Button Remapping for GameCube Games such as Super Smash Bros Melee

ECE445 Design Document

Team 14 - Michael Qian, Srikanth Yaganti, Yeda Wu
ECE 445 Design Document - Spring 2020
TA: Evan Widloski

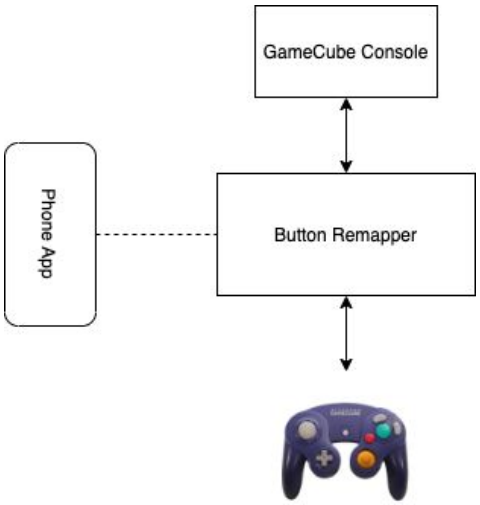
1 Introduction

1.1 Problem and Solution Overview

In fighting games, it is usually beneficial to remap certain buttons to perform different actions for ease of doing combos. For example, a player might want to remap the X button on their controller from "jump" to "attack". This is present in the game settings of many popular fighting games except for Super Smash Bros Melee for the Nintendo GameCube.

Our goal is to create an adapter that sits between the GameCube and GameCube controller. The controller will plug into the adapter which plugs into the GameCube. Users will have a phone app where they can choose how to remap their buttons. Users can then load the adapter with multiple button reconfigurations and toggle through these configurations on the adapter. This adapter will then take in the signals of the button presses of the controller and translate them to signal button presses based on the button remapping. This hardware will also allow for button remapping for any other GameCube games.

