Universal Game Controller

By
Evan Lee
Neil Singh
Charles Van Fossan

Final Report for ECE 445, Senior Design Spring 2017
TA: Eric Clark

May 2017
Group 48

Abstract

For any avid gamer, having multiple systems are a must, but the expense of having multiple controllers for each might create upkeep costs and clutter issues. Our Universal Gaming Controller addresses this problem by connecting to a Bluetooth console dongle that is able to connect to different types of consoles. By having a single controller that could work on multiple systems, the cost of being able to utilize the console the way it is supposed to be greatly diminishes. Our controllers should not exceed the cost of a normal commercial controller and the various dongles should be as inexpensive as possible so that buying new dongles does not have a large financial impact on the consumer. This way, consumers do not need to worry as much about the cost of getting the maximum amount of controllers for their consoles.

Contents

1 Introduction

1.1 Objective

With the advancement of technology, the world of gaming is continuously expanding. This can be observed just by looking at the technical specifications of video game consoles over the years. One of the earliest video consoles, the Nintendo Entertainment System, started the industry out with a CPU that ran at 1.79 MHz [1]. When comparing it to the modern day version of the console, the Wii U, which clocks in at around 3 GHz, the difference is astounding [2]. Today, owning multiple gaming consoles can allow an avid gamer to experience a wide variety of games from different generations. While this is great for the progression of gaming, having new hardware released every couple of years starts to have its monetary impact.

One of the largest problems gamers will experience is the abundance of gaming controllers; having multiple controllers for each gaming console can cause significant issues regarding upkeep cost and clutter. Moreover, the latest generation of console controllers all have relatively similar shapes and button/analog stick layouts but with small, subtle differences. This can make having different types of controllers seem redundant and unnecessarily expensive while also causing some gamers who are comfortable with a certain controller layout problems adapting to using another type of controller.

1.2 Our Solution

Our goal is to eliminate these problems by providing a universal game controller, or UniCon, that connects to many different types of gaming consoles. This eliminates the need for the storage of the plethora of controllers for the various consoles and also the adaptation period of using a new console. Moreover, this ultimately allows consumers to only have to purchase one type of controller that can be used with all of their consoles and thus saving them money. We plan to develop a single controller that can be connected (wirelessly over Bluetooth) to various dongles, which then can be plugged into the console themselves. In addition, we also intend to create an application for mobile devices such that the user can define custom mappings of the universal controller's buttons to the specific console's. For the scope of this class, the only two consoles that will be supported are the Nintendo GameCube and Nintendo 64, but hopefully in the future more gaming consoles such as the Xbox, PlayStation, or Wii U could be supported.

1.2 High Level Requirements

- Controller must control the console the same as commercial controllers for the GameCube and N64 consoles

- Controller must work wirelessly to connect to dongles that plug into the consoles
- Controller must be able to have custom button mapping profiles which are able to be set from another device such as a laptop or smartphone

1.3 Block Diagram

As shown in Figure 1, there are two main components to this project: the controller and dongle. The controller will be powered by an onboard rechargeable battery and have a microcontroller unit (MCU) that will process all I/O between the physical buttons, vibration motor and the Bluetooth adapter which will likely be built into the MCU. The dongle will have a similar MCU, powered by each console and data output through a single data pin. Lastly, there will be an application interface between the controller MCU and a smartphone that will allow the user to reprogram the mappings of each button.

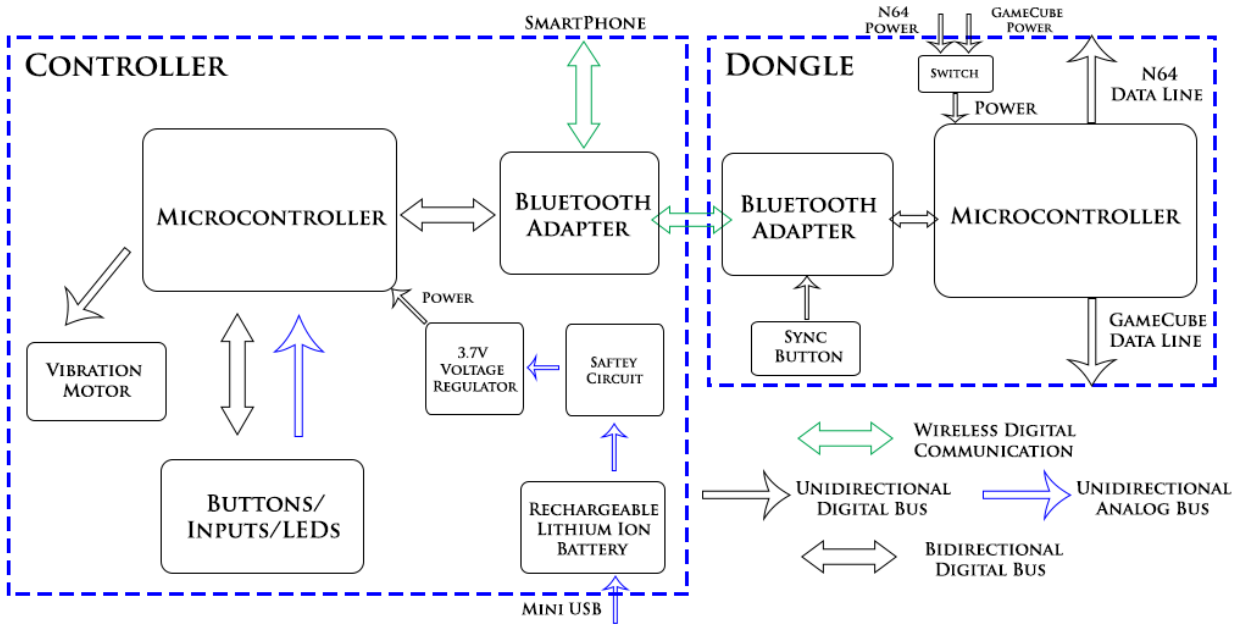


Figure 1: Block Diagram

2.1 Controller

2.1.1 Lithium Ion Battery and Charging Circuit

The power supply for this controller will be a 3.7V single cell Lithium Ion battery capable of storing 1300mAh of energy. This power supply was taken from the original Wii U Pro controller that we have repurposed for this project. Because this is a Lithium Ion battery, we will take extra safety precautions when dealing with charging and designing circuits using the battery (see Ethics and Safety section). In order to operate safely, a single cell, 3.7V battery must remain in the range of 3.0 to 4.2V. Any usage outside of this range poses a serious safety risk to the user.

For our controller's battery, we implemented a MAX 1555 charging integrated circuit that takes in a DC or USB power source. This charging IC allows us to use a maximum 7V power source for a DC supply and a maximum 6V power source for an USB supply [3]. This charging IC was selected because it not only helps safely charge the battery, it also does not allow the battery to be charged over 4.2V. Once the battery voltage goes over 4.2V, the MAX 1555 will cut off the battery from the power source.

2.1.2 Under voltage Protection Circuit

As mentioned in the above section, the MAX 1555 charging IC ensures that the lithium battery is not charged over 4.2V. However, in order to ensure safe usage of the battery, we had to create a protection circuit that would cut the battery off from the load in the event the battery is discharged passed 3.0V.

We implemented our under voltage protection circuit using a LT1495 OpAmp, a LT1389 1.25V shunt regulator, a PMOS transistor, and six resistors of varying resistance. Figure 2 shows the circuit design implemented in LTspice, a circuit simulation software. The cutoff voltage, in our case 3.0V, is determined by the ratio of Resistor 1 and Resistor 2, which are set at 3.656 MΩ and 3 MΩ. This under voltage protection circuit is connected serially to the MAX 1555 charging integrated circuit so that together they form the upper and lower limits of the battery's usage.

