ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

MacroME: The Programmable GameCube Controller

Team #67

DEAN BISKUP
(dbiskup2@illinois.edu)
ADITHYA RAJAN
(adithya2@illinois.edu)

TA: Chi Zhang/Megan Roller

May 8, 2020

Abstract

MacroME is a project that aims to make certain complex input combinations in fighting games, such as *Super Smash Bros.*, easier for beginners. In these games, there are often complex move combinations that give you a competitive edge over the opponent, with no way to remap the button layout of the controller. The original solution to this problem was an adapter that translated inputs from a standard GameCube controller into remapped inputs. In contrast, MacroMe provides both programmable macro capability as well as button remapping capabilities housed in the form factor of a GameCube controller.

Contents

1 Project Motivation

This section includes our motivations and inspirations behind creating MacroME. We discuss a general overview of the product and its expected functionalities. We also compare existing solutions and the original project that proposed a solution to the problem, and defend how MacroME differentiates itself from and improves upon the existing products.

1.1 Problem Statement

Super Smash Brothers is one of the most famous video game franchises, with titles such as *Super Smash Bros. Melee* frequently ranked among the best fighting games of all time [1][2]. While current iterations of the *Super Smash Bros.* series allow the game's controls to be changed in-game, older titles such as *Melee* did not allow for this feature, forcing all players to play with default controls. In addition, *Super Smash Bros.*, like other fighting games, features combinations of moves (combos) that may be too difficult to execute for beginners, yet required if they are to compete with more experienced players. This creates an issue where the learning curve is too steep, resulting in player burnout especially among new players who are trying to learn the game. A few examples of these high level, complex combos are described in Appendix A.

1.2 Solution

1.2.1 Solution Overview

Our solution is "MacroME: The Programmable GameCube Controller," a fully functional Nintendo GameCube controller, but with extra features to enhance the player experience. The first primary functionality of MacroME is to allow button remapping, where the user can remap buttons on the controller to different actions. For example, the user could choose that the X button instead presses the L or R buttons. The second primary functionality of MacroME is to allow for programmable macroinstructions (macros). This will allow the user to select a string of inputs, timed frame by frame, for the controller to automatically execute upon the press of a button. By doing this, the user can perform complex strings of inputs for certain combos or techniques in the game.

One of the key components of MacroME is that it looks and feels just like a standard GameCube controller. For this reason, our custom PCB will be fit inside the standard GameCube controller shell, with all the buttons in the same place as the original controller. MacroME will connect to the game console through a GameCube connector cable, while also be able to connect to a PC through USB for programming macros and button layouts. Additionally, the saved layouts and macros will be stored on the controller itself, so a PC is not necessary to use the player's stored configurations.

We hope that MacroME will encourage beginner players to try the more complex characters in the *Super Smash Bros.* games, and make access to competitive mechanics less intimidating. Additionally, MacroME can help players used to other button layouts learn

the game more easily through the button remapping feature. As an added bonus, due to the popularity of the GameCube controller, there are many adapters on the market that would allow this controller to also be used for PC, PlayStation, or Xbox games, expanding our user base beyond just GameCube players.

1.2.2 Inspiration and Differing Factors

MacroME is based off of Project 14 from Spring 2020 of ECE 445: "Button Remapping for GameCube Games such as Super Smash Bros Melee" [3]. This project achieved similar goals by creating an adapter that remaps GameCube controller signals. The adapter sits between a standard GameCube controller and the console, and is programmed by connecting to a smartphone app via Bluetooth.

On the market, there are several custom GameCube controllers that allow for button remapping. For example, the B0XX [4] and SmashBox [5] controllers are fighting game styled controllers utilizing arcade buttons as their inputs and allow for custom button mappings. Additionally, in modern additions to the *Super Smash Bros'* franchise, there actually are button remapping capabilities built into the game. However, this falls short when trying to transfer those layouts to other games or consoles, since the layouts do not follow the controller itself.

MacroME differentiates itself from existing market solutions and the solution proposed in Project 14 in several key ways. First, MacroME contains all of the hardware within the form factor of a physical GameCube controller. Second, MacroME adds the functionality of programmable macros. Third, MacroME's process of programming the controller forgoes Bluetooth, opting instead for a wired connection. These differences allow our controller to be more portable by not requiring any setup when connecting to a new console, as well as adds functionality that is more beneficial to beginning and veteran players alike.

Lastly, MacroME's target price is significantly cheaper than the B0XX and SmashBox controllers, both of which are priced at around $200 [5]. Project 14 had a similar goal of having a much lower price compared to current products on the market.

1.3 High Level Requirements

- MacroME must have persistent memory so that the controller does not need to be reprogrammed each time it is disconnected from power.
- MacroME must have a maximum latency of 16.67 ms between button press and signal output, which is equivalent to less than 1 frame of latency at 60 frames per second [6].
- The GUI program and the MacroME controller must allow for macros with both analog stick and button inputs per frame, up to a length of 60 frames (1 second).

Figure 1: Physical Design of the MacroME Controller

1.4 Visual Aid

Figure 1 shows the physical design of the MacroME controller. The controller is very similar in appearance to a stock GameCube controller, but with two notable exceptions. First, MacroME has four extra buttons near the center of the controller. These are the macro buttons, and each can be programmed to perform a different macro string as defined by the user. Second, there is a USB-C port at the top of the controller (not visible). This USB-C port is used to communicate with the PC when the controller is being programmed.

