# Universal Game Controller

**ECE 445 Design Document**

Evan Lee, Peter Van Fossan, and Neil Singh
Group 48
TA: Eric Clark
2/24/2017

# 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]. While this is great for the progression of gaming, having new hardware released every couple of years starts to have its monetary impact. With each new console, an individual must buy new controllers that tend to be quite expensive. Unfortunately, there is no way to avoid this situation because previous controllers tend to be incompatible with new consoles. On top of that, with multiple consoles, there would be many different controllers that both take up space and time to acclimate to the change in controller layout.

Our goal is to eliminate this problem by providing a universal game controller that connects to many different consoles. This eliminates the need for the storage of the plethora of controllers for the various consoles and also the adaptation period. However, 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 right now, the only two consoles that will be supported are the Nintendo GameCube and Nintendo 64, but hopefully in the future more could be developed.

## 1.2 Background

For any avid gamer, having multiple systems are a must, but the expense of having four controllers for each might prevent some from realizing their console's true potential. 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.

## 1.3 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

# 2 Design

There are two major components to our design: 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.

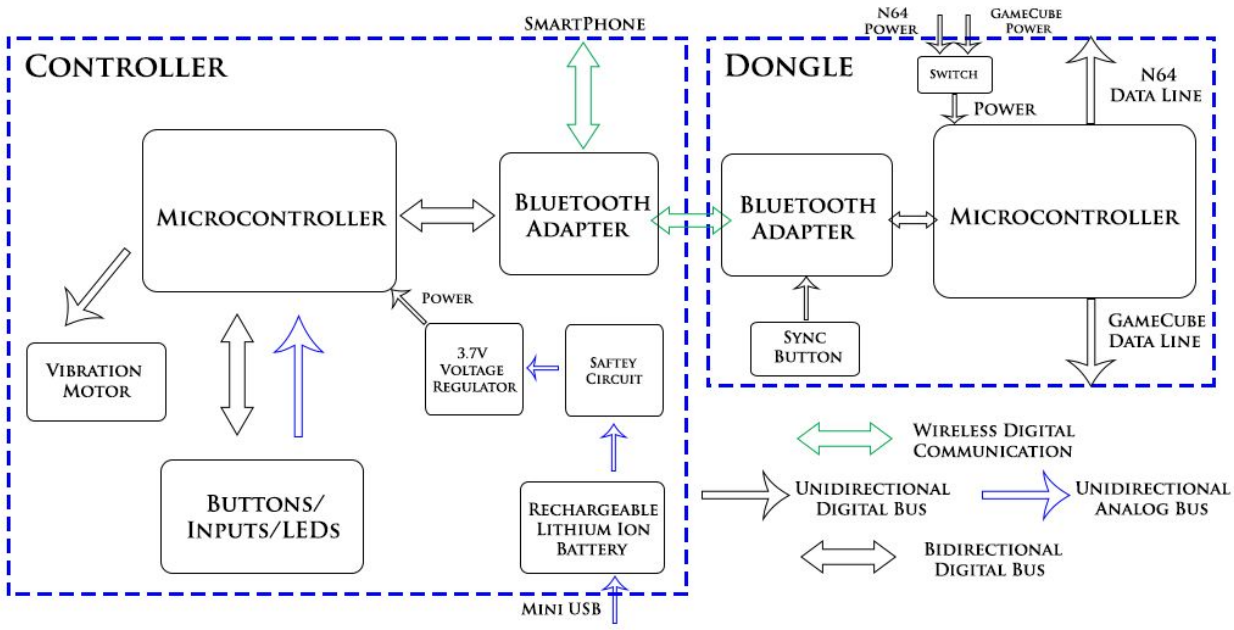


**Figure 1: Block Diagram**

## 2.1 Controller

### 2.1.1 Lithium Ion Battery

The power supply for this controller will be a 3.7V Lithium Ion battery capable of storing 1300mAh of energy. This power supply was taken from the original Wii U Pro controller that we will repurpose 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). It will be charged with the standard battery charger that comes with the Wii U Pro controller through a mini USB interface. The battery should be able to stay in the range of 3.0v to 4.2v to ensure safe operations. Additionally, we will also be implementing an integrated circuit to ensure the battery charges safely.

| Requirements | Verification |
|---|---|
| The battery is able to output voltage in the range of 3.0v to 4.2v | A) Fully charge the battery<br>B) Use voltmeter to measure output voltage and ensure it is less than 4.2v<br>C) Completely discharge battery<br>D) Use voltmeter to measure output voltage and ensure it is greater than 3.0v |

### 2.1.2 Safety Circuit

This circuit ensures that the battery is not being used if it is dead or overcharged (from 3.0v to 4.2v). This prevents the battery from failing and causing chemical fires. The safety circuit we will be using is built into the battery pack for the Wii U Pro Controller.

| Requirements | Verification |
|---|---|
| Ensure no current is drawn from its source when the voltage supplied is not within the 3.0v to 4.2v range. | A) Ensure current is drawn when supplied a voltage of 3.7v<br>B) Ensure current is not drawn when supplied a voltage of 4.3v<br>C) Ensure current is not drawn when supplied a voltage of 2.9v |

### 2.1.3 Voltage Regulator

The voltage regulator is put in place so that regardless of the battery voltage (which will vary with how charged it is), the microcontroller receives a constant 3.7 volts. We will be using the Texas Instruments TLV70237DBVR, which is a 3.7v linear voltage regulator with a 300mA current output.

| Requirements | Verification |
|---|---|
| 1. Provides 3.7v ± 5% when given an input voltage of 3.0v-4.2v<br>2. Can operate at 300mA of current | 1. For both 3.0v and 4.2v, measure that the output voltage is within 5% of 3.7v using a voltmeter<br>2. Draw 300mA of current at 3.7v using a resistor of 12 ⅓ Ω |

### 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.

The microcontroller we have chosen for this design is the TI CC2640 RGZ. It contains 30 GPI/O pins for interfacing with the buttons and is clocked at 48MHz so it should be able to support microsecond precision. We chose this for its low power consumption and low cost, but also because it contains a bluetooth controller as well.

| Requirements | Verification |
|---|---|
| 1. Should be able to respond to the console polls with complete data in less than 16 ⅔ ms (1 frame) so that there is no discernible delay.<br>2. Should be able to update its internal button mapping and respond to the request within 1 second. | 1. Send a "get button info" command to the microcontroller and ensure that the response takes no longer than 16 ⅔ ms.<br>2. Send a "update button mapping" command to the microcontroller and ensure that the response takes no longer than 1 second. |

### 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.

| Requirements | Verification |
|---|---|
| 1. The bluetooth must be able to operate within a 10 meter radius from the dongle | 1. Measure 10 meters from the console/dongle and verify the controller can connect to the dongle. |

### 2.1.6 Vibration Motor

The motor will be housed within the controller and can be turned on or off by the microcontroller, which receives this information from the dongle. The motor is used as part of the gaming experience.

| Requirements | Verification |
|---|---|
| 1. The motor must be able to be turned on by the microcontroller using a 3.7V input. | 1.<br>a. Verify the motor works by connecting it to a protoboard and providing a 3.7V input.<br>b. Manually send a bluetooth signal from the dongle to the microcontroller to trigger the motor in the controller |

### 2.1.7 Buttons/Inputs

We will be using the Wii U Pro Controller for our controller shell.

