Universal Chip Programmer Project Document

May 14th, 2023

By:
Bryson Flint
Elizabeth Lindsay
Darius Salagean

Table of Contents

1. Overview

1.1. Executive Summary

Based on the Glasgow Explorer hardware platform, a universal chip programmer that is able to leverage existing chip-level debug weaknesses will be delivered, in addition to investigating new chips of related chip families. The project is currently in the completed phase. A Jupyter Notebook GUI connects to the Glasgow Explorer and gateware/applet code written in Amaranth HDL is used to build the system. Using a 48 Hz clock, the system is able to write and read program memory from a PIC16F1615 microcontroller which is connected to the system using a custom PCB breakout board. Data that is read is saved to a text file.

1.2. Team Contacts and Protocols

Table 1: Team Member Roles

Name	**Contact**	**Role**	**Contributions**
Elizabeth Lindsay	lindsael@oregonstate.edu	Note taker Leader	Designing GUI and save functionality in Jupyter Notebook
Bryson Flint	flintbr@oregonstate.edu	Devil's advocate Time Manager	Recording Waveforms, decoding commands, and PCB
Darius Salagean	salageda@oregonstate.edu	Facilitator Coordinator	Coding applet for Glasgow in Amaranth HDL

Table 2: Team Protocols

Protocol	**Assessment Parameters**
Weekly progress reports	Each member must attend weekly group meetings, and participate by reporting any progress that they have made.
Project partner meeting	Attend bi-weekly meeting with a project partner that will last upwards of two hours.

Late/Absent	If a group member is to be late, the group member must notify the group in advance, and if they are going to be absent, they must give the group one day's notice.
Comments and clear communication in code	Code should be commented neatly, functions should have headers with purposeful explanations, and variables and file names should have clear names.
Meeting notes	Group note taker will take notes during project partner meetings. Notes will be uploaded to the team google drive.
Team google drive	Information related to the project will be stored in the team drive. Information will be sorted in folders, and there will be no loose files.
Respectful communication	Team members are expected to be respectful of each other's ideas, and listen to group member communication in a respectful manner.

1.3. Gap Analysis

The Purpose of this project is to research the firmware of microcontrollers and data chips to help develop the security of these devices. The reason this is being done is to lower the odds of a data chip being hacked into and the data being stolen. The ultimate goal is to find a solution to prevent any data chip from being broken into.

Assumptions that can be made for this project is to aim for a result that works universally but this is likely to not to be accomplished. The research being done is to make it apply to the data chips accessible but won't be able to make it completely universal. It can also be assumed that the Glasgow explorer will be used along with the Jupyter notebook. This will require using the code Amaranth.

The project starts by referencing the documents provided by the project partner [1], [2]. These documents go over how to extract the firmware and different methods of trying to prevent this.

1.4. Timeline

Table 3: Project Timeline

Task	Expected Completion	Responsible Member	Completed (y/n)
Attend first group meeting	Week 1 Monday 9/26/2022	All group members	Yes
Meet with project partner	End of Week 2 10/2/2022	All group members	Yes
Choose project option and goals	End of Week 3 10/14/2022	All group members	Yes
Complete Project Document Section 1 Draft	End of Week 3 10/14/2022	All group members	Yes
Communication Evaluation	Week 4 Friday 10/21/2022	All group members	Yes
Complete Project Document Section 2 Draft	Week 6 Friday 11/4/2022	All group members	Yes
Complete Design Impact Assessment Draft	Week 6 Friday 11/4/2022	All group members	Yes
Record PIC commands	Week 9 Monday 11/21/2022	Bryson Flint	Yes
Make Amaranth test code	Week 9 Monday 11/21/2022	Darius Salagean	Yes
Instal and learn Jupyter Notebook	Week 9 Monday 11/21/2022	Elizabeth Lindsay	Yes
Final System Design	End of Week 10 12/2/2022	All group members	Yes
Glasgow Explorer Expected Arrival	End of Week 12 1/20/2023	All group members	Yes
Submit PCB for Manufacturing	End of Week 15 2/10/2023	Bryson	Yes
Glasgow Training Meeting 1	End of Week 18 3/3/2023	All group members	Yes
System Verification 1	Wednesday of	All group	Yes

	Week 20 3/15/2023	members	
Glasgow Training Meeting 2	End of Week 20 3/17/2023	All group members	Yes
Glasgow Training Meeting 3	End of Spring Break 3/31/2023	Darius Salagean	Yes
Final Implementation	End of Week 25 5/5/2023	All group members	Yes
Final System Checkoff	Wednesday of Week 26 5/10/2023	All group members	Yes
Presentation and Poster	End of Week 30 6/9/2023	All group members	No

1.5. References and File Links

[1] M.Schink and J.Obermaier, “Exception(al) Failure - Breaking the STM32F1 Read-out Protection,” 17 March 2020, Accessed: 11 October 2022. [Online]. Available: https://blog.zapb.de/stm32f1-exceptional-failure/

[2] J.Obermaier and S. Tatschner, “ Shedding too much Light on a Microcontroller’s Firmware Protection,” Accessed: 11 October 2022. [Online]. Available: https://www.usenix.org/system/files/conference/woot17/woot17-paper-obermaier.pdf

1.6. Revision Table

Table 4: Revision Table Section 1

5/14/2023	Elizabeth Lindsay: Added to timeline and updated completion status
5/10/2023	Elizabeth Lindsay: Updated executive summary to match project status
11/18/2022	Darius Salagean: Added to timeline and updated executive summary.
10/31/2022	Bryson Flint: Addressed comments for section one Darius Salagean: Addressed comments for section one, added to timeline. Elizabeth Lindsay: Formatted table of contents

10/10/2022	Bryson Flint: Initial document creation. Set up section outlines. Initial content for 1.3 Darius Salagean: Initial content for 1.4 Elizabeth Lindsay: Initial content for 1.1, 1.2

2. Impacts and Risks

2.1. Design Impact Statement

One public, health, and safety impact that the universal programmer has is related to reducing the number of programmers manufactured and the reducing the risk of dangers during chip production. The chemicals that workers are exposed to during production are extremely dangerous, as well as all the radiation and electrostatic particles that their suits protect them from [1]. Workers wear protective PPE to protect themselves, but incidents still happen, so lowering the amount of chips needing to be produced, by creating a universal chip would minimize the risk.

Related to health and safety impacts, there could be a positive environmental impact associated with this project, due to the production of a universal programmer, where less programmers need to be made later, since there's just one programmer. Taiwan Semiconductor Manufacturing Company, the largest manufacturer in the world, produced 15 million tons of carbon emissions in 2020 [2]. Therefore using one universal programmer, instead of multiple programmers would have a positive impact in this area.

One cultural and social impact of the universal programmer relates to exploiting data gathered by a microcontroller. For example, the STM32F1 series of microcontrollers was discovered to allow for around 90% of its firmware to be downloaded [3]. STM32 microcontrollers are used in a variety of applications such as printers and vehicle electronics. By downloading firmware, an experienced programmer would be able to see exactly how the microcontroller is controlling the device it is in and gain access to industry-secret data that they could leverage to cause harm. With the creation of a universal programer we would be enabling more people to be able to recreate or discover security flaws who would be able to take the responsible course of action and report their findings to the manufacturer.

The primary economic factor of the Universal chip programmer is the chips themselves. The project can use different forms of debuggers, FPGA boards, data analyzers, and chips. All of these come in different ranges of prices, but the reason chips are more expensive is because of the current chip shortage caused by COVID 19. According to Bain & Company [4], the chip shortage is said to end in roughly 2024. The reason this is impactful is because our project is trying to expose chips on their security flaws and the research could impact the shortage by making debuggers lower in value by replicating

their use with a basic FPGA board, therefore changing the security approach on these chips.

2.2. Risks

Table 5: Risk Table

Risk ID	Risk Description	Risk Category	Risk Probability	Risk Impact	Performance Indicator	Action Plan
R1	Project Catches Fire	Hardware	Low	High	Project is engulfed in flames.	Take out fire and replace needed parts.
R2	Lose/Stolen Hardware	Hardware	Low	Med	Parts are missing and can not be found between members.	Replace the missing parts.
R3	Hardware doesn't arrive on time	Timeline	Med	High	Shipping delays on hardware, or the package gets lost in transit.	Check shipping links and then meet with a project partner about using existing hardware.
R4	Teammate has emergency	Organizati on	Low	High	Teammate doesn't show up to meeting unexpectedly	Teammate lets the rest of the team know 24hrs in advance.
R5	Faulty Hardware	Technical	Med	High	Hardware behaves unexpectedly and a cause can not be determined	Test with a multimeter, refer to the product datasheet, talk with the project partner about getting new hardware.
R6	Not meeting original goals	Organizati onal	Med	Med	Technical goals are not completed on time, or assignments are missing.	Find out why the goal wasn't meant and adjust accordingly.
R7	Go over	Cost	Med	Med	Needed Finances	Find equivalent

	300$ budget				exceed $300.	hardware that is cheaper, talk to the project partner about sponsorship.

2.3. References and File Links

2.3.1. References

[1] "StackPath," *www.ehstoday.com*. https://www.ehstoday.com/industrial-hygiene/article/21917701/safety-in-the-semiconductor-industry

[2] S. Shead, "The global chip industry has a colossal problem with carbon emissions,"*CNBC*, 03 Nov. 2021. [Online]. Available: https://www.cnbc.com/2021/11/03/tsmc-samsung-and-intel-have-a-huge-carbon-footprint.html

[3] M.Schink and J.Obermaier, "Exception(al) Failure - Breaking the STM32F1 Read-out Protection," 17 March 2020, Accessed: 11 October 2022. [Online]. Available: https://blog.zapb.de/stm32f1-exceptional-failure/

[4] P. hanbury, A. Hoecker and M. Schallehn, "A chip shortage recovery guide," *Bain.com*, 25 March 2022, Accessed: 04 Nov 2022. [Online]. Available: https://www.bain.com/insights/a-chip-shortage-recovery-guide/#:~:text=The%20automation%2 and%20 industrial%20sectors,2023%20(see%20 Figure%201).

2.3.2. File Links

2.4. Revision Table

Table 6: Revision Table Section 2

10/24/2022	Bryson Flint: Initial content for 2.2 R1 and R2. Darius Salagean: Initial section outline, content for 2.2 R5 and R6. Elizabeth Lindsay: Initial table creation for 2.2, content for 2.2 R3, R4, and R7.
4/26/2023	Elizabeth Lindsay: Added content for impact statement from impact design assessment.

3. Top-Level Architecture

3.1. Block Diagram

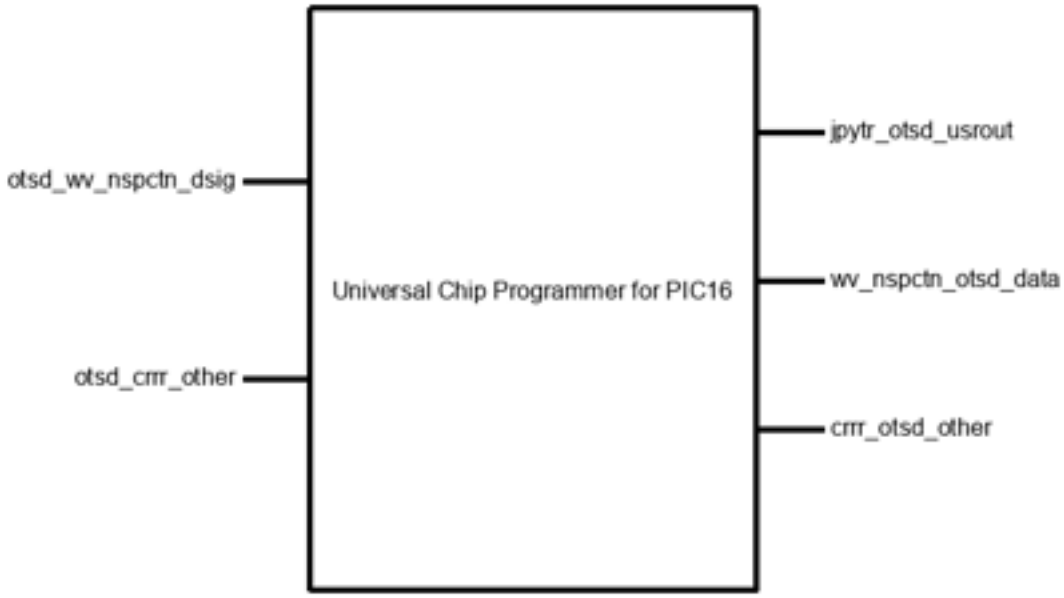


Fig. 1. Black Box Diagram for Universal Programmer

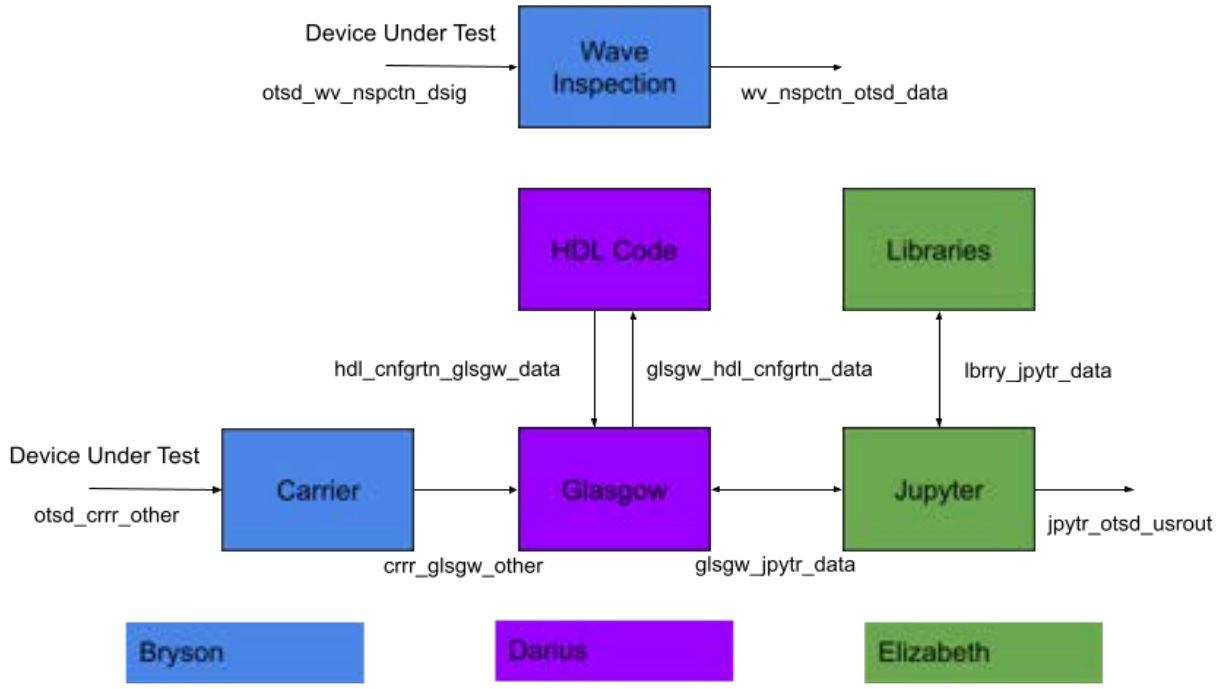


Fig. 2. Block Diagram for Universal Programmer

3.2. Block Descriptions

Table 7: Block Description Table

Name	**Description**

Carrier Champion: Bryson Flint	A PCB that will connect the Glasgow explorer to the PIC microcontroller. The connection will be made using 2 JTAG connections. There will also be pins attached to help read the signals being sent to the Glasgow from the microcontroller, as well as a connector to allow programming from the PICKit 3. The ultimate purpose of this block is to allow a stable connection between the microcontroller and the Glasgow.
Glasgow Champion: Darius Salagean	This will be the hardware to act as the universal programmer. A Glasgow Interface Explorer will be used to create signals that will mimic programming commands that will be sent to the device under test. The HDL Configuration code will be uploaded to the Glasgow to configure the internal FPGA. The Jupyter block would connect to the Glasgow and use the Library to send instructions for which command should be executed. The Glasgow would also transmit data received from the PIC15F1615 to Jupyter.
Jupyter Champion: Beth Lindsay	This will be where a Jupyter notebook program will be implemented to analyze the signals received from the universal programmer. The universal programmer will be implemented using a Glasgow Explorer, which will be what the Jupyter Notebook is receiving signals from. Through this connection, the Jupyter notebook will read data from the Glasgow Explorer pertaining to each of the 5 commands that the universal programmer will be running on the microcontroller. These commands include: load configuration, load data from program memory, read from program memory, increment address, and reset address. The purpose of the Jupyter Notebook receiving these signals is to organize the signals received into a readable graph for users to analyze. This will be accomplished by writing code that takes the received signals as inputs and then organizes the data into each of the different commands and plots the signals to a graph. The code contained in the Jupyter Notebook will leverage functions from the library that will allow the output from the Glasgow Explorer to be viewed by the user to determine that each of the commands ran as intended, and allows for verification that each of the programmed commands are successful in execution.

