hibernate and consume very little power. Using this technique and the hardware interrupts supported by the chip line; all sound processing can be done efficiently and without any delay in time as well.

The main microcontroller decision was very important for this project as it brings the hardware LED cube, the controller input, and the software together. To allow the cube to play multiple animations and choose between several games, our microcontroller choice had to support enough flash memory to store all the code. The TM4C123GH6PM Tiva C Series microcontroller is used as the project's main microcontroller. It's a 32bit ARM based processor that uses the smaller thumb16 instruction set, thus reducing code size. The Tiva chip comes in a 64-pin package and 256 kilobytes of memory. This allows the chip to support plenty of games along with supporting features such as sound due to the high pin count. The board to host this microcontroller is a separate board to allow the project to remain modular. This board hosts the Tiva controller and the MSP430 microcontroller used for sound. This board then directly connects to the LED cube circuit.

C. External Memory Storage

Adding the extra sound features to the GameQube brought a lot of additional challenges. The biggest of the challenges was how to store large amounts of media assets to be read and played later by the microcontrollers. The simplest solution was to use an SD card and have the microcontroller communicate directly to it through the SPI interface mode that the card supports. This works out particularly well, since SD cards run at the same 3.3V voltage level as all of the microcontrollers used in the project. The only obstacle was how to read and address different files on the card. A file system could have been implemented, but was ultimately rejected due to the bulkiness it would have added to the software. Instead, a simple interface was designed in which file start and end address would be hard coded onto the microcontroller so that it knows where to read the media.

D. Memory Registers

This project involves different types of logic devices and small logic circuits. One of the necessities that come 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 will help tremendously in this project for many aspects such as keeping track of any large buffers that may be used for the volumetric display.

The type of register used in the GameQube is what's known as a FIFO register (First In First Out). This is very much like a 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 reduction in the number of pins to output data. This also means that the interface to FIFOs of different memory capacities is almost always the same. Because of this, small FIFO register chips can hold a large amount of data without getting ridiculous in how many pins there are and be swapped out later for larger capacities if it is necessary. 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 makes 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 them as would be the case in a direct communication link.

E. Integrated Circuits (ICs)

One of the problems faced in designing the volumetric display for this project is 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. The IC used to solve all these dilemmas is the TLC5940 LED Driver.

This chip allows for up to 16 outputs rated at an impressive max current level, which will drive the LEDs with more than enough power. One of the things that make 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 provides 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.

F. Game Controller

The game controllers used for input to the GameQube are GameCube controllers. Two variants have been tested and currently work: an official Nintendo wired GameCube controller and a Mad Catz wireless controller. The wired and wireless controllers both work identically. The wireless controllers work using an RF receiver, allowing it to be used without any additional software configuration.

The controllers have 6 buttons, two triggers, a D-pad, and two joysticks. The D-pad is used for static directional control while the joysticks allow for more dynamic movement and acceleration.

The controllers plug into a GameCube controller hub extracted from a GameCube and implanted into the housing of the GameQube. This hub hosts up to four controllers and allows for one power source to be plugged in and power all controllers.

G. Power Supply

The power supply is an official GameCube power supply. The GameCube AC adapter provides 12V DC at 3.25A with input directly from the wall. 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.

III. Overall Design

The overall design is shown in Fig. 1.

