Design Strategy for a
3-Phase Variable Frequency Drive (VFD)

By

Omar David Muñoz

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

2011

Table of Contents

List of Figures

Abstract

This paper describes the design processes for a 3-Phase Variable Frequency Drive (VFD) as broken up into two stages: The AC-DC converter and the DC-AC converter. It acknowledges three (3) design versions, all. The output of the motor drive is 3-phase pulse width modulation (PWM) ranging in possible operation frequencies from 0Hz to 13kHz. The frequency control circuit uses an Arduino Uno based of the ATMEGA 328p.

Acknowledgements

I would like to thank both Michael Marshall and Stanley Ho for their assistance in code development with the Arduino; as well as Professor Slivosky for her assistance in algorithm development for the code implemented in the micro controller.

I. INTRODUCTION

THIS document describes the design road taken when looking at a 3-phase Variable Frequency Drive (VFD).

These motor drives are designed to be used in conjunction with a 3-phase induction motor. Because these motors typically only have an on/off state of operation, a VFD is needed if multiple operation speeds are desired. Also, apart from selectable speeds, the efficiency of the overall system is increased due to the fact that the motor only sees the necessary amount of input power to achieve desirable output power. Also, the motor can be slowly brought up to speed, eliminating huge start-up current spikes.

For simplistic Purposes, this motor drive will be split into two main components: the AC-DC and the DC-AC converters (which will contain the control circuit). The first part of the drive creates a DC source which is used by stage 2. In the second stage, the AC signal is reconstructed via pulse width modulation (PWM). The DC-AC converter (H-Bridge) is controlled by a preprogrammed micro-controller (Arduino Uno development board with a central ATMEGA 328p micro-controller).

An H-Bridge functions by controlling switches (in this case fully controlled bi-directional switches, MOSFETs). By alternating which switches are on, voltage and current flow can be directed in any direction desired. For this particular case, we are interested in producing a PWM signal that will be produced by toggling a switch ON/OFF for the positive portion of a sin wave, and doing a similar operation to a separate switch for the negative portion of a sin wave.

The output of the motor drive is a PWM signal with fluctuating frequency. The frequency changes in respect to the input to the Arduino Uno. As the potentiometer (analog control) changes the output frequency changes. This in turn allows for lower or higher frequencies than the standard 60Hz to motor. The design used is ideal due to the thermal properties of the system. If the desired output of the drive is a sin wave, fast switching is required for high resolution; this would mean that the switches would overheat after a short operation. The fact that most of the switching loss seen at the transition time between on and off period due to the power loss by R_{DS-ON}. If the switching is slowed to produce a sin wave at lower resolution, the motor would step operate in short burst corresponding to the stepping of the signal. This is hard on the motor and would damage both windings and bearings.

The solution to this problem is to produce PWM and exploit the inductive properties of the motor. The PWM allows for less switching time for the MOSFETs and consequently less power loss. During peak power, there is minimal switching, translating into less loss. When the PWM signal is fed to the motor, it acts as a low pass filter and the current flow due to the PWM now becomes sinusoidal. Because there is no "stepping" in the current, once turned over the motor will not stutter and will operate smoothly.

II. HARDWARE DESIGN OVERVIEW (VERSION 1)

A. AC-DC Converter

The first portion of the motor drive takes the AC input signal and rectifies it, sending the positive signal to a capacitor bank. The full-wave rectifier is connected as seen below in figure 2.1.
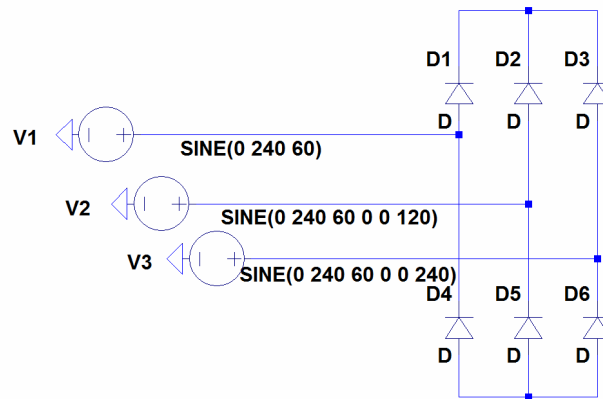

Figure 2.1 Rectifier circuit
(discreet component design)

For simplicity, I have opted to use a pre-packaged 3-phase rectifier from Vishay/Semiconductors rated beyond the 240 3-phase signal voltage used in the application. (part number 36MT80). From here, fully rectified signal is produced and then fed to the capacitor bank. This bank needs to be rated beyond 340V which is found by Equation 2.1

Voltage Rating: V_{RMS}*1.414
Current Rating: same as Diode current rating

Equation 2.1 rating for the capacitor is given
by the peak voltage value that will be
seen by the capacitor bank

This can either be obtained from building a capacitor bank or using one large capacitor capable of meeting the needed criteria. Also note that the smaller the Equivalent Series Resistance (ESR) the less loss encountered, also the smaller the need for increased current rating compensation. Once again, for simplicity, I have opted to use 2 Vishay/BC Components 470uF aluminum capacitors rated at 400V DC and 3.1A ripple current. Because the expected ripple current for this application is about 4A (as referenced by Square D Products), 2 capacitors will be stacked in parallel to improve current rating by two. When looked at together, the diode bridge and the capacitor bank make a DC source with little to no voltage ripple.

NOTE: The capacitors used for this application can be electrolytic, but they may never be fully discharged, as this could damage the components due to the limited charge/recharge cycles of the component.
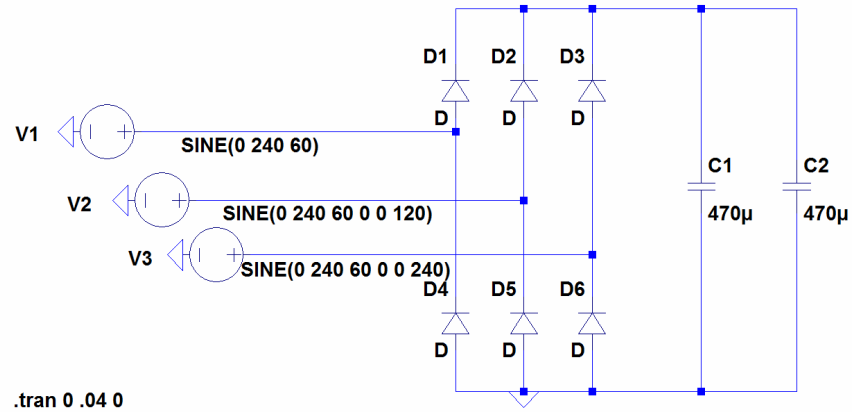
Figure 2.2 AC-DC Converter Circuit: Circuit for the
AC-DC converter on a discrete component level

B. DC-AC Converter

An H-bridge converter using six N-channel MOSFETs is used. The configuration is as seen in Figure 2.3. A vertical pair of switches corresponds to a single phase. The Gate of the MOSFETs is connected to ON/OFF signals which control the PWM output. All of the signals going into the gate are separate connections.
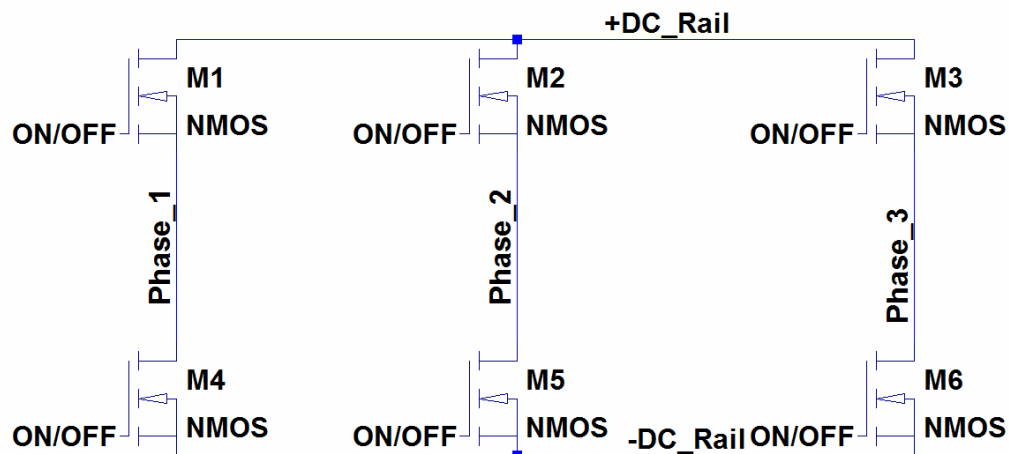
Figure 2.3 H-bridge Configuration: terminals have
been labeled to identify phase

NOTE: that there is no ground reference for the switches as the positive and negative DC rails come from the capacitor.

The junction between the source and drain of the upper and lower MOSFET is connected to the load, in this case an induction motor. Because the motor has a ground reference, when the top switch opens, the voltage at the motor is positive (+). When the bottom MOSFET is open, the motor sees a negative (-) voltage signal. Figure 2.4 shows the flow of current, 2.4 (top) depicts positive flow, 2.4 (bottom) depicts negative flow.
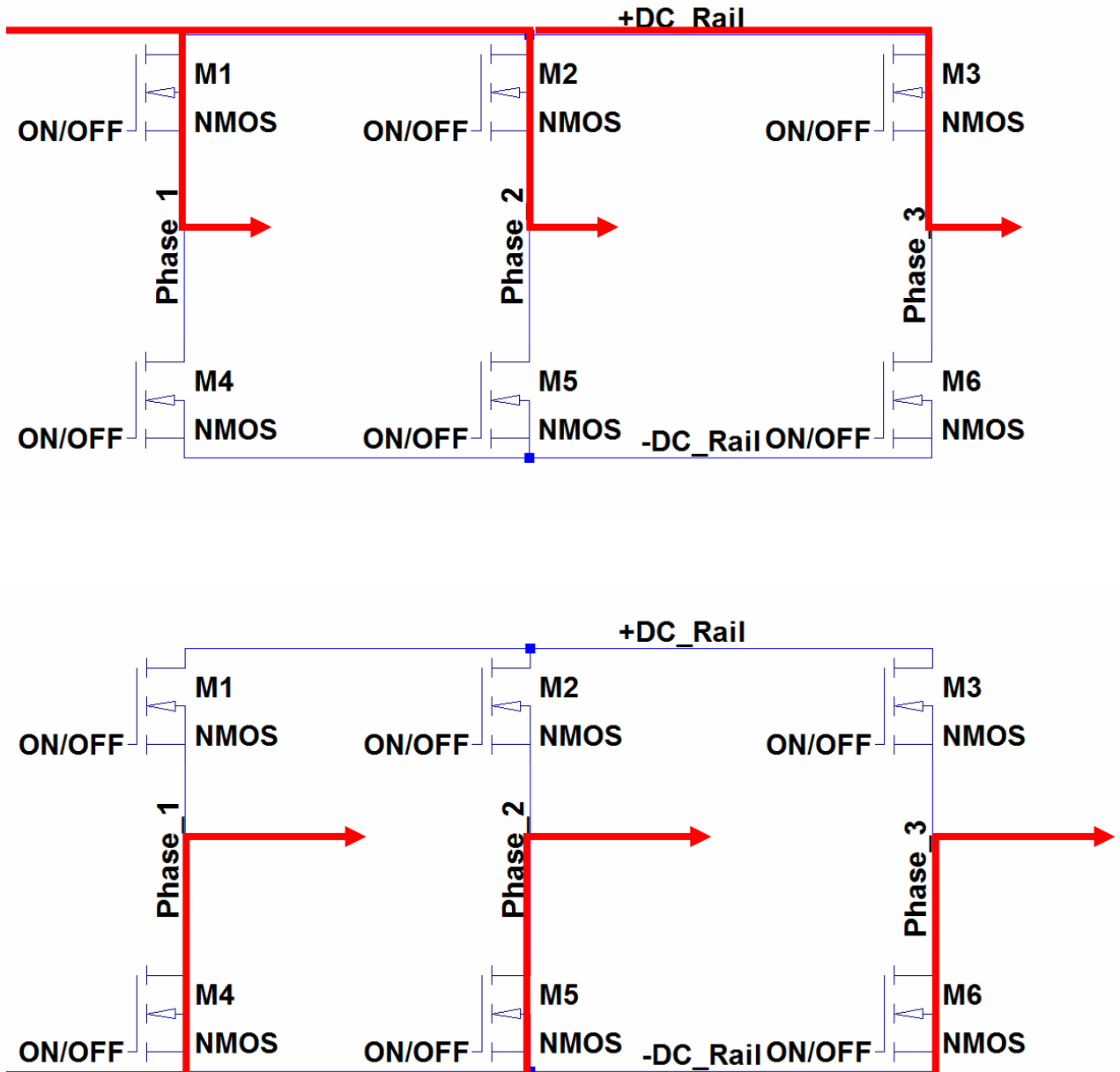
Figure 2.4 H-Bridge Current Flow:
(top) positive current flow,
2.4 (bottom) negative current flow

