BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

Lab Sheet - 1

Learning Objectives:

i) **Getting started with HDL program using Icarus Simulator**
ii) **Understand basic Verilog language primitives (e.g. module, data types, identifiers, vectors, registers, keywords etc.)**
iii) **To understand the various types of modelling**
iv) **Lab Exercises**

What is Verilog?

Verilog is one of the two-major Hardware Description Languages (HDL) used by hardware designers in industry and academia. VHDL is the other one. Many feel that Verilog is easier to learn and use than VHDL. VHDL was made an IEEE Standard in 1987, and Verilog in 1995. Verilog is very similar to C-language.

Verilog allows a hardware designer to describe designs at a high level of abstraction such as at the architectural or behavioral level as well as the lower implementation levels (i. e., gate and switch levels) leading to Very Large Scale Integration (VLSI) Integrated Circuits (IC) layouts and chip fabrication. A primary use of HDLs is the simulation of designs before the designer must commit to fabrication.

The Verilog language provides the digital designer with a means of describing a digital system at a wide range of levels of abstraction, and, at the same time, provides access to computer-aided design tools to aid in the design process at these levels.

Verilog Structure

Verilog differs from regular programming languages (C, Pascal, ...) in 3 main aspects:

1) Simulation time concept,
2) Multiple threads, and
3) Some basic circuit concepts like network connections and primitive gates.

If you know how to program in C and you understand basic digital design then learning Verilog will be easy.

1. Modules

In Verilog, circuit components are designed inside a **module**. Modules can contain both structural and behavioral statements. Structural statements represent circuit components like logic gates, counters, and microprocessors. Behavioral level statements are

programming statements that have no direct mapping to circuit components like loops, if-then statements, and stimulus vectors which are used to exercise a circuit.

A module starts with the keyword module followed by an optional module name and an optional port list. The key word endmodule ends a module.

Syntax: **module <module_name > (<module_terminal_list>);**
.....
<module_internals>
.....
.....
endmodule

Example:

```
`timescale 1ns / 1ps
//create a NAND gate out of an AND and an Invertor
module some_logic_component (c, a, b);
       // declare port signals
  output c;
  input a, b;
      // declare internal wire
  wire d;
      //instantiate structural logic gates
  and a1(d, a, b); //d is output, a and b are inputs
  not n1(c, d);    //c is output, d is input
endmodule
```

2. **Levels of Abstraction in Verilog Programming**:

I. Gate Level Modeling
II. Data Flow Modeling
III. Behavioral Modeling
IV. RTL Modeling

Gate Level Modeling
This is the basic level of modeling in terms of logic gates and the connections between these gates. Most digital designs are now done at the gate level or higher levels of abstractions. At gate level, the circuit is described in terms of gates say AND, OR etc. Hardware design at this level is intuitive for a user who is familiar with the basic knowledge of Digital logic Design. This allows the user to see a direct correspondence between the Verilog Description and the Circuit Diagram.

Example:

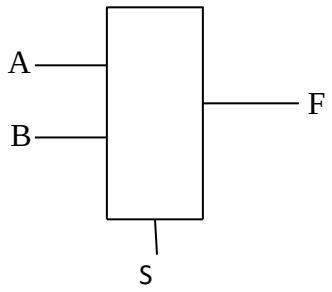

Figure 1. A 2:1 Multiplexer

In terms of Logic Gates, F= OR (AND (S, A), AND ((NOT (S)), B))

Code:

```
module mux2to1_gate (a,b,s,f);
      input a,b,s;
      output f;
      wire c,d,e;

      not n1(e,s);// e=~s
      and a1(c,a,s);
      and a2(d,b,e);
      or o1(f,c,d);
endmodule
```

For Further reading on Gate Level Modeling refer Chapter 5 of the book “Verilog HDL” by Samir Palnitkar.

Data Flow Modeling

The design at this level specifies how the data flows between the hardware registers and how the data is processed. For small circuits the gate level modeling works well as the number of gates is limited. However, in complex designs the designers may have to concentrate on implementing the function than bother about the gates. Verilog allows a circuit to be designed in terms of the data flow between registers and how a design processes data rather than instantiation of gates using expressions (=), operators like (&,|, ?) etc.. and continuous assignments(the assign statement).

The 2:1 Multiplexer can be written as

F= (S&A)|(~S&B);
Or
F = S ? A : B;

Code:

```
module mux2to1_df (a,b,s,f);
      input a,b,s;
      output f;
      assign f = s ? a : b;
endmodule
```

For Further reading on Data Flow Modeling refer Chapter 6 of the book “Verilog HDL” by Samir Palnitkar.

