

Hardware Modeling using Verilog
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 12
Verilog Description Styles

In the lectures that we have seen so far, we have looked at a number of examples where we took some modeling instances and showed; how we can create the description using Verilog. Now in this lecture, we shall be starting our discussion on the various ways in which we can model or create the description of a system or a digital block that we are trying design.

So, the topic of a lecture is Verilog description styles.

(Refer Slide Time: 01:05)

Description Styles in Verilog

- Two different styles of description:
 1. Data Flow
 - Continuous assignment *Using assignment statements.*
 2. Behavioral
 - Procedural assignment *Using procedural statements similar to a program in high-level language.*
 - Blocking
 - Non-blocking

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

So, broadly speaking, there are two different styles of description; I have already given you some examples of these. So, you have seen some examples, but broadly speaking the description styles can be categorized into two different types the first one is the so called data flow description style which uses continuous assignments using the assign statement that you have already seen in many examples. So, this uses such kind of assignment statements using assign. Now as an alternative, you can have behavioral design or description styles as well where you use some kind of procedural statements which are somewhat similar to a program in a high level language that too sometimes, right, you

see whenever you write a program in a high level language like C or any such language you try to specify; what exactly you are trying to do the different steps.

So, this is what you or this is how you specify what you are trying to do. So, in that sense, the behavioral model is somewhat similar. So, you use some kind of procedural statements; which is similar to the construct; which are available in a high level language; these will see and here in place of continuous assignment, we shall be using a different kind of assignment statement called procedural assignment which again is of 2 different types; one is called blocking assignment, other is called non blocking assignment.

So, over the next few lectures, we shall be discussing these different alternatives and variations and see that which one to use when and why, fine.

(Refer Slide Time: 03:08)

