Three-phase Pulse Width Modulated AC/DC Rectifier and DC/AC Inverter

Carmen Ngo

ECE499: Capstone Design Project

Project Supervisor: Luke Dosiek

March 26, 2020

REPORT SUMMARY

Electric vehicles use inverter and rectifier circuits in order to convert power from one type to another. They run on direct current batteries, but their motors can run on direct current (DC) or alternating current (AC). There is a need for both of these circuits if the motor runs on AC. The objective of this project is to successfully replicate driving and braking in an electric vehicle, and understand its power flow. Both systems should start to convert the input power as soon as the source is "on" and should continue this conversion without interruption. In the driving system, a DC power supply will be the source and the inverter should convert that to three-phase alternating AC power to spin a three-phase AC induction motor, which should continue to spin. Sinusoidal pulse width modulation will be the technique used to control the MOSFETs in the inverter circuit. The three-phase AC signals should be 50Hz and 120 degrees phase shifted from one another. This induction motor should then be coupled to a DC machine and computer to control torque. In the braking system, the AC permanent magnet motor should be spun by the DC machine, which is controlled by the computer. This three-phase AC voltage should then be converted to DC voltage through the rectifier circuit to power a power resistor.

The results of this project include a successful driving system without the DC machine and computer combination to control torque, and a semi-successful braking system. The braking system needs real-time inputs that can be compared in the Arduino microcontroller, which requires the use of other components not mentioned in this report. The AC induction motor started spinning as soon as power was provided and continued without interruption. The frequency of each three-phase AC signal was 50Hz and they were 120 degrees phase shifted from each other.

TABLE OF CONTENTS

REPORT SUMMARY	**1**
TABLE OF CONTENTS	**2**
TABLE OF FIGURES AND TABLES	**3**
INTRODUCTION	**4**
BACKGROUND	**5**
DESIGN REQUIREMENTS	**10**
DESIGN ALTERNATIVES	**13**
PRELIMINARY PROPOSED DESIGN	**15**
FINAL DESIGN	**20**
FINAL IMPLEMENTATION	**26**
PERFORMANCE ESTIMATES AND RESULTS	**27**
PRODUCTION SCHEDULE	**30**
COST ANALYSIS	**33**
USER'S MANUAL	**35**
DISCUSSION, CONCLUSIONS, AND RECOMMENDATIONS	**37**
ACKNOWLEDGEMENTS	**39**
REFERENCES	**39**
APPENDICES	**41**

TABLE OF FIGURES AND TABLES

Figure 1: Block diagram of driving system 10
Figure 2: Block diagram of braking system 11
Figure 3: Top level block diagram of driving system 15
Figure 4: Top level block diagram of braking system 15
Figure 5: Pin diagram of MOSFET driver 15
Figure 6: Detailed block diagram of driving system 16
Figure 7: Circuit diagram of inverter circuit 17
Figure 8: Detailed block diagram of braking system 18
Figure 9: Circuit diagram of rectifier circuit 18
Figure 10: Top level block diagram of driving system 20
Figure 11: Pin diagram of IRS2184 half-bridge gate driver 20
Figure 12: Typical connections for the IRS2184 half-bridge gate driver 21
Figure 13: Triangle and three-phase sinusoidal waves 22
Figure 14: Triangle and sine wave comparison 22
Figure 15: Output signals of the three triangle/sine wave comparisons 23
Figure 16: Inverter Circuit 23
Figure 17: Top level block diagram of braking system 24
Figure 18: Three-phase sinusoidal waves 24
Figure 19: Rectifier Circuit 25
Figure 20: Output signals for the most positive 25
Figure 21: Output signals for the most negative 25
Figure 22: Simulink output of inverter 28
Figure 23: Actual output of the inverter circuit 28
Figure 24: Simulink output of the rectifier 29
Figure 25: Actual output of the rectifier circuit 29
Table 1: Cost Analysis 33
Appendix Figure 1: SPWM for inverter circuit 43
Appendix Figure 2: SPWM technique in Simulink 43
Appendix Figure 3: Arduino code for inverter circuit 49
Appendix Figure 4: Final inverter circuit with LC filter 50
Appendix Figure 5: MATLAB calculations for rectifier circuit 52
Appendix Figure 6: Simulink calculations for rectifier circuit 53
Appendix Figure 7: Arduino code for rectifier circuit 54
Appendix Figure 8: Final rectifier circuit with capacitor filter 55
Appendix Figure 9: Final inverter circuit implementation 55
Appendix Figure 10: Arduino code for real-time rectifier circuit 57

INTRODUCTION

The burning of fossil fuels has been an issue for the environment for hundreds of years now. Harmful emissions have continued to pollute the air as a result of using oil, coal, and gas as sources of energy. Electric vehicles, which will also be referred to as electric cars or EVs, have become more popular as people are becoming increasingly aware of the long-term effects of the use of fossil fuels. They are an alternative to vehicles that run with an Internal Combustion Engine (ICE), which will also be referred to as gasoline cars. There are advantages and disadvantages to both types of vehicles, but the long-term effects of gasoline cars are far worse than those of electric cars. Electric cars can use electricity generated by solar and wind energy, which do not release dangerous emissions into the atmosphere.

Electric cars have batteries that run on direct current (DC) power while the motors can run on either alternating current (AC) or DC power. DC powered motors need a DC to DC converter to step up and step down voltage. AC powered motors need an inverter to convert DC power to AC. Regenerative braking is a method of increasing range in electric cars where kinetic energy created from driving is converted to stored energy in the car's battery [1].

The goal of this project is to successfully replicate and understand the power flow in an electric vehicle. This final device will mimic driving and regenerative braking in an electric vehicle. The remainder of this report will be comprised of historical and societal context of electric cars, previous work, different types of pulse width modulation (PWM) control techniques, design requirements, design alternatives, the preliminary proposed design, the final design, the final implementation, performance estimates and results, the production schedule, the cost analysis, a discussion, conclusions, and recommendations.

BACKGROUND

Around 1890, the first electric vehicle was introduced in the United States by William Morrison, a chemist. At this time, electric cars were more attractive in terms of operation and aesthetics compared to early gasoline cars, which required gear changes and a hand crank to start the engine and were accompanied by vibrations, unpleasant smells, and noise. However, as more oil was discovered and gas became more affordable, gasoline cars became more impressive than electric cars since they could operate for longer distances [2]. In addition, a few inventions such as the electric starter and muffler in the early 1890s-1900s made gasoline cars even more intriguing [3]. According to the United Automobile, Aerospace and Agricultural Implement Workers of America (UAW), "With the invention of lithium-ion batteries in the 1980s, EVs started taking baby steps to compete with ICEs, in terms of price and range. Today, EVs can travel over 300 miles on a charge, and take an 80% charge in a half an hour" [4]. Also, the average electric car produces fewer greenhouse gas emissions than the average gasoline car [5].

There are two types of electric vehicles, all-electric vehicles (AEVs) and plug-in hybrid electric vehicles (PHEVs). Both types charge from the electrical grid and "in part by regenerative braking, which generates electricity from some of the energy normally lost when braking" [6]. AEVs run only on electricity and they take 30 minutes or a full day to recharge. PHEVs run on electricity and then switch to using an ICE, which can use hydrogen in a fuel cell, biofuels, or other alternative fuels [6]. As of March 2020, the longest range of an EV is 390 miles with the Tesla Model S [7].

According to David Reichmuth, a senior vehicles engineer, "Based on data on power plant emissions released in February 2018, driving on electricity is cleaner than gasoline for most

drivers in the US…. For electric vehicles, the calculation includes both power plant emissions and emissions from the production of coal, natural gas and other fuels power plants use" [8]. This analysis comes from emissions estimates for gasoline and fuels production from Argonne National Laboratory and power plants emissions data released by the US EPA. Greenhouse gases get trapped in the Earth's atmosphere and humans have been the main cause of the increase of these gases for the past 150 years. In 2017, it was found that transportation contributed 28.9% of these emissions while electricity production contributed 27.5%. Over 90% of the fuel used for transportation is petroleum based, which includes mostly gasoline, and about 62.9% of electricity comes from burning fossil fuels. However, there has been a decrease in carbon dioxide emissions due to the shift from coal to natural gas, the increased use of renewable energy options, and milder temperatures that required less electricity use [9].

In the development of electric vehicles, there are economic, manufacturability, and ethical issues that arise. The production of electric cars is much more straightforward when compared to the production of gasoline cars. This fact could lead to a reduction of jobs related to manufacturing parts for a gasoline car such as the ICE, exhaust systems, and fuel systems. However, this could also lead to new jobs for manufacturing parts for an electric car such as batteries, electric motors, and braking systems. These parts may need to be imported, which would again negatively affect current workers. The economic state of electric cars will not be beneficial if components have to be imported from other countries into the United States.

In order to make lithium-ion batteries, cobalt and lithium have to be mined in large quantities. Child labor, injuries, death, lung disease, and contamination of waterways are

becoming more prevelant issues as there is an increase demand for these resources [4]. However, replicating the inner circuitry of an electric vehicle will not involve any of these issues.

Ben Nelson, an ordinary man from Wisconsin, has an instructable post online with steps to build an electric car. He starts off with a used car and removes anything related to a gasoline car. Then, he proceeds to add the necessary parts for an electric car such as motors, batteries, and controller. Ben is successful in getting his new electric car to run and has videos to prove it and show his process throughout [10]. However, he does not mention the circuitry required.

Instead of an ICE, electric cars have an electric motor that is powered by a controller, which receives its power from rechargeable batteries. In an electric vehicle, there needs to be a controller between the DC battery and the DC or AC motor [11]. AC is an electric current that periodically reverses direction while DC only flows in one direction. Three-phase power is the most common method to transfer power. DC voltage can be converted to three-phase AC using an inverter. Three-phase AC voltage can be converted to DC using a rectifier. In the electric car system, DC power is converted to three-phase AC by switching transistors on and off using the controller. This creates three sine waves with the same frequency and voltage amplitude, and a phase difference of ⅓ or 120 degrees.

In 1959, Mohamed Atalla and Dawon Kahng invented the metal-oxide-semiconductor field-effect transistor (MOSFET). The MOSFET then led to the power MOSFET and the single-chip microprocessor. The power MOSFET allowed electric vehicles to work at higher switching frequencies, and reduced power losses and prices. The single-chip microcontroller allowed command of the drive control and battery [3].

In a rectifier circuit, pulse width modulation (PWM) uses transistors to switch the three-phase AC voltage on and off based on the most positive and most negative input signals. The output is then filtered to obtain a smooth DC output [12]. In V. Vaideeswaran and N. Sankar's paper in the International Journal of Engineering and Advanced Technology (IJEAT), they discuss some examples of PWM techniques for rectifiers - hysteresis-band PWM (HB-PWM), carrier-based sinusoidal PWM (CB-SPWM), and voltage oriented control hysteresis-band PWM (VOC HB-PWM). These techniques require the use of PI controllers. They use an AC line voltage of 110V and their reference DC output voltage is 180V. Their conclusions are that the VOC HB-PWM gives the most accurate results, but the CB-SPWM control technique has the smallest total harmonic distortion (THD) [13].

In Ferdinant Visser's master thesis project for Delft University of Technology, he focuses on four PWM techniques - hysteresis modulation, carrier modulation, space vector modulation, and harmonic elimination. Hysteresis modulation is a current control method where a controller generates a sinusoidal reference waveform that is compared to the actual phase current. Carrier based modulation is a voltage control method where a carrier and a reference waveform are compared. Space vector modulation is a voltage control method that averages three switching state vectors over an interval of one switching cycle. Selective harmonic elimination (SHE) is a method where square wave switching is combined with PWM that controls the fundamental output voltage. Visser also explains the difference between a passive and active AC/DC converter. An active AC/DC converter uses transistors while the passive one uses diodes. In the end, his conclusions are that the hysteresis modulation performs a little better although the robustness of it is worse, so space vector modulation is the overall best PWM method [14].

