USING CHIPSCOPE WITH PROJECT NAVIGATOR TO DEBUG FPGA APPLICATION

Jyothirmai Palle (P.No - 870814 0564)

Sowmya Kurella (P.No - 870406 0949)

This thesis is presented as a part of degree of Master of Science in Electrical Engineering

Blekinge Institute of Technology

May, 2011

Blekinge Institute of Technology

School of Engineering:

Department of Electrical Engineering

Supervisors: ***Anders Hultgren, Carina Nilsson***

Examiner: ***Jörgen Nordberg***

Contact Information:

Authors:

1. Jyothirmai Palle

Email: jyothirmai.aksha@gmail.com, jypa09@student.bth.se

2. Sowmya Kurella

Email: sowmyakv6@gmail.com, soku09@student.bth.se

University Advisor(s):

1. Anders Hultgren

Email: anders.hultgren@bth.se

2. Carina Nilsson

Email: carina.nilsson@bth.se

School of Engineering	web: www.bth.se/tek
Blekinge Institute of Technology	Phone: +46 457 385 000
SE-371 79, Karlskrona	Fax: +46 457 271 29
Sweden	

ABSTRACT

In the technological advancement of FPGAs - Field Programmable Gate Arrays, there have been many paths for DSP applications where FPGA has many advantages in providing optimal device utilization adaptability and high design flexibility. FPGAs allow appropriate customization of hardware at acceptable prices. They are cost as well as time saving when making configurations of standard products. FPGA technology is indispensible in regard with long-term availability and harsh industrial environments.

Basing on these features Alstom Power AB, Växjö (Mr. Sonny Bui) designed a controller using VHDL coding which has been implemented using Xilinx ISE on a FPGA board (Virtex-II Pro Board, FG256). Chipscope Logic Analyzer is used to capture the logic in VHDL coding and display all the internal node signals of the circuit using JTAG interface. Most of the thesis work deals with effective working and understanding of Chipscope technology.

ACKNOWLEDGEMENT

We wish to express our immense gratitude to our supervisors Mr. Anders Hultgren and Ms. Carina Nilsson of Blekinge Institute of Technology for their constant support and motivation. We would like to thank Carina Nilsson for her encouragement and valuable time with all her will and guidance.

We are thankful for our parents and friends who rendered their support throughout our thesis project. We would also like to extend our thanks to Alstom Power AB, Växjö which provided us with all the required equipments and environment.

CONTENTS

1. INTRODUCTION

Alstom Power AB design and produce electrostatic precipitators, ESPs, for cleaning of industrially polluted gases. Research, development and production of the control and electrification systems of ESPs are conducted at Alstom's Växjö site. Providing electrostatic precipitators based on high frequency technology, Alstom runs research projects on development of precipitator controller and observer with Blekinge Institute of Technology and Växjö University. Our thesis work is part of different studies concerning implementation methods of the developed control algorithms. So these ESP control systems consist of power converters which include three main parts namely- controller, power converter and an observer [2].

In this paper one controller, developed and implemented in the research project, is implemented using Xilinx Chipscope Logic Analyzer in order to better understand the features of the Chipscope technology for real time debugging. The main contribution of the controller is to compare the incoming information with a set of points and adjust system accordingly. The Fig 1.1 gives the overview of a part of the control system for an ESP.

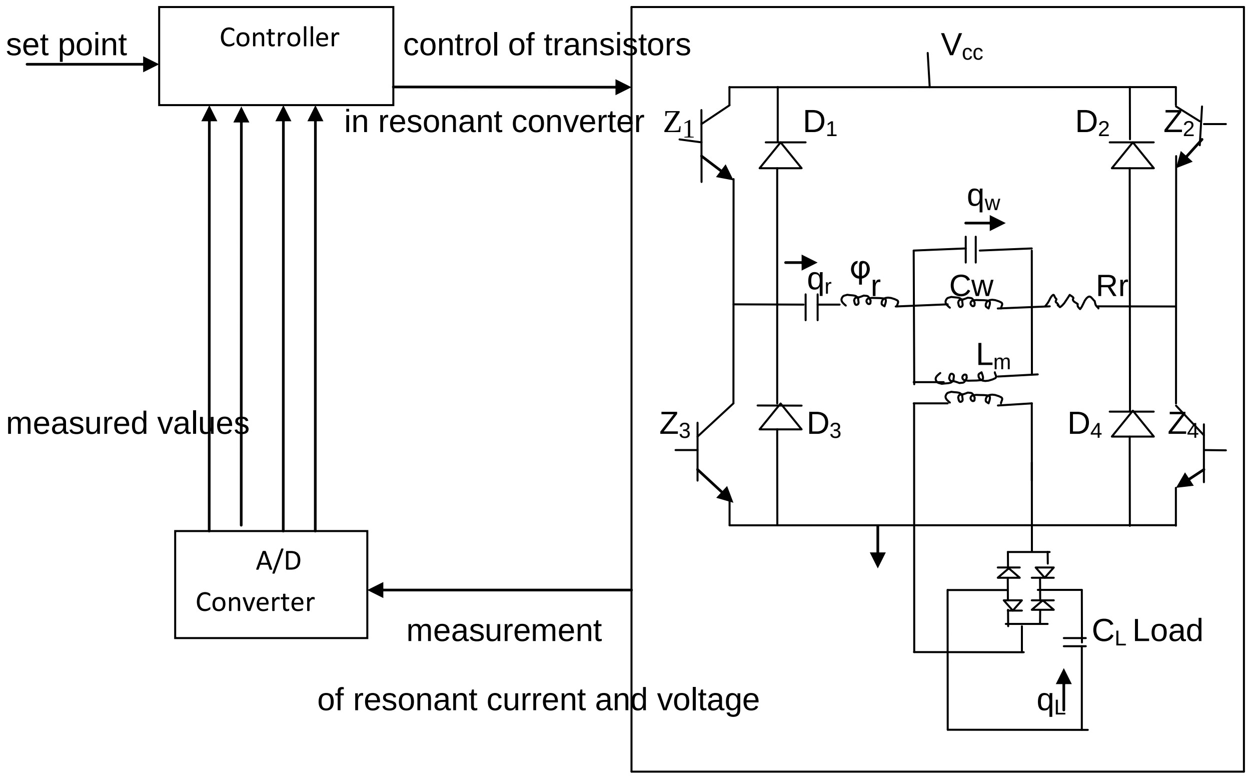


Fig 1.1: Overview of part of the control system for an ESP

The implemented controller calculations are algebraic and logic calculations including a state machine. A resonant converter is described by four different states based on resonant current, voltage across capacitor, voltage across leakage capacitance and voltage across load.

In this context, controller is a state feedback controller. On estimated values that are read from the observer, the controller switches the different states in space. Those are: I = -1, 0, 1, and basing on different states it provides the system with positive or negative voltage [3].

2. XILINX DEVELOPMENT BOARD

2.1Objective:

The main objective is to implement a controller using Xilinx ISE 8.2i on FPGA board (VIRTEX-II Pro, xc2v100 FG256 chip). It is a design which is fully simulated and requires board level testing. JTAG interface acts like communicator between FPGA and PC.

This report supports users how to use Xilinx Project Navigator and Chipscope for different methods to perform calculations on signals inside FPGA. The other main task is to show a user how to integrate the Chipscope tool into a complex design in an efficient manner to authenticate its operation**.**

2.2 Software:

In the thesis project a development board from Xilinx is used: Xilinx ISE 8.2i on FPGA board (VIRTEX-II Pro, xc2v100 FG256 chip). Xilinx ISE offers analysis tools for both design implementation and design concept. The ISE tools help to reach optimal design results with timing predictability and reconfiguration with greater flexibility and size. Xilinx ISE maps the design over the available chip area. Once after generating the bit file and loading it on to the kit, we have an access to analyze the design using Chipscope Pro Analyzer.

Larger the silicon area, the longer is the design time and high is the cost. In order to eliminate these serious causes, a good vendor of FPGA products, XILINX introduced a new revolutionary testing tool for large circuits. This is what we call Chipscope Logic Analyzer which is connected to VHDL code and implemented with the circuit under test on the same FPGA.

2.2.1 Chipscope Organization and its Importance:

Chipscope is a complete solution for both software and hardware. It doesn’t depend upon test bench. To capture signal samples the Integrated Logic Analyzer (ILA) tool is used and Integrated Bus Analyzer (IBA) for buses.

ILA: This is a core which allows the user to view and trigger on signals in the hardware design. One ILA supports one clock input, 64 internal trigger signals and one external signal. Maximum data/trigger width is 64 bits. There is a possibility of storing multiple ILA’s on same FPGA. Each ILA core uses

32Kbits of capture storage for Virtex-II family. IBA cores are used to monitor system buses.

ICON: The second type of core is called Integrated Control core (ICON). Internal RAM signals are exported using ICON core. ILAs use a common ICON to communicate through JTAG interface with the PC. This interface is used to configure the architecture of FPGA. These cores are very useful to manipulate signals directly from hardware during run time [1].

The third component is Chipscope Logic Analyzer. It has 3 features:

- Configuration of ILA's and ICON
- Transferring data samples to the PC
- Analysis of stored signal samples

Each ILA data input has a node associated with tested circuit. The Chipscope Logic Analyzer has different options for displaying the captured signals. The data can be taken in any base types like binary, decimal, hexadecimal, octal or ASCII [4].

2.2.2 Chipscope Pro Debugging Overview:

Chipscope Pro software is used to perform verification inside a circuit. It follows a general procedure of inserting the Chipscope Pro cores in the design using either Core generator or Core Inserter. Then the design is implemented using the Project Navigator and the device is configured. After the device configuration, the design is analyzed using the Chipscope Pro Analyzer.

In this project we have used Chipscope Pro Core Inserter flow. The main reason behind using the Chiscope Pro Core Inserter is the core generation and insertions are carried in one step. The VHDL codes also do not require any modification to insert the core in the code. If there is any need to exclude the core or generate a new core, the CDC file in the Project Navigator can be removed and a new one can be generated.

2.3 Hardware:

The hardware used is a prototype board equipped with an FPGA called xc2v1000-FG256from the Virtex-II family whose speed is set to -6. The board has a parallel IV cable which is used for JTAG configuration as shown in Fig 2.1.

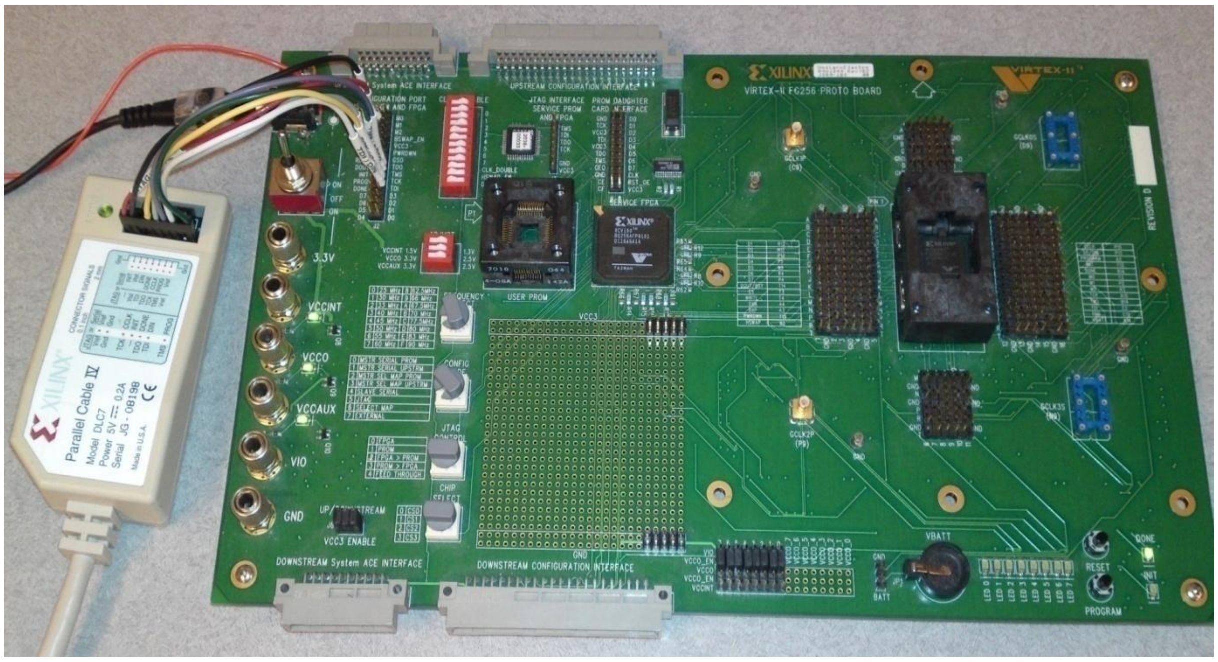


Fig 2.1: Lab equipment, i.e. the used Xilinx development board.

3. CONTROLLER STRUCTURE

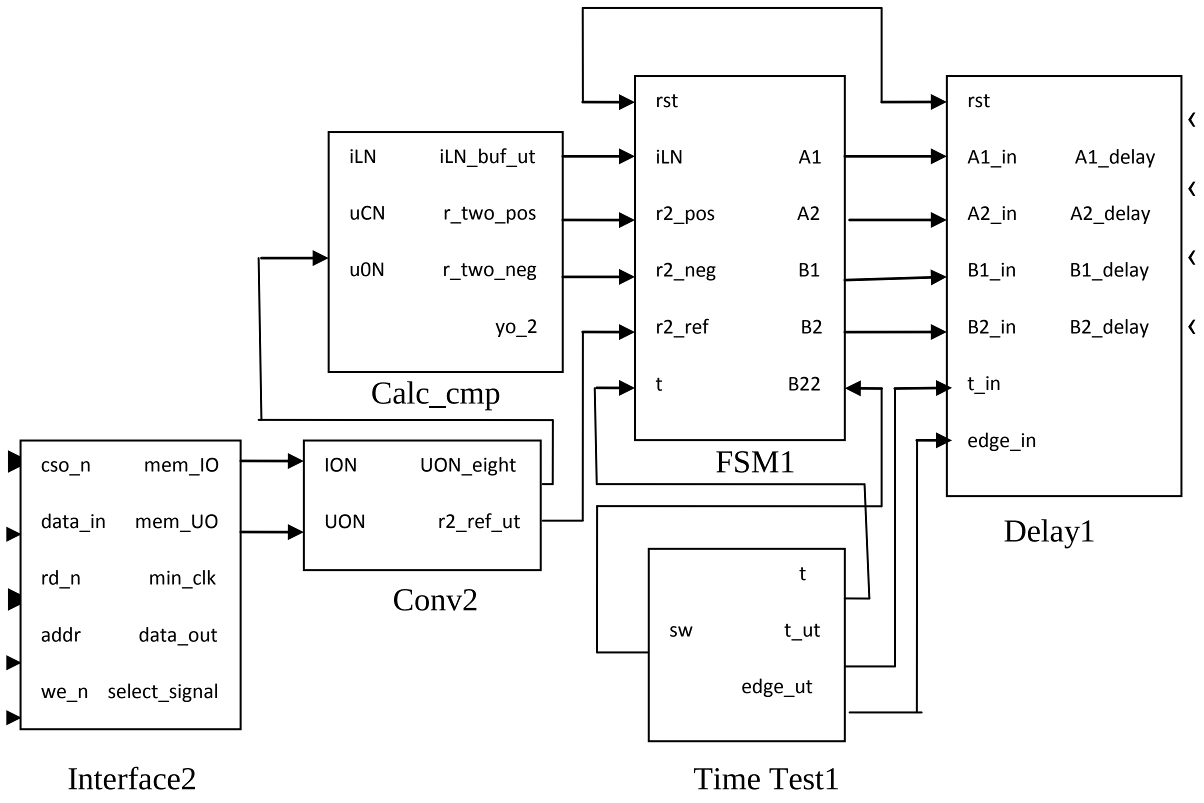


Fig 3.1 Design of Controller

The programming of this FPGA controller was made by Mr.Sonny Bui of Alstom Power AB, Växjö. The function of each block of the controller shown in Fig 3.1 is described below:

3.1 Interface2: This is the very first design block in this hierarchy which acts like communication between FPGA and digital signal processor. This part provides resonant voltage, ULN and the resonant current ILN to the FPGA.

3.2 Conv2: Conv2 looks for a reference value from the values stored in Interface2.

3.3 calc_cmp: Normalizing calculations of values ILN and UCN are done by the calc_cmp. Upcoming reference value of conv2 is compared by squaring and adding of ILN and UCN values.

3.4 FSM: FSM evaluates the values of R2 with R2 reference values. Depending upon the comparison, the state of the system is determined. There are 4states. Each state has a maximum and a minimum time. In these cases, if the time exceeds the maximum limit, the system will be asked to stop the state. The physical system's positive or negative voltage depends upon the state of the system.

3.5 Time Test: The responsibility of observing track of timers between FSM and delay is maintained by Time Test.

3.6 Delay: There is a constant maintenance of 1μs between opening of two switches and closing of next two corresponding switches where all switches are open. This makes the system to disperse remaining energy and prevent shortcuts [2].

3.7 Process: The design entry in the Project Navigator looks like as in Fig 3.2.

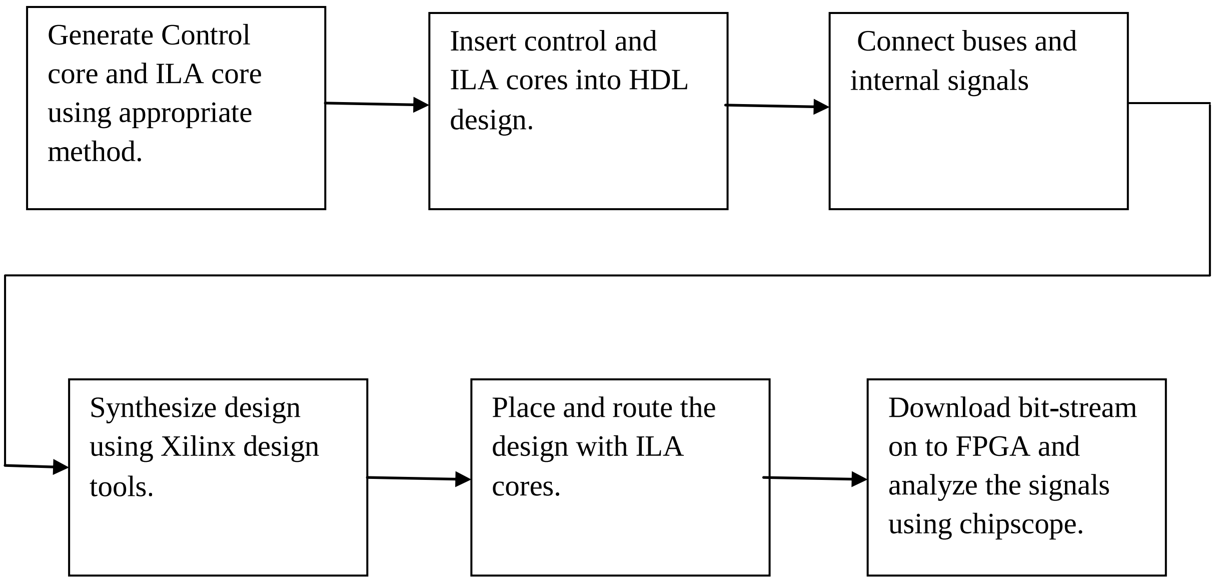

Generate Control core and ILA core using appropriate method.
Insert control and ILA cores into HDL design.
Connect buses and internal signals
Synthesize design using Xilinx design tools.
Place and route the design with ILA cores.
Download bit-stream on to FPGA and analyze the signals using chipscope.

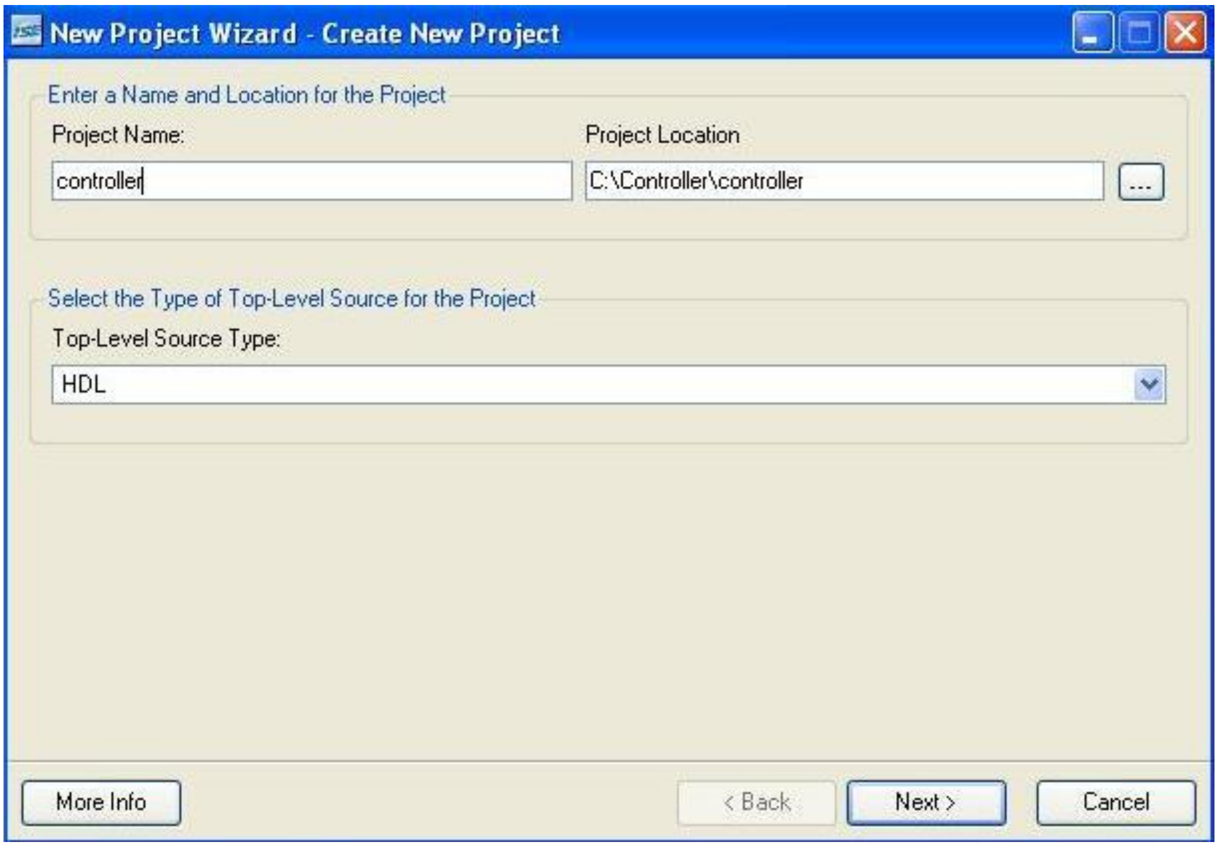
Fig 3.2: Design entry in Project Navigator

3.8 Using Project Navigator:

Digital design logics on FPGAs are mostly implemented using Xilinx Project Navigator. Managing complex designs and coordinating are the most important features of this Project Navigator. It has an easy defined usage with the convenience tools to capture the complete information about the design with timely decisions. It has an easy access to architecture wizards, language templates, graphical tools to analyze design etc. Design entry in the Project Navigator has the following steps:

3.8.1 Creating a New project:

1. Open Project Navigator and select **File →New Project**. Fig 3.3 refers to the naming of the new project and selecting its location.


