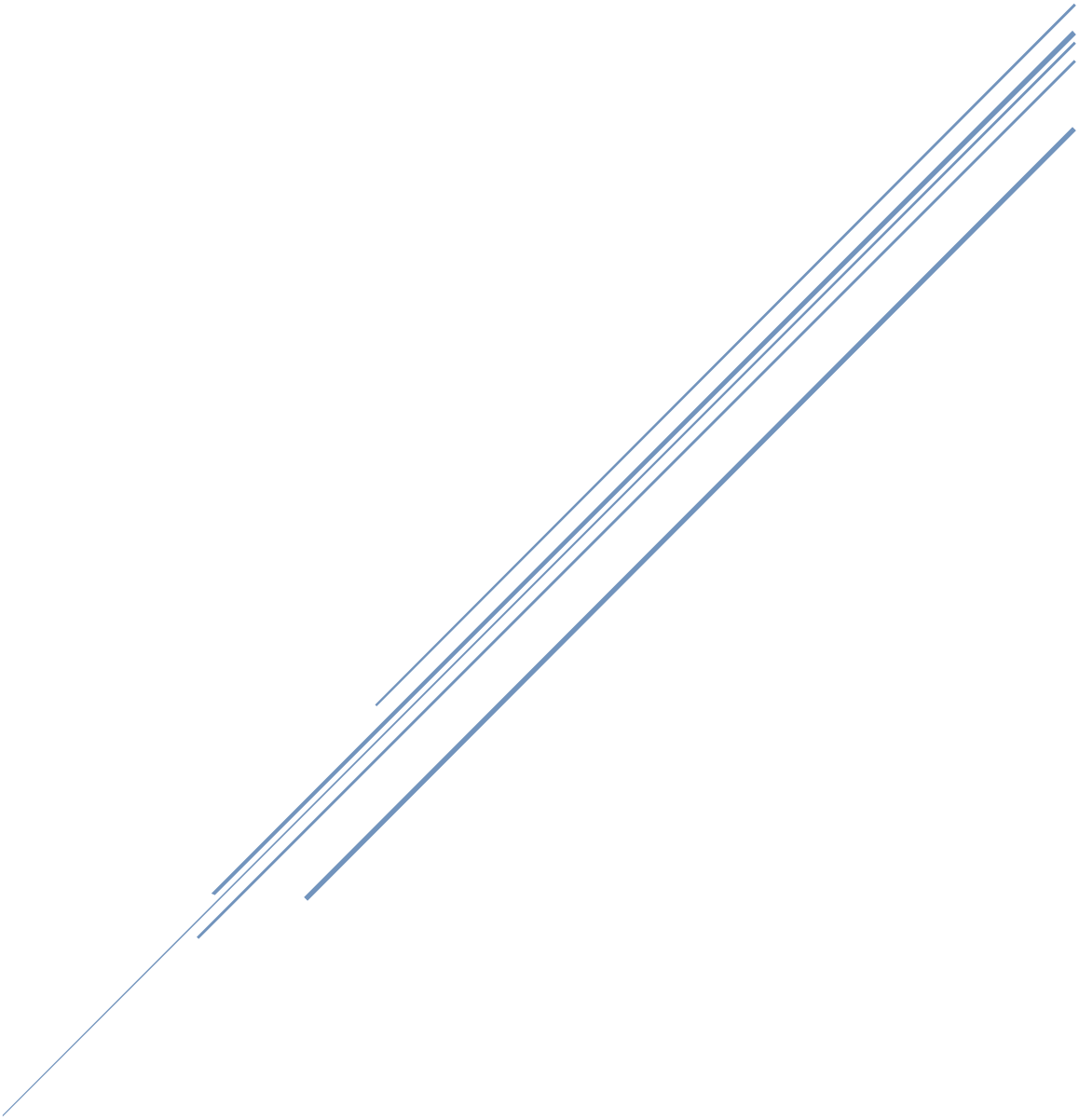
MANUAL FOR CPLD

Digital Circuits Lab



IIT Dharwad

This manual discusses the following chapters in detail:

- **Chapter 1**

  It depicts clear vision about the evolution of the different Programming Logic Devices.

  PROM, PAL,PLA and CPLD are explained briefly.

- **Chapter 2**
  It describes Altera 3000A CPLD. Necessary terminologies are explained in detail.

- **Chapter 3**
  It describes the Hardware Description Language in detail through certain examples.

- **Chapter 4**
  It explains how to familiarize with Quartus software for programming a logic function.

Chapter 1
Evolution

1.1 Introduction

A programmable logic device (PLD) is an electronic component used to build reconfigurable digital circuits. Unlike integrated circuits (IC) which consist of logic gates and have a fixed function, a PLD has an undefined function at the time of manufacture. Before the PLD can be used in a circuit it must be programmed (reconfigured) by using a specialized program. It reduces design time and thus reduces time for the product to reach the market. It consists of arrays of AND and OR gates, which can be programmed to realize required logic function.

1.1.1 Programmable Read Only Memory (PROM)

The first type of user programmable chip that could implement logic circuits was the Programmable Read Only Memory (PROM), in which address line can be used as logic circuits inputs and data lines as output. Since PLD consists of array of AND and OR gates we can design different logic functions. In PROM type PLD, AND array is fixed and OR array is programmable. This type PLD is able to produce all the minterms for a given set of variables. But most of logic function requires fewer number of terms than the entire minterm produced. Thus this makes an inefficient architecture for realizing logic circuits. Actually PROM is a memory chip on which data is written only once. Once a data is written onto it, it remains there forever. Therefore, it is also called as a **one-time programmable memory**. Memory chip is delivered blank and the programmer transfers the data on to it. A blank PROM consists of many fuses which can be selectively burned out during the first programming. They are used in applications like, computer bios where reprogramming is not required.

Internal structure of each block in the PROM is shown in the below figure. Here, A and B are the address inputs and Y is the data output. AND arrays are fixed to select each row (address location) for corresponding inputs. As shown in the figure, data in each memory location is determined by the fuses in the OR array. If the fuse is not burned off, charge

send through the row is received at the output, indicating a logic one and when the fuse is burned off, the signal cannot reach the data output and is detected as logic zero. Thus, in PROM binary data is stored using fuses.

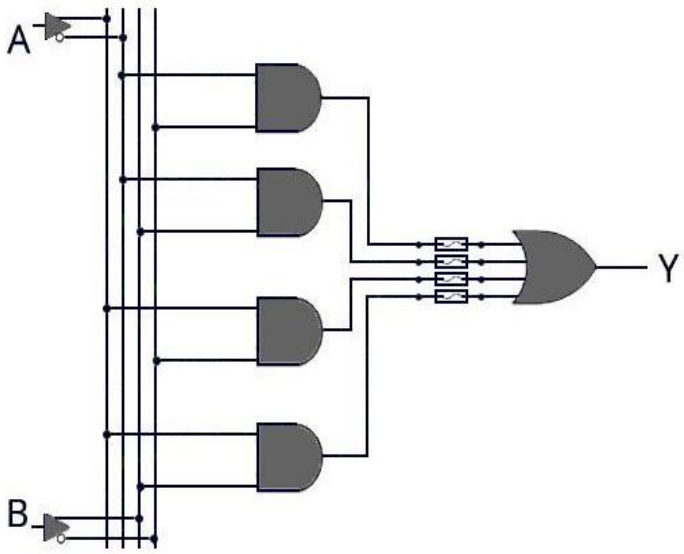


Fig 1.1 PROM

1.1.2. Programmable Logic Array (PLA)

The first device designed after PROM specially for implementing logic circuits was programmable Logic Array(PLA). It consists of two level of logic gates. A programmable AND array and a programmable OR array. In PLA, inputs and its complement form is available, so that any inputs can be AND together in AND plane and can develop product terms. These product terms can be added together in OR plane. PLA's are well suited for implementing the logic circuits. But it creates high expense for the manufacturer and offered poor speed performance. Since both AND plane and OR plane has to configure, it was difficult to manufacture. It also introduces significant delays in execution of the corresponding given function. To overcome all these problems Programmable Array Logic was introduced. The below figure depicts internal structure in which fuses are connected to the AND gate which indicates AND gate is programmable

