Digital Logic Smart Breadboard

ECE 445 Design Document

Group 28
Arpan Choudhury arpanc2
Robert Conklin rmc2
Joseph Yang josephy2

TA: Yichi Zhang

4/18/2020

1. Introduction

1.1 Problem and Solution

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. This increase in complexity makes the difficulty in debugging the system significantly larger, and can lead to a strong sense of frustration. This experience could potentially drive away people interested in the field of electrical engineering.

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.

1.2 Visual Aid

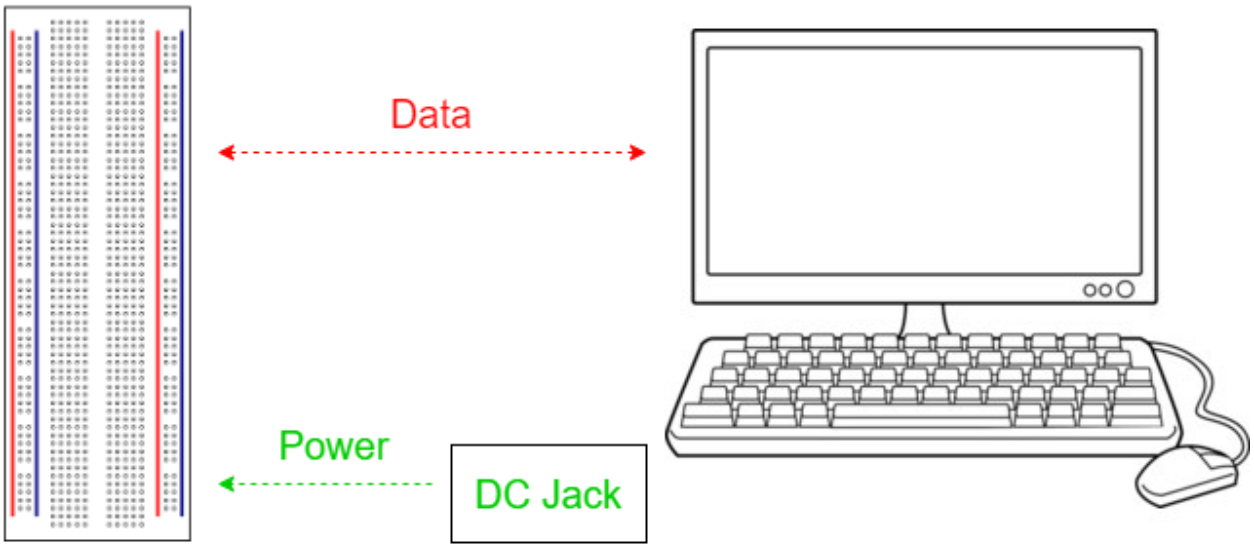

Figure 1. Visual Representation of the Device [1,2]

1.3 High-level requirements list

- 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

2. Design

2.1 Block Diagram

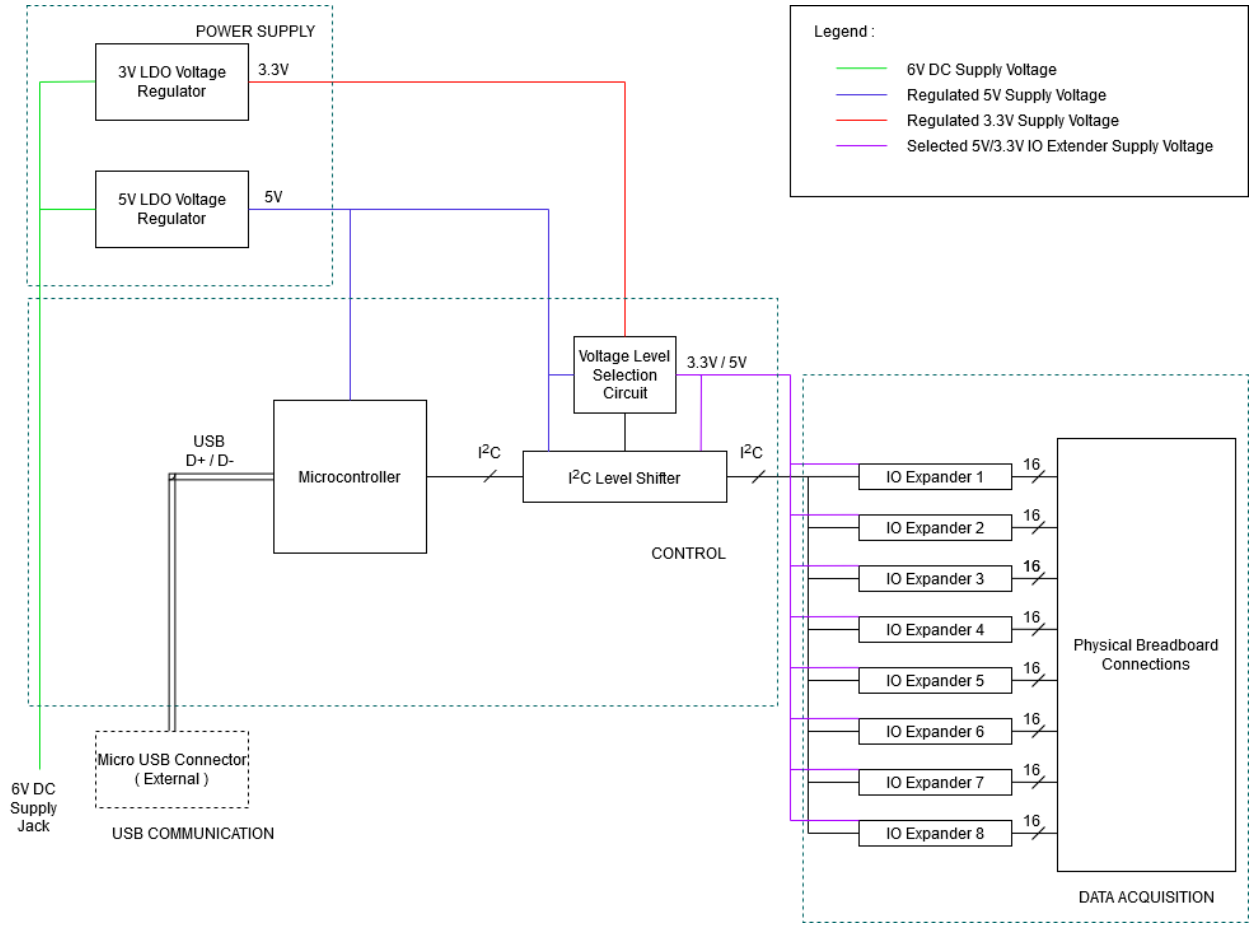

Figure 2. Block Diagram

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.