1.5 Block Diagram

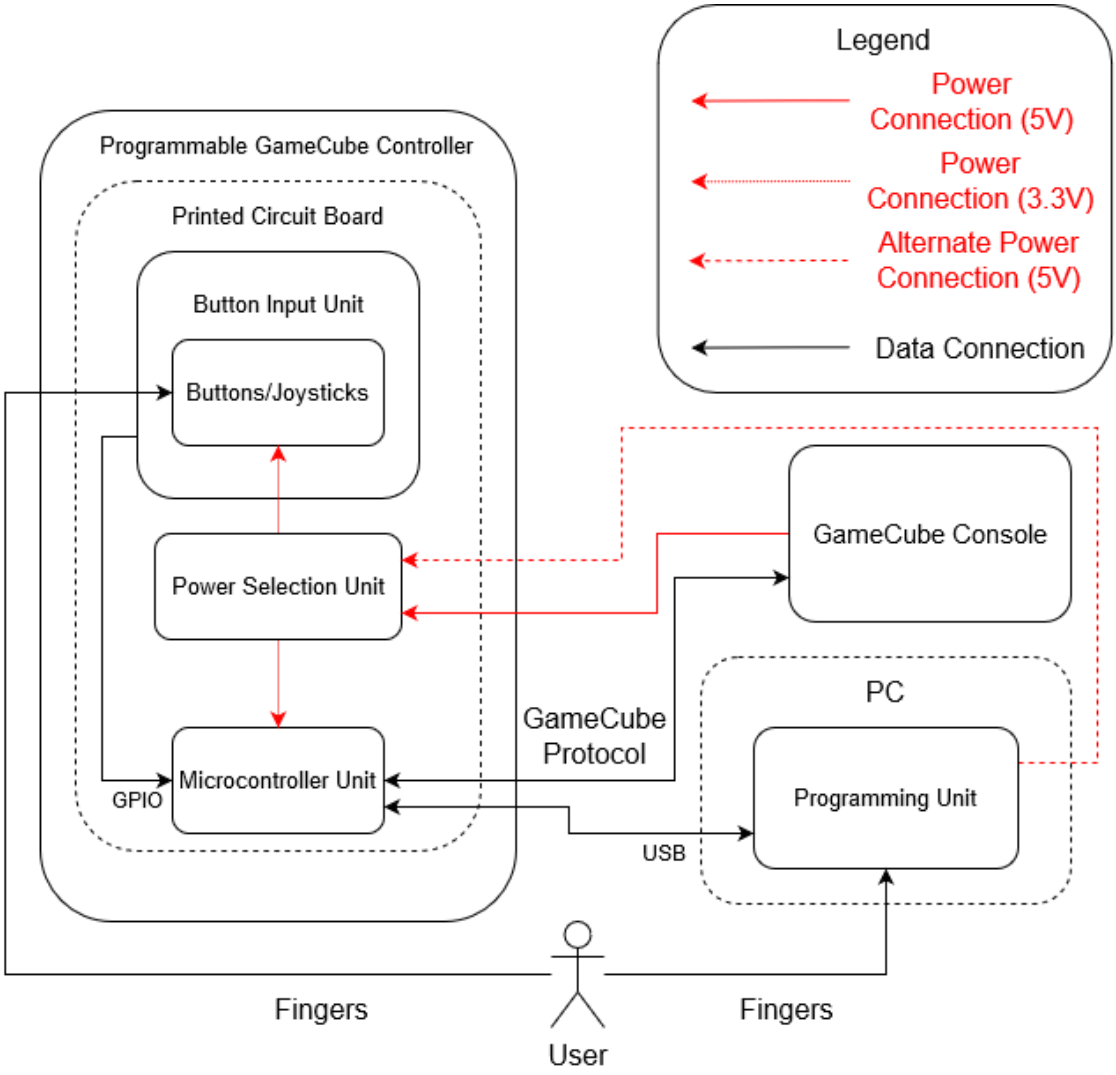

Figure 2: Block diagram of MacroME

Figure 2 shows the block diagram for 'MacroME'. The block diagram demonstrates typical operation of the controller. The user will program their remapped buttons and macros through the Programming Unit (PU) on the PC. The Microcontroller Unit (MCU) then is able to store these settings, and during gameplay translate the user's button inputs into the desired remaps or macros.

2 Project Implementation

The following section is a detailed explanation of the implementation of MacroME. There are overviews of the individual subsystems, which discuss their purposes, how they relate to the accomplishing the high level requirements, and the components used to implement them in the hardware. The hardware implementation is reflected by the circuit schematics and PCB layout sections. The circuit schematics show the implementation of the various subsystems in the circuit design itself. The PCB layout shows the placement of the components on the physical board. There is also a discussion of the software implementation, including the logic flowchart and module designs, as well as justifications for decisions made in the software design

2.1 Component Subsystems

In this section, we cover the individual subsystem units in detail. See Appendix B for each subsystem's requirements and verification tables.

2.1.1 Power Selection Unit

The Power Selection Unit (PSU) involves a simple circuit that chooses where the power to the microcontroller and peripherals comes from. The SAMD51 family of microcontrollers has a nominal input voltage of around 3.3V. This is perfect when connected to the GameCube console itself, since the GameCube outputs a 3.43V power line, but not ideal when the controller is connected to the PC via USB, as USB outputs a 5V power signal. Thus, the PSU is required to detect when USB is connected, and if so, use a linear voltage regulator to drop the voltage down to 3.3V for the microcontroller to use.

2.1.2 Button Input Unit

The Button Input Unit (BIU) includes the physical controller, the buttons, and the analog sticks on the controller. The BIU is the same size and shape as a standard GameCube controller, as through the BIU is how the user physically interacts with the game console. The raw inputs from the user will be sent to the Microcontroller Unit, which processes the inputs and makes any remappings or macro actions as necessary.

2.1.3 Microcontroller Unit

The Microcontroller Unit (MCU) is the main processing unit of the MacroME controller. It receives all inputs from the BIU, and outputs the remapped buttons or macros to the game console over the GameCube protocol (see Appendix C) within a single frame. The MCU has persistent memory, so that the stored button remapping and macros stay across power cycles, since a controller of this type will not be consistently powered. While in normal operation, the inputs to the MCU are the pressed buttons and analog sticks from the BIU. In programming mode, the MCU takes inputs from the PC application through USB.

Due to our familiarity with the platform and its relative power, we chose to use the SAMD51 family of ARM Cortex-M4 microcontrollers for the MCU [7]. For persistent memory, we use the GD25Q16C QSPI flash chip [8]. This chip is a 2MB flash storage chip that communicates with the SAMD51 Microcontroller over Quad SPI (QSPI), enabling MacroME to store user configurations even when the device is not powered.

2.1.4 Programming Unit

The Programming Unit (PU) allows the user to program the controller with different button mappings and macros. The user interacts with the Programming Unit through a GUI program on their PC that communicates with the MacroME controller through a USB connection. The macros that the user can program using the PU allow for any number of button presses at a time, with a resolution of 1 frame, for up to 60 frames.

2.2 Schematics

This section includes the circuit schematics for MacroME. The schematics are split up into various components. Each individual schematic block is connected to the others through global labels.

2.2.1 Flash Memory

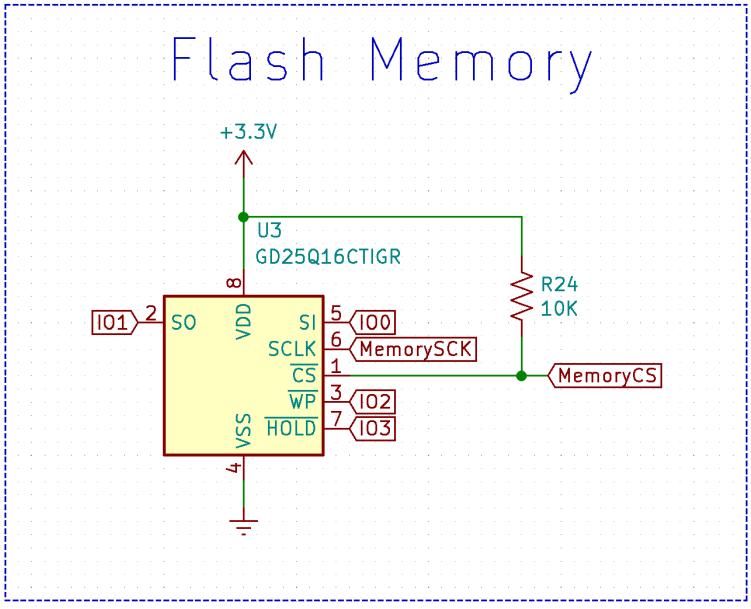

Figure 3: Circuit Schematic for Flash Memory

Figure 3 shows how the memory is connected to the microcontroller. The memory interfaces with the micro controller via the various IO pins. These help the microcontroller store and fetch the programmed button remaps and macros in the flash memory.

Figure 3 is also a representation of how the MCU fulfills its persistent memory requirement in the circuit design itself. With the addition of flash memory, the microcontroller can store configuration settings even when not powered. Therefore, the controller will not need to be reprogrammed between uses.

