Design and Integration of an Expandable Radio Frequency Switch Matrix Using Cots Components

A project present to

The Faculty of the Department of Aerospace Engineering

San Jose State University

in partial fulfillment of the requirements for the degree

Master of Science in Aerospace Engineering

By

Zachary Pirkl

May 2014

approved by

Dr. Papadopoulos

Faculty Advisor

The Designated Project Committee Approves the Project Titled

DESIGN AND INTEGRATION OF AN EXPANDABLE RADIO FREQUENCY SWITCH MATRIX

USING COTS COMPONENTS

By

Zachary Pirkl

APPROVED FOR THE DEPARTMENT OF AEROSPACE ENGINEERING

San José State University

May 2014

5/8/14

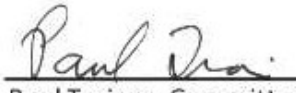
Dr. Periklis Papadopoulos, Committee Chair — Date

Department of Aerospace Engineering

9 May 14

Dr. Nikos J. Mourtos, Committee Member — Date

Department of Aerospace Engineering



4/22/14

Paul Trainer, Committee Member — Date

Space Systems/Loral, LLC

ABSTRACT

DESIGN AND INTEGRATION OF AN EXPANDABLE RADIO FREQUENCY SWITCH MATRIX

USING COTS COMPONENTS

By

Zachary Pirkl

The object of this project was to design and integrate a Radio Frequency Switch Matrix that can be easily expandable to additional pathways for spacecraft ground testing. The standard design used two command highway pathways to deliver one uplink and one downlink pathway for spacecraft payload testing. A microcontroller was used to command the switches to the desired position and to provide real time telemetry updates. The programming of the Switch Matrix interfaced with standard payload testing equipment using a standard IEEE 802.3 Ethernet port and commanded via SCPI protocol. The purpose of this project was to perform a feasibility study of an in-house built Radio Frequency Switch Matrix for SSL to use with the growing complexities of satellite payload testing. The Switch Matrix works for radio frequencies up to 40GHz.

ACKNOWLEDGEMENTS

Our Vision:

We create space-based solutions that improve the human experience

Our Mission:

Architect and build the most reliable, affordable spacecraft and space systems to enable global communications, education, entertainment, health services, disaster recovery, and Earth observation

Our Values:

- Act With Integrity
- Do It right
- Learn, Apply, Improve,
- Make the Company Stronger

- John Celli, President, SSL

I, first and foremost, would like to thank SSL for the opportunity to further my breath of knowledge and partake in this feasibility study that could have a lasting impact on how we test our satellites during their development. I would like to thank Paul Trainer at SSL who has mentored and overseen the project since day one of this project. I would also like to thank Dr. Mourtos and Dr. Papadopoulos, their teaching and support over the years from my undergrad to my Master's Program has laid the foundation of a successful future in the aerospace industry. Their legacy at San José State University is one that should always be remembered and revered. I would like to thank my friends and family who have supported me during my years in the

Master's Program. Their understanding and patience during this difficult time has made working full time and pursing a Master's Degree bearable.

Table of Contents

List of Figures

Nomenclature

A	Amp
C	Capacitance
dB	Decibel
dBm	Decibels per milliwatt
KB	Kilobyte
GHz	Gigahertz
H_{FE}	DC Current Gain
I	Current
I_c	Collector Current
In	Inch
mA	milliamp
Mb	Megabyte
Mbits/s	Megabits per second
MHz	Megahertz
R	Resistance
V	Volt
V_{CEO}	Collector-Emitter Voltage
V_{CE}	Collector-Emitter Saturation Voltage
V_{BE}	Base-Emitter Saturation Voltage
W	Watt

Acronyms

CAD	Computer Aided Drafting
COTS	Commercial Off The Shelf
DPS	Digital Signal Processing
EEPROM	Electrically Erasable Programmable Read Only Memory
GUI	General User Interface
ICSP	In Circuit Serial Programming
IEEE	Institute of Electrical and Electronics Engineers
LAN	Local Area Network
LCD	Liquid Crystal Display
MEMS	Micro Electro Mechanical Systems
MISO	Master In Slave Out
MOSI	Master Out Slave In
MSM	Microwave Switch Matrix
RF	Radio Frequency
SCK	Serial Clock
SCPI	Standard Commands for Programmable Instruments
SDM	Switch Drive Matrix
SM	Switch Matrix
SPI	Serial Peripheral Interface
SS-TDMA	Satellite Switched Time Division Multiple Access
SSL	Space Systems/Loral, LLC
TTL	Transistor-Transistor Logic

1.0 INTRODUCTION

A Radio Frequency Switch Matrix provides three practical applications in the spacecraft industry. The first is to provide on-orbit redundancies of payload units in the event of a unit failure. The second is to improve satellite signal performance by selecting the strongest signals received and routing them to other payload units, and the third application is for improving efficiency of radio frequency ground testing. This project focuses on the latter application.

1.1MOTIVATION

SSL is currently conducting a feasibility study into the potential of upgrading their current payload test equipment. The two most viable options are to either purchase expensive equipment or to develop in-house equipment. While many units are required to be designed and built to create a full suite of radio frequency test equipment, this project focuses on determining the feasibility of designing, manufacturing, and integrating a completely COTS RF Switch Matrix.

1.2LITERATURE REVIEW

Radio Frequency Switch Matrices have been used since some of the earliest communication satellites. Intelsat VI used two 6x6 Switch Matrices to carry 120 Mbit/s traffic over its frequency bands [1]. According to R. Gupta, “the use of microwave switch matrices (MSMs) on-board communications satellites enhances satellite capacity by providing full and flexible interconnectivity between signal of up- and down- link beams. The frequency reuses resulting from the introduction of microwave switching for satellite switched time division multiple access (SS-TDMA) operation significantly enhance the satellite utilization efficiency” [2]. In other words, radio frequency performance can be improved through a switch matrix by

selecting the strongest signal received from an antenna and routing it through the switch matrix to a downconverter or digital signal processing (DSP) hardware [3]. Another important application of Switch Matrices is to provide system redundancy between RF units [4]. In addition to satellite applications, Switch Matrices can be used in a similar fashion for ground stations to boost efficiency [5].

Figure 1.2-1 below shows an example of a 4x4 matrix that could be used to route RF signals from any one of four inputs to any one of the four outputs:

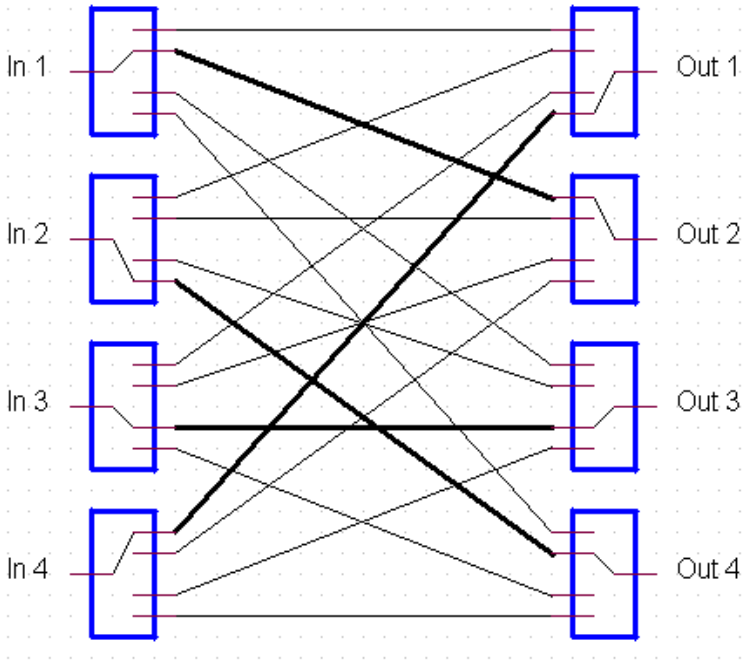


Figure 1.2-1: 4x4 Switch Matrix Function View

[6]

The newest promise in improvement in Switch Matrix technology is with the use of Micro-Electro-Mechanical Systems (MEMS) switches [7]. MEMS switches take advantage of the benefits of both mechanical and

semiconductor switches. They provide mechanical movement to open and close the switch configuration similar to mechanical switches to provide better signal isolation, but are compact like semiconductor switches [8]. E. Siew et al. has developed a 3x3 prototype RF MEMS Switch Matrix which is capable of “working up to 60GHz with good RF performance such as return loss of at least 20dB, isolation of at least 78dB and insertion loss of at most 0.58dB. The proposed design (27×45μm2) is at least 10, 000 times more compact than the previously reported RF switch matrices. In addition, the 3×3 MEMS switch matrix can be easily expanded to larger matrices using Clos network” [9]. MEMS technology shows great promise for future satellite usage to reduce the weight of RF systems. While this technology meets all of the requirements of this project, and merits additional research, RF ground test equipment has no requirement on size and weight of the project. Therefore MEMS technology will not be pursued for this project.

This project is a feasibility study to determine the company’s ability to develop an inexpensive Switch Matrix. Therefore the majority of research into this topic has been on commercially available COTS products. One of the leading suppliers of Switch Matrices is Agilent Technologies, which has been providing Switch Matrices for over 20 years [10]. A Switch Matrix is just one unit required for RF Testing Equipment. A standard Agilent Test System Rack includes many other crucial components that are mounted together in a portable rack. Some other components that are part of the rack include a spectrum analyzer, signal generators, reference sources, and filters [11].

Each Agilent Switch Matrix is customized to their customer’s needs, meaning each Switch Matrix is unique. The various attributes that can be customized includes the number of input and output ports, the frequency of operation, power specification, equal path length, switching speed, and signal conditioning [12].

There are 3 types of Switch Matrix designs for RF Pathways. Each method has advantages and disadvantages.

- Common Highway. Figure 1.2-3 shows such an example. The advantages of a common highway are that it is the simplest and lowest cost design for a switch matrix. However, you can only connect to one output at a time. [12]

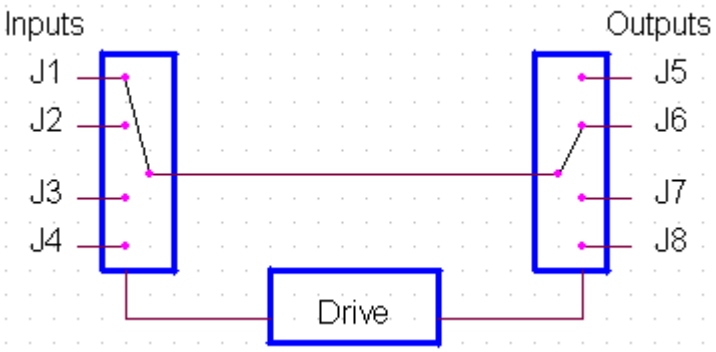


Figure 1.2-2: Common Highway Design

[12]

- Full Access (Blocking). Figure 1.2-4 shows such an example. The full access (blocking) design allows the use of multiple active channels at the same time. This provides higher flexibility for RF testing. This added flexibility has the disadvantage of a higher cost and the limitation of any input port can connect to only one output port. [12]

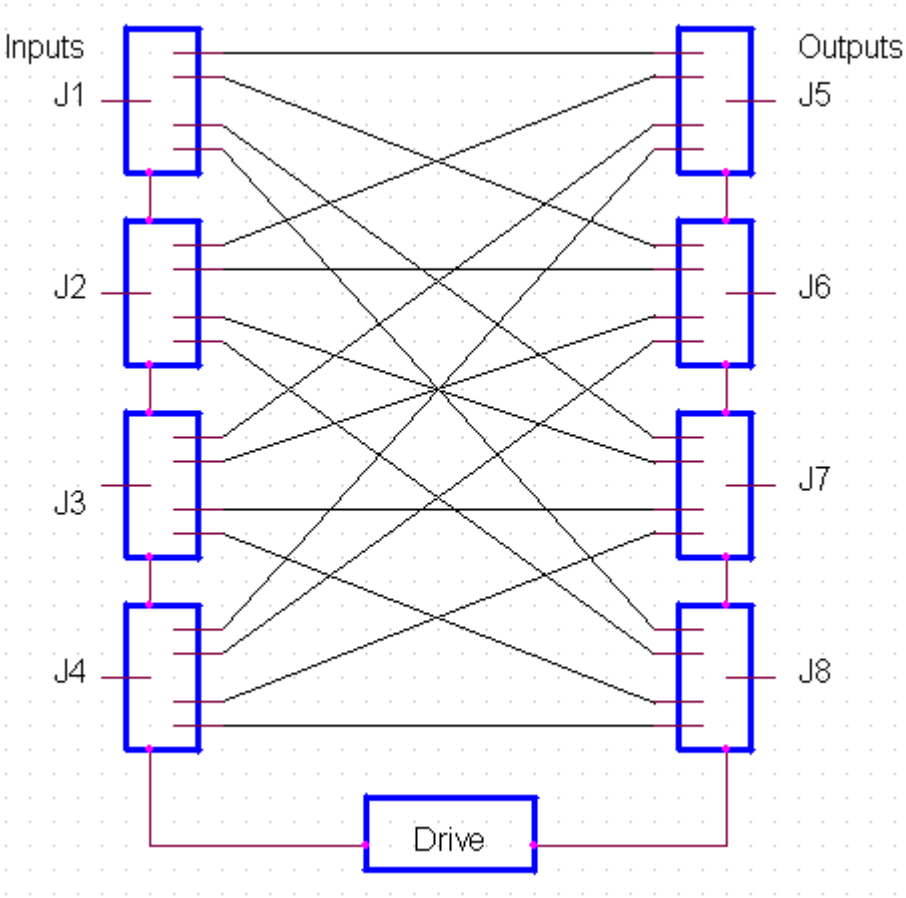


Figure 1.2-4: Full Access (Blocking) Design

[12]

- Full Access (Non-Blocking). Figure 1.2-5 shows such an example. This design allows connections between any input port to any output port simultaneously, multiple active channels, and can connect any input port to all output ports at the same time. The disadvantage of this high flexibility design is cost and a reduction of RF performance. There is low isolation between output ports connected to the same input port, the bandwidth is limited by a power divider, and there is a higher insertion loss. [12]

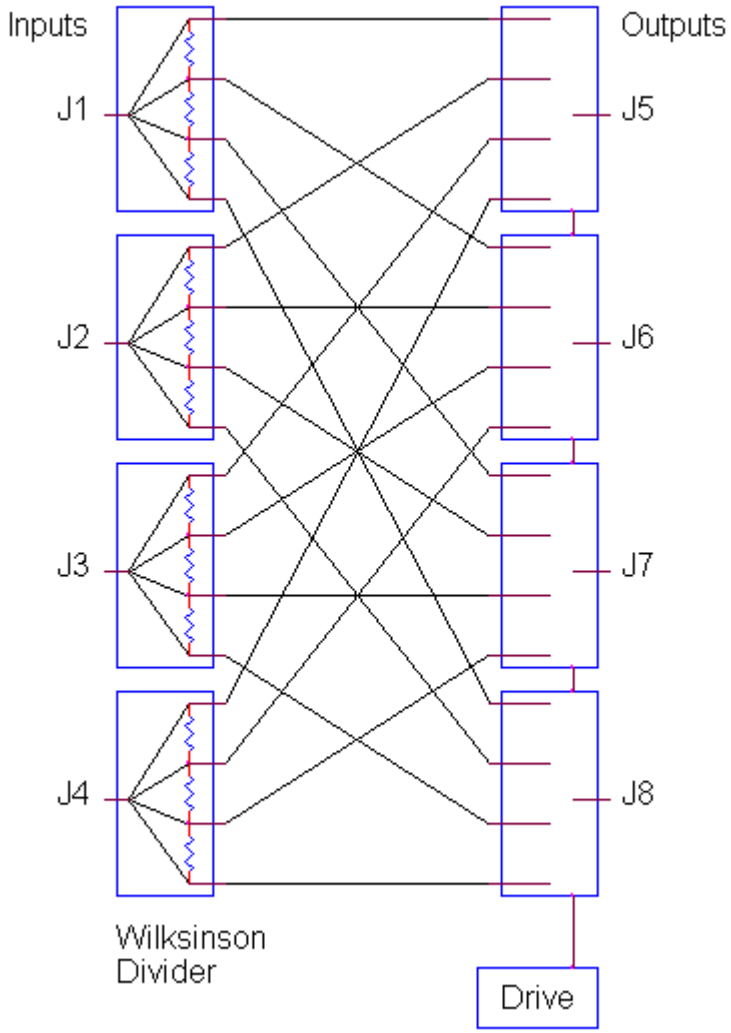


Figure 1.2-5: Full Access (Non-Blocking) Design

[12]

A standard Agilent Switch Matrix is designed to have the primary components located in the following locations. The front panel of the Switch Matrix unit has the input and output ports of the switches. On the inside of the unit, coax cables are routed from the interface ports to their respective switch ports. In the aft section of the unit, the switch control circuit boards, power supply, and the power connectors, which provide the voltage and current to switch the individual switches, can be found [13].

2.0 SYSTEM REQUIREMENTS

The requirements of this project are flowed down from SSL to interface with existing payload test equipment. The requirements are broken into three subcategories: Radio Frequency Requirements, Electronic Requirements, and Mechanical Requirements.

Radio Frequency Requirements:

- One uplink RF Path and One downlink RF Path shall be provided.
- Each RF Path shall consist of six inputs and six outputs.
- RF can enter through any input and exit through any output.
- Each Path shall have an RF test coupler to allow for signal verification.
- Available frequency of switch matrix shall be from 400 MHz to 31.5 GHz at low power <30 dBm max.
- Capability of using 3 cable power calibration method shall be available.

Electronic Requirements:

- All switches shall be controllable via SCPI commands via IEEE 802.3 Ethernet port.
- Telemetry of all switch positions shall be provided.

- The number of actuations per switch shall be recorded and stored between power-off cycles.

- The time from any command sent to operation execution and telemetry response shall be less than 10 seconds.

- If communication is lost, all switches shall remain in the present configuration.

- If communication is lost, telemetry shall be restored to show current configuration.

- A serial interface display shall display switch positions and status.

- Switch Drive Matrix (SDM) shall be designed to have the capability of commanding up to 16 switches.

- A Failsafe button to reset switches to safe position shall be provided.

- Watchdog timeout trigger shall be initiated if controller is unresponsive.

Mechanical Requirements:

- The Switch Matrix shall be built into standard 2U 19” chassis.

- Space and design for adding power detection/calibration devices shall be provided.

- Equal cable lengths shall be provided for equal RF losses.

- The Switch Matrix shall have the ability to add 3 dB pads for return loss improvement.

3.0 SYSTEM OVERVIEW DESIGN

The primary requirements have been defined for this project. Below is a preliminary high level concept design which shows the system overview of how the switch matrix is designed and built for this project.

The components are as followed. See the following page and Appendix A for a system overview drawing.

- 4 Switches that each provide 6x1 positions.

- 2 Switch per pathway using Common Highway Design.

- There are RF Instruments between the switch paths to make RF measurements as required.

- An ATmega 2560 microcontroller with an Ethernet Shield will be used to control the switches.

- To control switches, a Control Circuit will be designed to be able to pulse each switch.

- To collect telemetry, a Telemetry Circuit will be designed to gather switch position Telemetry and display to a visible display for the user.

- An Interface Board will need to be designed to connect all of the switch connectors and route all of the wiring to the appropriate board.

- 28V Power Supply to provide power to the Control Board to switch the switches.

- Power Regulator Board to regulate the voltage from the 28V Power Supply to supply the microcontroller required voltage and current.

- Watchdog Circuit to provide a heartbeat of the system. In the case of a failure, the watchdog will reset the microcontroller. This will allow the switch matrix to be used for months at a time without a system failure.

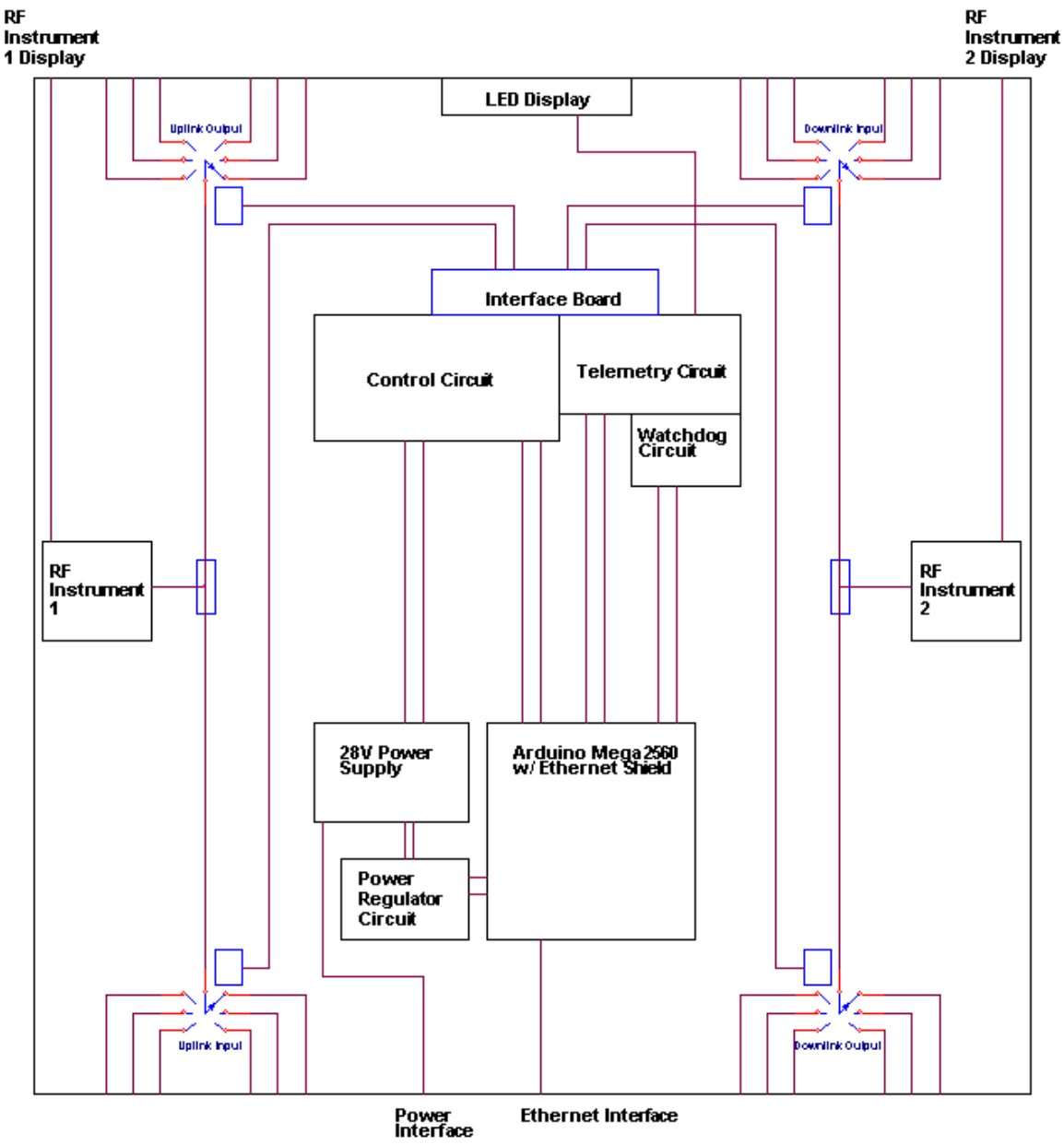


Figure 3-3: Switch Matrix System Overview

See Appendix A for a larger diagram of the Switch Matrix System Overview.

4.0 CRITICAL HARDWARE

4.1MICROWAVE SWITCHES

The switches for this project were provided by SSL. They are the given switches in which then entire switch matrix is designed around. Four Radiall R583833250 switches are used in the switch matrix design. Each switch is a six to one pathway switch ranging from 0 to 40GHz. The below figure is an image of a switch used in the Switch Matrix.

Figure 4.1-1: Radiall R583833250 Switch

Each switch is a latching switch that actuates via a 28V, 125mA pulse. The nominal switch time is less than 15ms. However as a factor of safely, the programming pulses the switch for 75ms. The switch has a 25 pins D-SUB male connector on the aft end of the switch.

The wiring diagram below is the schematic for one position within the switch. Each position in the switch has one forward coil and one reverse coil.

