Design and Simulation of Interconnected H-Bridge Inverter

By

Samuel Muehleck

Senior Project

COMPUTER ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

June, 2012

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ACKNOWLEDGEMENTS

Special thanks to Professor Taufik. He has a remarkable ability to make learning fun. It was a privilege to take his courses. I would not have completed this senior project without him.

ABSTRACT

In this project, the research, design and simulation of an interconnected H-bridge single-phase inverter was explored. The inverter design was divided into three separate sections. First, a VHDL programmed CPLD was used to generate control signals for the transistors of the inverter circuit. Second, power electronics was used to create an inverter circuit to change a DC input to a 60 Hz AC output. Lastly, additional circuits were used to link the generated control signals to the MOSFET transistors of the inverter circuit. The operation of the three components was then validated with computer simulations. Results of computer simulation demonstrate the operation of the proposed single-phase inverter. Recommendations for building the hardware prototype of the proposed inverter will be presented in this report.

CHAPTER 1. INTRODUCTION

With the increasing use of renewable energy, particularly grid-tied photovoltaics, the need to have a good quality single-phase inverter has become important. The conventional technology used for single phase inverters typically consists of either square-wave or pwm inverters. The square wave type is the simplest method to produce AC from DC; however, it suffers from low frequency harmonics which causes difficulty in filtering out the noise to prevent these harmonics to return back to the primary side of the transformer. The pwm inverter, on the other hand, forces the harmonics to be way up higher than the fundamental (line) frequency; thus, easing up the filtering requirement of the inverter. However, the major drawback of the pwm inverter is the increased switching losses due to the frequent switching actions of the electronic switches within the inverter.

This project proposes an improved version of the square-wave inverter by increasing the number of steps per one period of the desired frequency with the use of additional electronic switches. With the conventional square wave inverter, there are only two pulses generated by the controller to mimic an AC output. This is in fact what is causing the low harmonic distortion of the inverter. To lessen these low harmonic distortions in square wave technique, the approach is then to increase the number of steps per period. This is the approach taken by the proposed inverter by adding the number of steps to seven; hence, the name seven-level H-bridge interconnected inverter. In essence, the seven-step AC output voltage generated by the inverter will push the low harmonic to above seven times the line frequency. For a 60-Hz AC system, this means the harmonic content of the inverter output will be at 420 Hz and above. This is a significant improvement over 180 Hz and above offered by the conventional square wave technology.

CHAPTER 2. BACKGROUND

As previously mentioned, the purpose of an inverter is to convert DC power to AC power. Inverters are an integral part of many technologies including uninterruptable power supplies, induction heating, high-voltage direct current power transmission, variable frequency drives, electric vehicle drives, and multiple renewable energy applications. All of these technologies use inverters to achieve different goals, but all produce AC power from a DC input.

There are many varieties of inverter designs. The most common topology uses what is referred to as the H-bridge topology. Its basic configuration is shown in Figure 2-1. This topology is used in conjunction with either the square wave, or pulse width modulation (PWM) switching schemes.

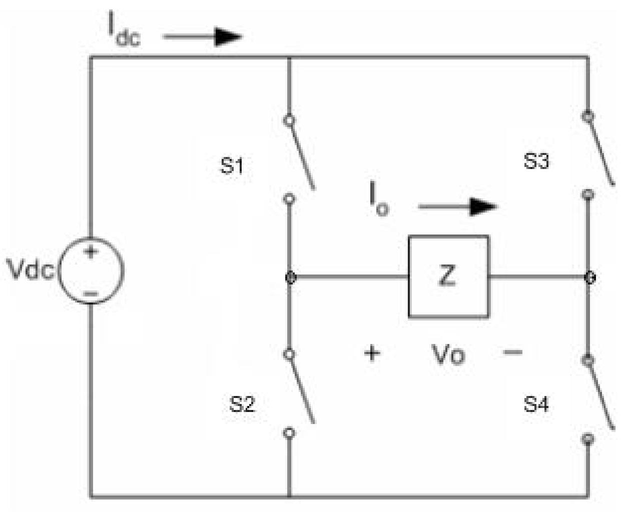


Figure 2-1. H-Bridge Single-Phase Inverter

The square-wave switching scheme is a method for controlling the switches (labeled S1 through S4) in order to achieve a square wave AC output signal. The AC output is achieved by using a control signal with a 50% duty cycle wired to S1 and S4. An inverted copy of the same signal is also wired to S2 and S3. This switching scheme ensures that S1 and S4 are always on when S2 and S3 are off. It should be easily seen how such a switching scheme creates the square wave output shown in Figure 2-2. The advantage of using an H-bridge inverter is that only a single, simple control signal is required to control four transistors. The disadvantage, however, is that the square-wave output is a low quality AC signal that injects many harmonics into any loads to which it is powering.

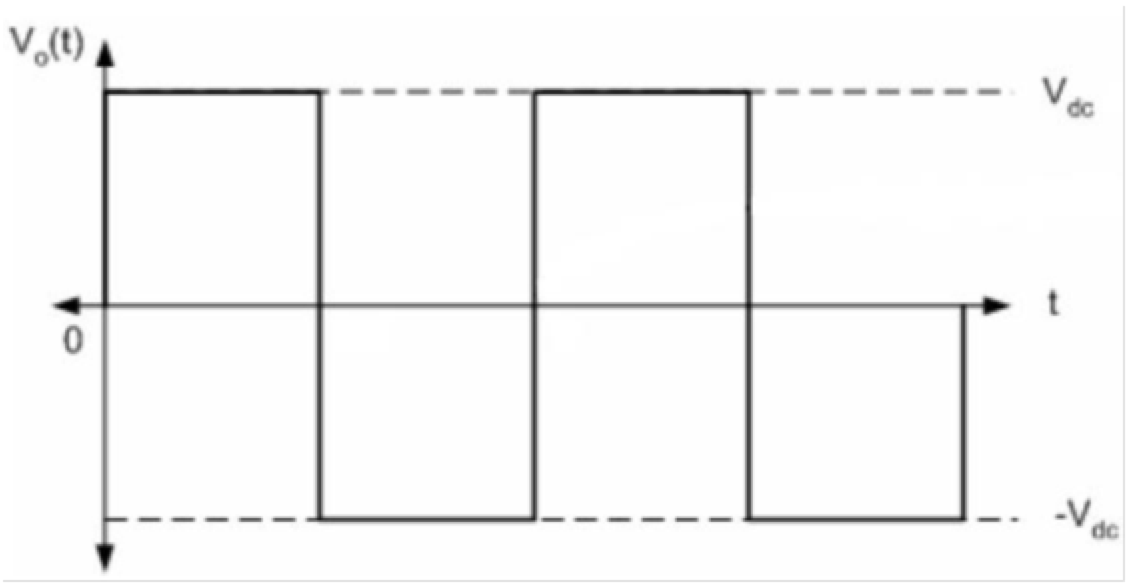


Figure 2-2. Square Wave Inverter Output

As mentioned previously, PWM control signals can be used with the same H-bridge topography. The disadvantage of the PWM switching scheme is that it is more complicated than the square-wave switching scheme. Multiple, relatively complex control signals are needed to control the transistors of the PWM inverter. The advantage, however, of the PWM switching scheme is that it is able to generate a more perfect sinusoidal AC output, which some loads prefer. An example PWM inverter output is shown in Figure 2-3.