1.2 Visual Aid

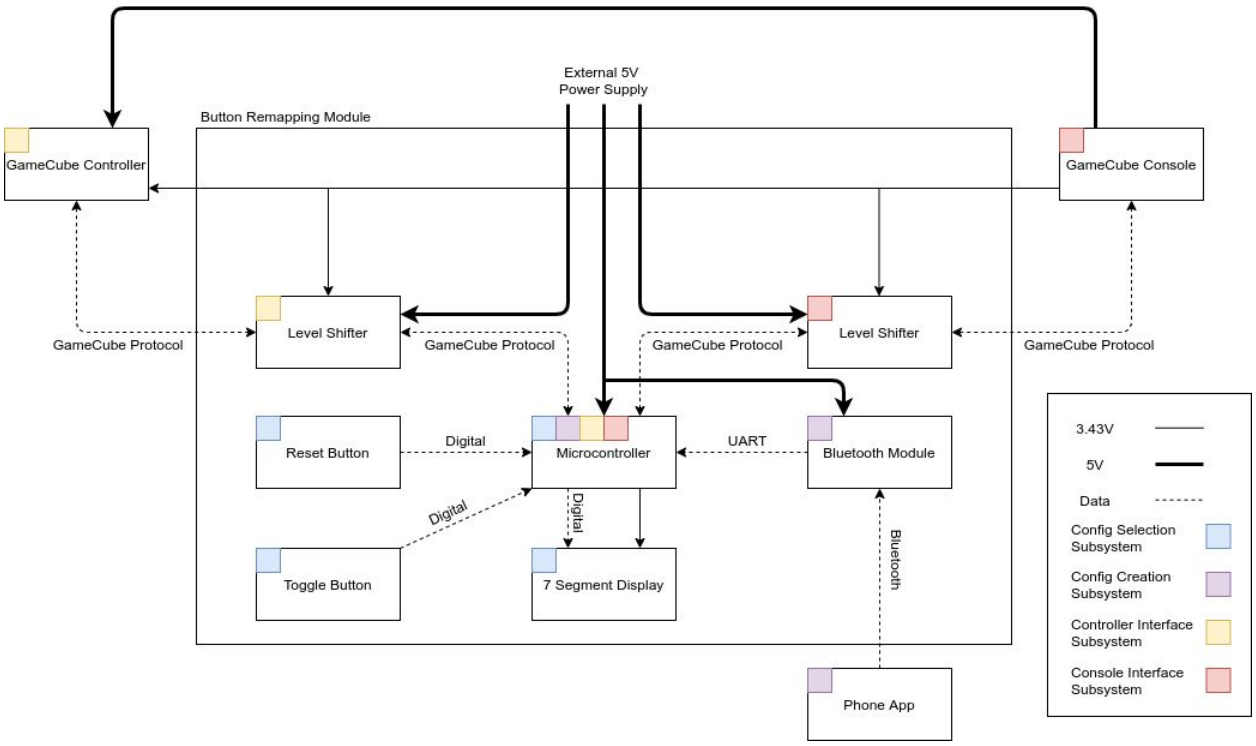


1.3 High-level requirements list

- Phone app is able to communicate with microcontroller via bluetooth to send the button-remapping schemes
- Microcontroller is able to read the controller inputs
- Microcontroller is able to send the remapped button inputs to the console

2 Design

2.1 Block Diagram



2.1.1 Phone App

A phone app is required to allow users to easily set button remapping configurations for their controller. This app will connect to the microcontroller via bluetooth module HC-05. App should be able to store multiple preset configurations and users should be able choose which to use. We are considering sending the data as a string that will be received and parsed by the microcontroller which will store the configurations in a struct.

2.1.2 Button Remapping Module

2.1.2.1 Microcontroller

The microcontroller is required to communicate with 3 external devices. First it should be able to communicate with the phone app via bluetooth, then it is required to write and read data from GameCube Console, and finally it will be required to read and write data from/to GameCube console. This requires our microcontroller to have one RX and TX ports for reading and writing data from bluetooth device and at least 2 I/O pins for reading and writing to console and controller via bitbanging using a custom software serial library for communication via GameCube protocols. We have determined that the ATmega328 microcontroller matches all these criteria and

is therefore a good choice for our project. In addition the microcontroller is supposed to mimic the polling that GameCube Console does for the controller. To meet the high-level requirements, the microcontroller should be able to successfully act as a middle man for the communication between controller and console and remap the button data as they communicate. Microcontroller will be required to store configurations so that users can easily change between configs within the remapping module itself. We are considering storing this data in the microcontroller EEPROM of size 1 KB.

2.1.2.2 Bluetooth Module

Required for communication with the phone app. It will contain a bluetooth receiver for receiving configurations for button remappings from the phone. We have opted to use the HC-05 Bluetooth module because of its simplicity and compatibility with the STM32F103.

2.1.2.3 Toggle Button

Since we wish to be able to store and switch between multiple button remapping configurations, a button is needed to cycle through the number of configurations stored within the microcontroller (4 or 8 configurations).

2.1.2.4 7-Segment Display

This will display the current button remapping configuration being implemented by the microcontroller. It is simply a set of LEDs driven by the microcontrollers digital output pins. All remapping configurations will have a number associated with it, and this display will show the current configuration being used.

2.1.2.5 Reset Button

This button will reset the current configuration and the microcontroller will then simply pass the data from the GameCube controller on to the console without any button remapping.

2.1.2.6 5V Power Supply

This will power both the microcontroller and the bluetooth module. It will need to supply 0.3mA and between 1.8V to 5.5V to the microcontroller. It will need to supply 30mA and between 4 to 6V to the bluetooth module.

2.1.2.7 Level Shifter

This is used to convert the microcontroller’s signals from 5V to 3.43V, which is the voltage level used by the GameCube controller and console for digital data.

2.1.3 GameCube Controller

GameCube Controller will communicate with the Microcontroller to send button data. This data will be remapped by the microcontroller and sent to the GameCube console.

2.1.4 GameCube Console:

GameCube Console will receive the proper button mappings from the microcontroller. It will also provide power to all other modules.