New Project Wizard - Create New Project

Enter a Name and Location for the Project

Project Name: controller

Project Location: C:\Controller\controller

Select the Type of Top-Level Source for the Project

Top-Level Source Type: HDL

Fig 3.3: Creating a New Project

2. It opens as the above figure. Give the project name and project location. It should be noted that the project location should be in hard disk (C) drive and spaces are not allowed in the project name.

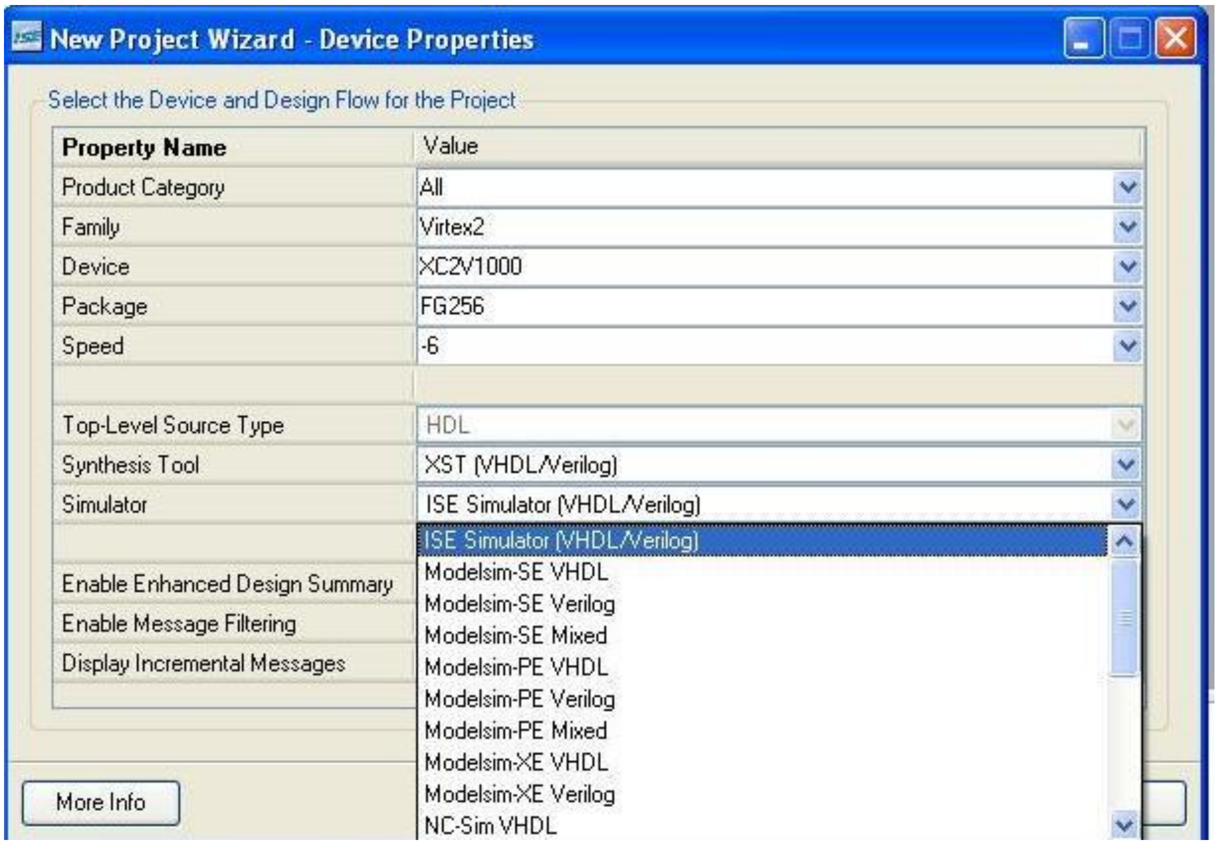
3. Set the **Top-Level Module Type** to **HDL** and click **Next**.

3.8.2 Selecting the Device properties and Design flow:

1. Select the **Device properties** like Family of the FPGA device used in the project, device name, package of the device and its speed.

2. The **Synthesis tool** used in this project is **XST (VHDL/Verilog)**. The simulator can be chosen as per available tools like the Modelsim or an ISE Simulator. Here in this project, we use **ISE Simulator (VHDL/Verilog)**

2. Select **Next** to complete this step. Later, the device and design flow of the project is chosen which can be viewed from Fig 3.4.



New Project Wizard - Device Properties

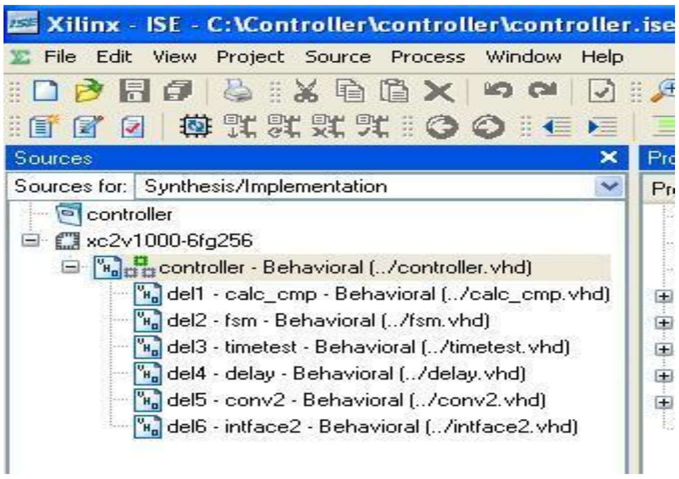
Select the Device and Design Flow for the Project

Property Name	Value
Product Category	All
Family	Virtex2
Device	XC2V1000
Package	FG256
Speed	-6
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISE Simulator (VHDL/Verilog)
Enable Enhanced Design Summary	
Enable Message Filtering	
Display Incremental Messages	

Fig 3.4: Choosing Device Properties

3.8.3 Adding source files or new sources:

1. Now select **Project → Add source** instead of Add copy of source. Navigate to the folder where you stored VHDL files (controller & sub-files). These files are stored in a hierarchy which looks like as in Fig 3.5.



Xilinx - ISE - C:\Controller\controller\controller.ise

File Edit View Project Source Process Window Help
Sources
Sources for: Synthesis/Implementation

```
controller
xc2v1000-6fg256
  controller - Behavioral (../controller.vhd)
    del1 - calc_cmp - Behavioral (../calc_cmp.vhd)
    del2 - fsm - Behavioral (../fsm.vhd)
    del3 - timetest - Behavioral (../timetest.vhd)
    del4 - delay - Behavioral (../delay.vhd)
    del5 - conv2 - Behavioral (../conv2.vhd)
    del6 - intface2 - Behavioral (../intface2.vhd)
```

Fig 3.5: Hierarchy of sources

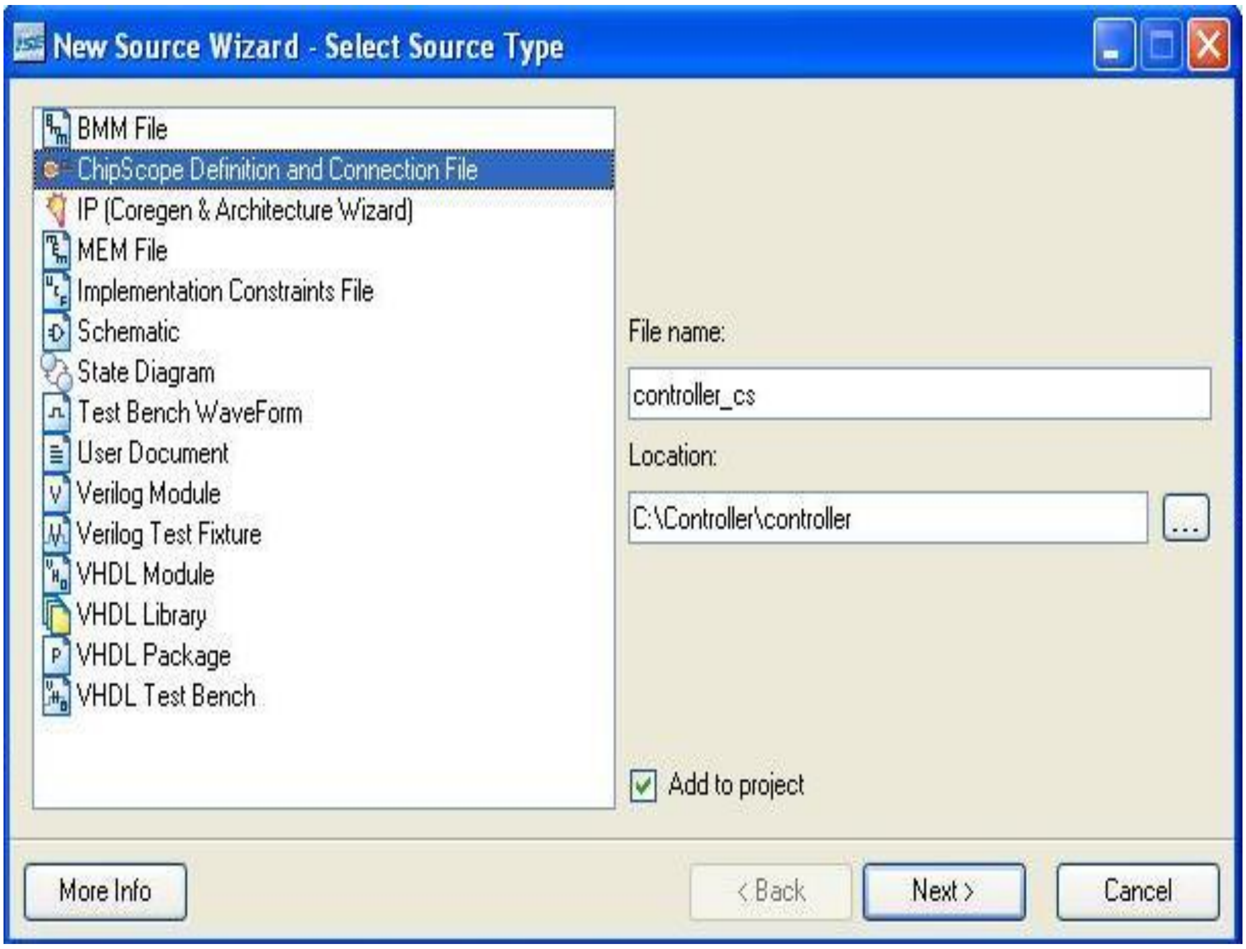
2. Project Navigator will keep the hierarchy of the files showing which modules are instantiated to the top module.

3. If something goes wrong with synthesizing the design, then there shows some message in **console** and even shows '?' next to the modules which are missing.

4. Hierarchy looks something like the above figure in Sources window under top module.

3.8.4 Adding Chipscope Definition and Connection File:

1. Now right click on the top module (controller) of the design which ought to be verified and select **New Source**. Then, select **Chipscope Definition and Connection File** and give some appropriate name to it as shown in Fig 3.6.



File name: controller_cs

Location: C:\Controller\controller

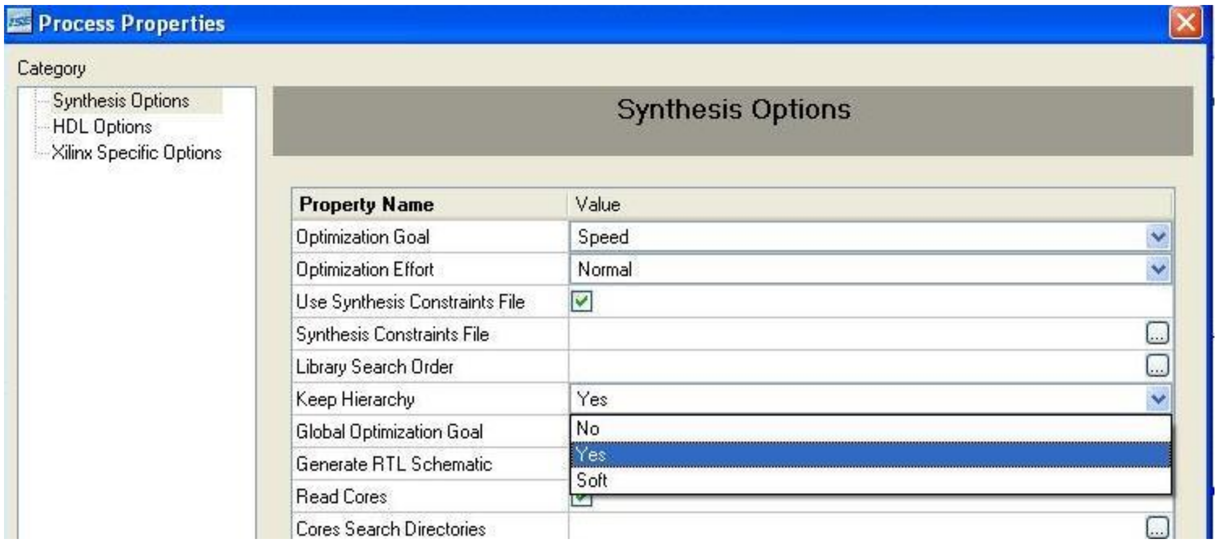
✓ Add to project

Fig 3.6: Adding Chipscope Definition and Connection File

2. Click on **Next** to finish adding this file.

3. Adding this file implies that the Project uses tool called Chipscope. So, it is not necessary to add this file to every project.

3.8.5 Selecting Process Properties:



Process Properties

Category
- Synthesis Options
- HDL Options
- Xilinx Specific Options

Synthesis Options

Property Name	Value
Optimization Goal	Speed
Optimization Effort	Normal
Use Synthesis Constraints File	✓
Synthesis Constraints File	
Library Search Order	
Keep Hierarchy	Yes
Global Optimization Goal	No
Generate RTL Schematic	Yes
Read Cores	Soft
Cores Search Directories	

Fig 3.7: Selecting synthesis options

1. In order to identify easily and preferred signal names under each ILA, right click on **Synthesize-XST** properties. Select **Keep Hierarchy→ 'Yes'** or **'Soft'** as shown in Fig 3.7.

2. Click on **Apply** and **OK** to finish this step.

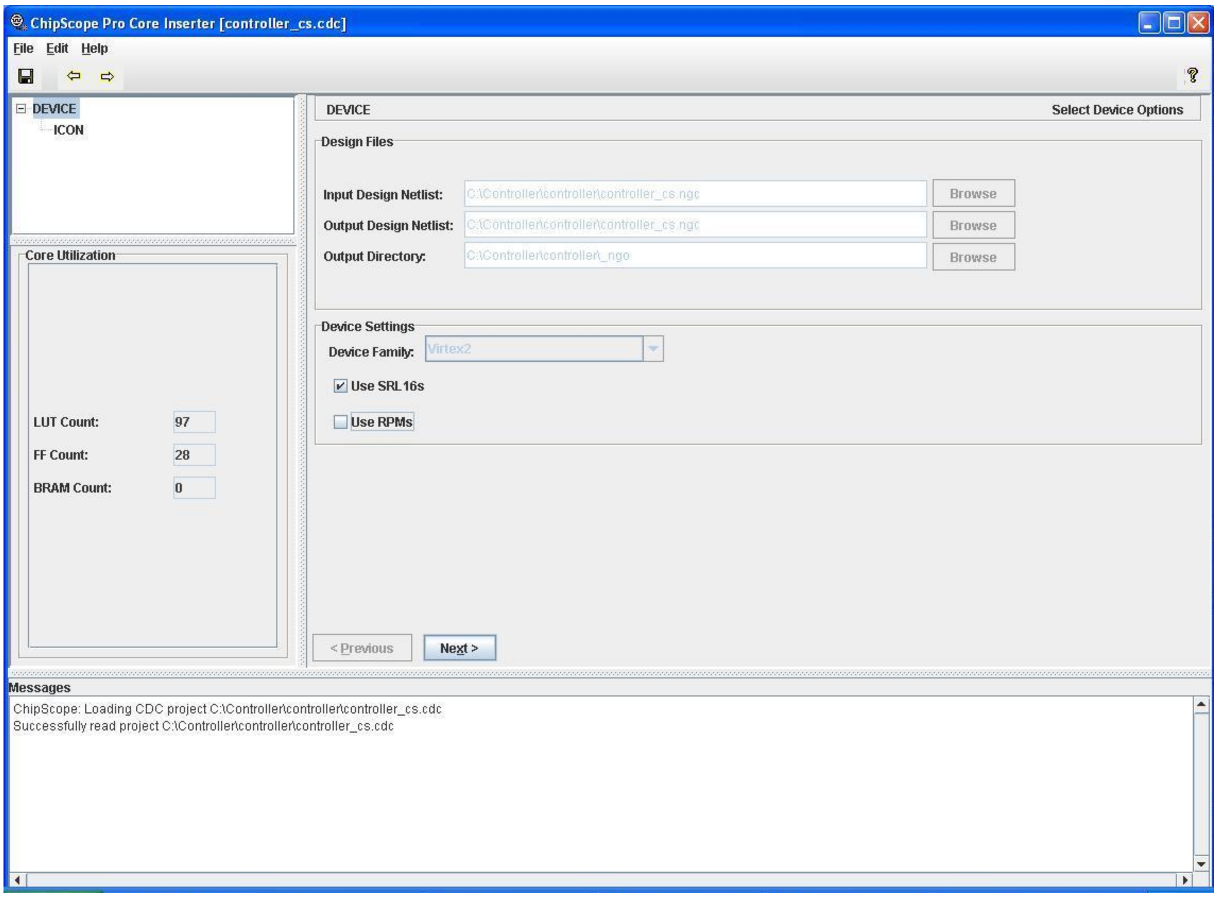
3. Double click on the **Chipscope Definition and Connection source File** which intends to open **Chipscope Pro Core Inserter**.

4. Since this core inserter project was launched from the Project Navigator, it is not necessary to selecting **File→New Project**. The project opens and the input and output design netlist are automatically filled in.

3.8.6 Selecting Device Settings:

1. These settings include **Device Family, SRL16 usage** and **RPM usage** which can be viewed from Fig 3.8. The target FPGA device family is displayed in the **Device family** field if only the Chipscope Core Inserter was launched from Project Navigator. The optimization of cores depends upon different families.

2. Using SRL16s: This checkbox is selected to know whether cores are generated or not using. If this checkbox is not selected, then the SRL16s components are replaced with multiplexers and flip-flops which affect the generated ILA cores .This option of the device settings is chosen by default for the Virtex-II, Virtex-4,Virtex-5,Spartan-3,Spartan-3E and Spartan-3A device families.



Design Files

Input Design Netlist: C:\Controller\controller\controller_cs.ngc

Output Design Netlist: C:\Controller\controller\controller_cs.ngc

Output Directory: C:\Controller\controller_ngo

Device Settings

Device Family: Virtex2

[✓] Use SRL16s

[ ] Use RPMs

Core Utilization

LUT Count: 97

FF Count: 28

BRAM Count: 0

Messages

```
ChipScope: Loading CDC project C:\Controller\controller\controller_cs.cdc
Successfully read project C:\Controller\controller\controller_cs.cdc
```

Fig 3.8: Selecting Device Properties

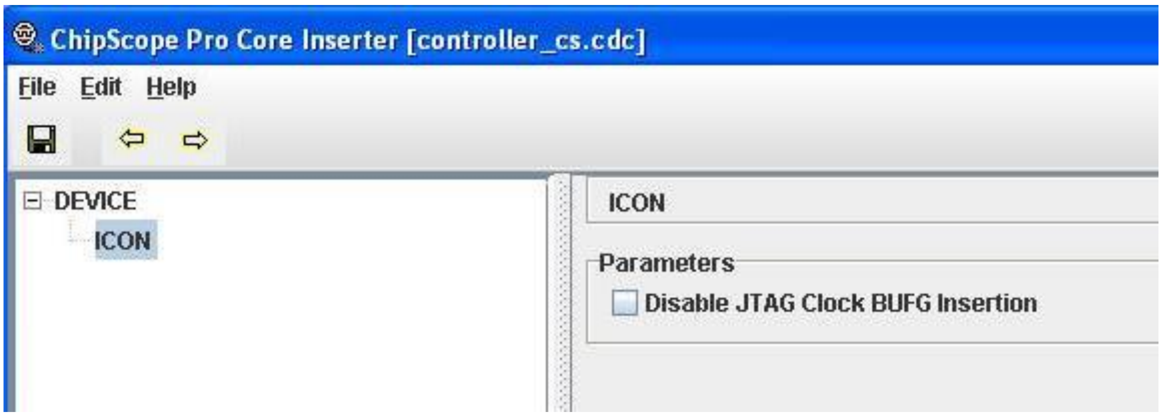
3. Using RPMs: Using RPM (Relationally Placed Macros) makes the ILA cores to be optimized for placement. If we use most of the resources from the device, this use of RPM's make some mesh.

4. In this project, we don't check this box because the design is too large and needs more resources. Therefore it is very difficult to meet the placement constraints. When the device settings are done, click **Next**.

5. Core Utilization Panel: There is a core utilization Panel to the left most side of the Chipscope core inserter. This gives an estimated count of look-up table (LUT), block RAM (BRAM) and flip-flop (FF) that are being used by the Chipscope cores called ICON and ILA cores. This count is stored in design netlist.

3.8.7 Selecting ICON options:

1. ICON core is the core which acts like a controller and connects all ILA cores to the JTAG boundary scan chain. Its parameters are defined by clicking of the **ICON** option as shown in Fig 3.9.



☐ Disable JTAG Clock BUFG Insertion

Fig 3.9: Selecting ICON Parameters

2. **Disabling JTAG Clock BUFG Insertion:** This makes the implementation tools to select the JTAG Clock using normal routing resources rather than using global clock routing resources. Clock in every project by default uses global clock resource (BUFG). If the device is having some external resources, we can choose this option. After selecting the appropriate option for ICON parameters click **Next** or **New ILA**.

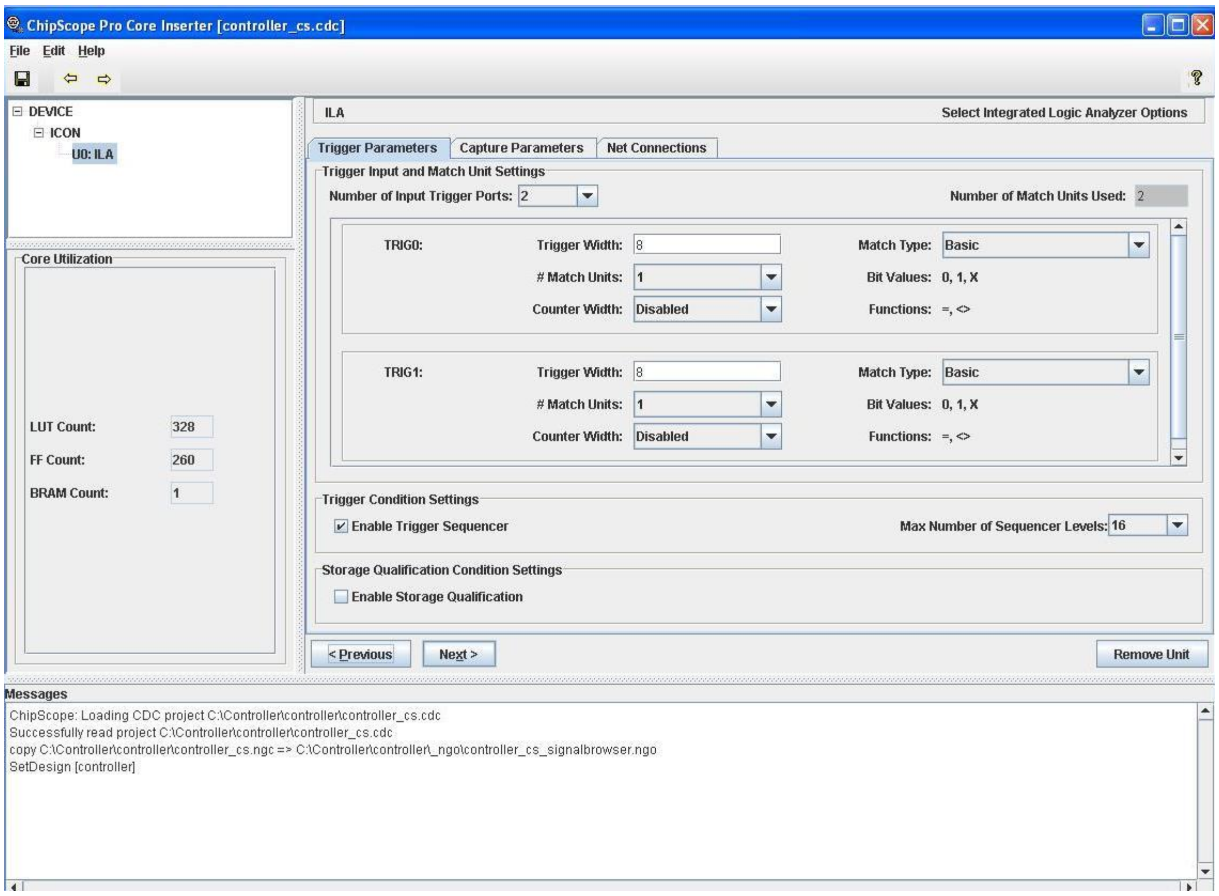
3.8.8 Selecting ILA core trigger and parameter options:

The Fig 3.10 shows that a new ILA has been created in the device hierarchy under ICON.

