Analysis of Single Phase Matrix Converter on Wound Field Synchronous Machine

By

Jianyang Liu

A thesis submitted in partial fulfillment of

the requirements for the degree of

Master of Science

(Electrical Engineering)

at the

UNIVERSITY OF WISCONSIN – Madison

2015

Abstract

This article focuses on the application of Single Phase Matrix Converter on the field excitation for wound field synchronous machine. As the price for rare earth metal is increasing all these years, the cost to build permanent machine is also increasing. Wound field synchronous machine will be a good substitution since no rare earth metal is needed. However, would field synchronous machine has slip rings which will cause a number of problems. In this research project, a method by using single phase matrix converter with a rotating transformer is proposed to eliminate the slip ring. The first part of the project is to study the use of a Single Phase Matrix Converter (SPMC) to step up the frequency in order to reduce the size of the transformer; the second part of the project is the design of a rotating transformer to step up the voltage for the field winding of WFSM. The rectifier diodes bridge is embedded in the rotating transformer so there will not be slip rings in the WFSM. The proposed system is verified in the simulation model and the hardware for the Single Phase Matrix Converter is also built.

Acknowledgement

First and foremost, I would like to extend my sincere gratitude to my supervisor, Professor. Lipo, for all his instructive teaching and advising on my thesis. It is his patience and illuminating instruction in electric machines and power electronics makes me confident in these years' study.

I am also deeply indebted to all faculty members from WEMPEC, who have helped me to develop the fundamental and essential academic competence. Their lectures enrich me the understanding in power electronics, machine design and machine control.

Special thanks should go to my roommate Hao Jiang. who helped me to come through the challenges in the hardware implementation. Also, I appreciate Junjian Zhao, Jiyao Wang, Ye Li, Bo Zhu, Parikshith Channegowda for their help through the project. Thank all the WEMPEC students for their friendship and encouragement during my study at the University of Wisconsin.

Highest tribute shall be paid to my beloved parents. Thanks for their financial support for all these years' study and for their selfless love throughout my whole life.

Table of Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION AND STATE-OF-ART-REVIEW

1.1 Introduction

Since the price for rare earth metal is increasing drastically these years, people are beginning to look for a good substitution for Permanent Magnet Machines. Because most Permanent Magnet Machines and Wound Field Synchronous Machine belong to synchronous machine, there is no need to use rare earth metal to construct a Wound Field Synchronous Machine. However, there are some problems that constrain the using of the Wound Field Synchronous Machine. One of the most important problems is the using of slip rings for WFSM.

In industry, people are using brushless exciter, a small synchronous machine, as a kind of substitution for slip rings. But it is hard to be used to drive a larger synchronous machine, As the cost for this system needs to be considered. Some other people are using an inverter with a rotating transformer to provide the field excitation, but the size for this system is quite big due to the DC bus capacitor.

In this research project, a way to use Single Phase Matrix Converter with Rotating Transformer as a substitution for Slip Ring is proposed. Since in Matrix converter family the bulky capacitor is not needed as well as the high frequency transformer is small, the system is not only efficiency but compact.

1.2 Literature Review on Single Phase Matrix Converter

The matrix converter topology was first proposed by Gyugyi in 1976 [1]. He proposed a three phase matrix converter topology which can achieve direct AC to AC frequency conversion. After that, researches based on Three Phase Matrix Converter or Dual Bridge Matrix Converter has grown, since both converters have high voltage transfer ratio. The first Single Phase Matrix Converter was realized by Zuckerberger [2] in 1997. The benefit for using direct AC to AC matrix converter over inverter is the absence of bulky capacitor. Although there are many research projects on Matrix Converter, the application in industry is limited. However, with the latest fast Power Electronics switches and DSP, the application of Matrix Converter in industry in feasible. The limitation for matrix converter is mainly on the safe commutation strategy due to the bi-direction current flowing, among all these years people have proposed many successful commutation strategy [3-20]. The application for Single Phase Matrix Converter is not as broad as three phase matrix converter due to the lower voltage transfer ratio and the safe commutation strategy problem. There are some applications that focus on the transformer load for Single Phase Matrix Converter [21-22], however, their modulation method creates high frequency harmonics so that the loss for the transformer is not optimized.

1.3 Bi-directional Switches

The Matrix Converter is different from an Inverter since the input is AC but not DC. As a result, the switches used in Matrix Converter must have bi-direction

conductivity. Traditional MOSFET used in inverter is shown in Figure.1.1 (a). However, this switch could only be used for single directional voltage since there is no reverse blocking capability. Figure 1.1 (b) shows the bridge connection of bi-directional conduction switch. The advantage of this connection method is it has the minimal number of the controlled MOSFET than (c) (d) (e) and (f). But some huge problems was generated by using this switch on Matrix Converter that there is no path for the reverse current to flow, which makes safe commutation almost impossible. The using of Diode Bridge also makes this kind of switch has more conduction loss than other kinds of topologies. Figure 1.1 (c) and (d) show the series connection of MOSFETs as a Bi-Directional switch. There is a benefit of using topology (d) that both MOSFETs share a common source, so that we can use one power supply for the two MOSFETs' gate drive. This will simplify the circuit and also reduce the cost of the circuit. Figure 1.1 (e) and (f) show anti-parallel connecting of MOSFETs and IGBTs. In fact they share a same topology, but (f) are relatively new devices since each of the MOFETs can block reverse voltage. Topology (f) has been used for many applications for Three Phase Matrix Converter, but for low power application (d) it may be a better choice.

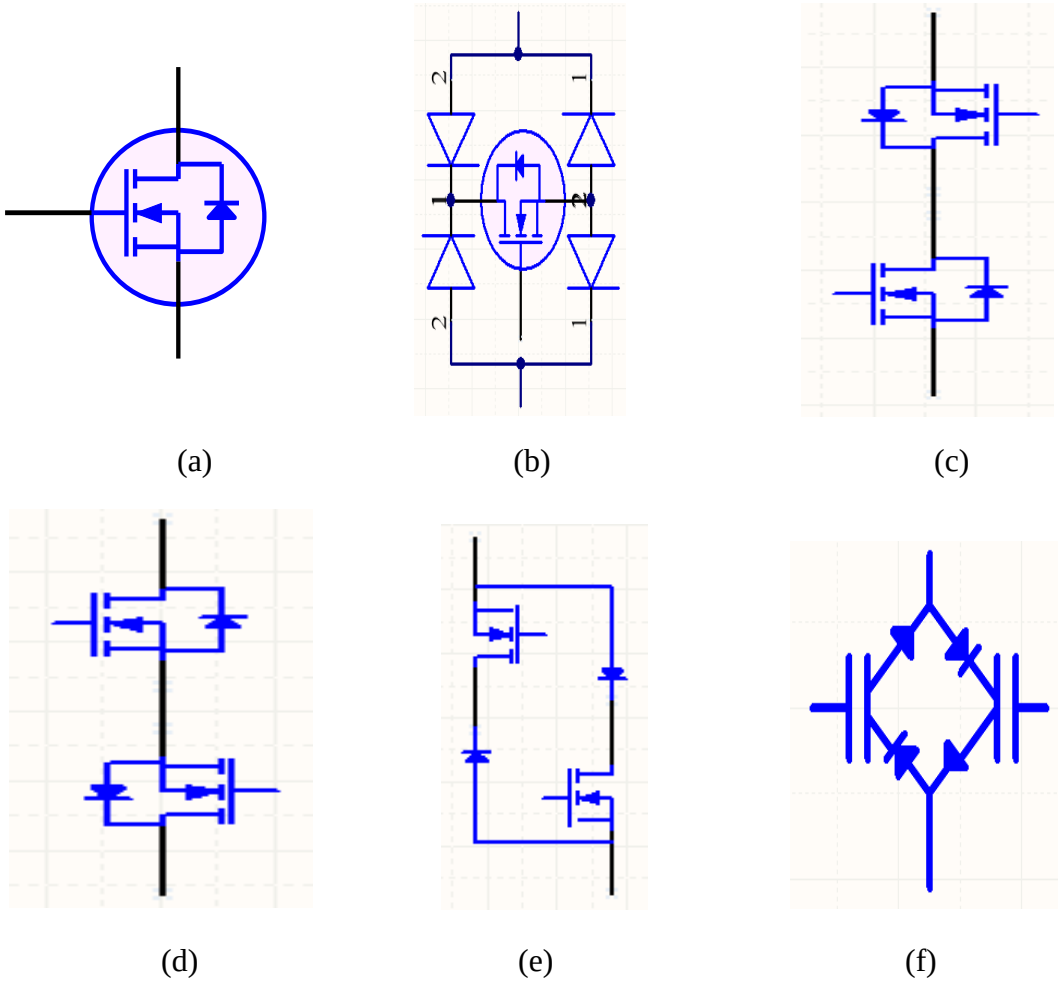

Figure 1.1. Switches used in Inverter and Matrix Converter: (a) Traditional single direction MOSFET used in inverter (b) Bridge connection bi-directional MOSFET (c) Series connection of bi-directional MOSFET share common drain (d) Series connection of bi-directional MOSFET share common source (e) Anti-parallel connected MOSFET (f) Anti-parallel connected reverse blocking IGBT

1.4 Other Issues Related to Matrix Converter

1.4.1 Input Filter Design

The main advantage of the Matrix Converter is the absence of the bulky DC Link capacitor. However, even in Matrix Converter an input filter is required to reduce

the switching noise to effect the input voltage. It acts as a buffer circuit from the output to the input AC source. Figure 1.2 shows the stage for the input filter in the Single Phase Matrix Converter. By using the input filter, the input voltage and current will not suffer from a sudden change from the output switching noise. More important, the input current for the matrix converter is discontinuous, thus the input filter can also reduce the effect when output harmonics current flow into in the input AC source. One thing that has to be noticed is at certain current level, the size of the input filter has become comparable to a DC link capacitor. There are two types of the input filters, one is passive filter which consists only a single or multiple stages of LC circuit. This kind of input filter is easy to implement and compact in size. The other one is the active input filter, which will have a better performance but is more complicated than the passive one and not compact in size. For both of the input filters, it is all in need of a damping resistor in parallel with the inductor to damp out the spike voltage and current. Figure 1.4 shows the single stage passive LC input filter for Single Phase Matrix Converter.

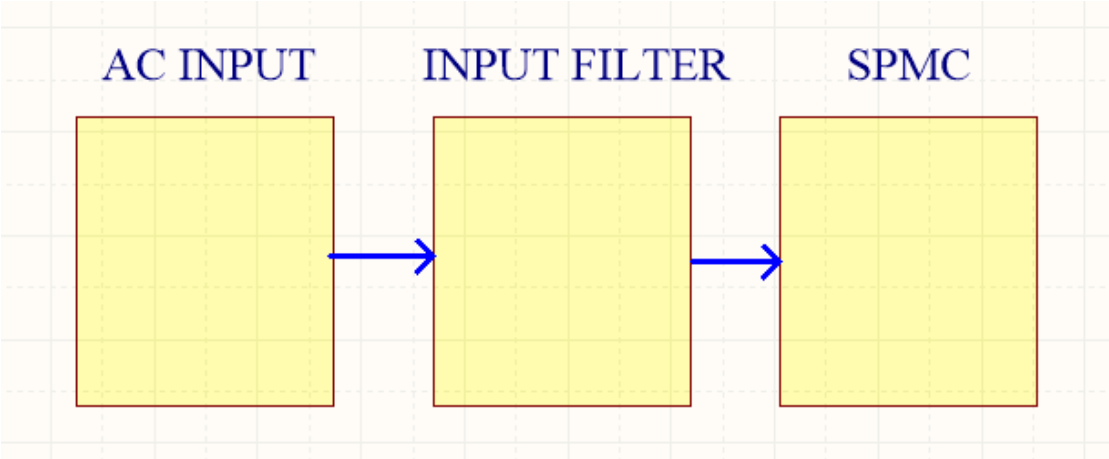

Figure 1.2 Input State of the Single Phase Matrix Converter

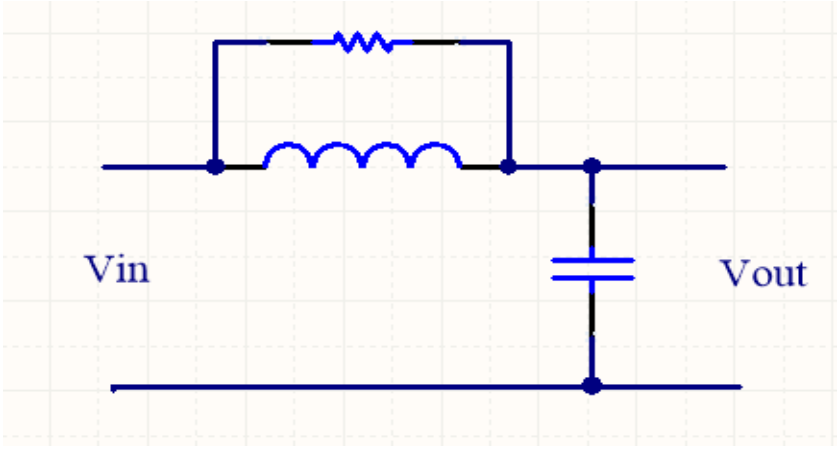

Figure 1.3 Single Stage Passive LC input Filter

1.4.2 Clamp and Snubber Circuit

Clamp Circuit is a very important part for every matrix converter since Matrix Converter does not have a passive reverse freewheeling diode. The current can only find a path to flow when an enable signal comes to the switch. As a result, the clamp and snubber circuit are needed to provide a path for current to flow during commutation or fault condition. When all the switches are turned off, the clamp circuit can also help to damp out the energy stored in the induction load which prevents the spike voltage to damage the switches. In most common case, the clamp circuit consists of two bridge circuits from the input side and output side in Figure.1.3. The capacitor in parallel with the bridge circuit can store the transient energy of the induction load, and the resistor in parallel with the capacitor can help to damp out the energy stored in the capacitor. For the bi-directional switches, they also need their own snubber circuit to reduce the effect of the switches capacitance. This snubber circuit is a very compact one which consists of only a resistor and a capacitor in parallel with the switch.

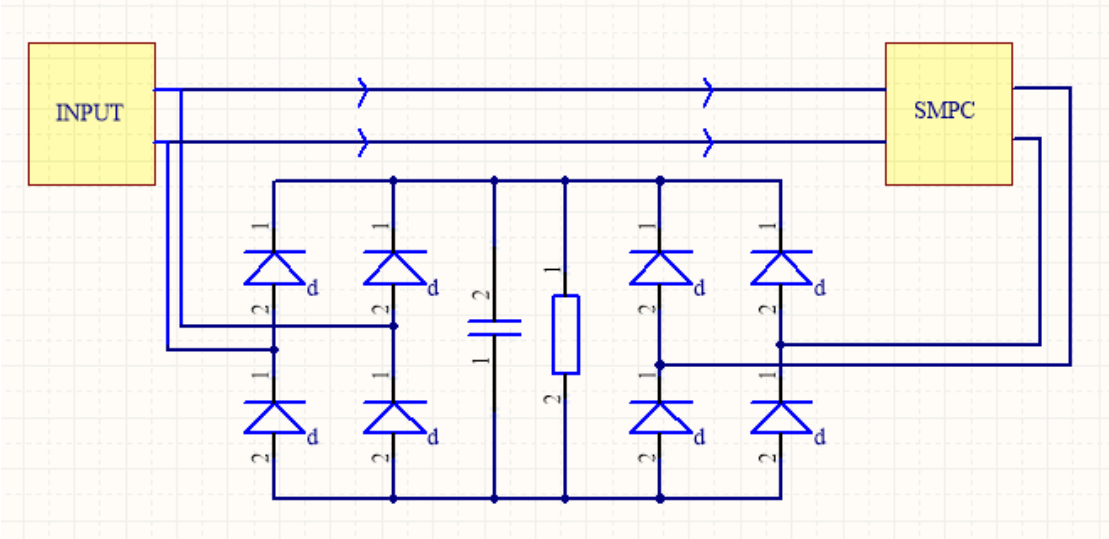

Figure 1.4 Clamp Circuit for Single Phase Matrix Converter

1.5 Scope of the Thesis

- In Chapter 2, the theoretical foundation for the Single Phase Matrix Converter and its modulation method are explained. Two different types of topologies and their commutation strategies are proposed.
- In Chapter 3, the simulation result for the SMPC is presented and the limitation of using simulation software is pointed out.
- In Chapter 4, the experiment prototype is presented and the experiment result is analyzed.
- In Chapter 5 the final conclusion and suggested future work is presented.

Chapter 2

Modeling of SPMC Used for WFSM

2.1 Topology for Single Phase Matrix Converter

Single Phase Matrix Converter is consisted of four bi-directional switches, in this research project two different topologies for Single Phase Matrix Converter is built. One of them is using the bridge circuit to create a bi-directional switch shows in Figure.2.1.

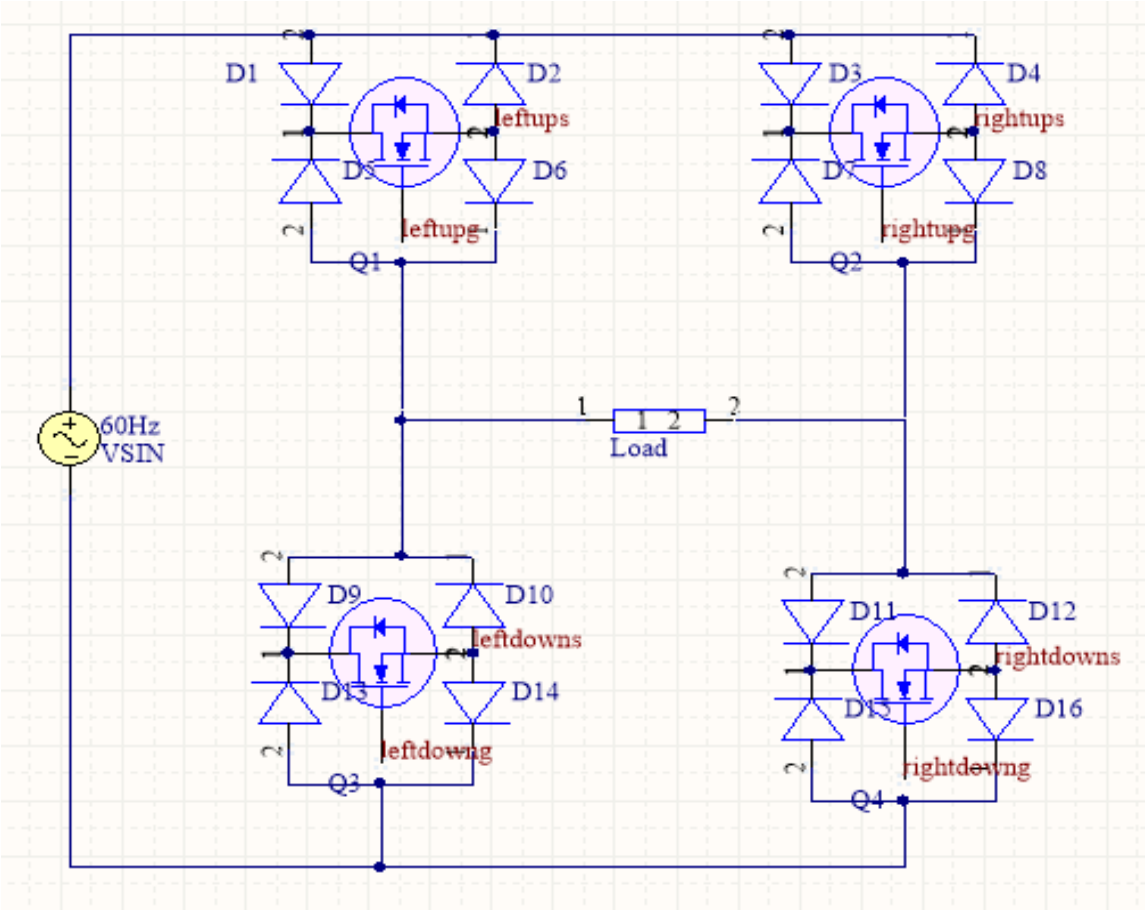

Figure 2.1 Bridge Connection for Single Phase Matrix Converter

Using this kind of topology will reduce the number of the active controllable switches (MOSFET or IGBT). However, because of the lack of reverse blocking component this kind of topology is hard to achieve safe commutation during each deadtime.

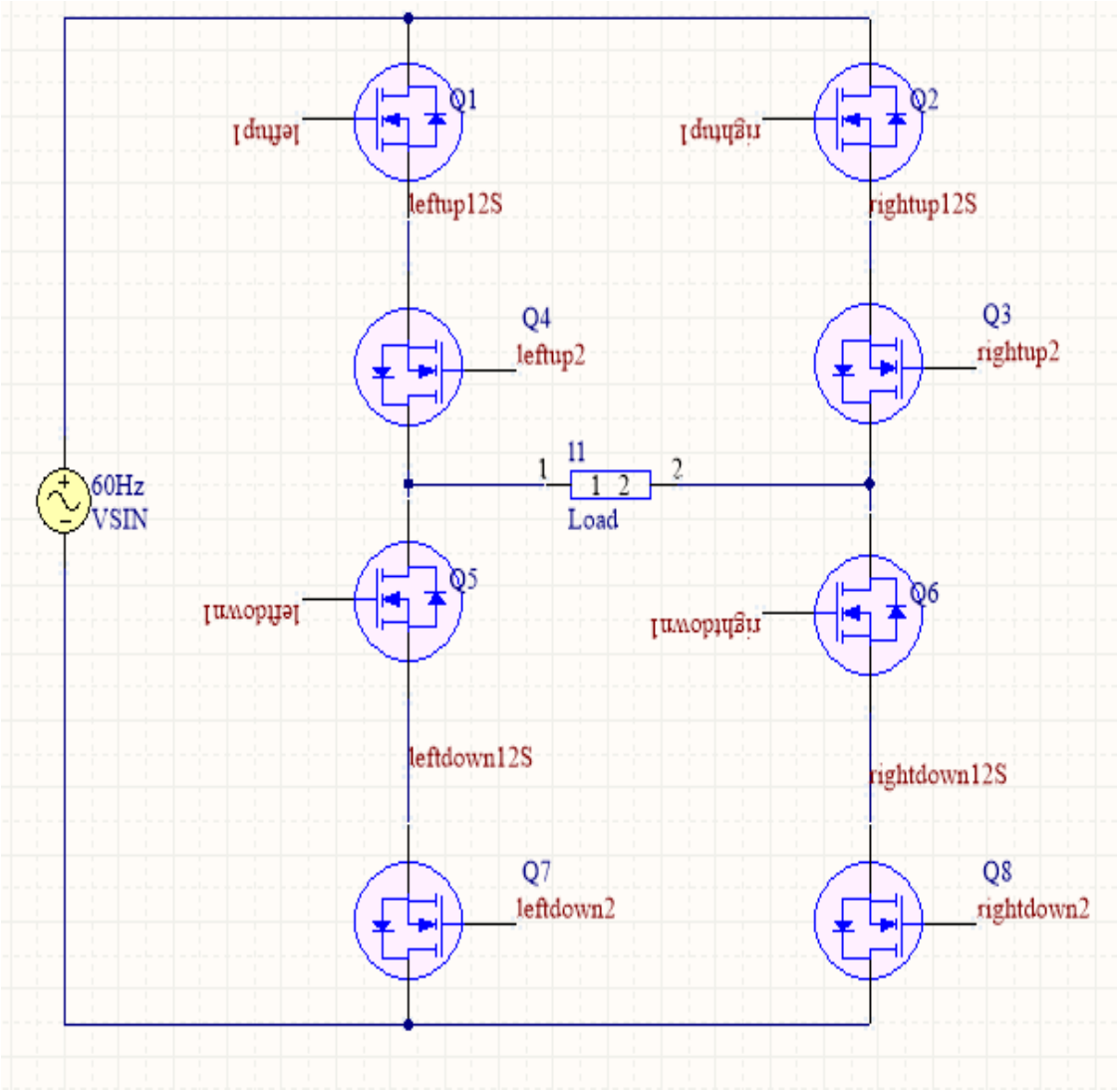

Figure 2.2 Single Phase Matrix Converter Using Back to Back Connection Switches

Figure 2.2 shows the matrix converter using back to back connection topology, and the benefit of using this topology is that each bi-directional switch only needs one power supply for the gate drive. More importantly, this topology has a reverse blocking element which make safe commutation possible.

2.2 Modulation Method

The application of Single Phase Matrix Converter is not as broad as Three Phase Matrix converter. However, there are still some applications of using Single Phase Matrix converter to reduce the size of the transformer. The modulation method for the traditional modulation is shown in Figure 2.3.

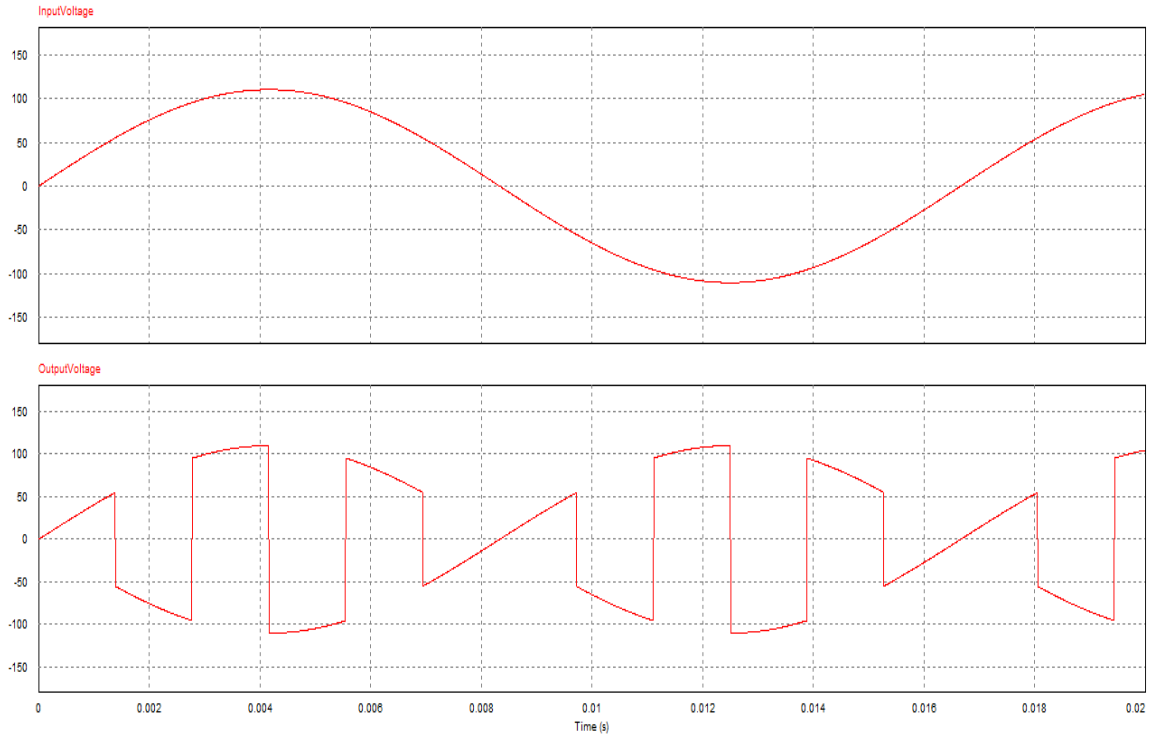

Figure 2.3 Traditional Modulation Method Based on SPMC for Transformer Application

The advantage of using this kind of modulation method is, the output voltage could has the largest voltage transfer ratio based on a certain amount of the input voltage. This also means the output of the circuit has more volts per Hz during each switching period. The disadvantage of this modulation method is that during each switching interval, the Volts per Hz is not a constant value, which will introduce some higher order harmonics voltage and current in the output, and cause more loss in the transformer.