2.2 SubSystems

2.2.1 Config Creation Subsystem (Phone App, Bluetooth, Microcontroller)

The purpose of this subsystem is to allow the user to create button configurations and have them stored on the microcontroller. The bluetooth module will allow communication of the user configurations between the phone app and the microcontroller. User configurations will be stored in structs in the microcontroller's EEPROM. There is 1 KB of EEPROM on the ATmega328p, so it is more than enough memory to store configurations. We expect to use 11 bytes of data to store the button configurations (1 byte per remapped button), and we also plan to only allow users to make 10 different configurations. This means we will be only using 110 bytes of EEPROM. To pass configs between the microcontroller, we will be passing them as strings. Each config will be converted to a 11 char length string. The index of a character in the string is the default button value and the value of the character is the remapped button value.

Configs will be stored on the microcontroller in a struct similar to what is below:

```
#define A 0
#define B 1
…
#define D_RIGHT 10
struct Config {
      uint8_t a;
      uint8_t b;
      uint8_t l;
      uint8_t r;
      uint8_t x;
      uint8_t y;
      uint8_t z;
      uint8_t d_up;
      uint8_t d_down;
      uint8_t d_left;
      uint8_t d_right;
} config;
```

Requirements	Verification
1) Microcontroller receives configs from the phone app	1) User presses a button on the phone app to transfer configurations via bluetooth to the microcontroller.

2.2.2 Config Selection Subsystem (Reset button, Toggle Button, 7-segment display, Microcontoller)

Users will click the reset button to clear the configurations. The toggle button will be used to switch between a maximum of 10 different configurations, and the 7-segment display is for showing which configuration is being used. All configurations are stored in EEPROM. When the toggle button is pressed, the new corresponding configuration will be copied from EEPROM to SRAM to allow for faster accesses and remapping.

Requirements	Verification
1) Resetting configurations is functional; set button configuration to default settings 2) Toggling configurations is functional; switching between the stored configurations on the microcontroller	1) Store a configuration on the microcontroller. When the user clicks the reset button, controller inputs are now back to the default mapping. 2) Store 2 configurations on the microcontroller. When the user clicks on the toggle button, controller inputs are switched from one configuration to the next.

2.2.3 Controller Interface Subsystem (GameCube Controller, Microcontroller, Level Shifter)

The microcontroller's purpose here is to send polling requests to the GameCube controller and read responses from the GameCube controller. In order to do this, we must follow the GameCube protocol for bit representation and polling the controller button values. Because this is not UART, we are unable to use the TX and RX pins on the microcontroller. To interpret and send these digital signals, we will be bit banging with a digital pin on the microcontroller.

2.2.3.1 Bit representation

Zero and one bits are represented in GameCube's unique protocol. A zero bit is low for 3µs and then high for 1µs. A one bit is low for 1µs and then high for 3µs.

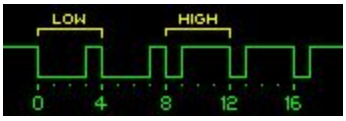


Figure 1. Zero and one bits for GameCube[4]

2.2.3.2 Polling

The GameCube console polls the controller for inputs with a 24-bit command sequence (0100 0000 0000 0011 0000 0010). The console replies with a 8 byte sequence.

Data format sent by the controller:

- Byte 0 - 0, 0, 0, Start, Y, X, B, A
- Byte 1 - 1, L, R, Z, D-up, D-down, D-right, D-left
- Byte 2 - Joystick X Value (8bit)
- Byte 3 - Joystick Y Value (8bit)
- Byte 4 - C-Stick X Value (8bit)
- Byte 5 - C-Stick Y Value (8bit)
- Byte 6 - Left Button Value (8bit)
- Byte 7 - Right Button Value (8bit)

Requirements	Verification
1) Microcontroller should successfully pass the polling data to GameCube Controller 2) Microcontroller should successfully receive data from GameCube controller and successfully remap the button inputs to the configurations set by the user 3) Microcontroller should mimic GameCube controller data transfer of 4µs per bit. About 250kbits/second. It should be able to read the data from the controller at a rate of 250kbits/sec. Baud Rate of 115200 for GameCube controller	1) Connect GameCube Controller's data pin to I/O pin on microcontroller and run program which do bitbanging to send polling data and to receive responses from GameCube controller. Test to see if we receive 64-bit data of button inputs from the controller. 2) Send the 24-bit command sequence (0100 0000 0000 0011 0000 0010) to the controller from the microcontroller and verify the pad will respond with 64-bit data via bit banging. Going to be using a custom bit banging software API for GameCube protocol. 3) Verify that microcontroller correctly parses and remaps 64-bits of data based on the config sent by phone app. 4) Verify that microcontroller can read data from GameCube controller at a rate of 250Kbits/sec by connecting it to the

	oscilloscope and measuring that each bit takes about 4μs to send.

2.2.4 Console Interface Subsystem (GameCube Console, Microcontroller, Level Shifter)

The microcontroller's purpose here is to detect polling command sequences from the GameCube console, and send the remapped button inputs of the GameCube controller to the console. In order to do this, we will be bit banging using a digital pin on the microcontroller.

Requirements	Verification
1) Mirocontroller should successfully pass 64-bit remapped data from the GameCube controller to the GameCube Console via bitbanging of GameCube Protocol API 2) Microcontroller should successfully mimic controller responses to polling by the GameCube Console. Sending data at a rate of 250Kbits/second. Baud Rate of 115200. 8 bytes of data in the format stated above.	1) Connect GameCube Console data pin to I/O pin on microcontroller and run program to send responses to polling requests from the console via bitbanging. Verify to see if the console has detected a controller. With an official controller attached, there is a typical interval of about 8ms between successive updates. Check to see if polling updates requests fall in that time interval. We can do this by printing time difference between two consecutive update request and check if its around 8ms. 2) Verify that microcontroller correctly sends 64-bits of controller data by seeing how the game responds to different button inputs. Test via oscilloscope to see if the transfer rate of data from microcontroller to console is 4μs per bit.

2.3 Software

2.3.1 Polling Algorithm

The GameCube console polls the controller roughly every 8ms. Our goal is to respond to the controller with remapped button configurations as quickly as possible. A simple approach would be to poll the controller whenever we receive a poll request from the console, read the controller response, and send the remapped controller response to the console. This method may not be fast enough though.

Figure 2. Naive method's steps for replying remapped values to the console.

Our approach is to instead poll the controller separately from the console's polling. The microcontroller will poll the controller for its values in between the console's poll requests. We will then remap the controller's values and have that ready to be sent to the console before the console requests for the controller values.

Figure 3. Our method's steps for replying remapped values to the console.

2.3.2 Phone App

There will be 2 screens on the app: a menu screen to display configurations and one to edit a configuration. The menu screen will display all the configurations that the user has made. Users will have options to select a configuration, create a new configuration, or send the configurations to the microcontroller. When the user selects one of the configurations, a new screen will appear for editing the configuration. In this screen, they are able to remap their button values. Also, the app will cap the user to only be able to create 10 different configurations.