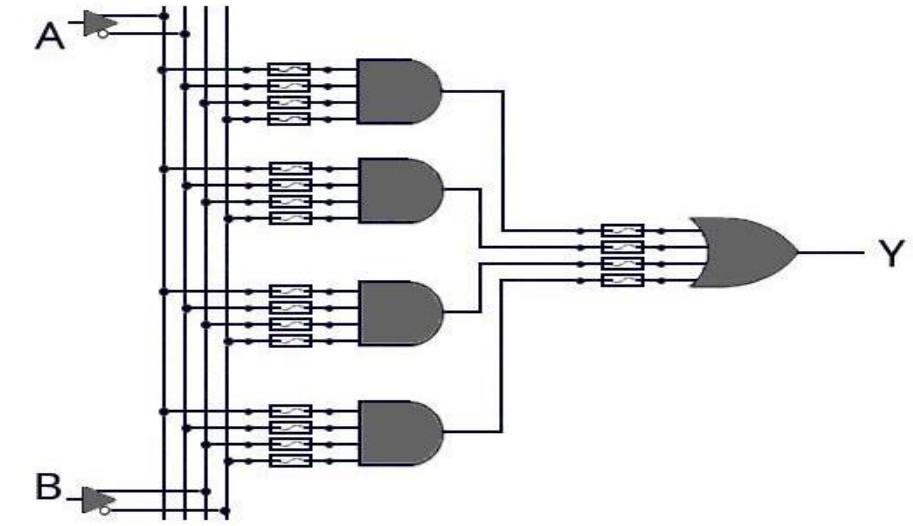


Fig.1.2 PAL

1.1.3 Programmable Array Logic (PLA)

In Programmable Array Logic AND plane is programmable and the OR plane is fixed. To compensate lack of generality caused by fixed OR plane several variants of PAL with different number of inputs and outputs and various size OR gates were implemented. PAL's usually contain flip-flops at the outputs of the OR gates, so that sequential circuits also can be realized. PAL and PLA are grouped to be called as **Simple Programmable Logic Devices**.

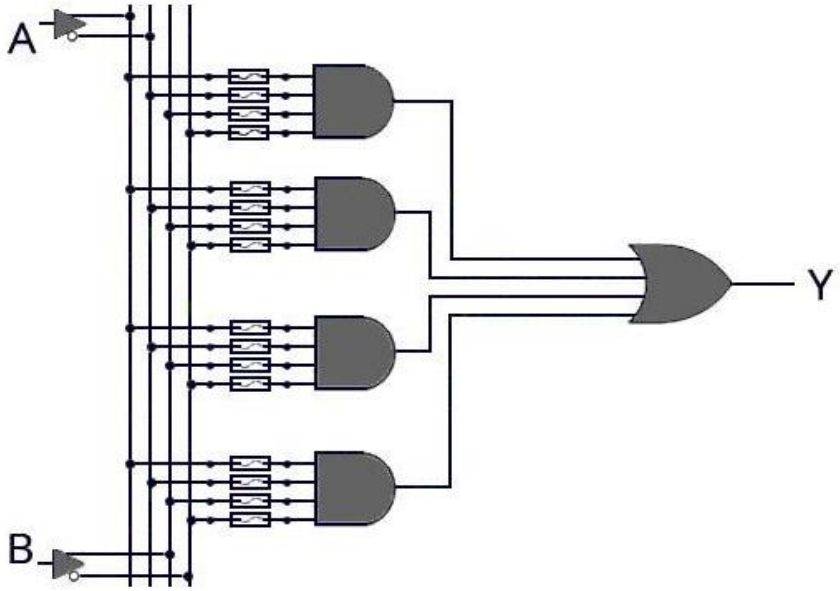


Fig.1.3 PLA

1.1.4 Complex Programmable Logic Device (CPLD)

As the technology advanced it has become possible to increase the capacity of SPLD's. The difficulty in increasing the capacity is the structure of programmable logic planes grow quickly in size as the number of input is increased. So the only feasible way is to integrate the SPLD onto a single chip and provide an interconnect to all the SPLD blocks. These SPLD blocks collectively called CPLD.

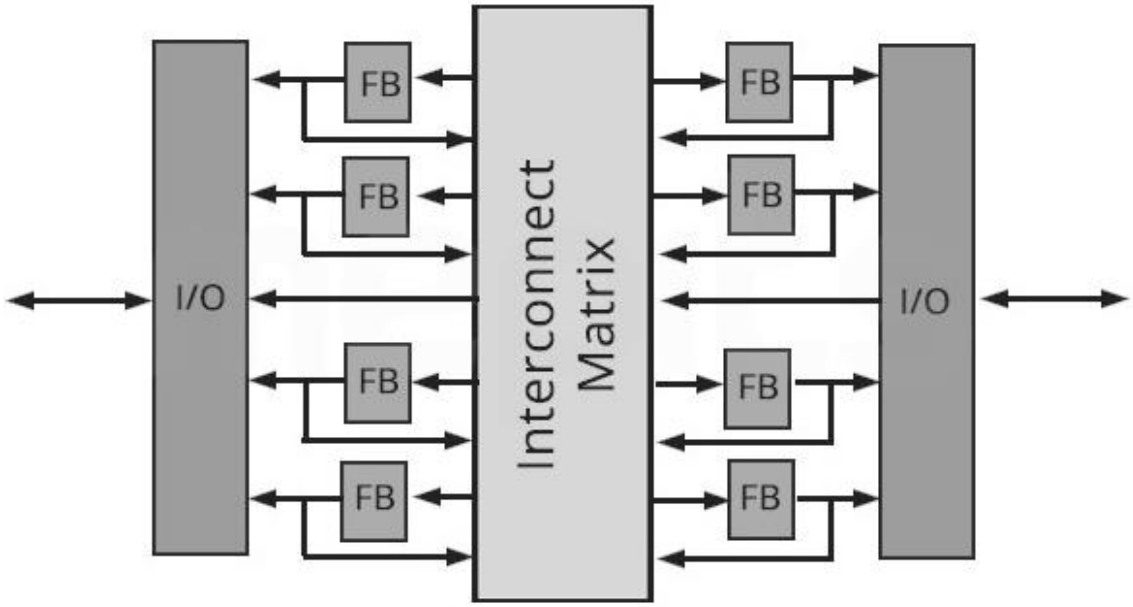


Fig 1.4 CPLD

CPLD is defined as the network of **PLDs** that are connected together through a switching matrix. General block diagram of a **CPLD** is shown here. The global interconnection matrix, as shown in the figure, is reconfigurable and so we can change the connections between the Functional Blocks depending on our requirement.

Chapter 2

CPLD MAX 3000A

2.1 Introduction

MAX 3000A devices are low–cost, high–performance devices based on the Altera MAX architecture. Fabricated with advanced CMOS technology, the EEPROM–based MAX 3000A devices operate with a 3.3-V supply voltage and provide 600 to 10,000 usable gates. MAX 3000A devices contain 32 to 512 macrocells, combined into groups of 16 macrocells called logic array blocks (LABs). Each macrocell has a programmable–AND/fixed–OR array and a configurable register with independently programmable clock, clock enable, clear, and preset functions. The following figure depicts the architecture of MAX 3000A.

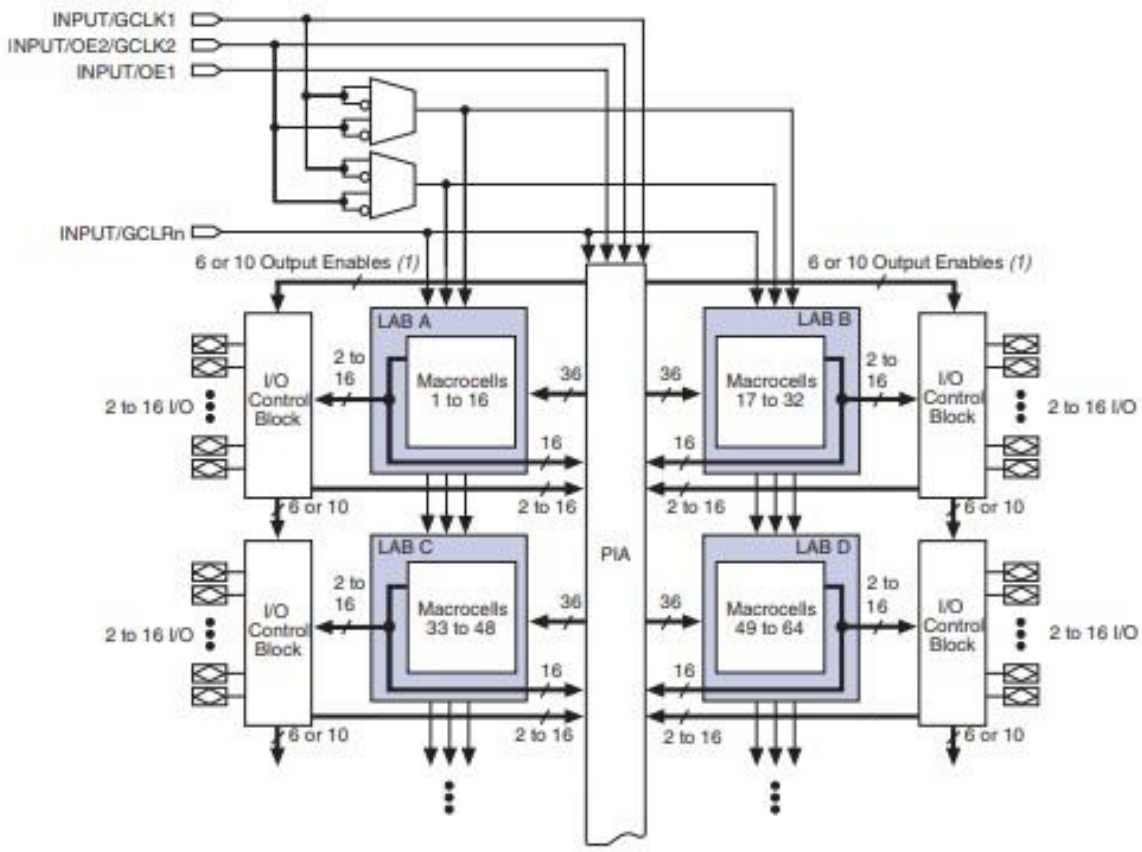


Fig 2. 1 MAX 3000A

2.1.1 Macrocells

MAX 3000A macrocells can be individually configured for either sequential or combinatorial logic operation. Macrocells consist of three functional blocks: logic array, product–term select matrix, and programmable register. The below figure depicts a macrocell:-

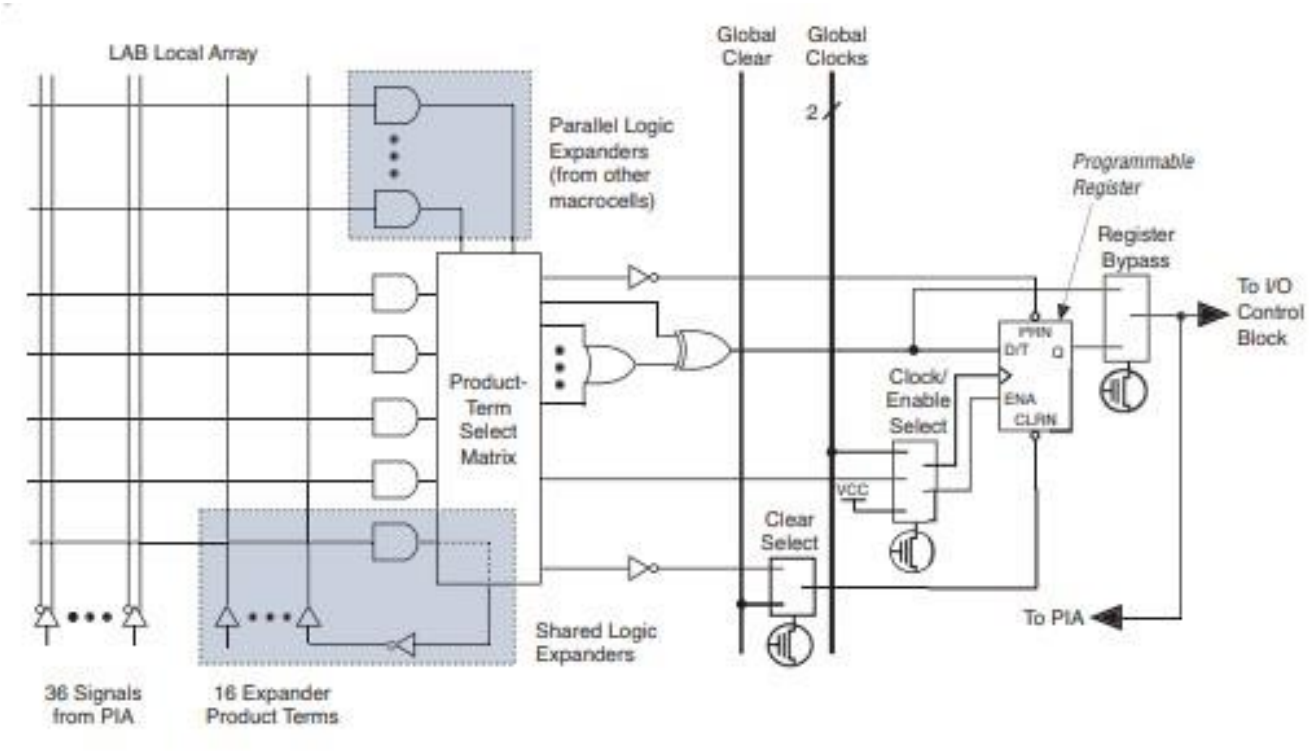


Fig 2.2 Macrocell

In this macrocell AND array is programmable and it is connected to the main switch called Programmable Interconnected Array(PIA). So the input to the AND array is obtained from the PIA. For one macrocell 36 input lines are available. The configurable AND gate will create the product terms and fed to the OR gate. Here AND array is not permanently connected to OR array. A product term select matrix is allocated between the AND array and OR array. These select matrix will operate according to the requirement of the user. If we are realizing a combinational circuit, there is no need of using register(flip-flop). Here there is provision for the obtaining output directly by bypassing the register. But if we are realizing a sequential circuit, the requirement of a register is sufficient. If the realizing logic function has too many product terms, the output from a macrocell can be act as input to the another macrocell.

2.1.2 Programmable Interface Array

It is a global bus that finds a path for a signal source to the any destination on the device. All MAX 3000A dedicated inputs, I/O pins, and macrocell outputs feed the PIA, which makes the signals available throughout the entire device. Only the signals required by each LAB are actually routed from the PIA into the LAB.

2.1.3 Input/ Output Blocks

The I/O control block allows each I/O pin to be individually configured for input, output, or bidirectional operation. All I/O pins have a tri–state buffer that is individually controlled by one of the global output enable signals or directly connected to ground or VCC. When the tri–state buffer control is connected to ground, the output is tri-stated (high impedance), and the I/O pin can be used as a dedicated input. When the tri–state buffer control is connected to VCC, the output is enabled.