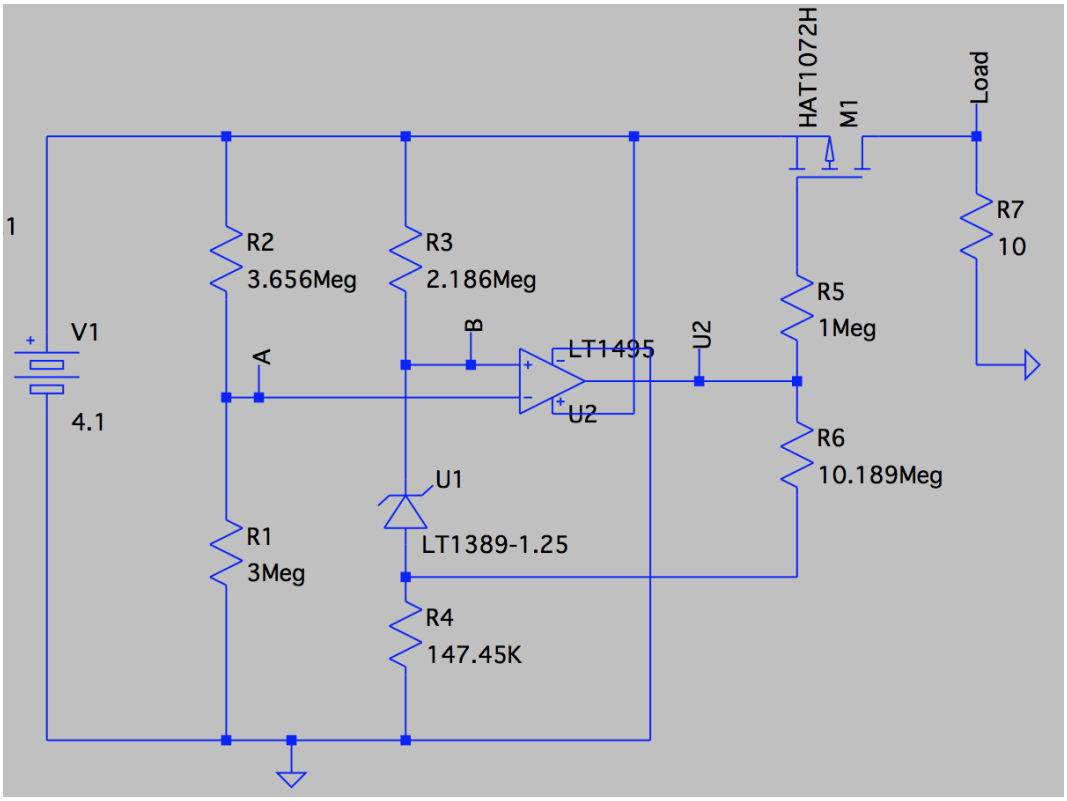


Figure 2: Under voltage Protection Circuit

2.1.4 Microcontroller

The MCU for the controller handles the interpretation of the inputs from the physical controller and relays the information to the Bluetooth adapter when requested. It also allows the Bluetooth adapter to be connected to either the dongle or the smartphone. In the case of a connection to the smartphone, the microcontroller enables the controller to have custom profiles dictated by the smartphone.

We chose to use the TI CC2650 RGZ as our controller and dongle microcontrollers. It contains 30 GPIO pins for interfacing with the buttons and is clocked at 48MHz so it is able to support microsecond precision. Figure 3 shows the schematic of the CC2650 and the numerous GPIO pins connected to the button inputs. We not only chose this microcontroller for its low power consumption and low cost, but also because it contains a Bluetooth controller built into it as well. We looked at other Bluetooth microcontrollers, but none that we could find had as many GPIO pins as the CC 2650 RGZ model. We used a JTAG connection to program the microcontrollers from a CC2650 Launchpad Development Kit we purchased.

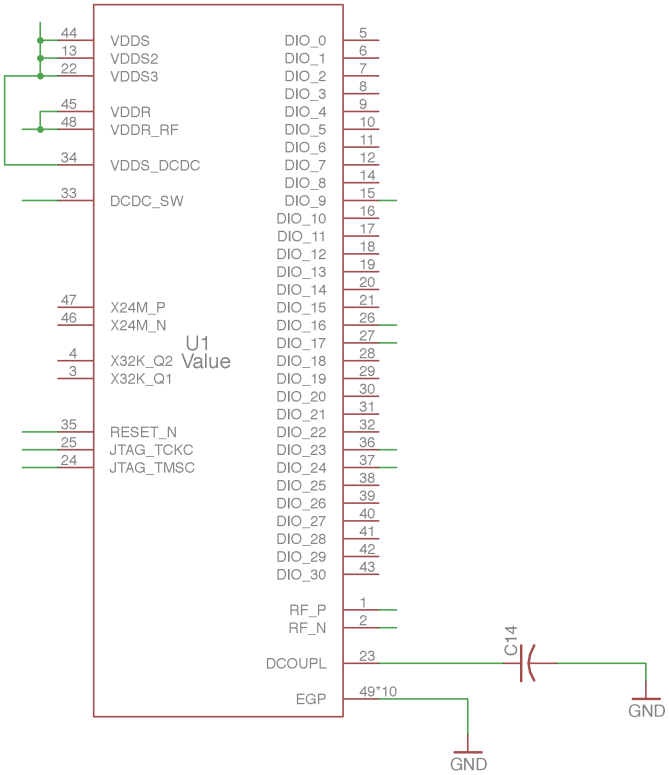


Figure 3: Controller Microcontroller Schematic

For our purposes, we required this microcontroller to be able to relay the informational button states of the controller to a connected device within five video game frames, which is around 83 ⅓ ms. This is to ensure that there is no noticeable lag between a button press and the corresponding action in the video game.

2.1.5 Bluetooth Adapter

The purpose of the controller's Bluetooth adapter will be two-fold. First, it will be used to receive information from the dongle and transmit information regarding the states of the controller's buttons and the position of the joysticks to the paired dongle to respond to the console's poll. Second, it will used to receive button mapping settings from a smartphone. Because the controller's Bluetooth will be connected to multiple Bluetooth devices (dongle and smartphone), the controller's Bluetooth adapter will act as the master in the piconet. We will set the Bluetooth power to be 2.5 mW to have a 10 meter radius, which we think will be adequate for our purposes. For our purposes, we require the Bluetooth adapter to function within five meters of the console dongle, which is a typical distance between player and screen.

2.1.6 Buttons/Inputs

We will be using the Nintendo Wii U Pro Controller for our controller shell. This controller was selected because of its similarity with most modern day controllers, its large number of button/analog stick inputs that could be mapped to most controllers in use, and because of its relative low price compared to other modern day controllers. This controller was also selected to keep with the Nintendo theme of the entire project.

