ECE 445

SENIOR DESIGN LABORATORY

DESIGN DOCUMENT

MacroME: The Programmable GameCube Controller

Team #67

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

TA: Chi Zhang/Megan Roller

April 17, 2020

Contents

1 Introduction

1.1 Objective

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.

We propose "MacroME: The Programmable GameCube Controller," a fully functional Nintendo GameCube controller that is able to be programmed with remappable buttons and macroinstructions (macros) of multiple button presses. This controller will look and feel like a standard GameCube controller, but will allow the user to program button remaps or combination macros onto the controller via a GUI on a PC.

We hope that MacroME will encourage beginner players to try the more complex characters in the *Super Smash Bros.* games, and make entry to competitive play less intimidating. In addition, 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.

1.2 Solution Overview

MacroME is a programmable controller that looks like a standard GameCube controller visually, but with additional features. 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 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.

1.3 Background

1.3.1 Inspiration

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.

1.3.2 Example Combos

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

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 [6]. 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 [7]. 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.

1.4 Visual Aid

Figure 1: Layout of a standard Nintendo GameCube Controller

Figure 1 shows the standard design of a GameCube controller [8]. MacroME will utilize a standard GameCube controller shell, and as such will look very similar to existing official and third-party controllers. However, there will be extra buttons for macros, as well as an extra port at the top of the controller for a USB-C connection.

1.5 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 [9].
- 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).

2 Design

2.1 Block Diagram

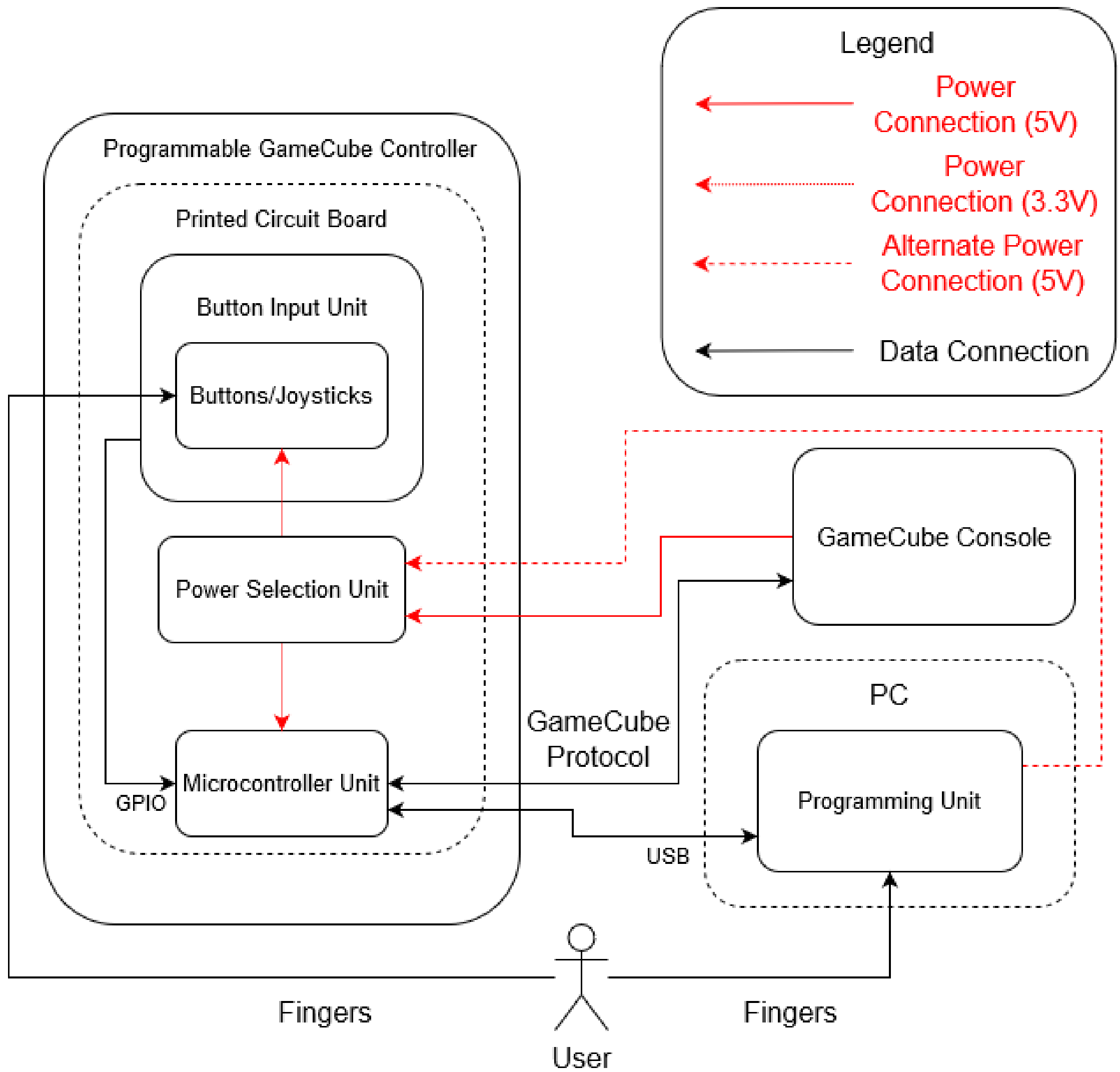

Figure 2: Block diagram of MacroME

Figure 2 shows the proposed 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 remappings or macros.

2.2 Physical Design

Figure 3: Physical Design of the MacroME Controller

Figure 3 shows the proposed physical design of the MacroME controller. The controller is very similar to the image in Figure 1 of 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 will be used to communicate with the PC when the controller is being programmed.

2.3 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.

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).

2.4 Button Input Unit

The Button Input Unit (BIU) includes the physical controller, the buttons, and the analog sticks on the controller. The BIU will be 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.

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.

2.5 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 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 [10].

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.

2.5.1 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 [11].

Table 1: GameCube Controller Response Protocol [11]

Byte 0	0 0 0 Start Y X B A
Byte 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 [11][12]. 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 1. The transfer speed is around 4 microseconds per bit, or a baud rate of 256000 bits per second.

2.5.2 Persistent Memory

For persistent memory, we will use the GD25Q16C QSPI flash chip [13]. This chip is a 2MB flash storage chip that communicates with the SAMD51 Microcontroller over Quad SPI (QSPI). This enables MacroME to store user configurations even when the device is not powered.

2.6 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.

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.

2.7 Tolerance 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 [12]. 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 [11]. 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 [10], 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. With a reasonable amount of care for the speed of our embedded software, this is more than enough clock cycles for our functionality.

3 Costs

3.1 Labor

The cost of engineering development for MacroME is estimated at $50 per person (total of 2 people), 10 hours per week, for the semester (a total of 15 weeks). With an extra 2.5× modifier for exigent circumstances, this totals to $37,500.

In terms of manufacturing labor, this project requires a few holes cut into an existing GameCube controller shell in order to allow for extra buttons. The ECE Machine Shop [14] estimated an afternoon's worth of work for one machinist, which is $50 per hour for 3 hours, coming out to $150.

3.2 Materials and Parts

Table 2: 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

3.3 Total Costs

The sum of the bill of materials (Table 2) and labor costs comes out to $37,684.03. This cost is for the development of the project and the prototyping of 1 unit.

4 Schedule

Table 3 shows the team schedule over the course of the semester.

Table 3: Schedule for the team

Week	Task	Who
1	Begin design of overall circuit schematics	Rajan
	Implement the GameCube protocol on button presses in software	Biskup
2	First draft of PCB Layout	Rajan
	Implement programming mode with USB connection to PC	Biskup
3	Finalize PCB layout and print PCB for prototyping	Rajan
	Create a terminal program to allow for button remapping	Biskup
4	Solder parts onto PCB	All
	Place PCB into controller shell and test connections	Rajan
	Implement and test persistent storage using flash memory	Biskup
5	Test communication between PU and MCU and verify requirements for PU	Rajan
	Implement programmable macros in software	Biskup
6	Test latency requirements for MCU and communication with game console	Rajan
	Create a GUI wrapper for the terminal program for PC	Biskup
7	Revise PCB design and order again if necessary	Rajan
	Bug fixes and various improvements	Biskup
8	Mock Demo	All
	Troubleshoot issues and ensure all requirements are met	All
9	Final Demo	All
	Final testing before demo	All
10	Final Presentation	All
	Prepare for final presentation	All

5 Ethics and Safety

5.1 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] [15]. 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. Thus, it would be impossible to misrepresent this controller as an unmodified GameCube controller and sneak it into tournaments.

Additionally, we plan to make both the software and hardware design of MacroME open-source, such that the public may benefit from the design knowledge gained throughout this project, as well as accept criticism and suggestions of our technical work, in accordance with points 5 and 7 of the IEEE Code of Ethics [16].

5.2 Safety

There are few safety considerations for this project. Because of its nature as a video game controller, all the systems are at low voltage and current. 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 during the design and prototyping process. 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.

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. (Nov. 2019). ”Wavedash”, [Online]. Available: https://www.ssbwiki.com/Wavedash (visited on 04/01/2020).

[7] ——, (Mar. 2020). ”Smash Directional Influence”, [Online]. Available: https://www.ssbwiki.com/Smash_directional_influence (visited on 04/01/2020).

[8] E. Amos. (Aug. 2010). ”Nintendo GameCube Photo History”, [Online]. Available: https://commons.wikimedia.org/wiki/User:Evan-Amos/VOGM/GameCube (visited on 04/14/2020).

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

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

[11] C. Wang. (Dec. 2015). ”GameCube Controller Interface”, [Online]. Available: https://os.mbed.com/users/christopherjwang/code/gamecube_controller/ (visited on 04/01/2020).

[12] 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).

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

[14] Electrical and Computer Engineering Department. (2020). ”Machine Shop”, [Online]. Available: https://ece.illinois.edu/directory/office/10 (visited on 04/11/2020).

[15] 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).

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