In an inverter circuit, PWM uses transistors to switch the DC voltage on and off in a defined sequence to produce the AC output voltage and frequency. Higher switching frequencies lead to cleaner waveforms to the motor due to the increase in pulses over each half wave. The torque and speed of the motor can be controlled using PWM by changing the period of the voltage pulses [15]. Malik Zaid, a student at the University of Engineering and Technology in Lahore, wrote a paper about a three-phase inverter where he explains that inverters with PWM help to reduce harmonics that may distort the output voltage. He does simulations to show the phase/line voltages and differences for 120 and 180 degree conduction. For 120 conduction specifically, only two switches of the six transistors conduct at a time. Waveforms for phase and line to line voltages were shown in addition to the hardware. After passing one of the line voltage phase differences through a filter, an AC sine wave is clearly presented [16].

According to Anna University in Chennai, India, “The most efficient method of internal control of [voltage source inverters (VSI)] is by a PWM control technique used within the inverter itself.” There are two main types of PWM techniques - carrier based and carrier less modulation. Carrier based techniques include sinusoidal PWM, modified PWM, random PWM, third harmonic injection PWM, and space vector modulation. Carrier less techniques include delta modulation, specific harmonic elimination, and wavelet modulation. Sinusoidal PWM (SPWM) compares a three-phase balanced sinusoidal reference voltage signal with a high-frequency common triangular carrier voltage signal. The intersection point of the two signals determines the switching of each MOSFET [17].

DESIGN REQUIREMENTS

The final design of this project will consist of two separate systems. The first system will replicate driving an electric vehicle. Below in Figure 1 is a block diagram of the first system, which uses an inverter circuit.

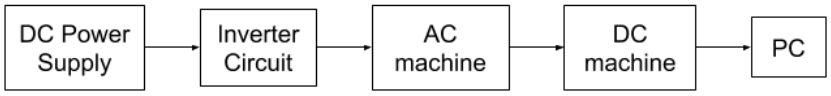


Figure 1: Block diagram of driving system

A 40VDC power supply will be set to an appropriate level based on component ratings. It will be the input to the inverter circuit. The inverter circuit will convert the DC voltage to three-phase AC voltage. It will be built with six of the same type of transistor. The voltage going into each drain of the three MOSFETs connected to the high-side output is 24VDC while the gate voltage for all six should be between 10-20VDC based on their ratings and will be controlled with an Arduino Mega 2560 Rev3 microcontroller [18]. The Arduino's operating voltage is 5VDC, which the USB regulates when connected to power [19]. A gate driver is needed to increase this voltage to between 10-20VDC.

One half-bridge gate driver will control a pair of MOSFETs. For each gate driver, a 1uF capacitor will connect the high-side floating supply, VB, to its supply return, VS, and a diode will connect VB to the low-side and logic fixed supply, VCC [20]. They will each have an external input of 12VDC into VCC, and 5VDC from the Arduino into the logic input for shutdown, SD, which is referenced to the low-side return, COM. A resistor will be connected between each output of the gate driver and corresponding gate of each MOSFET to limit the

current, which should be less than the rated 20-40mA of the Arduino [19]. It will be 4.7kΩ since 12VDC is going into the MOSFETs [21].

MOSFETs control voltage, so they will control the output voltage of the inverter circuit that will become the input to the AC induction motor. Voltage, current, and power into the MOSFETs should not exceed 60V, 70A, and 230W, respectively [18]. Sinusoidal pulse width modulation (SPWM) will be used to control the output of the inverter circuit through Arduino code [22]. The input to the AC induction motor should not be greater than 15VAC, 2A, and 30W. The three-phase AC signals should be 50Hz and 120 degrees phase shifted. Inductors and capacitors will be used to create a low pass filter at the output of the inverter circuit to filter out noise and reduce harmonic components, so the inductor value will be 12mH and the capacitor value will be 110uF [23]. The AC induction motor should start spinning as soon as power is provided and run continuously without interruption. The DC machine will be torque controlled by the Sciamble hardware from the University of Minnesota and computer to simulate a mechanical load in an EV.

The second system will replicate braking in an electric vehicle. Below in Figure 2 is a block diagram of the second system, which uses a rectifier circuit.

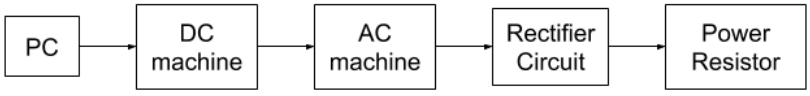


Figure 2: Block diagram of braking system

A computer will control the speed of the DC machine, which will be coupled to the AC permanent magnet motor. The three-phase AC power will be the input to the rectifier circuit. The rectifier circuit will convert the three-phase AC voltage to DC voltage. The circuit will again be

built with six of the same type of transistor. However, there will be six low-side gate drivers instead. To ensure low supply impedance, each gate driver will have two 0.1uF capacitors connected between pins 1 (VS) and 4 (GND), and between pins 5 (VS) and 8 (GND) [24]. The gate voltage going into each of the MOSFETs will be 12VDC and controlled with an Arduino Mega 2560 Rev3 microcontroller. A 4.7kΩ resistor will be connected between the output of the gate driver and gate of each MOSFET to limit the current. The Arduino will be powered by a USB connection to the computer, which allows it to output 5VDC and the gate driver will increase this voltage to 12VDC.

The MOSFETs will control the output voltage of the rectifier circuit that will become the input to the power resistor. Again, voltage, current, and power into the MOSFETs should not exceed 60V, 70A, and 230W, respectively. MATLAB calculations will determine the PWM to control the output of the rectifier circuit through Arduino code. The input to the DC motor should not be greater than 36VDC, 4A, and 144W. The input to the AC permanent magnet motor, which operates as a generator, should not be greater than 28VAC, 7A, and 196W. The input to the power resistor should not be greater than 500VDC, 120A, and 140W [25]. A 1000uF capacitor will be used to filter the output of the rectifier circuit due to the larger output voltage [26]. The power resistor acts like a battery that is charging in an EV, so the DC power should be measurable as soon as power is provided to the system, which should continue without interruption.

DESIGN ALTERNATIVES

The first major design alternative that was considered was the type of source for each system. The source for the inverter system could have been from the computer instead of the DC power supply. The source for the rectifier system could have been the three-phase AC power from the wall outlet instead of the computer. In the first scenario, the source would have been the computer connected to a DC machine. The DC machine would have then been connected to a DC generator with DC power as the output. This output would become the input to the inverter circuit, which would convert DC voltage to three-phase AC to power an AC induction motor. In the second scenario, the source would have been the three-phase AC power from the wall outlet connected as the input to the rectifier circuit. The rectifier circuit would convert three-phase AC voltage to DC to power a DC motor. The motor would have then been connected to a DC machine that would be controlled by the computer to replicate a mechanical load. Using the computer to create a three-phase AC source instead of using the wall outlet as a source was a safer approach and using a DC supply was a better representation of a battery than the combination of the computer, DC machine, and DC generator. The design that was chosen more accurately portrays the braking and driving processes in an electric vehicle.

There were also some alternatives considered for three-phase PWM control which included delta modulation and space vector modulation (SVM). Delta modulation uses a sine voltage as a reference signal and a delta-shaped signal as a carrier signal, which is generated by integrating the output pulse. These two signals are compared in a comparator, where the output error signal is passed into a hysteresis comparator and quantized into an upper or lower limit to determine the point for the switching pulses. This is a technique that is easy to implement, and

has continuous inverter voltage control and direct control of the line harmonics. However, the inverter output waveform is not synchronized with the control signal. SVM is specifically used for three-phase induction motor drives. It reduces the lower output AC voltage that happens with sinusoidal pulse width modulation (SPWM) and generates an output voltage and current with less harmonic distortion. However, it is a more complicated control technique [17].

In a paper written by Renu Sharma (a post graduate power systems student at the Institute of Engineering and Technology), Deepak Kumar Goyal (an assistant professor at Government Engineering College) and Harpreet Singh (an assistant professor at the Institute of Engineering and Technology), they conclude that SVM has become the most popular and important technique for three-phase voltage source inverters for the control of AC induction motors, because it generates less THD than the SPWM control technique. However, they also agree that SPWM is most widely used due to its ease of implementation [27].

In the final design, SPWM was chosen for the inverter circuit, because it is easy to implement and control, and is compatible with most modern digital systems [17]. The comparisons necessary in SPWM will be executed in MATLAB to determine which pair of MOSFETs are on or off, which will be hard-coded into Arduino. The rectifier circuit will use MATLAB to calculate the most positive and most negative input signals for a period to determine the MOSFETs switch on and off times to be hard-coded into Arduino as well.

PRELIMINARY PROPOSED DESIGN

The three-phase pulse width modulated AC/DC rectifier and DC/AC inverter can be split up into two separate systems of five different blocks each. Below in Figure 3 is a top level block diagram of the driving system.

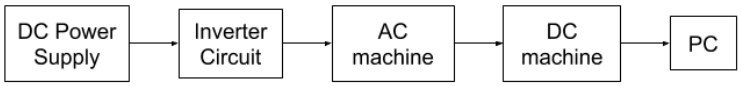


Figure 3: Top level block diagram of driving system

These five blocks consist of the source, inverter circuit, AC induction motor, DC machine, and computer. Below in Figure 4 is a top level block diagram of the braking system.

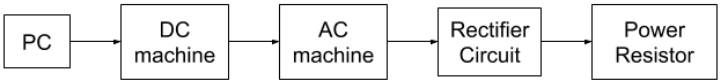


Figure 4: Top level block diagram of braking system

These five blocks consist of the computer, DC machine, AC permanent magnet motor, rectifier circuit, and load.

For both systems, the same circuit can be used and will be built on a breadboard. Six IRFP054 power MOSFETs will be used in combination with three gate drivers. These will be the MIC4422 MOSFET drivers, which have been used in previous projects and can be used in combination with Arduino's 5VDC output to increase it to 12VDC for the gate of the transistors [28] [29]. Below in Figure 5 is a pin diagram of the MOSFET driver.

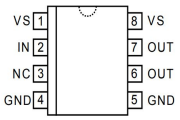


Figure 5: Pin diagram of MOSFET driver

Pin 2 (IN) will be connected to the Arduino's 5VDC logic output, pins 4 and 5 (GND) will be connected to ground, pins 1 and 8 (VS) will be connected to an external 12VDC input with 0.1uF capacitors connecting those pins back to 4 and 5 (GND), respectively, to ensure low supply impedance, and pins 6 and 7 (OUT) will be connected to the gates of two MOSFETs [24]. The outputs of the driver will be 12VDC each. The MOSFETs are N-Channel MOSFETs, because P-Channel MOSFETs are more difficult to control. This specific IRFP054 MOSFET was also chosen based on its voltage, current, and power ratings. The maximum voltage, current, and power it can handle is 60V, 70A, and 230W, respectively. The transistors need 10-20VDC at its gate to switch on, which is why a gate driver is necessary. A 4.7kΩ resistor will also be connected to the gate to limit current. The motors will come from the Sciamble brand from the University of Minnesota, but only the DC machine will be controlled directly by the computer. The DC supply and computer will be provided by Professor Luke Dosiek. The Arduino will eventually be powered by an external battery pack with the Arduino code loaded. Each system should convert the voltage from DC to AC and AC to DC as soon as power is provided and run continuously without interruption.

Below in Figure 6 is a detailed block diagram of the driving system.

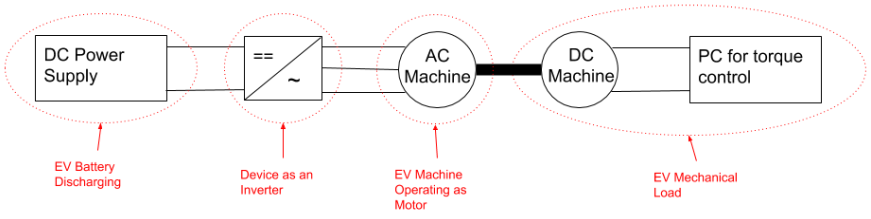


Figure 6: Detailed block diagram of driving system

For driving, the 40VDC supply will be used as the battery of an EV. The inverter will convert the DC voltage to three-phase AC voltage to power the AC induction motor, which represents the

main traction motor in an EV. The input to the AC induction motor should not be greater than 15VAC, 2A, and 30W. Below in Figure 7 is a circuit diagram of the inverter circuit.

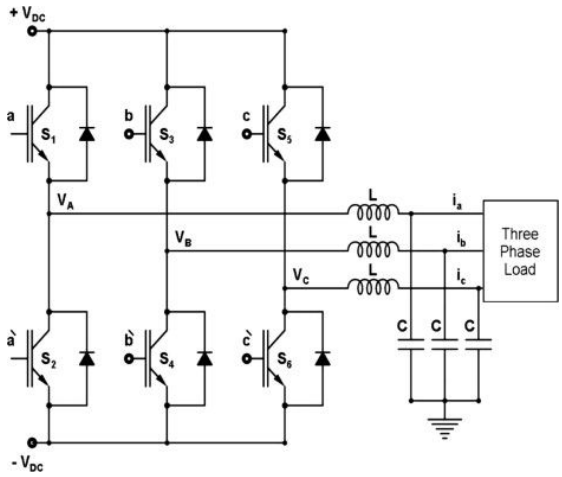


Figure 7: Circuit diagram of inverter circuit

A 17VDC signal will be connected to the drain of each transistor in the top row. The source of each transistor in the bottom row will be connected to ground. The sources of the top row transistors will be connected to the drains of the bottom row transistors and these will be the signals necessary to create the outputs. The comparison of a triangle wave to a sine wave with a slightly smaller amplitude will be done in MATLAB to determine which of the six MOSFETs are on or off [22]. This logic will then be implemented into the Arduino code to turn on or off a switch. The output frequency should be around 50Hz, so the inductor value should be 2mH and the capacitor value should be 5.087mF. These values can be changed as needed. The three outputs will then be connected to the three-phase AC induction motor. The AC motor is then mechanically coupled to the DC machine, which is torque-controlled by the lab computer to simulate the mechanical load of an EV such as friction, drag, and acceleration/inertia.

Below in Figure 8 is a detailed block diagram of the braking system.

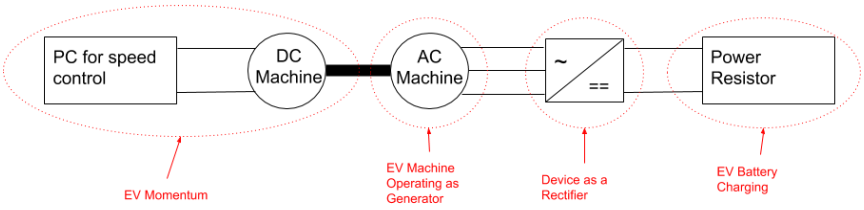


Figure 8: Detailed block diagram of braking system

The braking system will use the same DC machine as the driving system. The computer will control the speed of the DC machine, which represents an EV putting its momentum back into the system. The DC machine is mechanically coupled to the AC permanent magnet motor, which spins as a generator and still represents a traction motor in an EV. The input to the AC permanent magnet motor should not be greater than 28VAC, 7A, and 196W. The rectifier will convert the three-phase AC voltage to DC voltage to power the power resistor. The input to the power resistor should not be greater than 500VDC, 120A, and 140W. Below in Figure 9 is a circuit diagram of the rectifier circuit.

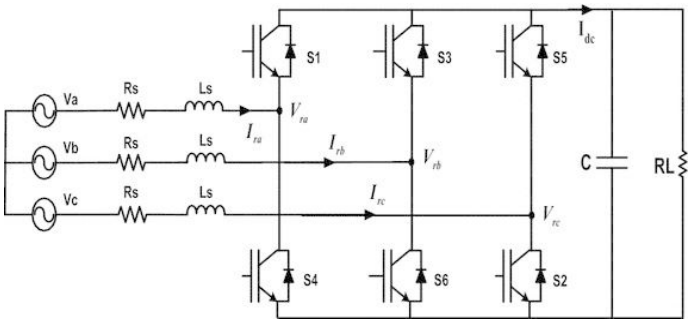


Figure 9: Circuit diagram of rectifier circuit

Each drain of the transistors in the top row will be connected to each other. Each source of the transistors in the bottom row will be connected to each other. The sources of the top row transistors will be connected to the drains of the bottom row transistors and this will be where each of the three-phase AC input signals are connected. The comparison of AC voltages from

each of the three phases will be compared in MATLAB to determine which MOSFETs are switched on or off [29]. The timing will be calculated in MATLAB as well, so it can be implemented into the Arduino code. A 1000uF capacitor will be used to filter the output due to the larger output power. This value can be changed as needed. This output is connected to a power resistor, which represents the EV battery being recharged.

The goal is to replicate what happens in an electric vehicle using AC and DC motors. The two systems should start moving as soon as power is provided and should continue to run without interruption. The voltage in the power resistor should be measurable at any time. If there is time, bidirectional power flow can be implemented in both the inverter and rectifier circuits in which the systems can switch between driving and braking in real time.

FINAL DESIGN

The final design follows very closely with the preliminary proposed design. However, the circuitry is slightly modified due to changes of certain circuit parts, which will be discussed in this section. The design is still made up of two separate systems where the first system replicates driving and the second system replicates braking in an electric vehicle.

Below in Figure 10 is the same top level block diagram of the driving system as previously presented.

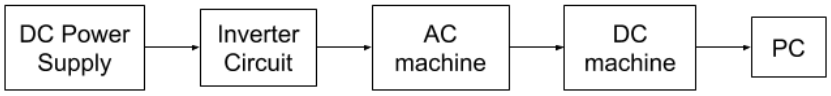


Figure 10: Top level block diagram of driving system

As mentioned earlier, a 40VDC power supply is used as the DC voltage input into the inverter circuit, which is used to convert the DC voltage to three-phase AC voltage. The inverter circuit used six IRFP054 power MOSFETs in combination with three IRS2184 half-bridge gate drivers, which replaced the original three MIC4422 low-side gate drivers. This replacement was due to switching time issues with the Arduino and realizing that the two outputs of the MIC4422 had to be externally connected to each other, which meant they had the same exact output instead of opposite and different outputs as desired. The IRS2184 half-bridge gate driver is still capable of increasing the Arduino's 5VDC output to 12VDC for the gate of the MOSFETs. Below in Figure 11 is a pin diagram of the IRS2184 half-bridge gate driver [20].

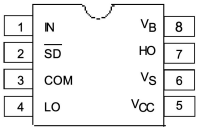


Figure 11: Pin diagram of IRS2184 half-bridge gate driver

Below in Figure 12 is a circuit diagram of typical connections to use this gate driver [20].

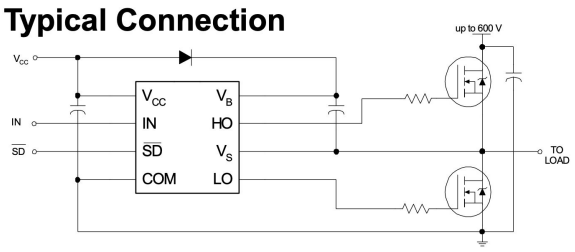


Figure 12: Typical connections for the IRS2184 half-bridge gate driver

The input to the gate driver's VCC was 12VDC. The input to the logic input for shutdown, SD, that is reference to the low-side return, COM, was 5VDC from the Arduino. The logic input for high-side and low-side gate driver outputs, in phase with high-side, IN, came from an Arduino output pin. The capacitor connecting the high-side floating supply, VB, to the high-side floating supply return, VS, was 1uF. This is a bootstrap capacitor whose value should be high enough to provide the required gate charge of the high-side MOSFET. The equation used was Q = C * V where Q is the gate charge required by the MOSFET, C is the bootstrap capacitance, and V is the bootstrap voltage. It is recommended that the charge stored by the capacitor is about 50 times more than the required gate charge at operating VCC. Q was found to be 160nC at 10V, so C = (160nC / 10V) * 50 = 800nF and the closest capacitor available was 1uF [18] [30]. The other capacitors connecting VCC to ground and the MOSFET voltage input to ground were 470uF, which were chosen by trial and error to minimize noise. A bootstrapping diode was required in the circuit, so a standard 1N4148 diode was chosen since its VR rating is higher than the voltage rating of the gate driver [20] [30] [31]. A 4.7kΩ resistor will be connected from the low-side and high-side gate outputs to the MOSFET gates to limit current. VS will also be connected to the source and drain connection between a pair of MOSFETs.

The logic input for each gate driver is based on SPWM, which was executed in MATLAB (See Appendix Figure 1). The same SPWM technique was simulated in Simulink (See Appendix Figure 2). Below in Figure 13 is a graph of the SPWM technique from MATLAB, where a triangle wave with a switching frequency of 500Hz is compared to three sine waves, which have amplitudes of 0.8V, frequencies of 50Hz, and are 120 degrees phase shifted from each other. Below in Figure 14 is a graph of the same triangle from Simulink, but it is only compared to the 0 degree phase shifted sine wave and the output of this comparison is shown.

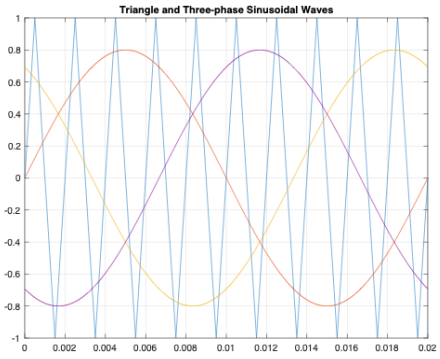


Figure 13: Triangle and three-phase sinusoidal waves

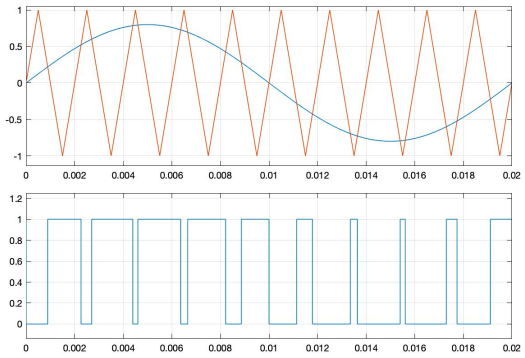


Figure 14: Triangle and sine wave comparison

This comparison was done with the 120 degree and 240 degree phase shifted sine waves as well. The MOSFETs are paired up, so when the output signal is high or 1, the first MOSFET is on while the second is off. Below in Figure 15 is a graph of the three output signals overlapping and is used to determine the logic for the Arduino. The actual amplitude of each signal is 1, but for the purposes of showing the three signals more clearly, the amplitudes were changed.

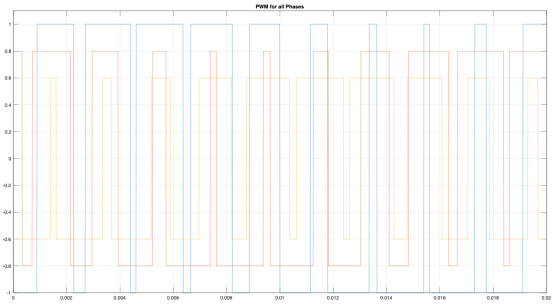


Figure 15: Output signals of the three triangle/sine wave comparisons

The on and off times were found using this graph and then hard-coded into Arduino (See Appendix Figure 3). Below in Figure 16 is the circuit diagram of the inverter circuit.