2.2.2 Voltage Regulator

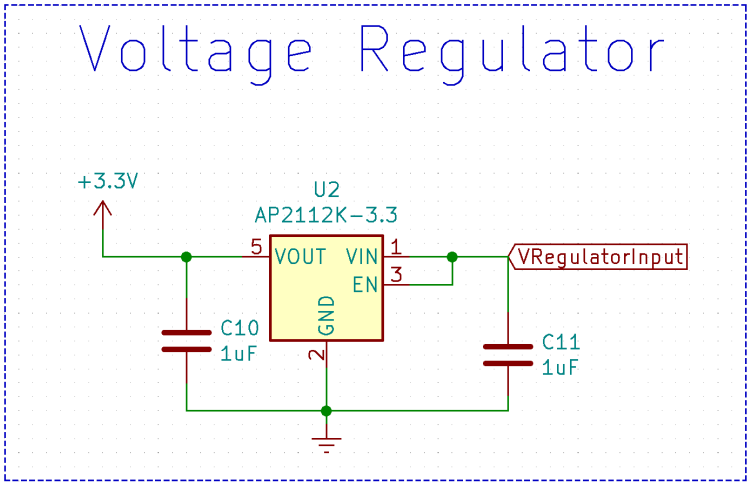

Figure 4: Circuit Schematic for Voltage Regulator

Figure 4 shows the circuit schematic of the voltage regulator. When the voltage regulator receives a +5V signal from the USB power, it outputs a +3.3V signal instead to the rest of the components.

Figure 4 is also a representation of the PSU in the circuit design itself. The voltage regulator is the central piece that drives the success of the PSU, and the circuit design reflects how it receives power inputs from the USB port or the GameCube controller before power selection. The capacitors connected to the various pins on the voltage regulator were selected based on the suggested connection setups in the AP2112K-3.3TRG1 datasheet [9]

2.2.3 Microcontroller and Inputs

Figure 5 shows the circuit schematic of the microcontroller and the button inputs. The microcontroller receives digital inputs from the buttons and analog inputs from the triggers and joysticks. All components are powered with 3.3V received from the voltage regulator. The microcontroller changes the inputs based the button remaps specified by the user, which are stored in the flash memory. The digital buttons are represented with switches and the analog joysticks are represented with analog inputs that map to the x and y values of the joystick position.

Figure 5 is also a representation of the MCU and the BIU in the schematics, as it includes the microcontroller with all the buttons and analog sticks. The microcontroller supply

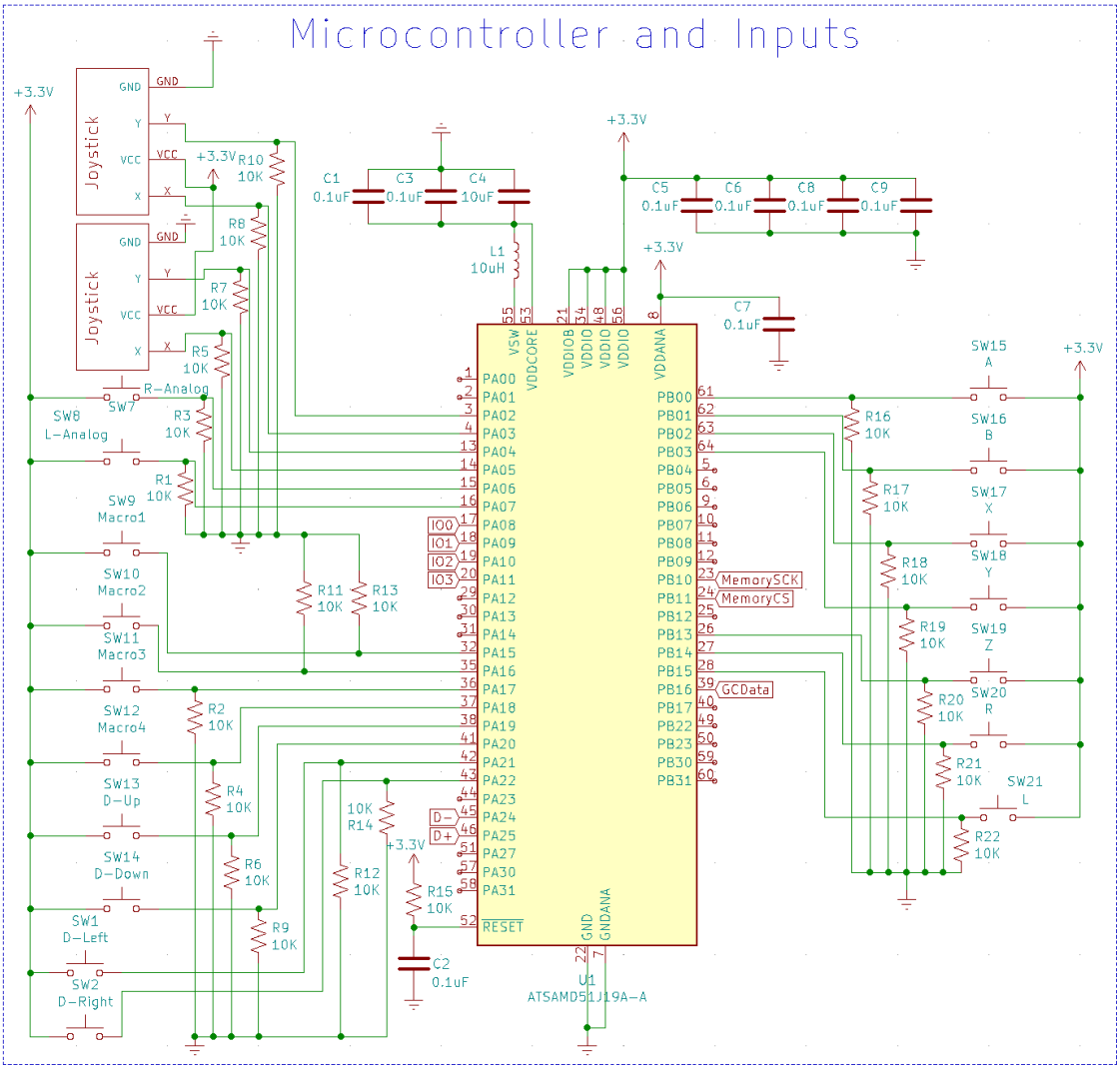

Figure 5: Circuit Schematic for Microcontroller and Inputs

pins were connected according to the connections suggested in the ATSAMD51J19A-AU datasheet [7].

2.2.4 Connectors

Figure 6 shows how the USB-C wire connects the controller to the PC. The program on the PC specifies button remaps and combination macros, which are saved on the flash memory. The microcontroller applies these changes to the user inputs. Data is transferred to the microcontroller via the D-/D+ pins.

Figure 7 shows how the GameCube console interfaces with the controller. The console is able to power the components on the board by supplying +3.3V when the controller is not being powered via USB. GameCube commands are sent to the console via the DATA line on the connector