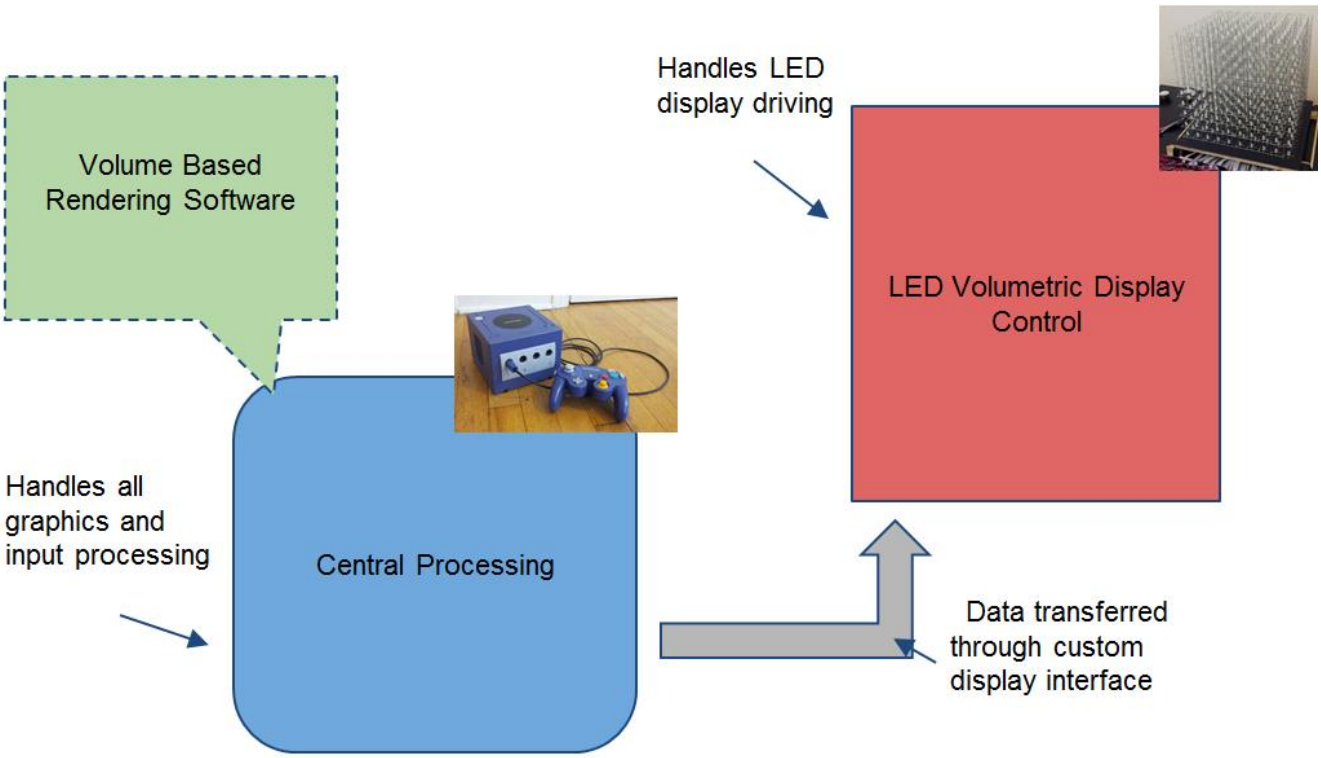


Fig. 1. Overall Design of the GameQube.

The LED Volumetric Display controls the overall display for the GameQube. This is done by multiplexing through each of the layers in the display one at a time, flashing the appropriate colors on the LEDs by controlling the LED drivers, and then checking for the next video frame. All of this is done at very high speeds so that every layer of the cube appears to be on at the same time. Central processing includes the main Tiva microcontroller board which hosts the software and sends data to the LED display. The main microcontroller board also hosts the sound control.

Additionally, the cube and its components are contained in a housing unit. The base of housing arranged in a rectangular prism made of wood. The LED circuit, main microcontroller, power control, speakers, and GameCube controller board are stored in the rectangular base. The LED cube sits on top of the base and is covered by an acrylic shell. The LED cube has a thin layer of wood as support which stands on the base and is secured using pins, allowing the cube to lifted giving access to the electronics inside.

The high level C++ software that creates the games and animations are hosted and programmed onto the Tiva microcontroller. This software runs through a graphical rendering set-up to draw and create objects on the cube. The software then breaks down the software to output 1000 bytes per frame, allowing for each LED to have a color value.