Library Champion: Beth Lindsay	The library contains the specific functions that the Jupyter Notebook calls. These functions are based on the HDL code that is run on the microcontroller, as well as data gathered from using a logic analyzer in sync with a programmer for the pic16. The functions included in the library are of the five commands that the Glasgow will run which includes: Load Configuration, Load data from program memory, Read data from load memory, Read data increment address, and Read data reset. The purpose of the library is to have a block that deals with creating each function, and handling the bulk of actually executing each command. This is managed by calling the functions included in the library within the Jupyter notebook, and having the return of each of the functions displayed within the Jupyter notebook. This block is necessary to the system design and system requirements by satisfying the conditions that the universal programmer will be able to program a microcontroller, that the system will Load Configuration, Load data from program memory, Read data from load memory, Read data increment address, and Read data reset, and that the system will be able to record data.
Wave Inspection Champion: Bryson Flint	This block is the pre operation to the project. Using a programmer and data analyzer with our selected Microcontroller (PIC16), this will show waves of commands performed by the microcontroller. The logic analyzer will read the function waves from inputting a command to the PIC16 microcontroller. Then the function waves will be outputted to the Data Receiver (Saleae Software). Checking off this block will be verifying the input values active high and active low are correct and the output waves contain the expected function readings that will be replicated in a later part of the project. Specifically, the waves will be used to show that an FPGA/Glasgow explorer can replicate this data. The command it receives and the responses it gives will be recorded by the logic analyzer. The programmer will supply the commands to the microcontroller.
HDL Configuration Champion: Darius Salagean	This will be the code responsible for configuring the hardware within the Universal Programmer block. The Universal Programmer block will be a Glasgow Explorer FPGA which will send data to a chosen microcontroller. The code will configure the Glasgow Explorer to replicate the digital signals for programming commands needed to read and write data to a PIC16F1615 (PIC) microcontroller. Commands replicated will be Load Configuration, Load Data for Program Memory, Read Data from Program Memory, Increment Address, and Reset address. The code will be written using the Amaranth HDL language and uploaded directly onto the Glasgow Explorer.

3.3. Interface Definitions

Table 8: Interface Table

Name	Properties
otsd_wv_nspctn_dsig	● **Logic-Level:** 2.64V (high) Minimum high reading ● **Logic-Level:** 0.66V (low) Maximum low reading ● **Max Frequency:** 5MHz
otsd_crrr_other	● **Other:** Configured for PIC16F1615 carrier board ● **Other:** Design in KiCad 6.0 ● **Other:** Based on reference PCB for NXP chip
jpytr_otsd_usrout	● **Type:** Data will be saved to a file ● **Type:** Data will be displayed to the user as printed text ● **Usability:** This output needs to be used intuitively by 9/10 users who report that the output is understandable..
lbrry_jpytr_data	● **Messages:** When a command is run from the library, data is passed from the jupyter notebook, into the library, and then provides a return to the Jupyter notebook. ● **Messages:** Each of the commands run on the Glasgow corresponds to a function in the library including: Load Configuration, Load data from program memory, Read data from load memory, Read data increment address, and Read data reset. ● **Other:** The library code will be imported into the Jupyter notebook code for use within the Jupyter block..
wv_nspctn_otsd_data	● **Messages:** Increment Address 6 bit opcode followed by delay(1us min.). ● **Messages:** Load Data 6 bit opcode followed by delay(1 us min.) and then 16 bits of data. ● **Messages:** Read Data 6 bit opcode followed by delay(1 us min.) and then 16 bits of data ● **Messages:** Load Config 6 bit opcode followed by delay(1us min.) and then 16 bits of data.

hdl_cnfgrtn_glsgw_data	● **Messages:** Data to be read from memory, 16 bits wide ● **Other:** Needs to configure a 3.3V and Ground pin. ● **Other:** Needs to set a variable clock signal, max of 5MHz
crrr_glsgw_other	● **Other:** Printed PCB ● **Other:** Connects to Glasgow Explorer ● **Other:** Final schematic of design
glsgw_jpytr_data	● **Messages:** The Jupyter Notebook will send 6 bit opcodes to the Glasgow ● **Messages:** The Jupyter Notebook will send 14 bit data to the Glasgow ● **Messages:** The Jupyter Notebook will receive 14 bit data from the Glasgow.
glsgw_hdl_cnfgrtn_data	● **Messages:** Data to be loaded into memory, 16 bits wide ● **Messages:** Program code to be sent, 6 bits wide. ● **Other:** Internal clock signal, 10 MHz max

3.4. References and File Links

3.4.1. References

3.4.2. File Links

3.5. Revision Table

Table 9: Revision Table Section 3

5/14/2023	Bryson Flint: Updated Block Diagram to contain interface names.
3/10/2023	Darius Salagean: Block Description table and Interface Table added.
3/6/2023	Darius Salagean: Initial section creation and Block Diagram figures added.

4. Block Validations

4.1. Carrier

4.1.1. Description

A PCB that will connect the Glasgow explorer to the PIC microcontroller. The connection will be made using 2 JTAG connections. There will also be pins attached to help read the signals being sent to the Glasgow from the microcontroller, as well as a connector to allow programming from the PICKit 3. The ultimate purpose of this block is to allow a stable connection between the microcontroller and the Glasgow.

4.1.2. Design

The design of the PCB is to sit on top of the Glasgow explorer and allow the microcontroller to attach right in the middle of the PCB. This will allow minimal wiring and a secure connection to be made. In figure 3, J5 and J7 will be pins that will allow the data being sent to the glasgow to be read while programming.

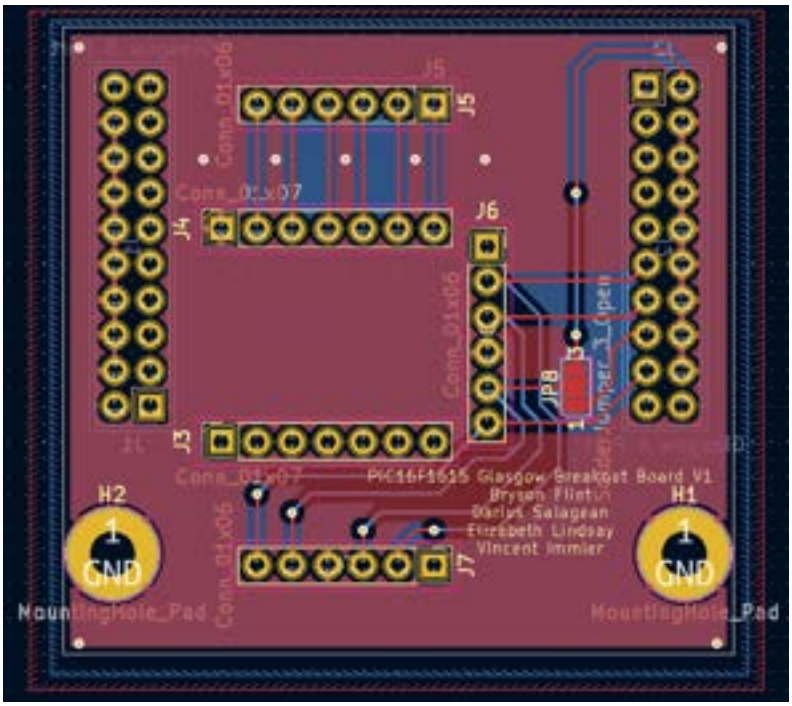


Fig. 3. PCB Schematic

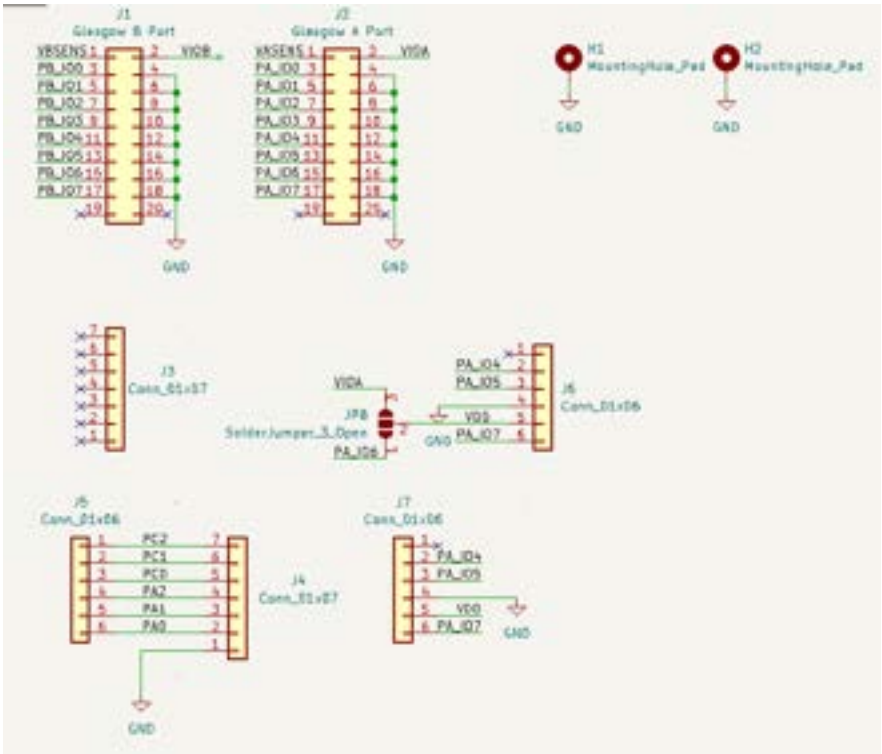


Fig. 4. Schematic of Design

4.1.3. General Validation

The first choice when designing the PCB was choosing the software. KiCad was chosen to develop this software because the design courses at Oregon State University teach KiCad, which definitely makes it a more convenient option. An alternative program could have been Altium, which has a lot more options for users. The options can be seen here [1], the reason this isn't worth using this program is because it is not free and the team does not have training to use this software. The project partner also supplied us with a file that was already using KiCad, it made the most sense to continue using this program to develop the PCB for the project.

The other notable choices made was the actual PCB itself. There are no surface mount parts on the PCB and it uses entirely Jtags and pins. This is mainly because the PCB acts mostly as a "Carrier" hence the name of the block. The pins will solder to the Glasgow and the PIC16, this will allow a secure connection and remove possibility of the parts coming apart unintentionally. The PIC16 comes on its own board with pin holes, this allows the design to get away from using surface mounts. The alternative to the design we chose would be to just put a bunch of pins onto a board and then connect them using jumpers and wires instead of using traces. However, this would be messy looking and not be as efficient as just using traces to connect everything.

4.1.4. Interface Validation

Table 10: Interface Validation table for Carrier

otsd_crrr_other : Input		
Other: Design in KiCad 6.0	Learned how to use this tool and was used to develop the previous model.	PCB was drafted and confirmed in KiCad 6.0.
Other: Based on reference PCB for NXP chip	This will allow a similar protocol to be made from a different chip to the chip we are using.	Old file will be referenced to the current file showing the changes that were made.
Other: Configured for PIC16F1615 carrier board	This will allow the chip we chose to work with the Glasgow and fit the PCB.	The pins on the PCB were measured to the pin layout of the PIC16F1615.
crrr_glsgw_other : Output		
Other: Connects to Glasgow Explorer	This is the chosen FPGA and requires connection.	The pins on the PCB were measured to the pin layout of the Glasgow Explorer.
Other: Final schematic of design	Showing the connections that are being made in a simple form.	Final schematic is presented with the PCB.
Other: Printed PCB	Actually having the PCB.	PCB is in person and presented with the PIC16 and Glasgow Explorer.

4.1.5. Verification Plan

1. Present the printed PCB with the schematics ready to show and explain the connections being made.

2. Show that the PCB fits the Glasgow Explorer and the PIC16 securely.

3. Show the old PCB provided by the project partner and compare it to the PIC16 PCB.

4.1.6. References and File Links

[1] https://www.altium.com/altium-designer/compare/kicad-eda

4.1.7. Revision Table

Table 11: Revision Table for Carrier

3/10/2023	Bryson Flint: Section 4.1 Created and first draft made.

4.2. Wave Inspection

4.2.1. Description

This block is the pre operation to the project. Using a programmer and data analyzer with our selected Microcontroller (PIC16), this will show waves of commands performed by the microcontroller. The logic analyzer will read the function waves from inputting a command to the PIC16 microcontroller. Then the function waves will be outputted to the Data Receiver (Saleae Software). Checking off this block will be verifying the input values active high and active low are correct and the output waves contain the expected function readings that will be replicated in a later part of the project. Specifically, the waves will be used to show that an FPGA/Glasgow explorer can replicate this data. The command it receives and the responses it gives will be recorded by the logic analyzer. The programmer will supply the commands to the microcontroller.

4.2.2. Design

The programmer comes with a USB cable that plugs into the laptop and allows the commands to be inputted. The microcontroller comes with pins that connect to the programmer. Additionally, a 3.3V power supply will also connect to the microcontroller. Unfortunately the program that connects to the programmer from the PC is a bit inconsistent and sometimes doesn't require the outside voltage source but this will be supplied regardless. To see the signals being outputted from the microcontroller, The data analyzer will connect with some clips and run its own software to show the signal waves.Refer to figure number 4 to see how data analyzer is connected to the programmer/microcontroller. There will also be a hex file uploaded to the programmer, the actual data of the uploaded code isn’t actually important to finding the signal waves, but it allows the data analyzer to see all of the commands being used to program the code.

Looking at figure 5 below shows that this block is not exactly integrated into the main portion of the project. However, the findings from this block is crucial in understanding the commands that will be replicated in the finished product. The goal of this project is to replicate the wave data that is given from the microcontroller onto the FPGA board, which in this case is the Glasgow Explorer.