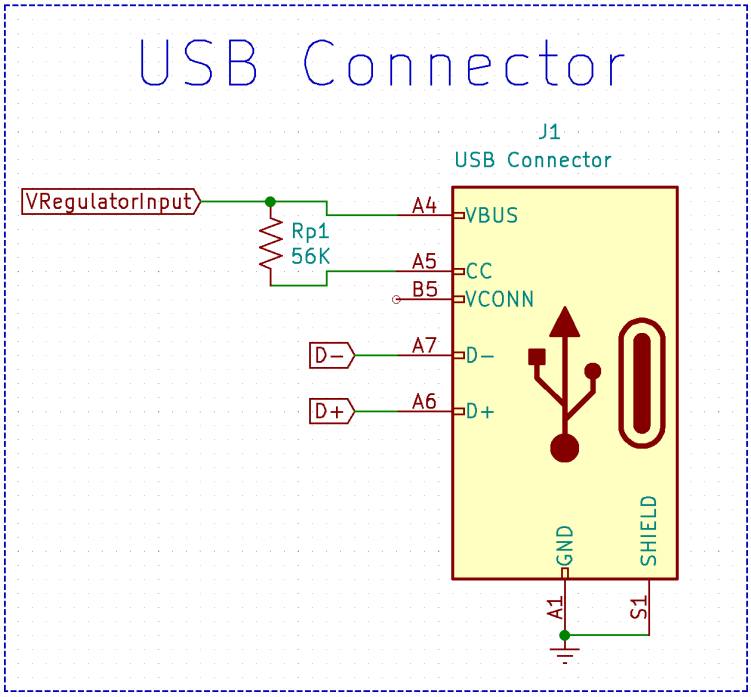

Figure 6: Circuit Schematic for USB Connector

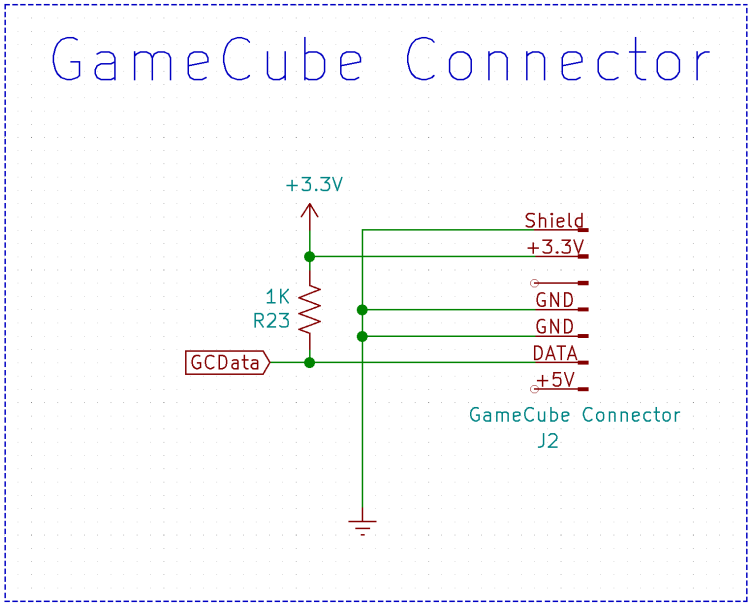

Figure 7: Circuit Schematic for GameCube Connector

Figures 6 and 7 are also representations in the circuit design itself of how the MCU connects to external devices like a PC which is running the PU or the game console.

2.3 PCB Layout

This section includes visual representations of the PCB layout, both on the KiCAD PCB editor, and on a 3D view for a more realistic representation of the final product [10]. For the KiCAD views, red represents copper on the front side, and green represents copper on the back side. For the 3D view, silver represents copper SMD pads, and some components have 3D models placed above their footprints.

2.3.1 Front Layout

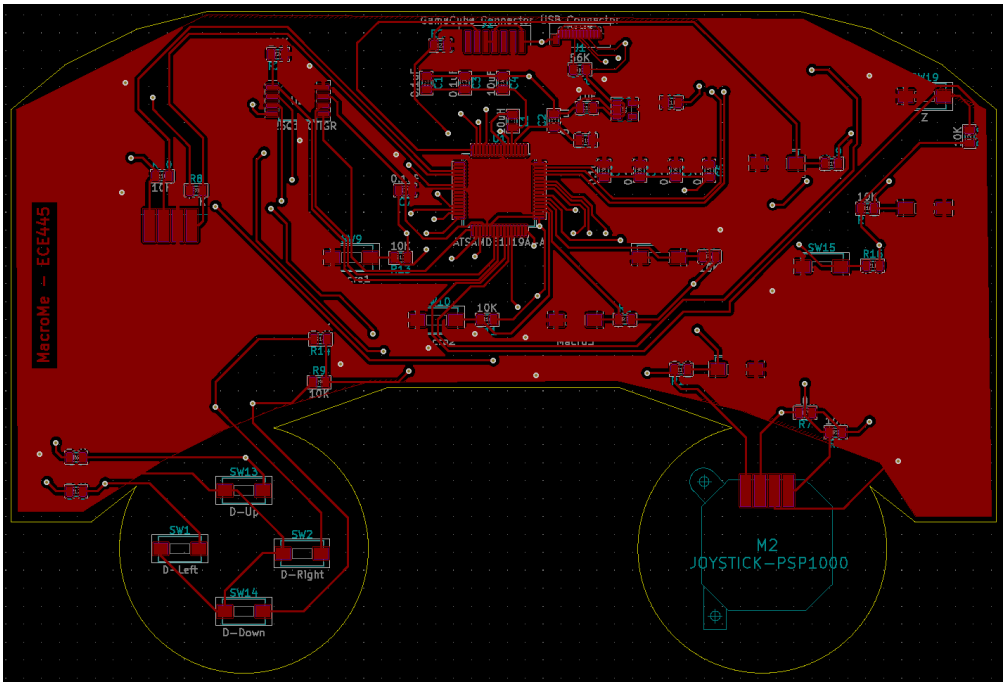

Figure 8: KiCad view of PCB front layout

Figure 8 shows a view of the component placement on the front of the PCB. Copper is represented by the color red in the figure. The parts, including the buttons, joysticks, and chips, are placed according to their position under the housing of a GameCube controller, in accordance with the high level requirements. There is a copper pour connected to +3.3V on the front of the board, allowing components to easily connect to power. This allows for easier routing between components as it eliminates the need for additional tracks for connecting to power, especially within the given space constraints.

Figure 9 shows a 3-Dimensional view of the PCB layout on the front of the PCB. The copper SMD pads are represented in silver, and there also exist some 3D models for certain chips and components. This figure provides an additional visualization of the parts, tracks, and copper pour on the front of the board.

