1 Cover Page

2 Table of Contents

3 Introduction (about 3-5 pages)

The problem being solved.
High level description of your solution.
How well did your design meet your expectations? Does it do what you thought it would, is it as accurate as you thought it would be, etc?

4 Detailed System Requirements (several pages)

This is the details list of the requirements of your design such that it solves the problem you are addressing

The most basic requirement for our project is the shape and size necessary for it to be used. The final board we print must be the correct shape to plug into a Gamecube memory card slot, while also being lightweight and small enough that it will not sag while hanging out of the slot (realistically probably a maximum of a few ounces). Ideally, it would be small enough to fit into a gutted memory card casing, but that is unrealistic. Instead, we aim to print a board as small as possible and then 3D print a plastic casing that mounts it properly into the memory card slot.

Ease of use is a high priority for the design of this project, so the goal is for the chip to be “plug and play”. This means that to be used, our chip only need be plugged into the memory card slot, and it would do the rest of the work from there. Additionally, for the “internet of things” scaling of this project where many chips plugged into Gamecubes all put data one website, which will require monitoring by an overseer. In this case, the ideal scenario is that our website could support somewhere upwards to fifty devices, which is about the maximum number of Gamecubes used at once in the early stages of the largest tournaments. That is a bit more of a stretch goal, and even the ability to support roughly five systems at once would be sufficient to track the matches of all MIOM top 100 ranked players (the Melee equivalent of the AP poll) at a given tournament, which is where essentially all the demand for statistics lies. Most tournaments take place in a ballroom-type space, so to support a tournament-wide network of devices, the supported range would need to be 100-200 feet for the largest tournaments. Once again if we scale down to only track top level players, the range would only need to be 50 feet or even smaller if the device is nearby a Wifi router.

The ultimate user interface we have in mind right now is a MATLAB GUI that would interpret database information and display it for a user. The database information will come from a Hadoop system on a Raspberry Pi that will act as the hub for all the Wi-Fi communication.

The embedded intelligence will need to interpret the data on the EXI interface. It will have to accomplish this without modifying the gameplay in any way, as modifications of the game that adversely affect gameplay are not accepted in the competitive community, so the more invasive the project is, the less likely it will actually be used. This data will be output from

Gamecube memory in the form of assembly code, so we will need a way interpret this assembly code and put it in a form we can work with. It will need to read all of the assembly code, looking for triggers related to the relevant data we desire, and format text string output that can be passed over SPI. It is important that all irrelevant game data is weeded out on the chip that plugs into the memory card slot before being transmitted over Wi-Fi, so that we minimize the amount of data that needs to be sent.

Since the board would be operated in conjunction with a GameCube, and does not need to be portable or mobile, it can be plugged into the wall with a DC power jack. The memory card pin operates at a 3.3 V range, and the DC power jack can certainly be regulated to this.

5 Detailed project description (20+ pages)

5.1 System theory of operation (how the whole thing works)

5.2 System Block diagram

Overall system block diagram showing how it is divided into subsystems, and the interfaces between the subsystems

5.3 Memory card slot SPI data acquisition and statistics creation

Subsystem requirements. These are the requirements that this particular subsystem must meet. (The combination of all of the subsystem and interface requirements must encompass the overall system requirements.)
This should include a schematic (hardware) or flow chart (software) for the subsystem, the function of the subsystem, the interfaces to other subsystems, etc. There should also be information about why you made the engineering decisions that you did – choice of components, kind of interface, choice of programming language, etc. Schematics and software need to be described in addition to having a schematic or a code listing. For hardware, you should describe how your circuit works (for example, how your amplifier circuit and filters are designed/operate); for software, you should describe the overall flow of the code (is it interrupt driven, state diagrams, etc.)
If there are communications protocols involved in this subsystem, description and state transition diagrams should be included.
Subsystem Testing. Described how this subsystem was tested to ensure functionality

5.3.1 Data output from console

The Gamecube internal memory is constantly updated with large amounts of information relating to the state of the game. This is where all the relevant data relating to the statistics originates. The problem is that, in general, aggregate descriptions or statistics of this data are not made available to the player by any means. In order to make game data available to an

external observer of the system in real time, it is necessary to utilize Assembly code that modifies the operation of the game.
With a Gamecube game like Super Smash Bros. Melee, one can write custom code that performs an extra function and is installed into the game essentially as a “cheat code”. This functionality is typically used to do simple and noninvasive things such as adding new skins for characters and stages, or using different songs for the background music of the game. These types of things are only possible due to intensive reverse-engineering that has been performed by many hobbyists over the years to understand the way that Melee’s code functions. In the case of this project, code modification can be added that sends a more comprehensive set of game data to the memory card slot to be later analyzed.
A generous member of the Melee community provided us with the code necessary to be added into the game in order to get all the relevant information to make any statistic we want. This information includes each character in the game, which action they are performing, what the previous move was that they managed to land on an opponent, the previous move they were hit by, their “percent” (Melee’s version of health), the state of their shield, the buttons currently being pressed by their user, how many lives they have remaining, and how many moves they have consecutively landed in a “combo” on their opponent, if applicable. All these pieces of data need to be read by our device in order to enable the creation of accurate and useful statistics. This code needs to send its data to memory card slot B, while memory card slot A is still used for the memory card save data. The data sent to memory card slot B is a raw string of hex, but it is sent in a specific order. All the components of game data mentioned above are sent in sequence to the memory card slot to be read by hardware plugged into it.

5.3.2 Reading game data through SPI bus

Once the proper game data is being sent by the system to memory card slot B, the task at hand is to actually acquire that data in our device. This will be done through SPI communication, as the gamecube memory card slots are EXI buses. EXI is simply an SPI protocol that operates in the (0,0) mode, which means the clock is low when idle and the data is sampled on the rising edge of the clock. Below in Figure 1 is the pinout for a normal Gamecube memory card, showing which pins are involved in the SPI communication.
In order to read this SPI bus, our board will need to have pins that feed into the slot in the same manner as a regular memory card, and then connect the relevant pins to the microcontroller that will process the data. For purposes of testing and development, we need a breakout board that brings these pins out so we can use jumper cables to connect them to our development board. As the final product will be one board, bringing these pins out will simply just be a component of it and there will be no need for cables. In both cases, the only strict requirements are that the board has the proper shape to fit into and rest inside the memory card slot, as well as the correct placement of conductors to line up with the pins inside the slot.
In order to connect to the SPI, our microcontroller only needs to be connected to four of the pins provided to the memory card slot: common ground, DO (data out), CS (chip select) and clock. The ground, chip select, and clock should be connected to the corresponding pins of the SPI

being used on the microcontroller, and the data out pin should be connected to the MOSI (master out slave in) pin of the SPI used on the microcontroller.

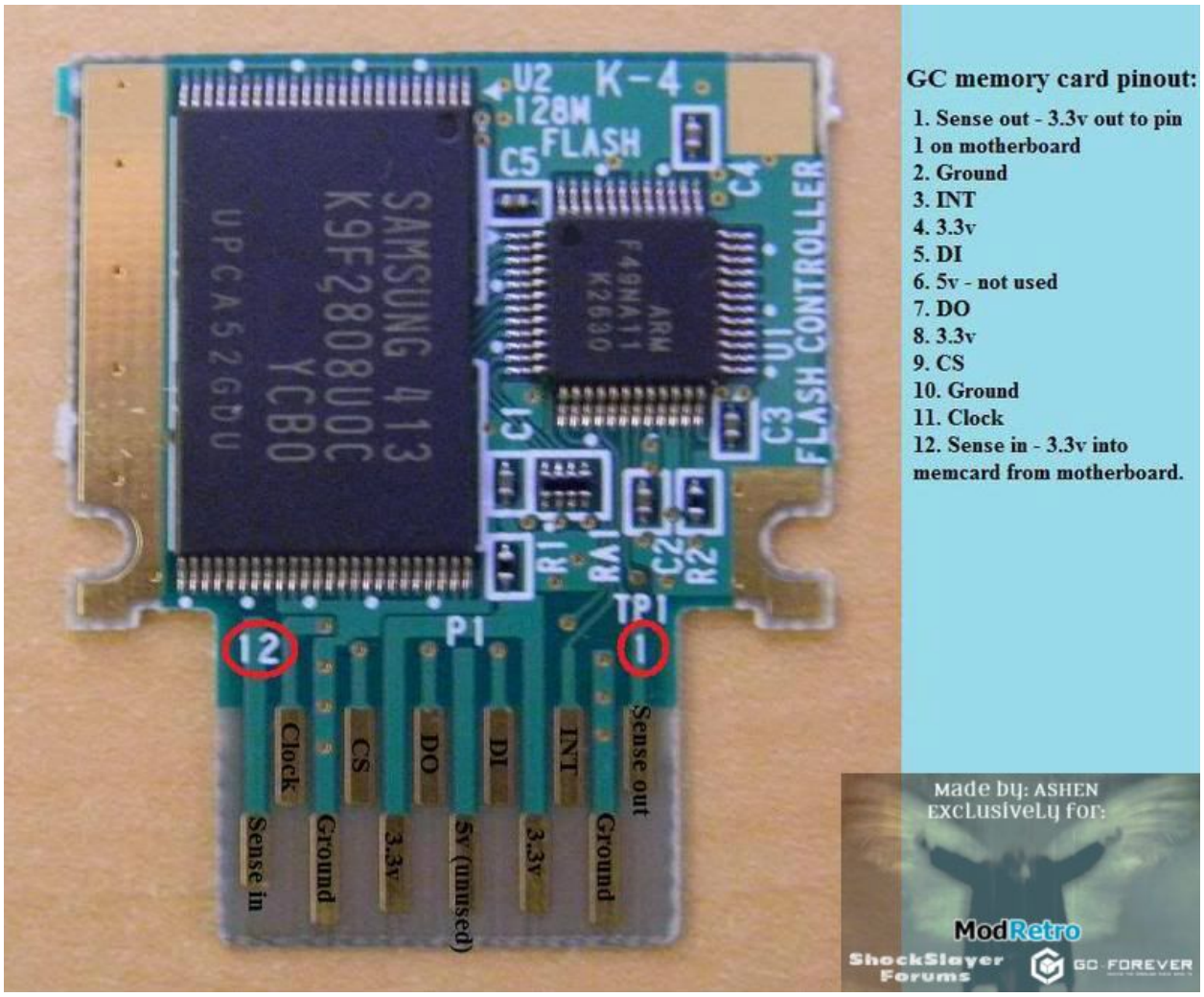

Figure 5.3.2.1: Gamecube memory card pinout

The console acts as the master on this SPI bus, so consequently our device needs to act as the slave. The only role we need to fill on the SPI is simply to read the data output by the master. The microcontroller needs to be initialized as a slave that expects the (0,0) mode of SPI operation as described above. Then it simply needs to read the data coming across the buffer, which should contain all the relevant game data necessary to make statistics about the game. This data reading will be driven by interrupts; the microcontroller should be reading from the buffer when the transmission flag is high.

5.3.2 Creating statistics from game data

Once the proper data is being acquired by the microcontroller from the console, we enter an algorithm for the acquisition of statistics about the game at hand. Data is sent about every frame

of the game (since Melee is a 60 FPS game, this means we are acquiring 60 data packets per second). Each time a packet of data is received by the microcontroller, it will evaluate it and update a running total of each statistic in question and send the data to the Wifi chip so it may be posted on the website. This code must be robust enough to track all the desired statistics accurately, while being fast enough to handle the large data inflow and outflow.
In order to easily create statistics, the data will need to be well-organized. As the data is taken in from the SPI, it will be stored in structures, one for each character. There will be a separate structure for each character that stores each of the statistics about it over time, and there will be individual commands that check and update the running value of each statistic inside those structures. This structure will then be output to the Wifi chip as a list of hex values in a certain order, similarly to the way the data is sent out of the memory card slot, but now we are sending a stream of statistics rather than a stream of game data.
The entire validity of this project hinges upon the ability to record meaningful statistics. There will be value in storing the simplest bits of information, such as the exact details of how many lives the winning character had at the end of the match, the average damage each character managed to take before being knocked out of the arena and losing a life, the hit percentage of each player with a certain move, or the percentage success rate of the player with certain technical inputs such as “wavedashing” and “l-cancelling”. The true value of acquiring live statistics will be in more complex statistics such as average damage output for a “combo”, the percentage of times a player managed to “edgeguard” their opponent after knocking them offstage, or the average number of openings/hits a player required in order to secure a kill.

5.4 Wi-Fi Transmission

5.4.1 Arduino IDE

An Arduino IDE is used to interface with the Sparkfun ESP8266 Thing Dev. On this IDE, it is available to download the board through the board manager which makes it compatible with the Arduino system. Then, the appropriate PubSubClient manager is available which received minor edits to be able to perform with the specific scripts that need to be run.

5.4.2 Sparkfun ESP8266 Thing Dev

Using the Sparkfun ESP8266 Thing Dev, Wi-Fi transmission will be set up to the send data to the server on the pi (explained in the next section). The transmitter will take the bus output from the SPI interface outputted from the microcontroller after retrieving it from the GameCube. This information from the microcontroller is the selected statistics picked out by the microcontroller, in select packets.

Figure 1. Sparkfun ESP8266 Thing Dev

5.4.3 MQTT.fx

Multiple topics on the Mosquitto MQTT server will be created on the pi in order to accommodate the different packets of data that will have to be stored. MQTT.fx is setup to test and observe the what data is published to each individual topic, as well as to monitor the specific topics. This is helpful not only to the real time database that will be receiving the data, but also to the correct transmission of the buffer.

5.4.4 Flowchart for inter-subsystem connection

Subsystem 2 acts as a means for the first and third subsystems to communicate with each other. Therefore, the Sparkfun device will have to receive the output buffer from subsystem 1 and then through the process outlined below create a payload that can be sent to a topic on the MQTT server on the raspberry pi.

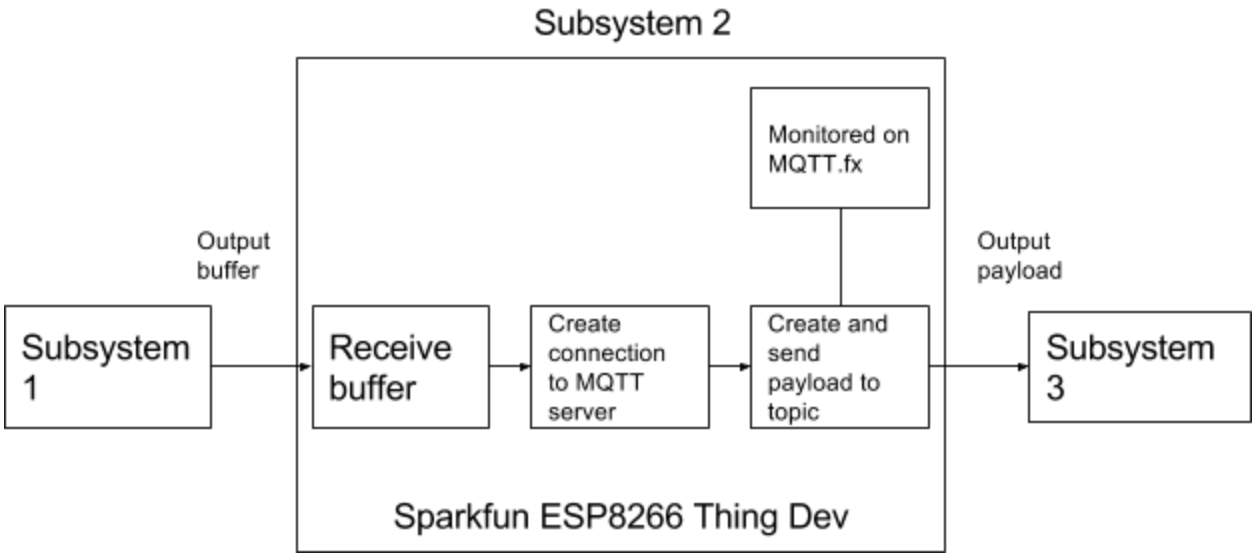

Figure 2. Subsystem 2 Flowchart

5.5 Website

5.5.1 Raspberry Pi and the Server

The Raspberry Pi is set up with a Mosquitto MQTT Server that can receive data from the Thing Dev.
Information that is received will be stored into two overarching databases, which we call Real Time Database, and Archive Database. These databases are full of tables written in MySQL.

5.5.2 Real Time Database

The Real Time Database will have four tables, one for character playing in a match. These tables will be updated every 10 seconds (or as fast as the Thing Dev can send information) with columns on relevant statistics that we want to see live, such as number of hits landed or successful combos.

5.5.3 Archive Database

The Archive Database will contain data that stored after a match is over: matches, what day the match took place, and who competed against who. Tables from the Real Time Database will be copied into the Archive Database. These tables can then be accessed on a website.

5.5.4 Website Application using Flask

Flask is a micro webdevelopment framework for Python. We will build a website that will have all the data accessible on it.

The website will need to display the information in a readable as well as visual format. In order to be user friendly, the user should be able to select what information will be displayed, filtering by things such as time frame, player, character, type of statistic, etc.

5.4 Interfaces

To the extent that the interfaces between subsystems need further explanation, do so. (Parts of this may be covered in the subsystem descriptions.)

6 System Integration Testing

6.1 Describe how the integrated set of subsystems was tested.

6.2 Show how the testing demonstrates that the overall system meets the design requirements

7 Users Manual/Installation manual

(The number of pages depends on how complicated it is to install and use. This section will be different for products that are more “consumer oriented versus business oriented.)

7.1 How to install your product

7.2 How to setup your product

7.3 How the user can tell if the product is working

7.4 How the user can troubleshoot the product

8 To-Market Design Changes

Due to budget and time limitations, your functional prototype may differ significantly from the product you would take to market. In addition, having designed a working prototype, you are now a lot smarter about how best to solve the problem your project addresses. This section should describe the changes that you would make to your design before you would sell it commercially.

9 Conclusions

10 Appendices

Complete hardware schematics
Complete Software listings

Relevant parts or component data sheets (do NOT include the data sheets for the microcontroller or other huge files but give good links to where they may be found.)