ECE/RS-493
FINAL REPORT
GMU Logic Analyzer

Abstract

The GMU logic analyzer project is intended to provide students at GMU with access to a logic analyzer without incurring some further cost to either the student or ECE department. The project accomplishes this goal by only requiring a basys3 FPGA, computer, and the cables necessary to be used as probes for the device. All of these are items that the target user is already required to obtain. The GMU logic analyzer is a 16-channel digital logic analyzer. It takes input voltages of 3.3V. The primary target device is the msp430 fr6989 launchpad, which has a maximum clock speed of 16 MHz. The logic analyzer is able to sample at ranges between 32 MHz and 12 kHz to allow it to effectively capture the outputs of the target device. The device can trigger off of both serial and parallel inputs, allowing it to capture SPI and I2C communication protocols. The device functions similarly to commercial logic analyzers and outputs comparable results.

Date of Submission:

Team Members:
Spencer Lombardo
Robert Minter
Jacob Samuels
Jordan Whitesell

Faculty Supervisor: Jens-Peter Kaps

Executive Summary

Logic analyzers are electronic instruments that are used to capture and display signals from a digital circuit/system. Logic analyzers range from simple devices which have to be attached to a PC and cost around $200 having 8 channels and 100 MS/s sampling rate to more complex industrial models costing more than $10,000 and having 32 channels and 500 MS/s sampling rate. These devices offer great utility for students in Electrical and Computer Engineering programs. However, even the cost of the cheapest models of logical analyzers are prohibitively expensive for an engineering student.

On this project, we will fulfill the task of designing, building and implementing a 16 channel logic analyzer. This logic analyzer will detect, record, and store signals from inputs connected to circuits. This logic analyzer will be paired with a front end to display signals graphically, allowing for easy analysis of the circuit and its functionality. For this device, we focused on utilizing the Basys3 Board, an FPGA trainer board that each George Mason student must purchase for class.

In our design process, we developed a smart triggering system, that prompts the system to begin reading and sending data to our front end. We developed a python based web front end that took the data received by the Logic analyzer and output it in a format that is easy to look at and interpret.

Contents

Approach

George Mason students in the Electrical and Computer Engineering (ECE) major take courses that further knowledge and ability to design, prototype and produce engineered solutions to everyday problems. During their classes, they take multiple technical labs that focus on giving the students hands on experience with the tools of the trade, such as digital and analog circuits and microcontrollers. The use of microcontrollers also includes using extra peripherals that provide increased functionality to the system that is being tested. Where the students lack is access to hardware that assists them in the debugging of the circuits and systems they design for the labs. One such tool is a logic analyzer. This device is used to connect plot over time the activity that happens at points of digital circuits. It records the logical highs and lows of a given point of a circuit and plots it for the user to see. These tools provide students with the ability to verify the functionality of a circuit at individual points and identify where a problem might be occurring in the system. ECE students lack access to logic analyzers that would aid them in debugging issues within some of their technical labs. ECE students can purchase one on their own, but commercial solutions tend to be either extremely costly, or not have the feature set that is required for their testing.

The GMU Logic analyzer is one solution to this problem. The logic analyzer is a “free” alternative to students purchasing their own logic analyzer. It utilizes the students purchased FPGA trainer board, the Basys3 board, to build a device capable of recording digital data. Since the students purchase the Basys3 board for other classes. The cost of it is not included in the price for the solution. The device can connect to host pc, where the data is displayed to the user on a browser based front end. The front end was developed in Python and JavaScript, utilizing

HighCharts display for easy chart compilation from the data collected.

Other solutions to the problem included utilizing the students purchased MSP430 training board, a microcontroller-based trainer board used to aid in the teaching of upper level courses. This solution was not feasible though, as some of the primary targets for the logic analyzer is in the labs featuring the MPS430.

Each member provided support in the development of the logic analyzer. In summary, Spencer Lombardo acted as the project manager, and focused on the integration and testing of the system. He worked to take the individual pieces that other members contributed and integrate them together. Robert Minter focused on the front-end of the project, where he developed the system to capture data from the connected device and display it to an easy to understand front end. Jacob Samuels focused on building the sampling system for the project. The system is built on the Basys3 boards FPGA, and takes in digital data, and samples it using a reconfigurable sampling clock. He also developed a component on the FPGA to status data to a built in seven segment display on the Basys3 Board. Jordan Whitesell focused on the triggering unit of the device. The triggering unit is a part of the FPGA, that takes in the sampled data, and determines if the system should begin recording or not. This unit lets the user configure when and what the data should look like before its recorded.

Technical Section

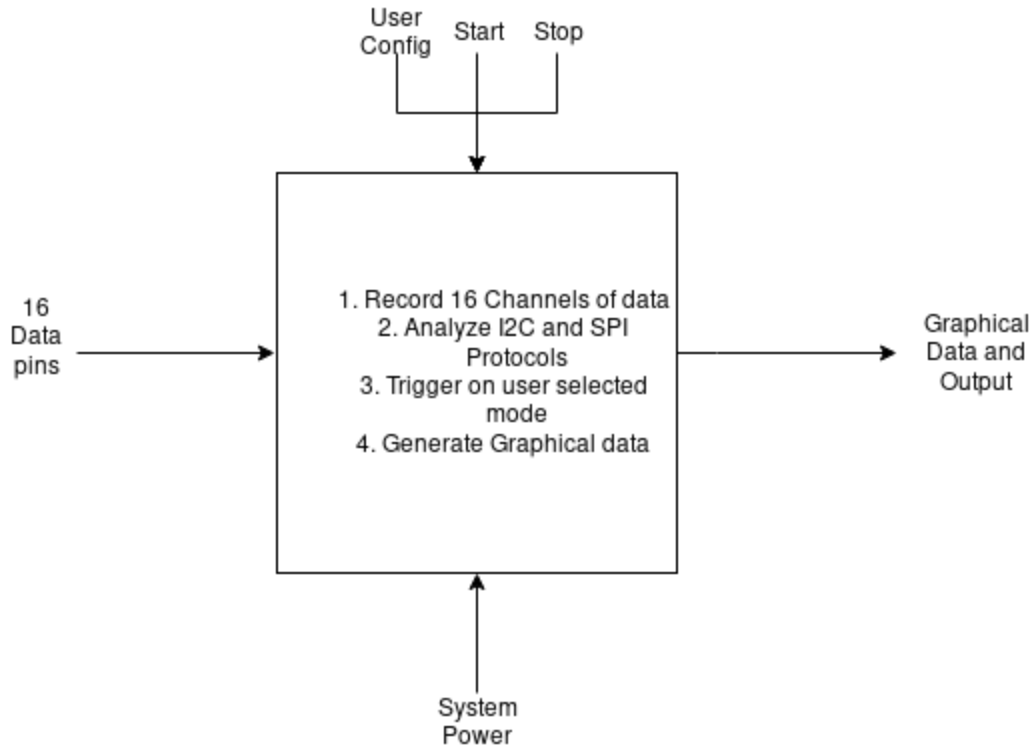
Requirements:

The design started with the following requirements:

- The logic analyzer shall record digital signals and output them to the user.
- The logic analyzer shall be built on the Basys3 FPGA board
- The logic analyzer shall contain 16 channel inputs using onboard PMOD headers
- The logic analyzer shall connect ot a local PC using USB2
- The logic analyzer shall decode SPI and I2C communication protocols.
- The logic analyzer shall contain smart triggers that trigger on
    - One channel Serial Input
    - Up to 16 channel Parallel input
    - Start button on front end GUI
- The logic analyzer shall record a few samples before the trigger happens
- The logic analyzer shall include an optional breakout board that allows for easy connection to the MSP430

Level 0 Decomposition

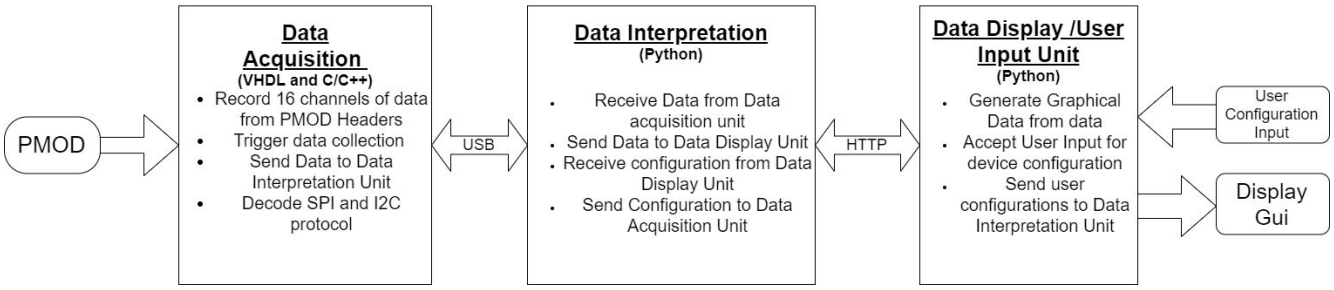
Following these requirements, a simple system decomposition can be derived:



The design started with a simple system overview. From this decomposition we can see the basic functionality of the system. Into the system is 16 data pins. The system triggers off of those pins, records the data, generates graphical output and outputs it to the user. The system requires power, and can take in user configurations, as well stopping and starting.

Level 1 Decomposition and Architecture

From this decomposition, the system is then broken down into three separate units. The three units are as follows:



Reading the figure from left to write, we see the same design. The Basys3 Board contains 4 GPIO pins, labelled PMOD. These PMOD pins are swapped in for the 16 data pins, and is where the data is grabbed from.

This data is provided to the Data Acquisition unit. This unit represents the Basys3 board section of the solution. This unit is expected to record the 16 channels of data grabbed by the PMOD headers. The unit triggers the data collection, and sends the data to the data interpretation unit VIA USB. It also holds the functionality of decoding SPI and I2C communication protocols. this unit is designed and implemented using VHDL, and flashes onto the Basys3 board for operation.

The Data acquisition unit records and sends the PMOD data to the Data Interpretation unit VIA USB. The Data Interpretation unit takes the data from the Data Acquisition unit, and formats it in a way that the Data Display unit can understand. It also received configuration instructions from the Data Display Unit, formats them in a way for the Data Acquisition unit, and

sends them off to configure the device. It serves as one part of the pc-side of the solution. It's implemented using python, and contains different functions to receive data, write data to a file for the front end to display, and send configurations to the Data Acquisition unit.

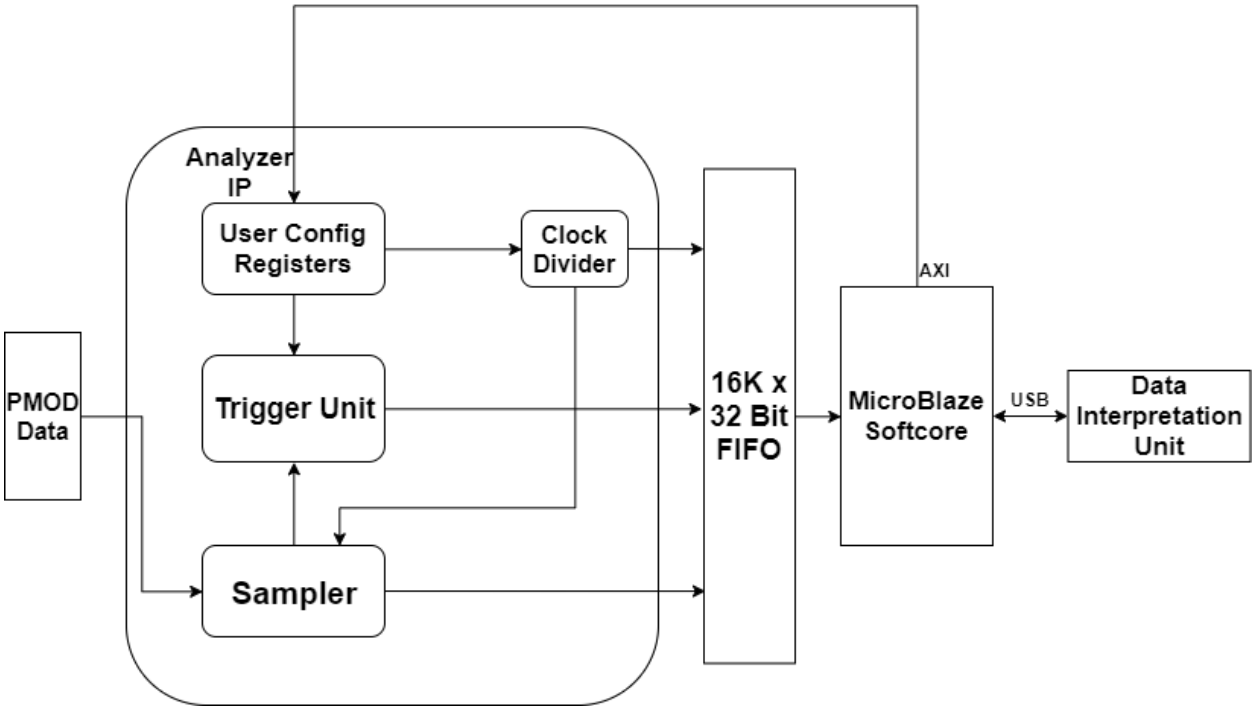
The Data display unit is the final piece of the solution, which consists of a browser based display, that grabs data formatted by the Data Interpretation unit and displays it in a human readable format. It allows for users to view the data, manipulate the chart displayed as needed to interpret that data, and provide a platform to export the data for more data manipulation. It also provides the user a graphical user interface to configure the Data Acquisition unit.

Diving deeper into the design, we look at the design of each individual unit.

Data Acquisition Unit:

The Data acquisition unit is built on the FPGA residing on the Basys3 Board. It is contained on the Xilinx Artix 7 FPGA, and utilizes the boards built in USB controller and PMOD headers to complete the functionality it contains. It serves the function as system triggering unit and data recorder.

Overall System design:



This unit contains multiple different subunits that focus on different functions. We first have the IP that was provided by Xilinx for free use. That includes the Fifo, which acts as a buffer between the data ready to be sent and the Microblaze softcore processor. The microblaze softcore processor is a synthesized processor that contains the logic to determine if the data is ready to be sent and provide an easy to use USB communication protocol. Next comes the

Analyzer IP that we created for the project.. It contains the User configuration registers, which hold the registers that hold the different trigger and clock configurations. Next is the sampling unit, which takes in the raw PMOD data, and samples it based off the configurable sampling clock. This data is provided to the FIFO that acts as a buffer for the data. We also designed a simple clock divider. This was due to the lack of low clock rate values the Xilinx Clock Wizard could provide. the clock divider provides the sampling clock for which the sampler uses to hold the data. Last comes the triggering unit. This unit functions to signal to the onboard processor that data is ready and valid, so that it can be transmitted to the front end. The following sections will describe in better detail their designs:

User Configuration Registers:

The user configuration registers are AXI-Lite defined and operated registers. The choice to make them AXI-lite based was to ensure an easy communication protocol between the Microblaze softcore processor and the user registers. The registers are 32 bits wide, and contain the user configarations for things like the desired clock and The Registers are defined as follows:

Register Number	Function
Register 0	Trigger Register. The first bit is used to hold whether or not the system has been triggered and can be read from
Register 1	Holds the 16 bit trigger pattern
Register 2	Holds the channels in which the parallel trigger operates off of
Register 3	Contains a multitude of configuration options for the serial trigger unit. the register contents are described after this table
Register 4	Contains the bit which enables device operation or not. setting this bit to zero ensures no data reading or triggering
Register 5	Contains the selector bits for the clock divider. the selection is described below this chart

Clock Divider Selector Options:

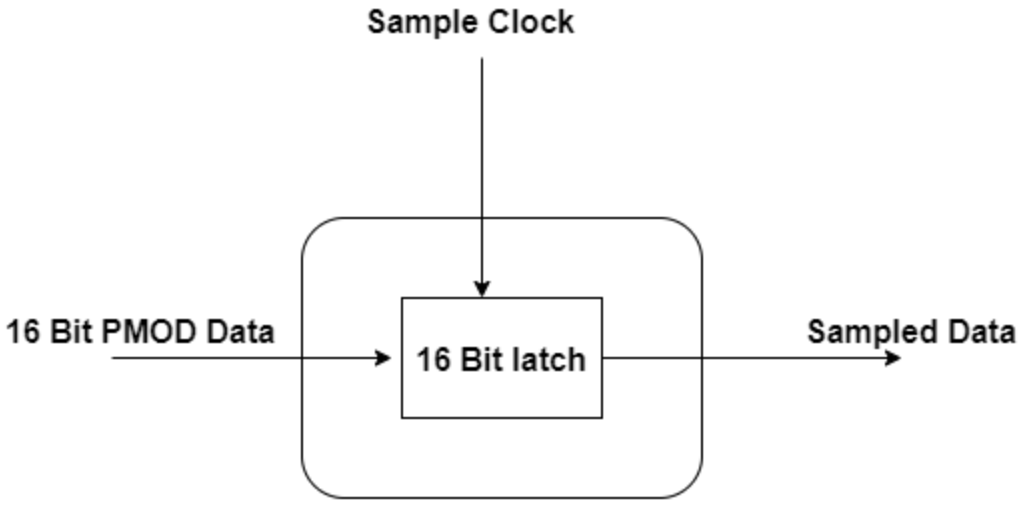
Bit pattern	**Clock Speed**
0000	12.2KHz
0001	24.4KHz
0010	48.8KHz
0011	97.7KHz
0100	195KHz
0101	390KHz
0110	781KHz
0111	1.56MHz
1000	3.125MHz
1001	6.25MHz
1010	12.5MHz
1011	25MHz
1100	50MHz

Register 3 Bit Definition

Bits	**Definition**
31:11	Reserved
10	Serial Reset bit. Resets the serial trigger when set to 1
9	Mode select. This selects between parallel triggering mode and serial triggering mode
8:4	Clock select. This selects the channel which contains the external clock required for serial triggering
3:0	Channel Select. Selects the channel to check for the serial trigger off of.