IV. Hardware Design Details

The hardware design was modeled to allow modularity between the major sections. The LED controller circuit and main microcontroller are on separate to allow for adjustments and changes that don't affect the overall design. Additionally the power control is separated to allow for modifying the power that dissipates through the cube.

A. LED Cube Design

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

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 also helps to perform any sort of color correction at a hardware level. There will then be three LED driver sections each with 7 TLC5940s, 10 sockets for ribbon cables, with 10 pins connectors, and 12 miscellaneous pins which can be used to drive additional LEDs.

To multiplex the layers of the display, the MSP430F5529 is implemented in the design with a clock frequency of 25MHz, closely matching the TLC5940 max serial data transfer speed of 30MHz. 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. The display goes through the cycle of flashing all 10 layers at a rate of 300Hz. In order to receive video data from the main processor, a separate shared memory buffer is used to allow an asynchronous data transfer to occur between the two devices.

The video data stored in the external RAM 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. Figure 2 below shows the LED control design.

Finally, a special interface was designed for the main microcontroller to be able to send video data to the display. There are 10 pins in this interface, 8 of which are used to send an entire byte of data at a time. The ninth pin is used to clock the data to the display and the last pin is used to keep everything in sync. When the display is done processing a frame of data it pulls the sync pin low for the main microcontroller to know to send new data. Since the display checks for new video data at a constant 60Hz, the sync pin can also be used by the main microcontroller to maintain a 60Hz frame rate. Also, because the displays refreshes at 300Hz it also supports frame rates of 60Hz, 50.8Hz, 43.6Hz, 38Hz, 33.9Hz if the main microcontroller can't run fast enough for 60Hz.