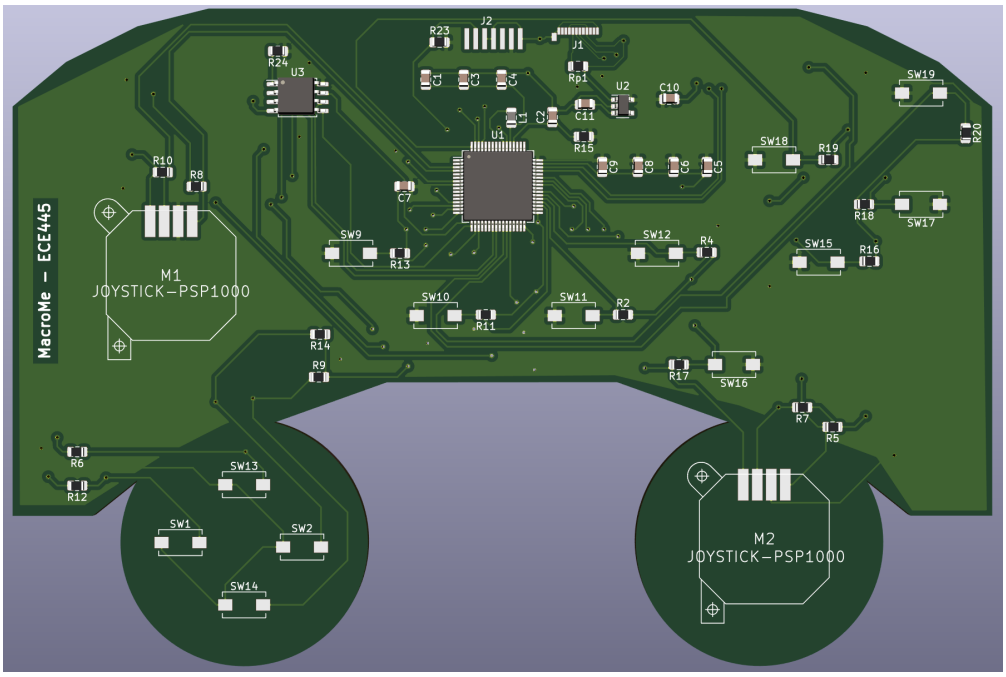

Figure 9: 3D view of PCB front layout

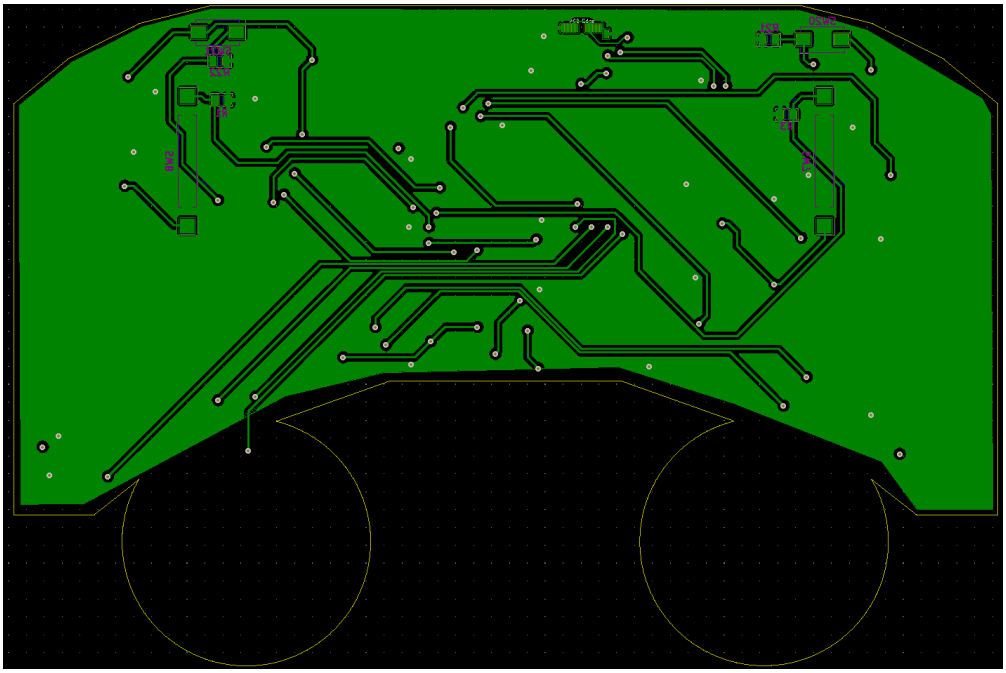

Figure 10: KiCad view of PCB back layout

2.3.2 Back Layout

Figure 10 shows a view of the component placement on the front of the PCB. Copper is represented by the color green in the figure. The analog and digital components of the

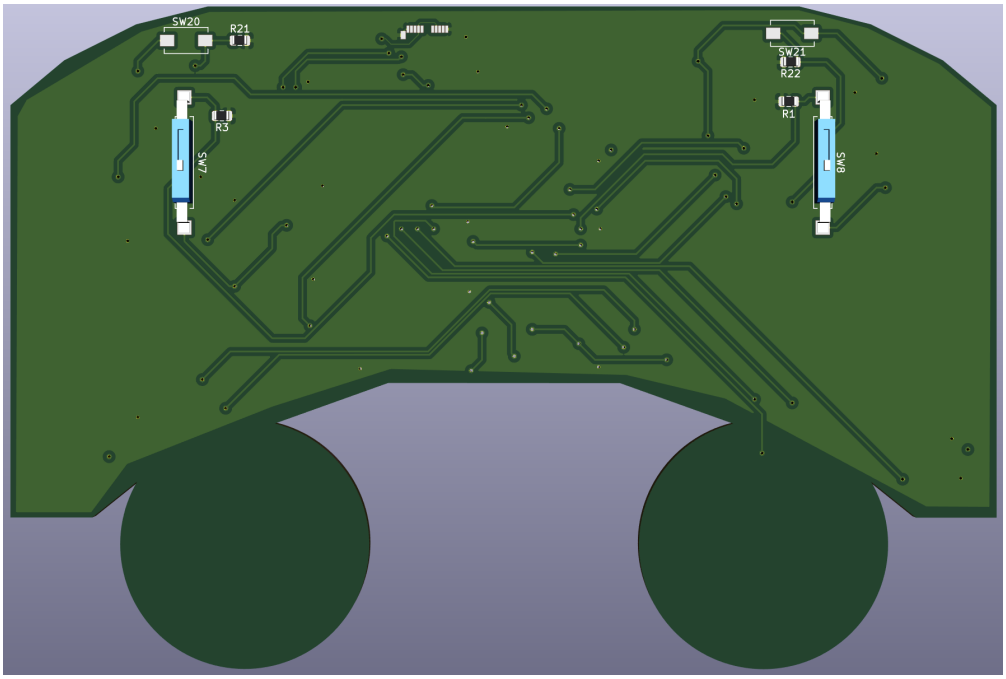

Figure 11: 3D view of PCB back layout

trigger buttons are also on the back. Some tracks are routed on the back due to space constraints on the front, or blockages from other copper tracks. There is also a ground pour on the back of the board, to allow parts to easily connect to ground. This also allows for easier routing between components as it eliminates the need for additional tracks for connecting to power, especially within the given space constraints.

Figure 11 shows a 3-Dimensional view of the PCB layout on the back of the PCB. The trigger buttons lie on the back of the board. A slider switch provides the analog values, and a normal button switch provides the digital value. The copper SMD pads are represented in silver. This figure provides an additional visualization of the tracks and copper pour on the back of the board.

2.4 Software Design

This section includes an in-depth discussion of decisions and designs for the software architecture of MacroME.

2.4.1 Choice of Programming Language

For the code running on the microprocessor, we decided to use the CircuitPython programming language [11]. CircuitPython is a lightweight implementation of Python for microprocessors, with specific support for the SAMD51 family of microprocessors sporting the ARM Cortex-M4. We chose CircuitPython for a few main reasons:

1. CircuitPython is well supported and documented on the Adafruit website and fo-

rums.

2. CircuitPython includes library support for utilizing the GD25Q16C flash memory chip through QSPI.
3. CircuitPython is easy to prototype with - there is support for directly flashing code through a USB connection to a PC allowing quick development and debugging.

Additionally, we had previously found that CircuitPython was fast enough to meet our latency requirements. The equations and mathematics for this latency analysis can be found in Appendix D.

Unfortunately, CircuitPython does have some drawbacks. The most important drawback with regards to this project is the lack of interrupts [12]. This means that we instead have to check on an incoming polling request from the console, as opposed to having an incoming poll request interrupt our processor for immediate action.