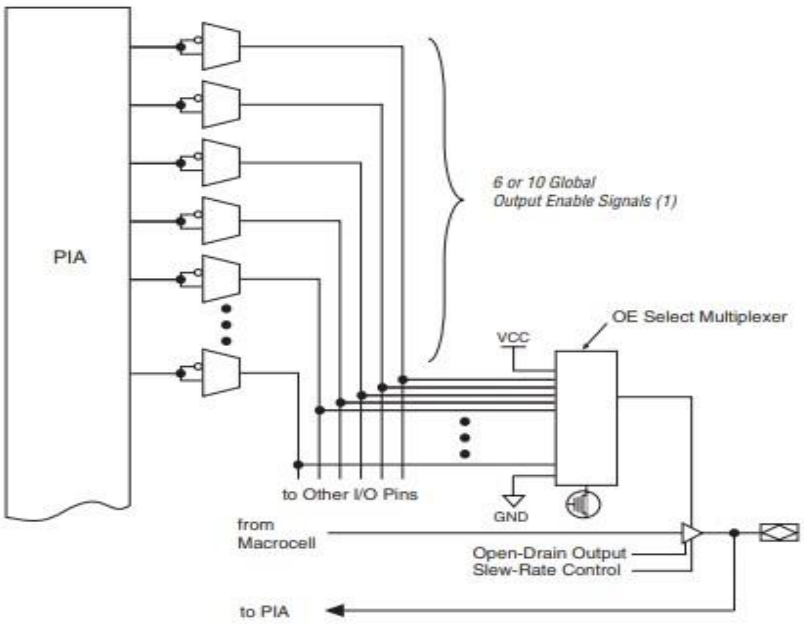


Fig 2.3 I/O Block

Chapter 3

HDL Language

3.1 Introduction

Hardware Description Language (HDL), is a software programming language used to model a piece of hardware. Small digital circuits can be easily drawn and implemented. If the circuit contains many number of logic gates, it is difficult to draw and execute the circuit, so we found out a language to code the function.

3.2 VHDL

VHDL stands for very high-speed integrated circuit hardware description language. It is a programming language used to model a digital system by dataflow, behavioral and structural style of modeling.

3.2.1 Design Units

The following terms constitutes design unit of VHDL:

a) Entity

It is external view of a model. It specifies the input, output and the corresponding data type of the respected input and output.

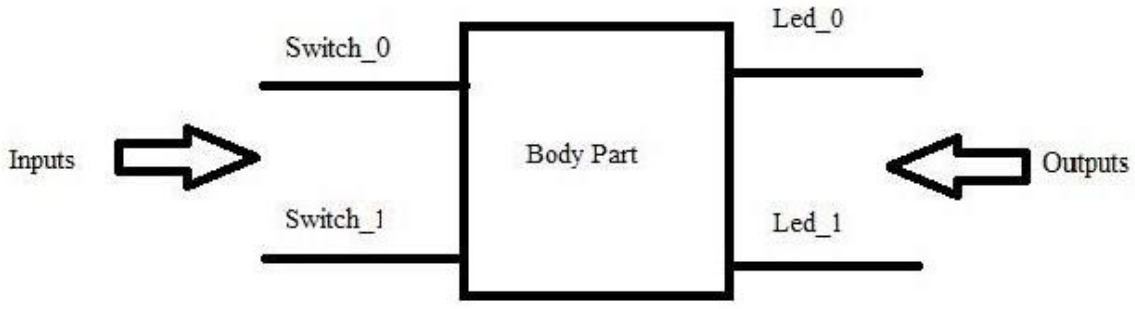


Fig 3.1 Entity

In the above figure Switch_0, Switch_1 act as input for a particular logical circuit. Similarly Led_0,

Led_1 act as output. So we have to declare the respected variables according the required datatype.

The format for the declaration is as follows:

Entity <Name of the entity> is Port (variable 1: input/output datatype; variable

2 : input/output datatype); End entity<entity name> so according to the given format

for the above mentioned problem, entity will be described as Entity Switches is

Port (Switch_0: in STD_LOGIC;

Switch _1: in STD_LOGIC;

Led_0: out STD_LOGIC;

Led_1: out STD_LOGIC);

End Switches;

Here 'in' means input and 'out' means output. STD_LOGIC is datatype which takes 9 different values.

b.) Architecture Body

Architecture is used to define the function of the model. For example, consider a condition that Led_0 is connected to Switch_0 and Led_1 is connected to Switch_1. In order to establish these condition, we have write the program in architecture body. As mentioned above, there are three types of modelling to specify an architecture.

1. Behavioral modelling: It is used for larger designs in which drawing a circuit becomes complex.
2. Structural modelling: Internal conditions are clear and straightforward. It used for smaller designs.
3. Data Flow: It is similar to structural modelling.

The format for writing a program in architecture body in behavioral modelling is as follows: architecture behavioral of < entity name> is begin

<program>

End behavioral;

So for the above problem it will be,

Architecture behavioral of

Switches is begin

Led_0 <= Switch_0;

Led_1<= Switch_1;

End behavioral;

Let's look into an another example for the detailed understanding of VHDL language Consider we have to implement a half adder function, for that we have to take 2 variables as input and 2 variables as output. The following code implements a half adder function in structural modelling and data type modelling.

Let a, b be the input and sum, carry be the output. All the variables take BIT type data.

1. Structural modelling

Entity Half_adder is

Port (a,b :in BIT;

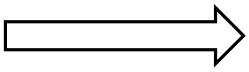
Sum, carry: out BIT); End

Half_adder; architecture

structure_HA of Half_adder is

component XOR1

port (p, q : in BIT;



Before beginning the main body, two components are defined. The first component is XOR and second component is AND

```
r: out BIT);
end component;

component AND1
port( x,y :in BIT;
z :out BIT);
end
component;
begin
X1: XOR1 port map(a,b,sum);
A1:AND1 port map(a,b,carry);
End structure_HA;
```

In structural modelling, there is separate VHDL design for an AND gate and EXOR gate. These two designs are combined together and it will have mapped to global inputs.

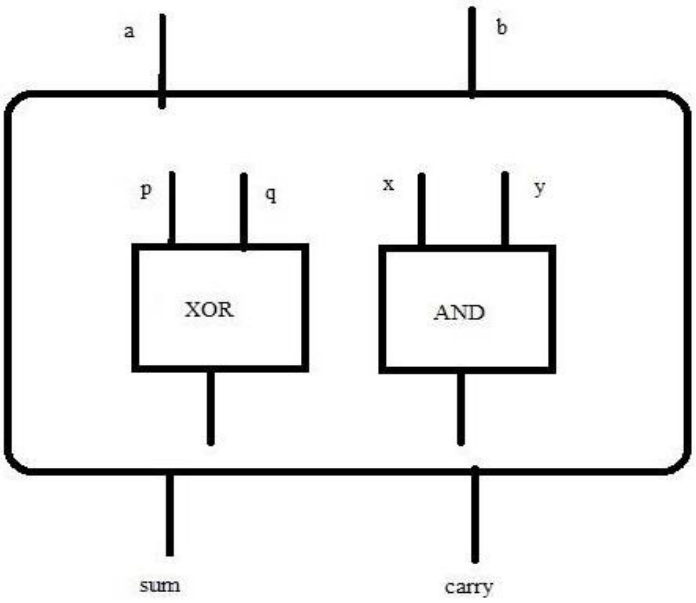


Fig 3.2 Representing half adder in structural modelling

Here variable ‘a’ will be connected to ‘p’ and ‘x’ .similarly variable ‘b’ will be connected to ‘q’ and ‘y’ . The output of XOR and AND are connected to sum and carry respectively. Hence my mapping the half adder function can be implemented.

2. Data Flow modelling

```
Entity Half_adder is
Port (a,b :in BIT;
    Sum, carry: out BIT); End
Half_adder; architecture
dataflow_HA of Half_adder is
begin sum <= a XOR b;
carry <= a AND b;
end dataflow_HA;
```

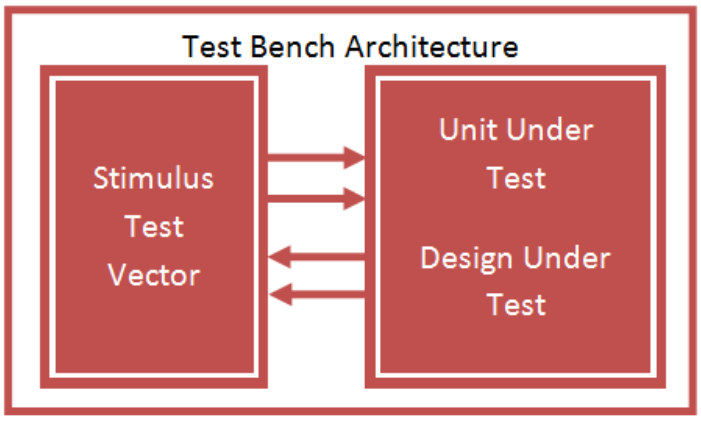
3 . Behavioral Modelling

This modelling is preferred for sequential logic design, where the instructions execute sequentially in the architecture block. Consider an example of VHDL modeling of D flip flop using behavioral modelling style.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff is
port( din: in std_logic;
clk: in std_logic;
rst: in std_logic;
dout: out std_logic);
```

```
end dff;


architecture behavioral of dff is
begin
process(rst,clk,din)
begin
if (rst='1') then
dout<='0';
elsif(rising_edge(clk)) then
dout<= din;
end if;
end process;


end behavioral;
```

VHDL TEST BENCH



Any design written in HDL programing is to be tested by applying different test input vectors. The VHDL code is written to apply the test inputs and verify the functionality of design.