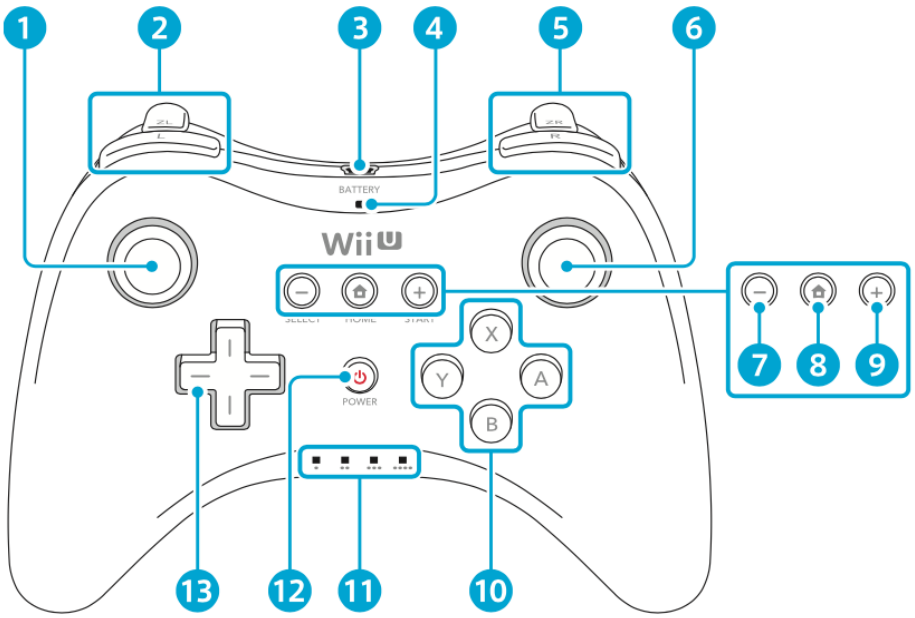


Figure 4: Front of Controller [4]

All the inputs feed into the microcontroller, separated by only either a potentiometer or transistor switched by a digital button.

To avoid button bouncing, we will sample the inputs frequently and perform software debouncing within the microcontroller. If the state of the button is constant for various samples in a row, we know that the

button is not bouncing. However, if the button state changes every sample, the button most likely is bouncing. Because of this, the button will only be treated as being pushed if it is not bouncing.

The left analog stick (1 in Figure 4) and the right analog stick (6) will each be connected to a dual axis potentiometer that adjusts two voltages based on how the analog stick is moved, as well the push button on each analog stick connected to digital input pins. The ZL trigger (top of 2) and the ZR trigger (top of 5) will be connected to single axis potentiometer that will adjust a single voltage based on how far down the trigger is. The directional pads' (13) four directions will be connected as digital buttons, along with the A, X, Y, and B buttons (10), the select, start, and home buttons (7, 8, 9 respectively), the L and R buttons (bottom of 2, bottom of 5), and power button (12). The LED's (11) will be connected to digital output pins on the microcontroller, separated by a resistor. The battery LED (4) will be connected to a multi-colored LED, separated by a resistor.

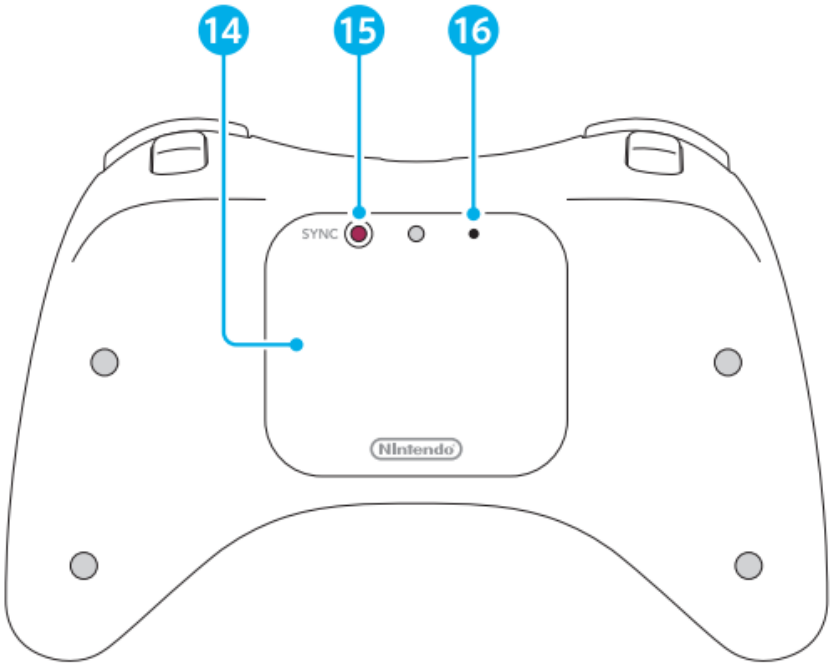


Figure 5: Back of Controller [4]

The sync button (15 on figure 5) will be connected by a digital button to the microcontroller, as well as the reset button (16).

2.1.7 Motor

In many games, a vibrating controller is used to enhance a player's gaming experience. For our controller, we used a repurposed motor that came with the original Wii U Pro controller. We designed and implemented a circuit that allowed a microcontroller unit to toggle the motor on and off with a very small amount of current. We required that the motor circuit must allow the motor to be turned on when a connected microcontroller, specifically the CC 2650 described above, applied a voltage to the circuit

using less than 8 mA of current. The 8 mA was a strict limit as the CC 2650 MCU can only provide 8 mA of current per GPIO pin.

To implement the motor circuit, we used a 1N4001 Diode and a TIP120 transistor. Figure 6 shows how the motor is connected to these components and to the battery. The MCU GPIO input is connected to one pin of the MCU and is the input to the motor circuit. If the MCU provides a voltage at that pin, the motor will begin to vibrate.

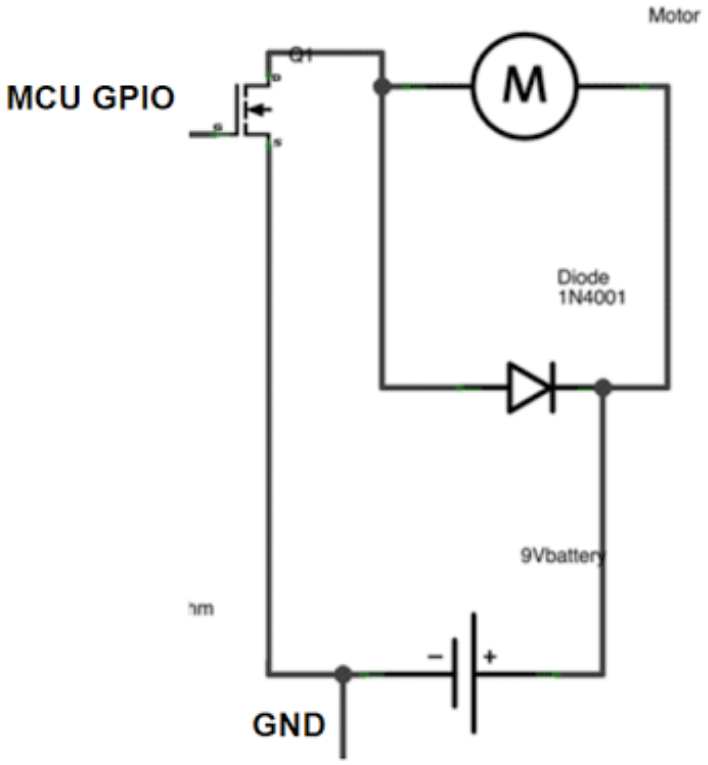


Figure 6: Motor Circuit [5]

2.2 Dongle

2.2.1 Power Switch

The dongle will be able to plug into both a Nintendo 64 console and a Nintendo GameCube console, so the microcontroller must be able to tell which hardware communication protocol to use and which data lines to write to. In order to configure the microcontroller to a certain console, we will be using a physical switch on the dongle. One position of the switch will indicate that the dongle is connected to a Nintendo 64, while the other position will indicate that it is connected to a Nintendo GameCube. This switch was also necessary to provide protection to the rest of the circuit in the event an user connects the dongle to both the Nintendo 64 and the Nintendo GameCube.