1. **Selecting Number of trigger ports**: Choose the number of ports which you want to view in Chipscope Logic Analyzer. ILA in individual supports nearly 16 separate trigger ports. Once after choosing number of trigger ports, an option emerges for each of these ports. Each trigger port is tagged with **TRIGn**, where **'n'** is the port number from 0 to 15. Now select each trigger option which includes **trigger width**, and **number of match units** connected to the trigger port and type of these match units.

2. **Trigger Width:** Each port is individual signal with a number of bits. Range value for this field supports from 1 to 256.

3. Selecting the Match unit number: In order to detect the events, a comparator called match unit is connected to the trigger port. The Overall trigger condition is formed by combining the logic of one or more match units. And this overall trigger condition is used to control the data capture. Increasing the number of match units per trigger port also increases the logic resources usage.



Core Utilization

LUT Count: 328
FF Count: 260
BRAM Count: 1

Trigger Input and Match Unit Settings

Number of Input Trigger Ports: 2
Number of Match Units Used: 2

TRIG0: Trigger Width: 8 | Match Type: Basic
# Match Units: 1 | Bit Values: 0, 1, X
Counter Width: Disabled | Functions: =, <>

TRIG1: Trigger Width: 8 | Match Type: Basic
# Match Units: 1 | Bit Values: 0, 1, X
Counter Width: Disabled | Functions: =, <>

Trigger Condition Settings

[✓] Enable Trigger Sequencer
Max Number of Sequencer Levels: 16

Storage Qualification Condition Settings

[ ] Enable Storage Qualification

Messages
```
ChipScope: Loading CDC project C:\Controller\controller\controller_cs.cdc
Successfully read project C:\Controller\controller\controller_cs.cdc
copy C:\Controller\controller\controller_cs.ngc => C:\Controller\controller\_ngo\controller_cs_signalbrowser.ngo
SetDesign [controller]
```

Fig 3.10 Selecting Trigger Parameters for ILA

4. **Choosing Match unit Type**: There are 6 types of match units.

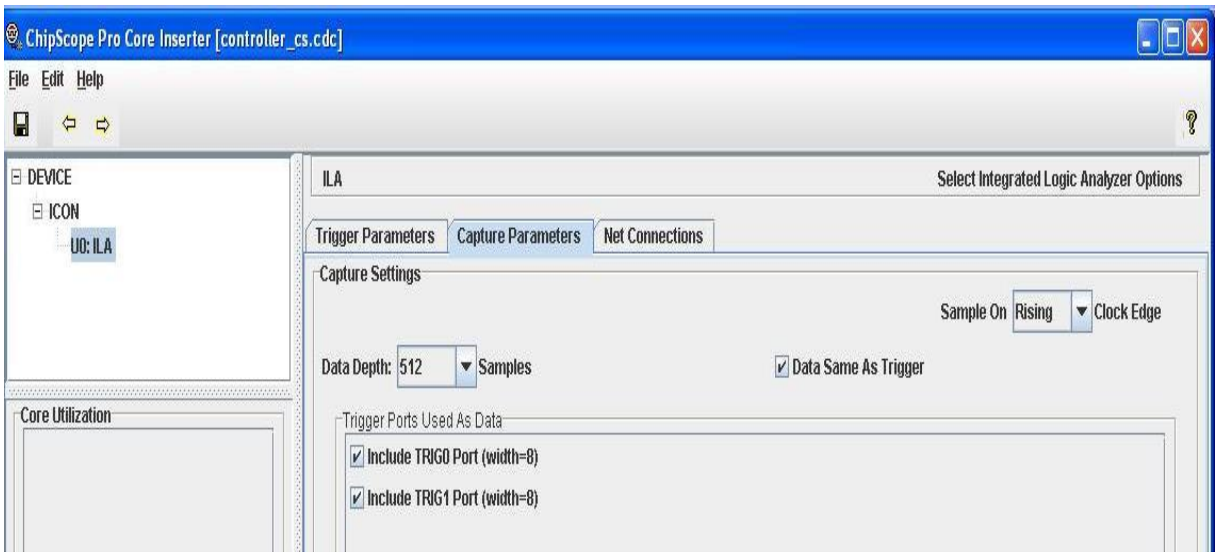
- Basic type: This type is used to compare data signals where transition detection is not important.
- Basic w/edges: This type is used to compare control signals where transition detection is important.
- Extended: This type is used to compare data signals where magnitude is important.
- Extended w/edges: This type is used to compare data signals where magnitude and transition detection is important.
- Range: This type is used to compare data signals where a specific range of values are taken.
- Range w/edges: This type is used to compare data signals where transition detection and specific range of values are considered.

5. **Choosing Match unit counter width**: There is counter present on output of each match unit which is configurable during run time to count a specific number of match unit events. **Counter width** setting is **Disabled** by default.

6. Trigger condition settings: Enabling trigger sequencer enables to configure from one level to next at run time using either contiguous or non-contiguous sequence of match function events**.**

7. Storage Qualification Condition Settings: This condition is used to define when to start and end the capture of process. The ILA core executes the **storage qualification condition**. This condition is a Boolean combination of match function events. In this project we are not checking the storage qualification condition. After completing this step click **Next**.

3.8.9 Selecting ILA core capture parameters:


ChipScope Pro Core Inserter [controller_cs.cdc]

ILA — Select Integrated Logic Analyzer Options

Capture Settings

Sample On: Rising Clock Edge

Data Depth: 512 Samples

✓ Data Same As Trigger

Trigger Ports Used As Data

✓ Include TRIG0 Port (width=8)

✓ Include TRIG1 Port (width=8)

Fig 3.11: Selecting Capture Parameters

1. **Capture settings**: Select the ILA core's maximum capacity to store the sample buffer which is called as **Data Depth** which can be seen in Fig 3.11. It is the count of data width bits contributed by each BRAM unit. Block RAM resources are counted by resource utilization estimator feature in Chipscope Core Inserter.

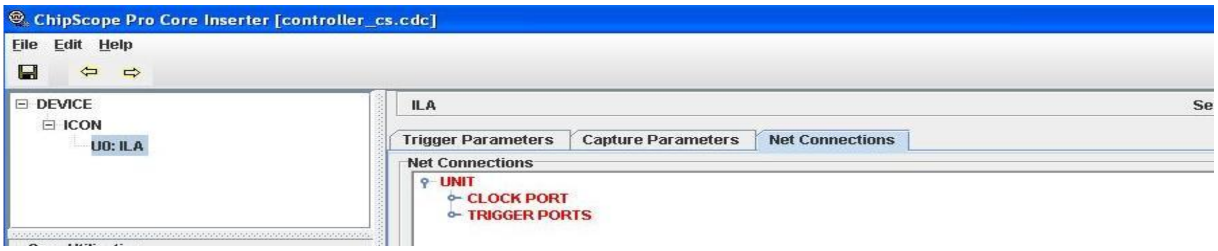
2. **Selecting the Data type**: Data type can be of two sources.

- **Data separate from trigger**: This condition is chosen whenever we want to limit the amount of data to be captured and the data port is different from the trigger port.
- **Data same as trigger**: Both the trigger and data ports are same and identical. When this option is chosen the ILA core doesn't take any input separately for data.

After choosing the above option click **Next**.

i. Selecting Net Connections:

1. This selection of Net Connections includes CLOCK PORTS and TRIGGER PORTS if the data is same as trigger as shown in Fig 3.12. If the data is not same as trigger, then this tab also includes DATA PORTS.



ChipScope Pro Core Inserter [controller_cs.cdc]

File Edit Help

DEVICE
ICON
U0: ILA

ILA

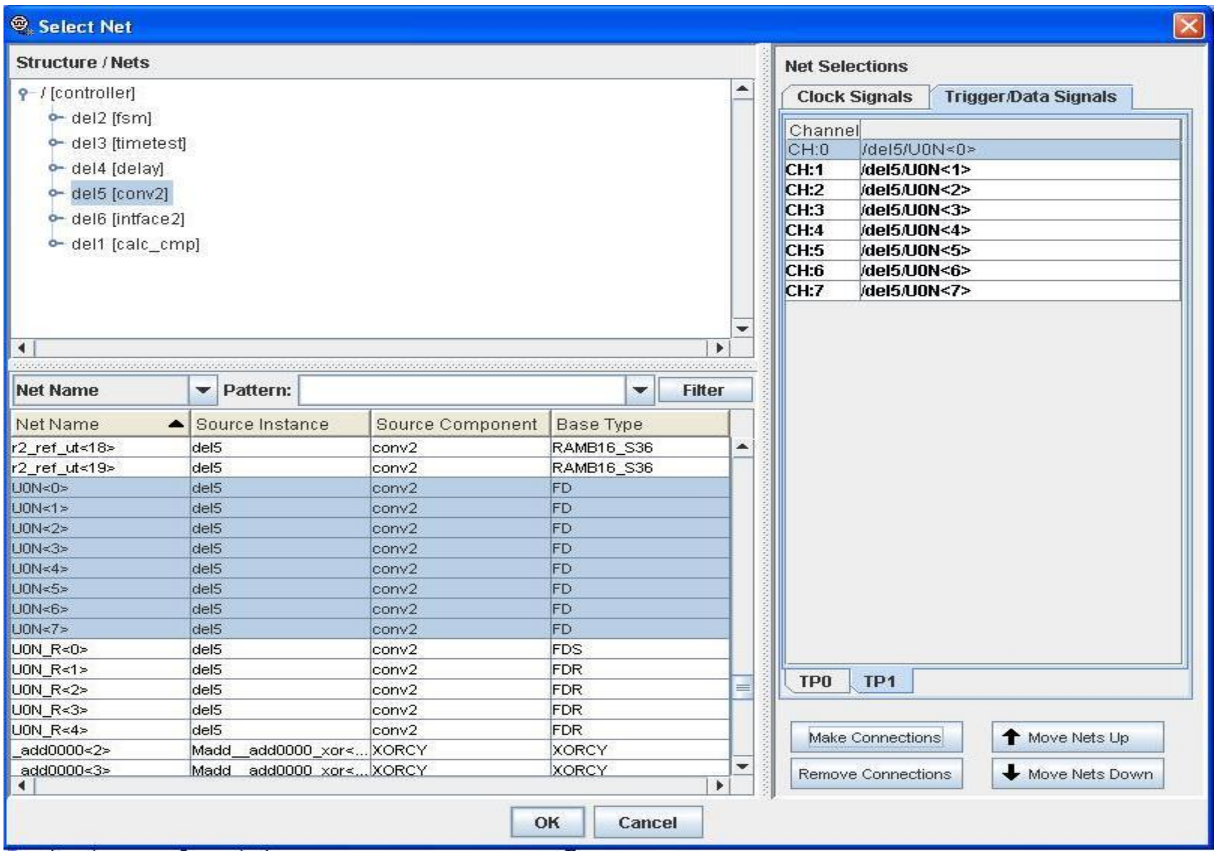
Se

Trigger Parameters | Capture Parameters | Net Connections

Net Connections
UNIT
CLOCK PORT
TRIGGER PORTS

Fig 3.12: Net Connections

2. In this project, we choose data same as trigger. These are the connections made to signals for each ILA with specific port names. Double Click on **CLOCK PORT** to make the connection with ILA clock ports which are available and working. The appearance of these ports in red color indicates that there is no connection made to the signals.



Select Net

Structure / Nets

/ [controller]
del2 [fsm]
del3 [timetest]
del4 [delay]
del5 [conv2]
del6 [intface2]
del1 [calc_cmp]

Net Name | Pattern: | Filter

Net Name	Source Instance	Source Component	Base Type
r2_ref_ut<18>	del5	conv2	RAMB16_S36
r2_ref_ut<19>	del5	conv2	RAMB16_S36
U0N<0>	del5	conv2	FD
U0N<1>	del5	conv2	FD
U0N<2>	del5	conv2	FD
U0N<3>	del5	conv2	FD
U0N<4>	del5	conv2	FD
U0N<5>	del5	conv2	FD
U0N<6>	del5	conv2	FD
U0N<7>	del5	conv2	FD
U0N_R<0>	del5	conv2	FDS
U0N_R<1>	del5	conv2	FDR
U0N_R<2>	del5	conv2	FDR
U0N_R<3>	del5	conv2	FDR
U0N_R<4>	del5	conv2	FDR
_add0000<2>	Madd__add0000_xor<...	XORCY	XORCY
_add0000<3>	Madd__add0000_xor<...	XORCY	XORCY

Net Selections

Clock Signals | Trigger/Data Signals

Channel	
CH:0	/del5/U0N<0>
CH:1	/del5/U0N<1>
CH:2	/del5/U0N<2>
CH:3	/del5/U0N<3>
CH:4	/del5/U0N<4>
CH:5	/del5/U0N<5>
CH:6	/del5/U0N<6>
CH:7	/del5/U0N<7>

TP0 | TP1

Make Connections | Move Nets Up
Remove Connections | Move Nets Down

OK | Cancel

Fig 3.13: Selecting signals for Net Connections

3. We can select each appropriate ILA under **structure nets** and select the trigger ports for the signals under that particular net. Just drag the signals under net name and click on **Make Connections** as shown in Fig 3.13.

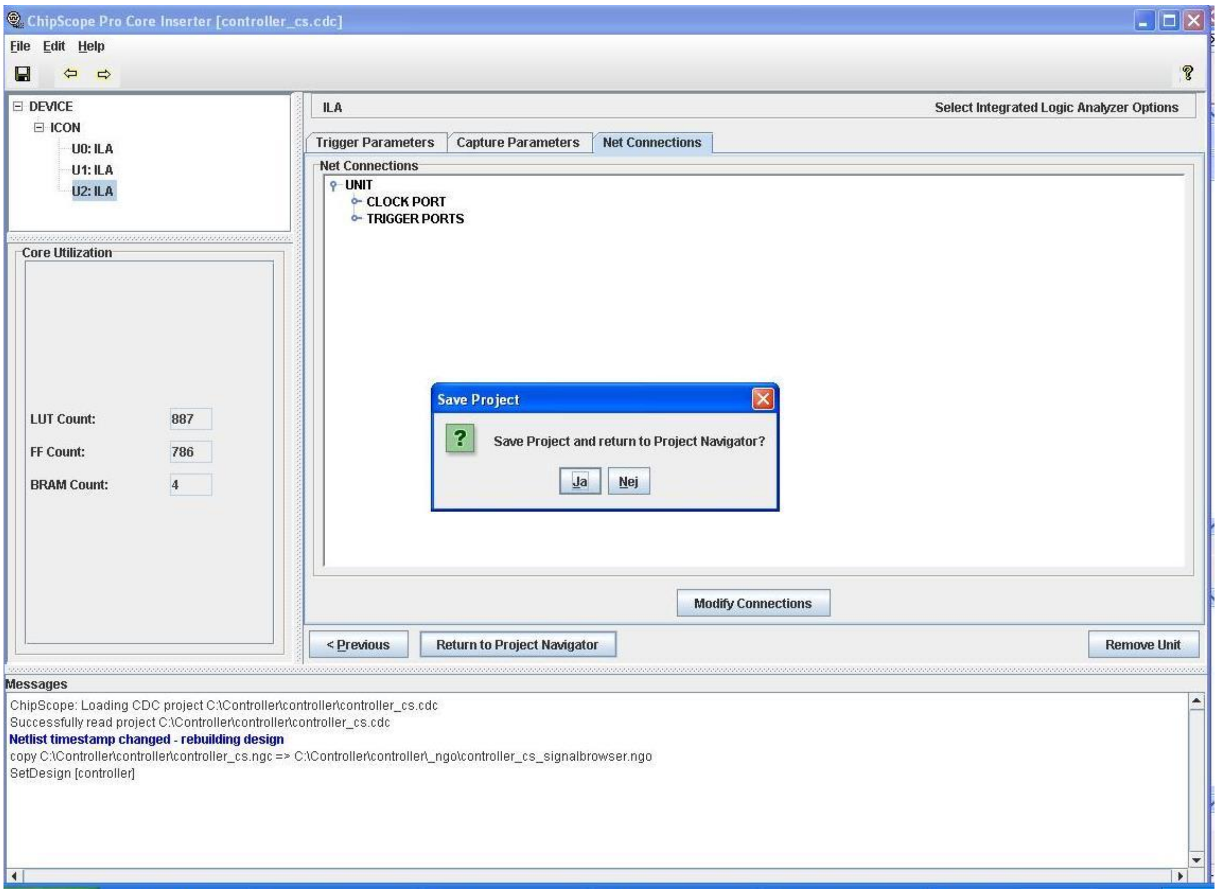
5. **Net Name**: This is the name of the Net as it shows in EDIF file.

6. **Source Instance**: This is the name of the source through which the net is driven at the current level of hierarchy.

7. **Source Component**: It describes the component type.

8. **Base Type**: This can be a black box component or a primitive component.

9. After finishing this click **OK.**



LUT Count: 887
FF Count: 786
BRAM Count: 4

Save Project and return to Project Navigator?

```
ChipScope: Loading CDC project C:\Controller\controller\controller_cs.cdc
Successfully read project C:\Controller\controller\controller_cs.cdc
Netlist timestamp changed - rebuilding design
copy C:\Controller\controller\controller_cs.ngc => C:\Controller\controller\_ngo\controller_cs_signalbrowser.ngo
SetDesign [controller]
```

Fig 3.14: Returning to Project Navigator

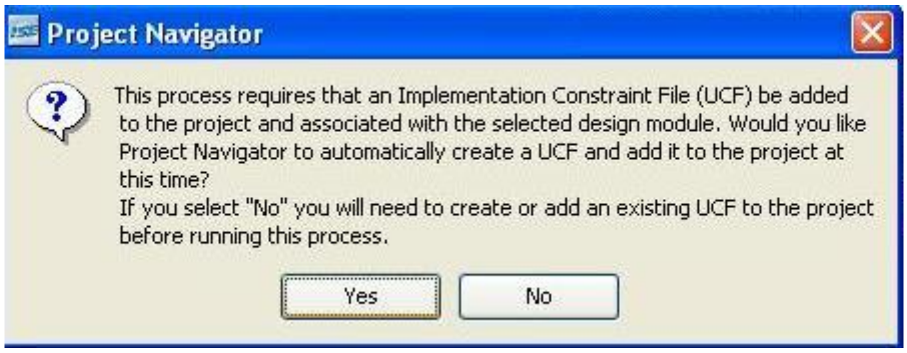
10. So Once after choosing the clock, trigger and data nets, a window opens as in the above figure asking to **Return to Project Navigator.** When you click on **Return to Project Navigator**, it terminates to a dialog box asking to **Save Projector and return to Project Navigator?**

11. Click on **Ja** to save the project as shown in Fig 3.14.

3.8.11 Creating Timing Constraints:

1. **Timing Constraints**: These constraints are used to meet all the design requirements according to the implementation tools. They also ensure that all constraints were met.

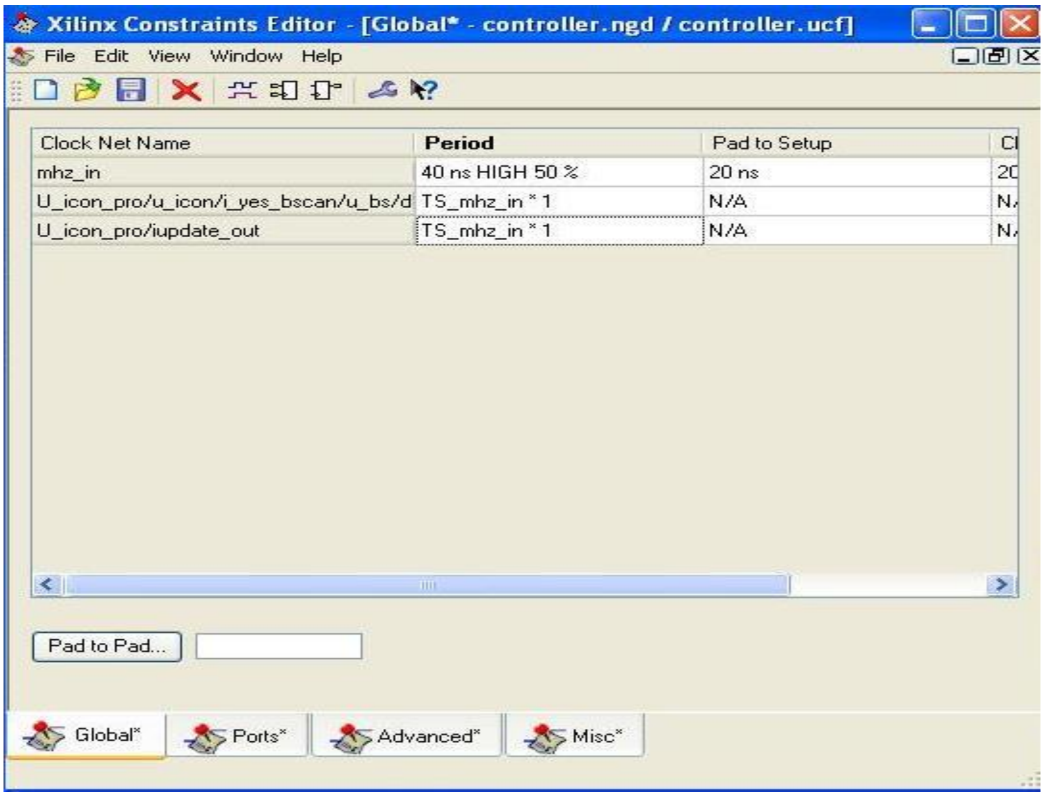
2. Now expand User Constraints in the processes tab. Double click on the **Create Timing Constraints**, then a dialog box opens asking whether to add the **User Constraints file** to the project.


Project Navigator

This process requires that an Implementation Constraint File (UCF) be added to the project and associated with the selected design module. Would you like Project Navigator to automatically create a UCF and add it to the project at this time?
If you select "No" you will need to create or add an existing UCF to the project before running this process.