Clock Divider

The Clock divider is the unit that provides the sampling clock for the rest of the system. As defined above, the range of frequencies that can be used for the sampling clock is between 1.6Mhz - 12.5KHz. This provides a wide range of sampling frequencies to detect most signals that could need to be detected. it also allows for a theoretical maximum data rate of 16MSPS The unit is created using a simple 27 Bit counter, and using the select bits to multiplex out specific bits to set the clock rate.



Sampler

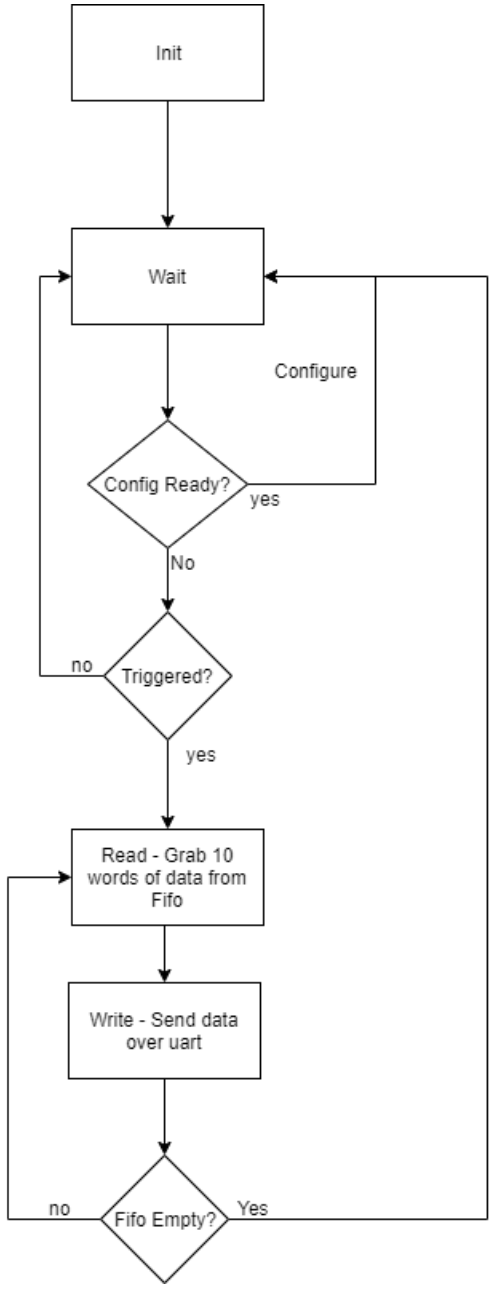
The sampling unit is the component that takes in the raw PMOD data, and samples it based of the sampling clock. This is done using a simple 16 bit latch, that grabs data on the rising edge of the sampling clock, and holding unit the next rising edge. this data is then sent out from the sampler to the rest of the system.

Trigger Unit:

The trigger unit looks for a set pattern coming from the sampled pmod data. When this pattern is found by the trigger unit, it will send a trigger signal to the FIFO. This causes the FIFO to store the sampled data, then send it to the PC via USB. The trigger unit can be broken down into three main components: the parallel trigger unit, the serial trigger unit, and the trigger hold unit. The parallel and serial trigger units operate similarly, but differ in how they check for their patterns. The parallel trigger unit looks for a pattern across a configurable set of channels across the entire range of inputs pins, and will output a trigger signal when this pattern is sent to the system. The serial trigger unit uses the same configuration registers for its pattern, and uses additional

configuration registers to set which channels receive data data and the external clock. It stores the pmod data in a shift register that is operated by the previously mentioned external clock. When the bits in the shift register match the pattern, the serial trigger unit will output a trigger signal. The trigger hold keeps the trigger signal high for a configurable duration to keep the fifo operational for the desired amount of time.

MicroBlaze use and algorithm



Data Interpretation Module

Overall PC Data Flow Chart

This Module is a python server that serves as a middle point for the data that flows between the GUI and the Basys3 Board

It functions as such: Server:

When the server receives a request it determines the endpoint then executes the function associated with that endpoint. When the endpoint is determined to be one of the config endpoints the method is determined. If the method is GET, then the route is parsed for the desired config id, which is queried from the database and loaded, serialized, and returned via the API response. However, if the requested config does not exist the standard 404 NOT FOUND response is returned via the API response. If the method is DELETE the route is parsed for the desired config id which is queried from the database and deleted returning a 204 NO CONTENT via the API response whether the config existed or not. If the method is POST request body is parsed for the desired config id which is queried from the database. If there is no entry one is created using the data from the request body returning 201 CREATED via the API response. I f there is an entry the API returns a 400 BAD REQUEST via the API response. If the method is PUT the route is parsed for the desired config id which is queried from the database and loaded. If the entry exists the entry is updated using the data from the request body returning 204 NO CONTENT via the API response. If the entry does not exist the API returns a 400 BAD

REQUEST via the API response. When the endpoint is determined to be the data endpoint the serve queries the data storage for all the data and then processes the data according to the config currently marked active. If there is not data the API returns empty data in the shape requested by active config. If the method requested is anything other than GET the 405 METHOD NOT ACCEPTED is returned via the API response.

PC:

The data packing and serial communication protocols are detached from the server and run as separate asynchronous processes. There are two of such processes one for communication of raw data from the USB channel. Another for sending the user config when a config change is detected.

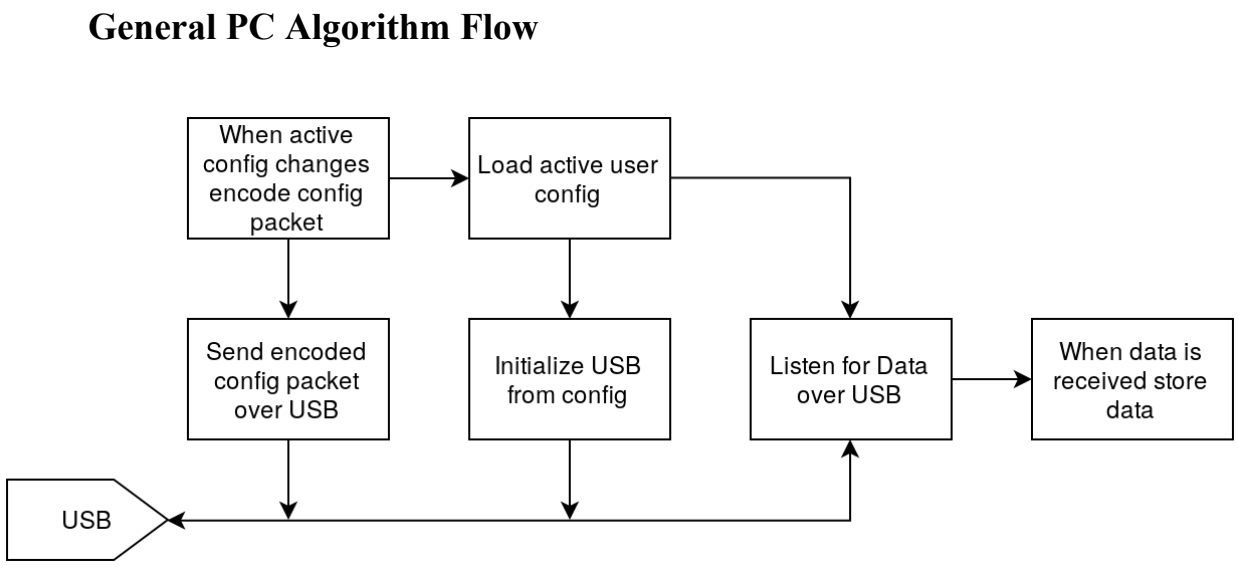


Figure 14: USB Controller Algorithm

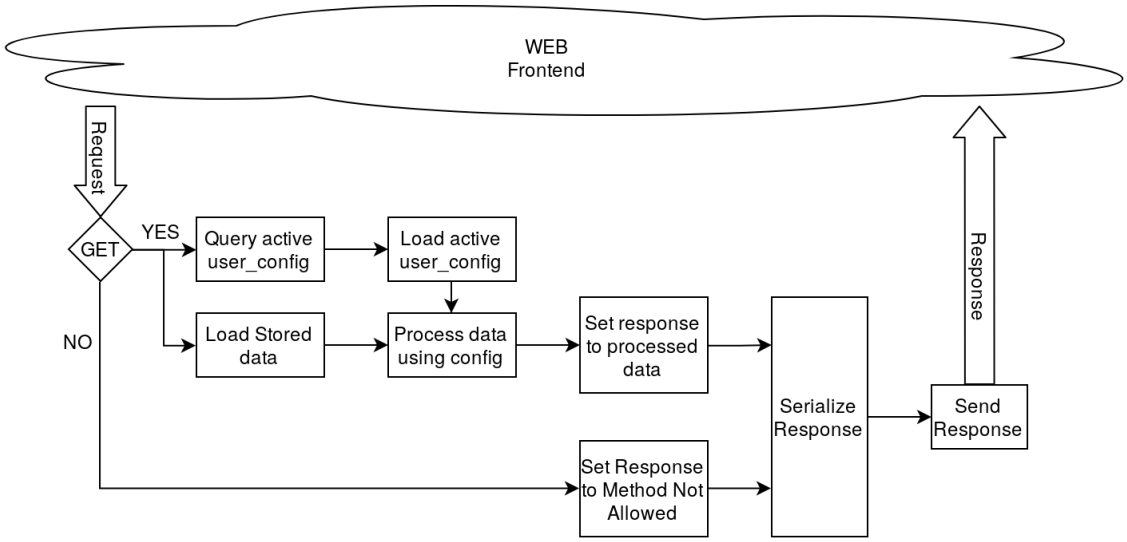


Figure 15: Data Endpoint Algorithm

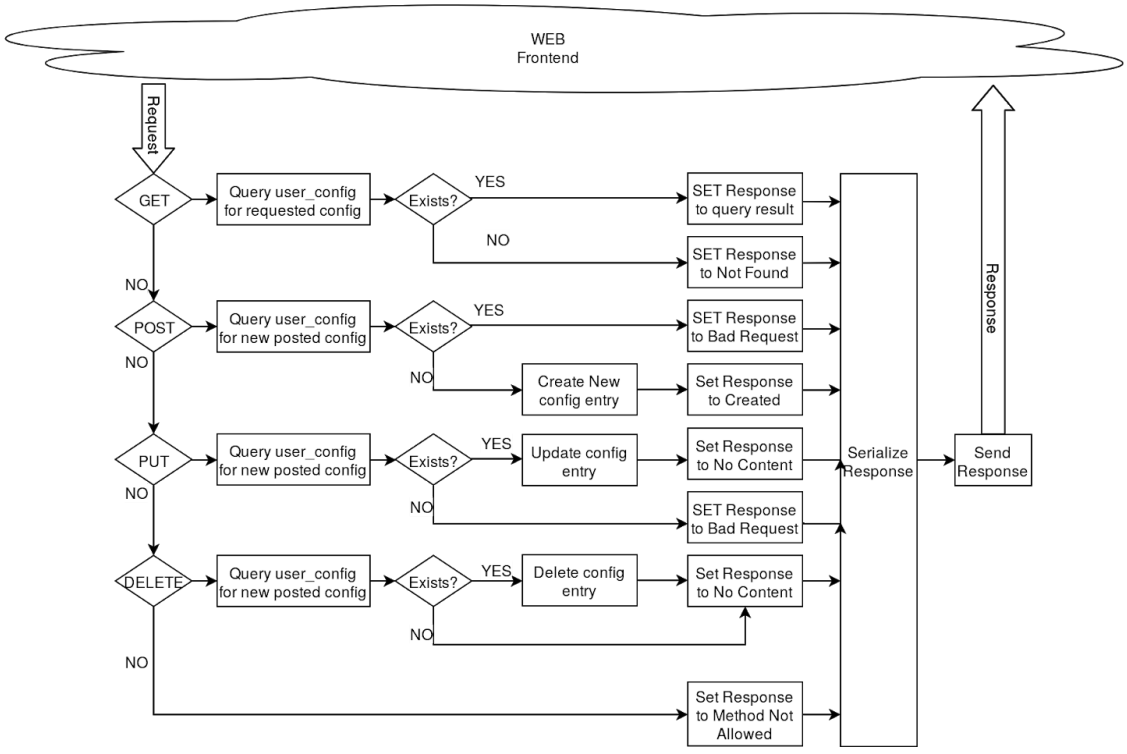


Figure 16: User Configuration Algorithm

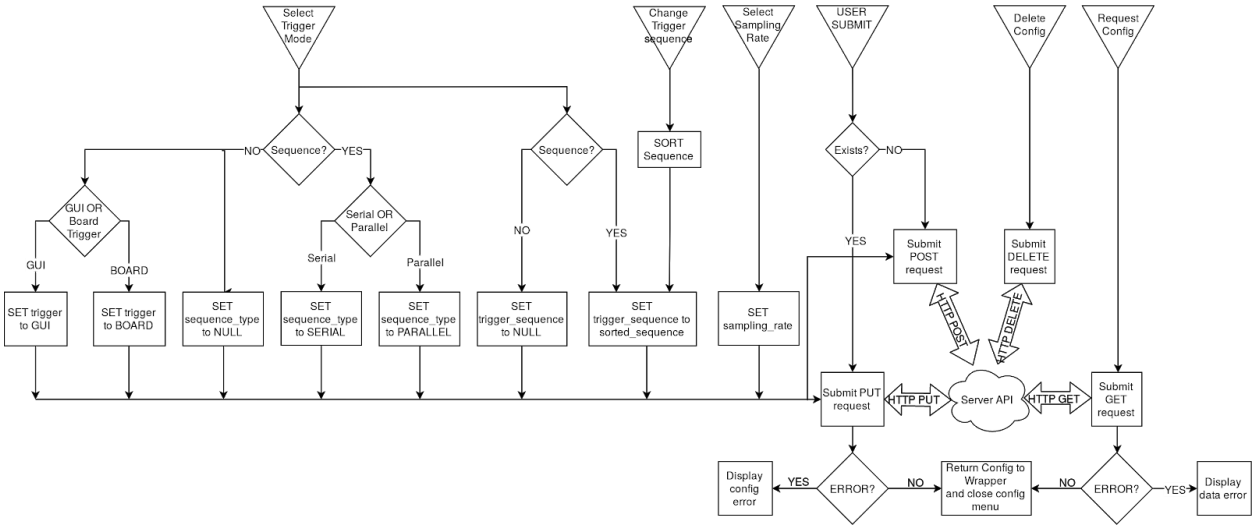


Figure 17: User Configuration Endpoint

Data Display Module

This module is a web front-end that takes the data sent to it from the server and processes it and turns it into a graph for interpretation of the user.

It functions as such:

Upon system init the GUI starts with the default view with no data loaded and the default user config loaded. Upon entering the user has the ability to alter the user config, load stored user configs, and delete user configs. From the user config menu the user is able to alter the trigger sequence, select the trigger mode, and select the sampling rate. When the user submits the config changes the frontend submits the appropriate request to either update or create the current config, then close the menu. From the data display the user can toggle auto-refresh or manually refresh the data. When data is received the display logic determines to appearance of each channel based off the data structures in the API response.