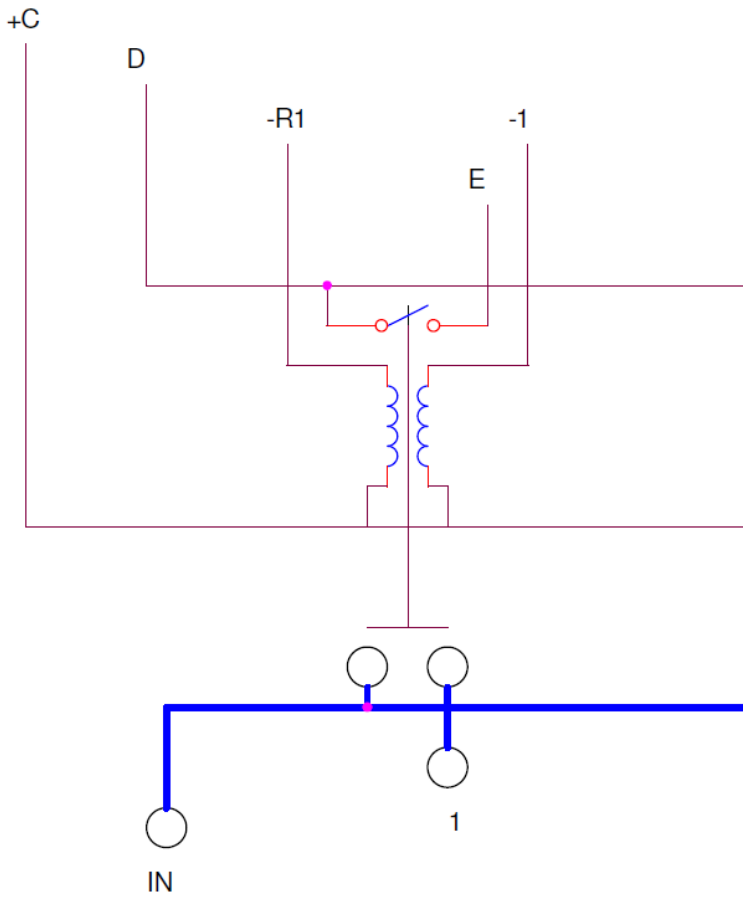


Figure 4.1-2: Single Position Schematic

To activate the coil to choose a particular switch position, the positive 28 volts enters in the +C wire and exits out the -1 wire. This creates a magnetic field due to the coil and pushes the center bar down. When the center bar is down, it creates a pathway for the RF signal to pass out that particular port. In addition, it also closes the latch between wires D and E. This provides a telemetry position. To open the RF switch position, a 28V pulse would enter through the +C wire and exit out the –R1 wire. This creates a magnetic field in the opposite direction and pushes the center bar up. This opens the RF pathway and opens the telemetry relay. A schematic of all switch positions and how they are tied together can be found in Appendix B.

4.2CONTROL SYSTEM

The Switch Matrix is controlled by an Arduino Mega 2560 microcontroller. This microcontroller uses an ATmega 2560 chip that runs at 16MHz. The microcontroller consists of 54 digital input/output and 16 analog inputs pins. Each output pin is able to output 5 volts at 40mA. The Arduino platform was chosen due to its programmable flexibility. The Arduino programming language is based off of the C++ programming language. All standard C++ libraries and functions are available to the Arduino. Uploading the code to the Arduino board only requires a standard USB connection. Below is a figure of the Arduino Mega Pro microcontroller created by Sparkfun Electronics.

Figure 4.2-1: Sparkfun Arduino Mega Pro 5V

Source: https://www.sparkfun.com/products/11007
Creative Commons images are CC BY-NC-SA 3.0
Photo taken by Juan Peña

This particular microcontroller was selected because of the versatility of the input and output pins. Pins that were required for this project had wire-wrap pins soldered to the board so the board is able to fit into a 0.1” prototype circuit board.

4.3LAN / ETHERNET CONNECTION

The Arduino Mega 2560 alone is unable to connect via LAN by itself. To be able to transmit and receive commands and telemetry via Ethernet connection, additional hardware is required. To meet the demands of this project and to appropriately interface with the Arduino Mega 2560, the Arduino Ethernet Shield has been selected for this project. The Ethernet Shield is controlled by a W5100 Ethernet chip capable of 10/100Mb connection speed. The Shield has a standard RF-45 connection that is IEEE 802.3 compliant. Below is a figure of the Arduino Ethernet Shield.

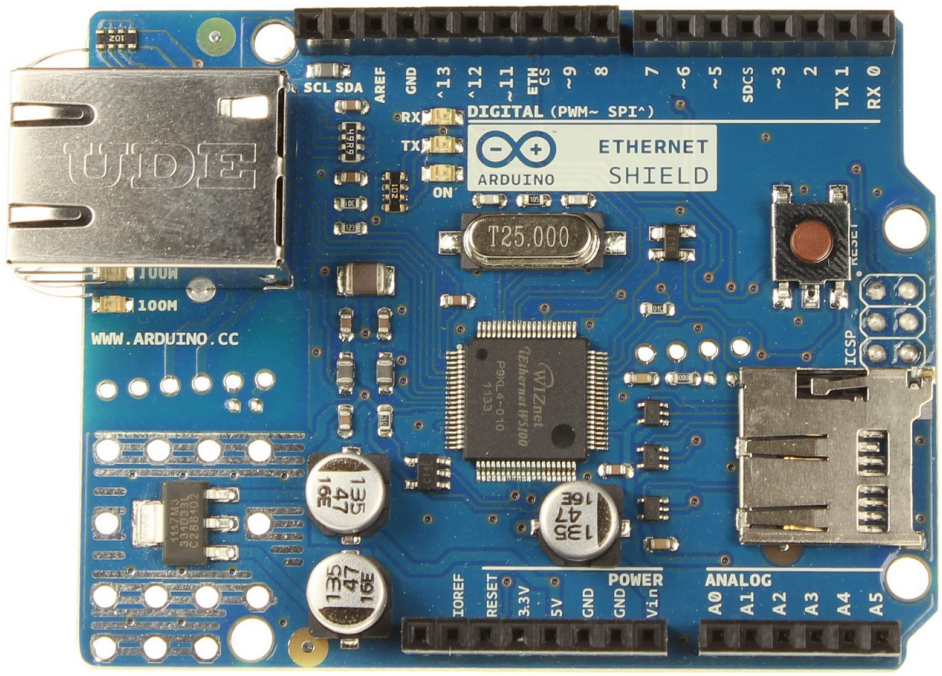


Figure 4.3-1: Arduino Ethernet Shield

Source: http://arduino.cc/en/Main/ArduinoEthernetShield
Creative Commons images are CC BY-SA 3.0

The Ethernet Shield communications to the Arduino Mega 2560 via the SPI bus which is through the ICSP headers on both boards. This includes the following pins: MOSI, MISO, SCK, Reset, +5V, and return pins. In addition,

the slave select pins on both boards are used to select the W5100 chip on the Ethernet Shield.

5.0 SWITCH DRIVE MATRIX CIRCUIT

A Switch Drive Matrix Circuit was designed to control the switches. This method allows for the most amount of available switches with the least amount of components and wiring. The method allows for every individual switch position for all switch command ground lines be wired together. In addition, the six return position lines are tied together, essentially making the return line, R, one pathway for six coils. Figure 5-1 below shows the configuration of the Switch Drive matrix.

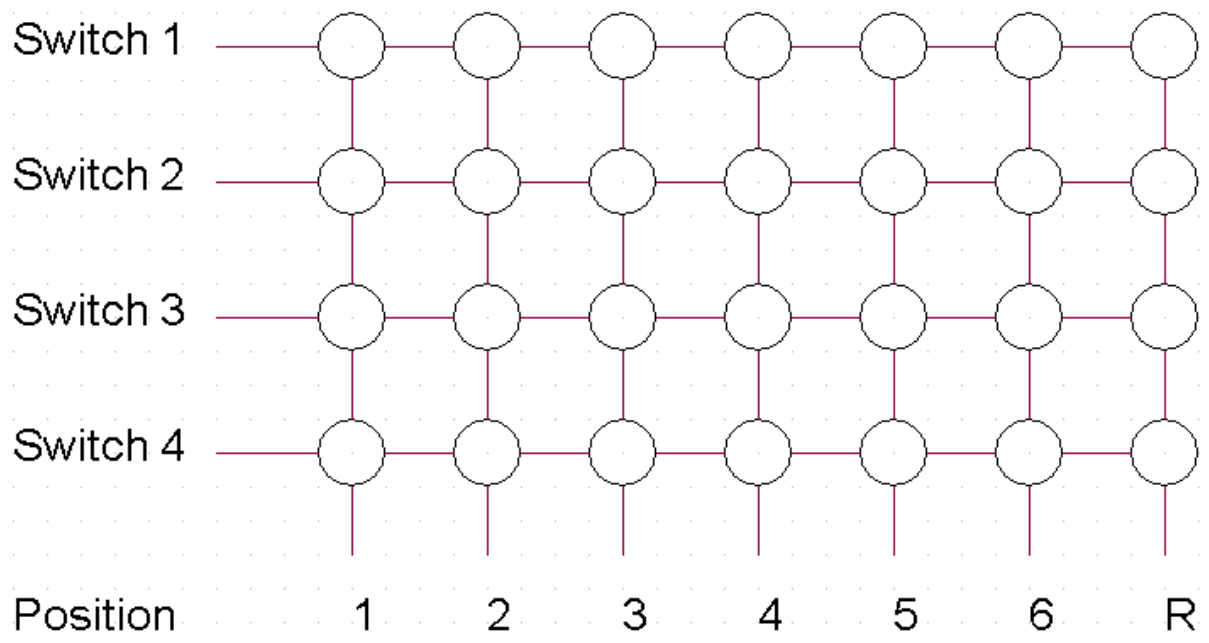


Figure 5-1: Switch Drive Matrix Concept

For example, to pulse switch 1 to position 4, the switch 1 select line would be activated and the position 4 select line would be activated. This would allow current to flow through the switch 1 position 4 coil and activate that position. Since the other switch select lines and position select lines do not complete an electrical circuit, there is no current to activate those coils.

Figure 5-2 below shows the electrical schematic for a single coil.

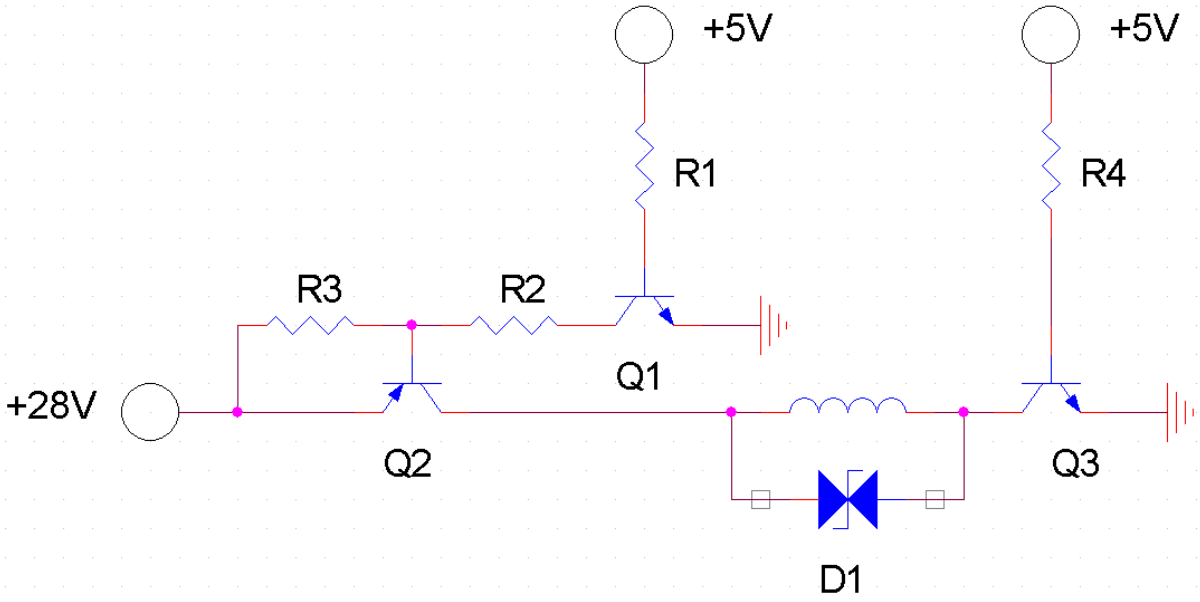


Figure 5-2: Single Electrical Pathway Schematic

The above pathway design can be broken down into two parts; the Switch Select Transistor (Q2) and the Position Select Transistor (Q3). Transistors are used for this project over relays due to the limited life expectancy of relays. A typical electromechanical relay has a life expectancy of one to two million cycles. The switches used in the project have a life expectancy of two million cycles per position. This would mean that the control circuit would fail prior to the switches failing.

5.1Switch Select Component Design

To active the switch select transistor, a PNP transistor is used. PNP transistors are used for high side switching, which means that PNP transistors are used prior to the circuit resistive load, close the voltage source. This is opposed to an NPN transistor which is used for low side switching after the circuit resistive load, close to ground.

For the PNP transistor to work, if the base of the transistor is lower than the voltage at the emitter, current will flow from the emitter to the collector. Since the microcontroller can only output 5V, a special circuit was designed to nominally keep the base of the transistor Q2 at 28V. NPN transistors only let current flow from collector to emitter when the base voltage is higher than the voltage at the emitter. NPN transistors are the only transistors in this design that can be directly actuated by a microcontroller.

The theory behind the switch select high switching circuit is as follows. Resistor R3 supplies the base of transistor Q2 with 28V, preventing current to pass through transistors Q2. When the user wishes to activate the switch select, the microcontroller sends current through resistor R1. Resistor R1 is sized to allow the correct amount of current to fully saturate transistors Q1. When transistor Q1 is fully saturated, it allows current to flow from its collect to its emitter. Since the emitter on transistor Q1 is grounded, the current that is nominally preventing transistors Q2 from saturating flows to ground. The amount of current that flows to ground is determined by R2. While transistor Q1 is saturated by the microcontroller, the base of transistor Q2 has now been set to a lower voltage than its emitter, which allows current to flow from its emitter to collector, activating the desired switch. In short terms, this design pulls the voltage at the base of Q2 down when transistor Q1 has been activated.

The following are the calculations for the different components in this design:

First, the maximum current that will pass across transistor Q2 is calculated. Per section 4.1, the nominal activation current for a coil is 0.125 amps. To reduce the amount of wires and components, the six return coils in the switch are tied together, allowing only one master reset command to activate all six return coils. This means that the maximum current across transistor Q2 is:

$$I_c = 0.125\,Amps * 6\,coils$$

$$I_c = 0.75\,Amps$$

This value is the collector current at transistors Q2. Since this transistor has to pass 0.75 amps at 28 volts, PNP power transistor TIP32 is used. The TIP32 transistor works up to 40 volts and 3 amps with a minimum gain (H_{fe}) of 10. Note that minimum gain is used in all calculations because it is a worst-cast value.

Due to the gain of transistor Q2, transistors Q1 will not need to sink as much current to activate the TIP32.

$$I_c = I_b * H_{fe} \qquad (5.1)$$

Where I_b is the base current.

Rearrange the equation

$$I_b = \frac{I_c}{H_{fe}}$$

$$I_b = \frac{.75}{10} = 0.075\,A$$

This means that transistors Q1 must be able to sink 0.075 amps to activate transistor Q2. Power transistors TIP31 has been selected as transistor Q1. It has the identical properties as the TIP32, but an NPN transistor. Below is a figure of critical information about the TIP31 and TIP 32.

TIP31 and TIP32 Critical Data		
V_{CEO}	Maximum Collector-Emitter Voltage	40
I_C	Maximum Collector Current (amps)	3
H_{FE}	Minimum DC Current Gain	10
V_{CE}	Collector-Emitter Saturation Voltage	1.2
V_{BE}	Base-Emitter Saturation Voltage	1.8

Figure 5.1-1: Critical TIP31 & TIP 32 Transistor Data

Next, to make sure 0.075 amps is able to flow to ground, resistor R2 is calculated from Ohm's Law.

$$R=\frac{V}{I} \qquad (5.2)$$

For this calculation, V_{r2} is the voltage across resistor R2 and I_{r2} is the current across resistor R2.

$$R_2=\frac{V_{r2}}{I_{r2}}$$

To calculate V_{r2}, the voltage drop across the base-emitter voltage (V_{be}) of Q2 and the Collector-emitter(V_{ce}) must be taken into account.

$$V_{r2}=V-V_{be2}-V_{ce1} \qquad (5.3)$$

$$V_{r2}=5-1.8-1.2=2V$$

Completing equation 5.2:

$$R_2=\frac{2}{.075}=26.6\acute{6}\,\Omega$$

The closest standard resistor value is a 24 Ω resistor which is used.

The next value to calculate is resistor R1. The value of R1 determines how saturated transistor Q1 is by limiting the current that flows over it. Modifying equation 5.1:

$$I_{b1}=\frac{I_{c1}}{H_{fe}}=\frac{.075}{10}=0.0075\,amps$$

This value means that it will take 0.0075 amps to drive transistor Q1 to saturation. Since this current comes directly from the microcontroller, it is not always a precise value. Therefore a safety factor of 2 shall be included to guarantee that 75 mA is sunk to ground. Therefore I_{b1} = .015 amps.

A similar process using equations 5.2 is repeated except now to calculate voltage, only the base-emitter voltage drop across transistor Q1 is taken into account.

$$V_{r2}=V-V_{be1} \qquad (5.4)$$

$$V_{r2}=5-1.8=3.2V$$

Completing equation 5.2:

$$R_2=\frac{3.2}{.015}=213.3\,\Omega$$

The closest standard resistor value plus an additional margin is a 300 Ω resistor which is used.

Resistor R3 is used to supply the 28 volts at the base of transistor Q2. The only requirement for resistor R3 is that is must be at a high enough

value. If the value is too low, there would be enough current flow to both resistor R2 and to prevent transistor Q2 from fully turning on. The general technique is to make resistor R3 at least an order of magnitude greater than R2. As an extra margin a safely for this circuit, roughly two orders of magnitude are used for resistor R3. Therefore resistor R3 is using a 2.2k Ω resistor.

5.2 Position Select Component Design

Position selection design is much simpler than switch selection. Because the position selection is downstream of the coil, only a low side switch concept is required. Only a NPN transistor with a current limiting base resistor is required to activate the return side of the circuit. However, there are two versions of the return side: The first is when activating a single coil to move to a particular switch position. This requires the use of only one coil. The second is when activating the six return coils. This pathway requires different components.

5.2.1 Single Coil Low Side Switching

As mentioned previously, a single coil requires 125 mA to activate. NPN transistor TIP31 has been selected to be used as transistors Q3 due to high tolerance levels and gain value. The following are the calculations to calculate values for resistor R4. Using equation 5.1, the following values are determined:

$$I_b = \frac{I_c}{H_{fe}} = \frac{0.125}{10} = .0125\,A$$

This value means that it will take 0.0125 amps to drive transistor Q3 to saturation. Since this current comes directly from the microcontroller, it is not always a precise value. Therefore a safety factor of 2 shall be included

to guarantee that 12.5 mA is sunk to ground. Therefore I_b = .025 amps. Using equation 5.4, the voltage across resistor R4 can be calculated.

$$V_{r4}=V-V_{be}=5-1.8=3.2\,V$$

Using equation 5.2:

$$R_4=\frac{V_{r4}}{I_{r4}}=\frac{3.2}{.025}=128\,\Omega$$

The closest standard resistor value is a 120 Ω resistor which is used.

5.2.2 Return Coil Low Side Switching

In section 5.1, it was determined that 0.75 amps were required to activate all six return coils at once. Therefore while the equations and concepts are the same as section 5.2.1, a different transistor is required. In section 5.2.1, only 25 mA were required to saturate the transistor. The microcontroller is able to output 40 mA, so there is no issue. However, if a TIP31 is used for return coil low side switching, it would require a 150 mA pulse from the microcontroller. Since this is not possible, a different power transistor with a higher gain must be used. For this project, the TIP120 was selected because of its higher gain value. The TIP120 is an NPN Darlington Transistor. A Darlington Transistor is essentially two transistors together to increase the current gain. Below is a figure of critical information about the TIP120:

TIP120 Critical Data		
V_{CEO}	Maximum Collector-Emitter Voltage	60
I_C	Maximum Collector Current (amps)	5
H_{FE}	Minimum DC Current Gain	1000
V_{CE}	Collector-Emitter Saturation Voltage	2
V_{BE}	Base-Emitter Saturation Voltage	2.5

Figure 5.2.2-1: Critical TIP120 Transistor Data

Following the same calculations as section 5.2.1:

$$I_b = \frac{I_c}{H_{fe}} = \frac{0.75}{1000} = .00075\,A$$

A factor of safely of two is added to the current is included. Therefore I_b = .0015 amps. Using equation 5.4, the voltage across resistor R4 can be calculated.

$$V_{r4} = V - V_{be} = 5 - 2.5 = 2.5V$$

Using equation 5.2:

$$R_4 = \frac{V_{r4}}{I_{r4}} = \frac{2.5}{.0015} = 1666.\acute{6}\,\Omega$$

The closest standard resistor value is a 1.6k Ω resistor which is used.

The final component in the Switch Drive Matrix Circuit is the bi-directional zener diode D1. A snubber diode is required across all inductive loads. In the case of this project, the activation coils are inductive loads. If a snubber

diode is present, the current will loop through the coil and dissipate through the magnetic field. If a snubber diode is not present parallel of an inductive load, and current is pulsed through the load; the voltage across the inductive load will begin to rise to a point of damaging components. For this project, the TIP31 and TIP32 transistors will break if over 40 volts are placed across the collector. However, since the low side of all the switches are tied together, a standard diode would cause other switches to have unwanted activations.

The bi-directional zener diode, also known as a transient voltage suppressor, is a semiconductor device that does not let current pass in any direction unless a certain voltage level is reach. Once that voltage is reached, the diode will break down and let current pass until the voltage is reduced. In this case, the bi-directional zener diode is set to break down at 36 volts. A 1.5KE36CA transient voltage suppressor is used in this design. The concept is that when the inductive coil voltage rises and hits 36 volts, the zener diode will break down and let current pass through back to the coil and dissipate itself. 36 volts was chosen because the switch coils only activate within a voltage pulse from 24 to 30 volts and will not activate at 36 volts, but leave enough margin of safety to protect the electrical components that break at 40 volts.

See Appendix C for the schematic of the entire Switch Drive Matrix Circuit, including the different pathways. The Switch Drive Matrix is easily expandable to include additional switches. Adding an extra switch would only require adding two transistors and three resistors as the low side switching components are shared between all switches.

6.0 TELEMETRY CIRCUIT

The telemetry circuit provides real time status updates of all switch positions. The circuit is divided into two parts: switch position telemetry and an LCD Display.

6.1SWITCH TELEMETRY

Switch position is provided by internal wiring in each switch. As shown in Appendix B, there is a telemetry wire that runs through the switch. Similar to the command terminals, there is a common telemetry terminal and six position terminals. When the switch is in an open position, relays for the individual positions are open. This breaks continuity through the telemetry wire. When a switch has been placed in a particular position, the corresponding telemetry relay is closed which provides electrical continuity through the telemetry wire. See Figure 6.1-1 below for a switch telemetry diagram.