2.2.2 Microcontroller

This microcontroller will handle button and joystick information from the controller (from the Bluetooth adapter) and will transmit this information directly to the console. To do this, the microcontroller keeps track of the controller button and analog stick states. When a button is pressed on the controller, the dongle microcontroller will receive this information and update the button states. When polled for information by the console, the microcontroller will send these button states to the console using the bidirectional data pin. In order to do this, it will need to map the controller information to the appropriate communication protocol based on what console it is connected to. The microcontroller must also use the same console communication protocol to receive polls and other requests from the console and forward this information to the controller. Based on the position of the switch, the microcontroller must be able to use the corresponding console communication protocol.

In order to have a functioning controller, we required this microcontroller to be able to respond to the Nintendo 64 and Nintendo GameCube polls for information within 16 ⅔ ms. This is the timeout rate for these particular consoles, so a console will think that a controller has been unplugged if it does not receive a response to its polls within that time frame.

Again, we chose to use the CC 2650 RGZ microcontroller. Unlike the controller, the dongle did not need approximately 30 GPIO pins, but we chose to use this microcontroller for continuity as well as for its low power consumption, microsecond precision, and low cost.

2.2.3 Bluetooth Adapter

This Bluetooth adapter will be used to send polls and other useful information to the controller and to receive button and joystick information from the controller. As stated above, the dongle must be able to connect with the controller's Bluetooth adapter, but will act as a slave device in the piconet. We again require the Bluetooth adapter to maintain a connection to the controller within 5 meters of each other.

2.3 Printed Circuit Board

Because we were repurposing the shell of a Nintendo Wii U Pro controller, we had to design a PCB that would fit in the shell with all the buttons and analog sticks lined up. This required precise measurements of the original PCB that came with the controller and careful design of the new PCB using the EAGLE software. Figure 7 shows the PCB in the EAGLE software. Notice the precise placements of the button pads as to match up with the original PCB and controller design. The PCB of the dongle was much simpler to design as it did not have any shape constraints. To use wireless Bluetooth communication with the microcontrollers, we also had to design an impedance matching circuit that would match the load of an antenna. Figure 8 shows this impedance matching circuit for a 50 Ω antenna.

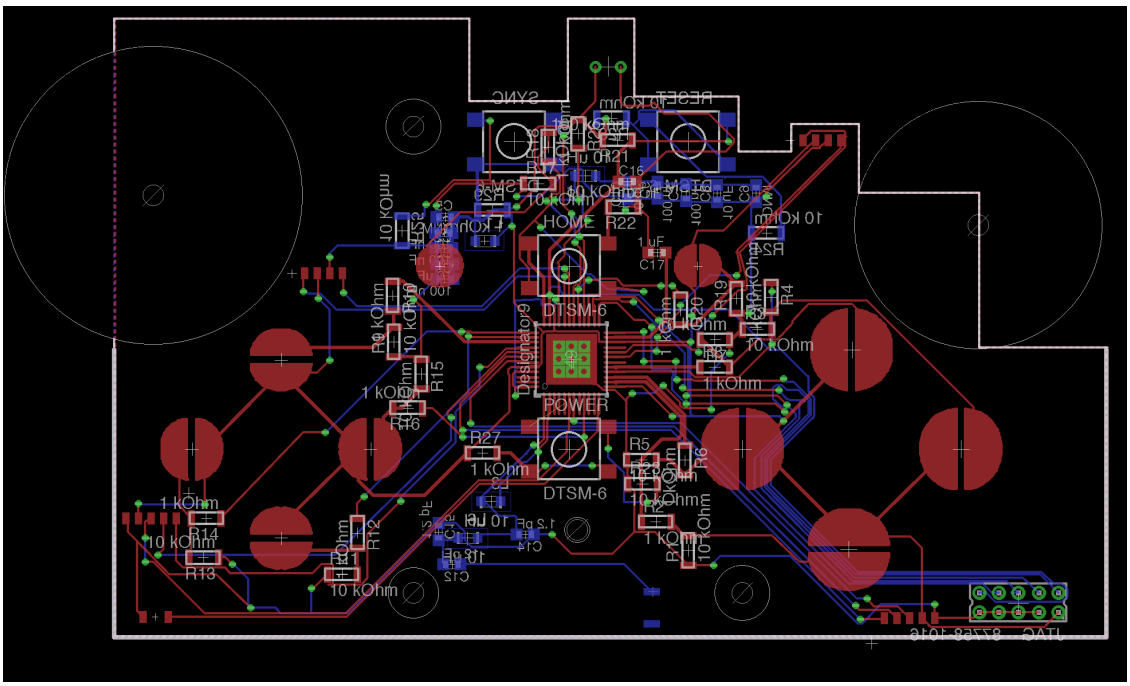


Figure 7: Controller PCB

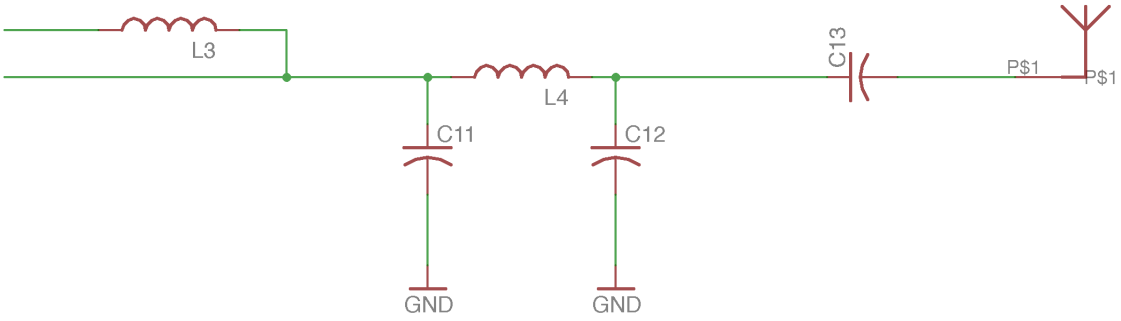


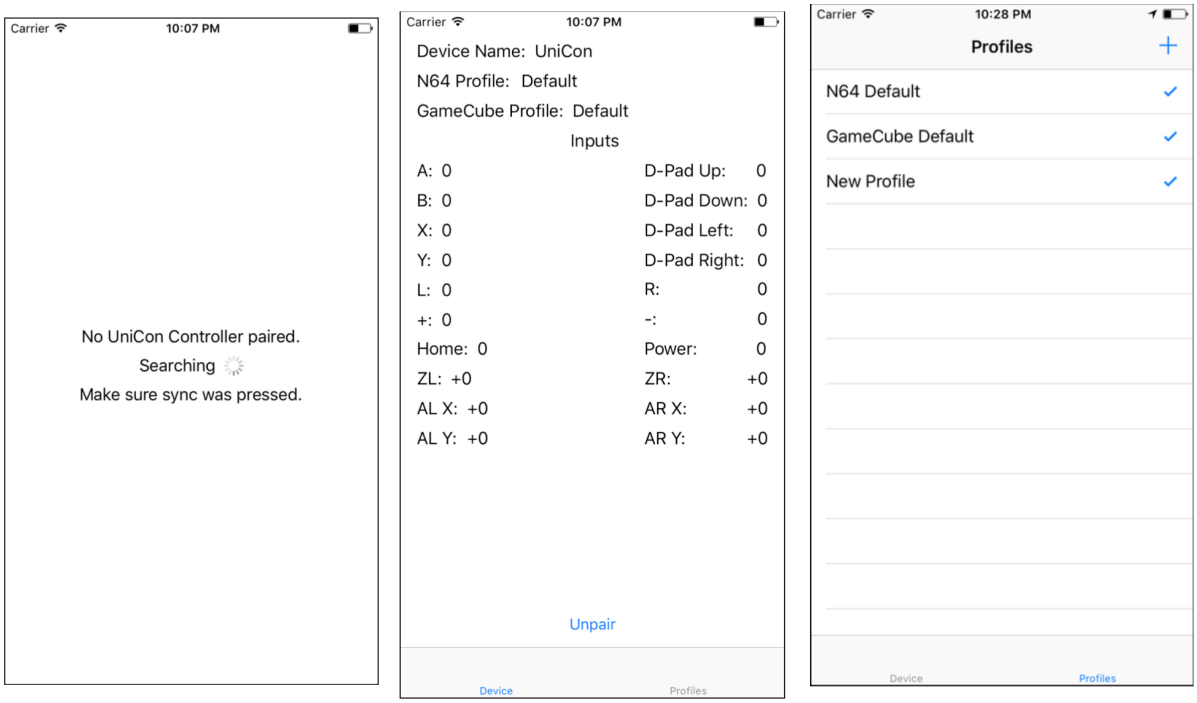
Figure 8: Impedance Matching Circuit

2.4 Software

2.4.1 iOS Application

This iOS application is used by the user to create custom button mapping profiles and to upload profiles to the controller. We chose to use an iOS application written in Swift because of our familiarity with the iOS software and the Swift programming language. We considered creating an Android application in Java as well, but due to time constraints we only created the iOS version.

