LAB 1
Introduction to VHDL and Altera Quartus with the design of Full Adder and reproduce the output of FPGA board

Objectives

- *Comprehend* the concept of hardware descriptive languages in general and VHDL in particular.
- *Illustrate* software tool Quartus with the design of full adder
- *Produce* the result using hardware FPGA by loading the bit file.

Familiarize yourself with VHDL and Quartus:

Introduction to Quartus:

Altera Quartus II is design software produced by Altera. Quartus II enables analysis and synthesis of HDL designs, which enables the developer to compile their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Quartus includes an implementation of VHDL and Verilog for hardware description, visual editing of logic circuits, and vector waveform simulation.

Introduction to VHDL

VHDL is a hardware description language. It described the behavior of an electronic circuit or system, from which the physical circuit or system can then be attained (implemented). VHDL stands for VHSIC Hardware Description Language. VHSIC is itself an abbreviation for Very High-Speed Integrated Circuits, an initiative funded by US DOD in the 1980s that led to the creation of VHDL. VHDL is intended for circuit synthesis as well as circuit simulation. However, though VHDL is fully simulated, not all constructs are synthesizable.

A fundamental motivation to use VHDL (or its competitor, Verilog) is that VHDL is a standard, technology/vendor independent language, and is therefore portable and reusable. The two main immediate applications of VHDL are in the field of Programmable Logic Devices (including CPLDs and FPGAs) and ASICs. A final note regarding VHDL is that contrary to regular computer programs which are sequential, its statements are inherently concurrent (parallel). In VHDL, only statements placed inside *a Process, Function or Procedure* are executed sequentially.

As mentioned above, one of the major utilities of VHDL is that it allows the synthesis of a circuit or system in a programmable device (PLD or FPGA) or in an ASIC. The steps followed during such a project are summarized in figure 1. 1

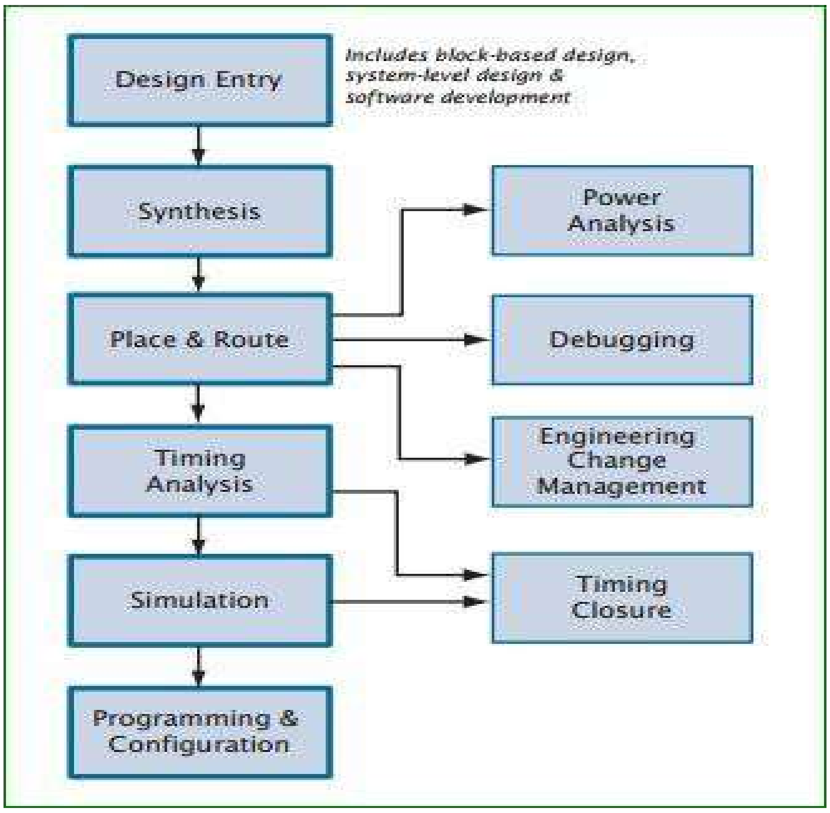

Design Entry
Includes block-based design, system-level design & software development
Synthesis
Place & Route
Timing Analysis
Simulation
Programming & Configuration
Power Analysis
Debugging
Engineering Change Management
Timing Closure

Figure 1.1 Quartus II Design Flow

Fundamental VHDL Units

Library declaration: Contains a list of all libraries to be used in the design. For example: ieee, std, work etc. Their declarations are as follows

```
Library ieee;

USE ieee.std_logic_1164.all; USE ieee.std_logic_arith.all;
```

Entity:

VHDL files are basically divided into two parts, the entity and the architecture. The entity is basically where the circuits (in & out) ports are defined. There is a multitude of I/O ports available, but this lab will only deal with the two most usual ones, the INput and OUTput ports. (Other types of ports are for example the INOUT and BUFFER ports.) The entity of the circuit in figure 1.2 should look something like below. Please notice that comments in the code are made with a double-dash (--).

