Digital Logic Smart Breadboard

By
Arpan Choudhury
Robert Conklin
Joseph Yang

Final Report for ECE 445, Senior Design, Spring 2020
TA: Yichi Zhang

May 8th, 2020
Group No. 28

Abstract

We considered the tedious process of basic circuitry prototyping on breadboards and developed a device to assist in the debugging of breadboarded circuits. The Digital Logic Smart Breadboard is capable of reading and writing digital logic values to each individual row on the breadboard. The state of the board will be configured by, and communicated back to, a host PC, which the user can use to adjust inputs and observe outputs of their logic circuit. Our project is based on the Educational Smart Breadboard [1] from the Spring 2018 semester, which created a standalone device that reads voltages from sections of the breadboard and displays them on an integrated touchscreen. The key improvement we made was instead of using a multiplexing structure in our design, we opted to use IO expanders to interface with the breadboard rows. With this implementation, instead of only being able to read subsections of the breadboard, we are able to read and write to individual rows on the breadboard simultaneously. By the end of the 8-week timeframe, we were able to develop a comprehensive block diagram for our circuitry, sketch out a 3-D graphical model for our physical design, complete a full device schematic, and implement an intuitive graphical user interface for our users.

Contents

1. Second Project Motivation

1.1 Updated Problem Statement

When it comes to engineering prototyping, the breadboard is the gold standard for early hardware testing. However, as the system being prototyped gets larger in size, for example a 4-bit calculator as implemented in ECE385 for Lab 3, debugging can become extremely challenging. This is due to the overlapping clusters of wires, along with limited insight into what is going on in each part of the circuit. As logic circuits get more complex, the number of important test points greatly increases, and circuit probing becomes a more time consuming and involved process, which can be extremely frustrating. This experience could potentially drive away people interested in the field of electrical engineering.

1.2 Updated Solution

The goal of this project is to make debugging easier on a breadboard. We intend to do so in two main ways through a smart breadboard. First, we will be able to read digital logic values in every row on the breadboard using IO expanders communicating with a microcontroller via I^2C. Second, the breadboard will be able to write logic values to each of the rows on the breadboard using the same IO expanders. The state of the breadboard will be configured by, and communicated back to, the host PC. The user will then be able to adjust inputs to their logic circuit, and observe the output of their circuit on the PC, like shown in Figure 1.

The device will be configured over USB 2.0 by a host PC using a software library that allows users to configure and interact with the device. On top of the software library, the user is able to read and write logical values to and from the individual rows of the breadboard via command line interface or a graphical user interface. Within the command line interface, users can also automate testing and verification of their design by writing a sequence of inputs to the breadboard, reading a sequence of outputs from the breadboard, and comparing these outputs to a set of known desired values for the design.

This design solution differs from the original project in three key ways. The first is that our device focuses on debugging digital logic circuits, instead of mixed signal circuits, as with the original project [1]. We chose to focus on digital circuits, because these types of circuits tend to be more difficult to debug using traditional debugging methods for entry-level engineers and hobbyists. Secondly, our design uses IO expanders to read and write digital logic values to each row of the breadboard, instead of a hierarchical tree muxing structure that connected the rows of the breadboard to the ADC pins of a microcontroller [1]. This difference means our device is unable to read analog values from the breadboard, but is able to have much faster board read speeds as a result. Finally, our design communicates with a host PC over USB, and is configured using either a command line interface or a graphical user interface, whereas the original design implemented a touch screen interface to make the device standalone [1]. While there are benefits to designing a standalone device, this comes at a cost of making automation of testing measurably harder for the user.

1.3 Updated High Level Requirements

- The device can read and write logic values at 3.3V or 5V to each row of the breadboard.
- The device is capable of communicating with a host PC over USB 2.0.
- The user can configure the device and receive data from the breadboard through a command line interface or a circuit debugging graphical user interface.

1.4 Updated Visual Aid

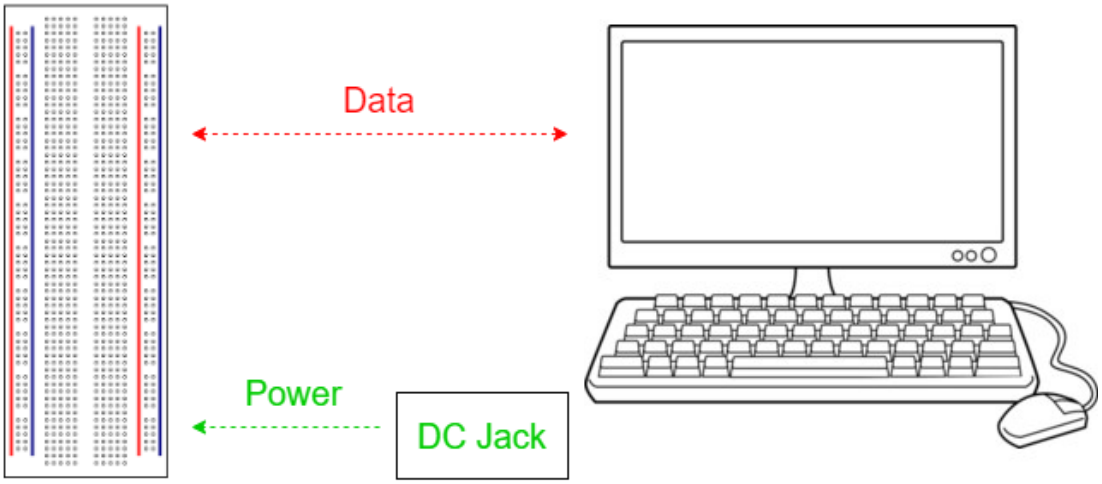

Figure 1. Visual Representation of the Device [2,3]