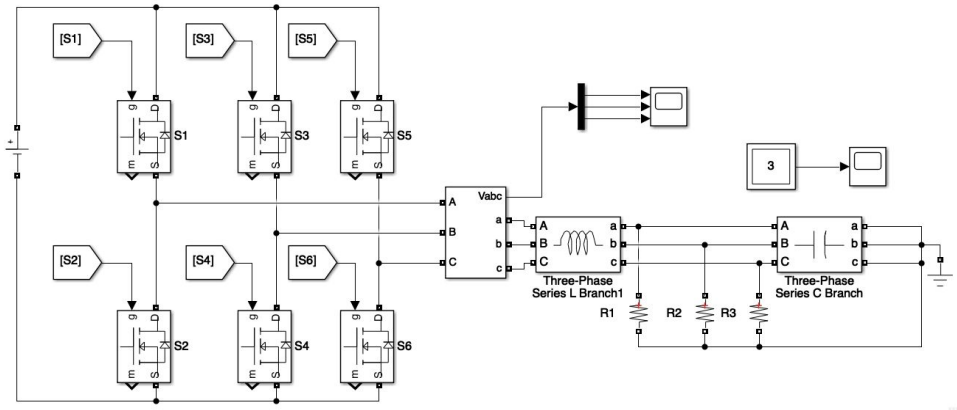


Figure 16: Inverter Circuit

The MOSFETs are paired vertically, so MOSFET 1 and 2 are controlled by one of the three half-bridge gate drivers. An LC filter was implemented into the final inverter circuit (See Appendix Figure 4). Originally, a 1mH inductor in combination with a 1000uF electrolytic capacitor was used, but the inductor started to burn out. A 12mH inductor was available, so using the formula to find cutoff frequency for an LC filter using SPWM,

$\sqrt{(sine\ wave\ frequency) * (switching\ frequency)} = \sqrt{50 * 500} = 158.1$Hz, and Simulink to test

capacitor values to filter out noise, the conclusion was that a 110uF non-electrolytic capacitor would be used. The cutoff frequency was calculated using the formula fc = $\frac{1}{2\pi\sqrt{RC}}$ to be 138.5Hz.

Below in Figure 17 is the same top level block diagram of the braking system as previously presented.

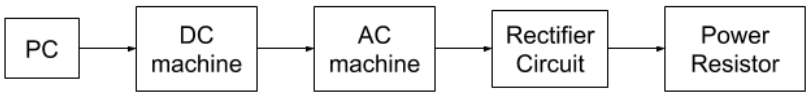


Figure 17: Top level block diagram of braking system

The final design of the rectifier circuit is consistent with almost everything in the preliminary proposed design except for the fact that there will be six instead of three low-side gate drivers controlling each of the six MOSFETs.

The logic input for each gate driver is based on the most positive and most negative input signals from the AC machine. Calculations were done in MATLAB and Simulink (See Appendix Figures 5 and 6). Below in Figure 18 is a graph of a typical three-phase AC output.

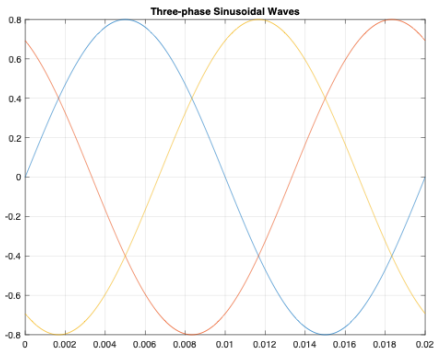


Figure 18: Three-phase sinusoidal waves

Below in Figure 19 is the circuit diagram of the rectifier circuit.

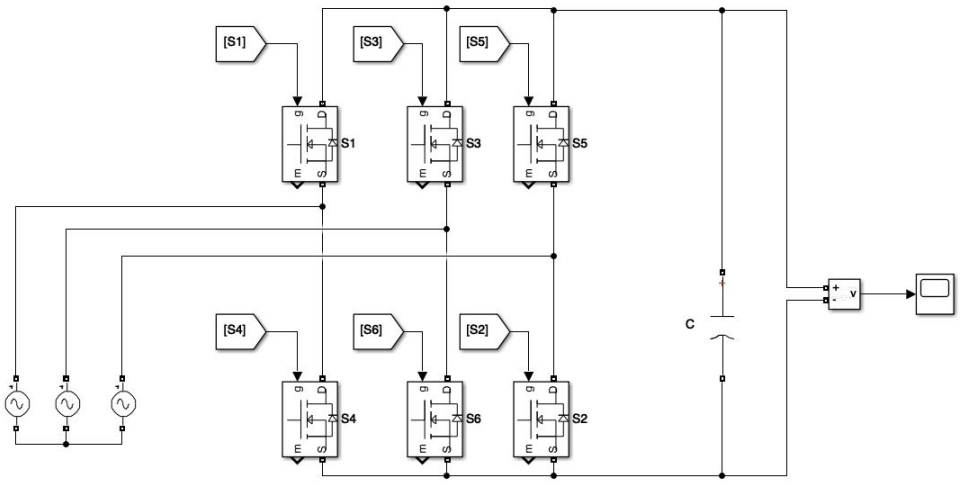


Figure 19: Rectifier Circuit

Below in Figures 20 and 21 are the outputs corresponding to the most positive and most negative input signals that determine the logic for the Arduino.

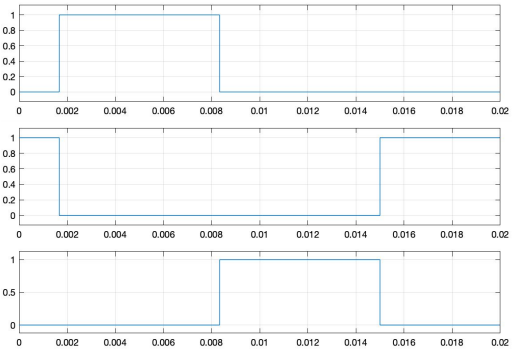


Figure 20: Output signals for the most positive

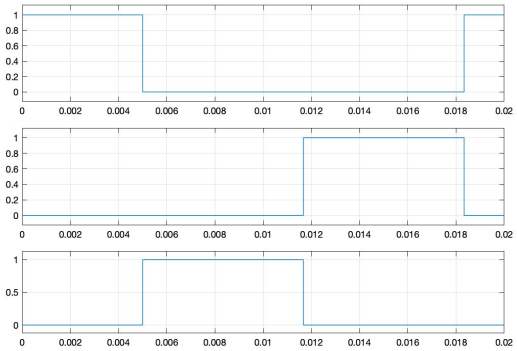


Figure 21: Output signals for the most negative

Figure 20 resembles the top row of MOSFETs in the rectifier circuit while Figure 21 resembles the bottom row. Each graph represents the on and off times that were hard-coded into Arduino for each of the six MOSFETs (See Appendix Figure 7). A capacitor filter was implemented into the final rectifier circuit (See Appendix Figure 8). A 1000uF electrolytic capacitor was used.

FINAL IMPLEMENTATION

After the inverter circuit was built and tested by observing oscilloscope waveforms, it was connected to the DC power supply. The AC induction motor tried to spin with the converted AC power, but it was too weak, so the drain for the top row of MOSFETs was connected to 24VDC (See Appendix Figure 9).

After the rectifier circuit was built and tested by again observing oscilloscope waveforms, the three-phase AC signals from the AC machine were connected. However, the Arduino code was based on a perfect start to a period of three-phase AC signals, which was not practical.

PERFORMANCE ESTIMATES AND RESULTS

Prior to starting the project, the estimated performance of the project based on the preliminary design was to successfully replicate driving and braking in an electric vehicle. This included a working inverter and rectifier circuit, real-time inputs that would be controlled by the Arduino, using SPWM to control the MOSFETs in the inverter and comparing each of the three-phase AC input signals to control the MOSFETs in the rectifier, and integration with an AC induction motor for the inverter and with a power resistor for the rectifier. If there was time after all of this was accomplished, bidirectional power flow would be attempted.

In the driving scenario, the DC power supply would send DC voltage into the inverter circuit, which would then convert that voltage to three-phase AC voltage. This three-phase AC voltage would then power the AC machine acting as an induction motor instantly and continuously. The AC motor should move. The three-phase AC output signals should each have a frequency of 50Hz and be 120 degrees phase shifter from each other. A DC machine controlled by a computer would be coupled to the AC motor to control its torque.

In the braking scenario, the same DC machine controlled by a computer would be coupled to an AC machine acting as a permanent magnet. The AC machine would send three-phase AC voltage signals into the rectifier circuit, which would convert the three-phase AC voltage to DC instantly and continuously. This DC voltage would then power a power resistor, which could be measured using a voltmeter.

After working on the project for ten weeks, the driving system runs continuously and as soon as power is provided. A real-time input was used and SPWM was used to control the MOSFETs. The driving system also moves the AC induction motor when 24VDC is applied to

the MOSFETs with a current draw of 0.62A. The motor can start spinning at 20VDC, but the motor has to be manually started. The output frequency of each of the three-phase AC signals is 50Hz and they are 120 degrees phase shifted. Below in Figure 22 is the Simulink output of the inverter.

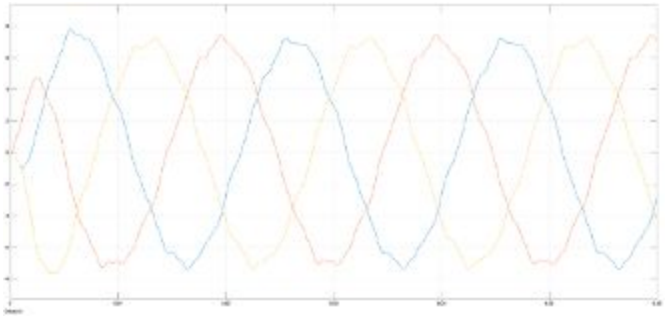


Figure 22: Simulink output of inverter

Below in Figure 23 is the actual output of the inverter circuit.

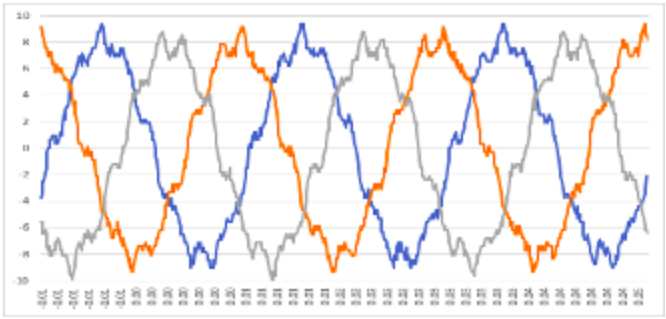


Figure 23: Actual output of the inverter circuit

The DC machine could have been connected, but it was decided that the AC induction motor did not run as strongly as anticipated, so it did not seem necessary.

The braking system runs continuously and as soon as power is provided, but it does not use real-time inputs. It does use the DC machine connected to the computer and then to the AC machine. However, the code that was written for the rectifier circuit relies on the input starting perfectly at the beginning of a sine wave period, which is inconsistent with using a real-time

input. The three-phase AC voltage was not fully converted to DC, so a power resistor was not powered. Below in Figure 24 is the Simulink output of the rectifier.

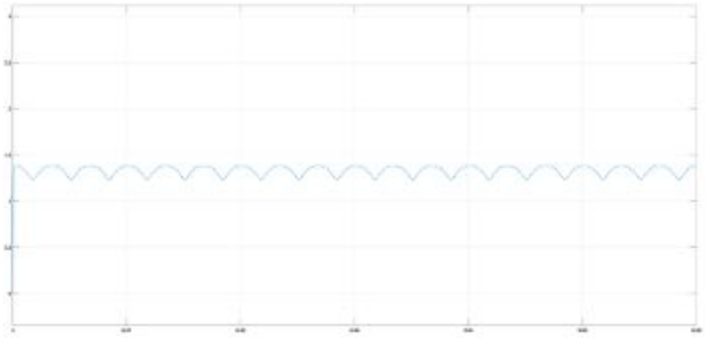


Figure 24: Simulink output of the rectifier

Below in Figure 25 is the actual output of the rectifier circuit.

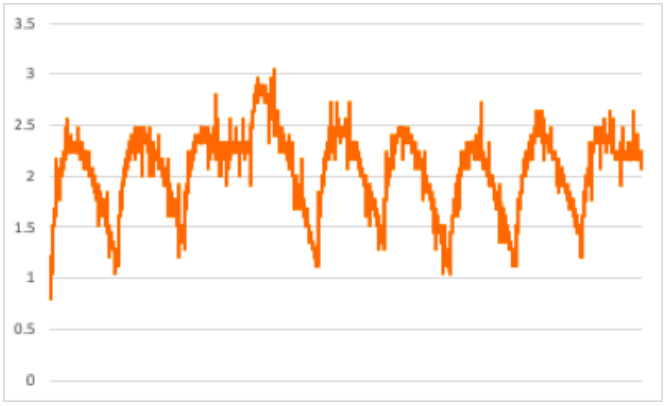


Figure 25: Actual output of the rectifier circuit

The actual output does not accurately depict what the output should look like, because the code was written for a perfect input, which is not realistic.