The iOS application starts by searching for the UniCon controller. If it does not find a controller after five seconds, it displays the message “Make sure sync was pressed.” to the user (Figure 9).



Figures 9, 10, and 11: Searching for controller, Device information screen, Profile screen

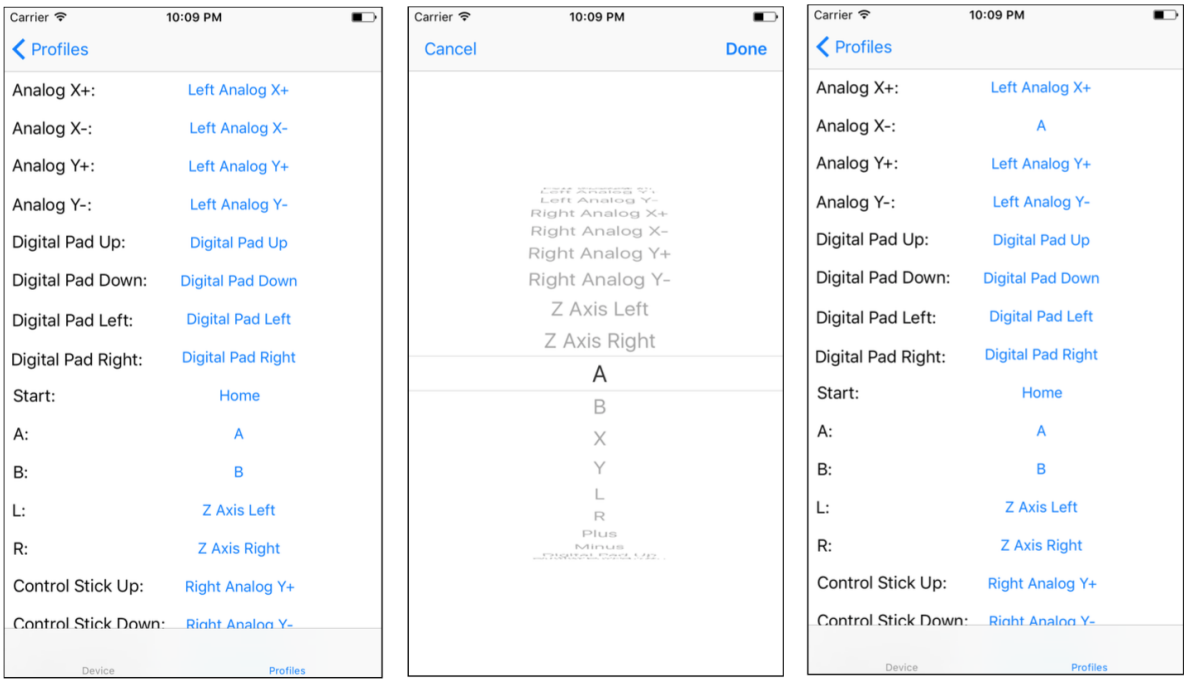
Upon discovering the controller, the application connects to it and registers for device state updates via Bluetooth. The application then allows the user to switch between two tabs: one that gives device information, and one that allows the user to select button mapping profiles. From the device information screen, the application displays the controller name, the profiles it is currently using, and the state of each input, which is continuously updated (Figure 10). One may also unpair their controller from the application from the device information screen if they wish. The tab bar at the bottom is missing images to designate the device and profile tabs for now.

As shown in Figure 11, the other tab is the “profiles” tab, which allows the user to manage his or her profiles for custom button mapping. A list of all the profiles the user has created is displayed with a checkmark next to an entry if it is the current profile being used (there is one for the N64 and one for the GameCube).

To add a new profile, the user simply needs to click on the plus button at the top right of the home screen and they will be presented with a popup dialog. From there, the user enters the name of the new profile and specifies whether it is a N64 profile or a GameCube profile. If the user enters an empty profile name or a name that is already in use, the application will let them know of the error. Finally, the user can press “Done” to finish adding his or her new profile. The new profile will be added to the list of profiles.

When a user clicks on a profile to edit it, they will be presented with a screen that lists the button mappings that the profile uses (Figure 12). By clicking on a specific button, the user has the option to specify the new mapping for the corresponding input. In Figure 8, the button corresponding to “Analog

X-” is clicked, which then leads to Figure 13. In this screen, the user is allowed to choose from any of the UniCon controller’s inputs to assign to the button mapping. Once the user presses “Done”, the profile is updated with the new value (Figure 14).



Figures 12, 13, and 14: Add profile screen, Updated profile list screen, Button mapping

2.4.2 Microcontroller

The software of the CC 2650 microcontrollers was split into two parts for the controller and dongle. Both parts were developed and tested on a TI CC 2650 Launchpad Development Kit.

The controller portion of the software was relatively simple to implement. The microcontroller would scan the GPIO pins connected to the button or analog stick inputs and send out Bluetooth messages updating the dongle’s microcontroller whenever a button status was changed. The MCU also received information from the dongle’s microcontroller, such as when to toggle the motor.

The dongle MCU’s software was much harder to implement. First and foremost, we had to be able to read and write to the consoles’ data pins. This included implementing the console controller protocols (described in the following sections), which was not an easy task. The microcontroller needed to be able to detect when command or poll was sent to the dongle by carefully reading the data line for a start bit and interpreting the message to create a response. We had to implement an error detection algorithm that allowed the microcontroller to miss a bit of information from the console and still be able to correctly identify the command type. When the microcontroller detected that a poll or request was sent, it would trigger an interrupt that would then handle the message.

2.4.3 Nintendo 64 Controller Protocol

The physical connection for a Nintendo 64 controller consists of three pins: Ground, Bidirectional Data, and Power. The power required for the controller to be used is 3.3V [4], which it receives from the console.

Both the console and the controller can write to the data line. To do this without a tri-state buffer, the N64 console implements an open collector scheme to write to the data line. Both the console and controller are connected to the input of the open collector and can either drive the line low or drive nothing using high impedance. The output of the open collector is connected to a pull-up resistor. If neither the console or controller are driving the data line, the output of the data line is pulled up to high. If either the console or the controller drive the data line low, the output of the open collector is low. Neither the console or the controller can drive the data line high.

Both written data and a clock is transmitted through the Data pin through a self-clocked signal at a rate of 4 nanoseconds per bit. To represent a high byte, the data pin will be pulled low for 1 nanosecond, then high for the next 3 nanoseconds. Similarly, a low byte is represented by the data pin being pulled low for 3 nanoseconds and high for 1 nanosecond. When no bits are being transmitted, the data pin remains high, but as soon as the data pin goes low, transmission starts.