1.5 Updated Block Diagram

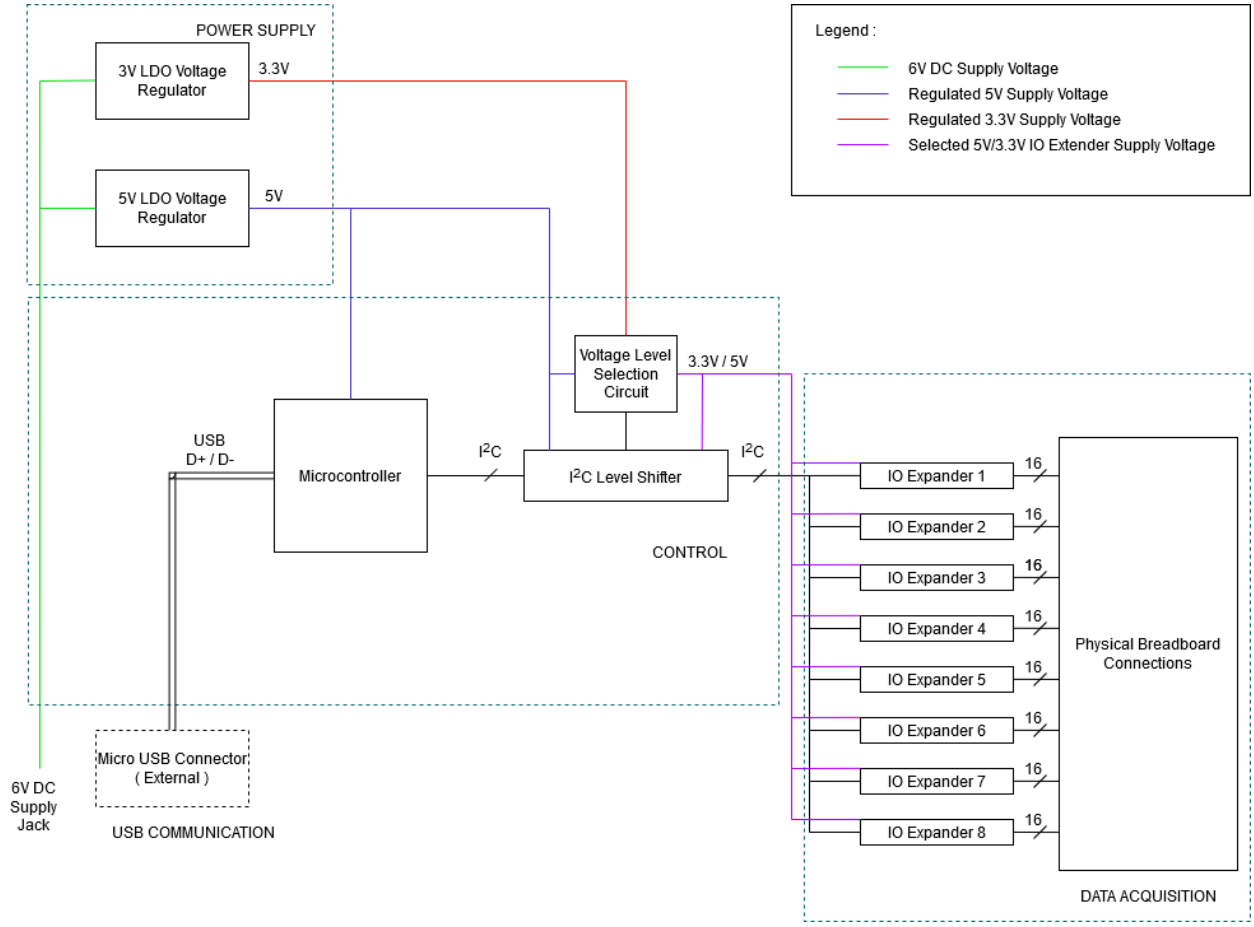

Figure 2. Block Diagram

Our device can be split into four distinct blocks, shown in Figure 2, each with specific functions in facilitating the reading and writing of signals on a breadboard. First, the control block consists of two components: a microcontroller and I^2C Level Shifter. The microcontroller controls the level shifter which then manages the signals from the IO expanders in the Data Acquisition block via I^2C. The control block also features a voltage level selection circuit to distribute the needed voltage values to dedicated IO expanders. Second, the power supply block provides necessary power to corresponding components. Third, the Data Acquisition block consists of several IO expanders that connect externally to our breadboard with the purpose of reading/writing voltage values. Finally, the USB Communication block externally connects to a host PC, through which users interact with our device.

2. Second Project Implementation

2.1 Implementation Details

In this section, we will discuss the implementation of our project one subsection at a time. Chapter 2.1.1 explores the complete physical design of the device, addressing feedback received in the design review. Chapters 2.1.2-2.1.5 detail the four subsections of our device: Control, Power, Data Acquisition, and USB Communication. Chapter 2.1.6 focuses on the Graphical User Interface that the user can use on the host PC to configure the device. We will provide figures such as a 3D model and a GUI design to help illustrate the layout and operation of our device.