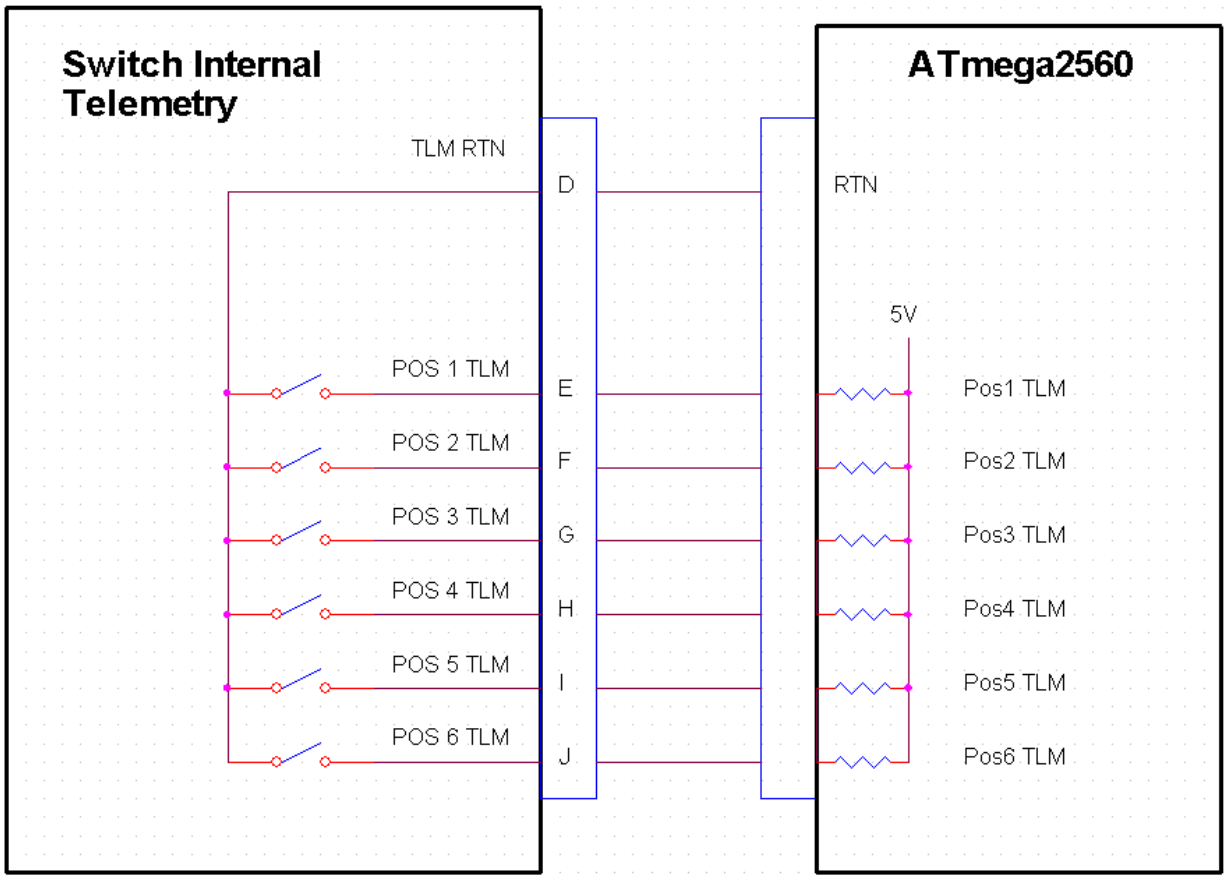


Figure 6.1-1: Switch Telemetry Diagram

To be able to read whether a telemetry relay is closed or open, the microcontroller is programmed to read the positions. The microcontroller digital and analog input pins are able to read voltage readings between zero and five volts. To prevent floating values for each of the microcontroller input pins, a pull up resistor to five volts to used. This allows the input pin to read a high (+5V) state. The advantage of using an Arduino Microcontroller is that the ATmega2560 chip has internal 20k Ω pull up resistor that can be activated via software. The telemetry common terminal is wired to the Arduino ground pin.

When a telemetry relay is closed, a path to ground is created. This pulls the microcontroller input pin to a low (0V) state. The programming, which is described in more detail in Section 11 reads all of the switch position states

and determines which pins read a high or a low state and is assigned to a particular switch position.

6.2DISPLAY

There are two ways for the user to determine current switch positions. The first is through the General User Interface (GUI) by sending SCPI commands, or through an LCD screen on the front of the unit that displays all current switch positions. This section focuses on the design of the latter. The GUI is described in greater detail in Section 11.

The LCD screen used in this project is a Sparkfun Serial Enabled 16x2 LCD screen. The screen displays red on black characters and is controlled via a 5V TTL serial input. The below figure is an image of the LCD screen used for this project.

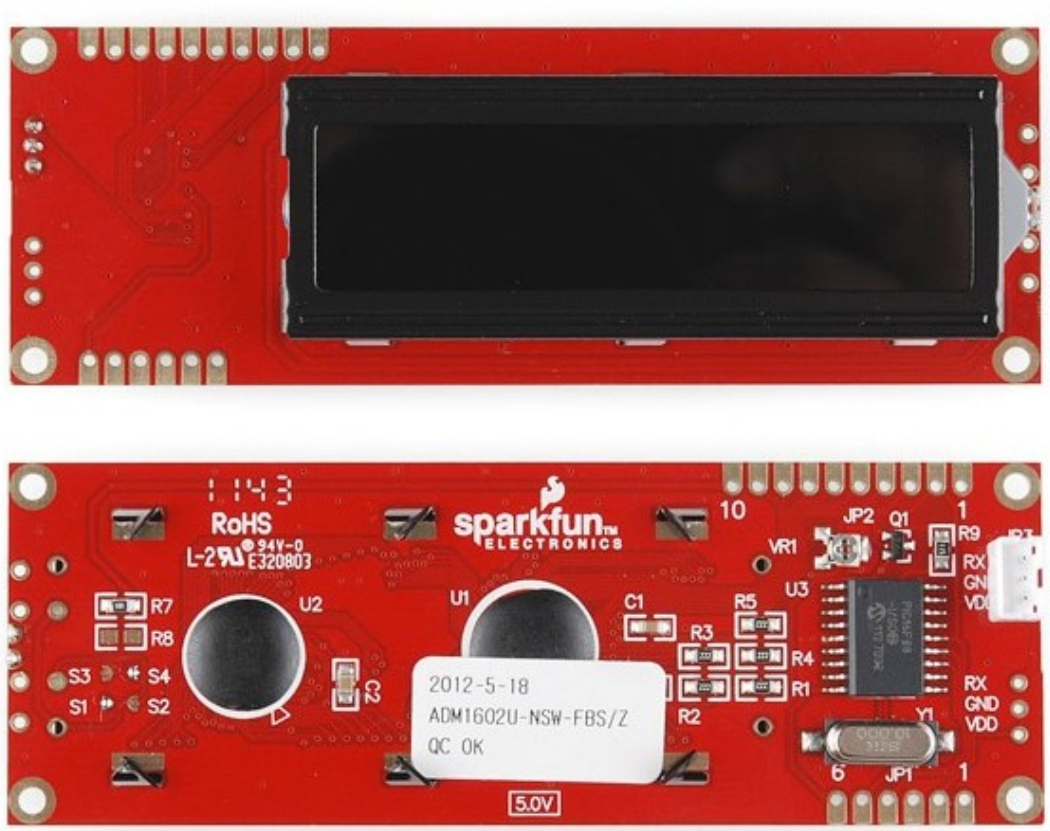


Figure 6.2-1: Sparkfun LCD Screen

Source: https://www.sparkfun.com/products/9394
Creative Commons images are CC BY-NC-SA 3.0
Photo taken by Juan Peña

The LCD screen is controlled via three pins: a power pin, a ground pin, and a receive pin. The power pin is wired up to the microcontroller +5 volt pin. This provided the LCD screen the voltage and current for the LCD backlight. The ground pin is wired to the microcontroller ground pin to ground the power supply of the LCD screen. The receive pin is wired to a microcontroller digital output pin. This pin controls the characters on the LCD screen. The Sparkfun screen uses standard Arduino code serial libraries which make the device easy to incorporate. See the figure below for a wiring diagram of the LCD display.

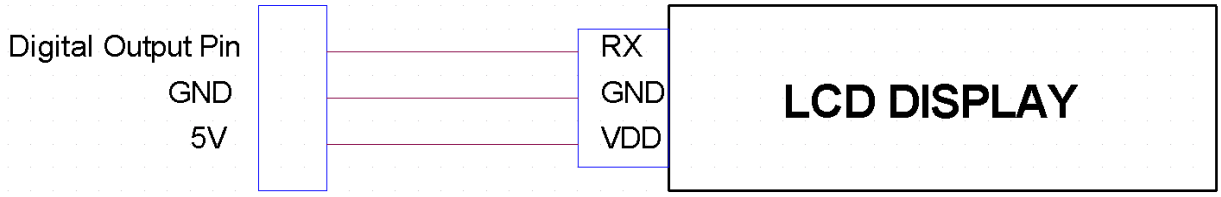


Figure 6.2-2: LCD Wiring Diagram

Switch position telemetry is displayed on the first row of the LCD display. Since the LCD screen does not require a delay between displaying values, switch position telemetry is displayed as real time values for the user.

The second row of the LCD display is reserved for displaying the last SCPI command sent and the heartbeat of the system. As mentioned previously, the microcontroller is commanded via SCPI protocol. The user can send SCPI commands to either control the switch or request telemetry. Section 11 will describe the SCPI protocol used in this project. The LCD screen displays the last SCPI command sent to system starting on the first position of the second line of the screen. The heartbeat display feature will also be discussed in Section 11.

See Appendix D for a full schematic of the telemetry circuit. It includes wiring for the four switches used and the LCD display.

7.0 INTERFACE CIRCUIT

The Interface Circuit is broken up into two major parts. The first part is an interface cable that connects the 25 pin male D-SUB connector on the aft end of the Radiall Switch to a 26 pin ribbon connector. A 26 pin ribbon connector is used because the Interface Circuit is built on a 0.1” prototype circuit board, and the ribbon connector mates with this spacing. The cable itself has a 25 pin female D-SUB connector and a 26 pin female ribbon cable connector. See Figure 7-1 below for an image of the interface cables used.

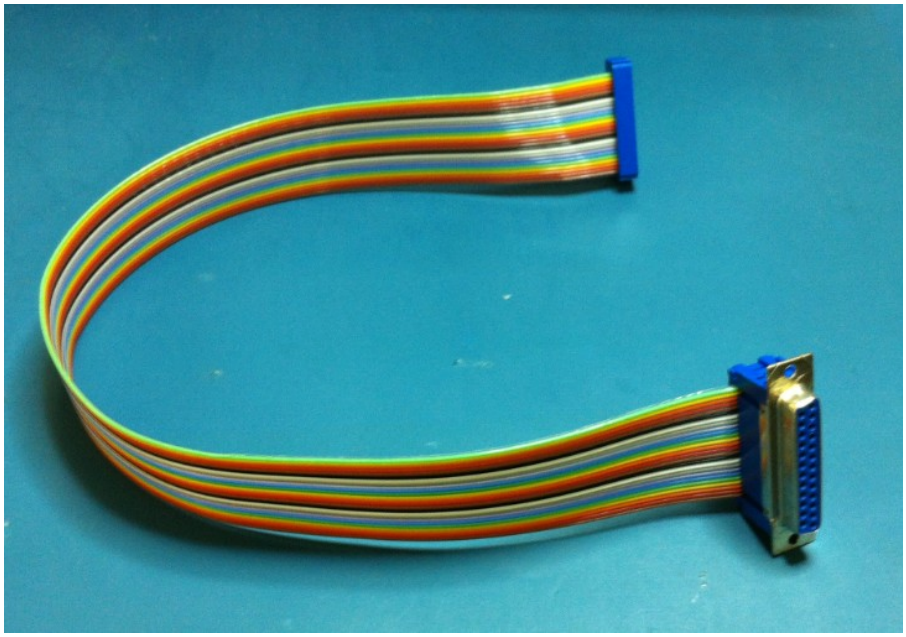


Figure 7-1: Interface Cable

The second part of the Interface Circuit is the board itself. The board routes all telemetry and command wires from the switches to the appropriate command and telemetry circuits. This organizes the harness wiring to aid in building and any required troubleshooting.

In addition, all return command paths are equipped with a 1N4007 diode in line to the ground to prevent current leakage and unwanted stray switch actuations.

For the full Interface Circuit Schematic, see Appendix E for a detailed view of command and telemetry wire routing.

8.0 POWER REGULATION CIRCUIT

Power for the Switch Matrix uses the following scheme. AC Voltage enters through a Qualtex 862-06/002 EMI Power Line Filter. The filter has an IEC connector for AC voltage wired in series with a fuse and an on/off power switch. On the output of the filter, the ground line is connected to the Switch Matrix chassis ground, and the hot AC lines are wired into an Acopian AC to 28V DC Power Converter. This converter is supplied by SSL. The 28V converter supplies the required voltage to actuate the Radiall switches. However the microcontroller requires an input voltage between 4 and 12 volts.

In this instance, 9 volts was selected to power the microcontroller. A minimum of 7 volts is required to output a stable 5 volts on the digital output pins. To achieve a stable 9 volt supply to the microcontroller, a NJM78M09FA Voltage regulator is used. This voltage regulator can input a maximum of 35 volts and outputs 9 volts at a maximum of 500 mA.

A voltage regulator requires an input capacitor and an output capacitor. The input capacitor prevents oscillation and reduces the power supply ripple. An input capacitor of 0.33 mF is wired between the 28V input and ground. The output capacitor aids phase compensation of the internal error amplifier of the regulator. An output capacitor of 0.1 mF is wired between the 9 volt output and ground. Below is a schematic of the 9V power regulation circuit.

A full schematic of the power scheme used for this Switch Matrix can be found in Appendix F.

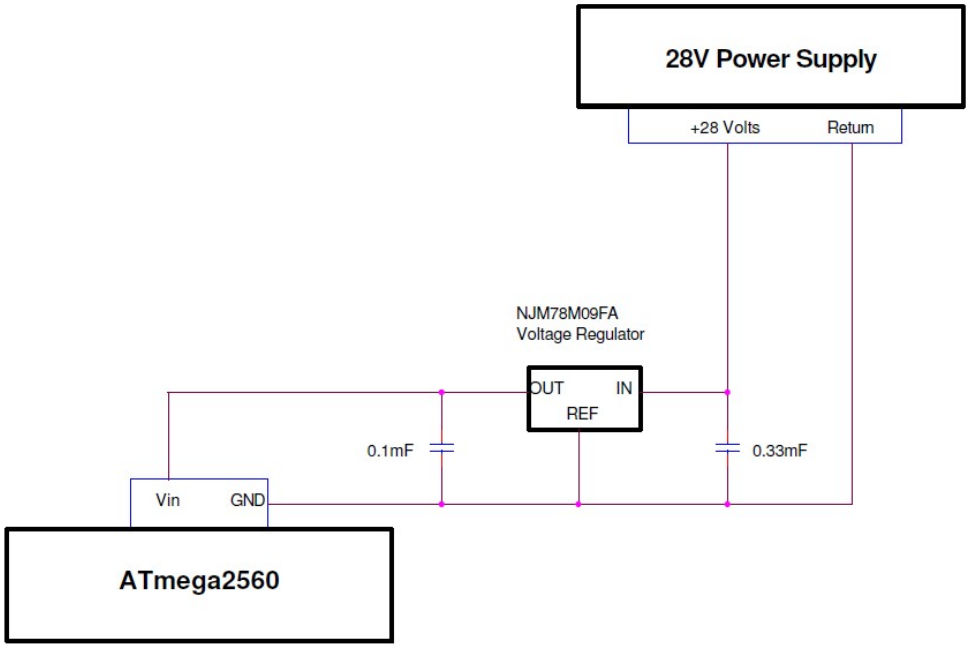


Figure 8-1: 9 Volt Power Regulation Circuit

An additional consideration to take into account is the heat generated from the voltage regulator. Power generated is equal to the voltage times the current.

$$P = V * I \qquad (8.1)$$

In the case of a voltage regulator, the voltage is the voltage delta between the input and the output of a voltage regulator. The nominal current draw of the microcontroller is 250 mA. However the regulator can output up to 500 mA. The minimum and maximum power dissipated is as follows:

$$P_{MIN} = (28 - 9) * 0.25 = 4.75\ Watts$$

$$P_{MAX} = (28 - 9) * 0.5 = 9.5 Watts$$

Therefore, worst case, the voltage regulator will have to dissipate 9.5 watts. This is much more than the regulator can dissipate on its own. Therefore, a large heat sink was mounted and thermal paste was applied to the voltage regulator to aid in the heat dissipation. The figure below is an image of the built 9 volt power regulator with its attached heat sink.

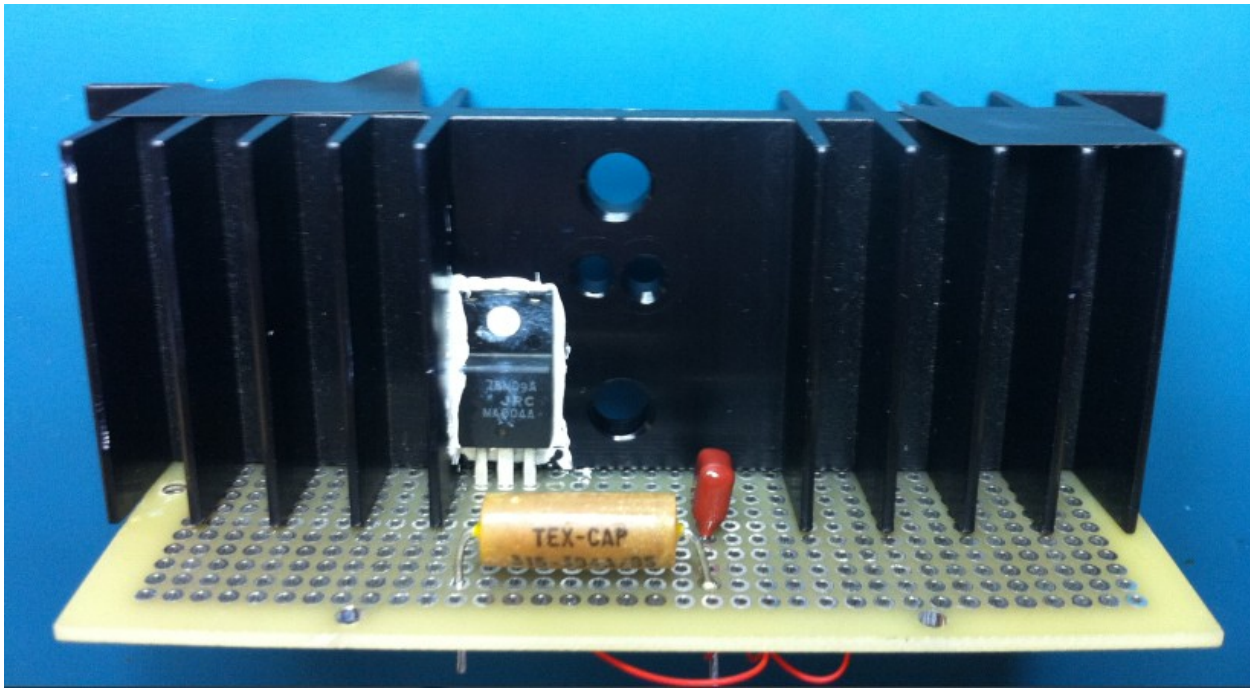


Figure 8-2: Built 9 Volt Power Regulation Circuit with Heat sink

9.0 WATCHDOG CIRCUIT

A Watchdog Circuit provides a failsafe to circuits and microcontrollers. For systems such as this Switch Matrix, that is required to be on for days or months without failure, a watchdog circuit is required. The Watchdog Circuit, in this instance, works by resetting the microcontroller if the microcontroller becomes unresponsive. The concept works that when the microcontroller is working properly, it regularly “pats” the watchdog circuit. If the microcontroller fails or the code becomes stuck, the microcontroller is unable to “pat” the circuit. After a set amount of time, the Watchdog Circuit will automatically reset the microcontroller to a desired state. See Figure 9-1 for a concept diagram of the Watchdog Circuit.

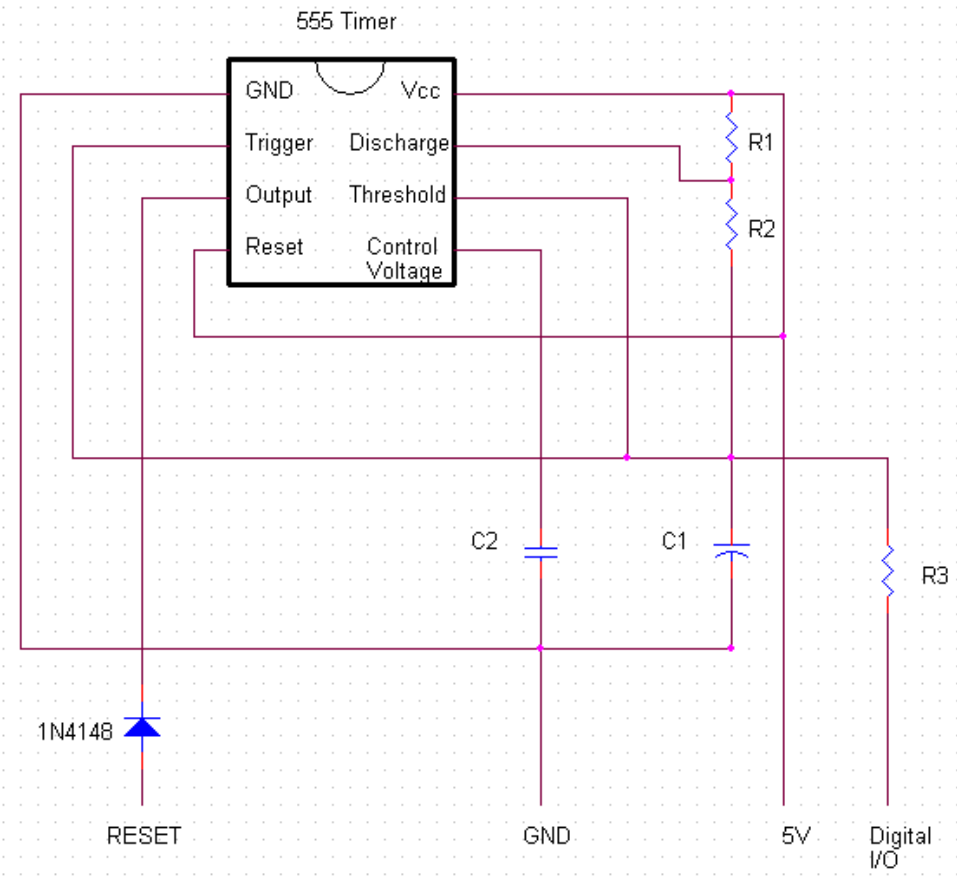


Figure 9-1: Watchdog Circuit Concept

For this Watchdog Circuit, a 555 timer is used. A 555 timer is an integrated circuit that is used in this application in astable operation mode. The output of the 555 timer is tied to the reset pin on the microcontroller. A capacitor (C1) begins to charge through two resistors (R1 and R2) from the microcontroller 5 volt pin. Once the capacitor reaches 3.33 Volts which is 2/3 of the supply voltage, the 555 timer pulls down the output pin and discharges the capacitor through resistor R2. While the output pin is pulled low, this resets the microcontroller. Once the capacitor discharges to 1/3 of the supply voltage, the output pin is returned to a high position and begins the charge cycle over again.

To “pat” the Watchdog Circuit, the microcontroller pulls down a digital output pin to sink current from the capacitor through a resistor (R3) and prevents the capacitor to charge to 2/3 of the supply voltage and reset the microcontroller. The length of time it takes to charge and discharge the capacitor is determined by the size of capacitor (C1) and both resistors (R1 and R2). For this project, the Watchdog Circuit was sized to reset the microcontroller after 15 seconds without a “pat” with a 100ms pulse to the reset pin. 15 seconds was choose because it was noted that the microcontroller took roughly 8 seconds to fully boot up before the watchdog programming module began. 15 seconds gave a factor of safety without having the microcontroller be unresponsive for too long. The reset pulse was chosen to be 100ms to allow for the microcontroller to fully reset. Below are the required Watchdog timing equations.

$$T_{CHARGE} = 0.67 * (R1 + R2) * C1 \qquad (9.1)$$

$$T_{DISCHARGE} = 0.67 * R2 * C1 \qquad (9.2)$$

A 100µF capacitor was used for capacitor C1. The following standard resistor sizes were selected to as closely match the targeted 15 second T_{CHARGE} and the 100ms $T_{DISCHARGE}$; where R1 is 200k Ω and R2 is 1.3k Ω. Solving equations 9.1 and 9.2 give a T_{CHARGE} of 13.5 seconds and $T_{DISCHARGE}$ of 90ms.

In addition to the components previously defined, the following are components added to protect the microcontroller from voltage fluctuations and current spikes. For C2, a 10nF capacitor is added between control voltage 555 timer pin and ground to reduce noise to prevent false triggers of the reset pin. A 1N4148 Diode is in line from the reset pin to the 555 timer output pin to protect the reset pin of the microcontroller. A 560 Ω resistors is used for R3 to sink the current from the capacitor while the microcontroller

“pats” or discharges the Watchdog Circuit. The below figure is the Watchdog Circuit built in the circuit board.

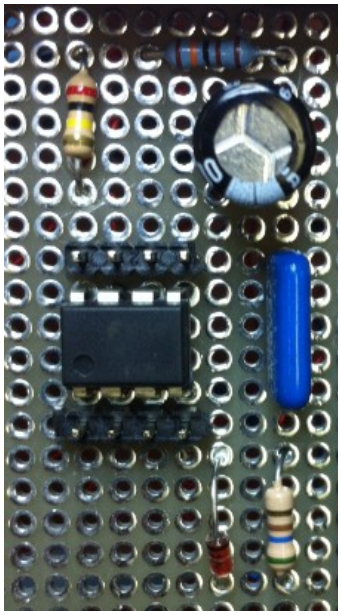


Figure 9-2: Watchdog Circuit

See Appendix G for a full schematic of the Watchdog Circuit design.