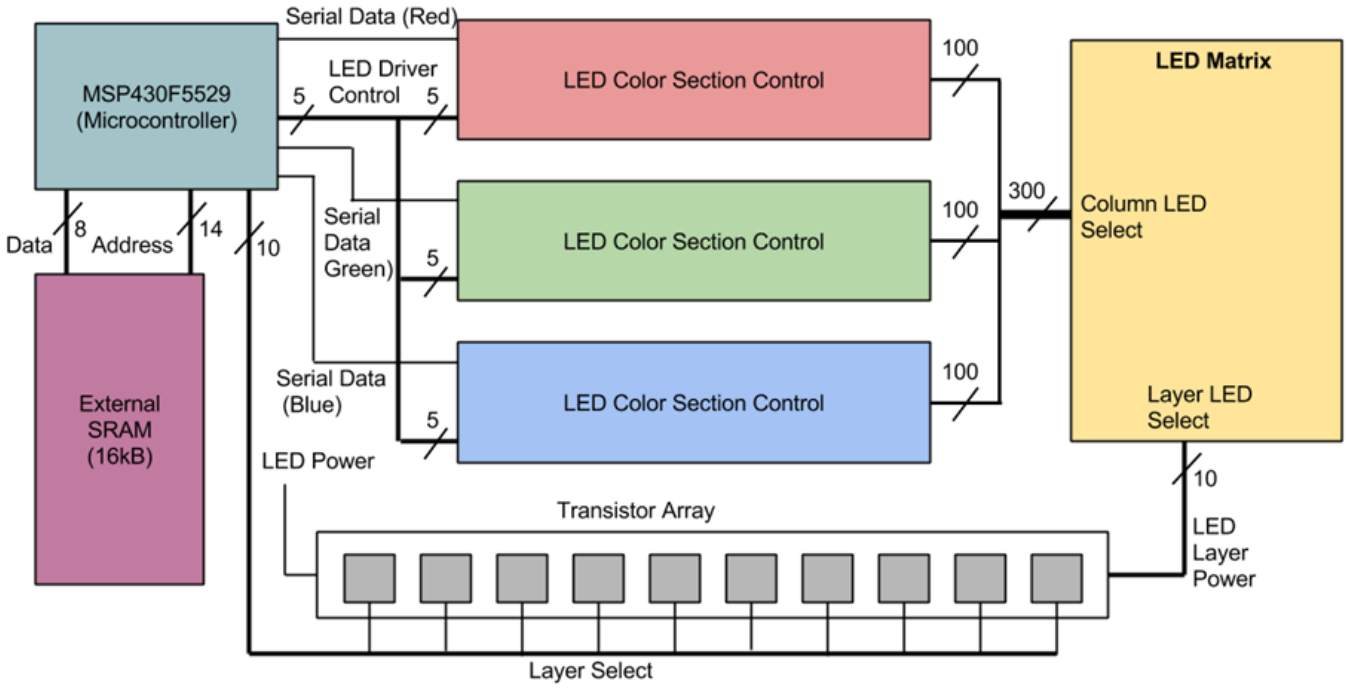


Fig. 2. LED Control Design.

B. Microcontroller Design

Connected to the 10 pin interface is the main microcontroller board. This board is centered on the Tiva TM4C123GH6PM. The chip is configured with decoupling capacitors and an external oscillator for stable power and timing. One port of pins is reserved for the 10 pin display interface. With 64 pins, and 39 available for GPIO, the Tiva proved to be a good choice as many more pins ended up getting used than originally planned. Three pins are configured for sound input and three other pins are configured for sound output.

The input syncs with the source of the sound files, the MSP430G2553. Sound is produced on the main board by using PWM to work as a digital to analog converter. This output is sampled at 32 kHz to produce audio. Large music sound files are stored and read from an SD card, whereas short sound effect files are stored in the extra program space of the chip as constant arrays. This allows for both music and sound effects to be mixed at the same time to produce a single audio stream. There are two separate PWM channels for a distinct left and right channel. Input audio is routed through an op-amp and mixed with the synthesized sound in an analog fashion. The final mix is then routed to the ADC of the main microcontroller for visualization and then output to the speakers as well.

C. Power Design

The MSP430, Tiva microcontrollers run at 3.3V as well does the GameQube. The GameCube controllers take in 3.3V and 5V. To allows 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 3.3V line goes to the main microcontroller board which provides power to the LED circuit.

D. Input Design

The GameCube controllers use 5 pins: two for power, two for ground, and one bidirectional data line. Our design incorporates the GameCube controller hub that provides a single source of power, allowing the main microcontroller to only use four pins for controllers. Each pin is mapped to one of the bidirectional lines for each controller. While the software only supports two player games, four player controls is possible. The hardware configuration for the GameCube controllers is very simple. The burden of getting the GameCube controllers to work correctly is put on the software configuration.

V. Software Design Details

The main goal of the software is 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

A. User Interface

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, shown in more detail in Fig. 3.

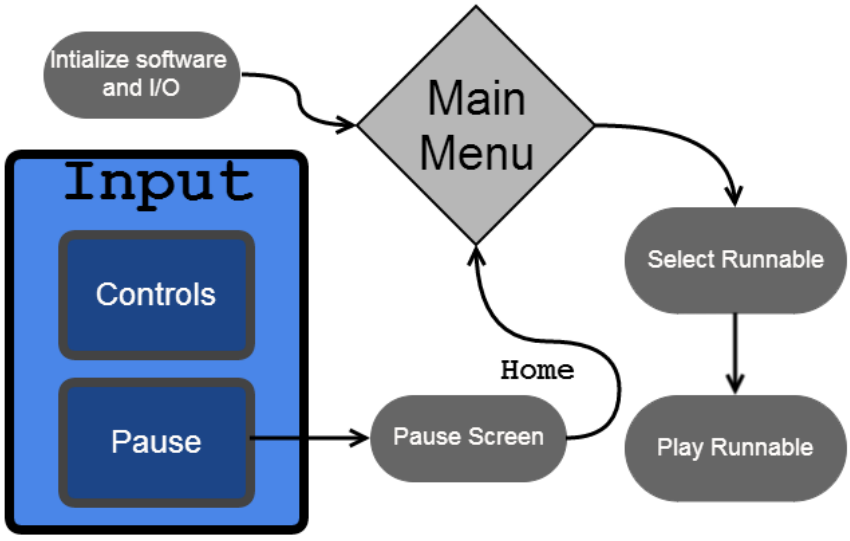


Fig. 3. The GameQube user interface flowchart.