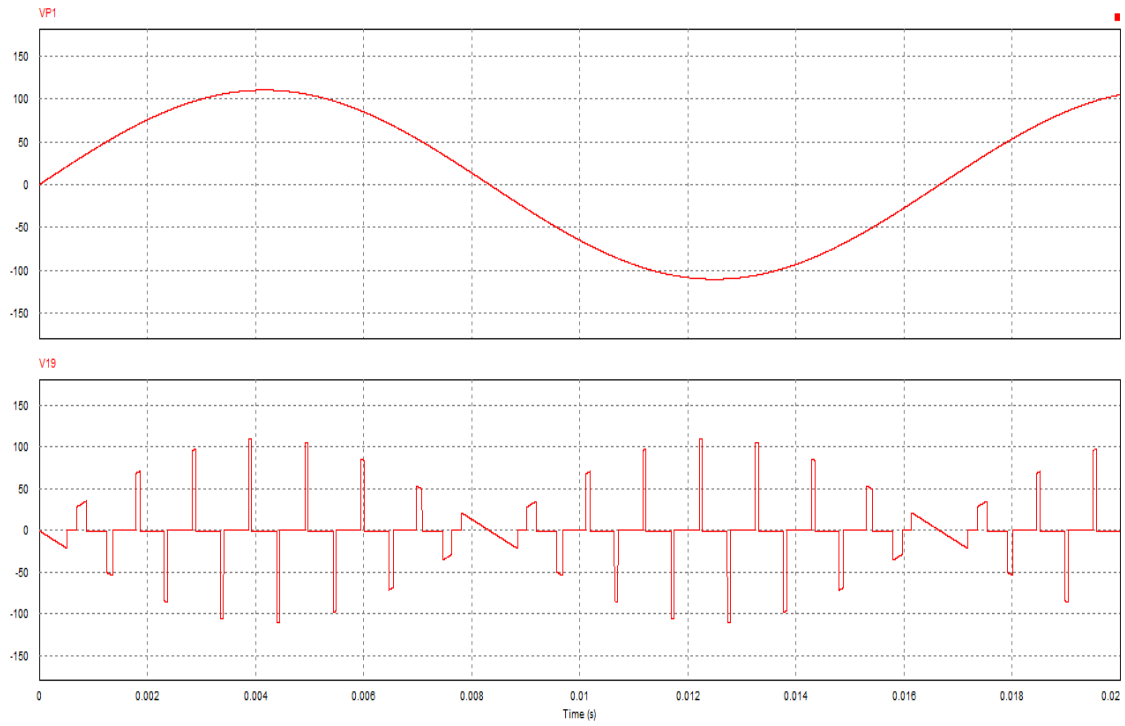

Figure 2.4 New Modulation Method Based on SPMC for Transformer Application

Since the disadvantage exists on the traditional modulation method for SPMC. A new modulation method is proposed in this research project. In this modulation method, during each switching interval the output Volts per Hz is kept at a constant value (shows on Figure 2.4). As a result, the fundamental component of the output voltage is almost constant during each switching interval, this modulation method will reduce the output harmonic significantly thus the loss for the transformer will also reduce to a reasonable value.

2.3 Commutation Strategy

The most challenge part for the matrix converter is safe commutation. Not only because the switch number is doubled compared to an inverter, but because the polarity for the AC input voltage is constant changing, these differences make safe commutation

very crucial for matrix converter. A new commutation strategy based on back to back connection of the MOSFET is proposed in this research project to achieve safe commutation as shown blow. Switch Xa (x=1 to 4) conducts the current from up to down and switch Xb (x=1 to 4) conducts the current from down to up.

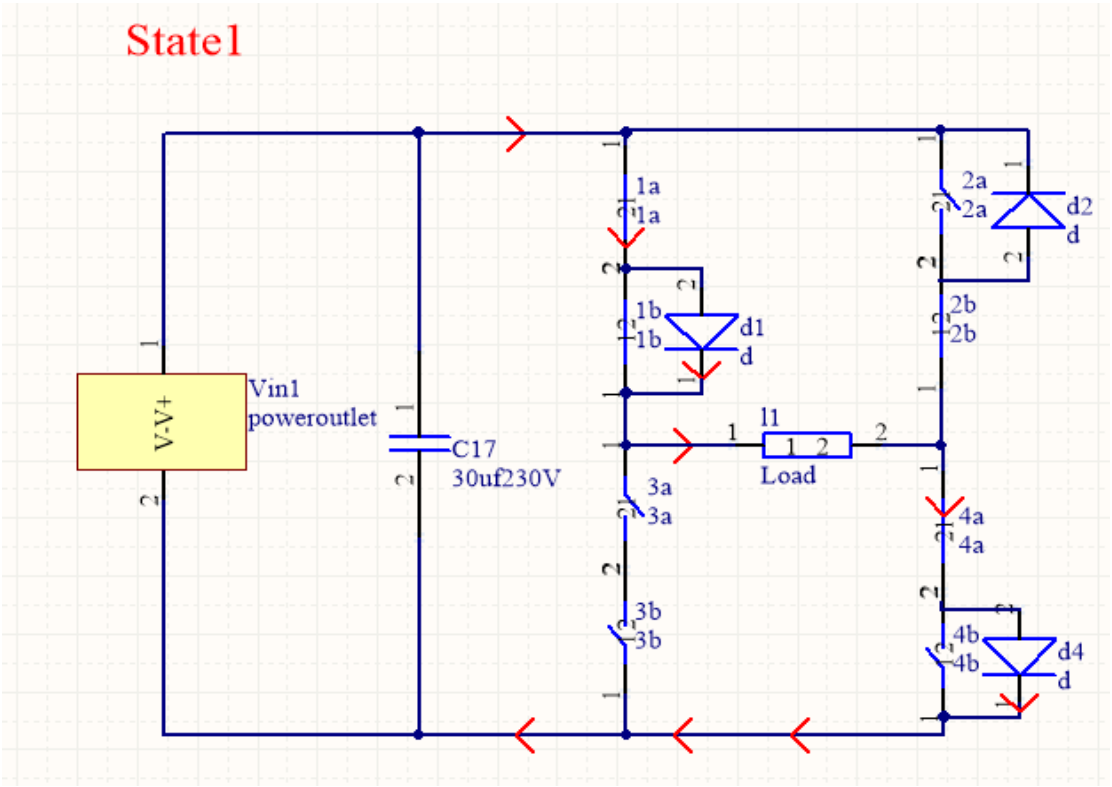

Figure 2.5 State One

First of all, we assume the circuit started at positive polarity when the input voltage is increasing from zero. The current flow is shown on Figure 2.5. In this state, switch 1a and 4a areconducting the positive current for the load. Switch 2b is also closed to prepare for state two. Since Matrix Converter always has an induction load, when we move from state one into state two, the current in the load should not be terminated, as a force shut down of the system can cause large voltage spike which will damage the devices. Thus, a current path must be provided for current continuing to flow after a positive or negative input cycle. This leads to state 2 in Figure 2.6.

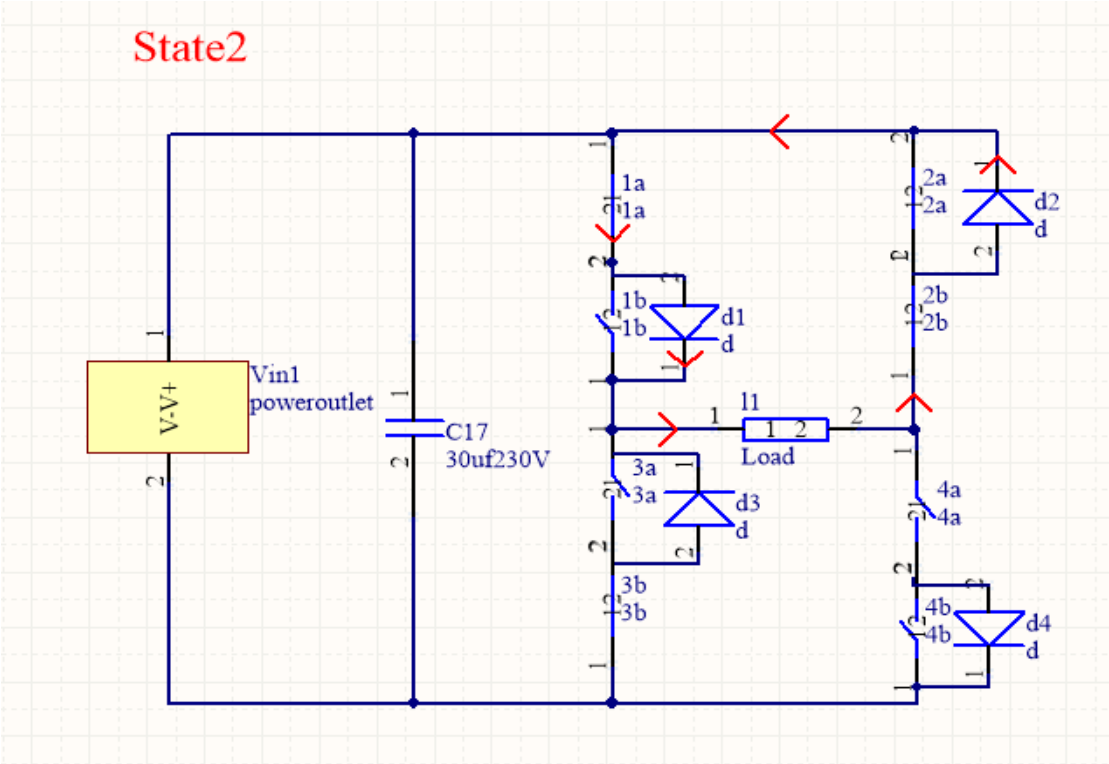

Figure 2.6 State Two

State two is a zero state followed by state one. In this state, no positive or negative input voltage is connected to the load. The current of the load maintains the same direction as state one. Switch 2b and 1a are enabled to continue conducting the current flows in positive polarity. Switch 3b is enabled for the next state.

In state three shown in Figure 2.6. If the load is a large inductive load, the positive current may not die out in state two. So we need to continue providing a path for current to flow in state three. In this state, switch 1a is turned off and switch 2b and 3b is turned on to conduct this positive current. Meanwhile, negative Volts per Hz is applied to the load. Switch 3a is turned on this time to provide a current path for next state.

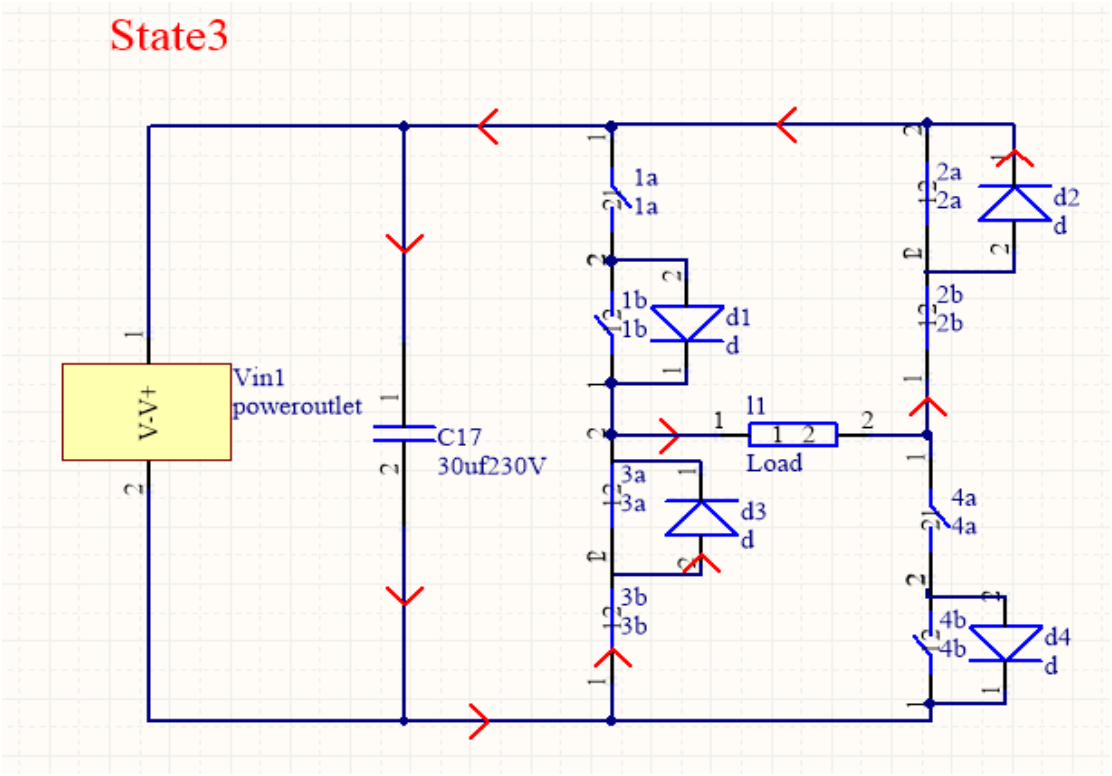

Figure 2.7 State Three

When positive output current dies out in state three, the current begins to flow in the negative direction shown in Figure 2.8. In this state, negative Volts per Hz is applied to the load and the current is flowing in the negative direction. Switch 2b is turned off and switch 4b is turned on to provide current path for next state.

In state five shown in Figure 2.9 switch 2a is turned off and the output current is continue to flow in the negative direction. This state is similar to the zero state in state two. Switch 1b is turned on to provide current path for the next state.

In state six shown in Figure 2.10, the input voltage is still in positive direction and a positive V/Hz is applied to the load. and negative output current has to die out before the output current becomes positive. Switch 3a is turned off to make the current flow through switch 1b. Switch 1a is turned on to provide a current path for the next state.

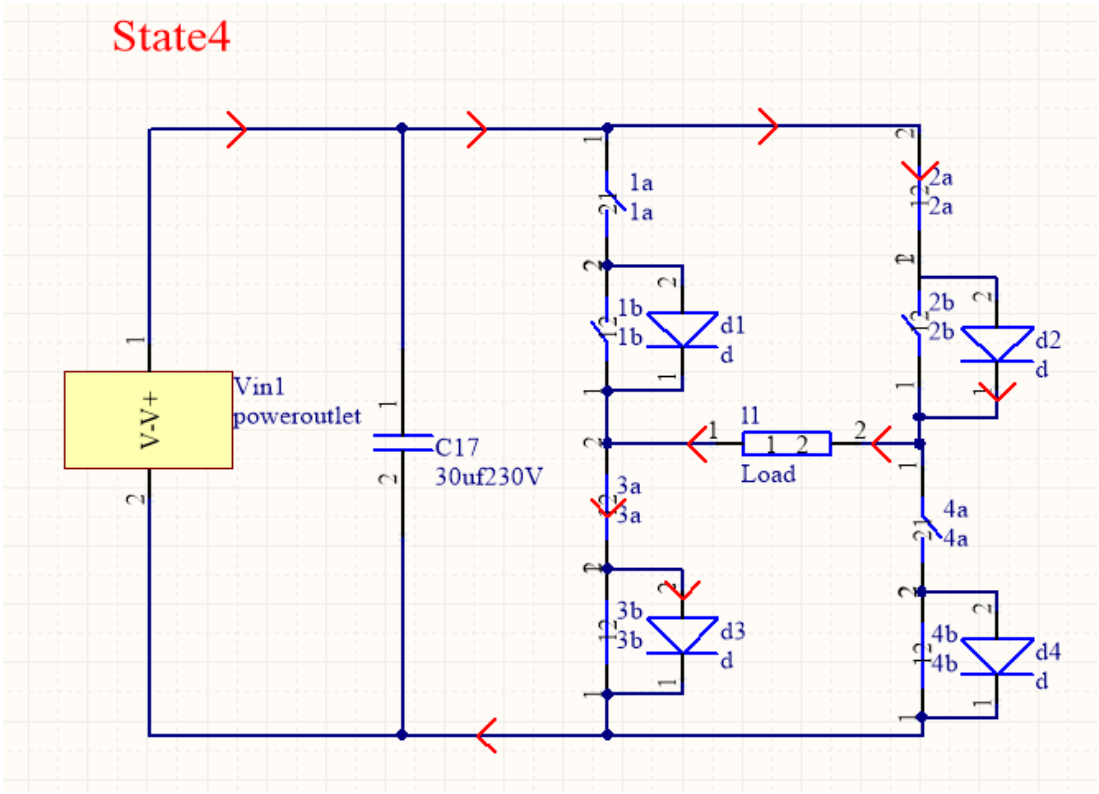

Figure 2.8 State Four

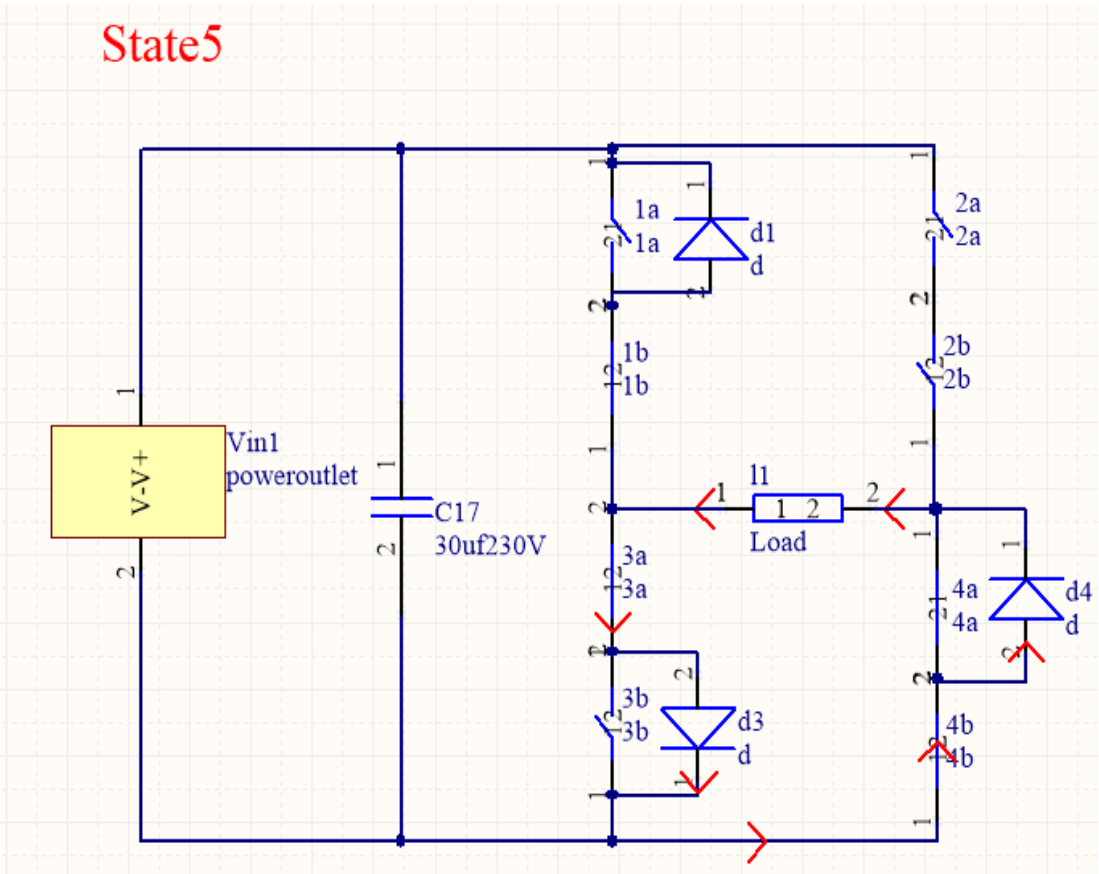

Figure 2.9 State Five

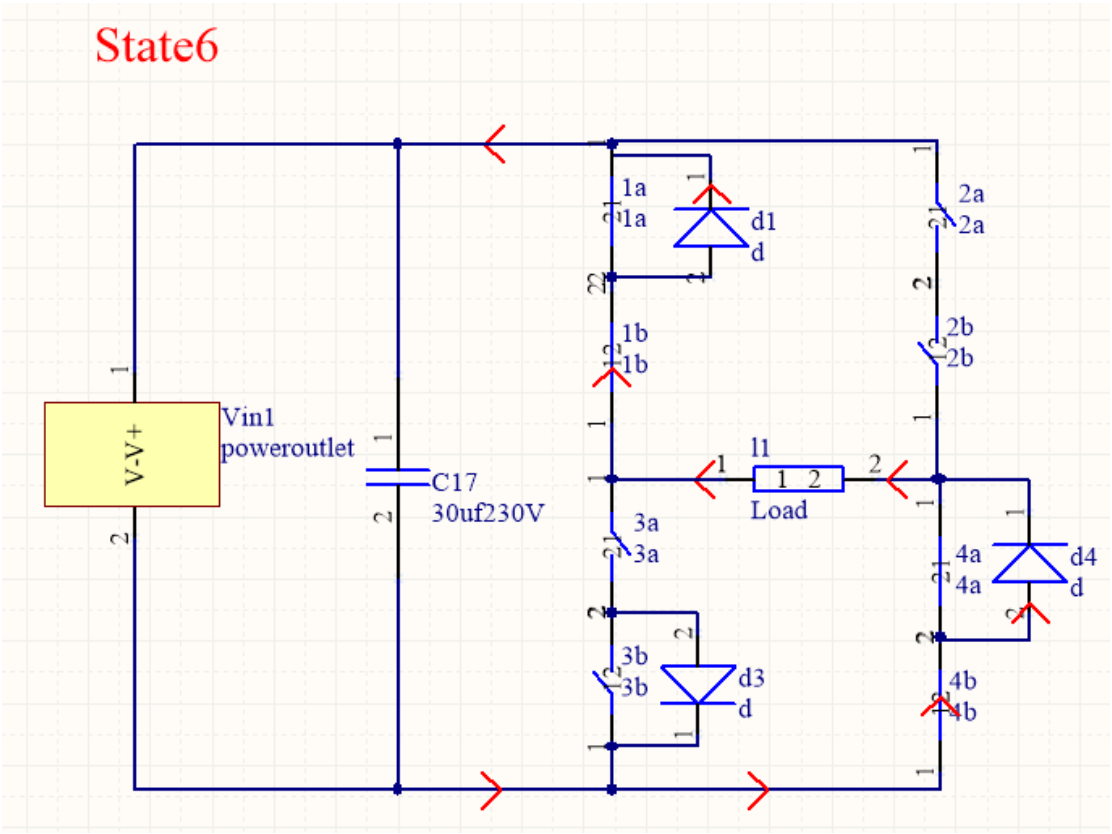

Figure 2.10 State Six

When input voltage polarity is still positive, after state 6 the following state will become state 1 again.

When input voltage changes polarity from positive to negative, the last state is state four. But in this state, switch 4b is not turned on as shown in Figure 2.11, since there is no zero state after this state. In this state, switch 2b is turned on to provide a current path for the next state (same as state three).

When input voltage polarity is negative and the voltage is increasing from zero. After the negative current from last state dies out as shown in Figure 12, switch 2a is turned off and switch 4a is turned on for the next state.

In State eight shown in Figure 2.13, the output current keeps flowing in positive direction and switch 1a is turned on for the next state.

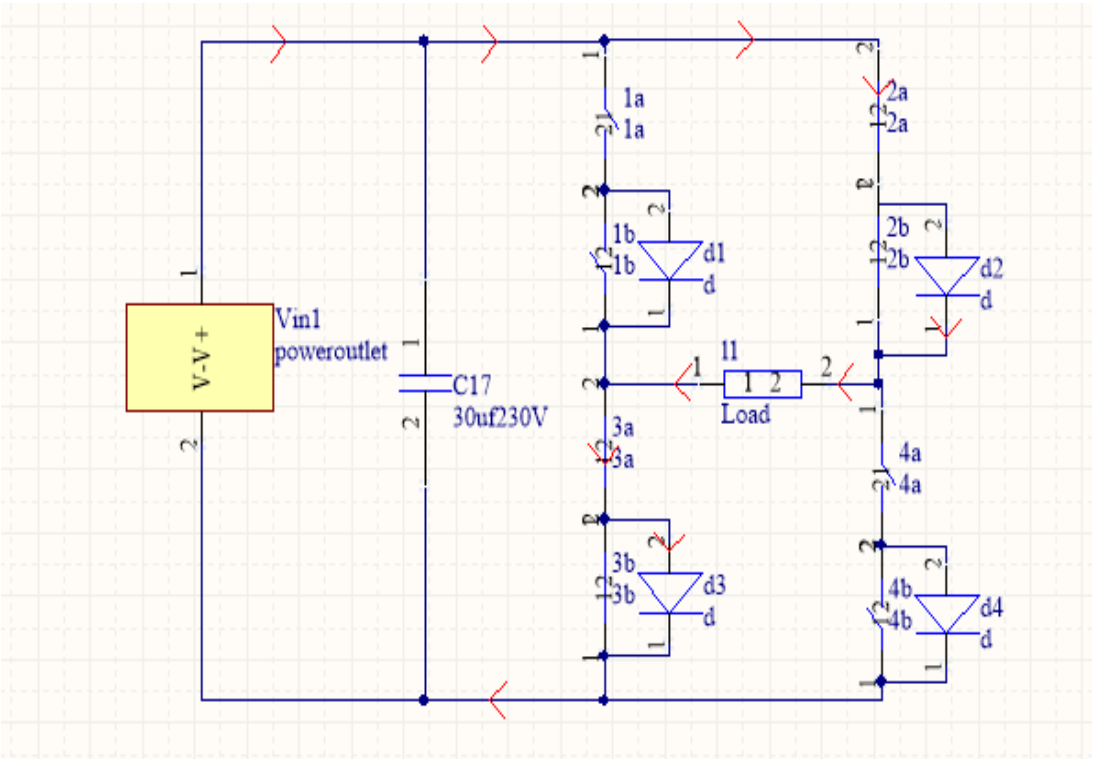

Figure 2.11 State when Input Voltage Change from Positive to Negative

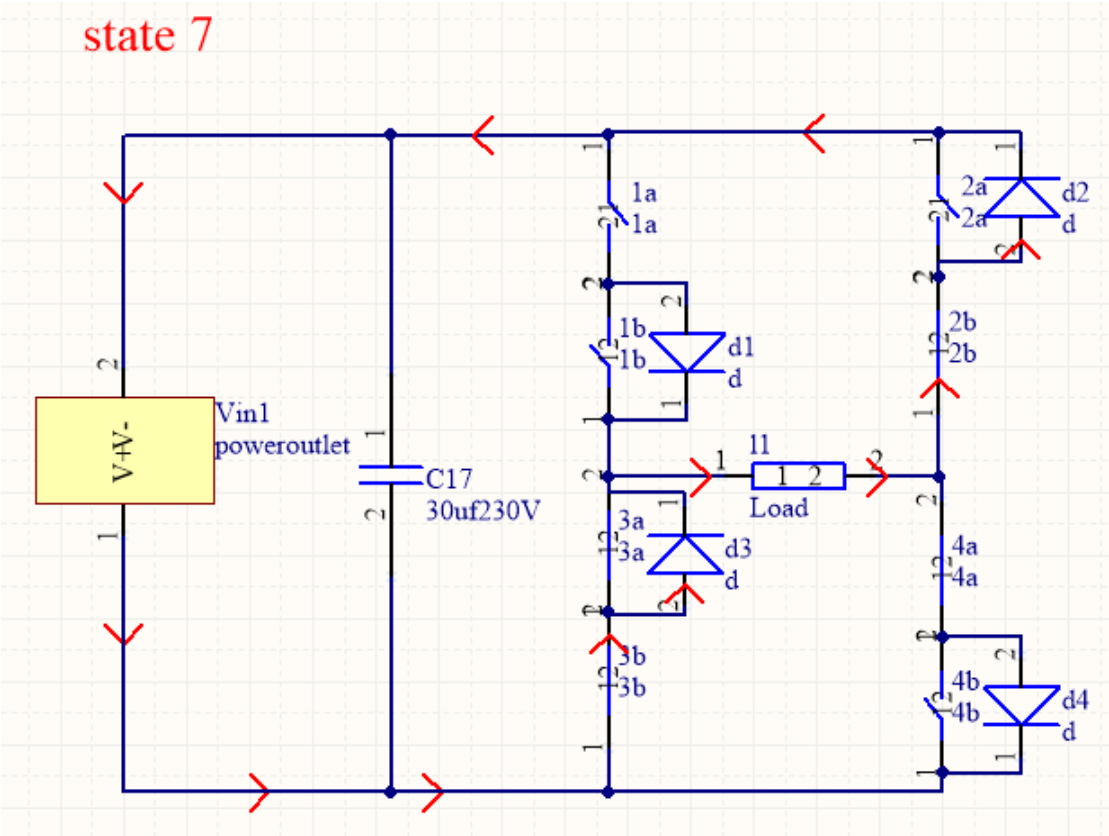

Figure 2.12 State Seven

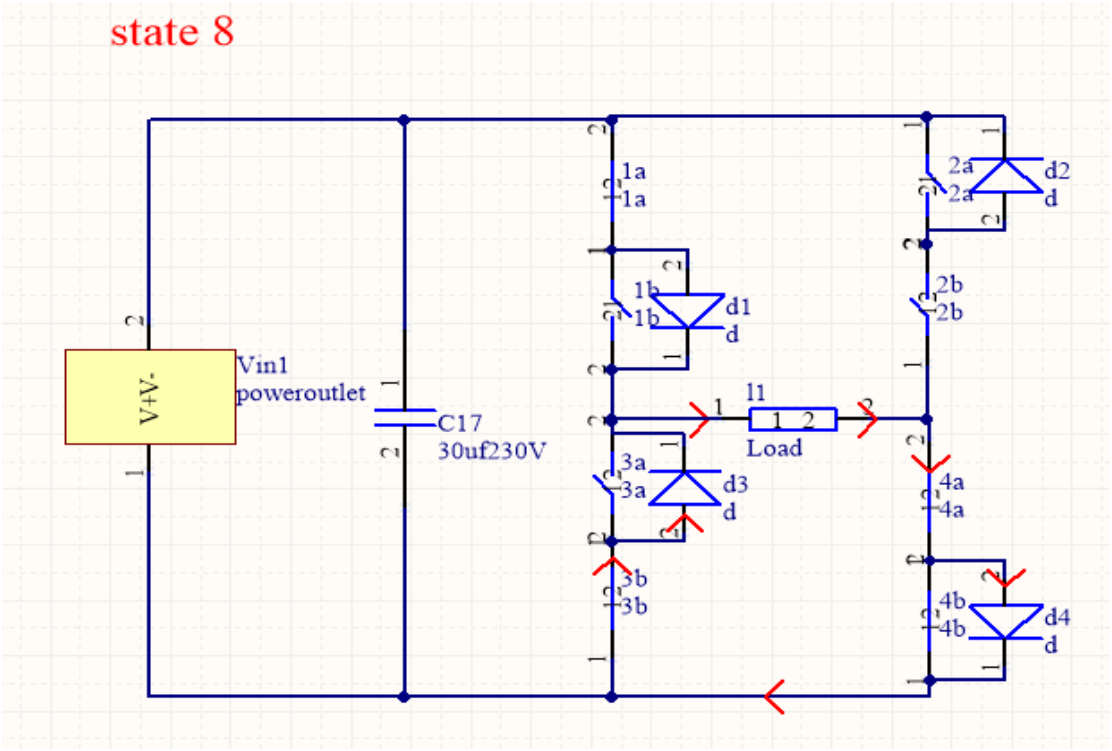

Figure 2.13 State Eight

In state nine shown on Figure 2.14, the positive current keeps flowing through switch 1a and 4a, and switch 1b is turned on for the next state.

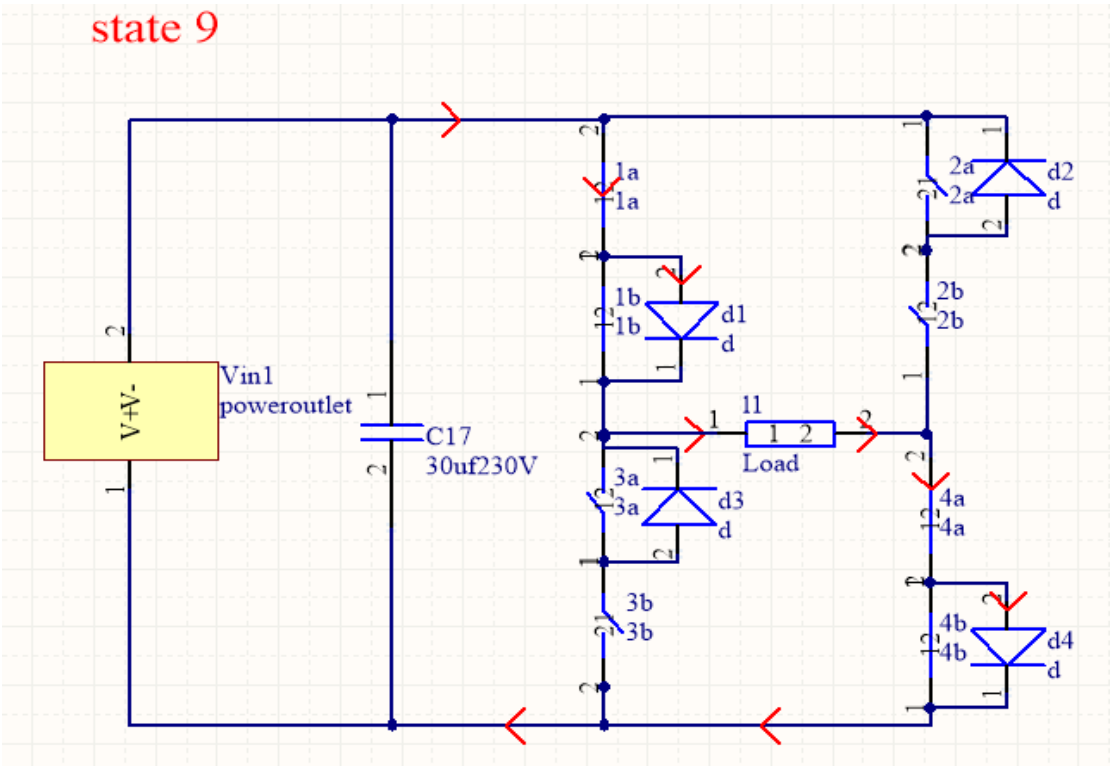

Figure 2.14 State Nine

In state ten shown on Figure 2.15, after the positive current dies out, the negative current begins to flow. In this state negative Volts/Hz is applied to the load and switch 2a is turned on to provide a current path for the next state.

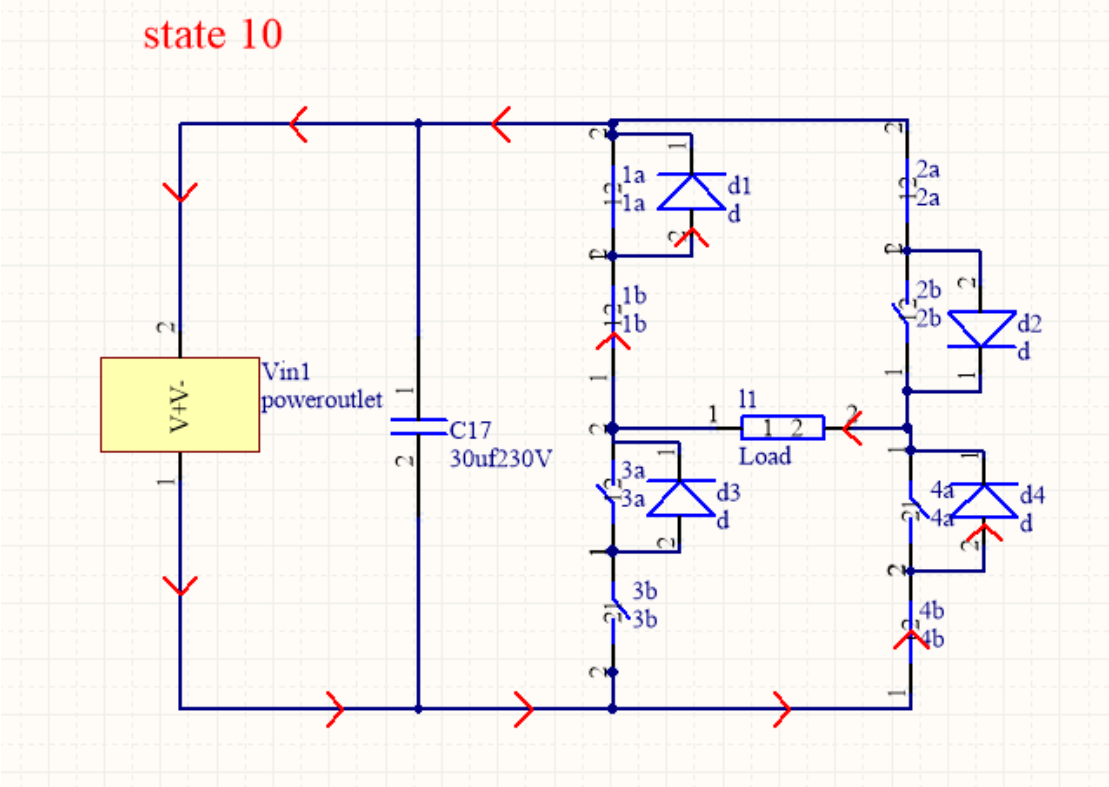

Figure 2.15 State Ten

In state eleven shown in Figure 2.16, the load is in zero state again. The negative current continues flowing through switch 1b and 2a. Switch 3a is turned on to provide a path for current flow in the next state.

In state twelve shown in Figure 2.17, even though the current keeps flowing in the negative direction, in this state the positive Volts/Hz is applied to the load, and switch 3b is turned on to provide a path for the current flow for the next state.

If the input voltage is still in negative polarity after state twelve, the next state will be state seven again. If the input voltage is changing from negative polarity into the positive polarity, the next state is shown in Figure 2.18.

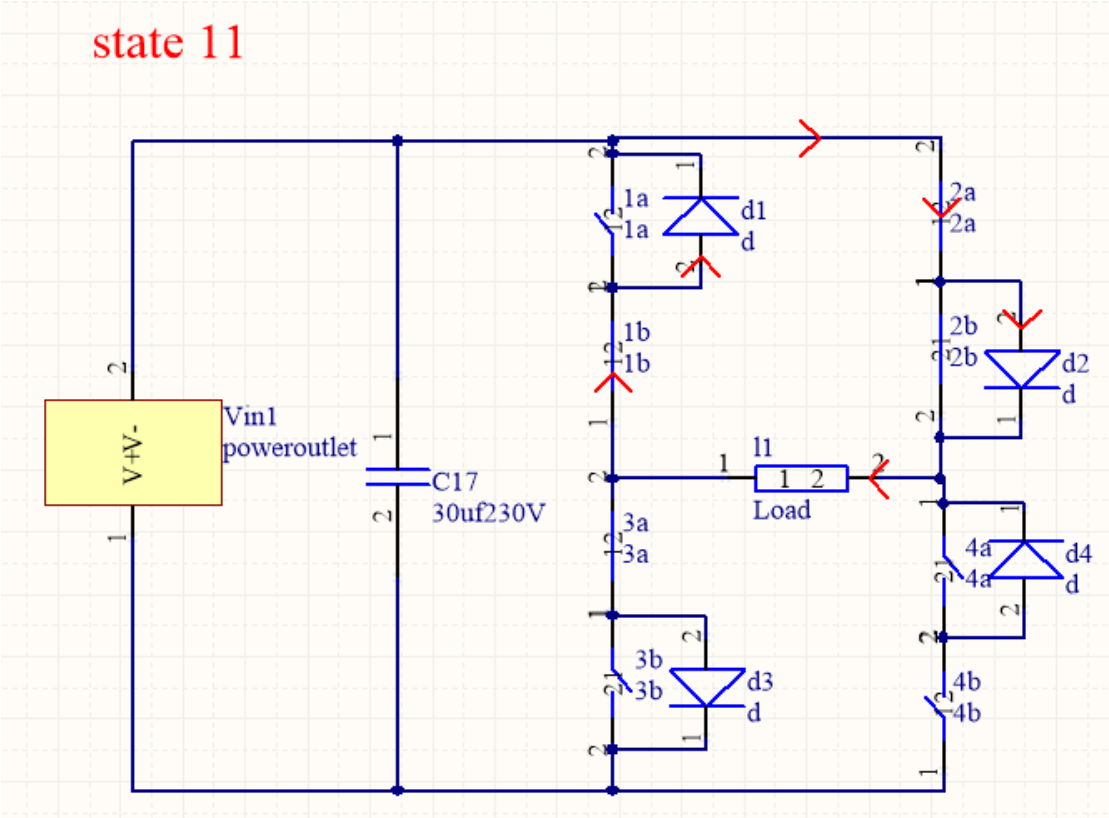

Figure 2.16 State Eleven

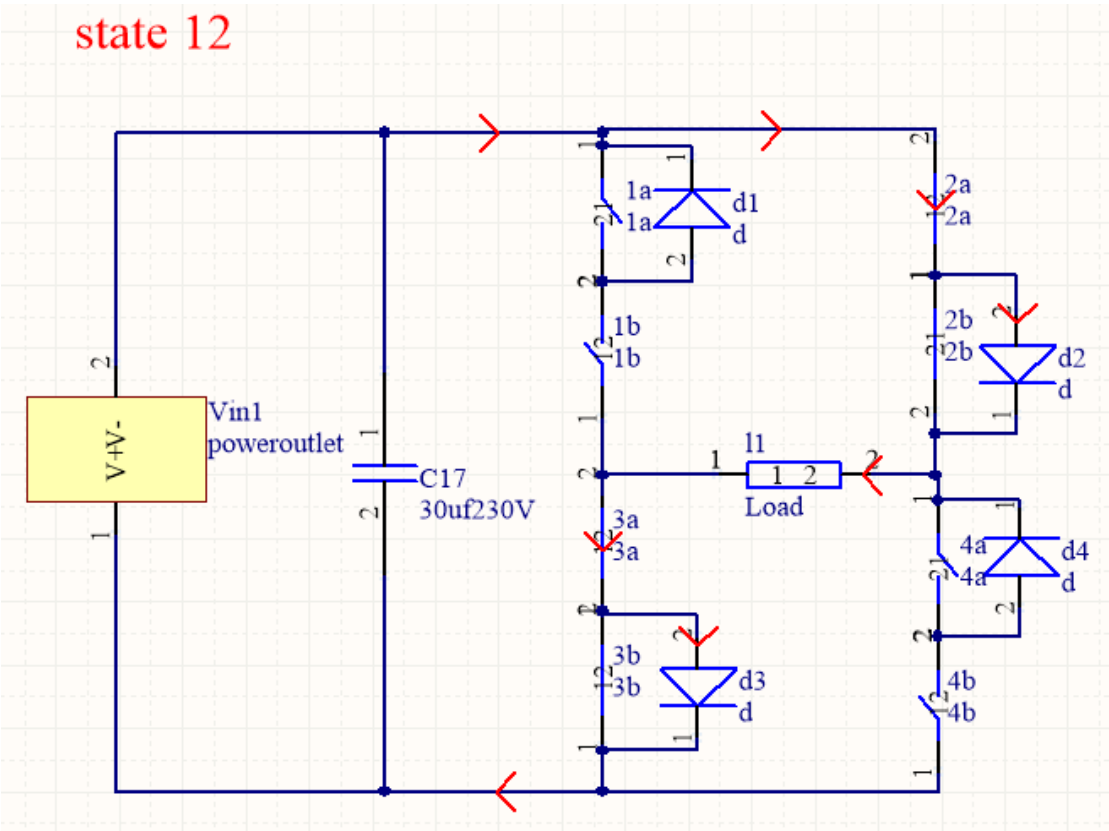

Figure 2.17 State Twelve

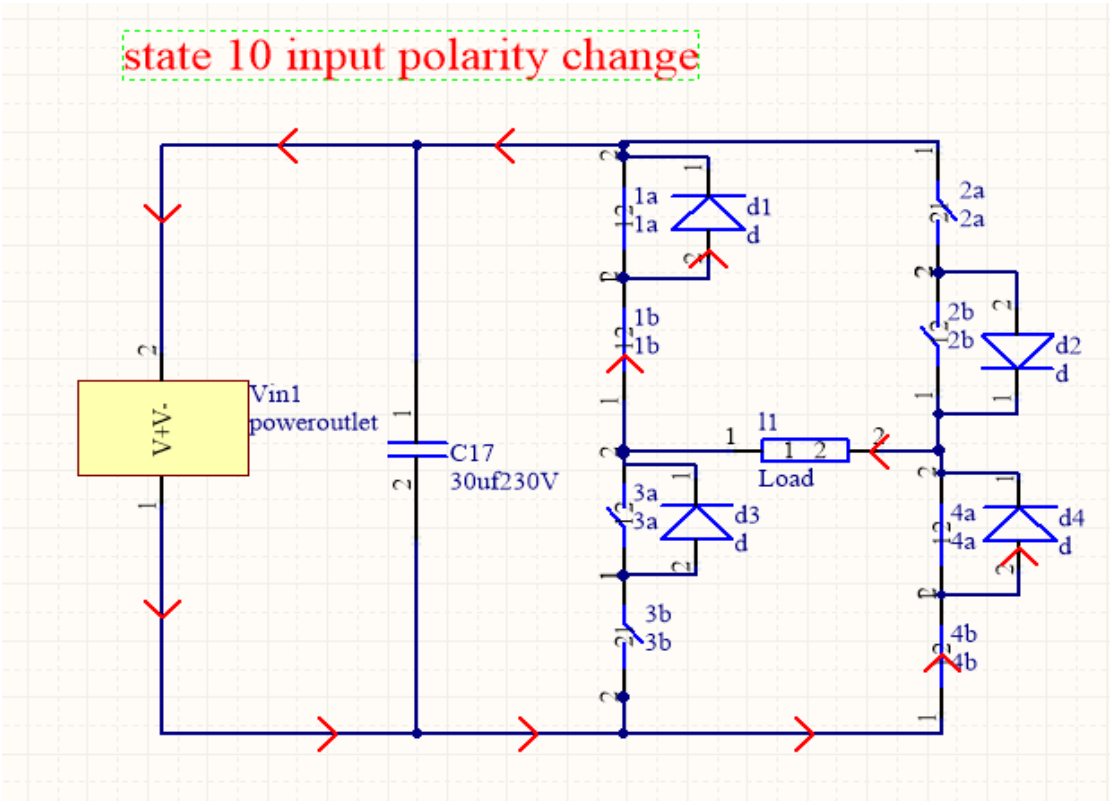

Figure 2.18 Modified State Ten

In Figure 2.18, it shows the transient state when input voltage is changing from negative polarity into positive polarity. The current is still flowing in the negative direction on the load, but switch 1a and 4a is turned on to provide a path for the current to flow when this negative current dies out.

The switching truth table for the Single Phase Matrix Converters is shown in Table 2.1. It could be concluded from the table that during each switching cycle, besides the two switches that will conduct the current at that state, there is also another switch turned on in order to conduct current for the next state. Thus, safe commutation is achieved by this new commutation method, since in every state and its next state there is always a path for the current to flow. The switches conduct current in each state is bolded in the table.

Table 2.1 Truth Table for Each Switching State

Input Polarity	State	Switch On	Switch off
Positive	1	**1a**,**4a**,1b,2b	2a,3a,3b,4b
	2	**1a**,2a,**2b**,3b	3a,4a,1b,4b
	3	2a,3a,**2b**,**3b**	1a,4a,1b,4b
	4	**2a**,**3a**,3b,4b	1a,4a,1b,2b
	5	**3a**,4a,1b,**4b**	1a,2a,2b,3b
	6	1a,4a,**1b**,**4b**	2a,3a,2b,3b
	If the Input Voltage is still Positive, Next state is 1 again		
Negative	7	3a,4a,**2b**,**3b**	1a,2a,1b,4b
	8	1a,**4a**,**3b**,4b	2a,3a,1b,2b
	9	**1a**,**4a**,1b,4b	2a,3a,2b,3b
	10	1a,2a,**1b**,**4b**	3a,4a,2b,3b
	11	**2a**,3a,**1b**,2b	1a,4a,3b,4b
	12	**2a**,**3a**,2b,3b	1a,4a,1b,4b
	If the Input Voltage is still Negative, Next state is 7 again		
Positive to Negative	13	**2a**,**3a**,2b,3b	1a,4a,1b,4b
	7	3a,4a,**2b**,**3b**	1a,2a,1b,4b
Negative to Positive	14	1a,4a,**1b**,**4b**	2a,3a,2b,3b
	1	**1a**,**4a**,1b,2b	2a,3a,3b,4b

Chapter 3

Software Simulation for SMPC

3.1 Software

The software used for simulation of the Single Phase Matrix is Powersim as this software is compact and fast in simulation. More importantly, Powersim focuses on simulation for power electronics and electric machines, so that comparing to other simulation software it is optimized, and the result for the simulation is easier to converge.

3.2 Signal Generation

The most important part for this project's simulation is to generate the correct gate signal for each switch. Since in this Single Phase Matrix Converter the modulation method is neither SPWM nor SVPWM, so existing models cannot be applied to this simulation directly. In order to simulate the model, a block called piece wise linear voltage source in Powersim can be used.

By using the piece wise linear voltage source, the PWM signal for the four bidirectional switches are shown in Figure 3.1. Because all the switch are set to be ideal for simulation, so that no deadtime is added this time. However, in experiment prototype the deadtime has to be added to PWM signal in order to avoid short circuit

for the power supply, since the switch will not be turned on or off simultaneously when positive or negative voltage signal are applied to the gate terminal.

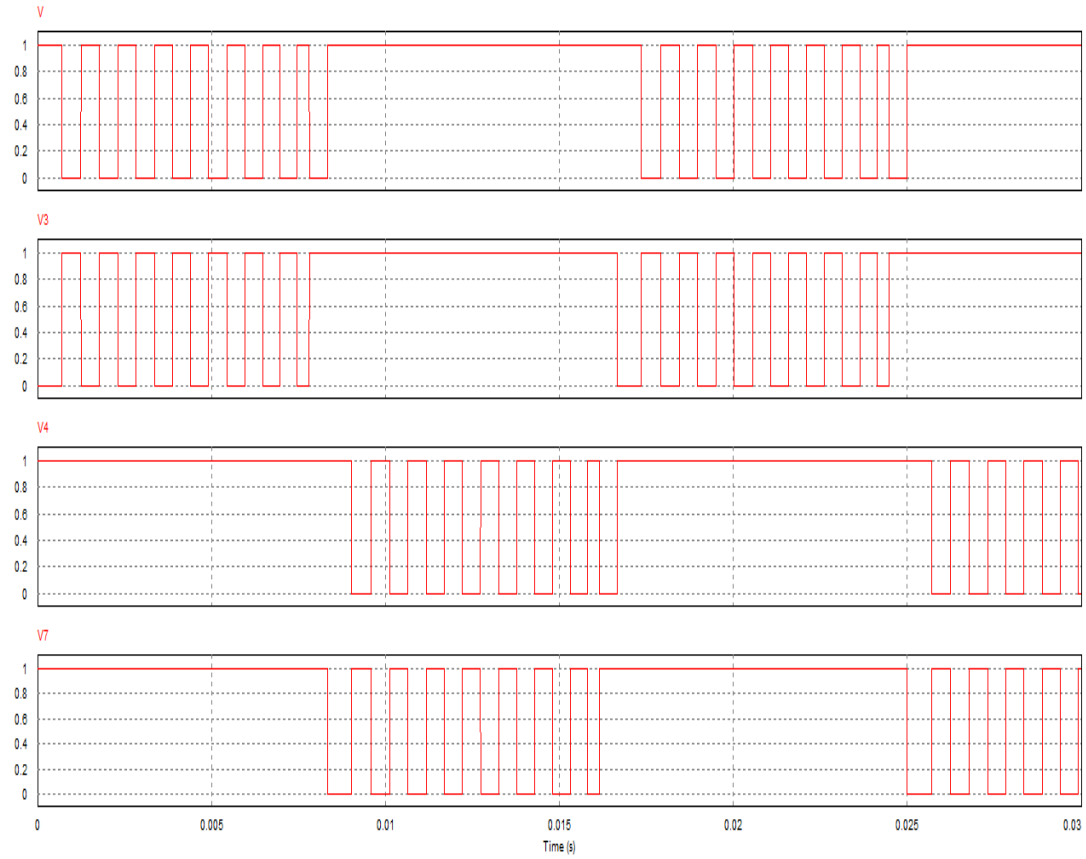

Figure 3.1 PWM Signal for Switches

3.3 Output Voltage and Current Waveform

In ideal condition, where there are no parasitic inductance and capacitance, there are no commutation noise for the circuit. The input voltage and output voltage and current waveform are showed in Figure 3.2 and 3.3.

The load for this circuit is field winding of the Wound Field Synchronous Machine, which is a very large inductive component in series with a small resistive component shown in Figure 3.4. The higher order harmonics in the output voltage can be filtered by the large inductor, thus the transformer output current waveform is a 960 Hz sine

wave with lower THD shows in Figure 3.3. More importantly, since the large inductor component exists in Field Winding, the field current is a pure DC current with very small Ripple shows in Figure 3.3.

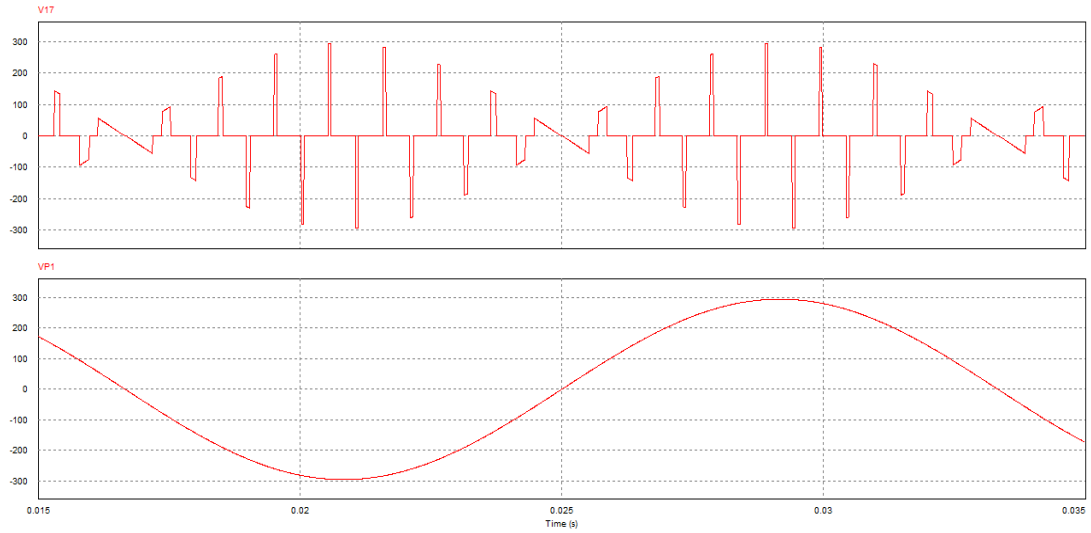

Figure 3.2 Output Voltage and Input Voltage Waveform

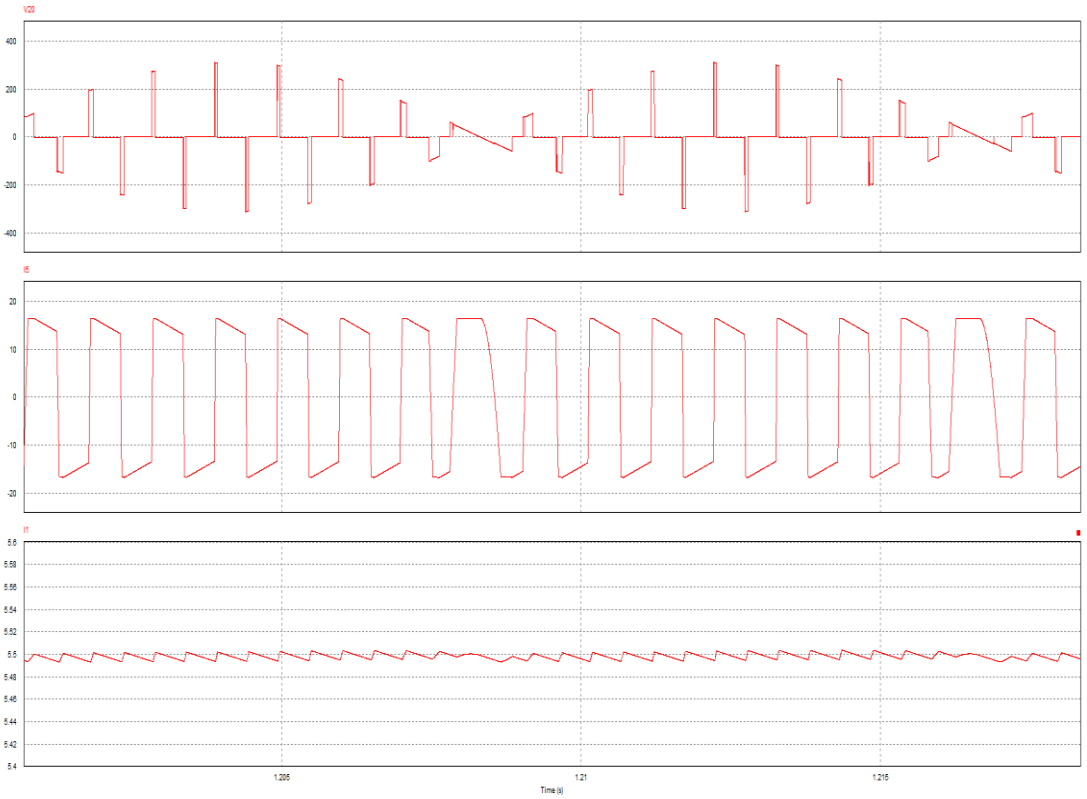

Figure 3.3 Output Voltage and Output Current on Transformer Side and DC Field Current after Diode Bridge

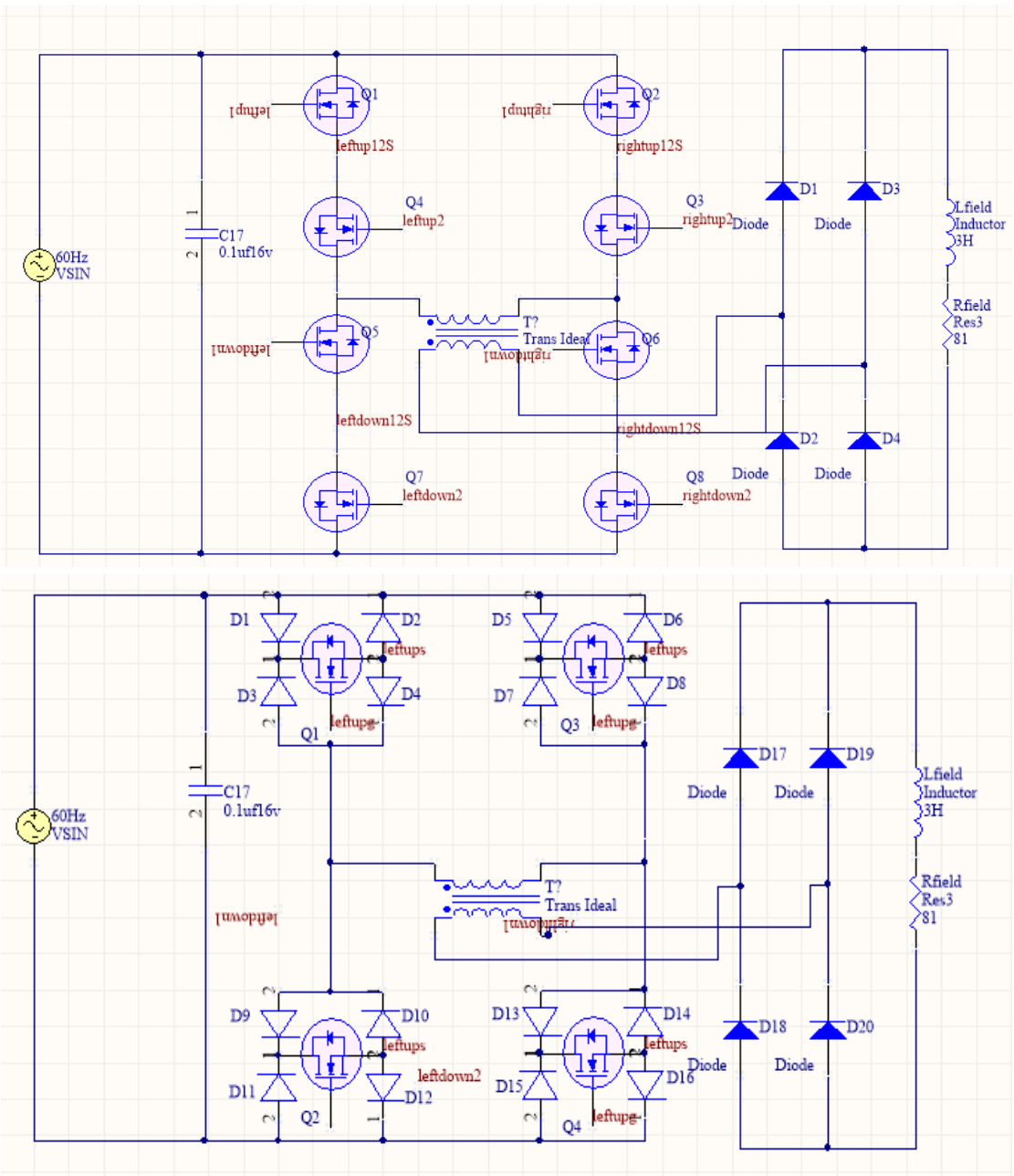

Figure 3.4 Two Different Topologies with Load Circuit Included

Chapter 4

Experiment Set Up and Prototype Experiment Result

4.1 DSP Setup

The micro controller used in this research project is Texas Instrument's TMS28335. An experiment kit is used in this prototype to generate gate signals for each switch and also provide control and synchronization for the whole system. This DSP has 18 separate channels to generate PWM waveform and also has 16 Channels of Analog to Digital conversion.

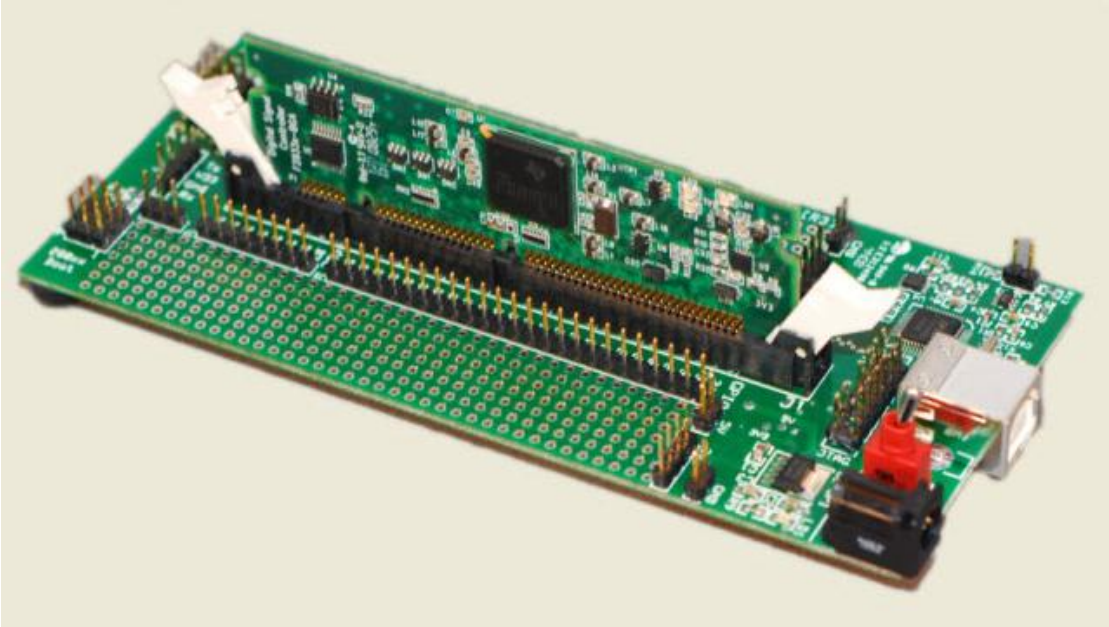

Figure 4.1 TMS320F28335 Experimenter Kit

Figure 4.2 shows the PWM gate signal generated for the MOSFETS. Within the graph, it shows each MOSFET turns in and turns off time. The deadtime for each

switching pair is also included in this driving signal for the four switch models, since there will be short circuit if both switches on one leg are turned on simultaneously. Also, there must be one and only one switch turns on at each moment to avoid an open circuit for the output due to the big inductive load.

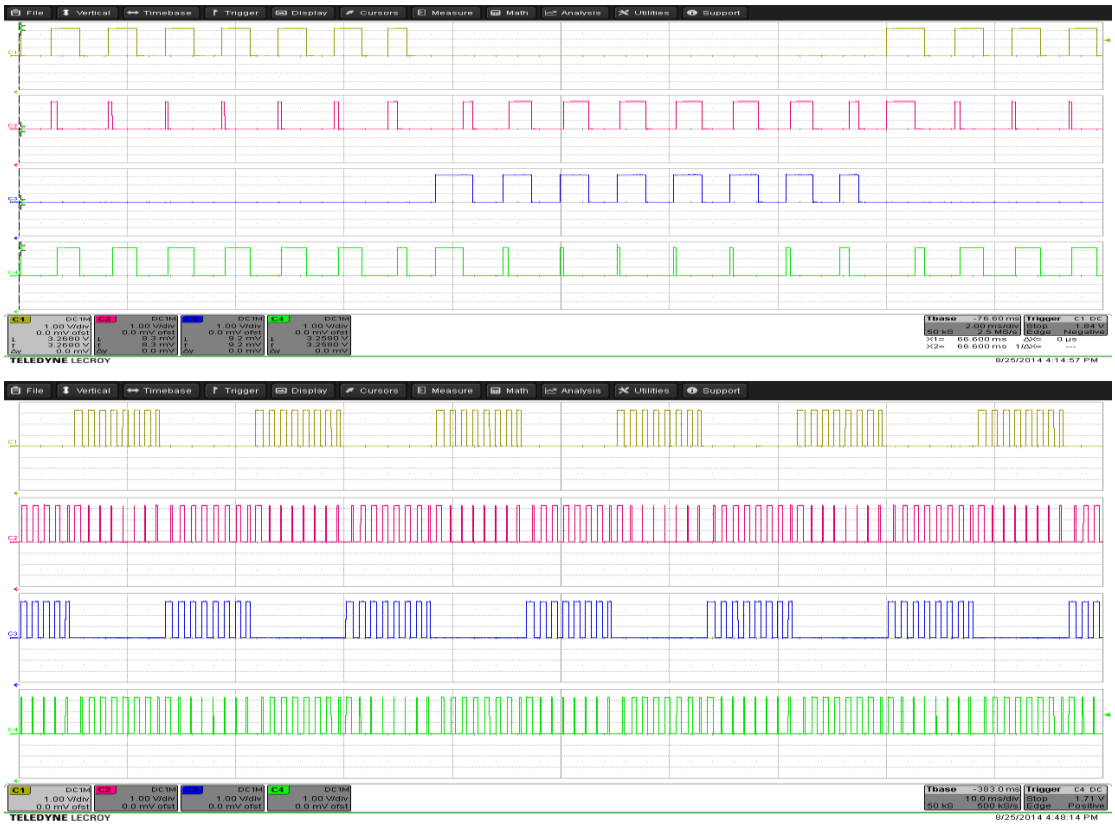

Figure 4.2 DSP Output Gate Signal

4.2 Phase Detecting Circuit

One big issue that makes Matrix Converter different from inverter is the input voltage source. For AC input voltage source, the switches cannot switch begin its sequence at any instance time like they act in an inverter since the phase must be known before switching. Thus, a phase detecting circuit must be built in order to find the zero crossing point for the AC voltage. The topology for this phase detecting circuit is shown in Figure 4.3.

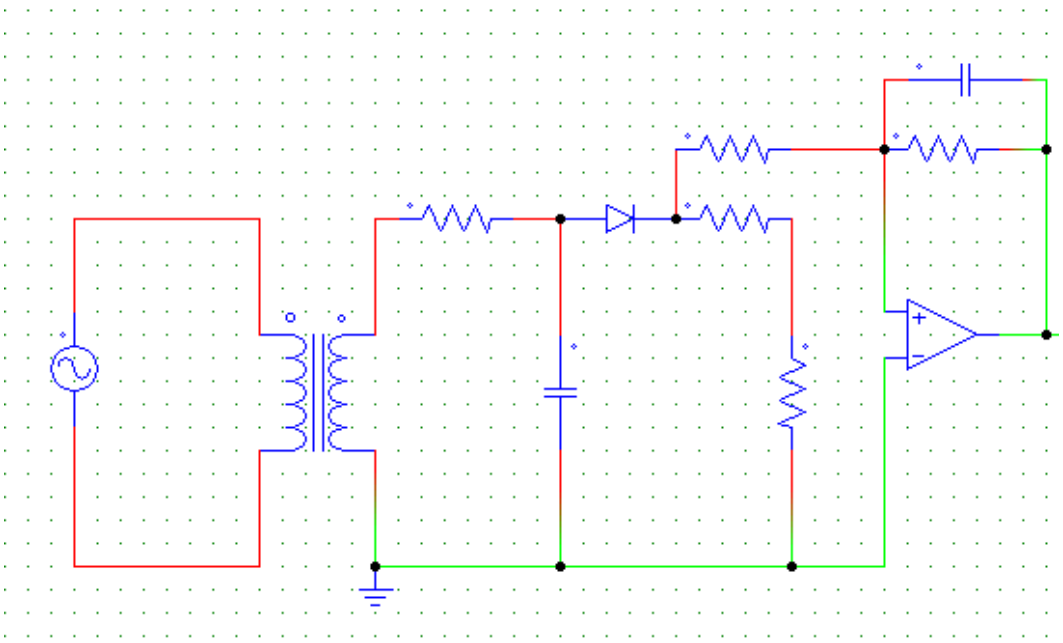

Figure 4.3 Phase Detecting Circuit

For this circuit, a signal transformer is applied to the power source in order to step down the 230 Volts input Voltage and isolate the ground. Since the input current is discontinuous as well as the commutation between switches may cause unwanted voltage and current spike during fault condition, the line AC voltage may suffer a large distortion shown in Figure 4.4.

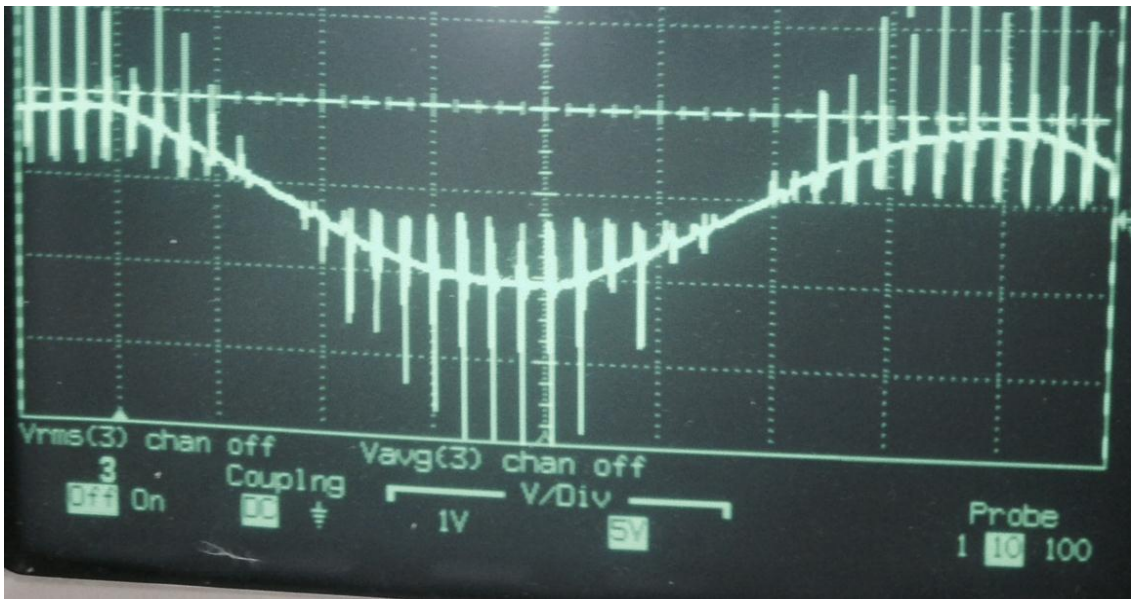

Figure 4.4 Input AC Voltage Distortion

Figure 4.5 shows the filtered voltage waveform after the low pass filter and the diode on the secondary side of the transformer. It can be observed from the figure that comparing to the unfiltered input AC voltage, a much cleaner wave form has been obtained. This filtered voltage is then sent to the comparator to generate a 60 Hz square wave shown in Figure 4.6. This square wave can trigger the external interrupt for DSP 28335 to synchronize the PWM signal by their rising edge.

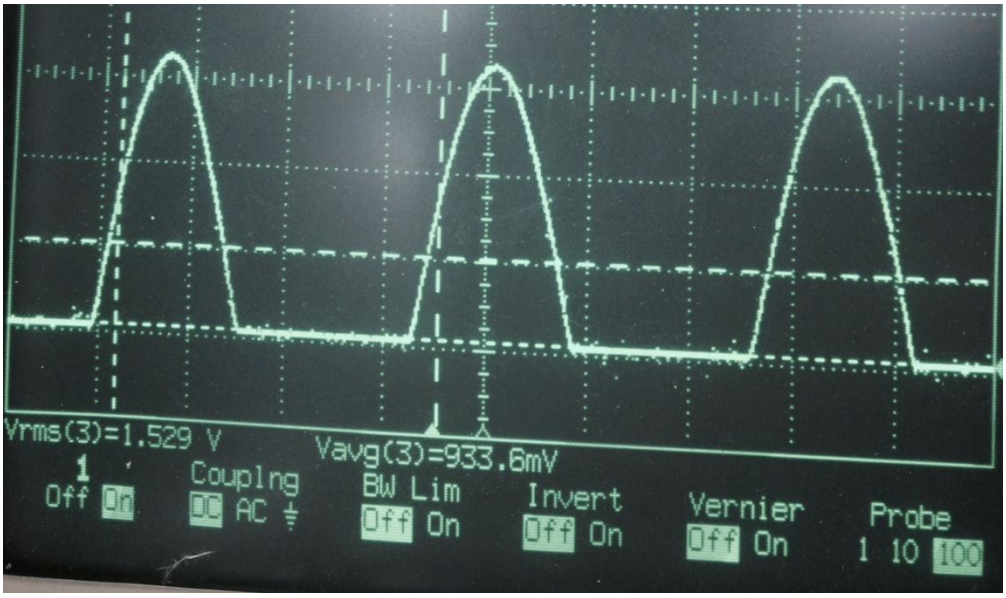

Figure 4.5 Filtered AC Waveform

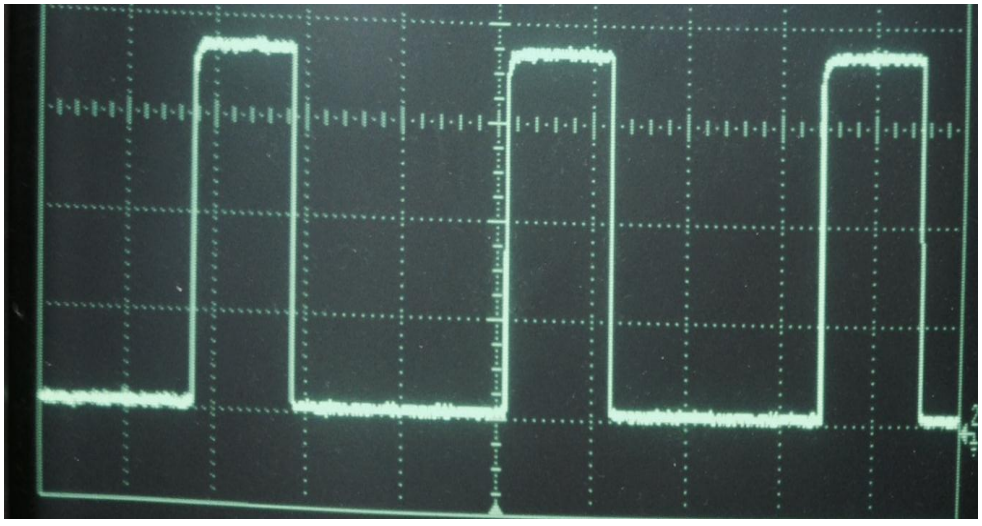

Figure 4.6 Output Digital Signal from Comparator

4.3 Main Power Circuit

4.3.1 Input Line Filter

Although one of the advantages of using matrix converter instead of inverter is the lack of bulky DC converter, an input line filter has to be added to the system since the discontinuous of the input current and the high frequency component from the output will distort the power source. The input voltage waveform without connecting the input line filter is showed in Figure 4.4. Thus, for real application the input line filter must be integrated with the whole system to prevent such distortion.

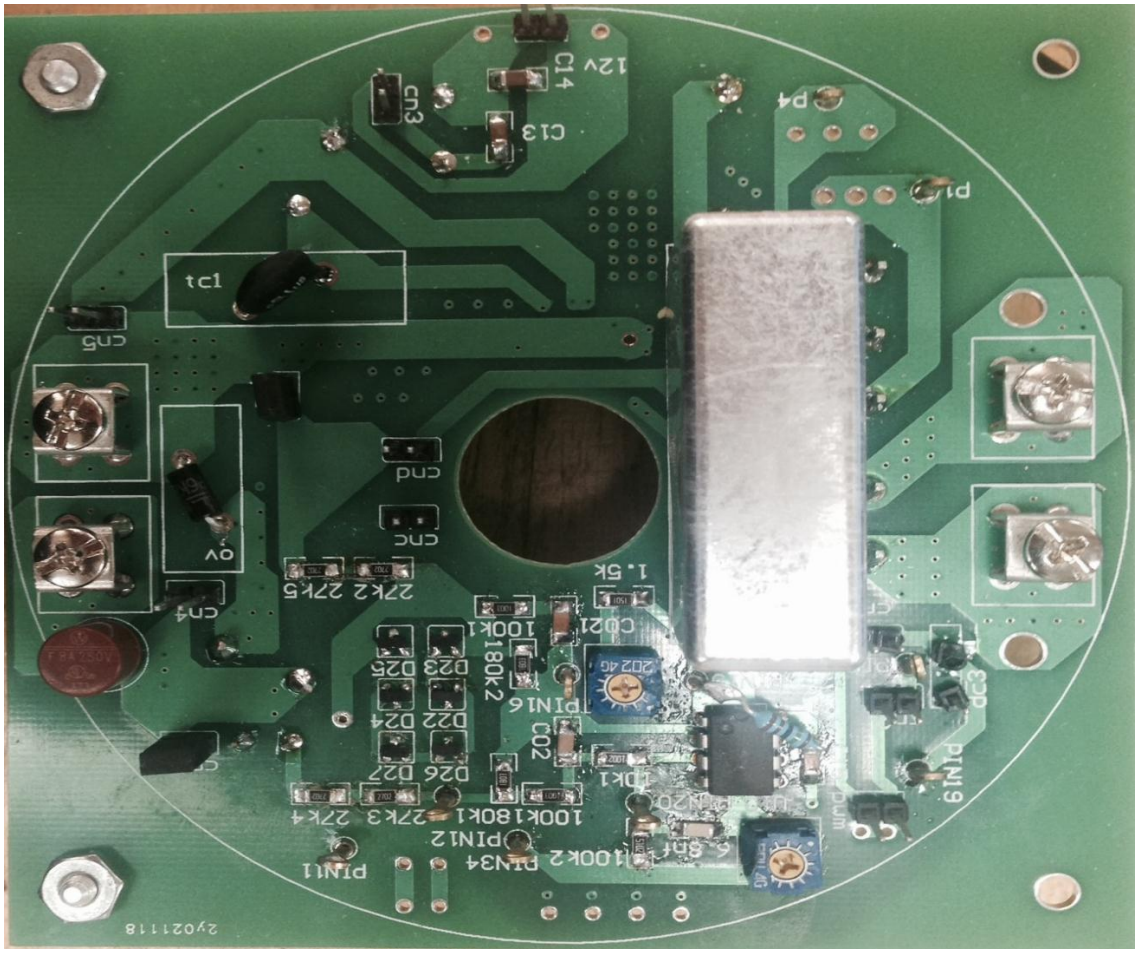

Figure 4.7 Input Line Filter

The inductor value is chosen to be 330 micro Henries, and the capacitor value is chosen to be 30 micro Farads. This creates a resonant frequency of 1600Hz, which is

lower than the switching frequency and much higher than the input frequency. The damping resistor are chosen to be 4 ohms, which is chosen from$\sqrt{L/C}$. The overall input line filter is showed in Figure 4.7.

4.3.2 Clamping and Snubber Circuit

For clamping circuit, the diodes bridge needs to response fast to the current. So fast silicon carbide diodes are chosen to build the rectifier bridge. The value for the charging capacitor is 30 micro Farads. The discharge resistor has a value of 100k ohms in order to dissipate the energy stored in the charging capacitor.

A 0.1 micro Farads is also connected in parallel with each MOSFET to prevent any voltage and current spike to damage the switching unit.

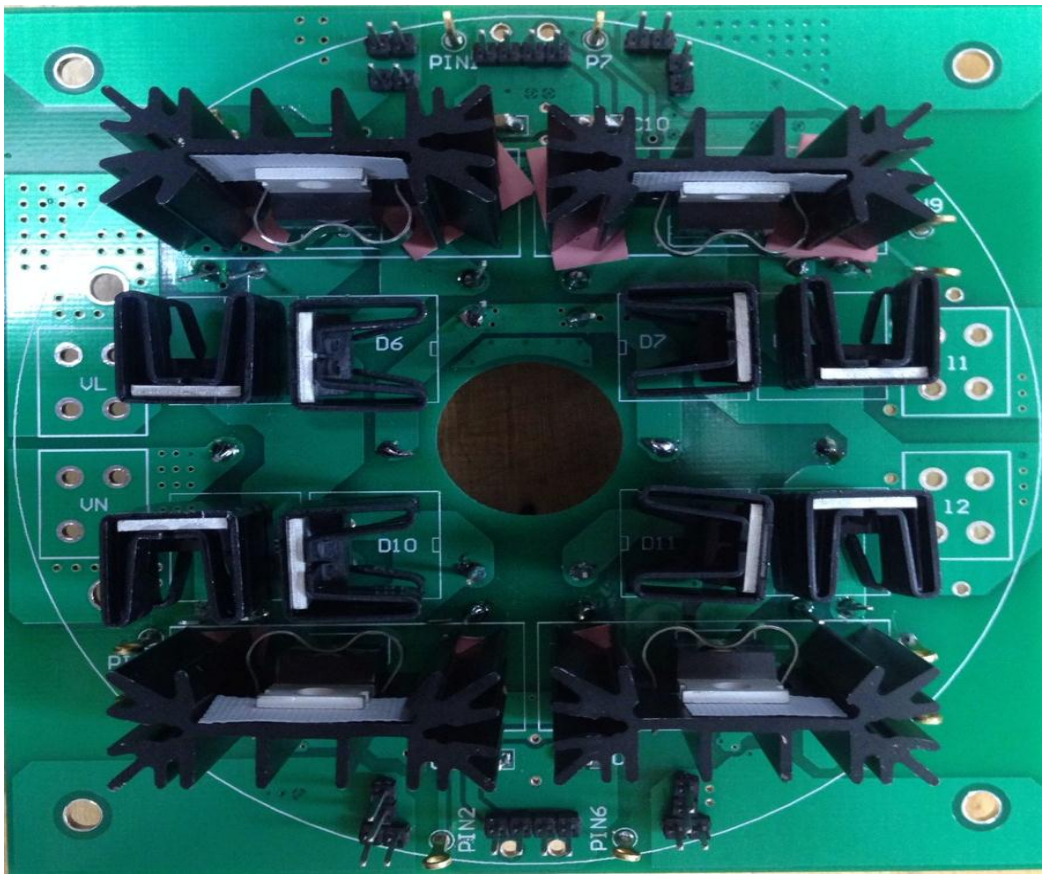

Figure 4.8 Front Side of the Power Circuit for Four Switch Model

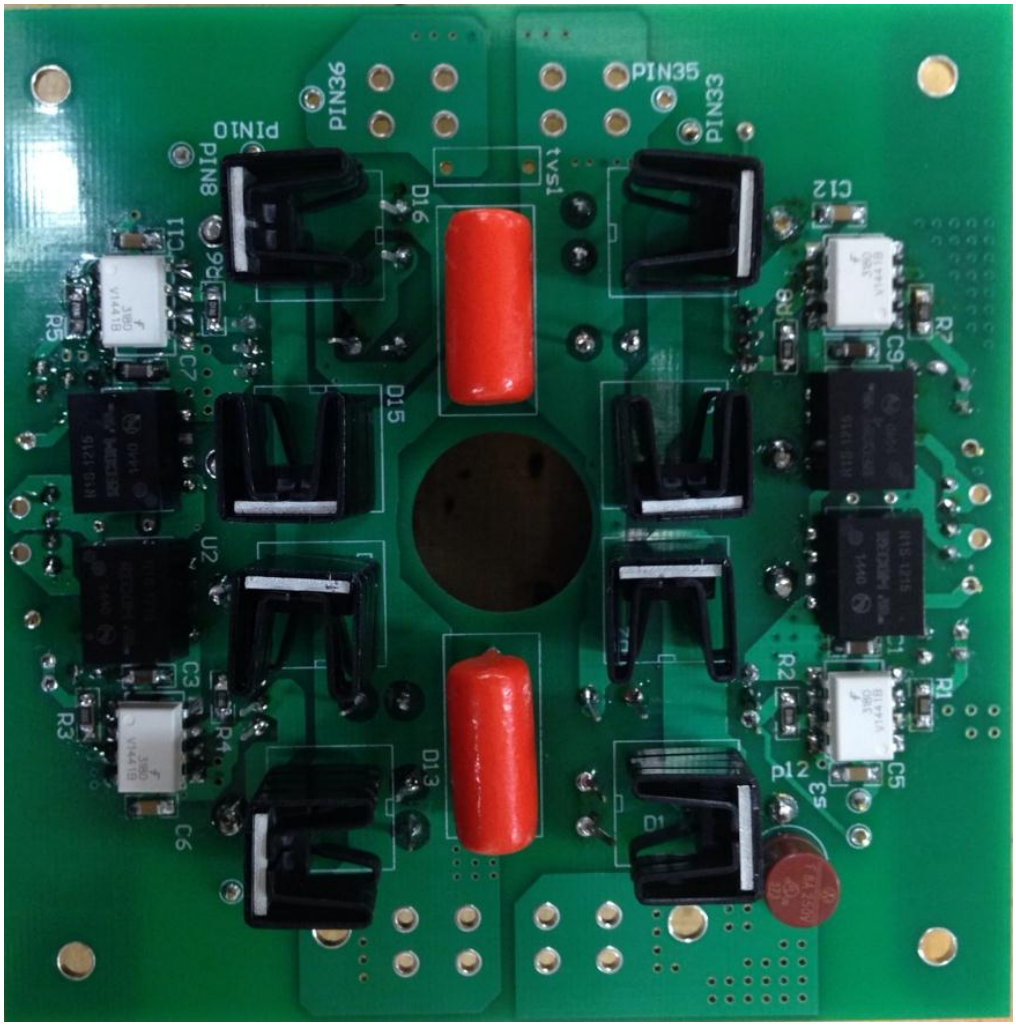

Figure 4.9 Back Side of the Power Circuit for Four Switch Model

4.3.3 Main Power Circuit for Four Switch Model

Figure 4.8 and Figure 4.9 show the front and back sides of the power circuit for the four switch model. Silicon Carbide diodes are chosen for the diodes bridge circuit applied to the MOSFET since they have to conduct high frequency current.

4.3.4 Main Power Circuit for the Eight Switch Model

Figure 4.10 shows the power circuit for the eight switch model. It can be observed from the hardware that the bi-directional switch unit are consisted of two separate MOSFETs by back to back connection. The PCB for the eight switch model is larger than the one for four switch model since the number of the gate drive circuit is doubled.

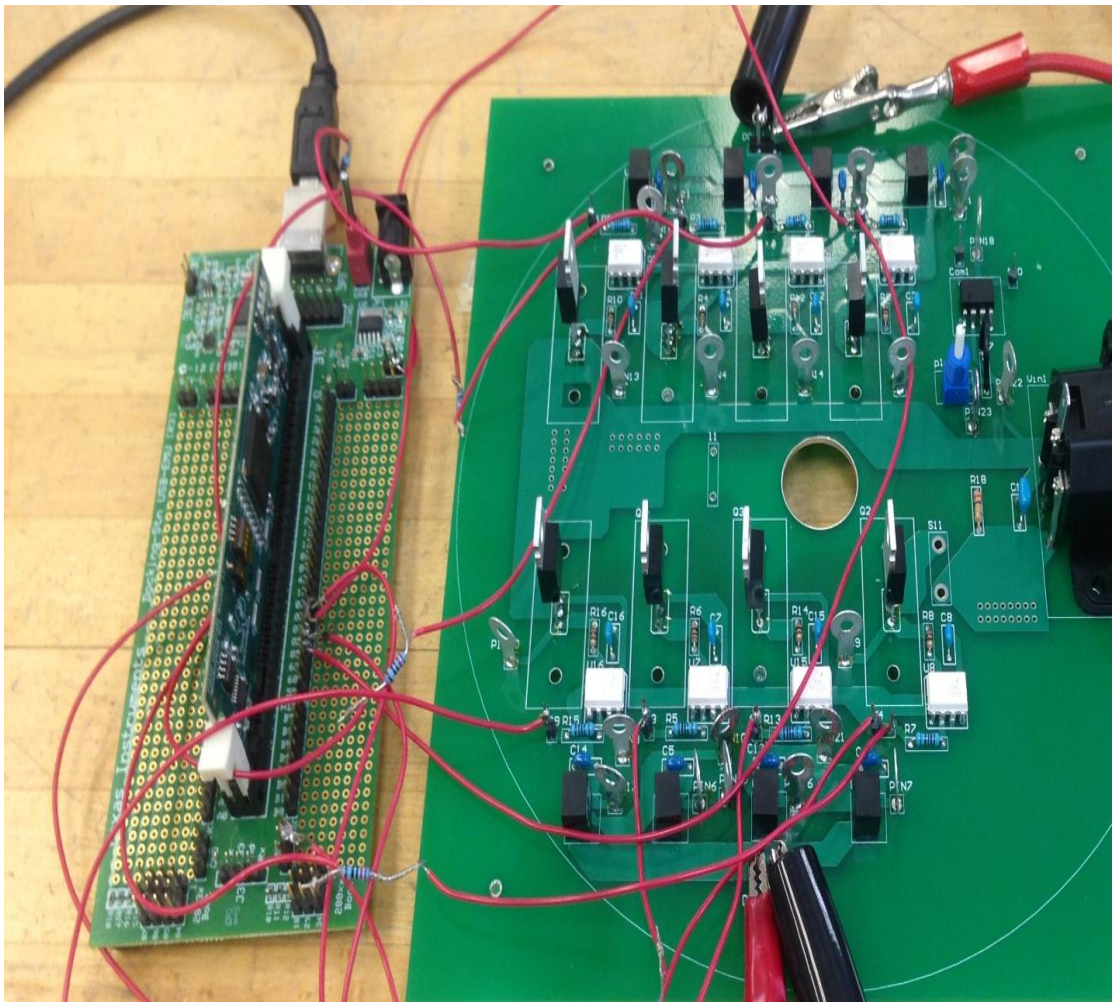

Figure 4.10 Power Circuit for Eight Switch Model

4.4 Experiment Result

Figure 4.11 and Figure 4.12 show the output voltage waveform for the eight switch model and the four switch model. It can be observed from the waveform that for the four switch model the voltage drop is higher, which is caused by the additional diodes bridge connecting to the MOSFET. More importantly, the four switch model has a much larger voltage spike than the eight switch model when switches are turned on and off. In four switch model, there is only one switch can conduct the current so that for each leg of the bridge, there must be deadtime to avoid the short circuit for the input power

supply. However, when large inductive load are applied to the circuit, this could cause a large voltage spike to turn off the current which may damage the switch and other component in the circuit.

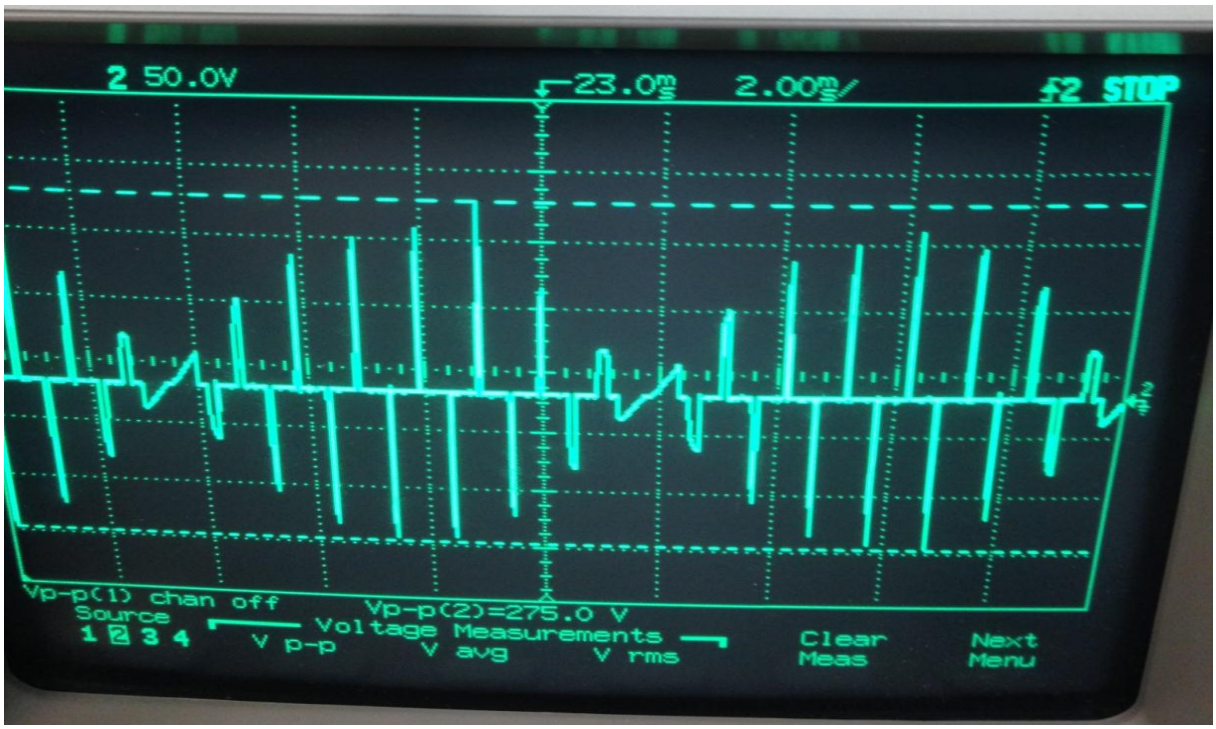

Figure 4.11 Output Voltage Waveform for Eight Switch Model

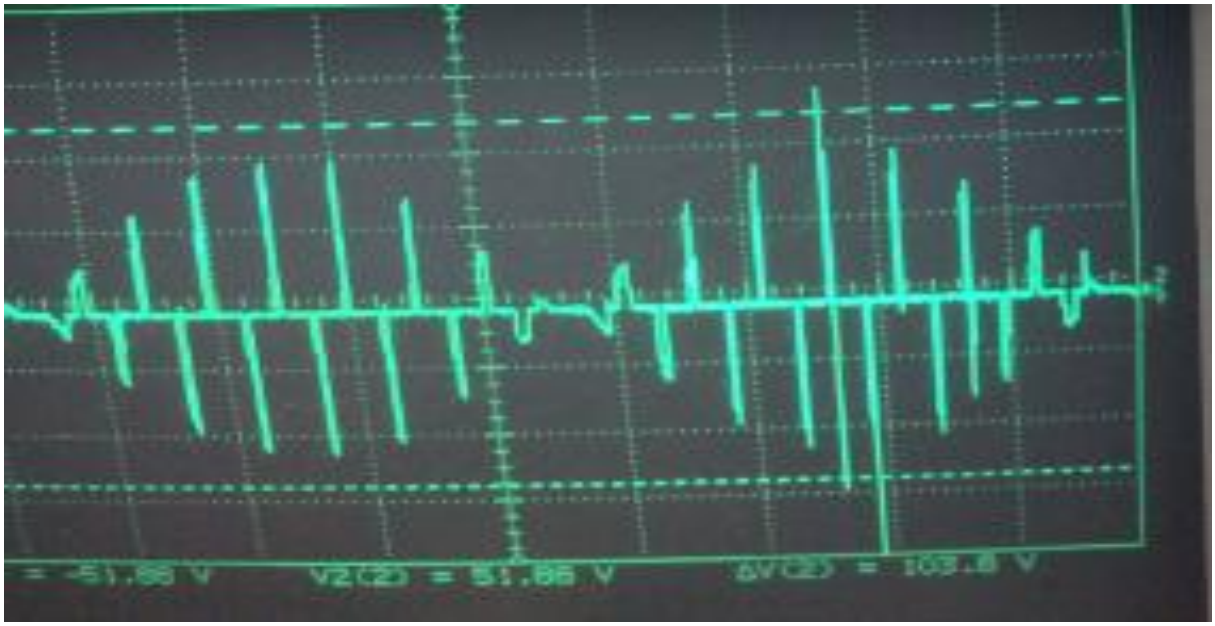

Figure 4.12 Output Voltage Waveform for Four Switch Model

However, for the eight switch model, the current can flow from the other MOSFET

in the switching unit when one of the MOSFET is turned off. This will not only reduce the level of the voltage spike, but also protect the component from the voltage spike.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Two different topologies of Single Phase Matrix Converters are designed and successfully implemented in this research project. According to the result from the experiment prototype, the experiment output voltage waveform matches the simulation output voltage waveform. For the four switch model, there are some limitations based on the load type since a large inductive load may cause large spike during commutation stage, which will damage the MOSFET and other devices. For eight switch model, the loss is less and there will be no commutation problem. However, the controllable switch number is doubled and the gate drive circuit for the switch is also doubled.

5.2 Future Work

Future work will be concentrated on integrating the whole system with the stating transformer and the Wound Field Synchronous Machine. Also, a current control loop will be added to the Matrix Converter.

List of References

[1] L. Gyugyi and B. R. Pelly, *Static power frequency changers : theory, performance, and application*. New York: Wiley, 1976.

[2] A. Zuckerberger, D. Weinstock, and A. Alexandrovitz, "Single-phase matrix converter," *Iee Proceedings-Electric Power Applications*, vol. 144, pp. 235-240, Jul 1997.

[3] L. X. Wei and T. A. Lipo, "Investigation of 9-switch dual-bridge matrix converter operating under low output power factor," *2003 Ieee Industry Applications Conference, Vols 1-3*, pp. 176-181, 2003.

[4] T. Matsuo, S. Bernet, R. S. Colby, and T. A. Lipo, "Application of the matrix converter to induction motor drives," *Ias '96 - Conference Record of the 1996 Ieee Industry Applications Conference, Thirty-First Ias Annual Meeting, Vols 1-4*, pp. 60-67, 1996.

[5] Z. Idris, M. K. Hamzah, and M. F. Saidon, "Implementation of Single-Phase Matrix Converter as a direct AC-AC converter with commutation strategies," *2006 IEEE Power Electronics Specialists Conference, Vols 1-7*, pp. 1244-1250, 2006.

[6] N. Nguyen-Quang, D. A. Stone, C. M. Bingham, and M. P. Foster, "Comparison of single-phase matrix converter and H-bridge converter for radio frequency induction heating," *2007 European Conference on Power Electronics and Applications, Vols 1-10*, pp. 94-102, 2007.

[7] P. N. A. M. Yunus and M. K. Hamzah, "Safe Commutation for an AC-DC Single Phase Matrix Converter (SPMC) with Combination of LC Filter and Damping Resistor," *2012 Ieee International Conference on Power and Energy (Pecon)*, pp. 570-575, 2012.

[8] D. Barater, G. Buticchi, C. Concari, L. Concari, and G. Franceschini, "Single-Phase Matrix Converter for Active Power Filter Applications," *39th Annual Conference of the Ieee Industrial Electronics Society (Iecon 2013)*, pp. 8528-8533, 2013.

[9] P. Drabek and M. Pittermann, "Novel Primary High Voltage Traction Converter with Single-Phase Matrix Converter," *2009 Ieee Vehicle Power and Propulsion Conference, Vols 1-3*, pp. 1245-1248, 2009.

[10] S. Sunter and O. Aydogmus, "Implementation of a single-phase matrix converter induction motor drive," *Electrical Engineering*, vol. 90, pp. 425-433, Jun 2008.

[11] M. K. Hamzah, Z. Idris, A. Saparon, and M. S. Yunos, "FPGA Design of Single-phase Matrix Converter Operating as a Frequency Changer," *2008 Ieee 2nd International Power and Energy Conference: Pecon, Vols 1-3*, pp. 1124-1129, 2008.

[12] M. S. Hapeez, N. R. Hamzah, and M. K. Hamzah, "Comparison of the Experimental Results of a Newly Developed Interfacing Method on a Single-

Phase Matrix Converter Employing Safe Commutation Strategy," *2009 Ieee Student Conference on Research and Development: Scored 2009, Proceedings,* pp. 411-414, 2009.

[13] H. J. Cha and P. N. Enjeti, "A new ride-through approach for matrix converter fed adjustable speed drives," *Conference Record of the 2002 Ieee Industry Applications Conference, Vols 1-4,* pp. 2555-2560, 2002.

[14] S. Ratanapanachote, H. J. Cha, and P. N. Enjeti, "A digitally controlled switch mode power supply based on matrix converter," *Ieee Transactions on Power Electronics,* vol. 21, pp. 124-130, Jan 2006.

[15] S. Angkititrakul and R. W. Erickson, "Control and implementation of a new modular matrix converter," *Apec 2004: Nineteenth Annual Ieee Applied Power Electronics Conference and Exposition, Vols 1-3,* pp. 813-819, 2004.

[16] A. Alesina and M. G. B. Venturini, "Solid-State Power Conversion - a Fourier-Analysis Approach to Generalized Transformer Synthesis," *Ieee Transactions on Circuits and Systems,* vol. 28, pp. 319-330, 1981.

[17] A. Schuster, "A matrix converter without reactive clamp elements for an induction motor drive system," *Pesc 98 Record - 29th Annual Ieee Power Electronics Specialists Conference, Vols 1 and 2,* pp. 714-720, 1998.

[18] D. G. Holmes and T. A. Lipo, "Implementation of a Controlled Rectifier Using Ac-Ac Matrix Converter Theory," *Pesc 89 Record, Vols 1 and 2,* pp. 353-359, 1989.

[19] E. C. Aeloiza, P. N. Enjeti, O. C. Montero, and L. A. Moran, "Analysis and design of a new voltage sag compensator for critical loads in electrical power distribution systems," *Conference Record of the 2002 Ieee Industry Applications Conference, Vols 1-4,* pp. 911-916, 2002.

[20] A. V. Stankovic and T. A. Lipo, "A novel control method for input-output harmonic elimination of the PWM boost type rectifier under unbalanced operating conditions (vol 16, pg 603, 2001)," *Ieee Transactions on Power Electronics,* vol. 16, pp. 888-888, Nov 2001.

[21] H. M. Hanafi, M. K. Hamzah, and N. R. Hamzah, "Modeling of Electronic Transformer Design with the Implementation of Single-phase Matrix Converter Using MATLAB/Simulink," *2009 Ieee Student Conference on Research and Development: Scored 2009, Proceedings,* pp. 407-410, 2009.

[22] C. Y. Gu, H. S. Krishnamoorthy, P. N. Enjeti, and Y. D. Li, "Medium-Voltage (MV) Matrix Converter Topology for Wind Power Conversion Using Medium-Frequency Transformer (MFT) Isolation," *2014 Twenty-Ninth Annual Ieee Applied Power Electronics Conference and Exposition (Apec),* pp. 3084-3090, 2014.