Behavioral Modeling

This is the highest level of abstraction provided by Verilog. The design at this level is similar to an algorithm. This design is very similar to „C" programming. A module can be implemented in terms of the desired design algorithm without looking into the hardware details using structured procedures (like *always* and *initial*), conditional statements (like *if* and *else*) and multiway branching (like *case*, *casex* and *casez*).
[Note: THOUGH THE CODING STYLE IS SIMILAR TO C, THE DIFFERENCE IS THAT, THE VERILOG PROGRAM CONSISTS OF MODULES THAT MAY RUN CONCURRENTLY. Remember, that you need to simulate hardware and not software.]

The 2:1 Multiplexer can be written as
If(s= = 1)
F=A;
Else
F=B

Code:

```
module mux2to1_beh(a,b,s,f);
      input a,b,s;
      output f;
      reg f;
      always@(s or a or b)
            if(s==1) f = a;
            else f = b;
endmodule
```

For Further reading on Behavioral Modeling refer Chapter 7 of the book "Verilog HDL" by Samir Palnitkar.

RTL Modeling

The term Register Transfer Level Modeling refers to the Verilog description that uses a combination of both Behavioral and Data Flow constructs that is synthesizable. More on this later.

For the input and output connections for the ports, please refer Sec 4.2.3 Port Connection Rules.

Simulation

Testbench or Stimulus

Once a design block is completed, it must be tested for its correctness. The functionality of a design block can be tested by applying stimulus and checking the results. We call such a block Stimulus or TestBench. It is recommended to keep the design and the stimulus blocks separate. Given below is an example to test the 2:1 Multiplexer we have designed so far.

Code:

```
module testbench;
      reg a,b,s;
      wire f;
      mux2to1_gate mux_gate (a,b,s,f);
      initial
            begin
                  $monitor(,$time," a=%b, b=%b, s=%b f=%b",a,b,s,f);
                  #0 a=1'b0;b=1'b1;
                  #2 s=1'b1;
                  #5 s=1'b0;
                  #10 a=1'b1;b=1'b0;
                  #15 s=1'b1;
                  #20 s=1'b0;
                  #100 $finish;
            end
endmodule
```

Note: To test out the data flow model and the behavioral model of the 2:1 Multiplexer, replace

```
mux2to1_gate mux_gate (a,b,s,f);
with
mux2to1_df mux_df (a,b,s,f);
or with
mux2to1_beh mux_beh (a,b,s,f);
```

Please refer section 2.5 and 2.6 of the book "Verilog HDL" by Samir Palnitkar.

Lab Exercises:

Note: Icarus Verilog Simulator has to be used for implementing the Lab exercises.

(1) Implement BCD to gray code conversion using following:

i. Gate level model
ii. Dataflow modeling

Write a test bench which includes all cases and checks the correctness of the design.

(2) Implement a 4 bit magnitude comparator using

i. Behavioral modeling

ii. Dataflow modeling

iii. Gate Level modeling

Write a test bench to check the correctness of the design.

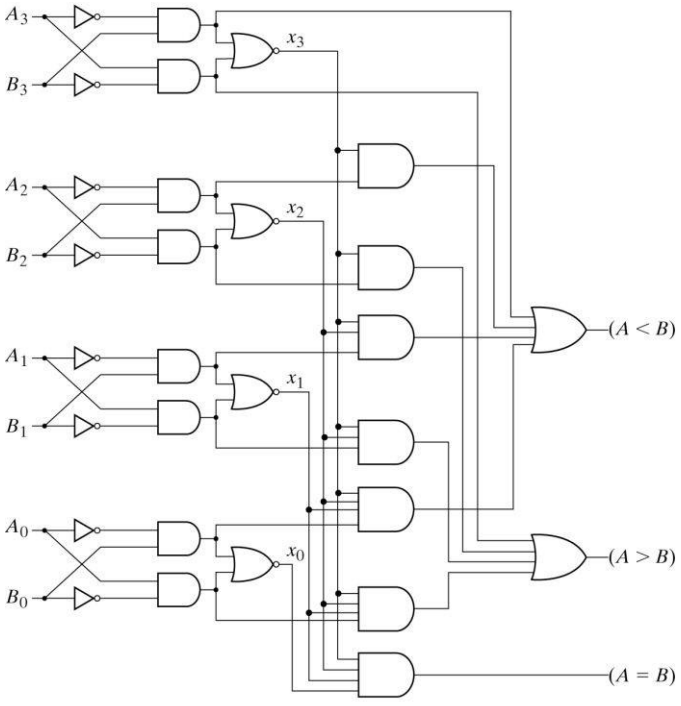

Fig. 4-17 4-Bit Magnitude Comparator

Take Home Exercises:

(1.a.) Implement a 1-bit full adder using following:

i. Gate level model
ii. Data flow model

Write a test bench which includes all cases and checks the correctness of the design.

(1.b.) Design a 4-bit adder by using 1-bit adder. Write a test bench which checks the correctness of the design.

(1.c.) Design a 4 – bit Adder / Subtractor with a select line S. Write a test bench which checks the correctness of the design.