2.1.1 Physical Design

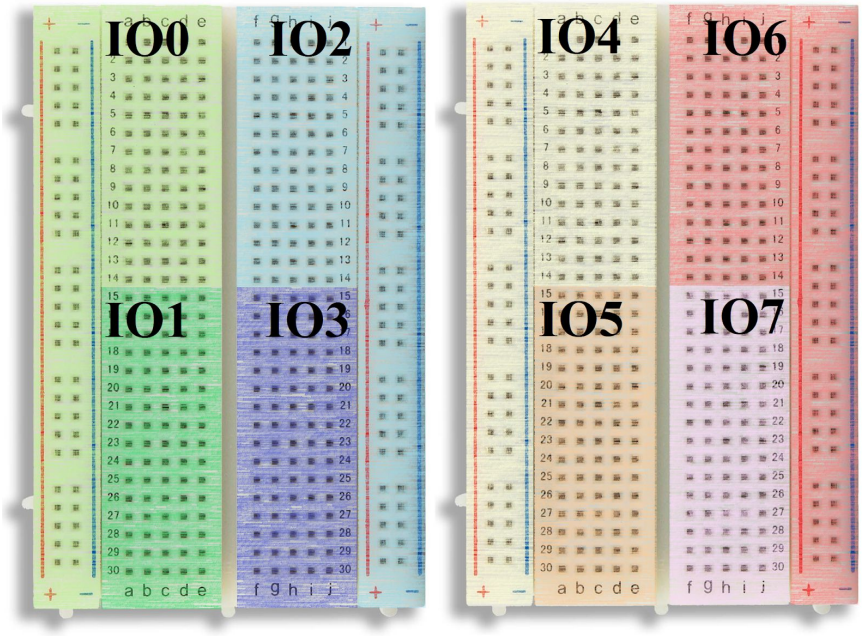

Figure 3. Coverage of IO Expanders on Breadboard

Figure 3 is a physical representation of our device. Each IO expander covers a section equal to a quarter of the each breadboard's rows, totaling 8 IO expanders for two breadboards. There will be two IO expanders on each breadboard which toggle connections to the supply voltage and ground for the power rails, instead of reading or writing logic values directly to the breadboard rows. These IO expanders cannot read logic values from the power and ground rails, as they instead function to make or break connections between the power and ground rails and the supply voltage and ground for the device. This allows those rails to be used as either power rails, or as bus lines if the power supply connections are not enabled. This design choice was made to allow greater current draw from the supply rails at the cost of being unable to read values directly from any signals the users wired to the supply lines.

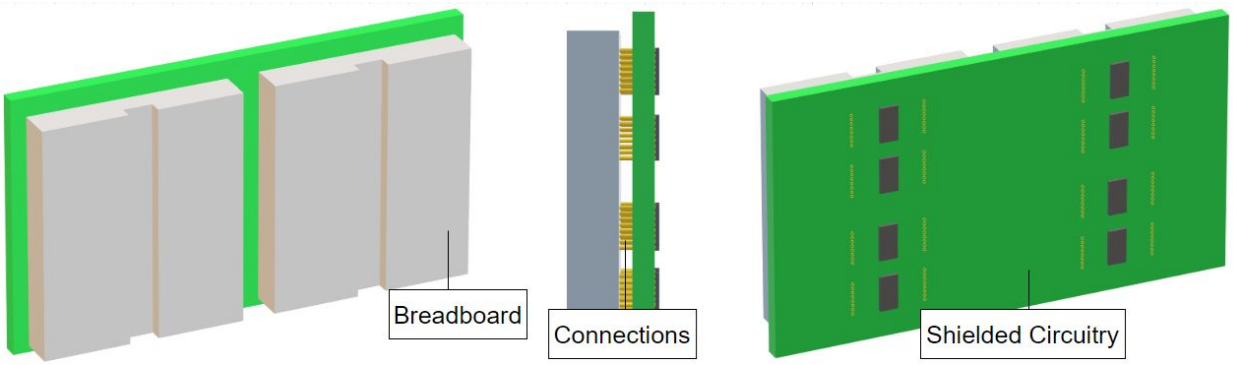

Figure 4. Physical Layout of Breadboard and Circuitry

The added circuitry lies on a PCB attached to the bottom of the breadboards, connecting through the rails on each row of the breadboard. This is demonstrated in Figure 4, where the shielded PCB is shown in green and the breadboards in white. We created this 3D model based on feedback in the design review to make our physical organization clearer. The copper wires joining the two are the connections from the IO expanders to the rails on the breadboards. For the PCB itself, we decided to position the IO expanders according to their coverage on the breadboard to be consistent. Spreading out the IO expanders also decreases the potential heat dissipation in a given area. The microcontroller and other components would be near the middle of the PCB, away from the IO expanders. In terms of the breadboard area, this is under the space between the rails of the two sections of the breadboard. There are few connections between the IO expanders and rails here, which makes it a perfect place for the microcontroller and remaining components.

2.1.2 Control

The control block handles the processing of instructions from the host PC and configuring the breadboard's state based on user input. It consists of an ATmega32U4 microcontroller and a voltage selection circuit to toggle between the two logical high voltage values. The control block also handles the I^2C communication with the IO expanders, processes the acquired data, and packages it for the host PC to receive over USB and display to the user. The key functions that will be handled by this block are the processing of the configuration data sent from the host PC, and the setting of the configuration registers of the IO expanders according to this data. This includes setting the mode (input/output) of each GPIO pin on the IO expander, and the logic value to be written to the output mode pins. Additionally, any other configuration data or instructions will be processed by the control block and sent to the IO expanders, such as configuration of interrupts for future development of the device, as discussed later in Chapter 3.4.2.

2.1.3 Power System

The power system will consist of two power circuits in order to provide the board with two voltage levels: one at 5V and one at 3.3V. Each of these power circuits will consist of a voltage regulator and its needed components (capacitors and resistors). The components must have a large current tolerance as there is a relatively large amount of current going through the circuit at peak draw. Because the output of the 5V regulator is close to the input, the component we use must be a low

dropout regulator. We chose the XRP6272IDBTR-F voltage regulator for this purpose. We also decided to use the same voltage regulator for the 3.3V Power Circuit as well, because it can provide the desired output with minimal configuration. This voltage regulator can handle up to a 10W draw, while our components draw 8.1W maximum, so the regulator works for our purposes.

2.1.4 Data Acquisition

The data acquisition block consists of 8 IO expanders, each managing 16 rows on the breadboard. This allows the data acquisition block to read and write logic values to all 120 breadboard rows, and toggle connections to the board power supply for 8 power rails (4 positive voltage rails, and 4 ground voltage rails). Each IO expander controls 16 distinct electrical contact points on the breadboard, and is capable of reading and writing digital logic values to each of these rows. A total of eight IO expander pins are dedicated to enabling/disabling direct connections between the positive and negative power supply rails and the positive and ground terminals of the soltage selection circuit, allowing the user to draw larger amounts of current from the supply rails at the operating voltage of the rest of the breadboard. Shown in Figure 5 is a single IO expander schematic as included in our design.