```
library ieee;
use ieee.std_logic_1164.all;
entity dff_tb is
end dff_tb;
architecture behavior of dff_tb is
component dff
    port(
        din : in  std_logic;
        clk : in  std_logic;
        rst : in  std_logic;
        dout : out  std_logic
       );
    end component;

   signal din : std_logic := '0';
   signal clk : std_logic := '0';
   signal rst : std_logic := '1';
   signal dout : std_logic;

constant clk_period : time := 10 ns;
begin
```

```
uut: dff port map (
        din => din,
        clk => clk,
        rst => rst,
        dout => dout
      );

 clk_process :process
 begin
 clk <= '0';
 wait for clk_period/2;
 clk <= '1';
 wait for clk_period/2;
 end process;

 stim_proc: process
 begin

 rst <= '1';
 wait for 50 ns;

 rst <= '0';
 din <= '0';
 wait for 50 ns;
```

```
  rst <= '0';
  din <= '1';
  wait;

  end process;

END;
```

Chapter 4 Quartus Software

1. Introduction

This chapter is intended to give a general overview of a typical CAD flow for designing circuits that are implemented by using CPLD/FPGA devices, and shows how this flow is realized in the Quartus II software. The tutorial of the entire process is divided in two parts and is illustrated by giving stepby-step instructions for using the Quartus II software to implement a very simple circuit in a CPLD device on Helium board: (*a*) simulation using MODELSIM (*b*) generating *.svf* file and configuring the device using urJTAG. Make sure that you have QuartusII, Modelsim (installed alongwith Quartus II), and urJTAG installed on your machine. While it is preferred to use ubuntu, both the software work fine on MICROSOFT WINDOWS too. The screen shots in the tutorial were obtained using the Quartus II version 13.0. Note that the versions above 13.0 do not support MAX3000 CPLD.

A typical FPGA CAD flow is illustrated in Figure 4.1.

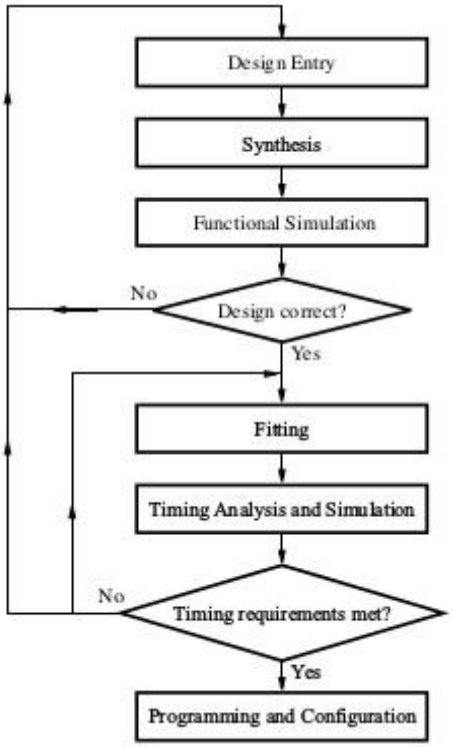


Figure 4.1 A typical design flow

1. Design Entry: The desired circuit is specified either by means of a schematic diagram, or by using a hardware description language, such as VHDL or Verilog. We shall use VHDL description for design entry.

2. Synthesis: The entered design is synthesized into a circuit that consists of the logic elements (LEs) provided in the /FPGA chip.

3. Functional Simulation: The synthesized circuit is tested to verify its functional correctness; this simulation does not take into account any timing issues.

