December 2017

Manual for Automation of Dc-microgrid Component Using Matlab/Simulink and FPGA's

Kavya Shree Kumar
University of Wisconsin-Milwaukee

MANUAL FOR AUTOMATION OF DC-MICROGRID COMPONENT USING MATLAB/SIMULINK AND FPGA'S

by

Kavya Shree Kumar

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Engineering

at

The University of Wisconsin- Milwaukee

December 2017

ABSTRACT

MANUAL FOR AUTOMATION OF DC-MICROGRID COMPONENT USING MATLAB/SIMULINK AND FPGA'S

by

Kavya Shree Kumar

The University of Wisconsin-Milwaukee, 2017
Under the Supervision of Professor Robert M Cuzner

Solar Energy is one of the abundantly available renewable energy source. Solar panels are semiconductor materials which capture the solar energy from every band in the visible light spectrum, infrared spectrum and ultra violet spectrum and converts it into electrical energy.
The DC community microgrid is used to supplement utility electrical power supplied to the neighbored with renewable sources such as solar panels, emergency back-up power through batteries or generators. Smart Cloud Interconnected environment increases the standard of living and facilitates ease to rectify faults, debug components and reinstate or replace obsolete components with newer ones.

Automation of the DC microgrid components provides a simple yet efficient way to connect to the grid and to every component in the grid remotely. It is essential to find the node of failure in the grid for technicians and engineers to work on and to debug the issue to facilitate smooth running of the grid without shutdown. FPGAs are used as target devices for end synthesis of the model that is created on Simulink. These FPGAs are links between cloud and power electronics components. To utilize the energy resource efficiently we need to monitor the input and output

of every component at every node in the grid. Simulating models on Simulink will let us connect the component and test engineer to the grid to detect any flaws or failures on time. FPGAs are easily reprogrammable and have long life with excellent capability to withstand stress.

This thesis report provides a set of procedures to create and simulate a real time component model and to generate HDL files to build a clean code which can be redeployed on target FPGAs.

To

my parents,

my brother,

and

Prof Jim

Cummins

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I thank Professor Robert M Cuzner for his support both at a personal level and for carrying out this research project. I thank Professor Jim Cummins for providing valuable insights and feedbacks and mostly for being an inspiration. I thank Professor Devendra K Misra and Professor Guangwu Xu for being a crucial part of my thesis defence. I thank Mr. Rasoul Hosseni, Mr. Rounak Siddaiah for their extended help throughout the project. I thank Mr. Vishak Aprameya Shivakumar Kanakapura and Mr. Arun Kumar Koralagundi Matt for proof-reading and supporting this work. Importantly, I thank University of Wisconsin-Milwaukee.

I thank my parents and my brother for their invaluable motivation and support.

CHAPTER 1: INTRODUCTION

Solar panels convert the sun's light in to usable solar energy using N-type and P-type semiconductor material. When sunlight is absorbed by these materials, the solar energy knocks electrons loose from their atoms, allowing the electrons to flow through the material to produce electricity. This process of converting light (photons) to electricity (voltage) is called the photovoltaic (PV) effect. Currently solar panels convert most of the visible light spectrum and about half of the ultraviolet and infrared light spectrum to usable solar energy. Solar energy technologies use the sun's energy and light to provide heat, light, hot water, electricity, electricity to run every appliance at home including cooling.

Edison defended that the world should run off direct current (DC) electricity. The modern conception is quite like what Edison had in mind, but with the added benefit that the local DC grid can operate either in parallel with the surrounding AC grid or in isolation from it. DC microgrids, with their much smaller footprints, avoid most transmission and distribution losses. They also eliminate the waste of energy associated with the conversion of AC to DC, which is required for so many of the electrical loads found these days: LED lights, variable-speed motors, computers, televisions, and countless other forms of consumer electronics-loads that account for a steadily increasing fraction of the electricity consumed. (Some estimates suggest that most of the electricity used in commercial buildings already goes to serve such loads.) In a time of increased concern about energy efficiency and carbon-dioxide emissions, DC microgrids obviously have a great deal to offer. Yet traditional AC distribution systems are still all most people ever think about when you use the term “power grid.”

Load	Conventional household loads					
	Service Voltage	*Service Feed (A)*	*Peak Load (A)*	P_{avg} *(kW)*	*Usage per Month (Hr)*	*Monthly KW-Hr*
A/C	230	20	16	3.6	360	1296
Range	230	30	22	5	100	500
Water Heater	230	30	15	3.5	120	420
Dryer	230	30	22	5	20	100
Washer	115	10	4.3	0.5	10	5
Furnace*	115		2.3	0.26	0	0
Dishwasher	115	20	11.3	1.3	10	13
Refrigerator	***115***	***20***	***4.3***	***0.5***	***150***	75
Freezer	115	20	3	0.35	60	21
Microwave	***115***	***20***	***8.7***	***1.0***	***5***	5
Level 1 Outlets	***115***	***30***	***17***	0.8/ ***0.5***	***100***	***83***
Level 1 Outlets	***115***	***15***	***9***	0.5/ ***0.1***	***100***	***53***
Lighting	***115***	***15***	***6***	***0.6***	***100***	***57.6***
Ceiling Fans	115	10	0.5	0.06	200	12
Total Month Usage	**2641 kW-hr**					
Power Bill	**$309.21**					

Load	Mixed AC and DC Fed Appliances (DC loads are highlighted)					
	Service Voltage	*380V Service Feed (A)*	*380V Peak Load (A)*	P_{avg} *(kW)*	*Usage per Month (Hr)*	*Monthly KW-Hr*
A/C	380	10	2	0.76	360	272
Range	230	30	22	5	100	500
Water Heater	48	10	4	1.5	120	185
Dryer	65	10	7	2.45	20	49
Washer	65	5	1	0.3	10	3
Furnace*	65		0.5	0.15	0	0
Dishwasher	65		2	0.51	10	5.1
Refrigerator	***48***	5	***1***	***0.18***	***150***	***27***
Freezer	48		0.5	0.13	60	7.6
Microwave	***48***		***2***	***0.9***	***5***	***4.5***
Ceiling Fans	48		0.3	0.12	200	24.12
Outlets	115	30	17	0.8	100	83
Native DC Loads	***12***	***5***	***0.9***	0.3/ ***0.1***	***100***	***30.5***
Lighting	***12***		***0.3***	0.12 /***0.1***	***100***	***11.5***
Total AC Usage per Month	**540 kW-hr**					
Power Bill	**$63.23**					

Figure 1.1 Comparison between power usage, monthly estimate etc. of AC grid and DC grid

DC microgrids hold great promise, too, for ordinary residential and commercial buildings, where they could service the many electrical loads that use DC. These include LED lighting and, increasingly, charging stations for electric vehicles, whose hefty batteries demand (or produce) DC. Heating, ventilation, and air-conditioning (HVAC) equipment and various household appliances are also well suited to powering with DC. That's because the most energy-efficient types of HVAC equipment and appliances incorporate variable-speed motor drives, for which AC power from the regional grid must be converted to DC internally. So, it would be straightforward and more efficient to power such motors directly from a local DC source.

Community microgrids have emerged as an alternative to address the rising societal demands for electric infrastructures that are able to provide premium reliability and power quality levels while at the same time being economically and environmentally friendly. Community microgrids aim primarily at supplying electricity to a group of consumers in a neighborhood or several connected neighborhoods near Community microgrids enable unique opportunities for every day consumers to take advantage of renewable energy resources, such as solar, and to utilize independently produced energy resources, such as combined heat and power (CHP), through a shared use of such resources. The community microgrid can also be used to supplement utility electrical power supplied to the neighbored with renewable sources such as solar panels, emergency back-up power through batteries or generator.

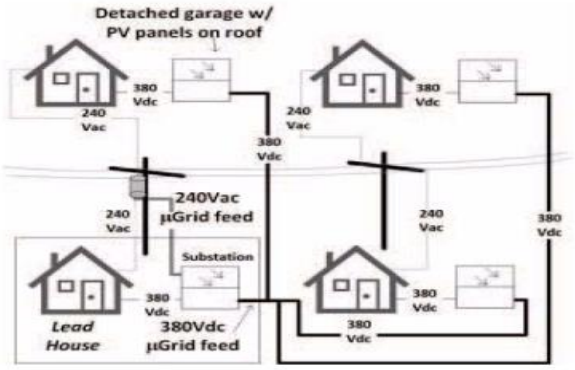

Figure 1.2: Model home concept of DC community micro grid

A community DC microgrid is proposed that is embedded within an urban neighborhood in an underserved community in Milwaukee, Wisconsin (Figure 1.2). This community microgrid will be a proving ground for residential DC power distribution and sharing of renewable energy resources between homes. The system will be built into existing homes that have been vacated because of a long period of economic recession in the area in such a way that homes can be occupied while their energy usage is being monitored and managed with in a non-invasive way.

Renovated homes will either operate off the 380Vdc or from the utility. The detached home garage for each home will be equipped with solar PV and battery energy storage with a lead home acting as substation that connects the microgrid into the larger utility through a 240Vac gateway connection. The homes will also be outfitted with readily available DC appliances so that a large portion of the home load can be directly fed from DC.

Automation of DC micro grid:

A successful transmission automation system is the foundation for a high level of functionality and flexibility in energy usage. A smart transmission grid increases overall grid reliability and efficiency while reducing line losses and faults. It incorporates decentralized energy sources and delivers electricity through the power lines on demand. An optimally engineered power transmission network must be both economically viable and physically feasible.

DC power grids are opening opportunities for safe, efficient, and reliable generation, transmission, distribution, and consumption mainly based on DC technology. In this respect, DC micro grids are gaining increasing research interest in recent years as an effective solution to integrate several types of distributed renewable energy resources, energy storage, and DC loads for future DC homes. Figure 1.3 shows a bird eye view of DC micro grid components and its interface. One such component is a Voltage Source Inverter. This report will be a guideline for automating such component and my thesis project showcases how to automate a Voltage Source Inverter and in turn provides a path to automate similar components in the DC micro grid.

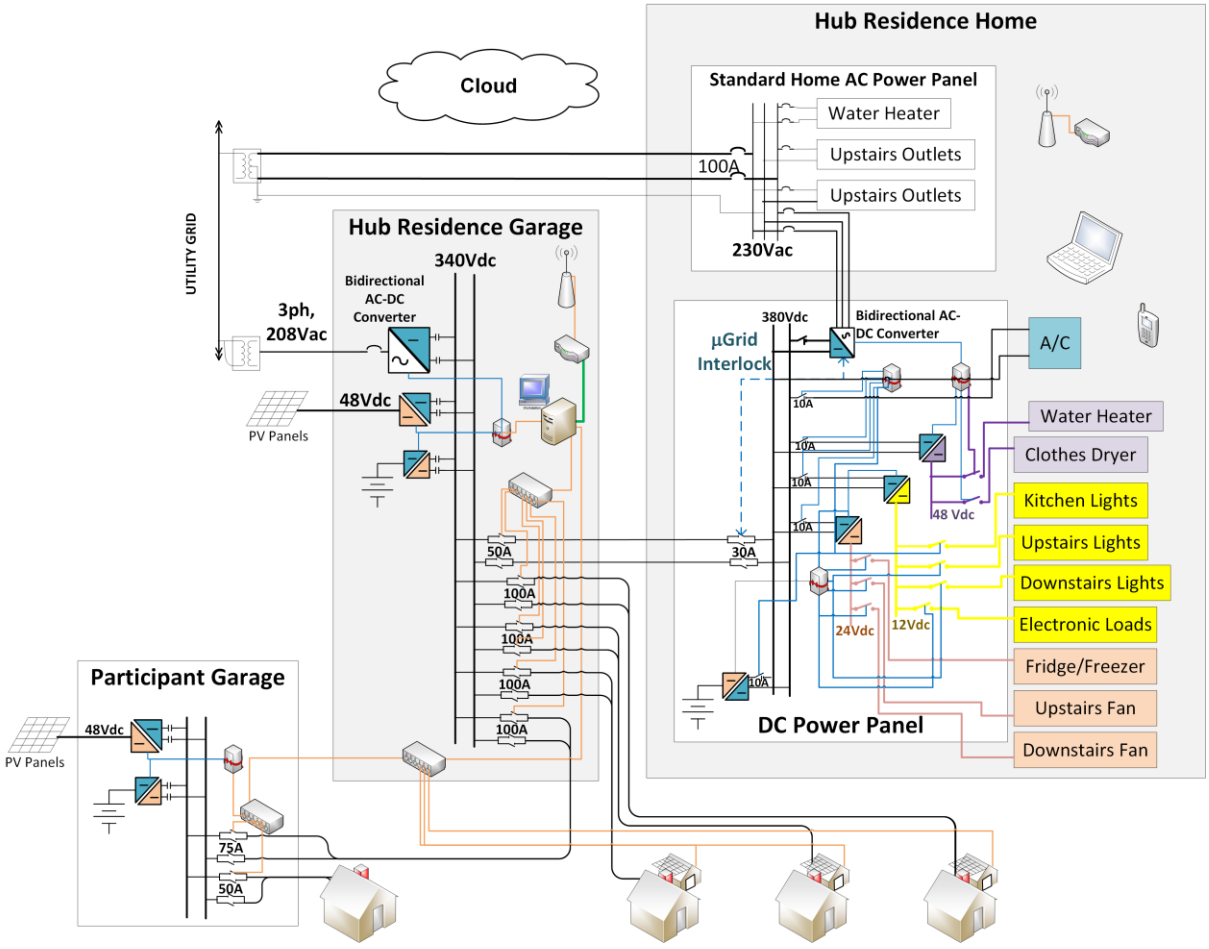

Figure 1.3: DC micro grid components in the proposed neighborhood

CHAPTER 2: REAL TIME TARGET FPGA

2.1 Introduction

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. This feature distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are custom manufactured for specific design tasks. Although one-time programmable (OTP) FPGAs are available, the dominant types are SRAM based which can be reprogrammed as the design evolves.

2.2 SOC FPGAs

Processors and FPGAs (field-programmable gate arrays) are the hardworking cores of most embedded systems. Integrating the high-level management functionality of processors and the stringent, real-time operations, extreme data processing, or interface functions of an FPGA (Field Programmable Gate Array) into a single device forms an even more powerful embedded computing platform.

SoC FPGA devices integrate both processor and FPGA architectures into a single device. Consequently, they provide higher integration, lower power, smaller board size, and higher bandwidth communication between the processor and FPGA. They also include a rich set of peripherals, on-chip memory, an FPGA-style logic array, and high-speed transceivers.

Figure 2.1: (Left) DSP and FPGA integrated into single SOC FPGA;
Figure 2.2: (Right) ASIC with CPU migrated into ARM based SOC FPGA

2.2.1 ASSP

ASSP - Application Specific Standard Parts. A chip that is designed for a specific application but not customized for a system or a customer. They are designed for an application but target to sell in a wider market and multiple customers. Design and implementation is similar to ASIC process. Examples are: A standalone USB interface chip, PCIE controller, Ethernet Controller etc. An audio/video encoder/decoder chip.

2.2.2 ASIC

ASIC - Application Specific Integrated Circuit. A chip that is custom designed for a specific application. They are designed by companies for self-use or for a specific customer targeting a specific application and a very specific system. They are customized for high performance and low power for a given system. ASIC need not be just digital logic they can be digital or analog or mix of both. Example: A 24x24 10G ethernet switch that is custom designed for a specific system to meet performance/power demand.

2.2.3 SOC

SOC - System On Chip. New trend with more and more integration of components on a chip. Contains one or more processor cores Microprocessor or Microcontroller or DSP or GPU etc. On chip memory, hardware accelerators, peripheral functions. Ethernet controller, PCIE controller, SATA, USB etc. An ASIC or ASPP can be an SOC or a non-SOC. An ASIC/ASSP with processor is a SOC. An ASIC/ASSP without processor is a non-SOC. Software development is equally important in SOC designs along with H/W to control the processor cores on the chip.

2.2.4 Difference between FPGA and ASIC

FPGA	ASIC
* Field Programmable Gate Array	* Application Specific IC
* FPGA based designs follow a relatively simpler design process Functional Specification --> HDL --> Synthesis --> Place and Route --> Timing Verification --> Download code on FPGA and Verify	* Long and complex design process compared to FPGA based design. Refer blog post named "ASIC Design Flow"
* Since hardware is already available, no non-recurring expenses (NRE)	* NRE involves cost for circuit design and mask design and hence expensive
* Because of above 2 reasons, faster "time-to-market"	* Above 2 reasons make it a longer "time-to-market" product
* Same FPGA can be reprogrammed to suit another application, so reusable and flexible	* Made for a specific application and hence cannot be reprogrammed to suit another application
* Standard array design and hence most times, hardware (array cells, transistors and interconnects) goes un-used. Wastage is un-avoidable.	* Full custom design with every transistor and interconnect designed to serve the purpose and so no wastage of hardware
* Better suited if the required numbers is less * Relatively larger than an ASIC	* Because of the NRE, ASIC based designs are better if the required numbers are large * For the same functionality, ASICs can be much smaller than an FPGA
* Relatively slower than an ASIC (MHz)	* Faster than an FPGA (GHz)
** FPGAs are frequently used in ASIC design to emulate the entire ASIC or parts of the ASIC in order to verify design, functionality etc.	

Figure 2.3: Differences between FPGA and ASIC

2.3 Why Altera SOC FPGA?

Table 1: Commercially-Available SoC FPGA Devices

	Altera SoC	Xilinx Zynq 7000 EPP	Microsemi SmartFusion2
Processor	ARM Cortex-A9	ARM Cortex-A9	ARM Cortex-M3
Processor Class	Application processor	Application processor	Microcontroller
Single or Dual Core	Single or Dual	Dual	Single
Processor Max. Frequency	1.05 GHz	1.0 GHz	166 MHz
L1 Cache	Data: 32 KB Instruction: 32 KB	Data: 32 KB Instruction: 32 KB	No data cache Instruction: 8 KB
L2 Cache	Unified: 512 KB, with Error Correction Code	Unified: 512 KB	Not Available
Memory Management Unit (MMU)	Yes	Yes	Yes
Floating Point Unit/NEON Multimedia Engine	Yes	Yes	Not available
Acceleration Coherency Port (ACP)	Yes	Yes	Not available
Interrupt Controller	Generic (GIC)	Generic (GIC)	Nested, vectored (NVIC)
On-Chip Processor RAM	64 KB, with ECC	256 KB, no ECC	64 KB, no ECC
Direct Memory Access Controller	8-channel ARM DMA330 32 peripheral requests (FPGA + hard processor system	8-channel ARM DMA330 4 peripheral requests (FPGA only)	1-channel HPDMA 4 requests
External Memory Controller	Yes	Yes	Yes
Memory Types Supported	LPDDR2, DDR2, DDR3L, DDR3	LPDDR2, DDR2, DDR3L, DDR3	LPDDR, DDR2, DDR3
External Memory ECC	16 bit, 32 bit	16-bit	8 bit, 16 bit, 32 bit
External Memory Bus Max. Frequency	400 MHz (Cyclone V SoC), 533 MHz (Arria V SoC)	667 MHz	333 MHz
Processor Peripherals	1x Quad SPI controller 1x NAND controller 2x 10/100/1G Ethernet controller 2x USB 2.0 On the Go (OTG) controller 1x SD/MMC/SDIO controller 2x UART 4x I²C controller 2x CAN controller 2x SPI master, 2x SPI slave controller 4x 32 bit general-purpose timers 2x 32 bit watchdog timers	1x Quad SPI controller 1x static memory controller (NAND, NOR, or SSRAM) 2x 10/100/1G Ethernet controller 2x USB 2.0 OTG controller 2x SD/SDIO controller 2x UART 2x I²C controller 2x CAN controller 2x SPI controllers (master or slave) 2x 16 bit triple-mode timer/counters 1x 24 bit watchdog timer	1x 10/100/1G Ethernet controller 2x USB 2.0 OTG controller 2x UART 2x I²C controller 1x CAN controller 2x SPI 2x general-purpose timers 1x watchdog timer 1x real-time clock (RTC)
FPGA Fabric	Cyclone V, Arria V	Artix-7, Kintex-7	Fusion2
FPGA Logic Density Range	25 K to 462 K LE	28K to 444 K LC	6 K to 146 K LE
Hardened Memory Controllers in FPGA	Up to 3, with ECC	Not available	Not available
High-speed Transceivers	Available at all densities	Higher-density devices only	Higher-density devices only
Analog Mixed Signal (AMS)	Not available	2 x 12-bit, 1 MSPS analog-to-digital converters (ADCs)	Not available
Boot Sequence	Processor first, FPGA first, or both	Processor first	Processor first

Figure 2.4: Commercially available SOC FPGAs and the differences in features among them.

2.4 Altera DE1 SOC FPGA

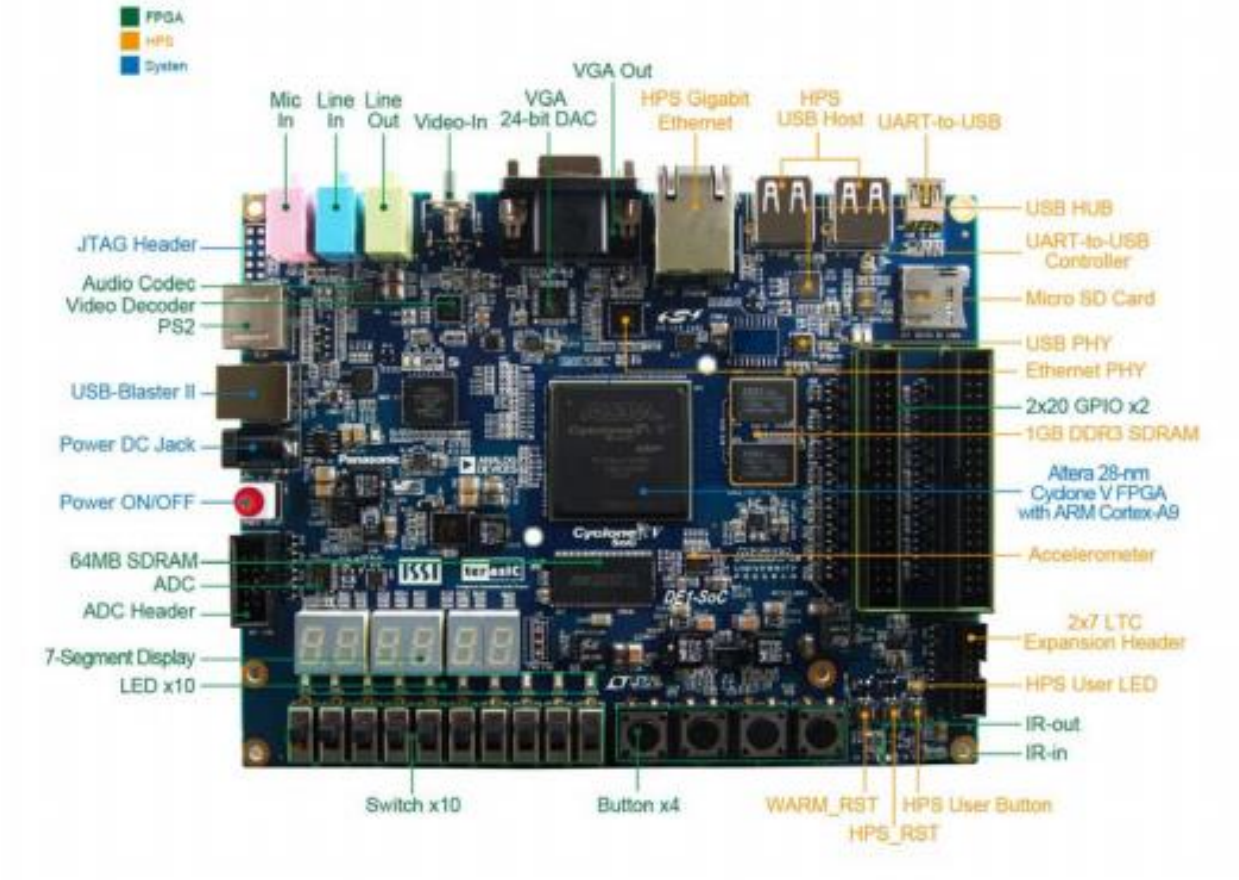

Figure 2.5: Top view of DE1 SOC FPGA

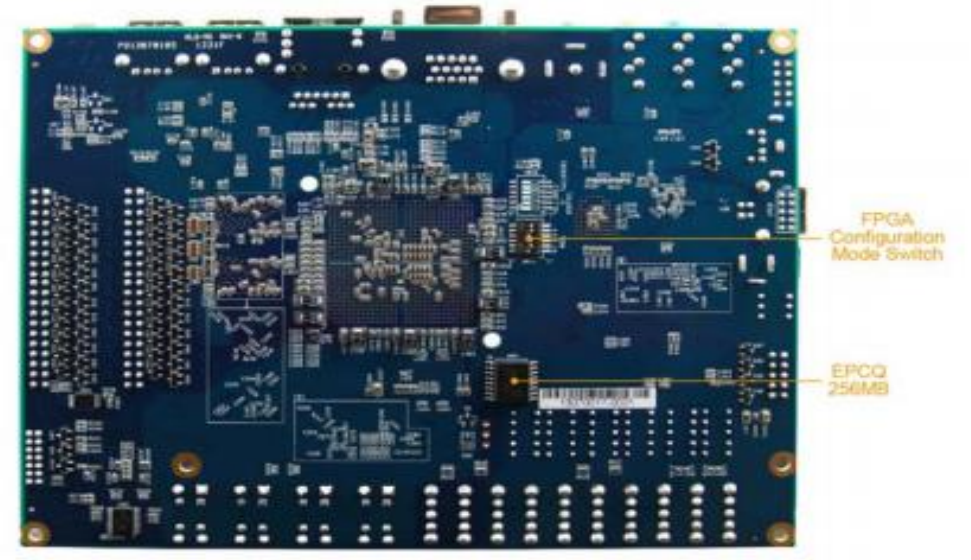

Figure 2.6: Bottom view of DE1 SOC FPGA

FPGA Hardware:

The FPGA hardware consists of Altera Cyclone® V SE 5CSEMA5F31C6N device and following are the features included in it:

1. Altera serial configuration device – EPCQ256
2. USB-Blaster II onboard for programming; JTAG Mode
3. 64MB SDRAM (16-bit data bus)
4. 4 push-buttons
5. 10 slide switches
6. 10 red user LEDs
7. Six 7-segment displays
8. Four 50MHz clock sources from the clock generator
9. 24-bit CD-quality audio CODEC with line-in, line-out, and microphone-in jacks
10. VGA DAC (8-bit high-speed triple DACs) with VGA-out connector
11. TV decoder (NTSC/PAL/SECAM) and TV-in connector
12. PS/2 mouse/keyboard connector
13. IR receiver and IR emitter
14. Two 40-pin expansion headers with diode protection
15. A/D converter, 4-pin SPI interface with FPGA

HPS Hardware:

The HPS hardware consists of 800MHz Dual-core ARM Cortex-A9 MP Core processor and following are its features:

1. 1GB DDR3 SDRAM (32-bit data bus)
2. 1 Gigabit Ethernet PHY with RJ45 connector
3. 2-port USB Host, normal Type-A USB connector

4. Micro SD card socket
5. Accelerometer (I2C interface + interrupt)
6. UART to USB, USB Mini-B connector
7. Warm reset button and cold reset button
8. One user button and one user LED
9. LTC 2x7 expansion header.

2.5 Block Diagram of Altera DE1 SOC FPGA

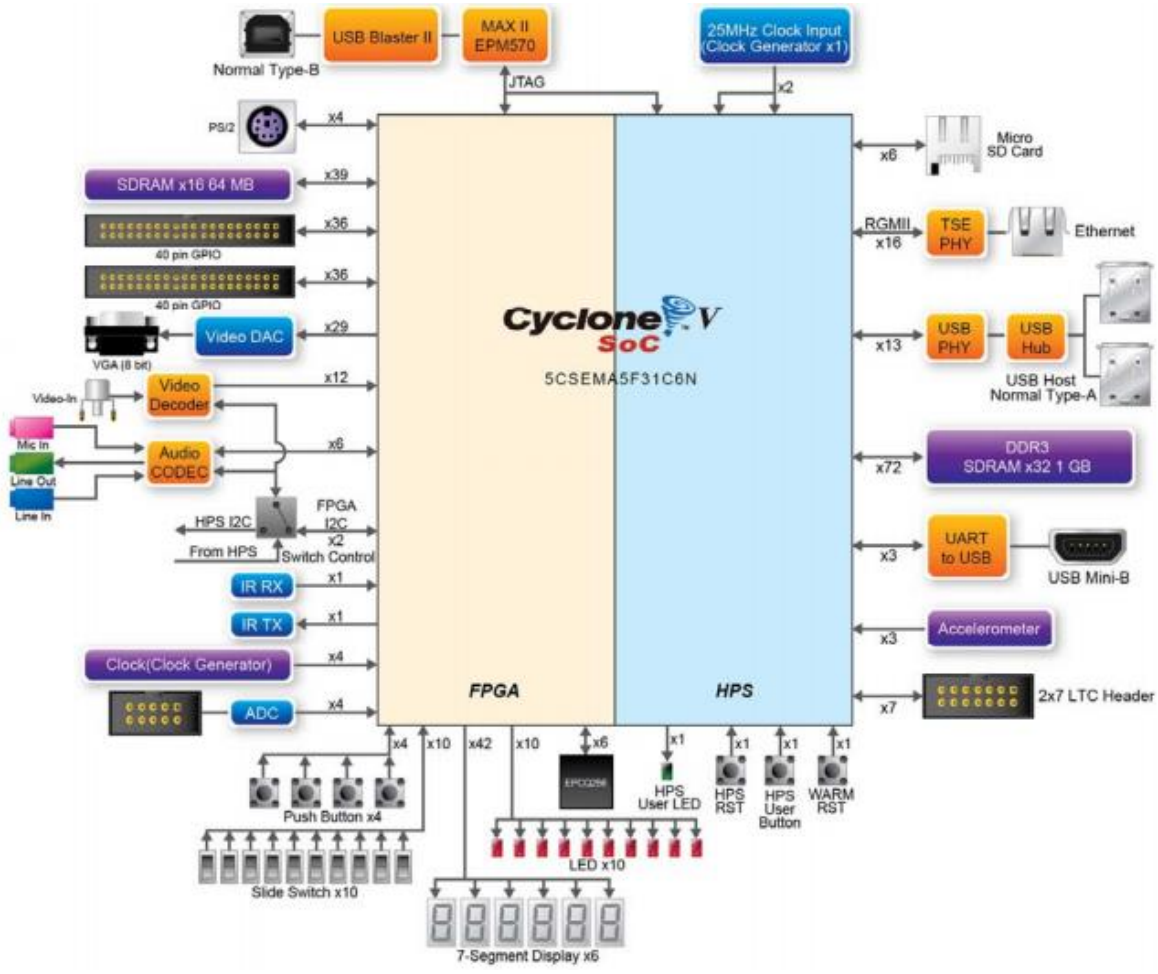

Figure 2.7: Block diagram of FPGA and HPS on DE1 SOC FPGA

The FPGA and HPS on SOC FPGA consists of the following elements:

FPGA Device:

1. Cyclone V SoC 5CSEMA5F31 Device
2. Dual-core ARM Cortex-A9 (HPS)
3. 85K programmable logic elements
4. 4,450 Kbits embedded memory
5. fractional PLLs
6. hard memory controllers

Configuration and Debug

1. Quad serial configuration device – EPCQ256 on FPGA
2. Onboard USB-Blaster II (normal type B USB connector)

Memory Device

1. 64MB (32Mx16) SDRAM on FPGA
2. 1GB (2x256Mx16) DDR3 SDRAM on HPS
3. Micro SD card socket on HPS

Communication

1. Two port USB 2.0 Host (ULPI interface with USB type A connector)
2. UART to USB (USB Mini-B connector)
3. 10/100/1000 Ethernet
4. PS/2 mouse/keyboard
5. IR emitter/receiver
6. I2C multiplexer

Connectors

1. Two 40-pin expansion headers

2. One 10-pin ADC input header
3. One LTC connector (one Serial Peripheral Interface (SPI) Master, one I2C and one GPIO interface)

Display

1. 24-bit VGA DAC

Audio

1. 24-bit CODEC, Line-in, Line-out, and microphone-in jacks

Video Input

1. TV decoder (NTSC/PAL/SECAM) and TV-in connector

ADC

1. Fast throughput rate: 1 MSPS
2. Channel number: 8
3. Resolution: 12-bit
4. Analog input range: 0 ~ 2.5V or 0 ~ 5V as selected via the RANGE bit in control register

Switches, Buttons, and Indicators

1. user Keys (FPGA x4, HPS x1)
2. 10 user switches (FPGA x10)
3. 11 user LEDs (FPGA x10, HPS x 1)
4. HPS reset buttons (HPS_RESET_n and HPS_WARM_RST_n)
5. Six 7-segment displays

Sensors

1. G-Sensor on HPS

Power

1. 2V DC input

CHAPTER 3: MATLAB/SIMULINK

3.1 Introduction

The name MATLAB stands for MATrix LABoratory. MATLAB was written originally to provide easy access to matrix software developed by the LINPACK (linear system package) and EISPACK (Eigen system package) projects. MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming environment. Furthermore, MATLAB is a modern programming language environment: it has sophisticated data structures, contains built-in editing and debugging tools, and supports object-oriented programming. These factors make MATLAB an excellent tool for teaching and research.

It has powerful built-in routines that enable a very wide variety of computations. It also has easy to use graphics commands that make the visualization of results immediately available. Specific applications are collected in packages referred to as toolbox. There are toolboxes for signal processing, symbolic computation, control theory, simulation, optimization, and several other fields of applied science and engineering.

Simulink, developed by The MathWorks, is a commercial tool for modeling, simulating and analyzing dynamic systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. It offers tight integration with the rest of the MATLAB environment and can either drive MATLAB or be scripted from it. Simulink is widely used in control theory and digital signal processing for simulation and design.

3.2 MATLAB Toolkit

MATLAB provides various toolboxes for users to performs different functionalities. The toolboxes which were extensively used on this model for automation.

3.2.1 HDL Coder

HDL Coder generates portable, synthesizable VHDL and Verilog code from MATLAB functions, Simulink models, and Stateflow charts. The generated HDL code can be used for FPGA programming or ASIC prototyping and design. HDL Coder provides a workflow advisor that automates the programming of Xilinx and Altera FPGAs. It provides an ease to control HDL architecture and implementation, highlight critical paths, and generate hardware resource utilization estimates. HDL Coder provides traceability between your Simulink model and the generated HDL code, enabling code verification for high-integrity applications adhering to DO-254 and other standards.

3.2.2 HDL Verifier

HDL Verifier automatically generates test benches for Verilog and VHDL design verification. MATLAB or Simulink can be directly used to stimulate your design and then analyze its response using HDL co-simulation or FPGA-in-the-loop with Xilinx and Intel FPGA boards. This approach eliminates the need to author standalone Verilog or VHDL test benches.

HDL Verifier also generates components that reuse MATLAB and Simulink models natively in simulators from Cadence, Mentor Graphics, and Synopsys. These components can be used as verification checker models or as stimuli in more complex test-bench environments such as those that use the Universal Verification Methodology (UVM).

3.2.3 Stateflow

Stateflow is an environment for modeling and simulating combinatorial and sequential decision logic based on state machines and flow charts. Stateflow lets the user combine graphical and tabular representations, including state transition diagrams, flow charts, state transition tables, and truth tables, to model how the system reacts to events, time-based conditions, and external input signals.

With Stateflow we can design logic for supervisory control, task scheduling, and fault management applications. Stateflow includes state machine animation and static and run-time checks for testing design consistency and completeness before implementation.

3.2.4 Simulink Coder

Simulink Coder (formerly Real-Time Workshop) generates and executes C and C++ from Simulink diagrams, Stateflow charts, and MATLAB functions. The generated source code can be used for real-time and non-real-time applications, including simulation acceleration, rapid prototyping, and hardware-in-the-loop testing. The generated code can be tuned and monitored using Simulink or run and interact with the code outside MATLAB and Simulink.

3.2.5 Embedded Coder

Embedded Coder generates readable, compact, and fast C and C++ code for use on embedded processors, on-target rapid prototyping boards, and microprocessors used in mass production. Embedded Coder enables additional MATLAB Coder and Simulink Coder configuration options and advanced optimizations for fine-grain control of the generated code's functions, files, and data. These optimizations improve code efficiency and facilitate integration with legacy code, data types, and calibration parameters used in production. It can incorporate a third-party development

environment into the build process to produce an executable for turnkey deployment on your embedded system. Embedded Coder offers built-in support for AUTOSAR and ASAP2 software standards. It also provides traceability reports, code interface documentation, and automated software verification to support DO-178, IEC 61508, and ISO 26262 software development.

3.2.6 MATLAB Coder

MATLAB Coder generates readable and portable C and C++ code from MATLAB code. It supports most of the MATLAB language and a wide range of toolboxes. It is used to integrate the generated code into projects as source code, static libraries, or dynamic libraries. It is also used to generate code within the MATLAB environment to accelerate computationally intensive portions of your MATLAB code. MATLAB Coder lets you incorporate legacy C code into your MATLAB algorithm and into the generated code.

By using MATLAB Coder with Embedded Coder, it can further optimize code efficiency and customize the generated code. The numerical behavior of the generated code is verified using software-in-the-loop (SIL), FPGA-in-the-loop (FIL) and processor-in-the-loop (PIL) execution.

CHAPTER 4: DC-MICROGRID COMPONENT

Voltage Source Inverter model is simulated for real-time synthesis. The word ‘inverter’ in the context of power-electronics denotes a class of power conversion (or power conditioning) circuits that operates from a dc voltage source or a dc current source and converts it into ac voltage or current. The ‘inverter’ does reverse of what ac-to-dc ‘converter’ does.

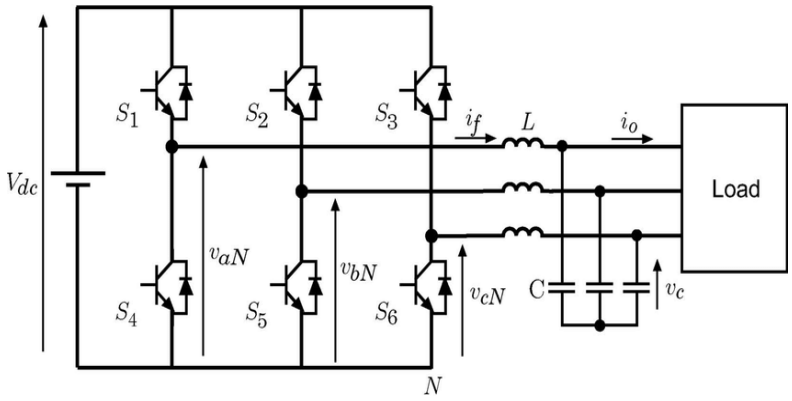

Figure 4.1: 3 Phase Voltage Source Inverter

4.1 General Theory of PWM (Pulse Width Modulation)

PWM (Pulse Width Modulation) technology was put forward based on an important conclusion in the sample control theory that when two groups of pulses with the same impulse area but different waveforms are input to an inertial link, the effectiveness of these two groups of impulses are the same. The main principle of PWM technique can be briefly described as: Through ON/OFF control on the semiconductor switching components, a series of pulses with the same amplitude and different width are generated on the output port to replace the sinusoidal wave or other waveforms required. The duty cycle of the output waveform needs to be modulated by a certain rule and as a result both the output voltage and output frequency of the inverter can be regulated.

4.1.1 Basic Principle of SPWM (Sinusoidal PWM)

Among all PWM schemes, SPWM is one of the most popular and simple methods utilized in power inverter and motor control fields. Its main features can be summarized as sine-triangle wave comparison. As shown in Figure 4.2, a sine wave (modulated wave, magenta) is compared with a triangle wave (carrier wave, green) and when the instantaneous value of the triangle wave is less than that of the sine wave, the PWM output signal (orange) is in high level (1). Otherwise it is turned into the low level (0). The level switching edge is produced at every moment the sine wave intersects the triangle wave. Thus, the different crossing positions result in variable duty cycle of the output waveform.

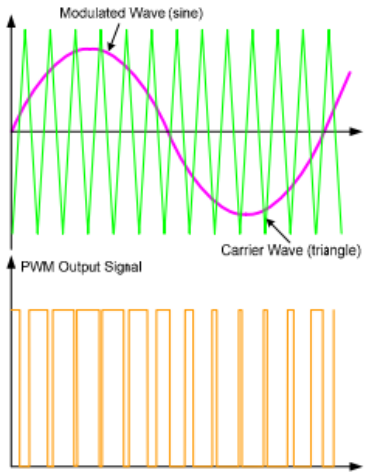

Figure 4.2: SPWM Waveform Generation

4.1.2 Implementation Method

In terms of the basic principle of SPWM illustrated above, it's easy to implement using analog circuit (Figure 4.3). Sine and triangle waves are respectively generated by specially designed circuits and then fed to the properly selected comparator which can output the desired SPWM signal. But the control precision and reliability of this scheme are always not so satisfying due to the complicated circuit structure as well as the instability of the parameters of all analog devices.

With the development of the microcontroller, nowadays the software implementation for SPWM is mostly adopted to realize high precision control.

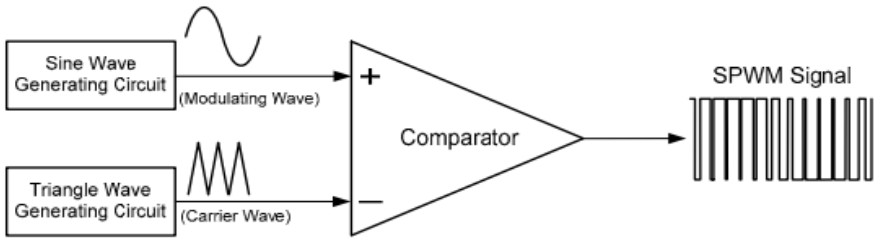

Figure 4.3: SPWM Implementation

The method utilizing the natural intersection points of sine wave with triangle wave to realize PWM is called “Natural Sampled Method”. It’s able to demonstrate the true moments the pulse is started and ended and the SPWM waveform is much closer to sine wave. This method is not adopted in most control applications due to the random intersection points of sine and triangle wave which results in complicated calculation and difficult real-time implementation.

To overcome these disadvantages, another new method called “Regular Sampled Method” was put forward. It is widely used in engineering applications nowadays. It is based on the principle that a certain moment is selected in every cycle of the triangle carrier wave to find the corresponding value of the sine wave voltage which is introduced to sample on triangle wave and the sample result determines the ON/OFF moments of the power devices, ignoring whether the sine wave and triangle wave intersects in this moment or not.

A more practical method named “Average Symmetric Regular Sampled Method” (illustrated in Figure 4.4) is applied in most control cases. In Figure 6, the sampled moment is given on the trough point of the triangle wave, then centered by the corresponding value of sine wave voltage, a horizontal line is drawn to intersect the triangle wave on both sides so the leading and trailing edges of PWM waveform are decided upon that. The leading edge is a little wider which just compensates the narrow trailing edge and therefore as an average consideration the effectiveness

of this method is almost equivalent to that of the natural sampled method.

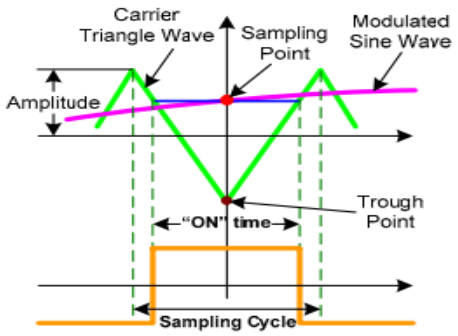

Figure 4.4: Symmetric Regular Sample Method

4.1.3 Modulation Index

When the amplitude of the modulated sine wave is larger than that of the carrier triangle wave, over modulation occurs. Once the sine wave reaches the peak of the triangle, the PWM pulses will obtain the maximum width so the modulation will enter the state of saturation (Figure 4.5). Therefore, the item “Modulation index” (represented by m) defined by the ratio of the amplitude of the modulated wave to that of the carrier wave is introduced to describe the modulation state. When 0<m<1, the linear relationship between the input and PWM output voltage is maintained. If the value of modulation index exceeds 1, this linear mode cannot be kept anymore and the special control strategy for over modulation is required.

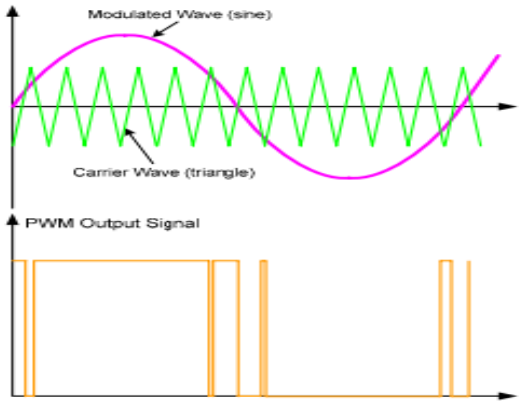

Figure 4.5: SPWM saturation

4.1.4 Phase SPWM

For 3-phase AC load, the SPWM signals to trigger the six power switches in the voltage source inverter is generated by comparison of the 3-phase sine waves with the same triangle wave (Figure 4.6).

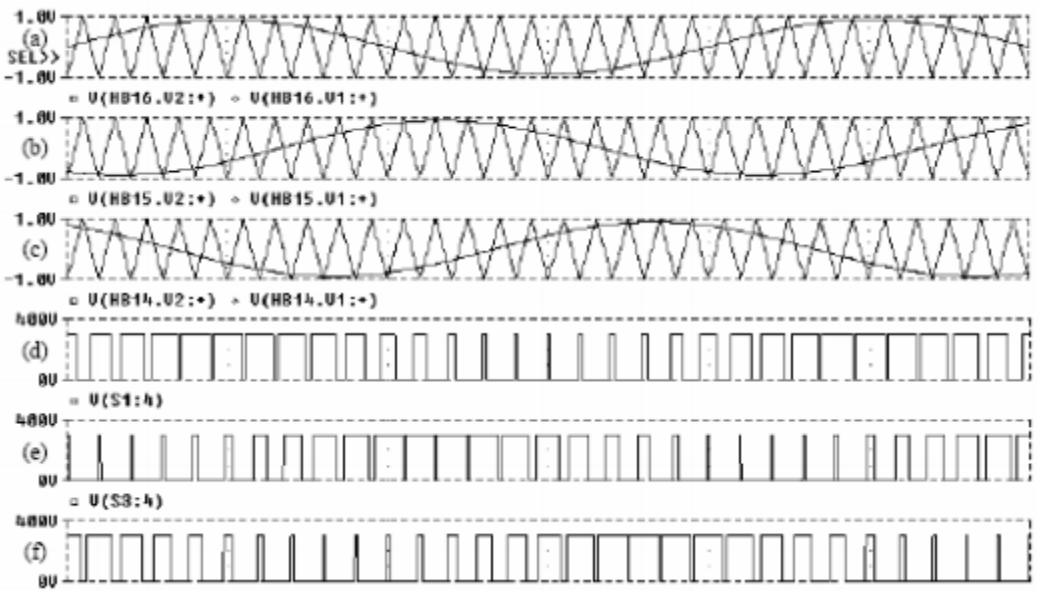

Figure 4.6: 3 Phase Sinusoidal PWM waveforms

4.2 Creating a VSI Model using Simulink

Steps for creating a VSI Model on Simulink for Automation.

1. On the MATLAB Home tab, click Simulink.
2. In the Simulink start page, choose a HDL Code generation template
3. After selecting the HDL Code generation template, click Create Model
4. A new empty model using the template settings and contents opens in the Simulink Editor.
5. To build models that are compatible with the HDL Coder software, use blocks from the HDL block set or other supported block set on Library Browser view.
6. Other HDL model supported block sets are Simulink, DSP System Toolbox,

Communication Toolbox, HDL Demo Library, HDL Verifier, Stateflow.

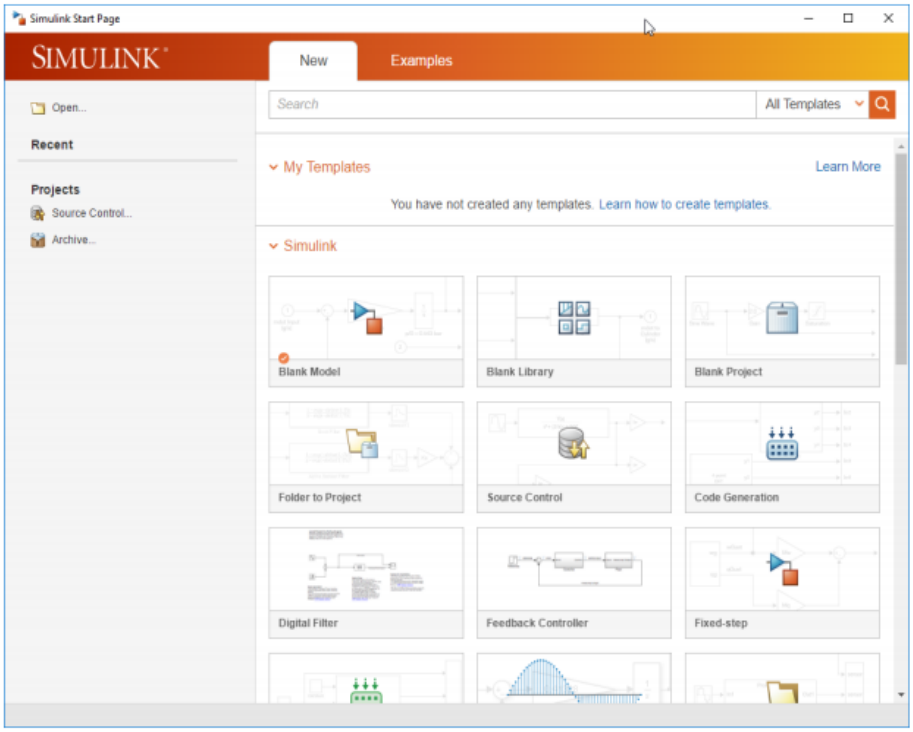

Figure 4.7: Simulink template browser view

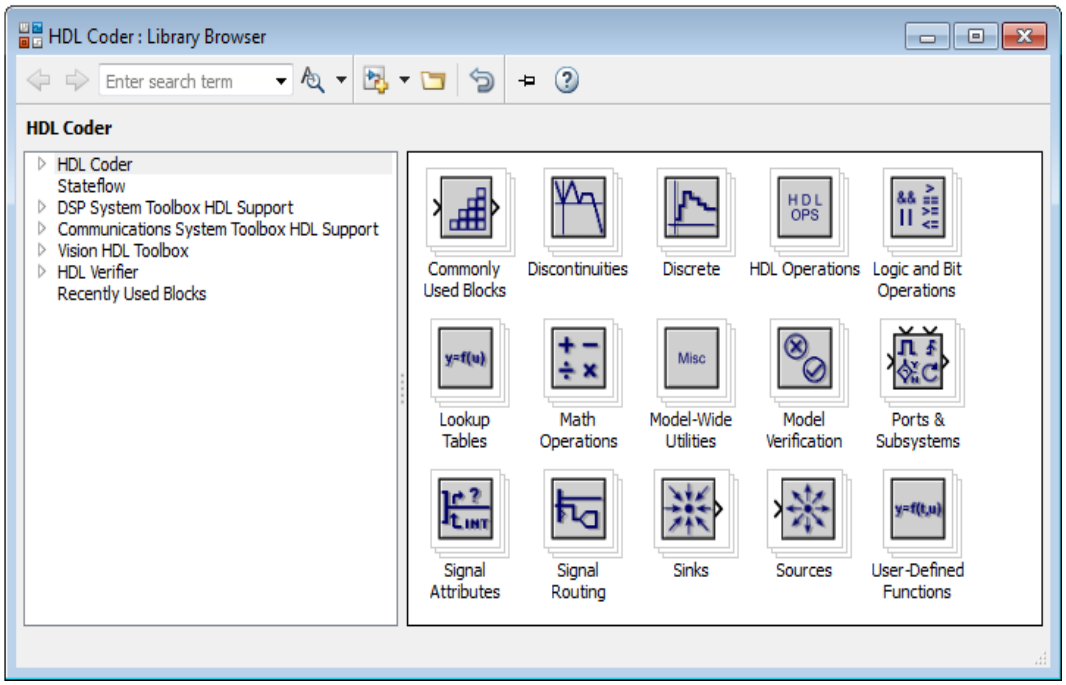

Figure 4.8: Simulink Library browser, HDL Coder blocks view

4.2.1 Creating VSI Model using supported block set

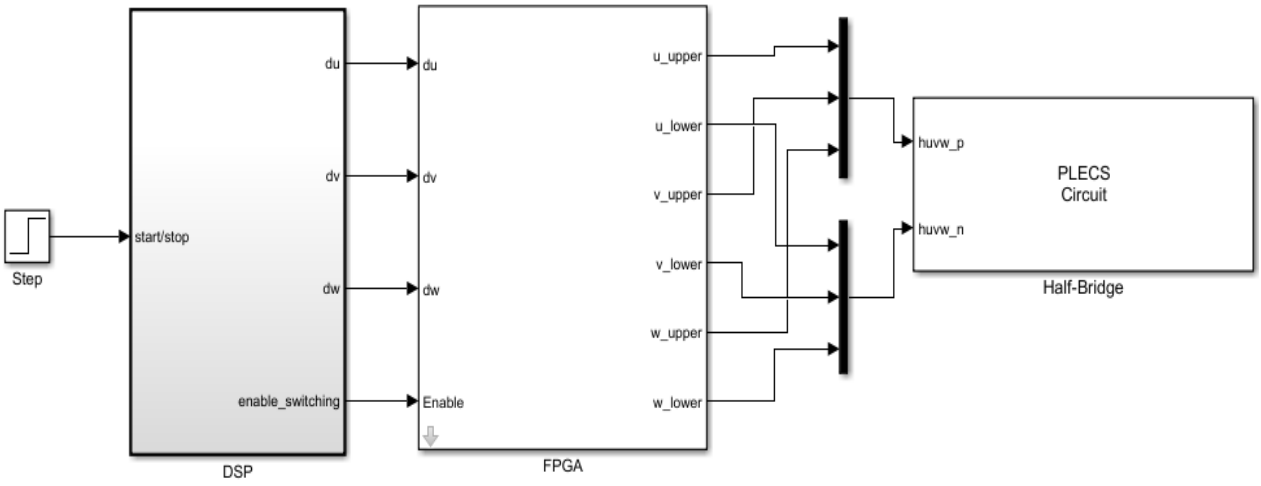

Figure 4.9: VSI model subsystem view

The Step and the DSP (Digital Signal Processing) subsystem contains block components that facilitate generation of 3 Phase Sinusoidal PWM signals du, dv, dw. The inside view of the DSP subsystem is given in Figure 4.10. The FPGA subsystem consists of components that facilitate the generation of six switching nodes which will furthermore switch transistors of 3-Phase Voltage Source Inverter present in the Half-Bridge subsystem. The Step subsystem in the above system view is a free running counter which can be controlled by user and will start or stop based on the user input under automation.

The Figure 4.10 shows the inside view of DSP subsystem and the Step subsystem is also incorporated into it on the form of start, stop and counter free running blocks. The trigonometric blocks generate sinusoidal signals with input phase angle from the Phase Angle Generator subsystem. The HDL coder mainly supports single and double data types and hence the conversions in the model should be represented in that form. The Data Type Conversion blocks throughout the VSI model is used to convert signed or unsigned / fixed or variable length integers into single or double data type. Max count is 250 for FPGA has 25MHz frequency.

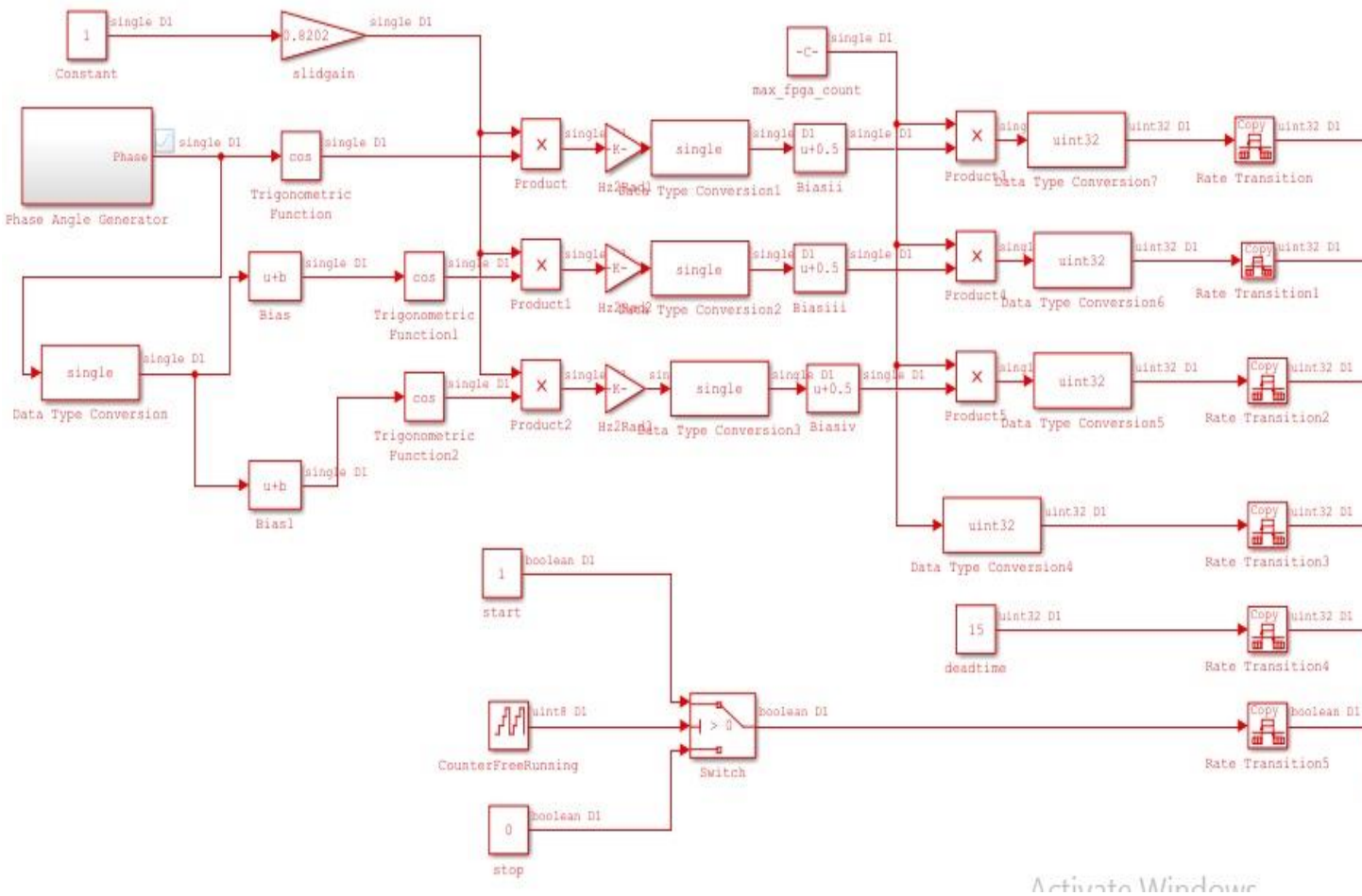

Figure 4.10: Inside view of DSP Subsystem

The sinusoidal signals are multiplied with value the user provides using slidergain (0.8202) and then the frequency of the resulting signal is converted from hertz to radians using gain blocks (value 0.5/Element wise multiplication). The bias blocks before trigonometric functional block has value +/- 2pi/3, this value is given as an input to trigonometric blocks for the generation of three 180degrees shifted sinusoidal signal. The bias blocks (value 0.5) after frequency conversion is used to bring the signal to 0.5+Cos (+/- 2pi/3 +phase angle) form. The Rate transition blocks are used to match the signal execution frequency of DSP subsystem with FPGA subsystem. The processing of DSP subsystem takes place in the HPS and FPGA subsystem takes place in the FPGA part of SOC-FPGA.

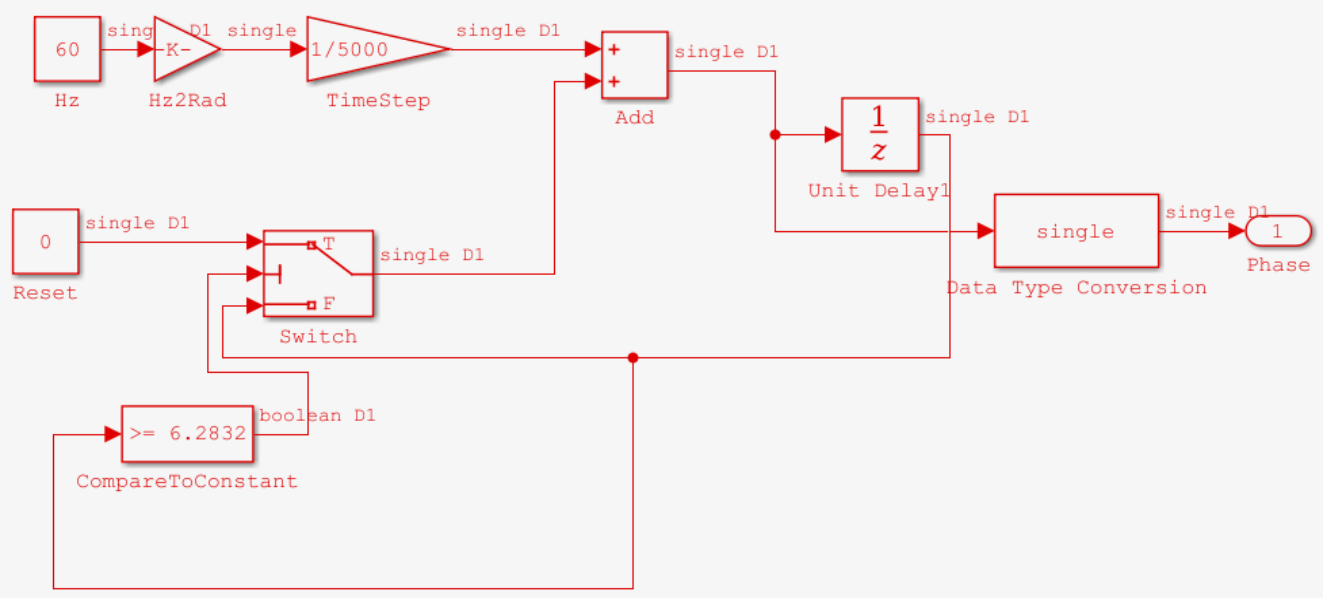

Figure 4.11: Inside view of Phase Angle Generator

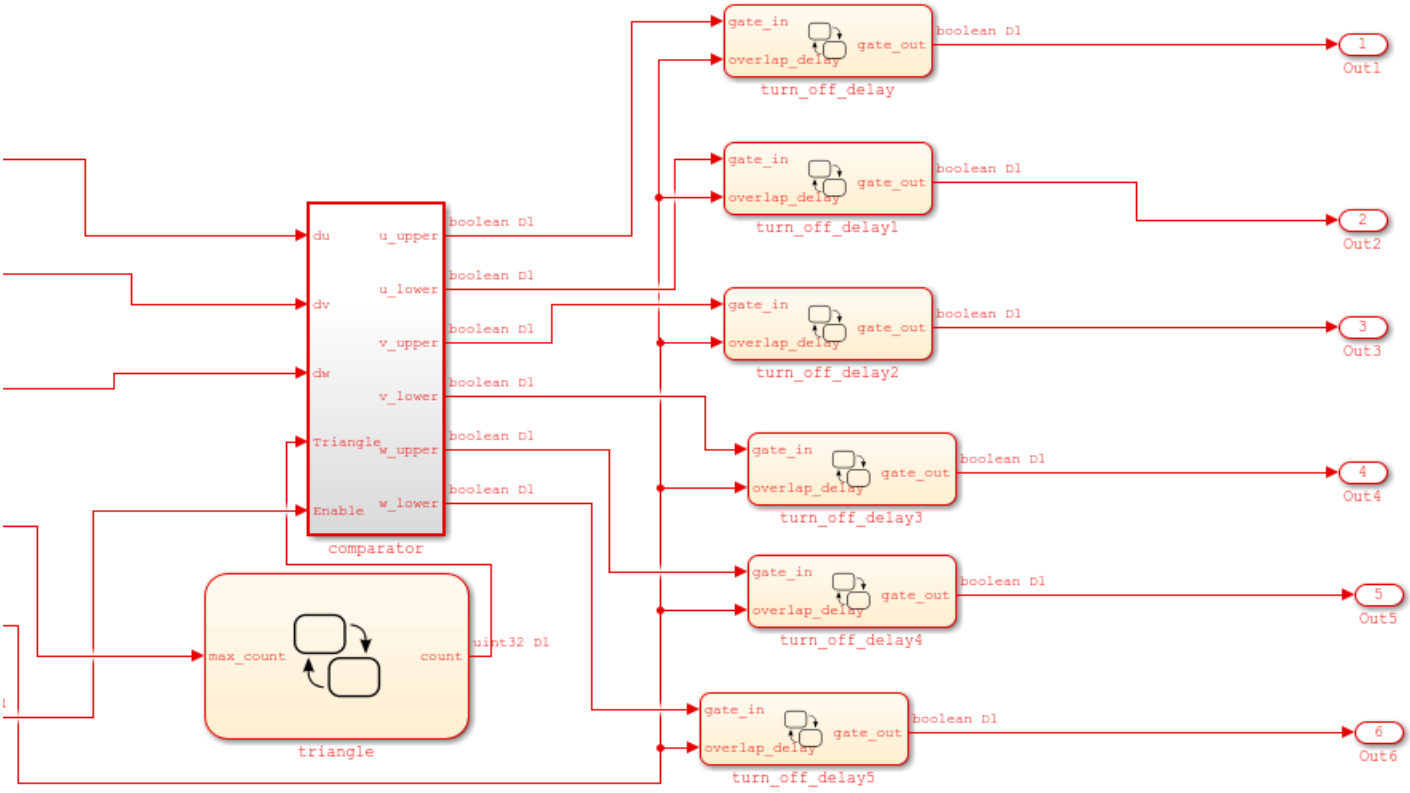

Figure 4.12: The inside view of FPGA subsystem

The FPGA subsystem consists of triangle subsystem to generate triangular carrier signal, the comparator subsystem compares the 3 phase signals with the triangular wave and provides the

output to turn off delay subsystems to switch on or off the output nodes. The triangle and turn off delay subsystems are stateflow generated. The 6 outputs from this subsystem is given to 3 phase VSI on Half Bridge.

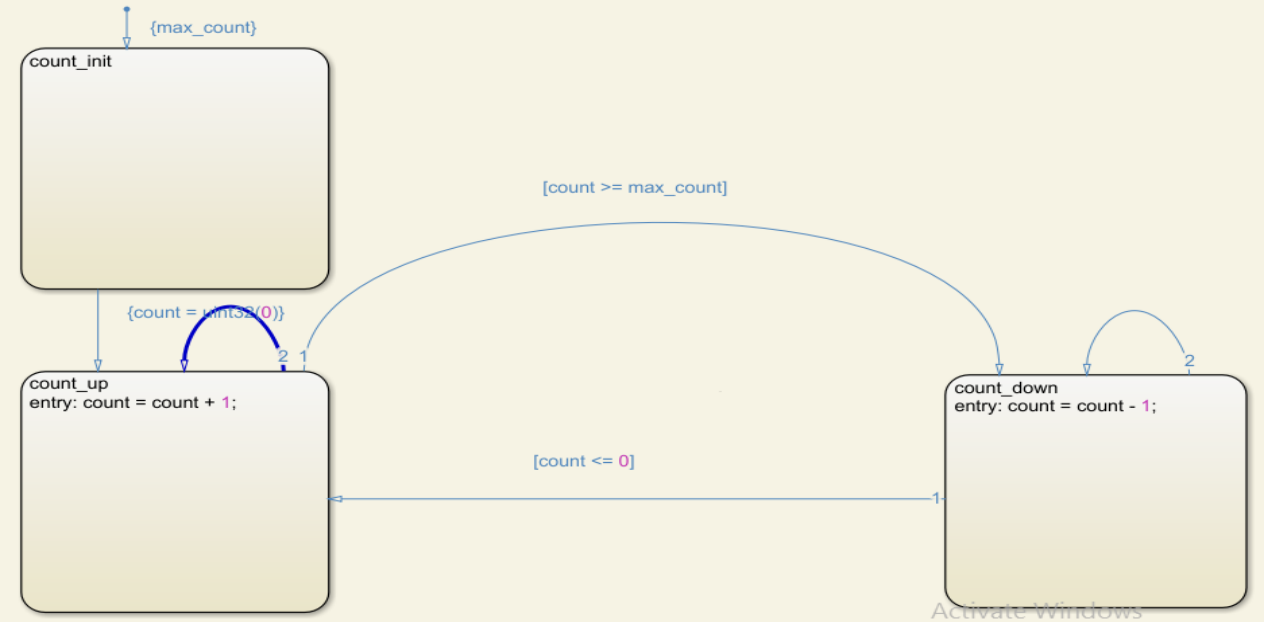

Figure 4.13: Triangle subsystem

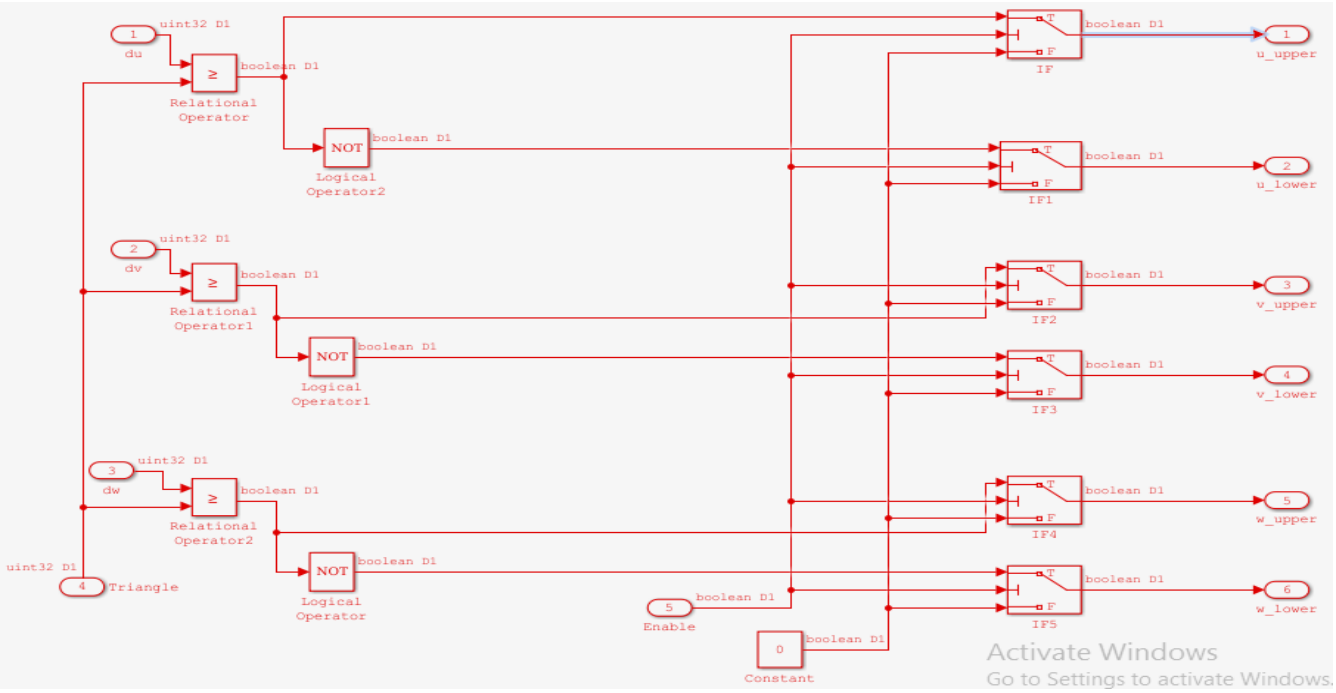

Figure 4.14: Comparator Subsystem

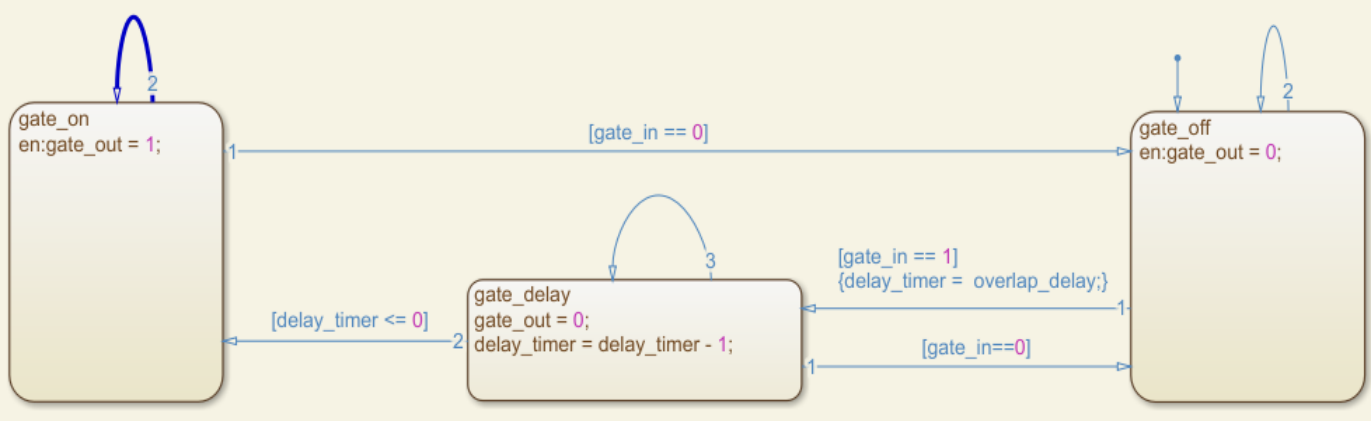

Figure 4.15: The turn-off-delay subsystem

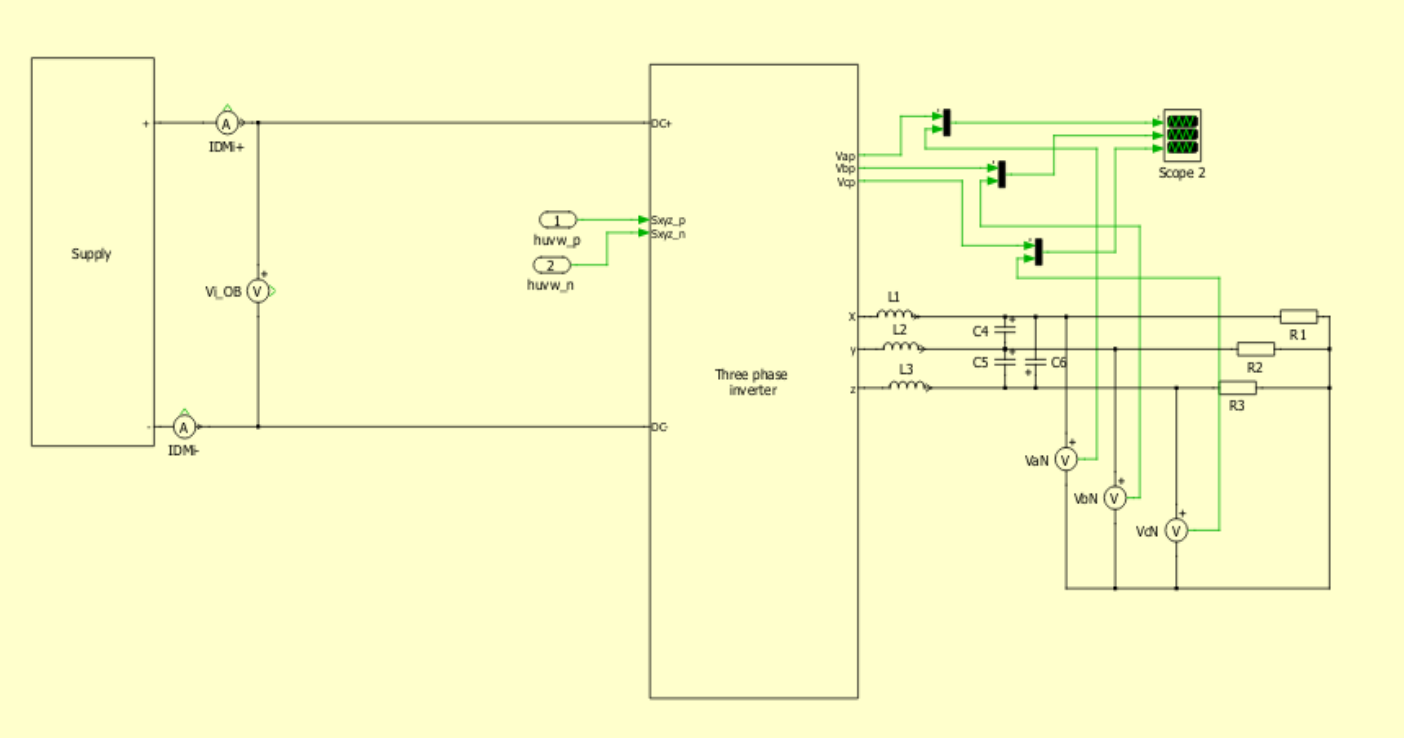

Figure 4.16: Half Bridge internal view

The PLECS tool helps to generate real time simulation environment. The above figure is a PLECS generated subsystem wherein the scope shows the integrity of the design. The subsystem consists of supply, 3 phase inverter and resistive load. The output of FPGA subsystem is provided to Sxyz_p and Sxyx_n of inverter. The 6 inputs switch the 6 transistors inside the inverter.

CHAPTER 5: HDL CODE GENERATION METHODOLOGY

HDL Code Generation from Simulink: The HDL Workflow Advisor generates VHDL and Verilog code from Simulink and Stateflow. With Simulink, algorithm is modeled using a library of more than 200 blocks, including Stateflow charts. This library provides complex functions, such as the Viterbi decoder, FFT, CIC filters, and FIR filters, for modeling signal processing and communications systems and generating HDL code.

5.1 Generating HDL Code

First step in generating HDL code is to run compatibility check on the model for verification. On Simulink browser toolbar, under Analysis select Design Verifier, Check compatibility and Model. Under Simulation select Model Configuration Parameters, change the values of Solver to fit the model requirements. Simulation time provided is 0 to Infinite as shown in Figure 5.1, this simulation will run until it is externally terminated. Solver Options to Type: Fixed-Step and Solver: Discrete. Fixed step size can be auto or 2.0E-08 which is the period based on FPGA board frequency as shown in Figure 5.2.

Then select apply for the changes and run the model, wait for it to compile and simulation to start. Once the simulation starts, verify whether the model has satisfied the user requirements, the VSI Model has scopes to verify its integrity.

Stop the simulation after few minutes and check report trace which is a html file and a full Coverage report for the model. This report formulates the model verification summary into a single file and has details of the verification and execution of every block and subsystem in the model.

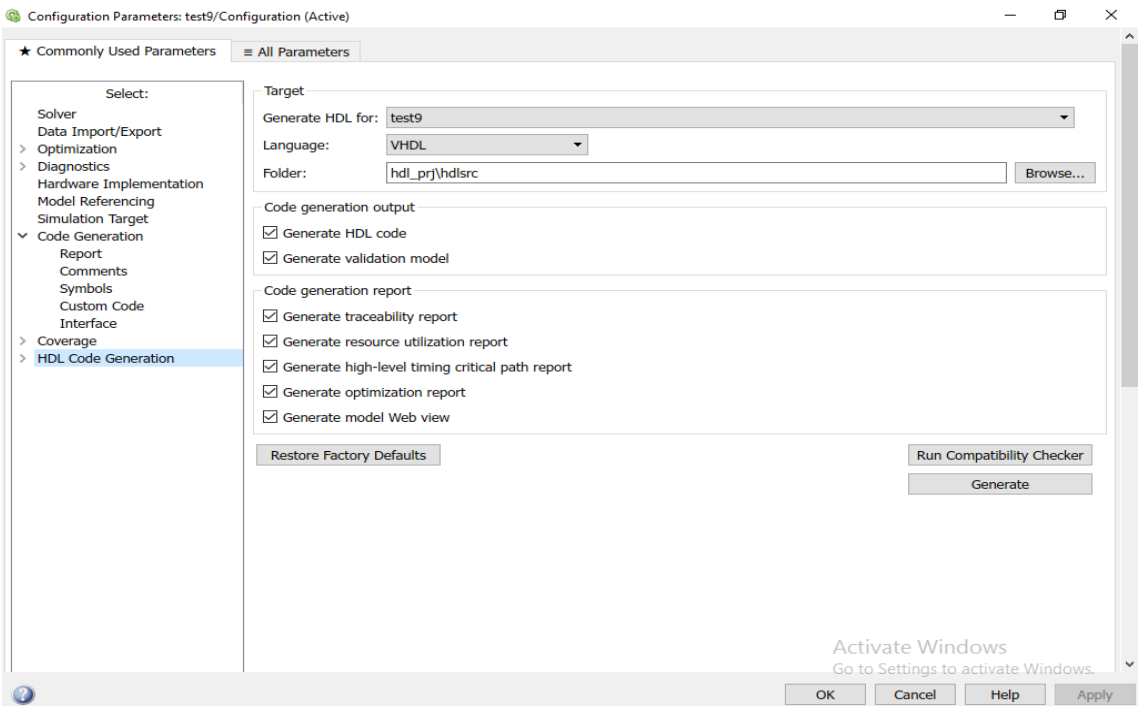

Figure 5.1: Configuration Parameter: HDL Code Generation

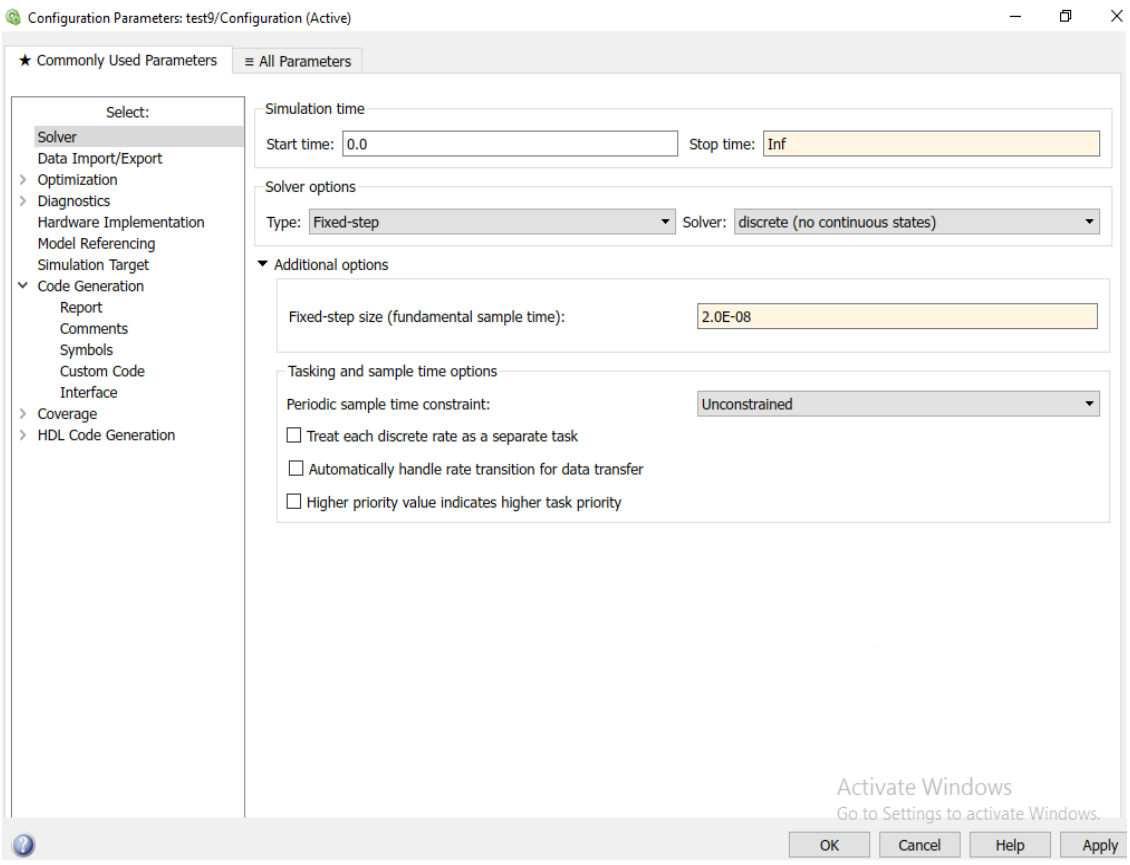

Figure 5.2: Configuration Parameter: Solver

To generate HDL code, under Code select Generate HDL option which will generate HDL code in a folder that is specified in the configuration parameter settings, as shown in Figure 5.1. Simultaneously we can also see the step by step report generation commands on MATLAB command prompt as shown in Figure 5.3.

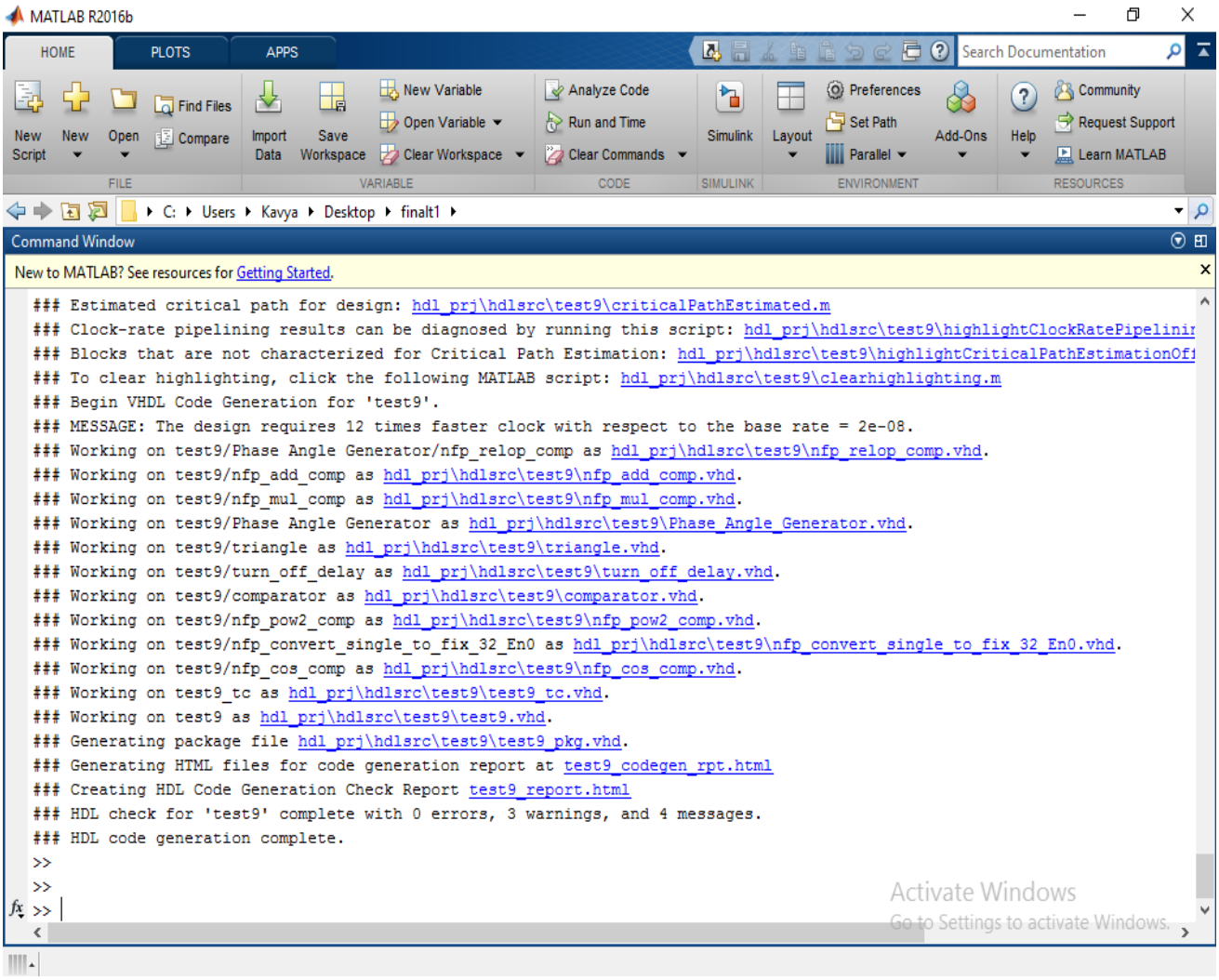

Figure 5.3: HDL Code Generation process shown in MATLAB command prompt

After the code generation completes we can access the code in the path specified. The folder will contain all the VHDL or Verilog files of every subsystem and library files included. A code generation will be formulated at the end of this process which includes code traceability report, resource utilization, optimization, high level timing critical path reports.

5.2 FPGA-IN-THE-LOOP METHOD

The FPGA-in-the-loop (FIL) simulation provides the capability to use Simulink or MATLAB software for testing designs in real hardware for any existing HDL code. The HDL code can be either manually written or software generated from a model subsystem.

Communication Channel: FIL provides the communication channel for sending and receiving data between Simulink and the FPGA. This channel can be a JTAG, Ethernet, or PCI Express connection. Communication between Simulink and the FPGA is strictly synchronized to provide a reliable verification environment.

Open the FPGA-in-the-loop Wizard and for FIL options select FIL simulation with as Simulink as we have our HDL code from Simulink model. Board options are provided with the SOC FPGA Development Board that we have. Figure 5.4 shows the options that were selected for FIL options.

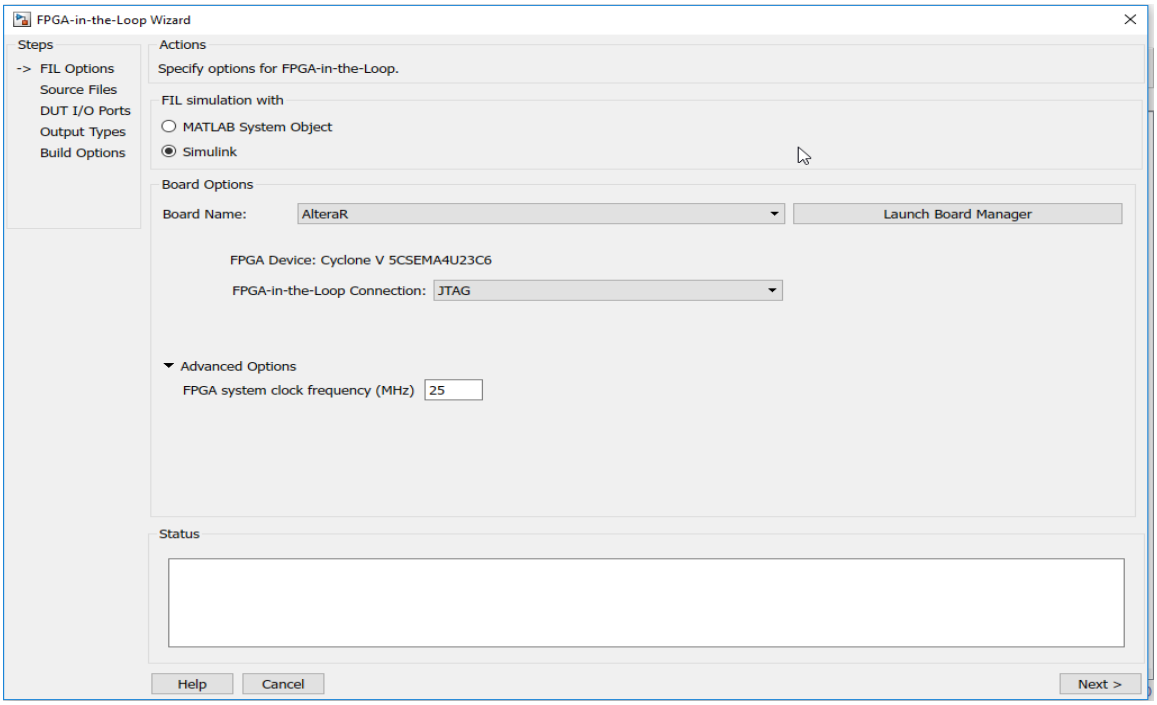

Figure 5.4: FIL options for FIL Wizard

Select the previously generated HDL files from the folder that was specified. Select all the files and mark the Top-Level file to obtain entity and architecture (in the case of VHDL) as shown in Figure 5.5.

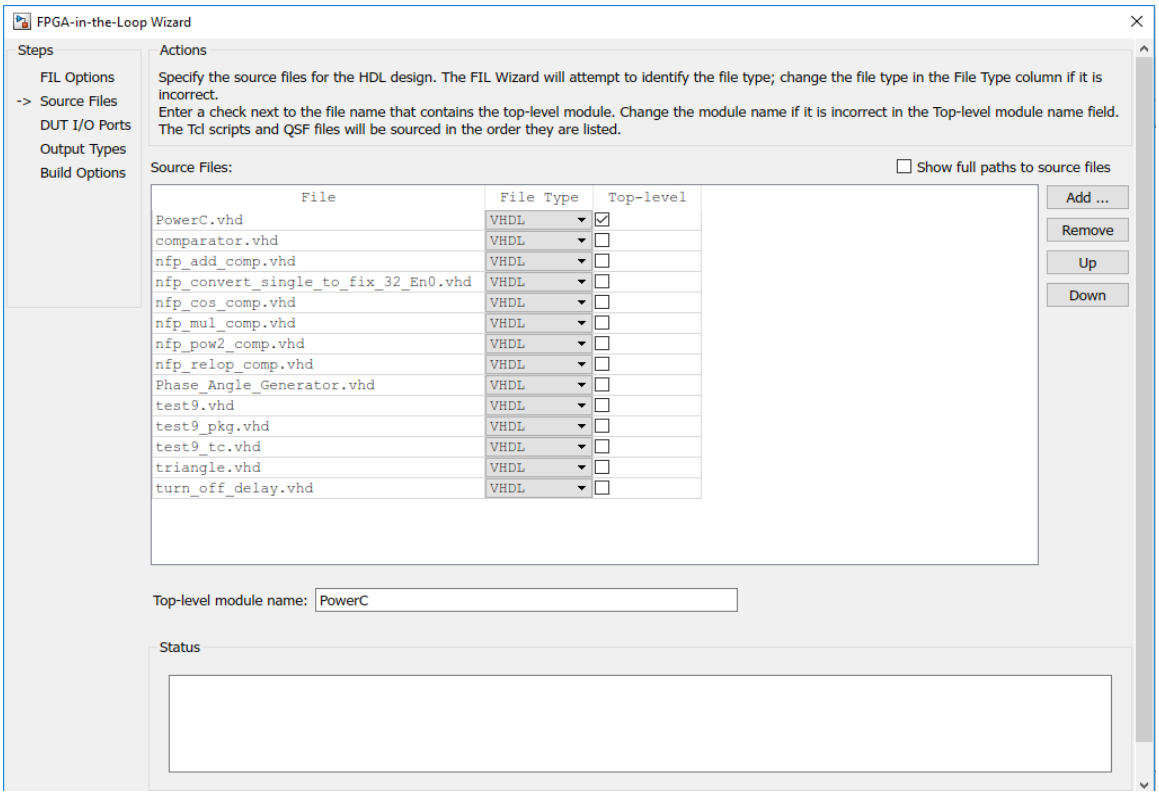

Figure 5.5: HDL Source files

The DUT IO Ports lists the input and output signals from the top level hdl file and these signals can be generated manually or automatically. Along with the signals, the port width to define number of pins to be used, direction – in or out and type – clock, reset, gpio can also be chosen. These information is user specific and for VSI model below Figure 5.6 shows the options that were consider for Input/output ports.

Figure 5.6 shows the output options windows, wherein the output signals can be defined with respect to the bit width (1), data type (Boolean), Sign (unsigned) and fraction length (0), the values inside the brackets are the values chosen for VSI model as outputs are ON/OFF switches.

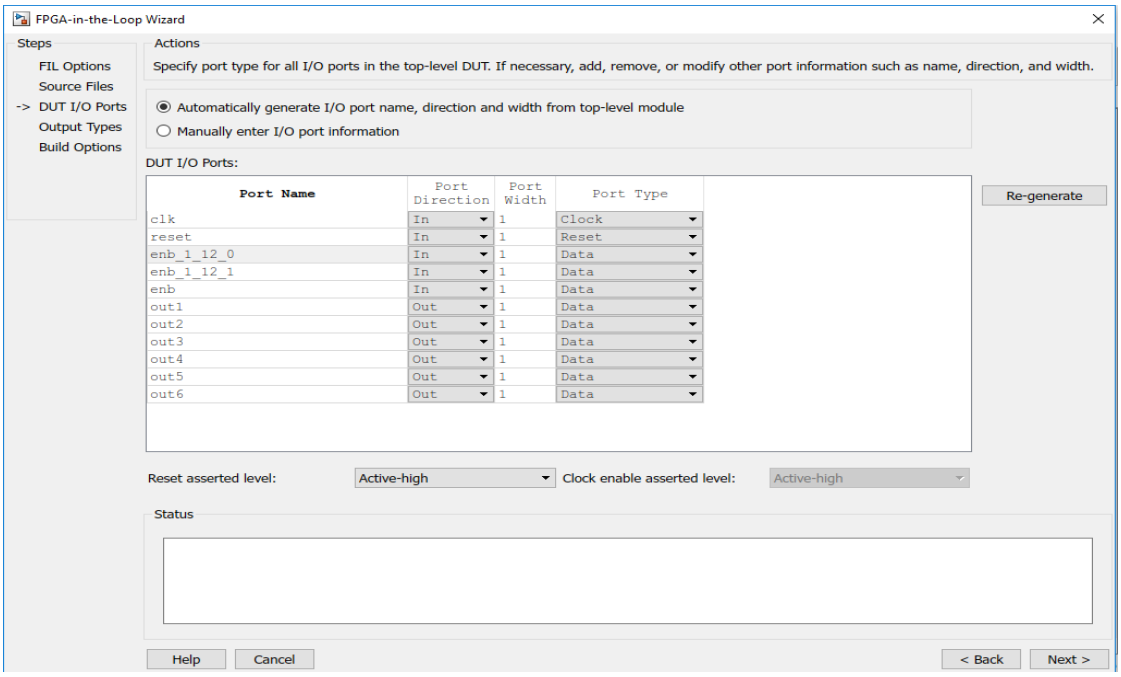

Figure 5.6: DUT IO port option window

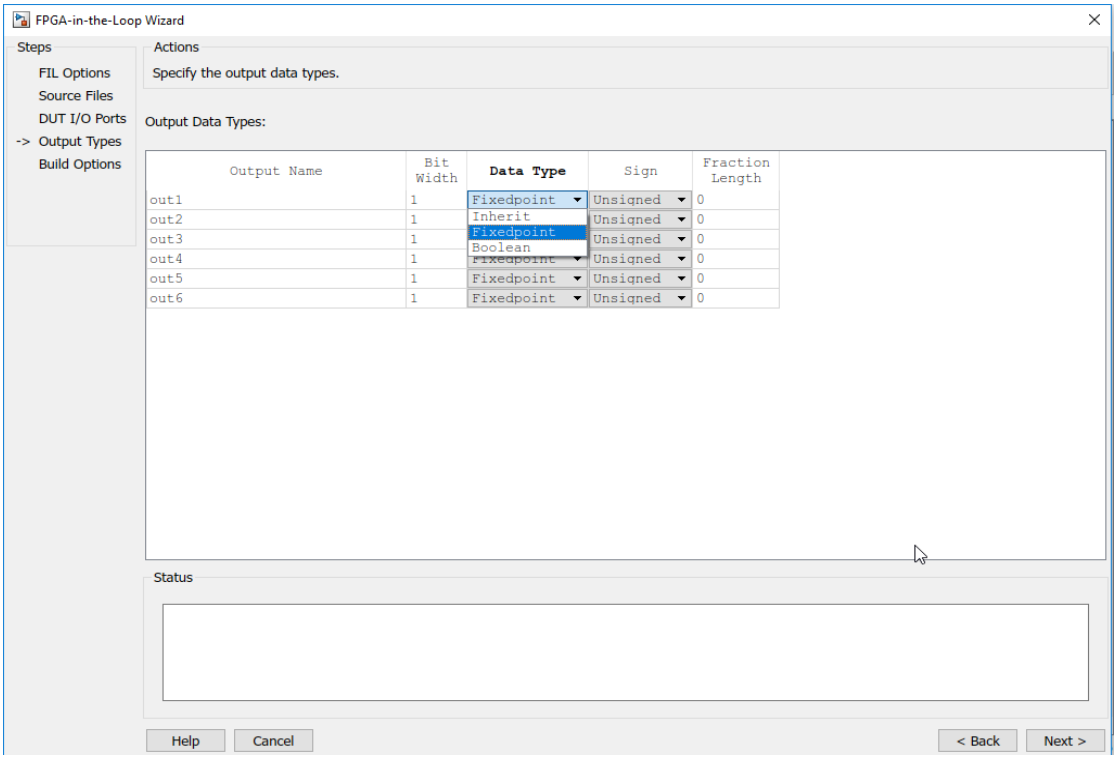

Figure 5.7: FIL output data type window

Next window lets us choose the folder where we would like to store some more HDL files created in the process, along with HDL files QSF which is a Quartus Project file will also be created. With FIL, we can automate the model directly from Simulink without having to open a Quartus tool and to build everything from the scratch. This is the most convenient method for automation.

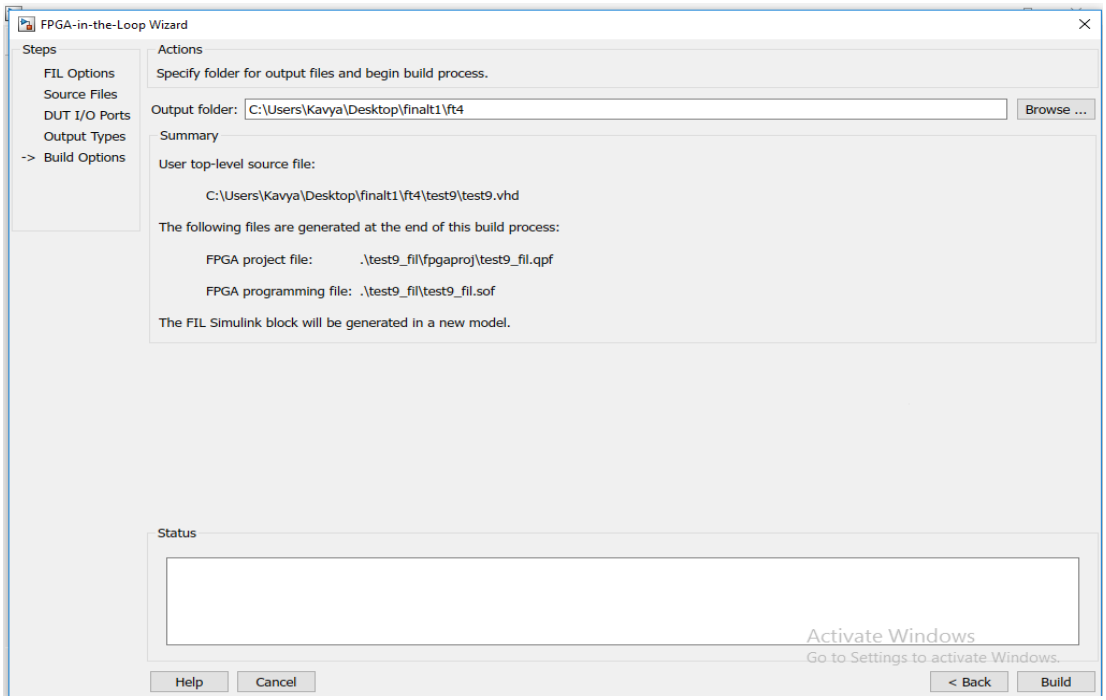

Figure 5.8: FIL Build window

Figure 5.8 below shows a system model that has been newly generated from the FIL process. This is a custom block that can be used in different models to generalize the VSI procedure. This model can be directly loaded onto target FPGA and can also be monitored in real time from Simulink. The FIL model can be connected to Simulink blocks at its output and be verified for its integrity. The newly generated FIL model also comes with a set of instructions to make connections with the hardware and loading the model.

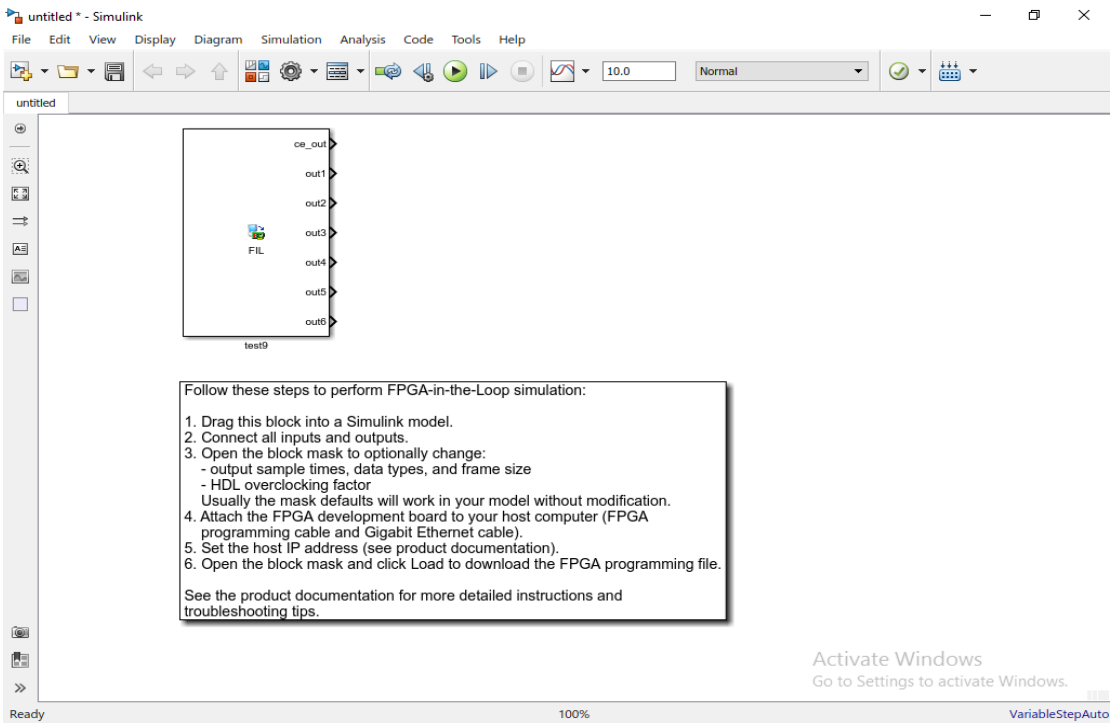

Figure 5.9: FIL system

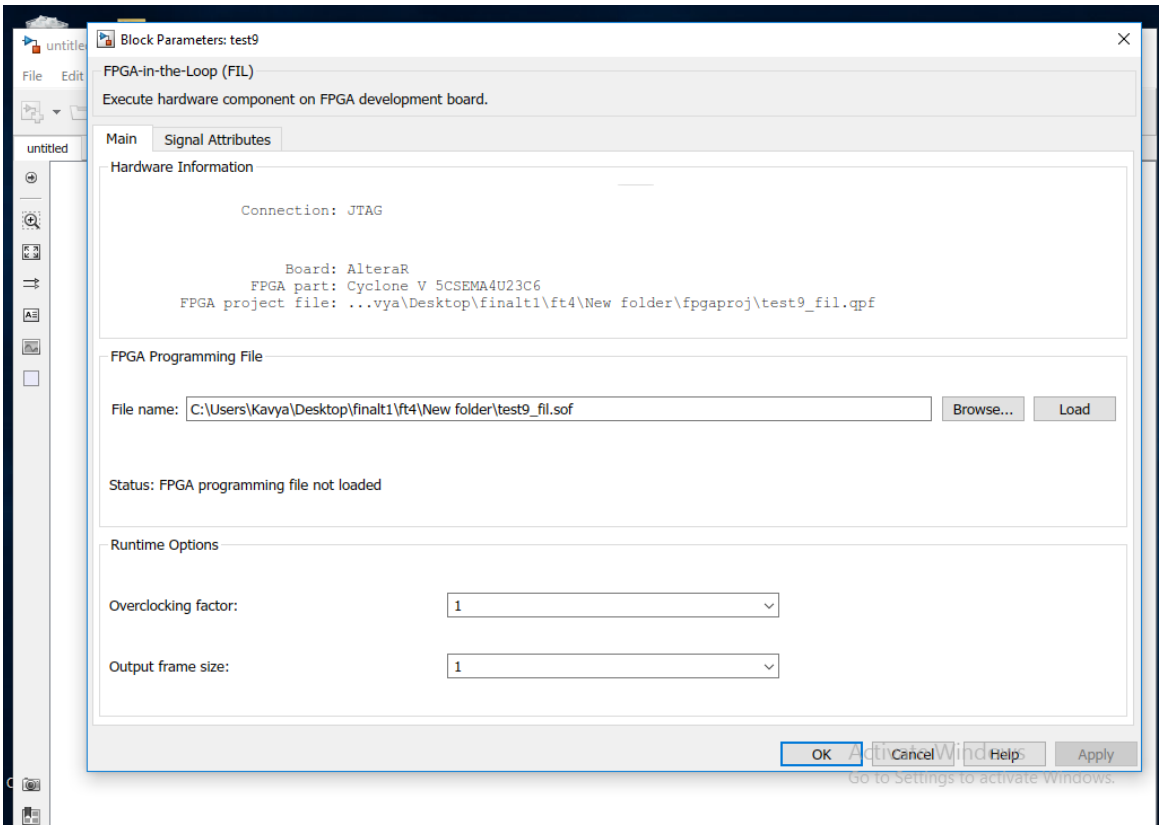

Figure 5.10: Loading the FIL onto the target hardware

By right clicking on the FIL model we can load the VSI model onto a hardware, the above Figure 5.9 shows the description of the target hardware and the file folder.

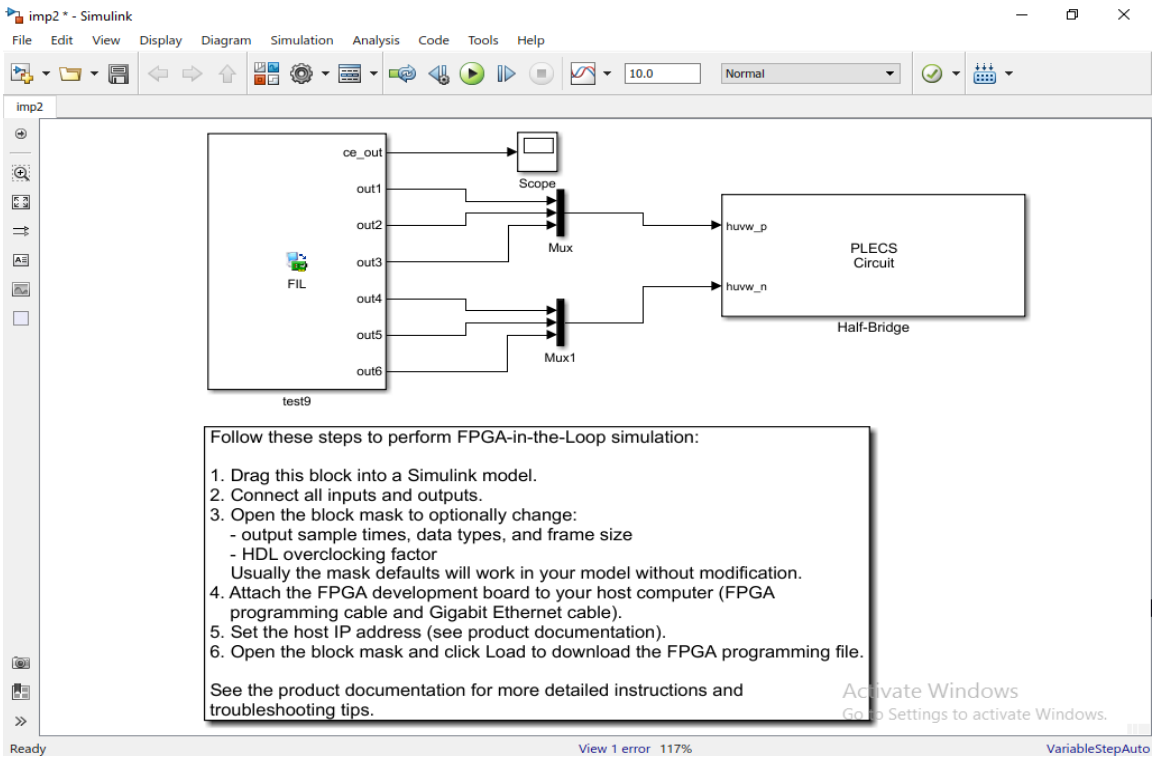

Figure 5.11: FIL subsystem connected to a Half Bridge

Validation and Verification are done by connecting the PLECS subsystem along with the multiplexer block and running the FIL module. Below Figure 5.11 and Figure 5.12 shows the output of the FIL module, there are three 3 phase pwm signals zoomed in and zoomed out waveforms.

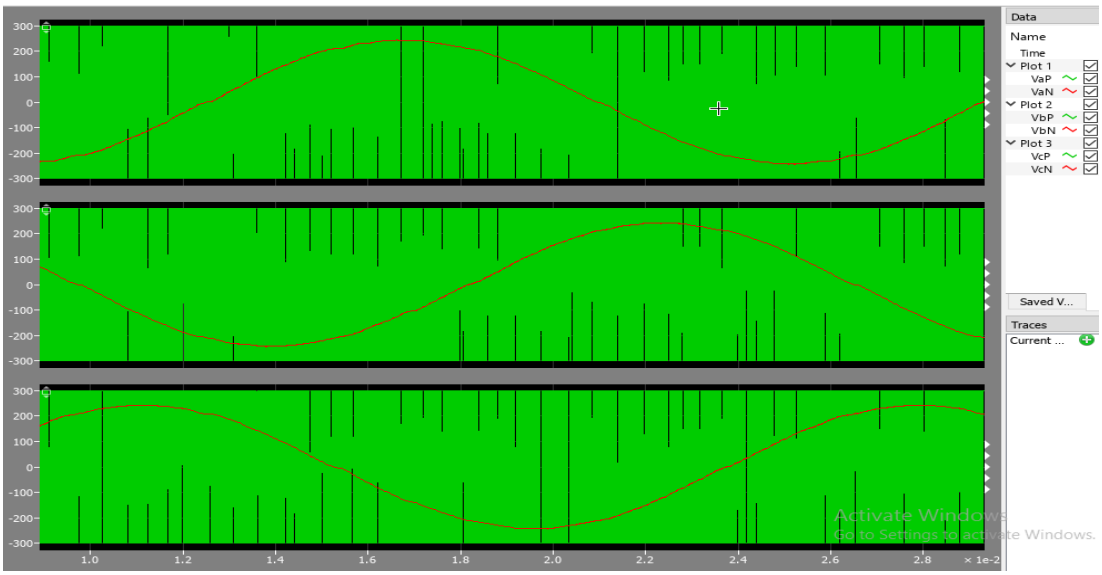

Figure 5.12: Output waveform of FIL, zoomed out

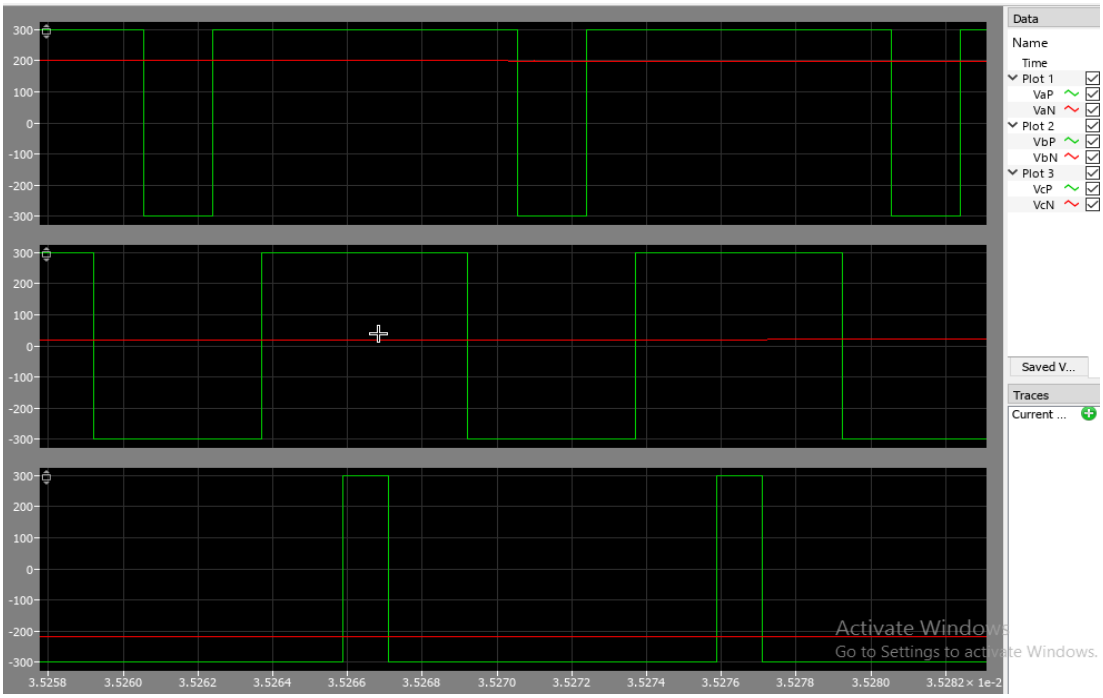

Figure 5.13: Output waveform of FIL, zoomed in

5.2 Interfacing Quartus Pro II: Method 2

The Intel Quartus Prime software provides a complete design environment for FPGA and SoC designs. The user interface supports easy design entry, fast processing, and straightforward device

programming. The Intel Quartus Prime Pro Edition software enables next generation synthesis, physical optimization, design methodologies, and FPGA architectures. The Intel Quartus Prime Pro Edition software provides unique features not available in other Intel Quartus Prime software editions.

5.3.1 Analysis and Synthesis

Analysis & Synthesis considers specific logic as it generates the database and performs logic synthesis and optimizes the design.

As it generates the database, Analysis & Synthesis examines the logical completeness and consistency of the project, and checks for boundary connectivity and syntax errors (for example, mistyped keywords in Verilog or VHDL, or incorrect port names in entity instantiations).

As it generates the database, Analysis & Synthesis also synthesizes and optimizes your design. For example, it infers flipflops, latches and state machines from "behavioral" languages, such as Verilog HDL and VHDL. In addition, the module replaces operators, such as + or -, with modules from the Altera library of parameterized modules (LPM) functions, which are optimized for Altera devices.

5.3.2 Analysis and Elaboration

VHDL designs are hierarchic by nature. The top entity instantiates signals, components and processes. Each component instantiates additional signals, components and processes. The compiler explores the top-down design hierarchy and builds an interconnection table until it reaches the building blocks of the design. At this step, the building blocks are still generic RTL constructs: Logic gates, registers, memories, etc.

5.3.3 Place and Route

It is a stage in the design of printed circuit boards, integrated circuits, and field-programmable gate arrays. As implied by the name, it is composed of two steps, placement and routing. The first step, placement, involves deciding where to place all electronic components, circuitry, and logic elements in a generally limited amount of space. This is followed by routing, which decides the exact design of all the wires needed to connect the placed components. This step must implement all the desired connections while following the rules and limitations of the manufacturing process.

5.3.4 Assembler

This stage generates a programming file and layout the I/O assignments test on the board.

5.3.5 TimeQuest Timing Analysis

Timing analysis measures the delay of a signal reaching a destination in a synchronous system. The TimeQuest Timing Analyzer is a powerful ASIC-style timing analysis tool that uses industry standard constraint, analysis, and reporting methodologies. The Quartus II Fitter optimizes the placement of logic in the device to meet timing constraints.

During timing analysis, the TimeQuest analyzer analyzes the timing paths in the design, calculates the propagation delay along each path, checks for timing constraint violations, and reports timing results as slack in the Report pane and in the Console. You can customize the reports to view precise timing information about specific paths. You can then determine whether the design

requires additional timing constraints or exceptions, or if the design requires logic changes or place and route constraints.

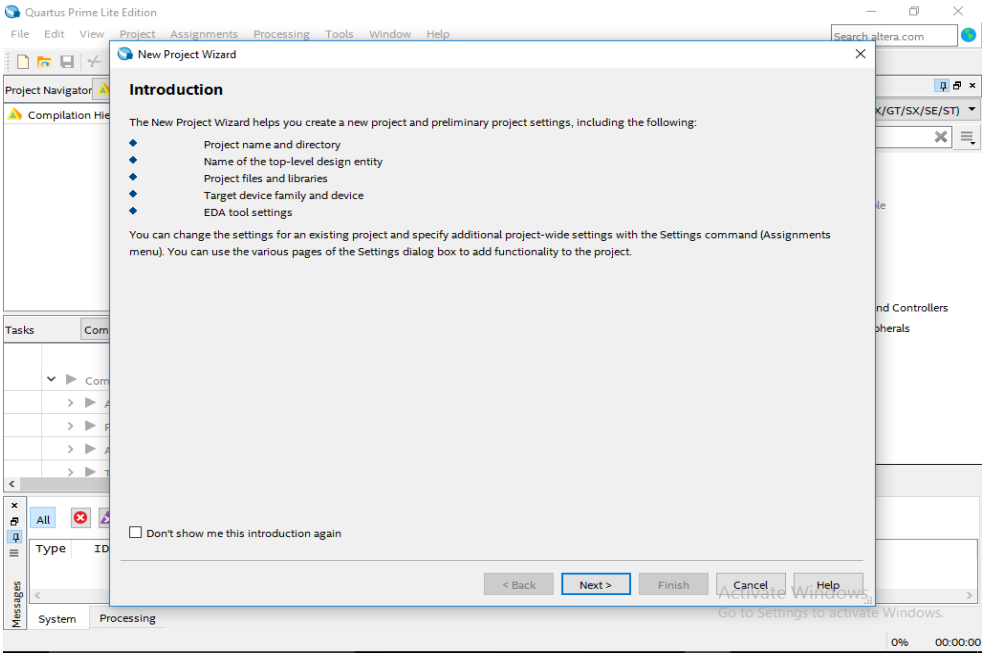

Figure 5.14: Creating New Project on Quartus Prime

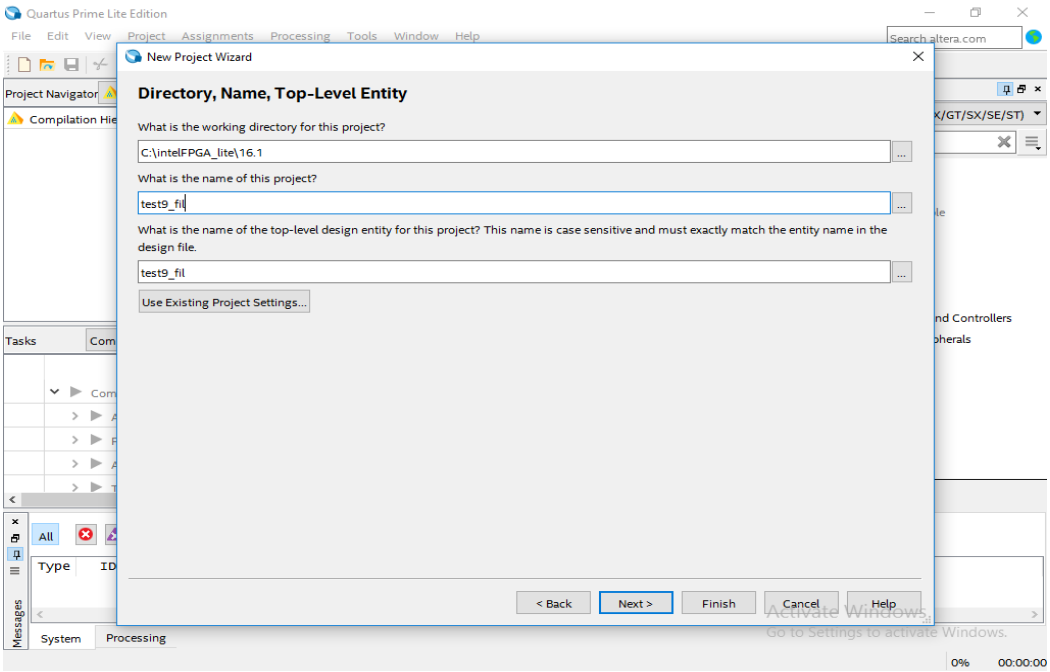

Figure 5.15: Adding name of the top-level entity

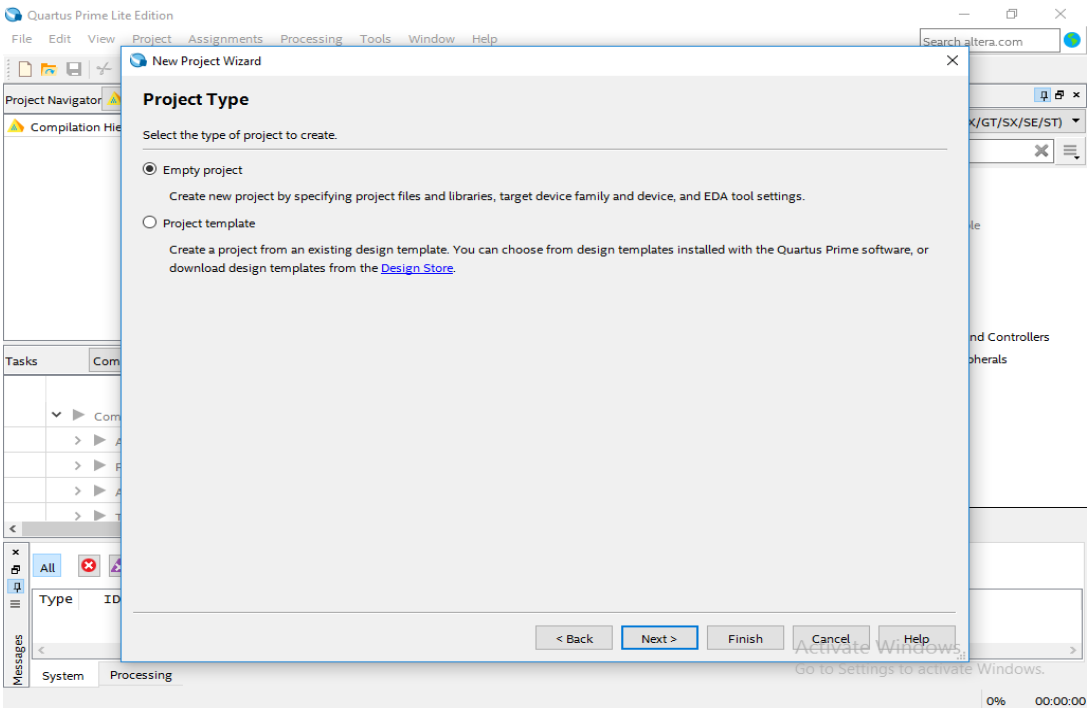

Figure 5.16: creating an empty project to add the HDL generated files

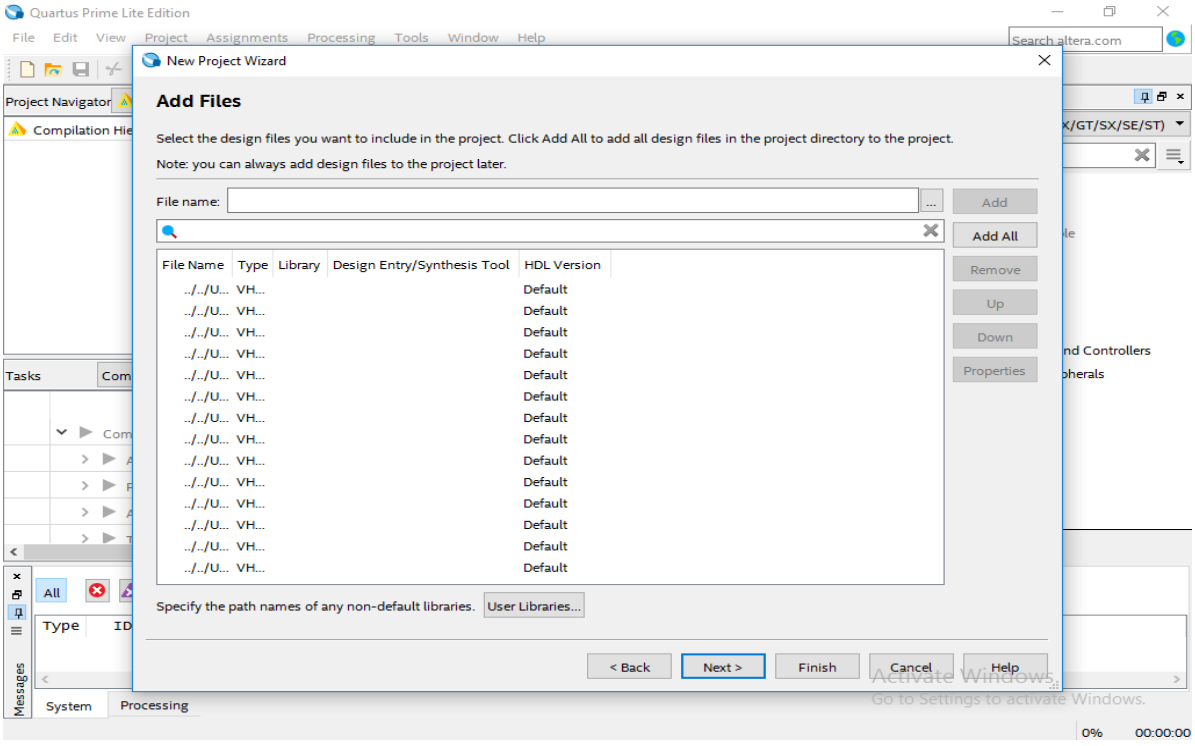

Figure 5.17: Adding the generated files to the empty project

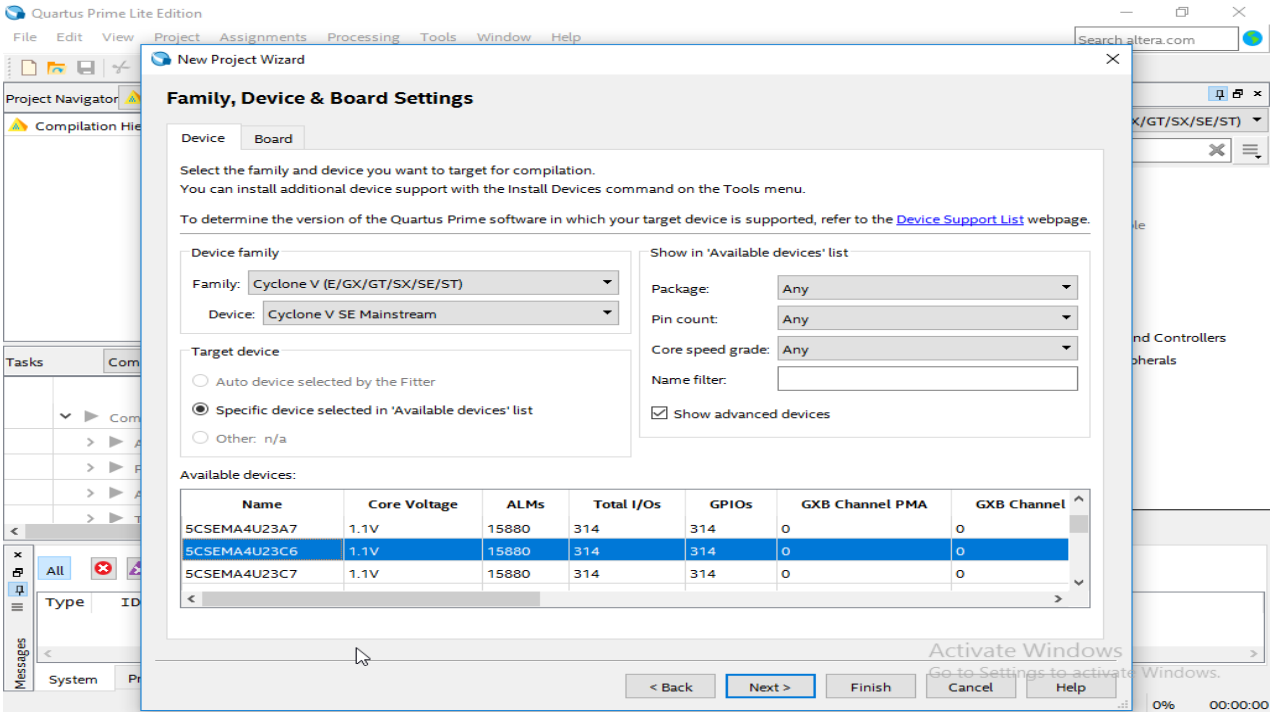

Figure 5.18: Proving FPGA board details

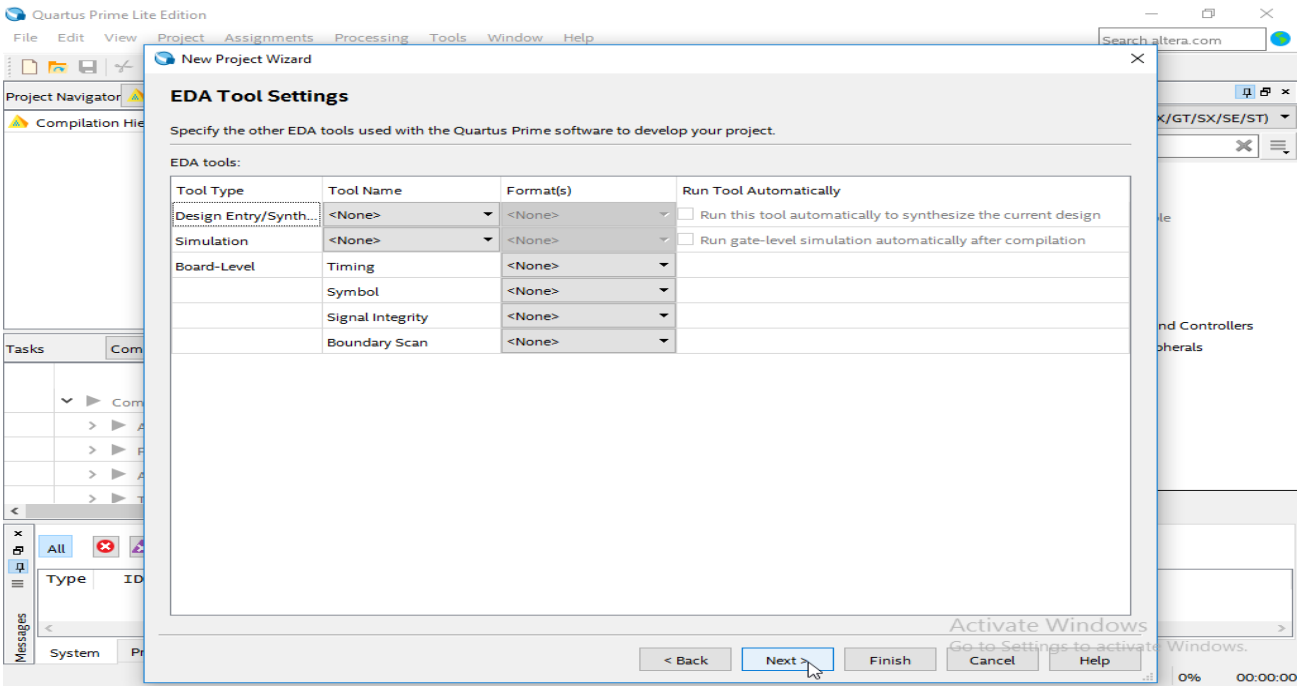

Figure 5.19: EDA settings to run on default quartus tools

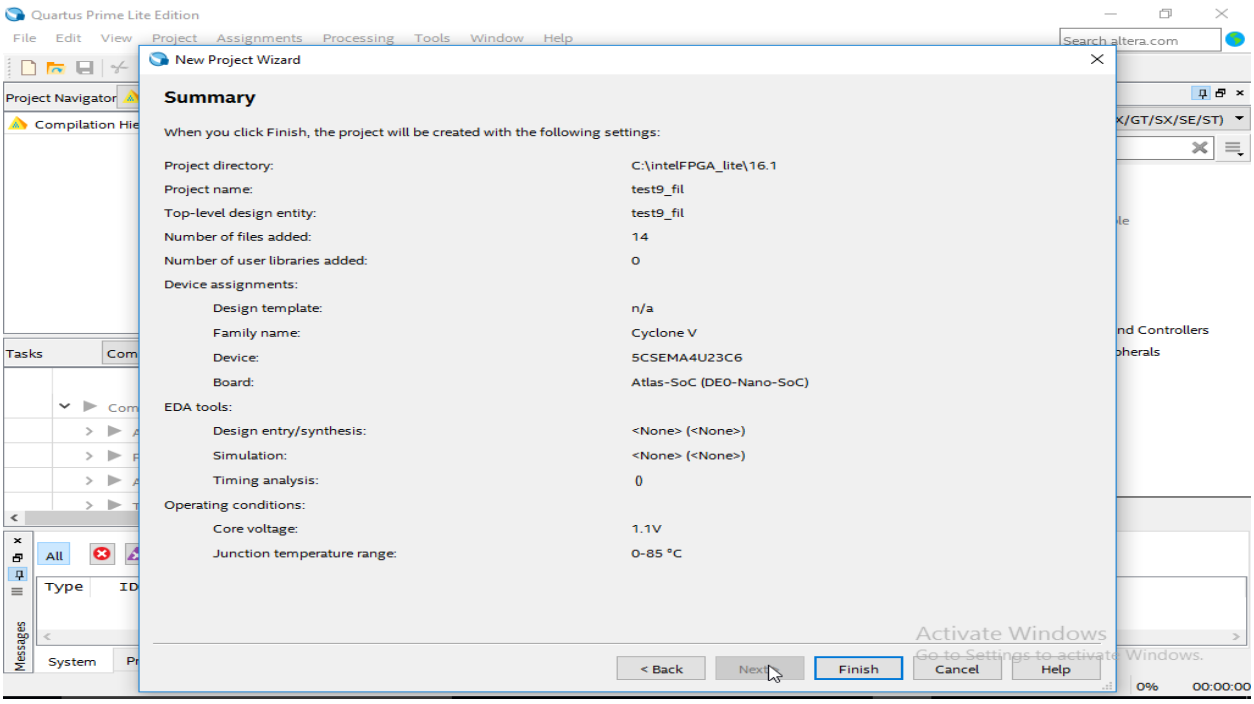

Figure 5.20: Summary of the new project with all the modified details

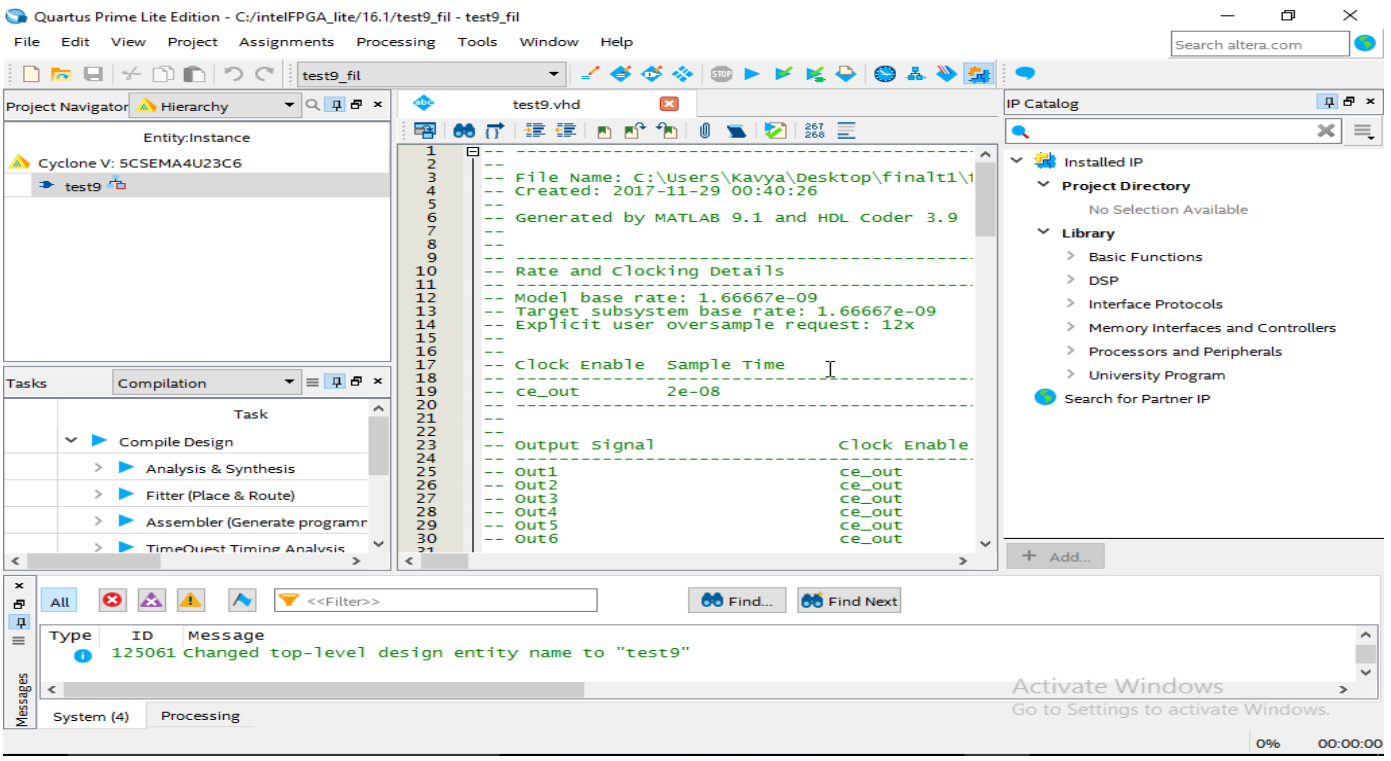

Figure 5.21: View of the Quartus browser

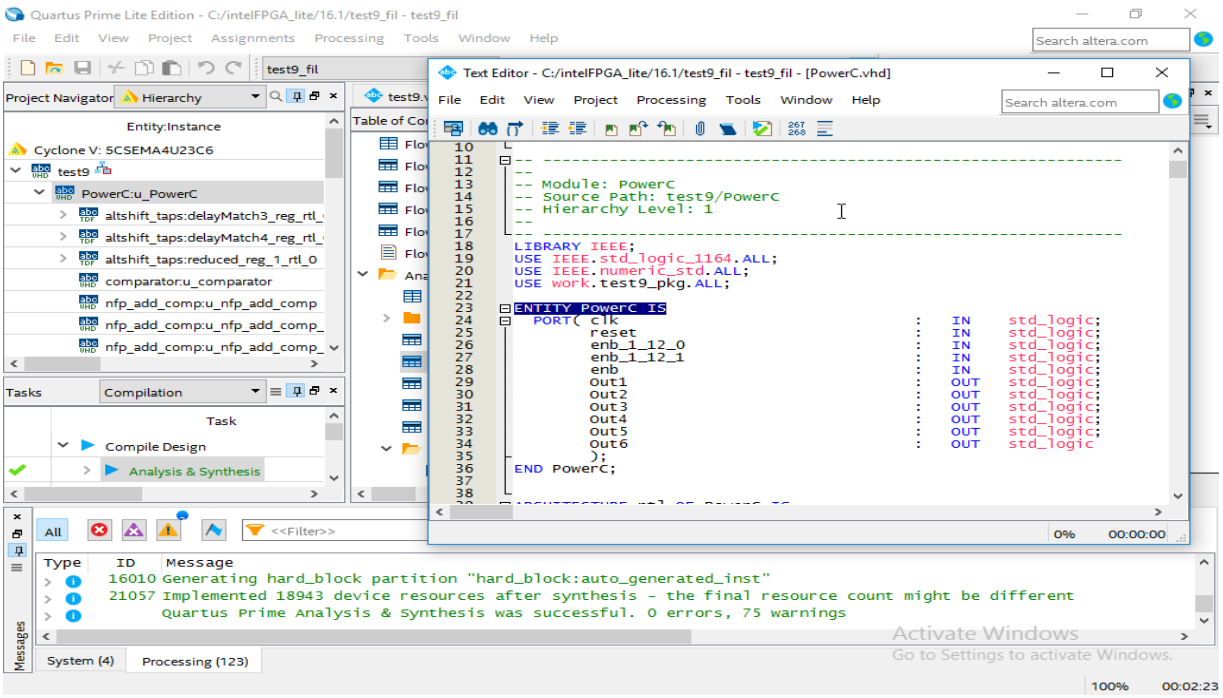

Figure 5.22: Left window has files expanded and right window has a top level entity file open, the 6 switching nodes are out1 through out6.

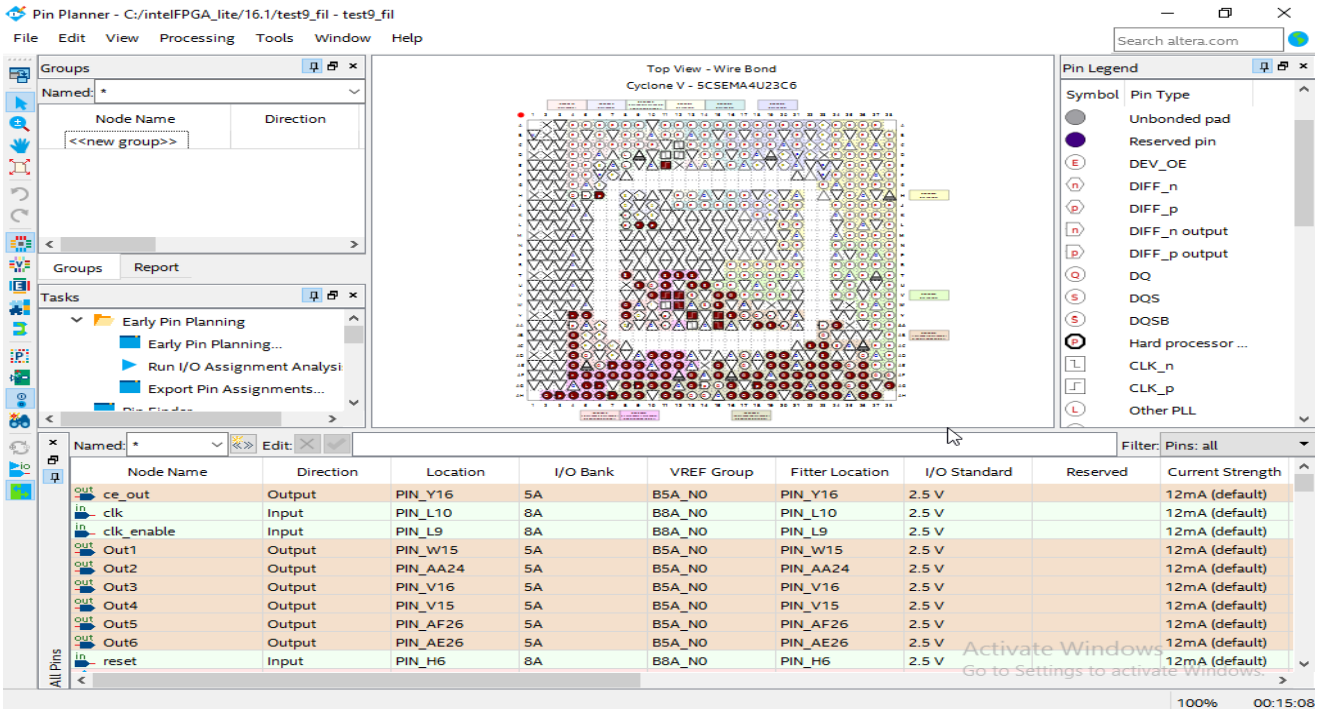

Figure 5.23: Pin Planner to assign pin location

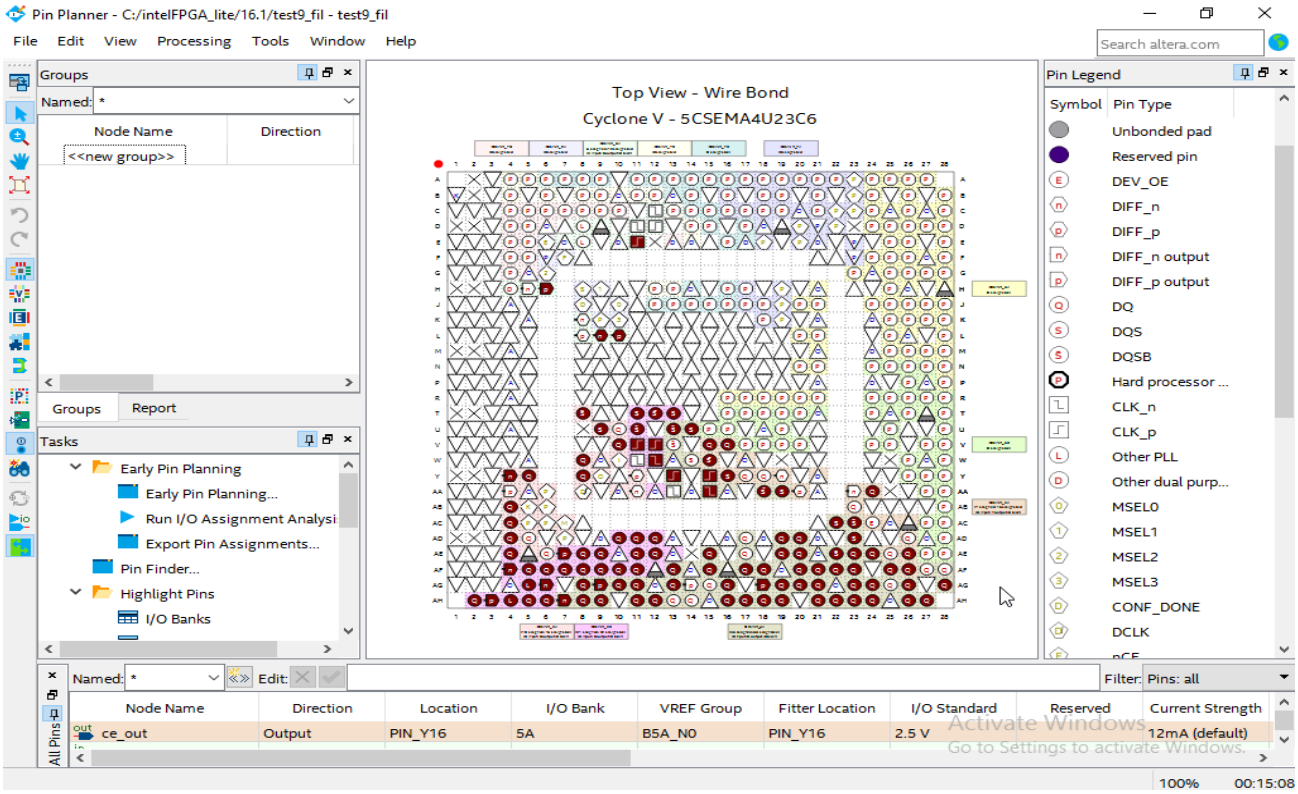

Figure 5.24: Detailed view of the SOC FPGA board on Pin Planner

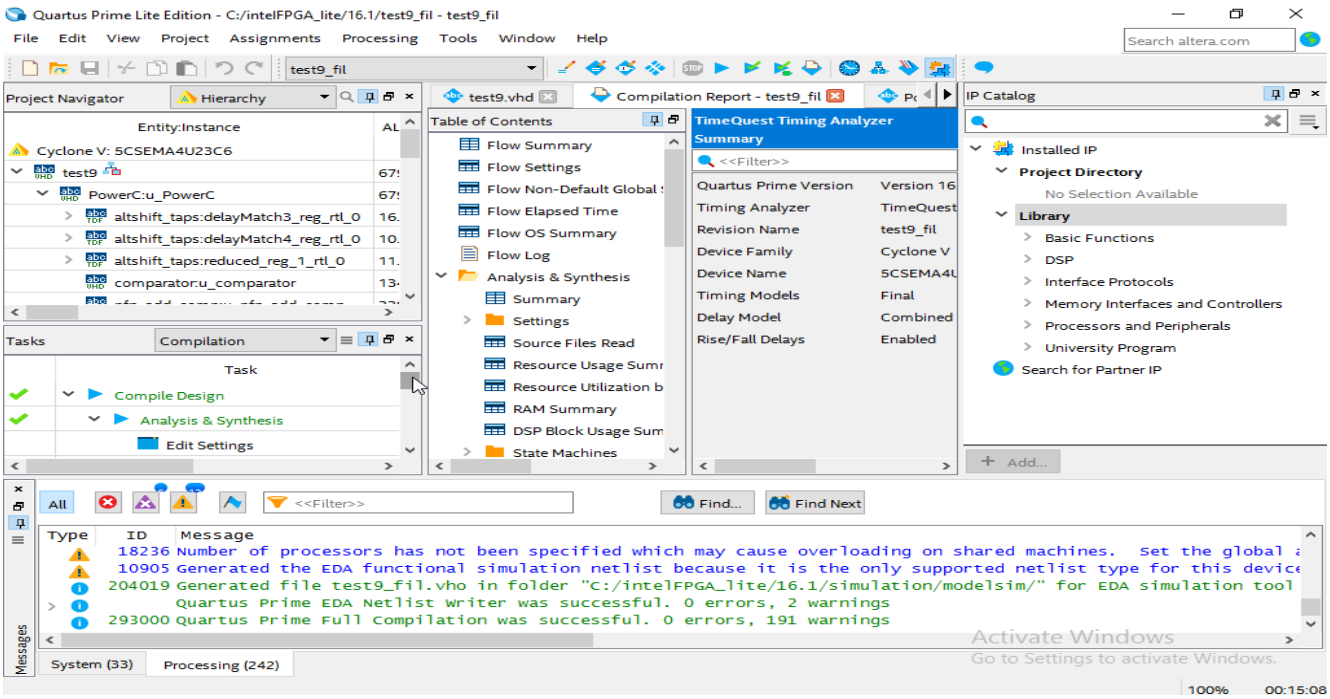

Figure 5.25: Compile the entire design with no errors to create .sof file for FPGA

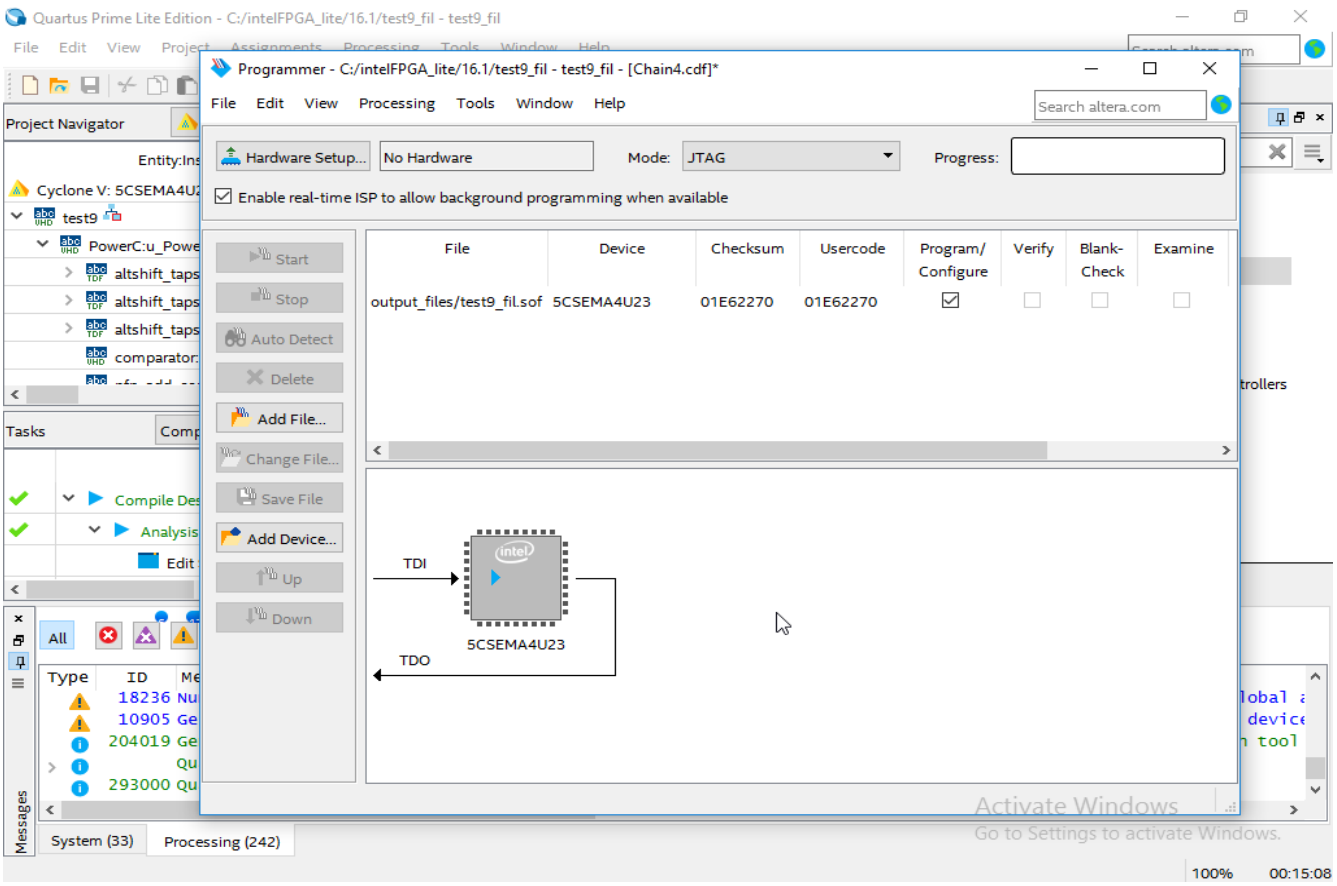

Figure 5.26: View of the Programmer window to deploy .sof file on to hardware.

CHAPTER 6: CONCLUSIONS

I. Simulink/MATLAB provides an excellent platform for the creation of DC micro grid components using various available tools and to simulate them for verification and automation.

II. Simulink has excellent model to target or code to target hardware interface and wide range of FPGAs and ASICs can be considered as target hardware. It has interface with custom FPGA boards to run our very own application specific model.

III. To generate FPGA synthesizable code, HDL coder and its supported blocks should be used to create the model. VSI model used only blocks from HDL coder and stateflow.

IV. As FPGAs work on logic, only single or double data types should be considered. Few blocks such as rate transition blocks can use signed or unsigned integer format to transition the frequency of operation from one subsystem to another.

V. To make the model cloud compatible, the real-time user input blocks should be present outside the FIL subsystem.

VI. The target here is a DE1 SOC FPGA board. A FPGA board should be chosen based on the capacity of RAM, the number of registers available and required clock frequency.

VII. The model can be simultaneously deployed on target hardware and connected to other components on model. The FPGA can provide an output that can further run the other subsystems in line inside the model that is running.

VIII. The automation of code from VSI model can be achieved in 3 different ways.

IX. Method 1: A HDL code can be generated and stored in a folder and further this code can

be used in FIL wizard to create a FIL subsystem. This FIL subsystem provides a direct path to deploy on target hardware.

X. Method 2: A HDL code that is generated and stored in a folder can be called in on Quartus II software. A project can be created in the name of the model and this name should also match the model name. This in turn can be synthesized and analyzed and later deployed onto a target hardware using programmer.

XI. Method 3: Another direct way of automating the model is to use the HDL workflow adviser to generate HDL code, check for compatibility with the FIL and deploy the code on target hardware. These 3 steps take place in a single process when HDL workflow adviser wizard is invoked.

XII. The FIL model synthesized from method 1 can be used as a custom subsystem in other model which require a VSI component in its design.

CHAPTER 9: FUTURE WORK

1. The same set of procedures can be applied to create other component models.
2. The model input parameters can be connected to cloud to change parameters such as slider gain, switching frequency etc.
3. The FPGA can be connected to cloud and monitored via cloud to test the redeployments.
4. The FPGAs can be made intelligent by employing data mining algorithm, so that it can be taught to take few decisions at low priority failure.
5. Every component in the grid can be made accessible to the test engineer, when at any priority failure, the node of failure can be easily found.

REFERENCES

[1] Cuzner, R. M.; Palaniappan, K.; Shen, Z. J., “System Specification for a DC Community Microgrid and Living Laboratory Embedded in an Urban Environment,” International Conference on Renewable Energy Research and Applications (ICRERA), November 2015

[2] FIL Simulation with HDL Workflow: https://www.mathworks.com/help/hdlverifier/ug/fil-simulation-with-hdl-workflow-advisor-for-matlab.html

[3] DC Micro Grid and the virtues of local electricity: https://spectrum.ieee.org/green-tech/buildings/dc-microgrids-and-the-virtues-of-local-electricity

[4] Mini S. Thomas, Vinay Kumar Chandna, Seema Arora, "Indoor power line channel characterization for data transfer and frequency response measurements", *Sustainable Mobility Applications Renewables and Technology (SMART) 2015 International Conference on*, pp. 1-8, 2015

[5] A. P. Mello, M. Sperandio, D. P. Bernardon, L. L. Pfitscher, L. N. Canha, M. Ramos, D. Porto, R. Pressi, "Intelligent system for automatic reconfiguration of distribution network with distributed generation", *Power Engineering Energy and Electrical Drives (POWERENG) 2015 IEEE 5th International Conference on*, pp. 383-388, 2015, ISSN 2155-5532.

[6] Automatic code generation: https://www.ethz.ch/content/dam/ethz/special-interest/mavt/dynamic-systems-n-control/idsc-dam/Lectures/Embedded-Control-Systems/OtherNotes/Automatic_Code_Generation.pdf

[7] Simulink, Simulation, Code generation: http://retis.sssup.it/~marco/files/lesson23_Simulink.pdf

[8] Embedded coder: https://www.mathworks.com/tagteam/72312_embedded-coder.pdf

[9] HDL coder, users guide: http://cn.mathworks.com/help/pdf_doc/hdlcoder/hdlcoder_ug.pdf

[10] HDL coder: https://www.mathworks.com/tagteam/72320_hdl-coder.pdf

[11] Muhammad Asadullah, Ahsan Raza, "An overview of home automation systems", *Robotics and Artificial Intelligence (ICRAI) 2016 2nd International Conference on*, pp. 27-31, 2016.

[12] Garima, Nidhi Agarwal, S. R. N. Reddy, "Design & development of daughter board for Raspberry Pi to support Bluetooth communication using UART", *Computing Communication & Automation (ICCCA) 2015 International Conference on*, pp. 949-954, 2015.

[13] Andrew Wightwick, Basel Halak, "Secure communication interface design for IoT applications using the GSM network", *Circuits and Systems (MWSCAS) 2016 IEEE 59th International Midwest Symposium on*, pp. 1-4, 2016, ISSN 1558-389

[14] Simulink Code Generation, a MATLAB/SIMULINK Document file: https://www.mathworks.com/matlabcentral/answers/uploaded_files/56491/Simulink%20Code%20Generation.pdf