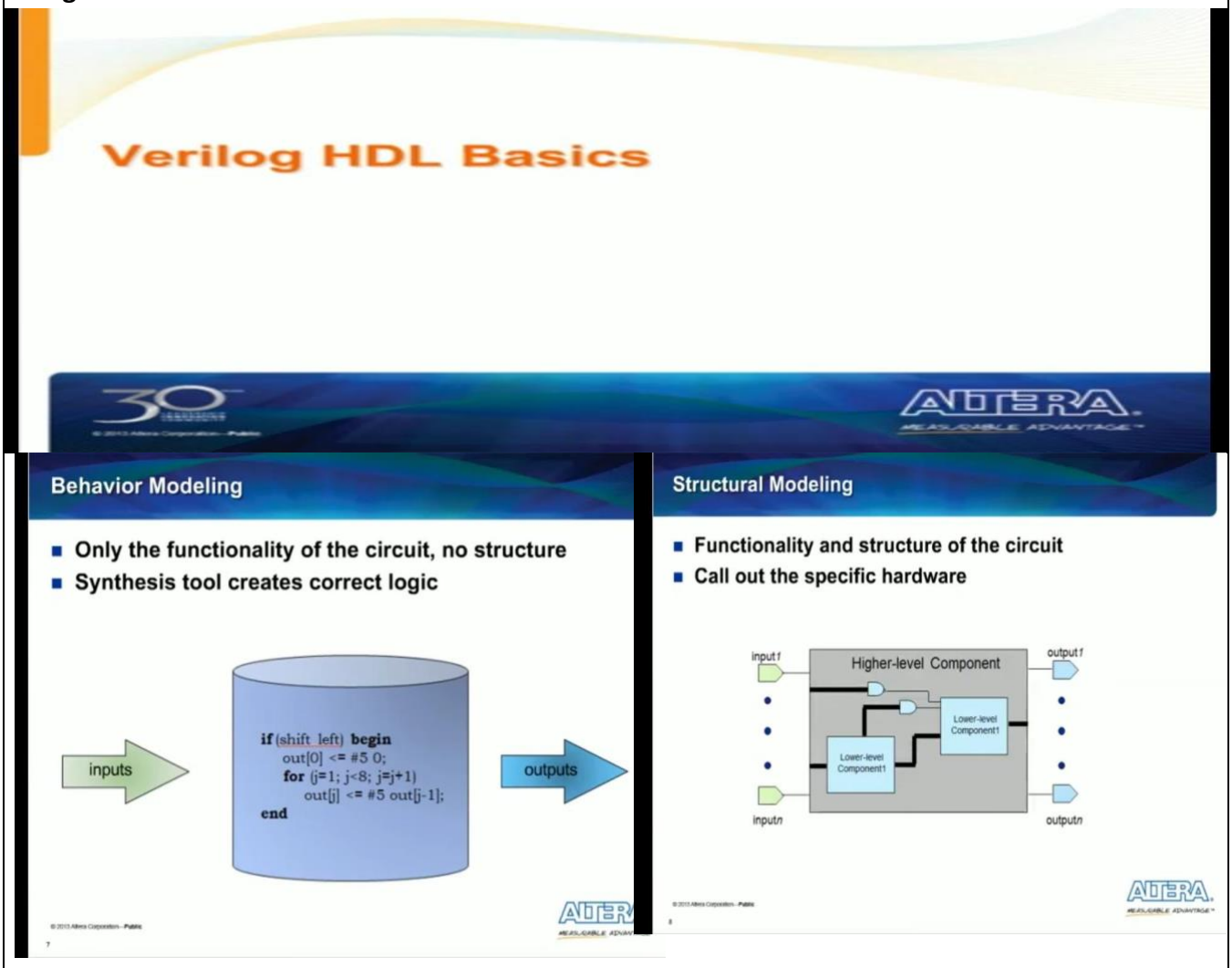
DAILY ASSESSMENT FORMAT

Date:	**02-06-2020**	**Name:**	**Sahana S R**
Course:	**Digital design using HDL**	**USN:**	**4al17ec083**
Topic:	**•FPGA Basics Architecture Application and Uses •Verilog HDL Basics by intel •Verilog testbench code to verify the design under test (DUT)**	**Semester & Section:**	**6th sem B sec**
Github Repository:	**sahanasr-course**		