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.

B. Games and Animations

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.

The final game implemented on the GameQube is "Invaders", based on the classic game "Space Invaders". A player starts off as a spaceship represented as a rectangle at the bottom of the LED cube. Nine aliens, represented as cubes at the top of the LED cube move back and forth, and eventually downwards. The goal is for the spaceship to kill all aliens by shooting an LED bullet and hitting them. The spaceship has to kill all spaceships before one of them shoots the spaceship or one of them reaches the spaceship at the bottom.

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.

C. Embedded Software

A lot of code needed to be written on the embedded level to control the microcontrollers at their basic levels. Whereas most of the software focuses on the overall processing of data for the project as a whole, there still needed to be the lower level written to denote exactly how pins would be utilized or clock speeds would be run.

For the LED display microcontroller this involved setting up the IO pins as well as initializing timers and hardware SPI features. The main oscillator was set to 25MHz, but the PWM clock only runs at half the oscillator speed, whereas SPI runs at the full clock speed. The timers that were initialized are used to maintain a constant time interval when multiplexing through the layers of the display. Gamma tables were also created as a quick way of decoding a byte color into a byte value for each of the three primary colors. These table values were also generated to take gamma into consideration as well as maintaining constant overall brightness of the LED while different colors are mixing.

The main microcontroller features most of the high level code but also required some careful control over the low level features. The max heap and stack size had to be carefully decided and set for the processor as well as enabling of some of the peripheral features like ADC. The biggest challenge was writing the low level code to interface with the input controllers. The timing is very critical for the projects chosen controllers and any details involving clock speed configuration could ruin the timings. Sound was another aspect that needed to be programmed carefully at the lower embedded level. The interrupt based interface needed to be understood and communicated through two very different chips.

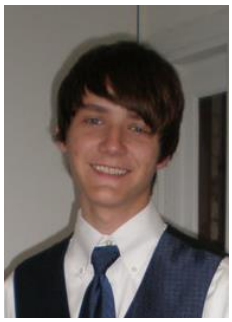
The sound driver microcontroller needed to have it's hardware interrupt system setup. The embedded code also involved programming the interface between the microcontroller and the SD card. This involved hard coding the different types of messages that could be sent or received. One of the more clever embedded level programming techniques used for sound was to take advantage of the extra program space and use it as additional ROM to store media assets. Storing small audio samples this way allowed for real time mixing of different sounds, since reading from the chips ROM takes up virtually no extra communication time as opposed to adding an extra SD card.

VI. Conclusion

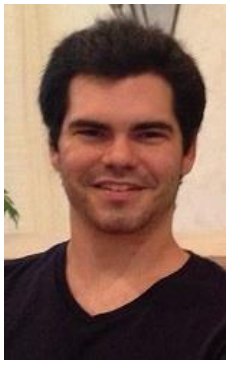
Through completing this project, we have gained valuable experience and knowledge that will no doubt assist us in the future. Our original vision of an LED cube for games grew into an entertainment system used for games, animations, and sound visualizing. With the help of our education and resources 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.

Biography

Omar Alami is a graduating Computer Engineering student at the University of Central Florida. He is working part-time as a Software Engineer at The DiSTI Corporation. Omar plans to transition into full-time upon graduation and pursue a Master's in Computer Science part-time at UCF beginning Fall 2014.



Stephen Monn is a student at the University of Central Florida graduating with a Bachelors Degree in Computer Engineering. He currently works as a student for Lockheed Martin and will continue working for the company in a full time position after graduation.



Matthew Dworkin is a senior at the University of Central Florida. He plans to graduate with his Bachelors of Science in Computer Engineering in May of 2014. After graduation, he wants to start a career in the Orlando area working as a software developer.