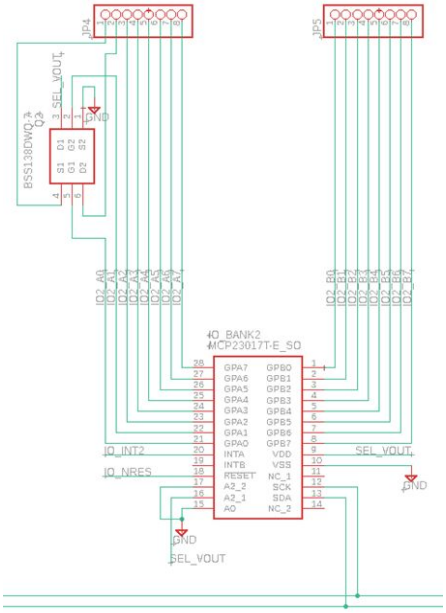

Figure 5. Single IO Expander

2.1.5 USB Communication

The USB block communicates with the PC via a standard USB 2.0. In order to communicate effectively between the host PC and our device, we use the open source library libusb to manage the USB communication with the microcontroller [4]. This library allows users to communicate with our hardware without additional work, like building a dedicated kernel. In addition, with our design, we chose to synchronously transfer I/O data between our device and the PC. Because we are only dealing with a single device with 257-bits of data per transfer, using synchronous transfer is the best method due to its simplistic nature [5]. Lastly, building a Python library and graphical user interface is a necessity for this project. The library will consist of functions to initialize USB communication, configure the power state of the device, read data from the breadboard, and, lasly, write voltage values on the breadboard. Similarly, the GUI is built on top of the Python library so that the users have a convenient accessibility to our device.

2.1.6 Graphical User Interface

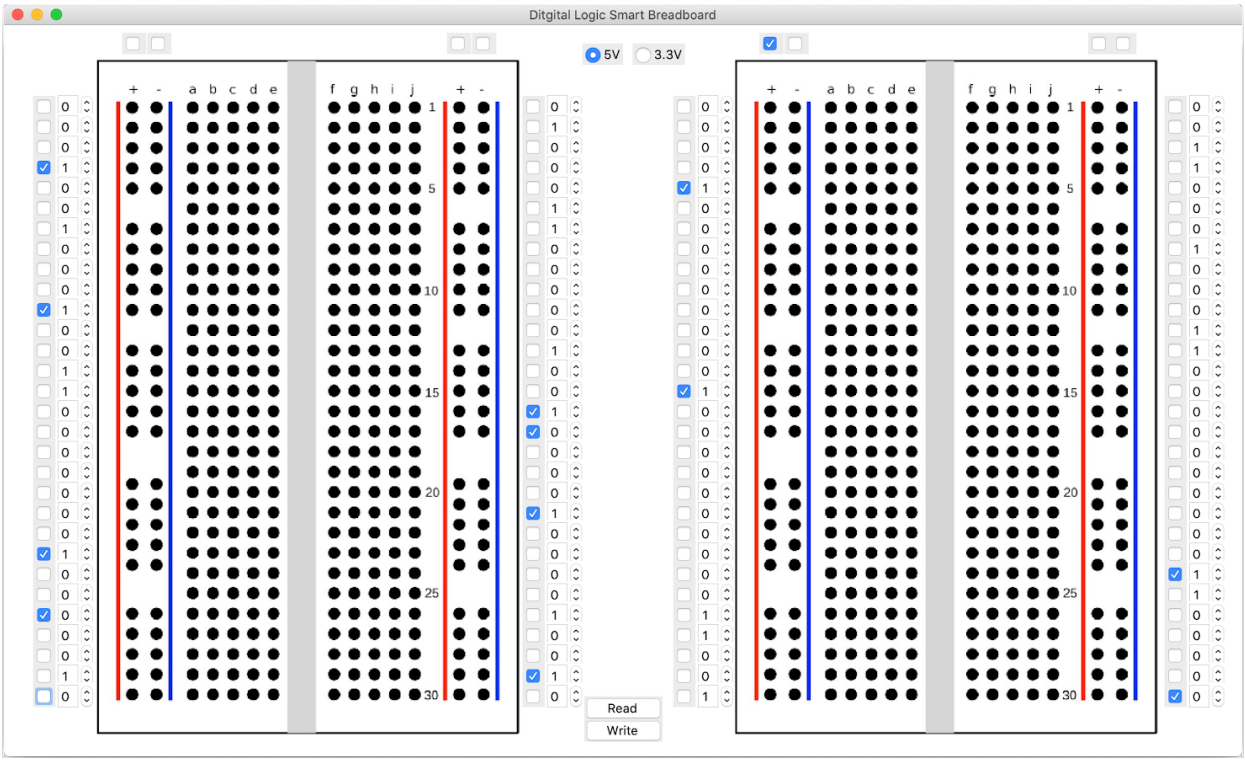

Figure 6. Graphical User Interface

Figure 6 is the user interface built using Tkinter [6]. It has two buttons to specify the users' commands, reading or writing, and two radio buttons to indicate voltage values, 5V or 3.3V. Each row on the breadboard has a checkbox and a spinbox, except for the bus lines which only have checkboxes for writing purposes. Additionally, the breadboards were retrieved online [7] and displayed on the window using Tkinter canvas widget. In order to perform a read, the users will press the 'Read' button. Then, the breadboard voltage values will be loaded onto their corresponding spin boxes on the GUI. If the users want to perform a writing command, they will specify a voltage value through the radio button on top of the GUI, select the checkboxes and spinbox values of the breadboard rows, and press the 'Write' button. The data that is specified on the GUI will then configure our IO expanders and display the results on the physical breadboard. In the end, we wish to provide an intuitive GUI, so that the users can conveniently operate our device.

2.2 Implementation Analysis

In this section, we will provide graphs and calculations to support and analyze our implementations. For the software analysis, we will present the operational FSM and the software flowchart to discuss the distinctive states which the device will operate in. On the other hand, for hardware analysis, we will provide bandwidth and timing calculations to show the hardware propagation delay and discuss the speed of the board..

2.2.1 FSM Flowchart

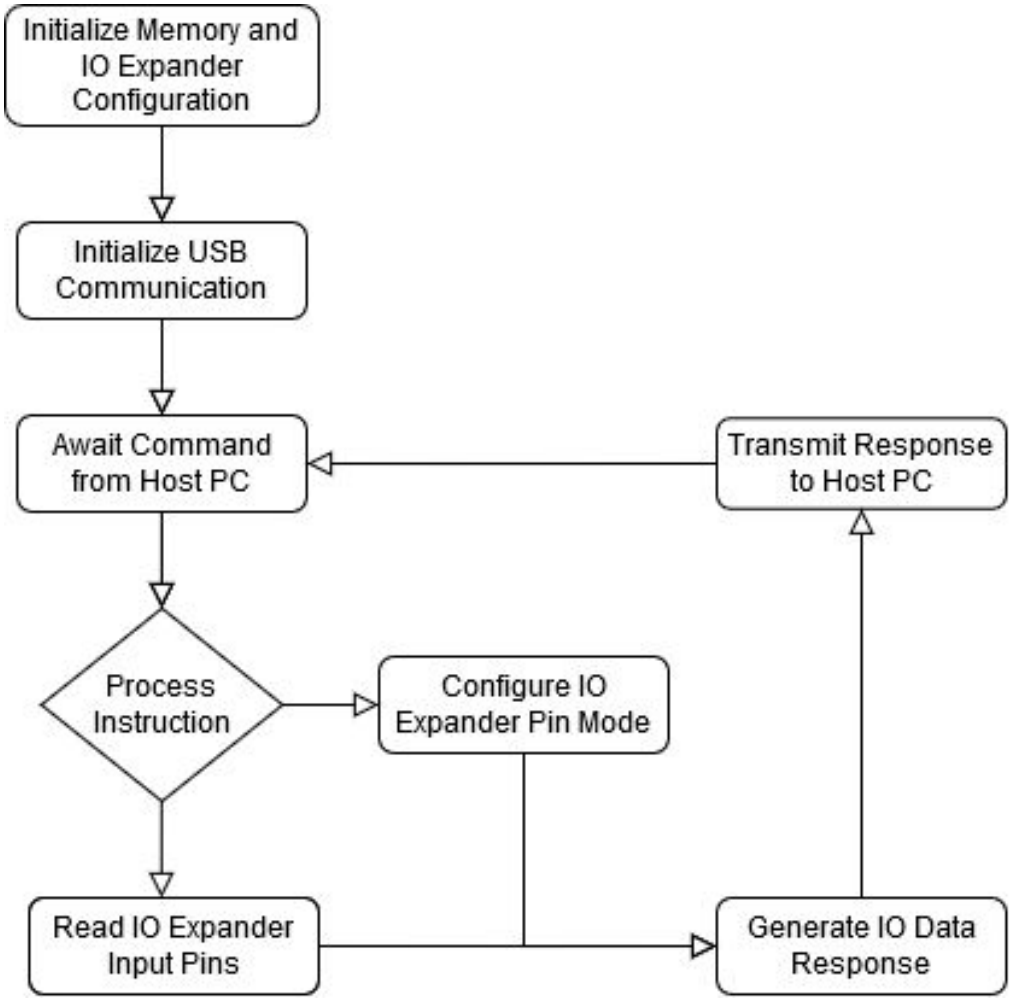

Figure 7. Operation FSM

The model of the flow through which the device goes through is visualized above in Figure 7, the Operation FSM. Upon startup, the breadboard goes through an initialization process before waiting in a state to receive instructions from the host PC. The device performs a different operation based on the instruction it receives, then transmits resulting data back to the host PC.

