12-1-2019

SOLID-STATE MARX GENERATOR UTILIZING IGBTS AND AN ISOLATED GATE DRIVER SYSTEM

Benjamin Barnett

SOLID-STATE MARX GENERATOR UTILIZING IGBTS AND AN ISOLATED GATE DRIVER SYSTEM

by

BENJAMIN M. BARNETT

Presented to the Faculty of the Honors College of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

HONORS BACHELOR OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2019

ACKNOWLEDGMENTS

I wish to thank Dr. David Wetz of the College of Electrical Engineering for sponsoring and overseeing this Senior Project and my associated Honors College contribution. I am very grateful to Dr. Wetz for his input and willingness to offer opportunities throughout my last year in college including research experience and study abroad in Japan I would have been unable to achieve otherwise. I would like to thank my fellow senior engineering team members Hayden Atchison, Jared Bates, and Vincent Gutierrez for their support and contributions to successfully achieve a finished product.

From the College of Physics, I must thank Dr. Haleh Hadavand, Dr. Amir Farbin, but most especially Dr. Jonathan Asaadi for his mentorship as I grew as a researcher over my few years at UT Arlington. I wish to also thank my project partner during my time with Dr. Asaadi, Hunter Sullivan, for his help and advice.

I would like to thank my friends in the UT Arlington IEEE student organization for our commiseration, discussion, and homework help. I also wish to thank my friend Dr. Randall Gladen for his research help and comradery in discussions of physics, Japan, lofty future goals, and video games. I am grateful as well for my friends from the Honors College, the Terry Scholars student organization, and my roommates for all the enjoyable times we shared.

I must humbly thank my parents for their longstanding sacrifices and loving support as I pursue my dreams. They have given much to help me shoot as high as I can

not just in university, but in life. I also thank my brother, for his ambitions have always inspired me to keep dreaming.

A very special thanks is in order to the Terry Foundation for believing in me and blessing me with the resources to go to college and make all of this possible.

December 10, 2019

ABSTRACT

SOLID-STATE MARX GENERATOR UTILIZING IGBTS AND AN ISOLATED GATE DRIVER SYSTEM

Benjamin M. Barnett, B.S. E.E.

The University of Texas at Arlington, 2019

Faculty Mentor: David Wetz

Marx generators are pulsed power generators useful for applications such as particle accelerators, agricultural food treatment, medical research and treatments, flashed x-rays, laser welding and ablation, and high-power microwaves. This type of generator is often high voltage, and provides short (nanosecond – microsecond long) pulses of power. Many well-documented designs utilize spark gap switches, but recent advancements in solid-state power electronics components like IGBTs make high-power transistor switching possible. Although limited to low voltage (<10kV) by the current transistor technology, there are clear benefits to this design, namely precise switching timing, allowing for pulse shaping, and RLC ringing reduction. The aim of this Senior engineering project was to build a five-stage solid-state Marx generator that could generate a 100us, 50V, 2500W output pulse. The design discussed in this work details implementation of IGBTs and isolated gate drivers to achieve this goal. The design has been demonstrated to

generate the specified pulse with experimentally acceptable losses. In addition, a microcontroller was utilized to control pulse timing to achieve not only the project specifications but to attempt output pulse shaping and display information to a user.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

BACKGROUND

1.1 Introduction to Marx Generators

1.1.1 Basic Function

A Marx generator is a type of pulsed power generator invented by Erwin Otto Marx in 1924 [1]. It has a number of "stages" containing a capacitor, resistors, and some type of switch. It functions by charging the capacitors in the circuit in a parallel configuration via a charging voltage, and then discharging those capacitors in a series configuration, so that the output voltage into a load is the charging voltage multiplied by number of capacitor stages of the generator. A Marx generator is changed between charging and discharging states using switches of some kind.

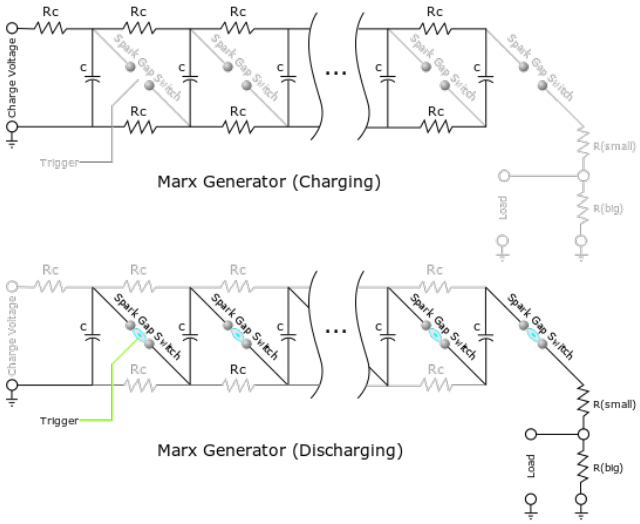

Figure 1.1: Marx Generator Topology During Charging and Discharging States [2]

1.1.2 Applications

Historically, Marx generators have served several uses in research and industrial applications, notably particle accelerators, flashed x-ray production, high-power microwave research, agricultural food treatment, medical research and treatments, and laser welding and ablation. One particularly interesting example is the Z-Machine at Sandia National Laboratory in Albuquerque, New Mexico. The Z-Machine utilizes thirty-six units of 6MV thirty-stage Marx generators to conduct high-energy-density physics research. Many experiments performed are related to plasmas and fusion energy research. [3]

1.1.3 Spark Gap Switches

In order to change the circuit topology and discharge the capacitors into a load, high voltage Marx generators typically employ spark gap switches. Specifically, midplane triggered spark gap switches may be used, which function by a controlled breakdown of a dielectric pressurized gas in between two conductive plates. A “midplane” plate spaced equally between these two plates is biased to a certain voltage, and controlled to cause a breakdown arcing from one conductive plate to the midplane. Once this happens, the other half also breaks down, and the switch is closed via arcing through the pressurized gas inside the spark gap. The behavior of this breakdown follows a curve, called the Paschen curve, that relates the type of gas, its pressure, and the distance between the plates to the breakdown voltage. These spark gap controlled type of a Marx generators tend to be operated in ranges of 10’s of kilovolts to 10’s of megavolts and have been well studied since the early days of pulsed power research.

1.1.4 Solid-state Transistors

By comparison, the use of solid-state switches in Marx generators is a relatively new endeavor. Since the advent of the transistor, solid-state electronics and their manufacturing processes have steadily improved. Although there is still much work to be done to get to the same 10's of kilovolts to 10's of megavolts range that traditional spark gap switches can handle, power electronics transistors like the power MOSFET and IGBT have come a long way. Currently, some of the highest rated power MOSFETs like the IXYS IXTT1N450HV can handle up to 4.5kV before breaking down. The newer viability of these power electronics transistors in lower voltage Marx generators also brings several innate advantages over spark gap switching designs in a Marx generator pulse power topology. Spark gap switches cannot be turned off once activated, but solid-state power transistors can be turned on and off at will.

1.2 Pulse Shaping

Because of the ability to turn the solid-state transistor switches on and off at will, different Marx generator circuit topologies can be made, for example, utilizing diodes instead of resistors in the circuit. In addition, with solid-state switches, the triggering of different stages can be asynchronously coordinated by some sort of digital logic controller, resulting in an output pulse with a desired pulse length and shape.

CHAPTER 2

DESIGN CONSIDERATIONS

2.1 MOSFETs, IGBTs, and BJTs

2.1.1 BJTs

Bipolar Junction Transistors are current-controlled devices. A BJT has three connection nodes: base, collector, and emitter. Based on the current supplied to the base, the current flowing between the collector and emitter is regulated.

2.1.1 Power MOSFETs

Metal Oxide Semiconductor Field Effect Transistors are voltage-controlled devices with four connection nodes: gate, drain, source, and bulk. Utilizing a biasing voltage between gate and drain terminals allows for control of current flow across the drain to source. Very commonly the bulk node is tied to the source node internally with the fabrication of the transistor. Power MOSFETs are capable of carrying some of the highest power among solid-state transistors.

2.1.1 IGBTs

Insulated Gate Bipolar Transistors are effectively a hybrid between BJT and MOSFET transistor technologies. They are voltage controlled and have three terminals: gate, collector, and emitter. Due to their unique design, they utilize two types of carriers, electrons and holes, and achieve an ON state with high switching speed and minimal saturation voltages. However, they still do not handle as much power as certain Power MOSFETs. [4]

2.2 Component Selection Guidelines

The specifications of our Senior engineering project were to build a five-stage solid-state Marx generator that could generate a 100us, 50V, 2500W output pulse. It was required to use IGBTs. Since every application or project may be different, it is worth discussion general considerations used to choose circuit components in this project. The specific components chosen for this project and justification will be discussed in the next chapter.

2.2.1 Resistors

Resistors must be an appropriate value, small enough to allow current for charging of the capacitors during charging state, but large enough to minimize current flow and incentivize current flow through the IGBT switches when they are enabled and the capacitors are stacked in a series configuration. Remember that the resistors must also have an appropriate power rating for the currents running through them at any one point in time. The pulsed power rating is lower than the constant power rating, however by how much depends on the length of the pulse. One effective way of estimating the resistor value is to run a circuit simulation.

2.2.2 Capacitors

Capacitors are rated by voltage, and therefore as a first point the capacitors used in the circuit must be able to withstand greater than the charging voltage applied to them. The capacitor size (in Farads) specifies how much charge can be stored internally to the resistor. Larger capacitors can store more charge, therefore sustaining longer-lasting output pulses than smaller value capacitors. The drawback is that the recharge time of large capacitors is also longer. It is also important to consider the pulse rise time (usually rated in volts per

microsecond). Larger capacitors also tend to have lower pulse rise times. In a Marx generator configuration, the largest effective capacitance is desired that can still meet charging voltage and pulse rise time requirements.

By knowing the value of the load, the number of stages, and the charging voltage, the current through the capacitors when rectified to a series configuration can be calculated simply through Ohm's Law based on the value of the load. Using on the current flow through the capacitors, the pulse rise time can be calculated. Since

$$I = C * \frac{dV}{dt}$$

then the pulse rise time can be found by simple re-arrangement.

$$\frac{dV}{dt} = \frac{I}{C}$$

Expressing this in volts per microsecond allows for easy comparison to component datasheets.

2.2.3 IGBTs

In this project, we were required to use IGBTs, so I will only discuss the general selection guidelines for IGBTs. However, Power MOSFETs function very similarly and many of the same rules apply for selection of a Power MOSFET device. Key items in IGBT selection are collector-to-emitter maximum current flow, gate threshold and turn on / saturation voltages, and switching time. The IGBT should be able to withstand current flows greater than the maximum currents required by the circuit switches. Depending on the gate driver (discussed next), the gate threshold and saturation voltages must also be considered. Switching time of the IGBT should be fast enough to not distort any output

pulse generated. Other considerations in IGBT selection include gate drive currents and negative gate drive capabilities.