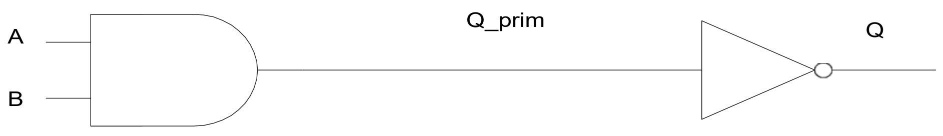


Figure 1.2 NAND gate using AND & NOT gate

```
ENTITY nandgate IS PORT(
A: IN BIT;
B: IN BIT;
Q: OUT BIT -- Note! No ‘;’ here!
);
END nandgate;
```

Now that we have defined the I/O interface to the rest of the world for the NAND gate, we should move on to the architecture of the circuit.

Architecture:

The entity told us nothing about how the circuit was implemented, this is taken care of by the architecture part of the VHDL code. The architecture of the NAND gate matching the entity above could then be written as something like this...

```
ARCHITECTURE dataflow OF nandgate IS
    SIGNAL Q_prim: BIT;
BEGIN
    Q_prim <= A AND B;     -- The AND-operation.
    Q <= NOT Q_prim;       -- The inverter.
END dataflow;
```

Behavior modeling, Processes

We can write a behavior architecture body of an entity which describes the function in an abstract way. Such an architecture body includes only process statements.

Processes are used:

- For describing component behavior when they cannot be simply modeled as delay elements.
- To model systems at high levels of abstraction.

Process contains:

Conventional programming language constructs. A process is a sequentially executed block of code, which contains.

- Evaluating expressions.
- Conditional execution.
- Repeated execution.
- Subprogram calls.
- Variable assignments, e.g., x := y , which, unlike signal assignment, take effect immediately.
- if-then-else and loop statements to control flow, and
- Signal assignments to external signals.

Notes:

1. Signal assignment statements specify the new value and the time at which the signal is to acquire this value. The textual order of the concurrent signal assignment statements (CSAs) do NOT effect the results.
2. Processes contain sensitivity lists in which signals are listed, which determine when the process executes.
3. In reality, CSAs are also processes without the process, begin and end keywords.

Structure modeling

Structural model: A description of a system in terms of the interconnection of its components, rather than a description of what each component does. A structural model does NOT describe how output events are computed in response to input events.

A VHDL structural description must possess:

- The ability to define the list of components.
- The definition of a set of signals to be used to interconnect them.
- The ability to uniquely label (distinguish between) multiple copies of the same component.

Design and Implementation of Four-bit adder:

In this lab you will design, test, and simulate a basic logic circuit using the Quartus II development software. The circuit you will create is a four-bit adder using both behavioral and structural VHDL coding. Below is a schematic diagram of the complete circuit.

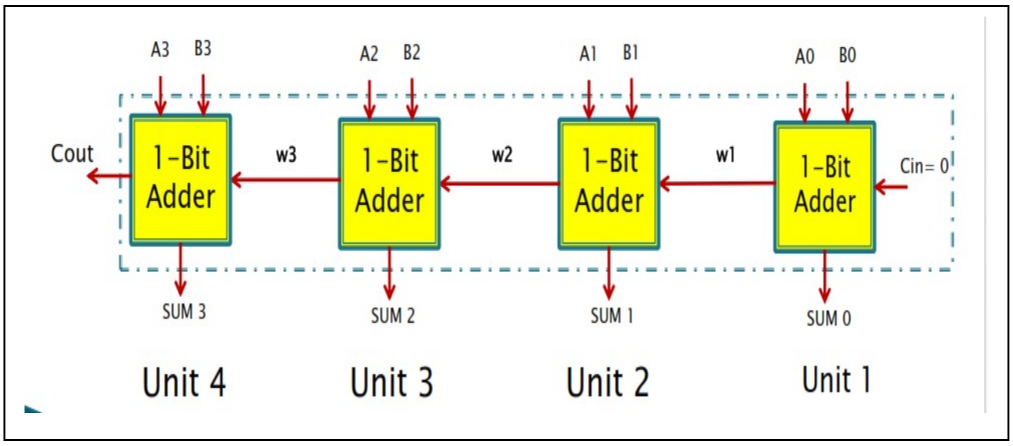


Figure 1.3 4- bit adder

Below is a detailed diagram of the full adder circuit used. It consists of three two-input AND gates, two XOR gates, and one three-input OR gate. The individual gates will be coded separately using behavioral style VHDL, and then stitched together using structural style VHDL to create the full adder.