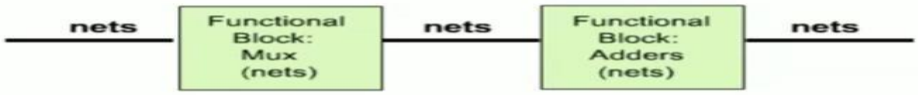
FORENOON SESSION DETAILS

Image of session



Data Types

- **Net data type - represents physical interconnect between structures (activity flows)**



- **Variable data type - represents element to store data temporarily**

17

Parameter

- Value assigned to a symbolic name
- Must resolve to a constant at compile time
- Can be overwritten at compile time
- localparam – same as parameter but cannot be overwritten

```
parameter size = 8;
localparam outsize = 16;
reg [size-1:0] dataa, datab;
reg [outsize-1:0] out
```

- Verilog-2001 style, include with module declaration

```
module mult_acc
   #(parameter size = 8)
   (...);
```


23

Blocking vs. Nonblocking Assignments

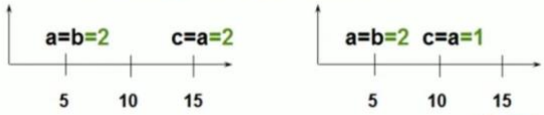
Blocking (=)

```
initial
  begin
      a = #5 b;
      c = #10 a;
  end
```

Nonblocking (<=)

```
initial
  begin
      a <= #5 b;
      c <= #10 a;
  end
```

Assuming initially a=1 and b=2



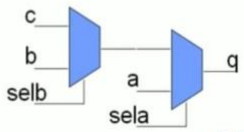
if-else Statements

- Format:

```
if <condition1>
   {sequence of statement(s)}
else if <condition2>
   {sequence of statement(s)}
   ...
else
   {sequence of statement(s)}
```

- Example:

```
always @ * begin
   if (sela)
      q = a;
   else if (selb)
      q = b;
   else
      q = c;
end
```



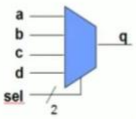
case Statement

- Format:

```
case {expression}
   <condition1> :
      {sequence of statements}
   <condition2> :
      {sequence of statements}
   ...
   default : -- (optional)
      {sequence of statements}
endcase
```

- Example:

```
always @ * begin
   case (sel)
      2'b00 : q = a;
      2'b01 : q = b;
      2'b10 : q = c;
      default : q = d;
   endcase
end
```



Synchronous vs. Asynchronous

Synchronous Preset & Clear

```
module dff_sync (
   input d, clk, sclr, spre,
   output reg q
);
   always @(posedge clk)
   begin
      if (sclr)
         q <= 1'b0;
      else if (spre)
         q <= 1'b1;
      else
         q <= d;
   end
endmodule
```

Asynchronous Clear

```
module dff_async (
   input d, clk, aclr,
   output reg q
);
   always @(posedge clk,
            posedge aclr)
   begin
      if (aclr)
         q <= 1'b0;
      else
         q <= d;
   end
endmodule
```


53

Clock Enable

Clock Enable

```
module dff_ena (
   input d, enable, clk;
   output reg q
);

/* If clock enable port does not exist in
   target technology, then a mux in
   front of the d input is generated */

   always @( posedge clk )
      if (enable)
         q <= d;

endmodule
```


54

Function Definition - Multiplier

```
function [15:0] mult;
   input [7:0] a, b;
   reg [15:0] r;
   integer i;
begin
   if (a[0] == 1)
      r = b;
   else
      r = 0;
   for (i =1; i <=7; i = i + 1) begin
      if (a[i] == 1)
         r = r + b << i;
   end
   mult = r;
end
endfunction
```


57

Functions vs. Tasks

Functions	Tasks
▪ Always execute in zero time – Cannot pause their execution – Cannot contain any delay, event, or timing control statements	▪ May execute in non-zero simulation time – May contain delay, event, or timing control statements
▪ Must have at least one input argument – Inputs may not be affected by function	▪ May have zero or more input, output, or inout arguments
▪ Arguments may not be outputs and inouts	▪ Modify zero or more values
▪ Always return a single value	▪ May call functions or other tasks
▪ May call another function but not a task	

60

```
module test_counter;
  reg clk, clr;
  wire [7:0] out;

  counter CNT (clr, clk, out);

  initial  clk = 1'b0;

  always  #5 clk = ~clk;

  initial
    begin
      clr = 1'b1;
      #15 clr = 1'b0;
      #200 clr = 1'b1;
      #10 $finish;
    end
```

```
  initial
    begin
      $dumpfile ("counter.vcd");
      $dumpvars (0, test_counter);
      $monitor ($time, " Count: %d", out);
    end

endmodule
```


Report – Report can be typed or hand written for up to two pages.