To interact with the controller, the console sends a command byte to the controller and then the controller will respond with its response. Below is a summary of the commands the console may send to the controller.

Command Byte	Summary
0x00	Identify: The controller responds with 3 bytes to identify characteristics about itself.
0x01	Data: The controller responds with 4 bytes of button and joystick data.
0xFF	Reset: The controller resets its internal registers and responds with its identification bytes.
0x02	Read: Read from the controller pack memory space. The controller responds with the 32-bit value stored in memory at the address specified by the operands of the command.
0x03	Write: Write to the controller pack memory space. The controller writes the 32-bit value at an address, both of which are specified as the operands of the command. The controller responds with a data CRC.

Table 1: Summary of N64 Commands [6]

The rumble motor of the controller is attached as a rumble motor pack. To initiate a rumble, the console must write an odd number to any address at 0x8000 or above. To stop the rumble, the console simply writes an even number to these addresses. We will be simulating a memory pack in the dongle's microcontroller software to support the "Read" and "Write" commands, as well as rumble.

2.4.4 Nintendo GameCube Controller Protocol

The GameCube controller protocol is very functionally similar to the Nintendo 64 protocol, although surprisingly a bit simpler. They both operate at the same frequency and in the same physical manner (using a 1K pull up resistor [5] to drive the data line). The only differences are the commands the console sends and the responses the controller gives. Outlined below are the different commands the console can send with a summary of the function they perform.

Command Bytes	Summary
0x01 (9 bits)	Identify: The console probes the controller port to see if any controller is attached. The controller responds to let the console know it is there.
0x400302 (24 bits)	Data: The controller responds with 8 bytes of data describing the current state of the buttons and analog sticks.

Table 2: Summary of GameCube Commands [7]

Below is a plot of the 24 bit command word (A) sent by the GameCube console, as well as the 64 bit response (B) from the controller. It is apparent to see that a high bit is represented as 1 low bit then 3 high bits, and that a low bit is represented as 3 low bits and 1 high bit. Also it shows that when the response is finished, the line remains high to signal that there is no more data needed. This is because all bits start by going low for at least one cycle, and therefore when the line goes low, the console and controller will know that data is being transmitted.

3 Verification

3.1 Safety Circuit

Because the upper voltage limit was taken care of by the MAX 1555 IC, we only needed to verify that the under voltage cutoff circuit functioned properly. To test this, we simply connected a function generator to the input of the circuit to act as a battery. We also connected a multimeter to the output of the circuit and measured the current of the output of the circuit to a load. We then swept the input voltage from 4.0V down to 2.9V in .01V increments and observing the output current on the multimeter. When the voltage dropped below 3.0V, we saw that no current was being drawn by the load. Figure 15 is a simulation of the

circuit given a input voltage sweep run on LTspice, a hardware simulation software. It is clear that when the input voltage (shown in green) drops below 3.0V, the output voltage (shown in red) drops to zero.

Figure 15: Simulation of Safety Circuit

3.2 Controller Microcontroller

To verify that the controller's MCU relayed the button state information to a connected device within five frames (or 83 ⅓ ms), we connected the Launchpad CC2650 to one of our computers. We then wrote a quick application that simulated the dongle that printed out timestamps of the button updates it received from the controller MCU. We then quickly pressed and un-pressed a button on the Launchpad that acted like a controller button. We then verified that the time in between the timestamps of the pressed button update and the un-pressed button update was less than 83 ⅓ ms.

To verify that the MCU correctly updated the button mapping within one second of receiving a profile from the iOS application, we connected the Launchpad MCU to our iOS application. We then edited the Bluetooth communication software so that the controller MCU sent an acknowledge message back to the iOS application upon button mapping completion. We then connected a laptop with a simulation of the iOS application to the Launchpad MCU and once again edited the software to display the timestamps of the Bluetooth communication messages on a console. Finally, we sent a profile from the iOS application

simulation to the MCU and verified that the time difference between the timestamps of sending the profile and receiving the acknowledge message was less than one second.

3.3 Bluetooth Adapter

Unfortunately, we were unable to get the Bluetooth communication between the controller and dongle functioning correctly, so we were unable to complete this verification. However, if we continued this project and correctly implemented the Bluetooth Adapter, we could easily verify it worked by measuring out five meters between the controller and the dongle and using the controller.

3.4 Dongle Microcontroller

There were two ways to verify that the dongle was able to respond to console polls and requests within the timeout timeframe (16 ⅔ ms). The first was to connect the data pin to an oscilloscope and verify on there that the dongle was sending the correct responses back to the console within the allotted timeframe. The second was to just simply connect the dongle to the console and turning the console on. If the dongle did not correctly respond to the polls, the console would not recognize that a controller was connected and a message would appear on the screen saying that no controller was detected. By connecting the dongle to the console, turning it on, and not seeing an error message, we verified that the dongle was correctly and efficiently responding to the console.

3.5 iOS Application

To verify that the iOS application was able to connect with a controller within 10 seconds of turning the controller, we simply used a stopwatch and timed the amount of time needed to connect. We were able to see that the amount of time was well below the 10 second requirement.

3.6 Motor Circuit

To verify that the motor could be turned on with less than 8 mA, we simply connected the motor circuit to two function generators. One function generator acted like the battery, and the other acted like the MCU toggling the motor on or off. We set a current limit on the function generator acting like the MCU to 7.9mA and then supplied the circuit with power. By seeing the motor vibrate while only using 7.9mA of current, we were able to verify that our motor circuit functioned correctly.

4 Costs

We estimate a fixed salary of $40/hour, 20 hours/week for three people for a complete design. This leads us to the following labor cost calculation.

$$3 * \frac{\$40}{hour} * \frac{20\ hours}{week} * 16\ weeks * 2.5 = \$96{,}000 \quad \text{(Eq. 1)}$$

We will be using the following parts in our design.

Part	Cost
2 Microcontrollers (TI CC2650 RGZ)	$4.73 x 2 = $9.46
PCB (est.)	~$4.00
Assorted resistors, switches, wires, transistors (Digikey, est.)	~$3.00
Voltage Regulator (Digikey, TLV70237DBVR)	$0.50
Wii U Pro Controller (Nintendo)	$50.00
3D Print of Dongle (3D Printing Price Check)	$6.00
Total	**$72.96**

Table 3: Cost of Parts

Thus, the overall cost of developing the controller and dongle becomes $96,072.96.

5 Ethics and Safety

Our biggest safety concern in this project was dealing with the lithium ion battery that we used to power our controller. If we did not take proper precautions, we could cause serious problems such as dangerously high currents or combustion. Lithium ion batteries can cause especially large fires if used improperly. To prevent any safety violations, we closely followed the ECE 445 Safe Practice For Lead Acid and Lithium Batteries document [8].

In reference to the ACM Code of Ethics, Section 1.6, “Computing professionals are obligated to protect the integrity of intellectual property. Specifically, one must not take credit for other's ideas or work, even in cases where the work has not been explicitly protected by copyright, patent, etc”[9]. The protocol of communication between the game controllers and game consoles was not created by us, nor did we reverse engineer the protocols ourselves. We will not take credit for the technology behind the protocols.

It is also important that we are careful in regards to voltages that we are manipulating on the dongles. It is possible to cause harm to both the Nintendo 64 and GameCube through neglectful design of our devices. It is our responsibility to prevent damage to the property of others according to the ACM Code of Ethics,

Section 1.2, “Avoid harm to others. "Harm" means injury or negative consequences, such as undesirable loss of information, loss of property, property damage, or unwanted environmental impacts”[10].