10.0 SWITCH MATRIX FEATURES

10.1 EMERGENCY SAFING

One of the electrical requirements for the Switch Matrix is to provide the user an emergency safe button. The concept is that in case of an issue or if the user wishes to stop all RF transmissions, a button on the Switch Matrix can be pressed.

To do this, a normally closed switch button is used. The button is in line between the +5 volts from the microcontroller to an input pin on the microcontroller. In addition, a 10k Ω pull down resistor is installed from the input pin to ground. When the button is not pressed, the input pin reads the +5 volts. However, when the button is pressed, the path to +5 volts is broken and the input pin is pulled low via the pull down resistor. The wiring of the emergency safe button can be seen in Appendix J.

When the microcontroller reads the input pin as pulled low, it directs the program to the EmergencySafeCode module in the code. In this module, the code displays that the Switch Matrix is now safing via the LCD display. Then each switch in the matrix is returned to the open position and prevents any RF signals to pass through the switches. Each switch is pulsed individually in order due to the lack of current to pulse all switches open simultaneously. To prevent the transistors from overheating, a one second delay is added to allow time for the system to cool down in case the operator holds the button down for prolong periods of time. At the end of the safing sequence, the LCD displays that the system is now safe.

The below figure is an image of the LCD display prior to the emergency safe button being pressed. The button is located on the bottom right of the image.

Figure 10.1-1: Emergency Safe Before

The below figure is an image of the LCD display after the emergency safe button has been pressed. Note that now the switch positions now read “0’ which is the open position for the switches.

Figure 10.1-2: Emergency Safe After

10.2 EXTERNAL RESET BUTTON

An external reset button is provided to the user in the event the operator wishes to reset the microcontroller in the Switch Matrix. The ATmega2560

microcontroller is equipped with a reset button located on the microcontroller. However this is not reachable to the operator without disassembling the unit. Instead, an external normally open switch button is used. The microcontroller has a reset pin that if pulled to ground, will reset the microcontroller. Therefore the button is wired inline between the reset pin and ground. The button is placed in the back of the switch matrix near the power switch. The button is mounted to the unit and can be accessed via a small hole in the unit.

10.3 SWITCH ACTIVATION COUNTER

The electrical mechanical switches that are used in this Switch Matrix are rated to two million actuations per position. While this is a large number, the Switch Matrix is expected to be used for spacecraft testing for many years. Therefore a switch activation counter is required to keep track of the number of activations per switch position commanded for each switch.

For the microcontroller to retain the number of switch actuations between power cycles, the number must be written to the EEPROM of the microcontroller. The ATmega2560 microcontroller has 4 KB of memory for EEPROM. Nominally, when writing to the EEPROM, you can only save a number from 0 to 255. Since this problem requires writing numbers up to at least two million, this is insufficient. To bypass this limitation, a library titled EEPROMAnything was implemented which increases the amount of bytes can be used to generate a number above the standard 1 byte which only allows a number from 0 to 255. The library automatically reads and writes to the required number of bytes. For this project, each switch positions was allocated 5 bytes of EEPROM memory, which means each counter can count to over a trillion. These parameters are more than sufficient for this feasibility study, knowing that the read / write capabilities of the EEPROM memory is less. See Appendix H for the EEPROMAnything library.

Each time a switch is commanded to a particular position, the program will read the activation counter for the switch position, add one and write the new value back to EEPROM memory. To access the counter telemetry, see Section 11 for commands and telemetry for the switch activation counter.

11.0 PROGRAMMING DEVELOPMENT

11.1 STANDARD COMMANDS FOR PROGRAMMABLE INSTRUMENTS

The Switch Matrix is commanded using the Standard Commands for Programmable Instruments (SCPI) protocol. SCPI was developed in 1990 and is the standard syntax for all test equipment and instruments for commanding. With a standard set of commands, it is easy to interface many different instruments to each other, and the Switch Matrix is no exception. For this project, SCPI commands are broken into two section; commands to actuate switches and commands to receive telemetry. The below two figures are a list of all SCPI commands that are available to the Switch Matrix and their function.

Commands for Switch Actuations	
SCPI Command	Output
S(1,1)	Activate Switch 1 to Position 1
S(1,2)	Activate Switch 1 to Position 2
S(1,3)	Activate Switch 1 to Position 3
S(1,4)	Activate Switch 1 to Position 4
S(1,5)	Activate Switch 1 to Position 5
S(1,6)	Activate Switch 1 to Position 6
S(1,7)	Return Switch 1 to OPEN Position
S(2,1)	Activate Switch 2 to Position 1
S(2,2)	Activate Switch 2 to Position 2
S(2,3)	Activate Switch 2 to Position 3
S(2,4)	Activate Switch 2 to Position 4
S(2,5)	Activate Switch 2 to Position 5
S(2,6)	Activate Switch 2 to Position 6
S(2,7)	Return Switch 2 to OPEN Position

Commands for Switch Actuations	
SCPI Command	Output
S(3,1)	Activate Switch 3 to Position 1
S(3,2)	Activate Switch 3 to Position 2
S(3,3)	Activate Switch 3 to Position 3
S(3,4)	Activate Switch 3 to Position 4
S(3,5)	Activate Switch 3 to Position 5
S(3,6)	Activate Switch 3 to Position 6
S(3,7)	Return Switch 3 to OPEN Position
S(4,1)	Activate Switch 4 to Position 1
S(4,2)	Activate Switch 4 to Position 2
S(4,3)	Activate Switch 4 to Position 3
S(4,4)	Activate Switch 4 to Position 4
S(4,5)	Activate Switch 4 to Position 5
S(4,6)	Activate Switch 4 to Position 6
S(4,7)	Return Switch 4 to OPEN Position

Figure 11.1-1: Commands for Switch Actuations

Commands for Switch Matrix Telemetry		
SCPI Command	Description	Output
*IDN?	Displays Identification and version of Switch Matrix	SSL,LCU/Switch Matrix,Prototype,A1.00
S1?	Displays Switch 1 telemetry Position	Integer from 0 to 6. Where 0 is OPEN position
S2?	Displays Switch 2 telemetry Position	Integer from 0 to 6. Where 0 is OPEN position
S3?	Displays Switch 3 telemetry Position	Integer from 0 to 6. Where 0 is OPEN position
S4?	Displays Switch 4 telemetry Position	Integer from 0 to 6. Where 0 is OPEN position
T1?	Displays Activation Counter of Switch 1	Six integers, delimited by a comma (,) where each integer is the corresponding position's counter
T2?	Displays Activation Counter of Switch 2	Six integers, delimited by a comma (,) where each integer is the corresponding position's counter
T3?	Displays Activation Counter of Switch 3	Six integers, delimited by a comma (,) where each integer is the corresponding position's counter
T4?	Displays Activation Counter of Switch 4	Six integers, delimited by a comma (,) where each integer is the corresponding position's counter

Figure 11.1-2: Commands for Switch Matrix Telemetry

11.2 CODE LOGIC

The Sparkfun ATmega2560 Microcontroller is programmed in the Arduino Programming Language. The Arduino Programming Language is based off of C++ and can be expanded through the standard C++ libraries. The structure of an Arduino Code is broken into three sections: Initialization, Setup, and Loop.

Initialization

During the initialization phase of the code, the building blocks of the Switch Matrix program are defined. First the libraries used are defined. This program calls the following libraries:

- Ethernet.h & SPI.h are called to initialize the Arduino Ethernet shield and provide Ethernet access
- SoftwareSerial.h is used to drive the LCD display as defined in Section 6.2
- EEPROM.h and EEPROMAnything.h is used to read and write to the EEPROM memory as defined in Section 10.3.

In addition, the initialization phase defines and maps the microcontroller pins to variable names, defines the variables used, and configures the Ethernet connection.

Setup

During the setup section, each microcontroller pin is defined as an input or an output pin. The Ethernet client begins and connects to the router, and the LCD display is initialized.

Loop

The loop section of the code is where the Switch Matrix is controlled. The loop section is broken down into four modules that loop indefinitely. Once each module has completed its designed task, the program will move to the next module. Once the last module has completed its task, the first module will begin again. The four modules are as follows:

- CheckforClient
- TelemetryCode

- EmergencySafeCode
- Watchdog

The CheckforClient module checks to see if the user has input any commands to the Switch Matrix. If no commands are seen, the module ends and moves to the next module. If the module sees a command from the user, the command is broken down into the individual characters and enters a state machine. If the command is equal to one of the commands as defined in Figures 11.1-1 and 11.1-2 it enters a predetermined function and performs its action. If the input from the user does not match any defined command, it ignores the input and clears the client. The following are the list functions and their output:

- SCPI_IDENTITY: This function displays the identification and version of the Switch Matrix to the user
- SCPI_Telemetry: This function displays the desired switch position telemetry to the user
- EEPROMCounterRead: This function reads the position counters of the desired switch and displays each position counter values delimited by a comma.
- ExecuteCommand: This function drives the desired switch to the desired position. In addition, it increments the EEPROM counter for the respective position.

The TelemetryCode module reads all of the current switch positions and displays the positions to the LCD display. This provides a real-time telemetry positions to the LCD display.

The EmergencySafeCode module checks to see if the emergency safe button is pressed. If it button is not pressed, the program continues to the next module. If the button is pressed, it actives all of the switches and

returns them to a safe configuration. See Section 10.1 for more information on the emergency safing.

The Watchdog module has two functions. This first is to pull down the watchdog pin to discharge the capacitor in the watchdog circuit. This “pats” the watchdog circuit. Once the “pat” is complete, it returns the watchdog pin to a high position to allow the capacitor to continue charging. The second function of the Watchdog module is to display a heartbeat on the LCD display. This gives the user a visual cue that the Switch Matrix is functional. The heartbeat blinks at rate of about 1 beat per second. This rate is arbitrary, and is only show alive status.

The figure below is a flowchart of the Switch Matrix program. See Appendix I for the entire code used for the Switch Matrix.

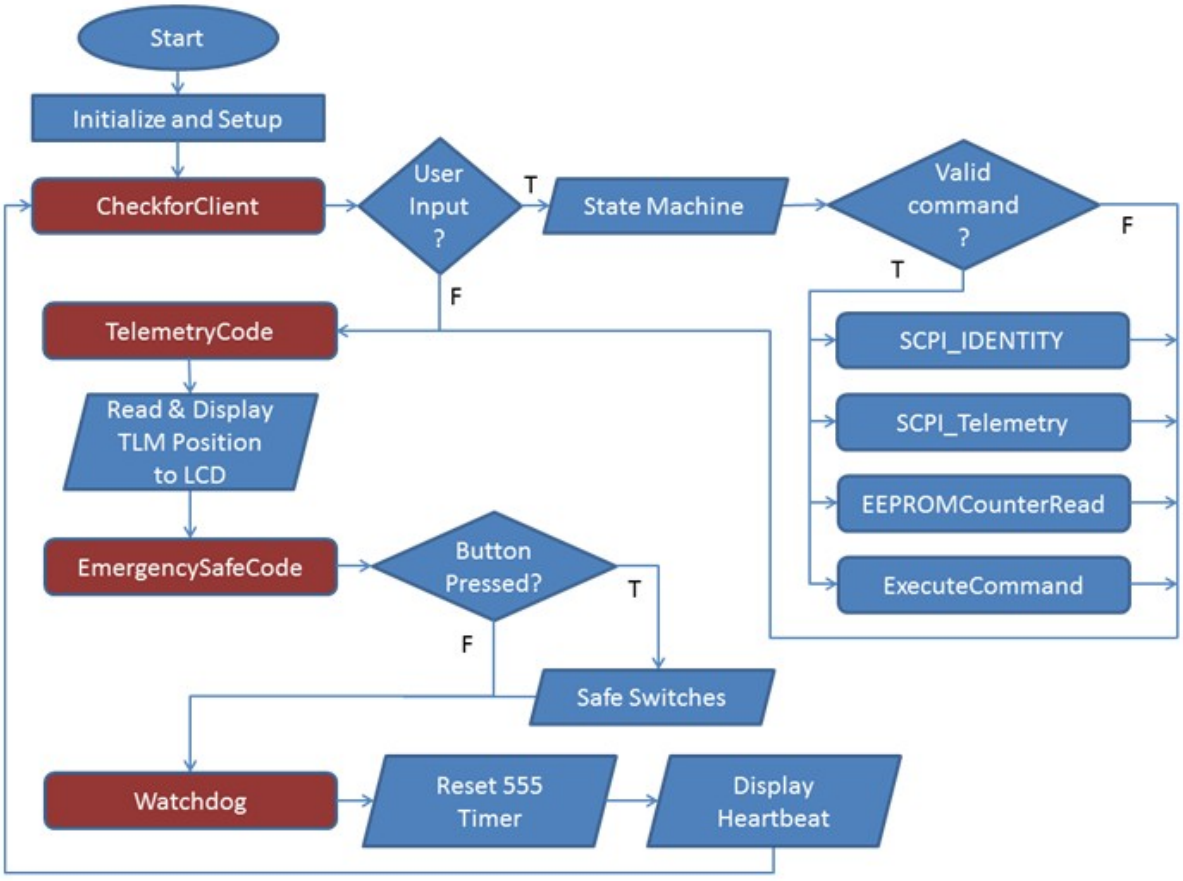


Figure 11.2-1: Switch Matrix Program Flowchart

11.3 GENERAL USER INTERFACE

To provide the user with a clean and professional interface and to reduce the chance of incorrect commands being sent, a General User Interface (GUI) was created. The GUI was created by Paul Trainer at SSL. The GUI allows the operator to select which position they desire via a drop down menu. Once the switch position is selected, the operator presses the “Execute” button. The GUI sends the desired switch actuation command to the Switch Matrix code and pulls the switch position telemetry and the actuation counter for the commanded switch from the code and displays it on the GUI. Once the command has been executed, the “Execute” button will turn from green to red to give the user a visual cue that the command has been sent. In addition, an Identity Test button is offered to the user which pulls the identity and version of the Switch Matrix in use. See Figure 11.3-1 below for a screenshot of the Switch Matrix GUI. Figure 11.3-2 shows a block diagram to represent the switch positions called out per Figure 11.3-1.

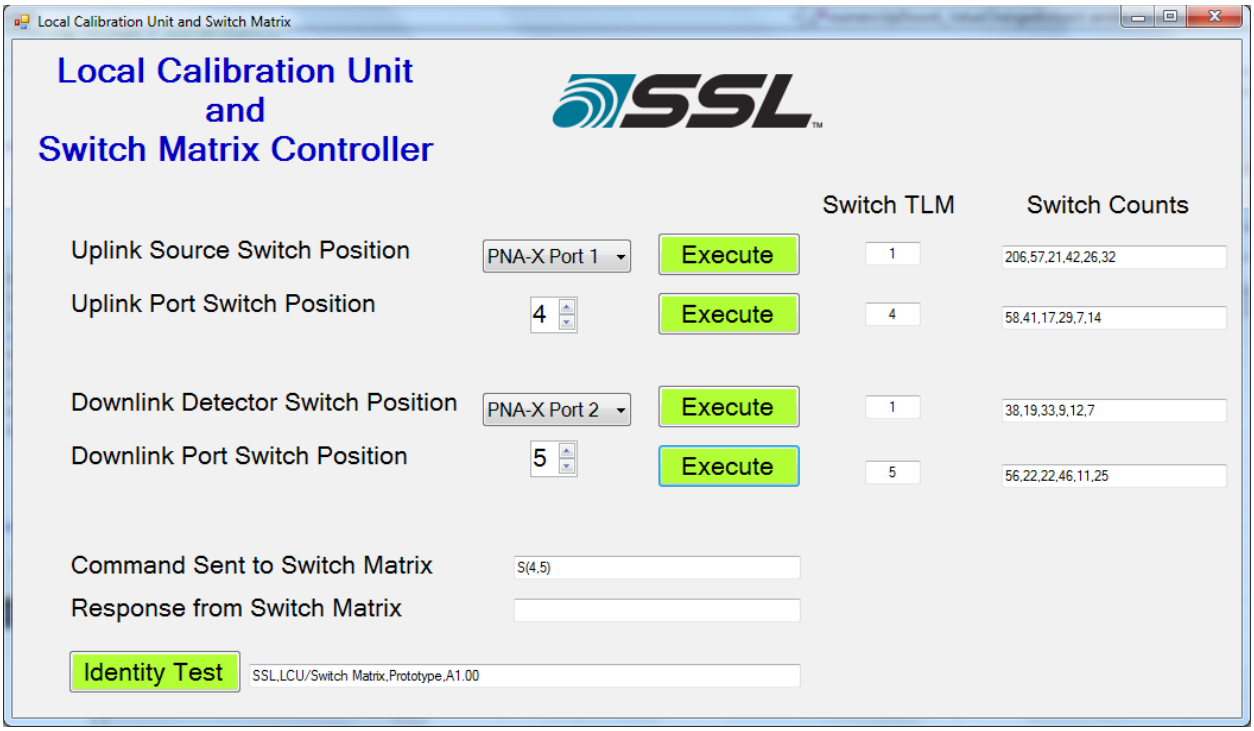


Figure 11.3-1: Switch Matrix GUI

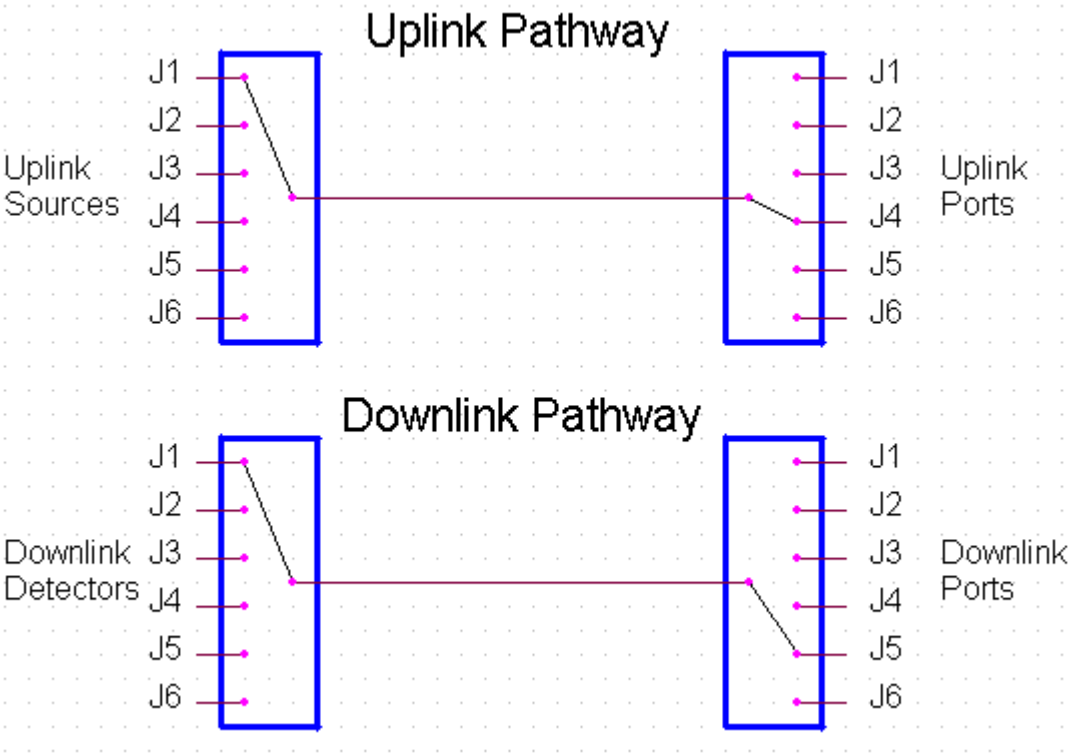


Figure 11.3-2: Switch Position Block Diagram

12.0 MECHANICAL INTEGRATION

At this point, all electrical design has been completed. This section covers the building and mechanical integration of the Switch Matrix

12.1 SWITCH MATRIX WIRING

All of the previously designed circuits must be built and wired. To do this, a prototype board with 0.1” spacing is used to hold all of the components. Each component is soldered into place to prevent components from loosening. Then each wire is installed by hand via the wire wrap method. Wire wrapping is used because it provides easier troubleshooting if an issue arises during the build. In total, over 220 wires are installed on the circuit boards. See Appendix J for a schematic of all wiring used for all circuits for this project. The below figure is an image taken of the bottom of the prototype board which shows some of the wires installed.

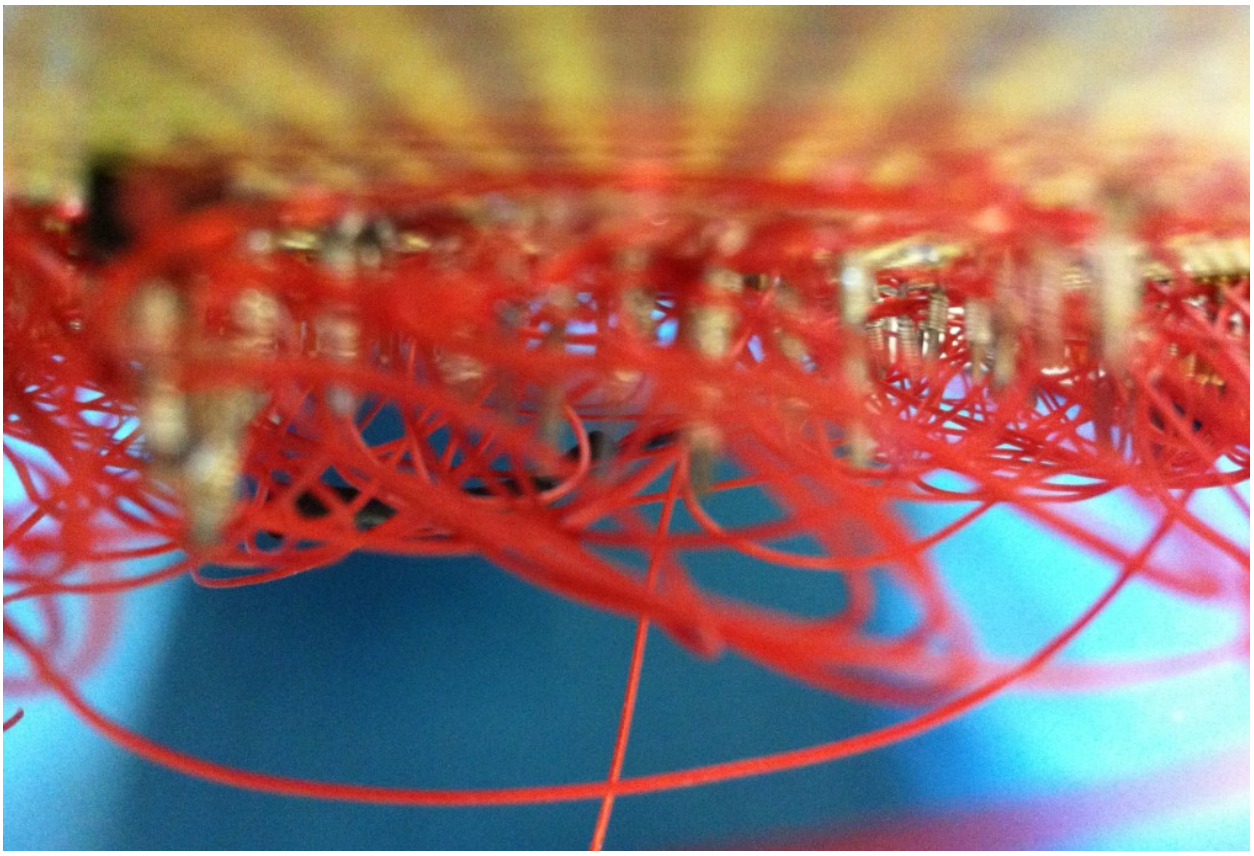


Figure 12.1-1: Switch Matrix Wiring Image

12.2 UNIT DESIGN

The unit that houses the electronics and switches is a 2U 19” chassis that is compatible to mount to a standard rack. To allow for component mounting, the front plate and back plate were machined and the front plate

was painted. See Appendix K for the CAD file of the Front Plate and Appendix L for the CAD file of the Back Plate.

The front plate was machined with openings for two of the switches, the LCD display and mounting holes, emergency safe button, and two coax cable ports. Each coax cable port provides an interface from the switch's common RF pathway port to the inside of the Switch Matrix unit.

The back plate was machined with openings for two of the switches, the Ethernet Port, the power line filter, the external reset button, and three coax cable ports. Two of the coax cable ports provide an interface from the inside of the Switch Matrix unit to the switch's common RF pathway port. The third is to provide an RF test device interface.

The switches and coax cable ports were machined in line and equal distance from each other. This allows for equal cable distances between the switches. This improves RF performance and repeatability.

12.3 SYSTEM INTEGRATION

To fully integrate the system, the electrical components were mounted to the unit. The circuit boards, the power supply, and the power regulator were mounted to the bottom of the unit. The switches were held down by U-brackets which were also mounted to the bottom of the unit. The below figure is an image of the Switch Matrix with all components fully integrated.

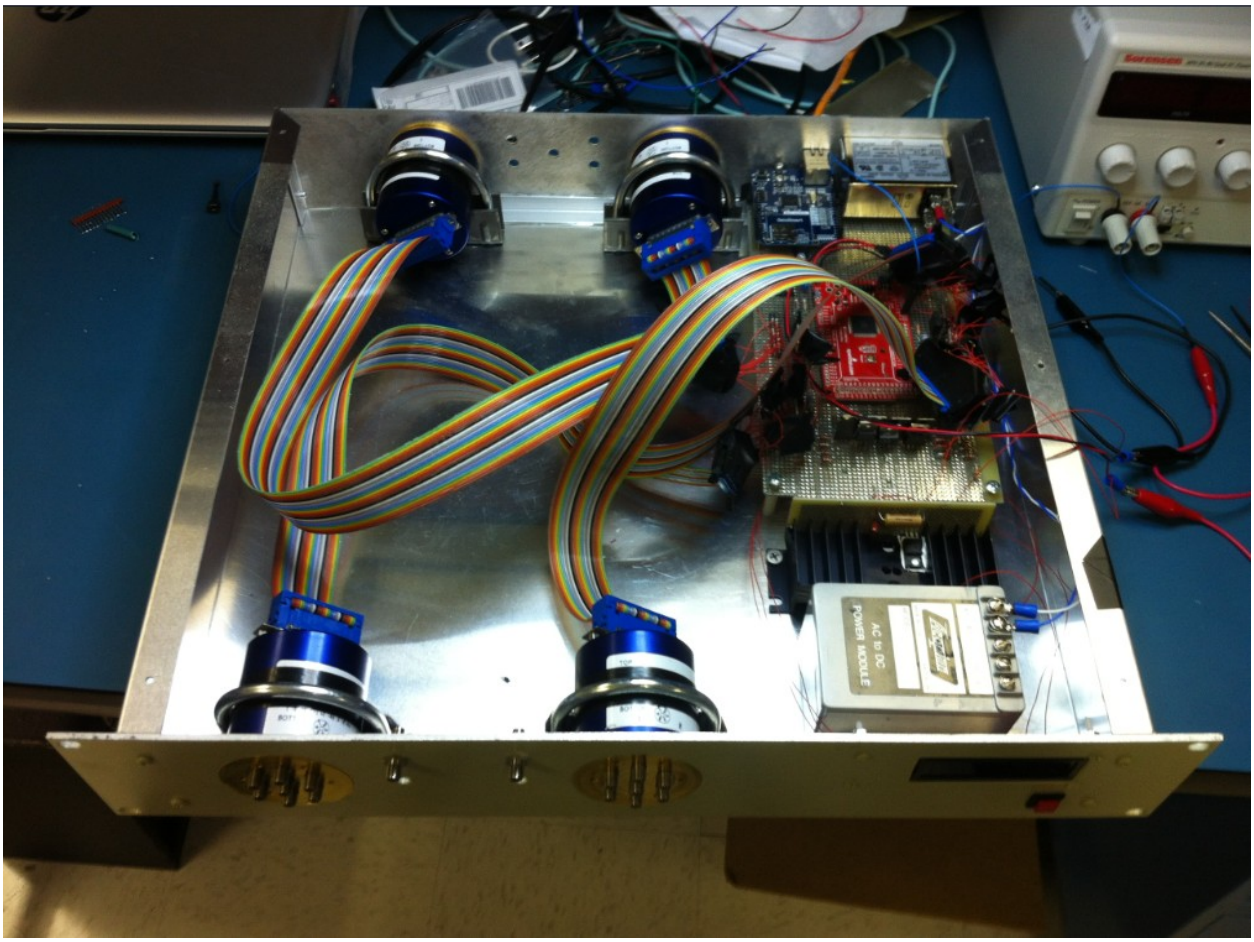


Figure 12.3-1: Switch Matrix Fully Integrated

13.0 CONCLUSION

In this project, a fully functional and reliable RF Switch Matrix was designed and integrated used COTS components. The Switch Matrix is able to direct RF signals up to 40GHz with nominal repeatability and low insertion loss. The Switch Matrix is programmed in a robust way and uses SCPI protocol which can be interfaced easily by different systems. The unit can be operational for months at a time due to the implementation of a watchdog circuit and emergency safing features.

This project successfully passed the feasibility study at SSL and will now set the precedent and be the archetype for all future SSL built RF Switch Matrices. Not only was this Switch Matrix built for a fraction of the price as a competitor's switch matrix, it also has the ability to be expanded with additional RF pathways and features.

See the figure below for the final fully built and integrated image of the SSL Radio Frequency Switch Matrix.

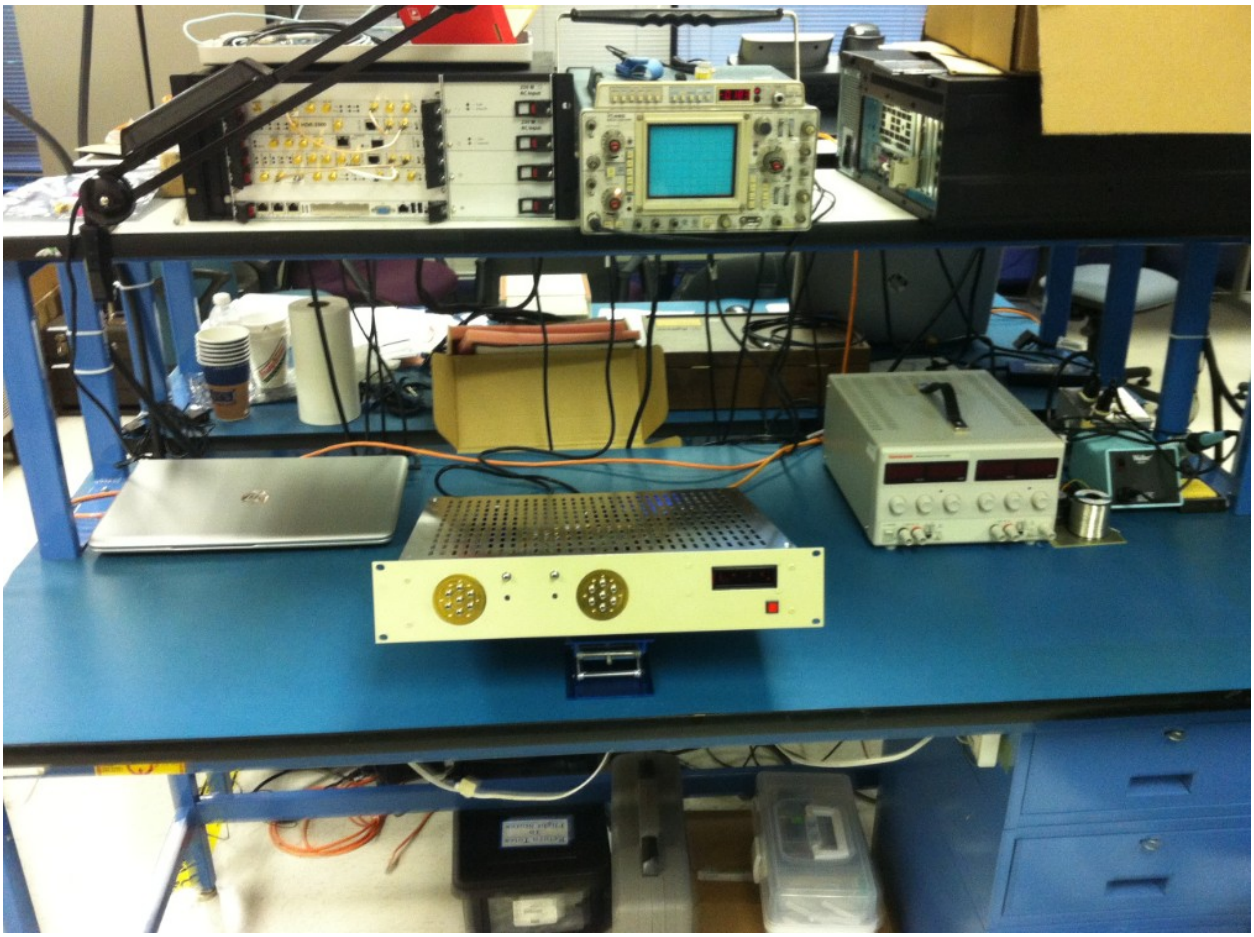


Figure 13-1: Finished Switch Matrix

REFERENCES

[1] Assal, F.T.; Gupta, R.; Betaharon, Khodadad; Zaghloul, A.I.; Apple, J., "A Wide-Band Satellite Microwave Switch Matrix for SS/TDMA Communications," *IEEE Journal on Selected Areas in Communications,* vol.1, no.1, pp.223-231, January 1983
doi: 10.1109/JSAC.1983.1145897

[2] Gupta, R.; Assal, F.; Hampsch, T., "A microwave switch matrix using MMICs for satellite applications," *Microwave Symposium Digest, 1990, IEEE MTT-S International*, vol.2, pp.885-888, 8-10 May 1990
doi: 10.1109/MWSYM.1990.99720

[3] U-yen, K.; Lu Dong; Kenney, J.S., "A low-loss high-reliability microwave switch matrix for smart antenna systems," *Microwave Symposium Digest, 2004 IEEE MTT-S International*, vol.2, pp.1125-1128, 6-11 June 2004
doi: 10.1109/MWSYM.2004.1339183

[4] Daneshmand, M.; Mansour, R.R., "C-type and R-type RF MEMS Switches for Redundancy Switch Matrix Applications," *Microwave Symposium Digest, 2006. IEEE MTT-S International*, pp.144-147, 11-16 June 2006
doi: 10.1109/MWSYM.2006.249415

[5] Voudouris, K.; Athanasopoulos, N.; Meir, A.; Manor, D.; Tsiakas, P.; Georgas, I.; Petropoulos, I.; Agapiou, G., "2x2 Switch Matrix for WiMAX Relay Station Applications," *Microwave and Wireless Components Letters, IEEE* , vol.21, no.8, pp.424-426, Aug. 2011
doi: 10.1109/LMWC.2011.2158532

[6] Kaleem, S.; Humbla, S.; Rentsch, S.; Trabert, J.; Stopel, D.; Muller, J.; Hein, M.A., "Compact Ka-band reconfigurable switch matrix with power failure redundancy," *2012 The 7th German Microwave Conference (GeMiC),* pp.1-4, 12-14 March 2012

[7] Chan, E.; Daneshmand, M.; Mansour, R.R.; Ramer, R., "Monolithic crossbar MEMS switch matrix," *Microwave Symposium Digest, 2008 IEEE MTT-S International*, pp.129-132, 15-20 June 2008
doi: 10.1109/MWSYM.2008.4633120

[8] Sinha, S.; Bansal, D.; Rangra, K.J., "RF MEMS compact T-type switch design for switch matrix applications in space telecommunication," *2012 International Conference on Advances in Engineering, Science and Management (ICAESM),* pp.130-135, 30-31 March 2012

[9] Siew, E.; King Yuk Chan; Ramer, R.; Dzurak, A., "Design of a RF NEMS switch matrix," *Antennas and Propagation (APSURSI), 2011 IEEE International Symposium,* pp.12-15, 3-8 July 2011
doi: 10.1109/APS.2011.5996369

[10] Agilent Technologies, “Agilent RF/Microwave Switching Solutions,” Agilent Technologies, USA, 2008, Print

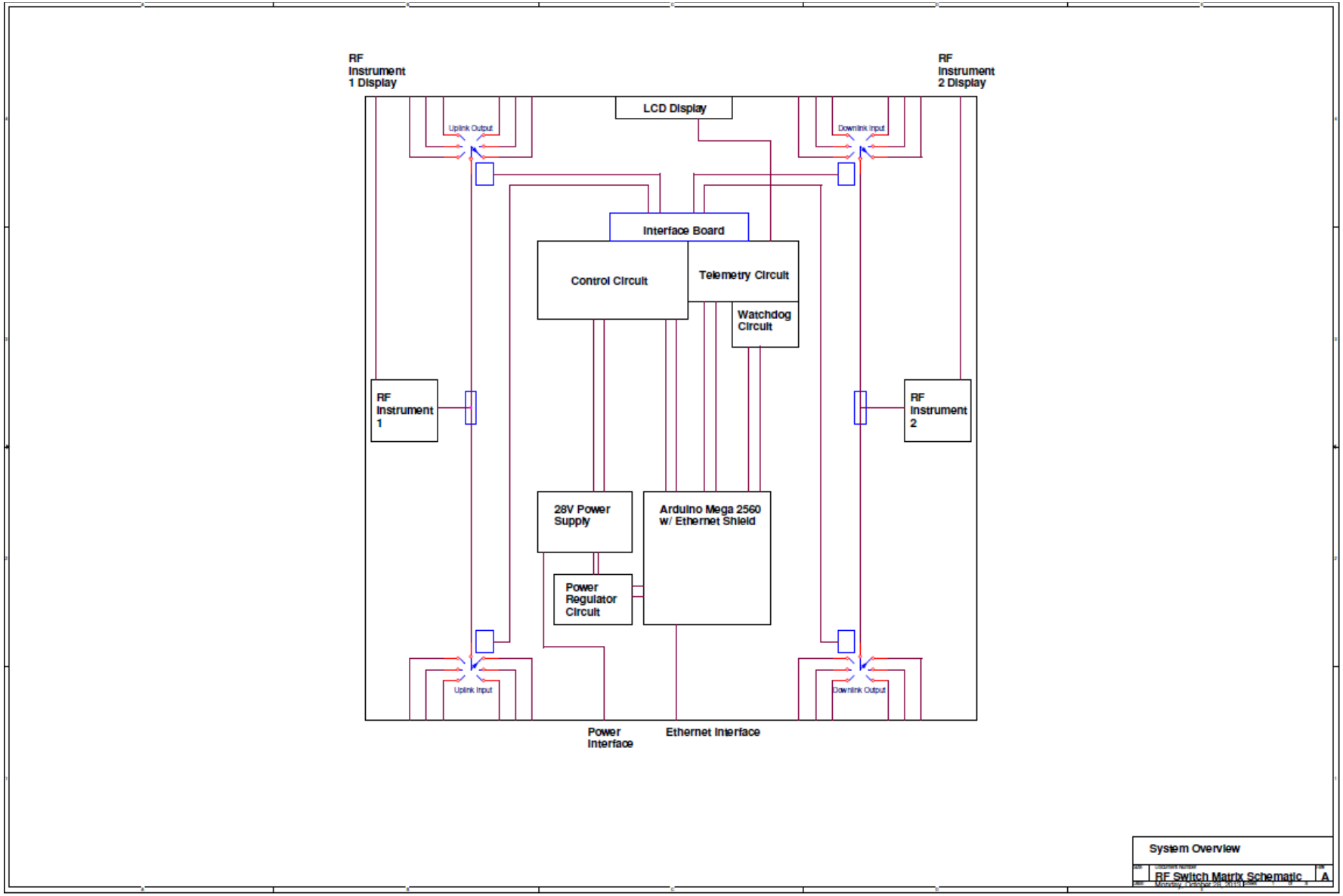
[11] Agilent Technologies, “Agilent's Payload RF Test Solution,” Agilent Technologies, USA, 2011, Print

[12] Agilent Technologies, “Agilent Custom Switch Matrices,” Agilent Technologies, USA, 2001, Print

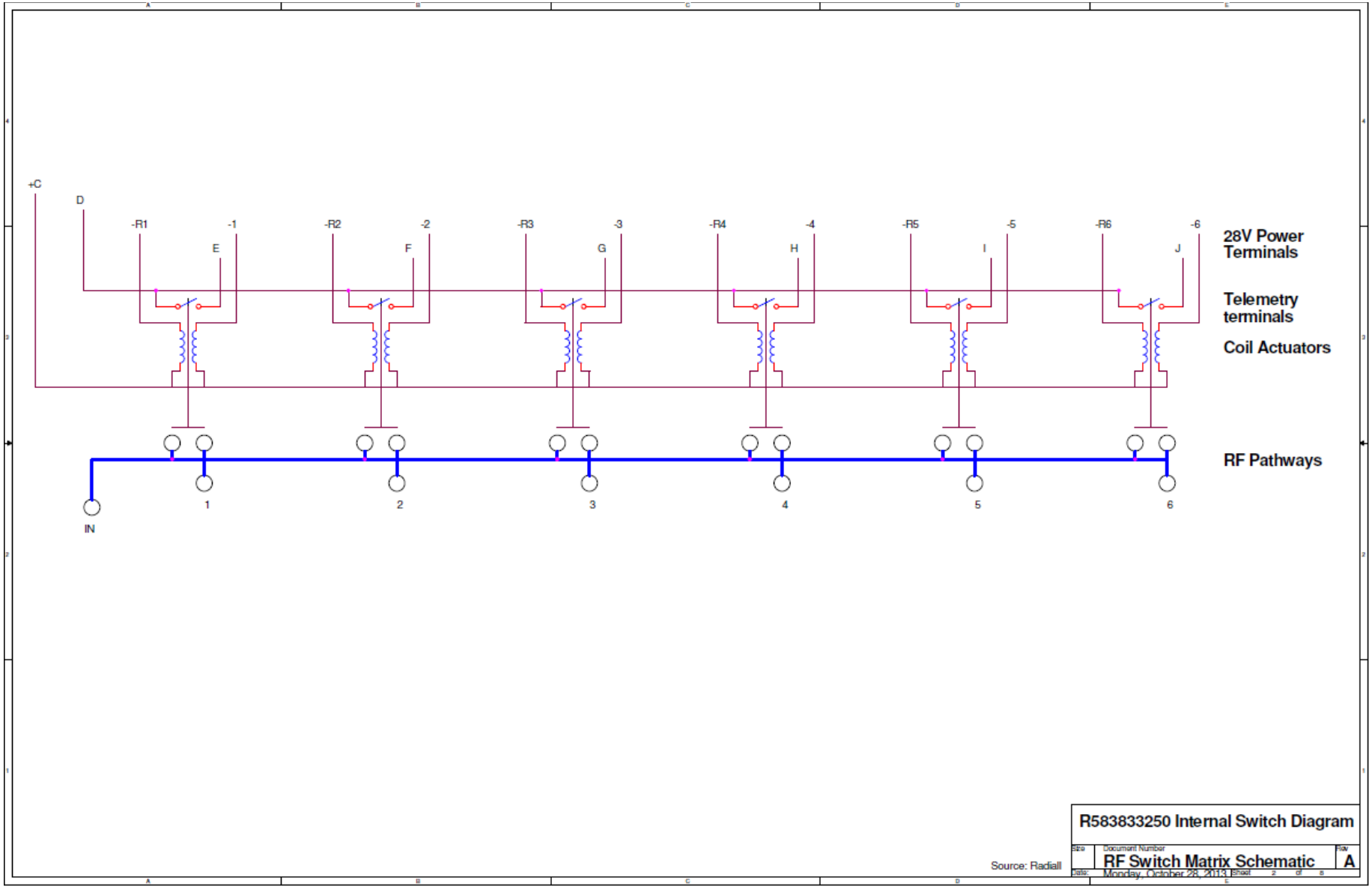
[13] Agilent Technologies, “Agilent Switch Matrix - Product Overview,” Agilent Technologies, USA, 2009, Print

APPENDICES

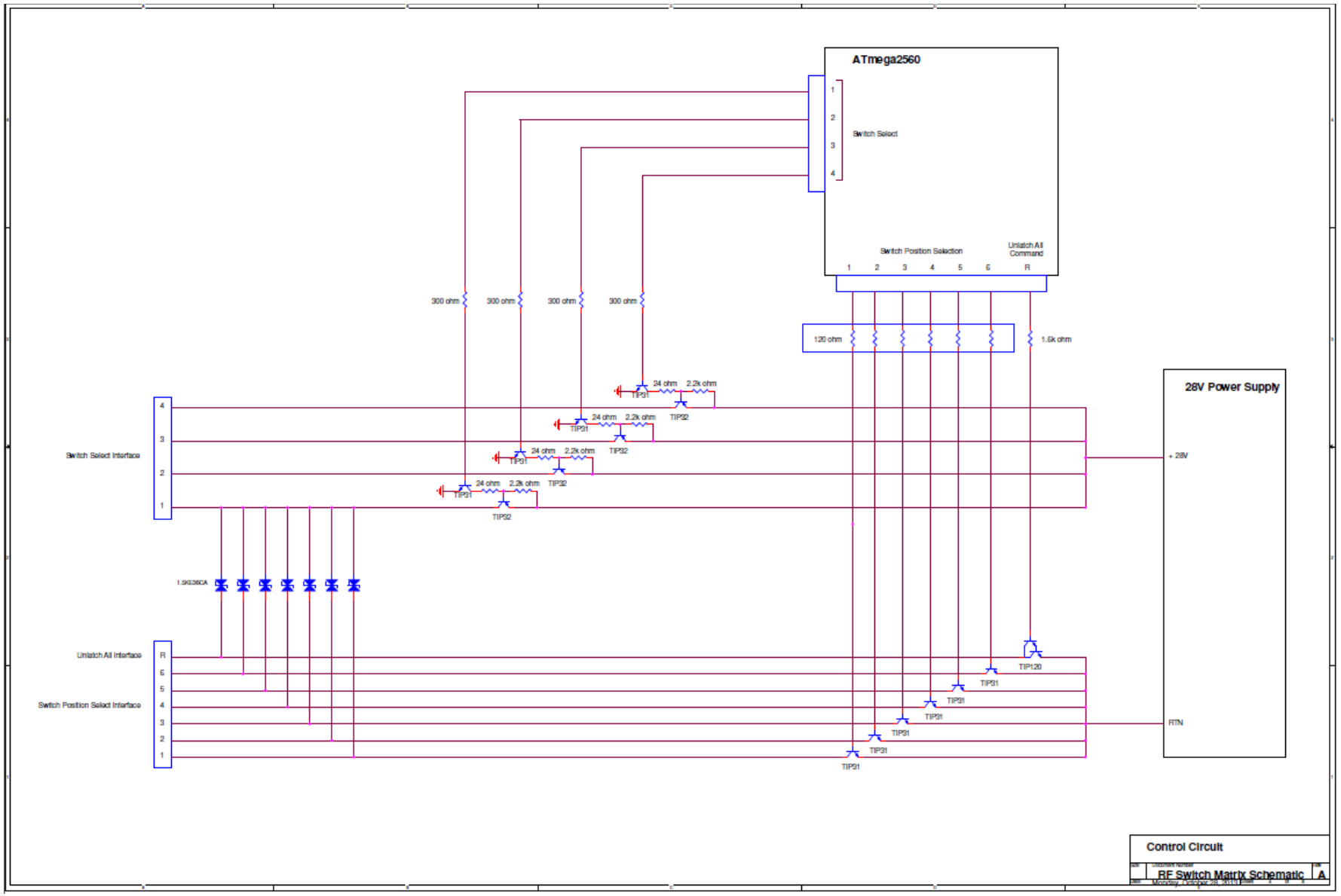
APPENDIX A: System Overview Drawing



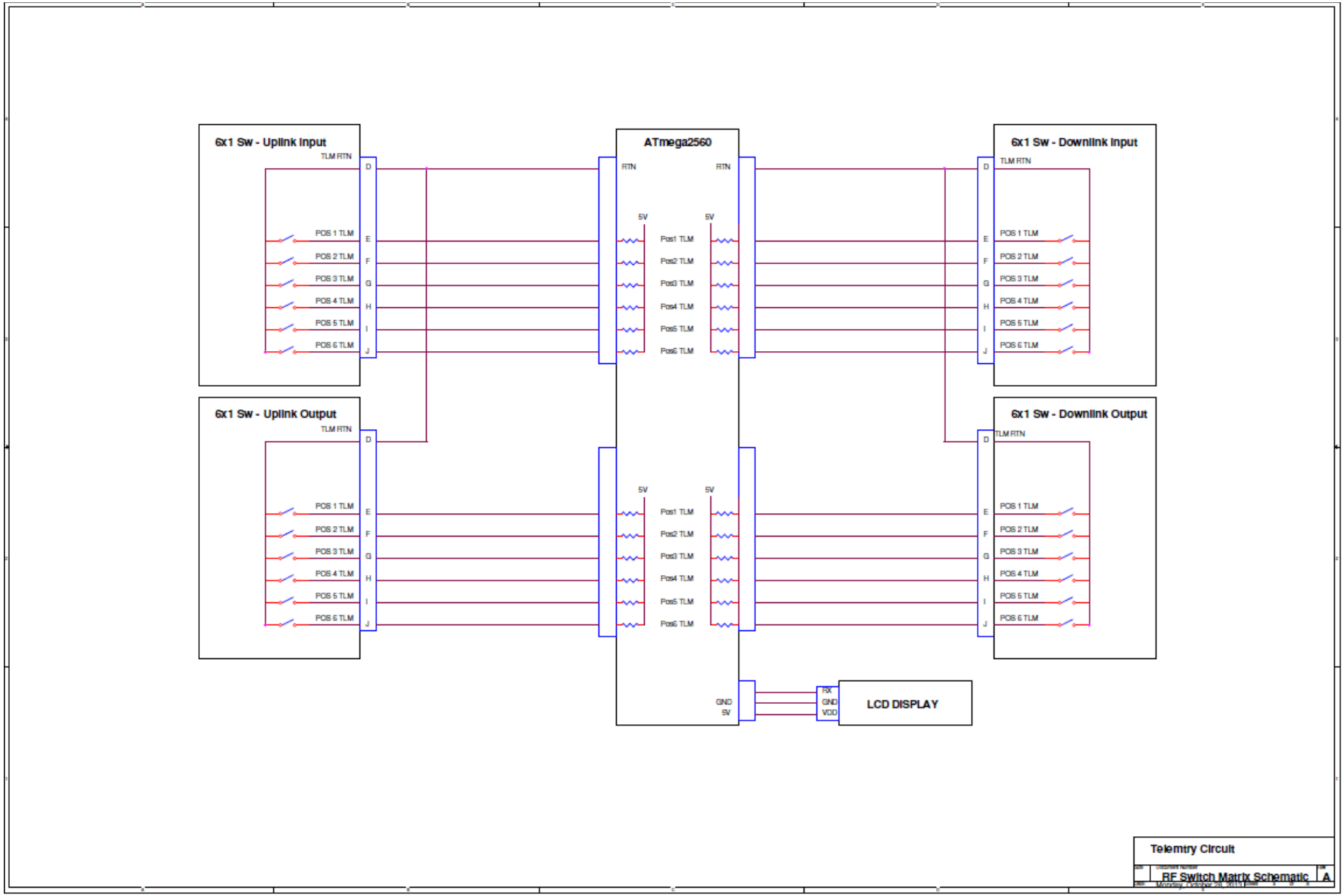
APPENDIX B: R583833250 Internal Switch Schematic



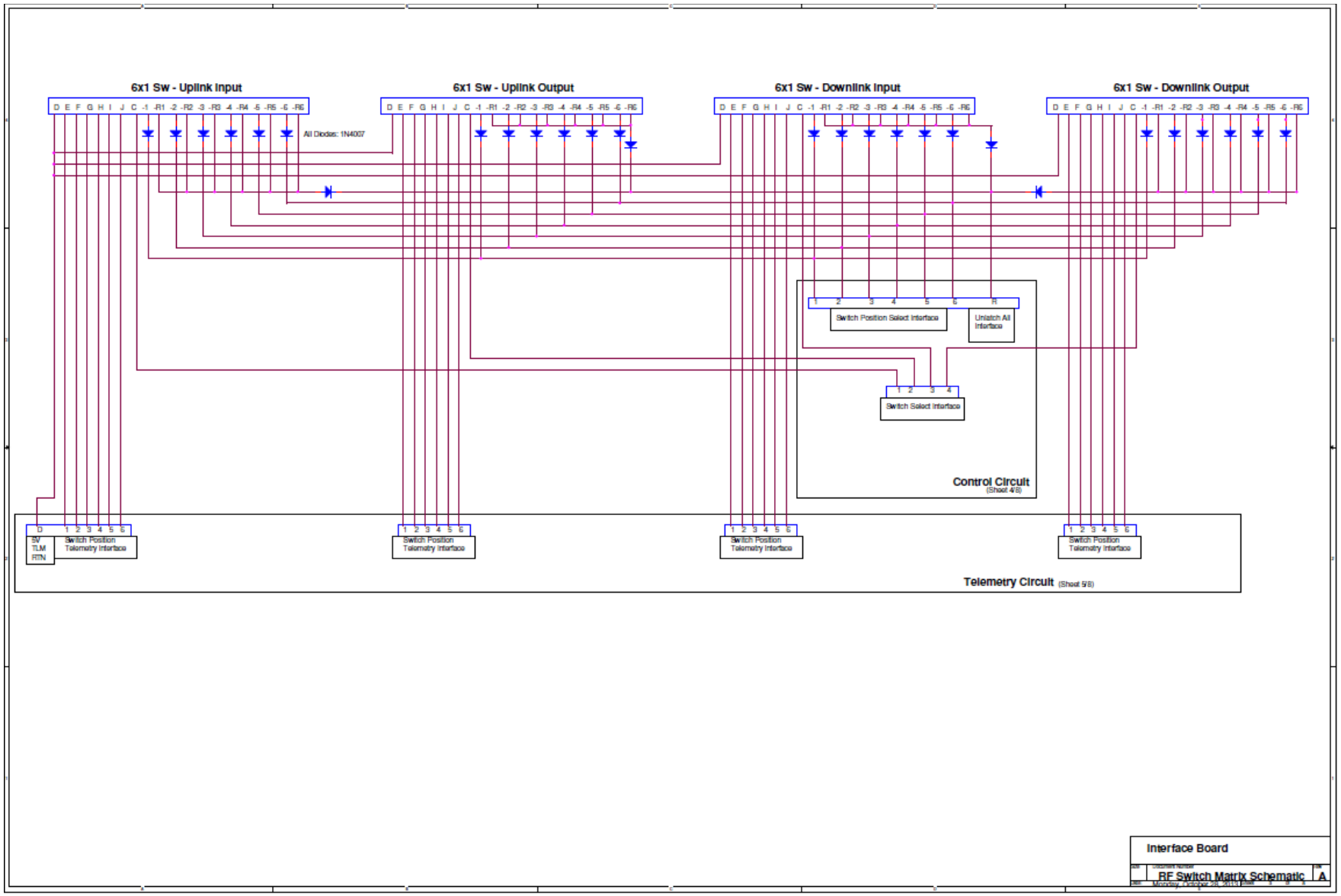
APPENDIX C: Switch Drive Matrix Schematic



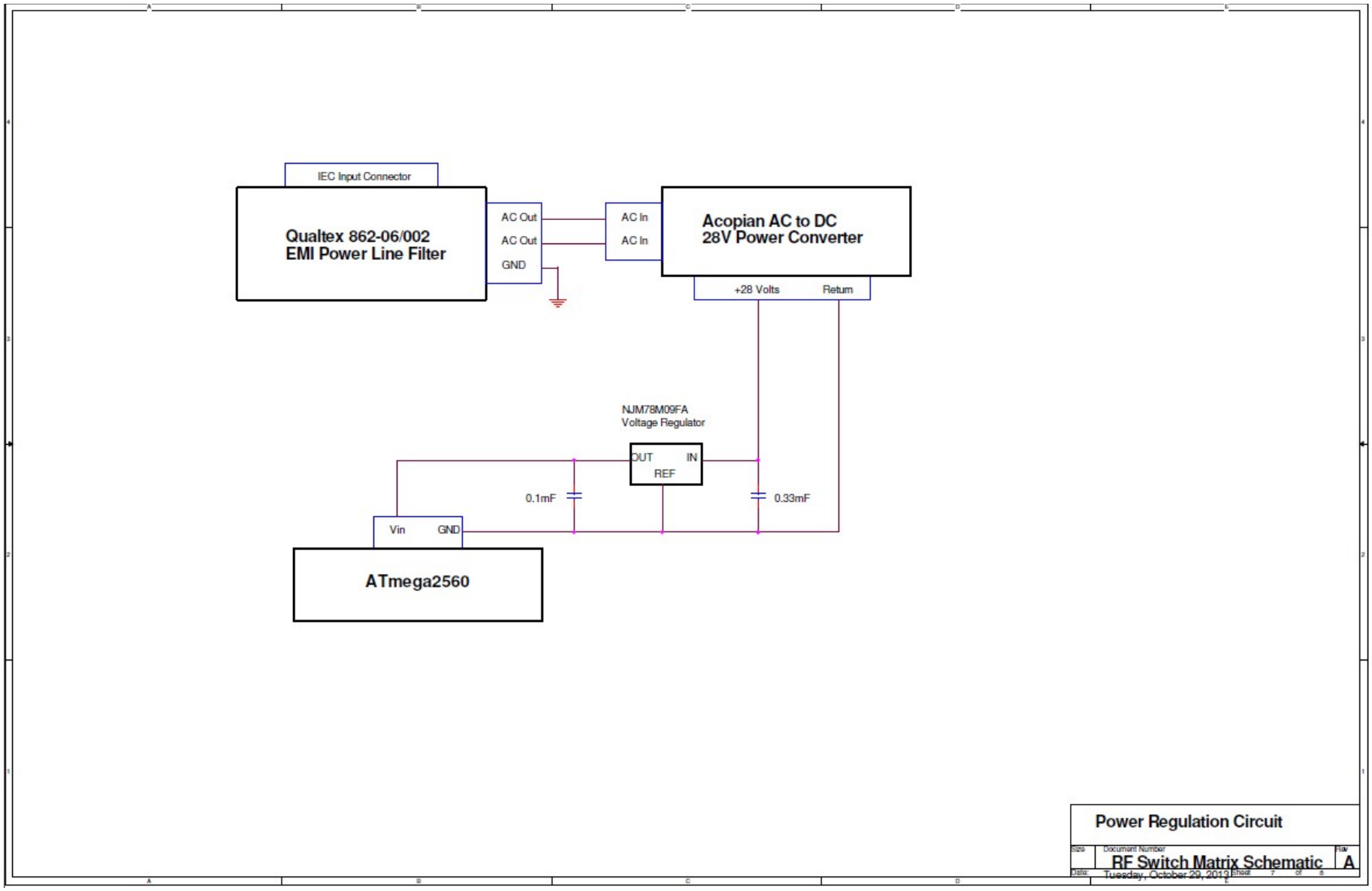
APPENDIX D: Telemetry Circuit Schematic



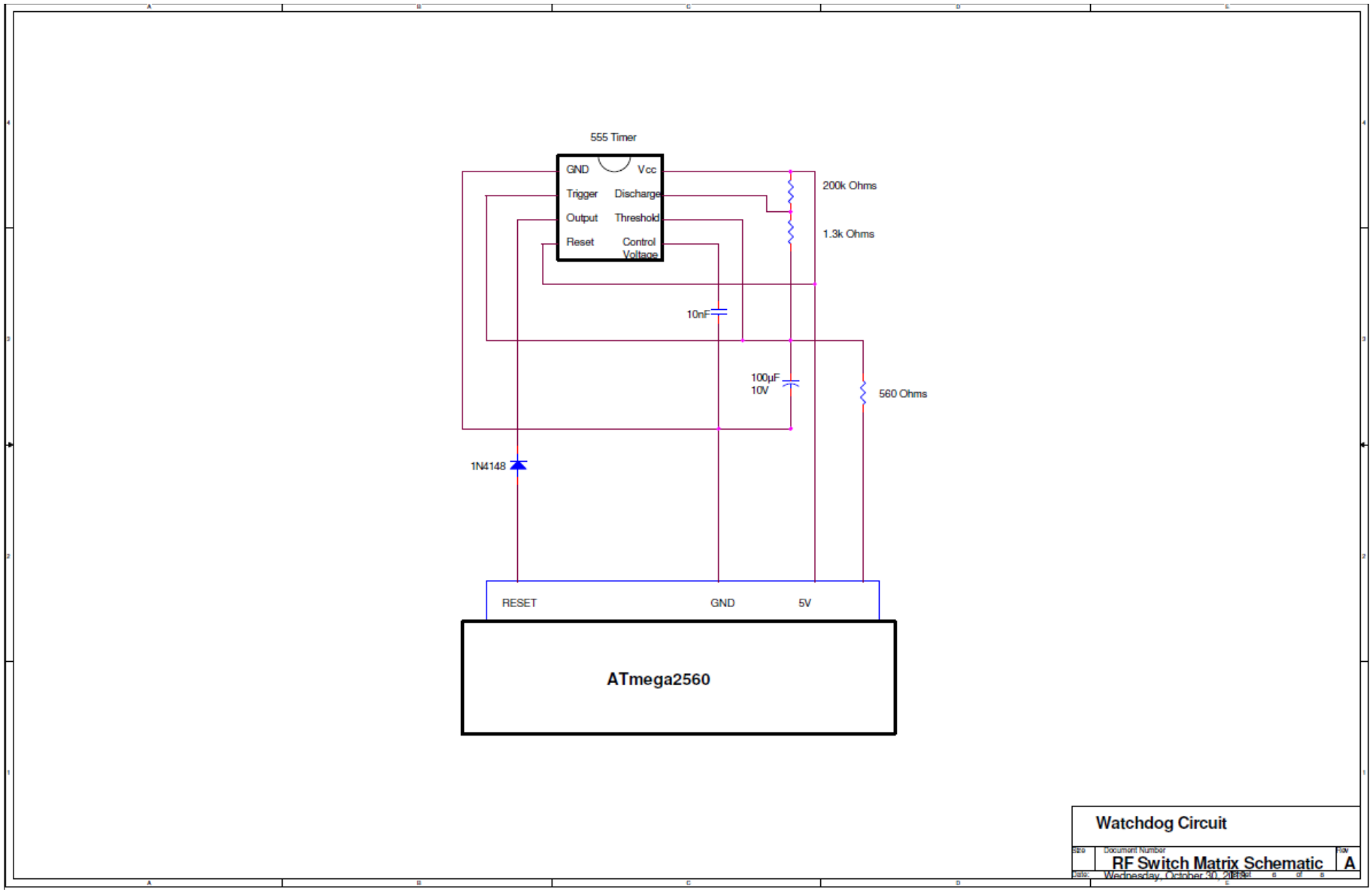
APPENDIX E: Interface Circuit Schematic



APPENDIX F: Power Regulation Circuit Schematic


IEC Input Connector
Qualtex 862-06/002
EMI Power Line Filter
AC Out
AC Out
GND
AC In
AC In
Acopian AC to DC
28V Power Converter
+28 Volts
Return
NJM78M09FA
Voltage Regulator
OUT
IN
REF
0.1mF
0.33mF
Vin
GND
ATmega2560
Power Regulation Circuit
Size
Document Number
RF Switch Matrix Schematic
Rev
A
Date:
Tuesday, October 29, 2013
Sheet 7 of 8

APPENDIX G: Watchdog Circuit Schematic


555 Timer
GND
Vcc
Trigger
Discharge
Output
Threshold
Reset
Control Voltage
200k Ohms
1.3k Ohms
10nF
100µF 10V
560 Ohms
1N4148
RESET
GND
5V
ATmega2560
Watchdog Circuit
Document Number
RF Switch Matrix Schematic
A
Wednesday, October 30, 2013

APPENDIX H: EEPROMAnything Library

```
#include <EEPROM.h>
#include <Arduino.h>  // for type definitions

template <class T> int EEPROM_writeAnything(int ee, const T& value)
{
   const byte* p = (const byte*)(const void*)&value;
   unsigned int i;
   for (i = 0; i < sizeof(value); i++)
        EEPROM.write(ee++, *p++);
   return i;
}

template <class T> int EEPROM_readAnything(int ee, T& value)
{
   byte* p = (byte*)(void*)&value;
   unsigned int i;
   for (i = 0; i < sizeof(value); i++)
        *p++ = EEPROM.read(ee++);
   return i;
}
```

APPENDIX I: Switch Matrix Program

```
//......LCU / SWITCH MATRIX CODE.........//
//......Author.........Zack Pirkl.....................//

//....CONTAINS THE FOLLOWING FEATURES:....//
//..........Controls 4 Switches.............................//
//..........Ethernet Commanding.....................//
//..........LCD Display.....................................//
//..........Watchdog (Heartbeat Monitor)..........//
//..........EEPROM Activation Counter.............//

//////////////////////////////////////////////////
//////////   COMMANDING DIRECTIONS    /////////
//////////////              PuTTY            /////////////////
//////////  IP Address: 192.168.0.77  //////////////
//////////////        Port 23 (Telnet)     ///////////////
////  S(a,b)  Where a = SwSelect, b = SwPos /////
//////////////////////////////////////////////////
//////////  Switch Position Telemetry  ////////////
////////////  Sa?  Where a = SwSelect  ////////////
//////////////////////////////////////////////////
//////  Activation Counter TLM Directions ///////
///////////  Ta?  Where a = SwSelect ////////////
//////////////////////////////////////////////////

//Define Libraries
#include <Ethernet.h>
#include <SPI.h>
#include <SoftwareSerial.h>
#include <EEPROM.h>
#include "EEPROMAnything.h"

//Define Switch Position Arduino Pins
#define Pos1 34
#define Pos2 35
#define Pos3 36
#define Pos4 37
#define Pos5 38
#define Pos6 39
#define ReturnPos 40
```

```
//Define Switch Selection Arduino Pins
#define Sw1 30
#define Sw2 31
#define Sw3 32
#define Sw4 33

//Define Switch Telemetry Arduino Pins
#define Sw1Pos1TLM 2
#define Sw1Pos2TLM 3
#define Sw1Pos3TLM 4
#define Sw1Pos4TLM 5
#define Sw1Pos5TLM 6
#define Sw1Pos6TLM 7

#define Sw2Pos1TLM 22
#define Sw2Pos2TLM 23
#define Sw2Pos3TLM 24
#define Sw2Pos4TLM 25
#define Sw2Pos5TLM 26
#define Sw2Pos6TLM 27

#define Sw3Pos1TLM A0
#define Sw3Pos2TLM A1
#define Sw3Pos3TLM A2
#define Sw3Pos4TLM A3
#define Sw3Pos5TLM A4
#define Sw3Pos6TLM A5

#define Sw4Pos1TLM A6
#define Sw4Pos2TLM A7
#define Sw4Pos3TLM A8
#define Sw4Pos4TLM A9
#define Sw4Pos5TLM A10
#define Sw4Pos6TLM A11

//Define Emergency Safe Switch Arduino Pin
#define EmergencySafe A12
int EmergencySafeState = 0;

//Define Watchdog Heartbeat Monitor Arduino Pin
#define WatchdogPin 29

//Define State Machine and Counters
int state, i;

//Define generic Switch Command Variables
```

```
int SwSelect, SwPos;

//Define Switch Telemtry Values and Initialize.
int Sw1TLM, Sw2TLM, Sw3TLM, Sw4TLM;
int       Sw1Pos1TLMstate,       Sw1Pos2TLMstate,       Sw1Pos3TLMstate,
Sw1Pos4TLMstate, Sw1Pos5TLMstate, Sw1Pos6TLMstate = 0;
int       Sw2Pos1TLMstate,       Sw2Pos2TLMstate,       Sw2Pos3TLMstate,
Sw2Pos4TLMstate, Sw2Pos5TLMstate, Sw2Pos6TLMstate = 0;
int       Sw3Pos1TLMstate,       Sw3Pos2TLMstate,       Sw3Pos3TLMstate,
Sw3Pos4TLMstate, Sw3Pos5TLMstate, Sw3Pos6TLMstate = 0;
int       Sw4Pos1TLMstate,       Sw4Pos2TLMstate,       Sw4Pos3TLMstate,
Sw4Pos4TLMstate, Sw4Pos5TLMstate, Sw4Pos6TLMstate = 0;

//Define EEPROM Write Activation Counter Variables
int SwSelectCounter, SwPosCounter;
int  Sw1Pos1Counter,  Sw1Pos2Counter,  Sw1Pos3Counter,  Sw1Pos4Counter,
Sw1Pos5Counter, Sw1Pos6Counter;
int  Sw2Pos1Counter,  Sw2Pos2Counter,  Sw2Pos3Counter,  Sw2Pos4Counter,
Sw2Pos5Counter, Sw2Pos6Counter;
int  Sw3Pos1Counter,  Sw3Pos2Counter,  Sw3Pos3Counter,  Sw3Pos4Counter,
Sw3Pos5Counter, Sw3Pos6Counter;
int  Sw4Pos1Counter,  Sw4Pos2Counter,  Sw4Pos3Counter,  Sw4Pos4Counter,
Sw4Pos5Counter, Sw4Pos6Counter;

//Define LCD Transmit PIN
SoftwareSerial mySerial(42,43); // pin 43 = TX

/////////////////////////////////////////////////////////////////////////
//CONFIGURE ETHERNET
/////////////////////////////////////////////////////////////////////////
byte ip[] = { 192, 168, 0, 77 };   //Manual setup only
//byte gateway[] = { 192, 168, 0, 1 }; //Manual setup only
//byte subnet[] = { 255, 255, 255, 0 }; //Manual setup only

// If need to change the MAC address (Very Rare)
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

EthernetServer server = EthernetServer(23); //Port 23 for Telnet
/////////////////////////////////////////////////////////////////////////
//Pins 10,11,12 & 13 are used by the ethernet shield

void setup()
{
//Define Digital OUTPUT Pins
  pinMode(Pos1, OUTPUT);
  pinMode(Pos2, OUTPUT);
```

```
  pinMode(Pos3, OUTPUT);
  pinMode(Pos4, OUTPUT);
  pinMode(Pos5, OUTPUT);
  pinMode(Pos6, OUTPUT);
  pinMode(ReturnPos, OUTPUT);

  pinMode(Sw1, OUTPUT);
  pinMode(Sw2, OUTPUT);
  pinMode(Sw3, OUTPUT);
  pinMode(Sw4, OUTPUT);

//Define Digital and Analog INPUT Pins
//Switch 1
  pinMode(Sw1Pos1TLM, INPUT_PULLUP);  //Activate Internal Pullup Resistor
  pinMode(Sw1Pos2TLM, INPUT_PULLUP);
  pinMode(Sw1Pos3TLM, INPUT_PULLUP);
  pinMode(Sw1Pos4TLM, INPUT_PULLUP);
  pinMode(Sw1Pos5TLM, INPUT_PULLUP);
  pinMode(Sw1Pos6TLM, INPUT_PULLUP);

//Switch 2
  pinMode(Sw2Pos1TLM, INPUT_PULLUP);
  pinMode(Sw2Pos2TLM, INPUT_PULLUP);
  pinMode(Sw2Pos3TLM, INPUT_PULLUP);
  pinMode(Sw2Pos4TLM, INPUT_PULLUP);
  pinMode(Sw2Pos5TLM, INPUT_PULLUP);
  pinMode(Sw2Pos6TLM, INPUT_PULLUP);

//Switch 3
  pinMode(Sw3Pos1TLM, INPUT_PULLUP);
  pinMode(Sw3Pos2TLM, INPUT_PULLUP);
  pinMode(Sw3Pos3TLM, INPUT_PULLUP);
  pinMode(Sw3Pos4TLM, INPUT_PULLUP);
  pinMode(Sw3Pos5TLM, INPUT_PULLUP);
  pinMode(Sw3Pos6TLM, INPUT_PULLUP);

//Switch 4
  pinMode(Sw4Pos1TLM, INPUT_PULLUP);
  pinMode(Sw4Pos2TLM, INPUT_PULLUP);
  pinMode(Sw4Pos3TLM, INPUT_PULLUP);
  pinMode(Sw4Pos4TLM, INPUT_PULLUP);
  pinMode(Sw4Pos5TLM, INPUT_PULLUP);
  pinMode(Sw4Pos6TLM, INPUT_PULLUP);

//Define Emergency Safe Button Inputs
  pinMode(EmergencySafe, INPUT);
```

```
delay(2000);          //Delay to allow for initialization

   Serial.begin(9600); // Opens serial port, sets data rate to 9600 bps
///NOTE: Serial Monitor is only used for code Troubleshooting///

Serial.println("Start");

Ethernet.begin(mac, ip); //for manual setup.  Begins Ethernet Connection
server.begin();
Serial.println(Ethernet.localIP());  //Prints IP address in Serial Monitor

  mySerial.begin(9600); // set up LCD display serial port for 9600 baud
  delay(500); // wait for display to boot up


  //mySerial.write(124); // set LCD Contrast
  //mySerial.write(155);   (128 MIN - 157 MAX)


  mySerial.write(254); // cursor to beginning of first line
  mySerial.write(128);

  mySerial.write("               "); // clear display + legends

  mySerial.write(254); // cursor to beginning of second line
  mySerial.write(192);
  mySerial.write("               ");

}

void loop()
{
  CheckForClient();     //Looks for Ethernet Commands; goes to loop
  TelemetryCode();      //Loops TLM Code
  EmergencySafeCode();  //Looks for Emergency Safe Button
    Watchdog();         //'Pats' the Watchdog Heartbeat Monitor and Displays
Heartbeat
 }

void CheckForClient()
{
  EthernetClient client = server.available();

    if (client) {
```

```
    Serial.println("Connected");

    while (client.connected()) {
       if (client.available()) {
         char c = client.read();

          // command parser is implemented as a simple state machine
          switch (state){
            case 0:
              if (c == '*') {
                SCPI_IDENTITY();  //Goes to SCPI Identity Test Function
              }
              if (c == 'T') {
                    EEPROMCounterRead();  //Goes to Switch Activation Counter
Function
              }
              if (c == 'S') state++;
              else {  //Reset the State Machine
                 state = 0;
                 SwSelect = NULL;
                 SwPos = NULL;
                 return;
                   }
              break;
            case 1:
              if (c == '(')
                    {
                    state++;
                    break;
                    }
                 if (c == '1' || c == '2' || c == '3' || c == '4')
                    {
                       SwSelect = c - 48;
                    state = 0;
                       SCPI_Telemetry();    //Goes to Switch Position GUI Display
Function
                    }
               else {  //Reset the State Machine
                 state = 0;
                 SwSelect = NULL;
                 SwPos = NULL;
                 client.flush();
                 delay(100);
                 return;
                   }
              break;
```

```
            case 2:
              SwSelect = c - 48;
              state++;
              break;
            case 3:
              if ((c == ':') || (c == ',') || (c == '.') || (c == ';')) state++;
               else {  //Reset the State Machine
                  state = 0;
                  SwSelect = NULL;
                  SwPos = NULL;
                  client.flush();
                  delay(100);
                  return;
                    }
              break;
             case 4:
              SwPos = c - 48;
              state++;
              break;
            case 5:
              if (c == ')') {
                    Serial.println("SwSelect:");
                    Serial.println(SwSelect);
                    Serial.println("SwPos:");
                    Serial.println(SwPos);
//                        client.println("Command Loaded");
//                        client.println("Selected Switch: ");
//                        client.println(SwSelect);
//                        client.println("Position: ");
//                        client.println(SwPos);
                           executeCommand();       //Goes to Command Execute
Function
                    state = 0;
                delay(100);
                return;
                }
              else {  //Reset the State Machine
                  state = 0;
                  SwSelect = NULL;
                  SwPos = NULL;
                  client.flush();
                  delay(100);
                  return;
                    }
              break;
            }
```

```
          }
        }
    }
}


void executeCommand() {  //This function drives all required Arduino Pins to activate appropriate switch and position

  //Commanding for Switch 1
  if(SwSelect == 1)
    {
     if(SwPos == 1) {
       digitalWrite(ReturnPos, HIGH);  //Resets all Switch 1 Positions to OPEN
        delay(5);
        digitalWrite(Sw1, HIGH);
        delay(75);
        digitalWrite(Sw1, LOW);
        delay(5);
        digitalWrite(ReturnPos, LOW);
       delay(250);
       digitalWrite(Pos1, HIGH);      //Drives Switch 1 latch to desired Position
        delay(5);
        digitalWrite(Sw1, HIGH);
        delay(75);
        digitalWrite(Sw1, LOW);
        delay(5);
        digitalWrite(Pos1, LOW);
         EEPROM_readAnything(0, Sw1Pos1Counter);  //Read EEPROM counter
            Sw1Pos1Counter = Sw1Pos1Counter + 1;      //Add +1 to EEPROM counter
             EEPROM_writeAnything(0, Sw1Pos1Counter); //Write new value to EEPROM counter

            mySerial.write(254);
            mySerial.write(192); // cursor to 1st position on second line
            mySerial.write("S(1,1)        "); // write out the CMD
       return;
      }

     if(SwPos == 2) {
       digitalWrite(ReturnPos, HIGH);
        delay(5);
        digitalWrite(Sw1, HIGH);
        delay(75);
        digitalWrite(Sw1, LOW);
```

```
   delay(5);
   digitalWrite(ReturnPos, LOW);
  delay(250);
  digitalWrite(Pos2, HIGH);
   delay(5);
   digitalWrite(Sw1, HIGH);
   delay(75);
   digitalWrite(Sw1, LOW);
   delay(5);
   digitalWrite(Pos2, LOW);
    EEPROM_readAnything(5, Sw1Pos2Counter);
    Sw1Pos2Counter = Sw1Pos2Counter + 1;
    EEPROM_writeAnything(5, Sw1Pos2Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(1,2)         "); // write out the CMD
  return;
 }

if(SwPos == 3) {
  digitalWrite(ReturnPos, HIGH);
   delay(5);
   digitalWrite(Sw1, HIGH);
   delay(75);
   digitalWrite(Sw1, LOW);
   delay(5);
   digitalWrite(ReturnPos, LOW);
  delay(250);
  digitalWrite(Pos3, HIGH);
   delay(5);
   digitalWrite(Sw1, HIGH);
   delay(75);
   digitalWrite(Sw1, LOW);
   delay(5);
   digitalWrite(Pos3, LOW);
    EEPROM_readAnything(10, Sw1Pos3Counter);
    Sw1Pos3Counter = Sw1Pos3Counter + 1;
    EEPROM_writeAnything(10, Sw1Pos3Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(1,3)         "); // write out the CMD
  return;
 }
```

```
 if(SwPos == 4) {
   digitalWrite(ReturnPos, HIGH);
    delay(5);
    digitalWrite(Sw1, HIGH);
    delay(75);
    digitalWrite(Sw1, LOW);
    delay(5);
    digitalWrite(ReturnPos, LOW);
   delay(250);
   digitalWrite(Pos4, HIGH);
    delay(5);
    digitalWrite(Sw1, HIGH);
    delay(75);
    digitalWrite(Sw1, LOW);
    delay(5);
    digitalWrite(Pos4, LOW);
     EEPROM_readAnything(15, Sw1Pos4Counter);
     Sw1Pos4Counter = Sw1Pos4Counter + 1;
     EEPROM_writeAnything(15, Sw1Pos4Counter);

          mySerial.write(254);
          mySerial.write(192); // cursor to 1st position on second line
          mySerial.write("S(1,4)         "); // write out the CMD
   return;
  }

if(SwPos == 5) {
   digitalWrite(ReturnPos, HIGH);
    delay(5);
    digitalWrite(Sw1, HIGH);
    delay(75);
    digitalWrite(Sw1, LOW);
    delay(5);
    digitalWrite(ReturnPos, LOW);
   delay(250);
   digitalWrite(Pos5, HIGH);
    delay(5);
    digitalWrite(Sw1, HIGH);
    delay(75);
    digitalWrite(Sw1, LOW);
    delay(5);
    digitalWrite(Pos5, LOW);
     EEPROM_readAnything(20, Sw1Pos5Counter);
     Sw1Pos5Counter = Sw1Pos5Counter + 1;
     EEPROM_writeAnything(20, Sw1Pos5Counter);
```

```
            mySerial.write(254);
            mySerial.write(192); // cursor to 1st position on second line
            mySerial.write("S(1,5)         "); // write out the CMD
       return;
      }

     if(SwPos == 6) {
       digitalWrite(ReturnPos, HIGH);
        delay(5);
        digitalWrite(Sw1, HIGH);
        delay(75);
        digitalWrite(Sw1, LOW);
        delay(5);
        digitalWrite(ReturnPos, LOW);
       delay(250);
       digitalWrite(Pos6, HIGH);
        delay(5);
        digitalWrite(Sw1, HIGH);
        delay(75);
        digitalWrite(Sw1, LOW);
        delay(5);
        digitalWrite(Pos6, LOW);
         EEPROM_readAnything(25, Sw1Pos6Counter);
         Sw1Pos6Counter = Sw1Pos6Counter + 1;
         EEPROM_writeAnything(25, Sw1Pos6Counter);

            mySerial.write(254);
            mySerial.write(192); // cursor to 1st position on second line
            mySerial.write("S(1,6)         "); // write out the CMD
       return;
      }
//Unlactch Switch to Return Position
     if(SwPos == 7) {
       digitalWrite(ReturnPos, HIGH);
        delay(5);
        digitalWrite(Sw1, HIGH);
        delay(75);
        digitalWrite(Sw1, LOW);
        delay(5);
        digitalWrite(ReturnPos, LOW);
       return;

            mySerial.write(254);
            mySerial.write(192); // cursor to 1st position on second line
            mySerial.write("S(1,7)         "); // write out the CMD
      }
```

```
    }

    //Commanding for Switch 2
if(SwSelect == 2)
  {
   if(SwPos == 1) {
      digitalWrite(ReturnPos, HIGH);
       delay(5);
       digitalWrite(Sw2, HIGH);
       delay(75);
       digitalWrite(Sw2, LOW);
       delay(5);
       digitalWrite(ReturnPos, LOW);
      delay(250);
      digitalWrite(Pos1, HIGH);
       delay(5);
       digitalWrite(Sw2, HIGH);
       delay(75);
       digitalWrite(Sw2, LOW);
       delay(5);
       digitalWrite(Pos1, LOW);
        EEPROM_readAnything(30, Sw2Pos1Counter);
        Sw2Pos1Counter = Sw2Pos1Counter + 1;
        EEPROM_writeAnything(30, Sw2Pos1Counter);

            mySerial.write(254);
            mySerial.write(192); // cursor to 1st position on second line
            mySerial.write("S(2,1)         "); // write out the CMD
      return;
     }

    if(SwPos == 2) {
      digitalWrite(ReturnPos, HIGH);
       delay(5);
       digitalWrite(Sw2, HIGH);
       delay(75);
       digitalWrite(Sw2, LOW);
       delay(5);
       digitalWrite(ReturnPos, LOW);
      delay(250);
      digitalWrite(Pos2, HIGH);
       delay(5);
       digitalWrite(Sw2, HIGH);
       delay(75);
       digitalWrite(Sw2, LOW);
       delay(5);
```

```
   digitalWrite(Pos2, LOW);
    EEPROM_readAnything(35, Sw2Pos2Counter);
    Sw2Pos2Counter = Sw2Pos2Counter + 1;
    EEPROM_writeAnything(35, Sw2Pos2Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(2,2)         "); // write out the CMD
  return;
 }

if(SwPos == 3) {
  digitalWrite(ReturnPos, HIGH);
   delay(5);
   digitalWrite(Sw2, HIGH);
   delay(75);
   digitalWrite(Sw2, LOW);
   delay(5);
   digitalWrite(ReturnPos, LOW);
  delay(250);
  digitalWrite(Pos3, HIGH);
   delay(5);
   digitalWrite(Sw2, HIGH);
   delay(75);
   digitalWrite(Sw2, LOW);
   delay(5);
   digitalWrite(Pos3, LOW);
    EEPROM_readAnything(40, Sw2Pos3Counter);
    Sw2Pos3Counter = Sw2Pos3Counter + 1;
    EEPROM_writeAnything(40, Sw2Pos3Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(2,3)         "); // write out the CMD
  return;
 }

if(SwPos == 4) {
  digitalWrite(ReturnPos, HIGH);
   delay(5);
   digitalWrite(Sw2, HIGH);
   delay(75);
   digitalWrite(Sw2, LOW);
   delay(5);
   digitalWrite(ReturnPos, LOW);
  delay(250);
```

```
   digitalWrite(Pos4, HIGH);
    delay(5);
    digitalWrite(Sw2, HIGH);
    delay(75);
    digitalWrite(Sw2, LOW);
    delay(5);
    digitalWrite(Pos4, LOW);
     EEPROM_readAnything(45, Sw2Pos4Counter);
     Sw2Pos4Counter = Sw2Pos4Counter + 1;
     EEPROM_writeAnything(45, Sw2Pos4Counter);

          mySerial.write(254);
          mySerial.write(192); // cursor to 1st position on second line
          mySerial.write("S(2,4)         "); // write out the CMD
   return;
  }

if(SwPos == 5) {
   digitalWrite(ReturnPos, HIGH);
    delay(5);
    digitalWrite(Sw2, HIGH);
    delay(75);
    digitalWrite(Sw2, LOW);
    delay(5);
    digitalWrite(ReturnPos, LOW);
   delay(250);
   digitalWrite(Pos5, HIGH);
    delay(5);
    digitalWrite(Sw2, HIGH);
    delay(75);
    digitalWrite(Sw2, LOW);
    delay(5);
    digitalWrite(Pos5, LOW);
     EEPROM_readAnything(50, Sw2Pos5Counter);
     Sw2Pos5Counter = Sw2Pos5Counter + 1;
     EEPROM_writeAnything(50, Sw2Pos5Counter);

          mySerial.write(254);
          mySerial.write(192); // cursor to 1st position on second line
          mySerial.write("S(2,5)         "); // write out the CMD
   return;
  }

 if(SwPos == 6) {
   digitalWrite(ReturnPos, HIGH);
    delay(5);
```

```
        digitalWrite(Sw2, HIGH);
        delay(75);
        digitalWrite(Sw2, LOW);
        delay(5);
        digitalWrite(ReturnPos, LOW);
       delay(250);
       digitalWrite(Pos6, HIGH);
        delay(5);
        digitalWrite(Sw2, HIGH);
        delay(75);
        digitalWrite(Sw2, LOW);
        delay(5);
        digitalWrite(Pos6, LOW);
         EEPROM_readAnything(55, Sw2Pos6Counter);
         Sw2Pos6Counter = Sw2Pos6Counter + 1;
         EEPROM_writeAnything(55, Sw2Pos6Counter);

             mySerial.write(254);
             mySerial.write(192); // cursor to 1st position on second line
             mySerial.write("S(2,6)         "); // write out the CMD
       return;
      }
//Unlactch Switch to Return Position
     if(SwPos == 7) {
       digitalWrite(ReturnPos, HIGH);
        delay(5);
        digitalWrite(Sw2, HIGH);
        delay(75);
        digitalWrite(Sw2, LOW);
        delay(5);
        digitalWrite(ReturnPos, LOW);

             mySerial.write(254);
             mySerial.write(192); // cursor to 1st position on second line
             mySerial.write("S(2,7)         "); // write out the CMD
       return;
      }
   }

//Commanding for Switch 3
  if(SwSelect == 3)
    {
     if(SwPos == 1) {
       digitalWrite(ReturnPos, HIGH);
        delay(5);
        digitalWrite(Sw3, HIGH);
```

```
   delay(75);
   digitalWrite(Sw3, LOW);
   delay(5);
   digitalWrite(ReturnPos, LOW);
  delay(250);
  digitalWrite(Pos1, HIGH);
   delay(5);
   digitalWrite(Sw3, HIGH);
   delay(75);
   digitalWrite(Sw3, LOW);
   delay(5);
   digitalWrite(Pos1, LOW);
    EEPROM_readAnything(60, Sw3Pos1Counter);
    Sw3Pos1Counter = Sw3Pos1Counter + 1;
    EEPROM_writeAnything(60, Sw3Pos1Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(3,1)         "); // write out the CMD
  return;
 }

if(SwPos == 2) {
  digitalWrite(ReturnPos, HIGH);
   delay(5);
   digitalWrite(Sw3, HIGH);
   delay(75);
   digitalWrite(Sw3, LOW);
   delay(5);
   digitalWrite(ReturnPos, LOW);
  delay(250);
  digitalWrite(Pos2, HIGH);
   delay(5);
   digitalWrite(Sw3, HIGH);
   delay(75);
   digitalWrite(Sw3, LOW);
   delay(5);
   digitalWrite(Pos2, LOW);
    EEPROM_readAnything(65, Sw3Pos2Counter);
    Sw3Pos2Counter = Sw3Pos2Counter + 1;
    EEPROM_writeAnything(65, Sw3Pos2Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(3,2)         "); // write out the CMD
  return;
```

```
 }

if(SwPos == 3) {
  digitalWrite(ReturnPos, HIGH);
   delay(5);
   digitalWrite(Sw3, HIGH);
   delay(75);
   digitalWrite(Sw3, LOW);
   delay(5);
   digitalWrite(ReturnPos, LOW);
  delay(250);
  digitalWrite(Pos3, HIGH);
   delay(5);
   digitalWrite(Sw3, HIGH);
   delay(75);
   digitalWrite(Sw3, LOW);
   delay(5);
   digitalWrite(Pos3, LOW);
    EEPROM_readAnything(70, Sw3Pos3Counter);
    Sw3Pos3Counter = Sw3Pos3Counter + 1;
    EEPROM_writeAnything(70, Sw3Pos3Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(3,3)         "); // write out the CMD
  return;
 }

if(SwPos == 4) {
  digitalWrite(ReturnPos, HIGH);
   delay(5);
   digitalWrite(Sw3, HIGH);
   delay(75);
   digitalWrite(Sw3, LOW);
   delay(5);
   digitalWrite(ReturnPos, LOW);
  delay(250);
  digitalWrite(Pos4, HIGH);
   delay(5);
   digitalWrite(Sw3, HIGH);
   delay(75);
   digitalWrite(Sw3, LOW);
   delay(5);
   digitalWrite(Pos4, LOW);
    EEPROM_readAnything(75, Sw3Pos4Counter);
    Sw3Pos4Counter = Sw3Pos4Counter + 1;
```

```
    EEPROM_writeAnything(75, Sw3Pos4Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(3,4)         "); // write out the CMD
  return;
  }

if(SwPos == 5) {
   digitalWrite(ReturnPos, HIGH);
    delay(5);
    digitalWrite(Sw3, HIGH);
    delay(75);
    digitalWrite(Sw3, LOW);
    delay(5);
    digitalWrite(ReturnPos, LOW);
   delay(250);
   digitalWrite(Pos5, HIGH);
    delay(5);
    digitalWrite(Sw3, HIGH);
    delay(75);
    digitalWrite(Sw3, LOW);
    delay(5);
    digitalWrite(Pos5, LOW);
     EEPROM_readAnything(80, Sw3Pos5Counter);
     Sw3Pos5Counter = Sw3Pos5Counter + 1;
     EEPROM_writeAnything(80, Sw3Pos5Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(3,5)         "); // write out the CMD
  return;
  }

 if(SwPos == 6) {
   digitalWrite(ReturnPos, HIGH);
    delay(5);
    digitalWrite(Sw3, HIGH);
    delay(75);
    digitalWrite(Sw3, LOW);
    delay(5);
    digitalWrite(ReturnPos, LOW);
   delay(250);
   digitalWrite(Pos6, HIGH);
    delay(5);
    digitalWrite(Sw3, HIGH);
```

```
        delay(75);
        digitalWrite(Sw3, LOW);
        delay(5);
        digitalWrite(Pos6, LOW);
        EEPROM_readAnything(85, Sw3Pos6Counter);
        Sw3Pos6Counter = Sw3Pos6Counter + 1;
        EEPROM_writeAnything(85, Sw3Pos6Counter);

            mySerial.write(254);
            mySerial.write(192); // cursor to 1st position on second line
            mySerial.write("S(3,6         "); // write out the CMD
       return;
      }
//Unlactch Switch to Return Position
     if(SwPos == 7) {
       digitalWrite(ReturnPos, HIGH);
        delay(5);
        digitalWrite(Sw3, HIGH);
        delay(75);
        digitalWrite(Sw3, LOW);
        delay(5);
        digitalWrite(ReturnPos, LOW);

            mySerial.write(254);
            mySerial.write(192); // cursor to 1st position on second line
            mySerial.write("S(3,7         "); // write out the CMD
       return;
      }
   }

//Commanding for Switch 4
  if(SwSelect == 4)
    {
     if(SwPos == 1) {
       digitalWrite(ReturnPos, HIGH);
        delay(5);
        digitalWrite(Sw4, HIGH);
        delay(75);
        digitalWrite(Sw4, LOW);
        delay(5);
        digitalWrite(ReturnPos, LOW);
       delay(250);
       digitalWrite(Pos1, HIGH);
        delay(5);
        digitalWrite(Sw4, HIGH);
        delay(75);
```

```
   digitalWrite(Sw4, LOW);
   delay(5);
   digitalWrite(Pos1, LOW);
    EEPROM_readAnything(90, Sw4Pos1Counter);
    Sw4Pos1Counter = Sw4Pos1Counter + 1;
    EEPROM_writeAnything(90, Sw4Pos1Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(4,1)        "); // write out the CMD
  return;
 }

if(SwPos == 2) {
  digitalWrite(ReturnPos, HIGH);
   delay(5);
   digitalWrite(Sw4, HIGH);
   delay(75);
   digitalWrite(Sw4, LOW);
   delay(5);
   digitalWrite(ReturnPos, LOW);
  delay(250);
  digitalWrite(Pos2, HIGH);
   delay(5);
   digitalWrite(Sw4, HIGH);
   delay(75);
   digitalWrite(Sw4, LOW);
   delay(5);
   digitalWrite(Pos2, LOW);
    EEPROM_readAnything(95, Sw4Pos2Counter);
    Sw4Pos2Counter = Sw4Pos2Counter + 1;
    EEPROM_writeAnything(95, Sw4Pos2Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(4,2)        "); // write out the CMD
  return;
 }

if(SwPos == 3) {
  digitalWrite(ReturnPos, HIGH);
   delay(5);
   digitalWrite(Sw4, HIGH);
   delay(75);
   digitalWrite(Sw4, LOW);
   delay(5);
```

```
   digitalWrite(ReturnPos, LOW);
  delay(250);
  digitalWrite(Pos3, HIGH);
   delay(5);
   digitalWrite(Sw4, HIGH);
   delay(75);
   digitalWrite(Sw4, LOW);
   delay(5);
   digitalWrite(Pos3, LOW);
    EEPROM_readAnything(100, Sw4Pos3Counter);
    Sw4Pos3Counter = Sw4Pos3Counter + 1;
    EEPROM_writeAnything(100, Sw4Pos3Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(4,3)        "); // write out the CMD
  return;
 }

 if(SwPos == 4) {
  digitalWrite(ReturnPos, HIGH);
   delay(5);
   digitalWrite(Sw4, HIGH);
   delay(75);
   digitalWrite(Sw4, LOW);
   delay(5);
   digitalWrite(ReturnPos, LOW);
  delay(250);
  digitalWrite(Pos4, HIGH);
   delay(5);
   digitalWrite(Sw4, HIGH);
   delay(75);
   digitalWrite(Sw4, LOW);
   delay(5);
   digitalWrite(Pos4, LOW);
    EEPROM_readAnything(105, Sw4Pos4Counter);
    Sw4Pos4Counter = Sw4Pos4Counter + 1;
    EEPROM_writeAnything(105, Sw4Pos4Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(4,4)        "); // write out the CMD
  return;
  }

if(SwPos == 5) {
```

```
  digitalWrite(ReturnPos, HIGH);
   delay(5);
   digitalWrite(Sw4, HIGH);
   delay(75);
   digitalWrite(Sw4, LOW);
   delay(5);
   digitalWrite(ReturnPos, LOW);
  delay(250);
  digitalWrite(Pos5, HIGH);
   delay(5);
   digitalWrite(Sw4, HIGH);
   delay(75);
   digitalWrite(Sw4, LOW);
   delay(5);
   digitalWrite(Pos5, LOW);
    EEPROM_readAnything(110, Sw4Pos5Counter);
    Sw4Pos5Counter = Sw4Pos5Counter + 1;
    EEPROM_writeAnything(110, Sw4Pos5Counter);

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("S(4,5)         "); // write out the CMD
  return;
 }

if(SwPos == 6) {
  digitalWrite(ReturnPos, HIGH);
   delay(5);
   digitalWrite(Sw4, HIGH);
   delay(75);
   digitalWrite(Sw4, LOW);
   delay(5);
   digitalWrite(ReturnPos, LOW);
  delay(250);
  digitalWrite(Pos6, HIGH);
   delay(5);
   digitalWrite(Sw4, HIGH);
   delay(75);
   digitalWrite(Sw4, LOW);
   delay(5);
   digitalWrite(Pos6, LOW);
    EEPROM_readAnything(115, Sw4Pos6Counter);
    Sw4Pos6Counter = Sw4Pos6Counter + 1;
    EEPROM_writeAnything(115, Sw4Pos6Counter);

        mySerial.write(254);
```

```
            mySerial.write(192); // cursor to 1st position on second line
            mySerial.write("S(4,6)         "); // write out the CMD
       return;
      }
//Unlactch Switch to Return Position
     if(SwPos == 7) {
       digitalWrite(ReturnPos, HIGH);
        delay(5);
        digitalWrite(Sw4, HIGH);
        delay(75);
        digitalWrite(Sw4, LOW);
        delay(5);
        digitalWrite(ReturnPos, LOW);

            mySerial.write(254);
            mySerial.write(192); // cursor to 1st position on second line
            mySerial.write("S(4,7)         "); // write out the CMD
       return;
      }
   }

  return;
}


void SCPI_IDENTITY(){           //Function is called when user wants to see unit
identity information
   EthernetClient client = server.available();

    if (client) {
      while (client.connected()) {
        if (client.available()) {
          char c = client.read();

            // command parser is implemented as a simple state machine
            switch (state){
              case 0:
                   if (c == 'I') state++;
                else {  //Reset the State Machine
                  state = 0;
                  SwSelectCounter = NULL;
                  SwPosCounter = NULL;
                  client.flush();
                  delay(100);
                  return;
                    }
```

```
  break;

case 1:
      if (c == 'D') state++;
   else {  //Reset the State Machine
      state = 0;
      SwSelectCounter = NULL;
      SwPosCounter = NULL;
      client.flush();
      delay(100);
      return;
        }
  break;

case 2:
      if (c == 'N') state++;
   else {  //Reset the State Machine
      state = 0;
      SwSelectCounter = NULL;
      SwPosCounter = NULL;
      client.flush();
      delay(100);
      return;
        }
  break;

  case 3:
      if (c == '?'){
    client.print("SSL,LCU/Switch Matrix,Prototype,A1.00\n");
      mySerial.write(254);
      mySerial.write(192); // cursor to 1st position on second line
      mySerial.write("*IDN? SSL:A1.00"); // write out the CMD

    state = 0;
    delay(100);
    return;
    }

   else {  //Reset the State Machine
      state = 0;
      SwSelectCounter = NULL;
      SwPosCounter = NULL;
      client.flush();
      delay(100);
      return;
        }
```

```
            break;
              }
          }
        }
    }
}

void EEPROMCounterRead(){       //Function is called when user wants to see
Activation Counter for a desired switch and position

  EthernetClient client = server.available();

   if (client) {
     while (client.connected()) {
       if (client.available()) {
         char c = client.read();

           // command parser is implemented as a simple state machine
           switch (state){
             case 0:
                  if (c == '1' || c == '2' || c == '3' || c == '4')
                       {
                          SwSelect = c - 48;
                          state++;
                       }
                 else {  //Reset the State Machine
                   state = 0;
                   SwSelectCounter = NULL;
                   SwPosCounter = NULL;
                   client.flush();
                   delay(100);
                   return;
                     }
               break;
             case 1:
                if (c == '?'){
                       if (SwSelect == 1)
                              {
                                EEPROM_readAnything(0, Sw1Pos1Counter);
                                EEPROM_readAnything(5, Sw1Pos2Counter);
                                EEPROM_readAnything(10, Sw1Pos3Counter);
                                EEPROM_readAnything(15, Sw1Pos4Counter);
                                EEPROM_readAnything(20, Sw1Pos5Counter);
                                EEPROM_readAnything(25, Sw1Pos6Counter);

                                client.print(Sw1Pos1Counter);
```

```
                client.print(',');
                client.print(Sw1Pos2Counter);
                client.print(',');
                client.print(Sw1Pos3Counter);
                client.print(',');
                client.print(Sw1Pos4Counter);
                client.print(',');
                client.print(Sw1Pos5Counter);
                client.print(',');
                client.print(Sw1Pos6Counter);
                client.print('\n');

                    mySerial.write(254);
                      mySerial.write(192); // cursor to 1st position on
second line
                        mySerial.write("T1?            "); // write out the
CMD
                return;
              }

            if(SwSelect == 2)
              {
                EEPROM_readAnything(30, Sw2Pos1Counter);
                EEPROM_readAnything(35, Sw2Pos2Counter);
                EEPROM_readAnything(40, Sw2Pos3Counter);
                EEPROM_readAnything(45, Sw2Pos4Counter);
                EEPROM_readAnything(50, Sw2Pos5Counter);
                EEPROM_readAnything(55, Sw2Pos6Counter);

                client.print(Sw2Pos1Counter);
                client.print(',');
                client.print(Sw2Pos2Counter);
                client.print(',');
                client.print(Sw2Pos3Counter);
                client.print(',');
                client.print(Sw2Pos4Counter);
                client.print(',');
                client.print(Sw2Pos5Counter);
                client.print(',');
                client.print(Sw2Pos6Counter);
                client.print('\n');
                return;

                    mySerial.write(254);
                      mySerial.write(192); // cursor to 1st position on
second line
```

```
                                    mySerial.write("T2?            "); // write out the
CMD
                                }

                               if(SwSelect == 3)
                                 {
                                     EEPROM_readAnything(60, Sw3Pos1Counter);
                                     EEPROM_readAnything(65, Sw3Pos2Counter);
                                     EEPROM_readAnything(70, Sw3Pos3Counter);
                                     EEPROM_readAnything(75, Sw3Pos4Counter);
                                     EEPROM_readAnything(80, Sw3Pos5Counter);
                                     EEPROM_readAnything(85, Sw3Pos6Counter);

                                     client.print(Sw3Pos1Counter);
                                     client.print(',');
                                     client.print(Sw3Pos2Counter);
                                     client.print(',');
                                     client.print(Sw3Pos3Counter);
                                     client.print(',');
                                     client.print(Sw3Pos4Counter);
                                     client.print(',');
                                     client.print(Sw3Pos5Counter);
                                     client.print(',');
                                     client.print(Sw3Pos6Counter);
                                     client.print('\n');

                                         mySerial.write(254);
                                           mySerial.write(192); // cursor to 1st position on
second line
                                             mySerial.write("T3?            "); // write out the
CMD
                                     return;
                                  }

                               if(SwSelect == 4)
                                 {
                                     EEPROM_readAnything(90, Sw4Pos1Counter);
                                     EEPROM_readAnything(95, Sw4Pos2Counter);
                                     EEPROM_readAnything(100, Sw4Pos3Counter);
                                     EEPROM_readAnything(105, Sw4Pos4Counter);
                                     EEPROM_readAnything(110, Sw4Pos5Counter);
                                     EEPROM_readAnything(115, Sw4Pos6Counter);

                                     client.print(Sw4Pos1Counter);
                                     client.print(',');
                                     client.print(Sw4Pos2Counter);
```

```
                                client.print(',');
                                client.print(Sw4Pos3Counter);
                                client.print(',');
                                client.print(Sw4Pos4Counter);
                                client.print(',');
                                client.print(Sw4Pos5Counter);
                                client.print(',');
                                client.print(Sw4Pos6Counter);
                                client.print('\n');

                                    mySerial.write(254);
                                      mySerial.write(192); // cursor to 1st position on
second line
                                        mySerial.write("T4?            "); // write out the
CMD
                                return;
                            }
                  state = 0;
                  delay(100);
                  return;
                  }
                 else {  //Reset the State Machine
                    state = 0;
                    SwSelectCounter = NULL;
                    SwPosCounter = NULL;
                    client.flush();
                    delay(100);
                    return;
                      }
                 break;
               }
             }
           }
      }
}


void SCPI_Telemetry(){      //Function is called when user wants to see Switch
Position for a desired switch

  EthernetClient client = server.available();

   if (client) {
     while (client.connected()) {
       if (client.available()) {
         char c = client.read();
```

```
            // command parser is implemented as a simple state machine
            switch (state){
              case 0:
                if (c == '?') {
                        if (SwSelect == 1)
                                {
                                 mySerial.write(254);
                                        mySerial.write(192); // cursor to 1st position on
second line
                                 mySerial.write("S1?           "); // write out the CMD

                                 mySerial.write(254);
                                        mySerial.write(196); // cursor to 1st position on
second line
                                 mySerial.write(Sw1TLM); // write out the TLM

                                  Sw1TLM = Sw1TLM - 48;
                                Serial.println(Sw1TLM);
                                        client.print(Sw1TLM);
                                  client.print('\n');
                                }

                        if (SwSelect == 2)
                                {
                                 mySerial.write(254);
                                        mySerial.write(192); // cursor to 1st position on
second line
                                 mySerial.write("S2?           "); // write out the CMD

                                 mySerial.write(254);
                                        mySerial.write(196); // cursor to 1st position on
second line
                                 mySerial.write(Sw2TLM); // write out the TLM

                                  Sw2TLM = Sw2TLM - 48;
                                Serial.println(Sw2TLM);
                                        client.print(Sw2TLM);
                                  client.print('\n');
                                }
                        if (SwSelect == 3)
                                {
                                 mySerial.write(254);
                                        mySerial.write(192); // cursor to 1st position on
second line
                                 mySerial.write("S3?           "); // write out the CMD
```

```
                        mySerial.write(254);
                              mySerial.write(196); // cursor to 1st position on
second line
                        mySerial.write(Sw3TLM); // write out the TLM

                         Sw3TLM = Sw3TLM - 48;
                       Serial.println(Sw3TLM);
                              client.print(Sw3TLM);
                         client.print('\n');
                       }
                  if (SwSelect == 4)
                       {
                        mySerial.write(254);
                              mySerial.write(192); // cursor to 1st position on
second line
                        mySerial.write("S4?           "); // write out the CMD

                        mySerial.write(254);
                              mySerial.write(196); // cursor to 1st position on
second line
                        mySerial.write(Sw4TLM); // write out the TLM

                         Sw4TLM = Sw4TLM - 48;
                       Serial.println(Sw4TLM);
                              client.print(Sw4TLM);
                         client.print('\n');
                    }
            state = 0;
            delay(100);
            return;
            }
          else {  //Reset the State Machine
             state = 0;
             SwSelectCounter = NULL;
             SwPosCounter = NULL;
             client.flush();
             delay(100);
             return;
               }
          break;
        }
      }
    }
  }
}
```

```
void TelemetryCode(){
  //TELEMETRY CODE
//SWITCH 1
  Sw1Pos1TLMstate = digitalRead(Sw1Pos1TLM);
  Sw1Pos2TLMstate = digitalRead(Sw1Pos2TLM);
  Sw1Pos3TLMstate = digitalRead(Sw1Pos3TLM);
  Sw1Pos4TLMstate = digitalRead(Sw1Pos4TLM);
  Sw1Pos5TLMstate = digitalRead(Sw1Pos5TLM);
  Sw1Pos6TLMstate = digitalRead(Sw1Pos6TLM);

//SWITCH 2
  Sw2Pos1TLMstate = digitalRead(Sw2Pos1TLM);
  Sw2Pos2TLMstate = digitalRead(Sw2Pos2TLM);
  Sw2Pos3TLMstate = digitalRead(Sw2Pos3TLM);
  Sw2Pos4TLMstate = digitalRead(Sw2Pos4TLM);
  Sw2Pos5TLMstate = digitalRead(Sw2Pos5TLM);
  Sw2Pos6TLMstate = digitalRead(Sw2Pos6TLM);

//SWITCH 3
  Sw3Pos1TLMstate = digitalRead(Sw3Pos1TLM);
  Sw3Pos2TLMstate = digitalRead(Sw3Pos2TLM);
  Sw3Pos3TLMstate = digitalRead(Sw3Pos3TLM);
  Sw3Pos4TLMstate = digitalRead(Sw3Pos4TLM);
  Sw3Pos5TLMstate = digitalRead(Sw3Pos5TLM);
  Sw3Pos6TLMstate = digitalRead(Sw3Pos6TLM);

//SWITCH 4
  Sw4Pos1TLMstate = digitalRead(Sw4Pos1TLM);
  Sw4Pos2TLMstate = digitalRead(Sw4Pos2TLM);
  Sw4Pos3TLMstate = digitalRead(Sw4Pos3TLM);
  Sw4Pos4TLMstate = digitalRead(Sw4Pos4TLM);
  Sw4Pos5TLMstate = digitalRead(Sw4Pos5TLM);
  Sw4Pos6TLMstate = digitalRead(Sw4Pos6TLM);

//SWITCH 1 WRITE TO LCD
  if (Sw1Pos1TLMstate == LOW) {
        Sw1TLM = '1';
       }
  if (Sw1Pos2TLMstate == LOW) {
        Sw1TLM = '2';
       }
  if (Sw1Pos3TLMstate == LOW) {
        Sw1TLM = '3';
       }
```

```
  if (Sw1Pos4TLMstate == LOW) {
        Sw1TLM = '4';
       }
  if (Sw1Pos5TLMstate == LOW) {
        Sw1TLM = '5';
       }
  if (Sw1Pos6TLMstate == LOW) {
        Sw1TLM = '6';
       }
     if (Sw1Pos1TLMstate == HIGH && Sw1Pos2TLMstate == HIGH &&
Sw1Pos3TLMstate == HIGH && Sw1Pos4TLMstate == HIGH &&
Sw1Pos5TLMstate == HIGH && Sw1Pos6TLMstate == HIGH) {
        Sw1TLM = '0';
       }

//SWITCH 2 WRITE TO LCD
  if (Sw2Pos1TLMstate == LOW) {
        Sw2TLM = '1';
       }
  if (Sw2Pos2TLMstate == LOW) {
        Sw2TLM = '2';
       }
  if (Sw2Pos3TLMstate == LOW) {
        Sw2TLM = '3';
       }
  if (Sw2Pos4TLMstate == LOW) {
        Sw2TLM = '4';
       }
  if (Sw2Pos5TLMstate == LOW) {
        Sw2TLM = '5';
       }
  if (Sw2Pos6TLMstate == LOW) {
        Sw2TLM = '6';
       }
     if (Sw2Pos1TLMstate == HIGH && Sw2Pos2TLMstate == HIGH &&
Sw2Pos3TLMstate == HIGH && Sw2Pos4TLMstate == HIGH &&
Sw2Pos5TLMstate == HIGH && Sw2Pos6TLMstate == HIGH) {
        Sw2TLM = '0';
       }

//SWITCH 3 WRITE TO LCD
  if (Sw3Pos1TLMstate == LOW) {
        Sw3TLM = '1';
       }
  if (Sw3Pos2TLMstate == LOW) {
        Sw3TLM = '2';
```

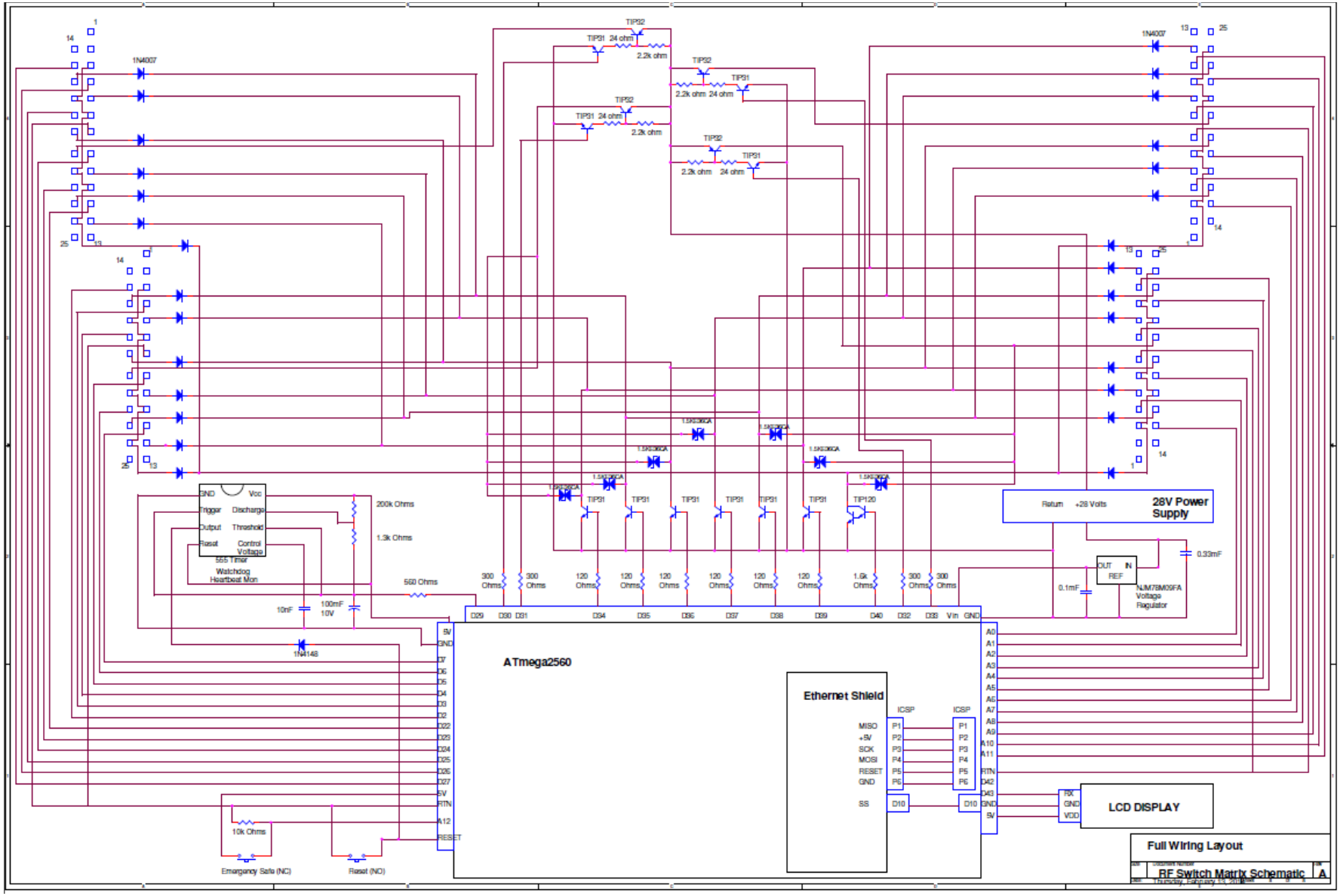
```
      }
  if (Sw3Pos3TLMstate == LOW) {
       Sw3TLM = '3';
      }
  if (Sw3Pos4TLMstate == LOW) {
       Sw3TLM = '4';
      }
  if (Sw3Pos5TLMstate == LOW) {
       Sw3TLM = '5';
      }
  if (Sw3Pos6TLMstate == LOW) {
       Sw3TLM = '6';
      }
    if (Sw3Pos1TLMstate == HIGH && Sw3Pos2TLMstate == HIGH &&
Sw3Pos3TLMstate == HIGH && Sw3Pos4TLMstate == HIGH &&
Sw3Pos5TLMstate == HIGH && Sw3Pos6TLMstate == HIGH) {
       Sw3TLM = '0';
      }

//SWITCH 4 WRITE TO LCD
  if (Sw4Pos1TLMstate == LOW) {
       Sw4TLM = '1';
      }
  if (Sw4Pos2TLMstate == LOW) {
       Sw4TLM = '2';
      }
  if (Sw4Pos3TLMstate == LOW) {
       Sw4TLM = '3';
      }
  if (Sw4Pos4TLMstate == LOW) {
       Sw4TLM = '4';
      }
  if (Sw4Pos5TLMstate == LOW) {
       Sw4TLM = '5';
      }
  if (Sw4Pos6TLMstate == LOW) {
       Sw4TLM = '6';
      }
    if (Sw4Pos1TLMstate == HIGH && Sw4Pos2TLMstate == HIGH &&
Sw4Pos3TLMstate == HIGH && Sw4Pos4TLMstate == HIGH &&
Sw4Pos5TLMstate == HIGH && Sw4Pos6TLMstate == HIGH) {
       Sw4TLM = '0';
      }


  mySerial.write(254);
```

```
  mySerial.write(128); // cursor to 1st position on first line

  mySerial.write(Sw1TLM); // write out the TLM value

  mySerial.write(254);
  mySerial.write(132); // cursor to 5th position on first line

  mySerial.write(Sw2TLM); // write out the TLM value

  mySerial.write(254);
  mySerial.write(137); // cursor to 10th position on first line

  mySerial.write(Sw3TLM); // write out the TLM value

  mySerial.write(254);
  mySerial.write(141); // cursor to 14th position on first line

  mySerial.write(Sw4TLM); // write out the TLM value

 return;
}

void EmergencySafeCode(){    //Emergency Safe is activated by a button on the Switch Matrix, it returns all Switches to OPEN position.

   EmergencySafeState = digitalRead(EmergencySafe);
     if (EmergencySafeState == LOW) {

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("Safing...      "); // write out the CMD

        //Unlatch Switch 1
            digitalWrite(ReturnPos, HIGH);
             delay(5);
             digitalWrite(Sw1, HIGH);
             delay(75);
             digitalWrite(Sw1, LOW);
             delay(5);
             digitalWrite(ReturnPos, LOW);
          delay(250);

        //Unlatch Switch 2
            digitalWrite(ReturnPos, HIGH);
             delay(5);
             digitalWrite(Sw2, HIGH);
```

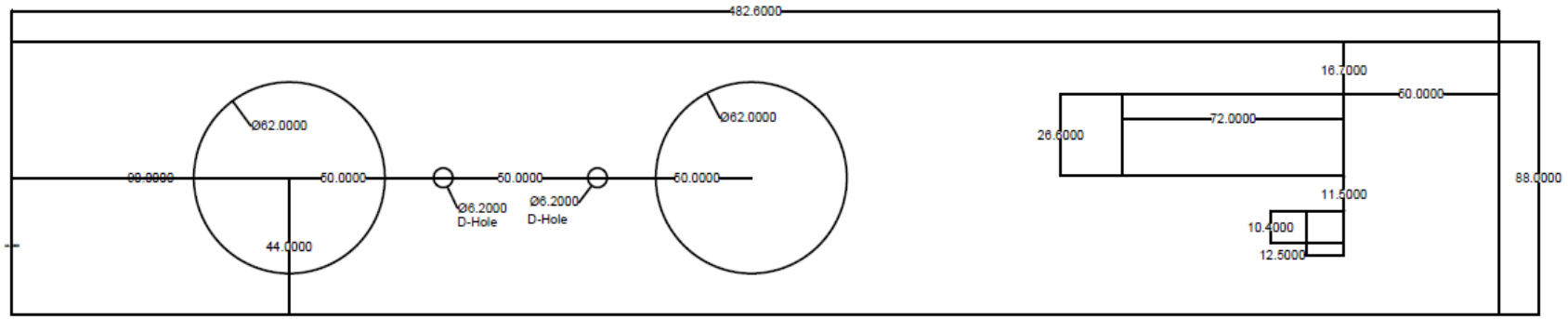
```
            delay(75);
            digitalWrite(Sw2, LOW);
            delay(5);
            digitalWrite(ReturnPos, LOW);
          delay(250);
        //Unlatch Switch 3
            digitalWrite(ReturnPos, HIGH);
            delay(5);
            digitalWrite(Sw3, HIGH);
            delay(75);
            digitalWrite(Sw3, LOW);
            delay(5);
            digitalWrite(ReturnPos, LOW);
          delay(250);
        //Unlatch Switch 4
            digitalWrite(ReturnPos, HIGH);
            delay(5);
            digitalWrite(Sw4, HIGH);
            delay(75);
            digitalWrite(Sw4, LOW);
            delay(5);
            digitalWrite(ReturnPos, LOW);
          delay(250);


         delay(1000);  // Delay to prevent constant 28V Pulses.

        mySerial.write(254);
        mySerial.write(192); // cursor to 1st position on second line
        mySerial.write("Safe          "); // write out the CMD
        return;
      }
      else {return;}
}

void Watchdog()                //Function "pats" Watchdog Circuit and Displays
Heartbeat on LCD screen.  AKA Heartbeat Monitor
{
   pinMode(WatchdogPin, OUTPUT); //The WatchdogPin line goes Low
   delay(50);                  //Wait for capacitor discharge
   pinMode(WatchdogPin, INPUT);  //The WatchdogPin line goes Hi-Z

i = i + 1;
if(i < 20)
{
  //Display Heartbeat on LCD Screen
```

```
  mySerial.write(254);
  mySerial.write(207); // cursor to 16th position on second line
  mySerial.write('*'); // write out the TLM value
  }

if(i > 20)
{
  //Blink Heartbeat on LCD Screen
  mySerial.write(254);
  mySerial.write(207); // cursor to 16th position on second line
  mySerial.write(' '); // write out the TLM value
}
if(i == 40) i=0;    //reset counter
return;
}
```

APPENDIX J: Switch Matrix Full Wiring Schematic



APPENDIX K: Front Plate CAD Drawing


482.6000
Ø62.0000
00.0000
50.0000
44.0000
50.0000
Ø6.2000
D-Hole
Ø6.2000
D-Hole
50.0000
Ø62.0000
16.7000
50.0000
72.0000
26.6000
11.5000
10.4000
12.5000
88.0000

APPENDIX L: Back Plate CAD Drawing