# VARIABLE FREQUENCY DRIVE USING PWM

*The final schematic is on page 6 and the code can be found at github.com/hannahvsawiuk*

## Introduction

After finishing 2nd year in ECE, I had been introduced to motors and pulse width modulation (PWM). Although I only dealt with the DC applications of PWM in school projects, I recognized that it was a powerful technique and I became interested in its other uses.

I had used PWM for controlling the speed of a DC motor, so I began researching electric motors. But, because of my interest in electromagnetism, I focused my research on AC induction motors. From there, I learned about DC-AC inverters and AC-AC converters. This combined with my previous knowledge of rectifiers and switch-mode power supplies lead to my interest in variable frequency drives.

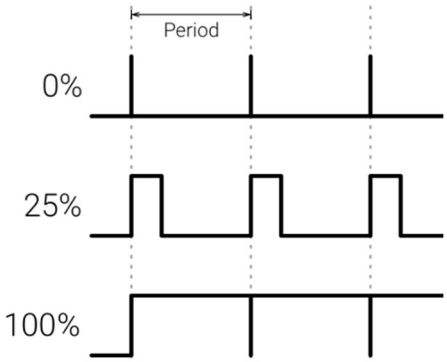



Figure 1: DC PWM

## PWM Theory Overview

In 2nd year, I learned about how motor speed is proportional to the average voltage delivered, and so the pulse width can be altered to vary the speed. So, for a DC motor, by delivering a square wave with 25% of the width will result in 25% of the maximum motor speed.

But, a sinusoidal current is needed for AC motors. To achieve this, the pulse width of the delivered voltage can be modulated to result in a sinusoidal average voltage. In figure 2, a sinusoidal PWM voltage signal (green) and its corresponding average voltage (blue) are shown. Since an induction motor is also an inductive load, a sinusoidal PWM voltage would generate a sinusoidal current.

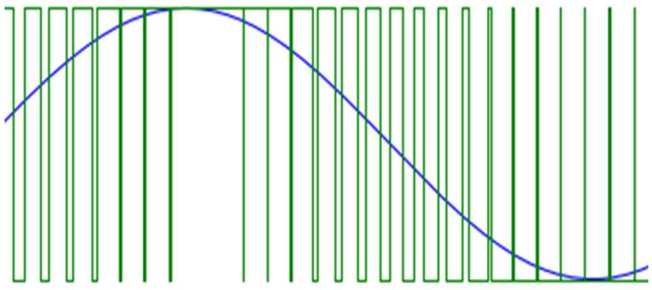


Figure 2: AC PWM

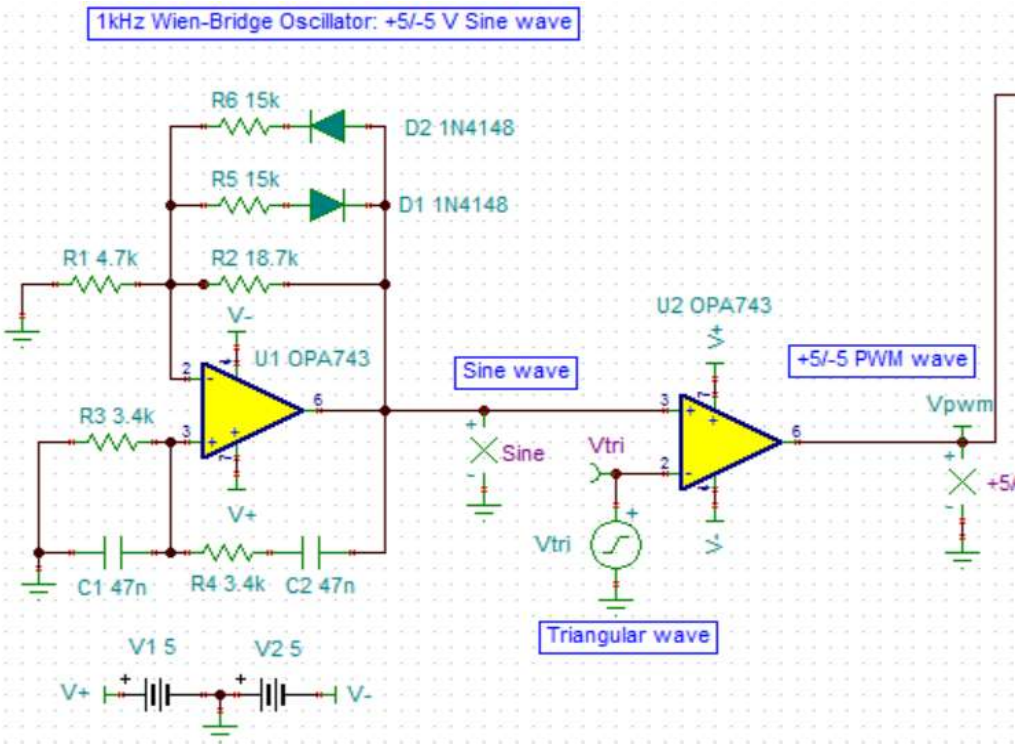



Figure 3: +5/-5 1kHz AC PWM generator

**Developing the schematic**

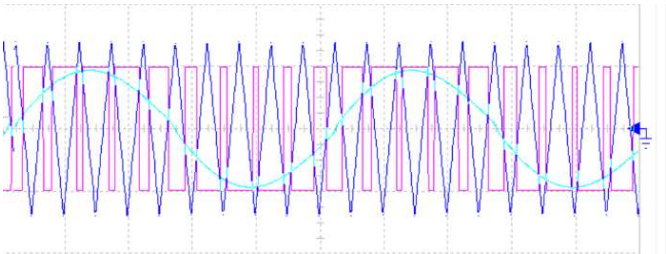
Since I had no experience with pulse width modulated sinusoids, I designed a test circuit to further my understanding. I used TINA-TI, a SPICE based software developed by Texas Instruments, to create the schematic. The circuit starts with a Wien-Bridge oscillator which produces a 1kHz sine wave. This signal is then fed into a comparator with a triangular signal (Vtri). The output of the comparator (Vpwm) is a PWM sine wave signal. Figure 4 shows the inputs and outputs of the comparator (U2). When the non-inverting input is greater than the inverting input, the comparator outputs a logic 1. If the non-inverting input is less, then a logic 0 is outputted. This produces a PWM signal with increasing and decreasing widths that changes correspondingly to the input sinusoid.

Figure 4: U2 comparator input and output signals

Sine

Vtri

Vpwm



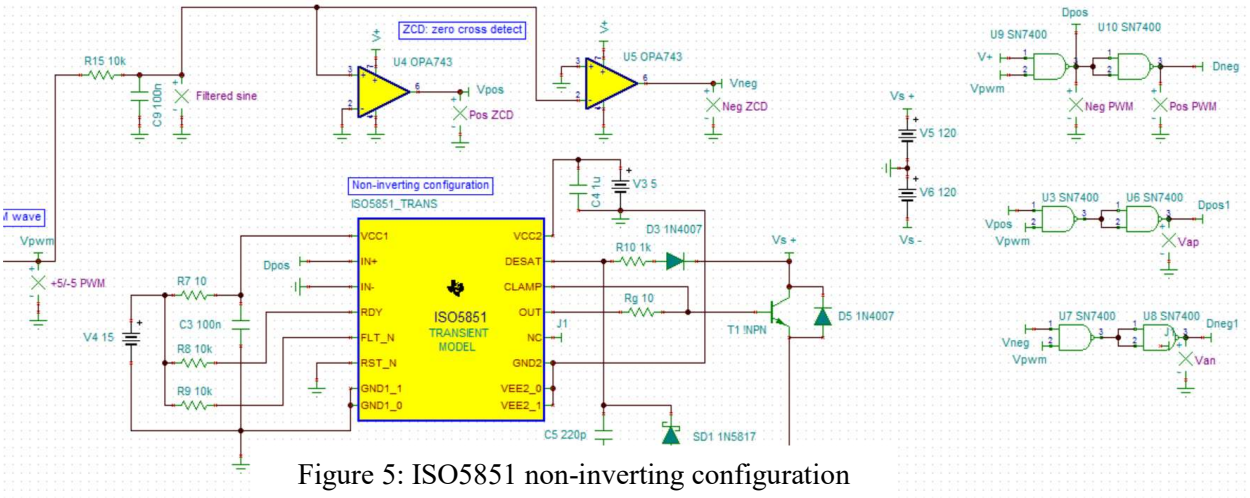



Figure 5: ISO5851 non-inverting configuration

Once the pulse-width modulated sinusoid was generated, I could manipulate the signal to become an appropriate input for a specific transistor driver. The PWM signal could be used directly with transistors, but this could result in transistor fault and current surges, which could damage equipment. Since transistors cannot switch states instantaneously, dead time is important to decrease the possibility of error.

In the first version of my design, I chose the ISO5851 since it is a very basic driver. I was limited in my choices due to the library constraints of the software. In my final design (page 6), I chose an Si8234 isolated gate driver because of its adjustable dead time, switching frequency, peak output current, and high isolation voltage.

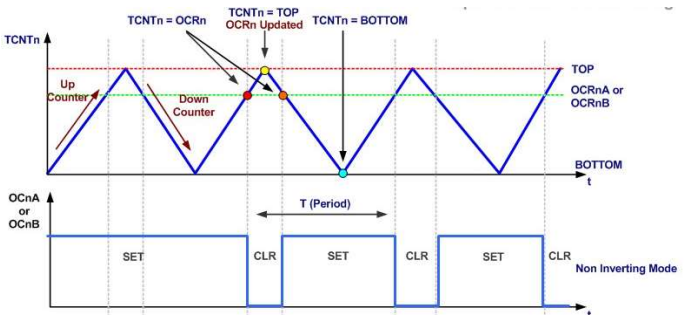



Figure 6: How output compare works

**Creating the code**

The code found at github.com/hannahvsawiuk in the 3 Phase PWM repository.

Once I finished the design, I began developing code for the driver system. The first code generated a pulse-width modulated sinusoid at a set frequency in a single pin. This was achieved by using two timers: one timer set to CTC (clear timer on compare) mode with a TOP value of its output compare register and the other set to Fast PWM mode with the default TOP value (255).

When the counter of the first timer reaches its output compare register value, an interrupt occurs and a routine is executed. This routine updates the output compare register of the second timer with a value from a look up table corresponding to the current index, and increments the look up table index. When the second counter is equal to the new register value, the output pin is cleared. The pin is set when the counter begins again. The look up table values increase and decrease sinusoidally which change the width of a single pulse in the PWM waveform. This method is called “fundamental direct digital synthesis”.

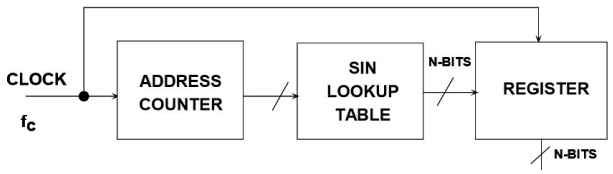



Figure 7: Updating the output compare register with a look up table value

Once the fixed frequency single phase code was tested, I designed a 3-phase fixed frequency version by adding timers. This was difficult because interrupt priority becomes an issue when more than one timer or interrupt is involved. I carefully chose the purpose of each timer to limit disturbances in the interrupt service routine and compare match clearing of the output pins.

To transition the code to a variable frequency version, I made the output compare register of the interrupt timer dependent on an input variable. Because the speed at which the look up tables are cycled through (one cycle is one period) depends on both the TOP value of its output compare register and the system clock, the TOP value can be changed to manipulate the frequency of the PWM signals. The challenge was to find a method that was simple. I tried several options and settled on using a potentiometer. The potentiometer was connected to an ADC pin of the microcontroller and a 5-volt source. The potentiometer changed the voltage at the ADC pin between 0 and 5 volts which would correspond to digital values between 0 and 1023.

In the main program loop, the digital value is extracted and converted to a frequency by multiplying by 120/1023 (120 Hz frequency range / maximum digital value). Then, in the interrupt service routine, the compare register was updated to the new value and the frequency would change.

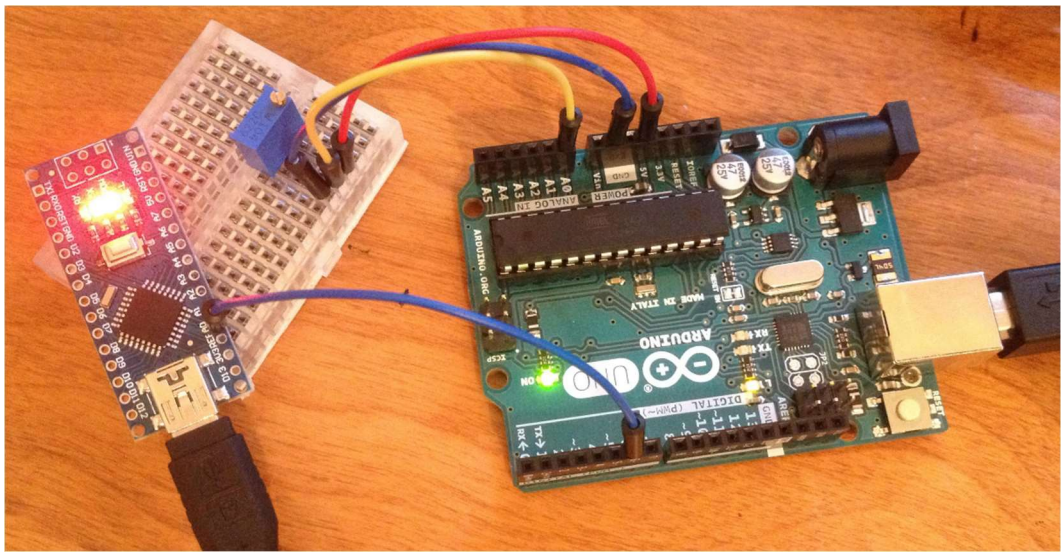



Figure 8: Testing setup

Overall, I found that the setup was the most difficult portion of the code. Knowing the constraints of interrupts and timers was essential. This was especially true during the progression from a smaller scale single phase fixed frequency version to a three-phase variable frequency model.

**Testing methods**

The Arduino Nano (ATmega328P) was used to do initial tests. A pin was set as an analog input and the baud rate was changed to improve the quality of the data. It was difficult to receive accurate measurements, but the overall behaviour could be extrapolated from the Arduino plots (figure 9). Once I have access to a lab, I will conduct further tests using an oscilloscope. To change the frequency of the PWM signals, a potentiometer was used. However, it possessed a low sensitivity (would only increase in resistance slightly after many turns). In future tests, a sensitive potentiometer will be used to see the system's response to rapid changes in the frequency.

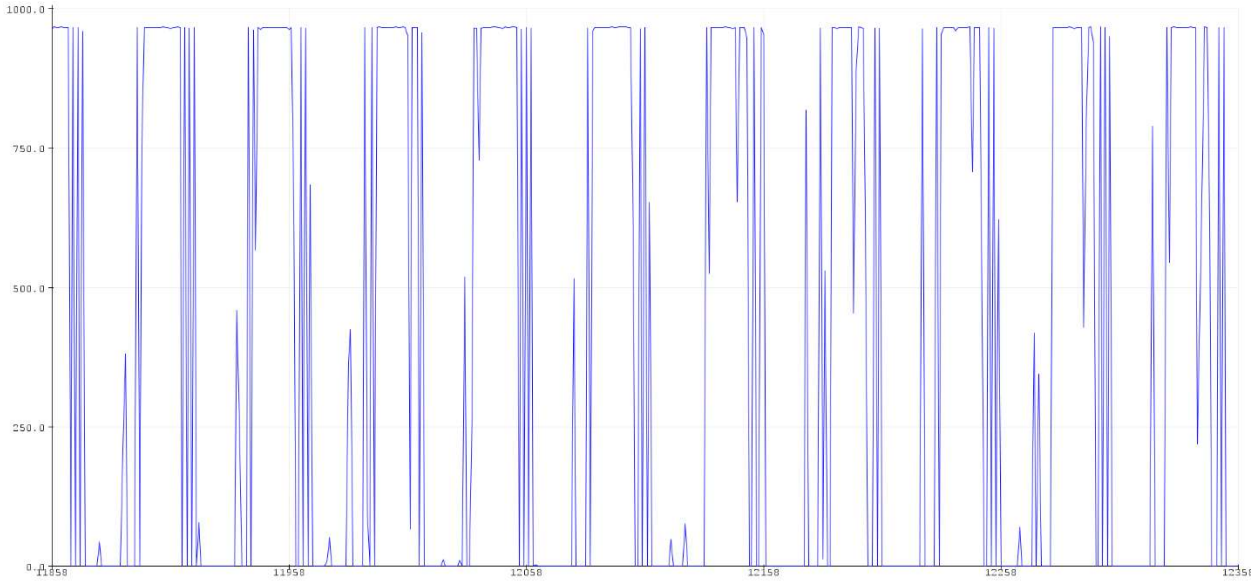



Figure 9: Test results

**Design specifications**

*SPICE based software:* TINA-TI

*Low-level simulator:* Atmel Studio 7.0 (for register level debugging)

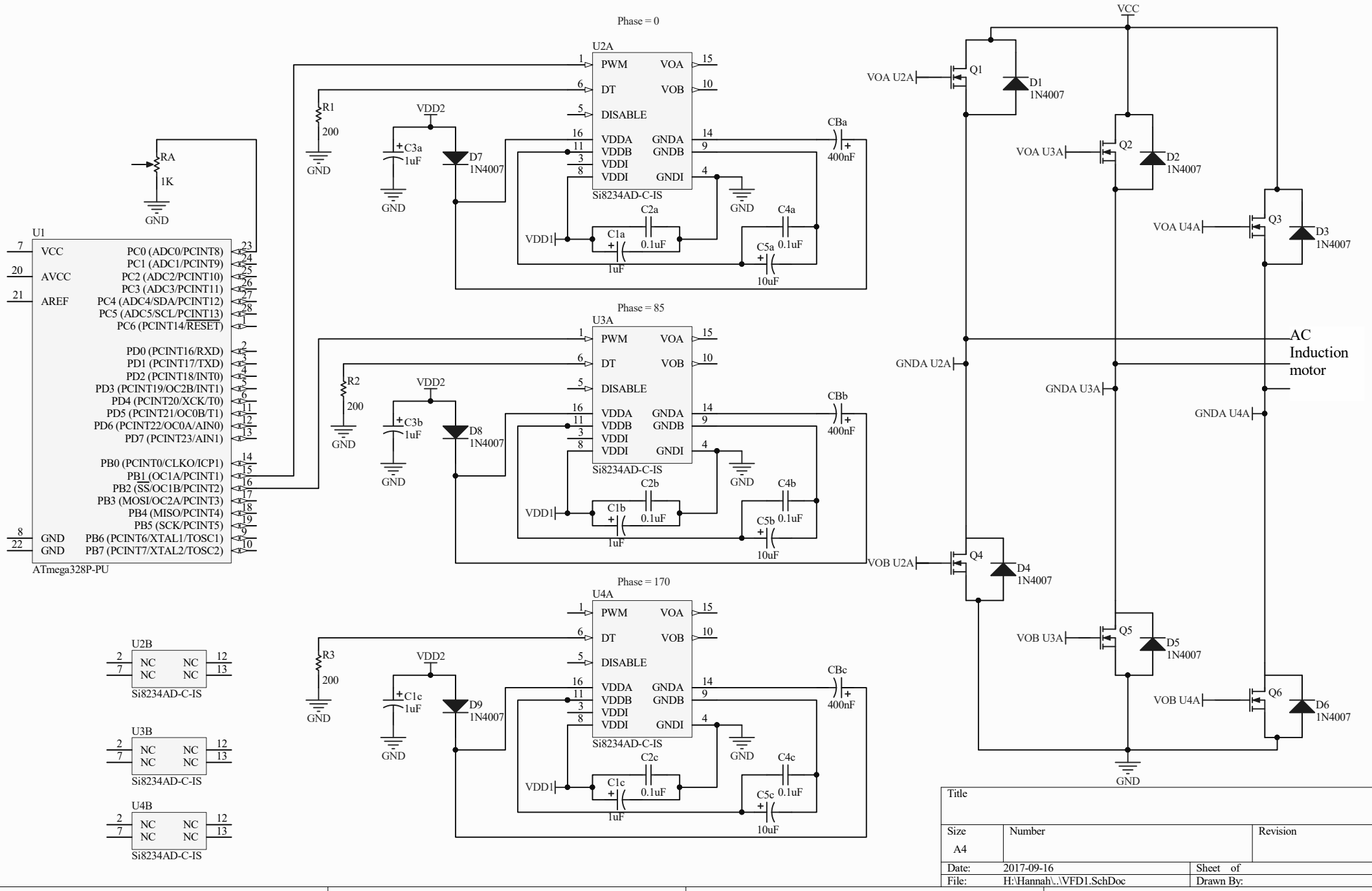
*IDE:* Arduino IDE

*Controller:* Arduino Uno (ATmega328P-PU)

*Schematic software:* Altium Designer

*Low pass filter designer*: http://sim.okawa-denshi.jp/en/CRtool.php

*Isolation bootstrap calculator:* https://www.silabs.com/tools/Pages/bootstrap-calculator.aspx


Phase = 0
Phase = 85
Phase = 170
U1
ATmega328P-PU
U2A
U3A
U4A
U2B
U3B
U4B
Si8234AD-C-IS
RA
1K
R1
R2
R3
200
C3a
C3b
C1c
1uF
D7
D8
D9
1N4007
CBa
CBb
CBc
400nF
C2a
C2b
C2c
C1a
C1b
C1c
0.1uF
C4a
C4b
C4c
C5a
C5b
C5c
10uF
VDD1
VDD2
GND
VCC
VOA U2A
VOA U3A
VOA U4A
VOB U2A
VOB U3A
VOB U4A
GNDA U2A
GNDA U3A
GNDA U4A
Q1
Q2
Q3
Q4
Q5
Q6
D1
D2
D3
D4
D5
D6
AC
Induction
motor
Title
Size
A4
Number
Revision
Date:
2017-09-16
Sheet of
File:
H:\Hannah\..\VFD1.SchDoc
Drawn By: