ECU Module Tester Proposal

Joseph Galbraith, Electrical Engineering

Sponsored by PSI Power

Project Advisor: Mr. Randall

March 21, 2019

Evansville, Indiana

Acknowledgements

I would like to thank Matt Keil for his advice regarding design considerations and PSI Power for providing testing equipment and components.

Table of Contents

List of Figures

List of Tables

Appendix

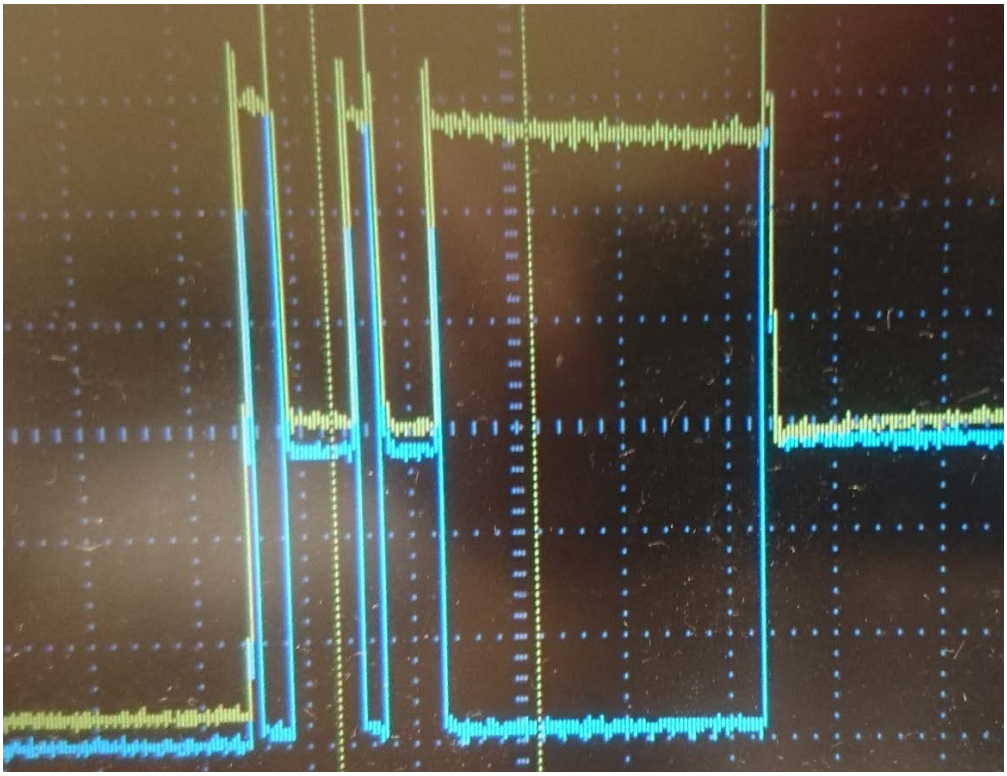
I. Introduction

This project is a customized tester for the injector class of module manufactured by PSI Power that will not only fix current issues faced with their current tester but also introduce new features. The custom tester is desired by the company because their current tester, LabView, is inaccurate and requires frequent upkeep of the program code. Along with fixing these issues, the custom tester can be operated entirely from a graphic user interface (GUI) and will stress test an injector module's components to ensure its reliability in the field. This new tester will enable PSI Power employees in the production department to better test the accuracy and quality of modules being manufactured.

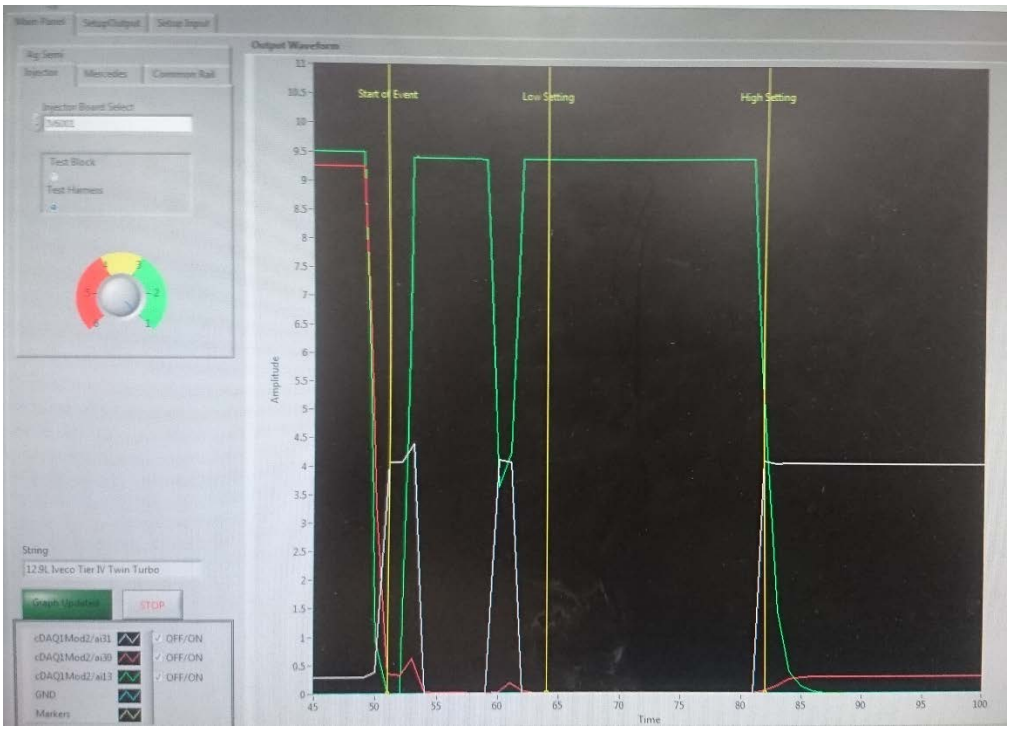
II. Problem Definition

PSI Power designs and manufactures a module (figure 2) that detects the signal coming from a vehicle's engine control unit (ECU) to the fuel injectors. Once the ECU is done opening an engine's fuel injectors, the module sends out its own signal to open these fuel injectors at key times in order to create up to 30% more power and up to 3 miles per gallon better fuel economy according to PSI Power's data. The company currently uses a testing kit called LabView to verify that their modules are outputting at the correct times and voltages after receiving the injector input signal, but LabView doesn't provide a signal reading accurate enough to satisfy the company. The analog to digital conversion (ADC) process that LabView uses to take readings is simply not fast enough to create an accurate analog reconstruction of a module's 1 MHz output signal, leaving up to a ±5 µs tolerance. Due to the precise timings which the injector class of module must output signals, a ±5 µs tolerance is not ideal. Additionally, LabView does not load

the module being tested, so there is no way of testing whether a module's components can withstand the current needed to drive an injector solenoid in the field. Another complaint from PSI Power about LabView is that every time the tester software is updated by the manufacturer, National Instruments, parts of the company written testing program code are broken. This requires an engineer to fix the code before LabView can be used again by production employees. One last flaw of LabView is that it needs to be restarted every time the module part number being tested is changed. These issues take up company time and reduce quality control. Both could be avoided with a customized tester.



[Figure 7: The high and low signal outputs of a specific module, the IV6902, viewed on an oscilloscope]



[Figure 8: The high and low signal outputs of a specific module, the IV6902, viewed on an LabView]

III. Requirements for New Tester

- Load box
- Single signal drive circuitry
- Dual signal sense circuitry
- GUI
    - Display high and low signal outputs from module
    - Display tolerance lines
    - Module information read from external file
        - Module names
        - Expected event timings

        - Injection event start lengths
        - Trigger voltage
- All logical instruction done on field programmable gate array (FPGA) with no external microprocessor
    - Intel NIOS II processor can be implemented on FPGA

IV. Solution

To satisfy the company, this new custom tester draws 1 A of current through the module being tested to simulate conditions it will face while in use. An automotive injector solenoid (Figure 1) was used to load the module since that is the load it will drive while in operation. This load circuit includes a diode for voltage protection to prevent flyback voltage from the solenoid reaching the module and the tester. An injector module has, at most, 8 signals controlling 6 injectors. 6 low signals go to 6 separate injectors to create a ground connection during operation. The remaining 2 high signals are split between the 6 injectors to create a positive voltage connection for each injector during operation. The modules are triggered by the voltage amplitude on the high signal and injection event length on the low signal. The high signal trigger voltage is specific to each module, so the voltage source driving it from the tester is variable between 3.3V and 5V. The tester also triggers a module to output by recreating the injection event from the ECU on a single low signal. The drive circuity on the tester recreates this injection event by pulling the low signal to ground for the time specific to that module. When both conditions are met, a properly functioning module outputs on a single high and low signal. After the module is triggered, the tester reads the high and low signals' event timings within ±1

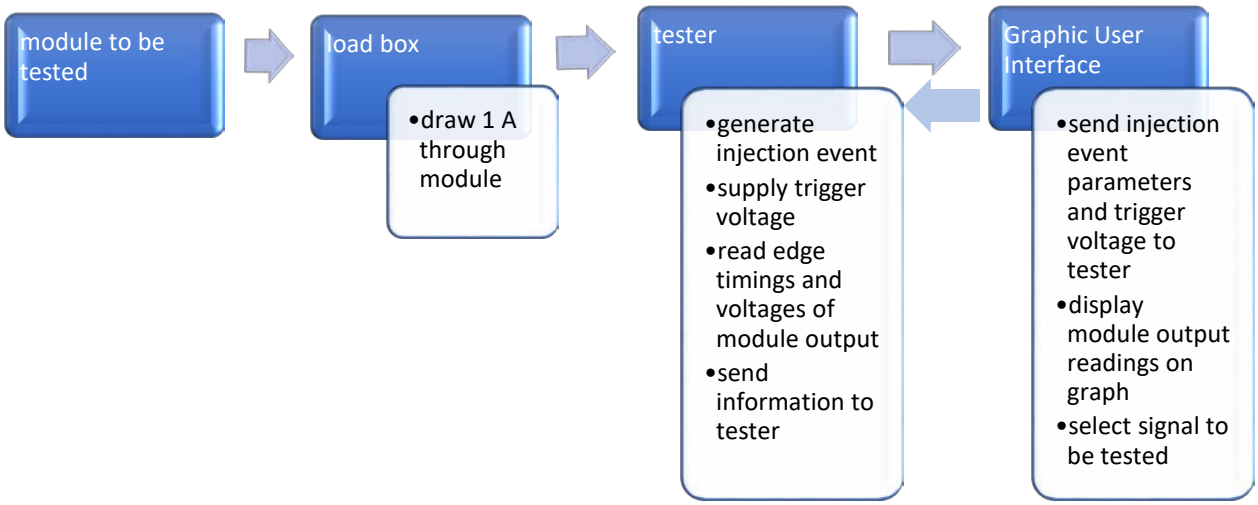
µs for a satisfactory tolerance. Edge detection logic and a timer is implemented on a MAX10M08SAE FPGA (Figure 3) to detect and record the output signals' event timings. For the signal edges to be safely and cleanly detected by the FPGA, the signals are compared and converted to 3.3 V. The voltage amplitudes of both signals are also recorded. To keep design costs minimal, an external high-speed ADC was not used. Instead, a low-speed ADC implemented within the integrated circuit of the MAX10 takes readings of the level edges of the signal. The output signal is voltage divided resulting in a linear voltage scale with a maximum of 3.3V to provide accurate voltage amplitude readings that do not exceed the MAX10's ADC input limits. A selector is necessary to cycle between the 8 signals that need testing on each module. A solid-state switching array is used so the signal switching can be controlled from the tester's GUI. For the tester to communicate with the GUI via universal asynchronous receiver/transmitter (UART), the NIOS II processor was implemented on the MAX10 FPGA. The NIOS II was coded in C to utilize UART protocol. In conjunction, a UB232R (figure 4) chip module from FTDI (Future Technology Devices International) was used as a UART to USB converter to allow communication between the NIOS II and the GUI's USB port on the computer.

The GUI was made using Microsoft Visual Studios 2017 to display the module's output signals on a graph with specific tolerance lines. Upon startup, the GUI parses a CSV (comma separated values) file for module part names with corresponding injection event lengths, tolerance lines, and trigger voltages. A drop box on the GUI allows the user to select the module part number being tested. Once selected, the GUI sends the corresponding injection event length and trigger voltage to the tester to be recorded, and the GUI displays the tolerance lines on the graph so that the user can easily determine whether the module's output signal is within tolerance. Timing ticks are displayed on the x-axis so that, in the event of a faulty module, the

user can see how far out of tolerance the output signal is. A start button on the GUI signals the tester to create an injection event and read the module's output signals. Every time an edge is detected and recorded, an interrupt is sent to the NIOS II to run the ADC and store both edge timing and ADC value. After the end of a testing cycle, the tester sends Visual Studios the timings of rising and falling edges as well as ADC data from the tester. Visual Studios converts the ADC data back to voltage amplitudes and uses the data to create a digital recreation of both output signals on the GUI. A second button on the GUI allows the user to change between tolerance lines for testing a module in high and low modes of operation.

The client's specifications included keeping costs at a minimum and to ensure the tester is capable of being expanded in the future to test multiple signals and possibly multiple modules at the same time. Making the tester expandable meant adding programming headers so that it can be reprogrammed and reading all module specifications from a comma separated values (CSV) file ensures that it can easily be added to or amended. To make this expandability possible, the client put no specifications on the size of the tester. Since 1 A of current runs through the load box, it is completely encased to prevent the user from electrocution. This complies with the IEEE code of ethics "to hold paramount the safety… of the public" (IEEE Code of Ethics #1). The diagram shown below illustrates the tester through the process of testing a module.