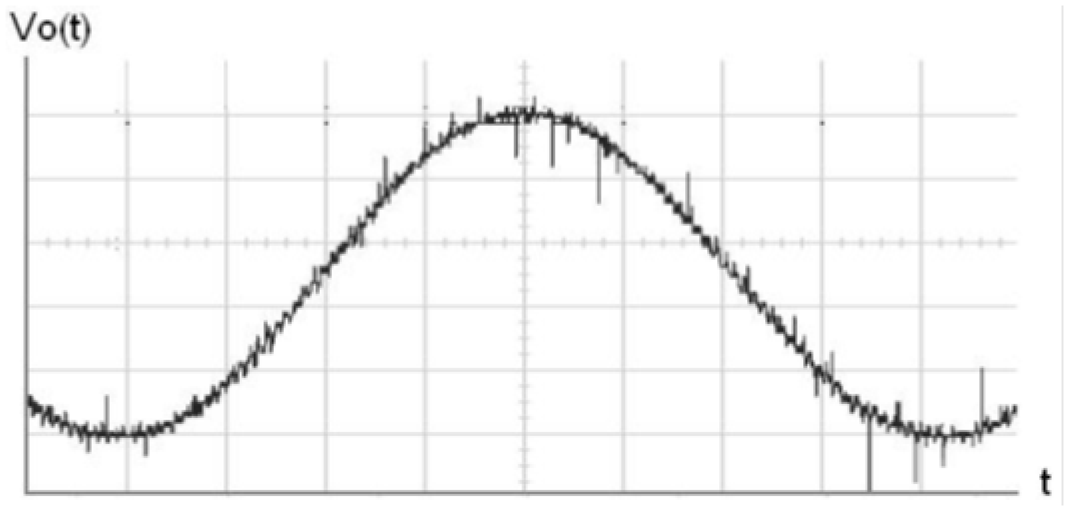


Figure 2-3. Typical Filtered PWM Inverter Output

This paper presents a proposed new single-phase interconnected H-bridge inverter (or multistep inverter). One advantage of a multistep inverter is that it provides a more sinusoidal output voltage than an inverter with a basic square-wave switching scheme. Another advantage is that its control signals are less complicated than that of the PWM inverter. However, the multistep inverter does not generate as high of quality of sinusoidal output voltage as that of the PWM switching scheme inverter. It does, though, provide an inverted signal of sufficient quality for most loads. Furthermore, the control signals required by the multistep inverter are relatively complicated compared to the single control signal of the square wave H-bridge inverter. In other words, the multistep inverter is a compromise between a complicated, but high quality PWM inverter, and a simple, but low quality square wave inverter. Figures 2-4 and 2-5 below show the circuit diagram and the ideal output voltage for the multistep inverter.

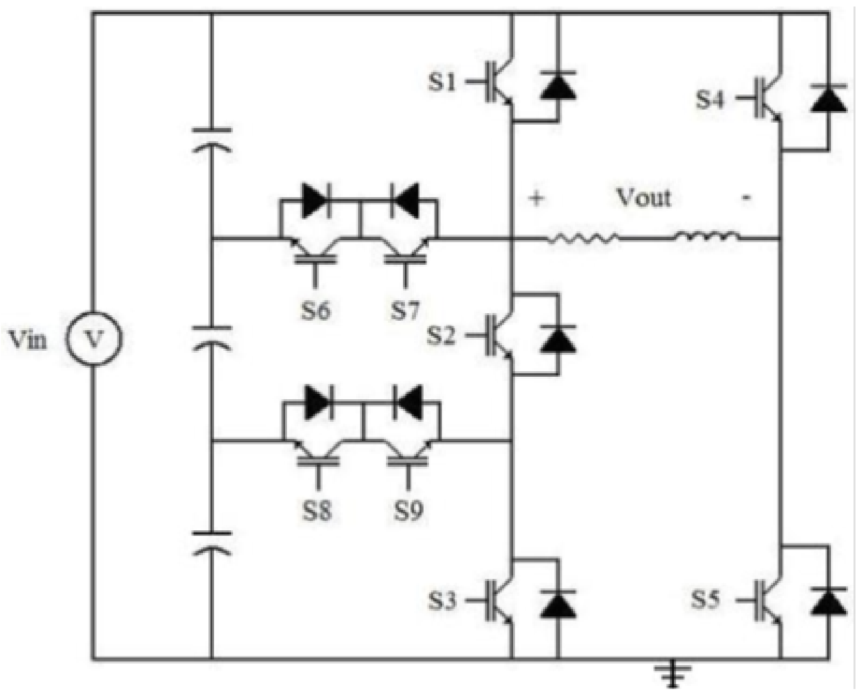


Figure 2-4. Power Stage of Multi-Step Inverter

The nine MOSFETs of the multistep inverter of Figure 2-4 are switched on and off in a controlled manner in order to control the flow of power through the circuit and to generate an inverted stair step output (Vout) from a DC input voltage (Vin), as shown in Figure 2-5.

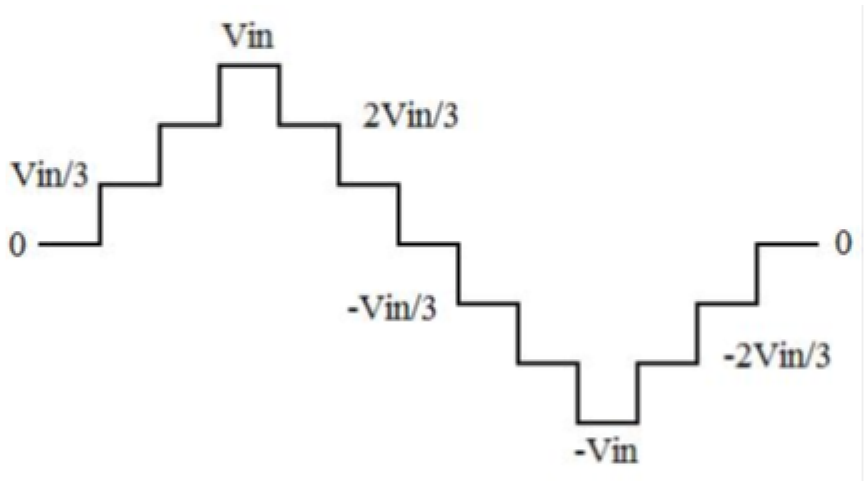


Figure 2-5. Multistep Single-Phase Inverter Output

CHAPTER 3. PSPICE SIMULATION

After studying the inverter topology and determining the control signals, the next step of the design was to simulate the interconnected H-bridge inverter circuit. The PSPICE schematic of the inverter circuit is shown in Figure 3-1. SBREAKS were used in order to simulate the switching characteristics of near ideal transistors. An input voltage (Vin) of 10 V was chosen arbitrarily for the simulation. The expected output will be an AC voltage with a peak-to-peak voltage of 20 Volts (10 V to -10 V)

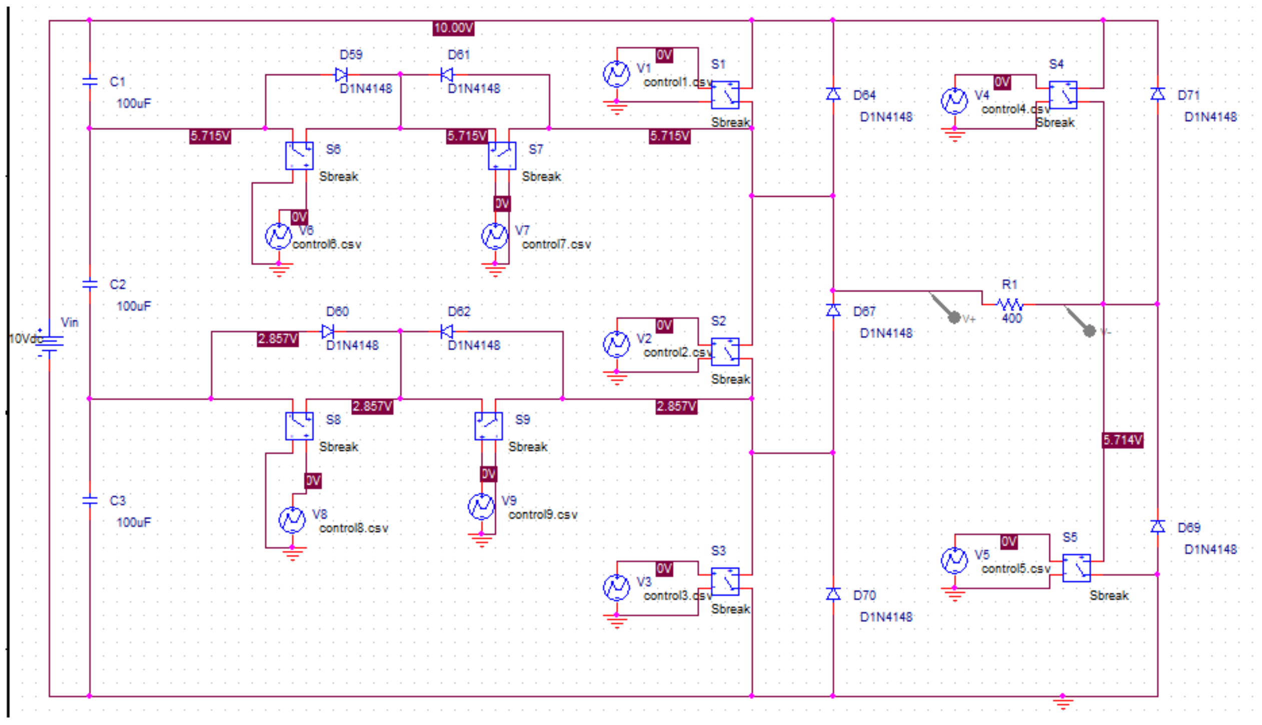


Figure 3-1. PSPICE Schematic

I decided to simulate the control signals using piece-wise linear voltage sources. By using piece-wise linear sources, one can precisely control exactly how a source behaves. The specific piece-wise linear voltage source used is called “VPWL_F_RE_FOREVER.” This voltage source creates its output based upon the input it receives from a csv file (comma separated value). This file includes a series of pairs of numbers denoting time and voltage value. This series of pairs define one period of the piece-wise linear source. The period defined by the csv file is then repeated indefinitely. In this case, every piece-wise linear source represents a separate control signal. Using piece-wise linear sources to implement the control signals is far easier to both implement and understand, than attempting to implement the control signals with vpulses in series, as was my initial implementation. A sample period of the nine control signals are shown in Figure 3-2.

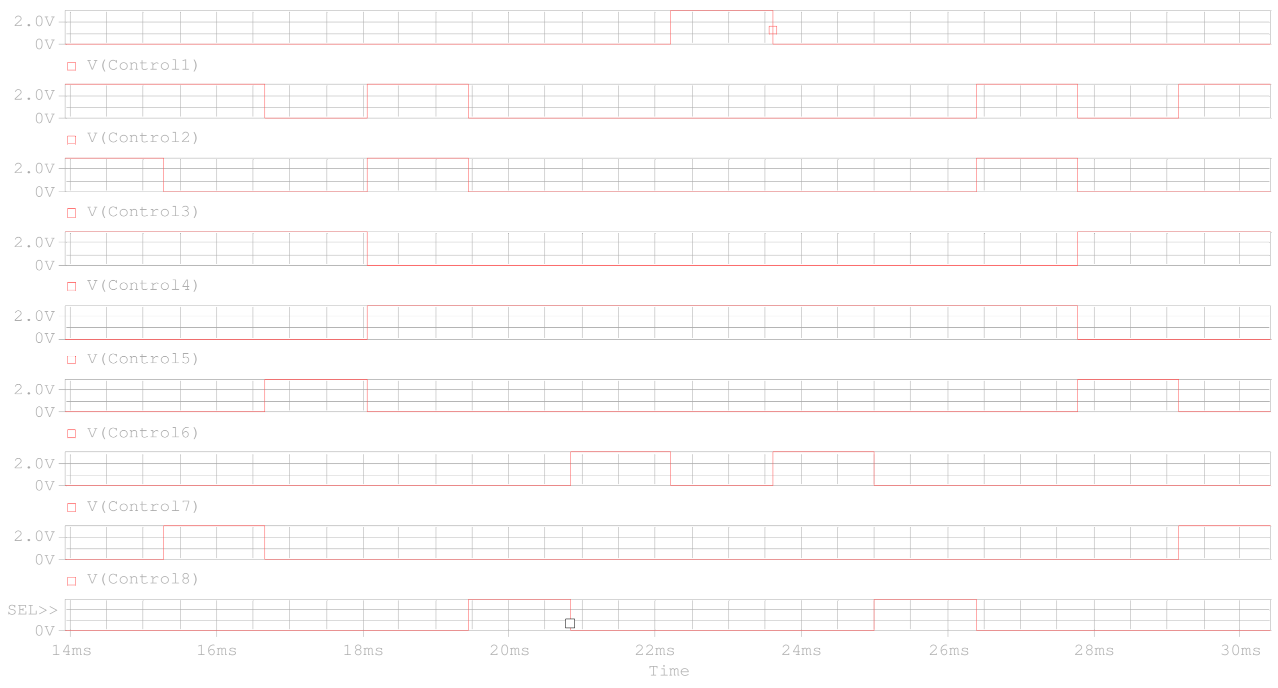
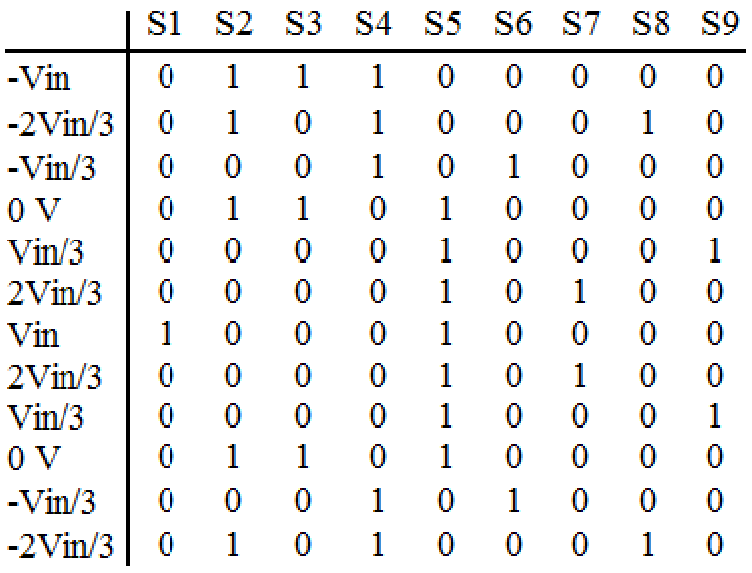


Figure 3-2. One Period of Control Signals

The above logic used for the above control signals was derived from performing circuit analysis on the inverter circuit. That analysis resulted in the table of control signals seen in Table I. In Table I a ‘1’ denotes a transistor that should be “on” in order to achieve given output voltages.



	S1	S2	S3	S4	S5	S6	S7	S8	S9
-Vin	0	1	1	1	0	0	0	0	0
-2Vin/3	0	1	0	1	0	0	0	1	0
-Vin/3	0	0	0	1	0	1	0	0	0
0 V	0	1	1	0	1	0	0	0	0
Vin/3	0	0	0	0	1	0	0	0	1
2Vin/3	0	0	0	0	1	0	1	0	0
Vin	1	0	0	0	1	0	0	0	0
2Vin/3	0	0	0	0	1	0	1	0	0
Vin/3	0	0	0	0	1	0	0	0	1
0 V	0	1	1	0	1	0	0	0	0
-Vin/3	0	0	0	1	0	1	0	0	0
-2Vin/3	0	1	0	1	0	0	0	1	0

Table I. One Period of Control Signals

The simulated seven-step inverter output is shown in Figure 3-3. The Fourier analysis of that output is shown in Figure 3-4.

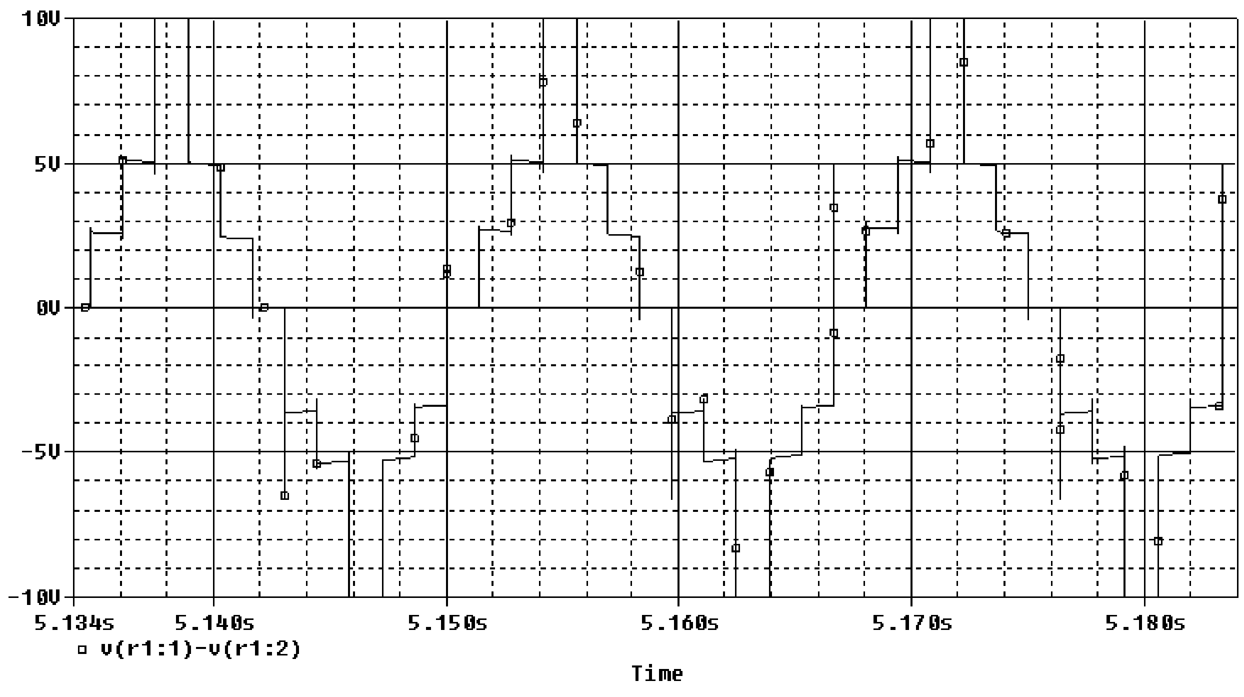


Figure 3-3. Simulated Seven-Step Inverter Output Voltage

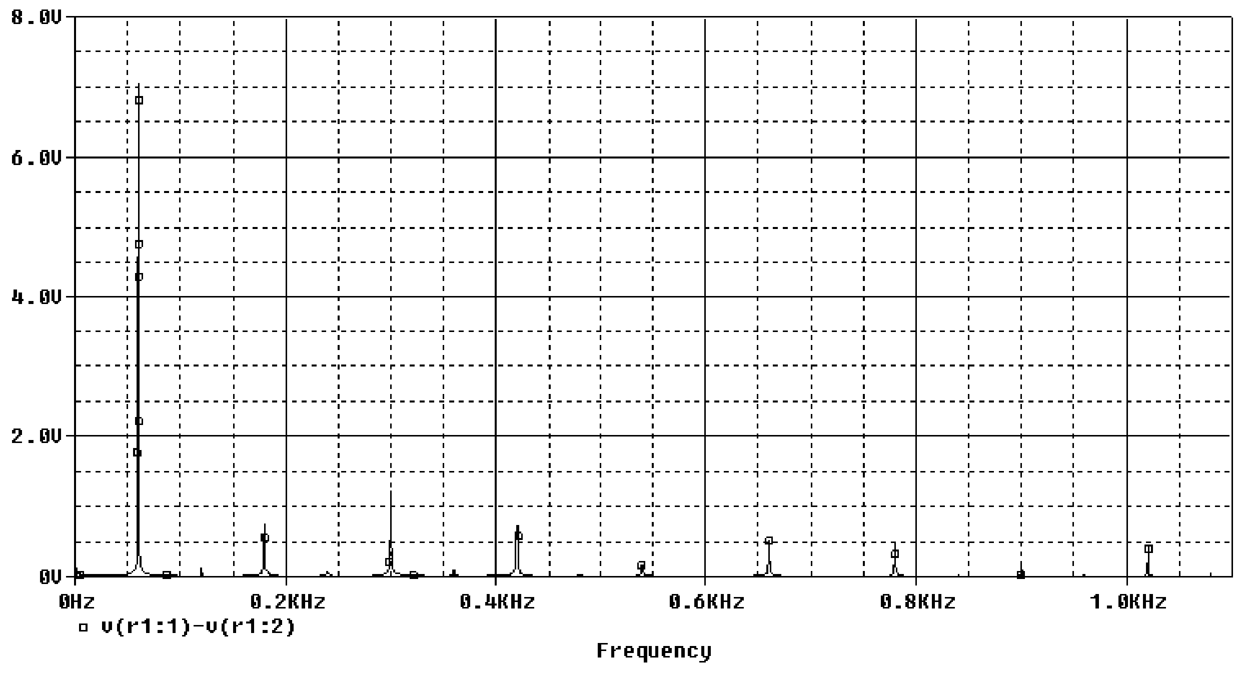


Figure 3-4. Simulated Seven-Step Inverter Fourier Analysis

I also simulated a simpler five-step inverter as well as a square wave inverter so that conclusions can be drawn regarding the seven-step inverter design. Those conclusions will be discussed later in this paper. Those simulation outputs are shown in figures 3-5 thru 3-8 below.

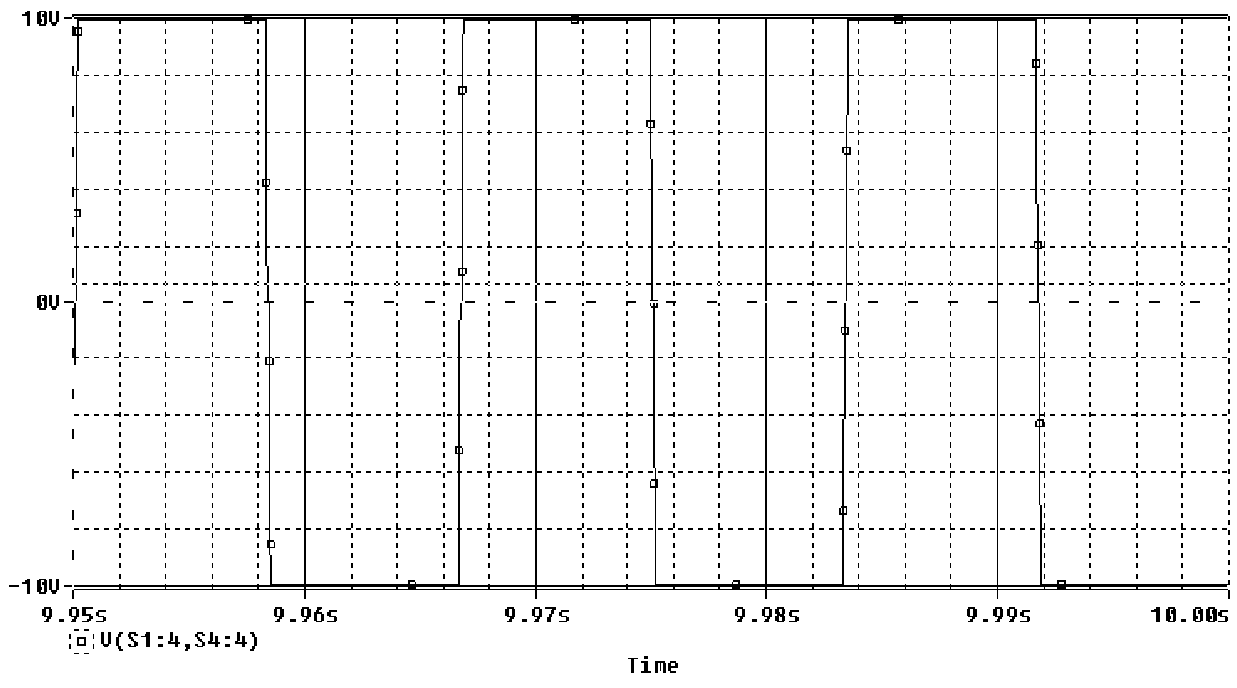


Figure 3-5. Simulated Square Wave Inverter Output Voltage

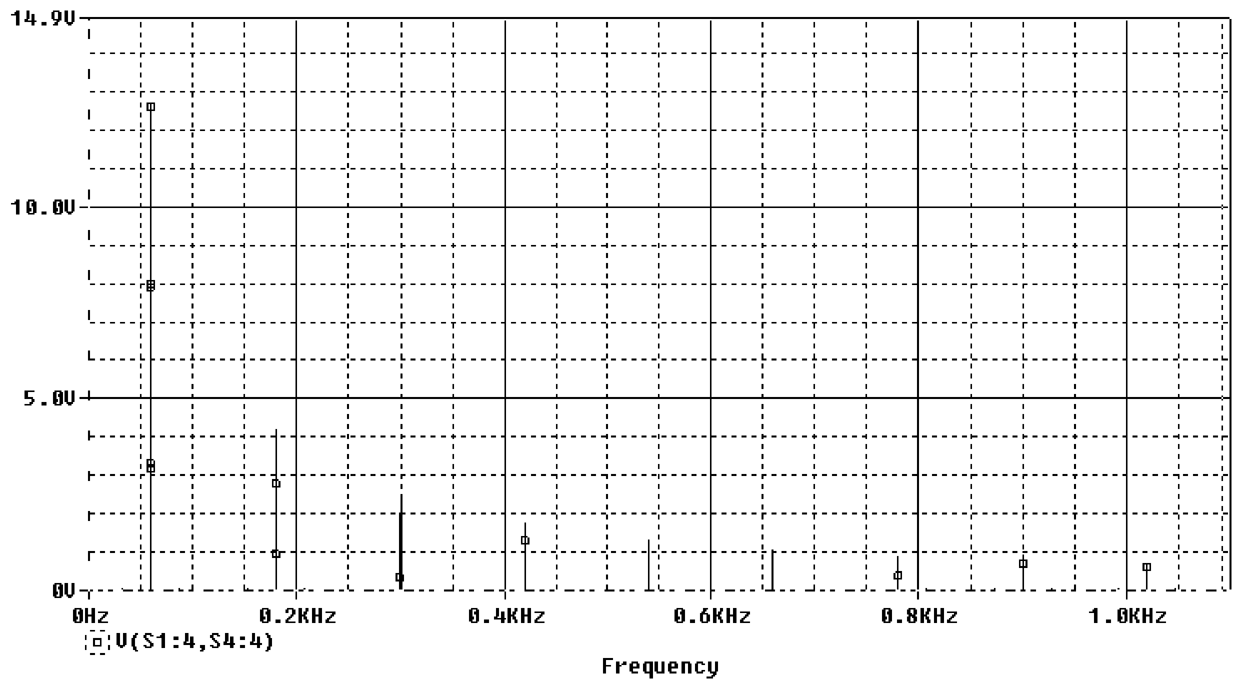


Figure 3-6. Simulated Square Wave Inverter Fourier Analysis

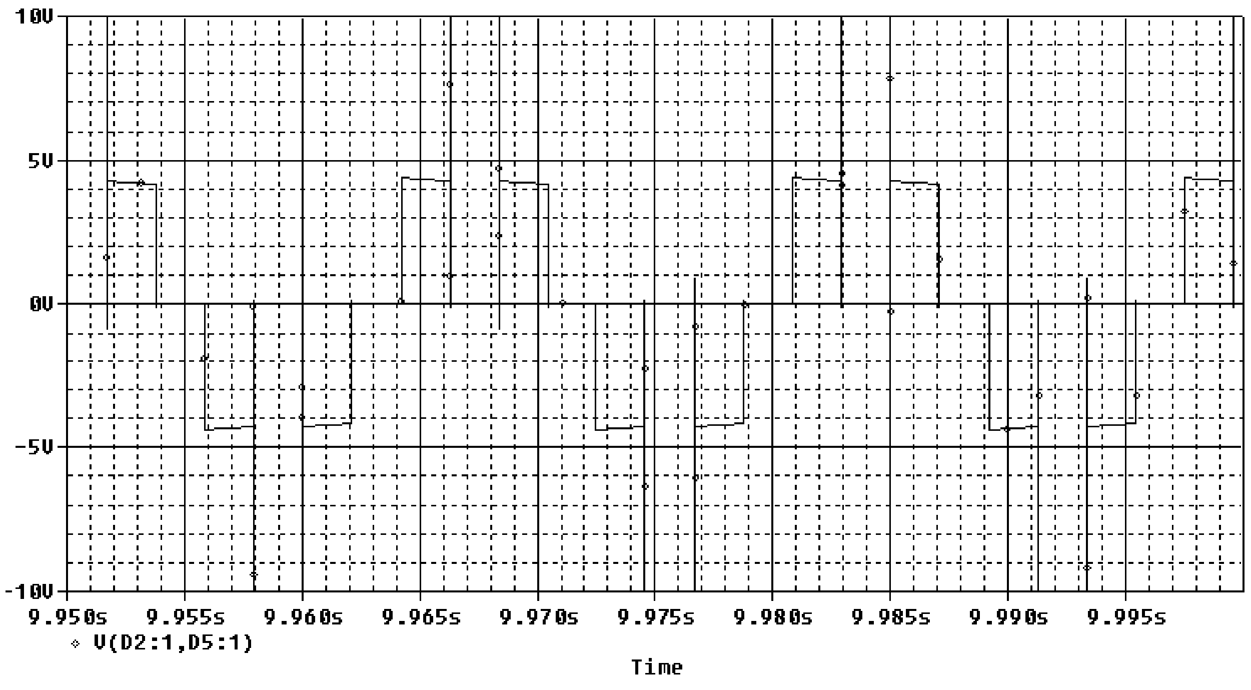


Figure 3-7. Simulated Five-Step Inverter Output Voltage

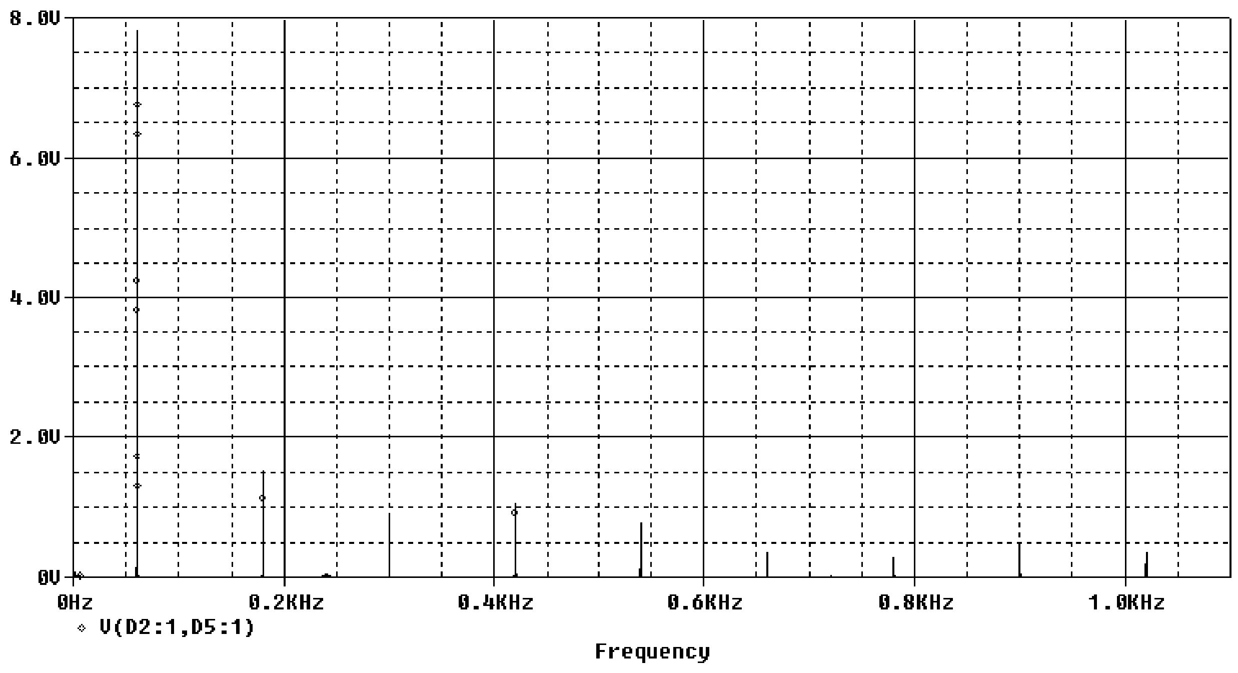


Figure 3-8. Simulated Five-Step Inverter Fourier Analysis

As a result of these simulations, it was decided that multi-step inverters are of sound design and that they will result in less lower harmonics than result from a square wave inverter. Those conclusions will be further discussed later in this paper.

CHAPTER 4. HARDWARE PROTOTYPE DESIGN RECOMMENDATIONS

After successfully simulating the inverter circuit using PSpice, the next step in the design was to choose a method of implementing those control signals. Drawing upon school experience, it was decided that the best way to generate these control signals was to design a finite state machine (FSM) which would be implemented using a complex programmable logic device (CPLD). In this case, a CPLD can be programmed to implement a FSM with 13 different states. One state for each of the output levels of one period of the inverter output waveform and a 13th state for dead time. The "dead time" state briefly sets all transistors to the "off" position. This is necessary to prevent undesired draining of the capacitors due to crossover, resulting from the non-ideal rise and fall times of the transistors. As the state machine changes from one state to the next, the CPLD changes its nine outputs to the different transistor switches. Figure 4-1 illustrates the state machine. In the figure below, switches with a value of '1' denote the transistors of the inverter that will be set to the "on" position. All other transistors would be set to "off."