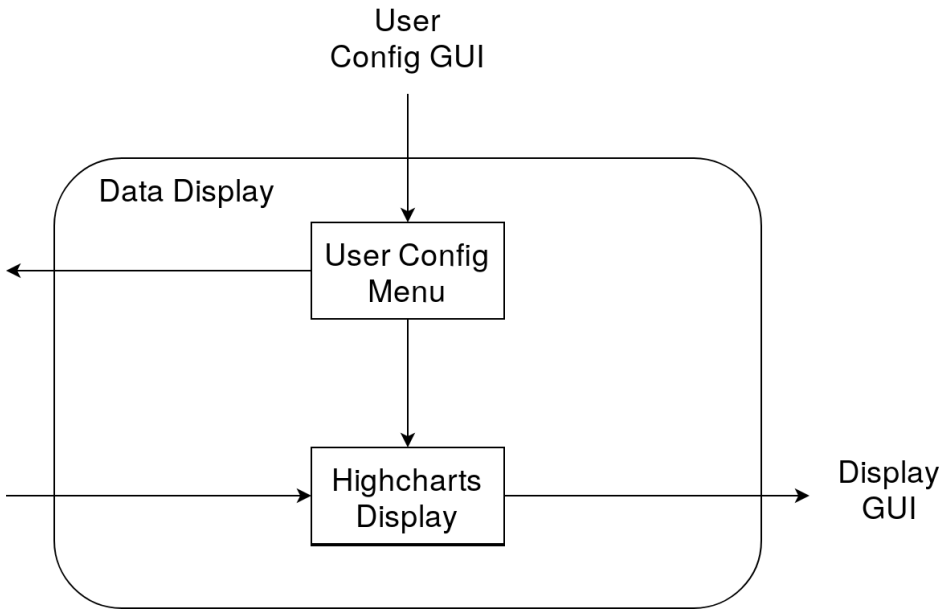


Figure 20: Data Displaying Algorithm

Experimentation:

Test Case 1

Goal:

To evaluate the basic operation and functionality of the logic analyzer.

Requirement Tested

1, 2, 3, 4, 6b, 6c, 6d, 7

System Components:

PC, Basys3 Board configured with Logic analyzer bitstream, USB cable, Commercial Logic Analyzer, MSP430 Trainer board

Testing Process:

- Attach Basys3 to PC using USB cable, and PMOD to breakout board. Attach MSP430 to connectors on board. Run testing firmware on the MSP430. Observe when a waveform appears and for how long

Data Processing and Visualization:

The data shall be gathered in 3 major points

1. Visually on board. A signal will flash when the board is triggered to operate. This can be

recorded with pictures.

2. On PC through Screen. This allows for recording of packets that are sent from the FPGA over to the USB serial port.
3. Visually on both Commercial front end and out front end.

Evaluation

Compare the difference in graph quality, and accuracy between commercial solution and our solution. Note the transfer speed by which the data moves.

Results

From these testing so far, we have these results

1.2 Test Case 2

Goal:

Verify operation of voltage regulator, SPI and I2C communication decoding.

Requirements Tested:

5, 6a, 8

System components

Logic Analyzer, MSP430, External variable power supply, PC

Testing Process

Start by running test 2 on Basys3 board and MSP430 while they are connected to the breakout board. Once completed attach external variable power supply to input channels and test the reading of voltages of 1.5v-5v.

Data collection and Visualization

Visual trigger on board Visual graph showing the SPI communication protocol decoding and I2C communication protocol decoding. Visual representation of variable voltages recorded

Evaluation

Verification of functionality.

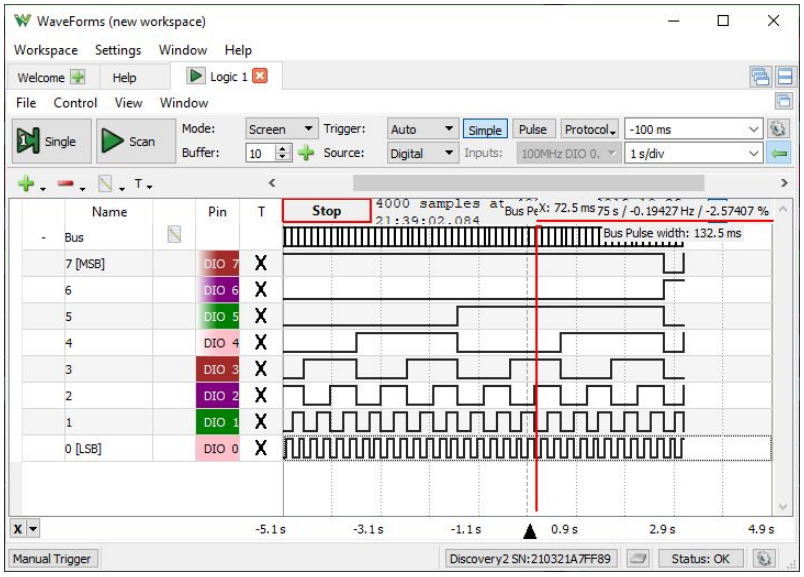
Results

Has yet to be completed

Experiment Validation Using Evaluation Criteria

Test 1:

From test one, we can look at the output that was sent to our front end. Since this was a large concern for functionality, we compare that graph to the following:



While there are a few differences, the main thing to notice is the general shape of our graphed data. While they dont both show the same amount of data, both are the same graph, of a binary

counter counting upwards, at a rate that can be seen as 4Hz on the lowest channel. This verifies that we have a working unit, that triggered its data off of a parallel defined pattern. We have not done formal testing of the transfer speeds.

Test 2:

Has yet to be tested

Issues

Reason for the project

The project was requested from Dr. Kaps in order to provide a free logic analyzer for students in his upper level engineering labs. The students already own the Basys3 board, and during ECE 447, the use of a logic analyzer can be beneficial towards the success of the lab.

Potential use of the project

As stated above, it is beneficial to have a logic analyzer in ECE 447's technical lab. During one such lab, called the DHT-22 Lab, students use an MSP430 to communicate with a temperature sensor that doesn't use a standard communication protocol. Because of this its recommended for students to use an oscilloscope or logic analyzer to determine what is happening on the communication pins. Without access to one of these devices, it's nearly impossible for the students to succeed in this lab.

Maintainability maintenance

Since the target user is a student at GMU, the project is not likely to need an update or maintenance unless the related labs change hardware. If the ECE department switched which FPGA the students are required to purchase, the design might have to be updated to account for this. Some components, such as the Microblaze softcore processor are Xilinx IPs, so if the new board can use these, maintenance should be easy. The overall utilization of basys3's resources is

somewhat low, so as long as the replacement board has a comparable or greater number of CLBs, this should not be an issue. A change in microcontrollers used in these labs should only affect usability of the logic analyzer if their clock speed is significantly faster than that of the MSP430 launchpad or if the new microcontroller outputs more than 3.3 volts. In this case, an external device can be used to step the voltage down to 3.3 volts. Due to the nature of FPGAs, the design and components are easily customizable, and could be changed to meet new requirements. The components do not have significant interdependence, and therefore can be easily modified with some knowledge about their inputs and outputs.

Retirement of the project

The project has the capability to last quite a long time. Its usage as an alternative replacement to GMU students purchasing their own logic analyzer means that the project can be functional until GMU ECE department no longer requires students to use the Basys3 board. At that time the project would either need to be retired or effort would need to be spent porting the project to the new platform.

Administrative Part

Project Progress

The design is almost complete. due to issues we haven't been able to fully test the functionality of the project. There has been confirmed readings of correct data into the front end, but nothing has been finalized and formally tested. The changes in design on the following section have caused a large delays in the project that have made it so finishing the testing has not been feasible.

Change in design,

Clock had to be updated to own ip

We struggled with data looking incorrect for quite some time, until finally we realised that the data we were receiving was being sampled off of a 100Mhz clock. This was incorrect, for we stated above that we wanted to sample from between 12.5KHz to 1Mhz. It turned out that the Xilinx Clock Wizard IP that we used to provide us a smaller clocked, couldn't reduce the clock farther than 400KHz. Due to this delay, we had to create a new clock divider component and integrate it differently into our design

Seven segment display addition

We wished to utilize the Seven Segment display on the FPGA to show status updates and the sampling clock frequency. This design was simple but required some time to finish

Delays due to issues that arose like fifo reading and sampling clock issues

As stated above, we spent a large amount of time trying to figure out what is wrong with our data reading. At first we spent a few weeks convinced that our usage of the FIFO between the microblaze and our IP was incorrect, but actually we determined that it was clocking wizards issue. While the fix was quick to implement, it wasted a large amount of time that could've been testing.

Funds Spent

Each of us had a Basys3 Board available from a previous class.

Product number	Description	Order Qty.	Price (USD)	Ext. (USD)
MC74LCX16245DTG	Bus Transceivers Bus Transceivers Low Voltage CMOS 16-Bit Transceiver	10	$0.68	$6.82
GRM033C71A104KE14D	Multilayer Ceramic Capacitors MLCC - SMD/SMT Multilayer Ceramic Capacitors MLCC - SMD/SMT .1UF 10V 10% 0201	100	$0.03	$2.60
M20-9740646	Headers & Wire Housings Headers & Wire Housings 06+06 DIL HORIZONTAL PIN HEADER TIN	20	$0.43	$8.68
68691-420H	Headers & Wire Housings Headers & Wire Housings	20	$0.32	$6.38

LF	BERGSTIK II 100CC DR			
68021-118H LF	Headers & Wire Housings Headers & Wire Housings 18P SR UNSHRD HRD .76 GOLD OVER NI	10	$0.80	$7.98
			Total	$32.46

Man Hours Devoted

Jacob - 100 Hours

Spencer - 150 Hours

Robert - 150 Hours

Jordan - 120 Hours

Lessons Learned

Knowledge and skills learned

Spencer

I gained a better understanding of the designing process for a solution. As the Project Manager, I also gained a better respect for those managers that can properly lead a team to success. More specifically. I learned how to and how not to lead a team to divy up tasks, and ensure progress is made on the tasks themselves. I also learned about AXI component usages, and how to integrate them into projects in many different formats.

Robert

I learned more about the creation and integration of a user interface that is easy and intuitive to use. I also learned more about the use of dynamic apis and serial communication protocols. I also learned a lot about working with a team to create a product in a short period of time.

Jordan

I learned about digital system design, AXI communication protocols, and design and implementation of a project as a team. I gained experience using Vivado and Xilinx SDK for the implementation of a digital system. I learned the value of testing at each step, especially when working as a team.

Jacob

I increased my understanding of integration of FPGA and VHDL design from a production standpoint. This includes using tools designed and protocols designed by Digilent and Xilinx such as AXI communication. Project and development planning were also skills I have picked up over the course of this project. The value of what I learned about working as a team, project development, and troubleshooting will be invaluable moving forward.

Appendix A: Proposal

GMU Basys3 Logic Analyzer

Spencer Lombardo, Jordan Whitesell, Robert D. Minter, Jacob Samuels

March 1, 2019

1 Executive Summary

Logic analyzers are electronic instruments that are used to capture and display signals from a digital circuit/system. Logic analyzers range from simple devices which have to be attached to a PC and cost around $200 having 8 channels and 100 MS/s sampling rate to more complex industrial models costing more than $10,000 and having 32 channels and 500 MS/s sampling rate. These devices offer great utility for students in Electrical and Computer Engineering programs. However, even the cost of the cheapest models of logical analyzers are prohibitively expensive for an engineering student.

On this project, we will fulfill the task of designing, building and implementing a 16 channel logic analyzer. This logic analyzer will detect, record, and store signals from inputs connected to circuits. This logic analyzer will be paired with a front end to display signals graphically, allowing for easy analysis of the circuit and its functionality.

1.1 About the team

Our team consists of four students who have extensive work in both FPGA controllers and python based front ends. Each student has completed undergrad work that included the designing

and flashing of digital circuitry on an FPGA and building a MIPS based processor in VHDL. Each of the students is also taking a more intensive FPGA course that is focused on advanced uses of FPGA and how to design on them.

Project Manager: Spencer Lombardo

Spencer is a senior in the ECE department at George Mason. He pursuing his bachelors in Computer Engineering with a concentration in Signal Processing, as well as minor in Mathematics. In addition to the relevant course work above, he has been the Learning Assistant for multiple VHDL labs, helping to run and teach the students taking the courses in working with the Xilinx Vivado suite, debugging VHDL, and general concepts based around the courses. Spencer also has a passion in working with embedded systems and would like to find work in implementing hardware acceleration using FPGA's and ASIC's in the signal processing field, preferably in a musical aspect.

Jacob Samuels Jacob is a senior at George Mason University pursuing a bachelors in Computer Engineering with a concentration in robotics. He has taken many courses, including the relevant stated above, dealing with computer architecture, Xilinx Vivado suite, and general concepts that are contained within the courses taken. Jacob has a passion for drones and aviation and hoping to find a career working in the aerospace industry.

Jordan Whitesell Jordan is a senior in the ECE department at GMU. He is pursuing a degree in Electrical Engineering with a concentration in Computer Engineering and is interested in work using VHDL to develop FPGA technologies.

Robert D. Minter Robert is a senior Electrical Engineering major concentrating in computer engineering and minoring in computers science. Throughout his education at George mason, Robert has interned at Micro Technologies in Manassas as a Software engineer. During his internship Robert has developed multiple large python projects for analysis and display of large datasets.

2 Problem Statement

2.1 Motivation

This project stems from two main sources. One influence is from a lack of available tools such as oscilloscopes and logic analyzers for labs later in the college career of ECE students at GMU. This leads to labs such as ECE 447, Micro-controllers, to be more challenging than they need to be due to this lack of helpful tools. These tools let the students analyze the circuits that they built for the micro-processor to ensure correct function, as well as monitor different communication protocols on select pins to make sure the protocol is functioning properly. The other influence comes from the availability of Basys3 boards. These boards are Artix-7 equipped launchpads that allow for the reprogramming of the internal FPGA to test different circuits. Each GMU ECE student must purchase one of these from their Introduction to Digital circuits lab, where they learn the basics of digital circuit design, and VHDL modeling and synthesizing.

2.2 Identification of Need

For labs following the digital circuits class, such as the microprocessor lab and computer organization lab, a logic analyzer can be very beneficial to the success of the students. The

students would have access to a tool for virtually ”free”, provided they have taken ECE 332, where the Basys3 board is required to complete the class.

2.3 Market/Application review

Looking at what is currently available on the market, logic analyzers are expensive devices. With standard ones starting at $200 and increasing exponentially, they tend to have limited options and features for ones that are within the budget of schools and students. Due to the easy access students have to an FPGA launchpad, such as the Basys3 board, an advanced logic analyzer can be built for it and made available for the students to flash on their board. This gives the students access to a logic analyzer to use in labs for free.

3 Approach

3.1 Approach

The Basys3 board contains a Artix-7 FPGA, enough PMOD headers for 16 channels, and available communication over the UART-USB controller built into the board. The plan is to build a logic analyzer, and accompanying front end system, that runs locally on a host machine. The host machine connects to the FPGA, establishes connection, and assigns the channels that are hooked up to a circuit. Then when the analyzer is triggered, it records and sends data over usb to the front end, where it is displayed for the user to look at and scrub through.

The basic design of the FPGA is a Microblaze softcore processor that connects to the on board UART controller. This processor waits for the detection of a trigger, either rising or falling edge on an input signal, and then grabs the data stored in RAM to be communicated over usb. This

RAM is populated by a component that connects to the PMOD headers on the Basys3. This component accesses a sampling clock, and records data on the rising edge of the clock. In order to incorporate the detection of sequences, and the analysis of communication protocols such as I2C and SPI, additional hardware on the board can be designed using state machines.

The front end will be designed from the ground up using python into a small webserver that the local PC hosts. This server simply serves as a connection from the host to the Basys3 board, and will grab the data and send it to the graphical interface. This interface is designed in python to run on any web browser, and allows cross platform operation. The interface will include a time graph of each 16 channels, and allow for a plethora of analysis options. These include closing channels to allow for less clutter, signal scrubbing and selection of representation between the signals.

3.2 Alternative Approach

Due to the limitation on hardware available, and wanting to make the logic analyzer available to the entirety of the GMU ECE department of students, the Basys3 board is a fast requirement. The initial design of using the microblaze softcore into a UART-USB driver is the obvious solution. Another approach to the problem is to use an external microprocessor and display to make the system completely enclosed. Rather than requiring a local computer, a system such as a raspberry pi W zero, with a 5’ LED touch display could function as a completely closed off box that anyone could plug into a power source and connect to circuit.