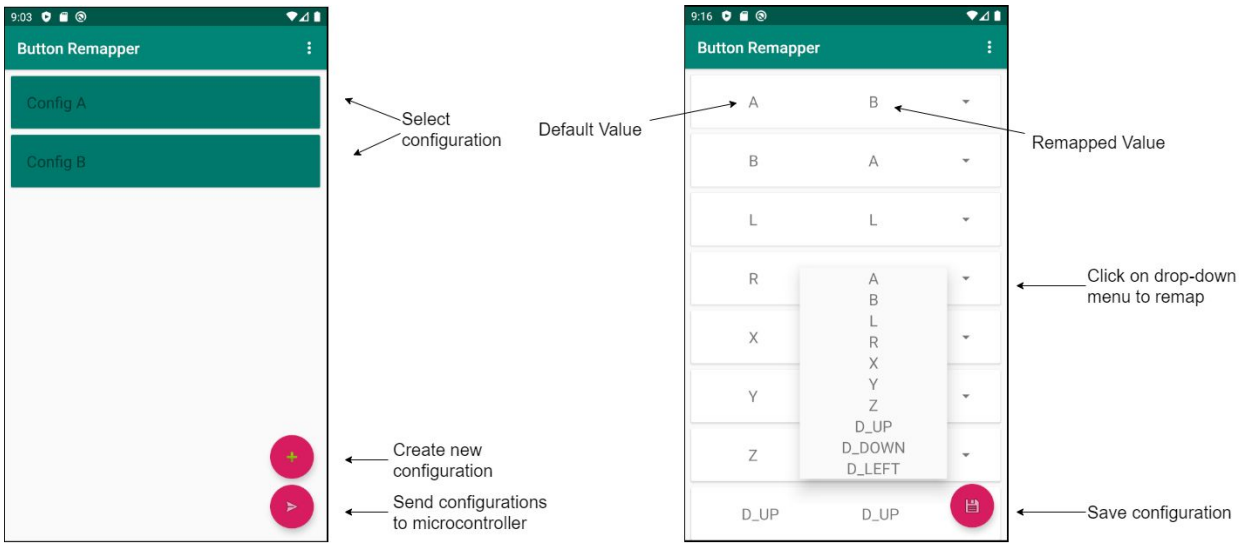


Figure 4. Design of menu screen and configuration editing screen of the app

2.4 Tolerance Analysis

The Gamecube console polls for controller values every 8ms. In this time period, the microcontroller must do the following:

1. Send the polling request to the controller
2. Detect the response from the controller
3. Remap the controller response values
4. Detect the polling request from the console
5. Send the remapped values to the console

Step 1 will take 4 * 24μs = 72μs. Each bit takes 4μs to send, and there are 24 bits in the polling request[4].
Step 2 will take 4 * 8 * 8μs = 256μs because we must read 8 bytes of data from the GameCube controller.
Step 4 will take 4 * 24μs = 72μs because we are reading 24 bits from the console.
Step 5 will take 4 * 8 * 8μs = 256μs because we are sending 8 bytes of data to the console.

In total, all steps but step 3 will require 72 + 72 + 256 + 256 = 656μs. This means that we have 7.344ms for the microcontroller to perform the remapping step and any other delays. When running the microcontroller at 16MHz, we will have 117,504 clock cycles for this, which should be more than adequate.