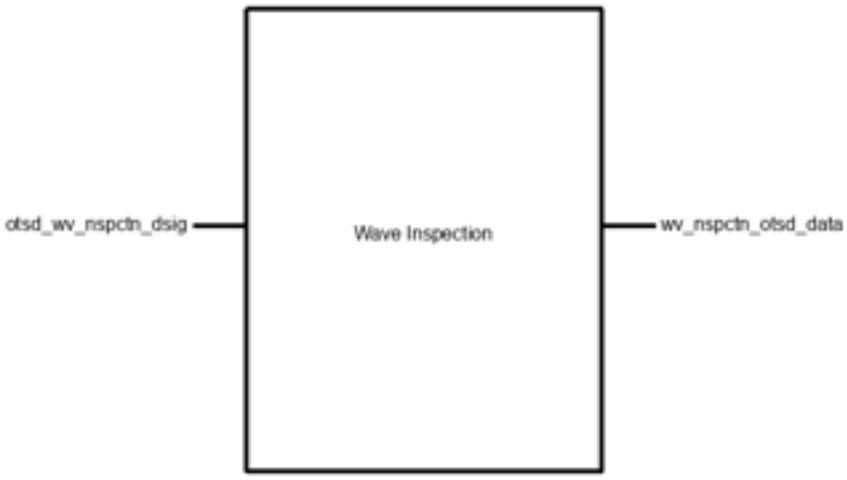


Fig. 5. Black Box Diagram

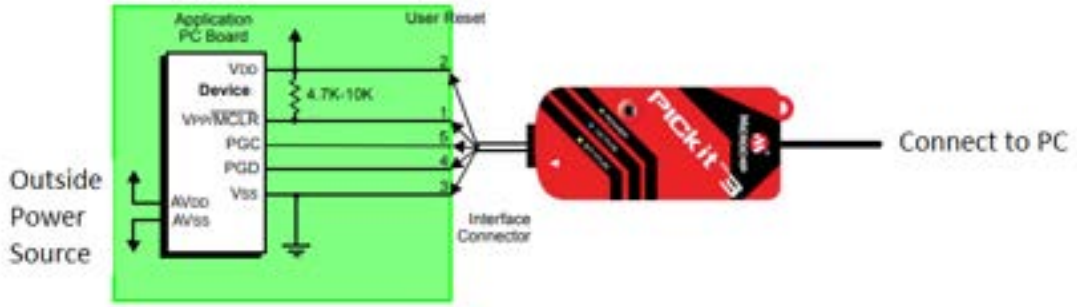


Fig. 6. Wiring Diagram and Connections

4.2.3. General Validation

The objective of the project is to expose security flaws in common chips, this design was chosen because it's simple. Having a simple design makes it easier to show the security flaws and how easily accessible they can be. This block as shown in the block diagram (figure 1) is pretty much separate from the final product of the project. The reason it's important is because the project will be trying to recreate the results from the programmer provided with the PIC16 microcontroller. In short, the programmer can supply commands to the microcontroller and make it output specific data. The data analyzer will pick up these signals and show us the results of the commands. One of the first documentations provided to the team is by M.Schink and J.Obermaier [1], this is relevant because they show how cheap and simple it can be done and it is important to keep that pattern when presenting the possible flaws in different chips. Additionally, since the project is trying to show that the security is vulnerable enough that anyone can do it, it is important to use the most accessible chips to work with.

There is no need to complicate this process and waste time making it more automated or official. This will actually do a good job showing how easy it is to pick up on signals that are not usually meant to be seen. This will also allow the team to put their efforts into the actual project of replicating the signals from the programmer into the glasgow explorer. Since this is a pre operation to the project, none of the budget is not put into this, instead everything needed to see the signals is provided by the Project Partner. Which also means that this was pretty much the only option for inspecting the waves as we were provided with this design and given a few tips on how to operate it. However, the team was given the opportunity to choose between a few different microcontrollers to study for this project. The PIC16 was chosen because it is far more simple and will require less time than other chips. The more complicated chips can be used to test the universal aspect of the project, but for the purpose of this block, using the simple one will allow more data to be obtained early on and overall be more useful.

The alternative to the design we chose/were given would be to create a system that has a better data analyzer and a different microcontroller with more functions. This would allow more data to be obtained and create a more universal design for copying chips. Any more changes than this would pretty much integrate this block with the rest of the blocks which pretty much defeat the purpose of the project. The purpose is to show how cheap it can be to obtain this data and implement it in a separate device. Additionally, not enough was given in the project budget to afford more equipment for this portion of the project. Due to the chip shortage P. Hanbury, A. Hoecker, and M. Schallehn talked about the chip shortage that was relevant during COVID-19 [2]. This is relevant because in deciding what microcontroller and parts to use for the project, this issue makes the options limited. Therefore the provided and cheap options is clearly the design the project has to go with.

4.2.4. Interface Validation

Table 12: Interface Validation table for Wave Inspection

otsd_wv_nspctn_dsig : Input (commands)		
Max Frequency: 5MHz	Section 8.0 page 31 is where the timing (100 ns Tckl and Tckh = 200ns = 5MHz) is specified of the microcontroller data sheet. [3]	The output from the data analyzer will show the max frequency being done.
Vmax: 2.64V (high) Minimum high reading	2.64V is the specified value for when the signal is at its active high. [3] Section 8.0 page 31.	A voltage reader will be used to verify roughly 2.64V is being read when in active high.
Vmin: 0.66V (low) maximum low reading	0.66V is the specified value for when the signal is at its active	A voltage reader will be used to verify 0.66V is being read when

	low. [3] Section 8.0 page 31.	in active low
wv_nspctn_otsd_data : Output (waves)		
Messages: Load Config. 6 bit opcode followed by delay(1us min.) and then 16 bits of data.	Definition of command on section 4.3 page 17 Figure 4-1. [3]	The Saleae software will show the waveform of the command including the opcode, delay, and data.
Messages: Load Data 6 bit opcode followed by delay(1 us min.) and then 16 bits of data.	Definition of command on section 4.3 page 18 Figure 4-2. [3]	The Saleae software will show the waveform of the command including the opcode, delay, and data.
Messages: Read Data 6 bit opcode followed by delay(1 us min.) and then 16 bits of data.	Definition of command on section 4.3 page 18 Figure 4-3 . [3]	The Saleae software will show the waveform of the command including the opcode, delay, and data.
Message: Increment Address 6 bit opcode followed by delay(1us min.).	Definition of command on section 4.3 page 19 Figure 4-4. [3]	The Saleae software will show the waveform of the command including the opcode, delay, and data.

4.2.5. Verification Plan

1. Demonstrate and assemble the microcontroller/programmer as seen in figure 6.

2. Demonstrate the IPE and IDE software required to load the hex file onto the microcontroller from the programmer.

3. Connect an oscilloscope probe to demonstrate the input **otsd_wv_nspctn_dsig.**

4. Show that the Vmax for active high is 2.64V (minimum high reading), Vmin for active low is 0.66V (maximum low reading), and the frequency maxes out at 5 MHz.

5. Disconnect oscilloscope and prepare connections to demonstrate output.

6. Connect the data analyzer to the existing assemble microcontroller/programmer. Use the first three channels and ground. Channel 1 goes to VAP, Channel 2 goes to PGD, Channel 3 goes to PGC, Ground goes to VCC. Does not need to connect in any particular order.

7. Open the Saleae software to begin reading the output **wv_nspctn_otsd_data**.

8. Once the IPE software is ready to program and the Saleae software is ready to record, start recording data and then proceed to program the microcontroller.

9. Stop the recording once the IPE Software finishes programming the microcontroller.

10. Locate the messages for the four outputs described in the interface table, Load config., Load Data, Read Data, and Increment Address. Demonstrate that the values they have specified match the expected waves.

4.2.6. References and File Links

[1] M.Schink and J.Obermaier, “Exception(al) Failure - Breaking the STM32F1 Read-out Protection,” 17 March 2020, Accessed: 11 October 2022. [Online]. Available: https://blog.zapb.de/stm32f1-exceptional-failure/

[2] P. Hanbury, A. Hoecker and M. Schallehn, “A chip shortage recovery guide,” *Bain.com*, 25 March 2022, Accessed: 04 Nov 2022. [Online]. Available: https://www.bain.com/insights/a-chip-shortage-recovery-guide/#:~:text=The%20automation%2 and%20 industrial%20sectors,2023%20(see%20 Figure%201).

File Links:

MicroController Data Sheet:

[3] **https://ww1.microchip.com/downloads/en/DeviceDoc/40001720C.pdf**

4.2.7. Revision table

Table 13: Revision Table for Wave Inspection

3/6/23	Bryson Flint: Document imported into Project Document and edited to match.
2/10/23	Bryson Flint: Description edited, Further detail in intro and included wire diagram, added more information in general validation and made design more clear, interfaces updated to current project status, verification plan detailed further. Revision statement added.
1/20/2023	Bryson Flint: Document Created and first draft made.

4.3. Jupyter

4.3.1. Description

This will be where a Jupyter notebook program will be implemented to analyze the signals received from the universal programmer. The universal programmer will be implemented using a Glasgow Explorer, which will be what the Jupyter Notebook is receiving signals from. Through this connection, the Jupyter notebook will read data from the Glasgow Explorer pertaining to each of the 5 commands that the universal programmer will be running on the microcontroller. These commands include: load configuration, load data from program memory, read from program memory, increment address, and reset address.

The purpose of the Jupyter Notebook receiving these signals is to organize the signals received into a readable graph for users to analyze. This will be accomplished by writing code that takes the received signals as inputs and then organizes the data into each of the different commands and plots the signals to a graph. The code contained in the Jupyter Notebook will leverage functions from the library that will allow the output from the Glasgow Explorer to be viewed by the user to determine that each of the commands ran as intended, and allows for verification that each of the programmed commands are successful in execution. This block will satisfy system requirements through allowing the universal programmer to connect to a user interface. It will also satisfy the requirement of creating a user interface that is “clear” to 9/10 users which will be satisfied through the block verification testing.

4.3.2. Design

The Jupyter notebook block will take inputs from the library, as well as the glasgow explorer. The connection to the glasgow explorer will be bidirectional, so it will function as an output as well, since one of the purposes of the Jupyter notebook is to send program data to the FPGA, and in return receive data back from it.

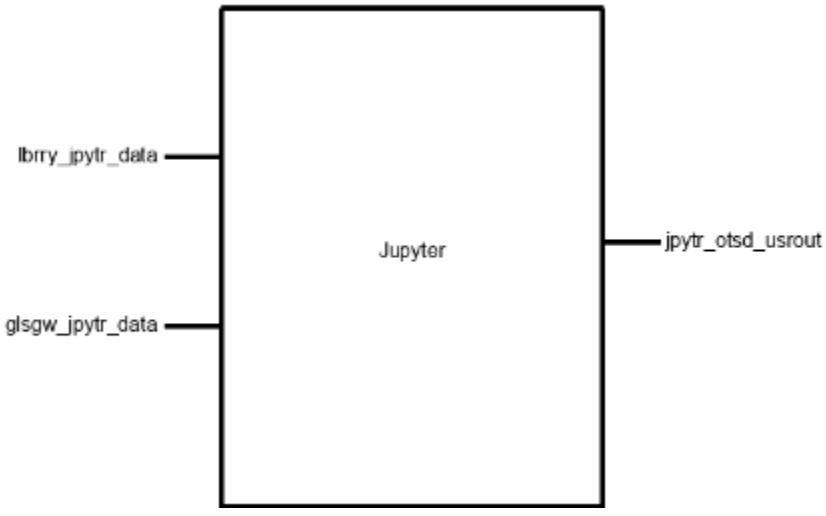


Fig. 7. Black box diagram

Code outline:

Import library():Imports library functions for each of the commands

Run Commands()

Load configuration(): runs specific library command on the Glasgow Explorer.

Read data config(): receives data in return from the function.

Load data from program memory(): runs specific library command on the Glasgow Explorer.

Read data load memory(): receives data in return from the function.

Read from program memory(): runs specific library command on the Glasgow Explorer.

Read data read memory(): receives data in return from the function.

Increment address(): runs specific library command on the Glasgow Explorer.

Read data increment address(): receives data in return from the function.

Reset address(): runs specific library command on the Glasgow Explorer.

Read data reset(): receives data in return from the function.

Process data()

Create graphs(inputs from each of the read data functions): Outputs graphs for each of the commands that are reasonable and intuitive to the user.

Graphs: The Jupyter Notebook interface allows for code to be run in block chunks, and the final block will be a graph block instead of a code block, and it will display each of the 5 graphs that were generated.

4.3.3. General Validation

In order to meet the requirements given by the rest of the system, the Jupyter Notebook block must receive data from the Glasgow Explorer about the results of each command run, implement the commands using library functions, as well as provide a readable output of the results of each of the functions for the user to analyze. All of this will be

implemented by writing code into a Jupyter Notebook application that sends and receives data to the Glasgow Explorer universal programmer.

Jupyter Notebook was chosen to be the application used for this block because it is an open source GUI that is programmable using python[1]. Using an open source application is important, as the use of one has no cost to the group. The use of python within the Jupyter notebook will allow for custom design and flexibility of how the commands will be run, and how the data received from the commands will be presented to the user. This application is also compatible with the Glasgow Explorer from which it needs to be able to communicate with[2].

From the Jupyter Notebook, commands will be imported from the library block and then run through the GUI onto the Glasgow Explorer. The library is separate from the Jupyter Notebook block, since these commands are also needed when creating the HDL for the Glasgow in another block, so this will allow for simpler implementation of the whole system. Pulling out the library from the Jupyter Notebook is a design choice that is not technically necessary to the success of the overall project or for the functionality of the block, but it makes implementation of the block simpler, and more manageable. It ensures that other blocks using the library will be using the same code as the Jupyter Notebook to prevent any confusion, or mismatched commands.

Processing the data into graphs meets the needs of the block by providing the user with a readable output from the commands. The creation of a readable and understandable user output is also a project partner requirement, so it satisfies a whole requirement by itself. Graphs are chosen to be the preferred output method, because the data received will be in the form of signals, and when those signals are plotted, the user will be able to compare the signals to those of the signals generated by a separate logic analyzer. This will allow for verification that each of the commands worked correctly if they match the signals generated by the logic analyzer, which also meets another engineering requirement for the system.

Other ways to implement a GUI that could send and receive data to a universal programmer would be to use LabView as a GUI instead of Jupyter Notebook[3]. This alternative option was not chosen as it is not open source like Jupyter Notebook is, so each of the team members would need to acquire a license for it. Not being open source also means that learning the application would be more difficult as there would be less online resources. Overall, Jupyter Notebook was chosen over LabView, as there was a choice to implement Jupyter Notebook and the Glasgow in tandem, or use LabView for everything. The team came to this decision because there is a learning curve for each option, but the team had experience with Verilog and python, and learning LabView wouldn’t be as accessible.

4.3.4. Interface Validation

Table 14: Interface Validation table for Jupyter

jpytr_otsd_usrout : Output		
Type: Provides a user with waveforms in the form of a graph that allow the user to see that each of the functions work correctly.	This output was chosen to be in the form of graphs so that the user would be able to compare the graph generated from the Jupyter Notebook with the one generated from the logic analyzer.	This property is met in the design in the portion that processes data and will generate a graph for each of the five commands that are run.
Usability: This output needs to be used intuitively by 9/10 users who report that the output is understandable..	In order to prove intuitive usability as required by the project partner this property was added. It allows for testing by people who are not familiar with the project to prove the usability.	In order to meet this in the design, the graphs generated in the process data function will be clearly labeled and have clear formats for the user to understand. All graphs will be displayed in their own block within the Jupyter Notebook for further clarity. The functions in read data will be easily accessible for users to run without needing extensive knowledge of Python.
lbrry_jpytr_data : Input		
Messages: Each of the commands run on the Glasgow corresponds to a function in the library including: Load Configuration, Load data from program memory, Read data from load memory, Read data increment address, and Read data reset.	The Jupyter Notebook needs to be able to access each of the 5 commands in order to run them on the Glasgow explorer, so the library must provide them to the Jupyter Notebook.	This is implemented in the design by using the imported functions from the library in each of their corresponding run commands() calls so that the command is run on the Glasgow Explorer.
Other: The library code will be imported into the Jupyter notebook code for use within the Jupyter block.	For the sake of simplicity and to not write duplicate code, a library of common commands including those run on the Glasgow Explorer was pulled	The first section of code defined in the Jupyter Notebook will be importing the library of functions for future use.

	out of the Jupyter code. These functions then need to be imported to the Jupyter code so that they can be used.	
glsgw_jpytr_data : Input		
Messages: The Jupyter Notebook will receive 14 bit memory data from the Glasgow Explorer.	This interface is necessary in order to receive the correct data from the Glasgow explorer about the return of each command function.	This is implemented in each of the read data commands, where the signals are received from the correct bit of memory for each signal.
Other: The Jupyter Notebook will receive data corresponding to each of the five commands.	The Jupyter Notebook must receive data for each of the commands so that each of the commands that are run can be plotted and analyzed, without the signals from each command crossing over each other and getting combined.	This is implemented by creating a read data() function for each of the 5 commands instead of just creating one read data function.