NOTE: It is important the H-Bridge does not short the source. This would happen if two pairs (top and bottom) of MOSFETs are open at once. In order to avoid this problem, sufficient dead time must be allotted.

C. DC-AC Controller

An Arduino Uno will be used to drive the gates at a logic output voltage of 5V. Turn on times will be determined by a look up table. These values of this table are synthesized using the circuit in Figure 2.5.
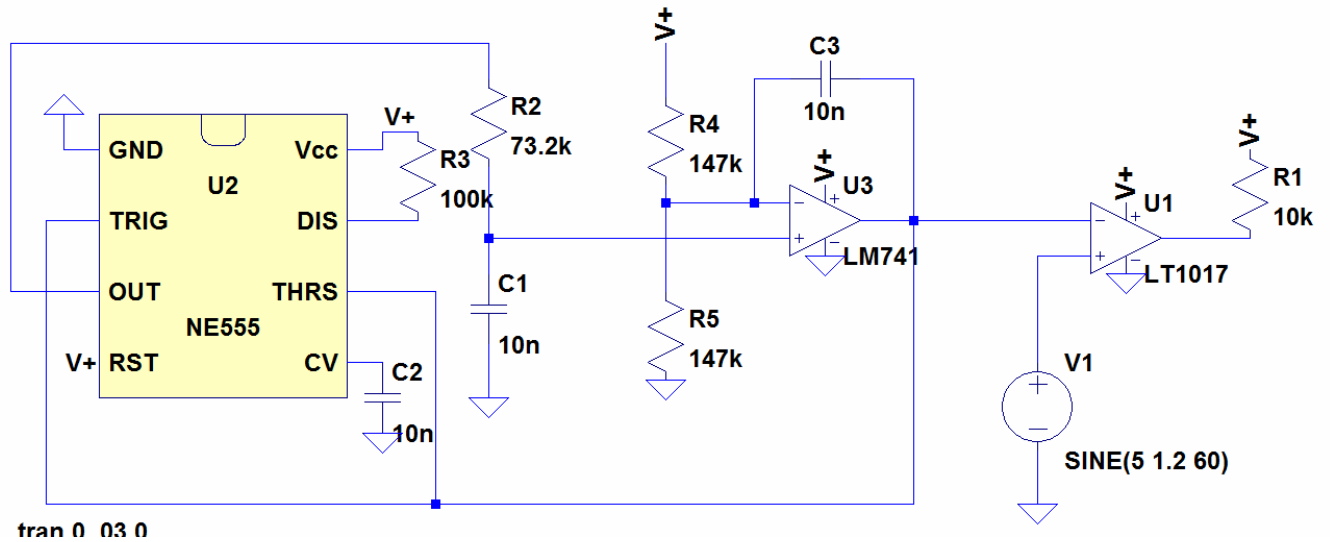
Figure 2.5 PWM Generation Circuit: The circuit consist of a
comparator with a triangle wave and a sin wave for inputs,
the output of the circuit is PWM.

This circuit runs a triangle wave at a frequency of 1kHz and a sin wave varying in frequencies of 40Hz, 50Hz, 60 Hz, and 70Hz through a comparator. The output from this comparator is a PWM signal. When filtered it yields the sin wave input to the comparator. See Figure 2.6 below for the simulation circuit and the output at different frequencies.
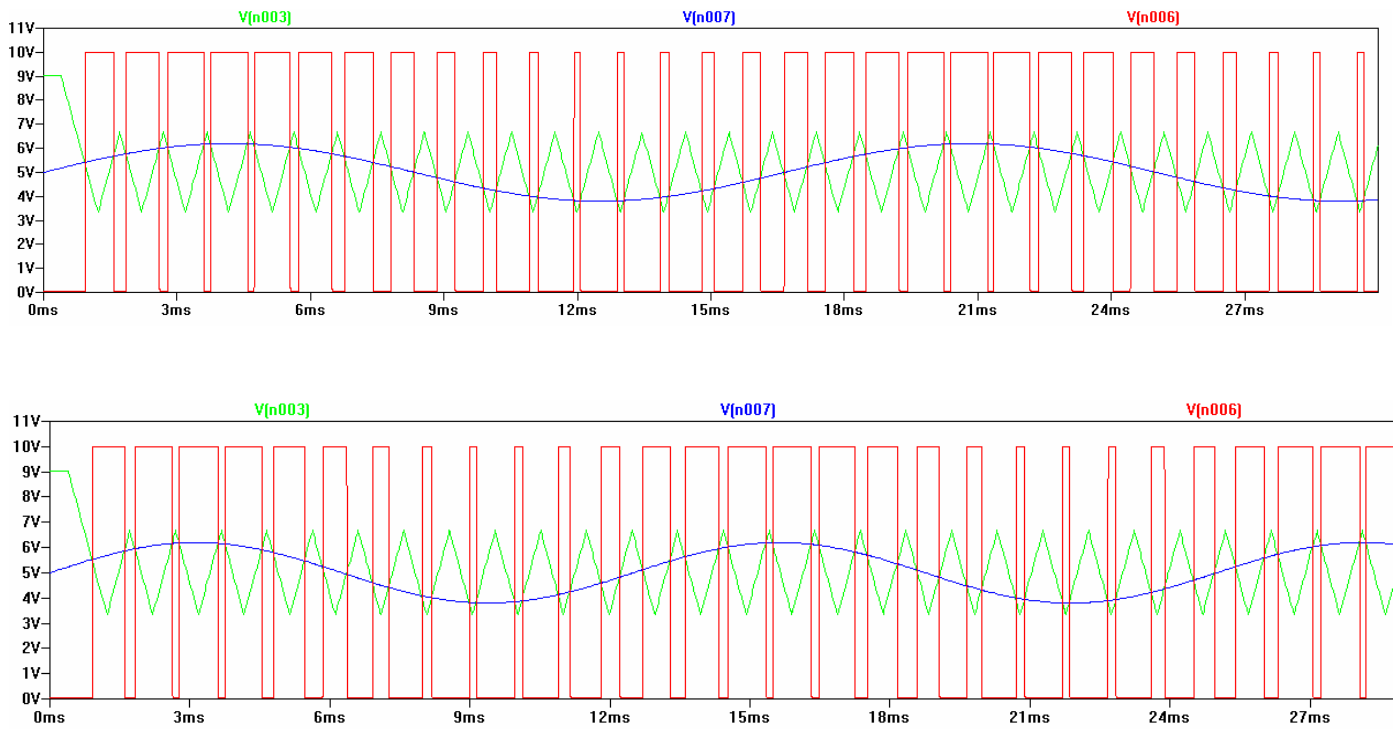
Figure 2.6 PWM Simulation Results:
(top) 60Hz sin input
(Bottom) 80Hz sin input
Input: Blue Sin wave, Green Triangle wave
Output: Red Pulse Width Modulation

The PWM circuit compares the two inputs. If they were overlapped, when $V_{tri}<V_{sin}$ then the PWM is high until the transition to $V_{tri}>V_{sin}$ at which time the PWM is low.
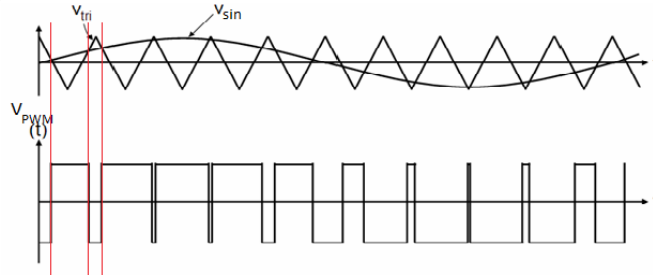
Figure 2.7 PWM Example: Visual representation of
Signal processing at the voltage comparator
only first two transitions are shown

The goal of using the look up table is to eliminate the need to create a sin wave generator and facilitate the programming. The flow chart for the program when controlling only a single phase is as follows:

Start >check for wanted frequency > access look up table for high/low times>output high/low times (half a period to top MOSFET)> output high/low times (half a period to bottom MOSFET)>restart

The reason for only choosing to run half of the period is that it will reduce the size of the look up table as well as guarantee that there will be no time when both switches are on as the result of the program needing to run through the complete string of numbers before it can restart and run them for the next MOSFET. In other words, this guarantees dead time, as the program must restart the look up table process for every half period.

D. Design Version 1 Evaluation

The Biggest error found with this design is the necessity for the micro-controller to operate with interrupts at a speed of 10s of us. This is not possible as the ATMEGA chip has trouble operating at lower resolutions than 10s of ms.

The next error is the MOSFET operation. It is difficult to turn on the MOSFET and keep it on even though the selected MOSFETs (IRFP360) have a threshold voltage of 4V (the voltage that must be applied at the Gate terminal for the switch to conduct). This turn on problem is due to the floating effect cause by not referencing the H-Bridge to ground. Because of the configuration, the switches also tend to "stick" to a position. When they are managed to be turned on, they cannot be turned off.

NOTE: Due to these errors being present at lower power scenarios, the configuration was not implemented at high power levels.

III. HARDWARE DESIGN OVERVIEW (VERSION 2)

A. DC-AC Converter Update

The original converter was unable to function due to time limitations of the micro-controller as well as problems with the "floating MOSFETs. The restructured program eliminates the look up table instead only uses a single look up table for all frequencies instead of one table for each frequency. More look up tables lead to an increased amount in storage space on the micro-controller IC. In this case the reduction frees up space on the microcontroller.