Day 2

What is Verilog

IEEE industry standard. Hardware Description Language. Used to describe a digital System.

Use in both hardware Simulation & Synthesis

- Behavior Modeling

Only the functionality of the Circuit. no structure

Synthesis tool - creates Correct logic.

- Structural Modeling

Functionality and Structure of the Circuit

Call out the Specific hardware

- More Terminology

→ Register Transfer Level

→ Synthesis → RTL Synthesis

Verilog - Basic Modeling Structure

- Data Types : Net datatype
- Module Instantiation
- Port Connection Rules
- operator precedence
- Parameter
- Arithematic Operators
- Relation operators
- Equality operators
- Logic operators
- Shift operators
- Miscellaneous operator

Full Adder

```
module full_adder (s, co, a, b, c);
  input a, b, c;
  output s, co;
  assign s = a ^ b ^ c;
  assign co = (a&b) | (b&c) | (c&a);
endmodule
```

4 bit Shift register

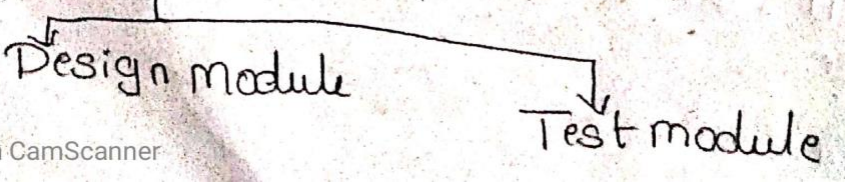
```
module shiftreg_4bit (clock, clear, A, E);
 input clock, clear, A;
 output reg E;
 reg B C, D;
always @ (posedge clock or negedge clear)
 begin
  if (!clear) begin
   B<=0; C<=0; D<=0; E<=0;
  end
  else begin
       E<=D;
       D<= C;
       C<= B
       B<= A
      end
  end
 endmodule
```

Automatic Verification of output

```
module fulladder (a, b, c, S Cout)
 input a, b, c;
 output S, Co
assign S = a^b^c;
 endmodule
```

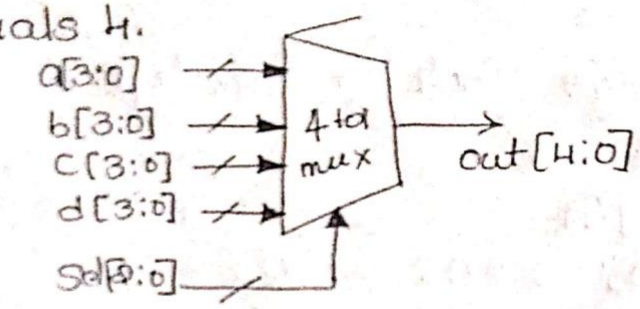
→ Synchronous and Asynchronous

→ Program Structure



4 to 1 Multiplexer

A multiplexer is a digital element that transfers data from one of the N inputs to the output based on the Select Signal. The case shown below is when N equals 4.



Assign Statement

```
module mux_4to1_assign( input[3:0] a, input[3:0] b,
    input[3:0]c, input[3:0]d, input[1:0]Sel, output[3:0]out);
assign out = Sel[1] ? (Sel[0] ? d:c):(Sel[0] ? b:a);
endmodule
```

Case Statement

```
module mux_4to1_case( input[3:0] a, input[3:0] b
    input[3:0] c, input[3:0]d, input[1:0]Sel, output[3:0]out)
always @ (a or b or c or d or sel)
   begin
  case(Sel)
    2'b00 : out <= a;
    2'b01 : out <= b;
    2'b10 : out <= c;
    2'b11 : out <= d;
  endcase
   end
endmodule.
```