Name	Muhammad Asad
Reg. #	2019-EE-383
Marks	25 23-11-20

Experiment # 6

Design combinational circuit of adder using HDL language

Objective:

- *Introduction of* HDL *language and its syntax*
- *Introduction of Modelsim Software*
- *Designing of full adder circuit in Verilog using ModelSim tool*

Pre-Lab:

- Download the ModelSim Software on your laptops

Theory:

Manual methods for designing logic circuits are feasible only when the circuit is small. For anything else (i.e., a practical circuit), designers use computer-based design tools. Prototype integrated circuits are too expensive and time consuming to build, so all modern design tools rely on a hardware description language to describe, design, and test a circuit in software before it is ever manufactured.

A **hardware description language (HDL)** is a computer-based language that describes the hardware of digital systems in a textual form. It resembles an ordinary computer programming language, such as C, but is specifically oriented to describing hardware structures and the behavior of logic circuits. It can be used to represent logic diagrams, truth tables, Boolean expressions, and complex abstractions of the behavior of a digital system. One way to view an HDL is to observe that it describes a relationship between signals that are the inputs to a circuit and the signals that are the outputs of the circuit.

Simulation detects functional errors in a design without having to physically create and operate the circuit. Errors that are detected during a simulation can be corrected by modifying the appropriate HDL statements. The stimulus (i.e., the logic values of the inputs to a circuit) that tests the functionality of the design is called a test bench. Thus, to simulate a digital system, the design is first described in an HDL and then verified by simulating the design and checking it with a test bench, which is also written in the HDL.

Verilog HDL Syntax:

1. Each statement must be terminated using semicolon (;)
2. Start with a keywork **module**
    - Define the name of the function
    - Name the variables for inputs and outputs

e.g. **module lab6(A, B, C, D);**

3. Define the inputs and outputs of the circuits and any middle wire
    - **input A, B, C;**
    - **output D;**
    - **wire X**
4. Write the body of the function which you want to execute
5. End with a keywork **endmodule**
6. Make a test_bench to simulate the above circuit.
    - **module** test_bench();
7. Define registers and wires to input the values and see the output
    - wire D; *(For Outputs)*
    - reg A, B, C; *(For Inputs)*
8. Call the defined function
    - lab6 test(A, B, C, D);
9. Assign the values to the inputs and simulate the circuit.

initial ***(initialization)***
begin ***(beginning of sequence)***
A=0; B=0; C=1; *(any input for which you want to check or all the combinations)*
end
endmodule

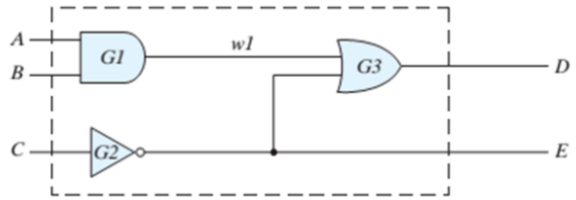


Figure 1 Circuit to demonstrate an HDL

```
module  Simple_Circuit (A, B, C, D, E);
 output        D, E;
 input         A, B, C;
 wire          w1;

 and           G1 (w1, A, B); // Optional gate instance name
 not           G2 (E, C);
 or            G3 (D, w1, E);
endmodule
```

Figure 2 Combinational Logic Modeled

Exercise 1:

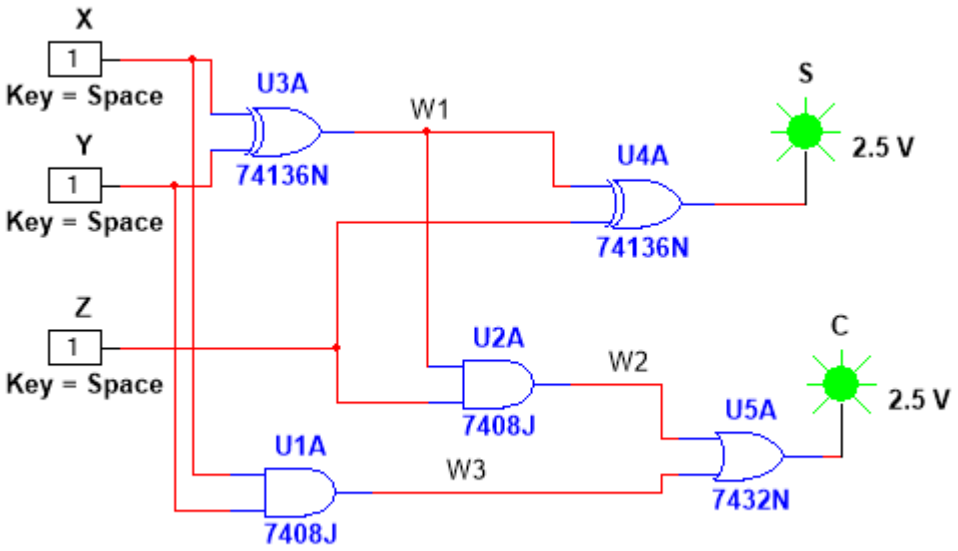
Design and full adder circuit and implement on ModelSim.

Designing Calculations:

S = X'Y'Z + X'YZ' + XY'Z' + XYZ
C = X'YZ + XY'Z + XYZ' + XYZ = **Z(X'Y + XY') + XY**

Truth table ?
K-map

Circuit Diagram:

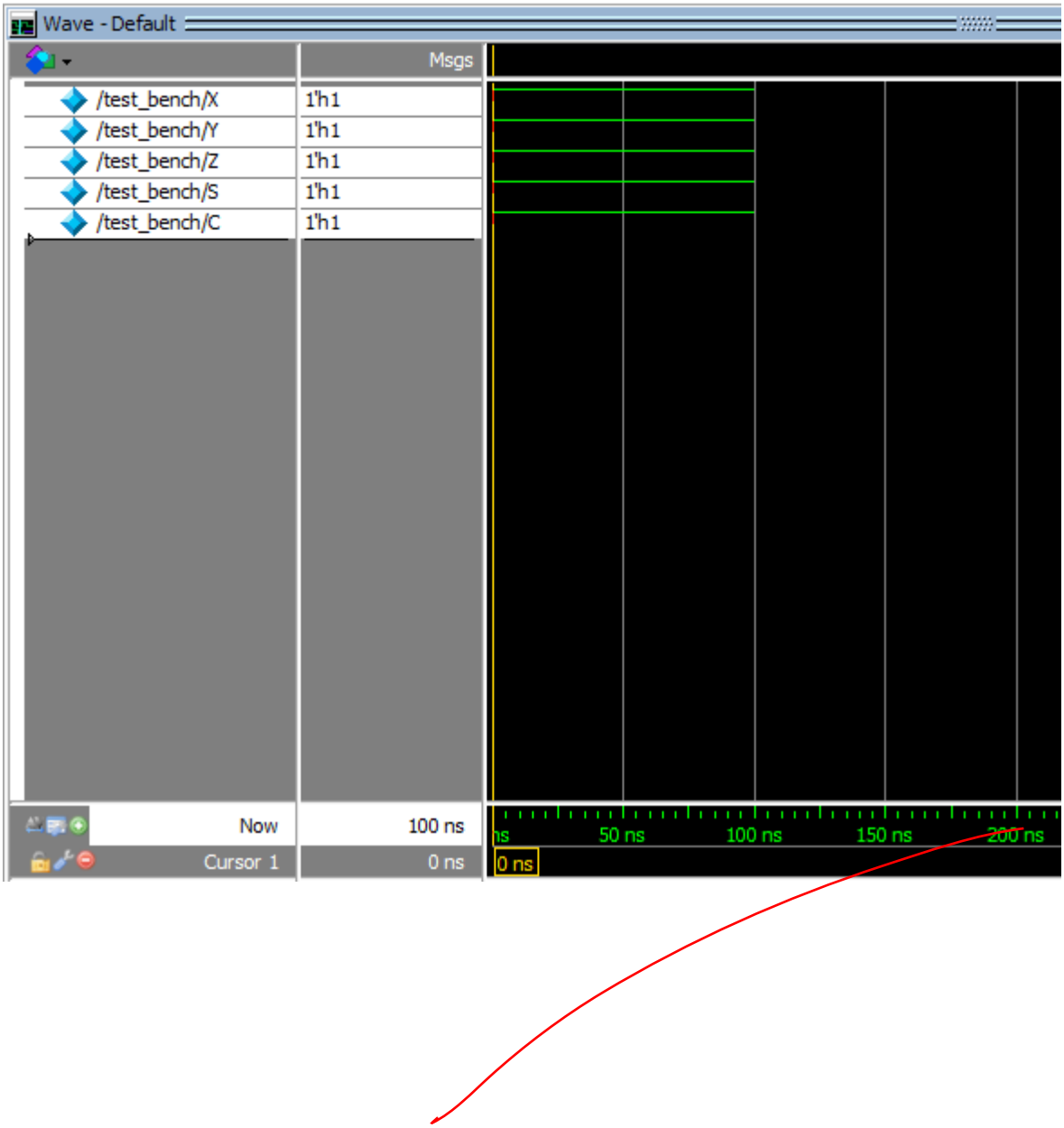


HDL codes:

