From: Team Butterworth
The Ohio State University
154 W. 12th Ave.
Columbus, OH 43210

Date: March 2, 2009

Subject: Proposal for Digital Stimulus/Monitor Unit

To: Professor Joanne E. DeGroat
205 Dreese Laboratories
Department of Electrical Engineering
2015 Neil Avenue
Columbus, OH 43210

Dear Professor DeGroat:

Team Butterworth is pleased to submit the attached proposal for the Digital Stimulus/Monitor Unit (DSMU). Team Butterworth is comprised of the top senior students in the Electrical and Computer Engineering Department at The Ohio State University, with a breadth of knowledge perfectly suited for this project. We understand the urgency of completing the DSMU before June. We at Team Butterworth can and will meet all the requirements of this important project.

The students of Team Butterworth are prepared to design a new DSMU, and we look forward to your reply.

Sincerely,

Team Butterworth

Encl: Proposal

Digital Stimulus/Monitor Unit Proposal
Team Butterworth

Due Date: Friday, March 13th

Abstract

This report outlines a proposal for the design and implementation of a digital stimulus/monitor unit (DSMU) that will measure and output digital signals at a rate of up to 5 MHz The design is based around a Spartan 3 FPGA, host computer software, and a USB interface, which will together carry out all of the unit's needed functions. The project will be compartmentalized into these three major components, from the point of view of design and functionality. The Spartan 3 was chosen as the basis of the project due to its high speed and relatively low cost, which make it an attractive option. A BASYS Evaluation Board will be used, complete with this FPGA and a convenient API for USB 2.0 communication. The input or output status of each of the eight FPGA ports will be chosen with tri-state buffers and a select pin, allowing for fast and easy configuration.

USB 2.0 will be used to communicate with the host, with the help of the Digilent Port Communications API. Communication will occur through a custom protocol, based on several different functions dealing with the transmission of data back and forth between the host and FPGA.

The host software will be created in using the Qt 4.4 open-source application development framework, which will permit the use of C++ as the development language. The Qt framework will allow the team to quickly develop GUI applications, through is GUI layout software, Qt Designer. The resulting software will provide the user with a graphical interface, which will allow for the configuration of the ports as inputs or outputs, and will allow the user to specify output waveforms and their frequencies. Both inputs and outputs will be monitored and displayed graphically for the user.

A general plan and schedule for project development are laid out in the report; the project will be fully designed and implemented over the course of the ten-week quarter. The financial aspects of the plan are examined as well – the total cost will be around $90.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Purpose

This document proposes the redesign of an electronic device to interface digital input and output signals to a Device Under Test (DUT). The proposed device, a Digital Stimulus/Monitor Unit (DSMU), is needed to support research thrusts in the area of high frequency digital circuit design, specifically in Professor Degroat's embedded systems lab. This document outlines both the hardware and software development strategies proposed to realize this device.

1.2. Background

The design of high speed digital logic circuitry is often hindered by the designer's ability to debug its operation. The complex dependencies between a myriad of rapidly changing system states makes it difficult to apply traditional approaches such as scoping the system output. Additionally, it is often desirable to decompose digital systems into smaller, easier to manage sub-components, building a complete solution on multiple layers of logical abstraction. This common design methodology creates the need for each sub-component designer to develop test drivers or stubs, imitating the intended behavior of interfaced sub-components. The design of a system to easily provide such a configurable interface would greatly assist in the development of high speed digital systems.

It is desired for the DSMU to interface eight 5V digital I/O signals to a DUT at frequencies of up to 5MHz. The DSMU should be a portable unit, interfaced to a Windows host through a USB 2.0 High Speed connection. Achieving the full 5MHz operation with real-time USB communication presented the most difficulty in previous implementations, and presents the largest area for improvement on this redesign. The host should be able to fully qualify the

operation of each port. This includes the ability to specify its frequency, and direction (input or output). In addition, the host should provide an interface to control the output waveforms. It should support both outputs of a specific duty cycle as well as repeating complex waveforms. The signals sent and received by the Digital Stimulus Monitor Unit should be displayed graphically by the host.

Information on the previous implementation of the DSMU will serve as a starting point for the redesign. The earlier design was based on the use of a PIC microcontroller and achieved an operation frequency of 12 kHz, the main target for specification improvement in the upgrade. The last design improved upon the base specifications by making all eight ports configurable as inputs or outputs, a feature that is hoped to be carried through to the present model. The design made use of a custom circuit board, which is hoped to be replaced with a BASYS project board meant for use with an FPGA and USB 2.0. The first implementation's source code is also available as a basis for implementing a GUI and will be studied for other insights that it may provide. Where the old design works well, such as the way its GUI was laid out, it will be used as a model; where it has room for improvement, as in its inability to display the I/O data, targets for upgrading the design are found.

1.3. Scope

This proposal addresses the development of the DSMU hardware and software systems to interface with a variety of DUTs as well as the budget, resources, and timeline allotted for this project. It does not however include the development of any test drivers, or the imitation of any specific devices. Rather, it aims to address the development of a general interface that makes the implementation of such test drivers a trivial matter.

2. Discussion

2.1. Approach

The approach of Team Butterworth will be based around three main development areas, the FPGA Hardware, Communication Protocol, and Host Software. This overall breakdown is shown in Figure 1.

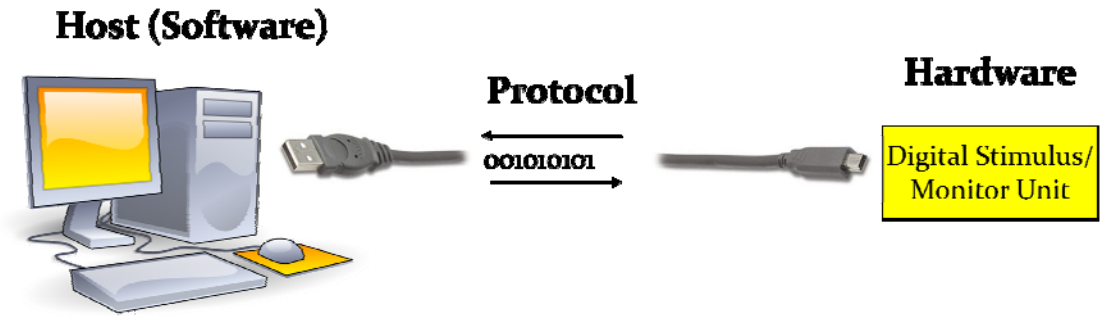

Figure 1: Approach Breakdown[1]

2.1.1. Hardware

It is proposed that the unit be upgraded by including a Xilinx Spartan-3 FPGA to replace the previously installed PIC. This particular family of FPGA will be used because of its high speed characteristics and low cost. With the use of USB 2.0 for transfer between the FPGA and the host, the FPGA will be capable of transferring data at 480 Mbps. This will allow the DSMU to operate at or above the specified 5 MHz frequency, while the original PIC design only operated at a maximum of 12 KHz.