The flow of tester operations should be similar to the chart below



a. Manufacturability and Sustainability

The module was printed on a single PCB (printed circuit board) so a single wiring diagram can be made for reproduction. This ensures that the logic written works properly with the corresponding pin layout on any testers produced in the future. The connection made from tester to PC is a standard mini USB to USB which is cheap and widely available. The connection from the module test harness to the tester is RS-232 so the user can only connect to the tester one way, eliminating improper connections. Additionally, the pre-existing LabView test harnesses can be reused since they are also interfaced via RS-232 connection. The tester's power source comes from a voltage supply as opposed to USB. In the event of a short on the tester, the power supply will shut off instead of high current being transferred to the computer's USB port. The

tester's safety and sustainability are ensured with an enclosure to prevent against damage or undesired manipulation. Most of the components that populate the tester PCB come directly from the PSI Power stockroom because of their tested reliability.

b. Results

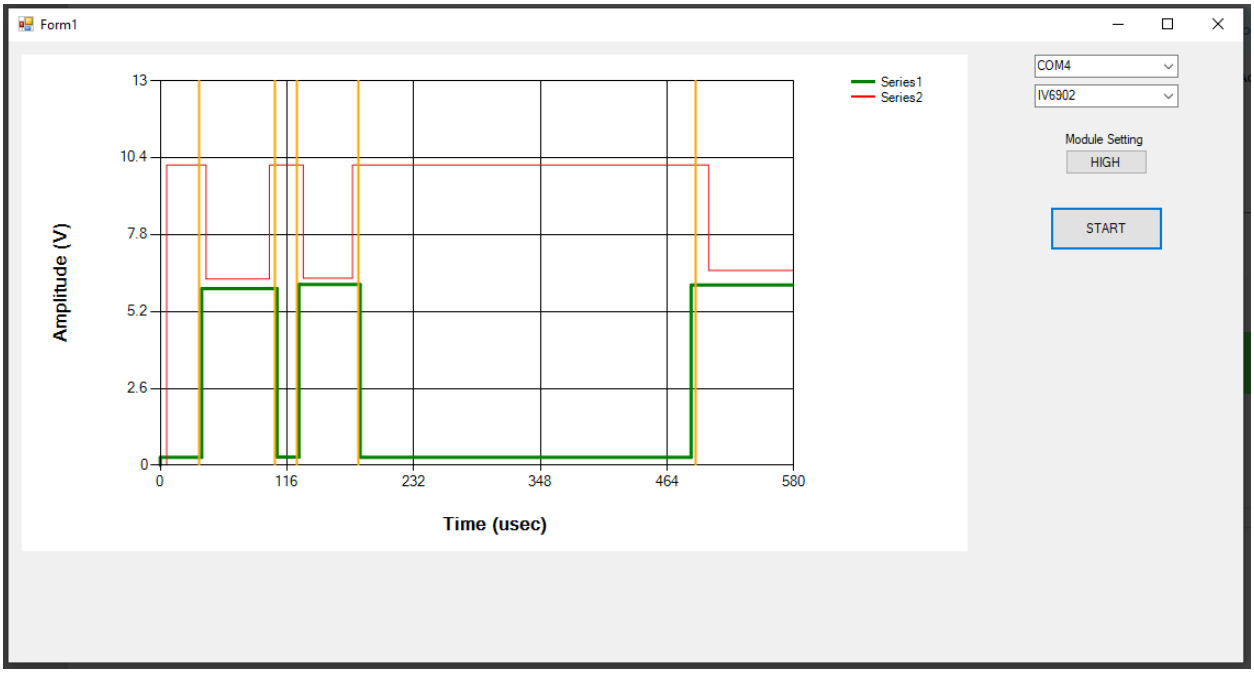
The user can attach an injector module to a previously existing LabView test harness and plug it into the RS-232 connector on the ECU module tester. The user must plug the USB cable into the USB port of the computer being used for testing and power both the module and tester with +12 V from a voltage supply at the testing station. Once the GUI is opened on the computer, it parses the designated CSV file on startup for all module names and corresponding injection event lengths, trigger voltages, and tolerance lines. The parameters are then stored in the Visual Studios code until the GUI is closed. The user can select the drop box labeled "COM PORTS" and view a list of all connected communication (COM) ports. The connected COM ports are updated every time the drop box is reopened. Once the appropriate COM port is found, it can be selected to open USB communications on. The "MODULES" drop box displays all module names that were read from the CSV file on startup. Once a module is selected, the GUI sends the injection event start length and trigger voltage to the tester via USB and is stored in the MAX10 FPGA. Additionally, the appropriate tolerance lines will appear on the graph after a module is selected. A button for the module mode of operation defaults to the high setting and displays "HIGH" on startup of the GUI and displays the corresponding tolerance line on the output graph. If the "HIGH" button is selected, the button displays "LOW" and changes the displayed tolerance line to the time corresponding to that module in low mode of operation. Once the start

button is selected, a signal is sent from the GUI to the tester via USB to test the module. The trigger voltage corresponding to the selected module is output to the high signal, and the corresponding injection event is created on the low signal. At the end of the injection event creation, the tester starts to record edge timings and voltage amplitudes. Recording stops soon after the expected end of events for that module, and all edge timings and ADC values are sent to the GUI. The GUI converts the ADC data into real voltage amplitudes and displays the modules output on the graph. After graphing, the GUI will send another start signal to the tester. This process is done 100 times every time the “START” button is pressed. If the company desires to change the number of tests per cycle, it can easily be updated in the code.

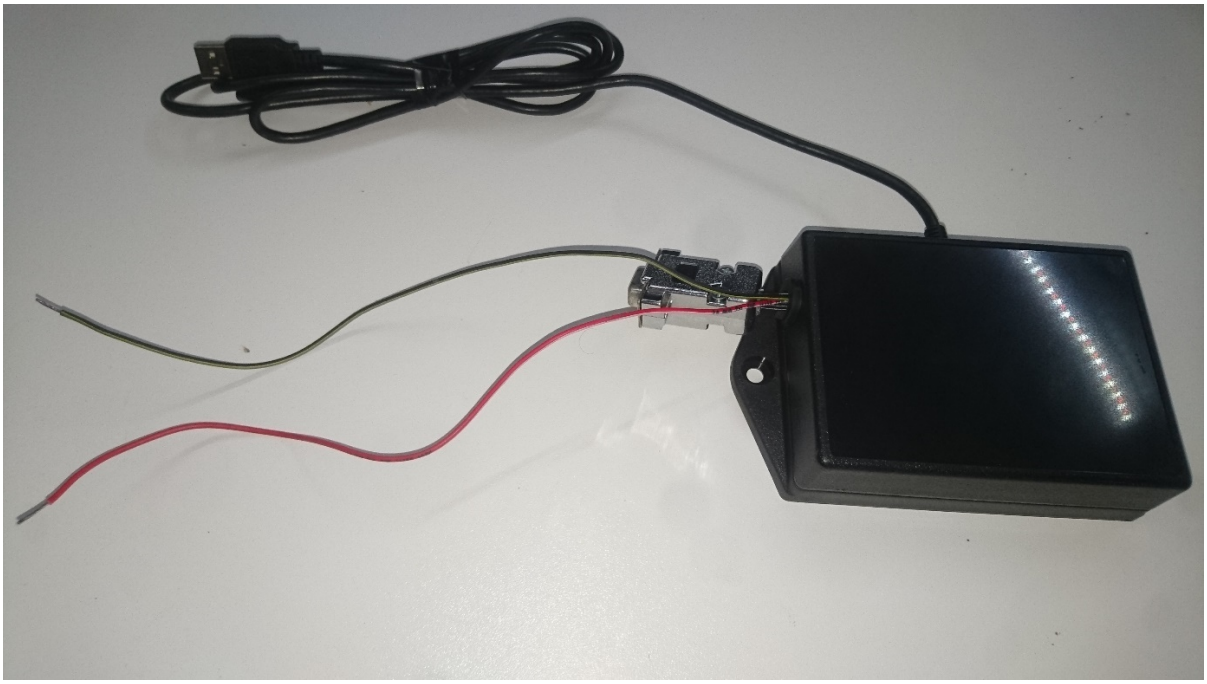
Figure 5 below is the current GUI upon startup.

[Figure 5: The COM Port and Module selection drop down-menus can be seen in the top right corner. Below the drop-down menus is the module setting button (high/low) and start button.]

Figure 6 below is the current GUI testing a specific module, the IV6902



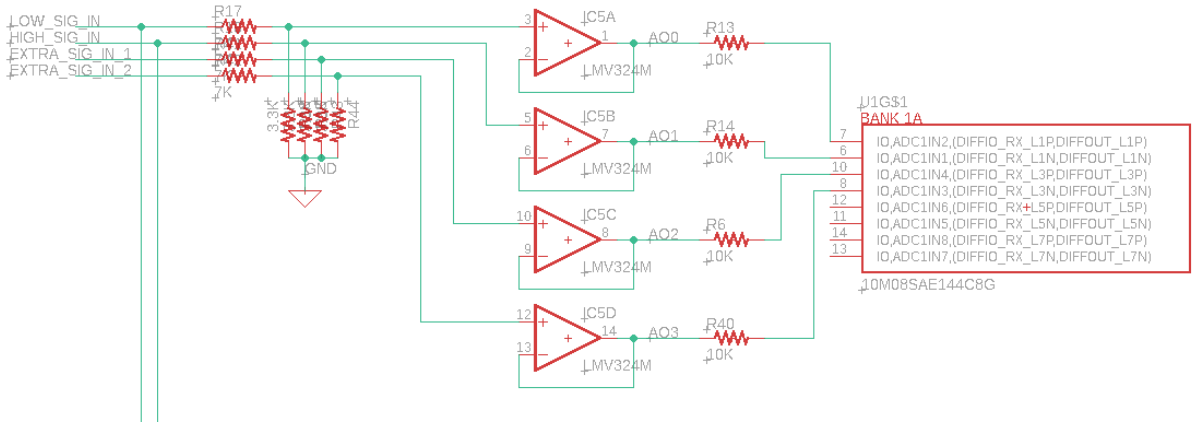
[Figure 6: Upon module selection, the graph axis and tolerance lines are populated. The graphed module output appears when the start button is pressed]



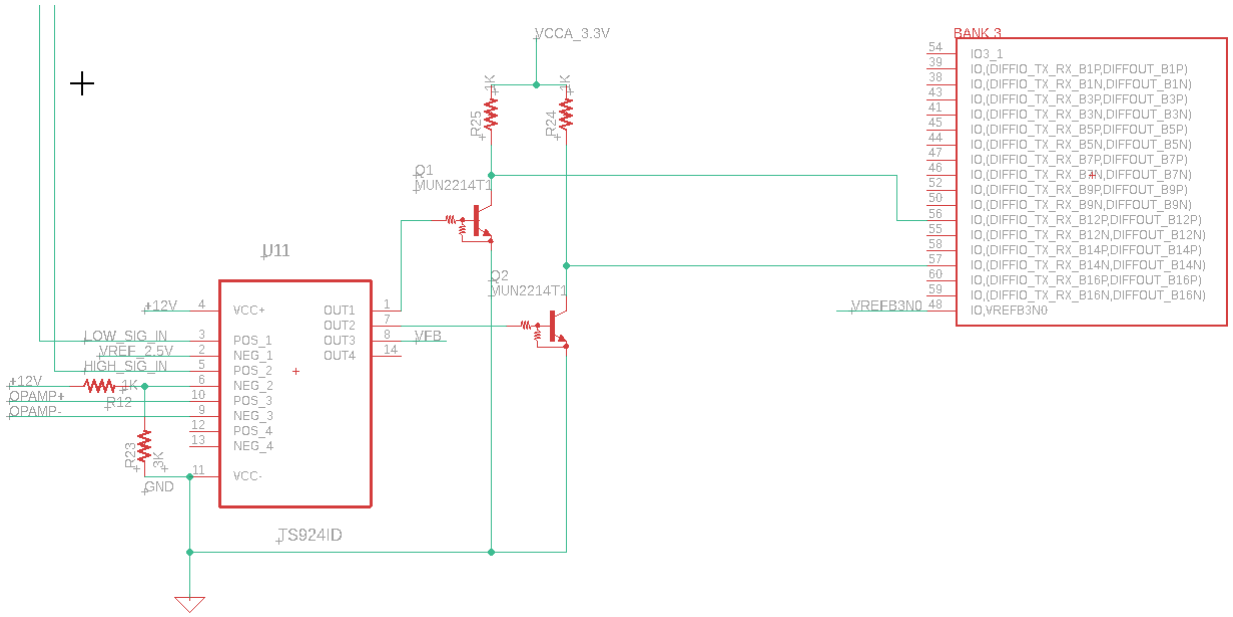
[Figure 18: Finished tester with external power connections, USB connector, and RS-232 connector]

V. Hardware Design

a. Sense Circuitry



[Figure 9: ADC Circuitry]

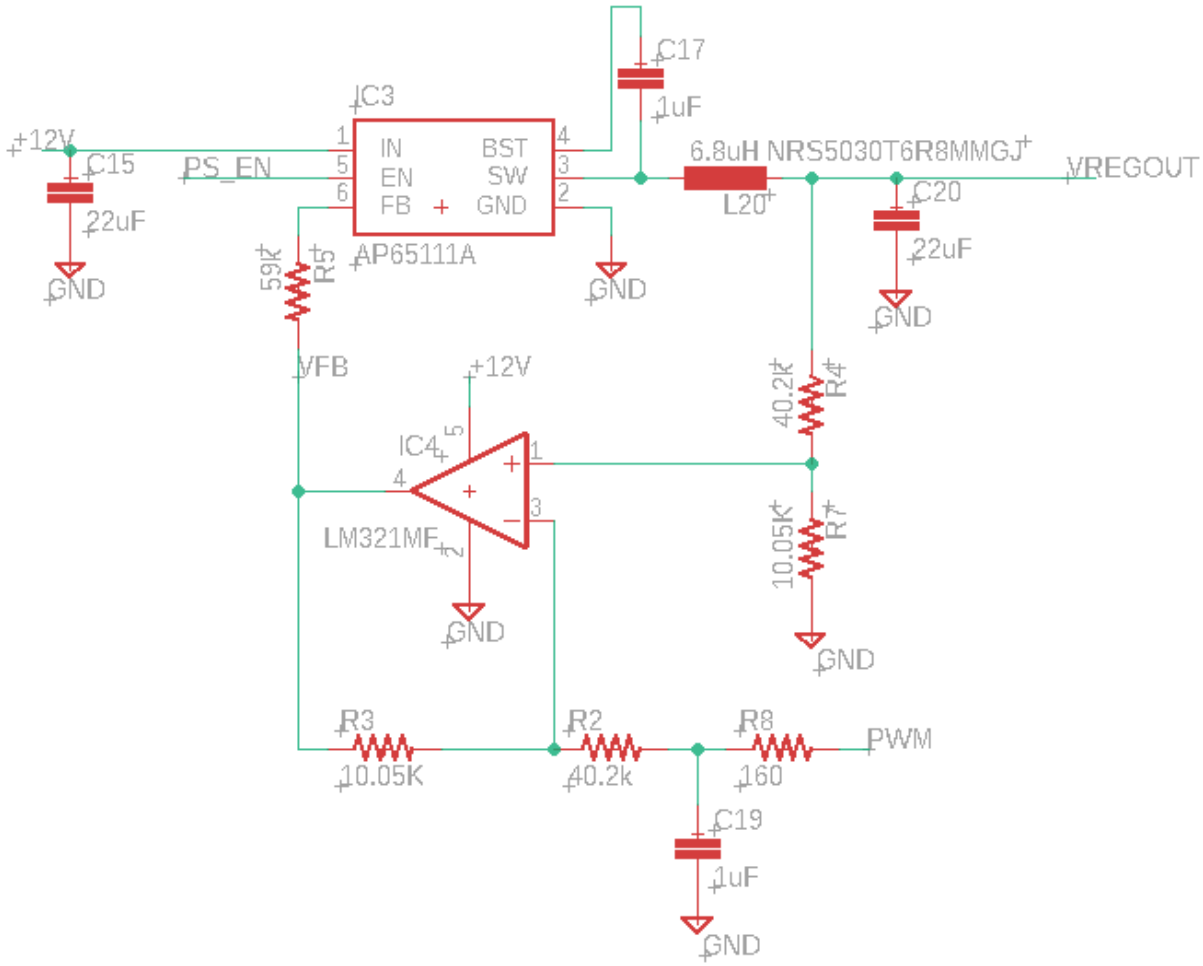


[Figure 10: Edge Detection Circuitry]

The voltage divider was designed to divide a maximum of 12V input down to a maximum of 3.3V at the MAX10's ADC input. The LM324QDR high-speed op amps serve as a high

impedance input to protect the MAX10's inputs. The CD4050B is a non-inverting Schmitt trigger that converts the output signals from the module into a 3.3V rising and falling edge that can be detected by the MAX10's logic.

b. Variable Voltage Supply



[Figure 11: Variable Voltage Supply Circuitry]