4. Fitting (Place and Route): The CAD Fitter tool determines the placement of the LEs(logic elements) defined in the netlist into the LEs in an actual CPLD/FPGA chip; it also chooses routing wires in the chip to make the required connections between specific LEs.

5. Timing Analysis: Propagation delays along the various paths in the fitted circuit are analyzed to provide an indication of the expected performance of the circuit

6. Timing Simulation: The fitted circuit is tested to verify both its functional correctness and timing.

7. Programming and Configuration: The designed circuit is implemented in a physical FPGA chip by programming the configuration switches that configure the LEs and establish the required wiring connections.

2. Getting started

2.1 Creating a project

To begin a new logic circuit design which is referred as *project*, the first step is to create a folder in appropriate directory to hold its files. For this tutorial, example design is that of a Two Bit Adder described in VHDL and saved as TwoBitAdder.vhd with corresponding testbench saved as adder tb.vhdl.

You may follow the following steps with me.

1. Open QuartusII software.

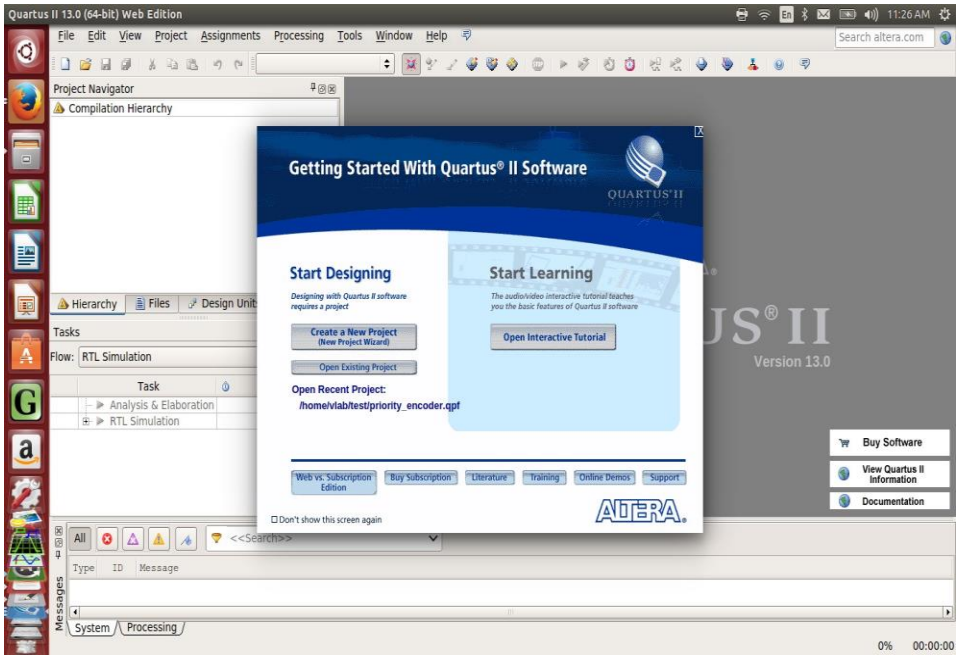


Figure 4.2 Open project window

2. Click New Project Wizard which pops a dialog window that tells you about various project settings. Click ”Next” to go to the 1/5 pages of the ”New Project Wizard”.

3. Fill in the fields appropriately. I enter the working directory to be Tutorial VHDL. The project must have a name, same as the top-level design entity that will be included in the project. I enter TwoBitAdder. Refer Fig. 4.3. Now Press “Next”.

4. You will go to the next page which will prompt you to add the necessary files to the project. Ichoose TwoBitAdder.vhd file to the project as indicated in Fig. 4.4 and press ”Next”.

5. This page tells Quartus which device you are using for implementation of the design. Though itis not absolutely necessary to fill up the details now (if you are interested only in simulation), it is advised to fill up the details of the device to be used. Helium uses MAX3000A family, PLCC

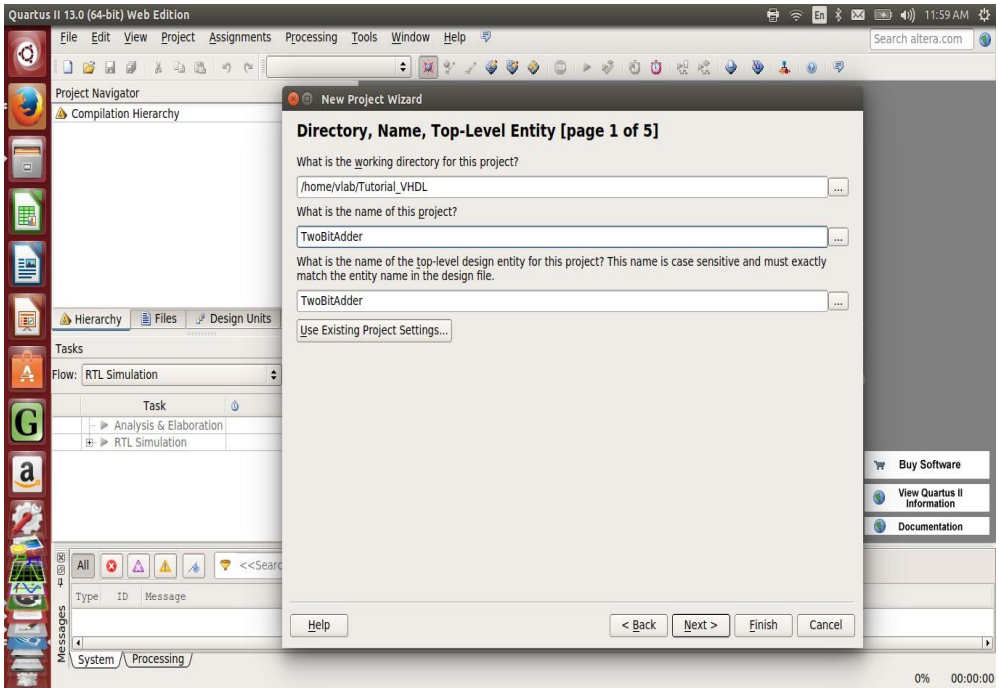


Figure 4.3 Page 1 of 5

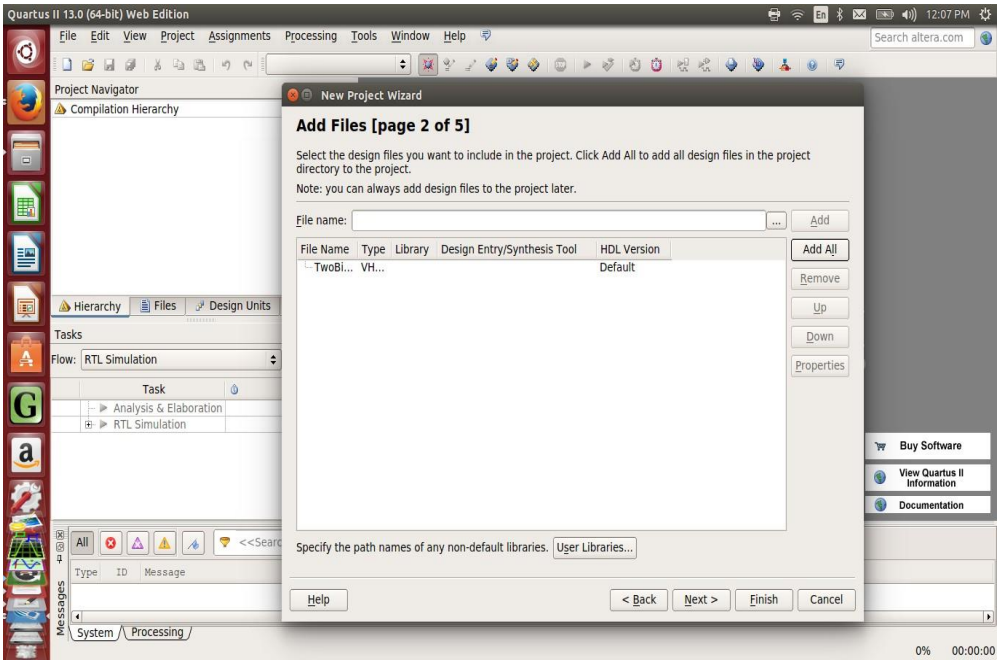


Figure 4.4 Page 2 of 5

package, 44pins, speed grade of 10 and has 64 macrocells. Fill up all these details and say “Next” an shown in Fig. 4.5.

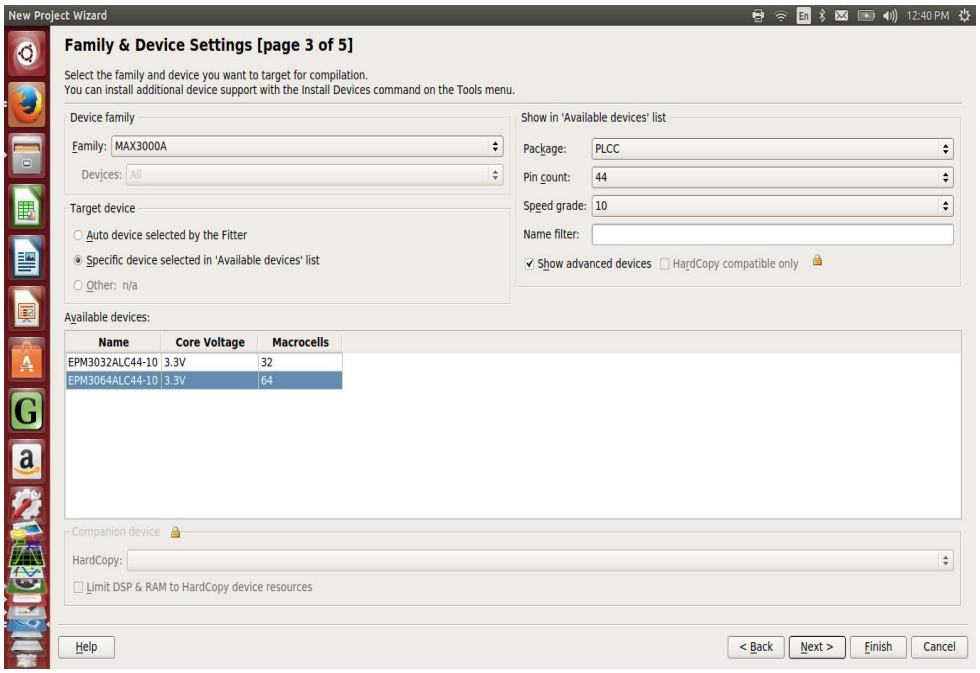


Figure 4.5 Page 3 of 5