4.3.5. Verification Plan

1. Create Outline of Jupyter Notebook Python including all code blocks and graph blocks.
2. Import Library block functions for each of the five commands.
3. Verify **lbrry_jpytr_data** by checking that there are 5 different commands - one for each function, and that each command was properly imported by running the command and checking that an output exists.
4. Create an api that will wrap the jupyter notebook allowing program data to be sent and received.
5. Write Python code for creating graphs for each of the 5 functions and generate graphs in correct graph block chunks.
6. Verify **glsgw_jpytr_data** by checking that there are 5 different inputs received from the Glasgow Explorer that are independent of each other.
7. Verify **glsgw_jpytr_data** again by confirming that the generated graphs contain expected data. If the data is not expected, then the functions need to be changed, and re-tested so that the universal programmer can be confirmed as replicating those 5 commands as required by the project partner.
8. Verify **jpytr_otsd_usrout** by collecting a group of 10 students to try out the GUI, and ask their opinions on how intuitive the interface is.
9. Take input from the students and revise the GUI until the condition is met as required by the project partner.

These steps are the best ways to verify the interfaces, as they test each of the interfaces thoroughly and in a way that could be recreated. The steps also verify some of the engineering and partner requirements for the whole system.

4.3.6. References and File Links

References

[1] M. Driscoll, “Jupyter Notebook: An Introduction”, *Real Python* [e-journal]. https://realpython.com/jupyter-notebook-introduction/#:~:text=The%20Jupyter%20Notebook%20is%20an,the%20people%20at%20Project%20Jupyter.

[2] “Glasgow”. Github: 2023. https://github.com/GlasgowEmbedded/glasgow

[3] “What is LabVIEW”, *National Instruments* [e-journal]. https://www.ni.com/en-us/shop/labview.html

File Links

Block Diagram P2

4.3.7. Revision Table

Table 15: Revision Table for Jupyter

2/11/23	Elizabeth Lindsay: Revised document based on instructor comments and peer reviews. Content changes included adding information to the description about how the block was necessary to the project, and which system requirements it satisfied. Other changes were adding a short description to the block design, in order to further explain the block design and reasoning behind it. More sources were added to the validation in order to further prove why the design was chosen; specifically sources related to explaining what the glasgow explorer is/does were added. Finally the verification plan was shortened in order to be more concise, so that it is easier to read and follow. A short paragraph was also added to the end to explain why this validation process would work for the block.
1/20/23	Elizabeth Lindsay: Created and wrote remaining sections. Edited and added content to sections 1 and 4.
1/12/23	Elizabeth Lindsay: Created interface properties for section 4.
11/30/22	Elizabeth Lindsay: Created description for section 1.

4.4. Library

4.4.1. Description

The library contains the specific functions that the Jupyter Notebook calls. These functions are based on the HDL code that is run on the microcontroller, as well as data gathered from using a logic analyzer in sync with a programmer for the pic16. The functions included in the library are of the five commands that Glasgow will run which includes: Load Configuration, Load data from program memory, Read data from load memory, Read data increment address, and Read data reset. The purpose of the library is to have a block that deals with creating each function, and handling the bulk of actually executing each command. This is managed by calling the functions included in the library within the Jupyter notebook, and having the return of each of the functions displayed within the Jupyter notebook. This block is necessary to the system design and system requirements by satisfying the conditions that the universal programmer will be able to program a microcontroller, that the system will Load Configuration, Load data from program memory, Read data from load memory, Read data increment address, and Read data reset, and that the system will be able to record data.

4.4.2. Design

The only connection that the library will have to other blocks will be to the Jupyter Notebook, but the interface will be bidirectional. This means that the Library block will be both receiving and sending data to the Jupyter notebook. This is essential, as the commands that the Library houses include a write command which needs to send data, and a read command that will need to receive data.

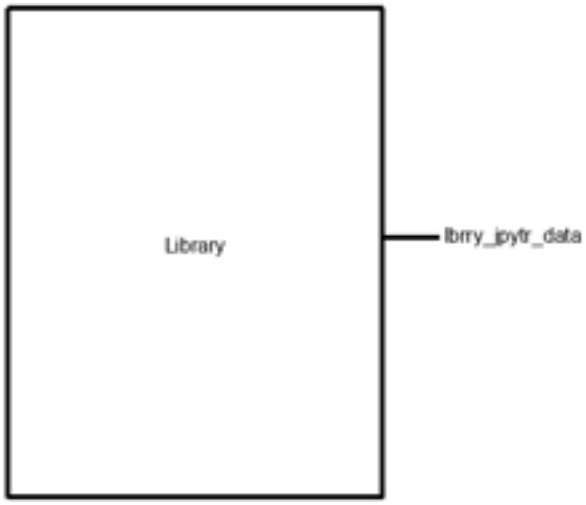


Fig. 8. Library block diagram

Code Outline:

Load Configuration(): sends the opcode for load configuration.
Load data(): sends the opcode for load data.
Read data(): sends the opcode for reading data. Then receives and saves the read data.
Increment address(): sends the opcode for increment address.
Reset(): sends the opcode for reset.

4.4.3. General Validation

In order to meet the requirements of the system, the Library must contain 5 different functions, each of them corresponding to a specific command that the team intends to run on the Glasgow Explorer. These five functions are: Load Configuration, Load data from program memory, Read data from load memory, Read data increment address, and Read data reset.

These five functions were chosen as they are the most basic programming commands to send, and are necessary for reading data[1]. This was determined through reading through the data sheet for our chosen chip. The data sheet also included the opcode necessary to send the command to the Glasgow Explorer, so each of the functions that are implemented are hard coded with the opcode that they are meant to send.

Some of the commands are more complex than others, specifically the read data command. The read data command needs to not only send an opcode but it needs to receive the data that it is requesting. The data that it receives is in the form of 14 bit memory data [2]. So when running the read data command and receiving data back, there needs to be a spot to store that 14 bit memory data, so that it can be saved. This was implemented by creating a variable to save the data into, so that it could be printed to the user, and saved to a file later, in order to satisfy system requirements.

4.4.4. Interface Validation

Table 16: Interface Validation table for Library

lbrry_jpytr_data : Output		
Messages: When a command is run from the library, data is passed from the jupyter notebook, into the library, and then provides a return to the Jupyter notebook.	The purpose of the Jupyter Notebook is to run commands on the Glasgow, so that means that it needs to first retrieve those commands from the library and communicate in both directions with it.	This is implemented in the instantiation for the library within the jupyter notebook code, as well as when the commands from the library are successfully called on the jupyter notebook. These commands are deemed successful when they can

		provide accurate inputs and outputs.
Messages: Each of the commands run on the Glasgow corresponds to a function in the library including: Load Configuration, Load data from program memory, Read data from load memory, Read data increment address, and Read data reset.	The Jupyter Notebook needs to be able to access each of the 5 commands in order to run them on the Glasgow explorer, so the library must provide them to the Jupyter Notebook.	This is implemented in the design by using the imported functions from the library in each of their corresponding run commands() calls so that the command is run on the Glasgow Explorer.
Other: The library code will be imported into the Jupyter notebook code for use within the Jupyter block.	For the sake of simplicity and to not write duplicate code, a library of common commands including those run on the Glasgow Explorer was pulled out of the Jupyter code. These functions then need to be imported to the Jupyter code so that they can be used.	The first section of code defined in the Jupyter Notebook will be importing the library of functions for future use.

4.4.5. Verification Plan

1. Create two different files for code- one for the library, one for the jupyter notebook.
2. Instantiate functions for each of the five commands, where they send their specific opcodes, and receive data when needed.
3. In the Jupyter Notebook, import Library block functions for each of the five commands.
4. Verify **lbrry_jpytr_data** by checking that there are 5 different commands - one for each function, and that each command was properly imported by running the command and checking that an output exists. Verify that for the read data function, there is data being printed to the screen and saved to a file

4.4.6. References and File Links

References:

[1] “PIC16(L)F1615/9”, *Microchip*,

https://ww1.microchip.com/downloads/en/DeviceDoc/40001770D.pdf

[2] "PIC16(L)F1615/9 Memory Programming", *Microchip,* https://ww1.microchip.com/downloads/en/DeviceDoc/40001720C.pdf

File links:

Block Diagram P2

4.4.7. Revision Table

Table 17: Revision Table for Library

3/10/23	Elizabeth Lindsay: Created content for each section including copying over the description and interface property sections.

4.5. Glasgow

4.5.1. Description

This will be the hardware to act as the universal programmer. A Glasgow Interface Explorer will be used to create signals that will mimic programming commands that will be sent to the device under test. The HDL Configuration code will be uploaded to the Glasgow to configure the internal FPGA. The Jupyter block would connect to the Glasgow and use the Library to send instructions for which command should be executed. The Glasgow would also transmit data received from the PIC15F1615 to Jupyter.

4.5.2. Design

The universal programmer project is attempting to read and write memory from a PIC microcontroller. The Glasgow Explorer hardware will be used for the Glasgow block. The **hdl_cnfgrth_glsgw_data** from HDL Configuration will configure the FPGA on the Glasgow Explorer. The **glsgw_hdl_cnfgrtn_data** interface describes the data that the Glasgow Explorer will receive and the HDL Code will need to properly process the data and set the Glasgow Explorer to perform the correct action. The **glsgw_jpytr_data : Output** interface describes the data that the Jupyter Notebook will send and receive from the Glasgow. Fig. 9 shows the black box diagram.

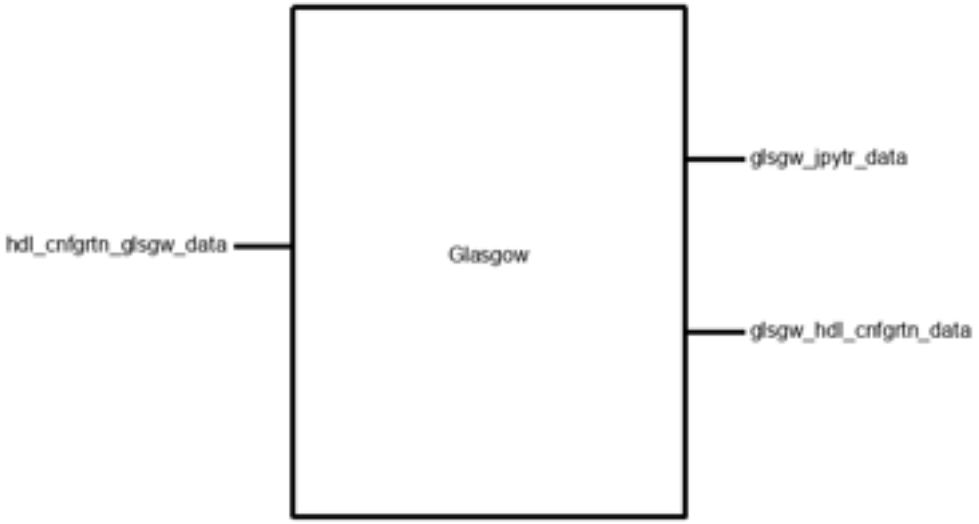


Fig. 9. Black box diagram of Glasgow block.

Glasgow Explorer has two ports, A and B, which have 8 GPIO pins that can supply 150mA of current and 1.8V - 5V. Each port also has a 5V power supply at the VIO_ pin. The layout of the ports can be seen in Figure 10 below. GND and VIOA will be used as power and ground for the PIC, while the clock signal will be sent on PAIO4 and data on PAIO5.

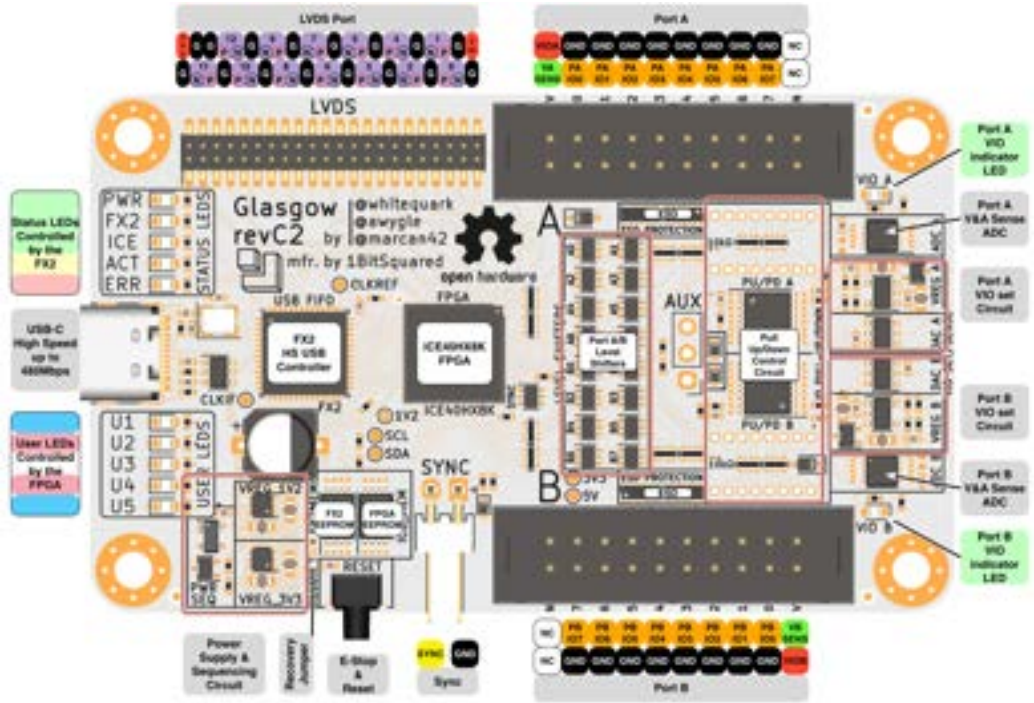


Fig. 10. Port Layout for the Glasgow Explorer

4.5.3. General Validation

When we first met with our project partner we were given two options for hardware to use to create the universal programmer. The first was the myRIO my National Instruments and the second was the Glasgow Explorer, which is open source hardware. We chose to use the Glasgow Explorer as it had the most flexibility. The price for the Glasgow was also much better at 145$, compared to the myRIO at 592$ for the basic academic version. This means that all 3 group members could have access to the Glasgow hardware, vs. only 1 myRIO.

The Glasgow Explorer would require code to be written in Jupyter Notebook, which uses Python. All of our group members are familiar with Python. The MyRIO would require code to be written using LabView. LabView is a proprietary software that would require all of our group members to learn an entirely new programming language to use.

Due to the connection between the PIC and Glasgow being a PCB board, the pins PAIO4 and PAIO5 were chosen as they worked the best with the layout of the PIC breakout board. Port A was chosen for this same reason, though Port B could have also been used as they are interchangeable.