2.4.2 Logic Flowchart

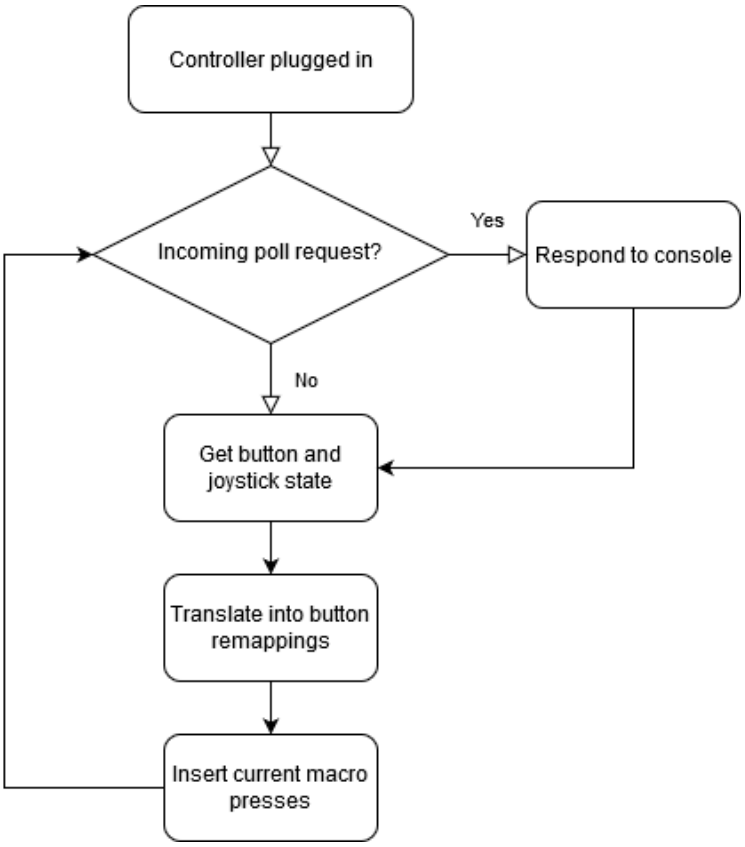

Figure 12: Software logic flowchart for the microcontroller

Figure 12 shows the logical flowchart of the software running on the SAMD51 microcontroller. Due to the lack of interrupts available in CircuitPython, the microcontroller will instead check for an incoming poll request from the console every loop, and if there is

one, immediately respond to the console with the translated button state. Otherwise, the micrcontroller will perform the normal routine of getting the button inputs, translating the inputs into the configured remappings, and inserting the current macro inputs that may be going on at that time.

2.4.3 Software Module Design

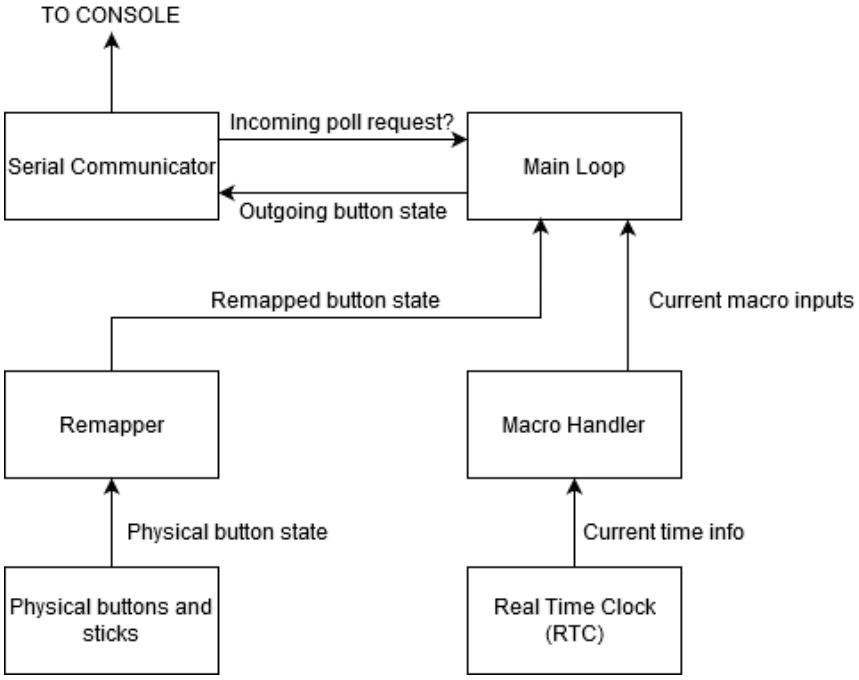

Figure 13: Software architecture for the microprocessor code

Figure 13 shows the software modules present on the microcontroller that handle the tranlating and macro functions while MacroME is connected to the GameCube console. In this figure, we can see how the logical flowchart in Figure 12 is implemented in the software. The Remapper and Macro Handler together create the modified button input data, and the Main Loop then sends that button data to the Serial Communicator. The Serial Communicator is responsible for converting the data into GameCube Protocol format, and send that information directly to the console. Note the addition of the Real Time Clock (RTC) as input to the MacroHandler. This is required so that the microcontroller can keep track of when one frame (16.67 ms) has passed and it is time to proceed to the next input of the macro sequence. The correct operation and use of the RTC is a key part of ensuring that our high level requirement of successfully performing frame-by-frame macros is met.

2.5 Materials and Parts

Table 1 shows the Bill of Materials for the prototyping of one MacroME unit.

Table 1: Bill of Materials

Part	**Manufacturer**	**Part #**	**Units**	**Unit Cost**
ARM Cortex-M4	Microchip	ATSAMD51J19A-AU	1	$4.09
GameCube Controller	Generic	-	1	$12.99
USB-C Connector	GCT	USB4085-GF-A	1	$1.37
QSPI Flash Chip	GigaDevice Semi	GD25Q16CTIGR	1	$0.51
3.3V Regulator	Diodes Inc.	AP2112K-3.3TRG1	1	$0.47
Custom PCB	PCBWay	-	1	$9.60
Misc. resistors, diodes, and capacitors	-	-	-	$5.00
Total	-	-	-	$34.03

3 Conclusions

In this section, we provide an overview of how we took steps to implement our design into a deliverable product. We discuss what were were able to accomplish, as well as future work that can be done to complete the full implementation of the product. Since this project was designed and implemented remotely due to the 2020 COVID-19 Pandemic, some elements in the design are left as uncertainties due to the lack of lab access and physical testing. We also discuss the ethical and safety considerations that were taken when designing MacroME.

3.1 Implementation Summary

In terms of the hardware, the circuit design and PCB design are completed. The PCB design has been done with the high level requirements and BIU requirements in mind, as the size of the board itself has been restricted to the under 140mm in width and 100mm in height – small enough to fit within the GameCube controller housing [13]. The completion of the PCB layout implements two of our three high level requirements (HLRs) in hardware. Our first HLR states that the controller must have persistent memory, included through the use of the QSPI flash chip. Additionally, the shape of the PCB layout assists in our third HLR. This HLR states that the GUI program and the controller must allow for macros with all buttons and analog sticks as inputs. The PCB design contains all the buttons and sticks of a standard GameCube controller, and therefore, all GameCube buttons and joysticks can be used for macros on MacroME provided that the software supports it.

Additionally, the software architecture and flowcharts logically describe the flow the software would have in order to fulfill our HLRs. We were able to decide on CircuitPython as our programming language, and work with the language's strengths and weaknesses to come up with a feasible software design that would achieve MacroME's requirements. However, since we do not have any of the hardware physically, we were unable to actually implement and test any of the software intended for the microcontroller.

3.2 Unknowns and Uncertainties

Due to the realities of the 2020 COVID-19 Pandemic, there are many uncertainties that persist in the implementation of MacroME due to the lack of resources, physical testing, and lab equipment. Some of our subsystem requirements, listed in Appendix B, require lab equipment such as multimeters that are not available remotely. Additionally, since we do not have access to soldering irons, PCB ordering, or any physical parts, we are unable to assemble a prototype of MacroME and test the different hardware and software components.

In terms of the microprocessor software, we are unable to test any of the code due to the lack of a physical unit. While we were able to create a detailed software design, we are ultimately unable to foresee any of the potential pitfalls and additional difficulties that could arise while programming the actual microprocessor. There are undoubtedly

nuances and details to implementing our software in CircuitPython that would require us to further tweak our design that we could only discover through the process of physically testing our software.