2.2.4 Gate Drivers

Selection of a gate driver goes together with selection of an IGBT. The IGBT's turn on and saturation gate voltages dictate important characteristics of the gate driver, as do the IGBT's switching speed, gate drive currents, and negative gate drive ability. The driver should be able to output sufficient voltages applied between gate and emitter so as to drive the IGBT into an ON state in saturation. It must do this with switching speed capabilities equal to or faster than those of the IGBT. When enabled, the gate driver must also be able to provide enough initial gate drive burst current to the gate of the IGBT. Although the IGBT is a voltage-controlled device, in order to quickly turn on the transistor, the gate driver must be able to quickly deposit charge at the gate to create the potentials needed to turn on the device. If the IGBT has the capability, the driver should also be able to supply negative voltages to the gate during the OFF state. This is to help pull charge out of the IGBT gate, expediting turn off speed and preventing any uncoordinated turn-ons. It is also important to consider the microcontroller used, and whether the logic level supplied is compatible with the gate driver logic levels.

Finally, it is crucial in a Marx configuration that the IGBT gate driver have isolation between logic side and power side, and utilize an isolated power source so as to create an isolated reference point at each IGBT emitter. This is because the gate turn on voltage is referenced to each IGBT's emitter node, which may or may not be directly connected to the main ground.

2.2.5 Microcontroller

The microcontroller should be chosen to interface with the gate drivers. It should have sufficient processor speed and a sufficient number of output pins to control the drivers and interface with any other devices like control buttons or displays. Perhaps it is also desired to have timer, interrupt, and UART capabilities. Maybe it is desired to use a familiar logic controller to speed up the embedded software development. The requirements for a microcontroller can vary greatly between projects so it is important to consider the "big picture" of what the project will be designed to do and make sure the microcontroller has the capability to be the logical brain of the product.

2.2.6 Voltage Regulation

In order to power components like a 5V or 3.3V microcontroller, gate drivers, or other peripherals from a single power supply, voltage regulation components must be used. Linear voltage regulators might be used depending on the range of voltages. Buck convertor modules may be used instead of linear voltage regulators for greater efficiencies in stepping down power supplies with voltages well above peripheral demands. Isolated DC-DC convertors (or AC-DC, depending on power supply) should be used to power the gate drivers and create isolation between IGBT emitter grounds and the main power supply ground. The isolated DC-DC converters should ideally supply the gate drivers with both positive IGBT on state voltages and negative off state voltages.

2.2.7 Blocking Diodes

Blocking diodes should be used at the input to the charging circuit to resist reverse currents and biases that may be created in the circuit and block unwanted current backflow into the power supply. An ideal blocking diode blocks more than the peak pulse voltage,

allows for high forward current flow to charge the capacitors, and has a minimal forward voltage drop. Switching times are effectively irrelevant since the circuit is always charging.

2.2.8 Power Supply

The power supply chosen should convert the 120VAC 60Hz power from a typical US wall connection to a DC voltage level that can power all the power consuming components in the circuit. Assume the capacitors are not charged when calculating the power supply current needed at any given time.

CHAPTER 3

FINAL DESIGN

3.1 Final Circuit Details

The final design for the five-stage solid-state 50V, 2500W, 100us output pulse Marx generator, using IGBTs, is the result of simulation, breadboard prototyping, testing, more prototyping, and fabrication. A major issue came up during initial breadboard testing when our team realized we did not have isolation for our gate drivers. Therefore, the gate-emitter voltage output intended by the drivers was instead erroneously referenced from gate to ground. This was noted in the previous chapter, however it is an important point worth re-iterating: the gate drivers should have isolation, and be powered from an isolated power source (like an isolated DC-DC converter) to create emitter references that are distinct from the main ground.

3.1.1 Schematic

A block schematic of the final circuit design is shown below. The 24VDC supplied by the power supply is dropped to 5V to power the microcontroller. The microcontroller uses 5V logic, and interfaces with the isolated gate drivers, which are powered by +15/-15 V isolated DC-DC converters that also draw power from the 24 VDC power supply. The gate drivers connect to the gate and emitter of each respective IGBT in the circuit for ON/OFF control. Another buck converter draws from the power supply and outputs roughly 12V, through a switch, then through a blocking diode (2V forward voltage drop) to charge the capacitors at 10V. If the user toggles the switch, they may supply their own

charging voltage to the capacitors at up 100VDC through a screw terminal connection. This path is also protected by a blocking diode. When the IGBTs are quickly turned on for 100us and then back off, the resulting pulse is output to a test load (specified as 50 ohms) which is also connected externally via a screw terminal. Although not labeled, the resistors are 10kΩ 7W rated, and the capacitors are 0.22uF. Also not shown is the LCD and buttons used to interface with the microcontroller and display pulse information.

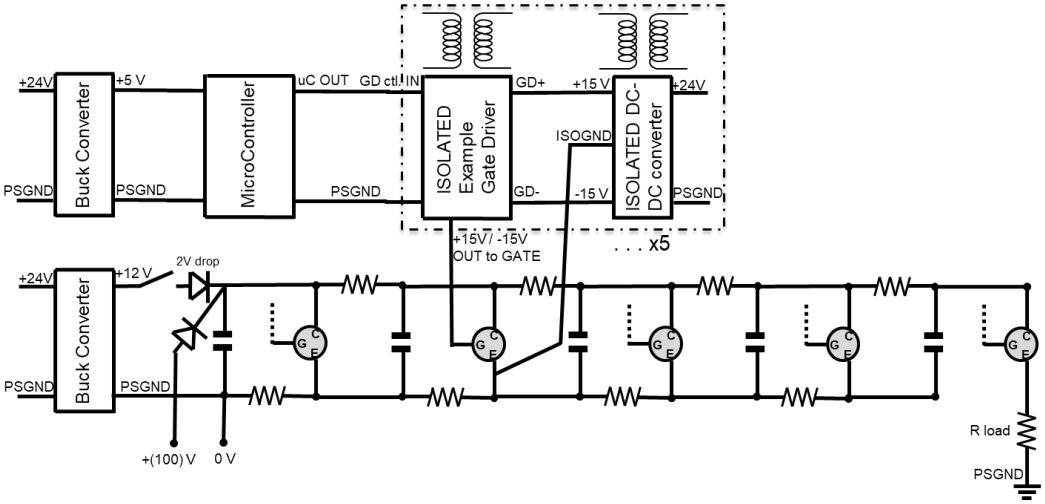

Figure 3.1: Block Schematic of Final Circuit Design

3.2 Final Parts List

3.2.1 Resistors

The resistors chosen are 10kΩ, 7W resistors capable of handling pulse currents of 50A or greater, as specified. Our team arrived at these values by simulation, conducted by team member Hayden Atchison.

3.2.2 Capacitors

The capacitor sizes were also calculated using simulation. Our team settled on 0.22uF capacitors. Because there could be 50A or more at any moment according to our specifications, the capacitors needed to sustain at least 230V/us as according to the earlier

equation. We chose the WIMA MKP1F032204D00JD00 model for its low cost, DC voltage rating of up to 250V, and superior pulse rise time of 650V/us. In retrospect, given our pulse rise time requirements it may have been very possible to use larger capacitors that had comparable ratings but could sustain longer-lasting output pulses. As seen in the results discussed in a later chapter, the resulting pulse achieved decays quickly, and were we to start again, I would advocate for larger capacitors. However, our team's choice was accurate in meeting project specifications.

3.2.3 IGBTs

At roughly $10 per part, the FGY40T120SMD IGBTs chosen were slightly more expensive than other comparable IGBTs we found, but had a much higher power rating of 882W. With maximum collector current rated at 80A that allows for thermal overhead, well over the 50A requirement, and 500ns switching speeds, they were ideal for our application. The gate-emitter voltage on state drive range is between 7.5V and 25V, and the maximum off state gate-emitter voltage is -25V, which works perfectly with the +15/-15V output by the chosen gate driver. The chosen IGBT can have a minimal up to 2V loss across collector to emitter while on.

3.2.4 Gate Drivers

Originally, we used a HCNW3120 IGBT gate driver, but during breadboard testing, it was discovered that this driver was not isolated and thus the IGBTs were not getting an appropriately referenced 15V gate-to-emitter to fully turn on. Therefore, a switch was made to an Infineon 1EDC20H12AH gate driver. This isolated driver was capable of drawing from the 24VDC power supply, being controlled by a 5V logic microcontroller, and

outputting an isolated +15V and -15V in on and off states to the IGBT gate. In addition, this gate driver was very affordable.

3.2.5 Microcontroller

More will be discussed about the microcontroller, its code, and the ability to control pulse length and shape in the next chapter. For now, it should be mentioned that a Microchip PIC16LF15356 was chosen for its 5V logic ability, availability of timers, interrupts, and number of output pins. However, this chip would have been difficult and much more time consuming to program so a DM164143 Evaluation Board was chosen instead. This board uses a PIC16F15376, a chip from the same family but with more I/O pins and some extra features. The advantage of the evaluation board is that it is prefabricated to connect to a computer via USB connection for instant programming.

3.2.6 Voltage Regulation

Buck convertors with input ranges of 3V to 40V and output ranges of 1.5V to 35V at up to 3A were ordered in order to power the 5V microcontroller and supply the 10V charging voltage at up to 2A to the capacitor stages. These buck converters were ordered from Amazon.com and do not list a model number. However, they are listed as using LM2596S-ADJ surface mounted buck modules and SANYO solid-state capacitors.

For the gate driver power, Murata MEA1D2415SC model isolated DC-DC converters were used. These could be supplied with an input voltage of 21.6V to 26.4V, perfect for the 24VDC power supply used in our design. They have a dual output voltage of +15V and -15V, with up to 33mA and -33mA respective output currents, for an output power up to 1W.

3.2.7 Blocking Diodes

Because a user can choose to supply an external charging voltage of up to 100VDC, and therefore with the 5-stage design generate pulses up to 500V, the blocking diodes needed to block at least that much reverse voltage. The Rectron FR257-B diodes chosen are rated to block double that, up to 1000V reverse voltage, while having a high forward current rating of 2.5A to allow fast capacitor charging. They also hold a minimal typical forward voltage drop of 1.3V.

3.2.8 Power Supply

Choosing a power supply was a relatively simple task after choosing the components. A power budget was made to estimate the power draw that the circuit would consume. The power supply would need to generate more than that amount of power. In our case, assuming a low 75% efficiency for the buck converters, and the rated 85% efficiency of the isolated DC-DC converters, with a 24VDC power supply shows that a 3A power supply would be sufficient, with 426mA of unused current draw. To be safe, we used a 24VDC, 4A AC-DC power supply adapter. This interfaced with our circuit board via a 2.5mm x 5.5mm power jack.

Table 3.1: Estimated Power Budget

Part	Quantity	Voltage (V)	Current (mA)	Efficiency	Eff. Losses (mW)	Power (mW)
Input to Microcontroller Buck Converter	1	5	-400	75%	-	-2000
Output from Microcontroller Buck Converter	1	5	400	-	-500	1500
Microcontroller	1	5	-300	-	-	-1500
LCD	1	5	-5	-	-	-25
Input to Cap. Stage Buck Converter	1	12	-2222	75%	-	-26666
Output from Cap. Stage B.C. (after diode)	1	10	2000	-	-6666	20000
All 5 Capacitor Stages	1	10	-2000	-	-	-20000
Input to DC-DC converter	5	24	-47	85%	-	-5640
Output from DC-DC converter MAX		15	33	-	-345	2475
		-15	-33	-	-345	2475
Power supply	1	24	4000	-	-	96000
Current and Power Margin			**1426**			**66619**

3.2.9 Other Parts

Two Phoenix Contact 1731721 model screw terminals were used as connections for the external voltage and the load attachment. Rated for 630V and 24A, these terminals were ideal. A Mountain Switch model 108-MS550A switch was utilized to toggle between the default 10V charging voltage and an external input charging voltage. Because of the blocking diode, the switch was only required to block the 10V from the charging buck converter. Rated at 125VAC, this was no problem. Lastly, an LCD scrapped from a previous electrical engineering class (equivalent to a NMTC-S16205DRGHS-07) was used with a 10kΩ potentiometer in combination with N-RBB model OFF-(ON) pushbuttons and 820Ω pulldown resistors to allow for interaction with the microcontroller and display of information.

3.2.10 Complete Parts List

A complete parts list is shown in Appendix A. The cost is just under $200 for all components. This estimate does not include PCB fabrication costs.

3.3 Printed Circuit Board

3.3.1 PCB Schematic

After finalizing testing, a PCB schematic was created by team member Hayden Atchison in Autodesk Eagle PCB design software. It interfaces and connects the components as desired. A photo of the annotated schematic is shown in Appendix A.

3.3.2 Board Layout

From the schematic, a two-layer board design was also conducted in Eagle by Hayden. In the board layout, the dimensions of the parts are to-scale with a reference grid, and “traces” are routed to electrically connect all components. The traces within the Marx

circuit were made thicker to handle the 50A or greater currents that could be flowing at a given moment. This board layout is also shown in Appendix A, just like the schematic. After I helped Hayden review and fix some key connections on the board, due to time restrictions we were unable to re-design and re-fabricate the PCB for our final project demonstrations. So, there is one important detail still incorrect. In the board layout, there is one missing 10kΩ resistor. It should connect between the collector of the fourth IGBT and the collector of the fifth IGBT. In the picture shown in Appendix A, this means the resistor should jump the middle pin of Q4 (node N$11, second IGBT from right) to the middle pin of Q6 (the IGBT to the rightmost side). In our project we did this by a quick appended soldering job on the back of the board.

CHAPTER 4

MICROCONTROLLER

4.1 Background

I was in charge of the microcontroller portion of this senior design project, and was therefore responsible for correct programming, timing on/off outputs, and interfacing. Our project specifications listed that the output pulse must be 100us long, but did not list how this could be achieved. Although timing could be done with a simple signal generator, the addition of a functional microcontroller allowed for project portability, and pulse timing flexibility. Able to clock at 32MHz, the microcontroller processor was capable of effectively turning on or off any output instantaneously, and with multiple outputs was able to control each of the 5 IGBT gate drivers individually, a task impossible for a single or double output laboratory signal generator. To go above and beyond the project requirements, I decided to see if we could perform pulse shaping by staggering the on and off timers of the five unique IGBT stages. I also ensured the user could cycle through different pre-programmed pulse “profiles”, which would be displayed to the LCD screen.

4.2 Programming and Testing

A complete code listing is held in Appendix B. Small excerpts of this code are included in the relevant sections below for the sake of space and brevity of explanation.

4.2.1 Configuration of Oscillators and Timers

To configure the oscillator, a "config_OSC" function was written. This function is called by main to change oscillator settings. The oscillator selected is the High Frequency Internal Oscillator, HFINTOSC, configured to run at 32 MHz with a divider ratio of 1:1.

```
void config_OSC (void) {
    OSCCON1 = 0x00;
    OSCCON2 = 0x00;
    OSCCON3 = 0x00;
    OSCCON1bits.NOSC = 0x6;
    OSCFRQbits.HFFRQ = 0x6;
    OSCCON1bits.NDIV = 0x0;
}
```

The timers are configured so that Timer 0 is used for a 5ms delay used for LCD function and debouncing, and Timer 2 is used to create a 50us delay for LCD function and can be used for as low as 12us for pulse timing. Timer 1 is unused.

Timer 0 is used in 16-bit mode, synced to the 32 MHz HFINTOSC clock. In accordance with a pre-scaler and post-scaler, TMR0L and TMR0H values are calculated to trigger a rollover interrupt roughly every 5ms.

```
void config_T0(void) {
    T0CON0 = 0x0;
    T0CON1 = 0x0;
    T0CON0bits.T0EN = 0;
    T0CON0bits.T016BIT = 1;
    T0CON1bits.T0CS = 0x3;
    T0CON1bits.T0ASYNC = 1;
    T0CON0bits.T0OUTPS = 0x2;
    T0CON1bits.T0CKPS = 0x0;
    TMR0L = 0xC4;
    TMR0H = 0x9;
    PIR0bits.TMR0IF = 0;
    PIE0bits.TMR0IE = 1;
}
```

Timer 2 is programmed similarly. A clock source of FOSC is chosen with pre-scale and post-scale values used to cause a Timer 2 interrupt every 12us. By utilizing five of these Timer 2 interrupts, a ~50us long delay of "LCD_short_delay" can be created.

```
void config_T2(void) {
    T2CON = 0x0;
    T2CLKCONbits.CS = 0x2;
    T2CONbits.ON = 0;
    T2CONbits.CKPS = 0x6;
    T2CONbits.OUTPS = 0x2;
    T2PR = 0;
    PIR4bits.TMR2IF = 0;
    PIE4bits.TMR2IE = 1;
}
```

4.2.2 LCD Code

The "LCD_init" function was written to initialize the LCD upon startup. This function utilizes the "LCD_long_busy" function, which uses Timer 0 for a 5ms delay, and "LCD_short_busy" which uses Timer 2 five times in succession to create a 50us delay. Upon startup, the LCD must be sent a specific set of commands, with appropriate timing in between commands, in order for the LCD to be initialized and be ready to display messages. This information is described on page 45 of the HD44780U LCD driver datasheet.

The "LCD_profileDisplay" function makes use of the profile information stored in an array of different pulse profiles and displays this information to the screen. This information is the length of the pulse and the number of times the pulse repeats with a time in between. As the user cycles through the profiles using the Profile button, these details are displayed to the screen.

4.2.3 Button Interrupt Code

The button interrupt code is multifaceted. There is the Interrupt Service Routine, which determines whether the interrupt was caused by a change to an input pin or just a timer, but when an input change does occur, variable "buttonIntention" is toggled on. The main function is always checking the state of "buttonIntention", and when it is set, the if-statement in the while loop evaluates as true, and an action is called depending on which

button was pressed. For example, if the Profile button was pressed, the “profileShift” function is called, which cycles to the next profile stored and displays the new pulse profile details to the screen. If the start button is pressed, the “start” function is called setting the state of “programStarted” as 1, displaying to the LCD, enabling the IGBTs, waiting the length of the pulse specified by the profile, then disabling the IGBTs and clearing the “programStarted” variable.

4.3 Component Level Testing

Programming and testing must go hand in hand. For the code that was written, each part was tested with hardware to confirm functionality and do debugging. Because software debugging was unavailable due to the nature of the code, an output test pin to an oscilloscope was used to confirm functionality in many cases.

4.3.1 Timer Output

By configuration, the timer interrupts were to fire at specified times after being enabled. By toggling the test pin within the interrupt service routine, each timer could be individually tested to output the desired on-off frequency. The following test code was used for the ISR and main function.

```
void __interrupt() isr(void) {
     if(TMR0IF == 1){
         T0CON0bits.T0EN = 0; //STOP Timer0
         TMR0 = 0; //Clear Timer0 counter value
         TESTPIN = !TESTPIN;
         TMR0IF = 0; //Clear Timer0 interrupt flag
         T0CON0bits.T0EN = 1; //START Timer0
     }
     if(TMR2IF == 1){
         T2CONbits.ON = 0; //STOP Timer2
         TMR2 = 0; //Clear Timer2 counter value
         TMR2IF = 0; //Clear Timer2 interrupt flag
         TESTPIN = !TESTPIN;
         T2CONbits.ON = 1; //START Timer2
     }
 }
```

```
void main(void) {
    INTCONbits.PEIE = 1;
    INTCONbits.GIE = 1;
    config_IO();
    config_CN();
    config_OSC();
    config_T0();
    config_T2();

    //Enable desired timer
    T1CONbits.ON = 1;
    //T2CONbits.ON = 1;

    while (1) {}
    return;
}
```

When each timer was enabled, a toggling of the test pin was correctly observed on an oscilloscope verifying that the timers functioned as desired.

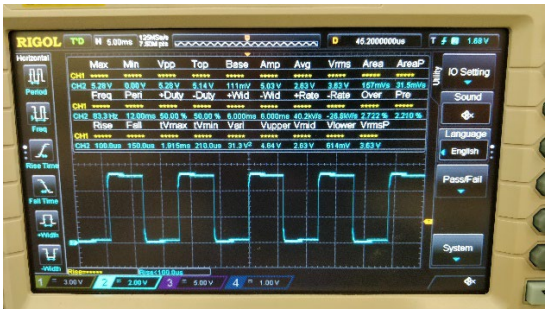

Figure 4.1: Working 5ms Output Toggle

Figure 4.2: Working 12us Output Toggle

4.3.2 LCD Function

After successfully testing the timers, next was debugging the LCD functionality. In this case, the test pin was used within the main function after all configuration function calls to confirm that all of the LCD initialization and setup (which used the timers) completed successfully. When the LCD failed to display a test message successfully, it was found that the initialization function was incorrect and the test message was being sent before the LCD was fully on. Because the screen was dark, a 10kΩ potentiometer was used to configure the contrast setting to no avail. But after fixing this initialization issue, the

LCD was able to display the test message and confirm the LCD function, and the potentiometer was used to set the appropriate contrast.