Fig 3.15: Adding UCF to the Project

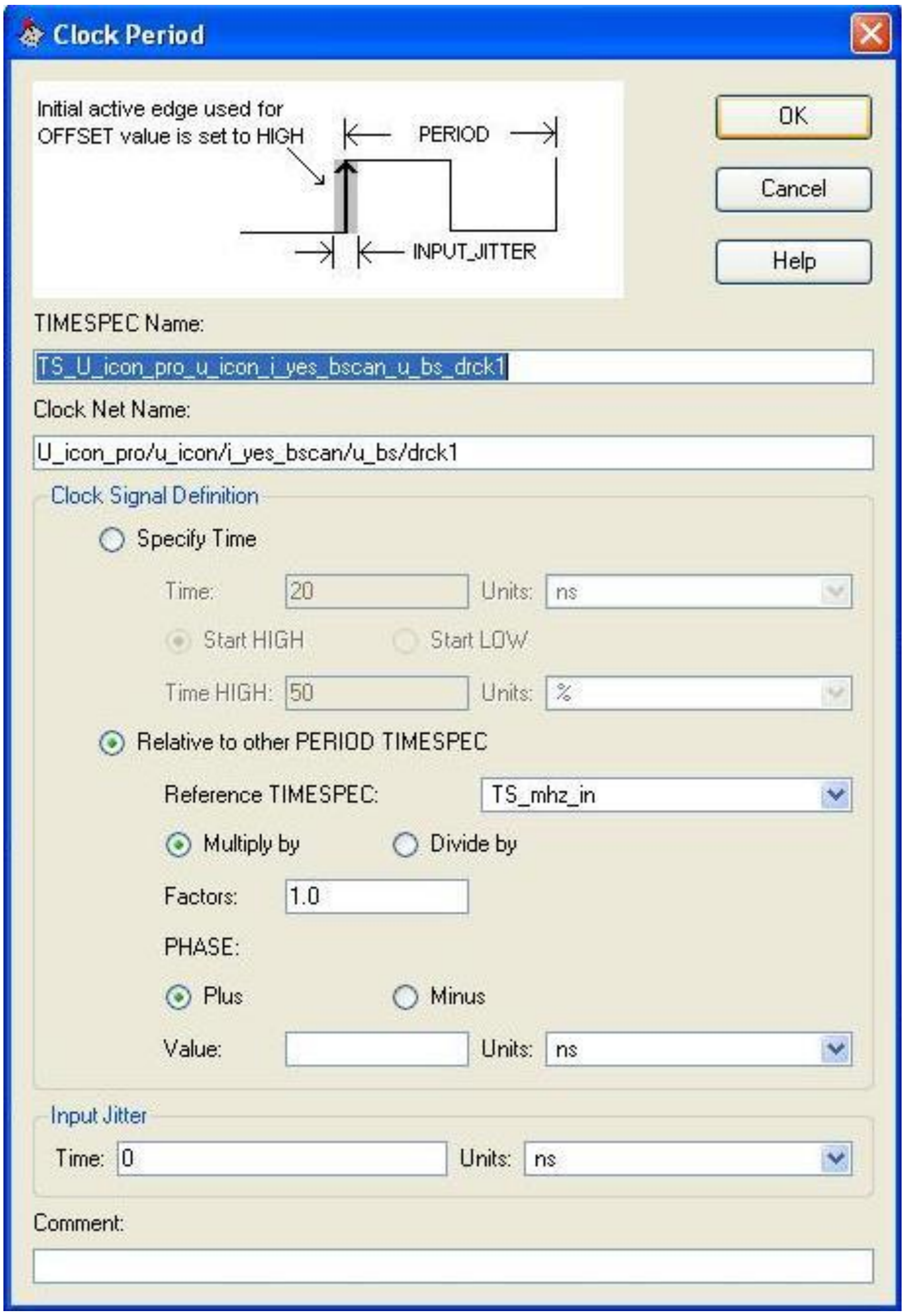
3. So click **Yes** to add the user constraints file (.ucf) to the project as shown in Fig 3.15.


Xilinx Constraints Editor - [Global* - controller.ngd / controller.ucf]

Clock Net Name	Period	Pad to Setup	Cl
mhz_in	40 ns HIGH 50 %	20 ns	2C
U_icon_pro/u_icon/i_yes_bscan/u_bs/d	TS_mhz_in * 1	N/A	N/
U_icon_pro/iupdate_out	TS_mhz_in * 1	N/A	N/

Fig 3.16: Specifying Constraints

4. Select each and every clock present under **Clock Net Name** in the Constraints Editor **Global tab** which can be viewed from Fig 3.16. Give the specified **Period** for each clock periods. Double click the empty **Period** field to specify with some values as in Fig 3.17.



TIMESPEC Name: TS_U_icon_pro_u_icon_i_yes_bscan_u_bs_drck1

Clock Net Name: U_icon_pro/u_icon/i_yes_bscan/u_bs/drck1

Clock Signal Definition

Specify Time

Time: 20 Units: ns

Start HIGH Start LOW

Time HIGH: 50 Units: %

[✓ Relative to other PERIOD TIMESPEC]

Reference TIMESPEC: TS_mhz_in

[✓ Multiply by] Divide by

Factors: 1.0

PHASE:

[✓ Plus] Minus

Value: Units: ns

Input Jitter

Time: 0 Units: ns

Comment:

Fig 3.17: Specifying Clock Period

5. Click **OK to** finish the step**.**

6. Now select the **Pad to Setup** field and **OFFSET** field to set output delay constraint.

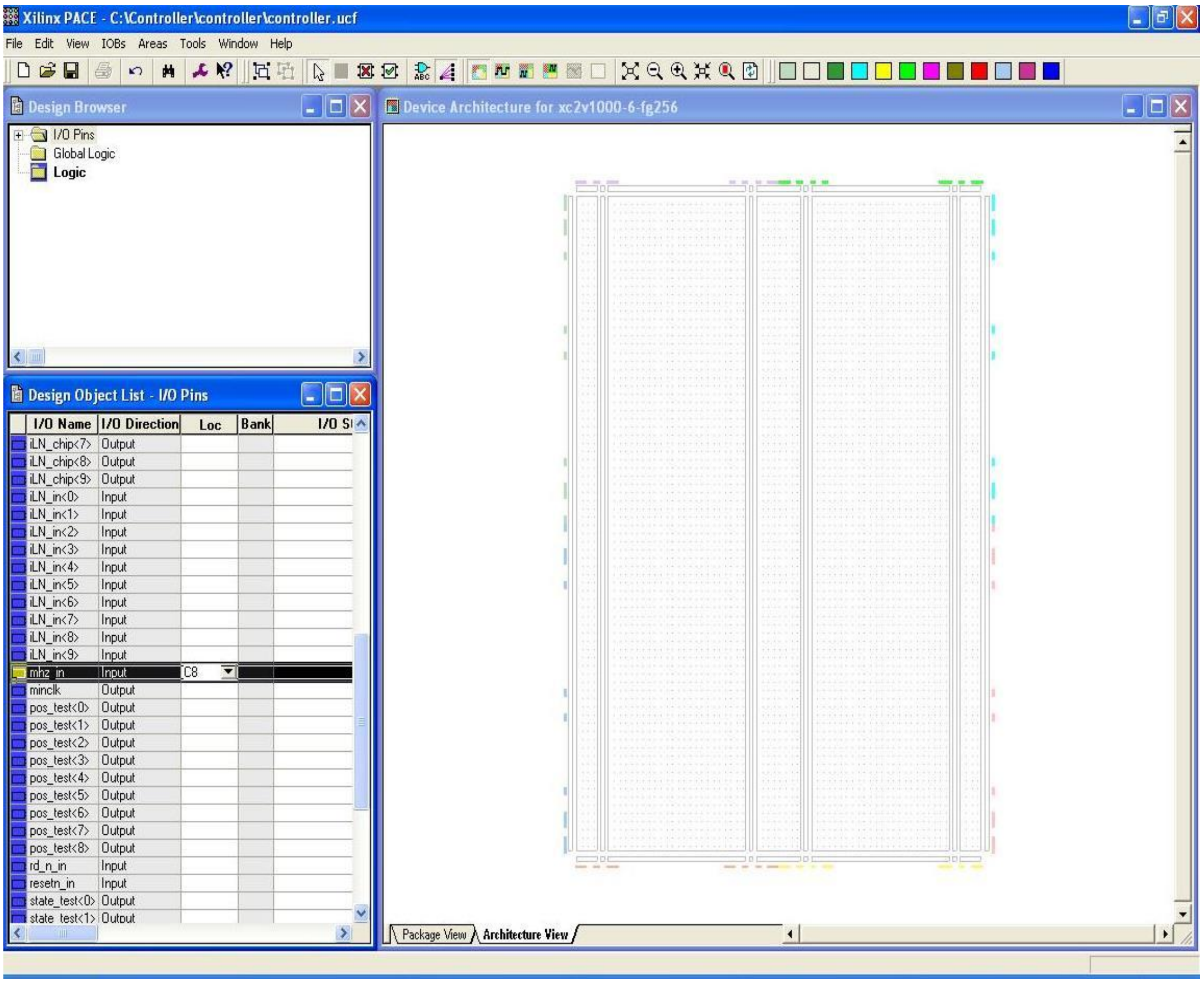
7. Save the timing constraints and click **OK** to continue.

8. Close the **Constraints** Window.

3.8.12 Assigning Package pins:

1. Select the top module from the Sources window.

2. Double click on **Assign Package Pins** process. It emerges to a **Pinout and Area Constraints Editor (PACE)** window as shown in Fig 3.18. Select the **Design Object List- I/O Pins** and assign the **Clock signal** with the running pin .This can be done with Virtex-II user guide which gives the location of Clock enable switches.


Xilinx PACE - C:\Controller\controller\controller.ucf
File Edit View IOBs Areas Tools Window Help
Design Browser
I/O Pins
Global Logic
Logic
Device Architecture for xc2v1000-6-fg256
Design Object List - I/O Pins
Package View
Architecture View

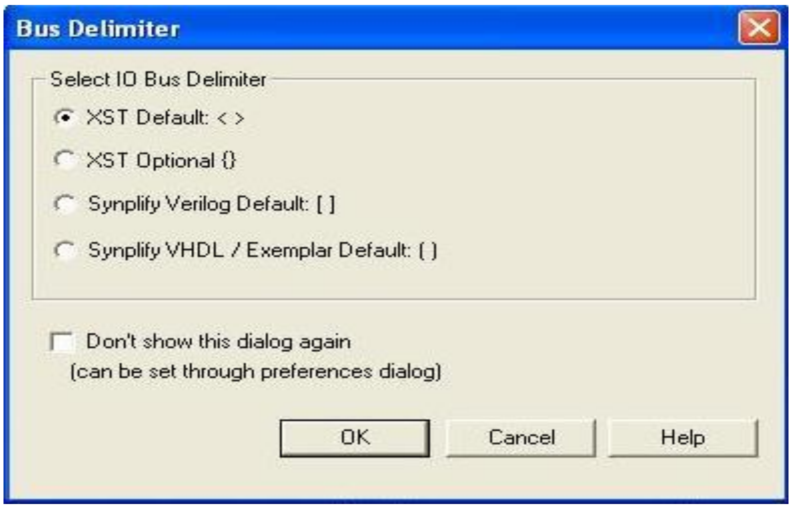
I/O Name	I/O Direction	Loc	Bank	I/O St
iLN_chip<7>	Output			
iLN_chip<8>	Output			
iLN_chip<9>	Output			
iLN_in<0>	Input			
iLN_in<1>	Input			
iLN_in<2>	Input			
iLN_in<3>	Input			
iLN_in<4>	Input			
iLN_in<5>	Input			
iLN_in<6>	Input			
iLN_in<7>	Input			
iLN_in<8>	Input			
iLN_in<9>	Input			
mhz_in	Input	C8		
minclk	Output			
pos_test<0>	Output			
pos_test<1>	Output			
pos_test<2>	Output			
pos_test<3>	Output			
pos_test<4>	Output			
pos_test<5>	Output			
pos_test<6>	Output			
pos_test<7>	Output			
pos_test<8>	Output			
rd_n_in	Input			
resetn_in	Input			
state_test<0>	Output			
state_test<1>	Output			

Fig 3.18: Assigning Package pins

4. Here we gave '**C8**' as Clock pin as per the Virtex-II manual.

5. Save the file by selecting **File→Save**. This drives to select a bus delimiter type. Now select XST Default <> and click **OK**.

6. Close **PACE.** The package pins location is stored by following a dialogue box which looks as in Fig 3.19.



Bus Delimiter

Select IO Bus Delimiter

(● XST Default: < >

( ) XST Optional {}

( ) Synplify Verilog Default: [ ]

( ) Synplify VHDL / Exemplar Default: ( )

☐ Don't show this dialog again (can be set through preferences dialog)

Fig 3.19: Saving the package pin location

Now this shows an orange mark on the implementation tools, which indicates that the UCF file has been modified**.**

3.8.13 RTL Schematic:

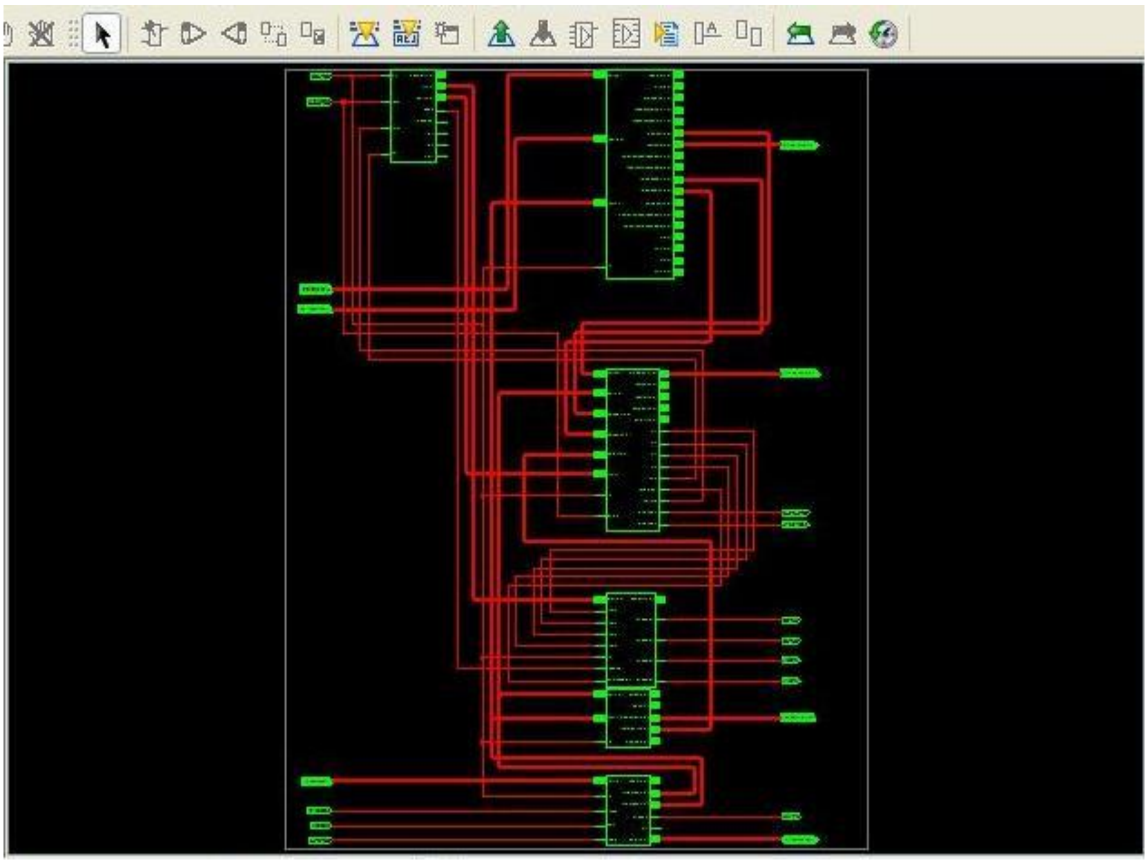


Fig 3.20: RTL Schematic

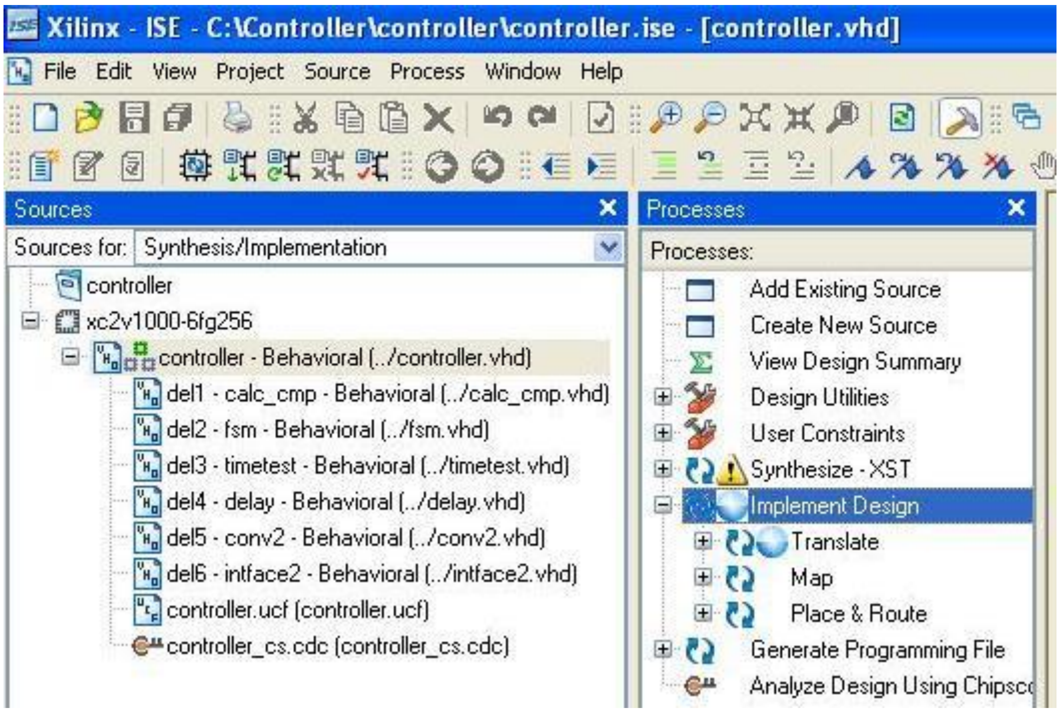
1. In order to view Schematic of the design, select the top module of the project. Double click on the **View RTL Schematic** under **Synthesize-XST** of **Processes** tab. It emerges into a new window like in the Fig 3.20.

2. This is a controller instance block diagram with 6 ILAs.

3.8.14 Implementation:

1. **Implementing Design**: It is the process of translating, mapping, placing and routing and generating a BIT file.

2. Now select the top module in sources window and double click on the **Implement Design** in Processes tab which can be seen in the Fig 3.21.


Xilinx - ISE - C:\Controller\controller\controller.ise - [controller.vhd]

File Edit View Project Source Process Window Help

Sources

Sources for: Synthesis/Implementation

- controller
- xc2v1000-6fg256
  - controller - Behavioral (../controller.vhd)
    - del1 - calc_cmp - Behavioral (../calc_cmp.vhd)
    - del2 - fsm - Behavioral (../fsm.vhd)
    - del3 - timetest - Behavioral (../timetest.vhd)
    - del4 - delay - Behavioral (../delay.vhd)
    - del5 - conv2 - Behavioral (../conv2.vhd)
    - del6 - intface2 - Behavioral (../intface2.vhd)
    - controller.ucf (controller.ucf)
    - controller_cs.cdc (controller_cs.cdc)

Processes

Processes:

- Add Existing Source
- Create New Source
- View Design Summary
- Design Utilities
- User Constraints
- Synthesize - XST
- Implement Design
  - Translate
  - Map
  - Place & Route
- Generate Programming File
- Analyze Design Using Chipsc

Fig 3.21: Implementing Design

3. **Translate process**: It converts all input design netlists into a single merged NGD netlist. This netlist consists of the logic in design. This performs logical design rule checks, timing specification and automatically adds the user constraints file (UCF) to the merged netlist.

4. **Mapping the Design**: In this process, the design is plotted into Configurable Logic Blocks (CLBs) and Input/output Blocks (IOBs) for all logic elements. Mapping process runs according to design rule check by final mapped netlist and performs optimal device optimizations.

3.8.15 Evaluating Logical Blocks in Design using Timing Analysis:

1. Expand Place and Route process in processes tab. Double click on **Generate Post-Map & Route Static Timing**. In order to view the report, click '+' sign to Generate Post-Map Static Timing and double click on **Analyze Post-Map Static Timing Report** as shown in the Fig 3.22.