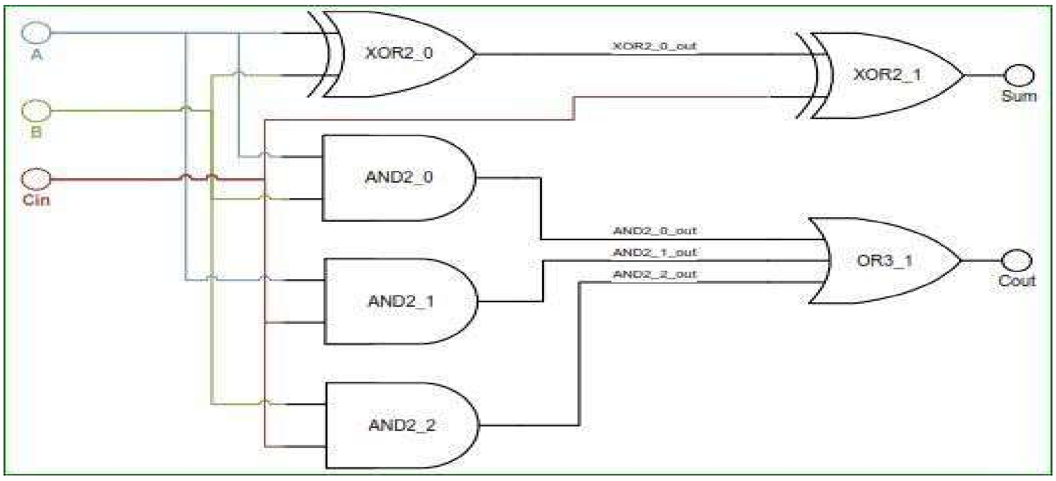


Figure 1.4 4- bit adder gate-level

Task 1: Create a New Project

1. **Start the Quartus II software.**

   From the Windows Start Menu, select:

```
All Programs → Other Apps →Altera → Quartus II 13.1 → Quartus II 13.1 (32-
Bit)
```

2. **Start the New Project Wizard.**

   If the opening splash screen is displayed, select: **Create a New Project (New Project Wizard)**, otherwise from the Quartus II Menu Bar select: **File → New Project Wizard**.

3. **Select the Working Directory and Project Name.**

Working Directory	H:\Altera_Training\Lab1
Project Name	Lab1
Top-Level Design Entity	Lab1

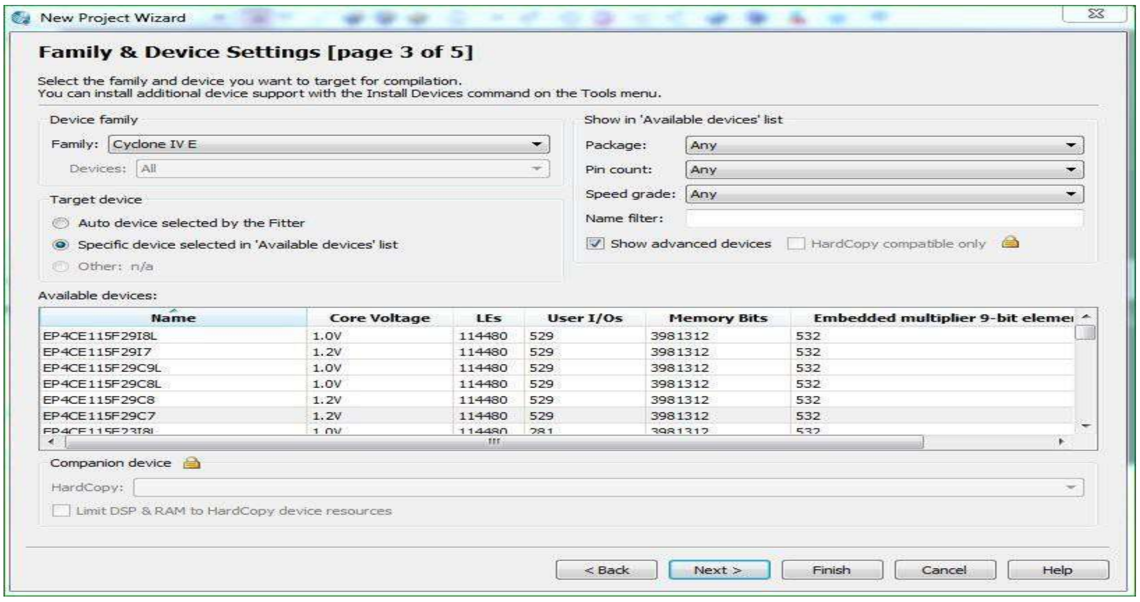
   Click **Next** to advance to page 2 of the New Project Wizard.

   *Note: A window may pop up stating that the chosen working directory does not exist. Click **Yes** to create it.*

4. **Click Next again as we will not be adding any preexisting design files at this time.**

5. **Select the family and Device Settings.**

- From the pull-down menu labeled **Family**, select **Cyclone IV**.
- In the list of available devices, select **EP4CE115F29C7N**. Click **Next.**



New Project Wizard

Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.
You can install additional device support with the Install Devices command on the Tools menu.

Device family
Family: Cyclone IV E
Devices: All

Target device
Auto device selected by the Fitter
[✓ Specific device selected in 'Available devices' list]
Other: n/a

Show in 'Available devices' list
Package: Any
Pin count: Any
Speed grade: Any
Name filter:
[✓ Show advanced devices]
HardCopy compatible only

