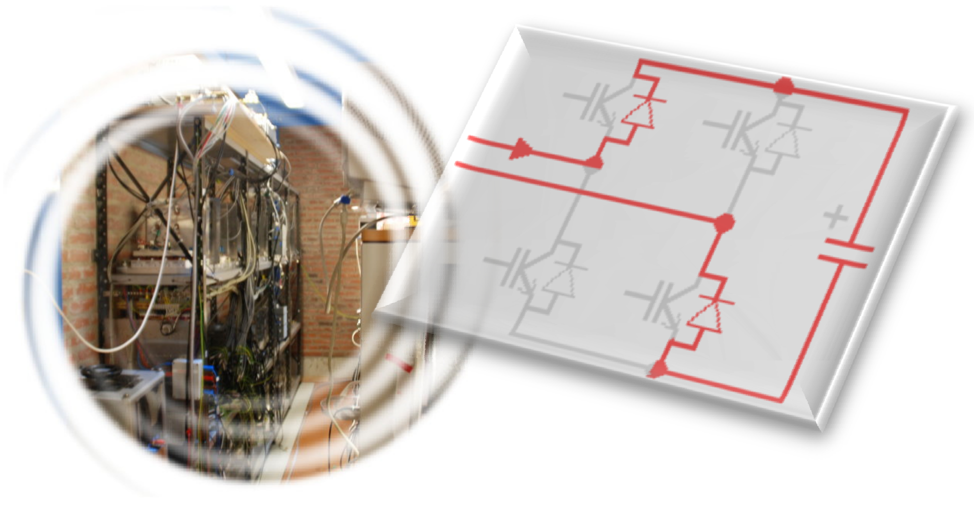
Medium Voltage
Modular Multi-Level Inverter

- Master thesis -

Group 1030



Authors:

Cristian Sandu

Nicoleta Carnu

Valentin Constantin Costea

Supervisor:

Stig Munk-Nielsen

Co-Supervisors:

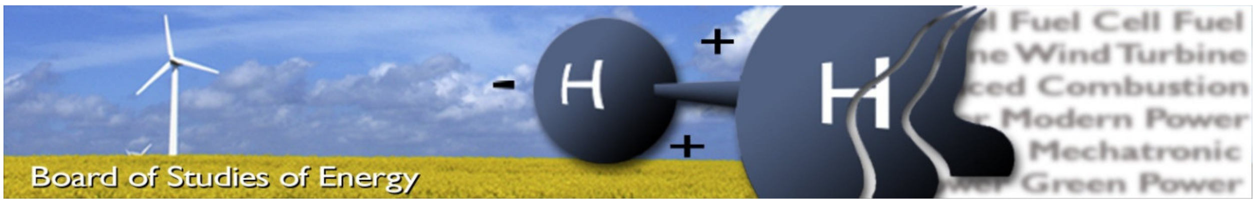
Paul Thøgersen (kk-electronic)

Allan Holm Jørgensen (Vestas)

Florin Lungeanu (Vestas)

Institute of Energy Technology – Pontoppidanstraede 101

Aalborg University, Aalborg, Denmark



Title:	Medium Voltage Modular Multi-Level Inverter
Semester:	PED-4
Project period:	Feb-Iul 2008
ECTS:	30
Supervisor:	Stig-Munk Nielsen
Project group:	1030

____

Sandu Cristian

____

Nicoleta Carnu

____

Valentin Constantin Costea

Copies:	2
Pages, total:	108
Appendix:	2 (212 pages)
Supplements:	0

ABSTRACT:

This project presents the design, building and control of a three phase modular multilevel inverter.

The inverter is simulated with respect to the 3 main modulation methods and 3 submodulation methods. The design and the way units are connected are also explained with direct link to their functionality. The inverter topology, a cascaded full H-Bridge inverter allows a multi-level functionality with 9 levels line-to-line and 5 levels line-to-neutral.

The modulation strategies are explained for the simulations as well as for the FPGA implementation. The control was simulated using Simulink and Plecs toolboxes inside the Matlab platform. Also FPGA code simulation were carried out. The main aspects of the modulations were expresses the main focus being the simulation and construction of the inverter in order to provide a working platform for either motor control or grid connection.

By signing this document, each member of the group confirms that all participated in the project work and thereby that all members are liable for the content of the report that each member had done.

I) TABLE OF CONTENTS

1 ABSTRACT

This project presents the design and building of a nine level three phase Pulse Step Inverter. Experimental setup description is introduced along with the its related functions.

Staircase modulation and two multilevel carrier based Pulse Width Modulation schemes (phase-shifted and level-shifted modulation schemes) are used to test the inverter functionality.

2 ACKNOWLEDGEMENTS

We would like to express our utmost consideration to our main supervisor Associate Professor Stig Munk Nielsen for his guidance throughout the project work.

Our special thanks to our co-supervisor Paul Thøgersen from KK-Electrnic for the helpful comments he has made with respect to our project and for the financial support.

Further we would like to acknowledge Florin Lungeanu and Alan Holm Jørgensen from Vestas for their advices and financial support. Last but not least we are grateful to our families for their patience and love.

1 INTRODUCTION

This chapter provides a brief overview of multilevel inverter topologies currently available on the market that can integrate the system of wind energy production. The project description and the imposed constrains are introduced. Finally, the project layout is presented.

Nowadays, electrical engineer researchers endeavor to provide solutions for a wider range of applications by optimizing and adapting existing configurations of voltage source converters as well by developing new configurations that present potential. A new option with respect to the converter hardware design includes also the multilevel technology. Currently in use are the following multilevel converter topologies:

- Neutral point clamped converter
- Flying capacitor converter
- Cascaded H-bridge converter

Each having advantages and downsides that are presented in [1]. Among these, the latter also known as Pulse Step Inverter (PSI) (Figure 1-1:) has proved a particular good adjusting capability to the requirements of a broad number of applications [2]. It must be point out that the system complexity of PSIs does not increase with the number of levels to the same degree as there is the case with neutral point clamped converters and flying capacitor converter.

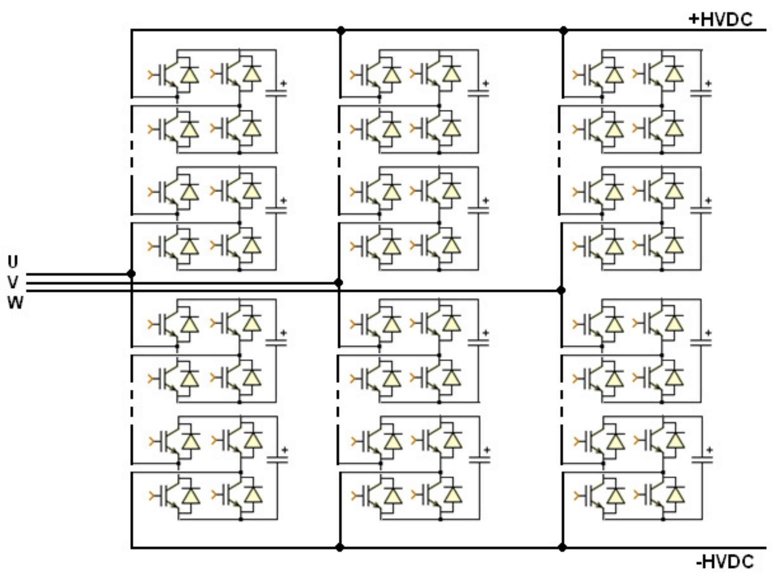


Figure 1-1: Three phase Pulse Step Inverter (PSI)

At first the units were designed with individual dc sources for solar panels and intelligent batteries management in automotive applications. Later on by replacing the dc sources with polarized capacitors its usage was extended also for utility applications, a detailed description of the features, feasibility and control schemes of the multilevel cascaded H-bridge inverter is presented in [3]. Further medium voltage (MV) AC drives also benefit of this type of convertor. An example is Gen 3 Perfect Harmony drive from Robicon which is an IGBT cascaded H-bridge inverter-fed MV drive that is an all-in-one, factory pre-wired and tested system resulting in footprint and cost reduction [4].

1.1.1 VSC- HVDC MODULAR CASCADED H-BRIDGE INVERTER FOR WIND POWER APPLICATIONS

Power generation worldwide tries to meet an ever-increasing power demand and at the same time to orientate towards alternative sources of energy. This mainly because the resources for fossil fuel power plants are predicted to disappear and the disposal of the nuclear power wastes is still at issue. In this context wind power industry emerged and continued to develop. Only in Europe over the last decade attained an average market growth of 40% of wind energy [5]. This has lead to the cost reduction caused by the doubling cycle of production of large-scale, grid-connected wind turbines, that occur almost every three years now [6]. Following the trend, the fourth respectively the fifth generation of wind turbines rated in the range of 3-5MW are designed to be connected in large wind power parks [7]. Up until recently onshore wind farms were preferred to those offshore because of insufficient reliability of tower foundations (supporting structures), tower inability to resist corrosion and the high costs implied. As tower technology for wind turbines evolved, the offshore wind power farms are pushed deeper into the waters where they can no longer disturb the landscape and make use more effectively of the wind distribution [8]. Most offshore wind turbines are equipped with fixed-speed or variable-speed converter-controlled induction generators in order to keep the production costs at a low level. Regarding the undersea transmission lines, at first it was issued that the use of high voltage direct current (HVDC) lines is not a feasible solution unless very long distances involved or the necessity of interconnecting to other grids. This considerations were reviewed when it was found that high capacitance per length unit on the high voltage alternative current (HVAC) transmission lines lead to significant increase of the reactive power thus requiring compensation devices.

In [9] and [10] technical and economical aspects about offshore wind farms connected through dc link to the grid were analyzed. Both assessed that the use of voltage source converter (VSC)-HVDC is a feasible alternative that solves a number of problems such as reliability and stability, that connection of sustainable energy is associated with.

1.1.2 EXISTING CONFIGURATIONS

One on market solution for VSC-HVDC is the Two-level converter with series connected IGBTs that switch synchronously. This has the advantage of a simple topology with modular structure that enables the implementation of conventional pulse with modulation (PWM) or space vector modulation scheme. Also redundancy makes it suitable for systems that require high reliability. Contrariwise it has a particularly high du/dt on the rising as well as on the falling edges, which results in the constraint of keeping the switching frequency low. Further this limitation causes a high level of total harmonic distortion that requires the use of large size LC on the output terminals. But the common-mode voltage that his type of converter fails to mitigate (fails to fully suppress) can have harmful consequences for MV drives [1].

There are also the HVDC Line Commutated Converters based on thyristor valve technology but these require large size space for the offshore substations and ancillary services for low wind speeds, which makes them unsuitable for offshore wind farm applications [9].

In what follows the main features of PSIs are presented [11]:

- The possibility of attaining lighter weight of the wind turbine by replacing the inverter onboard the nacelle with a multi-pulse rectifier instead of having a full inverter;
- Submarine power collection systems based on *DC bus and DC transmission* respectively *multiple level DC*, enable the individual variable speed operation of the wind turbines [12]
- Potential of increasing the tip-speed in order to lower the torque which would lead to costs reduction (possible due to lack of noise limitations for offshore parks) [12]
- Reduced semiconductor stress because of a lower average switching frequency (lower du/dt).
- Regardless of the chosen modulation scheme a high quality output voltage is obtained (no output filter required) leading to reduced electromagnetic compatibility problems.
- Existence of numerous redundant states.
- Low common mode voltage (it can be even annulled if advanced modulation strategies applied
- Less losses over HVDC transmission lines for distances longer than 75 km [12]

It must be point out that the system complexity does not increase with the number of levels as there is the case with the other multilevel topologies.

Silicon usage is ARGUABLY the most IMPORTANT issue in terms of cost efficiency and often regarded as a disadvantage of multilevel cascaded H-bridge inverters. Therefore an important aspect that needs to be taken into account is the voltage rating of the switches. Instead of using IGBT Press Pack Modules as a two-level converter would require, it can be used standard cheap IGBTs (within 600 - 1200V range), depending on the application and has the advantage of eliminating the necessity to equalize the voltage sharing on the series connected capacitors.

Therefore compared with its competitors, the PSI has potential of becoming a viable and cost effective solution for medium and high voltage applications. Consequently there is strong motivation to continue the research in this direction and developing control strategies in order to exploit it at its full operation capacity.

1.2 PROJECT DESCRIPTION

The aim of this project is to study and develop PWM schemes for a three phase modular cascaded H-bridge multilevel voltage source converter.

Beside the staircase modulation which is typical only to this type of converters, there are two carrier-based PWM schemes, phase shifted and level shifted modulation. Space vector modulation can also be implemented but it is not subject of this project.

A three phase modular multilevel inverter with nine voltage levels is to be built in order to implement and then assess the effectiveness of the modulation schemes.

1.3 PROBLEM STATEMENT

1.4 PROJECT LIMITATION

The project limitations that had been set for this project are:

- Study of three main modulations only: Staircase, Phase shifted and Level shifted (with IPD, POD and APOD)
- Voltage limitation to 565 VDC
- Current limitation to 20 Amps
- 3 phases with 8 units per leg

1.5 PROJECT LAYOUT

In order to ease the reading of this report, and to create an overall image of the project, a short description of the main chapters is given here.

- Introduction
- System structure
- Modulation
- Hardware
- Software
- Experimental work
- Conclusions
- Further work
- Nomenclature
- Bibliograph
- Appendix

The *Introduction* chapter presents an overview of the existing multilevel configurations and their possible applications in wind energy industry. The project objectives are introduced and limitations are set.

The *System structure* chapter gives an application overview. A description of how high power hardware and its control (software, communication, etc.) it is also presented.

In the *Modulation* chapter are presented the modulation schemes (phase shifted, level shifted and staircase used to test) used to test the inverter. In this chapter simulation results are given.

The *Hardware* describes the hardware that was done supplementary for the last semester project. The new hardware relates to power supply of the inverter, protections and new relay control logic

The *Software* chapter contains the software description that was used in order to control the system. The software represents the actual implementation that has been done on the inverter.

In *Experimental work* chapter are included data acquisitions from parts of the system. Most of the tests were done with respect to the new units.

The observations made to the system were made in the *Conclusion* chapter.

In *Further work* chapter possible unit design are described, this was proposed at the beginning of the project. The design contains the most relevant information about how the system should look like with respect to modularity.

2 SYSTEM OVERVIEW

The system overview chapter describes the system in order to provide a picture of how the system is composed and which are its main components. The system is presented with its main elements highlighted in order to allow a better understanding of how the system is made. Also the control logic is presented with respect to its main logic devices like FPGAs, DSP and microcontrollers to show their role in the application. The description of why two H bridges legs have been used as well as their functionality difference is shown.

2.1 ABOUT THE SYSTEM

The system contains several modules, each with its own role. The roles of the modules are described in detail in the following sub chapters. In the next paragraphs a short description of the system construction and functionality is described, the rest being the topic of the other following chapters.

The main representation of the inverter is depicted by Figure 2-1. In the figure, the inverter is connected to the main FPGA, the latter being the main control system of the inverter. The FPGA controls the inverter based on data received from the DSP, the secondary FPGA and on the units DC voltages.

The inverter contains a set of IGBT units, H bridges as described further in Unit Types. The units are connected in series as represented by Figure 2-1. A unit contains a set of capacitors which is measured by a voltage sensor placed on each unit. The value read from the voltage sensor is passed to the ADC module than to the FPGA. For this application the MVDC have been replaced with a LVDC (565 VDC from a 400 VAC line).

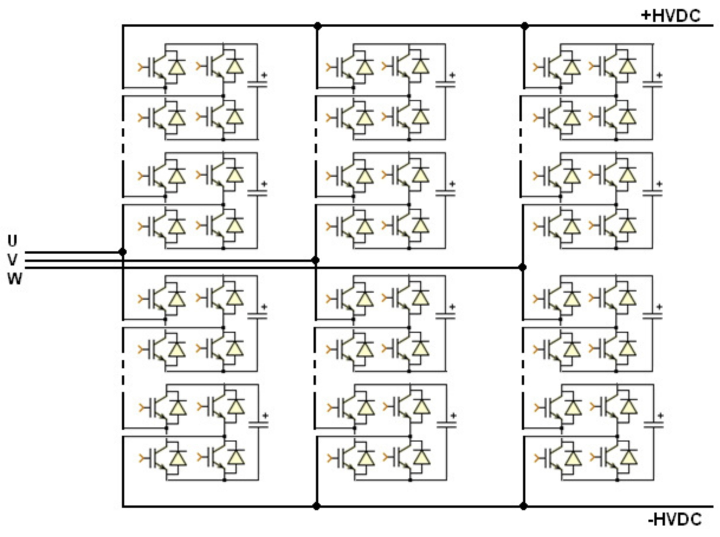


Figure 2-1: Main inverter configuration

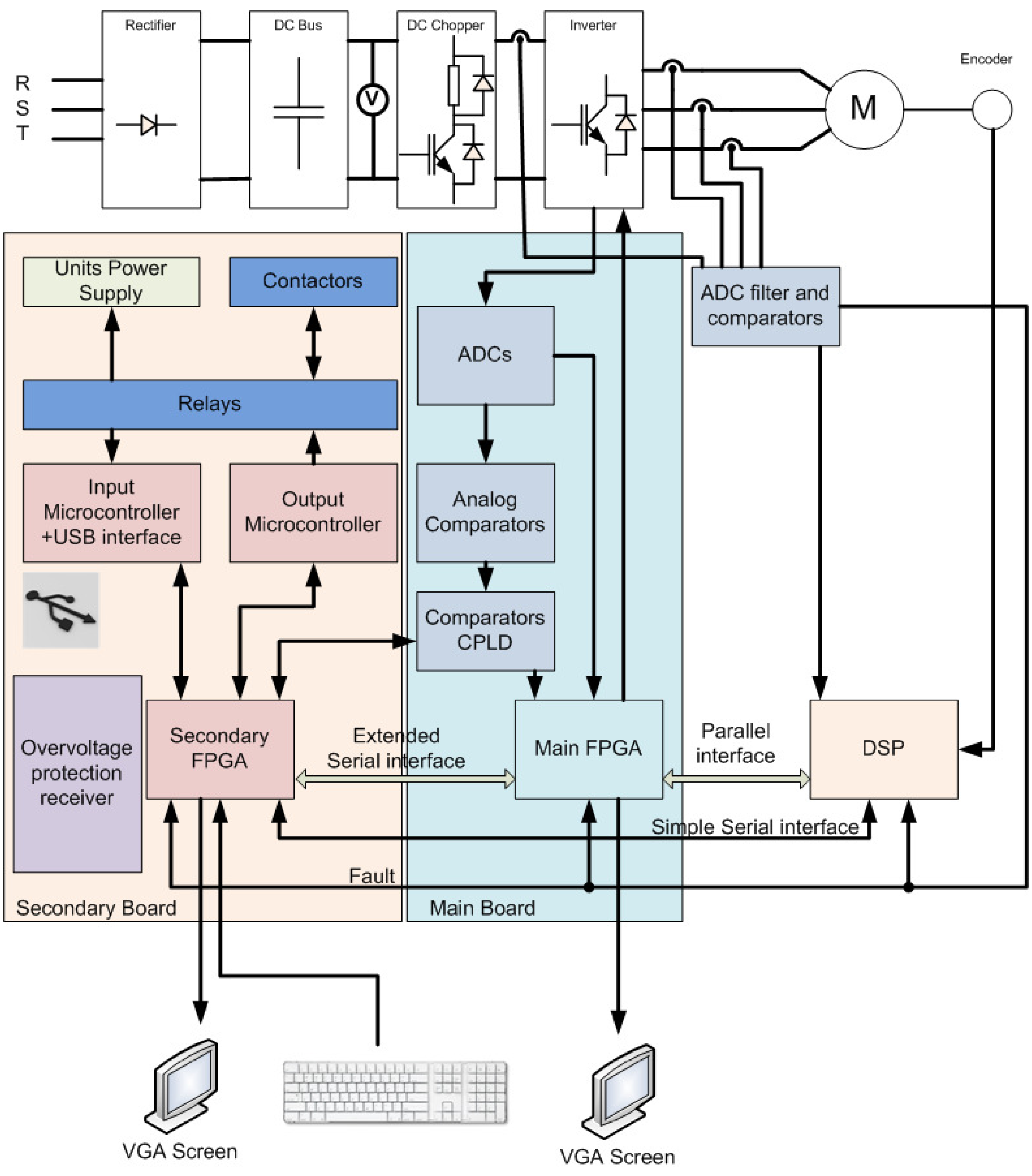


Figure 2-2: The system overview

The DC Chopper is used for protection purposes in order to reduce the DC bus voltage in case of overvoltage protection and to assure that the DC bus voltage will remain constant when the load determines the work in quadrant 2 and 4. This is described in further details in the Hardware chapter.

The DC Bus and the rectifier have the role to create a stable DC voltage. The difference between this rectifier and a normal one is represented by extra care considered for overvoltage.

The overvoltage mentioned earlier for both the rectifier and the DC chopper was taken into consideration because, in case of fault, the inverter unit capacitors may get connected in series and will discharge in the DC bus. This is not a desired event but with the probability that it can happen the consideration to design the power supply to withstand the overvoltage was made.

The motor used by the application has the role only to provide a load in order to see how the inverter will behave in all the 4 quadrants. The motor is made by the ABB and rated 7,5 kW at 400 V. More details about the motor will be given in the control chapter.

The ADCs connected with the FPGA is responsible for measuring the unit voltages as well as the currents from the leg sections. The ADC connected with the DSP is responsible with acquiring the output and DC currents and voltages. These parameters are required for the motor control, while the ones acquired by the FPGA are used for the modulation and system monitoring.

The analog comparators have the role to compare the analog value from the ADC filters with a predefined value. The value is preset during testing to the actual parameters tolerated by the system. The signal from them is passed to the comparators CPLD which will generate a fault signal and will send the data to the secondary FPGA in order to determine the fault source. The comparators CPLD communication is described in [13].

The contactors are used to control the power flow for the power supply, inverter and load. These are required for protections purposes as well as for better control the capacitor charge or discharge. The contactors are controlled by small relays which in turn are controlled by a microcontroller. The role of these contactors is described in the power supply subchapter of the hardware chapter.

The relays used in the application controls the power supply of the units because, the units must be power consequently due to the large inrush current absorbed by the internal switched mode power supply of each unit. Due to this, the units are powered in groups of 4 with 2-3 seconds in between.

The input microcontroller is used to monitor the large contactors for their states in order to determine is a contractor is closed or opened due to external interaction. One role of this is to determine when the over-voltage protection is turned on by checking the chopper contactors and/or the main contactor. The chopper contactor can be controller either by the FPGA or by external logic.

The output microcontroller is used to control the main relays and also to provide a code for automatic system connection. The microcontroller can work in two ways, independent (during automatic power-up or power-down) or controlled by the FPGA.

The main FPGA has the following roles:

- Monitor unit voltages
- Monitor leg section currents
- Handle the gates signals

- Modulation control and selection
- Mathematical operations based on lookup tables stored in a FLASH memory
- Connect the entire logic blocks together
- Provide an interface with the user system

The secondary FPGA has the following roles:

- Control the contactors/relays
- Communicate with the analog comparators
- Monitor the contactors
- Handling the user input interface
- Protections
- Handle the fault signals and act accordingly
- Provide an interface with the user system

The DSP has only one role which is to control the motor as a load and to provide a simple platform on which the control of the inverter can be implemented without the need to program the FPGAs or any other logic system. The DSP can be considered the main control item for the entire system even if the main FPGA is responsible will all the management of the inverter well functionality. Therefore the DSP tasks are:

- Read the output voltage
- Read the output currents
- Read the DC voltage
- Read the motor encoder
- Motor control

All the control items have been placed on two boards named “the cake” and “the biscuit” (See Figure 2-3). The names were given during construction and they remained like this throughout the project. The main board contains the main FPGA with everything required to control the units while the secondary board contains the error handling the contactors controls. The DSP is placed on an intermediary board in order to reduce the distances between the DSP and the FPGAs.

The communication between the FPGAs is done over serial differential lines while between the FPGA and the DSP is done over parallel connection. The communication is an important part of this project because without it the various components are not able to communicate one-another. All the communication topics are discussed in the hardware chapter.

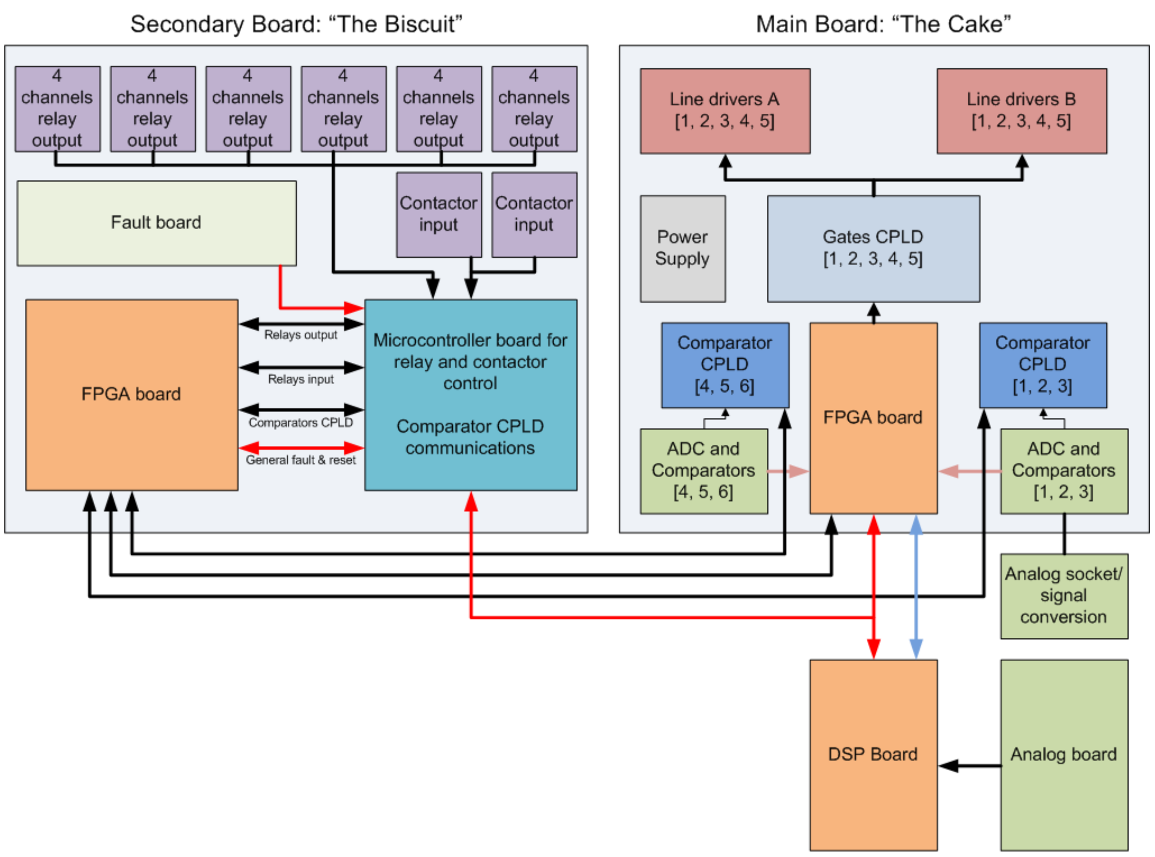

Secondary Board: "The Biscuit"
Main Board: "The Cake"
4 channels relay output
Fault board
Contactor input
FPGA board
Relays output
Relays input
Comparators CPLD
General fault & reset
Microcontroller board for relay and contactor control
Comparator CPLD communications
Line drivers A [1, 2, 3, 4, 5]
Line drivers B [1, 2, 3, 4, 5]
Power Supply
Gates CPLD [1, 2, 3, 4, 5]
Comparator CPLD [4, 5, 6]
Comparator CPLD [1, 2, 3]
ADC and Comparators [4, 5, 6]
ADC and Comparators [1, 2, 3]
Analog socket/ signal conversion
DSP Board
Analog board

Figure 2-3: System board placement

2.2 UNIT TYPES

The IGBT unit of the inverter can be either single leg unit or double leg unit (see Figure 2-4 and Figure 2-5). For each of these two types of configuration various states exist with respect to the IGBT states. The Figure 2-4, the single leg unit, has 3 states for each of the two possible current directions while Figure 2-5 has 9 states. These states have been counted without taking into consideration short-circuit the DC bus. All of these states will be described in the following subchapters.

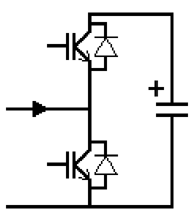


Figure 2-4: Single leg unit

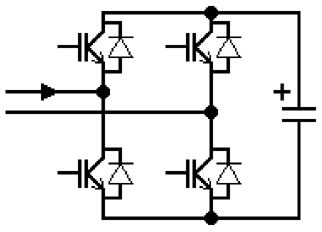


Figure 2-5: Double leg unit

The name of cascaded H Bridge converter comes from the way units are connected one another and the connection is represented by Figure 2-6 and Figure 2-7.

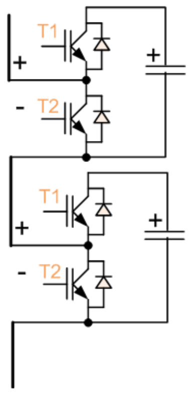


Figure 2-6: Single leg unit connection

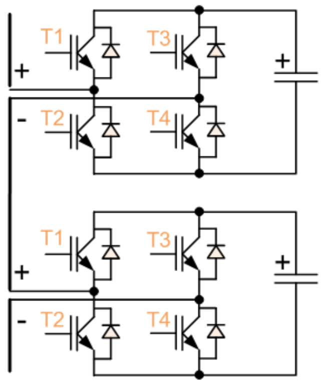


Figure 2-7: Double leg unit connection

2.2.1 SINGLE LEG UNIT

The single leg unit is characterized by 2 serial connected IGBT as Figure 2-4 shows. For this type of unit, the connection made with the other units or to the large inverter DC bus is important because it will operate correctly only if proper connections are made. The unit connections are made similar with Figure 2-1. The DC bus line of the unit is connected to the next unit positive input or to the negative DC bus line of the large inverter. The output of one leg is represented by the mid-point of the leg.

The states for a single leg unit are presented in Table 2-1. The states refer to IGBT command states and to the current flow in the unit. Several states may have the same effect and they are represented in order to show the current path for each of the situations. In the table, the commanded IGBT are marked with blue and the current path is marked with red. The current direction is marked with an arrow on the positive line of the unit.

In case of a direct flow current, in states 00 and 10, the capacitor is connected therefore the unit can be represented as a single capacitor. These states are used for capacitor charging and usage. The opposite state for these is the state 01 where the unit behaves like a simple wire therefore directly conducting from the positive to the negative side of the large inverter. This state removes the unit capacitor from the circuit and may cause it to discharge due to loads inside the unit, the discharging resistors and the internal resistors. For this application the discharging resistors have not been mounted because one role of the application is to maintain a constant voltage level across the units.

For the reverse current flow the states 00 and 01 have the same effect thus conducting the current towards the previous unit or to the positive DC bus line of the large inverter. Either way, these states have the capacitor removed from the circuit. Opposite the state 00 or 01 is the state 10 which reversely polarize the capacitor causing it to discharge.

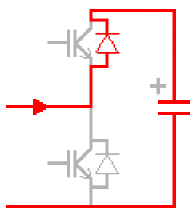
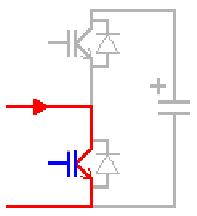
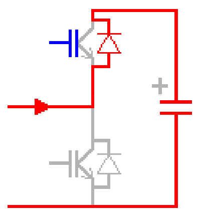
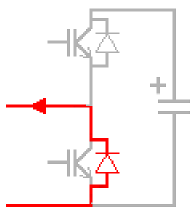
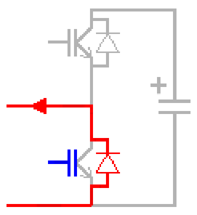
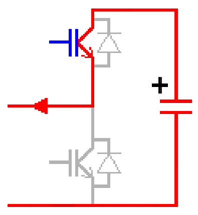
	State 00	**State 01**	**State 10**
Direct current flow			
Inverse current flow			

Table 2-1: States of a single leg unit

By using the Table 2-1 the proper ways of controlling the unit are highlighted. The pros of this type of unit are that it is compact, low component count, reduced switched loses as the capacitor is very close to each transistor. If an inverter will be build by using this type of unit no DC current control is possible for the large inverter DC bus and, during reverse current flow the capacitor can only be discharged. The current path during this reverse current flow doesn't permit voltage balance to take place in order to provide capacitor re-charging. The functionality of this unit in 4 quadrants in only possible if the voltage from the load is directly transferred to the DC bus without passing it to the unit capacitor.

2.2.2 DOUBLE LEG UNIT

The double leg unit, as depicted by Figure 2-5, consists of a full H-bridge. The full H-bridge when compared with the half H-bridge is not sensitive when it comes with the connection in the large inverter. The connector of this type of unit are located at the mid-point of each leg and by having both legs identical there is no strict way of connection as it is for the half H-bridge.

This type of unit is characterized by many more states then a half H-bridge is. Even if there are only two major states for the unit (capacitor connected to the system or direct connection of the input terminals) these are achieved by selecting one of the 9 possible states for each of the two possible current flow directions.

The states for this type of unit are presented in Table 2-2. The states where the capacitor is connected to the system are: 00/00, 00/01, 01/10, 10/00 and 10/01. The opposite states, where the unit behaves like a conductor are: 00/10, 01/00, 01/01 and 10/10.

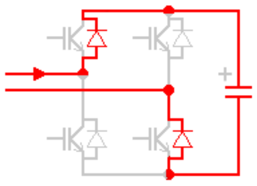
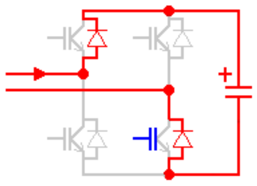
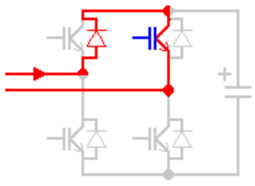
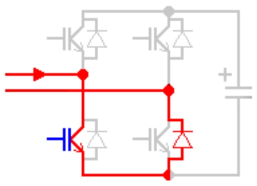
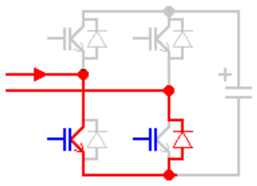
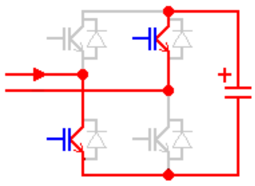
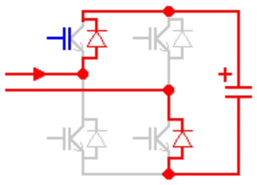
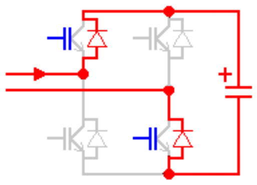
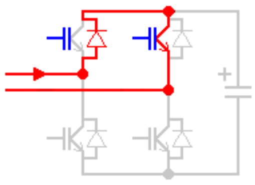
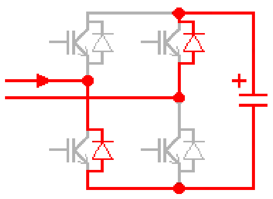
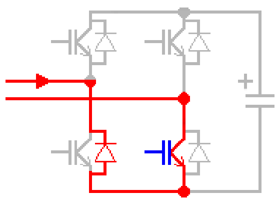
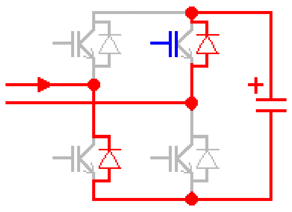
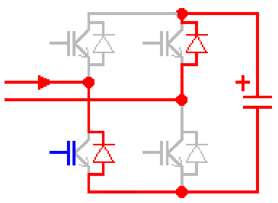
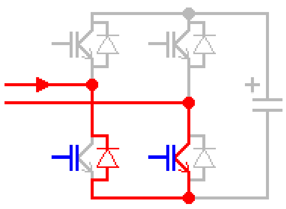
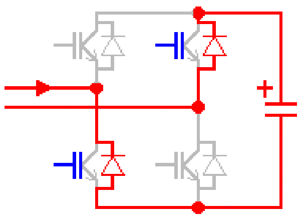
	State xx / 00	State xx / 01	State xx / 10
State 00 / xx			
State 01 / xx			
State 10 / xx			

Table 2-2: States of a double leg unit for positive current path

For the reverse current path, the states are shown in Table 2-3. In this table, the current path can cross the capacitor in the opposite way (state 10/01). The states in which the capacitor is connected to the system are: 00/00, 00/10, 01/00 and 01/10. There are also 4 states for direct conduction. These states are: 00/01, 10/01, 10/00 and 10/10.

	State xx / 00	State xx / 01	State xx / 10
State 00 / xx			
State 01 / xx			

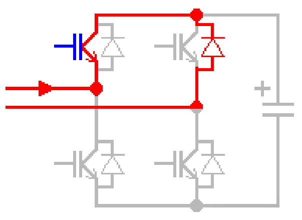
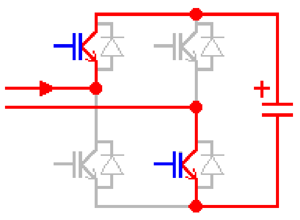
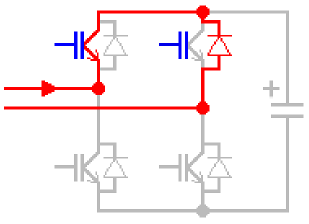
State 10 / xx			

Table 2-3: States of a double leg unit for negative current path

When compared with the half H-bridge configuration, there are double the numbers of semiconductors used by this type of unit while the rest of the parameters are identical. The extra two IGBTs allows control of power flow in both directions by maintain the possibility of voltage balancing and control. The possibility of bidirectional power flow gives the possibility of controlling the DC bus of the large inverter in the case of using several other inverters connected to the same DC bus. The use of this type of unit in a multi-level inverter gives the possibility to compensate for the DC bus instability by extracting energy from the capacitors.

3 MODULATION

In this chapter main modulation schemes for PSI are analyzed. Specifically phase-shifted and level-shifted modulations, both multilevel carrier based Pulse Width Modulation, respectively the staircase modulation, a dedicated modulation scheme for this particular inverter configuration. Performance assessment is made based on the obtained simulation results.

3.1 ABOUT THE MODULATIONS

Phase-shifted and level shifted modulation schemes are PWM carrier based schemes that belong to the unipolar modulation category. This type of modulation uses one triangular carrier and two sinusoidal modulating waves that have the same amplitude and frequency but which are 180^0 out of phase. The line output voltage takes values in the interval $[0; V_{dc}]$ for the positive half-cycle, respectively in the interval $[-V_{dc}; 0]$ for the negative half-cycle from here coming the “unipolar” name [14]. In [14] harmonic spectrum analysis presents a decreased level of low-order harmonics in comparison with bipolar PWM which has only one modulation wave and one carrier.

Staircase modulation is only used in association with inverters that have cascaded multilevel configurations where its implementation is facilitated. In this case the output voltage has an approximated sinusoidal waveform into small voltage steps. The higher the number of voltage steps, the lower harmonic distortion is obtained; this to the point where filters are no longer required. The inverter units can be independently controlled, thus contributing to the reduction of commutation losses [13].

3.2 ABOUT THE SIMULATIONS

The simulations were conducted in Matlab Simulink with the help of Plecs. The model for the modulations was implemented in S-functions. The code for all the S-Functions is included into the Appendix E. The appendix B also contains all the waveforms obtained for the simulations.

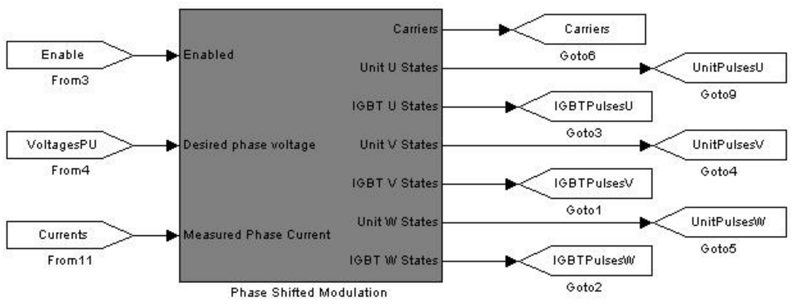


Figure 3-1: Example modulator block

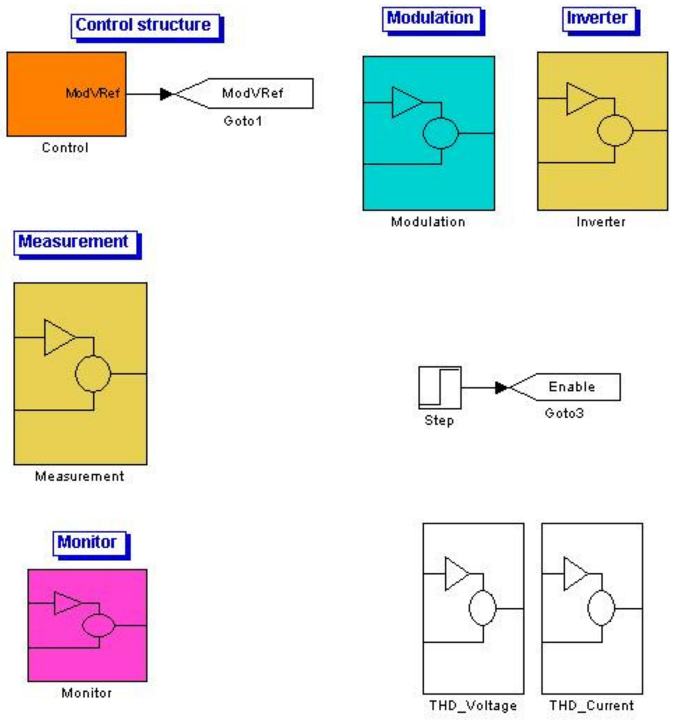


Figure 3-2: Main modulation window

The Control block presented in Figure 3-2 contains a simple sine generator to generate the reference sine waves. The Modulation block contains the modulation block for the current simulation. The Inverter is the plecs model of the inverter. The measurement block splits the output voltage and current into 3 signals each to be processed by the modulation. The monitor, THD_Voltage and THD_current block are used for monitoring and plotting the waveforms. The THD blocks measure the voltage and current THD.

The Plecs model on which the modulations had run is shown in Figure 3-3. The DC Supply contains 2 DC voltage supplies connected in series with the neutral connected in the middle. The inverter contains the schematic from the beginning of chapter 2 with 8 units per leg, double legs per unit. The load is an RL filter with R = 20 ohms and L = 23mH

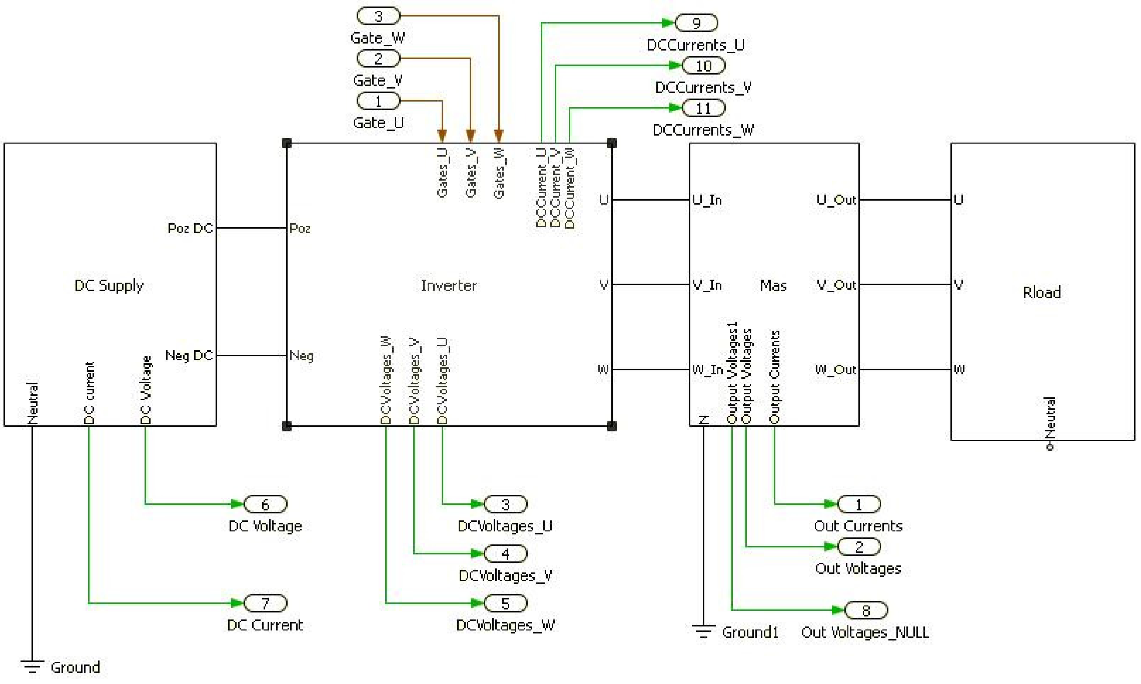


Figure 3-3:Main plecs model used for simulations

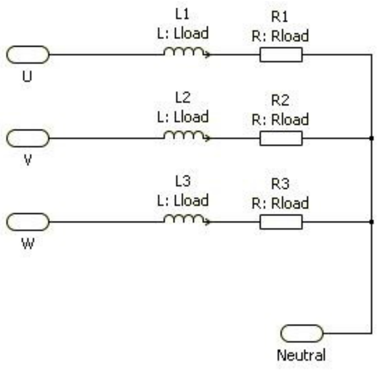


Figure 3-4: Simulations load

3.3 PHASE-SHIFTED MULTICARRIER MODULATION

Phase-shifted multicarrier modulation is a derivate of unipolar modulation. As such, this is an adapted version designed to suit PSI's requirements imposed by its topology. Namely, adjustments were made in connection with the number of carriers that increases with the number of voltage levels. The expression indicating the relation between the two is [14]:

$$n_{carrier\_number} = m_{nr\_of\_voltage\_levels} - 1$$

All carriers must have the same frequency and peak to peak amplitude. The angle by which any two juxtaposed carriers need to be shifted is given by [14]:

$$\varphi_{cr} = \frac{360^0}{m_{nr\_of\_voltage\_levels} - 1}$$

In this (3) case only one modulating wave, whose amplitude and frequency is generated by a control algorithm (e.g. for motor control, grid connection, etc.), is sufficient to be further a term of comparison for the carriers. Following the above, the number of triangular carriers needed for this inverter modulation is 8 ($v_{cr1}, v_{cr2}, \ldots\ v_{cr8}$) with a 45^0 phase displacement from one another. Every carrier controls one unit, specifically two IGBTs of a unit. They are being continuously compared with the modulating wave in order to generate the gate signals for the IGBT's. The frequency modulation index and amplitude modulation index are calculated as:

$$m_f = \frac{f_{cr}}{f_m} \qquad m_a = \frac{\widehat{V_{mA}}}{\widehat{V_{cr}}}$$

To be noted that phase shifted modulation the GBT's switching frequency coincides with the carrier frequency $f_{sw} = f_{cr}$.

In Figure 3-6, in the first plot are illustrated the carriers and the modulating wave corresponding to the U phase. At first sight it may seem that there are only four carriers although there are eight, they being overlapped two by two due to the fact that they are in the mirror. This is caused mainly because of the inverter's functionality constraints of having solely eight units active at all time (see (2)). Each time one unit from the upper part of a phase is conducting, its analog unit from the lower part of the same phase must be always turned off. Consequently there are four carriers controlling the upper unities of one phase, the other four being in opposition with the first by 180^0 leading to carriers overlap.

In the second plot of Figure 3-6, the IGBT pulses for the eight units of phase U are presented. This modulation scheme ought to provide a three phase sinusoidal voltage of adjustable frequency and amplitude with five voltage levels per phase respectively 9 voltage levels between phases at the output of the inverter.

To assess the modulation scheme efficacy, simulations were carried using Simulink and Plecs toolboxes from Matlab simulation platform. To simplify calculations, all quantities were transformed in per unit. In order to rigorously evaluate the inverter's response, simulations were run at 600 [Hz], 1200[Hz] and 2400[Hz] switching frequency. They can be found in Appendix B. The phase-shifted PWM algorithm was incorporated in an S-function block whose logic diagram is given in Figure 3-5. The diagram represents how the modulation works like inside the simulations.

The initialize counters will reset the counter to their initial values. Each time there is a sample time, the counters are incremented. When the counter had reached its maximum value, the sign will change and the counter will decrement until reaches 0. At this point the process is repeated all over again. During the counting process, when the reference gets bigger than the carrier, the corresponding unit of the carrier will start conducting until the reference drops under the carrier when the unit will be turned off. This is done by the determine state block. The IGBT state block determines which IGBTs from inside the units are switched.

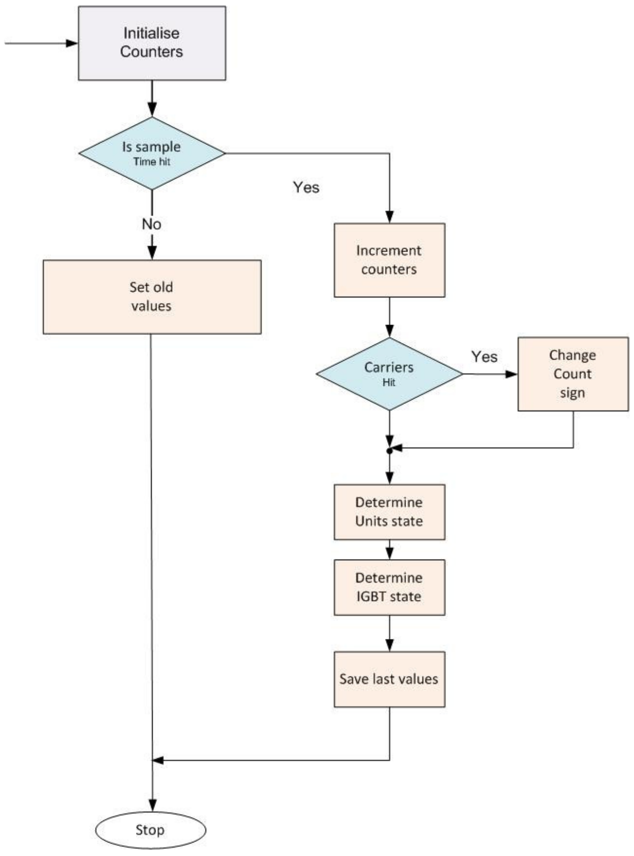


Figure 3-5. Phase Shifted PWM logic diagram

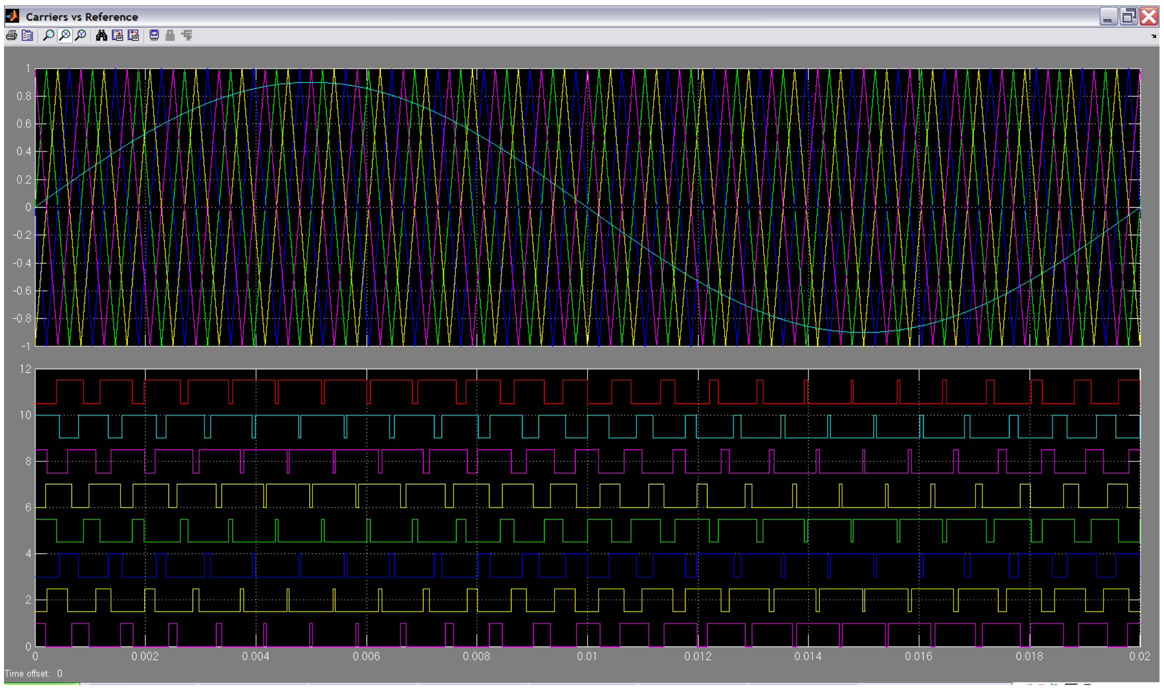
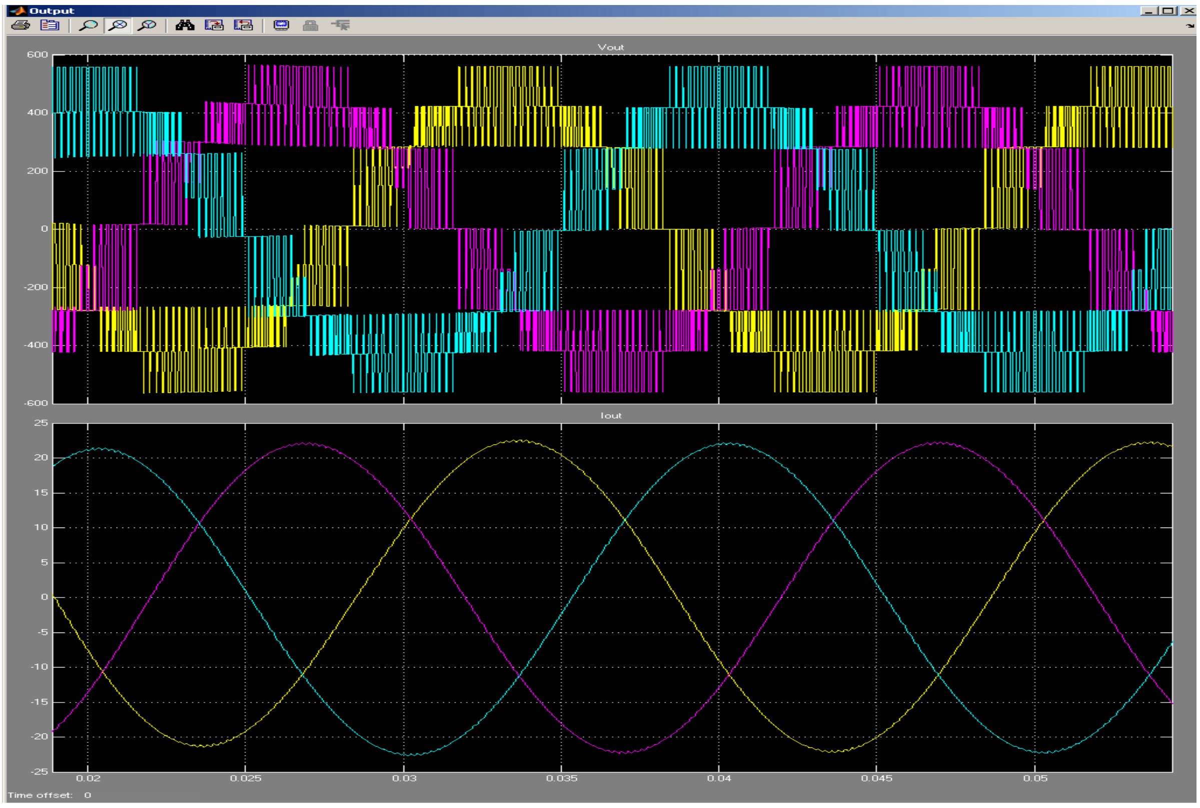


Figure 3-6. Carriers vs. the modulating wave and the unit states at $m_f = 48,\ m_a = 0.9$



3.4 LEVEL-SHIFTED MULTICARRIER MODULATION

This modulation scheme is similar to the previous in certain aspects. The required number of triangular carriers is calculated with the same expression as for the phase shifted modulation all having the same amplitude and frequency. The frequency modulation remains also unchanged. The difference is that here they are vertically disposed one after another, with the bands covering the whole [-1; 1] interval and the amplitude modulation index is redefined as [14]:

$$m_a = \frac{\widehat{V_m}}{\widehat{V_{cr}}(m_{nr\_of\_voltage\_levels} - 1)} \qquad \text{for } m_a \in [0,1]$$

Based on phase disposition level-shifted multicarrier modulations can be divided into the following three subcategories [15]:

- In phase disposition (IPD)
- Alternative phase opposite disposition (APOD)
- Phase opposite disposition (POD) – the carriers above below the zero reference have opposite phase disposition with respect to those above

All of these subcategories differ by the way the carriers are displaced. The displacement does not affect the amplitude or the frequency of the carriers. They can be in the mirror (POD) with respect to the 0 line, alternative (APOD) where one carrier is shifted by 180 degrees or in phase (IPD) where only the level differs among them.

The common for all these subcategories is the requirement of unit shifting. After each step, the units must be rotated in a cyclic way in order to prevent large difference between the unit voltages. This may represent a problem when compared with the staircase for example, because the rotation is only made at the end of one period of the fundamental and not during any switching step. From the losses point of view, the units are switched at the lowest rate possible and still be able to create a staircase similar output.

The number of carriers required for this modulation equals the number of units per leg.

The logic diagram for the implementation in Simulink is depicted by Figure 3-5.

The initialize counter block resets the counters to their initial value. While no sample time had occurred, the old values are sampled for output. When a sample time do occurs, the counters are incremented. When a counter reaches the maximum position it is decremented until it reaches 0, point in which the process will restart with incrementation.

When the reference value gets bigger than the carrier the corresponding unit will be switched on. For the opposite situation, when the reference value is lower than the carrier, the unit will be switched off. This will determine the actual unit states. The IGBT states are determined from the unit states.

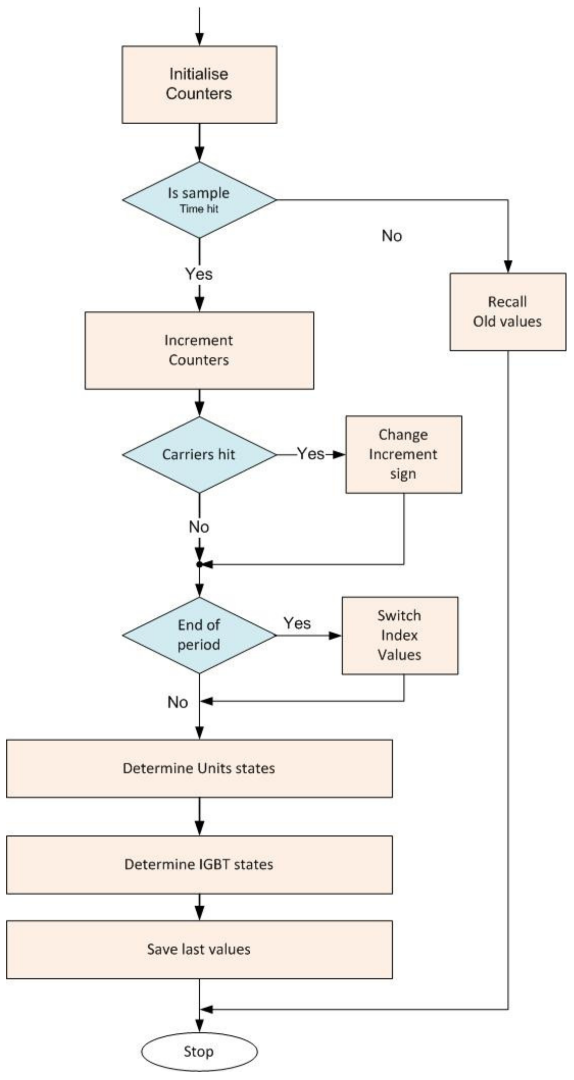


Figure 3-8. Level Shifted PWM logic diagram

3.4.1 IN PHASE DISPOSITION (IPD)

The in phase disposition, or IPD, is based on a single carrier that is multiplied across the entire voltage range. The difference between any two carriers is represented only by the voltage offset, offset which represent the actual step size of the modulation.

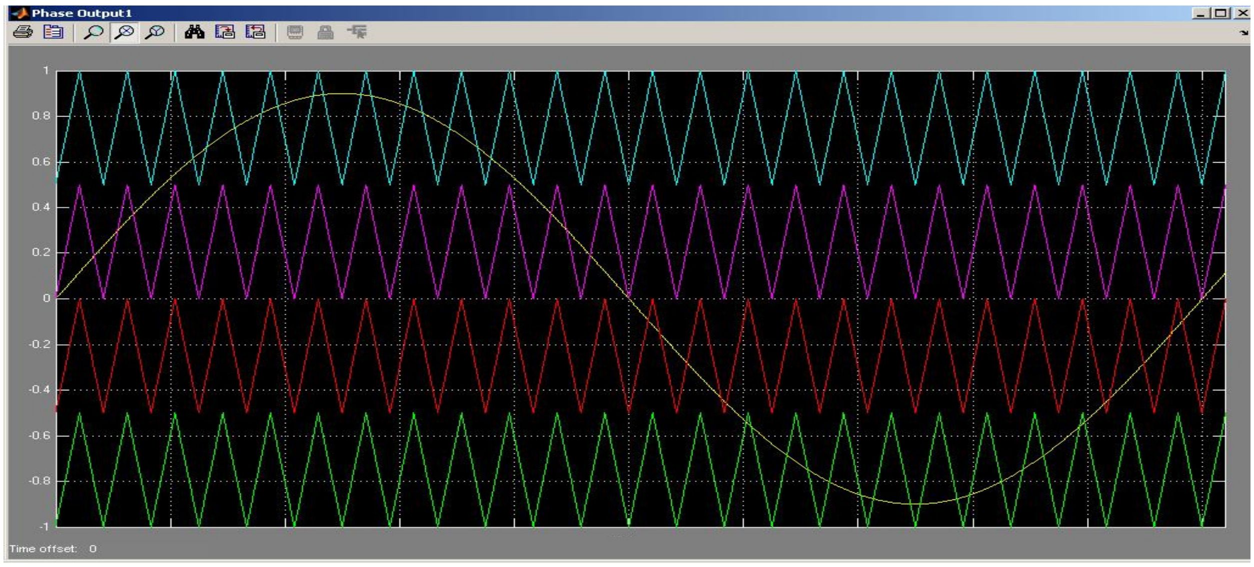


Figure 3-9. The carriers and the modulating wave – in-phase level shifted modulation

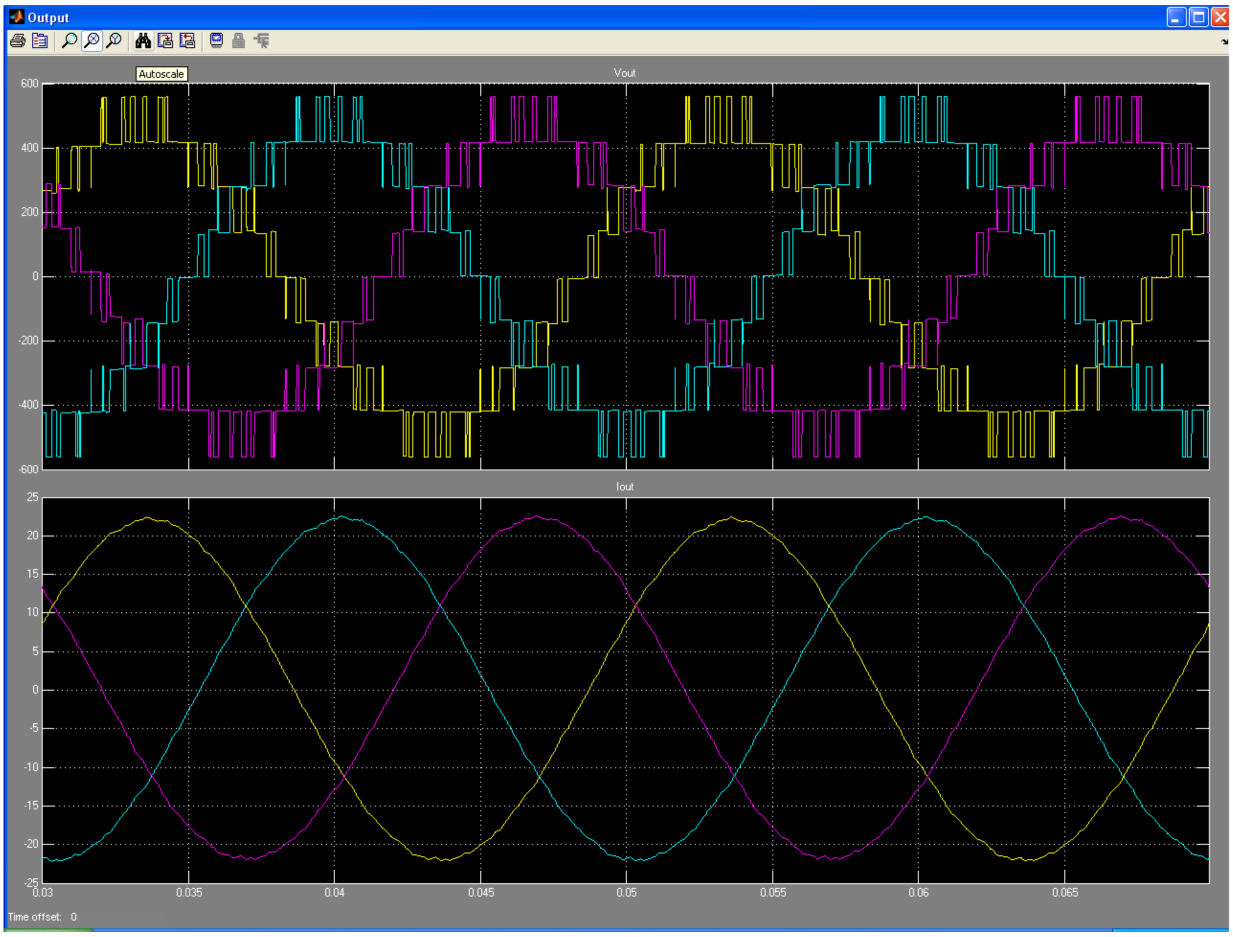


Figure 3-10 The output voltage and current

3.4.2 ALTERNATIVE PHASE OPPOSITE DISPOSITION (APOD)

The alternative phase opposite disposition, or APOD, is based on two carriers that have varies in the initial starting voltage level and phase. These two carriers are then multiplies consequently over the entire voltage range.

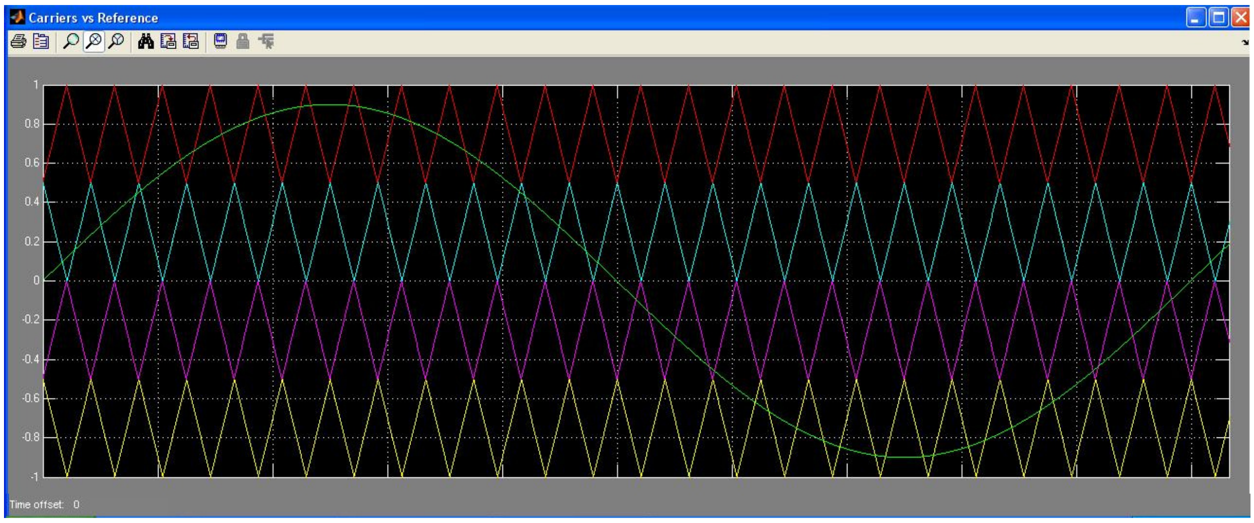


Figure 3-11 Triangular carriers vs. modulating wave – alternative phase opposite

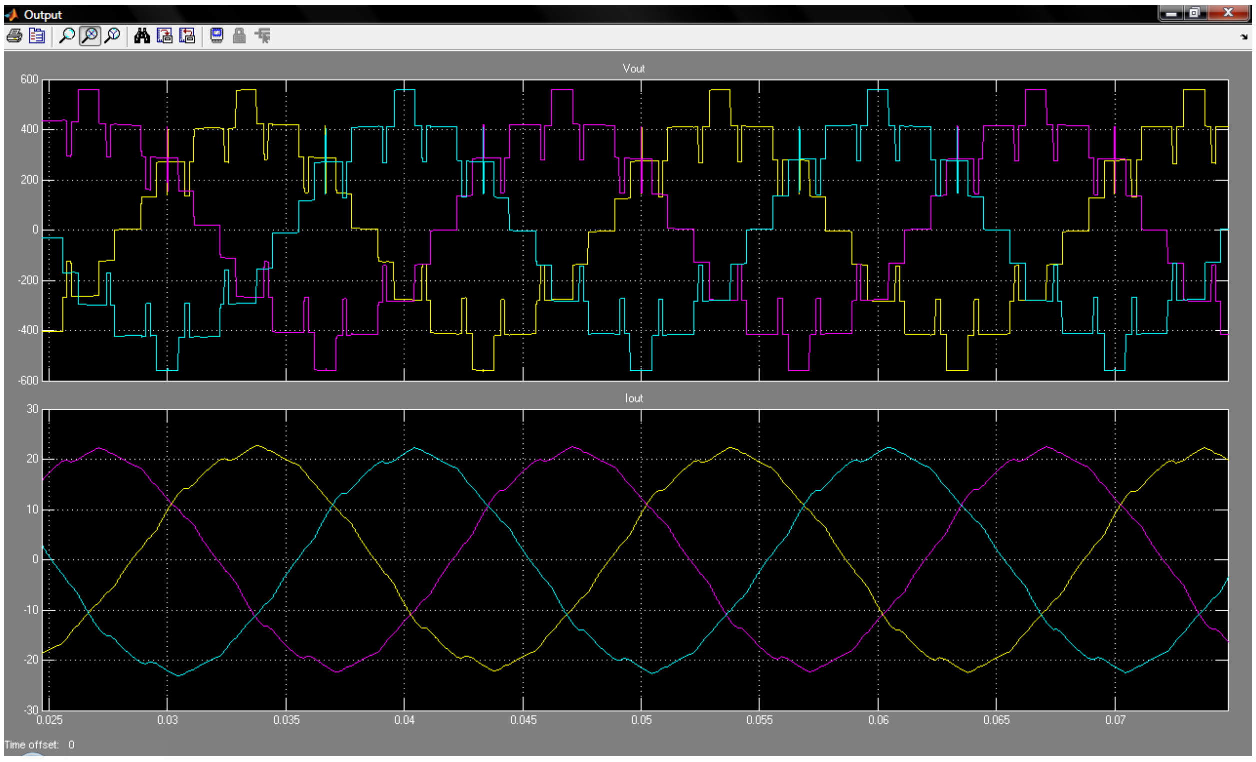


Figure 3-12 The output voltage and current

3.4.3 PHASE OPPOSITE DISPOSITION (POD)

The phase opposite disposition, or POD, uses two carriers, one for the positive voltage levels and one for the negative voltage levels. The negative voltage levels are shifted by 180 degrees with respect to the carrier for the positive voltage levels. The carriers are multiplied for their corresponding voltage level sign in order to fill the entire voltage range.

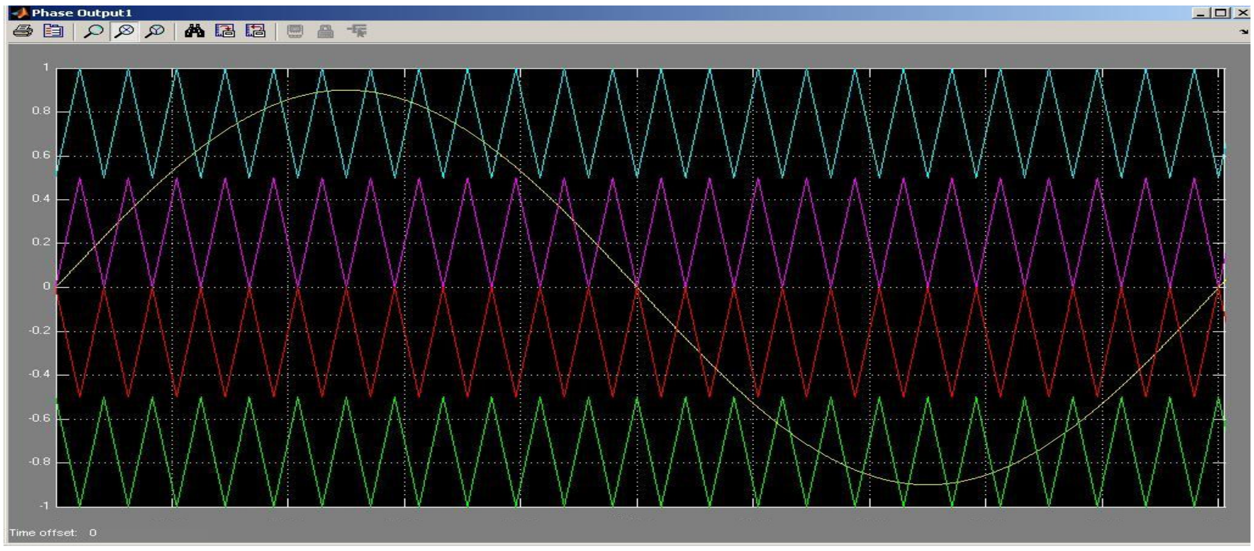


Figure 3-13 The carriers and the modulating

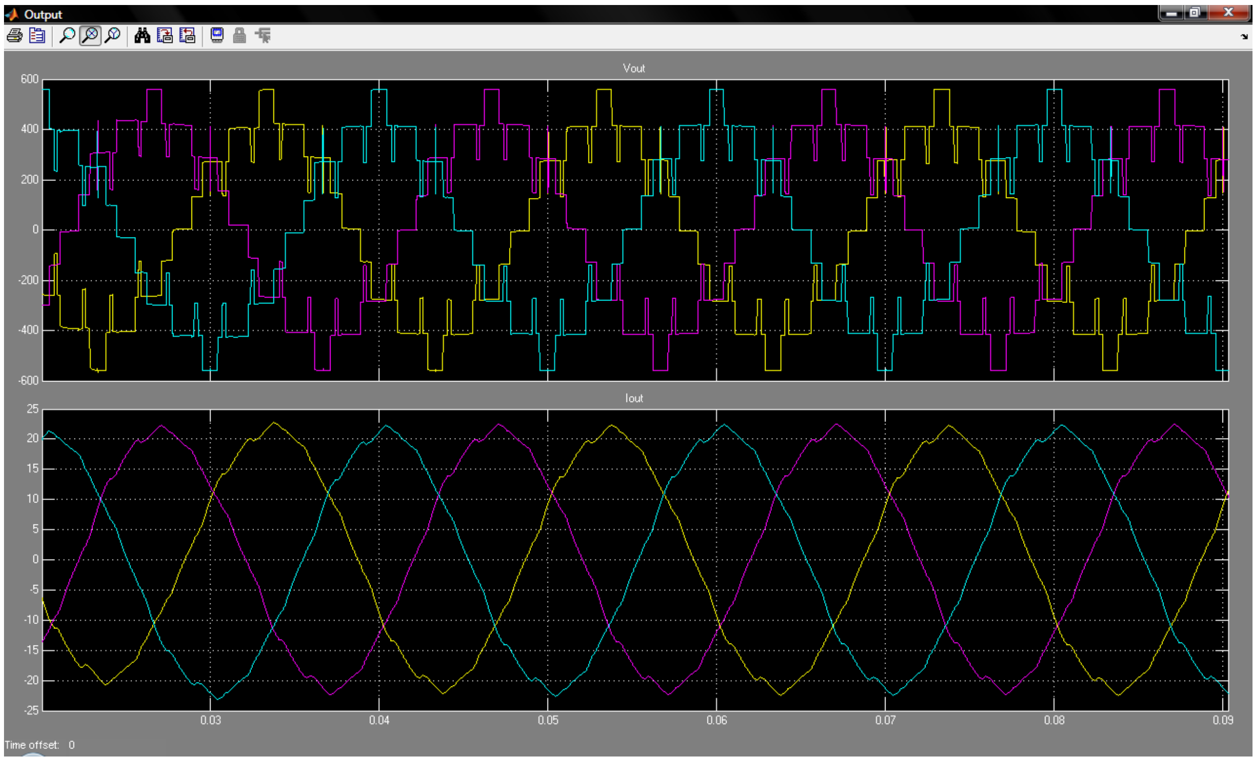


Figure 3-14 The output voltage and current

3.5 STAIRCASE MODULATION

Essentially, as the name indicates, the staircase modulation generates an output waveform that follows a staircase pattern. The output wave form is created based on proper selection of the units to be turned on. The selection is made with respect to the reference voltage as well with the help of a set of predefined voltage levels.

The reference voltage is similar with that used for the other modulations while the preset voltage levels are determined based on a coefficient (τ) and on the DC bus voltage. The voltage level is calculated for each individual step. The voltage level is calculated based on the following formula:

$$V[x] = \tau * \frac{VDC}{n_{units}}$$

Equation 3.1: Voltage level calculation

Where:

$V[x]$	– voltage threshold for the X level
τ	– coefficient
V_{DC}	– The DC bus voltage
n_{units}	– Number of units per leg

The coefficient role is to adjust the level when the next step will be applied. The coefficient for all the simulations made for this application was set at 0,5. This value represents half step. When the voltage reference gets bigger than one threshold value, the inverter will jump to the voltage level represented by the threshold value.

Similar with level shifted modulation a unit shift is required in order to maintain a constant voltage across the units. Because the output waveform of one leg can be generated by switching among the units of the leg a voltage balancing based on sorting can be implemented. The sorted values represent the voltages across the units of that particular leg. By measuring the voltage across one unit, at the next step the units with the highest voltage across their DC bus will be switched ON.

The logic schematic of the algorithm is presented in Figure 3-15.

The initialize steps block sets the internal parameters to their predefined values (eg. The steps are calculated).

The “is sample hit” block determines if the current sampling time is equal or bigger than the preset sample time. If the sample time hit did not occurred the previous values are outputted. For a sample time hit, the current level is determined based on the preinitialized steps. The level, which can be an integer value between 0 and the number of units per section gives the number of units turned on for the upper section. The number of switched on units for the lower section is calculated as the difference between the number of units per section and the level number. [13]

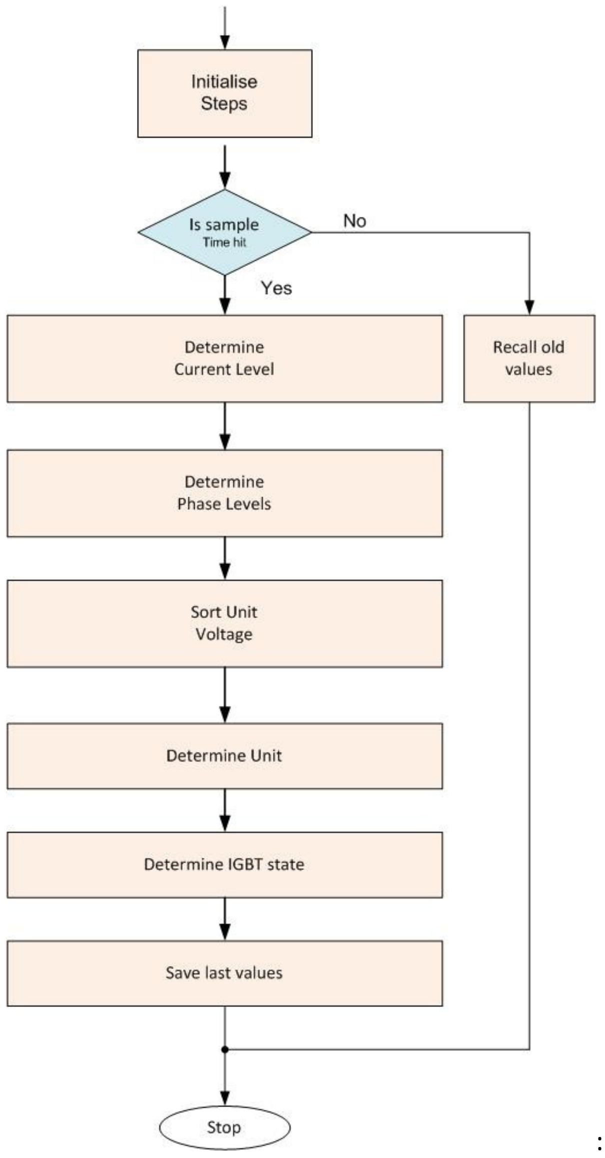


Figure 3-15. Logic scheme of the staircase modulation algorithm

The determine phase levels block determine the quadrant for which the calculation are made. The quadrant is determined from the current, the voltage being known.

The sort unit does a sort on the lower and upper sections of the leg individualy. The results are then passed to the determine unit block which determines the units that need to switch based on the sort result and the phase level.

The determine IGBT block calculates which IGBT will actually be switched. After this block the values are saved to be used for the next step until the sample time will be hit again.

3.6 CONCLUSION

The harmonic content of the output voltage waveform is an important criterion for evaluating the quality of the output voltage.

- The number of switching per modulation cycle is dependent of the carrier frequency in the case of PWM based modulations.
- The synthesized output voltage has a low content of harmonic distortion for all modulations schemes, even at low frequencies.

- Tabel 3-1 Harmonics analysis for 600 Hz

Method		**3^{th}Harmonic**		**5^{th}Harmonic**		**7^{th}Harmonic**		**Total THD**	
		Voltage [%]	Current [%]	Voltage [%]	Current [%]	Voltage [%]	Current [%]	Voltage [%]	Current [%]
Phase Shifted Modulation		0.2	0.16	3	0.63	9	1.76	20	2.67
Level Shifted Modulation	IPD	0.8	0.18	15	0.48	15	0.6	15	0.9
	APOD	0.4	0.01	18	0.01	18	1	20	1.2
	POD	0.02	0.025	7	0.05	7	0.02	7	1,2
Staircase Modulation		0.02	0.6	7	0.3	7	1.1	7	2.3

-

- Tabel 3-2 Harmonics analysis for 1200 Hz

Method		**3^{th}Harmonic**		**5^{th}Harmonic**		**7^{th}Harmonic**		**Total THD**	
		Voltage [%]	Current [%]	Voltage [%]	Current [%]	Voltage [%]	Current [%]	Voltage [%]	Current [%]
Phase Shifted Modulation		0.2	0.06	16.7	0.02	10	0.2	6.8	4
Level Shifted Modulation	IPD	0.4	0.6	2.2	0.2	1	0.2	6.8	4
	APOD	0.2	0.1	2.4	0.6	4.5	1	7	1.2
	POD	0.4	0.02	4.16	0.58	9.46	1.89	26.4	2
Staircase Modulation		0.03	0.008	7	2.4	6.28	1.35	24	4.65

- Tabel 3-3 Harmonics analysis for 2400 Hz

Method		**3^{th}Harmonic**		**5^{th}Harmonic**		**7^{th}Harmonic**		**Total THD**	
		Voltage [%]	Current [%]	Voltage [%]	Current [%]	Voltage [%]	Current [%]	Voltage [%]	Current [%]
Phase Shifted Modulation		0.03	0.002	18.6	0.06	12.82	0.03	25.4	0.08
Level Shifted Modulation	IPD	0.4	0.2	4.87	0.5	3	0.8	10	1.3
	APOD	0.3	0.16	5	0.5	3	0.68	10	1.3
	POD	0.l2	0.2	2.35	0.62	9	1.78	20.6	2.7
Staircase Modulation		0.025	0.005	6.95	2.4	6.28	1.35	24	4.65

4 HARDWARE IMPLEMENTATION

The hardware chapter describes the hardware that was done supplementary to the last project without taking into consideration the boards or wires that were multiplied, modified or replaced. Compared with the last project where a inverter single leg was realized. A set of more than 20 new boards have been multiplied in order to allow the new 2 legs of the inverter to function properly. The new hardware relates to power supply of the inverter, protections and new relay control logic. The FPGAs connectivity is not presented in this chapter due to connectivity and can be found in the software chapter. The schematics for this chapter are located in Appendix A.

4.1 ABOUT THE HARDWARE

The hardware implementation is based on the use of 24 pairs of 2-pack IGBT from Semikron: SKiiP 2 and 3 type modules both rated 1200 V. The modules have the gate driver on-board so there was no need to make a gate driver board. The connection with the FPGA is made through a set of cables over differential lines. The units are powered from a 24 VDC source for the SKiiP 3 units and from a 15 VDC power supply for the SKiiP 2 units.

The differential cables helps in reducing the noise caused by the IGBT switching. The IGBTs gate driver does not support differential signals so a board was created for the interface. The interface board used was designed for the previous project (see [13]) and for this project only multiplication were made.

The SKiiP 2 based units, shown in Figure 4-1, are connected in parallel with a Toshiba 1200V/150 Amps IGBT pack. The Toshiba IGBT pack has no on-board IGBT driver so an Skyper 32 gate driver was used to control them. The interface for this type of gate driver is identical with that used for the rest of the gate drivers. The units do not have a voltage sensor on board so an external voltage sensor was used. The SKiiP 2 units are rated 1200 V and 1200 Amps

The SKiiP 3 based units, shown by Figure 4-2, are connected back to back in order to obtain the required unit configuration. These units come with a voltage sensor on board so no external sensors were required. The SKiiP 3 units are rated 1200 V and 2400 Amps.

In both cases the connection were made with aluminum boards and discharging resistors were placed as well as decoupling capacitors on the IGBT module pins.

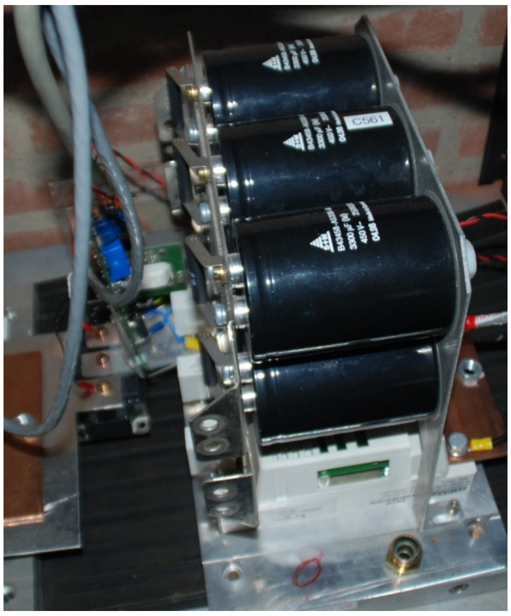


Figure 4-1: SKiiP 2 based unit

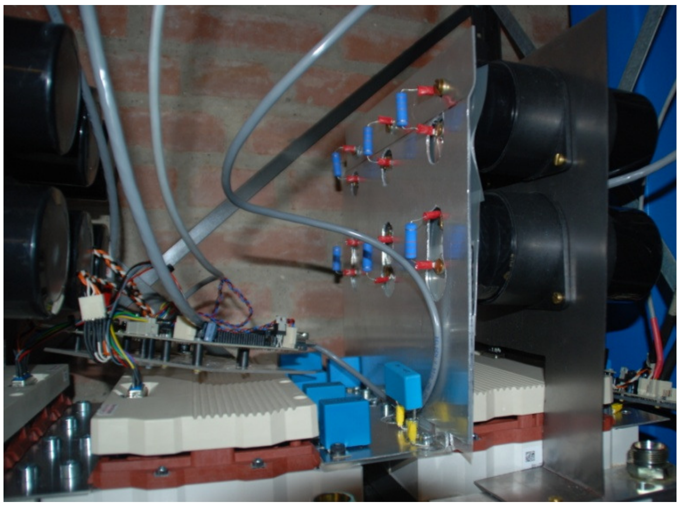


Figure 4-2: SKiiP 3 based unit

The gate drivers are controlled by the FPGA via a set of CPLD (a Xilinx CPLD XC95108). The basic schematic of the subsystem is presented in Figure 4-3. The CPLDs are used in order to allow the large number of units to be controlled by the FPGA. The communication from the FPGA to the CPLD is serial at 31 MHz in order to be converted into parallel communication for gate driver control. Because the communication is made over differential lines a set of line drivers with 2 transmitters and 2 receivers embedded were used.

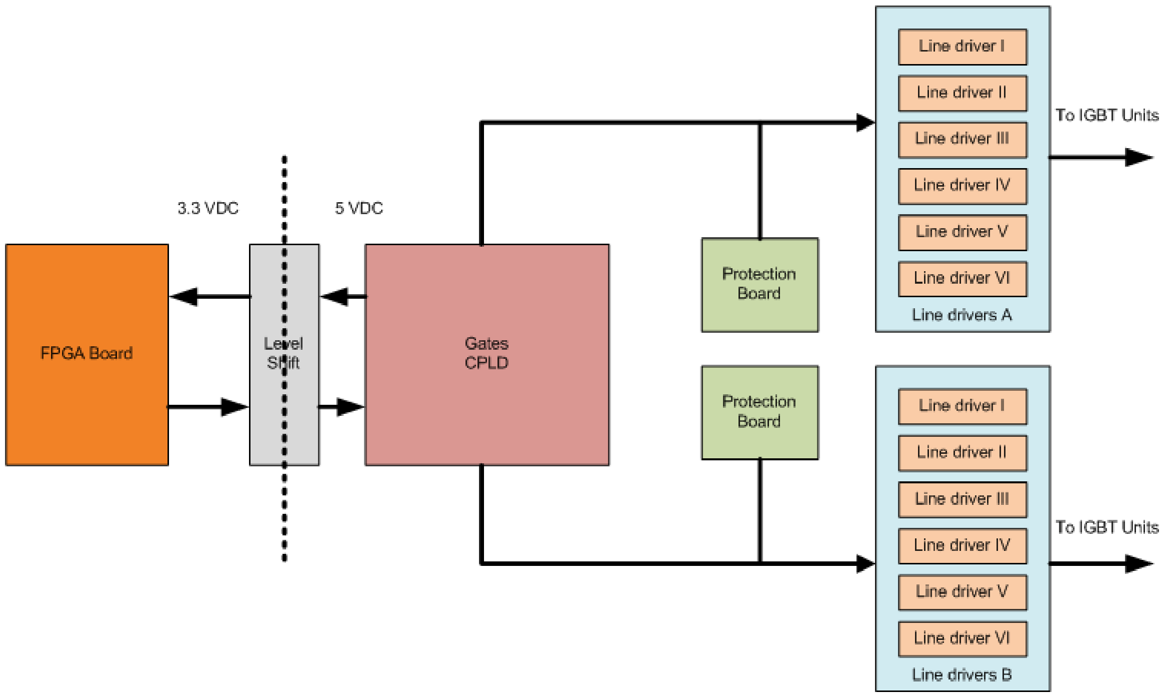


Figure 4-3: FPGA side gate driver control

The Figure 4-3 contains the FPGA side gate driver control with the line drivers represented. The FPGA runs at 3,3 VDC and a level shifter was also required. The protection boards helps protect the CPLD from overvoltages that can be caused by the line drivers. The delay time from the FPGA to the actual gate driver is less than 1 µs in normal operation while in case of fault, the response time is 10 ns. The response times are strictly hardware related and were measured without taking into consideration the software implementation.

The main FPGA (shown in Figure 4-4) is used to control the entire system, the secondary FPGA and the DSP are based on the main FPGA. The FPGA is a Spartan 3A 1800 DSP while the secondary FPGA is a Spartan 3 AN 750. The DSP used is a TMS320F28335 with an incorporated FPU.

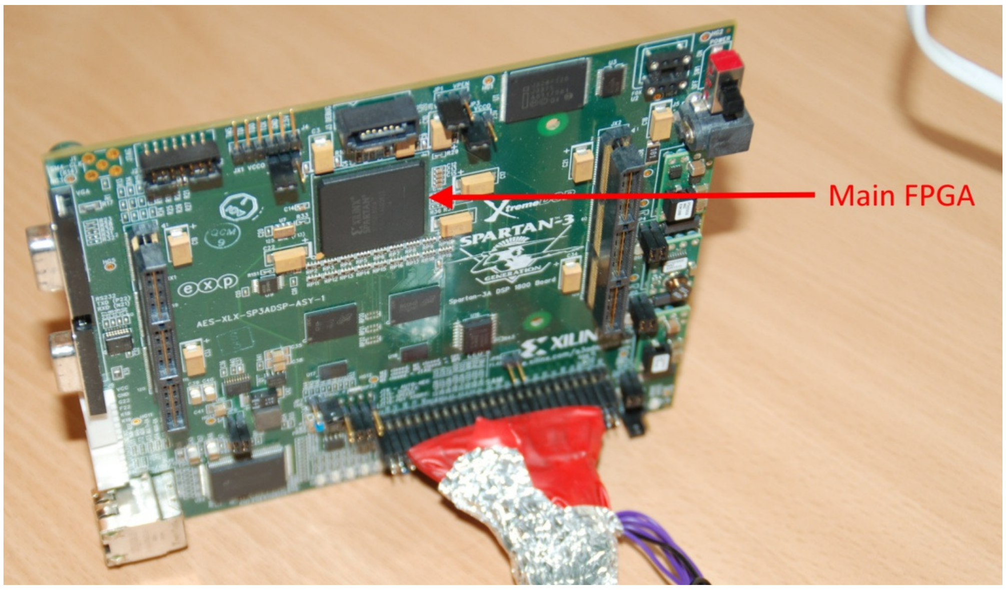


Figure 4-4: Main FPGA Board

The FPGA connect with a set of 6 ADCs made also for the last project and presented in [13] as well as the analog comparator boards. The ADCs can convert data at a speed of 2 MSPS and uses also serial communication with the FPGA. The 12 channels per ADC give the possibility of 72 channels to be acquired in less than 3 µs. The ADCs boards used are based on the AD7266 made by Analog devices.

Besides the main control elements of the inverter there is a set of other components and submodules that were designed for this project like:

- DC power supply board with overvoltage protection for inverter DC bus
- Relay control board in order to control the system contactors by intermediary relays
- Overvoltage protection board
- ADC gain amplifier board for the SKiiP 3 units

The supplementary SKiiP 3 based IGBT units were also assembled by mounting capacitors on them as well as discharging resistors and decoupling capacitors.

After all the hardware boards were made and the units constructed more than 100 meters of cable was used to connect the units with the main control units. The cable was mostly shielded in order to protect the signals from noise. Twisted pair cable was used for the IGBT gate drivers communication as well for the analog signals from the SKiiP 3 based units.

4.2 RELAY CONTROL

The relay control board handles the control of the relays, monitors the contactors and distributes the signal from the FPGA to the comparators CPLDs. The board contains 2 microcontrollers one for the relay control and the other one for the contactor monitor. The board is described in the following subchapters with respect to the functionality of each submodule.

The relay control board is shown in Figure 2-1.

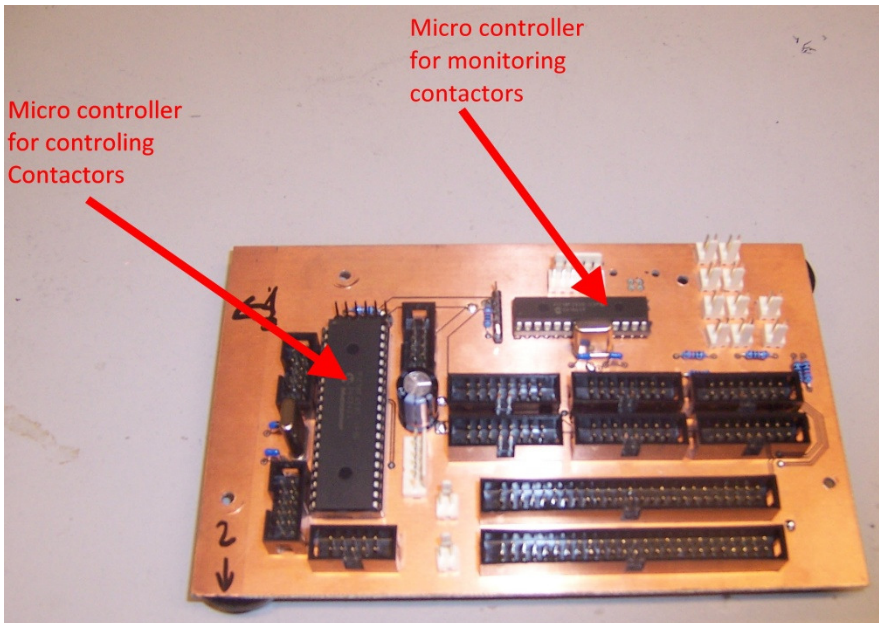

Micro controller for monitoring contactors
Micro controller for controling Contactors

Figure 4-5: Relay control board

4.2.1 RELAY OUTPUT

The relays used in this application are controlled by a microcontroller from Microchip PIC18F4580. The microcontroller interfaces with the secondary FPGA thought a SPI interface. The SPI interface on the microcontroller is embedded into hardware so no special protocol software needed to be implemented besides the SPI control.

The microcontroller can control up to 24 relays from which only 20 relays are used, the other 4 being auxiliary relays or reserves. The relays are divided into 3 categories:

- Unit power supply
- Auxiliary

- Power supply relays

The unit power supply relays help in controlling the power up the units in group of 4. This power-up is required due to the large inrush current of the switched mode power supply of each unit. Therefore the decision was taken to power-up a group at a time. The delay between two consecutive groups power up is 3 seconds. The power up can be done automatic by the "Automatic control of units power up". A "manual" power up of units is also implemented in order to allow FPGA control over the relays states.

The auxiliary relays, as mentioned earlier are considered reserved. They are going to be used only in case of failure for other relays.

The power supply relays are used to control the actual behavior of the power supply. The relays control in turn the contactors which are powered from 220 VAC. The relays offers galvanic insulation as well as the possibility of same level of logic in order to be able to `OR` or `AND` two signals. The relays for the main contactor as well as the relay for the chopper are controller by two signals as the system schematic from appendix A shows.

The microcontroller software used to control the relays is divided as well into several part. The section that deals with units power supply contains the automatic power up and the manual override.

The command decoder also stores the mapping data of the relays. If a relays malfunctions, the software can remap the corresponding relay to another one without reprogramming the main software. The remap is done over the SPI interface by the secondary FPGA.

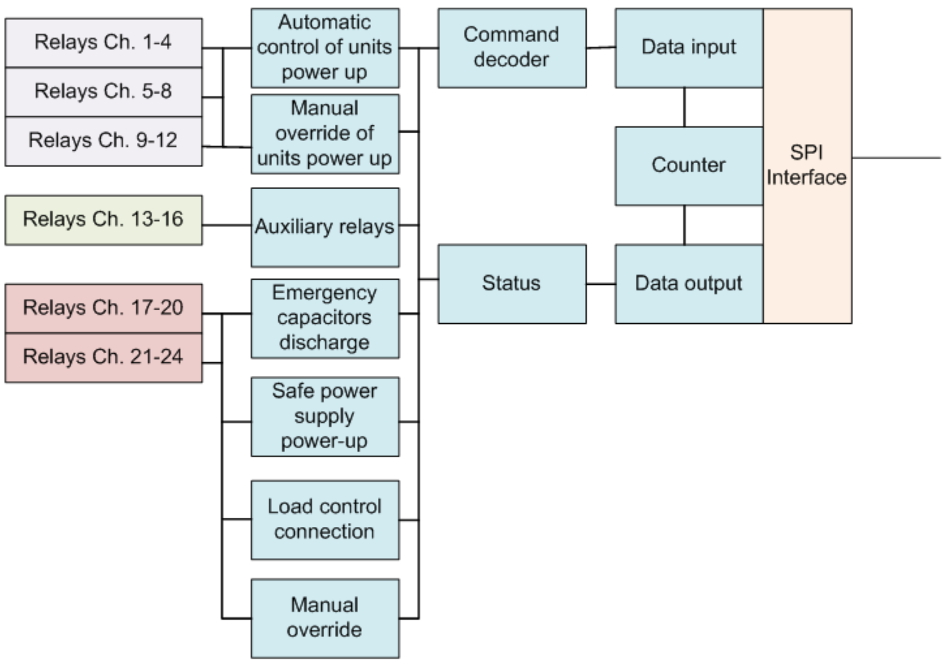


Figure 4-6: Relay output control

4.2.2 RELAY INPUT

The relay input board receives the states from the contactors with the help of auxiliary normal open contacts of the contactors. The signals are then send to the secondary FPGA in order to determine the actual states of the contactors. The contactors may fail to open/close or can be commanded by the overvoltage protection. In order to determine the actual state of the inverter, the relay input board is used in direct connection with the relay output board.

The relay input control is implemented on a microcontroller from Microchip PIC18F2550 which also provides to possibility to connect to a computer via a USB connection. The connection is only provided and no code was implemented on it.

The communication with the FPGA is done over a SPI connection where the microcontroller is the master. This solution was selected because the FPGA number of connectors is limited and the amount of connections required is quite large.

The diagram which represents the main blocks used by the relay input control is shown in Figure 4-1. The command decoder is also a command encoder. Its task is to determine what does the FPGA requires and also to prepare the data for sending.

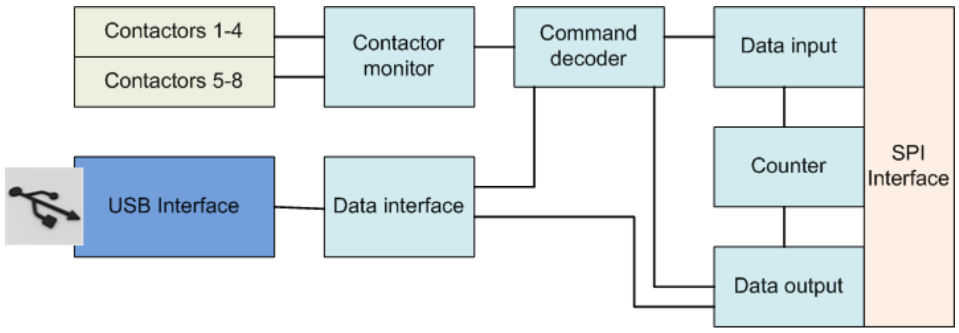


Figure 4-1: Relay input control

4.2.3 SIGNAL DISTRIBUTION

Besides the relay control, the board also splits the connection between the secondary FPGA and the analog comparators CPLD. The board also handles the fault and reset signals from the system by providing access to a set of open drain lines one for reset and the other for fault signals. To these lines, devices are connected in order to trigger faults, devices like the overvoltage protection board or the DSP.

4.3 OVERVOLTAGE PROTECTION BOARD

The overvoltage protection board is placed on the power supply and used to measure the DC Bus voltage. The voltage is measured with the help of a voltage divider and the output is then compared with a preset value. The ratio between the measured value and the output is 900:1 thus allowing a 1,8 kV to be measured safety. The limit does not come from the output comparator but from the resistor limits. The resistors are rated 150 kOhms and 350 V. Because there are 6 resistors, the measured voltage is divided between them therefore a resistor, at 900 VDC, will have 150 V. At 2 kV, the voltage drop on one resistor is 333 V. The power dissipation on one resistor is less than 1 W.

The measure resistor, the one on which the voltage drop will be measured, it is rated 1 kohm. The measured voltage is then passed to two comparators in order to produce two signals. The two signals are then passed to a fiber optic and the other one to optocoupler.

The fiber optic will offer insulation between the DC bus and the FPGA. The insulation between the overvoltage board and the command relay is assured by the supplementary relay. The optocoupler was intended for cable connection between other boards and for simplification it was connected directly a relay.

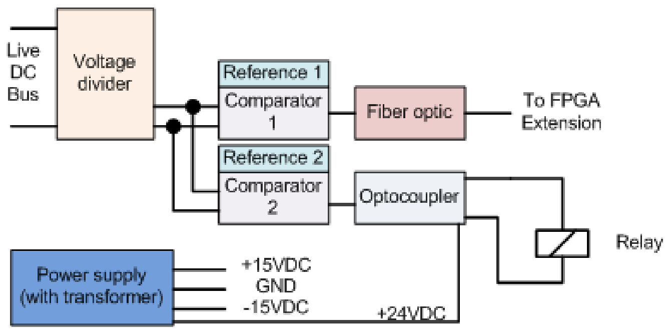


Figure 4-7: Overvoltage protection board

The basic schematic of the comparator control is depicted in Figure 4-8.

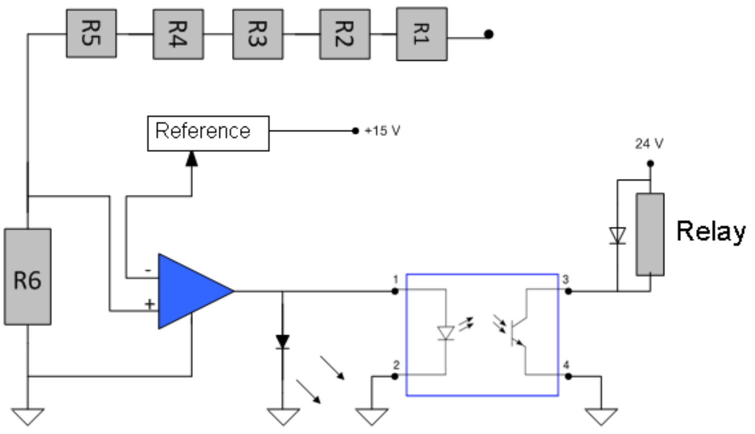


Figure 4-8: Basic schematic of the comparator control

The reference 1 and reference 2 are not the same value, the difference between the two voltages compared is about 50 V. The difference was set in order to allow the FPGA to take all the necessary precautions, if possible, to prevent the hardware protection to kick-in.

When the hardware protection is on, the relay is turned on, the main contactor is therefore switched OFF and the copper contactor is switched ON. All other contactors remain in their initial position and are not affected by the over-voltage protection board. For more details see Figure 4-9.

In the Figure 4-9 the voltage represents the measured voltage at the input of the board, the reference 1 is the reference for the optic fiber while reference 2 is the reference for the opto-coupler. When the voltage crosses over one reference the corresponding signal will be switched LOW. The low state is also for safety because, if the signal is missing then either the board is damaged or the system is not connected. Therefore the system has an active low control which implies that the relays normal-closed contacts will be used.

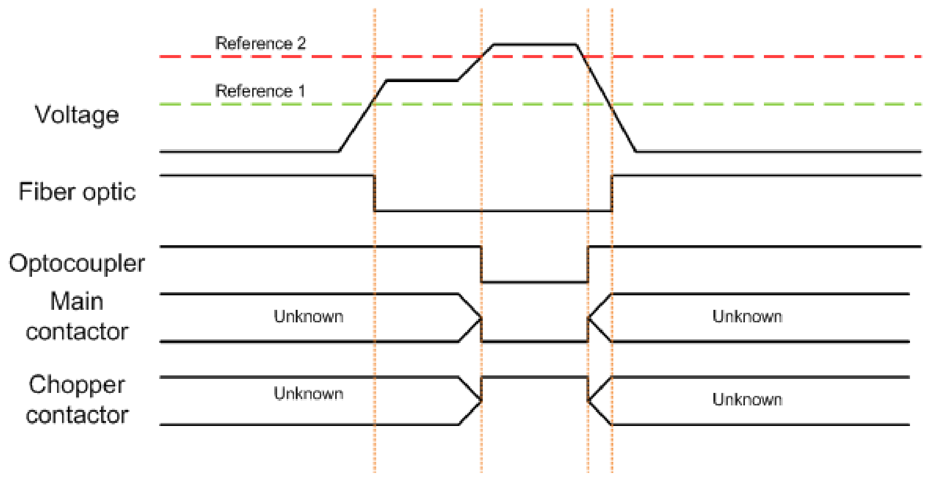


Figure 4-9: Over-voltage commands and timings

4.4 MAIN POWER SUPPLY

The power supply for the inverter had been constructed in order to have a regulated DC power supply capable of withstanding voltages up to 1,3 kV. Everything that the power supply incorporates permits working at such high voltages. Even if the power supply takes its main power from a 400 VAC grid, the high voltage protection was required in order to prevent hardware faults to occur when the unit's capacitors discharge in the DC bus. The 1,3kV can be reached inside the DC bus when by wrong control of the switches, the unit capacitors gets all connected in series with to the DC bus. The power supply contains the following relevant blocks:

- Rectifier
- DC Bus capacitors
- Capacitor charge control
- Chopper
- Overvoltage protection (with varistors)

The components are represented by Figure 4-10. The configuration of the DC power circuits of the supply together with the main contactors and connectors. The resistors colored in green represent the DC chopper resistors while the yellow resistors represent the charging resistors for the power supply capacitors. The diodes blocks are part of the rectifier while the block starting with letter K represents the contactors. The signals corresponding for the main contactors are handling by auxiliary relays located in the lower left of the figure. The actual construction of the power supply is represented by Figure 4-11.

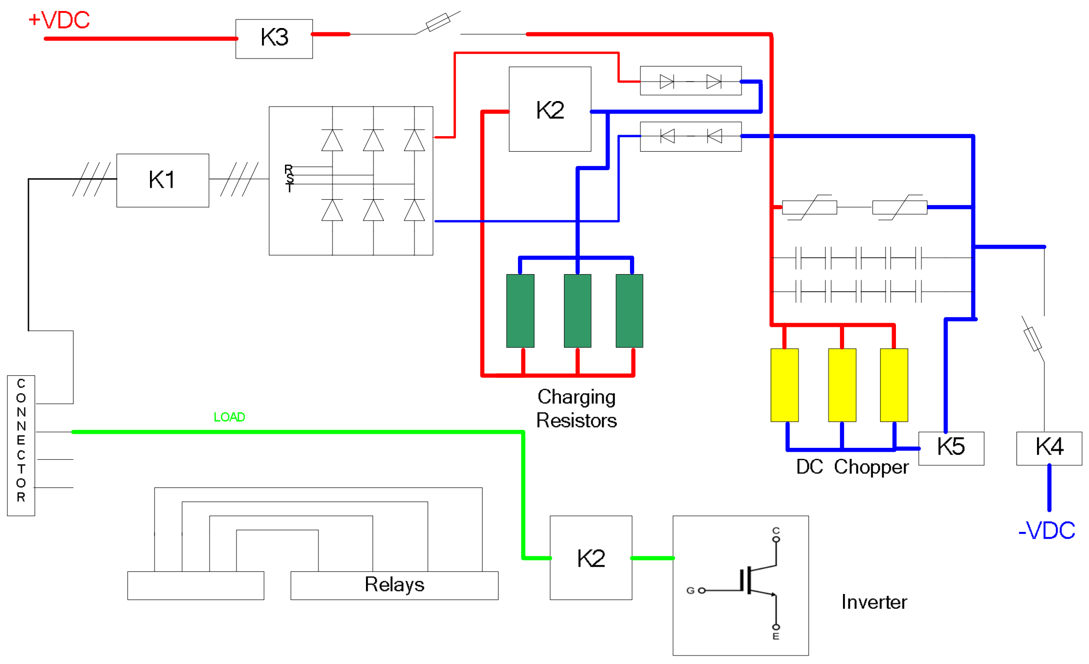


Figure 4-10: Supply configuration diagram

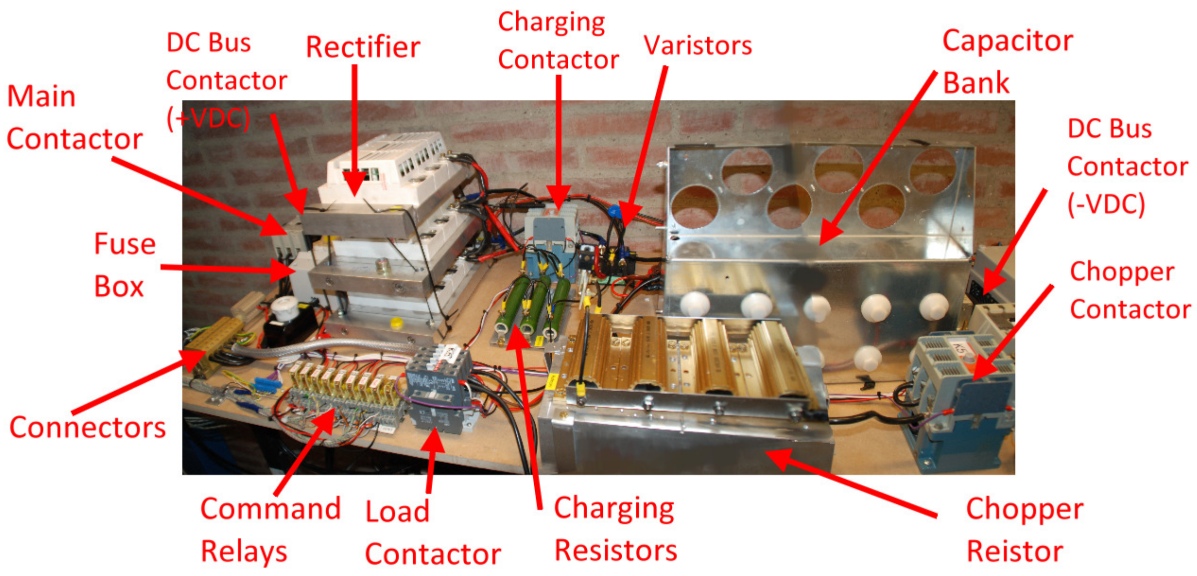


Figure 4-11: Power Supply and Main components

The main contactors are rated for 50 A and the contacts are connected in series in order for them to withstand high voltage inputs.

4.4.1 CONTACTORS

The power supply is also equipped with a set of contactors to allow the possibility to control the system. Therefore 6 contactors have been used, contractors with 690 voltage rating on each individual contact. The contactors used and their role is described better in Table 4-1.

Number	Description
K1	Main contactor is used to connect the power supply to the 400 VAC grid.
K2	The charging control of the power supply DC Bus. This contactor short-circuits the charging resistors thus taking the system to a normal operation mode.
K3	The contactors connected the positive DC Bus line with the positive line of the inverter.
K4	The contactors connected the negative DC Bus line with the negative line of the inverter.
K5	The chopper contactor will close down as soon as overvoltage event is triggered by one of the protection elements (voltage measurement or overvoltage protection board)
K6	The load contactor is responsible with connecting the load with the inverter.

Table 4-1: Power supply contactors roles

The complete power supply schematic can be seen in Appendix 1.

4.4.2 RECTIFIER

The rectifier differs from a normal rectifier due to high-voltage protections that have been required. Therefore instead of 6 diodes, the rectifier contains an additional 4 diodes in order to allow an increased voltage blocking level. The diodes are placed as depicted by Figure 4-12.

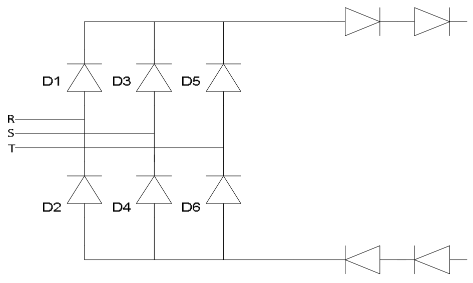


Figure 4-12: Bridge Rectifier

4.4.3 DC BUS CAPACITORS

The DC bus capacitors have been connected in groups of 5 in series and the groups have been connected in parallel. This type of connection ensures that the capacitors can withstand higher voltages and also provide a low voltage ripple for the inverter. The capacitor configuration is depicted by Figure 4-13.

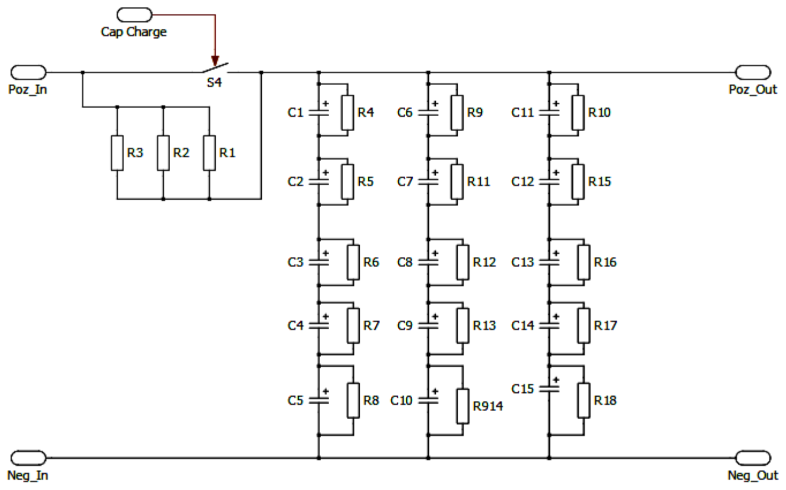


Figure 4-13: Capacitors configuration

The parallel resistors of each capacitor are 30 kOhms, value which allows the capacitor to discharge and provides a current path in case of fault when the contactors open. The resistors are rated 450 V and 5 W.

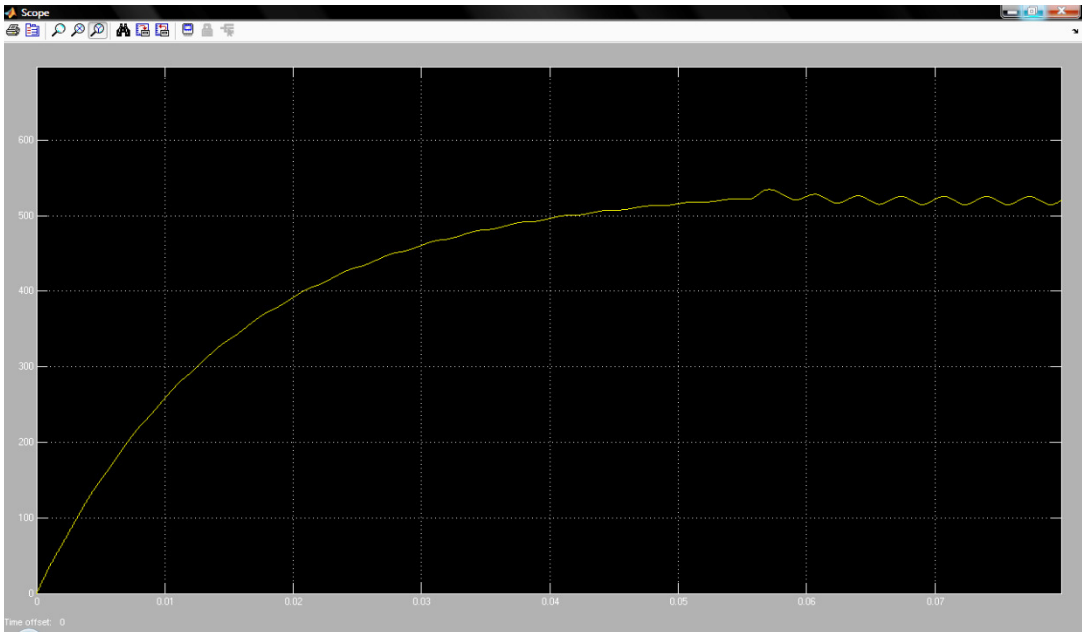


Figure 4-14 Charging rates for Capacitors

The capacitors are charged at startup through a set of charging resistors. The resistors are also used to charge the units DC Bus capacitors close to their nominal operating voltage in order to limit the inrush current. The resistors will be short-circuited with the help of contactor K5 when the charging is complete.

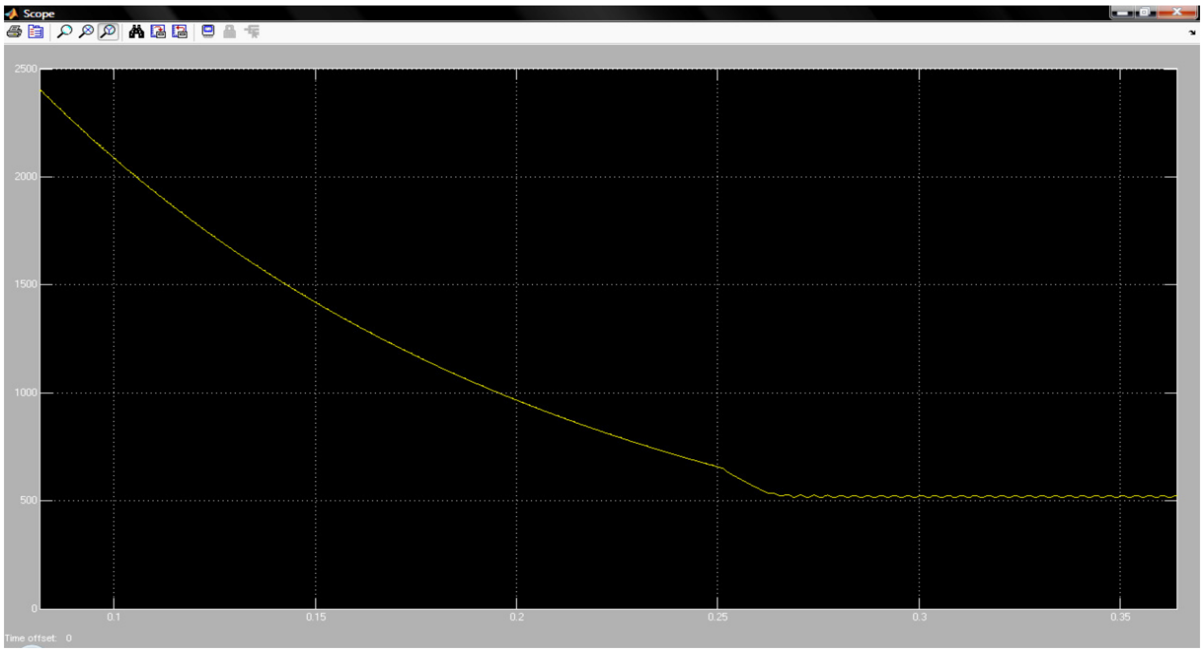


Figure 4-15 Discharging rates of the capacitors

The charge complete event is triggered by the FPGA which measures the unit DC bus voltages while the DSP measures. In order to properly charge the inverted units DC bus capacitors, the units must be controlled in order to provide time for all the capacitors to charge. Half of the number of units on each leg will be switched on, each at a time in order to allow the capacitors to charge.

4.4.4 THE DC CHOPPER

The DC Chopper is used to lower the DC Bus voltage in care of overvoltage. The voltage should be lowered fast enough in order to protect the main DC Bus capacitors. A normal DC chopper would use an IGBT and a certain chopping frequency but for this application, where the power supply was only made for this application because no other power supply was available, the decision to place a contactor seemed to be the simplest solution. The contactor will remain connected as long as the DC bus voltage is over the preset limit. During the time the DC Chopper is connected the main contactor is disconnected.

The discharge resistors are rated 100 Ohms at 200 W and are connected in parallel thus the total equivalent resistor will be 25 Ohms at 800 W. The DC chopper is depicted by Figure 4-16.

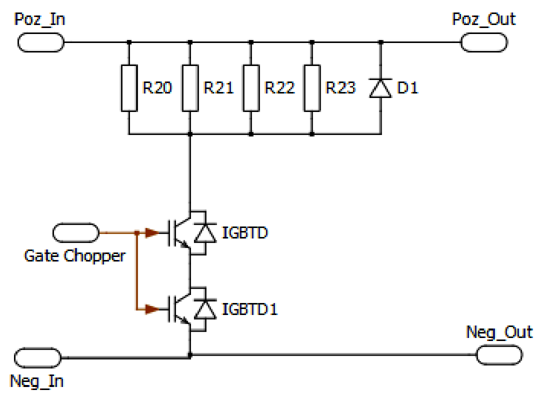


Figure 4-16 DC Chopper

4.4.5 OVERVOLTAGE PROTECTION

The overvoltage protection was implemented in two ways:

- with varistor
- with logic and measurement

The varistor is used to transform the voltage into a current, by lowering its resistance, in order to determine the fuses to break up. This method is the most appropriate for this power supply because no other cheap method will limit the voltage fast enough. The downside of this method is the fact that it will protect only once, after which the fuses must be changed. The fuses used are ultra fast fuses rated 50 Amps.

The logic method is not as fast as the one with varistor but the role is the same. This method uses a voltage divider to measure the voltage. The voltage divider has a ratio of 900:1. Therefore, it is capable to measure voltages up to 2 kV. The limit is imposed by the resistors which form the voltage divider. A resistor is rated 150 kOhms, 350 V. The 2 kV allows a limit of 330 V per resistor. The power dissipation on each resistor will be no more than 1 W.

4.4.6 THE VOLTAGE SENSORS CONNECTIONS

The voltage sensors are placed on the power supply board at 3 locations:

- Mains
- DC Bus
- Load

A total of 7 voltage sensors have been used. The main voltage sensors are used to detect the voltage level at the input in order to determine the charging state of the main DC bus.

4.4.7 THE AUXILIARY RELAYS

In order to control the high power contactors, several auxiliary relays are used. The relays are rated 24 VDC while the contactor coils are rated 220 VAC. The relays also provide a level of insulation between the two types of signals (24 VDC versus 230 VAC). Each relay is responsible for a single contactor except the main and chopper contactor which can be triggered also by the overvoltage protection mechanism. (see Over Voltage protection subchapter)

4.4.8 SIMULATIONS

In order to check the results also a simulation of the supply is made. The input parameter is the voltage from the gird. The simulation it is very complex it includes also models of the wire inductance and components that resemble to the real components. There were also included the contactors which were controlled by an S-Function with code located in Appendix E.

The Figure 4-17 shows the bridge Rectifier shows the simulated PLECS circuit:

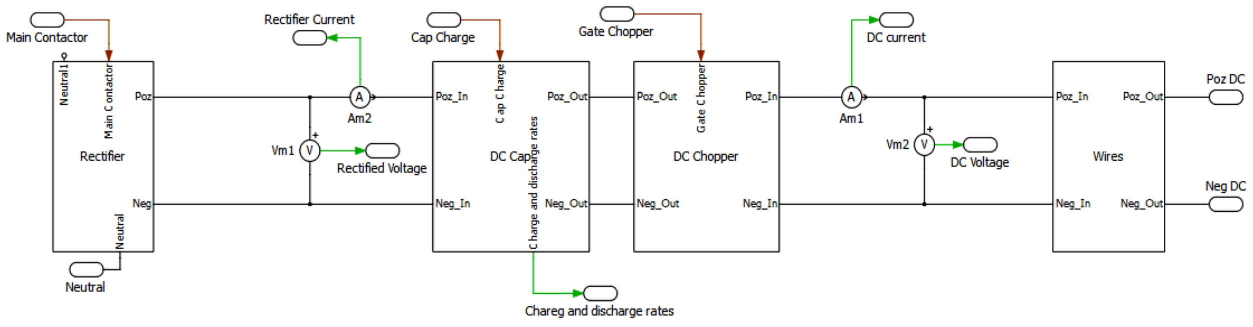


Figure 4-17 Simulated Circuit of the DC power supply

The simulation results are placed in the Appendix-D where there are displayed the wave forms corresponding to the output voltage and currents. The supply was loaded with a $R = 30\ \Omega$ resistor. The simulation of the power supply was made separately from the main simulation due to their combined complexity.