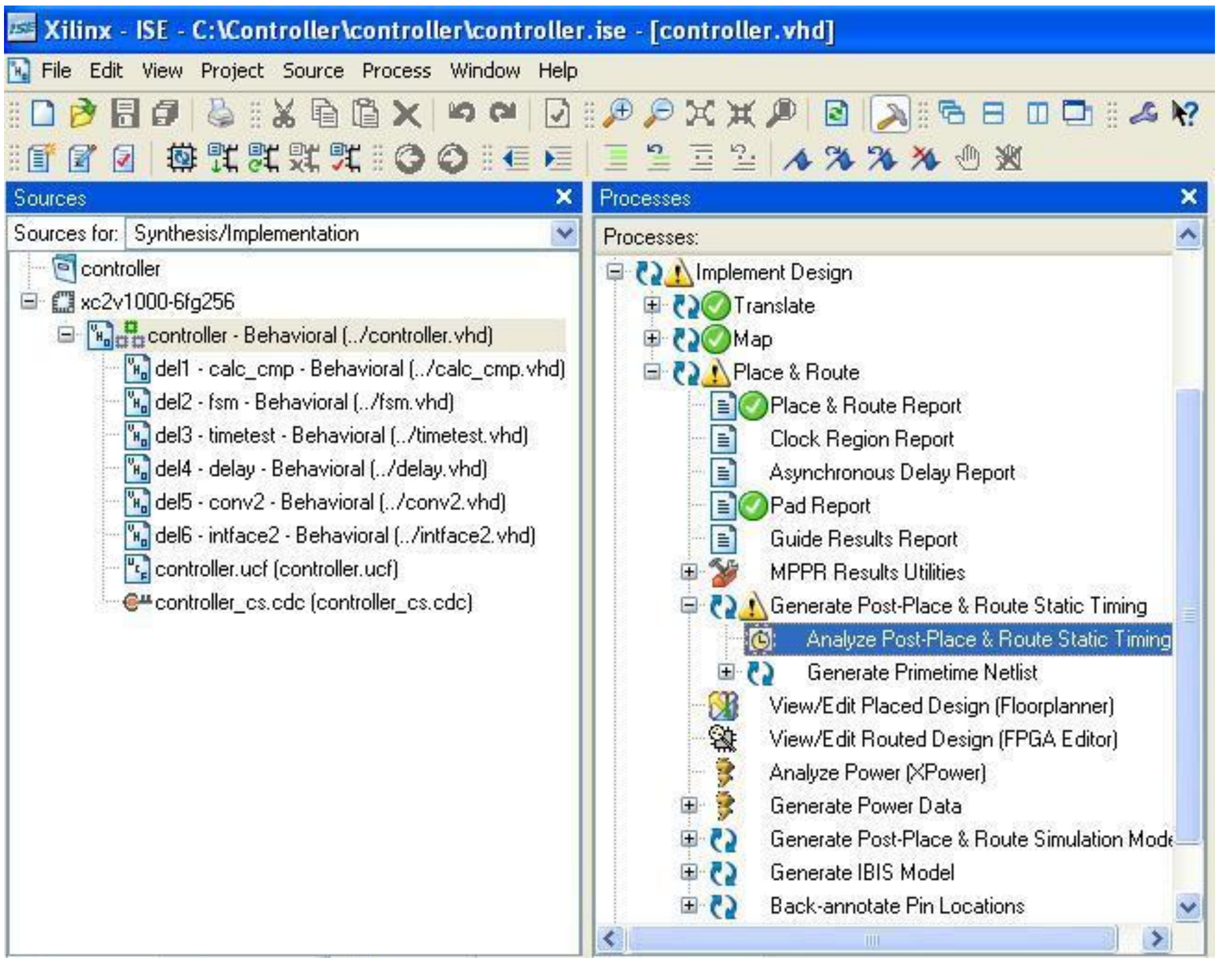

Xilinx - ISE - C:\Controller\controller\controller.ise - [controller.vhd]
File Edit View Project Source Process Window Help
Sources
Sources for: Synthesis/Implementation
controller
xc2v1000-6fg256
controller - Behavioral (../controller.vhd)
del1 - calc_cmp - Behavioral (../calc_cmp.vhd)
del2 - fsm - Behavioral (../fsm.vhd)
del3 - timetest - Behavioral (../timetest.vhd)
del4 - delay - Behavioral (../delay.vhd)
del5 - conv2 - Behavioral (../conv2.vhd)
del6 - intface2 - Behavioral (../intface2.vhd)
controller.ucf (controller.ucf)
controller_cs.cdc (controller_cs.cdc)
Processes
Processes:
Implement Design
Translate
Map
Place & Route
Place & Route Report
Clock Region Report
Asynchronous Delay Report
Pad Report
Guide Results Report
MPPR Results Utilities
Generate Post-Place & Route Static Timing
Analyze Post-Place & Route Static Timing
Generate Primetime Netlist
View/Edit Placed Design (Floorplanner)
View/Edit Routed Design (FPGA Editor)
Analyze Power (XPower)
Generate Power Data
Generate Post-Place & Route Simulation Mode
Generate IBIS Model
Back-annotate Pin Locations

Fig 3.22: Timing Analyzer

2. Exit the Timing Analyzer by **File→Exit**.

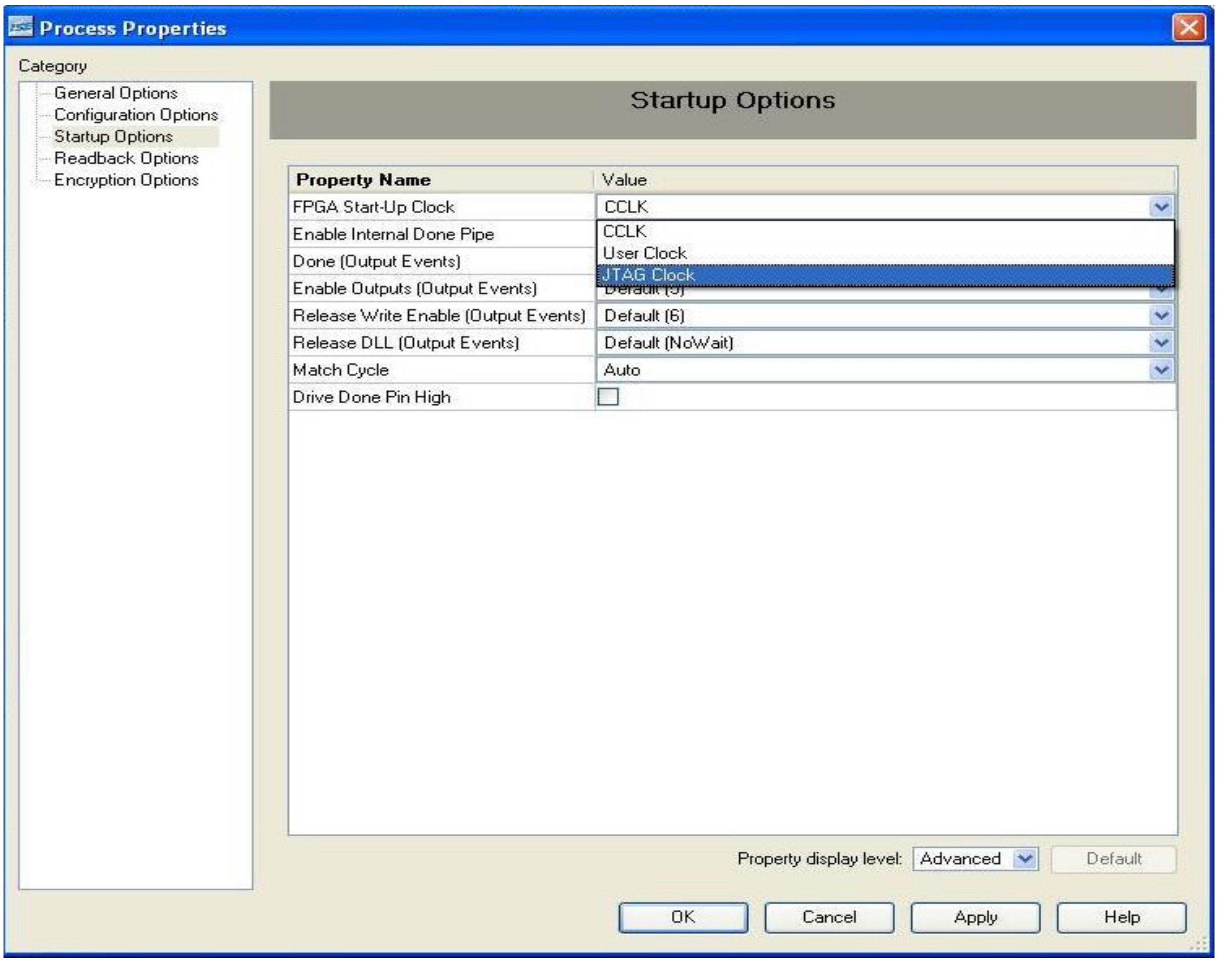
3.8.16 Place and Route:

Once after going through the mapping process, the design which is plotted to logical blocks are placed on to the FPGA and then routed.

3.8.17 Generating Programming File:

Programming file is a file which is acceptable and understandable by an FPGA device for configuration. One FPGA device requires one bit file in a boundary scan chain. We can generate the bit file only after the design is completely routed. Since, we are using the Chipscope Logic Analyzer with a JTAG interface we need to change the programming file properties.

1. Right click on the **Generate Programming File** process and select the **Properties**.



Property Name	Value
FPGA Start-Up Clock	CCLK
Enable Internal Done Pipe	
Done (Output Events)	
Enable Outputs (Output Events)	Default (5)
Release Write Enable (Output Events)	Default (6)
Release DLL (Output Events)	Default (NoWait)
Match Cycle	Auto
Drive Done Pin High	☐

Fig 3.23: Changing Properties of Generating Bit File

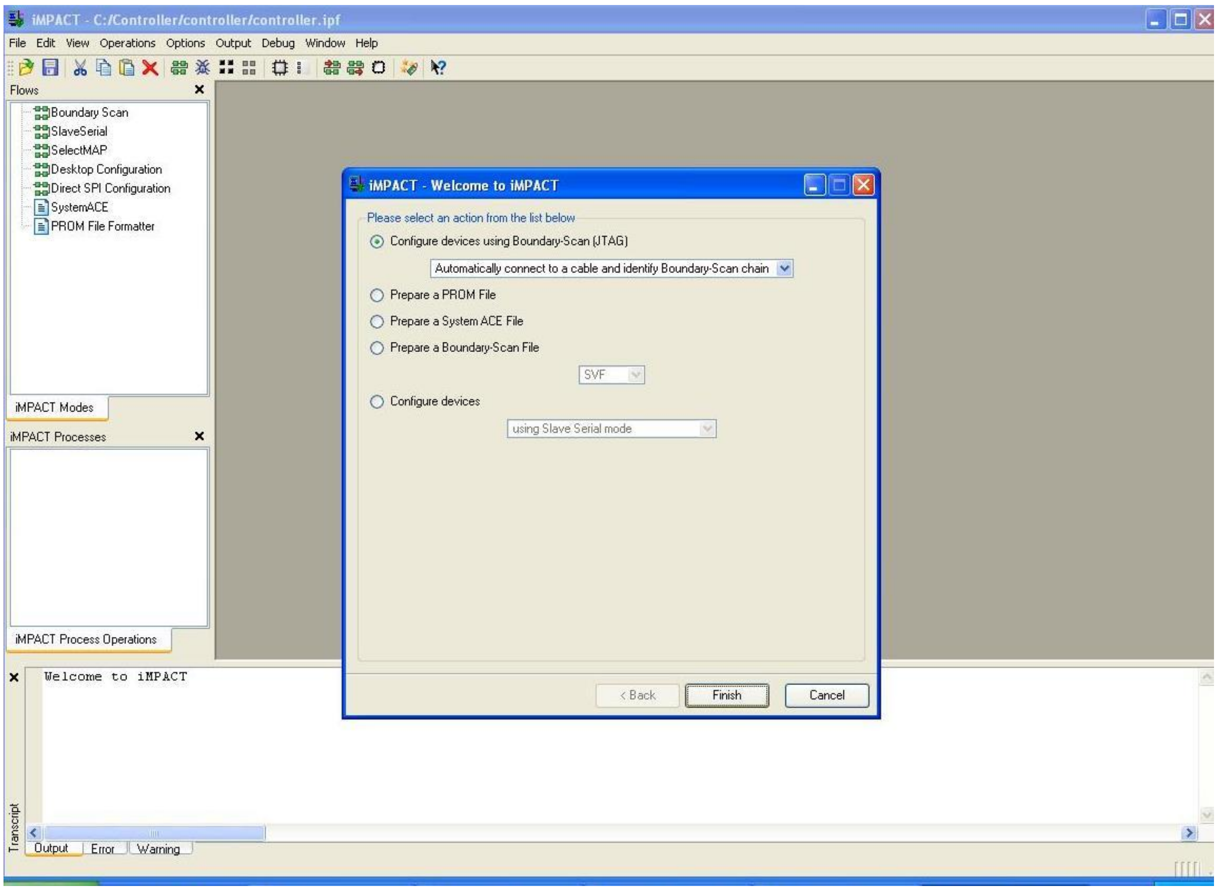
2. Select **Startup Options** under category and select **FPGA Start-Up Clock** to **JTAG Clock** as viewed from Fig 3.23 and click on **Apply** and select **OK** to exit the window.

3. Now Double click on **Generate Programming File** to produce a bit stream of this Design.

3.8.18 Configuring Device:

It is the process of loading design-specific information onto FPGA device using Parallel cable-IV.

1. Double Click on the **Configure Device (iMPACT)** then, a dialogue box appears which looks like as in Fig 3.24.



Please select an action from the list below

[✓ Configure devices using Boundary-Scan (JTAG)]

Automatically connect to a cable and identify Boundary-Scan chain

Prepare a PROM File

Prepare a System ACE File

Prepare a Boundary-Scan File

SVF

Configure devices

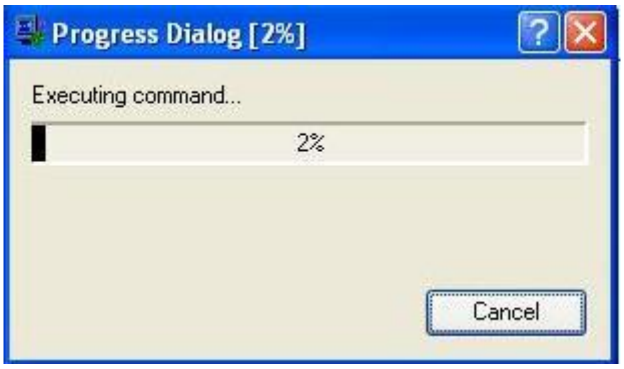
using Slave Serial mode

```
Welcome to iMPACT
```

Fig 3.24: Configuring Device

2. Select the **Configure devices using Boundary Scan (JTAG)** and click on **Finish**. Selecting this option will automatically connects to a cable and identifies the device in Boundary Scan Chain.

3. Then Progress dialog box opens which looks like as in Fig 3.25.



Executing command...

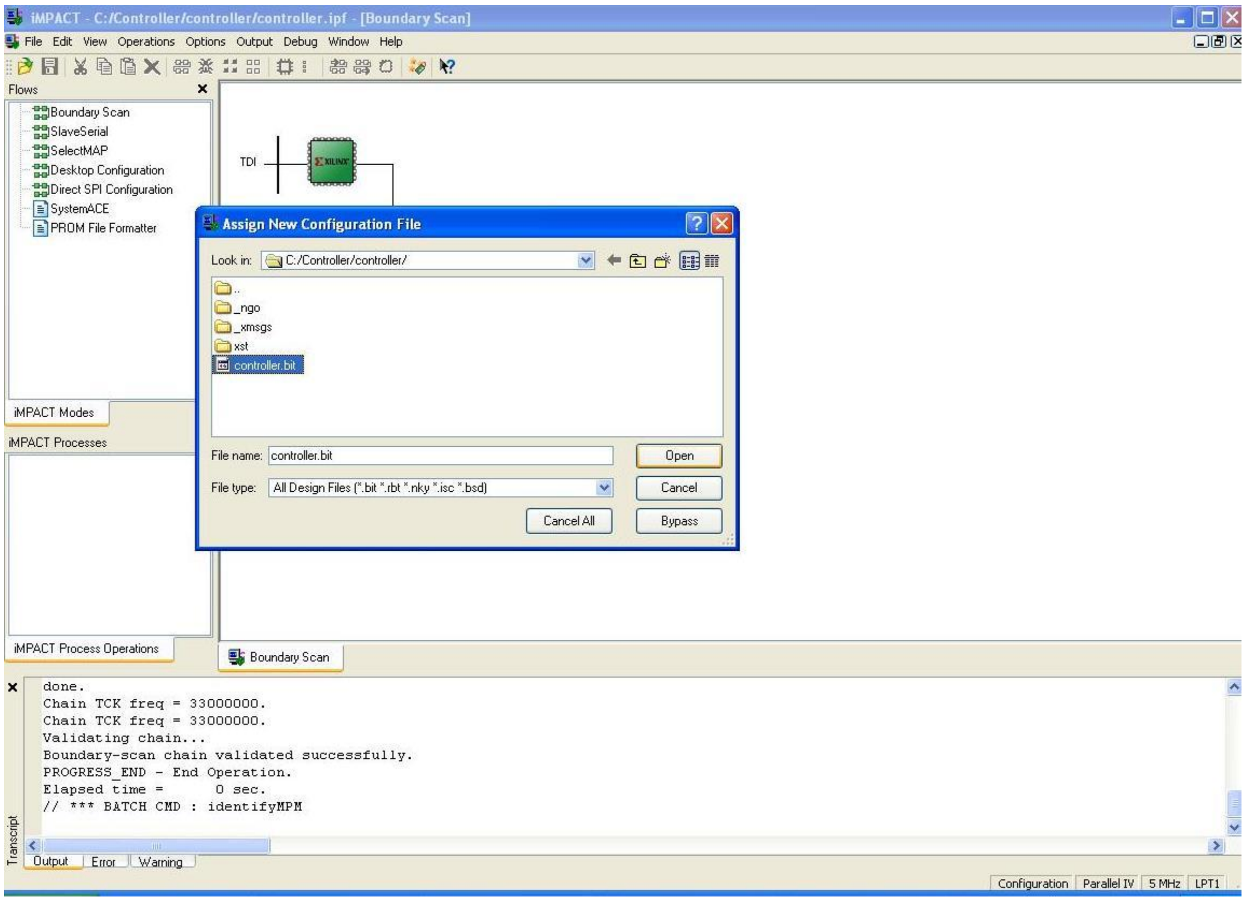
2%

Fig3.25: Configuring Device Data

4. The devices which are detected in JTAG chain will be displayed in iMPACT window.

5. The **Assign New Configuration File** dialog box opens. Now assign the **Controller.bit** file which can be viewed from Fig 3.26 to the device found in the JTAG chain.

6. Click **Open**.



Look in: C:/Controller/controller/

..
_ngo
_xmsgs
xst
controller.bit

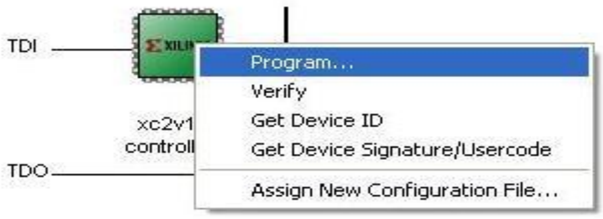
File name: controller.bit

File type: All Design Files (*.bit *.rbt *.nky *.isc *.bsd)

```
done.
Chain TCK freq = 33000000.
Chain TCK freq = 33000000.
Validating chain...
Boundary-scan chain validated successfully.
PROGRESS_END - End Operation.
Elapsed time =      0 sec.
// *** BATCH CMD : identifyMPM
```

Fig 3.26: Assigning Configuration File

7. Now right click on the **Device** and select **Program** as shown in Fig 3.27 so that it starts to program the device.



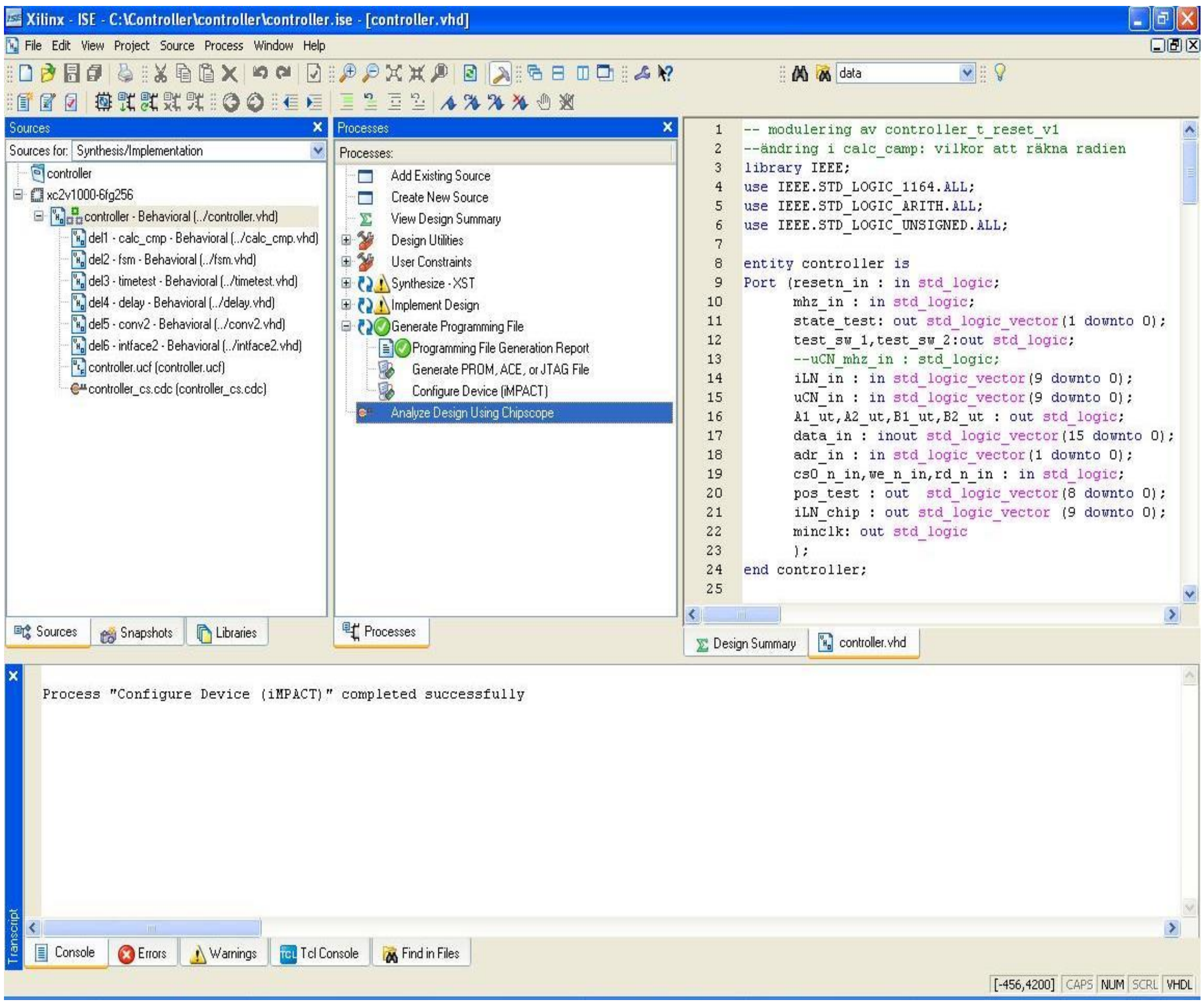
TDI
xc2v1
controll
TDO
Program...
Verify
Get Device ID
Get Device Signature/Usercode
Assign New Configuration File...

Fig 3.27: Programming the Device

8. A **Program Succeeded** message is displayed once after completely programming the device. Close **iMPACT** without saving.

Program Succeeded

9. Once after Configuring **iMPACT** was successful, return to the main window of the Project Navigator. Now double click on **Analyze Design Using Chipscope** in the **Processes** tab as shown in Fig 3.28.



Xilinx - ISE - C:\Controller\controller\controller.ise - [controller.vhd]

```
-- modulering av controller_t_reset_v1
--ändring i calc_camp: vilkor att räkna radien
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity controller is
Port (resetn_in : in std_logic;
      mhz_in : in std_logic;
      state_test: out std_logic_vector(1 downto 0);
      test_sw_1,test_sw_2:out std_logic;
      --uCN_mhz_in : std_logic;
      iLN_in : in std_logic_vector(9 downto 0);
      uCN_in : in std_logic_vector(9 downto 0);
      A1_ut,A2_ut,B1_ut,B2_ut : out std_logic;
      data_in : inout std_logic_vector(15 downto 0);
      adr_in : in std_logic_vector(1 downto 0);
      cs0_n_in,we_n_in,rd_n_in : in std_logic;
      pos_test : out  std_logic_vector(8 downto 0);
      iLN_chip : out std_logic_vector (9 downto 0);
      minclk: out std_logic
      );
end controller;
```

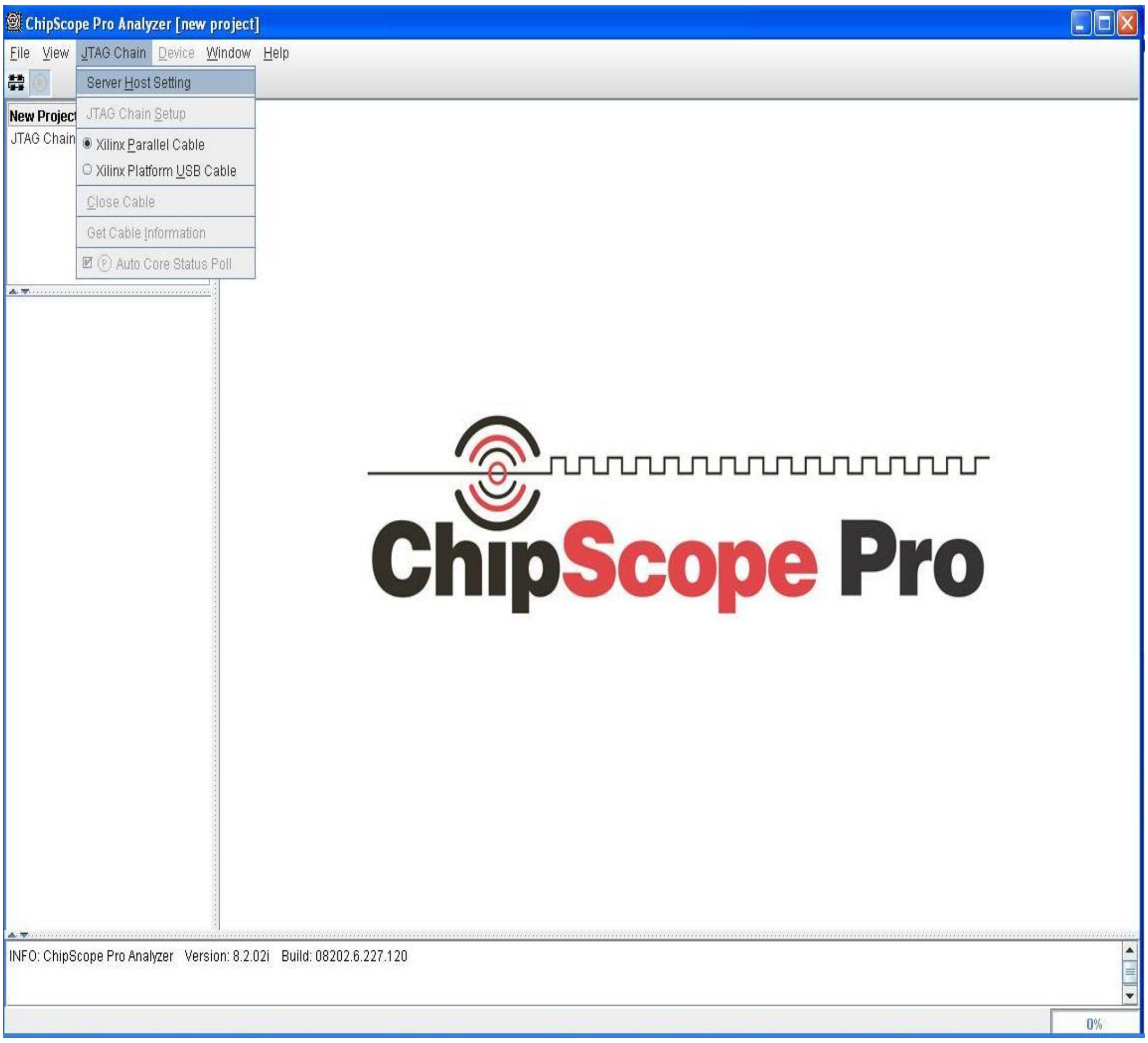
```
Process "Configure Device (iMPACT)" completed successfully
```

File Edit View Project Source Process Window Help
data
Sources
Sources for: Synthesis/Implementation
controller
xc2v1000-6fg256
controller - Behavioral (../controller.vhd)
del1 - calc_cmp - Behavioral (../calc_cmp.vhd)
del2 - fsm - Behavioral (../fsm.vhd)
del3 - timetest - Behavioral (../timetest.vhd)
del4 - delay - Behavioral (../delay.vhd)
del5 - conv2 - Behavioral (../conv2.vhd)
del6 - intface2 - Behavioral (../intface2.vhd)
controller.ucf (controller.ucf)
controller_cs.cdc (controller_cs.cdc)
Processes
Processes:
Add Existing Source
Create New Source
View Design Summary
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Programming File Generation Report
Generate PROM, ACE, or JTAG File
Configure Device (iMPACT)
Analyze Design Using Chipscope
Sources
Snapshots
Libraries
Processes
Design Summary
controller.vhd
Transcript
Console
Errors
Warnings
Tcl Console
Find in Files
[-456,4200]
CAPS
NUM
SCRL
VHDL

Fig 3.28: Selecting option to analyze Design using Chipscope

4. ANALYZING CORES OF DESIGN USING CHIPSCOPE LOGIC ANALYZER

4.1 Opening the project:



INFO: ChipScope Pro Analyzer Version: 8.2.02i Build: 08202.6.227.120

Fig 4.1: Server Host Setting

1. Here, the **Chipscope Pro Analyzer** application is directly opened from the Project Navigator's source file called **Chipscope Connection and Definition File**. This implies that there is no requirement of creating a new project.
2. The analyzer's client application needs a connection with the server. To do that, select the **JTAG Chain** and click on **Server Host Setting** as shown in Fig 4.1.
3. This pops out with a dialog box. This box consists of **Server Settings** which includes **Server Port** number and **Password**. The default port number used is **localhost: 50001.** For the local mode setting, password is not necessary.

4. Click on **OK** to complete the step as shown in Fig 4.2.

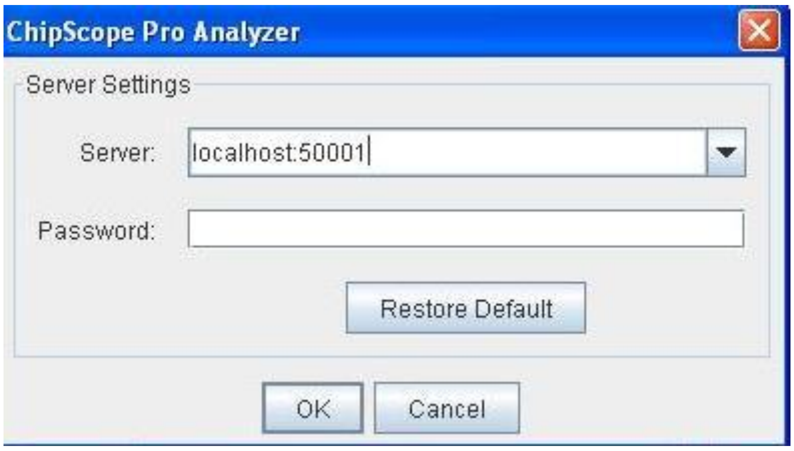


Fig 4.2: Server Settings

4.2 Opening Xilinx Parallel Cable:

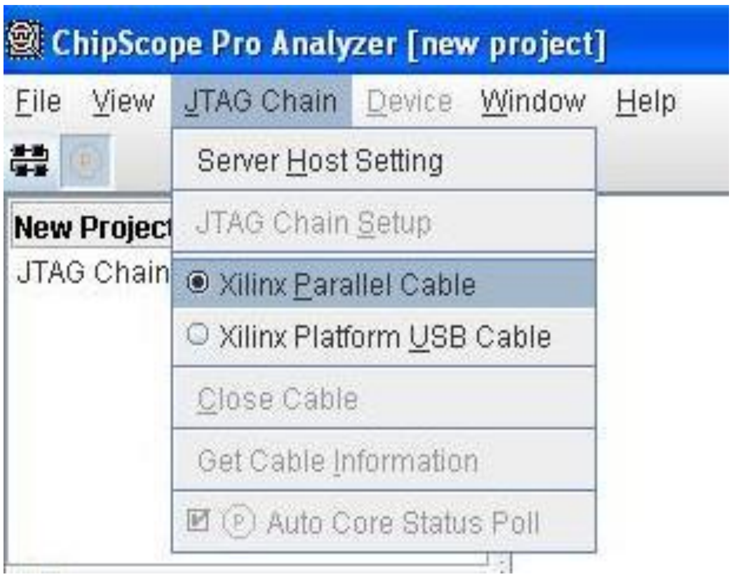


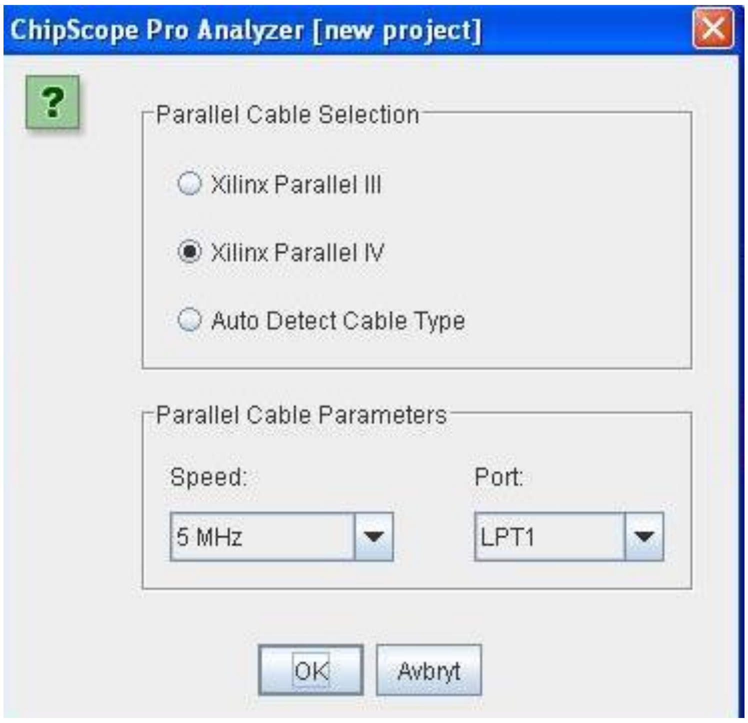
Fig 4.3: Opening of Parallel Cable

1. Make sure that the cable is connected to one of the parallel ports of the computer.

2. Select the **JTAG Chain** and Click on **Xilinx Parallel Cable** as shown in Fig 4.3. This pops out with a dialogue box for the selection of **Parallel Cable Selection** and its parameters like **Speed** and P**ort**.

3. If the cable used is Parallel Cable IV, then we have a chance of selecting the speed of the cable. Some of the speeds available are 625MHz, 1.25MHz, 2.5MHz, 5MHZ or 10 KHz [3]. Fig 4.4 describes the selection of parallel cable parameters.

4. If something goes wrong with the boundary scan chain composition, then it implies that the cable is not connected to the correct port.


ChipScope Pro Analyzer [new project]
?
Parallel Cable Selection
Parallel Cable Parameters
OK
Avbryt

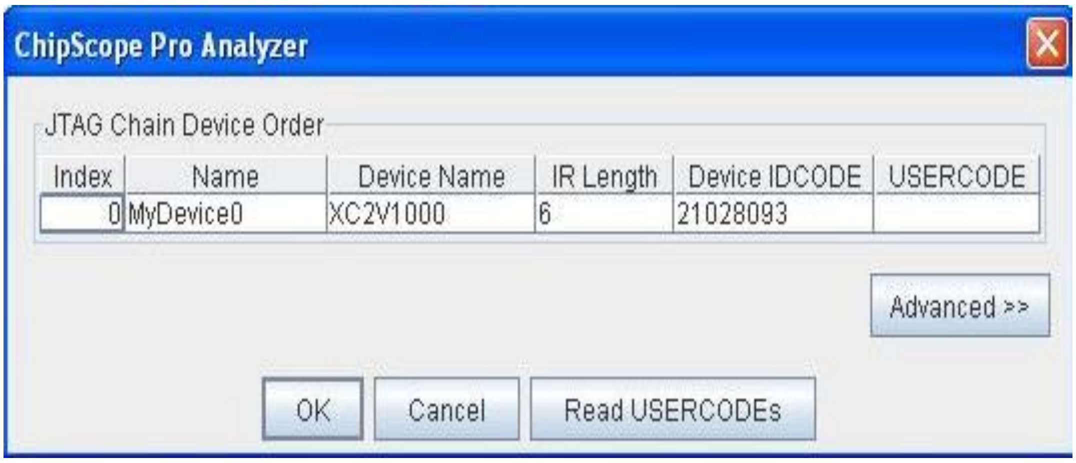
Xilinx Parallel III [✓ Xilinx Parallel IV] Auto Detect Cable Type

Speed: 5 MHz

Port: LPT1

Fig 4.4: Parallel Cable Parameters

4.3 Setting Boundary Scan Chain:



Index	Name	Device Name	IR Length	Device IDCODE	USERCODE
0	MyDevice0	XC2V1000	6	21028093	

Fig 4.5: JTAG Chain Device Order

1. The analyzer queries through the successfully downloaded cable for the devices in Boundary Scan Chain.
2. Here in this project, the detected device is **XC2V1000** and the **IR Length** is nothing but its Instruction Register length used for the Chipscope Pro Cores.

3. **Device ID CODE** is an eight digit hexadecimal code generated by an option chosen by a user during Generating Bit File options **– g UserID**.
4. By selecting **Read USERCODEs** option which can be seen in Fig 4.5, **USERCODE** can be examined from the device.

4.4 Configuration of the Device:

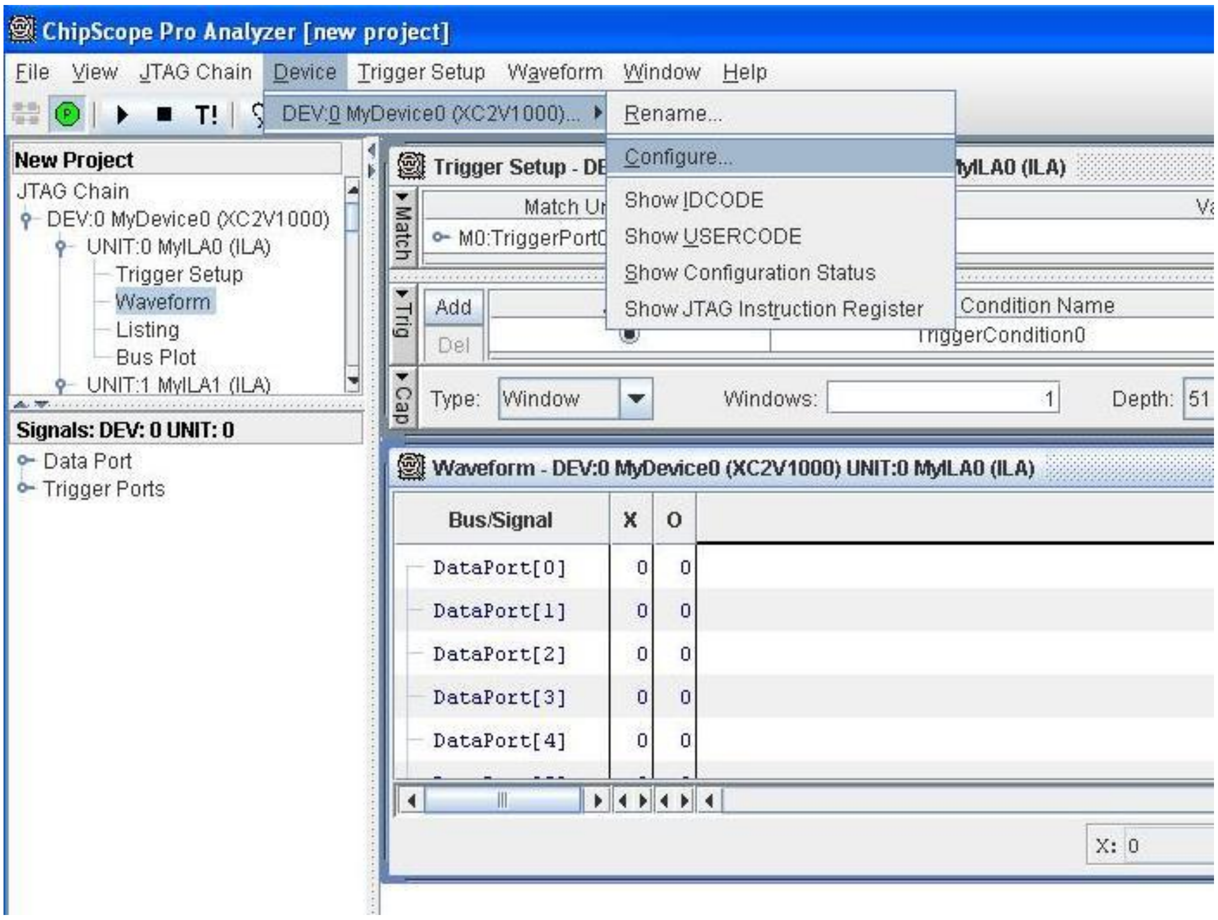


Fig 4.6: Device Configuration

1. If the FPGA device is to be configured with the downloaded cable in JTAG mode, select the device which we desire to configure and select the **Configure** option from **Configure menu option** as shown in Fig 4.6.

2. This pops out **Configuration dialog box** which includes a configuration file to be chosen and this choice by default will be found empty.

3. Now click on **Select New File** to select the **BIT file** to be downloaded for Configuring the Device.

4. Select the configuring file using browser which is already generated with proper **BitGen** settings.

5. Now click **open** to download the bit file as shown in Fig 4.7.

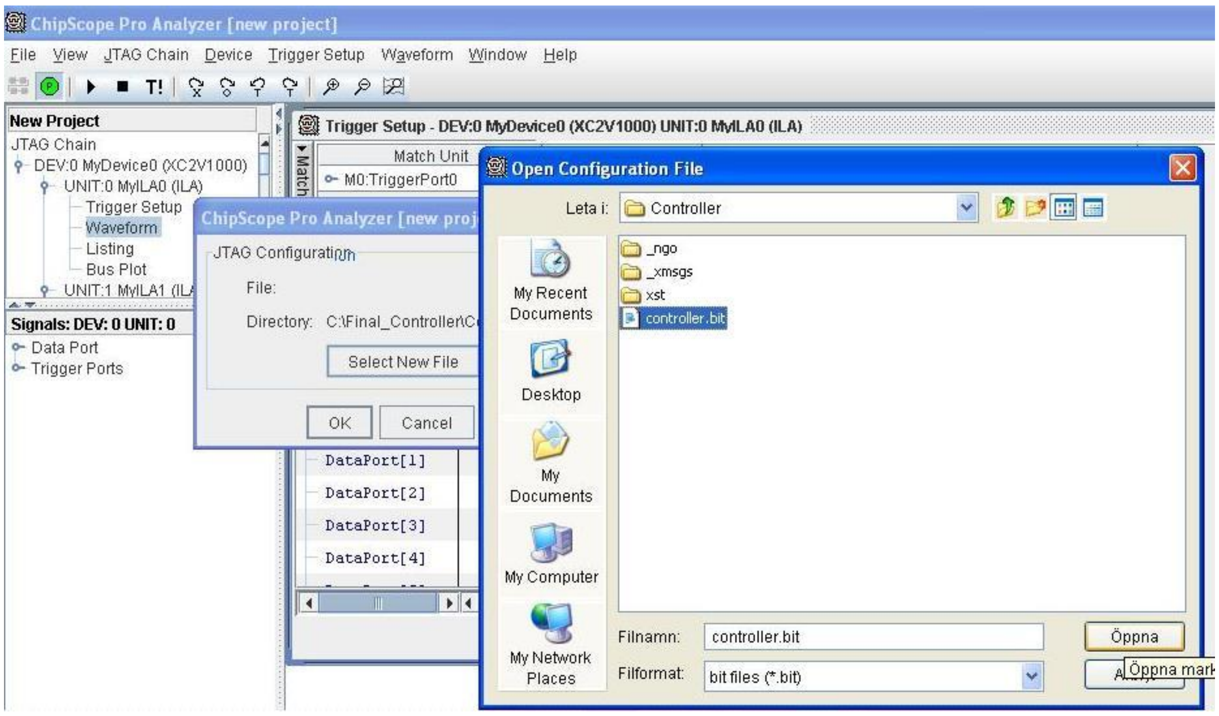


Fig 4.7: Open Configuration File

4.5 Plots Window:

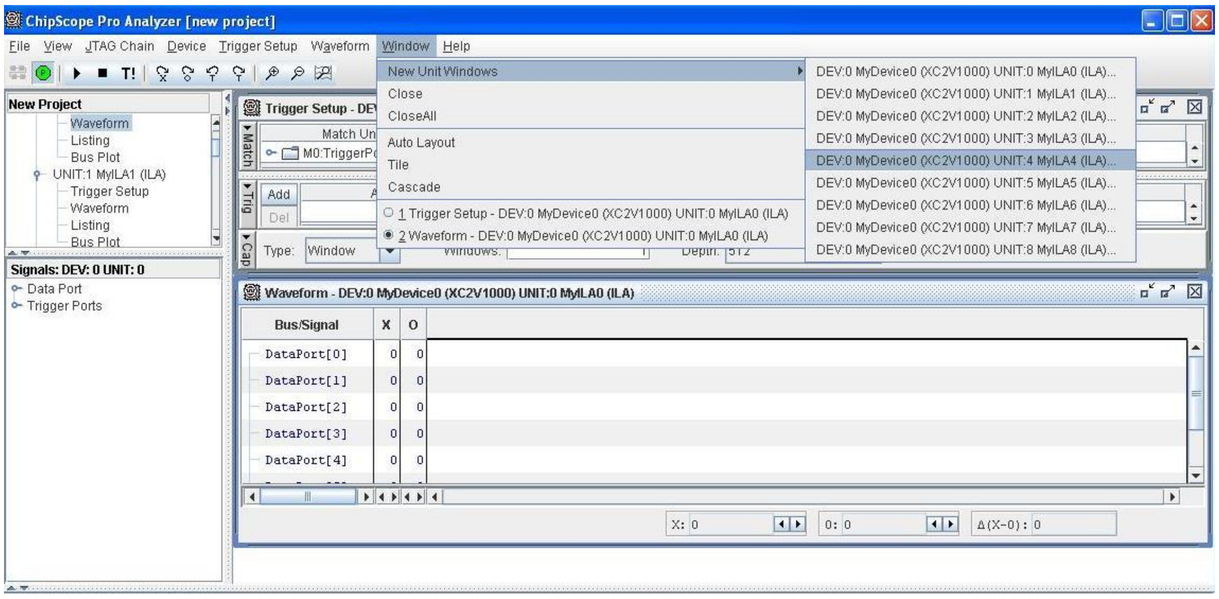

ChipScope Pro Analyzer [new project]
File View JTAG Chain Device Trigger Setup Waveform Window Help
New Unit Windows
Close
CloseAll
Auto Layout
Tile
Cascade
1 Trigger Setup - DEV:0 MyDevice0 (XC2V1000) UNIT:0 MyILA0 (ILA)
2 Waveform - DEV:0 MyDevice0 (XC2V1000) UNIT:0 MyILA0 (ILA)
DEV:0 MyDevice0 (XC2V1000) UNIT:0 MyILA0 (ILA)...
DEV:0 MyDevice0 (XC2V1000) UNIT:1 MyILA1 (ILA)...
DEV:0 MyDevice0 (XC2V1000) UNIT:2 MyILA2 (ILA)...
DEV:0 MyDevice0 (XC2V1000) UNIT:3 MyILA3 (ILA)...
DEV:0 MyDevice0 (XC2V1000) UNIT:4 MyILA4 (ILA)...
DEV:0 MyDevice0 (XC2V1000) UNIT:5 MyILA5 (ILA)...
DEV:0 MyDevice0 (XC2V1000) UNIT:6 MyILA6 (ILA)...
DEV:0 MyDevice0 (XC2V1000) UNIT:7 MyILA7 (ILA)...
DEV:0 MyDevice0 (XC2V1000) UNIT:8 MyILA8 (ILA)...
Waveform - DEV:0 MyDevice0 (XC2V1000) UNIT:0 MyILA0 (ILA)
Bus/Signal X O
DataPort[0] 0 0
DataPort[1] 0 0
DataPort[2] 0 0
DataPort[3] 0 0
DataPort[4] 0 0

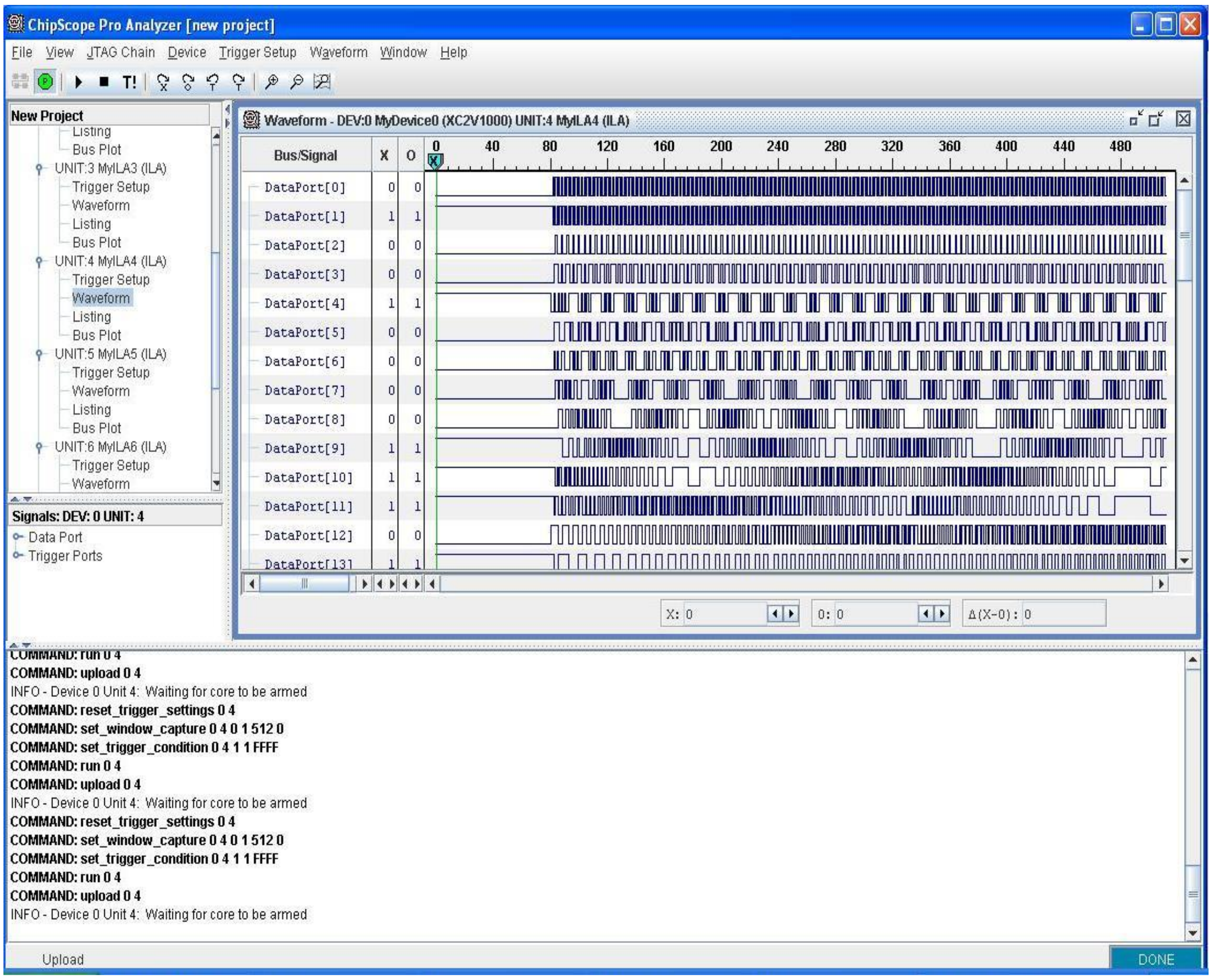
Fig 4.8: waveform window

1. To view a particular ILA’s trigger input waveform, select **Window** and click on **New Unit Windows** which shows all the ILA’s present in the design.

2. It pops out a dialog box with options which we choose, **Trigger setup**, **Waveform**, **Listing** and **Bus plot** as shown in Fig 4.8.

3. There is chance to perform various operations on the waveform window like the **Bus creation**, **renaming** and **selection of the radix type**.

4. Now set the criteria of the design. Select the **Data depth** or **Number of the samples** to be viewed.



Bus/Signal	X	O
DataPort[0]	0	0
DataPort[1]	1	1
DataPort[2]	0	0
DataPort[3]	0	0
DataPort[4]	1	1
DataPort[5]	0	0
DataPort[6]	0	0
DataPort[7]	0	0
DataPort[8]	0	0
DataPort[9]	1	1
DataPort[10]	1	1
DataPort[11]	1	1
DataPort[12]	0	0
DataPort[13]	1	1

```
COMMAND: run 0 4
COMMAND: upload 0 4
INFO - Device 0 Unit 4:  Waiting for core to be armed
COMMAND: reset_trigger_settings 0 4
COMMAND: set_window_capture 0 4 0 1 512 0
COMMAND: set_trigger_condition 0 4 1 1 FFFF
COMMAND: run 0 4
COMMAND: upload 0 4
INFO - Device 0 Unit 4:  Waiting for core to be armed
COMMAND: reset_trigger_settings 0 4
COMMAND: set_window_capture 0 4 0 1 512 0
COMMAND: set_trigger_condition 0 4 1 1 FFFF
COMMAND: run 0 4
COMMAND: upload 0 4
INFO - Device 0 Unit 4:  Waiting for core to be armed
```

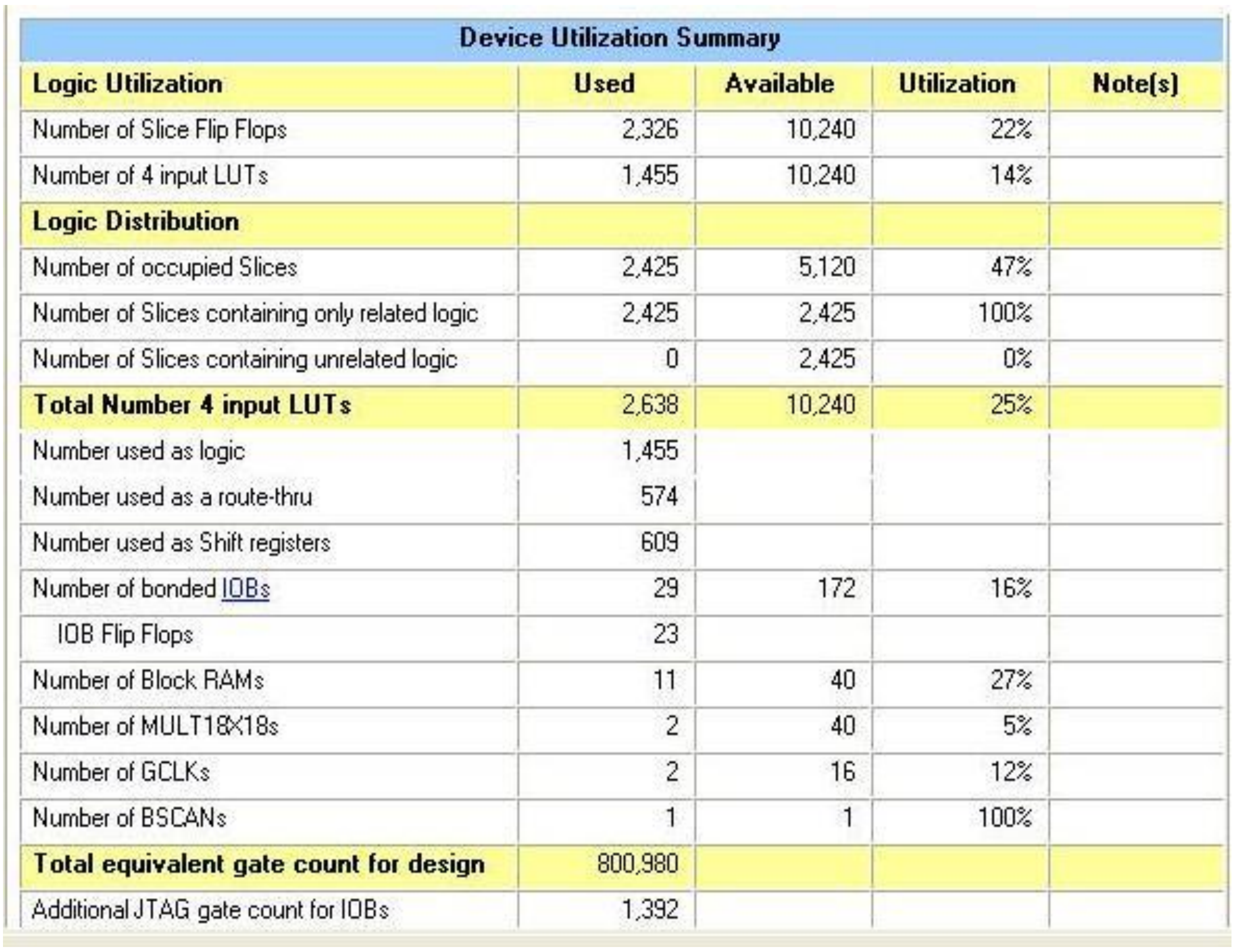
Fig 4.9: Triggered Waveforms

5.The waveforms in the Fig 4.9 can be obtained by clicking on **T!** icon. There are **X** and **O** markers used to accurate the calculations.

6. If there is a change in the selection of trigger waveforms of an ILA , the whole process of selecting the trigger inputs in Chipscope Core Inserter, Synthesizing and Implementing the Design should be re-iterated.

4.6 Results: Design Summary

After returning to the Project Navigator, click on the **Summary** of **Design Overview** in **FPGA Design Summary tab**. A winodw pops out on the right most side of the Project Navigator as shown in the Fig 4.10. The efficiency of controller which is implemented on the FPGA can be better understood by the following table.



Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	2,326	10,240	22%	
Number of 4 input LUTs	1,455	10,240	14%	
Logic Distribution				
Number of occupied Slices	2,425	5,120	47%	
Number of Slices containing only related logic	2,425	2,425	100%	
Number of Slices containing unrelated logic	0	2,425	0%	
Total Number 4 input LUTs	2,638	10,240	25%	
Number used as logic	1,455			
Number used as a route-thru	574			
Number used as Shift registers	609			
Number of bonded IOBs	29	172	16%	
IOB Flip Flops	23			
Number of Block RAMs	11	40	27%	
Number of MULT18X18s	2	40	5%	
Number of GCLKs	2	16	12%	
Number of BSCANs	1	1	100%	
Total equivalent gate count for design	800,980			
Additional JTAG gate count for IOBs	1,392			

Fig 4.10: Device Utilization Summary

5. CHIPSCOPE PRO CORE GENERATOR

Chipscope Pro tool is used to integrate Hardware cores into FPGA devices to study both software and hardware behaviors. The Chipscope has two debugging tools namely the Core Generator and Core Inserter. Inserting the cores into the FPGA design by these tools enables to capture the access nodes, bus, interval signals and the other IP cores.

This chapter guides the user through the process of debugging using Chipscope Pro Core Generator where as the method of using Chipscope Pro Core Inserter was described in the chapter 4.

Chipscope Pro Core Generator is used to generate:

- Integrated Controller core(ICON)
- Integrated Logic Analyzer core(ILA)
- Integrated Bus Analyzer core connect On-chip Peripheral Bus core(IBA/OPB)
- Integrated Bus Analyzer core connect Processor Local Bus core(IBA/PLB)
- Virtual Input/output core(VIO)
- Integrated Bit Error Ratio Test core(IBERT)

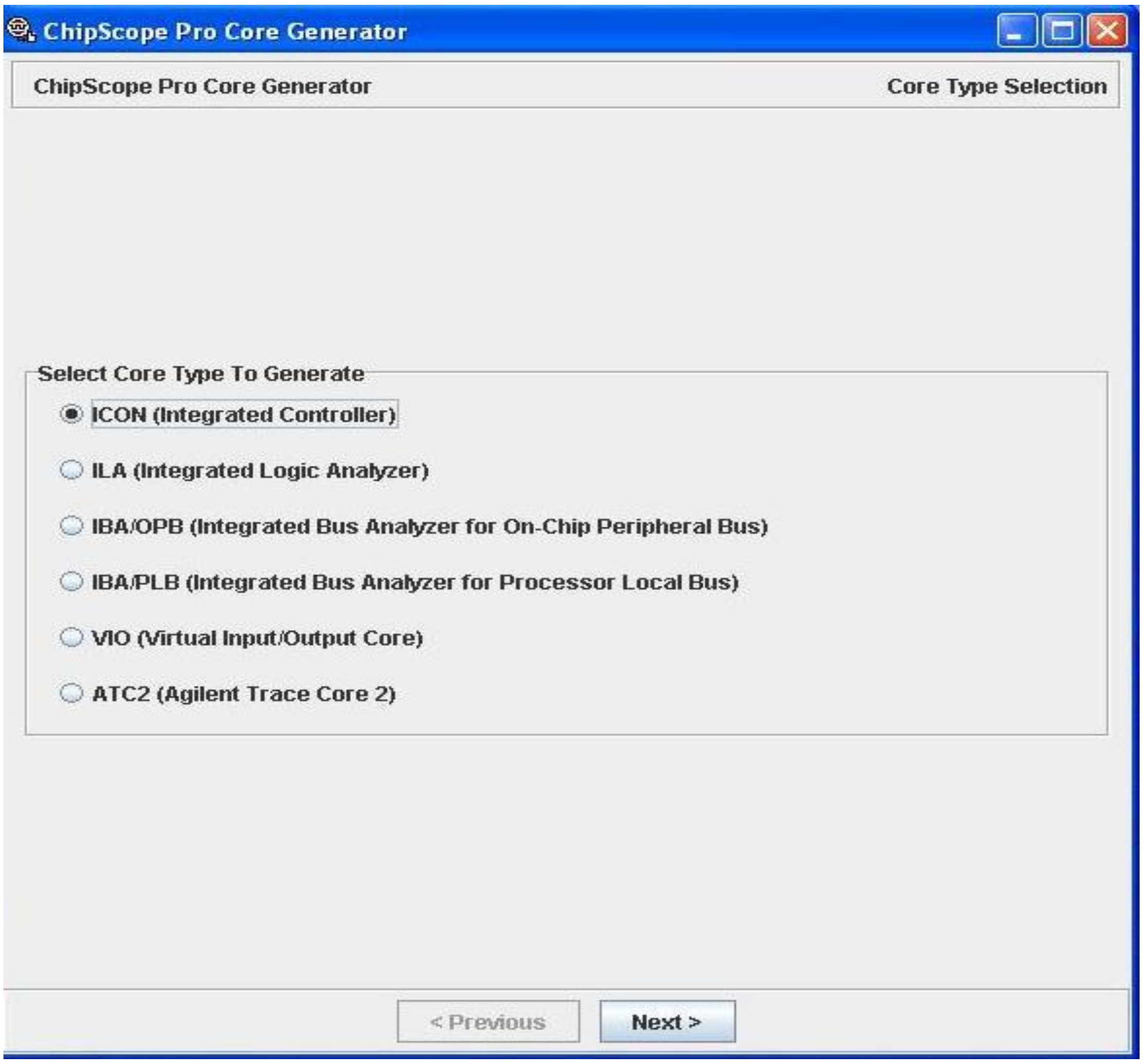
Once after generating the needed core for a project, the user has to instantiate them by inserting them into VHDL design. After instantiating cores user has to run Synthesis, Implementation and Generating Programming file [1].

5.1 Selecting type of core to be generated:

1. The very first window of core generator opens with selection of the core needed for the Project.

2. Now we select the **Integrated Controller (ICON)** core. If the project has Chipscope modules in the design then it is must to generate the ICON core. This core communicates with the PC and FPGA device through JTAG interface. This core controls all the other cores which are instantiated in VHDL coding.

3. So select **ICON** and click on **Next** as shown in Fig 5.1.


ChipScope Pro Core Generator

ChipScope Pro Core Generator — Core Type Selection

Select Core Type To Generate

- [✓ ICON (Integrated Controller)]
- ILA (Integrated Logic Analyzer)
- IBA/OPB (Integrated Bus Analyzer for On-Chip Peripheral Bus)
- IBA/PLB (Integrated Bus Analyzer for Processor Local Bus)
- VIO (Virtual Input/Output Core)
- ATC2 (Agilent Trace Core 2)

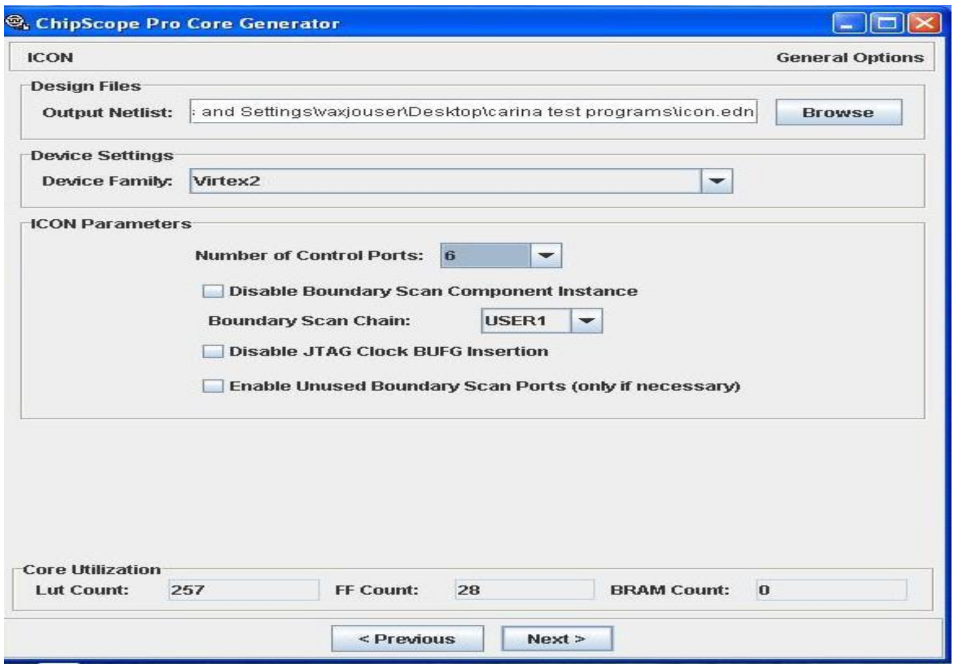
Fig 5.1: Selecting type of core

5.2 Selecting ICON settings and Parameters:

1. Choose the design files destination**.** By default the directory tends to an installed path. User can change this option and click on **Browse** to navigate the new path.

2. Selecting the **Device settings**: Click on the pull down list of **Device settings** and choose the family of the FPGA device.

3. **Choosing ICON Parameters**: Now Select the **Number of Control Ports** to be connected with ICON as shown in Fig 5.2. It can handle upto 15ILAs, IBA/OPB, IBA/PLB or VIO. User has to choose the same number of control ports as there are other cores in the design. For Example: If the project needs two ILA modules and one VIO module then the user has to generate an ICON with three control ports.


ChipScope Pro Core Generator

ICON — General Options

Design Files
Output Netlist: : and Settings\waxjouser\Desktop\carina test programs\icon.edn

Device Settings
Device Family: Virtex2

ICON Parameters
Number of Control Ports: 6
☐ Disable Boundary Scan Component Instance
Boundary Scan Chain: USER1
☐ Disable JTAG Clock BUFG Insertion
☐ Enable Unused Boundary Scan Ports (only if necessary)

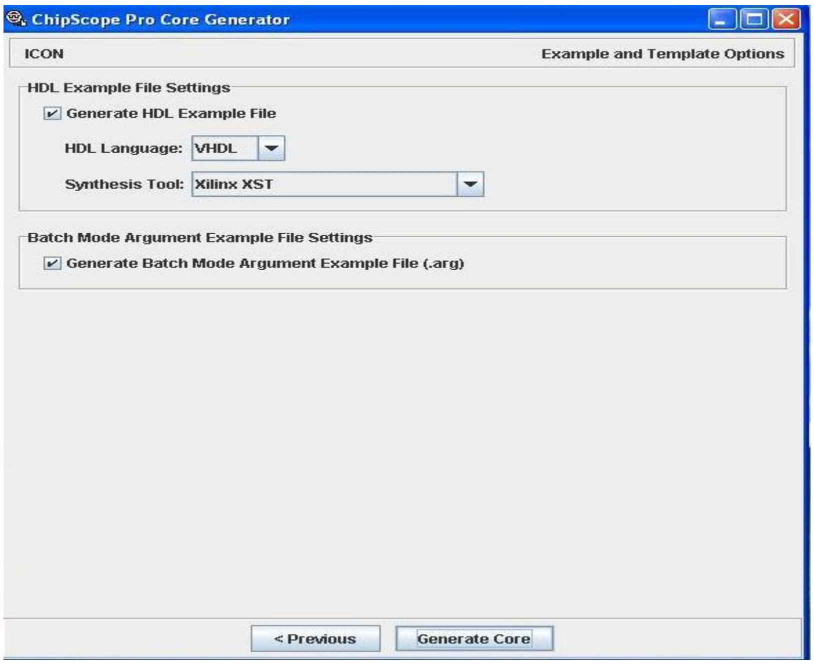
Core Utilization
Lut Count: 257 FF Count: 28 BRAM Count: 0

Fig 5.2: Selecting ICON Parameters

4. **Disabling Boundary Scan Component Instance**: By default this Boundary scan component is instantiated inside the ICON. Selecting this will disable the instantiation of Boundary scan component.

5. Make suitable Device settings and click on **Next**.

5.3 Example and Template Options:


ChipScope Pro Core Generator

ICON — Example and Template Options

HDL Example File Settings
☑ Generate HDL Example File
HDL Language: VHDL
Synthesis Tool: Xilinx XST

Batch Mode Argument Example File Settings
☑ Generate Batch Mode Argument Example File (.arg)

Fig 5.3: Selecting Example and Template Options

1. **HDL Example File Settings**: User can choose **Generate HDL Example File** which is used to instantiate into VHDL coding. The **Synthesis Tool** can be chosen as **Xilinx XST** as shown in Fig 5.3.

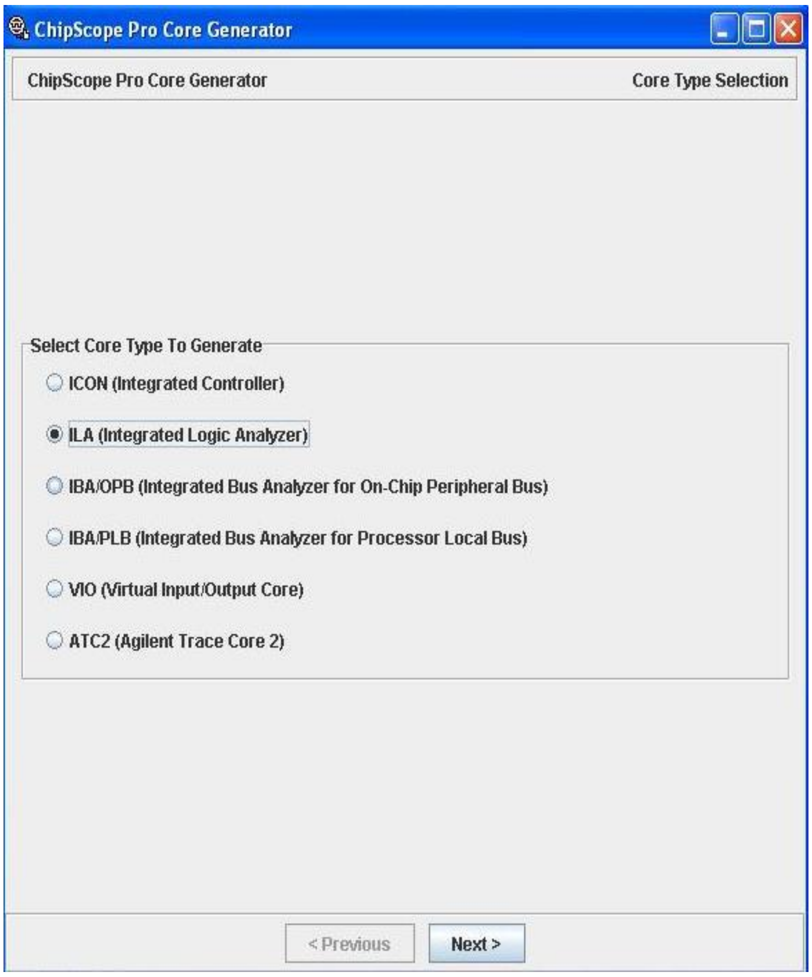
2. **Batch Mode Argument Example File Settings**: By selecting this option, it creates a batch mode argument example file which consists of all the arguments needed for generation of an ICON core.