Some suggestions that would improve the performance of the project would be to use real-time AC voltage inputs for the braking system with a non-inverting amplifier, implement feedback control for real-time tuning of magnitude and frequency of the inverter output, and average value for rectifier output, and feedback for speed, torque, or position control of a motor load, include a mechanical load in the driving scenario, and create bidirectional power flow.

PRODUCTION SCHEDULE

FALL 2019

1. Set up meeting time for the rest of the term
2. Did background research about three-phase AC, rectifier and inverter circuits; tried to find instructables of rectifiers and inverters
3. Did more background research on three-phase inverters, PWM control techniques for rectifiers and inverters; looked for MATLAB/Simulink examples for PWM, rectifiers, and inverters; found schematics to simulate
4. Looked for parts for three-phase PWM inverters and rectifier swith LC or capacitor filters; identified main research question - replicating an EV; decided not to work on jonathan's microgrid project
5. Asked for help on deciding switches; decided to go with MOSFETs; made a list of use cases, how testing will work, what will be tested
6. Decided on research question - bidirectional power converter for automotive applications; figured out voltage, current, and power requirements for transistors based on available motors
7. Had components picked out and ready to order; decided on PWM technique - SVM
8. Started working on Simulink simulations - AC voltage source, DC voltage source
9. Continued working on Simulink simulations; figured out how to enable PWM; chose to use Arduino; worked on 498 poster; sent in SRG application
10. Worked on 498 paper

WINTER 2020

1. Finished inverter Simulink models and MATLAB code; decided against Multisim circuits
2. Built inverter circuit with one gate driver and two MOSFETs - realized two outputs must be connected, so six gate drivers are necessary; started Arduino code for MOSFETs
3. Submitted 2nd SGR for gate drivers; finished Simulink model for PWM rectifier; built single phase PWM inverter circuit with two MOSFETs and two gate drivers and found distortions and no AC sine wave output and realize single phase has four MOSFETs
4. Set up single phase PWM inverter circuit and found distortions with delay of microseconds but not ms - the timing was off and too much current was being drawn, but a 1k resistor helps; tested different frequencies with one gate driver and one MOSFET - microseconds gives a frequency of 400Hz with distortions at the drain closest to MOSFET, gate closest to MOSFET and Arduino input, ms gives 0.5Hz with no distortions, 10^-5s gives 50Hz with almost not distortion at drain closest to the MOSFET but distortions at gate closest to the MOSFET but a clean squarewave at Arduino input
5. Used the same code but for single phase four MOSFETs, but the timing was still off; tried a pair and that worked - got 50Hz and clean Arduino input with 10^-5s; adding a 1k resistor to source and ground worked for single phase; tested a 10^-4s delay and added 3us delay - no difference, capacitance value keeps them from turning on and off quickly enough, pair of MOSFETs on at the same time when they should not be; tested the input with 17VDC to the gate driver and 5VDC into the drain o f the top MOSFETs - added 82us deadband but had to fix the rest of the code; gate driver can only go up to 18VDC;

need to look into high side gate drivers with two outputs or inverter gates - getting half bridge gate drivers without needing a delay since the signals are just opposites

6. 82us delay works the same as last week; half-bridge gate drivers came - they work, coded for single phase; need to code for three-phase
7. Started coding three-phase case; implemented the third half-bridge gate driver to inverter - calculated voltage to drain has to be around 20VDC to start; connected the sources together to oscilloscope ground with resistor but only one oscilloscope was grounded
8. Finalized inverter circuit; started code for rectifier circuit; put 1000uF and 1mH as LC filter, the inductor sparked; moved to 12mH and 110uF for cutoff freq = 138.5Hz - used two 220uF electrolytic capacitors to make a non-polarized one; will decide if dc motor is connected to ac motor for torque
9. Finished code for rectifier; three-phase AC power will come from PC to DC motor to move AC motor - three outputs stepped down for the Arduino to read, but Arduino cannot read negative input signals; permanent magnet cannot be used; worked on presentation
10. Worked on poster; integrated hard-coded Arduino code to rectifier circuit; worked on final paper

Some improvements that could have been made would have included some testing or in depth data sheet reading in the Fall term and starting multiple meetings a week in the first week. The testing or in depth data sheet reading would have given a better idea of how the circuit would be connected and led to a change in gate drivers in the first week. Having multiple meetings a week at the start of the term would have helped to keep on track with the implementation schedule.

COST ANALYSIS

ITEM	COST/UNIT	NUMBER	TOTAL	REQUESTED	REFERENCE
Arduino Mega 2560 R3	$14.99	1	$14.99	$14.99	Amazon
N-Channel MOSFET IRFP054PBF	$3.05	18	$54.90	$64.31	Mouser
Electrolytic Capacitor Assortment 0.1uF-1000uF	$14.99	1	$14.99	$14.99	Amazon
Resistor Kit Assortment 0 Ohm-1M Ohm	$10.86	1	$10.86	$10.86	Amazon
DIP Inductor Assorted Kit 1uH to 1mH	$10.99	1	$10.99	$10.99	Amazon
Power Resistor	$15.06	4	$60.24	$69.65	Mouser
Low-side MOSFET Driver MIC4422YN	$2.14	9	$19.26	$27.25	Mouser
Half-bridge Gate Driver IRS2184PBF	$2.71	6	$16.26	$26.19	Mouser
					Total Cost = $239.23

Table 1: Cost Analysis

The Arduino Mega board, boxes of electrolytic capacitors and resistors, bag of inductors, and power resistors can be used again. Capacitors, resistors, and inductors that were used were not put back into the boxes or bag.

USER'S MANUAL

In order to operate the driving system, three different DC voltages need to be applied to the inverter circuit. The first is a 12VDC voltage at 0.17A that powers the three IRS2184 half-bridge gate drivers. The second is a 24VDC voltage at 1.62A that powers the three-phase AC induction motor. Some voltage is lost as heat through the MOSFET and LC filter, but the output voltage to the AC induction motor should not exceed 15VAC at 2A. The third is a 5VDC voltage from the Arduino into the logic input for shutdown, SD, pin of each gate driver. There are three non-electrolytic filter capacitors, so one lead of each is connected to each of the three-phase AC input wires and one lead of an inductor. The other lead of the inductor is connected to the sources of the three MOSFETs connected to the high-side output. The remaining lead of the filter capacitors are all connected to each other (See Appendix Figure 4).

The Arduino code names each pin before the setup section, where it sets each pin to be an output. The loop is where each pair of MOSFETs is set to be on or off in a defined sequence for one period in the code, but is repeated due to the Arduino's loop (See Appendix Figure 3).

In order to operate the braking system, one DC voltage and three AC voltages are applied to the circuit. A 12VDC voltage at 0.17A powers the six MIC4422 low-side gate drivers. The three AC voltages come from the three-phase AC permanent magnet motor where each wire is connected to the source and drain connection between two MOSFETs. An electrolytic filter capacitor is connected to the three MOSFET sources connected together on its negative lead and to the three drains connected together on its positive end (See Appendix Figure 8).

The Arduino code names each pin before the setup section, where it sets each pin to be an output. The loop is where each MOSFET is set to be on or off in a defined sequence for one

period, but is repeated due to the Arduino's loop (See Appendix Figure 7). However, this loop should set each MOSFET to be on or off depending on the most positive or most negative signal input in real-time for one period in the code (See Appendix Figure 10).

DISCUSSION, CONCLUSIONS, AND RECOMMENDATIONS

Electric vehicles are becoming a lot more prevalent in the world today due to their advantages over gasoline cars. Today, the top ten longest range electric vehicles can travel a minimum of 201 miles. These do not include PHEVs [7]. EVs require inverters and converters to work. All electric cars run on DC batteries. If an electric car has an AC motor, it needs an inverter to convert DC power to AC for the motor, but it also needs a converter to power other parts of the car such as the windshield wipers or radio. If an electric car has a DC motor, it needs a converter to convert DC power to DC for the motor, which means stepping up or stepping down the voltage of the battery.

The goal of this project was to successfully replicate driving and braking in an electric vehicle. The final product should aid in the understanding of the power flow in one as well. The inverter circuit used a very simple PWM control technique, which made it easier to code into Arduino from MATLAB. At first, SVM was almost chosen as the technique to be used in this project, but it was too complicated to understand and then implement in MATLAB or Arduino. The inputs into the inverter circuit were hard-coded in Arduino, but that was necessary for the circuit, because that was one way to make sure the MOSFETs were turning on and off at the correct times to create sinusoidal waves. However, the rectifier circuit should not have had hard-coded Arduino inputs. It should have used real-time input signals, but the rectifier circuit was not started until the end of the project, so this was not initially considered as an issue. Arduino code was also written already for the rectifier circuit with real-time input signals, but the negative values could not be read with the parts available (See Appendix Figure 10). The rectifier can still be worked on, but the driving system works as expected.

There are many new components that can be added to this project such as feedback systems, regenerative braking investigations, and even further understanding of power loss in an electric vehicle system. A different PWM approach could be taken on the driving system to create a cleaner output. The input sources could also be switched as mentioned in the design alternatives section.

Throughout this capstone project experience, the ability to do good research has been one of the most important skills necessary to choose parts, techniques, and even for understanding. Understanding a topic close to fully is one of the most important lessons, because it really helps during any step in the process. It is difficult to move to a new part of the project without understanding what happened in the previous part. This is why it was so important to understand why a part did not work or why a different part would work better. Without research, finding new parts or techniques would be very challenging. At the beginning of the project, learning about how an electric vehicle worked was not easy. However, after doing many hours of research and trying to understand articles, journals, and even videos, it became easier.

ACKNOWLEDGEMENTS

Thank you to Union College, Prof. L. Dosiek, Prof. J. Hedrick, Prof. J. Sanchez, Gene Davison, and SRG Funding.

REFERENCES

[1] Toll, Micah. “Regenerative Braking: How It Works and Is It Worth It in Small EVs?” *Electrek*, 24 Apr. 2018.
[2] Department of Energy. “The History of the Electric Car.” *Energy.gov*, U.S. Department of Energy, 15 Sept. 2014.
[3] Wikipedia. “History of the Electric Vehicle.” *Wikipedia*, Wikimedia Foundation, 4 Feb. 2020.
[4] UAW Research Department. “Taking the High Road: Strategies For a Fair EV Future.” *UAW*, The International Union, United Automobile, Aerospace and Agricultural Implement Workers of America (UAW), Jan. 2020.
[5] Reichmuth, David. “Are Electric Vehicles Really Better for the Climate? Yes. Here's Why.” *Union of Concerned Scientists*, Union of Concerned Scientists, 11 Feb. 2020.
[6] Office of Energy Efficiency and Renewable Energy. “Electric Vehicle Basics.” *Office of Energy Efficiency and Renewable Energy*, U.S. Department of Energy, 2020.
[7] Loveday, Steven. “10 Longest Range Electric Cars of 2020.” *CARFAX*, CARFAX, Inc, 5 Mar. 2020.
[8] Reichmuth, David. “New Data Show Electric Vehicles Continue to Get Cleaner.” *Union of Concerned Scientists*, Union of Concerned Scientists, 26 Feb. 2020.
[9] “Sources of Greenhouse Gas Emissions.” *EPA*, United States Environmental Protection Agency, 13 Sept. 2019.
[10] Nelson, Ben. “Build Your Own Electric Car!” *Instructables*, Autodesk, Inc., 3 Sept. 2019.
[11] Brain, Marshall. “How Electric Cars Work.” *HowStuffWorks*, HowStuffWorks, 27 Mar. 2002.
[12] katkimshow, “Fundamentals of Power Electronics: Three-Phase Diode Rectifier Basics,” YouTube, 18 March 2018.
[13] V. Vaideeswaran and N. Sankar, "Control Techniques of Three Phase PWM Rectifier." *International Journal of Engineering and Advanced Technology (IJEAT)*, Volume-8, Issue-2S, December 2018.
[14] Ferdinant Visser, "Design and implementation of a bi-directional 3 phase converter for a 30kW range extender application," Delft University of Technology, 12 September 2011.

[15] Glampe, Mike. “Pulse Width Modulation in VFDs - A Primer on How PWM Is Used in Drives.” *KEB*, KEB America Inc., 11 July 2018.
[16] Malik Zaid, "Three Phase Inverter," LinkedIn Corporation, 31 December 2016.
[17] Anna University, Chennai, "11_chapter 3," Anna University, Chennai, pp. 33-55.
[18] *Power MOSFET*, IRFP054, Vishay Siliconix, 14 March 2011, pp. 1-2.
[19] *Arduino Mega 2560*, Arduino, 2019.
[20] *Half-Bridge Gate Driver*, IRS2184, Infineon, 2006, pp. 1-9.
[21] Osborne, Mark. “Arduino PWM MOSFET Gate Resistor.” *Becoming Maker*, WordPress, 24 Oct. 2015.
[22] Engineering Funda, "PWM Inverter (Working, Principle, 3-phase Inverter, Waveform, Sine PWM inverter) Engineering Funda," YouTube, 27 April 2017.
[23] Mohankumar, D. “PWM Inverter.” *ElectroSchematics.com*, AspenCore, Inc., 7 July 2010.
[24] *9A-Peak Low-Side MOSFET Driver*, MIC4422, Micrel, August 2005, pp. 1-12.
[25] *140 Watt TO-247 Power Resistor*, TEH140, Ohmite, 2020, pp. 1.
[26] LearningaboutElectronics, "What is a Smoothing Capacitor?,” LearningaboutElectronics, 2018.
[27] Renu Sharma, Deepak Kumar Goyal, and Harpreet Singh, "Analysis of Space Vector PWM for Three Phase Inverter and Comparison with SPWM," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, Vol. 5, Issue 1, January 2016.
[28] Beurnii, "How to chose a mosfet driver," *Arduino Forum*, Arduino, 16 July 2015.
[29] g4ce, "Guidance for MOSFET circuit to increase PWM voltage level," *Arduino Forum*, Arduino, 21 July 2017.
[30] Diodes Incorporated. “Gate Drivers.” *Diodes Incorporated*, Diodes Incorporated, 2020.
[31] Small Signal Fast Switching Diodes, 1N4148, Vishay Semiconductors, 6 July 2017, pp. 1.
[32] Prasad, Sai. “How to Decide the Cut off Frequency of an LC Filter for a Spwm?” *Electrical Engineering Stack Exchange*, Stack Exchange Inc., 6 Dec. 2017.

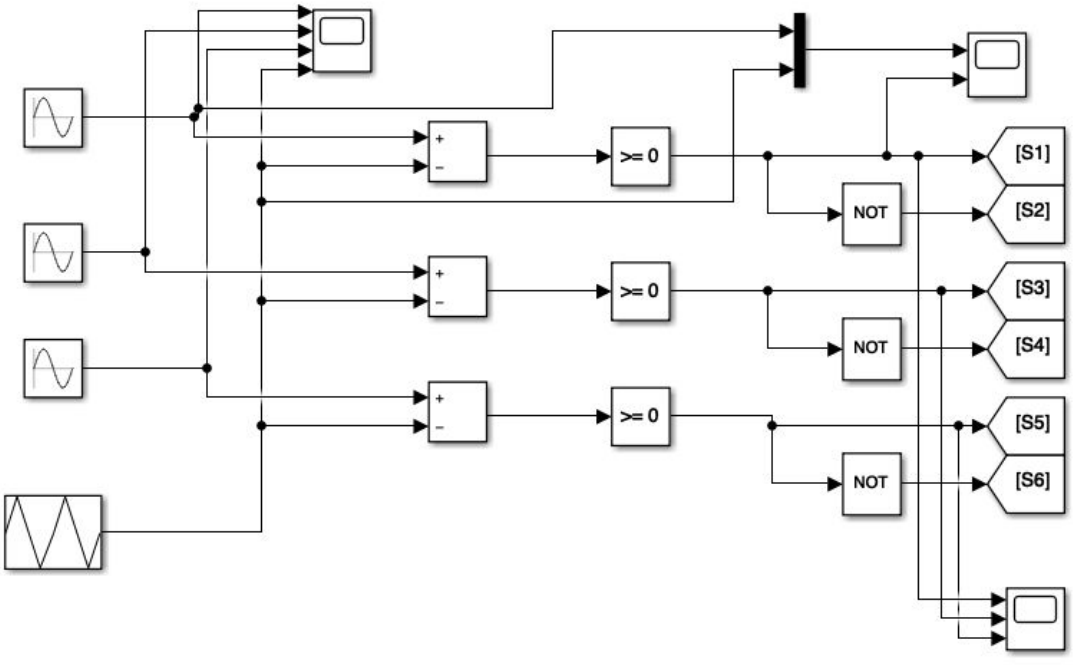
APPENDICES

```
clc, clear all, close all

fs = 1E5;
t2 = 1/fs;
time = 0:t2:0.02;
f = 50;
last = length(time);

%Graph for Triangle and Three-phase Sinusoidal Waves
triangle = sawtooth(2*pi*500*(time+5E-4),0.5);
sinusoidal = 0.8*sin(2*pi*f*time);
sinusoidal2 = 0.8*sin((2*pi*f*time)+(120*pi/180));
sinusoidal3 = 0.8*sin((2*pi*f*time)+(240*pi/180));
figure(1)
plot(time, triangle)
hold on
plot(time, sinusoidal)
hold on
plot(time, sinusoidal2)
hold on
plot(time, sinusoidal3)
grid on
title('Triangle and Three-phase Sinusoidal Waves')

%PWM for 0-Degree Phase Shifted
for var = 1:last
   diff = sinusoidal(var)-triangle(var);
   if diff > 0
      pwm(var) = 1;
   elseif diff == 0
      pwm(var) = 0;
   else
      pwm(var) = -1;
   end
   var = var + t2;
```

```
end
figure(2)
plot(time, pwm)
grid on
title('PWM for 0-Degree Phase Shifted Sinusoidal Wave')

%PWM for 120-Degree Phase Shifted
for var = 1:last
   diff = sinusoidal2(var)-triangle(var);
   if diff > 0
      pwm2(var) = 1;
   elseif diff == 0
      pwm2(var) = 0;
   else
      pwm2(var) = -1;
   end
   var = var + t2;
end
figure(3)
plot(time, pwm2)
grid on
title('PWM for 120-Degree Phase Shifted Sinusoidal Wave')

%PWM for 240-Degree Phase Shifted
for var = 1:last
   diff = sinusoidal3(var)-triangle(var);
   if diff > 0
      pwm3(var) = 1;
   elseif diff == 0
      pwm3(var) = 0;
   else
      pwm3(var) = -1;
   end
   var = var + t2;
end
figure(4)
plot(time, pwm3)
grid on
```

```
title('PWM for 240-Degree Phase Shifted Sinusoidal Wave')

% PWM for all Phases
figure(5)
plot(time, pwm)
hold on
plot(time, pwm2)
hold on
plot(time, pwm3)
grid on
title('PWM for all Phases')
axis([0 0.02 -1 1.1])
```

Appendix Figure 1: SPWM for inverter circuit



Appendix Figure 2: SPWM technique in Simulink

```
int MOSFET1 = 5; //MOSFET1 connected to pin
int MOSFET3 = 10; //MOSFET3 connected to pin
```

```
int MOSFET5  = 11; //MOSFET3 connected to pin

int MOSFETarray [] = {0, 32, 71, 89, 140, 160, 214, 226, 270, 296, 335, 367, 392, 439, 461,
519, 523, 572, 590, 636, 666, 697, 739, 761, 818, 821, 876, 886, 938, 963, 999, 1036, 1063,
1115, 1125, 1178, 1181, 1238, 1262, 1304, 1335, 1363, 1409, 1427, 1478, 1482, 1540, 1560,
1607, 1632, 1666, 1705, 1731, 1773, 1785, 1839, 1861, 1912, 1930, 1967, 2000};

void setup() {
  // put your setup code here, to run once:
pinMode(MOSFET1, OUTPUT); //sets digital pin  as output
pinMode(MOSFET3, OUTPUT); //sets digital pin  as output
pinMode(MOSFET5, OUTPUT); //sets digital pin  as output
}

void loop() {
  // put your main code here, to run repeatedly:
//before end of blue line
//MOS3 ON
digitalWrite(MOSFET1, LOW); //MOSFET1 off
digitalWrite(MOSFET5, LOW); //MOSFET5 off
digitalWrite(MOSFET3, HIGH); //MOSFET3 on
delayMicroseconds(MOSFETarray[1]*10);
//MOS3 OFF
digitalWrite(MOSFET3, LOW);
delayMicroseconds((MOSFETarray[2]-MOSFETarray[1])*10);
//MOS3 ON
digitalWrite(MOSFET3, HIGH);
delayMicroseconds((MOSFETarray[3]-MOSFETarray[2])*10);
//MOS1 ON
digitalWrite(MOSFET1, HIGH);
delayMicroseconds((MOSFETarray[4]-MOSFETarray[3])*10);
//MOS5 ON
digitalWrite(MOSFET5, HIGH);
delayMicroseconds((MOSFETarray[5]-MOSFETarray[4])*10);
//MOS5 OFF
digitalWrite(MOSFET5, LOW);
delayMicroseconds((MOSFETarray[6]-MOSFETarray[5])*10);
//MOS3 OFF
```

```
digitalWrite(MOSFET3, LOW);
delayMicroseconds((MOSFETarray[7]-MOSFETarray[6])*10);
//MOS1 OFF
digitalWrite(MOSFET1, LOW);
delayMicroseconds((MOSFETarray[8]-MOSFETarray[7])*10);

//second
//MOS1 ON
digitalWrite(MOSFET1, HIGH);
delayMicroseconds((MOSFETarray[9]-MOSFETarray[8])*10);
//MOS3 ON
digitalWrite(MOSFET3, HIGH);
delayMicroseconds((MOSFETarray[10]-MOSFETarray[9])*10);
//MOS5 ON
digitalWrite(MOSFET5, HIGH);
delayMicroseconds((MOSFETarray[11]-MOSFETarray[10])*10);
//MOS5 OFF
digitalWrite(MOSFET5, LOW);
delayMicroseconds((MOSFETarray[12]-MOSFETarray[11])*10);
//MOS3 OFF
digitalWrite(MOSFET3, LOW);
delayMicroseconds((MOSFETarray[13]-MOSFETarray[12])*10);
//MOS1 OFF
digitalWrite(MOSFET1, LOW);
delayMicroseconds((MOSFETarray[14]-MOSFETarray[13])*10);

//third
//MOS1 ON
digitalWrite(MOSFET1, HIGH);
delayMicroseconds((MOSFETarray[15]-MOSFETarray[14])*10);
//MOS5 ON
digitalWrite(MOSFET5, HIGH);
delayMicroseconds((MOSFETarray[16]-MOSFETarray[15])*10);
//MOS3 ON
digitalWrite(MOSFET3, HIGH);
delayMicroseconds((MOSFETarray[17]-MOSFETarray[16])*10);
//MOS3 OFF
digitalWrite(MOSFET3, LOW);
```

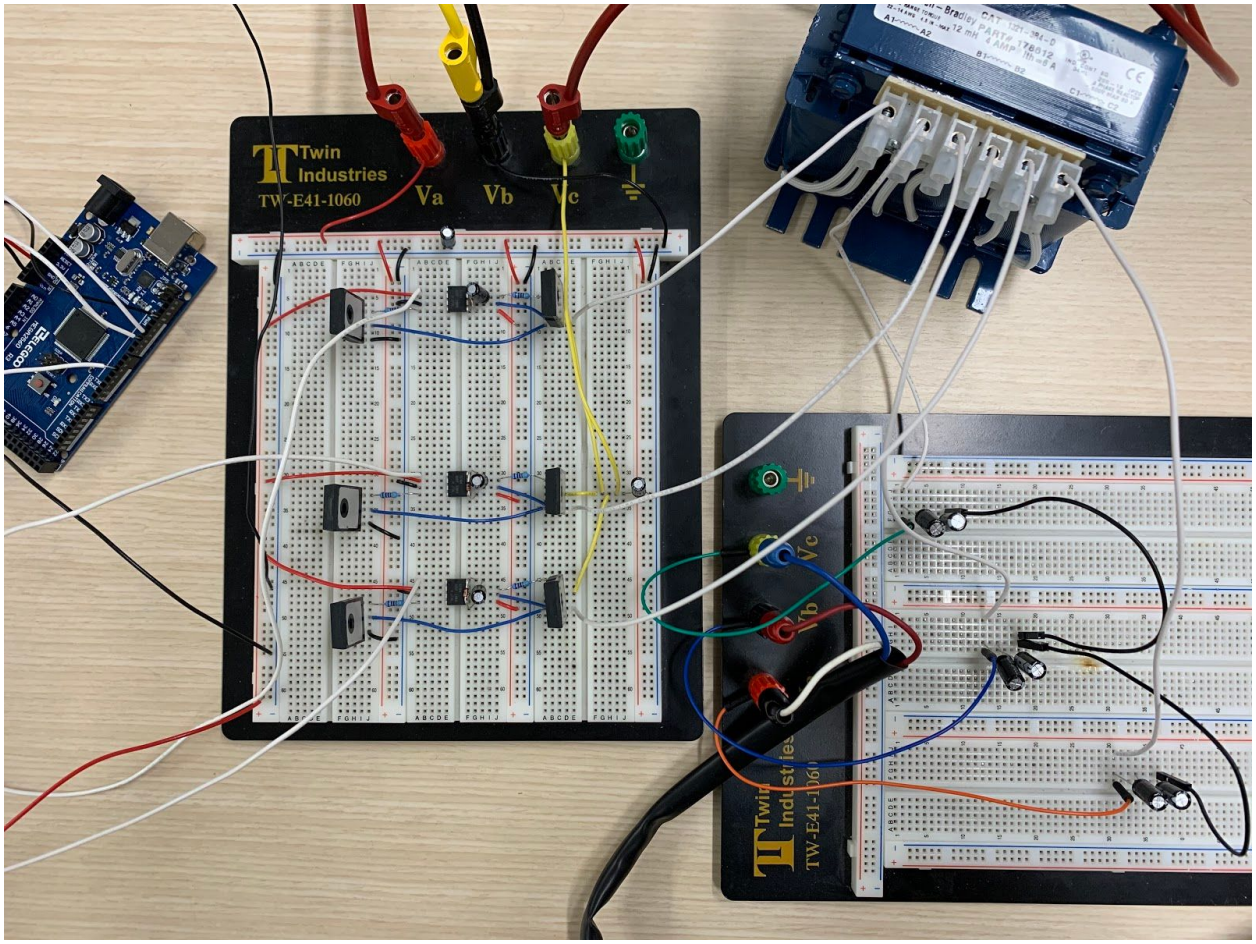
```
delayMicroseconds((MOSFETarray[18]-MOSFETarray[17])*10);
//MOS5 OFF
digitalWrite(MOSFET5, LOW);
delayMicroseconds((MOSFETarray[19]-MOSFETarray[18])*10);
//MOS1 OFF
digitalWrite(MOSFET1, LOW);
delayMicroseconds((MOSFETarray[20]-MOSFETarray[19])*10);

//fourth
//MOS1 ON
digitalWrite(MOSFET1, HIGH);
delayMicroseconds((MOSFETarray[21]-MOSFETarray[20])*10);
//MOS5 ON
digitalWrite(MOSFET5, HIGH);
delayMicroseconds((MOSFETarray[22]-MOSFETarray[21])*10);
//MOS3 ON
digitalWrite(MOSFET3, HIGH);
delayMicroseconds((MOSFETarray[23]-MOSFETarray[22])*10);
//MOS3 OFF
digitalWrite(MOSFET3, LOW);
delayMicroseconds((MOSFETarray[24]-MOSFETarray[23])*10);
//MOS5 OFF
digitalWrite(MOSFET5, LOW);
delayMicroseconds((MOSFETarray[25]-MOSFETarray[24])*10);
//MOS1 OFF
digitalWrite(MOSFET1, LOW);
delayMicroseconds((MOSFETarray[26]-MOSFETarray[25])*10);

//fifth
//MOS5 ON
digitalWrite(MOSFET5, HIGH);
delayMicroseconds((MOSFETarray[27]-MOSFETarray[26])*10);
//MOS1 ON
digitalWrite(MOSFET1, HIGH);
delayMicroseconds((MOSFETarray[28]-MOSFETarray[27])*10);
//MOS3 ON
digitalWrite(MOSFET3, HIGH);
delayMicroseconds((MOSFETarray[29]-MOSFETarray[28])*10);
```

```
//MOS3 OFF
digitalWrite(MOSFET3, LOW);
delayMicroseconds((MOSFETarray[30]-MOSFETarray[29])*10);
//MOS1 OFF
digitalWrite(MOSFET1, LOW);
delayMicroseconds((MOSFETarray[31]-MOSFETarray[30])*10);

//sixth
//MOS5 OFF
digitalWrite(MOSFET5, LOW);
delayMicroseconds((MOSFETarray[32]-MOSFETarray[31])*10);
//MOS5 ON
digitalWrite(MOSFET5, HIGH);
delayMicroseconds((MOSFETarray[33]-MOSFETarray[32])*10);
//MOS1 ON
digitalWrite(MOSFET1, HIGH);
delayMicroseconds((MOSFETarray[34]-MOSFETarray[33])*10);
//MOS3 ON
digitalWrite(MOSFET3, HIGH);
delayMicroseconds((MOSFETarray[35]-MOSFETarray[34])*10);
//MOS1 OFF
digitalWrite(MOSFET1, LOW);
delayMicroseconds((MOSFETarray[36]-MOSFETarray[35])*10);

//seventh
//MOS3 OFF
digitalWrite(MOSFET3, LOW);
delayMicroseconds((MOSFETarray[37]-MOSFETarray[36])*10);
//MOS5 OFF
digitalWrite(MOSFET5, LOW);
delayMicroseconds((MOSFETarray[38]-MOSFETarray[37])*10);
//MOS5 ON
digitalWrite(MOSFET5, HIGH);
delayMicroseconds((MOSFETarray[39]-MOSFETarray[38])*10);
//MOS3 ON
digitalWrite(MOSFET3, HIGH);
delayMicroseconds((MOSFETarray[40]-MOSFETarray[39])*10);
//MOS1 ON
```

```
digitalWrite(MOSFET1, HIGH);
delayMicroseconds((MOSFETarray[41]-MOSFETarray[40])*10);
//MOS1 OFF
digitalWrite(MOSFET1, LOW);
delayMicroseconds((MOSFETarray[42]-MOSFETarray[41])*10);

//eighth
//MOS3 OFF
digitalWrite(MOSFET3, LOW);
delayMicroseconds((MOSFETarray[43]-MOSFETarray[42])*10);
//MOS5 OFF
digitalWrite(MOSFET5, LOW);
delayMicroseconds((MOSFETarray[44]-MOSFETarray[43])*10);
//MOS5 ON
digitalWrite(MOSFET5, HIGH);
delayMicroseconds((MOSFETarray[45]-MOSFETarray[44])*10);
//MOS3 ON
digitalWrite(MOSFET3, HIGH);
delayMicroseconds((MOSFETarray[46]-MOSFETarray[45])*10);
//MOS1 ON
digitalWrite(MOSFET1, HIGH);
delayMicroseconds((MOSFETarray[47]-MOSFETarray[46])*10);
//MOS1 OFF
digitalWrite(MOSFET1, LOW);
delayMicroseconds((MOSFETarray[48]-MOSFETarray[47])*10);

//NINTH
//MOS5 OFF
digitalWrite(MOSFET5, LOW);
delayMicroseconds((MOSFETarray[49]-MOSFETarray[48])*10);
//MOS3 OFF
digitalWrite(MOSFET3, LOW);
delayMicroseconds((MOSFETarray[50]-MOSFETarray[49])*10);
//MOS3 ON
digitalWrite(MOSFET3, HIGH);
delayMicroseconds((MOSFETarray[51]-MOSFETarray[50])*10);
//MOS5 ON
digitalWrite(MOSFET5, HIGH);
```

```
delayMicroseconds((MOSFETarray[52]-MOSFETarray[51])*10);
//MOS1 ON
digitalWrite(MOSFET1, HIGH);
delayMicroseconds((MOSFETarray[53]-MOSFETarray[52])*10);
//MOS1 OFF
digitalWrite(MOSFET1, LOW);
delayMicroseconds((MOSFETarray[54]-MOSFETarray[53])*10);

//TENTH
//MOS5 OFF
digitalWrite(MOSFET5, LOW);
delayMicroseconds((MOSFETarray[55]-MOSFETarray[54])*10);
//MOS3 OFF
digitalWrite(MOSFET3, LOW);
delayMicroseconds((MOSFETarray[56]-MOSFETarray[55])*10);
//MOS3 ON
digitalWrite(MOSFET3, HIGH);
delayMicroseconds((MOSFETarray[57]-MOSFETarray[56])*10);
//MOS1 ON
digitalWrite(MOSFET1, HIGH);
delayMicroseconds((MOSFETarray[58]-MOSFETarray[57])*10);
//MOS5 ON
digitalWrite(MOSFET5, HIGH);
delayMicroseconds((MOSFETarray[59]-MOSFETarray[58])*10);
//MOS5 OFF
digitalWrite(MOSFET5, LOW);
delayMicroseconds((MOSFETarray[60]-MOSFETarray[59])*10);
}
```

Appendix Figure 3: Arduino code for inverter circuit



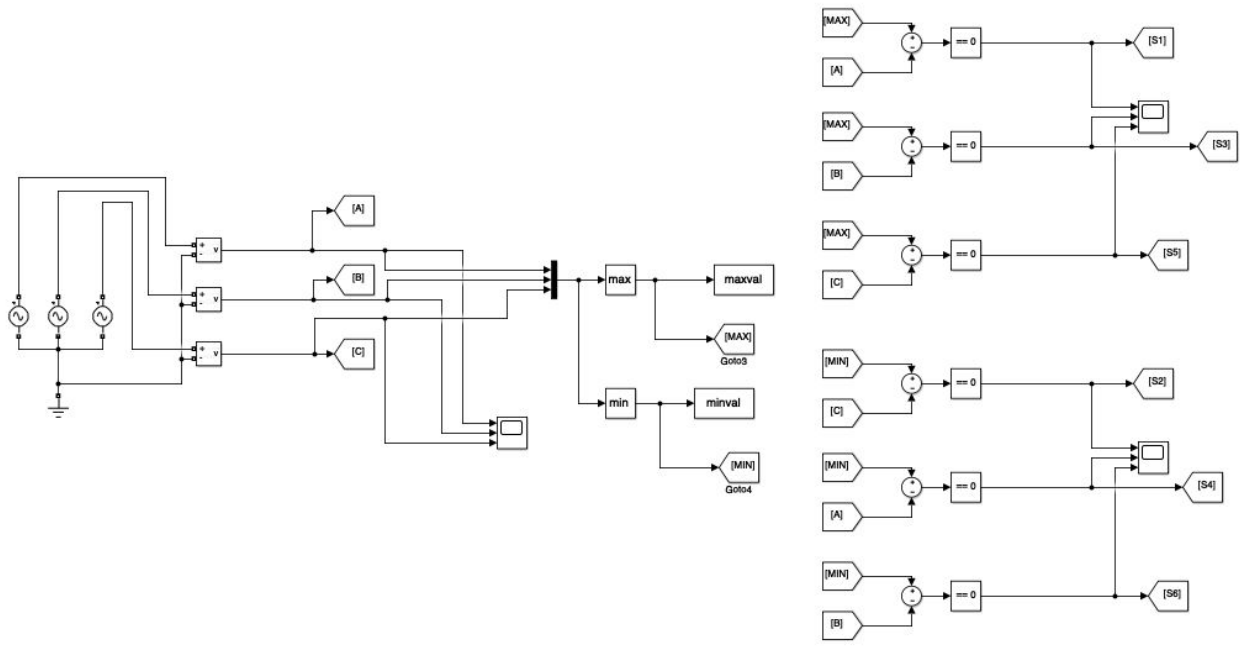
Appendix Figure 4: Final inverter circuit with LC filter

```
clc, clear all, close all

fs = 1E5;
t2 = 1/fs;
time = 0:t2:0.02;
f = 50;
last = length(time);

%Graph for Three-phase Sinusoidal Waves
a = 0.8*sin(2*pi*f*time);
b = 0.8*sin((2*pi*f*time)+(120*pi/180));
c = 0.8*sin((2*pi*f*time)+(240*pi/180));
```

```
figure(1)
plot(time, a)
hold on
plot(time, b)
hold on
plot(time, c)
grid on
title('Three-phase Sinusoidal Waves')

vec = zeros([1,last]);
MOS1 = vec;
MOS2 = vec;
MOS3 = vec;
MOS4 = vec;
MOS5 = vec;
MOS6 = vec;

for v = 1:last
   calc = a(v) - b(v);
   calc2 = a(v) - c(v);
   calc3 = b(v) - c(v);
   if calc > 0
      if calc2 > 0
         MOS1(v) = 1;
         if calc3 > 0
            MOS2(v) = -1;
         elseif calc3 < 0
            MOS6(v) = -1;
         end
      elseif calc2 < 0
         MOS5(v) = 1;
         MOS6(v) = -1;
      end
   elseif calc < 0
      if calc2 > 0
         MOS3(v) = 1;
         MOS2(v) = -1;
      elseif calc2 < 0
```

```
        MOS4(v) = -1;
        if calc3 > 0
            MOS3(v) = 1;
        elseif calc3 < 0
            MOS5(v) = 1;
        end
      end
   end
end

figure(2)
plot(time, MOS1)
hold on
plot(time, MOS2)
hold on
plot(time, MOS3)
hold on
plot(time, MOS4)
hold on
plot(time, MOS5)
hold on
plot(time, MOS6)
grid on
```

Appendix Figure 5: MATLAB calculations for rectifier circuit

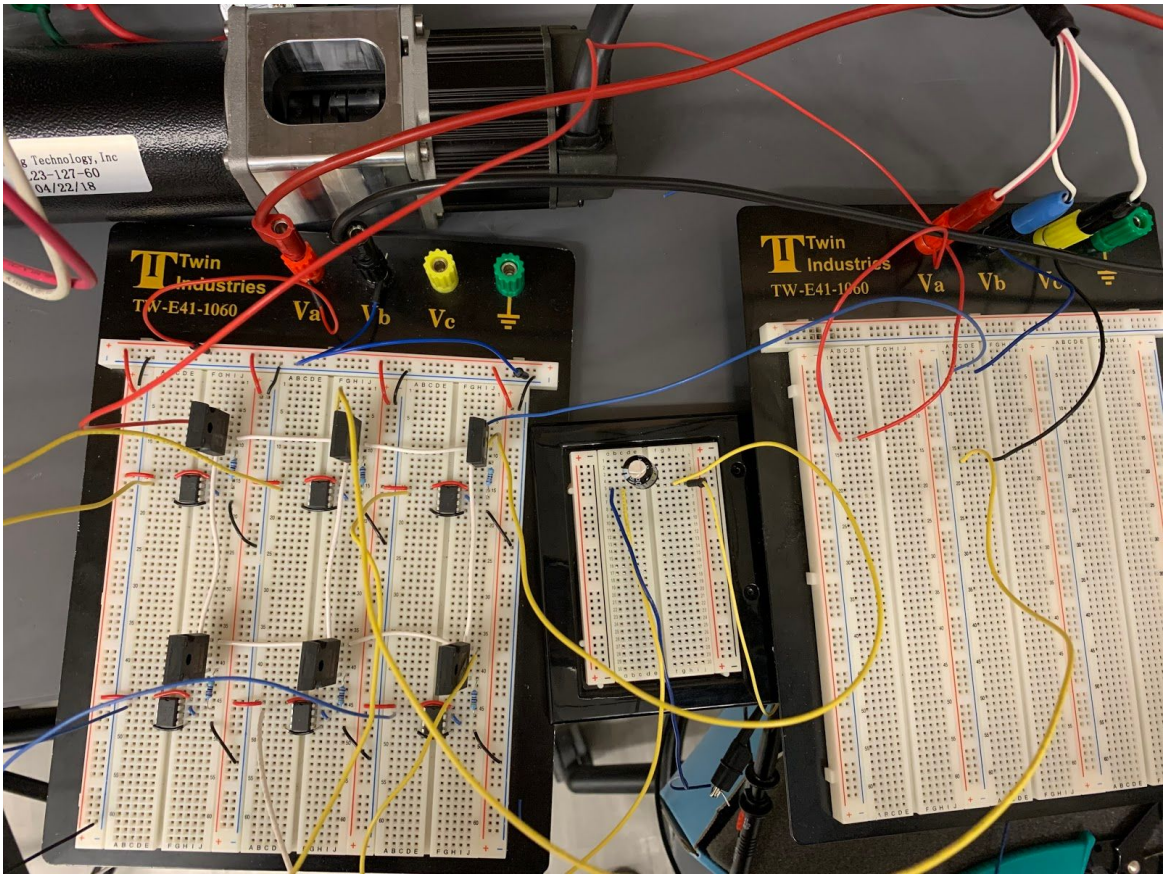


Appendix Figure 6: Simulink calculations for rectifier circuit

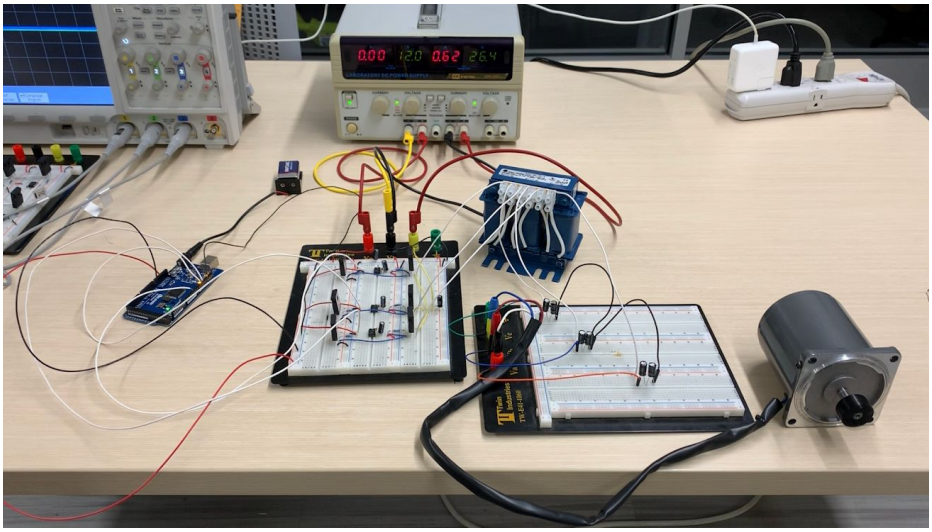
```
int MOSFET1 = 3; //MOSFET1 connected to pin
int MOSFET2 = 5; //MOSFET2 connected to pin
int MOSFET3 = 6; //MOSFET3 connected to pin
int MOSFET4 = 9; //MOSFET4 connected to pin
int MOSFET5  = 10; //MOSFET5 connected to pin
int MOSFET6 = 11; //MOSFET6 connected to pin

int RECTarray [] = {0,167,500,833,1167,1500,1834,2000};

void setup() {
  // put your setup code here, to run once:
pinMode(MOSFET1, OUTPUT); //sets digital pin  as output
pinMode(MOSFET2, OUTPUT); //sets digital pin  as output
pinMode(MOSFET3, OUTPUT); //sets digital pin  as output
pinMode(MOSFET4, OUTPUT);
pinMode(MOSFET5, OUTPUT);
pinMode(MOSFET6, OUTPUT);
}
void loop() {
```

```
  // put your main code here, to run repeatedly:
digitalWrite(MOSFET1, LOW);
digitalWrite(MOSFET2, LOW);
digitalWrite(MOSFET4, LOW);
digitalWrite(MOSFET5, LOW);
digitalWrite(MOSFET6, LOW);

digitalWrite(MOSFET3, HIGH);
delayMicroseconds(RECTarray[1]*10);

digitalWrite(MOSFET3, LOW);
digitalWrite(MOSFET1, HIGH);
digitalWrite(MOSFET2, HIGH);
delayMicroseconds((RECTarray[2]-RECTarray[1])*10);
digitalWrite(MOSFET2, LOW);
digitalWrite(MOSFET6, HIGH);
delayMicroseconds((RECTarray[3]-RECTarray[2])*10);
digitalWrite(MOSFET1, LOW);
digitalWrite(MOSFET5, HIGH);
delayMicroseconds((RECTarray[4]-RECTarray[3])*10);
digitalWrite(MOSFET6, LOW);
digitalWrite(MOSFET4, HIGH);
delayMicroseconds((RECTarray[5]-RECTarray[4])*10);
digitalWrite(MOSFET5, LOW);
digitalWrite(MOSFET3, HIGH);
delayMicroseconds((RECTarray[6]-RECTarray[5])*10);
digitalWrite(MOSFET4, LOW);
digitalWrite(MOSFET2, HIGH);
delayMicroseconds((RECTarray[7]-RECTarray[6])*10);
}
```

Appendix Figure 7: Arduino code for rectifier circuit



Appendix Figure 8: Final rectifier circuit with capacitor filter



Appendix Figure 9: Final inverter circuit implementation

```
int a = ;
int b = ;
int c = ;

int MOSFET1 = ; //MOSFET1 connected to pin
int MOSFET2 = ; //MOSFET2 connected to pin
int MOSFET3 = ; //MOSFET3 connected to pin
int MOSFET4 = ; //MOSFET4 connected to pin
int MOSFET5  = ; //MOSFET5 connected to pin
int MOSFET6 = ; //MOSFET6 connected to pin

void setup() {
  // put your setup code here, to run once:
  pinMode(a, INPUT);
  pinMode(b, INPUT);
  pinMode(c, INPUT);
  pinMode(MOSFET1, OUTPUT); //sets digital pin  as output
  pinMode(MOSFET2, OUTPUT); //sets digital pin  as output
  pinMode(MOSFET3, OUTPUT); //sets digital pin  as output
  pinMode(MOSFET4, OUTPUT); //sets digital pin  as output
  pinMode(MOSFET5, OUTPUT); //sets digital pin  as output
  pinMode(MOSFET6, OUTPUT); //sets digital pin  as output
}

void loop() {
  // put your main code here, to run repeatedly:
  A_H = digitalRead(a, HIGH);
  B_H = digitalRead(b, HIGH);
  C_H = digitalRead(b, HIGH);
  A_L = digitalRead(a, LOW);
  B_L = digitalRead(b, LOW);
  C_L = digitalRead(b, LOW);


  calc = a - b;
  calc2 = a - c;
  calc3 = b - c;
  if calc > 0
    if calc2 > 0
      digitalWrite(MOSFET1, HIGH);
      if calc3 > 0
        digitalWrite(MOSFET2, HIGH);
      elseif calc3 < 0
        digitalWrite(MOSFET6, HIGH);
```

```
    end
  elseif calc2 < 0
    digitalWrite(MOSFET5, HIGH);
    digitalWrite(MOSFET6, HIGH);
  end
 elseif calc < 0
  if calc2 > 0
    digitalWrite(MOSFET3, HIGH);
    digitalWrite(MOSFET2, HIGH);
  elseif calc2 < 0
    digitalWrite(MOSFET4, HIGH);
    if calc3 > 0
      digitalWrite(MOSFET3, HIGH);
    elseif calc3 < 0
      digitalWrite(MOSFET5, HIGH);
    end
  end
 end
}
```

Appendix Figure 10: Arduino code for real-time rectifier circuit