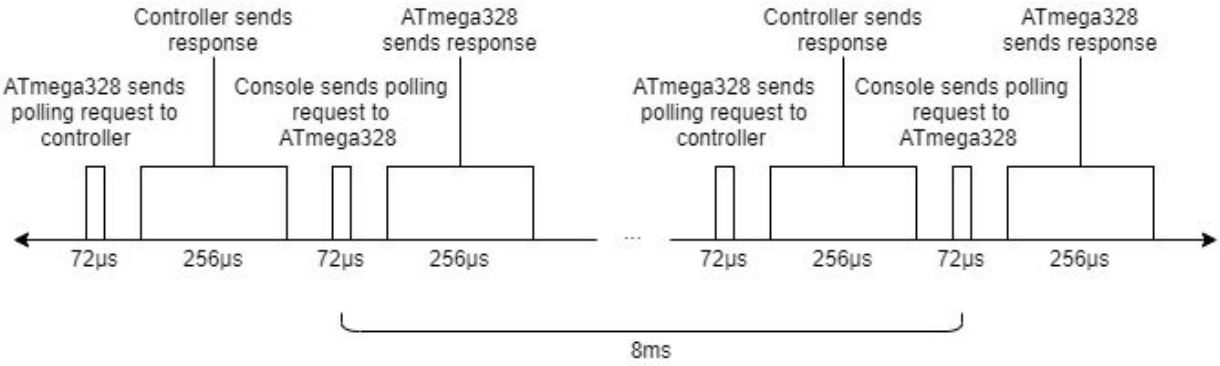
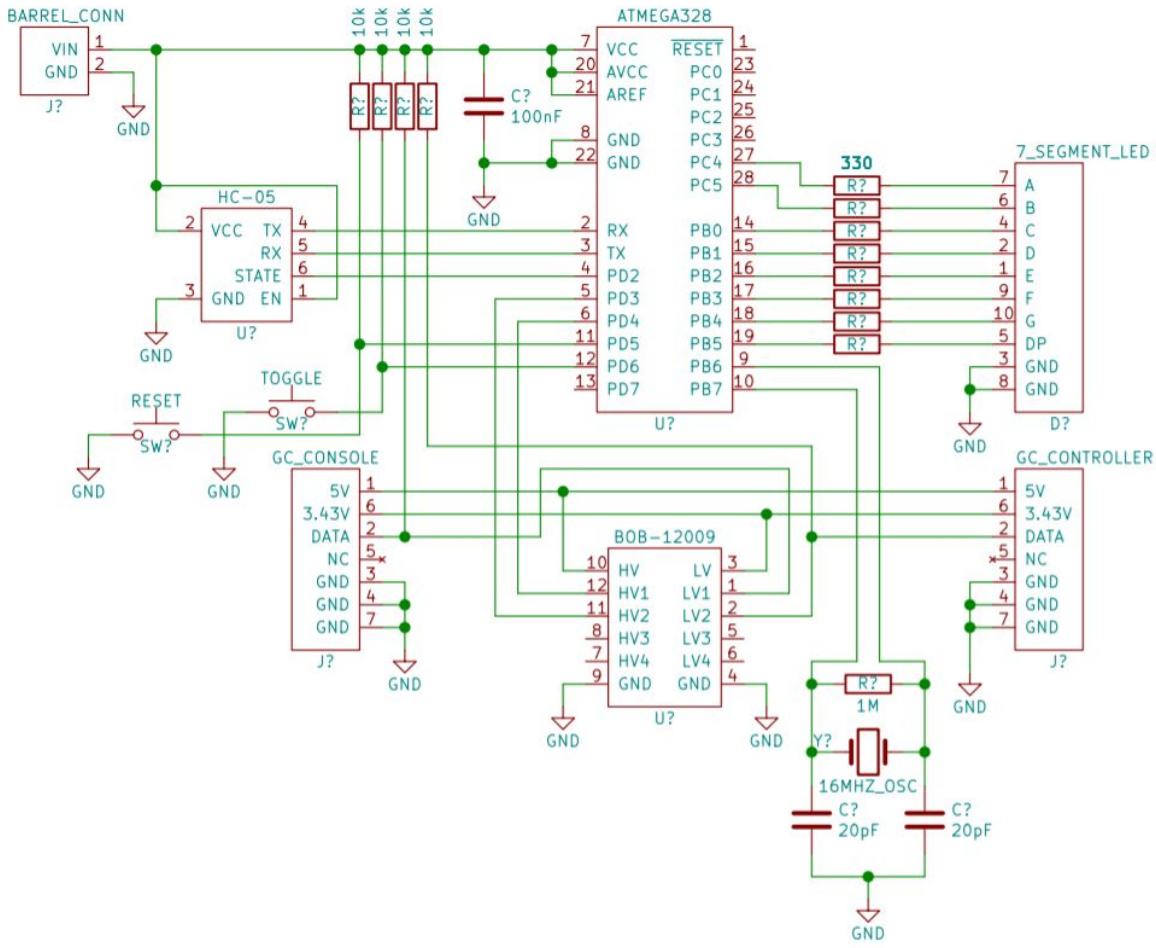


Figure 5. Timeline for polling and responses

2.5 Schematic



3. Cost and Schedule

3.1 Cost Analysis

At $50 per hour and 10 hrs per week for 3 people for 16 weeks, we expect this to result in $24,000 for the labor cost. Below is the cost breakdown for one manufacturing one part:

Part	Quantity	Cost (prototype)	Cost (bulk)
ATmega328 Microcontroller	1	1.90 (Digikey, ATMEGA328-PU-ND)	1.58 (Digikey, ATMEGA328-PU-ND)
Gamecube extension cord	1	5.85 (Amazon, company: Mizar)	2.75 (Amazon, company: Pegly)

HC-05 Bluetooth Module	1	19.14 (Digikey, 1738-1164-ND)	2.58 (DHgate, seller: Tenypure)
Assorted resistors, capacitors, buttons, crystals, sockets	1	3.00 (Digikey, est.)	0.25 (Digikey, est.)
5V Power supply	1	4.69 (Amazon, company: Yosoo)	2.30 (DHgate, seller: Vizgiz)
Level shifter	1	2.95 (Sparkfun, BOB-12009)	0.54 (Digikey, 296-21929-2-ND)
PCB	1	2.30 (PCBWay)	0.13 (PCBWay)
Total		**39.83**	**10.13**

In total, if we wish to create 10 parts, then the overall development cost will be **$24,398.30**.