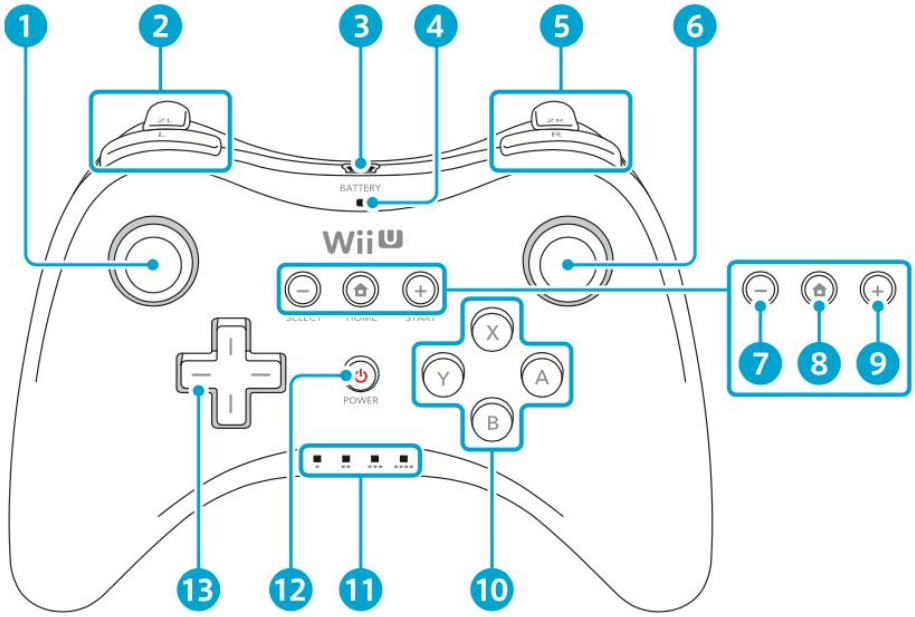


Figure 2: Front of Controller [3]

All the inputs feed into the microcontroller, separated by only either a potentiometer or digital button. The left analog stick (1 in Figure 2) 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 pad's (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.

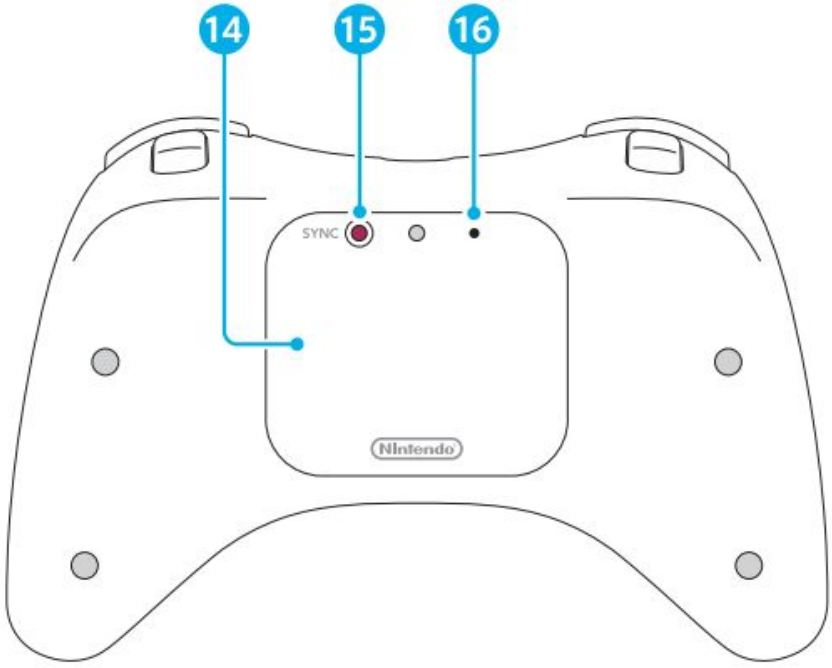


**Figure 3: Back of Controller [3]**

The sync button (15 on figure 3) will be connected by a digital button to the microcontroller, as well as the reset button (16).

| Requirements | Verification |
|---|---|
| 1. Inputs must be of high quality that feel good to use.<br>2. Inputs must be able to relay their informational state (pressed for buttons, axis position for joysticks) to the microcontroller immediately. | 1. Ensure that the inputs can be used comfortably.<br>2. Ensure that the microcontroller can output the input's state to the smartphone connected to it with precision of 16 ⅔ ms. |

## 2.2 Dongle

### 2.2.1 Power Switch

Because the dongle will be able to plug into both a Nintendo 64 console and a Nintendo Gamecube console, 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.

| Requirements | Verification |
|---|---|
| 1. The switch must be able to be switched easily but not accidently | 1. Verify you can switch the switch with ease but it will not move on its own. |

### 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. 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 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 communication protocol.

The microcontroller we have chosen for this design is the TI CC2640 RGZ. It contains 30 GPI/O pins for interfacing with the console's signals and is clocked at 48MHz so it should be able to support microsecond precision. We chose this for its low power consumption and low cost, but also because it contains a bluetooth controller as well.

| Requirements | Verification |
|---|---|
| 1. Can output signals with microsecond resolution.<br>2. Can convert the controller's state to the connected console's protocol and output it within 1 frame (16 ⅔ ms). | 1. Output a signal that alternates low and high every microsecond and verify on an oscilloscope.<br>2. Measure how long it takes to convert and output the controller's state for both protocols and ensure it takes less than a frame. |

### 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.

| Requirements | Verification |
|---|---|
| 1. The bluetooth must be able to operate within a 10 meter radius from the dongle | 1. Measure 10 meters from the controller and verify the dongle can connect to the controller. |

## 2.3 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 [4]**

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.

## 2.4 Nintendo GameCube Controller Protocol

The GameCube controller protocol is very functionally similar to the Nintendo 64 protocol, although surprisingly a bit more simple. 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 [5]**

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.

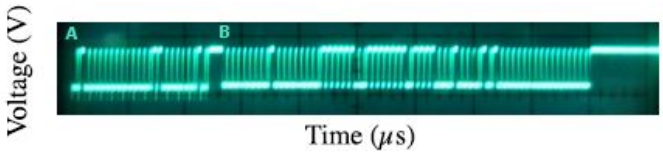


**Figure 4: Example of GameCube Request and Response [5]**

## 2.5 Schematics

### 2.5.1 Joystick

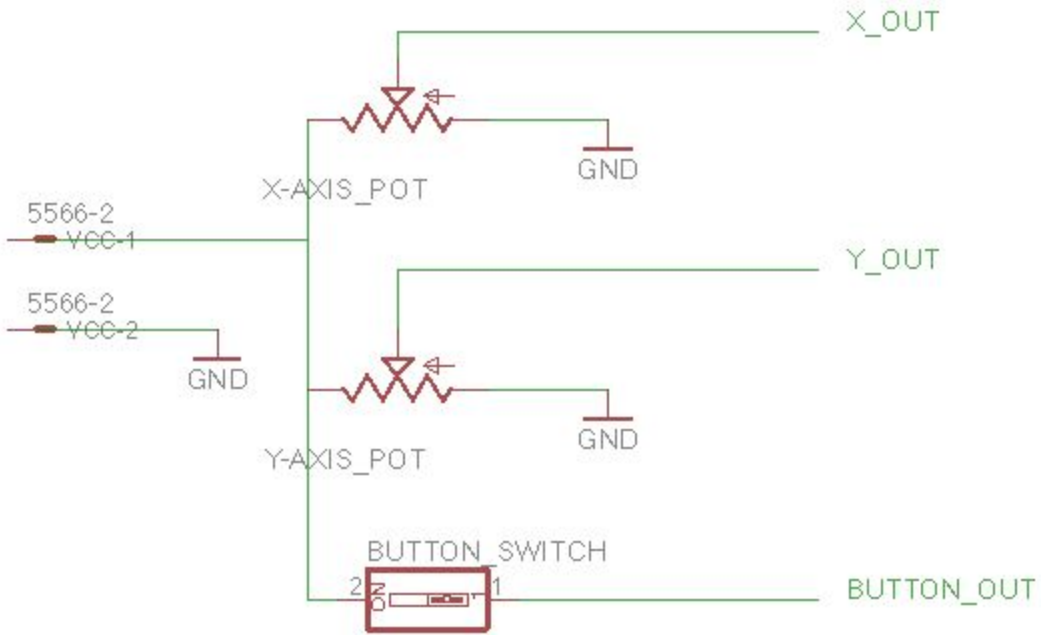


**Figure 5: Joystick Schematic**

### 2.5.2 Microcontroller and IO connections

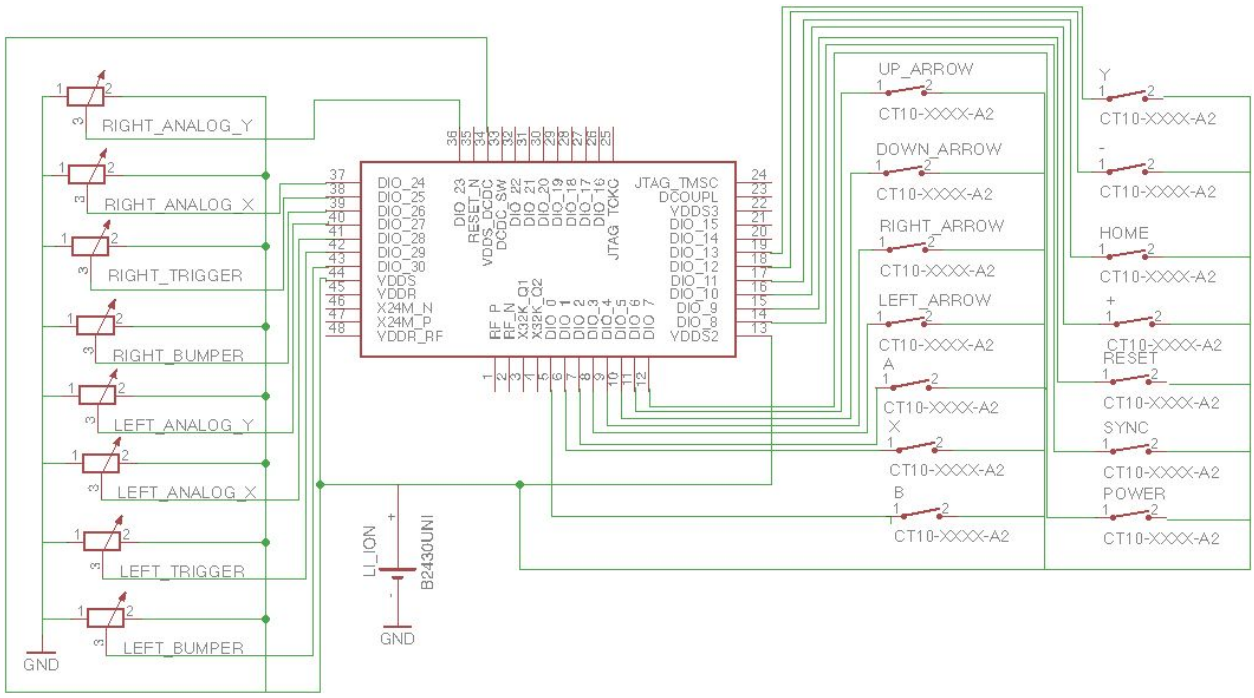


**Figure 6: Microcontroller/IO Schematic**

## 2.6 Software

Our design incorporates a good amount of software, executed by the microcontrollers in both the controller and the dongle. The purpose of the software for the controller is twofold; one, it must be able to connect to a smartphone (in our case iPhone) to allow for custom button mappings to be set. There will also be software on the iPhone that will enable this to be done. On top of this, the software on the controller must be able to interpret the physical button state and relay this information to the dongle when requested after syncing via bluetooth to the dongle.

For the dongle, the software is slightly more interesting. It must first be able to sync to the controller over bluetooth and then determine whether it is connected to the N64 or the GameCube (by examining the state of the dongle switch). After that the software waits for commands from the console. It will interpret this command and then send a data request to the controller. The controller will respond with its current button state and the software must convert this data into a format that the corresponding console will understand. Finally the software will output the response to the console.

Below is a communication flow showing the communication between the controller, dongle and console.

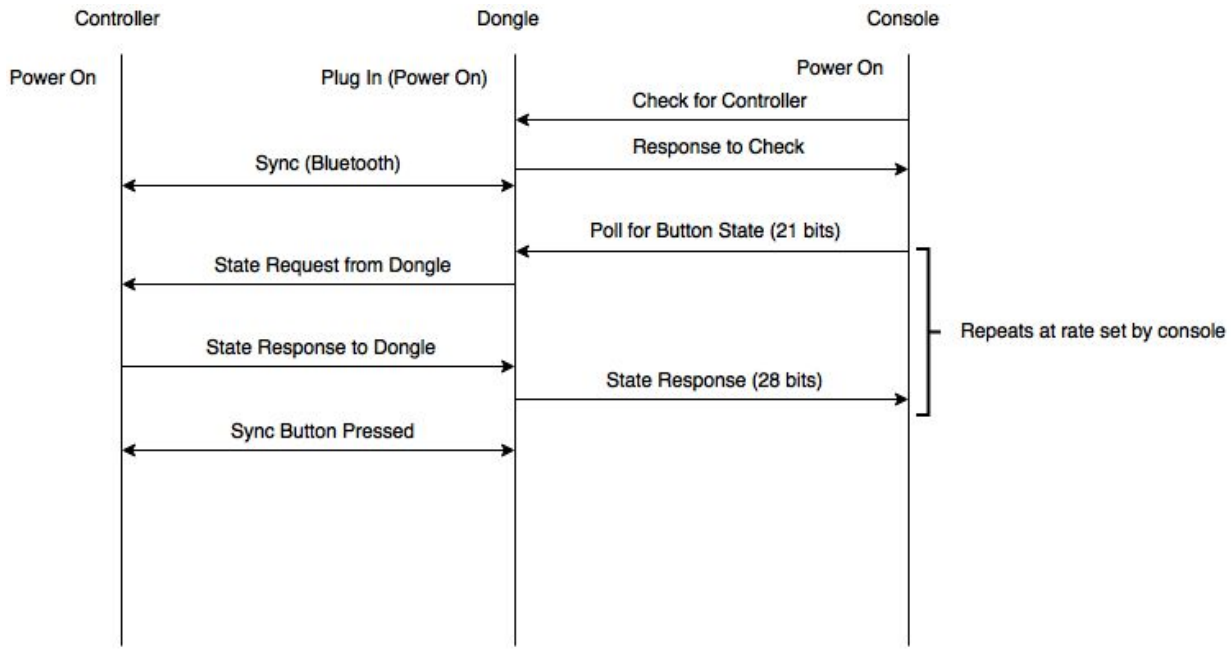


**Figure 7: Communication Between Controller, Dongle, and Console**

## 2.7 Tolerance Analysis

In order to capture the position of the joysticks to send to the console, the controller utilizes two potentiometers for each joystick. Based on the position of the joystick, the pots will set variable resistances to drop the voltages through the joystick. The microcontroller can then read the output voltages of the pots to determine the position of the joysticks and send this information to the console. The potentiometer resistances for the left and right directions for each joystick range from 1.60 kΩ (left) to 5.7 kΩ (right). The potentiometer resistances for the up and down directions for each joystick range from 1.4 kΩ  (up) to 5.76 kΩ (down). The output voltages for the left and right pot connected to a standard 3.7 V power supply will be 3.53V (left) to 1.88 V (right), and the output voltages for the up and down pot will be 2.9 V (up) and 0.73 V (down). On startup, the microcontroller will read the voltage output for the each potentiometer to read the neutral position voltage.

The controller communication protocols for the Nintendo 64 and Nintendo GameCube both use two bytes to represent the position of each joystick, one for the up and down direction and another for the left and right direction. To represent the position of the joysticks in bytes, the microcontroller must normalize the range of output voltages for each direction to 256 values. In order to avoid overflow and underflow, we will provide a 5% tolerance for each direction's resistance range so that our output voltages remain in our range. Our resistance range for the left and right direction then becomes 1.6 kΩ - .205 kΩ to 5.7 kΩ + .205 kΩ, or 1.395 kΩ to 5.905 kΩ. The resistance range for the up and down direction becomes 1.4 kΩ - .218 kΩ to 5.76 kΩ + .218 kΩ, or 1.182 kΩ to 5.978 kΩ. In order to avoid overflow and underflow for rare instances when the voltages drop outside our tolerance range, we will also be clamping the voltage reading on the microcontroller. If the value of the joystick position after normalization is less than -128

(for 2's complement), the value of the joystick position becomes -128. Likewise, if the value of the joystick position after normalization is greater than 128, the value just becomes 128.

## 3 Costs

We estimate a fixed salary of $40/hour, 5 hours/week for three people for a complete design. This leads us to the following labor cost calculation.

$$3 * \frac{\$40}{hr} * \frac{5hr}{wk} * 16wks * 2.5 = \$24{,}000$$

Equation 1: Labor Cost

We will be using the following parts in our design.

| Part | Cost |
|---|---|
| 2 Microcontrollers (TI CC2640 RGZ) | $4.73 x 2 = $9.46 |
| PCB (est.) | ~$4.00 |
| Assorted resistors, switches, wires (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 $24,072.96.

## 4 Schedule

| Week | Peter | Evan | Neil |
|---|---|---|---|
| **2/27** | Finalize PCB design | Begin design of Dongle on AutoCad | Reverse engineer Wii U Pro parts to get better understanding of hardware components |

| | | | |
|---|---|---|---|
| **3/6** | Begin programming controller microcontroller | Finish design of Dongle, 3D print prototype. Being programming dongle microcontroller | Continue reverse engineering of Wii U Pro parts |
| **3/13** | Continue programming controller microcontroller | Continue programming dongle microcontroller | Begin programming mobile application for button mapping |
| **3/27** | Continue programming controller microcontroller. Begin preliminary tests for bluetooth communication with dongle | Continue programming dongle microcontroller. Begin preliminary tests for bluetooth communication with dongle | Continue programming mobile application for button mapping |
| **4/3** | Finalize controller microcontroller, begin assembly | Finalize dongle microcontroller, begin assembly | Finalize mobile application |
| **4/10** | Have testable prototype ready | Have testable prototype ready | Have testable prototype ready |
| **4/17** | Debug | Debug | Begin Final Report |
| **4/24** | Prepare Final Presentation / Debug | Prepare Final Presentation / Debug | Prepare Final Presentation / Debug |

## 5 Ethics and Safety

Our biggest safety concern in this project deals with the lithium ion battery that we will be using to power our controller. If we do not take proper precautions, we can 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 will closely follow the ECE 445 Safe Practice For Lead Acid and Lithium Batteries document [6].

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”[7]. 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”[8]. To prevent this we will use a microcontroller that can not output more than 5V DC.

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

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

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

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

[7] "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.

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

[9]J. Buxton, "Li-Ion battery charging requires accurate voltage sensing," 1995. [Online]. Available: http://www.analog.com/en/analog-dialogue/articles/li-Ion-battery-charging-accurate-voltage-sensing.html. Accessed: Feb. 25, 2017.