2.2.2 Software Flowchart

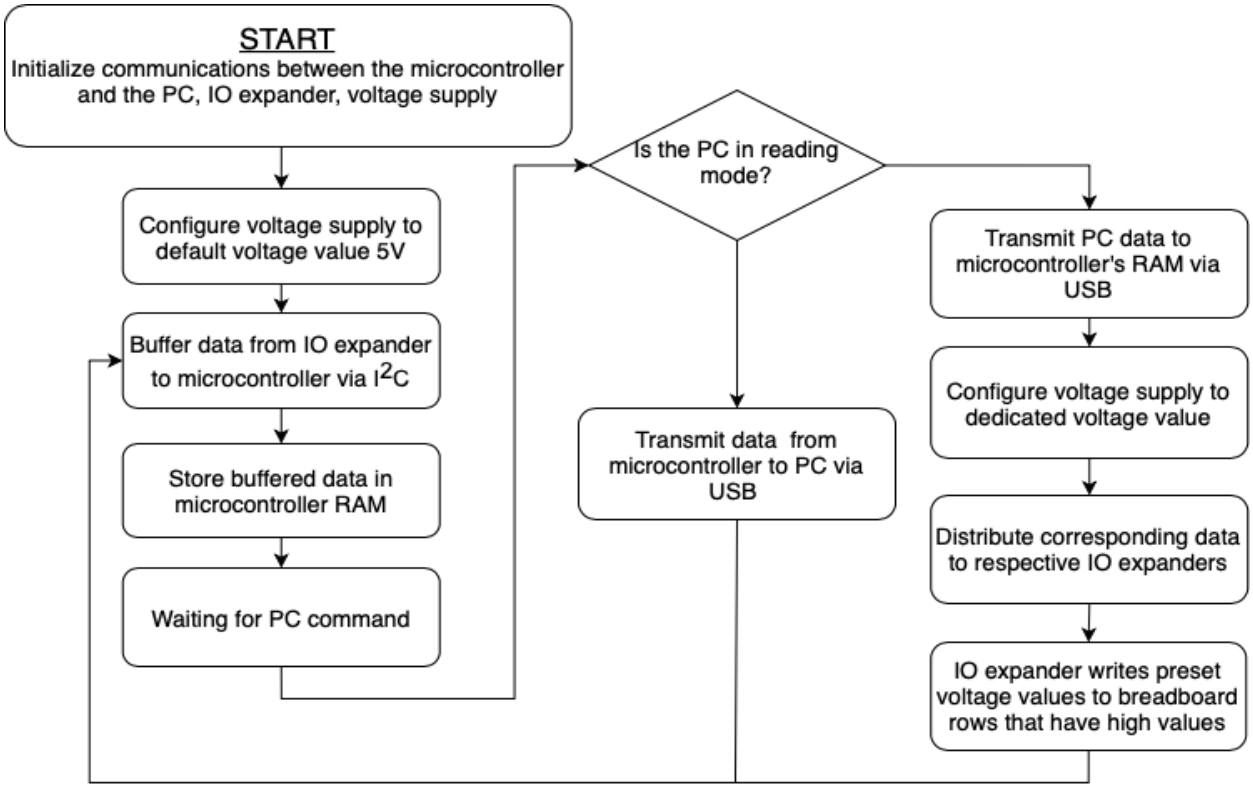

Figure 8. Software Flowchart

In Figure 8, we are able to show the process of how software operates in our device. When the device gets turned on, it first initializes all communications between the PC, microcontroller, IO expander, and voltage supply. It then enters a wait state for the PC to send either a reading or writing command. Depending on the command, the device will communicate with the IO expander and voltage supply to perform different tasks. After finishing its task, the device will return to its wait state, waiting for the next PC command.

2.2.3 Bandwidth Calculation

An important aspect of our design is the rate at which data can be acquired from the data acquisition block. Due to hardware limitations, the fastest clocking speed of the I^2C bus is 1 MHz, or a clock period of 1 us. As shown in Figure 9, the example I^2C communication framework, a total of 47 SCL clock cycles are required to receive 16 bits from a single IO expander on our device. As there are 8 IO expanders total on our device, this brings us to 368 clock cycles to complete a total board read, or a 368 µs period. Since each board read consists of 128 bits of data, the bandwidth of the data acquisition block is 348 kb/s, which is far lower than the 12 Mbit/s bandwidth of the USB 2.0 full-speed standard our microcontroller uses to transmit data to the host PC.

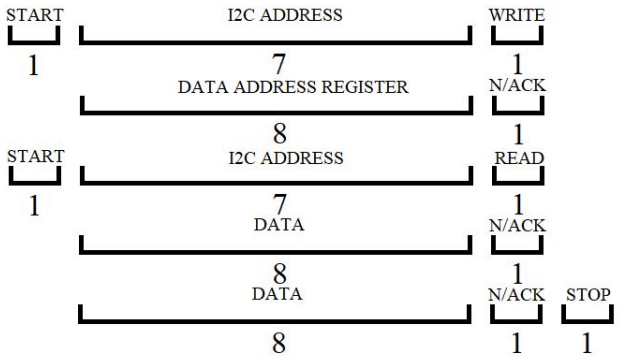

Figure 9. I²C Data Read Communication Framework

2.2.4 Hardware Timing Calculations

Finally, the responsiveness of our device is also important for ensuring it is capable of quickly and efficiently testing circuits according to the user's configuration of the device. This means ensuring the device is capable of quickly reconfiguring the state of the breadboard to the user's desired configuration. There are two major cases where response time is important.

The first case is where the voltage level is changed, and the second is for constant voltage reads. For voltage change writes, the voltage selection circuit needs to select the new operating voltage, and allow for the power mux's output to reach the desired voltage level, which takes a maximum of 60 µs. After this, the IO expanders need to go through a reset to ensure they will operate correctly at the newly selected voltage, and this takes another 2 µs. Also, because they then have gone through a power-on reset, they need to be reconfigured to operate as the user intends, which takes a minimum of 592 µs to have the configuration registers reset, as setting each set of registers requires 74 clock cycles, or 592 total cycles, or 592 µs. Finally, the actual read operation takes 368 µs to complete, as shown in chapter 2.2.3. This brings the total time for a voltage change board update to 1.022 ms.

In the case of a full board update without a voltage change operation only takes 0.960 ms to complete. The most important detail, however, is that a full board state change is the longest part of this process, and takes up over half of the response time in the case that each IO expander has to be overwritten. Should the user only seek to change a small number of inputs (given they are on the same sub-board of our device), the time to update the board will be significantly shorter.

3. Second Project Conclusions

3.1 Implementation Summary

The problem we set out to solve with our design was to make logic circuit debugging easier. Our physical design shows our device is non-intrusive, as none of the structure of the original breadboard is altered. We designed a full device circuit schematic, enabling the user to debug via a GUI or command line interface. This aspect of our design makes debugging much easier, since users will not have to go through a mess of wires just to see where the problem in a circuit lies. Arpan Choudhury was in charge of the physical design and power system; Robert Conklin was in charge of the control and data acquisition; Joseph Yang was in charge of the USB communication and graphical user interface.

3.1.1 Physical Organization

As shown in Figure 4, the PCB containing our device's circuitry would attach to the breadboard through the rails on the bottom of the breadboard. We had produced a full device schematic before the design review, and had analyzed it to minimize the risk of a mistake being in our design. All these circuits would be placed on the back of a PCB which will have its front mounted to the bottom of the breadboards, and will be inside a shielded enclosure, therefore minimizing the risk to the user when they use our device.