4 System Design

4.1 Functional Decomposition

The system must fulfill the following requirements:

1. Record 16 channels of data from PMOD headers located on the board
2. Interpret Channel data
    (a) Decode SPI and I2C
    (b) Decode Sequences as necessary
    (c) Detect potential Clock signals
3. Listen for Triggers
    (a) Record on trigger
    (b) stop recording on trigger
4. Generate Graphical data

A Level 0 approach to the following requirements can be show with the following diagram:

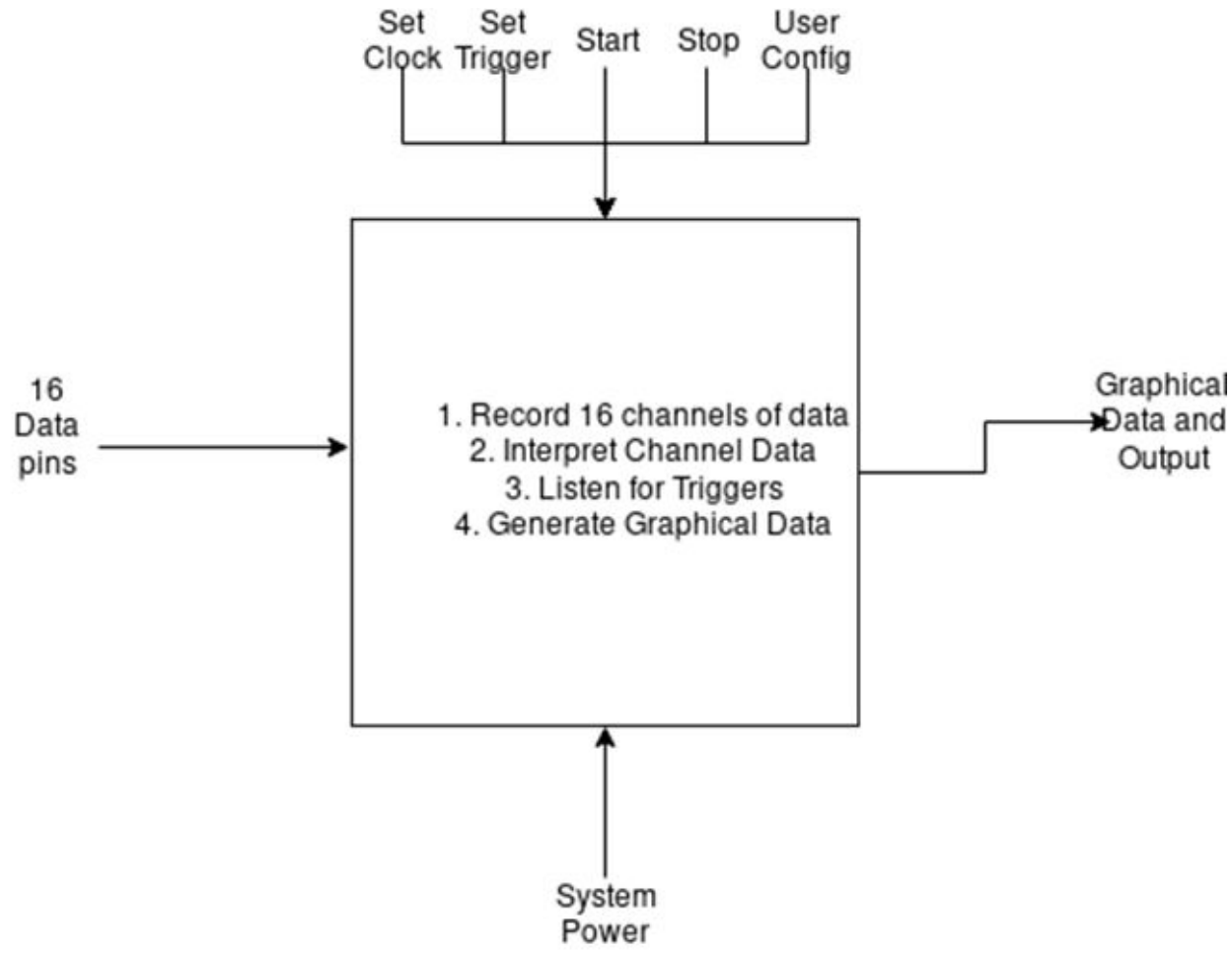


Figure 1: Level 0 Decomposition

There will be an external requirement of system power, and a system input of 16 possible data pins.

The user also has the ability to configure different things in the system, such as the sequence to detect and if a group of signals should be i2c. The user can also set the sampling clock, the trigger, and start and stop the system.

The system will output to the web browser the graphical data and any meaning full analysis that needs to be made.

Digging deeper we get the Level 1 decomposition in the following diagram.

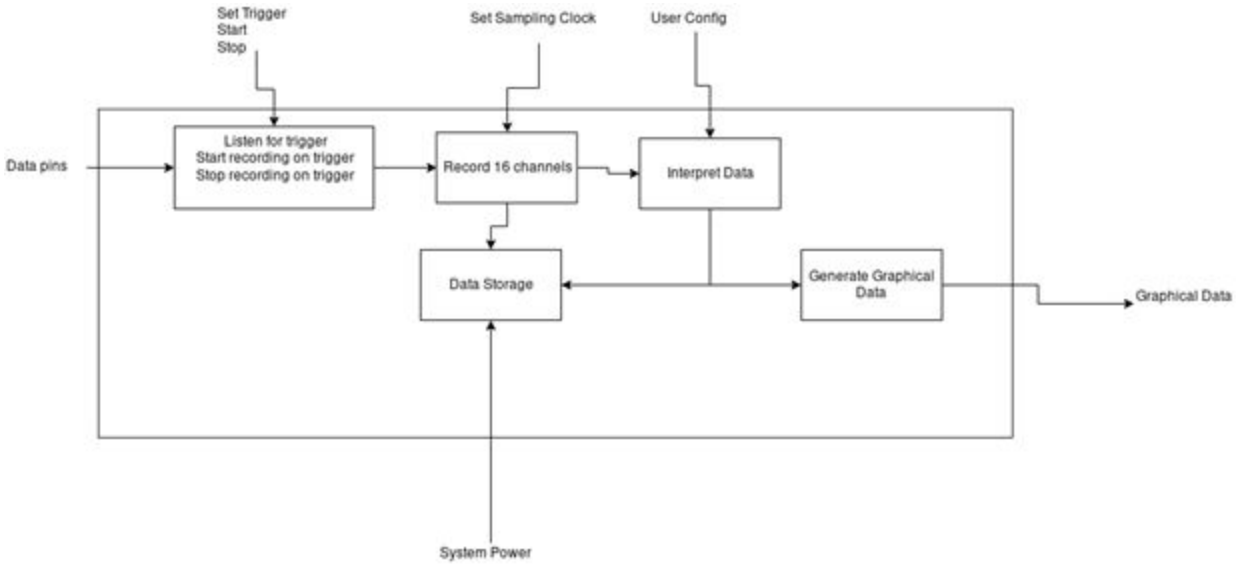


Figure 2: Level 1 Decomposition

We can see from the figure that the data pins will first connect to the system that waits for a trigger. When the trigger happens, either from user input or from a assigned trigger, the system starts recording the data. this data is recorded at the sampling frequency that is assigned by the user. This data then flows into both the interpreter and the data storage. The data storage holds the data from transmission over to the graphical user interface, while the interpreter checks the recorded data for different things, such as a sequence, or the recognition of communication protocols like I2C and SPI.

4.2 System Architecture

Figure 3: General Physical Architecture

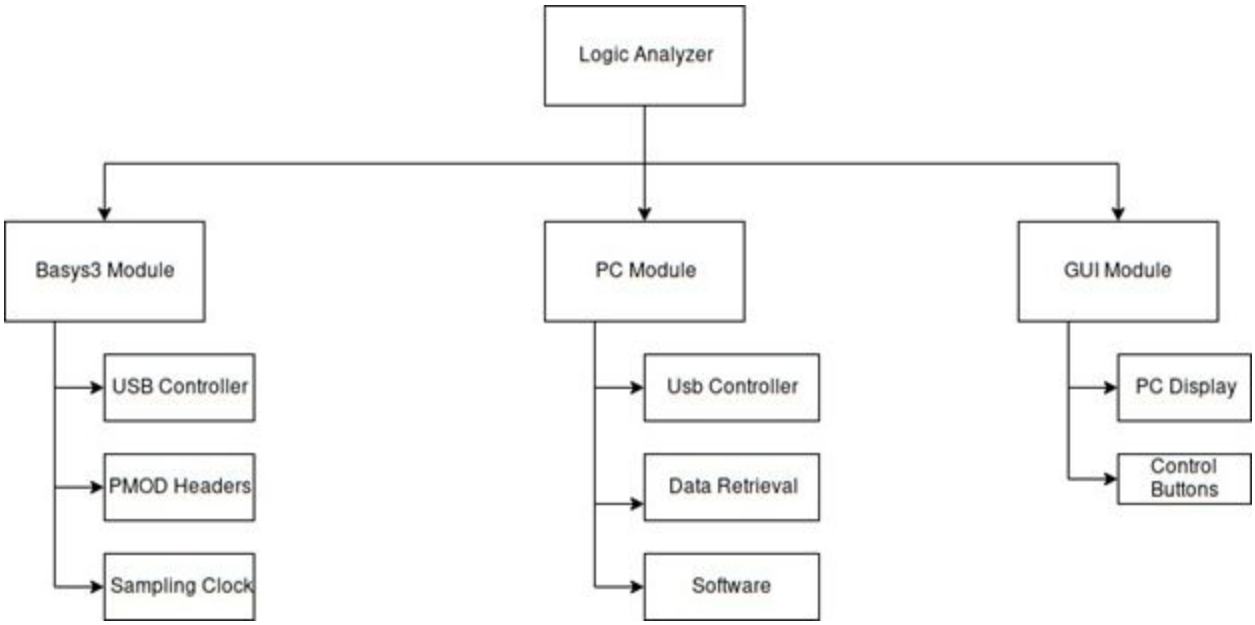
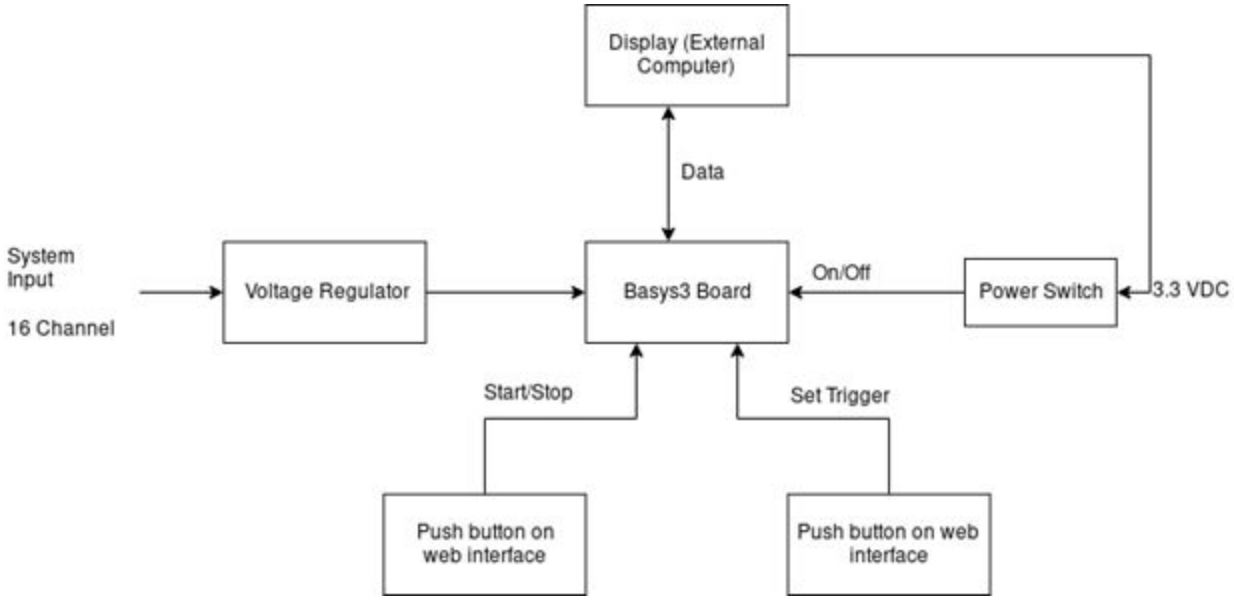


Figure 4: System Architecture



5 Prelimiary Experiment plan

There are a few requirements that must be filled for the project. They go as follows.

1. Test data recording at required sampling clock rates
2. Test Recognition of sequences
3. Test Detection of rising edges for triggers
4. Recognition of I2C and SPI protocols.
5. Validate the signal recorded is being displayed properlyHere is a list of tests to be run to ensure these are met:

1. Program a microprocessor to blink at a set rate on one of its pins, and connect that to achannel on the logic analyzer. Then test to ensure the trigger causes recording, and ensure the recording is accurate.
2. Program the microprocessor to i2c communication and SPI communication sending simplemessages. Test to ensure the protocol is working properly to be detected by the logic analyzer.

6 Project plan

The plan for the project is as follows:

- Set up a microblaze processor on Basys3 FPGA.
- Set up the UART-usb module to communicate with a host pc.
- Write a simple python script to send data over usb and echo it back.
- Send a simple message over from the FPGA.
- Ensure the message is received and echoed back.
- Design communication protocol for configuration.
- Setup system to sample on a clock from PMOD headers.

- Test and ensure recording of data to internal RAM/Registers.
- Write system of grabbing data and sending over USB to python server.
- Test and ensure system is sending sampled data.
- Implement user configurations such as trigger and sampling clock assignment.
- Test configuration on communication protocol.
- Ensure data is sampled properly and triggered properly.
- Design front-end to display data, and configure system.
- Implement sequence detection on the FPGA side.
- Implement I2C and SPI communication detection.

7 Potential Problems

The only foreseeable problem is speed and transmission limitation on the usb2 communcation port on the Basys3 board. The limitation on transmission speed can limit the ability to have live view of signal from the logic analyzer. It might also limit how much data is recorded, limiting the sampling rate and therefore max frequency of the signal that is recorded using the logic analyzer.

Appendix B: Design Document

GMU Basys3 Logic Analyzer

Spencer Lombardo, Jordan Whitesell, Robert D. Minter, Jacob Samuels

May 3, 2019

1 Problem Statement

1.1 Motivation

This project stems from two main sources. One influence is from a lack of available tools such as oscilloscopes and logic analyzers for labs later in the college career of ECE students at GMU. This leads to labs such as ECE 447, Micro-controllers, to be more challenging than they need to be due to this lack of helpful tools. These tools let the students analyze the circuits that they built for the micro-processor to ensure correct function, as well as monitor different communication protocols on select pins to make sure the protocol is functioning properly. The other influence comes from the availability of Basys3 boards. These boards are Artix-7 equipped launchpads that allow for the reprogramming of the internal FPGA to test different circuits. Each GMU ECE student must purchase one of these from their Introduction to Digital circuits lab, where they learn the basics of digital circuit design, and VHDL modeling and synthesizing.

1.2 Identification of Need

For labs following the digital circuits class, such as the microprocessor lab and computer organization lab, a logic analyzer can be very beneficial to the success of the students. The

students would have access to a tool for virtually ”free”, provided they have taken ECE 332, where the Basys3 board is required to complete the class.

1.3 Market/Application review

Looking at what is currently available on the market, logic analyzers are expensive devices. With standard ones starting at $200 and increasing exponentially, they tend to have limited options and features for ones that are within the budget of schools and students. Due to the easy access students have to an FPGA launchpad, such as the Basys3 board, an advanced logic analyzer can be built for it and made available for the students to flash on their board. This gives the students access to a logic analyzer to use in labs for free.

	Digilent		Saleae	Our Solution
	Analog Discovery 2	Digital Discovery	Logic 8	GMULA
Cost	279	199	399	100-149*
OS compability	Windows, Mac, Linux			
# of Channels	16	32	8	16
Max Sampling Frequency	100Ms/s	800Ms/s	100Ms/s	100Ms/s
Voltage tolerance	1.8v-5v	1.2v-3.3v	1.8v-5.5v	1.8v-5v
Supported Application	Waveforms		Saleae Logic	Our own

Figure 1: Currently Offered Solutions

2 Approach

2.1 Approach

The Basys3 board contains a Artix-7 FPGA, enough PMOD headers for 16 channels, and available communication over the UART-USB controller built into the board. The plan is to build a logic analyzer, and accompanying front end system, that runs locally on a host machine. The host machine connects to the FPGA, establishes connection, and assigns the channels that are hooked up to a circuit. Then when the analyzer is triggered, it records and sends data over usb to the front end, where it is displayed for the user to look at and scrub through.

The basic design of the FPGA is a Microblaze softcore processor that connects to the on board UART controller. This processor waits for the detection of a trigger, either rising or falling edge on an input signal, and then grabs the data stored in RAM to be communicated over usb. This RAM is populated by a component that connects to the PMOD headers on the Basys3. This component accesses a sampling clock, and records data on the rising edge of the clock. In order to incorporate the detection of sequences, and the analysis of communication protocols such as I2C and SPI, additional hardware on the board can be designed using state machines.

The front end will be designed from the ground up using python into a small webserver that the local PC hosts. This server simply serves as a connection from the host to the Basys3 board, and will grab the data and send it to the graphical interface. This interface is designed in python to run on any web browser, and allows cross platform operation. The interface will include a time graph of each 16 channels, and allow for a plethora of analysis options. These include closing channels to allow for less clutter, signal scrubbing and selection of representation between the signals.

3 System Design

3.1 Functional Requirements

The system must fulfill the following requirements:

1. Record 16 channels of data from PMOD headers located on the board
2. Analyze I2C and SPI Protocols
3. Record on selected Trigger Mode:
    (a) Trigger on parallel read sequence.
    (b) Trigger on single channel trigger sequence.
    (c) Physical on-board button press.
    (d) Gui button press.
4. Generate Graphical data
5. Front end should be easy to install. I.E. one click script
6. Front end should work on all major operating systems.