The code used for the construction of the sin waves is a modified version of the code written by Martin Nawrath under the name *Arduino DDS Sin wave Generator*. Originally, the code produces a sin wave PWM signal based on the reference voltage seen at the wiper of a potentiometer (a voltage divider is created by the potentiometer). This reference voltage then determines how fast the program reads through the look up table to produce a series of highs and lows.

The modified version of the code takes out unnecessary bits of code, sets up additional timers, applies delays, and outputs to multiple pins. In order to output a signal at different pins it is necessary to use different times. These timers control the set up for the output. Two additional pins are needed for the additional phases. The delay is not hard set; instead it runs off the initial look up table. Delay time is changed depending on how fast the look up table is read through. This is achieved by starting the look up sequence at a different value in the array for each phase. Three phases can be created using a single look up table. Because there is 256 values in the look up table (0-255), a third (1/3) of that is 85 and two thirds (2/3) is 170. Meaning that when the first phase is reading initial value (0), the second phase is reading 85th value, and the third is at the 170th value.

The second Problem was keeping the transistors in the ON state. A solution to the problem is using a MOSFET driver. IC GATE DRIVER HV 3PHASE 44-PLCC (part number IRS2336JTRPBF) from International Rectifier was used. It uses a charge pump to drive the gate voltage higher than the rail voltage or low by decreasing the voltage through an open collector circuit. This logic comes from manipulating the new PWM output from the micro-controller discussed earlier in this section; which is fed into the driver. The internal circuitry to the driver takes the logic level input and uses that to control the output which goes to the MOSFTE gate.

Figure 3.1 shows the internal structure to the Gate Driver. It first takes in a logic level signal which is fed into a noise filtering circuit. Next it proceeds to a buffer where dead time (delay) is created; here the enable signal for the IC is also taken into account. After that, it proceeds to level shifters that ensure the voltages are within an operating window. Next, values are stored in a latch portion where its values are fed into the driver or the open collector circuit. The driver turns on the MOSFET by applying power to the output pin of the driver and powering the external circuitry (charge pump). The open collector acts as a drain which dumps voltage at the gate to ground quickly, discharging both the charge pump and the parasitic capacitance of the switch itself.
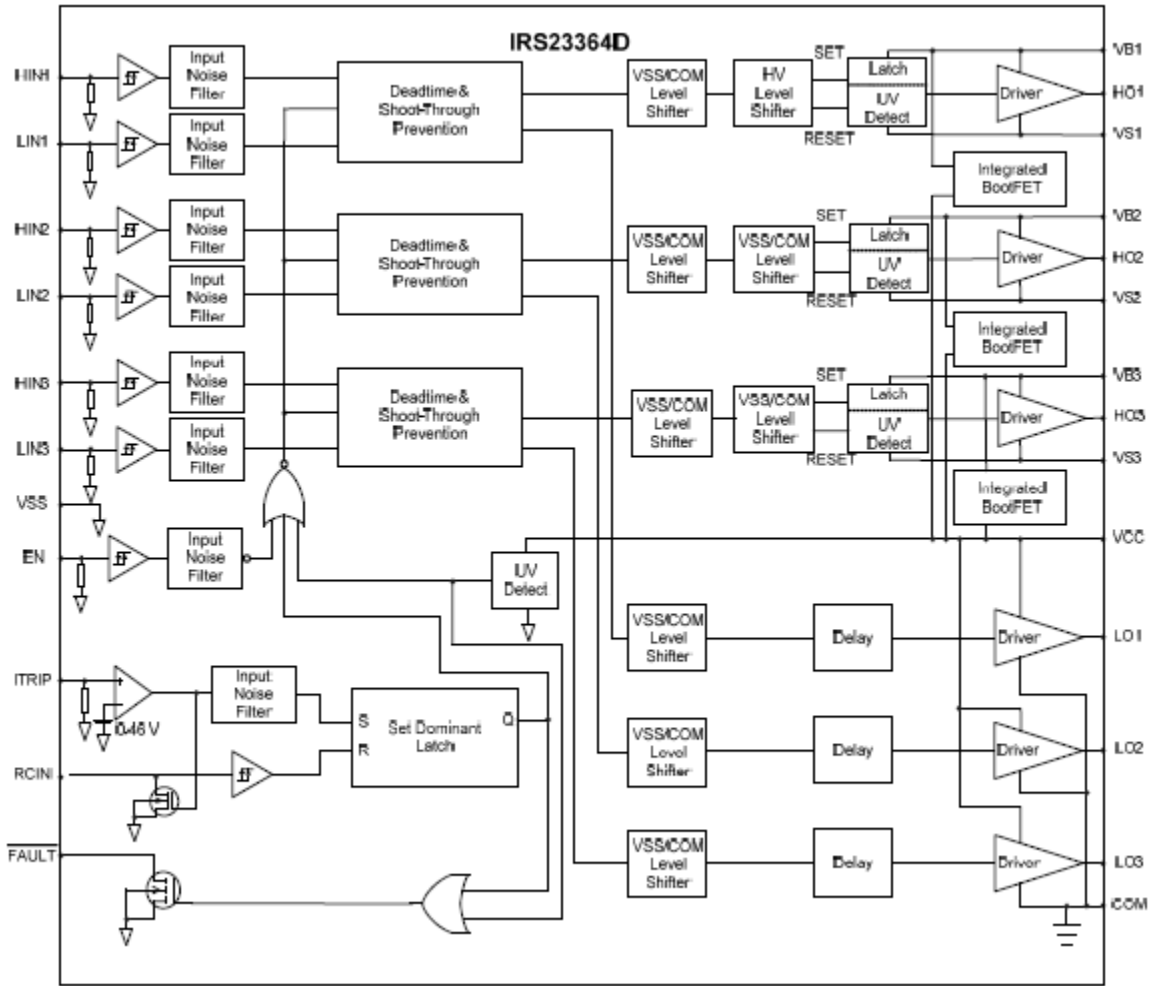
Figure 3.1 Internal Gate Driver Circuit:
The internal structure to the IRS2336JTRPBF
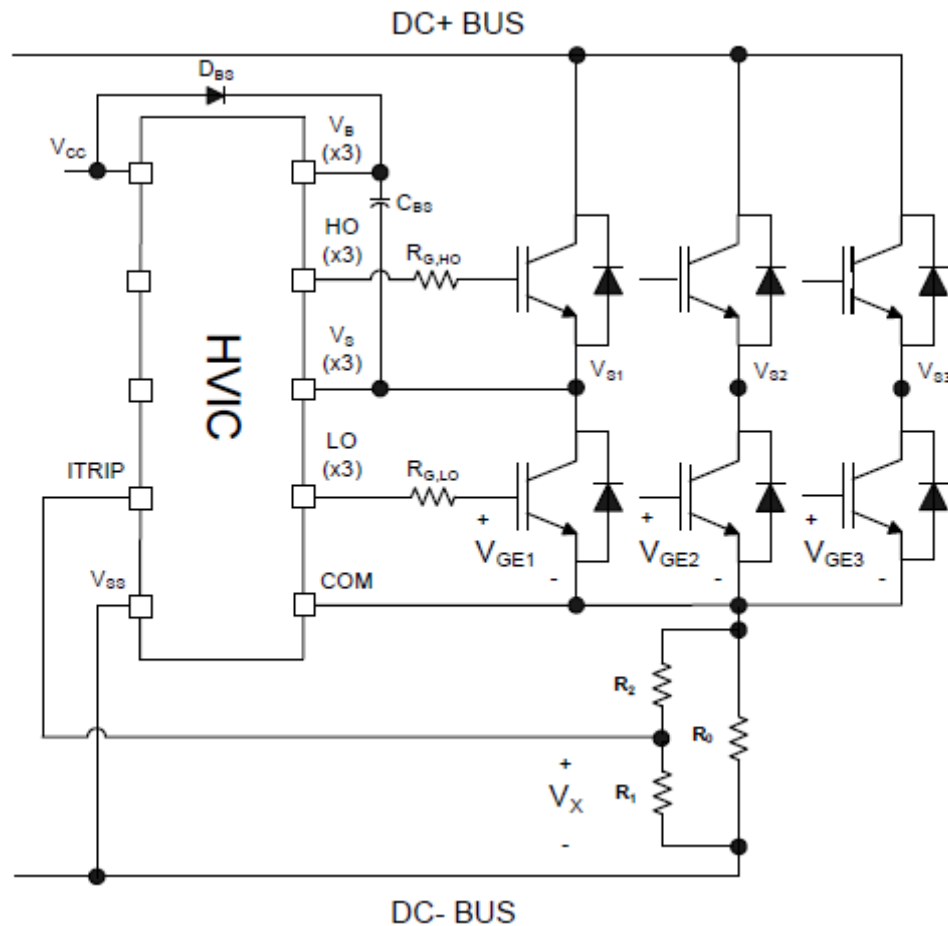
Figure 3.2 Gate Driver IC Typical Application: Shows a basic implementation of the International Rectifier Gate Driver with external BJT switches

Figure 3.2 only shows a single phase, but the charge pump can be seen in the figure as the capacitor connecting the V_S and V_B pins. The resistors are inserted in order to draw sufficient current for the switches to conduct. The additional circuit at the bottom is for over current protection. For simplicity, we will bypass this function.

Each of the 3 signals output from the Arduino is run through a low pass filter. This creates typical 3 phase at the logic voltages; since the output is from a micro-controller, it cannot go negative so the signal from the Arduino has a DC offset of ½
$V_{peak-peak}$. The simple addition of a DC source with the positive (+) terminal referenced to ground and the negative (-) terminal connected to the output of the filter will bring down the sin wave so that it crosses the x-axis at one half a period.

```
pin11  o---[ 270 Ohm ]---•---[ 4.7mH ]---•---[ 4.7mH ]---•---[ 270 Ohm ]---•-------o  out
                         |               |               |               |   2
                       47nF           100nF            47nF             [ ]  /
                         |               |               |             [ ] |  3
                         |               |               |              1  |  100 K
  gnd   o---------------•---------------•---------------•---------------•---•------o
                                                                                    gnd
```
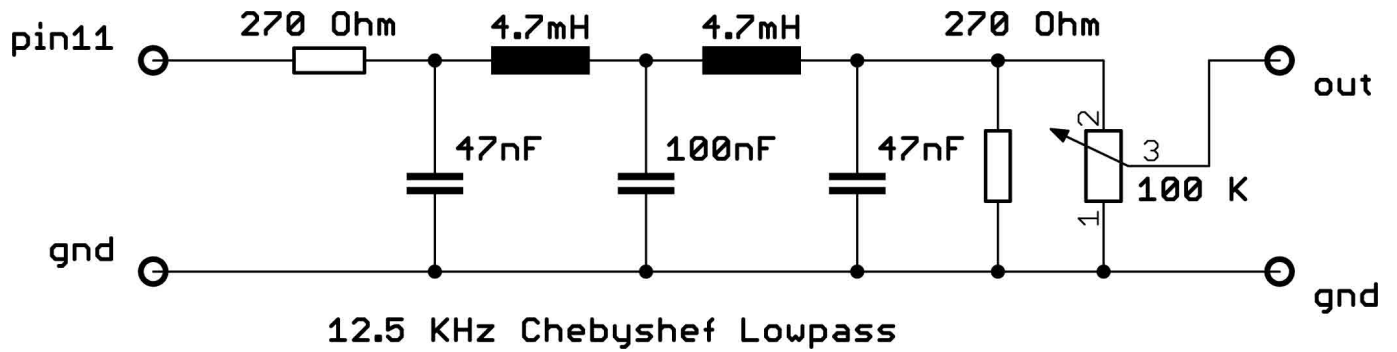
12.5 KHz Chebyshef Lowpass