4.5.4. Interface Validation

Table 18: Interface Validation table

hdl_cnfgrtn_glsgw_data : Output		
Messages: Data to be read from memory, 16 bits wide	16 bits was chosen as data in the microcontroller is 14 bits wide, however 16 clock cycles are needed to output data on the data line as outlined by the PIC programming guide.	For a PIC16F1615 [1]: ● 16 clock cycles are required to send out data from PIC. (4.3 Program/Verify Commands) ● Data being read is sent as some unknown value x, 14 data bits, and a final 0 bit. (Figure 4-3) ● 16 bits is in line with set specifications.
Other: Needs to set a variable clock signal, max of 5MHz	5Mhz was chosen based on the timing specifications of the PIC datasheet. This can be variable as the clock is not always on.	For a PIC16F1615 [1]: ● The Clock low and high pulse both need to last for a minimum of 100 ns. (Table 8-1: AC/DC Characteristics Timing Requirements For Program/Verify Mode) ● This means the clock period is 200 ns = 5Mhz which is

		what was chosen.
Other: Needs to configure a 3.3V and Ground pin	3.3V was chosen as this is a standard for low voltage microcontrollers.	For a PIC16F1615 [2]: ● Maximum standard operating voltage on the VDD pin is +5.5V. Minimum voltage for 32MHz internal clock operation is 2.5V. (35.3 DC Characteristics Table 35-1) ● 3.3V is within these parameters and allows for full use of the microcontroller.
glsgw_jpytr_data : Output		
Messages: The Jupyter Notebook will send 14 bit data to the Glasgow	Since the PIC microcontroller uses 14 bit data memory, anything we want to load into the PIC needs to be 14 bits.	For a PIC16F1615 [1]: ● Data is sent as the start bit 0, 14 data bits, and end bit 0. (Table 4-1) ● The start and stop bits are set by the HDL code.
Messages: The Jupyter Notebook will receive 14 bit data from the Glasgow	Since the PIC microcontroller uses 14 bit data memory, anything we read from the PIC will be in 14 bit form.	For a PIC16F1615 [1]: ● Data is read as an unknown bit x, 14 data bits LSB to MSB, and end bit x. (Table 4-3) ● Only need the 14 data bits, the start and stop bits can be cut off.
Messages: The Jupyter Notebook will send 6 bit opcodes to the Glasgow	The PIC microcontroller uses 6 bits for the available programing opcodes.	For a PIC16F1615 [1]: ● Program commands are 6 bits long and consist of 5 data bits and an unknown value x. (Table 4-1) ● 6 bits is in line with the set specifications.
glsgw_hdl_cnfgrtn_data : Input		
Messages: Data to be loaded into memory, 16 bits wide	16 bits was chosen based on the requirements outlined by the PIC programming guide.	For a PIC16F1615 [1]: ● 16 clock cycles are required to send in data to the PIC. (4.3 Program/Verify

		Commands) ● Data is sent as the start bit 0, 14 data bits, and end bit 0. (Table 4-1) ● 16 bits is in line with the set specifications.
Messages: Program code to be sent, 6 bits wide.	6 bits was chosen as the size of programming commands is 6 bits	For a PIC16F1615 [1]: ● Program commands are 6 bits long and consist of 5 data bits and an unknown value x. (Table 4-1) ● 6 bits is in line with the set specifications.

4.5.5. Verification Plan

The HDL code will be uploaded to the Glasgow Explorer hardware which would configure it. The Wave inspection block will be connected to the Glasgow in order to get readings of the waveforms. The connection needed would be to the variable clock signal, data line, and clear line. The verification process will be as follows:

1. The Wave Inspection block will begin a recording.
2. A chosen command will be sent to the Glasgow from the Jupyter Notebook.
3. Once the command is finished the recording will be stopped.
4. The recording will be inspected and checked if it matches the waveform seen in the PIC data sheet [1] Table 4-1.
    a. If the command was read data, the Jupyter Notebook must display the same data read as the data seen in the recording.
5. This process will be repeated for all 5 commands.
6. The system passes verification if all 5 commands match the expected waveforms and the read command sends data to the Jupyter Notebook.

4.5.6. References and File Links

[1] "Pic12(L)F1612/16(L)F161X memory programming," *Microchip*, Sep-2013. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/40001720C.pdf. [Accessed: 20-Jan-2023].

[2] "PIC16(L)F1615/9 Data Sheet," *Microchip*, Oct-2014. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/40001770D.pdf. [Accessed: 20-Jan-2023].

4.5.7. Revision Table

Table 19: Revision Table for Library

3/12/23	Darius Salagean: Initial section creation. All parts filled out.

4.6. HDL Configuration

4.6.1. Description

This will be the code responsible for configuring the hardware within the Universal Programmer block. The Universal Programmer block will be a Glasgow Explorer FPGA which will send data to a chosen microcontroller. The code will configure the Glasgow Explorer to replicate the digital signals for programming commands needed to read and write data to a PIC16F1615 (PIC) microcontroller. Commands replicated will be Load Configuration, Load Data for Program Memory, Read Data from Program Memory, Increment Address, and Reset address. The code will be written using the Amaranth HDL language and uploaded directly onto the Glasgow Explorer.

4.6.2. Design

The universal programmer project is attempting to read and write memory from a PIC microcontroller. The hdl_cnfgrth_glsgw_data interface describes the necessary ports that the HDL code must configure on theGlasgow Explorer so that it will be able to transmit and receive data from the PIC using its specifications. The Glasgow Explorer will need to be configured to supply a clock signal, power, ground, and a two way connection to send and receive data. The glsgw_hdl_cnfgrtn_data interface describes the data that the Glasgow Explorer will receive and the HDL Code will need to properly process the data and set the Glasgow Explorer to perform the correct action. Fig. 11 shows the black box diagram. Once the HDL code is written it will be downloaded onto the FPGA using a USB connection.

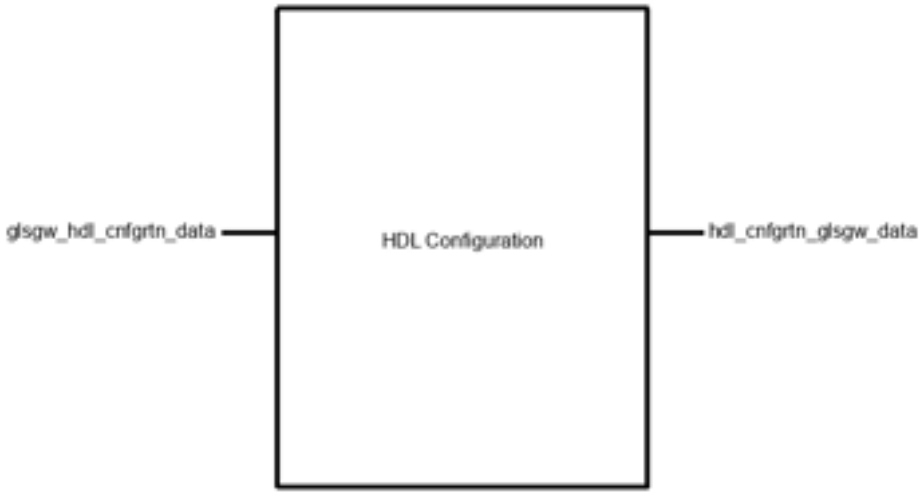


Fig. 11. Black box diagram of HDL Configuration block.

The code will be designed to respond to 5 certain inputs and data. Each of the 5 commands has an associated opcode that the code will check for and perform a specific operation. In the case of Load Configuration and Load Data, there will also be data given as an input that will need to be sent. The Read Data command will have incoming data that it will need to store. All commands will send data on the falling edge of the clock and have a delay of 1us between command and data read/write. Pseudocode for this implementation is seen in Figure 12.

```
from amaranth import *

class Config(Elaboratable):

    def __init__(self, command, data):

    self.cmd = command
    self.data = data

        # Ports
        self.3v3  = Signal()
        self.GND = Signal()
        self.clk = Signal()
        self.send = Signal()

    def elaborate(self, platform):
        m = Module()

        if slef.cmd = Load Config {
            send config command
            delay clock
            send data
            delay clock
        }
        if slef.cmd = Load Data {
            send load data command
            delay clock
            send data
            delay clock
        }
        if slef.cmd = Read Data {
            send config command
            delay clock
            read data
            delay clock
        }
        if slef.cmd = Inc Address {
            send Inc command
            delay clock
        }
        if slef.cmd = Reset Address {
            send reset command
            delay clock
        }

        return m
```

Fig. 12. Pseudocode for the HDL Config code written in Amaranth

4.6.3. General Validation

For our project, we were required to use the Glasgow Explorer as our FPGA hardware. The Glasgow Explorer is an open source project, so what we make could easily be replicated by someone else if we were to share our code and set-up. Since our system is

using a Glasgow Explorer the Amaranth HDL language was chosen to make the code. This is due to the fact that the Glasgow firmware is written in Python 3 and the FPGA is modified using Amaranth HDL [1], which is a Python derived language.

The PIC microcontroller was chosen due to its simple design. It is an 8 bit microcontroller with only 10 programming commands in total and one section of program memory. Other microcontrollers we could have chosen were the STM32U4 and the NXP PA16. The NXP PA16 is also an 8 bit microcontroller, however it has RAM, EEPROM, and Flash memory that all have specific commands required to access and read. The STM32U4 is a 32-bit microcontroller which has SRAM and Flash memory. Since it is 32 bits, it has a much larger memory set which makes it much more difficult to read the contents of program memory.

The PIC requires an external power source and ground connection in order to function, so configuring the Glasgow Explorer to have these is necessary. The PIC is programmed by using a clock connection and a two way data bus, so these connections also must be configured by the HDL. The clock signal is not constant and needs to be delayed between commands, so the clock signal needs to be variable. The Glasgow Explorer has 16 GPIO pins, meaning it has enough to connect to the PIC. The pins can also output specific voltages, which makes delivering a constant 3.3v possible. The HDL code will configure the internal hardware to make the data pin an output when sending data and an input when receiving data.

4.6.4. Interface Validation

Table 20: Interface Validation table for HDL Configuration

hdl_cnfgrtn_glsgw_data : Output		
Messages: Data to be read from memory, 16 bits wide	16 bits was chosen as data in the microcontroller is 14 bits wide, however 16 clock cycles are needed to output data on the data line as outlined by the PIC programming guide.	For a PIC16F1615 [2]: ● 16 clock cycles are required to send out data from PIC. (4.3 Program/Verify Commands) ● Data being read is sent as some unknown value x, 14 data bits, and a final 0 bit. (Figure 4-3) ● 16 bits is in line with set specifications.
Other: Needs to set a variable clock signal, max of 5MHz	5Mhz was chosen based on the timing specifications of the PIC datasheet. This can be variable as the clock is not always on.	For a PIC16F1615 [2]: ● The Clock low and high pulse both need to last for a minimum of 100 ns. (Table

		8-1: AC/DC Characteristics Timing Requirements For Program/Verify Mode) ● This means the clock period is 200 ns = 5Mhz which is what was chosen.
Other: Needs to configure a 3.3V and Ground pin	3.3V was chosen as this is a standard for low voltage microcontrollers.	For a PIC16F1615 [3]: ● Maximum standard operating voltage on the VDD pin is +5.5V. Minimum voltage for 32MHz internal clock operation is 2.5V. (35.3 DC Characteristics Table 35-1) ● 3.3V is within these parameters and allows for full use of the microcontroller.
glsgw_hdl_cnfgrtn_data : Input		
Messages: Data to be loaded into memory, 16 bits wide	16 bits was chosen based on the requirements outlined by the PIC programming guide.	For a PIC16F1615 [2]: ● 16 clock cycles are required to send in data to the PIC. (4.3 Program/Verify Commands) ● Data is sent as the start bit 0, 14 data bits, and end bit 0. (Table 4-1) ● 16 bits is in line with the set specifications.
Messages: Program code to be sent, 6 bits wide.	6 bits was chosen as the size of programming commands is 6 bits	For a PIC16F1615 [2]: ● Program commands are 6 bits long and consist of 5 data bits and an unknown value x. (Table 4-1) ● 6 bits is in line with the set specifications.

4.6.5. Verification Plan

The HDL code would normally be uploaded to the Glasgow Explorer hardware which would configure it to be able to perform set actions based on the input opcodes and data. For verification, the HDL code can be simulated using a test bench to demonstrate

how the code would perform if it were uploaded to the Glasgow Explorer. This simulation would create a waveform that can be used to verify the specifications of the interfaces. If the simulated waveforms match the specifications, it is expected that the code will perform the same when uploaded to the Glasgow Explorer. The verification process will be as follows:

1. Amaranth HDL code will be written and a test bench will be presented.
2. The test bench will perform the following tests
    a. The Load Data For Program Memory command will be sent (0 1 0 0 0 x)
    b. The 14 bit opcode data for adding 13 to W register (1 0 1 1 0 0 0 0 0 1 1 1 1 1) will be sent.
    c. The Read Data From Program Memory command will be sent (0 0 1 0 0 x)
3. Code will be ran and generate a .vcd file.
4. .vcd file will be ran with GTKWave in order to display the created wave form.
5. The waveform must have the following signals to be considered complete:
    a. A constant output that is high (will represent the 3.3V pin).
    b. A constant output that is low (will represent the 0V pin).
    c. A clock signal which does not exceed a 5Mhz period.
    d. The simulated Load Data For Program Memory that was sent by the test bench. The execution is seen in Fig. 13.
        i. The program command bits 0 1 0 0 0 x will be sent.
        ii. After a delay of 1 us (TDLY), the 0 start bit will be sent followed by the 14 data bits from the test bench, followed by the 0 stop bit.
    e. The simulated Read Data From Program Memory that was sent by the test bench following the data from the previous command. The execution is seen in Fig. 14.
        i. The program command bits 0 0 1 0 0 x will be sent.
        ii. After a delay of 1 us, an unknown value x will be received, followed by the same 14 data bits sent earlier, and finally a 0.

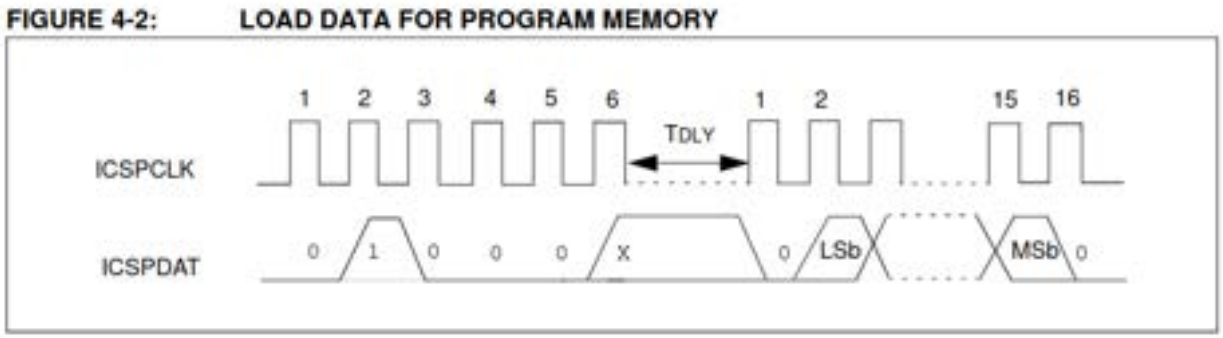


Fig. 13. Load Data example execution from programing datasheet [2]

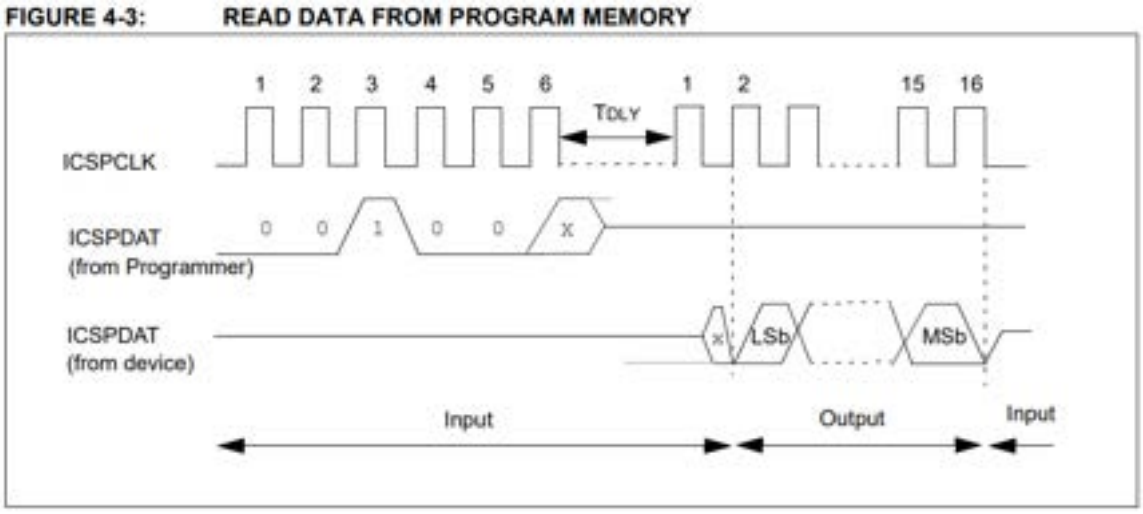


Fig. 14. Read Data example execution from programing datasheet [2]

4.6.6. References and File Links

[1] "Glasgow interface explorer," Crowd Supply. [Online]. Available: https://www.crowdsupply.com/1bitsquared/glasgow. [Accessed: 20-Jan-2023].

[2] "Pic12(L)F1612/16(L)F161X memory programming," *Microchip*, Sep-2013. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/40001720C.pdf. [Accessed: 20-Jan-2023].

[3] "PIC16(L)F1615/9 Data Sheet," *Microchip*, Oct-2014. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/40001770D.pdf. [Accessed: 20-Jan-2023].

4.6.7. Revision Table

Table 21: Revision Table for HDL Configuration

3/6/23	Darius Salagean: HDL Configuration put into project document. Final edits made based on review.
2/11/23	Darius Salagean: General Validation section was redone to include reasoning for selection of specific components, other options, and specifics of how the HLD code meets requirements based on feedback from Hunter, Benjamin, and Rachale.
2/11/23	Darius Salagean: Pseudocode was made and added to the design section based on feedback from Hunter and Benjamin.

2/11/23	Darius Salagean: Description updated to specifically reference the Glasgow Explorer based on feedback from Hunter Pitzler.
2/10/23	Darius Salagean: Updated Verification plan to match new interfaces and include more detail in description based on feedback from Rachael Cate.
2/10/23	Darius Salagean: Interface table and block diagram updated to include an input interface based on feedback from Benjamin Johnson
1/20/23	Darius Salagean: Initial document creation and sections filled out.

5. System Verification Evidence

5.1. Universal Constraints

5.1.1. The system may not include a breadboard

Our project is made up of code and specific hardware. There are no electrical connections that are made using a breadboard, because of this, the universal constraint is met.

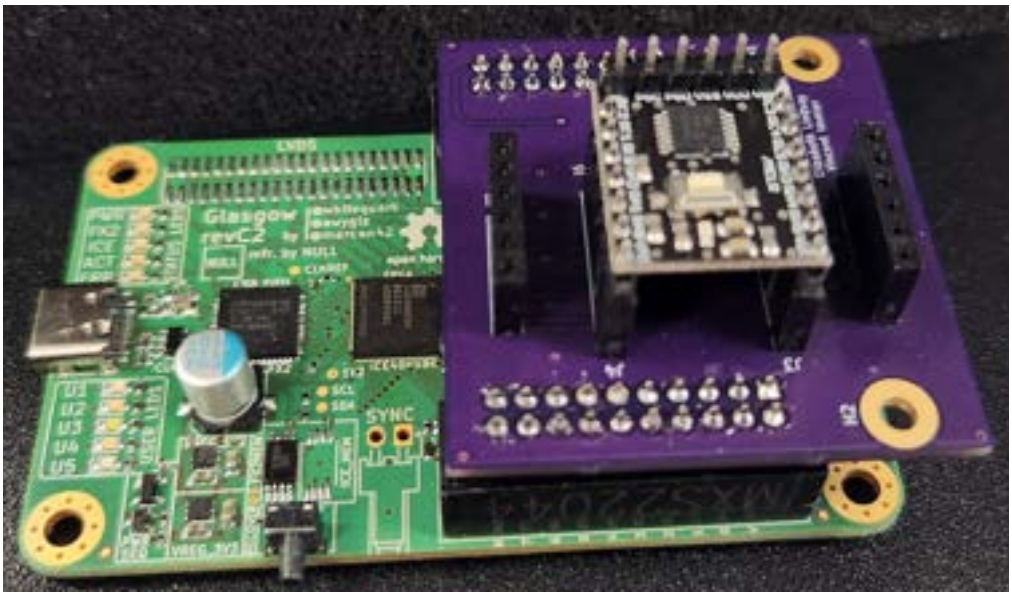


Fig. 15. Image of the Final System With all Connections

5.1.2. The final system must contain a student designed PCB.

Our system includes a PCB that acts like a static connection, because of this, we do not need surface mount pads as part of our design. For this reason, the constraint must be modified to remove the required 30

surface mount pads. With this modification in place, the universal constraint is met. Approval for modification to this constraint is seen in Fig. 17 at the end of this section.

5.1.3. All connections to PCBs must use connectors.

Our PCB uses either Jtag connections or female headers for connections. There are pin outs that are used for taking measurements and not part of the system. For these reasons, this universal constraint is met.

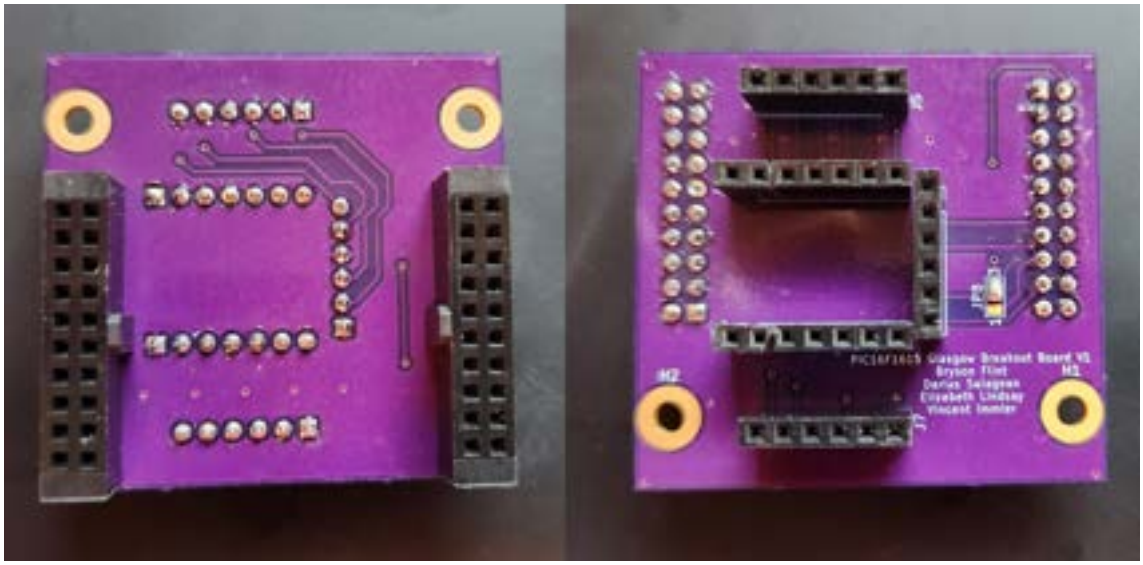


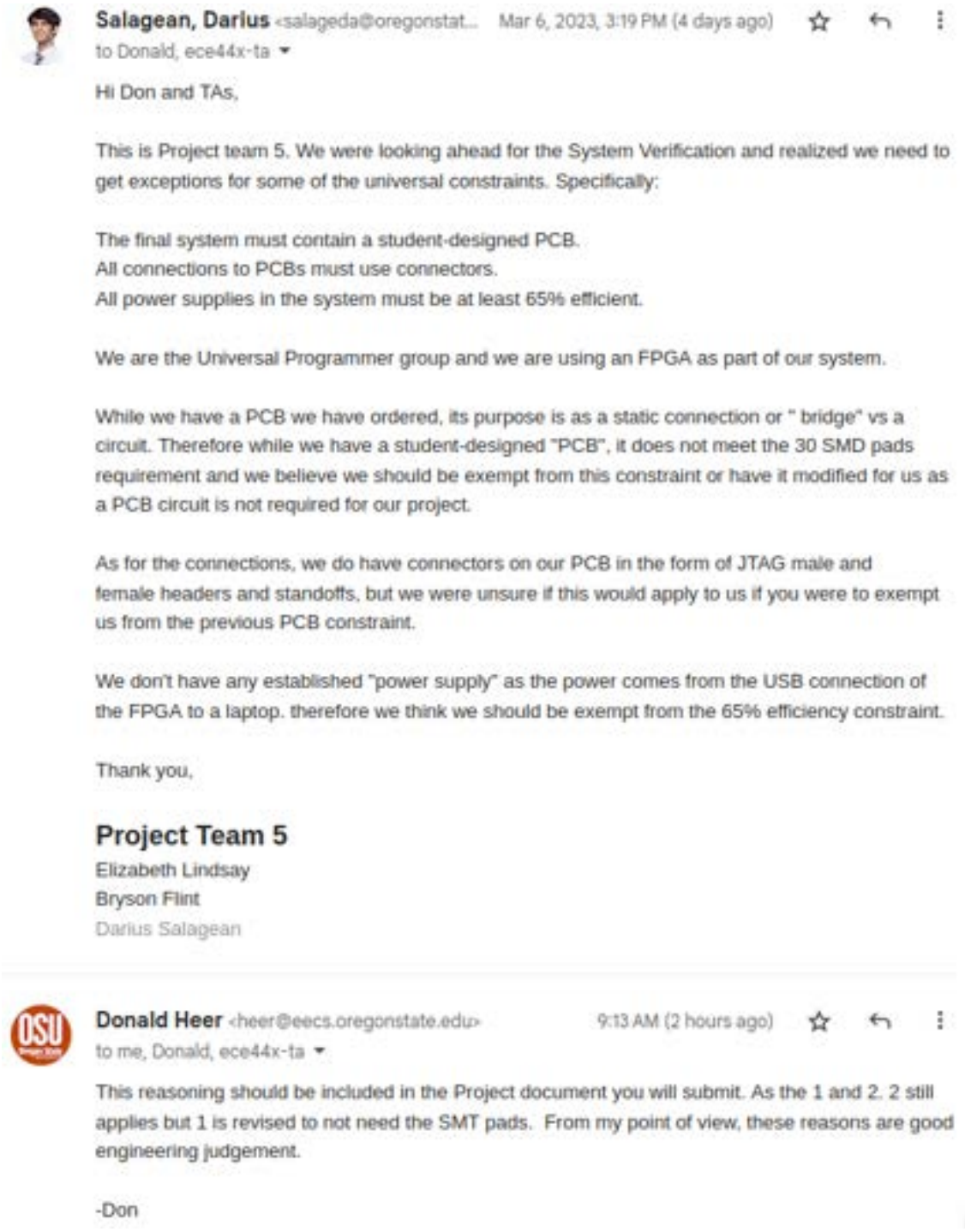
Fig. 16. Front and Back of PCB

5.1.4. All power supplies in the system must be at least 65% efficient.

The power supply in the project is the laptop the device is connecting to. There is no other external power source that was made by us for the project, and therefore there is no measurable efficiency. For these reasons, this constraint is not required. Approval for this is seen in Fig. 17 at the end of this section.

5.1.5. The system may be no more than 50% built from purchased modules.

The system does use a few pre-purchased modules, such as, the Glasgow Explorer, PIC16F1615 (Microcontroller), Data analyzer, and programmer. However, these modules are being connected into a larger system that uses our built modules which will overall have our system made up of built modules instead of purchased modules. For this reason, this constraint is met. The blocks that are purchased are the glasgow and the logic analyzer, while the built modules are the glasgow code, library, jupyter, and pcb. This means that only 33% of the modules are purchased.



Salagean, Darius <salageda@oregonstat... Mar 6, 2023, 3:19 PM (4 days ago)
to Donald, ece44x-ta

Hi Don and TAs,

This is Project team 5. We were looking ahead for the System Verification and realized we need to get exceptions for some of the universal constraints. Specifically:

The final system must contain a student-designed PCB.
All connections to PCBs must use connectors.
All power supplies in the system must be at least 65% efficient.

We are the Universal Programmer group and we are using an FPGA as part of our system.

While we have a PCB we have ordered, its purpose is as a static connection or " bridge" vs a circuit. Therefore while we have a student-designed "PCB", it does not meet the 30 SMD pads requirement and we believe we should be exempt from this constraint or have it modified for us as a PCB circuit is not required for our project.

As for the connections, we do have connectors on our PCB in the form of JTAG male and female headers and standoffs, but we were unsure if this would apply to us if you were to exempt us from the previous PCB constraint.

We don't have any established "power supply" as the power comes from the USB connection of the FPGA to a laptop. therefore we think we should be exempt from the 65% efficiency constraint.

Thank you,

Project Team 5
Elizabeth Lindsay
Bryson Flint
Darius Salagean

Donald Heer <heer@eecs.oregonstate.edu> 9:13 AM (2 hours ago)
to me, Donald, ece44x-ta

This reasoning should be included in the Project document you will submit. As the 1 and 2. 2 still applies but 1 is revised to not need the SMT pads. From my point of view, these reasons are good engineering judgement.

-Don

Fig. 17. Email from Donald Heer approving modification to PCB constraint and removal of Power Supply constraint.

5.2. Requirements

5.2.1. Save Data

5.2.1.1. **Project Partner Requirement:**

The universal programmer system will be able to record data.

5.2.1.2. **Engineering Requirement:**

The universal programmer system will save results in a way that is clear and accessible to 9 out of 10 users.

5.2.1.3. **Testing Method:**

Test

5.2.1.4. **Verification Process:**

A user will be shown the Jupyter code and given an explanation as to how to use the Jupyter Notebook interface. The User will then be asked to run the code that saves the data, find the file on their own, and look at the file contents We will then ask if the data they see is clear and if the file was accessible. A signature in the form of a Google Form will collect the user's response.

Pass Condition:

9 out of the 10 users who are tested certify that the data was saved in a way that is clear and accessible.

5.2.1.5. **Testing Evidence:**

Link to video demonstration Performed on 5/13/2023

	Timestamp	Please write your name.	Do you certify that the data you have seen is saved in a way that is clear and accessible to you?
2	5/13/2023 19:00:43	Amber Nickels	Yes
3	5/13/2023 19:03:29	Hana Steffen	Yes
4	5/13/2023 19:06:01	Nicholas Slugg	Yes
5	5/13/2023 19:09:41	Casper Ko	Yes
6	5/13/2023 19:13:39	Skylar Green	Yes
7	5/13/2023 19:18:18	Kaylee Chiang	No
8	5/13/2023 19:24:08	Lillie Case	Yes
9	5/13/2023 19:29:32	Jenny Sai	Yes
10	5/13/2023 19:33:50	Atlas Waugh	Yes
11	5/11/2023 12:18:02	Aden Hurdstrom	Yes
12	5/11/2023 12:18:50	Vanessa grunewald	Yes
13	5/11/2023 12:19:11	Sofia Gregson	Yes
14	5/11/2023 12:50:43	Brooke Aduviri	Yes
15	5/11/2023 12:51:14	Denis Cristurean	Yes
16	5/11/2023 12:51:24	Jack Hanline	Yes
17	5/11/2023 12:51:24	Ivy Parker	Yes
18	5/11/2023 12:51:33	Jon Kuyper	Yes
19	5/11/2023 14:05:09	William Castro Ramires	Yes
20	5/11/2023 14:22:23	Cole Ferguson	Yes
21	5/13/2023 18:02:57	Zoe Downen	Yes

Fig. 18. Responses From 20 Users. 19 out of 20 Say Yes.

5.2.2. Stable Interface

5.2.2.1. **Project Partner Requirement:**

The universal programmer will be able to connect to a user interface.

5.2.2.2. **Engineering Requirement:**

The universal programmer interface won't crash during 9 out 10 command executions.

5.2.2.3. **Testing Method:**

Test

5.2.2.4. **Verification Process:**

All cells of the Jupyter code will be run using a new kernel each time.

Pass Condition:

The Jupyter code executes without a printed error or kernel crash at least 9 out of the 10 runs

5.2.2.5. **Testing Evidence:**

Link to video demonstration Performed on 5/14/2023

5.2.3. Commands

5.2.3.1. **Project Partner Requirement:**

The universal programmer will be able to program a microcontroller

5.2.3.2. **Engineering Requirement:**

The system will produce 5 commands (Load Config, Load, Read, Incr, Reset) for the PIC16 family of uC.

5.2.3.3. **Testing Method:**

Demonstration

5.2.3.4. **Verification Process:**

The system will be connected to a wave analyzer device and begin recording. Each individual command will be executed and the recording will stop. The resulting wave forms will be compared to the wave forms from the PIC16F1615 datasheet.

Pass Condition:

The recorded wave forms match the wave forms found in the datasheet.