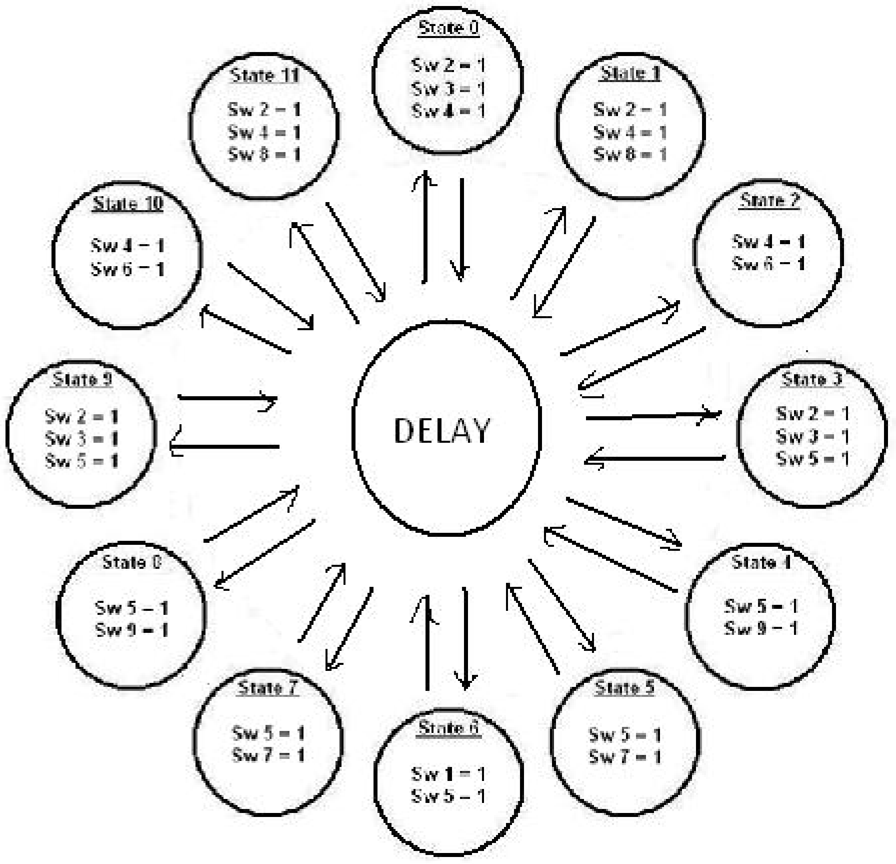
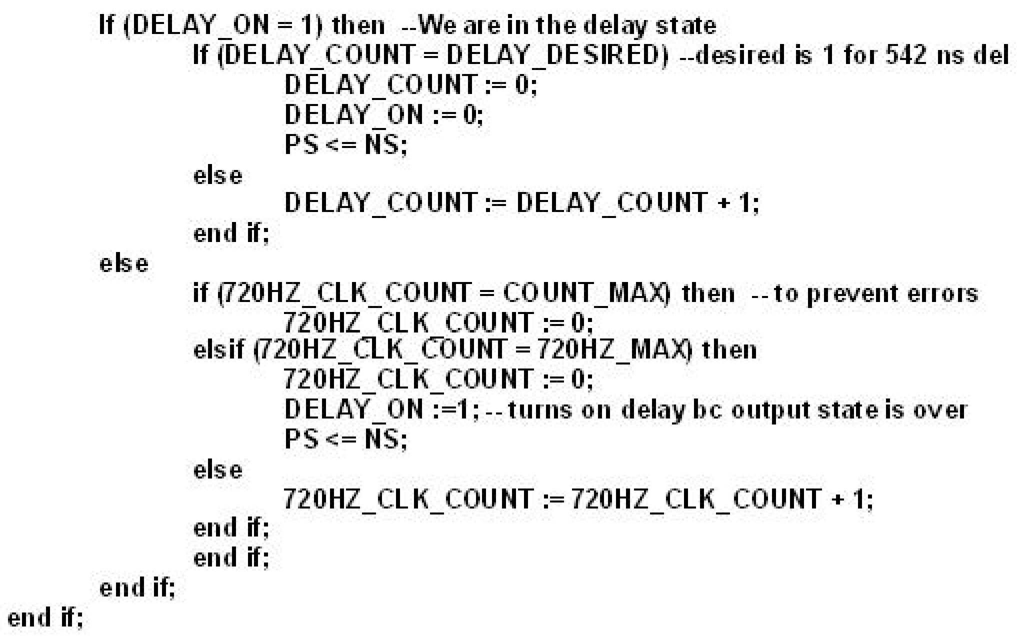


Figure 4-1. Control Signal FSM Diagram

The Digilent XC2-XL system board, which includes the CoolRunner 2 CPLD was chosen to control the MOSFETs of this inverter due to the author's familiarity with its use. Additionally, CoolRunner 2 CPLD's are good choice because they combine very low power with high speed, high density, and high I/O counts in a single device. In summary, the CoolRunner 2 was chosen for the prototype due to its ease of use, cost effectiveness, low power consumption and high-speed operation.

A dead time state lasting 542 ns was chosen for a few reasons. 542 ns is a relatively short period of time compared to the period of the 60 Hz output. Additionally, it is a sufficient length of time to prevent the unwanted crossover state between two transistors that occurs due to the non ideal rise and fall times of transistors. Lastly, 542 ns is the smallest dead time that can be implemented using the 1.8432 MHz clock included with the Digilent system board.

Implementing the dead time requires the creation of two clocks within the synchronous portion of the VHDL code. One clock for the dead time, and another clock to time the length of the 12 different output states. The code used to generate this timing is shown in Figure 4-2 below.



```
    If (DELAY_ON = 1) then  --We are in the delay state
        If (DELAY_COUNT = DELAY_DESIRED) --desired is 1 for 542 ns del
            DELAY_COUNT := 0;
            DELAY_ON := 0;
            PS <= NS;
        else
            DELAY_COUNT := DELAY_COUNT + 1;
        end if;
    else
        if (720HZ_CLK_COUNT = COUNT_MAX) then  --to prevent errors
            720HZ_CLK_COUNT := 0;
        elsif (720HZ_CLK_COUNT = 720HZ_MAX) then
            720HZ_CLK_COUNT := 0;
            DELAY_ON :=1; --turns on delay bc output state is over
            PS <= NS;
        else
            720HZ_CLK_COUNT := 720HZ_CLK_COUNT + 1;
        end if;
        end if;
    end if;
end if;
```

Figure 4-2. VHDL Code to Implement Delay and Output State Clocks

The logic design for the hardware prototype uses the Coolrunner 2 CPLD to generate two different clock signals, one for the approximately 1.389 ms output states (1/12 of the AC cycle), and one for the 542 ns dead time state. The Coolrunner 2 also implements the state machine as well as the logic that determines the next state the machine should transition to. The CPLD also outputs the control signals to the amplification portion of the design through header B shown in the CPLD overview of Figure 4-3. All of the prototypical code for the CPLD state machine implementation can be found in the appendices at the end of this paper.

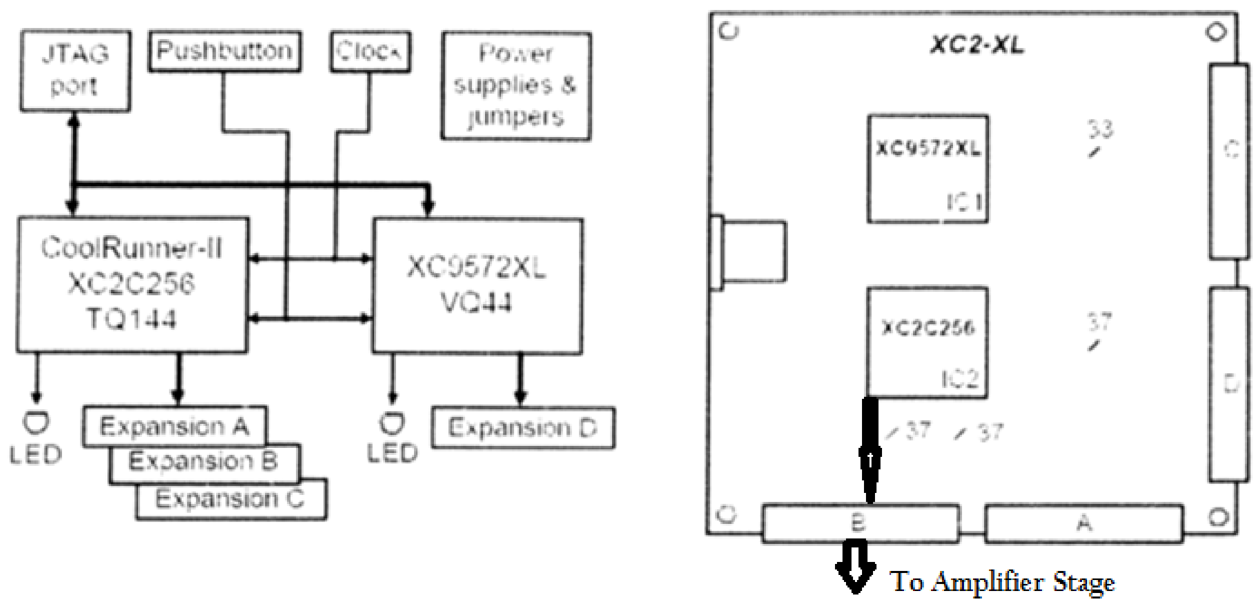


Figure 4-3. High Level Overview of the Digilent Board
Source: Digilent. 2004. *Digilent XC2-XL System Board Reference Manual.*

Amplification Circuit- As previously mentioned, the CoolRunner 2 CPLD simultaneously outputs the nine control signals to the amplification stage of the prototype design through header B shown above. An amplification stage is necessary because the transistors of the inverter circuit require gate voltages higher than that which is outputted by the Digilent board. In order to provide this gain in the control signals, TL081 operational amplifiers can be used in a non-inverting amplifier configuration.

Power Inverter Circuit Prototype- It was necessary to select capacitors, diodes and transistors for the inverter design. MOSFETs were chosen for a few reasons. First, MOSFETs contain an anti-parallel schottky diode which will eliminate the need for the purchase of additional diodes for this prototype. Second, MOSFETs have fast switching characteristics and high efficiency. Finally, MOSFETs are turned on by a voltage source rather than a current source like BJTs. Since the Coolrunner 2 generated control signals are voltage sources, it was appropriate to use MOSFETs. The IRFZ34N N-Channel MOSFET was chosen for the prototype design. It is thru-hole rather than surface mount, which will make building a prototype much simpler. It has a maximum current capacity of 29 A and a maximum Vds of 55 V, which far exceeds what's necessary for this prototype at a minimal cost of $1.20 apiece. It has a very low $R_{ds(on)}$ of .040 ohms which will lead to low resistive losses and higher efficiency. Also, the IRFZ34N MOSFET has a rise time of 49 ns and a fall time of 40 ns, both of which are very low and more than sufficient for this prototype.

For the purpose of the prototype, 4.7 micro Farad capacitors with a maximum voltage of 25 Volts were chosen. These were chosen because I had a surplus of them available so they could be used at no cost. By putting two of these capacitors in parallel, a capacitor equivalence of 9.4 micro Farads is achieved. A capacitor of this size will provide a sufficient hold-up time for this prototype. Lastly, 25 Volts is more than adequate for this prototype since the most voltage that should ever be across a capacitor is 3.34 Volts for a 10 Volt DC input.

CHAPTER 5. TESTING

As mentioned earlier in this paper, the goal of this senior project was to simulate a seven-step inverter and draw conclusions on whether it produces less lower harmonics than a square wave inverter. For that reason, I simulated three different inverters, a square wave inverter, a five-step inverter and a seven-step inverter. Table II shows the amplitudes of the lower harmonics for each of those inverters.

Frequency (Hz)	Sq Wave (V)	5 Step (V)	7 Step (V)
60	12.65	7.81	7.0437
120	0	0	0.116
180	4.22	1.54	0.769
240	0	0.04	0.059
300	2.52	0.91	1.23
360	0	0	0.116
420	1.78	1.07	0.746
480	0	0	0.022
540	1.35	0.793	0.186
600	0	0	0.015
660	1.09	0.373	0.536
720	0	0	0.013
780	0.913	0.303	0.513
840	0	0	0.023
900	0.783	0.42	0.163
960	0	0	0.015
1020	0.652	0.35	0.42

Table II. Fourier Analysis Comparison

For ease of comparison, Figure 5-1 is a graph comparing the Fourier analysis of each of the three simulated inverter circuits.

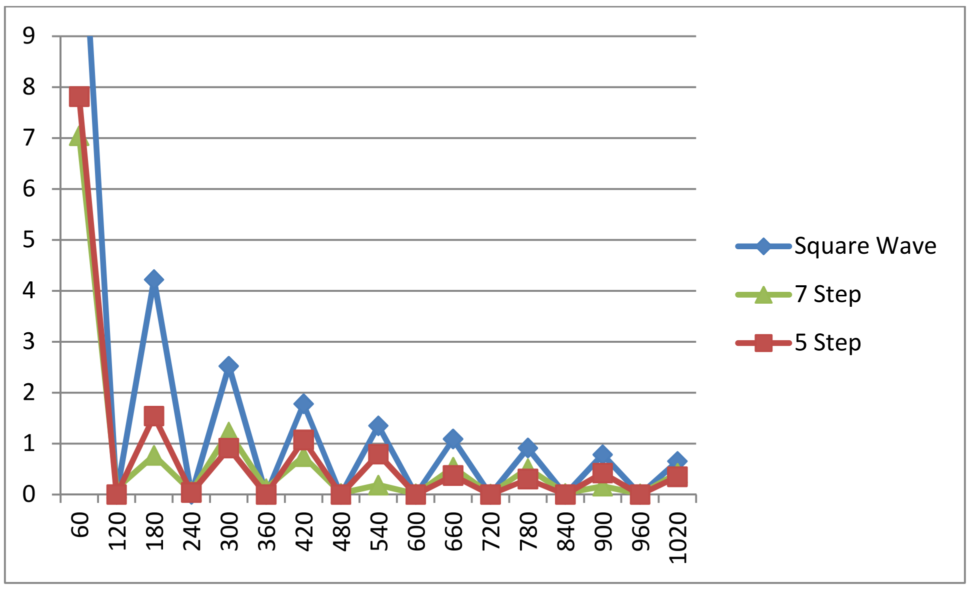


Figure 5-1. Fourier Analysis Comparison (Volts versus Hertz)

By looking at Figure 5-1, one can tell that the harmonics of the multistep inverters are lower than the harmonics of the square wave inverter, especially for the lower harmonics. The harmonics of the seven-step inverter are nearly nonexistent compared to the harmonics of the square wave inverter. This is what we expected. Since the multistep inverter designs were able to decrease the lower harmonics of the output voltage, those harmonics will be easier to filter out than the large harmonics of the square wave inverter.

CHAPTER 6. CONCLUSION AND FURTHER WORK

In this project, the research, design and simulation of an interconnected H-bridge single-phase inverter was explored. The inverter was simulated and recommendations were made for implementing a hardware prototype. The purpose of this senior project was to draw conclusions on whether the inverter produces less lower harmonics than a square wave inverter. For that reason, three different inverter circuits were simulated, a square wave inverter, a five-step inverter and a seven-step inverter. The simulations showed that interconnected H-bridge multistep inverters do succeed in diminishing the lower harmonics of the outputted AC voltage. All harmonics were decreased and most notably the 180 Hz harmonic was reduced from 4.22 V for the square wave inverter to .769 V for the seven-step inverter. Additionally, the seven-step inverter provided a "cleaner" output voltage than the five-step inverter. The conclusion can be drawn that the more steps that an inverter output voltage contains, the cleaner (less harmonics) the AC output voltage. The more steps of an inverter output voltage, the more accurately it approximates a pure sinusoidal AC voltage. The control signals for the multistep inverter designs are more complicated than the square wave inverter's control signals, but the cleaner voltage output of the multistep inverter designs make them better suited for modern inverter applications such as converting the DC power of solar panels to clean AC power.

If one were to commercially produce this interconnected H-bridge inverter for sale, modifications could be made to the prototype discussed in order to achieve: reduced size, cost savings and higher efficiency. A commercial version should not use a separate CPLD development board to create the control signals. A smaller CPLD board would be used that draws its power from the DC power inputted to the inverter rather than drawing power from separate batteries as the prototype CPLD board does. Additionally, surface mount integrated circuits would be used instead of thru-hole circuits in order to reduce size and increase efficiency. Also, MOSFET drivers can be used in the control signal amplification stage of the design rather than non-inverting operation amplifiers. Lastly, the MOSFETs and capacitors chosen for a commercial design would not be as robust as those chosen for the prototype. Cost would be reduced by using MOSFETs and capacitors that meet but not exceed the voltage and current ratings required.

BIBLIOGRAPHY

Digilent. 2004. *Digilent XC2-XL System Board Reference Manual.*

Mealy, Bryan. 2004. *The Low-Carb VHDL Tutorial*

Taufik. 2006. *Introduction to Power Electronics*

APPENDIX A - FSM PROTOTYPE VHDL CODE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity clock is
   Port ( CLK : in  STD_LOGIC;
          CLK_OUT : out  STD_LOGIC);
end clock;

architecture Behavioral of clock is

--signal COUNT : integer range 0 to 15359 := 0; -- intermediate count signal
constant COUNT_MAX : integer := 1800000;
signal CLK_LOCAL : STD_LOGIC;

begin

--Assign signals to the outputs
CLK_OUT <= CLK_LOCAL;

--Making a little counter / Clock Divider (15 bits)

process (CLK)
--Declare clock divider count variable
--Change this variable in order to change the frequency.
--Use 15359 for 60Hz, some fraction of that for higher frequencies

variable COUNT : integer range 0 to 1843200 := 0;

begin

            if (rising_edge(CLK)) then
            if (COUNT = COUNT_MAX) then --Check to see if we are at final count
                  COUNT := 0; --Reset Clock
                  CLK_LOCAL <= NOT CLK_LOCAL; --Update outputs
            else
                  COUNT := COUNT + 1; --Increment clock
            end if;
      end if;
end process;

end Behavioral;



--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity stat_mach is
   Port ( CLK : in  STD_LOGIC;
         CLR : in  STD_LOGIC;
         SW1 : out  STD_LOGIC;
         SW2 : out  STD_LOGIC;
         SW3 : out  STD_LOGIC;
         SW4 : out  STD_LOGIC;
         SW5 : out  STD_LOGIC;
         SW6 : out  STD_LOGIC;
         SW7 : out  STD_LOGIC;
         SW8 : out  STD_LOGIC;
         SW9 : out  STD_LOGIC);
end stat_mach;

architecture Behavioral of stat_mach is
        type state_type is (ST0,ST1,ST2,ST3,ST4,ST5,ST6,ST7,ST8,ST9,ST10,ST11,DELAY);
        signal PS,NS : state_type;
        constant COUNT_MAX : integer := 1800000;  --added to implement clock  log in this cpld
        constant DELAY_DESIRED : integer := 1;  -- used for implementing larger delays
        constant 720HZ_MAX : integer := 1280;  -- used for setting length of output states of FSM

        variable 720HZ_CLK_COUNT : integer range 0 to 1843200 := 0; --allows for up to 1.84
gHz
        variable DELAY_COUNT : integer range 0 to 1843200 := 0;


SHARED variable DELAY_ON : integer range 0 to 1000000 := 0; -- 1 or 0 for delay state
SHARED variable COUNT : integer range 0 to 11 := 0; -- used to implement delay state

begin

	sync_proc: process(CLK,NS,CLR)
	begin
		if (CLR = '0') then
			PS <= ST0;
	--		PS <= PS;	-- Holds the present state when button is pressed?????z
	--		COUNT := 0;

		elsif (rising_edge(CLK)) then
----------------------------------------------------------------------------- code in segment below is for
clock.
--logic below uses counters to determine the timing for the delay and output states of the FSM

			If (DELAY_ON = 1) then  --We are in the delay state
				If (DELAY_COUNT = DELAY_DESIRED) --desired is 1 for 542 ns
del
					DELAY_COUNT := 0;
					DELAY_ON := 0;
					PS <= NS;
				else
					DELAY_COUNT := DELAY_COUNT + 1;
				end if;
			else
				if (720HZ_CLK_COUNT = COUNT_MAX) then  -- to prevent errors
					720HZ_CLK_COUNT := 0;
				elsif (720HZ_CLK_COUNT = 720HZ_MAX) then
					720HZ_CLK_COUNT := 0;
					DELAY_ON :=1; -- turns on delay bc output state is over
					PS <= NS;
				else
					720HZ_CLK_COUNT := 720HZ_CLK_COUNT + 1;
				end if;
				end if;
			end if;
		end if;


	end process sync_proc;

-------------------------------------------------

comb_proc: process(PS)

variable COUNT : integer range 0 to 11 := 0; -- used to implement delay state
```

```
begin
        case PS is
                when ST0 => -- items regarding state ST0
                        COUNT := 1;
                        NS <= DELAY;

                when ST1 => -- items regarding state ST1
                        COUNT := 2;
                        NS <= DELAY;

                when ST2 => -- items regarding state ST2
                        COUNT := 3;
                        NS <= DELAY;

                when ST3 => -- items regarding state ST3
                        COUNT := 4;
                        NS <= DELAY;

                when ST4 => -- items regarding state ST4
                        COUNT := 5;
                        NS <= DELAY;

                when ST5 => -- items regarding state ST5
                        COUNT := 6;
                        NS <= DELAY;

                when ST6 => -- items regarding state ST6
                        COUNT := 7;
                        NS <= DELAY;

                when ST7 => -- items regarding state ST7
                        COUNT := 8;
                        NS <= DELAY;

                when ST8 => -- items regarding state ST8
                        COUNT := 9;
                        NS <= DELAY;

                when ST9 => -- items regarding state ST9
                        COUNT := 10;
                        NS <= DELAY;

                when ST10 => -- items regarding state ST10
                        COUNT := 11;
                        NS <= DELAY;

                when ST11 => -- items regarding state ST11
```

```
                        COUNT := 0;
                        NS <= DELAY;

                when DELAY => -- items regarding state DELAY
-- the conditionals determine which will be the NS after the delay
                        if ( COUNT = 0) then
                                NS <= ST0;
                        elsif (COUNT = 1) then
                                NS <= ST1;
                        elsif (COUNT = 2) then
                                NS <= ST2;
                        elsif (COUNT = 3) then
                                NS <= ST3;
                        elsif (COUNT = 4) then
                                NS <= ST4;
                        elsif (COUNT = 5) then
                                NS <= ST5;
                        elsif (COUNT = 6) then
                                NS <= ST6;
                        elsif (COUNT = 7) then
                                NS <= ST7;
                        elsif (COUNT = 8) then
                                NS <= ST8;
                        elsif (COUNT = 9) then
                                NS <= ST9;
                        elsif (COUNT = 10) then
                                NS <= ST10;
                        elsif (COUNT = 11) then
                                NS <= ST11;
                        end if;

                when others => -- the catch-all condition
                        NS <= ST0; -- make it to these three statements
                        COUNT := 0;
                        DELAY_ON :=0;
        end case;
end process comb_proc;

with PS select
        SW1 <= '0' when ST0,
                '0' when ST1,
                '0' when ST2,
                '0' when ST3,
                '0' when ST4,
                '0' when ST5,
                '1' when ST6,
                '0' when ST7,
                '0' when ST8,
                '0' when ST9,
                '0' when ST10,
                '0' when ST11,
                '0' when others;

with PS select
        SW2 <= '1' when ST0,
                '1' when ST1,
                '0' when ST2,
                '1' when ST3,
                '0' when ST4,
                '0' when ST5,
                '0' when ST6,
                '0' when ST7,
                '0' when ST8,
                '1' when ST9,
                '0' when ST10,
                '1' when ST11,
                '0' when others;

with PS select
        SW3 <= '1' when ST0,
                '0' when ST1,
                '0' when ST2,
                '1' when ST3,
                '0' when ST4,
                '0' when ST5,
                '0' when ST6,
                '0' when ST7,
                '0' when ST8,
                '1' when ST9,
                '0' when ST10,
                '0' when ST11,
                '0' when others;

with PS select
        SW4 <= '1' when ST0,
                '1' when ST1,
                '1' when ST2,
                '0' when ST3,
                '0' when ST4,
                '0' when ST5,
                '0' when ST6,
                '0' when ST7,
                '0' when ST8,
                '0' when ST9,
                '1' when ST10,
                '1' when ST11,
                '0' when others;

with PS select
        SW5 <= '0' when ST0,
                '0' when ST1,
                '0' when ST2,
                '1' when ST3,
                '1' when ST4,
                '1' when ST5,
                '1' when ST6,
                '1' when ST7,
                '1' when ST8,
                '1' when ST9,
                '0' when ST10,
                '0' when ST11,
                '0' when others;

with PS select
        SW6 <= '0' when ST0,
                '0' when ST1,
                '1' when ST2,
                '0' when ST3,
                '0' when ST4,
                '0' when ST5,
                '0' when ST6,
                '0' when ST7,
                '0' when ST8,
                '0' when ST9,
                '1' when ST10,
                '0' when ST11,
                '0' when others;

with PS select
        SW7 <= '0' when ST0,
                '0' when ST1,
                '0' when ST2,
                '0' when ST3,
                '0' when ST4,
                '1' when ST5,
                '0' when ST6,
                '1' when ST7,
                '0' when ST8,
                '0' when ST9,
                '0' when ST10,
                '0' when ST11,
                '0' when others;

with PS select
```

```
        SW8 <= '0' when ST0,
               '1' when ST1,
               '0' when ST2,
               '0' when ST3,
               '0' when ST4,
               '0' when ST5,
               '0' when ST6,
               '0' when ST7,
               '0' when ST8,
               '0' when ST9,
               '0' when ST10,
               '1' when ST11,
               '0' when others;

with PS select
        SW9 <= '0' when ST0,
               '0' when ST1,
               '0' when ST2,
               '0' when ST3,
               '1' when ST4,
               '0' when ST5,
               '0' when ST6,
               '0' when ST7,
               '1' when ST8,
               '0' when ST9,
               '0' when ST10,
               '0' when ST11,
               '0' when others;
end Behavioral;
```