Figure 3.3 Chebyehef Low-Pass Filter: the low pas filter used to
go from PWM to a Sin wave it cuts off at 12.5kHz as well
as filter out the 32kHz signal used for sampling
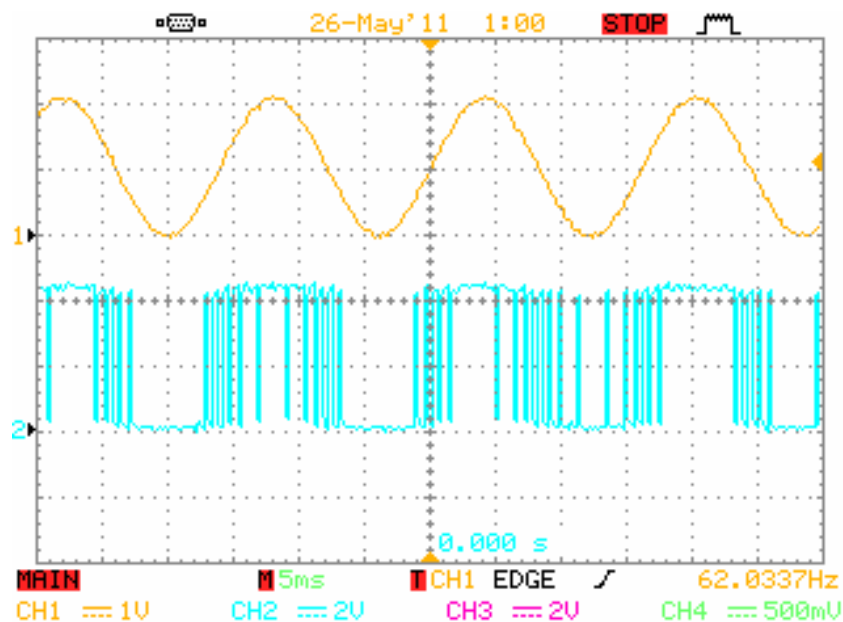
Figure 3.4 Low-Pass Filter Input/Output:
(top) single phase output at the low pass filter
(bottom) PWM before filtering

   Next, each of the sin waves sees two half wave rectifying circuits as in Figure 3.2. The purpose of using two instead of a full wave is to separate the portion above the x-axis from the portion below the x-axis. Now we have six half wave rectifier outputs, two per phase. Keep in mind that apart from the separation of the of the two portions of the sin wave, the bottom tips of the signal is "chopped off" due to the turn on voltage of the diodes in the rectifier. This will work in our advantage because it gives a small amount of dead time.

Figure 3.5 Half Wave Rectifier Circuit: Circuit applied to a single phase
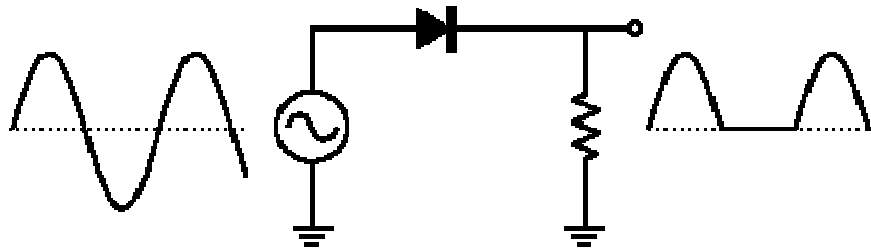with a pull down resistor at output (only shows the + portion)
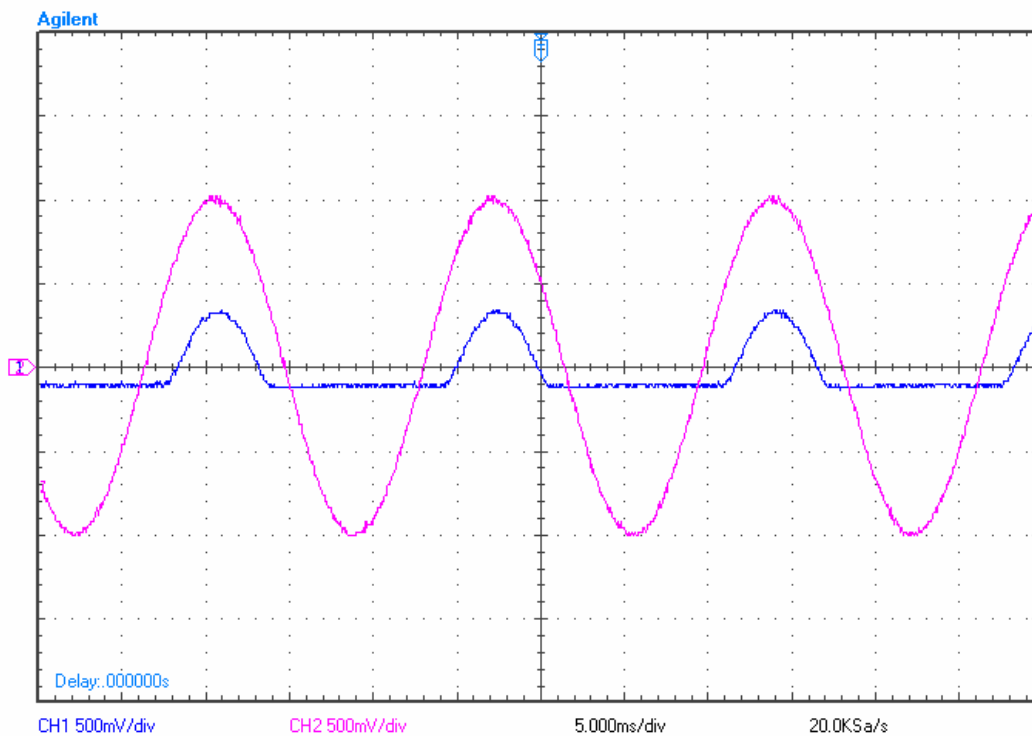
Figure 3.6 Half Wave Rectifier Input/Output:
Signals after the half wave rectifier circuit

   To create the logic needed to control the gate driver we use a comparator. In one signal we feed in a triangle wave constructed with the use of a 555 timer as seen in Figure 2.5. The design of this circuit as well as the triangle wave generator can be seen in the appendix. In the other input the half-wave signals are fed. At this point we are still working with six signals and circuits, but it is necessary to use only one triangle wave for as a reference for everything. In addition to only one triangle wave, it should not go below 0V, as this will cause switching signals for the MOSFETs when not desired.

B. *Design Version 2 Evaluation*

The second design uses a half wave rectifier to split the control signal in to high and low so they can be routed to its corresponding MOSFET. The main issue when using this approach is the voltage consumption of the circuit. As seen in figure 3.5, the difference between the sin wave and the rectified wave is around 0.6V which is typical voltage consumption of the diode. The problem with this is that most of the signal is consumed, and the dead time is close to 60% (30% coming from 2 half-wave circuits corresponding to 1 phase). The processing of the signal can also be reduced in the sense that there is no need for the triangle wave when the PWM has already been generated. This will free up room on the circuit board, as well as being more cost effective.

IV. Hardware design overview (version 3)

A. DC-AC Converter Update

The third version of motor drive again focuses on the revision of the DC-AC converter. It still uses the Arduino with the same software. The difference is in the hardware that manipulates the signal. Directly after the Arduino, we still have the low-pass filter, but the sin wave generated output from this filter is now fed into two comparators (LM339 was used). The first comparator sees the sin wave at the positive (+) input, while the second comparator sees it at the negative (-) input. The rails for the comparator are +5V for the positive, and Ground for the negative with the reason being that the signal will only swing between these values at the input to the comparators as well as on the output. The necessity for the two comparators is to split the logic and create two enable signals: one for the high side and another for the low side.
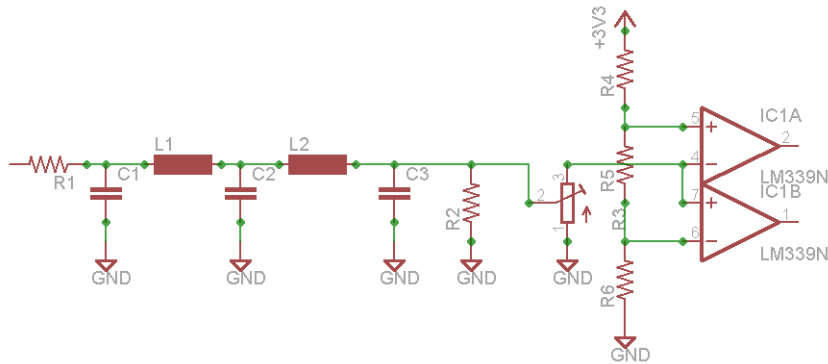

Figure 4.1 Low-Pass Filter with Comparators:
Connected to LM339 comparators

NOTE: a voltage divider sets the comparing voltage
The 3.3V is taken directly from the Arduino

$$V1 = 3.3\left(1 - \frac{R4}{R4 + R5 + R6}\right)$$

$$V_2 = 3.3\left(1 - \frac{R_4 + R_5}{R4 + R_5 + R_6}\right)$$

Equation 4.1 nodal voltages

The additional terminal for the comparators will see a reference DC voltage. It will decide the dead time in-between the transitions for the signal routing. As previously mentioned the dead time is needed to create the time gape needed to ensure no cross over timer when both switches will be conducting at once.

The output for the comparators will be a square wave which changes in frequency with response to the PWM output from the Arduino. They will match in frequency. The duty cycle is controlled by the dc reference voltage at the additional input to the comparators. Two comparators are used to create two separate signals with respect to both the positive and negative portions of the filtered sin wave (if the DC offset is removed).
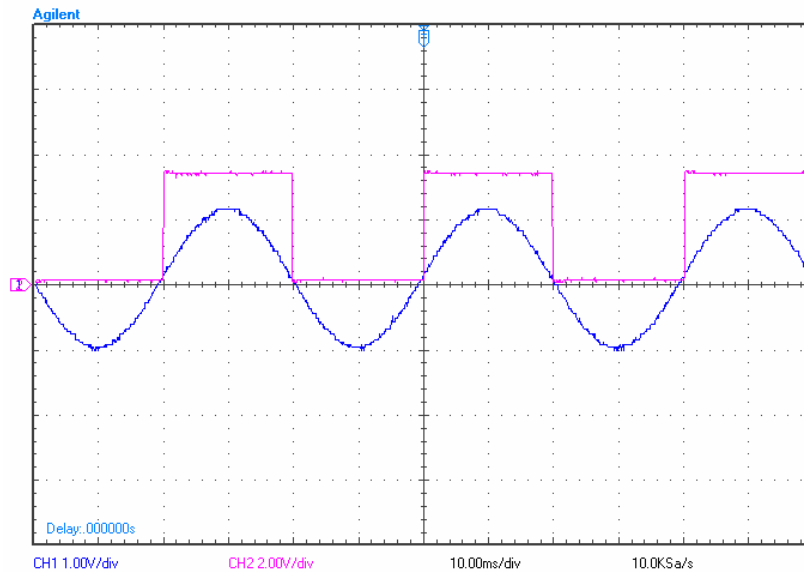
Figure 4.2 Enable Circuit Input/Output:
CH1 is the filtered sin wave, CH2 is the enable signal