2.2 Physical Design

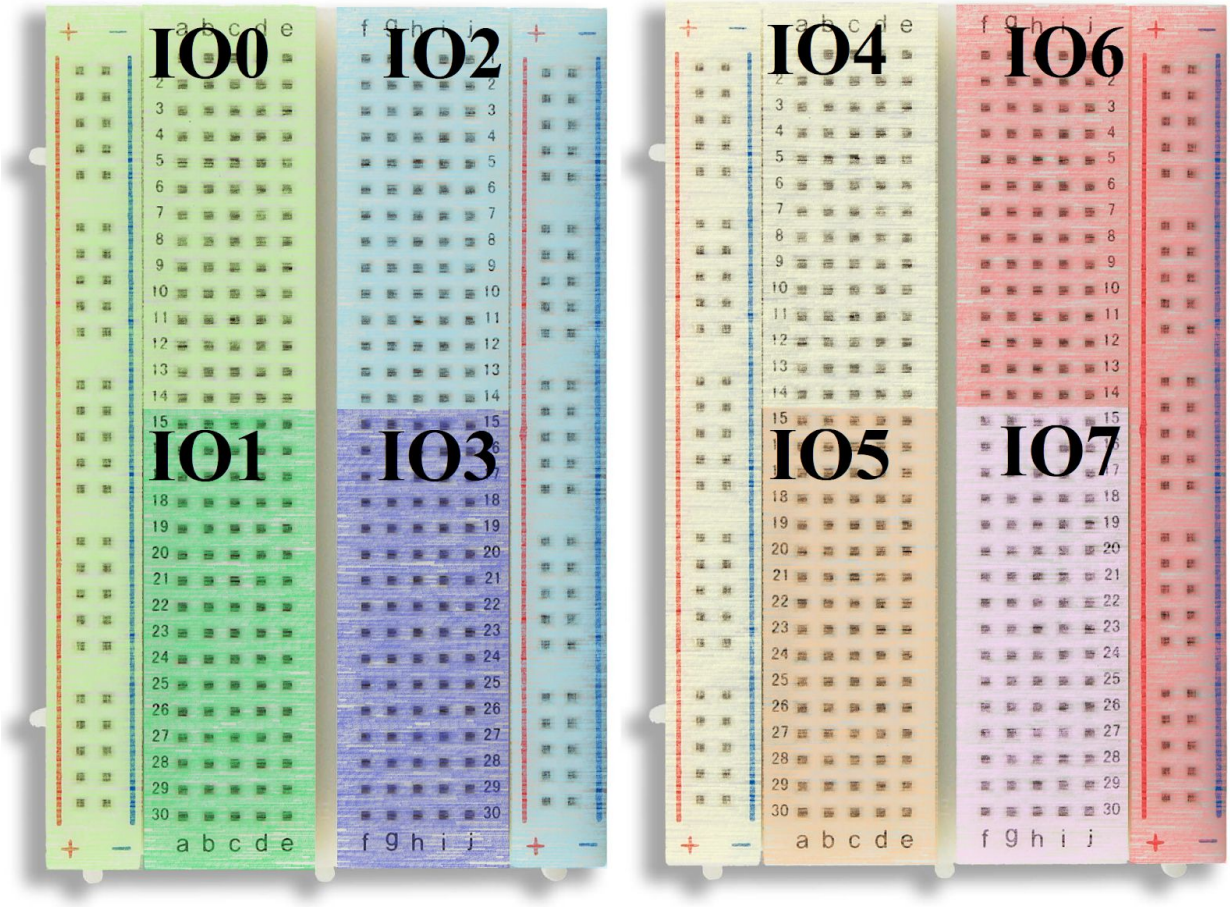

Figure 3. Physical Design

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.

2.3 Block Level Descriptions

The device will be mounted to the backs of two 400-tie modular breadboards [3], connecting the metal rails on the back of the breadboard to a PCB integrating all the blocks in the design. Each of these rails will connect to a line on one of the eight IO expanders in our design. This allows the device to interface with each signal in the logic circuit implemented on the board, as well as read and write logic values in each of these rows. Figure 3 shows a typical 400-tie breadboard, onto the back of which our PCB would be attached to access the signals in each row.

2.3.1 Control

The control block will consist of an ATmega32U4 microcontroller that will communicate with the host PC over USB and the data acquisition block over I^2C. Its EAGLE schematic is shown below in Figure 4. The microcontroller will use a single wire to control a voltage selection circuit, the EAGLE design of which can be seen in Figure 5, to set the operating voltage of the IO expanders. The microcontroller will operate at 5V from the 5V regulated supply from the power system. Due to the required capability of operating at multiple voltages, a level shifter is required between the microcontroller and I^2C peripheral devices. The EAGLE sketch for the level shifter is modeled in Figure 6.

The control system will also process the instructions and configuration information sent from the host PC, appropriately configure the required peripheral devices, acquire requested data, and transmit it back to the host PC. This process will be handled on the device, at the request of the host PC.