There is also a list of requirements for the graphical interface. It should cover the following functionality:

1. Scroll through time axis.
2. Zoom in on the time axis.
3. Collapse Channels.
4. Define names/groups for channels.
5. Assign the sampling clock, trigger mode.

6. Start and stop the data recording.
7. Interpret I2C and SPI communication protocols.
8. Concat and Display selected channels.
9. Display information in selected Radix.
10. Enable and disable selected channels.

3.2 Level 0 Decomposition

A Level 0 approach to the following requirements can be show with the following diagram:

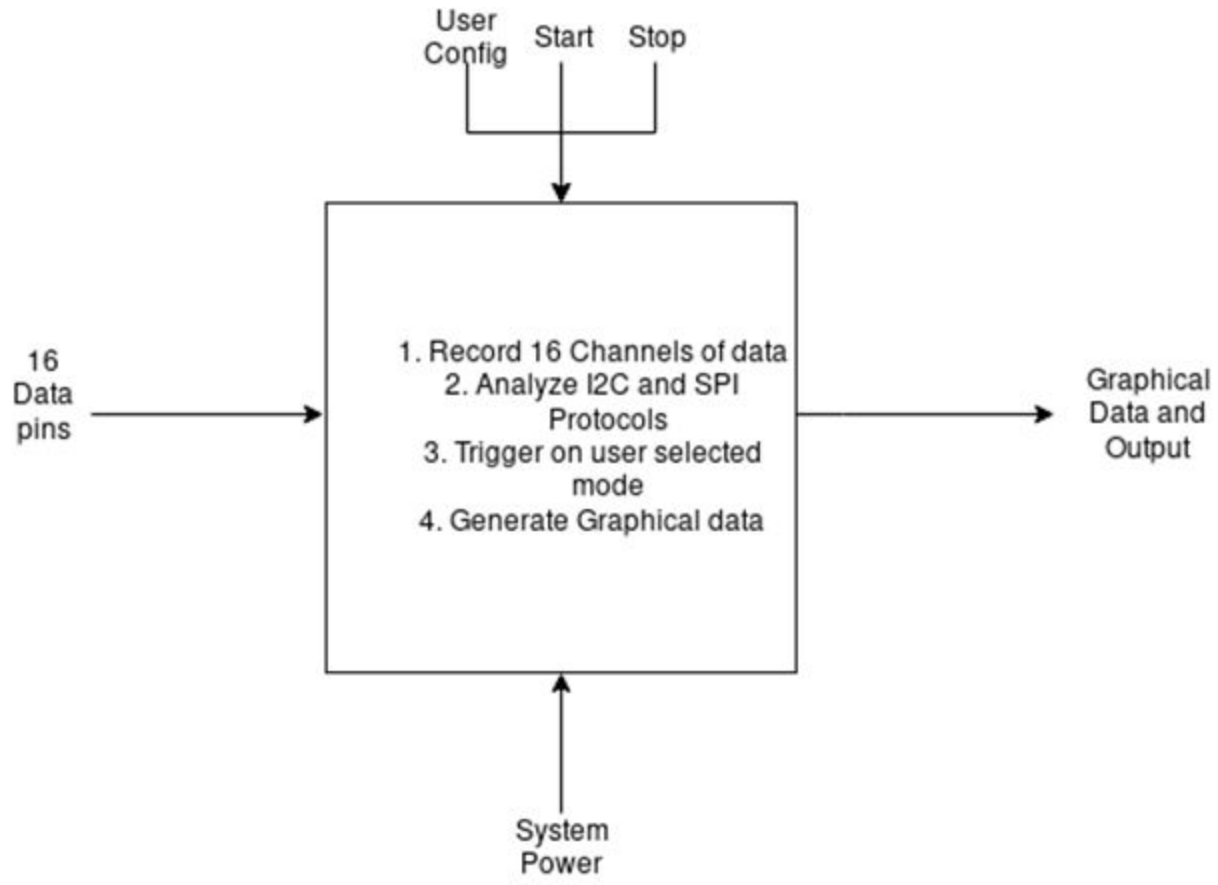


Figure 2: Level 0 Decomposition

There will be an external requirement of system power, and a system input of 16 possible data pins.

The user also has the ability to configure different things in the system, such as the sequence to detect and if a group of signals should be i2c. The user can also set the sampling clock, the trigger, and start and stop the system.

The system will output to the web browser the graphical data and any meaning full analysis that needs to be made.

3.3 Level 1 Decomposition

Digging deeper we get the Level 1 decomposition in the following diagram.

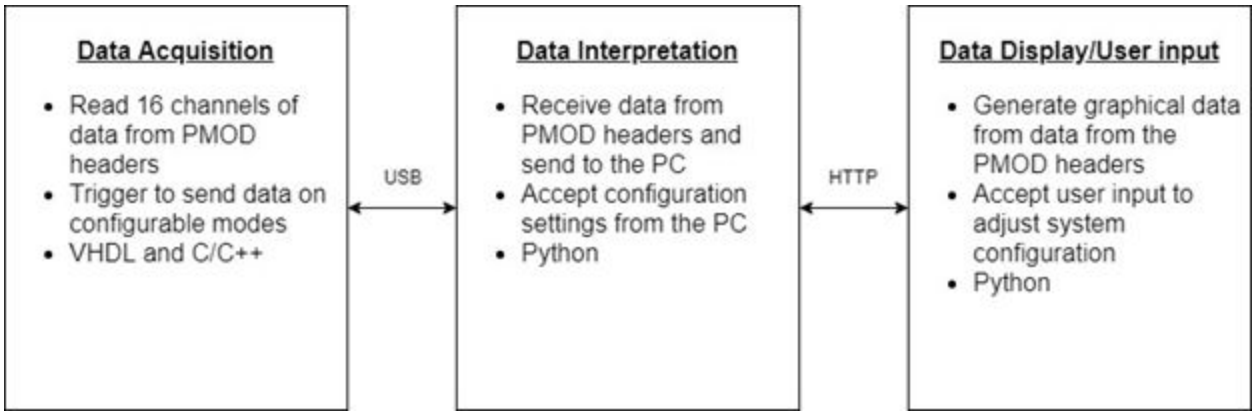


Figure 3: Level 1 Decomposition

We can see from the figure that the data pins will first connect to the system that waits for a trigger. When the trigger happens, either from user input or from a assigned trigger, the system starts recording the data. this data is recorded at the sampling frequency that is assigned by the user. This data then flows into both the interpreter and the data storage. The data storage holds the data from transmission over to the graphical user interface, while the interpreter checks the

recorded data for different things, such as a sequence, or the recognition of communication protocols like I2C and SPI.

3.4 Level 2 Decomposition

3.4.1 Data Acquisition Module:

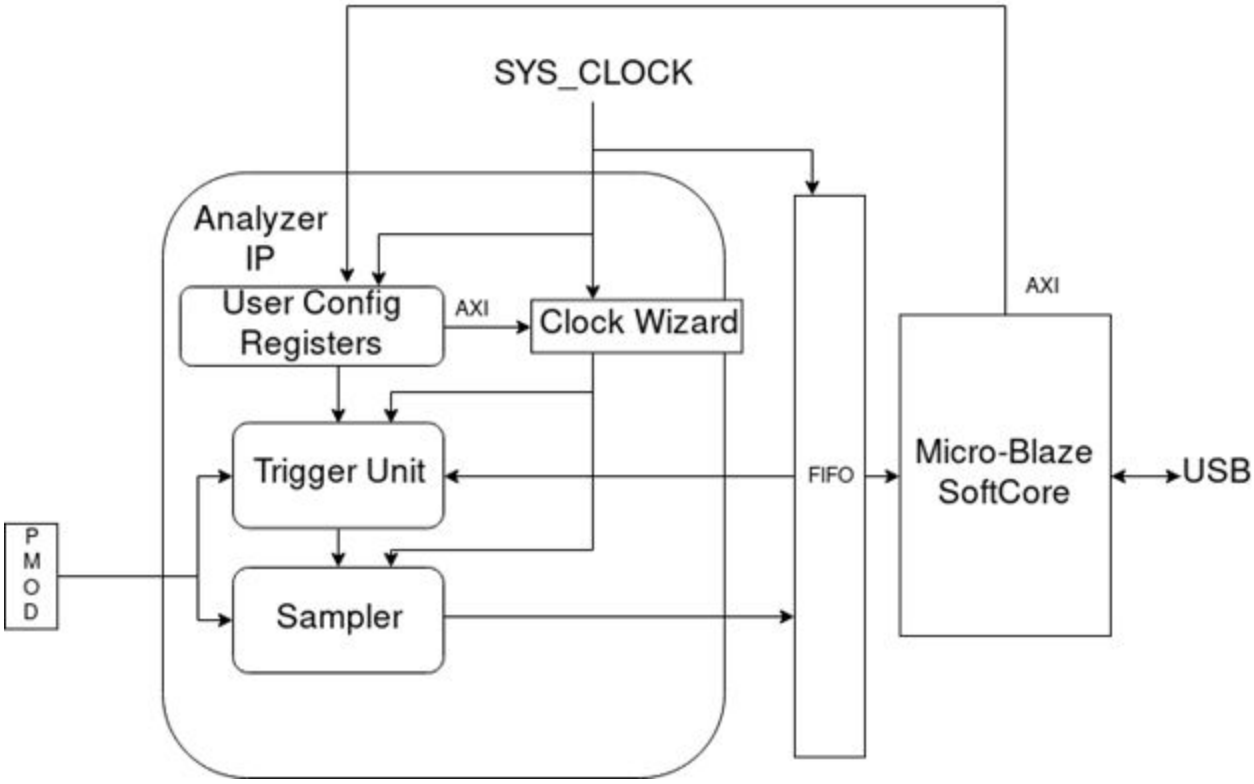


Figure 4: Data Acquisition Level 2 Decomposition

For this module there is 6 primary components and have different communication protocols that are connecting them together

1. User Config Registers - Store the configuration for the rest of the components.
2. Trigger unit - Takes in the data and checks against sequence and selection for triggers.
3. Sampler - Grabs the data and stores it. Then on trigger sends it to the FIFO.

4. FIFO - First in First Out buffer that sends the data to the softcore for transmission to PC.
5. Clock Wizard - Sets up and outputs the sampling clock for the data. Can be reconfiguredfrom the user config registers.
6. Micro-Blaze Soft-core - A soft-core processor for USB communication to the PC.

3.4.2 Data Interpretation Module:

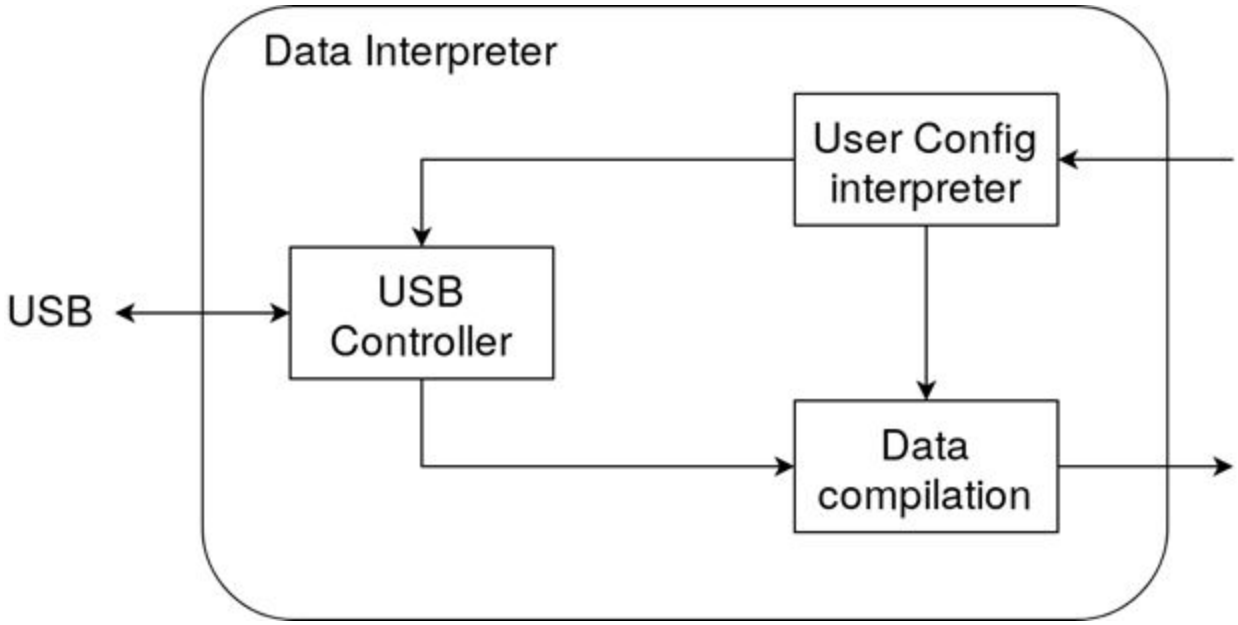
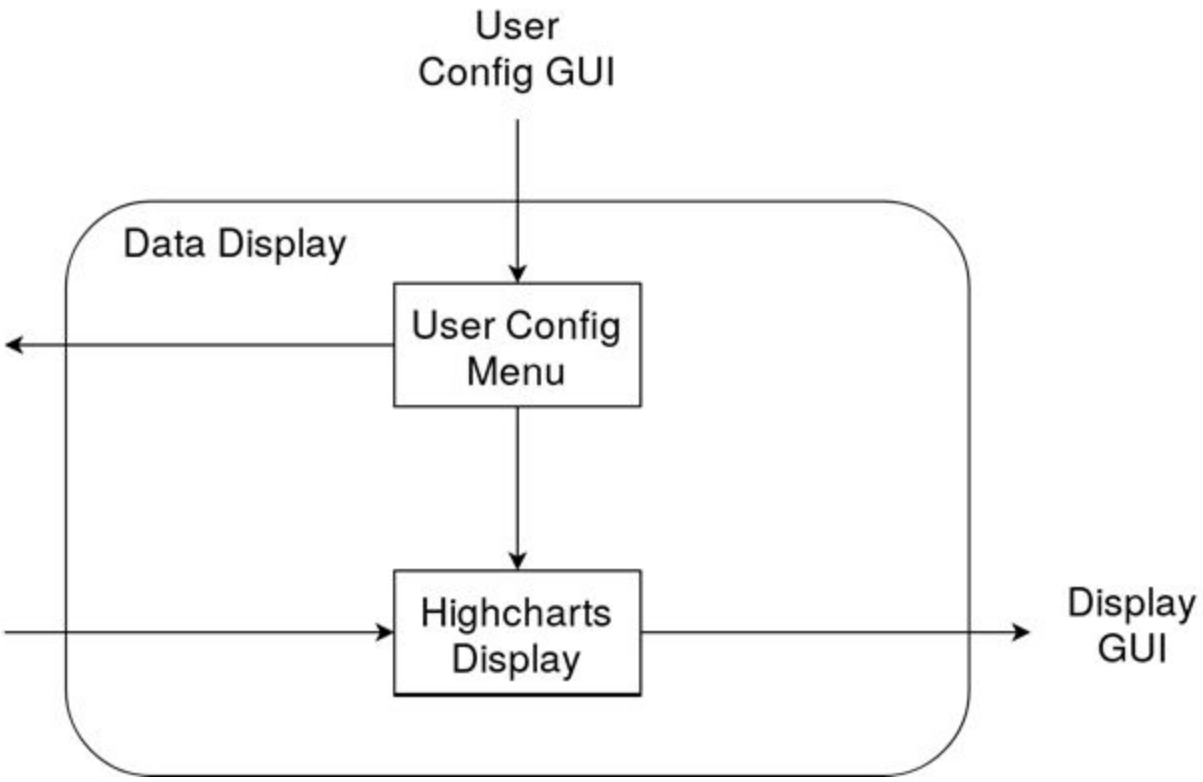


Figure 5: Data interpretation Module

3.4.3 Data Display Module:



4 Background Knowledge:

4.1 AXI communication

AXI is a burst-based communication protocol used for controlling and managing functional blocks in a SoC. Microblaze uses an AXI connection to interact with its connected peripherals. An AXI connection is composed of the following five unidirectional channels: read address, read data, write address, write data, and write response. The read address and write address channels send register address information as well as various ID tags and control information between master and slave devices. These channels also signal the receiving device to start sending data or to inform the receiving device that data is about to be sent. After receiving this signal, data is

sent through the write data or read data channel repeatedly until the transaction is complete. A signal is sent at the end of data transfer to signal the completion of data transfer. For a write operation, the slave device will send a write response signal to communicate that it has received all data. Each transaction uses a combination of valid and ready signals from master and slave in what is known as a handshake process. When sending data, the sending device will signal that the data is valid to the recipient. The recipient will signal that it is ready to the sender. If both ready and valid signals are not asserted during a data transfer, the transfer is stalled until both are asserted.