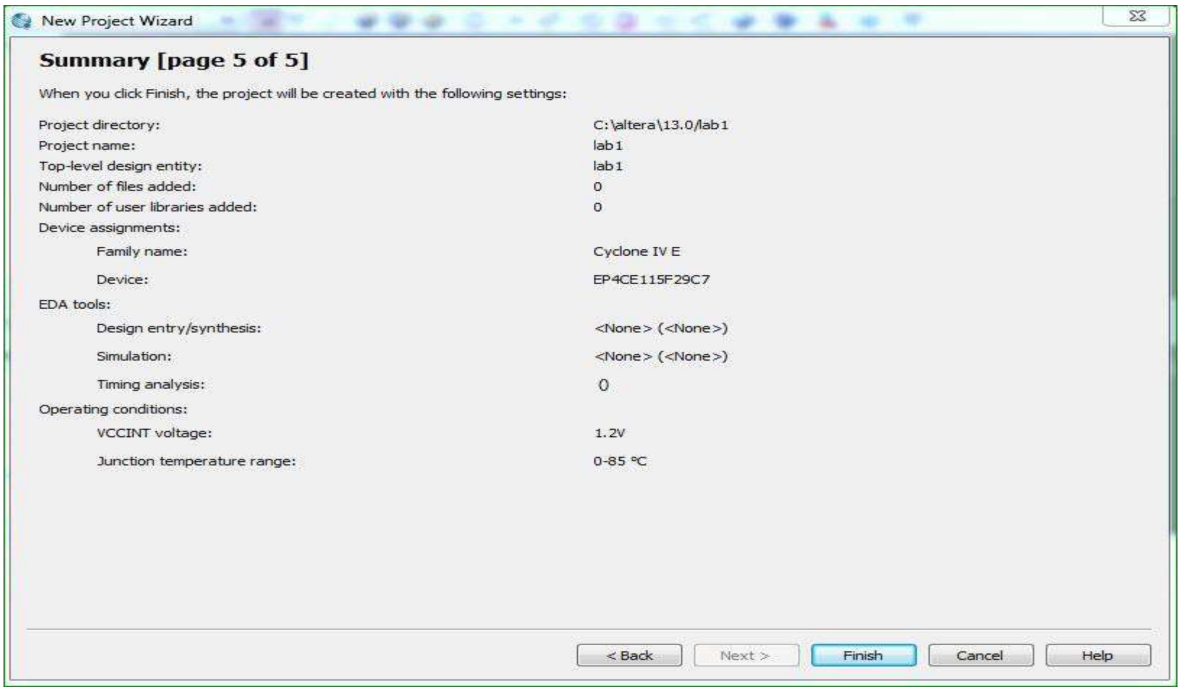
Available devices:

Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit eleme
EP4CE115F29I8L	1.0V	114480	529	3981312	532
EP4CE115F29I7	1.2V	114480	529	3981312	532
EP4CE115F29C9L	1.0V	114480	529	3981312	532
EP4CE115F29C8L	1.0V	114480	529	3981312	532
EP4CE115F29C8	1.2V	114480	529	3981312	532
EP4CE115F29C7	1.2V	114480	529	3981312	532
EP4CE115F23I8L	1.0V	114480	281	3981312	532

Companion device
HardCopy:
Limit DSP & RAM to HardCopy device resources

6. **Click Next again as we will not be using any third-party EDA tools**

7. **Click Finish to complete the New Project Wizard**


New Project Wizard

Summary [page 5 of 5]

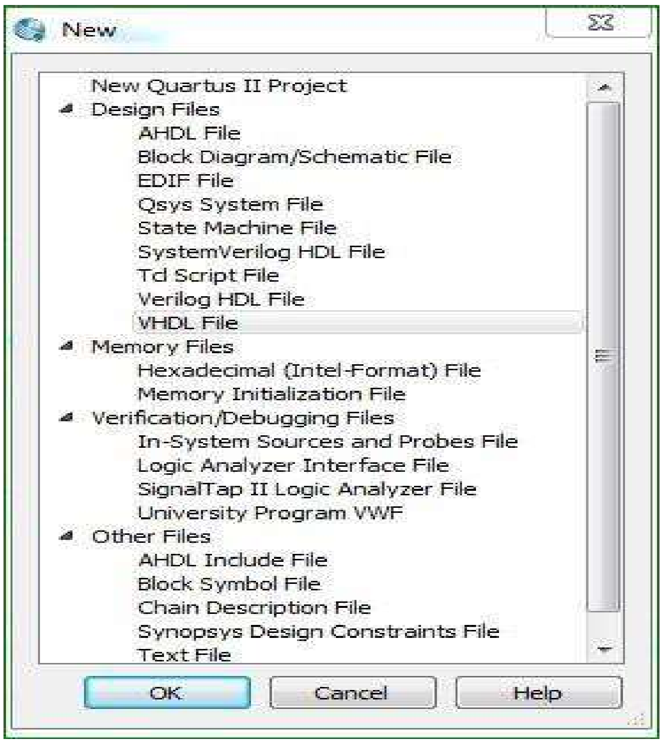
When you click Finish, the project will be created with the following settings:

Project directory:	C:\altera\13.0/lab1
Project name:	lab1
Top-level design entity:	lab1
Number of files added:	0
Number of user libraries added:	0
Device assignments:	
Family name:	Cyclone IV E
Device:	EP4CE115F29C7
EDA tools:	
Design entry/synthesis:	<None> (<None>)
Simulation:	<None> (<None>)
Timing analysis:	0
Operating conditions:	
VCCINT voltage:	1.2V
Junction temperature range:	0-85 ℃

Task 2: Create, Add, and Compile Design Files

1. **Create a new Design File.**

- Select: **File** → **New** from the Menu Bar.
- Select: **VHDL File** from the **Design Files** list and click **OK**.



