MODULE-5 INTRODUCTION TO VERILOG

Verilog Behavioral description: Structure, Variable Assignment Statement, Sequential Statements, Loop Statements, Verilog Behavioral Description of Multiplexers (2:1, 4:1, 8:1). (Section 3.1 to 3.4 (only Verilog) of Text 3)

Verilog Structural description: Highlights of Structural description, Organization of structural description, Structural description of ripple carry adder. (Section 4.1 to 4.2 of Text 3)

5.1 Behavioral Description highlights

The behavioral description is a powerful tool to describe the systems for which the digital logic structures are not known or hard to generate. Examples of such system are complex arithmetic units, computer control units, and some biological mechanisms.

In dataflow modeling a system was described by using logic equations. In behavioral modeling a system is described by showing how the outputs behave according to the changes in inputs.

In this style of description (modeling), it is not necessary to know the logic diagram of the system, but we must know the behavior of the outputs in response to the changes in the inputs

The primary mechanisms used for behavioral modeling in Verilog are

i) Initial statement: An initial statement executes only once. It begins its execution at the start of the simulation which is at time 0. The syntax for the initial statement is,

```
initial
[timing control]
Procedural_statements
```

*timing_control can be a delay or an event control.
*Where procedural_statement can be one of the following:

1. Continuous_assignment
2. Conditional_statement
3. Case_statement
4. Loop_statement
5. Wait_statement
6. Sequential_block

*execution of an initial statement will result in execution of the procedural_statement once.
Here is an example of an initial statement

```
reg yurt;
……….
initial
Yurt=2;
```

In the above example the initial statement contains a procedural assignment with no timing control(delay). The initial statement executes at time 0 and yurt is assigned value 2 at time 0 only.

Consider

```
Reg curt;
……….
Initial
#5 curt=1;
```

Register curt is assigned value 2 at time 5. The initial statement starts execution at time 0 but completes execution at time 5.

ii) __al ways statement__: Just like the **initial** statement, an **always** statement also begins execution at time 0. But In contrast to the **initial** statement, an **always** statement executes repeatedly. The syntax of an always statement is:

```
          always
            [timing_control]
           Procedural_statement.                  //same as explained in initial statement.]
```

Example:

```
          always
           clk=~clk;                              //will loop indefinitely.
```

In the above example the **always** statement has one procedural_statement. Since always statement executes repeatedly and there is no timing control the statement clk=~clk will loop indefinitely. Therefore an **always** statement must have a timing control (delay or an event).
Consider

```
          always
          #5 clk=~clk;
```

This **always** statement upon execution, produces a waveform with a period of 10 time units.

5.2 Structure of Verilog Behavioral Description

__**Verilog description**__

```
module halfadd(a, b, sum, carry);
input a, b;
output sum, carry;
reg sum, carry;                         /*since sum and carry are outputs and they are written inside
                                        "always", they should be declared as "reg" or else it will result
                                        in syntax error/*
always @(a, b)
     begin
        #10 sum = a^b;                  //statement1-procedural as it is inside always.
        #10 carry = a&b;                //statement2- procedural as it is inside always.
     end
endmodule
```

In the above example
*The name of the module is halfadd. It has two inputs a and b, two outputs sum and carry.

*Any signal declared as output should be declared as a register (reg).Therefore sum and carry are declared as registers.

5.3 Sequential statements

There are several statements associated with the behavioral descriptions which appear inside initial or always in Verilog.

5.3.1 IF statement

"IF" is a sequential statement that appears inside inside always or initial in Verilog. An "IF" statement selects a sequence of statements for execution, depending upon the value of a condition. The condition can be an expression that evaluates to a Boolean value. The general form of an IF statement is:

Verilog if format

```
If (boolen expression)
begin
   statement1;
   statement2; ….
end
else
begin
   statement a;
   statement b; ….
end
```

The execution of "if" is controlled by the Boolean expression. If the expression is true, then statements A are executed. If the expression is false, statements B are executed.

```
if(temp==1)                 //Verilog
  temp=s1;
else
temp=s2;
```

Execution of IF as ELSE-IF

Verilog

```
if (Boolean expression1) begin
     statement 1;
statement 2;
  end
  else if (Boolean expression2) begin
```

```
    statement i;
    statement ii;

  end

  else begin
    statement a;
    statement b;
  end
```

