Spring 2022

Removing Physical Presence Requirements for a Remote and Automated World - API Controlled Patch Panel for Conformance Testing

Hunter George Wells
University of New Hampshire, Durham

Follow this and additional works at: https://scholars.unh.edu/honors

Part of the Digital Circuits Commons, Digital Communications and Networking Commons, Electrical and Electronics Commons, Other Electrical and Computer Engineering Commons, Signal Processing Commons, and the Systems and Communications Commons

Recommended Citation

This Senior Honors Thesis is brought to you for free and open access by the Student Scholarship at University of New Hampshire Scholars' Repository. It has been accepted for inclusion in Honors Theses and Capstones by an authorized administrator of University of New Hampshire Scholars' Repository. For more information, please contact Scholarly.Communication@unh.edu.
Removing Physical Presence Requirements for a Remote and Automated World - API Controlled Patch Panel for Conformance Testing

ECE Student: Hunter G. Wells
Faculty Advisor: Professor Qiaoyan Yu

Department of Electrical and Computer Engineering
University of New Hampshire
April 24th, 2022

Advisor Signature of Approval

Signature  

Date 05/17/2022
Abstract:

Quality assurance test engineers at the UNH-InterOperability Lab must run tests that require driving and monitoring a selection of DC signals. While the number of signals is numerous, there are limited ports on the test equipment, and only a few signals need patching for any given test. The selection of signals may vary between the 209 different tests and must be re-routed frequently. Currently, testers must leave their desk to manually modify the test setup in another room. This posed a considerable issue at the onset of the COVID-19 Pandemic when physical access was not possible. In order to enable future remote testing and human-free automation, there is a need for a remote patching device. Such a device was designed, constructed, and deployed. The process of designing the product involved defining design constraints, developing conceptual designs, selecting components and generating an electrical schematic, prototyping and iterating PCB versions, and final deployment and testing of the product.
Table of Contents

Advisor Signature of Approval ........................................... 2
Abstract: ............................................................................... 3
Table of Contents ................................................................. 4
Project Overview: ................................................................. 5
Literature Review and Background: ......................................... 8
Project Goals: ....................................................................... 9
Methods: ............................................................................... 10
  Design Constraints ............................................................ 10
  Conceptual Design #1 ......................................................... 11
  Component Selection and Testing ......................................... 13
  Conceptual Design #2 ......................................................... 14
  Component List ................................................................. 17
  Schematic ......................................................................... 18
  PCB development and 1st Prototype .................................... 21
  2nd Prototype ................................................................... 25
  3rd Prototype ................................................................... 27
  User Interface and Serial Communication Control .................. 29
Results: ............................................................................... 37
Future Work: ....................................................................... 41
Timeline: ............................................................................. 42
Budget: ............................................................................... 43
Discussion and Conclusions: ............................................... 45
Acknowledgments ............................................................... 47
References: ......................................................................... 48
 Appendices: ....................................................................... 50
  Appendix 1 - Original Senior Project Gantt Chart .................. 50
  Appendix 2 - Microcontroller Code ....................................... 50
  Appendix 3 - 100BASE-T1 Signal Matrices ............................ 72
Project Overview:

The current COVID-19 pandemic has highlighted a need for employee productivity to be less reliant on physical presence to manually manipulate tools and apparatuses. When employees are unable to make physical changes to an apparatus used to achieve their primary product or service, the product or service is heavily obstructed. According to Dr. Kaushik [1], “...COVID-19 has impacted human life and business organizations at a global level and challenged employers, HR managers and consultants to re-think, redesign and re-imagine in a new way of transition and emergence of the remote work environment. A paradigm shift has taken place at the workplace in many ways …” More employees and entire businesses are likely to embrace and utilize remote or hybrid models. Even after the COVID-19 pandemic subsides, a greater portion of employees will work from home as compared to before the COVID-19 pandemic and require methods of completing tasks that once relied on physical proximity to a device or apparatus.

This project solves the physical presence limitation as it concerns conformance testing at the UNH-IOL. Prior to this project, testing required significant manual manipulation and remained unprotected from disruptions to lab access. Globally, companies have turned to technology to “safeguard against current and potential future pandemics” [2] [3]. The UNH-IOL is no different and is pursuing similar technological safeguards. The signal patching device designed, will help to alleviate the need for human intervention. The tool provides defense against the current COVID-19 pandemic, remains robust to future disruptions and events, and enables the adoption of remote or hybrid models of working in the future.

Some employees at the UNH-IOL were prevented from completing their job due to the remote limitations presented by the COVID-19 pandemic. It became extremely clear that some positions would benefit from a tool that would remove the need for an employee to be physically
present. In these positions, an employee is required to perform a series of conformance tests. Importantly, the employee (a.k.a. Technician or tester) must manually modify the setup between tests, an impossible task without being in the same room as the setup.

Further, this testing service consumes a significant portion of an employee's time. To perform all possible tests in this service might take an employee between 3 to 8 weeks. An employee will test 9 to 10 devices in a year. In an effort to reduce the human time cost of the service, improve testing quality and consistency, and alleviate employees of tedium, there is a push to automate this particular set of tests. Automation of these tests would reduce the time to complete tests to about 2 hours of setup, and 6 hours of processing by an automation engine. This could allow an employee to set up nearly 4 devices in a workday and have results waiting by the start of business the next day. Full automation will require development to eliminate the need for humans to control and analyze tests, and will also require the elimination of the need for humans to manipulate the test setup between tests. This project addressed the elimination of reliance on human manipulation.

Conformance testing is carried out by interfacing with an ASIC through signals exposed on a PCB. The apparatus in question will be referred to as the test setup, and consists of the ASIC-PCB Device Under Test (DUT), oscilloscope, test driver tool, and electrical cabling to connect all equipment together. Each DUT must be subjected to various sequences of stimuli and monitored for the response, a process that composes testing in this scenario. DUTs are stimulated and monitored (tested), in part, through a selection of digital DC signals carried between the equipment in the test setup by the cabling. There are 209 tests that a DUT could be subjected to. Each of these tests requires only a portion of available digital DC signals, and the Oscilloscope and test tool have limited numbers of ports for driving or monitoring these signals. However, all available signals are used collectively among the tests. This creates a need for the cabling to be disconnected, moved, and reconnected between tests to ensure that the needed signals are patched to the correct ports on the equipment.
Previously, the testing service was reliant on a human to manually adjust these connections among the DUT and the test equipment. There was a need to remove this dependence on physical presence so that the testing service could be more resilient to uncontrollable events (such as future epidemiological events) that would prevent a technician from accessing the test setup. The removal of such dependence has also allowed for greater flexibility in shifting to remote or hybrid modes of work and brings the testing service one step closer to the goal of automation.
Literature Review and Background:

The need to patch a set of signals to a set of destinations has been a problem that shows up several times throughout history. This includes landline telephone calls, closed-circuit security video, and more recently internet and other telecommunications. This has come to be known as switching. Switching fabric theory is a large field in and of itself. The switching fabric used in this project is known as a crossbar matrix switch, crosspoint switch, or matrix switch. Additionally, it has the qualifiers of being non-blocking (multiple patch routes can be made on the same matrix) and bi-directional (signals can propagate in both directions through a route. Hardware used for telephone exchange (the process of connecting two or more telephone lines) is highly specialized to be repurposed for this project. Similarly, ethernet switching largely uses software to perform the desired routes.

The need for such crossbar switches was common and as such there are many ICs that accomplish this task. As an example, Analog Devices' AD75019 Crosspoint Switch Array “contains 256 analog switches in a 16 × 16 array. Any of the X or Y pins may serve as an input or output. Any or all of the X terminals may be programmed to connect to any or all of the Y terminals” per the datasheet [5]. The switches are controlled by a 256-bit shift register and the ICs can be chained together to create larger arrays. This is even more functionality than is provided in the product of this project. The issue is that it is built for Radio Frequency signals with a tool having a bandwidth of 20MHz. Very little current can be passed at the DC frequency making it a poor choice for the DC signals used in the testing service. Nonetheless, an eval board for a similar IC costs $459.70 supporting signals up to +/- 5V and 60 MHz. DC would be limited to 20 mA and voltage would decrease significantly at that current level [8].

The device that comes closest to the functionality achieved in this project is available for purchase through Pickering Interfaces Ltd, a company that specializes in switching devices.
Their PXI 28x16 Matrix Module, 1-pole 2A 60W allows for DC signals up to 60 watts to be patched through the crosspoint switch. The mechanical relays in the design have a bandwidth of 10MHz and DC resistance of 0.35 ohms [6]. The module has a price tag of $5260 and requires a chassis to operate. Most chassis from Pickering start at $1195, and the software appears to be free [7]. In analogy to this project, this would support 28 input/output signals with 16 busses which can be used as an additional 16 output only signals. The price tag on this device is not feasible for the benefits it would provide the lab. That said, this device would suffer from less crosstalk issues, current driving issues, and take less development time from the UNH-IOL. Ultimately, it was more beneficial to develop a product within UNH-IOL.