AXI4-Lite is a subset of the AXI4 protocol much of the complexity removed, and it is mostly compatible with AXI4. The ID tag and most control signals are not used in AXI4-Lite protocols. The simplicity of AXI4-Lite makes it very useful for configuring control registers.

Microblaze's primary I/O bus uses an AXI interface, and Xilinx's clock wizard IP uses an AXI connection between itself and Microblaze for its dynamic reconfiguration setting. To accommodate for this, the on-board components that communicate with Microblaze should have be packaged with AXI4 functionality to allow them to send and receive data as defined by the AXI4-Lite protocol. The trigger unit will take user input from the GUI for one of its triggering modes, and it will therefore need an AXI connection from Microblaze. The user configuration registers also take their configuration data from the selections made by the user in the GUI, which will require AXI communication. The sampler sends its data to Microblaze, so it should do so as defined by AXI protocols.

4.2 Serial Peripheral Interface (SPI)

Serial Peripheral Interface, or SPI, is a serial communication protocol that is used to communicate between two devices. It works by using two shift registers, one in each devices controller, and sending data over the bus, shifting it out of each register. There are two main types of implementations, ones with 3 wires and others with 4. The more common standard is to use four. SPI is a full duplex interface

The four wires are seen as such:

- SS - Slave select: active low, when pulled low it indicates for the slave to start reading and sending data.
- SCLK - Clock, the shared clock that is given by the master so that the communication is in sync.
- MOSI - Master out slave in - The wire where the master's data shifts out, runs into the slave input.
- MISO - Master in Slave out - The wire where the slaves data is shifted in.

4.3 Inter-Integrated Circuit (I2C)

A serial communication protocol that is used to communicate between two devices. Is normally implemented with 3 wires, described as such:

- SCLK - The synchronous clock
- SDA - Data line
- Ground

This protocol follows this diagrams description:

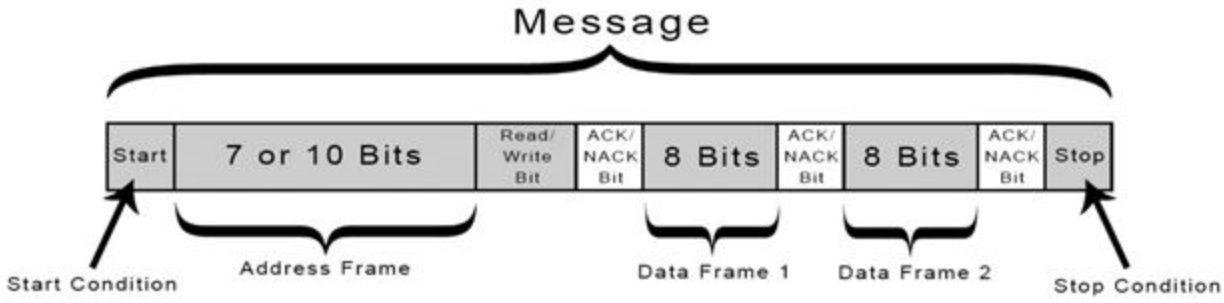


Figure 6: I2C description

4.4 Communication Protocol

A communication protocol must be designed for user configuration of the Logic Analyzer. The following is the specification for such protocol:

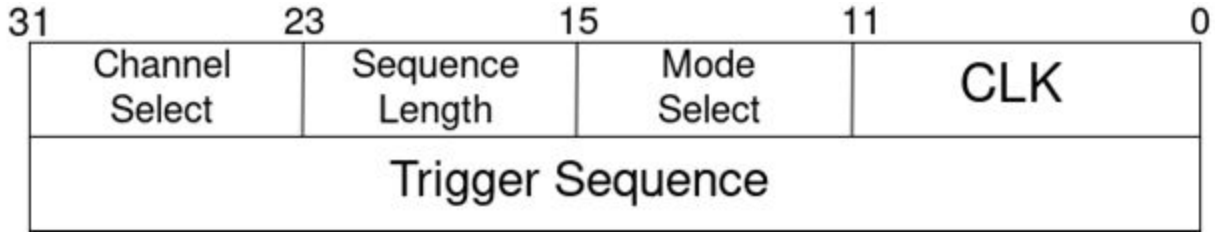


Figure 7: Communication Protocol Diagram

For the protocol we decided on 64 bits of data needs to be transmitted. This allows for some user extensibility for future applications. The following is the explanation for each section and how the bits are parsed:

- Bits 0-11 : 12 Bits for sampling clock selection. This allows for 2^12 different sampling clock frequencies.
- Bits 12-15 : 4 Bits for trigger mode selection. This gives 16 different trigger modes. We will be using only 2 of these bits for 4 trigger modes.

- Bits 16-23 : 8 Bits for trigger sequence length. This specifies the length of the serial trigger sequence. We will be used 4 of these bits due to the limitation of a max of 16 bits for sequence detection.
- Bits 24 - 31 : 8 bits to select which channel the serial trigger will check against. Again we only use 4 of these bits because we are only reading in 16 channels of data.
- Bits 32 - 63 : 32 bits for the user defined trigger sequence.

5 Design:

5.1 Data Acquisition Module:

5.1.1 User Configuration Registers:

This module will contain all the registers that are used to configure each other module that requires user configuration, such as the trigger unit. This is the block design of the user configuration registers.

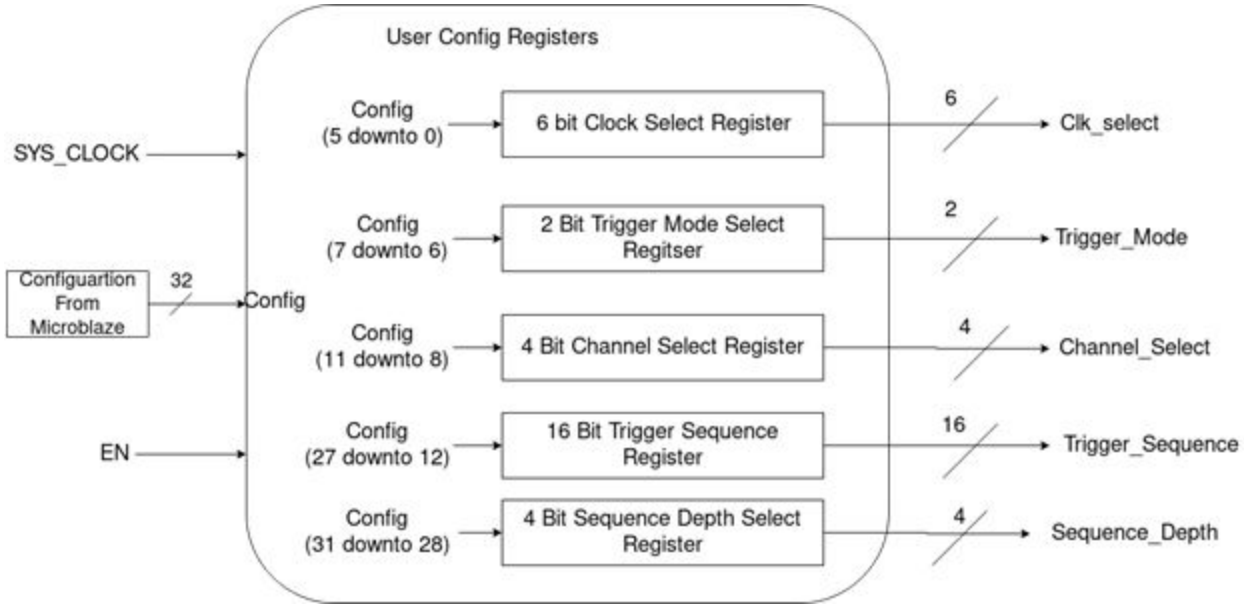


Figure 8: User Configuration Register Block Diagram

5.1.2 Triggering Unit

This module takes in the data every clock cycle, and checks against different user configurations to tell if the data should be recorded. This can be done one of four ways:

- Physical Button - The center button on the Basys3 board can be pressed to trigger the board to start recording
- Gui Button - A Button on the GUI can be pressed, and a trigger is sent through the PC to the board to start recording
- Parallel Sequence detection - The board can test all 16 inputs of the PMOD headers against a 16 bit user configurable trigger sequence. When each single channel matches the exact bit in the trigger sequence, then the board will start recording
- Single channel Serial Sequence Detection - The board will test a single channel until it matches a user configurable trigger sequence. This sequence can be anywhere from 1 to 16 bits long.

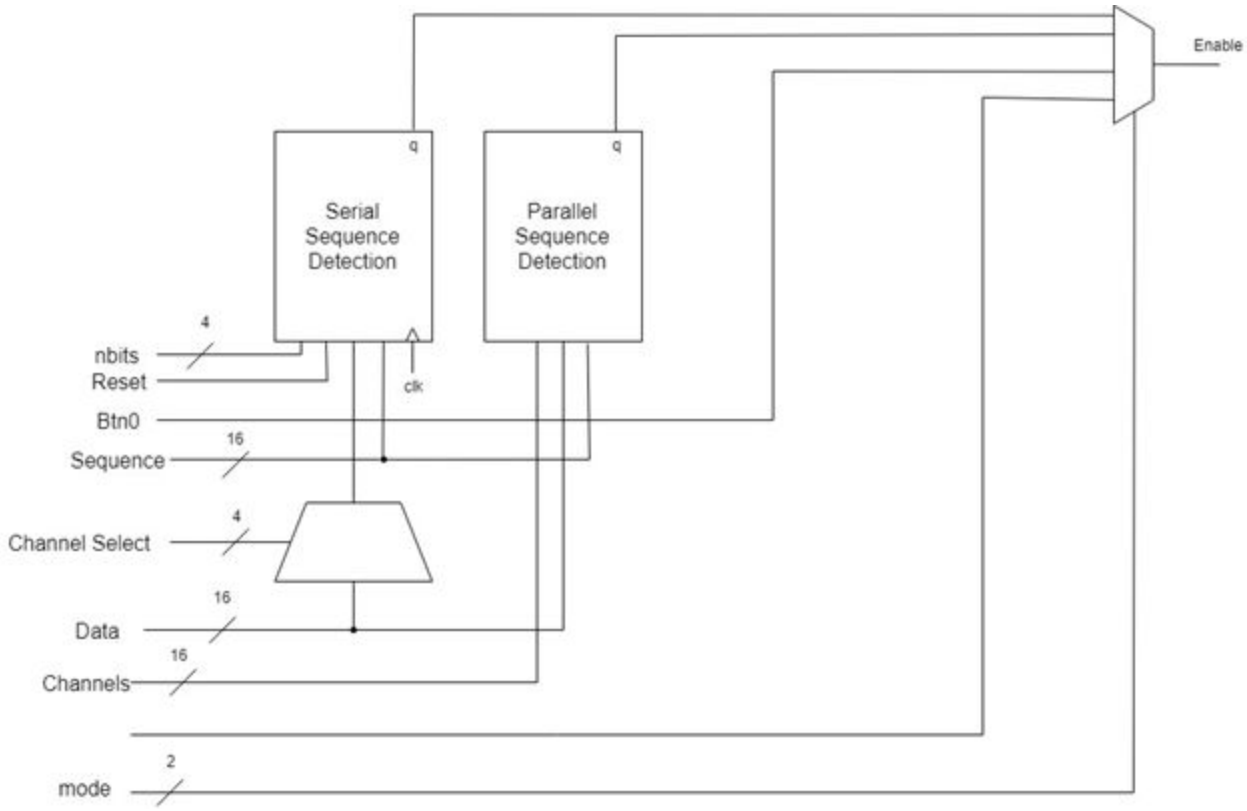


Figure 9: Top Level Trigger Block Diagram

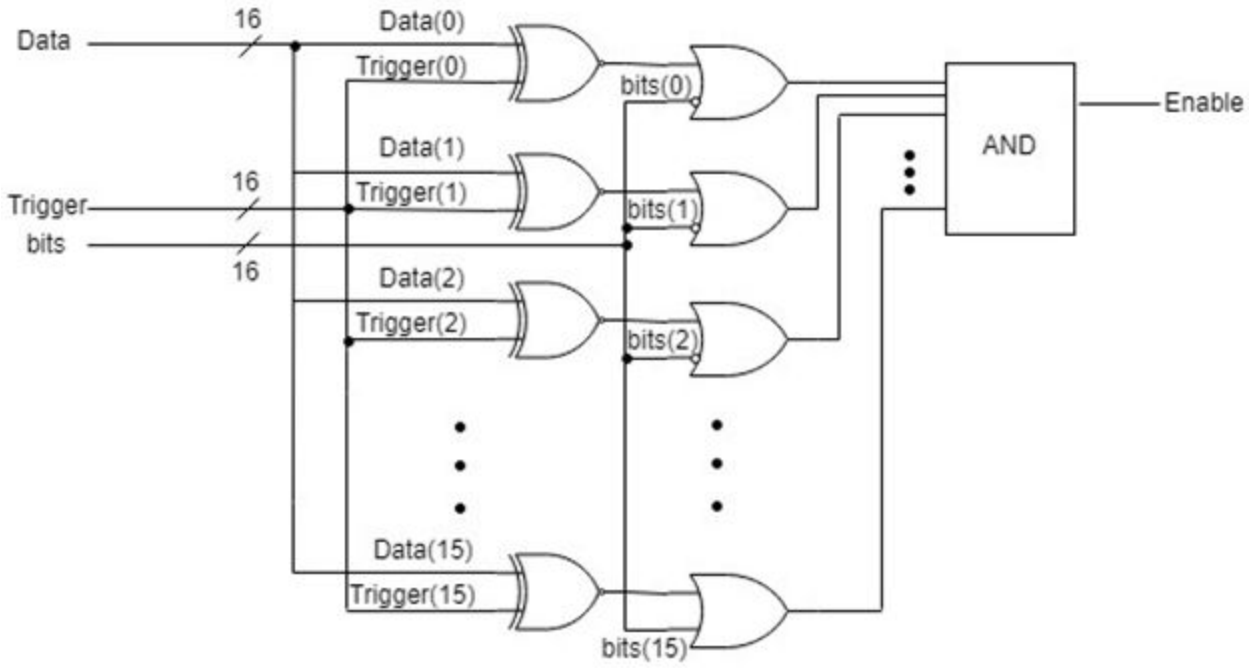


Figure 10: Parallel Sequence Detection Block Diagram

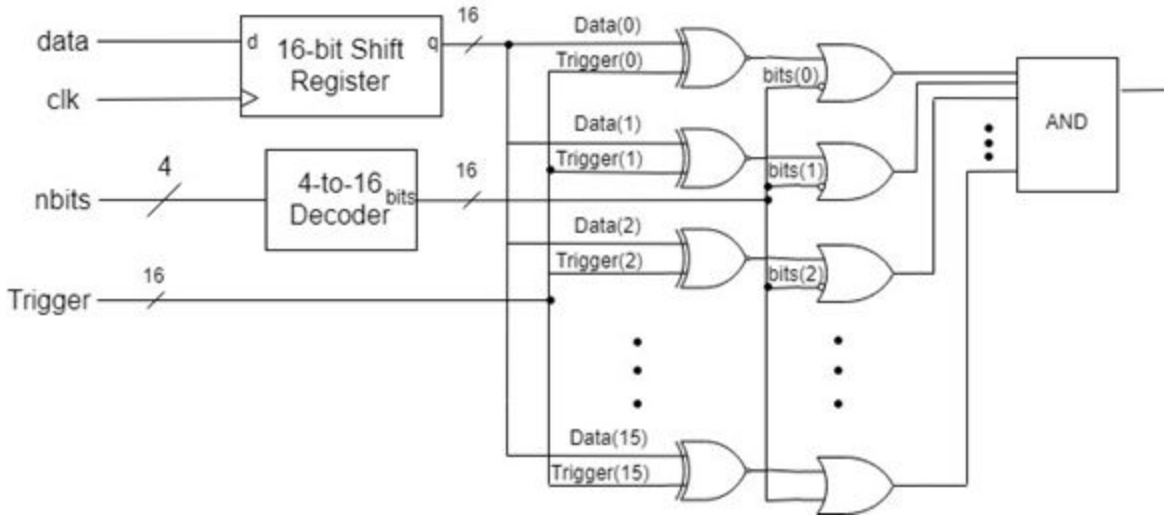


Figure 11: Serial Sequence Detection Block Diagram

5.1.3 Sampler Unit

This module will sample the data, and on a trigger will begin outputting to the FIFO that connects to the Microblaze Softcore. This unit will contain a FIFO buffer on its own that will store a small subset of data and send it out when the trigger is enabled.

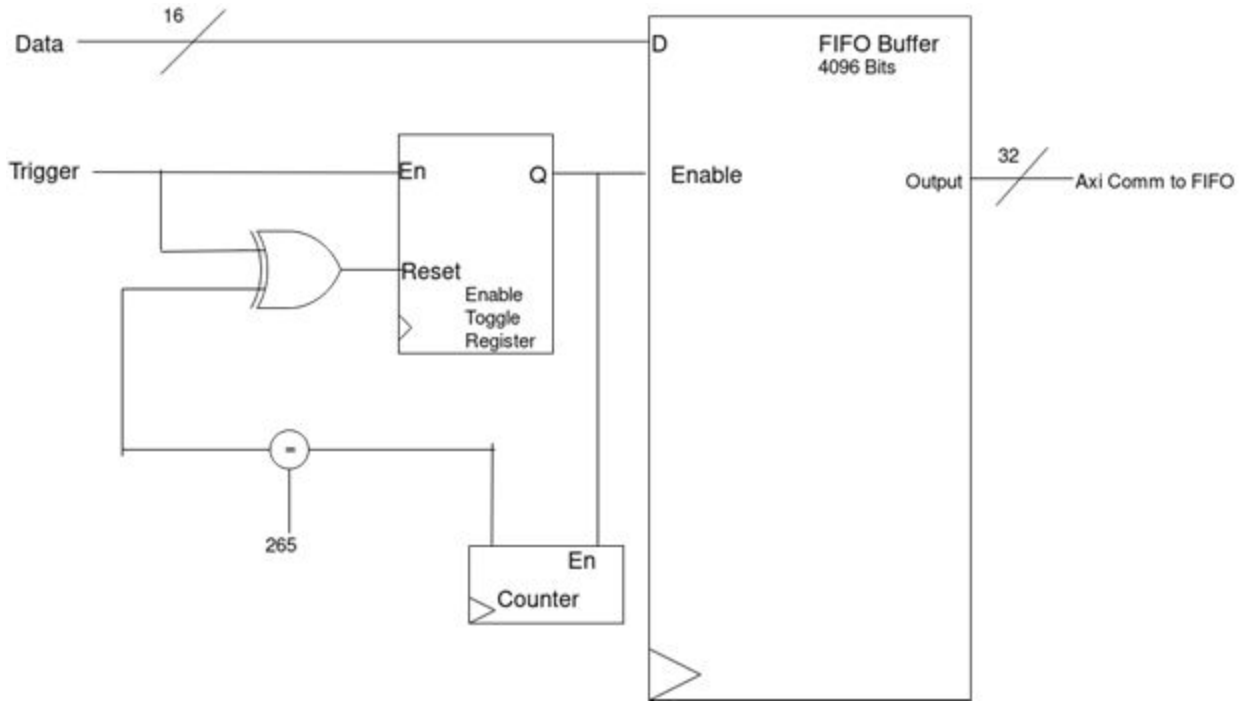


Figure 12: Sampler Block Diagram

5.1.4 Clock Wizard

The clock wizard IP that Xilinx provides for its FPGA's will be used to provide an extensible, re-configurable clock for sampling. This will communicate with the user configuration registers to be configured to the proper sampling clock rate requested.

5.1.5 FIFO

This component will also be from the FIFO IP provided by Xilinx for their FPGA's.

5.1.6 Microblaze Softcore Processor

This is a softcore provided by Xilinx for their FPGA's, like the basys3 board. We looked at different provided softcores, as well as other solutions, and the following decision matrix was formulated to chose from them:

The decision for Microblaze came down its ease of use, and the bitwidth we had access to. Picoblaze was a close second for use but it didnt include enough bitwidth to send data fast enough.

Softcore	Open Source	Easy of Use	Bit Width	Language	Speed	Total
Score	5%	10%	40%	20%	25%	100%
MicroBlaze	0	5	5 (32 bit)	2 (VHDL/Verilog & C/C++)	2 (50MHz)	3.4
OpenFire	1	4	5 (32 bit)	1 (Verilog & C/C++)	1 (25MHz)	2.9
PicoBlaze	0	5	1 (8 bit)	2 (VHDL/Verilog & C/C++)	2 (50MHz)	1.8
MIPS from ECE445	1	2	5 (32 bit)	1 (VHDL/MIPS)	3 (100MHz)	3.2
Full Custom (VHDL)	1	1	5 (N bit)	0 (Custom)	3 (100MHz)	2.9

The softcore will be running a basic algorithm and state machine to send the data over to the PC. It will wait for a trigger from the intermediary FIFO for when it is full, to start grabbing the data and sending it over USB.

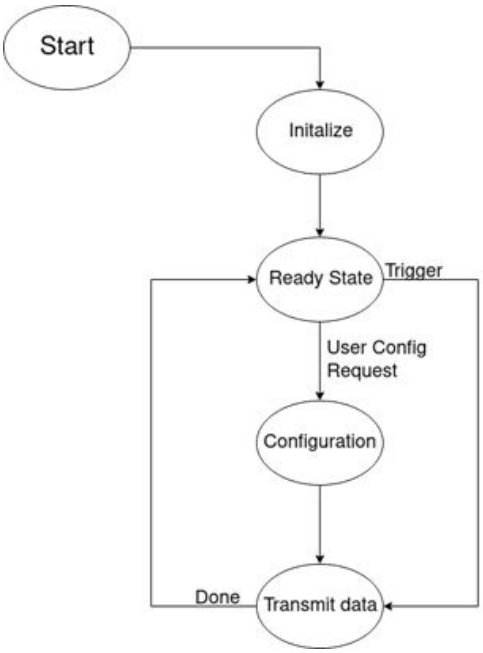


Figure 13: State Diagram of MicroBlaze Soft-Core

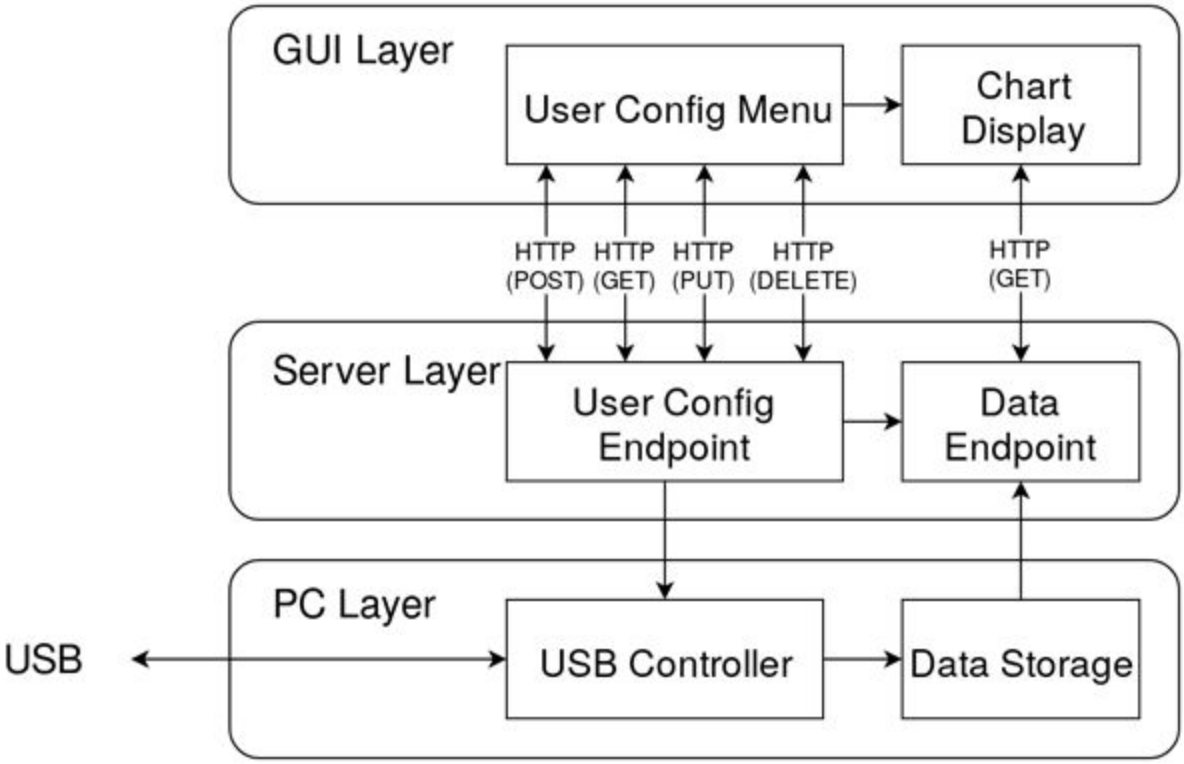
On the Microblaze, the softcore will do a very simple operation. Its primary focus is to wait for one of two triggers, Either from the FIFO signaling it has data to transmit, or from the usb controller for reading data from the PC.

If the FIFO signals that there is data to be transmitted, it goes into a routine of grabbing the data from the FIFO until the FIFO is empty, and sending it over USB to the PC.

If the USB controller reads data from the PC, it is assumed to be the 64 bits of the configuration packet. This is then processed, parsed, and sent over to the user configuration registers over the AXI protocol, so they can be updated.

5.2 Data Interpretation Module

5.2.1 Overall PC Data Flow Chart



This Module is a python server that serves as a middle point for the data that flows between the GUI and the Basys3 Board

It functions as such: Server:

When the server receives a request it determines the endpoint then executes the function associated with that endpoint. When the endpoint is determined to be one of the config endpoints the method is determined. If the method is GET, then the route is parsed for the desired config id, which is queried from the database and loaded, serialized, and returned via the API response. However, if the requested config does not exist the standard 404 NOT FOUND response is returned via the API response. If the method is DELETE the route is parsed for the desired config id which is the queried from the database and deleted returning a 204 NO CONTENT via the API response whether the config existed or not. If the method is POST the request body is

parsed for the desired config id which is the queried from the database. If there is no entry one is created using the data from the request body returning 201 CREATED via the API response. I f there is an entry the API returns a 400 BAD REQUEST via the API response. If the method is PUT the route is parsed for the desired config id which is the queried from the database and loaded. If the entry exists the entry is updated using the data from the request body returning 204 NO CONTENT via the API response. If the entry does not exist the API returns a 400 BAD REQUEST via the API response. When the endpoint is determined to be the data endpoint the serve queries the data storage for all the data and then processes the data according to the config currently marked active. If there is not data the API returns empty data in the shape requested by active config. If the method requested is anything other than GET the 405 METHOD NOT ACCEPTED is returned via the API response.

PC:

The data packing and serial communication protocols are detached from the server and run as separate asynchronous processes. There are two of such processes one for communication of raw data from the USB channel. Another for sending the user config when a config change is detected.

5.2.2 General PC Algorithm Flow

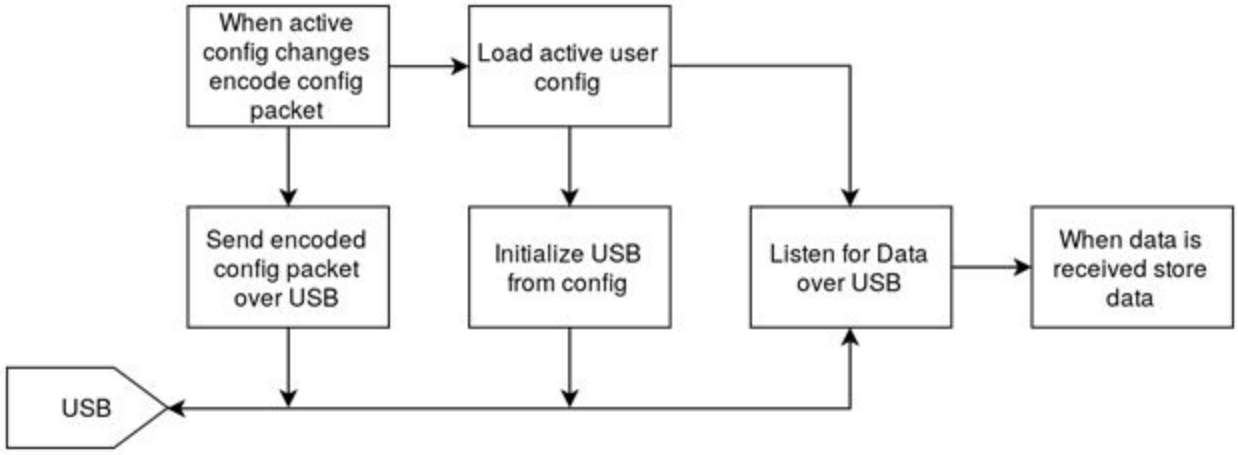


Figure 14: USB Controller Algorithm

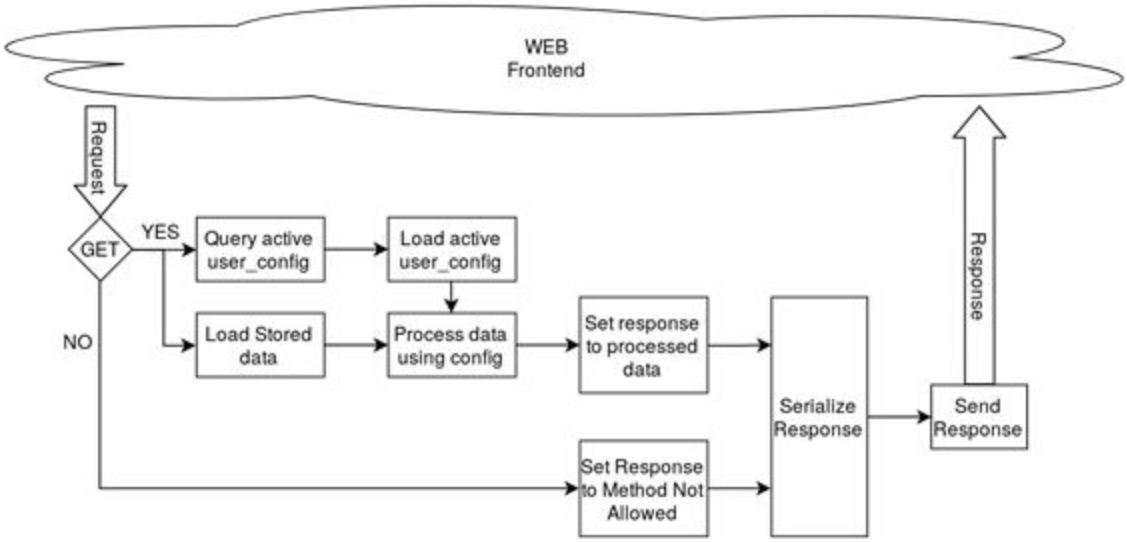


Figure 15: Data Endpoint Algorithm

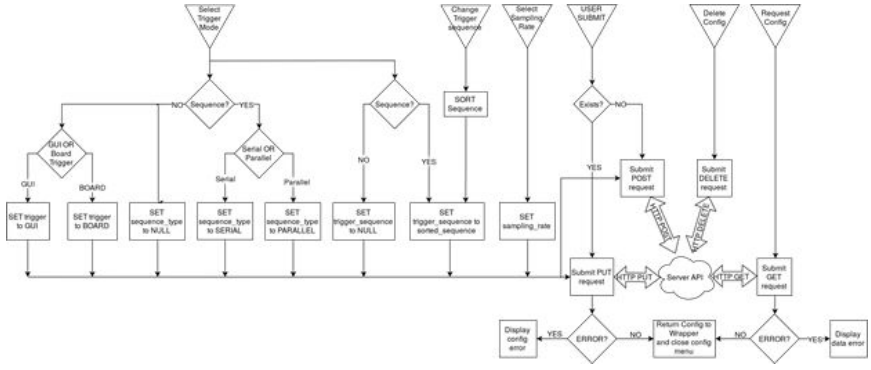


Figure 16: User Configuration Algorithm

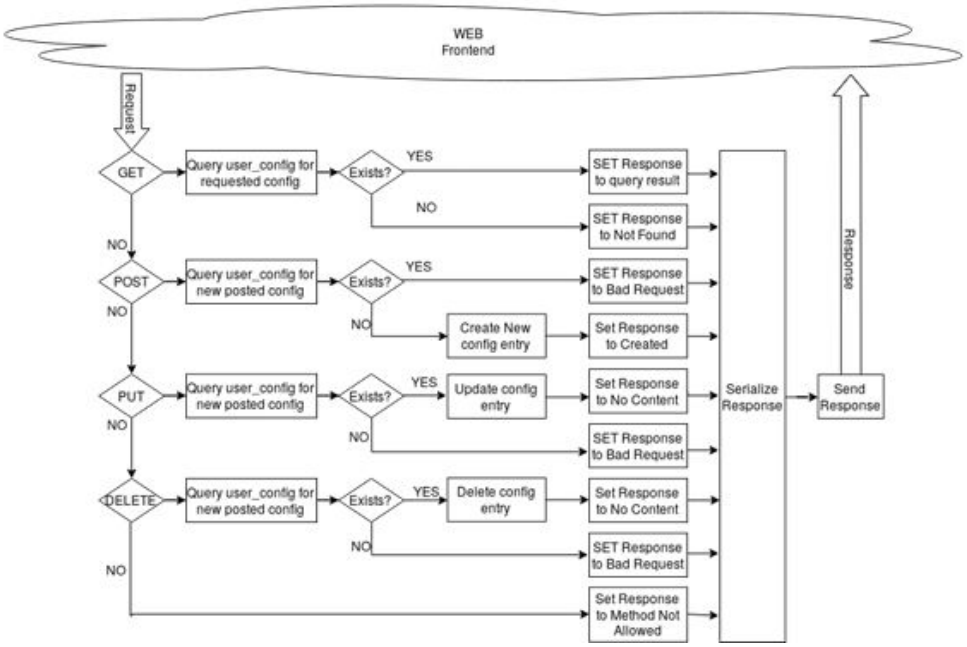


Figure 17: User Configuration Endpoint

5.2.3 I2C and SPI Interpretation:

When channels are selected as being for I2C or SPI communcation channels, these channels are processed in an addition process located on the python interpretation server. This will be shown by displaying the channels matched to I2C or SPI as one line and displaying the data sent, and in case of I2C, the slave address and the data address will be displayed.

it will follow the following algorithms:

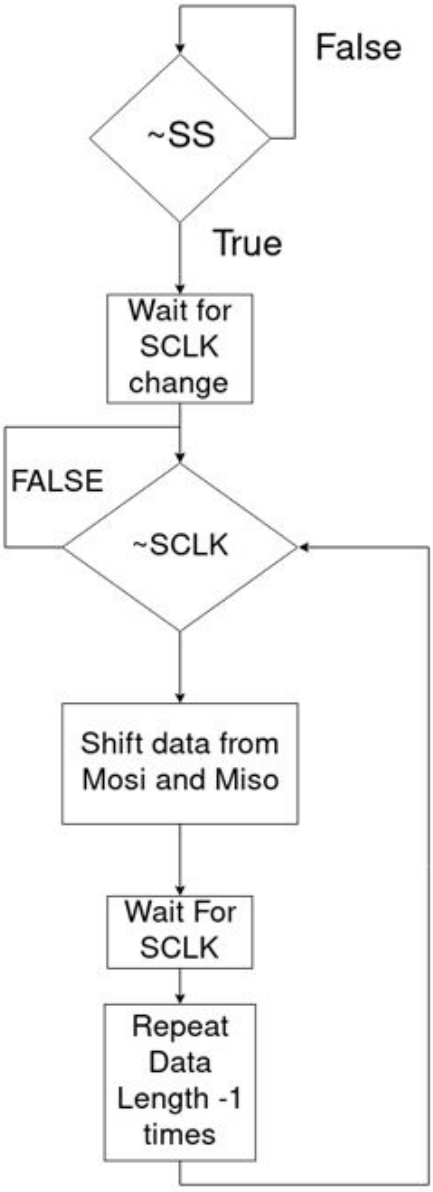


Figure 18: SPI Algorithm

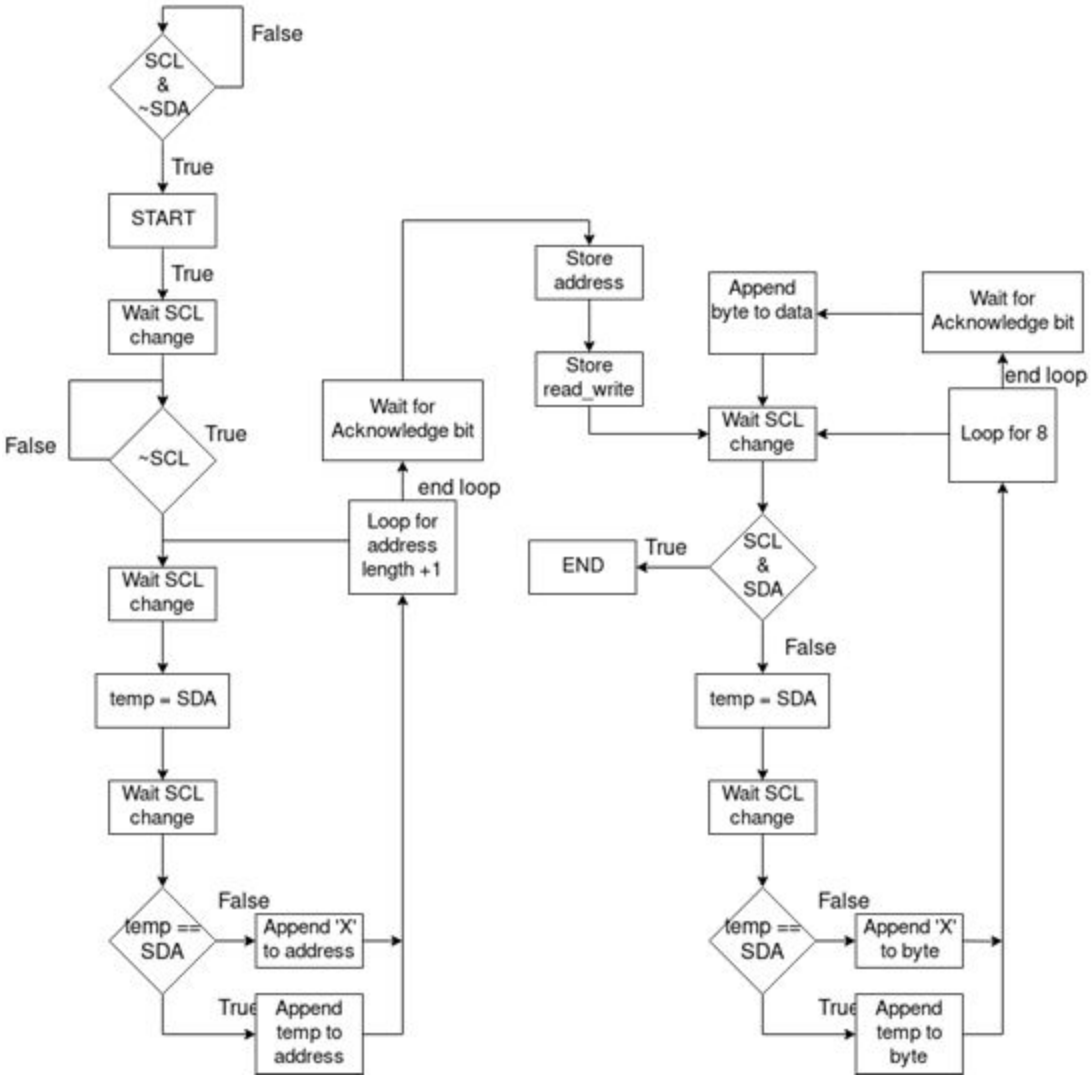


Figure 19: I2C Algorithm

5.3 Data Display Module

This module is a web front-end that takes the data sent to it from the server and processes it and turns it into a graph for interpretation of the user.

It functions as such:

Upon system init the GUI starts with the default view with no data loaded and the default user

config loaded. Upon entering the user has the ability to alter the user config, load stored user configs, and delete user configs. From the user config menu the user is able alter the trigger sequence, select the tigger mode, and select the sampling rate. When the user submits the config changes the frontend submits the appropriate request to either update or create the current config, then close the menu. From the data display the user can toggle auto-refresh or manually refresh the data. When data is received the display logic determines to appearance of each channel based off the data structures in the API response.

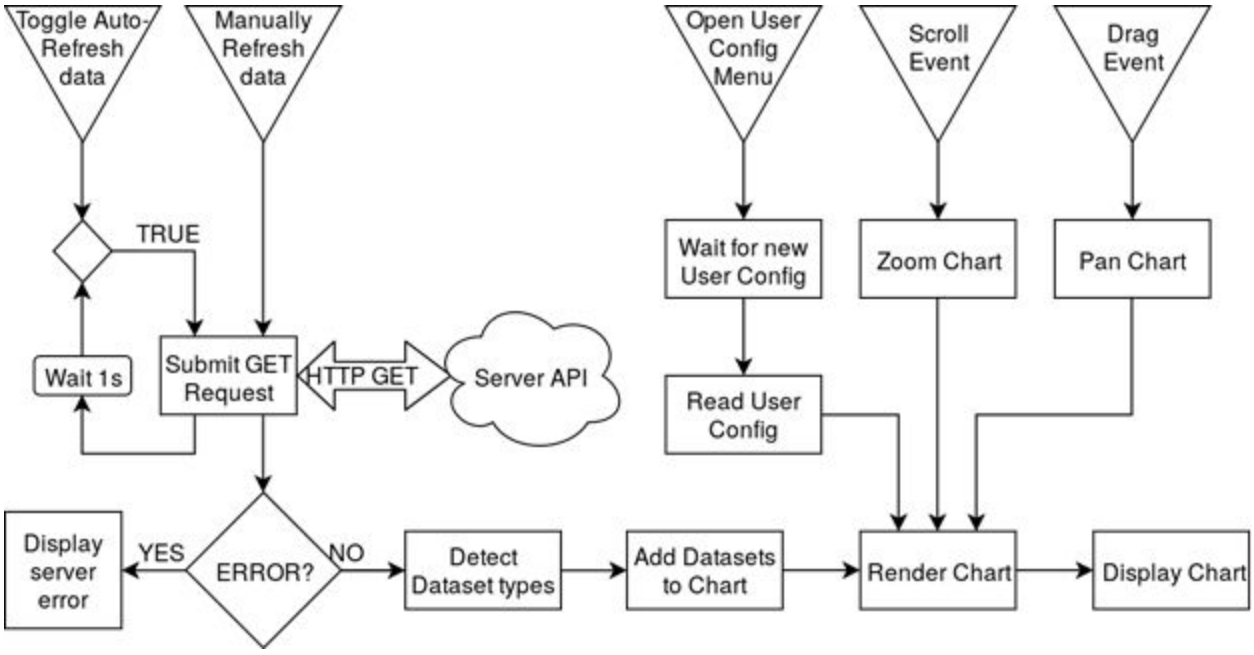


Figure 20: Data Displaying Algorithm

6 Early Prototyping:

6.1 Communcation with USB

So far there has been an effort to ensure some of the functionality is feasible to perform. The first

such was get communication with USB working on the FPGA. So far we have a working Basys3 bitstream that allows a server on one side to echo information that is sent over it from the Basys3 board. The following is a screenshot of the python script printing and echoing back the data:

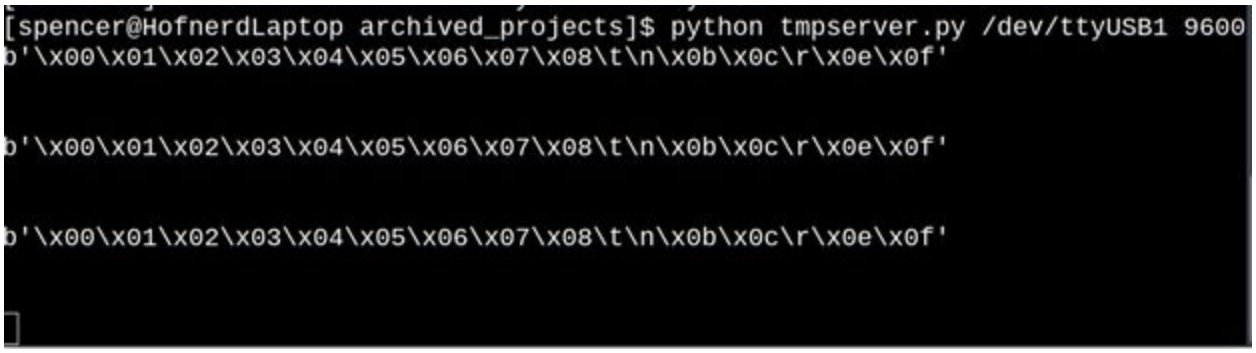


Figure 21: USB Communication example

7 Project Plan:

7.1 Project Plan Schedule

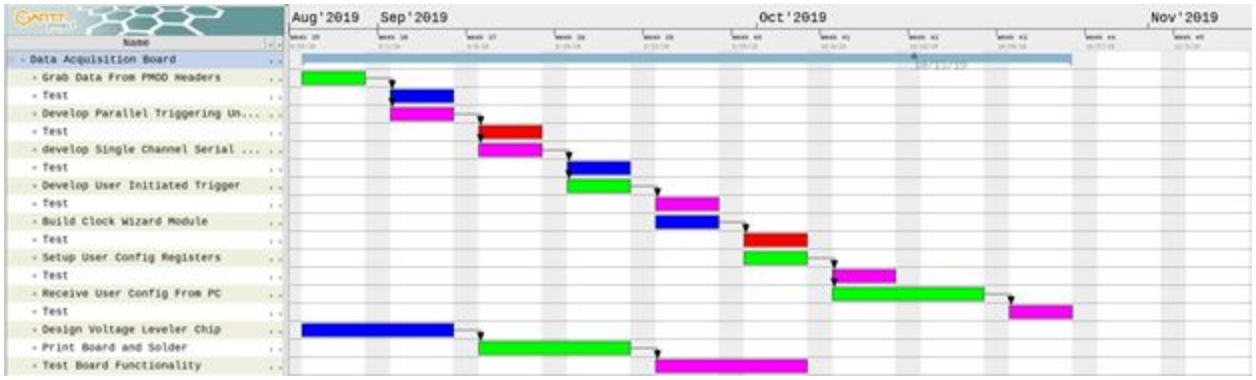


Figure 22: Data Acquisition Board Project Plan Gantt Chart

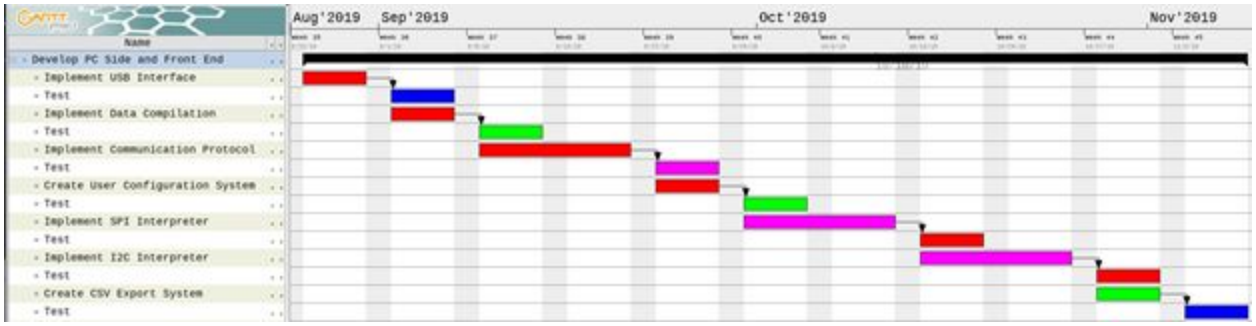


Figure 23: PC Software Gantt Chart

7.2 Experiment Plans:

The following is a list of things that must be tested, and in a general order to ensure proper functionality:

1. Microblaze properly outputs UART Communcation **COMPLETED**
2. Server receives data properly and GUI displays the UART data
3. PMOD headers are read from on a sampling clock with data output to the server functioning
4. Configuration packet changes configuration registers on Basys3 board
5. Clock wizard is dynamically modifiable
6. Parallel trigger works correctly
7. Serial trigger works correctly
8. SPI is parsed correctly9. I2C is parsed correctly
10. Breakout board with voltage leveler works as intended

The design consists of 10 tests:

1. Setup Xilinx Microblaze to operate on a Basys3 FPGA and send packets of data over UARTto a PC. This teset will ensure all components are working correctly and errors

in the system are not due to hardware.

2. Receive data over UART from Basys3 FPGA and display data on the GUI. This test will ensure that the data is correctly read by our backend python server and the GUI properly interfaces with said server.
3. Program a microcontroller to blink an LED at set intervals in a specific pattern. Then we will connect both the logic analyzer. This test will verify that sampling, and data output and input from PMOD headers are working correctly.
4. We will send and process the Configuration packet and test to ensure the registers were properly modified. This will be done in two ways:

(a) 16 bits of the trigger sequence on the built in LED's

(b) Remaining 16 bits will be decoded into 7 segment display and posted on there.

5. Using the sample circuit from test 3, we will vary the sampling frequency and verify that the data is being sampled at the different frequencies.
6. We will connect one basys3 board into another using the PMOD headers. The primary Basys3 Board will function as a data sampler, and the other will be used as a counter. This second board will have a 16 bit counter with its output on the PMOD headers. We will then configure a 16 bit trigger sequence, and test that the sequence the sequence is recognized on the board, and triggers data to be sampled.
7. Using a similar fashion, we will test the serial pattern on one channel functions as intended.
8. We will set up a MSP430 that is programmed to communicate with a SPI enabled peripheral.We will then connect the pins to the Basys3 board, and test that the server

correctly recognizes and displays the SPI protocol

9. Repeat this test, but using I2C rather than SPI
10. Attach the Basys3 board to the designed breakout board and verify that the analyzer and triggering unit still function as intended.

8 References

1. http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/