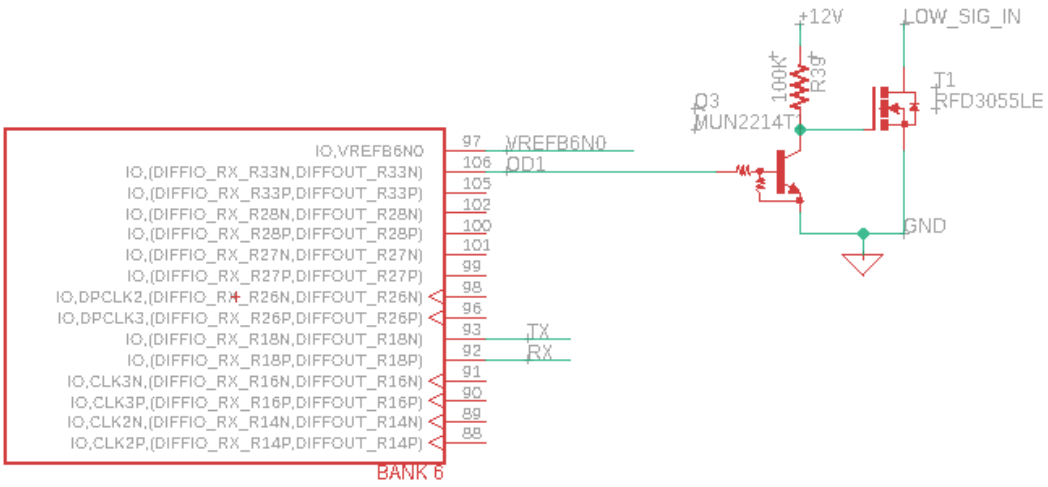
The variable voltage regulator supply's the high signal of the module's output. To make the regulators voltage adjustable by code, a PWM (pulse width modulated) signal is to a voltage

subtractor circuit along with the output of the voltage regulator. The voltage subtractors output is sent to the feedback pin of the regulator, causing the regulator to output a voltage proportional to the PWM signal. $Vfb = \frac{10.05K\Omega}{40.2K\Omega}(Vo - Vpwm)$

$$.8V = .25(Vo - Vpwm)$$

$$Vo = 3.2V + Vpwm$$

c. Drive Circuitry

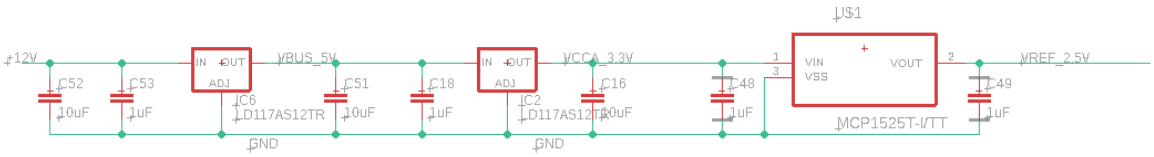


[Figure 12: Drive Circuitry]

The drive circuit used to pull down the low signal of the module is created using a high current capable RFD3055LE MOSFET. A MMUN2215 BJT is used to create an open drain. Until triggered by the tester, the BJT remains on by supplying current to the base with a MAX10 output pin, opening the path to ground and keeping the gate of the MOSFET inactive. When creating an injection event, the MAX10

output is low, which deactivates the BJT, activates the MOSFET with 12V, and pulls the low signal to ground.

d. Power Regulation

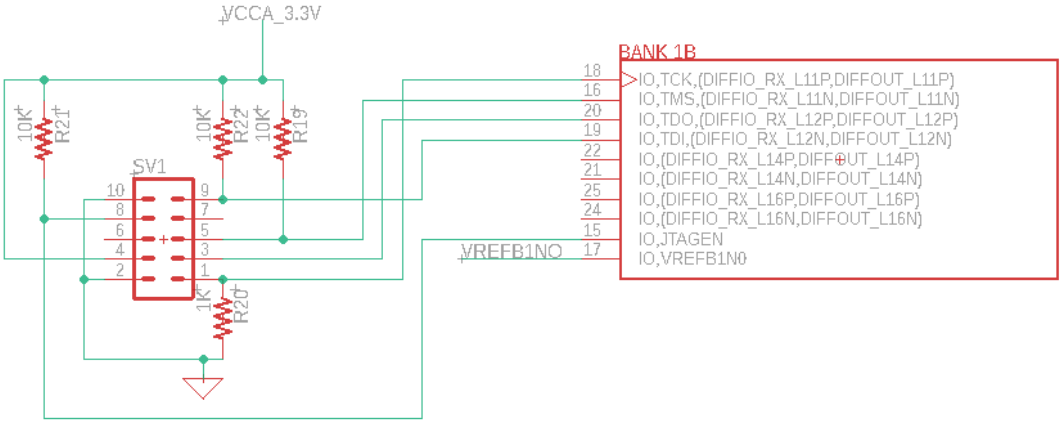


[Figure 13: Power Regulation Circuitry]

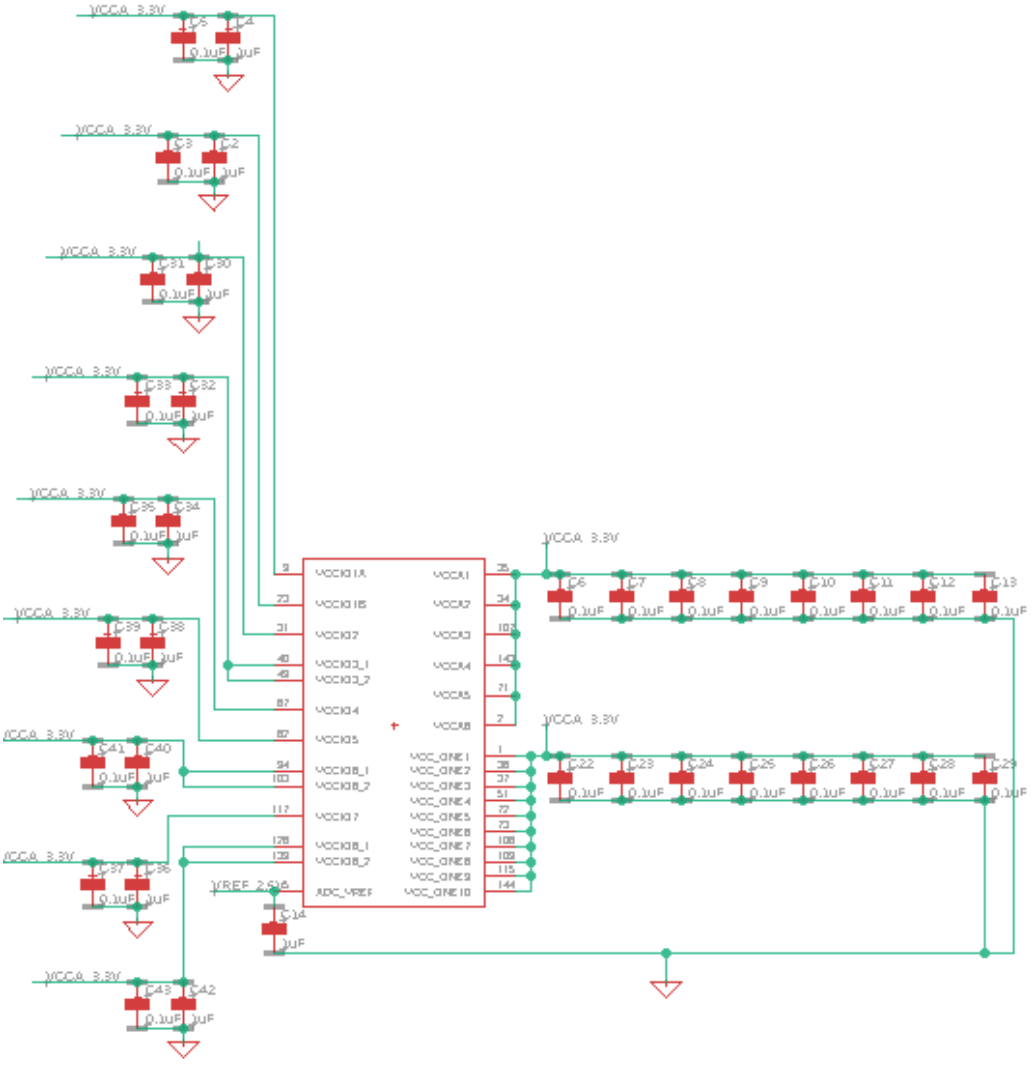
Power is supplied to the tester from the 12V supply instead of a USB connection for the contingency that the tester malfunctions. If a short occurs, it will draw current from the power supply instead of the computer's USB port. 5V and 3.3V LD117 buck regulators are used because of their availability in the PSI Power stockroom. The MCP1525T is used for a 2.5V buck converter because none are stocked at PSI Power, and this was the standard converter used on the development board. The filter capacitor values used on the input and outputs of the buck converters are all recommended operating conditions from the manufacturers.

e. Power Conditioning, Oscillator, and Programming Circuit

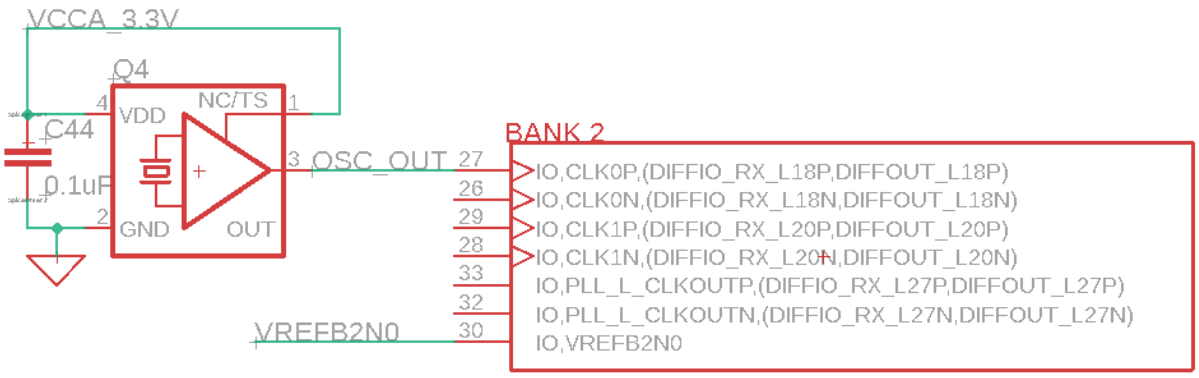
All three circuit designs were sourced from the MAX10M08 evaluation board for their tested reliability



[Figure 14: Programming Header Circuitry]

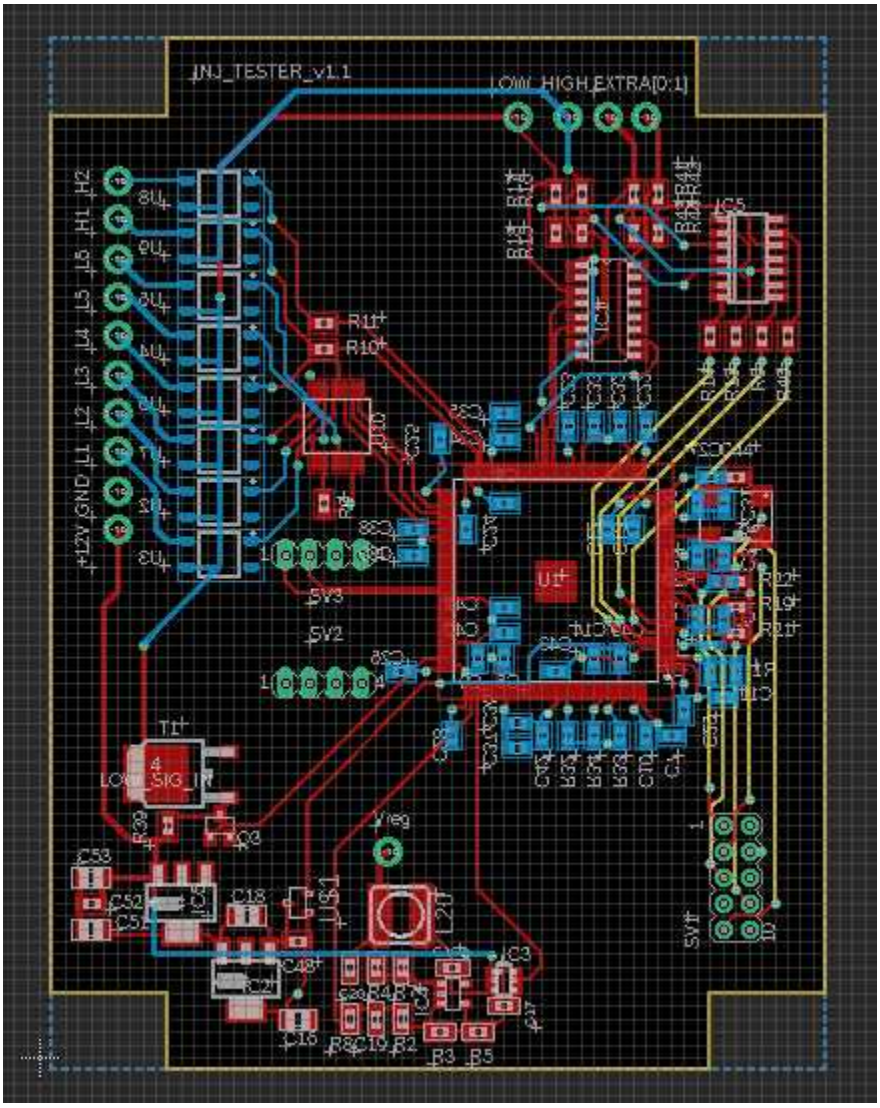


[Figure 15: FPGA Power Conditioning Circuitry]



[Figure 16: 50 MHz Oscillator Circuitry]

f. PCB



[Figure 17: Tester PCB]

The PCB is designed on a 4-layer board for its capacitive effect and size efficiency. The top and bottom layers are ground planes, and the middle 2 layers are 3.3V, creating a capacitive effect that will reduce overall signal noise. Filter capacitors are placed as close to the MAX10 as possible to condition the supplied voltage. The USB input and all signal inputs are placed on the left side so that can be used as the connection side once mounted in an enclosure. Power

regulation is placed in the opposite corner of the sense circuitry to avoid noise resulting from high current. The programming circuit is placed away from the power regulation circuit for the same purpose.

VI. Verilog Modules

a. Rise Edge Detection

The rise edge detection logic stores the value of the signal from the previous clock cycle and compares it with the current signal. That way, the accuracy of the edge detection logic is down to the nearest 20 nanoseconds. If the signal goes from low to high, a pin tied to the NIOS II is pulled high for approximately 1.2 microseconds so that the NIOS II has time to register an interrupt. The time of edge detection is read from the counter module and stored in the edge detection module to be read by the NIOS II when the interrupt is registered.

b. PWM

The PWM module is used to create an analog voltage of either 2.8 V or 0.8 V. If the input from the NIOS II is high, the PWM output is set to output 2.8 V. If the input from the NIOS II is low, the PWM output is set to 0.8 V. This is done by using an 8 bit register that is incremented every clock cycle. If the value of the register exceeds the value for the appropriate voltage, the PWM output is low. Otherwise, the PWM output is high at 3.3 V.

c. Fall Edge Detection

The fall edge detection logic stores the value of the signal from the previous clock cycle and compares it with the current signal. That way, the accuracy of the edge detection logic is down to the nearest 20 nanoseconds. If the signal goes from high to low, a pin tied to the NIOS II is pulled high for approximately 1.2 microseconds so that the NIOS II has time to register an interrupt. The time of edge detection is read from the counter module and stored in the edge detection module to be read by the NIOS II when the interrupt is registered.

d. Injection Event Creation

The injection event inputs the counter module's value and the current module's injection event length. If a start signal is received from the NIOS II, the counter value is reset, and the output going to the injection event creation hardware is set low. This pulls the low signal to ground. If the value of the counter exceeds the value of the module's injection event length, the output is high, and the low signal is let go. This also resets the counter so that the counter value is zero when the edge detection is enabled.

e. Counter

The counter increments a 6-bit register every time the 50 MHz clock is high. Once the register reaches 50, it is cleared, and the 16-bit microsecond register is incremented. This creates an accurate counter that increments every microsecond that is output to the other modules.