3. Click on **Generate Core.** This creates the EDIF netlist, code examples and NCF file.

5.4 Generating ILA core:

1. After generating the core, the user can go back and **start over** again to generate new cores.

2. Now select the **ILA (Integrated Logic Analyzer)** and click **Next** as shown in Fig 5.4.


ChipScope Pro Core Generator

ChipScope Pro Core Generator — Core Type Selection

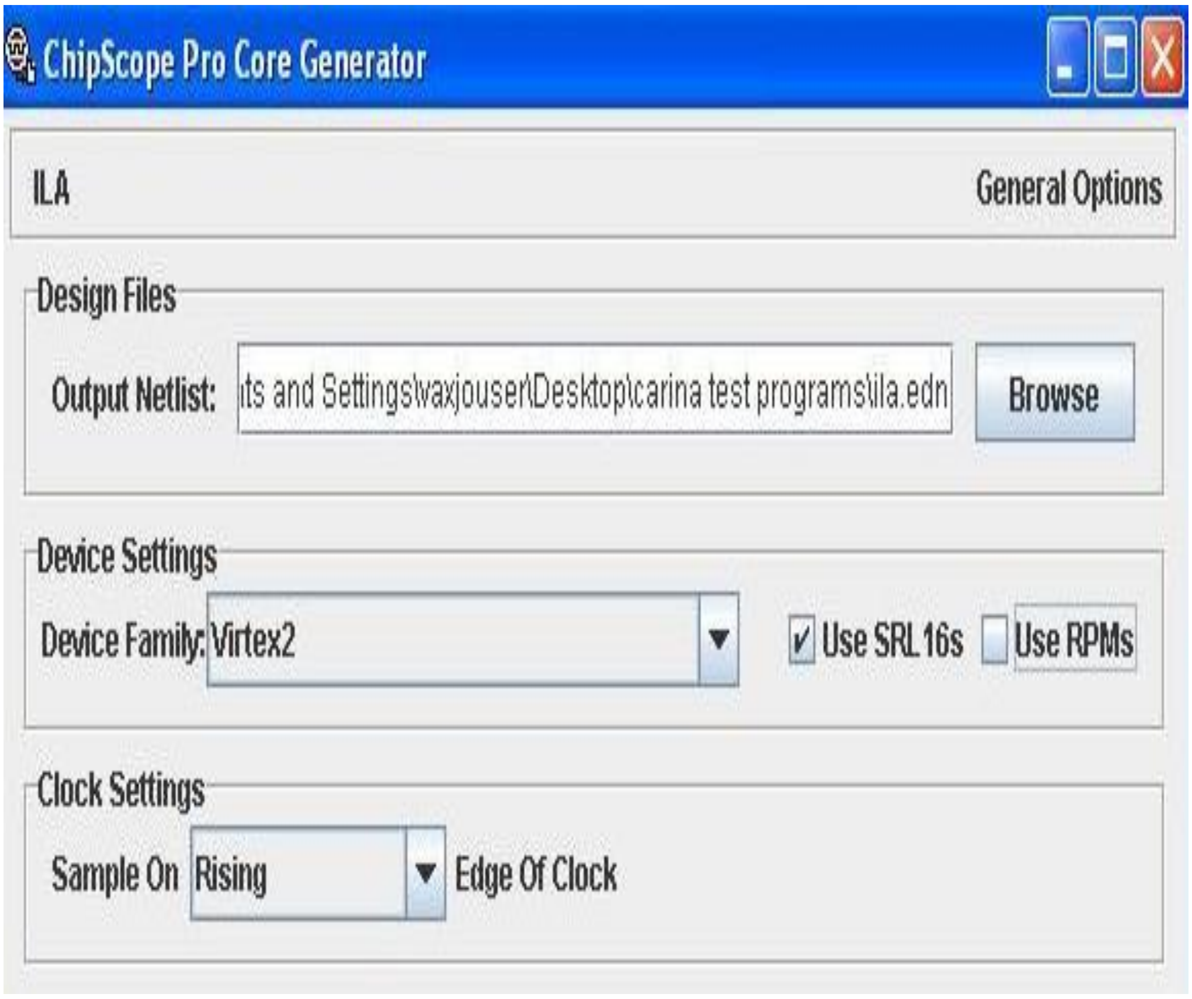
Select Core Type To Generate

- ICON (Integrated Controller)
- [✓ ILA (Integrated Logic Analyzer)]
- IBA/OPB (Integrated Bus Analyzer for On-Chip Peripheral Bus)
- IBA/PLB (Integrated Bus Analyzer for Processor Local Bus)
- VIO (Virtual Input/Output Core)
- ATC2 (Agilent Trace Core 2)

< Previous | Next >

Fig 5.4: Selecting the ILA Core

5.5 Selecting ILA core settings:


ChipScope Pro Core Generator

ILA General Options

Design Files

Output Netlist: its and Settings\waxjouser\Desktop\carina test programs\ila.edn

Device Settings

Device Family: Virtex2 [✓ Use SRL16s] Use RPMs

Clock Settings

Sample On Rising Edge Of Clock

Fig 5.5: ILA Core settings

1. User can click on **Browse** to save the **Design Files** in a new directory. By default this destination is made to installed path of core generator.

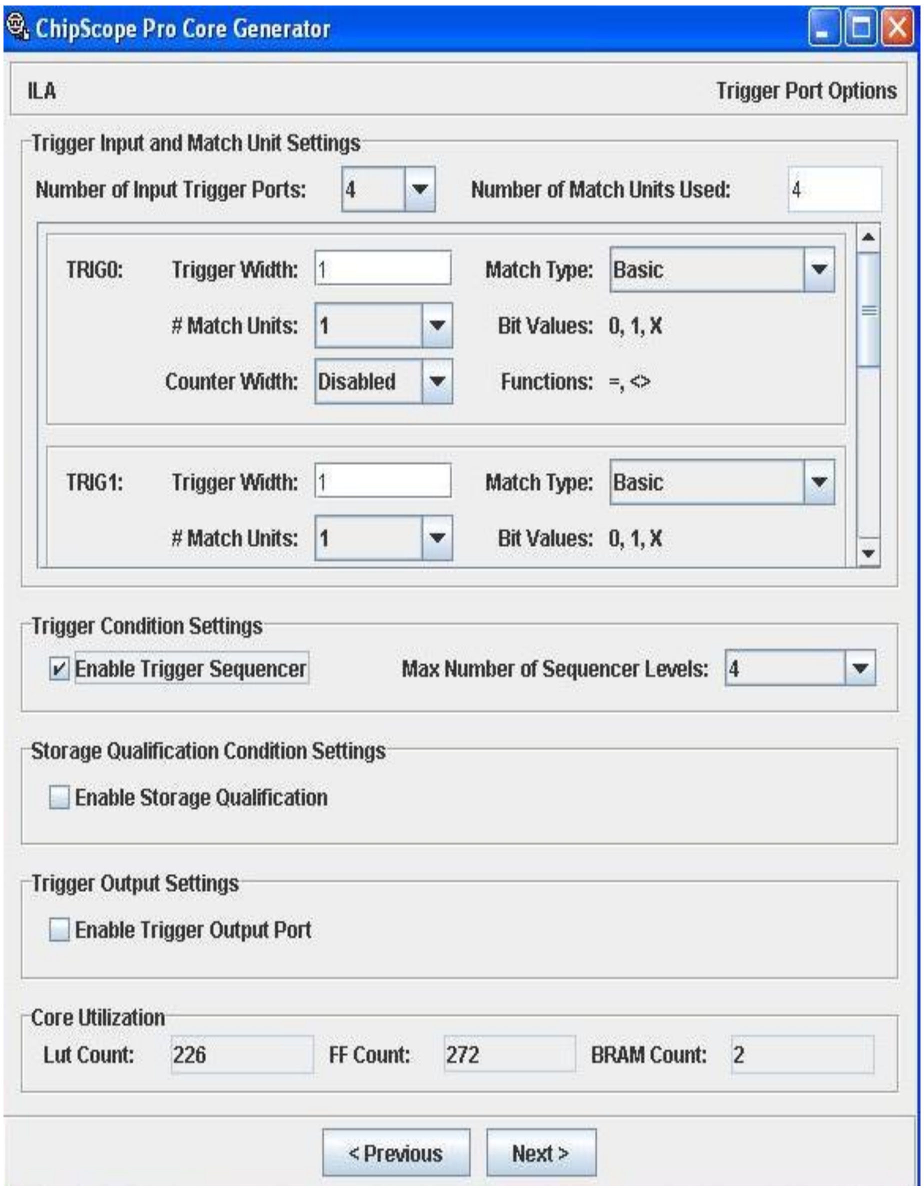
2. Now select the FPGA **Device Family** to appropriate device used in the project as shown in the Fig 5.5.

3. Also select the **Use SRL16s** and **Use RPMs** according to the project settings. The best usage of these options is described in chapter 4.

4. The User can either select the samples on **Rising** or **Falling** Edge of clock signal to capture the Data.

5. Now Click on **Next** to finish with these settings.

5.6 Selecting Trigger Input Settings:



Trigger Input and Match Unit Settings
Number of Input Trigger Ports: 4
Number of Match Units Used: 4

TRIG0: Trigger Width: 1
Match Type: Basic
Match Units: 1
Bit Values: 0, 1, X
Counter Width: Disabled
Functions: =, <>

TRIG1: Trigger Width: 1
Match Type: Basic
Match Units: 1
Bit Values: 0, 1, X

Trigger Condition Settings
[✓] Enable Trigger Sequencer
Max Number of Sequencer Levels: 4

Storage Qualification Condition Settings
[ ] Enable Storage Qualification

Trigger Output Settings
[ ] Enable Trigger Output Port

Core Utilization
Lut Count: 226
FF Count: 272
BRAM Count: 2

Fig 5.6: Selecting Trigger input settings

1. **Trigger Input and Match Unit Settings**: Most of these settings are same as the Chipscope Pro Core Inserter settings. Select the **Number of Input Trigger Ports** for an ILA as shown in the Fig 5.6. Every Individual ILA can hold 16 separate trigger ports. User can select the number from the pull down list.

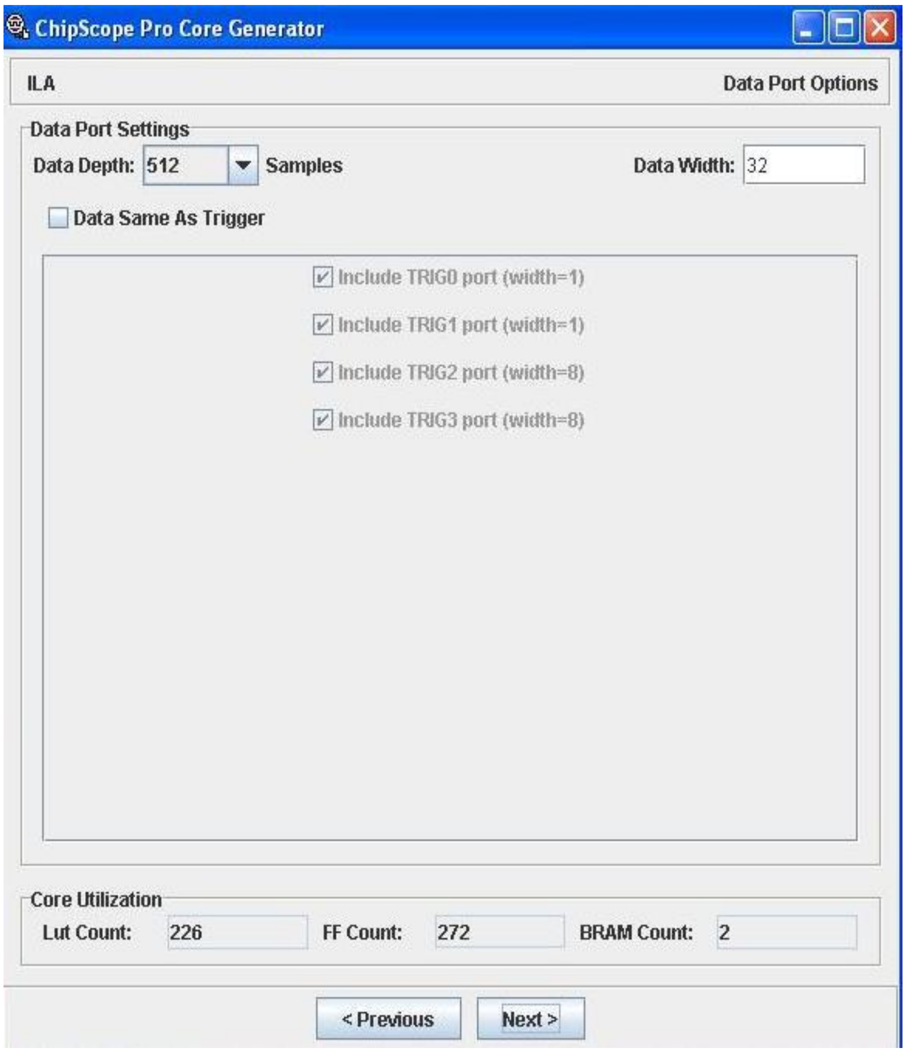
2. Now select the **Trigger width** of each signal. By default, we choose **Match Type** to **Basic**, **Match Units** to **1**, and **Counter width** to **Disabled**.

3. **Trigger Condition Settings**: Select **Enabling Trigger Sequencer** enables to configure from one level to the next at run time.

4. **Selecting Storage Qualification Condition Settings**: Enabling this option requires more usage of the resources. It is better not to choose this option if the design has limited resources.

5. Click **Next**.

5.7 Selecting ILA Data port settings:


ChipScope Pro Core Generator

ILA — Data Port Options

Data Port Settings

Data Depth: 512 Samples — Data Width: 32

☐ Data Same As Trigger

☑ Include TRIG0 port (width=1)

☑ Include TRIG1 port (width=1)

☑ Include TRIG2 port (width=8)

☑ Include TRIG3 port (width=8)

Core Utilization

Lut Count: 226 — FF Count: 272 — BRAM Count: 2

Fig 5.7: ILA Data port settings

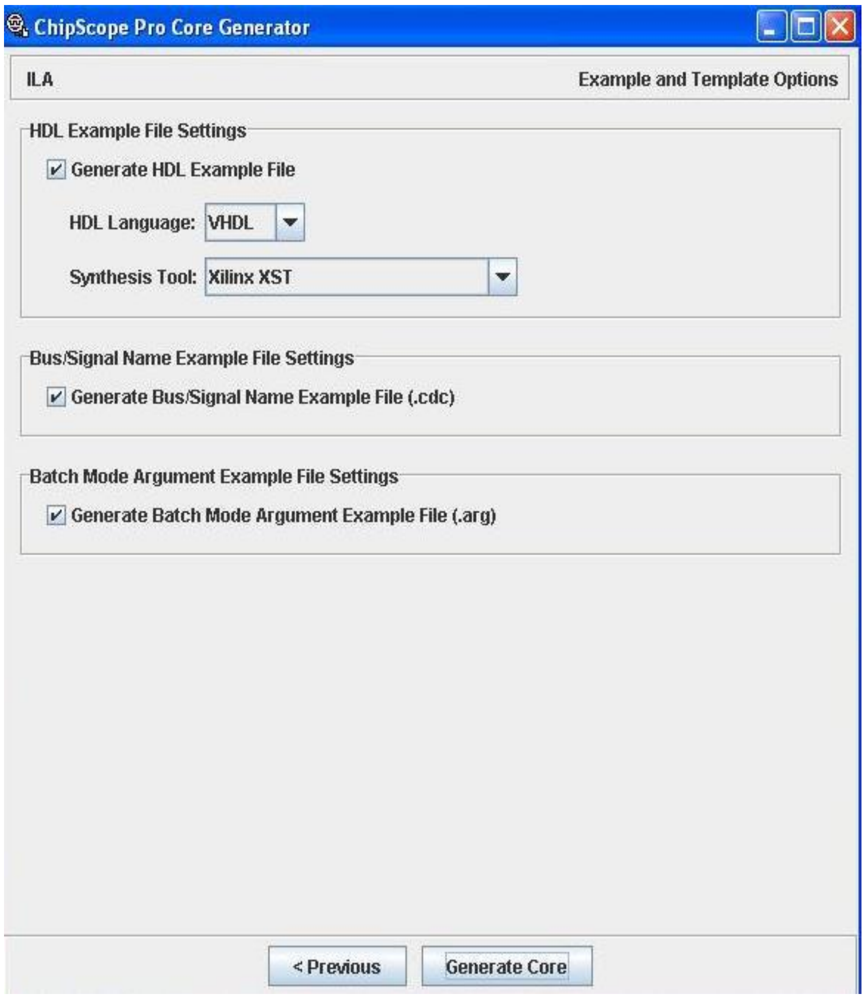
1. Capture settings: Select the ILA core’s maximum capacity to store the sample buffer which is called as Data Depth as shown in Fig 5.7. It is the count of data width bits contributed by each BRAM unit. The Block RAM resources are counted by the resource utilization estimator feature in Chipscope Core Inserter.

2. Selecting the data type: Data type can be of two sources.

- **Data separate from trigger**: This condition is chosen whenever we want to limit the amount of data to be captured. The data port is different from the trigger port.
- **Data same as trigger**: Both the trigger and data ports are same and identical. When this option is chosen then ILA core does not take any input separately for data.

3. In this Project, we select **Data same as trigger**. Click on **Next.**

5.8 HDL Example File Settings:



ChipScope Pro Core Generator

ILA — Example and Template Options

HDL Example File Settings
- [✓] Generate HDL Example File
- HDL Language: VHDL
- Synthesis Tool: Xilinx XST

Bus/Signal Name Example File Settings
- [✓] Generate Bus/Signal Name Example File (.cdc)

Batch Mode Argument Example File Settings
- [✓] Generate Batch Mode Argument Example File (.arg)

Fig 5.8: HDL Example File Settings

1. This generates the **HDL Example file** as shown in Fig 5.8 with all the trigger information given by the user for each ILA. This is the HDL file which has to be instantiated into the design.

2. Check on **Bus/Signal Name Example File Settings** and **Batch Mode Argument Example File Settings**.

3. Click on **Generate core**. The core will be generated in VHDL language.

5.9 Connecting ICON in User VHDL Program:

1. Open the generated core **controller_icon_xst_example.vhd** file.

2. Select and copy the part under **ICON Core Component Declaration** to user VHDL program just before the **ILA declaration** in the top module.

3. Copy the part under **ICON Core Signal Declaration** to user VHDL program after the **signal declaration** of all ILA signals.

4. Copy the part under **ICON Core Instance** to user VHDL program just before the **ILA Instance.**

5.10 Connecting ILA in User VHDL Program:

1. Open the generated core **controller_ila_xst_example.vhd** file.

2. Copy the part under **ILA Core Component Declaration** to User ILA VHDL program before **begin** statement of the **architecture**.

3. Copy the part under **ILA Core Instance** to User ILA VHDL program just before **end** of the **architecture**.

4. Now rename the Instance port Map signals.

    =>control, to=> control0

5. Rename the triggers to appropriate signals.

Now save all the files with the changes made in the VHDL programs of ICON and ILA's. Run the Synthesis, Implementation and Generating the Programming File processes, and analyze the logic using Chipscope Pro Analyzer.

6. MERITS AND DEMERITS OF CHIPSCOPE

6.1 Merits:

1. While analyzing the signals, the probe delays are reduced to a greater extent when compared to other external Logic analyzers.

2. Degradation of circuit performance caused due to probing is reduced.

3. Price of the Chipscope tool compared to External Logic Analyzers is very low.

4. Very convenient and portable to run and analyze the circuitry on an FPGA.

6.2 Demerits:

1. Designing clock frequency is very much faster than the sampling rate.

2. Resources availability is limited on an FPGA device because implementing a small design requires large amount of area on an FPGA.

6.3 Errors and Solutions:

Some of the errors which we came across during the project are discussed below. The solutions and suggestions also follow with each error.

Error1: Timing: 3223- Timing constraint PATH “TS_U_T0_D_path” TIG; ignored during timing analysis.
ParHelpers: 81- The following clock signals have USE LOW SKEW LINES constraint specified. The router was not able to completely route using the LOW SKEW resources.

Solution: For Timing: 3223 Warning, the constraint will be ignored if other constraints with higher priority override it. The warning can also be caused due to no path recognized in design. In trace there is an option “-tsi”. You can also check tsi report (*.tsi) whether any constraint was overridden.

Error2: NgdBuild: 455-logical net ‘N2’ has multiple driver(s):

Solution: This error indicates that one pin is driving more than 1 signal, or it has more than one source. Either multiple IBUF or OBUF components are connected in series. The best way is to remove this IBUF to eliminate the error.

Make sure that IBUFG component is not connected between 2 design elements or DCM.

Error3: ERROR: DesignRules: 10 - Net check: The signal "U_ila_pro_/i_no_d/u_ila/u_trig/u_tm/g_nmu/0/u_m/u_mu/i_mut_gand/u_matc h/cfg _data_63" is completely unrouted.

Solution: Change the match type from "Basic" to "Basic w/ edges".

- Change the trigger width to a number that, when divided by eight, do not leave a remainder of 1, 2, 3, or 4. For example if the trigger Width is 20, change it to 21.

- Disable RPM's.

Error4: When the Status Register read back by iMPACT is all ‘0’.

Solution: This may occur due to all pins pulled down to GND or this security option can be set in BitGen options.

7. CONCLUSION

FPGA has its wide applications in various fields with complex designs. In the past debugging and verification of such designs was a difficult task. Chipscope is a tool which is a complete solution for both hardware and software. It is a technology very helpful in implementation with timing predictability and effort.

In this project we have studied and analyzed both the Chipscope Pro Core Generator and Chipscope Pro Core Inserter software's, the former is most suitable for real time applications and the latter can be better used for prototype design verifications.

This methodology of using Chipscope gives the users a good flexibility to know the functionality of each part under design. The resources used by ICON and ILA are in few percents only, but the main limitation of the Chipscope methodology is the size of Block RAMs.

This paper includes Chipscope modules ICON and ILA for the run-time calculations of FPGA circuit. This thesis can be extended to generate virtual I/O module of Chipscope which allocates virtual inputs and outputs of the concerned FPGA circuit during real time debugging.

8. REFERENCES

[1] Xilinx, Chipscope Pro Software and Cores User Guide (Chipscope Pro Software 8.2i), UG029 (v8.2) September 18, 2006.

[2] Martien Spierings and Peter van Otterdijk, “Designing VHDL in Matlab, VHDL implementation using Xilinx System Generator in Matlab Simulink”, Bachelor thesis, Växjö University, Sweden, 2008.

[3] Jahnavi and Srinivasulu, Implementation of Observer on FPGA board, Master thesis, University of Kalmar, 2006.

[4] Orest Oltu, Petru Lucian Milea and Alexandru Simion, Testing of Digital circuitry using Xilinx Chipscope Logic Analyzer, Politehnica University of Bucharest, Romania, IEEE, 2005.