Project Goals:

The first goal of this project was to define what a solution might look like and its required functionality. It would be important to consider the use case, corner cases, and future expandability while defining the requirements that the solution needed to fulfill. This would include an in-depth analysis of the signals that need patching, their use, electrical characteristics (voltage, voltage swing, current), and an analysis of all tests to determine all required combinations and permutations of signal patch states.

The second goal was to create a high-level design for the solution. This would include selecting parts that satisfied the requirements determined in the first step and creating conceptual diagrams. This goal planned to involve teams at the IOL to help turn these high-level designs into the schematics and files that would be printed through JLC PCB.

After creating, testing, and refining prototypes, the primary goal was to settle on a complete and final solution that could be controlled from a computer and reliably and accurately patch signals. This final solution would then be deployed, utilized, and reviewed as part of the normal scheduled testing service.
The final phase and goal were to enable the solution to be controllable via a RESTful API. This API would allow users to more readily control the device, and provide a handle for a future automation engine to be able to modify the signal patching between tests. This goal was eventually dropped from the project as described later in this document.

Methods:

Design Constraints

The design process began by defining the requirements and constraints of the tool. This included a survey of all the tests shown in Appendix 3 - 100BASE-T1 Signal Matrices. The survey yielded a collection of matrices that describe the needed signal connections for each test. From this process of listing the required per-test signal connections, it was determined that all tests can be performed with four or fewer driving signals (considered inputs to the board). Hence the first design requirement was a tool that could patch at least four lines (called buses in this project). Further, this survey showed that a total of 18 signals need to be available for patching, necessitating that the tool has at least 18 port connections. It was also noted at this stage, that some signals, if not tied to a low logic level (ground) may cause undesired behavior in devices under test. The tool must be able to hold a selection of signals to ground to avoid this.

It was also at this stage that other electrical characteristics like signal voltage, current, acceptable propagation delays, noise levels, and impedances were set. Signals are often 1.2V or 3.3V logic. On rare occurrences, they may operate on 5v or even exhibit very low voltage swings of 0.1V between LOW and HIGH levels. The tool needs to be able to work with all these voltages and, in some scenarios, adjust the logic levels so two signals of different voltage levels can be tied together. In the testing service, the finest time division used in testing is 30 ns.
Propagation delays and rise time delays cannot surpass 30 ns. Any Gaussian or mains 60Hz noise must be kept below +/- 0.1V to avoid errors in interpreting the signals. A bound on the maximum electrical current was hard to determine given the vast number of devices that will need to work with this board. A value of 100 mA was chosen based on typical digital logic current driving capabilities. With the signals being single-ended and digital, impedance matching was not identified as a need for this device. The overall resistance of the path would simply need to pass the full current of signals without substantial voltage drop. Signals that would need to drive more than a few milliamps of current would operate at 3.3V. So voltage drop at full current would need to be less than 0.7V.

**Conceptual Design #1**

Thus began the brainstorming Phase. The first design concept was based roughly on a crossbar switching fabric, with solid-state relays connecting desired signals to one of 6 buses. The functional diagram of this modified crossbar matrix is shown in Figure 1 and a conceptual diagram is shown in Figure 2.

*Figure 1 - Functional crossbar diagram showing 3 signals patched together on Bus 1*
This design did not have enough ports for the 18 signals let alone surplus ports for future growth. All the relays were to be controlled directly from I/O pins on the microcontroller. The significant quantity of IO pins to achieve this necessitated two large microcontrollers operating in a Master-Slave configuration. Further, this design could not tie signals to ground. Finally, the solid-state relays posed concerns about representing signals with fidelity. Solid-state relays are limited in current and voltage and do add to the rise time and propagation delay. At this point,
the extent of these effects was not measured or known. Additionally, for signals that have an extremely small voltage swing of ~0.1V, solid-state relays may not reflect the signal sufficiently.

Component Selection and Testing

With a starting concept, however, it was possible to move forward with selecting components like the solid-state relays and testing them for the concerns mentioned above. Two samples of solid-state relays were chosen for their bi-directionality, voltage range, and propagation delays according to the datasheets. These relays were tested by passing a rising and falling DC Digital Signal through the closed relay. Two oscilloscope channels were used to plot the signal before entering the relay and after exiting the relay. Tools on the oscilloscope then enable the difference in 50% crossings to be measured as shown in an example in Figure 3.

![Figure 3 - Plot of DC Digital rising edge before the relay (Blue) and after the relay (Red). 9.07 ns delay in 50% signal crossing which is less than the 30 ns time division used in testing.](image-url)
In anticipation of the second design concept other components like mechanical relays, shift registers, and a logic shifter IC was also purchased and tested at this stage.

Conceptual Design #2

The second design concept first added shift registers as the drivers for all relays. This enabled a single serial communication line on the microcontroller to be used to control the vast number of relays in parallel. To circumvent some of the noise concerns posed by the solid-state relays, 2 of the 6 buses were switched over to mechanical relays. Mechanical relays are more expensive and have a limited number of cycles given the moving parts in their design. For this reason, the other 4 buses remained solid-state. The user is able to use the mechanical buses for more sensitive signals, or when the solid-state relays fall under suspicion of inaccurately representing signals. An updated functional diagram is shown in figure 4.

Figure 4 - Expanded functional crossbar diagram - Bus 1 is shown connecting signals 1, 8, and 17.
The number of channels in the design also increased from 16 to 24. This provided room for all 18 signals, 4 channels for a logic booster, and 2 extra channels for future growth. The two mechanical relay buses also featured a GND connection which is treated as a 25th signal that can be connected to only these two buses for the purpose of grounding signals that can’t be left floating. An updated conceptual diagram is shown in figure 5.

**Figure 5 - Second conceptual design diagram**

It was at this stage that the signal booster circuit of the design was formalized and added. This element of the design is a secondary functionality that allows logic level shifting among the 1.2V, 3.3V, and 5V standards that are often seen in the pool of signal
implementations. The datasheet for the IC that was tested and chosen for this project describes it as a tri-state Dual Supply Translating Transceiver. It can level shift two independent signals (two inputs to the booster with two corresponding outputs) and will be treated as 4 new signals for the product to handle. 6 additional mechanical relays are used to control the two reference supply voltages to the VCCA and VCCB pins. When all 6 relays are open the device enters a high impedance state. Each reference supply pin is connected to three relays, which when closed tie the pin to 1.2V, 3.3V, or 5V respectively, thus only 2 of these 6 relays are closed at any given time. The circuit diagram for this logic shifter is shown in figure 6.

Figure 6 - Signal logic shifter/booster circuit
Component List

The following table describes the components that were chosen and purchased for this project. Given that mechanical relays use coils to actuate, the shift registers used for them are designed as high-current current-sinks rather than digital voltage drivers. The list of components is shown in table 1.

<table>
<thead>
<tr>
<th>Part</th>
<th>Component Name</th>
<th>Component URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift Register</td>
<td>SN74HC595PW</td>
<td><img src="https://www.digikey.com/en/products/detail/texas-instruments/SN74HC595PW/1571273" alt="SN74HC595PW" /></td>
</tr>
<tr>
<td>Solid-State Relay</td>
<td>MC74VHC4066DR2G</td>
<td><img src="https://www.digikey.com/en/products/detail/onsemi/MC74VHC4066DR2G/1480972" alt="MC74VHC4066DR2G" /></td>
</tr>
<tr>
<td>Mechanical Relay</td>
<td>9007-05-00</td>
<td><img src="https://www.digikey.com/en/products/detail/coto-technology/9007-05-00/301696" alt="9007-05-00" /></td>
</tr>
<tr>
<td>Logic Shifter</td>
<td>74AXP2T45DCH</td>
<td><img src="https://www.digikey.com/en/products/detail/nexperia-usa-inc/74AXP2T45DCH/13575029?sid=N4lgTCBcDalowBYCCANACmAgggrAAGUB5TOANhxAF0BfIA" alt="74AXP2T45DCH" /></td>
</tr>
<tr>
<td>BNC Connectors</td>
<td>B6251C1-NT3G-50</td>
<td><img src="https://www.newark.com/amphenol/b6251c1-nt3g-50/connector-bnc-jack-str-50-ohm/dp/23M5399" alt="B6251C1-NT3G-50" /></td>
</tr>
<tr>
<td>Microcontroller</td>
<td>Arduino Nano</td>
<td><img src="https://www.amazon.com/ELEGOO-Arduino-ATmega328P-Without-Compatible/dp/B0713X9K23/ref=sr_1_1_sspa?dchild=1&amp;keywords=Arduino+nano&amp;qid=1627584337&amp;sr=1-1-spons&amp;psc=1&amp;spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEzTTRWQUUpWMEORkNEJmVuY3J5cHRZEikPUEwNTc5MTE0M1Q2Tzc4NU9YT0hETyZibmNyeXB0ZWRBZEIkPUEwNjY5Mjg5MTdYSSuo3UVNSUdZaVS3aWRnZXROYW1IPXNwX2F0ZiZhY3Rpb249Y2xpY2tSZWRpcmVjdCZkb05vdExvZ0NsaWRnPXRydWU=" alt="Arduino Nano" /></td>
</tr>
<tr>
<td>1.2V Voltage Regulator</td>
<td>MCP1826S-1202E/A</td>
<td><img src="https://www.digikey.com/en/products/detail/micro" alt="MCP1826S-1202E/A" /></td>
</tr>
</tbody>
</table>
Table 1 - Selected Components

<table>
<thead>
<tr>
<th></th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3V Voltage Regulator</td>
<td>LD1117AV33</td>
</tr>
<tr>
<td>DC Barrel Jack</td>
<td>PJ-037A</td>
</tr>
</tbody>
</table>

At this stage, the datasheets of the selected components were referenced to create the electrical schematic matching the design diagrams shown in figures 4, and 5. The final schematic is shown below in figure 7.
Figure 7 - Electrical schematic
Figure 7a - Solid-state relay bus connections
In brief, each bus is comprised of 24 (25 for mechanical buses) relays. One leg of all 24 or 25 relays is electrically connected. The remaining leg of each relay is tied to one of the 24 BNC connectors and ground for the 25th relay. Closing any relay will tie the corresponding signal to the bus, and when multiple relays are closed on the same bus, the signals are electrically connected. Each relay also has a control line that is driven by a shift register.

**PCB development and 1st Prototype**

At this stage, the schematic was provided to a colleague, Govind Krishram Kothimbakam Kothimbakam, who worked to turn it into a PCB that could be ordered through a PCB
manufacturer JLC PCB. During this process, the PCB was reviewed for discrepancies with the provided schematic. The first PCB created by Kothimbakam is shown below in figure 8.

![Figure 8 - V0.1 of the PCB](image)

The first revision of the board was received, populated with a small subset of components. The populated board is shown below in figure 9.
Figure 9 - V0.1 of the PCB populated with minimal components for testing

The following issues were found with the first revision:
Figure 10 - Solid-state relay 3 had pins tied to ground rather than the bus

- The footprint for the logic shifter was incorrect and the IC could not be soldered to the board.

- While soldering of SMDs, the solder would often flow under the package and create shorts with vias placed under the package. Future revisions would move all vias out from under SMD footprints.

Figure 11 - Bus 5 had an improper connection to a shift register output.
Aside from these issues, the Arduino was able to drive the shift registers properly and in turn, close the populated relays independently. When closed, the relays successfully patched signals in and out of the bus. Further verification of the traces was performed with a connectivity meter on an unpopulated board and revealed no issues.

2nd Prototype

Kothimbakam determined that a 6-layer board would be preferable over the 4-layer board used in the first prototype in order to reduce via count and trace length. With the extra layers, however, there were more errors in the trace routing. Examples are shown in below table 2. Errors are shown in red and some correct traces are overlaid in black. Vias often overlapped creating undesired shorts. Other oddities posed no issue other than odd geometry.
Table 2 - PCB V0.2 issues found in visual inspection

To verify this prototype all traces and nets were inspected visually and then with a connectivity meter. Afterward, the microcontroller and shift registers were populated. The microcontroller then turned on a single relay control line. The active relay control line, and the control lines immediately before and after it was probed at the relay footprint with a voltmeter to ensure that just the desired relay would be closed. This was repeated for all 152 relay control lines. No issues were found outside of those found in the first step.

The solid-state relays were then populated. An LED was placed on the first signal and the relay for the first signal closed. Each consecutive relay was closed and the corresponding, preceding, and following signals were tied, in turn, to 3.3V. When successful the LED would illuminate only when the corresponding signal was tied to 3.3V and not when the preceding and following signals were tied to 3.3V. By running the procedure, it was found that signal 20 was held high when closed regardless of the bus used. Even on an unpopulated board, a connectivity meter confirmed that signal 20 was connected to VCC. Further Inspection showed that the net for BNC_20 (signal 20) passed through a VCC via an inner layer as shown in figure 12.
3rd Prototype

Experience with the second revision highlighted the importance of verifying the traces before the board was printed. The third revision is shown below in figure 13.
Figure 13 - Version 3 of the PCB.

The same verification methods used on the 2nd prototype were used here as well. Only two errors were found. The DC power jack footprint was placed so that the plug faced into the board prohibiting the plug from being inserted. There was also one trace that was completely routed to the via as shown in figure 14. This problem was resolved by soldering a jumper wire in place of the incomplete via.
User Interface and Serial Communication Control

The microcontroller is the first layer of abstraction sitting between the direct control of the 152 relays and the human user. It must be able to take commands from the user (or eventual API server) and use that to control the relays. To start, the desired command structure was created as follows:

- [Bus#].[Signal1Dec(INPUT)].[Signal2Dec(OUTPUT)]. ... [.SignalNDec(OUTPUT)]
  - overwrites the state of a bus
  EX:  2.01.14.15  6.05.02.03  1.10.15

3.14.15.19.02.06.15.03.15.19 (will ignore duplicate signals 15 and 19)
5.14.19

  Bus#: Values '1' through '6'
  SignalNDec: Decimal Values 01 through 25

- B.[VCC IN in volts: 1, 3, 5, or Z].[VCC OUT in volts: 1, 3, 5, or Z]
  - overwrites the state of the Booster
  EX:  B.1.5  B.5.3  B.Z.Z
Z places the booster into a high impedance state so the BNCs can be used for other signals

reset - resets everything to default settings
cur state - prints the current state of everything
help - prints the usage
Signals can't be used as both an input and output.
Signals can't be used as an input (or output) on more than one bus.

From there, pseudocode was written and iteratively made progressively more complex.

```plaintext
init(); // init pins, vars, serial communication, set default outputs

onReceiveCommand(); // Receive the command from serial, pass command to handleCommand()

handleCommand(command); // Parse command, Call low level functions to execute command, call printState()

printState(); // called by handleCommand(), prints the current state of the switcher to the serial interface

Level 1 Pseudocode

init() {  
    // init pins  
    // init vars  
    // start serial communication  
    // set default outputs  
}

loop() {  
    // listen for command  
}

onReceiveCommand() {  // function not used  
    command = receivedString; // Receive the command from serial  
    handleCommand(command);  
}
handleCommand(command) {
    if (1 == parse(command)) // Parse command - set global state variables
        shiftOutStates(); // shift out the state vectors on the shift
        printState();
}

parse(command) { // Parse command - set global state variables
    if is bus command
        // remove signals that appear more than once in the single command
        // save applicable Bus State variable to a temp variable
        // save string to applicable Bus State variable (variable used to store
        current state of a given bus)
        // set list of inputs
        // set list of outputs
        // throw error if a signal exists in both input and output list
        // throw error if a signal appears in the output list twice
        // throw error if a signal appears in both the bus and booster output list
            // throwing an error causes parse to return 0 and prints error to
            serial
        // restore applicable Bus State variable to original value
        // regenerate lists of outputs and inputs
        // if no error, create SS vector, and mechanical vector, return 1
            // switch on bus, and the SS and Mech vector with mask that zeros out
            current bus
        // for each signal, switch on bus, switch on hex, or a bit position with
        current SS and Mech vector
            else if is booster command
                // throw error if output signal of the booster is an output
            elsewhere, stay in Hi Z state and return 0
                // else add to booster input output list
            // create mechanical vector, return 1 TBD, will require using relays to
            connect proper voltage to VCCA and VCCB. tri state covers the connection to
            the signals
    }

    printState() {
Level 2 Pseudocode

With pseudocode for reference, the full code was written and tested throughout development to ensure they were implemented properly. The full final code can be found in Appendix 2 - Microcontroller Code. The microcontroller receives command over a UART connection with a 115200 baud rate. For the user interface the term “Booster” refers to the logic shifter circuit. For simplicity, the serial monitor in Arduino is the easiest to use. Below are screenshots showing the use of the serial user interface.

![Figure 15 - UI boot up splash screen with usage information](image-url)
Figure 16 - User sends “1.01.02” command by pressing enter. This connects BNC 1 to BNC 2 using the first bus, a solid-state bus, and designates BNC 1 as an input to the board. The state of the tool is printed.
Figure 17 - User sends “02.08.23.05” which connects BNC 23 and BNC 5 to BNC 8 with BNC 8 being designated as an input to the board.

Figure 18 - User sends “6.18.19.20.21.22” command which connects BNCS 18 through 22 together on Mechanical relay bus 6.

Figure 19 - User tries to use signal 18 as input to a second bus. Error thrown, state not updated.
Figure 20 - User tries to use signal 18 as an output on a second bus. Error thrown, state not updated.

Figure 21 - User tries to use signal 2 as an output on a second bus. Error thrown, state not updated.
You said: RESET

Command successfully parsed!
Applying State .....................................................
State Applied!

The current state is:
Solid State Bus 1.
Solid State Bus 2.
Solid State Bus 3.
Mechanical Bus 5.
Mechanical Bus 6.

Booster: B.
Booster Enabled: 0

Execution Time: 69 ms.

Figure 22 - User resets the state

You said: B.1.5

Command successfully parsed!
Applying State .....................................................
State Applied!

The current state is:
Solid State Bus 1.
Solid State Bus 2.
Solid State Bus 3.
Mechanical Bus 5.
Mechanical Bus 6.

Booster: B.1.5
Booster Enabled: 1

Execution Time: 68 ms.

Figure 23 - User sets booster to boost signals from 1.2V to 5V
Figure 24 - User connects BNC 1 to the first input to the booster, BNC 19, and connects the first output of the booster, BNC 21, to BNC 2. This configuration takes 1.2V signals in on BNC 1, logic shifts them to 5V logic and outputs them on BNC 2 using the mechanical relay buses (solid-state could be used if desired).

One of the original goals set for this project was to create a RESTful API Server to take HTML requests and issue the UART commands to the microcontroller. Achieving this goal would have put the tool one step closer to incorporation with an automation engine. It would not, however, have brought the tool any closer to being a minimum viable product. Ultimately, the goal was not pursued to allow time for the completion of the tool.

Results:

The third revision of the board was fully populated and tested with a representative collection of patching configurations. All cases were tested with both a rising and falling edge with the source before entering the board and the outputs from the board being plotted on an
oscilloscope. The test cases included a signal commonly used in the testing service on a solid-state relay, the same signal being output on more than one BNC using both solid-state and mechanical relay buses, short pulses of the same signal (300ns pulse and 30ns pulse), signals patched through all buses to a representative selection of BNCs, signals passing through the logic shifter with all 9 configurations of the logic shifter.

The design meets all of the pre-defined design constraints including, patching more than 18 signals, shifting the voltage of signals, tying signals to ground, and introducing propagation delays and rise-time delays of less than 30 ns. Under high current draws, a 3.3V signal on the solid-state relay buses experiences a nearly 2V voltage drop. This issue does not exist for mechanical relays which have a combined resistance of 5 ohms (2.5 ohms per relay on the path) allowing them to be used for high current draw applications as well. To demonstrate the product an Arduino created 5 different digital pulse patterns that would be differentiated by a human eye. These were connected to 5 of the BNCs. Oscilloscope channels were connected to 4 of the unused BNCs, and all remaining BNCs have LEDs attached. The 5 patterns were then patched in several combinations to the oscilloscope and LEDs to show the function of the tool. The final revision is now ready for deployment in the testing service. The product has been deployed for only a few weeks preventing significant data collection on device performance in the use case scenario. The final board is shown in figure 25 and the estimated cost per board is shown in table 3.
**Figure 25 - Finished product ready for deployment in testing service**

<table>
<thead>
<tr>
<th>Item Name</th>
<th>Price</th>
<th>Quantity</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPIC6C595DRG4 - High Current Shift Register</td>
<td>$1.04</td>
<td>7</td>
<td>$7.25</td>
</tr>
<tr>
<td>SN74HC595PW - Shift Register</td>
<td>$1.07</td>
<td>12</td>
<td>$12.90</td>
</tr>
<tr>
<td>MC74VHC4066DR2G - Solid-State Relay</td>
<td>$0.48</td>
<td>24</td>
<td>$11.46</td>
</tr>
<tr>
<td>9007-05-00 - Mechanical Relay</td>
<td>$1.12</td>
<td>56</td>
<td>$62.45</td>
</tr>
<tr>
<td>74AXP2T45DCH - Logic Shifter</td>
<td>$0.57</td>
<td>1</td>
<td>$0.57</td>
</tr>
<tr>
<td>Arduino</td>
<td>$5.99</td>
<td>1</td>
<td>$5.99</td>
</tr>
<tr>
<td>BNC Connectors</td>
<td>$1.59</td>
<td>24</td>
<td>$38.16</td>
</tr>
<tr>
<td>Item Description</td>
<td>Price</td>
<td>Quantity</td>
<td>Total</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>--------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>MCP1826S-1202E/AB (1.2v Voltage regulator)</td>
<td>$1.20</td>
<td>1</td>
<td>$1.20</td>
</tr>
<tr>
<td>LD1117AV33 (3.3v Voltage regulator)</td>
<td>$0.66</td>
<td>1</td>
<td>$0.66</td>
</tr>
<tr>
<td>PJ-037A (DC Barrel Jack)</td>
<td>$0.60</td>
<td>1</td>
<td>$0.60</td>
</tr>
<tr>
<td>V0.3 PCB</td>
<td>$45.15</td>
<td>1</td>
<td>$45.15</td>
</tr>
<tr>
<td>PPTC151LFBN-RC</td>
<td>$0.83</td>
<td>2</td>
<td>$1.65</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>$188.04</strong></td>
</tr>
</tbody>
</table>

*Table 3 - Estimated cost per board*
Future Work:

Given the device has not been in service for longer than a week, feedback from users is limited. Future work on this device would include formal measurements of bandwidth, impedance, current driving capabilities of the logic shifter, and reliability repeatability testing. This data would be used to create a datasheet of the tool describing its electrical characteristics. Future design work would include, improving the current driving capabilities of the solid-state relay buses, and adding alternative signal interfaces like pin headers alongside the BNCs. Errors in the traces like the trace that does not reach via and the backward DC power jack footprint on backward would also need to be fixed. The silkscreen could be improved to label the relays and shift registers by their bus and BNC signal rather than pure enumeration. Currently, the single bus reset commands like “1.r” in the serial user interface does not appear to properly reset the bus leaving relays closed that should not be. Finally, the serial interface should be wrapped in a RESTful API Server to enable future incorporation in the testing service and automation efforts.
Timeline:

Figure 26 shows a breakdown of the project timeline including milestones of the project and major due dates for the Senior Project. The milestones resemble the major points discussed above in the Methods section. Appendix 1 - Original Senior Project Gantt Chart shows the original timeline for comparison. As discussed previously, four of the original tasks were either not necessary or cut for time.

![Figure 26 - Senior Project Gantt Chart](image-url)
Budget:

Table 4 shows the budget and costs of the project. It should be noted that the UNH-IOL will be covering all costs of this project, and this budget is presented for awareness purposes only.

<table>
<thead>
<tr>
<th>Item Name</th>
<th>Price</th>
<th>Quantity</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPIC6C595DRG4 - High Current Shift Register</td>
<td>$1.04</td>
<td>30</td>
<td>$31.08</td>
</tr>
<tr>
<td>SN74HC595PW - Shift Register</td>
<td>$1.07</td>
<td>50</td>
<td>$53.74</td>
</tr>
<tr>
<td>MC74VHC4066DR2G - Solid-State Relay</td>
<td>$0.48</td>
<td>100</td>
<td>$47.74</td>
</tr>
<tr>
<td>9007-05-00 - Mechanical Relay</td>
<td>$1.12</td>
<td>175</td>
<td>$195.14</td>
</tr>
<tr>
<td>74AXP2T45DCH - Logic Shifter</td>
<td>$0.57</td>
<td>10</td>
<td>$5.70</td>
</tr>
<tr>
<td>Arduino</td>
<td>$17.98</td>
<td>1</td>
<td>$17.98</td>
</tr>
<tr>
<td>BNC Connectors</td>
<td>$1.59</td>
<td>100</td>
<td>$159.00</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>$510.38</strong></td>
</tr>
<tr>
<td>V0.1 PCB</td>
<td></td>
<td>5</td>
<td><strong>$72.63</strong></td>
</tr>
<tr>
<td>PPTC151LFBN-RC</td>
<td>$0.83</td>
<td>16</td>
<td><strong>$13.23</strong></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>$13.23</strong></td>
</tr>
<tr>
<td>V0.2 PCB</td>
<td></td>
<td>5</td>
<td><strong>$187.33</strong></td>
</tr>
<tr>
<td>MCP1826S-1202E/AB (1.2v Voltage regulator)</td>
<td>$1.20</td>
<td>10</td>
<td><strong>$12.00</strong></td>
</tr>
<tr>
<td>LD1117AV33 (3.3v Voltage regulator)</td>
<td>$0.66</td>
<td>10</td>
<td><strong>$6.60</strong></td>
</tr>
<tr>
<td>PJ-037A (DC Barrel Jack)</td>
<td>$0.60</td>
<td>10</td>
<td><strong>$6.00</strong></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>$24.56</strong></td>
</tr>
<tr>
<td>V0.3 PCB</td>
<td></td>
<td>5</td>
<td><strong>$225.76</strong></td>
</tr>
</tbody>
</table>

Items yet to be purchased - to improve the deployment process
<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
<th>Quantity</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNC Cables</td>
<td>$7.50</td>
<td>35</td>
<td>$262.50</td>
</tr>
<tr>
<td>BNC To Grab Hook</td>
<td>$13.48</td>
<td>35</td>
<td>$471.81</td>
</tr>
</tbody>
</table>

Table 4 - Budget
Discussion and Conclusions:

The tool was funded in its entirety by the UNH-IOL. The biggest question, however, is whether the final cost of the tool is worth the time saved in man-hours and if another readily available tool could have provided a solution with a lower cost. As is true with all projects, economics of scale will improve the reward to cost ratio when the tool is used in a growing number of test setups. While the tool allows the setup to be modified over the course of testing it does add an additional setup requirement. There is time that needs to be taken to connect all the signals to the device at the beginning of each testing reservation. This will be less than the time taken to manually route and re-route signals throughout testing and should provide benefits as the team at UNH-IOL learns how to use the tool effectively. Currently, UNH is not prohibited from accessing the facilities in person. As such technicians are physically present to adjust the testing setup. That said, the COVID-19 pandemic has changed the way we treat minor illnesses like coughs and colds. More technicians need to be able to work effectively from home and tolerate interruptions in lab access. This is where this tool will shine the most, allowing for testing to continue when remote work is either required or convenient. The greatest benefit will be derived from automation which is still unrealized. Automation would allow a single technician to test 4 devices in a day compared to the 1 device every 3 to 8 weeks, and would not be possible without this tool.

A single comparable device could be purchased from Pickering Interfaces Ltd for around $7000. This tool was developed with half of the $2000 parts budget and with nearly 200 paid employee hours valued at approximately $4000. Further copies of the product could be produced for around $200 each. This means that the total cost of the first board, $6000, is still less than the cost of the commercially available product with each subsequent device being an order of magnitude less than the commercially available product. Finally, the project provides
more than simply monetary value. The UNH-IOL company values include Innovation Culture\textsuperscript{1} and Employee Empowerment and Growth\textsuperscript{2}. The SPE Signal Switcher is more than just a device that enables test setups to be manipulated remotely or autonomously; it resembles the empowerment of employees to identify problems in their responsibilities and enact meaningful change while also accelerating their academic growth.

The most impactful engineering projects bring about meaningful change in the lives of people as they go about their daily routines, tasks, and jobs. This project solved a pressing issue in conformance testing at the UNH-IOL. Testing once required significant manual manipulation, remained unprotected from disruptions to lab access and was unable to be automated. With the completion of this project, this testing no longer needs manual human intervention. Testing can be performed without any need for a human to be physically present. It provides defense against the current COVID-19 pandemic, remains robust to future disruptions and events, and enables the adoption of remote or hybrid models of work in the future. It serves as the stepping stone toward the reduced costs, improved quantity, quality, consistency, and improved employee experience that comes with the eventual automation of the tests.

The best projects for learning embody literature review, novel problem solving, and hands-on learning opportunities. This project required significant learning and expansion of the student’s knowledge set. It consisted of solving a yet unsolved issue in the student’s daily work life. The process presented many challenges to overcome as designs were conceived, parts selected, prototypes created and tested, and obstacles encountered. The project provided opportunities to learn about many aspects of design, including - identifying constraints, and testing designs - as well as skills like creating functional PCBs, finding previous solutions to

\textsuperscript{1} Sharing our knowledge and learning new concepts helps our employees and interns become today’s pioneers and tomorrow’s entrepreneurs. We take an active role in cooperating with our partners and peers to make improvements to the connected world.

\textsuperscript{2} Our employees are continuously learning and involved in all aspects of the organization. Our belief is that every employee has the ability to effect change within our organization, the global industries we serve, and their career path. Through open communication and a continuous learning environment everyone has the keys they need to open new doors for themselves and the organization while pursuing the projects that excite them.
similar problems, and programming microcontrollers. Alltogether, the aspects of this project were enriching and impactful.

Acknowledgments

Thank you to:

- UNH-IOL, the Research working group, and the IOL Ignite fund for funding this project
- Robert Noseworthy, and Jason Sisk for their technical advising in this project
- Professor Qiaoyan Yu for her terrific academic advising on this project
- Govind Krishram Kothimbakam Kothimbakam for the tremendous work on the PCB development in this project
- Everyone working in the Single Pair Ethernet Working group for their feedback on the project
References:


[8] Analog Devices “EVAL-AD8113 AD8113 Evaluation Board”, Product Page,
Appendices:

Appendix 1 - Original Senior Project Gantt Chart

Appendix 2 - Microcontroller Code

```c
/*
Project:
API Controlled Single Pair Ethernet Signal Switcher
Purpose:
Provide programmatic control of the routing of analog signals
Scope:
PCB Board design, part selection, physical board testing, signal integrity validation,
Applications:
SPE 100Base-T1 PCS, PHYC, TC10, and maybe other speeds

Project Lead:
Hunter Wells
Reports to:
Bob Noseworthy
*/
```
Project Description:
SPE PCS, PHYC, and TC10 conformance testing requires frequent switching/patching of signals to/from/amongst the DUT, BitPhyer, Oscilloscope, and other test equipment. The time taken to leave one's desk and check on or move cables is significant and tedious. A device that can remotely patch panels will eliminate the time taken to manually re-patch signals, allow for easier remote testing capabilities, and provide one additional step towards complete automation of testing.
This will be for 100Base-T1, but should be able to apply to any technology with similar signal patching needs like other SPE technologies. May also be sold by the IOL for customers to match our test setup in house. The device will likely use an Arduino based controller with a GPIO signal to control the switching.
May also include signal logic booster with configurable output level.

```
01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19
01,02
03,05
04,0a,0f
10,11
19,18
09,13
B.1,3

add command reset, and cur state
*/
```

```
#define COMMAND_LENGTH 78
// 3 chars per 24 signals plus two chars for the bus num and one char newline
#define NUM_OF_CHANNELS 25
#define NUM_OF_BUSSES 6
#define BOOSTER_OUT_1_CHANNEL 21
#define BOOSTER_OUT_2_CHANNEL 22
char command[COMMAND_LENGTH];
unsigned long curState[NUM_OF_BUSSES + 1];
// 32 bits of storage per bus (only need 24 to 25. one more vector for the signal booster LSB is channel 1
```
int busInputs[NUM_OF_CHANNELS];
int busOutputs[NUM_OF_CHANNELS];
bool boosterActive = false;

#define OE_BAR_PIN 9
#define SER_PIN 7 //8 for other board
#define RCLK_PIN 8 //7 for other board
#define SRCLK_PIN 6

void handleCommand(char command[COMMAND_LENGTH]); // call parse command, and if successful push the states out to the shift registers
bool parseCommand(char command[COMMAND_LENGTH]); // given the command, check it for errors, then set the global variables accordingly
bool parseChannels(char command[COMMAND_LENGTH], int * channels); // fill out an array representing all the channels in the command from the array of chars that represent the command
void pushCurState(); // takes global state variables, creates a binary vector to turn on the proper relays, calls shiftOutStates
void shiftOutStates(byte stateVector[19]); // takes the binary vector from previous function and shifts out the vector in the proper order Do I need a smaller function to shift out a single bit? or does it make sense to just have the code in a loop in this function?
void printCurrState(); // print out the current state of all circuit elements
void printUsage(); // Print usage instructions to user
void initVars(); // set all variables to disconnected and high-Z

void setup() { // init pins, vars, serial communication, set default outputs
  // init pins
  pinMode(OE_BAR_PIN, OUTPUT);
pinMode(SER_PIN, OUTPUT);
pinMode(RCLK_PIN, OUTPUT);
pinMode(SRCLK_PIN, OUTPUT);

  // init vars
  initVars();

  // start serial port at 9600 bps and wait for port to open:
  Serial.begin(115200);
  while (!Serial) {

}
for (int i = 0; i < 100; i++) {
  Serial.println();
}

Serial.println(F("Welcome to the Single Pair Ethernet Signal Switcher UI.

"));
printUsage();

// set default outputs
digitalWrite(OE_BAR_PIN, HIGH);
digitalWrite(SER_PIN, LOW);
digitalWrite(SRCLK_PIN, LOW);
digitalWrite(RCLK_PIN, LOW);

// output enable = false for SS, Mech, and booster
// shiftOutStates(); // shift out the state vectors on the shift
// output enable = true for SS, mech, and booster

void loop() {
  // Listen for command
  if (Serial.available() > 0) {

    int i = 0;
    for (int j = 0; j < 10; j++) {
      Serial.println();
    }
    long start = millis();
    for (int j = 0; j < COMMAND_LENGTH - 1; j++) {
      command[j] = " ";
    }
    while (Serial.available() > 0) {
      // get incoming byte:
      if (i > COMMAND_LENGTH - 2) {
        Serial.println(F("Command length too long!!\nIGNORING LAST CHARS IN COMMAND!!!"));
        Serial.read();
        command[COMMAND_LENGTH - 2] = 10;
      } else {
command[i] = Serial.read();
Serial.print(F("You said: 
"));
Serial.print(command[i]);
Serial.print(F(" . which is ASCII ");
Serial.print(int(command[i]));
if (command[i] >= 97 && command[i] <= 122) {
  command[i] = command[i] - 32;
}
Serial.println(F(".");
i++;
}
Serial.print(F("You said:
"));
Serial.println(command);
handleCommand(command);
Serial.print(F("\nExecution Time: ");
Serial.print((millis() - start));
Serial.println(F(" ms.");
}
}

void handleCommand(char command[COMMAND_LENGTH]) {
  if (true == parseCommand(command)) {
    Serial.println(F("Command successfully parsed!");
    // output enable = false for SS, Mech, and booster
digitalWrite(OE_BAR_PIN, HIGH);
pushCurState();
digitalWrite(OE_BAR_PIN, LOW);
    // output enable = true for SS, mech, and booster
  }
  printCurrState();
}

/*
parse(command) { // Parse command - set global state variables
  if is bus command
  // remove signals that appear more than once in the single command (And
  signals not available on the bus)
  // save applicable Bus State variable to a temp variable
  // save string to applicable Bus State variable (variable used to store
  current state of a given bus) Skipping
*/
// set list of inputs
// set list of outputs
// throw error if a signal exists in both input and output list
// throw error if a signal appears in the output list twice
// throw error if a signal appears in both the bus and booster output list
// throwing an error causes parse to return 0 and prints error to serial
// restore applicable Bus State variable to original value
// regenerate lists of outputs and inputs
// if no error, create SS vector, and mechanical vector, return 1
// switch on bus, and the SS and Mech vector with mask that zeros out current bus
// for each signal, switch on bus, switch on hex, or a bit position with current SS and Mech vector
else if is booster command
   // throw error if output signal of the booster is an output elsewhere, stay in Hi Z state and return 0
   // else add to booster input output list
   // create mechanical vector, return 1 TBD, will require using relays to connect proper voltage to VCCA and VCCB. tri state covers the connection to the signals
} */

bool parseCommand(char command[COMMAND_LENGTH]) {
      initVars();
      return false;
             command[3] == 'P') {
      printUsage();
      return false;
   } else if (command[1] == '.' && command[0] == '1' || command[0] == '2' || command[0] == '3' || command[0] == '4' || command[0] == '5' ||
              command[0] == '6')) {
      int bus = command[0] - 48;
      int channels[NUM_OF_CHANNELS];
      for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++) {
         channels[i] = 0;
      }
      if (command[2] == 'R') {
unsigned long prevBusState = curState[bus - 1];
int prevInputs[NUM_OF_CHANNELS];
for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++) {
    prevInputs[i] = busInputs[i];
}
int prevOutputs[NUM_OF_CHANNELS];
for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++) {
    prevOutputs[i] = busOutputs[i];
}
Serial.print(F("Resetting bus "));
Serial.print(bus);
Serial.println(F("."));
for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++) {
    //Serial.print(F("prevBusState = "));
    //Serial.println(prevBusState);
    if (prevBusState%2 == 1) {
        int tempChannel = i + 1;
        for (int j = 0; j <= NUM_OF_CHANNELS - 1; j++) {
            if (busInputs[j] == tempChannel) {
                for (int k = j; k <= NUM_OF_CHANNELS - 1; k++) {
                    if (k <= NUM_OF_CHANNELS - 2) {
                        busInputs[k] = busInputs[k + 1];
                        busInputs[k + 1] = 0;
                    } else if (k <= NUM_OF_CHANNELS - 1) {
                        busInputs[k] = 0;
                    }
                }
            }
        }
    }
    for (int j = 0; j <= NUM_OF_CHANNELS - 1; j++) {
        if (busOutputs[j] == tempChannel) {
            for (int k = j; k <= NUM_OF_CHANNELS - 1; k++) {
                if (k <= NUM_OF_CHANNELS - 2) {
                    busOutputs[k] = busOutputs[k + 1];
                    busOutputs[k + 1] = 0;
                } else if (k <= NUM_OF_CHANNELS - 1) {
                    busOutputs[k] = 0;
                }
            }
        }
    }
    prevBusState = prevBusState >> 1;
curState[bus-1] = 0;
return false;

  Serial.println(F("Invalid command."));
  printUsage();
  return false;
}

if (!parseChannels(command, channels)) {
  return false;
}

// Serial.print(F("BUS:"));
// Serial.println(bus);
/* Serial.print(F("CHANNELS:"));
Serial.print(channels[0]);
for (int i = 1; i <= NUM_OF_CHANNELS - 1; i++) {
  Serial.print(F(","));
  Serial.print(channels[i]);
}
Serial.println(); */

  // remove signals that appear more than once in the single command (And signals not available on the bus)
  for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++) {
    /* Serial.println(F("loop1"));
    Serial.print(F("CH:"));
    Serial.print(channels[i]); */
    if (channels[i] > NUM_OF_CHANNELS || ((bus == 1 || bus == 2 || bus == 3 || bus == 4) && channels[i] > NUM_OF_CHANNELS - 1)) {
      Serial.print(F("Selected channel "));
      Serial.print(channels[i]);
      Serial.println(F(" is not a valid channel number."));
      if (i == 0) {
        Serial.println(F("Can't ignore the channel since it is in the Input position.
Invalid command."));
        return false;
      } else {
        Serial.println(F("Ignoring the channel."));
      }
    } else {
      Serial.println(F("Ignored the channel.
"));
    }
    for (int k = i; k <= NUM_OF_CHANNELS - 2; k++) { // remove invalid channel
Serial.print(F("loop1.5:"));
Serial.print(i);
Serial.print(F("."));
Serial.print(k);
Serial.println();
*/
channels[k] = channels[k+1];
channels[k+1] = 0;
}
if (i == NUM_OF_CHANNELS - 1) {
  channels[i] = 0;
}

if (i == NUM_OF_CHANNELS - 1) {
  channels[i] = 0;
} else {
  int temp = channels[i];
  for (int j = i + 1; (j <= NUM_OF_CHANNELS - 1 && temp != 0); j++) {
    //Serial.println(F("loop2"));
    while (temp == channels[j]) { // remove duplicate channel
      for (int k = j; k <= NUM_OF_CHANNELS - 1; k++) {
        /*Serial.print(F("loop3:"));
        Serial.print(i);
        Serial.print(F("."));
        Serial.print(j);
        Serial.print(F("."));
        Serial.print(k);
        Serial.println();*/
        if (k <= NUM_OF_CHANNELS - 2) {
          channels[k] = channels[k+1];
          channels[k+1] = 0;
        } else if (k <= NUM_OF_CHANNELS - 1) {
          channels[k] = 0;
        }
      }
    }
  }
/*Serial.print(F("CHANNELS:"));
Serial.print(channels[0]);
for (int i = 1; i <= NUM_OF_CHANNELS - 1; i++) {
  Serial.print(F(","));
  Serial.print(channels[i]);
}*/
Serial.println();
// save applicable Bus State variable to a temp variable
unsigned long prevBusState = curState[bus-1];
int prevInputs[NUM_OF_CHANNELS];
for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++) {
    prevInputs[i] = busInputs[i];
}
int prevOutputs[NUM_OF_CHANNELS];
for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++) {
    prevOutputs[i] = busOutputs[i];
}
/*Serial.print(F("prevBusState = "));
Serial.println(prevBusState);
Serial.print(F("Bus Inputs: "));
Serial.print(busInputs[0]);
for (int i = 1; i <= NUM_OF_CHANNELS - 1; i++) {
    Serial.print(F","));
    Serial.print(busInputs[i]);
}
Serial.print(F("\nBus Outputs: "));
Serial.print(busOutputs[0]);
for (int i = 1; i <= NUM_OF_CHANNELS - 1; i++) {
    Serial.print(F","));
    Serial.print(busOutputs[i]);
}
Serial.println();
Serial.println(F("Removing current bus channels from Input Output lists.")));*/
for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++) {
    //Serial.print(F("prevBusState = "));
    //Serial.println(prevBusState);
    if (prevBusState%2 == 1) {
        int tempChannel = i + 1;
        for (int j = 0; j <= NUM_OF_CHANNELS - 1; j++) {
            if (busInputs[j] == tempChannel) {
                for (int k = j; k <= NUM_OF_CHANNELS - 1; k++) {
                    if (k <= NUM_OF_CHANNELS - 2) {
                        busInputs[k] = busInputs[k+1];
                        busInputs[k+1] = 0;
                    } else if (k <= NUM_OF_CHANNELS - 1) {
                        busInputs[k] = 0;
                    }
                }
            }
        }
    }
}
for (int j = 0; j <= NUM_OF_CHANNELS - 1; j++) {
    if (busOutputs[j] == tempChannel) {
        for (int k = j; k <= NUM_OF_CHANNELS - 1; k++) {
            if (k <= NUM_OF_CHANNELS - 2) {
                busOutputs[k] = busOutputs[k+1];
                busOutputs[k+1] = 0;
            } else if (k <= NUM_OF_CHANNELS - 1) {
                busOutputs[k] = 0;
            }
        }
    }
}
prevBusState = prevBusState >> 1;

/*Serial.print(F("Bus Inputs: "));  
Serial.print(busInputs[0]);  
for (int i = 1; i <= NUM_OF_CHANNELS - 1; i++) {
    Serial.print(F","));  
    Serial.print(busInputs[i]);
}  
Serial.print(F("\nBus Outputs: "));  
Serial.print(busOutputs[0]);  
for (int i = 1; i <= NUM_OF_CHANNELS - 1; i++) {
    Serial.print(F","));  
    Serial.print(busOutputs[i]);
}  
Serial.println();  */
prevBusState = curState[bus-1];  
// set list of inputs  
int curBusInput = channels[0];  
bool isInPrevInputs = false;  
int i;  
//Serial.print(F("All Inputs: "));  
for (i = 0; (busInputs[i] != 0 && !isInPrevInputs && i <= NUM_OF_CHANNELS - 1); i++) {
    //Serial.print(busInputs[i]);  
    if (busInputs[i] == curBusInput) {
        isInPrevInputs = true;  // Do we want to allow an input to be used
twice? why not just add more outputs to the other bus that had it as an input first? NO this will allow us to remove inputs from the busInputs list

    Serial.println(F("\nError: Input already in use."))); 
    } /*else { 
    Serial.print(F("")); 
    */
    
if (!isInPrevInputs) { 
    busInputs[i] = curBusInput; 
    //Serial.print(busInputs[i]); 
} 
//Serial.println(); 
// set list of outputs 
int curBusOutputs[NUM_OF_CHANNELS]; 
bool duplicateOutput = false; 
for (i = 0; (i <= NUM_OF_CHANNELS - 2 && channels[i + 1] != 0); i++) { 
    curBusOutputs[i] = channels[i + 1]; 
    for (int j = 0; (j <= NUM_OF_CHANNELS - 1 && busOutputs[j] != 0); j++) { 
        if (curBusOutputs[i] == busOutputs[j]) { 
            duplicateOutput = true; 
            Serial.print(F("Error: Output channel ")); 
            Serial.print(curBusOutputs[i]); 
            Serial.println(F(" is already in use."))); 
        } 
    } 
} 
curBusOutputs[i] = 0; 
if (!duplicateOutput) { 
    int j = 0; 
    //Serial.print(F("All Outputs: ")); 
    while (busOutputs[j] != 0 && j <= NUM_OF_CHANNELS - 1) { 
        //Serial.print(busOutputs[j]); 
        //Serial.print(F,”")); 
        j++; 
    } 
    for (i = 0; (i <= NUM_OF_CHANNELS - 1 && curBusOutputs[i] != 0); i++) 
    { 
        busOutputs[j] = curBusOutputs[i]; 
        //Serial.print(busOutputs[j]); 
        /*if (i + 1 <= NUM_OF_CHANNELS - 1 && curBusOutputs[i + 1] != 0) { 
        Serial.print(F("")); 
        }*/
    }
```cpp
j++; // shouldn't need to catch if j is larger than the size of array because in theory if there are no duplicates in busOutputs and curBusOutputs then
}
//Serial.println();
}
bool isInBothInputAndOutput = false;
bool isInBothBusAndBoosterOutputs = false;
for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++) {
  for (int j = 0; j <= NUM_OF_CHANNELS - 1; j++) {
    if (busOutputs[i] == busInputs[j] && busOutputs[i] != 0) {
      isInBothInputAndOutput = true;
      Serial.print(F("Error: Channel "));
      Serial.print(busOutputs[i]);
      Serial.println(F(" cannot be used as an Input and Output at the same time."));
    }
  }
  if (boosterActive && (busOutputs[i] == BOOSTER_OUT_1_CHANNEL ||
    busOutputs[i] == BOOSTER_OUT_2_CHANNEL)) {
    isInBothBusAndBoosterOutputs = true;
    Serial.print(F("Error: Channel "));
    Serial.print(busOutputs[i]);
    Serial.println(F(" is currently being used as a signal booster output and cannot be used as an Output."));
  }
// throw error if a signal is used as a bus input more than once
// throw error if a signal appears in the output list twice
// throw error if a signal exists in both input and output list
// throw error if a signal appears in both the bus and booster output list
if (isInBothBusAndBoosterOutputs || isInBothInputAndOutput ||
  duplicateOutput || isInPrevInputs) {
  // restore lists of outputs and inputs
  for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++) {
    busInputs[i] = prevInputs[i];
  }
  for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++) {
    busOutputs[i] = prevOutputs[i];
  }
  return false;
} else { // if no error, create SS vector, and mechanical vector,
```
return 1

curState[bus - 1] = 0;

// for each signal, switch on bus, switch on hex, or a bit position
with current SS and Mech vector
for (int i = 0; (i <= NUM_OF_CHANNELS - 1 && channels[i] != 0); i++)
{
    /*Serial.print(F("curBusState = "));
    Serial.println(curState[bus - 1]);
    Serial.println(F("ERROR WITH INTEGER MATH. Need to use long
variables."));*/
    long oneBitMask = 1;
    curState[bus - 1] += oneBitMask << (channels[i] - 1);
}
  /*Serial.print(F("curBusState = "));
Serial.println(curState[bus - 1]);*/
}
} else if (command[1] == '.' && (command[0] == 'B')) { // else if is booster command
        Serial.println(F("Invalid command."));
        printUsage();
        Serial.println(F("Both VCCA and VCCB must be 'Z' or both must be
'1','3', or '5'. Neither VCC can be '0','2','4','6',..., '9'"));
        return false;
    }
    /*if (command[2] == command[3]) {
        // do we want to be able to boost a signal from one voltage to the
same voltage? it may help with current driven applications
    }*/
    // throw error if output signal of the booster is an output elsewhere,
    stay in Hi Z state and return 0
    bool isInBothBusAndBoosterOutputs = false;
    for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++)
    {
        if (busOutputs[i] == BOOSTER_OUT_1_CHANNEL || busOutputs[i] ==
BOOSTER_OUT_2_CHANNEL)
            isInBothBusAndBoosterOutputs = true;
        Serial.print(F("Error: Channel 
"));
        Serial.print(busOutputs[i]);
        Serial.println(F(" is currently being used as an Output and cannot
"));
    }
be used as a signal booster output.

return false;

}  
}  
if (!isInBothBusAndBoosterOutputs) { // else add to booster input output
  list  
    boosterActive = true;
    curState[NUM_OF_BUSSES] = 0;
    switch (vccA) {
      case 1: // 5
        curState[NUM_OF_BUSSES] += 4;
        break;
      case 3:
        curState[NUM_OF_BUSSES] += 2;
        break;
      case 5: // 1
        curState[NUM_OF_BUSSES] += 1;
        break;
      default:
        Serial.println(F("GOT SOMETHING ODD!!!")
        curState[NUM_OF_BUSSES] = 0;
        boosterActive = false;
        printUsage();
        Serial.println(F("Neither VCC can be
'0','2','4','6',...,'9'\t"));
        return false;
        break;
    }
  switch (vccB) {
    case 1: // 5
      curState[NUM_OF_BUSSES] += 32;
      break;
    case 3:
      curState[NUM_OF_BUSSES] += 16;
      break;
    case 5: // 1
      curState[NUM_OF_BUSSES] += 8;
      break;
    default:
      Serial.println(F("GOT SOMETHING ODD!!!"));
      curState[NUM_OF_BUSSES] = 0;
      boosterActive = false;
void printUsage();
    Serial.println(F("Neither VCC can be
'0','2','4','6',...,'9'"));
    return false;
    break;
} else {
    boosterActive = false;
    curState[NUM_OF_BUSES] = 0;
}
}
} else {
    Serial.println(F("Invalid command."));
    printUsage();
    return false;
}
return true;

bool parseChannels(char command[COMMAND_LENGTH], int * channels) {
    int i = 2;
    int j = 0;
    while(command[i] != 10) {
        if (j > NUM_OF_CHANNELS - 1) {
            Serial.print(F("You can only specify a maximum of "));
            Serial.print(NUM_OF_CHANNELS);
            Serial.println(F(" channels.")));
        }
        if (command[i] == '.' || command[i+1] == '.') {
            Serial.println(F("Invalid command."));
            printUsage();
        }
        char temp[3];
        temp[0] = command[i];
        temp[1] = command[i+1];
        temp[2] = 0;
        char * pEnd;
        channels[j] = strtol(temp,&pEnd,10);  // was 16 i.e. hex. now decimal
        //Serial.println(F("\n"));
        //Serial.println(temp);
        //Serial.print(F("INTERGER:"));
        //Serial.println(channels[j]);
        j++;
i += 2;
if (command[i] != '.' && command[i] != '0'){
    Serial.println(F("Invalid command."));
    printUsage();
    return false;
}
if (command[i] != '0') {
    i++;
}
return true;

// takes global state variables, creates a binary vector to turn on the proper relays
void pushCurState() {
    //unsigned long curState[NUM_OF_BUSSES + 1]; // 32 bits of storage per bus (only need 24 to 25. one more vector for the signal booster LSB is channel 1
    //bool boosterActive;

    // 24*4 + 25*2 + 6 (1v, 3.3v, 5v for VCCA and VCCB) = 152 bits = 19 bytes (19 shift registers, 12 for the SS relays, and the rest for the mechanical relays on busses 5 and 6 and for the booster circuit)
    // one SPDT relay will be used to connect VCCB to either the selected voltage or to ground and will resemble the state of boosterActive

    byte stateVector[19] = {0};

    for (int i = 0; i < (19*8); i++) {
        int stateVectorByteDELME = i/8;
        int stateVectorBitDELME = i%8;
        int curStateBusDELME;
        int curStateBitDELME;

        if (i >= 0 && i <= NUM_OF_CHANNELS - 1) { // first bus
            curStateBusDELME = 0;
            curStateBitDELME = i - 0;
        } else if (i >= NUM_OF_CHANNELS - 1 && i <= 2*(NUM_OF_CHANNELS - 1) - 1) { // second bus
            curStateBusDELME = 1;
            curStateBitDELME = i - (NUM_OF_CHANNELS - 1);
```c
} else if (i >= 2*(NUM_OF_CHANNELS - 1) && i <= 3*(NUM_OF_CHANNELS - 1) - 1) { // third bus
    curStateBusDELME = 2;
    curStateBitDELME = i - 2*(NUM_OF_CHANNELS - 1);
} else if (i >= 3*(NUM_OF_CHANNELS - 1) && i <= 4*(NUM_OF_CHANNELS - 1) - 1) { // fourth bus
    curStateBusDELME = 3;
    curStateBitDELME = i - 3*(NUM_OF_CHANNELS - 1);
} else if (i >= 4*(NUM_OF_CHANNELS - 1) && i <= 4*(NUM_OF_CHANNELS - 1) + NUM_OF_CHANNELS - 1) { // fifth bus
    curStateBusDELME = 4;
    curStateBitDELME = i - 4*(NUM_OF_CHANNELS - 1);
} else if (i >= 4*(NUM_OF_CHANNELS - 1) + NUM_OF_CHANNELS && i <= 4*(NUM_OF_CHANNELS - 1) + 2*NUM_OF_CHANNELS - 1) { // sixth bus
    curStateBusDELME = 5;
    curStateBitDELME = i - (4*(NUM_OF_CHANNELS - 1) + NUM_OF_CHANNELS);
} else if (i >= 4*(NUM_OF_CHANNELS - 1) + 2*NUM_OF_CHANNELS && i <= 19*8 - 1) { // Booster
    curStateBusDELME = 6;
    curStateBitDELME = i - (4*(NUM_OF_CHANNELS - 1) + 2*NUM_OF_CHANNELS);
} else {
    Serial.println(F("Got Something odd in pushCurState()."));
}

/*Serial.print(F("pushCurState(): i = "));
Serial.print(i);
Serial.print(F"," stateVectorByteDELME = ");
Serial.print(stateVectorByteDELME);
Serial.print(F"," stateVectorBitDELME = ");
Serial.print(stateVectorBitDELME);
Serial.print(F"," curStateBusDELME = ");
Serial.print(curStateBusDELME);
Serial.print(F"," curStateBitDELME = ");
Serial.println(curStateBitDELME);*/

long oneBitMask = 1;
long temp = curState[curStateBusDELME] & (oneBitMask << curStateBitDELME);

/*Serial.print(F("curState[curStateBusDELME] = "));
Serial.print(curState[curStateBusDELME]);
Serial.println($CURSTATEBITDELME$);*/
```
Serial.print(F("\n\n\n"));
for (int i = 0; i <= 19 - 1; i++) {
    Serial.println(stateVector[i]);
}
Serial.println(F("\n\n\n"));
shiftOutStates(stateVector);
}

void shiftOutStates(byte stateVector[19]) {
    Serial.print(F("Applying State"));
    digitalWrite(SER_PIN, LOW);
digitalWrite(SRCLK_PIN, LOW);
digitalWrite(RCLK_PIN, LOW);

    for(int i = (19*8)-1; i >= 0; i--) {
        byte bitToSend = stateVector[i/8] & (0x1 << (i%8));

        if(bitToSend) {
            digitalWrite(SER_PIN, HIGH);
            //Serial.print(F("1"));
        } else {
```c
void printCurrState() {
  Serial.println(F("\n\nThe current state is:"));
  for (int bus = 0; bus <= NUM_OF_BUSSES; bus++) {
    if (bus <= 5) {
      if (bus >= 0 && bus <= 3) {
        Serial.print(F("Solid-State "));
      } else {
        Serial.print(F("Mechanical "));
      }
      Serial.print(F("Bus "));
      Serial.print(bus + 1);
    } else if (bus == 6) {
      Serial.print(F("\nBooster:    B"));
    }
  }
  Serial.print(F(".
"));
  unsigned long tempState = curState[bus];
  for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++) {
    if (tempState%2 == 1) {
      if (bus == 6) {
        Serial.print(6-((i%3)*2 + 1));
      } else {
        if(i < 9) {
```
void printUsage() {
    Serial.print(F("Usage: \n[Bus#].[Signal1Dec(INPUT)].[Signal2Dec(OUTPUT)].
... .[SignalNDec(OUTPUT)]\n\n"));
    Serial.print(F(" - over writes the state of a bus\n"));
    Serial.print(F(" EX:  2.01.14.15  6.05.02.03  1.10.15
3.14.15.19.02.06.15.03.15.19 (will ignore duplicate signals 15 and 19)
5.14.19\n"));
    Serial.print(F(" Bus#: Values '1' through '6'\nSignalNDec:
Decimal Values 01 through 25\n"));
    Serial.print(F("\n[Bus].[VCC IN in volts: 1, 3, 5, or Z].[VCC OUT in volts:
1, 3, 5, or Z]\n" - over writes the state of the Booster\nEX: B.1.5  B.5.3  B.Z.Z\nZ places the booster into a high impedance state so the BNCs can be used for other signals\n"));
    Serial.print(F("\nDO NOT USE SINGLE BUS RESET UNTIL OTHERWISE NOTIFIED
4-11-22!\n"));//Serial.print(F("\n[Bus#].r\n - resets just the specified bus\nEX:  1.r  2.r  6.r\n"));
    Serial.print(F("\nreset - resets everything to default settings\ncur\nstate - prints the current state of everything\nhelp - prints the usage\n"));
    Serial.print(F("\n\nSignals can't be used as both an input and output.\n"));
    Serial.print(F("Signals can't be used as an input (or output) on more than one bus.\n"));
}

void initVars() {
    for (int i = 0; i <= NUM_OF_BUSSES; i++) {
        Serial.print(F("0");
        Serial.print(i + 1);
        if (tempState > 1) {
            Serial.print(F(".");
        }
        tempState = tempState >> 1;
        Serial.println();
    }
    Serial.println(boosterActive);
    Serial.println("Booster Enabled: ");
    Serial.println(boosterActive);
}
curState[i] = 0;
}
for (int i = 0; i <= NUM_OF_CHANNELS - 1; i++) {
    busInputs[i] = 0;
    busOutputs[i] = 0;
}
boosterActive = false;
Appendix 3 - 100BASE-T1 Signal Matrices

<table>
<thead>
<tr>
<th>Test Suite:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs to Board</th>
<th>In Or Out</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope_TRIG_OUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BP_TRIG0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_LOC_RECV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_LINK_STATUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_INH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_SLEEP_FAIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_SLEEP_REQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_SLEEP_ABORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_WAKE_REQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_WAKE_FORWARD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LA_PG_TRIG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNUSED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scope_TX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scope_TX_EN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input Conflict!!!
<table>
<thead>
<tr>
<th>Inputs to Board</th>
<th>In Or Out</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope_TRIG_OUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BP_TRIG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_LOC_RCVR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_LINK_STATUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_INH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_SLEEP_FAIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERBOARD BUS 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERBOARD BUS 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERBOARD BUS 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERBOARD BUS 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERBOARD BUS 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERBOARD BUS 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LA_PG_TX_EN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input Conflict!
### Input Conflict

While this seems to suggest that there shouldn’t be any need for inputs to be outputs and vice versa, if all pins are dual purpose it would allow us to stack two units together with some signals serving as an interface.
<table>
<thead>
<tr>
<th>Inputs to Board</th>
<th>In Or Out</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope TRIG OUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BP TRIG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT LOC RCVR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT LINK STATUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_INH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_SLEEP_FAIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_SLEEP_REQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_SLEEP_ABORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_WAKE_REQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUT_WAKE_FORWARD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LA_PG_TRIG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNUSED</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**WHILE THIS SEEMS TO SUGGEST THAT THERE SHOULDN'T BE ANY NEED FOR INPUTS TO BE OUTPUTS AND VERSA, IF ALL PINS ARE DUAL PURPOSE IT WOULD ALLOW US TO STACK TWO UNITS TOGETHER WITH SOME SIGNALS SERVING AS AN INTERFACE**
WHILE THIS SEEMS TO SUGGEST THAT THERE SHOULDN'T BE ANY NEED FOR INPUTS TO BE OUTPUTS AND VISAVERA, IF ALL PINS ARE DUAL PURPOSE IT WOULD ALLOW US TO STACK TWO UNITS TOGETHER WITH SOME SIGNALS SERVING AS AN INTERFACE.
WHILE THIS SEEMS TO SUGGEST THAT THERE SHOULDN'T BE ANY NEED FOR INPUTS TO BE OUTPUTS AND V/SAVERSA, if all pins are dual purpose it would allow us to stack two units together with some signals serving as an interface.