The Spartan-3 FPGA that will be used to implement the DSMU is the XC3S100E-4TQ144C, which offers 240 configurable logic blocks (CLBs) with a maximum 108 user I/O pins. This feature will all 8 channels of the DSMU selectable as an input or output, depending on the user's needs, similar to the configuration of the previous design. This is an improvement on the original design requirement of two programmable channels. This feature will be

[1] Images from www.123macmini.com (USB Cable) and www.clker.com (Computer Clip Art)

accomplished by connecting both of the FPGA's input and output pins for a single data channel together. Two tri-state buffers, with an additional select pin, will be used to ensure that only one of the FPGA's pins will be connected to the banana plug. This sub-circuit is shown in Figure 2.

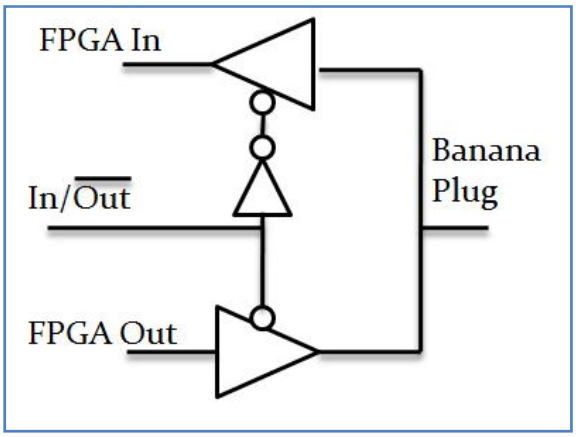

Figure 2: Tri-State Buffer Circuit

Using the tri-state buffers with a select line will ensure that the input and output pins of the FPGA are not shorted together.

The XC3S100E offers 240 CLBs with 100K system gates. This amount of gates will easily allow an output data buffer to be implemented onboard the FPGA if necessary. The use of a buffer on the FPGA will not only save board space, but also lower the total cost of the unit.

To help with meeting the USB 2.0 protocol, an external board will also be used. The BASYS System Board by Digilent provides an interface between a Spartan-3 FPGA and a USB 2.0 port. This board is designed to support the Xilinx XC3S100E-4TQ144C, the FPGA chosen for reasons stated above. The board provides a physical USB interface for the FPGA, reducing the amount of work that needs to be done by the team. The board also provides a large number of pre-wired external inputs and outputs (switches, buttons, and LEDs). As a result, the number of free I/O pins is limited to sixteen. However, the current design goal is to have eight configurable I/O pins, requiring a total of 24 I/O pins from the FPGA. To meet this requirement, the team plans to de-solder some of the existing hardware, such as the switches, to make up the

eight pin difference between the available and required pins. Should this prove to be impossible, the team will make some of the pins dedicated input or output, much like the original design requirement. A dedicated input or output pin only requires one pin from the FPGA. User control can be maximized by having two dedicated inputs (one pin each), two dedicated outputs (one pin each), and four configurable input/output pins (three pins each). This design would use all sixteen of the available pins from the board. However, this configuration will only be used as a last resort if modifications to the board fail, and the plan is for all eight pins to be configurable.

The BASYS Board provides not only the physical wiring for the USB interface, but code for both the FPGA and host computer, which will greatly reduce design and development time. The board simulates an 8-bit parallel connection between the DSMU and host computer, so the state machines on the FPGA will be designed to match this. The provided code is structured for eight data registers and one address register, which are read from or written to according to three signals from the host: WRITE (whether to read or write data), ASTB (address strobe), and DSTB (data strobe). A more in-depth description of the Digilent Parallel Interface Module specification can be found on the Digilent web site, at www.digilentinc.com.

Of the eight data registers in the parallel interface, only three are necessary for communication between the host and DSMU. One will be used for FPGA input, and is reserved for reprogramming information sent by the host computer. One will be used for FPGA output, and is reserved for sending the current value of all eight external pins to the host computer. Finally, one register will serve as a status register to let the host computer know when new data is ready to be sent. A status bit letting the DSMU know when the host computer has read the data is not necessary, because the data strobe will remain low until the data is read.

In order to monitor the USB connection without interrupting data collection, the VHDL for the FPGA will include a USB controller state machine running concurrently with all other processes. The main USB controller machine will monitor the address and data strobes, and will control the flow of data between the parallel interface and the FPGA's eight internal registers as dictated. The USB controller machine will also update some internal status registers keeping track of when output data has been read, and when new input data is available. The machine will also clear the new data status bit for the host controller when data has been read. The VHDL code for implementing the USB controller machine is included with the purchase of the BASYS board.

The data flow to and from the three USB registers will be controlled by two different state machines: a USB input state machine and a USB output state machine.

The USB input state machine will be in a wait state during normal operation. When the USB controller indicated that new information has been received from the host computer, the USB input machine will update internal status registers to inform the other machines that the DSMU is in reprogramming mode. The USB input machine will then collect each byte of data as it is read in, interpret the information, and update the status for each pin. When the machine recognizes the end of the transmission from the host, the USB input machine will change internal status back to normal mode, and return to a waiting state. A state diagram for the machine can be found in Figure 3.

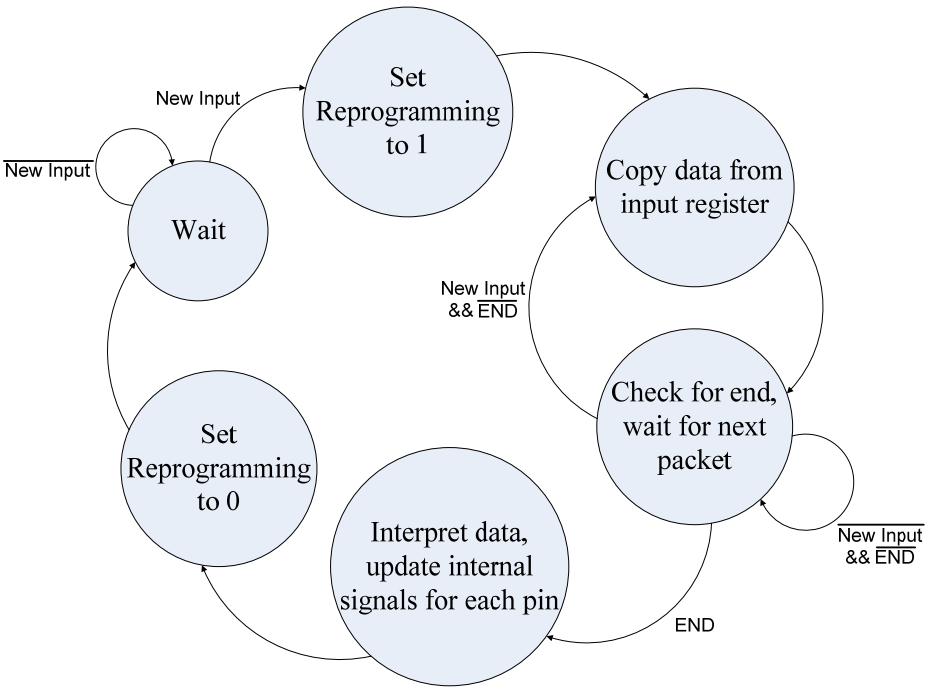

Figure 3: State Diagram for USB Input Machine

The USB output state machine will manage sending data to the host computer, as well as managing the USB status register. The input/output state machines will be using a data collection strobe, based on the oscillator of the BASYS board, to read input values at a uniform frequency. The USB output machine will also use this strobe to determine whether or not there is new data from the external pins. Then, it will check whether the last set of values has been read by the host computer. If so, it will put the new values into the output register and set the new data status bit for the host computer. If not, the machine will add the values to a larger buffer for data to be sent, and continue with collection. If the most data in the output buffer is read while the machine is waiting for the next set of data and there is another byte waiting in the buffer, the USB input machine will move the next value into the output register. A state diagram for the USB output machine can be found in Figure 4.

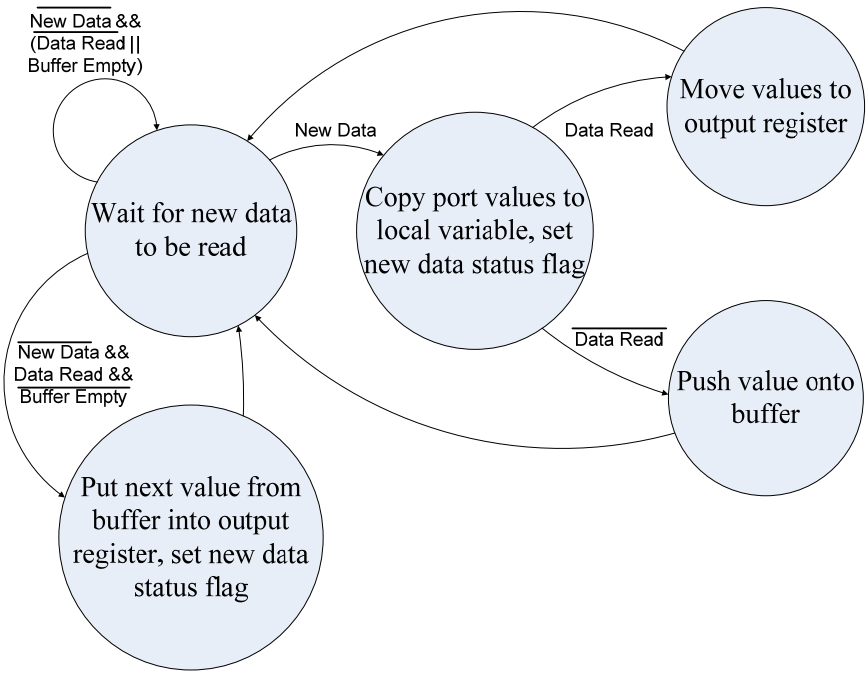

Figure 4: State Diagram for USB Output Machine

Given the difference in speed of data collected by the USMU and the speed of the processor on the host computer, it is not expected to ever reach a buffer overflow. However, should this occur during testing, the USB output machine will set a buffer overflow status bit in the status register for the host computer.

Finally, the status of each of the eight configurable pins will be controlled by a separate but identical generic pin state machine. The generic pin state machine will keep track of whether the pin is an input or output pin, as well as include a waiting state during reprogramming, to ensure that the outputting of data is synchronous. If the pin is in the input state, the state machine will check if the DSMU is in reprogramming mode. If the DSMU is in reprogramming mode, it will transition to the waiting state. Else, the state machine will copy the value supplied by the Device Under Test, at a rate specified by the data collection strobe. If the pin is in the output state, the state machine will first set a counter. This counter tracks the time until the next

value should be written, for waveforms at less than the maximum frequency. The data collection strobe will be used as the maximum frequency. The machine will then enter a waiting state in which the counter is decremented, as well as check whether the DSMU has entered reprogramming mode. If the device is in reprogramming mode, the state machine will enter the reprogramming/wait state. Else, when the counter expires, the new value will be written both to the external pin and the internal value register (for the USB output machine), and the counter will be reset. The state diagram for the generic pin state machine can be found in Figure 5.

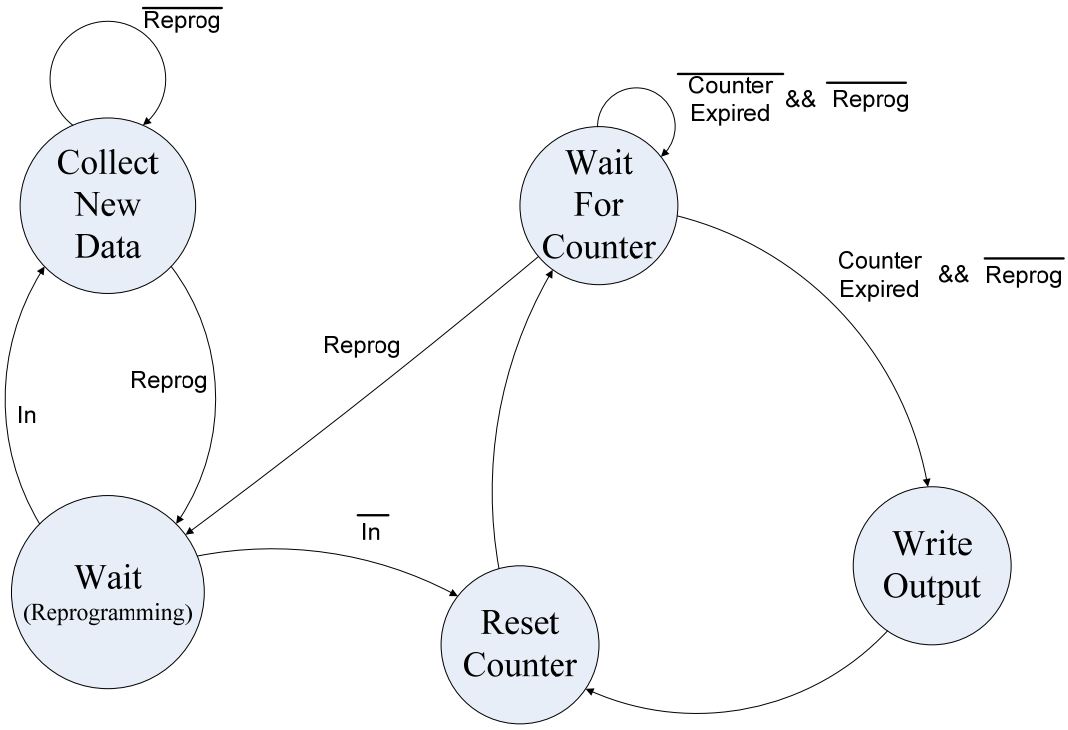

Figure 5: State Diagram for Generic Pin State Machine

2.1.2. Protocol

All of the data that is transmitted between the host and the FPGA will use a custom protocol. The protocol will be based on five major functions. Four of them involve the transfer of data from the host to the FPGA, and will serve to completely reconfigure the device. The last

deals with the transfer of I/O status data back to the host. Below is a list of the five protocol functions.

Configuration Functions

1. Transmission of a signal from the host to the FPGA indicating a change in the status of the ports. This function will prepare the FPGA to receive new data.
2. Transmission of a signal from the host to the FPGA to specify each port as input or output, as well as transmit the intended output frequency for each port. This data is will then be stored on the FPGA.
3. Transmission of a signal from the host to the FPGA representing the waveform to be placed on each output port. The waveforms will be represented by a series of bits, whether the waveform is a complex waveform a PWM signal. Before the actual signals, a header containing the port number and signal length will be passed in order to aid with processing the sequence.
4. Transmission of a signal from the host to the FPGA to signal the synchronized transmission of the output signals.

I/O Status Functions

5. Transmission of a signal from the FPGA to the host containing the logic status of all eight ports in a one-byte packet. These packets will be sent continuously until the user decides to reconfigure the ports.

A figure showing the headers and data sent with each function transmission is shown in Table 1.

Function	Header	Packet	
1	1001	(Prepares FPGA for new configuration)	
2	1010	8-bits of inputs/outputs	8 x 4-bit frequencies
3	1011	1-byte sequence length	Sequence
4	1100	(starts outputs together)	

5		Logic status of all ports in 1 byte

Table 1: Protocol Functions

2.1.3. Software

In order to handle the information being sent between the host and the FPGA, host software must be written. The team will utilize the Qt 4.4 open-source GUI development framework to create multi-threaded real-time software to interface with the FPGA. This framework will allow for the development of GUIs using C++ as the development language. The Qt designer software has been used to create a mock up of the graphical interface. The interface will allow users to configure I/O ports and view their status in real-time. An image of the interface to configure a port is shown in Figure 6.

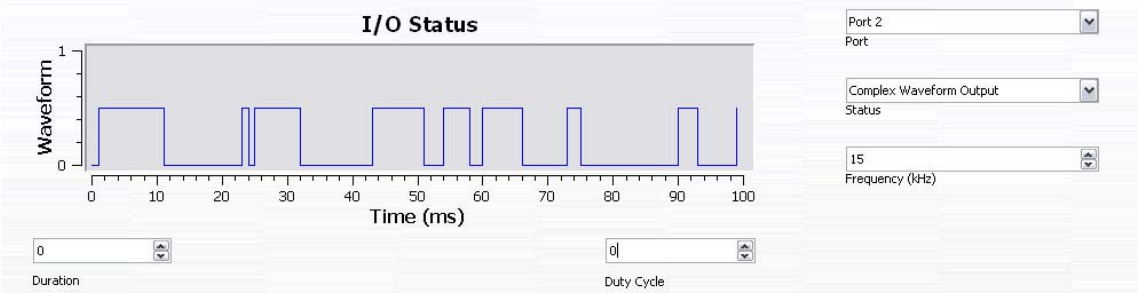

Figure 6: Configuration GUI

It is proposed that this interface will support graphical waveform configuration approaches found in commercial software such as Xilinx, where the user may toggle the waveform by clicking on its image. The GUI will provide "tab" functionality, which will allow the developers to separate an I/O status tab from this port configuration tab. The I/O status tab is shown on the following page in Figure 7. It will allow for graphical representations of the input and output data to be displayed on the screen in a readable manner and color will indicate if a port is input or output. Additionally, this window will allow the user to configure a specific port as a trigger port, to allow the user to capture and view data at a critical time of operation. Both

the configurations and I/O status will be able to be restored and saved via a configuration or status file.

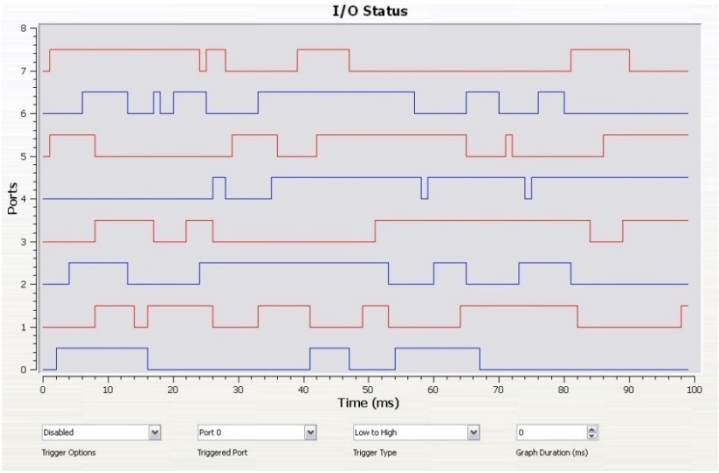

Figure 7: I/O Status GUI

Behind these graphical interfaces, a GUI Update Thread will run to periodically update the images on the screen. This will allow the user can to get the feedback he or she wants in near-real time, without bogging down the host by graphing after every status update is received. In addition to this GUI Update Thread, a variety of threads will be run in order to coordinate status and configuration transfers to and from the FPGA. Figure 8 shows the interaction between these threads.

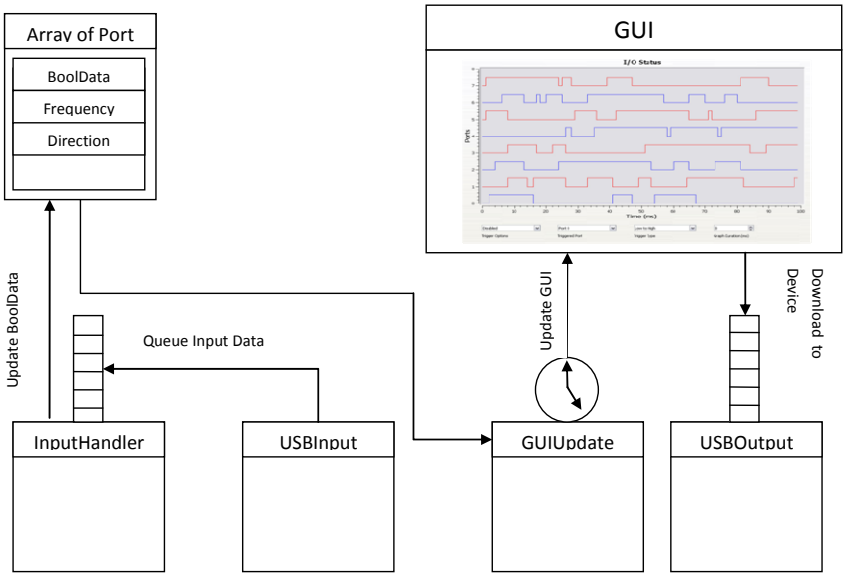

Figure 8: Thread Interaction

Aside from the GUI Update Thread there are six main classes to support the operation of the DSMU software. The port class, shown in the upper right, represents of a single port of the DSMU. It contains all necessary configuration information including direction, frequency, and waveform. It also contains a BoolData object, which stores the I/O status over time. The BoolData class will generally be used to store real-time Boolean data for a specific duration. It will allow stored data to come from a trigger, or to be non-triggered. As shown, this BoolData ultimately will be used by the GUI Update Thread in order to update the I/O status graph. All other objects shown on the diagram are Thread objects to support the USB 2.0 interface.

The USBInput thread class will constantly retrieve I/O status data from the FPGA. This will be accomplished through the use of the Digilent Port Communications API. This API allows the user to read from and write to a predetermined set of registers on the Spartan-3 FPGA. As specified in the Hardware Approach section of this document, communication from the FPGA to the Host will utilize a status and a data register. Thus, this USB Input thread, will repeatedly read

the FPGA status. If there is data in its output buffer, it will read that data until the buffer is empty. Otherwise it will sleep for a short amount of time to be determined through experimentation. This approach will allow the host to refrain from busy polling, while not missing data through the use of the FPGA buffer. To ensure that no data is lost, the USB Input Data class will not parse any information, rather it will pass it off to an Input Handler class to perform any necessary processing.

The USB Input Handler class will act as the consumer for the USB Input Thread. At the current time, it is proposed that this thread will simply use the I/O status bytes provided to update the global port representation as shown in the diagram. The structure of this producer consumer relationship between the USBInput and Input Handler threads leaves the opportunity for more complicated FPGA to Host data streams to be used.

The USBOutput Thread will send information to the FPGA at the user's request, as specified by the GUI interaction. This thread will function to download new configuration requests to the FPGA following the protocol defined above in the Protocol Approach section. The Digilent Port Communications API will once again be used to interface to the FPGA. As defined in the hardware section, a single byte length register will be used to store any Host to FPGA data. Thus, this thread will break down configuration packets into single bytes, repeatedly using the data transfer API calls to configure the device.

More detailed specifications for all of the classes, including all public and private member and functions, as well as their functionality can be found in Appendix A.

2.2. Result

Team Butterworth proposes the aforementioned design strategies because they provide distinct advantages in upgrading the DSMU.

With respect to hardware, the Spartan-3 FPGA will operate at much higher clock speeds than the original PIC design. Because of this, the FPGA will be capable of 5 MHz frequencies for all 8 channels. The use of a USB 2.0 interface will also ensure that the FPGA can download new waveforms while in operation with minimal delay. The original design required 3 output channels, 3 input channels, and 2 programmable channels. With the FPGA approach, the team should be able to implement 8 programmable channels. This feature reduces the limitations of the DSMU.

The speed of the FPGA would not help meet data frequency specifications if the data processing took many clock cycles. Therefore, most of the data processing, such as triggers, will take place on the host PC. This will reduce the number of clock cycles the FPGA requires to handle the status of each I/O port. The host software will also allow the user to view and configure data with a GUI that is easy to use. The data will be displayed on the host machine in clear manner. The GUI will also offer the ability to save graphs and port configurations for the purpose of reviewing data and the waveforms sent to the DUT. The user will also be able to open previously saved port configurations, which will allow him or her to run the same tests multiple times without having to remember the port configurations. A print functionality for the graphs will also be included.

The classes being implemented by the team will run as separate threads so that the GUI will remain responsive to the user's inputs. The user will then be able to configure ports, save

data, and print graphs while the DSMU is transmitting data. The finished overall system with these features will make a positive addition to the embedded systems laboratory at Ohio State.

2.3. Statement of Work

A full breakdown of each task can be found in the Timeline in Appendix B. This section will provide a broad overview of the main tasks identified by the group. The following tasks will be performed to realize the physical DSMU device:

Task 1: The BASYS development board will be ordered along with the appropriate connectors to interface I/O. Basic tests will be performed using the BASYS's on-board I/O.

Task 2: The Digilent USB 2.0 communication API will be tested to verify it operates as expected.

Task 3: The USB input and output state machines will be written in VHDL and verified on the FPGA.

Task 4: The generic pin state machine will be written in VHDL and verified on the FPGA.

Task 5: Simulations will be run with one channel set as an input and another set as an output, with the two channels directly connected to test the devices sub-component interfaces.

Through this process, the protocol development personnel will assist to ensure protocol standards are followed by the host and FPGA software and firmware. The following tasks will be performed to fulfill the protocol portion of the DSMU project:

Task 1: Test the FPGA to Host communication of I/O status

Task 2: Test the Host to FPGA communication of device configuration

The final section of the project, the software, will utilize the protocol personnel for assistance with the USB interface. The following tasks will be performed by software personnel to fulfill the host portion of the DSMU project:

Task 1: Create a GUI shell with no functional parts in order to map out what functions will be needed. (Already completed)

Task 2: Implement a generic port class to hold all port data.

Task 3: Develop software for real-time graphing of Boolean data.

Task 4: Develop code to support USB communication through separate USB output and USB input threads.

Task 5: Test the host software.

3. Resources

3.1. Personnel

Team Butterworth will design and build the DSMU as well as write the host software. Additionally, previous team members and instructions staff will be consulted to ensure the best possible design for the DSMU. Based on the level of expertise in different fields, the members of Team Butterworth will separate into three different teams. The software team will develop the host software, and will consist of Pat Wensing and Robert Richards since they have the strongest software background; the protocol team, consisting of Bill Isaacs and Steve Schairbaum will specify the protocol for transmissions between the host and the DSMU, as well as aid in constructing the unit physically; the hardware team will deal with programming the FPGA using VHDL and be made up of Hannah Driscoll and John Defrain, since hardware is their specialty.

3.2. Facilities / Equipment

The laboratories of the Electrical and Computer Engineering Department at The Ohio State University include adequate space for this effort. Team Butterworth will also use their personal computers in order to facilitate the design process. The specific computer equipment required includes the following:

- Two PC Workstations
    - Microsoft Windows Workstations (XP or Vista)
    - QT Designer 4.4 Open Source
    - G++ Compiler
    - Xilinx Project Navigator (Version 10.1 or later)
    - One USB 2.0 Port

4. Costs

4.1. Financial

Team Butterworth's estimated costs for the project are expected to be below the project budget of $100. The BASYS Spartan-3 board, with an academic discount, will cost $60.00. The external circuitry to interface the BASYS board with the banana plugs will be constructed on perforated circuit board to save costs, and will cost approximately $10.00 for the board, components, wire, and solder. Additionally, the connectors for the banana plugs will cost approximately $13.00, and the project box to house the DSMU will cost $7.00. Therefore, the total expected project cost is $90.00. The remaining $10.00 of the $100 budget will go to

miscellaneous unforeseen costs. A visual breakdown of the construction costs can be seen in Figure 9.

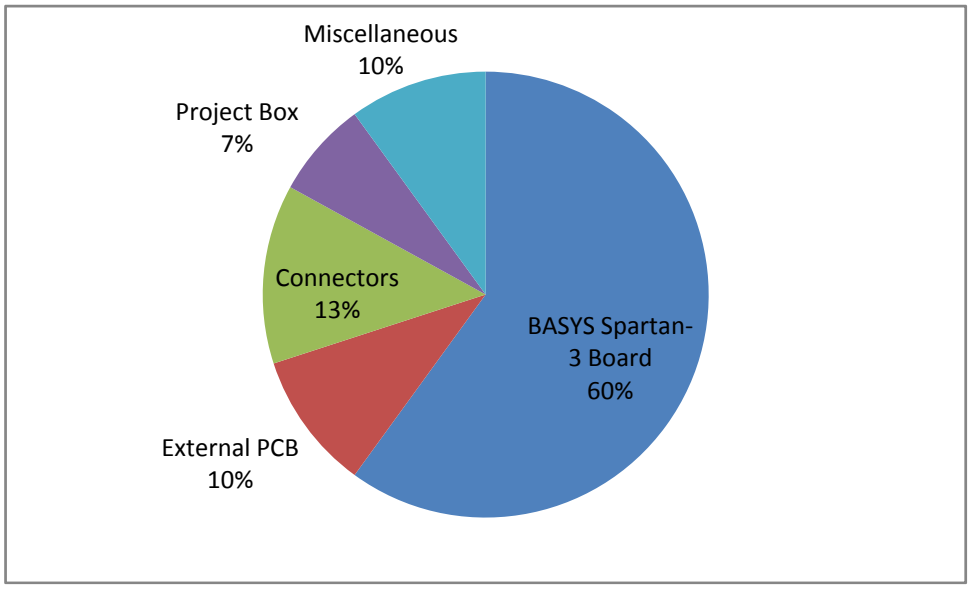

Figure 9: Breakdown of Construction Costs

Time and budget permitting, Team Butterworth may have a PCB for the external circuitry manufactured by Express PCB, instead of using perforated circuit board, at approximately $20.00 per board.

4.2. Timeline

A full timeline for this project can be found in Appendix B, detailing day by day activities. The timeline is broken up into Initial Software Development, Initial FPGA Development, Electrical/Physical Development, and Complete System Testing and Refinement. Table 2 details week by week activities. Primary development efforts will focus on the development of the physical interconnections to the BASYS development board, and the USB interface between the host and FPGA. Concurrent efforts will implement the host software and FPGA firmware for this communication. The easier task, providing data from the FPGA to the

host, will initially be faced due to the simplicity of the protocol for transfers in this direction. This milestone should be reached by April 10th. During the following week, transfers of data from the host to the FPGA, including all device configuration transmission protocols, will be implemented and tested. Thus, through these goals, the USB interface should be functional by the third week of the quarter, ending on April 17th.

After completing this core functionality, the host and FPGA development personnel will take one week to focus on development that indirectly utilizes the USB interconnection. For the host software, this will entail development of graphing capabilities. For the FPGA, this will entail implementing state machines for the sampling and output of I/O data. By focusing on the USB implementation first, the team will be able to focus on the most difficult problem by itself, prior to adding any additional complexity to the system. Additionally, since the USB functionality is most likely to face design modifications, developing a stable set of core functionality early in the project will benefit the other areas that rely on a stable interface.

Week	Task
1	USB Input Software, Hardware Familiarity, Physical Wiring
2	USB FPGA to Host Testing/Development, Physical Construction
3	USB Host to FPGA Development/Testing
4	FPGA I/O Handling Testing/Development, Host Graphing
5	Host System Integration Testing, FPGA System Integration Testing
6	Host System Integration Testing, FPGA System Integration Testing
7	Complete System Testing and Refinement
8	Complete System Testing and Refinement, Final Documentation
9	Complete System Testing and Refinement, Final Documentation
10	Final Documentation

Table 2: Weekly Task Breakdown

5. Conclusion

5.1. Summary

In this document Team Butterworth has proposed a redesign of the Digital Stimulus Monitor Unit. This redesign is based up a Spartan-3 FPGA as part of a BASYS evaluation board which communicates with a Windows host over a USB 2.0 interface. Background on the previous DSMU, as well as motivation for producing a working unit, were provided at the start of this document. A discussion of the approach was provided, detailing the specific design decisions that will motivate implementation. A time and budget breakdown has proposed a solution that is capable of being produced both on time and under budget.

In summary, the proposed DSMU redesign will provide an affordable solution to the testing and debugging of high speed digital circuits. Through the integration of an FPGA, the solution aims to make dramatic frequency specification improvements over a previous PIC microcontroller based solution. The use of the Qt development framework for the host, coupled with the Digilent USB 2.0 communications API will allow design to be focused on higher level tasks, accelerating the overall development time for this project. Through these development decisions, a working prototype will be created over the course of a ten week build cycle that meets all desired specifications.

6. Contact

For more information regarding this proposal please contact Team Butterworth through its team leader, Robert Richards, at richards.838@osu.edu.

7. Bibliography

Basys System Board. Digilent, Inc. 01 Mar. 2009 <http://www.digilentinc.com/Products/Detail.cfm?Prod=BASYS>.

Digilent Parallel Interface Module Reference Manual. 10 Aug. 2004. Digilent, Inc. 01 Mar. 2009 <http://digilentinc.com/Data/Products/ADEPT/DpimRef%20programmers%20manual.pdf>.

DIGITAL STIMULUS MONITOR UNIT. June 2208. The Ohio State University. 01 Mar. 2009 <http://www.ece.osu.edu/~degroat/ECE582/DigPulseRespMonINFO/Documentation/DSMU%20FINAL%20DOC.docx>.

DPCUTIL Programmer's Reference Manual. 3 June 2005. Digilent, Inc. 01 Mar. 2009 <http://www.digilentinc.com/Data/Products/ADEPT/DPCUTIL%20Programmers%20%20Reference%20Manual.pdf>.

Extended Spartan-3A FPGA Family Overview. 31 July 2008. Xilinx, Inc. 01 Mar. 2009 <http://www.xilinx.com/support/documentation/data_sheets/ds706.pdf>.

Qt - a cross-platform application and UI development framework. Nokia Corporation. 01 Mar. 2009 <http://www.qtsoftware.com/products/appdev/platform/qt-for-windows>.

Appendix A: Host Software Source Code and Documentation

```
/**
   \brief Class for storing real time Boolean data for a specific duration
   The BoolData class is used for storing data over a given time period.
   The class supports storing both triggered and non-triggered data. For non-triggered data, each
time that a (bool,time) pair is added, all old (bool,time) pairs outside of the given time period
are removed from the data set. For triggered data, only data before the trigger time can expire,
and data occurring after duration + trigger time is not added to the structure.

*/
class BoolData : public QwtData
{
   public:
        /*!\specDetails
            \brief Constructor for BoolData Class
            \ensures
                The new data object with have a maxTime of 1 second, triggers disabled, a trigger
                time of 0 seconds, and no bool/time values.
        **/
        BoolData();

        /*!\specDetails
             \brief Returns the number of (bool,time) pairs in the data set
             \ensures The size of the data set will be returned
        **/
        size_t size() const;

        /*!\specDetails
          \brief Returns the i-th time value associated with the data set
          \return  The i-th time value associated with the data set
          \param[in] i the time value to return. i=0 is the oldest time value
          \requires     0 <= i < the size of the data set
        **/
        virtual double x(size_t i) const;

        /*!\specDetails
            \brief Returns the i-th bool value associated with the data set
            \return  The i-th bool value associated with the data set (0 for false, 1 for true)
            \param[in] i the bool value to return. i=0 is the oldest bool value
            \requires     0 <= i < the size of the data set
        **/
        virtual double y(size_t i) const;

        /*!\specDetails
             \brief Creates a copy of the distinguished parameter
             \return A pointer to the copy
        **/
        virtual QwtData * copy() const;

        /*!\specDetails
            \brief Sets the duration to save old Boolean data before removing it from the Boolean
                       data vectors
            \param[in] m the duration to save old data (in seconds)
            \ensures maxTime = m
        **/
        void setMaxTime(double m);

        /*!\specDetails
            \brief Adds a time/bool pair to the data set
            \param[in] x the time value for the time/bool pair
            \param[in] y the bool value for the time/bool pair
            \requires x > the time value of the most recently added pair (i.e data is added with
                       increasing time values)
            \ensures For non-triggered, the (x,y) pair will be added to the data set, and all
                    expired data will be removed.
                    For triggered, the data will only be added if the time value is less than
                    trigger time + maxTime.
        **/
        void addData(bool x,double y);

        /*!\specDetails
            \brief Sets the trigger time for the data set
```

```
            \param[in] t the time at which the trigger occured
            \requires t >=0
            \ensures triggerTime = t
        **/
        void setTriggerTime(double t);

        /*!\specDetails
            \brief Enables the trigger for the data set
            \ensures triggerActive=true
        **/
        void enableTrigger();

        /*!\specDetails
            \brief Disabled the trigger for the data set
            \ensures triggerActive=false
        **/
        void disableTrigger();
    private:
        deque<bool>   values; ///< Vector of bool values
        deque<double> times;  ///< Vector of time values
        double maxTime;          ///< Maximum time between the first and last time value (i.e.
                                      the duration to save old data)
        double triggerTime;      ///< Time of trigger, (i.e. minimum time value for data during
                                          trigger mode)
        bool triggerActive;      ///< Boolean to store if trigger mode is active
};

/**
   \brief Thread class for parsing all data recieved by the USB interface.

The Input Handler class functions to allow the USB input class to focus
solely on keeping the USB buffer empty, while offloading all
packet parsing to this thread. This helps to ensure that no data packets
are lost during an attempt to parse them. The USBInput class
feeds Input data to this class, while a wait condition signals this
thread that new data is available. The wait conditions prevent busy waiting,
and allow this thread to sleep while there is nothing to process.
*/
class InputHandler : public QThread
{
   Q_OBJECT
      public:

        /*!\specDetails
            \brief Constructor for Input Handler Class
            \ensures
                stopped = false
        **/
        InputHandler();

        /*!\specDetails
            \brief Request for thread to halt
            \requires
                Thread is running
            \ensures
                stopped = true
        **/
        void stop();
        /*!\specDetails
            \brief Adds data to the processing queue and wakes the thread if it is sleeping
            \param[in] i pointer to the USB input data to be parsed
            \ensures InputData is added to the dataIn processing queue
                    Thread will be woken up if it is sleeping with the newData condition
        **/
        void addData(InputData * i);
```

```
    protected:

        /*!\specDetails
                  \brief Thread code for the InputHandler class
            \ensures The thread will begin running, continually parsing all data in the dataIn
                       queue. The thread will continue to parse data until the stopped flag has
                       been set.
        **/
        void run();
    private:

        QWaitCondition newData; ///< Used to wake thread during periods of previous inactivity
        deque<InputData *> dataIn; ///< Queue of data waiting to be parsed
        volatile bool stopped;  ///< Holds whether a request to stop has been issued
};

 /**
   \brief Class representing a port on the Digital Stimulus/Monitor Device

   The Port class is used for storing data about a specific port.
   Data stored in the structure includes: The type of port (Disabled, Input, Complex Output
   Waveform, Duty Cycle Output Waveform).
   The InputHandler class will modify a given Port to update its condition if it is an input
   port, and the USBOutput class will
   read the data in all Ports then deliver the information across the USB to the device

*/

 class Port
 {
     public:
         /**
               \brief Enumeration of the types of ports
               */


        enum PortType
         {
             DISABLED,
             INPUT,
             OUTPUT_WAVEFORM,
             OUTPUT_DUTY_CYCLE
         };

        /*!\specDetails
            \brief Constructor for Port Class
            \ensures
                freq = 0, type = DISABLED, mutex is unlocked
        **/

         Port();

        /*!\specDetails
            \brief Sets the frequency of the waveform on the port
            \param[in] f the integer denoting the new frequency of the waveform
            \requires     0<= f <= 5,000,000 (5 MHz)
        **/

         void setFreq(int f);

        /*!\specDetails
            \brief Returns frequency of the waveform on the port
            \return  The frequecy of the waveform on the port
        **/

         int getFreq();

        /*!\specDetails
            \brief Sets the port's type
            \param[in] t the enumerated integer denoting the port's new type
            \requires     0 (DISABLED) <= t < 3 (OUTPUT_DUTY_CYCLE)
```

```
        **/

         void setType(PortType t);

                 /*!\specDetails
            \brief Returns the port's type
            \return  The port's type (enumerated integer)
        **/

         PortType getType();

         BoolData outputWaveform;  ///< public BoolData class containing information about the
                                       port's output waveform
         BoolData inputWaveform; ///< public BoolData class containing information about the
                                     port's input waveform

         QMutex mutex;  ///< for controlling when processes can access a certain port

    private:
         int freq;  ///< the frequency of the signal
         PortType type; ///< enumerated integer for the port type
};

/**
  \brief Class running as a thread for outputting data to the device

  The USBOutput thread will be used whenever the user wants to download new data to the Digital
  Stimulus/Monitor Unit.
  The thread will be waiting for any command to come in, then execute the proper functions
  depending on whether the command
  coming into the thread was the DOWNLOAD command or the STOP command.  The thread will only be
  active when it is downloading.
  It will be inactive otherwise.

*/


class USBOutput : public QThread
{
    Q_OBJECT

    public:

        /*!\specDetails
            \brief Constructor for USBOutput Class
            \ensures
                commandMutex is unlocked, commands is empty, newCommand is waiting to run
        **/

         USBOutput();

        /*!\specDetails
            \brief Function that will download all ports to the device via USB
            \requires
                Thread is running
            \ensures
                newCommand = DOWNLOAD, commands[END_OF_QUEUE] = newcommand, where END_OF_QUEUE is
                the end fartherst away from the end currently being executed
        **/

         void DownloadToDevice();

        /*!\specDetails
            \brief Request for thread to halt
            \requires
                Thread is running
            \ensures
                newcommand = STOP
        **/

         void stop();
```

```
    protected:

        /*!\specDetails
            \brief Thread code for the USBOutput class
                       \ensures The thread will begin running, waiting for commands to come in to
                        the commands queue. The thread will continue to run until the stopped
                       flag has been set.

        **/

         void run();

    private:

        /**
               \brief Enumeration of the commands to be handled by USBOutput
               */
         enum OutputCommand  ///< enumerated integer representing the command requested of the
thread
         {
             DOWNLOAD,
             STOP
         };

         QMutex commandMutex; ///< Mutex for controlling when processes can access the thread
         volatile deque<OutputCommand> commands; ///< Double ended queue of enumerated integers
                                                    representing commands requested of the
                                                    thread
         QWaitCondition newCommand; ///< Used to wake thread during periods of previous
                                              inactivity
 };

  /**
   \brief Class running as a thread for reading data being input from the device

   The USBInput thread will be used to read data coming in from the Digital Stimulus/Monitor
   Unit.  It will constantly be
   reading the USB connection to see if there is new data, as long as the thread has not been
   stopped

*/

 class USBInput : public QThread
 {
     Q_OBJECT

     public:

         /*!\specDetails
             \brief Constructor for USBInput Class
             \ensures
                 stopped = false
         **/

         USBInput();

        /*!\specDetails
             \brief Request for thread to halt
             \requires
                 Thread is running
             \ensures
                 stopped = true
         **/

         void stop();

     protected:

         /*!\specDetails
             \brief Thread code for the USBInput class
```

```
            \ensures The thread will begin running, constantly checking for inputs to come in
               from the device. The thread will continue to run until the stopped flag has been
               set.
        **/

         void run();

     private:
         volatile bool stopped; ///< Boolean representing whether the thread is stopped or not
 };

  /**
   \brief Class running as a thread for updating the GUI

   The GUIUpdate thread will be used to update the GUI.  The thread will periodically update the
   GUI at a certain rate.  There
   will be a timer for determining exactly how often the GUI is updated.  This thread will run
   unless it has been stopped.

*/

 class GUIUpdate : public QThread
 {
     Q_OBJECT

     public:
         /*!\specDetails
             \brief Constructor for GUIUpdate Class
             \ensures
                 stopped = false
         **/
          GuiUpdate();

        /*!\specDetails
             \brief Request for thread to halt
             \requires
                 Thread is running
             \ensures
                 stopped = true
         **/

          void stop();

    protected:

        /*!\specDetails
             \brief Thread code for the GUIUpdate class
             \ensures The thread will begin running, updating the GUI periodically. The thread
                      will continue to run until the stopped flag has been set.
         **/
          void run();

     private slots:

           /*!\specDetails
             \brief Private slot for denoting that the timer has stopped
             \requires
             Thread is running
             \ensures
                       Thread will be woken to perform a GUI Update
         **/

          void timerExpired();

    private:

          QTimer timer; ///< QTimer class repesenting how often the GUI will update
          QWaitCondition timerExpired; ///< used to wake thread during periods of previous
                                           inactivity
          volatile bool stopped; ///< Boolean representing whether the thread has stopped or not
 };
```

Appendix B: Project Timeline