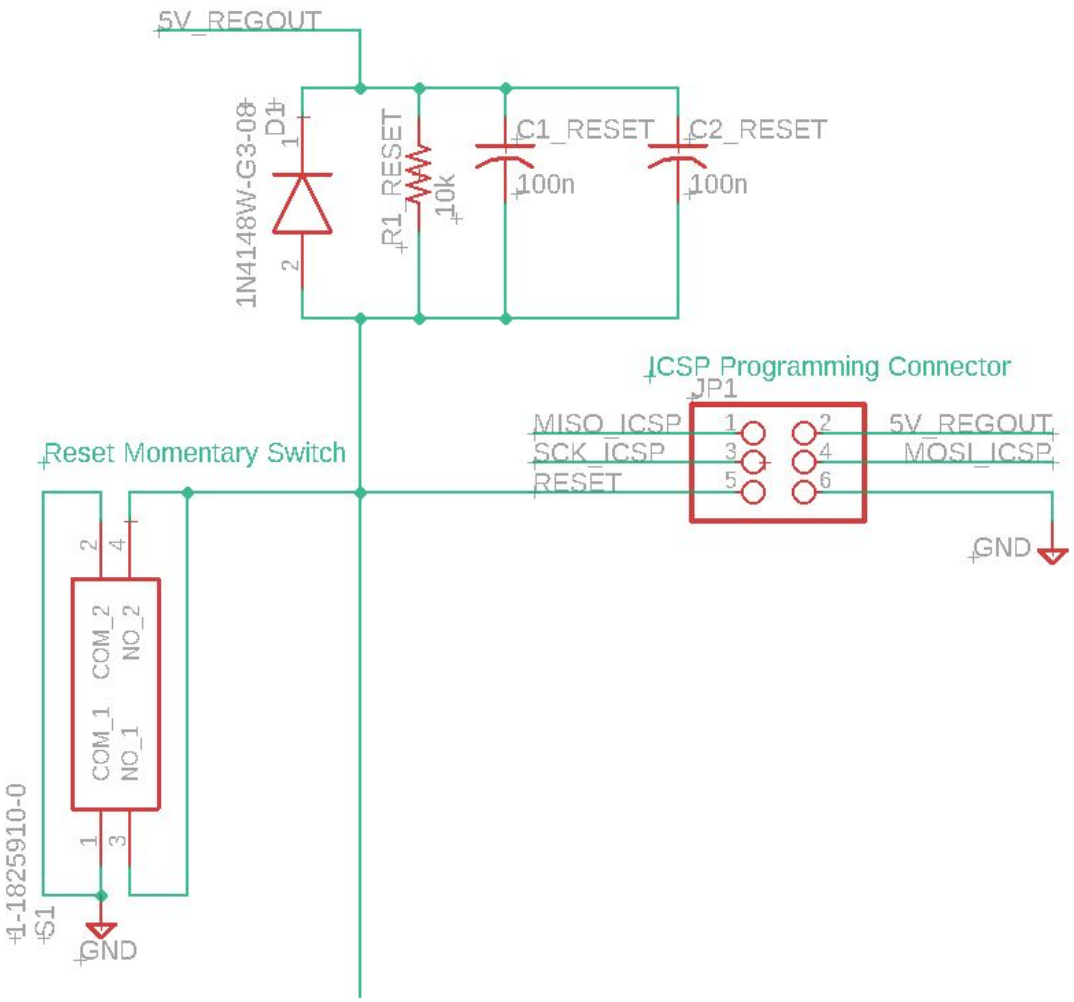

Figure 4. MCU ICSP Programming Circuit

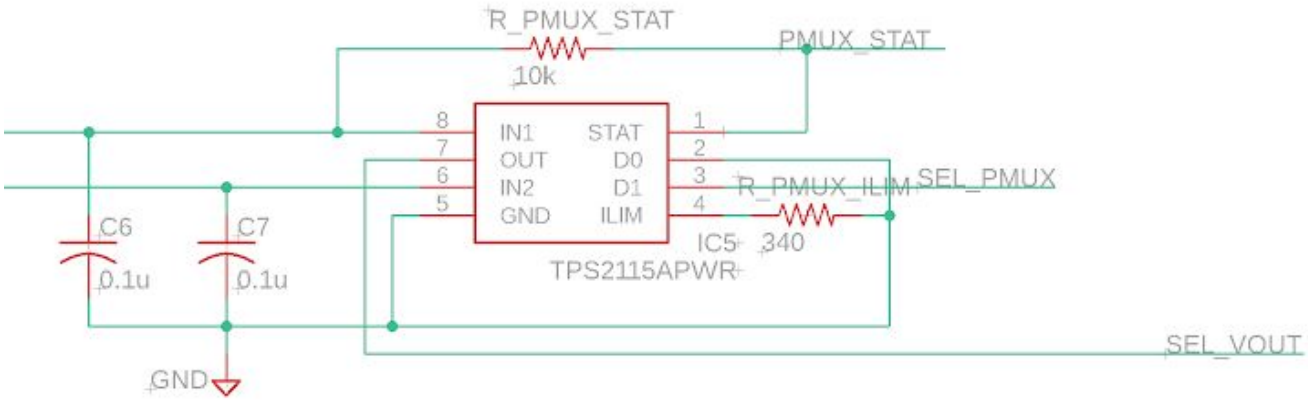

Figure 5. Voltage Selection Circuit

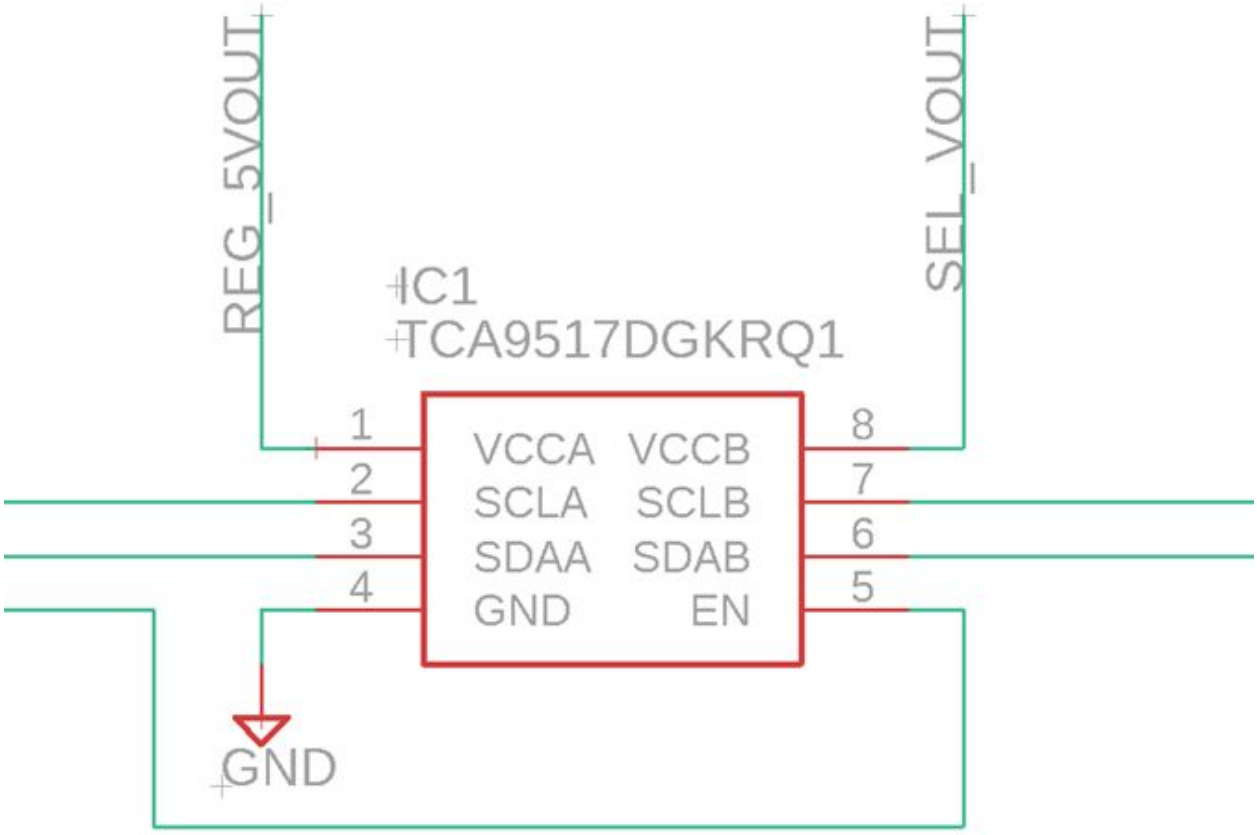

Figure 6. I²C Level Shifter

Requirements	Verifications	Points (15)
Microcontroller 1) Communicate with Data Acquisition block over I²C	Verification Process: 1) Communication Verification a) Obtain a multimeter. b) Configure a predetermined output state for IO Expander pins. c) Place ends of multimeter at each predefined row and ground. d) Verify that each row holds its corresponding logic value, as defined in the microcontroller software.	10

Voltage Selection Circuit 1) Switch between an operating voltage of either 5V or 3.3V	Verification Process: 2) Voltage Verification a) Set voltage selection circuit to for 5V b) Obtain a multimeter c) Place ends of multimeter at output and ground d) Ensure voltage difference is 5V with an allowed variation of 10% e) Repeat for 3.3V	5

Table 1. Requirements and Verification for Control System

2.3.2 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 5V Power Circuit uses the XRP6272IDBTR-F voltage regulator, as seen in Figure 7. We 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. Figure 8 shows we just add a 390kΩ resistor between the VOUT and ADJ pins, as well as a 105kΩ resistor between the ADJ pin and ground to change the output from 5V to 3.3V.