Lastly, we were unable to fully flesh out the design and implementation of the GUI program due to time constraints. In our original design document, this GUI program was specified as one of the lower priority items left for the later weeks of the project, and therefore was unable to be completed during the few weeks we had for the implementation phase of this project.

While we are confident in our design, we unfortunately have no way of verifying its success and feasibility without constructing and debugging a real-word prototype.

3.3 Ethics and Safety

3.3.1 Ethics

The main ethical question that comes up in the design of this project is whether the use of our controller would constitute a breach of competitive integrity. Many players believe that the use of controllers that are not standard GameCube controllers should be considered cheating. However, recent pushes towards more ergonomic and modern controllers have been made, such as allowing controllers such as the SmashBox [5] [14]. However, it is likely that the additional functionality of programmable macros would make this controller illegal in a tournament setting. The target audience of MacroME is beginners who are looking to begin learning higher-skilled techniques or play with friends in a casual setting, so we believe this to not be an issue.

In terms of players fraudulently representing MacroME as tournament legal, we do not believe it will be possible at any tournament that checks controllers. While MacroME does try to look as similar to a traditional GameCube controller as possible, there are extra buttons that will be on the controller for the macros, and final products will also have MacroME branding. Thus, it would be impossible to misrepresent this controller as an unmodified GameCube controller and sneak it into tournaments. To comply with points 1 and 9 of the IEEE Code of Ethics [15], information clearly stating that MacroME is not a tournament legal controller will be disclosed to the public.

Additionally, we plan to make both the software and hardware design of MacroME open-source, such that the all members of the public may benefit from the design knowledge gained throughout this project, supporting point 8 of the IEEE Code of Ethics. We would also be able to accept criticism and suggestions relating to our technical work, in accordance with points 5 and 7 of the IEEE Code of Ethics.

3.3.2 Safety

There are few safety considerations for this product. Because of its nature as a video game controller, all the systems are at low voltage and current. Because of this, the risk of injury due to the electrical systems is extremely low and at worst would cause only small shocks.

Still, information will be provided on the cable to inform users not to use the controller while in wet environments. Additionally, the controller is of small size and weight, thus the likelihood of serious injury from dropping it on a foot or other body part is very low. Our main safety considerations are for the students and workers during the design and prototyping process. The design, prototyping, and manufacturing processes utilize tools that can be dangerous, so we will make sure that while soldering, taking apart GameCube controllers, and taking part in other lab activities, all students will adhere to strict safety standards as advised by the course staff and the ECE department.

3.4 Future Work and Project Improvements

Given a year to complete this project instead of a few weeks, there are several improvements that can be made both on the design and implementation of this project.

1. The GUI Program would need to be implemented and improved with visual confirmation of which buttons were being remapped to which. This could be done by a diagram of the MacroME controller in the application, with different colors on different buttons based on their remapped values. It would also be good to make the GUI program cross platform so it could be run on Windows, Mac, and Linux.
2. With a year to complete this project, there would be more time to try different microprocessors in order to find one that is more adequately priced for the use case. In this project, our choice of microprocessor is drastically overpowered as seen in our latency analysis (Appendix D). If we could do further analysis on various cheaper and less powerful processors, we could better price our product instead of spending on a microprocessor whose features we are not taking full advantage of.
3. Since MacroME has a USB connection, additional functionality could be added to allow MacroME to be used as a XInput controller over USB [16]. To choose whether MacroME is in programming mode or controller mode when connected to USB, a physical switch added to the top or center of the controller could be used. This addition would allow MacroME to be natively compatible with some other consoles such as XBox and PC.

With these improvements, MacroME could become a more sophisticated, fully-featured, and price effective product.

References

[1] Game Informer Staff. (May 2019). "The 30 Greatest Fighting Games of All Time", [Online]. Available: https://www.gameinformer.com/2019/04/25/the-30-greatest-fighting-games-of-all-time (visited on 04/01/2020).

[2] B. Bernstein. (May 2019). "20 Best Fighting Games of All-Time: The Ultimate List", [Online]. Available: https://heavy.com/games/2015/01/best-fighting-games/ (visited on 04/01/2020).

[3] M. Qian, S. Yaganti, and Y. Wu. (2020). "Button Remapping for GameCube Games such as Super Smash Bros. Melee", [Online]. Available: https://courses.engr.illinois.edu/ece445/getfile.asp?id=16677 (visited on 04/01/2020).

[4] 20XX. (2020). "B0XX Controller", [Online]. Available: https://b0xx.com/ (visited on 04/01/2020).

[5] Hit Box Arcade. (2020). "Smash Box", [Online]. Available: https://www.hitboxarcade.com/pages/smash-box (visited on 04/01/2020).

[6] Smash Wiki Community. (Mar. 2020). "Frame", [Online]. Available: https://www.ssbwiki.com/Frame (visited on 04/02/2020).

[7] Microchip Technology. (2019). "SAM D5x/E5x Family Datasheet", [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/60001507E.pdf (visited on 04/11/2020).

[8] GigaDevice Semiconductor. (May 2016). "GD25Q16C Family NOR Flash", [Online]. Available: https://www.gigadevice.com/flash-memory/gd25q16c/ (visited on 04/14/2020).

[9] Diodes Incorporated. (Jun. 2017). "AP2112K-3.3TRG1", [Online]. Available: https://www.diodes.com/assets/Datasheets/AP2112.pdf (visited on 05/05/2020).

[10] Kicad Developers Team. (2020). "KiCad EDA", [Online]. Available: https://kicad-pcb.org (visited on 05/07/2020).

[11] Adafruit Industries. (2020). Circuitpython, [Online]. Available: https://circuitpython.org/ (visited on 05/01/2020).

[12] L. Fried. (Dec. 2018). "Comment on Issue #19 in Adafruit_CircuitPython_TSL2561", [Online]. Available: https://github.com/adafruit/Adafruit_CircuitPython_TSL2561/issues/19#issuecomment-447613493 (visited on 05/07/2020).

[13] Dimensions.guide. (Apr. 2020). "GameCube Controller", [Online]. Available: https://www.dimensions.guide/element/gamecube-controller (visited on 05/06/2020).

[14] J. Cuellar. (Apr. 2018). "Mr. Wizard on the Smash Box", [Online]. Available: https://www.eventhubs.com/images/2018/apr/17/mr-wizard-smash-box/ (visited on 04/01/2020).

[15] IEEE. (2016). "IEEE Code of Ethics", [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html (visited on 02/08/2020).

[16] Microsoft Corporation. (May 2018). Getting started with xinput, [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/xinput/getting-started-with-xinput (visited on 05/08/2020).

[17] Smash Wiki Community. (Nov. 2019). "Wavedash", [Online]. Available: https://www.ssbwiki.com/Wavedash (visited on 04/01/2020).

[18] ——, (Mar. 2020). ”Smash Directional Influence”, [Online]. Available: https://www.ssbwiki.com/Smash_directional_influence (visited on 04/01/2020).
[19] C. Wang. (Dec. 2015). ”GameCube Controller Interface”, [Online]. Available: https://os.mbed.com/users/christopherjwang/code/gamecube_controller/ (visited on 04/01/2020).
[20] J. Ward. (Mar. 2004). ”Nintendo GameCube Controller Protocol”, [Online]. Available: http://www.int03.co.uk/crema/hardware/gamecube/gc-control.html (visited on 04/01/2020).

Appendix A Example Combos

To clarify the types of difficult combos that programmable macros help make easy, two examples of complex maneuvers are described in this appendix.

Wavedashing is a technique that can be performed in *Super Smash Bros. Melee* that involves performing an air dodge diagonally into the ground, causing the character to slide a short distance [17]. It has become considered an essential technique for *Melee* gameplay, but it is difficult for beginners to consistently pull off the precise inputs.

Smash Directional Input (SDI) is a technique that can be performed in all *Super Smash Bros.* games that involves the player repeatedly inputting a control stick direction while getting hit by an attack, thus slightly altering their character's position and allowing their character to escape possible follow-up attacks [18]. Performing optimal SDI requires the player to input a new control stick input each frame, which is both unrealistic for beginners and causes unnecessary wear-and-tear on the controller.

Appendix B Requirements and Verification Tables

In this appendix are the requirements and verification tables for each subsystem.

Table 2: PSU Requirements and Verifications

Requirement	**Verification**
1. The PSU must correctly switch between USB and GameCube power, such that the output power to the microcontroller is always around 3.3V (± 0.2V), and never greater than 3.6V.	A. Connect the PSU to a GameCube console via a GameCube cable. Use a multimeter to ensure that the output from the PSU is less than 3.6V and around 3.3V (± 0.2V). B. Connect the PSU to a PC using a USB cable. Use a multimeter to ensure that the output from the PSU is less than 3.6V and around 3.3V (± 0.2V). C. Connect the PSU both to a GameCube console via a GameCube cable and to a PC via USB. Use a multimeter to ensure that the output from the PSU is less than 3.6V and around 3.3V (± 0.2V).

Table 3: BIU Requirements and Verifications

Requirement	**Verification**
1. The buttons, printed circuit board, joysticks, and housing for the BIU must be no larger than the size of a standard GameCube controller.	A. Place the PCB populated with all components and hardware inside the shell of a GameCube controller. Verify that the housing closes over the PCB.

Table 4: MCU Requirements and Verifications

Requirement	**Verification**
1. The MCU must be able to read currently pressed buttons, translate to remapped inputs, and output the remapped inputs and/or macros in under 16.67 ms (1 frame of latency). 2. The MCU must have persistent memory, so that it does not need to be reprogrammed every time it is powered up.	A. Use the microcontroller's built in clock to time the processing of button inputs, verifying that the reported time is less than 16.67 ms at least 19/20 times. B. Program the controller, then restart it at least 5 times. Verify after each restart that the designated remappings and macros are still correctly outputted.

Table 5: PU Requirements and Verifications

Requirement	**Verification**
1. The PU must allow button remapping through the GUI. 2. The PU must allow for up to all buttons and analog sticks to be pressed during each frame of macro input. 3. The PU must allow for a maximum macro length of 60 frames, equivalent to 1 second of automated input.	A. Connect the controller to the GUI and enter several (>5) button remaps and macros. Verify that the controller outputs a signal with the remapped controls. B. Program a macro that includes all buttons being pressed during one frame. Verify that the controller outputs a signal that includes all buttons being pressed for one frame. C. Program a 60 frame macro. Verify that the controller outputs a signal that correctly performs the macro at each frame.

Appendix C The GameCube Protocol

The GameCube protocol is a kind of serial communication using a 3.3V bidirectional data line. Communication is initiated by the console sending a 24-bit string to the controller, after which the controller responds with 8 bytes of analog input and button data. Each string of bits is terminated by an extra, single (high) stop bit [19].

Table 6: GameCube Controller Response Protocol [19]

Byte 0	0 0 0 Start Y X B A
Byte 1	1 L R Z Up Down Right Left
Byte 2	Joystick X-Value
Byte 3	Joystick Y-Value
Byte 4	C-Stick X-Value
Byte 5	C-Stick Y-Value
Byte 6	Left Trigger Value
Byte 7	Right Trigger Value

The console polls the controller roughly every 6 ms, however, the actual polling rate is set by the individual game [19][20]. When the controller polls, it sends a 24-bit string `0100 0000 0000 0011 0000 0000`, followed by the single high stop bit. The controller must then respond with an 8 byte string, followed by the single high stop bit. The details of this response string are detailed in Table 6. The transfer speed is around 4 microseconds per bit, or a baud rate of 256000 bits per second.

Appendix D Latency Analysis

One of the components that is vital to the success of MacroME is that all of MacroME's processings adds up to less than one frame of input lag. This means that starting from button input, MacroME must see that input, translate it into the desired remapping, and output that button press along with any macros that are currently executing all within 16.67 ms. Understanding the amount of time and microcontroller clock cycles we have is crucial to writing software that can perform all these tasks within the allotted time frame.

The GameCube console polls for controller values roughly every 6 ms [20]. During each of these polls, our SAMD51 microcontroller will have to respond to the GameCube console with an 8 byte sequence according to the GameCube protocol indicating the current state of the buttons. The GameCube console communicates at a baud rate of 256000, so each bit takes around 3.9μs to send [19]. Only accounting for the communication with the GameCube console, this takes up

$$3.9\mu s \times (24 + 1 + 8 \times 8 + 1)\,\text{bits} = 351\mu s \tag{1}$$

In equation 1, the 24 is the polling request from the console, while the 8×8 is the 8 byte response from MacroME. We add 1 to each of these, since the GameCube protocol calls for a high 1 at the end of a string sequence.

Subtracting this "blocked" time from our total processing time, rounded up to three sets of polling per frame, we get

$$16.67 \times 10^3\mu s - (351\mu s \times 3) = 15.617 \times 10^3\mu s = 15.617\text{ms} \tag{2}$$

From equation 2, we see that MacroME has 15.62 ms of time, per frame, to perform its essential functions. The SAMD51 sports a 120 MHz ARM Cortex-M4 [7], so this amount of time is equivalent to

$$15.62 \times 10^{-3}\text{s} \times \frac{120 \times 10^6\text{cycles}}{1\text{s}} = 1874400\text{cycles} \tag{3}$$

From equation 3, we find that, in order to successfully meet our requirement of being responsive within one frame of input lag, the software for the microcontroller must be able to completely run within 1,874,400 clock cycles.

Within these 1,874,400 clock cycles, our code needs to execute the following general steps.

1. Read all the inputs from the buttons.
2. Translate those buttons based on the configured button remapping.
3. If there is a macro being executed, include those button inputs.
4. Prepare the data into GameCube protocol ready to be sent to the console.

We can further break down these actions into individual code operations that we can test individually.

1. 18 GPIO reads (1 per button and analog input).
2. Up to 18 variable reads and writes.
3. Read and update current macro index (1 variable read/write), read macro string at that index (1 array read), and up to 18 variable read/writes.
4. Translate 18 variables into GameCube protocol string.

Table 7 shows the speed of these various code blocks. The numbers in table 7 are found using an Adafruit Metro-M4, which utilizes the same SAMD51 ARM processor as we do, running the above code blocks using CircuitPython. Note that, due to the overhead of running a Python interpreter on a Microprocessor, the compiled C versions of these code blocks will likely be significantly faster.

Table 7: CircuitPython Cortex-M4 Empirical Code Speed

Action	**Execution Time (μs)**
GPIO Read	9.21
Variable R/W	4.42
Read from list index	2.19
Translate to GameCube Protocol (18 variables)	138.23

Combining these together,

$$9.21 \times 18 + 4.42 \times 18 + 4.42 + 2.19 + 4.42 \times 18 + 138.23 = 469.74\mu\text{s} \tag{4}$$

This resulting time of 469.74 μs is a mere 3% of the available time, allowing us plenty of time to complete our computations during each frame.