C:/Modeltech_pe_edu_10.4a/examples/full_adder.v (/test_bench) - Default

```
Ln#
 1  module full_adder (X,Y,Z,S,C);
 2  input X,Y,Z;
 3  output S,C;
 4  wire W1,W2,W3;
 5  xor G1(W1,X,Y);
 6  xor G2(S,W1,Z);
 7  and G3(W2,W1,Z);
 8  and G4(W3,X,Y);
 9  or G5(C,W2,W3);
10  endmodule
11  module test_bench();
12  reg X,Y,Z;
13  wire S,C;
14  full_adder test(X,Y,Z,S,C);
15  initial
16  begin
17  X=1;Y=1;Z=1;
18  end
19  endmodule
20
```

Output waveforms:

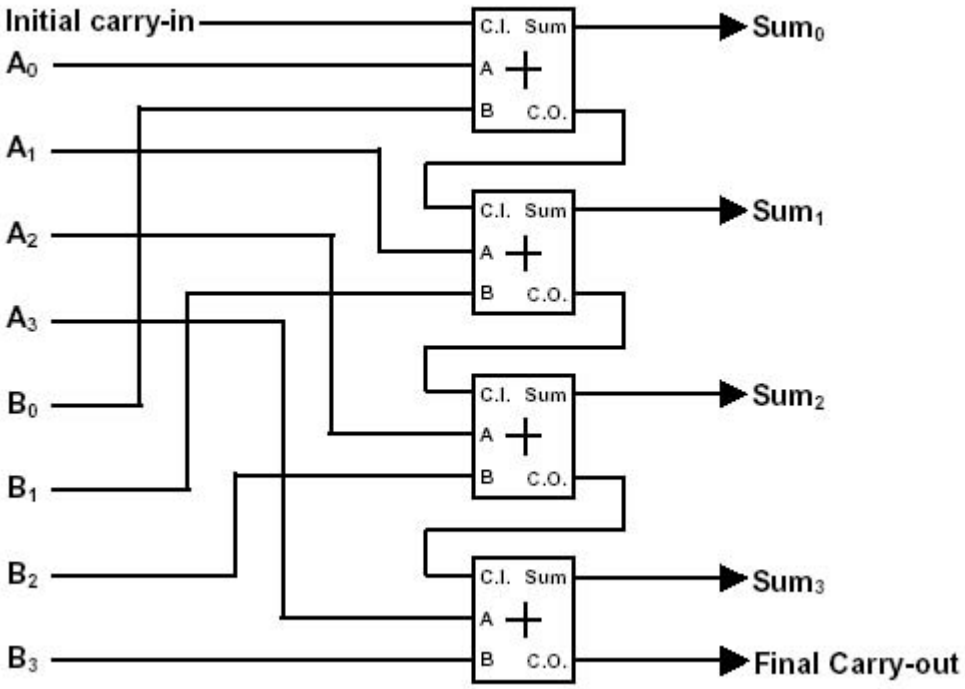


Exercise 2:

Design a 4-bits parallel adder and implement on ModelSim.

Hint: You will be using arrays

Circuit Diagram:

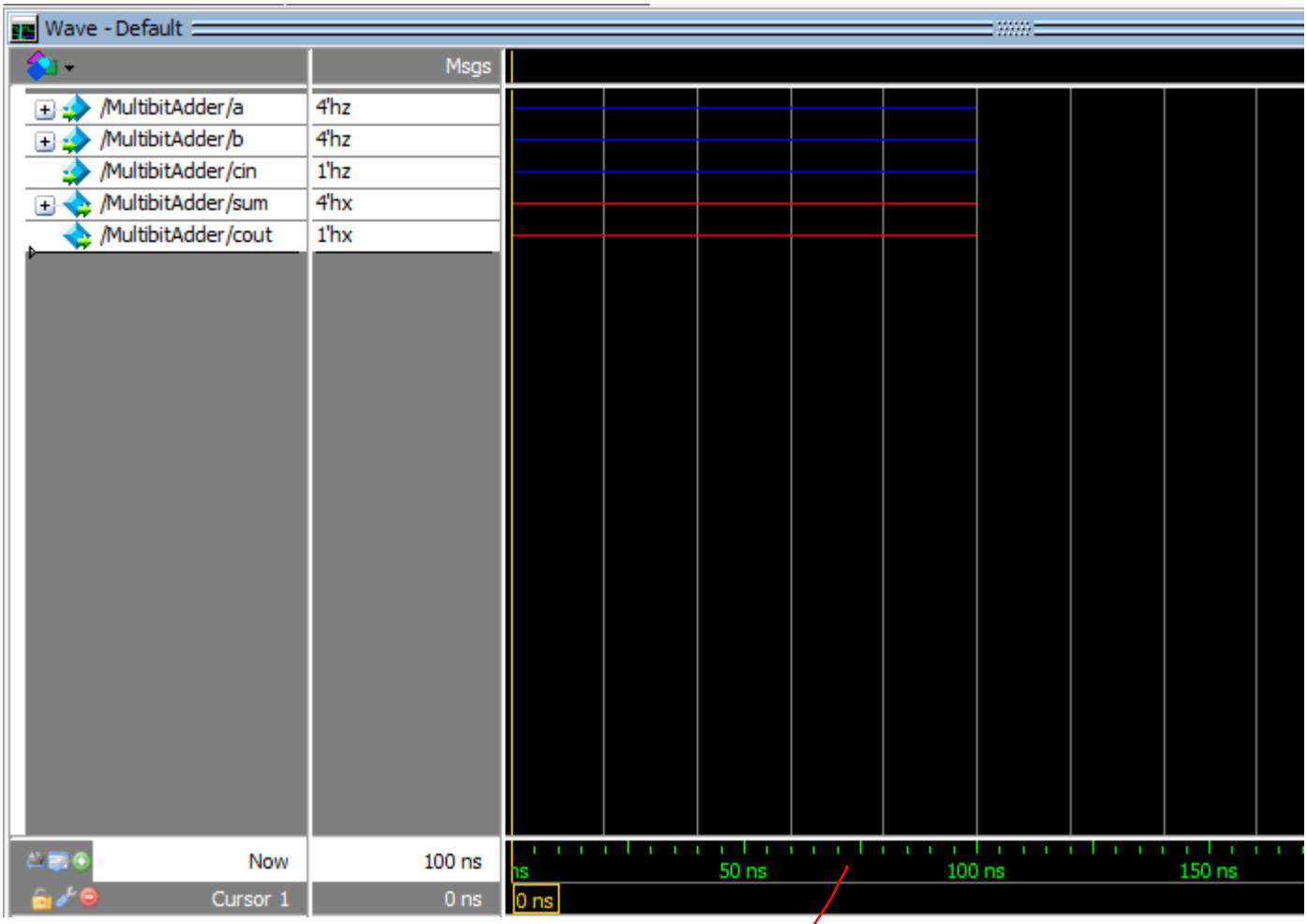


HDL codes:

C:/Modeltech_pe_edu_10.4a/examples/my-test.v (/MultibitAdder) - Default

```
Ln#
  1  module MultibitAdder(a,b,cin,sum,cout);
  2  input [3:0] a,b;
  3  input cin;
  4  output [3:0]sum;
  5  output cout;
  6  assign {cout,sum}=a+b+cin;
  7  endmodule
  8  module TestModule;
  9  // Inputs
 10  reg [3:0] a;
 11  reg [3:0] b;
 12  reg cin;
 13
 14  // Outputs
 15  wire [3:0] sum;
 16  wire cout;
 17
 18
 19  // Instantiate the Unit Under Test (UUT)
 20  MultibitAdder uut (
 21  .a(a),
 22  .b(b),
 23  .cin(cin),
 24  .sum(sum),
 25  .cout(cout)
 26  );
 27
 28  initial begin
 29  // Initialize Inputs
 30  a = 0;
 31  b = 0;
 32  cin = 0;
 33  // Wait 100 ns for global reset to finish
 34  #100;
 35
 36  a = 2;
 37  b = 3;
 38  cin = 1;
 39
 40  // Wait 100 ns for global reset to finish
 41  #100
 42
 43  end
 44  endmodule
```

Output waveforms:



Conclusions :-

In this lab;

- ✓ We will learnt about how we can make Full_Adder on Modelsim Software.
- ✓ We also learnt how we get Output in the form of Wave.
- ✓ We also learnt how we make a code for 4-bit Parallel_Adder and then see it's Output waveform.