The square wave is then fed into an input to an AND logic gate. Available on hand were NAND gates, a buffering NAND gate was added in an inverting configuration to replicate the behavior of an AND gate. The other input is the original unfiltered PWM output from the Arduino board. This allows the square wave to act as an enable for the signal to pass only when desired. When both the PWM and the enable are high, the logic output will be high. When one is low, the output is low. The resulting signal is a replica of the enable signal, but instead of being consistently high for a given duty cycle, the PWM will be visible during this time. There will be three of these filtering circuits and six control signals will be produced. Two signals from a single filter correspond to a single phase. These signals are then fed into a gate driver.
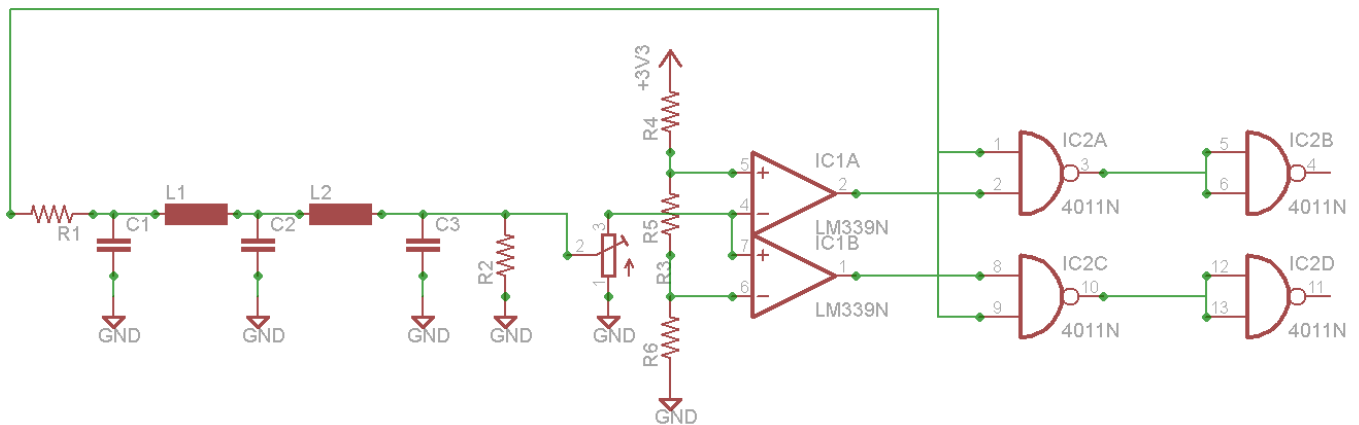

Figure 4.3 Control Circuit Configuration (High Logic):
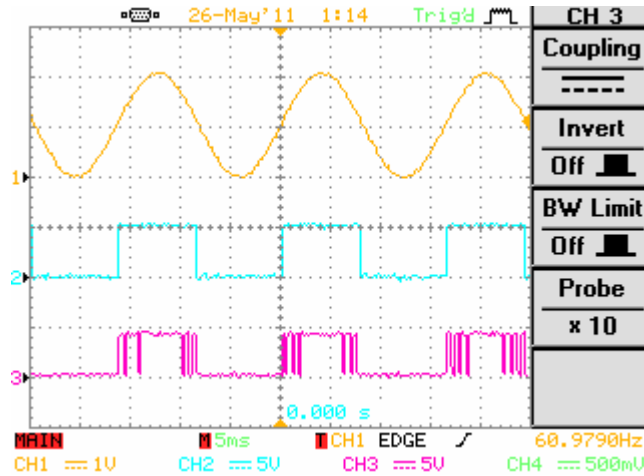shows the configuration of the logic gates after the comparators

Figure 4.4 Important Control Circuit Waveforms:
The progression of the signals through the control circuit
CH1 is the filtered signal
CH2 is the enable signal after the comparators
CH3 is the logic fed into the gate driver after the NAND gates

The purpose of using a gate driver is the application of a charge pump circuit to the gate of the MOSFET. By doing so, it allows the gate voltage to be lifted and maintain the ON position in the switch. It is an application of the "bootstrap effect". The circuit for the gate driver was designed according to the published article: *Design Tips: Using Monolithic High Voltage Gate Drivers, By: A. Merello, A. Rugginenti, and M. Grasso*.

B. Driver Component Sizing

Bootstrap Capacitor:

$$\Delta V_{BS} \leq V_{CC} - V_F - V_{GE\,min} - V_{CEon}$$

V_{CC} – the IC voltage supply
V_{CEon} – the emitter collector voltage (in this case Drain Source)
V_F – bootstrap diode forward voltage
V_{GEmin} – the min gate emitter voltage (Gate Drain)

Equation 4.2 finds the minimum voltage drop that guarantees the MOSFT is on

$$C_{BOOT\,min} = \frac{Q_{TOT}}{\Delta V_{BS}}$$

$$Q_{TOT} = Q_G + Q_{LS} + (I_{LK\_GE} + I_{QBS} + I_{LK} + I_{LK\_DIODE} + I_{LK\_CAP} + I_{DS-}) \cdot T_{HON}$$

Q_G – Gate charge needed to turn ON
Q_{LS} – Charge required by the internal level shifters
I_{LK_GE} – Gate Source leakage current
I_{QBS} – Quiescent Current
I_{LK} – Floating section leakage current
I_{LK_DIODE} – Bootstrap diode leakage current
I_{LK_CAP} – bootstrap capacitor leakage current
I_{DS} – Desat diode bias (when ON)
T_{HON} – High side ON Time

Equation 4.3 uses equation 4.2 to calculate the needed capacitor voltage

NOTE: values take from data sheet; some may or may not apply depending on type of switch used

Bootstrap Diode:
   The diode must have a BV > DC+ and a fast recovery time (<100ns) to minimize the amount of charge fed back from the bootstrap capacitor to V_{CC} supply.

For additional design strategies refer to article:
Design Tips: Using Monolithic High Voltage Gate Drivers By: A. Merello, A. Rugginenti, and M. Grasso.
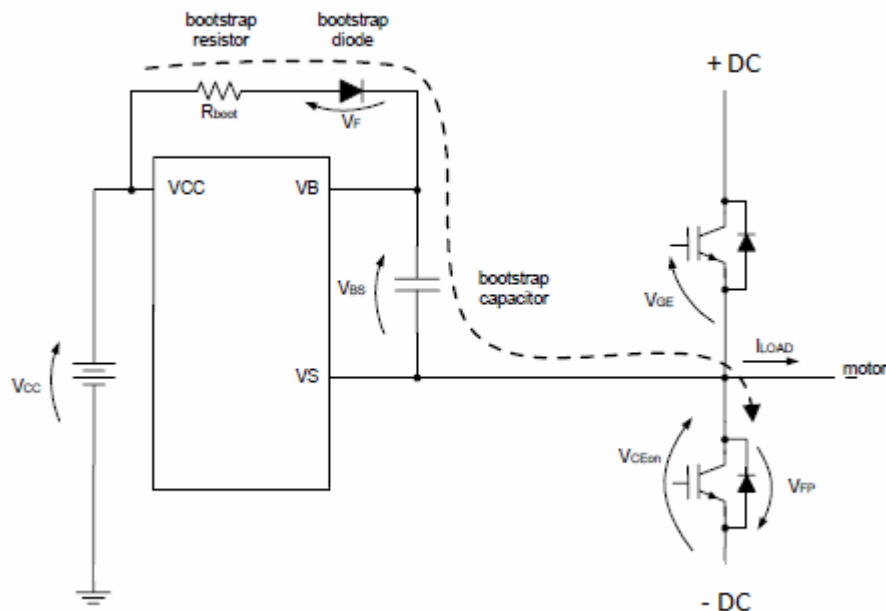

Figure 4.5 Charge Pump Circuit:
The component connections made for a single phase on the driver

V. TESTING

A. *Small Scale*

The testing was broken down into two categories: small and large scale power circuits. The controlling circuit was kept the same. The small scale testing was essential because the power handling capabilities of the circuit at its initial prototype levels was not possible. Due to the large power demands, the large IRFP360PBF MOSFETs from International Rectifier were traded in for smaller switches IRF510 from International Rectifier as well. The DC rails were reduced to 10V and obtained from a DC power supply instead of a rectified signal. Testing for the drive was done for a range of 40-80 Hz. Key nodes were probed and their signals analyzed. The captures included in this report are the result of the small scale testing.

While a complete large scale test was not completed due to a lack of components (PCB), the small scale test was a success. All the signals behaved as predicted. Additional testing is required, but must be done using a combination of PCB to ensure proper connections as well as proper power capabilities.

VI. PCB DESIGN

The control circuit was separated from the power circuit as much as possible in an attempt to isolate any induced voltages from the large currents seen in the power circuit. Included in the appendix are the gerber files for both the control circuit and the power circuit. On the power circuit, the capacitors and the rectifier will be mounted separately and jumpers used to link the parts. This was done to isolate any problems that might be encounter as well as to save board space.

VII. CONTINUED WORK

This purpose of this area is to describe any additional steps that can be taken to improve the design if more time was available.

A. Additional Features

Due to the applications for the motor drive, a GUI would be useful. In the original design, a user interface was proposed; but due to the space restrictions on the Arduino in terms of timers and outputs, interfacing an LCD screen and buttons was not possible.

Additional expansion of for this project would include the integration of pushbuttons that control a digital potentiometer. Because the Arduino only depends on the reference voltage at the analog to digital converter (ADC), the precise control of a digital potentiometer would be ideal. It would give more control over the frequency selection, as well as keep the output stable (no wonder in the frequency output due to the elimination of an analog wiper on the potentiometer.

The additional power supply rails for the logic level circuits are provided by a battery pack. Because the unit is meant to stand alone, additional design work would be necessary to eliminate the need for a battery pack. The only problem is the use of 3-phase as an input. Due to its high potential, large and costly transformers would be required to step down the voltage to a safe level. A possible fix to the problem can be the addition of a small solar array to keep a battery bank charged for use by the logic circuits.

Last suggestion would be the complete integration of all the separate parts into one PCB. I would suggest that a Faraday Cage be added around the controlling circuit to shield it from any induced voltages from the power circuit

VIII. CONCLUSION

The basic idea for this senior project was taken from the functional explanations of an H-Bridge as described in *Advanced Power Electronics: EE411* by Dr. Taufik and Dr. Dolan. The functionality and principals of a PWM H-Bridge was used as a base for idea development. Unique circuitry was designed and developed as needed for the motor drive. This report documents the development in chronological order as problems with the original design were identified.

The main difference between the H-Bridges discussed in the text book and the one described in this paper is the ability for selectable frequency outputs. A micro-controller provides the initial control signal and analog circuitry processes the original signal into a usable control signal. The power circuit for the drive consisted of a bridge rectifier, capacitor bank, and H-bridge. These are the most stable designs. The control circuit can use the most improvement.

IX. REFERENCES

Taufik, and Dolan, *Advanced Power Electronics*. 5th Revision. San Luis Obispo:
Cal Poly State   University, San Luis Obispo, 2011. Print.

Taufik, and Dolan, *Introduction to Power Electronics* . 8th Revision. San Luis Obispo:
Cal Poly State University, San Luis Obispo, 2010. Print.

Franco, S. *Design with Operational Amplifiers and Analog Integrated Circuits*. 3rd . McGraw-Hill, 2002. 413-415. Print.

Merello, A., A. Rugginent, and M. Grasso. "Using Monolithic High Voltage Gate Drivers."*International Rectifier Data Sheet*. International Rectifier, Web. <http://www.irf.com/technical-info/designtp/dt04-4.pdf>.

Prodanov, Vladimir. "Relaxation Oscillators with 50% Duty Cycle, Triangle Wave Generators." EE409 Lecture Notes. Cal Poly State University, San Luis Obispo. San Luis Obispo. 11/2/2010. Lecture.

Prodanov, Vladimir. "Pulse-Width Modulation." EE409 Lecture Notes. Cal Poly State University, San Luis Obispo. San Luis Obispo. 11/4/2010. Lecture.

Nawrath, Martin. "Arduino DDS Sine Wave Generator."*Lab III: Laboratory for Experimental Computer Science at the Academy of Media Arts Cologne*. Academy of Media Arts Cologne, n.d. Web. <http://interface.khm.de/index.php/lab/experiments/arduino-dds-sinewave-generator/>.

Data Sheets:

Three Phase Bridge (Power Modules), 25/35 A, Vishay, http://www.vishay.com/docs/93565/93565.pdf

Power MOSFET, Vishay Siliconix, http://www.vishay.com/docs/90292/sihfp360.pdf

Aluminum Capacitors, Power High Ripple Current, Screw Terminals, Vishay BC components, http://www.vishay.com/docs/28371/101102ph.pdf

Quad Comparator, Fairchild Semiconductor, http://www.fairchildsemi.com/ds/LM/LM239A.pdf

High Voltage 3 Phase Gate Driver IC, International Rectifier, http://www.irf.com/product-info/datasheets/data/irs2336.pdf

8-bit Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash, Atmel, http://www.atmel.com/dyn/resources/prod_documents/doc8161.pdf
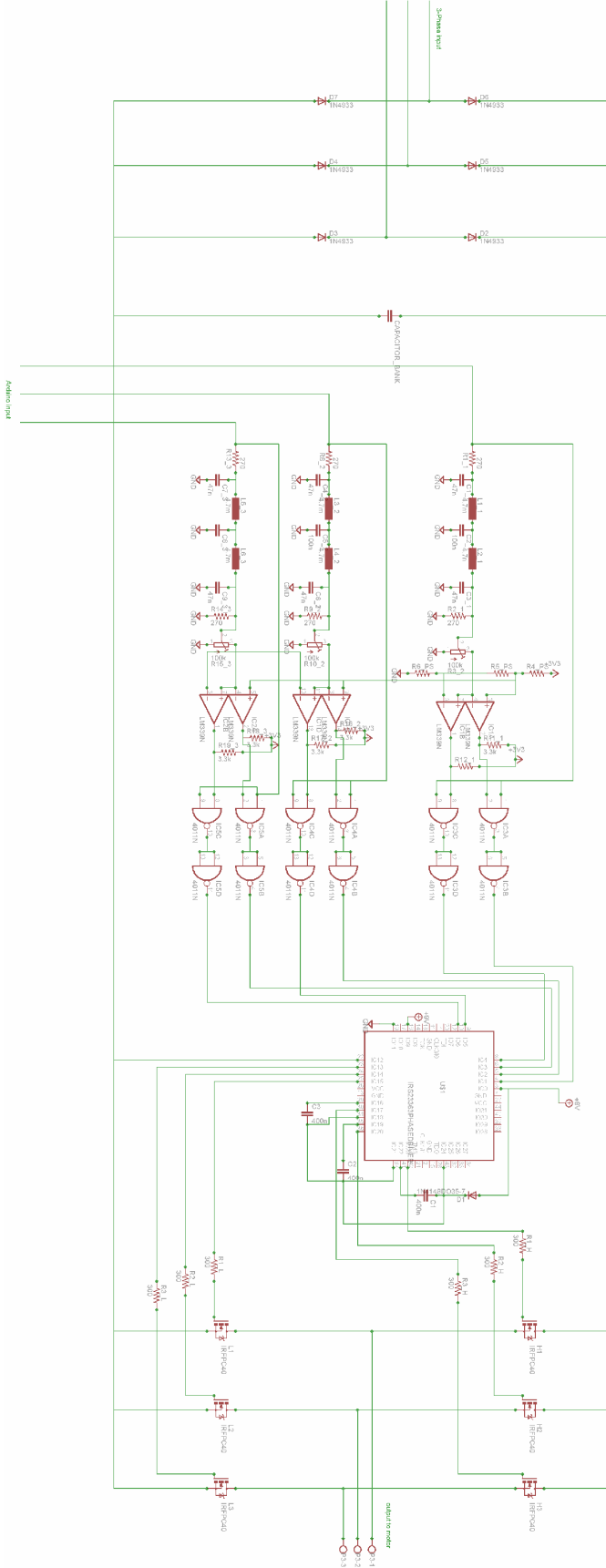
X. Appendies

A. Schematic

Figure 11.6
The complete design at component
level

SPECIAL NOTES:
Component sizing is included
The "Gate Driver IC" is the
IRS2336JTRPBF
  By International Rectifier
The Input sources not shown are the
  connections to a 3 phase input source
  as well as the Arduino

B. Parts List and Cost

| Part	Quantity	Cost Per Unit
3-Phase Bridge Rectifier (p/n: 36MT80):	1	16.21
470uF Aluminum Capacitor 400V/3A (p/n: MAL210216471E3)	2	21.85
N-Channel Power MOSFET (p/n: IRFP360PBF)	6	6.80
3-Phase Gate Driver (p/n: IRS2336JTRPBF)	1	7.82
General Purpose Comparators (p/n:LM339)	2	0.75
100k Pot Multiple Turn	3	1.00
Single Turn 10k Pot	1	2.99
Arduino Uno (ATMEGA 328p)	1	29.95
4.7mH inductor (p/n: TSL0808RA-472JR13-PF)	6	1.28
IC Gate NAND (p/n: 74HC00N)	3	0.48
Assorted Resistors	-	-
Assorted Capacitors	-	-
Power Circuit PCB	1	79.90
Control Circuit PCB	1	79.90
Miscellaneous connectors	-	-
Total Cost Per Unit	315.07 +miscellaneous parts and tax	

C. PCB Layout
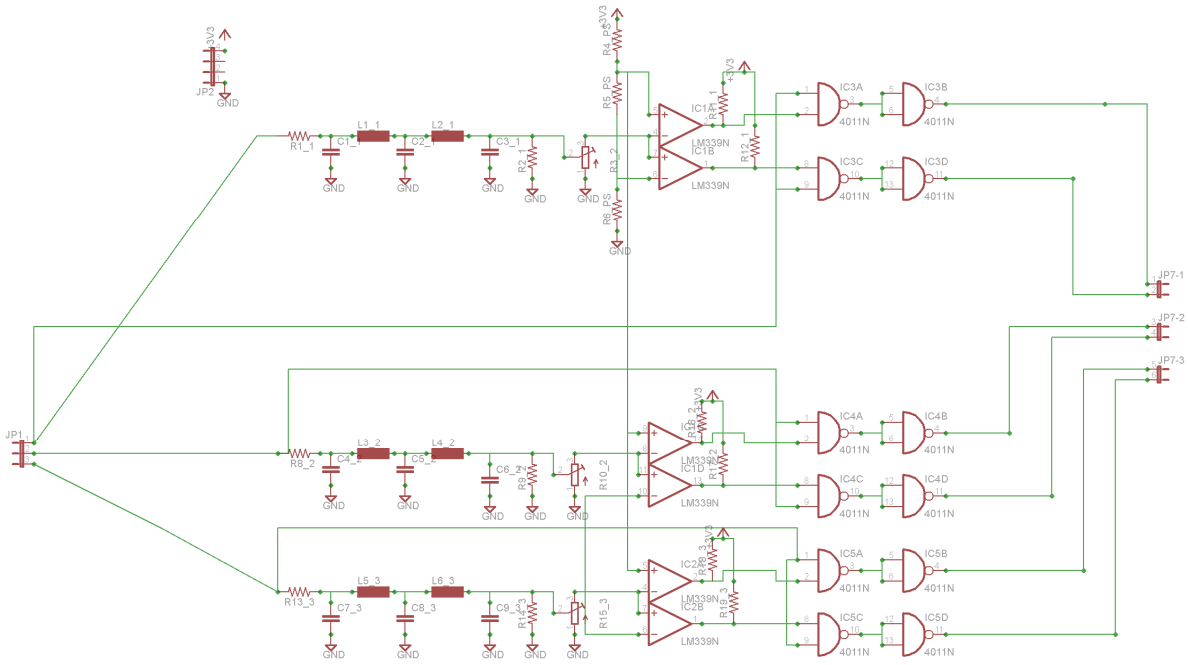
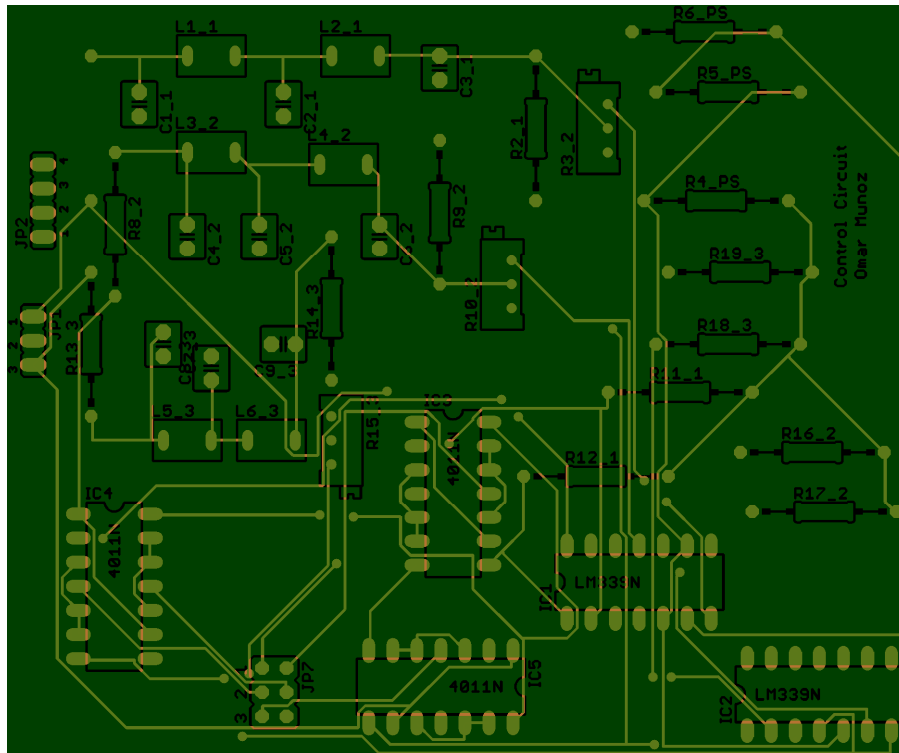Figure 11.1 The control circuit at component level

Figure 11.2 Control circuit component layout on PCB

Figure 11.3 top level of control circuit PCB layout
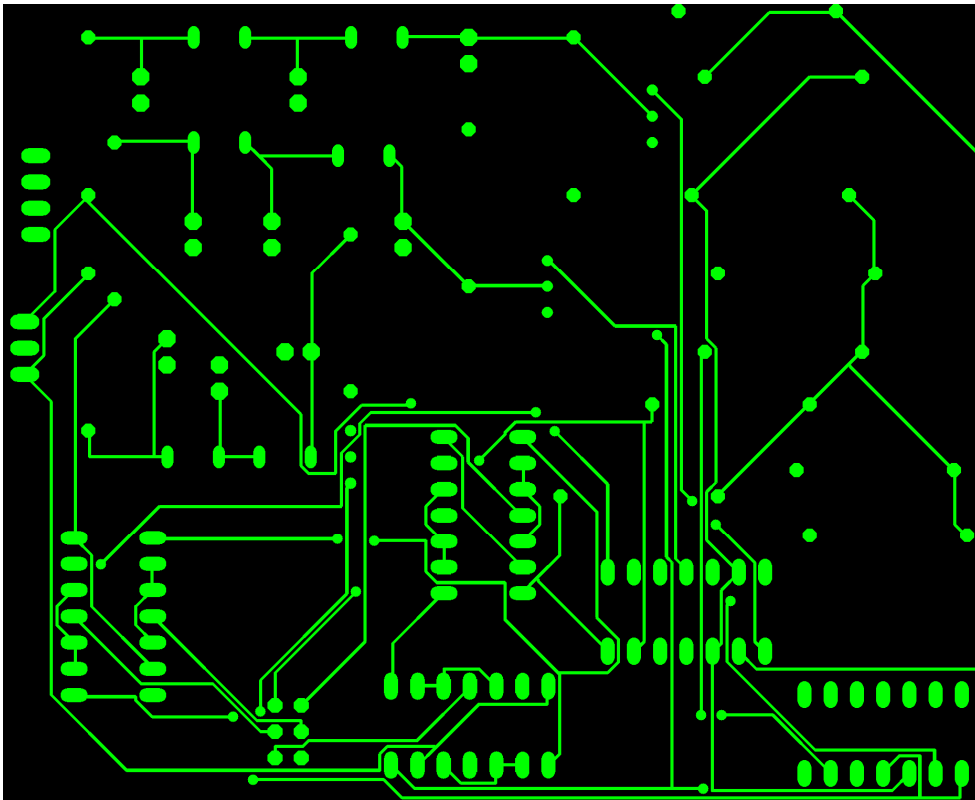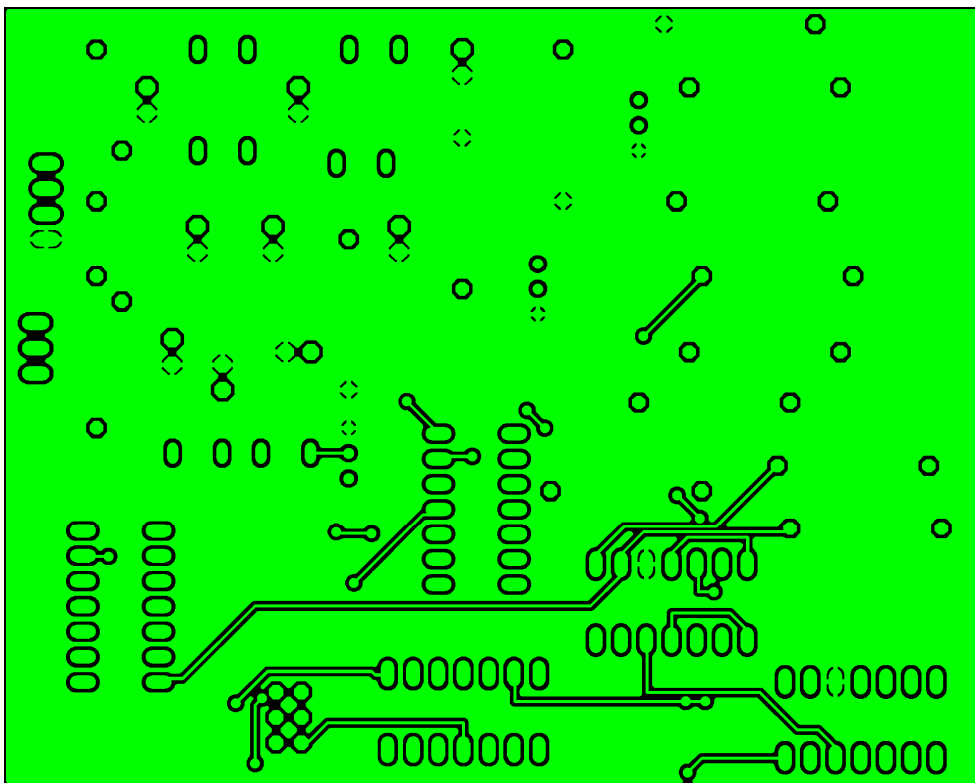

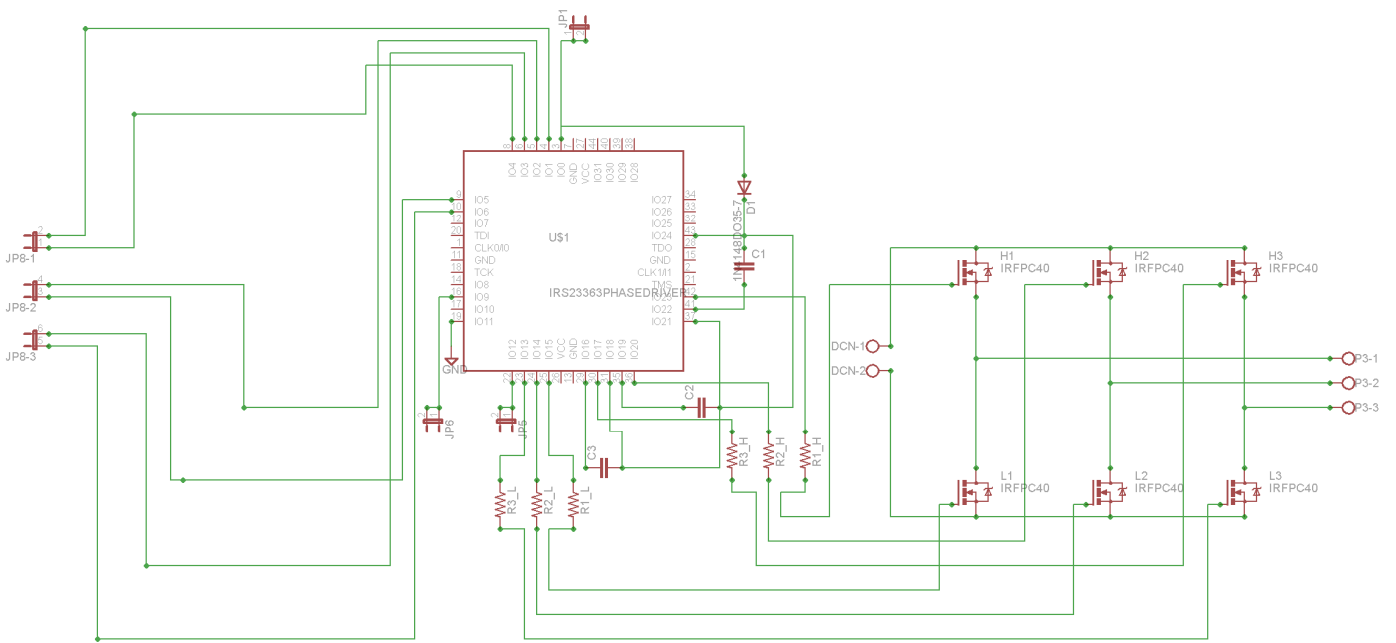Figure 11.4 bottom level of control circuit PCB layout

Figure 11.5 Diver Circuit (Power Circuit with Gate Driver) at component level
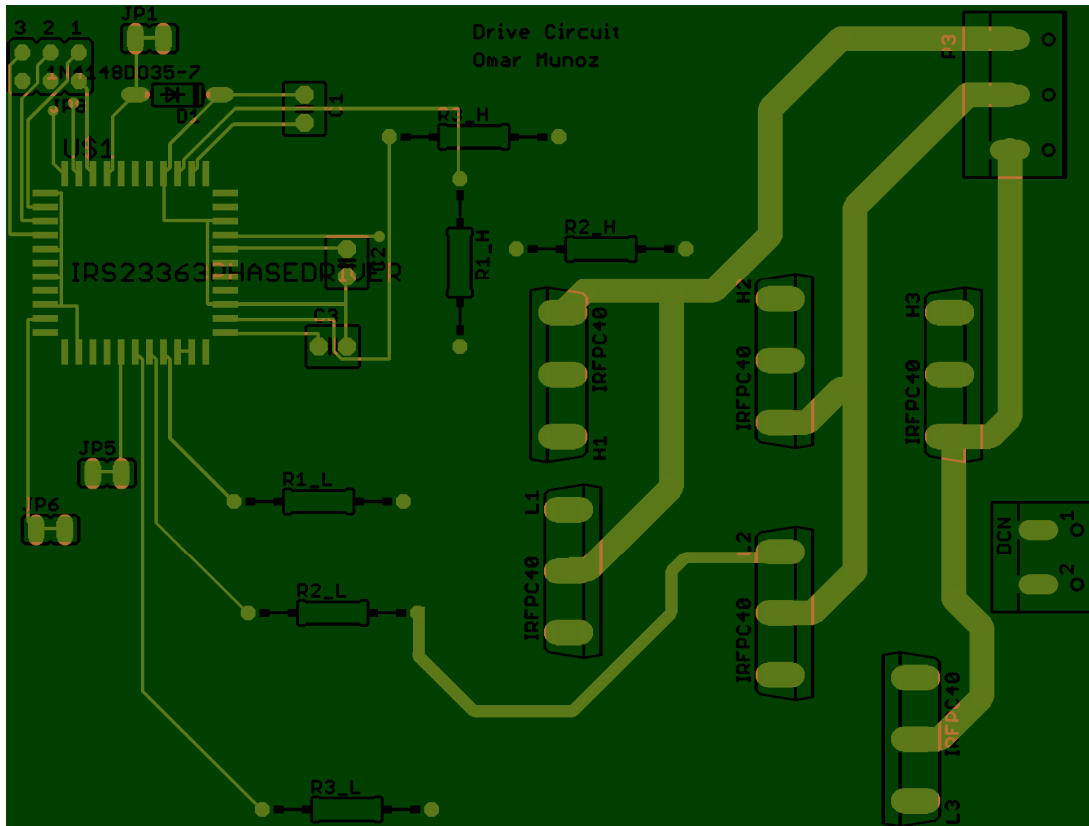

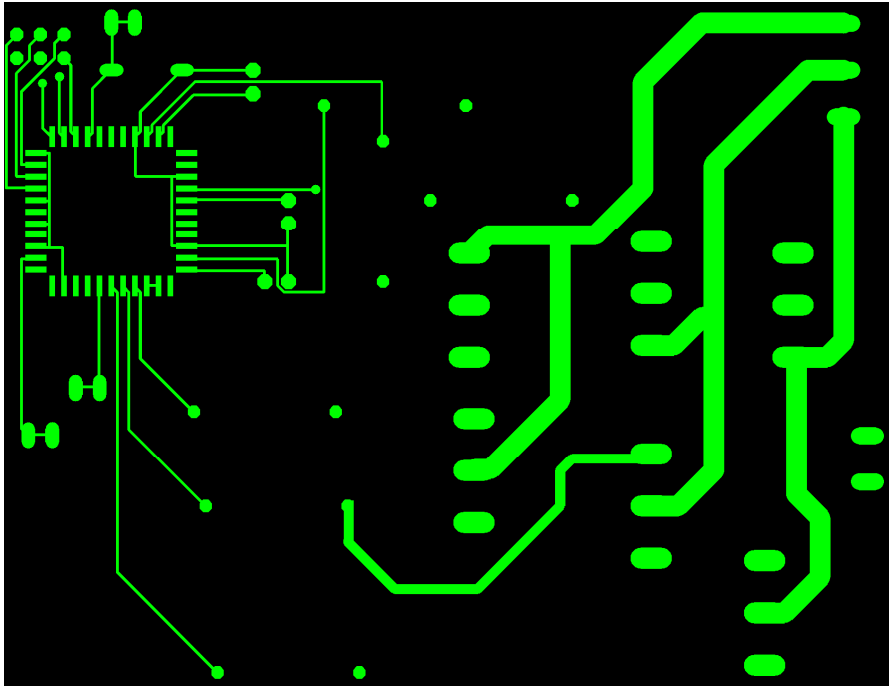Figure 11.6 Driver Circuit component layout on PCB

Figure 11.5 top level of Diver Circuit (Power Circuit with Gate Driver) circuit PCB layout
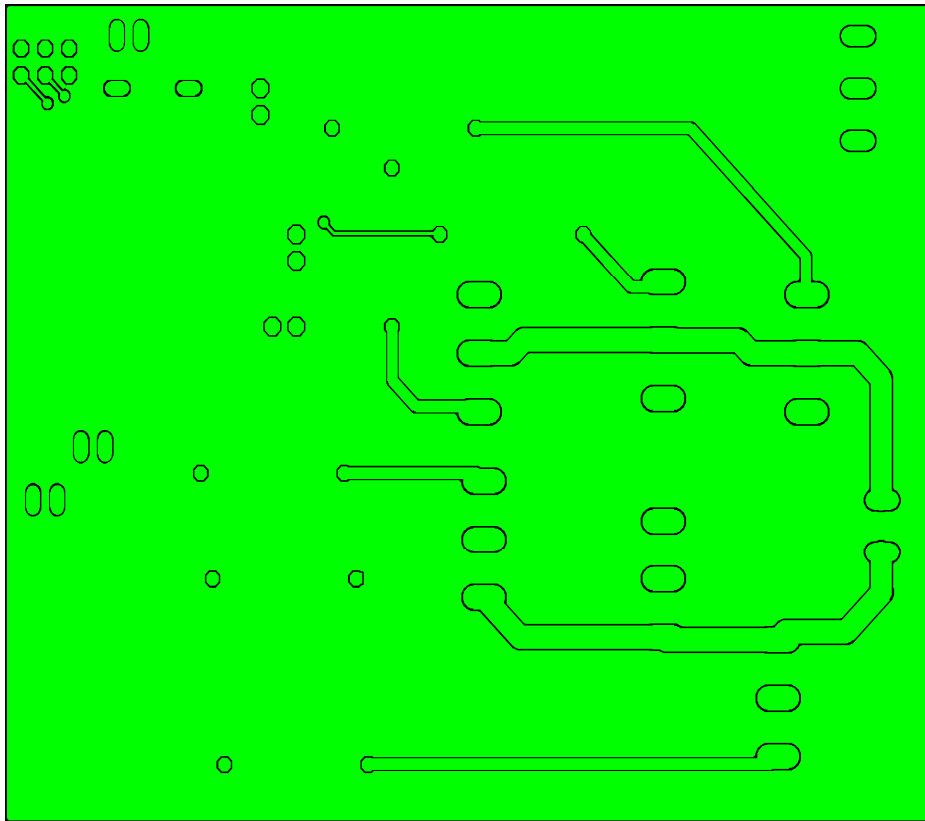

Figure 11.5 bottom level of Diver Circuit (Power Circuit with Gate Driver) circuit PCB layout

D. Arduino C Code

```
/****************************************************************
 * DDS-sinewave - 3phase
 *
 ****************************************************************/

#include "avr/pgmspace.h"
#include "avr/io.h"

// table of 256 sine values / one sine period / stored in flash memory
PROGMEM  prog_uchar sine256[] =
{

127,130,133,136,139,143,146,149,152,155,158,161,164,167,170,173,176,178,181,184,187,190,192,195,198,200,203,205,208,2
10,212,215,217,219,221,223,225,227,229,231,233,234,236,238,239,240,242,243,244,245,247,248,249,249,250,251,252,252,25
3,253,253,254,254,254,254,254,254,254,253,253,253,252,252,251,250,249,249,248,247,245,244,243,242,240,239,238,236,234
,233,231,229,227,225,223,221,219,217,215,212,210,208,205,203,200,198,195,192,190,187,184,181,178,176,173,170,167,164,
161,158,155,152,149,146,143,139,136,133,130,127,124,121,118,115,111,108,105,102,99,96,93,90,87,84,81,78,76,73,70,67,64,
62,59,56,54,51,49,46,44,42,39,37,35,33,31,29,27,25,23,21,20,18,16,15,14,12,11,10,9,7,6,5,5,4,3,2,2,1,1,1,0,0,0,0,0,0,0,1,1,1,2,
2,3,4,5,5,6,7,9,10,11,12,14,15,16,18,20,21,23,25,27,29,31,33,35,37,39,42,44,46,49,51,54,56,59,62,64,67,70,73,76,78,81,84,87,
90,93,96,99,102,105,108,111,115,118,121,124

};

#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))

#define PWM_OUT_1   11       // PWM output on pin 11
#define PWM_OUT_2   10       // PWM output on pin 10
#define PWM_OUT_3   9     // PWM output on pin 9
#define LED_PIN       13       // LED status on pin 13
#define TEST_PIN 7     // Scope trigger on pin 7
#define POTEN_IN0      // Potentiometer on pin 0

#define OFFSET_1 85       // Offset for second-phase
#define OFFSET_2 170       // Offset for third-phase

double dfreq;
const double refclk = 31376.6;        // measured
const uint64_t twoTo32 = pow(2, 32);   // compute value at startup and use as constant

// variables used inside interrupt service declared as voilatile
volatile uint8_t icnt;          // var inside interrupt
volatile uint8_t icnt1;          // var inside interrupt
volatile uint8_t c4ms;          // counter incremented every 4ms
volatile uint32_t phase_accum;       // pahse accumulator
volatile uint32_t tword_m;      // dds tuning word m

//****************************************************************
void setup()
{
  pinMode(LED_PIN, OUTPUT);       // sets the digital pin as output
  Serial.begin(115200);          // connect to the serial port
  Serial.println("DDS Test");
```

```
  pinMode(TEST_PIN, OUTPUT);                     // sets the digital pin as output
  pinMode(PWM_OUT_1, OUTPUT);                    // PWM output / frequency output
  pinMode(PWM_OUT_2, OUTPUT);                    // PWM output / frequency output
  pinMode(PWM_OUT_3, OUTPUT);                    // PWM output / frequency output

  // Setup the timers
  setup_timer1();
  setup_timer2();

  // disable interrupts to avoid timing distortion
  cbi (TIMSK0, TOIE0);                           // disable Timer0 !!! delay() is now not available
  sbi (TIMSK2, TOIE2);                           // enable Timer2 Interrupt

  dfreq = 1000.0;                                // initial output frequency = 1000.0 Hz
  tword_m = twoTo32 * dfreq / refclk;            // calulate DDS new tuning word
}


//****************************************************************
void loop()
{
  if (c4ms > 250)                                // timer / wait for a full second
  {
    c4ms = 0;
    dfreq = analogRead(POTEN_IN);                // read Poti on analog pin 0 to adjust output frequency from 0..1023 Hz

    cbi (TIMSK2, TOIE2);                         // disble Timer2 Interrupt
    tword_m = twoTo32 * dfreq / refclk;          // calulate DDS new tuning word
    sbi (TIMSK2, TOIE2);                         // enable Timer2 Interrupt

    Serial.print(dfreq);
    Serial.print("  ");
    Serial.println(tword_m);
  }
}


//****************************************************************
// timer1 setup
// set prscaler to 1, PWM mode to phase correct PWM,  16000000/512 = 31.25kHz clock
void setup_timer1(void)
{
  // Timer1 Clock Prescaler to : 1
  sbi (TCCR1B, CS10);
  cbi (TCCR1B, CS11);
  cbi (TCCR1B, CS12);

  // Timer0 PWM Mode set to Phase Correct PWM
  cbi (TCCR1A, COM1A0);                          // clear Compare Match
  sbi (TCCR1A, COM1A1);
  cbi (TCCR1A, COM1B0);                          // clear Compare Match
  sbi (TCCR1A, COM1B1);

  sbi (TCCR1A, WGM10);                           // Mode 1  / Phase Correct PWM
  cbi (TCCR1A, WGM11);
  cbi (TCCR1B, WGM12);
  cbi (TCCR1B, WGM13);
}
```

```c
//************************************************************
// timer2 setup
// set prscaler to 1, PWM mode to phase correct PWM,  16000000/512 = 31.25kHz clock
void setup_timer2(void)
{
 // Timer2 Clock Prescaler to : 1
 sbi (TCCR2B, CS20);
 cbi (TCCR2B, CS21);
 cbi (TCCR2B, CS22);

 // Timer2 PWM Mode set to Phase Correct PWM
 cbi (TCCR2A, COM2A0);                            // clear Compare Match
 sbi (TCCR2A, COM2A1);

 sbi (TCCR2A, WGM20);                             // Mode 1  / Phase Correct PWM
 cbi (TCCR2A, WGM21);
 cbi (TCCR2B, WGM22);
}


//************************************************************
// Timer2 Interrupt Service at 31.25kHz = 32us
// this is the timebase REFCLOCK for the DDS generator
// FOUT = (M (REFCLK)) / (2 exp 32)
// runtime : 8 microseconds ( inclusive push and pop)
ISR(TIMER2_OVF_vect)
{
 sbi(PORTD, TEST_PIN);                      // Test / set PORTD,TEST_PIN high to observe timing with a oscope

 phase_accum += tword_m;                    // soft DDS, phase accu with 32 bits
 icnt = phase_accum >> 24;                  // use upper 8 bits for phase accu as frequency information

 OCR2A = pgm_read_byte_near(sine256 + icnt);    // read value fron ROM sine table and send to PWM DAC

 OCR1A = pgm_read_byte_near(sine256 + (uint8_t)(icnt + OFFSET_1));

 OCR1B = pgm_read_byte_near(sine256 + (uint8_t)(icnt + OFFSET_2));

 if (icnt1++ == 125)                        // increment variable c4ms every 4 milliseconds
 {
  c4ms++;
  icnt1 = 0;
 }
 cbi(PORTD, TEST_PIN);                      // reset PORTD,TEST_PIN
}
```