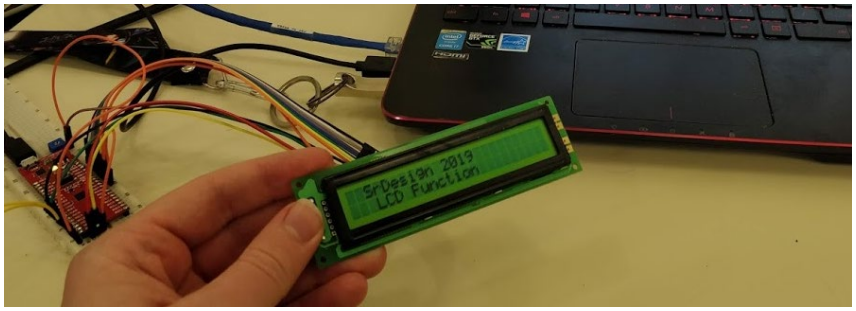

Figure 4.3: Working LCD Display

4.4 Interface Board

4.4.1 Design

In order to easily interface the LCD, buttons, and the main Marx generator PCB which would hold the microcontroller, I designed and manufactured a simple breakout board. The LCD connections for data and power, as well as the contrast potentiometer are on board in addition to the buttons and pulldown resistors. This board was milled on campus in the Electrical Engineering makerspace in Nedderman Hall.

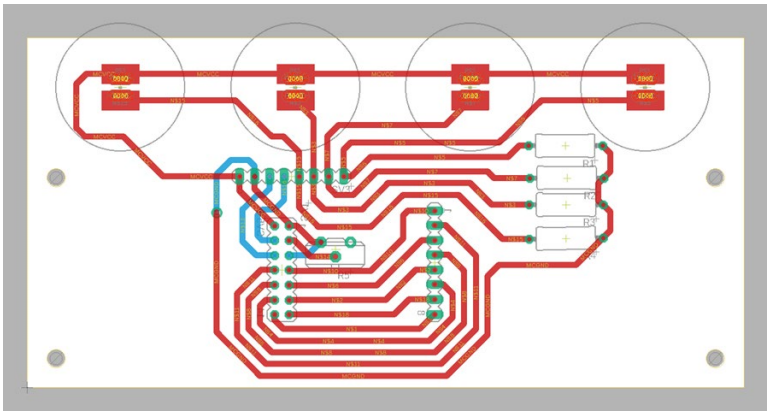

Figure 4.4: LCD and Button Board Layout

Figure 4.5: Top View of LCD and Button Breakout Board

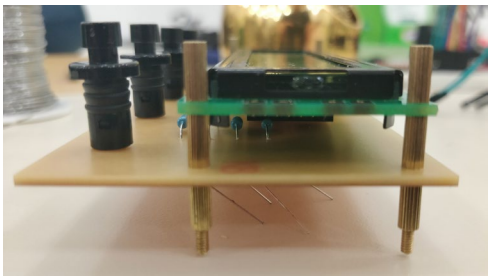

Figure 4.6: Side View of Board Showing Button Height and LCD Mounting

4.4.2 Testing

After soldering the board together, it was tested for shorts. Some time was spend debugging a short across the buttons, before confirming that the microcontroller was mounted on a faulty breadboard. Then, upon confirming that the board was wired as intended, the microcontroller was connected to the board. The button input and LCD display were tested and confirmed working.

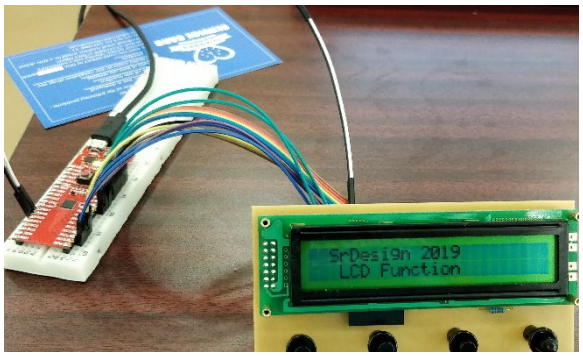

Figure 4.7: Testing of LCD with Microcontroller Mounted on Breadboard

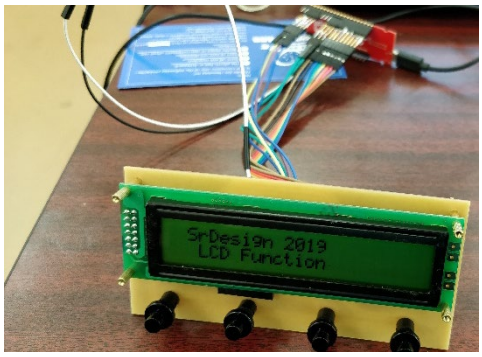

Figure 4.8: LCD and Button Function After Removing From Faulty Breadboard

CHAPTER 5

FINAL TESTING

5.1 Breadboard

The complete circuit setup had been previously built and tested on a breadboard. However, results were rough because the timing of the IGBTs was not precise. By interfacing the microcontroller with the gate drivers, concise on and off driver pulses were observed. Then, a 100us 50V pulse was correctly generated. Unfortunately, due to the size of the capacitors, the output pulse decayed quickly across the 10kΩ test load. Using larger capacitors would increase the sustain and allow for longer pulses.

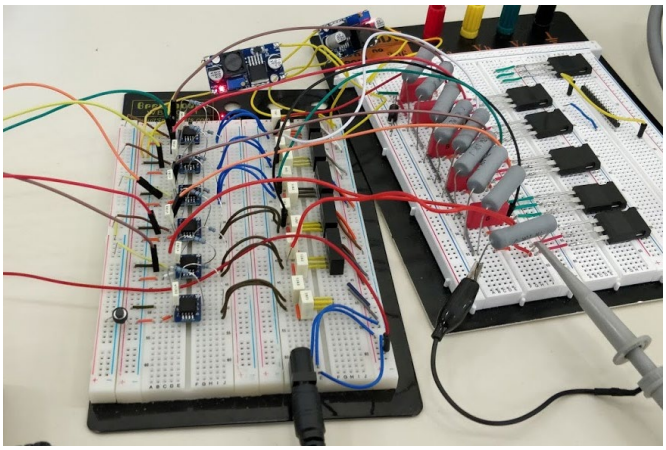

Figure 5.1: IGBTs, Isolated Gate Drivers, Isolated DC-DC Converters, and Buck Converters and Their Associated Components Mounted on a Breadboard for Testing

Figure 5.2: 100us Long Driver Pulse

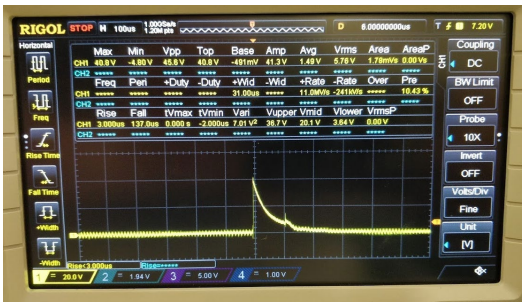

Figure 5.3: 100us Long Output Pulse

5.2 Main PCB

After confirming the function of the microcontroller together with the main circuit, and successfully generating a 50V 100us pulse as specified for the project requirements, the components were soldered to the PCB. As noted in an earlier chapter, an 8th resistor was added between the collector nodes of the 4th and 5th IGBTs. The microcontroller on the main board was connected to the LCD and button breakout board via male-to-female jumper wires. LCD, button, and pulse output function were tested and confirmed to be working as intended after moving the components from the breadboard to the PCB.

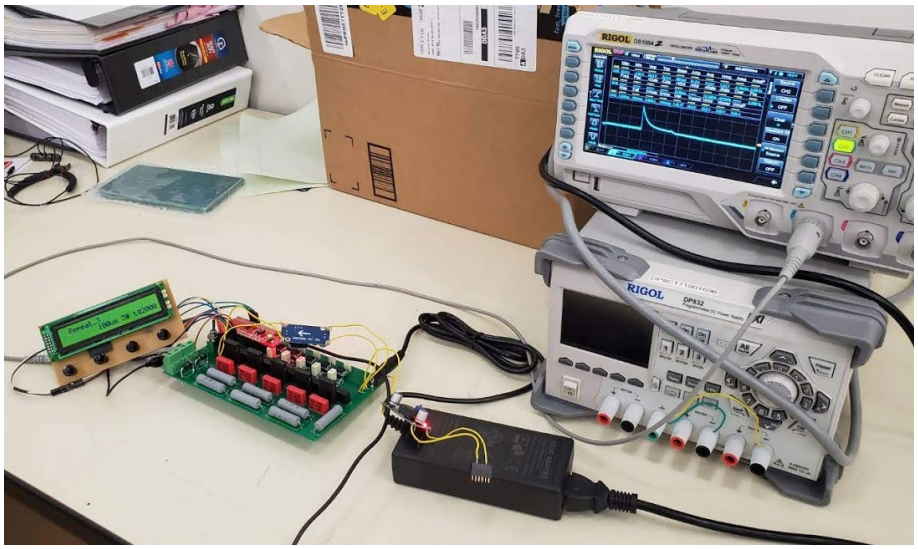

Figure 5.4: LCD and Button Board, Main Marx Generator Board and Power Brick are Shown During Final Project Presentations and a Successful Pulse is Shown on the Oscilloscope

5.3 Pulse Shaping

The “increment” and “decrement” functions of the microcontroller code are used to demonstrate the rough pulse shaping ability of the Marx generator circuit. By toggling different stages on asynchronously, a sort of shaping can be achieved. It is not an ideal way to achieve a shaped pulse, because of the resistance between stages. See Figure 5.6. As an example, when the fifth control IGBT switch is closed (right side dotted line), there is a path through the load to ground. And then when the first IGBT switch is closed, capacitor 1 and capacitor 2 are connected in series. By triggering just this first stage, a voltage pulse of 1x the charging voltage is generated across the load, although there will be some power losses through the path of resistances demarked by the dashed arrow.

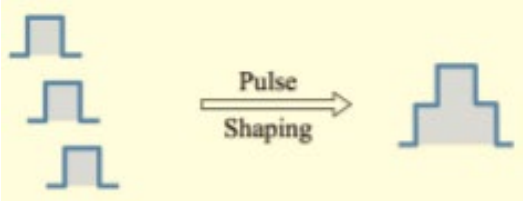

Figure 5.5: Pulse Shaping [5]

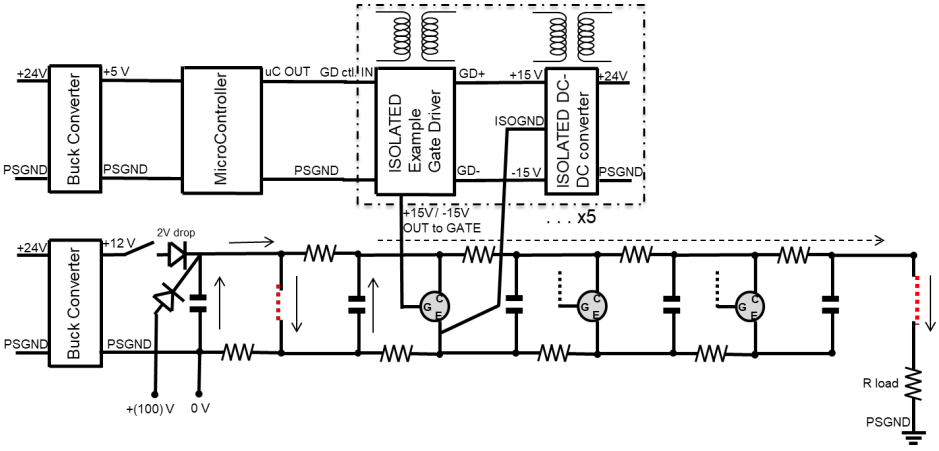

Figure 5.6: Example of Marx Generator Triggering Only One Stage

In applications requiring true pulse shaping, typically it is achieved with a pulse forming network, Linear Transformer Driver (LTD) [5], or other pulse power generator that does not have this drawback. Individual Marx generators by themselves are by design made to trigger all stages simultaneously. A better way to achieve a desired pulse shaping using Marx generators would be to make an array of a number of these 5-stage Marx generators, and trigger each Marx generator asynchronously and have the output pulses sum into the same load. However, due to time restrictions, only one solid-state Marx generator board was fabricated. Even still, asynchronously triggering the stages of a single Marx generator does generate some interesting results.