1. The next page is for EDA settings. Make sure that ModelSim-Altera appears in the Tool Name column for Simulation. Press “Next”.

2. Summary page pops up that is shown in Fig. 4.6. Click “Finish” to complete the process of creating project.

2.2 Compiling the logic design

Once the project is created, we are ready to compile our VHDL file. The file *TwoBitAdder.vhd* in this example is processed by several Quartus II tools (compiler) that analyze the code, synthesize the circuit, and generate an implementation of it for the target device.

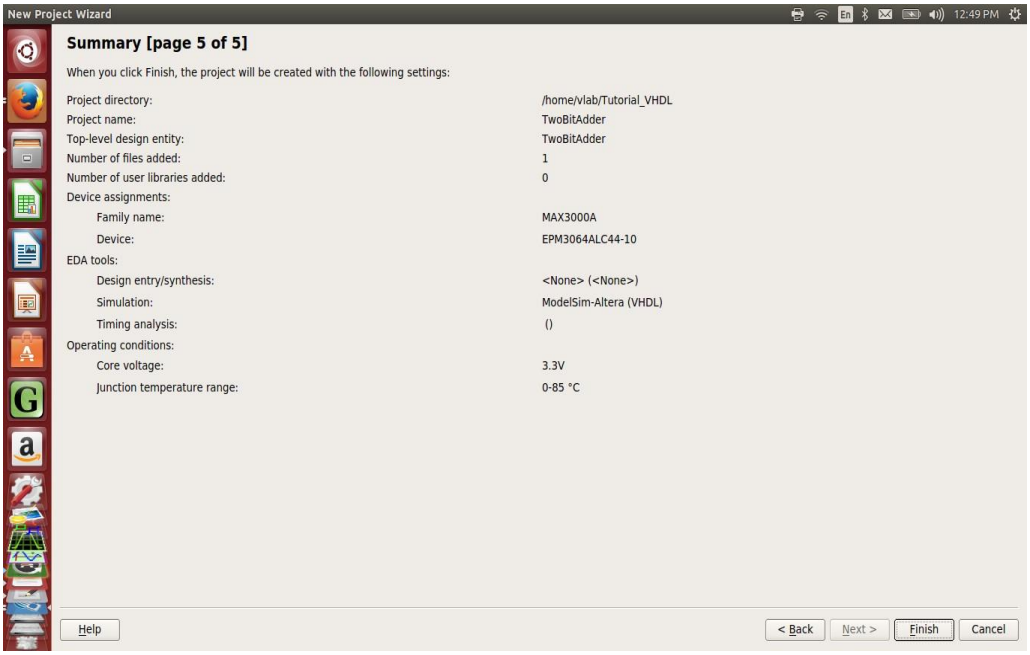


Fig. 4.6 Page 5 of 5

Run the Compiler by selecting Processing → Start Compilation, or by clicking I icon on the toolbar. Successful (or unsuccessful) compilation is indicated in a pop-up box, click OK. In the message window, at the bottom of the figure, various messages are displayed. In case of errors, appropriate messages are shown. The compilation report shown in Fig. 4.7 includes various sections as included in the Table of contents. The "Flow Summary" summarizes the status of compilation, resource utilization by the top entity, number of pins used, etc.

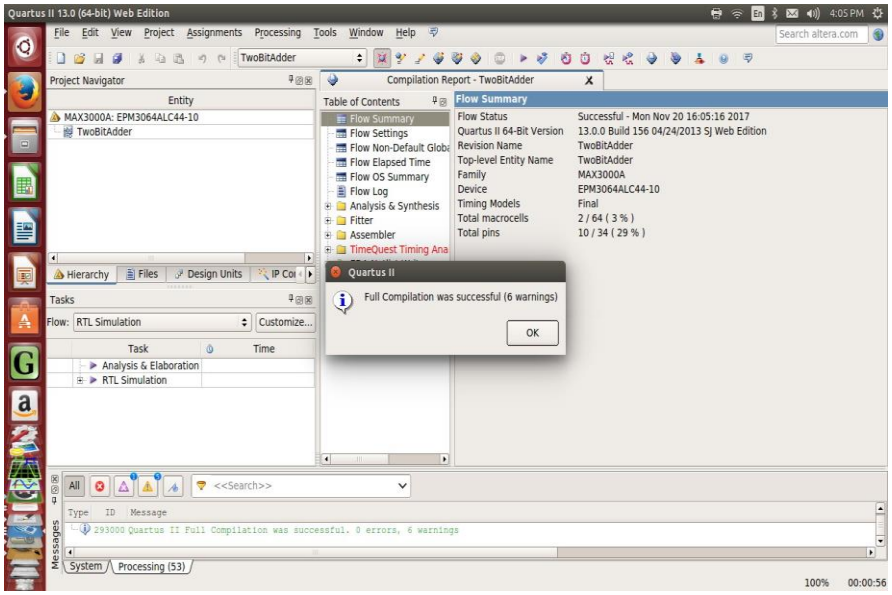


Fig. 4.7 Compilation report

After compilation, you can view the logic network described in VHDL by selecting Tools → Netlist Viewer → RTL Viewer as shown in Fig. 4.9.

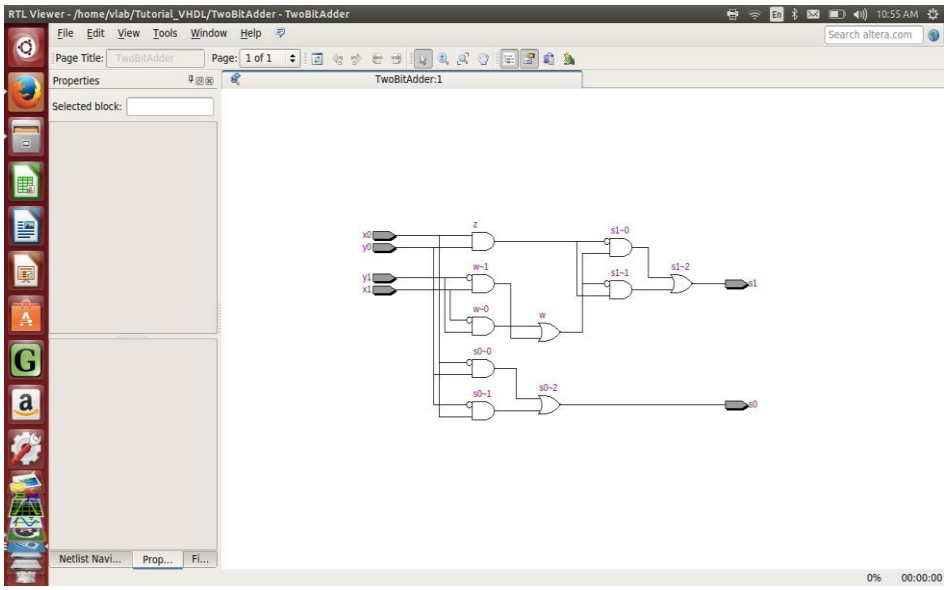


Fig. 4.8 RTL Viewer

2.3 Simulation using ModelSim-Altera

1. Let's first add (or provide the path if saved elsewhere while adding file) the testbench file (or provide the path if saved elsewhere while adding file) "Testbench.vhd" in our example to the project by selecting Project → "Add/Remove files" in the Files menu.

2. Once the testbench is added, goto "EDA Tool Settings"→ Simulation from the menu bar on the left.

3. It will open the window shown in Fig below.

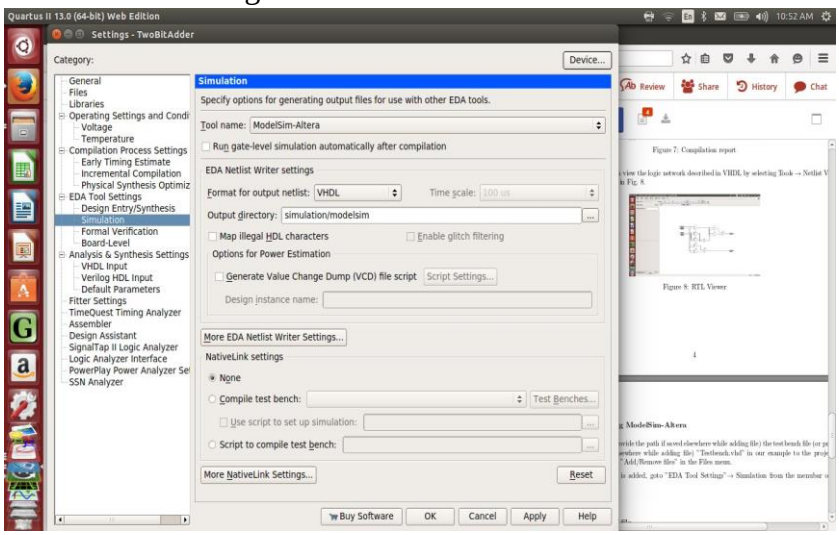
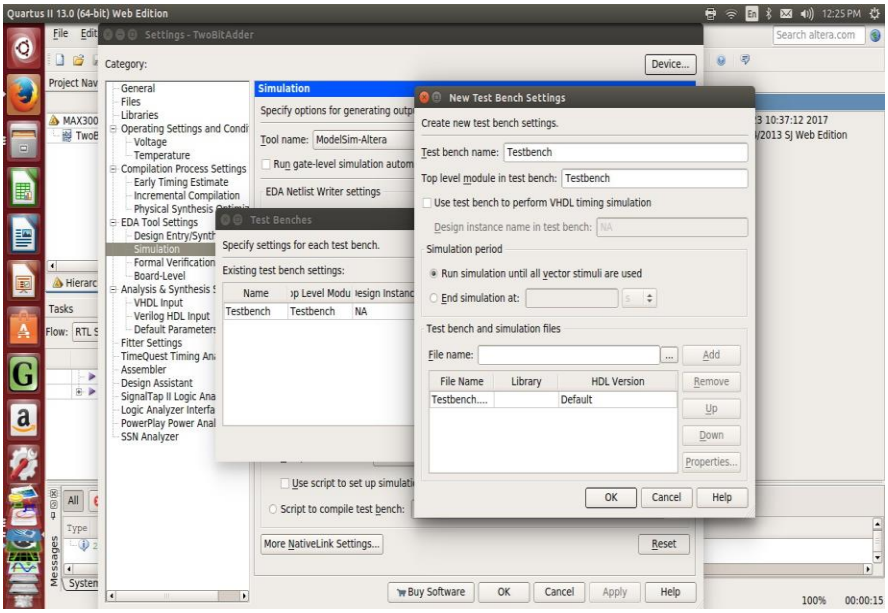


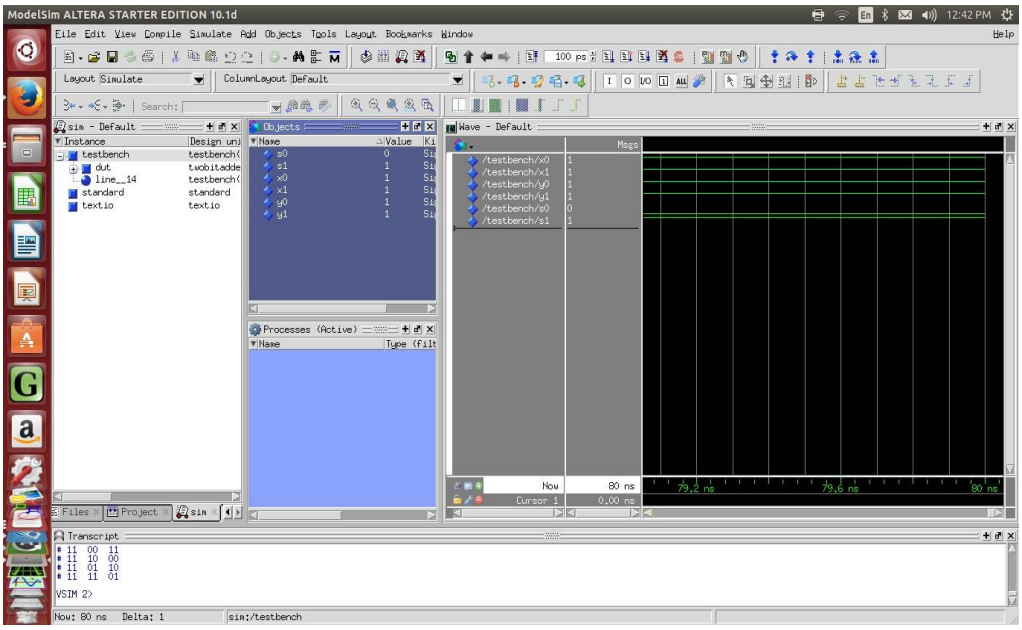
Fig. 4.9 Adding and compiling testbench-1

4. Click “compile test bench” from “NativeLink Settings” and thereafter “Testbenches”. This will pop up a window to specify each testbench settings. Click “New” which will open one more pop up window for “New Test Bench Settings”. Fill in the details like Testbench Name, Top Level Entity, and “add” Testbench. Refer Fig below.



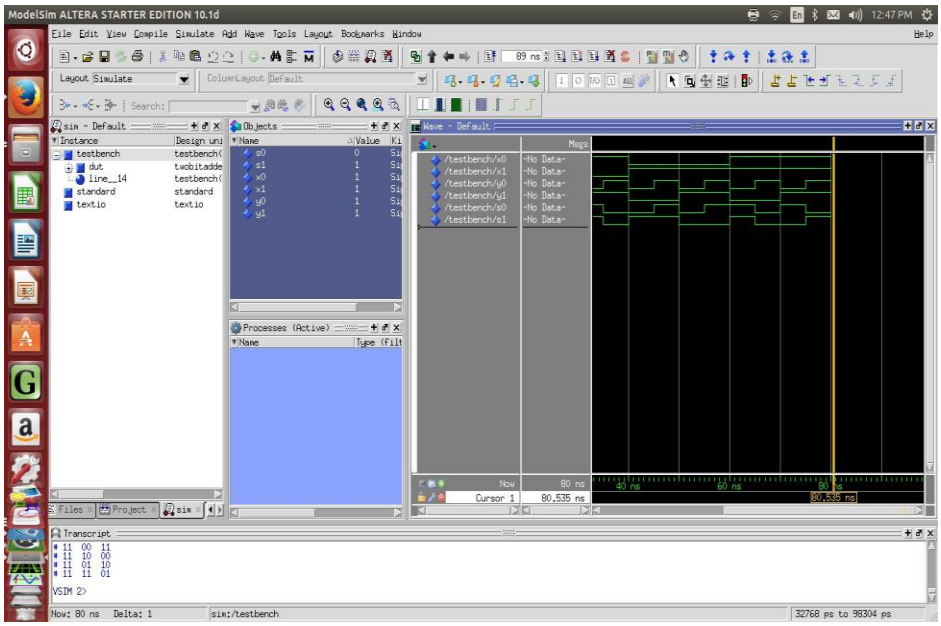