Appendices DSP 28335 Code

Appendix A: Eight Switch Model

```
//###########################################################################
//
// FILE:	Example_2833xEPwm3UpAQ.c
//
// TITLE:	Action Qualifier Module Upcount mode.
//
// ASSUMPTIONS:
//
//	This program requires the DSP2833x header files.
//
//	Monitor the ePWM1 - ePWM3 pins on a oscilloscope as
//	described below.
//
//		EPWM1A is on GPIO0
//		EPWM1B is on GPIO1
//
//		EPWM2A is on GPIO2
//		EPWM2B is on GPIO3
//
//		EPWM3A is on GPIO4
//		EPWM3B is on GPIO5
//
//	As supplied, this project is configured for "boot to SARAM"
//	operation.	The 2833x Boot Mode table is shown below.
//	For information on configuring the boot mode of an eZdsp,
//	please refer to the documentation included with the eZdsp,
//
//	   $Boot_Table:
//
//		GPIO87	GPIO86	GPIO85	GPIO84
//		 XA15	 XA14	 XA13	 XA12
//		  PU	  PU	  PU	  PU
//	   ========================================
//		  1	  1	  1	  1	Jump to Flash
//		  1	  1	  1	  0	SCI-A boot
```

```
//                1      1      0      1      SPI-A boot
//                1      1      0      0      I2C-A boot
//                1      0      1      1      eCAN-A boot
//                1      0      1      0      McBSP-A boot
//                1      0      0      1      Jump to XINTF x16
//                1      0      0      0      Jump to XINTF x32
//                0      1      1      1      Jump to OTP
//                0      1      1      0      Parallel GPIO I/O boot
//                0      1      0      1      Parallel XINTF boot
//                0      1      0      0      Jump to SARAM       <-
"boot to SARAM"
//                0      0      1      1      Branch to check boot mode
//                0      0      1      0      Boot to flash, bypass ADC
cal
//                0      0      0      1      Boot to SARAM, bypass
ADC cal
//                0      0      0      0      Boot to SCI-A, bypass
ADC cal
//                                            Boot_Table_End$
//
// DESCRIPTION:
//
//      This example configures ePWM1, ePWM2, ePWM3 to produce an
//      waveform with independant modulation on EPWMxA and
//      EPWMxB.
//
//      The compare values CMPA and CMPB are modified within the ePWM's ISR
//
//      The TB counter is in upmode for this example.
//
//      View the EPWM1A/B, EPWM2A/B and EPWM3A/B waveforms
//      via an oscilloscope
//
//
//###########################################################################
#######
// $TI Release: 2833x/2823x Header Files and Peripheral Examples V133 $
// $Release Date: June 8, 2012 $
//###########################################################################
#######
//leftup1 epwm1 rightdown1 epwm2 rightup2 epwm4 leftdown2 epwm3

#include "DSP28x_Project.h"         // Device Headerfile and Examples Include File
```

```
typedef struct
{
    volatile struct EPWM_REGS *EPwmRegHandle;
    Uint16 EPwm_CMPA_Direction;
    Uint16 EPwm_CMPB_Direction;
    Uint16 EPwmTimerIntCount;
    Uint16 EPwmMaxCMPA;
    Uint16 EPwmMinCMPA;
    Uint16 EPwmMaxCMPB;
    Uint16 EPwmMinCMPB;
    Uint16 intcnt;
}EPWM_INFO;



void InitEPwm1Example(void);
void InitEPwm2Example(void);
void InitEPwm3Example(void);
void InitEPwm4Example(void);
void InitEPwm5Example(void);
void InitEPwm6Example(void);
interrupt void epwm1_isr(void);
interrupt void epwm2_isr(void);
interrupt void epwm3_isr(void);
interrupt void epwm4_isr(void);
interrupt void epwm5_isr(void);
interrupt void epwm6_isr(void);
void update_compare1(EPWM_INFO*);
void update_compare2(EPWM_INFO*);
void update_compare3(EPWM_INFO*);
void update_compare4(EPWM_INFO*);
void update_compare5(EPWM_INFO*);
void update_compare6(EPWM_INFO*);
interrupt void xint01(void);


// Global variables used in this example
EPWM_INFO epwm1_info;
EPWM_INFO epwm2_info;
EPWM_INFO epwm3_info;
EPWM_INFO epwm4_info;
EPWM_INFO epwm5_info;
EPWM_INFO epwm6_info;
```

```
// Configure the period for each timer
//#define EPWM1_TIMER_TBPRD    39040    // Period register
//#define EPWM2_TIMER_TBPRD    39040    // Period register
//#define EPWM3_TIMER_TBPRD    39040    // Period register
//#define EPWM4_TIMER_TBPRD    39040    // Period register
//#define EPWM5_TIMER_TBPRD    39040    // Period register
//#define EPWM6_TIMER_TBPRD    39040    // Period register


void main(void)
{
// Step 1. Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the DSP2833x_SysCtrl.c file.
     InitSysCtrl();

// Step 2. Initalize GPIO:
// This example function is found in the DSP2833x_Gpio.c file and
// illustrates how to set the GPIO to it's default state.
// InitGpio();    // Skipped for this example

// For this case just init GPIO pins for ePWM1, ePWM2, ePWM3
// These functions are in the DSP2833x_EPwm.c file
     InitEPwm1Gpio();
     InitEPwm2Gpio();
     InitEPwm3Gpio();
     InitEPwm4Gpio();
     InitEPwm5Gpio();
     InitEPwm6Gpio();

     EALLOW;

        //GpioCtrlRegs.GPBPUD.bit.GPIO32 = 0;//pull up
        //GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 2;//make it syncin

        GpioCtrlRegs.GPAPUD.bit.GPIO30 = 0;
        GpioCtrlRegs.GPAMUX2.bit.GPIO30 = 0;    // GPIO30 = GPIO30
        GpioCtrlRegs.GPADIR.bit.GPIO30 = 0;          // GPIO30 = input NO
SAMPLEING code added this time
        GpioIntRegs.GPIOXINT1SEL.bit.GPIOSEL = 30;        // Xint1 connected to
GPIO30
```

```
    EDIS;

// Step 3. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
    DINT;

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the DSP2833x_PieCtrl.c file.
    InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
    IER = 0x0000;
    IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example.  This is useful for debug purposes.
// The shell ISR routines are found in DSP2833x_DefaultIsr.c.
// This function is found in DSP2833x_PieVect.c.
    InitPieVectTable();
    MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
    InitFlash();

// Interrupts that are used in this example are re-mapped to
// ISR functions found within this file.
    EALLOW;  // This is needed to write to EALLOW protected registers
    PieVectTable.EPWM1_INT = &epwm1_isr;
    PieVectTable.EPWM2_INT = &epwm2_isr;
    PieVectTable.EPWM3_INT = &epwm3_isr;
    PieVectTable.EPWM4_INT = &epwm4_isr;
    PieVectTable.EPWM5_INT = &epwm5_isr;
    PieVectTable.EPWM6_INT = &epwm6_isr;
    PieVectTable.XINT1 = &xint01;


    EDIS;      // This is needed to disable write to EALLOW protected registers

// Step 4. Initialize all the Device Peripherals:
// This function is found in DSP2833x_InitPeripherals.c
// InitPeripherals();  // Not required for this example
```

```
// For this example, only initialize the ePWM

    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
    EDIS;

    InitEPwm1Example();
    InitEPwm2Example();
    InitEPwm3Example();
    InitEPwm4Example();
    InitEPwm5Example();
    InitEPwm6Example();

    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
    EDIS;


// Step 5. User specific code, enable interrupts:

// Enable CPU INT3 which is connected to EPWM1-3 INT:
    PieCtrlRegs.PIECTRL.bit.ENPIE = 1; // Enable Pieblock not sure if it is needed
    IER |= M_INT3;
    IER |= M_INT1;

// Enable EPWM INTn in the PIE: Group 3 interrupt 1-3
    PieCtrlRegs.PIEIER3.bit.INTx1 = 1;
    PieCtrlRegs.PIEIER3.bit.INTx2 = 1;
    PieCtrlRegs.PIEIER3.bit.INTx3 = 1;
    PieCtrlRegs.PIEIER3.bit.INTx4 = 1;
    PieCtrlRegs.PIEIER3.bit.INTx5 = 1;
    PieCtrlRegs.PIEIER3.bit.INTx6 = 1;
    PieCtrlRegs.PIEIER1.bit.INTx4 = 1;

    XIntruptRegs.XINT1CR.bit.POLARITY=1;
    XIntruptRegs.XINT1CR.bit.ENABLE=1;
// Enable global Interrupts and higher priority real-time debug events:
    EINT;   // Enable Global interrupt INTM
    ERTM;   // Enable Global realtime interrupt DBGM
    //EPwm1Regs.TBSTS.bit.SYNCI=0;
// Step 6. IDLE loop. Just sit and loop forever (optional):
    for(;;)
    {
```

```
        asm("                NOP");
    }

}

interrupt void epwm1_isr(void)
{
    // Update the CMPA and CMPB values
    update_compare1(&epwm1_info);

    // Clear INT flag for this timer
    EPwm1Regs.ETCLR.bit.INT = 1;

    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}


interrupt void epwm2_isr(void)
{

    // Update the CMPA and CMPB values
    update_compare2(&epwm2_info);

    // Clear INT flag for this timer
    EPwm2Regs.ETCLR.bit.INT = 1;

    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

interrupt void epwm3_isr(void)
{

    // Update the CMPA and CMPB values
    update_compare3(&epwm3_info);

    // Clear INT flag for this timer
    EPwm3Regs.ETCLR.bit.INT = 1;

    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}
```

```
interrupt void epwm4_isr(void)
{

    // Update the CMPA and CMPB values
    update_compare4(&epwm4_info);

    // Clear INT flag for this timer
    EPwm4Regs.ETCLR.bit.INT = 1;

    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

interrupt void epwm5_isr(void)
{

    // Update the CMPA and CMPB values
    update_compare5(&epwm5_info);

    // Clear INT flag for this timer
    EPwm5Regs.ETCLR.bit.INT = 1;

    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

interrupt void epwm6_isr(void)
{

    // Update the CMPA and CMPB values
    update_compare6(&epwm6_info);

    // Clear INT flag for this timer
    EPwm6Regs.ETCLR.bit.INT = 1;

    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

interrupt void xint01(void)
{   epwm1_info.intcnt=0;
     epwm1_info.EPwmTimerIntCount=0;
     EPwm1Regs.TBCTR = 0x0000;
     epwm2_info.EPwmTimerIntCount=0;
```

```
    EPwm2Regs.TBCTR = 0x0000;
    epwm3_info.EPwmTimerIntCount=0;
    EPwm3Regs.TBCTR = 0x0000;
    epwm4_info.EPwmTimerIntCount=0;
    EPwm4Regs.TBCTR = 0x0000;
    epwm5_info.EPwmTimerIntCount=0;
    EPwm5Regs.TBCTR = 0x0000;
    epwm6_info.EPwmTimerIntCount=0;
    EPwm6Regs.TBCTR = 0x0000;
    EPwm1Regs.DBCTL.bit.OUT_MODE = DB_DISABLE;
    EPwm3Regs.DBCTL.bit.OUT_MODE = DB_DISABLE;

    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

void InitEPwm1Example()
{

   // Setup TBCLK
   //EPwm1Regs.TBSTS.bit.SYNCI=0;
   EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
   EPwm1Regs.TBPRD = 39040;            // Set timer period
   EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;       // Disable phase loading
   EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;
   EPwm1Regs.TBPHS.half.TBPHS = 0;            // Phase is 0
   EPwm1Regs.TBCTR = 0x0000;                        // Clear counter
   EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV2;         // Clock ratio to
SYSCLKOUT
   EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV2;

   // Setup shadow register load on ZERO
   EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
   EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
   EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
   EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

   // Set Compare values
   EPwm1Regs.CMPA.half.CMPA = 0;       // Set compare A value
   EPwm1Regs.CMPB = 19520;                    // Set Compare B value

   // Set actions
   //EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;                 // Set PWM1A on
Zero
   //EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;               // Clear PWM1A on
```

```
event A, up count
    EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;
    EPwm1Regs.AQCTLA.bit.CBU = AQ_CLEAR;

    // Active high complementary PWMs - Setup the deadband
        EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        EPwm1Regs.DBRED = 30;
        EPwm1Regs.DBFED = 30;

    // Interrupt where we will change the Compare Values
    EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;       // Select INT on Zero
event
    EPwm1Regs.ETSEL.bit.INTEN = 1;                  // Enable INT
    EPwm1Regs.ETPS.bit.INTPRD = ET_1ST;             // Generate INT on 1st
event

    // Information this example uses to keep track
    // of the direction the CMPA/CMPB values are
    // moving, the min and max allowed values and
    // a pointer to the correct ePWM registers
    //epwm1_info.EPwm_CMPA_Direction = EPWM_CMP_UP; // Start by increasing
CMPA & CMPB
//     epwm1_info.EPwm_CMPB_Direction = EPWM_CMP_UP;
    epwm1_info.EPwmTimerIntCount = 0;               // Zero the interrupt counter
    epwm1_info.EPwmRegHandle = &EPwm1Regs;          // Set the pointer to the
ePWM module
    //epwm1_info.EPwmMaxCMPA = EPWM1_MAX_CMPA;      // Setup
min/max CMPA/CMPB values
    //epwm1_info.EPwmMinCMPA = EPWM1_MIN_CMPA;
   // epwm1_info.EPwmMaxCMPB = EPWM1_MAX_CMPB;
   // epwm1_info.EPwmMinCMPB = EPWM1_MIN_CMPB;
    epwm1_info.intcnt=0;

}


void InitEPwm2Example()
{
    // Setup TBCLK
    EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
    EPwm2Regs.TBPRD = 39040;          // Set timer period
    EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;      // Disable phase loading
```

```
    EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
    EPwm2Regs.TBPHS.half.TBPHS = 0x0000;           // Phase is 0
    EPwm2Regs.TBCTR = 0x0000;                      // Clear counter
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV2;       // Clock ratio to SYSCLKOUT
    EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV2;

    // Setup shadow register load on ZERO
    EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    // Set Compare values
    EPwm2Regs.CMPA.half.CMPA = 0;          // Set compare A value
    EPwm2Regs.CMPB = 5856;                 // Set Compare B value

    // Set actions
    EPwm2Regs.AQCTLA.bit.CBU = AQ_CLEAR;           // Clear PWM2A on Period
    EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;             // Set PWM2A on event A, up count

    //EPwm2Regs.AQCTLB.bit.PRD = AQ_CLEAR;         // Clear PWM2B on Period
    //EPwm2Regs.AQCTLB.bit.CBU = AQ_SET;           // Set PWM2B on event B, up count

    // Active high complementary PWMs - Setup the deadband
      //EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
    // EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
      //EPwm2Regs.DBCTL.bit.IN_MODE = DBA_ALL;
      //EPwm2Regs.DBRED = 30;
      //EPwm2Regs.DBFED = 30;


    // Interrupt where we will change the Compare Values
    EPwm2Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;      // Select INT on Zero event
    EPwm2Regs.ETSEL.bit.INTEN = 1;                 // Enable INT
    EPwm2Regs.ETPS.bit.INTPRD = ET_1ST;            // Generate INT on 1st event

    // Information this example uses to keep track
```

```
    // of the direction the CMPA/CMPB values are
    // moving, the min and max allowed values and
    // a pointer to the correct ePWM registers
  // epwm2_info.EPwm_CMPA_Direction = EPWM_CMP_UP;        // Start by increasing CMPA
  // epwm2_info.EPwm_CMPB_Direction = EPWM_CMP_DOWN;      // and decreasing CMPB
    epwm2_info.EPwmTimerIntCount = 0;                     // Zero the interrupt counter
    epwm2_info.EPwmRegHandle = &EPwm2Regs;                // Set the pointer to the ePWM module
  // epwm2_info.EPwmMaxCMPA = EPWM2_MAX_CMPA;              // Setup min/max CMPA/CMPB values
  //   epwm2_info.EPwmMinCMPA = EPWM2_MIN_CMPA;
  //   epwm2_info.EPwmMaxCMPB = EPWM2_MAX_CMPB;
  //   epwm2_info.EPwmMinCMPB = EPWM2_MIN_CMPB;

}


void InitEPwm3Example(void)
{


    // Setup TBCLK
    EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
    EPwm3Regs.TBPRD = 39040;           // Set timer period
    EPwm3Regs.TBCTL.bit.PHSEN = TB_ENABLE;      // Disable phase loading
    EPwm3Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
    EPwm3Regs.TBPHS.half.TBPHS = 0x0000;          // Phase is 0
    EPwm3Regs.TBCTR = 0x0000;                     // Clear counter
    EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV2;       // Clock ratio to SYSCLKOUT
    EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV2;

    // Setup shadow register load on ZERO
    EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

  // Set Compare values
    EPwm3Regs.CMPA.half.CMPA = 0; // Set compare A value
    EPwm3Regs.CMPB = 5000;               // Set Compare B value
```

```
    // Set Actions
    EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;              // Set PWM3A on event B,
up count
    EPwm3Regs.AQCTLA.bit.CBU = AQ_CLEAR;            // Clear PWM3A on
event B, up count

    //EPwm3Regs.AQCTLB.bit.ZRO = AQ_TOGGLE;         // Toggle EPWM3B on
Zero

    // Active high complementary PWMs - Setup the deadband
        EPwm3Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        EPwm3Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        EPwm3Regs.DBRED = 30;
        EPwm3Regs.DBFED = 30;

    // Interrupt where we will change the Compare Values
    EPwm3Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;       // Select INT on Zero
event
    EPwm3Regs.ETSEL.bit.INTEN = 1;                  // Enable INT
    EPwm3Regs.ETPS.bit.INTPRD = ET_1ST;             // Generate INT on 3rd
event

    // Start by increasing the compare A and decreasing compare B
   // epwm3_info.EPwm_CMPA_Direction = EPWM_CMP_UP;
   // epwm3_info.EPwm_CMPB_Direction = EPWM_CMP_DOWN;
    // Start the cout at 0
    epwm3_info.EPwmTimerIntCount = 0;
    epwm3_info.EPwmRegHandle = &EPwm3Regs;
 //   epwm3_info.EPwmMaxCMPA = EPWM3_MAX_CMPA;
  // epwm3_info.EPwmMinCMPA = EPWM3_MIN_CMPA;
 //   epwm3_info.EPwmMaxCMPB = EPWM3_MAX_CMPB;
 //   epwm3_info.EPwmMinCMPB = EPWM3_MIN_CMPB;
}

void InitEPwm4Example(void)
{


    // Setup TBCLK
    EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
    EPwm4Regs.TBPRD = 39040;            // Set timer period
    EPwm4Regs.TBCTL.bit.PHSEN = TB_ENABLE;      // Disable phase loading
```

```
    EPwm4Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
    EPwm4Regs.TBPHS.half.TBPHS = 0x0000;           // Phase is 0
    EPwm4Regs.TBCTR = 0x0000;                      // Clear counter
    EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV2;       // Clock ratio to SYSCLKOUT
    EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV2;

    // Setup shadow register load on ZERO
    EPwm4Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm4Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm4Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm4Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

   // Set Compare values
    EPwm4Regs.CMPA.half.CMPA = 0; // Set compare A value
    EPwm4Regs.CMPB = 10000;               // Set Compare B value

    // Set Actions
    EPwm4Regs.AQCTLA.bit.CAU = AQ_SET;            // Set PWM3A on event B, up count
    EPwm4Regs.AQCTLA.bit.CBU = AQ_CLEAR;          // Clear PWM3A on event B, up count

    //EPwm4Regs.AQCTLB.bit.ZRO = AQ_TOGGLE;       // Toggle EPWM3B on Zero

    // Active high complementary PWMs - Setup the deadband
        //EPwm4Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        //EPwm4Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        //EPwm4Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        //EPwm4Regs.DBRED = 30;
        //EPwm4Regs.DBFED = 30;

    // Interrupt where we will change the Compare Values
    EPwm4Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;     // Select INT on Zero event
    EPwm4Regs.ETSEL.bit.INTEN = 1;                // Enable INT
    EPwm4Regs.ETPS.bit.INTPRD = ET_1ST;           // Generate INT on 3rd event


    // Start the cout at 0
    epwm4_info.EPwmTimerIntCount = 0;
    epwm4_info.EPwmRegHandle = &EPwm4Regs;
```

```
}

void InitEPwm5Example(void)
{


    // Setup TBCLK
    EPwm5Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
    EPwm5Regs.TBPRD = 39040;            // Set timer period
    EPwm5Regs.TBCTL.bit.PHSEN = TB_ENABLE;      // Disable phase loading
    EPwm5Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
    EPwm5Regs.TBPHS.half.TBPHS = 0x0000;          // Phase is 0
    EPwm5Regs.TBCTR = 0x0000;                     // Clear counter
    EPwm5Regs.TBCTL.bit.HSPCLKDIV = TB_DIV2;        // Clock ratio to
SYSCLKOUT
    EPwm5Regs.TBCTL.bit.CLKDIV = TB_DIV2;

    // Setup shadow register load on ZERO
    EPwm5Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm5Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm5Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm5Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

   // Set Compare values
    EPwm5Regs.CMPA.half.CMPA = 0; // Set compare A value
    EPwm5Regs.CMPB = 5000;              // Set Compare B value

    // Set Actions
    EPwm5Regs.AQCTLA.bit.CAU = AQ_SET;            // Set PWM3A on event B,
up count
    EPwm5Regs.AQCTLA.bit.CBU = AQ_CLEAR;          // Clear PWM3A on
event B, up count

    //EPwm5Regs.AQCTLB.bit.ZRO = AQ_TOGGLE;        // Toggle EPWM3B on
Zero

    // Interrupt where we will change the Compare Values
    EPwm5Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;     // Select INT on Zero
event
    EPwm5Regs.ETSEL.bit.INTEN = 1;                // Enable INT
    EPwm5Regs.ETPS.bit.INTPRD = ET_1ST;           // Generate INT on 3rd
event
```

```
    // Start the cout at 0
    epwm5_info.EPwmTimerIntCount = 0;
    epwm5_info.EPwmRegHandle = &EPwm5Regs;

}

void InitEPwm6Example(void)
{


    // Setup TBCLK
    EPwm6Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
    EPwm6Regs.TBPRD = 39040;            // Set timer period
    EPwm6Regs.TBCTL.bit.PHSEN = TB_ENABLE;      // Disable phase loading
    EPwm6Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
    EPwm6Regs.TBPHS.half.TBPHS = 0x0000;          // Phase is 0
    EPwm6Regs.TBCTR = 0x0000;                     // Clear counter
    EPwm6Regs.TBCTL.bit.HSPCLKDIV = TB_DIV2;        // Clock ratio to
SYSCLKOUT
    EPwm6Regs.TBCTL.bit.CLKDIV = TB_DIV2;

    // Setup shadow register load on ZERO
    EPwm6Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm6Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm6Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm6Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

   // Set Compare values
    EPwm6Regs.CMPA.half.CMPA = 0; // Set compare A value
    EPwm6Regs.CMPB = 0;                 // Set Compare B value

    // Set Actions
    EPwm6Regs.AQCTLA.bit.CAU = AQ_SET;              // Set PWM3A on event B,
up count
    EPwm6Regs.AQCTLA.bit.CBU = AQ_CLEAR;            // Clear PWM3A on
event B, up count

    //EPwm6Regs.AQCTLB.bit.ZRO = AQ_TOGGLE;         // Toggle EPWM3B on
Zero

    // Interrupt where we will change the Compare Values
    EPwm6Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;       // Select INT on Zero
event
```

```
    EPwm6Regs.ETSEL.bit.INTEN = 1;                  // Enable INT
    EPwm6Regs.ETPS.bit.INTPRD = ET_1ST;             // Generate INT on 3rd
event


    // Start the cout at 0
    epwm6_info.EPwmTimerIntCount = 0;
    epwm6_info.EPwmRegHandle = &EPwm6Regs;

}
//left up 1
void update_compare1(EPWM_INFO *epwm_info)
{

    if(epwm_info->EPwmTimerIntCount == 15)
    {
          epwm_info->EPwmTimerIntCount = 0;
          epwm_info->intcnt++;
          epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
          epwm_info->EPwmRegHandle->CMPB=0;
    }


    else
    {     if(epwm_info->EPwmTimerIntCount == 0)
          {

              epwm_info->EPwmRegHandle->CMPA.half.CMPA=30;
              epwm_info->EPwmRegHandle->CMPB=25989;

          }
           else
          {     if(epwm_info->EPwmTimerIntCount == 1)
                {
            EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
                                    EPwm1Regs.DBCTL.bit.POLSEL                  =
DB_ACTV_HIC;
                                    EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL;
                                    EPwm1Regs.DBRED = 30;
                                    EPwm1Regs.DBFED = 30;
                   epwm_info->EPwmRegHandle->CMPA.half.CMPA=7733;
                   epwm_info->EPwmRegHandle->CMPB=27774;
              }
              else
```

```
            {     if(epwm_info->EPwmTimerIntCount == 2)
                   {
                   epwm_info->EPwmRegHandle->CMPA.half.CMPA=8524;
                    epwm_info->EPwmRegHandle->CMPB=28197;
                   }
                  else
                  {    if(epwm_info->EPwmTimerIntCount == 3)
                         {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8761;
                             epwm_info->EPwmRegHandle->CMPB=28320;
                         }
                       else
                       {    if(epwm_info->EPwmTimerIntCount == 4)
                              {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8800;
                                epwm_info->EPwmRegHandle->CMPB=28282;
                              }
                             else
                             {   if(epwm_info->EPwmTimerIntCount == 5)
                                   {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8677;
                                     epwm_info->EPwmRegHandle->CMPB=28044;
                                   }
                                  else
                                  {    if(epwm_info->EPwmTimerIntCount == 6)
                                       {

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8254;

    epwm_info->EPwmRegHandle->CMPB=27253;
                                       }
                                       else
                                       {    if(epwm_info->EPwmTimerIntCount == 7)
                                            {

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=6468;

    epwm_info->EPwmRegHandle->CMPB=19520;
                                            }
                                            else
                                            {
```

```
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

    epwm_info->EPwmRegHandle->CMPB=0;
                                if(epwm_info->EPwmTimerIntCount == 9)
                                                                    {
                                EPwm1Regs.DBCTL.bit.OUT_MODE                =
DB_DISABLE;
                                                                    }

                                        //EPwm1Regs.DBCTL.bit.POLSEL      =
DB_ACTV_HIC;
                                        //EPwm1Regs.DBCTL.bit.IN_MODE     =
DBA_ALL;
                                        //EPwm1Regs.DBRED = 30;
                                        //EPwm1Regs.DBFED = 30;
                                    }

                                }
                            }
                        }
                    }
                }
            }


        epwm_info->EPwmTimerIntCount++;
        epwm_info->intcnt++;
    }

    return;
}

// right down 1
void update_compare2(EPWM_INFO *epwm_info)
{
    if(epwm_info->EPwmTimerIntCount == 15)
    {
        epwm_info->EPwmTimerIntCount = 0;
        epwm_info->intcnt++;
        epwm_info->EPwmRegHandle->CMPA.half.CMPA=13052;
        epwm_info->EPwmRegHandle->CMPB=19520;
```

```
    }


    else
    {     if(epwm_info->EPwmTimerIntCount == 0)
          {
               epwm_info->EPwmRegHandle->CMPA.half.CMPA=30;
               epwm_info->EPwmRegHandle->CMPB=19520;
          }
           else
          {     if(epwm_info->EPwmTimerIntCount == 1)
               {
                  epwm_info->EPwmRegHandle->CMPA.half.CMPA=7733;
                  epwm_info->EPwmRegHandle->CMPB=11787;
               }
               else
               {     if(epwm_info->EPwmTimerIntCount == 2)
                     {
                     epwm_info->EPwmRegHandle->CMPA.half.CMPA=8524;
                      epwm_info->EPwmRegHandle->CMPB=10996;
                     }
                    else
                    {     if(epwm_info->EPwmTimerIntCount == 3)
                              {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8761;
                                   epwm_info->EPwmRegHandle->CMPB=10759;
                              }
                         else
                         {     if(epwm_info->EPwmTimerIntCount == 4)
                                 {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8800;
                                    epwm_info->EPwmRegHandle->CMPB=10720;
                                 }
                               else
                               {     if(epwm_info->EPwmTimerIntCount == 5)
                                     {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8677;
                                       epwm_info->EPwmRegHandle->CMPB=10843;
                                     }
                                    else
                                    {     if(epwm_info->EPwmTimerIntCount == 6)
```

```
                                    {

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8254;

    epwm_info->EPwmRegHandle->CMPB=11266;
                                    }
                                    else
                                    {   if(epwm_info->EPwmTimerIntCount == 7)
                                        {

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=6468;

    epwm_info->EPwmRegHandle->CMPB=13052;
                                        }
                                        else
                                        {   if(epwm_info->EPwmTimerIntCount == 8)

{epwm_info->EPwmRegHandle->CMPA.half.CMPA=19520;

epwm_info->EPwmRegHandle->CMPB=25989;}

                                                else
                                                {   if(epwm_info->EPwmTimerIntCount
== 9)
                                                    {

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=11787;

    epwm_info->EPwmRegHandle->CMPB=27774;
                                                    }
                                                    else
                                                    {
    if(epwm_info->EPwmTimerIntCount == 10)
                                                        {

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=10996;

    epwm_info->EPwmRegHandle->CMPB=28197;
                                                        }
                                                        else
                                                        {
    if(epwm_info->EPwmTimerIntCount == 11)
                                                            {
```

```
epwm_info->EPwmRegHandle->CMPA.half.CMPA=10759;

epwm_info->EPwmRegHandle->CMPB=28320;
                                                    }
                                                    else
                                                    {
if(epwm_info->EPwmTimerIntCount == 12)
                                                        {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10720;

epwm_info->EPwmRegHandle->CMPB=28282;
                                                        }
                                                         else
                                                        {
if(epwm_info->EPwmTimerIntCount == 13)
                                                            {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10843;

epwm_info->EPwmRegHandle->CMPB=28044;
                                                            }
                                                            else
                                                            {
if(epwm_info->EPwmTimerIntCount == 14)
                                                                {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=11266;

epwm_info->EPwmRegHandle->CMPB=27253;
                                                                }

                                                                    }
                                                                }
                                                            }
                                                        }
                                                    }
                                                }


                                    }
                                }
                            }
```

```
                    }
                }
            }
        }



        epwm_info->EPwmTimerIntCount++;
        epwm_info->intcnt++;
    }

    return;
}

//left down2
void update_compare3(EPWM_INFO *epwm_info)
{

    if(epwm_info->EPwmTimerIntCount == 15)
    {
        epwm_info->EPwmTimerIntCount = 0;
        epwm_info->intcnt++;
        epwm_info->EPwmRegHandle->CMPA.half.CMPA=6468;
        epwm_info->EPwmRegHandle->CMPB=19520;

    }


    else
    {     if(epwm_info->EPwmTimerIntCount == 0)
          {
               epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
              epwm_info->EPwmRegHandle->CMPB=0;


                    //EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
                    //EPwm3Regs.DBCTL.bit.IN_MODE = DBA_ALL;
                    //EPwm3Regs.DBRED = 30;
                    //EPwm3Regs.DBFED = 30;
              //epwm_info->EPwmRegHandle->CMPA.half.CMPA=25989;
              //epwm_info->EPwmRegHandle->CMPB=32571;
          }
           else
          {     if(epwm_info->EPwmTimerIntCount == 1)
```

```
        {     EPwm3Regs.DBCTL.bit.OUT_MODE = DB_DISABLE;
        }
              //if(epwm_info->EPwmTimerIntCount == 1)
            // {    epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
               //    epwm_info->EPwmRegHandle->CMPB=0;
                  //epwm_info->EPwmRegHandle->CMPA.half.CMPA=27774;
                  //epwm_info->EPwmRegHandle->CMPB=30786;
             //}
            // else
             //{    if(epwm_info->EPwmTimerIntCount == 2)
                //      {
                  //                epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                  //                epwm_info->EPwmRegHandle->CMPB=0;
                    //epwm_info->EPwmRegHandle->CMPA.half.CMPA=28197;
                     //epwm_info->EPwmRegHandle->CMPB=30363;
                   // }
                  //     else
                  //     {     if(epwm_info->EPwmTimerIntCount == 3)
                        //     {    epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                        //          epwm_info->EPwmRegHandle->CMPB=0;

//epwm_info->EPwmRegHandle->CMPA.half.CMPA=28320;
                                 //epwm_info->EPwmRegHandle->CMPB=30240;
                             //}
                        //else
                   //     {     if(epwm_info->EPwmTimerIntCount == 4)
                        //       {epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                          //              epwm_info->EPwmRegHandle->CMPB=0;

//epwm_info->EPwmRegHandle->CMPA.half.CMPA=28282;
                                  //epwm_info->EPwmRegHandle->CMPB=30278;
                               // }
                           //    else
                           //    {  if(epwm_info->EPwmTimerIntCount == 5)
                             //                                                          {
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                            //                epwm_info->EPwmRegHandle->CMPB=0;

//epwm_info->EPwmRegHandle->CMPA.half.CMPA=28044;
                                    //epwm_info->EPwmRegHandle->CMPB=30516;
                                //    }
                             //    else
                                  //{   if(epwm_info->EPwmTimerIntCount == 6)
                                //      {
```

```
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                                //        epwm_info->EPwmRegHandle->CMPB=0;

    //epwm_info->EPwmRegHandle->CMPA.half.CMPA=27253;

    //epwm_info->EPwmRegHandle->CMPB=31307;
                                // }
                               //    else
                               //    {    if(epwm_info->EPwmTimerIntCount == 7)
                                //        {
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                              //
    epwm_info->EPwmRegHandle->CMPB=0;
//

    //epwm_info->EPwmRegHandle->CMPA.half.CMPA=19520;

    //epwm_info->EPwmRegHandle->CMPB=39040;
                                       //}
                                       //else
                                       //{
                                           if(epwm_info->EPwmTimerIntCount == 8)
                                            {


    epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

    epwm_info->EPwmRegHandle->CMPB=25989;

                                            }
                                            else
                                            {    if(epwm_info->EPwmTimerIntCount
== 9)
                                                {
                                    EPwm3Regs.DBCTL.bit.OUT_MODE        =
DB_FULL_ENABLE;
                                    EPwm3Regs.DBCTL.bit.POLSEL          =
DB_ACTV_HIC;
                                    EPwm3Regs.DBCTL.bit.IN_MODE         =
DBA_ALL;
                                    EPwm3Regs.DBRED = 30;
                                    EPwm3Regs.DBFED = 30;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=7733;
```

```
epwm_info->EPwmRegHandle->CMPB=27774;



//epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

//epwm_info->EPwmRegHandle->CMPB=0;
                                                }
                                                else
                                                {
if(epwm_info->EPwmTimerIntCount == 10)
                                                    {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8524;

epwm_info->EPwmRegHandle->CMPB=28197;
                                                }
                                                 else
                                                 {

if(epwm_info->EPwmTimerIntCount == 11)
                                                        {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8761;

epwm_info->EPwmRegHandle->CMPB=28320;


//epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

//epwm_info->EPwmRegHandle->CMPB=0;
                                                        }
                                                        else
                                                        {

if(epwm_info->EPwmTimerIntCount == 12)
                                                            {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8800;

epwm_info->EPwmRegHandle->CMPB=28282;
                                                            }
                                                            else
```

```
                                                    {

if(epwm_info->EPwmTimerIntCount == 13)
                                                        {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8677;

epwm_info->EPwmRegHandle->CMPB=28044;

//epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

//epwm_info->EPwmRegHandle->CMPB=0;
                                                        }
                                                        else
                                                        {

if(epwm_info->EPwmTimerIntCount == 14)
                                                            {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8254;

epwm_info->EPwmRegHandle->CMPB=27253;
                                                            }

                                                        }









                                                    }
                        }
                    }
                }
            }
        }
```

```
        epwm_info->EPwmTimerIntCount++;
        epwm_info->intcnt++;
    }

    return;
}
//right up2
void update_compare4(EPWM_INFO *epwm_info)
{

    if(epwm_info->EPwmTimerIntCount == 15)
    {
        epwm_info->EPwmTimerIntCount = 0;
        epwm_info->intcnt++;
        epwm_info->EPwmRegHandle->CMPA.half.CMPA=6468;
        epwm_info->EPwmRegHandle->CMPB=13052;

    }


    else
    {     if(epwm_info->EPwmTimerIntCount == 0)
          {
              epwm_info->EPwmRegHandle->CMPA.half.CMPA=19520;
              epwm_info->EPwmRegHandle->CMPB=25989;
          }
           else
          {     if(epwm_info->EPwmTimerIntCount == 1)
                {
                   epwm_info->EPwmRegHandle->CMPA.half.CMPA=11787;
                   epwm_info->EPwmRegHandle->CMPB=27774;
                }
                else
                {     if(epwm_info->EPwmTimerIntCount == 2)
                       {
                       epwm_info->EPwmRegHandle->CMPA.half.CMPA=10996;
                        epwm_info->EPwmRegHandle->CMPB=28197;
                       }
                      else
                      {     if(epwm_info->EPwmTimerIntCount == 3)
                                {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10759;
```

```
                        epwm_info->EPwmRegHandle->CMPB=28320;
                    }
                else
                {   if(epwm_info->EPwmTimerIntCount == 4)
                    {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10720;
                        epwm_info->EPwmRegHandle->CMPB=28282;
                    }
                    else
                    {   if(epwm_info->EPwmTimerIntCount == 5)
                        {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10843;
                            epwm_info->EPwmRegHandle->CMPB=28044;
                        }
                        else
                        {   if(epwm_info->EPwmTimerIntCount == 6)
                            {

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=11266;

    epwm_info->EPwmRegHandle->CMPB=27253;
                            }
                            else
                            {   if(epwm_info->EPwmTimerIntCount == 7)
                                {

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=13052;

    epwm_info->EPwmRegHandle->CMPB=19520;
                                }
                                //else
                                //{
                                //
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                                //
    epwm_info->EPwmRegHandle->CMPB=0;
                                //}

                                else
                                {   if(epwm_info->EPwmTimerIntCount == 8)
                                    {
```

```
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

    epwm_info->EPwmRegHandle->CMPB=19520;
                                        }
                                        else
                                        {    if(epwm_info->EPwmTimerIntCount
== 9)
                                             {

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=7733;

    epwm_info->EPwmRegHandle->CMPB=11787;
                                             }
                                             else
                                             {
    if(epwm_info->EPwmTimerIntCount == 10)
                                                  {

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8524;

    epwm_info->EPwmRegHandle->CMPB=10996;
                                                  }
                                                  else
                                                  {
    if(epwm_info->EPwmTimerIntCount == 11)
                                                       {

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8761;

    epwm_info->EPwmRegHandle->CMPB=10759;
                                                       }
                                                       else
                                                       {
    if(epwm_info->EPwmTimerIntCount == 12)
                                                            {

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8800;

    epwm_info->EPwmRegHandle->CMPB=10720;
                                                            }
                                                            else
                                                            {
    if(epwm_info->EPwmTimerIntCount == 13)
                                                                 {
```

```
epwm_info->EPwmRegHandle->CMPA.half.CMPA=8677;

epwm_info->EPwmRegHandle->CMPB=10843;
                                                    }
                                                    else
                                                    {
if(epwm_info->EPwmTimerIntCount == 14)
                                                        {

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8254;

epwm_info->EPwmRegHandle->CMPB=11266;
                                                        }
                                        //              else
                                        //              {
if(epwm_info->EPwmTimerIntCount == 15)
                                                        //  {
                                                        //
epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                                                        //
epwm_info->EPwmRegHandle->CMPB=0;
                                                        //  }
                                                        //  else
                                                        //  {
                                                        //
epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                                                        //
epwm_info->EPwmRegHandle->CMPB=19520;
                                                        //  }
                                                        }
                                                    }
                                                }
                                            }
                                        }

                                        }
                                    //}
                                }
                            }
                        }
                    }
                }
```

```
            }
        }



        epwm_info->EPwmTimerIntCount++;
        epwm_info->intcnt++;
    }

    return;
}
//right up1
void update_compare5(EPWM_INFO *epwm_info)
{

     if(epwm_info->EPwmTimerIntCount == 22)
        {     epwm_info->EPwmRegHandle->TBPRD=19520;
              epwm_info->EPwmTimerIntCount = 0;
              epwm_info->intcnt++;
              epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
              epwm_info->EPwmRegHandle->CMPB=0;
        }
        else
        {     if(epwm_info->EPwmTimerIntCount == 0)
             {      epwm_info->EPwmRegHandle->TBPRD=39040;
                   epwm_info->EPwmRegHandle->CMPA.half.CMPA=25989;
                   epwm_info->EPwmRegHandle->CMPB=32571;
             }
              else
             {      if(epwm_info->EPwmTimerIntCount == 1)
                   {    epwm_info->EPwmRegHandle->TBPRD=39040;
                       epwm_info->EPwmRegHandle->CMPA.half.CMPA=27774;
                       epwm_info->EPwmRegHandle->CMPB=30786;
                   }
                   else
                   {      if(epwm_info->EPwmTimerIntCount == 2)
                         {epwm_info->EPwmRegHandle->TBPRD=39040;
                         epwm_info->EPwmRegHandle->CMPA.half.CMPA=28197;
                          epwm_info->EPwmRegHandle->CMPB=30363;
                         }
                        else
                        {     if(epwm_info->EPwmTimerIntCount == 3)
                                    {    epwm_info->EPwmRegHandle->TBPRD=39040;
```

```
epwm_info->EPwmRegHandle->CMPA.half.CMPA=28320;
                                    epwm_info->EPwmRegHandle->CMPB=30240;
                                }
                            else
                            {    if(epwm_info->EPwmTimerIntCount == 4)
                                   { epwm_info->EPwmRegHandle->TBPRD=39040;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=28282;
                                     epwm_info->EPwmRegHandle->CMPB=30278;
                                   }
                                 else
                                 {   if(epwm_info->EPwmTimerIntCount == 5)

{ epwm_info->EPwmRegHandle->TBPRD=39040;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=28044;
                                       epwm_info->EPwmRegHandle->CMPB=30516;
                                     }
                                    else
                                    {    if(epwm_info->EPwmTimerIntCount == 6)

    {    epwm_info->EPwmRegHandle->TBPRD=39040;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=27253;

    epwm_info->EPwmRegHandle->CMPB=31307;
                                        }
                                        else
                                        {   if(epwm_info->EPwmTimerIntCount == 7)

    {    epwm_info->EPwmRegHandle->TBPRD=39040;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=19520;

    epwm_info->EPwmRegHandle->CMPB=39040;
                                            }


                                            else
                                            {    if(epwm_info->EPwmTimerIntCount
== 8)

    {    epwm_info->EPwmRegHandle->TBPRD=39040;
```

```
epwm_info->EPwmRegHandle->CMPA.half.CMPA=32571;

epwm_info->EPwmRegHandle->CMPB=39040;

                                                }
                                                   else
                                                  {
    if(epwm_info->EPwmTimerIntCount == 9)

{     epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

epwm_info->EPwmRegHandle->CMPB=7732;
                                                                  }
                                                else
                                                {
    if(epwm_info->EPwmTimerIntCount == 10)

    {     epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=11266;

epwm_info->EPwmRegHandle->CMPB=19520;
                                                     }
                                                     else
                                                     {
    if(epwm_info->EPwmTimerIntCount == 11)

    {epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

epwm_info->EPwmRegHandle->CMPB=8524;
                                                        }
                                                        else
                                                        {
    if(epwm_info->EPwmTimerIntCount == 12)
                                                             {

    epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10843;
```

```
epwm_info->EPwmRegHandle->CMPB=19520;
                                                    }
                                                    else
                                                    {
    if(epwm_info->EPwmTimerIntCount == 13)

    {epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

    epwm_info->EPwmRegHandle->CMPB=8760;
                                                  }
                                                   else
                                                   {
    if(epwm_info->EPwmTimerIntCount == 14)

    {epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10720;

epwm_info->EPwmRegHandle->CMPB=19520;
                                                     }
                                                     else
                                                     {
    if(epwm_info->EPwmTimerIntCount == 15)

    {     epwm_info->EPwmRegHandle->TBPRD=19520;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

    epwm_info->EPwmRegHandle->CMPB=8800;
                                                          }
                                                 else
                                                      {
    if(epwm_info->EPwmTimerIntCount == 16)
                                                        {

epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10758;

epwm_info->EPwmRegHandle->CMPB=19520;
                                                       }
                                                      else
```

```
                                                                {
    if(epwm_info->EPwmTimerIntCount == 17)
                                                                    {
epwm_info->EPwmRegHandle->TBPRD=19520;
epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
epwm_info->EPwmRegHandle->CMPB=8676;
                                                                    }
                                                            else
                                                                {
    if(epwm_info->EPwmTimerIntCount == 18)
                                                                {
epwm_info->EPwmRegHandle->TBPRD=19520;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=10996;
epwm_info->EPwmRegHandle->CMPB=19520;
                                                                }
                                                                else
                                            {if(epwm_info->EPwmTimerIntCount    ==
19)
                                            {
epwm_info->EPwmRegHandle->TBPRD=19520;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
epwm_info->EPwmRegHandle->CMPB=8254;
                                            }
                    else
```

```
{     if(epwm_info->EPwmTimerIntCount == 20)

{

epwm_info->EPwmRegHandle->TBPRD=19520;

      epwm_info->EPwmRegHandle->CMPA.half.CMPA=11787;

epwm_info->EPwmRegHandle->CMPB=19520;

           }

                                                  else

                                        {if(epwm_info->EPwmTimerIntCount == 21)

                                        {

                                          epwm_info->EPwmRegHandle->TBPRD=19520;

      epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

                                          epwm_info->EPwmRegHandle->CMPB=6468;

                                         }
```

```
}

}

}

}

}

}
}
}}}}}}}
//else
// {      if(epwm_info->EPwmTimerIntCount == 15)
//        {
 //
```

```
epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                                //            epwm_info->EPwmRegHandle->CMPB=4380;
                                //       }
                                   //     else
                                   //         {  if(epwm_info->EPwmTimerIntCount == 16)
                                   //                 {
                                     //
epwm_info->EPwmRegHandle->CMPA.half.CMPA=15120;
                                     //
epwm_info->EPwmRegHandle->CMPB=19520;
                                     //              }
                                       //            else
                                     //              {     if(epwm_info->EPwmTimerIntCount
== 17)
                                     //                      {
                                     //
epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                                     //
epwm_info->EPwmRegHandle->CMPB=4400;
                                     //                      }




                                        //                              }



                                   //     }


                                //    }


                                                                        //     }
                                                                    //     }
                                                                //       }
                                                                //}
                                                            //}
                                                        //}

                                                    //}
                                                //}
                                            }
```

```
                                    }
                                }
                            }
                        }
                    }
                }



                epwm_info->EPwmTimerIntCount++;
                epwm_info->intcnt++;
            }

            return;
}

//right down2
void update_compare6(EPWM_INFO *epwm_info)
{

    if(epwm_info->EPwmTimerIntCount == 22)
            {
                epwm_info->EPwmTimerIntCount = 0;
                epwm_info->intcnt++;
                epwm_info->EPwmRegHandle->TBPRD=39040;
                epwm_info->EPwmRegHandle->CMPA.half.CMPA=19520;
                epwm_info->EPwmRegHandle->CMPB=39040;

            }


            else
            {     if(epwm_info->EPwmTimerIntCount == 0)
                  {
                       epwm_info->EPwmRegHandle->TBPRD=39040;
                      epwm_info->EPwmRegHandle->CMPA.half.CMPA=32571;
                      epwm_info->EPwmRegHandle->CMPB=39040;
                  }
                   else
                  {
                          if(epwm_info->EPwmTimerIntCount == 1)
                            {epwm_info->EPwmRegHandle->TBPRD=19520;
                            epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                        //epwm_info->EPwmRegHandle->CMPB=0;
```

```
                    epwm_info->EPwmRegHandle->CMPB=7732;
                   }
                  else
                  {    if(epwm_info->EPwmTimerIntCount == 2)
                        {   epwm_info->EPwmRegHandle->TBPRD=19520;

//epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                            //epwm_info->EPwmRegHandle->CMPB=0;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=11266;
                       epwm_info->EPwmRegHandle->CMPB=19520;
                        }

                     else
                     {    if(epwm_info->EPwmTimerIntCount == 3)
                          { epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
                            epwm_info->EPwmRegHandle->CMPB=8524;
                          }
                        else
                        {   if(epwm_info->EPwmTimerIntCount == 4)

{ epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10843;
                             epwm_info->EPwmRegHandle->CMPB=19520;
                            }
                           else
                           {    if(epwm_info->EPwmTimerIntCount == 5)

    {     epwm_info->EPwmRegHandle->TBPRD=19520;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

    epwm_info->EPwmRegHandle->CMPB=8760;
                                    }
                                    else
                                    {   if(epwm_info->EPwmTimerIntCount == 6)

    {     epwm_info->EPwmRegHandle->TBPRD=19520;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=10720;
```

```
epwm_info->EPwmRegHandle->CMPB=19520;
                                        }


                                        else
                                        {   if(epwm_info->EPwmTimerIntCount
== 7)
{   epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

epwm_info->EPwmRegHandle->CMPB=8800;
                                            }
                                            else
                                            {
if(epwm_info->EPwmTimerIntCount == 8)

{   epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10758;

epwm_info->EPwmRegHandle->CMPB=19520;
                                                }
                                                else
                                                {
if(epwm_info->EPwmTimerIntCount == 9)

{   epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

epwm_info->EPwmRegHandle->CMPB=8676;
                                                    }
                                                    else
                                                    {
if(epwm_info->EPwmTimerIntCount == 10)

{   epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10996;

epwm_info->EPwmRegHandle->CMPB=19520;
                                                        }
```

```
                                                    else
                                                    {
if(epwm_info->EPwmTimerIntCount == 11)

{     epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

epwm_info->EPwmRegHandle->CMPB=8254;
                                                }
                                                 else
                                                 {
if(epwm_info->EPwmTimerIntCount == 12)

{     epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=11787;

epwm_info->EPwmRegHandle->CMPB=19520;
                                                    }
                                                    else
                                                    {
if(epwm_info->EPwmTimerIntCount == 13)

{     epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

epwm_info->EPwmRegHandle->CMPB=6468;
                                                        }
                                                        else
                                                        {
if(epwm_info->EPwmTimerIntCount == 14)

{     epwm_info->EPwmRegHandle->TBPRD=19520;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

epwm_info->EPwmRegHandle->CMPB=0;
                                                            }

                                                    else
                                                    {
if(epwm_info->EPwmTimerIntCount == 15)
```

```
{epwm_info->EPwmRegHandle->TBPRD=39040;
epwm_info->EPwmRegHandle->CMPA.half.CMPA=25989;
epwm_info->EPwmRegHandle->CMPB=32571;}
                                                    else
                                                    {
    if(epwm_info->EPwmTimerIntCount == 16)
                                                        {
epwm_info->EPwmRegHandle->TBPRD=39040;
epwm_info->EPwmRegHandle->CMPA.half.CMPA=27774;
epwm_info->EPwmRegHandle->CMPB=30786;
                                                        }
                                                 else
                                                    {
    if(epwm_info->EPwmTimerIntCount == 17)
                                                       {
epwm_info->EPwmRegHandle->TBPRD=39040;
epwm_info->EPwmRegHandle->CMPA.half.CMPA=28197;
epwm_info->EPwmRegHandle->CMPB=30363;
                                                       }
                                              else
                                                    {
    if(epwm_info->EPwmTimerIntCount == 18)
                                                       {
epwm_info->EPwmRegHandle->TBPRD=39040;
epwm_info->EPwmRegHandle->CMPA.half.CMPA=28320;
epwm_info->EPwmRegHandle->CMPB=30240;
                                                       }
                                              else
{     if(epwm_info->EPwmTimerIntCount == 19)
                                                        {
```

```
epwm_info->EPwmRegHandle->TBPRD=39040;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=28282;

epwm_info->EPwmRegHandle->CMPB=30278;
                                                    }
                                                  else

{if(epwm_info->EPwmTimerIntCount == 20)
                                                    {

epwm_info->EPwmRegHandle->TBPRD=39040;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=28044;

epwm_info->EPwmRegHandle->CMPB=30516;
                                                    }



                                                     else
                                                     {

    if(epwm_info->EPwmTimerIntCount == 21)

    {epwm_info->EPwmRegHandle->TBPRD=39040;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=27253;

    epwm_info->EPwmRegHandle->CMPB=31307;

                                                          }


                    }}}}}}}
            }}}}}}}}}}}}}}
```

```
            epwm_info->EPwmTimerIntCount++;
            epwm_info->intcnt++;
        }

        return;
}

//===========================================================
==============
// No more.
//===========================================================
==============
```

Appendix B: Four Switch Model

```
//###########################################################################
#######
//
// FILE:       Example_2833xEPwm3UpAQ.c
//
// TITLE:      Action Qualifier Module Upcount mode.
//
// ASSUMPTIONS:
//
//      This program requires the DSP2833x header files.
//
//      Monitor the ePWM1 - ePWM3 pins on a oscilloscope as
//      described below.
//
//          EPWM1A is on GPIO0
//          EPWM1B is on GPIO1
//
//          EPWM2A is on GPIO2
//          EPWM2B is on GPIO3
//
//          EPWM3A is on GPIO4
//          EPWM3B is on GPIO5
//
//      As supplied, this project is configured for "boot to SARAM"
//      operation.    The 2833x Boot Mode table is shown below.
//      For information on configuring the boot mode of an eZdsp,
```

```
//         please refer to the documentation included with the eZdsp,
//
//            $Boot_Table:
//
//              GPIO87     GPIO86      GPIO85     GPIO84
//              XA15       XA14        XA13       XA12
//              PU         PU          PU         PU
//            ========================================
//                1          1           1          1      Jump to Flash
//                1          1           1          0      SCI-A boot
//                1          1           0          1      SPI-A boot
//                1          1           0          0      I2C-A boot
//                1          0           1          1      eCAN-A boot
//                1          0           1          0      McBSP-A boot
//                1          0           0          1      Jump to XINTF x16
//                1          0           0          0      Jump to XINTF x32
//                0          1           1          1      Jump to OTP
//                0          1           1          0      Parallel GPIO I/O boot
//                0          1           0          1      Parallel XINTF boot
//                0          1           0          0      Jump to SARAM      <-
"boot to SARAM"
//                0          0           1          1      Branch to check boot mode
//                0          0           1          0      Boot to flash, bypass ADC
cal
//                0          0           0          1      Boot to SARAM, bypass
ADC cal
//                0          0           0          0      Boot to SCI-A, bypass
ADC cal
//                                                         Boot_Table_End$
//
// DESCRIPTION:
//
//      This example configures ePWM1, ePWM2, ePWM3 to produce an
//      waveform with independant modulation on EPWMxA and
//      EPWMxB.
//
//      The compare values CMPA and CMPB are modified within the ePWM's ISR
//
//      The TB counter is in upmode for this example.
//
//      View the EPWM1A/B, EPWM2A/B and EPWM3A/B waveforms
//      via an oscilloscope
//
//
```

```
//###########################################################################
#######
// $TI Release: 2833x/2823x Header Files and Peripheral Examples V133 $
// $Release Date: June 8, 2012 $
//###########################################################################
#######
//leftup1 epwm1 rightdown1 epwm2 rightup2 epwm4 leftdown2 epwm3

#include "DSP28x_Project.h"         // Device Headerfile and Examples Include File


typedef struct
{
    volatile struct EPWM_REGS *EPwmRegHandle;
    Uint16 EPwm_CMPA_Direction;
    Uint16 EPwm_CMPB_Direction;
    Uint16 EPwmTimerIntCount;
    Uint16 EPwmMaxCMPA;
    Uint16 EPwmMinCMPA;
    Uint16 EPwmMaxCMPB;
    Uint16 EPwmMinCMPB;
    Uint16 intcnt;
}EPWM_INFO;



void InitEPwm1Example(void);
void InitEPwm2Example(void);
void InitEPwm3Example(void);
void InitEPwm4Example(void);

interrupt void epwm1_isr(void);
interrupt void epwm2_isr(void);
interrupt void epwm3_isr(void);
interrupt void epwm4_isr(void);

void update_compare1(EPWM_INFO*);
void update_compare2(EPWM_INFO*);
void update_compare3(EPWM_INFO*);
void update_compare4(EPWM_INFO*);

interrupt void xint01(void);
```

```
// Global variables used in this example
EPWM_INFO epwm1_info;
EPWM_INFO epwm2_info;
EPWM_INFO epwm3_info;
EPWM_INFO epwm4_info;


// Configure the period for each timer
//#define EPWM1_TIMER_TBPRD    39040    // Period register
//#define EPWM2_TIMER_TBPRD    39040    // Period register
//#define EPWM3_TIMER_TBPRD    39040    // Period register
//#define EPWM4_TIMER_TBPRD    39040    // Period register
//#define EPWM5_TIMER_TBPRD    39040    // Period register
//#define EPWM6_TIMER_TBPRD    39040    // Period register


void main(void)
{
// Step 1. Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the DSP2833x_SysCtrl.c file.
     InitSysCtrl();

// Step 2. Initalize GPIO:
// This example function is found in the DSP2833x_Gpio.c file and
// illustrates how to set the GPIO to it's default state.
// InitGpio();    // Skipped for this example

// For this case just init GPIO pins for ePWM1, ePWM2, ePWM3
// These functions are in the DSP2833x_EPwm.c file
     InitEPwm1Gpio();
     InitEPwm2Gpio();
     InitEPwm3Gpio();
     InitEPwm4Gpio();

     EALLOW;

        //GpioCtrlRegs.GPBPUD.bit.GPIO32 = 0;//pull up
        //GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 2;//make it syncin

        GpioCtrlRegs.GPAPUD.bit.GPIO30 = 0;
        GpioCtrlRegs.GPAMUX2.bit.GPIO30 = 0;    // GPIO30 = GPIO30
        GpioCtrlRegs.GPADIR.bit.GPIO30   =   0;          //   GPIO30   =   input   NO
SAMPLEING code added this time
```

```
    GpioIntRegs.GPIOXINT1SEL.bit.GPIOSEL = 30;      // Xint1 connected to
GPIO30



    EDIS;

// Step 3. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
    DINT;

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the DSP2833x_PieCtrl.c file.
    InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
    IER = 0x0000;
    IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example.    This is useful for debug purposes.
// The shell ISR routines are found in DSP2833x_DefaultIsr.c.
// This function is found in DSP2833x_PieVect.c.
    InitPieVectTable();
    MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
    InitFlash();

// Interrupts that are used in this example are re-mapped to
// ISR functions found within this file.
    EALLOW;    // This is needed to write to EALLOW protected registers
    PieVectTable.EPWM1_INT = &epwm1_isr;
    PieVectTable.EPWM2_INT = &epwm2_isr;
    PieVectTable.EPWM3_INT = &epwm3_isr;
    PieVectTable.EPWM4_INT = &epwm4_isr;

    PieVectTable.XINT1 = &xint01;



    EDIS;       // This is needed to disable write to EALLOW protected registers
```

```
// Step 4. Initialize all the Device Peripherals:
// This function is found in DSP2833x_InitPeripherals.c
// InitPeripherals();  // Not required for this example

// For this example, only initialize the ePWM

    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
    EDIS;

    InitEPwm1Example();
    InitEPwm2Example();
    InitEPwm3Example();
    InitEPwm4Example();


    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
    EDIS;


// Step 5. User specific code, enable interrupts:

// Enable CPU INT3 which is connected to EPWM1-3 INT:
    PieCtrlRegs.PIECTRL.bit.ENPIE = 1; // Enable Pieblock not sure if it is needed
    IER |= M_INT3;
    IER |= M_INT1;

// Enable EPWM INTn in the PIE: Group 3 interrupt 1-3
    PieCtrlRegs.PIEIER3.bit.INTx1 = 1;
    PieCtrlRegs.PIEIER3.bit.INTx2 = 1;
    PieCtrlRegs.PIEIER3.bit.INTx3 = 1;
    PieCtrlRegs.PIEIER3.bit.INTx4 = 1;

    PieCtrlRegs.PIEIER1.bit.INTx4 = 1;

    XIntruptRegs.XINT1CR.bit.POLARITY=1;
    XIntruptRegs.XINT1CR.bit.ENABLE=1;
// Enable global Interrupts and higher priority real-time debug events:
    EINT;   // Enable Global interrupt INTM
    ERTM;   // Enable Global realtime interrupt DBGM
    //EPwm1Regs.TBSTS.bit.SYNCI=0;
// Step 6. IDLE loop. Just sit and loop forever (optional):
    for(;;)
```

```
    {
        asm("                NOP");
    }

}

interrupt void epwm1_isr(void)
{
    // Update the CMPA and CMPB values
    update_compare1(&epwm1_info);

    // Clear INT flag for this timer
    EPwm1Regs.ETCLR.bit.INT = 1;

    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}


interrupt void epwm2_isr(void)
{

    // Update the CMPA and CMPB values
    update_compare2(&epwm2_info);

    // Clear INT flag for this timer
    EPwm2Regs.ETCLR.bit.INT = 1;

    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

interrupt void epwm3_isr(void)
{

    // Update the CMPA and CMPB values
    update_compare3(&epwm3_info);

    // Clear INT flag for this timer
    EPwm3Regs.ETCLR.bit.INT = 1;

    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}
```

```
interrupt void epwm4_isr(void)
{

    // Update the CMPA and CMPB values
    update_compare4(&epwm4_info);

    // Clear INT flag for this timer
    EPwm4Regs.ETCLR.bit.INT = 1;

    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}



interrupt void xint01(void)
{   epwm1_info.intcnt=0;
     epwm1_info.EPwmTimerIntCount=0;
     EPwm1Regs.TBCTR = 0x0000;
     epwm2_info.EPwmTimerIntCount=0;
     EPwm2Regs.TBCTR = 0x0000;
     epwm3_info.EPwmTimerIntCount=0;
     EPwm3Regs.TBCTR = 0x0000;
     epwm4_info.EPwmTimerIntCount=0;
     EPwm4Regs.TBCTR = 0x0000;

     //EPwm1Regs.DBCTL.bit.OUT_MODE = DB_DISABLE;
     //EPwm3Regs.DBCTL.bit.OUT_MODE = DB_DISABLE;

     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

void InitEPwm1Example()
{

    // Setup TBCLK
    //EPwm1Regs.TBSTS.bit.SYNCI=0;
    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
    EPwm1Regs.TBPRD = 19531;            // Set timer period
    EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;       // Disable phase loading
    EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;
    EPwm1Regs.TBPHS.half.TBPHS = 0;             // Phase is 0
    EPwm1Regs.TBCTR = 0x0000;                               // Clear counter
```

```
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV2;        // Clock ratio to
SYSCLKOUT
    EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV2;

    // Setup shadow register load on ZERO
    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    // Set Compare values
    EPwm1Regs.CMPA.half.CMPA = 0;       // Set compare A value
    EPwm1Regs.CMPB = 0;                  // Set Compare B value

    // Set actions
    //EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;                 // Set PWM1A on
Zero
    //EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;               // Clear PWM1A on
event A, up count
    EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;
    EPwm1Regs.AQCTLA.bit.CBU = AQ_CLEAR;

    // Active high complementary PWMs - Setup the deadband
        EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        EPwm1Regs.DBRED = 30;
        EPwm1Regs.DBFED = 30;

    // Interrupt where we will change the Compare Values
    EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;       // Select INT on Zero
event
    EPwm1Regs.ETSEL.bit.INTEN = 1;                  // Enable INT
    EPwm1Regs.ETPS.bit.INTPRD = ET_1ST;             // Generate INT on 1st
event

    // Information this example uses to keep track
    // of the direction the CMPA/CMPB values are
    // moving, the min and max allowed values and
    // a pointer to the correct ePWM registers
    //epwm1_info.EPwm_CMPA_Direction = EPWM_CMP_UP; // Start by increasing
CMPA & CMPB
//     epwm1_info.EPwm_CMPB_Direction = EPWM_CMP_UP;
    epwm1_info.EPwmTimerIntCount = 0;               // Zero the interrupt counter
```

```
    epwm1_info.EPwmRegHandle = &EPwm1Regs;             // Set the pointer to the
ePWM module
    //epwm1_info.EPwmMaxCMPA = EPWM1_MAX_CMPA;              // Setup
min/max CMPA/CMPB values
    //epwm1_info.EPwmMinCMPA = EPWM1_MIN_CMPA;
   // epwm1_info.EPwmMaxCMPB = EPWM1_MAX_CMPB;
   // epwm1_info.EPwmMinCMPB = EPWM1_MIN_CMPB;
    epwm1_info.intcnt=0;

}


void InitEPwm2Example()
{
    // Setup TBCLK
    EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
    EPwm2Regs.TBPRD = 19531;           // Set timer period
    EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;      // Disable phase loading
    EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
    EPwm2Regs.TBPHS.half.TBPHS = 0x0000;           // Phase is 0
    EPwm2Regs.TBCTR = 0x0000;                       // Clear counter
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV2;        // Clock ratio to
SYSCLKOUT
    EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV2;

    // Setup shadow register load on ZERO
    EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    // Set Compare values
    EPwm2Regs.CMPA.half.CMPA = 0;          // Set compare A value
    EPwm2Regs.CMPB = 0;                     // Set Compare B value

    // Set actions
    EPwm2Regs.AQCTLA.bit.CBU = AQ_CLEAR;                  // Clear PWM2A
on Period
    EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;                   // Set PWM2A on
event A, up count

    //EPwm2Regs.AQCTLB.bit.PRD = AQ_CLEAR;                  // Clear PWM2B
on Period
    //EPwm2Regs.AQCTLB.bit.CBU = AQ_SET;                   // Set PWM2B on
```

```
event B, up count

    // Active high complementary PWMs - Setup the deadband
     //EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
    // EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
     //EPwm2Regs.DBCTL.bit.IN_MODE = DBA_ALL;
     //EPwm2Regs.DBRED = 30;
     //EPwm2Regs.DBFED = 30;


    // Interrupt where we will change the Compare Values
    EPwm2Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;             // Select INT on
Zero event
    EPwm2Regs.ETSEL.bit.INTEN = 1;                        // Enable INT
    EPwm2Regs.ETPS.bit.INTPRD = ET_1ST;                   // Generate INT on
1st event

    // Information this example uses to keep track
    // of the direction the CMPA/CMPB values are
    // moving, the min and max allowed values and
    // a pointer to the correct ePWM registers
   // epwm2_info.EPwm_CMPA_Direction = EPWM_CMP_UP;        // Start by
increasing CMPA
   // epwm2_info.EPwm_CMPB_Direction = EPWM_CMP_DOWN;      // and
decreasing CMPB
    epwm2_info.EPwmTimerIntCount = 0;                     // Zero the interrupt
counter
    epwm2_info.EPwmRegHandle = &EPwm2Regs;                // Set the pointer to
the ePWM module
   // epwm2_info.EPwmMaxCMPA = EPWM2_MAX_CMPA;              // Setup
min/max CMPA/CMPB values
  //   epwm2_info.EPwmMinCMPA = EPWM2_MIN_CMPA;
  //   epwm2_info.EPwmMaxCMPB = EPWM2_MAX_CMPB;
  //   epwm2_info.EPwmMinCMPB = EPWM2_MIN_CMPB;

}


void InitEPwm3Example(void)
{


    // Setup TBCLK
    EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
```

```
    EPwm3Regs.TBPRD = 19531;            // Set timer period
    EPwm3Regs.TBCTL.bit.PHSEN = TB_ENABLE;      // Disable phase loading
    EPwm3Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
    EPwm3Regs.TBPHS.half.TBPHS = 0x0000;          // Phase is 0
    EPwm3Regs.TBCTR = 0x0000;                     // Clear counter
    EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV2;        // Clock ratio to
SYSCLKOUT
    EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV2;

    // Setup shadow register load on ZERO
    EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

   // Set Compare values
    EPwm3Regs.CMPA.half.CMPA = 0; // Set compare A value
    EPwm3Regs.CMPB = 0;                 // Set Compare B value

    // Set Actions
    EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;              // Set PWM3A on event B,
up count
    EPwm3Regs.AQCTLA.bit.CBU = AQ_CLEAR;            // Clear PWM3A on
event B, up count

    //EPwm3Regs.AQCTLB.bit.ZRO = AQ_TOGGLE;         // Toggle EPWM3B on
Zero

    // Active high complementary PWMs - Setup the deadband
        EPwm3Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        EPwm3Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        EPwm3Regs.DBRED = 30;
        EPwm3Regs.DBFED = 30;

    // Interrupt where we will change the Compare Values
    EPwm3Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;       // Select INT on Zero
event
    EPwm3Regs.ETSEL.bit.INTEN = 1;                  // Enable INT
    EPwm3Regs.ETPS.bit.INTPRD = ET_1ST;             // Generate INT on 3rd
event

    // Start by increasing the compare A and decreasing compare B
   // epwm3_info.EPwm_CMPA_Direction = EPWM_CMP_UP;
```

```
   // epwm3_info.EPwm_CMPB_Direction = EPWM_CMP_DOWN;
     // Start the cout at 0
     epwm3_info.EPwmTimerIntCount = 0;
     epwm3_info.EPwmRegHandle = &EPwm3Regs;
 //    epwm3_info.EPwmMaxCMPA = EPWM3_MAX_CMPA;
   // epwm3_info.EPwmMinCMPA = EPWM3_MIN_CMPA;
 //    epwm3_info.EPwmMaxCMPB = EPWM3_MAX_CMPB;
 //    epwm3_info.EPwmMinCMPB = EPWM3_MIN_CMPB;
}

void InitEPwm4Example(void)
{


     // Setup TBCLK
     EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
     EPwm4Regs.TBPRD = 19531;            // Set timer period
     EPwm4Regs.TBCTL.bit.PHSEN = TB_ENABLE;       // Disable phase loading
     EPwm4Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
     EPwm4Regs.TBPHS.half.TBPHS = 0x0000;            // Phase is 0
     EPwm4Regs.TBCTR = 0x0000;                       // Clear counter
     EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV2;         // Clock ratio to
SYSCLKOUT
     EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV2;

     // Setup shadow register load on ZERO
     EPwm4Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
     EPwm4Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
     EPwm4Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
     EPwm4Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

   // Set Compare values
     EPwm4Regs.CMPA.half.CMPA = 0; // Set compare A value
     EPwm4Regs.CMPB = 0;                 // Set Compare B value

     // Set Actions
     EPwm4Regs.AQCTLA.bit.CAU = AQ_SET;              // Set PWM3A on event B,
up count
     EPwm4Regs.AQCTLA.bit.CBU = AQ_CLEAR;             // Clear PWM3A on
event B, up count

     //EPwm4Regs.AQCTLB.bit.ZRO = AQ_TOGGLE;          // Toggle EPWM3B on
Zero
```

```
    // Active high complementary PWMs - Setup the deadband
        //EPwm4Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
        //EPwm4Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
        //EPwm4Regs.DBCTL.bit.IN_MODE = DBA_ALL;
        //EPwm4Regs.DBRED = 30;
        //EPwm4Regs.DBFED = 30;

    // Interrupt where we will change the Compare Values
    EPwm4Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;       // Select INT on Zero
event
    EPwm4Regs.ETSEL.bit.INTEN = 1;                  // Enable INT
    EPwm4Regs.ETPS.bit.INTPRD = ET_1ST;             // Generate INT on 3rd
event


    // Start the cout at 0
    epwm4_info.EPwmTimerIntCount = 0;
    epwm4_info.EPwmRegHandle = &EPwm4Regs;

}

void update_compare1(EPWM_INFO *epwm_info)
{

    if(epwm_info->EPwmTimerIntCount == 16)
    {
        epwm_info->EPwmTimerIntCount = 0;
        epwm_info->intcnt++;
        EPwm1Regs.TBPRD = 19531;
        epwm_info->EPwmRegHandle->CMPA.half.CMPA=30;
        epwm_info->EPwmRegHandle->CMPB=19501;
    }


    else
    {   if(epwm_info->EPwmTimerIntCount == 0)
        {
            EPwm1Regs.TBPRD = 39062;
           epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
           epwm_info->EPwmRegHandle->CMPB=26004;

        }
         else
        {   if(epwm_info->EPwmTimerIntCount == 1)
```

```
            {    EPwm1Regs.TBPRD = 39062;
                 epwm_info->EPwmRegHandle->CMPA.half.CMPA=7738;
                 epwm_info->EPwmRegHandle->CMPB=27790;
            }
            else
            {     if(epwm_info->EPwmTimerIntCount == 2)
                   {
                    EPwm1Regs.TBPRD = 39062;
                   epwm_info->EPwmRegHandle->CMPA.half.CMPA=8529;
                    epwm_info->EPwmRegHandle->CMPB=28213;
                   }
                  else
                  {     if(epwm_info->EPwmTimerIntCount == 3)
                          {
                           EPwm1Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8767;
                              epwm_info->EPwmRegHandle->CMPB=28336;
                          }
                        else
                        {     if(epwm_info->EPwmTimerIntCount == 4)
                              {
                                  EPwm1Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8805;
                                 epwm_info->EPwmRegHandle->CMPB=28298;
                              }
                             else
                             {    if(epwm_info->EPwmTimerIntCount == 5)
                                   {
                                      EPwm1Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8681;
                                    epwm_info->EPwmRegHandle->CMPB=28060;
                                   }
                                  else
                                  {     if(epwm_info->EPwmTimerIntCount == 6)
                                        {
                                              EPwm1Regs.TBPRD = 39062;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8259;

    epwm_info->EPwmRegHandle->CMPB=27269;
                                        }
```

```
                                    else
                                    {     if(epwm_info->EPwmTimerIntCount == 7)
                                          {
                                                EPwm1Regs.TBPRD = 39062;

      epwm_info->EPwmRegHandle->CMPA.half.CMPA=6472;

      epwm_info->EPwmRegHandle->CMPB=19531;
                                          }
                                    //revision
                                          else
                                          {

if(epwm_info->EPwmTimerIntCount == 8)
                                                                              {

EPwm1Regs.TBPRD = 19531;

      epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

      epwm_info->EPwmRegHandle->CMPB=0;
                                                                              }
                        else
                                                            {
      if(epwm_info->EPwmTimerIntCount == 9)
                                                                  {

EPwm1Regs.TBPRD = 39062;

      epwm_info->EPwmRegHandle->CMPA.half.CMPA=6503;

      epwm_info->EPwmRegHandle->CMPB=27239;
                                                                  }
                                    else
                                                                                    {
      if(epwm_info->EPwmTimerIntCount == 10)

      {

EPwm1Regs.TBPRD = 39062;

            epwm_info->EPwmRegHandle->CMPA.half.CMPA=8289;

            epwm_info->EPwmRegHandle->CMPB=28030;
```

```
    }
                                                    else
                {   if(epwm_info->EPwmTimerIntCount == 11)
                    {
EPwm1Regs.TBPRD = 39062;
                        epwm_info->EPwmRegHandle->CMPA.half.CMPA=8712;
                        epwm_info->EPwmRegHandle->CMPB=28268;
                    }
                                                            else
                        {   if(epwm_info->EPwmTimerIntCount == 12)
                            {
EPwm1Regs.TBPRD = 39062;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8835;
    epwm_info->EPwmRegHandle->CMPB=28306;
                            }
                                                                    else
                            {   if(epwm_info->EPwmTimerIntCount ==
13)
                                {
EPwm1Regs.TBPRD = 39062;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8797;
    epwm_info->EPwmRegHandle->CMPB=28182;
```

```
                                        }
                                                                    else
                                        {    if(epwm_info->EPwmTimerIntCount
== 14)
                                            {
    EPwm1Regs.TBPRD = 39062;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8559;
    epwm_info->EPwmRegHandle->CMPB=27770;
                                            }
    else
                                            {
    if(epwm_info->EPwmTimerIntCount == 15)
                                                {
                                                                EPwm1Regs.TBPRD
= 39062;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=7768;
    epwm_info->EPwmRegHandle->CMPB=25973;
                                                }


                                        }}}}}}}
                                    }
                                }
```

```
                                }
                            }
                        }
                    }
                }
            }



        epwm_info->EPwmTimerIntCount++;
        epwm_info->intcnt++;
    }

    return;
}

// right down
void update_compare2(EPWM_INFO *epwm_info)
{

    if(epwm_info->EPwmTimerIntCount == 16)
    {
         EPwm2Regs.TBPRD = 39062;
         epwm_info->EPwmTimerIntCount = 0;
         epwm_info->intcnt++;
         epwm_info->EPwmRegHandle->CMPA.half.CMPA=13089;
         epwm_info->EPwmRegHandle->CMPB=39032;
    }


    else
    {     if(epwm_info->EPwmTimerIntCount == 0)
          {
               EPwm2Regs.TBPRD = 19531;
              epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
              epwm_info->EPwmRegHandle->CMPB=19531;

          }
           else
          {     if(epwm_info->EPwmTimerIntCount == 1)
              {    EPwm2Regs.TBPRD = 39062;
                  epwm_info->EPwmRegHandle->CMPA.half.CMPA=13058;
                  epwm_info->EPwmRegHandle->CMPB=31324;
              }
```

```
            else
            {     if(epwm_info->EPwmTimerIntCount == 2)
                  {
                   EPwm2Regs.TBPRD = 39062;
                  epwm_info->EPwmRegHandle->CMPA.half.CMPA=11272;
                   epwm_info->EPwmRegHandle->CMPB=30533;
                  }
                 else
                 {    if(epwm_info->EPwmTimerIntCount == 3)
                        {
                         EPwm2Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10849;
                            epwm_info->EPwmRegHandle->CMPB=30295;
                        }
                     else
                     {    if(epwm_info->EPwmTimerIntCount == 4)
                          {
                              EPwm2Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10726;
                             epwm_info->EPwmRegHandle->CMPB=30257;
                          }
                         else
                         {   if(epwm_info->EPwmTimerIntCount == 5)
                             {
                                EPwm2Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10764;
                               epwm_info->EPwmRegHandle->CMPB=30380;
                             }
                            else
                            {    if(epwm_info->EPwmTimerIntCount == 6)
                                 {
                                       EPwm2Regs.TBPRD = 39062;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=11002;

    epwm_info->EPwmRegHandle->CMPB=30803;
                                 }
                                 else
                                 {    if(epwm_info->EPwmTimerIntCount == 7)
                                      {
                                            EPwm2Regs.TBPRD = 39062;
```

```
epwm_info->EPwmRegHandle->CMPA.half.CMPA=11793;

epwm_info->EPwmRegHandle->CMPB=32590;
                                        }
                                        else
                                        {

if(epwm_info->EPwmTimerIntCount == 8)

{      EPwm2Regs.TBPRD = 19531;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;

epwm_info->EPwmRegHandle->CMPB=0;
                                                                        }
                //revision

                                                    else
                                                      {
if(epwm_info->EPwmTimerIntCount == 9)
                                                            {      EPwm2Regs.TBPRD =
39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=19561;

epwm_info->EPwmRegHandle->CMPB=32559;
                                                            }
                                        else
                                                                                {
if(epwm_info->EPwmTimerIntCount == 10)

{

EPwm2Regs.TBPRD = 39062;

        epwm_info->EPwmRegHandle->CMPA.half.CMPA=11823;

        epwm_info->EPwmRegHandle->CMPB=30773;

}
                                                    else

                {     if(epwm_info->EPwmTimerIntCount == 11)
```

```
                    {
EPwm2Regs.TBPRD = 39062;
                        epwm_info->EPwmRegHandle->CMPA.half.CMPA=11032;
                        epwm_info->EPwmRegHandle->CMPB=30350;
                    }
                                                    else
                        {   if(epwm_info->EPwmTimerIntCount == 12)
                            {
EPwm2Regs.TBPRD = 39062;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=10794;
    epwm_info->EPwmRegHandle->CMPB=30227;
                            }
                                                        else
                            {   if(epwm_info->EPwmTimerIntCount ==
13)
                                {
EPwm2Regs.TBPRD = 39062;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=10756;
    epwm_info->EPwmRegHandle->CMPB=30265;
                                }
                                                    else
                                {   if(epwm_info->EPwmTimerIntCount
```

```
== 14)
                                                    {
    EPwm2Regs.TBPRD = 39062;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=10879;

    epwm_info->EPwmRegHandle->CMPB=30503;
                                                    }
    else
                                                    {
    if(epwm_info->EPwmTimerIntCount == 15)
                                                        {
                                                            EPwm2Regs.TBPRD
= 39062;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=11302;

    epwm_info->EPwmRegHandle->CMPB=31294;
                                                        }



                                                    }}}}}}}
                                                }
                                            }
                                        }
                                    }
                                }
                            }
```

```
        }



        epwm_info->EPwmTimerIntCount++;
        epwm_info->intcnt++;
    }

    return;
}

//right up
void update_compare3(EPWM_INFO *epwm_info)
{

    if(epwm_info->EPwmTimerIntCount == 16)
    {     EPwm3Regs.TBPRD = 19531;
          epwm_info->EPwmTimerIntCount = 0;
          epwm_info->intcnt++;
          epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
          epwm_info->EPwmRegHandle->CMPB=0;
    }



    else
    {     if(epwm_info->EPwmTimerIntCount == 0)
          {
               EPwm3Regs.TBPRD = 39062;
              epwm_info->EPwmRegHandle->CMPA.half.CMPA=19561;
              epwm_info->EPwmRegHandle->CMPB=32559;


          }
           else
          {     if(epwm_info->EPwmTimerIntCount == 1)
                {
                     EPwm3Regs.TBPRD = 39062;
                    epwm_info->EPwmRegHandle->CMPA.half.CMPA=11823;
                    epwm_info->EPwmRegHandle->CMPB=30773;
                }
                else
                {      if(epwm_info->EPwmTimerIntCount == 2)
                       {
                        EPwm3Regs.TBPRD = 39062;
                       epwm_info->EPwmRegHandle->CMPA.half.CMPA=11032;
```

```
                epwm_info->EPwmRegHandle->CMPB=30350;
               }
              else
              {    if(epwm_info->EPwmTimerIntCount == 3)
                   {
                    EPwm3Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10794;
                       epwm_info->EPwmRegHandle->CMPB=30227;
                   }
                 else
                 {    if(epwm_info->EPwmTimerIntCount == 4)
                      {
                          EPwm3Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10756;
                        epwm_info->EPwmRegHandle->CMPB=30265;
                      }
                     else
                     {   if(epwm_info->EPwmTimerIntCount == 5)
                         {
                            EPwm3Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10879;
                          epwm_info->EPwmRegHandle->CMPB=30503;
                         }
                        else
                        {    if(epwm_info->EPwmTimerIntCount == 6)
                             {
                                   EPwm3Regs.TBPRD = 39062;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=11302;

    epwm_info->EPwmRegHandle->CMPB=31294;
                             }
                             else
                             {    if(epwm_info->EPwmTimerIntCount == 7)
                                  {
                                        EPwm3Regs.TBPRD = 39062;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=13089;

    epwm_info->EPwmRegHandle->CMPB=39032;
                                  }
```

```
                                        //revision
                                            else
                                            {
if(epwm_info->EPwmTimerIntCount == 8)
    {     EPwm3Regs.TBPRD = 19531;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
    epwm_info->EPwmRegHandle->CMPB=19531;
                                                                    }
                    else
                                                        {
    if(epwm_info->EPwmTimerIntCount == 9)
                                                            {     EPwm3Regs.TBPRD =
39062;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=13058;
    epwm_info->EPwmRegHandle->CMPB=31324;
                                                            }
                                        else
                                                                                {
    if(epwm_info->EPwmTimerIntCount == 10)
    {
         EPwm3Regs.TBPRD = 39062;
        epwm_info->EPwmRegHandle->CMPA.half.CMPA=11272;
        epwm_info->EPwmRegHandle->CMPB=30533;
    }
                                                    else
                    {     if(epwm_info->EPwmTimerIntCount == 11)
                        {
EPwm3Regs.TBPRD = 39062;
```

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10849;

epwm_info->EPwmRegHandle->CMPB=30295;

}

else

{ if(epwm_info->EPwmTimerIntCount == 12)

{

EPwm3Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10726;

epwm_info->EPwmRegHandle->CMPB=30257;

}

else

{ if(epwm_info->EPwmTimerIntCount == 13)

{

EPwm3Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=10764;

epwm_info->EPwmRegHandle->CMPB=30380;

}

else

{ if(epwm_info->EPwmTimerIntCount == 14)

{

EPwm3Regs.TBPRD = 39062;

```
epwm_info->EPwmRegHandle->CMPA.half.CMPA=11002;

epwm_info->EPwmRegHandle->CMPB=30803;

                                        }

else

                                        {
if(epwm_info->EPwmTimerIntCount == 15)

                                            {

                                                EPwm3Regs.TBPRD
= 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=11793;

epwm_info->EPwmRegHandle->CMPB=32590;

                                            }



                                    }}}}}}}
                                }

                            }
                        }
                    }
                }
            }
        }


    epwm_info->EPwmTimerIntCount++;
```

```
        epwm_info->intcnt++;
    }

    return;
}
//left down
void update_compare4(EPWM_INFO *epwm_info)
{

    if(epwm_info->EPwmTimerIntCount == 16)
    {
        EPwm4Regs.TBPRD = 39062;
        epwm_info->EPwmTimerIntCount = 0;
        epwm_info->intcnt++;
        epwm_info->EPwmRegHandle->CMPA.half.CMPA=6472;
        epwm_info->EPwmRegHandle->CMPB=19531;
    }


    else
    {     if(epwm_info->EPwmTimerIntCount == 0)
          {
               EPwm4Regs.TBPRD = 19531;
              epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
              epwm_info->EPwmRegHandle->CMPB=0;

          }
           else
          {     if(epwm_info->EPwmTimerIntCount == 1)
                {
                    EPwm4Regs.TBPRD = 39062;
                   epwm_info->EPwmRegHandle->CMPA.half.CMPA=6503;
                   epwm_info->EPwmRegHandle->CMPB=27239;
                }
                else
                {     if(epwm_info->EPwmTimerIntCount == 2)
                      {
                       EPwm4Regs.TBPRD = 39062;
                      epwm_info->EPwmRegHandle->CMPA.half.CMPA=8289;
                       epwm_info->EPwmRegHandle->CMPB=28030;
                      }
                     else
                     {     if(epwm_info->EPwmTimerIntCount == 3)
                                {
```

```
                    EPwm4Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8712;
                        epwm_info->EPwmRegHandle->CMPB=28268;
                    }
                else
                {   if(epwm_info->EPwmTimerIntCount == 4)
                    {
                        EPwm4Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8835;
                        epwm_info->EPwmRegHandle->CMPB=28306;
                    }
                    else
                    {   if(epwm_info->EPwmTimerIntCount == 5)
                        {
                            EPwm4Regs.TBPRD = 39062;

epwm_info->EPwmRegHandle->CMPA.half.CMPA=8797;
                            epwm_info->EPwmRegHandle->CMPB=28182;
                        }
                        else
                        {   if(epwm_info->EPwmTimerIntCount == 6)
                            {
                                EPwm4Regs.TBPRD = 39062;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8559;

    epwm_info->EPwmRegHandle->CMPB=27770;
                            }
                            else
                            {   if(epwm_info->EPwmTimerIntCount == 7)
                                {
                                    EPwm4Regs.TBPRD = 39062;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=7768;

    epwm_info->EPwmRegHandle->CMPB=25973;
                                }
                                else
                                {

if(epwm_info->EPwmTimerIntCount == 8)
```

```
{      EPwm4Regs.TBPRD = 19531;
epwm_info->EPwmRegHandle->CMPA.half.CMPA=30;
epwm_info->EPwmRegHandle->CMPB=19501;
                                                                    }
                                                  //revision
            else
                                                    {
if(epwm_info->EPwmTimerIntCount == 9)
                                                        {
EPwm4Regs.TBPRD = 39062;
epwm_info->EPwmRegHandle->CMPA.half.CMPA=0;
epwm_info->EPwmRegHandle->CMPB=26004;
                                                        }
                                    else
                                                                        {
if(epwm_info->EPwmTimerIntCount == 10)
{
EPwm4Regs.TBPRD = 39062;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=7738;
    epwm_info->EPwmRegHandle->CMPB=27790;
}
                                                  else
            {   if(epwm_info->EPwmTimerIntCount == 11)
                {
EPwm4Regs.TBPRD = 39062;
                    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8529;
                    epwm_info->EPwmRegHandle->CMPB=28213;
                }
```

```
                                                    else
                                {   if(epwm_info->EPwmTimerIntCount == 12)
                                    {
    EPwm4Regs.TBPRD = 39062;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8767;
    epwm_info->EPwmRegHandle->CMPB=28336;
                                    }
                                                        else
                                    {   if(epwm_info->EPwmTimerIntCount ==
13)
                                        {
EPwm4Regs.TBPRD = 39062;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8805;
    epwm_info->EPwmRegHandle->CMPB=28298;
                                        }
                                                            else
                                        {   if(epwm_info->EPwmTimerIntCount
== 14)
                                            {
    EPwm4Regs.TBPRD = 39062;
    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8681;
```

```
    epwm_info->EPwmRegHandle->CMPB=28060;

                                                }

    else

                                                {
    if(epwm_info->EPwmTimerIntCount == 15)

                                                    {

                                                        EPwm4Regs.TBPRD
= 39062;

    epwm_info->EPwmRegHandle->CMPA.half.CMPA=8259;


    epwm_info->EPwmRegHandle->CMPB=27269;

                                                    }




                                                }}}}}}}
                                            }

                                        }
                                    }
                                }
                            }
                        }
                    }
                }


        epwm_info->EPwmTimerIntCount++;
        epwm_info->intcnt++;
    }

    return;
}
```