When the decrement function is called, shape 1 is displayed, and when the increment function is called, shape 2 is displayed. Shape 1 is composed by the following:

```
marx5 = 1;
marx1 = 1;
pulseTimer12us();
marx1 = 0;
marx2 = 1;
pulseTimer12us();
marx2 = 0;
marx3 = 1;
marx4 = 1;
marx5 = 1;
pulseTimer12us();
pulseTimer12us();
pulseTimer12us();
marx3 = 0;
marx4 = 0;
marx5 = 0;
```

which toggles pulse output IGBT (marx5) on, then turns on one stage for 12us, the next stage for 12us, then the remaining 3 stages on for 36us.

Shape two is composed by the following code which enables output, then turns on the first stage for 12us, and with the first stage still on, turns on the next two stages, which are on for 24us. The first stage is turned off, and the remaining stage is turned on. After 12us, stages 2, 3, and 4 are turned off.

```
marx5 = 1;
marx1 = 1;
pulseTimer12us();
marx2 = 1;
marx3 = 1;
pulseTimer12us();
pulseTimer12us();
marx1 = 0;
marx4 = 1;
pulseTimer12us();
marx2 = 0;
marx3 = 0;
marx4 = 0;
```

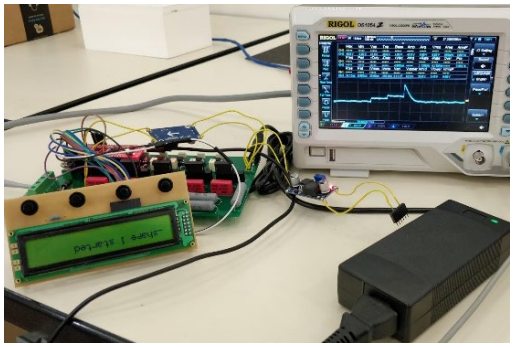

Figure 5.7: Shape 1 Displayed on Oscilloscope, with LCD Screen Displaying Pulse Name

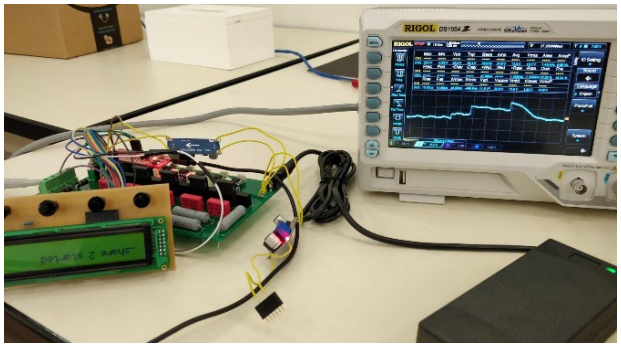

Figure 5.8: Shape 2 Displayed on Oscilloscope, with LCD Screen Displaying Pulse Name

Resulting oscilloscope outputs are shown in Figures 5.7 and 5.8. Although the shapes are rough, they demonstrate the pulse shape can very roughly be adjusted by toggling on and off different stages at different times. By doing this, although it is not ideal, the pulse height, length, and shape are all able to be adjusted. A pulse shaping design utilizing an array of multiple Marx generators would have more capability and would be available to generate smoother pulse shapes.

Figure 5.9: Shape 1 on Oscilloscope

Figure 5.10: Shape 2 on Oscilloscope

CHAPTER 6

CONCLUSIONS

6.1 Summary

Our team successfully designed, tested, and fabricated a 100us, 50V, 2500W capable solid-state Marx generator using IGBTs and isolated gate drive circuitry. The design also allows for a user to supply their own charging voltage up to 100V, and theoretically create a 500V output pulse. However, this function has not been tested.

Pulse shaping was successfully attempted and shown by triggering individual Marx generator stages asynchronously. It is noted that this is not an ideal form of pulse shaping, but it did exhibit interesting results. All in all, the project was a complete success and it met all project requirements. Additionally, by interfacing with the microcontroller, the project was capable of allowing a user to choose a pulse profile (pulse length, number of pulses, time between pulses) and attempt pulse shaping.

6.2 Future Work

If one were to continue this work, I would recommend using larger capacitors that are capable of the required pulse rise time. In this way, the capacitors would not fully drain in between charging and discharging, but instead perhaps only 10% to 30% of the capacitor would have to be recharged. Because they would store more charge, these larger capacitors would be able to sustain longer pulse times.

I would also like to see perhaps five or more identical boards of the 5-stage solid-state Marx generator fabricated and working in tandem to generate higher power output

pulses and truly test pulse shaping abilities. Lastly, I would fix the missing resistor from collector of IGBT 4 to collector of IGBT 5 in the PCB board design.

APPENDIX A

DESIGN DETAILS

Table A.1: Complete Parts List

Part Name	MPN	Cost EA	Total Qty	Cost Total
IGBT Gate Driver	726-1EDC20H12AHXUMA1	2.71	5	13.55
3.3Ω Resistor	660-MF1/4DC3R30F	0.10	5	0.50
10Ω Resistor	660-MFS1/4DCT52R10R0	0.10	5	0.50
0.1uF Capacitor	80-R82DC3100DQ50J	0.30	5	1.50
1uF Capacitor	80-R82DC4100AA60J	0.32	10	3.19
DC-DC Converter	580-MEA1D2415SC	5.42	5	27.10
Toggle Switch	108-MS550A	2.15	1	2.15
2-connection screw terminal	651-1731721	2.45	2	4.90
1000V 2.5A diode	583-FR257-B	0.42	2	0.84
Buck converter	LM2596 DC to DC Buck Converter 6 pack	11.75	1	11.75
24VDC power brick	LEDENET-POWER-4001-2.5MM	24.98	1	24.98
WIMA 0.22uF capacitor	MKP1F032204D00JD00	0.49	5	2.45
10kΩ 7W resistor	-		8	0.00
IGBT	512-FGY40T120SMD	11.08	5	55.40
PIC16F15376 DM164143 microcontroller dev board	579-DM164143	11.97	1	11.97
LCD	[668-NC-S16205DRGHS]	4.28	1	4.28
10k potentiometer	[581-601030]	0.3	1	0.30
1/4 W 820Ω resistors	from personal kit	0.15	4	0.60
2x7 bent pin header	from personal kit	12.99	1	12.99
1x8 bent pin header	from personal kit	0	2	0.00
Pushbuttons	107-N-RBB	1.03	5	5.15
Power Jack	CP-059BH-ND	0.89	1	0.89
Total Cost				**184.99**

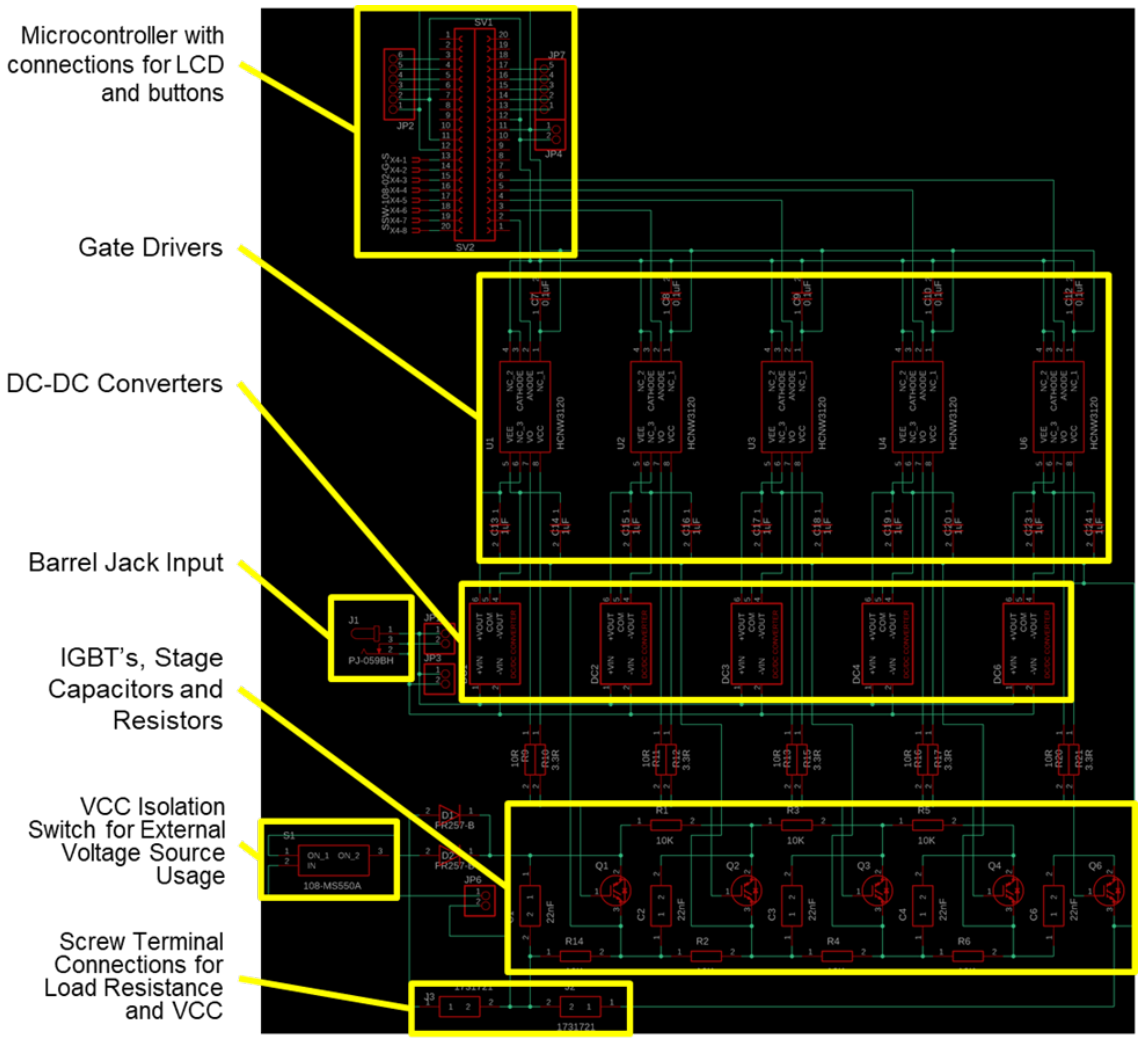

Figure A.1: PCB Schematic (Credit: Hayden Atchison)

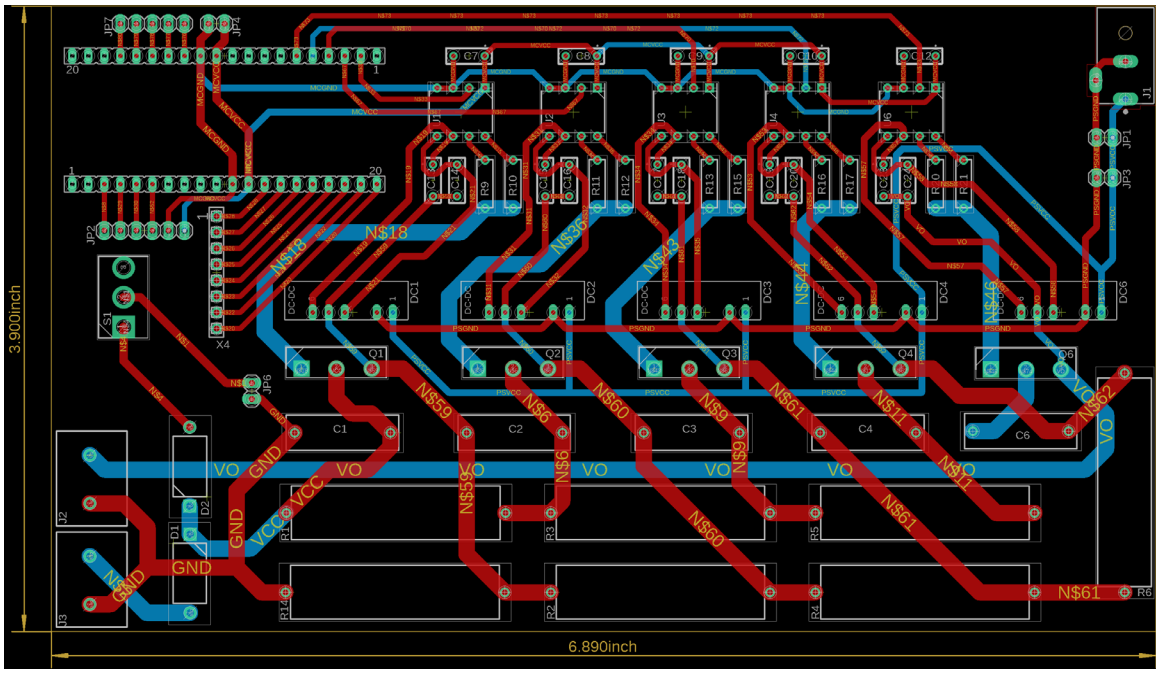

Figure A.2: PCB Board Layout (Credit: Hayden Atchison)

APPENDIX B

MICROCONTROLLER DETAILS

Microcontroller Code Listing

```
//
=============================================================================
===
// Information
//
=============================================================================
===

/*
 * File:        MarxGenMain.c
 * Author:      Ben Barnett
 * For:         Team 6, Marx Generator
 * Semester:    Spring 2019
 */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 * MARX GENERATOR
 *
 * --- Components: ---
 *  - LCD Display
 *      o  2 lines, 16 char. per line
 *  - Buttons
 *      o  Start and Stop
 *      o  Profile
 *      o  Increment
 *      o  Decrement
 *  - Microcontroller
 *
 * --- Functionality: ---
 *  - Allow user to set the PROFILE:
 *      o  The number of output pulses
 *      o  The length of time of each pulse
 *      o  The length of time between each pulse end and start
 *  - Display profile settings on the LCD
 *      o  Display profile setting changes as they occur on the LCD
 *  - Allow user to start and stop the system Profile output
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

//
=============================================================================
===
// Pragma statements
//
=============================================================================
===

// CONFIG1
#pragma config FEXTOSC = OFF    // External Oscillator mode selection
bits->Oscillator not enabled
#pragma config RSTOSC = HFINT32    // Power-up default value for COSC
bits->HFINTOSC with OSCFRQ= 32 MHz and CDIV = 1:1
```

```
#pragma config CLKOUTEN = OFF    // Clock Out Enable bit->CLKOUT
function is disabled; i/o or oscillator function on OSC2
#pragma config CSWEN = ON    // Clock Switch Enable bit->Writing to
NOSC and NDIV is allowed
#pragma config FCMEN = ON    // Fail-Safe Clock Monitor Enable bit-
>FSCM timer enabled

// CONFIG2
#pragma config MCLRE = ON    // Master Clear Enable bit->MCLR pin is
Master Clear function
#pragma config PWRTE = OFF    // Power-up Timer Enable bit->PWRT
disabled
#pragma config LPBOREN = OFF    // Low-Power BOR enable bit->ULPBOR
disabled
#pragma config BOREN = ON    // Brown-out reset enable bits->Brown-out
Reset Enabled, SBOREN bit is ignored
#pragma config BORV = LO    // Brown-out Reset Voltage Selection-
>Brown-out Reset Voltage (VBOR) set to 1.9V on LF, and 2.45V on F
Devices
#pragma config ZCD = ON    // Zero-cross detect disable->Zero-cross
detect circuit is disabled at POR.
#pragma config PPS1WAY = ON    // Peripheral Pin Select one-way
control->The PPSLOCK bit can be cleared and set only once in software
#pragma config STVREN = ON    // Stack Overflow/Underflow Reset Enable
bit->Stack Overflow or Underflow will cause a reset

// CONFIG3
#pragma config WDTCPS = WDTCPS_31    // WDT Period Select bits->Divider
ratio 1:65536; software control of WDTPS
#pragma config WDTE = OFF    // WDT operating mode->WDT Disabled,
SWDTEN is ignored
#pragma config WDTCWS = WDTCWS_7    // WDT Window Select bits->window
always open (100%); software control; keyed access not required
#pragma config WDTCCS = SC    // WDT input clock selector->Software
Control

// CONFIG4
#pragma config WRTC = OFF    // UserNVM self-write protection bits-
>Write protection off
//#pragma config SCANE = available    // Scanner Enable bit->Scanner
module is available for use
#pragma config LVP = ON    // Low Voltage Programming Enable bit->Low
Voltage programming enabled. MCLR/Vpp pin function is MCLR.

// CONFIG5
#pragma config CP = OFF    // UserNVM Program memory code protection
bit->UserNVM code protection disabled
//#pragma config CPD = OFF    // DataNVM code protection bit->DataNVM
code protection disabled


//
=============================================================================
===
// Import Header Files
```

```
//
================================================================================
===

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <xc.h>
#include <math.h>

//
================================================================================
===
// Define statements
//
================================================================================
===

// Outputs to Marx gate drivers
#define marx5       LATAbits.LATA0
#define marx4       LATAbits.LATA1
#define marx3       LATAbits.LATA2
#define marx2       LATAbits.LATA3
#define marx1       LATAbits.LATA4

// Outputs and inputs for LCD display
#define OC_RS       14739

// RB0-RB7 is the LATB that LCD screen is attached to and gets written
to
#define LCD_data    LATB

#define LCD_rs      LATAbits.LATA6
#define LCD_en      LATAbits.LATA7

// Inputs from buttons
#define startButton          PORTCbits.RC4
#define profileButton        PORTCbits.RC5
#define incrementButton      PORTCbits.RC6
#define decrementButton      PORTCbits.RC7


//#define _XTAL_FREQ 32000000
#define TESTPIN LATCbits.LATC0

//
================================================================================
===
// Global Variables
//
================================================================================
===

// ISR Toggle Variables (MUST be declared volatile)
volatile uint8_t programStarted = 0; // 0: Program not started, no
IGBTs on, no pulse   1: Program started, pulse output
```

```
volatile uint8_t T0_interrupt_counter = 0; // for Timer 0 interrupt
volatile uint8_t waitTMR0IF = 0;
volatile uint8_t waitTMR2IF = 0;
volatile uint8_t waitPulseTimer = 0;
volatile uint8_t waitLCD_long_busy = 0;
volatile uint8_t stateStart = 0;
volatile uint8_t stateShift = 0;
volatile uint8_t stateDecrement = 0;
volatile uint8_t stateIncrement = 0;
volatile uint8_t buttonIntention = 0;

// --- --- --- --- --- --- ---
// Profile Structure:
// --- --- --- --- --- --- ---
struct Profile {
    char name[15]; // Name of profile for displaying to screen later
    short numberOfPulses;  // Number of pulses that will occur for one
Profile run
    short lengthOfPulse;  // Length (uSec) of each pulse produced
    short timeBetweenPulses;  // Time (uSec) between consecutive pulse
outputs
};

// lengthOfPulse and timeBetweenPulses should be divisible by 50 (us)
for later function of prgm
struct Profile defaultP = {"Default", 1, 100, 0};
struct Profile repeat3 = {"Repeat_3", 3, 100, 2000};
struct Profile shortPulse = {"Short_50us", 1, 50, 0};
struct Profile shortPulseRepeat3 = {"Sh_50_R3", 3, 50, 2000};
struct Profile longPulse = {"Single_300", 1, 300, 0};
struct Profile longPulseRepeat3 = {"Sin_300_R3", 3, 300, 2000};
struct Profile custom = {"Custom", 1, 100, 0};

// Now it is an array of pointers to Profile structures
struct Profile *profileArray[] = {&defaultP, &repeat3, &shortPulse,
&shortPulseRepeat3, &longPulse, &longPulseRepeat3, &custom};
short currentProfile = 0;
short profileArrayLength = 7;


//
================================================================================
===
// Function Prototypes
//
================================================================================
===

void config_IO(void);
void config_CN(void);
void config_OSC (void);
void config_T0(void);
void config_T2(void);
void LCD_short_busy(void);
void LCD_long_busy(void);
void LCD_init();
void LCD_senddata(unsigned char var);
```

```
void LCD_sendstring(unsigned char *var);
void LCD_command(unsigned char var);
char* concat3(const char *s1, const char *s2, const char *s3);
void LCD_profileDisplay();
void start();
void profileShift();
void decrement();
void increment();
void pulseTimer12us();
//void interrupt isr(void);


//
================================================================================
===
// General Configuration
//
================================================================================
===

// --- --- --- --- --- --- ---
// Configure Inputs and Outputs
// --- --- --- --- --- --- ---
void config_IO(void) {
    // --- Set output for 5 gate drivers and LCD rs and en: ---
    TRISA = 0; // Set all Port A I/O to output
    LATA = 0; // Set all Port A outputs to LOW/0
    ANSELA = 0; // Turn Port A analog off (Digital only)

    // --- Set output for  LCD data: ---
    TRISB = 0; // Set all Port B I/O to output
    LATB = 0; // Set all Port B outputs to LOW/0
    ANSELB = 0; // Turn Port B analog off (Digital only)

    // --- Set input from buttons: ---
    //TRISC = 0b11111000;  // Set Port C 3-7 to input for the buttons
    TRISC = 0xF8;  // Set Port C 3-7 to input for the buttons
    LATC = 0;  // Set all Port C outputs LOW
    ANSELC = 0;  // Turn Port C analog off (Digital only)

    // NOTE: There are no internal pull-down resistors. Need them on
the PCB.
}

// --- --- --- --- --- --- ---
// Configure Change Notification for Button Interrupts
// --- --- --- --- --- --- ---
void config_CN(void){
    //Clear Interrupt flag
    IOCIF = 0;
    // Clear all of Port C 0-7 interrupt flags
    IOCCF = 0x00;

    // Enable interrupts on Positive Edge of the buttons RC3-RC7
    IOCCP3 = 1;
    IOCCP4 = 1;
    IOCCP5 = 1;
```

```
    IOCCP6 = 1;
    IOCCP7 = 1;

    IOCIE = 1; //Enable Interrupt
}

// --- --- --- --- --- --- ---
// Configure the oscillator:
// --- --- --- --- --- --- ---

//NOTE:  CLSWE clock switch enable, turn on to make sure clock can
switch
void config_OSC (void) {
    // Clear registers
    OSCCON1 = 0x00;
    OSCCON2 = 0x00;
    OSCCON3 = 0x00;

    // OSCCON1:
    // Use High Freq. Internal Oscillator (HFINTOSC @ 1 - 32 MHz)
    //OSCCON1bits.NOSC = 0b110;
    OSCCON1bits.NOSC = 0x6;

    // OSCFRQ:
    // Configure HFINTOSC to 32 MHz
    // OSCFRQbits.HFFRQ = 0b110;
    OSCFRQbits.HFFRQ = 0x6;

    // Divide clock by 1
    //OSCCON1bits.NDIV = 0b0000;
    OSCCON1bits.NDIV = 0x0;
}

// --- --- --- --- --- --- ---
// Configure the Timers
// --- --- --- --- --- --- ---

// Things needed:
// 50 uSec used for LCD Short Delay
// 5 mSec used for LCD Long Delay; Long Delay only used in startup
sequence
// Make 200 mSec delay for switch debouncing from LCD Long Delay
// ~5 msec, used for long LCD delay and button debouncing
void config_T0(void) {
    // Clear Timer Registers
    T0CON0 = 0x0;
    T0CON1 = 0x0;

    T0CON0bits.T0EN = 0; // Turn off Timer 0
    T0CON0bits.T016BIT = 1; // Operate Timer 0 as 16-bit mode
    T0CON1bits.T0CS = 0x3; // Select HFINTOSC 32MHz
    T0CON1bits.T0ASYNC = 1; // Sync Timer 0 to system clocks

    T0CON0bits.T0OUTPS = 0x2; // Postscaler (divider)
    T0CON1bits.T0CKPS = 0x0; // Prescaler

    // Time, 5 ms * Freq = 8 MHz  / Postscaler,16 = need 2500 ticks
```

```
    // TMR0H = 2500ticks_needed/256counting, rounded to integer
    //TMR0L is the remainder needed to get exactly 2500

    // Set TMR0L = 196, TMR0H =9
    TMR0L = 0xC4;
    TMR0H = 0x9;

    // Interrupts
    // Register 10-10 PIR0
    // 1: Interrupt occured 0: no
    PIR0bits.TMR0IF = 0; // Clear Interrupt Flag
    // Register 10-2 PIE0
    PIE0bits.TMR0IE = 1; // Timer0 overflow interrupt enabled
}

// 12 usec, used for ~50us LCD short delay and pulse timing
void config_T2(void) {
    // Clear Timer Registers
    T2CON = 0x0;

    //T2CLKCONbits.CS = 0b0010;
    T2CLKCONbits.CS = 0x2; // Select whatever Fosc is as Timer2 clock
    T2CONbits.ON = 0; // Turn off (and reset) Timer2
    T2CONbits.CKPS = 0x6; // Prescale
    T2CONbits.OUTPS = 0x2; // Postscale

    // PR does not matter, the timing is only based on the prescaler
and postscaler
    T2PR = 0;

    // Interrupts
    // Register 10-14 PIR4
    // 1: Interrupt occured 0: no
    PIR4bits.TMR2IF = 0; // Clear Interrupt Flag
    // Register 10-6 PIE4
    PIE4bits.TMR2IE = 1; // Timer2 to match PR2 value interrupt enable
}


//
================================================================================
===
// LCD Functionality
//
================================================================================
===

// --- --- --- --- --- --- ---
// LCD Delay Functions
// --- --- --- --- --- --- ---
// ~ 50 us delay
void LCD_short_busy(void) {
    int i = 0;
    for(i = 0; i<5; i++){
        T2CONbits.ON = 1; // Turn on Timer 2
        while(waitTMR2IF != 1);
        T2CONbits.ON = 0; // Turn off Timer 2
```

```
        TMR2 = 0; // Clear Timer 2 counter
        waitTMR2IF = 0; //Clear wait flag
        waitPulseTimer = 0;
        TMR2IF = 0; // Clear Timer 2 interrupt flag
    }
}

// ~ 5 ms delay
void LCD_long_busy(void) {
    // Set LCD flag to ON so interrupt knows whether to deal with LCD
    //  or with button presses on interrupt service routine
    waitLCD_long_busy = 1;
    // Clear timer flag
    PIR0bits.TMR0IF = 0;
    T0CON0bits.T0EN = 1; // Turn on Timer 0
    while(waitTMR0IF != 1);
    T0CON0bits.T0EN = 0; // Turn off Timer 0
    TMR0 = 0; // Clear Timer 0 counter
    waitTMR0IF = 0; //Clear wait flag
    waitLCD_long_busy = 0; // Clear LCD_long_delay indicator flag
    TMR0IF = 0; // Clear Timer 0 interrupt flag
}

// --- --- --- --- --- --- ---
// Initialize the LCD
// --- --- --- --- --- --- ---
void LCD_init() { //See page 45 of the HD44780U datasheet
    // Wait for more than 15 ms after Vcc rises to 4.5V
    // 15 ms delay
    int i;
    for (i = 0; i < 4; i++) { // Run 5 ms delay 3 times
        LCD_long_busy();
    }

    // FUNCTION set
    // 8-bit interface, 2 lines, 5x8 char font
    // 0b0000 0000 0011 1000
    LCD_command(0x0038);

    // Wait for more than 4.1 msec
    // Use 5 ms delay of LCD Long Busy at least once
    LCD_long_busy();
    LCD_long_busy();

    // FUNCTION set
    // 8-bit interface, 2 lines, 5x8 char font
    // 0b0000 0000 0011 1000
    LCD_command(0x0038);

    // Wait for more than 100 usec
    // Run 50 us LCD Short Busy delay at least twice
    LCD_short_busy();
    LCD_short_busy();
    LCD_short_busy();

    // FUNCTION set
    // 8-bit interface, 2 lines, 5x8 char font
```

```
    // 0b0000 0000 0011 1000
    LCD_command(0x0038);

    // Safe to wait
    LCD_short_busy();
    LCD_short_busy();

    // final FUNCTION set
    // 8-bit interface, 2 lines, 5x8 char font
    // 0b0000 0000 0011 1000
    LCD_command(0x0038);

    // Safe to wait
    LCD_short_busy();
    LCD_short_busy();

    // Display OFF
    // Cursor OFF
    // Blinking OFF
    LCD_command(0x0008); // 0b0000 0000 0000 1000

    // Add 37 usec delay for LCD_command(0x0008)
    // Use LCD Short Busy 50us
    LCD_short_busy();
    LCD_short_busy();

    // Clears entire display, Set DDRAM address 0
    LCD_command(0x0001);

    // Don't need a delay for this
    // Toss in 50 us to be safe
    LCD_short_busy();
    LCD_short_busy();

    // Entry Mode:
    // I/D = 1, increment by 1
    // S = 0, no shift
    // 0b0000 0000 0000 0110
    LCD_command(0x0006);

    // Need 37 us delay, use LCD Short Busy 50us
    LCD_short_busy();
    LCD_short_busy();
}

// --- --- --- --- --- --- ---
// LCD Data Sending Functions
// --- --- --- --- --- --- ---
void LCD_senddata(unsigned char var) { //the input will be an unsigned
character
    LCD_data = var; // Write character value to LCD_data

    LCD_rs = 1; // Enable writing to LCD
    LCD_en = 1; // Enable LCD
    LCD_short_busy(); // Add delay as specified in data sheet
    LCD_en = 0; // Disable LCD
    LCD_short_busy(); // Add delay
```

```
}

//NOTE: This function uses just "char" and not "unsigned char"
void LCD_sendstring(unsigned char *var) { // For sending a string of
text to the LCD
    while (*var) { // while there are characters left in the array
        LCD_senddata(*var++); // send each character to the LCD
    }
    LCD_short_busy(); // Added delay
}

// For sending character command to LCD
void LCD_command(unsigned char var) {
    LCD_data = var;
    LCD_rs = 0;
    LCD_en = 1;
    LCD_short_busy();
    LCD_en = 0;
    LCD_short_busy();
}

void LCD_profileDisplay(){
    char firstRow[20];
    strcpy(firstRow, "       ");
    char *firstRowStr = profileArray[currentProfile]->name; // Send the
name of the current profile to a string
    strcat(firstRow, firstRowStr);

    // Convert profile int number data to strings
    char numberStr[7];
    char lengthStr[7];
    char timeStr[7];

    sprintf(numberStr, "%d", profileArray[currentProfile]-
>numberOfPulses);
    sprintf(lengthStr, "%d", profileArray[currentProfile]-
>lengthOfPulse);
    sprintf(timeStr, "%d", profileArray[currentProfile]-
>timeBetweenPulses);

    char secondRow[20];
    strcpy(secondRow, "       ");
    strcat(secondRow, lengthStr);
    strcat(secondRow, "us ");
    strcat(secondRow, numberStr);
    strcat(secondRow, "# ");
    strcat(secondRow, "tg");
    strcat(secondRow, timeStr);

    LCD_command(0x0001); // Clear display
    LCD_short_busy();
    LCD_command(0x0002); // Return home
    LCD_short_busy();

    int AA = 0x0;
    LCD_command(AA); // Set DDRAM address
    LCD_short_busy(); // 50 us delay for previous command
```

```
    LCD_sendstring(firstRow); // Transfer string to the screen
    LCD_short_busy();

    int A = 0x0080 + 0x0040 + 0x0002; // 0b 0000 0000 1100 0000
    LCD_command(A); // Move to second line
    LCD_short_busy(); // Add 50 usec delay for previous command
    LCD_sendstring(secondRow); // Transfer string to the screen
    LCD_short_busy(); // 50 us delay for previous command
    LCD_command(0xC);
    LCD_short_busy();
}


//
=============================================================================
===
// Button Function
//
=============================================================================
===

void start(){
    programStarted = 1; // Change state of program to 1, started
    // Display to the LCD that program has started
    char startedWord[] = "      __started__";
    LCD_command(0x0001); // Clear display
    LCD_short_busy();
    LCD_command(0x0002); // Return home
    LCD_short_busy();

    int A = 0x0080 + 0x0040 + 0x0002; // 0b 0000 0000 1100 0000
    LCD_command(A); // Move to second line
    LCD_short_busy(); // Add 50 usec delay for previous command
    LCD_sendstring(startedWord); // Transfer string T to the screen
    LCD_short_busy(); // 50 us delay for previous command
    LCD_command(0xC);
    LCD_short_busy();

    int i = 0;
    for(i = 0; i < profileArray[currentProfile]->numberOfPulses; i++){
        // Turn on IGBTs
        marx1 = 1;
        marx2 = 1;
        marx3 = 1;
        marx4 = 1;
        marx5 = 1;

        // Wait length profileArray[currentProfile].lengthOfPulse;
        // If this is in 50s of uSec, can use a function to loop
        int i;
        int j = profileArray[currentProfile]->lengthOfPulse / 50;
        for (i=0; i < j; i++){
            LCD_short_busy();
        }
        // Turn off IGBTs
        marx1 = 0;
        marx2 = 0;
```

```
        marx3 = 0;
        marx4 = 0;
        marx5 = 0;

        // Wait length profileArray[currentProfile].timeBetweenPulses;
        // If this is in 50s of uSec, can use a function to loop
        int m;
        int n = profileArray[currentProfile]->timeBetweenPulses / 50;
        for (m=0; m < n; m++){
            LCD_short_busy();
        }
    }

    // Wait 50ms
    i = 0;
    for(i = 0; i < 10; i++){
        LCD_long_busy();
    }

    programStarted = 0; // Change state of program to 0, stopped
    stateStart = 0; // Reset start state after interrupt then main loop
called this function
}

void profileShift(){
    // If at the end of profile array
    if(currentProfile == (profileArrayLength - 1)){
        // Reset the profile to the first in array
        currentProfile = 0;
        // Display the current profile to LCD
        LCD_profileDisplay();
    } else {
        currentProfile++;
        // Display the current profile to LCD
        LCD_profileDisplay();
    }
    stateShift = 0; // Reset shift state after interrupt then main loop
called this function
}

void decrement(){
    programStarted = 1; // Change state of program to 1, started
    // Display to the LCD that program has started
    char startedWord[] = "      _shape 1 started";
    LCD_command(0x0001); // Clear display
    LCD_short_busy();
    LCD_command(0x0002); // Return home
    LCD_short_busy();

    int A = 0x0080 + 0x0040 + 0x0002; // 0b 0000 0000 1100 0000
    LCD_command(A); // Move to second line
    LCD_short_busy(); // Add 50 usec delay for previous command
    LCD_sendstring(startedWord); // Transfer string T to the screen
    LCD_short_busy(); // 50 us delay for previous command
    LCD_command(0xC);
    LCD_short_busy();
```

```
    // Turn on IGBTs
    marx5 = 1;

    marx1 = 1;
    pulseTimer12us();
    marx1 = 0;

    marx2 = 1;
    pulseTimer12us();
    marx2 = 0;

    marx3 = 1;
    marx4 = 1;
    marx5 = 1;
    pulseTimer12us();
    pulseTimer12us();
    pulseTimer12us();
    marx3 = 0;
    marx4 = 0;
    marx5 = 0;

    // Wait 50ms
    int i = 0;
    for(i = 0; i < 10; i++){
        LCD_long_busy();
    }

    programStarted = 0; // Change state of program to 0, stopped
    stateDecrement = 0; // Reset decrement state after interrupt then
main loop called this function
}

void increment(){
    programStarted = 1; // Change state of program to 1, started
    // Display to the LCD that program has started
    char startedWord[] = "      _shape 2 started";
    LCD_command(0x0001); // Clear display
    LCD_short_busy();
    LCD_command(0x0002); // Return home
    LCD_short_busy();

    int A = 0x0080 + 0x0040 + 0x0002; // 0b 0000 0000 1100 0000
    LCD_command(A); // Move to second line
    LCD_short_busy(); // Add 50 usec delay for previous command
    LCD_sendstring(startedWord); // Transfer string T to the screen
    LCD_short_busy(); // 50 us delay for previous command
    LCD_command(0xC);
    LCD_short_busy();

    // Turn on IGBTs
    marx5 = 1;

    marx1 = 1;
    pulseTimer12us();
    marx2 = 1;
    marx3 = 1;
    pulseTimer12us();
```

```
    pulseTimer12us();
    marx1 = 0;
    marx4 = 1;
    pulseTimer12us();
    marx2 = 0;
    marx3 = 0;
    marx4 = 0;

    // Wait 50ms
    int i = 0;
    for(i = 0; i < 10; i++){
        LCD_long_busy();
    }

    programStarted = 0; // Change state of program to 0, stopped
    stateIncrement = 0; // Reset increment state after interrupt then
main loop called this function
}


//
================================================================================
===
// Pulse Function
//
================================================================================
===

void pulseTimer12us(void) {
    T2CONbits.ON = 1; // Turn on Timer 2
    while (waitPulseTimer != 1); // Wait for interrupt single
    T2CONbits.ON = 0; // Turn off Timer 2
    TMR2 = 0; // Clear Timer 2 counter
    TMR2IF = 0; // Clear Timer 2 interrupt flag
    waitPulseTimer = 0;
    waitTMR2IF = 0;
}


//
================================================================================
===
// Interrupt service routine
//
================================================================================
===

void __interrupt() isr(void){
    // If button is pressed...
    if(IOCIF == 1 && buttonIntention == 0) {
        if (programStarted){ // Pulse profile output has started
already
            // Do nothing upon button push
        } else { // Pulse profile output has not yet started
            buttonIntention = 1; // Change state to indicate that
button was somehow pressed intentionally or unintentionally
        }
```

```
    }

    // ... 100 ms debouncing and determination of button pushes
    // Timer 0, used for debouncing wait time here
    if(TMR0IF == 1 && waitLCD_long_busy == 0){

        T0CON0bits.T0EN = 0; //Stop Timer0
        T0_interrupt_counter++; // increment the counter of interrupts
that have occurred

        if(T0_interrupt_counter >= 20) { // 5ms of T2 * 20 = 100ms have
passed
            // Configure the timer back to clean state for next use
            T0_interrupt_counter = 0; // Reset counter
            TMR0 = 0; //Clear Timer0 counter value
            T0CON0bits.T0EN = 0; //STOP Timer0
            TMR0IF = 0; //Clear Timer0 interrupt flag

            // Determine which button was pressed, and do appropriate
action
            if(startButton){
                stateStart = 1;
            }
            if(profileButton){
                stateShift = 1;
            }
            if(decrementButton){
                stateDecrement = 1;
            }
            if(incrementButton){
                stateIncrement = 1;
            }
        }
        else { // 100ms have not yet passed
            TMR0 = 0; //Clear Timer0 counter value so that timer0 can
resume counting
            TMR0IF = 0; //Clear Timer0 interrupt flag
            T0CON0bits.T0EN = 1; //Start Timer0 again
        }
    }

    // Timer 0, used for LCD_long_delay here
    if(TMR0IF == 1 && waitLCD_long_busy == 1){
        T0CON0bits.T0EN = 0;
        TMR0 = 0;
        TMR0IF = 0;
        waitTMR0IF = 1;
    }

    // Timer 2, used for LCD short delay and 12ms pulse timer
    if(TMR2IF == 1){
         T2CONbits.ON = 0; //STOP Timer2
         TMR2 = 0; //Clear Timer2 counter value
         TMR2IF = 0; //Clear Timer2 interrupt flag
         waitTMR2IF = 1;
         waitPulseTimer = 1;
    }
```

```
    IOCCF = 0x00;
    IOCIF = 0; //Clear CN Interrupt flag
}

//
=========================================================================
===
// Main function
//
=========================================================================
===

int main(void) {
    WDTCON0bits.SWDTEN = 0x0; // Ensure Watchdog Timer is totally
disabled

    // Register 10-1 INTCON
    INTCONbits.PEIE = 1; // Enable peripheral interrupt
    INTCONbits.GIE = 1; // Enable global interrupt

    // --- Call configuration functions: ---
    config_IO();
    config_OSC();
    config_T0();
    config_T2();
    // Initialize the LCD
    LCD_init();
    // Lastly, configure button interrupt ability
    config_CN();

    // --- Loop forever: ---
    while (1) {
        if(buttonIntention){
            TMR0 = 0; //Clear Timer0 counter value
            T0CON0bits.T0EN = 0; //STOP Timer0
            TMR0IF = 0; //Clear Timer0 interrupt flag

            buttonIntention = 0;
            waitLCD_long_busy = 0; // Just to be safe
            T0CON0bits.T0EN = 1; //START Timer0
        }
        if(stateStart == 1){
            start();
        }
        if(stateShift == 1){
            profileShift();
        }
        if(stateDecrement == 1){
            decrement();
        }
        if(stateIncrement == 1){
            increment();
        }
    }
    return 0;
}
```

REFERENCES

[1] E. Marx, “Versuche über die Prüfung von Isolatoren mit Spanningsstößen [Experiments on the Testing of Insulators using High Voltage Pulses],” *Elektrotechnische Zeitschrift (in German)*, vol. 25, p. 652–654, pub. 1924. *ISSN 0424-0200. OCLC 5797229*

[2] *Marx Generator Charging and Discharging States*, by ZooFari. Public Domain, https://commons.wikimedia.org/w/index.php?curid=8321326

[3] M. Savage et. al., “Pulsed power performance of the Z machine: ten years after the upgrade,” in *IEEE International Pulsed Power Conference, Brighton, England, June 18-22, 2017.* Available: https://www.osti.gov/servlets/purl/1463425

[4] N.A., “IGBT (Insulated Gate Bipolar Transistor),” *Rohm.com*, Available: https://www.rohm.com/electronics-basics/igbt/igbt

[5] W. Jiang, H. Sugiyama, A. Tokuchi. “Smart pulsed power by solid-state LTD”. IEEE Transactions on Plasma Science, Vol. 42, No. 11. Nov. 2014.

BIOGRAPHICAL INFORMATION

Benjamin Barnett graduates in December 2019 with an Honors Bachelor of Science in Electrical Engineering and a minor in Physics. During his time at UT Arlington, Ben has participated in several undergraduate research projects, ranging from applying machine learning to particle physics datasets, helping prototype a neutrino detector, and building a Farnsworth fusor. Although an electrical engineer by trade, Ben has an interest in engineering applied to physics and hopes to someday work in research and development of fusion reactors. In the meantime, Ben plans to take a long nap after graduation.