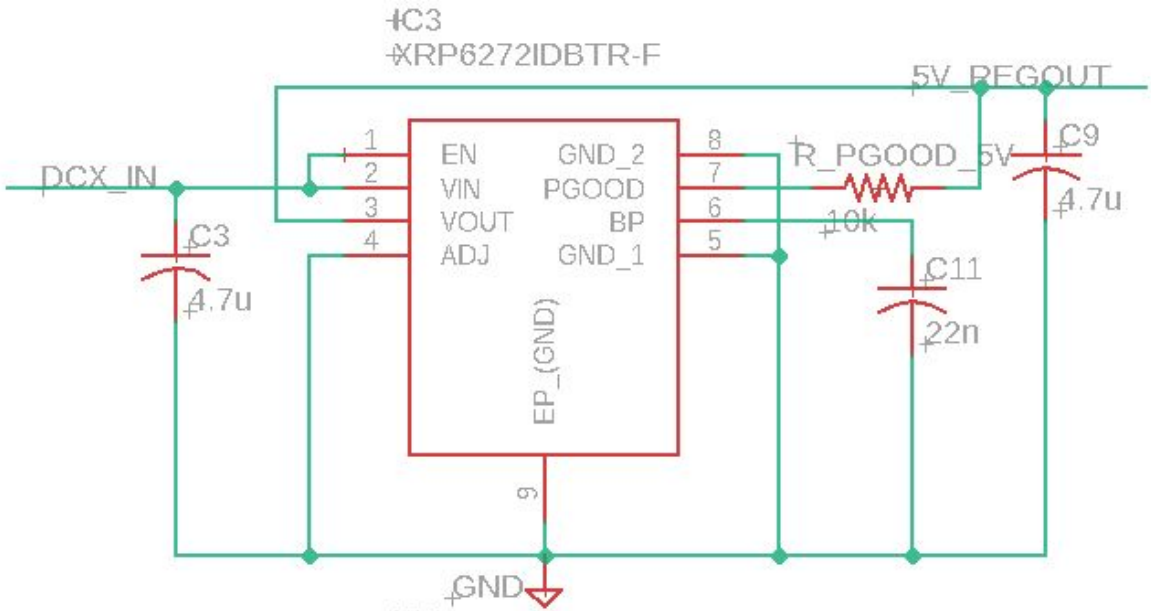

Figure 7. 5V Power Circuit

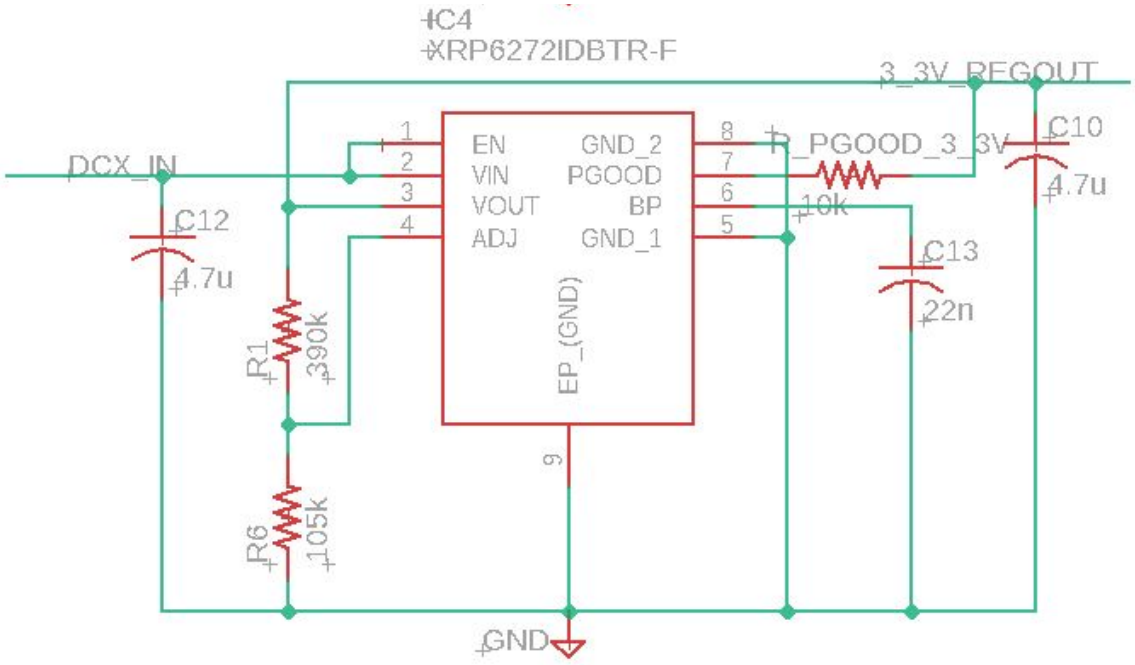

Figure 8. 3.3V Power Circuit

The voltage regulators take in the input voltage and supply a different output voltage. One regulator will supply 5V, and the other will supply 3.3V. Because the output of the first regulator is close to the input, the component we use must be a low dropout regulator. These components must have a large current tolerance as there is a relatively large amount of current going through the circuit at peak draw.

Requirements	Verifications	Points (10)
5V Power Circuit 1) Provide voltage at 5V +/- 10% 2) Supply a maximum of 2 A	Verification Process: 1) Voltage Verification a) Obtain a multimeter b) Place ends of multimeter at output and ground c) Ensure voltage difference is 5V with an allowed variation of 10% 2) Current Verification a) Obtain a multimeter. b) Probe current of power supply with equivalent load resistance of ~2.5 ohms. c) Ensure current is within 1.7-2.0A range.	5

3.3V Power Circuit 1) Provide voltage at 3.3V +/- 10% 2) Supply a maximum of 2 A	Verification Process: 1) Voltage Verification a) Obtain a multimeter. b) Place terminals of the multimeter at output and ground. c) Ensure voltage difference is 3.3V with an allowed variation of 10%. 2) Current Verification a) Obtain an multimeter b) Probe current of power supply with equivalent load resistance of ~1.7 ohms. c) Ensure current is within 1.7-2.0A range.	5

Table 2. Requirements and Verification for Power System

2.3.3 Data Acquisition

The data acquisition block consists of eight IO expander ICs, interfacing with the control block's microcontroller via I^2C. Each IO expander also utilizes a 2-channel NMOS IC to disable and enable supply voltage and ground connections to the corresponding supply rails on the breadboard, as shown in Figure 9. The IO expanders must operate at either 3.3.V and 5V, and communicate with the microcontroller at whichever operating voltage they are using at the time. This is done using a high-speed bidirectional I^2C level shifter, specifically used in I^2C applications. This allows the microcontroller to always operate at 5V, while allowing the peripheral devices to operate at a user-defined voltage. Each IO expander makes 16 connections to individual rows on the breadboard, allowing for read/write operations to occur on any row on the breadboard. In addition, the power and ground rails can be connected to the logical high voltage and ground respectively, or be disconnected from the supply for use as normal bus lines.

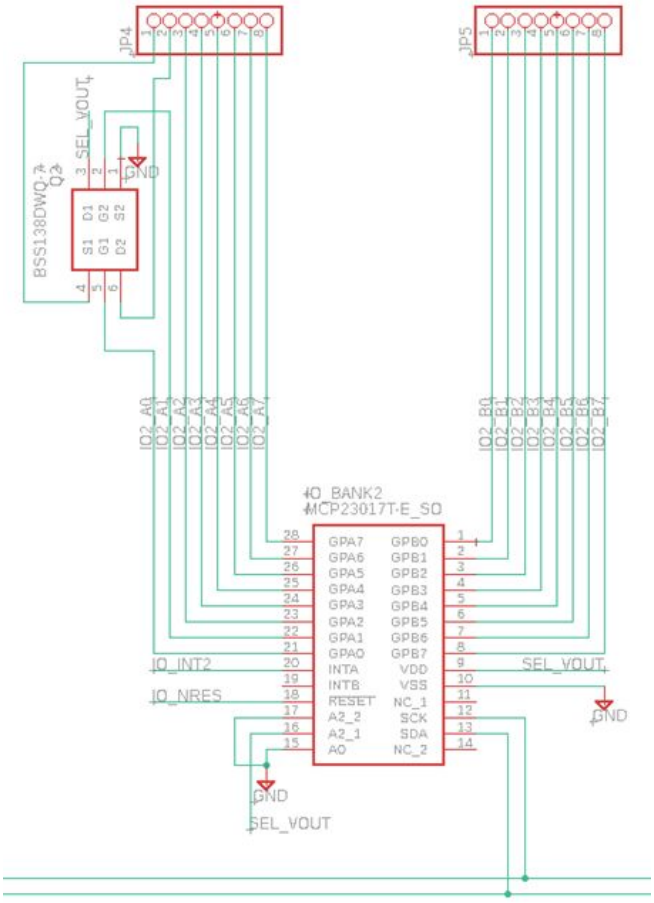

Figure 9. Single IO Expander

Requirements	Verifications	Points(10)
IO Expanders 1) Read and write 5V and 3.3V voltage values to the breadboard	Verification Process: 1) Voltage Verification a) Obtain a multimeter. b) Set IO Expander to write 5V c) Place terminals of the multimeter at row and ground. d) Ensure it is 5V with allowed +/- 10% variation. e) Repeat with 3.3V f) Repeat for each IO Expander.	10

Table 3. Requirements and Verification for Data Acquisition

2.3.4 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 breadboard, we are planning to use the open source library libusb [4] to manage the USB communication with the microcontroller. This library allows users to communicate with the hardware without additional work like building a dedicated kernel. There are multiple benefits for choosing libusb. First, it is an active open source library which our users have the benefit of accessing online forums. Second, it is compatible with multiple platforms, including Windows, IOS, Linux, etc. Third, libusb already supports multiple languages. It very well suits our needs because the educational breadboard could be used across a broad scope of operating systems and coding environments for beginners from diverse backgrounds.