5.2.3.5. **Testing Evidence:**

Link to video demonstration Performed on 5/8/2023

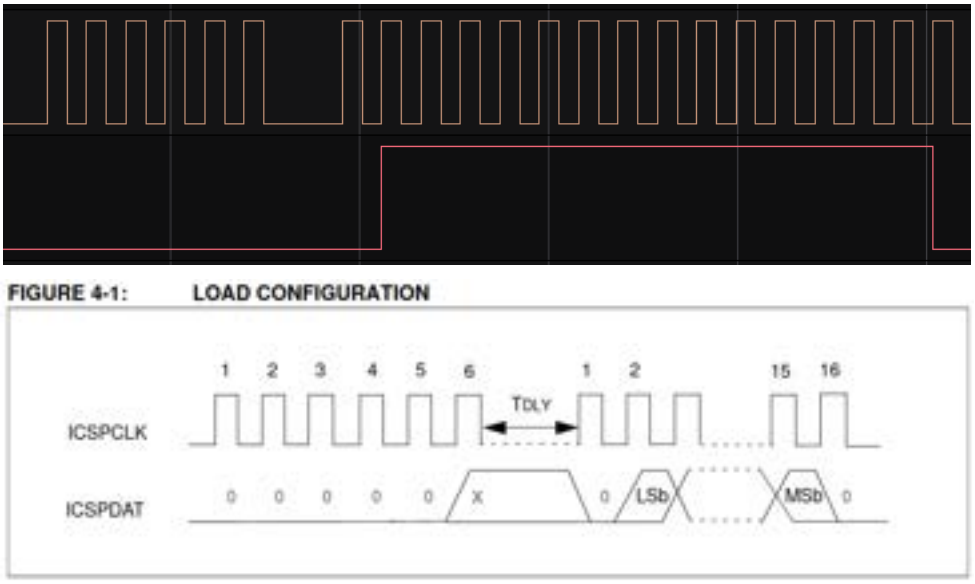


Fig. 19. Recorded Load Configuration Command and Data Sheet Specification

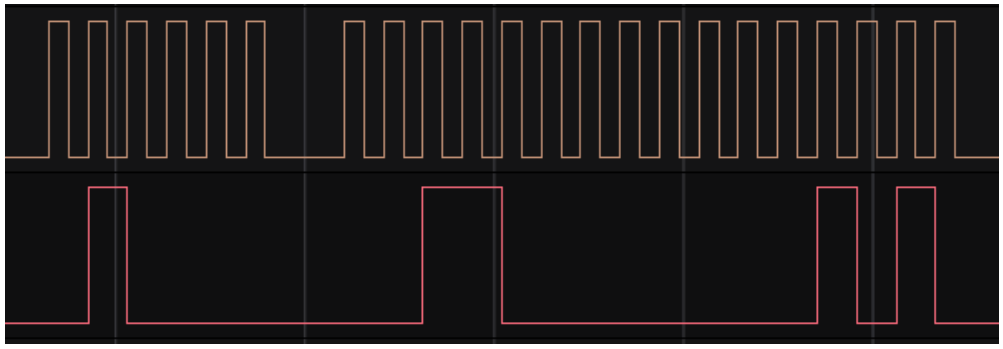


FIGURE 4-2: LOAD DATA FOR PROGRAM MEMORY

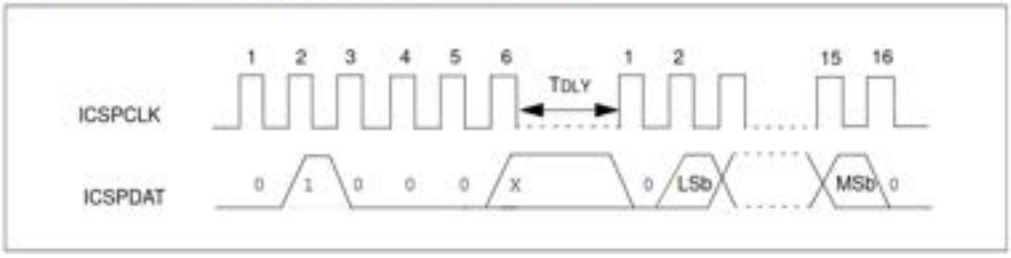


Fig. 20. Recorded Load Data Command and Data Sheet Specification

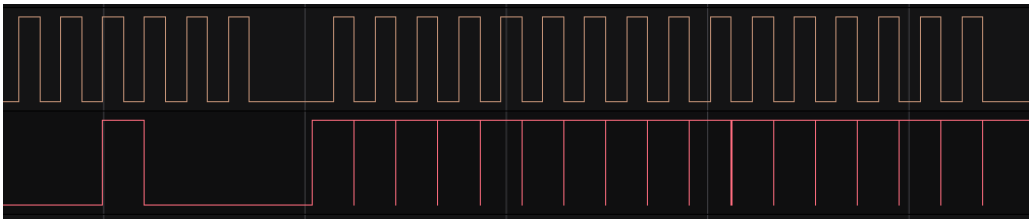


FIGURE 4-3: READ DATA FROM PROGRAM MEMORY

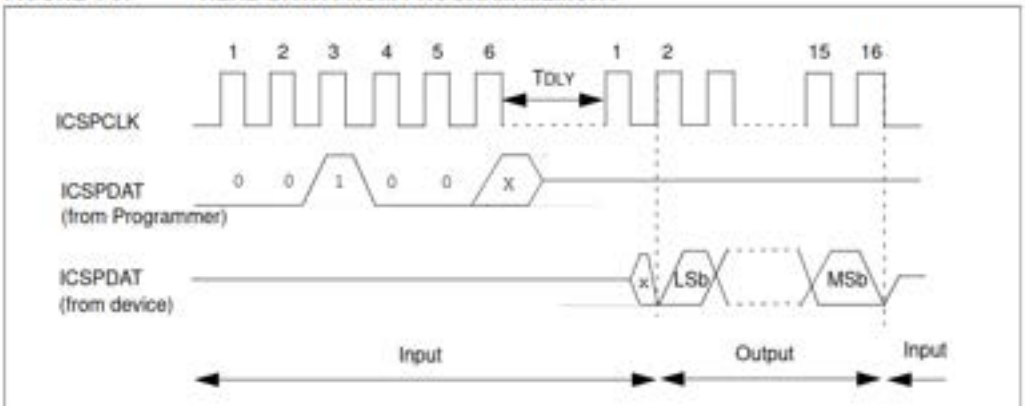


Fig. 21. Recorded Read Data Command and Data Sheet Specification

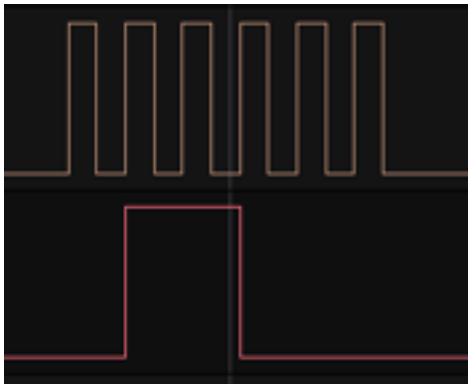


FIGURE 4-4: INCREMENT ADDRESS

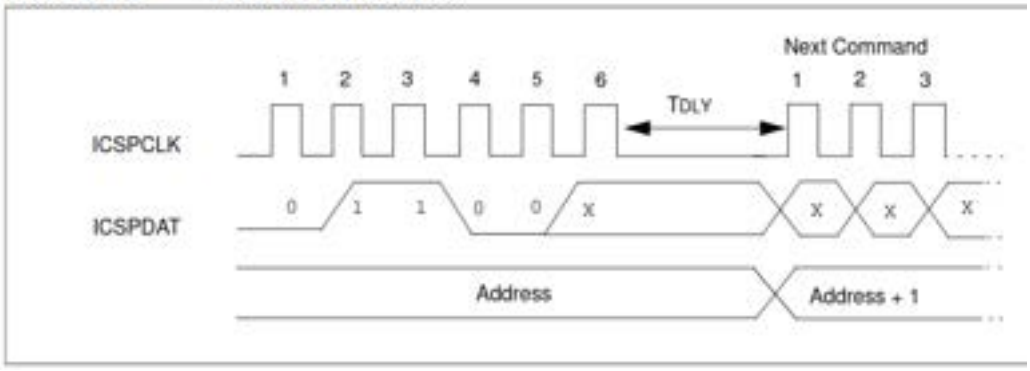


Fig. 22. Recorded Increment Address Command and Data Sheet Specification

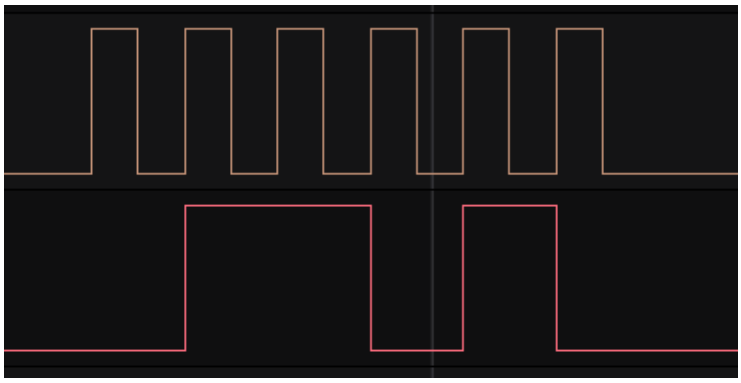


FIGURE 4-5: RESET ADDRESS

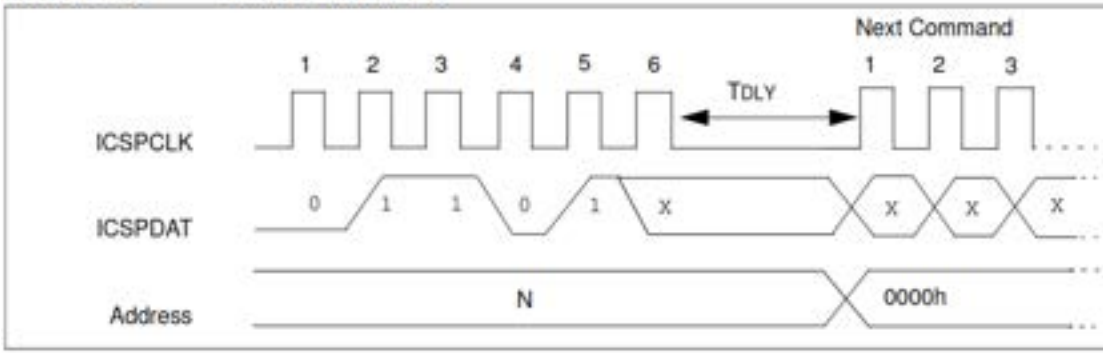


Fig. 23. Recorded Reset Address Command and Data Sheet Specification

5.2.4. Connection

5.2.4.1. **Project Partner Requirement:**

The universal programmer must be able to connect to the microcontroller it is programing

5.2.4.2. **Engineering Requirement:**

The system will connect to the target microcontroller securely enough that when the system is turned upside down, it will not disconnect.

5.2.4.3. **Testing Method:**

Test

5.2.4.4. **Verification Process:**

The microcontroller will be connected to the carrier board, which will then be connected to the universal programmer. After which, the whole connected system will be turned upside down and held for 30 seconds.

Pass Condition:

The system does not come apart after being held upside for 30 seconds.

5.2.4.5. **Testing Evidence:**

Link to video demonstration Performed on 5/8/2023

5.2.5. GUI

5.2.5.1. **Project Partner Requirement:**

The universal programmer will be able to connect to a user interface.

5.2.5.2. **Engineering Requirement:**

The system will have 9 out of 10 users execute programmer commands from a Jupyter notebook and report that the process was "clear".

5.2.5.3. **Testing Method:**

Test

5.2.5.4. **Verification Process:**

A user will be shown the code interface and explained how to use Jupyter Notebook. The User will then be asked to run each of the programming commands. We will then ask if the process of executing commands was clear. A signature in the form of a Google Form will collect the user's response.

Pass Condition:

9 out of 10 users certify that the process of executing commands was clear.

5.2.5.5. **Testing Evidence:**

Link to video demonstration Performed on 5/10/2023

1	Timestamp	Please write your name.	Do you certify that the process of executing commands was clear?
2	5/9/2023 20:37:58	Sofia Gregson	Yes
3	5/9/2023 20:41:37	Jon Kuyper	Yes
4	5/9/2023 20:47:32	Vanessa Grunewald	Yes
5	5/9/2023 20:50:48	Alexander Prestwich	Yes
6	5/9/2023 20:50:51	Ivy Parker	Yes
7	5/9/2023 21:03:09	William Castro Ramires	Yes
8	5/9/2023 21:04:08	Denis Cristurean	Yes
9	5/9/2023 21:04:51	Jack Hanline	Yes
10	5/9/2023 21:51:09	Gianna Negrete	Yes
11	5/9/2023 22:15:29	Gabriella Justen	Yes
12	5/10/2023 9:43:27	Stuart Allen	Yes

Fig. 24. Responses From 11 Users. 11 out of 11 Say Yes.

5.2.6. Reading

5.2.6.1. **Project Partner Requirement:**

The universal programmer will be able to read program memory from a microcontroller.

5.2.6.2. **Engineering Requirement:**

The universal programmer will read the program memory from a PIC uC without any errors.

5.2.6.3. **Testing Method:**

Demonstration

5.2.6.4. **Verification Process:**

The PIC microcontroller will be put into Program/Verify mode. Then, the Device ID will be read from the memory address 8006h. The received value will be compared to the expected value from the datasheet.

Pass Condition:

The data that is saved from the reading process matches the Device ID that is listed in the datasheet.

5.2.6.5. **Testing Evidence:**

Link to video demonstration Performed on 5/10/2023

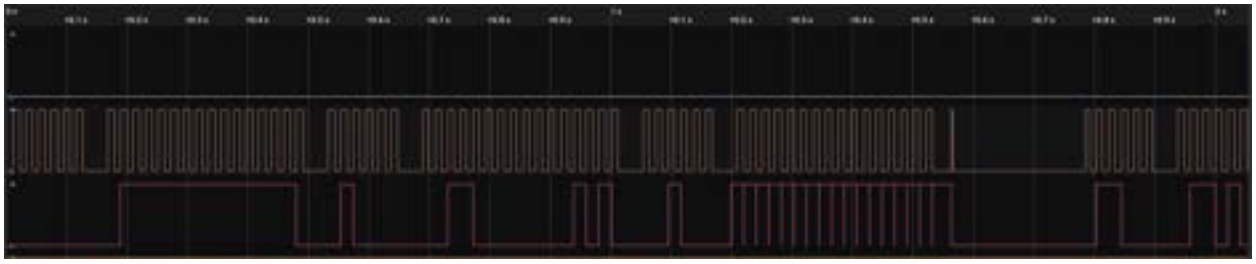


Fig. 25. Recording of all 5 commands back to back in 2.05 seconds

5.2.7. Speed

5.2.7.1. **Project Partner Requirement:**

The universal programmer needs to mimic the speed of the microcontroller's normal programmer.

5.2.7.2. **Engineering Requirement:**

The universal programmer will execute a command in under 10 seconds.

5.2.7.3. **Testing Method:**

Inspection

5.2.7.4. **Verification Process:**

The system will execute each of the 5 commands one at a time. A timer will record time passed from the start of execution until the end for each command.

Pass Condition:

None of the 5 commands exceed a time of 10 seconds.

5.2.7.5. **Testing Evidence:**

Link to video demonstration Performed on 5/10/2023

5.2.8. Visualize data

5.2.8.1. **Project Partner Requirement:**

The universal programmer will be able to connect to a user interface.

5.2.8.2. **Engineering Requirement:**

The data from the programmer will display data in a way that is clear to 9 out of 10 users

5.2.8.3. **Testing Method:**

Test

5.2.8.4. **Verification Process:**

A user will be shown the Jupyter code and given an explanation as to how to use the Jupyter Notebook interface. The User will then be asked to run the code that reads program memory and look at the displayed results. We will then ask if the data they see is clear. A signature in the form of a Google Form will collect the user's response.

Pass Condition:

9 out of 10 users will report that the data that is displayed is clear.