3.1.2 Graphical User Interface

The GUI has 2 radio buttons to configure power values, 120 spinboxes to display a voltage value of a given breadboard row, 124 checkboxes to mark whether a given row is being written or not, and 2 buttons to specify users' commands. The whole GUI was built using Tkinter in Python [6]. The breadboards were retrieved online [7] and displayed on the window using Tkinter canvas widget. Furthermore, all the buttons and check marks are responsive and configurable.

3.1.3 Hardware Timing & Bandwidth Calculations

The response time of our device is extremely important in its effectiveness at tackling circuit debugging. A full board write, including a voltage level change, would take around 1 ms to complete, while a full board write without a voltage level change would take 960 µs. These are worst case estimates, and most board writes would be far shorter. Additionally, the simplest way to decrease these response times would be to increase the bandwidth between the control and data acquisition blocks, as discussed in chapter 3.4.1 as a future hardware improvement to our design.

3.2 Unknown, Uncertainties, Testing Needed

Due to the inaccessibility to the lab, we are unable to implement any hardware for our project past the design phase. This includes ordering our parts, testing our components, soldering our components onto the PCB, and attaching the PCB to the breadboard. Without a lab, assembling these components and verifying they were assembled properly would be very difficult. To verify that our design's IO expanders read and write 5V and 3.3V to the breadboard, we would need to go through a voltage verification process. This involves using an oscilloscope to debug the I^2C communication between the IO expanders and microcontroller. The verification process would

consist of setting the IO expanders to a predetermined output state, and using the multimeter to ensure the output on the breadboard matches the desired output as configured through the microcontroller. An oscilloscope probed between the data acquisition block and the microcontroller measuring voltage would be integral to verifying that data is being transferred from the breadboard rows to the microcontroller. The microcontroller is also integral in verifying the functions in usblib, but this is a component we cannot configure with our limited resources. Even if orders were not delayed due to the pandemic, the limited time we have is insufficient to properly set up and test the microcontroller.

3.3 Ethics and Safety

There are a number of safety issues that could arise depending on the user's decision in creating a circuit. Careless actions such as using too high a voltage or shorting a line somewhere could damage used chips or burn certain components. The user should be wary of powering a component at too high a current and voltage, as touching such components could cause burns. A regular breadboard is typically rated at 5W [8], so operating components at values above this could result in parts catching on fire or exploding, which endangers people in the surrounding area. As a result, the circuitry we are using operates at a maximum of 5V, with current to each pin at a low value.

The ACM Code of Ethics and Professional Conduct specifies avoiding harm, 'unless there is a compelling ethical reason to do otherwise.' [9] In accordance with these guidelines, we will shield our circuitry from the user such that it is not easily tampered with. The higher wattage elements of our device will not be accessible to the user, so they will not be able to come in contact with anything that could cause harm. With a commercial product, we would also warn the user of potential consequences of poor circuitry design.

Our approach to the potential safety issues with our device fall in line with the first IEEE code of ethics: "to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, and to disclose promptly factors that might endanger the public or the environment" [10]. The safety of the user will always be of utmost importance.

3.4 Project Improvements

There are a number of improvements that can be done on our design. While some of these improvements require large scale changes to the current hardware, some improvements can be done with minimal to no hardware changes to the existing design.

3.4.1 Hardware Improvements

The first improvement to our design would be to increase the number of modular breadboards available to users. The current design is unable to support an increase in the number of modular breadboards without large scale hardware architecture changes. This would include an overhaul of the power supply circuits to accommodate the increased power draw, a new microcontroller with a larger number of IO pins and I^2C communication ports to handle the increased number of IO

expanders required to increase the number of IO expanders. Additionally, a good deal of the communication facilitation hardware such as the level shifters would need to be changed, as changing to a more complex microcontroller would likely operate at a lower voltage level than our current one. Finally, the software would also need to be rewritten to function on the new microcontroller.

The second improvement to our design would be to increase the communication speed between the control block and the data acquisition block. The current microcontroller is capable of clocking the data acquisition block hardware at 1 MHz. The microcontroller is the bottleneck in our current hardware, as it is only capable of clocking I^2C communication at speeds 16 times slower than the microcontroller's clock frequency. As the microcontroller is clocked at 16 MHz, which is the maximum clock speed for this IC, this means any further increase in clock speed would require a different microcontroller. However, replacing the microcontroller would only yield a maximum 70% increase in bandwidth, as the data acquisition block can only handle a maximum clocking frequency of 1.7 MHz. Any further increases would require changes to the IO expanders used in the data acquisition block. Bandwidth could be increased by a maximum of 1000% from the original by selecting a new microcontroller capable of communicating over SPI at speeds up to 10 MHz, as the IO expanders in our design have variants that use SPI instead of I^2C, and are capable of handling much higher clocking speeds. However, utilizing the SPI IO expanders would require a much larger number of pins on the microcontroller, and the higher clocking speeds would require more careful design considerations for placing the ICs and routing the copper on the board.

3.4.2 Software Improvements

The third improvement to our design would be to test and develop glitch detection functionality. Our existing design is already capable of asynchronously recognizing input state changes, and capturing them for the user to view the changing state that triggered an interrupt in the IO expander. This is accomplished by using the interrupt-on-change pin functionality of the IO expanders, allowing the microcontroller to detect the IO expander interrupt, and read back the input state of the IO expander at the time of the pin change. This functionality would allow users to detect glitches in their logic, and determine exactly where and when they are occurring, allowing for much easier debugging.

The final improvement to our design would be to improve the functionality of the GUI to support automation of testing and validation by implementing a block-based programming interface specifically for this purpose. The current GUI is designed to support debugging, and is incapable of automating testing and validation of circuits, as this functionality is delegated to the software library written to facilitate communication with the device. A further improvement of our project would be to implement this functionality into our GUI, allowing users to automate testing and validation using a graphical interface instead of writing code to interface with the device.