2. Copy and paste the following code into your new VHDL file, then save it by selecting **File →Save**. Name the file **bitadder** and click **Save** in the **Save As** dialog box.

```
library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
entity bitadder is port (
    A: in std_logic;
    B: in std_logic;
    C: in std_logic;
    sum: out std_logic;
    cout : out std_logic);
end bitadder;
architecture behavorial of bitadder is
begin
    sum<= A xor B xor C;
    cout<= (A and B) or (A and C) or (B and c);
end behavorial;
```

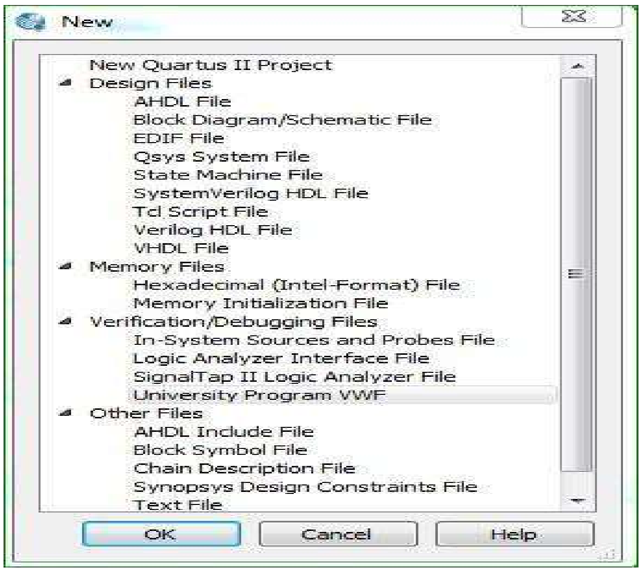
3. Create another new VHDL file following the directions in step 1 of task 2.
4. Copy and paste the following code into your new VHDL file, then save it by selecting **File → Save**. Name the file **adder4** and click **Save** in the **Save As** dialog box.

```
library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
entity adder4a is port (
      A: in std_logic_vector(3 downto 0);
      B: in std_logic_vector(3 downto 0);
      C: in std_logic;
      sum: out std_logic_vector(3 downto 0);
      cout : out std_logic);
end adder4a;

architecture behavorial of adder4a is
      component bitadder is port (
            A: in std_logic;
            B: in std_logic;
            C:  in std_logic;
            sum: out std_logic;
            cout : out std_logic);
      end component;

signal w1, w2, w3: std_logic;
begin
unit1: bitadder port map (A(0), B(0),'0',sum(0),w1);
unit2: bitadder port map  (A(1), B(1), w1, sum(1), w2);
unit3: bitadder port map (A(2), B(2), w2, sum(2), w3);
unit4: bitadder port map (A(3), B(3), w3, sum(3), cout);
end behavorial;
```

Task 3: Simulate Design using Quartus II

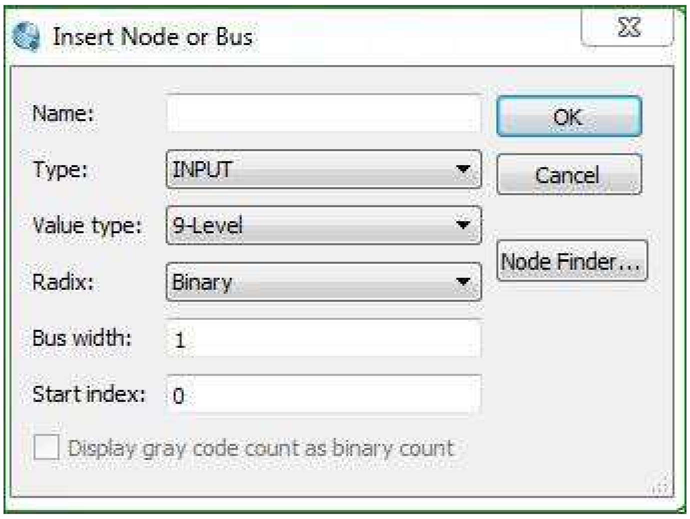
1. **Create a Vector Waveform File (vwf)**

   - Click the **New File** icon on the Menu Bar
   - Select **Vector Waveform File** under the **Verification/Debugging Files** heading.



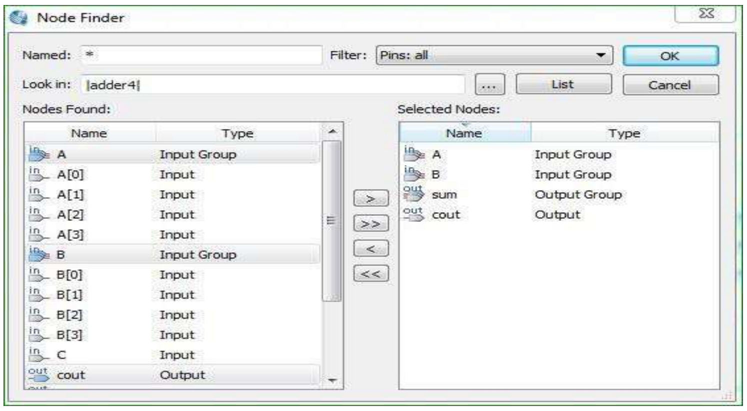
2. **Add Signals to the Waveform**

Select **Edit** → **Insert** → **Insert Node or Bus…** to open the **Insert Node or Bus** window.


Name:
Type: INPUT
Value type: 9-Level
Radix: Binary
Bus width: 1
Start index: 0

3. Click the **Node Finder…** button in the **Insert Node or Bus** window to open the **Node Finder** window. From the **Filter** drop-down menu, select **Pins: all** and click the **List** button. This will displayall the inputs and outputs to the circuit.

   Highlight all the **Input Groups A, B**, and **Output Group** Sum in the **Node Finder** window and click the right arrow to select them. Then click **OK** to close the **Node Finder** window.



Node Finder

Named: * Filter: Pins: all

Look in: |adder4|

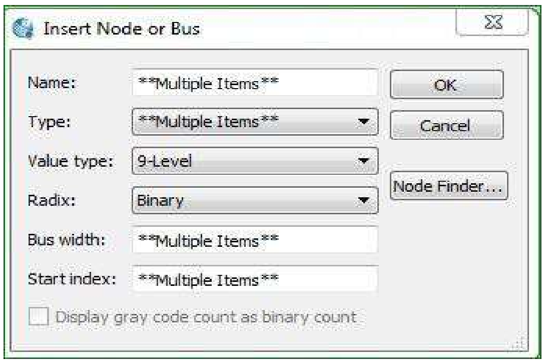
Nodes Found:

Name	Type
A	Input Group
A[0]	Input
A[1]	Input
A[2]	Input
A[3]	Input
B	Input Group
B[0]	Input
B[1]	Input
B[2]	Input
B[3]	Input
C	Input
cout	Output

Selected Nodes:

Name	Type
A	Input Group
B	Input Group
sum	Output Group
cout	Output

4. In the **Insert Node or Bus** window, change **Radix** to **Binary**. This will make it easier for us to enter values and interpret the results.



Insert Node or Bus

Name: **Multiple Items**

Type: **Multiple Items**

Value type: 9-Level

Radix: Binary

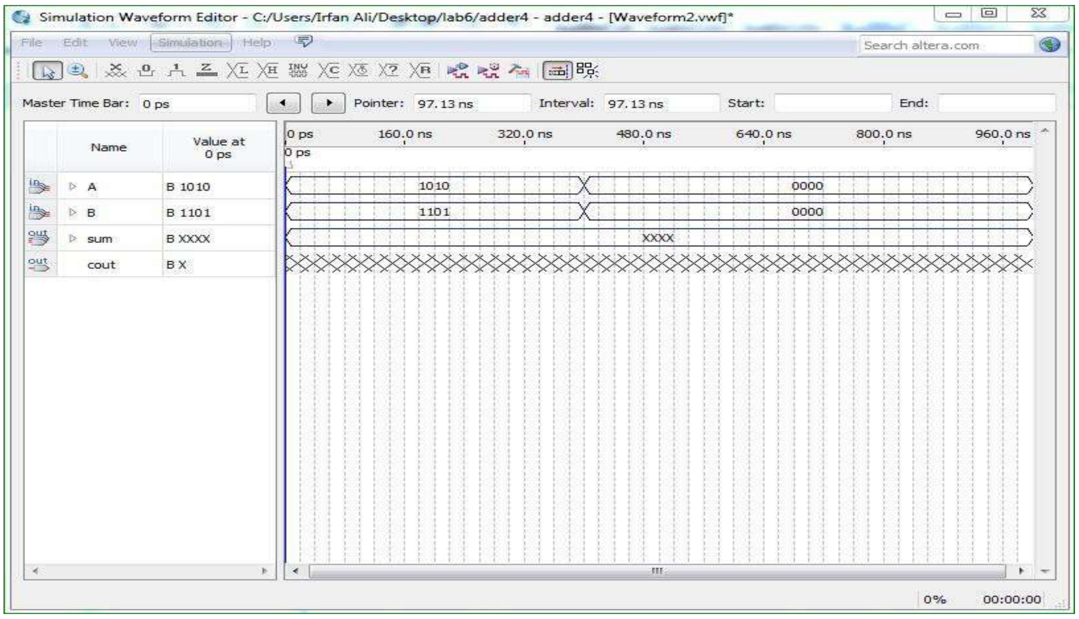
Bus width: **Multiple Items**

Start index: **Multiple Items**

Display gray code count as binary count

5. Input waveforms can be drawn in different ways. The most straightforward way is to indicate a specific time range and specify the value of a signal. To illustrate this approach, click the mouse on the A waveform near the 0-ns point and then drag the mouse to the 400-ns point. The selected time interval will be highlighted in blue. Press **Ctrl + Alt + B** to open the **Arbitrary Value** window to enter the values manually.

We will use this approach to create the waveform for B, which should change from 0 to 400 ns.



Simulation Waveform Editor - C:/Users/Irfan Ali/Desktop/lab6/adder4 - adder4 - [Waveform2.vwf]*

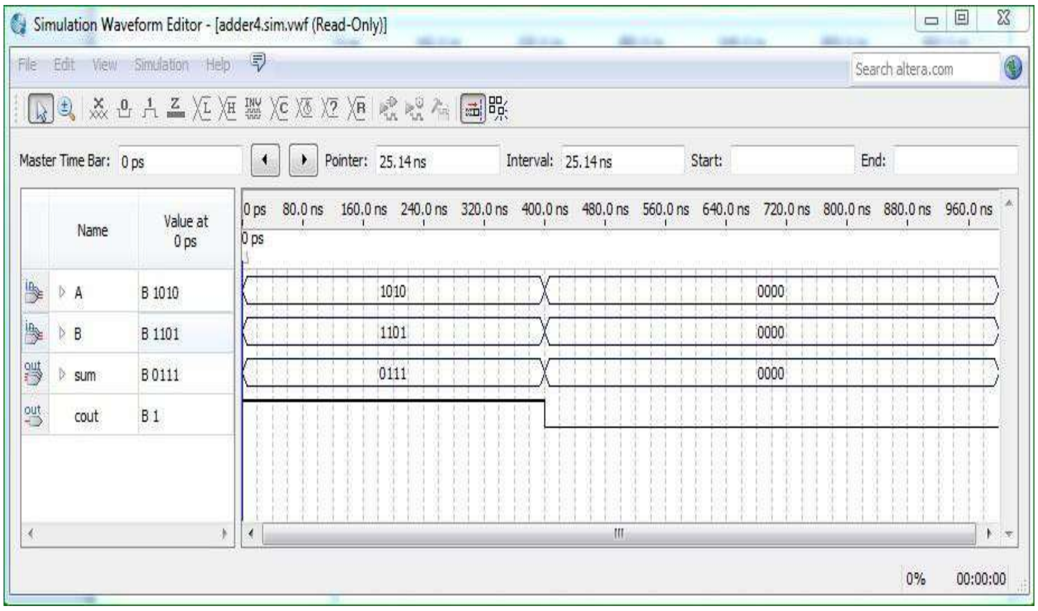
Master Time Bar: 0 ps Pointer: 97.13 ns Interval: 97.13 ns Start: End:

Name	Value at 0 ps
A	B 1010
B	B 1101
sum	B XXXX
cout	B X

Leave **Sum** with a value of undefined (X), as values for this will be generated when the simulation runs. Click the **Save** icon on the Menu bar and save the vwf file with the filename **adder4.vwf**.

6. In the Main window, select *Simulation* | *Options* and then select *Quartus II Simulator*. Select *OK*.
7. In the Main window, select *Simulation* and then select *Run Functional Simulation*.
8. Now you should see your simulation output with the outputs defined.

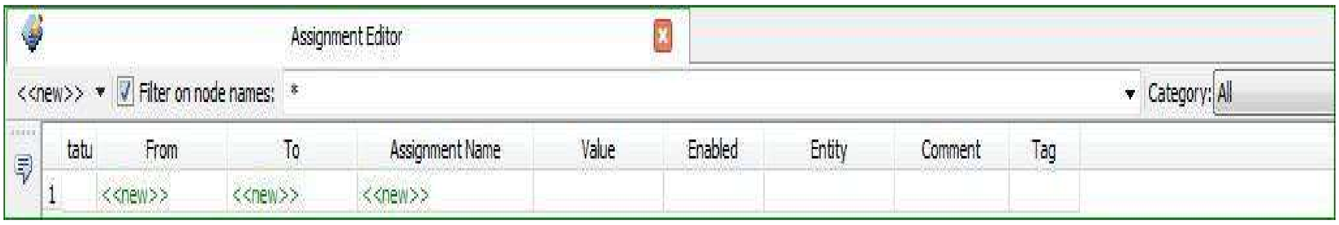
Note: The file will indicate "read-only" meaning you can't edit it.


Simulation Waveform Editor - [adder4.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 25.14 ns
Interval: 25.14 ns
Start:
End:
Name
Value at 0 ps
A B 1010
B B 1101
sum B 0111
cout B 1
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
1010
0000
1101
0000
0111
0000
0%
00:00:00

Task 4: Implementing the Design on the DE2 Board

1. From the Menu Bar select: **Assignments** → **Assignment Editor**

2. Double-click **<<new>>** in the **To** column and select **Node Finder** button, List down all the input and output pins and click OK.



tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	<<new>>	<<new>>	<<new>>					

3. Double-click the adjacent cell **Assignment Name** and select **Location (Accepts wilds cards/groups).**

   Double-click the adjacent cell **Value** and assign the pin number.

   Continue to assign the pins as seen in the table below.

	To	Location	DE2 Board Description
1	A[0]	PIN_AB28	SW0
2	A[1]	PIN_AC28	SW1
3	A[2]	PIN_AC27	SW2
4	A[3]	PIN_AD27	SW3
5	B[0]	PIN_AB27	SW4
6	B[1]	PIN_AC26	SW5
7	B[2]	PIN_AD26	SW6
8	B[3]	PIN_AB26	SW7
10	Sum[0]	PIN_G19	LED0
11	Sum[1]	PIN_F19	LED1
12	Sum[2]	PIN_E19	LED2
13	Sum[3]	PIN_F21	LED3
14	Cout	PIN_F18	LED4

4. Save the pin assignments by selecting **File** → **Save** from the Menu Bar, or by clicking the Save button on the toolbar.

5. Compile the design again by clicking the Start Compilation button on the toolbar.

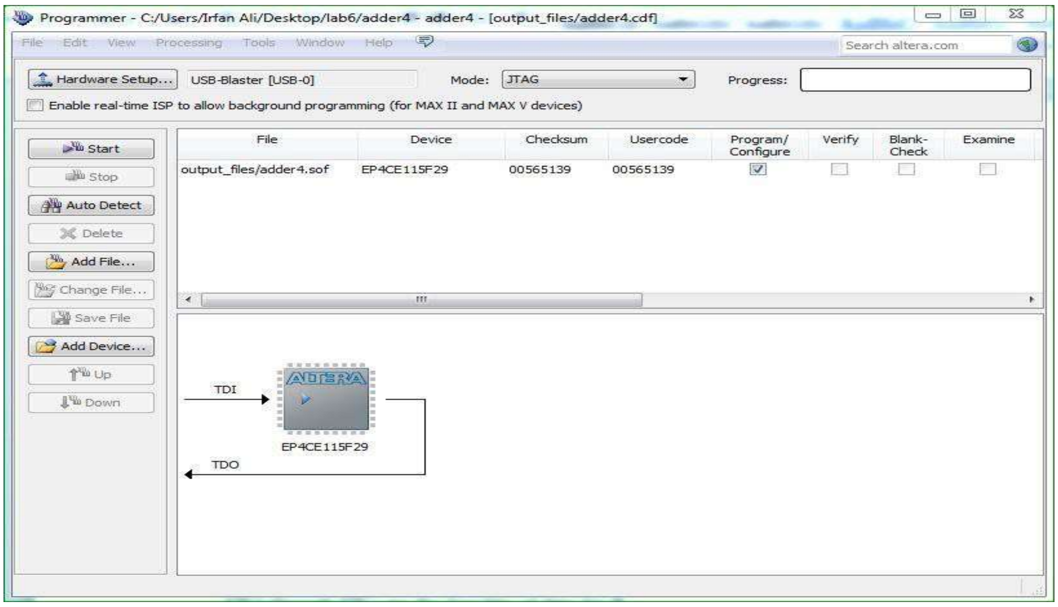
6. Plug in and power on the DE2 board. Make sure that the **RUN/PROG Switch for JTAG/AS Modes** is in the **RUN** position.

7. In the Quartus II window click the **Programmer** button on the Toolbar to open the Programmer window

The **Hardware Setup…** must be **USB-Blaster [USB-0]**. If not, click the **Hardware Setup…** button and select **USB-Blaster [USB-0]** from the drop-down menu for **currently selected hardware**.

Mode should be set to **JTAG**.

Make sure that the **File** is **adder4.sof**, **Device** is **EP4CE115F29C7N** and the **Program/Configure** box is checked.


Programmer - C:/Users/Irfan Ali/Desktop/lab6/adder4 - adder4 - [output_files/adder4.cdf]

Hardware Setup... USB-Blaster [USB-0]   Mode: JTAG   Progress:

Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine
output_files/adder4.sof	EP4CE115F29	00565139	00565139	✓			

Then click the **Start** button to program the DE2 board. When the progress bar reaches 100%, programming is complete.

8. You can now test the program on the DE2 board by using the toggle switches located along the bottom of the board.

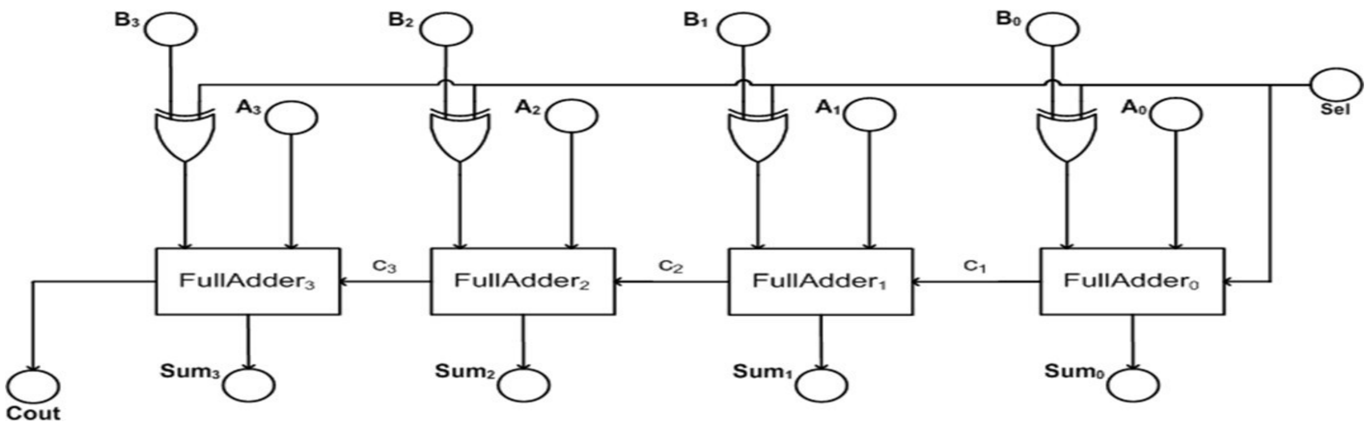
   SW0 through SW3 are the four bits of data for A.
   SW4 through SW7 are the four bits of data for B.

   The output of the operation is displayed on the first four LEDs (LED0-LED3) located above the blue push buttons on the DE2 board.

   LED4 is the indicator for Cout.

Practice Tasks: Transfer VHDL code into the FPGA board

Use below diagram to write a VHDL code for a 4-bit Subtractor/Adder. Show the waveform.



Rubric for lab assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed all the tasks and showed the results without any help of the instructor.	4	
Good	The student completed all the tasks and showed the results with minimal help of the instructor.	3	
Average	The student partially completed the task and showed results.	2	
Worst	The student did not complete the task.	1	

Instructor Signature:________________________________Date: ______________________