5.2.8.5. **Testing Evidence:**

Link to evidence Performed on 5/10/2023

1	Timestamp	Please write your name,	Do you certify that the data displayed to you is clear?
2	5/9/2023 20:38:10	Sofia Gregson	Yes
3	5/9/2023 20:41:52	Jon Kuyper	Yes
4	5/9/2023 20:45:53	Aden Hurdstrom	Yes
5	5/9/2023 20:48:03	Vanessa Grunewald	Yes
6	5/9/2023 20:51:08	Ivy Parker	Yes
7	5/9/2023 20:51:19	Alex Prestwich	Yes
8	5/9/2023 21:03:32	William Castro Ramires	Yes
9	5/9/2023 21:04:22	Denis Cristurean	Yes
10	5/9/2023 21:05:10	Jack Hanline	Yes
11	5/9/2023 21:51:19	Gianna Negrete	Yes
12	5/9/2023 22:15:53	Gabriella Justen	Yes
13	5/10/2023 9:51:41	Stuart Allen	Yes

Fig. 26. Responses From 12 Users. 12 out of 12 Say Yes.

5.3. References and File Links

5.4. Revision Table

Table 22: Revision Table Section 5

5/14/2023	Darius Salagean: Updated testing videos for Save Data and Stable Interface. Added photos of PCB and completed system to Section 5.1
5/10/2023	Darius Salagean: Evidence added for Speed, Reading, GUI, and Visualize Data
5/9/2023	Bryson Flint: Evidence added for Commands and connection
4/26/2023	Darius Salagean: Content for 5.2.3 through 5.2.8 added
3/14/2023	Darius Salagean: Added link for Data save and Stable Interface testing evidence
3/10/2023	Darius Salagean: Initial section creation. Initial content for 5.2.1 and part of 5.1 Bryson Flynt: Initial content for 5.2.2 and part of 5.1

6. Project Closing

6.1. Future recommendations

6.1.1. Technical recommendations

The earliest technical recommendation for the project is to learn the Glasgow Explorer early on in the year. The reason for this

recommendation is because it has a large learning curve and resources on it are limited as it is new. In order for our group to be able to complete this project we had to receive 3 training sessions from someone in Germany over the course of 6 weeks. Having these done as early as possible will ensure there is no rush, as our group only completed these at the end of Spring break. This reference [1] will help learning the Glasgow Explorer. The next recommendation is to follow this [2] reference PCB that was provided by the project partner to create the carrier PCB for the system. This is an example that shows the Glasgow being connected to a different device and can work as a good early point for designing a PCB with a chosen chip. It is also helpful to study the original example as it has many good PCB rules, such as avoiding traces with right angles and using test pads. Another technical recommendation is to make sure some group members, if not all group members, understand python. Python will be demanded of you when using Jupyter notebook and using the language Amaranth on the Glasgow Explorer. Having solid fundamentals in the language will help when learning how to use these pieces of software and will reduce the time needed to debug code. This [3] is a link to learning more about python. The last technical recommendation, thoroughly read the data sheet of the chosen microcontroller for the project. Several instances of reading the data sheet for our chosen microcontroller made sense of the data being viewed. There are also important flow charts that go through a typical programming cycle. Use this [4] guide to help understand reading data sheets, they can be overwhelming at first.

6.1.2. Global Impact recommendations

One global recommendation is to choose a chip that is commonly used, so that it would be able to help advance chip security globally. A chip that is used in common household appliances could be a good target, as they would be used everyday. This could then further help chip security and maintain chip security in homes, so that data cannot be gathered without people's knowledge, and so that the chip firmware can be secure [5]. Another global recommendation is to choose a chip to create a programmer for, where the programmer can be expanded on so that it can possibly program an entire family. This relates to the chip shortage and creating a programmer that is more “universal” can have a positive impact [6].

6.1.3. Teamwork recommendations

One teamwork recommendation that really worked for this team is to have weekly meetings outside of our class time, and to specifically meet at a

time that is spaced out from class time. Our group met once every week sometime Monday through Friday, which kept everyone in the group up-to-date on progress. The space between lectures and our meeting also meant there was time for each group member to work on their assigned task and have a check-in that was timely. Another recommendation is to get to know your teammates outside of technical meetings. This can mean just grabbing dinner and drinks one night, or to get coffee, so that you can bond and find things in common outside of just the project [7]. Our group had dinner during the Fall term, and went to an event together during Winter term. These moments helped us see each other more as friends than just co-workers / project members and so we were much more involved in the success of each other and the project as a whole.

6.2. Project Artifact Summary with Links

6.2.1. User Sign off form for Data Save

https://forms.gle/UA5CuYEtQVVxo1Zt7

6.2.2. User Sign off form for GUI

https://forms.gle/V9sCBxKYu4B5Sbo3A

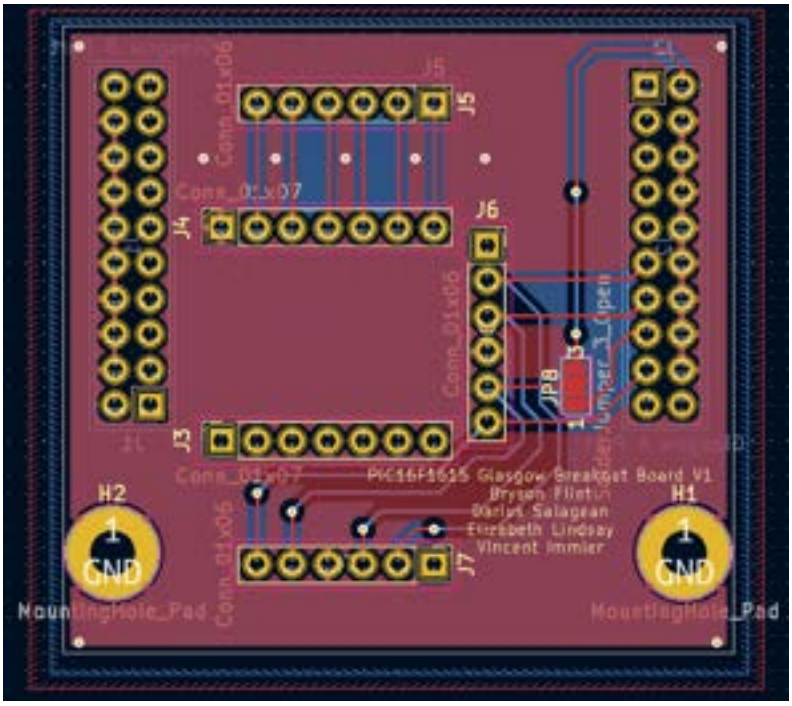
6.2.3. User Sign off form for Visualize Data

https://forms.gle/8wTavfPQ8uwsJBSdA

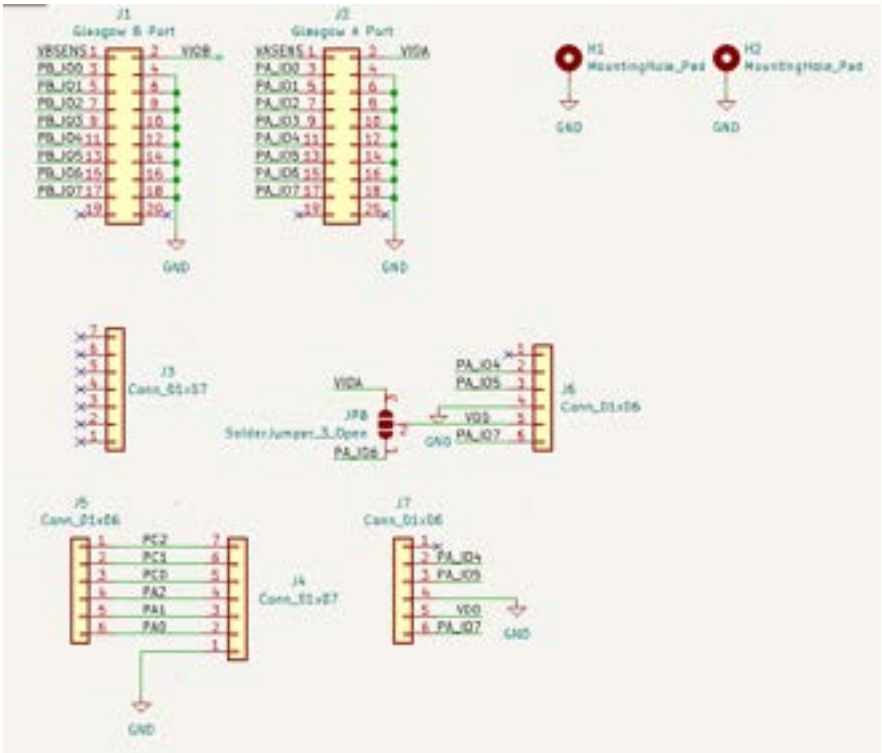
6.2.4. PCB KiCad project zip file link

https://drive.google.com/file/d/14F4AQEj3Jojlgt3XZFVn7VOksHo_LSR0/view?usp=sharing

6.2.5. PCB layout



6.2.6. PCB Schematic



6.2.7. Excel spreadsheet of gathered data
https://docs.google.com/spreadsheets/d/1QUk4qd4LdQEFy9CZT_XsMNiZtIICVjEa/edit#gid=1941618922

6.2.8. PIC applet code for initializing Glasgow Interface commands and building applet
https://drive.google.com/file/d/1VIsvY6NOQ0NZN6kRUcl4zWsaYfYz6U8q/view?usp=sharing

6.2.9. PIC gateware code for hardware implementation
https://drive.google.com/file/d/1kuMYDKdll50mMCCNOeua7WAra97CO0Zr/view?usp=sharing

6.2.10. Jupyter code for GUI
https://drive.google.com/file/d/1SeYfbElcuBu8lF2GmlrTOcoUy8G821g7/view?usp=sharing

6.2.11. Jupyter Library code for functions
https://drive.google.com/file/d/1UZcqrgEcF-IdP0poFvzYnwObY_2fpL9G/view?usp=sharing

6.3. Presentation Materials

Showcase Link:
https://eecs.engineering.oregonstate.edu/project-showcase/projects/?id=XsGF0agoaspWCPlm

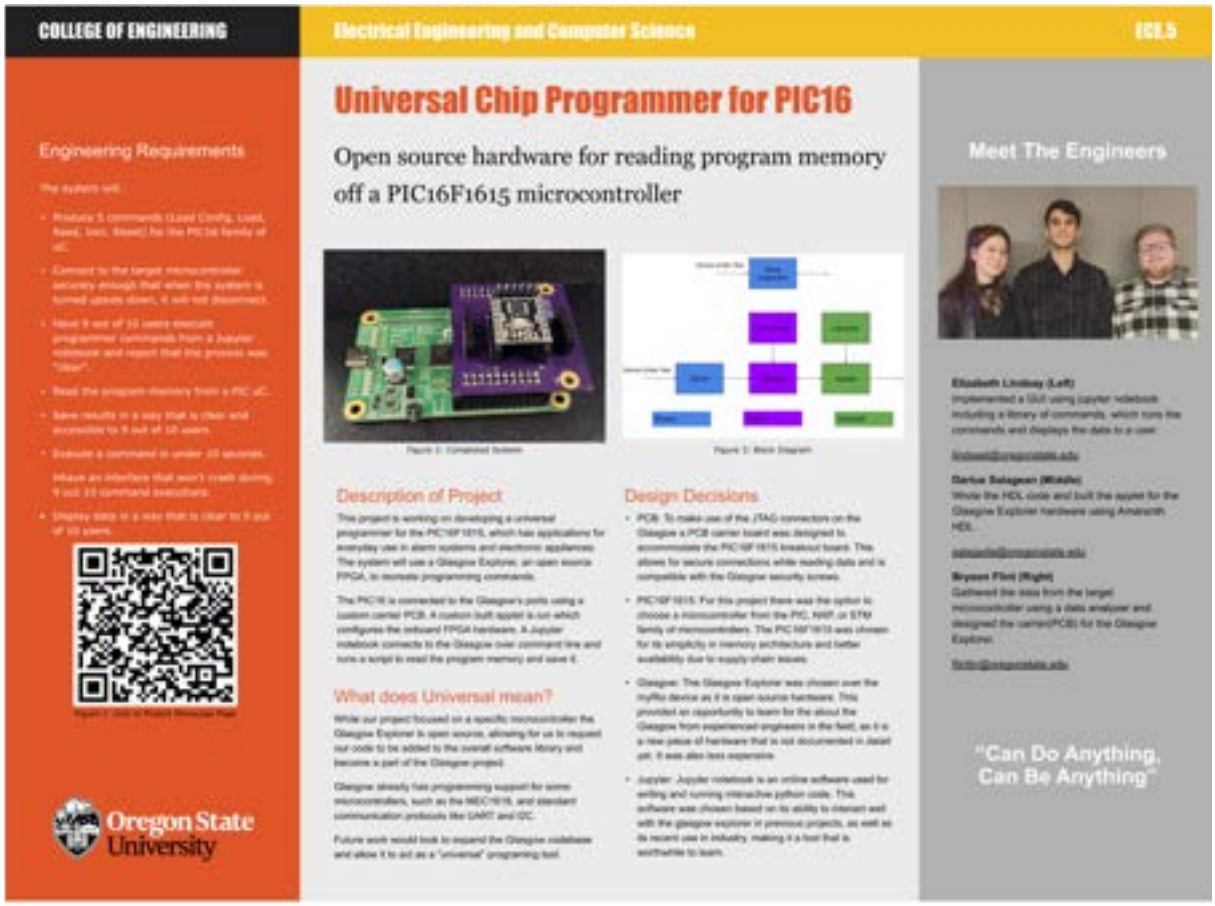


Fig. 27. Expo Poster

6.4. References and File Links

[1] GlasgowEmbedded, “Glasgow Debug Tool,” *GitHub*. [Online]. Available: https://github.com/GlasgowEmbedded/glasgow. [Accessed: 26-Apr-2023].

[2] https://drive.google.com/file/d/1pHvo2i1HZeHUw2EdkK22PL7KFjMrrKxw/view?usp=sharing

[3] “Python for beginners,” *Python.org*. [Online]. Available: https://www.python.org/about/gettingstarted/. [Accessed: 26-Apr-2023].

[4] M. Grusin, “How to Read a Datasheet,” *SparkFun Electronics*, 17-Nov-2010. [Online]. Available: https://www.sparkfun.com/tutorials/223. [Accessed: 26-Apr-2023].

[5] M.Schink and J.Obermaier, “Exception(al) Failure - Breaking the STM32F1 Read-out Protection,” 17 March 2020, Accessed: 11 October 2022. [Online]. Available: https://blog.zapb.de/stm32f1-exceptional-failure/

[6] P. hanbury, A. Hoecker and M. Schallehn, “A chip shortage recovery guide,” *Bain.com*, 25 March 2022, Accessed: 04 Nov 2022. [Online]. Available: https://www.bain.com/insights/a-chip-shortage-recovery-guide/#:~:text=The%20automation%2 and%20 industrial%20sectors,2023%20(see%20 Figure%201).

[7] R. Baker, “Team bonding: Why it's important and how you can encourage it,” *nTask*, 13-Jan-2023. [Online]. Available: https://www.ntaskmanager.com/blog/team-bonding/#:~:text=When%20team%20members%20are%20bonded,each%20other%20through%20difficult%20challenges. [Accessed: 26-Apr-2023].

6.5. Revision Table

Table 23: Revision Table Section 6

5/14/2023	Darius Salagean: Updated sections 6.1.1 and 6.1.3 based on draft feedback.
5/10/2023	Darius Salagean and Bryson Flint: Added more artifacts and updated placeholders with full links.
4/26/2023	Sections created by Bryson Flint. Artifact section filled out by Darius Salagean. 6.1.1 filled out by Bryson Flint. 6.1.2 and 6.1.3 filled out by Elizabeth Lindsay.

A. Appendix

Link to Glasgow homepage: https://www.crowdsupply.com/1bitsquared/glasgow

PIC16 Programming datasheet:
https://ww1.microchip.com/downloads/en/DeviceDoc/40001720C.pdf

PIC16 Electrical datasheet:
https://ww1.microchip.com/downloads/en/DeviceDoc/40001770D.pdf