4.5 SWITCHED MODE POWER SUPPLY

4.5.1 DESIGN CONSIDERATIONS

It is very frequently required to convert unregulated DC voltage to a regulated output voltage at a certain voltage level or in many applications it is required to have more the one output with different voltage levels. The regulation is achieved by adjusting the on and off time of the switching element. A switch mode power supply provides that. In the project it is required to provide different voltage levels in order to supply different kinds of components for example the cooler and the small commands components [16]. Such a converter it is called a step-down converter or buck-converter. A basic schematic is described in Figure 4.18

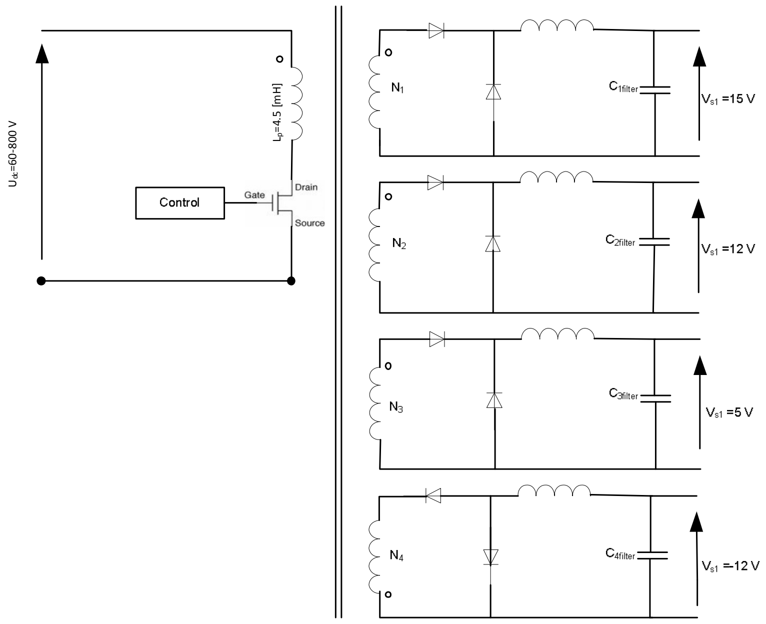


Figure 4.18 Switch Mode Power supply basic diagram

Basic SMPS parameters design and design considerations:

Input Voltage V_p=800 Vdc

Output Voltage:

V_{s1} =15 [V]	V_{rp1}=20m [V]	I_{01min}=0.250 [A]	I_{01max}=2 [A]
V_{s2}=12 [V]	V_{rp2}=20m [V]	I_{02min}=0.250 [A]	I_{02max}=1 [A]
V_{s3}=5 [V]	V_{rp3}20m [V]	I_{03min}=0.250 [A]	I_{03max}=5 [A]
V_{s2}=-12 [V]	V_{rp1}=20m [V]	I_{01min}=0.250 [A]	I_{01max}=1 [A]

No	Name	Symbol	Value	Minimum
1	Input voltage	V_p	800 VDC	60 VDC
2	Output 1- Voltage	V_{S1}	15 VDC	n/a
3	Output 1- Current	I_{01}	2 A	0.25 A
4	Output 2- Voltage	V_{S2}	12 VDC	n/a
5	Output 2- Current	I_{02}	1 A	0.25 A
6	Output 3- Voltage	V_{S3}	5 VDC	n/a
7	Output 3- Current	I_{03}	5 A	0.25 A
8	Output 4- Voltage	V_{S4}	-12 VDC	n/a
9	Output 4- Current	I_{04}	1 A	0.25 A

Table 4-2 Outputs of the SMPS

The design specification was placed in the Appendix C tougher with the specific steps that have to be followed. The calculation formulas were included also the results after the calculation.

4.5.2 EXPERIMENTAL RESULTS

In order to have a good power supply a basic schematic was constructed by using OrCAD and also to construct the PCB for the flyback converter. The basic schematic it is showed in the Figure 4.18 (from underneath)

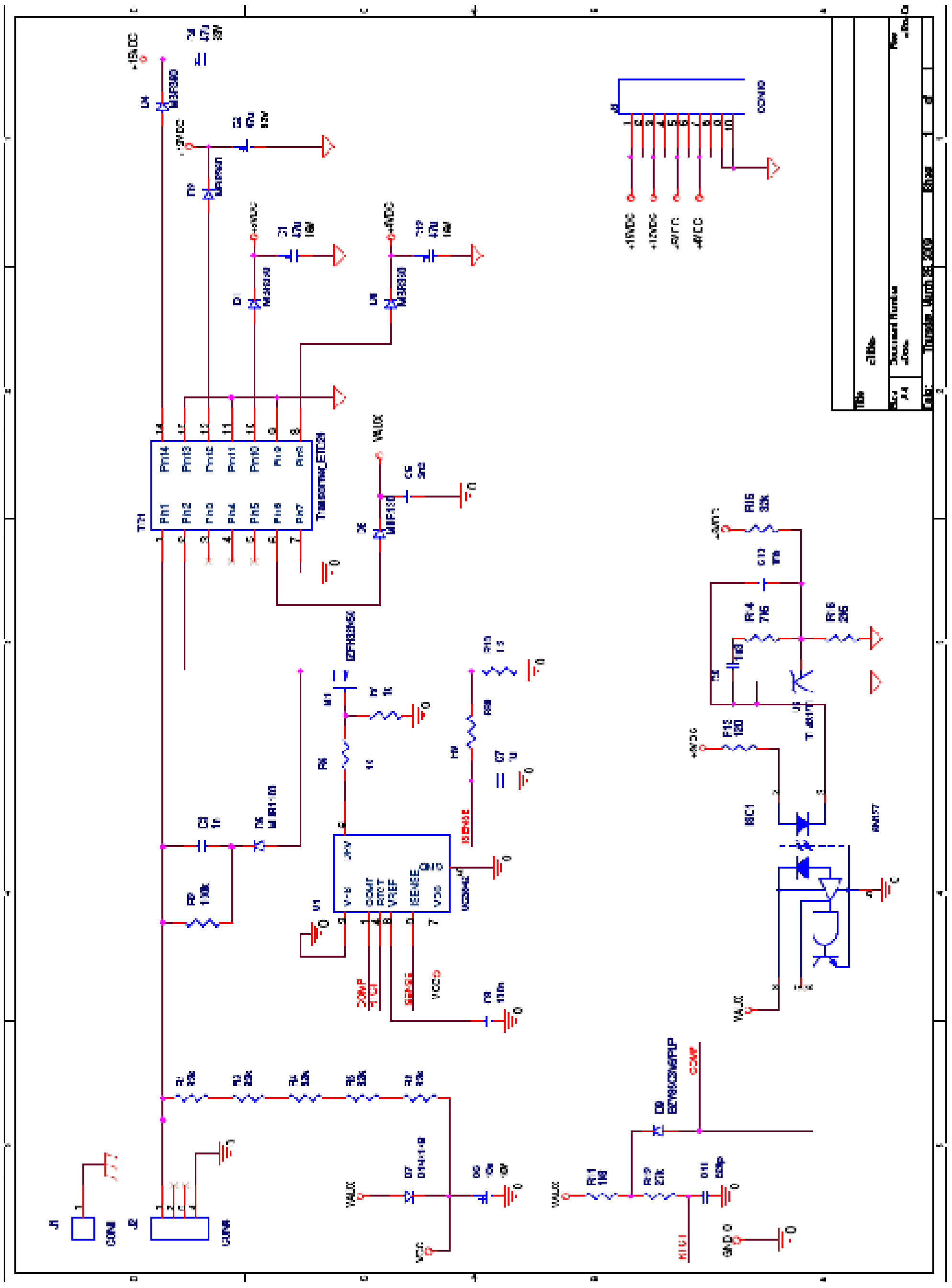


Figure 4.19 OrCAD schematic for Flyback Converter

The schematic is describing the principle of the flyback converter and basic components like the MOSFET involved in the circuit and the transformer used to lower the input voltage to the needed output voltage. The MOSFET has been chosen based on the input voltage and based on the needed output power of the converter. The transformer was selected to correspond to the chosen input voltages and also the output voltages and output currents. Taking in consideration the parameters, some consideration must be made for the circuit, the most important is the insulation between the circuits. This is important due to the fact that the primary circuit it is working with high voltage. Another important parameter in the process is the switching frequency of the converter so that the transformer and the filtering components can be made much smaller and lighter, leading to a low cost for manufacturing. For the transformer has been used ferrite core type ETD-34(Figure 4.20)

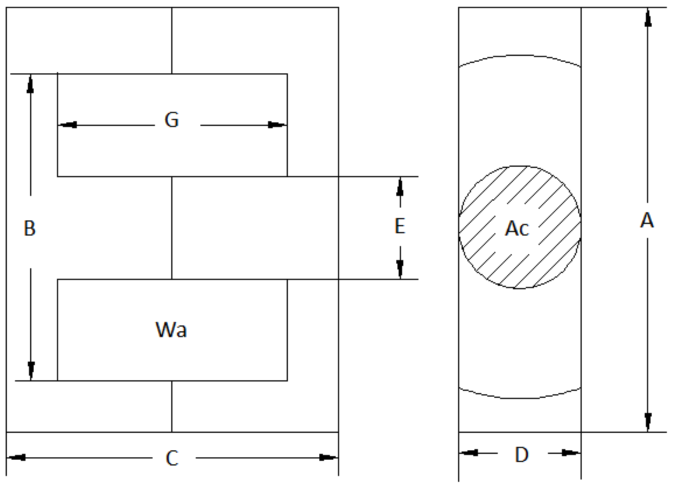


Figure 4.20 Dimensions for ETD Ferrites Cores

Part No.	A [cm]	B [cm]	C [cm]	D [cm]	E [cm]	G [cm]
ETD-34	3.5	2.56	3.46	1.110	1.110	2.36

Table 4-3 Dimensions data for ETD-34

In the Table 4-3 are presented basic parameters of core. The parameters represent the physical dimensions of the transformer core. The magnetic core plays an important role due to the fact that it stores energy for each conduction period.

After the design parameters where determined, the transformer was realised according to the calculation. The winding where placed on bobbin on core ETD34 given by design calculations. The Figure 4.21 show the construction of the transformer including even the placement of the windings.

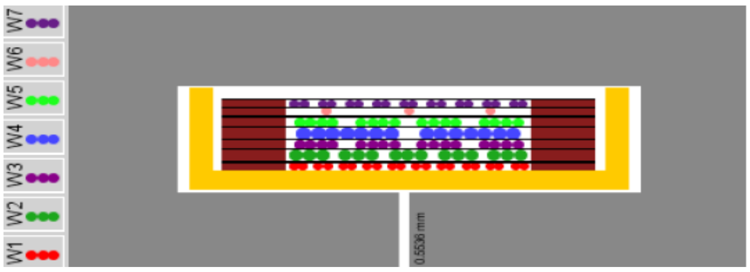


Figure 4.21 Transformer Construction [17]

In the Figure 4.21 the primary winding is represented by W1 colored in red, the secondary windings are: W2 colored in green which is corresponding to the 15 V, W3 - purple corresponding to 12 V, W4 – blue corresponding to 5V, W6 – light green corresponding to -12V and W7 which is common with W1. Between the winding special insulation yellow polyester tape (UL) was put to ensure a proper insulation [17].

The next step in design was to test the realized transformer to ensure that the results where corresponding to the calculations. The tests where done by providing voltage to the primary winding and then measuring the output voltage with an oscilloscope on each one of the windings to guarantee that the ration is according to the design (Figure 4.22, Figure 4.22,Figure 4.24).

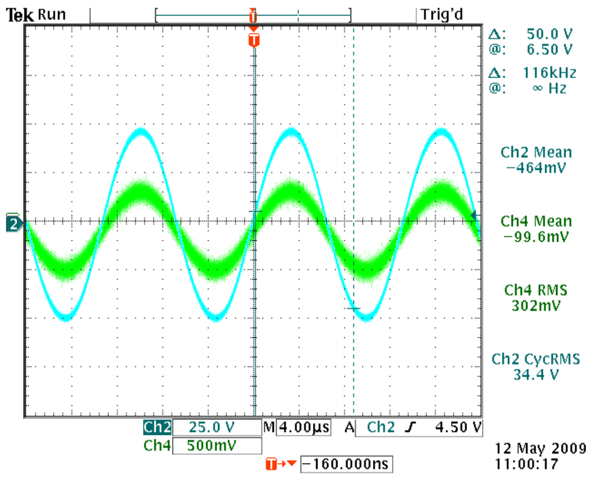


Figure 4.22 Measurements for 5V secondary winding

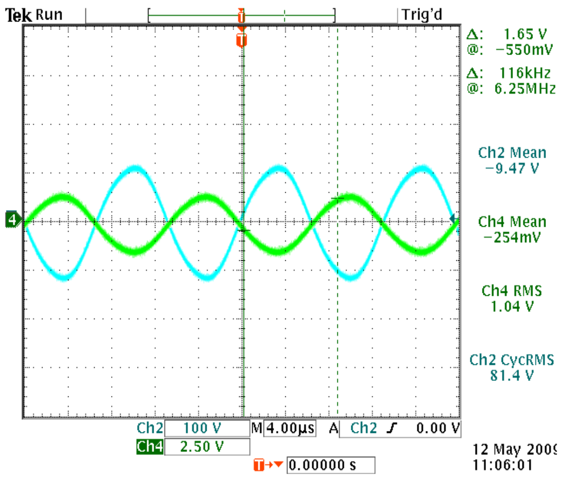


Figure 4.23 Measurements for 15V winding

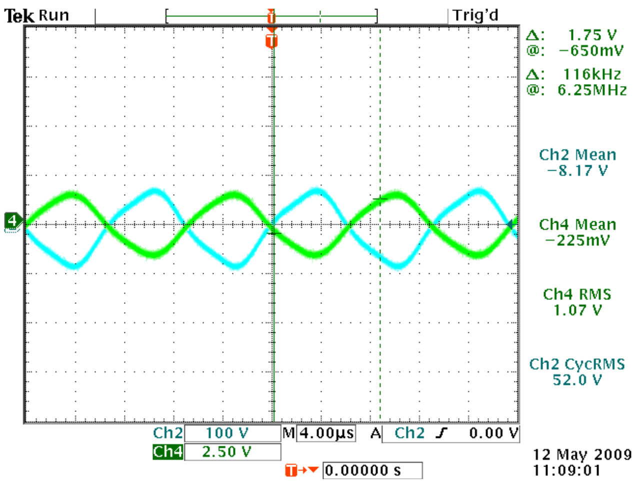


Figure 4.24 Measurements for 12V winding

The test that where carried proved that the calculations where made correctly and wanted ratios where achieved leading to the fact that the transformer is suitable for the supply.

Another important part in developing a power supply is to choose a suitable MOSFET for the required input voltage. Taking into account that the power supply will work in discontinuous mode and at a range 53.3:1 the MOSFET has to withstand breakdown voltage ratings of 1000V. The power switch it is placed in series with the primary of the transformer (Figure 4.18). The current in the primary winding of the transformer has the shape of a ramp which starts from zero to peak value given by Equation 4.1:

$$I_{peak} = \frac{V_{in} \cdot T_{on}}{L_{pri}}$$

Equation 4.1 [18]

In order to reach the steady state for a short period of time the output power of the converter must comply with the following relationship:

$$P_{out} \leq f_{sw} \cdot \frac{L_{pri} \cdot I_{pk}^2}{2}$$

Equation 4.2

Where:

- P_{out} output power
- L_{pri} primary inductance
- f_{sw} the switching frequency
- I_{pk} peak current

To obtain the wanted output voltage it is necessary to have a good control of the power switch, that control it is provided by a PWM Controller UC3845AN. The controller uses a feedback loop to determine if the output is correct. The internal error of the amplifier is not used in this case. The error is given by the voltage regulator TL431 which is connected to primary side trough a 6N137.The voltage across the optocoupler gives the operation frequency and the peak current that goes trough the power switch for each period or cycle. This procedure will give the necessary compensation for UC3845AN which sets the peak current. The diode D9 which is connected at the output of the optocoupler , the diode will raise the voltage to set the frequency of the oscillator. Based on the fact that the current on the optocoupler is limited by a resistor R11. The resistor value it is set by the output saturation of the optocoupler. The 6N137 has a current transfer ration of 100% which makes the current led aprox. 6 mA. A limit is added to the gain variations in order to give the right current. The value of the current it is given by a resistor R13 which is calculated with Equation 4.3 [18]:

$$R_{13} = \frac{\left(5V - (V_{TL431} + V_{LED})\right)}{8mA} = 120K\Omega$$

Equation 4.3

The resistor dedicated for voltage measurement is set by the current sense. For example one milliamp leads to one kilo ohm per volt. To improve voltage regulations the outputs for voltage measuring are split between al the outputs. The most important which is the 5V is connected directly to the to the UC3845AN trough the 6N137 which is voltage susceptible taking in consideration that the resistors connected to the 12V and 15V are less susceptible to voltage variations. The amount of current that is sensed can be determined by the resistors R15 and R16. The resistors are calculated by [18]:

$$R_{15} = \frac{\left(V_{5V} - V_{ref}\right)}{0.8 I_m} = 2.5\ K\Omega$$

Equation 4.4

$$R_{15} = \frac{(V_{12V} - V_{ref})}{0.2 I_m} = 32\ K\Omega$$

Equation 4.5

Where:

- V_{5V} 5V output
- V_{12V} 12V output
- V_{ref} reference voltage
- I_m current (1 mA)

As it was mention above the 5V output is the most important due to the fact that is the most sensitive so it is used as voltage feedback loop for compensation. The gain of the open loop is represented by Equation 4.6 [18]:

$$G_{DC} = \frac{(V_{in} - V_{out})^2 \cdot N_{sec}}{V_{in} V_e N_{pri}} = \frac{(800 - 5)^2 \cdot 4}{800 \cdot 1 \cdot 250} = 12.64\ dB$$

Equation 4.6

The value of the gain is maxim for open loop control. By reducing the line input to the minimum, the bandwidth of the close loop will also decrease but this time for closed loop control. The maximum bandwidth that can be achieved is [18]:

$$f_{band} = \frac{f_{sw}}{5} = 15\ KHz$$

Equation 4.7

An important consideration is made when the supply is working at minimum input voltage, at this point the duty cycle must be at maximum value which is 80%. In this case the voltage across the regulator will be at the maximum level of 8.5V [18].

Another matter is the deadtime of the power switch which is set to be minim. The timer it is set by the capacitors and resistors. The capacitance is considered from controller datasheet. The timing resistance R12 placed at the input RT/CT of the controller is determined by multiplying the voltage corresponding to 100% duty cycle divided by auxiliary voltage (5V) will give the resulting resistance (Equation 4.8) [18]:

$$R_T = \frac{V_{VLOon}(16K)}{V_{aux}} = 27\ K\Omega$$

Equation 4.8

An important issue is the working frequency, because the minimum frequency at which the power supply can work is when the error of the controller is at the minimum output voltage (V_{outmin}=0.85V). The frequency is calculated based on Equation 4.9 [18] :

$$f_{low} = \frac{(V_{VLOon(min)} + V_{input(ctrl)})}{V_{VLOon}} \cdot 140KHz = 75\ KHz$$

Equation 4.9

Where:

- $V_{VLOon(min)}$ minimum on voltage (0.85V)
- f_{low} lowest working frequency
- $V_{input(ctrl)}$ controller input voltage
- 140 KHz is which is the operating frequency of the oscillator

In order to function proper the control needs also a current sense resistor (R10) which is connected to the source of the power MOSFET. To determine this resistance in conditions of minimum input voltage it is necessary to use peak current determined in the design calculation. To have linear operation mode the sense voltage must be limited at 1V. The equation describing the resistance [18] :

$$R_{sc} = \frac{V_{sc}}{I_{pk}} = 1.2\ \Omega\ at\ P = 1W$$

Equation 4.10

Where:

- V_{sc} sense voltage
- I_{pk} peak current
- R_{sc} sense resistor

To avoid instabilities for example due to high range and short time input voltages and also current ripples, it is important to have a current slope delay circuit. This kind of circuit will also provide a delay function caused by the current sense resistor (R9) and capacitor (C7). The amount of delay introduced is 0.7µs. The value of the capacitance ca varies from 0.4µF to 1µF, which in this case is selected to be maximum. In exchange the value of the resistor can be calculated based on the Equation 4.11 [18]:

$$R_9 = \frac{T_d}{C_d} = 700\ \Omega$$

Equation 4.11

It is know that the whole controller will draw energy from the system. The start-up current for the controller is <0.7mA. The necessary energy during start-up is stored in a capacitor C6 which is connected at the input pin of the controller. It is also important that the breakdown voltage to be limited by a set of resistances connected in series. The total resistance for start-up is given by Equation 4.12 [18] :

$$R_{stratup} = \frac{V_{in(\min)}}{I_{startup}} = 410\ K\Omega$$

Equation 4.12

The resulting resistance is divided in 5 resistors connected in series R1, R3, R4, R5, R8. The total power dissipated on them is given by Equation 4.13:

$$P_{Rstratup} = \frac{V_{in(\max)}^2}{R_{startup}} = \frac{800^2}{410000} = 1.56\ W$$

Equation 4.13

The conclusion is that each one of the resistors will dissipate 0.31 W. Taking in consideration that the total power is 84.4 W , the resistors are dissipating 1.84% from the total amount of power.

An important part in designing the SMPS is the design of the PCB. The design was made using the OrCAD software with regard to insulation problems because in the primary part of the supply is located the high voltage part. The circuit it is presented in Figure 4.25:

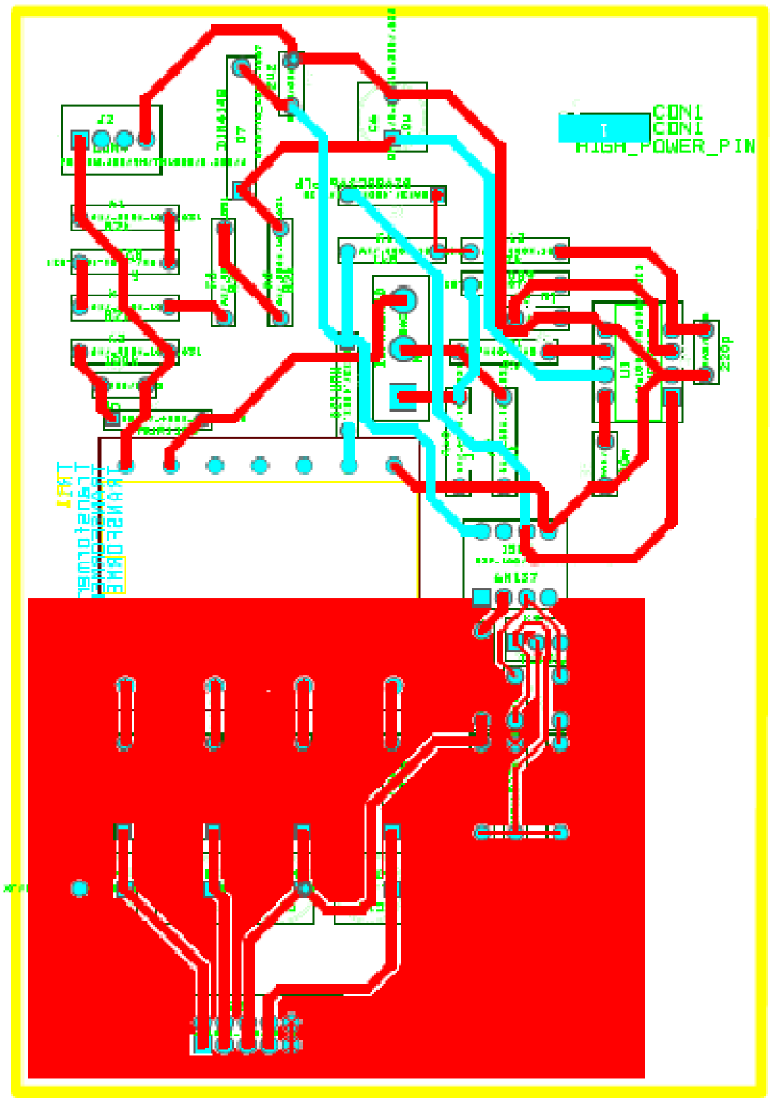


Figure 4.25 PCB Layout of SMPS

As the figure shows the Layout is composed from two layers of copper. The red layer represents the bottom part of the PCB and the green layer represents top of the PCB. The trace of the PCB where made considering all the facts like RFI radiation, component reliability, efficiency and stability. Like any other traces these ones also have resistance and inductance. These factors can lead to high voltage transitions as consequence of large variations of the current that flows through the traces. For example the trace from an amplifier that are near to the power signals can be influenced so the amplifier will get very unstable. In the design it is good to consider having traces thick and short in order to minimize the inductive effect and the resistive. Additional attention was paid to the layout that is around the capacitive filter. For example if the capacitors where placed in parallel within a straight line and placed nearby the source will get hot due to the ripple current. There are many aspects to consider when designing a PCB, most of them very important in well functionality of the supply [19].

4.5.3 OVERALL RESULTS AND CONCLUSIONS

Overall results are deducted after conducting experimental test on the supply. So problems have been encountered due to the PCB imperfections occurred in the manufacturing process. The entire have been corrected until satisfying results where accomplish.

In order to perform the laboratory tests some minimum conditions must be fulfilled:

- The minimum voltage necessary for the supply to work is 60 V, for safety reasons the tests for maximum voltage was not done.
- The loads where chosen so the nominal currents have been reached.

The test setup is displayed in the Figure 4.26 :

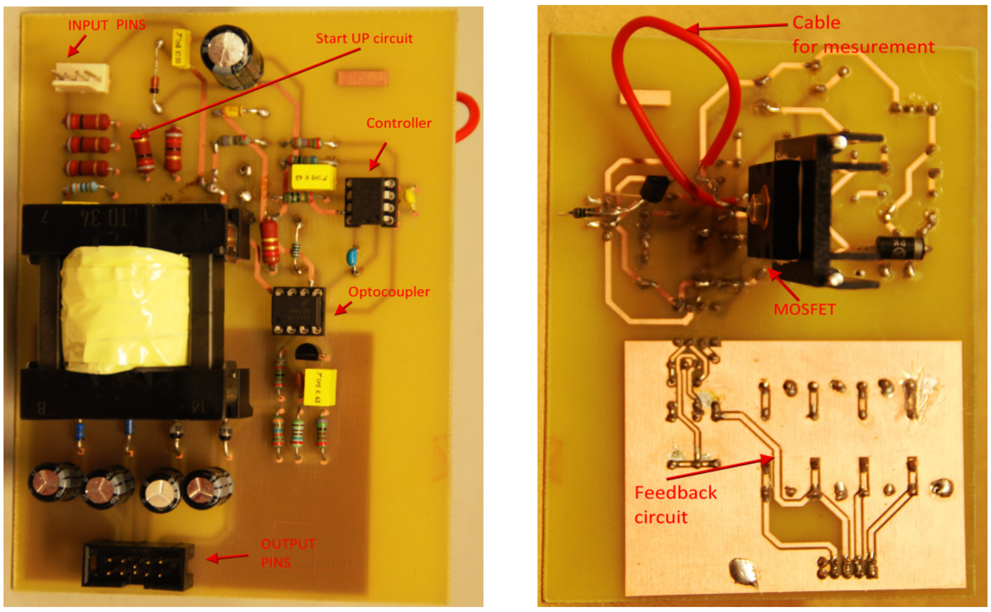


Figure 4.26 Top and bottom PCB of SMPS

The basic components of the SMPS are displayed in the Figure 4.26. A cable was attached to the drain of the MOSFET. The purpose is to measure the current flow trough the transistor.

Several tests where done in order to determine the each output of the supply. The first test was done for the 15 V output. The results where plotted using an oscilloscope.

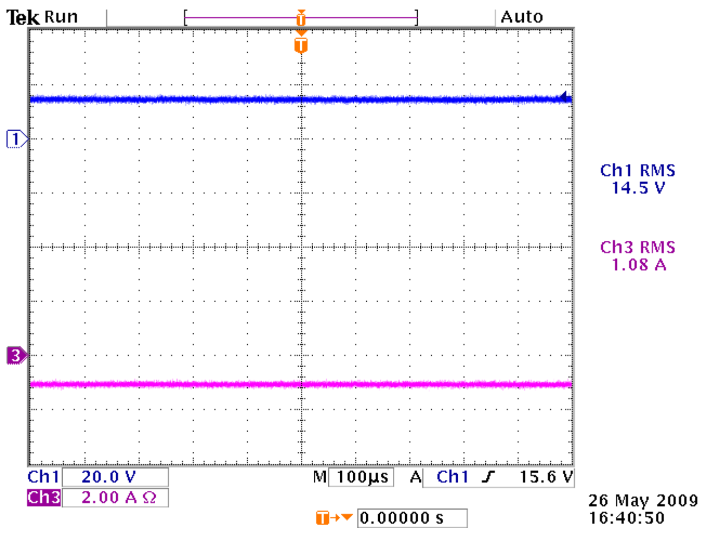


Figure 4.27 Output 1 – CH1 Voltage output, – CH3 current output

In Figure 4.27 it is shown the output current coloured in magenta and output voltage coloured in blue. The amplitude of the voltage is according to the wanted output. To test also the how it behaves also a load was connected. The load is a resistor of 12 Ω.

The conclusion is that for this output the target for 15V with maximum current of 2A was reached.

The second test was for the 12 V output. This is shown in Figure 4.28 coloured in the same manner like the one from above.

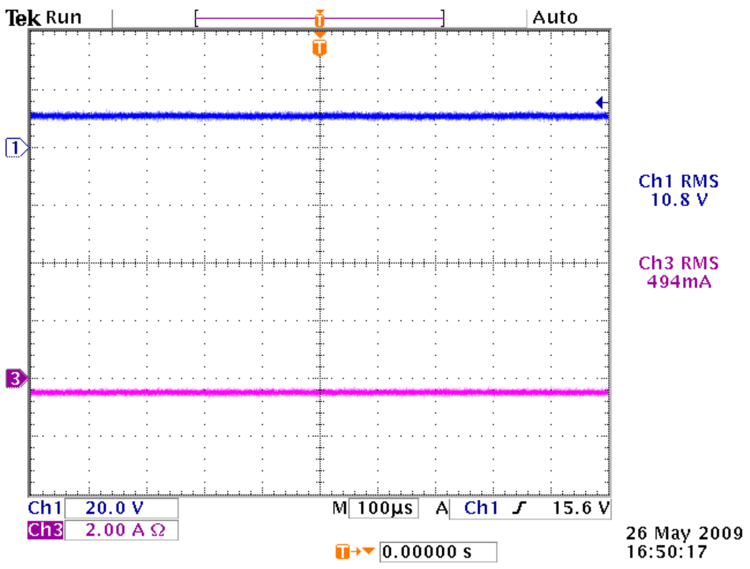


Figure 4.28 Output 2 – CH1 Output voltage, - CH3 current output

The conclusion is that also this output behaves well taking in consideration the target of 12 V at a maximum current of 1A.

The most important test is the one for 5V because this voltage is used as a feedback for the control loop. So it is important because the other outputs depend of this output.

The Figure 4.29 is also plotted by using the same procedure like the others and also the output was loaded with a 4 Ω resistor.

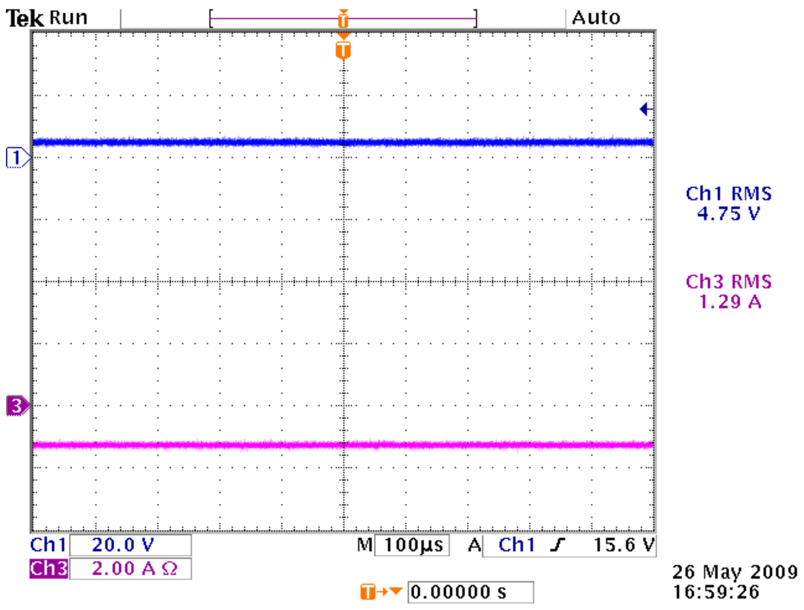


Figure 4.29 Output 3 – CH1 Output voltage, - CH3 current output

The entire test showed that the power supply it is behaving according to the design. Due to the fact that a 3.5 Ω resistor was available the test was performed with that resistor. The power supplies made in now days use closed control loop so it is very important to have a good feed back loop and also a good galvanic separation between them provide by optocouplers.

Flyback topologies are widely used because of their simple, robust design. Of course like any other system it has also drawbacks. One of the major advantages is that it does not need an inductive filter like other topologies. The fact that transistors today can withstand more and more voltage makes the topology fitted also for high voltage applications like in this case. The range is very wide because of the transistors variety. The most frequently known supplies are form 110VAC and 220VAC to DC wide range.

5 SOFTWARE IMPLEMENTATION

The software chapter contains the software description that was used in order to control the system. The software represents the actual implementation that has been done on the inverter. The data structures, memory map as well as the implementation of mathematical operations on the FPGA by using a look-up table based on a flash memory is described. The software is also described in direct relation to hardware due to the logic connectivity especially between FPGAs and the rest of the system. The software described herein relates to FPGAs, DSP and microcontrollers used in the application.

5.1 INTER FPGA COMMUNICATION

The main FPGA is connected with the secondary FPGA through an extended serial interface. A normal serial interface uses 3 or 4 signals in order to do a bidirectional communication and signaling. The Interface used here is based on both serial and parallel interface. The basic 4 wires serial communication, plus the optional signals, is presented in Table 5-1.

No	Signal name	Description
1	Clock	The main clock signal
2	Chip select	The chip select flag
3	MISO	Master In, Slave Out
4	MOSI	Master Out, Slave In
5	INT	Interrupt (Optional)
6	Ready	The ready flag (ex: when high the device can be used) (Optional)

Table 5-1: Basic serial communication interface

The interface used for this communication is based on multiplying the MISO and MOSI signals and using the interrupt line in order to signal the master that data exist in the buffer. Therefore a communication line based on 4 MISO and 4 MOSI have been created. The communication is made over differential lines in order to be able to increase the frequency. The base clock frequency is 133 MHz which is not the actual communication frequency. Depending on the cable the frequency will be adjusted.

The communication is made over LVDS unbalance lines. The unbalanced lines were used because they are simpler in the sense of connectivity. The difference lines require resistors to be placed at the end of the cabled between the positive and negative lines.

The role of this communication is the data exchange between the two FPGAs. The secondary FPGA interfaces with a keyboard and therefore allows the user to input values. The user input values are send to the main FPGA or handled locally depending on their role and purpose.

This interface also allows the error to be sent from one FPGA to the other, errors like faults, control errors, parameter errors or other kind of data. The error sent by the main FPGA refers mainly to control errors or measured errors. The control errors represent invalid states of the input parameters.

The secondary FPGA also interfaces with the microcontrollers which control the relays and therefore the power supply of the entire system. The main FPGA requires the status of these in order to better control the inverter and to determine its status.

The protocol and the data bus are exemplified in Appendix F Section 2. These are required in order to know how they interface one-another and how the data is interface between them.

The pin-outs of the communication are shown in Appendix H Section 1 and are used by the software and by the FPGAs main pin types.

5.2 FPGA CONNECTION WITH DSP

The main FPGA connects with the DSP through a parallel interface. The parallel interface is used to connect the DSP with a memory device or with an external peripheral. Therefore, the FPGA emulates a memory device in order to allow DSP to connect and exchange data with the FPGA. The data exchange, from the DSP point of view is as simple as writing to a memory like an ASRAM. The data is therefore read and write to a preset memory location. The data bus line of 16 bits plus the address bus of 20 bits allows a memory space of 1Mbit. The amount of data is more than enough for the application. The total accessible memory space is 262 Kbit because only the first 14 bits of the address bus are used.

The connection is made as depicted by Figure 5-1. In the figure there are some extra communications like reset and fault. These signals even if they are spread system wide, the FPGA acts like a router in order to simplify the connection.

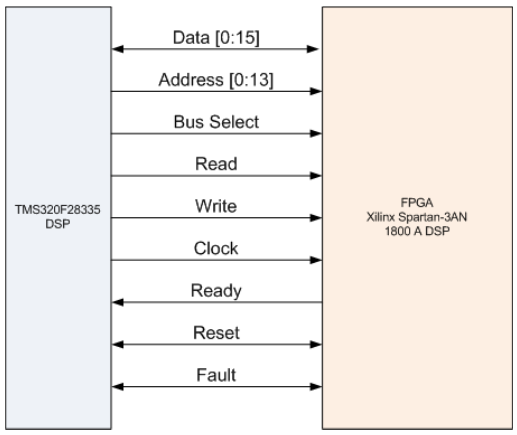


Figure 5-1: DSP – FPGA connection

5.2.1 LOGIC CONNECTION

The logic interface, as depicted by Figure 5-2 showed the communication bus between the two logic devices. The DSP uses the external memory interface in order to communicate with its peripheral device. The FPGA software contains a set of parallel port control structures as well as a data bus and address bus.

The control structure helps in synchronization with the DSP as the FPGA is a slave for the DSP. The parallel control assures that proper response will be issued by the FPGA according to the DSP request. Signals like Read, Write, Clock and chip select are the main signals that the DSP provides. These signals establish the main protocol on which the entire communication is made.

The data bus is 16 bits wide and it is fed into a set of data latches in order for the parallel control to manage them. The address bus is only 14 bits wide (limited by the design) and it is directly connected with the parallel control. In order to simplify the design inside the FPGA and to allow full control over the data communication at full speed, the entire parallel subsystem interfaces with a dual port RAM. The dual port RAM allows two systems to operate at different clocks and still synchronize. Also, the main advantage is that the two parts of a dual port RAM memory can operate independently

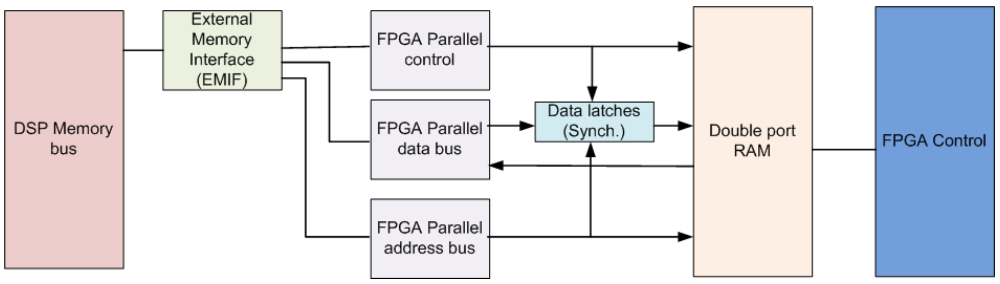


Figure 5-2: FPGA with DSP communication

The FPGA ports used for the data bus are configured as Input/Output ports and have been programmed as direct ports without buffering in latches.

The entire communication and the parameters of the control is located in Appendix F Section 8.

5.3 GATES CPLD

The gates CPLD software had been updated for this project in order to provide a better synchronization between the units. The current version supports selection of fault signals that will form the main fault signal. As the Gates CPLD role is also to connect the faults from all the units it connects to, it also generates a single fault signal from all the units. Because not all the units post two fault signals, or some ports are left unconnected as backup or are reserved, the fault signal from these port must be ignored as well as pulses shall not be send to them. Due to this, the units can be validated or invalidated (turned OFF).

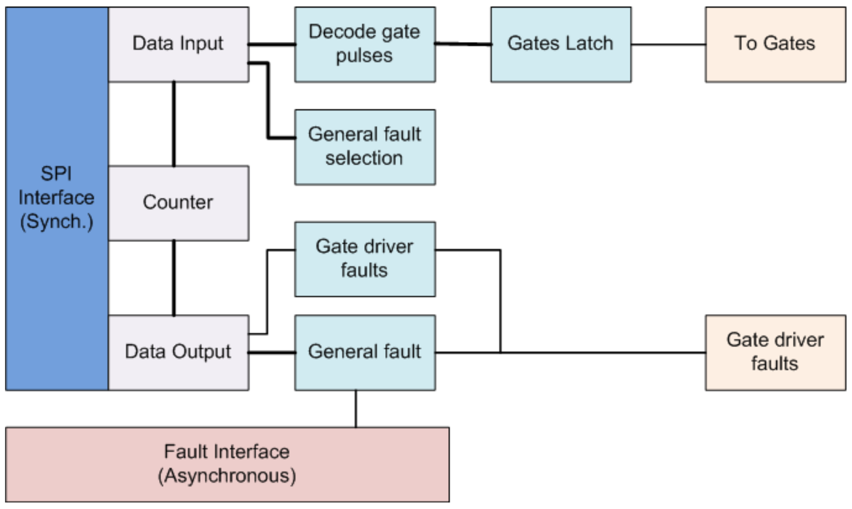


Figure 5-3: Gates CPLD Software blocks

The gate CPLD uses a standard SPI interface as described in the previous report (See [13]). Over this SPI interface, the data output and data input structures have been modified in order to allow the new enable signals. Therefore the bits used by the interface are presented in Table 5-2 while the pulses order and alignment are presented in Figure 5-4.

Bit	Data Input 1	Data Input 2	Data output
0	Enable 0	Enable 1	The fault flag of the entire connection
1	Unit 0 State	Unit 1 State	The state of the internal Fault
2	Unit 0 Leg 0 Level	Unit 0 Leg 1 Level	Unit 0 - Fault
3	Unit 1 Leg 0 Level	Unit 1 Leg 1 Level	Unit 0 - Over temperature
4	Unit 2 State	Unit 3 State	Unit 1 - Fault
5	Unit 2 Leg 0 Level	Unit 2 Leg 1 Level	Unit 1 - Over temperature
6	Unit 3 Leg 0 Level	Unit 3 Leg 1 Level	Unit 2 - Fault
7	Unit 4 State	Unit 5 State	Unit 2 - Over temperature
8	Unit 4 Leg 0 Level	Unit 4 Leg 1 Level	Unit 3 - Fault
9	Unit 5 Leg 0 Level	Unit 5 Leg 1 Level	Unit 3 - Over temperature
10	Enable Fault 0	Enable Over Temp 0	Unit 4 - Fault
11	Enable Fault 1	Enable Over Temp 1	Unit 4 - Over temperature
12	Enable Fault 2	Enable Over Temp 2	Unit 5 - Fault
13	Enable Fault 3	Enable Over Temp 3	Unit 5 - Over temperature
14	Enable Fault 4	Enable Over Temp 4	Unit 6 - Fault
15	Enable Fault 5	Enable Over Temp 5	Unit 6 - Over temperature
16	Enable Fault 6	Enable Over Temp 6	Unit 7 - Fault
17	Enable Fault 7	Enable Over Temp 7	Unit 7 - Over temperature
18	Enable Fault 8	Enable Over Temp 8	Unit 8 - Fault
19	Enable Fault 9	Enable Over Temp 9	Unit 8 - Over temperature
20	Enable Fault 10	Enable Over Temp 10	Unit 9 - Fault

21	Enable Fault 11	Enable Over Temp 11	Unit 9 - Over temperature
22	Enable Fault 12	Enable Over Temp 12	Unit 10 - Fault
23	Reserved	Reserved	Unit 10 - Over temperature
24	Reserved	Reserved	Unit 11 - Fault
25	Reserved	Reserved	Unit 11 - Over temperature
26	Reserved	Reserved	Unit 12 - Fault
27	Reserved	Reserved	Unit 12 - Over temperature
28-31	Reserved	Reserved	Reserved

Table 5-2: Bit description for the Gate CPLD

The data input bits on position 0 will set the enable flag of the entire communication system. If the bits are low, the states and levels of the units will be ignored, and all the output signals for the gate drivers will be cleared. In this situation, the system is turned off immediately after setting the bit low (see Figure 5-4). If no enable flags should be set or faults signal to be read, the communication may stop right after the first bit.

The unit states defines the actual way the unit is working. If 0, the unit will be off condition in which the IGBTs are shutdown. If the state is on (logic high), the levels for the corresponding unit will be on therefore for that particular unit 2 IGBT will be switched on, one on each leg. The actual states of the IGBTs will be set only at step 11 unless the enable flags are low (see Figure 5-5).

The enable fault signals are used to configure each fault signal received from the IGBT drivers as a source for the global fault signal. Each bit of the enable fault data set will be bit “AND” with the corresponding fault signal of the unit.

The fault flag of the entire connection (bit 0) is high if any of the input fault signals is high. This is not affected by the enable fault flags.

The internal fault flag (bit 1) represents the actual state of the external fault signal. This flag is affected by the enable fault flags.

The unit faults and over temperature faults are identical with those presented in the previous report (see [13]).

The bits are aligned according with Figure 5-4. The MOSI lines are set on the falling edge of the clock therefore a read can be made on the rising edge. The MISO lines are shifted 180 degrees therefore they will be set on the rising edge of the clock and read by the master on the falling edge.

The clock frequency can be as high as 40 MHz, limit imposed by the CPLD internal arrangement of logic. At a clock rate of 33 MHz, the entire system can be updated at a total speed of 1 MHz due to data bus size of 33 bits. The data bus can be reduced to at least the first 10 bits. After the first 10 bits of data, the communication may stop (CS goes low) and the remaining bits will not be set. If this is desired, the CS line should go low on the rising edge of the clock. The communication can be reduced to 1 bit if it is desired to shutdown the system as described earlier in this chapter (see description of bit 0 of the communication protocol).

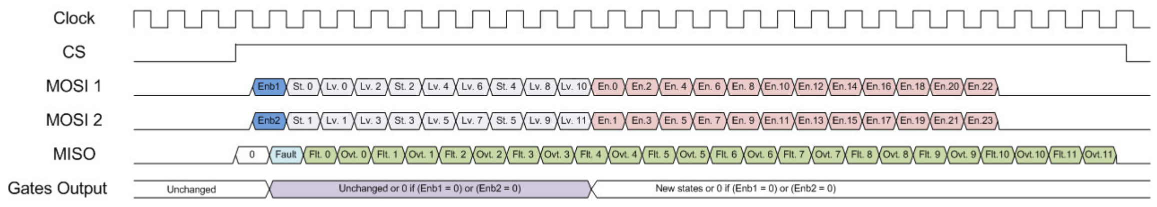

Clock
CS
MOSI 1
MOSI 2
MISO
Gates Output
Enb1
Enb2
Fault
Unchanged
Unchanged or 0 if (Enb1 = 0) or (Enb2 = 0)
New states or 0 if (Enb1 = 0) or (Enb2 = 0)

Figure 5-4: Gates CPLD communication data order

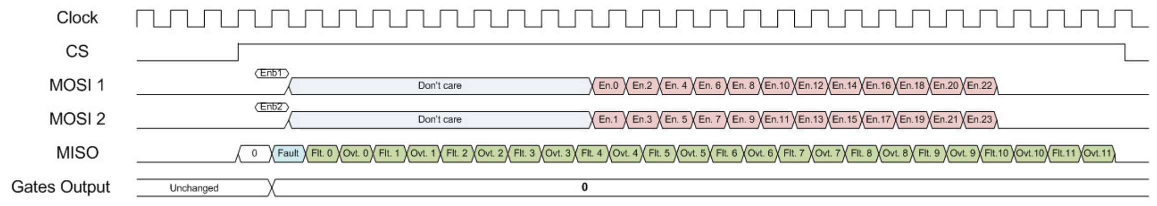


Figure 5-5: Gates CPLD communication data example (enable OFF)

5.4 MAIN FPGA IMPLEMENTATION

The main FPGA contains the software related with modulation techniques, actual control of the IGBTs as well as fault management and system monitor. The software contained by this FPGA is the most relevant as it is used to control all other devices used in the system. It can control the contactors though paths like: Main FPGA -> Secondary FPGA -> Microcontroller -> Auxiliary relays -> Contactors. This path is possible due to serial communication between the two FPGAs and also through a serial communication between the secondary FPGA and the microcontroller after which all the data is send in parallel to the relays. The software simulation for the FPGA done in ISE (Programming software from Xilinx) is found in Appendix G.

The main FPGA contains several state machines that help in stepping throughout the processes in order to accomplish the desired result. The FPGA input is given by several devices like:

- The secondary FPGA: gives the go ahead after the power supply has stabilized as well as the fault signal is something is wrong with the measured values
- The DSP gives the main control reference for the inverter and the main measured values of the currents and voltages
- The ADC provides the unit voltages in order to monitor them
- The gates CPLD provides the fault signals of the IGBT gate drivers

5.4.1 MAIN CONTROL

The main control of the FPGA is to handle the inverter modulations. The modulations are therefore handled by a state machine inside the FPGA which runs at 125 MHz (the top speed for the control structure inside the FPGA is 145 MHz – value obtained from the Xilinx compiler). All the measurements, control, unit selection and mapping are done in 5 steps which allow a control refresh each 40 ns. The actual speed at which the gates drivers are refreshed is also depended on the gate CPLD communication and the CPLD possibility. Therefore the actual refresh rate for the IGBTs is given by the sum of the two values which is 32 ns for the gates CPLD and the total is 72 ns. The response speed is totally dependent on the ADC possibility which is set at 2 MSPS for a channel therefore all channels are acquired in 3 µs. The value of 3 µs represents the actual response time of the main control. During this period, the gates CPLD can be updated 40 times with values obtained from the partial reads from the system. The fault response is limited to 40 ns.

As stated earlier the control is divided on 5 stages each represented in Figure 5-6. Each stage is triggered by the previous step and all reference to the rising edge of the main clock. Most of the stages take 1 cycle to complete except with the ADC which is based on several cycles. This is compensated to a single cycle by using latches and registers in order to allow the control to better process the data and handle the faults.

5.4.1.1 STAGE 0 – CLOCK AND ADC

The Clock block inputs the main clock as well as the maximum value for the counter which is set by the DSP. The output of this block, the reference counter is based for carrier generation.

The ADC reads the data from the actual ADCs and receives the measurement done by the DSP. The ADC requires 30 clocks at 30 MHz to complete all the acquisition but the data output is maintained between period in order to allow the control structure to handle the data. The data output is scaled to Q8 for the acquired values while the data received from the DSP can have any Q value. For testing the Q8 was used also for the DSP.

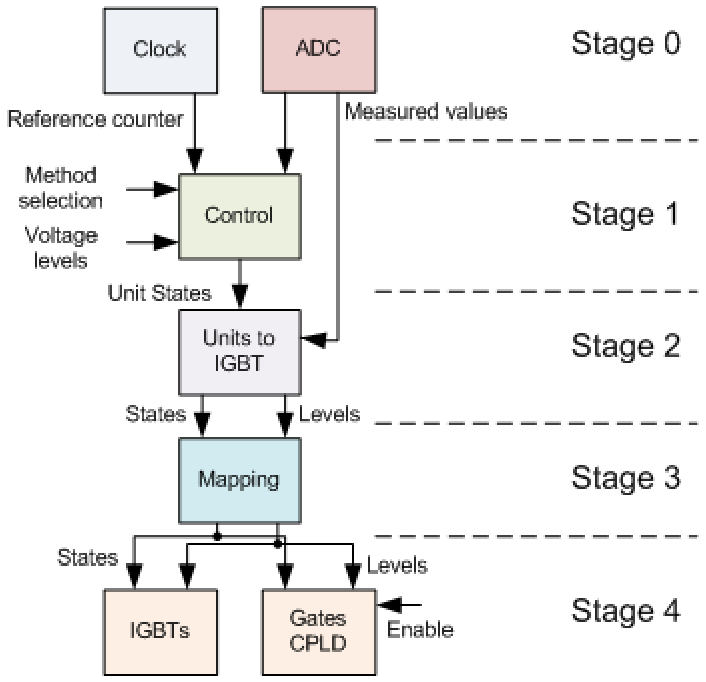


Figure 5-7: FPGA Control stages

5.4.1.2 STAGE 1 – THE CONTROL BLOCK

The control block contains the modulation strategies which are selected by the DSP when the enable flag in software is turned OFF. The voltage levels are used only by the staircase modulation and are defined for all three phases as a single set of values. These values represent the trigger state for the next level. By default, the voltage levels are set to half of the step size. The values are given in Q17 with a range of [0, 2] which represent the actual voltage output in per unit in the range of [-1, 1]. In order to simplify the way values are compared inside the FPGA the offset of 1 was considered.

The method selection is as follows:

Method ID	**Type**	**Sub Type**
000	Staircase modulation	-
001	Phase shifted modulation	-
010	Level shifted modulation	IPD
011	Level shifted modulation	APOD
100	Level shifted modulation	POD

Tabel 5-1: Method selection bits

The method selected the corresponding modulation block inside the control structure. As stated earlier the method cannot be changed with the enable flag on in order to prevent undefined conditions.

The output of this block is selected with respect to the current method from the three blocks. If a block is not used, the enable flag is set to 0. The block is only used to connect all the modulation blocks together.

5.4.1.3 STAGE 2 – UNITS TO IGBT CONVERSION BLOCK

The Units to IGBT block converts the states for all three phases received from the control structure. The conversion is made with respect the quadrant in which the corresponding phase is located. The input value represents the state of the units like capacitive and conductive. The states are converted to a unit state of ON or OFF and to a leg level which represent the actual IGBT. A level of 0 represents the lower IGBT is turned on while a state of 1 represent the upper IGBT to be turned ON. The conversion only handles the required number of cells of 24 divided into 8 per phase.

5.4.1.4 STAGE 3 – MAPPING

The mapping routes the signals received from the conversion block to the actual location of the gate driver connections. This is required because the system is equipped with 30 outputs from which only 24 are used. The rest of 6 links are reserved and can be used if a port fails. The mapping was done by test the port functionality and by cable number.

5.4.1.5 STAGE 4 – COMMUNICATION

The communication block posts the data from the entire modulation block to the gates CPLD. Because the gates CPLD communication is done to a different speed than the modulation block, latches are used to proper set the output values.

5.4.2 STAIRCASE MODULATION

The staircase modulation was implemented on the FPGA with the help of state machines. The state machine defines 5 stages from which the first and the last are idle stages. The stages 1, 2 and 3 are active stages during the following tasks are done:

Stage	Name	Operations
0	Begin stage	- resets the used values to their default state
1	Sort stage	- sorts the units by voltage level for the upper and lower sections of each phase
2	Voltage level calculation	- Determine the actual number of cells that will be set for the upper and lower sections of each phase
3	Determine active units	- Determine based on the voltage level calculation and sorting which cells will be turned ON or OFF
4	Idle stage	- Signals the end of the modulation and prepares the jump to the begin stage for the next clock cycle

Tabel 5-2: Staircase modulation stages

5.4.2.1 SORTING

The sorting only handles 4 units, number which represents the cell count for the upper and lower sections of one leg. The sorting uses comparators and additions in order to sort the values in descending order. The output vector represents the index of the sorted units. For example, the index position 0 represents the location of cell 0 with respect to the other units. A vector with the values [0, 1, 2, 3] tells us that the units are already sorted while a vector like [3, 1, 0, 2] tells us that the unit 0 which has the value 3 is the last unit in the sorting vector while unit 2 (with value 0) has the lowest input value compared with the other 3. For a better understanding on how the sorting works a print screen from the unit simulation is presented in Figure 5-8. The clock from the figure is the simulation clock and not the actual clock rate. The voltage0, voltage1, voltage2 and voltage3 represent the input variables while the sorted0, sorted1, sorted2 and sorted 3 represent the output of the block. The done flag is set at the same time with the enable flag which signals that a single clock cycle is required for the sorting.

The sorting algorithm is not based on iterations as the FPGA can process parallel data by its design therefore the entire sort can be done in a single clock cycle. The sorting block implementation on the DSP requires a maximum clock frequency of 143 MHz, this being the limitation imposed to the rest of the blocks in the modulation.

For further details and examples see Appendix F Section 1.

5.4.2.2 DETERMINE VOLTAGE LEVEL

The voltage level is determined based on the reference voltage and the preset voltage levels[1]. Depending on the comparison, the corresponding number of units will be on for the upper and lower sections of an inverter leg. A representation of how the number of units turned ON is calculated is shown in Figure 5-9. The M represents the number of units turned On for the lower part while N represents the number of units turned ON for the upper part. When all the units in the upper part are ON, the output voltage is the +VDC.

[1] The voltage level is set by the DSP prior to any operation or the default values will be used

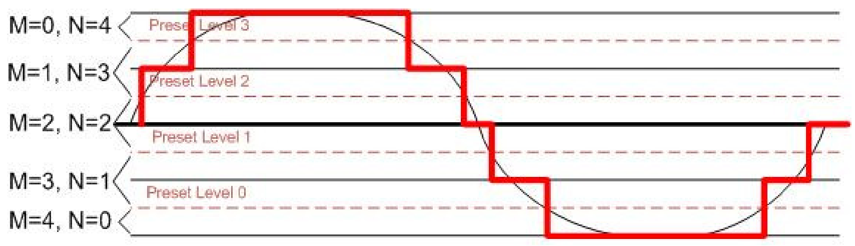


Figure 5-10: Voltage levels

5.4.2.3 SELECT UNITS STATES

The unit states are selected based on the sorted values and on the voltage level. The level number is taken from the voltage level determination block and compared with the sorted values. Both values are in the range of [0, 3]. If the sorted index value for one unit is less than then the voltage level than the unit is selected, if not it is turned OFF.

For example if the voltage level is 3 and the sorted values are {3, 1, 2, 0} then the units 2, 3 and 4 are turned ON while unit 0 is OFF.

5.4.3 PHASE SHIFTED MODULATION

The phase shifted modulation also implements a state machine in order to cope with the operation order as well as with the timings.

The operations required for this modulation are:

- Generate the carriers
- Compare carrier with reference
- Determine unit states

The carriers are generated based on the system clock. The counter on which the system is based on is incremented up to a preset value by the DSP then is decremented. This process is repeated all over again until the modulator is stopped. As an example of how the carrier is generated and how the comparison is made is shown in Figure 5-11 for a single unit, phase and carrier.

For this method a total of 8 carriers are required. Because each 2 carrier are in mirror, only 4 carriers needs to be implemented while the other 4 represent the difference between the maximum counter and the main carrier.

At initialization one carrier is the minimum value, one takes the maximum value while the other two get the half of the counter maximum value.

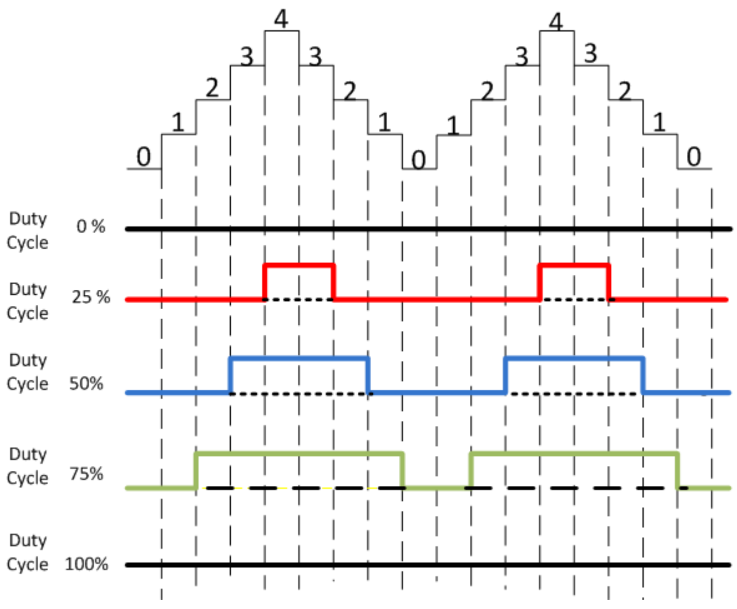


Figure 5-11: Counter and comparison made for output pulses

The unit states of each unit are calculated with respect to their reference and with all 4 counters. If the reference is less than the carrier the unit will turn ON. The state is maintained until the reference is bigger than the carrier when the unit will be turned OFF. This applies for all the units and phases.

5.4.4 LEVEL SHIFTED MODULATION

The levels shifted modulation is different from the phase shifted modulation when it comes to implementation. The difference consists in the fact that units must be cycles at each period of the fundamental plus the level of the maximum value of the counter is divided among all the 4 required carriers. The carrier calculation is based on a single counter which is has its maximum value 4 times lower than the maximum value. To the value of this counter, the corresponding offset is added in order to obtain the proper carrier. The carriers are also calculated with respect to the sub method.

The comparison is made in the same as for the phase shifted modulation. The difference for unit selection consists in the way how the units to be switched are selected due to the cyclic rotation of the units. The comparison needs only to be made with respect to the lower or upper units, the corresponding unit from the other leg section is triggered in the opposite way.

5.4.5 MODULATION IMPLEMENTATION

The modulation was implemented on the FPGA into a single unit with the representation from Figure 5-12.

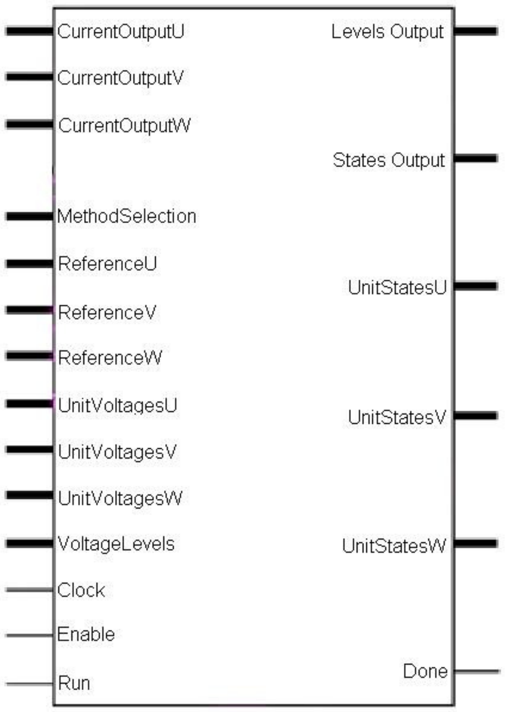


Figure 5-12: Modulation block

The output of the block is common for all the modulation as it outputs the states for the units and the levels (first two outputs). The UnitStatesU, UnitStatesV and UnitStatesW outputs the states only for the units on the corresponding phases. The Done output triggers the next step to take place (eg. Send the data data to the IGBTs).

The first three inputs are used only by the staircase modulation as well as the UnitVoltages U, V, W and Voltage Levels. The real input is the ReferenceU, ReferenceV and ReferenceW which represent the main reference from the control structure. The method selection input selects the modulation methods which are:

- 0 – Staircase
- 1 – Phase shifted
- 2 – Level shifted – IPD
- 3 – Level shifted – APOD
- 4 – Level shifted – IPOD

The Clock represent the main clock based on which the modulation is calculated. The Enable triggers the modulation to run. If false, all the values inside the modulator will be cleared including the counters. The Run is high only when the modulation has to run. This is used in order to integrate the modulation into a state machine.

5.5 DSP SOFTWARE IMPLEMENTATION

The DSP software represents the main control of the entire system. This implements a simple U/V motor control in order to show the system functionality. Also code had been written to allow a simple RL load to be tested.

The DSP software was implemented as any other control software without any consideration to the inverter topology. The DSP task is only to provide the reference voltages for the modulation. The PWM module inside the DSP is not used, being replaced with the FPGA. The duty cycles are calculated and then sent to the FPGA over the external interface. For the DSP the FPGA is just a normal external memory mapped at the address 0x10000 (Zone 7). Therefore, a structure can be defined and a variable of the structure allocated at the beginning of Zone 7. After this step, all the communication between the DSP and FPGA is done transparently as easy as writing to a normal variable.

The software that was implemented only shows that the DSP can be used to control this inverter without any supplementary programming required. For this a single unit had been written which configures the zone 7 and the I/O pins of the DSP. The data type structure has also been defined and can be accessed throughout the software. Also the helper functions for numeric conversion have been included in the unit file. The pin-out, memory map and number representation have been already described at the beginning of this chapter.

The DSP board made by Spectrum Digital with code eZDSP28335 has been used. The board also includes an ASRAM on zone 6 which uses the same data and address pins as the FPGA.

5.6 SECONDARY FPGA SOFTWARE IMPLEMENTATION

The secondary FPGA is used to communicate with the comparators CPLD, main FPGA, and the both relay microcontrollers. The communication is serial on differential lines with the main FPGA and on single ended lines with the other components.

The communication with the microcontrollers on the relay board is done according with the microcontroller documentation for the hardware interface and with the protocol described in the relay output microcontroller subchapter.

The secondary FPGA tasks are simple and will not be described in detail. The tasks are:

- Control the relays
- Interface with a keyboard
- Handle the comparators CPLD communication
- Handle the communication with the main FPGA

5.7 RELAY OUTPUT MICROCONTROLLER

The relay output microcontroller handles the communication between the secondary FPGA and the relays. Between the relays and the microcontroller a set of 6 boards with optocouplers and open collector ICs help in controlling the relays with 24 VDC.

The protocol as well as the commands is shown in Appendix F Section 6.

5.8 RELAY INPUT MICROCONTROLLER

The relay input microcontroller is used to read the contactors states. The states are then sent to the secondary FPGA for validation. The data is send over an SPI connection on 4 wires. The relay outputs 5 bytes with the format identical with that from the relay output microcontroller including the CRC calculation.

The protocol and the commands for the relay input microcontroller are found in Appendix F Section 7.

6 EXPERIMENTAL WORK

The experimental work that was conducted included several data acquisitions from parts of the system. Most of the tests were done mostly with respect to the new units.

6.1 THE SETUP

The setup is shown in Figure 6-1 with the main component highlighted.

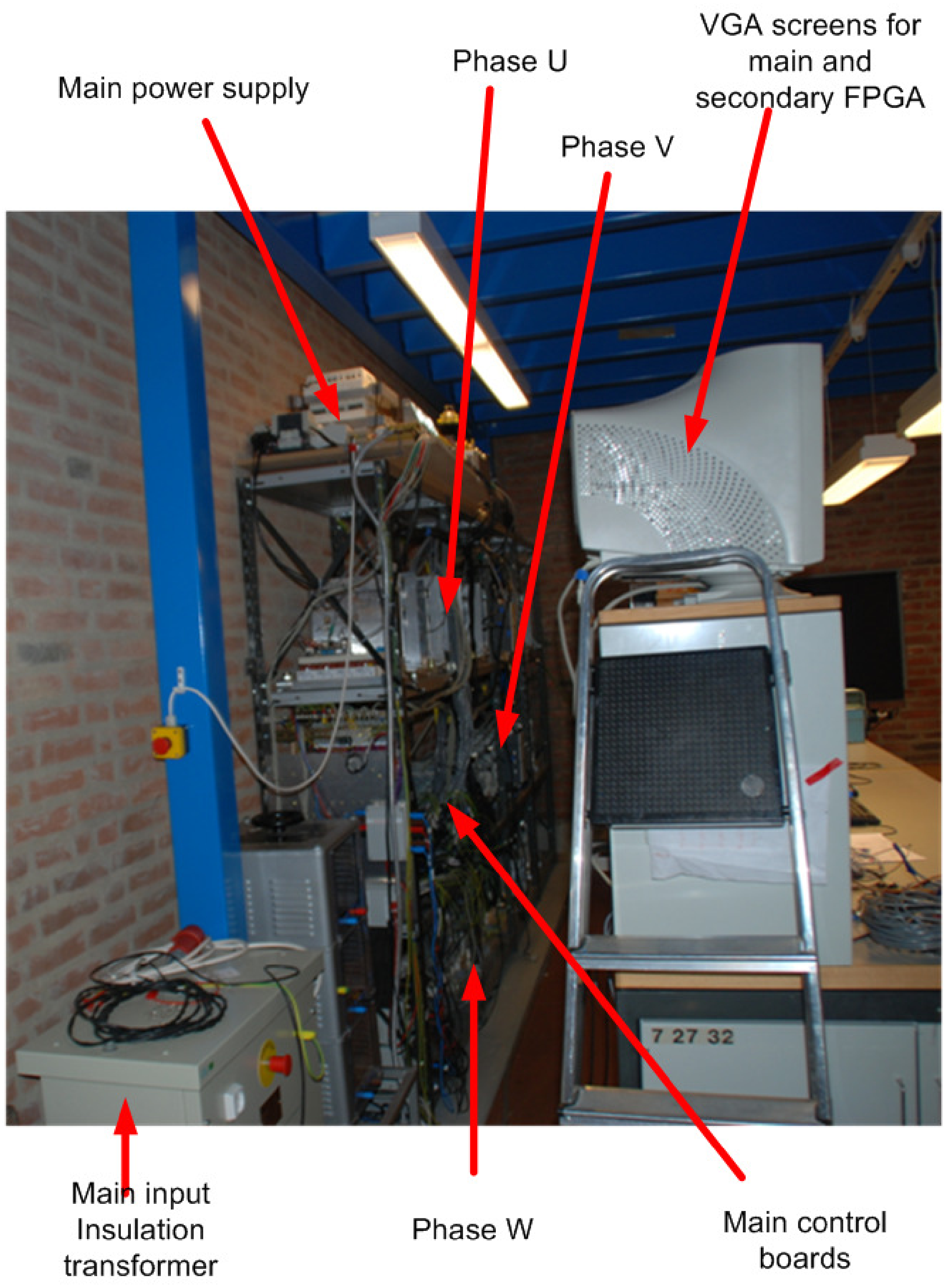

Main power supply
Phase U
VGA screens for main and secondary FPGA
Phase V
Main input Insulation transformer
Phase W
Main control boards

Figure 6-1: Setup

The setup has the each phase on a separate metallic shelf. The insulation transfer was used for protection purposes as it represents the main interface between the grid and the inverter. The power supply of the inverter, located at the top connects with the other three phases on the edges of the shelf. The front-most self side represents the +VDC while the back most self side is the –VDC. The main control boards are located in the middle in order to minimize the distance between them and the IGBTs.

The main control unit of the setup is represented in Figure 6-2. The FPGAs and DSP were not shown for clarity reason. The DSP would have covered most of the picture because it sits in front of the main board on a support.

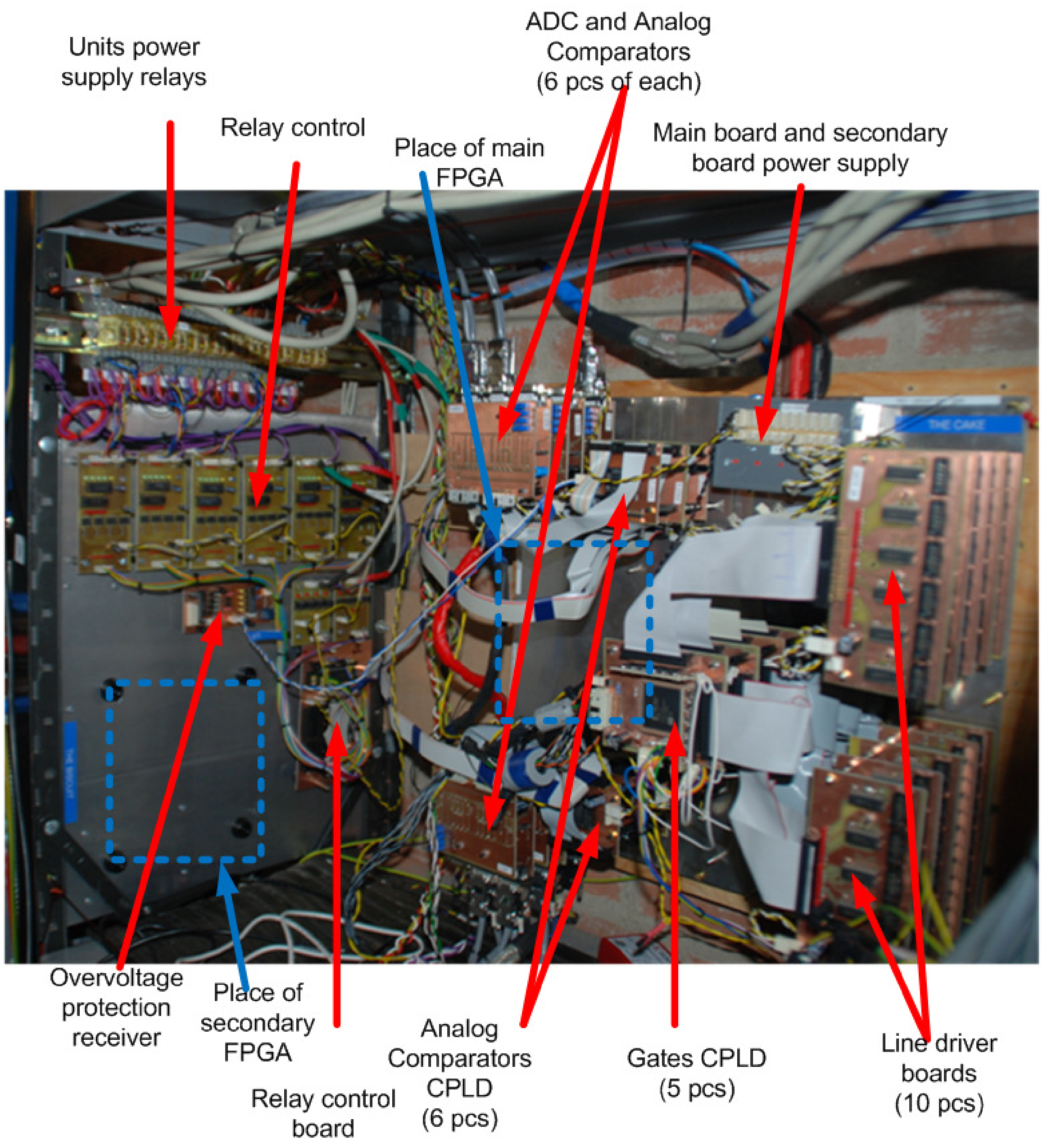

Units power supply relays
Relay control
Place of main FPGA
ADC and Analog Comparators (6 pcs of each)
Main board and secondary board power supply
Overvoltage protection receiver
Place of secondary FPGA
Relay control board
Analog Comparators CPLD (6 pcs)
Gates CPLD (5 pcs)
Line driver boards (10 pcs)

Figure 6-2: Setup control boards (FPGA boards and the DSP were removed for clarity)

The set-up control board are placed on aluminum sheets in order to ground them more easily as well as the other cables shields. In the above figure, the left board, called the secondary board, contains the relay control board as well as the overvoltage protection receiver. The main board, located on the right and called the main board in chapter 2, contains the main FPGA, the gates CPLD, ADC, analog comparators and analog comparators CPLDs.

The high power unit boards are shown in Figure 6-3.

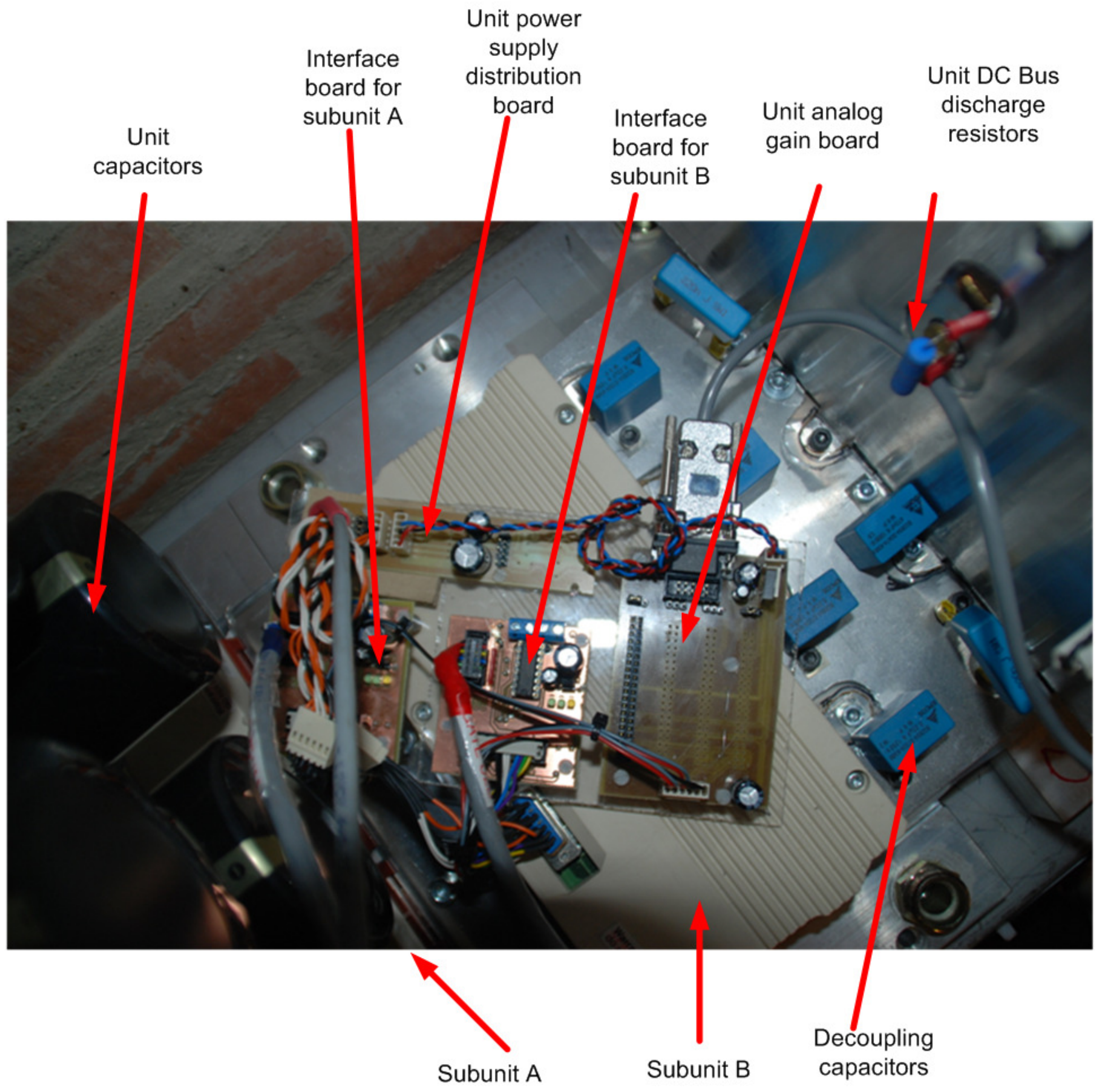

Unit capacitors
Interface board for subunit A
Unit power supply distribution board
Interface board for subunit B
Unit analog gain board
Unit DC Bus discharge resistors
Subunit A
Subunit B
Decoupling capacitors

Figure 6-3: Unit boards

The unit boards for subunits A and B as well as the unit analog gain board are placed on a single board which is located right on top of the corresponding units. This placement of the board ensures that the noise is kept to a minimum.

6.2 THE ADC TESTS

The ADC test was conducted with respect to the SKiiP 3 units in order to test the analog gain boards and the FPGA acquisition. First the ADC data communication was tested in order to ensure that data arrive property to the FPGA. The test was made with the FPGA running and connected with the ADC. The data was acquired with a logic analyzer and shown in Figure 6-4. In the figure the first three lines represent the channel address (0, 1, 2, 3, 4, and 5). The 3rd line represents the chip select command, an active low signal. The 4th line is not connected to anything while the 5th line represents the ADC clock. The lines 6 and 7 represent the data received by the FPGA form the ADC.

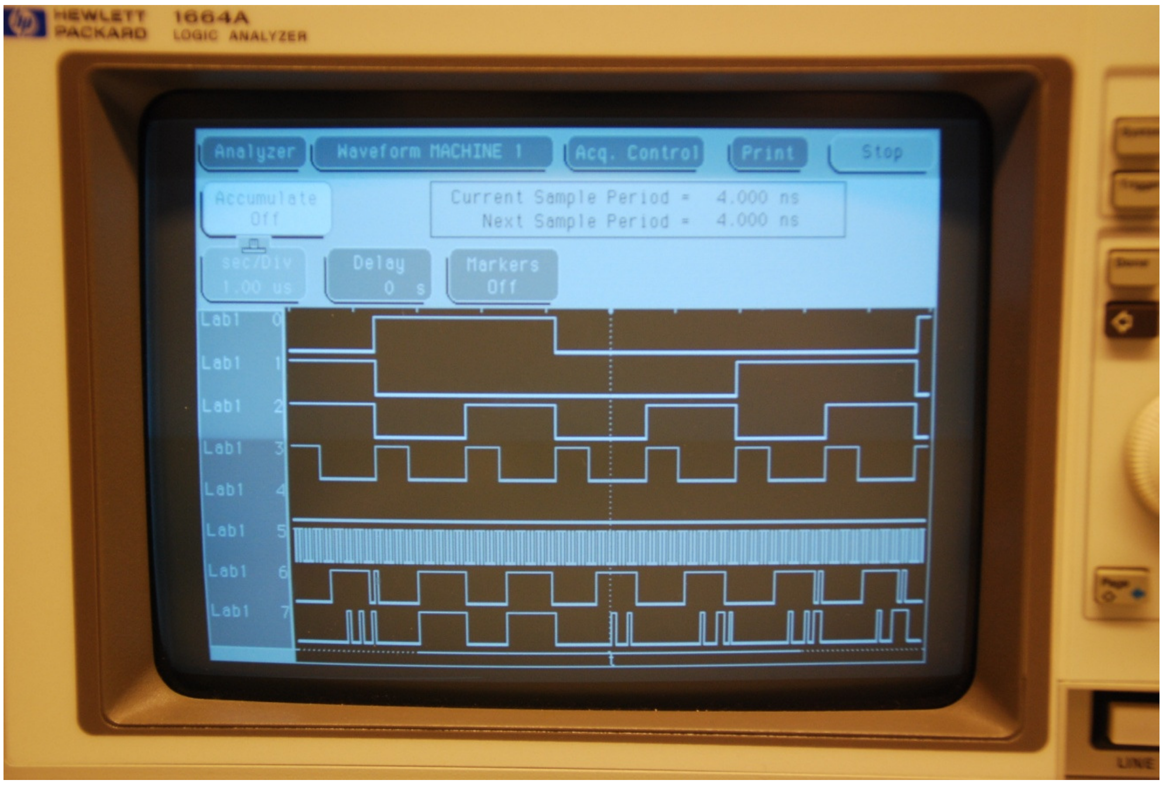
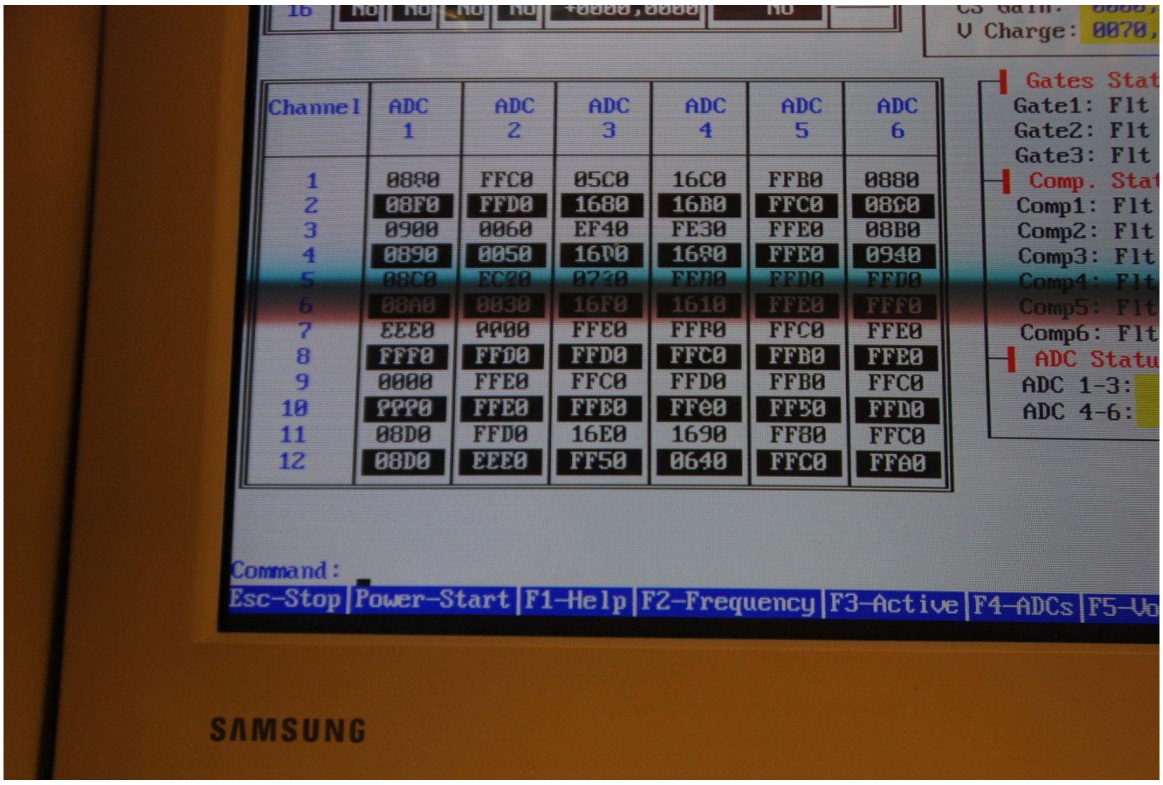


Figure 6-4: ADC Communication

The data was handled by the FPGA and transferred to the screen where the actual measurement was represented in hexa-decimal representation in Figure 6-5. The values that represent the 8 units of the U leg are located at ADC 3 and ADC 4, lines 2, 4, 6 and 11. The value of 0x1680 in hexa-decimal representation is the equivalent of 20 V. The scale is valid only for these units while for the others it was measured for the last project (See [13]). The values correspond to the acquisition made with the scope meter as presented in Figure 6-6. The test was also made to detect the gain between the measured value and the actual one. The gain was determined to be 276 after several measurements for various voltages.



V Charge: 0070,

Channel	ADC 1	ADC 2	ADC 3	ADC 4	ADC 5	ADC 6
1	0880	FFC0	05C0	16C0	FFB0	0880
2	08F0	FFD0	1680	16B0	FFC0	08C0
3	0900	0060	EF40	FE30	FFE0	08B0
4	0890	0050	16D0	1680	FFE0	0940
5	08C0	EC20	0720	FEA0	FFD0	FFD0
6	08A0	0030	16F0	1610	FFE0	FFF0
7	EEE0	0000	FFE0	FFF0	FFC0	FFE0
8	FFF0	FFD0	FFD0	FFC0	FFB0	FFE0
9	0000	FFE0	FFC0	FFD0	FFB0	FFC0
10	FFF0	FFE0	FFE0	FFe0	FF50	FFD0
11	08D0	FFD0	16E0	1690	FF80	FFC0
12	08D0	EEE0	FF50	0640	FFC0	FFA0

Gates Stat
Gate1: Flt
Gate2: Flt
Gate3: Flt
Comp. Stat
Comp1: Flt
Comp2: Flt
Comp3: Flt
Comp4: Flt
Comp5: Flt
Comp6: Flt
ADC Statu
ADC 1-3:
ADC 4-6:

Command:

Figure 6-5: FPGA screen with the displayed ADC values

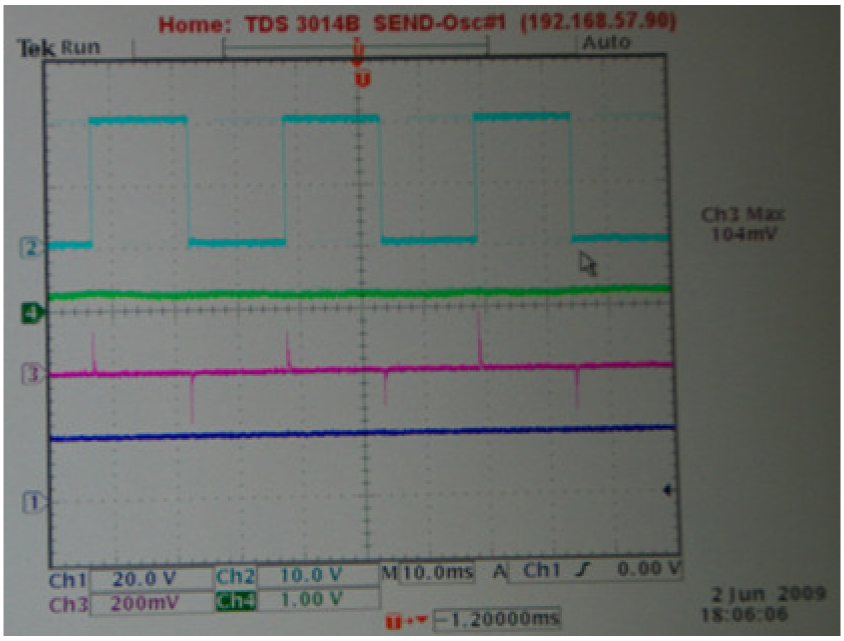


Figure 6-6: Acquisition made with the oscilloscope for one unit (Ch1: DC bus voltage of the unit; Ch2: output wave form; Ch3: none, Ch4: Measured voltage before analog gain board)

The acquisition presented in Figure 6-6 was made by firing the IGBTs of one unit through the gates CPLD. By doing this acquisition the gates CPLD, interface boards and IGBT units were tested as well as the analog gain board.

6.3 GATE DRIVERS

The tests over the gate drivers was made in order to test if the new software for the CPLD works as well as the interface boards, communication, gate drivers and IGBTs.

The test was made with a microcontroller connected with the gates CPLD and giving the signals from Figure 6-7. In the figure the lines are:

0. Not Connected
1. Data output 0
2. Chip select
3. Clock
4. Output enable
5. Not Connected
6. Not connected
7. Data output 1

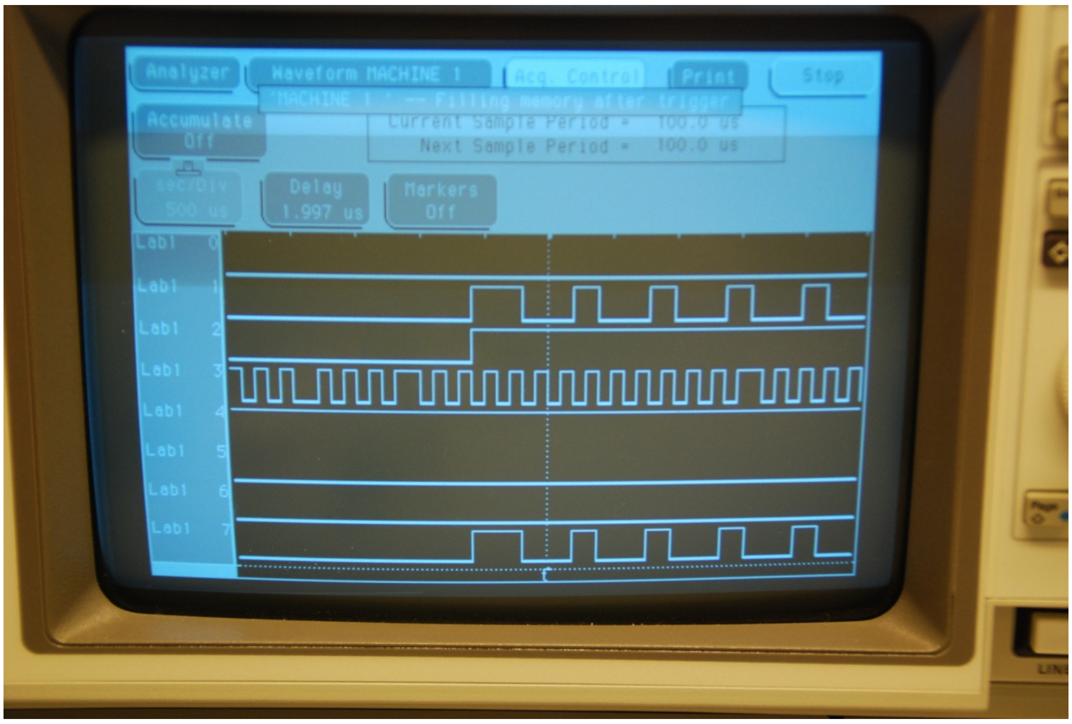


Figure 6-7: Gates test signal

The data is outputted as presented in the gates CPLD sections in chapter 5 as well in the Appendix E. The signal presented herein represents the triggering for the first unit only. The other pulses, according to the protocol, are offline. The clock delays are caused by the microcontroller due to variable number of operations done per cycle. The same output was generated by the FPGA and the results are found in Figure 6-6.

7 CONCLUSION

The main purpose of the project was to build a three phase cascaded full H-bridge inverter in order to perform tests with several modulations strategies. The modulations were simulated in Matlab and Plecs with a model identical with that found in the lab. The results obtained were satisfactory taking into account that no filters were used. The measured THD was lower than 5 % in most cases.

Phases shifted modulation has proved that better simulated results are obtained at lower switching frequencies.

Therefore the level shifted modulation is not suitable for lower frequencies but the simulated results are improved with the frequency increase. This is not quite optimal due to the lack of voltage balance that the method produces over the IGBT units. This can prove to be unsuitable for high voltage inverters due to the large stress.

Staircase is easy to implement and the result can prove satisfactory.

Laboratory tests were performed to individually components of the setup. The communication between components proved their functionality. The IGBT gate drivers were tested in connection with the CPLD communication and proved that worked. ADC was tested as well as the voltage sensors of the new IGBT units. The FPGA output towards the gate drivers was also tested with success.

Software tests were also performed on the FPGA software modulations in order to test the implementation capabilities. The FPGA software prove to be working properly during simulations.

The power supply control as well as automatic power up of IGBT units has also been tested with 3 seconds delay between power-up of consecutive units. This prove to be a good timing in order to limit the inrush current when powering up the IGBT gate drivers. This concludes the functionality of the system relays.

8 FURTHER WORK - POSSIBLE UNIT DESIGN

The chapter describes the possible unit design that was proposed at the beginning of the project. The design contains the most relevant information about how the system should look like with respect to modularity. The design is based on having a unit with all the electronics on-board including a switched mode power supply, gate drivers, microcontrollers and other components that would assure a proper functionality.

8.1 MAIN SYSTEM COMPONENTS

The modular multi-level inverter contains a series of components each with its own role. The main components are the IGBT units which form the three phases of the inverter. On each phase there are 10 IGBT units allowing up to 6 levels per phase and 11 levels in total. A unit can handle voltages up to 600 V but 800 V can also be achieved if the units are pushed at higher voltages. By having 600 V (800 V) per unit, the DC bus can have a voltage of 3 KV (4 KV).

The system components are:

- IGBT Units (30 units in total)
- Current sensors (4 sensors with at least 6 KV isolation)
- Voltage sensors (4 sensors for voltages up to 4 KV)
- Contactors (2 contactors)
- DC Power Supply

The IGBT units contain the IGBT packs as well as on-board electronics and power supply for individual functionality.

The current and voltage sensors are used for measuring the parameters used by the main control as a feedback loop.

The contactors are used in order to allow connection/disconnection of the units for normal operation or for fault related problems.

The DC Power supply is the main energy provider for the inverter as it does not use solar cells, batteries nor is it used as a static compensator.

All of these components are joined together by the control system which has as its main components an FPGA. The FPGA is used to centralize the data from the sensors and units in order to be able to control the whole inverter according to the user and application requirements.

The system is represented in Figure 8-1 without the control components as they will be presented in the following sub-chapters.

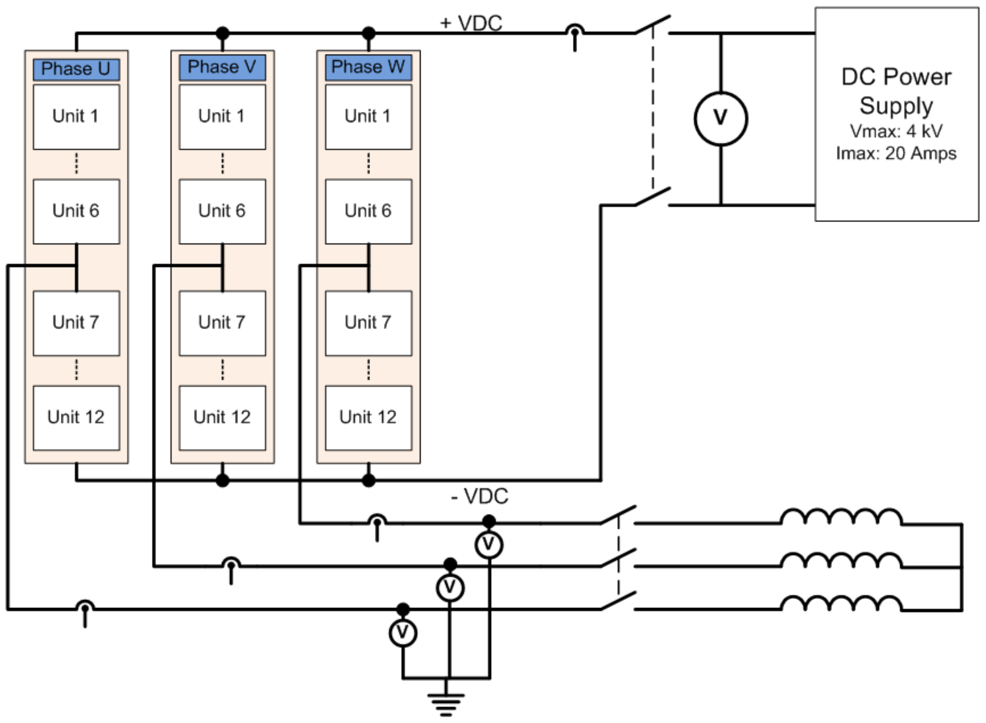


Figure 8-1: Application structure (the control blocks have been omitted for representation purposes)

8.2 MAIN LOGIC COMPONENTS

The main logic components represent the base of the application as they are responsible for the application functionality. The components that the main system uses are:

- FPGA
- ADC module
- Computer communication
- Units communication
- Inverter temperature control

All of these components can be represented in a hierarchical way in order to better understand the connectivity and relation between them. Such representation is depicted in Figure 8-2.

The FPGA is used to control the inverter as it centralizes the data from all the units and sensors. The FPGA analyze the data and commands each unit in order to achieve the desired response as the implemented control structure dictates.

The ADC module is required as the FPGA does not incorporate any analog to digital converter. The ADC is responsible to acquire the data from the current and voltage sensors in order to provide the feedback loop for the control.

The computer communication layer is represented by two interfaces one based on wireless with speeds up to 128 kbps and the other with optic fibers for high speed data transfer. The wireless system uses two RS232 wireless modems in order to command the turn ON/OFF the system or to set the current working parameters. The high speed interface is dedicated to monitor the voltages, currents and the actual units. On this interface measurements will be sent to the computer as fast as possible.

The unit communication is achieved by using two optic fibers for each unit: one for transmission and one for receiving. The data rate on these optic fibers can be up to 144 Mbps (5 m) depending on length and fiber quality. Both FPGAs are capable of communication speed up to 650 Mbps, the speed is limited only by the fiber interface.

The inverter temperature control is responsible with the enclosure temperature monitoring and cooling. The cooling system contains several fans that provide fresh air to the inverter mainly to the units. The temperature monitoring is done with the help of several NTC temperature sensors placed in appropriate location where temperature build-up may occur.

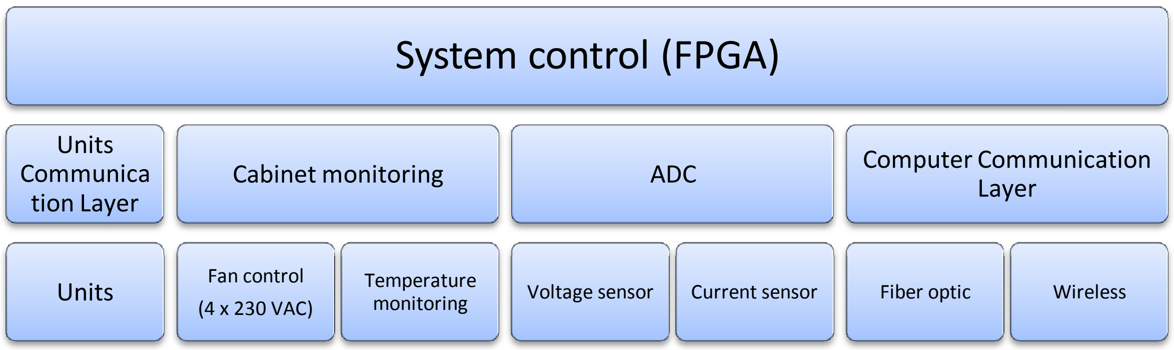


Figure 8-2: Main system logic components

As a better overview of the entire system from the control logic point of view is depicted in Figure 8-3. In the figure the isolation lines are marked in order to show where communication between components is made over fiber optics. By using this figure, the role of each block can be shown as well as the relation between them.

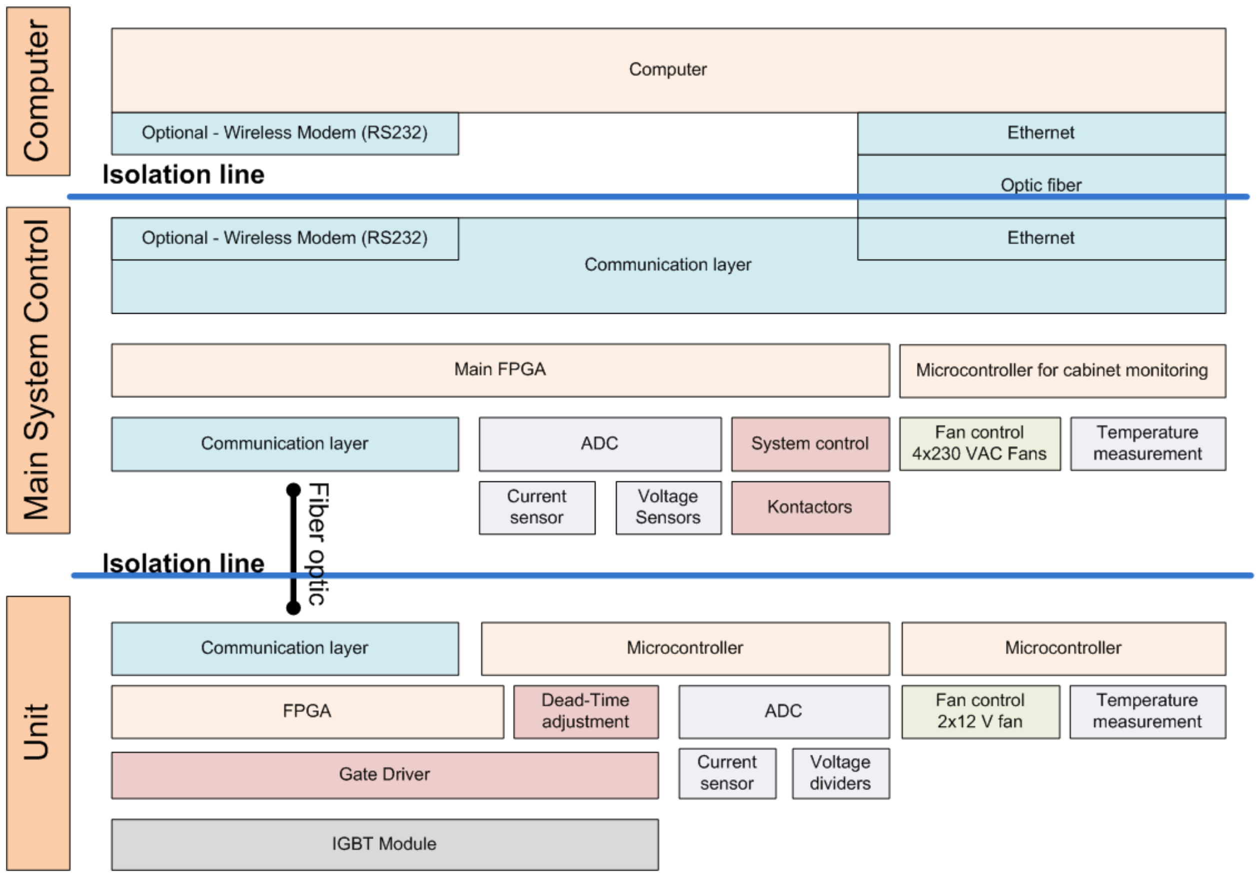


Figure 8-3: Entire system logic structure

8.3 UNIT COMPONENTS

The IGBT units are composed of 4 main sub-systems each handling a different part. The subsystems are:

- IGBT High power unit
- Switched mode power supply
- Controller
- Temperature control
- Analog comparator board
- Fiber optic board

The IGBT High power unit contains the gate driver and the IGBT pack, DC Bus capacitors and charging resistors, various filters and the current and voltage sensors. The module interfaces with all the other boards. The controller gives the signals to the gate driver and reads the desaturation status plus the readings from the sensors, the switched mode power supply offers voltage to the boards while the temperature control boards reads the IGBT pack NTC sensor.

The switched mode power supply connects to the DC bus of the inverter in order to generate all the required voltage levels. The power supply uses the energy from the DC therefore it discharges the unit capacitors.

The main schematic of a unit is represented by Figure 8-4.

The controller is represented by one FPGA and one microcontroller connected over a parallel interface. The interface is used to send data between them. The FPGA will handle the communication with the master controller (the system FPGA) and the pulses validation. The microcontroller role is to provide ADC functionality and the capability of controlling the PWMs. Both components have an individual EEPROM in order to be programmed one at a time or both in the same time over the serial connection from the master controller.

The inverter temperature control is used to optimize the fan use in order to prevent units DC capacitors to discharge if not required. The whole application is based on controlling the DC capacitor charge of each unit so the power consumption of the units must be controlled starting with the cooling system. The temperature control will monitor the IGBT pack NTC sensor as well as the heat sink temperature and the fan speed in order to better control the unit cooling.

The analog comparator board is used to compare the analog signals received from the voltage dividers and the current sensors to a predefined voltages corresponding to a threshold. If the threshold is reached or jumped, the comparators will trigger a fault signal that will be picked up by the FPGA. The fault signal will have a shut-down effect over the entire unit.

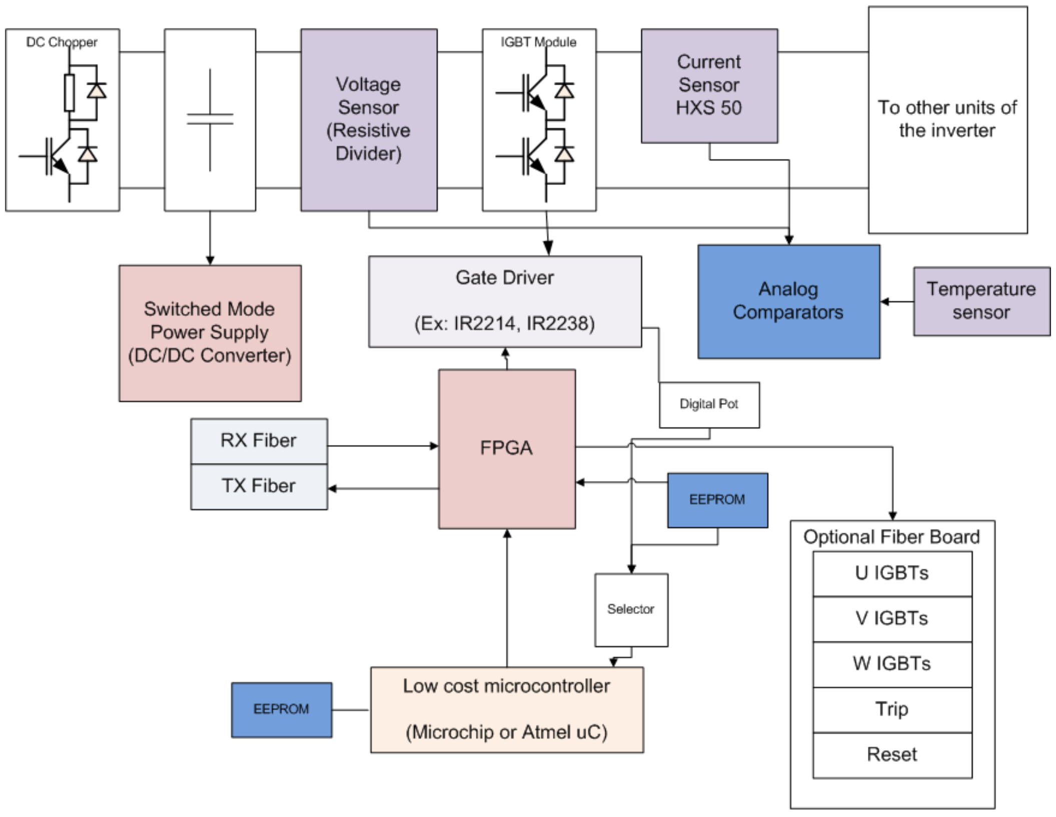


Figure 8-4: Unit connectivity

The fiber optic board is used in order to communicate with the main system over fiber optic at relative high speed. The fiber optic board uses analog circuits in order to boost the transmitted signal and a smith trigger for the receiving part. The fiber optic board communicated with the FPGA on differential lines in order to allow a low influence over the communication lines. The differential lines also allow a high speed communication to take place without the concerned over noise or other perturbing factors.

The DC chopper is used to protect the DC bus from over-voltages as well as providing a way to rapidly decrease the DC bus voltage as the command structure decides to.

A unit can also have a hierarchic representation:

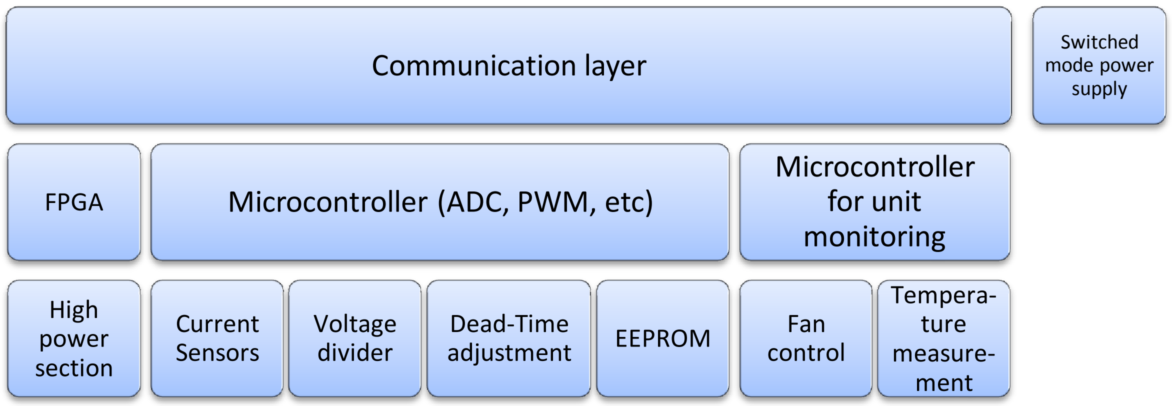

Communication layer
Switched mode power supply
FPGA
Microcontroller (ADC, PWM, etc)
Microcontroller for unit monitoring
High power section
Current Sensors
Voltage divider
Dead-Time adjustment
EEPROM
Fan control
Tempera-ture measure-ment

Figure 8-5: Main unit logic components

The Figure 8-5 shows the basic unit components and subsystems while Figure 8-4 shows how the entire unit components are connected one another.

8.4 SYSTEM CONSTRUCTION

The system can be constructed into a metallic enclosure in order to be compact and to prove that the system is modularized and it is quite simple to remove/replace one unit. The system enclosure is cooled by the help of several fans that provide air flow to the units. The enclosure also prevents the touch of exposed wires and parts which can cause electrocution. By assembling the system in this way, the possibility of moving it as a whole system is therefore possible.

8.5 UNIT - HIGH POWER MODULE

The high power module contains all the high power components of a single unit like the IGBTs filters etc. The high power module also incorporates DC link capacitors and charging resistors plus a set of voltage and current measurement devices like current sensors and voltage dividers.

8.5.1 IGBT

The IGBT used in this application is a standard 6 Pack IGBT unit made by Danfoss code 25H1200T. The internal schematic of the pack is represented in Figure 8-6.

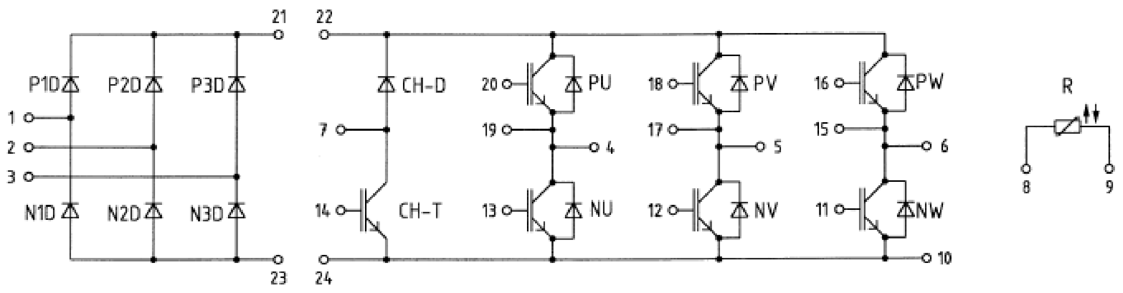


Figure 8-6: IGBT internal schematic (picture taken from the IGBT Pack datasheet)

The IGBT contains a DC chopper as well as a three phase rectifier. For the application only the DC chopper and two legs of the inverter side will be used. All the related components with the parts that are not used will not be mounted on the board but place for future mounting will be left.

8.5.2 GATE DRIVER

The gate driver used in this application should be able to control all the required switches. A gate driver dedicated for this type of IGBT pack had been selected from the International rectifier: IR22381. The driver offers adjustable dead-time as well as desaturation protection for the IGBTs. The schematic of this gate driver is shown in

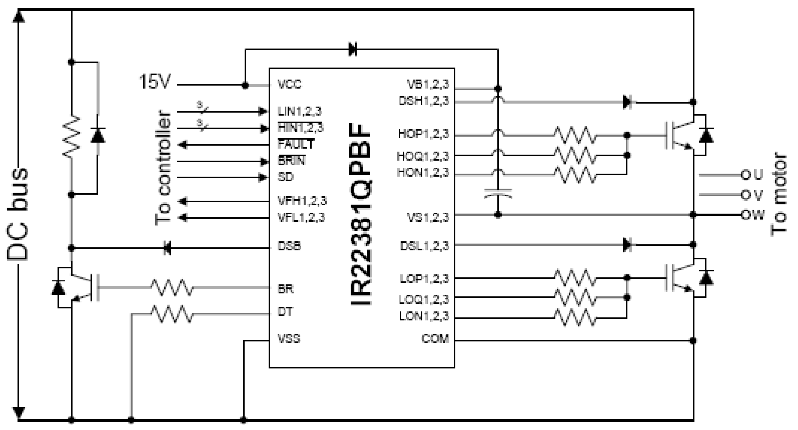


Figure 8-7: Gate driver connection schematic (picture taken from the gate driver datasheet)

The gate driver does not offer any isolation between the inverter and the control part, isolation not required for the current application. Among the signals that the driver can provide to the control part are:

- General fault (an open collector output)
- Desaturation signals for each IGBT from the inverter

The dead-time can be adjusted with a simple resistor but for this application a digital potentiometer was used in order to be able to adjust the dead-time by digital means. On the board place had been left for a simple resistor and for a potentiometer if manual adjustment is desired.

8.5.3 VOLTAGE DIVIDERS

The cheapest solution for voltage measurement for low voltage measurement is the resistive divider. When compared to voltage sensors like LEM LV-25 which cost more than 300 DKK, the voltage dividers offers the lowest price on the market with the cost of isolation. The voltage divider does not offer any galvanic isolation between the measured voltage and the measurement device. For this application this solution was used because no isolation is required, each unit being capable of handling on its own. Further on, each unit communicates with the central control system through fiber optics which is the main isolation barrier between the units and between the units and the system.

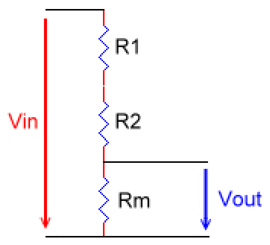


Figure 8-8: Voltage divider

The voltage dividers are placed in those locations where the importance of voltage measurement could be crucial. There are several voltage dividers that are used for the following purposes:

- Output phase measurement (3 voltage dividers)
- Input phase measurement (3 voltage dividers)
- DC Bus voltage measurement after and before the filter (2 voltage dividers)
- DC Bus capacitor voltage measurement (1 voltage divider)

The output phase voltage measurement can be used on the control board with a low pass filter in order to obtain the sine wave for the ADC.

The input phase voltage measurement is required in order to determine the input voltages and to check for wrong connection of the inverter (the output connected to the input). It can also be used to check for input voltage waveform and number of phases present. This will help in determining the limitation of the inverter with respect to the DC capacitor value (determine the voltage ripple).

The DC bus voltage measurement helps in establishing the real DC voltage in order for the PWM duty cycle calculation.

The DC Bus capacitor voltage measurement helps in determining the charge state of the capacitor useful at power up. The charging resistor is short-circuited when the DC bus voltage is very close to the capacitor voltage (charge complete). At this point, the relay that will disconnect the DC charging resistor will be switched ON so all the current from and towards the capacitors will pass through it.

The voltage dividers are sized in such a way that a 0,6 W or 1W resistors are required for the main resistors. The smallest resistor must be connected in parallel with a capacitor (depending on the voltage divider location) in order to filter the voltage. This resistor and the capacitor can be rated 0,6 W or less and MUST be placed as close as possible to the board terminals for the command board. The output voltage of the voltage dividers has a rating of 800 V / 3.3 V. This is not quite equal for each voltage divider as the number of resistors as well as resistors value may differ from one voltage divider to the other. For the software implementation the resistors will be measured for each unit in order to determine the actual gain of each set of voltage dividers.

In order to calculate the voltage divider the peak voltage for which the divider will be used must be known. The output voltage of the voltage divider is (see Figure 8-8):

$$V_{out} = V_{in}\frac{R_M}{R_{total}}$$

$$R_{total} = R_1 + R_2 + \cdots + R_n + R_M$$

Equation 8-1: Voltage output divider formula

In Equation 8-1 the V_{in} represents the input voltage while the V_{out} represents the output voltage. The R_1 to R_n represents the main resistors and R_M represents the measurement resistor. In order to size them the power dissipation on each should be measured by using the basic formula: $P = UI = U^2R$.

The voltage divider must be able to charge a small capacitor and also to offer a low current for the ADC therefore a current of 1 mA is more than enough for a voltage divider.

8.5.3.1 INPUT/OUTPUT PHASE VOLTAGE DIVIDER

The output voltage divider can have a reduced number of resistors because the voltage is not constant on it and switched between VDC and 0. Therefore a certain amount of time for cooling is therefore provided.

For this divider the peak voltage is 800 V and the average duty cycle is 0,5. The total resistance required in order to extract 1 mA will be:

$$R_{total} = \frac{U}{I} = \frac{800V}{0{,}001A} = 800ohms$$

Each resistor that will be used for this voltage divider has a voltage limit which can be up to 350 V for small 0,6 wire resistor. In order to prevent this limit to be reached a voltage difference of 50-100 V will be enough to prevent the resistor from being destroyed. Because of this, the minimum number of resistors that can be used is:

$$n_{resistors\_max} = \frac{V}{V_{limit}} = \frac{800V}{250V} = 3{,}2 \rightarrow 4\,resistors$$

$$n_{resistors\_min} = \frac{V}{V_{limit}} = \frac{800V}{300V} = 2{,}6 \rightarrow 3\,resistors$$

In order to reduce the track length and the number of resistors used, 3 resistors will be used for power dissipation in the voltage divider. The actual resistors that will be used is hard to determine due to standardized values of the resistors. A table can be the better approach in order to determine the resistors that will be selected. A great care should be taken for the voltage output in order not to burn the ADC of the microcontroller, therefore, the output (V_M) should be limited to 3,3 V. By taken into consideration the duty cycle average of 0,5 a resistor of 0,6 W can have the power dissipation up to 0,4 - 0,45 W. The limitation is imposed in order not to burn the resistor and also to have a 0,6 W, 1% tolerance resistor because 1 W with 1 % tolerance is not that commonly used. In order to simplify calculation will consider $R_1 = R_2 = R_3 = R$.

No.	R [kΩ]	R_M [kΩ]	R_{total} [kΩ]	I [mA]	W_R [mW]	V_R [V]	V_M [V]	V_{total} [V]
1	200	2,4	602,4	1,328	350	265,60	3,19	828,30
2	220	2,7	662,7	1,207	320	265,58	3,26	809,97
3	240	2,7	722,7	1,107	290	265,67	2,99	883,30
4	240	3,0	723,0	1,106	294	265,56	3,32	795,30
5	270	3,0	813,0	0,984	261	265,68	2,95	894,30
6	270	3,3	813,3	0,983	261	265,58	3,25	813,30
7	300	3,3	903,3	0,8856	235	265,69	2,93	903,30
8	300	3,6	903,6	0,885347	235	265,6	3,19	828,30

Table 8-1: Phase Voltage divider resistor selection

From the Table 8-1 we selected configuration 3 because it fits the power dissipation requirements, the current output is very close to 1 mA plus it leaves a relative small margin of error for the ADC. The resistors in this case should be wire resistors (not SMD) mounted at 5 mm one from the other and at 5 mm from the board (long pins). The voltage drop on a resistor is around 265 V plus the voltage can be measured up to 883 V.

The resistors that are used for the phase output can also be used for the phase input voltage divider because the latter also has variable voltage level (being a sine wave).

8.5.3.2 DC BUS VOLTAGE DIVIDERS

The DC Bus voltage divider is somehow different when compared with the phase output/input voltage divider. This is due to the fact that the voltage is almost stationary at a high voltage level therefore the voltage divider should have at least 3 resistors for power dissipation SMD type with 1 % tolerance should be considered in order to ease the placing on the board and to reduce the track length. Because a relative constant voltage will be found in the DC bus the current that the voltage divider should supply can be between 0,3 and 0,6 mA. Also the SMD components does not support a large voltage drop on them therefore the voltage drop should be maintain at least 50 V lower then the SDM resistor voltage. A normal 1206 SMD has a voltage rating of about 200 V (data taken from various manufacturers, ex: Panasonic). With this rating the voltage drop on the component should not go above 150 V.

Being a voltage divider the minimum number of resistors that must be used is:

$$n_{resistors} = \frac{V}{V_{limit}} = \frac{800V}{150V} = 5{,}33 \rightarrow 6\ resistors$$

As the formula above specifies, a minimum of 6 resistors should be used therefore will consider $R_1 = R_2 = R_3 = R_4 = R_5 = R_6 = R$. A number of 7 resistors is unjustified because the track length will become greater therefore only 6 resistors for power dissipation will be used.

The total resistance required will be around:

$$R_{total\_min} = \frac{800V}{0{,}6mA} = {\sim}1{,}3\ M\Omega$$

$$R_{total\_max} = \frac{800V}{0{,}3mA} = {\sim}2{,}6\ M\Omega$$

$$R_{total\_av} = \frac{R_{total\_min} + R_{total\_max}}{2} = {\sim}1{,}9\ M\Omega$$

Equation 8-2: Total resistance (minimum, maximum, average) for DC voltage divider

By considering R_{total} = 1900 kΩ the individual resistance value will be 330 kΩ for 6 resistors (values obtained by using standardized resistor values – 316,6 being the actual value required). By using 330 kΩ resistor the total resistance will be R_{total} = 1980 kΩ.

The selection of the resistors can also be made by using a table being the simplest way. The values displayed in the table represent only the most relevant selections that can be made with respect to the total resistance limitation as Equation 8-2 shows.

No.	R	R_M	R_{total}	I	W_R	V_R	V_M	V_{total}

	[kΩ]	[kΩ]	[kΩ]	[mA]	[mW]	[V]	[V]	[V]
1	220	5,1	1325,1	0,603	80	132,82	3,08	857,42
2	220	5,6	1325,6	0,603	80	132,77	~~3,38~~	~~Over~~
3	240	5,6	1445,6	0,553	74	132,82	3,10	851,87
4	240	6,2	1446,2	0,553	71	132,76	~~3,43~~	~~Over~~
5	270	6,2	1626,2	0,491	65	132,82	3,05	865,56
6	270	6,8	1626,8	0,491	65	132,78	~~3,34~~	~~Over~~
5	300	6,8	1806,8	0,442	59	132,83	3,01	876,83
6	300	7,5	1807,5	0,442	59	132,78	~~3,32~~	~~Over~~
7	330	7,5	1987,5	0,402	53	132,87	3,02	874,5
8	330	8,2	1988,2	0,402	53	132,87	3,30	800,13
9	360	8,2	2168,2	0,368	49	132,83	3,03	872,57
10	360	9,1	2169,1	0,368	49	132,77	~~3,36~~	~~Over~~
11	390	9,1	2349,1	0,340	45	132,82	3,10	851,87
12	390	10	2350,0	0,340	45	132,77	~~3,40~~	~~Over~~
13	430	10	2590,0	0,308	41	132,82	3,09	854,70
14	430	11	2591,0	0,308	41	132,77	~~3,40~~	~~Over~~

Table 8-2: DC voltage divider resistor selection

From Table 8-2 the final selection is made with respect to current, power dissipation and measured voltage. The better candidates are numbers 3, 5 and 11. Number 5 will be selected because the current is relatively high, the measured voltage output is scaled almost to the maximum range of 3,3 V and allows the maximum value of 76 V over measurement and the power dissipation and voltage drop fits the resistor requirements.

8.5.4 CURRENT SENSORS

The current sensor used for each module allows local control of the unit IGBT switches with respect to the current sense. The current sensor will only be placed on one phase the other phase that will be used will be wire-strapped.

The current sensor used is a low cost current sensor made by LEM (HX15) with voltage output so it can be directly connected to the microcontroller.

8.5.5 DC BUS

The unit DC bus contains several basic components like filters, capacitors, a current sensor (see 8.5.4 - Current sensors), voltage sensors (see 8.5.3 - Voltage dividers), charging resistor and relay. The unit DC bus is interrupted in order to let space for a filter if required for other applications.

8.5.5.1 PROTECTION

The DC bus contains a single varistor connected between the positive line and the unit ground. The role of this varistor is to prevent the DC bus for increasing beyond normal range in order to protect the various components which use the DC bus (capacitors, SMPS, voltage dividers etc). The varistor limits the DC bus to 1 kV.

The role of this varistor is not to protect the equipment against long periods over-voltages but for short periods of time.

8.5.5.2 MEASUREMENT

The DC bus has current and voltage sensors used for the control system in order to determine the power capability of the inverter by determining the voltage level as well as current value. These two parameters also helps in determining the power flow across the unit.

The current sensor is a LEM unit used to measure the currents up to 30 Amps. The sensor is located before the IGBT module and after the filter (if used). For more details see 8.5.4 - Current sensors.

The voltage dividers measure the DC bus voltage in 3 points:

- After the rectifier (before the filter)
- After the filter (before the IGBT module pack)
- At the DC capacitors

The measurement before the filter is not going to be used for this application it is left as an potential Add-on for further usages of the inverter unit.

The measurement between the filter and the IGBT module pack is required because it is used for both unit control as DC voltage measurement as to determine when the capacitors are fully charged.

The DC capacitor voltage measurement is used to measure the voltage near the capacitors after the charging resistor. This measurement is only used at boot-time in order for the system to determine when the capacitors are charged. (See: 8.5.5.3 - Capacitors charge control)

8.5.5.3 CAPACITORS CHARGE CONTROL

The capacitor charge must be controlled in order to prevent a large inrush current to be absorbed from the grid or from an external DC bus power supply when the inverter gets connected. In order to control the capacitor charge at boot-up a charge resistor is therefore used. The charge time is directly influence by the size of the power resistor R17 from Figure 8-9.

The charging resistor limits the current flowing towards the capacitors when the system is first connected.

When the voltage difference between the actual DC bus and the capacitors is small enough (around 10-20 V) the charging resistor is short-circuited with the help of a relay RL1 from Figure 8-9. After charge is complete, the relay must withstand the full current flowing from and towards the capacitors.

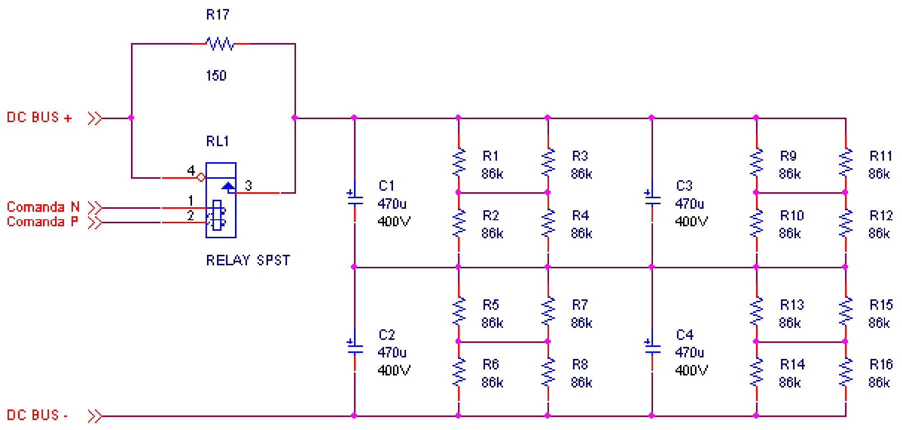


Figure 8-9: Schematic of the DC Bus capacitors with short-circuit relay (RL1)

Charging resistors tend to heat-up so great care should be considered to the resistors power dissipation. The resistor is placed in the proximity of the capacitors with an isolation material covering it.

The charging resistor is selected by considering a maximum of 4 Amps current during charge. This will determine a charge resistor of:

$$R = \frac{U}{I} = \frac{600V}{4A} = 150\Omega$$

By considering a 150 Ω charging resistor the time constant will be:

$$T = RC = 150 * 1{,}65 * 10^{-3} = 0.2475s$$

In order to determine the amount of time the capacitor needs to charge, a 100 mA current will be the lowest value considered for which the charge resistor will be used to charge the capacitor. The amount of time required to achieve the imposed current threshold is calculated from the charging current equation:

$$I = \frac{V_{input}}{R} * e^{-\frac{t}{RC}}$$

Equation 8-3: Charging current

$$t = -R * C * ln\left(\frac{RI}{V}\right) = -150\Omega * 1{,}65mF * \ln\left(\frac{150\Omega * 100mA}{600V}\right) = 0{,}91s$$

Equation 8-4: Required time for capacitor charging

For this application a single charge resistor of 150 Ω rated 8 W is used and will charge the capacitor in 0,91 seconds.

8.5.5.4 DC BUS DISCHARGE

The DC bus is used by several subsystems and therefore it discharges. The discharge process depends on the current absorbed which is not constant. Therefore it is hard to establish a time constant or a load for the DC bus. For this application the purpose is to maintain the DC bus charged as long as possible. The unit consumers must be optimized in order not to extract unnecessary energy and also to provide a discharge path for the capacitors. The discharge resistors on the capacitors will not be mounted for this application even if the resistors will be calculated in this chapter.

Besides the DC filter, IGBT module pack (load) and the discharge resistors, the DC bus is discharged by several factors like:

- A LED which shows voltage presence in the DC Bus (max 8 mA)
- The switched mode power supply (variable current absorbed from the DC bus)
- Internal DC capacitors resistors

The discharge resistors must discharge the capacitors in less than 3 minutes. If we consider that 5 time constants (5T = 3*60s) is more than enough time for the capacitors to discharge then the required resistor is:

$$R = \frac{5T}{C} = \frac{3 * 60}{1{,}65} = 109\ k\Omega$$

Equation 8-5: Required discharge resistor

8.5.6 OUTPUT STAGE

The output stage of the inverter was design so that a voltage divider and a current sensor can be placed on each phase. Because of the application structure, only one phase will have the voltage divider and current sensor mounted.

8.6 UNIT COMMUNICATION BOARD

The communication board contains the fiber optics required for unit to system communication. The board contains a set of fast fiber optics receivers and transmitters with level shifters in order to make it compatible for 3.3 V systems. The board uses 5 V in order to boost the transmitter signal and also to filter the receiving signal.

This board will be located on the unit as well as on the main system. The board was created for high speed communication for speeds up to 125 MSymbols/second.

The main components of the boards are represented by one fiber optic emitter (HFBR-15X7Z) and one receiver (HFBR-25X6Z). The transmitter has a set of components analog amplifiers and triggers in order to boost the signal by forcing a current into the fiber optic LED. The receiver contains a set of filters with the role of creating a smith trigger in order to filter the received signal. Both the transmitter and the receiver are digital devices but for this board are used as analog devices.

The electrical connection with the board is made over differential lines at 3,3 V. The differential signal allows high speed data to pass through without a large interface from the other signals.

8.7 UNIT – ANALOG COMPARATORS

The analog comparator board connects on top of the control board and provides protection for over-voltages and over-currents by comparing the analog values received from the voltage dividers and from current sensors with a set of preset values. The preset values are set with the help of several potentiometers one for each set of parameters. The parameters that will have individual voltage reference are:

- Input voltages (3 voltages)
- DC voltages (2 voltages)
- Output voltages (3 voltages)
- DC Current (1 current)
- Output currents (3 currents)

The values in parenthesis are the maximum number of possible sensors or voltage dividers. For this application only the DC voltages, output voltages and one output current will be used. Space on the board is left for the rest of the components.

The schematic used to obtain the desired reference voltage is shown in Figure 8-10. In the figure the fixed resistor will be placed only if the potentiometer is omitted. In that case, the resistors will determine the voltage output as a normal voltage divider will. The capacitors C48 and C49 from the figure are used to provide a relative stable voltage output for the reference voltage. The potentiometer will be used to adjust the reference voltage. Most of the reference voltages will have a potentiometer. The resistive divider will only be used for the currents and DC voltages because these are determined by the components used. The output/input voltages may differ so for these a potentiometer will be used.

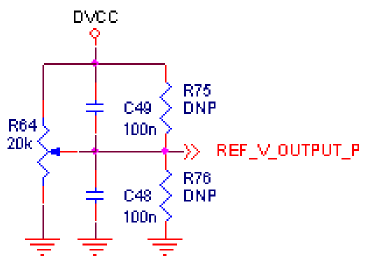
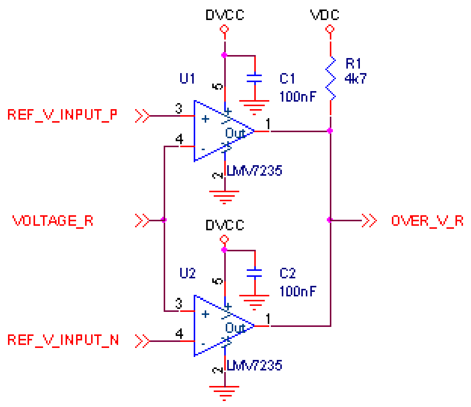


Figure 8-10: Analog comparators reference voltage selection

The number of comparators per signal varies depending on the value that will be compared. For example, the DC voltage cannot be negative (from design) and therefore only one comparator is requires (see Figure 8-12). For parameters that can have negative values two comparators are used, one for the lowest value and one for the upper value. Both absolute values (the negative and positive) represent the limited voltage/current for that particular sensor. The schematic of such comparator is depicted by Figure 8-11.



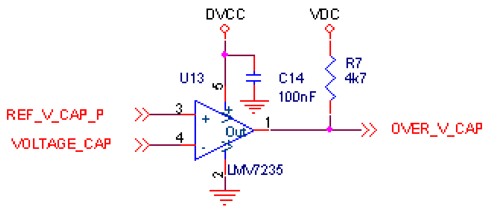


Figure 8-11: Double comparator

Figure 8-12: Simple comparator

The input analogical value is taken after the filter for the parameters that can have a negative value. The comparators are open collector so several of these can be connected on a single line with a pull-up resistor.

9 BIBLIOGRAPHY

[1] B. Wu, *High-power converters and AC drives*. Wiley-IEEE Pres, 2006.

[2] A. C. Rufer, N. Schibli, and C. Briguet, "A direct 4-quadrant multilevel converter for 16(2/3) traction system".

[3] F. Z. Peng, J. W. McKeever, and D. J. Adams, "Cascade Multilevel Inverters for Utility Applications".

[4] Sustainable Facility. (2009, May) Gen 3 Perfect Harmony drive from Rubicon. [Online]. http://www.sustainablefacility.com/Articles/Products/3a6c94b4ece38010VgnVCM100000f932a8c0

[5] P. S. Perez, D. Van Hertem, J. Driesen, and R. Belmans, "Wind power in the European Union: grid connection and regulatory issues," pp. 776-783, 2006.

[6] T. Ackermann, *Wind power in power systems*, 0470855088, Ed. Wiley, 2005.

[7] W. Brook-Hart, "Concrete foundations for offshore wind turbines ," february 2009.

[8] A. P. S. Weimers Lars, "HVDC Light, the Transmission Technology of the future," 2001.

[9] P. Bresesti, W. L.Kling, R. L. Hendriks, and R. Vailati, "HVDC Connection of offshore wind farms to the transmission system," no. IEEE Transactions on energy conversion, vol. 22, no.1, 2007.

[10] N. B. Negra, J.Todorovic, and T. f. Ackermann, "Loss evaluation of HVAC and HVDC transmission solutions for large offshore wind farms," *Science direct - Electric power systems research no.76*, pp. 916-927, 2006.

[11] S. Khomfoi and L. M. Tolbert, *Chapter 31 Multilevel Power Converters*.

[12] J. Weatherill, "First International Workshop of Feasibility of HVDC Transmission Networks for offshore Wind Farms," 2000.

[13] S. Cristian, C. Valentin, and C. Nicoleta, "Medium Voltage Bidirectional DC to DC Modular-Multilevel-Power Converter for MW power rating," Aalborg University, Aalborg, Semester report, January 2009.

[14] B. Wu, *High-power converters and AC drives*. Willey, IEEE Press.

[15] N. C. V. C. Cristian Sandu, "Medium Voltage Bidirectional DC to DC Modular-Multilevel-Power Converter for MW power rating".

[16] N. C. V. C. Cristian Sandu, " Medium voltage bidirectional dc to DC modular-multilevel-power converter for MW power rating," 2009.

[17] S. A. Bashi, N. F. Mailah, M. Z. Kadir, and K. H. Leong, "Generation of Triggering Signals for Multilevel Converter," vol. European Journal of Scientific Research, Vol.24 No.4, pp. 548-555, 2008.

[18] P. Abraham, *Switching Power Supply Design*, 2nd ed., 978-0070522367, Ed. Waban: McGraw-Hill, 1998.

[19] www6.poweresim.com. (2009, Apr.) Power Simulator. [Online]. www6.poweresim.com

[20] M. Brown, "Very Wide Input Voltage Power Supply," On Semiconductors Manual, 2002.

[21] M. Brown, "SWITCHMODE Power Supplies Reference Manual and Design Guide," On Semiconductor Manual, May 1999.

[22] C. Wm.T.McLyman, *Transformer and Inductor Design Handbook*. New York: Marcel Dekker, 2004.

10 NOMENCLATURE

$n_{carrier\_number}$ – numbers of carriers

$m_{nr\_of\_voltage\_levels}$ – number of voltage levels

φ_{cr} – phase shift angle

f_{cr} – carrier frequency

f_m – modulating wave frequency

m_f – modulation frequency index

m_a – amplitude modulation index

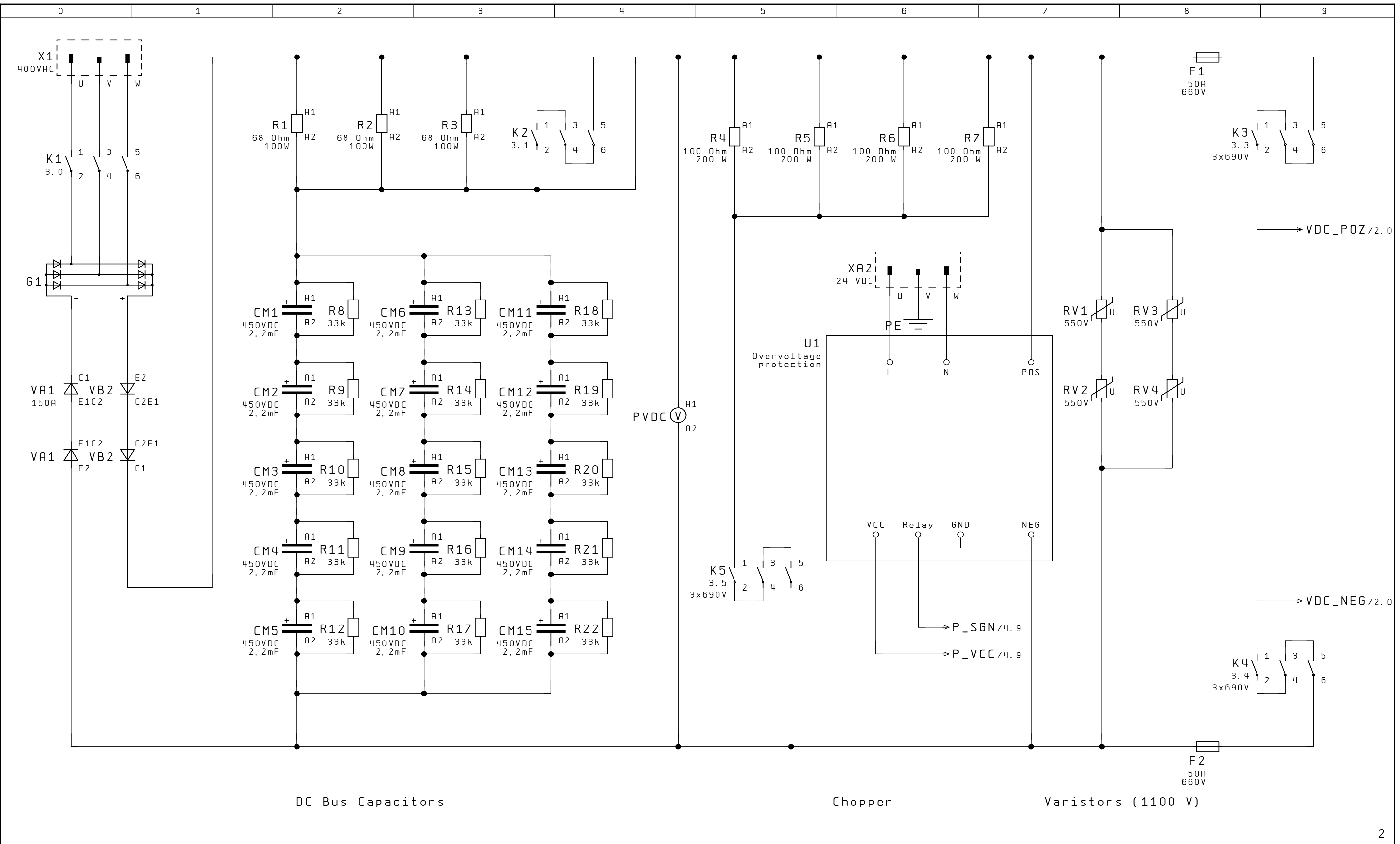
$\widehat{V_{ma}}$ – peak amplitude of the modulating wave

$\widehat{V_{cr}}$ – peak amplitude of the carrier

$\widehat{V_m}$ – peak amplitude of the modulating wave

TBD	To be discussed	ASCII	American Standard for ….
IC	Integrated Circuit	NC	Not connected
OpAmp	Operational Amplifier	EN	Enable
ADC	Analog-to-Digital Converter	FPU	Floating point unit
DAC	Digital-to-Analog Converter	CLK	Clock
AVCC	Analog Voltage supply	HRF	Human Readable Form
DVCC	Digital Voltage supply	OVC	Over Current
IGBT	Insulated Gate Bipolar Transistor	OE	Output Enable
DC	Direct Current	FLOPS	FLoating point Operations Per Second
MSPS	Mega Samples per second	MIPS	Millions of instructions per second
MOSI	Master Output Slave Input		
MISO	Master Input Slave Output		
CPLD	Complex programmable logic device		
FPGA	Field-programmable gate array		
HID	Human Interface Device		
VGA	Video Graphics Array		
OVV	Over Voltage	OVT	Over Temperature
LUT	Look-up table		

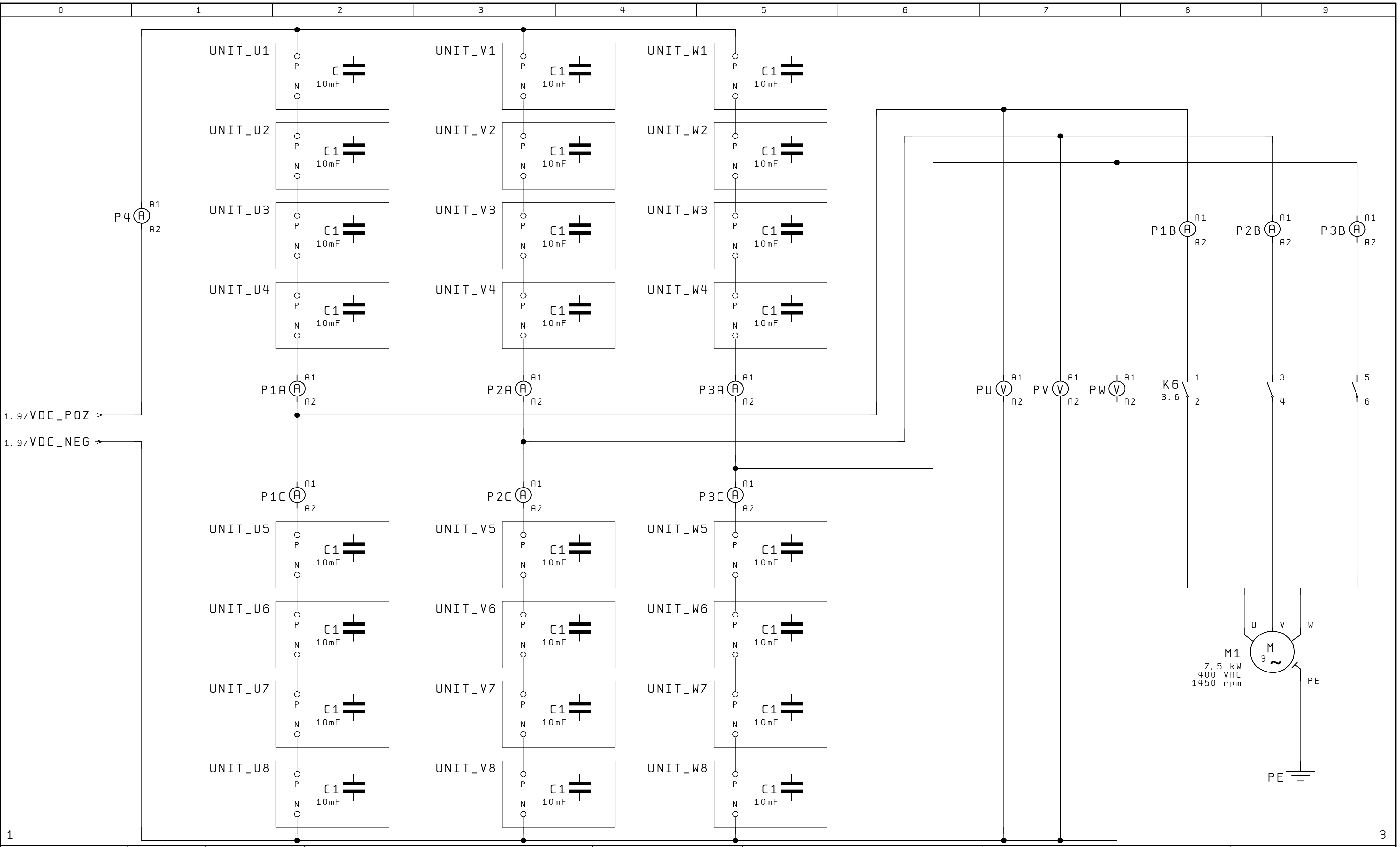
11 APPENDIX



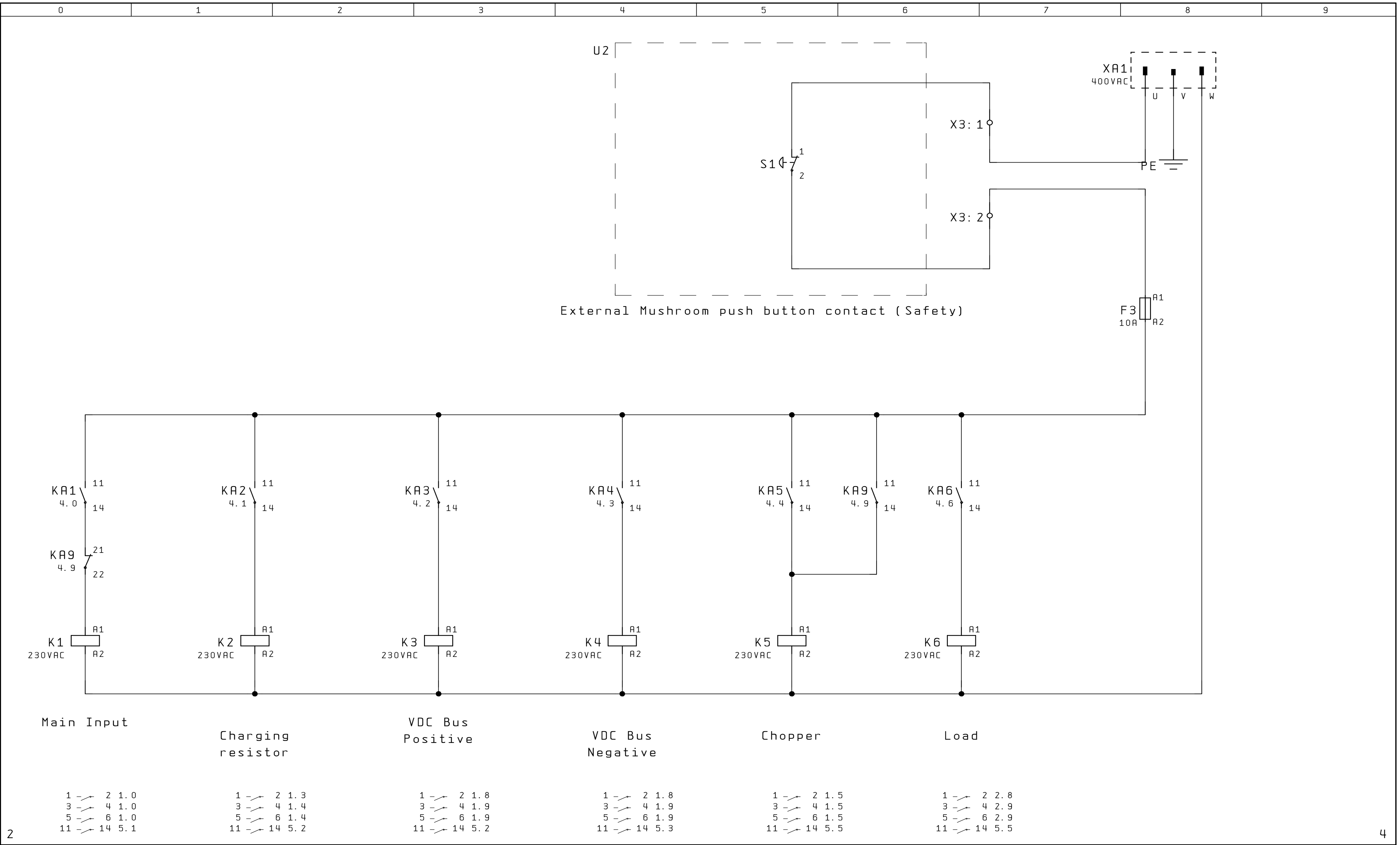
			Date	30.Oct.2008	Aalborg University	A	High power project architecture	MMLC - HVDC Project
			User	Cristian Sandu				=
			Check.	31.May.2009				+
Change	Date	Name	Stand.		First	Created for	Created by	Group 930/2008-2009

Pg. 1

5 Pg.



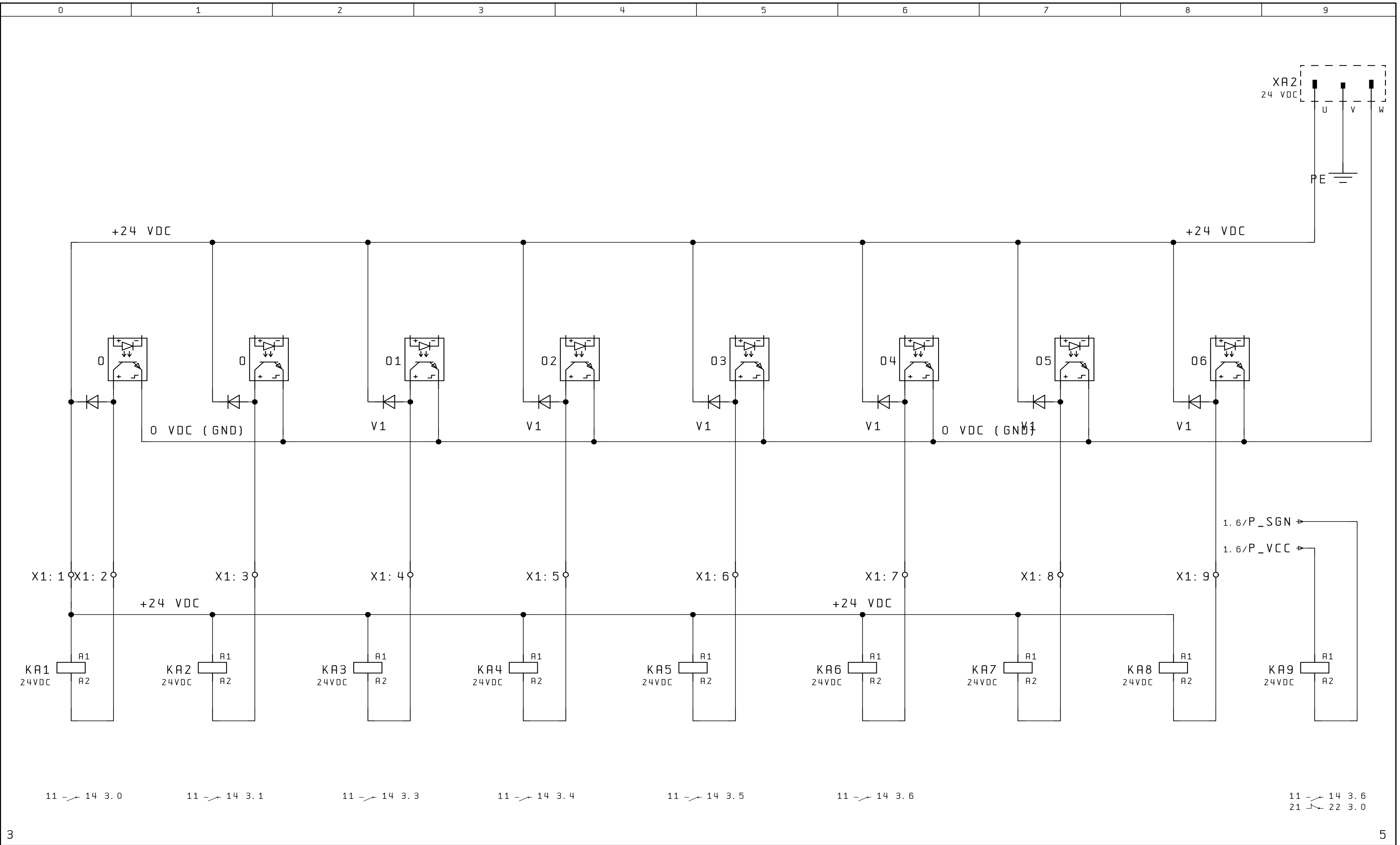
			Date		Aalborg University			A	High power project architecture	MMLC - HVDC Project	=
			User	Cristian Sandu							+
			Check.	31. May. 2009						Group 930/2008-2009	Pg. 2
Change	Date	Name	Stand.		First	Created for	ePLAN	Created by			5 Pg.



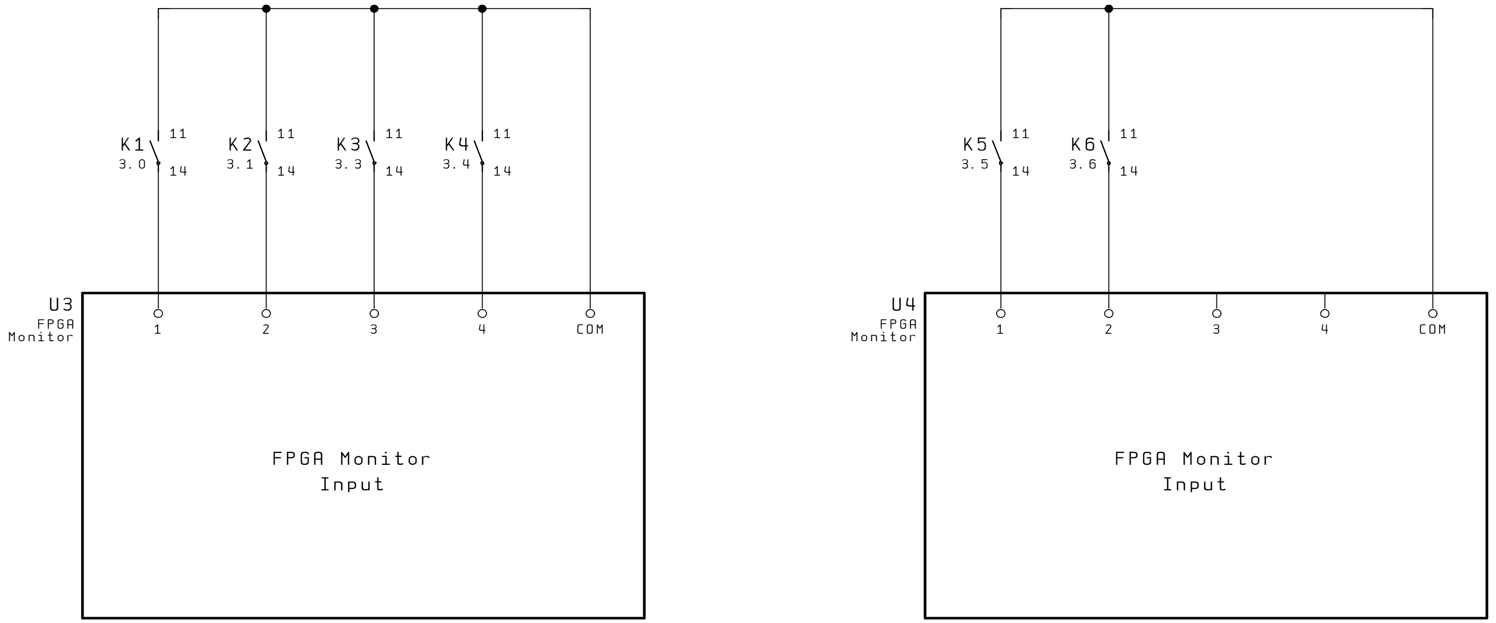
2

4

			Date	21.Nov.2008	Aalborg University	A	High power project architecture	MMLC - HVDC Project = +
			User	Cristian Sandu				
			Check.	31.May.2009				Group 930/2008-2009 Pg. 3
Change	Date	Name	Stand.		First / Created for	Created by		5 Pg.

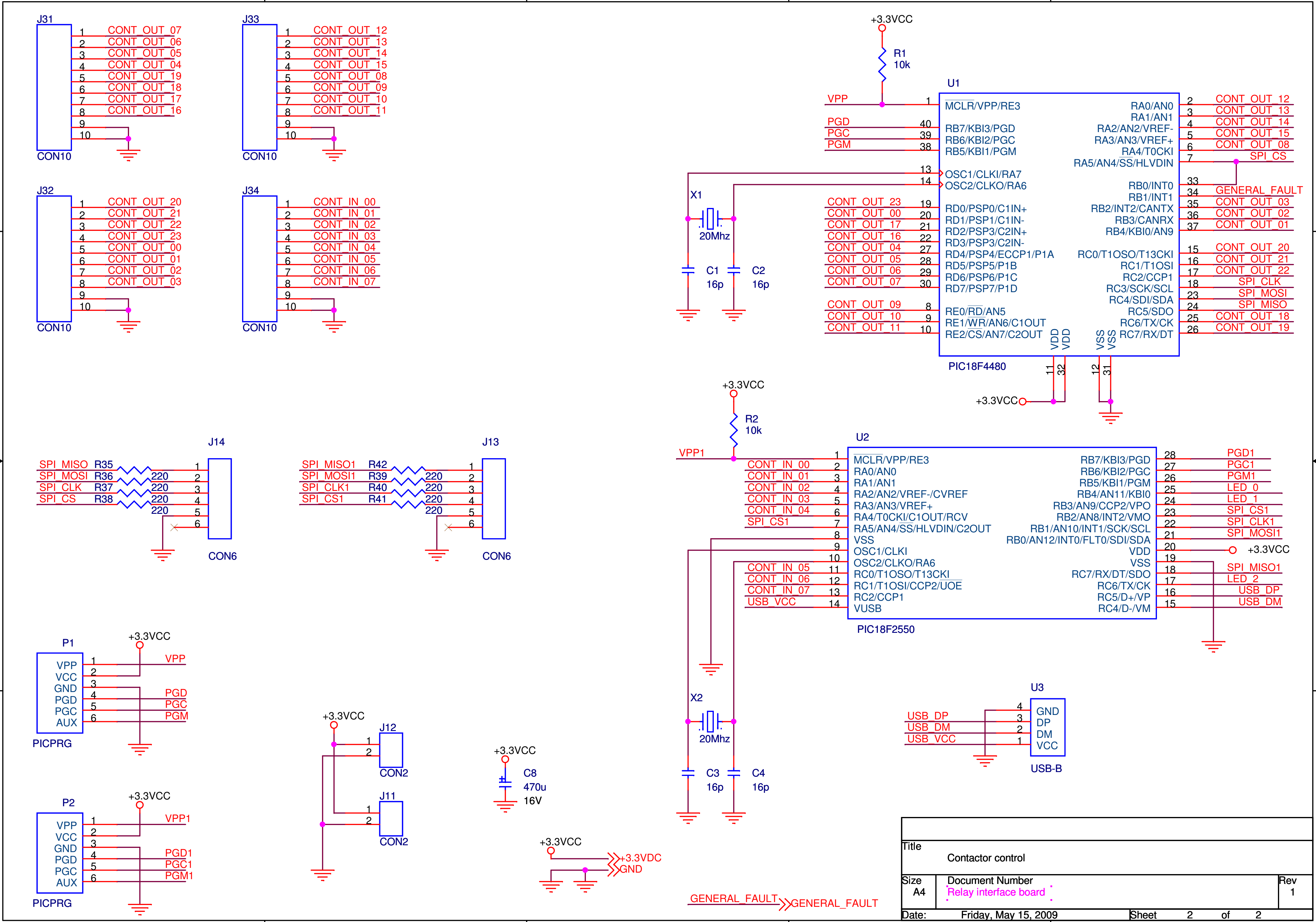


			Date		Aalborg University	A	High power project architecture	MMLC - HVDC Project	=
			User	Cristian Sandu					+
			Check.	31.May.2009				Group 930/2008-2009	Pg. 4
Change	Date	Name	Stand.		First / Created for	Created by			5 Pg.

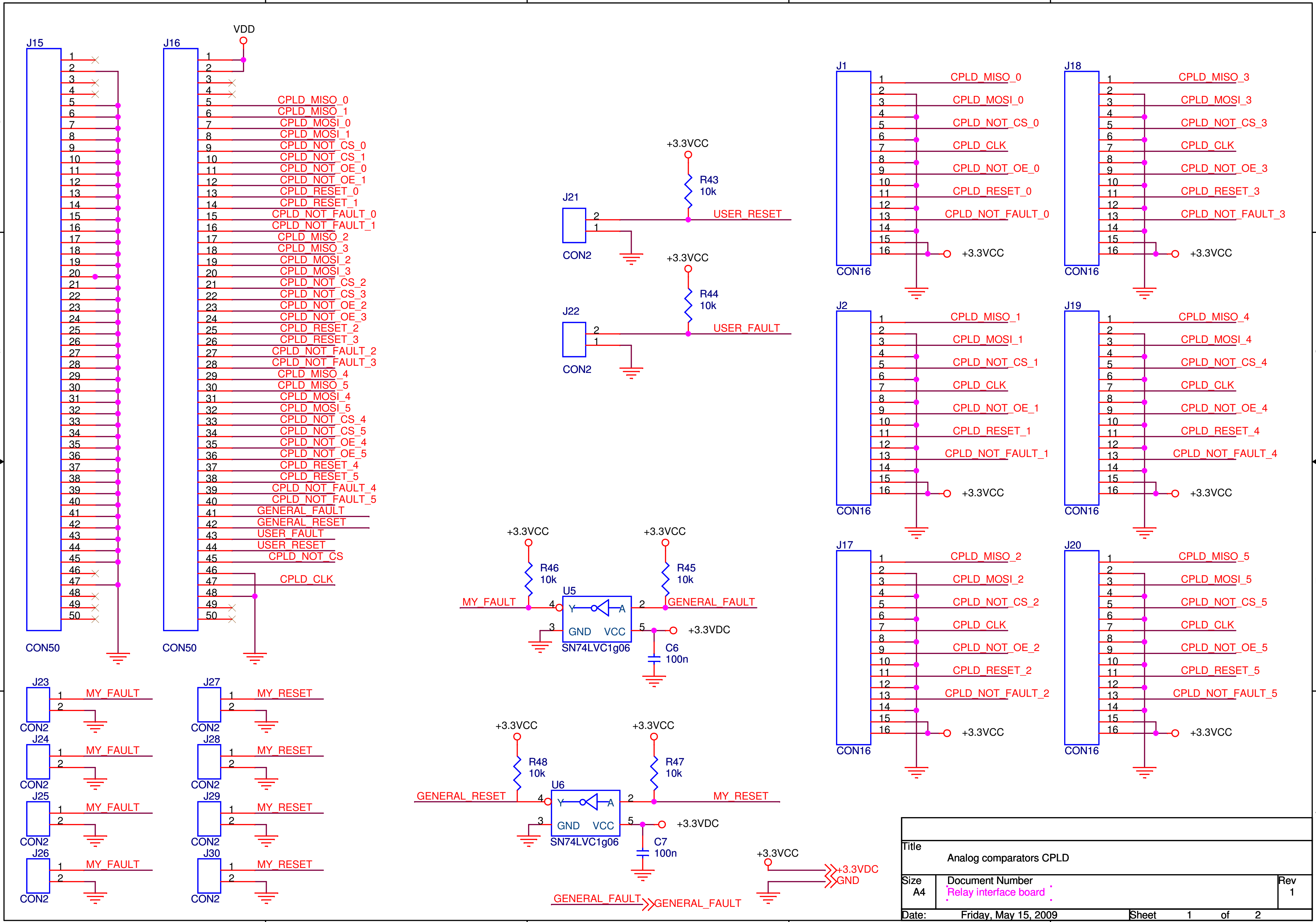


4

		Date		Aalborg University		A	High power project architecture	MMLC - HVDC Project	=
		User	Cristian Sandu						+
		Check.	31.May.2009					Group 930/2008-2009	Pg. 5
Change	Date	Name	Stand.	First	Created for	Created by			5 Pg.


J31 CON10: 1 CONT_OUT_07, 2 CONT_OUT_06, 3 CONT_OUT_05, 4 CONT_OUT_04, 5 CONT_OUT_19, 6 CONT_OUT_18, 7 CONT_OUT_17, 8 CONT_OUT_16, 9, 10
J33 CON10: 1 CONT_OUT_12, 2 CONT_OUT_13, 3 CONT_OUT_14, 4 CONT_OUT_15, 5 CONT_OUT_08, 6 CONT_OUT_09, 7 CONT_OUT_10, 8 CONT_OUT_11, 9, 10
J32 CON10: 1 CONT_OUT_20, 2 CONT_OUT_21, 3 CONT_OUT_22, 4 CONT_OUT_23, 5 CONT_OUT_00, 6 CONT_OUT_01, 7 CONT_OUT_02, 8 CONT_OUT_03, 9, 10
J34 CON10: 1 CONT_IN_00, 2 CONT_IN_01, 3 CONT_IN_02, 4 CONT_IN_03, 5 CONT_IN_04, 6 CONT_IN_05, 7 CONT_IN_06, 8 CONT_IN_07, 9, 10
+3.3VCC
R1 10k
U1 PIC18F4480
VPP 1 MCLR/VPP/RE3
PGD 40 RB7/KBI3/PGD
PGC 39 RB6/KBI2/PGC
PGM 38 RB5/KBI1/PGM
13 OSC1/CLKI/RA7
14 OSC2/CLKO/RA6
CONT_OUT_23 19 RD0/PSP0/C1IN+
CONT_OUT_00 20 RD1/PSP1/C1IN-
CONT_OUT_17 21 RD2/PSP3/C2IN+
CONT_OUT_16 22 RD3/PSP3/C2IN-
CONT_OUT_04 27 RD4/PSP4/ECCP1/P1A
CONT_OUT_05 28 RD5/PSP5/P1B
CONT_OUT_06 29 RD6/PSP6/P1C
CONT_OUT_07 30 RD7/PSP7/P1D
CONT_OUT_09 8 RE0/RD/AN5
CONT_OUT_10 9 RE1/WR/AN6/C1OUT
CONT_OUT_11 10 RE2/CS/AN7/C2OUT
RA0/AN0 2 CONT_OUT_12
RA1/AN1 3 CONT_OUT_13
RA2/AN2/VREF- 4 CONT_OUT_14
RA3/AN3/VREF+ 5 CONT_OUT_15
RA4/T0CKI 6 CONT_OUT_08
RA5/AN4/SS/HLVDIN 7 SPI_CS
RB0/INT0 33
RB1/INT1 34 GENERAL_FAULT
RB2/INT2/CANTX 35 CONT_OUT_03
RB3/CANRX 36 CONT_OUT_02
RB4/KBI0/AN9 37 CONT_OUT_01
RC0/T1OSO/T13CKI 15 CONT_OUT_20
RC1/T1OSI 16 CONT_OUT_21
RC2/CCP1 17 CONT_OUT_22
RC3/SCK/SCL 18 SPI_CLK
RC4/SDI/SDA 23 SPI_MOSI
RC5/SDO 24 SPI_MISO
RC6/TX/CK 25 CONT_OUT_18
RC7/RX/DT 26 CONT_OUT_19
VDD 11, VDD 32, VSS 12, VSS 31
X1 20Mhz
C1 16p
C2 16p
+3.3VCC
R2 10k
U2 PIC18F2550
VPP1 1 MCLR/VPP/RE3
CONT_IN_00 2 RA0/AN0
CONT_IN_01 3 RA1/AN1
CONT_IN_02 4 RA2/AN2/VREF-/CVREF
CONT_IN_03 5 RA3/AN3/VREF+
CONT_IN_04 6 RA4/T0CKI/C1OUT/RCV
SPI_CS1 7 RA5/AN4/SS/HLVDIN/C2OUT
8 VSS
9 OSC1/CLKI
10 OSC2/CLKO/RA6
CONT_IN_05 11 RC0/T1OSO/T13CKI
CONT_IN_06 12 RC1/T1OSI/CCP2/UOE
CONT_IN_07 13 RC2/CCP1
USB_VCC 14 VUSB
RB7/KBI3/PGD 28 PGD1
RB6/KBI2/PGC 27 PGC1
RB5/KBI1/PGM 26 PGM1
RB4/AN11/KBI0 25 LED_0
RB3/AN9/CCP2/VPO 24 LED_1
RB2/AN8/INT2/VMO 23 SPI_CS1
RB1/AN10/INT1/SCK/SCL 22 SPI_CLK1
RB0/AN12/INT0/FLT0/SDI/SDA 21 SPI_MOSI1
VDD 20 +3.3VCC
VSS 19
RC7/RX/DT/SDO 18 SPI_MISO1
RC6/TX/CK 17 LED_2
RC5/D+/VP 16 USB_DP
RC4/D-/VM 15 USB_DM
X2 20Mhz
C3 16p
C4 16p
J14 CON6
SPI_MISO R35 220 1
SPI_MOSI R36 220 2
SPI_CLK R37 220 3
SPI_CS R38 220 4
5
6
J13 CON6
SPI_MISO1 R42 220 1
SPI_MOSI1 R39 220 2
SPI_CLK1 R40 220 3
SPI_CS1 R41 220 4
5
6
P1 PICPRG: 1 VPP, 2 VCC, 3 GND, 4 PGD, 5 PGC, 6 AUX
+3.3VCC VPP PGD PGC PGM
P2 PICPRG: 1 VPP, 2 VCC, 3 GND, 4 PGD, 5 PGC, 6 AUX
+3.3VCC VPP1 PGD1 PGC1 PGM1
+3.3VCC
J12 CON2 1 2
J11 CON2 1 2
+3.3VCC
C8 470u 16V
+3.3VCC +3.3VDC
GND
GENERAL_FAULT GENERAL_FAULT
U3 USB-B
USB_DP 3 DP
USB_DM 2 DM
USB_VCC 1 VCC
4 GND
5 4 3 2 1
D C B A

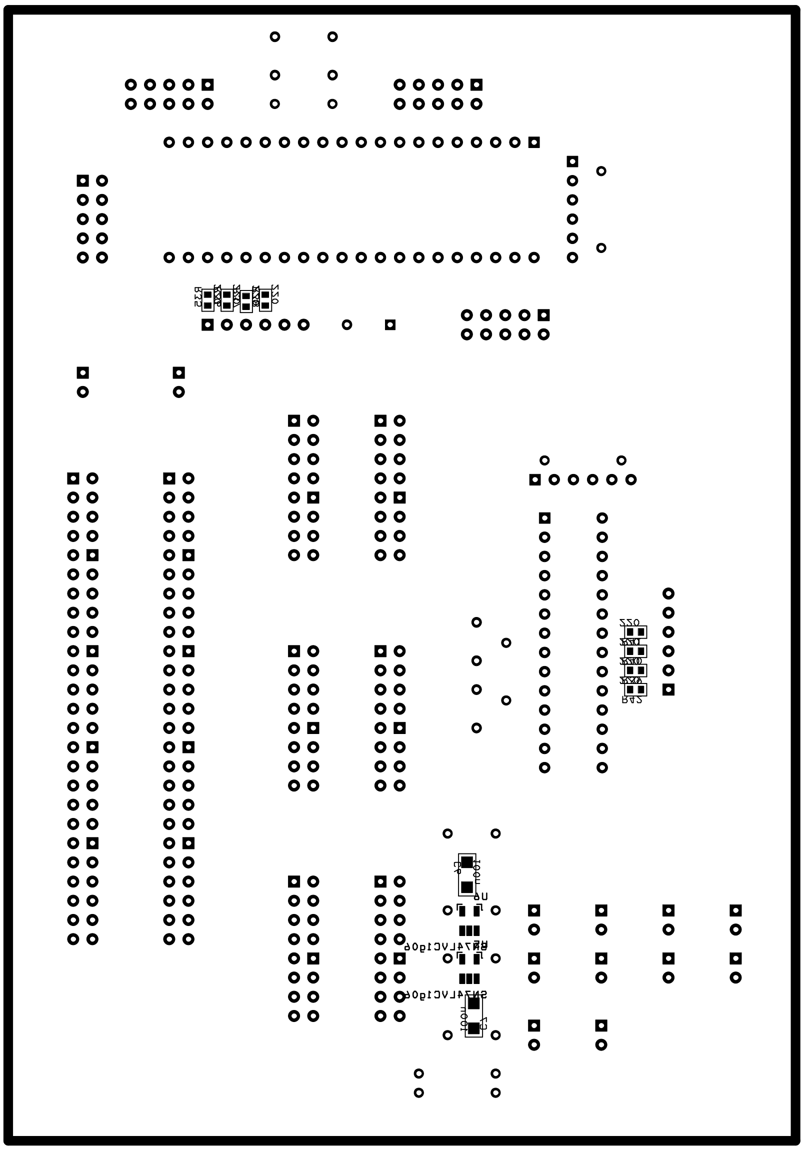
Title		
Contactor control		
Size A4	Document Number Relay interface board	Rev 1
Date: Friday, May 15, 2009	Sheet 2 of 2	

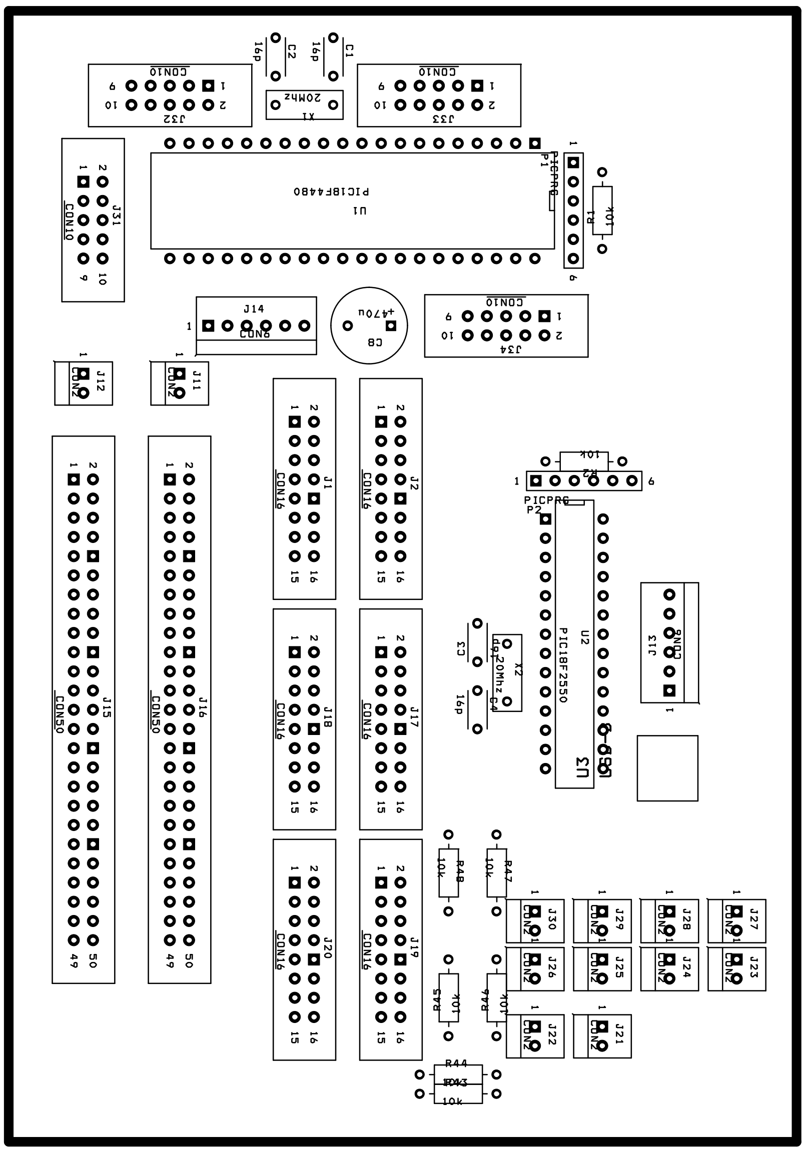


Title
Analog comparators CPLD

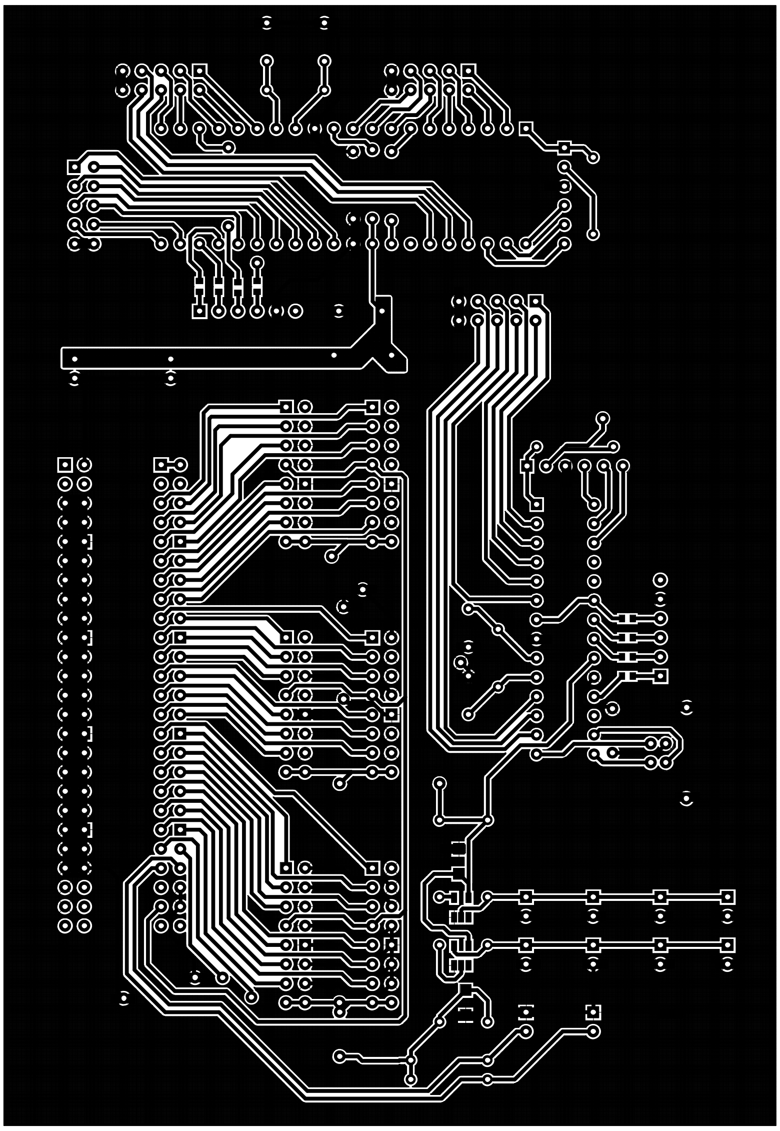
Size	Document Number	Rev
A4	Relay interface board	1

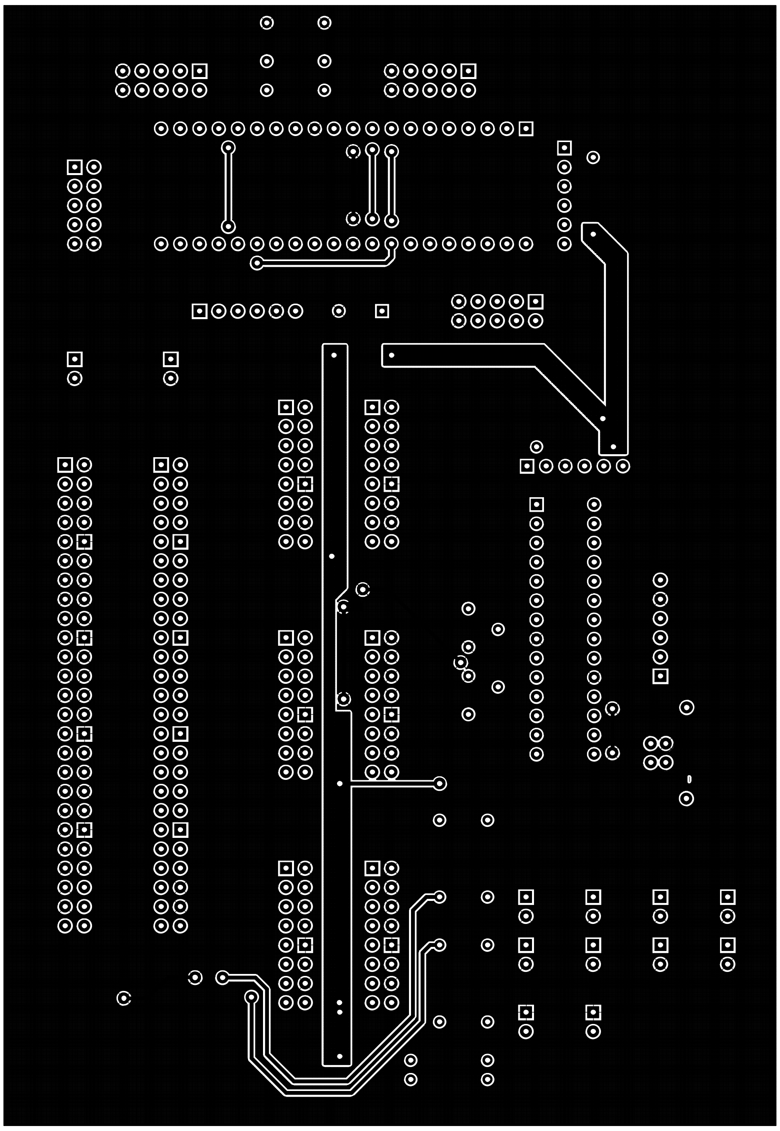
Date: Friday, May 15, 2009 Sheet 1 of 2

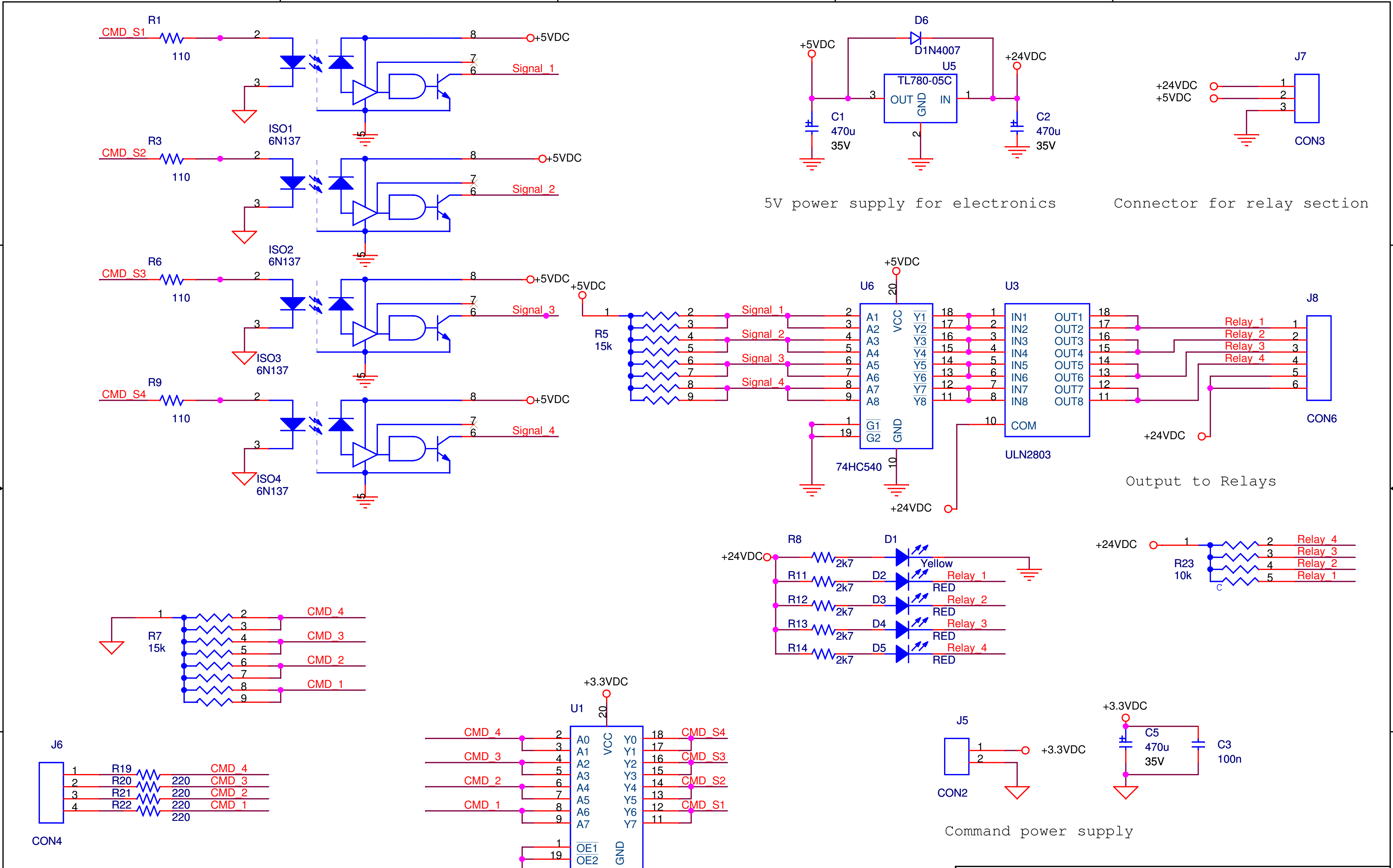




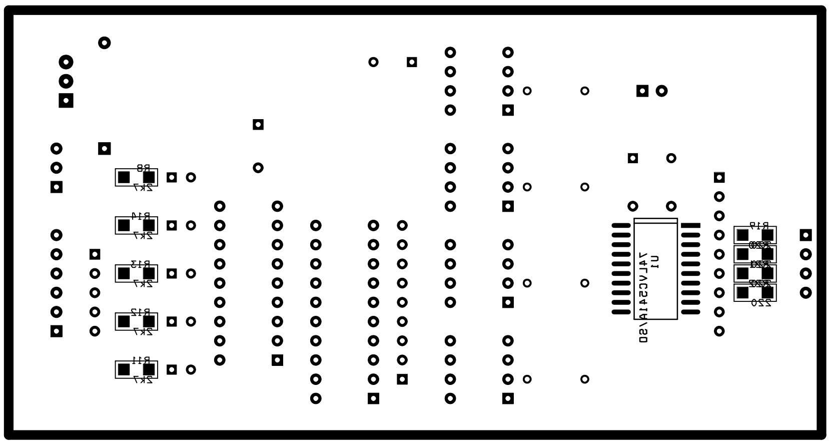
PIC18F4480
U1
PIC18F2550
U2
U3
J31
J32
J33
J34
J14
J12
J11
J15
J16
J1
J2
J18
J17
J20
J19
J13
J21
J22
J23
J24
J25
J26
J27
J28
J29
J30
CON10
CON16
CON50
CON6
CON2
PICPRG
P1
P2
R1
R2
R44
R45
R46
R47
R48
10k
C1
C2
C3
C4
C8
16p
+470u
X1
X2
20Mhz

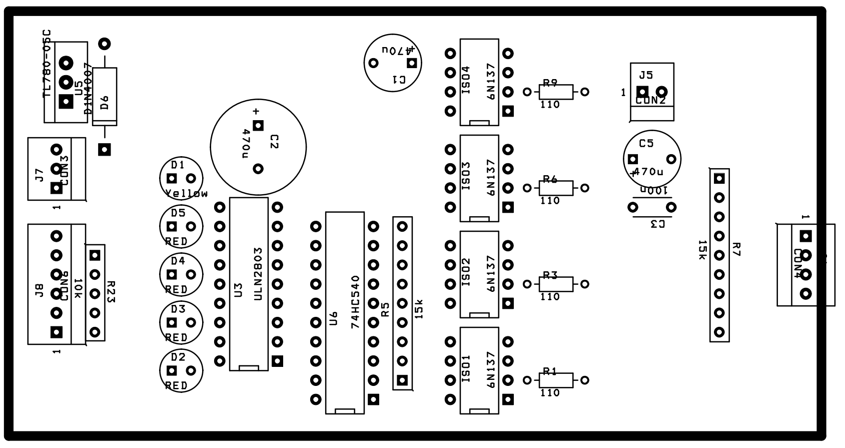




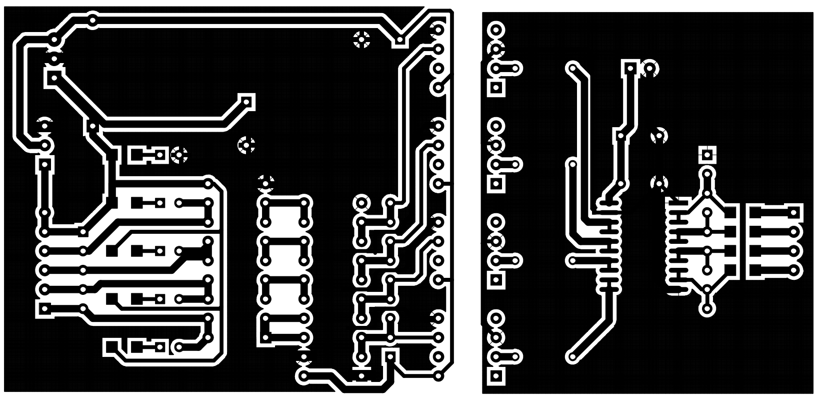


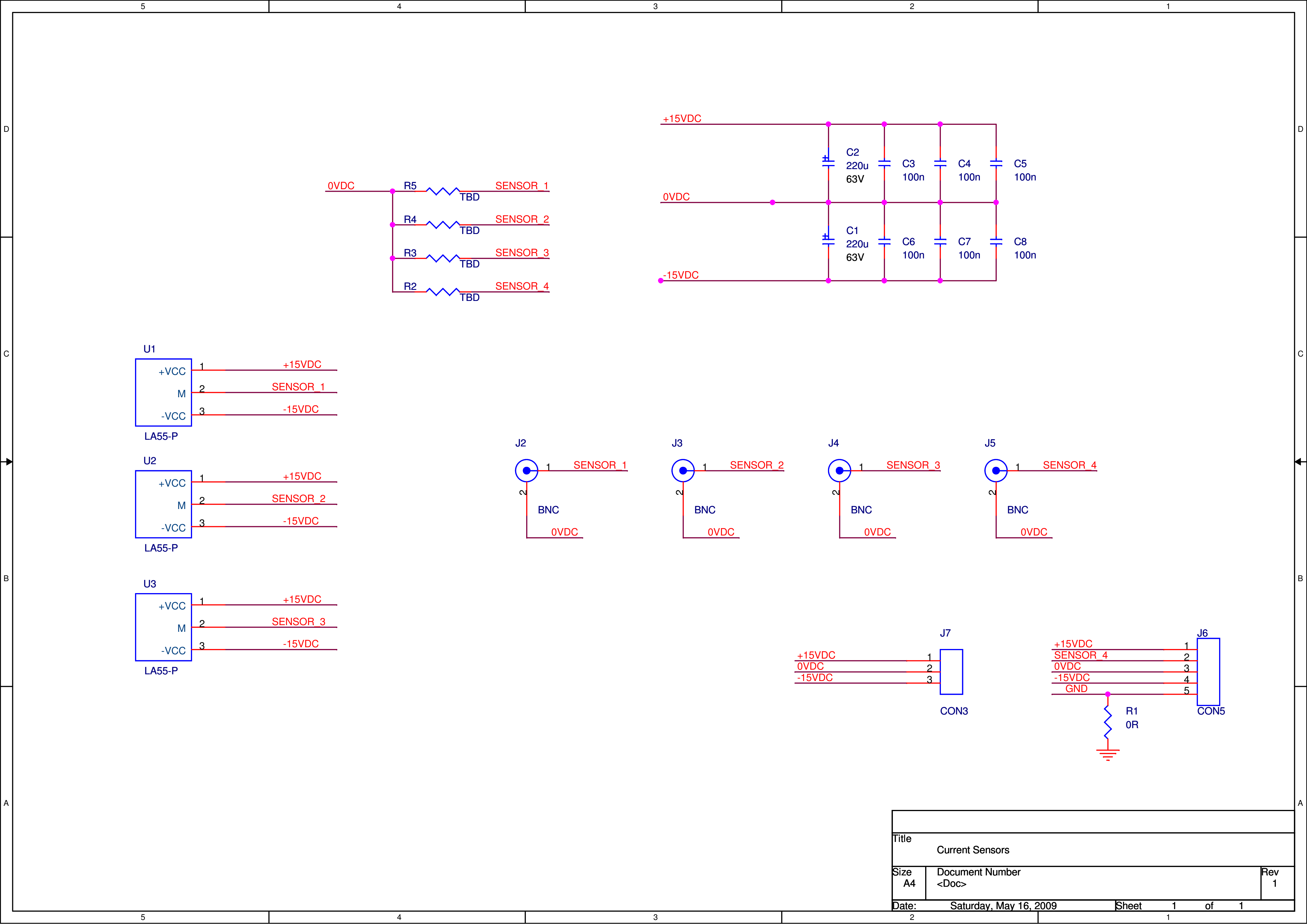
Title	Contactor output	
Size A4	Document Number Contactor Output	Rev C
Date: Friday, May 01, 2009	Sheet 1 of 1	


R8
2k7
R14
2k7
R13
2k7
R12
2k7
R11
2k7
U1
74LVC541A/SO
220

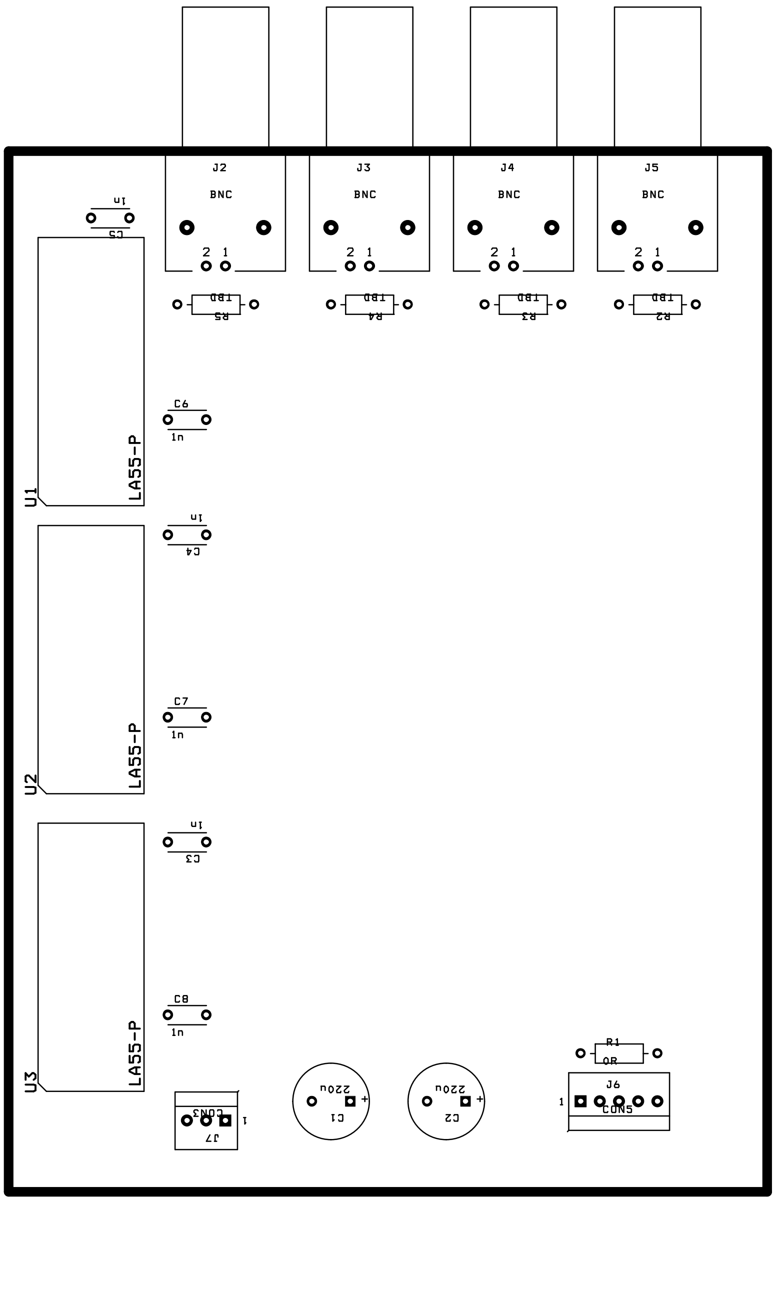


TL780-05C
U5
D1N4007
D6
C1
470u
C2
470u
ISO4
6N137
ISO3
6N137
ISO2
6N137
ISO1
6N137
R9
110
R6
110
R3
110
R1
110
J5
CON2
1
C5
470u
100n
C3
R7
15k
CON4
1
J7
CON3
1
J8
CON6
1
10k
R23
D1
Yellow
D5
RED
D4
RED
D3
RED
D2
RED
U3
ULN2803
U6
74HC540
R5
15k

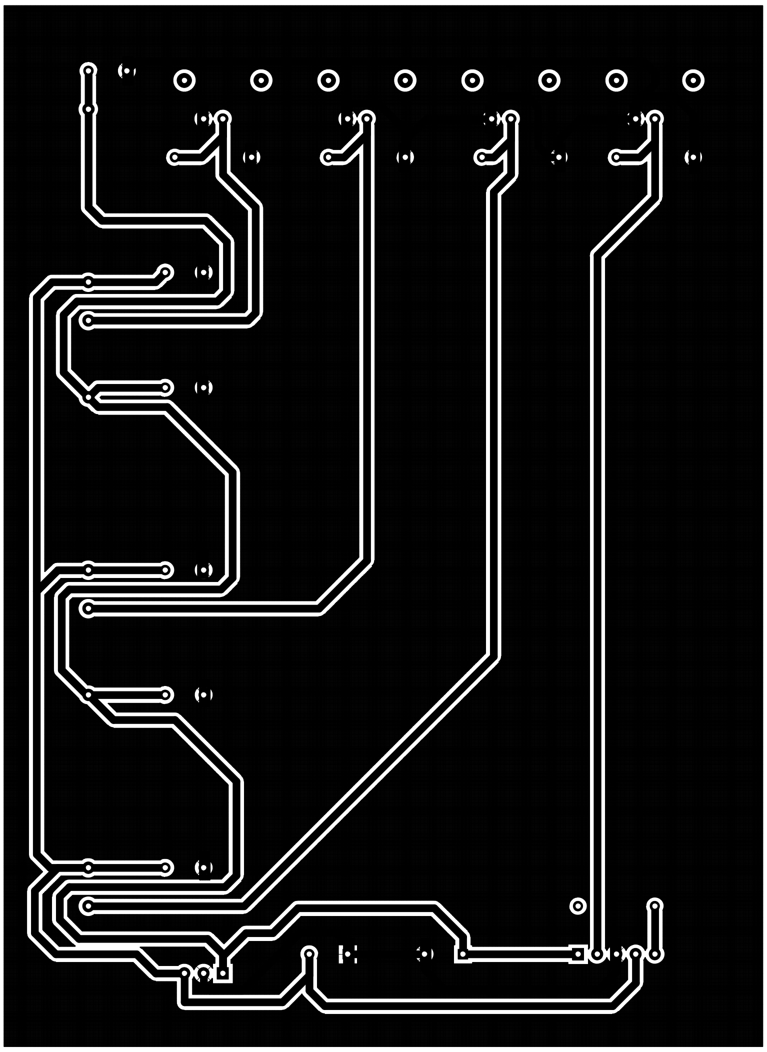


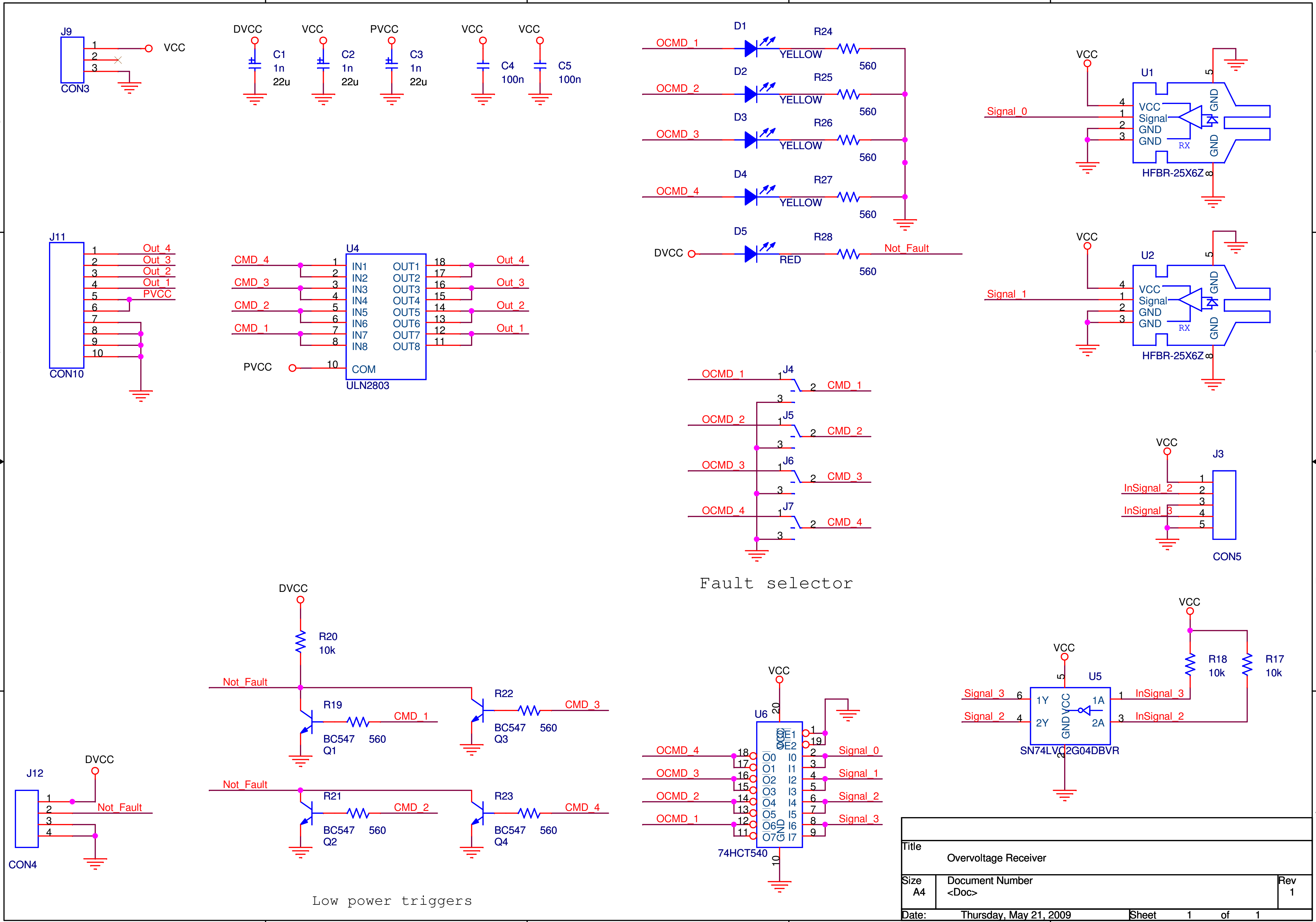


Title	Current Sensors	
Size A4	Document Number <Doc>	Rev 1
Date: Saturday, May 16, 2009	Sheet 1 of 1	

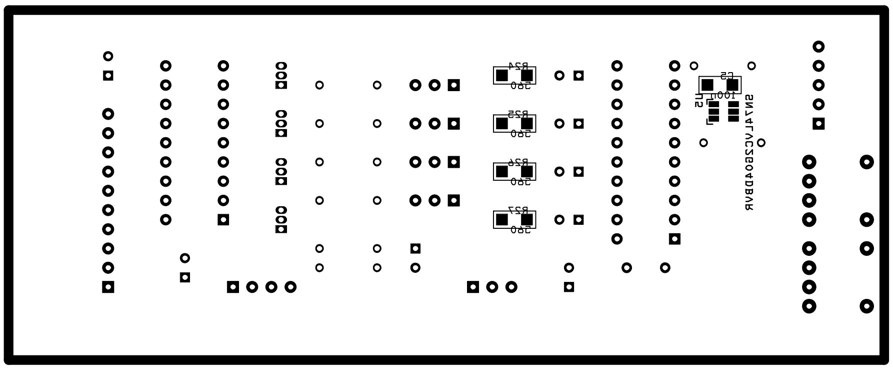


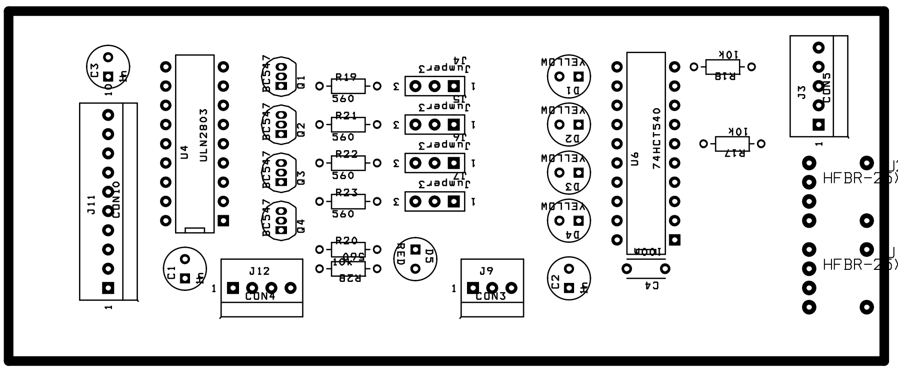
J2
BNC
J3
BNC
J4
BNC
J5
BNC
2 1
C5
1n
R5
R4
R3
R2
TBD
C6
1n
C4
1n
C7
1n
C3
1n
C8
1n
U1
LA55-P
U2
LA55-P
U3
LA55-P
J7
CON3
C1
220u
C2
220u
R1
0R
J6
CON5
1

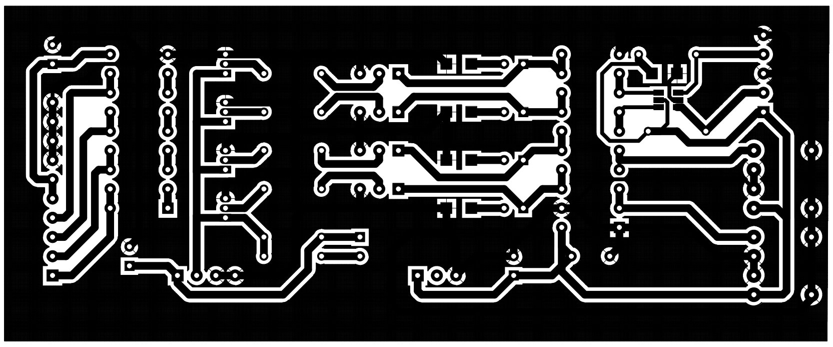


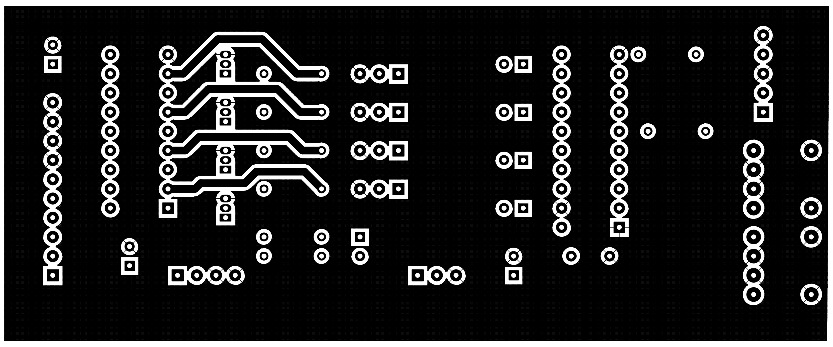


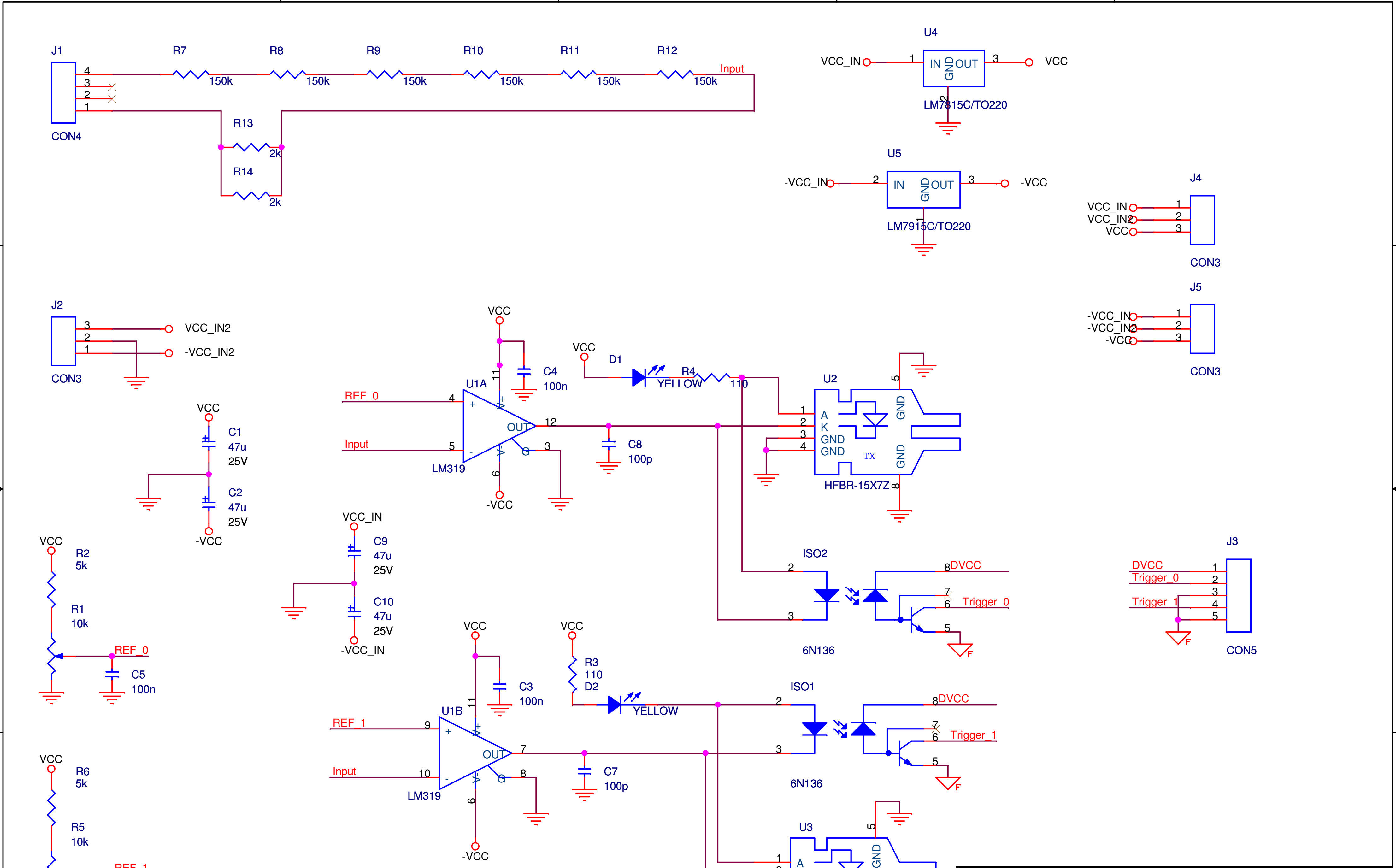
Title	Overvoltage Receiver	
Size A4	Document Number <Doc>	Rev 1
Date:	Thursday, May 21, 2009	Sheet 1 of 1



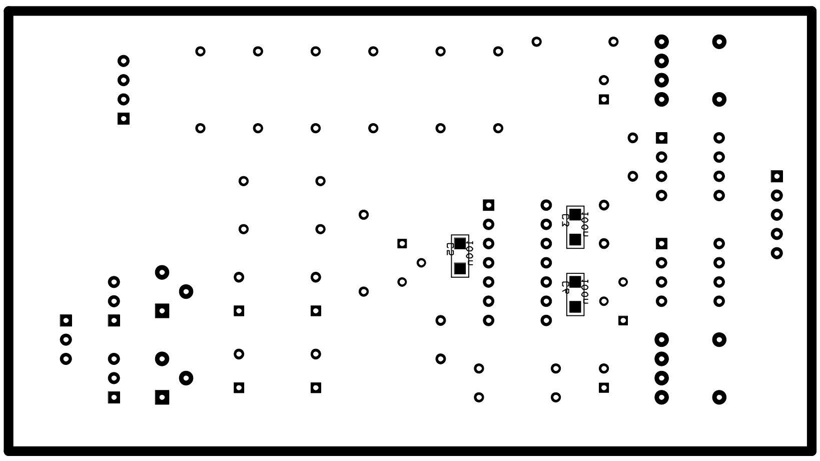

J11
CON10
C3
10u
C1
U4
ULN2803
BC547
Q1
Q2
Q3
Q4
R19
R21
R22
R23
R20
R28
560
10K
J4
J5
J6
J7
Jumper3
D5
RED
J12
CON4
J9
CON3
D1
D2
D3
D4
YELLOW
C2
U6
74HCT540
C4
100n
R18
R17
10K
J3
CON3
U2
U1
HFBR-25X6Z

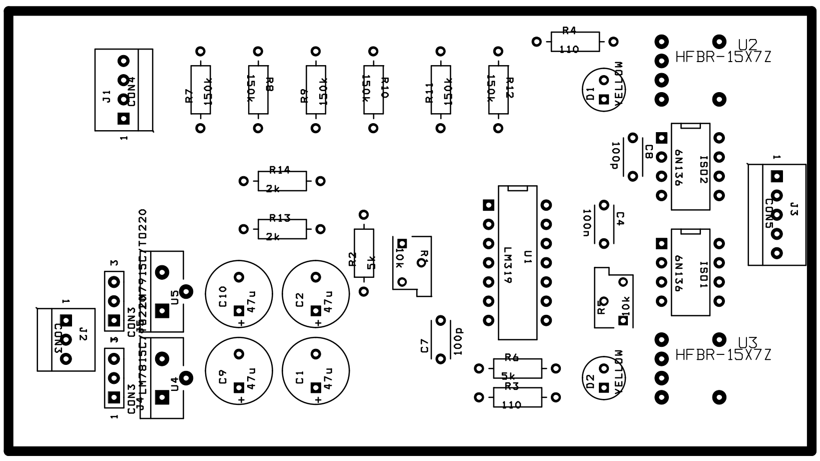


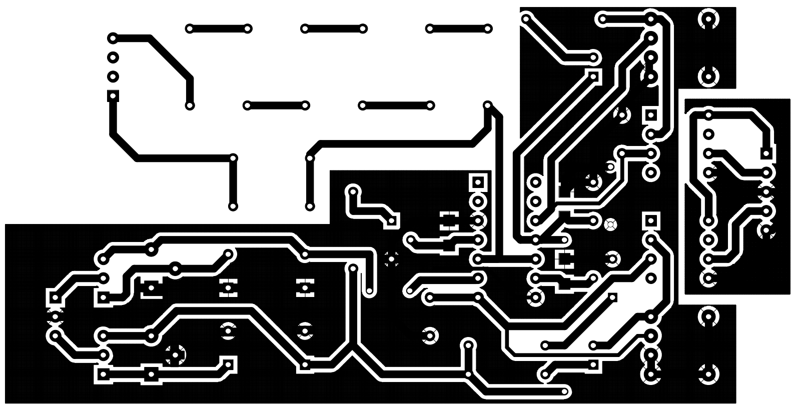


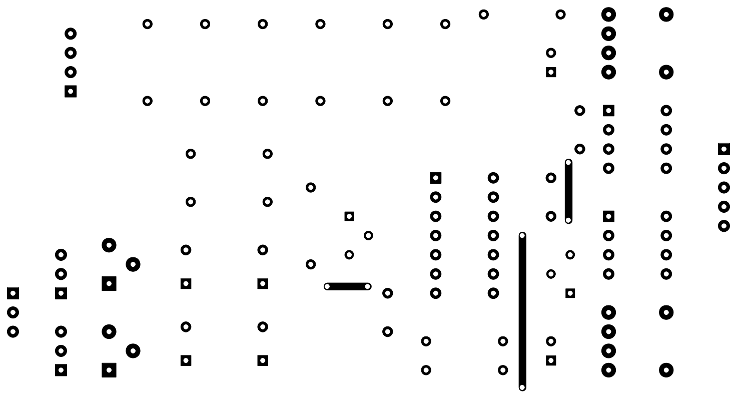


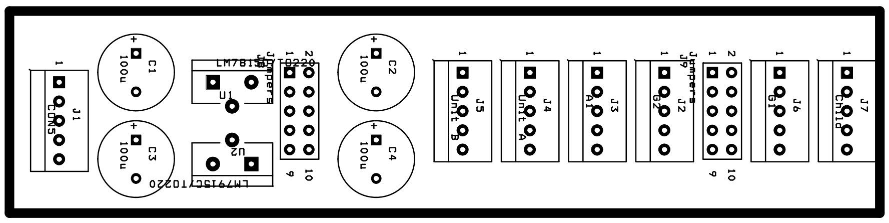
Title	Overvoltage Transmit	
Size A4	Document Number <Doc>	Rev 1
Date: Monday, May 18, 2009	Sheet 1 of 1	

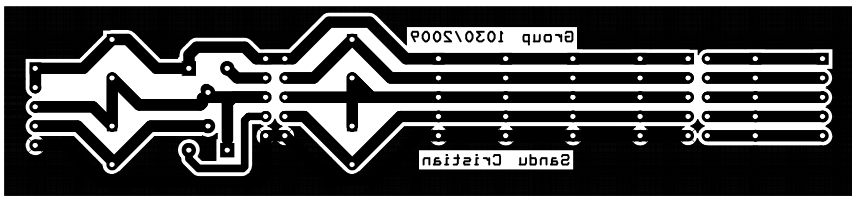

C2
100n
C3
100n
C4
100n

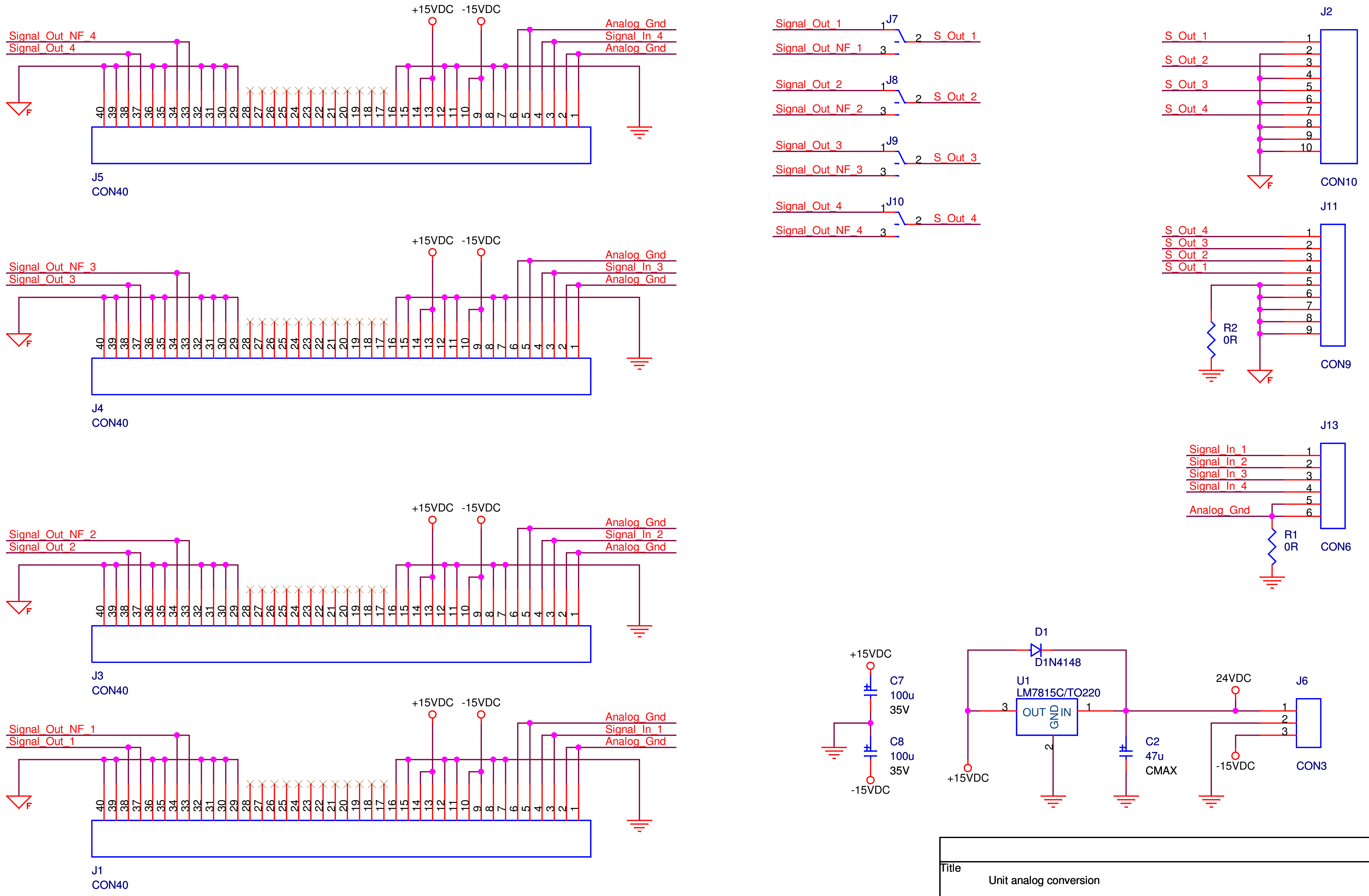

J1
CON4
R7
150k
R8
150k
R9
150k
R10
150k
R11
150k
R12
150k
R4
110
U2
HFBR-15X7Z
D1
YELLOW
C8
100p
6N136
ISO2
J3
CON5
R14
2k
R13
2k
R2
5k
R1
10k
U1
LM319
C4
100n
6N136
ISO1
R5
10k
LM7915C/TO220
U5
C10
47u
C2
47u
J2
CON3
CON3
J5
J4
CON3
LM7815C/TO220
U4
C9
47u
C1
47u
C7
100p
R6
5k
R3
110
D2
YELLOW
U3
HFBR-15X7Z





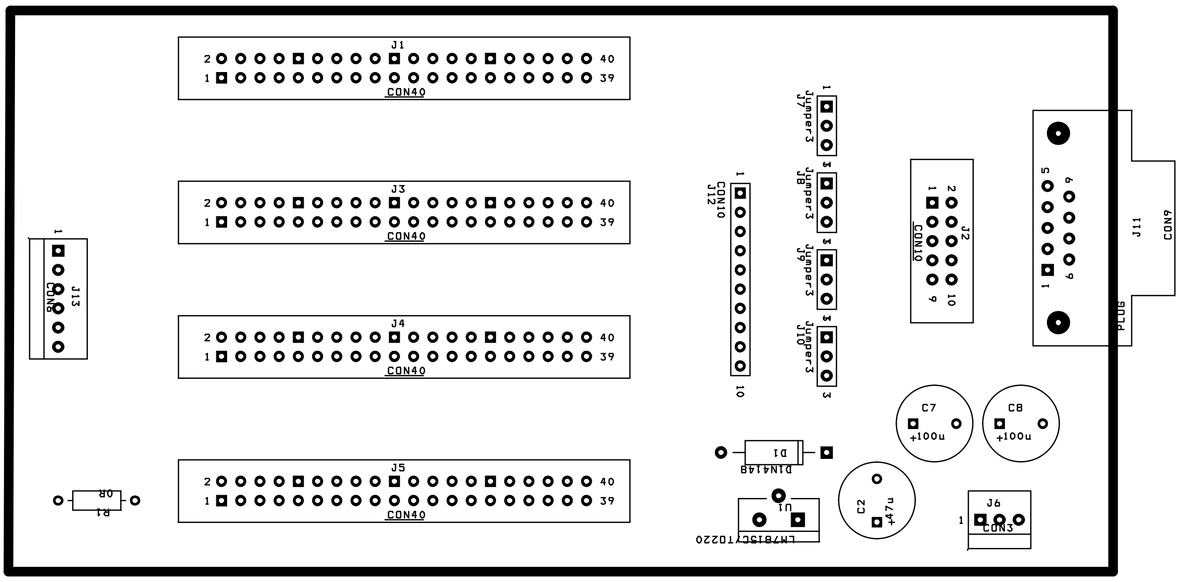

J1
CON5
C1
C2
C3
C4
100u
LM7815C/TO220
LM7915C/TO220
U1
U2
J8
Jumpers
J9
J5
Unit B
J4
Unit A
J3
A1
J2
G2
J6
G1
J7
Child

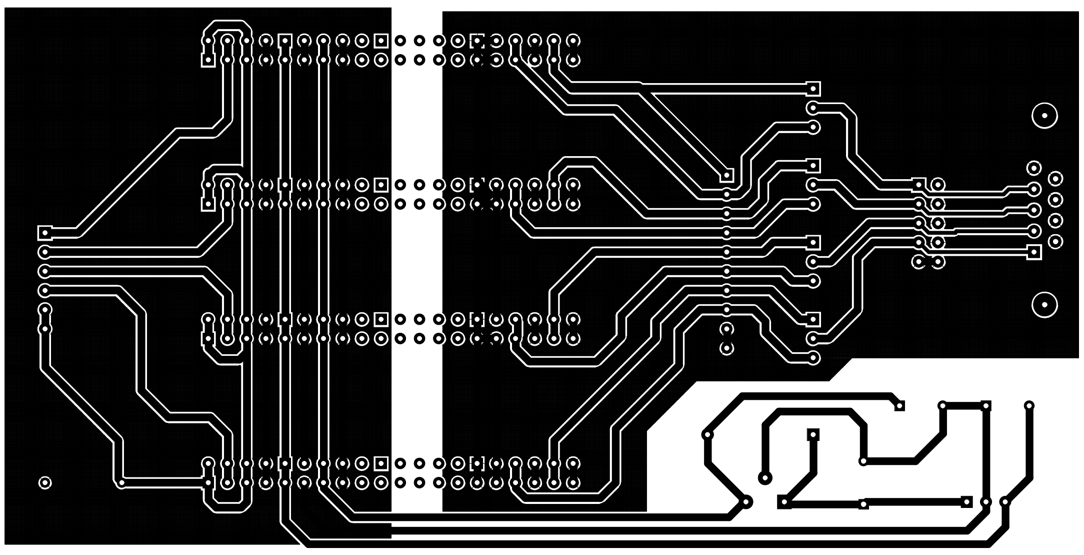

Group 1030/2009
Sandu Cristian

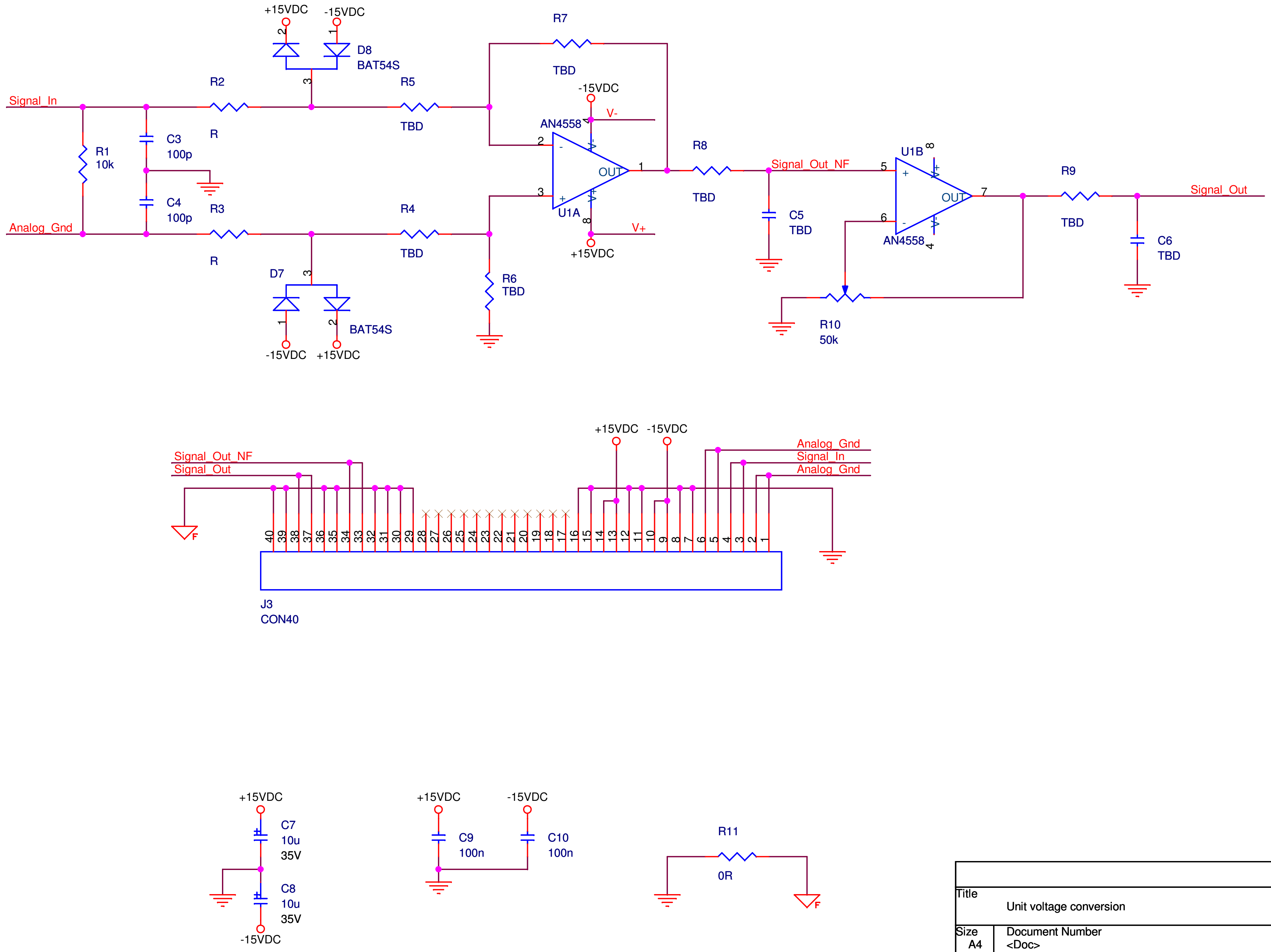


Title
Unit analog conversion

Size A4	Document Number <Doc>	Rev 1
Date: Saturday, May 02, 2009	Sheet 1 of 1	


J1
CON40
J3
CON40
J4
CON40
J5
CON40
J13
CON6
R1
0R
J7
J8
J9
J10
Jumper3
J12
CON10
D1
D1N4148
U1
LM7815C/TO220
C2
+47u
C7
+100u
C8
+100u
J2
CON10
J11
CON9
PLUG
J6
CON3

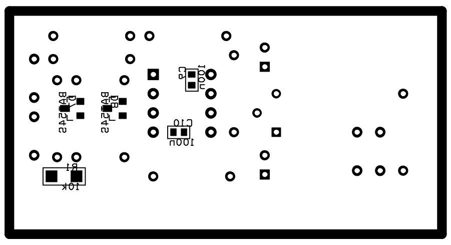


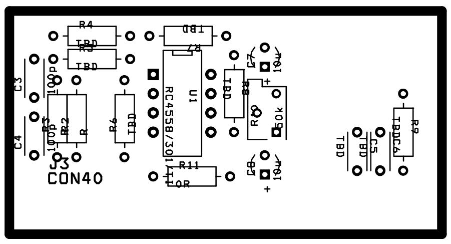


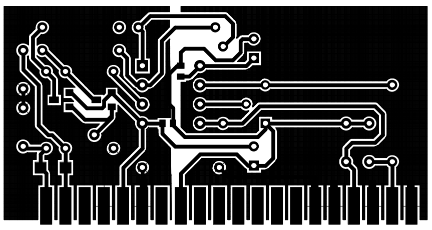
Title
Unit voltage conversion

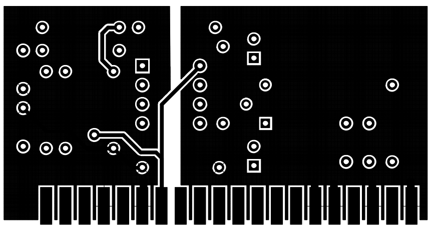
Size	Document Number	Rev
A4	<Doc>	1

Date: Friday, May 01, 2009 Sheet 1 of 1




R4
TBD
R3
TBD
TBD
R7
C3
100p
C4
100p
R5
R2
R
R6
TBD
U1
RC4558/301/
TBD
R8
C7
10u
+
R10
50k
R11
10R
C8
10u
+
J3
CON40
TBD
TBD
C5
TBD
C6
R9





APPENDIX B. MODULATIONS

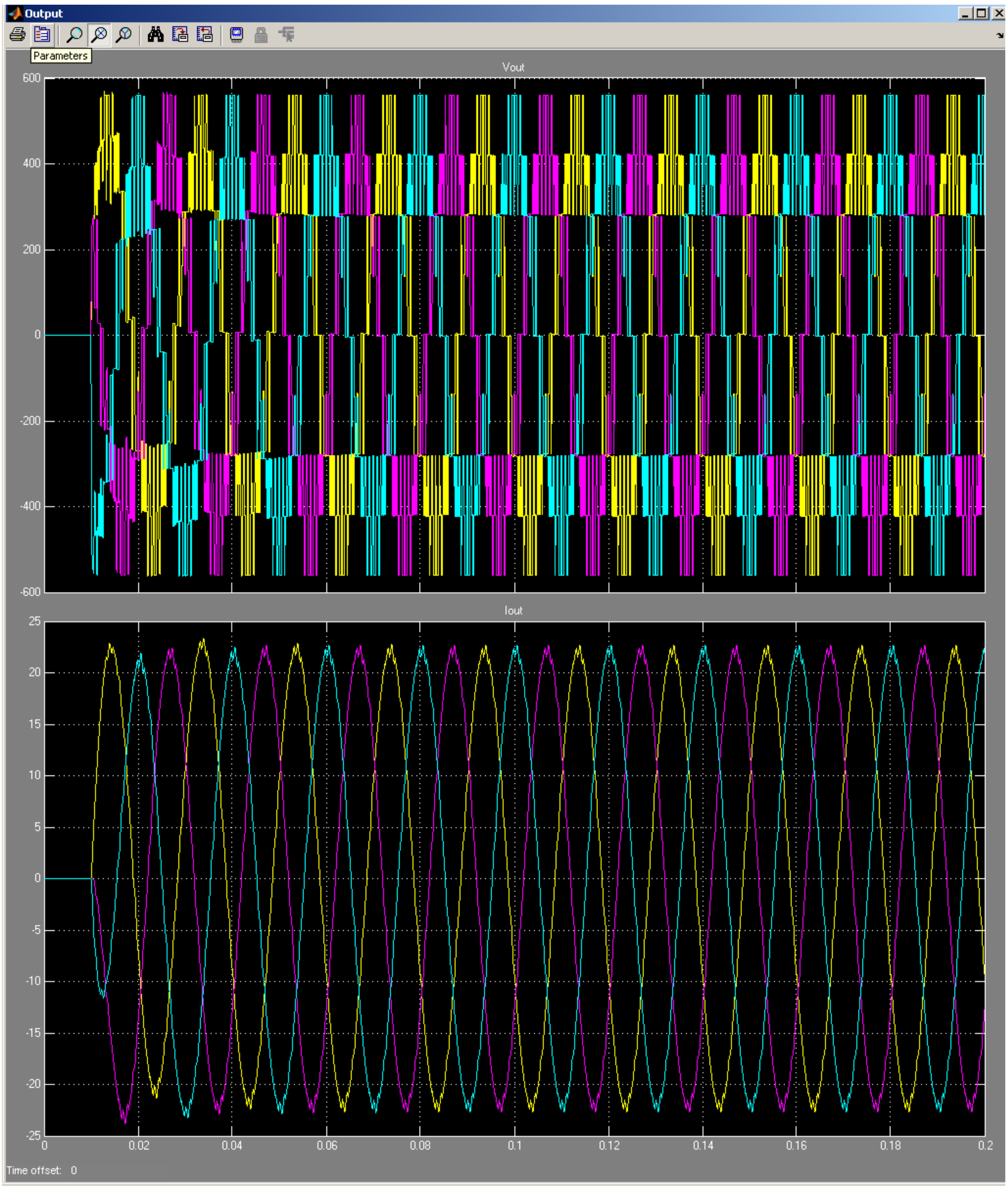


Figure 1 Output voltage and current for Phase Shifted Modulation at 600 Hz

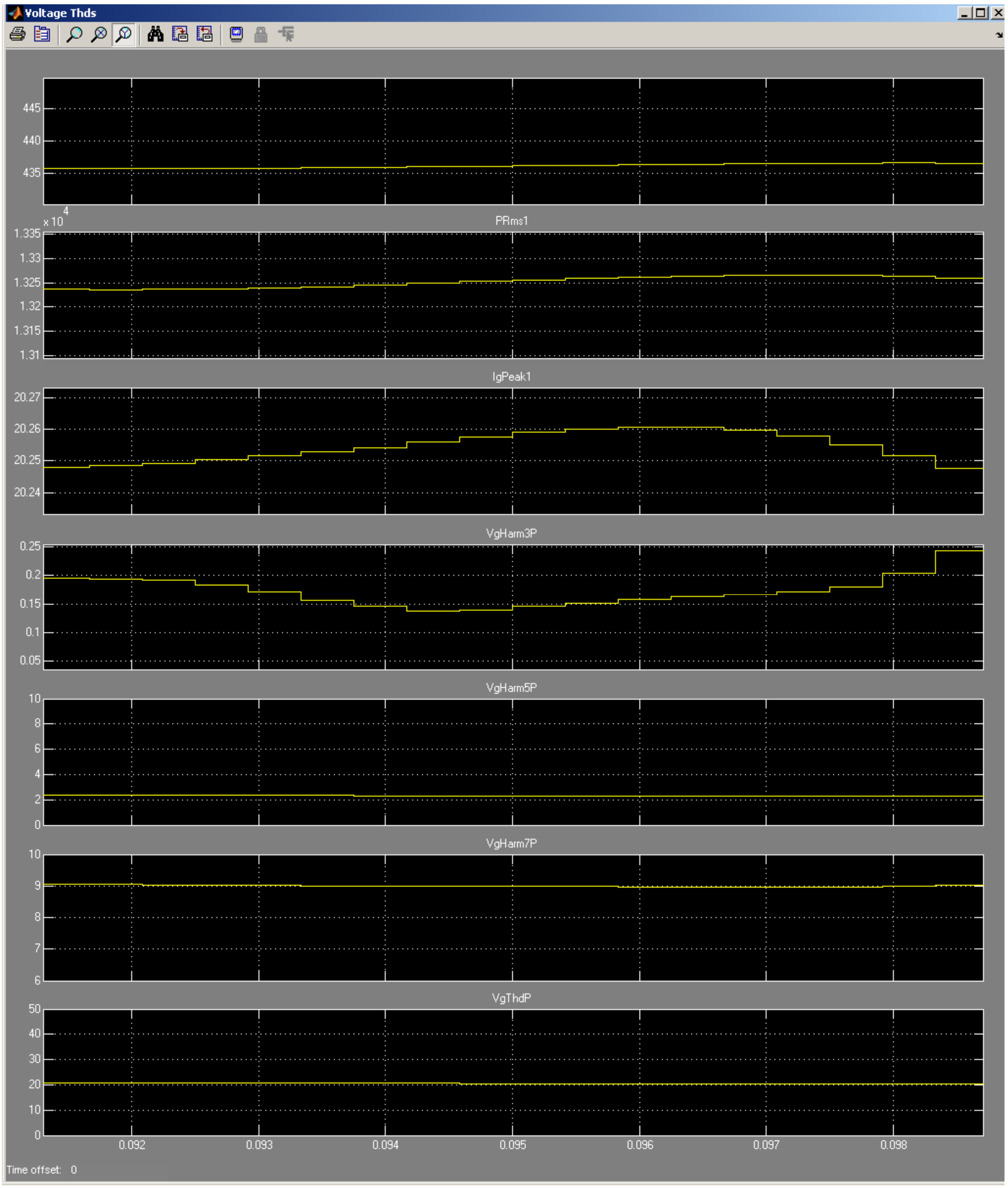


Figure 2 Voltage Harmonics for Phase Shifted Modulation at 600 Hz

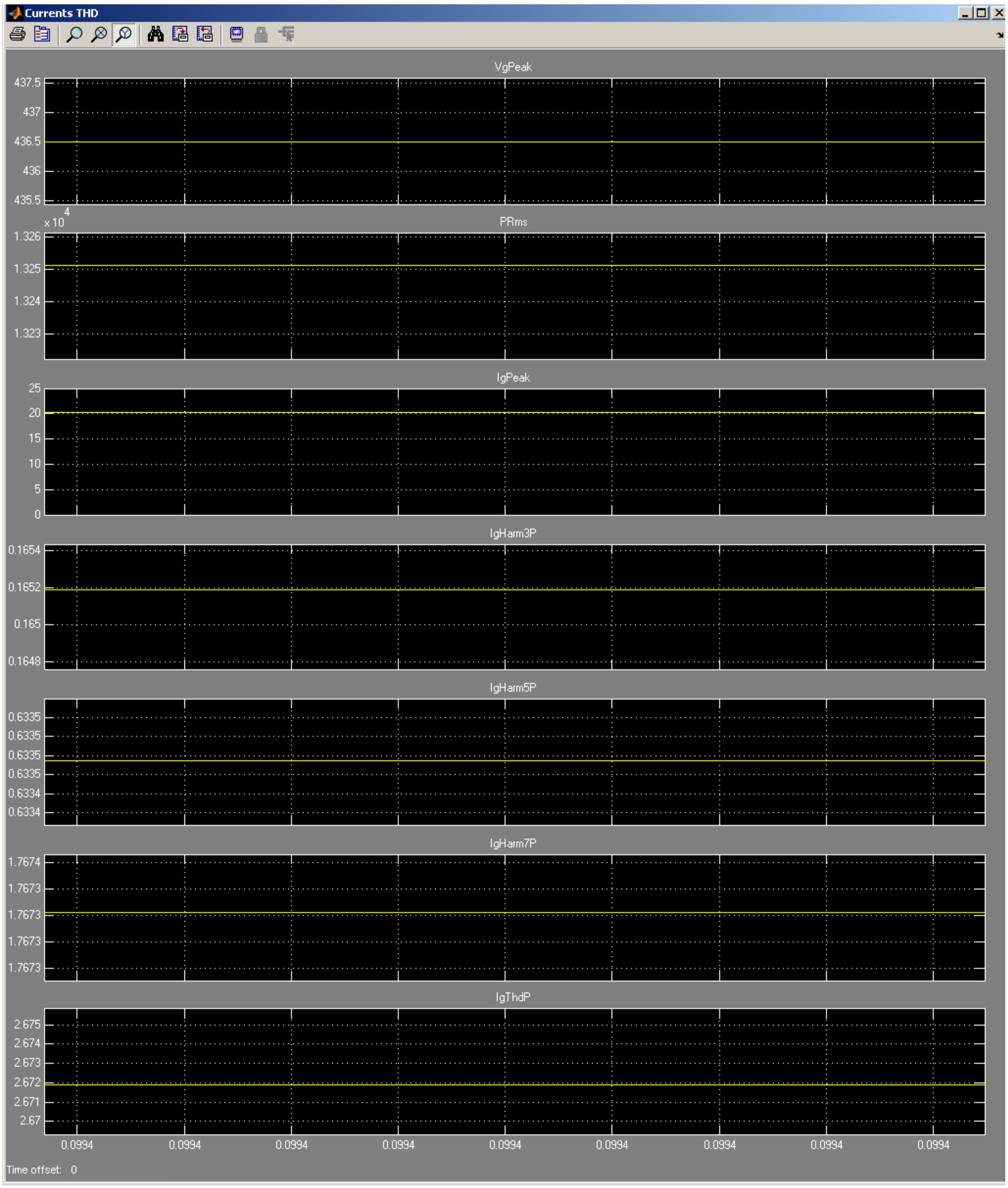


Figure 3 Current Harmonics for Phase Shifted Modulation at 600 Hz

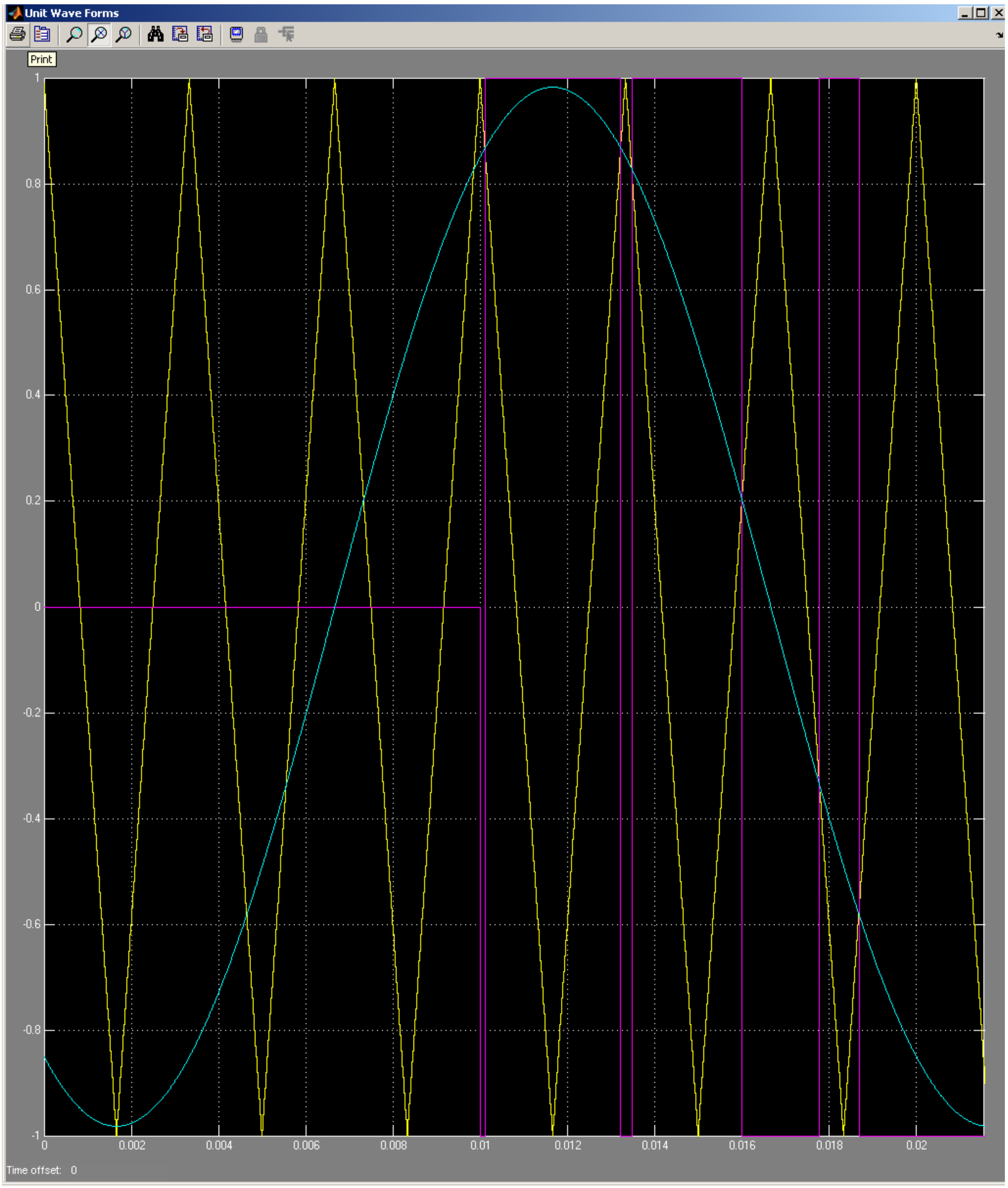


Figure 4 Carriers for Phase Shifted Modulation for 600 Hz

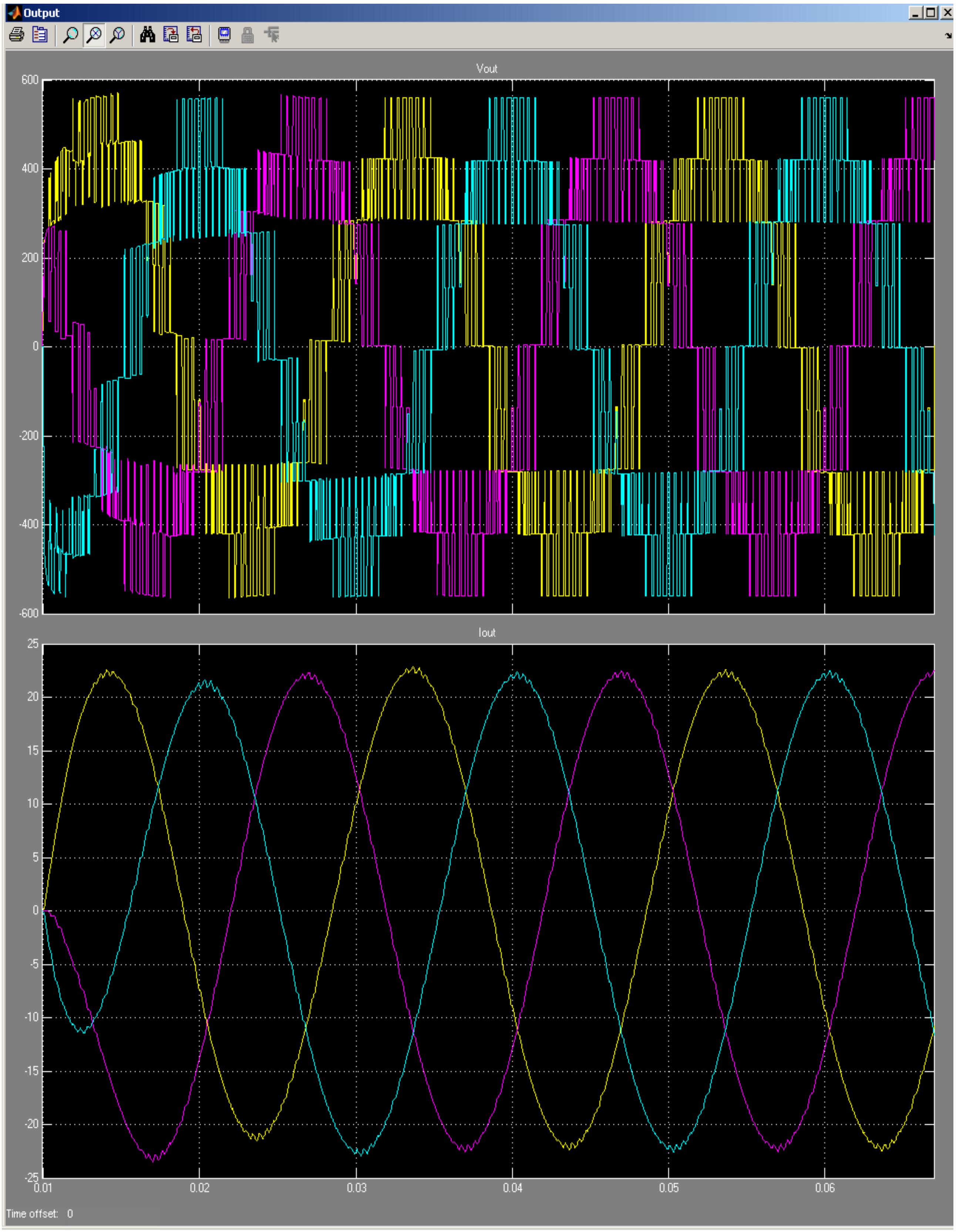


Figure 5 Output voltage and current for Phase Shifted Modulation at 1200 Hz

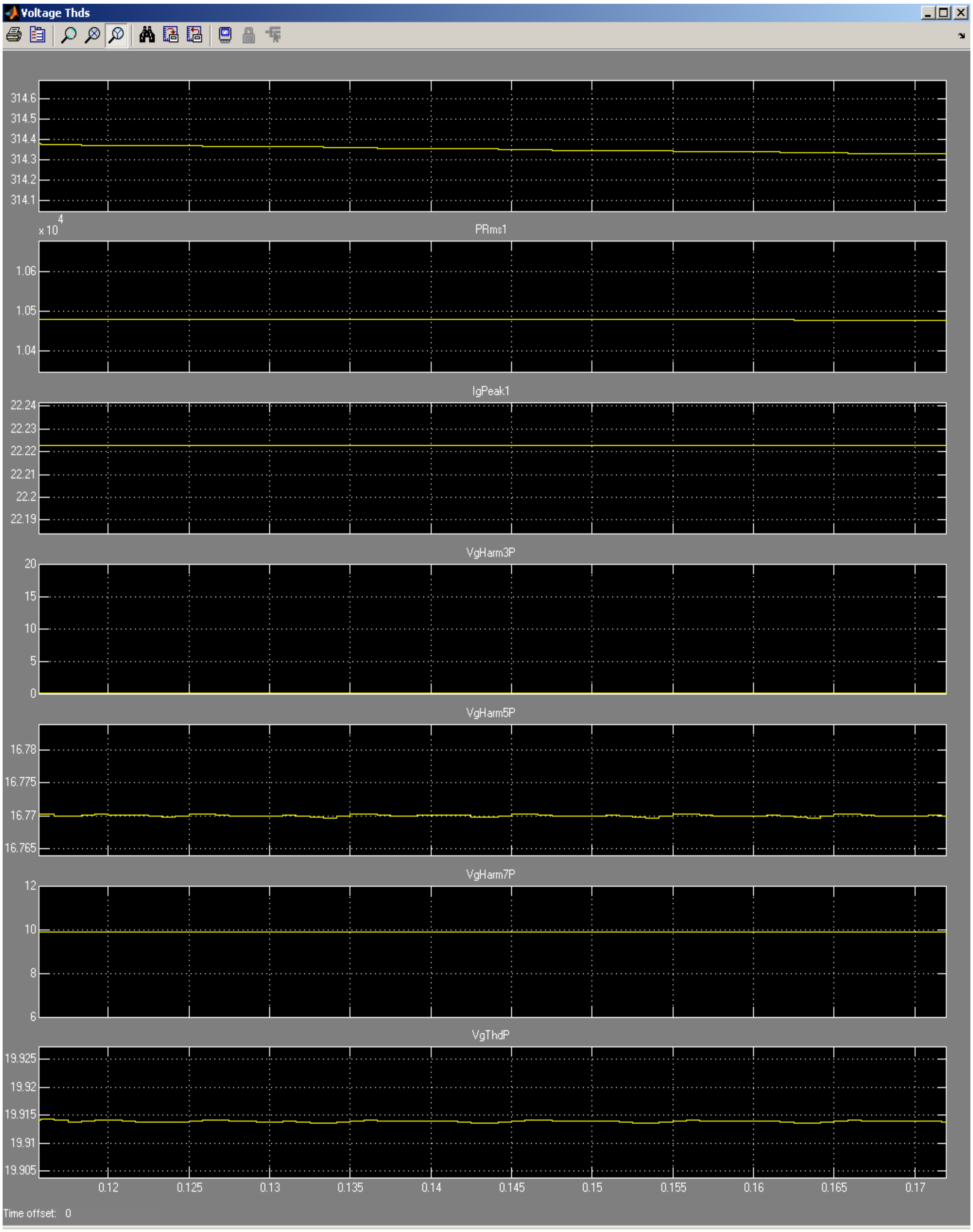


Figure 6 Voltage Harmonics for Phase Shifted Modulation at 1200 Hz

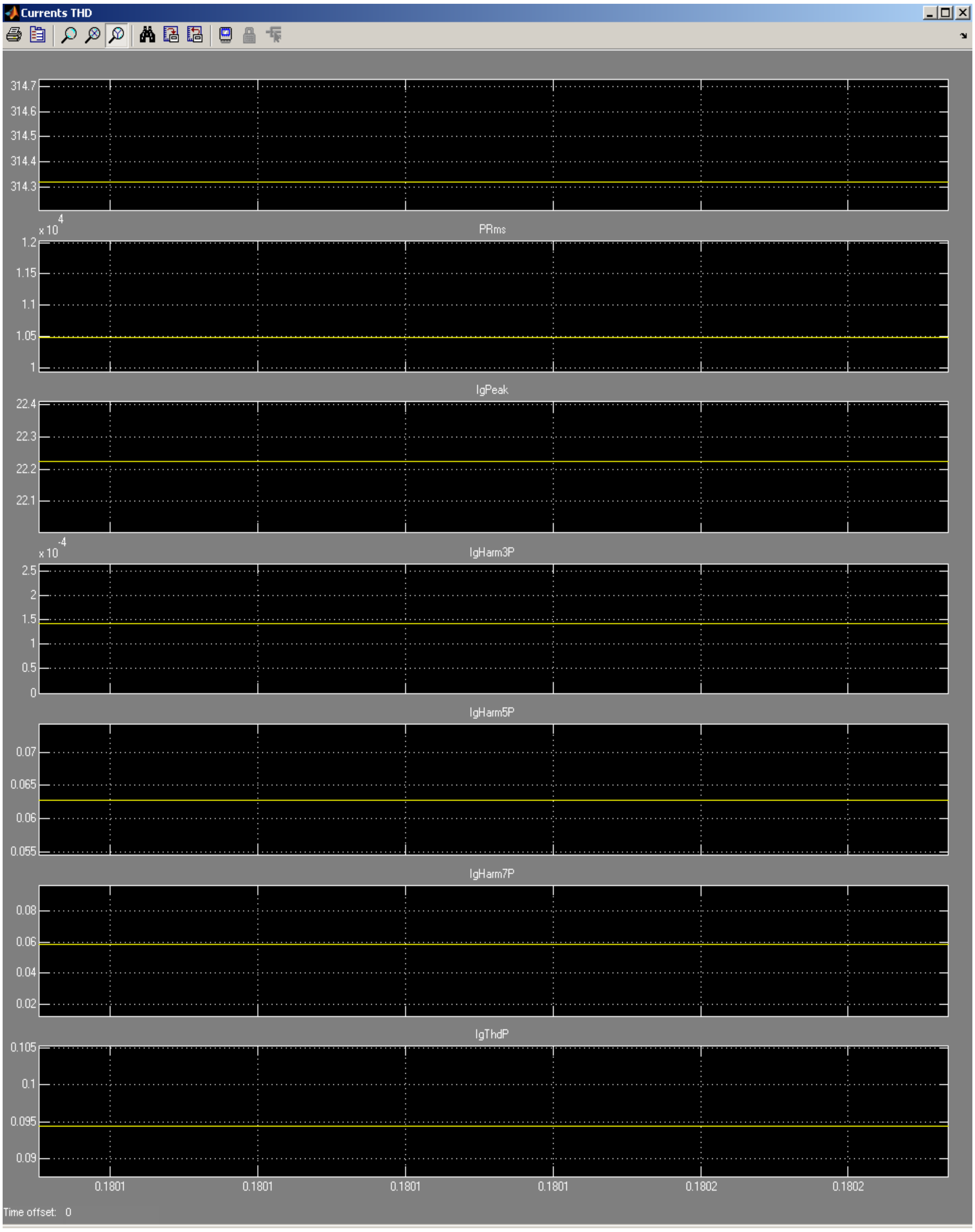


Figure 7 Current Harmonics for Phase Shifted Modulation at 1200 Hz

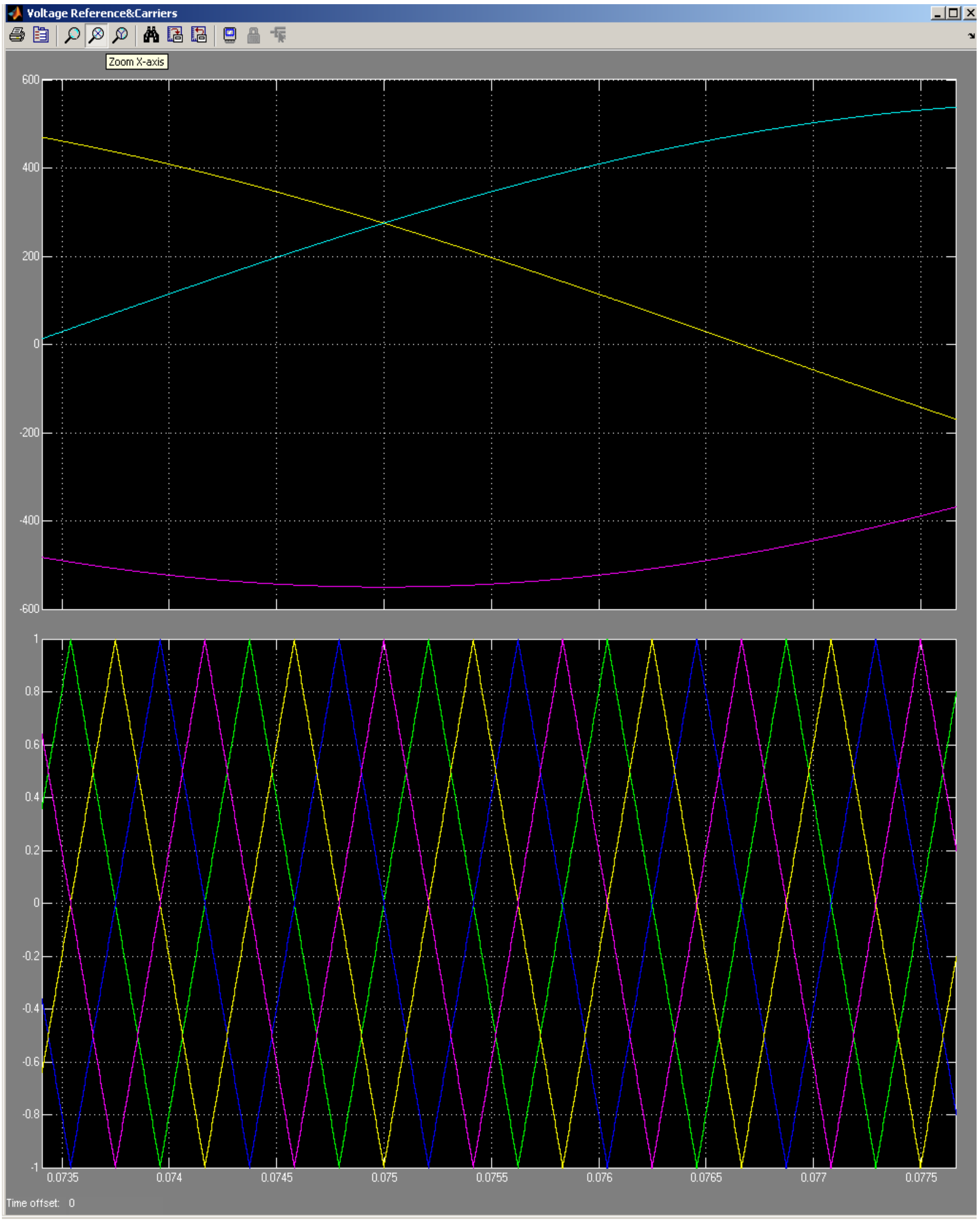


Figure 8 Carriers for Phase Shifted Modulation for 1200 Hz

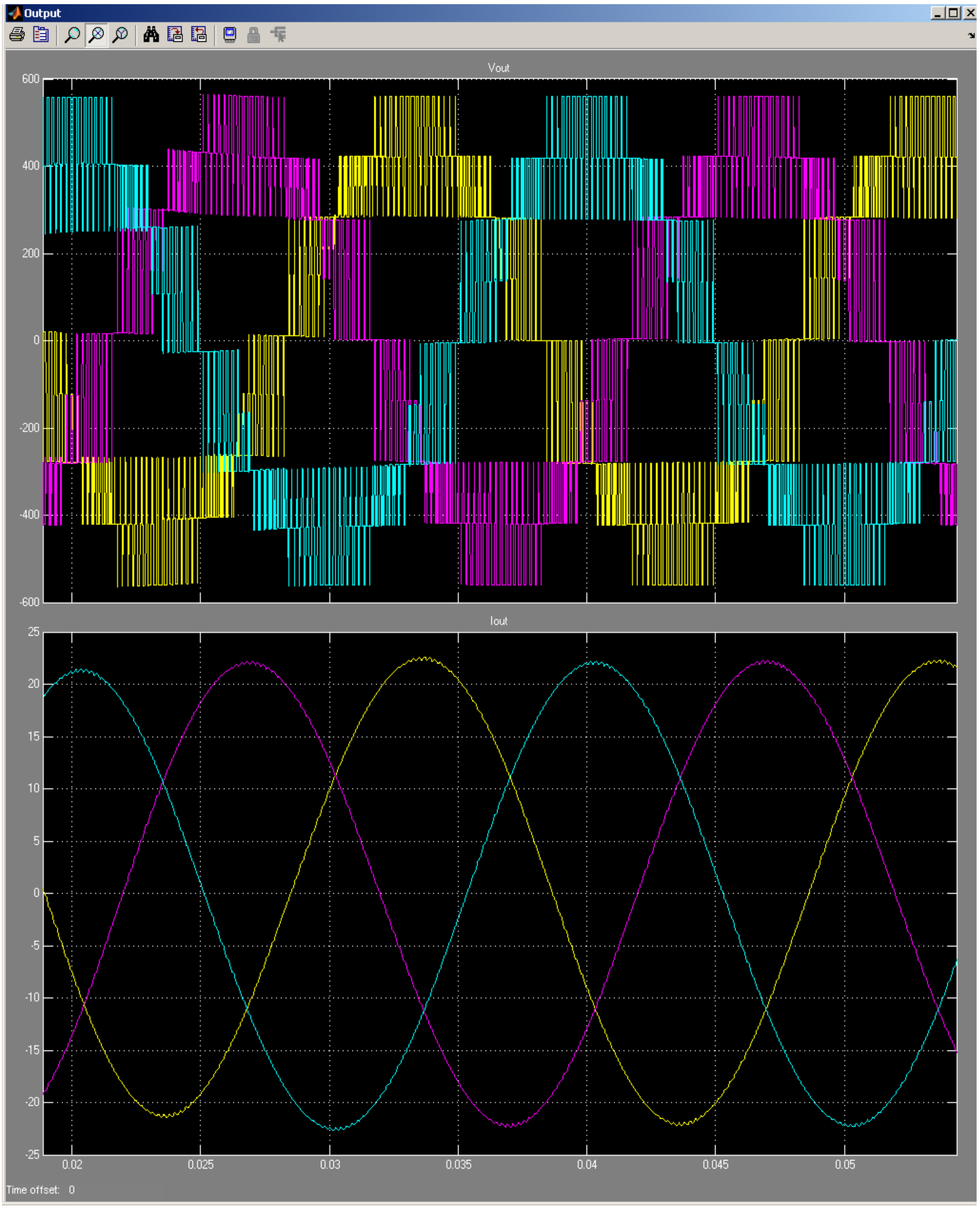


Figure 9 Output voltage and current for Phase Shifted Modulation at 2400 Hz

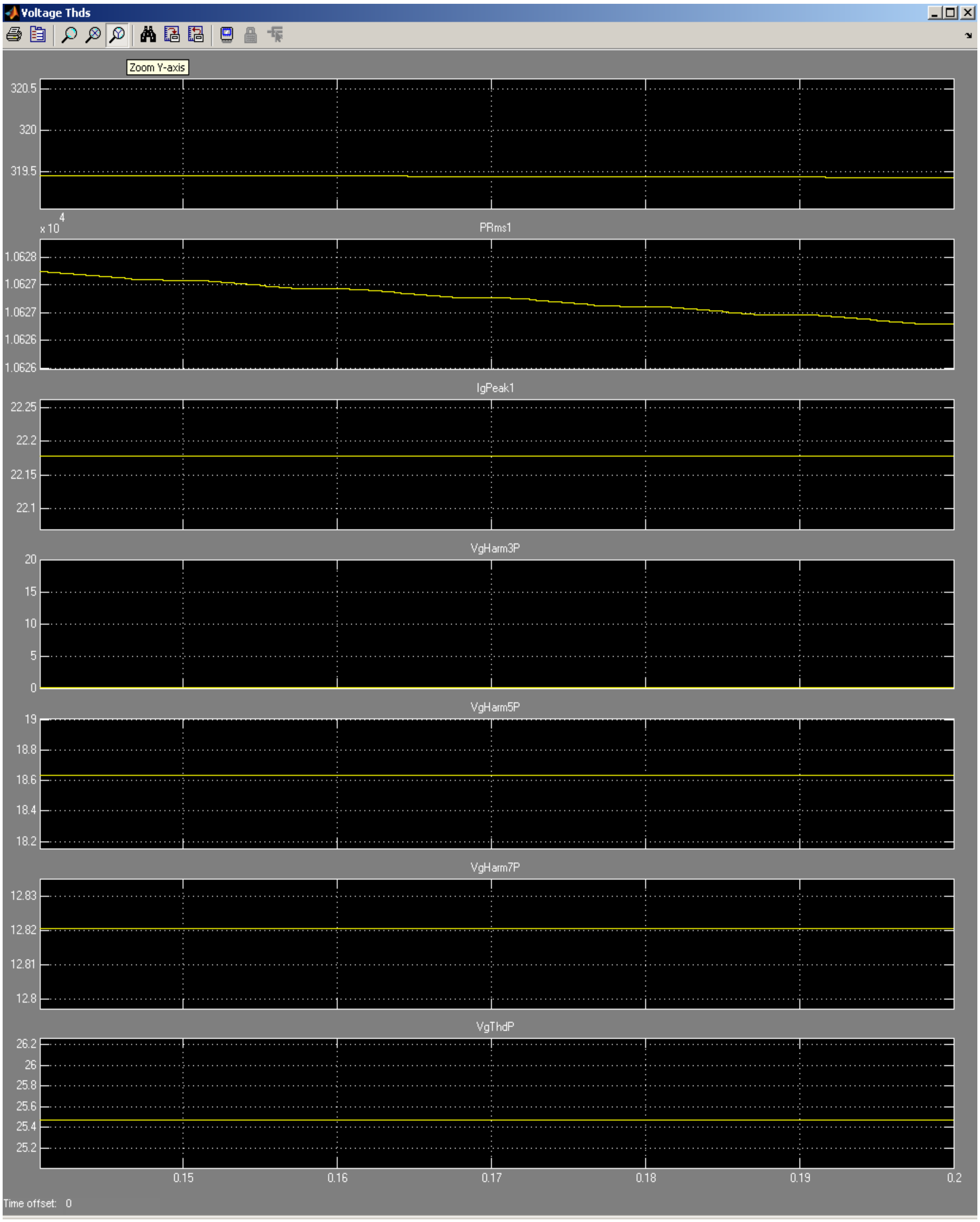


Figure 10 Voltage Harmonics for Phase Shifted Modulation at 2400 Hz

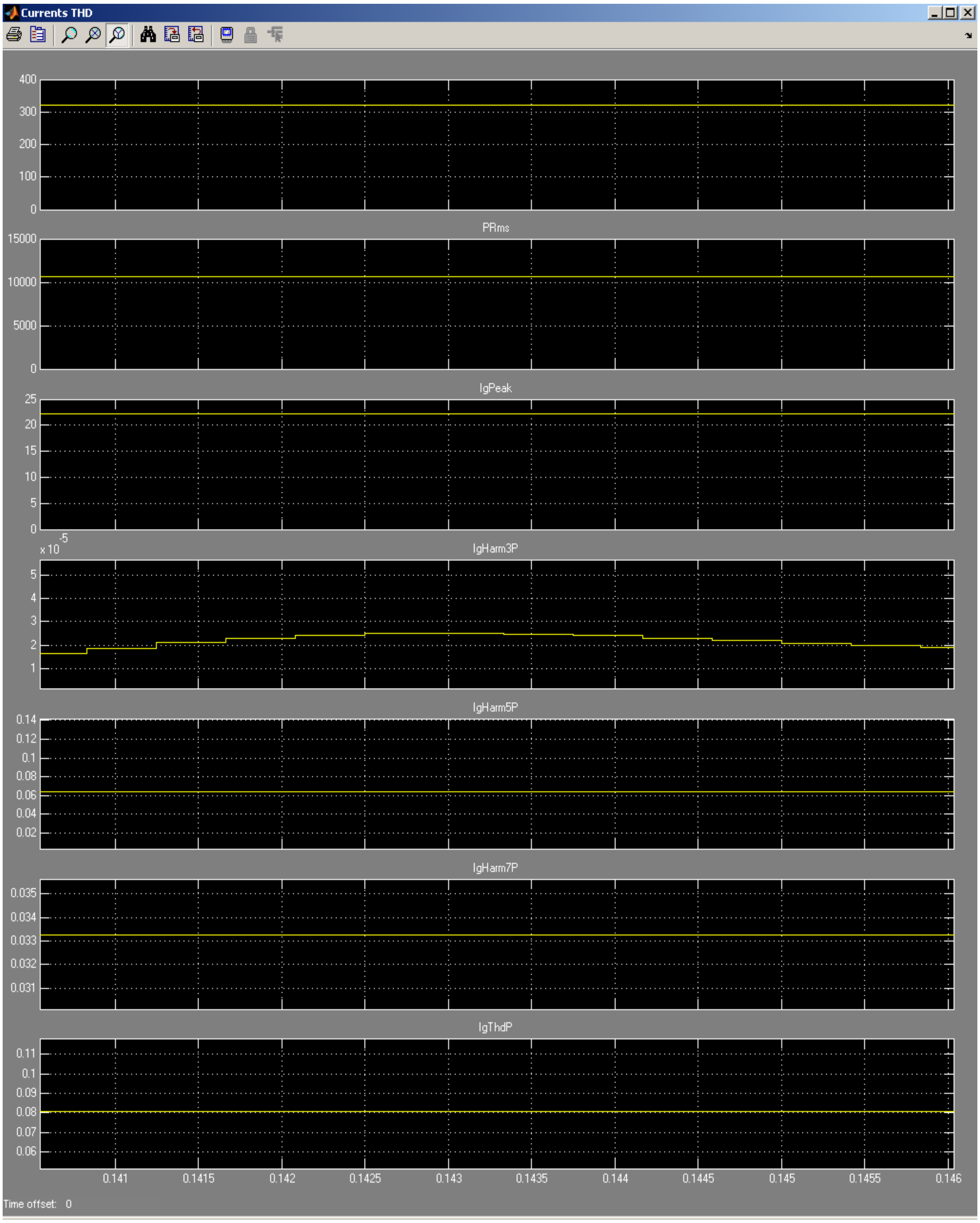


Figure 11 Current Harmonics for Phase Shifted Modulation at 2400 Hz

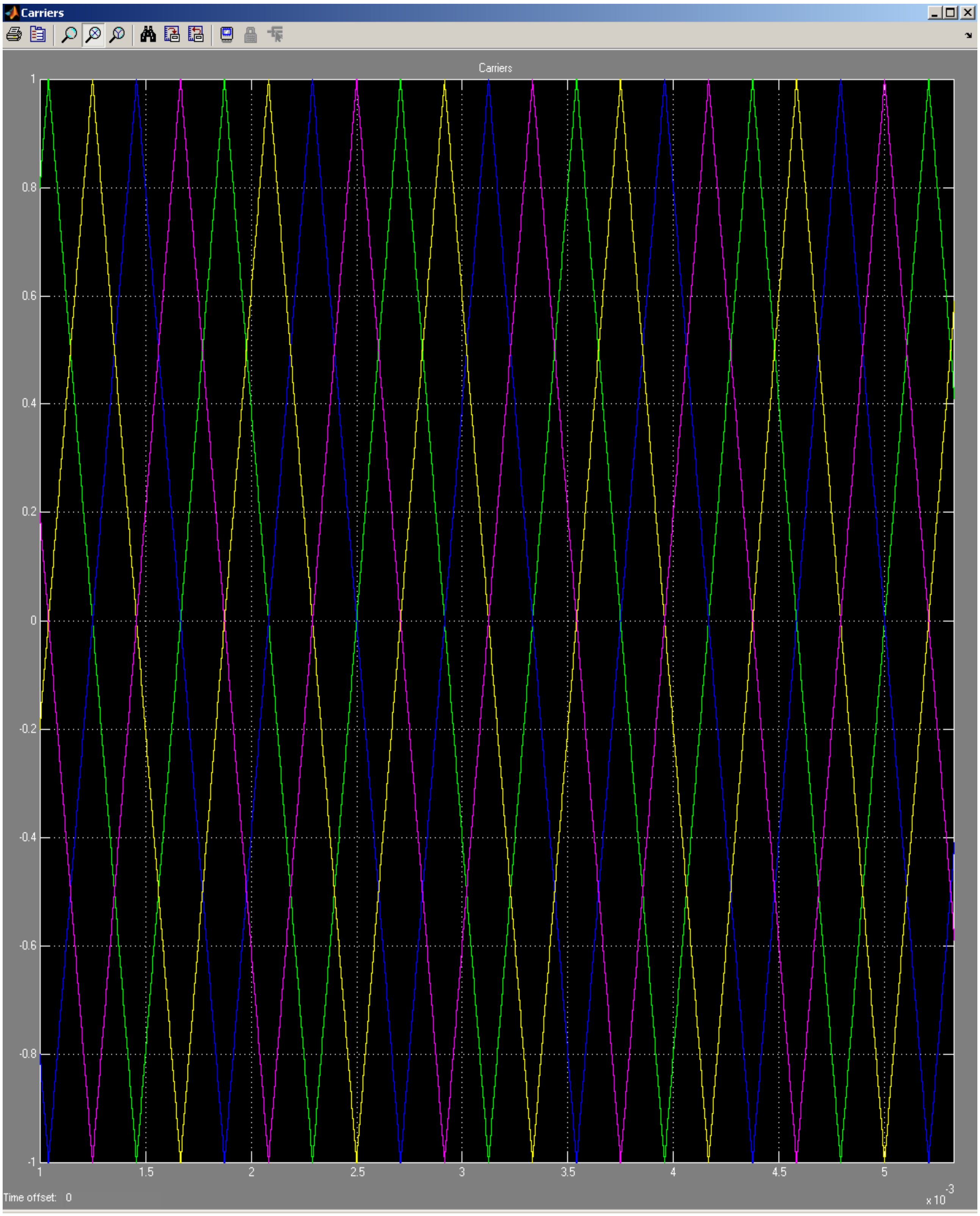


Figure 12 Carriers for Phase Shifted Modulation for 2400 Hz

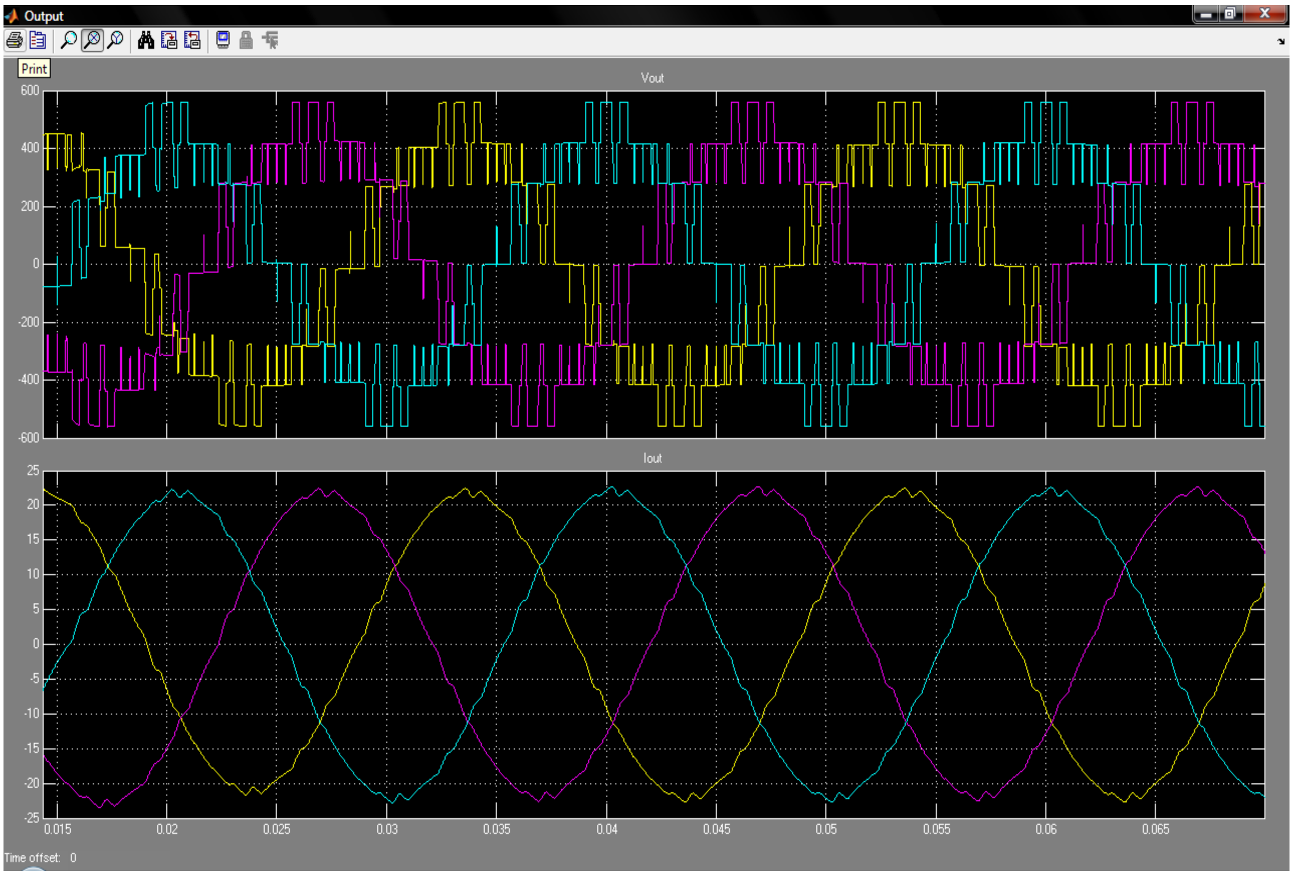


Figure 13 Output voltage and current for Level Shifted Modulation IPD at 600 Hz

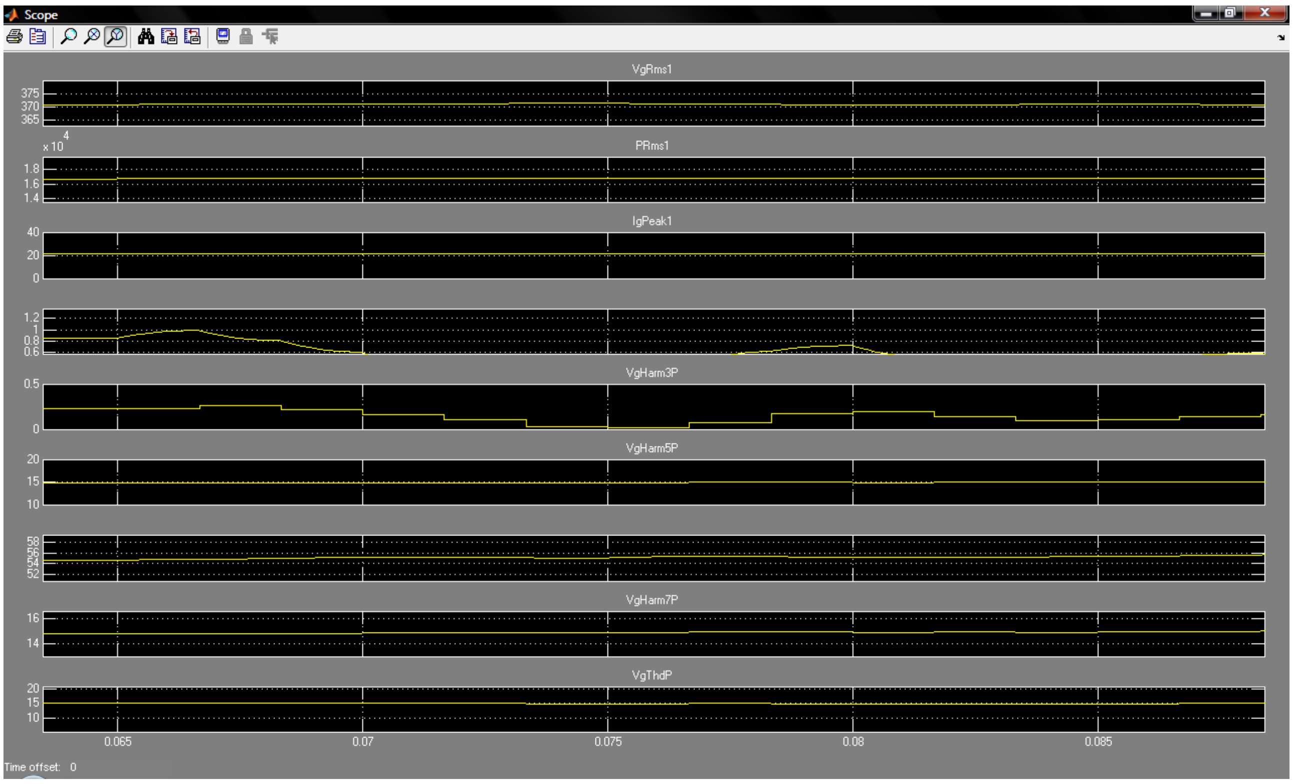


Figure 14 Voltage Harmonics for Level Shifted Modulation IPD at 600 Hz

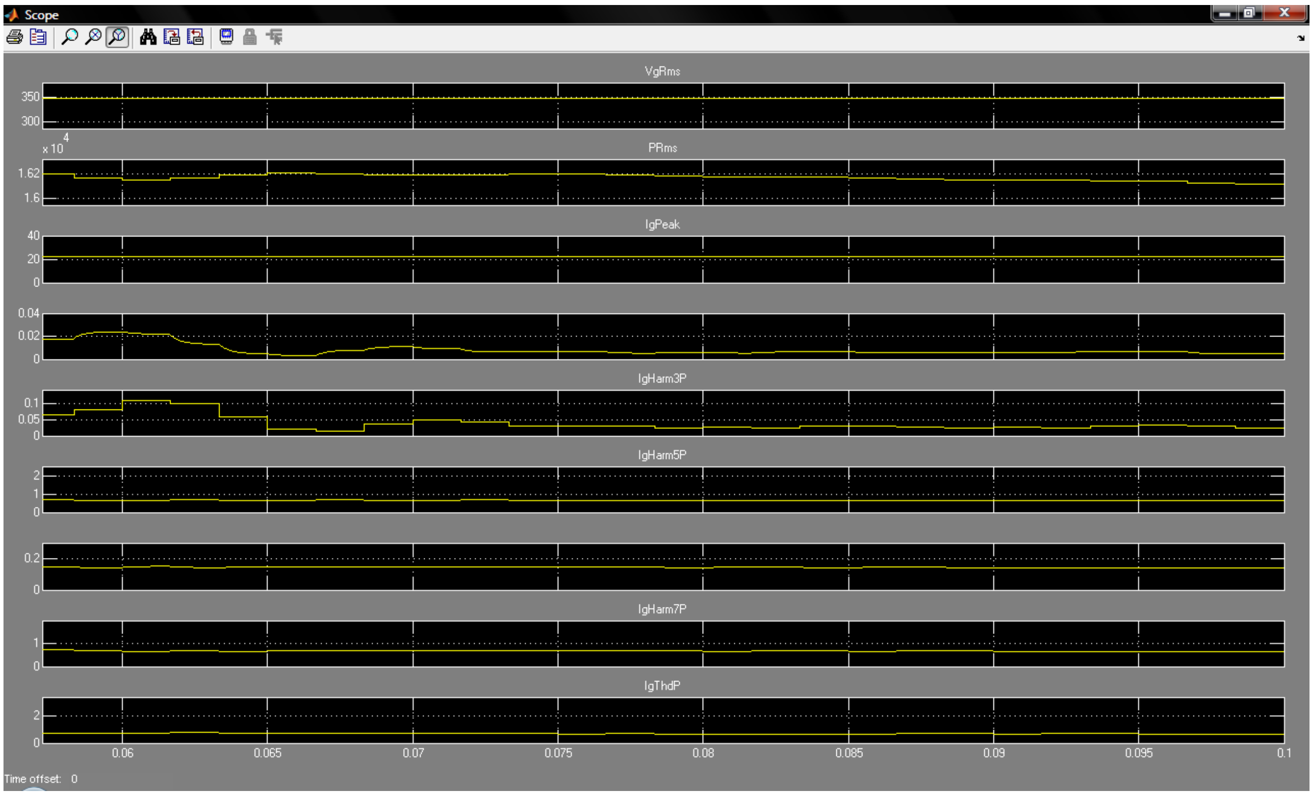


Figure 15 Current Harmonics for Level Shifted Modulation IPD at 600 Hz

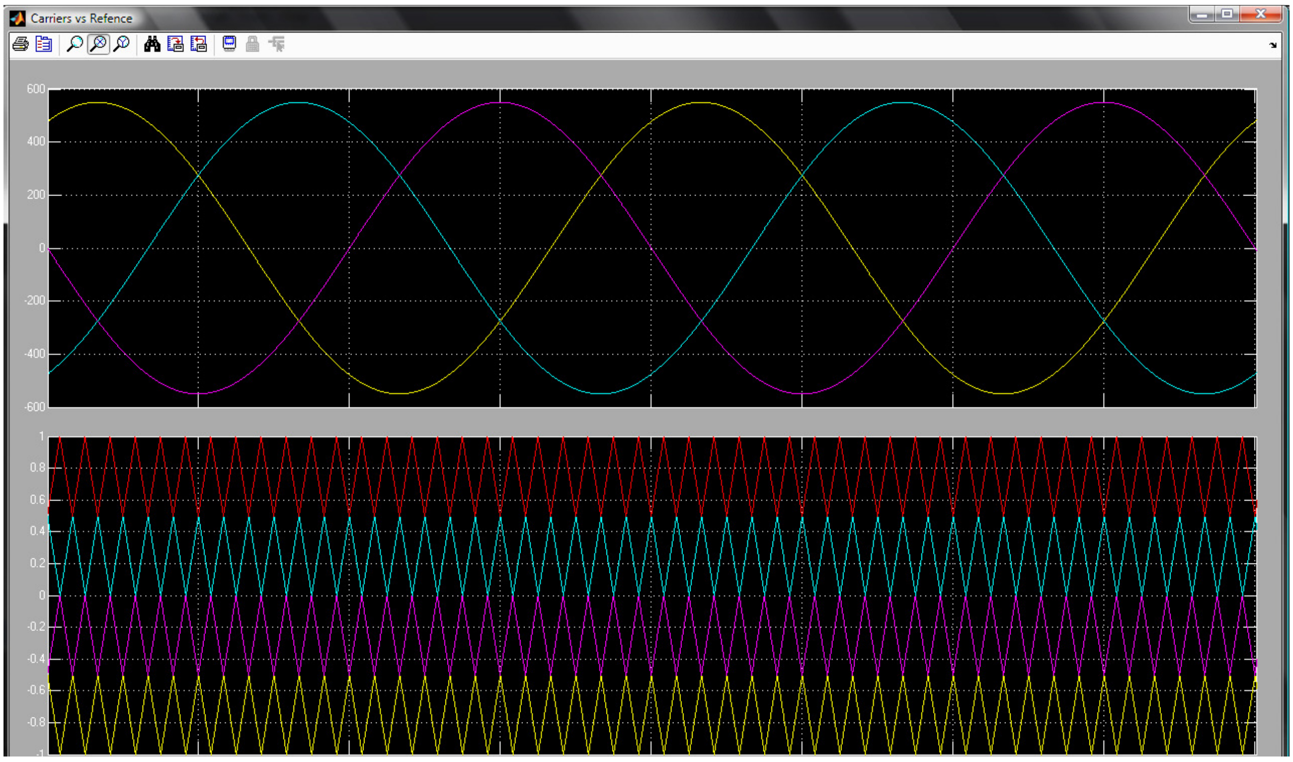


Figure 16 Carriers for Level Shifted Modulation IPD for 600 Hz

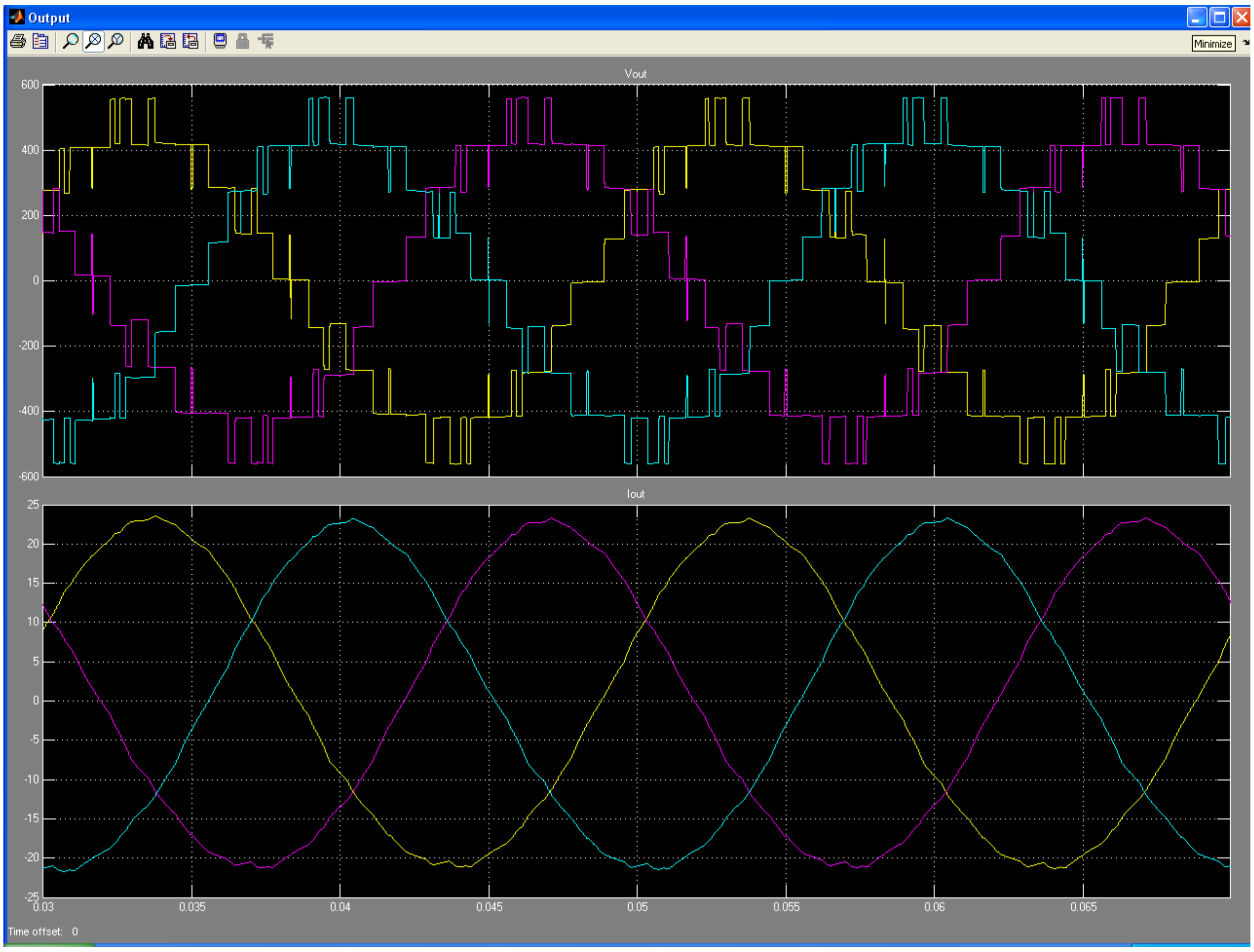


Figure 17 Output voltage and current for Level Shifted Modulation IPD at 1200 Hz

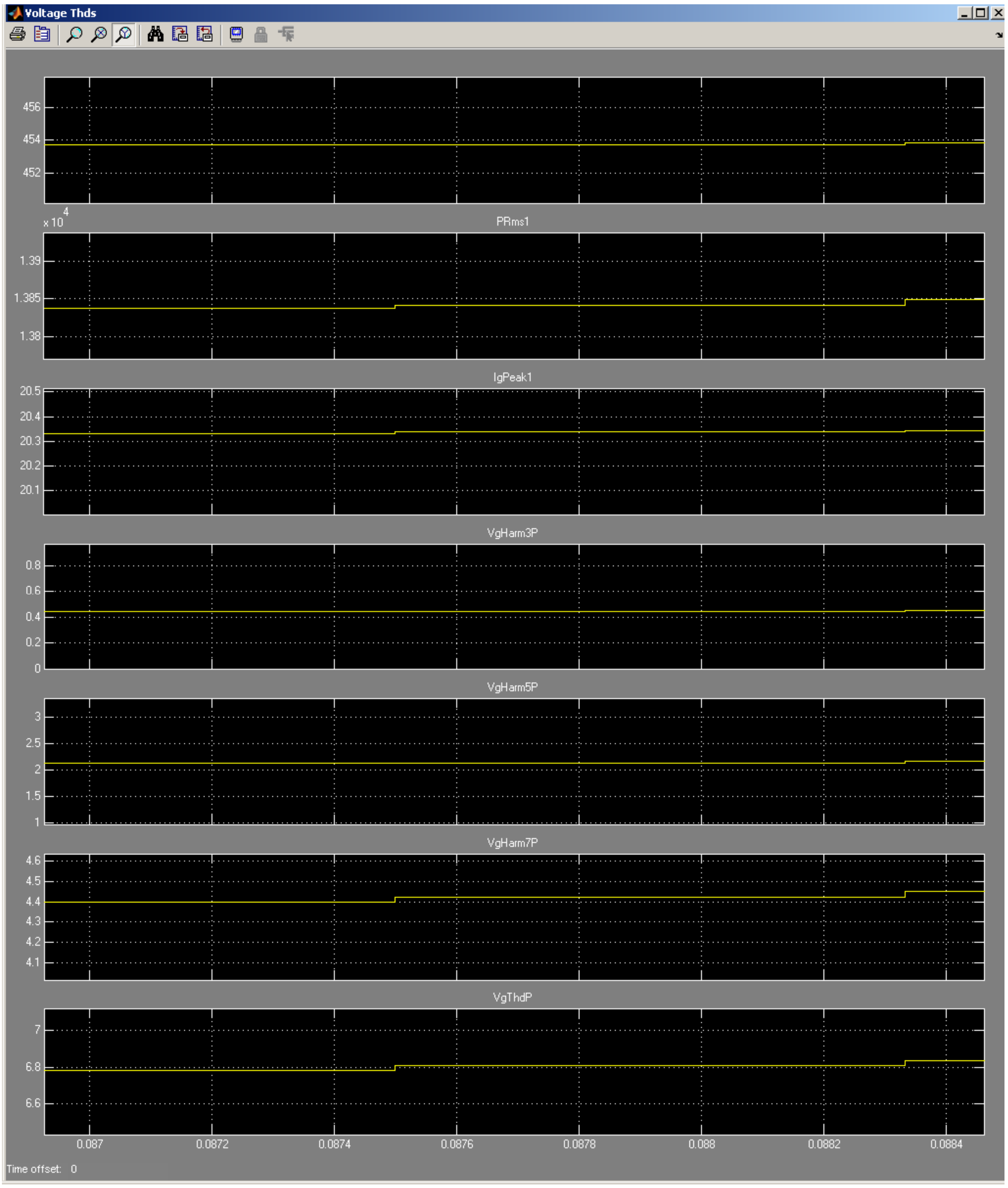


Figure 18 Voltage Harmonics for Level Shifted Modulation IPD at 1200 Hz

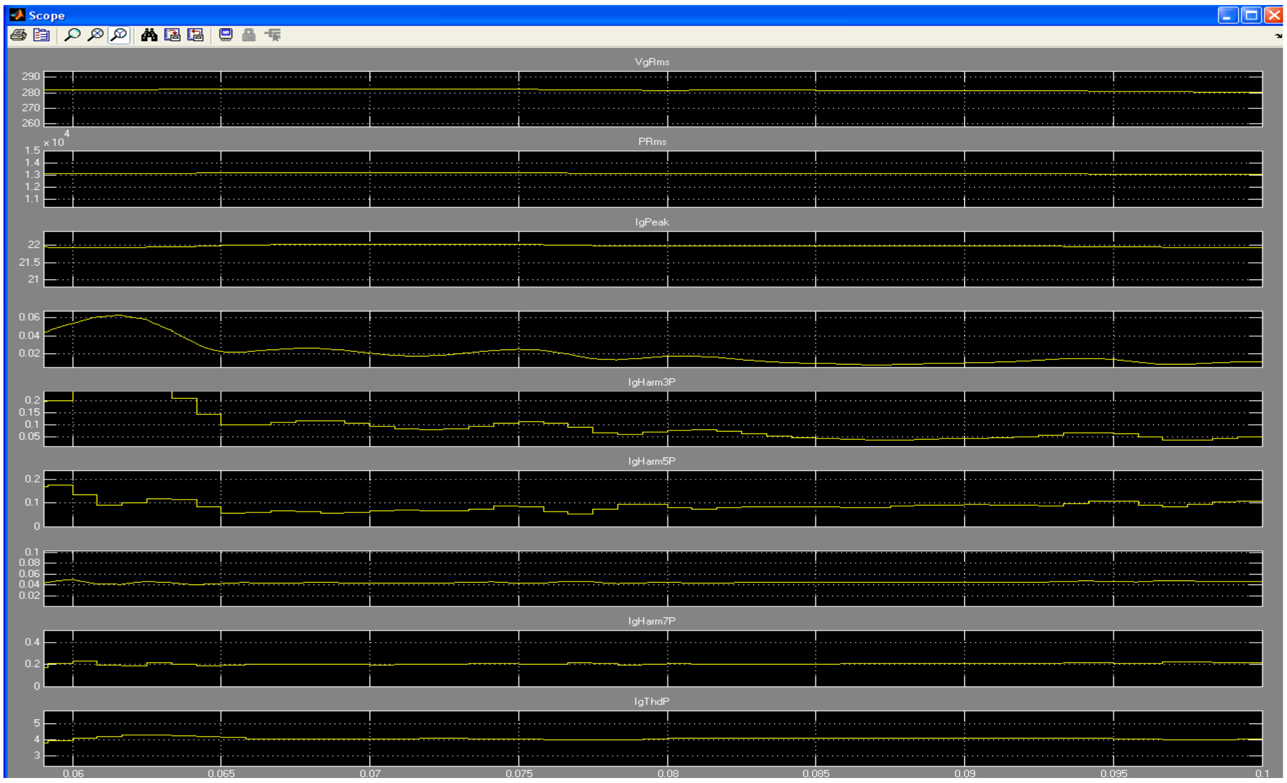


Figure 19 Current Harmonics for Level Shifted Modulation IPD at 1200 Hz

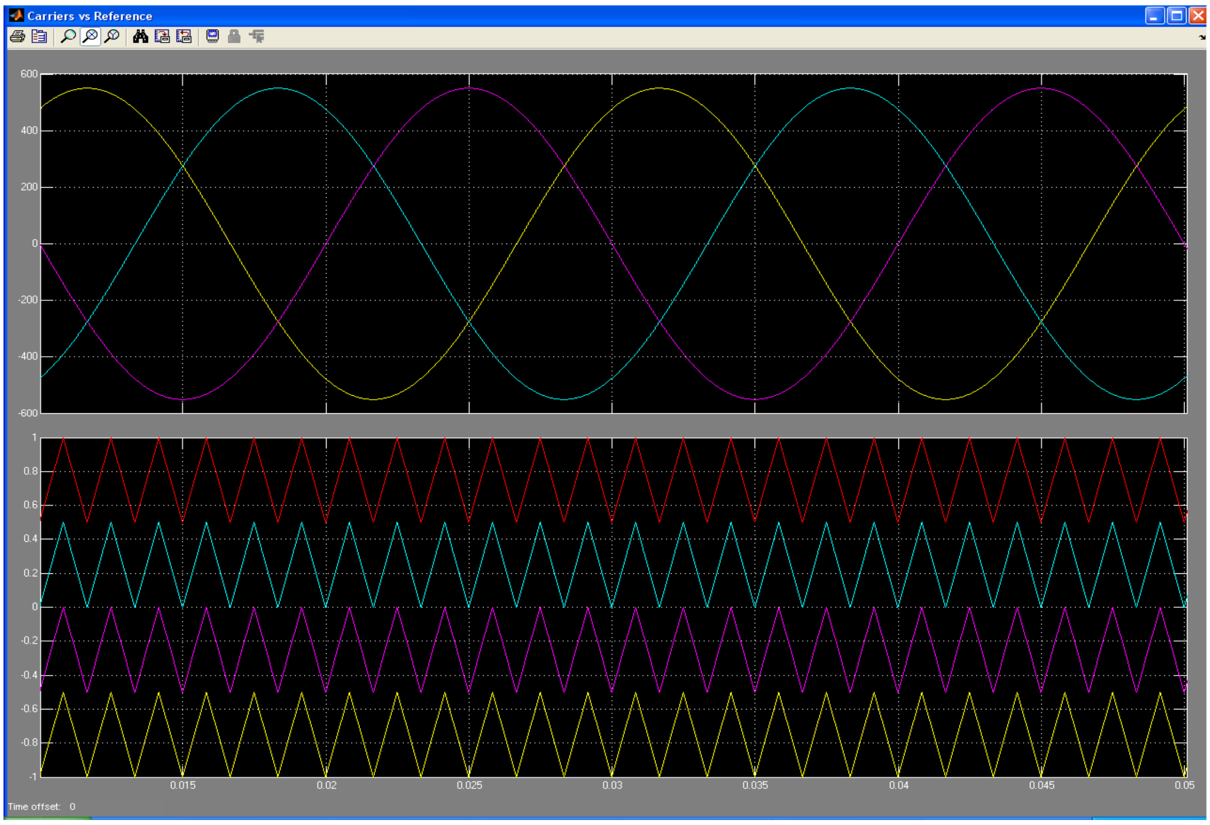


Figure 20 Carriers for Level Shifted Modulation IPD for 1200 Hz

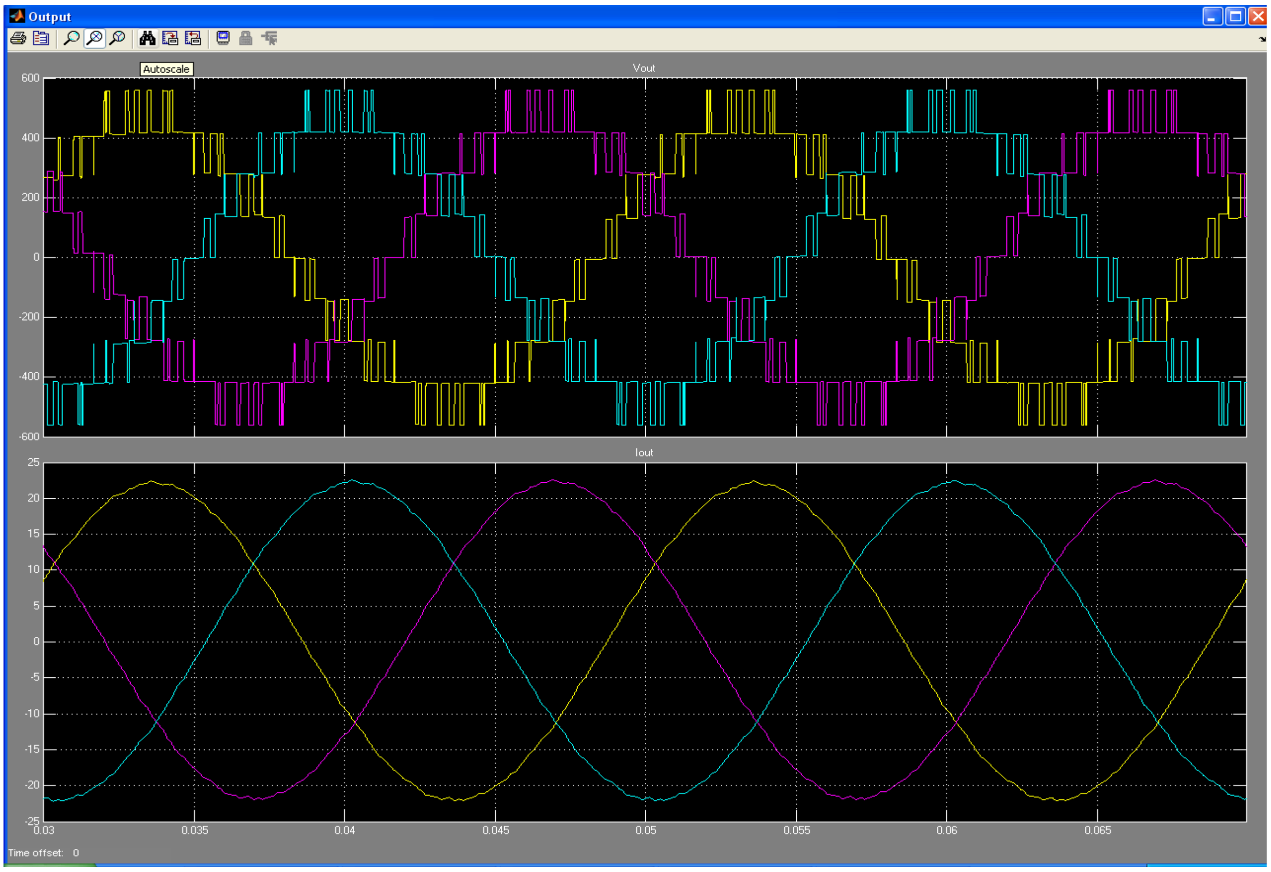


Figure 21 Output voltage and current for Level Shifted Modulation IPD at 2400 Hz

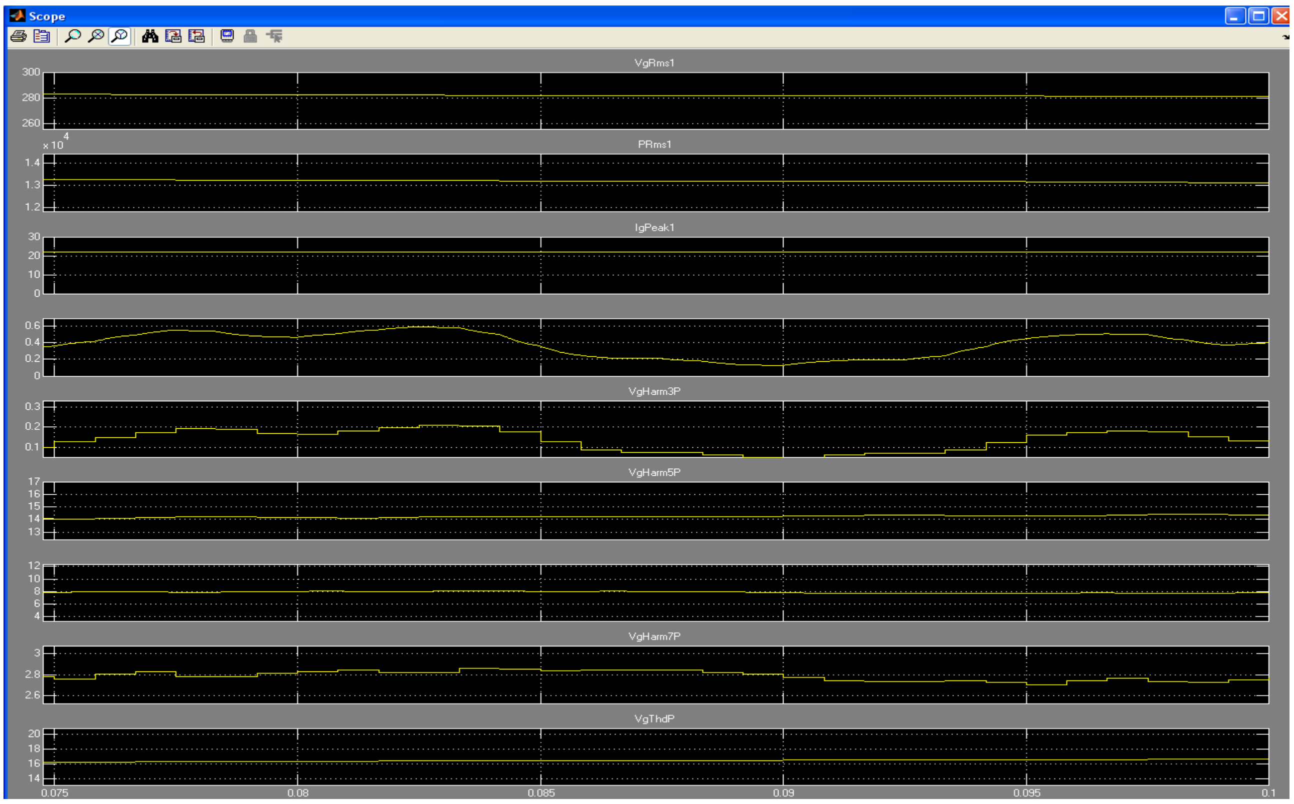


Figure 22 Voltage Harmonics for Level Shifted Modulation IPD at 2400 Hz

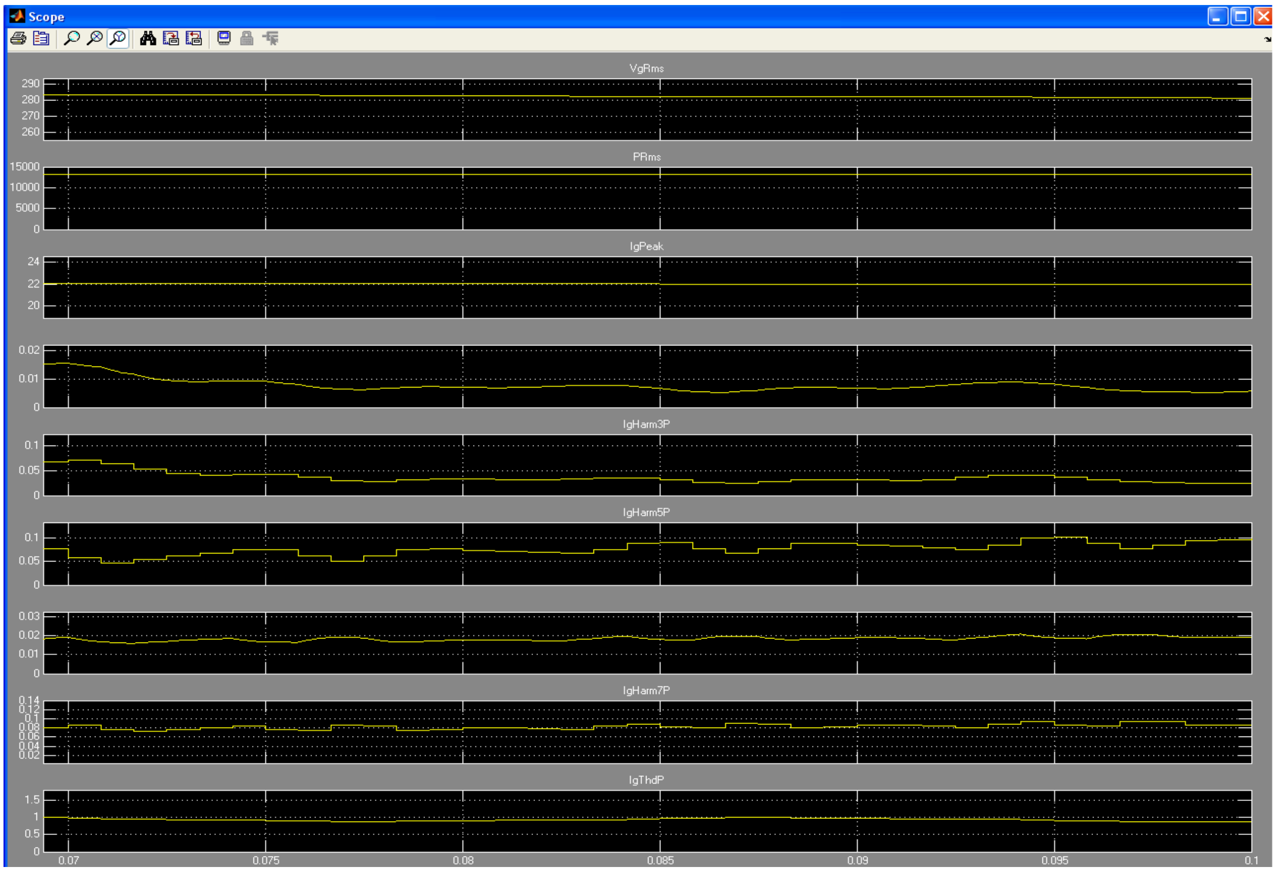


Figure 23 Current Harmonics for Level Shifted Modulation IPD at 2400 Hz

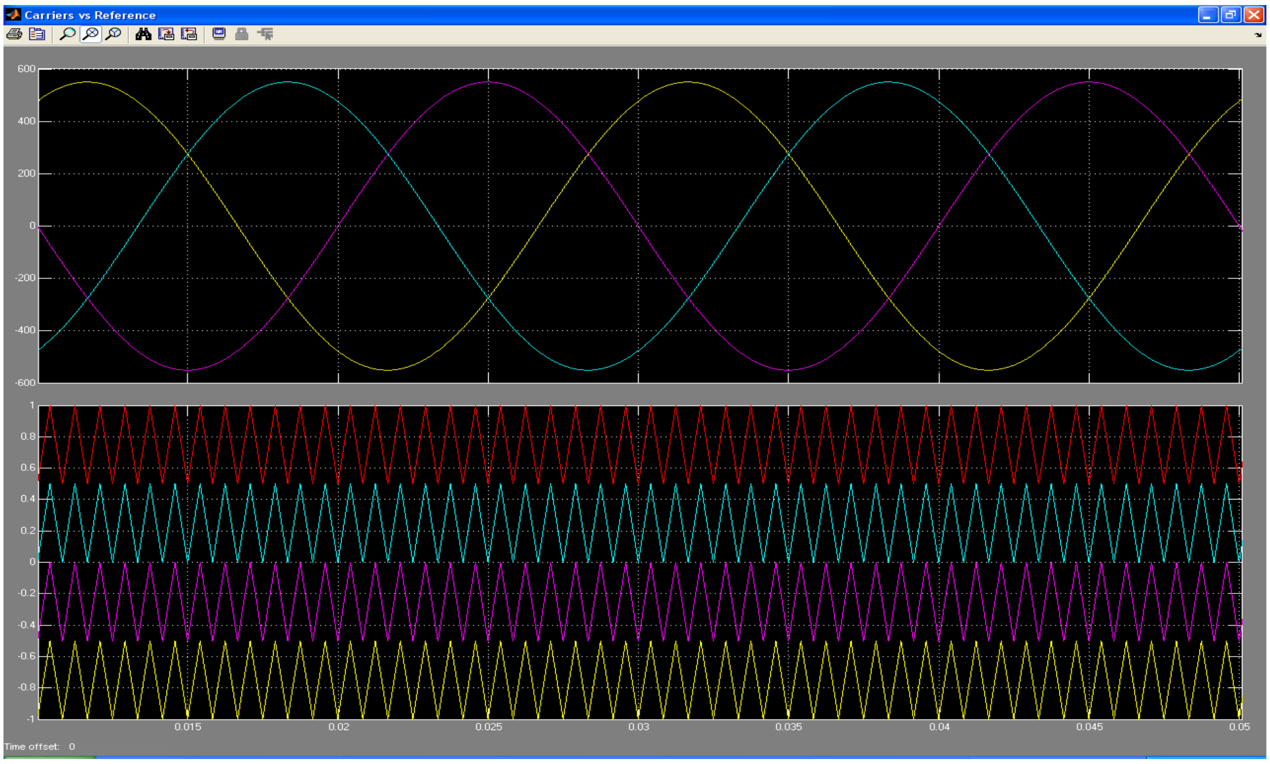


Figure 24 Carriers for Level Shifted Modulation IPD for 2400 Hz

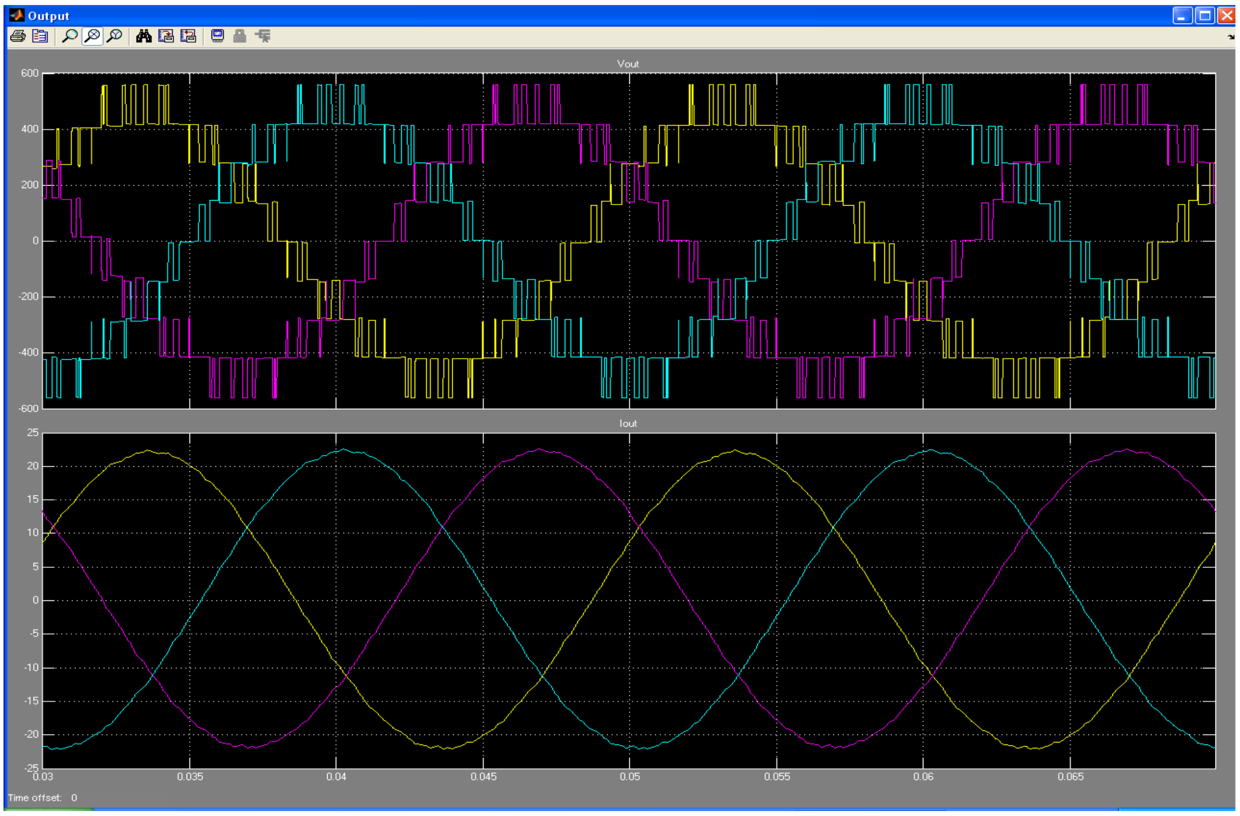


Figure 25 Output voltage and current for Level Shifted Modulation APOD at 600 Hz

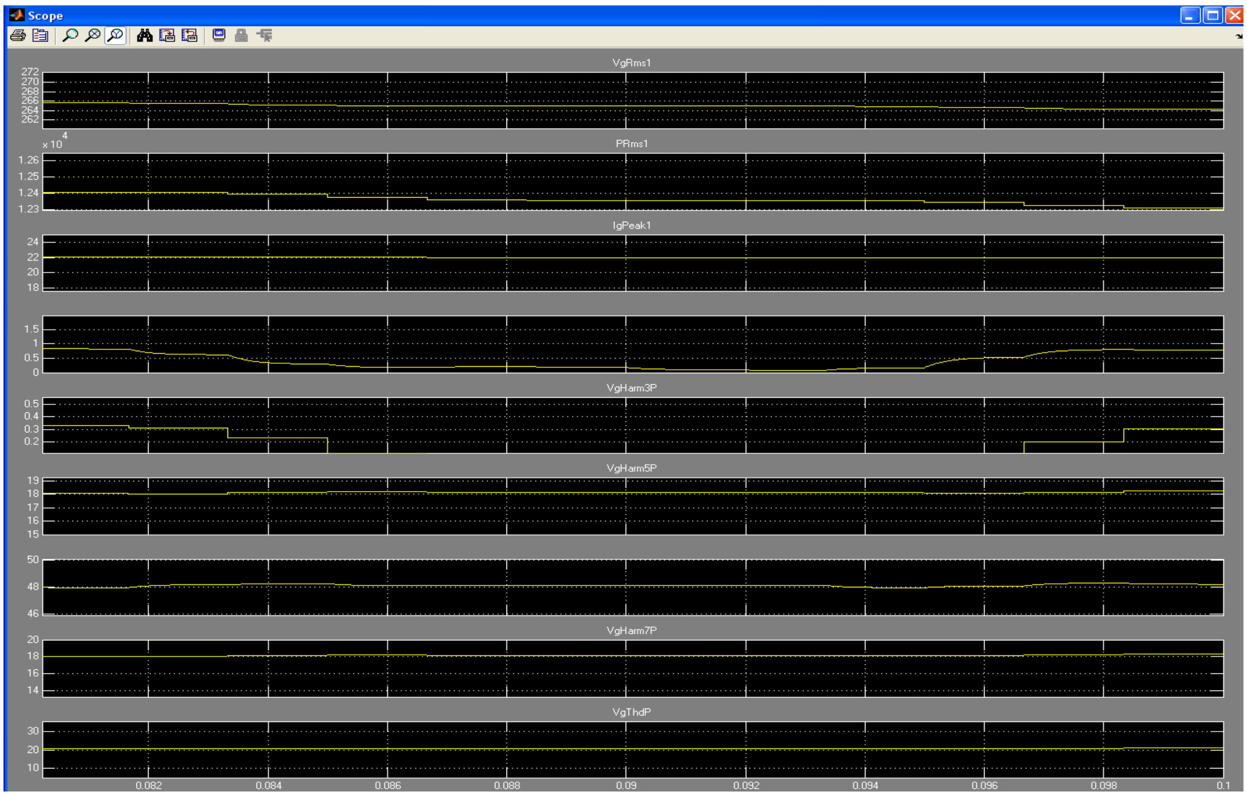


Figure 26 Voltage Harmonics for Level Shifted Modulation APOD at 600 Hz

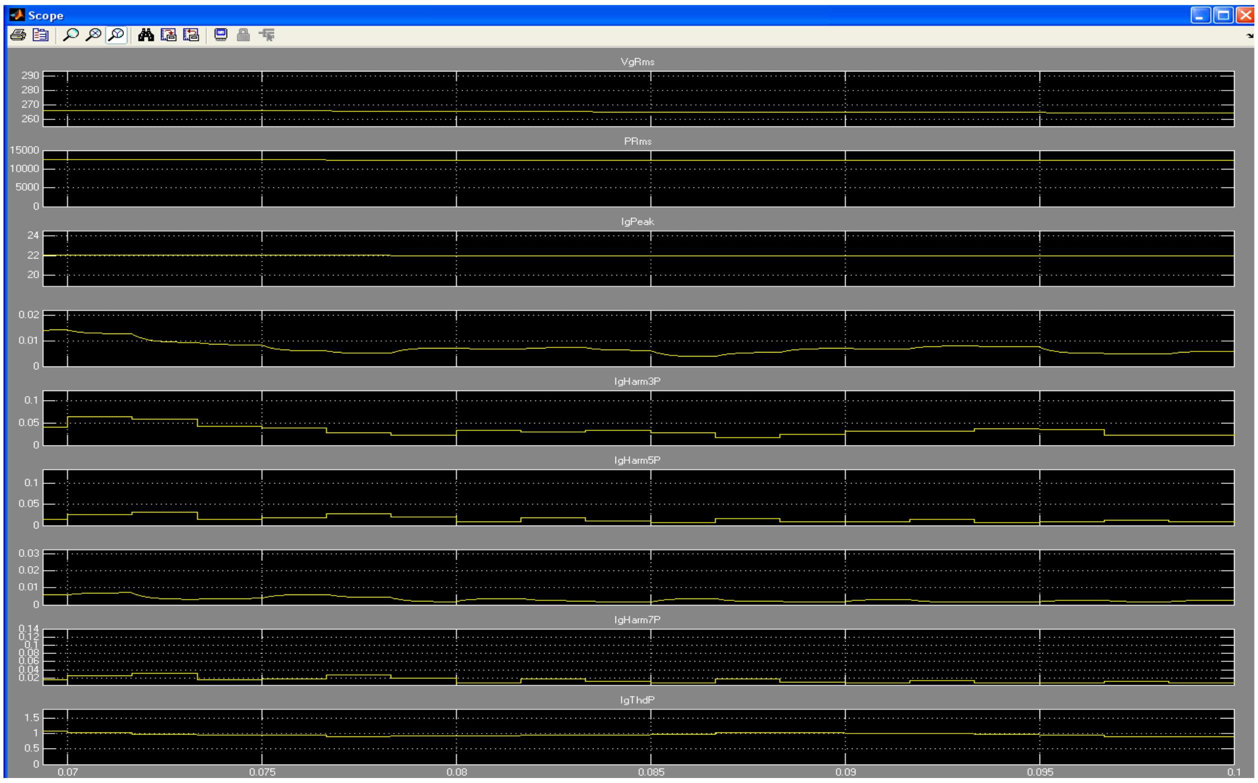


Figure 27 Current Harmonics for Level Shifted Modulation APOD at 600 Hz

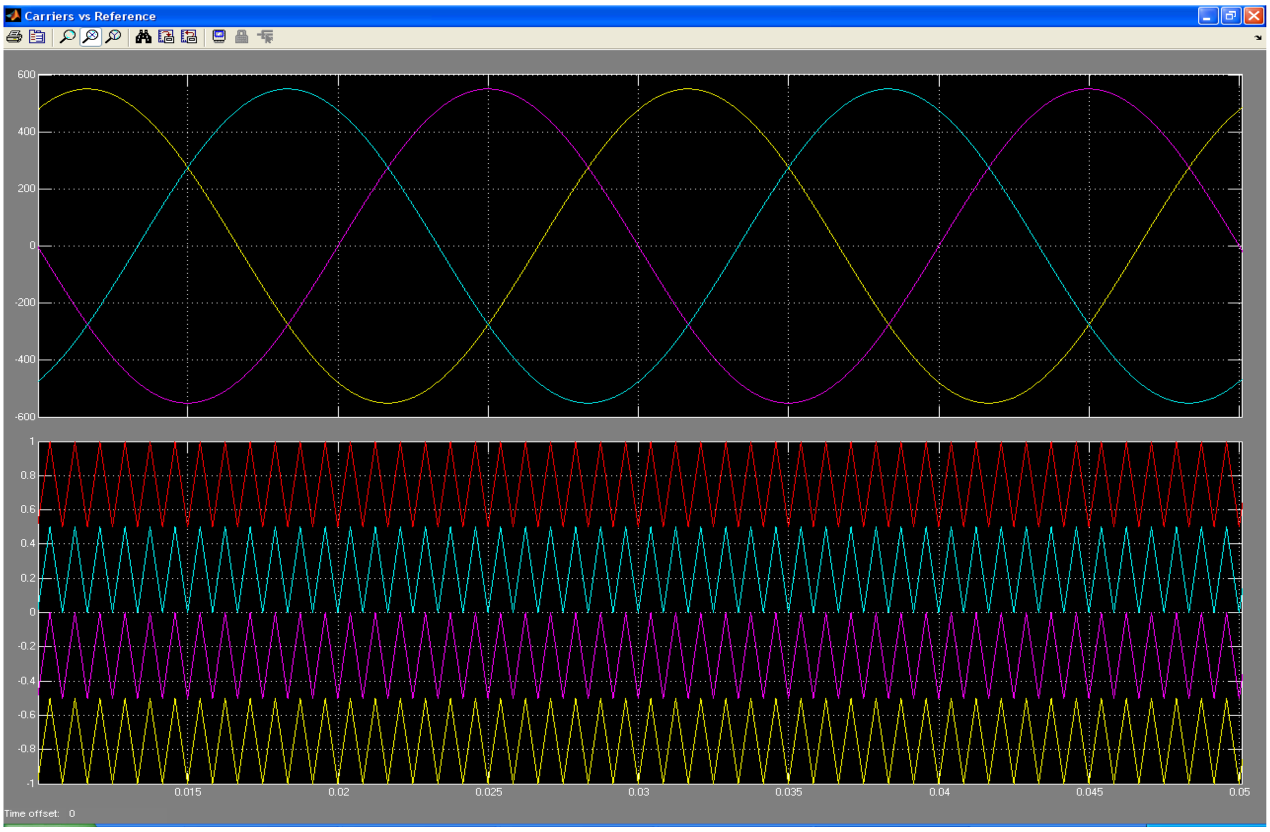


Figure 28 Carriers for Level Shifted Modulation APOD for 600 Hz

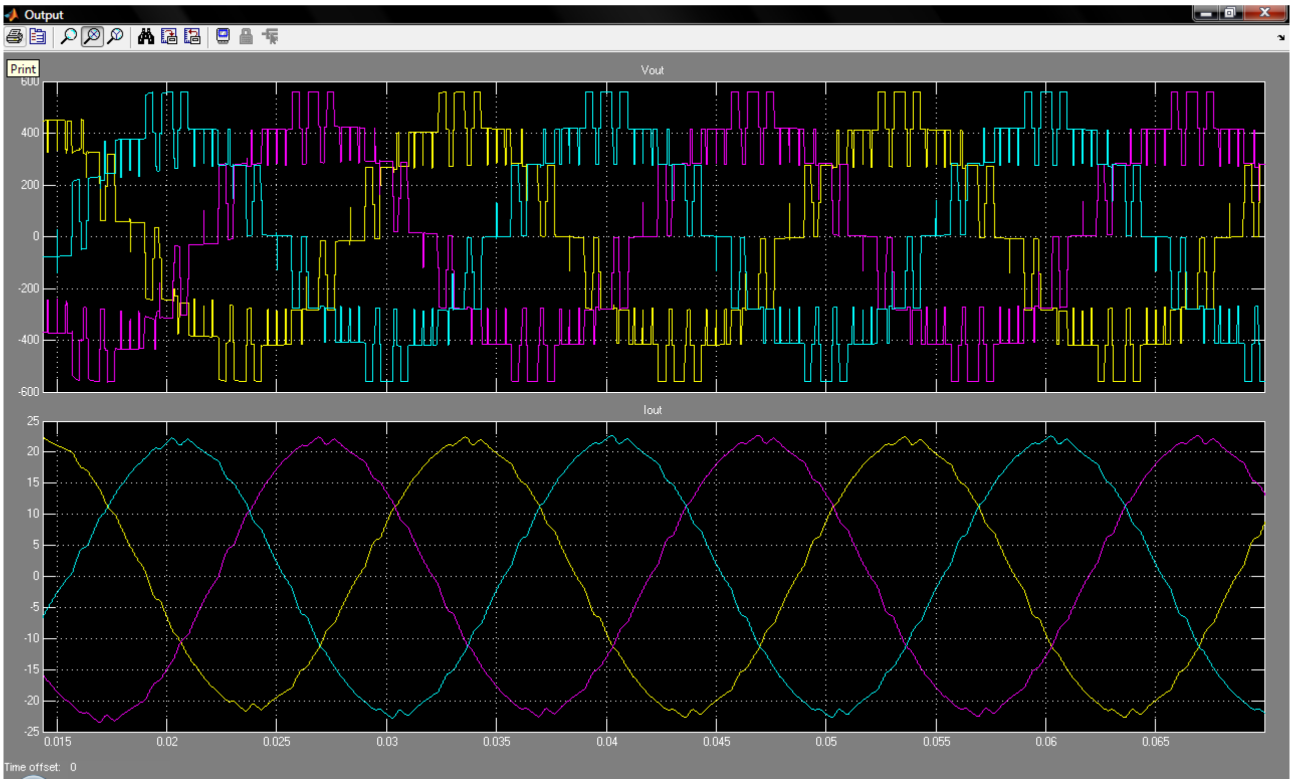


Figure 29 Output voltage and current for Level Shifted Modulation APOD at 1200 H

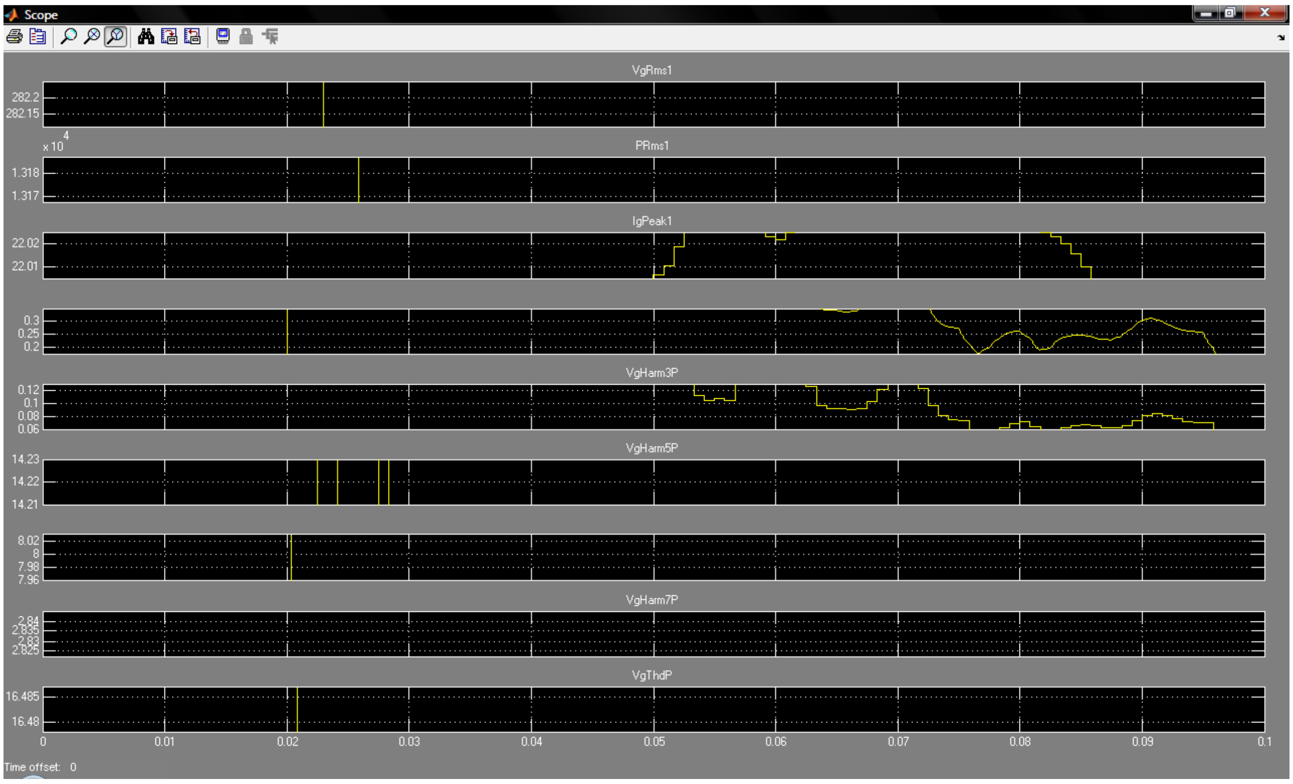


Figure 30 Voltage Harmonics for Level Shifted Modulation APOD at 1200 Hz

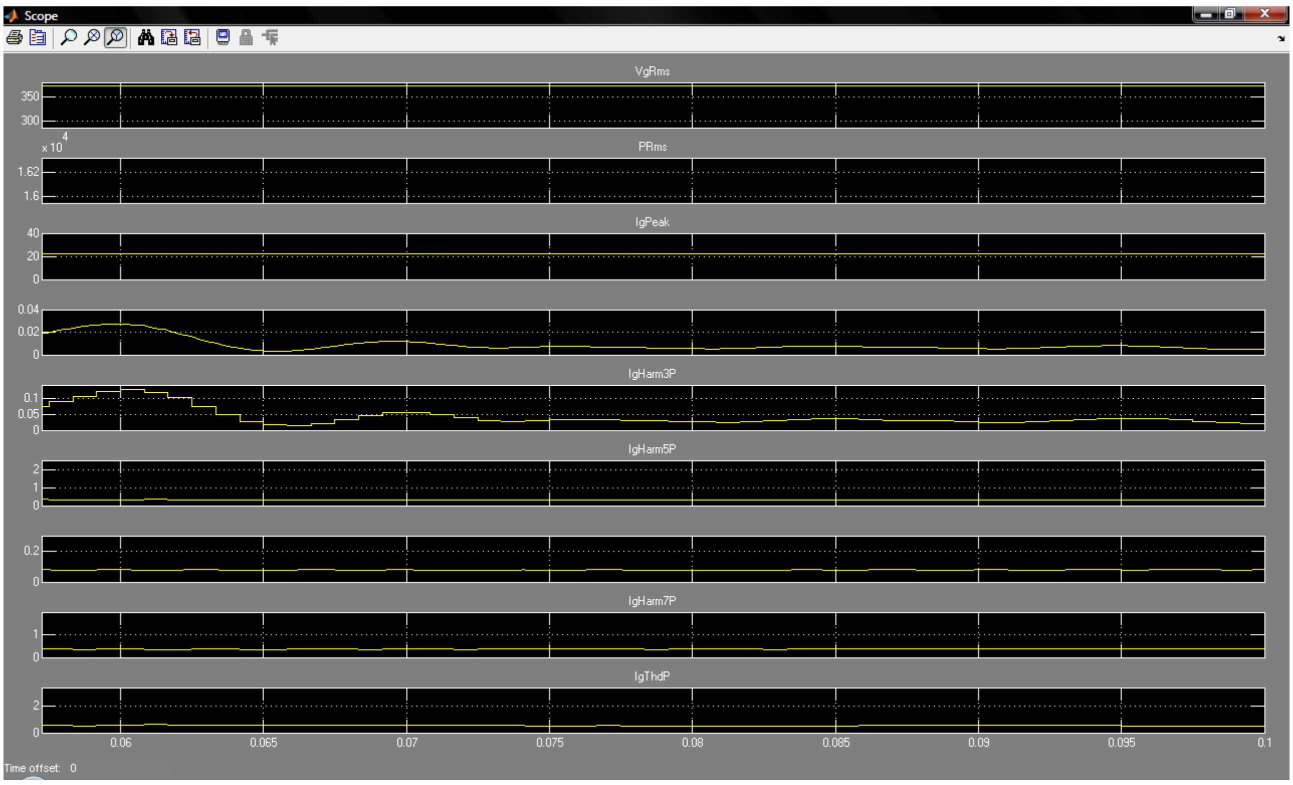


Figure 31 Current Harmonics for Level Shifted Modulation APOD at 1200 Hz

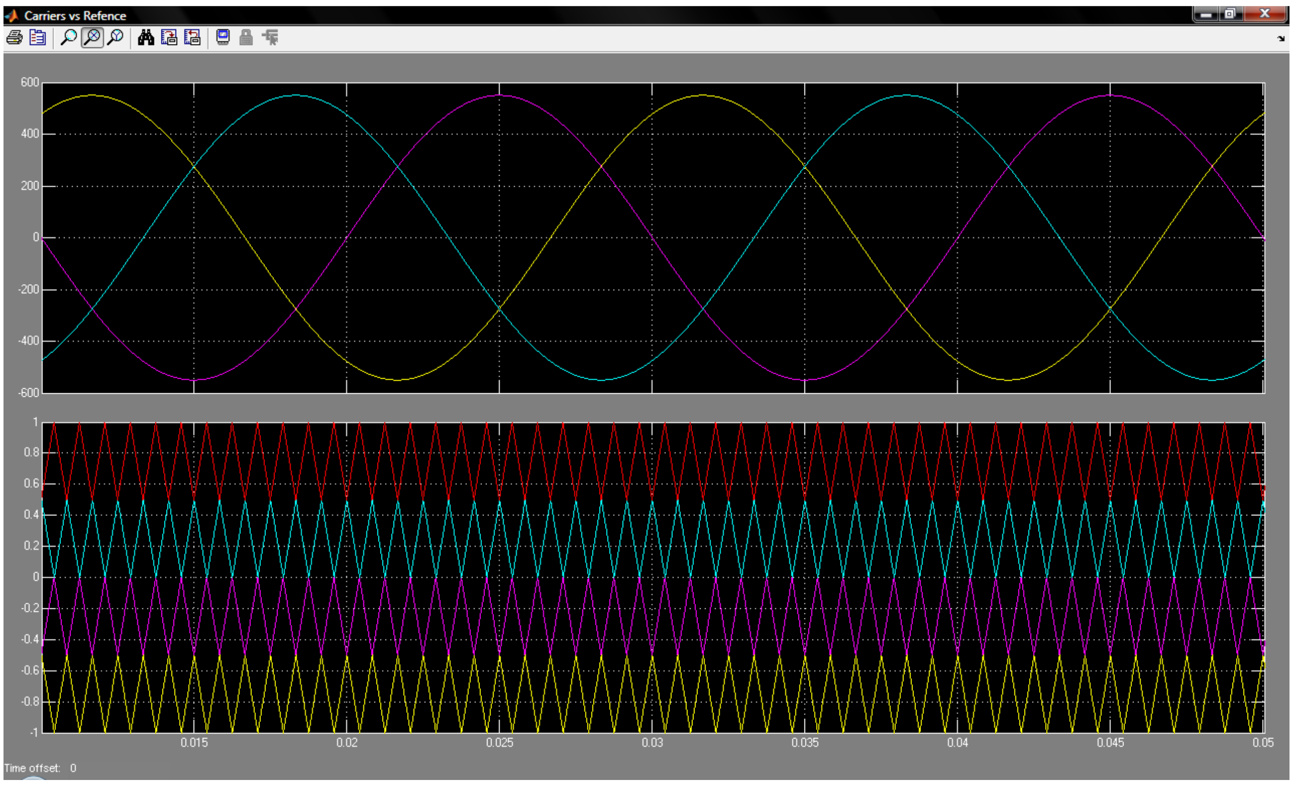


Figure 32 Carriers for Level Shifted Modulation APOD for 1200 Hz

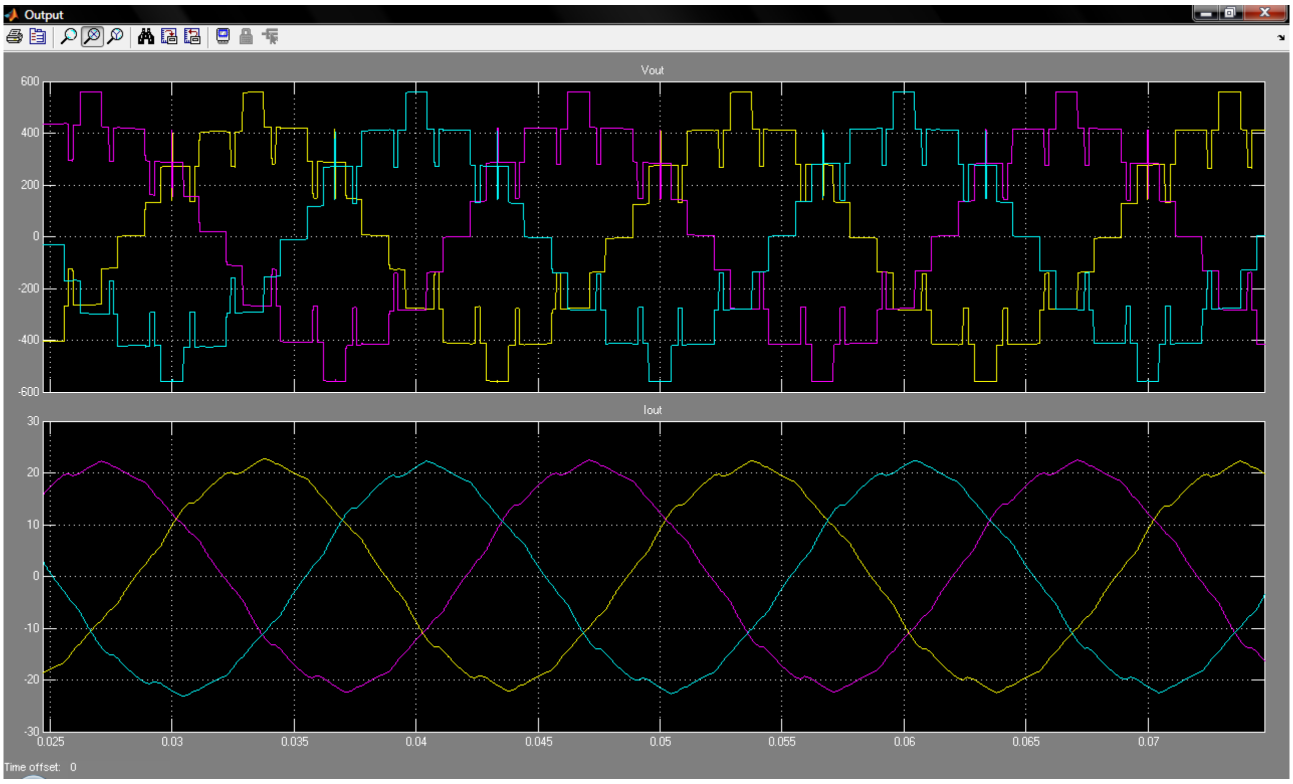


Figure 33 Output voltage and current for Level Shifted Modulation APOD at 2400 Hz

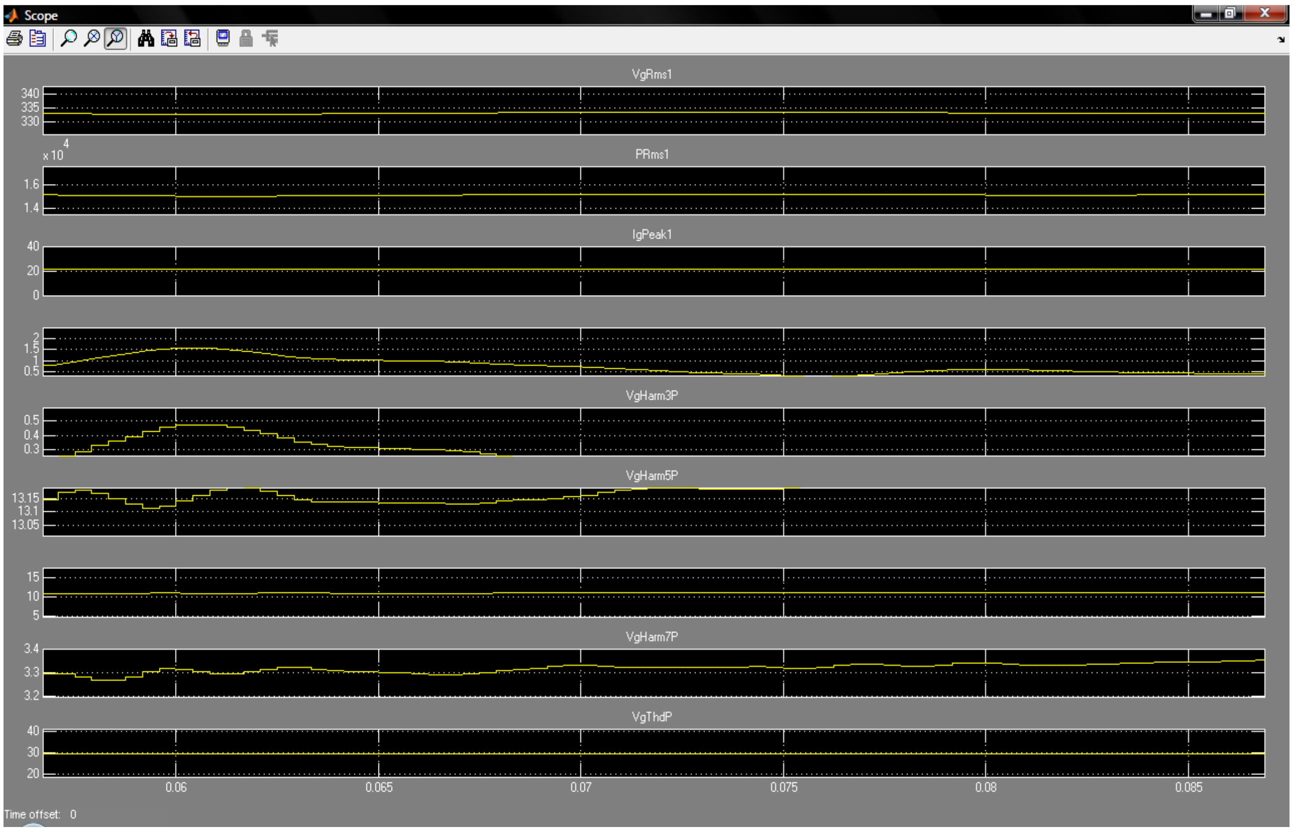


Figure 34 Voltage Harmonics for Level Shifted Modulation APOD at 2400 Hz

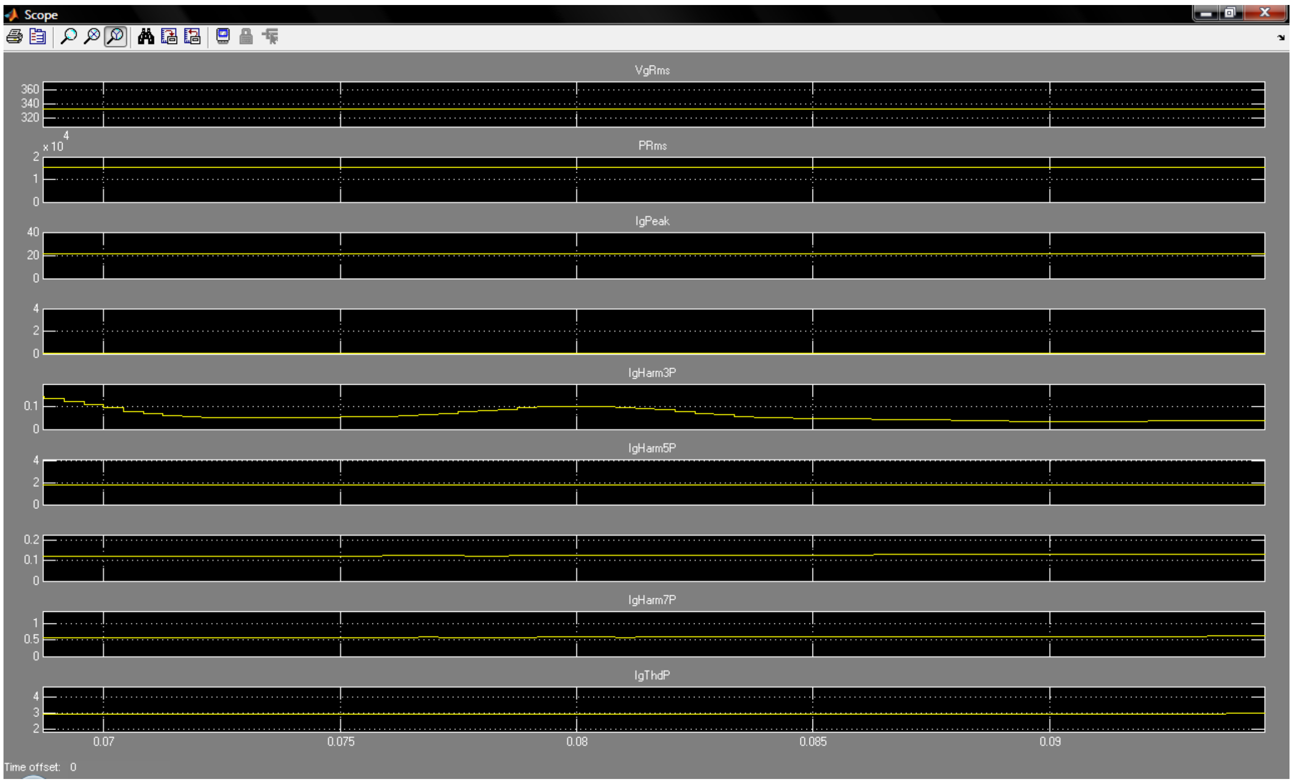


Figure 35 Current Harmonics for Level Shifted Modulation APOD at 24000 Hz

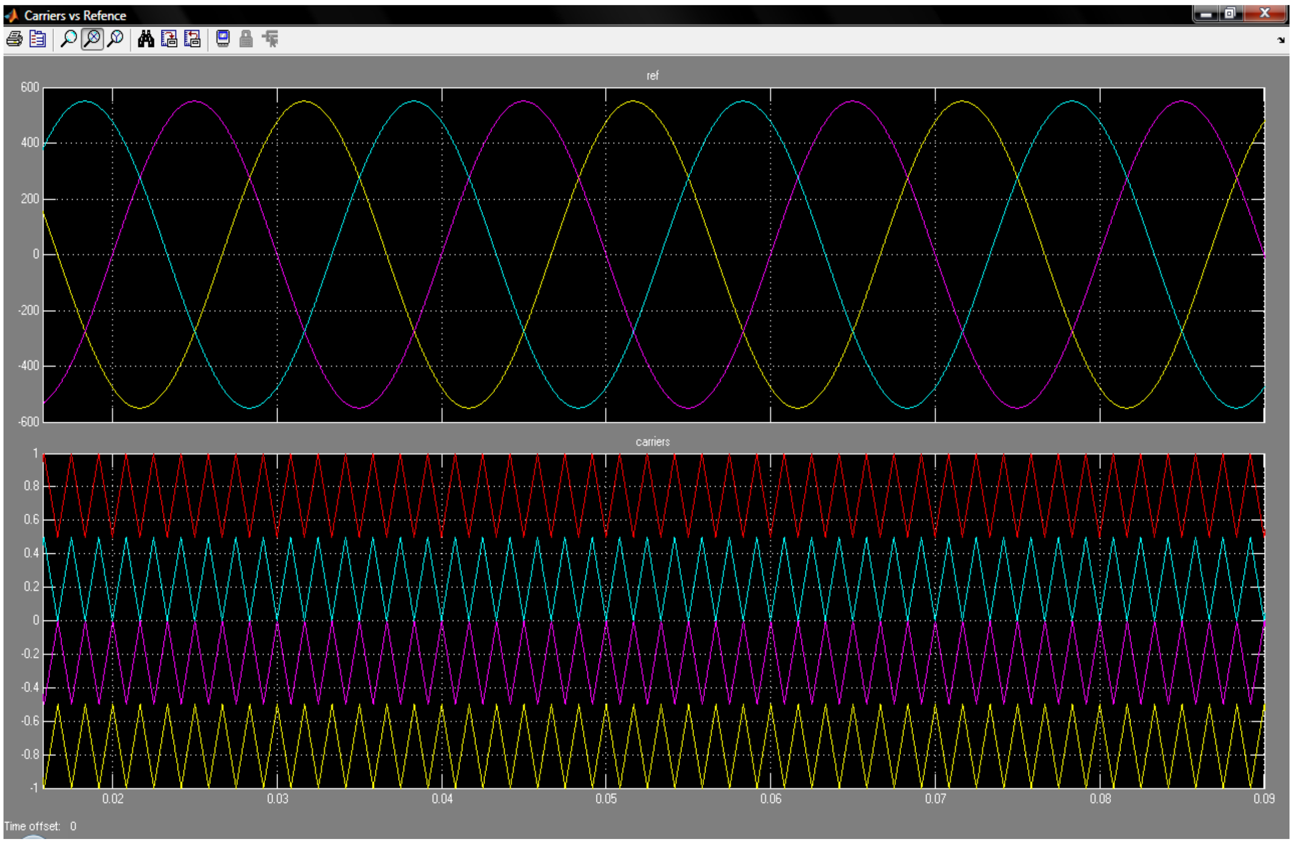


Figure 36 Carriers for Level Shifted Modulation APOD for 2400 Hz

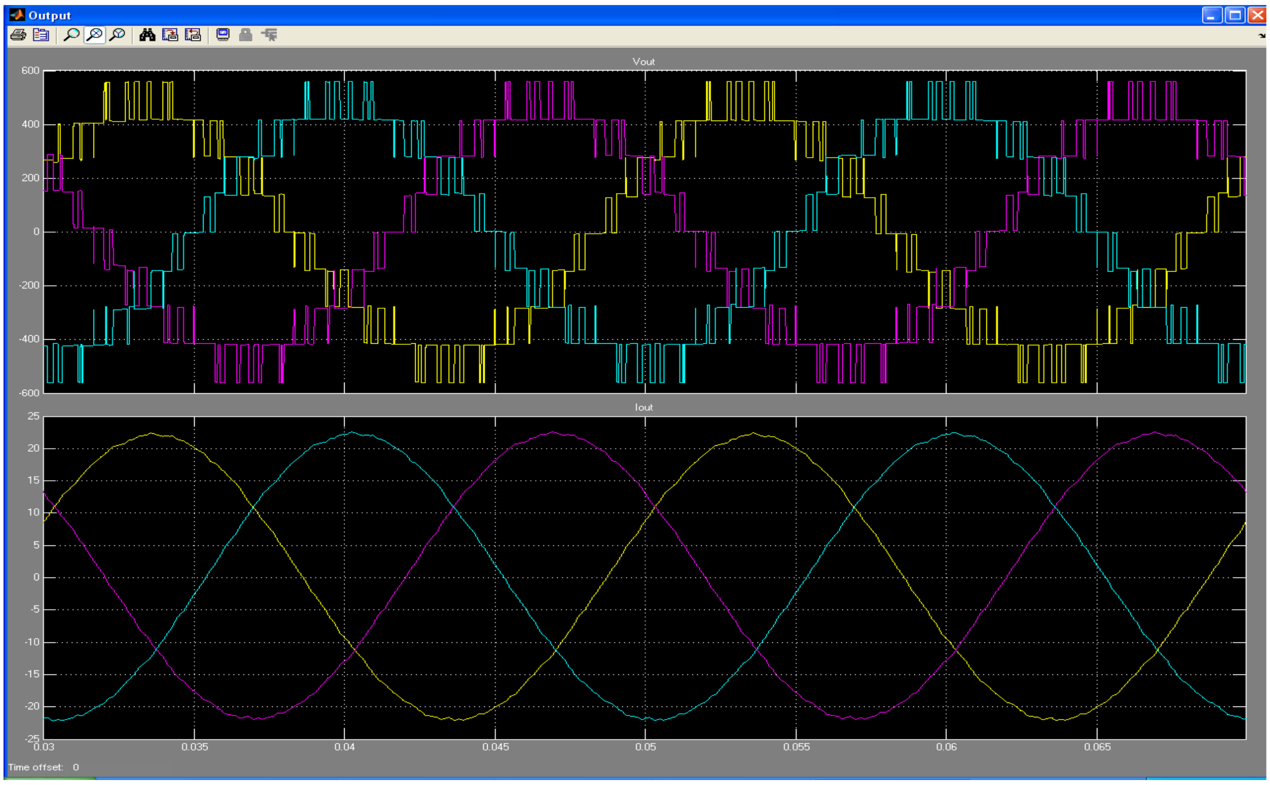


Figure 37 Output voltage and current for Level Shifted Modulation POD at 600 Hz

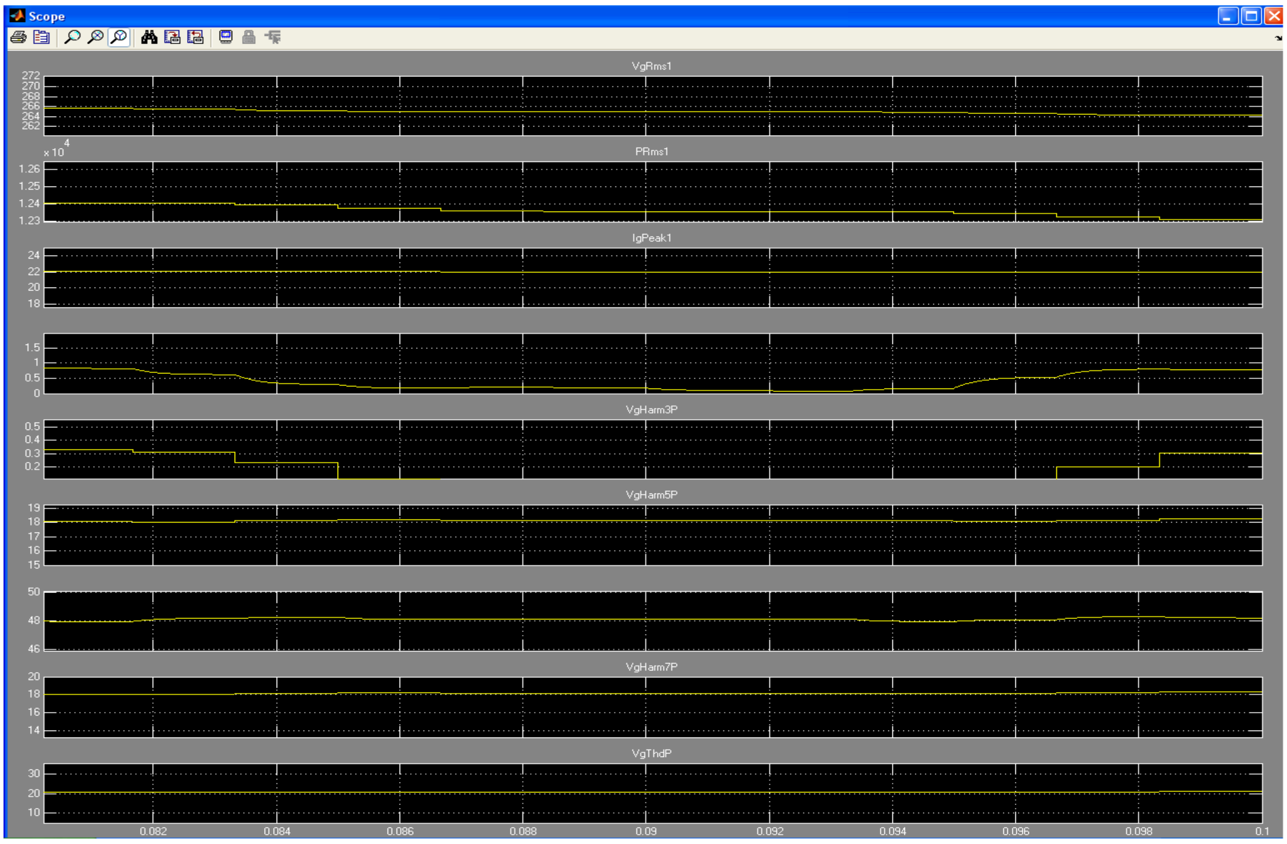


Figure 38 Voltage Harmonics for Level Shifted Modulation POD at 600 Hz

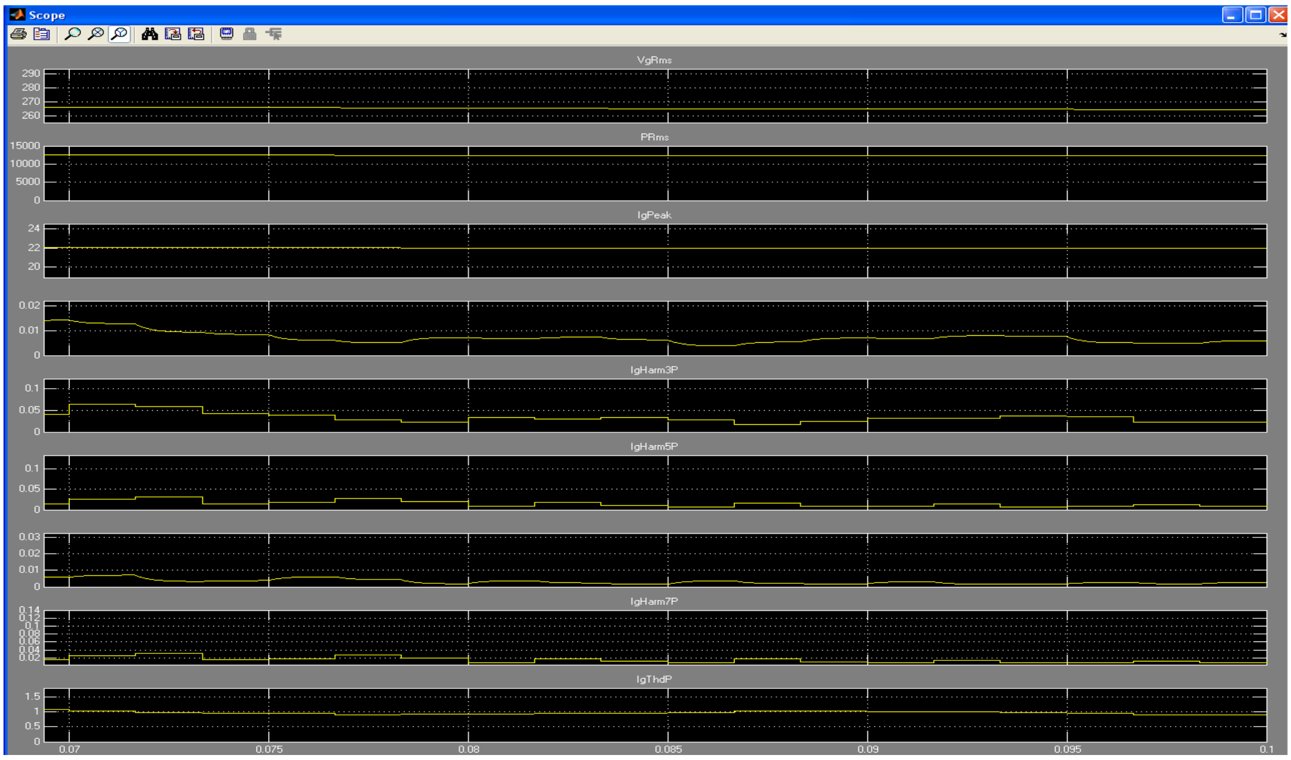


Figure 39 Current Harmonics for Level Shifted Modulation POD at 600 Hz

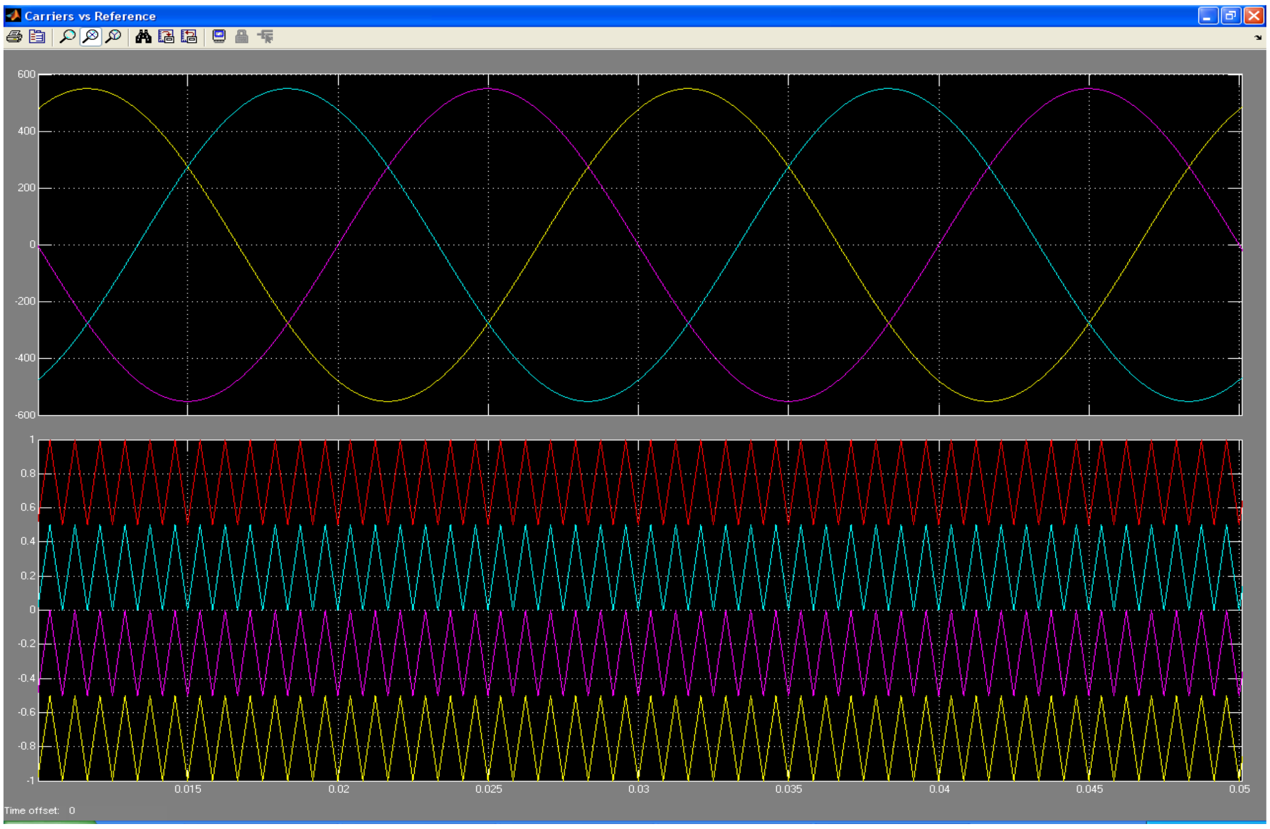


Figure 40 Carriers for Level Shifted Modulation POD for 600 Hz

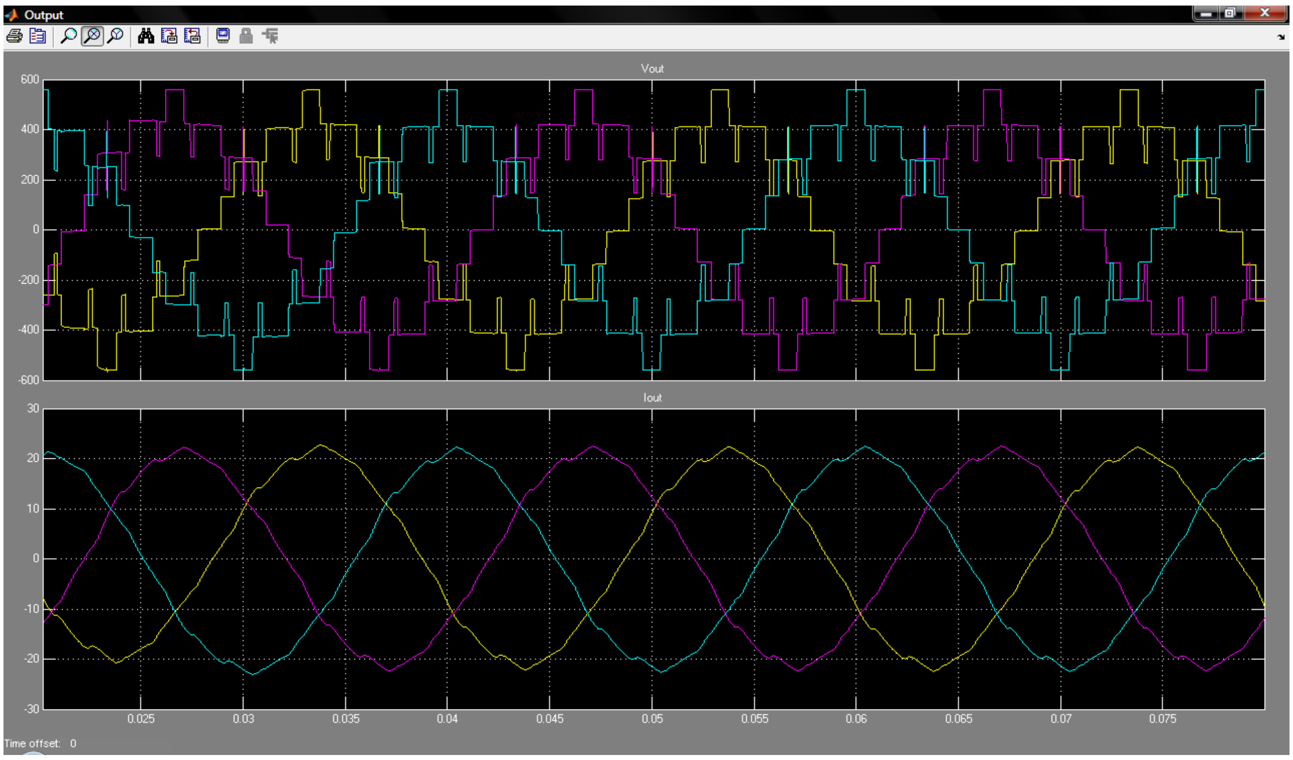


Figure 41 Output voltage and current for Level Shifted Modulation POD at 1200 Hz

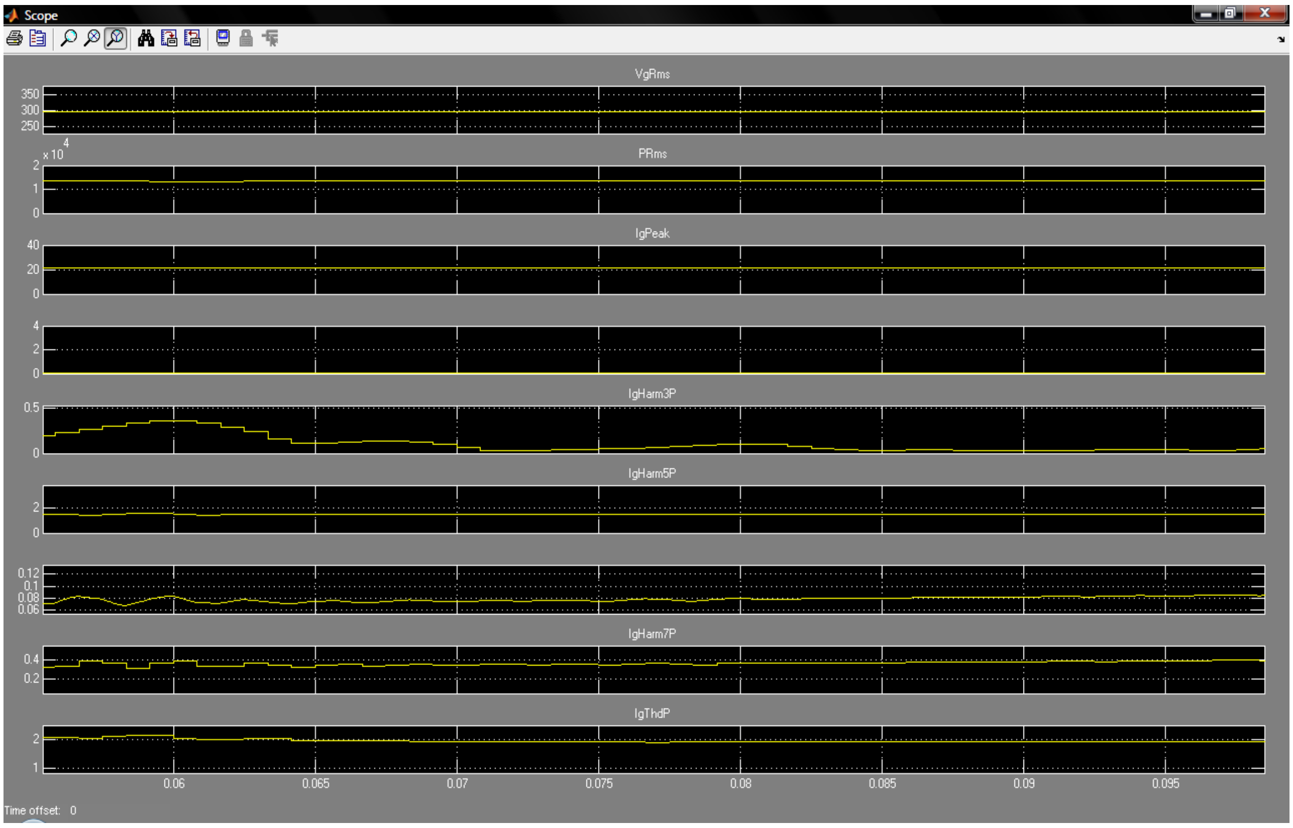


Figure 42 Voltage Harmonics for Level Shifted Modulation POD at 1200 Hz

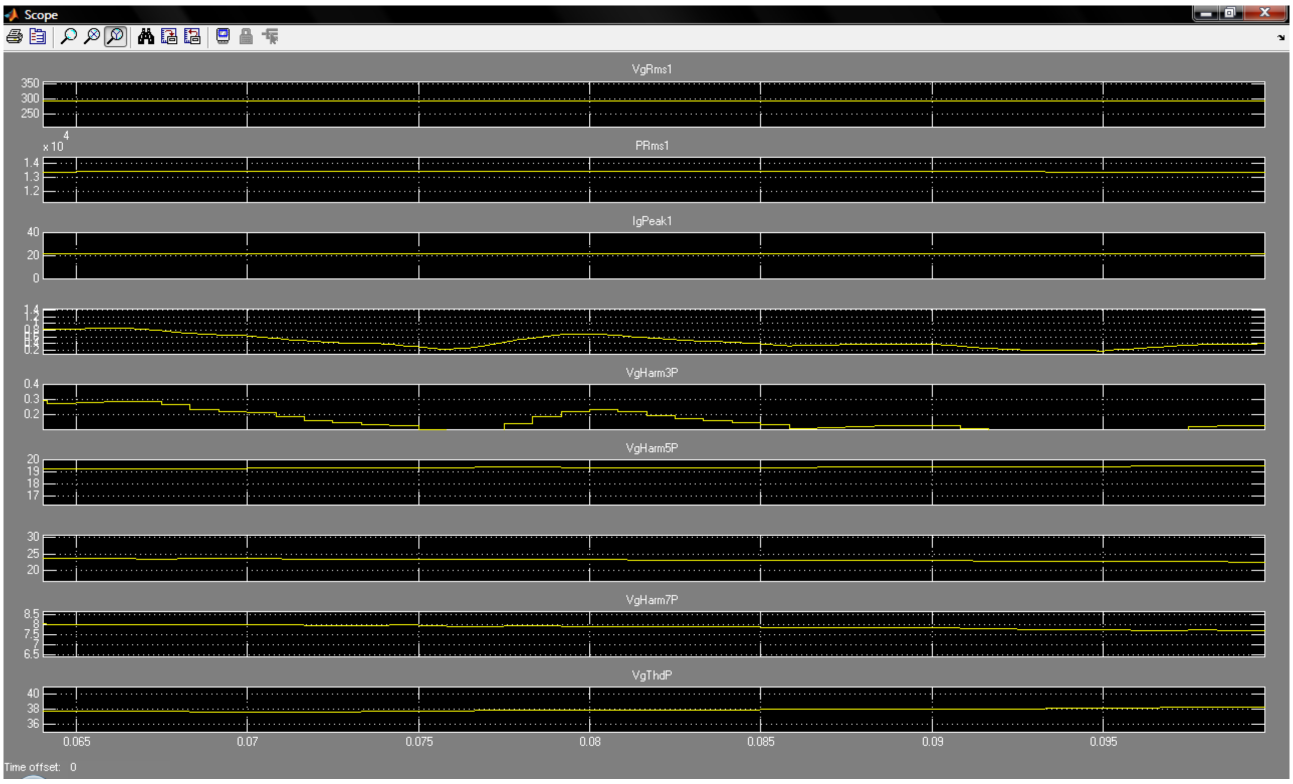


Figure 43 Current Harmonics for Level Shifted Modulation POD at 1200 Hz

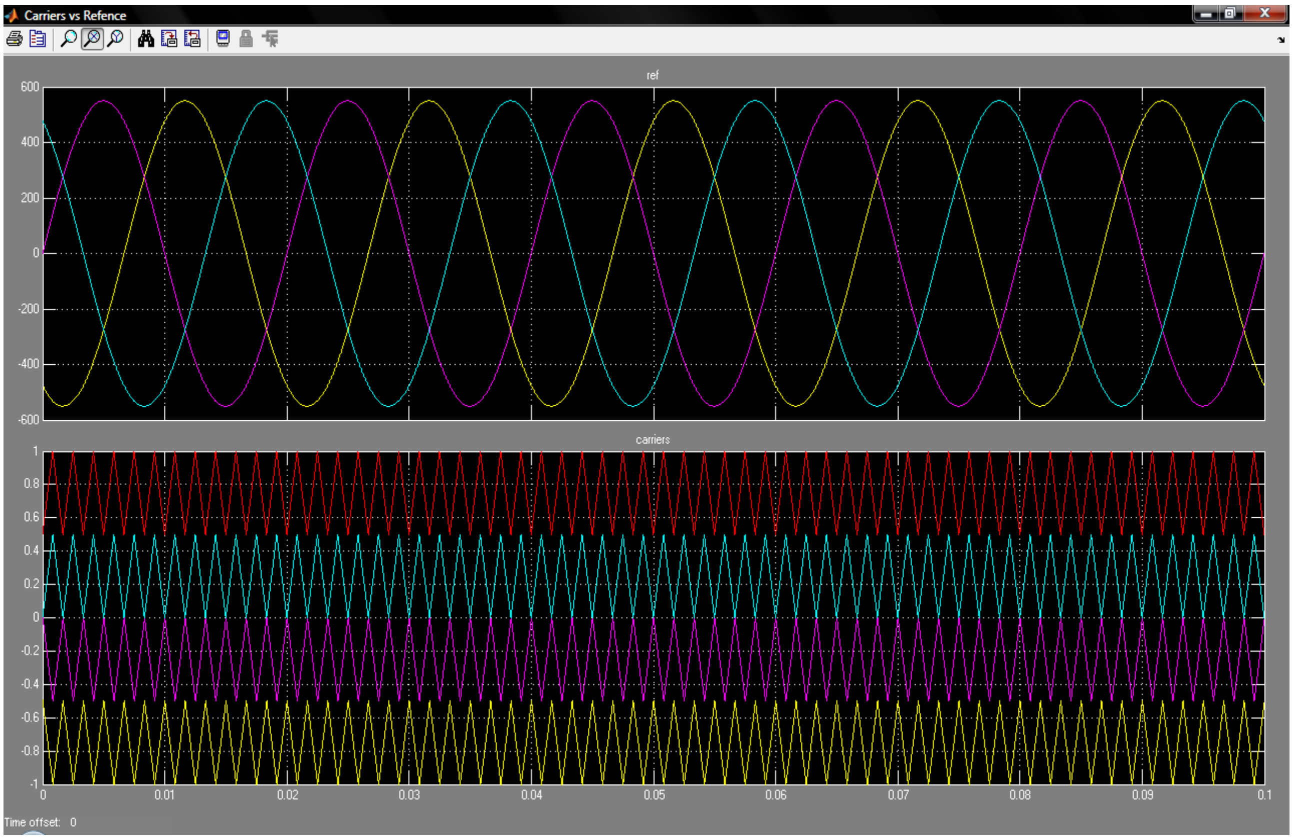


Figure 44 Carriers for Level Shifted Modulation POD for 1200 Hz

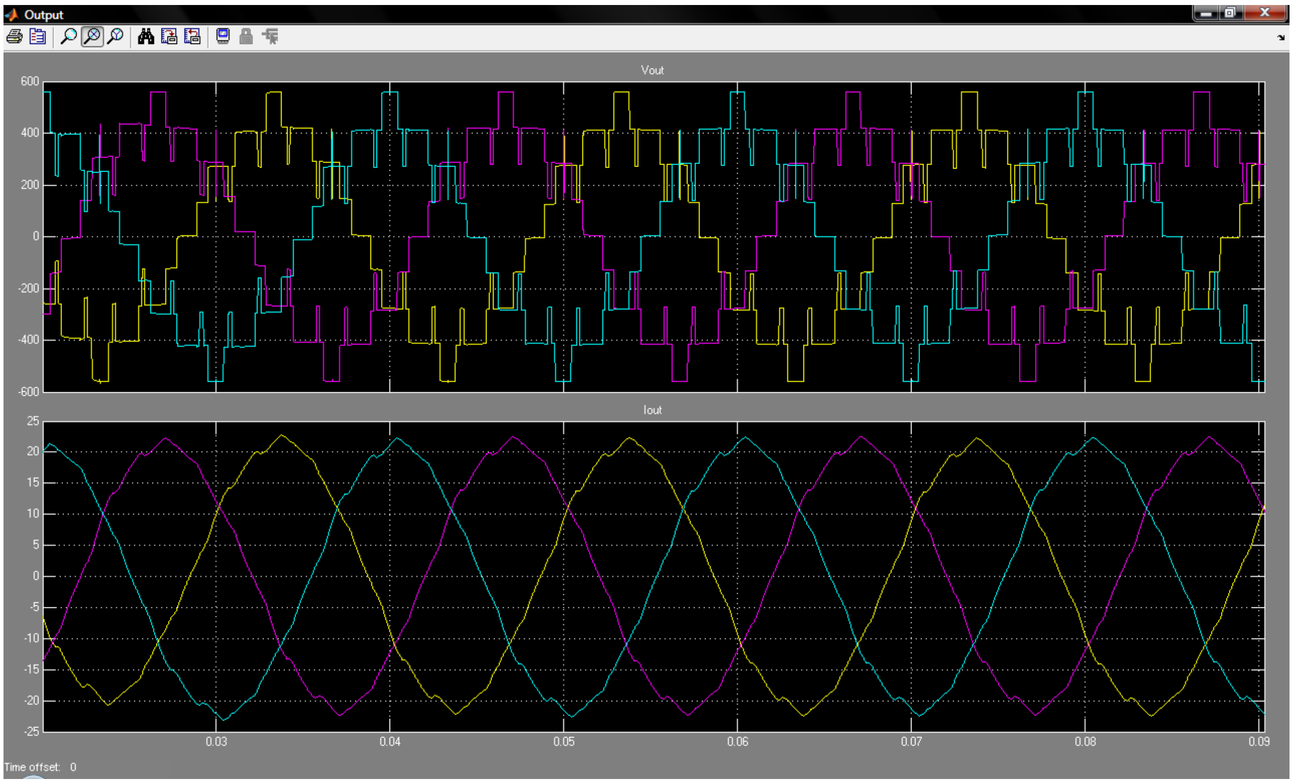


Figure 45 Output voltage and current for Level Shifted Modulation POD at 2400 Hz

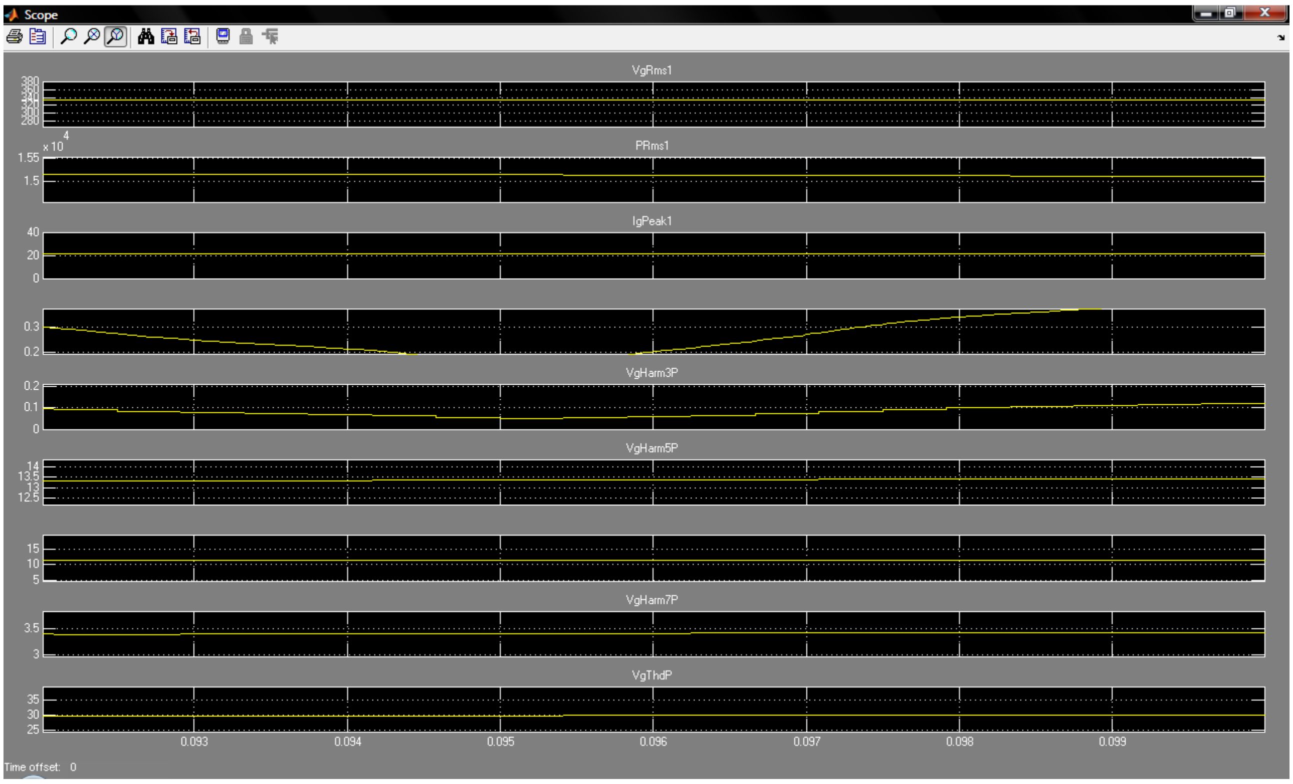


Figure 46 Voltage Harmonics for Level Shifted Modulation POD at 2400 Hz

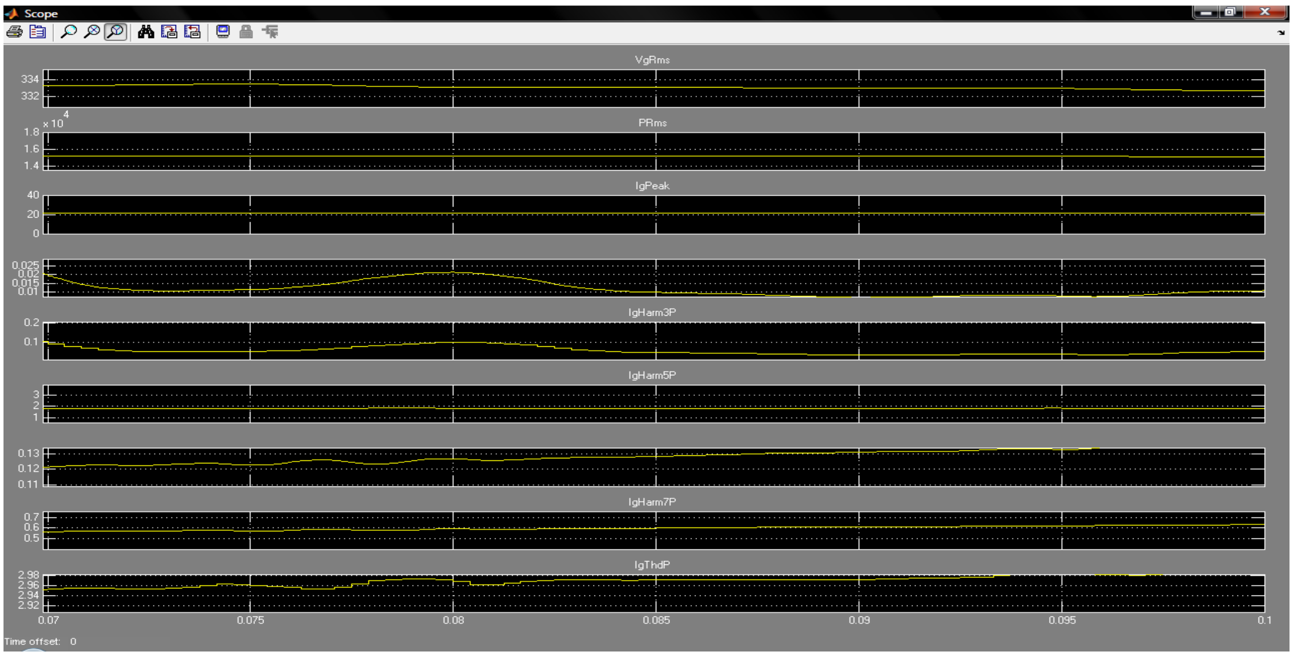


Figure 47 Current Harmonics for Level Shifted Modulation POD at 2400 Hz

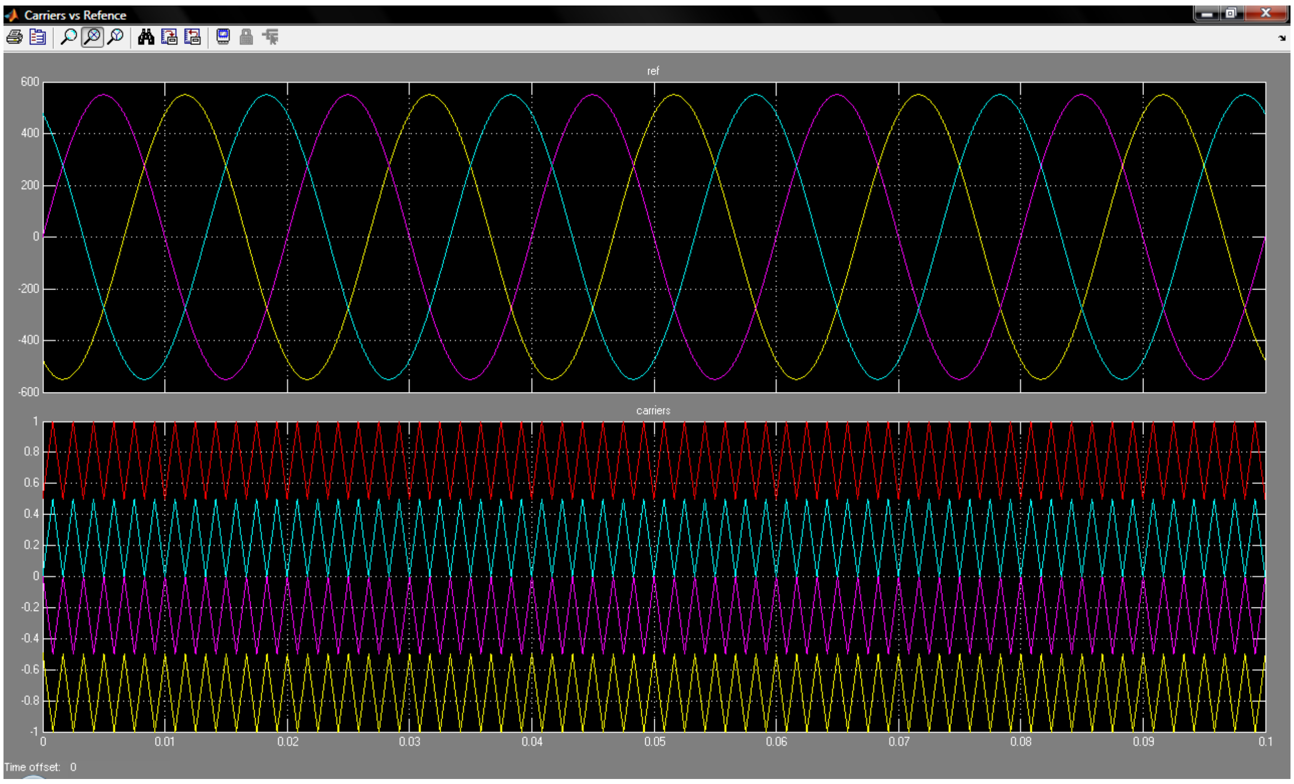


Figure 48 Carriers for Level Shifted Modulation POD for 2400 Hz

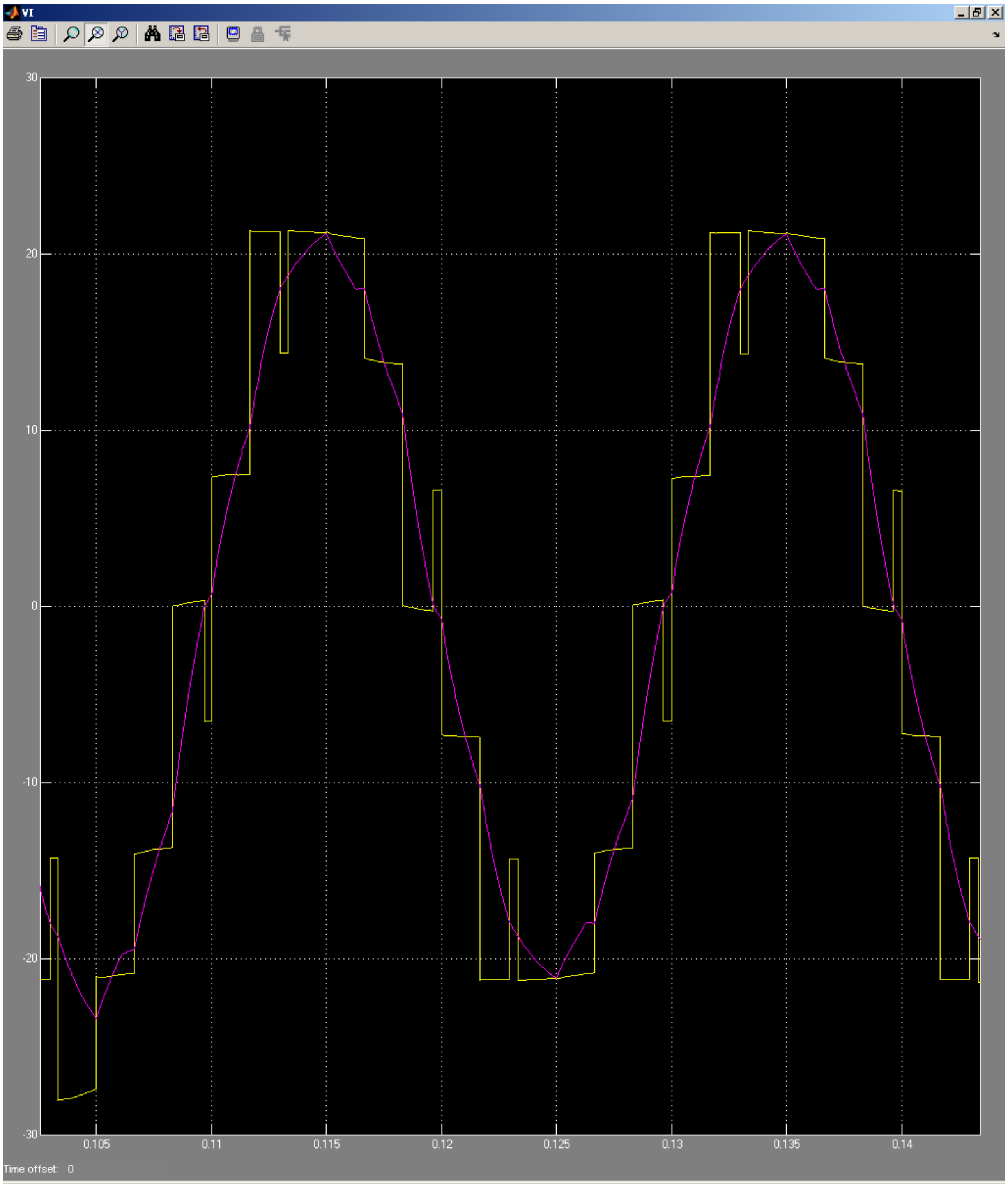


Figure 49 Output voltage and current for Staircase Modulation at 600 Hz

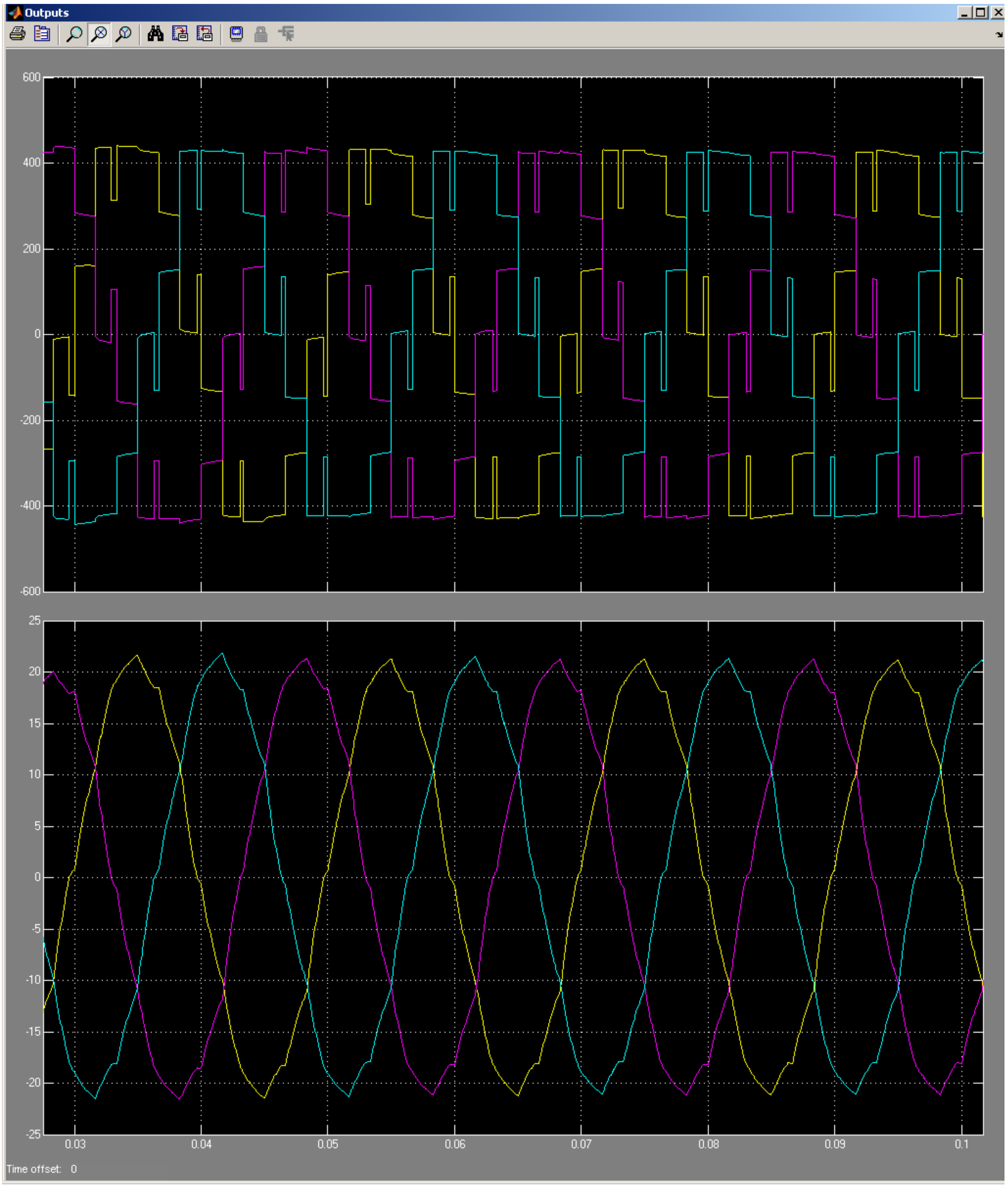


Figure 50 Voltage Harmonics for Staircase Modulation at 600 Hz

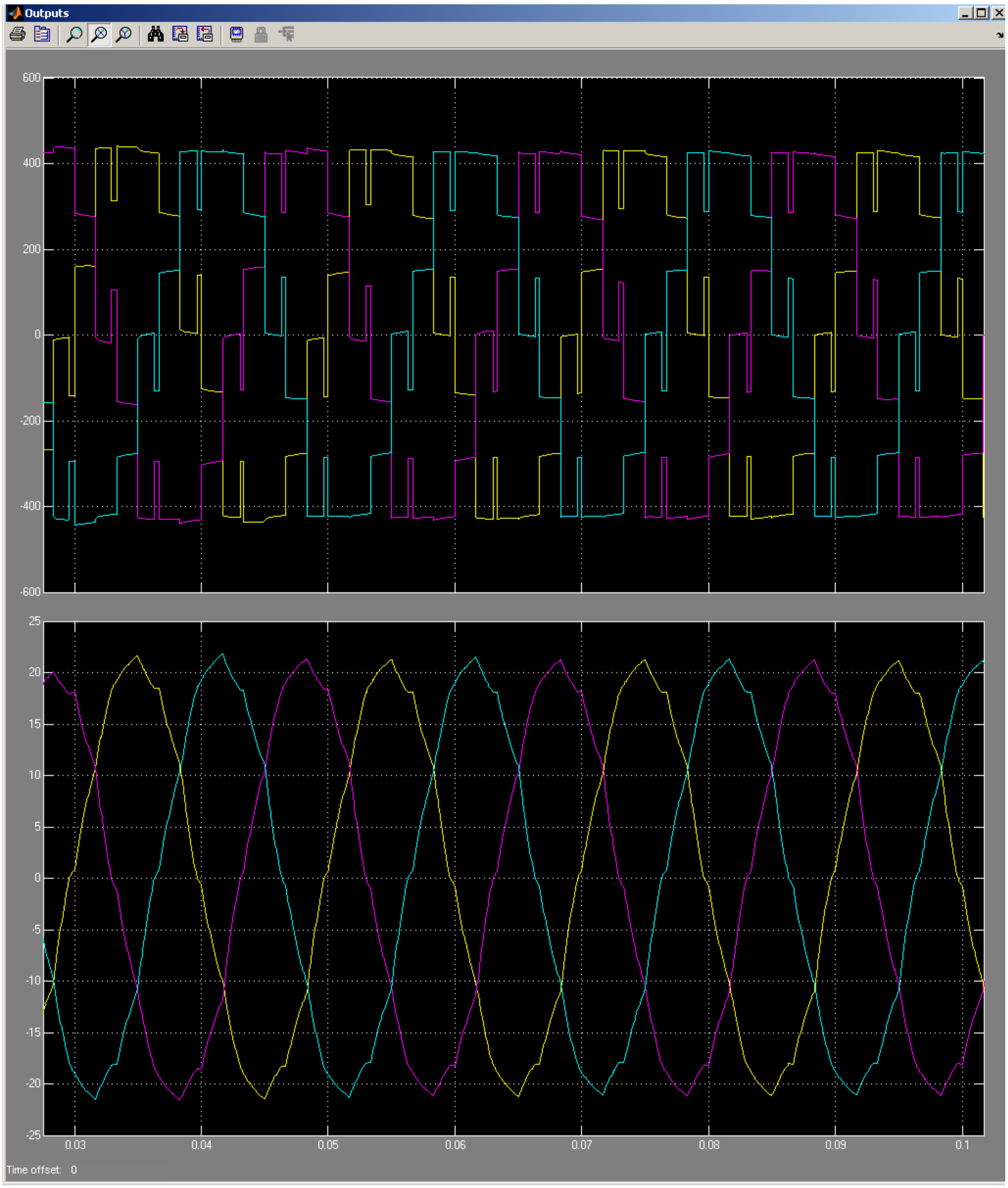


Figure 51 Current Harmonics for Staircase Modulation at 600 Hz

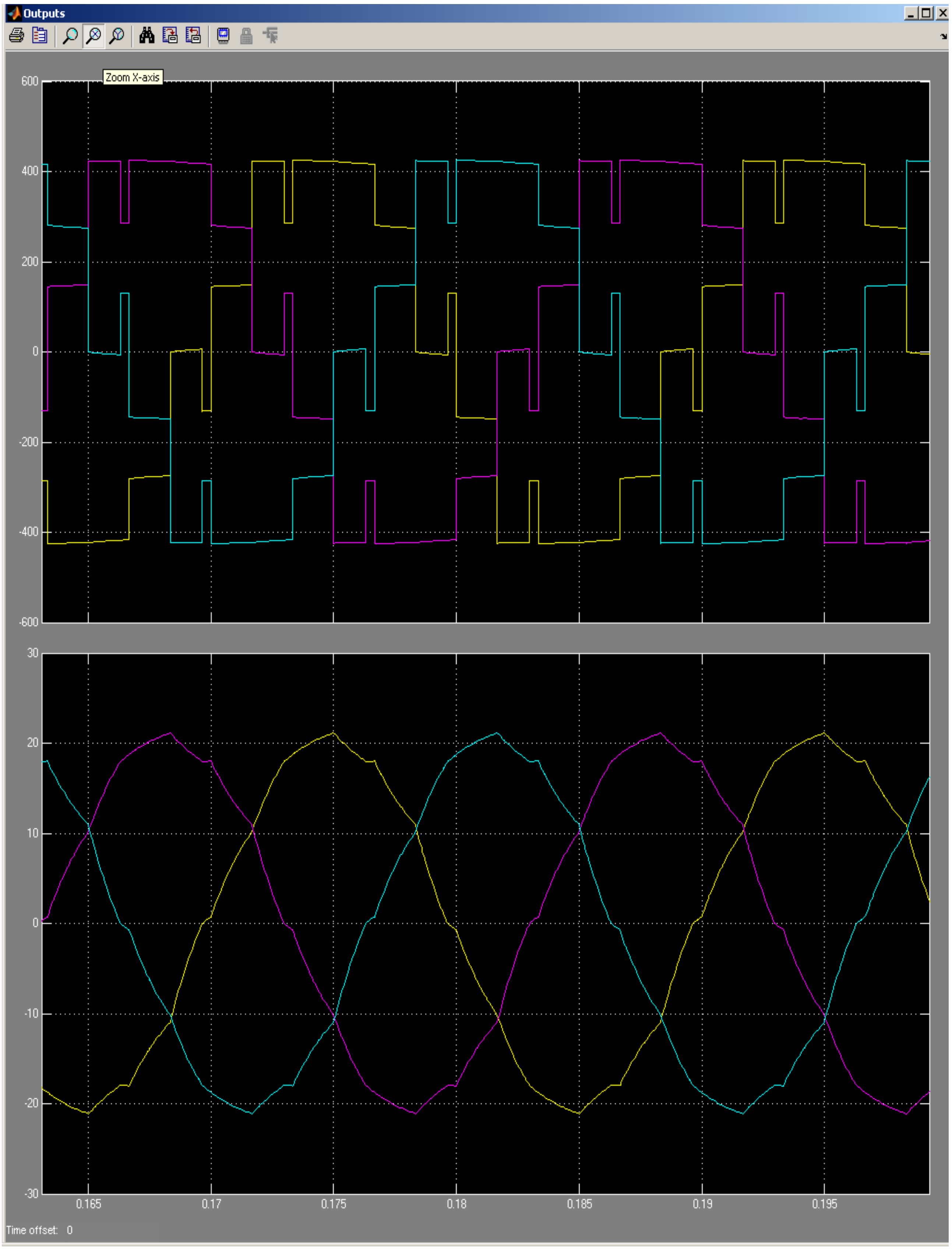


Figure 52 Output voltage and current for Staircase Modulation at 1200 Hz

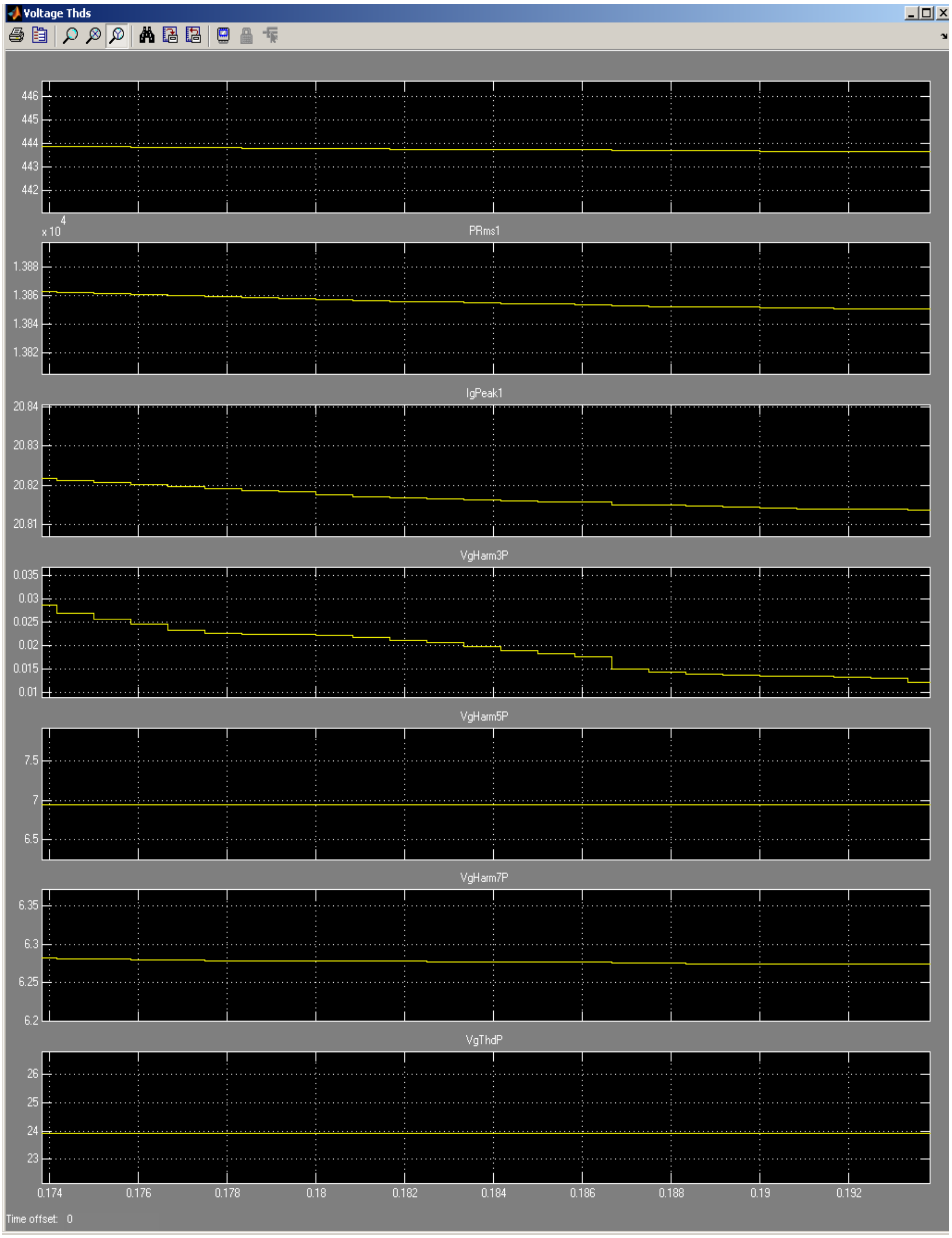


Figure 53 Voltage Harmonics for Staircase Modulation at 1200 Hz

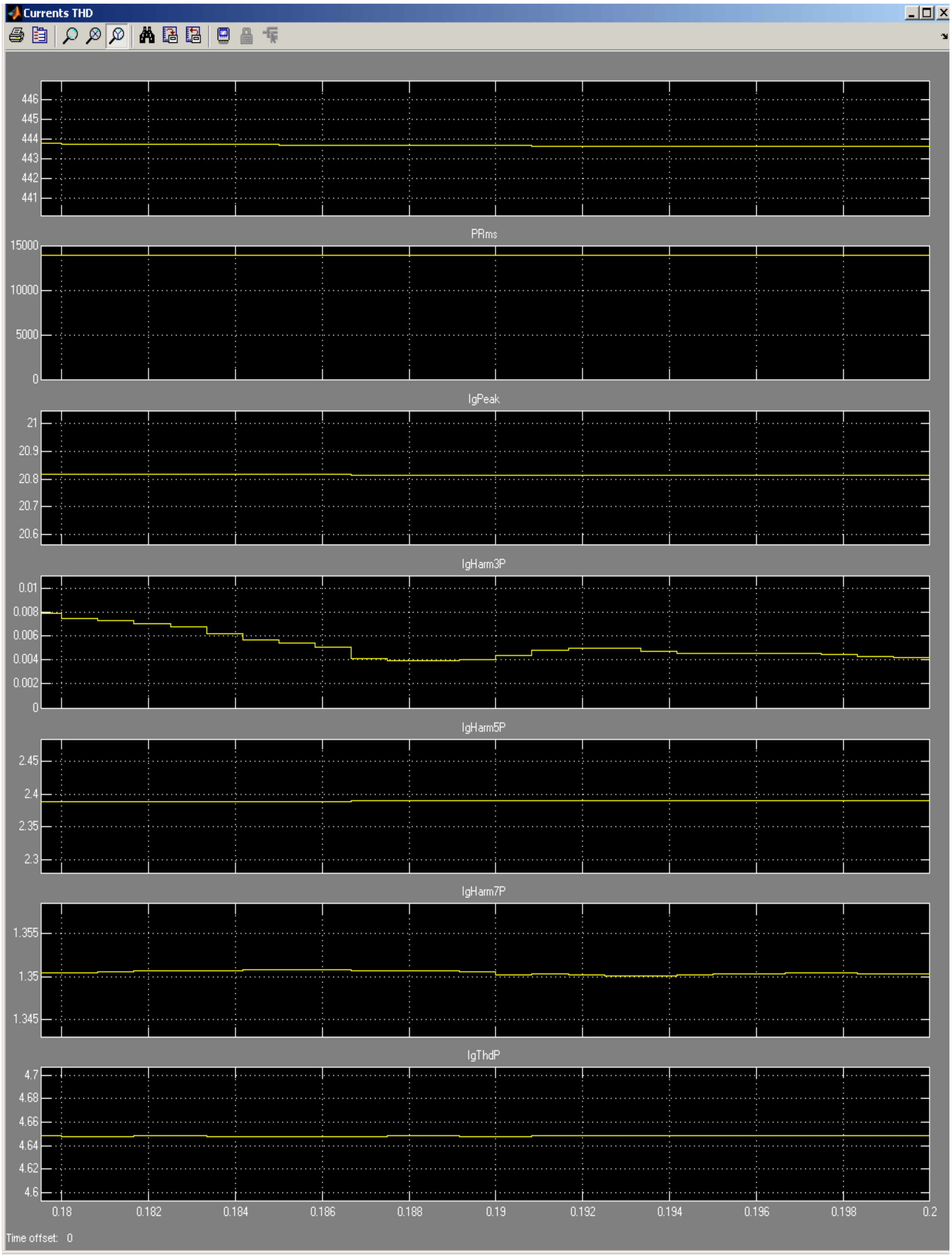


Figure 54 Current Harmonics for Staircase Modulation at 1200 Hz

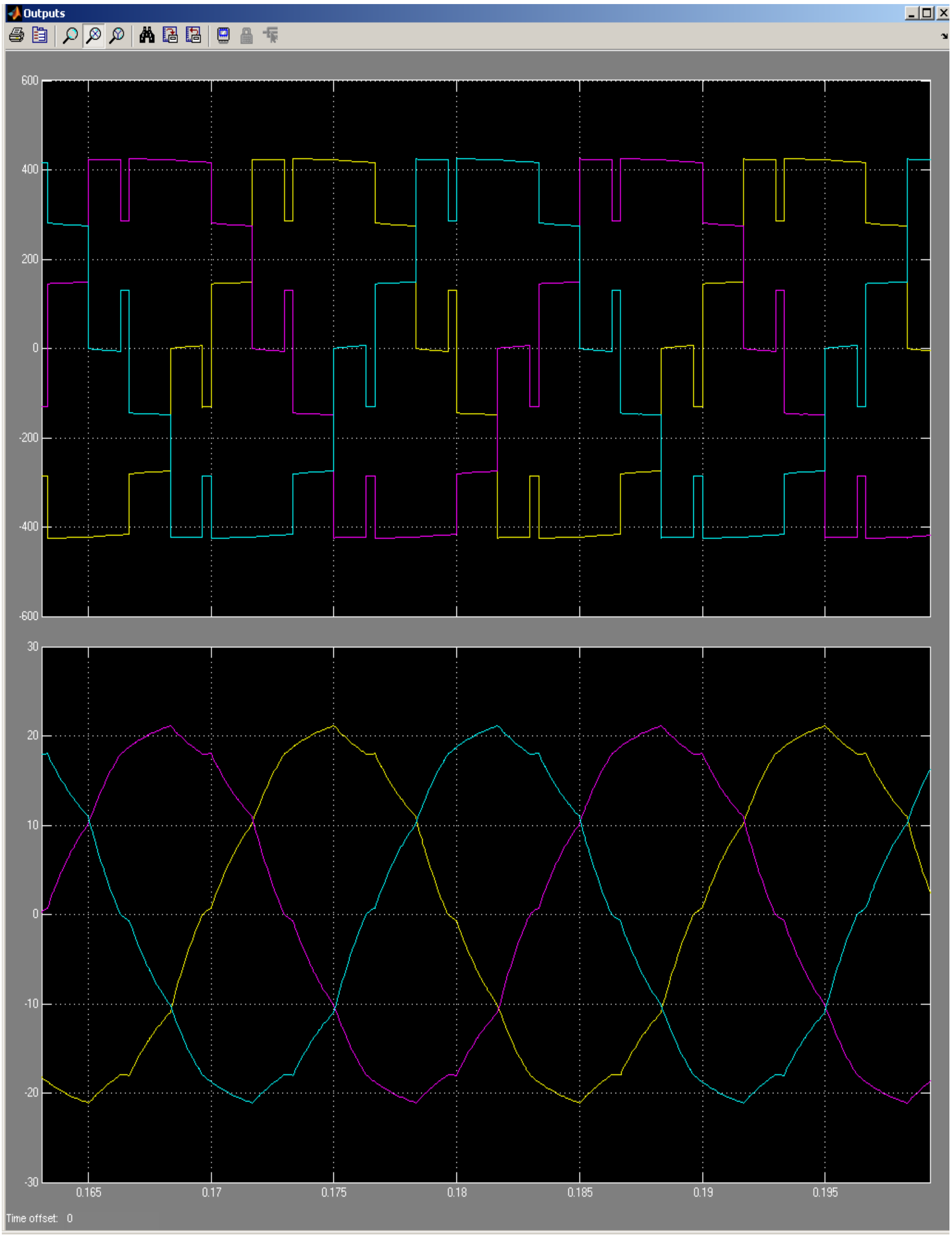


Figure 55 Output voltage and current for Staircase Modulation at 2400 Hz

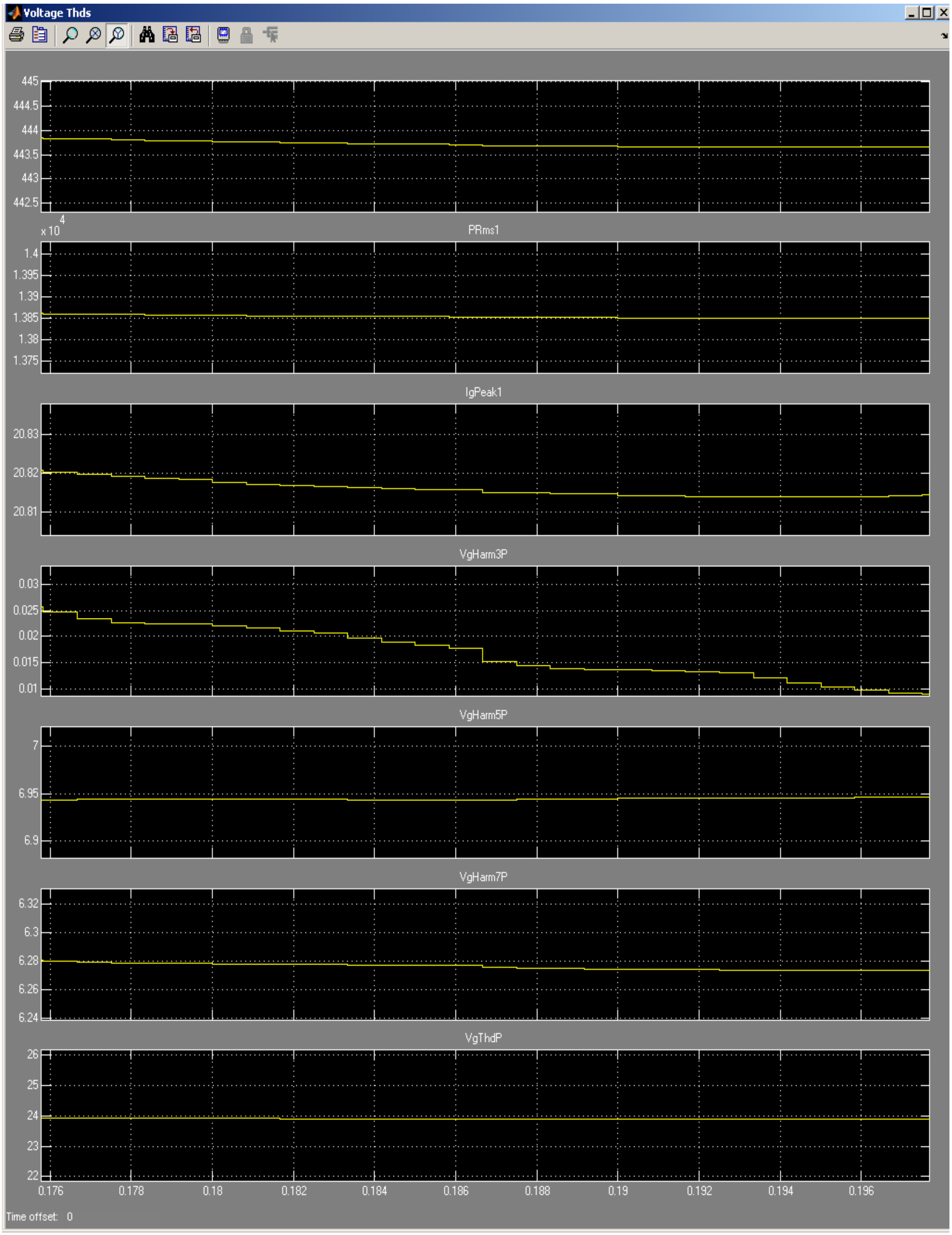


Figure 56 Voltage Harmonics for Staircase Modulation at 2400 Hz

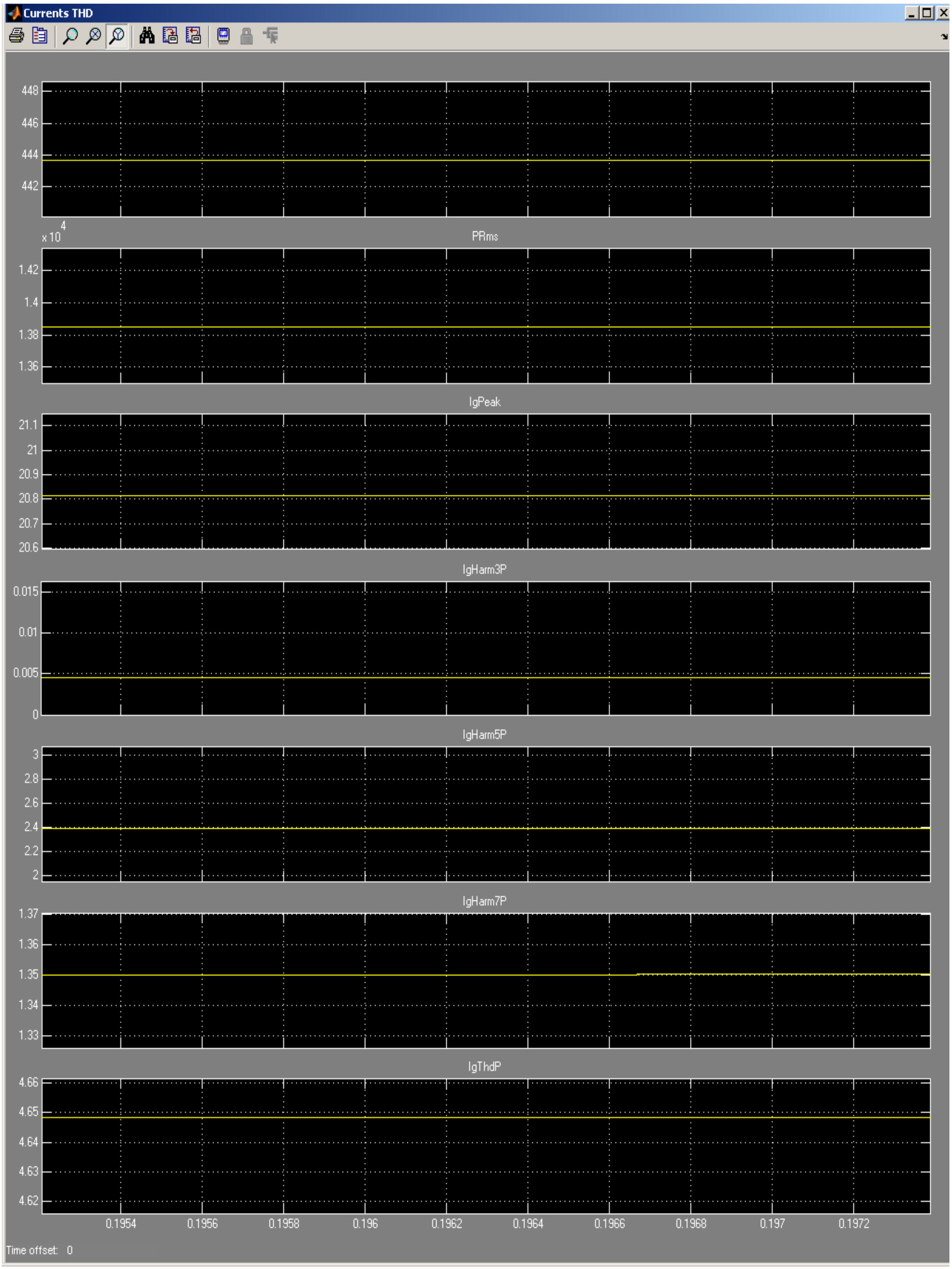


Figure 57 Current Harmonics for Staircase Modulation at 2400 Hz

Modular Multi-Level Inverter

Part 1: Unit Design

Section A: High power module

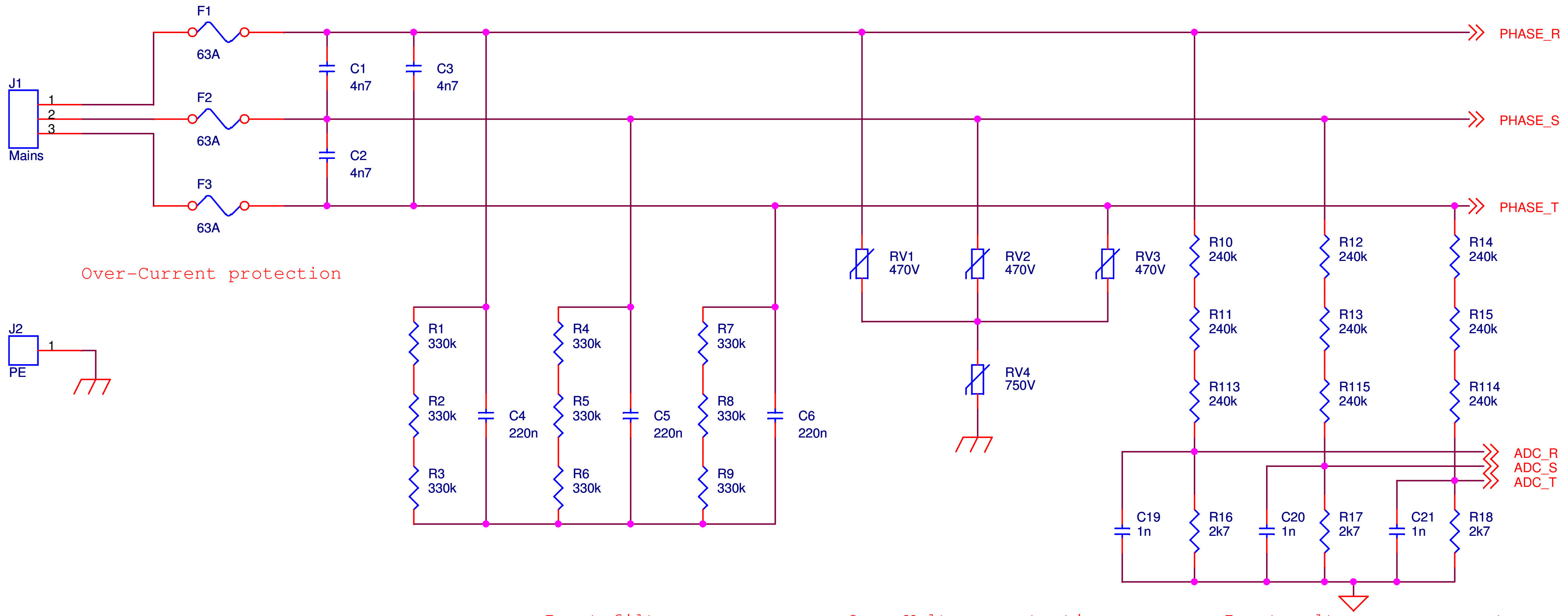
Master Thesis

Authors:
Cristian Sandu
Nicoleta Carnu
Valentin Costea

Supervisor:
Stig Munk Nielsen

Aalborg University 2009

Title <Title>		
Size A4	Document Number <Doc>	Rev 1
Date: Thursday, March 12, 2009	Sheet 1 of 1	



Input filter

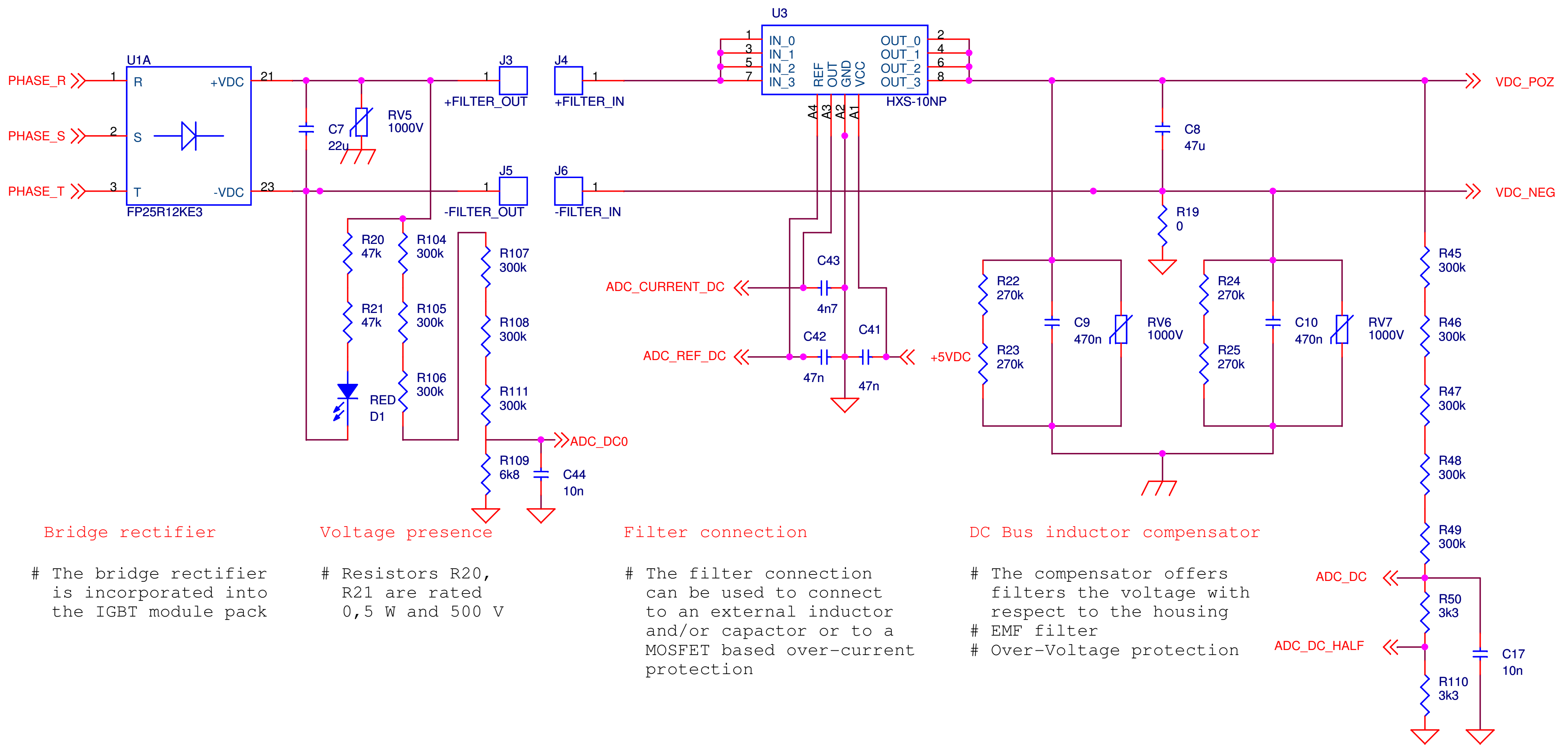
Over-Voltage protection

Input voltage measurement

# Resistors R1-R7 are wire resistors
  rated 250 V; 0,6 W, 5 % tollerance
# Each group resistor will be placed
  in series under the coresponding
  capacitor
# The capactors must be rated 600 V

# Resistors R10-R15 are wire resistors
  rated 350 V; 0,6 W, 1 % tollerance
  or less
# Resistor R16-R18 are SMD rated 250 V;
  0,5 W, 1 % tollerance or less
# Resistors R16-R18 must be place as
  close as possible to the ADC, while
  resistors R10-R15 must be place near
  the bridge input
# The resistor are only rated 0,6 W
  because the voltage has zero
  crossing so time to cool-down
  exists

Title <Title>		
Size A4	Document Number <Doc>	Rev <RevCo
Date: Sunday, March 15, 2009	Sheet 1 of 1	



Bridge rectifier

# The bridge rectifier
  is incorporated into
  the IGBT module pack

Voltage presence

# Resistors R20,
  R21 are rated
  0,5 W and 500 V

Filter connection

# The filter connection
  can be used to connect
  to an external inductor
  and/or capactor or to a
  MOSFET based over-current
  protection

DC Bus inductor compensator

# The compensator offers
  filters the voltage with
  respect to the housing
# EMF filter
# Over-Voltage protection

Inductance compesator

# C7 capacitor is rated
  1000 V and MUST be
  placed as close as
  possible to the output
  of the rectifier
# The role is to eliminate
  the track impedance

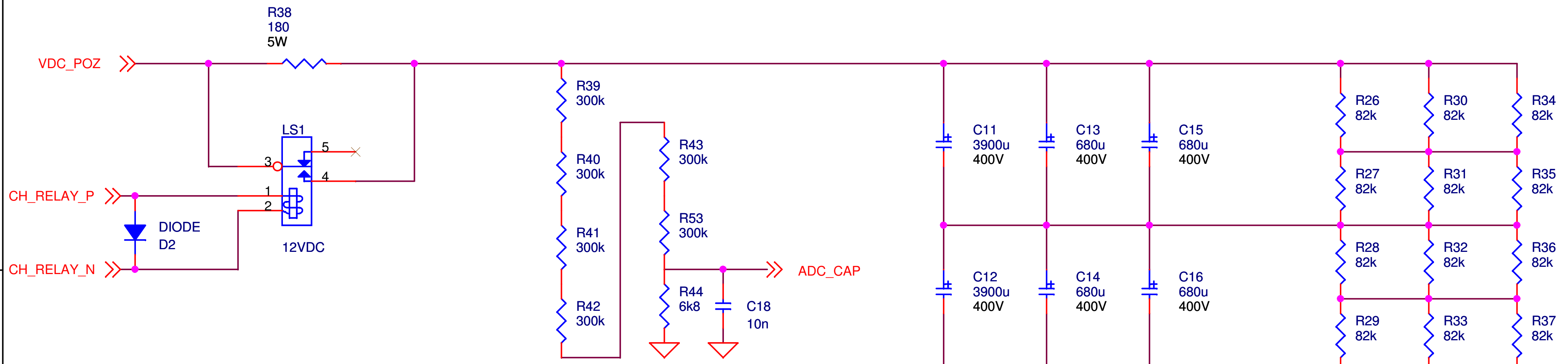
Control connection

# R19 is rated 1 W and
  is used to connect the
  control section to the
  negative voltage line

Over-Voltage protection

# RV5 is used to protect
  the pozitive line from
  the rectifier to
  overvoltages

Title <Title>		
Size A4	Document Number <Doc>	Rev <RevCo
Date: Sunday, March 15, 2009	Sheet 1 of 1	



DC Charging control

```
# The relay must handle
  full current rating
# Resistor R38 is rated
  10 W, 500 V and should
  be cauted with a termic
  insulation material
```

Capacitors voltage measurement

```
# The resistors R39-R43 are
  rated 1 W while R44 is rated
  0,6 W and all are 1% tolerance
# The resistor R44 must be
  placed as close as possible
  to the ADC input pin or
  filter input
```

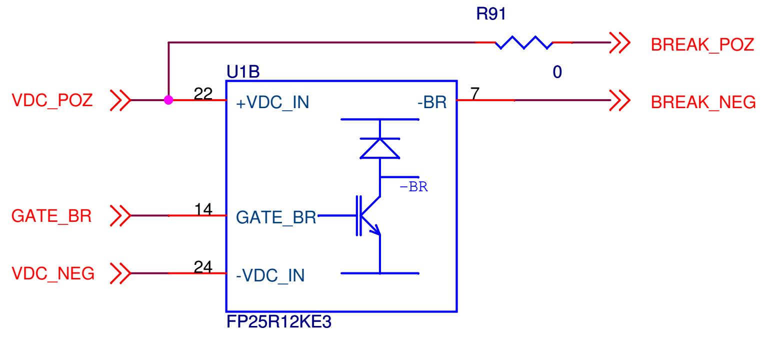
DC Capacitors

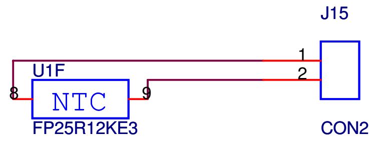
```
# The capacitors are
  mounted on the PCB
  in a compact configuration
  so that the arrea where
  the capacitors are placed
  is kept to a minimum
# The equivalent capacitance
  is 1012 uF
```

Discharge/Balance resistors

```
# The resistors have the
  role to assure a 5 min.
  discharge time
# The voltage between
  capacitors is also
  maintained
# Resistors are rated 1W
  250 V
```

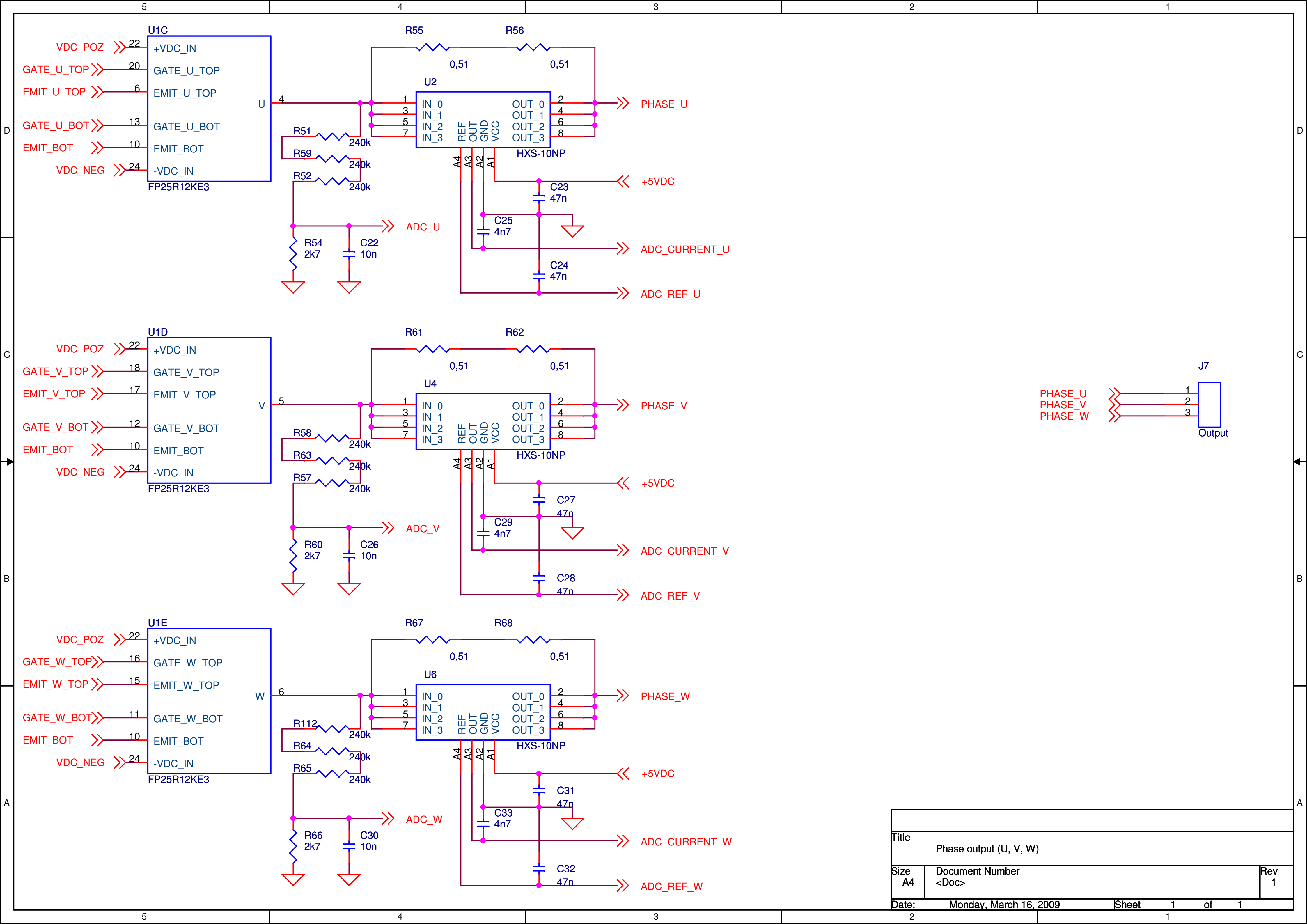
Title	<Title>	
Size A4	Document Number <Doc>	Rev 1
Date:	Sunday, March 15, 2009	Sheet 1 of 1



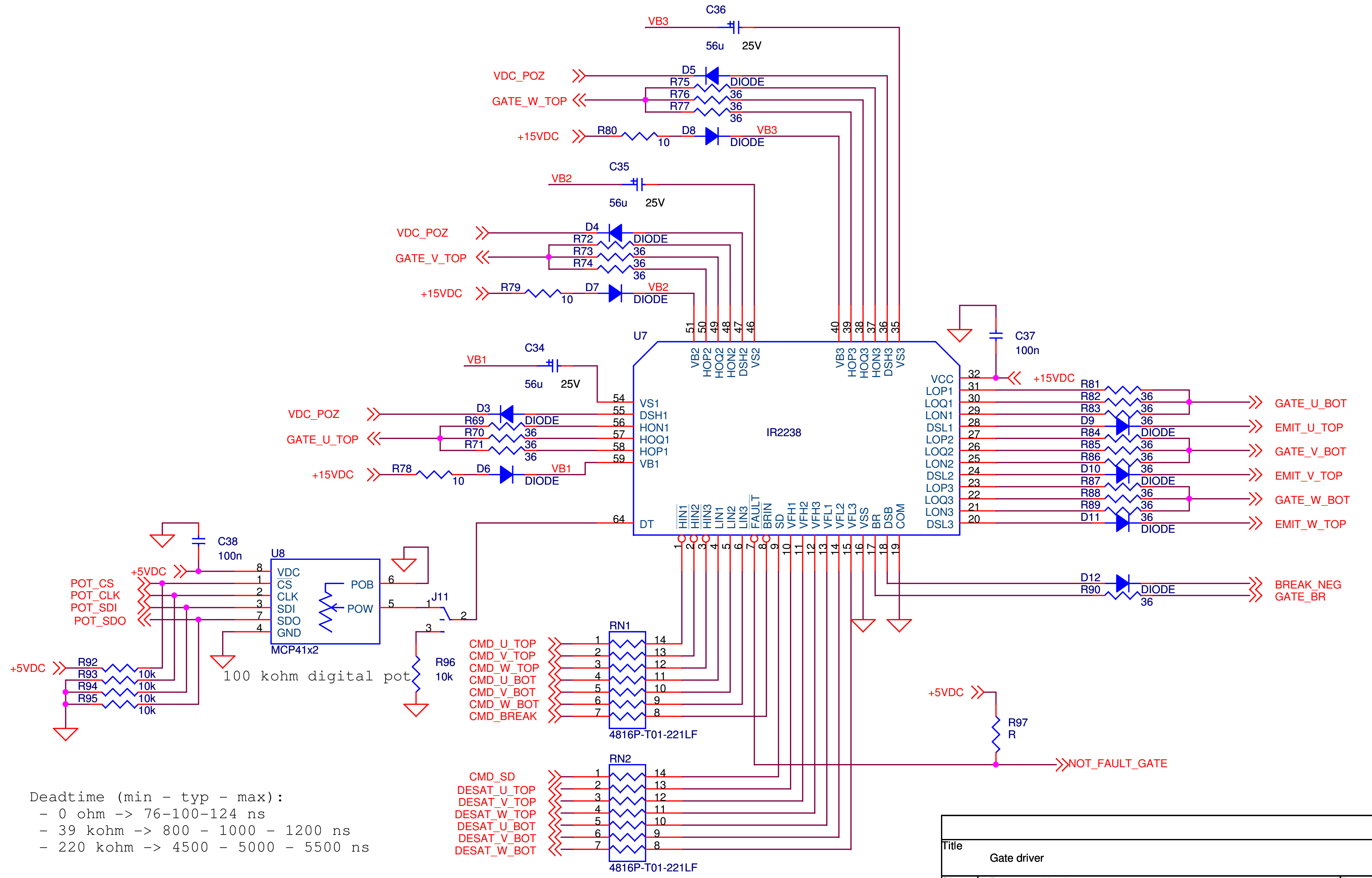


NTC Connector

Title	Break chopper and NTC Connector	
Size A4	Document Number <Doc>	Rev 1
Date:	Thursday, March 12, 2009	Sheet 1 of 1



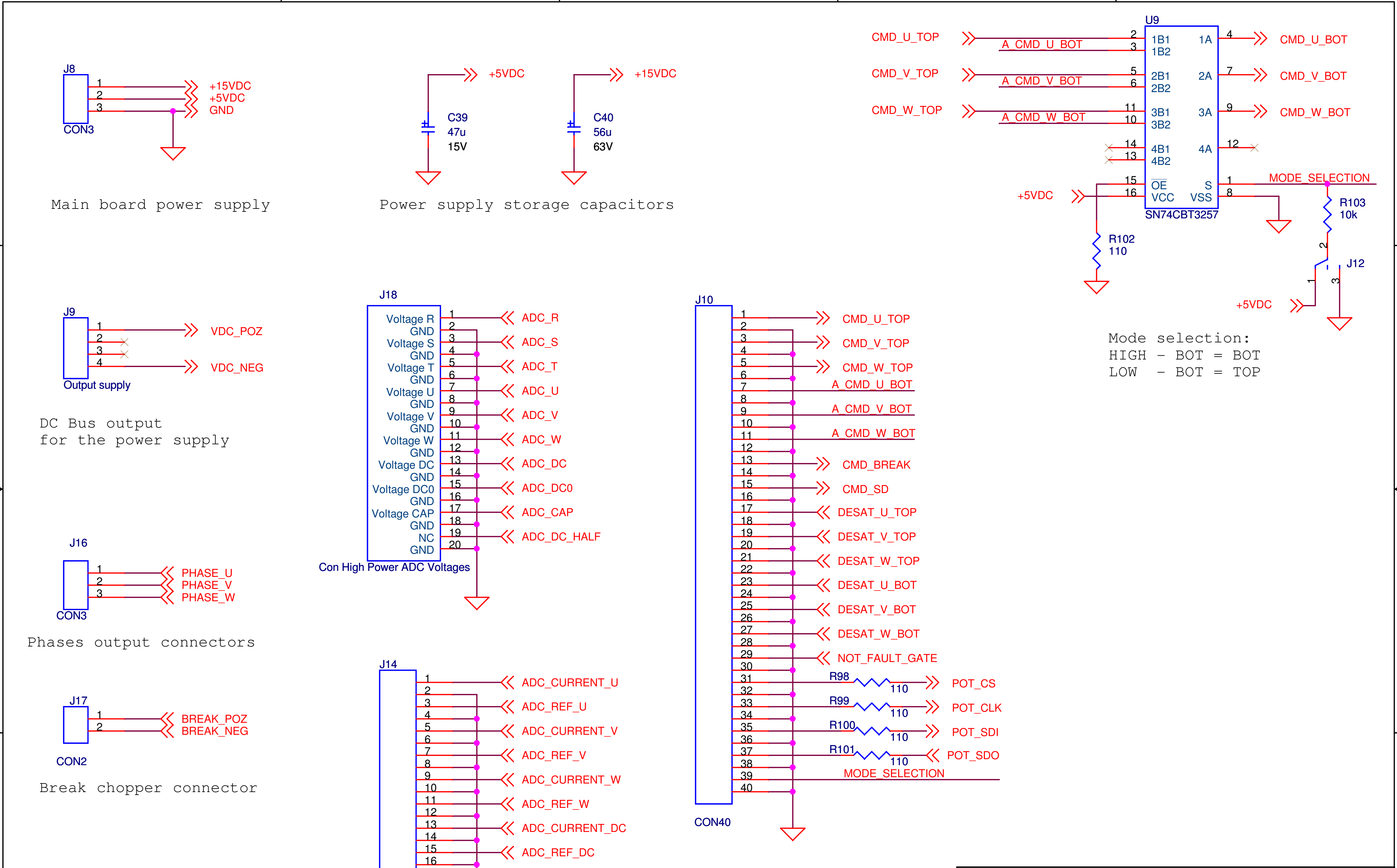
Title	Phase output (U, V, W)	
Size A4	Document Number <Doc>	Rev 1
Date: Monday, March 16, 2009	Sheet 1 of 1	



```
Deadtime (min - typ - max):
 - 0 ohm -> 76-100-124 ns
 - 39 kohm -> 800 - 1000 - 1200 ns
 - 220 kohm -> 4500 - 5000 - 5500 ns
```

RN1, RN2: 220 ohms resistor array

Title	Gate driver	
Size A4	Document Number <Doc>	Rev 1
Date: Thursday, March 12, 2009	Sheet 1 of 1	



Title	<Title>	
Size A4	Document Number <Doc>	Rev 1
Date:	Friday, March 13, 2009	Sheet 1 of 1

Modular Multi-Level Inverter

Part 1: Unit Design

Section B: Command module

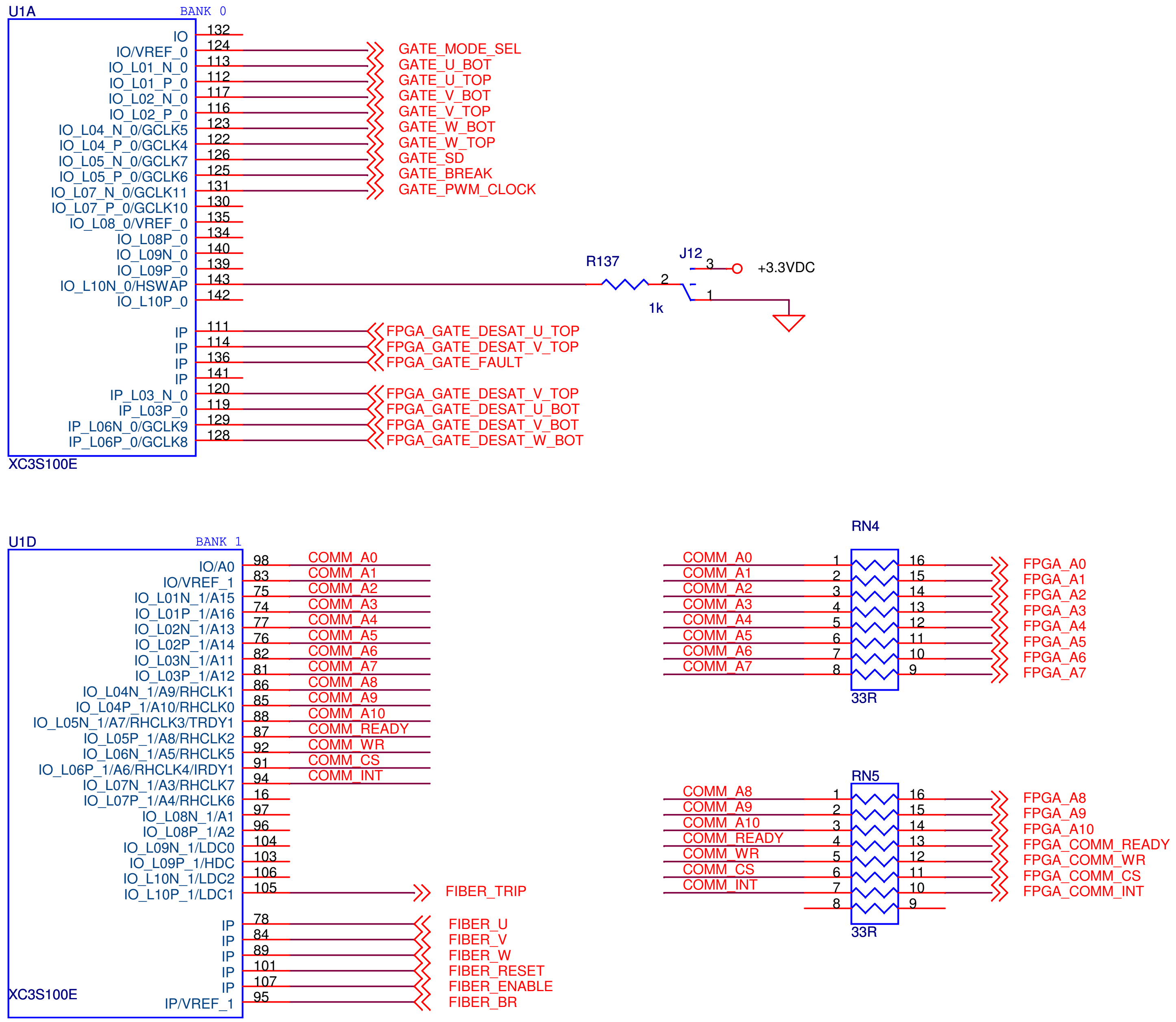
Master Thesis

Authors:
Cristian Sandu
Nicoleta Carnu
Valentin Costea

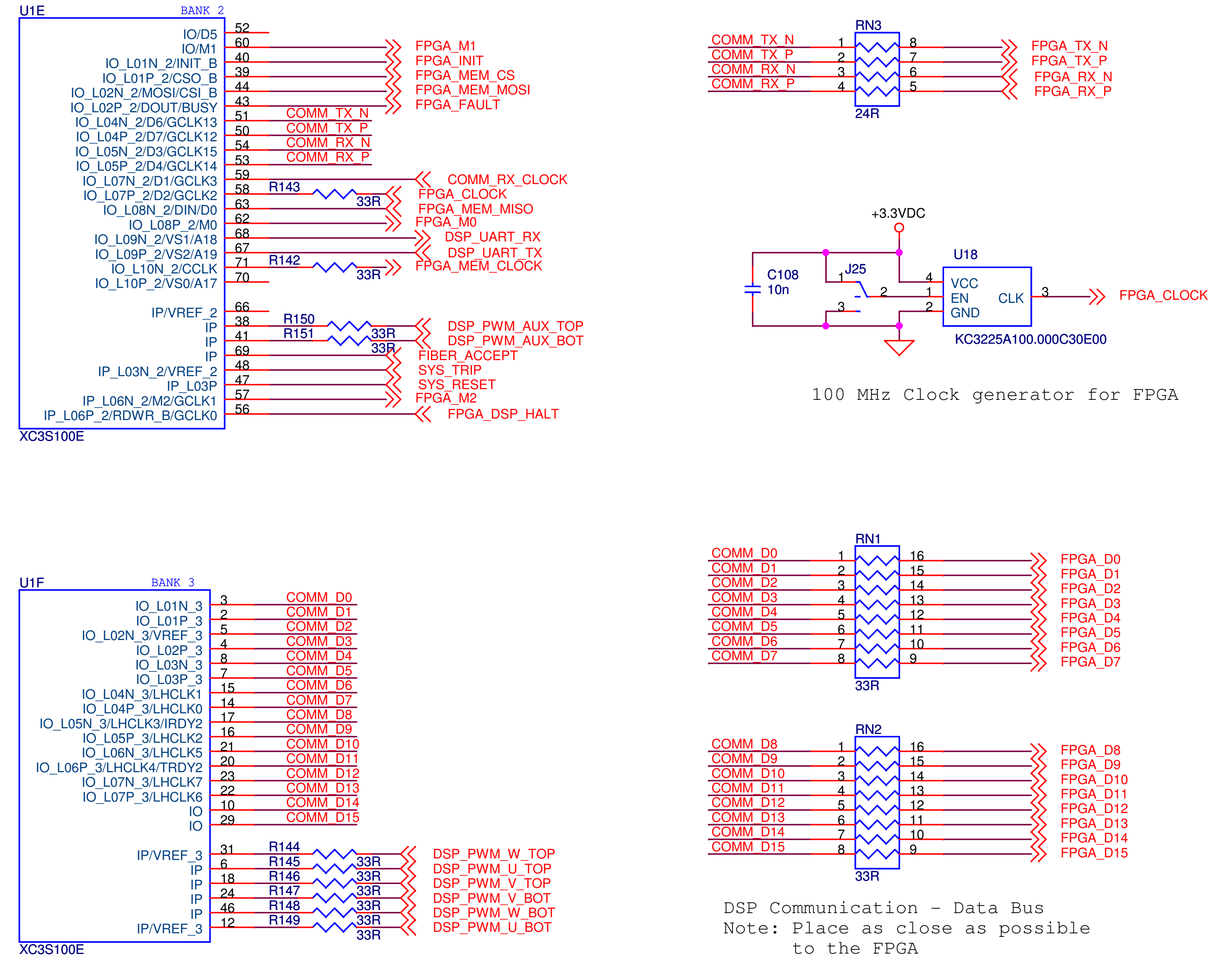
Supervisor:
Stig Munk Nielsen

Aalborg University 2009

Title	Introduction	
Size A4	Document Number <Doc>	Rev 1
Date: Sunday, March 15, 2009	Sheet 1 of 14	



Title	FPGA - Bank 0, 1	
Size A4	Document Number <Doc>	Rev 1
Date: Sunday, March 15, 2009	Sheet 2 of 14	

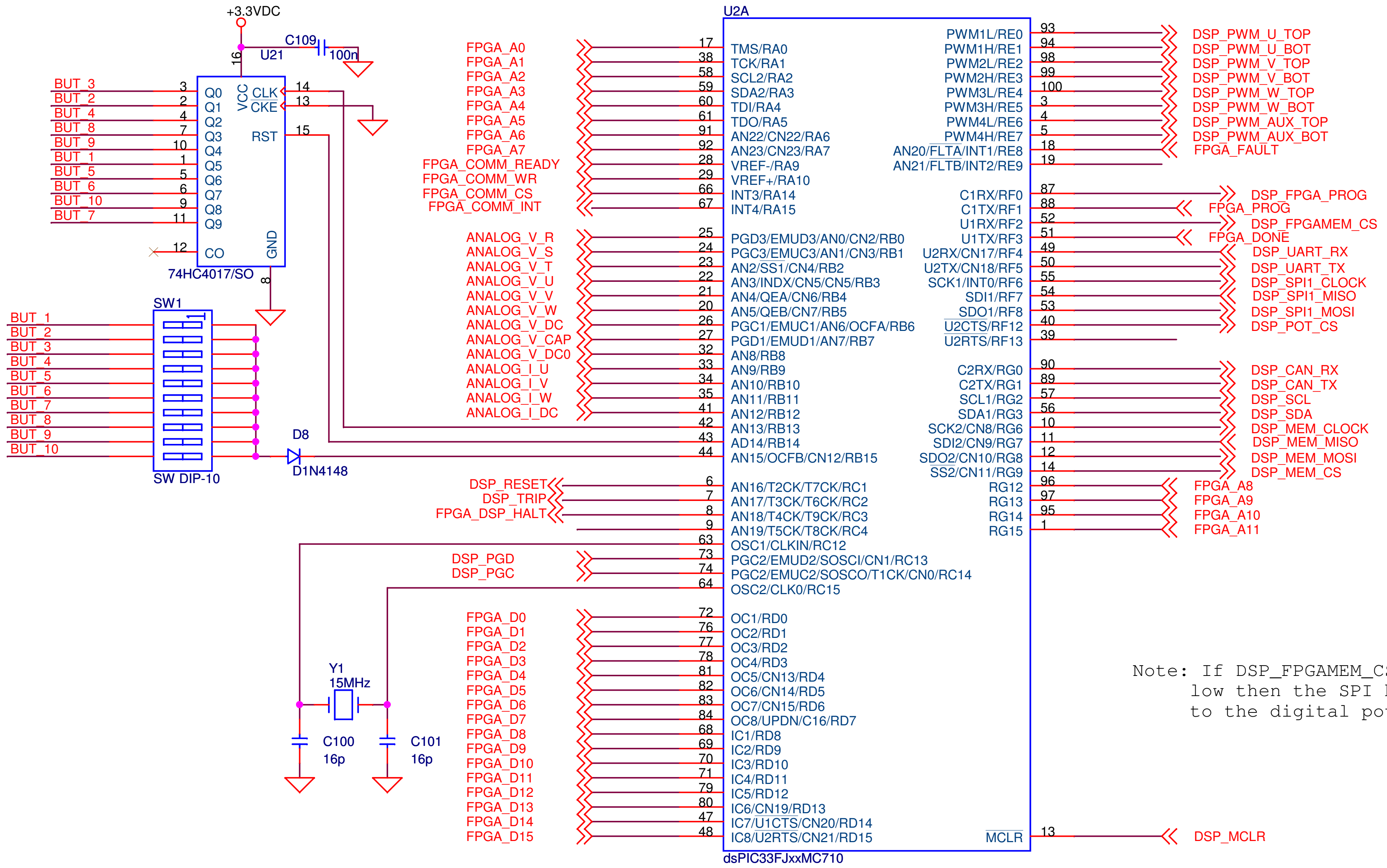


100 MHz Clock generator for FPGA

DSP Communication - Data Bus
Note: Place as close as possible
to the FPGA

DSP PWM Signal Inputs

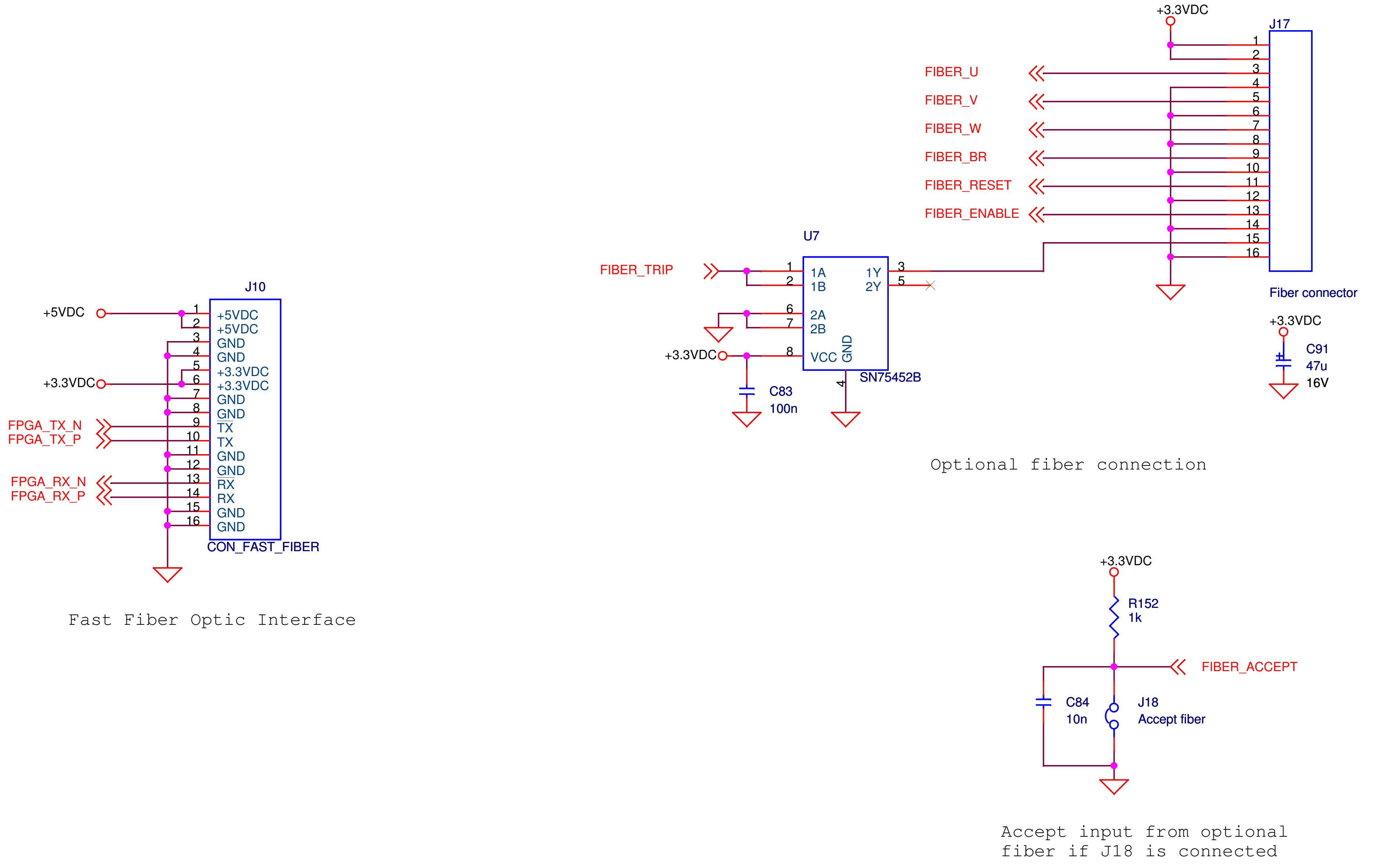
Title	FPGA - Bank 2, 3	
Size A4	Document Number <Doc>	Rev 1
Date: Sunday, March 15, 2009	Sheet 3 of 14	



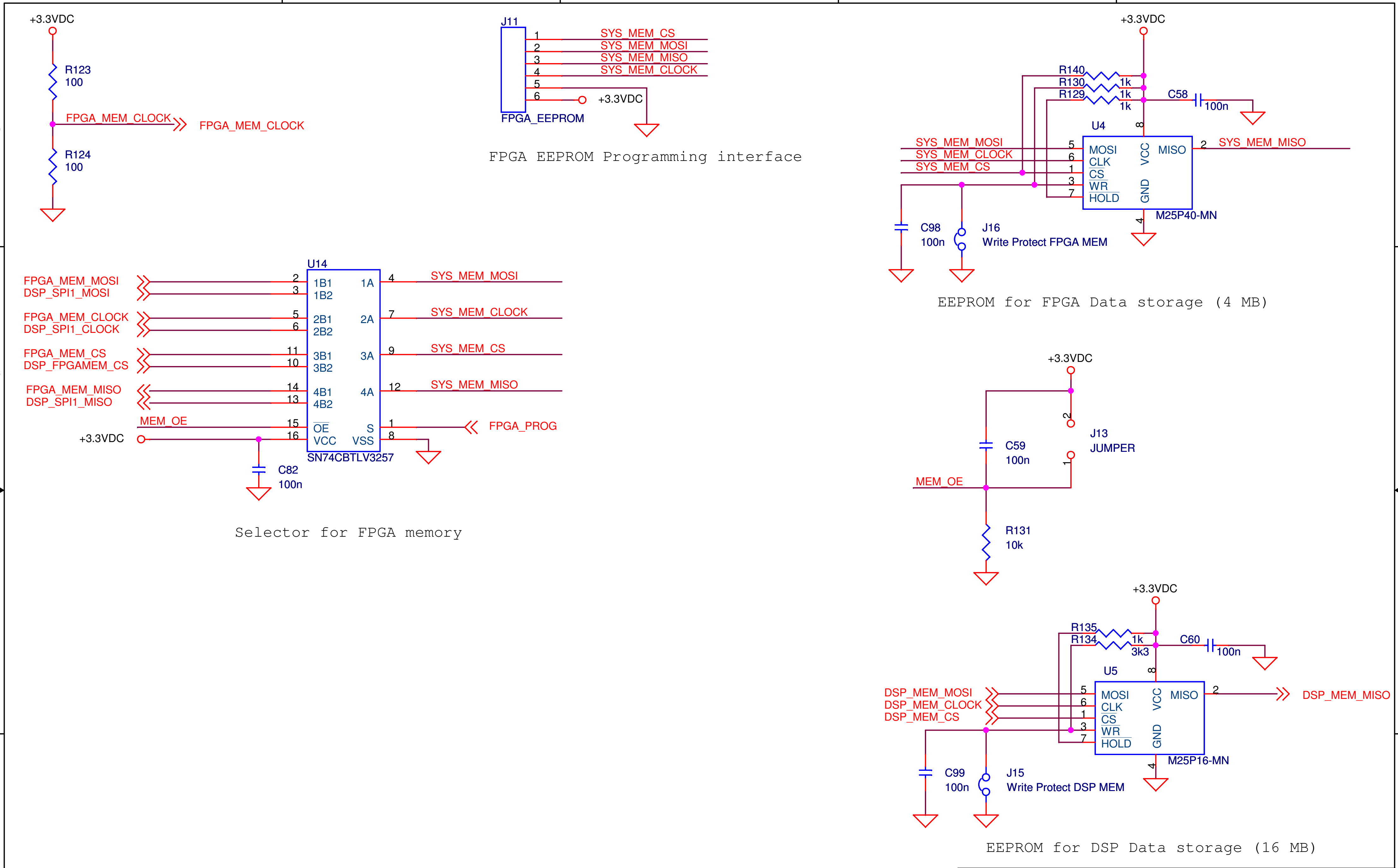
+3.3VDC
C109
100n
U21
16
VCC
CLK 14
CKE 13
RST 15
GND 8
74HC4017/SO
BUT_3 3 Q0
BUT_2 2 Q1
BUT_4 4 Q2
BUT_8 7 Q3
BUT_9 10 Q4
BUT_1 1 Q5
BUT_5 5 Q6
BUT_6 6 Q7
BUT_10 9 Q8
BUT_7 11 Q9
12 CO
SW1
BUT_1
BUT_2
BUT_3
BUT_4
BUT_5
BUT_6
BUT_7
BUT_8
BUT_9
BUT_10
SW DIP-10
D8
D1N4148
Y1
15MHz
C100
16p
C101
16p
U2A
dsPIC33FJxxMC710
FPGA_A0 17 TMS/RA0
FPGA_A1 38 TCK/RA1
FPGA_A2 58 SCL2/RA2
FPGA_A3 59 SDA2/RA3
FPGA_A4 60 TDI/RA4
FPGA_A5 61 TDO/RA5
FPGA_A6 91 AN22/CN22/RA6
FPGA_A7 92 AN23/CN23/RA7
FPGA_COMM_READY 28 VREF-/RA9
FPGA_COMM_WR 29 VREF+/RA10
FPGA_COMM_CS 66 INT3/RA14
FPGA_COMM_INT 67 INT4/RA15
ANALOG_V_R 25 PGD3/EMUD3/AN0/CN2/RB0
ANALOG_V_S 24 PGC3/EMUC3/AN1/CN3/RB1
ANALOG_V_T 23 AN2/SS1/CN4/RB2
ANALOG_V_U 22 AN3/INDX/CN5/CN5/RB3
ANALOG_V_V 21 AN4/QEA/CN6/RB4
ANALOG_V_W 20 AN5/QEB/CN7/RB5
ANALOG_V_DC 26 PGC1/EMUC1/AN6/OCFA/RB6
ANALOG_V_CAP 27 PGD1/EMUD1/AN7/RB7
ANALOG_V_DC0 32 AN8/RB8
ANALOG_I_U 33 AN9/RB9
ANALOG_I_V 34 AN10/RB10
ANALOG_I_W 35 AN11/RB11
ANALOG_I_DC 41 AN12/RB12
42 AN13/RB13
43 AD14/RB14
44 AN15/OCFB/CN12/RB15
DSP_RESET 6 AN16/T2CK/T7CK/RC1
DSP_TRIP 7 AN17/T3CK/T6CK/RC2
FPGA_DSP_HALT 8 AN18/T4CK/T9CK/RC3
9 AN19/T5CK/T8CK/RC4
63 OSC1/CLKIN/RC12
DSP_PGD 73 PGC2/EMUD2/SOSCI/CN1/RC13
DSP_PGC 74 PGC2/EMUC2/SOSCO/T1CK/CN0/RC14
64 OSC2/CLK0/RC15
FPGA_D0 72 OC1/RD0
FPGA_D1 76 OC2/RD1
FPGA_D2 77 OC3/RD2
FPGA_D3 78 OC4/RD3
FPGA_D4 81 OC5/CN13/RD4
FPGA_D5 82 OC6/CN14/RD5
FPGA_D6 83 OC7/CN15/RD6
FPGA_D7 84 OC8/UPDN/C16/RD7
FPGA_D8 68 IC1/RD8
FPGA_D9 69 IC2/RD9
FPGA_D10 70 IC3/RD10
FPGA_D11 71 IC4/RD11
FPGA_D12 79 IC5/RD12
FPGA_D13 80 IC6/CN19/RD13
FPGA_D14 47 IC7/U1CTS/CN20/RD14
FPGA_D15 48 IC8/U2RTS/CN21/RD15
PWM1L/RE0 93 DSP_PWM_U_TOP
PWM1H/RE1 94 DSP_PWM_U_BOT
PWM2L/RE2 98 DSP_PWM_V_TOP
PWM2H/RE3 99 DSP_PWM_V_BOT
PWM3L/RE4 100 DSP_PWM_W_TOP
PWM3H/RE5 3 DSP_PWM_W_BOT
PWM4L/RE6 4 DSP_PWM_AUX_TOP
PWM4H/RE7 5 DSP_PWM_AUX_BOT
AN20/FLTA/INT1/RE8 18 FPGA_FAULT
AN21/FLTB/INT2/RE9 19
C1RX/RF0 87 DSP_FPGA_PROG
C1TX/RF1 88 FPGA_PROG
U1RX/RF2 52 DSP_FPGAMEM_CS
U1TX/RF3 51 FPGA_DONE
U2RX/CN17/RF4 49 DSP_UART_RX
U2TX/CN18/RF5 50 DSP_UART_TX
SCK1/INT0/RF6 55 DSP_SPI1_CLOCK
SDI1/RF7 54 DSP_SPI1_MISO
SDO1/RF8 53 DSP_SPI1_MOSI
U2CTS/RF12 40 DSP_POT_CS
U2RTS/RF13 39
C2RX/RG0 90 DSP_CAN_RX
C2TX/RG1 89 DSP_CAN_TX
SCL1/RG2 57 DSP_SCL
SDA1/RG3 56 DSP_SDA
SCK2/CN8/RG6 10 DSP_MEM_CLOCK
SDI2/CN9/RG7 11 DSP_MEM_MISO
SDO2/CN10/RG8 12 DSP_MEM_MOSI
SS2/CN11/RG9 14 DSP_MEM_CS
RG12 96 FPGA_A8
RG13 97 FPGA_A9
RG14 95 FPGA_A10
RG15 1 FPGA_A11
MCLR 13 DSP_MCLR

```
Note: If DSP_FPGAMEM_CS is
      low then the SPI bus is selected
      to the digital potentiometer
```

Title	DSP	
Size A4	Document Number <Doc>	Rev 1
Date: Sunday, March 15, 2009	Sheet 4 of 14	



Title	Optic fiber interface	
Size A4	Document Number <Doc>	Rev 1
Date: Sunday, March 15, 2009	Sheet 5 of 14	



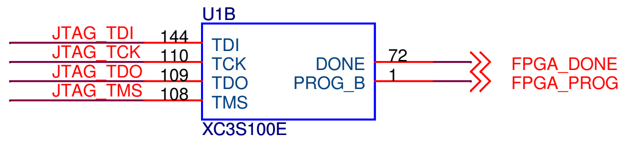
FPGA EEPROM Programming interface

EEPROM for FPGA Data storage (4 MB)

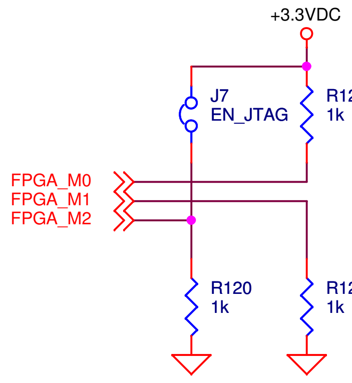
Selector for FPGA memory

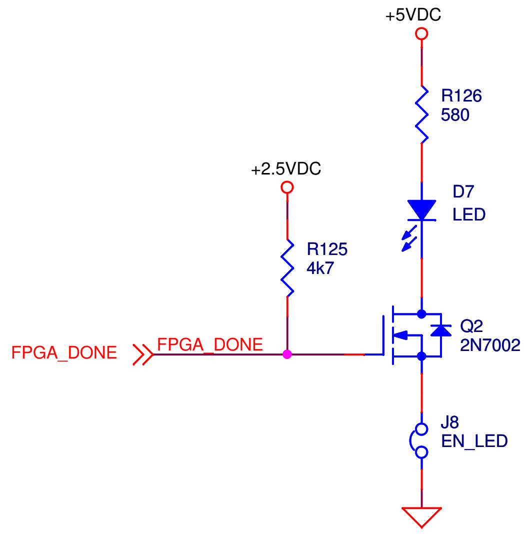
EEPROM for DSP Data storage (16 MB)

Title	Memory	
Size A4	Document Number <Doc>	Rev 1
Date: Sunday, March 15, 2009	Sheet 7 of 14	

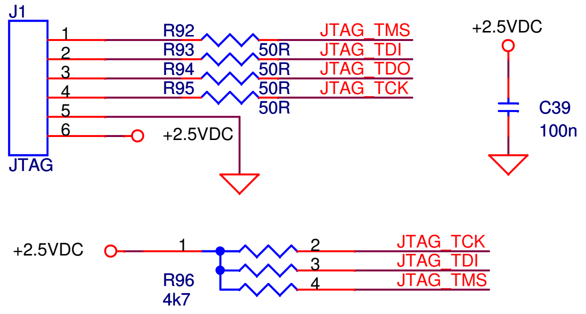


FPGA Interface

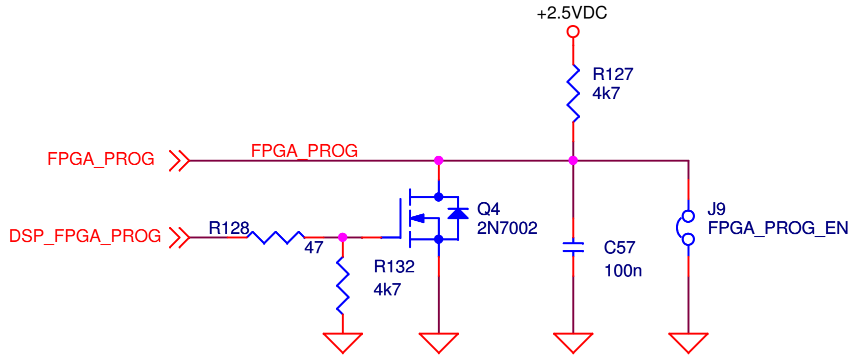




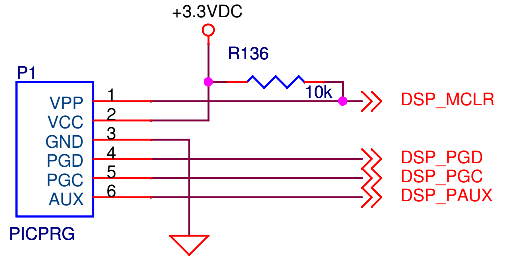
FPGA Done signaling



FPGA JTAG Connector

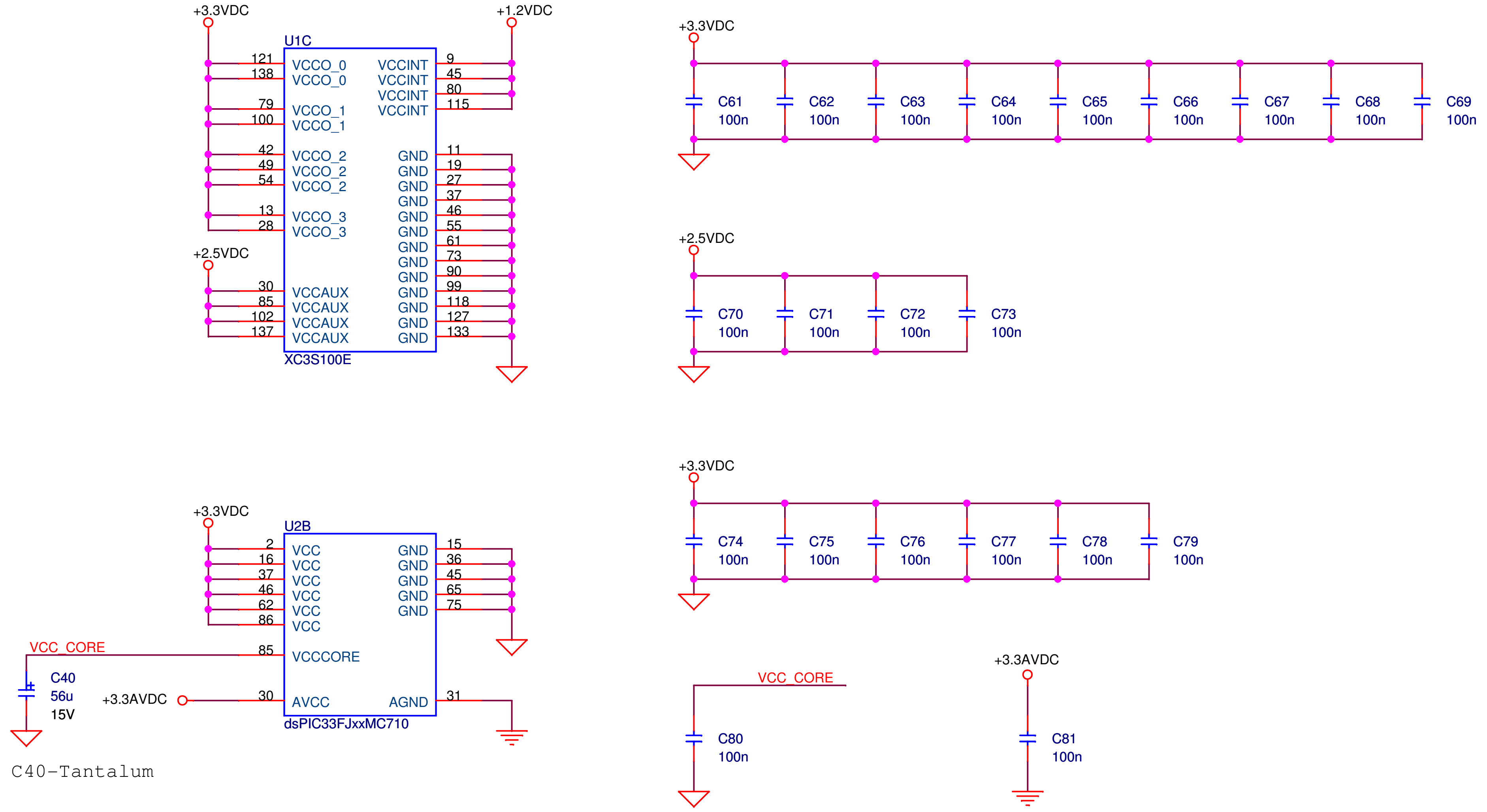


Select FPGA Programming state

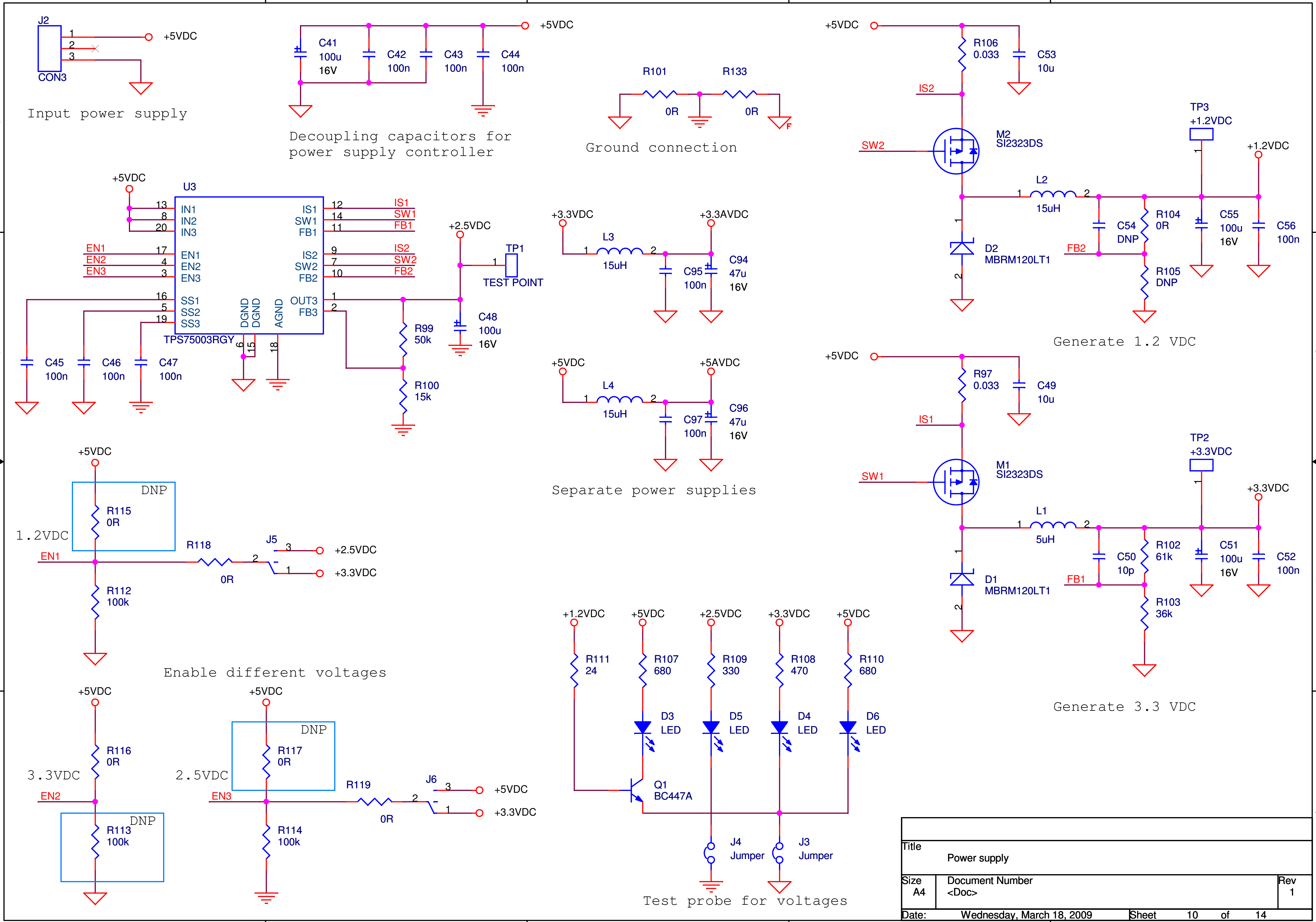


DSP Programming interface

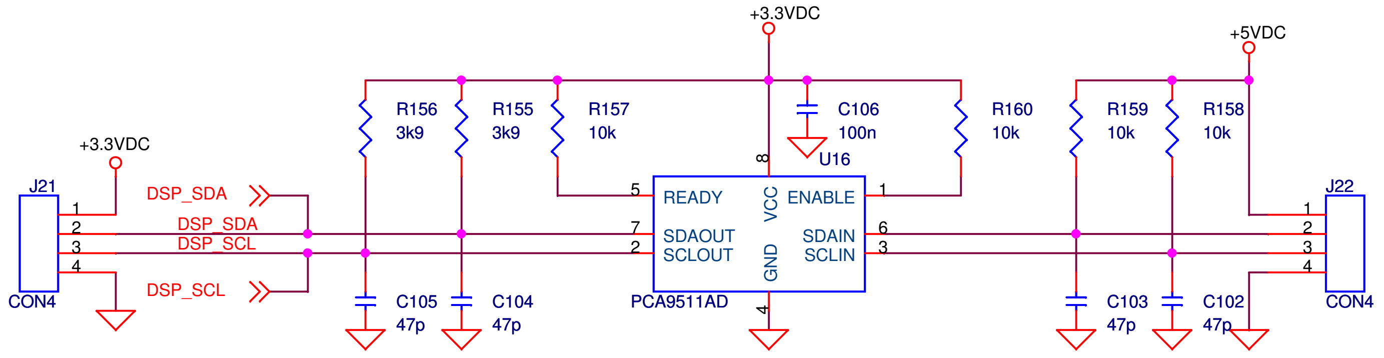
Title	FPGA and DSP Programming	
Size A4	Document Number <Doc>	Rev 1
Date: Sunday, March 15, 2009	Sheet 8 of 14	



Title	FPGA and DSP power			
Size A4	Document Number <Doc>			Rev 1
Date:	Sunday, March 15, 2009	Sheet 9	of	14



Title	Power supply	
Size A4	Document Number <Doc>	Rev 1
Date: Wednesday, March 18, 2009	Sheet 10 of 14	

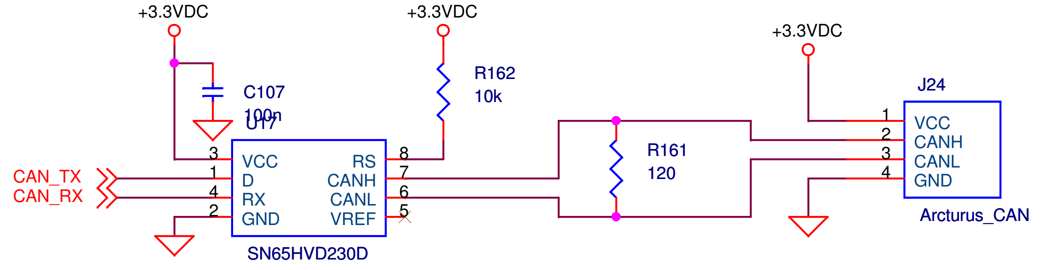


3.3 V link

5 V link

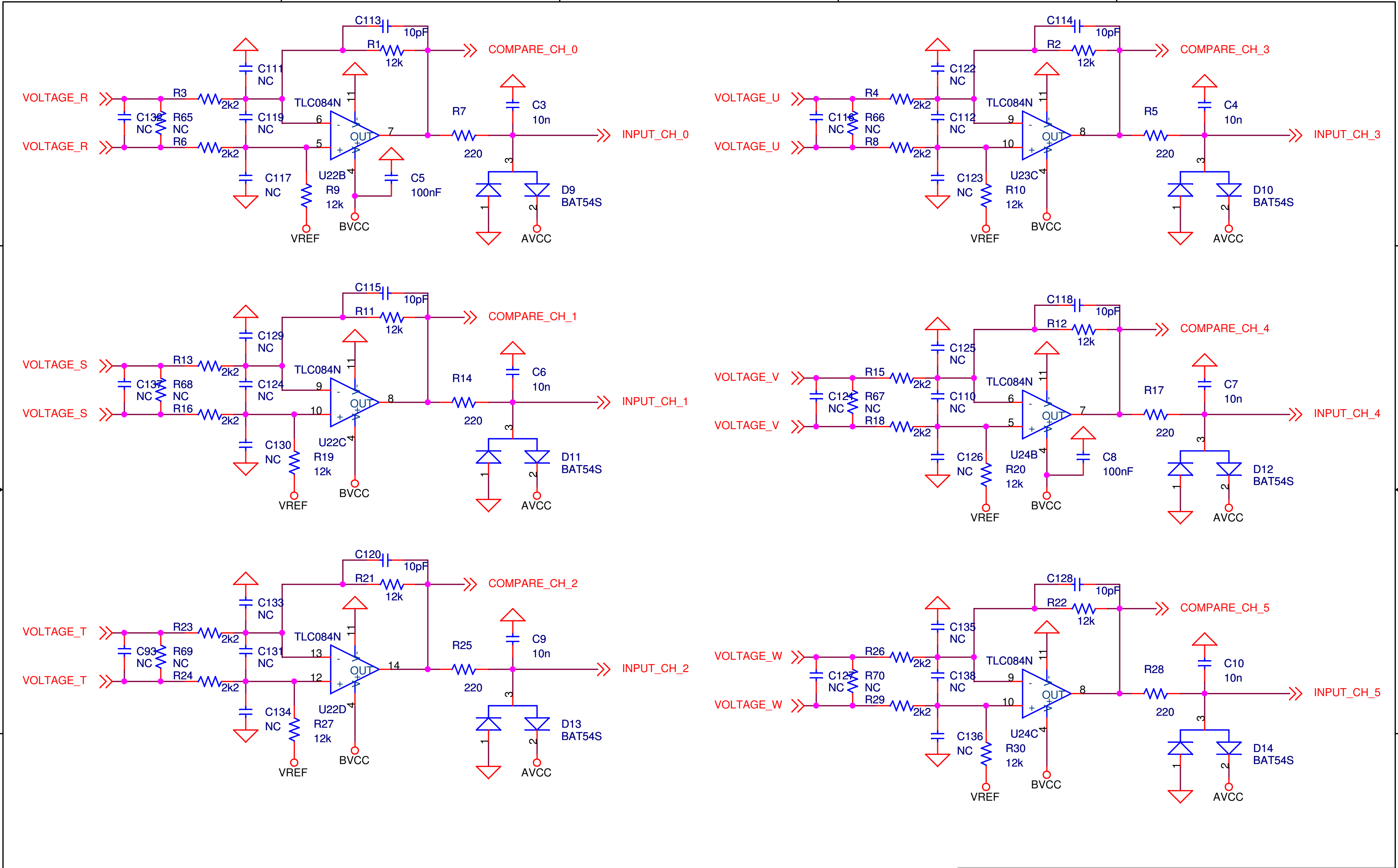
I2C Communicaton for 3.3 and 5 V systems

Note: Capacitors will be placed ONLY if required

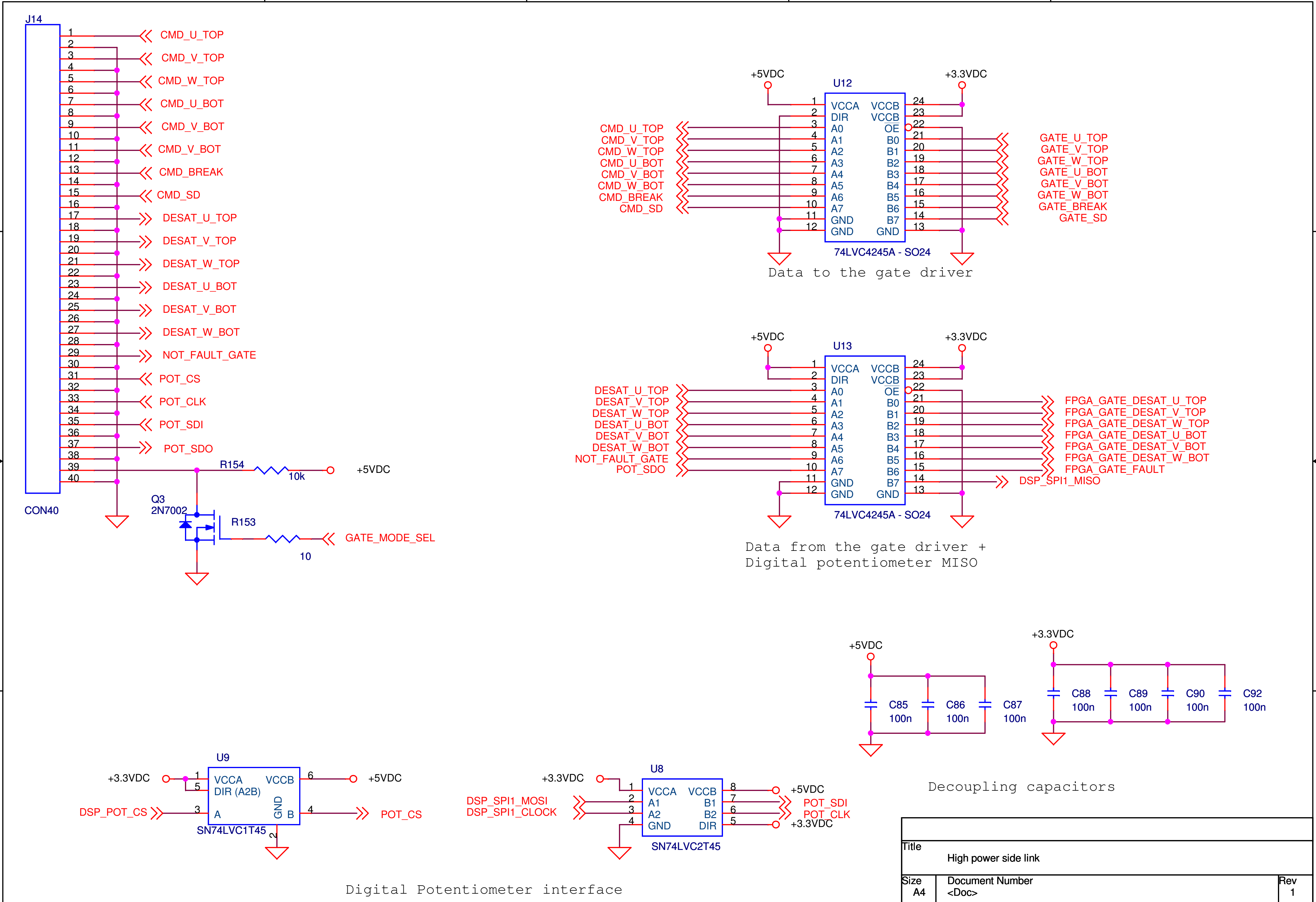


CAN Interface

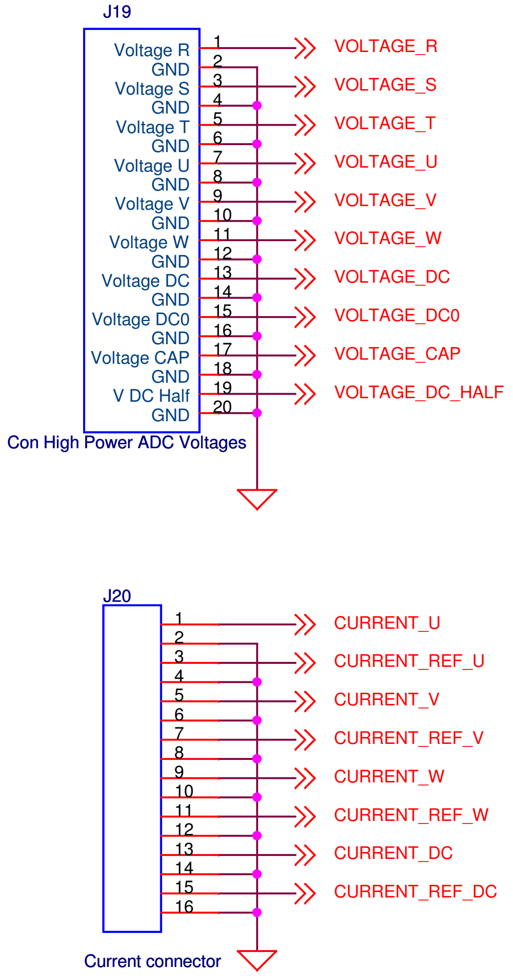
Title	DSP Communication	
Size A4	Document Number <Doc>	Rev 1
Date: Sunday, March 15, 2009	Sheet 11 of 14	



Title	Filter	
Size A4	Document Number <Doc>	Rev 1
Date: Sunday, March 15, 2009	Sheet 12 of 14	



Title	High power side link	
Size A4	Document Number <Doc>	Rev 1
Date: Sunday, March 15, 2009	Sheet 13 of 14	



Sensors connectors

Title	Connectors	
Size A4	Document Number <Doc>	Rev 1
Date: Sunday, March 15, 2009	Sheet 14 of 14	

5 4 3 2 1

Modular Multi-Level Inverter

Part 1: Unit Design

Section C: Fast Fiber optic interface

Master Thesis

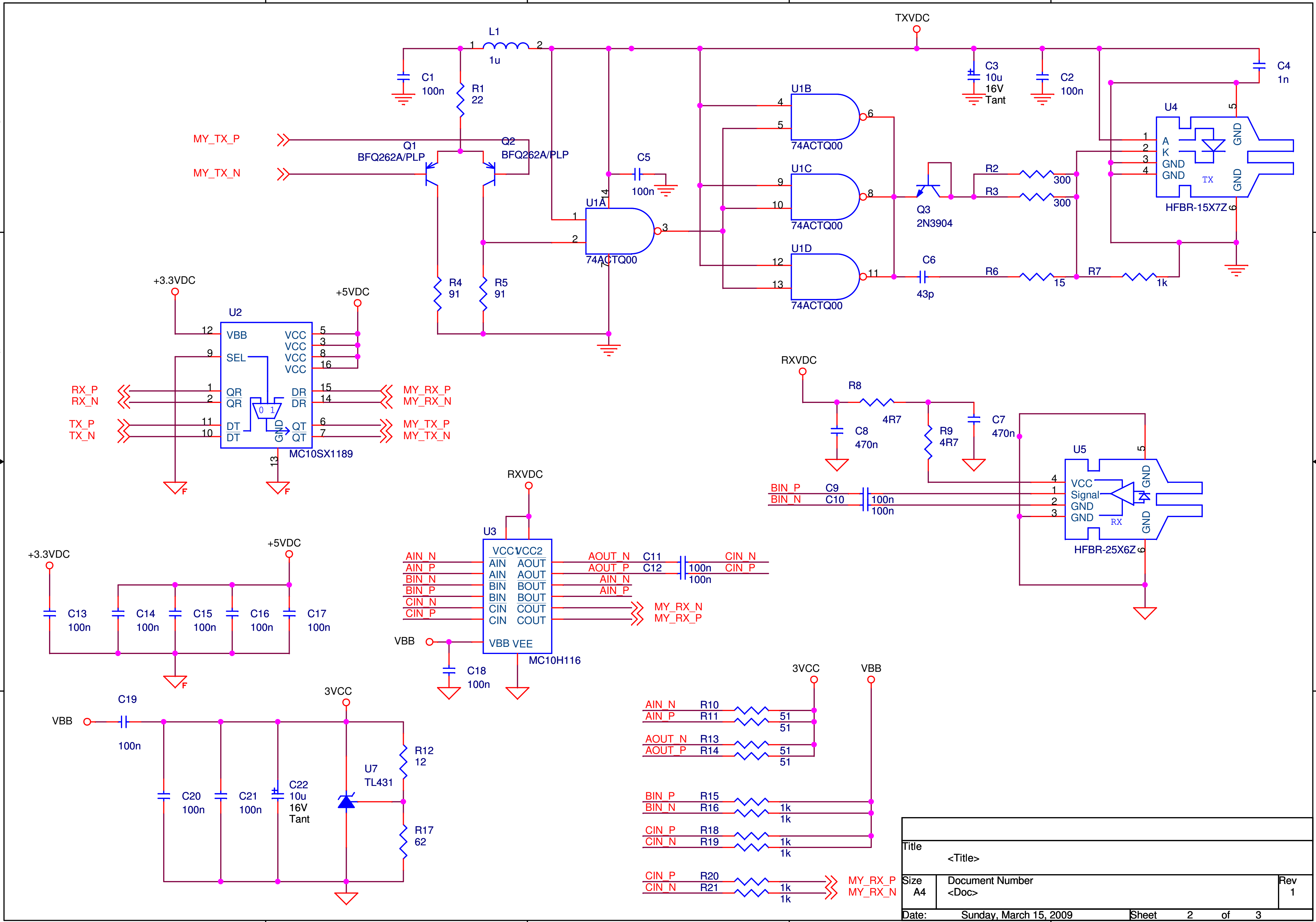
Authors:
Cristian Sandu
Nicoleta Carnu
Valentin Costea

Supervisor:
Stig Munk Nielsen

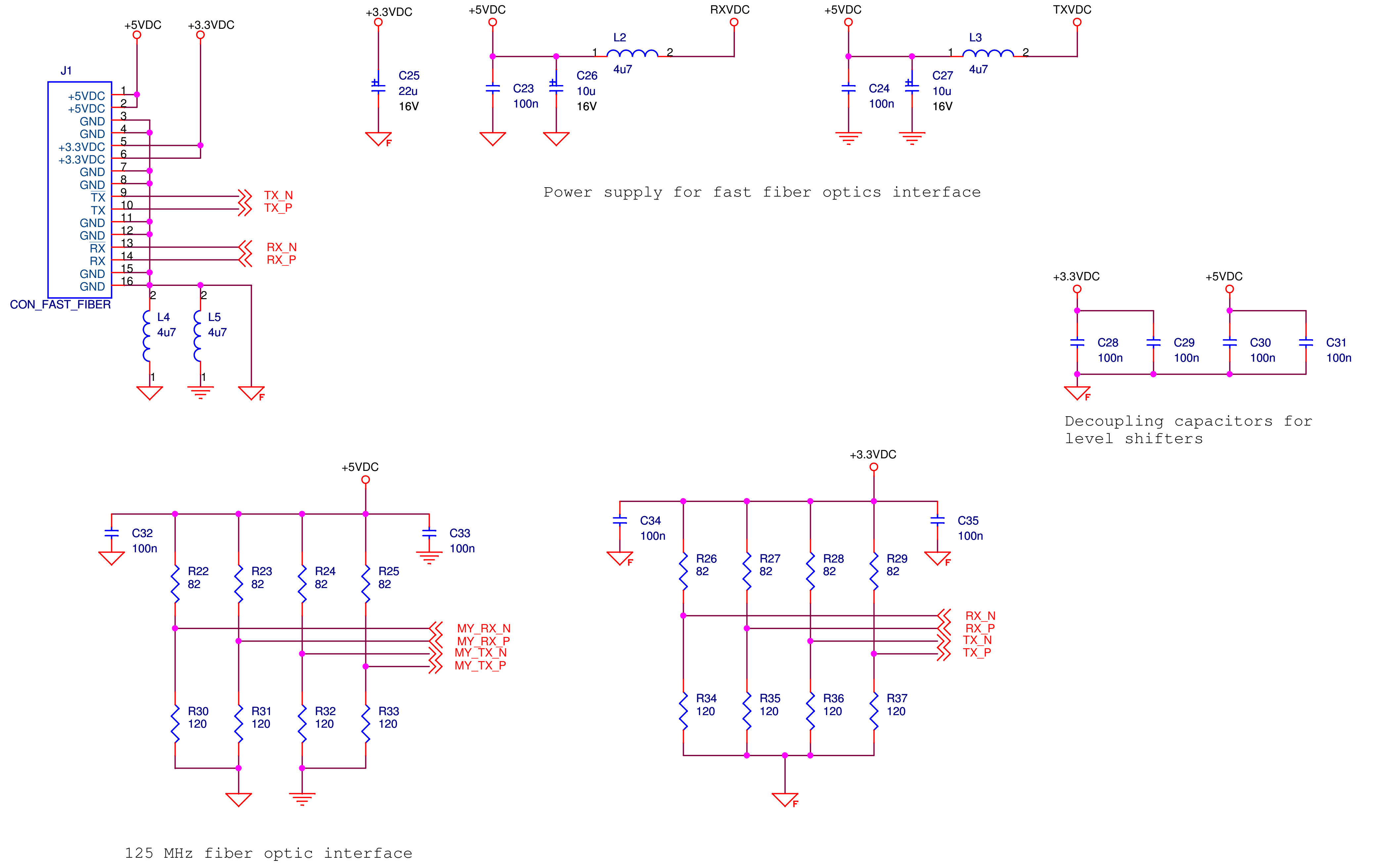
Aalborg University 2009

Title	Introduction	
Size A4	Document Number <Doc>	Rev 1
Date: Sunday, March 15, 2009	Sheet 1 of 3	

5 4 3 2 1



Title	<Title>	
Size A4	Document Number <Doc>	Rev 1
Date:	Sunday, March 15, 2009	Sheet 2 of 3



Title	Power supply	
Size A4	Document Number <Doc>	Rev 1
Date: Sunday, March 15, 2009	Sheet 3 of 3	

Modular Multi-Level Inverter

Part 1: Unit Design

Section D: Protection Module

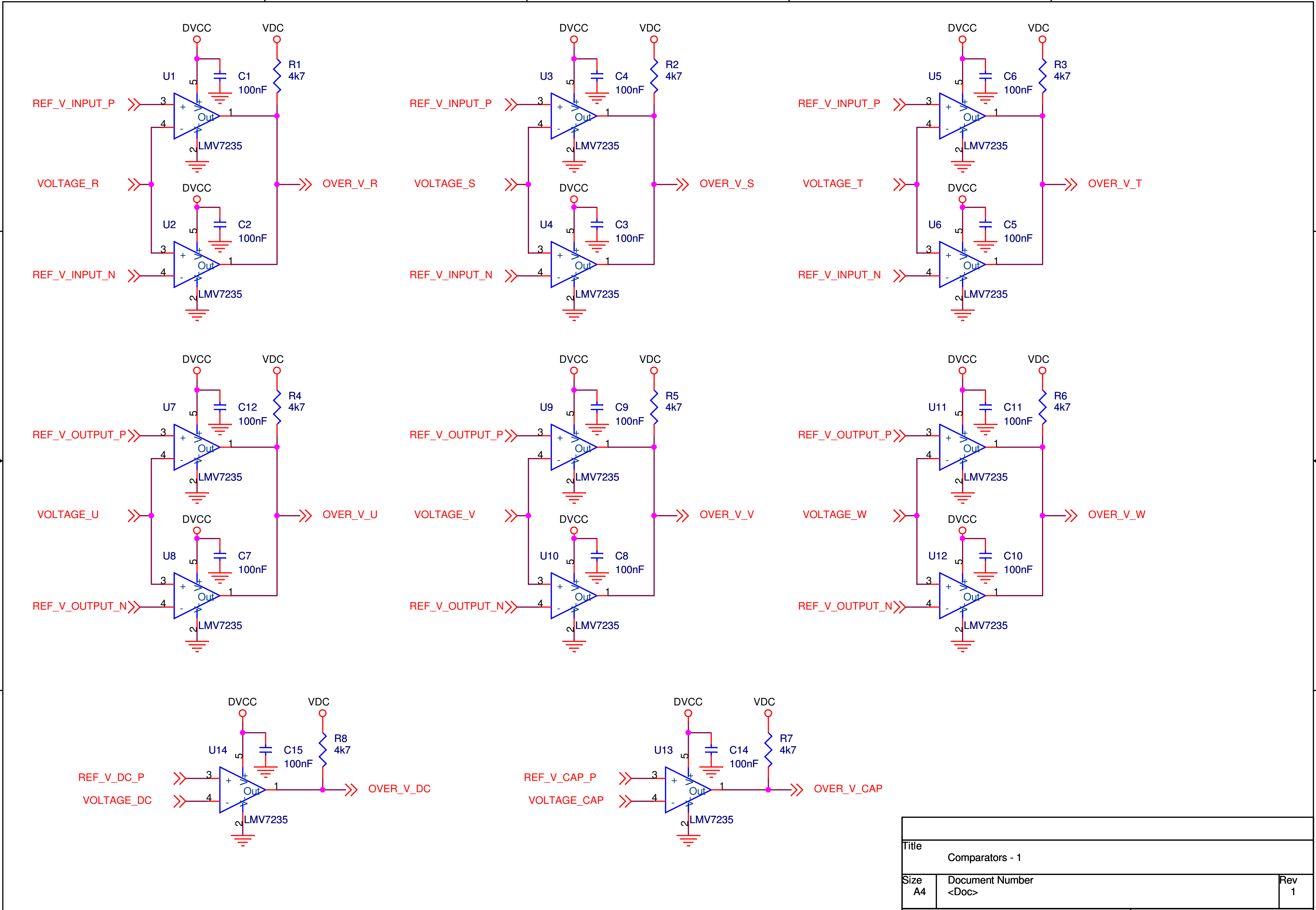
Master Thesis

Authors:
Cristian Sandu
Nicoleta Carnu
Valentin Costea

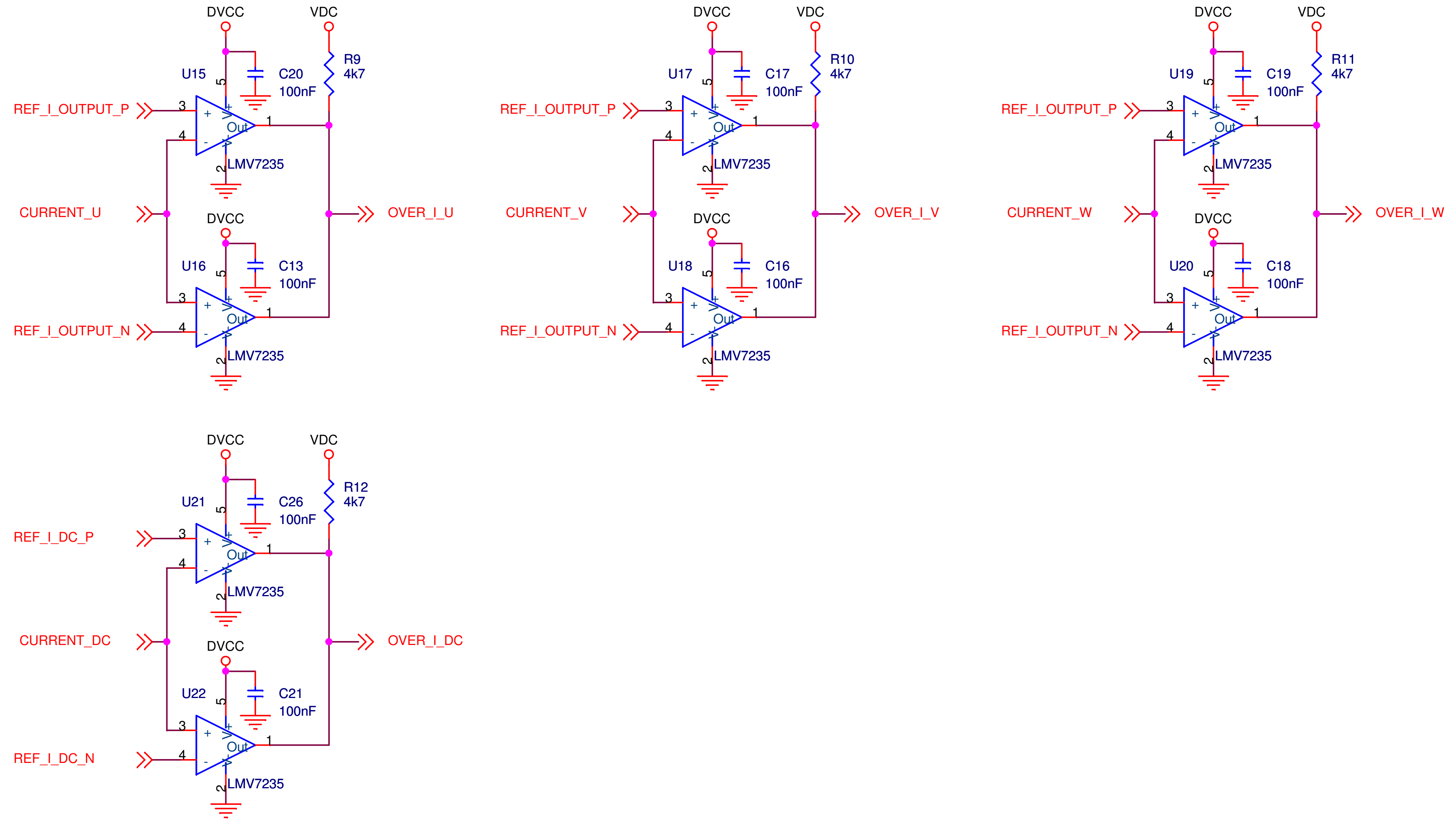
Supervisor:
Stig Munk Nielsen

Aalborg University 2009

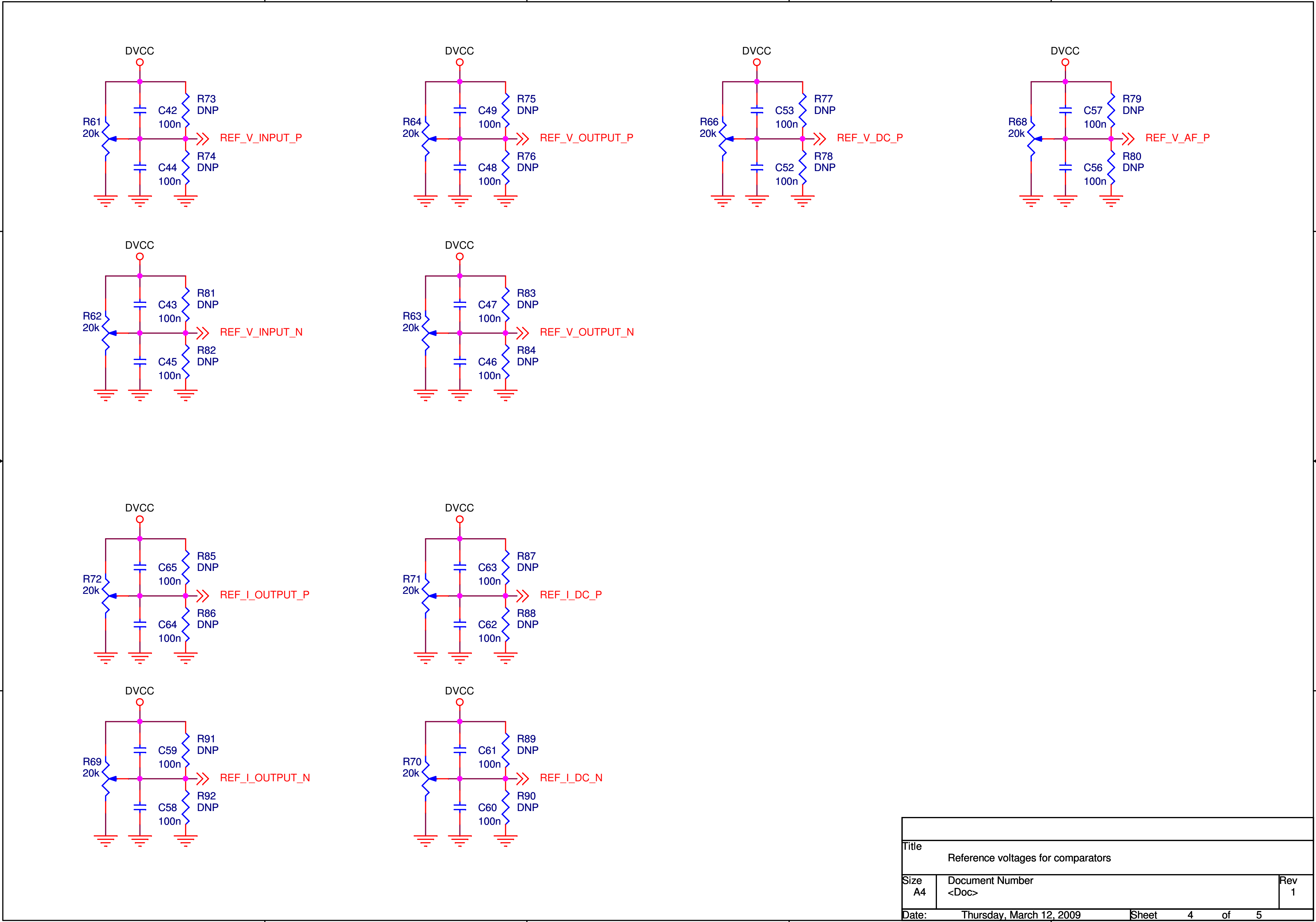
Title: Introduction		
Size: A4	Document Number: <Doc>	Rev: 1
Date: Thursday, March 12, 2009	Sheet 1 of 5	



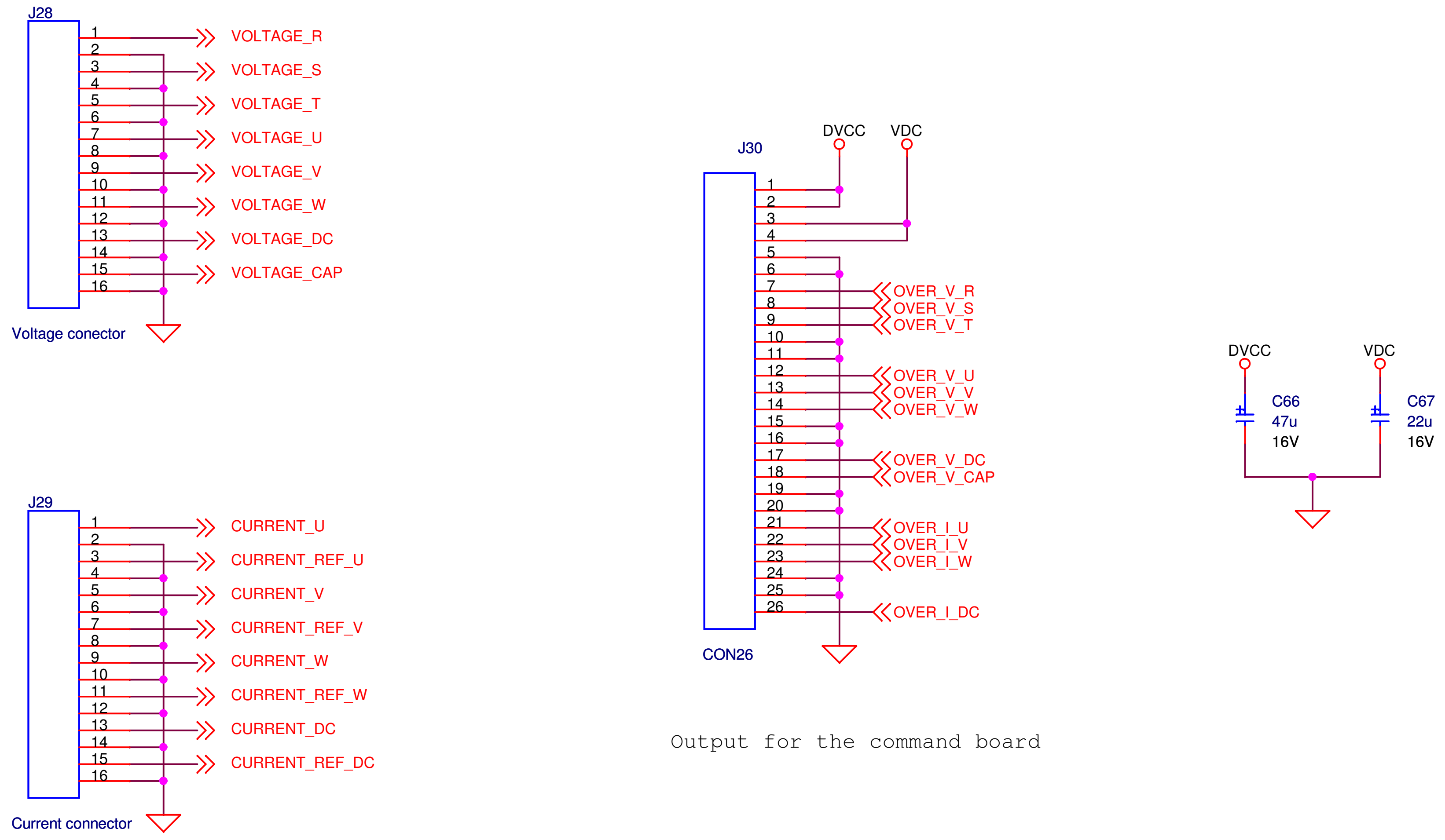
Title	Comparators - 1		
Size A4	Document Number <Doc>		Rev 1
Date:	Thursday, March 12, 2009	Sheet 2 of 5	



Title	Comparators - 2		
Size A4	Document Number <Doc>		Rev 1
Date:	Thursday, March 12, 2009	Sheet 3 of 5	



Title	Reference voltages for comparators	
Size A4	Document Number <Doc>	Rev 1
Date: Thursday, March 12, 2009	Sheet 4 of 5	



Title	Connectors	
Size A4	Document Number <Doc>	Rev 1
Date: Thursday, March 12, 2009	Sheet 5 of 5	

Modular Multi-Level Inverter

Part 1: Unit Design

Section E: Switched mode power supply

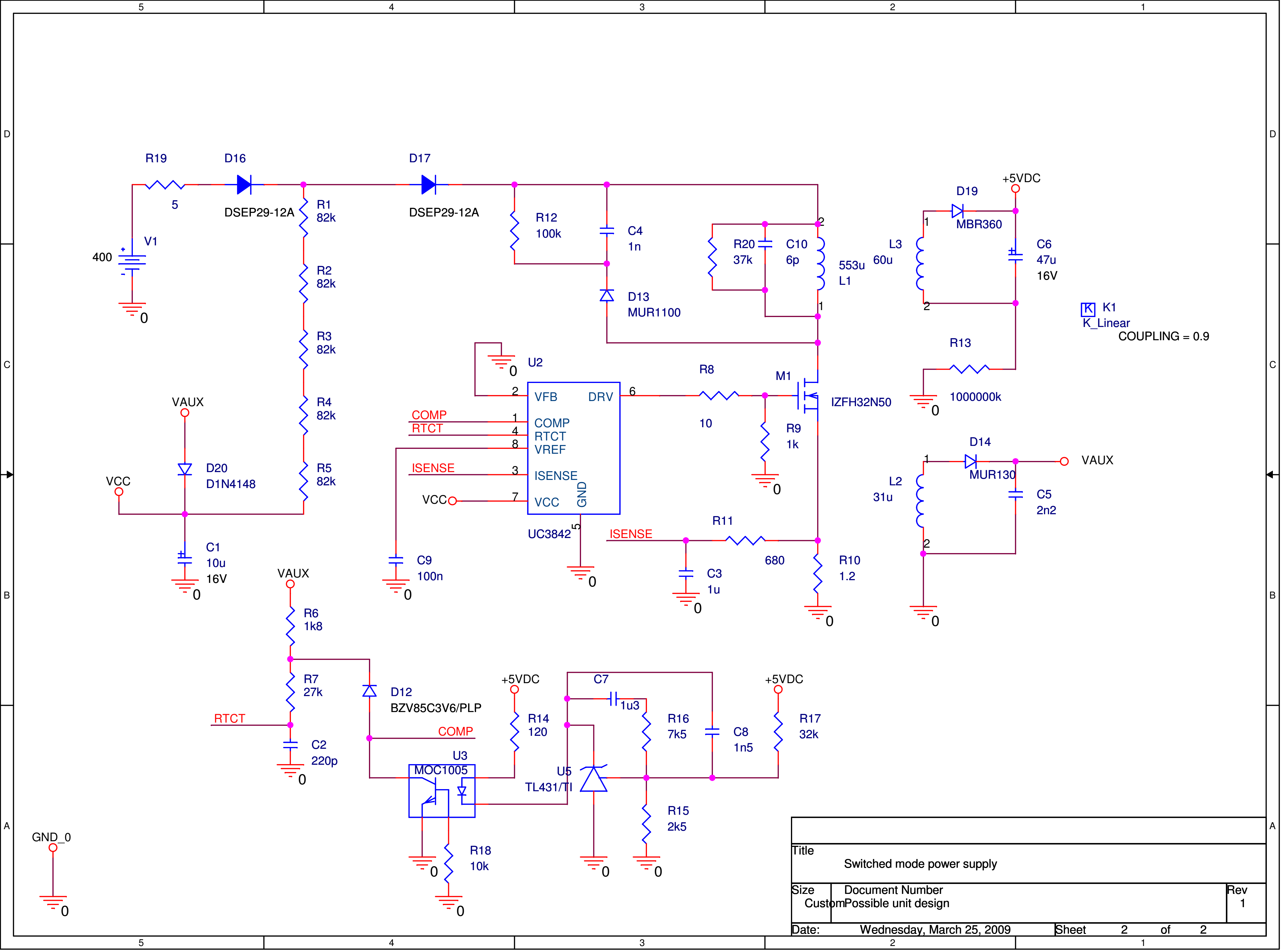
Master Thesis

Authors:
Valentin Costea
Cristian Sandu
Nicoleta Carnu

Supervisor:
Stig Munk Nielsen

Aalborg University 2009

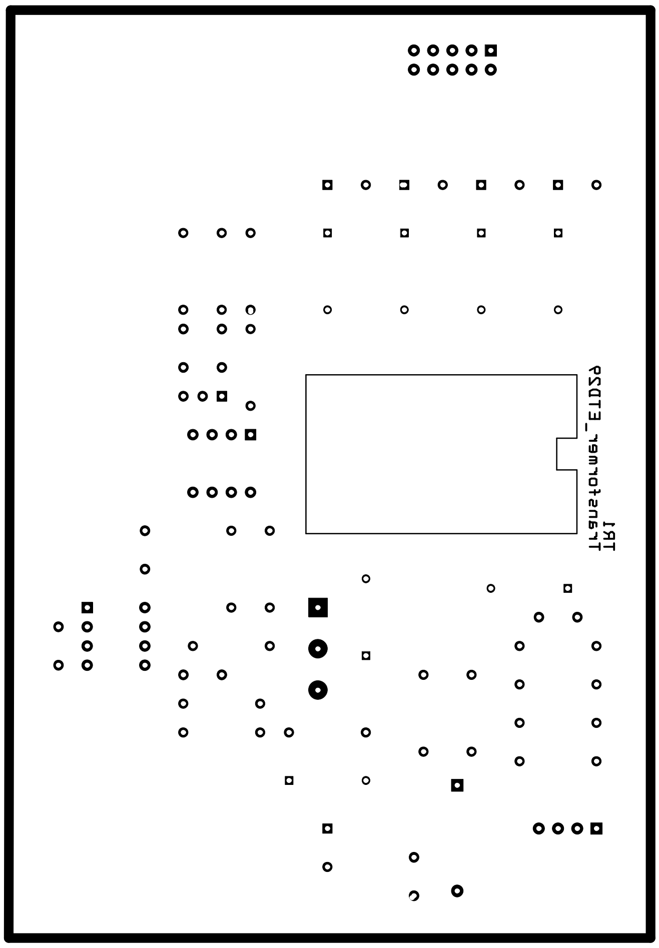
Title	Switched mode power supply	
Size A4	Document Number Possible unit design	Rev 1
Date: Sunday, May 31, 2009	Sheet 1 of 2	

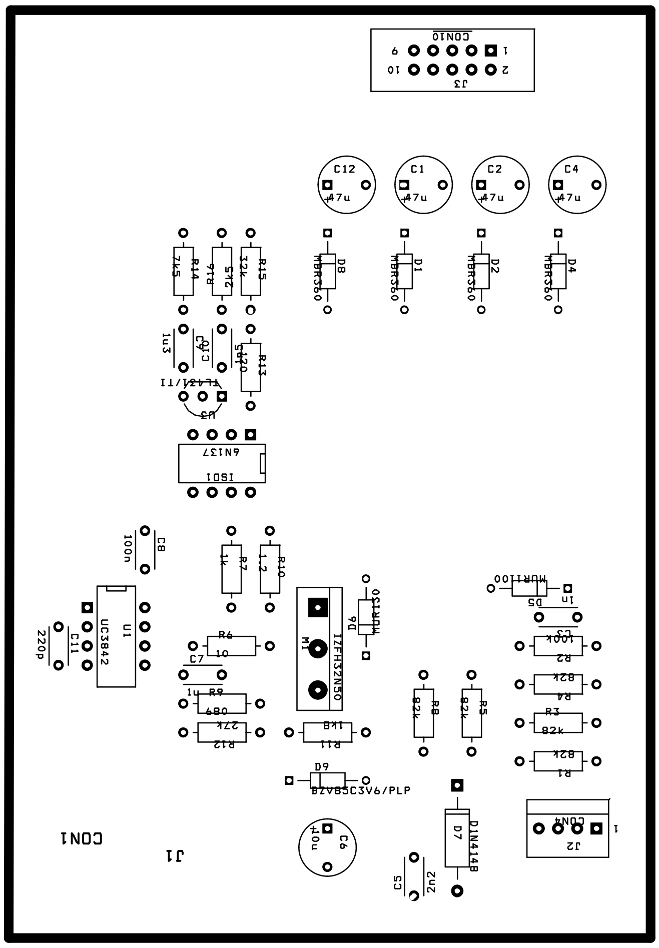


Title
Switched mode power supply

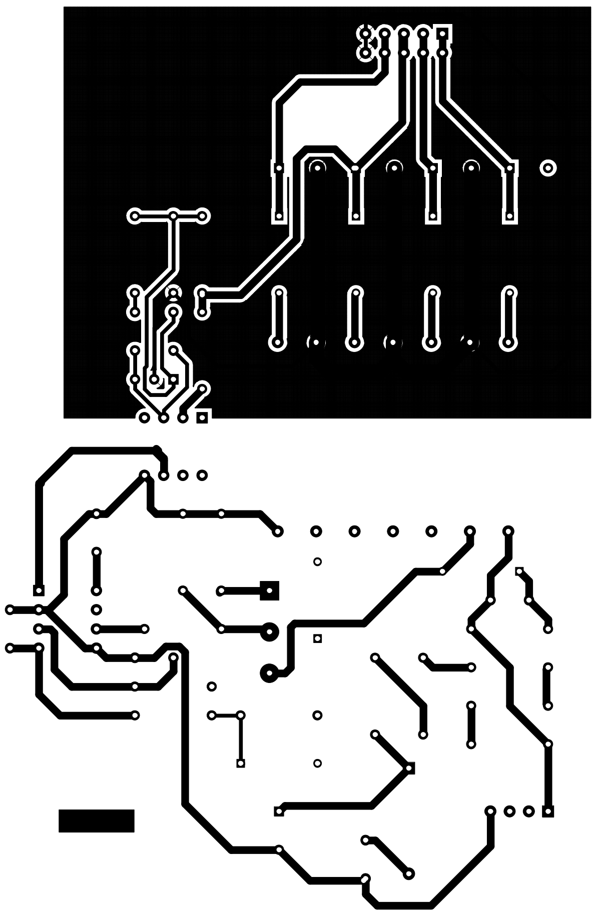
Size	Document Number	Rev
Custom	Possible unit design	1

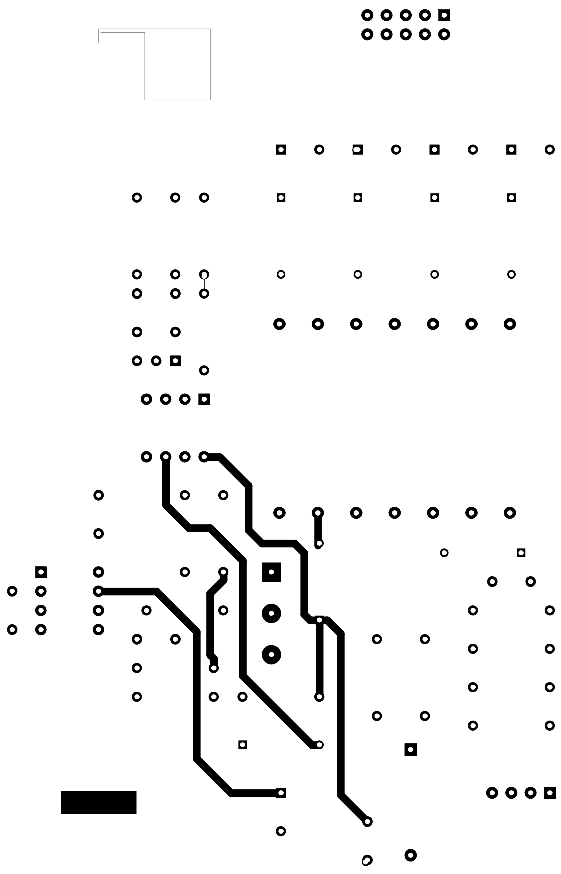
Date: Wednesday, March 25, 2009 Sheet 2 of 2


TR1
Transformer_EI30



CON10
J3
1
2
9
10
C12
C1
C2
C4
47u
D8
D1
D2
D4
MBR360
R14
7k5
R18
2k2
R15
32k
1u3
C9
R13
TL431/TI
U3
6N137
ISO1
C8
100n
R7
1k
R10
1.2
U1
UC3842
C11
220p
R6
10
C7
1u
R9
680
R12
27k
M1
IRFH32N50
D6
MUR130
R11
1k8
D9
BZV85C3V6/PLP
C6
10u
C5
2n2
D7
D1N4148
MUR100
D5
1n
C3
R2
100k
R4
82k
R3
82k
R1
82k
R8
82k
R5
82k
CON4
J2
CON1
J1





APPENDIX D HARDWARE

A.1 INTER-FPGA COMMUNICATION

The pin-outs of the connection are presented in Table D-1. The P and N represent the positive and negative lines of the differential signals.

Name	Main FPGA pin	Main FPGA signals	Extension board pin	FPGA Board pin	Secondary FPGA pin	Secondary FPGA signals
Clock	P: B14 N: A14	L26	P: J1/3 N: J1/5	P: TX 29 N: TX30	P: AA10 N: AB10	TX_CLK L15
MISO 0	P: D16 N: C15	L21	P: J1/4 N: J1/6	P: TX25 N: TX26	P: AA8 N: AB8	TXP_4 L12
MISO 1	P: B15 N:A15	L23	P: J1/7 N: J1/9	P: TX21 N: TX22	P: Y7 N: AB7	TXP_3 L10
MISO 2	P: D13 N: C12	L30	P: J1/8 N: J1/10	P: TX13 N: TX14	P: AA6 N: AB6	TXP_2 L08
MISO 3	P: F15 N: E15	L20	P: J1/13 N: J1/15	P: TX9 N: TX10	P: AB3 N: AA4	TXP_1 L04
INT	P: A12 N: B12	L29	P: J1/14 N: J1/16	P: TX5 N: TX6	P: AB2 N: AA3	TXP_0 L03
MOSI 0	P: E14 N: F14	L24	P: J1/17 N: J1/19	P: RX5 N: RX6	P: A4 N: B4	RXP_0 L31
MOSI 1	P: C10 N: D10	L34	P: J1/16 N: J1/20	P: RX9 N: RX10	P: B6 N: A5	RXP_1 L28
MOSI 2	P: G15 N: H15	L16	P: J1/23 N: J1/25	P: RX13 N: RX14	P: A7 N: A6	RXP_2 L26
MOSI 3	P: C11 N: D11	L32	P: J1/24 N: J1/26	P: RX21 N: RX22	P: A9 N: A8	RXP_3 L22
Chip Select	P: G12 N: H12	L35	P: J1/27 N: J1/29	P: RX25 N: RX26	P: A10 N: C10	RXP_4 L21
Auxiliary	P: A4 N: B4	L45	P: J1/28 N: J1/30	P: RX29 N: RX30	P: A12 N: A11	RX_CLK L18

Table D-1: Pin-out for inter FPGA communication

A.2 DSP WITH FPGA INTERFACE

The connection between the DSP and the FPGA is made over a shielded cable no longer then 10 cm. The connection lines are shown by Table D-2. The FPGA connector is the SAM extension line. Due to FPGA configurability of pin roles, the lines have been configured to fit the FPGA. Therefore in the pinout table the actual signal names are those given by the DSP and not by the FPGA board design.

P2 DSP Pin	DSP Pin Name	FPGA Pin	SAM name	SAM pin no	SAM pin no	SAM Name	FPGA Pin	DSP Name	DSP Pin
		n/a	VCC	01	02	VCC	n/a		
		n/r	TDO	03	04	GND	n/a		
		n/r	TMS	05	06	CLK	AA14	CLK	48
		n/r	TDI	07	08	GND	n/a		

		n/r	PROG	09	10	TCLK	n/r		
		n/a	GND	11	12	GND	n/a		
3	D0	V22	OE	13	14	INIT	n/r		
4	D1	AC26	A0	15	16	WE	V24	D2	5
6	D3	AB23	A2	17	18	A1	AB26	D4	7
		n/a	2,5VDC	19	20	A3	AB24	D5	8
9	D6	AA23	D0	21	22	2,5VDC	n/a		
11	D8	U20	D2	23	24	D1	V21	D7	10
13	D10	AA25	D4	25	26	D3	AA24	D9	12
15	D12	U18	D6	27	28	D5	U19	D11	14
17	D14	Y23	D8	29	30	D7	Y22	D13	16
19	A0	T20	D10	31	32	D9	U21	D15	18
21	A2	Y25	D12	33	34	D11	Y24	A1	20
23	A4	T17	D14	35	36	D13	T18	A3	22
25	A6	V18	A4	37	38	D15	W23	A5	24
27	A8	AA22	A6	39	40	A5	V19	A7	26
40	ZCS6	L23	IRQ	41	42	GND	n/a		
43	XWR	V23	RST	43	44	CE	V25	A9	28
39	XRDY	n/r	Done	45	46	BRDY	P21	XRD	44
		n/r	CLK	47	48	F_Done	n/r		
		n/a	GND	49	50	n/c	n/c	-	

Table D-2: DSP – FPGA pinouts

A.3 RELAY OUTPUT

The pin-outs of the microcontroller for relay output are presented in Table D-3.

Id	Relay	Microcontroller pin	Role
1	A1-1	C1	Reserved
2	A1-2	C2	Reserved
3	A1-3	C3	Reserved
4	A1-4	D0	Reserved
5	A2-1	D1	Auxiliary
6	A2-2	B4	Auxiliary
7	A2-3	B3	Auxiliary
8	A2-4	B2	Auxiliary
9	A3-1	D7	Units U1-U2
10	A3-2	D6	Units U3-U4
11	A3-3	D5	Units U5-U6
12	A3-4	D4	Units U7-U8
13	A4-1	C7	Units V1-V4
14	A4-2	C6	Units V5-V8
15	A4-3	D2	Units W1-W4
16	A4-4	D3	Units W5-W8
17	A5-1	A0	Contactor K1
18	A5-2	A1	Contactor K2
19	A5-3	A2	Contactor K3
20	A5-4	A3	Contactor K4
21	A6-1	A4	Contactor K5

22	A6-2	E0	Contactor K6
23	A6-3	E1	Aux Relay K7
24	A6-4	E2	Aux Relay K8

Table D-3: Relay output microcontroller pins

APPENDIX E SOURCE CODE FOR S-FUNCTIONS IN SIMULINK

E.1 LEVEL SHIFTED

```
// LevelShifted
//
// Level Shifted modulation
//
// Copyright:
//    Sandu Cristian - 2008
//    sanducristian@gmail.com
//       Code  created  for  the  Project  of  the  10th
semester at
//    Aalborg University
//



#define S_FUNCTION_NAME  LevelShifted
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#ifdef MATLAB_MEX_FILE
#include <math.h>
#endif


#define SWITCHING_FREQUENCY          (900.0)
#define HALF_UNIT_COUNT              (4)
#define  UNIT_COUNT                              (2  *
HALF_UNIT_COUNT)
#define MAX_VALUE                    (100)
#define  SAMPLE_TIME                           (1.0  /
(SWITCHING_FREQUENCY * MAX_VALUE))
#define MAX_CYCLES_ZERO_CROSS        (5)

#define COUNT_DELTA                    (MAX_VALUE * 2.0
/ UNIT_COUNT)
#define  LEVEL_SIZE                            (1.0  /
HALF_UNIT_COUNT)


#define TYPE_IPD                     (0)
#define TYPE_APOD                    (1)
#define TYPE_POD                     (2)

#define USE_TYPE                     TYPE_IPD

#define OP_NONE                      (0)
#define OP_UNIT_CYCLE                (1)
#define OP_VOLTAGE_BALANCE           (2)

#define OP_TYPE                      OP_UNIT_CYCLE


double * dValues;             // Global double values
for the class
int * nValues;               // Global integer values
for the class
SimStruct * baseStruct;

#define  nCounts(a)         nValues[(0  *  UNIT_COUNT)  +
(a)]
#define  nSign(a)           nValues[(1  *  UNIT_COUNT)  +
(a)]
#define  nIndexesU(a)      nValues[(2  *  UNIT_COUNT)  +
(a)]
#define  nIndexesV(a)      nValues[(3  *  UNIT_COUNT)  +
(a)]
#define  nIndexesW(a)      nValues[(4  *  UNIT_COUNT)  +
(a)]
#define nCyclesU         nValues[(5 * UNIT_COUNT) + 0]
#define nCyclesV         nValues[(5 * UNIT_COUNT) + 1]
#define nCyclesW         nValues[(5 * UNIT_COUNT) + 2]

#define nSignVoltageU    nValues[(5 * UNIT_COUNT) + 3]
#define nSignVoltageV    nValues[(5 * UNIT_COUNT) + 4]
#define nSignVoltageW    nValues[(5 * UNIT_COUNT) + 5]

// Only positive carriers
#define dCariers(a)          dValues[( 0 * UNIT_COUNT)
+ (a)]

#define dCurSenseU           dValues[( 1 * UNIT_COUNT)
+ 0]
#define dCurSenseV           dValues[( 1 * UNIT_COUNT)
+ 1]
#define dCurSenseW           dValues[( 1 * UNIT_COUNT)
+ 2]

#define dOldUnitStateU(a)    dValues[( 2 * UNIT_COUNT)
+ (a)]
#define dOldUnitStateV(a)    dValues[( 3 * UNIT_COUNT)
+ (a)]
#define dOldUnitStateW(a)    dValues[( 4 * UNIT_COUNT)
+ (a)]

#define dOldLegStateU0(a)    dValues[( 5 * UNIT_COUNT)
+ (a)]
#define dOldLegStateU1(a)    dValues[( 6 * UNIT_COUNT)
+ (a)]
#define dOldLegStateV0(a)    dValues[( 7 * UNIT_COUNT)
+ (a)]
#define dOldLegStateV1(a)    dValues[( 8 * UNIT_COUNT)
+ (a)]
#define dOldLegStateW0(a)    dValues[( 9 * UNIT_COUNT)
+ (a)]
#define dOldLegStateW1(a)    dValues[(10 * UNIT_COUNT)
+ (a)]

#define dCarrierOffsets(a)   dValues[(11 * UNIT_COUNT)
+ (a)]

/*====================*
 * S-function methods *
 *====================*/

/*             Function:             mdlInitializeSizes
===============================================
 * Abstract:
 *     The  sizes  information  is  used  by  Simulink  to
determine the S-function
 *       block's  characteristics  (number  of  inputs,
outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S){
    ssSetNumSFcnParams(S, 0);   /* Number  of  expected
parameters */
    if              (ssGetNumSFcnParams(S)              !=
ssGetSFcnParamsCount(S)) {
        /* Return  if  number  of  expected  !=  number  of
actual parameters */
        return;
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, 3)) return;
    /*Input Port 0 */
    ssSetInputPortWidth(S,  0, 1); /* Enabled */
    ssSetInputPortDataType(S, 0, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  0, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortRequiredContiguous(S,     0,     1);
/*direct input signal access*/

    /*Input Port 0 */
    ssSetInputPortWidth(S,     1,   3);   /*   Desired
voltages */
    ssSetInputPortDataType(S, 1, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  1, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S, 1, 1);
    ssSetInputPortRequiredContiguous(S,     1,     1);
/*direct input signal access*/

    /*Input Port 0 */
    ssSetInputPortWidth(S,  2, 3); /* Phase currents
*/
    ssSetInputPortDataType(S, 2, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  2, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S, 2, 1);
    ssSetInputPortRequiredContiguous(S,     2,     1);
/*direct input signal access*/

    /*
     * Set direct feedthrough flag (1=yes, 0=no).
     * A port has direct feedthrough if the input is
used in either
     *  the  mdlOutputs  or  mdlGetTimeOfNextVarHit
functions.
     *                                           See
matlabroot/simulink/src/sfuntmpl_directfeed.txt.
     */
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S, 7)) return;

    /* Output Port 0 - Carriers */
    ssSetOutputPortWidth(S, 0, UNIT_COUNT);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 0, COMPLEX_NO);

    /* Output Port 1 - Unit U states */
    ssSetOutputPortWidth(S, 1, UNIT_COUNT);
    ssSetOutputPortDataType(S, 1, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 1, COMPLEX_NO);

    /* Output Port 2 - IGBT U States */
    ssSetOutputPortWidth(S, 2, 2 * UNIT_COUNT);
    ssSetOutputPortDataType(S, 2, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 2, COMPLEX_NO);

    /* Output Port 3 - Unit V states */
    ssSetOutputPortWidth(S, 3, UNIT_COUNT);
    ssSetOutputPortDataType(S, 3, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 3, COMPLEX_NO);

    /* Output Port 4 - IGBT V States */
    ssSetOutputPortWidth(S, 4, 2 * UNIT_COUNT);
    ssSetOutputPortDataType(S, 4, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 4, COMPLEX_NO);

    /* Output Port 5 - Unit W states */
    ssSetOutputPortWidth(S, 5, UNIT_COUNT);
    ssSetOutputPortDataType(S, 5, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 5, COMPLEX_NO);

    /* Output Port 6 - IGBT W States */
    ssSetOutputPortWidth(S, 6, 2 * UNIT_COUNT);
    ssSetOutputPortDataType(S, 6, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 6, COMPLEX_NO);



    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, UNIT_COUNT * 15);
    ssSetNumIWork(S, UNIT_COUNT * 10);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    ssSetOptions(S, 0);
}


/*         Function:         mdlInitializeSampleTimes
=========================================
 * Abstract:
 *      This  function  is  used  to  specify  the  sample
time(s) for your
 *      S-function. You must register the same number
of sample times as
 *    specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct *S){
    ssSetSampleTime(S, 0, SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}



#define  MDL_INITIALIZE_CONDITIONS      /*  Change  to
#undef to remove function */
#if defined(MDL_INITIALIZE_CONDITIONS)
  static void mdlInitializeConditions(SimStruct *S){
  }
#endif /* MDL_INITIALIZE_CONDITIONS */



#define  MDL_START   /*  Change  to  #undef  to  remove
function */
#if defined(MDL_START)
  /*                Function:                 mdlStart
=====================================================
==
   * Abstract:
   *      This  function  is  called  once  at  start  of
model execution. If you
   *     have states that should be initialized once,
this is the place
   *    to do it.
   */
  static void mdlStart(SimStruct *S) {
    int i, j;

    // Retrieve global variables
    dValues = ssGetRWork(S);
    nValues = ssGetIWork(S);
    baseStruct = S;

    // Initialize indexes
    for (i = 0; i < UNIT_COUNT; i++){
        nIndexesU(i) = i;
        nIndexesV(i) = i;
        nIndexesW(i) = i;
    }

    // Do initialization of carriers levels
    #if USE_TYPE == USE_IPD
        for (i = 0; i < HALF_UNIT_COUNT; i++){
            dCarrierOffsets(i) = i * LEVEL_SIZE;
            dCarrierOffsets(i  +  HALF_UNIT_COUNT)  =  -
(i + 1) * LEVEL_SIZE;

            nCounts(i) = 0;
            nCounts(i + HALF_UNIT_COUNT) = 0;

            nSign(i) = 1.0;
            nSign(i + HALF_UNIT_COUNT) = 1.0;
        }
    #endif
    #if USE_TYPE == TYPE_APOD
        j = 1;
        for (i = 0; i < HALF_UNIT_COUNT; i++){
            dCarrierOffsets(i) = i * LEVEL_SIZE;
            dCarrierOffsets(i  +  HALF_UNIT_COUNT)  =  -
(i + 1) * LEVEL_SIZE;
```

```
            nCounts(i)  =  ((i  %  2)  ==  0)  ?  0  :
MAX_VALUE;
            nCounts(i  +  HALF_UNIT_COUNT)  =  ((i  %  2)
== 0) ? MAX_VALUE : 0;

            nSign(i) = ((i % 2) == 0) ? -1.0 : +1.0;
            nSign(i  +  HALF_UNIT_COUNT)  =  ((i  %  2)  ==
0) ? +1.0 : -1.0;

            if ((i % 2) == 1){
                j += 2;
            }
        }
    #endif
    #if USE_TYPE == TYPE_POD
        for (i = 0; i < HALF_UNIT_COUNT; i++){
            dCarrierOffsets(i) = i * LEVEL_SIZE;
            dCarrierOffsets(i  +  HALF_UNIT_COUNT)  =  -
(i + 1) * LEVEL_SIZE;

            nCounts(i) = 0;
            nCounts(i + HALF_UNIT_COUNT) = MAX_VALUE;

            nSign(i) = 1.0;
            nSign(i + HALF_UNIT_COUNT) = -1.0;
        }
    #endif

  }
#endif /*  MDL_START */


#define MDL_SET_DEFAULT_PORT_DATA_TYPES
static void mdlSetDefaultPortDataTypes(SimStruct *S){
  ssSetInputPortDataType(S, 0, SS_DOUBLE);
  ssSetOutputPortDataType(S, 0, SS_DOUBLE);
}


static void HandleCariers(){
    int i;

    for (i = 0; i < UNIT_COUNT; i++){
        if (nSign(i) == 0) nSign(i) = 1;

        nCounts(i) += nSign(i);

        if (nCounts(i) >= MAX_VALUE) nSign(i) = -1;
        if (nCounts(i) <= 0) nSign(i) = 1;

        dCariers(i)     =     dCarrierOffsets(i)     +
(nCounts(i) * LEVEL_SIZE / MAX_VALUE);
    }
}


static void HandleCurrentSign(){
    const  real_T    *PhaseCurrent   =  (const  real_T*)
ssGetInputPortSignal(baseStruct, 2);

    // get the current sign. If not set (value 0) set
the sign to pozitive
    if (PhaseCurrent[0] > 0.5) dCurSenseU = 1.0;
    if (PhaseCurrent[0] < -0.5) dCurSenseU = -1.0;
    if (dCurSenseU == 0) dCurSenseU = 1.0;

    if (PhaseCurrent[1] > 0.5) dCurSenseV = 1.0;
    if (PhaseCurrent[1] < -0.5) dCurSenseV = -1.0;
    if (dCurSenseV == 0) dCurSenseV = 1.0;

    if (PhaseCurrent[2] > 0.5) dCurSenseW = 1.0;
    if (PhaseCurrent[2] < -0.5) dCurSenseW = -1.0;
    if (dCurSenseW == 0) dCurSenseW = 1.0;
}



static void HandleUnitStates(){
    const   real_T      *  DesiredVoltage    =   (const
real_T*) ssGetInputPortSignal(baseStruct, 1);
    int i;
    int nMySign;

    #if OP_TYPE == OP_NONE
        for (i = 0; i < HALF_UNIT_COUNT; i++){
            // Set the unit states according with the
carrier and the desired voltage
            dOldUnitStateU(i)  =  (DesiredVoltage[0]  <
dCariers(i)) ? -1 : 1;
            dOldUnitStateV(i)  =  (DesiredVoltage[1]  <
dCariers(i)) ? -1 : 1;
            dOldUnitStateW(i)  =  (DesiredVoltage[2]  <
dCariers(i)) ? -1 : 1;

            dOldUnitStateU(HALF_UNIT_COUNT   +   i)   =
(DesiredVoltage[0] < dCariers(HALF_UNIT_COUNT + i)) ?
1 : -1;
            dOldUnitStateV(HALF_UNIT_COUNT   +   i)   =
(DesiredVoltage[1] < dCariers(HALF_UNIT_COUNT + i)) ?
1 : -1;
            dOldUnitStateW(HALF_UNIT_COUNT   +   i)   =
(DesiredVoltage[2] < dCariers(HALF_UNIT_COUNT + i)) ?
1 : -1;
        }
    #endif // OP_NONE
    #if OP_TYPE == OP_UNIT_CYCLE
        // Determine the New indeses
        if (nCyclesU > MAX_CYCLES_ZERO_CROSS){
            // Determine the sign
            nMySign = (DesiredVoltage[0] > 0) ? 1.0 :
-1.0;


            //   Now   let's   monitor   the   voltage
reference sign
            if (nSignVoltageU != nMySign){

            }
        }



        // Determine the unit states
        for (i = 0; i < HALF_UNIT_COUNT; i++){
            // Set the unit states according with the
carrier and the desired voltage
            dOldUnitStateU(nIndexsU(i))                =
(DesiredVoltage[0] < dCariers(i)) ? -1 : 1;
            dOldUnitStateV(nIndexsV(i))                =
(DesiredVoltage[1] < dCariers(i)) ? -1 : 1;
            dOldUnitStateW(nIndexsW(i))                =
(DesiredVoltage[2] < dCariers(i)) ? -1 : 1;

            dOldUnitStateU(nIndexsU(HALF_UNIT_COUNT  +
i)) = (DesiredVoltage[0] < dCariers(HALF_UNIT_COUNT +
i)) ? 1 : -1;
            dOldUnitStateV(nIndexsV(HALF_UNIT_COUNT  +
i)) = (DesiredVoltage[1] < dCariers(HALF_UNIT_COUNT +
i)) ? 1 : -1;
            dOldUnitStateW(nIndexsW(HALF_UNIT_COUNT  +
i)) = (DesiredVoltage[2] < dCariers(HALF_UNIT_COUNT +
i)) ? 1 : -1;
        }
    #endif // UNIT_CYCLE
}




static void HandleIGBTStates(){
    int i;
    double dNewStateU0, dNewStateU1;
    double dNewStateV0, dNewStateV1;
    double dNewStateW0, dNewStateW1;

    for (i = 0; i < UNIT_COUNT; i++){
        // Set the states according with the current
sign

        dNewStateU0   =   (dCurSenseU   >   0)   ?   1.0   :
((dOldUnitStateU(i) > 0) ? 1.0 : -1.0);
```

```
        dNewStateU1    =    (dCurSenseU    >    0)    ?
((dOldUnitStateU(i) > 0) ? 1.0 : -1.0) : 1.0;

        dNewStateV0  =  (dCurSenseV  >  0)  ?  1.0  :
((dOldUnitStateV(i) > 0) ? 1.0 : -1.0);
        dNewStateV1    =    (dCurSenseV    >    0)    ?
((dOldUnitStateV(i) > 0) ? 1.0 : -1.0) : 1.0;

        dNewStateW0  =  (dCurSenseW  >  0)  ?  1.0  :
((dOldUnitStateW(i) > 0) ? 1.0 : -1.0);
        dNewStateW1    =    (dCurSenseW    >    0)    ?
((dOldUnitStateW(i) > 0) ? 1.0 : -1.0) : 1.0;

        // Set state for leg 0 (towards positive)
        // - If idle (old state = 0) then take the
new state
        // - If different than new state than take 0
        // - Else 0
        if        (dOldLegStateU0(i)        ==        0){
dOldLegStateU0(i) = dNewStateU0; }
        else  if  (dOldLegStateU0(i)  !=  dNewStateU0)
dOldLegStateU0(i) = 0;

        if        (dOldLegStateV0(i)        ==        0){
dOldLegStateV0(i) = dNewStateV0; }
        else  if  (dOldLegStateV0(i)  !=  dNewStateV0)
dOldLegStateV0(i) = 0;

        if        (dOldLegStateW0(i)        ==        0){
dOldLegStateW0(i) = dNewStateW0; }
        else  if  (dOldLegStateW0(i)  !=  dNewStateW0)
dOldLegStateW0(i) = 0;


        // Set state for leg 0 (towards negative)
        if        (dOldLegStateU1(i)        ==        0){
dOldLegStateU1(i) = dNewStateU1; }
        else  if  (dOldLegStateU1(i)  !=  dNewStateU1)
dOldLegStateU1(i) = 0;

        if        (dOldLegStateV1(i)        ==        0){
dOldLegStateV1(i) = dNewStateV1; }
        else  if  (dOldLegStateV1(i)  !=  dNewStateV1)
dOldLegStateV1(i) = 0;

        if        (dOldLegStateW1(i)        ==        0){
dOldLegStateW1(i) = dNewStateW1; }
        else  if  (dOldLegStateW1(i)  !=  dNewStateW1)
dOldLegStateW1(i) = 0;
    }
}



/*                 Function:                 mdlOutputs
=====================================================
==
 * Abstract:
 *      In this function, you compute the outputs of
your S-function
 *      block. Generally outputs are placed in the
output vector, ssGetY(S).
 */
static void mdlOutputs(SimStruct *S, int_T tid){
    const  real_T     *  Enabled    =  (const  real_T*)
ssGetInputPortSignal(S,0);
    real_T                    *Cariers      =    (real_T
*)ssGetOutputPortRealSignal(S,0);
    real_T                      *UnitU      =    (real_T
*)ssGetOutputPortRealSignal(S,1);
    real_T                      *IgbtU      =    (real_T
*)ssGetOutputPortRealSignal(S,2);
    real_T                      *UnitV      =    (real_T
*)ssGetOutputPortRealSignal(S,3);
    real_T                      *IgbtV      =    (real_T
*)ssGetOutputPortRealSignal(S,4);
    real_T                      *UnitW      =    (real_T
*)ssGetOutputPortRealSignal(S,5);
    real_T                      *IgbtW      =    (real_T
*)ssGetOutputPortRealSignal(S,6);
```

```
    int i;          // Index counter

    // Retrieve global variables
    dValues = ssGetRWork(S);
    nValues = ssGetIWork(S);
    baseStruct = S;


    // If the sample time hit us ...
    if (ssIsSampleHit(S, 0, 0)){

        // handle the cariers
        HandleCariers();

        // handle the current sign
        HandleCurrentSign();

        // handle the unit states
        HandleUnitStates();

        //  Handle  IGBT  states  with  respect  to  the
Unit states
        HandleIGBTStates();
    }


    // Output the data
    for (i = 0; i < UNIT_COUNT; i++){
        Cariers[i] = dCariers(i);

        if (Enabled[0] != 0){
            UnitU[i] = dOldUnitStateU(i);
            UnitV[i] = dOldUnitStateV(i);
            UnitW[i] = dOldUnitStateW(i);

            IgbtU[i * 2 + 0] = dOldLegStateU0(i);
            IgbtU[i * 2 + 1] = dOldLegStateU1(i);

            IgbtV[i * 2 + 0] = dOldLegStateV0(i);
            IgbtV[i * 2 + 1] = dOldLegStateV1(i);

            IgbtW[i * 2 + 0] = dOldLegStateW0(i);
            IgbtW[i * 2 + 1] = dOldLegStateW1(i);
        } else {
            UnitU[i] = 0;
            UnitV[i] = 0;
            UnitW[i] = 0;

            IgbtU[i * 2 + 0] = 0;
            IgbtU[i * 2 + 1] = 0;

            IgbtV[i * 2 + 0] = 0;
            IgbtV[i * 2 + 1] = 0;

            IgbtW[i * 2 + 0] = 0;
            IgbtW[i * 2 + 1] = 0;
        }
    }

    //
    // END
    //
}


#undef  MDL_UPDATE   /*  Change  to  #undef  to  remove
function */
#if defined(MDL_UPDATE)
  /*                 Function:                 mdlUpdate
=====================================================
=
   * Abstract:
   *      This function is called once for every major
integration time step.
   *      Discrete states are typically updated here,
but this function is useful
   *      for performing any tasks that should only
take place once per
   *    integration step.
   */
  static void mdlUpdate(SimStruct *S, int_T tid){
  }
#endif /* MDL_UPDATE */



#undef MDL_DERIVATIVES  /* Change to #undef to remove
function */
#if defined(MDL_DERIVATIVES)
  /*              Function:             mdlDerivatives
=================================================
   * Abstract:
   *     In this function, you compute the S-function
block's derivatives.
   *     The derivatives are placed in the derivative
vector, ssGetdX(S).
   */
  static void mdlDerivatives(SimStruct *S){
  }
#endif /* MDL_DERIVATIVES */
```

```
/*               Function:              mdlTerminate
=====================================================
 * Abstract:
 *       In  this  function,  you  should  perform  any
actions that are necessary
 *       at  the  termination  of  a  simulation.   For
example, if memory was
 *      allocated  in  mdlStart,  this  is  the  place  to
free it.
 */
static void mdlTerminate(SimStruct *S){
}


#ifdef   MATLAB_MEX_FILE      /*  Is  this  file  being
compiled as a MEX-file? */
#include  "simulink.c"         /*  MEX-file  interface
mechanism */
#else
#include  "cg_sfun.h"            /*  Code  generation
registration function */
#endif
```

E.2 PHASE SHIFTED

```
// PhaseShifted
//
// Phase Shifted modulation
//
// Copyright:
//    Sandu Cristian - 2008
//    sanducristian@gmail.com
//       Code  created  for  the  Project  of  the  10th
semester at
//    Aalborg University
//



#define S_FUNCTION_NAME  PhaseShifted
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#ifdef MATLAB_MEX_FILE
#include <math.h>
#endif


#define                                    INV_SQRT_3
0.57735026918962576450914878050196
#define SATURATE(val, min, max)      { (val) = ((val)
< (min) ? (min) : ((val) > (max) ? (max) : (val))); }
#define MIN(a, b, c)                    (((a) < (b) ?
((a) < (c) ? (a) : (c)) : ((b) < (c) ? (b) : (c))))
#define MAX(a, b, c)                    (((a) > (b) ?
((a) > (c) ? (a) : (c)) : ((b) > (c) ? (b) : (c))))
#define SATURATION_MIN              (0.02)
#define SATURATION_MAX              (0.98)

#define SWITCHING_FREQUENCY         (200)
#define HALF_UNIT_COUNT             (4)
#define  UNIT_COUNT                          (2  *
HALF_UNIT_COUNT)
#define MAX_VALUE                   (200)
#define  SAMPLE_TIME                       (1.0  /
(SWITCHING_FREQUENCY * MAX_VALUE))

#define COUNT_DELTA                 (MAX_VALUE * 2.0
/ UNIT_COUNT)



double * dValues;             // Global double values
for the class
int * nValues;               // Global integer values
for the class
SimStruct * baseStruct;

#define nCounts(a)        nValues[(0  *  UNIT_COUNT)  +
(a)]
#define nSign(a)          nValues[(1  *  UNIT_COUNT)  +
(a)]

// Only positive carriers
#define dCariers(a)         dValues[( 0 * UNIT_COUNT)
+ (a)]

#define dCurSenseU          dValues[( 1 * UNIT_COUNT)
+ 0]
#define dCurSenseV          dValues[( 1 * UNIT_COUNT)
+ 1]
#define dCurSenseW          dValues[( 1 * UNIT_COUNT)
+ 2]

#define dOldUnitStateU(a)   dValues[( 2 * UNIT_COUNT)
+ (a)]
#define dOldUnitStateV(a)   dValues[( 3 * UNIT_COUNT)
+ (a)]
#define dOldUnitStateW(a)   dValues[( 4 * UNIT_COUNT)
+ (a)]

#define dOldLegStateU0(a)   dValues[( 5 * UNIT_COUNT)
+ i]
#define dOldLegStateU1(a)   dValues[( 6 * UNIT_COUNT)
+ i]
#define dOldLegStateV0(a)   dValues[( 7 * UNIT_COUNT)
+ i]
#define dOldLegStateV1(a)   dValues[( 8 * UNIT_COUNT)
+ i]
#define dOldLegStateW0(a)   dValues[( 9 * UNIT_COUNT)
+ i]
#define dOldLegStateW1(a)   dValues[(10 * UNIT_COUNT)
+ i]


/*====================*
 * S-function methods *
 *====================*/

/*            Function:            mdlInitializeSizes
===============================================
 * Abstract:
 *     The sizes information is used by Simulink to
determine the S-function
 *      block's characteristics (number of inputs,
outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S){
    ssSetNumSFcnParams(S, 0);   /* Number of expected
parameters */
    if             (ssGetNumSFcnParams(S)            !=
ssGetSFcnParamsCount(S)) {
        /* Return if number of expected != number of
actual parameters */
        return;
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, 3)) return;
    /*Input Port 0 */
    ssSetInputPortWidth(S,  0, 1); /* Enabled */
    ssSetInputPortDataType(S, 0, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  0, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortRequiredContiguous(S,    0,     1);
/*direct input signal access*/

    /*Input Port 0 */
    ssSetInputPortWidth(S,     1,   3);   /*  Desired
voltages */
    ssSetInputPortDataType(S, 1, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  1, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S, 1, 1);
    ssSetInputPortRequiredContiguous(S,    1,     1);
/*direct input signal access*/

    /*Input Port 0 */
    ssSetInputPortWidth(S,   2,  3); /* Phase currents
*/
    ssSetInputPortDataType(S, 2, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  2, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S, 2, 1);
    ssSetInputPortRequiredContiguous(S,    2,     1);
/*direct input signal access*/

    /*
     * Set direct feedthrough flag (1=yes, 0=no).
     * A port has direct feedthrough if the input is
used in either
     *   the   mdlOutputs   or   mdlGetTimeOfNextVarHit
functions.
     *                                            See
matlabroot/simulink/src/sfuntmpl_directfeed.txt.
     */
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 7)) return;

    /* Output Port 0 - Carriers */
    ssSetOutputPortWidth(S, 0, UNIT_COUNT);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 0, COMPLEX_NO);

    /* Output Port 1 - Unit U states */
    ssSetOutputPortWidth(S, 1, UNIT_COUNT);
    ssSetOutputPortDataType(S, 1, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 1, COMPLEX_NO);

    /* Output Port 2 - IGBT U States */
    ssSetOutputPortWidth(S, 2, 2 * UNIT_COUNT);
    ssSetOutputPortDataType(S, 2, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 2, COMPLEX_NO);

    /* Output Port 3 - Unit V states */
    ssSetOutputPortWidth(S, 3, UNIT_COUNT);
    ssSetOutputPortDataType(S, 3, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 3, COMPLEX_NO);

    /* Output Port 4 - IGBT V States */
    ssSetOutputPortWidth(S, 4, 2 * UNIT_COUNT);
    ssSetOutputPortDataType(S, 4, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 4, COMPLEX_NO);

    /* Output Port 5 - Unit W states */
    ssSetOutputPortWidth(S, 5, UNIT_COUNT);
    ssSetOutputPortDataType(S, 5, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 5, COMPLEX_NO);

    /* Output Port 6 - IGBT W States */
    ssSetOutputPortWidth(S, 6, 2 * UNIT_COUNT);
    ssSetOutputPortDataType(S, 6, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 6, COMPLEX_NO);



    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, UNIT_COUNT * 12);
    ssSetNumIWork(S, UNIT_COUNT * 2);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    ssSetOptions(S, 0);
}


/*         Function:         mdlInitializeSampleTimes
=========================================
 * Abstract:
 *      This function is used to specify the sample
time(s) for your
 *     S-function. You must register the same number
of sample times as
 *    specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct *S){
    ssSetSampleTime(S, 0, SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}



#define  MDL_INITIALIZE_CONDITIONS      /*  Change  to
#undef to remove function */
#if defined(MDL_INITIALIZE_CONDITIONS)
  static void mdlInitializeConditions(SimStruct *S){
  }
#endif /* MDL_INITIALIZE_CONDITIONS */



#define  MDL_START   /*  Change  to  #undef  to  remove
function */
#if defined(MDL_START)
  /*                Function:                mdlStart
=====================================================
==
   * Abstract:
   *      This  function  is  called  once  at  start  of
model execution. If you
   *     have states that should be initialized once,
this is the place
   *    to do it.
   */
  static void mdlStart(SimStruct *S) {
  }
#endif /*  MDL_START */


#define MDL_SET_DEFAULT_PORT_DATA_TYPES
static void mdlSetDefaultPortDataTypes(SimStruct *S){
  ssSetInputPortDataType(S, 0, SS_DOUBLE);
  ssSetOutputPortDataType(S, 0, SS_DOUBLE);
}


static void HandleCariers(){
    int i;


    if  ((nCounts(0)  ==  nCounts(1))  &&  (nCounts(0)  ==
0)){
        for (i = 1; i < UNIT_COUNT; i++){
            if (i % 2 == 0){
                nCounts(i) = nCounts(i - 1);
                nSign(i) = -1;
            } else {
                nCounts(i)    =    nCounts(i    -    1)    +
COUNT_DELTA;
                nSign(i) = 1;
            }

            if (nCounts(i) == MAX_VALUE){
                nSign(i) = -1;
            }
        }
    }

    for (i = 0; i < UNIT_COUNT; i++){
        if (nSign(i) == 0) nSign(i) = 1;

        nCounts(i) += nSign(i);

        if (nCounts(i) >= MAX_VALUE) nSign(i) = -1;
        if (nCounts(i) <= 0) nSign(i) = 1;

        dCariers(i) = 1.0 - (nCounts(i) * 2.0 / 200);
    }
}


static void HandleCurrentSign(){
    const  real_T     *PhaseCurrent    =  (const  real_T*)
ssGetInputPortSignal(baseStruct, 2);

    // get the current sign. If not set (value 0) set
the sign to pozitive
    if (PhaseCurrent[0] > 0.5) dCurSenseU = 1.0;
    if (PhaseCurrent[0] < -0.5) dCurSenseU = -1.0;
    if (dCurSenseU == 0) dCurSenseU = 1.0;

    if (PhaseCurrent[1] > 0.5) dCurSenseV = 1.0;
    if (PhaseCurrent[1] < -0.5) dCurSenseV = -1.0;
    if (dCurSenseV == 0) dCurSenseV = 1.0;

    if (PhaseCurrent[2] > 0.5) dCurSenseW = 1.0;
    if (PhaseCurrent[2] < -0.5) dCurSenseW = -1.0;
    if (dCurSenseW == 0) dCurSenseW = 1.0;
}


static void HandleUnitStates(){
    const   real_T       *  DesiredVoltage     =   (const
real_T*) ssGetInputPortSignal(baseStruct, 1);
```

```
    int i;

    for (i = 0; i < HALF_UNIT_COUNT; i++){
        // Set the unit states according with the
carrier and the desired voltage
        dOldUnitStateU(i) = (DesiredVoltage[0] >=
dCariers(i)) ? 1 : -1;
        dOldUnitStateV(i) = (DesiredVoltage[1] >=
dCariers(i)) ? 1 : -1;
        dOldUnitStateW(i) = (DesiredVoltage[2] >=
dCariers(i)) ? 1 : -1;

        dOldUnitStateU(HALF_UNIT_COUNT + i) = 0 -
dOldUnitStateU(i);
        dOldUnitStateV(HALF_UNIT_COUNT + i) = 0 -
dOldUnitStateV(i);
        dOldUnitStateW(HALF_UNIT_COUNT + i) = 0 -
dOldUnitStateW(i);
    }
}



static void HandleIGBTStates(){
    int i;
    double dNewStateU0, dNewStateU1;
    double dNewStateV0, dNewStateV1;
    double dNewStateW0, dNewStateW1;

    for (i = 0; i < UNIT_COUNT; i++){
        // Set the states according with the current
sign
        dNewStateU0 = 1;
        dNewStateU1 = (dOldUnitStateU(i) > 0) ? 1.0 :
-1.0;

        dNewStateV0 = 1;
        dNewStateV1 = (dOldUnitStateV(i) > 0) ? 1.0 :
-1.0;

        dNewStateW0 = 1;
        dNewStateW1 = (dOldUnitStateW(i) > 0) ? 1.0 :
-1.0;
/*
        dNewStateU0 = (dCurSenseU > 0) ? 1.0 :
((dOldUnitStateU(i) > 0) ? 1.0 : -1.0);
        dNewStateU1 = (dCurSenseU > 0) ?
((dOldUnitStateU(i) > 0) ? 1.0 : -1.0) : 1.0;

        dNewStateV0 = (dCurSenseV > 0) ? 1.0 :
((dOldUnitStateV(i) > 0) ? 1.0 : -1.0);
        dNewStateV1 = (dCurSenseV > 0) ?
((dOldUnitStateV(i) > 0) ? 1.0 : -1.0) : 1.0;

        dNewStateW0 = (dCurSenseW > 0) ? 1.0 :
((dOldUnitStateW(i) > 0) ? 1.0 : -1.0);
        dNewStateW1 = (dCurSenseW > 0) ?
((dOldUnitStateW(i) > 0) ? 1.0 : -1.0) : 1.0;
*/
        // Set state for leg 0 (towards positive)
        // - If idle (old state = 0) then take the
new state
        // - If different than new state than take 0
        // - Else 0
        if (dOldLegStateU0(i) == 0){
dOldLegStateU0(i) = dNewStateU0; }
        else if (dOldLegStateU0(i) != dNewStateU0)
dOldLegStateU0(i) = 0;

        if (dOldLegStateV0(i) == 0){
dOldLegStateV0(i) = dNewStateV0; }
        else if (dOldLegStateV0(i) != dNewStateV0)
dOldLegStateV0(i) = 0;

        if (dOldLegStateW0(i) == 0){
dOldLegStateW0(i) = dNewStateW0; }
        else if (dOldLegStateW0(i) != dNewStateW0)
dOldLegStateW0(i) = 0;
```

```
        // Set state for leg 0 (towards negative)
        if (dOldLegStateU1(i) == 0){
dOldLegStateU1(i) = dNewStateU1; }
        else if (dOldLegStateU1(i) != dNewStateU1)
dOldLegStateU1(i) = 0;

        if (dOldLegStateV1(i) == 0){
dOldLegStateV1(i) = dNewStateV1; }
        else if (dOldLegStateV1(i) != dNewStateV1)
dOldLegStateV1(i) = 0;

        if (dOldLegStateW1(i) == 0){
dOldLegStateW1(i) = dNewStateW1; }
        else if (dOldLegStateW1(i) != dNewStateW1)
dOldLegStateW1(i) = 0;
    }
}



/*              Function:              mdlOutputs
=======================================================
==
 * Abstract:
 *    In this function, you compute the outputs of
your S-function
 *    block. Generally outputs are placed in the
output vector, ssGetY(S).
 */
static void mdlOutputs(SimStruct *S, int_T tid){
    const real_T    * Enabled   = (const real_T*)
ssGetInputPortSignal(S,0);
    real_T                *Cariers   = (real_T
*)ssGetOutputPortRealSignal(S,0);
    real_T                 *UnitU    = (real_T
*)ssGetOutputPortRealSignal(S,1);
    real_T                 *IgbtU    = (real_T
*)ssGetOutputPortRealSignal(S,2);
    real_T                 *UnitV    = (real_T
*)ssGetOutputPortRealSignal(S,3);
    real_T                 *IgbtV    = (real_T
*)ssGetOutputPortRealSignal(S,4);
    real_T                 *UnitW    = (real_T
*)ssGetOutputPortRealSignal(S,5);
    real_T                 *IgbtW    = (real_T
*)ssGetOutputPortRealSignal(S,6);

    int i;        // Index counter

    // Retrieve global variables
    dValues = ssGetRWork(S);
    nValues = ssGetIWork(S);
    baseStruct = S;


    // If the sample time hit us ...
    if (ssIsSampleHit(S, 0, 0)){

        // handle the cariers
        HandleCariers();

        // handle the current sign
        HandleCurrentSign();

        // handle the unit states
        HandleUnitStates();

        // Handle IGBT states with respect to the
Unit states
        HandleIGBTStates();
    }


    // Output the data
    for (i = 0; i < UNIT_COUNT; i++){
        Cariers[i] = dCariers(i);

        if (Enabled[0] != 0){
            UnitU[i] = dOldUnitStateU(i);
            UnitV[i] = dOldUnitStateV(i);
            UnitW[i] = dOldUnitStateW(i);

            IgbtU[i * 2 + 0] = dOldLegStateU0(i);
            IgbtU[i * 2 + 1] = dOldLegStateU1(i);

            IgbtV[i * 2 + 0] = dOldLegStateV0(i);
            IgbtV[i * 2 + 1] = dOldLegStateV1(i);

            IgbtW[i * 2 + 0] = dOldLegStateW0(i);
            IgbtW[i * 2 + 1] = dOldLegStateW1(i);
        } else {
            UnitU[i] = 0;
            UnitV[i] = 0;
            UnitW[i] = 0;

            IgbtU[i * 2 + 0] = 0;
            IgbtU[i * 2 + 1] = 0;

            IgbtV[i * 2 + 0] = 0;
            IgbtV[i * 2 + 1] = 0;

            IgbtW[i * 2 + 0] = 0;
            IgbtW[i * 2 + 1] = 0;
        }
    }

    //
    // END
    //
}



#undef  MDL_UPDATE   /*  Change  to  #undef  to  remove
function */
#if defined(MDL_UPDATE)
  /*                Function:                mdlUpdate
=====================================================
=
   * Abstract:
   *     This function is called once for every major
integration time step.
   *      Discrete states are typically updated here,
but this function is useful
   *      for performing any tasks that should only
take place once per
   *    integration step.
   */
  static void mdlUpdate(SimStruct *S, int_T tid){
  }
#endif /* MDL_UPDATE */



#undef MDL_DERIVATIVES  /* Change to #undef to remove
function */
#if defined(MDL_DERIVATIVES)
  /*              Function:             mdlDerivatives
=================================================
   * Abstract:
   *     In this function, you compute the S-function
block's derivatives.
   *     The derivatives are placed in the derivative
vector, ssGetdX(S).
   */
  static void mdlDerivatives(SimStruct *S){
  }
#endif /* MDL_DERIVATIVES */



/*               Function:               mdlTerminate
=====================================================
 * Abstract:
 *       In  this  function,  you  should  perform  any
actions that are necessary
 *       at  the  termination  of  a  simulation.   For
example, if memory was
 *      allocated  in  mdlStart,  this  is  the  place  to
free it.
 */
static void mdlTerminate(SimStruct *S){
}


#ifdef   MATLAB_MEX_FILE      /*  Is  this  file  being
compiled as a MEX-file? */
#include  "simulink.c"         /*  MEX-file  interface
mechanism */
#else
#include  "cg_sfun.h"            /*  Code  generation
registration function */
#endif
```

E.3 STAIRCASE

```
// Staircase
//
// Staircase modulation
//
// Copyright:
//    Sandu Cristian - 2008
//    sanducristian@gmail.com
//       Code  created  for  the  Project  of  the  10th
semester at
//    Aalborg University
//



#define S_FUNCTION_NAME  Staircase
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#ifdef MATLAB_MEX_FILE
#include <math.h>
#endif


#define                                         INV_SQRT_3
0.57735026918962576450914878050196
#define  SATURATE(val,  min,  max)       {  (val)  =  ((val)
<  (min)  ?  (min)  :  ((val)  >  (max)  ?  (max)  :  (val))); }
#define  MIN(a,  b,  c)                     (((a)  <  (b)  ?
((a)  <  (c)  ?  (a)  :  (c))  :  ((b)  <  (c)  ?  (b)  :  (c))))
#define  MAX(a,  b,  c)                     (((a)  >  (b)  ?
((a)  >  (c)  ?  (a)  :  (c))  :  ((b)  >  (c)  ?  (b)  :  (c))))
#define SATURATION_MIN                 (0.02)
#define SATURATION_MAX                 (0.98)

#define SWITCHING_FREQUENCY            (6000)
#define HALF_UNIT_COUNT                (4)
#define  UNIT_COUNT                                  (2  *
HALF_UNIT_COUNT)
#define  SAMPLE_TIME                                (1.0  /
SWITCHING_FREQUENCY)

#define COUNT_DELTA                     (MAX_VALUE * 2.0
/ UNIT_COUNT)
#define  STEP_OFFSET                                (1.0  /
HALF_UNIT_COUNT)


#define PORTIN_ENABLE                  (0)
#define PORTIN_DESIRED_VOLTAGE         (1)
#define PORTIN_COEFFICIENT             (2)
#define PORTIN_UNIT_VOTLAGES_U         (3)
#define PORTIN_UNIT_VOTLAGES_V         (4)
#define PORTIN_UNIT_VOTLAGES_W         (5)
#define PORTIN_CURRENTS                (6)
#define PORTIN_VOLTAGES                (7)

#define PORTIN__COUNT                  (8)



#define PORTOUT_LEVELS                 (0)
#define PORTOUT_SORT_U                 (1)
#define PORTOUT_UNITSTATE_U            (2)
#define PORTOUT_IGBTSTATE_U            (3)
#define PORTOUT_SORT_V                 (4)
#define PORTOUT_UNITSTATE_V            (5)
#define PORTOUT_IGBTSTATE_V            (6)
#define PORTOUT_SORT_W                 (7)
#define PORTOUT_UNITSTATE_W            (8)
#define PORTOUT_IGBTSTATE_W            (9)
#define PORTOUT_QUADRANTS              (10)

#define PORTOUT__COUNT                 (11)


double * dValues;              // Global double values
for the class
int * nValues;                // Global integer values
for the class
SimStruct * baseStruct;
double dMainSteps[UNIT_COUNT + 1];

#define  nSortedU(a)        nValues[(0  *  UNIT_COUNT)  +
(a)]
#define  nSortedV(a)        nValues[(1  *  UNIT_COUNT)  +
(a)]
#define  nSortedW(a)        nValues[(2  *  UNIT_COUNT)  +
(a)]


#define nQuadrantU      nValues[(3 * UNIT_COUNT) + 0]
#define nQuadrantV      nValues[(3 * UNIT_COUNT) + 1]
#define nQuadrantW      nValues[(3 * UNIT_COUNT) + 2]

// Only positive carriers
#define  dUnitStateU(a)                 dValues[(  0  *
UNIT_COUNT) + (a)]
#define  dUnitStateV(a)                 dValues[(  1  *
UNIT_COUNT) + (a)]
#define  dUnitStateW(a)                 dValues[(  2  *
UNIT_COUNT) + (a)]

#define  dPhaseLevelU                   dValues[(  3  *
UNIT_COUNT) +  0]
#define  dPhaseLevelV                   dValues[(  3  *
UNIT_COUNT) +  1]
#define  dPhaseLevelW                   dValues[(  3  *
UNIT_COUNT) +  2]

#define  dCurSenseU                     dValues[(  4  *
UNIT_COUNT) +  0]
#define  dCurSenseV                     dValues[(  4  *
UNIT_COUNT) +  1]
#define  dCurSenseW                     dValues[(  4  *
UNIT_COUNT) +  2]

#define  dOldLegStateU0(a)             dValues[(  5  *
UNIT_COUNT) + i]
#define  dOldLegStateU1(a)             dValues[(  6  *
UNIT_COUNT) + i]
#define  dOldLegStateV0(a)             dValues[(  7  *
UNIT_COUNT) + i]
#define  dOldLegStateV1(a)             dValues[(  8  *
UNIT_COUNT) + i]
#define  dOldLegStateW0(a)             dValues[(  9  *
UNIT_COUNT) + i]
#define  dOldLegStateW1(a)              dValues[(10  *
UNIT_COUNT) + i]




/*====================*
 * S-function methods *
 *====================*/

/*             Function:             mdlInitializeSizes
===============================================
 * Abstract:
 *      The  sizes  information  is  used  by  Simulink  to
determine the S-function
 *       block's  characteristics  (number  of  inputs,
outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S){
    int i;

    ssSetNumSFcnParams(S,  0);   /*  Number  of  expected
parameters */
    if           (ssGetNumSFcnParams(S)              !=
ssGetSFcnParamsCount(S)) {
        /*  Return  if  number  of  expected  !=  number  of
actual parameters */
        return;
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if     (!ssSetNumInputPorts(S,     PORTIN__COUNT))
return;
    /*Input Port 0 */
    ssSetInputPortWidth(S,     PORTIN_ENABLE,   1);   /*
Enabled */
    ssSetInputPortDataType(S,            PORTIN_ENABLE,
SS_DOUBLE);
    ssSetInputPortComplexSignal(S,      PORTIN_ENABLE,
COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S,  PORTIN_ENABLE,
1);
    ssSetInputPortRequiredContiguous(S,
PORTIN_ENABLE, 1); /*direct input signal access*/

    /*Input Port 0 */
    ssSetInputPortWidth(S,     PORTIN_DESIRED_VOLTAGE,
3); /* Desired voltages */
    ssSetInputPortDataType(S,  PORTIN_DESIRED_VOLTAGE,
SS_DOUBLE);
    ssSetInputPortComplexSignal(S,
PORTIN_DESIRED_VOLTAGE, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S,
PORTIN_DESIRED_VOLTAGE, 1);
    ssSetInputPortRequiredContiguous(S,
PORTIN_DESIRED_VOLTAGE,   1);  /*direct  input  signal
access*/

    /* Input Port X */
    ssSetInputPortWidth(S,    PORTIN_COEFFICIENT,  1);
/*  */
    ssSetInputPortDataType(S,      PORTIN_COEFFICIENT,
SS_DOUBLE);
    ssSetInputPortComplexSignal(S,
PORTIN_COEFFICIENT, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S,
PORTIN_COEFFICIENT, 1);
    ssSetInputPortRequiredContiguous(S,
PORTIN_COEFFICIENT,    1);   /*direct   input   signal
access*/

    /* Input Port X */
    ssSetInputPortWidth(S,     PORTIN_UNIT_VOTLAGES_U,
UNIT_COUNT); /* Phase units voltages  */
    ssSetInputPortDataType(S,  PORTIN_UNIT_VOTLAGES_U,
SS_DOUBLE);
    ssSetInputPortComplexSignal(S,
PORTIN_UNIT_VOTLAGES_U, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S,
PORTIN_UNIT_VOTLAGES_U, 1);
    ssSetInputPortRequiredContiguous(S,
PORTIN_UNIT_VOTLAGES_U,   1);  /*direct  input  signal
access*/

        /* Input Port X */
    ssSetInputPortWidth(S,     PORTIN_UNIT_VOTLAGES_V,
UNIT_COUNT); /* Phase units voltages  */
    ssSetInputPortDataType(S,  PORTIN_UNIT_VOTLAGES_V,
SS_DOUBLE);
    ssSetInputPortComplexSignal(S,
PORTIN_UNIT_VOTLAGES_V, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S,
PORTIN_UNIT_VOTLAGES_V, 1);
    ssSetInputPortRequiredContiguous(S,
PORTIN_UNIT_VOTLAGES_V,   1);  /*direct  input  signal
access*/

        /* Input Port X */
    ssSetInputPortWidth(S,     PORTIN_UNIT_VOTLAGES_W,
UNIT_COUNT); /* Phase units voltages  */
    ssSetInputPortDataType(S,  PORTIN_UNIT_VOTLAGES_W,
SS_DOUBLE);
    ssSetInputPortComplexSignal(S,
PORTIN_UNIT_VOTLAGES_W, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S,
PORTIN_UNIT_VOTLAGES_W, 1);
    ssSetInputPortRequiredContiguous(S,
PORTIN_UNIT_VOTLAGES_W,   1);  /*direct  input  signal
access*/

        /* Input Port X */
    ssSetInputPortWidth(S,    PORTIN_CURRENTS,  3);  /*
Phase currents  */
    ssSetInputPortDataType(S,         PORTIN_CURRENTS,
SS_DOUBLE);
    ssSetInputPortComplexSignal(S,    PORTIN_CURRENTS,
COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S,
PORTIN_CURRENTS, 1);
    ssSetInputPortRequiredContiguous(S,
PORTIN_CURRENTS, 1); /*direct input signal access*/

        /* Input Port X */
    ssSetInputPortWidth(S,    PORTIN_VOLTAGES,  3);  /*
Phase voltages  */
    ssSetInputPortDataType(S,         PORTIN_VOLTAGES,
SS_DOUBLE);
    ssSetInputPortComplexSignal(S,    PORTIN_VOLTAGES,
COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S,
PORTIN_VOLTAGES, 1);
    ssSetInputPortRequiredContiguous(S,
PORTIN_VOLTAGES, 1); /*direct input signal access*/

    /*
     * Set direct feedthrough flag (1=yes, 0=no).
     * A port has direct feedthrough if the input is
used in either
     *  the   mdlOutputs   or   mdlGetTimeOfNextVarHit
functions.
     *                                             See
matlabroot/simulink/src/sfuntmpl_directfeed.txt.
     */
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if    (!ssSetNumOutputPorts(S,    PORTOUT__COUNT))
return;

    /* Output Port 0 - Carriers */
    ssSetOutputPortWidth(S, PORTOUT_LEVELS, 3);
    ssSetOutputPortDataType(S,          PORTOUT_LEVELS,
SS_DOUBLE);
    ssSetOutputPortComplexSignal(S,    PORTOUT_LEVELS,
COMPLEX_NO);

    /* Output Port 1 - Unit U Sorted values */
    ssSetOutputPortWidth(S,            PORTOUT_SORT_U,
UNIT_COUNT);
    ssSetOutputPortDataType(S,         PORTOUT_SORT_U,
SS_DOUBLE);
    ssSetOutputPortComplexSignal(S,    PORTOUT_SORT_U,
COMPLEX_NO);

    /* Output Port 2 - IGBT U States */
    ssSetOutputPortWidth(S,       PORTOUT_UNITSTATE_U,
UNIT_COUNT);
    ssSetOutputPortDataType(S,    PORTOUT_UNITSTATE_U,
SS_DOUBLE);
    ssSetOutputPortComplexSignal(S,
PORTOUT_UNITSTATE_U, COMPLEX_NO);

    /* Output Port 3 - Unit V states */
    ssSetOutputPortWidth(S,  PORTOUT_IGBTSTATE_U,  2  *
UNIT_COUNT);
    ssSetOutputPortDataType(S,    PORTOUT_IGBTSTATE_U,
SS_DOUBLE);
    ssSetOutputPortComplexSignal(S,
PORTOUT_IGBTSTATE_U, COMPLEX_NO);


    /* Output Port 1 - Unit V Sorted values */
    ssSetOutputPortWidth(S,            PORTOUT_SORT_V,
UNIT_COUNT);
    ssSetOutputPortDataType(S,         PORTOUT_SORT_V,
SS_DOUBLE);
    ssSetOutputPortComplexSignal(S,    PORTOUT_SORT_V,
COMPLEX_NO);

    /* Output Port 2 - Unit V states */
    ssSetOutputPortWidth(S,       PORTOUT_UNITSTATE_V,
UNIT_COUNT);
    ssSetOutputPortDataType(S,    PORTOUT_UNITSTATE_V,
SS_DOUBLE);
    ssSetOutputPortComplexSignal(S,
PORTOUT_UNITSTATE_V, COMPLEX_NO);

    /* Output Port 3 - Unit V IGBT states */
    ssSetOutputPortWidth(S,  PORTOUT_IGBTSTATE_V,  2  *
UNIT_COUNT);
    ssSetOutputPortDataType(S,    PORTOUT_IGBTSTATE_V,
SS_DOUBLE);
    ssSetOutputPortComplexSignal(S,
PORTOUT_IGBTSTATE_V, COMPLEX_NO);



    /* Output Port 1 - Unit W Sorted values */
    ssSetOutputPortWidth(S,            PORTOUT_SORT_W,
UNIT_COUNT);
    ssSetOutputPortDataType(S,         PORTOUT_SORT_W,
SS_DOUBLE);
    ssSetOutputPortComplexSignal(S,    PORTOUT_SORT_W,
COMPLEX_NO);

    /* Output Port 2 - Unit W unit states */
    ssSetOutputPortWidth(S,       PORTOUT_UNITSTATE_W,
UNIT_COUNT);
    ssSetOutputPortDataType(S,    PORTOUT_UNITSTATE_W,
SS_DOUBLE);
    ssSetOutputPortComplexSignal(S,
PORTOUT_UNITSTATE_W, COMPLEX_NO);

    /* Output Port 3 - Unit W igbt states */
    ssSetOutputPortWidth(S,  PORTOUT_IGBTSTATE_W,  2  *
UNIT_COUNT);
    ssSetOutputPortDataType(S,    PORTOUT_IGBTSTATE_W,
SS_DOUBLE);
    ssSetOutputPortComplexSignal(S,
PORTOUT_IGBTSTATE_W, COMPLEX_NO);

    ssSetOutputPortWidth(S, PORTOUT_QUADRANTS, 3);
    ssSetOutputPortDataType(S,      PORTOUT_QUADRANTS,
SS_DOUBLE);
    ssSetOutputPortComplexSignal(S,
PORTOUT_QUADRANTS, COMPLEX_NO);


    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, UNIT_COUNT * 15);
    ssSetNumIWork(S, UNIT_COUNT * 10);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    ssSetOptions(S, 0);
}


/*         Function:         mdlInitializeSampleTimes
=========================================
 * Abstract:
 *      This  function  is  used  to  specify  the  sample
time(s) for your
 *      S-function. You  must  register  the  same  number
of sample times as
 *    specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct *S){
    ssSetSampleTime(S, 0, SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}



#define  MDL_INITIALIZE_CONDITIONS      /*  Change  to
#undef to remove function */
#if defined(MDL_INITIALIZE_CONDITIONS)
  static void mdlInitializeConditions(SimStruct *S){
  }
#endif /* MDL_INITIALIZE_CONDITIONS */


#define  MDL_START    /*  Change  to  #undef  to  remove
function */
#if defined(MDL_START)
  /*                Function:                  mdlStart
=====================================================
==
   * Abstract:
   *      This  function  is  called  once  at  start  of
model execution. If you
   *     have states that should be initialized once,
this is the place
   *    to do it.
   */
  static void mdlStart(SimStruct *S)  {
    int i;

    // Initialize dMainSteps
    dMainSteps[0] = 1;
    for (i = 1; i < HALF_UNIT_COUNT + 1; i++){
        dMainSteps[i]    =    dMainSteps[i    -    1]    -
STEP_OFFSET * 2;
    }
  }
#endif /*  MDL_START */


#define MDL_SET_DEFAULT_PORT_DATA_TYPES
static void mdlSetDefaultPortDataTypes(SimStruct *S){
  ssSetInputPortDataType(S, 0, SS_DOUBLE);
  ssSetOutputPortDataType(S, 0, SS_DOUBLE);
}


static void StaircasePulses(){
    const   real_T      *  DesiredVoltage     =   (const
real_T*)         ssGetInputPortSignal(baseStruct,
PORTIN_DESIRED_VOLTAGE);
    const  real_T    * Coefficient   =  (const  real_T*)
ssGetInputPortSignal(baseStruct, PORTIN_COEFFICIENT);

    double nStepSize = 0;
    char bSetU = 0;
    char bSetV = 0;
    char bSetW = 0;
    int i;


    dPhaseLevelU = -1;
    dPhaseLevelV = -1;
    dPhaseLevelW = -1;

    bSetU = 0;
    bSetV = 0;
    bSetW = 0;


    // Determine the step size
    if (DesiredVoltage[0] < 0){
        nStepSize     =     STEP_OFFSET     *     (1     -
Coefficient[0]);
    } else {
        nStepSize = STEP_OFFSET * Coefficient[0];
    }

    //  Determine  the  sector  ID  in  which  the  current
voltage level is found
    for (i = 1; i < HALF_UNIT_COUNT + 1; i++){
        if  ((DesiredVoltage[0]   >   (dMainSteps[i]   +
nStepSize)) && (bSetU == 0)){
            dPhaseLevelU  =  (dMainSteps[i - 1]  +  1)  *
HALF_UNIT_COUNT * 0.5;
            bSetU = 1;
        }
        if  ((DesiredVoltage[1]   >   (dMainSteps[i]   +
nStepSize)) && (bSetV == 0)){
            dPhaseLevelV  =  (dMainSteps[i - 1]  +  1)  *
HALF_UNIT_COUNT * 0.5;
```

```
            bSetV = 1;
        }
        if  ((DesiredVoltage[2]  >  (dMainSteps[i]  +
nStepSize)) && (bSetW == 0)){
            dPhaseLevelW = (dMainSteps[i - 1] + 1) *
HALF_UNIT_COUNT * 0.5;
            bSetW = 1;
        }
    }

    if (bSetU == 0) dPhaseLevelU = (dPhaseLevelU + 1)
* HALF_UNIT_COUNT * 0.5;
    if (bSetV == 0) dPhaseLevelV = (dPhaseLevelV + 1)
* HALF_UNIT_COUNT * 0.5;
    if (bSetW == 0) dPhaseLevelW = (dPhaseLevelW + 1)
* HALF_UNIT_COUNT * 0.5;
}



static void BubleSort(const real_T * dInput, int *
nOutput){
    int nIndexHi[HALF_UNIT_COUNT];
    int nIndexLo[HALF_UNIT_COUNT];
    int nAux;
    int i, j;

    // Copy the values
    for (i = 0; i < HALF_UNIT_COUNT; i++){
        nIndexHi[i] = i;
        nIndexLo[i] = i;
    }

    // Sort the voltages for the upper part and lower
part
    for (i = 0; i < HALF_UNIT_COUNT - 1; i++){
        for (j = i + 1; j < HALF_UNIT_COUNT; j++){
            // Sort for the upper part
            if          (dInput[nIndexHi[i]]          <
dInput[nIndexHi[j]]){
                nAux = nIndexHi[i];
                nIndexHi[i] = nIndexHi[j];
                nIndexHi[j] = nAux;
            }

            // Sort for the lower part
            if (dInput[nIndexLo[i] + HALF_UNIT_COUNT]
< dInput[nIndexLo[j] + HALF_UNIT_COUNT]){
                nAux = nIndexLo[i];
                nIndexLo[i] = nIndexLo[j];
                nIndexLo[j] = nAux;
            }
        }

        // Output the vector
        nOutput[i] = nIndexHi[i];
        nOutput[i + HALF_UNIT_COUNT] = nIndexLo[i];
    }

    nOutput[HALF_UNIT_COUNT       -        1]        =
nIndexHi[HALF_UNIT_COUNT - 1];
    nOutput[UNIT_COUNT         -         1]          =
nIndexLo[HALF_UNIT_COUNT - 1];
}



static void VoltageBallance(){
    const real_T   * UnitVoltagesU  = (const real_T*)
ssGetInputPortSignal(baseStruct,
PORTIN_UNIT_VOLTAGES_U);
    const real_T   * UnitVoltagesV  = (const real_T*)
ssGetInputPortSignal(baseStruct,
PORTIN_UNIT_VOLTAGES_V);
    const real_T   * UnitVoltagesW  = (const real_T*)
ssGetInputPortSignal(baseStruct,
PORTIN_UNIT_VOLTAGES_W);
```

```
    int i;

    //  Sort  the  voltages  for  each  phase  for  both
upper and lower sections
    BubleSort(UnitVoltagesU, & nSortedU(0));
    BubleSort(UnitVoltagesV, & nSortedV(0));
    BubleSort(UnitVoltagesW, & nSortedW(0));

    //  Set  the  main  states  for  upper  and  lower
sections for each phase
    for (i = 0; i < HALF_UNIT_COUNT; i++){

        // Do the upper section
        dUnitStateU(nSortedU(i)) = (i < dPhaseLevelU)
? 1.0 : -1.0;
        dUnitStateV(nSortedV(i)) = (i < dPhaseLevelV)
? 1.0 : -1.0;
        dUnitStateW(nSortedW(i)) = (i < dPhaseLevelW)
? 1.0 : -1.0;


        // Do the lower section
        dUnitStateU(HALF_UNIT_COUNT                  +
nSortedU(HALF_UNIT_COUNT    +    i))    =    (i    <
(HALF_UNIT_COUNT - dPhaseLevelU)) ? 1.0 : -1.0;
        dUnitStateV(HALF_UNIT_COUNT                  +
nSortedV(HALF_UNIT_COUNT    +    i))    =    (i    <
(HALF_UNIT_COUNT - dPhaseLevelV)) ? 1.0 : -1.0;
        dUnitStateW(HALF_UNIT_COUNT                  +
nSortedW(HALF_UNIT_COUNT    +    i))    =    (i    <
(HALF_UNIT_COUNT - dPhaseLevelW)) ? 1.0 : -1.0;
    }
}


static void HandleCurrentSign(){
    const real_T   * PhaseCurrent   = (const real_T*)
ssGetInputPortSignal(baseStruct, PORTIN_CURRENTS);
    const real_T   * PhaseVoltages  = (const real_T*)
ssGetInputPortSignal(baseStruct, PORTIN_VOLTAGES);

    // get the current sign. If not set (value 0) set
the sign to pozitive
    if (PhaseCurrent[0] > 0.5) dCurSenseU = 1.0;
    if (PhaseCurrent[0] < -0.5) dCurSenseU = -1.0;
    if (dCurSenseU == 0) dCurSenseU = 1.0;

    if (PhaseCurrent[1] > 0.5) dCurSenseV = 1.0;
    if (PhaseCurrent[1] < -0.5) dCurSenseV = -1.0;
    if (dCurSenseV == 0) dCurSenseV = 1.0;

    if (PhaseCurrent[2] > 0.5) dCurSenseW = 1.0;
    if (PhaseCurrent[2] < -0.5) dCurSenseW = -1.0;
    if (dCurSenseW == 0) dCurSenseW = 1.0;


 //   nQuadrantU = (PhaseVoltages[0] < 0) ? 2 : 1;
//    nQuadrantV = (PhaseVoltages[1] < 0) ? 2 : 1;
//    nQuadrantW = (PhaseVoltages[2] < 0) ? 2 : 1;
    nQuadrantU     =     (dCurSenseU     <     0)     ?
((PhaseVoltages[0] > 0) ? 4 : 3) : ((PhaseVoltages[0]
> 0) ? 1 : 2);
    nQuadrantV     =     (dCurSenseV     <     0)     ?
((PhaseVoltages[1] > 0) ? 4 : 3) : ((PhaseVoltages[1]
> 0) ? 1 : 2);
    nQuadrantW     =     (dCurSenseW     <     0)     ?
((PhaseVoltages[2] > 0) ? 4 : 3) : ((PhaseVoltages[2]
> 0) ? 1 : 2);
}

static void HandleIGBTStates(){
    int i;
    double dNewStateU0, dNewStateU1;
    double dNewStateV0, dNewStateV1;
    double dNewStateW0, dNewStateW1;
    double dCurrentState, b;

    for (i = 0; i < UNIT_COUNT; i++){
        // Set the states according with the current
sign

        dCurrentState = (dUnitStateU(i) > 0) ? 1.0 :
-1.0;
        dNewStateU0  =  (dCurSenseU  >  0)  ?  1.0  :
dCurrentState;
        dNewStateU1   =   (dCurSenseU   >   0)   ?
dCurrentState : 1.0;

        dCurrentState = (dUnitStateV(i) > 0) ? 1.0 :
-1.0;
        dNewStateV0  =  (dCurSenseV  >  0)  ?  1.0  :
dCurrentState;
        dNewStateV1   =   (dCurSenseV   >   0)   ?
dCurrentState : 1.0;

        dCurrentState = (dUnitStateW(i) > 0) ? 1.0 :
-1.0;
        dNewStateW0  =  (dCurSenseW  >  0)  ?  1.0  :
dCurrentState;
        dNewStateW1   =   (dCurSenseW   >   0)   ?
dCurrentState : 1.0;


        if  (nQuadrantU  %  2  ==  0)  {  dNewStateU0  =
dNewStateU1; dNewStateU1 = 1.0; }


        b = dNewStateU1 = (dUnitStateU(i) > 0) ? 1.0
: -1.0;
        switch (nQuadrantU){
        case 1:
            dNewStateU0 = 1.0;
            dNewStateU1 = b;
            break;
        case 2:
            dNewStateU0 = 1.0;
            dNewStateU1 = b;
            break;
        case 3:
            dNewStateU0 = 1.0;
            dNewStateU1 = b;
            break;
        case 4:
            dNewStateU0 = 1.0;
            dNewStateU1 = b;
            break;
        }

        dNewStateV0 = 1.0;
        dNewStateV1 = (dUnitStateV(i) > 0) ? 1.0 : -
1.0;

        dNewStateW0 = 1.0;
        dNewStateW1 = (dUnitStateW(i) > 0) ? 1.0 : -
1.0;


        // Set state for leg 0 (towards positive)
        // - If idle (old state = 0) then take the
new state
        // - If different than new state than take 0
        // - Else 0
        if       (dOldLegStateU0(i)       ==       0){
dOldLegStateU0(i) = dNewStateU0; }
        else  if  (dOldLegStateU0(i)  !=  dNewStateU0)
dOldLegStateU0(i) = 0;

        if       (dOldLegStateV0(i)       ==       0){
dOldLegStateV0(i) = dNewStateV0; }
        else  if  (dOldLegStateV0(i)  !=  dNewStateV0)
dOldLegStateV0(i) = 0;

        if       (dOldLegStateW0(i)       ==       0){
dOldLegStateW0(i) = dNewStateW0; }
        else  if  (dOldLegStateW0(i)  !=  dNewStateW0)
dOldLegStateW0(i) = 0;

        // Set state for leg 0 (towards negative)
        if       (dOldLegStateU1(i)       ==       0){
dOldLegStateU1(i) = dNewStateU1; }
        else  if  (dOldLegStateU1(i)  !=  dNewStateU1)
dOldLegStateU1(i) = 0;

        if       (dOldLegStateV1(i)       ==       0){
dOldLegStateV1(i) = dNewStateV1; }
        else  if  (dOldLegStateV1(i)  !=  dNewStateV1)
dOldLegStateV1(i) = 0;

        if       (dOldLegStateW1(i)       ==       0){
dOldLegStateW1(i) = dNewStateW1; }
        else  if  (dOldLegStateW1(i)  !=  dNewStateW1)
dOldLegStateW1(i) = 0;
    }
}




/*                 Function:                mdlOutputs
=====================================================
==
 * Abstract:
 *     In this function, you compute the outputs of
your S-function
 *      block. Generally outputs are placed in the
output vector, ssGetY(S).
 */
static void mdlOutputs(SimStruct *S, int_T tid){
    const  real_T     *  Enabled    =  (const  real_T*)
ssGetInputPortSignal(S, PORTIN_ENABLE);

    real_T                    *Levels     =   (real_T
*)ssGetOutputPortRealSignal(S, PORTOUT_LEVELS);
    real_T                   *SortedU     =   (real_T
*)ssGetOutputPortRealSignal(S, PORTOUT_SORT_U);
    real_T                     *UnitU     =   (real_T
*)ssGetOutputPortRealSignal(S, PORTOUT_UNITSTATE_U);
    real_T                     *IgbtU     =   (real_T
*)ssGetOutputPortRealSignal(S, PORTOUT_IGBTSTATE_U);
    real_T                   *SortedV     =   (real_T
*)ssGetOutputPortRealSignal(S, PORTOUT_SORT_V);
    real_T                     *UnitV     =   (real_T
*)ssGetOutputPortRealSignal(S, PORTOUT_UNITSTATE_V);
    real_T                     *IgbtV     =   (real_T
*)ssGetOutputPortRealSignal(S, PORTOUT_IGBTSTATE_V);
    real_T                   *SortedW     =   (real_T
*)ssGetOutputPortRealSignal(S, PORTOUT_SORT_W);
    real_T                     *UnitW     =   (real_T
*)ssGetOutputPortRealSignal(S, PORTOUT_UNITSTATE_W);
    real_T                     *IgbtW     =   (real_T
*)ssGetOutputPortRealSignal(S, PORTOUT_IGBTSTATE_W);
    real_T                 *Quadrants     =   (real_T
*)ssGetOutputPortRealSignal(S, PORTOUT_QUADRANTS);

    int i;          // Index counter

    // Retrieve global variables
    dValues = ssGetRWork(S);
    nValues = ssGetIWork(S);
    baseStruct = S;


    // If the sample time hit us ...
    if (ssIsSampleHit(S, 0, 0)){

        // handle the main pulses
        StaircasePulses();

        // Do the voltage balancing
        VoltageBallance();

        // handle the current sign
        HandleCurrentSign();

        // Do the IGBT states
        HandleIGBTStates();
    }
    // Output the data
    Levels[0] = dPhaseLevelU;
    Levels[1] = dPhaseLevelV;
    Levels[2] = dPhaseLevelW;

    for (i = 0; i < UNIT_COUNT; i++){
        SortedU[i] = nSortedU(i);
        SortedV[i] = nSortedV(i);
        SortedW[i] = nSortedW(i);

        if (Enabled[0] != 0){
            // output the unit states
            UnitU[i] = dUnitStateU(i);
            UnitV[i] = dUnitStateV(i);
            UnitW[i] = dUnitStateW(i);

            // Output the igbt states
            IgbtU[i * 2 + 0] = dOldLegStateU0(i);
            IgbtU[i * 2 + 1] = dOldLegStateU1(i);

            IgbtV[i * 2 + 0] = dOldLegStateV0(i);
            IgbtV[i * 2 + 1] = dOldLegStateV1(i);

            IgbtW[i * 2 + 0] = dOldLegStateW0(i);
            IgbtW[i * 2 + 1] = dOldLegStateW1(i);

        } else {
            UnitU[i] = 0;
            UnitV[i] = 0;
            UnitW[i] = 0;

            IgbtU[i * 2 + 0] = 0;
            IgbtU[i * 2 + 1] = 0;
            IgbtV[i * 2 + 0] = 0;
            IgbtV[i * 2 + 1] = 0;
            IgbtW[i * 2 + 0] = 0;
            IgbtW[i * 2 + 1] = 0;
        }
    }
    Quadrants[0] = nQuadrantU;
    Quadrants[1] = nQuadrantV;
    Quadrants[2] = nQuadrantW;

}



#undef  MDL_UPDATE   /*  Change  to  #undef  to  remove
function */
#if defined(MDL_UPDATE)
  /*                 Function:                  mdlUpdate
=====================================================
=
   * Abstract:
   *     This function is called once for every major
integration time step.
   *      Discrete states are typically updated here,
but this function is useful
   *      for performing any tasks that should only
take place once per
   *     integration step.
   */
  static void mdlUpdate(SimStruct *S, int_T tid){
  }
#endif /* MDL_UPDATE */



#undef MDL_DERIVATIVES  /* Change to #undef to remove
function */
#if defined(MDL_DERIVATIVES)
  /*              Function:              mdlDerivatives
=================================================
   * Abstract:
   *     In this function, you compute the S-function
block's derivatives.
   *     The derivatives are placed in the derivative
vector, ssGetdX(S).
   */
  static void mdlDerivatives(SimStruct *S){
  }
#endif /* MDL_DERIVATIVES */



/*                Function:                mdlTerminate
=====================================================
 * Abstract:
 *       In  this  function,  you  should  perform  any
actions that are necessary
 *       at  the  termination  of  a  simulation.   For
example, if memory was
 *      allocated  in  mdlStart,  this  is  the  place  to
free it.
 */
static void mdlTerminate(SimStruct *S){
}


#ifdef   MATLAB_MEX_FILE      /*  Is  this  file  being
compiled as a MEX-file? */
#include  "simulink.c"         /*  MEX-file  interface
mechanism */
#else
#include  "cg_sfun.h"            /*  Code  generation
registration function */
#endif
```

E.4 POWER SUPPLY PROTECTIONS

```
//
// Protection for over/voltages, over-currents
//
// Copyright:
//    Sandu Cristian - 2008
//    sanducristian@gmail.com
//       Code  created  for  the  Project  of  the  10th
semester at
//    Aalborg University
//




#define S_FUNCTION_NAME  Protection
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#ifdef MATLAB_MEX_FILE
#include <math.h>
#endif


#define                                      INV_SQRT_3
0.57735026918962576450914878050196
#define  SATURATE(val,  min,  max)      {  (val)  =  ((val)
<  (min)  ?  (min)  :  ((val)  >  (max)  ?  (max)  :  (val))); }
#define  MIN(a,  b,  c)                    (((a)  <  (b)  ?
((a)  <  (c)  ?  (a)  :  (c))  :  ((b)  <  (c)  ?  (b)  :  (c))))
#define  MAX(a,  b,  c)                    (((a)  >  (b)  ?
((a)  >  (c)  ?  (a)  :  (c))  :  ((b)  >  (c)  ?  (b)  :  (c))))
#define SATURATION_MIN                (0.02)
#define SATURATION_MAX                (0.98)


#define LIMIT_DC_VOLTAGE              (700)
#define LIMIT_DC_VOLTAGE_LOW          (650)
#define MAIN_DC_VOLTAGE_OFFSET        (20)
#define MAIN_DC_CHARGE_DIFF           (50)
#define MAX_VOLTAGE                   (2800)
#define MAX_OUTCURRENT                (30)
#define MAX_CURRENT                   (100)

#define POS_CHOPPER                   (0)
#define POS_FAULT                     (1)
#define POS_CONTACTOR                 (2)
#define POS_CHARGE                    (3)
#define POS_ENABLE                    (4)
#define POS_DCVOLTAGEMAX              (5)
#define POS_DCVOLTAGEMAX_DET          (6)
#define POS_DCVOLTAGEMIN              (7)
#define POS_DCVOLTAGEMIN_DET          (8)
#define POS_DCVOLTAGE_DET             (9)

/*=====================*
 * S-function methods *
 *====================*/

/*           Function:            mdlInitializeSizes
===============================================
 * Abstract:
 *     The  sizes  information  is  used  by  Simulink  to
determine the S-function
 *      block's  characteristics  (number  of  inputs,
outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S){
    ssSetNumSFcnParams(S,  0);   /* Number  of  expected
parameters */
    if           (ssGetNumSFcnParams(S)            !=
ssGetSFcnParamsCount(S)) {
        /* Return  if  number  of  expected  !=  number  of
actual parameters */
        return;
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);
    
    if (!ssSetNumInputPorts(S, 7)) return;
    /*Input Port 0 */
    ssSetInputPortWidth(S,  0, 1); /* Reset */
    ssSetInputPortDataType(S, 0, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  0, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortRequiredContiguous(S,     0,     1);
/*direct input signal access*/

    /*Input Port 0 */
    ssSetInputPortWidth(S,  1, 3); /* Vout */
    ssSetInputPortDataType(S, 1, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  1, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S, 1, 1);
    ssSetInputPortRequiredContiguous(S,     1,     1);
/*direct input signal access*/

    /*Input Port 0 */
    ssSetInputPortWidth(S,  2, 3); /* Iout */
    ssSetInputPortDataType(S, 2, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  2, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S, 2, 1);
    ssSetInputPortRequiredContiguous(S,     2,     1);
/*direct input signal access*/

    /*Input Port 0 */
    ssSetInputPortWidth(S,  3, 1); /* VDC */
    ssSetInputPortDataType(S, 3, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  3, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S, 3, 1);
    ssSetInputPortRequiredContiguous(S,     3,     1);
/*direct input signal access*/

    /*Input Port 0 */
    ssSetInputPortWidth(S,  4, 1); /* Vrect */
    ssSetInputPortDataType(S, 4, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  4, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S, 4, 1);
    ssSetInputPortRequiredContiguous(S,     4,     1);
/*direct input signal access*/

    /*Input Port 0 */
    ssSetInputPortWidth(S,  5, 1); /* IDC */
    ssSetInputPortDataType(S, 5, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  5, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S, 5, 1);
    ssSetInputPortRequiredContiguous(S,     5,     1);
/*direct input signal access*/


    /*Input Port 0 */
    ssSetInputPortWidth(S,  6, 1); /* Run */
    ssSetInputPortDataType(S, 6, SS_DOUBLE);
    ssSetInputPortComplexSignal(S,  6, COMPLEX_NO);
    ssSetInputPortDirectFeedThrough(S, 6, 1);
    ssSetInputPortRequiredContiguous(S,     6,     1);
/*direct input signal access*/


    /*
     * Set direct feedthrough flag (1=yes, 0=no).
     * A port has direct feedthrough if the input is
used in either
     *  the   mdlOutputs   or   mdlGetTimeOfNextVarHit
functions.
     *                                             See
matlabroot/simulink/src/sfuntmpl_directfeed.txt.
     */
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S, 6)) return;
    /* Output Port 0 */
    ssSetOutputPortWidth(S, 0, 1);   /* Fault */
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 0, COMPLEX_NO);

    /* Output Port 0 */
    ssSetOutputPortWidth(S, 1, 1);   /* Chopper */
    ssSetOutputPortDataType(S, 1, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 1, COMPLEX_NO);

    /* Output Port 0 */
    ssSetOutputPortWidth(S,  2,  1);        /*  Main
contactor */
    ssSetOutputPortDataType(S, 2, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 2, COMPLEX_NO);

    /* Output Port 0 */
    ssSetOutputPortWidth(S,  3,  1);     /*  Capacitor
Charge */
    ssSetOutputPortDataType(S, 3, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 3, COMPLEX_NO);

    /* Output Port 0 */
    ssSetOutputPortWidth(S,  4,  1);      /*  Control
Enable */
    ssSetOutputPortDataType(S, 4, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 4, COMPLEX_NO);

    /* Output Port 0 */
    ssSetOutputPortWidth(S, 5, 10);   /* Debug */
    ssSetOutputPortDataType(S, 5, SS_DOUBLE);
    ssSetOutputPortComplexSignal(S, 5, COMPLEX_NO);


    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 10);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    ssSetOptions(S, 0);
}



/*         Function:         mdlInitializeSampleTimes
=========================================
 * Abstract:
 *      This function is used to specify the sample
time(s) for your
 *     S-function. You must register the same number
of sample times as
 *    specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct *S){
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S,                             0,
FIXED_IN_MINOR_STEP_OFFSET); // 0.0);
}



#define  MDL_INITIALIZE_CONDITIONS     /*  Change  to
#undef to remove function */
#if defined(MDL_INITIALIZE_CONDITIONS)
  static void mdlInitializeConditions(SimStruct *S){
  }
#endif /* MDL_INITIALIZE_CONDITIONS */



#define MDL_START   /* Change to #undef to remove
function */
#if defined(MDL_START)
  /*                 Function:                mdlStart
=====================================================
==
   * Abstract:
   *      This function is called once at start of
model execution. If you
   *     have states that should be initialized once,
this is the place
   *    to do it.
   */
  static void mdlStart(SimStruct *S) {
  }
#endif /*  MDL_START */
```

```
#define MDL_SET_DEFAULT_PORT_DATA_TYPES
static void mdlSetDefaultPortDataTypes(SimStruct *S){
  ssSetInputPortDataType(S, 0, SS_DOUBLE);
  ssSetOutputPortDataType(S, 0, SS_DOUBLE);
}


/*                Function:                mdlOutputs
=====================================================
==
 * Abstract:
 *      In this function, you compute the outputs of
your S-function
 *      block. Generally outputs are placed in the
output vector, ssGetY(S).
 */
static void mdlOutputs(SimStruct *S, int_T tid){
    const   real_T       *Reset     =   (const   real_T*)
ssGetInputPortSignal(S,0);
    const   real_T        *Vout     =   (const   real_T*)
ssGetInputPortSignal(S,1);
    const   real_T        *Iout     =   (const   real_T*)
ssGetInputPortSignal(S,2);
    const   real_T        *VDC      =   (const   real_T*)
ssGetInputPortSignal(S,3);
    const   real_T       *Vrect     =   (const   real_T*)
ssGetInputPortSignal(S,4);
    const   real_T        *IDC      =   (const   real_T*)
ssGetInputPortSignal(S,5);
    const   real_T        *Run      =   (const   real_T*)
ssGetInputPortSignal(S,6);
    real_T                    *Fault      =   (real_T
*)ssGetOutputPortRealSignal(S,0);
    real_T                  *Chopper      =   (real_T
*)ssGetOutputPortRealSignal(S,1);
    real_T              *MainContactor    =   (real_T
*)ssGetOutputPortRealSignal(S,2);
    real_T                 *CapCharge     =   (real_T
*)ssGetOutputPortRealSignal(S,3);
    real_T                   *Enable      =   (real_T
*)ssGetOutputPortRealSignal(S,4);
    real_T                    *Debug      =   (real_T
*)ssGetOutputPortRealSignal(S,5);

    int i;
    int bFault = 0;
    int bMajorFault = 0;
    int nDCVoltageMax = 0;
    int nDCVoltageMin = 0;
    int nDCDeterminedMax = 0;
    int nDCDeterminedMin = 0;
    int nDCDetermined = 0;

    // Retrieve the previous states
    Chopper[0] = ssGetIWorkValue(S, POS_CHOPPER);
    Fault[0] = ssGetIWorkValue(S, POS_FAULT);
    MainContactor[0]        =       ssGetIWorkValue(S,
POS_CONTACTOR);
    CapCharge[0] = ssGetIWorkValue(S, POS_CHARGE);
    Enable[0] = ssGetIWorkValue(S, POS_ENABLE);
    nDCVoltageMax         =         ssGetIWorkValue(S,
POS_DCVOLTAGEMAX);
    nDCDeterminedMax       =        ssGetIWorkValue(S,
POS_DCVOLTAGEMAX_DET);
    nDCVoltageMin         =         ssGetIWorkValue(S,
POS_DCVOLTAGEMIN);
    nDCDeterminedMin       =        ssGetIWorkValue(S,
POS_DCVOLTAGEMIN_DET);
    nDCDetermined         =         ssGetIWorkValue(S,
POS_DCVOLTAGE_DET);


    // If run is set then enable the main contactor
and set the enable flag
    //  the  check  for  charging  is  done  later  by
checking the charging state
    if (Run[0] != 0.0){
        MainContactor[0] = 1.0;
        Enable[0] = 1.0;
    }
    // Check for over DC Voltage in order to connect
the contactor
    if (Chopper[0]){
        if (VDC[0] < LIMIT_DC_VOLTAGE_LOW){
            Chopper[0] = 0;
        }
    } else {
        if (VDC[0] > LIMIT_DC_VOLTAGE){
            Chopper[0] = 1;
            if (VDC[0] > MAX_VOLTAGE){
                // Triger fault if the VDC get over
the limited voltage
                bMajorFault = 1;
            }
        } else {
            Chopper[0] = 0;
        }
    }


    // Check for charging
    if (CapCharge[0] == 0.0){
        if (nDCDeterminedMax == 0){
            if (Vrect[0] > nDCVoltageMax){
                nDCVoltageMax = Vrect[0];
                nDCVoltageMin = nDCVoltageMax;
            }
            if    (nDCVoltageMax    -    Vrect[0]    >=
MAIN_DC_VOLTAGE_OFFSET){
                nDCDeterminedMax = 1;
                nDCDeterminedMin = 0;
                nDCVoltageMin = nDCVoltageMax;
            }
        } else {
            if (nDCVoltageMin > Vrect[0]){
                nDCVoltageMin = Vrect[0];
            }

            if (nDCVoltageMin < Vrect[0] - 10){
                nDCDeterminedMin = 1;
                nDCDeterminedMax = 0;
            }
        }

        // No charging was made so check the voltages
        if    ((nDCVoltageMin    <=    VDC[0])    &&
nDCDeterminedMin){
            if (nDCDetermined > 5){
                CapCharge[0] = 1;
            } else {
                nDCDetermined++;
            }
        }
    }

    if (nDCDetermined < 10){
        // Don't run just yet
        Enable[0] = 0;

        if     (nDCVoltageMax     -     VDC[0]     <
MAIN_DC_VOLTAGE_OFFSET){
            nDCDetermined++;
        }
    }


    // Check for output overcurrent
    //if (Iout[0] > MAX_OUTCURRENT) bFault = 1;
    //if (Iout[1] > MAX_OUTCURRENT) bFault = 2;
    //if (Iout[2] > MAX_OUTCURRENT) bFault = 3;

    // if (IDC[0] > MAX_CURRENT) bFault = 4;


    // Check for major fault
    if (bMajorFault) {
        bFault = 10;
        MainContactor[0] = 0;
    }


    // Check for reset signal
    if (Reset[0] != 0){
        if (bFault == 0){
            Fault[0] = 0;
        }
    }

    // Check for fault
    if (bFault != 0) Fault[0] = bFault;

    // Check the enable flag for fault, charge, etc
    if (Fault[0]) Enable[0] = 0;
    if (CapCharge[0] == 0.0) Enable[0] = 0;


    // Save the values
    ssSetIWorkValue(S, POS_CHOPPER, Chopper[0]);
    ssSetIWorkValue(S, POS_FAULT, Fault[0]);
    ssSetIWorkValue(S,                 POS_CONTACTOR,
MainContactor[0]);
    ssSetIWorkValue(S, POS_CHARGE, CapCharge[0]);
    ssSetIWorkValue(S, POS_ENABLE, Enable[0]);
    ssSetIWorkValue(S,              POS_DCVOLTAGEMAX,
nDCVoltageMax);
    ssSetIWorkValue(S,          POS_DCVOLTAGEMAX_DET,
nDCDeterminedMax);
    ssSetIWorkValue(S,              POS_DCVOLTAGEMIN,
nDCVoltageMin);
    ssSetIWorkValue(S,          POS_DCVOLTAGEMIN_DET,
nDCDeterminedMin);
    ssSetIWorkValue(S,             POS_DCVOLTAGE_DET,
nDCDetermined);

    Debug[0] = nDCVoltageMax;
    Debug[1] = nDCDeterminedMax;
    Debug[2] = nDCVoltageMin;
    Debug[3] = nDCDeterminedMin;
    Debug[4] = nDCDetermined;


    //
    // END
    //
}



#undef  MDL_UPDATE    /*  Change  to  #undef  to  remove
function */
#if defined(MDL_UPDATE)
  /*              Function:              mdlUpdate
=====================================================
=
   * Abstract:
   *     This function is called once for every major
integration time step.
   *     Discrete states are typically updated here,
but this function is useful
   *      for performing any tasks that should only
take place once per
   *     integration step.
   */
  static void mdlUpdate(SimStruct *S, int_T tid){
  }
#endif /* MDL_UPDATE */



#undef MDL_DERIVATIVES  /* Change to #undef to remove
function */
#if defined(MDL_DERIVATIVES)
  /*             Function:            mdlDerivatives
=================================================
   * Abstract:
   *     In this function, you compute the S-function
block's derivatives.
   *     The derivatives are placed in the derivative
vector, ssGetdX(S).
```

```
   */
  static void mdlDerivatives(SimStruct *S){
  }
#endif /* MDL_DERIVATIVES */



/*                 Function:                 mdlTerminate
=====================================================
 * Abstract:
 *     In this function, you should perform any actions that
are necessary
 *     at the termination of a simulation.  For example, if
memory was
 *    allocated in mdlStart, this is the place to free it.
 */
static void mdlTerminate(SimStruct *S){
}


#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a
MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"          /* Code generation registration
function */
#endif
```

APPENDIX F. SOFTWARE

F.1 SORTING ALGORITHM

The sorting is based on equal and greater comparison while the lowest comparison done by AND the negated values of the other 2 comparisons. The result is summed together as explained in the following paragraphs.

The equality as greater comparison is done by comparing each value with all the ones after it. Therefore for 4 values a vector of 6 elements is used. The equality will be named E[x] with X being in the range of [0, 5]. The greater comparison will be named G[x] while the lowest comparison will be named L[x]. The signification of how the values are calculated is presented in Table F-1.

ndex	E[index]	G[index]
	$Input[0] == Input[1]$	$Input[0] > Input[1]$
	$Input[0] == Input[2]$	$Input[0] > Input[2]$
	$Input[0] == Input[3]$	$Input[0] > Input[3]$
	$Input[1] == Input[2]$	$Input[1] > Input[2]$
	$Input[1] == Input[3]$	$Input[1] > Input[3]$
	$Input[2] == Input[3]$	$Input[2] > Input[3]$

Table F-1: Sorting comparisons index

The Less comparison is calculated by the following equation:

$$L[x] = (not[E[x]])\ AND\ (not[G[x]]); with\ x \in \{0, 1, 2, 3, 4, 5\}$$

Equation F-1: Sorting less comparison

The offset used by the sorting is only used for duplicate values, for each duplicate the first value will be incremented with an initial value of 0. The offset is calculated as follows:

$$O[0] = E[0] + E[1] + E[2]$$

$$O[1] = E[3] + E[4]$$

$$O[2] = E[5]$$

$$O[3] = 0$$

Equation F-2: Sorting offset calculation

Several examples of how sorting works are given in the appendix following subchapters.

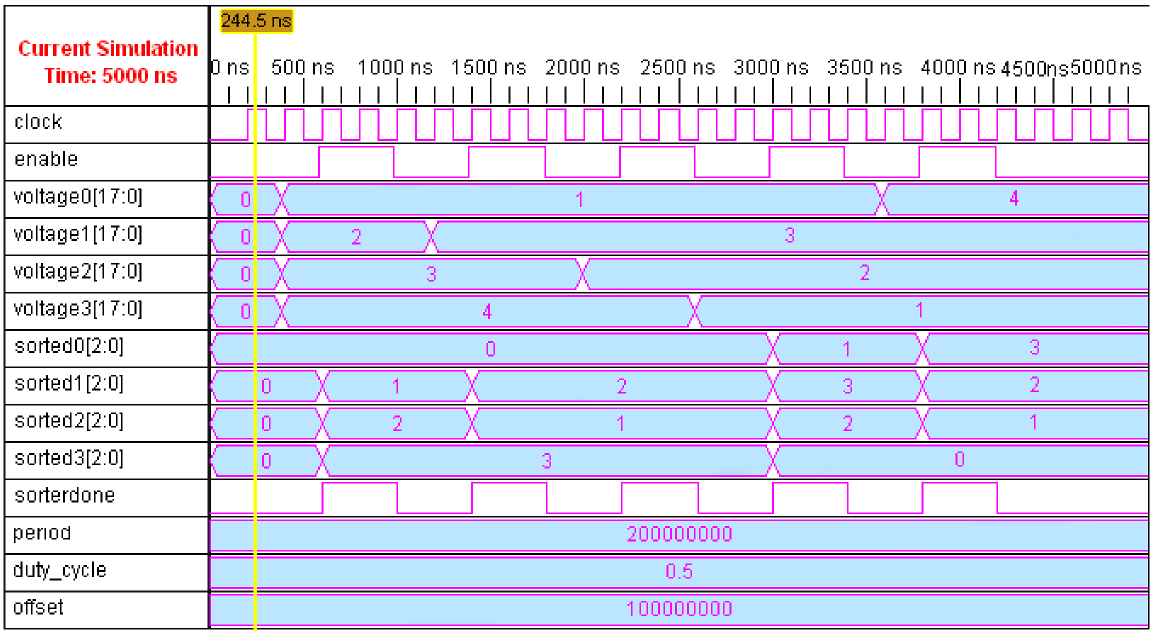


Figure F-1: FPGA Control stages

F.1.I EXAMPLE 1

As an example the vector [0, 1, 2, 3] will be sorted.

Input values	Equality E[x]	Grater G[x]	Less L[x]	Offset O[x]	Result
0	([0] = [1]) -> 0	([0] >[1]) -> 0	(! E) & (! G) -> 1	0	O[0] + G[0] + G[1] + G[2] = 0
1	([0] = [2]) -> 0	([0] > [2]) -> 0	(! E) & (! G) -> 1	0	O[1] + G[3] + G[4] + L[0] = 1
2	([0] = [3]) -> 0	([0] > [3]) -> 0	(! E) & (! G) -> 1	0	O[2] + G[5] + L[1] + L[3] = 2
3	([1] = [2]) -> 0	([1] > [2]) -> 0	(! E) & (! G) -> 1	0	O[3] + L[2] + L[4] + L[5] = 3
	([1] = [3]) -> 0	([1] > [3]) -> 0	(! E) & (! G) -> 1		
	([2] = [3]) -> 0	([2] > [3]) -> 0	(! E) & (! G) -> 1		

F.1.II EXAMPLE 2

The secondary example is with the values of [0, 2, 1, 3]:

Input values	Equality E[x]	Grater G[x]	Less L[x]	Offset O[x]	Result
0	([0] = [1]) -> 0	([0] >[1]) -> 0	(! E) & (! G) -> 1	0	O[0] + G[0] + G[1] + G[2] = 0
2	([0] = [2]) -> 0	([0] > [2]) -> 0	(! E) & (! G) -> 1	0	O[1] + G[3] + G[4] + L[0] = 2
1	([0] = [3]) -> 0	([0] > [3]) -> 0	(! E) & (! G) -> 1	0	O[2] + G[5] + L[1] + L[3] = 1
3	([1] = [2]) -> 0	([1] > [2]) -> 1	(! E) & (! G) -> 0	0	O[3] + L[2] + L[4] + L[5] = 3
	([1] = [3]) -> 0	([1] > [3]) -> 0	(! E) & (! G) -> 1		
	([2] = [3]) -> 0	([2] > [3]) -> 0	(! E) & (! G) -> 1		

F.1.III EXAMPLE 3

The third example is with the values of [0, 2, 2, 3]:

Input values	**Equality E[x]**	**Grater G[x]**	**Less L[x]**	**Offset O[x]**	**Result**
0	([0] = [1]) -> 0	([0] >[1]) -> 0	(! E) & (! G) -> 1	0	O[0] + G[0] + G[1] + G[2] = 0
2	([0] = [2]) -> 0	([0] > [2]) -> 0	(! E) & (! G) -> 1	1	O[1] + G[3] + G[4] + L[0] = 2
2	([0] = [3]) -> 0	([0] > [3]) -> 0	(! E) & (! G) -> 1	0	O[2] + G[5] + L[1] + L[3] = 1
3	([1] = [2]) -> 1	([1] > [2]) -> 0	(! E) & (! G) -> 0	0	O[3] + L[2] + L[4] + L[5] = 3
	([1] = [3]) -> 0	([1] > [3]) -> 0	(! E) & (! G) -> 1		
	([2] = [3]) -> 0	([2] > [3]) -> 0	(! E) & (! G) -> 1		

F.2 INTER FPGA COMMUNICATION

F.2.I PROTOCOL

The protocol for the communication is simple as it uses data packets to send data. A data packet is a set of bits, each with its own role. A packet contains 64 bits. These 64 bits are grouped into 3 sections:

Command
Parameters
Number

The format of the data packet is shown in Table F-2.

Offset	**Size**	**Name**
0	8	Command
8	8	Parameter 0
16	8	Parameter 1
24	8	Parameter 2
32	32	Number (see number format)

Table F-2: Data packet

The command is an 8 bit value which represents the actual command that is to be executed. The commands vary from the open of a contactor to a user parameter pass between the two. The commands will be described further inside the software documentation present on the attached CD.

The command parameters as well as the attached number have various functions depending on the issued command. Each command has its own parameters and number. Of course for some commands, the parameter, the number or both can be empty. This is the case of shut-down or power-up command.

F.2.II DATA BUS

The data bus for the FPGA to FPGA communication is composed of 4 MISO and 4 MOSI lines. Because the data packet has a size of 64 bits the bits have been split up over the 4 data lines. For better understanding of the bus the data flow is represented in Figure F-4.

The MOSI lines are synchronized with the Chip Select signal while the MISO lines are shifted 180°. The shift is necessary in order to allow the slave to properly respond to the CS line trigger.

Even if the data lines are differential LVDS in the Figure F-4 only the main signal have been represented. By using the 4 data lines, the clock cycles used are reduced 4 times from 64 cycles to 16 cycles. This leads to faster communication as well as to higher bandwidth.

F.3 MAIN FPGA – STAIRCASE MODULATION

As an overall picture of how the staircase modulation is working inside the FPGA a print screen of the Xilinx ISE simulation over the staircase modulation of one phase is depicted by Figure F-2.

The voltage levels are the default limits and are presented also as independent variable in myTest1, myTest2, myTest3 and myTest4. The reference variable represents the actual reference that was “received” from the DSP. The unit states are clearly set after 5 cycles. The stages of the state machine also switch, the current state being defined by the myCurrentState variable. The sorted values can also be noticed in the lower part of the figure in the calculated order. Because this is a simple simulation in which the unit states were tested, the voltage level of the units have been set to consecutive values of 0, 1, 2 and 3.

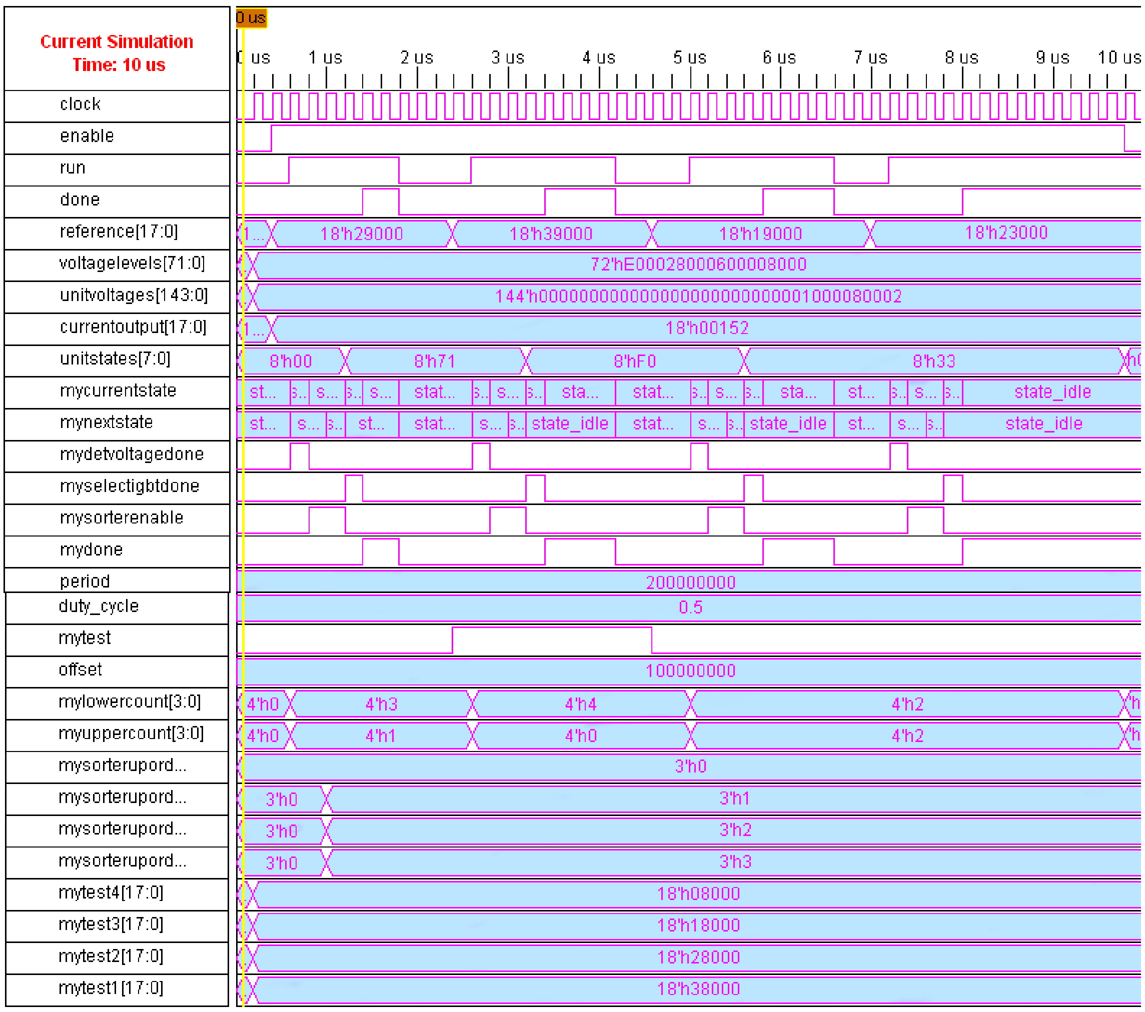


Figure F-3: Staircase simulation for FPGA

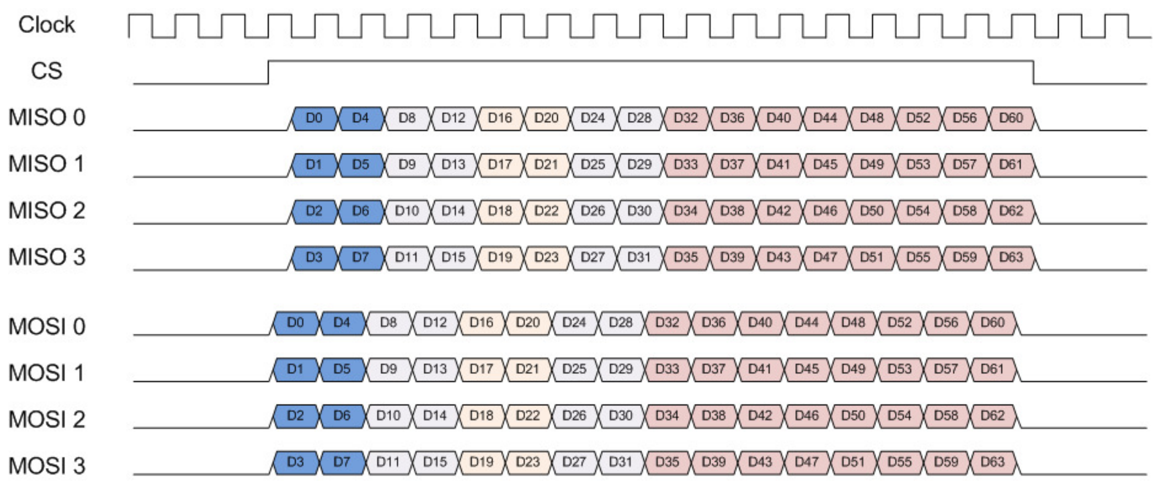

Clock
CS
MISO 0
MISO 1
MISO 2
MISO 3
MOSI 0
MOSI 1
MOSI 2
MOSI 3
D0 D4 D8 D12 D16 D20 D24 D28 D32 D36 D40 D44 D48 D52 D56 D60
D1 D5 D9 D13 D17 D21 D25 D29 D33 D37 D41 D45 D49 D53 D57 D61
D2 D6 D10 D14 D18 D22 D26 D30 D34 D38 D42 D46 D50 D54 D58 D62
D3 D7 D11 D15 D19 D23 D27 D31 D35 D39 D43 D47 D51 D55 D59 D63

Figure F-4: FPGA Communication bit order and position

In the Figure F-4 the blue represents the commands, with the lighter blue are the parameters while with light purple is the number.

F.4 FLASH MEMORY

The flash memory of the main FPGA is used to store values of various operations. These values were calculated offline, on a personal computer. The obtained values were transferred into the FPGA which in turned wrote them to the FLASH memory.

The mathematical functions for which the calculations were made are:

- F(x) = sin(x)
- F(x) = cos(x)
- F(x) = tan(x)
- F(x) = ctan(x)
- F(x) = asin(x)
- F(x) = acos(x)
- F(x) = atan(x)
- F(x) = actan(x)
- F(x) = ln(x)
- F(x) = log(x)
- F(x) = exp(x)
- F(x) = 10^X
- F(x) = 1 / x
- F(x) = $\sqrt{x}$
- F(x) = $\sqrt[3]{x}$

The “x” represents a number which format is described in F.4.i Number format.

The flash memory found on the main FPGA board only allows 8 bits of data to be transferred at one read/write operation. Therefore, in order to read the whole result, several steps must be followed during which 8 bits will be read at a time.

The flash memory is only used to store real numbers so the division by 0 will return 0. It is up to the main software to handle such exceptions not to the flash memory.

The function parameter "x" represents the actual address in the flash memory of the result. The full memory space of the flash memory is divided in order to allow access to results of all functions, all values and for all result bits. Therefore, the memory space is divided as presented in:

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Function				Number																		Section	
3	2	1	0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0

Figure F-5: Address format

The function parameter bits select the desired function. The definition of these bits is:

No.	Function Bits	Description
1	0000	F(x) = sin(x)
2	0001	F(x) = cos(x)
3	0010	F(x) = tan(x)
4	0011	F(x) = ctan(x)
5	0100	F(x) = asin(x)
6	0101	F(x) = acos(x)
7	0110	F(x) = atan(x)
8	0111	F(x) = actan(x)
9	1000	F(x) = $\ln(\lvert x \rvert)$
10	1001	F(x) = $\log(\lvert x \rvert)$
11	1010	F(x) = exp(x)
12	1011	F(x) = 10^x
13	1100	F(x) = 1 / x
14	1101	F(x) = $\sqrt{\lvert x \rvert}$
15	1110	F(x) = $\sqrt[3]{\lvert x \rvert}$
16	1111	Constants (PI, e, etc)

Table F-3: Memory function bits

The section bits describe the result value and may set the type of the input/output parameter of the function for the trigonometric function. The bit flags for the section bits are:

No.	Bit flags	Description
1	00	Result bits 7 – 0
2	01	Result bits 15 – 8
3	10	Result bits 23 – 16 (see F.5 Number format)
4	11	Result bits 31 – 24 (see F.5 Number format)

Table F-4: Section bit flags

The number section is on 18 bits and the representation is discussed in F.4.i Number format.

F.4.I NUMBER FORMAT

The function parameter is an integer value on 18 bits (with sign). The value represents a certain number in fixed point format. The number format had been established for the application as being 8 bits decimal, 9 bits integer plus one sign bit.

A) TRIGONOMETRIC FUNCTIONS

The sine, cosine, tangent and cotangent have an input range from $[-\pi, \pi\ ]$ therefore the input value will be limited to [-4, 4]. The number representation and limitation is presented in Table F-5. The output for the tangent and cotangent are normal number representation defined at application level.

Name	Domain of x				Domain of y			
	Sign	Bits of Integral part	Bits of Decimal part	Real Interval	Sign	Bits of Integral part	Bits of Decimal part	Real Interval
$y = \sin(x)$	17	16-15	14-0	[-π, π]	17	16	15-0	[-1, 1]
$y = \cos(x)$	17	16-15	14-0	[-π, π]	17	16	15-0	[-1, 1]
$y = \tan(x)$	17	16-15	14-0	[-π, π]	17	16-8	7-0	[-512, 511]
$y = \mathrm{ctan}(x)$	17	16-15	14-0	[-π, π]	17	16-8	7-0	[-512, 511]
$y = \arcsin(x)$	17	16	15-0	[-1, 1]	17	16-15	14-0	[-π, π]
$y = \arccos(x)$	17	16	15-0	[-1, 1]	17	16-15	14-0	[-π, π]
$y = \arctan(x)$	17	16-8	7-0	[-512, 511]	17	16-15	14-0	[-π, π]
$y = \mathrm{arctan}(x)$	17	16-8	7-0	[-512, 511]	17	16-15	14-0	[-π, π]

Table F-5: Numeric format for trigonometric functions

B) EXPONENTIAL FUNCTIONS

The exponential functions use e and the base of 10 for calculations. All the input values are in Q8 format. The Q8 format is defined at application level and will not be discussed here. The output is still a number in Q8 format.

For the logarithmic functions, the parameter number is assumed to be positive. The sign will be ignored and the result will be $f(x) = \exp(|x|)$.

Name	Domain of x				Domain of y			
	Sign	Bits of Integral part	Bits of Decimal part	Real Interval	Sign	Bits of Integral part	Bits of Decimal part	Real Interval
$y = \ln(\|x\|)$	17	16-8	7-0	[-512, 512]	17	variable	variable	n/a
$y = \log(\|x\|)$	17	16-8	7-0	[-512, 512]	17	variable	variable	n/a
$y = \exp(\mathrm{x})$	17	16-14	13-0	[-8, 8]	17	variable	variable	n/a
$y = 10^{\mathrm{X}}$	17	16-14	13-0	[-8, 8]	17	variable	variable	n/a

Table F-6: Exponential function input and output parameters interval

C) DIVISION

The input and output of the division is in Q8 number format. For the 0 value as input parameter the output value is 0. It is up to the main program to handle division by zero.

Name	Domain of x				Domain of y			
	Sign	Bits of Integral part	Bits of Decimal part	Real Interval	Sign	Bits of Integral part	Bits of Decimal part	Real Interval
$y = \frac{1}{x}$	17	variable	variable	n/a	17	variable	variable	n/a

Table F-7: Division input and output parameter range

The domain of X can be variable because the result is shifted depending on the Q value of X domain. For example the value 11 in Q5 is 352 while in Q6 is 704. The 1/x for the three values is 0,0909, 0,002841 and 0,00142 respectively. When shifted with the Q value, all three results will be 0,0909.

D) SQUARE ROOT

The square root is calculated without taking in consideration the sign. It is assumed that the sign is always positive. The number format of the input and output is Q8.

F.4.II EXCEPTIONS

The exceptions must be handled by the main software and represent faults in the mathematical processing. The faults that the application must solve are:

- Division by zero
- Square root of negative numbers
- Tangent parameter $\notin (89{,}88;\ -89{,}88)$
- Arcsine and arccosine parameter $\notin [-1; 1]$
- Exponential parameter $\notin [-5{,}54; 6{,}23]$

F.5 NUMBER FORMAT

For this application, a number format on 18 bits with 32 bits implementation has been used. The 32 bits implementation is required due to memory mappings on either 16 or 32 bits. 16 bit platform represents a step back for the FPGA capabilities as well as for the number resolution. In order to Increase the resolution and not to affect the data bus size of either the DSP or the RAM the 32 bit representation seemed to be the best option.

The number format had been chosen on 18 bits due to limitations by the FPGA multiplier and by flash memory available for extracting the desired result. The limit could have been set to 20 bits but there is no visible importance to it because the number range required for the application as well as resolution fits in the 18 bits number.

The application also requires the use of decimal values, values that are not integer. Because the FPGA does not natively support floating point, the support for integer multiplication is used in order to allow a fixed point representation. The fixed point representation used is based on the Q number format. The Q number format is implemented on 32 bits for this application as it will be described further.

F.5.I Q NUMBER FORMAT

The Q number format is a number format that allows flexible usage of bits for fixed point representation. The main number format used in this application is based on Q8, meaning that 8 bits are used for the decimal. Of course, other Q's are used all over the application depending on the role and purpose. Due to this, a list of signed number is shown in Table F-8. The table contains the Q, the corresponding number format as well as the range and resolution. The resolution represents the smallest undivided value that can be represented. All other values between the minimum and maximum are multiples of the resolution value. Values which are not an integer multiple of the resolution are represented to the closest integer multiple.

Q	Number																		Maximum	Minimum	Resolution
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0	S	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	131.071,00000000	-131072	1,0000000000
1	S	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D	65.535,50000000	-65536	0,5000000000
2	S	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	32.767,75000000	-32768	0,2500000000
3	S	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	D	16.383,87500000	-16384	0,1250000000
4	S	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	D	D	8.191,93750000	-8192	0,0625000000
5	S	N	N	N	N	N	N	N	N	N	N	N	N	D	D	D	D	D	4.095,96875000	-4096	0,0312500000
6	S	N	N	N	N	N	N	N	N	N	N	N	D	D	D	D	D	D	2.047,98437500	-2048	0,0156250000
7	S	N	N	N	N	N	N	N	N	N	N	D	D	D	D	D	D	D	1.023,99218750	-1024	0,0078125000
8	S	N	N	N	N	N	N	N	N	N	D	D	D	D	D	D	D	D	511,99609375	-512	0,0039062500
9	S	N	N	N	N	N	N	N	N	D	D	D	D	D	D	D	D	D	255,99804688	-256	0,0019531250
10	S	N	N	N	N	N	N	N	D	D	D	D	D	D	D	D	D	D	127,99902344	-128	0,0009765625
11	S	N	N	N	N	N	N	D	D	D	D	D	D	D	D	D	D	D	63,99951172	-64	0,0004882813
12	S	N	N	N	N	N	D	D	D	D	D	D	D	D	D	D	D	D	31,99975586	-32	0,0002441406
13	S	N	N	N	N	D	D	D	D	D	D	D	D	D	D	D	D	D	15,99987793	-16	0,0001220703
14	S	N	N	N	D	D	D	D	D	D	D	D	D	D	D	D	D	D	7,99993896	-8	0,0000610352
15	S	N	N	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	3,99996948	-4	0,0000305176
16	S	N	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	1,99998474	-2	0,0000152588
17	S	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	0,99999237	-1	0,0000076294

Table F-8: Q Number format values for signed 18 bit representation

The sign number format is not required in many cases. Therefore an unsigned number format is aloes required, for example if the value represents the DC voltage or other strictly positive values. For these cases the sign bit is omitted. The corresponding Q number values for the unsigned number are shown in Table F-9. In the table, only the maximum value is indicated, the minimum being 0. Identical with the case of signed Q number the actual number is an integer multiplier of the resolution. If a value is not a positive integer multiplier, the result will be rounded to the nearest positive integer multiplier of the resolution.

Q	Number																		Maximum	Resolution
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	262.143,00000000	1,0000000000
1	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D	131.071,50000000	0,5000000000
2	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	65.535,75000000	0,2500000000
3	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	D	32.767,87500000	0,1250000000
4	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	D	D	16.383,93750000	0,0625000000
5	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	D	D	D	8.191,96875000	0,0312500000
6	N	N	N	N	N	N	N	N	N	N	N	N	D	D	D	D	D	D	4.095,98437500	0,0156250000
7	N	N	N	N	N	N	N	N	N	N	N	D	D	D	D	D	D	D	2.047,99218750	0,0078125000
8	N	N	N	N	N	N	N	N	N	N	D	D	D	D	D	D	D	D	1.023,99609375	0,0039062500
9	N	N	N	N	N	N	N	N	N	D	D	D	D	D	D	D	D	D	511,99804688	0,0019531250
10	N	N	N	N	N	N	N	N	D	D	D	D	D	D	D	D	D	D	255,99902344	0,0009765625
11	N	N	N	N	N	N	N	D	D	D	D	D	D	D	D	D	D	D	127,99951172	0,0004882813
12	N	N	N	N	N	N	D	D	D	D	D	D	D	D	D	D	D	D	63,99975586	0,0002441406
13	N	N	N	N	N	D	D	D	D	D	D	D	D	D	D	D	D	D	31,99987793	0,0001220703
14	N	N	N	N	D	D	D	D	D	D	D	D	D	D	D	D	D	D	15,99993896	0,0000610352
15	N	N	N	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	7,99996948	0,0000305176
16	N	N	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	3,99998474	0,0000152588
17	N	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	1,99999237	0,0000076294
18	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	0,99999619	0,0000038147

Table F-9: Q Number format values for unsigned 18 bit representation

Both number representations (signed and unsigned) are used to represent operational result values. These values are encoded in a number format on 32 bits as described further.

F.5.II STORED NUMBER FORMAT

The 18 bit value implemented on a 32 bit platform implies that several bits will not be used, therefore, the remainder of the bits will be used for other purposes like the representation of the Q value (see Q number format) the flag that specified that the number is signed or unsigned as well as some other bits used for specific purposes. The number format used for RAM/ROM storage is represented in Table F-10. For DSP communication the bits D and E will be reserved as they are used only for memory storage for specific operations.

The total number of bits for a number is 18 from which the first one may represents the sign. In order to allow a flexible usage of the 18 bits and to know the Q value 6 bits have been used in order to store the Q value. 6 bits have been used instead of 5 in order to align the number to 24 bits. The remained 8 bits are used as flags for various purposes. The actual format is shown in Table F-10 and described in detail in the following paragraphs.

Offset 11								Offset 10								Offset 01								Offset 00							
Byte 3								Byte 2								Byte 1								Byte 0							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	R	R	R	D	E	S	Q value						Number																	

E	- Exception
S	- Signed
R	- Reserved
D	- Over domain (sine, cosine, etc)

Table F-10: Stored result number format

The number parameter is scaled according with the Q value and it's representation with respect to Q value and signed flag can be seen in Table F-8 for signed values or in Table F-9 for unsigned values.

The Q value is stored on 6 bits but only the 4-0 bits are used, the 6th bit is reserved for further use. By using this Q value storage, the numbers can have a variable length making the result more accurate and enlarging or reducing the number domain when necessary. The 5 bits allow 32 values for the Q therefore, as an example, the absolute maximum number will be 131072 while the absolute minimum number is 0,0000076294 for signed number.

The sign flag represents the fact that the result is signed or unsigned. This will affect the number representation as well as all the other operations.

The exception flag is used only if the requested value would have triggered an exception. This can be used to test the exception trigger mechanism as well as a redundancy check for the result.

The D flag represents the over-domain flag. This will not trigger an exception and it is mainly used for trigonometric functions. For example the sine function should return value in the interval of $[-\pi;\ \pi]$ but the result exists for a range of $[-4; 4]$. For the values outside the main interval of $[-\pi;\ \pi]$ the D flag is set to 1. For all other functions, the flag will be set only if the number is outside the representation capabilities of Q0 or Q17. For example 10^{10} would give a result bigger then 131072. In this case the D flag will be set.

F.6 RELAY OUTPUT

F.6.I PROTOCOL AND COMMUNICATION

The communication is based on a SPI communication over 4 lines, with the FPGA being the slave.

The communication is based on 5 bytes, from which 4 contains the data and the 5th contains the checksum. The data packet is presented in Table F-11.

Offset	Size	Description
0	1	Command
1	3	Parameters
4	1	Checksum

Table F-11: Relay output data packet

The commands represent the request made by the FPGA to the microcontroller. The possible commands can be noticed in Table F-12.

Command	Name	Description
0	Clear	Reset all the relays (turn them off)
1	Set output	Set the relays as stated in the parameters. Each bit of the parameters represent a relay state
2	Shutdown units	Turn off all the relays that power up the units
3	Shutdown contactors	Turn off all the power supply contactors
10	Automatic power on	Automatic power on the units with a 3 seconds delay between them
11	Automatic power on with delay	Automatic power on the units with a preset delay like set in the parameter 0
20	Main power contactor OFF	Turn off the main contactor of the power supply
21	Main power	Turn on only the main contactor of the power supply

	contactor On	
22	Charge contactor OFF	Turn off the charging contactor (state used during power-up)
23	Charging contactor ON	Turn on the charging contactor (charge complete)
24	DC Bus contactors OFF	Turn off both DC bus contactors in order to isolate the inverter from the power supply
25	DC Bus contactors ON	Connect the inverter with the DC bus of the power supply
26	Load contactor OFF	Disconnect the output of the inverter from the load (default state at power-up)
27	Load contactor ON	Connect the output of the inverter with the load
70	Chopper OFF	Turn OFF the contactor which maintains the chopper resistors connected to the DC bus
71	Chopper ON	Connect the chopper resistors to the main DC Bus

Table F-12: Command list

The checksum is calculated by using the following formula:

$$CRC_{sum} = Data[0] + Data[1] + Data[2] + Data[3]$$

$$CRC = (CRC_{sum} \ \& \ 0x0F) + \frac{CRC_{sum} \ \& \ 0x3C0}{16}$$

Equation F-3: CRC sum check for the relay output microcontroller

F.7 RELAY INPUT

The commands used by the relay input microcontroller are not many because it is an input only device with respect to the contactors while for the USB connection (not implemented) is a bidirectional communication. The commands are presented in Table F-13.

Command	Name	Description
0	Read	Read all the relays
1	Write USB	Write data from the 3 parameters to the USB
2	Read USB	Read 3 bytes from the USB

Table F-13: Relay input commands

F.8 DSP MEMORY SPACE

In order to determine what data is located where, a memory space had to be defined for the DSP and FPGA. The entire memory space of 1 Mbit is more than enough for the application and only a fraction of it will be used. Because the FPGA is capable of handling integer multiplication on 18 bits (with the sign included), the entire memory is therefore split into 32 bit sections called registers.

	Bits																																	
Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name	Section
	Offset + 1																Offset + 0																	
0	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Control Voltage U	Control
2	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Control Voltage V	
4	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Control Voltage W	
6	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Control 1 / VDC	
8	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage R	Main voltages
10	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage S	
12	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage T	
14	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage DC	
16	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage U	
18	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage V	
20	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage W	
22	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit U1	Units U
24	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit U2	
26	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit U3	
28	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit U4	
30	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit U5	
32	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit U6	
34	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit U7	
36	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit U8	
38	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit V1	Units V
40	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit V2	
42	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit V3	
44	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit V4	
46	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit V5	
48	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit V6	
50	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit V7	
52	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit V8	
54	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit W1	Units W
56	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit W2	
58	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit W3	
60	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit W4	
62	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit W5	
64	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit W6	
66	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit W7	
68	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Voltage Unit W8	
70	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Current R	Currents
72	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Current S	
74	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Current T	
76	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Current DC	
78	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Current U	
80	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Current V	
82	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Current W	
84	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Current U-hi	
86	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Current U-low	
88	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Current V-hi	
90	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Current V-low	
92	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Current W-hi	
94	-	-	-	-	-	-	-	S	Q	Q	Q	Q	Q	Q	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Current W-low	
96	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	Faults U	Faults
98	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	Faults V	
100	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	Faults W	
102	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	General Faults	
104	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	Triggers	Triggers
106	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Flags	Flags
108	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Status	Status

Table F-14: Memory map

The symbols used in Table F-14 are:

- 'A': available for both read and write
- 'R': read-value

- ‘W’: write-value (read as 0)
- ‘-‘: reserved (should be 0)
- ‘Q’: the number format of the corresponding number (see the Number format subchapter)
- ‘S’: specifies if the number is signed or unsigned

A) CONTROL VALUES

The control values memory map contains the main data in order to proper control the inverter. The data represents the desired voltage on each inverter leg. The data is calculated by the DSP and sent to the FPGA in order to apply the modulation algorithm. The 4th value, 1/VDC represents the already divided value for the VDC in order to ease the computation for the FPGA.

B) MAIN VOLTAGES

The main voltage represents the measured voltages of the power supply. For the position of these sensors see Power Supply subchapter in the Hardware chapter. These values are measured either by the FPGA or the DSP. The voltage R, S and T are measured by the FPGA while the rest by the DSP. In order to align every data to the FPGA standard every value that is passed from the DSP to the FPGA or vice-versa the data format will be on 18 bits with sign.

C) UNIT VOLTAGES

The unit voltages represent the measured values of the voltages across the units. These are measured by the FPGA and have the offset and gain already applied.

D) CURRENTS

The currents represent the actual values measured by the FPGA and DSP. The main currents, the DC current, U, V and W currents are measured by the DSP and used by the control structure. The currents for the upper and lower sections of the inverter legs as well as the input currents of the power supply units are measured by the FPGA. The currents measured by the FPGA are used for modulation and current control of the inverter units.

E) FAULTS

The faults represents the detailed values of the entire system with respect to units and modules. The faults U, V and W represent the detailed faults of all the units. The bits 0, 2, 4,…, 30 correspond to the main fault signals of the units while the bits on the odd positions (1, 3, 5 …, 31) correspond to over-temperature fault of the units.

The general faults represent the faults caused by the logic system due to overvoltage, overcurrent, etc. These values can only be read. In order to set a value, the triggers should be used. The lowest 16 bits are used by the FPGA internal logic while the upper 16 bits can be set with the help of the upper 16 bits of the trigger register.

F) TRIGGERS

The triggers are used to signal events inside the FPGA. These are defined at kernel level and have the highest priority. The difference between flags and triggers is that flags maintain their value after the effect had passed. Triggers will be switched to 0 after the corresponding task had completed its execution. Triggers can only be set not cleared. The value read of a certain trigger signals the state of the corresponding operation. A high state (1) represents that the operation is in progress. A low state (0) tells that the operation has ended or is not in progress.

Bit No.	Name	On Write (only for high state – 1, for low state it has no effect)
0	Reset	The system will reset the fault condition
1	Shutdown	The system will enter shutdown state
2	Units power up	The system will attempt to power up each unit at a turn. There is about 2-3 seconds between units switching on in order to allow the power supply to stabilize.
3	Units power down	The system will attempt to power down each group of units at a turn. It will take approximately 500 ms per unit to shutdown.
4	System power-up	The system will be powered up. The trigger will be cleared after all the units and the DC bus capacitors were charged.

Table F-15: Triggers register map

G) FLAGS

The flags set the main system states like enable, running etc.

Bit No.	Name	On Read	On Write
0	Enable	0 – The system is idle 1 – The system is running	0 – The system will shut-down 1 – The system will start
1	Fault	0 – No fault 1 – A fault had occurred	0 – No effect 1 – A fault signal will be issued

Table F-16: Flags register map

H) STATUS

The status register is used to show the status of individual components of the system. It is a read-write register in order to allow easy access to the states.

Bit No.	Name	Value meaning
0	Enable	0 – The system is idle 1 – The system is running
1	Trip	0 – No fault 1 – A fault had caused the system to stop
2	Relays operational	0 – The relays are stopped 1 – At least one relay is working
3	Chopper	0 – The chopper is OFF 1 – The chopper is ON
4	Main power	0 – The system is unpowered 1 – The system is powered
5	Charging	0 – The system is uncharged 1 – The system is charged
6	Load	0 – The load is disconnected 1 – The load is connected

7	Inverter connected	0 – The inverter DC bus is not connected 1 – The inverter DC bus is connected
8, 9	System state	00 – Disabled 01 – Test mode 1 10 – Test mode 2 11 – Normal mode
10	Output enable	0 – The gate driver signals are disabled 1 – The gate driver signals are enabled

Table F-17: Status register map

APPENDIX G. FPGA SOURCE CODE SIMULATIONS

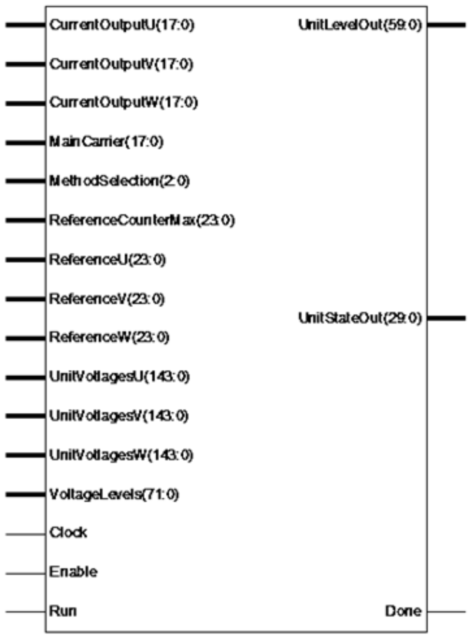


Figure G.1 Modulation Block

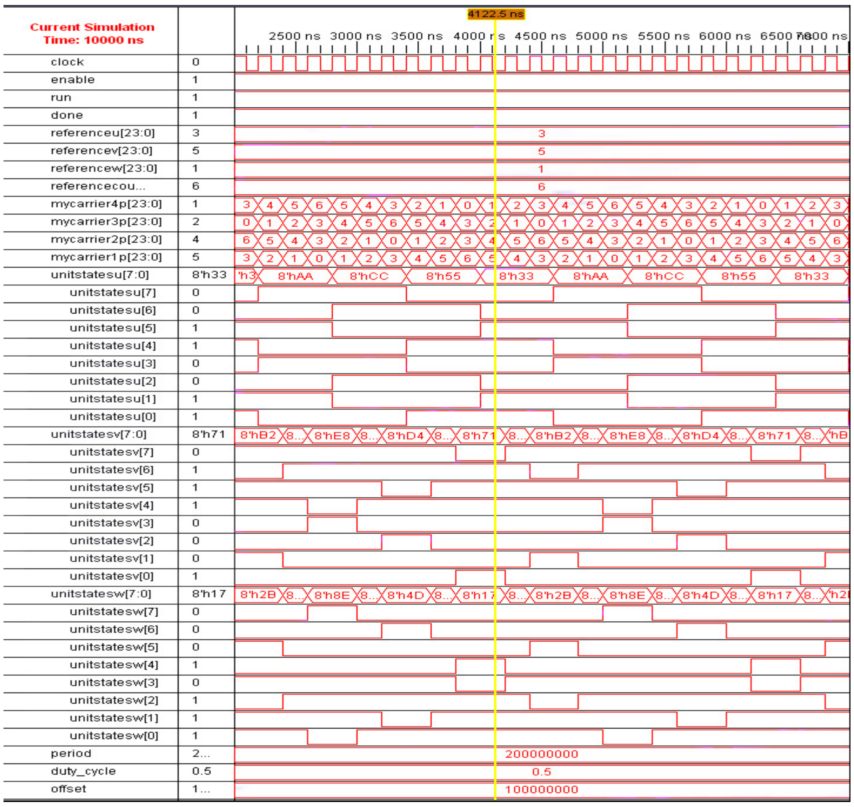


Figure G.2 Phase shifted Modulation

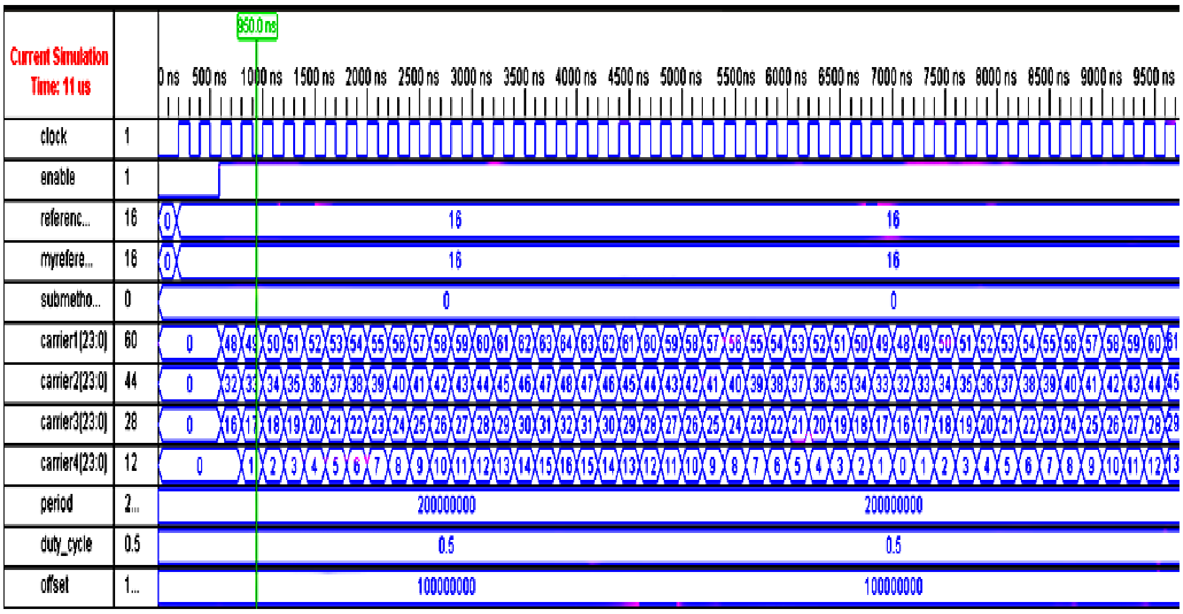


Figure G.3 Level Shifted Modulation IPD

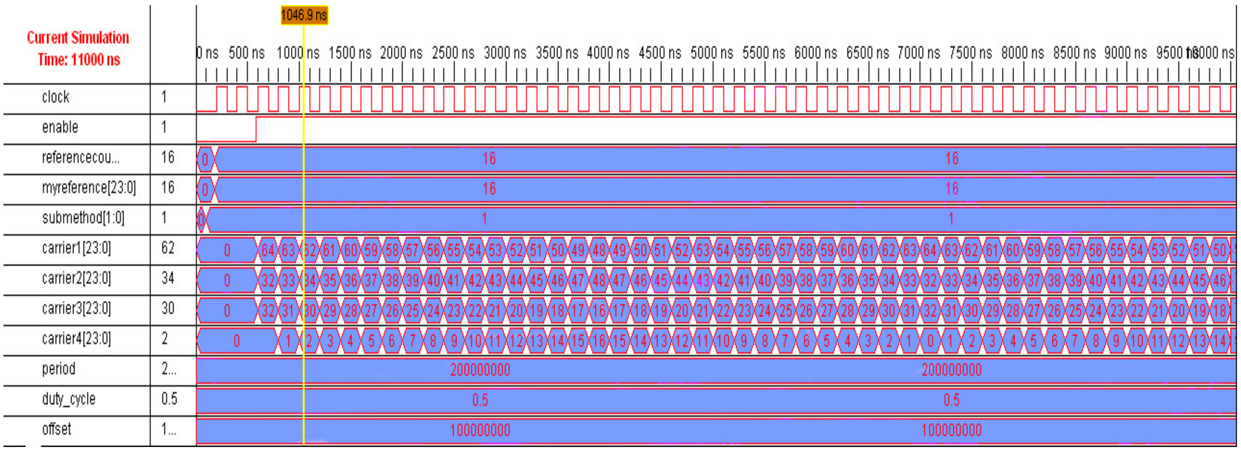


Figure G.4 Level Shifted Modulation APOD

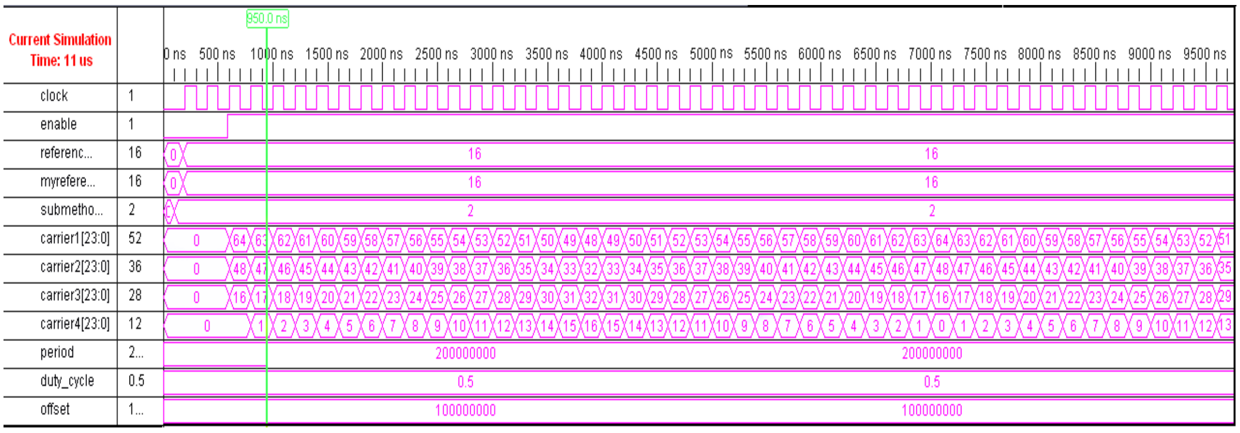
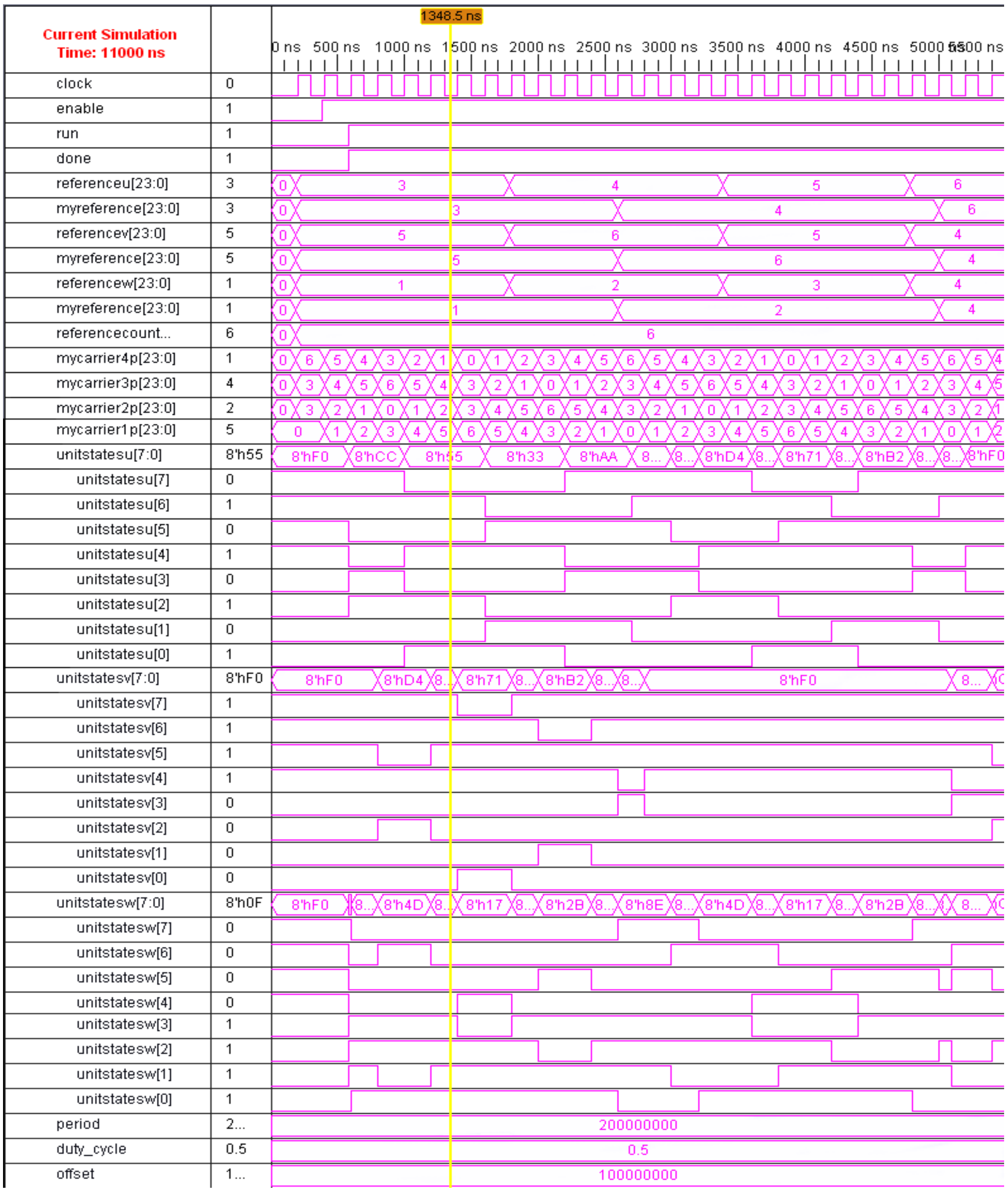


Figure G.5 Level Shifted Modulation POD



clock	0
enable	1
run	1
done	1
referenceu[23:0]	3
myreference[23:0]	3
referencev[23:0]	5
myreference[23:0]	5
referencew[23:0]	1
myreference[23:0]	1
referencecount...	6
mycarrier4p[23:0]	1
mycarrier3p[23:0]	4
mycarrier2p[23:0]	2
mycarrier1p[23:0]	5
unitstatesu[7:0]	8'h55
unitstatesu[7]	0
unitstatesu[6]	1
unitstatesu[5]	0
unitstatesu[4]	1
unitstatesu[3]	0
unitstatesu[2]	1
unitstatesu[1]	0
unitstatesu[0]	1
unitstatesv[7:0]	8'hF0
unitstatesv[7]	1
unitstatesv[6]	1
unitstatesv[5]	1
unitstatesv[4]	1
unitstatesv[3]	0
unitstatesv[2]	0
unitstatesv[1]	0
unitstatesv[0]	0
unitstatesw[7:0]	8'h0F
unitstatesw[7]	0
unitstatesw[6]	0
unitstatesw[5]	0
unitstatesw[4]	0
unitstatesw[3]	1
unitstatesw[2]	1
unitstatesw[1]	1
unitstatesw[0]	1
period	2...
duty_cycle	0.5
offset	1...

Figure G.6 Unit states

APPENDIX H. MAIN SOFTWARE

H.1 DSP

H.1.I WORKER.C

```
#include "main.h"


tTranformInvClark myAB2ABC;


//
// This function will be called each cycle
// All the tasks that the DSP must execute are found
in this
// function.
void Worker(void){
//  int i;
//  double dAlpha, dBeta;
    double dU, dV, dW;

    ///////////////////////////////////////////////
    //  Read the current and the DC Voltage
    ///////////////////////////////////////////////
    mAdc.read(&mAdc);

    // Now control the system
    // Good look


    ///////////////////////////////////////////////
    //  Compute the speed ramp
    ///////////////////////////////////////////////
    ramp_speed.calc(&ramp_speed);


    ///////////////////////////////////////////////
    //  Compute frequency ramp
    ///////////////////////////////////////////////
    ramp_freq.calc(&ramp_freq);



    ///////////////////////////////////////////////
    //  Connect frequency ramp with angle theta
generator
    ///////////////////////////////////////////////
    ramp_theta.fFrequency =
(ramp_freq.fOutputFrequency);
    ramp_theta.calc(&ramp_theta);


    ///////////////////////////////////////////////
    //  Connect frequency ramp with U/F calculator
    ///////////////////////////////////////////////
    vhz1.Freq = ramp_freq.fOutputFrequency; //
(ramp_freq.fOutputFrequency + fil.slip_comp_output);
    vhz1.calc(&vhz1);



    ///////////////////////////////////////////////
    //  Connect theta generator with the voltage
calculator
    ///////////////////////////////////////////////
    volt.fAngleTheta = ramp_theta.fOutputAngle;
    volt.fVoltageReference = vhz1.VoltOut;
    volt.calc(&volt);
//      spv.fVoltageAlpha = volt.fVoltageAlpha;
//      spv.fVoltageBeta = volt.fVoltageBeta;

    AB2ABC(volt.fVoltageAlpha, volt.fVoltageBeta, dU,
dV, dW, dSqrt3Div2);

    //////////////////////////////////////////////
    //
    // Copy the data from the ADC to the FPGA
    //
    //////////////////////////////////////////////

    // Copy voltages
    myStruct->VDC =
DSP2FPGA_float(mAdc.fDCBusVoltage);
    myStruct->VoltageU =
DSP2FPGA_float(mAdc.fVoltageU);
    myStruct->VoltageV =
DSP2FPGA_float(mAdc.fVoltageV);
    myStruct->VoltageW =
DSP2FPGA_float(mAdc.fVoltageW);

    // Copy currents
    myStruct->CurrentDC =
DSP2FPGA_float(mAdc.fDCBusCurrent);
    myStruct->CurrentU =
DSP2FPGA_float(mAdc.fCurrentU);
    myStruct->CurrentV =
DSP2FPGA_float(mAdc.fCurrentV);
    myStruct->CurrentW =
DSP2FPGA_float(mAdc.fCurrentW);

    if (mAdc.fDCBusVoltage == 0){
        myStruct->InvVDC = DSP2FPGA_float(1);
    } else {
        myStruct->InvVDC = DSP2FPGA_float(1 /
mAdc.fDCBusVoltage);;
    }


    // Set the control voltages in PU + 1 (range: [2
0])
    myStruct->ControlVoltageU = DSP2FPGA_float(dU /
565 + 1);
    myStruct->ControlVoltageV = DSP2FPGA_float(dV /
565 + 1);
    myStruct->ControlVoltageW = DSP2FPGA_float(dW /
565 + 1);

    nCnt2++;
    if (nCnt2 > 10) {
        nCnt2 = 0;

        nCnt++;
        if (nCnt >= 200) nCnt = 0;

        dUU[nCnt] = volt.fVoltageAlpha; // dU;
        dUV[nCnt] = volt.fVoltageBeta; // dV;
        dUW[nCnt] = dW;
    }
}
```

H.1.II TIMER.C

```
#include "main.h"


struct CPUTIMER_VARS CpuTimer0;
struct CPUTIMER_VARS CpuTimer1;
struct CPUTIMER_VARS CpuTimer2;


interrupt void mainTimerInterrupt(void){
    CpuTimer0.InterruptCount++;

    // Call the main worker
    Worker();

    // Acknowledge this interrupt to receive more
interrupts from group 1
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}



void InitializeTimer(){
    // 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.TINT0 = &mainTimerInterrupt;
    EDIS;    // This is needed to disable write to
EALLOW protected registers

    // Step 4. Initialize the Device Peripheral. This
function can be
    //         found in DSP2833x_CpuTimers.c
    InitCpuTimers();   // For this example, only
initialize the Cpu Timers



    // Configure CPU-Timer 0 to interrupt every 500
milliseconds:
    // 150MHz CPU Freq, 50 millisecond Period (in
uSeconds)
    ConfigCpuTimer(&CpuTimer0, 150, 500);



    // To ensure precise timing, use write-only
instructions to write to the entire register.
Therefore, if any
    // of the configuration bits are changed in
ConfigCpuTimer and InitCpuTimers (in
DSP2833x_CpuTimers.h), the
    // below settings must also be updated.

    CpuTimer0Regs.TCR.all = 0x4001; // Use write-only
instruction to set TSS bit = 0

    // Enable CPU INT1 which is connected to CPU-
Timer 0:
    IER |= M_INT1;

    // Enable TINT0 in the PIE: Group 1 interrupt 7
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
}





void InitCpuTimers(void){
    // CPU Timer 0
    // Initialize address pointers to respective
timer registers:
    CpuTimer0.RegsAddr = &CpuTimer0Regs;
    // Initialize timer period to maximum:
    CpuTimer0Regs.PRD.all  = 0xFFFFFFFF;
    // Initialize pre-scale counter to divide by 1
(SYSCLKOUT):
    CpuTimer0Regs.TPR.all  = 0;
    CpuTimer0Regs.TPRH.all = 0;
    // Make sure timer is stopped:
    CpuTimer0Regs.TCR.bit.TSS = 1;
    // Reload all counter register with period value:
    CpuTimer0Regs.TCR.bit.TRB = 1;
    // Reset interrupt counters:
    CpuTimer0.InterruptCount = 0;


// CpuTimer 1 and CpuTimer2 are reserved for DSP BIOS
& other RTOS
// Do not use these two timers if you ever plan on
integrating
// DSP-BIOS or another realtime OS.
//
// Initialize address pointers to respective timer
registers:
    CpuTimer1.RegsAddr = &CpuTimer1Regs;
    CpuTimer2.RegsAddr = &CpuTimer2Regs;
    // Initialize timer period to maximum:
    CpuTimer1Regs.PRD.all  = 0xFFFFFFFF;
    CpuTimer2Regs.PRD.all  = 0xFFFFFFFF;
    // Initialize pre-scale counter to divide by 1
(SYSCLKOUT):
    CpuTimer1Regs.TPR.all  = 0;
    CpuTimer1Regs.TPRH.all = 0;
    CpuTimer2Regs.TPR.all  = 0;
    CpuTimer2Regs.TPRH.all = 0;
    // Make sure timers are stopped:
    CpuTimer1Regs.TCR.bit.TSS = 1;
    CpuTimer2Regs.TCR.bit.TSS = 1;
    // Reload all counter register with period value:
    CpuTimer1Regs.TCR.bit.TRB = 1;
    CpuTimer2Regs.TCR.bit.TRB = 1;
    // Reset interrupt counters:
    CpuTimer1.InterruptCount = 0;
    CpuTimer2.InterruptCount = 0;

}

//-----------------------------------------------------
------------------------
// ConfigCpuTimer:
//-----------------------------------------------------
------------------------
// This function initializes the selected timer to
the period specified
// by the "Freq" and "Period" parameters. The "Freq"
is entered as "MHz"
// and the period in "uSeconds". The timer is held in
the stopped state
// after configuration.
//
void ConfigCpuTimer(struct CPUTIMER_VARS *Timer,
float Freq, float Period){
    Uint32  temp;

    // Initialize timer period:
    Timer->CPUFreqInMHz = Freq;
    Timer->PeriodInUSec = Period;
    temp = (long) (Freq * Period);
    Timer->RegsAddr->PRD.all = temp;

    // Set pre-scale counter to divide by 1
(SYSCLKOUT):
    Timer->RegsAddr->TPR.all  = 0;
    Timer->RegsAddr->TPRH.all  = 0;

    // Initialize timer control register:
    Timer->RegsAddr->TCR.bit.TSS = 1;      // 1 =
Stop timer, 0 = Start/Restart Timer
    Timer->RegsAddr->TCR.bit.TRB = 1;      // 1 =
reload timer
    Timer->RegsAddr->TCR.bit.SOFT = 0;
    Timer->RegsAddr->TCR.bit.FREE = 0;     // Timer
Free Run Disabled
    Timer->RegsAddr->TCR.bit.TIE = 1;      // 0 =
Disable/ 1 = Enable Timer Interrupt

    // Reset interrupt counter:
    Timer->InterruptCount = 0;
}
```

H.1.III EXTERNAL INTERFACE

```
#include "main.h"


void extInterface_SetIO(){
    // Make sure the XINTF clock is enabled
    SysCtrlRegs.PCLKCR3.bit.XINTFENCLK = 1;

    EALLOW;
    GpioCtrlRegs.GPCMUX1.bit.GPIO64 = 3;  // XD15
    GpioCtrlRegs.GPCMUX1.bit.GPIO65 = 3;  // XD14
    GpioCtrlRegs.GPCMUX1.bit.GPIO66 = 3;  // XD13
    GpioCtrlRegs.GPCMUX1.bit.GPIO67 = 3;  // XD12
    GpioCtrlRegs.GPCMUX1.bit.GPIO68 = 3;  // XD11
    GpioCtrlRegs.GPCMUX1.bit.GPIO69 = 3;  // XD10
    GpioCtrlRegs.GPCMUX1.bit.GPIO70 = 3;  // XD19
    GpioCtrlRegs.GPCMUX1.bit.GPIO71 = 3;  // XD8
    GpioCtrlRegs.GPCMUX1.bit.GPIO72 = 3;  // XD7
    GpioCtrlRegs.GPCMUX1.bit.GPIO73 = 3;  // XD6
    GpioCtrlRegs.GPCMUX1.bit.GPIO74 = 3;  // XD5
    GpioCtrlRegs.GPCMUX1.bit.GPIO75 = 3;  // XD4
    GpioCtrlRegs.GPCMUX1.bit.GPIO76 = 3;  // XD3
    GpioCtrlRegs.GPCMUX1.bit.GPIO77 = 3;  // XD2
    GpioCtrlRegs.GPCMUX1.bit.GPIO78 = 3;  // XD1
    GpioCtrlRegs.GPCMUX1.bit.GPIO79 = 3;  // XD0

    GpioCtrlRegs.GPBMUX1.bit.GPIO40 = 3;  // 
XA0/XWE1n
    GpioCtrlRegs.GPBMUX1.bit.GPIO41 = 3;  // XA1
    GpioCtrlRegs.GPBMUX1.bit.GPIO42 = 3;  // XA2
    GpioCtrlRegs.GPBMUX1.bit.GPIO43 = 3;  // XA3
    GpioCtrlRegs.GPBMUX1.bit.GPIO44 = 3;  // XA4
    GpioCtrlRegs.GPBMUX1.bit.GPIO45 = 3;  // XA5
    GpioCtrlRegs.GPBMUX1.bit.GPIO46 = 3;  // XA6
    GpioCtrlRegs.GPBMUX1.bit.GPIO47 = 3;  // XA7

    GpioCtrlRegs.GPCMUX2.bit.GPIO80 = 3;  // XA8
    GpioCtrlRegs.GPCMUX2.bit.GPIO81 = 3;  // XA9
    GpioCtrlRegs.GPCMUX2.bit.GPIO82 = 3;  // XA10
    GpioCtrlRegs.GPCMUX2.bit.GPIO83 = 3;  // XA11
    GpioCtrlRegs.GPCMUX2.bit.GPIO84 = 3;  // XA12
    GpioCtrlRegs.GPCMUX2.bit.GPIO85 = 3;  // XA13
    GpioCtrlRegs.GPCMUX2.bit.GPIO86 = 3;  // XA14
    GpioCtrlRegs.GPCMUX2.bit.GPIO87 = 3;  // XA15
    GpioCtrlRegs.GPBMUX1.bit.GPIO39 = 3;  // XA16
    GpioCtrlRegs.GPAMUX2.bit.GPIO31 = 3;  // XA17
    GpioCtrlRegs.GPAMUX2.bit.GPIO30 = 3;  // XA18
    GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 3;  // XA19

    GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 3;  // XREADY
    GpioCtrlRegs.GPBMUX1.bit.GPIO35 = 3;  // XRNW
    GpioCtrlRegs.GPBMUX1.bit.GPIO38 = 3;  // XWE0

    GpioCtrlRegs.GPBMUX1.bit.GPIO36 = 3;  // XZCS0
    GpioCtrlRegs.GPBMUX1.bit.GPIO37 = 3;  // XZCS7
    GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 3;  // XZCS6
    EDIS;

    // For all zone, set timings

    EALLOW;
    // All Zones---------------------------------
    // Timing for all zones based on XTIMCLK = 
SYSCLKOUT
    XintfRegs.XINTCNF2.bit.XTIMCLK = 0;
    // Buffer up to 3 writes
    XintfRegs.XINTCNF2.bit.WRBUFF = 3;
    // XCLKOUT is enabled
    XintfRegs.XINTCNF2.bit.CLKOFF = 0;
    // XCLKOUT = XTIMCLK
    XintfRegs.XINTCNF2.bit.CLKMODE = 0;
    EDIS;
}
```

H.1.IV EXTINTERFACE ASRAM

```
#include "main.h"


void extInterface_Asram(void){
    EALLOW;
    // Zone 7------------------------------------
    // When using ready, ACTIVE must be 1 or greater
    // Lead must always be 1 or greater
    // Zone write timing
    XintfRegs.XTIMING7.bit.XWRLEAD = 1; // 1
    XintfRegs.XTIMING7.bit.XWRACTIVE = 2; // 2
    XintfRegs.XTIMING7.bit.XWRTRAIL = 0; // 1

    // Zone read timing
    XintfRegs.XTIMING7.bit.XRDLEAD = 1;
    XintfRegs.XTIMING7.bit.XRDACTIVE = 3; // 3
    XintfRegs.XTIMING7.bit.XRDTRAIL = 0;

    // don't double all Zone read/write 
lead/active/trail timing
    XintfRegs.XTIMING7.bit.X2TIMING = 0;

    // Zone will not sample XREADY signal
    XintfRegs.XTIMING7.bit.USEREADY = 0;
    XintfRegs.XTIMING7.bit.READYMODE = 0;

    // 1,1 = x16 data bus
    // 0,1 = x32 data bus
    // other values are reserved
    XintfRegs.XTIMING7.bit.XSIZE = 3;
    EDIS;

    //Force a pipeline flush to ensure that the write 
to
    //the last register configured occurs before 
returning.
    asm(" RPT #7 || NOP");
}
```

H.1.V FPGA INTERFACE

```
#include "main.h"
#include "fpga.h"


// tFpgaOutput * myFpgaData = (tFpgaOutput*)0x100000;
// Set it to ZONE 6

TMyStruct sss0;
TMyStructFpga sss1;

TMyStruct * myStruct = & sss0; // = 
(TMyStruct*)0x100000;   // Set it to ZONE 6
TMyStructFpga * myStructFpga = & sss1; //  = 
(TMyStructFpga*)0x100000;  // Set it to ZONE 6

void extInterface_Fpga(void){
    EALLOW;
    // Zone 6 -----------------------------------
    // When using ready, ACTIVE must be 1 or greater
    // Lead must always be 1 or greater

    // Zone write timing
    XintfRegs.XTIMING6.bit.XWRLEAD = 1; // 1
    XintfRegs.XTIMING6.bit.XWRACTIVE = 2; // 2
    XintfRegs.XTIMING6.bit.XWRTRAIL = 0; // 1

    // Zone read timing
    XintfRegs.XTIMING6.bit.XRDLEAD = 1;
    XintfRegs.XTIMING6.bit.XRDACTIVE = 3; // 3
    XintfRegs.XTIMING6.bit.XRDTRAIL = 0;

    // don't double all Zone read/write
lead/active/trail timing
    XintfRegs.XTIMING6.bit.X2TIMING = 0;

    // Zone will not sample XREADY signal
    XintfRegs.XTIMING6.bit.USEREADY = 0;
    XintfRegs.XTIMING6.bit.READYMODE = 0;
```

```
    // 1,1 = x16 data bus
    // 0,1 = x32 data bus
    // other values are reserved
    XintfRegs.XTIMING6.bit.XSIZE = 3;
    EDIS;

    //Force a pipeline flush to ensure that the write
to
    //the last register configured occurs before
returning.
    asm(" RPT #7 || NOP");
}
```

H.1.VI MAIN .C

```
#include "main.h"


// Prototype statements for functions found within
this file.
interrupt void cpu_timer0_isr(void);


double my1;
float my2;
float my3;
double myRes;


void main(void) {
    // Initialize global variables
    InitializeGlobal();

    // Initialize user space
    InitializeUser();


    // Enable global Interrupts and higher priority
real-time debug events:
    EINT;   // Enable Global interrupt INTM
    ERTM;   // Enable Global realtime interrupt DBGM


    // Step 6. IDLE loop. Just sit and loop forever:
    for(;;);
}
```

H.1.VII OTHERS.C

```
#include "main.h"


// +W2230


unsigned long nPowerOf2[32];
// = {
//  1,2,4,8,16,32,64,128,256,512,1024,2048,
//
4096,8192,16384,32768,65536,131072,263144,524288,1048
576,2097152,4194304
//};


long GetSystemFreq(){
    long nSystemSpeed;

    nSystemSpeed = MAIN_CRISTAL_FREQ_KHZ;
    nSystemSpeed *= 0.5;

    // Calculate Coefficient based on Div Sel
    switch (DSP28_DIVSEL){
    case 0:
    case 1:
        nSystemSpeed *= 0.25;
        break;
    case 2:
        nSystemSpeed *= 0.5;
        break;
//  case 3: // do nothing
//      nSystemSpeed *= 1;
//      break;
    }

    if ((DSP28_PLLCR > 1) && (DSP28_PLLCR <= 10)) {
        nSystemSpeed *= DSP28_PLLCR;
    }

    nSystemSpeed *= 1000;

    return nSystemSpeed;
}
```

```
double FPGA2DSP_float(TFpgaNumber myNumber){
    double nValue = 0;

    if (myNumber.value == 0) return 0;

    nValue = myNumber.value;

    // If signed ...
    if (myNumber.sign){
        nValue = - (nPowerOf2[18] - myNumber.value) /
nPowerOf2[myNumber.q];
    } else {
        nValue = myNumber.value /
nPowerOf2[myNumber.q];
    }

    return nValue;
}


unsigned long DSP2FPGA_int(long nNumber){
    union {
        TFpgaNumber myNumber;
        unsigned long myResult;
    } myParam;


    myParam.myResult = 0;
    myParam.myNumber.q = 0;

    if (nNumber < 0){
        myParam.myNumber.sign = 1;

        if (nNumber <= -nPowerOf2[18]){
            myParam.myNumber.value = 0x20000;
        } else {
            myParam.myNumber.value = nPowerOf2[18] +
nNumber;
        }
    } else {
        myParam.myNumber.sign = 0;
        if (nNumber >= nPowerOf2[18]){
            myParam.myNumber.value = 0x3FFFF;
        } else {
            myParam.myNumber.value = nNumber &
0x3FFFF;
        }
    }

    return myParam.myResult;
}



unsigned long DSP2FPGA_float(double nNumber){
    union {
        TFpgaNumber myNumber;
        unsigned long myResult;
    } myParam;
//  double dT1, dT2;
    unsigned long nT3;

    if (nNumber == 0) return 0;

    myParam.myResult = 0;

    if (nNumber < 0){
        // For negative numbers
        myParam.myNumber.sign = 1;

        //  3 = (Q15) = 98304 = 011000000000000000
        // -3 = (Q15) = 163840 = 101000000000000000
        //  3 + (-3) = 0 - checked


             if (nNumber >= -(double)nPowerOf2[ 0]) {
myParam.myNumber.q =  17; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[17]; }
        else if (nNumber >= -(double)nPowerOf2[ 1]) {
myParam.myNumber.q =  16; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[16]; }
        else if (nNumber >= -(double)nPowerOf2[ 2]) {
myParam.myNumber.q =  15; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[15]; }
        else if (nNumber >= -(double)nPowerOf2[ 3]) {
myParam.myNumber.q =  14; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[14]; }
        else if (nNumber >= -(double)nPowerOf2[ 4]) {
myParam.myNumber.q =  13; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[13]; }
        else if (nNumber >= -(double)nPowerOf2[ 5]) {
myParam.myNumber.q =  12; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[12]; }
        else if (nNumber >= -(double)nPowerOf2[ 6]) {
myParam.myNumber.q =  11; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[11]; }
        else if (nNumber >= -(double)nPowerOf2[ 7]) {
myParam.myNumber.q =  10; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[10]; }
        else if (nNumber >= -(double)nPowerOf2[ 8]) {
myParam.myNumber.q =   9; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[ 9]; }
        else if (nNumber >= -(double)nPowerOf2[ 9]) {
myParam.myNumber.q =   8; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[ 8]; }
        else if (nNumber >= -(double)nPowerOf2[10]) {
myParam.myNumber.q =   7; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[ 7]; }
        else if (nNumber >= -(double)nPowerOf2[11]) {
myParam.myNumber.q =   6; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[ 6]; }
        else if (nNumber >= -(double)nPowerOf2[12]) {
myParam.myNumber.q =   5; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[ 5]; }
        else if (nNumber >= -(double)nPowerOf2[13]) {
myParam.myNumber.q =   4; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[ 4]; }
        else if (nNumber >= -(double)nPowerOf2[14]) {
myParam.myNumber.q =   3; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[ 3]; }
        else if (nNumber >= -(double)nPowerOf2[15]) {
myParam.myNumber.q =   2; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[ 2]; }
        else if (nNumber >= -(double)nPowerOf2[16]) {
myParam.myNumber.q =   1; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[ 1]; }
        else if (nNumber >= -(double)nPowerOf2[17]) {
myParam.myNumber.q =   0; myParam.myNumber.value =
nPowerOf2[18] + nNumber * nPowerOf2[ 0]; }
        else {
            // ERROR: To small number so make the
maximum value
            myParam.myNumber.q = 0x20000;
            myParam.myNumber.value = 0;
        }

        //nRes = part & 0x3FFFF + bSign * 0x1000000;
        //nRes += (nQvalue & 0x3F) << 18;
    } else {
        myParam.myNumber.sign = 0;

             if (nNumber < nPowerOf2[ 0]) {
myParam.myNumber.q =  18; myParam.myNumber.value =
nNumber * nPowerOf2[18]; }
        else if (nNumber < nPowerOf2[ 1]) {
myParam.myNumber.q =  17; myParam.myNumber.value =
nNumber * nPowerOf2[17]; }
        else if (nNumber < nPowerOf2[ 2]) {
myParam.myNumber.q =  16; myParam.myNumber.value =
nNumber * nPowerOf2[16]; }
        else if (nNumber < nPowerOf2[ 3]) {
myParam.myNumber.q =  15; myParam.myNumber.value =
nNumber * nPowerOf2[15]; }
        else if (nNumber < nPowerOf2[ 4]) {
myParam.myNumber.q =  14; myParam.myNumber.value =
nNumber * nPowerOf2[14]; }
        else if (nNumber < nPowerOf2[ 5]) {
myParam.myNumber.q =  13; myParam.myNumber.value =
nNumber * nPowerOf2[13]; }
        else if (nNumber < nPowerOf2[ 6]) {
myParam.myNumber.q =  12; myParam.myNumber.value =
nNumber * nPowerOf2[12]; }
        else if (nNumber < nPowerOf2[ 7]) {
myParam.myNumber.q =  11; myParam.myNumber.value =
nNumber * nPowerOf2[11]; }
        else if (nNumber < nPowerOf2[ 8]) {
myParam.myNumber.q =  10; myParam.myNumber.value =
nNumber * nPowerOf2[10]; }
        else if (nNumber < nPowerOf2[ 9]) {
myParam.myNumber.q =   9; myParam.myNumber.value =
nNumber * nPowerOf2[ 9]; }
        else if (nNumber < nPowerOf2[10]) {
myParam.myNumber.q =   8; myParam.myNumber.value =
nNumber * nPowerOf2[ 8]; }
        else if (nNumber < nPowerOf2[11]) {
myParam.myNumber.q =   7; myParam.myNumber.value =
nNumber * nPowerOf2[ 7]; }
        else if (nNumber < nPowerOf2[12]) {
myParam.myNumber.q =   6; myParam.myNumber.value =
nNumber * nPowerOf2[ 6]; }
        else if (nNumber < nPowerOf2[13]) {
myParam.myNumber.q =   5; myParam.myNumber.value =
nNumber * nPowerOf2[ 5]; }
        else if (nNumber < nPowerOf2[14]) {
myParam.myNumber.q =   4; myParam.myNumber.value =
nNumber * nPowerOf2[ 4]; }
        else if (nNumber < nPowerOf2[15]) {
myParam.myNumber.q =   3; myParam.myNumber.value =
nNumber * nPowerOf2[ 3]; }
        else if (nNumber < nPowerOf2[16]) {
myParam.myNumber.q =   2; myParam.myNumber.value =
nNumber * nPowerOf2[ 2]; }
        else if (nNumber < nPowerOf2[17]) {
myParam.myNumber.q =   1; myParam.myNumber.value =
nNumber * nPowerOf2[ 1]; }
        else if (nNumber < nPowerOf2[18]) {
myParam.myNumber.q =   0; myParam.myNumber.value =
nNumber * nPowerOf2[ 0]; }
        else {
            // ERROR: To big number so make the
maximum value
            myParam.myNumber.q = 0;
            myParam.myNumber.value = 0x3FFFF;
        }
    }
    return myParam.myResult;
}


/*
void DelayUs(volatile Uint32 Usec){
    while (Usec--){                   // 1us loop at
150MHz CPUCLK
        asm(" RPT #139 || NOP");
    }
}

*/
```

H.2 VHDL CODE

H.2.I MAIN.VHD

```
-------------------------------------------------------
---------------------------
-- Company: Aalborg university
-- Engineer: Cristian Sandu
--
-- Create Date:    09:05:52 04/25/2009
-- Design Name:
-- Module Name:    main - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-------------------------------------------------------
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity main is
    Port (
        sys_clock : in  STD_LOGIC;
        vga_clock : in  STD_LOGIC;
        not_reset: in STD_LOGIC;

        myButtons: in STD_LOGIC_VECTOR(3 downto 0);
        mySwitches: in STD_LOGIC_VECTOR(7 downto 0);
        myLeds: out STD_LOGIC_VECTOR(7 downto 0);

        --
        -- User input
        --
        PS2_MISO: out std_logic;
        PS2_MOSI: in std_logic;
        PS2_Clock: in std_logic;
        PS2_ChipSelect: in std_logic;

        --
        -- Video Connections
        --
        Video_Red : out  STD_LOGIC_VECTOR (3 downto 0);
        Video_Green : out  STD_LOGIC_VECTOR (3 downto
0);
        Video_Blue : out  STD_LOGIC_VECTOR (3 downto
0);
        Video_Vsync : out  STD_LOGIC;
        Video_Hsync : out  STD_LOGIC;

        --
        -- Gates CPLDs  (5 gate CPLDs)
        --
        -- Note: The index represents the CPLD (index 0
represents CPLD 0,
        --       index 1 <-> CPLD 1, index 2 <-> CPLD
3)
        --
        Gates_Clock: out STD_LOGIC_VECTOR(4 downto 0);
        Gates_ChipSelect: out STD_LOGIC_VECTOR(4 downto
0);
        Gates_OutputEnable: out STD_LOGIC_VECTOR(4
downto 0);
        Gates_MOSI: out STD_LOGIC_VECTOR(4 downto 0);
        Gates_MOSI2: out STD_LOGIC_VECTOR(4 downto 0);
        Gates_MISO: in STD_LOGIC_VECTOR(4 downto 0);
        Gates_Fault: in STD_LOGIC_VECTOR(4 downto 0);
        Gates_Reset: out STD_LOGIC_VECTOR(4 downto 0);


        --
        -- ADCs (2 sets of 3 ADCs IC with 2 data input
lines per ADC)
        --
        ADC_Address: out STD_LOGIC_VECTOR(5 downto 0);
-- ADC address lines
        ADC_ChipSelect: out STD_LOGIC_VECTOR(1 downto
0);     -- ADC Chip Select
        ADC_Clock: out STD_LOGIC_VECTOR(1 downto 0);
-- ADC Clock (max: 32 MHz)
        ADC_Data: in STD_LOGIC_VECTOR(11 downto 0);
-- ADC Data lines


        --
        -- DSP interface
        --
        DSP_Data: inout STD_LOGIC_VECTOR(15 downto 0);
        DSP_Addr: in STD_LOGIC_VECTOR(9 downto 0);
        DSP_Clock: in STD_LOGIC;
        DSP_CS: in STD_LOGIC;
        DSP_RD: in STD_LOGIC;
        DSP_WR: in STD_LOGIC
    );
end main;

architecture Behavioral of main is




    -------------------------------------------------------
--------
    --
    --
    --
    --
    --
    --              MAIN DSP Communication
    --
    --
    --
    --
    --


    signal DSP_Output: std_logic_vector(15 downto 0) :=
(others => '0');
    signal DSP_Input: std_logic_vector(15 downto 0) :=
(others => '0');

    --
    --  DSP Memory map data
    --
    signal datControlU: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datControlV: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datControlW: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datInvVDC: std_logic_vector(24 downto 0) :=
(others => '0');
    signal datVoltageR: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageS: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageT: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageDC:std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageU: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageV: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageW: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageU1: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageU2: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageU3: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageU4: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageU5: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageU6: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageU7: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageU8: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageV1: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageV2: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageV3: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageV4: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageV5: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageV6: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageV7: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageV8: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageW1: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageW2: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageW3: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageW4: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageW5: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageW6: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageW7: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datVoltageW8: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datCurrentR:  std_logic_vector(24 downto 0)
:= (others => '0');
    signal datCurrentS:  std_logic_vector(24 downto 0)
:= (others => '0');
    signal datCurrentT:  std_logic_vector(24 downto 0)
:= (others => '0');
    signal datCurrentDC: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datCurrentU:  std_logic_vector(24 downto 0)
:= (others => '0');
    signal datCurrentV:  std_logic_vector(24 downto 0)
:= (others => '0');
    signal datCurrentW:  std_logic_vector(24 downto 0)
:= (others => '0');
    signal datCurrentUhi: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datCurrentUlo: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datCurrentVhi: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datCurrentVlo: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datCurrentWhi: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datCurrentWlo: std_logic_vector(24 downto 0)
:= (others => '0');
    signal datFaultsU:   std_logic_vector(31 downto 0)
:= (others => '0');
    signal datFaultsV:   std_logic_vector(31 downto 0)
:= (others => '0');
    signal datFaultsW:   std_logic_vector(31 downto 0)
:= (others => '0');
    signal datFaults:    std_logic_vector(31 downto 0)
:= (others => '0');
    signal datTriggers:  std_logic_vector(31 downto 0)
:= (others => '0');
    signal datFlags:     std_logic_vector(31 downto 0)
:= (others => '0');
    signal datStatus:    std_logic_vector(31 downto 0)
:= (others => '0');
    signal datReference: std_logic_vector(11 downto 0)
:= (others => '0');
    signal datLeds:      std_logic_vector(7 downto 0)
:= (others => '0');
    signal datVoltageL0: std_logic_vector(17 downto 0)
:= (others => '0');
    signal datVoltageL1: std_logic_vector(17 downto 0)
:= (others => '0');
    signal datVoltageL2: std_logic_vector(17 downto 0)
:= (others => '0');
    signal datVoltageL3: std_logic_vector(17 downto 0)
:= (others => '0');
    signal datVoltageL4: std_logic_vector(17 downto 0)
:= (others => '0');
    signal datVoltageL5: std_logic_vector(17 downto 0)
:= (others => '0');
    signal datVoltageL6: std_logic_vector(17 downto 0)
:= (others => '0');
    signal datVoltageL7: std_logic_vector(17 downto 0)
:= (others => '0');
    signal datVoltageL8: std_logic_vector(17 downto 0)
:= (others => '0');

    signal DspSetReferenceClock: std_logic := '0';
    signal DspSetLeds: std_logic := '0';


    -------------------------------------------------------
--------
    --
    --
    --
    --
    --
    --                System clocks
    --
    --
    --
    --
    --


    component Clocks
        port (
            Clock125 : in  STD_LOGIC;
            Clock25 : in  STD_LOGIC;
            Clock62 : out  STD_LOGIC;
            Clock31 : out  STD_LOGIC;
            Clock15 : out  STD_LOGIC;
            Clock8  : out  STD_LOGIC;
            Clock12 : out  STD_LOGIC;
            Clock25Hz : out  STD_LOGIC;
            Clock2Hz : out STD_LOGIC;

            ControlClock: out STD_LOGIC;
            ControlReference: in STD_LOGIC_VECTOR(11
downto 0)
        );
    end component;

    signal myClock_62_Mhz: std_logic := '0';
    signal myClock_31_Mhz: std_logic := '0';
    signal myClock_15_Mhz: std_logic := '0';
    signal myClock_8_Mhz: std_logic := '0';
    signal myClock_12_Mhz: std_logic := '0';
    signal myClock_25_Hz: std_logic := '0';
    signal myClock_2Hz: std_logic := '0';
    signal myControlClock: std_logic := '0';




    ---------------------------------------------------
--------
    --
    --
    --
    --
    --
    --                  Gate drivers
    --
    --
    --
    --
    --


    component GateSet
        port (
            --
            -- Hardware Connections
            --
            Gates_Clock: out STD_LOGIC_VECTOR(4 downto
0);
            Gates_ChipSelect: out STD_LOGIC_VECTOR(4
downto 0);
            Gates_OutputEnable: out STD_LOGIC_VECTOR(4
downto 0);
            Gates_MOSI: out STD_LOGIC_VECTOR(4 downto
0);
            Gates_MOSI2: out STD_LOGIC_VECTOR(4 downto
0);
            Gates_MISO: in STD_LOGIC_VECTOR(4 downto
0);
            Gates_Fault: in STD_LOGIC_VECTOR(4 downto
0);
            Gates_Reset: out STD_LOGIC_VECTOR(4 downto
0);


            --
            -- Software Connections
            --
            Clock: in STD_LOGIC;                -- The
main input clock
            Running: in STD_LOGIC;           -- The main
running flag (if High the unit is operational)

            UnitLevels: in STD_LOGIC_VECTOR(59 downto
0);
            UnitStates: in STD_LOGIC_VECTOR(29 downto
0);
            Faults: out STD_LOGIC_VECTOR(59 downto 0);
            OverTemp: out STD_LOGIC_VECTOR(59 downto 0)
        );
    end component;
    --
    -- Gate drivers
    --
    signal myGate_Reset: std_logic_vector(4 downto 0)
:= "00000";
    signal myGate_SysReset: std_logic_vector(4 downto
0)  := "00000";
    signal myGate_GFault: std_logic_vector(4 downto 0)
:= "00000";
    signal myGate_OE: std_logic_vector(4 downto 0)  :=
"00000";
    signal myGate_SysOE: std_logic_vector(4 downto 0)
:= "00000";
    signal myGate_IGBT: std_logic_vector(119 downto 0)
:= (others => '0');                -- IGBTs for the raw
data display
    --signal myGate_Fault: std_logic_vector(35 downto
0) := (others => '0');                -- Faults for the
raw data display


    signal myUnitFaults: STD_LOGIC_VECTOR(59 downto 0)
:= (others => '0');
    signal myUnitOverTemp: STD_LOGIC_VECTOR(59 downto
0)  := (others => '0');

    signal myUnitStatesU: std_logic_vector(7 downto 0)
:= (others => '0');
    signal myUnitStatesV: std_logic_vector(7 downto 0)
:= (others => '0');
    signal myUnitStatesW: std_logic_vector(7 downto 0)
:= (others => '0');

    signal myUnitStatesOut: std_logic_vector(29 downto
0)  := (others => '0');
    signal myUnitLevelOut: std_logic_vector(59 downto
0)  := (others => '0');




    ---------------------------------------------------
------------------
    --
    --
    --
    --
    --
    --                  Base control structures
    --
    --
    --
    --
    --
    --
    --
    component MainControl Port (
        Clock: in std_logic;              -- Main system
clock
        Enable: in std_logic;         -- Enable the
conversion (if disable, set to 0 all outputs)

        Run: in std_logic;            -- If enable, it
converts the data input, if not, output the last data

        Done: out std_logic;              -- High when
the conversion is done

        -- Input data
        MethodSelection: in STD_LOGIC_VECTOR(2 downto
0);       -- Method selection

        -- Input data
        ReferenceU: in STD_LOGIC_VECTOR(23 downto 0);
        ReferenceV: in STD_LOGIC_VECTOR(23 downto 0);
        ReferenceW: in STD_LOGIC_VECTOR(23 downto 0);

        ReferenceCounterMax : in  STD_LOGIC_VECTOR (23
downto 0);    -- The maximum value for the counters

        -- Measured values
        UnitVotlagesU: in STD_LOGIC_VECTOR(143 downto
0);
        UnitVotlagesV: in STD_LOGIC_VECTOR(143 downto
0);
        UnitVotlagesW: in STD_LOGIC_VECTOR(143 downto
0);

        -- Current input values
        CurrentOutputU: in STD_LOGIC_VECTOR(17 downto
0);
        CurrentOutputV: in STD_LOGIC_VECTOR(17 downto
0);
        CurrentOutputW: in STD_LOGIC_VECTOR(17 downto
0);

        -- Voltage level definition
        VoltageLevels: in STD_LOGIC_VECTOR(71 downto
0);

        -- Output data
        UnitStatesU : out  STD_LOGIC_VECTOR (7 downto
0);
        UnitStatesV : out  STD_LOGIC_VECTOR (7 downto
0);
        UnitStatesW : out  STD_LOGIC_VECTOR (7 downto
0);


        -- Output data
        UnitStateOut : out  STD_LOGIC_VECTOR (29 downto
0);     -- States for 8 * 3 units
        UnitLevelOut : out  STD_LOGIC_VECTOR (59 downto
0)          -- Levels for 8 * 3 * 2 legs
    );
    end component;
    --
    --
    -- Control parameters
    --
    --
    signal myControlReference: std_logic_vector(11
downto 0) := x"0A0";      -- The counter for the
reference clock
    signal myUserControlReferenceChange: std_logic :=
'0';
    signal myUserControlReference: std_logic_vector(11
downto 0) := x"0A0";
--  signal myDSPControlReference: std_logic_vector(11
downto 0) := x"0A0";

    signal myVoltageCoeff_Inc: std_logic_vector(17
downto 0) := "000000000010000000";        -- = 0.5
    signal myVoltageCoeff_Dec: std_logic_vector(17
downto 0) := "000000000010000000";        -- = 0.5

    signal myVoltageSensors_Gain: std_logic_vector(17
downto 0) := "00" &  x"014F"; -- The gain for the
voltage sensors
    signal myCurrentSensors_Gain: std_logic_vector(17
downto 0) := "00" &  x"0055"; -- The gain for the
current sensors

    signal myVoltage_ChargeLevel: std_logic_vector(17
downto 0) := "000000000010000000";     -- The gain for
the charge complete

    signal myModulationRun : std_logic := '0';
    signal myModulationDone: std_logic := '0';
    signal myModulationMethod: std_logic_vector(2
downto 0) := "000";
    signal myModulationCounterMax: std_logic_vector(23
downto 0) := (others => '0');

    signal myReferenceU : std_logic_vector(23 downto 0)
:= (others => '0');
    signal myReferenceV : std_logic_vector(23 downto 0)
:= (others => '0');
    signal myReferenceW : std_logic_vector(23 downto 0)
:= (others => '0');




    ---------------------------------------------------
------------------
    --
    --
    --
    --
    --
    --              ADCs
```

```
    --
    --
    --
    --
    --
    --
    --
    component ADCs
        port (
            Clock: in STD_LOGIC;          -- The ADC
Clock
            Enable: in std_logic_vector(1 downto 0);
-- The running flag (If high than the unit is
operational)

            --
            -- Hardware interface
            --
            ADC_Address: out STD_LOGIC_VECTOR(5 downto
0);          -- ADC address lines
            ADC_ChipSelect: out STD_LOGIC_VECTOR(1
downto 0);        -- ADC Chip Select
            ADC_Clock: out STD_LOGIC_VECTOR(1 downto
0);             -- ADC Clock (max: 32 MHz)
            ADC_Data: in STD_LOGIC_VECTOR(11 downto 0);
-- ADC Data lines

            -- The ADC data for each channel
            -- ADC 1A
            Channel_10_0: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_10_1: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_10_2: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_10_3: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_10_4: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_10_5: out STD_LOGIC_VECTOR(11
downto 0);
            -- ADC 1B
            Channel_11_0: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_11_1: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_11_2: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_11_3: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_11_4: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_11_5: out STD_LOGIC_VECTOR(11
downto 0);
            -- ADC 2A
            Channel_20_0: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_20_1: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_20_2: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_20_3: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_20_4: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_20_5: out STD_LOGIC_VECTOR(11
downto 0);
            -- ADC 2B
            Channel_21_0: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_21_1: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_21_2: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_21_3: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_21_4: out STD_LOGIC_VECTOR(11
downto 0);
            Channel_21_5: out STD_LOGIC_VECTOR(11
downto 0);
            -- ADC 3A
            Channel_30_0: out STD_LOGIC_VECTOR(11
downto 0);
```

```
        Channel_30_1: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_30_2: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_30_3: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_30_4: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_30_5: out STD_LOGIC_VECTOR(11
downto 0);
        -- ADC 3B
        Channel_31_0: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_31_1: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_31_2: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_31_3: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_31_4: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_31_5: out STD_LOGIC_VECTOR(11
downto 0);
        -- ADC 4A
        Channel_40_0: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_40_1: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_40_2: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_40_3: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_40_4: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_40_5: out STD_LOGIC_VECTOR(11
downto 0);
        -- ADC 4B
        Channel_41_0: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_41_1: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_41_2: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_41_3: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_41_4: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_41_5: out STD_LOGIC_VECTOR(11
downto 0);
        -- ADC 5A
        Channel_50_0: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_50_1: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_50_2: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_50_3: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_50_4: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_50_5: out STD_LOGIC_VECTOR(11
downto 0);
        -- ADC 5B
        Channel_51_0: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_51_1: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_51_2: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_51_3: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_51_4: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_51_5: out STD_LOGIC_VECTOR(11
downto 0);
        -- ADC 6A
        Channel_60_0: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_60_1: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_60_2: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_60_3: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_60_4: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_60_5: out STD_LOGIC_VECTOR(11
downto 0);
        -- ADC 6B
        Channel_61_0: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_61_1: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_61_2: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_61_3: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_61_4: out STD_LOGIC_VECTOR(11
downto 0);
        Channel_61_5: out STD_LOGIC_VECTOR(11
downto 0);

        --
        -- Voltage section
        --
        Voltage_U0: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_U1: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_U2: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_U3: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_U4: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_U5: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_U6: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_U7: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_U8: out STD_LOGIC_VECTOR(17 downto
0);

        Voltage_V0: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_V1: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_V2: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_V3: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_V4: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_V5: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_V6: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_V7: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_V8: out STD_LOGIC_VECTOR(17 downto
0);

        Voltage_W0: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_W1: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_W2: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_W3: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_W4: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_W5: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_W6: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_W7: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_W8: out STD_LOGIC_VECTOR(17 downto
0);

        Voltage_R: out STD_LOGIC_VECTOR(17 downto
0);
        Voltage_S: out STD_LOGIC_VECTOR(17 downto
0);
```

```
            Voltage_T: out STD_LOGIC_VECTOR(17 downto
0);

            --
            -- Current section
            --

            Current_R: out STD_LOGIC_VECTOR(17 downto
0);
            Current_S: out STD_LOGIC_VECTOR(17 downto
0);
            Current_T: out STD_LOGIC_VECTOR(17 downto
0);

            Current_Uhi: out STD_LOGIC_VECTOR(17 downto
0);
            Current_Ulo: out STD_LOGIC_VECTOR(17 downto
0);
            Current_Vhi: out STD_LOGIC_VECTOR(17 downto
0);
            Current_Vlo: out STD_LOGIC_VECTOR(17 downto
0);
            Current_Whi: out STD_LOGIC_VECTOR(17 downto
0);
            Current_Wlo: out STD_LOGIC_VECTOR(17 downto
0);


            --
            -- Gain section
            --

            Voltage_Gain: in STD_LOGIC_VECTOR(17 downto
0);
            Current_Gain: in STD_LOGIC_VECTOR(17 downto
0)
        );
    end component;
    --
    -- ADC
    --
    signal myADC_Enable: std_logic_vector(1 downto 0)
:= "11";
    --
    -- ADC Values
    --
    signal myADC1: std_logic_vector (143 downto 0) :=
x"000000000000000000000000000000000000";
    signal myADC2: std_logic_vector (143 downto 0) :=
x"000000000000000000000000000000000000";
    signal myADC3: std_logic_vector (143 downto 0) :=
x"000000000000000000000000000000000000";
    signal myADC4: std_logic_vector (143 downto 0) :=
x"000000000000000000000000000000000000";
    signal myADC5: std_logic_vector (143 downto 0) :=
x"000000000000000000000000000000000000";
    signal myADC6: std_logic_vector (143 downto 0) :=
x"000000000000000000000000000000000000";
    --
    -- Voltage section
    --
    signal myVoltagesU: std_logic_vector (143 downto 0)
:= (others => '0');
    signal myVoltagesV: std_logic_vector (143 downto 0)
:= (others => '0');
    signal myVoltagesW: std_logic_vector (143 downto 0)
:= (others => '0');

    signal myVoltage_R: STD_LOGIC_VECTOR(17 downto 0)
:= (others => '0');
    signal myVoltage_S: STD_LOGIC_VECTOR(17 downto 0)
:= (others => '0');
    signal myVoltage_T: STD_LOGIC_VECTOR(17 downto 0)
:= (others => '0');
    --
    --      Current section
    --
    signal myCurrent_R: STD_LOGIC_VECTOR(17 downto 0)
:= (others => '0');
    signal myCurrent_S: STD_LOGIC_VECTOR(17 downto 0)
:= (others => '0');
    signal myCurrent_T: STD_LOGIC_VECTOR(17 downto 0)
:= (others => '0');
    signal myCurrent_Uhi: STD_LOGIC_VECTOR(17 downto 0)
:= (others => '0');
    signal myCurrent_Ulo: STD_LOGIC_VECTOR(17 downto 0)
:= (others => '0');
    signal myCurrent_Vhi: STD_LOGIC_VECTOR(17 downto 0)
:= (others => '0');
    signal myCurrent_Vlo: STD_LOGIC_VECTOR(17 downto 0)
:= (others => '0');
    signal myCurrent_Whi: STD_LOGIC_VECTOR(17 downto 0)
:= (others => '0');
    signal myCurrent_Wlo: STD_LOGIC_VECTOR(17 downto 0)
:= (others => '0');
    --
    -- Delay view
    --
    signal myVoltages_Delay: std_logic_vector (287
downto 0) := (others => '0');



    ---------------------------------------------------
--------
    --
    --
    --
    --
    --
    --           Data display
    --
    --
    --
    --
    --

    component DataDisplay port (
            Clock: in  STD_LOGIC;

            Command : in STD_LOGIC_VECTOR(383 downto
0);

            -- System states
            SystemState: in STD_LOGIC_VECTOR(1 downto
0);
            SystemRunning: in STD_LOGIC;
            SystemOutput: in STD_LOGIC;

            -- Unit handler
            IGBTs: in STD_LOGIC_VECTOR(63 downto 0);
            OverTemp: in STD_LOGIC_VECTOR(15 downto 0);
            OverVolt: in STD_LOGIC_VECTOR(15 downto 0);
            OverCurrent: in STD_LOGIC_VECTOR(15 downto
0);
            FaultA: in STD_LOGIC_VECTOR(15 downto 0);
            FaultB: in STD_LOGIC_VECTOR(15 downto 0);
            Active: in STD_LOGIC_VECTOR(15 downto 0);

            -- Measurements
            Voltages: in std_logic_vector (287 downto
0);
            VDC: in std_logic_vector(17 downto 0);
            IDC: in std_logic_vector (17 downto 0);
            Ihi: in std_logic_vector (17 downto 0);
            Ilo: in std_logic_vector (17 downto 0);
            Iout: in std_logic_vector (17 downto 0);
            Fref: in std_logic_vector (17 downto 0);
            Fout: in std_logic_vector (17 downto 0);

            -- System constants
            VoltageGain: in std_logic_vector(17 downto
0);
            CurrentGain: in std_logic_vector(17 downto
0);
            VoltageCharge: in std_logic_vector(17
downto 0);
            VoltageCoeff_Inc: in std_logic_vector(17
downto 0);
            VoltageCoeff_Dec: in std_logic_vector(17
downto 0);
            ControlFreqRef: in std_logic_vector(11
downto 0);
```

```
            -- Hardware connection
            Comp_OE: in std_logic_vector(5 downto 0);
            Comp_GFault: in std_logic_vector(5 downto
0);
            Comp_Reset: in std_logic_vector(5 downto
0);

            Gate_Reset: in std_logic_vector(2 downto
0);
            Gate_GFault: in std_logic_vector(2 downto
0);
            Gate_OE: in std_logic_vector(2 downto 0);

            ADC_Enable: in std_logic_vector(1 downto
0);

            Gate_IGBT: in std_logic_vector(71 downto
0);
            Gate_Fault: in std_logic_vector(35 downto
0);
            Gate_OvTemp: in std_logic_vector(35 downto
0);

            Comparator_Under: in std_logic_vector(71
downto 0);
            Comparator_Over: in std_logic_vector(71
downto 0);

            -- Contactors
            K_ChargeIn: in STD_LOGIC;
            K_ChargeOut: in STD_LOGIC;
            K_SourceIn: in STD_LOGIC;
            K_SourceOut: in STD_LOGIC;
            K_LoadRezIn: in STD_LOGIC;
            K_LoadRezOut: in STD_LOGIC;
            K_LoadTransIn: in STD_LOGIC;
            K_LoadTransOut: in STD_LOGIC;

            -- ADCs
            ADC1: in std_logic_vector (143 downto 0);
            ADC2: in std_logic_vector (143 downto 0);
            ADC3: in std_logic_vector (143 downto 0);
            ADC4: in std_logic_vector (143 downto 0);
            ADC5: in std_logic_vector (143 downto 0);
            ADC6: in std_logic_vector (143 downto 0);

            --
            -- Screen writer
            --
            Data_Addr : out std_logic_vector(12 downto
0);
            Data_Type : out std_logic_vector(6 downto
0);       -- The data type (see SCreenWriter.vhd
header)
            Data_Bool : out std_logic;
            Data_Number: out std_logic_vector(47 downto
0);

            Data_Ack : in std_logic;
-- Master: Acknowledge a data write
            Data_DataReady: out std_logic;
-- Slave: specify that data is ready
            Data_Ready : in std_logic
-- Master: Signals when data can be placed in the
buffer
        );
    end component;

    ---------------------------------------------------
--------
    --
    --
    --
    --
    --
    --
    --
    --              User input
    --
    --
    --
    --
    --

    component UserInput Port (
        Clock : in std_logic;

        --
        -- Hardware connections
        --
        MISO : out  STD_LOGIC;
        MOSI : in  STD_LOGIC;
        PS2Clock : in  STD_LOGIC;
        ChipSelect : in  STD_LOGIC;

        --
        -- Software
        --
        Key: out std_logic_vector(7 downto 0);
        Ascii: out std_logic_vector(7 downto 0);
        AsciiDone: out std_logic;

        InputData: out std_logic
    );
    end component;
    signal myMISO: std_logic := '0';
    --
    -- User input
    --
    signal myKey: std_logic_vector(7 downto 0) :=
"00000000";
    signal myAscii: std_logic_vector(7 downto 0) :=
"00000000";
    signal myAsciiDone: std_logic := '0';
    signal myInputData : std_logic := '0';
    signal myCommand: std_logic_vector(383 downto 0) :=
(others => '0');
    signal myCommandSize: std_logic_vector(5 downto 0)
:= "000000";



    ---------------------------------------------------
--------
    --
    --
    --
    --
    --
    --
    --
    --              User handler
    --
    --
    --
    --
    --

    component UserHandler Port (
        Clock : in STD_LOGIC;

        --
        -- Keyboard input
        --
        Key_ScanCodeId: in STD_LOGIC_VECTOR(7 downto
0);
        Key_Asci: in STD_LOGIC_VECTOR(7 downto 0);
        Key_AsciiDone: in std_logic;
        Key_InputData: in STD_LOGIC;

        --
        -- Output system parameters
        --
        -- Video output
        ScreenId: out STD_LOGIC_VECTOR(1 downto 0);
        ScrollPos: out STD_LOGIC_VECTOR(7 downto 0);

        -- System states
        SystemState: out STD_LOGIC_VECTOR(1 downto 0);
        SystemRunning: out STD_LOGIC;
        SystemOutput: out STD_LOGIC;
        SystemReset: out STD_LOGIC;

        -- Command output
        Command : out STD_LOGIC_VECTOR(383 downto 0);
        CommandSize: out std_logic_vector(5 downto 0);
        VoltageCoeff_Inc: out std_logic_vector(17
downto 0);
        VoltageCoeff_Dec: out std_logic_vector(17
downto 0);

        ControlReference: out std_logic_vector(11
downto 0);
        ControlReferenceIn : in std_logic_vector(11
downto 0);
        ControlReferenceChange : out std_logic;

        VoltageGain: out std_logic_vector(17 downto 0);
        CurrentGain: out std_logic_vector(17 downto 0);
        VoltageCharge: out std_logic_vector(17 downto
0)
    );
    end component;



    ---------------------------------------------------
--------
    --
    --
    --
    --
    --
    --
    --
    --              VGA System
    --
    --
    --
    --
    --

    component VGA
        port (
            --
            -- Hardware interface
            --
            sys_clock : in  STD_LOGIC;
            vga_clock : in  STD_LOGIC;
            ClockScreenWritter : in STD_LOGIC;
            reset: in STD_LOGIC;

            Color_R : out  STD_LOGIC_VECTOR (3 downto
0);
            Color_G : out  STD_LOGIC_VECTOR (3 downto
0);
            Color_B : out  STD_LOGIC_VECTOR (3 downto
0);
            Vsync : out  STD_LOGIC;
            Hsync : out  STD_LOGIC;

            ScreenId: in STD_LOGIC_VECTOR(1 downto 0);
            ScrollPos: in STD_LOGIC_VECTOR(7 downto 0);


            CursorX: in STD_LOGIC_VECTOR(6 downto 0);
            CursorY: in STD_LOGIC_VECTOR(6 downto 0);

            --
            -- Screen writer
            --
            Data_Addr : in std_logic_vector(12 downto
0);
            Data_Type : in std_logic_vector(6 downto
0);         -- The data type (see SCreenWriter.vhd
header)
            Data_Bool : in std_logic;
            Data_Number: in std_logic_vector(47 downto
0);

            Data_Ack : out std_logic;
-- Master: Acknowledge a data write
            Data_DataReady: in std_logic;
-- Slave: specify that data is ready
            Data_Ready : out std_logic
-- Master: Signals when data can be placed in the
buffer
        );
    end component;
```

```
    signal myScreenId: std_logic_vector(1 downto 0) :=
"00";
    signal myScrollPos: std_logic_vector(7 downto 0) :=
"00000000";
    --
    -- Screen variables
    --
    signal CursorX: STD_LOGIC_VECTOR(6 downto 0) :=
"0000000";
    signal CursorY: STD_LOGIC_VECTOR(6 downto 0) :=
"0000000";



    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  State machine variables
    --
    --
    --
    --

--  type TStateType is (State_DoControl,
State_DoUnitConversion, State_DoMapping,
State_DoTransmit);
--  signal myCurrentState, myNextState : TStateType;


    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Main system variables
    --
    --
    --
    --

    signal reset: std_logic := '0';



    signal myData_DataReady: std_logic := '0';
-- Slave: specify that data is ready
    signal myData_Addr : std_logic_vector(12 downto 0)
:= "0000000000000";
    signal myData_Type : std_logic_vector(6 downto 0)
:= "0000000";     -- The data type (see
SCreenWriter.vhd header)
    signal myData_Bool : std_logic := '0';
    signal myData_Ack : std_logic := '0';
    signal myData_Ready : std_logic := '0';
    signal myData_Number: std_logic_vector(47 downto 0)
:= (others => '0');


    --
    --
    -- System parameters
    --
    --
    signal mySystemState: std_logic_vector(1 downto 0)
:= "00";
    signal mySystemRunning: std_logic := '0';
    signal mySystemOutput: std_logic := '0';
    signal mySystemReset: std_logic := '0';
    signal mySystemFault: std_logic := '0';
    signal mySystemChargeComplete: std_logic := '0';

    signal myFref: std_logic_vector (17 downto 0) :=
"000000000000000000";
    signal myFout: std_logic_vector (17 downto 0) :=
"000000000000000000";
```

```
    signal myLedsOutput: std_logic_vector(7 downto 0)
:= (others => '0');

begin

    ---------------------------------------------------
---------------------------------
    --
    --
    --
    --
    --
    --
    --              System parameters
    --
    --
    --
    --
    --
    ---------------------------------------------------
---------------------------------

    reset <= not not_reset;

    ---------------------------------------------------
---------------------------------
    --
    --
    --
    --
    --
    --
    --              Create a system clock object to
generate various clocks
    --
    --
    --
    --
    --
    ---------------------------------------------------
---------------------------------

    SystemClocks: Clocks port map(
        Clock125 => sys_clock,
        Clock25 => vga_clock,
        Clock62 => myClock_62_Mhz,
        Clock31 => myClock_31_Mhz,
        Clock15 => myClock_15_Mhz,
        Clock8 => myClock_8_Mhz,
        Clock12 => myClock_12_Mhz,
        Clock25Hz => myClock_25_Hz,
        Clock2Hz => myClock_2Hz,
        ControlClock => myControlClock,
        ControlReference => myControlReference
    );

    ---------------------------------------------------
---------------------------------
    --
    --
    --
    --
    --
    --
    --              Gates
    --
    --
    --
    --
    --
    ---------------------------------------------------
---------------------------------

    GateSet_Block: GateSet port map(
        --
        -- Hardware Connections
        --
        Gates_Clock => Gates_Clock,
        Gates_ChipSelect => Gates_ChipSelect,
        Gates_OutputEnable => myGate_OE,
        Gates_MOSI =>  Gates_MOSI,
        Gates_MOSI2 =>  Gates_MOSI2,
        Gates_MISO => Gates_MISO,
        Gates_Fault => myGate_GFault,
        Gates_Reset => myGate_Reset,

        --
        -- Software Connections
        --
        Clock => myClock_15_Mhz,
        Running => mySystemRunning,

        UnitLevels => myUnitLevelOut,
        UnitStates => myUnitStatesOut,

--      UnitLevels(59 downto 8) => myUnitLevelOut(59
downto 8), --
"0000000000000000000000000000000000000000000000000000",
--      UnitLevels(7 downto 0) => mySwitches,
--      UnitStates(29 downto 4) => myUnitStatesOut(29
downto 4), -- "00000000000000000000000000",
--      UnitStates(3 downto 0) => myButtons,

        Faults => myUnitFaults,
        OverTemp => myUnitOverTemp
    );

    ---------------------------------------------------
---------------------------------
    --
    --
    --
    --
    --
    --
    --              VGA System:  Create a VGA Unit in
order to create an VGA output
    --
    --
    --
    --
    --
    ---------------------------------------------------
---------------------------------

    VGA_System: VGA port map(
        sys_clock    => sys_clock,
        vga_clock    => vga_clock,
        ClockScreenWritter => myClock_8_Mhz,
        reset    => reset,

        Color_R => Video_Red,
        Color_G => Video_Green,
        Color_B => Video_Blue,
        Vsync => Video_Vsync,
        Hsync => Video_Hsync,

        ScreenId => myScreenId, -- "00", -- myScreenId,
        ScrollPos => myScrollPos,

        CursorX => CursorX,
        CursorY => CursorY,

        --
        -- Screen writer
        --
        Data_Addr => myData_Addr,
        Data_Type => myData_Type,
        Data_Bool => myData_Bool,
        Data_Number  => myData_Number,

        Data_Ack   => myData_Ack,
        Data_DataReady => myData_DataReady,
        Data_Ready   => myData_Ready
    );
    CursorX <= "0001010" + myCommandSize;
    CursorY <= "0011100";



    ---------------------------------------------------
--------------------------------
    --
    --
    --
    --
    --
    --
    --            Display Unit is used to display the
system parameters on a VGA screen
    --
    --
    --
    --
    --
    ---------------------------------------------------
--------------------------------


    Data_Display: DataDisplay port map(
        Clock => myClock_15_Mhz,

        Command => myCommand,

        SystemState => mySystemState,
        SystemRunning => mySystemRunning,
        SystemOutput => mySystemOutput,
        --SystemReset => mySystemReset,

        VoltageGain => myVoltageSensors_Gain,
        CurrentGain => myCurrentSensors_Gain,
        VoltageCharge => myVoltage_ChargeLevel,
        VoltageCoeff_Inc => (others => '0'), --
myVoltageCoeff_Inc,
        VoltageCoeff_Dec => (others => '0'), --
myVoltageCoeff_Dec,
        ControlFreqRef => myControlReference,

        IGBTs => (others => '0'), --myIGBTs,
        OverTemp => (others => '0'), --myOverTemp,
        OverVolt => (others => '0'), --myOverVolt,
        OverCurrent => (others => '0'), --
myOverCurrent,
        FaultA => (others => '0'), --myFaultA,
        FaultB => (others => '0'), --myFaultB,
        Active => (others => '0'), --myActive,

        -- Hardware data
        Comp_OE => (others => '0'), --myComp_OE,
        Comp_GFault => (others => '0'), --
myComp_GFault,
        Comp_Reset => (others => '0'), --
myComp_SysReset,

        Gate_Reset => myGate_SysReset(2 downto 0),
        Gate_GFault => myGate_GFault(2 downto 0),
        Gate_OE => myGate_SysOE(2 downto 0),

        ADC_Enable => myADC_Enable,

        Gate_IGBT => myGate_IGBT(71 downto 0), --
myIGBTs, --
        Gate_Fault =>  myUnitFaults(35 downto 0),
        Gate_OvTemp => myUnitOverTemp(35 downto 0),

        Comparator_Under => (others => '0'), --
myComp_Under,
        Comparator_Over => (others => '0'), --
myComp_Over,

        K_ChargeIn => '0', --myKChargeIn,
        K_ChargeOut => '0', --myKChargeOut,
        K_SourceIn => '0', --myKSourceIn,
        K_SourceOut => '0', --myKSourceOut,
        K_LoadRezIn => '0', --myKLoadRezIn,
        K_LoadRezOut => '0', --myKLoadRezOut,
        K_LoadTransIn => '0', --myKLoadTransIn,
        K_LoadTransOut => '0', --myKLoadTransOut,

        -- ADCs
        ADC1 => myADC1,
        ADC2 => myADC2,
        ADC3 => myADC3,
        ADC4 => myADC4,
        ADC5 => myADC5,
        ADC6 => myADC6,

        Voltages => (others => '0'), -- myVoltages,
        VDC => (others => '0'), --myVDC,
        IDC => (others => '0'), --myIDC,
        Ihi => (others => '0'), --myIhi,
        Ilo => (others => '0'), --myIlo,
        Iout => (others => '0'), --myIout,
        Fref => (others => '0'), --myFref,
        Fout => (others => '0'), --myFout,

        Data_Addr => myData_Addr,
        Data_Type => myData_Type,
        Data_Bool => myData_Bool,
        Data_Number => myData_Number,

        Data_Ack => myData_Ack,
        Data_DataReady => myData_DataReady,
        Data_Ready => myData_Ready
    );



    ---------------------------------------------------
--------------------------------
    --
    --
    --
    --
    --
    --
    --            ADCs
    --
    --
    --
    --
    --
    ---------------------------------------------------
--------------------------------


    ADC_Block: ADCs port map(
            Clock => myClock_15_Mhz,            --
The ADC Clock (max 32 MHz)
            Enable => myADC_Enable,

            --
            -- Hardware interface
            --
            ADC_Address => ADC_Address,         --
ADC address lines
            ADC_ChipSelect => ADC_ChipSelect,     --
ADC Chip Select
            ADC_Clock => ADC_Clock,
-- ADC Clock (max: 32 MHz)
            ADC_Data => ADC_Data,
-- ADC Data lines

            -- The ADC data for each channel
            -- ADC 1A
            Channel_10_0 => myADC1( 11 downto  0),
Channel_10_1 => myADC1( 23 downto 12),  Channel_10_2 =>
myADC1( 35 downto 24),
            Channel_10_3 => myADC1( 47 downto 36),
Channel_10_4 => myADC1( 59 downto 48),
Channel_10_5 => myADC1( 71 downto 60),
            -- ADC 1B
            Channel_11_0 => myADC1( 83 downto 72),
Channel_11_1 => myADC1( 95 downto 84),
            Channel_11_2 => myADC1(107 downto 96),
Channel_11_3 => myADC1(119 downto 108),
            Channel_11_4 => myADC1(131 downto 120),
Channel_11_5 => myADC1(143 downto 132),
            -- ADC 2A
            Channel_20_0 => myADC2( 11 downto  0),
Channel_20_1 => myADC2( 23 downto 12),
            Channel_20_2 => myADC2( 35 downto 24),
Channel_20_3 => myADC2( 47 downto 36),
            Channel_20_4 => myADC2( 59 downto 48),
Channel_20_5 => myADC2( 71 downto 60),
            -- ADC 2B
            Channel_21_0 => myADC2( 83 downto 72),
Channel_21_1 => myADC2( 95 downto 84),
            Channel_21_2 => myADC2(107 downto 96),
Channel_21_3 => myADC2(119 downto 108),
            Channel_21_4 => myADC2(131 downto 120),
Channel_21_5 => myADC2(143 downto 132),
            -- ADC 3A
            Channel_30_0 => myADC3( 11 downto  0),
Channel_30_1 => myADC3( 23 downto 12),
            Channel_30_2 => myADC3( 35 downto 24),
Channel_30_3 => myADC3( 47 downto 36),
            Channel_30_4 => myADC3( 59 downto 48),
Channel_30_5 => myADC3( 71 downto 60),
            -- ADC 3B
            Channel_31_0 => myADC3( 83 downto 72),
Channel_31_1 => myADC3( 95 downto 84),
            Channel_31_2 => myADC3(107 downto 96),
Channel_31_3 => myADC3(119 downto 108),
            Channel_31_4 => myADC3(131 downto 120),
Channel_31_5 => myADC3(143 downto 132),
            -- ADC 4A
            Channel_40_0 => myADC4( 11 downto  0),
Channel_40_1 => myADC4( 23 downto 12),
            Channel_40_2 => myADC4( 35 downto 24),
Channel_40_3 => myADC4( 47 downto 36),
            Channel_40_4 => myADC4( 59 downto 48),
Channel_40_5 => myADC4( 71 downto 60),
            -- ADC 4B
            Channel_41_0 => myADC4( 83 downto 72),
Channel_41_1 => myADC4( 95 downto 84),
            Channel_41_2 => myADC4(107 downto 96),
Channel_41_3 => myADC4(119 downto 108),
            Channel_41_4 => myADC4(131 downto 120),
Channel_41_5 => myADC4(143 downto 132),
            -- ADC 5A
            Channel_50_0 => myADC5( 11 downto  0),
Channel_50_1 => myADC5( 23 downto 12),
            Channel_50_2 => myADC5( 35 downto 24),
Channel_50_3 => myADC5( 47 downto 36),
            Channel_50_4 => myADC5( 59 downto 48),
Channel_50_5 => myADC5( 71 downto 60),
            -- ADC 5B
            Channel_51_0 => myADC5( 83 downto 72),
Channel_51_1 => myADC5( 95 downto 84),
            Channel_51_2 => myADC5(107 downto 96),
Channel_51_3 => myADC5(119 downto 108),
            Channel_51_4 => myADC5(131 downto 120),
Channel_51_5 => myADC5(143 downto 132),
            -- ADC 6A
            Channel_60_0 => myADC6( 11 downto  0),
Channel_60_1 => myADC6( 23 downto 12),
Channel_60_2 => myADC6( 35 downto 24),
            Channel_60_3 => myADC6( 47 downto 36),
Channel_60_4 => myADC6( 59 downto 48),
Channel_60_5 => myADC6( 71 downto 60),
            -- ADC 6B
            Channel_61_0 => myADC6( 83 downto 72),
Channel_61_1 => myADC6( 95 downto 84),
Channel_61_2 => myADC6(107 downto 96),
            Channel_61_3 => myADC6(119 downto 108),
Channel_61_4 => myADC6(131 downto 120),
Channel_61_5 => myADC6(143 downto 132),


            Voltage_U1 => myVoltagesU(17 downto 0),
            Voltage_U2 => myVoltagesU(35 downto 18),
            Voltage_U3 => myVoltagesU(53 downto 36),
            Voltage_U4 => myVoltagesU(71 downto 54),
            Voltage_U5 => myVoltagesU(89 downto 72),
            Voltage_U6 => myVoltagesU(107 downto 90),
            Voltage_U7 => myVoltagesU(125 downto 108),
            Voltage_U8 => myVoltagesU(143 downto 126),
```

```
            Voltage_V1 => myVoltagesV(17 downto 0),
            Voltage_V2 => myVoltagesV(35 downto 18),
            Voltage_V3 => myVoltagesV(53 downto 36),
            Voltage_V4 => myVoltagesV(71 downto 54),
            Voltage_V5 => myVoltagesV(89 downto 72),
            Voltage_V6 => myVoltagesV(107 downto 90),
            Voltage_V7 => myVoltagesV(125 downto 108),
            Voltage_V8 => myVoltagesV(143 downto 126),

            Voltage_W1 => myVoltagesW(17 downto 0),
            Voltage_W2 => myVoltagesW(35 downto 18),
            Voltage_W3 => myVoltagesW(53 downto 36),
            Voltage_W4 => myVoltagesW(71 downto 54),
            Voltage_W5 => myVoltagesW(89 downto 72),
            Voltage_W6 => myVoltagesW(107 downto 90),
            Voltage_W7 => myVoltagesW(125 downto 108),
            Voltage_W8 => myVoltagesW(143 downto 126),

            Voltage_R => myVoltage_R,
            Voltage_S => myVoltage_S,
            Voltage_T => myVoltage_T,


            --
            -- Current section
            --

            Current_R => myCurrent_R,
            Current_S => myCurrent_S,
            Current_T => myCurrent_T,

            Current_Uhi => myCurrent_Uhi,
            Current_Ulo => myCurrent_Ulo,
            Current_Vhi => myCurrent_Vhi,
            Current_Vlo => myCurrent_Vlo,
            Current_Whi => myCurrent_Whi,
            Current_Wlo => myCurrent_Wlo,

            --
            -- gAIN SECTION
            --

            Voltage_Gain => myVoltageSensors_Gain,
            Current_Gain => myCurrentSensors_Gain
    );


    ---------------------------------------------------
---------------------------------
    --
    --
    --
    --
    --
    --
    --            User handler
    --
    --
    --
    --
    --
    ---------------------------------------------------
---------------------------------
    User_Handler: UserHandler port map(
        Clock => myClock_31_Mhz,

        Key_ScanCodeId => myKey,
        Key_Asci => myAscii,
        Key_AsciiDone => myAsciiDone,
        Key_InputData => myInputData,

        ScreenId => myScreenId,
        ScrollPos => myScrollPos,

        SystemState => mySystemState,
        SystemRunning => mySystemRunning,
        SystemOutput  => mySystemOutput,
        SystemReset => mySystemReset,

        Command  => myCommand,
        CommandSize => myCommandSize,
        VoltageCoeff_Inc => myVoltageCoeff_Inc,
        VoltageCoeff_Dec => myVoltageCoeff_Dec,

        ControlReference => myUserControlReference,
        ControlReferenceIn => myControlReference,
        ControlReferenceChange =>
myUserControlReferenceChange,

        VoltageGain => myVoltageSensors_Gain,
        CurrentGain => myCurrentSensors_Gain,
        VoltageCharge => myVoltage_ChargeLevel
    );


    ---------------------------------------------------
------------------------------
    --
    --
    --
    --
    --
    --
    --            Create an interface for user input
    --
    --
    --
    --
    --
    ---------------------------------------------------
------------------------------
    User_Input: UserInput port map(
        Clock => myClock_15_Mhz,
        --
        -- Hardware connections
        --
        MISO => myMISO,
        MOSI => PS2_MOSI,
        PS2Clock => PS2_Clock,
        ChipSelect => PS2_ChipSelect,

        --
        -- Software
        --
        Key => myKey,
        Ascii => myAscii,
        AsciiDone => myAsciiDone,

        InputData => myInputData
    );



    ---------------------------------------------------
------------------------------
    --
    --
    --
    --
    --
    --
    --            Control
    --
    --
    --
    --
    --
    ---------------------------------------------------
------------------------------
    Control: MainControl Port map (
        Clock => myClock_62_Mhz,                -- Main
system clock
        Enable => mySystemRunning,              --
Enable the conversion (if disable, set to 0 all
outputs)

        Run => myModulationRun,                 -- If
enable, it converts the data input, if not, output the
last data

        Done => myModulationDone,               -- High
when the conversion is done

        -- Input data
        MethodSelection => myModulationMethod,  --
Method selection

        -- Input data
        ReferenceU => myReferenceU,
        ReferenceV => myReferenceV,
        ReferenceW => myReferenceW,

        ReferenceCounterMax => myModulationCounterMax,
-- The maximum value for the counters

        -- Measured values
        UnitVotlagesU => myVoltagesU,
        UnitVotlagesV => myVoltagesV,
        UnitVotlagesW => myVoltagesW,

        -- Current input values
        CurrentOutputU => datCurrentU(17 downto 0),
        CurrentOutputV => datCurrentV(17 downto 0),
        CurrentOutputW => datCurrentW(17 downto 0),

        -- Voltage level definition
        VoltageLevels(17 downto 0) => datVoltageL0,
        VoltageLevels(35 downto 18) => datVoltageL1,
        VoltageLevels(53 downto 36) => datVoltageL2,
        VoltageLevels(71 downto 54) => datVoltageL3,

        -- Output data
        UnitStatesU => myUnitStatesU,
        UnitStatesV => myUnitStatesV,
        UnitStatesW => myUnitStatesW,


        -- Output data
        UnitStateOut => myUnitStatesOut,        --
States for 8 * 3 units
        UnitLevelOut => myUnitLevelOut          --
Levels for 8 * 3 * 2 legs
    );




    ---------------------------------------------------
--------------------
    --
    --
    --
    --
    --
    --
    --         Control the main reference for the clock
of the control algorithm
    --
    --
    --
    --
    --
    --
    ---------------------------------------------------
--------------------

    P_ControlMainReference: process
(DspSetReferenceClock, datReference,
myUserControlReferenceChange, myUserControlReference)
    begin
        if (DspSetReferenceClock = '0') and
(myUserControlReferenceChange = '1') then
            myControlReference <=
myUserControlReference;
        elsif (DspSetReferenceClock = '1') and
(myUserControlReferenceChange = '0') then
            myControlReference <= datReference;
        else
            null;
        end if;
    end process P_ControlMainReference;
    ---------------------------------------------------
---------------------------------
    --
    --
    --
    --
    --
    --
    --             DSP Signals
    --
    --
    --
    --
    --
    ---------------------------------------------------
---------------------------------

    P_DSP_Process: process (DSP_Clock, DSP_CS,
DSP_Input, DSP_Addr)
    variable myTempData: std_logic_vector(15 downto 0);
    variable myTempDataOut: std_logic_vector(15 downto
0);
    begin
        if (DSP_CS = '0') then
            if (DSP_Clock'event and DSP_Clock = '1')
then
                if (DSP_RD = '0') then
                    if (DSP_Addr(0) = '0') then
                        case (DSP_Addr(9 downto 1)) is
                            when "000000000" =>
DSP_Output <= datControlU(15 downto 0); myTempDataOut
:= "0000000" & datControlU(24 downto 16);
                            when "000000001" =>
DSP_Output <= datControlV(15 downto 0); myTempDataOut
:= "0000000" & datControlV(24 downto 16);
                            when "000000010" =>
DSP_Output <= datControlW(15 downto 0); myTempDataOut
:= "0000000" & datControlW(24 downto 16);
                            when "000000011" =>
DSP_Output <= datInvVDC(15 downto 0); myTempDataOut :=
"0000000" & datInvVDC(24 downto 16);

                            when "000000100" =>
DSP_Output <= datVoltageR(15 downto 0); myTempDataOut
:= "00000001001000" & datVoltageR(17 downto 16);
                            when "000000101" =>
DSP_Output <= datVoltageS(15 downto 0); myTempDataOut
:= "00000001001000" & datVoltageS(17 downto 16);
                            when "000000110" =>
DSP_Output <= datVoltageT(15 downto 0); myTempDataOut
:= "00000001001000" & datVoltageT(17 downto 16);
                            when "000000111" =>
DSP_Output <= datVoltageDC(15 downto 0); myTempDataOut
:= "00000001001000" & datVoltageDC(17 downto 16);
                            when "000001000" =>
DSP_Output <= datVoltageU(15 downto 0); myTempDataOut
:= "00000001001000" & datVoltageU(17 downto 16);
                            when "000001001" =>
DSP_Output <= datVoltageV(15 downto 0); myTempDataOut
:= "00000001001000" & datVoltageV(17 downto 16);
                            when "000001010" =>
DSP_Output <= datVoltageW(15 downto 0); myTempDataOut
:= "00000001001000" & datVoltageW(17 downto 16);

                            when "000001011" =>
DSP_Output <= myVoltagesU( 15 downto   0);
myTempDataOut := "00000001001000" & myVoltagesU( 17
downto  16);
                            when "000001100" =>
DSP_Output <= myVoltagesU( 33 downto  18);
myTempDataOut := "00000001001000" & myVoltagesU( 35
downto  34);
                            when "000001101" =>
DSP_Output <= myVoltagesU( 51 downto  36);
myTempDataOut := "00000001001000" & myVoltagesU( 53
downto  52);
                            when "000001110" =>
DSP_Output <= myVoltagesU( 69 downto  54);
myTempDataOut := "00000001001000" & myVoltagesU( 71
downto  70);
                            when "000001111" =>
DSP_Output <= myVoltagesU( 87 downto  72);
myTempDataOut := "00000001001000" & myVoltagesU( 89
downto  88);
                            when "000010000" =>
DSP_Output <= myVoltagesU(105 downto  90);
myTempDataOut := "00000001001000" & myVoltagesU(107
downto 106);
                            when "000010001" =>
DSP_Output <= myVoltagesU(123 downto 108);
myTempDataOut := "00000001001000" & myVoltagesU(125
downto 124);
                            when "000010010" =>
DSP_Output <= myVoltagesU(141 downto 126);
myTempDataOut := "00000001001000" & myVoltagesU(143
downto 142);

                            when "000010011" =>
DSP_Output <= myVoltagesV( 15 downto   0);
myTempDataOut := "00000001001000" & myVoltagesV( 17
downto  16);
                            when "000010100" =>
DSP_Output <= myVoltagesV( 33 downto  18);
myTempDataOut := "00000001001000" & myVoltagesV( 35
downto  34);
                            when "000010101" =>
DSP_Output <= myVoltagesV( 51 downto  36);
myTempDataOut := "00000001001000" & myVoltagesV( 53
downto  52);
                            when "000010110" =>
DSP_Output <= myVoltagesV( 69 downto  54);
myTempDataOut := "00000001001000" & myVoltagesV( 71
downto  70);
                            when "000010111" =>
DSP_Output <= myVoltagesV( 87 downto  72);
myTempDataOut := "00000001001000" & myVoltagesV( 89
downto  88);
                            when "000011000" =>
DSP_Output <= myVoltagesV(105 downto  90);
myTempDataOut := "00000001001000" & myVoltagesV(107
downto 106);
                            when "000011001" =>
DSP_Output <= myVoltagesV(123 downto 108);
myTempDataOut := "00000001001000" & myVoltagesV(125
downto 124);
                            when "000011010" =>
DSP_Output <= myVoltagesV(141 downto 126);
myTempDataOut := "00000001001000" & myVoltagesV(143
downto 142);

                            when "000011011" =>
DSP_Output <= myVoltagesW( 15 downto   0);
myTempDataOut := "00000001001000" & myVoltagesW( 17
downto  16);
                            when "000011100" =>
DSP_Output <= myVoltagesW( 33 downto  18);
myTempDataOut := "00000001001000" & myVoltagesW( 35
downto  34);
                            when "000011101" =>
DSP_Output <= myVoltagesW( 51 downto  36);
myTempDataOut := "00000001001000" & myVoltagesW( 53
downto  52);
                            when "000011110" =>
DSP_Output <= myVoltagesW( 69 downto  54);
myTempDataOut := "00000001001000" & myVoltagesW( 71
downto  70);
                            when "000011111" =>
DSP_Output <= myVoltagesW( 87 downto  72);
myTempDataOut := "00000001001000" & myVoltagesW( 89
downto  88);
                            when "000100000" =>
DSP_Output <= myVoltagesW(105 downto  90);
myTempDataOut := "00000001001000" & myVoltagesW(107
downto 106);
                            when "000100001" =>
DSP_Output <= myVoltagesW(123 downto 108);
myTempDataOut := "00000001001000" & myVoltagesW(125
downto 124);
                            when "000100010" =>
DSP_Output <= myVoltagesW(141 downto 126);
myTempDataOut := "00000001001000" & myVoltagesW(143
downto 142);
```

```
                            when "000100011" =>
DSP_Output <= datCurrentR(15 downto 0); myTempDataOut
:= "0000000" & datCurrentR(24 downto 16);
                            when "000100100" =>
DSP_Output <= datCurrentS(15 downto 0); myTempDataOut
:= "0000000" & datCurrentS(24 downto 16);
                            when "000100101" =>
DSP_Output <= datCurrentT(15 downto 0); myTempDataOut
:= "0000000" & datCurrentT(24 downto 16);
                            when "000100110" =>
DSP_Output <= datCurrentDC(15 downto 0); myTempDataOut
:= "0000000" & datCurrentDC(24 downto 16);
                            when "000100111" =>
DSP_Output <= datCurrentU(15 downto 0); myTempDataOut
:= "0000000" & datCurrentU(24 downto 16);
                            when "000101000" =>
DSP_Output <= datCurrentV(15 downto 0); myTempDataOut
:= "0000000" & datCurrentV(24 downto 16);
                            when "000101001" =>
DSP_Output <= datCurrentW(15 downto 0); myTempDataOut
:= "0000000" & datCurrentW(24 downto 16);

                            when "000101010" =>
DSP_Output <= datCurrentUhi(15 downto 0); myTempDataOut
:= "0000000" & datCurrentUhi(24 downto 16);
                            when "000101011" =>
DSP_Output <= datCurrentUlo(15 downto 0); myTempDataOut
:= "0000000" & datCurrentUlo(24 downto 16);
                            when "000101100" =>
DSP_Output <= datCurrentVhi(15 downto 0); myTempDataOut
:= "0000000" & datCurrentVhi(24 downto 16);
                            when "000101101" =>
DSP_Output <= datCurrentVlo(15 downto 0); myTempDataOut
:= "0000000" & datCurrentVlo(24 downto 16);
                            when "000101110" =>
DSP_Output <= datCurrentWhi(15 downto 0); myTempDataOut
:= "0000000" & datCurrentWhi(24 downto 16);
                            when "000101111" =>
DSP_Output <= datCurrentWlo(15 downto 0); myTempDataOut
:= "0000000" & datCurrentWlo(24 downto 16);

                            when "000110000" =>
DSP_Output <= datFaultsU(15 downto 0); myTempDataOut :=
datFaultsU(31 downto 16);
                            when "000110001" =>
DSP_Output <= datFaultsV(15 downto 0); myTempDataOut :=
datFaultsV(31 downto 16);
                            when "000110010" =>
DSP_Output <= datFaultsW(15 downto 0); myTempDataOut :=
datFaultsW(31 downto 16);
                            when "000110011" =>
DSP_Output <= datFaults(15 downto 0); myTempDataOut :=
datFaults(31 downto 16);

                            when "000110100" =>
DSP_Output <= datTriggers(15 downto 0); myTempDataOut
:= datTriggers(31 downto 16);
                            when "000110101" =>
DSP_Output <= datFlags(15 downto 0); myTempDataOut :=
datFlags(31 downto 16);
                            when "000110110" =>
DSP_Output <= datStatus(15 downto 0); myTempDataOut :=
datStatus(31 downto 16);

                            when "000110111" =>
DSP_Output <= "0000" & myControlReference;
myTempDataOut := (others => '0');

                            when "010000000" =>
DSP_Output <= "00000000" & mySwitches; myTempDataOut :=
(others => '0');
                            when "010000001" =>
DSP_Output <= "000000000000" & myButtons; myTempDataOut
:= (others => '0');

                            when "010000100" =>
DSP_Output <= "00000000" & myLedsOutput;

                            when others =>
                                myTempDataOut :=
(others => '0');
                                DSP_Output <= (others
=> '0');
                        end case;
```

```
                    else
                        DSP_Output <= myTempDataOut;
                    end if;
                else
                    if (DSP_WR = '0') then  -- DO: DSP
WRITE, FPGA READ
                        if (DSP_Addr(0) = '0') then
                            -- Store the LSB
                            myTempData := DSP_Input;
                        else
                            case (DSP_Addr(9 downto 1))
is
                            when "000000000" =>
datControlU <= DSP_Input(8 downto 0) & myTempData;
                            when "000000001" =>
datControlV <= DSP_Input(8 downto 0) & myTempData;
                            when "000000010" =>
datControlW <= DSP_Input(8 downto 0) & myTempData;
                            when "000000011" =>
datInvVDC <= DSP_Input(8 downto 0) & myTempData;

                            when "000000111" =>
datVoltageDC <= DSP_Input(8 downto 0) & myTempData;
                            when "000001000" =>
datVoltageU <= DSP_Input(8 downto 0) & myTempData;
                            when "000001001" =>
datVoltageV <= DSP_Input(8 downto 0) & myTempData;
                            when "000001010" =>
datVoltageW <= DSP_Input(8 downto 0) & myTempData;

                            when "000100110" =>
datCurrentDC<= DSP_Input(8 downto 0) & myTempData;
                            when "000100111" =>
datCurrentU <= DSP_Input(8 downto 0) & myTempData;
                            when "000101000" =>
datCurrentV <= DSP_Input(8 downto 0) & myTempData;
                            when "000101001" =>
datCurrentW <= DSP_Input(8 downto 0) & myTempData;

                            when "000110100" =>
datTriggers <= DSP_Input & myTempData;


                            when "000110111" =>
datReference <= myTempData(11 downto 0);
DspSetReferenceClock <= '1';
                            when "000111000" =>
myModulationRun <= myTempData(0);
                            when "000111001" =>
myModulationMethod <= myTempData(2 downto 0);
                            when "000111010" =>
myModulationCounterMax <= DSP_Input(7 downto 0) &
myTempData;

                            when "001000000" =>
datVoltageL0 <= DSP_Input(1 downto 0) & myTempData;
                            when "001000001" =>
datVoltageL1 <= DSP_Input(1 downto 0) & myTempData;
                            when "001000010" =>
datVoltageL2 <= DSP_Input(1 downto 0) & myTempData;
                            when "001000011" =>
datVoltageL3 <= DSP_Input(1 downto 0) & myTempData;
                            when "001000100" =>
datVoltageL4 <= DSP_Input(1 downto 0) & myTempData;
                            when "001000101" =>
datVoltageL5 <= DSP_Input(1 downto 0) & myTempData;
                            when "001000110" =>
datVoltageL6 <= DSP_Input(1 downto 0) & myTempData;
                            when "001000111" =>
datVoltageL7 <= DSP_Input(1 downto 0) & myTempData;
--                          when "001001000" =>
datVoltageL0 <= DSP_Input(1 downto 0) & myTempData;;
--                          when "001001001" =>
datVoltageL0 <= DSP_Input(1 downto 0) & myTempData;;
--                          when "001001010" =>
datVoltageL0 <= DSP_Input(1 downto 0) & myTempData;;
--                          when "001001011" =>
datVoltageL0 <= DSP_Input(1 downto 0) & myTempData;;
--                          when "001001100" =>
datVoltageL0 <= DSP_Input(1 downto 0) & myTempData;;

                            when "010000100" => datLeds
<= myTempData(7 downto 0); DspSetLeds <= '1';
```

```
                            when others =>
                                null;
                            end case;
                        end if; -- End DSP_Addr(0)
                    end if; -- DSP WRITE, FPGA READ
                end if; -- End DSP Read
            end if; -- END DSP Clock event
        else
            DspSetReferenceClock <= '0';
            DspSetLeds <= '0';
        end if; -- End DSP CS
    end process P_DSP_Process;
    -- DSP_OutputEnable <= (DSP_CS = '0') AND (DSP_RD =
'0');
    DSP_Data <= DSP_Output when ((DSP_CS = '0') AND
(DSP_RD = '0')) else (others => 'Z');
    DSP_Input <= DSP_Data;

    myReferenceU <= datControlU(23 downto 0);
    myReferenceV <= datControlV(23 downto 0);
    myReferenceW <= datControlW(23 downto 0);



    ---------------------------------------------------
---------------------------------
    --
    --
    --
    --
    --
    --
    --             Handle the led display
    --
    --
    --
    --
    --
    ---------------------------------------------------
---------------------------------


    P_LedHandler: process (DspSetLeds, datLeds)
    begin
        if (DspSetLeds'event and DspSetLeds = '0') then
            myLedsOutput <= datLeds;
        else
            -- Place here all other switched for leds
            null;
        end if;
    end process P_LedHandler;
    myLeds <= myLedsOutput;



    ---------------------------------------------------
---------------------------------
    --
    --
    --
    --
    --
    --
    --             State machine for the system
    --
    --
    --
    --
    --
    ---------------------------------------------------
---------------------------------



    --
    -- IGBT Pulses Map
    --
    myGate_IGBT( 0) <= myUnitStatesOut( 0) and
myUnitLevelOut( 0);      myGate_IGBT( 1) <=
myUnitStatesOut( 0) and (NOT myUnitLevelOut( 0));
    myGate_IGBT( 2) <= myUnitStatesOut( 0) and
myUnitLevelOut( 1);      myGate_IGBT( 3) <=
myUnitStatesOut( 0) and (NOT myUnitLevelOut( 1));
```

```
    myGate_IGBT( 4) <= myUnitStatesOut( 1) and
myUnitLevelOut( 2);      myGate_IGBT( 5) <=
myUnitStatesOut( 1) and (NOT myUnitLevelOut( 2));
    myGate_IGBT( 6) <= myUnitStatesOut( 1) and
myUnitLevelOut( 3);      myGate_IGBT( 7) <=
myUnitStatesOut( 1) and (NOT myUnitLevelOut( 3));
    myGate_IGBT( 8) <= myUnitStatesOut( 2) and
myUnitLevelOut( 4);      myGate_IGBT( 9) <=
myUnitStatesOut( 2) and (NOT myUnitLevelOut( 4));
    myGate_IGBT(10) <= myUnitStatesOut( 2) and
myUnitLevelOut( 5);      myGate_IGBT(11) <=
myUnitStatesOut( 2) and (NOT myUnitLevelOut( 5));
    myGate_IGBT(12) <= myUnitStatesOut( 3) and
myUnitLevelOut( 6);      myGate_IGBT(13) <=
myUnitStatesOut( 3) and (NOT myUnitLevelOut( 6));
    myGate_IGBT(14) <= myUnitStatesOut( 3) and
myUnitLevelOut( 7);      myGate_IGBT(15) <=
myUnitStatesOut( 3) and (NOT myUnitLevelOut( 7));
    myGate_IGBT(16) <= myUnitStatesOut( 4) and
myUnitLevelOut( 8);      myGate_IGBT(17) <=
myUnitStatesOut( 4) and (NOT myUnitLevelOut( 8));
    myGate_IGBT(18) <= myUnitStatesOut( 4) and
myUnitLevelOut( 9);      myGate_IGBT(19) <=
myUnitStatesOut( 4) and (NOT myUnitLevelOut( 9));
    myGate_IGBT(20) <= myUnitStatesOut( 5) and
myUnitLevelOut(10);      myGate_IGBT(21) <=
myUnitStatesOut( 5) and (NOT myUnitLevelOut(10));
    myGate_IGBT(22) <= myUnitStatesOut( 5) and
myUnitLevelOut(11);      myGate_IGBT(23) <=
myUnitStatesOut( 5) and (NOT myUnitLevelOut(11));
    myGate_IGBT(24) <= myUnitStatesOut( 6) and
myUnitLevelOut(12);      myGate_IGBT(25) <=
myUnitStatesOut( 6) and (NOT myUnitLevelOut(12));
    myGate_IGBT(26) <= myUnitStatesOut( 6) and
myUnitLevelOut(13);      myGate_IGBT(27) <=
myUnitStatesOut( 6) and (NOT myUnitLevelOut(13));
    myGate_IGBT(28) <= myUnitStatesOut( 7) and
myUnitLevelOut(14);      myGate_IGBT(29) <=
myUnitStatesOut( 7) and (NOT myUnitLevelOut(14));
    myGate_IGBT(30) <= myUnitStatesOut( 7) and
myUnitLevelOut(15);      myGate_IGBT(31) <=
myUnitStatesOut( 7) and (NOT myUnitLevelOut(15));
    myGate_IGBT(32) <= myUnitStatesOut( 8) and
myUnitLevelOut(16);      myGate_IGBT(33) <=
myUnitStatesOut( 8) and (NOT myUnitLevelOut(16));
    myGate_IGBT(34) <= myUnitStatesOut( 8) and
myUnitLevelOut(17);      myGate_IGBT(35) <=
myUnitStatesOut( 8) and (NOT myUnitLevelOut(17));
    myGate_IGBT(36) <= myUnitStatesOut( 9) and
myUnitLevelOut(18);      myGate_IGBT(37) <=
myUnitStatesOut( 9) and (NOT myUnitLevelOut(18));
    myGate_IGBT(38) <= myUnitStatesOut( 9) and
myUnitLevelOut(19);      myGate_IGBT(39) <=
myUnitStatesOut( 9) and (NOT myUnitLevelOut(19));
    myGate_IGBT(40) <= myUnitStatesOut(10) and
myUnitLevelOut(20);      myGate_IGBT(41) <=
myUnitStatesOut(10) and (NOT myUnitLevelOut(20));
    myGate_IGBT(42) <= myUnitStatesOut(10) and
myUnitLevelOut(21);      myGate_IGBT(43) <=
myUnitStatesOut(10) and (NOT myUnitLevelOut(21));
    myGate_IGBT(44) <= myUnitStatesOut(11) and
myUnitLevelOut(22);      myGate_IGBT(45) <=
myUnitStatesOut(11) and (NOT myUnitLevelOut(22));
    myGate_IGBT(46) <= myUnitStatesOut(11) and
myUnitLevelOut(23);      myGate_IGBT(47) <=
myUnitStatesOut(11) and (NOT myUnitLevelOut(23));
    myGate_IGBT(48) <= myUnitStatesOut(12) and
myUnitLevelOut(24);      myGate_IGBT(49) <=
myUnitStatesOut(12) and (NOT myUnitLevelOut(24));
    myGate_IGBT(50) <= myUnitStatesOut(12) and
myUnitLevelOut(25);      myGate_IGBT(51) <=
myUnitStatesOut(12) and (NOT myUnitLevelOut(25));
    myGate_IGBT(52) <= myUnitStatesOut(13) and
myUnitLevelOut(26);      myGate_IGBT(53) <=
myUnitStatesOut(13) and (NOT myUnitLevelOut(26));
    myGate_IGBT(54) <= myUnitStatesOut(13) and
myUnitLevelOut(27);      myGate_IGBT(55) <=
myUnitStatesOut(13) and (NOT myUnitLevelOut(27));
    myGate_IGBT(56) <= myUnitStatesOut(14) and
myUnitLevelOut(28);      myGate_IGBT(57) <=
myUnitStatesOut(14) and (NOT myUnitLevelOut(28));
    myGate_IGBT(58) <= myUnitStatesOut(14) and
myUnitLevelOut(29);      myGate_IGBT(59) <=
myUnitStatesOut(14) and (NOT myUnitLevelOut(29));
```

```
    myGate_IGBT(60) <= myUnitStatesOut(15) and
myUnitLevelOut(30);      myGate_IGBT(61) <=
myUnitStatesOut(15) and (NOT myUnitLevelOut(30));
    myGate_IGBT(62) <= myUnitStatesOut(15) and
myUnitLevelOut(31);      myGate_IGBT(63) <=
myUnitStatesOut(15) and (NOT myUnitLevelOut(31));
    myGate_IGBT(64) <= myUnitStatesOut(16) and
myUnitLevelOut(32);      myGate_IGBT(65) <=
myUnitStatesOut(16) and (NOT myUnitLevelOut(32));
    myGate_IGBT(66) <= myUnitStatesOut(16) and
myUnitLevelOut(33);      myGate_IGBT(67) <=
myUnitStatesOut(16) and (NOT myUnitLevelOut(33));
    myGate_IGBT(68) <= myUnitStatesOut(17) and
myUnitLevelOut(34);      myGate_IGBT(69) <=
myUnitStatesOut(17) and (NOT myUnitLevelOut(34));
    myGate_IGBT(70) <= myUnitStatesOut(17) and
myUnitLevelOut(35);      myGate_IGBT(71) <=
myUnitStatesOut(17) and (NOT myUnitLevelOut(35));



    --
    -- output data
    --
--  PS2_MISO <= '0';
    PS2_MISO <= myClock_25_Hz;


    --

    -- Gate signals
    --
--  Gates_Reset <= myButtons(0) & myButtons(0) &
myButtons(0) & myButtons(0) & myButtons(0);

--  Gates_OutputEnable <= myGate_OE;

    myGate_GFault <= Gates_Fault;
    myGate_SysReset <= myGate_Reset or (mySystemReset &
mySystemReset & mySystemReset & mySystemReset &
mySystemReset);
    Gates_Reset <= myGate_SysReset;
    --
    -- Output enable
    --
    myGate_SysOE(0) <= mySystemOutput and (not
mySystemFault) and (not myGate_SysReset(0));
    myGate_SysOE(1) <= mySystemOutput and (not
mySystemFault) and (not myGate_SysReset(1));
    myGate_SysOE(2) <= mySystemOutput and (not
mySystemFault) and (not myGate_SysReset(2));
    myGate_SysOE(3) <= mySystemOutput and (not
mySystemFault) and (not myGate_SysReset(3));
    myGate_SysOE(4) <= mySystemOutput and (not
mySystemFault) and (not myGate_SysReset(4));
    Gates_OutputEnable <= myGate_SysOE;

end Behavioral;
```

H.2.II MODULATION.VHD

```
--------------------------------------------------------
---------------------------
-- Company: Aalborg University
-- Engineer: Sandu Cristian
--
-- Create Date:    17:49:49 12/08/2008
-- Design Name:
-- Module Name:    Modulation - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- Methods are:
--   000 - Staircase
--   001 - Phase shift
--   010 - Level shift (IPD)
--   011 - Level shift (APOD)
--   100 - Level shift (POD)
--------------------------------------------------------
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Modulation is
    Port (
        Clock: in std_logic;            -- Main system
clock
        Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

        -- State machine parameters
        Run: in std_logic;          -- If enable, it
converts the data input, if not, output the last data
        Done: out std_logic;            -- High when
the conversion is done

        MethodSelection: in STD_LOGIC_VECTOR(2 downto
0);      -- Method selection

        -- Input data
        ReferenceU: in STD_LOGIC_VECTOR(23 downto 0);
        ReferenceV: in STD_LOGIC_VECTOR(23 downto 0);
        ReferenceW: in STD_LOGIC_VECTOR(23 downto 0);

        ReferenceCounterMax : in  STD_LOGIC_VECTOR (23
downto 0);    -- The maximum value for the counters

        -- Measured values
        UnitVotlagesU: in STD_LOGIC_VECTOR(143 downto
0);
        UnitVotlagesV: in STD_LOGIC_VECTOR(143 downto
0);
        UnitVotlagesW: in STD_LOGIC_VECTOR(143 downto
0);

        -- Current input values
        CurrentOutputU: in STD_LOGIC_VECTOR(17 downto
0);
        CurrentOutputV: in STD_LOGIC_VECTOR(17 downto
0);
        CurrentOutputW: in STD_LOGIC_VECTOR(17 downto
0);

        -- Voltage level definition
        VoltageLevels: in STD_LOGIC_VECTOR(71 downto
0);

        -- Output data
        UnitStatesU : out  STD_LOGIC_VECTOR (7 downto
0);
        UnitStatesV : out  STD_LOGIC_VECTOR (7 downto
0);
        UnitStatesW : out  STD_LOGIC_VECTOR (7 downto
0)
    );
end Modulation;


architecture Behavioral of Modulation is
--------------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Level Shift
    --
    --
    --
    --
    component Mod_LevelShift_Phases is
        Port (
            Clock: in std_logic;            -- Main
system clock
            Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

            -- State machine parameters
            Run: in std_logic;          -- If enable,
it converts the data input, if not, output the last
data
            Done: out std_logic;            -- High
when the conversion is done


            ReferenceCounterMax: in STD_LOGIC_VECTOR(23
downto 0);      -- Main counter for the carriers

            -- Submethod
            SubMethod: in std_logic_vector(1 downto 0);

            -- Input data
            ReferenceU: in STD_LOGIC_VECTOR(23 downto
0);
            ReferenceV: in STD_LOGIC_VECTOR(23 downto
0);
            ReferenceW: in STD_LOGIC_VECTOR(23 downto
0);


            -- Output data
            UnitStatesU : out  STD_LOGIC_VECTOR (7
downto 0);
            UnitStatesV : out  STD_LOGIC_VECTOR (7
downto 0);
            UnitStatesW : out  STD_LOGIC_VECTOR (7
downto 0)
        );
    end component;
    signal myLSEnable: std_logic := '0';
    signal myLSRun: std_logic := '0';
    signal myLSDone: std_logic := '0';
    signal myLSUnitStatesU: std_logic_vector(7 downto
0) := (others => '0');
    signal myLSUnitStatesV: std_logic_vector(7 downto
0) := (others => '0');
    signal myLSUnitStatesW: std_logic_vector(7 downto
0) := (others => '0');



    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Phase Shift
    --
    --
    --
    --
    component Mod_PhaseShift_Phases is
        Port (
            Clock: in std_logic;            -- Main
system clock
            Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

            -- State machine parameters
            Run: in std_logic;          -- If enable,
it converts the data input, if not, output the last
data
            Done: out std_logic;            -- High
when the conversion is done
```

```
            -- Input data
            ReferenceU: in STD_LOGIC_VECTOR(23 downto
0);
            ReferenceV: in STD_LOGIC_VECTOR(23 downto
0);
            ReferenceW: in STD_LOGIC_VECTOR(23 downto
0);

            ReferenceCounterMax : in  STD_LOGIC_VECTOR
(23 downto 0);   -- The maximum value for the counters

            -- Output data
            UnitStatesU : out  STD_LOGIC_VECTOR (7
downto 0);
            UnitStatesV : out  STD_LOGIC_VECTOR (7
downto 0);
            UnitStatesW : out  STD_LOGIC_VECTOR (7
downto 0)
        );
    end component;
    signal myPSEnable: std_logic := '0';
    signal myPSRun: std_logic := '0';
    signal myPSDone: std_logic := '0';
    signal myPSUnitStatesU: std_logic_vector(7 downto
0) := (others => '0');
    signal myPSUnitStatesV: std_logic_vector(7 downto
0) := (others => '0');
    signal myPSUnitStatesW: std_logic_vector(7 downto
0) := (others => '0');


    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Staircase
    --
    --
    --
    --
    component Mod_Staircase_Phases
        Port (
            Clock: in std_logic;            -- Main
system clock
            Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

            -- State machine parameters
            Run: in std_logic;          -- If enable,
it converts the data input, if not, output the last
data
            Done: out std_logic;            -- High
when the conversion is done

            -- Input data
            ReferenceU: in STD_LOGIC_VECTOR(17 downto
0);
            ReferenceV: in STD_LOGIC_VECTOR(17 downto
0);
            ReferenceW: in STD_LOGIC_VECTOR(17 downto
0);

            -- Measured values
            UnitVotlagesU: in STD_LOGIC_VECTOR(143
downto 0);
            UnitVotlagesV: in STD_LOGIC_VECTOR(143
downto 0);
            UnitVotlagesW: in STD_LOGIC_VECTOR(143
downto 0);

            -- Current input values
            CurrentOutputU: in STD_LOGIC_VECTOR(17
downto 0);
            CurrentOutputV: in STD_LOGIC_VECTOR(17
downto 0);
            CurrentOutputW: in STD_LOGIC_VECTOR(17
downto 0);

            -- Voltage level definition
            VoltageLevels: in STD_LOGIC_VECTOR(71
downto 0);

            -- Output data
            UnitStatesU : out  STD_LOGIC_VECTOR (7
downto 0);
            UnitStatesV : out  STD_LOGIC_VECTOR (7
downto 0);
            UnitStatesW : out  STD_LOGIC_VECTOR (7
downto 0)
        );
    end component;
    signal mySCEnable: std_logic := '0';
    signal mySCRun: std_logic := '0';
    signal mySCDone: std_logic := '0';
    signal mySCUnitStatesU: std_logic_vector(7 downto
0) := (others => '0');
    signal mySCUnitStatesV: std_logic_vector(7 downto
0) := (others => '0');
    signal mySCUnitStatesW: std_logic_vector(7 downto
0) := (others => '0');


    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Main parameters
    --
    --
    --
    --
    signal myMethod: std_logic_vector(1 downto 0) :=
(others => '0');
    signal myLevelShiftMethod: std_logic_vector(1
downto 0) := (others => '0');

    signal myUnitStatesU : std_logic_vector(7 downto 0)
:= (others => '0');
    signal myUnitStatesV : std_logic_vector(7 downto 0)
:= (others => '0');
    signal myUnitStatesW : std_logic_vector(7 downto 0)
:= (others => '0');

    signal myDone: std_logic := '0';

begin
    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Staircase
    --
    --
    --
    --
    Staircase: Mod_Staircase_Phases port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => mySCEnable,

        Run => mySCRun,
        Done => mySCDone,

        ReferenceU => ReferenceU(17 downto 0),
        ReferenceV => ReferenceV(17 downto 0),
        ReferenceW => ReferenceW(17 downto 0),

        UnitVotlagesU => UnitVotlagesU,
        UnitVotlagesV => UnitVotlagesV,
        UnitVotlagesW => UnitVotlagesW,

        CurrentOutputU => CurrentOutputU,
        CurrentOutputV => CurrentOutputV,
        CurrentOutputW => CurrentOutputW,

        VoltageLevels => VoltageLevels,

        -- output
        UnitStatesU => mySCUnitStatesU,
        UnitStatesV => mySCUnitStatesV,
        UnitStatesW => mySCUnitStatesW
    );



    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Phase shift
    --
    --
    --
    --
    PhaseShift: Mod_PhaseShift_Phases port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => myPSEnable,

        Run => myPSRun,
        Done => myPSDone,

        ReferenceU => ReferenceU,
        ReferenceV => ReferenceV,
        ReferenceW => ReferenceW,

        ReferenceCounterMax => ReferenceCounterMax,

        -- output
        UnitStatesU => myPSUnitStatesU,
        UnitStatesV => myPSUnitStatesV,
        UnitStatesW => myPSUnitStatesW
    );

    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Level shift
    --
    --
    --
    --
    LevelShift: Mod_LevelShift_Phases port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => myLSEnable,

        Run => myLSRun,
        Done => myLSDone,

        SubMethod => myLevelShiftMethod,

        ReferenceCounterMax => ReferenceCounterMax,

        ReferenceU => ReferenceU,
        ReferenceV => ReferenceV,
        ReferenceW => ReferenceW,


        -- output
        UnitStatesU => myLSUnitStatesU,
        UnitStatesV => myLSUnitStatesV,
        UnitStatesW => myLSUnitStatesW
    );




    ---------------------------------------------------
------------
    --
    --
    --
    --          Determine method
    --
    --
    --
    --
    -- Determine method in order for the other
modulation techniques
    -- to be selected
    --
    -- Input:
    --   Clock - Main system clock
    --   Enable - Main enable signal
    --   MethodSelection - The main input parameter for
the selection
    --
    -- Output:
    --   myMethod - The selected method (00 -
Staircase, 01 - Phase shift, 10 - Level shift)
    --   myLevelShiftMethid - The level shift
configuration (00 - IPD, 01 - APOD, 10 - POD)
    --
    -- Note:
    --   The parameters are set when the Enable flag is
0 and on the
    --   rising edge of the clock
    --
    P_DetermineMethod: process (Clock, Enable,
MethodSelection)
    begin
        if (Clock'event and Clock = '1') then
            if (Enable = '0') then
                case (MethodSelection) is
                when "000" =>        -- Set staircase
                    myMethod <= "00";
                    myLevelShiftMethod <= "00";

                when "001" =>        -- Set phase shift
method
                    myMethod <= "01";
                    myLevelShiftMethod <= "00";

                when "010" =>        -- Set level shift
method, IPD
                    myMethod <= "10";
                    myLevelShiftMethod <= "00";

                when "011" =>        -- Set level shift
method, APOD
                    myMethod <= "10";
                    myLevelShiftMethod <= "01";

                when "100" =>        -- Set level shift
method, POD
                    myMethod <= "10";
                    myLevelShiftMethod <= "10";

                when others =>
                    null;
                end case;
            end if;
        end if;
    end process P_DetermineMethod;




    ---------------------------------------------------
-----------------------------------------
```

```
    --
    --
    --
    --          # Set done output
    --
    --
    --
    --
    --
    --
    ---------------------------------------------------
------------------------------------------
    P_DetermineOutput: process(Clock, Enable, myMethod,
mySCDone, myPSDone, myLSDone)
    begin
        if (Enable = '0') then
            myUnitStatesU <= (others => '0');
            myUnitStatesV <= (others => '0');
            myUnitStatesW <= (others => '0');

            myDone <= '0';
        else
            if (Clock'event and Clock = '1') then
                case (myMethod) is
                when "00" =>     -- SC
                    if (mySCDone = '1') then
                        myDone <= '1';

                        myUnitStatesU <=
mySCUnitStatesU;
                        myUnitStatesV <=
mySCUnitStatesV;
                        myUnitStatesW <=
mySCUnitStatesW;
                    else
                        myDone <= '0';
                    end if;

                when "01" =>     -- PS
                    if (myPSDone = '1') then
                        myDone <= '1';

                        myUnitStatesU <=
myPSUnitStatesU;
                        myUnitStatesV <=
myPSUnitStatesV;
                        myUnitStatesW <=
myPSUnitStatesW;
                    else
                        myDone <= '0';
                    end if;

                when "10" =>     -- LS
                    if (myLSDone = '1') then
                        myDone <= '1';

                        myUnitStatesU <=
myLSUnitStatesU;
                        myUnitStatesV <=
myLSUnitStatesV;
                        myUnitStatesW <=
myLSUnitStatesW;
                    else
                        myDone <= '0';
                    end if;

                when others =>
                    myUnitStatesU <= (others => '0');
                    myUnitStatesV <= (others => '0');
                    myUnitStatesW <= (others => '0');

                    -- Done was set to true just to
allow resume
                    myDone <= '1';
                end case;
            end if;
        end if;
    end process P_DetermineOutput;



    P_DetermineRun: process(Run, myMethod)
    begin
        if (Run = '1') then
```

```
            case (myMethod) is
            when "00" =>
                mySCRun <= '1';
                myLSRun <= '0';
                myPSRun <= '0';
            when "01" =>
                mySCRun <= '0';
                myLSRun <= '1';
                myPSRun <= '0';
            when "10" =>
                mySCRun <= '0';
                myLSRun <= '0';
                myPSRun <= '1';
            when others =>
                mySCRun <= '0';
                myLSRun <= '0';
                myPSRun <= '0';
            end case;
        else
            mySCRun <= '0';
            myLSRun <= '0';
            myPSRun <= '0';
        end if;
    end process P_DetermineRun;

    --
    --
    --    Output the data
    --
    --

    Done <= myDone;


    mySCEnable <= '1' when (myMethod = "00") else '0';
    myPSEnable <= '1' when (myMethod = "01") else '0';
    myLSEnable <= '1' when (myMethod = "10") else '0';

    UnitStatesU <= myUnitStatesU;
    UnitStatesV <= myUnitStatesV;
    UnitStatesW <= myUnitStatesW;

end Behavioral;
```

H.2.III SORTER4.VHD

```
-------------------------------------------------------
---------------------------
-- Company: Aalborg University
-- Engineer: Sandu Cristian
--
-- Create Date:    11:47:48 10/18/2008
-- Design Name:
-- Module Name:    main - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 5.0
-- Additional Comments:
-- v7.0
--     - Jump from 12 bits to 18 bits
--     - Modification for 4 sorted values
-- v6.0
--     - Corrections made to the sorting algortithm
--     - The value vector had been split
-- v5.0
--     - Improved timings (88 Mhz)
--     - Single cycle
--     - Removed acknowledge pin (the done flag is reset
by lowering the
--       enable signal)
--     - Acquisition is done when enable is low, and on
the falling edge of the clock
-- v4.0
--     - Reduced number of resources (areea
optimization)
-- v3.0
--     - Sorting done in a single step (72 Mhz)
-- v2.0
--     - Remake and small optimisation of the main
sorting algoritm\
--     - Step size reduced to 2 cycles (175 Mhz clock) +
1 acknowledge
-- v1.0
--     - Sorter algorithm done
--     - 4 clock cycles for one sorting + 2 for
validation and Anknowledge
-------------------------------------------------------
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Sorter4 is
    port (
        Clock : in std_logic;
-- The clock
        Enable: in std_logic;
-- Enable the sorter

        -- The values to be sorted
        Voltage0: in std_logic_vector(17 downto 0);
        Voltage1: in std_logic_vector(17 downto 0);
        Voltage2: in std_logic_vector(17 downto 0);
        Voltage3: in std_logic_vector(17 downto 0);

        -- The sorted values
        Sorted0: out std_logic_vector(2 downto 0);
        Sorted1: out std_logic_vector(2 downto 0);
        Sorted2: out std_logic_vector(2 downto 0);
        Sorted3: out std_logic_vector(2 downto 0);

        SorterDone: out std_logic
-- 1: When sorting is complete
    );
end Sorter4;



architecture Behavioral of Sorter4 is
    type myTCoeff is array (0 to 3) of unsigned(2
downto 0); -- integer range 0 to 7;
--   type myTComp is array (27 downto 0) of unsigned(0
downto 0);
--   type myTCompX is array (27 downto 0) of unsigned(2
downto 0);
    type myTValues is array (0 to 3) of unsigned(17
downto 0);


    signal myValues: myTValues := (others =>
"000000000000000000"); -- The saved values

    signal mySort : myTCoeff := (others => "000");
-- The final sort

    signal mySorterDone: std_logic := '0';
-- True if sorting is done

begin

    P_Sorter: process(Clock, Enable, myValues,
mySorterDone)
    variable myComp_E : std_logic_vector( 5 downto 0)
:= (others => '0'); -- The main comparators used for
duplicate values
    variable myComp_G: std_logic_vector( 5 downto 0) :=
(others => '0');     -- The comparators used for
determining the maximum
```

```
    variable myComp_L: std_logic_vector( 5 downto 0) :=
(others => '0');    -- The comparators used for
determining the minimum
    variable myCoeff: myTCoeff := (others => "000");
-- The coefficients used if duplicates are found
    begin
        if (Clock'event and Clock = '1') then
            if (Enable = '1') then
                -- If (greate) elsif (less) else
(equal)
                -- Compute the comparison between the
greater
                if (myValues(0) > myValues(1)) then
myComp_G( 0) :=  '1'; else myComp_G( 0) :=  '0'; end
if;
                if (myValues(0) > myValues(2)) then
myComp_G( 1) :=  '1'; else myComp_G( 1) :=  '0'; end
if;
                if (myValues(0) > myValues(3)) then
myComp_G( 2) :=  '1'; else myComp_G( 2) :=  '0'; end
if;
                if (myValues(1) > myValues(2)) then
myComp_G( 3) :=  '1'; else myComp_G( 3) :=  '0'; end
if;
                if (myValues(1) > myValues(3)) then
myComp_G( 4) :=  '1'; else myComp_G( 4) :=  '0'; end
if;
                if (myValues(2) > myValues(3)) then
myComp_G( 5) :=  '1'; else myComp_G( 5) :=  '0'; end
if;


                -- Compare for equality
                if (myValues(0) = myValues(1)) then
myComp_E( 0) :=  '1'; else myComp_E( 0) :=  '0'; end
if;
                if (myValues(0) = myValues(2)) then
myComp_E( 1) :=  '1'; else myComp_E( 1) :=  '0'; end
if;
                if (myValues(0) = myValues(3)) then
myComp_E( 2) :=  '1'; else myComp_E( 2) :=  '0'; end
if;
                if (myValues(1) = myValues(2)) then
myComp_E( 3) :=  '1'; else myComp_E( 3) :=  '0'; end
if;
                if (myValues(1) = myValues(3)) then
myComp_E( 4) :=  '1'; else myComp_E( 4) :=  '0'; end
if;
                if (myValues(2) = myValues(3)) then
myComp_E( 5) :=  '1'; else myComp_E( 5) :=  '0'; end
if;


                myComp_L :=  (not myComp_E) and (not
myComp_G);
                -- Compute the individual coefficients
for the modified values
                myCoeff(0)  :=  ("00" & myComp_E(0) +
myComp_E(1) + myComp_E(2));
                myCoeff(1)  :=  "00" & myComp_E(3) +
myComp_E(4);
                myCoeff(2)  :=  "00" & myComp_E(5);
                myCoeff(3)  :=  "000";

                mySort(0)  <= ("00" & myComp_G( 0) +
myComp_G( 1)) + (myComp_G( 2) + myCoeff(0));
                mySort(1)  <= ("00" & myComp_G( 3) +
myComp_G( 4)) + ("00" & myComp_L( 0) + myCoeff(1));
                mySort(2)  <= ("00" & myComp_G( 5) +
myComp_L( 1)) + ("00" & myComp_L( 3) + myCoeff(2));
                mySort(3)  <= ("00" & myComp_L( 2) +
myComp_L( 4)) + ("00" & myComp_L( 5) + myCoeff(3));

                mySorterDone <= '1';
            else
                myCoeff := (others => "000");
                mySorterDone <= '0';
            end if;
        end if;
    end process P_Sorter;


    P_SaveValues: process (Clock, Enable, Voltage0,
Voltage1, Voltage2, Voltage3)
    begin
        if (Clock'event and Clock = '1') then
            if (Enable = '0') then
                myValues(0) <= unsigned(Voltage0);
                myValues(1) <= unsigned(Voltage1);
                myValues(2) <= unsigned(Voltage2);
                myValues(3) <= unsigned(Voltage3);
            end if;
        end if;
    end process P_SaveValues;

    --
    -- Output the values
    --
    SorterDone <= mySorterDone;
    Sorted0 <= std_logic_vector(mySort(0));
    Sorted1 <= std_logic_vector(mySort(1));
    Sorted2 <= std_logic_vector(mySort(2));
    Sorted3 <= std_logic_vector(mySort(3));
end Behavioral;
```

H.2.IV PS_CARRIERS.VHD

```
-------------------------------------------------------
---------------------------
-- Company: Aalborg Univeristy
-- Engineer: Cristian Sandu
--
-- Create Date:    01:30:58 05/31/2009
-- Design Name:
-- Module Name:    Mod_PS_Carriers - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-------------------------------------------------------
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Mod_PS_Carriers is
    Port (
        Clock : in  STD_LOGIC;
        Enable : in  STD_LOGIC;

        ReferenceCounterMax : in  STD_LOGIC_VECTOR (23
downto 0);    -- The maximum value for the counters

        Carrier1p : out  STD_LOGIC_VECTOR (23 downto
0);
        Carrier2p : out  STD_LOGIC_VECTOR (23 downto
0);
        Carrier3p : out  STD_LOGIC_VECTOR (23 downto
0);
        Carrier4p : out  STD_LOGIC_VECTOR (23 downto 0)
    );
end Mod_PS_Carriers;
```

```
architecture Behavioral of Mod_PS_Carriers is
    signal myCounterA : std_logic_vector(23 downto 0)
:= (others => '0');
    signal myCounterB : std_logic_vector(23 downto 0)
:= (others => '0');
    signal myCounterC : std_logic_vector(23 downto 0)
:= (others => '0');
    signal myCounterD : std_logic_vector(23 downto 0)
:= (others => '0');

    signal myCounterSignA: std_logic := '0';
    signal myCounterSignB: std_logic := '0';
    signal myCounterSignC: std_logic := '1';
    signal myCounterSignD: std_logic := '1';

    constant cmyCounterSignA : std_logic := '0';
    constant cmyCounterSignB : std_logic := '0';
    constant cmyCounterSignC : std_logic := '1';
    constant cmyCounterSignD : std_logic := '1';


    signal myReference : std_logic_vector(23 downto 0)
:= x"100000";



begin


    P_DetermineCarriers: process (Clock, Enable,
        ReferenceCounterMax, myReference,
        myCounterA, myCounterB, myCounterB, myCounterC,
        myCounterSignA, myCounterSignB, myCounterSignC,
myCounterSignD)
    begin
        if (Enable = '0') then
            myCounterA <= (others => '0');
            myCounterB <= "0" & ReferenceCounterMax(23
downto 1);
            myCounterC <= "0" & ReferenceCounterMax(23
downto 1);
            myCounterD <= ReferenceCounterMax;

            myCounterSignA <= cmyCounterSignA;
            myCounterSignB <= cmyCounterSignB;
            myCounterSignC <= cmyCounterSignC;
            myCounterSignD <= cmyCounterSignD;

            myReference <= ReferenceCounterMax;
        else
            if (Clock'event and Clock = '1') then

                if (myCounterA = x"00000") then
                    myCounterSignA <= '0';
                    myCounterA <= myCounterA + 1;
                elsif (myCounterA = myReference) then
                    myCounterSignA <= '1';
                    myCounterA <= myCounterA - 1;
                else
                    if (myCounterSignA = '0') then
                        myCounterA <= myCounterA + 1;
                    else
                        myCounterA <= myCounterA - 1;
                    end if;
                end if;

                if (myCounterB = x"00000") then
                    myCounterSignB <= '0';
                    myCounterB <= myCounterB + 1;
                elsif (myCounterB = myReference) then
                    myCounterSignB <= '1';
                    myCounterB <= myCounterB - 1;
                else
                    if (myCounterSignB = '0') then
                        myCounterB <= myCounterB + 1;
                    else
                        myCounterB <= myCounterB - 1;
                    end if;
                end if;

                if (myCounterC = x"00000") then
                    myCounterSignC <= '0';
                    myCounterC <= myCounterC + 1;
                elsif (myCounterC = myReference) then
                    myCounterSignC <= '1';
                    myCounterC <= myCounterC - 1;
                else
                    if (myCounterSignC = '0') then
                        myCounterC <= myCounterC + 1;
                    else
                        myCounterC <= myCounterC - 1;
                    end if;
                end if;

                if (myCounterD = x"00000") then
                    myCounterSignD <= '0';
                    myCounterD <= myCounterD + 1;
                elsif (myCounterD = myReference) then
                    myCounterSignD <= '1';
                    myCounterD <= myCounterD - 1;
                else
                    if (myCounterSignD = '0') then
                        myCounterD <= myCounterD + 1;
                    else
                        myCounterD <= myCounterD - 1;
                    end if;
                end if;

            end if;
        end if;
    end process P_DetermineCarriers;


    --
    --
    -- Output values
    --
    --

    Carrier1p <= myCounterA;
    Carrier2p <= myCounterB;
    Carrier3p <= myCounterC;
    Carrier4p <= myCounterD;
end Behavioral;
```

H.2.V LS_CARRIERS.VHD

```
-------------------------------------------------------
----------------------------
-- Company: Aalborg university
-- Engineer: Cristian Sandu
--
-- Create Date:    15:45:59 06/01/2009
-- Design Name:
-- Module Name:    Mod_LS_Carriers - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-------------------------------------------------------
----------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Mod_LS_Carriers is
    Port (
        Clock : in  STD_LOGIC;
        Enable : in  STD_LOGIC;

        SubMethod : in STD_LOGIC_VECTOR(1 downto 0);
-- 00 = IPD, 01 - APOD, 10 - POD

        ReferenceCounterMax : in  STD_LOGIC_VECTOR (23
downto 0);    -- The maximum value for the counters

        Carrier1 : out  STD_LOGIC_VECTOR (23 downto 0);
        Carrier2 : out  STD_LOGIC_VECTOR (23 downto 0);
        Carrier3 : out  STD_LOGIC_VECTOR (23 downto 0);
        Carrier4 : out  STD_LOGIC_VECTOR (23 downto 0)
    );
end Mod_LS_Carriers;

architecture Behavioral of Mod_LS_Carriers is
    signal myCounter : std_logic_vector(23 downto 0) :=
(others => '0');
    signal myCounterSign: std_logic := '0';


    signal myCounter1 : std_logic_vector(23 downto 0)
:= (others => '0');
    signal myCounter2 : std_logic_vector(23 downto 0)
:= (others => '0');
    signal myCounter3 : std_logic_vector(23 downto 0)
:= (others => '0');
    signal myCounter4 : std_logic_vector(23 downto 0)
:= (others => '0');

    constant cmyCounterSign : std_logic := '0';

    signal mySubMethod: std_logic_vector(1 downto 0) :=
"00";

    signal myReference : std_logic_vector(23 downto 0)
:= x"100000";

    signal myReference1 : std_logic_vector(23 downto 0)
:= x"000000";
    signal myReference2 : std_logic_vector(23 downto 0)
:= x"000000";
    signal myReference3 : std_logic_vector(23 downto 0)
:= x"000000";

begin

    P_DetermineCarriers: process (Clock, Enable,
        ReferenceCounterMax, myReference, SubMethod,
        myReference1, myReference2, myReference3,
        myCounter, myCounterSign)
    begin
        if (Enable = '0') then
            myCounter <= (others => '0');

            myCounter1 <= (others => '0');
            myCounter2 <= (others => '0');
            myCounter3 <= (others => '0');
            myCounter4 <= (others => '0');


            myCounterSign <= cmyCounterSign;

            mySubMethod <= SubMethod;

            myReference <= "00" &
ReferenceCounterMax(21 downto 0);
            myReference1 <= "0" &
ReferenceCounterMax(21 downto 0) & "0";
            myReference2 <= myReference1 + myReference;
            myReference3 <= ReferenceCounterMax(21
downto 0) & "00";
        else
            if (Clock'event and Clock = '1') then

                if (myCounter = x"00000") then
                    myCounterSign <= '0';
                    myCounter <= myCounter + 1;
                elsif (myCounter = myReference) then
                    myCounterSign <= '1';
                    myCounter <= myCounter - 1;
                else
                    if (myCounterSign = '0') then
                        myCounter <= myCounter + 1;
                    else
                        myCounter <= myCounter - 1;
                    end if;
                end if;

                case (mySubMethod) is
                when "01" =>    -- APOD
                    myCounter1 <= myReference3 -
myCounter;
                    myCounter2 <= myCounter +
myReference1;
                    myCounter3 <= myReference1 -
myCounter;
                    myCounter4 <= myCounter;

                when "10" =>    -- POD
                    myCounter1 <= myReference3 -
myCounter;
                    myCounter2 <= myReference2 -
myCounter;
                    myCounter3 <= myReference +
myCounter;
                    myCounter4 <= myCounter;

                when others =>  -- IPD
                    myCounter1 <= myCounter +
myReference2;
                    myCounter2 <= myCounter +
myReference1;
                    myCounter3 <= myCounter +
myReference;
                    myCounter4 <= myCounter;

                end case;
            end if;
        end if;
    end process P_DetermineCarriers;


    --
    --
    -- Output values
    --
    --

    Carrier1 <= myCounter1;
    Carrier2 <= myCounter2;
    Carrier3 <= myCounter3;
    Carrier4 <= myCounter4;

end Behavioral;
```

H.2.VI PHASESHIFT.VHD

```
-------------------------------------------------------
---------------------------
-- Company: Aalborg University
-- Engineer: Sandu Cristian
--
-- Create Date:    17:49:49 12/08/2008
-- Design Name:
-- Module Name:    Staircase - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
```

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


entity Mod_PhaseShift is
    Port (
        Clock: in std_logic;            -- Main system
clock
        Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

        -- State machine parameters
        Run: in std_logic;          -- If enable, it
converts the data input, if not, output the last data
        Done: out std_logic;            -- High when
the conversion is done

        -- Input data
        Reference: in STD_LOGIC_VECTOR(23 downto 0);
-- The reference counter

        Carrier1p : in STD_LOGIC_VECTOR (23 downto 0);
-- The carrier counter
        Carrier2p : in STD_LOGIC_VECTOR (23 downto 0);
        Carrier3p : in STD_LOGIC_VECTOR (23 downto 0);
        Carrier4p : in STD_LOGIC_VECTOR (23 downto 0);

        -- Output data
        UnitStates : out  STD_LOGIC_VECTOR (7 downto 0)
    );
end Mod_PhaseShift;



architecture Behavioral of Mod_PhaseShift is

    ---------------------------------------------------
--------------
    --
    --
    --                Main parameters
    --
    --
    --
    --

    signal myUnitStates: std_logic_vector(3 downto 0)
:= (others => '0');

    signal myReference: std_logic_vector(23 downto 0)
:= (others => '0');

    signal myDone: std_logic := '0';

begin



    P_ProcessTheStates: process (Clock, Enable, Run,
Reference, myReference,
        Carrier1p, Carrier2p, Carrier3p, Carrier4p)
    begin
        if (Enable = '0') then
            myUnitStates <= (others => '0');

            myReference <= Reference;

            myDone <= '0';
        else    -- Enable = 1
            if (Carrier1p = x"00000") then
                myReference <= Reference;
            end if;
```

```vhdl
                if (Run = '1') then
--                  if (myUnitStates(0) = '0') then
--                      if (Reference < Carrier1p) then
--                          myUnitStates(0) <= '1';
--                      end if;
--                  else
--                      if (Reference > Carrier1p) then
--                          myUnitStates(0) <= '0';
--                      end if;
--                  end if;

                    if (myReference < Carrier1p) then
                        myUnitStates(0) <= '1';
                    elsif (myReference > Carrier1p) then
                        myUnitStates(0) <= '0';
                    else
                        null;
                    end if;

                    if (myReference < Carrier2p) then
                        myUnitStates(1) <= '1';
                    elsif (myReference > Carrier2p) then
                        myUnitStates(1) <= '0';
                    else
                        null;
                    end if;

                    if (myReference < Carrier3p) then
                        myUnitStates(2) <= '1';
                    elsif (myReference > Carrier3p) then
                        myUnitStates(2) <= '0';
                    else
                        null;
                    end if;

                    if (myReference < Carrier4p) then
                        myUnitStates(3) <= '1';
                    elsif (myReference > Carrier4p) then
                        myUnitStates(3) <= '0';
                    else
                        null;
                    end if;


                    myDone <= '1';
                else
                    myDone <= '0';
                end if;
            end if;
    end process P_ProcessTheStates;



    --
    --
    --    output values
    --
    --

    Done <= myDone;


    UnitStates(0) <= myUnitStates(0);
    UnitStates(1) <= myUnitStates(1);
    UnitStates(2) <= myUnitStates(2);
    UnitStates(3) <= myUnitStates(3);
    UnitStates(4) <= (not myUnitStates(3)) and Enable;
    UnitStates(5) <= (not myUnitStates(2)) and Enable;
    UnitStates(6) <= (not myUnitStates(1)) and Enable;
    UnitStates(7) <= (not myUnitStates(0)) and Enable;

end Behavioral;
```

H.2.VII PS_PHASES.VHD

```
-------------------------------------------------------
---------------------------
-- Company: Aalborg University
-- Engineer: Sandu Cristian
--
-- Create Date:    17:49:49 12/08/2008
-- Design Name:
-- Module Name:    Staircase - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-------------------------------------------------------
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Mod_PhaseShift_Phases is
    Port (
        Clock: in std_logic;            -- Main system
clock
        Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

        -- State machine parameters
        Run: in std_logic;          -- If enable, it
converts the data input, if not, output the last data
        Done: out std_logic;            -- High when
the conversion is done

        -- Input data
        ReferenceU: in STD_LOGIC_VECTOR(23 downto 0);
        ReferenceV: in STD_LOGIC_VECTOR(23 downto 0);
        ReferenceW: in STD_LOGIC_VECTOR(23 downto 0);

        ReferenceCounterMax : in  STD_LOGIC_VECTOR (23
downto 0);    -- The maximum value for the counters

        -- Output data
        UnitStatesU : out  STD_LOGIC_VECTOR (7 downto
0);
        UnitStatesV : out  STD_LOGIC_VECTOR (7 downto
0);
        UnitStatesW : out  STD_LOGIC_VECTOR (7 downto
0)
    );
end Mod_PhaseShift_Phases;


architecture Behavioral of Mod_PhaseShift_Phases is

    ---------------------------------------------------
--------------
    --
    --
    --                  Main parameters
    --
    --
    --
    --

    signal myDone: std_logic := '0';

    signal myUnitStatesU: std_logic_vector(7 downto 0);
    signal myUnitStatesV: std_logic_vector(7 downto 0);
    signal myUnitStatesW: std_logic_vector(7 downto 0);



    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Per Phase Staircase
    --
    --
    --
    --

    component Mod_PhaseShift
        port (
            Clock: in std_logic;            -- Main
system clock
            Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

            -- State machine parameters
            Run: in std_logic;          -- If enable,
it converts the data input, if not, output the last
data
            Done: out std_logic;            -- High
when the conversion is done

            -- Input data
            Reference: in STD_LOGIC_VECTOR(23 downto
0);

            Carrier1p : in STD_LOGIC_VECTOR (23 downto
0);
            Carrier2p : in STD_LOGIC_VECTOR (23 downto
0);
            Carrier3p : in STD_LOGIC_VECTOR (23 downto
0);
            Carrier4p : in STD_LOGIC_VECTOR (23 downto
0);

            -- Output data
            UnitStates : out  STD_LOGIC_VECTOR (7
downto 0)
        );
    end component;



    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Carriers
    --
    --
    --
    --

    component Mod_PS_Carriers Port (
        Clock : in  STD_LOGIC;
        Enable : in  STD_LOGIC;

        ReferenceCounterMax : in  STD_LOGIC_VECTOR (23
downto 0);    -- The maximum value for the counters

        Carrier1p : out  STD_LOGIC_VECTOR (23 downto
0);
        Carrier2p : out  STD_LOGIC_VECTOR (23 downto
0);
        Carrier3p : out  STD_LOGIC_VECTOR (23 downto
0);
        Carrier4p : out  STD_LOGIC_VECTOR (23 downto 0)
    );
    end component;


    signal myPhaseUDone: std_logic := '1';
    signal myPhaseVDone: std_logic := '1';
    signal myPhaseWDone: std_logic := '1';
    signal myCarrier1p: std_logic_vector (23 downto 0)
:= (others => '0');
    signal myCarrier2p: std_logic_vector(23 downto 0)
:= (others => '0');
    signal myCarrier3p: std_logic_vector(23 downto 0)
:= (others => '0');
    signal myCarrier4p: std_logic_vector(23 downto 0)
:= (others => '0');

begin


    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Carriers
    --
    --
    --
    --
    myCarriers: Mod_PS_Carriers port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => Enable,

        ReferenceCounterMax => ReferenceCounterMax,

        Carrier1p => myCarrier1p,
        Carrier2p => myCarrier2p,
        Carrier3p => myCarrier3p,
        Carrier4p => myCarrier4p
    );


    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Per Phase Staircase
    --
    --
    --
    --


    myPhase_U: Mod_PhaseShift port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => Enable,

        Run => Run,
        Done => myPhaseUDone,

        Reference => ReferenceU,

        Carrier1p => myCarrier1p,
        Carrier2p => myCarrier2p,
        Carrier3p => myCarrier3p,
        Carrier4p => myCarrier4p,

        -- output
        UnitStates => myUnitStatesU
    );



    myPhase_V: Mod_PhaseShift port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => Enable,

        Run => Run,
        Done => myPhaseVDone,

        Reference => ReferenceV,

        Carrier1p => myCarrier1p,
        Carrier2p => myCarrier2p,
        Carrier3p => myCarrier3p,
        Carrier4p => myCarrier4p,

        -- output
        UnitStates => myUnitStatesV
    );


    myPhase_W: Mod_PhaseShift port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => Enable,

        Run => Run,
        Done => myPhaseWDone,

        Reference => ReferenceW,

        Carrier1p => myCarrier1p,
        Carrier2p => myCarrier2p,
        Carrier3p => myCarrier3p,
        Carrier4p => myCarrier4p,

        -- output
        UnitStates => myUnitStatesW
    );


    ---------------------------------------------------
-------------------------------------------
    --
    --
    --
    --     Idle state
    --
    --          # Set done output to tru
    --
    --
    --
    --
    --
    --
    ---------------------------------------------------
-------------------------------------------
    P_SetDoneFlag: process (Enable, myDone,
myPhaseUDone, myPhaseVDone, myPhaseWDone)
    begin
        if (Enable = '1') then
            myDone <= myPhaseUDone and myPhaseVDone and
myPhaseWDone;
        else
            myDone <= '0';
        end if;
    end process P_SetDoneFlag;


    --
    --
    --    output values
    --
    --

    Done <= myDone;

    UnitStatesU <= myUnitStatesU;
    UnitStatesV <= myUnitStatesV;
    UnitStatesW <= myUnitStatesW;
```

```
end Behavioral;
```

H.2.VIII LEVELSHIFT.VHD

```
------------------------------------------------------------
---------------------------
-- Company: Aalborg University
-- Engineer: Sandu Cristian
--
-- Create Date:    17:49:49 12/08/2008
-- Design Name:
-- Module Name:    Staircase - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
------------------------------------------------------------
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Mod_LevelShift is
    Port (
        Clock: in std_logic;            -- Main system
clock
        Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

        -- State machine parameters
        Run: in std_logic;          -- If enable, it
converts the data input, if not, output the last data
        Done: out std_logic;            -- High when
the conversion is done

        -- Input data
        Reference: in STD_LOGIC_VECTOR(23 downto 0);
-- The reference counter

        Carrier1 : in STD_LOGIC_VECTOR (23 downto 0);
-- The carrier counter
        Carrier2 : in STD_LOGIC_VECTOR (23 downto 0);
        Carrier3 : in STD_LOGIC_VECTOR (23 downto 0);
        Carrier4 : in STD_LOGIC_VECTOR (23 downto 0);

        -- Output data
        UnitStates : out  STD_LOGIC_VECTOR (7 downto 0)
    );
end Mod_LevelShift;


architecture Behavioral of Mod_LevelShift is

    ---------------------------------------------------
--------------
    --
    --
    --              Main parameters
    --
    --
    --
    --

    signal myUnitStates: std_logic_vector(7 downto 0)
:= (others => '0');

    signal myReference: std_logic_vector(23 downto 0)
:= (others => '0');

    signal myDone: std_logic := '0';

    signal myToggleStatesUp: std_logic := '0';
    signal myToggleStatesDown: std_logic := '0';

    signal myUnitL0 : std_logic := '1';
    signal myUnitL1 : std_logic := '0';
    signal myUnitL2 : std_logic := '1';
    signal myUnitL3 : std_logic := '0';

begin




    P_ProcessTheStates: process (Clock, Enable, Run,
Reference, myReference,
        Carrier1, Carrier2, Carrier3, Carrier4,
myUnitL0, myUnitL1, myUnitL2, myUnitL3,
        myToggleStatesDown, myToggleStatesUp)
    begin
        if (Enable = '0') then
            myUnitStates <= (others => '0');

            myReference <= Reference;

            myDone <= '0';

            myUnitL0 <= '1';
            myUnitL1 <= '0';
            myUnitL2 <= '1';
            myUnitL3 <= '0';
        else    -- Enable = 1

            if (Clock'event and Clock = '0') then
                if (Carrier1 = myReference(21 downto 0)
& "00") then
                    -- Togle between the two upper
units
                    myToggleStatesUp <=
myToggleStatesDown;
                end if;

                if (Carrier4 = x"000000") then
                    -- Set the reference
                    myReference <= Reference;

                    myToggleStatesDown <= not
myToggleStatesDown;
                end if;
            end if;


            if (Run = '1') then
--              if (myUnitStates(0) = '0') then
--                  if (Reference < Carrier1p) then
--                      myUnitStates(0) <= '1';
--                  end if;
--              else
--                  if (Reference > Carrier1p) then
--                      myUnitStates(0) <= '0';
--                  end if;
--              end if;

                if (myReference < Carrier1) then
                    myUnitL0 <= '1';
                elsif (myReference > Carrier1) then
                    myUnitL0 <= '0';
                else
                    null;
                end if;

                if (myReference < Carrier2) then
                    myUnitL1 <= '1';
                elsif (myReference > Carrier2) then
                    myUnitL1 <= '0';
                else
            null;
        end if;

        if (myReference < Carrier3) then
            myUnitL2 <= '1';
        elsif (myReference > Carrier3) then
            myUnitL2 <= '0';
        else
            null;
        end if;

        if (myReference < Carrier4) then
            myUnitL3 <= '1';
        elsif (myReference > Carrier4) then
            myUnitL3 <= '0';
        else
            null;
        end if;


        if (myToggleStatesUp = '1') then
            myUnitStates(4) <= myUnitL0;
            myUnitStates(6) <= myUnitL2;
        else
            myUnitStates(6) <= myUnitL0;
            myUnitStates(4) <= myUnitL2;
        end if;

        if (myToggleStatesDown = '1') then
            myUnitStates(5) <= myUnitL1;
            myUnitStates(7) <= myUnitL3;
        else
                myUnitStates(7) <= myUnitL1;
                myUnitStates(5) <= myUnitL3;
            end if;

            myDone <= '1';
        else
            myDone <= '0';
        end if;
    end if;
end process P_ProcessTheStates;



--
--
--    output values
--
--

Done <= myDone;


UnitStates(0) <= (not myUnitStates(7)) and Enable;
UnitStates(1) <= (not myUnitStates(6)) and Enable;
UnitStates(2) <= (not myUnitStates(5)) and Enable;
UnitStates(3) <= (not myUnitStates(4)) and Enable;
UnitStates(4) <= myUnitStates(4);
UnitStates(5) <= myUnitStates(5);
UnitStates(6) <= myUnitStates(6);
UnitStates(7) <= myUnitStates(7);

end Behavioral;
```

H.2.IX LS_PHASES.VHD

```
-------------------------------------------------------
---------------------------
-- Company: Aalborg University
-- Engineer: Sandu Cristian
--
-- Create Date:    17:49:49 12/08/2008
-- Design Name:
-- Module Name:    Staircase - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-------------------------------------------------------
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Mod_LevelShift_Phases is
    Port (
        Clock: in std_logic;            -- Main system
clock
        Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

        -- State machine parameters
        Run: in std_logic;          -- If enable, it
converts the data input, if not, output the last data
        Done: out std_logic;            -- High when
the conversion is done

        -- The submethod (00 - IPD, 01 - APOD, 10 -
POD, 11 - Reserved)
        SubMethod: in std_logic_vector(1 downto 0);

        ReferenceCounterMax: in STD_LOGIC_VECTOR(23
downto 0);      -- Main counter for the carriers

        -- Input data
        ReferenceU: in STD_LOGIC_VECTOR(23 downto 0);
        ReferenceV: in STD_LOGIC_VECTOR(23 downto 0);
        ReferenceW: in STD_LOGIC_VECTOR(23 downto 0);


        -- Output data
        UnitStatesU : out  STD_LOGIC_VECTOR (7 downto
0);
        UnitStatesV : out  STD_LOGIC_VECTOR (7 downto
0);
        UnitStatesW : out  STD_LOGIC_VECTOR (7 downto
0)
    );
end Mod_LevelShift_Phases;


architecture Behavioral of Mod_LevelShift_Phases is

    -------------------------------------------------------
--------------
    --
    --
    --                  Main parameters
    --
    --
    --
    --

    signal myDone: std_logic := '0';

    signal myUnitStatesU: std_logic_vector(7 downto 0);
    signal myUnitStatesV: std_logic_vector(7 downto 0);
    signal myUnitStatesW: std_logic_vector(7 downto 0);


    -------------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                   Carriers
    --
    --
    --
    --


    component Mod_LS_Carriers
    Port (
        Clock : in  STD_LOGIC;
        Enable : in  STD_LOGIC;

        SubMethod : in STD_LOGIC_VECTOR(1 downto 0);
-- 00 = IPD, 01 - APOD, 10 - POD

        ReferenceCounterMax : in  STD_LOGIC_VECTOR (23
downto 0);    -- The maximum value for the counters

        Carrier1 : out  STD_LOGIC_VECTOR (23 downto 0);
        Carrier2 : out  STD_LOGIC_VECTOR (23 downto 0);
        Carrier3 : out  STD_LOGIC_VECTOR (23 downto 0);
        Carrier4 : out  STD_LOGIC_VECTOR (23 downto 0)
    );
    end component;
    signal myCarrier1 : std_logic_vector(23 downto 0)
:= (others => '0');
    signal myCarrier2 : std_logic_vector(23 downto 0)
:= (others => '0');
    signal myCarrier3 : std_logic_vector(23 downto 0)
:= (others => '0');
    signal myCarrier4 : std_logic_vector(23 downto 0)
:= (others => '0');


    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Per Phase Staircase
    --
    --
    --
    --

    component Mod_LevelShift
        port (
            Clock: in std_logic;            -- Main
system clock
            Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

            -- State machine parameters
            Run: in std_logic;          -- If enable,
it converts the data input, if not, output the last
data
            Done: out std_logic;            -- High
when the conversion is done

            -- Input data
            Reference: in STD_LOGIC_VECTOR(23 downto
0);        -- The reference counter

            Carrier1 : in STD_LOGIC_VECTOR (23 downto
0);    -- The carrier counter
            Carrier2 : in STD_LOGIC_VECTOR (23 downto
0);

            Carrier3 : in STD_LOGIC_VECTOR (23 downto
0);

            Carrier4 : in STD_LOGIC_VECTOR (23 downto
0);

            -- Output data
            UnitStates : out  STD_LOGIC_VECTOR (7
downto 0)
        );
    end component;

    signal myPhaseUDone: std_logic := '1';
    signal myPhaseVDone: std_logic := '1';
    signal myPhaseWDone: std_logic := '1';
```

```
begin
    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Carriers
    --
    --
    --
    --
    myCarriers: Mod_LS_Carriers port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => Enable,

        SubMethod => SubMethod,

        ReferenceCounterMax => ReferenceCounterMax,

        Carrier1 => myCarrier1,
        Carrier2 => myCarrier2,
        Carrier3 => myCarrier3,
        Carrier4 => myCarrier4
    );


    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Per Phase Staircase
    --
    --
    --
    --


    myPhase_U: Mod_LevelShift port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => Enable,

        Run => Run,
        Done => myPhaseUDone,

        Reference => ReferenceU,

        Carrier1 => myCarrier1,
        Carrier2 => myCarrier2,
        Carrier3 => myCarrier3,
        Carrier4 => myCarrier4,

        -- output
        UnitStates => myUnitStatesU
    );


    myPhase_V: Mod_LevelShift port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => Enable,

        Run => Run,
        Done => myPhaseVDone,
        Reference => ReferenceV,

        Carrier1 => myCarrier1,
        Carrier2 => myCarrier2,
        Carrier3 => myCarrier3,
        Carrier4 => myCarrier4,

        -- output
        UnitStates => myUnitStatesV
    );



    myPhase_W: Mod_LevelShift port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => Enable,

        Run => Run,
        Done => myPhaseWDone,

        Reference => ReferenceW,

        Carrier1 => myCarrier1,
        Carrier2 => myCarrier2,
        Carrier3 => myCarrier3,
        Carrier4 => myCarrier4,

        -- output
        UnitStates => myUnitStatesW
    );


    ---------------------------------------------------
-----------------------------------------
    --
    --
    --
    --     Idle state
    --
    --         # Set done output
    --
    --
    --
    --
    --
    --
    ---------------------------------------------------
-----------------------------------------
    P_SetDoneFlag: process (Enable, myDone,
myPhaseUDone, myPhaseVDone, myPhaseWDone)
    begin
        if (Enable = '1') then
            myDone <= myPhaseUDone and myPhaseVDone and
myPhaseWDone;
        else
            myDone <= '0';
        end if;
    end process P_SetDoneFlag;


    --
    --
    --    output values
    --
    --

    Done <= myDone;

    UnitStatesU <= myUnitStatesU;
    UnitStatesV <= myUnitStatesV;
    UnitStatesW <= myUnitStatesW;

end Behavioral;
```

H.2.X STAIRCASE.VHD

```
-------------------------------------------------------
---------------------------
-- Company: Aalborg University
-- Engineer: Sandu Cristian
--
-- Create Date:    17:49:49 12/08/2008
-- Design Name:
-- Module Name:    SimpleUnit2ComplexUnit - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-------------------------------------------------------
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity MainControl is
    Port (
        Clock: in std_logic;              -- Main system
clock
        Enable: in std_logic;         -- Enable the
conversion (if disable, set to 0 all outputs)

        Run: in std_logic;            -- If enable, it
converts the data input, if not, output the last data

        Done: out std_logic;              -- High when
the conversion is done

        -- Input data
        MethodSelection: in STD_LOGIC_VECTOR(2 downto
0);      -- Method selection

        -- Input data
        ReferenceU: in STD_LOGIC_VECTOR(23 downto 0);
        ReferenceV: in STD_LOGIC_VECTOR(23 downto 0);
        ReferenceW: in STD_LOGIC_VECTOR(23 downto 0);

        ReferenceCounterMax : in  STD_LOGIC_VECTOR (23
downto 0);    -- The maximum value for the counters

        -- Measured values
        UnitVotlagesU: in STD_LOGIC_VECTOR(143 downto
0);

        UnitVotlagesV: in STD_LOGIC_VECTOR(143 downto
0);

        UnitVotlagesW: in STD_LOGIC_VECTOR(143 downto
0);


        -- Current input values
        CurrentOutputU: in STD_LOGIC_VECTOR(17 downto
0);

        CurrentOutputV: in STD_LOGIC_VECTOR(17 downto
0);

        CurrentOutputW: in STD_LOGIC_VECTOR(17 downto
0);


        -- Voltage level definition
        VoltageLevels: in STD_LOGIC_VECTOR(71 downto
0);

        -- Output data
        UnitStatesU : out  STD_LOGIC_VECTOR (7 downto
0);

        UnitStatesV : out  STD_LOGIC_VECTOR (7 downto
0);
```

```
        UnitStatesW : out  STD_LOGIC_VECTOR (7 downto
0);


        -- Output data
        UnitStateOut : out  STD_LOGIC_VECTOR (29 downto
0);     -- States for 8 * 3 units
        UnitLevelOut : out  STD_LOGIC_VECTOR (59 downto
0)          -- Levels for 8 * 3 * 2 legs
);
end MainControl;




architecture Behavioral of MainControl is
    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  State machine variables
    --
    --
    --
    --


    type TStateType is (State_DoControl,
State_DoUnitConversion, State_DoMapping,
State_DoTransmit, State_Idle);
    signal myCurrentState, myNextState : TStateType;

    signal myDone : std_logic := '0';

    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Control parameters
    --
    --
    --
    --
    component Modulation
        Port (
            Clock: in std_logic;            -- Main
system clock
            Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

            -- State machine parameters
            Run: in std_logic;          -- If enable,
it converts the data input, if not, output the last
data
            Done: out std_logic;            -- High
when the conversion is done

            MethodSelection: in STD_LOGIC_VECTOR(2
downto 0);      -- Method selection

            -- Input data
            ReferenceU: in STD_LOGIC_VECTOR(23 downto
0);
            ReferenceV: in STD_LOGIC_VECTOR(23 downto
0);
            ReferenceW: in STD_LOGIC_VECTOR(23 downto
0);

            ReferenceCounterMax : in  STD_LOGIC_VECTOR
(23 downto 0);    -- The maximum value for the counters

            -- Measured values
            UnitVotlagesU: in STD_LOGIC_VECTOR(143
downto 0);
```

```
            UnitVotlagesV: in STD_LOGIC_VECTOR(143
downto 0);
            UnitVotlagesW: in STD_LOGIC_VECTOR(143
downto 0);

            -- Current input values
            CurrentOutputU: in STD_LOGIC_VECTOR(17
downto 0);
            CurrentOutputV: in STD_LOGIC_VECTOR(17
downto 0);
            CurrentOutputW: in STD_LOGIC_VECTOR(17
downto 0);

            -- Voltage level definition
            VoltageLevels: in STD_LOGIC_VECTOR(71
downto 0);

            -- Output data
            UnitStatesU : out  STD_LOGIC_VECTOR (7
downto 0);
            UnitStatesV : out  STD_LOGIC_VECTOR (7
downto 0);
            UnitStatesW : out  STD_LOGIC_VECTOR (7
downto 0)
        );
    end component;
    signal myControlRun: std_logic := '0';
    signal myControlDone: std_logic := '0';

    signal myControlUnitStatesU: std_logic_vector(7
downto 0);
    signal myControlUnitStatesV: std_logic_vector(7
downto 0);
    signal myControlUnitStatesW: std_logic_vector(7
downto 0);



    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Unit Conversion parameters
    --
    --
    --
    --

    component Unit2Igbts
        Port (
            Clock: in std_logic;            -- Main
system clock
            Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

            Run: in std_logic;          -- If enable,
it converts the data input, if not, output the last
data

            Done: out std_logic;            -- High
when the conversion is done

            -- Input data
            UnitIn : in  STD_LOGIC_VECTOR ( 7 downto
0);
            CurrentOutput: in STD_LOGIC_VECTOR(17
downto 0);

            -- Output data
            UnitStateOut : out  STD_LOGIC_VECTOR ( 7
downto 0);    -- States for 8 * 3 units
            UnitLevelOut : out  STD_LOGIC_VECTOR (15
downto 0)          -- Levels for 8 * 3 * 2 legs
        );
    end component;

    signal myConvDoneU: std_logic := '0';
    signal myConvDoneV: std_logic := '0';
    signal myConvDoneW: std_logic := '0';
    signal myConvRun: std_logic := '0';

    signal myConvUnitStatesU: std_logic_vector( 7
downto 0) := (others => '0');
    signal myConvUnitLevelsU: std_logic_vector(15
downto 0) := (others => '0');
    signal myConvUnitStatesV: std_logic_vector( 7
downto 0) := (others => '0');
    signal myConvUnitLevelsV: std_logic_vector(15
downto 0) := (others => '0');
    signal myConvUnitStatesW: std_logic_vector( 7
downto 0) := (others => '0');
    signal myConvUnitLevelsW: std_logic_vector(15
downto 0) := (others => '0');


    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Unit Mapping parameters
    --
    --
    --
    --

    component UnitMapping
        Port (
            Clock: in std_logic;            -- Main
system clock
            Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

            Run: in std_logic;          -- If enable,
it converts the data input, if not, output the last
data

            Done: out std_logic;            -- High
when the conversion is done

            -- Input data from Unit2IGBTs
            UnitStateIn : in   STD_LOGIC_VECTOR (23
downto 0);          -- States for 8 * 3 units
            UnitLevelIn : in   STD_LOGIC_VECTOR (47
downto 0);          -- Levels for 8 * 3 * 2 legs

            -- Output data from Unit2IGBTs
            UnitStateOut : out   STD_LOGIC_VECTOR (29
downto 0);      -- States for 6 * 5 units
            UnitLevelOut : out   STD_LOGIC_VECTOR (59
downto 0)           -- Levels for 6 * 5 * 2 legs
        );
    end component;

    signal myMapDone: std_logic := '0';
    signal myMapRun: std_logic := '0';

    signal myMapUnitStates: std_logic_vector(29 downto
0) := (others => '0');
    signal myMapUnitLevels: std_logic_vector(59 downto
0) := (others => '0');



    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Transmit parameters
    --
    --
    --
    --
```

```
     signal myTransmitRun: std_logic := '0';
     signal myTransmitDone: std_logic := '0';

begin

   SYNC_PROC: process (Clock, Run)
   begin
       if (Run = '1') then
          if (Clock'event and Clock = '1') then
          myCurrentState <= myNextState;
        end if;
       else    -- Enable = 0
          myCurrentState <= State_Idle;
      end if;
   end process;


   OE_DoControl: process (myCurrentState)
   begin
      --insert statements to decode internal output
signals
      --below is simple example
      if myCurrentState = State_DoControl then
         myControlRun <= '1';
      else
         myControlRun <= '0';
      end if;
   end process;


    OE_DoConversion: process (myCurrentState)
   begin
      --insert statements to decode internal output
signals
      --below is simple example
      if myCurrentState = State_DoUnitConversion then
         myConvRun <= '1';
      else
         myConvRun <= '0';
      end if;
   end process;


    OE_DoMap: process (myCurrentState)
   begin
      --insert statements to decode internal output
signals
      --below is simple example
      if myCurrentState = State_DoMapping then
         myMapRun <= '1';
      else
         myMapRun <= '0';
      end if;
   end process;


    OE_DoTransmit: process (myCurrentState)
   begin
      --insert statements to decode internal output
signals
      --below is simple example
      if myCurrentState = State_DoTransmit then
         myTransmitRun <= '1';
      else
         myTransmitRun <= '0';
      end if;
   end process;


   NEXT_STATE_DECODE: process (myCurrentState,
myNextState, myControlDone, myConvDoneU, myConvDoneV,
myConvDoneW, myMapDone, myTransmitDone)
   begin
      --declare default state for next_state to avoid
latches
      myNextState <= myCurrentState;

      --insert statements to decode next_state
      case (myCurrentState) is
         when State_DoControl =>
            if myControlDone = '1' then
               myNextState <= State_DoUnitConversion;
            end if;
         when State_DoUnitConversion =>
            if myConvDoneU = '1' and myConvDoneV = '1'
and myConvDoneW = '1' then
               myNextState <= State_DoMapping;
            end if;
         when State_DoMapping =>
            if myMapDone = '1' then
               myNextState <= State_DoTransmit;
            end if;
         when State_DoTransmit =>
            if myTransmitDone = '1' then
               myNextState <= State_Idle;
            end if;
         when others =>
            myNextState <= State_Idle;
      end case;
   end process;


    ---------------------------------------------------
--------------------------------
    --
    --
    --
    --
    --
    --
    --              Modulation
    --
    --
    --
    --
    --
    ---------------------------------------------------
--------------------------------

    myModulation: Modulation port map(
        Clock => Clock,
        Enable => Enable,

        Run => myControlRun,
        Done => myControlDone,

        MethodSelection => MethodSelection,

        ReferenceU => ReferenceU,
        ReferenceV => ReferenceV,
        ReferenceW => ReferenceW,

        ReferenceCounterMax => ReferenceCounterMax,

        UnitVotlagesU => UnitVotlagesU,
        UnitVotlagesV => UnitVotlagesV,
        UnitVotlagesW => UnitVotlagesW,

        CurrentOutputU => CurrentOutputU,
        CurrentOutputV => CurrentOutputV,
        CurrentOutputW => CurrentOutputW,

        VoltageLevels => VoltageLevels,

        UnitStatesU => myControlUnitStatesU,
        UnitStatesV => myControlUnitStatesV,
        UnitStatesW => myControlUnitStatesW
    );


    ---------------------------------------------------
--------------------------------
    --
    --
    --
    --
    --
    --
    --              Unit conversion  (Unit 2 IGBTs)
    --
    --
    --
    --
    --
    ---------------------------------------------------
--------------------------------

    myUnit2Igbt_U: Unit2Igbts port map(
        Clock => Clock,
        Enable => Enable,

        Run => myConvRun,

        Done => myConvDoneU,

        UnitIn => myControlUnitStatesU,

        CurrentOutput => CurrentOutputU,

        UnitStateOut => myConvUnitStatesU,
        UnitLevelOut => myConvUnitLevelsU
    );

    myUnit2Igbt_V: Unit2Igbts port map(
        Clock => Clock,
        Enable => Enable,

        Run => myConvRun,

        Done => myConvDoneV,

        UnitIn => myControlUnitStatesV,

        CurrentOutput => CurrentOutputV,

        UnitStateOut => myConvUnitStatesV,
        UnitLevelOut => myConvUnitLevelsV
    );

    myUnit2Igbt_W: Unit2Igbts port map(
        Clock => Clock,
        Enable => Enable,

        Run => myConvRun,

        Done => myConvDoneW,

        UnitIn => myControlUnitStatesW,

        CurrentOutput => CurrentOutputW,

        UnitStateOut => myConvUnitStatesW,
        UnitLevelOut => myConvUnitLevelsW
    );


    ---------------------------------------------------
--------------------------------
    --
    --
    --
    --
    --
    --
    --              Unit mappings
    --
    --
    --
    --
    --
    ---------------------------------------------------
--------------------------------

    myUnitMapping: UnitMapping port map(
        Clock => Clock,
        Enable => Enable,

        Run => myMapRun,

        Done => myMapDone,

        UnitStateIn( 7 downto  0) => myConvUnitStatesU,
        UnitStateIn(15 downto  8) => myConvUnitStatesV,
        UnitStateIn(23 downto 16) => myConvUnitStatesW,

        UnitLevelIn(15 downto  0) => myConvUnitLevelsU,
        UnitLevelIn(31 downto 16) => myConvUnitLevelsV,
        UnitLevelIn(47 downto 32) => myConvUnitLevelsW,

        UnitStateOut => myMapUnitStates,
        UnitLevelOut => myMapUnitLevels
    );


    ---------------------------------------------------
-----------------------------------------
    --
    --
    --
    --          # Set done output to tru
    --
    --
    --
    --
    --
    --
    ---------------------------------------------------
-----------------------------------------
    P_SetDoneFlag: process (Enable, myDone, myMapDone)
    begin
        if (Enable = '1') then
            myDone <= myMapDone;
        else
            myDone <= '0';
        end if;
    end process P_SetDoneFlag;


    --
    -- Output data
    --
    UnitStateOut <= myMapUnitStates;
    UnitLevelOut <= myMapUnitLevels;


    UnitStatesU <= myControlUnitStatesU;
    UnitStatesV <= myControlUnitStatesV;
    UnitStatesW <= myControlUnitStatesW;

    Done <= myDone and myMapDone;

end Behavioral;
```

H.2.XI STAIRCASE_PHASES.VHD

```
-------------------------------------------------------
---------------------------
-- Company: Aalborg University
-- Engineer: Sandu Cristian
--
-- Create Date:    17:49:49 12/08/2008
-- Design Name:
-- Module Name:    Staircase - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-------------------------------------------------------
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Mod_Staircase_Phases is
    Port (
        Clock: in std_logic;            -- Main system
clock
        Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

        -- State machine parameters
        Run: in std_logic;          -- If enable, it
converts the data input, if not, output the last data
        Done: out std_logic;            -- High when
the conversion is done

        -- Input data
        ReferenceU: in STD_LOGIC_VECTOR(17 downto 0);
        ReferenceV: in STD_LOGIC_VECTOR(17 downto 0);
        ReferenceW: in STD_LOGIC_VECTOR(17 downto 0);


        -- Measured values
        UnitVotlagesU: in STD_LOGIC_VECTOR(143 downto
0);
        UnitVotlagesV: in STD_LOGIC_VECTOR(143 downto
0);
        UnitVotlagesW: in STD_LOGIC_VECTOR(143 downto
0);

        -- Current input values
        CurrentOutputU: in STD_LOGIC_VECTOR(17 downto
0);
        CurrentOutputV: in STD_LOGIC_VECTOR(17 downto
0);
        CurrentOutputW: in STD_LOGIC_VECTOR(17 downto
0);

        -- Voltage level definition
        VoltageLevels: in STD_LOGIC_VECTOR(71 downto
0);

        -- Output data
        UnitStatesU : out  STD_LOGIC_VECTOR (7 downto
0);
        UnitStatesV : out  STD_LOGIC_VECTOR (7 downto
0);
        UnitStatesW : out  STD_LOGIC_VECTOR (7 downto
0)
    );
end Mod_Staircase_Phases;


architecture Behavioral of Mod_Staircase_Phases is

    ---------------------------------------------------
--------------
    --
    --
    --                  Main parameters
    --
    --
    --
    --

    signal myUnitStatesU: std_logic_vector(7 downto 0);
    signal myUnitStatesV: std_logic_vector(7 downto 0);
    signal myUnitStatesW: std_logic_vector(7 downto 0);

    signal myDone: std_logic := '0';



    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Per Phase Staircase
    --
    --
    --
    --

    component Mod_Staircase
```

```
        port (
            Clock: in std_logic;            -- Main
system clock
            Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

            -- State machine parameters
            Run: in std_logic;          -- If enable,
it converts the data input, if not, output the last
data
            Done: out std_logic;            -- High
when the conversion is done

            -- Input data
            Reference: in STD_LOGIC_VECTOR(17 downto
0);

            -- Measured values
            UnitVoltages: in STD_LOGIC_VECTOR(143
downto 0);

            -- Current input values
            CurrentOutput: in STD_LOGIC_VECTOR(17
downto 0);

            -- Voltage level definition
            VoltageLevels: in STD_LOGIC_VECTOR(71
downto 0);

            -- Output data
            UnitStates : out  STD_LOGIC_VECTOR (7
downto 0)
        );
    end component;

    signal myPhaseUDone: std_logic := '1';
    signal myPhaseVDone: std_logic := '1';
    signal myPhaseWDone: std_logic := '1';

begin

    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Per Phase Staircase
    --
    --
    --
    --


    myPhase_U: Mod_Staircase port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => Enable,

        Run => Run,
        Done => myPhaseUDone,

        Reference => ReferenceU,
        UnitVoltages => UnitVotlagesU,
        CurrentOutput => CurrentOutputU,
        VoltageLevels => VoltageLevels,

        -- output
        UnitStates => myUnitStatesU
    );


    myPhase_V: Mod_Staircase port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => Enable,

        Run => Run,
        Done => myPhaseVDone,

        Reference => ReferenceV,
        UnitVoltages => UnitVotlagesV,
        CurrentOutput => CurrentOutputV,
        VoltageLevels => VoltageLevels,

        -- output
        UnitStates => myUnitStatesV
    );


    myPhase_W: Mod_Staircase port map(
        --
        -- Main parameters
        --
        Clock => Clock,
        Enable => Enable,

        Run => Run,
        Done => myPhaseWDone,

        Reference => ReferenceW,
        UnitVoltages => UnitVotlagesW,
        CurrentOutput => CurrentOutputW,
        VoltageLevels => VoltageLevels,

        -- output
        UnitStates => myUnitStatesW
    );


    ---------------------------------------------------
------------------------------------------
    --
    --
    --
    --     Idle state
    --
    --         # Set done output to tru
    --
    --
    --
    --
    --
    --
    ---------------------------------------------------
------------------------------------------
    P_SetDoneFlag: process (Enable, myDone,
myPhaseUDone, myPhaseVDone, myPhaseWDone)
    begin
        if (Enable = '1') then
            myDone <= myPhaseUDone and myPhaseVDone and
myPhaseWDone;
        else
            myDone <= '0';
        end if;
    end process P_SetDoneFlag;


    --
    --
    --    output values
    --
    --

    Done <= myDone;

    UnitStatesU <= myUnitStatesU;
    UnitStatesV <= myUnitStatesV;
    UnitStatesW <= myUnitStatesW;

end Behavioral;
```

H.2.XII UNIT2IGBT.VHD

```
-------------------------------------------------------
---------------------------
-- Company: Aalborg University
-- Engineer: Sandu Cristian
--
-- Create Date:    17:49:49 12/08/2008
-- Design Name:
-- Module Name:    SimpleUnit2ComplexUnit - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-------------------------------------------------------
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Unit2Igbts is
    Port (
        Clock: in std_logic;             -- Main system
clock
        Enable: in std_logic;        -- Enable the
conversion (if disable, set to 0 all outputs)

        Run: in std_logic;           -- If enable, it
converts the data input, if not, output the last data

        Done: out std_logic;             -- High when
the conversion is done

        -- Input data
        UnitIn : in  STD_LOGIC_VECTOR (7 downto 0);

        CurrentOutput: in STD_LOGIC_VECTOR(17 downto
0);

        -- Output data
        UnitStateOut : out  STD_LOGIC_VECTOR ( 7 downto
0);     -- States for 8 * 3 units
        UnitLevelOut : out  STD_LOGIC_VECTOR (15 downto
0)          -- Levels for 8 * 3 * 2 legs
    );
end Unit2Igbts;




architecture Behavioral of Unit2Igbts is
    signal myUnitStates: std_logic_vector( 7 downto 0)
:= (others => '0');
    signal myUnitLevels: std_logic_vector(15 downto 0)
:= (others => '0');

    signal myCurrentSign : std_logic := '0';

    signal myDone : std_logic := '0';

    constant mycCurrentJumpLevel_Poz:
std_logic_vector(17 downto 0) := "00" & x"00A0";       -
- 0,3 Amps = The limit imposed in order to determine
zero crossing
    constant mycCurrentJumpLevel_Neg:
std_logic_vector(17 downto 0) := "11" & x"FF5F";       -
- -0,3 Amps = The negative limit imposed for the
negative zero crossing


begin

    --      +-----------+------------+
    --     |           |            |
    --     +---+       +---+        |
    -- T1  |   |   T3  |   |        |
    --     /  ---      /  ---       |
    --  -||   / \    -||  / \       |
    --      \  ---      \  ---       |
    --       |   |       |   |        |
    --       +---+       +---+        |
    --  +   |             |          + |
    --  ----+             |          -----
    -- ----|-----------+          -----
    -- Load|           |            |
    --     +---+       +---+        |
    -- T2  |   |   T4  |   |        |
    --     /  ---      /  ---       |
    --  -||   / \   -||   / \       |
    --      \  ---      \  ---       |
    --       |   |       |   |        |
    --       +---+       +---+        |
    --       |             |            |
    --      +-----------+------------+
    --
    --
    -- States
    --   0: -- -- -- --
    --   1: T1             - Capacitor -> DC Bus
    --   2:    T2
    --   3:       T3       - DC BUS -> Load
    --   4:          T4
    --   5: T1    T3       - Current sign +/-: DC Bus <->
load -------------- Used
    --   6: T1       T4  - Current sign +: DC Bus ->
capacitor -> load ---- Used
    --   7:    T2 T3     - Current sign -: load ->
capacitor -> DC Bus ---- Used
    --   8:    T2    T4  - Current sign +/-: load <->
DC Bus
    --
    P_Simple2Complex: process(Clock, Enable, UnitIn,
myCurrentSign)
    begin
        if (Enable = '0') then
            myUnitStates <= (others => '0');
            myUnitLevels <= (others => '0');
            myDone <= '0';
        else
            if (Run = '1') then
                myUnitStates(7 downto 0) <= (others =>
'1');
                If (myCurrentSign = '0') then
                    myUnitLevels( 0) <= '1';
myUnitLevels( 1) <= UnitIn( 0);
                    myUnitLevels( 2) <= '1';
myUnitLevels( 3) <= UnitIn( 1);
                    myUnitLevels( 4) <= '1';
myUnitLevels( 5) <= UnitIn( 2);
                    myUnitLevels( 6) <= '1';
myUnitLevels( 7) <= UnitIn( 3);
                    myUnitLevels( 8) <= '1';
myUnitLevels( 9) <= UnitIn( 4);
                    myUnitLevels(10) <= '1';
myUnitLevels(11) <= UnitIn( 5);
                    myUnitLevels(12) <= '1';
myUnitLevels(13) <= UnitIn( 6);
                    myUnitLevels(14) <= '1';
myUnitLevels(15) <= UnitIn( 7);
                else
                    myUnitLevels( 1) <= '1';
myUnitLevels( 0) <= UnitIn( 0);
                    myUnitLevels( 3) <= '1';
myUnitLevels( 2) <= UnitIn( 1);
                    myUnitLevels( 5) <= '1';
myUnitLevels( 4) <= UnitIn( 2);
                    myUnitLevels( 7) <= '1';
myUnitLevels( 6) <= UnitIn( 3);
                    myUnitLevels( 9) <= '1';
myUnitLevels( 9) <= UnitIn( 4);
                    myUnitLevels(11) <= '1';
myUnitLevels(10) <= UnitIn( 5);
                    myUnitLevels(13) <= '1';
myUnitLevels(12) <= UnitIn( 6);
```

```
                    myUnitLevels(15) <= '1';
myUnitLevels(14) <= UnitIn( 7);
                end if;

                -- Signal that state has ended
                myDone <= '1';
            end if; -- Run
        end if; -- Enable
    end process P_Simple2Complex;



    --
    -- Determine the current sign based on the limits
imposed by the zero crossings
    --
    -- Note: The enable sign will be ignored because
when the system would start it will
    --      have the current sign determined
    P_DetermineCurrentSign: process(Clock,
CurrentOutput, myCurrentSign)
    begin
        if (Clock = '1' and Clock'event) then
--          if (myCurrentSign = '0') then
                -- If the current sign is +, monitor
the current value. If the value
                -- reached the negative cross value,
the sign will become '-'
                if (CurrentOutput <
mycCurrentJumpLevel_Neg) and (CurrentOutput > "10" &
x"0000") then
                    myCurrentSign <= '1';
                end if;
--          else
                -- If the current sign is '-' then the
current is compared with the
                -- pozitive current limit
                if (CurrentOutput >
mycCurrentJumpLevel_Poz) and (CurrentOutput < "10" &
x"0000") then
                    myCurrentSign <= '0';
                end if;
--          end if;
        end if;
    end process P_DetermineCurrentSign;



    --
    -- Output data
    --
    UnitStateOut <= myUnitStates;
    UnitLevelOut <= myUnitLevels;

    Done <= myDone;

end Behavioral;
```

H.2.XIII UNITMAPPING.VHD

```
-------------------------------------------------------
---------------------------
-- Company: Aalborg University
-- Engineer: Sandu Cristian
--
-- Create Date:    17:49:49 12/08/2008
-- Design Name:
-- Module Name:    SimpleUnit2ComplexUnit - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-------------------------------------------------------
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity UnitMapping is
    Port (
        Clock: in std_logic;            -- Main system
clock
        Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

        Run: in std_logic;          -- If enable, it
converts the data input, if not, output the last data

        Done: out std_logic;            -- High when
the conversion is done

        -- Input data from Unit2IGBTs
        UnitStateIn : in   STD_LOGIC_VECTOR (23 downto
0);         -- States for 8 * 3 units
        UnitLevelIn : in   STD_LOGIC_VECTOR (47 downto
0);         -- Levels for 8 * 3 * 2 legs

        -- Output data from Unit2IGBTs
        UnitStateOut : out  STD_LOGIC_VECTOR (29 downto
0);     -- States for 6 * 5 units
        UnitLevelOut : out  STD_LOGIC_VECTOR (59 downto
0)          -- Levels for 6 * 5 * 2 legs
    );
end UnitMapping;

architecture Behavioral of UnitMapping is
    signal myUnitStates: std_logic_vector(29 downto 0)
:= (others => '0');
    signal myUnitLevels: std_logic_vector(59 downto 0)
:= (others => '0');

    signal myDone : std_logic := '0';

begin
    P_UnitMap: process(Clock, Enable, UnitStateIn,
UnitLevelIn)
    begin
        if (Enable = '1') then
            if (Clock = '1' and Clock'event) then
                if (Run = '1') then
                    myUnitStates(20) <= UnitStateIn(
0);    myUnitLevels(40) <= not UnitLevelIn( 0);
myUnitLevels(41) <= UnitLevelIn( 1);
                    myUnitStates(19) <= UnitStateIn(
1);    myUnitLevels(38) <= not UnitLevelIn( 2);
myUnitLevels(39) <= UnitLevelIn( 3);
                    myUnitStates(18) <= UnitStateIn(
2);    myUnitLevels(36) <= not UnitLevelIn( 4);
myUnitLevels(37) <= UnitLevelIn( 5);
                    myUnitStates(23) <= UnitStateIn(
3);    myUnitLevels(46) <= not UnitLevelIn( 6);
myUnitLevels(47) <= UnitLevelIn( 7);
                    myUnitStates(22) <= UnitStateIn(
4);    myUnitLevels(44) <= not UnitLevelIn( 8);
myUnitLevels(45) <= UnitLevelIn( 9);
                    myUnitStates(21) <= UnitStateIn(
5);    myUnitLevels(42) <= not UnitLevelIn(10);
myUnitLevels(43) <= UnitLevelIn(11);
                    myUnitStates(26) <= UnitStateIn(
6);    myUnitLevels(52) <= not UnitLevelIn(12);
myUnitLevels(53) <= UnitLevelIn(13);
                    myUnitStates(25) <= UnitStateIn(
7);    myUnitLevels(50) <= not UnitLevelIn(14);
myUnitLevels(51) <= UnitLevelIn(15);
```

```
                    myUnitStates( 2) <= UnitStateIn(
8);    myUnitLevels( 4) <= not UnitLevelIn(16);
myUnitLevels( 5) <= UnitLevelIn(17);
                    myUnitStates( 1) <= UnitStateIn(
9);    myUnitLevels( 2) <= not UnitLevelIn(18);
myUnitLevels( 3) <= UnitLevelIn(19);
                    myUnitStates( 0) <=
UnitStateIn(10);    myUnitLevels( 0) <= not
UnitLevelIn(20);    myUnitLevels( 1) <=
UnitLevelIn(21);
                    myUnitStates( 8) <=
UnitStateIn(11);    myUnitLevels(16) <= not
UnitLevelIn(22);    myUnitLevels(17) <=
UnitLevelIn(23);
                    myUnitStates( 7) <=
UnitStateIn(12);    myUnitLevels(14) <= not
UnitLevelIn(24);    myUnitLevels(15) <=
UnitLevelIn(25);
                    myUnitStates( 6) <=
UnitStateIn(13);    myUnitLevels(12) <= not
UnitLevelIn(26);    myUnitLevels(13) <=
UnitLevelIn(27);
                    myUnitStates(14) <=
UnitStateIn(14);    myUnitLevels(28) <= not
UnitLevelIn(28);    myUnitLevels(29) <=
UnitLevelIn(29);
                    myUnitStates(13) <=
UnitStateIn(15);    myUnitLevels(26) <= not
UnitLevelIn(30);    myUnitLevels(27) <=
UnitLevelIn(31);
                    myUnitStates( 5) <=
UnitStateIn(16);    myUnitLevels(10) <= not
UnitLevelIn(32);    myUnitLevels(11) <=
UnitLevelIn(33);
                    myUnitStates( 4) <=
UnitStateIn(17);    myUnitLevels( 8) <= not
UnitLevelIn(34);    myUnitLevels( 9) <=
UnitLevelIn(35);
                    myUnitStates( 3) <=
UnitStateIn(18);    myUnitLevels( 6) <= not
UnitLevelIn(36);    myUnitLevels( 7) <=
UnitLevelIn(37);
                    myUnitStates(11) <=
UnitStateIn(19);    myUnitLevels(22) <= not
UnitLevelIn(38);    myUnitLevels(23) <=
UnitLevelIn(39);
```

```
                    myUnitStates(10) <=
UnitStateIn(20);    myUnitLevels(20) <= not
UnitLevelIn(40);    myUnitLevels(21) <=
UnitLevelIn(41);
                    myUnitStates( 9) <=
UnitStateIn(21);    myUnitLevels(18) <= not
UnitLevelIn(42);    myUnitLevels(19) <=
UnitLevelIn(43);
                    myUnitStates(17) <=
UnitStateIn(22);    myUnitLevels(34) <= not
UnitLevelIn(44);    myUnitLevels(35) <=
UnitLevelIn(45);
                    myUnitStates(16) <=
UnitStateIn(23);    myUnitLevels(32) <= not
UnitLevelIn(46);    myUnitLevels(33) <=
UnitLevelIn(47);

                    -- Signal that state has ended
                    myDone <= '1';
                else -- Run is set to 0
                    myDone <= '0';

                    -- no change for output
                    null;
                end if;
            end if; -- Clock event (1)
        else
            myUnitStates <= (others => '0');
            myUnitLevels <= (others => '0');
            myDone <= '0';
        end if;
    end process P_UnitMap;


    --
    -- Output data
    --
    UnitStateOut <= myUnitStates;
    UnitLevelOut <= myUnitLevels;

    Done <= myDone;

end Behavioral;
```

H.2.XIV MAINSTATEMACHINE.VHD

```
-------------------------------------------------------
---------------------------
-- Company: Aalborg University
-- Engineer: Sandu Cristian
--
-- Create Date:    17:49:49 12/08/2008
-- Design Name:
-- Module Name:    SimpleUnit2ComplexUnit - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-------------------------------------------------------
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity MainControl is
    Port (
        Clock: in std_logic;            -- Main system
clock
```

```
        Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

        Run: in std_logic;          -- If enable, it
converts the data input, if not, output the last data

        Done: out std_logic;            -- High when
the conversion is done

        -- Input data
        MethodSelection: in STD_LOGIC_VECTOR(2 downto
0);     -- Method selection

        -- Input data
        ReferenceU: in STD_LOGIC_VECTOR(23 downto 0);
        ReferenceV: in STD_LOGIC_VECTOR(23 downto 0);
        ReferenceW: in STD_LOGIC_VECTOR(23 downto 0);

        ReferenceCounterMax : in  STD_LOGIC_VECTOR (23
downto 0);   -- The maximum value for the counters

        -- Measured values
        UnitVotlagesU: in STD_LOGIC_VECTOR(143 downto
0);

        UnitVotlagesV: in STD_LOGIC_VECTOR(143 downto
0);

        UnitVotlagesW: in STD_LOGIC_VECTOR(143 downto
0);

        -- Current input values
        CurrentOutputU: in STD_LOGIC_VECTOR(17 downto
0);
```

```
        CurrentOutputV: in STD_LOGIC_VECTOR(17 downto
0);
        CurrentOutputW: in STD_LOGIC_VECTOR(17 downto
0);

        -- Voltage level definition
        VoltageLevels: in STD_LOGIC_VECTOR(71 downto
0);

        -- Output data
        UnitStatesU : out  STD_LOGIC_VECTOR (7 downto
0);
        UnitStatesV : out  STD_LOGIC_VECTOR (7 downto
0);
        UnitStatesW : out  STD_LOGIC_VECTOR (7 downto
0);


        -- Output data
        UnitStateOut : out  STD_LOGIC_VECTOR (29 downto
0);     -- States for 8 * 3 units
        UnitLevelOut : out  STD_LOGIC_VECTOR (59 downto
0)          -- Levels for 8 * 3 * 2 legs
);
end MainControl;




architecture Behavioral of MainControl is
    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  State machine variables
    --
    --
    --
    --

    type TStateType is (State_DoControl,
State_DoUnitConversion, State_DoMapping,
State_DoTransmit, State_Idle);
    signal myCurrentState, myNextState : TStateType;

    signal myDone : std_logic := '0';

    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Control parameters
    --
    --
    --
    --
    component Modulation
        Port (
            Clock: in std_logic;            -- Main
system clock
            Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

            -- State machine parameters
            Run: in std_logic;          -- If enable,
it converts the data input, if not, output the last
data
            Done: out std_logic;            -- High
when the conversion is done

            MethodSelection: in STD_LOGIC_VECTOR(2
downto 0);       -- Method selection

            -- Input data
```

```
            ReferenceU: in STD_LOGIC_VECTOR(23 downto
0);
            ReferenceV: in STD_LOGIC_VECTOR(23 downto
0);
            ReferenceW: in STD_LOGIC_VECTOR(23 downto
0);

            ReferenceCounterMax : in  STD_LOGIC_VECTOR
(23 downto 0);   -- The maximum value for the counters

            -- Measured values
            UnitVotlagesU: in STD_LOGIC_VECTOR(143
downto 0);
            UnitVotlagesV: in STD_LOGIC_VECTOR(143
downto 0);
            UnitVotlagesW: in STD_LOGIC_VECTOR(143
downto 0);

            -- Current input values
            CurrentOutputU: in STD_LOGIC_VECTOR(17
downto 0);
            CurrentOutputV: in STD_LOGIC_VECTOR(17
downto 0);
            CurrentOutputW: in STD_LOGIC_VECTOR(17
downto 0);

            -- Voltage level definition
            VoltageLevels: in STD_LOGIC_VECTOR(71
downto 0);

            -- Output data
            UnitStatesU : out  STD_LOGIC_VECTOR (7
downto 0);
            UnitStatesV : out  STD_LOGIC_VECTOR (7
downto 0);
            UnitStatesW : out  STD_LOGIC_VECTOR (7
downto 0)
        );
    end component;
    signal myControlRun: std_logic := '0';
    signal myControlDone: std_logic := '0';

    signal myControlUnitStatesU: std_logic_vector(7
downto 0);
    signal myControlUnitStatesV: std_logic_vector(7
downto 0);
    signal myControlUnitStatesW: std_logic_vector(7
downto 0);




    ---------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Unit Conversion parameters
    --
    --
    --
    --

    component Unit2Igbts
        Port (
            Clock: in std_logic;            -- Main
system clock
            Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

            Run: in std_logic;          -- If enable,
it converts the data input, if not, output the last
data

            Done: out std_logic;            -- High
when the conversion is done

            -- Input data
            UnitIn : in  STD_LOGIC_VECTOR ( 7 downto
0);
```

```
            CurrentOutput: in STD_LOGIC_VECTOR(17
downto 0);

            -- Output data
            UnitStateOut : out  STD_LOGIC_VECTOR ( 7
downto 0);     -- States for 8 * 3 units
            UnitLevelOut : out  STD_LOGIC_VECTOR (15
downto 0)          -- Levels for 8 * 3 * 2 legs
        );
    end component;

    signal myConvDoneU: std_logic := '0';
    signal myConvDoneV: std_logic := '0';
    signal myConvDoneW: std_logic := '0';
    signal myConvRun: std_logic := '0';

    signal myConvUnitStatesU: std_logic_vector( 7
downto 0) := (others => '0');
    signal myConvUnitLevelsU: std_logic_vector(15
downto 0) := (others => '0');
    signal myConvUnitStatesV: std_logic_vector( 7
downto 0) := (others => '0');
    signal myConvUnitLevelsV: std_logic_vector(15
downto 0) := (others => '0');
    signal myConvUnitStatesW: std_logic_vector( 7
downto 0) := (others => '0');
    signal myConvUnitLevelsW: std_logic_vector(15
downto 0) := (others => '0');


    -------------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Unit Mapping parameters
    --
    --
    --
    --

    component UnitMapping
        Port (
            Clock: in std_logic;            -- Main
system clock
            Enable: in std_logic;       -- Enable the
conversion (if disable, set to 0 all outputs)

            Run: in std_logic;          -- If enable,
it converts the data input, if not, output the last
data

            Done: out std_logic;            -- High
when the conversion is done

            -- Input data from Unit2IGBTs
            UnitStateIn : in   STD_LOGIC_VECTOR (23
downto 0);          -- States for 8 * 3 units
            UnitLevelIn : in   STD_LOGIC_VECTOR (47
downto 0);          -- Levels for 8 * 3 * 2 legs

            -- Output data from Unit2IGBTs
            UnitStateOut : out  STD_LOGIC_VECTOR (29
downto 0);     -- States for 6 * 5 units
            UnitLevelOut : out  STD_LOGIC_VECTOR (59
downto 0)          -- Levels for 6 * 5 * 2 legs
        );
    end component;

    signal myMapDone: std_logic := '0';
    signal myMapRun: std_logic := '0';

    signal myMapUnitStates: std_logic_vector(29 downto
0) := (others => '0');
    signal myMapUnitLevels: std_logic_vector(59 downto
0) := (others => '0');
```

```
    -------------------------------------------------------
-------------------
    --
    --
    --
    --
    --
    --
    --                  Transmit parameters
    --
    --
    --
    --


    signal myTransmitRun: std_logic := '0';
    signal myTransmitDone: std_logic := '0';


begin

   SYNC_PROC: process (Clock, Run)
   begin
        if (Run = '1') then
            if (Clock'event and Clock = '1') then
            myCurrentState <= myNextState;
         end if;
        else    -- Enable = 0
            myCurrentState <= State_Idle;
      end if;
   end process;


   OE_DoControl: process (myCurrentState)
   begin
      --insert statements to decode internal output
signals
      --below is simple example
      if myCurrentState = State_DoControl then
         myControlRun <= '1';
      else
         myControlRun <= '0';
      end if;
   end process;


    OE_DoConversion: process (myCurrentState)
   begin
      --insert statements to decode internal output
signals
      --below is simple example
      if myCurrentState = State_DoUnitConversion then
         myConvRun <= '1';
      else
         myConvRun <= '0';
      end if;
   end process;


    OE_DoMap: process (myCurrentState)
   begin
      --insert statements to decode internal output
signals
      --below is simple example
      if myCurrentState = State_DoMapping then
         myMapRun <= '1';
      else
         myMapRun <= '0';
      end if;
   end process;


    OE_DoTransmit: process (myCurrentState)
   begin
      --insert statements to decode internal output
signals
      --below is simple example
      if myCurrentState = State_DoTransmit then
         myTransmitRun <= '1';
      else
         myTransmitRun <= '0';
      end if;
   end process;
   NEXT_STATE_DECODE: process (myCurrentState,
myNextState, myControlDone, myConvDoneU, myConvDoneV,
myConvDoneW, myMapDone, myTransmitDone)
   begin
      --declare default state for next_state to avoid
latches
      myNextState <= myCurrentState;

      --insert statements to decode next_state
      case (myCurrentState) is
         when State_DoControl =>
            if myControlDone = '1' then
               myNextState <= State_DoUnitConversion;
            end if;
         when State_DoUnitConversion =>
            if myConvDoneU = '1' and myConvDoneV = '1'
and myConvDoneW = '1' then
               myNextState <= State_DoMapping;
            end if;
         when State_DoMapping =>
            if myMapDone = '1' then
               myNextState <= State_DoTransmit;
            end if;
         when State_DoTransmit =>
            if myTransmitDone = '1' then
               myNextState <= State_Idle;
            end if;
         when others =>
            myNextState <= State_Idle;
      end case;
   end process;


    ---------------------------------------------------
--------------------------------
    --
    --
    --
    --
    --
    --
    --             Modulation
    --
    --
    --
    --
    --
    ---------------------------------------------------
--------------------------------

    myModulation: Modulation port map(
        Clock => Clock,
        Enable => Enable,

        Run => myControlRun,
        Done => myControlDone,

        MethodSelection => MethodSelection,

        ReferenceU => ReferenceU,
        ReferenceV => ReferenceV,
        ReferenceW => ReferenceW,

        ReferenceCounterMax => ReferenceCounterMax,

        UnitVotlagesU => UnitVotlagesU,
        UnitVotlagesV => UnitVotlagesV,
        UnitVotlagesW => UnitVotlagesW,

        CurrentOutputU => CurrentOutputU,
        CurrentOutputV => CurrentOutputV,
        CurrentOutputW => CurrentOutputW,

        VoltageLevels => VoltageLevels,

        UnitStatesU => myControlUnitStatesU,
        UnitStatesV => myControlUnitStatesV,
        UnitStatesW => myControlUnitStatesW
    );

    ---------------------------------------------------
--------------------------------
    --
    --
    --
    --
    --
    --
    --             Unit conversion  (Unit 2 IGBTs)
    --
    --
    --
    --
    --
    ---------------------------------------------------
--------------------------------

    myUnit2Igbt_U: Unit2Igbts port map(
        Clock => Clock,
        Enable => Enable,

        Run => myConvRun,

        Done => myConvDoneU,

        UnitIn => myControlUnitStatesU,

        CurrentOutput => CurrentOutputU,

        UnitStateOut => myConvUnitStatesU,
        UnitLevelOut => myConvUnitLevelsU
    );

    myUnit2Igbt_V: Unit2Igbts port map(
        Clock => Clock,
        Enable => Enable,

        Run => myConvRun,

        Done => myConvDoneV,

        UnitIn => myControlUnitStatesV,

        CurrentOutput => CurrentOutputV,

        UnitStateOut => myConvUnitStatesV,
        UnitLevelOut => myConvUnitLevelsV
    );

    myUnit2Igbt_W: Unit2Igbts port map(
        Clock => Clock,
        Enable => Enable,

        Run => myConvRun,

        Done => myConvDoneW,

        UnitIn => myControlUnitStatesW,

        CurrentOutput => CurrentOutputW,

        UnitStateOut => myConvUnitStatesW,
        UnitLevelOut => myConvUnitLevelsW
    );


    ---------------------------------------------------
--------------------------------
    --
    --
    --
    --
    --
    --
    --             Unit mappings
    --
    --
    --
    --
    --
    ---------------------------------------------------
--------------------------------

    myUnitMapping: UnitMapping port map(
        Clock => Clock,
        Enable => Enable,

        Run => myMapRun,

        Done => myMapDone,

        UnitStateIn( 7 downto  0) => myConvUnitStatesU,
        UnitStateIn(15 downto  8) => myConvUnitStatesV,
        UnitStateIn(23 downto 16) => myConvUnitStatesW,

        UnitLevelIn(15 downto  0) => myConvUnitLevelsU,
        UnitLevelIn(31 downto 16) => myConvUnitLevelsV,
        UnitLevelIn(47 downto 32) => myConvUnitLevelsW,

        UnitStateOut => myMapUnitStates,
        UnitLevelOut => myMapUnitLevels
    );


    ---------------------------------------------------
----------------------------------------
    --
    --
    --
    --         # Set done output to tru
    --
    --
    --
    --
    --
    --
    ---------------------------------------------------
----------------------------------------
    P_SetDoneFlag: process (Enable, myDone, myMapDone)
    begin
        if (Enable = '1') then
            myDone <= myMapDone;
        else
            myDone <= '0';
        end if;
    end process P_SetDoneFlag;



    --
    -- Output data
    --
    UnitStateOut <= myMapUnitStates;
    UnitLevelOut <= myMapUnitLevels;


    UnitStatesU <= myControlUnitStatesU;
    UnitStatesV <= myControlUnitStatesV;
    UnitStatesW <= myControlUnitStatesW;

    Done <= myDone and myMapDone;

end Behavioral;
```

APPENDIX I. SWITCH MODE POWER SUPPLY

I.1.I CALCULATION AND DESIGN CONSIDERATIONS

To determine the right components for the supply calculations were made. The make an appropriate design steps have to be followed so a standard design was considered. The main steps that have to be followed:

- Power calculation to determine the proper switch unit.
- Transformer design
- The basic parameters for control
- Filter design

For the calculation the forward diode voltage was considered to be:

$$V_{dfw}=0.6\ [V]$$

Equation I.1 Forward diode voltage

Before any converter calculation is to be made, the total minimum and maximum output power of the converter will be calculated asP_{0min} and $P_{0\max}$:

$$P_{0\min} = (V_{s1} + V_{dfw})\, I_{01\min} + (V_{s2} + V_{dfw})\, I_{02\min} + (V_{s3} + V_{dfw})\, I_{03\min} + (V_{s4} + V_{dfw})\, I_{04\max}$$

$$P_{0\min} = 11.6\ [W]$$

Where:

- P_{omin} is the minimum output power

$$P_{0\max} = (V_{s1} + V_{dfw})\, I_{01\max} + (V_{s2} + V_{dfw})\, I_{02\max} + (V_{s3} + V_{dfw})\, I_{03\max} + (V_{s4} + V_{dfw})\, I_{04\max}$$

$$P_{0\max} = 84.4\ [W]$$

Equation I.2 The SWMP minimum and maximum power rating

Where:

- P_{omax} is the maximum output power

Calculate the total power :

$$P_{tot} = P_{0\max}(\frac{1}{\eta} + 1) = 84.4\left(\frac{1}{0.95} + 1\right) = 173.24[W]$$

Chosen frequency $f_{sw} = 75\ [KHz]$

$$\Rightarrow T = \frac{1}{f_{sw}} = 13.3\ [\mu s]$$

Equation I.3 Total switching period

Where:

- f_{sw} is the switching frequency
- T is the period

Transformer Efficiency $\eta = 0.95$

Calculate the electrical coefficient :

$$K_e = 0.145 \cdot (K_f)^2 \cdot (f_{sw})^2 \cdot (B_m)^2 \cdot 10^4$$

$$K_e = 0.145 \cdot (4)^2 \cdot (73 \cdot 10^3)^2 \cdot (0.2)^2 \cdot 10^4 = 52200$$

Where:

- $K_f = 4$ due to square wave input signal $K_f = 4.44$ due to sine wave
- f_{sw} is the switching frequency
- $B_m = 0.2$ is the flux density [Tesla]

Calculate the geometry coefficient :

$$K_g = \frac{P_{tot}}{2 \cdot K_e \cdot \alpha} = \frac{173.24}{2 \cdot 52200 \cdot 0.5} = 0.0331\ [cm^5]$$

Where:

- K_e is the electrical coefficient
- P_{tot} is the total power
- α is the regulation factor [%]

The leakage coefficient of the transformer: K=0.95

The total energy stored in the transformer will be :

$$W_{tot} = \frac{1}{K} = 1.052\ [J]$$

$$W_{flyback} = \frac{W_{tot}\, P_{0\max}}{f_{sw}} = 1.183 \cdot 10^{-3}\ [J]$$

Equation I.4 Total energy stored in the transformer

Where:

- W_{tot} total energy in the transformer
- $W_{flyback}$total energy from the converter

Based on the total amount of power the core of the transformer can be chosen and according to calculated geometry coefficient and frequency and energy the resulting core is (2): ETD-34

The core has the following characteristics:

- $A_t = 36.8\ [cm^2]$ surface area of the transformer
- $A_p = 0.716[cm^4]$ area product
- $W_a = 1.19\ [cm^2]$ window area
- $A_c = 0.6\ [cm^2]$ effective iron area
- $MLT = 5.2\ [cm]$ mean length turn
- $W_{TFe} = 22\ [grams]$ iron weight
- $W_{tcu} = 22.226\ [grams]$ copper weight
- $L_T = 3[cm]$ total length
- $W_T = 2[cm]$ total width
- $H_T = 3[cm]$ total height
- $MPL = 6.7[cm]$ magnetic path length
- $G = 1.94[cm]$ window length
- $F = 0.615[cm]$ window width

Maximum switching stress of the mosfet: $K_{fb} = 0.8$

$$V_{fm} = K_{fb} \cdot V_{inmin} = 480\ [V]$$

Where:

- V_{fm} is the min voltage
- K_{fb} switching stress
- V_{inmin} is the min input voltage

Maximum switching voltage of the mosfet:

$$V_{ds} = (F_{spike} + 1)(V_{inmax} + V_{fm})$$

$$V_{ds} = 1792\ [V]$$

Equation I.5 The voltage drop on the MOSFET

Where:

- V_{ds} is the voltage drain to source
- V_{inmax} is the max input voltage
- F_{spike} safe factor

Calculation of max duty cycle $V_{out} = DV_{in}$

$$T_{on} + T_r + T_{dt} = T = \frac{1}{f_{sw}} \text{ the chosen } D_{dt} = 0.1$$

Equation I.6 ON state period for MOSFET

Where:

- T_{on} period of time when the MOSFET is in conduction

- T_r recovery time
- T_{dt} dead time

$$R_{ds} = 1.8\ [\Omega]$$

Equation I.7 Drain to source resistance

Where:

- R_{ds} drain to source resistance

$$V_{dson} = \frac{P_{outmax}}{\eta \cdot V_{inmin}} R_{dson}$$

Where:

- V_{dson} the voltage drop when the switch is in conduction

$$V_{dson} = \frac{84.4}{0.95 \cdot 800} 1.8 = 0.266\ [V]$$

$$V_{fly} = N_{s1}\left(V_{s1} + V_{dfw}\right)$$

$$V_{fly} = 30.76(15 + 0.6)$$

$$V_{fly} = 480\ [V]$$

Where:

- V_{fly} voltage in the converter

$$T_{on}max = \frac{V_{fly}(1 - D_{dt})T}{(V_{inmin} - V_{ds})K + V_{fly}}$$

$$T_{on}max = \frac{479.85(1 - 0.1)1.33 \cdot 10^{-5}}{(600 - 0.266)0.95 + 479.85}$$

$$T_{on}max = 5.47\ [\mu s]$$

Equation I.8 The maximum ON period for the switch

Where:

- $T_{on}max$ the max period of time when the switch can be on

$$T_{on}min = \frac{V_{fly}(1 - D_{dt})T}{(V_{inmax} - V_{ds})K + V_{fly}}$$

$$T_{on}min = \frac{479.85(1 - 0.1)1.33 \cdot 10^{-5}}{(600 - 0.266)0.95 + 479.85}$$

$$T_{on}min = 4.633\ [\mu s]$$

Equation I.9 The minimum ON period for the switch

Where:

- $T_{on}min$ the min period of time when the switch can be on

Maximum duty cycle: $D_{max} = \frac{T_{on}max}{T} = 0.83$

Minimum duty cycle: $D_{min} = \frac{T_{on}min}{T} = 0.384$

Primary peak current : $I_{pk} = \frac{2W_{fly} \cdot f_{sw}}{V_{in}\min \cdot D_{max}}$

$$I_{pk} = \frac{2 \cdot 1.183 \cdot 10^{-3} \cdot 75 \cdot 10^{3}}{600 \cdot 0.411}$$

$$I_{pk} = 0.8\ [A]$$

Equation I.10 Primary peak current

<u>Primary RMS current</u> : $I_{prms} = \frac{I_{pk}}{\sqrt{3}}\sqrt{\frac{T_{on}max}{T}}$

$$I_{prms} = \frac{0.719}{\sqrt{3}}\sqrt{\frac{5.47 \cdot 10^{-6}}{1.33 \cdot 10^{-5}}}$$

Equation I.11 Primary RMS current

$I_{prms} = 0.266\ [A]$ the chosen wire type is $A_{WG}\#35$ with $d_1 \geq 0.14\ [mm]$

<u>Primary DC current</u> : $I_{pdc} = \frac{P_{outmax}}{V_{inmin} \cdot \eta}$

$$I_{pdc} = \frac{84.4}{600 \cdot 0.411}$$

$$I_{pdc} = 0.2\ [A]$$

Equation I.12 Primary DC current

Calculation of number of turns necessary for the primary winding:

$$N_p = \frac{V_p \cdot 10^4}{K_f \cdot K_u \cdot B_m \cdot f_{sw} \cdot A_p} = \frac{173.24 \cdot 10^4}{4 \cdot 0.4 \cdot 0.2 \cdot 75 \cdot 10^3 \cdot 0.7}$$

$$N_p = 250\ [turns]$$

Equation I.13 Primary number of turns

Where:

- $K_f = 4$ due to square wave input signal $K_f = 4.44$ due to sine wave
- K_u window utilization factor
- f_{sw} is the switching frequency
- $B_m = 0.2$ is the flux density [Tesla]
- A_c core area

Calculation of current density for primary:

$$J = \frac{P_{tot} \cdot 10^4}{K_f \cdot K_u \cdot B_m \cdot f_{sw} \cdot A_p} = \frac{173.24 \cdot 10^4}{4 \cdot 0.4 \cdot 0.2 \cdot 75 \cdot 10^3 \cdot 0.176}$$

$$J = 101 \, [\frac{A}{cm^2}]$$

Equation I.14 Current density

Where:

- $K_f = 4$ due to square wave input signal $K_f = 4.44$ due to sine wave
- K_u window utilization factor
- f_{sw} is the switching frequency
- $B_m = 0.2$ is the flux density [Tesla]
- A_p area product

Calculate the primary wire area :

$$A_{wp} = \frac{I_{prms}}{J} = \frac{0.716}{100.81} = 0.0174 \, [cm^2]$$

Equation I.15 Cross section of the wire

The chosen wire for primary is: $AWG\#$ 14 with area a=0.02002 [cm^2] with $\frac{\mu\Omega}{cm} = 82.8$

Calculation of primary inductance:

$$L_p = \frac{2 \cdot W_{fly}}{I_p^2}$$

$$L_p = \frac{2 \cdot 1.183 \cdot 10^{-3}}{0.719^2}$$

$$L_p = 4.5 \, [mH]$$

Equation I.16 Primary inductance

Calculation of primary resistance:

$$R_p = (MLT) \cdot (N_p) \cdot \frac{\mu\Omega}{cm} \cdot 10^{-6} = (5.2) \cdot (191) \cdot 82.8 \cdot 10^{-6} = 0.082 \, [\Omega]$$

Equation I.17 Primary resistance

Calculation of primary copper loss:

$$P_p = (I_p)^2 \cdot (R_p) = (0.11)^2 \cdot 0.082 = 9.92 \cdot 10^{-4} \, [W]$$

Equation I.18 Losses for primary

Determine the second number of turns :

$$N_{s1} = \frac{V_{s1} + V_{dfw}}{V_{fly}} = \frac{15 + 0.6}{479.85} \Longrightarrow N_{s1} = 4.8\ turns$$

$$N_{s2} = \frac{V_{s2} + V_{dfw}}{V_{fly}} = \frac{12 + 0.6}{479.85} \Longrightarrow N_{s2} = 3.9\ turns$$

$$N_{s3} = \frac{V_{s3} + V_{dfw}}{V_{fly}} = \frac{5 + 0.6}{479.85} \Longrightarrow N_{s3} = 1.9\ turns$$

$$N_{s4} = \frac{V_{s4} + V_{dfw}}{V_{fly}} = \frac{12 + 0.6}{479.85} \Longrightarrow N_{s4} = 3.9\ turns$$

Equation I.19 The number of turns for the secondary windings

First slave output :

$$I_{s1pk} = \frac{2 \cdot I_{s1max}}{1 - D_{max} - D_{dt}}$$

$$I_{s1pk} = \frac{2 \cdot 2}{1 - 0.411 - 0.1}$$

Equation I.20 Secondary 1 – Peak current

$I_{s1pk} = 8.17\ [A]$ the chosen wire type and size A_{WG}#35 with $d_1 \geq 0.14\ [mm]$

RMS current : $I_{s1rms} = \frac{I_{s1pk}}{\sqrt{3}}\sqrt{1 - D_{\max} - D_{dt}}$

$$I_{s1rms} = \frac{I_{s1pk}}{\sqrt{3}}\sqrt{1 - 0.411 - 0.1}$$

$$I_{s1rms} = 3.29\ [A]$$

Equation I.21 Secondary 2 – RMS current

$$L_{s1} = N_{s1}^2 \cdot L_p = 4.75\ [\mu H]$$

Equation I.22 Inductance for the first output

Second slave output

$$I_{s2pk} = \frac{2 \cdot I_{s2max}}{1 - D_{max} - D_{dt}}$$

$$I_{s2pk} = \frac{2 \cdot 2}{1 - 0.411 - 0.1}$$

Equation I.23 Secondary 2 – Peak current

$I_{s2pk} = 4.08\ [A]$ the chosen wire type and size A_{WG}#38 with $d_2 \geq 0.12\ [mm]$

RMS current : $I_{s2rms} = \frac{I_{s2pk}}{\sqrt{3}}\sqrt{1 - D_{\max} - D_{dt}}$

$$I_{s2rms} = \frac{4.08}{\sqrt{3}}\sqrt{1 - 0.411 - 0.1}$$

$$I_{s2rms} = 1.64\ [A]$$

Equation I.24 Secondary 2 – RMS current

$$L_{s2} = N_{s2}^2 \cdot L_p = 6.05\ [\mu H]$$

Equation I.25 Secondary 2 – Inductance

<u>3rd slave output:</u>

$$I_{s3pk} = \frac{2 \cdot I_{s3max}}{1 - D_{max} - D_{dt}}$$

$$I_{s3pk} = \frac{2 \cdot 5}{1 - 0.411 - 0.1}$$

Equation I.26 Secondary 3 – Peak current

$I_{s3pk} = 20.44\ [A]$ the chosen wire type and size A_{WG}#14 with $d_3 \geq 1.7\ [mm]$

RMS current : $I_{s3rms} = \frac{I_{s3pk}}{\sqrt{3}}\sqrt{1 - D_{\max} - D_{dt}}$

$$I_{s3rms} = \frac{20.44}{\sqrt{3}}\sqrt{1 - 0.411 - 0.1}$$

$$I_{s3rms} = 8.25\ [A]$$

Equation I.27 Secondary 3 – RMS current

$$L_{s3} = N_{s3}^2 \cdot L_p = 30.8\ [\mu H]$$

Equation I.28 Secondary 3 – Inductance

<u>4rd slave output:</u>

$$I_{s4pk} = \frac{2 \cdot I_{s4max}}{1 - D_{max} - D_{dt}}$$

$$I_{s4pk} = \frac{2 \cdot 1}{1 - 0.411 - 0.1}$$

Equation I.29 Secondary 4 – Peak current

$I_{s4pk} = 4.08\ [A]$ the chosen wire type and size $A_{WG}\#38$ with $d_3 \geq 0.12\ [mm]$

RMS current : $$I_{s4rms} = \frac{I_{s4pk}}{\sqrt{3}}\sqrt{1 - D_{\max} - D_{dt}}$$

$$I_{s4rms} = \frac{4.08}{\sqrt{3}}\sqrt{1 - 0.411 - 0.1}$$

$$I_{s4rms} = 1.64\ [A]$$

Equation I.30 Secondary 4 – RMS current

$$L_{s4} = N_{s4}^2 \cdot L_p = 6.05\ [\mu H]$$

Equation I.31 Secondary 4 – Inductance

I.1.II CAPACITIVE FILTER

The capacitive filter is required in order to provide a smooth voltage at the output for each output of the inverter. The capacitive filter was sized with respect to the maximum duty cycle, duty cycle reached when the DC bus voltage is at the lowest value.

$$C_{filter} = I_{spk}\frac{T_{on}max}{V_{rp} \cdot 0.25} \Longrightarrow C_{1filter} = I_{s1pk}\frac{T_{on}max}{V_{rp} \cdot 0.25} = 575.63[\mu F]$$

$$C_{2filter} = I_{s2pk}\frac{T_{on}max}{V_{rp} \cdot 0.25} = 505.613[\mu F]$$

$$C_{3filter} = I_{s3pk}\frac{T_{on}max}{V_{rp} \cdot 0.25} = 2.54 \cdot 10^3[\mu F]$$

$$C_{4filter} = I_{s4pk}\frac{T_{on}max}{V_{rp} \cdot 0.25} = 505.613[\mu F]$$

Due to standard values of the capacitors, the following capacitors have been used:

$$C_{filter} = C_{2filter} = C_{4filter} = 1[mF]$$

$$C_{3filter} = 3.3[mF]$$