Example program:

```
if(signal 1==1) //Verilog
temp=s1;
else if(signal 2==1)
 temp=s2;
else
temp=s3;
```

Examples1: *Behavioral description of the* ***2X1 Multiplexers with tristate output****.*
Flowchart & Logic symbol

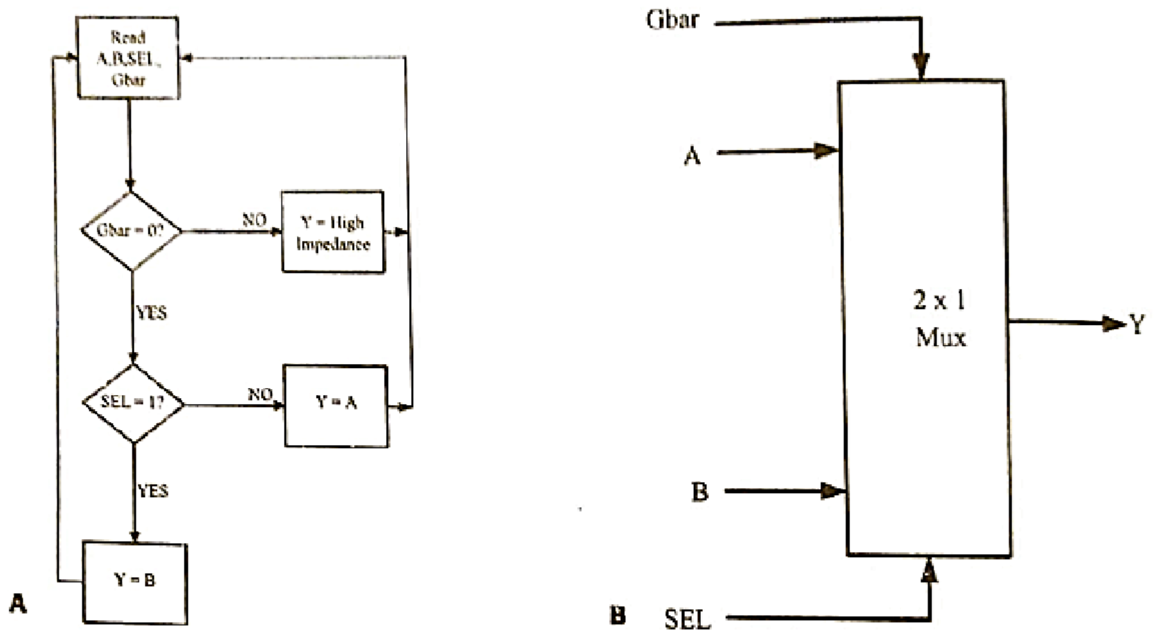


FIGURE 3.1 2x1 Multiplexer. (a) Flow chart. (b) Logic symbol.

Verilog Program using IF-ELSE and ELSE-IF

Verilog 2x1 Multiplexer Using IF-ELSE

```
module mux2x1 (A, B, SEL, Gbar, Y);
input A, B, SEL, Gbar;
output Y;
reg Y;
always @(SEL, A, B, Gbar)
begin
        if (Gbar == 1)
                Y =1'bz;
        else
        begin
                if (SEL)
                        Y = B;
                else
                        Y = A;
        end
end
endmodule
```

Verilog 2x1 Multiplexer Using ELSE-IF

```
module MUXBH (A, B, SEL, Gbar, Y);
input A, B, SEL, Gbar;
output Y;
reg Y;
 always @ (SEL, A, B, Gbar)
begin
                if (Gbar == 0 & SEL == 1)
                begin
                        Y = B;
                end
                else if (Gbar == 0 & SEL == 0))
                Y =A;
                else
                Y = 1'bz;
end
endmodule
```

5.3.2 Signal and Variable assignment

With the help of *Behavioral description* of a D-latch, here we study the difference between the signal- and Variable assignment statements.

A process is written based on signal-assignment statements, and then another program is written based on Variable assignment statements. A comparison of the simulation waveforms highlights the difference between the two methods.

Examples2: *Behavioral description of the* ***D-Latch.***

```
module D_latch (d, E, Q, Qb);
input d, E;
output Q, Qb;
reg Q, Qb;
always @ (d, E)
begin
        if (E == 1)
                begin
                        Q =     d;
                        Qb =    ~Q;
                end
end
endmodule
```

Examples3: *Behavioral description of a* ***Positive edge triggered JK Flip-Flop*** *using the CASE statements.*

```
module JK_FF (JK, clk, q, qb);
input [1:0] JK;
input clk;
output g, gb;
reg q, qb
always @(posedge clk)
begin
case (JK)
2'd0 :q =q;
2'd1 :q = 0;
2'd2 :q =1;
2'd3: q= ~q;
endcase
```

Examples4: *Behavioral description of a* ***3-bit Binary counter with Active High Synchronous Clear***

```
module CT_CASE (clk, clr, q);
input clk, clr
output [2:0] q;
reg [2:0] q;
initial
        q= 3'b101;
always @ (posedge clk)
begin
        if (clr == 0)
begin
        case (q)
                3'd0 : q = 3' d1;
                3'd1 : q = 3'd2;
               3'd2 : q = 3'd3;
               3'd3 : q = 3'd4;
               3'd4 : q = 3'd5;
               3'd5 : q = 3'd6;
               3'd6 : q =3'd7;
               3'd7 : q = 3'd0;
               endcase
               end
else
q= 3'b000;
end
endmodule
```

5.3.4 Loop statement

- Loop is used to repeat the execution of the statements inside its body; this repeatation is controlled by the range of an index parameter.
- The loop reduces the size of the code.
- Loop is a sequential statement that has to appear inside process in VHDL or inside always or initial in verilog.
- There are several formats of loop statement namely for,while, verilog repeat and forever.

a).For -loop

The general syntax of for loop in Verilog is as shown below.
The general syntax of for loop in both VHDL and Verilog is as shown below.

Verilog

for (initial_assignment;condition;step_assignment)
begin
        Statements… ……. ;
End

For loop execution in Verilog:

Example for Verilog **for** loop

for(i=0;i<=2;i=i+1)
 	
begin
 	
        if (temp[i]==1)
 	
        begin
 	
          result=result+2;
 	
        end
 	
end

b).while loop

A while loop is another iterative statement. The general format of while loop is

Verilog

while(condition)
    begin
      Statement 1;
Statement 2;
 	
    end

As long as the condition is true, all the statements within the body of the loop are executed. If the condition is false the loop is suspended and the next statement after loop is executed.
Note: 1. In a while loop, we must ensure that statements inside the while loop will cause the loop's condition to evaluate false or else the loop infinite.

"For" loop and "while" loop are common to both VHDL and Verilog except some minor differences in the syntax. But apart from that there are some language specific loop statements.

c). Verilog repeat: The repeat construct executes the set of statements between it's begin and end a fixed number of times. A repeat construct must contain a number which can be a constant, variable or a signal. It cannot contain an expression or condition.
Example:

```
repeat (32)
begin
        #100 i=i+1;
end
```

*In the above example, the statement is executed 32 times. Each time "i" is incremented and assigned to itself after a delay of 100 time units.

d). Verilog forever: The statement "forever" in Verilog repeats the loop endlessly. The loop does not contain any expression and executes until it encounters "$finish" task. A forever can be exited or stopped by using "disable" statement.

- "forever" loop is equivalent to "while" loop with an expression that always evaluates to true.
- One common use of "forever" is to generate clocks in code-oriented test benches.

```
Initial
Begin
        clk=1'b0;
Forever #20 clk=~clk;
End
```

5.4 Highlights of Structural description.

The structural description is a method of defining a system with its basic hardware components. The structural description is best implementation when the hardware components are known. For example consider 2:1 multiplexer, this can be easily implemented if structural descriptions of the basic components are known i.e. ‘AND, OR and NOT gates. Structural description can easily describe these components. Structural description is very close to schematic simulation. Here we are going to discuss about gate level and register level descriptions for VERILOG and VHDL.

Facts:

1. Structural description stimulates the system by describing its logical components (AND gate, OR gate and NOT gate).or can be a higher logical level such as Register Transfer Level (RTL) or processor level.

2. The structural description is more suitable rather than behavioral description for the system that required a specific design. Consider for example, a system is performing the operation A+B=C. In behavioral description, we usually write C= A+B and we have no choice in the type of adders used to perform this addition. In structural description, we can specify the type of the adder. For example, look-ahead adders.

3. All statements in structural description are concurrent. At any simulation time, all statements that have an event executed concurrently.

4. The main difference between the VHDL and VERILOG structural description is availability of the components to the user. Verilog recognizes all the primitive gates, such as AND, OR, XOR, NOT, and XNOR gates. Basic VHDL packages do not recognize any gates unless the package is linked to one or more libraries, packages, or modules that have the gate descriptions.

5.5 Organization of the structural description:

The following program (HDL code) describes a system with the help of structural description, Example 4.1. In this program architecture part has parts **declaration and instantiation**.

In the declaration part all the components used in the system description are declared. For example following description declares XOR gate component.

Component xor2

port (i1, i2: in

std_logic:o1:O1: out

std_logic:

end component;

The xor2 component has tow inputs “i1” and “i2”, and one output “o1”.

Once the component is used we can use the same component one or more times in the system description. The instantiation part of the code maps the generic input/output to the actual input/output of the system. For example, the statement

X1: xor2 port map (A, B, sum) ;

Maps A to input i1 of xor2, input B to input i2 of xor2., and output sum to output o1 of xor2. This mapping means that the logic relationship between A,B and sum is same as between i1, i2 and o1.

Verilog has a large number of built-in gates; for example the statement

Xor X1(sum, a , b);

Describes a two-input XOR gate, the inputs a and b, and the output is sum. X1 is an optional identifier for the gate: we can also write it as

Xor (sum, a ,b);

Verilog has a complete list of built-in primitive gates. The output of the gate sum has to be listed before the inputs a and b. Figure4.1 shows a list of gates and their code in Verilog.

Program 4.1:HDL Structural Description—Verilog

Verilog Structural Description

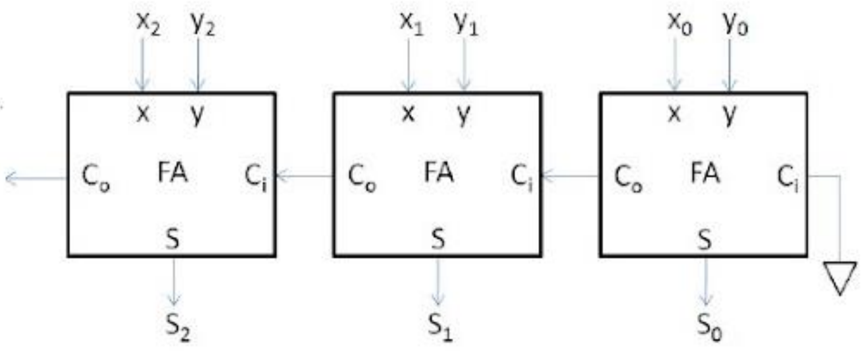
```
module system (a, b, sum, cout);
input a, b;
 output sum, cout;
 xor X1 (sum, a, b);               /* X1 is an optional identifier; it can be omitted.*/
 and a1 (cout, a, b);              /* a1 is optional identifier; it can be omitted.*/
 endmodule
```

Verilog Half Adder Description

```
module half_add (a, b, S, C);
    input a, b;
    output S, C;
    xor (S, a, b);
    and (C, a, b);
    endmodule
```

Example : Structural description of a 3 bit ripple-carry adder

The 3-bit ripple-carry adder is built using 3 1-bit full adders as shown in the following figure.



Verilog code for 1-bit full adder using structural modeling:

```
// Verilog code for 1-bit full adder

  module fulladder(A, B, Cin, SUM, COUT);
  input  A, B, Cin;
  output SUM, COUT;
  wire w1,w2,w3;
  //Structural code for one bit full adder
  xor G1(w1, A, B);
  xor G2(SUM, w1, Cin);
  and G3(w2, w1, Cin);
  and G4(w3, A, B);
  or G5(COUT, w2, w3);
endmodule
```

Then, instantiate the full adders in a Verilog module to create a 3-bit ripple-carry adder using structural modeling.

Following is the Verilog code for the 3-bit ripple-carry adder:

Verilog code for 3-bit ripple-carry adder:

```
module rippe_adder(X, Y, S, Co);
 input [2:0] X, Y;// Two 4-bit inputs
 output [2:0] S;
 output Co;
 wire a1, a2;
 // instantiating 3 1-bit full adders in Verilog
 fulladder u1(X[0], Y[0], 1'b0, S[0], a1);
 fulladder u2(X[1], Y[1], a1, S[1], a2);
 fulladder u3(X[2], Y[2], a2, S[2], Co);
endmodule
```