3.4.3 Design Approach Improvements

Our initial approach to this project strongly took into account the limited time available to complete it. This meant that our initial design had to take into account the design time for the schematic design, board layout, board assembly, and software features. Because of this, our original design was built on a limited scale, and only included a limited number of IO expanders. However, if instead the project timeline spanned a year instead of the 8 weeks we have, a number of aspects of our design could have been more comprehensively developed and improved.

The current hardware design cannot support faster communication between the data acquisition and control blocks, and is also unable to support any increase in the number of modular breadboards it can support. If instead we had a year to work on this project, we would have further improved our design to support a greater number of modular breadboards, along with faster communication speeds between the data acquisition and control blocks. However, this would greatly increase the complexity of our schematic design, board layout design, and also the assembly time of our PCB due to the increased complexity of our hardware architecture, and increased number of ICs to place on the board.

Additionally, the increased complexity of our hardware would also cause the software development to be more complicated as well. Overall, if the schedule had been extended to a full year, the initial planning and early design phases would have been far more crucial to the success of the project. Along with this, comprehensive advance scheduling of the project's progress that still maintained a strong degree of flexibility in case of surprise problems appearing in the process would be absolutely essential to the success of our project.

4. Progress Made on First Project

The first project we worked on was a portable bluetooth music player, aiming to reduce the bulk of listening to music while exercising. Before spring break and transition to our second project, we made considerable progress after the first design review, including the submission for the early bird PCBway order to our TA and the research and confirmation of our software scope for the project.

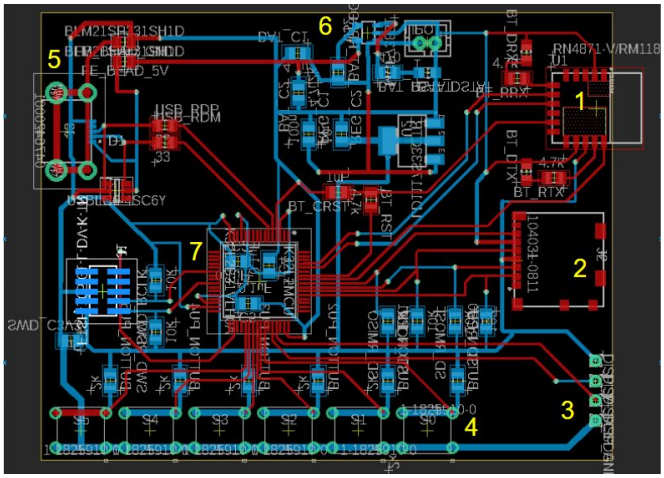

Figure 10. First Project Board Layout

Because a key point of our project was sizing the PCB, a small and compact board layout was integral to our design. Figure 10 is a representation of how small of a footprint we could size our components on. This board layout passed the PCBway audit and was the most we could do before the COVID-19 pandemic shut down PCB ordering. The important subsystems on the board layout are numbered and listed to clarify the physical organization.

Table 1. Software scope for Portable Bluetooth Music Player

Reuse	Develop
SD file system management	OLED display interface
Audio Transmission	Buttons configuration

Additionally, during the first design review, Professor Kumar pointed out the insufficient software details of this device. After doing some extensive research, we were able to define our software scope as the following: SD file system management, OLED display interface, audio transmission, and buttons configuration. The audio transmission describes the process of decoding MP3 files and encoding it to SBC, so that we can transmit data via UART. We know that there are multiple open source libraries that we can reuse to our advantage due to the time pressuring nature of the course. Therefore, Table 1 accurately partitions the sections of software we will reuse and develop. We will be using MBED's library to access the SD card as a file system [11]. And, we will be using libsndfile to decode MP3 and encode SBC [12]. Lastly, we will also be responsible for stitching all the software sections and FSM together to meet real time requirements.

References

[1] Chinemelum Chibuko, Minseong Kim, and Mostafa Elkabir, "Educational Smart Breadboard," [Online] Available: https://courses.engr.illinois.edu/ece445/pace/view-topic.asp?id=25937 [Accessed: March 22, 2020]

[2] Radio Locman, "Electronic Breadboard Templates," *radiolocman.com*, [Online]. Available: https://www.radiolocman.com/shem/schematics.html?di=33992 [Accessed: April 14th, 2020]

[3] Clipart Library, "Transparent Computer Clipart," *clipartkey.com*, [Online]. Available: https://www.clipartkey.com/view/hmRJbh_transparent-computer-clipart-white-computer-clipart/ [Accessed: April 14th, 2020]

[4] libusb, "libusb," *libusb.info*, 2012. [Online]. Available: https://libusb.info/ [Accessed: March 31, 2020]

[5] libusb, "Synchronous device I/O," *libusb.info*, 2012. [Online]. Available: http://libusb.sourceforge.net/api-1.0/libusb_io.html [Accessed: April 14, 2020]

[6] Python, 'tkinter — Python interface to Tcl/Tk,' *Lib/tkinter,* [Online]. Available: https://dlpng.com/png/1450867 [Accessed: May, 4, 2020]

[7] DLPNG, 'Download Free PNG Breadboard,' *DLPNG.com*, [Online]. Available: https://dlpng.com/png/1450867 [Accessed: May, 4, 2020]

[8] George Leger, "Common Breadboard Specifications," *circuitspecialists.com*, March 29, 2014. [Online] Available: https://www.circuitspecialists.com/blog/common-breadboard-specifications/ [Accessed: February 4, 2020]

[9] ACM, 'ACM Code of Ethics and Professional Conduct,' *acm.org*, [Online]. Available: https://www.acm.org/code-of-ethics [Accessed: February 4, 2020]

[10] IEEE, 'IEEE Code of Ethics,' *ieee.org*, [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html [Accessed: February 4, 2020]

[11] MBED, 'SD Card File System,' *mbed.com,* [Online]. Available: https://os.mbed.com/users/mbed_official/code/SDFileSystem/ [Accessed: March 3, 2020]

[12] Erikd, 'libsndfile,' *github.com*, [Online]. Available: https://github.com/erikd/libsndfile [Accessed: March 3, 2020]