With our design, we chose to synchronously transfer I/O data between our device and the PC. We chose to use synchronous device I/O because we are only dealing with 257-bits of data via USB 2.0. 257-bits could be broken down into three sections: 128-bits comes from the value of the voltage rows; 128-bits of read/write mode of each row on the breadboard; and the last bit is to configure the reading voltages of the IO expander (either 3.3V or 5V). Because we are only dealing with a single device with 257-bits of data, using synchronous transfer is the best method [5].

Lastly, building a Python library on top of the microUSB controller is necessary. The library will consist of functions to initialize USB communication, functions to configure the state of the device, functions to update the data from the breadboard, and lastly functions to read out the data values. Additionally, the library will parse the data into readable format, so that the user can conveniently specify which rows on the breadboard to write on.

The MicroUSB circuit as shown below in Figure 10. was based on the hardware design of the Arduino Leonardo schematic [6], a development board utilizing the same ATmega32U4 microcontroller.

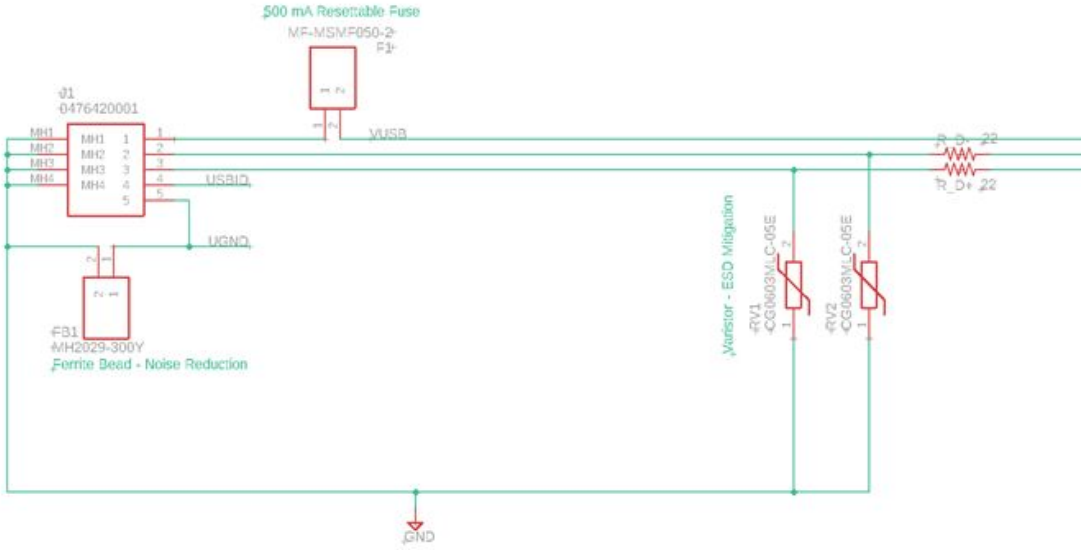

Figure 10. MicroUSB Circuit

Requirements	Verifications	Points (15)
microUSB 1) Synchronously read data from the microcontroller to the PC 2) Synchronously Write data from the PC to microcontroller	Verification Process: 1) Reading Verification a) Store '10101010' in the ATmega32U4 microcontroller's RAM b) Using usblib in Python, pyusb, to perform control_transfer() [7] c) Set all function parameters according to data sheet from ATmega32U4 d) Specifically, check bmRequestType parameter is set read e) After calling, make sure that '10101010' shows on the PC 2) Writing Verification a) Using usblib in Python, pyusb, to perform control_transfer() [7] b) Set all function parameters according to data sheet from ATmega32U4 c) Specifically, check bmRequestType parameter is set write d) Set data parameter to '01010101' e) Ensure we receive '01010101' in ATmega32U4 microcontroller's RAM	15

Table 4. Requirements and Verification for USB Communication

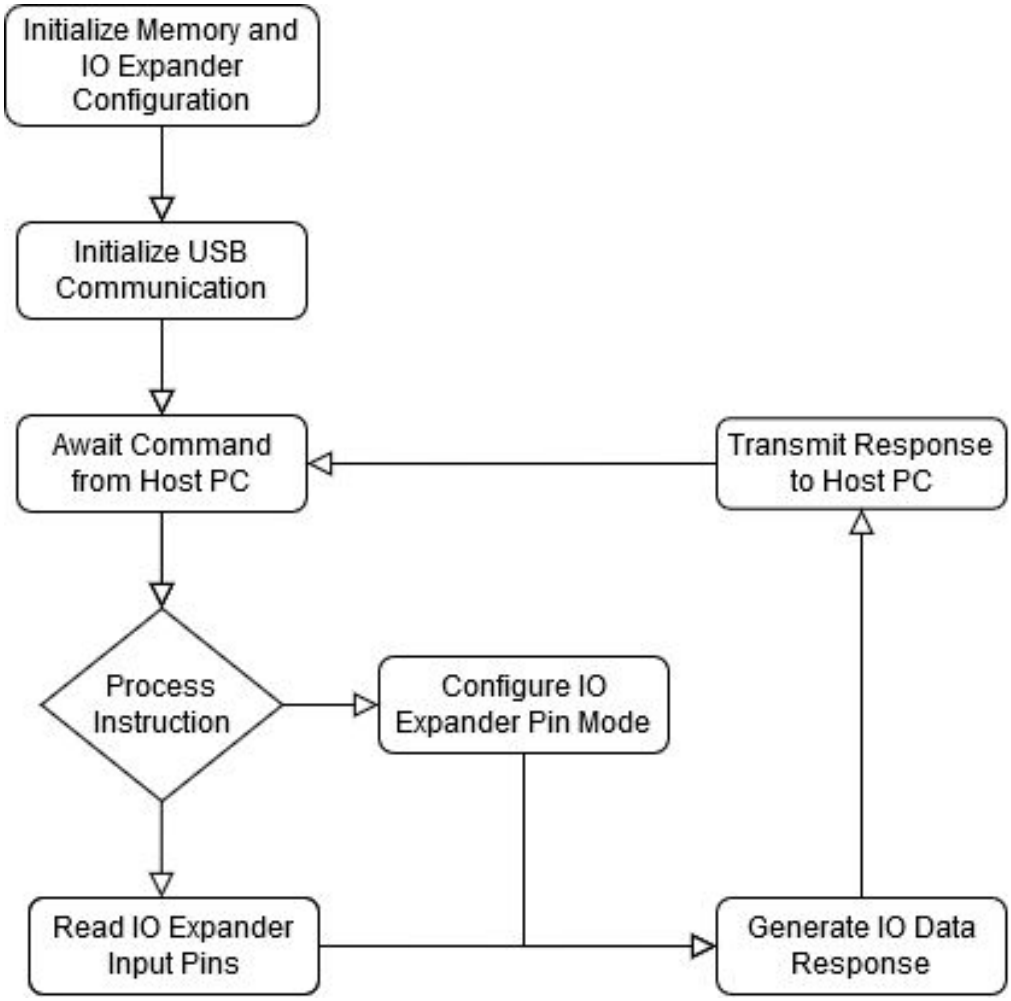

Figure 11. Operation FSM

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. The model of the flow through which the device goes through is visualized above in Figure 11, the Operation FSM.

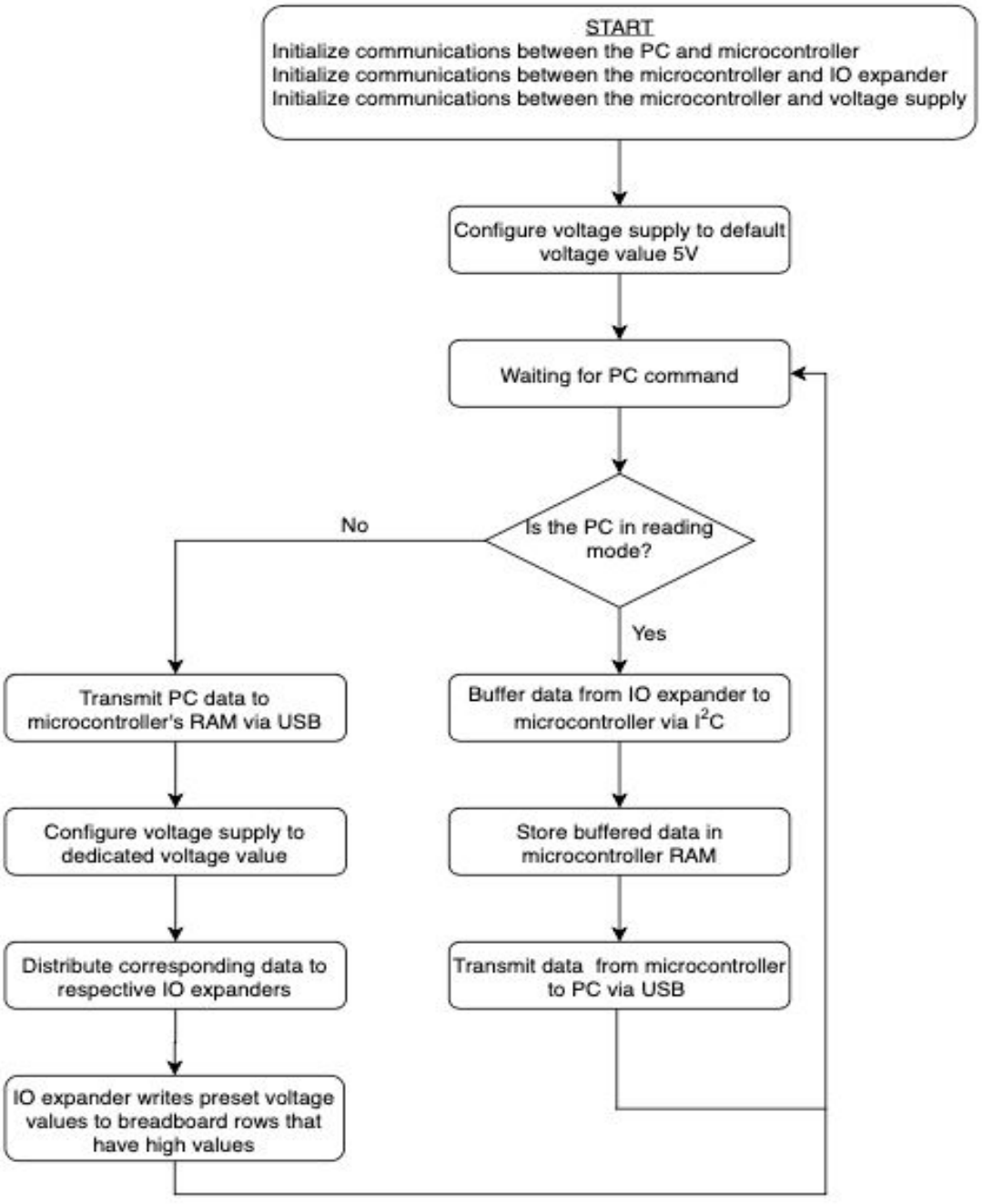

Figure 12. Software Flowchart

In figure 12, 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.

2.4 Tolerance Analysis

The USB transfer functionality is absolutely critical to the success of this project. Due to this, and our limited collective experience in USB communication, the USB data transfer was determined to be the greatest risk to the completion of our design. For the completion of this block, we decided to utilize the open-source library libusb to facilitate the USB communication between the host PC and our device. In libusb, there are two types of device I/O transfer methods: synchronous or asynchronous. In general, if the host is reading from the device, two steps are performed:

1. A request for data is sent to the device
2. The requested data will be received by the host

If the host is writing on the device, two steps are also performed:

1. Data is sent to the device
2. Host will receive a signal containing the success of the process

The difference between the synchronous and asynchronous methods is the number of function calls the two use. The synchronous method only performs on the function call for the above two steps. This is simpler because when the call returns, it tells the host if the transfer was complete or not. On the other hand, the asynchronous method performs 2 seperate function calls for reading or writing. This makes the transfer method able to communicate with multiple end-points, but also increases the complexity.

For our device, we decided to use synchronous I/O transfer because we are only dealing with one device and one array of data. We will be using USB 2.0 full-speed that has a rate of 12±0.24 Mbits/s or roughly 1.5 MB/s [8]. This satisfies the rate we needed as the total amount of data we will be transmitting between the PC and the device is 257-bits. However, there are potential problems that could cause issues. First, once a request is sent, it cannot be canceled. Second, when performing a synchronous call, the function will be blocked. In other words, the application will be in the libusb function until the transfer is completed. If one of our IO Expander does not respond, the command line on the PC would be frozen without displaying any error messages.

The USB 2.0 transfer speed is more than sufficient to transfer the maximum amount of data from the IO Expanders, even when they are operating at the maximum possible speed the microcontroller can manage. The complete I^2C data request framework is as shown in Figure 13. below :

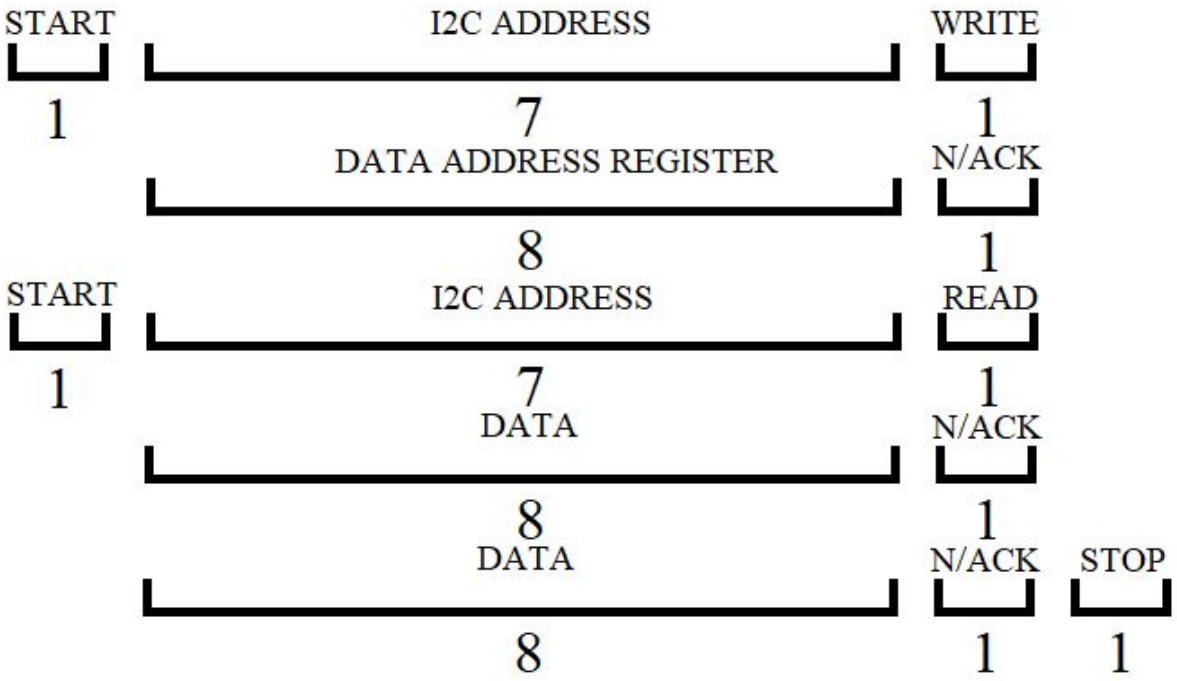

Figure 13. I²C Data Read Framework

This framework requires 46 SCL clock cycles to complete a full 16-bit read from each IO expander, as shown in the framework in Figure 13. To receive data from each IO expander, this requires a total of 46 * 8 = 368 clock cycles to read from all IO expanders. Using the microcontroller's maximum possible I²C clock speed of 1MHz, this brings the maximum theoretical full-board read frequency to ~2.72 kHz, or a total period of 368µs. Since each of these transfers contains a maximum of 128 bits of data (16 IO pins * 8 IO expanders), the data transfer rate of the IO expander block is 348 kb/s. This is far lower than the USB 2.0 full-speed standard of 12Mbit/s, and still will be the bottleneck in read operations.

3. Differences

3.1 Overview

The original project was the Educational Smart Breadboard [9], originally proposed by Chinemelum Chibuko, Minseong Kim, and Mostafa Elkabir in Spring 2018. It was a standalone device capable of reading voltages from the rows of a breadboard and displaying them on an integrated touchscreen display. Their design divided the full breadboard into 8 distinct sub-boards that were multiplexed into the microcontroller using a hierarchical tree mux structure, in addition to a smaller dedicated space for chip testing. Our design reads and writes logic values in every row on the breadboard using IO expanders and a microcontroller via I²C. The state of the breadboard will be configured by and communicated back to a host PC, which the user can use to adjust inputs and observe the output of their logic circuit.

Our approach to tackling the problems with debugging complex breadboard circuits differs from the original design in a few key details. First, instead of incorporating multiplexers in our design, we use IO expanders. With this implementation, instead of only reading subsections of the breadboard, we are able to read and write to individual rows on the breadboard. This also eliminates the need for a dedicated chip testing area, as our whole breadboard can achieve this functionality. However, the trade-off here is that our design is unable to read analog signals. Second, we dropped the standalone touch screen functionality in favor of a PC connected configurable device via USB 2.0. This lets the user conveniently configure the device with the help of usblib to gain command line experience.

3.2 Analysis

The original project's design had a hierarchical tree muxing structure as the core of its data acquisition architecture. This muxing structure allowed the ATmega328 microcontroller at the core of their processing block to select a particular sub-board on their breadboard device[9]. Muxing the breadboard rows into a set of shared pins on the microcontroller allowed the original design to read analog values on the breadboard directly from the microcontroller's ADC pins. However, this design choice limited the microcontroller to operating on a single sub-board at a time, disconnecting all unselected sub-boards from the microcontroller. This limitation greatly inhibits testing of more complex logic circuits, as their design's chip testing "can handle any chip that has stable input-output relationship that allows for fixed truth table."[10] This restriction on chip testing prevents the user from using the testing functionality of the board on more complex circuits, such as register ICs or FSM circuits. Our design choice of using IO expanders removed this restriction, as our design was capable of writing logic values to any combination of rows on the breadboard simultaneously. This capability allows for testing and debugging of any IC or logic circuit where a known set of inputs should produce a known set of outputs, such as with a register IC or FSM circuit. This design difference greatly increases the functionality of our device, and maximizes its utility for debugging, testing, and verification applications.

While a key drawback of using IO expanders in place of muxing the breadboard rows to the microcontroller's ADC pins is that we are limited to reading only digital logic values from the breadboard, we decided this was not a crucial feature for our design for two reasons. The first is that the type of analog circuits being tested on a breadboard by entry-level users would likely have fewer crucial test points for debugging, and therefore have less of a need for a more complex device similar to ours. However, digital logic circuits can have a large degree of complexity that would be well suited to having a large number of test points for debugging, which is why we prioritized digital circuit utility. The second is that the range of analog voltages that a device similar to ours would be able to tolerate is extremely limited, as the voltage range for simple ADC ICs places limits on the maximum voltage that can be applied to the rows on the breadboard. This limited voltage tolerance on ADC ICs limits the range of analogue values that can be present on the breadboard without damaging sensitive components. Digital logic circuits are not restricted by these low voltage tolerances, as most logic ICs operate at supply voltages

< 5V, which is where our microcontroller and IO expanders operate. This means that they are not as affected by the limited voltage tolerance range.

The second key difference is the interface through which users interact with the device. The original project was designed to be a standalone device utilizing a touchscreen display to allow for user configuration and data output. This design decision inhibited automated testing and verification, because it required the user to input the full truth table for whatever logic circuit or IC into the touch display to run tests. However, by instead making the device a USB peripheral, and making the user configure the device and read data through a Python library, we give the user a greater degree of freedom in operating the device. By instead using a code library to allow the user to configure the device and read data from it, automated testing and verification is possible. Along with this, testing of FSMs can now be done, as inputs to a circuit can be set, and the resulting state changes can be directly measured in the logic circuit without excessive probing. This decision to shift away from the touchscreen display greatly increases the utility of our device, while simultaneously decreasing the cost by eliminating an expensive component.

4. Cost and Schedule

4.1 Cost Analysis

4.1.1 Labor

We will use the equation 3.1 to estimate the labor cost for this project:

```
Labor Cost = Salary * Total Work Time * Number of Members
```

Equation 3.1 Labor Cost

According to the Engineering Career Center in UIUC, the average starting salary for a BS graduate is $78,159 per year in the 2017-2018 report [11]. Converting to hours, it will be $37.58 per hour per person. As for the total work time, according to the Office of Provost in UIUC, we are expected to put three times the amount of work time in the number of credit hours we receive per week [12]. That is 12 hours of work time per week for Senior Design. Lastly, we have a total of 3 engineers in this group. The total labor cost will sum up to be $21,646.08 for a semester.

4.1.2 Cost

Part #	Qty	Mft	Vendor	Desc	Price/Unit	Total
ATmega32U4	1	Microchip	Mouser	8-bit AVR Microcontroller	4.08	4.08
1N4148W-7-F	1	Diodes Incorporated	Mouser	SMD Switching/Power Diode	0.16	0.16
MF-MSMF050-2	1	Bourns	Mouser	500mA Resettable Fuse	0.42	0.42
MH2029-300Y	2	Bourns	Mouser	30 Ohm Ferrite Bead	0.10	0.20
TCA9517DGKRQ1	1	Texas Instruments	Mouser	Level-Shifting I^2C Buffer/Repeater	1.38	1.38
SN74HCS32QPWRQ1	1	Texas Instruments	Mouser	4-Channel OR Gate	0.59	0.59
XRP6272IDBTR-F	2	MaxLinear	Mouser	LDO Adjustable Voltage Regulator	0.98	1.96
TPS2115APWR	1	Texas Instruments	Mouser	2-Channel Manual Switching Power Mux	2.33	2.33
MCP23017T-E/SO	8	Microchip	Mouser	16-bit I^2C I/O Expander	1.20	9.60
0476420001	1	Molex	Mouser	MicroUSB Female Connector	0.77	0.77
BSS138DWQ-7	4	Diodes Incorporated	Mouser	2-Channel NMOS Transistor	0.40	1.60
CG0603MLC-05E	2	Bournes	Mouser	ESD 5V TVS Diode	0.41	0.82
FSM6JH	1	TE Connectivity	Mouser	Tactile Momentary Pushbutton Switch	0.13	0.13
CSTNE16M0V51Z000R0	1	Murata	Mouser	16.0 MHz Ceramic Resonator	0.31	0.31

Table 6. Cost for Single Order of Device Components

In table 6, after adding up all of the components, the device's largest components will cost $26.35. This is assuming we are only making one device. If we order parts in bulks, the cost of components will be a lot less.

Part #	Qty	Mft	Vendor	Desc	Price/Unit	Total
ATmega32U4	500	Microchip	Mouser	8-bit AVR Microcontroller	3.39	1,695.00
1N4148W-7-F	500	Diodes Incorporated	Mouser	SMD Switching/Power Diode	0.053	26.50
MF-MSMF050-2	500	Bourns	Mouser	500mA Resettable Fuse	0.238	119.00
MH2029-300Y	1000	Bourns	Mouser	30 Ohm Ferrite Bead	0.021	21.00
TCA9517DGKRQ1	500	Texas Instruments	Mouser	Level-Shifting I^2C Buffer/Repeater	0.798	399.00
SN74HCS32QPWRQ1	500	Texas Instruments	Mouser	4-Channel OR Gate	0.299	149.50
XRP6272IDBTR-F	1000	MaxLinear	Mouser	LDO Adjustable Voltage Regulator	0.538	269.00
TPS2115APWR	500	Texas Instruments	Mouser	2-Channel Manual Switching Power Mux	1.38	690.00
MCP23017T-E/SO	4000	Microchip	Mouser	16-bit I^2C I/O Expander	0.91	3,640.00
0476420001	500	Molex	Mouser	MicroUSB Female Connector	0.48	240.00
BSS138DWQ-7	2000	Diodes Incorporated	Mouser	2-Channel NMOS Transistor	0.124	248.00
CG0603MLC-05E	1000	Bournes	Mouser	ESD 5V TVS Diode	0.107	107.00
FSM6JH	500	TE Connectivity	Mouser	Tactile Momentary Pushbutton Switch	0.097	48.50
CSTNE16M0V51Z000R0	500	Murata	Mouser	16.0 MHz Ceramic Resonator	0.198	99.00

Table 7. Cost for Bulk Order of Device Components

If we were to mass produce the product and buy the parts in bulks the cost for the components will be a lot less. Assuming that we have 500 devices in one batch, in table 7, our cost per device is not $15.5 dollars per device. If we order a higher volume per batch, the price will

decrease even more. For convenience, we will proceed with using $15.5 as bulk production cost to calculate price tag per device.

4.1.3 Sum of Grand Total

Summing up the grand total, we estimate the total device's development cost is $21,672.49. However, if we want to put a price tag on this device, we will have to sum the bulk production cost per unit, board assembly fee, as well as revenue cost. This could be better explained with Equation 3.2.

```
Price = Bulk Production per Unit + Assembly Fee + Revenue
```

Equation 3.2 Device Price Calculation

Using the PCB assembly calculator [13] and preliminary estimates for device requirements, we estimated that the cost for assembling 500 units is $1,467.35, which is $2.935 per device. As for packaging, the cost of the material for packaging will be $255 [13], which is $0.51 per device. Additionally, we estimate it takes a minute to package 3 devices. The total assembly labor cost will be minimum wage multiplied by the number of packaging hours, that is $22.9 or $0.0458 per device. In the end, the total cost for bulk production will be $9,485.4 which is $18.99 per device.

4.2 Schedule

Week	**Task** (Assuming an 8-week period to build a working product)
3/23	Write up RFA by 3/27
Arpan:	Research possible project ideas and determine differences in implementation
Joseph:	Research possible project ideas and determine problem and success criteria
Robert:	Research possible project ideas and determine solution components
3/30	Write up Proposal by 4/3
Arpan:	Research power system and safety and ethics concerns
Joseph:	Determine problem, provide background and draft high level requirements
Robert:	Research solution components and design block diagram
4/6	Begin Design Document
Arpan:	Research components, draw power schematics and discuss safety concerns
Joseph:	Research USB drivers and software side of the project

Robert:	Work on interfacing the control system and data acquisition
4/13	Finish Design Document by 4/17
Arpan:	Finish possible 8-week timeline for device completion
Joseph:	Begin work on the command line interface for programming values
Robert:	Compile schematics, submit PCB design for approval and order required parts
4/20	Prepare for and attend Design Reviews by 4/24
Arpan:	Prepare presentation and oral explanation of power system
Joseph:	Prepare presentation and oral explanation of USB communication
Robert:	Prepare presentation and oral explanation of control and data acquisition
4/27	Finish up Final Report by 5/6
Arpan:	Assemble components and build device; Order new parts if necessary
Joseph:	Integrate feedback from design review for USB communication and interface
Robert:	Build device and integrate feedback from design review for relevant sections
5/4	Submit Notebooks by 5/7
Arpan:	Begin debugging and testing edge cases in power circuits
Joseph:	Debug and test cases in usb communication and interface
Robert:	Begin debugging of control system and data acquisition
5/11	Complete and demo final product
Arpan:	Finalize assembly of device and demo the product
Joseph:	Finish coding and software portion of the project
Robert:	Interface control, power, data and computer systems for a working project

Table 8. Calendar for Project Goals

5. Discussion of 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 [14], so operating components at values above this could result in parts catching on fire or exploding, which endangers people in the surrounding area. 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.' [15] 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" [16]. The safety of the user will always be of utmost importance.

6. Citations

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

[2] 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]

[3] Global Specialties, “GS-400: Solderless Breadboard, 400 Tie-Points, with Bus Strip” *globalspecialties.com,* 2017. [Online]. Available: https://www.globalspecialties.com/solderless-breadboards/breadboards-singles/37-gs-400.html [Accessed: April 2, 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] Arduino, “Arduino Board Leonardo,” *arduino.cc*, 2020. [Online]. Available: https://www.arduino.cc/en/Main/Arduino_BoardLeonardo [Accessed: April 10, 2020]

[7] Github, “pyusb, pyusb,” 2010. [Online]. Available: https://github.com/pyusb/pyusb [Accessed: April 15, 2020]

[8] USB, “2.0 Specification | USB,” 2018. [Online]. Available: https://www.usb.org/document-library/usb-20-specification [Accessed: April 15, 2020]

[9] Chinemelum Chibuko, Minseong Kim, and Mostafa Elkabir, “Educational Smart Breadboard,” *courses.engr.illinois.edu/ece445*, February 7, 2018. [Online] Available: https://courses.engr.illinois.edu/ece445/pace/view-topic.asp?id=25937 [Accessed: March 22, 2020]

[10] Chinemelum Chibuko, Minseong Kim, and Mostafa Elkabir, “Educational Smart Breadboard,” *courses.engr.illinois.edu/ece445*, May 2, 2018. [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwjihfXMxfDoAhUuAZ0JHVNpBQgQFjAAegQIARAB&url=https%3A%2F%2Fcourses.engr.illinois.edu%2Fece445%2Fgetfile.asp%3Fid%3D12787&usg=AOvVaw0CjRvgghpLUp_1NDQv0Y-V [Accessed: April 17, 2020]

[11] University of Illinois, “Illini Success Annual Reports,” 2020. [Online]. Available: https://illinisuccess.illinois.edu/annual-reports/ [Accessed: April 17, 2020]

[12] University of Illinois Office of the Provost, “Credit Hour Definition,” *provost.illinois.edu/policies,* 2020. [Online] Available: https://provost.illinois.edu/policies/policies/courses/credit-hour-definition/ [Accessed: April 17, 2020]

[13] BITTLE, ‘PCB Assembly Quote Calculator,’ 2020 [Online]. Available: https://www.7pcb.com/PCB-Assembly-Quote.php?d3=0&d5=1&c6=500&c8=10&c11=0&c13=1&c18=6&c20=1&c23=4&c25=0&send=Calculate&x=117&y=9 [Accessed: April 10, 2020]

[14] George Leger, “Common Breadboard Specifications,” *circuitspecialists.com*, March 29, 2014. [Online] Available: https://www.circuitspecialists.com/blog/common-breadboard-specifications/

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

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