3.2 Schedule

Week	Michael	Srikanth	Yeda
2/17	Create phone app to allow users to configure data	Work on design doc	Work on design doc
2/24	Work on sending data between phone app and bluetooth module	Clip GameCube cables such that we can connect the GameCube controller to male and female ends.	Design PCB
3/2	Write code to store configurations in EEPROM and read configurations	Mock console polling signal to the GameCube controller via a microcontroller	Mock console polling signal to the GameCube controller via a microcontroller
3/9	Read console polling signal via a microcontroller	Mock console polling signal to the GameCube controller via a microcontroller	Mock console polling signal to the GameCube controller via a microcontroller
3/16	Read console polling signal via a microcontroller	Read controller response signals after being sent a mocked console polling signal	Read controller response signals after being sent a mocked console polling signal

3/23	Mock controller response signal after obtaining a polling request from the console	Read controller response signals after being sent a mocked console polling signal	Rework PCB
3/30	Mock controller response signal after obtaining a polling request from the console	Read controller response signals after being sent a mocked console polling signal	Solder components onto PCB
4/6	Combine project components together on breadboard and test	Combine project components together on breadboard and test	Combine project components together on breadboard and test
4/13	Combine project components on PCB and test	Combine project components on PCB and test	Combine project components on PCB and test
4/20	Combine project components on PCB and test	Combine project components on PCB and test	Combine project components on PCB and test
4/27	Prepare presentation	Prepare presentation	Prepare presentation
5/4	Prepare presentation	Prepare presentation	Prepare presentation

4 Discussion of Ethics and Safety

From a hardware perspective, the safety concerns are few but not absent. For example, the GameCube console has a specified signal voltage of 3.43V, whereas the microcontroller and the rest of the modules all require 5V of power. Accidently miss wiring the power inputs of these two parts could result in the serious damage of both these parts. Furthermore, many of the parts used in this project will be susceptible to electrostatic discharge and thus, precautions must be taken to prevent damaging these parts such as using anti-static gloves and ESD wristbands.

For ethics, we hold responsibility for our project, which is the first rule in the IEEE Code of Ethics[6]. Additionally, our product could introduce a situation within professional gaming if the device is used when non-Nintendo/performance-enhancing devices are not allowed. This would be an unethical use of our device. In addition to this, it may be possible for people to tamper with our microcontroller such that certain button presses can lead to button macros. These two situations violates the IEEE Code of Ethics #9 because, if used in such a way, the trust in and reputation of professional gamers will be harmed[6].

5 Citations

[1] “GameCube,” *Wikipedia*, 09-Feb-2020. [Online]. Available: https://en.wikipedia.org/wiki/GameCube. [Accessed: 09-Feb-2020].

[2] “SMASH BOX,” *Hit Box Arcade*. [Online]. Available: https://www.hitboxarcade.com/products/smash-box. [Accessed: 07-Feb-2020].

[3] “B0XX Controller,” *B0XX*. [Online]. Available: https://b0xx.com/. [Accessed: 07-Feb-2020].

[4] “Nintendo Gamecube Controller Protocol,” *Nintendo Gamecube Controller Pinout*. [Online]. Available: http://www.int03.co.uk/crema/hardware/gamecube/gc-control.html. [Accessed: 14-Feb-2020].

[5] “8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash” *ATMEL*. [Online]. Available: https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf. [Accessed: 13-Feb-2020].

[6] “IEEE Code of Ethics,” *IEEE*. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 13-Feb-2020].