VII. NIOS II Code

The NIOS II code is in place to handle UART communications and interrupts. Each character read from the Visual Studios GUI has its own protocol. In the main function, all edge detection interrupts are registered. Once in the main function, the code waits and reads the communication port for a character. If an ‘H’ is received, the NIOS II receives the module’s injection event length and trigger voltage from the GUI. The injection event length is then sent to the injection event creation module to be stored. If an ‘@’ is received, the tester has been signaled to test the module. All edge detection interrupts are disabled so that it does not record the injection event recreation. All variables for edge timings and ADC values are reset, and the ADC is enabled. A start signal is pulsed to the create injection event module, the edge detection interrupts are enabled, and the flags are cleared after the injection event. Every time an edge is detected, the ADC is run to determine the voltage amplitude at the beginning of a level edge. Additionally, the time of edge detection is read from the edge detection module and is stored in an array. Once the counter time has exceeded the maximum time of module operation, all ADC values and edge timings are sent to the GUI.

VIII. Visual Studios GUI

On startup, the GUI parses the module parameters CSV file. It sorts each row into an array corresponding to appropriate parameter, and then adds the module names read to the “MODULES” drobox on the GUI. If the CSV file could not be read, an error box will appear on the GUI to inform the user. Also on startup, all tolerance lines are initialized. When the “COM PORTS” drop box is selected, the connected COM ports are detected and displayed on the drop box. If a COM port is selected from the “COM PORTS” drop box, USB communication is

initialized, and a serial connection is attempted on that port. If the connection were to fail, an error message would appear on the GUI. When a module is selected from the “MODULES” drop box, the module name array is parsed until it finds the selected module name. That array item number is then used select the corresponding parameters from the parameters’ arrays. An ‘H’ is sent to the NIOS II to initialize the tester to receive start parameters, then the injection event length and trigger voltage are sent to the tester. Finally, all tolerance line values are assigned according to the selected module and are displayed on the GUI’s graph. If the “MODULE SETTINGS” button is pressed, the GUI alternates the text between “HIGH” and “LOW”, and the extend tolerance line is updated accordingly. When the start button is selected, the GUI’s graph is cleared and an ‘@’ is sent to the tester. Once a “data_start” is received from the tester, the GUI reads and stores all edge timings and ADC values. The ADC values are converted back into analog voltages and stored in an array. The high and low signals’ are added to the graph, and the graph is updated with the new outputs. This start protocol runs 100 times before stopping to ensure the module’s output is consistently accurate.

Figures

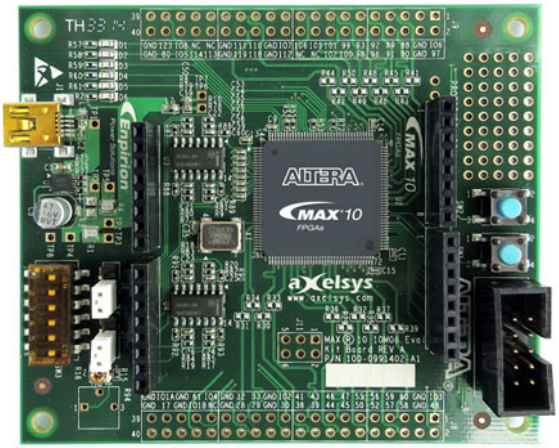
1.)

12V 1A Solenoid

2.)

PSI Power Injector Module

3.)



MAX10M08SAE144 Evaluation kit

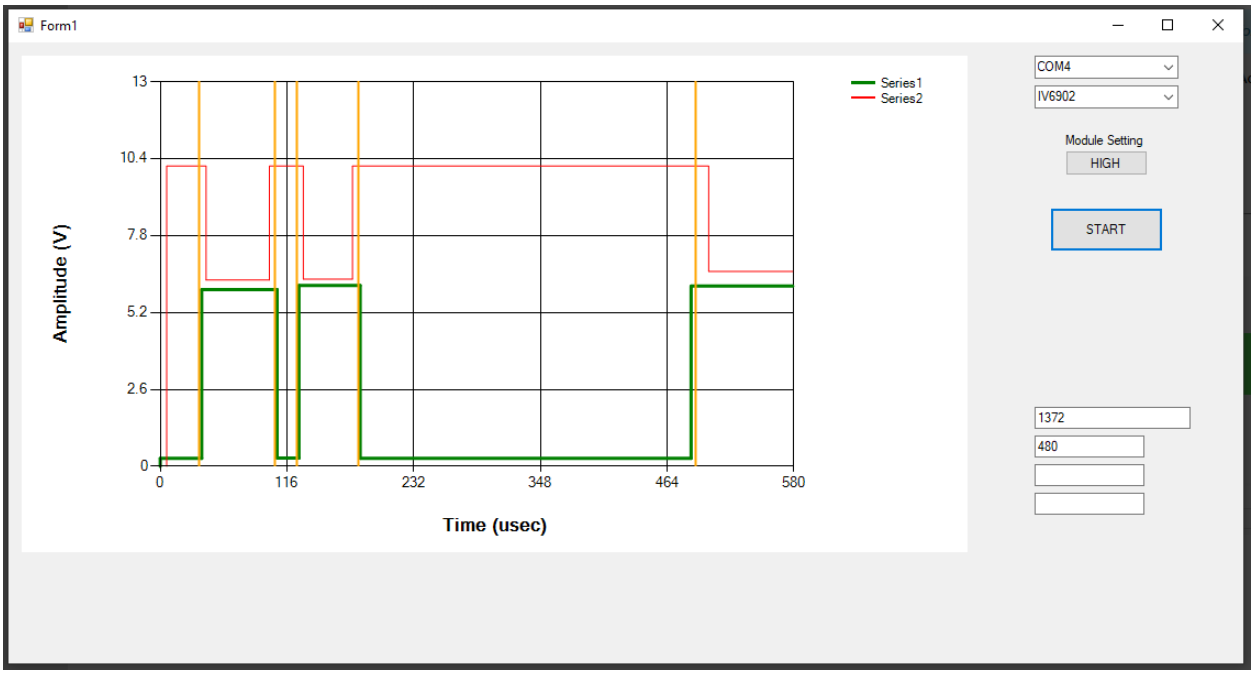
4.)

FTDI UB232R

5.)

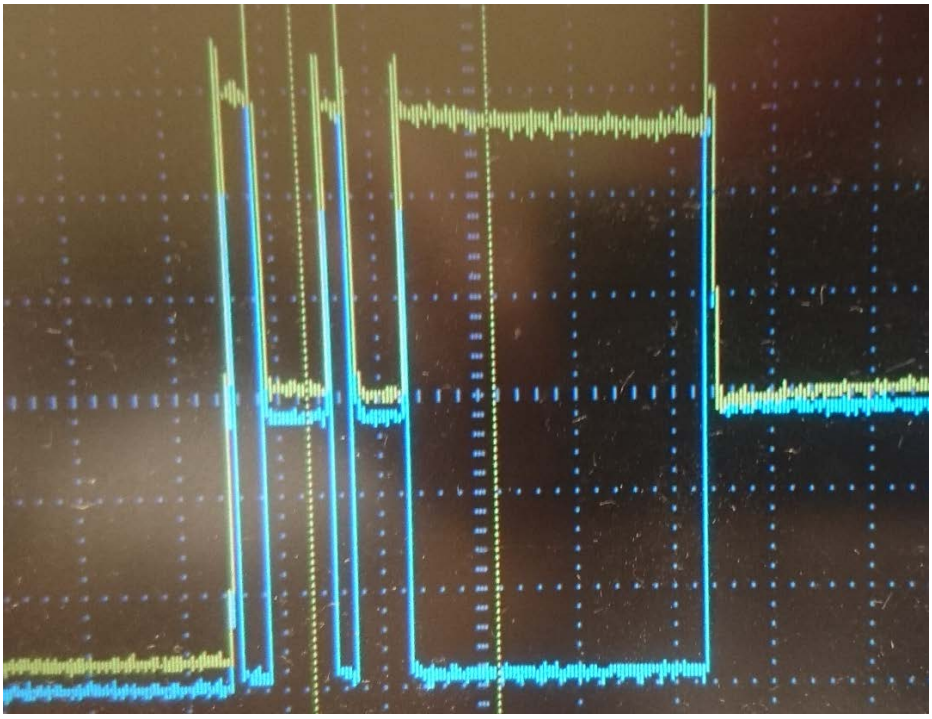
GUI on startup

6.)



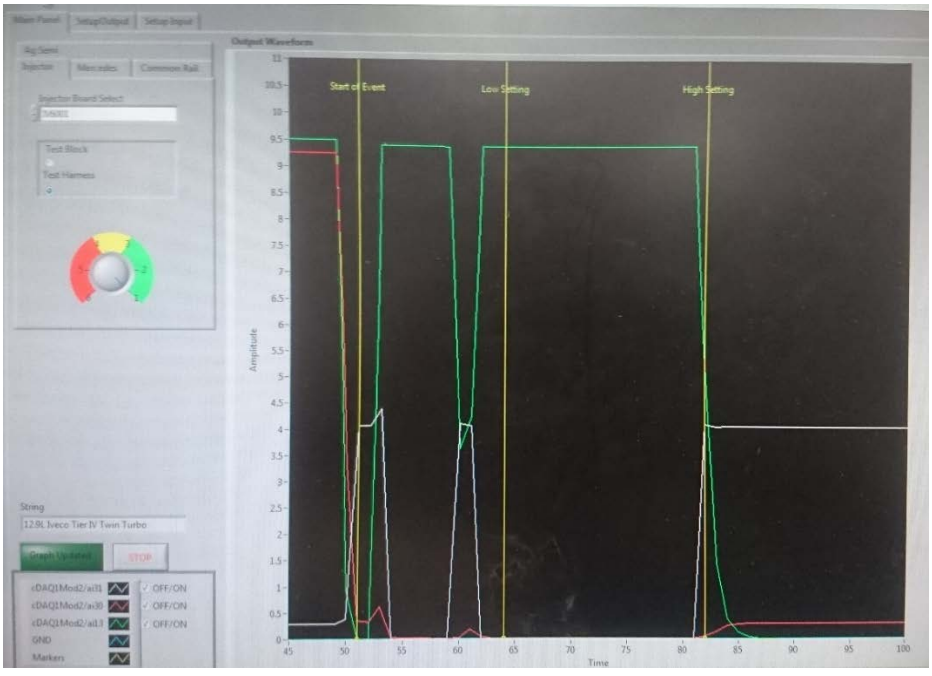
GUI in use

7.)



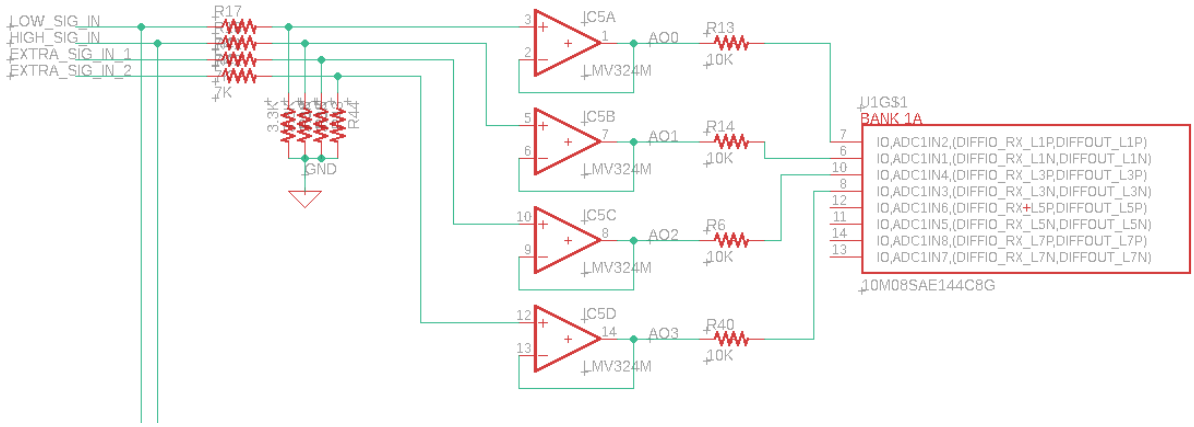
Module output on oscilloscope

8.)



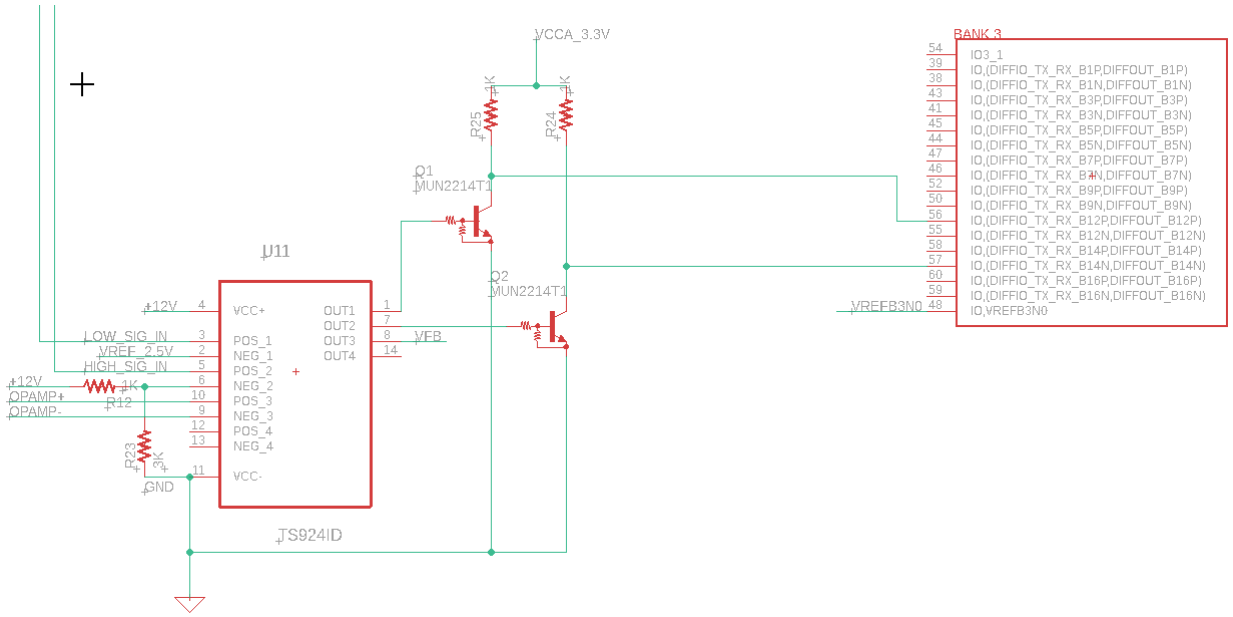
Module output on LabView

9.)