6 Conclusions

6.1 Accomplishments

We are incredibly proud of what we accomplished this semester. We designed and built a custom-fit PCB that fit inside the shell of the repurposed Wii U Pro Controller. Our microcontroller software worked perfectly, as the dongle was able to correctly interpret console polls and requests and then send the appropriate response. We were able to implement fully the Nintendo 64 and Nintendo GameCube communication protocols and were even able to play games on both consoles using the dongle. Our iOS application also worked perfectly, and we were able to create custom button mappings and upload them to the controller microcontroller.

6.2 Future Work

Our only shortcoming in our project was our inability to get the Bluetooth connection working between the controller and the dongle. This was due to our oversight on the need for an external oscillator for the antenna. However, we have a very good idea of what type of oscillator we need and how to implement it into our PCB. In the future, if we were to continue working on this project, we would need to perfect the Bluetooth communication.

We believe that this idea can also be expanded to support more consoles, not just the Nintendo 64 and the Nintendo GameCube. There are different gaming console brands, such as Xbox and PlayStation, that use a similar controller that could easily be supported by our universal gaming controller. If we were to support newer consoles that use wireless controllers, we could implement a direct Bluetooth connection between the controller and the console. The difficulty of adding support for these newer consoles would be reverse engineering the controller protocols, but we believe we are up to the task.

7 References

[1] P. Diskin, "NES Documentation," in *NESDev*, 2004. [Online]. Available: http://nesdev.com/NESDoc.pdf. Accessed: Feb. 7, 2017.

[2] "Wii U System Specs," in *Nintendo Today*, NintendoToday, 2011. [Online]. Available: http://nintendotoday.com/wii-u-system-specs. Accessed: Feb. 7, 2017.

[3] "MAX1551/MAX1555 SOT23 Dual-Input USB/AC Adapter 1-Cell Li+ Battery Chargers", 2017. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX1551-MAX1555.pdf. [Accessed: 03- May- 2017].

[4] "Wii U Operations Manual," in *Nintendo*. [Online]. Available: https://www.nintendo.com/consumer/downloads/wiiu_operations_manual_en_la.pdf. Accessed: Feb. 20, 2017.

[5] "Arduino - TransistorMotorControl", *Arduino.cc*, 2017. [Online]. Available: https://www.arduino.cc/en/Tutorial/TransistorMotorControl. [Accessed: 01- May- 2017].

[6] K. Thompson, "N64 Controller protocol," 2015. [Online]. Available: http://kirrenthompson.com/?p=53. Accessed: Feb. 20, 2017.

[7] "Nintendo GameCube Controller Pinout,". [Online]. Available: http://www.int03.co.uk/crema/hardware/gamecube/gc-control.html. Accessed: Feb. 7, 2017.

[8]. [Online]. Available: https://courses.engr.illinois.edu/ece445/documents/GeneralBatterySafety.pdf. Accessed: Feb. 20, 2017.

[9] "ACM Code of Ethics and Professional Conduct," in *acm.org*, 2017. [Online]. Available: https://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct#sect1. Accessed: Feb. 9, 2017.

[10] "IEEE code of ethics," in *ieee.org*, 2017. [Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html. Accessed: Feb. 9, 2017.

Appendix A (Software Flowcharts)

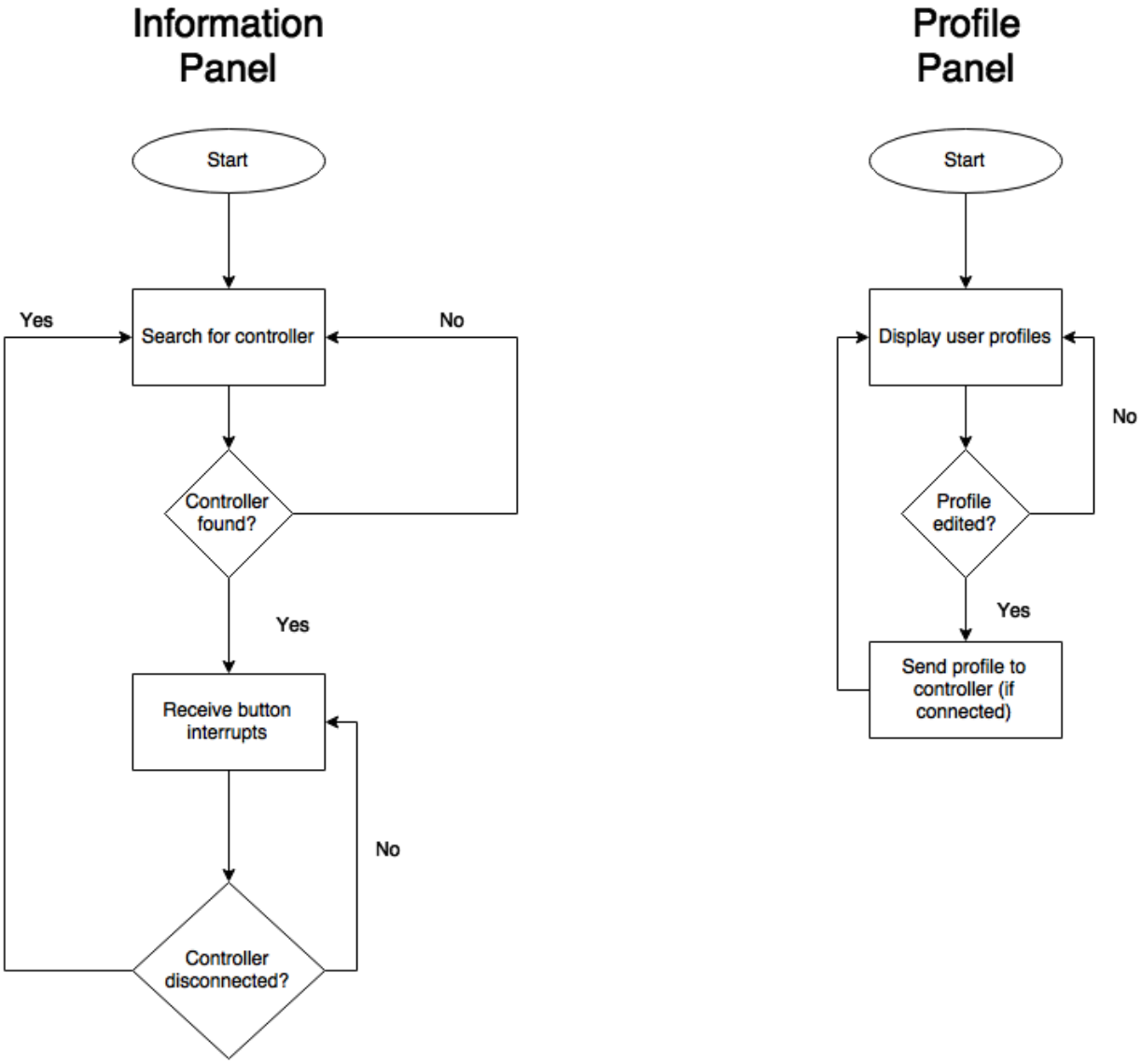


Figure 10: Mobile Application Software Flowchart

The mobile application consists of two separate entities (which can be switched through a tab view): an information viewer and a profile editor. The information viewer simply connects to the controller and then displays the button states of the controller. The profile editor lets the user input custom profile mappings and upload them to the controller (if one was connected in the information viewer).

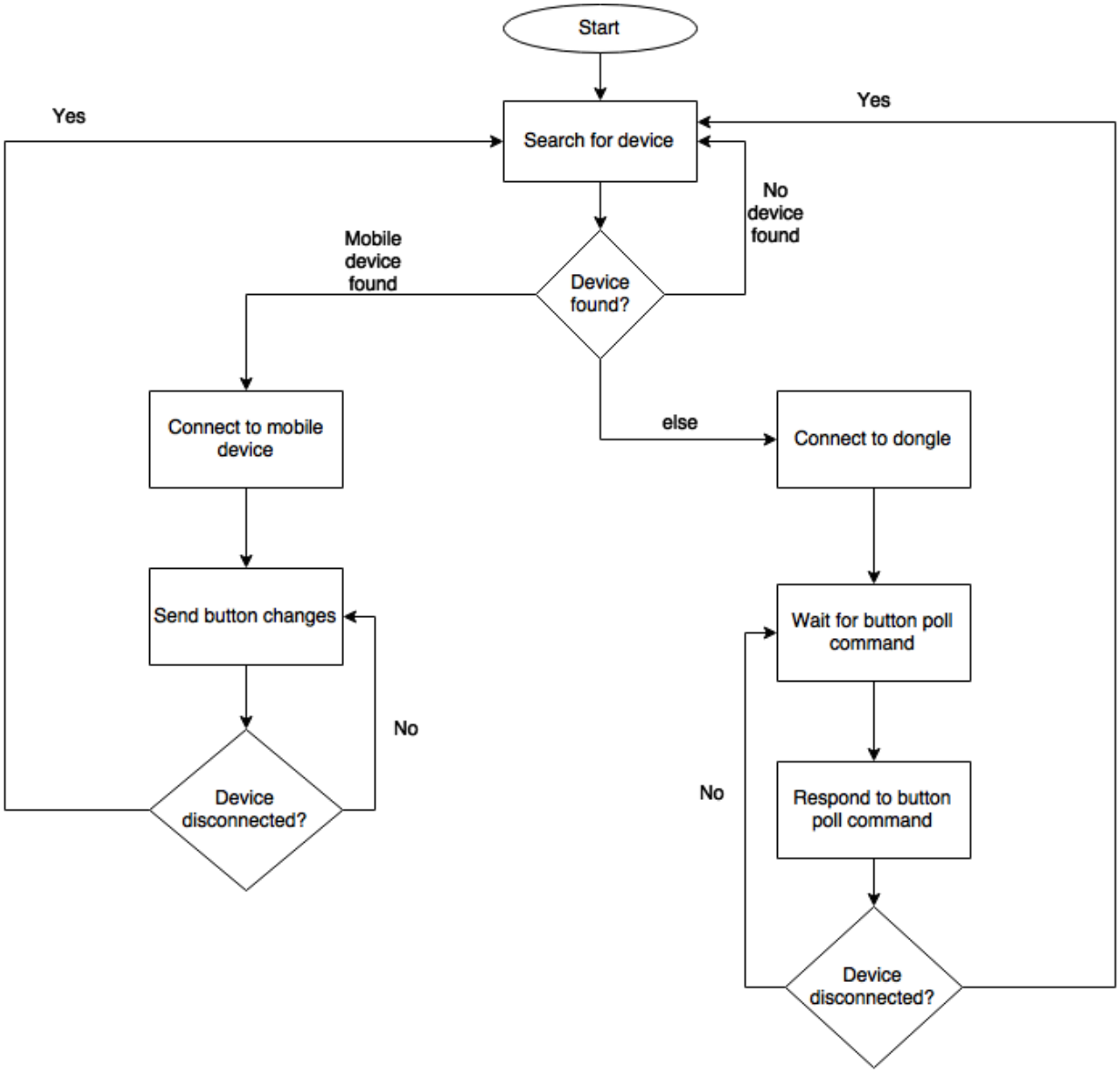


Figure 11: Controller Software Flowchart

The software for the microcontroller searches for a device and does two different things depending on whether it finds a dongle or a mobile device (it prefers to connect to the dongle when both are present). If it finds a mobile device, the controller will connect to the device and then send all the button changes that occur so that the mobile device can display that information to the user. When connected to the dongle, the controller waits for the dongle to send it a poll command, where the controller then responds with its button state. If either of the devices are disconnected, the controller goes back to search for another one to connect to.

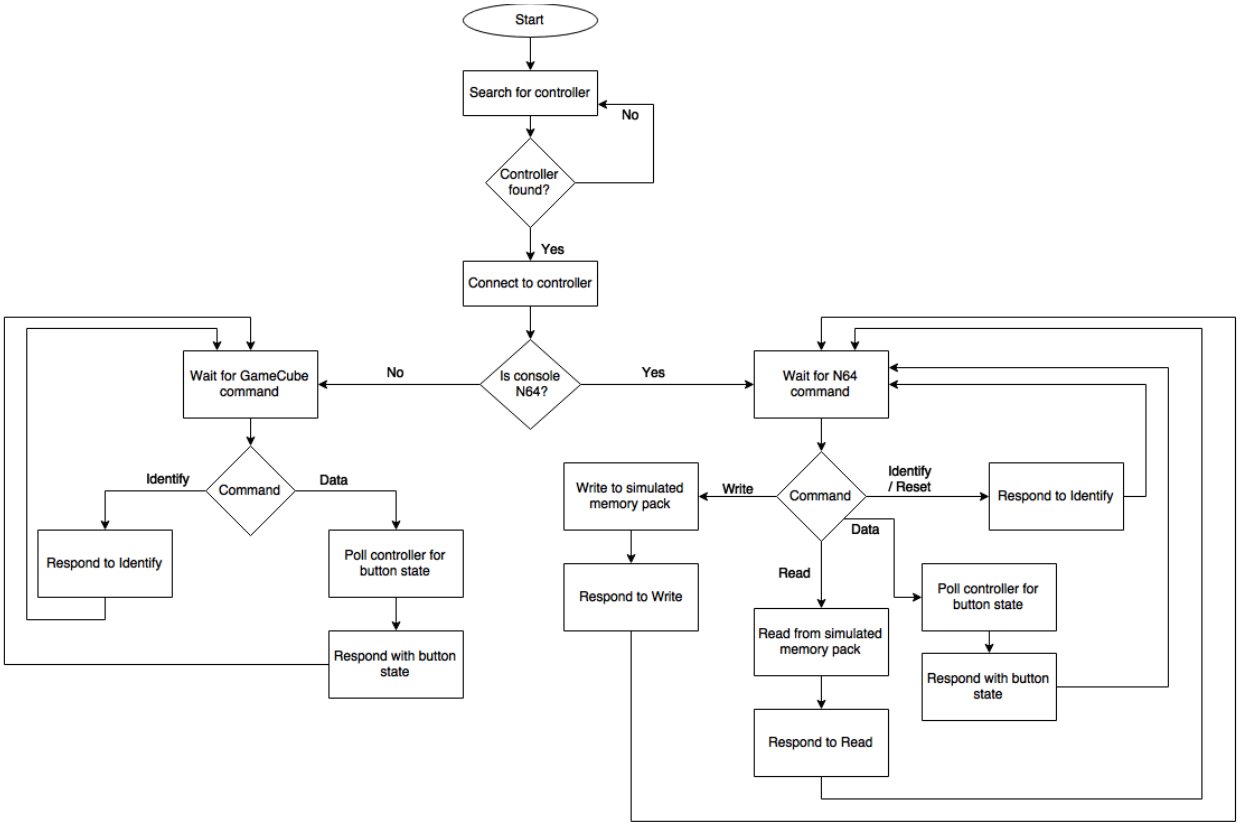


Figure 12: Dongle Software Flowchart

The dongle searches for a controller and connects to it when it finds one. Then it determines which profile it is using by examining its power source. Either way, the dongle will wait for commands from the console. For all commands except for the "Data" command, the dongle will process the request internally and respond to the console. However, for the "Data" command for both consoles, the dongle will request the controller's button state. Once receiving the information from the controller, the dongle will respond to the console with it and wait for another command.