5. Recompile the project for “successful full compilation”

6. Now go to “Tools” → “Run Simulation Tool”→ ”Run RTL Simulation”. This will open “Waveform window as shown in Fig below.



7. Set the time to about 90ns as the simulation runs for 80ns and double click on the waveform area to see the cursor. Use zoom buttons to view the waveforms appropriately.

8. Verify them for each combination. You can scroll down to check all combinations of inputs and their corresponding outputs.



While using testbench for simulation is the most elegant way for testing the correctness of the design, for very simple circuits with very few inputs, one can “force” the inputs to observe the output without writing the testbench.

2.4 Generating .svf file

Once the design is compiled, you can choose what pins to assign as input/output from Pin Planner under the Assignments tab in the Menu bar.A new window opens up, showing you the pin diagram of the device selected (in this case, MAX 3064), and below, the signal lines that need to be pin- assigned. The pins can be assigned by double clicking the location field. Follow the information in Table 2 to do the pin assignment. Warning: Sometimes the pin planner window may show the signals TDI, TDO, TMS, TCK. DO NOT assign these to any pins.

After pin assignment, recompile the design.

Now go to Tools→ Programmer.

A programmer window will open. You should see the project output file TwoBitAdder.pof in this window. If not, you may need to create a fresh project from the start. 4. In the programmer window, go to File Create (JAM, SVF...). A new window will open. Select the programming file type as Serial Vector Format (SVF). Browse to the directory where you wish to store the .svf file. Preferably, save it directly in C: drive.

The filename need not be the same as the project name. Let us give the name adder2.svf. The programming file is now ready.

2.4 Configuring the Helium board using urJTAG

1. Open “Terminal”and type jtag to go into JTAG shell.

2. Now type “cable ft2232 vid=0x0403 pid=0x6010” to connect to libftdi driver.

3. Type “detect” and it will display details of the target device like the Device ID, manufacturer, the devices connected in JTAG chain(part(0) as EPM3064, the only device connected, IR and Chain length. Manufacturer,

4. Now type svf¡¡ full path to .svf file¿¿. The svf file gets loaded into the CPLD board and the output of the VHDL program may be observed on the board.