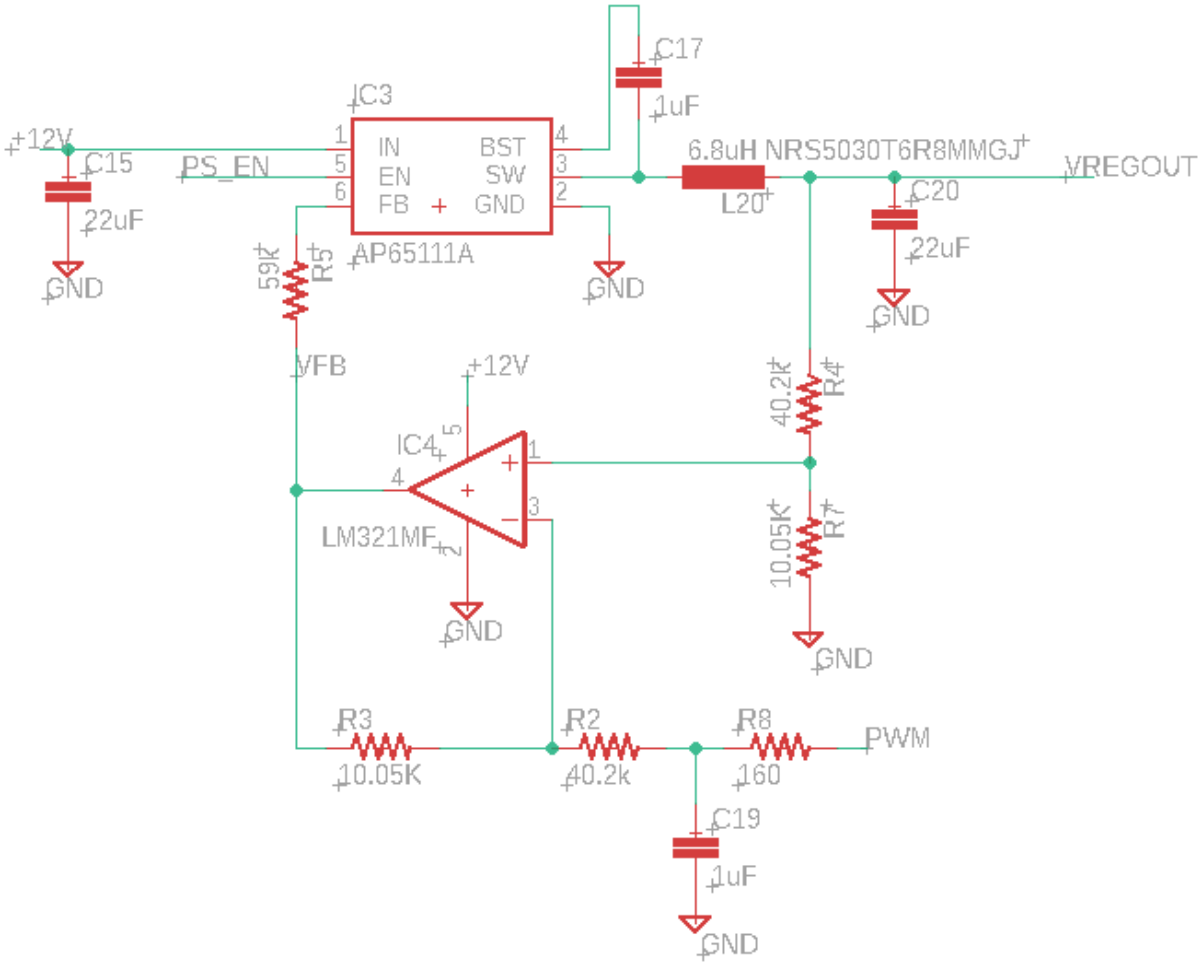
ADC Circuitry

10.)



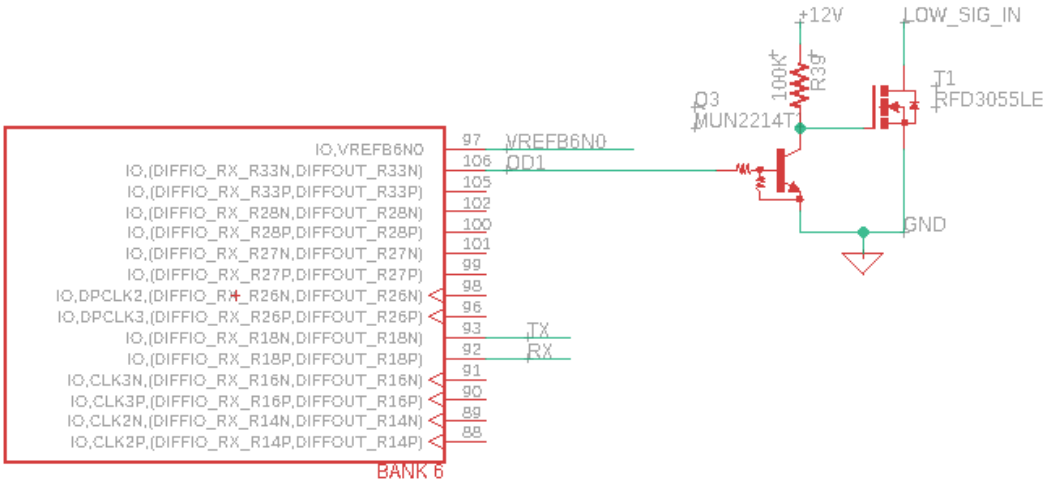
Edge Detection Circuitry

11.)



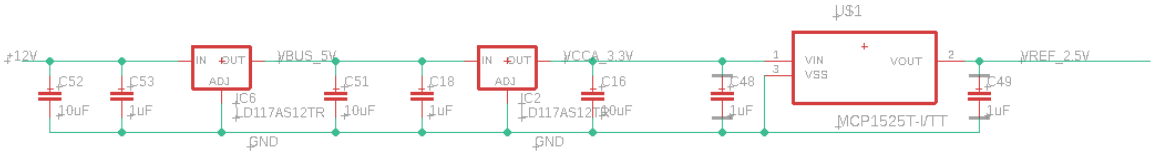
Variable Voltage Supply Circuitry

12.)



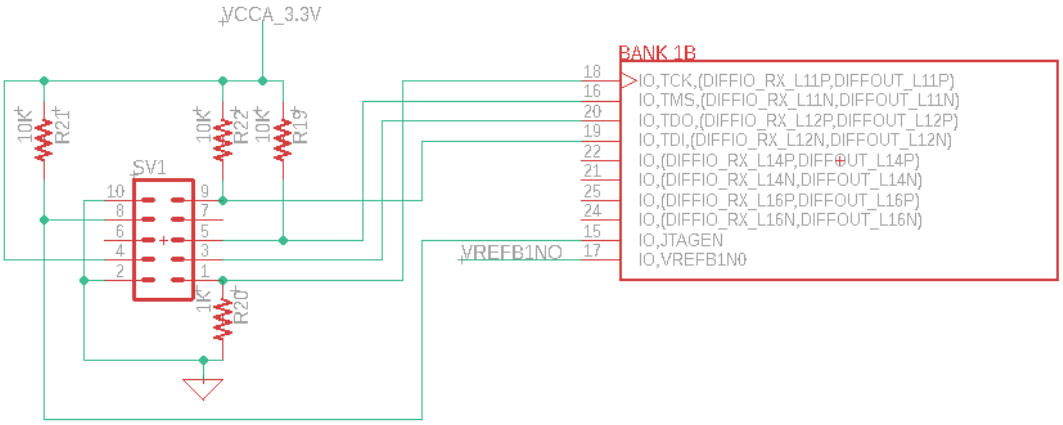
Drive Circuitry

13.)



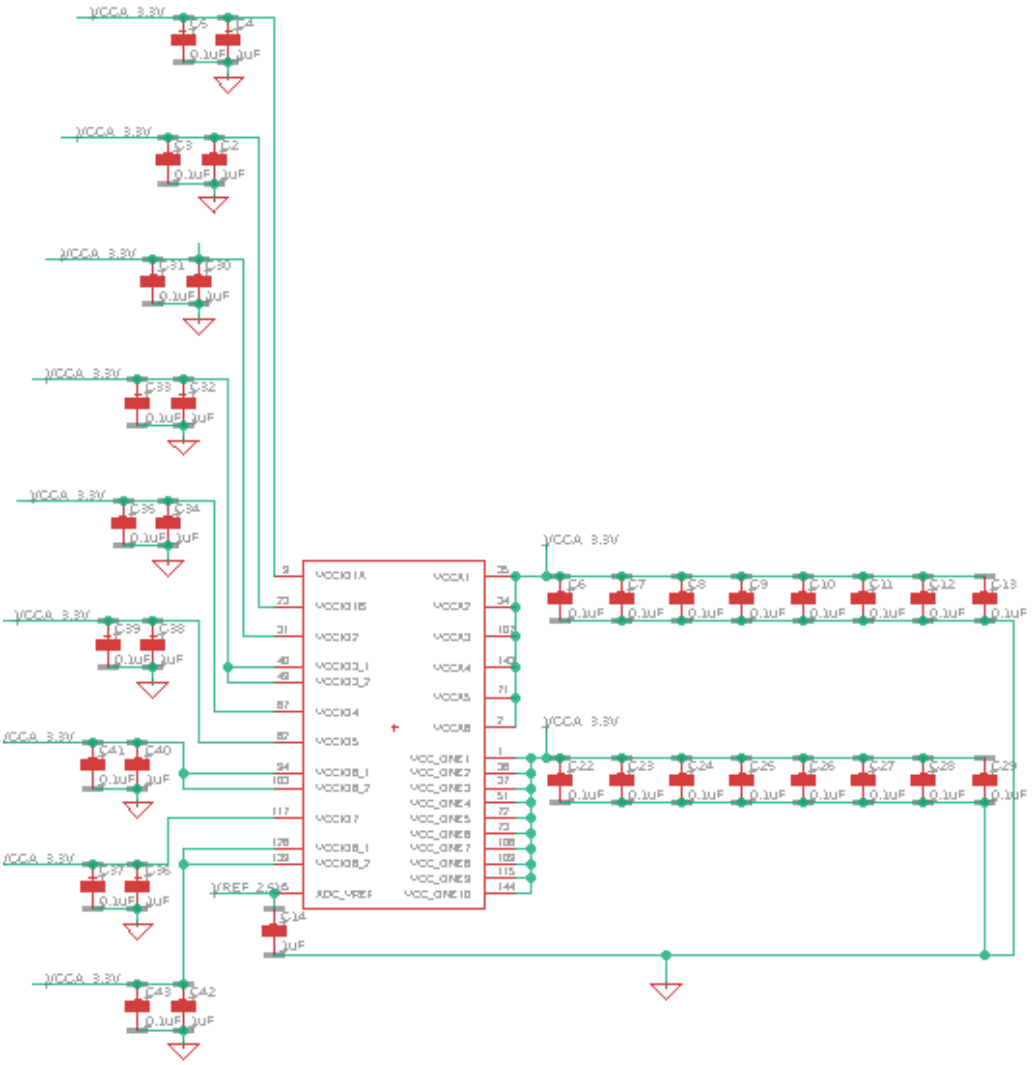
Power Regulation Circuitry

14.)



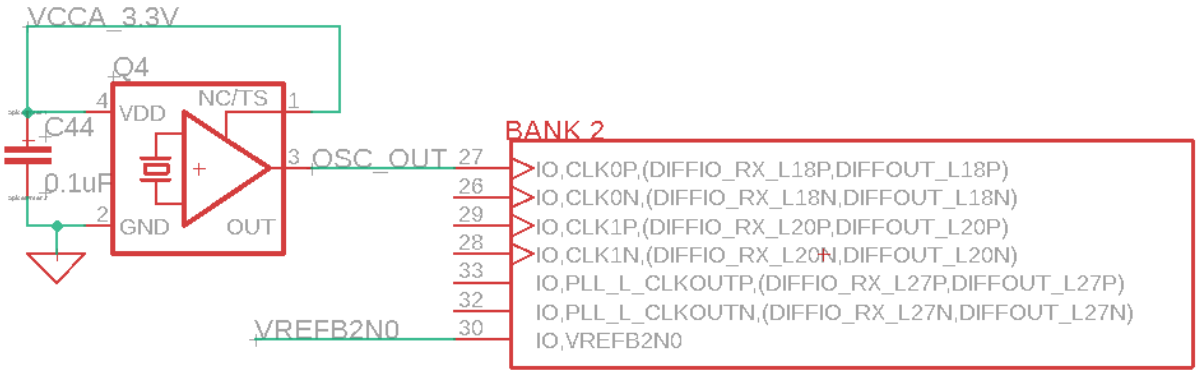
Programming Header Circuitry

15.)



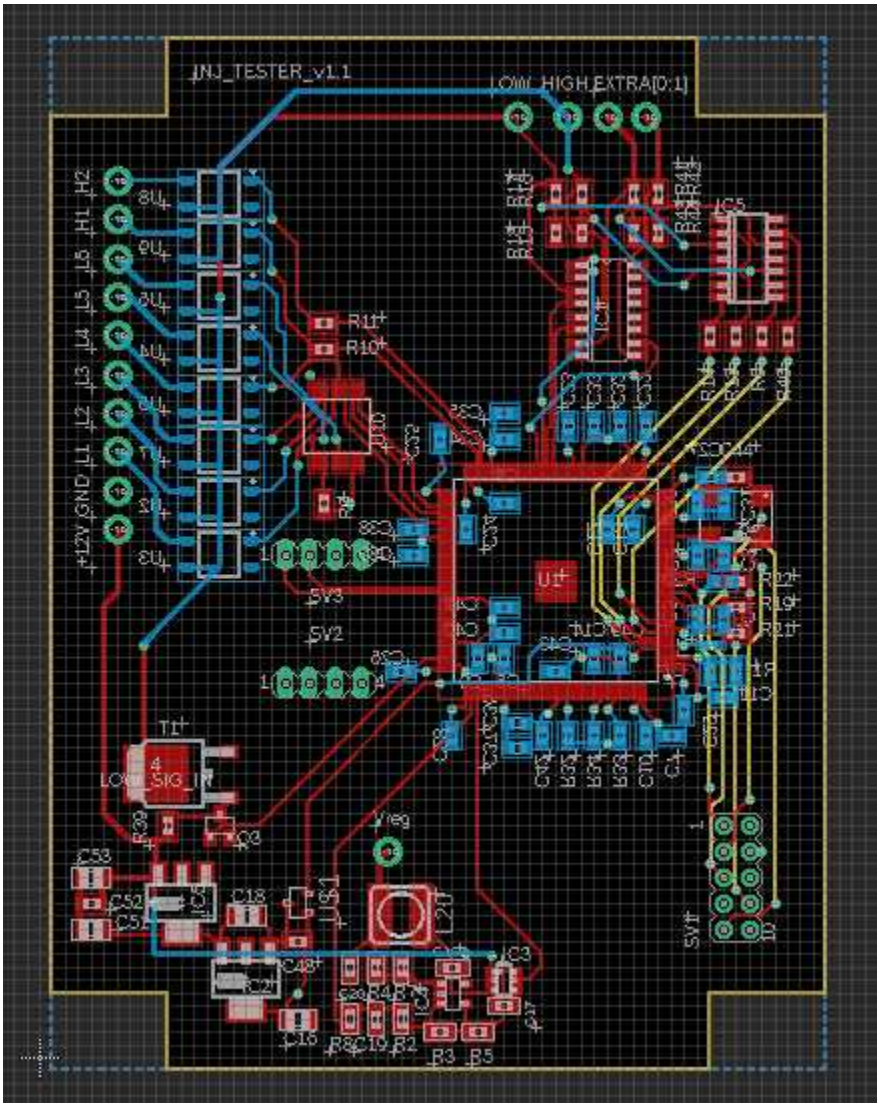
FPGA Power Conditioning Circuitry

16.)



50 MHz Oscillator Circuitry

17.)



Tester PCB

Costs

All costs are being covered by PSI Power, which is why one of the project specifications was to keep costs as low as possible. The MAX10M08 evaluation board that was used for development was already available in PSI Power's stockroom. Most components that populate

the PCB were already stocked at PSI Power. The wires, wire loom, RS-232 connector and enclosure were obtained cheaply by repurposing them from PSI Power's stockroom.

Table 1

Component Item ID	Reference	Description	Manufacturer	Price ($)
0.1µF 25V	C01	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C02	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C03	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C04	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C05	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C06	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C07	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C08	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C09	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C10	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C11	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C12	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C13	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C14	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
10µF 10V	C16	0603 10uF 10V 20% X5R	PSI Power's Choice	0.12888
1µF 25V	C18	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C22	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C23	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C24	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C25	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C26	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C27	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C28	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C29	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C30	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C31	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C32	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C33	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C34	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C35	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C36	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C37	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C38	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C39	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C40	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087

Component Item ID	Reference	Description	Manufacturer	Price ($)
0.1µF 25V	C01	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C02	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C03	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C04	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C05	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C06	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C07	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C08	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C09	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C10	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C11	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C12	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C13	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C14	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
10µF 10V	C16	0603 10uF 10V 20% X5R	PSI Power's Choice	0.12888
1µF 25V	C18	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C22	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C23	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C24	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C25	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C26	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C27	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C28	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C29	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C30	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C31	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C32	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C33	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C34	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C35	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C36	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C37	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C38	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C39	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C40	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C41	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C42	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
0.1µF 25V	C43	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
0.1µF 25V	C44	0603 100nF 25V 10% X7R	PSI Power's Choice	0.01035
1µF 25V	C48	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
1µF 25V	C49	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087
10µF 10V	C51	0603 10uF 10V 20% X5R	PSI Power's Choice	0.12888
10µF 10V	C52	0603 10uF 10V 20% X5R	PSI Power's Choice	0.12888
1µF 25V	C53	0603 1uF 25V 10% X7R	PSI Power's Choice	0.0087

40.2KΩ	R02	Resistor 0603 SMD 1%	PSI Power's Choice	0.0086
10.05KΩ	R03	Resistor 0603 SMD 1%	PSI Power's Choice	0.462
40.2KΩ	R04	Resistor 0603 SMD 1%	PSI Power's Choice	0.0086
59KΩ	R05	Resistor 0603 SMD 1%	PSI Power's Choice	0.1532
10KΩ	R06	Resistor 0603 SMD 1%	PSI Power's Choice	0.0124
10.05KΩ	R07	Resistor 0603 SMD 1%	PSI Power's Choice	0.462
160Ω	R08	Resistor 0603 SMD 1%	PSI Power's Choice	0.3
30Ω	R09	Resistor 0603 SMD 1%	PSI Power's Choice	0.0171
130Ω	R10	Resistor 0603 SMD 1%	PSI Power's Choice	0.009
130Ω	R11	Resistor 0603 SMD 1%	PSI Power's Choice	0.009
10KΩ	R13	Resistor 0603 SMD 1%	PSI Power's Choice	0.0124
10KΩ	R14	Resistor 0603 SMD 1%	PSI Power's Choice	0.0124
3.3KΩ	R15	Resistor 0603 SMD 1%	PSI Power's Choice	0.0064
7KΩ	R16	Resistor 0603 SMD 1%	PSI Power's Choice	0.0271
7KΩ	R17	Resistor 0603 SMD 1%	PSI Power's Choice	0.0271
3.3KΩ	R18	Resistor 0603 SMD 1%	PSI Power's Choice	0.0064
10KΩ	R19	Resistor 0603 SMD 1%	PSI Power's Choice	0.0124
1KΩ	R20	Resistor 0603 SMD 1%	PSI Power's Choice	0.01428
10KΩ	R21	Resistor 0603 SMD 1%	PSI Power's Choice	0.0124
10KΩ	R22	Resistor 0603 SMD 1%	PSI Power's Choice	0.0124
1KΩ	R32	Resistor 0603 SMD 1%	PSI Power's Choice	0.01428
1KΩ	R33	Resistor 0603 SMD 1%	PSI Power's Choice	0.01428
1KΩ	R34	Resistor 0603 SMD 1%	PSI Power's Choice	0.01428
100KΩ	R39	Resistor 0603 SMD 1%	PSI Power's Choice	0.002
10KΩ	R40	Resistor 0603 SMD 1%	PSI Power's Choice	0.0124
7KΩ	R41	Resistor 0603 SMD 1%	PSI Power's Choice	0.0271
7KΩ	R42	Resistor 0603 SMD 1%	PSI Power's Choice	0.0271
3.3KΩ	R43	Resistor 0603 SMD 1%	PSI Power's Choice	0.0064
3.3KΩ	R44	Resistor 0603 SMD 1%	PSI Power's Choice	0.0064
LD1117S33TR	IC2	) fixed positive voltage regulator	PSI Power's Choice	0.44
AP65111A	IC3	1.5A SYNCH DC/DC BUCK CON	PSI Power's Choice	0.2985
LMV324QDR	IC5	ail Output Operational Amplifier	PSI Power's Choice	0.5941
LD1117S50TR	IC6	Fixed Positive Voltage Regulato	PSI Power's Choice	0.45
BLM15HB121SN1	L1	Ferrite Bead 0402	PSI Power's Choice	0.145
NRS5030T6R8MMGJ	L20	6.8uH Fixed Inductor 5030	PSI Power's Choice	0.234
MMUN2215LT1G	Q1	ANS PREBIAS NPN 0.4W SOT2	PSI Power's Choice	0.0634
MMUN2215LT1G	Q2	ANS PREBIAS NPN 0.4W SOT2	PSI Power's Choice	0.0634
MMUN2215LT1G	Q3	ANS PREBIAS NPN 0.4W SOT2	PSI Power's Choice	0.0634
CB3LV-3C-50M0000	Q4	OSC 50MHz 3.3V 50ppm	PSI Power's Choice	1.57
MA05-2	SV1	5x2 Pin Header	PSI Power's Choice	0.1
RFD3055LE	T1	MOSFET N-MOSFET D-PAK TO	PSI Power's Choice	0.50352
MCP1525T-I/TT	U$1	.5 V Voltage Reference SOT-23-	PSI Power's Choice	0.79
10M08SAE144C8G	U1	MAX10 FPGA	PSI Power's Choice	21.78

AQY212GSX	U2	Solid State Relay SOP4	PSI Power's Choice	3.267
AQY212GSX	U3	Solid State Relay SOP5	PSI Power's Choice	3.267
AQY212GSX	U4	Solid State Relay SOP6	PSI Power's Choice	3.267
AQY212GSX	U5	Solid State Relay SOP7	PSI Power's Choice	3.267
AQY212GSX	U6	Solid State Relay SOP8	PSI Power's Choice	3.267
AQY212GSX	U7	Solid State Relay SOP9	PSI Power's Choice	3.267
AQY212GSX	U8	Solid State Relay SOP10	PSI Power's Choice	3.267
AQY212GSX	U9	Solid State Relay SOP11	PSI Power's Choice	3.267
74HC4051DB-T	U10	ılog Multiplexer/Demultiplexer	PSI Power's Choice	0.339
UB232R	SV2 & SV3	UART-USB module	PSI Power's Choice	17.5
				73.56763

Appendix

IV. Verilog Modules

a. Rise Edge Detection

```
module
HS_rise_edge_detection(CLOCK,SIGNAL,HS_RISE_DETECTED,q,cnt,cnt_at_HS_rise_edge,cnt_en,usec_co
unt);

input CLOCK;
input SIGNAL;
output reg HS_RISE_DETECTED;
output reg q;
output reg [15:0] cnt;
output reg cnt_en;
input [15:0] usec_count;
output reg [15:0] cnt_at_HS_rise_edge;

always @ (posedge CLOCK) begin
        if(cnt_en == 1) begin
                cnt <= cnt + 16'b1;
                end
        if(cnt_en == 0) begin
                cnt <= 0;
                end
        if(SIGNAL == 1) begin
                q <= 1'b1;
                end
        if(SIGNAL == 0) begin
                q <= 1'b0;
                end
end
```

```
always @ (SIGNAL) begin
        if(~q & SIGNAL) begin
                HS_RISE_DETECTED <= 1'b1;
                cnt_at_HS_rise_edge <= usec_count;
                cnt_en <= 1;
                end
        if(cnt >= 16'd4) begin          //send pulse (43eceive 75nsec) to processor to signal rising
edge detection
                HS_RISE_DETECTED <= 1'b0;
                cnt_en <= 0;
                end
end
endmodule
```

b. PWM

```
module PWM(CLOCK, PWM_signal, vtrig);

input CLOCK;
output reg PWM_signal;
input vtrig;

reg [7:0] count;

always @ (posedge CLOCK)
        begin
        count <= count + 1'b1;
        if(vtrig == 1)
                begin
                if(count > 8'b11001101)
                        begin
                        PWM_signal <= 1'b0;
                        end
                else
                        begin
                        PWM_signal <= 1'b1;
                        end
                end
        if(vtrig == 0)
                begin
                if(count > 8'b00111101)
                        begin
                        PWM_signal <= 1'b0;
                        end
                else
```

```
                    begin
                    PWM_signal <= 1'b1;
                    end
                end
        end

endmodule
```

c. Fall Edge Detection

```
module fall_edge_detection(CLOCK,SIGNAL,FALL_DETECTED,q,cnt,cnt_at_fall_edge,cnt_en,usec_count);

input CLOCK;
input SIGNAL;
output reg FALL_DETECTED;
output reg q;
output reg [15:0] cnt;
output reg cnt_en;
input [15:0] usec_count;
output reg [15:0] cnt_at_fall_edge;

always @ (posedge CLOCK) begin
        if(cnt_en == 1) begin
                cnt <= cnt + 16'b1;
                end
        if(cnt_en == 0) begin
                cnt <= 0;
                end
        if(SIGNAL == 1) begin
                q <= 1'b1;
                end
        if(SIGNAL == 0) begin
                q <= 1'b0;
                end
end


always @ (~SIGNAL) begin
        if(q & ~SIGNAL) begin
                FALL_DETECTED <= 1'b1;
                cnt_at_fall_edge <= usec_count;
                cnt_en <= 1;
```

```
                end
        if(cnt >= 16'd4) begin              //send pulse (45eceive 75nsec) to processor to signal rising
edge detection
                FALL_DETECTED <= 1'b0;
                cnt_en <= 0;
                end
end



endmodule
```

d. Create Injection Event

```
module pull_low(output reg OD1, input btn, input wire clk, input [12:0] start_length, output reg
reset_counter, input [15:0] usec_count, output reg btn_received);

always @ (posedge clk)
begin
        if(btn == 1)
                begin
                reset_counter <= 1'b1;
                btn_received <= 1'b1;
                end
        if(btn_received == 1 & usec_count < start_length)
                begin
                reset_counter <= 1'b0;
                OD1 = 1'b0;
                end
        if(btn_received == 1 & usec_count >= start_length)
                begin
                OD1 <= 1'b1;
                btn_received <= 1'b0;
                reset_counter <= 1'b1;
                end
        if(btn == 0 & btn_received == 0)
                begin
                reset_counter <= 1'b0;
                end
end
endmodule
```

e. Counter

```
module 46eceive_counter(CLOCK, RESET_USEC, usec, nsec, record, cnt_at_adc);

input CLOCK;
input RESET_USEC;
input record;

output reg [15:0] usec;
output reg [5:0] nsec; //counts by 20nsecs (50MHz clock)
output reg [15:0] cnt_at_adc;
wire CLOCK;


always @ (posedge CLOCK) begin
        if (RESET_USEC == 1) begin
        nsec <= 6'b0;
        usec <= 16'b0;
        end
        else if (nsec == 6'b110001) begin //if count is 49 (980ns)
        nsec <= 6'b0; //reset 20nsec count
        usec <= usec + 16'b1; //46eceive46a usec count
        end
        else nsec <= nsec + 6'b1; //if a usec not reached, 46eceive46a 20ns counter
end

endmodule
```

V. NIOS II Code

```
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "altera_modular_adc.h"
```

```
#include “altera_avalon_uart.h”
#include “altera_avalon_uart_regs.h”
#include <string.h>
#include <sys/alt_irq.h>

alt_u32 adc_data[2];          //data array for ADC ch1 and ch2 values
alt_u32 adc_ch1_data_high[50];
alt_u32 adc_ch1_data_low[50];
alt_u32 adc_ch2_data_high[50];
alt_u32 adc_ch2_data_low[50];
int high_amplitude[10];
int low_amplitude[10];
//alt_u32 adc_ch2_data[500];
alt_u32 timings[50];
alt_u32 points[10];
alt_u32 time;
int adc_data_diff;
int n=0,i=1;
volatile int edge_occured = 0,fall_edge_occured = 0,HS_edge_occured =
0,HS_fall_edge_occured = 0;
alt_u32 usec_count;
int stop_time = 0;
int max_extend = 0;
char rx_data;
alt_u32 start_param[2] = {1711,1600};           //start signal length for
each module stored in array, identifier sent from GUI tells tester which
start_signal to use
int RISES[10];
int FALLS[10];
int HS_RISES[10];
int HS_FALLS[10];
int first_adc_ch2_data_high;

void send_start()
{
      //pulse PIO “low” to signal module “pull_low” to create injection event
            //int j;
            IOWR_ALTERA_AVALON_PIO_DATA(LOW_BASE,1);
            //for(j=0;j<1000;j++);
            IOWR_ALTERA_AVALON_PIO_DATA(LOW_BASE,0);
}

void start_ADC()
{
      adc_set_mode_run_once(MODULAR_ADC_0_SEQUENCER_CSR_BASE);
      //continuous run mode set in sequencer
      adc_interrupt_enable(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE);
      //enable interrupts on ADC
}

void read_rx()
{
      while((IORD_ALTERA_AVALON_UART_STATUS(UART_0_BASE) & 0x080) != 0x080);
      //wait for rx read ready (bit 8 is read ready bit of uart status reg
      rx_data = IORD_ALTERA_AVALON_UART_RXDATA(UART_0_BASE);
            //read rx for module identifier
}
```

```
void get_start_parameters()
{
	char num[4];
	char ext[4];
	int vtrig;
	int i;

	for(i=0;i<4;i++)
	{
		while((IORD_ALTERA_AVALON_UART_STATUS(UART_0_BASE) & 0x080) !=
0x080);		//wait for rx read ready (bit 8 is read ready bit of uart
status reg
		num[i] = IORD_ALTERA_AVALON_UART_RXDATA(UART_0_BASE);
		//read rx for module identifier
	}

	for(i=0;i<4;i++)
		{
			while((IORD_ALTERA_AVALON_UART_STATUS(UART_0_BASE) & 0x080)
!= 0x080);		//wait for rx read ready (bit 8 is read ready bit of uart
status reg
			ext[i] = IORD_ALTERA_AVALON_UART_RXDATA(UART_0_BASE);
			//read rx for module identifier
		}

	while((IORD_ALTERA_AVALON_UART_STATUS(UART_0_BASE) & 0x080) != 0x080);
	//wait for rx read ready (bit 8 is read ready bit of uart status reg
	vtrig = ((int)IORD_ALTERA_AVALON_UART_RXDATA(UART_0_BASE))-48;

	stop_time = (((int)num[0]-48)*1000) + (((int)num[1]-48)*100) +
(((int)num[2]-48)*10) + ((int)num[3]-48);
	max_extend = (((int)ext[0]-48)*1000) + (((int)ext[1]-48)*100) +
(((int)ext[2]-48)*10) + ((int)ext[3]-48);

	IORD_ALTERA_AVALON_UART_RXDATA(UART_0_BASE);		//Dump RX buffer
b/c if character is misread there could be bits leftover to scew next reading
	IOWR_ALTERA_AVALON_PIO_DATA(START_LENGTH_BASE,stop_time);
	IOWR_ALTERA_AVALON_PIO_DATA(TRIGGER_VOLTAGE_BASE,vtrig);

}


void delay(x)
{
	//approximately x microseconds long

	int i,j;

	for(j=0;j<x;j++)
		{for(i=0;i<4;i++)
			{asm("nop");}		//assembly language no-op inserted
so compiler doesn't optimize out the delay loop
		}
}
```

```
static void rise_edge_ISR( void* context)
{
//IOWR_ALTERA_AVALON_PIO_DATA(HIGH_BASE,1);
adc_start(MODULAR_ADC_0_SEQUENCER_CSR_BASE);
RISES[edge_occured] = IORD_ALTERA_AVALON_PIO_DATA(RISE_EDGE_TIME_BASE);
adc_wait_for_interrupt(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE);
alt_adc_word_read(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE, adc_data,
MODULAR_ADC_0_SAMPLE_STORE_CSR_CSD_LENGTH );    //empty ADC sample store
register into adc_data array
adc_ch1_data_high[edge_occured] = adc_data[0];
adc_clear_interrupt_status(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE);
edge_occured++;
//IOWR_ALTERA_AVALON_PIO_DATA(HIGH_BASE,0);
//Acknowledge the IRQ : Clear the edgecapture register by writing to it
(individual bit setting is disabled so any write to the edge capture reg
clears it)
IOWR_ALTERA_AVALON_PIO_EDGE_CAP(RISE_EDGE_DETECTION_BASE,
RISE_EDGE_DETECTION_BIT_CLEARING_EDGE_REGISTER);
}

static void fall_edge_ISR( void* context)
{
//IOWR_ALTERA_AVALON_PIO_DATA(HIGH_BASE,1);
adc_start(MODULAR_ADC_0_SEQUENCER_CSR_BASE);
FALLS[fall_edge_occured] = IORD_ALTERA_AVALON_PIO_DATA(FALL_EDGE_TIME_BASE);
adc_wait_for_interrupt(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE);
alt_adc_word_read(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE, adc_data,
MODULAR_ADC_0_SAMPLE_STORE_CSR_CSD_LENGTH );    //empty ADC sample store
register into adc_data array
adc_ch1_data_low[fall_edge_occured] = adc_data[0];
      if(fall_edge_occured == 0)
      {
            adc_ch2_data_high[0] = adc_data[1];
      }
adc_clear_interrupt_status(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE);
fall_edge_occured++;
//IOWR_ALTERA_AVALON_PIO_DATA(HIGH_BASE,0);
//Acknowledge the IRQ : Clear the edgecapture register by writing to it
(individual bit setting is disabled so any write to the edge capture reg
clears it)
IOWR_ALTERA_AVALON_PIO_EDGE_CAP(FALL_EDGE_DETECTION_BASE,
FALL_EDGE_DETECTION_BIT_CLEARING_EDGE_REGISTER);
}

static void HS_rise_edge_ISR( void* context)
{
      //IOWR_ALTERA_AVALON_PIO_DATA(HIGH_BASE,1);
      if(HS_edge_occured == 0)
      {
            HS_RISES[HS_edge_occured] =
IORD_ALTERA_AVALON_PIO_DATA(HS_RISE_EDGE_TIME_BASE);
      }
      else
      {
            adc_start(MODULAR_ADC_0_SEQUENCER_CSR_BASE);
            HS_RISES[HS_edge_occured] =
IORD_ALTERA_AVALON_PIO_DATA(HS_RISE_EDGE_TIME_BASE);
```

```
            adc_wait_for_interrupt(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE);
            alt_adc_word_read(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE, adc_data,
MODULAR_ADC_0_SAMPLE_STORE_CSR_CSD_LENGTH );    //empty ADC sample store
register into adc_data array
            adc_ch2_data_high[HS_edge_occured] = adc_data[1];
            adc_clear_interrupt_status(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE);
      }


      HS_edge_occured++;
      //IOWR_ALTERA_AVALON_PIO_DATA(HIGH_BASE,0);
      //Acknowledge the IRQ : Clear the edgecapture register by writing to it
(individual bit setting is disabled so any write to the edge capture reg
clears it)
      IOWR_ALTERA_AVALON_PIO_EDGE_CAP(HS_RISE_EDGE_DETECTION_BASE,
HS_RISE_EDGE_DETECTION_BIT_CLEARING_EDGE_REGISTER);
}

static void HS_fall_edge_ISR( void* context)
{
      adc_start(MODULAR_ADC_0_SEQUENCER_CSR_BASE);
      HS_FALLS[HS_fall_edge_occured] =
IORD_ALTERA_AVALON_PIO_DATA(HS_FALL_EDGE_TIME_BASE);
      adc_wait_for_interrupt(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE);
      alt_adc_word_read(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE, adc_data,
MODULAR_ADC_0_SAMPLE_STORE_CSR_CSD_LENGTH );    //empty ADC sample store
register into adc_data array
      adc_ch2_data_low[HS_fall_edge_occured] = adc_data[1];
      adc_clear_interrupt_status(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE);
      HS_fall_edge_occured++;
      //Acknowledge the IRQ : Clear the edgecapture register by writing to it
(individual bit setting is disabled so any write to the edge capture reg
clears it)
      IOWR_ALTERA_AVALON_PIO_EDGE_CAP(HS_FALL_EDGE_DETECTION_BASE,
HS_FALL_EDGE_DETECTION_BIT_CLEARING_EDGE_REGISTER);

}

int main()
{
      int i,low_start,low_stop;

      ////////////initialize ADC data array/////////////
      for(i=0;i<2;i++)
            adc_data[i] = 0xDEADBEEF;

      ////////////////////////////register ISRs for edge detection
PIOs//////////////////////////////////
      alt_ic_isr_register(RISE_EDGE_DETECTION_IRQ_INTERRUPT_CONTROLLER_ID,
//alt_u32 ic_id
                                    RISE_EDGE_DETECTION_IRQ, //alt_u32 irq
                                    rise_edge_ISR, //alt_isr_func isr
                                    NULL,
                                    NULL);

      alt_ic_isr_register(FALL_EDGE_DETECTION_IRQ_INTERRUPT_CONTROLLER_ID,
//alt_u32 ic_id
```

```
                                    FALL_EDGE_DETECTION_IRQ, //alt_u32 irq
                                    fall_edge_ISR, //alt_isr_func isr
                                    NULL,
                                    NULL);

      alt_ic_isr_register(HS_RISE_EDGE_DETECTION_IRQ_INTERRUPT_CONTROLLER_ID,
//alt_u32 ic_id
                                    HS_RISE_EDGE_DETECTION_IRQ, //alt_u32 irq
                                    HS_rise_edge_ISR, //alt_isr_func isr
                                    NULL,
                                    NULL);

      alt_ic_isr_register(HS_FALL_EDGE_DETECTION_IRQ_INTERRUPT_CONTROLLER_ID,
//alt_u32 ic_id
                                    HS_FALL_EDGE_DETECTION_IRQ, //alt_u32 irq
                                    HS_fall_edge_ISR, //alt_isr_func isr
                                    NULL,
                                    NULL);
      ////////////////////////////////////////////////////////////////////////
////////////////////////////

      //ensure start signal to create injection event is off
      IOWR_ALTERA_AVALON_PIO_DATA(LOW_BASE,0);

      // ---------------ensure interrupt flags are cleared-------------------
--------

      IOWR_ALTERA_AVALON_PIO_DATA(HIGH_BASE,0);

      while(1)
      {
            while((IORD_ALTERA_AVALON_UART_STATUS(UART_0_BASE) & 0x080) !=
0x080);           //wait for rx read ready (bit 8 is read ready bit of uart
status reg

            if ('H' == IORD_ALTERA_AVALON_UART_RXDATA(UART_0_BASE))
                  //read rx for module identifier
            {
                  get_start_parameters();       //reads next 3 chars to get
start parameter number
            }

            if(IORD_ALTERA_AVALON_UART_RXDATA(UART_0_BASE) == '@')
      //start received from GUI
            {

      alt_ic_irq_disable(RISE_EDGE_DETECTION_IRQ_INTERRUPT_CONTROLLER_ID,
RISE_EDGE_DETECTION_IRQ);

      alt_ic_irq_disable(FALL_EDGE_DETECTION_IRQ_INTERRUPT_CONTROLLER_ID,
FALL_EDGE_DETECTION_IRQ);

      alt_ic_irq_disable(HS_RISE_EDGE_DETECTION_IRQ_INTERRUPT_CONTROLLER_ID,
HS_RISE_EDGE_DETECTION_IRQ);

      alt_ic_irq_disable(HS_FALL_EDGE_DETECTION_IRQ_INTERRUPT_CONTROLLER_ID,
HS_FALL_EDGE_DETECTION_IRQ);
```

```
                  //reset edge timing and ADC data
                  for(i=0;i<10;i++)
                  {
                        RISES[i] = 0;
                        FALLS[i] = 0;
                        HS_RISES[i] = 0;
                        HS_FALLS[i] = 0;
                        adc_ch1_data_high[i] = 0;
                        adc_ch1_data_low[i] = 0;
                        adc_ch2_data_high[i] = 0;
                        adc_ch2_data_low[i] = 0;
                  }
                  edge_occured = 0;             //reset the counter of the
rising edge data array
                  fall_edge_occured = 0;        //reset the counter of the
falling edge data array
                  HS_edge_occured = 0;          //reset the counter of the
high signal rising edge data array
                  HS_fall_edge_occured = 0;     //reset the counter of the
high signal falling edge data array
                  start_ADC();
                  send_start();

                  //delay enable of edge detection interrupts until after
injection event is finished
                  delay(stop_time);

                  //IOWR_ALTERA_AVALON_PIO_DATA(HIGH_BASE,1);

                  // ---------------ensure previous interrupt flags are
cleared---------------------------
                  IOWR_ALTERA_AVALON_PIO_EDGE_CAP(RISE_EDGE_DETECTION_BASE,
0);
                  IOWR_ALTERA_AVALON_PIO_EDGE_CAP(FALL_EDGE_DETECTION_BASE,
0);

      IOWR_ALTERA_AVALON_PIO_EDGE_CAP(HS_RISE_EDGE_DETECTION_BASE, 0);

      IOWR_ALTERA_AVALON_PIO_EDGE_CAP(HS_FALL_EDGE_DETECTION_BASE, 0);

                  // ---------------enable edge detection interrupt ports and
ISRs---------------------------

      alt_ic_irq_enable(RISE_EDGE_DETECTION_IRQ_INTERRUPT_CONTROLLER_ID,
RISE_EDGE_DETECTION_IRQ);

      alt_ic_irq_enable(FALL_EDGE_DETECTION_IRQ_INTERRUPT_CONTROLLER_ID,
FALL_EDGE_DETECTION_IRQ);

      alt_ic_irq_enable(HS_RISE_EDGE_DETECTION_IRQ_INTERRUPT_CONTROLLER_ID,
HS_RISE_EDGE_DETECTION_IRQ);

      alt_ic_irq_enable(HS_FALL_EDGE_DETECTION_IRQ_INTERRUPT_CONTROLLER_ID,
HS_FALL_EDGE_DETECTION_IRQ);
```

```
                //Setting interrupt mask register to 1 enables interrupts
                IOWR_ALTERA_AVALON_PIO_IRQ_MASK(RISE_EDGE_DETECTION_BASE,
RISE_EDGE_DETECTION_CAPTURE);
                IOWR_ALTERA_AVALON_PIO_IRQ_MASK(FALL_EDGE_DETECTION_BASE,
FALL_EDGE_DETECTION_CAPTURE);

      IOWR_ALTERA_AVALON_PIO_IRQ_MASK(HS_RISE_EDGE_DETECTION_BASE,
HS_RISE_EDGE_DETECTION_CAPTURE);

      IOWR_ALTERA_AVALON_PIO_IRQ_MASK(HS_FALL_EDGE_DETECTION_BASE,
HS_FALL_EDGE_DETECTION_CAPTURE);

                //IOWR_ALTERA_AVALON_PIO_DATA(HIGH_BASE,0);


                //delay sending data until after max extend to allow all
possible edges to be detected
                delay(max_extend);


                alt_putstr("data_start\n");         //signals gui to start
reading rx for data

                        for(i=0;i<10;i++)
                              {
                              printf("%lu\n",RISES[i]);
                              }
                        for(i=0;i<10;i++)
                              {
                              printf("%lu\n",FALLS[i]);
                              }
                        for(i=0;i<10;i++)
                              {
                              printf("%lu\n",adc_ch1_data_high[i]);
                              }
                        for(i=0;i<10;i++)
                              {
                              printf("%lu\n",adc_ch1_data_low[i]);
                              }
                        for(i=0;i<10;i++)
                              {
                              printf("%lu\n",HS_RISES[i]);
                              }
                        for(i=0;i<10;i++)
                              {
                              printf("%lu\n",HS_FALLS[i]);
                              }
                        for(i=0;i<10;i++)
                              {
                              printf("%lu\n",adc_ch2_data_high[i]);
                              }
                        for(i=0;i<10;i++)
                              {
                              printf("%lu\n",adc_ch2_data_low[i]);
                              }
            }
```

```
        }
        return 0;
}
```

VI. VISUAL STUDIOS GUI

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using System.IO;
using System.Windows.Forms.DataVisualization.Charting;
using Microsoft.VisualBasic;


namespace WindowsFormsApp1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      System.IO.Ports.SerialPort SerialPort1;        //define serial port
      int stopClicked = 0;
      float[] adcHIGH = new float[50];
      float[] adcLOW = new float[50];
      float[] HS_adcHIGH = new float[50];
      float[] HS_adcLOW = new float[50];
      int[] rise_edge = new int[50];
      int[] fall_edge = new int[50];
      int[] HS_rise_edge = new int[50];
      int[] HS_fall_edge = new int[50];
      string[] csv_module_name;
      string[] csv_start_sig;
      string[] csv_TS1_start;
      string[] csv_TS1_stop;
      string[] csv_TS2_start;
      string[] csv_TS2_stop;
      string[] csv_hi_ext;
```

```
    string[] csv_lo_ext;
    string[] csv_trig_volt;
    int no_of_modules,identifier;
    int tester_running = 0;

    LineAnnotation TS1_start = new VerticalLineAnnotation();
    LineAnnotation TS1_stop = new VerticalLineAnnotation();
    LineAnnotation TS2_start = new VerticalLineAnnotation();
    LineAnnotation TS2_stop = new VerticalLineAnnotation();
    LineAnnotation hi_ext = new VerticalLineAnnotation();
    LineAnnotation lo_ext = new VerticalLineAnnotation();
    Label TS1_start_label = new Label();
    Label TS1_stop_label = new Label();
    Label TS2_start_label = new Label();
    Label TS2_stop_label = new Label();
    Label hi_ext_label = new Label();
    Label lo_ext_label = new Label();

    private void Form1_Load(object sender, EventArgs e)
    {
      ports.Text = "COM PORTS";


      try
      {
        using (StreamReader str_rdr = new
StreamReader(@"C:\Users\jmgal\Documents\module_info_v6.csv"))
        {
          string s;
          int i;

          s = str_rdr.ReadLine();
          csv_module_name = s.Split(',');
          no_of_modules = csv_module_name.Length;
          s = str_rdr.ReadLine();
          csv_start_sig = s.Split(',');
          s = str_rdr.ReadLine();
          csv_TS1_start = s.Split(',');
          s = str_rdr.ReadLine();
          csv_TS1_stop = s.Split(',');
          s = str_rdr.ReadLine();
          csv_TS2_start = s.Split(',');
          s = str_rdr.ReadLine();
          csv_TS2_stop = s.Split(',');
          s = str_rdr.ReadLine();
          csv_hi_ext = s.Split(',');
          s = str_rdr.ReadLine();
          csv_lo_ext = s.Split(',');
```

```
      s = str_rdr.ReadLine();
      csv_trig_volt = s.Split(',');

      for (i = 0; i < no_of_modules; i++)
      {
         module.Items.Add(csv_module_name[i]);
      }
   }
}
catch
{
   MessageBox.Show("There was a problem loading the module data CSV file",
            "Invalid File Path",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
}
module.Text = "MODULES";

//chart characteristics that won't change, regardless of the module
chart1.ChartAreas[0].AxisX.Minimum = 0;
chart1.ChartAreas[0].AxisY.Maximum = 13;
chart1.ChartAreas[0].AxisY.Minimum = 0;
chart1.Series[0].ChartType = SeriesChartType.Line;
chart1.Series[1].ChartType = SeriesChartType.Line;
chart1.Series[0].Color = Color.Green;
chart1.Series[1].Color = Color.Red;
chart1.Series[0].BorderWidth = 3;

TS1_start.AxisX = chart1.ChartAreas[0].AxisX;
TS1_start.IsInfinitive = true;
TS1_start.LineWidth = 2;
TS1_start.LineColor = Color.Orange;
TS1_start.ClipToChartArea = chart1.ChartAreas[0].Name;
TS1_stop.AxisX = chart1.ChartAreas[0].AxisX;
TS1_stop.IsInfinitive = true;
TS1_stop.LineWidth = 2;
TS1_stop.LineColor = Color.Orange;
TS1_stop.ClipToChartArea = chart1.ChartAreas[0].Name;
TS2_start.AxisX = chart1.ChartAreas[0].AxisX;
TS2_start.IsInfinitive = true;
TS2_start.LineWidth = 2;
TS2_start.LineColor = Color.Orange;
TS2_start.ClipToChartArea = chart1.ChartAreas[0].Name;
TS2_stop.AxisX = chart1.ChartAreas[0].AxisX;
TS2_stop.IsInfinitive = true;
TS2_stop.LineWidth = 2;
TS2_stop.LineColor = Color.Orange;
TS2_stop.ClipToChartArea = chart1.ChartAreas[0].Name;
```

```
    hi_ext.AxisX = chart1.ChartAreas[0].AxisX;
    hi_ext.IsInfinitive = true;
    hi_ext.LineWidth = 2;
    hi_ext.LineColor = Color.Orange;
    hi_ext.ClipToChartArea = chart1.ChartAreas[0].Name;
    lo_ext.AxisX = chart1.ChartAreas[0].AxisX;
    lo_ext.IsInfinitive = true;
    lo_ext.LineWidth = 2;
    lo_ext.LineColor = Color.Orange;
    lo_ext.ClipToChartArea = chart1.ChartAreas[0].Name;

    TS1_start_label.Text = "Tri-State Start";
    TS1_stop_label.Text = "Tri-State End";
    TS2_start_label.Text = "Tri-State Start";
    TS2_stop_label.Text = "Tri-State End";
    hi_ext_label.Text = "Max Extend";
    lo_ext_label.Text = "Med Extend";
}



private void start_Click(object sender, EventArgs e)
{
    stopClicked = 0;
    stop.Enabled = true;
    string input,timing;
    int point;
    int h,i,n,m,l,lvl_cnt=0;
    int adc_max, adc_min, lastHIGH = 0, HS_lastFALL = 0;


    //for (n = 0; n < 500; n++)
    for (n = 0; n < 50; n++)
    {

        for (m = 0; m < 65536; m++)
        { for (l = 0; l < 100; l++) ; }

        chart1.Series[0].Points.Clear();
        chart1.Series[1].Points.Clear();

        SerialPort1.Write("@");        //0 = ascii 48 ➔ signals tester to start testing



        input = SerialPort1.ReadLine();

        while (input != "data_start")
```

```
        {
            input = SerialPort1.ReadLine();
            //SerialPort1.Write("@");       //0 = ascii 48 ➔ signals tester to start testing || continuously sent incase tester misreads '0', preventing the GUI from being stuck in a loop waiting for data that will never come
        }

        //retrieve LS rising edge times from tester
        for (i = 0; i < 10; i++)
        {
            input = SerialPort1.ReadLine();
            rise_edge[i] = Convert.ToInt32(input);

        }

        //retrieve LS falling edge times from tester
        for (i = 0; i < 10; i++)
        {
            input = SerialPort1.ReadLine();
            fall_edge[i] = Convert.ToInt32(input);

        }

        //retrieve LS ADC high amplitudes from tester
        for (i = 0; i < 10; i++)
        {
            input = SerialPort1.ReadLine();
            adcHIGH[i] = Convert.ToInt32(input);
        }

        //retrieve LS ADC low amplitudes from tester
        for (i = 0; i < 10; i++)
        {
            input = SerialPort1.ReadLine();
            adcLOW[i] = Convert.ToInt32(input);

        }

        //retrieve HS rising edge times from tester
        for (i = 0; i < 10; i++)
        {
            input = SerialPort1.ReadLine();
            HS_rise_edge[i] = Convert.ToInt32(input);

        }

        //retrieve HS falling edge times from tester
        for (i = 0; i < 10; i++)
```

```
{
   input = SerialPort1.ReadLine();
   HS_fall_edge[i] = Convert.ToInt32(input);

}

//retrieve HS ADC high amplitudes from tester
for (i = 0; i < 10; i++)
{
   input = SerialPort1.ReadLine();
   HS_adcHIGH[i] = Convert.ToInt32(input);
}

//retrieve HS ADC low amplitudes from tester
for (i = 0; i < 10; i++)
{
   input = SerialPort1.ReadLine();
   HS_adcLOW[i] = Convert.ToInt32(input);

}


/*
SerialPort1.Write("#");
input = SerialPort1.ReadLine();

while (input != "data_start_2")
{
   input = SerialPort1.ReadLine();
}

//retrieve high-side rising edge times from tester
for (i = 0; i < 10; i++)
{
   input = SerialPort1.ReadLine();
   HS_rise_edge[i] = Convert.ToInt32(input);
}

//retrieve high_side falling edge times from tester
for (i = 0; i < 10; i++)
{
   input = SerialPort1.ReadLine();
   HS_fall_edge[i] = Convert.ToInt32(input);
}
*/

/*
adc_max = adcY[0];
```

```
adc_min = adcY[0];
*/

for (i = 0; i < 10; i++)
{
   adcHIGH[i] = ((adcHIGH[i] * 12) / 4096);
   adcLOW[i] = ((adcLOW[i] * 12) / 4096);
   HS_adcHIGH[i] = ((HS_adcHIGH[i] * 12) / 4096);
   HS_adcLOW[i] = ((HS_adcLOW[i] * 12) / 4096);
}



//graph points in order from left to right so the visual studios chart doesn't try to fuck it up
//could use h after the if 60eceive60a b/c if statements validates equality
for (i = 0; i < Convert.ToInt16(csv_hi_ext[identifier]) + 100; i++)
{
   for (h = 0; h < 10; h++)
   {
      if (rise_edge[h] == i && rise_edge[h] != 0)
      {
         if (h == 0)
         {
            chart1.Series[0].Points.AddXY(rise_edge[h], 0);
            chart1.Series[0].Points.AddXY(rise_edge[h], adcHIGH[h]);
         }
         else
         {
            chart1.Series[0].Points.AddXY(rise_edge[h], adcLOW[h-1]);
            chart1.Series[0].Points.AddXY(rise_edge[h], adcHIGH[h]);
            lastHIGH = h;
         }
      }
      if (fall_edge[h] == i && fall_edge[h] != 0)
      {
         if (h == 0)
         {
            chart1.Series[0].Points.AddXY(rise_edge[h], adcHIGH[h]);
            chart1.Series[0].Points.AddXY(rise_edge[h], adcLOW[h]);
         }
         else
         {
            chart1.Series[0].Points.AddXY(fall_edge[h], adcHIGH[h]);
            chart1.Series[0].Points.AddXY(fall_edge[h], adcLOW[h]);
         }
      }
   }
}
```

```
            chart1.Series[0].Points.AddXY(Convert.ToInt16(csv_hi_ext[identifier]) + 99,
adcHIGH[lastHIGH]);

            for (i = 0; i < Convert.ToInt16(csv_hi_ext[identifier]) + 100; i++)
            {
               for (h = 0; h < 10; h++)
               {
                  if (HS_rise_edge[h] == i && HS_rise_edge[h] != 0)
                  {
                     if (h == 0)
                     {
                        chart1.Series[1].Points.AddXY(HS_rise_edge[h], 0);
                        chart1.Series[1].Points.AddXY(HS_rise_edge[h], HS_adcHIGH[h]);
                     }
                     else
                     {
                        chart1.Series[1].Points.AddXY(HS_rise_edge[h], HS_adcLOW[h – 1]);
                        chart1.Series[1].Points.AddXY(HS_rise_edge[h], HS_adcHIGH[h]);
                     }
                  }
                  if (HS_fall_edge[h] == i && HS_fall_edge[h] != 0)
                  {
                     if (h == 0)
                     {
                        chart1.Series[1].Points.AddXY(HS_fall_edge[h], HS_adcHIGH[h]);
                        chart1.Series[1].Points.AddXY(HS_fall_edge[h], HS_adcLOW[h]);
                     }
                     else
                     {
                        chart1.Series[1].Points.AddXY(HS_fall_edge[h], HS_adcHIGH[h]);
                        chart1.Series[1].Points.AddXY(HS_fall_edge[h], HS_adcLOW[h]);
                        HS_lastFALL = h;
                     }
                  }
               }
            }
            chart1.Series[1].Points.AddXY(Convert.ToInt16(csv_hi_ext[identifier]) + 99,
HS_adcLOW[HS_lastFALL]);

            chart1.Update();
            lvl_cnt = 0;
            lastHIGH = 0;
         }
      }


      private void ports_TextChanged(object sender, EventArgs e)
      {
```

```
        if (ports.Text != “COM PORTS”)
        {
            SerialPort1 = new System.IO.Ports.SerialPort(ports.Text,                    //COM port is read
from combobox selection
                                        115200,                     //baud rate
                                        System.IO.Ports.Parity.None,       //no parity bits
                                        8,                          //8bit 62eceive62ation
                                        System.IO.Ports.StopBits.One);     //one stop bit
          try
          {
            SerialPort1.Open();        //open serial connection on selected COM
            //SerialPort1.Write(“hello, is it me you’re looking for?”);
          }
          catch
          {
            MessageBox.Show(“The selected port is busy or available\nTry Again”,
                    “Port Not Found”,
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
          }
        }
    }

    private void ports_DropDown(object sender, EventArgs e)
    {
      string[] available_ports;
      available_ports = SerialPort.GetPortNames();    //find ports
      ports.Items.Clear();                    //clear old ports from combobox
      ports.Items.AddRange(available_ports);         //add new ports to combobox
    }


    ////////////////////////////////////////////////////// NEW MODULE SELECTED
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
    private void module_SelectedIndexChanged(object sender, EventArgs e)
    {
      int i;
      string input;

      if (module.Text != “MODULES”)
      {
        //pull identifier from the newly selected module key
        //i = (string)module.SelectedValue;
        //send module identifier to tester
        //SerialPort1.Write(i);            //this is what causes the module list to display both key and
value WHY? IDFK
```

```
for (i = 0; i < no_of_modules; i++)
{
   if (module.Text.Equals(csv_module_name[i]))
   {
      //signal tester to 63eceive start conditions
      SerialPort1.Write("H");
      //send start length in 4 digit format
      switch (csv_start_sig[i].Length)
      {
         case 2:
            SerialPort1.Write("0");
            SerialPort1.Write("0");
            SerialPort1.Write(csv_start_sig[i]);
            break;
         case 3:
            SerialPort1.Write("0");
            SerialPort1.Write(csv_start_sig[i]);
            break;
         case 4:
            SerialPort1.Write(csv_start_sig[i]);
            break;
      }
      //send high extend in 4 digit format
      switch (csv_hi_ext[i].Length)
      {
         case 2:
            SerialPort1.Write("0");
            SerialPort1.Write("0");
            SerialPort1.Write(csv_hi_ext[i]);
            break;
         case 3:
            SerialPort1.Write("0");
            SerialPort1.Write(csv_hi_ext[i]);
            break;
         case 4:
            SerialPort1.Write(csv_hi_ext[i]);
            break;
      }

      //send trigger voltage to tester
      if(csv_trig_volt[i] == "3.3")
      {
         SerialPort1.Write("0");
      }
      if (csv_trig_volt[i] == "5")
      {
         SerialPort1.Write("1");
      }
```

```
            identifier = i;

            break;
          }
        }

        textBox1.Text = csv_start_sig[identifier];
        textBox2.Text = csv_hi_ext[identifier];

        chart1.ChartAreas[0].AxisX.Maximum = Convert.ToInt16(csv_hi_ext[identifier]) + 100;

        chart1.Update();

        chart1.Annotations.Clear();



        TS1_start.X = Convert.ToInt16(csv_TS1_start[identifier]);
        chart1.Annotations.Add(TS1_start);
        TS1_start_label.Location = new Point(Convert.ToInt16(csv_TS1_start[identifier]), 0);


        TS1_stop.X = Convert.ToInt16(csv_TS1_stop[identifier]);
        chart1.Annotations.Add(TS1_stop);
        TS1_stop_label.Location = new Point(Convert.ToInt16(csv_TS1_stop[identifier]), 0);


        TS2_start.X = Convert.ToInt16(csv_TS2_start[identifier]);
        chart1.Annotations.Add(TS2_start);
        TS2_start_label.Location = new Point(Convert.ToInt16(csv_TS2_start[identifier]), 0);


        TS2_stop.X = Convert.ToInt16(csv_TS2_stop[identifier]);
        chart1.Annotations.Add(TS2_stop);
        TS2_stop_label.Location = new Point(Convert.ToInt16(csv_TS2_stop[identifier]), 0);

        //default to high setting when new module is selected
        stop.Text = “HIGH”;
        hi_ext.X = Convert.ToInt16(csv_hi_ext[identifier]);
        chart1.Annotations.Add(hi_ext);
        hi_ext_label.Location = new Point(Convert.ToInt16(csv_hi_ext[identifier]), 1);


        lo_ext.X = Convert.ToInt16(csv_lo_ext[identifier]);

      }
    }
```

```
    private void label1_Click(object sender, EventArgs e)
    {

    }

    private void stop_Click(object sender, EventArgs e)
    {
      if (stop.Text == "HIGH")
      {
        stop.Text = "LOW";
        chart1.Annotations.Remove(hi_ext);
        chart1.Annotations.Add(lo_ext);
      }
      else if (stop.Text == "LOW")
      {
        stop.Text = "HIGH";
        chart1.Annotations.Remove(lo_ext);
        chart1.Annotations.Add(hi_ext);
      }
    }

  }

}
```

References

"IEEE Code of Ethics." *IEEE - Advancing Technology for Humanity*, www.ieee.org/about/corporate/governance/p7-8.html.

"UM1724 User Manual." STMicroelectronics https://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf.