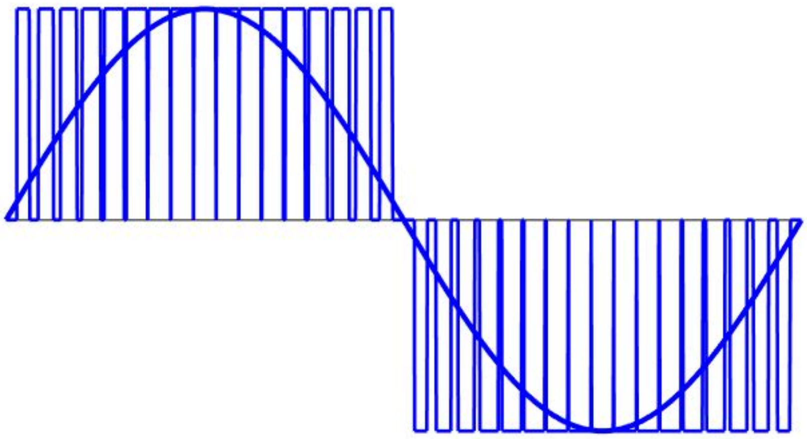
ELEX 7660 Digital System Design
Term Project
Single Phase Variable Frequency Drive



Abstract

This document is a lab report for the British Columbia Institute of Technology (BCIT) course ELEX 7660 detailing the steps taken to design, build and test a single phase variable frequency drive (VFD) using an Altera Cyclone IV DE0 Nano Field Programmable Gate Array (FPGA). The lab took place over a period of five weeks, and was completed by a team of two 3rd year electrical engineering students. The results of the lab were positive, with the VFD meeting most of the required specifications being successfully built, and both students gaining valuable insight into the design process.

Group 16
Tyson Whyte
Jacob Lagassé
Set 6S

Submitted: April 13th, 2018

Tyson Whyte
Jacob Lagasse

Single Phase Variable Frequency Drive

Introduction

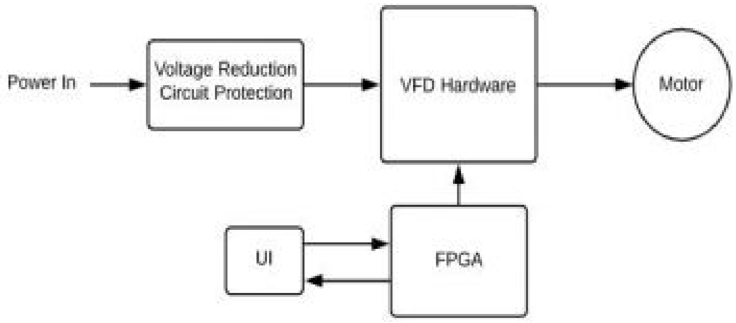
This project encompassed building a single phase variable frequency drive (VFD) to control the torque, and the speed of a single phase AC induction motor. Using variable frequency drives to operate AC induction motors results in reduced inrush current when starting the motor, higher torque at low speeds, tuneable torque to speed characteristics for differing motor applications, high speed control, and more efficient Operation. The motivation for this project came from the team members experience in field as electricians installing and servicing VFDs for high power motors; after having seen these devices in operation numerous times over many years, both team members were keenly interested in learning about their inner workings.

The VFD operates based off of the principle of frequency modulation. This is the most popular type of VFD design currently in use today. Using a rectified and filtered AC input, the DC signal is switched at different frequencies dependant upon the speed setting of the drive. This switching is performed by IGBTs to simulate the required power characteristics similar to regular AC signal operation. Using this method affords more control over the motor's power characteristics. [1]

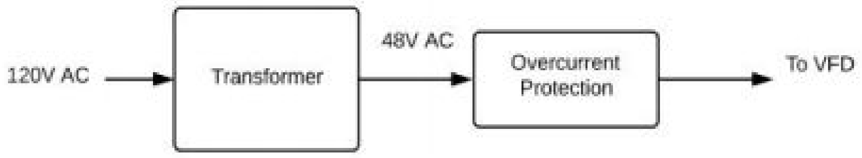
Using the Altera Cyclone IV FPGA board as a controller, the system controls the triggering of the IGBTs based off of inputs received from a push button for stop-start control, and a pair of buttons which allow the adjustment of the speed setting of the motor. This speed setting is displayed on a series of four 7-segment displays. [4]

The carrier produced by the switching frequencies can results in reflected wave voltages that can cause voltage spikes that are up to 2.5 times the intended voltage output. [1] For this reason the components utilised in this project have voltage ratings adequate for these spikes. The components are also sized to accommodate the typical inrush current of the motor to protect them during development and calibration of the drive. The VFD system utilises optical isolators to protect the FPGA from the inductive motor load. Additionally, to mitigate safety hazards our VFD will operate at approximately 60 V AC.

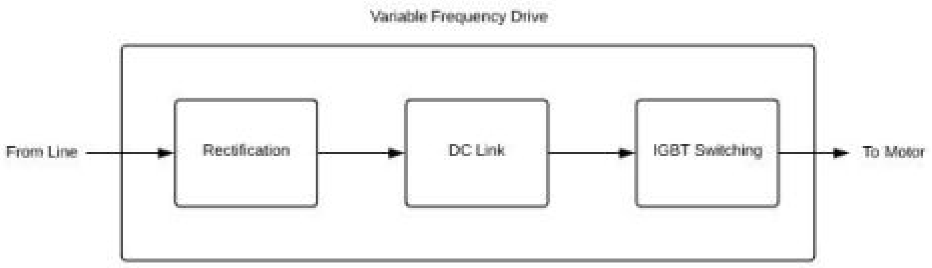
Design



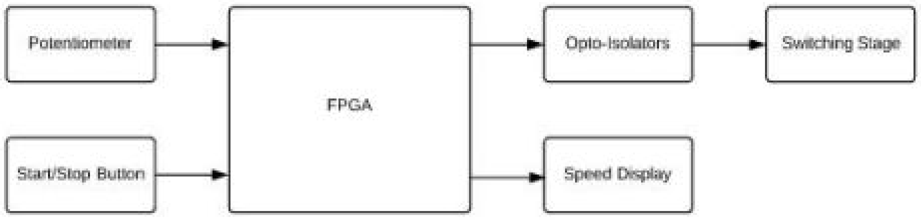
In order to eliminate problems with harmonics and ground faults, it is necessary to use a reduced voltage isolated voltage supply to operate the VFD. [1] This was accomplished by utilising a variac autotransformer to step down the line voltage of 120V AC to a voltage of approximately 60V AC to help mitigate safety hazards. This transformers has the capacity to supply the initial inrush current to the motor without damage (rated to 5A). Additionally, the transformer has overcurrent protection by means of a replaceable fuse. Overload protection is not vital since the current to the system is monitored during its operation at all times by means of a clamp-on multimeter.



In the rectification stage the system utilises a bridge rectifier. This rectifier has a peak inverse voltage rating (PIV) that is able to handle the reflected wave voltages. In the DC link stage the rectified signal is filtered into a DC signal that goes to the switching stage. In the switching stage IGBTs are used to switch the DC signal and change the frequency of the PWM output. [2]

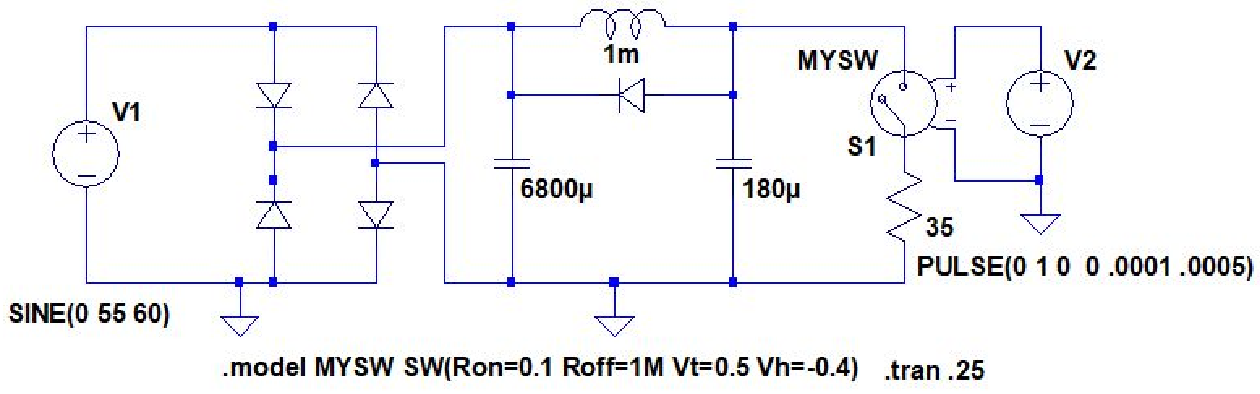


The FPGA is used to control the user interface as well as control the triggering in the switching stage of the VFD. In order to isolate the FPGA from the higher voltage circuits found within the VFD, opto isolators are used in order to relay the signals to the switching stage of the VFD. The VFD has a seperate DC power supply in order to operate the electronics such as the IGBTs and buttons. This greatly simplifies the design as the system does not have to reduce and condition the line voltage again in order to run the electronics. A separate power supply is used to power the switching stage with +16V DC, and to power the opto-isolators with +5V DC.



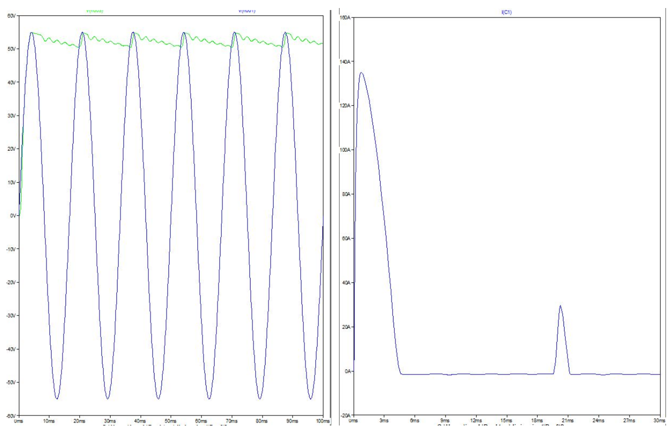
Simulation

Simulation of the system was accomplished by using LTspice to model the rectification, dc link and switching systems. With these simulations the team was able to determine the current and voltage ratings required for the various systems utilised.


V1
SINE(0 55 60)
1m
6800µ
180µ
MYSW
S1
V2
35
PULSE(0 1 0 0 .0001 .0005)
.model MYSW SW(Ron=0.1 Roff=1M Vt=0.5 Vh=-0.4)
.tran .25

LTspice Simulation of VFD Rectification, DC Link and Switching Systems

With the selected components the figures below indicate the simulated operation of our design with a switching load. As seen in the right figure, when voltage is initially applied to the system the 6800uF capacitor draws a large inrush current. This current was approximately 130A, which would have triggered our overcurrent protection, and damaged components. An easy solution to this is to slowly increase the voltage initially to change the capacitor slowly. Additionally, due to the switching load we found that we were generating transient voltages of up to 700V. Due to these voltage spikes we needed to install a free-wheeling diode in parallel with the inductor. The figure on the left shows our input voltage to the system and the smoothed ripple output that the load sees. This amount of ripple was adequate to power our load.



LTspice Simulation Voltage and Current Waveforms

Construction

The VFD system is composed of four primary systems: The first system is the rectifier, the second system is the DC link, the third system is the switching system and finally the fourth is the control system. These four systems are utilised to control a single phase 120V AC squirrel cage induction motor.

Rectifier System

The Rectifier System receives an AC input of 60 volts at a maximum of 5 amps and outputs a DC voltage of approximately 65 volts. The first stage of the system is a 120V AC rated variac that is used to adjust the 120V AC wall power to the required 60V. The output of the variac is connected to a bridge rectifier chip. This rectifier chip must have a peak inverse voltage rating (PIV) that is able to withstand the reflected wave voltages, however at the very minimum this rating must be as high as the peak voltage of the AC input. The bridge rectifier outputs a pulsating DC output to the DC Link System.

DC Link System

In the DC Link System the rectified signal is filtered into a flat DC signal that is output into the Switching System. The filter is a high power low-pass LC circuit utilising two capacitors, one inductor and one free-wheeling diode to protect the system from high voltage spikes from the inductor. On system startup there are significant inrush currents into this system (due to the capacitors charging), and care must be taken to mitigate them. For this project these currents were mitigated by slowly increasing the voltage on the variac from 0V to 60V AC when turning the VFD on.

Switching System

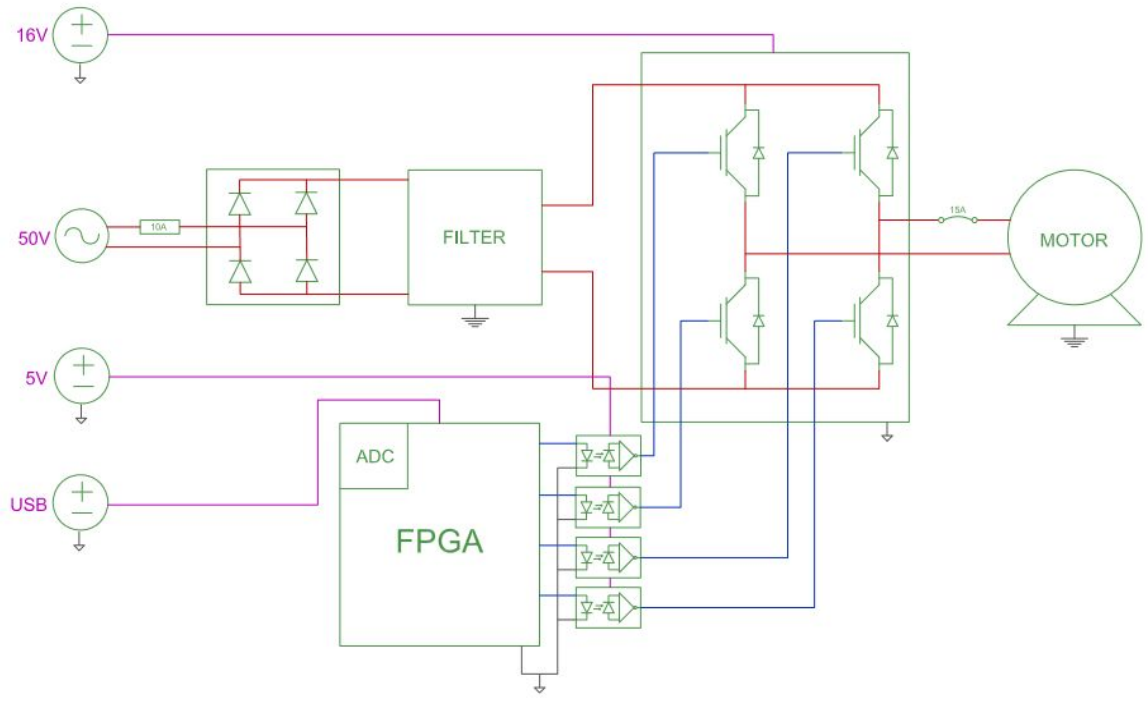
The Switching System contains a power module that contains six IGBTs in a convenient single-chip package. Four of the six IGBTs are utilised for this project. The DC Link System is fed into the power module and the IGBTs are switched on and off by the Control System, allowing power to flow from the negative and positive buses from the DC Link to the attached motor.

Control System

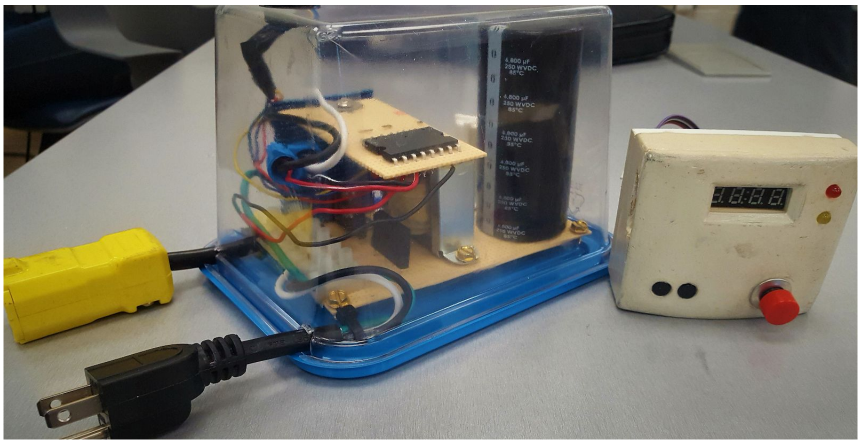
The Control System is the heart of the VFD, and contains the FPGA and the opto-isolators used to protect it from transient voltages from the Switching and DC Link systems. The opto-isolators utilised are high-speed to minimize propagation delay to the system, as the switching frequency of the VFD is quite high (2.5kHz). The FPGA is connected to the opto-isolators by way of two output pins that output the PWM, and the opto-isolators are in turn connected to the Switching System's IGBT module inputs.

Single Phase Variable Frequency Drive

Control of the FPGA is achieved by way of three buttons: two buttons to increase and decrease the speed and one button to start and stop the motor. Feedback from the system is given by way of a four digit 7-segment display and one led to indicate power on (red), and one led to indicate motor running (yellow).



VFD System Wiring Diagram



VFD System with Control Panel

Results

PWM Output

Testing of the VFD began with testing each system independently. After approximately two weeks of testing and modifying the verilog code to output the proper PWM signals, the system was able to accurately reproduce an SPWM waveform. Some difficulty was encountered in generating the sine and triangle waves in system verilog (sine values are difficult to generate on an FPGA).

After some time the team switched to generating the waves in Matlab (see Appendix A) and creating arrays of PWM values (1,0 and -1). These arrays were generated with a switching frequency of 2.5kHz (which is set by the triangle wave frequency in the code), and sine wave frequencies from 5Hz to 30Hz. A constant voltage to frequency ratio of 2V/Hz was utilised to maintain constant flux in the motor. It should be noted that these arrays are not included in the code due to their length of ten thousand values.

Buttons and Display

Creating the code for the display and the buttons initially went rather quickly, however after testing the code with the buttons, some difficulties developed. Debouncing the buttons became quite difficult, which was possibly due to the poor quality of the buttons utilised (caused by budget constraints). It took approximately one week of modifying and tweaking this code to achieve a control system that could reliably detect a button push and latch the result in the code. Subsequent testing produced positive results, with the system able to accurately detect button pushes and display the proper frequency selected on the 7-segment displays.

Rectifier System and DC Link

Building of this system presented significant challenges, as the initial filter in the DC Link System did not function when places under load, thought it initially seemed to function when tested unloaded (it displayed a flat DC output). After some investigation it was determined that the system required much larger inductors and capacitors, which were quickly ordered. See the parts list in Appendix B for the values. Subsequent testing of the new system produced positive results, with a flat DC output being produced under a load of 20 ohms at a voltage of 30V AC.

Final Testing

After having tested the VFD’s component systems separately with positive results, the team first tested the system with only the Switching System and the control system connected. This system required some modifications to the wiring and minor changes to the code, but was quickly able to produce the required results of controlling the igbts and output a PWM output that varied from 0Hz to 30Hz in 5Hz steps. This test was conducted with the motor disconnected due to the DC Link not being connected (for safety reasons).

The final test of the system was completed on the day of demoing the VFD to the ELEX 7660 class. All of the separate systems were connected and powered, and the team attempted to turn the motor. However, instantly after powering the system and activating the VFD, the overcurrent protection on the variac blew, and the system was subsequently powered off. The cause of this event was later determined to be the power supply for the IGBT power module not being a floating supply, which is required for supplying the high side IGBTs; their common points are connected to the motor output terminals, and without a floating supply the positive bus as shorted to ground through this connection point.

The DC Link and Rectifier Systems were disconnected, and the VFD was simply demoed with the Control and Switching Systems connected. The sinusoidal PWM output was displayed on an oscilloscope, and the system was shown working by turning it on and off, and by changing the frequency output from 0Hz to 30Hz.

These results are considered positive, as the team was able to successfully build and test a functional SPWM output. The VFD itself is considered functional, as the issue was determined to be hardware related. The digital system design of the VFD was fully operational, but hardware failure impeded the overall system design from completing the initial goal of controlling the motor. It is recommended that for future implementations of this design the IGBT chip be replaced with another IC package or individual IGBTs.

Appendix A

Code

MATlab PWM Array Generation Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%%% pwmgenerator.m
%%%
%%% Sinusoidal Pulse Width Modulation (SPWM) Generator and Plotter
%%%
%%% Authors: Tyson Whyte and Jacob Lagasse
%%%
%%% Description: This module generates the arrays for SPWM and plots
%%% the sinusoidal, triangle and PWM waveforms. It is highly
%%% customizable by changing the sampling frequency (fs), triangle
%%% wave frequency (Fc2),sine wave frequency (Fc) and the sine wave
%%% amplitude (amplitude).
%%%
%%% April 2018
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

%%%close and clear all
close all;
clear all;
clc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
%%% Design considerations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
      %%% must maintain Voltage/frequency ratio of nameplate at 120V/60Hz = 2
V/Hz
      %%% full voltage (60V DC source): freq = (60V)/(2V/Hz) = 30hz (half
speed)
      %%% 6 frequency steps: 5 to 30Hz
      %%% Hz Amplitude    Voltage
      %%% 5       0.17        10V
      %%% 10      0.33        20V
      %%% 15      0.50        30V
      %%% 20      0.67        40V
      %%% 25      0.83        50V
    %%% 30        1.00        60V
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%% Time, frequency and amplitude specifications
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   amplitude = 1;
   Fs = 5000000;                  % sampling frequency (samples/sec)
   Fc = 60;                        % frequency of sine wave in hertz
   Fc2 = 2100;                    % frequency of tri wave in hertz
   dt = 1/Fs;                     % seconds per sample
   StopTime = 1/(60);             %Stoptime in seconds
   t = (0:dt:StopTime-dt)';       %Time array to step through in seconds
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % pulse width generation

   ii = 0; %%counter for sine,triangle and PWM arrays

   %%%main loop to calculate waves and populate arrays
      for tt = (0:dt:StopTime-dt)
      ii = ii+1;%%increment counter
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%% Sine and triangle wave generation
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      ysin(ii) = amplitude*sin(2*pi*Fc*tt); %%calculate sine wave value
      ytri(ii) = triangle(2*pi*Fc2*tt); %%calculate triangle wave value
      yzero(ii) = 0*tt; %%zero line on graph
      ypwm(ii) = 0; %%set default pwm value

            if(ysin(ii)>0) %%for the postive half cycle
            if((ysin(ii)>ytri(ii))&&(ysin(ii)> 0.02)) %%if sin is greater
than triangle
                  ypwm(ii) = 1; %%Set pwm to one
            else
                  ypwm(ii) = 0; %%else set pwm to zero
            end
            end

            if(ysin(ii)<0) %%for the negative half cycle
            if((ysin(ii) < ytri(ii))&&(ysin(ii)< -0.02)) %%if sin is greater
than triangle
                  ypwm(ii) = -1; %%set pwm to negative one
            else
                  ypwm(ii) = 0; %%else set pwm to one
            end
            end
      end %%end for loop

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Plot the signals versus time
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   figure(1);
   plot(t,ysin,'bl','LineWidth', 3); %%plot sine wave
   hold on;
   plot(t,ytri);  %%plot triangle wave
   hold on;
   plot(t,yzero, 'k'); %plot zero line
   xlabel('time (in seconds)');
   title('Signal versus Time');
   zoom xon;
   set(gca,'Color',[1 1 1]) %%set background colour
   hold on;
   plot(t,ypwm, 'bl','LineWidth', 2); %%plot pwm output

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%triangle wave generator fuction (same use as sin() )
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 function y = triangle(t)
y = 2*abs(mod((t+1.5*pi)/pi, 2)-1)-1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Main VFD Module

```
/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
//Created by Jacob Lagasse and Tyson Whyte
//This file is the top level module for the VFD.
//It calls the pushbutton modules, display modules, and PWMgen module
/////////////////////////////////////////////////////////////////////////////

//VFD is the top-level module
//Inputs: CLOCK_50-the 50MHz clock,
//    button-the input from the start/stop buttons
//          speed_data-the frequency setting
//          button_u-the input signal from the frequency up //button
//          button_d-the input signal from the frequency down //button
//Outputs: leds-controls the 7 LEDs in the display
//          ct-enables which display to light up
//           LED_ON-enable the running light
//          RUN-enables the running mode
//           PWM_pos_out-switches IGBTs on positive pulses
//           PWM_neg_out-switches IGBTs on negative pulses
//           test-testing LED on FPGA
module VFD(input logic button, input logic CLOCK_50, input logic [31:0]
speed_data, input logic button_u, input logic button_d,
        output logic LED_ON, RUN, output logic [7:0] leds, output logic
[3:0] ct,output logic test, output logic PWM_pos_out, output logic
PWM_neg_out);

logic [7:0] freq_setting;
logic pulse_out_u;
logic pulse_out_d;
logic [7:0] counter_up = 32;//used for count up presses
logic [7:0] counter_down = 32;//used for counte down presses

always_comb begin
freq_setting = counter_up - counter_down;//keeps teh frequency setting
between 0 and 6
end

//this logic below keeps the frequency setting between 0 and 6 depending on
//button presses
always @(posedge pulse_out_u) begin
      if (freq_setting >= 6)
      counter_up <= counter_up;
      else
      counter_up <= counter_up+1;
      end

always @(posedge pulse_out_d) begin
      if (counter_down >= counter_up)
      counter_down <= counter_down;
      else
      counter_down <= counter_down+1;
end

//instantiate dependent modules.
button start_stop(.clk(CLOCK_50),.press_l(button),.press_u(button_u),
.press_d(button_d),.status(RUN),.pulse_out_u,.pulse_out_d);
lab1 display_speed(.CLOCK_50,.freq(freq_setting),.leds,.ct);
PWMgen  igbt_outputs(.PWM_pos_out, .PWM_neg_out, .freq_in(freq_setting),
.startstop_in(RUN), .CLOCK_50 ) ;
endmodule
///////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////
```

Display Modules

```
///////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
//Created by Jacob Lagasse and Tyson Whyte
//This file creates the functionality for display which displays the
//frequency settings of the VFD
//this is a modified version of lab 1
/////////////////////////////////////////////////////////////////////////////
display is the top-level display module
```

Tyson Whyte
Jacob Lagasse

```
//Inputs: CLOCK_50-the 50MHz clock,
//    freq-the frequency setting for the VFD
//Outputs: leds-controls the 7 LEDs in the display
//          ct-enables which display to light up
module display ( input logic CLOCK_50,     // 50 MHz clock
             input logic [7:0] freq,    // speed setting
             output logic [7:0] leds,  // 7-seg LED cathodes
             output logic [3:0] ct ) ; // digit enable

   logic [1:0] digit ;
   logic [3:0] idnum ;//frequency setting
   logic    clk ;

//instatiate dependant display modules
   digitselect dig_sel (.digit,.ct) ;
   displayfreq  disp_freq  (.digit,.freq,.idnum) ;
   decode decode_1 (.num(idnum),.leds) ;
   displayclk disp_clk ( CLOCK_50, clk ) ;

//cycles which isplay is powered
   always_ff @(posedge clk)
      digit <= digit + 1'b1 ;
endmodule
///////////////////////////////////////////////////////////////////////////


//This is a setup file for the display
///////////////////////////////////////////////////////////////////////////
// megafunction wizard: %ALTPLL%
// ...
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
// ...

module displayclk ( inclk0, c0);

      input inclk0;
      outputc0;

      wire [0:0] sub_wire2 = 1'h0;
      wire [4:0] sub_wire3;
      wire  sub_wire0 = inclk0;
      wire [1:0] sub_wire1 = {sub_wire2, sub_wire0};
      wire [0:0] sub_wire4 = sub_wire3[0:0];
      wire  c0 = sub_wire4;

      altpll altpll_component ( .inclk (sub_wire1), .clk
      (sub_wire3), .activeclock (), .areset (1'b0), .clkbad
      (), .clkena ({6{1'b1}}), .clkloss (), .clkswitch
      (1'b0), .configupdate (1'b0), .enable0 (), .enable1 (),
      .extclk (), .extclkena ({4{1'b1}}), .fbin (1'b1),
      .fbmimicbidir (), .fbout (), .fref (), .icdrclk (),
.locked (), .pfdena (1'b1), .phasecounterselect
({4{1'b1}}), .phasedone (), .phasestep (1'b1),
.phaseupdown (1'b1), .pllena (1'b1), .scanaclr (1'b0),
.scanclk (1'b0), .scanclkena (1'b1), .scandata (1'b0),
.scandataout (), .scandone (), .scanread (1'b0),
.scanwrite (1'b0), .sclkout0 (), .sclkout1 (),
.vcooverrange (), .vcounderrange ());

defparam
      altpll_component.bandwidth_type = "AUTO",
      altpll_component.clk0_divide_by = 25000,
      altpll_component.clk0_duty_cycle = 50,
      altpll_component.clk0_multiply_by = 1,
      altpll_component.clk0_phase_shift = "0",
      altpll_component.compensate_clock = "CLK0",
      altpll_component.inclk0_input_frequency = 20000,
      altpll_component.intended_device_family = "Cyclone IV E",
      altpll_component.lpm_hint = "CBX_MODULE_PREFIX=lab1clk",
      altpll_component.lpm_type = "altpll",
      altpll_component.operation_mode = "NORMAL",
      altpll_component.pll_type = "AUTO",
      altpll_component.port_activeclock = "PORT_UNUSED",
      altpll_component.port_areset = "PORT_UNUSED",
      altpll_component.port_clkbad0 = "PORT_UNUSED",
      altpll_component.port_clkbad1 = "PORT_UNUSED",
      altpll_component.port_clkloss = "PORT_UNUSED",
      altpll_component.port_clkswitch = "PORT_UNUSED",
      altpll_component.port_configupdate = "PORT_UNUSED",
      altpll_component.port_fbin = "PORT_UNUSED",
      altpll_component.port_inclk0 = "PORT_USED",
      altpll_component.port_inclk1 = "PORT_UNUSED",
      altpll_component.port_locked = "PORT_UNUSED",
      altpll_component.port_pfdena = "PORT_UNUSED",
      altpll_component.port_phasecounterselect = "PORT_UNUSED",
      altpll_component.port_phasedone = "PORT_UNUSED",
      altpll_component.port_phasestep = "PORT_UNUSED",
      altpll_component.port_phaseupdown = "PORT_UNUSED",
      altpll_component.port_pllena = "PORT_UNUSED",
      altpll_component.port_scanaclr = "PORT_UNUSED",
      altpll_component.port_scanclk = "PORT_UNUSED",
      altpll_component.port_scanclkena = "PORT_UNUSED",
      altpll_component.port_scandata = "PORT_UNUSED",
      altpll_component.port_scandataout = "PORT_UNUSED",
      altpll_component.port_scandone = "PORT_UNUSED",
      altpll_component.port_scanread = "PORT_UNUSED",
      altpll_component.port_scanwrite = "PORT_UNUSED",
          altpll_component.port_clk0 = "PORT_USED",
      altpll_component.port_clk1 = "PORT_UNUSED",
      altpll_component.port_clk2 = "PORT_UNUSED",
      altpll_component.port_clk3 = "PORT_UNUSED",
      altpll_component.port_clk4 = "PORT_UNUSED",
```

Single Phase Variable Frequency Drive

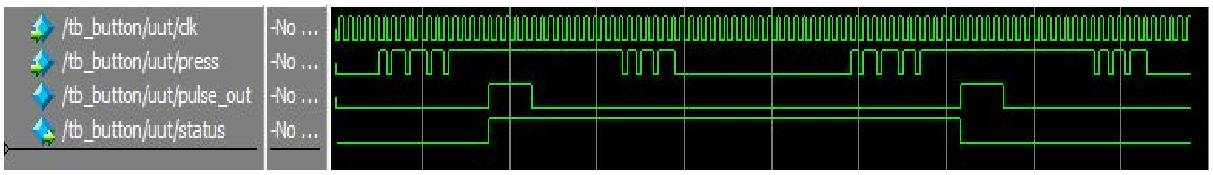
```
            altpll_component.port_clk5 = "PORT_UNUSED",
            altpll_component.port_clkena0 = "PORT_UNUSED",
            altpll_component.port_clkena1 = "PORT_UNUSED",
            altpll_component.port_clkena2 = "PORT_UNUSED",
            altpll_component.port_clkena3 = "PORT_UNUSED",
            altpll_component.port_clkena4 = "PORT_UNUSED",
            altpll_component.port_clkena5 = "PORT_UNUSED",
            altpll_component.port_extclk0 = "PORT_UNUSED",
            altpll_component.port_extclk1 = "PORT_UNUSED",
            altpll_component.port_extclk2 = "PORT_UNUSED",
            altpll_component.port_extclk3 = "PORT_UNUSED",
            altpll_component.width_clock = 5;
endmodule
////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////
//This decoder selects which 7 segment display will be active
// It converts a 2 bit input "digit" into a 4 bit active high output "ct"
module digitselect(input logic [1:0] digit,
output logic [3:0] ct) ;
always_comb begin
 unique case (digit)
0: ct = 4'b0001 ;
1: ct = 4'b0010 ;
2: ct = 4'b0100 ;
default: ct = 4'b1000 ; //works for the 3rd bit
 endcase
end
endmodule
////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////
//dislpayfreq is where the numbers to display are selected
//Inputs: digit-inputs which segmetn is active,
//    freq-the frequency setting for the VFD
//Outputs: idnum- which numbers are assigned to the 7 segment displays
module displayfreq (input logic [1:0] digit, input logic [7:0] freq,
output logic [3:0] idnum) ;
always_comb begin
//for the first frequency setting (5Hz)
if (freq == 1) begin
      unique case (digit)
      0: idnum = 4'd5 ;
      1: idnum = 4'd0 ;
      2: idnum = 4'd0 ;
      default: idnum = 4'd0 ; //works with the 3rd bit
      endcase
      end
      //for the second frequency setting (10Hz)
else if (freq == 2) begin
      unique case (digit)
      0: idnum = 4'd0 ;
      1: idnum = 4'd1 ;
      2: idnum = 4'd0 ;
      default: idnum = 4'd0 ; //works with the 3rd bit
      endcase
      end
else if (freq == 3) begin
      unique case (digit)
      0: idnum = 4'd5 ;
      1: idnum = 4'd1 ;
      2: idnum = 4'd0 ;
      default: idnum = 4'd0 ; //works with the 3rd bit
      endcase
      end
else if (freq == 4) begin
      unique case (digit)
      0: idnum = 4'd0 ;
      1: idnum = 4'd2 ;
      2: idnum = 4'd0 ;
      default: idnum = 4'd0 ; //works with the 3rd bit
      endcase
      end
else if (freq == 5) begin
      unique case (digit)
      0: idnum = 4'd5 ;
      1: idnum = 4'd2 ;
      2: idnum = 4'd0 ;
      default: idnum = 4'd0 ; //works with the 3rd bit
      endcase
      end
else if (freq == 6) begin
      unique case (digit)
      0: idnum = 4'd0 ;
      1: idnum = 4'd3 ;
      2: idnum = 4'd0 ;
      default: idnum = 4'd0 ; //works with the 3rd bit
      endcase
      end
else begin
      unique case (digit)
      0: idnum = 4'd0 ;
      1: idnum = 4'd0 ;
      2: idnum = 4'd0 ;
      default: idnum = 4'd0 ; //works with the 3rd bit
    endcase
      end
end
endmodule
```

```
///////////////////////////////////////////////////////////////////////////


///////////////////////////////////////////////////////////////////////////
//decode takes which number to display and decodes it to each LED segment
//It has a 4 bit input "num", which is converted into 7 bit active low
//outputs
module decode (input logic [3:0] num,
output logic [7:0] leds) ;
always_comb begin
 unique case (num)
0: leds = 8'b1100_0000 ; //bits are inverted due to the active low output
1: leds = 8'b1111_1001 ;
2: leds = 8'b1010_0100 ;
3: leds = 8'b1011_0000 ;
4: leds = 8'b1001_1001 ;
5: leds = 8'b1001_0010 ;
6: leds = 8'b1000_0010 ;
7: leds = 8'b1111_1000 ;
8: leds = 8'b1000_0000 ;
default: leds = 8'b1001_0000 ; //acts as bit 7
 endcase
end
endmodule
// 1000_0000 .
// 0100_0000 g
// 0010_0000 f
// 0001_0000 e
// 0000_1000 d
// 0000_0100 c
// 0000_0010 b
// 0000_0001 a
/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////
```

Button Modules

```
///////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
//Created by Jacob Lagasse and Tyson Whyte
//This file creates the functionality for the momentary contact active high
//push buttons used to interface with the VFD
///////////////////////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////////////////////////
//button is the top-level pushbutton module
//Inputs: clk-the 50MHz clock,
//    press_l-the start button,
//    press_u-the button to increase the frequency,
//    press_d-the button to decrease the frequency,
//Outputs: status-the on/off status of the VFD,
//          pulse_out_u-increases the frequency,
//          pusle_out_d-decreases the frequency
module button(input logic clk, press_l,press_u,press_d,
              output logic status=0,output logic pulse_out_u,
              output logic pulse_out_d);

//logical variables for the start/stop button
logic pulse_en_l;
logic pulse_out;
logic Q1,Q2,Q2n;
logic reset_ff_l = 0;
//logical variables for the frequency up button
logic pulse_en_u;
logic Q1_u,Q2_u,Q2n_u;
logic reset_ff_u = 0;
//Logical variable for the frequency down button
logic pulse_en_d;
logic Q1_d,Q2_d,Q2n_d;
logic reset_ff_d = 0;

//instantiate the module for the 3 push buttons
pulse
u1(.clk_p(clk),.press(press_l),.pulse_en(pulse_en_l),.reset(reset_ff_l));
d_ff
d1(.clk_ff(clk),.pulse_en(pulse_en_l),.D(press_l),.reset(reset_ff_l),.Q(Q1))
;
d_ff
d2(.clk_ff(clk),.pulse_en(pulse_en_l),.D(Q1),.reset(reset_ff_l),.Q(Q2));

pulse
u2(.clk_p(clk),.press(press_u),.pulse_en(pulse_en_u),.reset(reset_ff_u));
d_ff
d3(.clk_ff(clk),.pulse_en(pulse_en_u),.D(press_u),.reset(reset_ff_u),.Q(Q1_u
));
d_ff
d4(.clk_ff(clk),.pulse_en(pulse_en_u),.D(Q1_u),.reset(reset_ff_u),.Q(Q2_u));

pulse
u3(.clk_p(clk),.press(press_d),.pulse_en(pulse_en_d),.reset(reset_ff_d));
d_ff
d5(.clk_ff(clk),.pulse_en(pulse_en_d),.D(press_d),.reset(reset_ff_d),.Q(Q1_d
));
d_ff
d6(.clk_ff(clk),.pulse_en(pulse_en_d),.D(Q1_d),.reset(reset_ff_d),.Q(Q2_d));

//set the not values for the d-flipflops
assign Q2n = ~Q2;
//sets condition for pulse output
assign pulse_out = Q1 & Q2n;

assign Q2n_u = ~Q2_u;
assign pulse_out_u = Q1_u & Q2n_u;

assign Q2n_d = ~Q2_d;
assign pulse_out_d = Q1_d & Q2n_d;

//toggles the on/off status of the VFD with the debounced pushbutton pulse
always @(posedge pulse_out)
      status <= ~status;
endmodule
//////////////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////////////////
//pulse sets the timing conditions for the debouncer, and output a debounced
pulse
//Inputs: clk_p-the 50MHz clock,
//    press-the raw signal from the pushbutton
//Outputs: pulse_en-enable signal to activate the second d-flipflop,
//          reset-a condition that resets the D-flipflops
module pulse(input logic clk_p, press,
            output logic pulse_en, reset);
      int count = 0;
always @(posedge clk_p, negedge press) begin
//count sets the timer for the bebouncer
      if (press ==0)
      count <= 0;
      else
      count <= (count >= 25000)? 0:count+1;
end
//enable flipflops after the count has increased
      assign pulse_en = (count == 25000)? 1'b1:1'b0;
//after the putton is released reset the flipflops
always @(posedge clk_p) begin
      if ((count == 0) && (press ==0))
      reset = 0;
      else
      reset=1;
end
endmodule
//////////////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////////////
//d_ff creates the filp-flops used in the debouncer
//Inputs: clk_ff-the 50MHz clock,
//      pulse_en-enable signal to activate the second d-flipflop,
//          reset-a condition that resets the D-flipflops
//Outputs: Q-the output of the flip-flop
module d_ff(input logic clk_ff, pulse_en,D,reset,
             output logic Q=0);
      always @(posedge clk_ff) begin
```

```
        if (pulse_en == 1)
              Q <= D;
        if (reset == 0)
              Q <= 0;
        end
endmodule
/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
```



PWM Generation Modules

```
/////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
//Created by Jacob Lagasse and Tyson Whyte
//This file creates the PWM outputs to switch the IGBTs
/////////////////////////////////////////////////////////////////////////////
module PWMgen
   (
      //output signals
      output logic PWM_pos_out = 1,
      output logic PWM_neg_out = 1,
      //freq input from speed button module comment out for sim
      input logic  [31:0] freq_in,
       //stop and start buttons
      input logic startstop_in,
       input logic CLOCK_50 ) ;

//**//**//**//**//**//**//*****************************************************
//These arrays control the PWM outputs.
//The arrays are 10,000 elements long so they are not included here.
//Their contents cant be generated in the provided matlab code.
   logic [1:0] twentyfive_array [0:9999] = '{  };
   logic [1:0] thirty_array [0:9999] = '{  };

   // internal logic registers
      logic PWM_pos_out_next = 1; //next pos pwm value
      logic PWM_neg_out_next = 1 ; //next neg pwm value
      logic [1:0] next_freq [0:9999] = '{default:0};  //next freq look up
table
      logic [1:0] current_freq [0:9999] = '{default:0};  //current freq look
up table
      logic pwm_clk = 0; //50khz clk for pwm
```

```
        logic [31:0]freq_step = 0; //counter to step through LUT
        logic [31:0] counter_PWM = 0; //counter to generate pwm_clk
        //logic start_mode = 0;

        //comment out later ////////////
        //logic  [31:0] freq_in = 30;
        //logic startstop_in =1;
        /////////////////////////////////

        always_comb begin
        //latch values to ensure always holding a value
        PWM_pos_out_next = PWM_pos_out;
        PWM_neg_out_next = PWM_neg_out;
        next_freq = current_freq;

// set freq array to step through by reading pushbutton module output (ie
//set motor speed)
        if(startstop_in == 1) begin // start mode must have been pushed
              if(freq_in == 1) begin //freq 1 = 5Hz
              next_freq = five_array;
              end

              else if(freq_in == 2) begin //freq 2 = 10Hz
              next_freq = ten_array;
              end

              else if(freq_in == 3) begin //freq 3 = 15Hz
              next_freq = fifteen_array;
              end

              else if(freq_in == 4) begin //freq 4 = 20Hz
              next_freq = twenty_array;
              end

              else if(freq_in == 5) begin //freq 5 = 25Hz
              next_freq = twentyfive_array;
              end

              else if(freq_in == 6) begin //freq 6 = 30Hz
              next_freq = thirty_array;
              end

              else next_freq = '{default:0}; //else freq is 0
        end
        else next_freq = '{default:0}; //check for stop button (startstop_in
//==0)

        //set next pwm outputs
        if(current_freq[freq_step] == 1)begin
          PWM_pos_out_next = 0; //0 on
          PWM_neg_out_next = 1; //1 off
```

```
      end

      else if(current_freq[freq_step] == 3)begin
        PWM_pos_out_next = 1; //1 off
        PWM_neg_out_next = 0; //0 on
      end

      else if(current_freq[freq_step] == 0)begin
        PWM_pos_out_next = 1; //1 off
        PWM_neg_out_next = 1; //1 off
      end

      else begin
        PWM_pos_out_next = 1; //1 off
        PWM_neg_out_next = 1; //1 off
      end
      end

//frequency stepper for LUT position (one window of 200ms)
 always_ff @(posedge pwm_clk) begin
      freq_step <= freq_step + 1;

       if(freq_step>=9999) begin
      freq_step <= 0;
            //set freq to next freq array
            current_freq <= next_freq;
      end
 end

//pwm clock generator (50khz)
 always_ff @(posedge CLOCK_50) begin
         counter_PWM <= counter_PWM + 1;
   if(counter_PWM >= 499) begin
       counter_PWM <= 0;
       pwm_clk <= ~pwm_clk; //50kHz clock for pwm
   end
 end

//set PWM output pins
      always_ff @(posedge CLOCK_50) begin

      //set PWM OUTPUT pins
      PWM_pos_out <= PWM_pos_out_next;
      PWM_neg_out <= PWM_neg_out_next;
      end
endmodule
///////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
```

DE0 Nano Pin Assignments

```
set_location_assignment PIN_R8 -to CLOCK_50
set_location_assignment PIN_A15 -to LED[0]
set_location_assignment PIN_A13 -to LED[1]
set_location_assignment PIN_B13 -to LED[2]
set_location_assignment PIN_A11 -to LED[3]
set_location_assignment PIN_D1 -to LED[4]
set_location_assignment PIN_F3 -to LED[5]
set_location_assignment PIN_B1 -to LED[6]
set_location_assignment PIN_L3 -to LED[7]
set_location_assignment PIN_J15 -to KEY[0]
set_location_assignment PIN_E1 -to KEY[1]
set_location_assignment PIN_M1 -to SW[0]
set_location_assignment PIN_T8 -to SW[1]
set_location_assignment PIN_B9 -to SW[2]
set_location_assignment PIN_M15 -to SW[3]
set_location_assignment PIN_P2 -to DRAM_ADDR[0]
set_location_assignment PIN_N5 -to DRAM_ADDR[1]
set_location_assignment PIN_N6 -to DRAM_ADDR[2]
set_location_assignment PIN_M8 -to DRAM_ADDR[3]
set_location_assignment PIN_P8 -to DRAM_ADDR[4]
set_location_assignment PIN_T7 -to DRAM_ADDR[5]
set_location_assignment PIN_N8 -to DRAM_ADDR[6]
set_location_assignment PIN_T6 -to DRAM_ADDR[7]
set_location_assignment PIN_R1 -to DRAM_ADDR[8]
set_location_assignment PIN_P1 -to DRAM_ADDR[9]
set_location_assignment PIN_N2 -to DRAM_ADDR[10]
set_location_assignment PIN_N1 -to DRAM_ADDR[11]
set_location_assignment PIN_L4 -to DRAM_ADDR[12]
set_location_assignment PIN_M7 -to DRAM_BA[0]
set_location_assignment PIN_M6 -to DRAM_BA[1]
set_location_assignment PIN_L7 -to DRAM_CKE
set_location_assignment PIN_R4 -to DRAM_CLK
set_location_assignment PIN_P6 -to DRAM_CS_N
set_location_assignment PIN_G2 -to DRAM_DQ[0]
set_location_assignment PIN_G1 -to DRAM_DQ[1]
set_location_assignment PIN_L8 -to DRAM_DQ[2]
set_location_assignment PIN_K5 -to DRAM_DQ[3]
set_location_assignment PIN_K2 -to DRAM_DQ[4]
set_location_assignment PIN_J2 -to DRAM_DQ[5]
set_location_assignment PIN_J1 -to DRAM_DQ[6]
set_location_assignment PIN_R7 -to DRAM_DQ[7]
set_location_assignment PIN_T4 -to DRAM_DQ[8]
set_location_assignment PIN_T2 -to DRAM_DQ[9]
set_location_assignment PIN_T3 -to DRAM_DQ[10]
set_location_assignment PIN_R3 -to DRAM_DQ[11]
set_location_assignment PIN_R5 -to DRAM_DQ[12]
set_location_assignment PIN_P3 -to DRAM_DQ[13]
set_location_assignment PIN_N3 -to DRAM_DQ[14]
set_location_assignment PIN_K1 -to DRAM_DQ[15]
set_location_assignment PIN_R6 -to DRAM_DQM[0]
set_location_assignment PIN_T5 -to DRAM_DQM[1]
set_location_assignment PIN_L1 -to DRAM_CAS_N
set_location_assignment PIN_L2 -to DRAM_RAS_N
set_location_assignment PIN_C2 -to DRAM_WE_N
set_location_assignment PIN_F2 -to I2C_SCLK
set_location_assignment PIN_F1 -to I2C_SDAT
set_location_assignment PIN_G5 -to G_SENSOR_CS_N
set_location_assignment PIN_M2 -to G_SENSOR_INT
set_location_assignment PIN_A14 -to GPIO_2[0]
set_location_assignment PIN_B16 -to GPIO_2[1]
set_location_assignment PIN_C14 -to GPIO_2[2]
set_location_assignment PIN_C16 -to GPIO_2[3]
set_location_assignment PIN_C15 -to GPIO_2[4]
set_location_assignment PIN_D16 -to GPIO_2[5]
set_location_assignment PIN_D15 -to GPIO_2[6]
set_location_assignment PIN_D14 -to GPIO_2[7]
set_location_assignment PIN_F15 -to GPIO_2[8]
set_location_assignment PIN_F16 -to GPIO_2[9]
set_location_assignment PIN_F14 -to GPIO_2[10]
set_location_assignment PIN_G16 -to GPIO_2[11]
set_location_assignment PIN_G15 -to GPIO_2[12]
set_location_assignment PIN_E15 -to GPIO_2_IN[0]
set_location_assignment PIN_E16 -to GPIO_2_IN[1]
set_location_assignment PIN_M16 -to GPIO_2_IN[2]
set_location_assignment PIN_A8 -to GPIO_0_IN[0]
set_location_assignment PIN_D3 -to GPIO_0[0]
set_location_assignment PIN_B8 -to GPIO_0_IN[1]
set_location_assignment PIN_C3 -to GPIO_0[1]
set_location_assignment PIN_A2 -to GPIO_0[2]
set_location_assignment PIN_A3 -to GPIO_0[3]
set_location_assignment PIN_B3 -to GPIO_0[4]
set_location_assignment PIN_B4 -to GPIO_0[5]
set_location_assignment PIN_A4 -to GPIO_0[6]
set_location_assignment PIN_B5 -to GPIO_0[7]
set_location_assignment PIN_A5 -to GPIO_0[8]
set_location_assignment PIN_D5 -to GPIO_0[9]
set_location_assignment PIN_B6 -to GPIO_0[10]
set_location_assignment PIN_A6 -to GPIO_0[11]
set_location_assignment PIN_B7 -to GPIO_0[12]
set_location_assignment PIN_D6 -to GPIO_0[13]
set_location_assignment PIN_A7 -to GPIO_0[14]
set_location_assignment PIN_C6 -to GPIO_0[15]
set_location_assignment PIN_C8 -to GPIO_0[16]
set_location_assignment PIN_E6 -to GPIO_0[17]
set_location_assignment PIN_E7 -to GPIO_0[18]
set_location_assignment PIN_D8 -to GPIO_0[19]
set_location_assignment PIN_E8 -to GPIO_0[20]
set_location_assignment PIN_F8 -to GPIO_0[21]
set_location_assignment PIN_F9 -to GPIO_0[22]
set_location_assignment PIN_E9 -to GPIO_0[23]
set_location_assignment PIN_C9 -to GPIO_0[24]
set_location_assignment PIN_D9 -to GPIO_0[25]
set_location_assignment PIN_E11 -to GPIO_0[26]
set_location_assignment PIN_E10 -to GPIO_0[27]
set_location_assignment PIN_C11 -to GPIO_0[28]
set_location_assignment PIN_B11 -to GPIO_0[29]
set_location_assignment PIN_A12 -to GPIO_0[30]
set_location_assignment PIN_D11 -to GPIO_0[31]
set_location_assignment PIN_D12 -to GPIO_0[32]
set_location_assignment PIN_B12 -to GPIO_0[33]
set_location_assignment PIN_T9 -to GPIO_1_IN[0]
set_location_assignment PIN_F13 -to GPIO_1[0]
set_location_assignment PIN_R9 -to GPIO_1_IN[1]
set_location_assignment PIN_T15 -to GPIO_1[1]
set_location_assignment PIN_T14 -to GPIO_1[2]
set_location_assignment PIN_T13 -to GPIO_1[3]
set_location_assignment PIN_R13 -to GPIO_1[4]
set_location_assignment PIN_T12 -to GPIO_1[5]
set_location_assignment PIN_R12 -to GPIO_1[6]
set_location_assignment PIN_T11 -to GPIO_1[7]
set_location_assignment PIN_T10 -to GPIO_1[8]
set_location_assignment PIN_R11 -to GPIO_1[9]
set_location_assignment PIN_P11 -to GPIO_1[10]
set_location_assignment PIN_R10 -to GPIO_1[11]
set_location_assignment PIN_N12 -to GPIO_1[12]
set_location_assignment PIN_P9 -to GPIO_1[13]
set_location_assignment PIN_N9 -to GPIO_1[14]
set_location_assignment PIN_N11 -to GPIO_1[15]
set_location_assignment PIN_L16 -to GPIO_1[16]
set_location_assignment PIN_K16 -to GPIO_1[17]
set_location_assignment PIN_R16 -to GPIO_1[18]
set_location_assignment PIN_L15 -to GPIO_1[19]
set_location_assignment PIN_P15 -to GPIO_1[20]
set_location_assignment PIN_P16 -to GPIO_1[21]
set_location_assignment PIN_R14 -to GPIO_1[22]
set_location_assignment PIN_N16 -to GPIO_1[23]
set_location_assignment PIN_N15 -to GPIO_1[24]
set_location_assignment PIN_P14 -to GPIO_1[25]
set_location_assignment PIN_L14 -to GPIO_1[26]
set_location_assignment PIN_N14 -to GPIO_1[27]
set_location_assignment PIN_M10 -to PWM_pos_out
set_location_assignment PIN_L13 -to PWM_neg_out
set_location_assignment PIN_J16 -to button_d
set_location_assignment PIN_K15 -to button_u
set_location_assignment PIN_J13 -to RUN
set_location_assignment PIN_J14 -to button

set_location_assignment PIN_A2 -to qspb
set_location_assignment PIN_A8 -to qsa
set_location_assignment PIN_B8 -to qsb
set_location_assignment PIN_A12 -to ct[0]
set_location_assignment PIN_A5 -to leds[0]
set_location_assignment PIN_C11 -to ct[1]
set_location_assignment PIN_B6 -to leds[1]
set_location_assignment PIN_E11 -to ct[2]
set_location_assignment PIN_B7 -to leds[2]
set_location_assignment PIN_C9 -to ct[3]
set_location_assignment PIN_A7 -to leds[3]
set_location_assignment PIN_C8 -to leds[4]
set_location_assignment PIN_E7 -to leds[5]
set_location_assignment PIN_E8 -to leds[6]
set_location_assignment PIN_F9 -to leds[7]
set_location_assignment PIN_D5 -to kpc[3]
set_location_assignment PIN_A6 -to kpc[2]
set_location_assignment PIN_D6 -to kpc[1]
set_location_assignment PIN_C6 -to kpc[0]
set_location_assignment PIN_E9 -to kpr[0]
set_location_assignment PIN_F8 -to kpr[1]
set_location_assignment PIN_D8 -to kpr[2]
set_location_assignment PIN_E6 -to kpr[3]

set_location_assignment PIN_D9 -to rgb_din
set_location_assignment PIN_E10 -to rgb_clk
set_location_assignment PIN_B11 -to rgb_cs
set_location_assignment PIN_D11 -to rgb_dc
set_location_assignment PIN_B12 -to rgb_res

set_location_assignment PIN_G15 -to jstk_sel
set_location_assignment PIN_A10 -to ADC_CS_N
set_location_assignment PIN_B10 -to ADC_SADDR
set_location_assignment PIN_A9 -to ADC_SDAT
set_location_assignment PIN_B14 -to ADC_SCLK
set_location_assignment PIN_B3 -to spkr
set_location_assignment PIN_D12 -to point

set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to jstk_sel
```

Appendix B

Parts List

FPGA:
Intel DE0 Nano field programmable gate-array, Supplied

Motor:
Single phase, 115V, ¼ hp , 4 amperes full load (FLA), Supplied

Transformer:
Variable autotransformer, 10 ampere rating,Supplied

Overcurrent Protection:
10A circuit breaker, Supplied

Opto-Isolators:
Toshiba Semiconductor and Storage TLPN137(F), $1.88, Quantity 4
https://www.digikey.ca/product-detail/en/toshiba-semiconductor-and-storage/TLPN137-F/TLPN137-F
ND/7595004

Switching:
IGBT IC, $14.73
https://www.digikey.ca/product-detail/en/infineon-technologies/IGCM10F60GAXKMA1/IGCM10F60GA
XKMA1-ND/5960108

Rectification:
Bridge Rectifier, $2.37
https://www.digikey.ca/product-detail/en/diodes-incorporated/GBJ2004-F/GBJ2004-FDI-ND/815143

Filtering:
1mH Inductor, 10A, $24.88
https://www.digikey.ca/product-detail/en/hammond-manufacturing/157D/HM1524-ND/455024

6800uF capacitor, 250V, $38.16
https://www.digikey.ca/product-detail/en/cornell-dubilier-electronics-cde/382LX682M250B102VS/338-
1996-ND/2256604

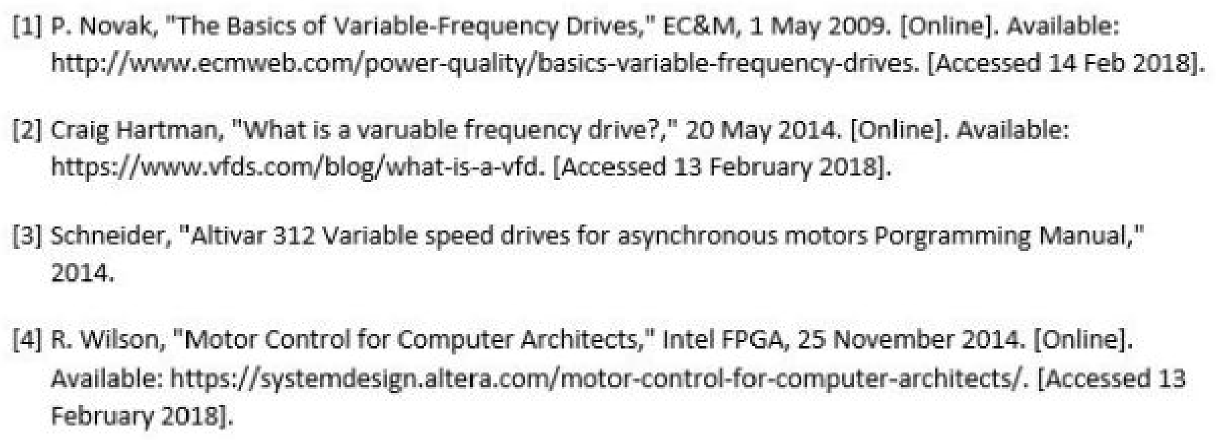
180uF capacitor, 400V, supplied

Push-Button:
SPDT PB x 3, Supplied

Display:
4 digit, 7 segment LCD Display, Supplied

Misc:
Connectors and wiring, Supplied

References



[1] P. Novak, "The Basics of Variable-Frequency Drives," EC&M, 1 May 2009. [Online]. Available: http://www.ecmweb.com/power-quality/basics-variable-frequency-drives. [Accessed 14 Feb 2018].

[2] Craig Hartman, "What is a varuable frequency drive?," 20 May 2014. [Online]. Available: https://www.vfds.com/blog/what-is-a-vfd. [Accessed 13 February 2018].

[3] Schneider, "Altivar 312 Variable speed drives for asynchronous motors Porgramming Manual," 2014.

[4] R. Wilson, "Motor Control for Computer Architects," Intel FPGA, 25 November 2014. [Online]. Available: https://systemdesign.altera.com/motor-control-for-computer-architects/. [Accessed 13 February 2018].