Data Flow Style: Continuous Assignment

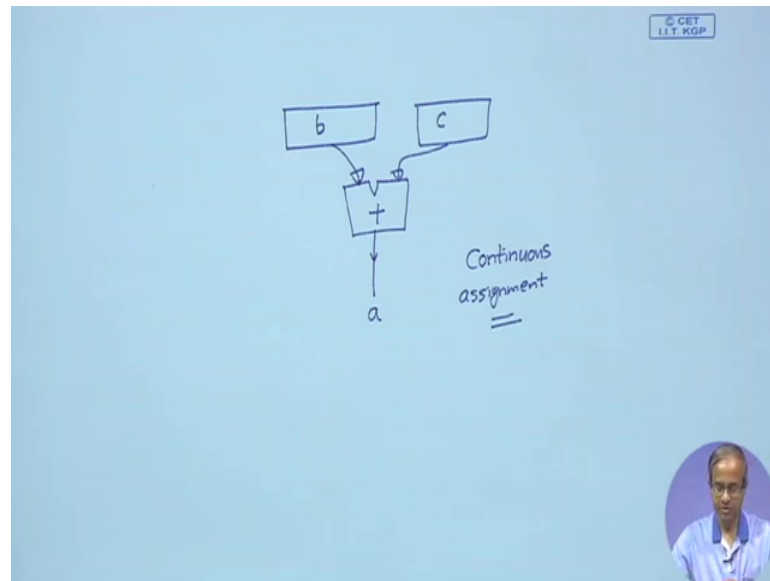
- Identified by the keyword “assign”.
- Forms a static binding between:
 - The “net” being assigned on the left-hand side (LHS).
 - The expression on the right-hand side (RHS), which may consist of both “net” and “register” type variables.
- The assignment is continuously active:
 - Almost exclusively used to model combinational circuits.
 - We shall also see some examples of modeling sequential circuit elements.

```
assign a = b + c;  
assign sig = Z[15];
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

So, let us start with continuous assignment or the data flow style which have already seen in sufficient detail, but just for the sake of completeness, let us also look at it once more. So, the data flow design style is identified by the keyword assign. So, whenever you want to make some assignment like a equal to b plus b or sin equal to Z 15; bit number 15 of a vector Z. We use this keyword assign before that now I mentioned earlier also whenever we use an assign this forms a static binding, it is not that whenever this statement is executed b plus is computed and the value is stored in a not like that whenever you write a equal to b plus c something like this will be happening.

(Refer Slide Time: 04:02)



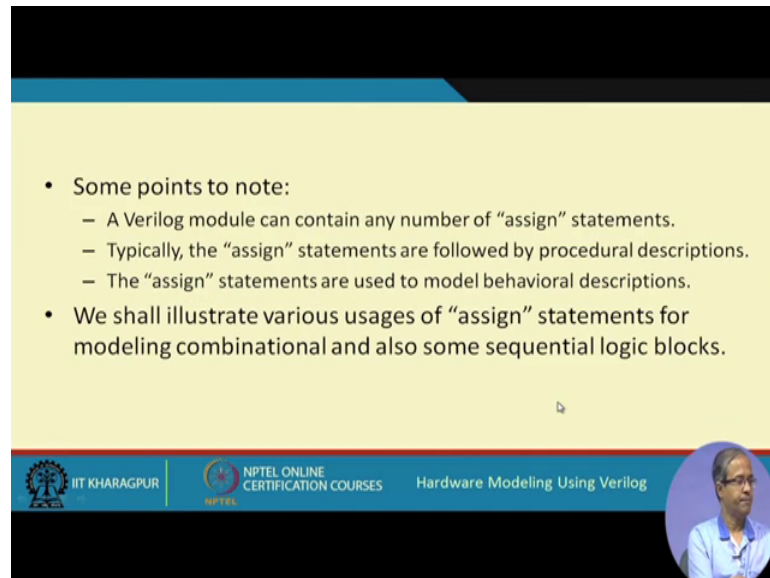
Let say b is a register, c is another register. So, there will be an adder which will be generated which will be adding b and c and the result will be generated or stored in a net which you are calling a. This a is not an registry just a net. So, the whenever either b or c will change, this a will change immediately after the delay of this adder. So, there is no concept of any clock or anything such an assignment we said earlier also is referred to as continuous assignment, right

So, there are some constraints in such assignment statements using assign first thing is that the left hand side must be your net type variable typically a wire, but; however, the expression that we are using on the right hand side here b plus c here $z = 15$; these expressions may consists of variables which can be either net or register or any combination of the 2 and this is called continuous assignment because this is continuously active; there is no signal that tells you that when this operation will be carried out and when the output will be stored in that target. So, it is not like that it is some kind of continuous assignment whenever one of the input changes the output net value will be changing or be affected right away after the necessary delay of the circuit, fine.

So, these kind of assign statements typically; we use it for modeling combinational circuits, but we shall see that we can also use this for modeling sequential circuit

elements which are of course, not very common, but we can also do this, we shall be taking some examples later, right.

(Refer Slide Time: 06:18)



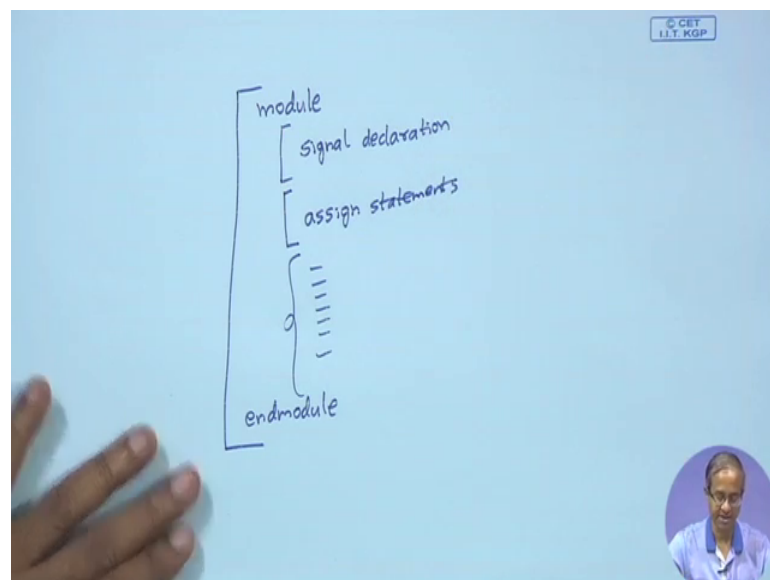
• Some points to note:

- A Verilog module can contain any number of “assign” statements.
- Typically, the “assign” statements are followed by procedural descriptions.
- The “assign” statements are used to model behavioral descriptions.

• We shall illustrate various usages of “assign” statements for modeling combinational and also some sequential logic blocks.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

(Refer Slide Time: 06:36)



© CET IIT KGP

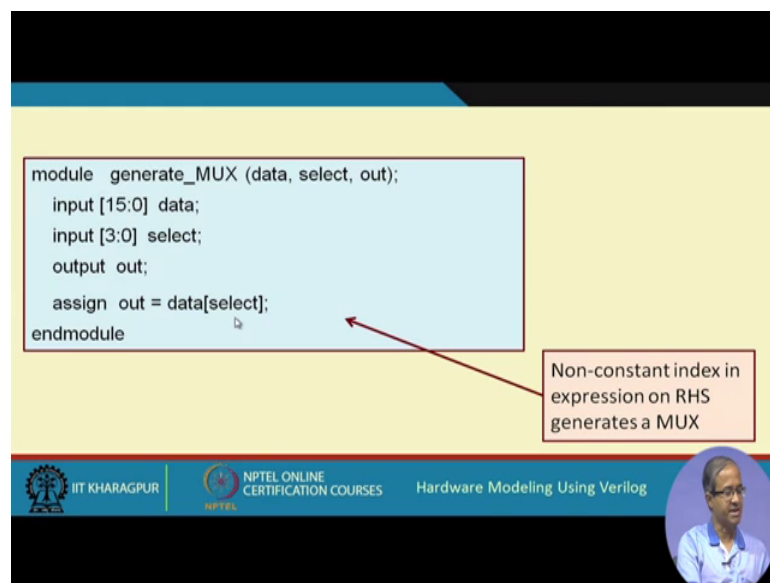
module
[signal declaration
assign statements
]
endmodule

Now in a Verilog module, we can have any number of assignment statements assign statements and this is a very typical structure. So, whenever we write a Verilog module let us say; this is a Verilog module, it starts with the keyword modulem it ends with end module.

So, in the beginning, we typically start with the signal declarations all the net and the registered variables are different there, then after that we usually club all the assign statements together, this is the normal practice and then the other kind of statements will come which you shall see later right this is how a typical Verilog module structure look like where the assign statements are typically placed in the beginning, right after the signal declarations. And another thing I also mentioned that in the assign statement, we just tell or say what we are trying to do suppose, you write a equal to b plus we are just saying that we want to add b plus b and c and we want to store the result back in a, but we are not telling what kind of adder to use how to add and so on. So, this is more like a behavioral description, we are specifying the behavior, but not the exact structure of the circuit.

So, now we shall see a number of examples; typical usages and while we are going through the examples, we shall also learn about a number of things there.

(Refer Slide Time: 08:20)



```
module generate_MUX (data, select, out);
  input [15:0] data;
  input [3:0] select;
  output out;
  assign out = data[select];
endmodule
```

Non-constant index in expression on RHS generates a MUX

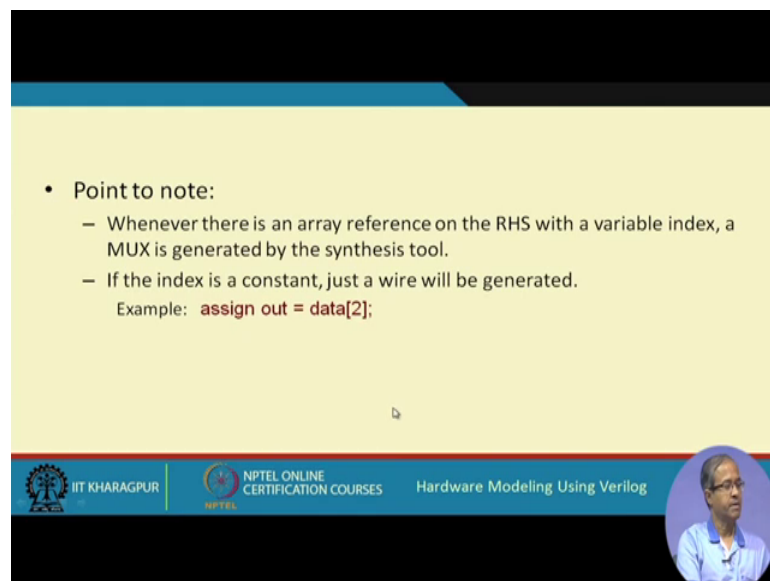
The slide features a light blue background with a white box containing the Verilog code. A red arrow points from a callout box on the right to the `data[select]` expression in the code. The footer includes the IIT Kharagpur logo, NPTEL Online Certification Courses logo, the text 'Hardware Modeling Using Verilog', and a circular portrait of a man in a blue shirt.

Fine, this is one very simple example; we saw an example of multiplexes earlier. So, here we are generating a MUX like whenever we have a module description like this. Now see now right here, we are not talking exactly about simulation, we are talking about what will happen when let us say we are using a synthesis tool to synthesize our circuit. So, for that purpose; what will happen how this synthesis tool will be carrying out synthesis and what kind of hardware blocks or modules will be generated.

So, let us see his example. So, in this example you see that we are using a module called generate MUX which as 3 parameters the input is a 16 bit quantity it is a 16 to one MUX 4 bit select line and a single bit output. So, we are simply using a single line description which says assign out equal to data of select. So, data is like a vector 16 bit vector select is a number 4 bit number which selects one of the bits and that particular bit goes to out. So, which is what MUXs.

Now whenever the Verilog synthesizer encounters such a description a multiplexer will be generated. So, how does this synthesizer know that a multiplexer will be required to be generated the keyword is a non constant index in the expression on the right hand side. So, whenever there is a vector which I am using with the non constant index select is a variable right it is not a constant. So, the index value is a variable then we will be generating a multiplexer. So, the data whatever is there that will be the input to the multiplexer and this variable will go to the select line and this out will be the output point.


(Refer Slide Time: 10:40)



• Point to note:

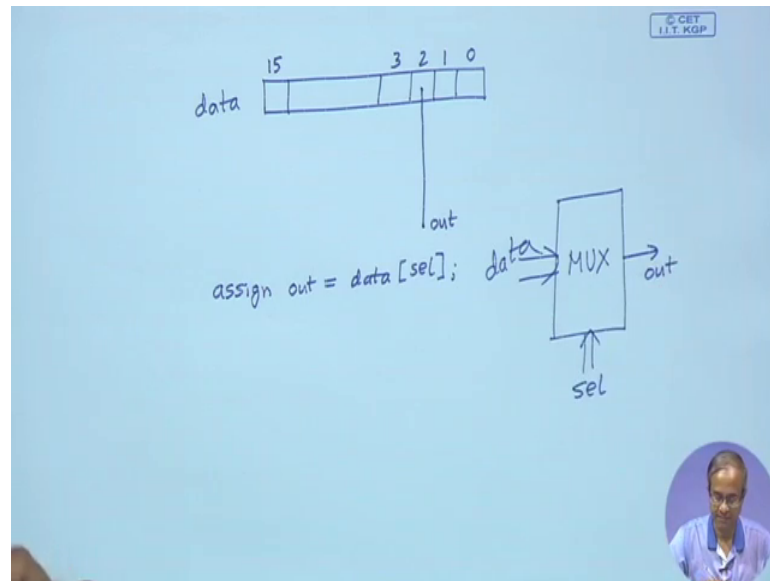
- Whenever there is an array reference on the RHS with a variable index, a MUX is generated by the synthesis tool.
- If the index is a constant, just a wire will be generated.
Example: `assign out = data[2];`

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog



Now, the point to note is that like as I had said with example, I have shown that whenever there is a array reference with the variable index, the multiplexer will be generated, but; however, if on the right side, it is not a variable, it is a constant, let us say like this assign out equal to data 2, then, no multiplex will be generated because you see data is nothing but an vector. Vector means a collection of bits.

(Refer Slide Time: 11:17)



So, it is a 16 bit vector. So, the index is the index values will go from 0 up to 15.

So, when we say assign out equal to data 2, what will happen is the data 2 is this bit; this bit will straight away be taken out and this will be called as out. So, no circuit will be generated, it will be just wiring; this particular bit will be connected or it will be given a name out; this will be what will happen if I give an assignment like this; no gates or no hardware blocks will be generated for an assignment like this, but if I write like in the previous example, assign out equal to data, but I give a variable sel, here I do not know which bit I want. So, here you need to synthesize or generate a multiplexer where the input will be your data your; this will be your select lines and on the output side you will be getting the output bit out right.

(Refer Slide Time: 12:38)

```
module generate_set_of_MUX (a, b, f, sel);  
  input [0:3] a, b;  
  input sel;  
  output [0:3] f;  
  assign f = sel ? a : b;  
endmodule
```

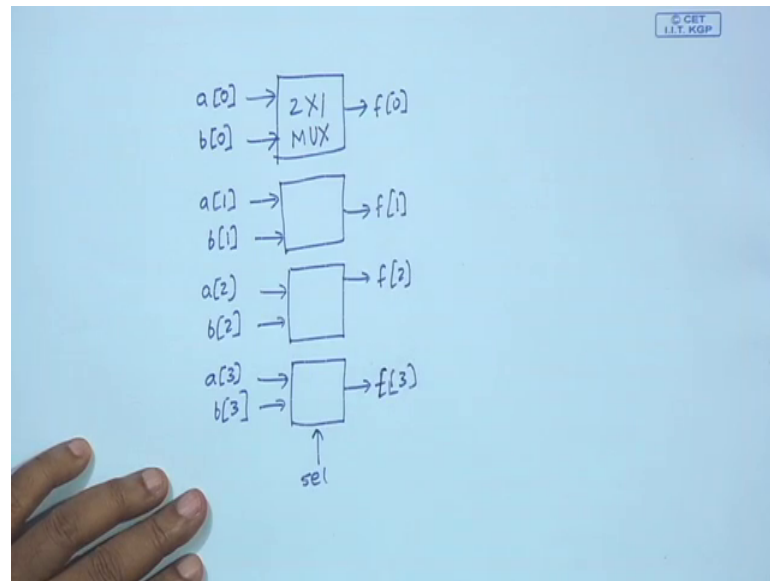
Conditional operator generates a MUX

The slide features a light blue background with a white box containing the Verilog code. A red arrow points from a callout box on the right to the conditional operator in the code. The footer includes the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and the text 'Hardware Modeling Using Verilog' next to a circular portrait of a man in a blue shirt.

So, let us take another example. Here this is also an example for multiplexer, but we are using a conditional operator here. So, what you are doing? Let us try to understand this a and b are 2-4 bit vectors. So, the index goes from 0 to 3 in this order, we have sel, similarly f is another 4 bit vector which is the output and sel is an input and in this conditional operator we are saying that if sel is true sel is a one bit vector if this is true means it is 1, then you select a else you select b, select a, means a will be assigned to f otherwise b will be assigned to f.

So, actually what will happen here? This will be generating a since every conditional operator will be generating a multiplexer because it is f, then else, but here, this a and b are both 4 bit vectors and f is also a 4 bit vector, you see whenever you have a conditional thing; what you are doing you are checking for a condition, if the conditional is true, you are selecting one, if the condition is false, you are selecting the other. So, what you are actually generating is a 2 to 1 MUX.

(Refer Slide Time: 14:00)



You will be generating a 2 to 1 multiplexer, but here what has happened in the example that I have given both a and b and f, they are all 4 bit quantities. So, actually it will not be 1 2 to 1 multiplexer, but 4 such multiplexers will be generated.

So, each of them will be having 2 inputs and 1 output; there will having a common select line Sel. This will be connected in parallel to all of them. Now in the inputs will be connecting for the first one a 0 and b 0, the second will be connected a 1 and b 1 third one will be connecting a 2 and b 2 and fourth one; we are connected a 3 and b 3 and in the output side, this will go to f 0, this will go to f 1, this will go to f 2 and this will go to f 3; f 3. So, you see a statement like that using conditional automatically generates an array of multiplexers like this and this multiplexers will all be 2 to 1 multiplexers, right.

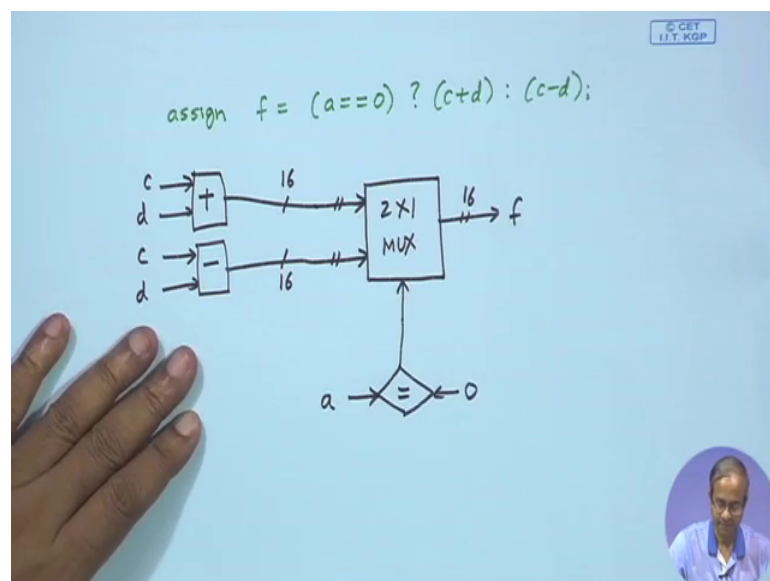
(Refer Slide Time: 15:38)

- Point to note:
 - Whenever a conditional is encountered in the RHS of an expression, a 2-to-1 MUX is generated.
 - In the previous example, since the variables “a”, “b” and “f” are vectors, an array of 2-to-1 MUX-es are generated.
 - What hardware will be generated by the following?
`assign f = (a==0) ? (c+d) : (c-d);`

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

So, here the point to observe is that. So, whenever you have a conditional expression like the one we have seen we have shown in the right hand side; automatically a 2 to 1 multiplexer will be generated. Now in the example which you are shown because a, b and f vectors; so, it was not one, but an array of 4 such multiplexers where generated. Now let us take an example, suppose I have given an assignment statement like this. So, what will be the hardware; this assignment statement will be generated. Let us try to work this out.

(Refer Slide Time: 16:15)



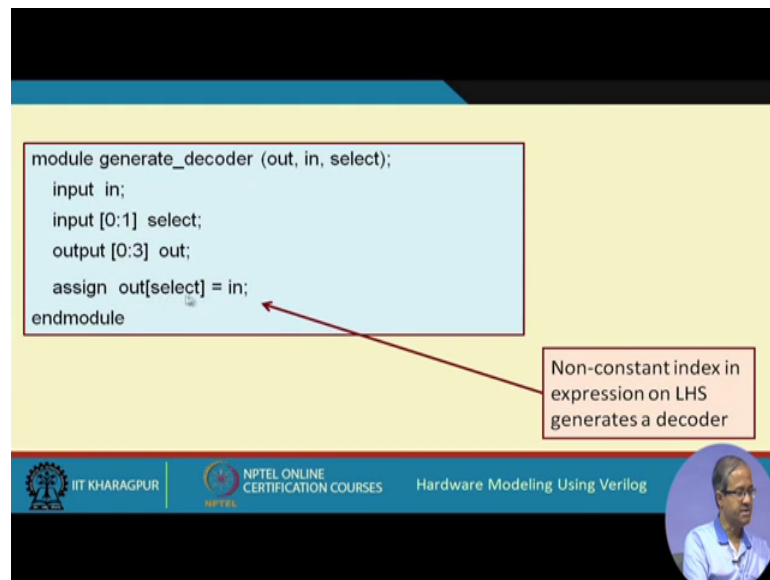
So, our statement that we have seen is assign f equal to this is a conditional. So, the condition checking that we are doing is a equality 0, if it is true, c plus d , if it is false c minus d .

So, this is what we are actually trying to do. Now let us see; so, here what will happen here? There will be a multiplexer which I will be; I am just showing one multiplexer, but actually; it will be an not one, but several multiplexers; I am using the array notation; 2 to one multiplexers using the array or vector the notation. So, it will be having 2 inputs, these are not single lines, but multiple lines. Similarly, it will be having an output not one multiple.

So, in the select line, we will be comparing a with 0. So, there will be a magnitude comparator. So, I need a magnitude comparator circuit which will be taking a as input, it will be comparing it with the number 0 and the result of the comparison will be fed to the select line. So, either this will be true or this will be false and on the other side, I use an adder where I have fed the numbers c and D this sum, I am connecting here and I also have a subtractor where again I have fed the number c and d , this I am connecting here.

So, this will be the kind of circuit that will be generated. Now let us say if this adders, let us say this is all 16 bit operation, this is number SeL , all 16 bits, then actually, this is a vector kind of a notation the output is f . So, actually it will be not one multiplexer, but 16 such multiplexers will be generated one for every bit of this data they are 16 bits. So, 16 bits have to be multiplexed. So, I need 16 2 to 1 multiplexers, right.

(Refer Slide Time: 18:33)



```
module generate_decoder (out, in, select);
  input in;
  input [0:1] select;
  output [0:3] out;
  assign out[select] = in;
endmodule
```

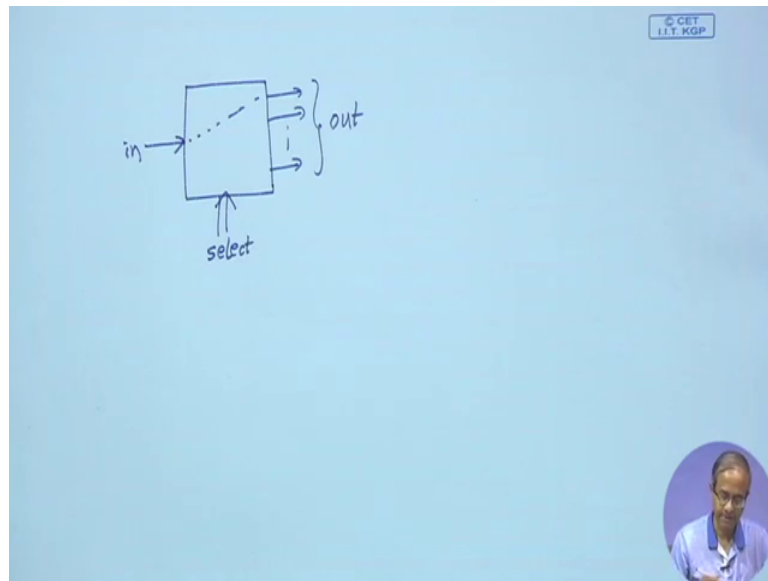
Non-constant index in expression on LHS generates a decoder

The slide shows a Verilog code snippet for a decoder. The code is enclosed in a light blue box. A red arrow points from a callout box on the right to the line 'assign out[select] = in;'. The callout box contains the text 'Non-constant index in expression on LHS generates a decoder'. The slide also features logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and the course title 'Hardware Modeling Using Verilog' at the bottom. A small circular portrait of a man is visible in the bottom right corner.

So, let us take another example. This is the example of a decoder means this is an example which will generating a decoder. So, here you see what you have given; here we have one input in is a single bit input, select is 2 bit vector and out is a 4 bit vector and what we have written a something like this out with in index of variable equal to in. So, now, the array access with the variable index is appearing on the left hand side. So, whenever such a thing happens a decoder will be synthesized.

So, if we have on the left hand side non constant index select, then a decoder will be generated. So, what the decoder will do the input value will be going to one of the outputs depending on select.

(Refer Slide Time: 19:34)



So, now your circuit will be something like this. You have a decoder or a de multiplexer whatever you call the input will be in single bit data, but in the output, there will be several lines, this will be your out and this input will be connected to one of the outputs and which one that will be determined by the select line select. So, in a assignment statement, if there is on the left hand side an expression like this out with the variable in the index then a decoder will be generated.

(Refer Slide Time: 20:19)

- Point to note:
 - A constant index in the expression on the LHS will not generate a decoder.
 - Example: `assign out[5] = in;`
This will simply generate a wire connection.
 - As a rule of thumb, whenever the synthesis tool detects a variable index in the LHS, a decoder is generated.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

Now, again if instead of a variable, there is a constant, this will again generate a simple wire connection, this in whatever is in it will directly be connected to the fifth bit of the vector out. So, there will be no circuit that will be generated just a wire. So, as I had said whenever the synthesis tool will find out a variable index in the left hand side, a decoder is generated and if it finds a variable index in the right hand side, a multiplexer will be generated. So, these are some simple rules which the synthesis tool often uses or utilizes, fine.

(Refer Slide Time: 21:02)

The slide displays a Verilog module definition for a level-sensitive latch. The code is as follows:

```

module level_sensitive_latch (D, Q, En);
  input D, En;
  output Q;
  assign Q = En ? D : Q;
endmodule

```

To the right of the code is a truth table:

En	D	Q _n
0	x	Q _{n-1}
1	0	0
1	1	1

Below the truth table, it says "Generates a D-type latch". A text box on the right explains: "Here is an example to describe a sequential logic element using 'assign' statement." An arrow points from this text box to the 'assign' statement in the code.

At the bottom of the slide, there are logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with the text "Hardware Modeling Using Verilog" and the number "11".

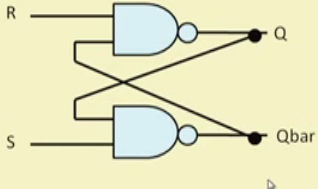
Now, here we take an example where we are using an assign statement to describe a sequential logic element which is actually a level sensitive day type latch. So, what does this do? this you will be capital. So, what is this description doing? There are 3 parameters D and enable at the inputs and Q is the output. So, what is this assignments statement doing, it is saying if enable is true; that means, it is 1, then D will be going to Q otherwise if enable is 0, Q will be going to Q, Q will be going to Q means it will be remembering its previous state there will no change.

So, this Q will be going to Q, this kind of assignment whenever it is there. So, the synthesizer will immediately deduce the 12, I need to generate a latch or a memory element because I have to memorize some data value. So, as this example says that for certain condition of the enable when enable is 0, I have to remember the value of Q because the output Q will be equal to the old value of Q.

So, this will correspond to the state table like this. So, when enable is 0, this Q will be selected. So, D will be do not care irrespective of D; whatever was the previous value of Q, I am denoting as Q_{n-1} that will be the new value of Q, I am calling it Q_n , but if enable is 1, then whatever is the value of D that will be copied to Q.

(Refer Slide Time: 23:01)

• Modeling a simple S-R latch:



S	R	Q_n
1	1	Q_{n-1}
0	1	0
1	0	1
0	0	?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

So, this single assign statement will be generated a D type latch. Similarly, if you look at a simple R S or S R type latch using a cross coupled gates, here I am showing NAND gates; cross coupled NAND gates. So, a circuit like this which is 2 input R and S and Q and Q bar. So, if you again look at this state table what will happen if both S and R are one and one then whatever is Q bar Q and Q bar, let us say Q is 1 and Q bar is 0, if this is 1; 1 will come 1 and 1 will be 0 and 0 and 1 will be 1; so no change similarly for the rivers.

So, when the inputs are 1 and 1 the previous input value will remain this is the behavior of this circuit, but if is S equal to 0 and R equal to 1, S equal to 0 will force this output to be become 1. So, Q bar will be 1 and Q will be 0 because if Q bar is 1, this one and R equal to 1 will make the output of NAND gate 0. So, Q will be 0 similarly if S is 1 and R is 0, the reverse will happen if R is 0, Q will be one and this 1 and 1 will make this 0 and hence this is 1, but 0 is an indeterminate state if we apply 0 and 0 then you see these are NAND gates; both output will become 1 and 1 which is inconsistent with Q and Q bar not only that after that if I apply 1; 1; what S Q_{n-1} that will be confusing to the


hardware and also to the simulator because this is not a validate state both Q and Q bar is 1.

So, which one will become 0 that will depend on the relative delays of this 2 gates which one is faster that will force the output to it to change first that is called race condition, fine. So, this circuit the exactly the way I have shown here; you can use 2 assign statement in one assignment; you are using the NAND of Q bar and R to generated Q and the other 1 Q and S to generate Q bar.


(Refer Slide Time: 25:23)

```
module sr_latch (Q, Qbar, S, R);
    input S, R;
    output Q, Qbar;
    assign Q = ~(R & Qbar);
    assign Qbar = ~(S & Q);
endmodule

module latchtest;
    reg S, R; wire Q, Qbar;
    sr_latch LAT (Q, Qbar, S, R);
    initial
    begin
        $monitor ($time, "S=%b R=%b, Q=%b, Qbar=%b",
            S, R, Q, Qbar);
        S = 1'b0; R = 1'b1;
        #5 S = 1'b1; R = 1'b1;
        #5 S = 1'b1; R = 1'b0;
        #5 S = 1'b1; R = 1'b1;
        #5 S = 1'b0; R = 1'b0;
        #5 S = 1'b1; R = 1'b1;
    end
endmodule
```




IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

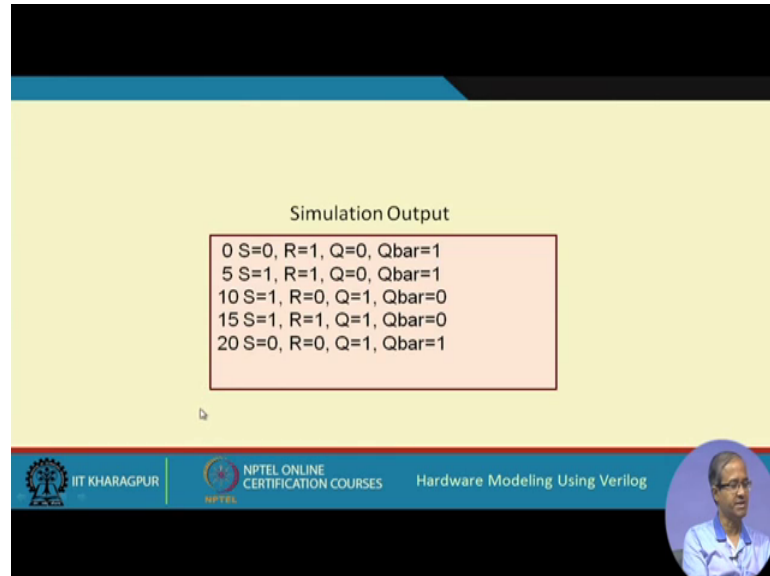


So, just like this 2 just 2 assignment statements this is NAND and the NOT R and Q bar, this is Q and S and Q, this is Q bar just to assign statement to be generating 2 NAND functions and this will be specifying the behavior of this latch.

Now, just for the purpose of testing we also wrote a sample test bench for this to check whether this latch is actually working like a latch should work. We wrote a test bench like this, hope where this S R latch was instantiated we call it lat. So, this S and R which were the inputs were declared as latch and Q and Q bar, the outputs were declared as wire and we monitored all the variables S R Q and Q bar. So, what we did initially, we said the input S R; R to 0 and 1. So, if the inputs are 0 and 1 the output should be 0, then we said the inputs to 1 and 1. So, that the output should not change, then you change to the other state 1 and 0. So, the outcome should become Q should become 1, then 1 and 1.

Then just for testing we applied the invalid condition 0, 0 followed by 1, 1 to see that what this simulator does here.

(Refer Slide Time: 26:53)



So, what this simulator did was something like this; first few lines was absolutely perfect what we expected S 0, R 1 generates Q 0 and Q bar 1, then we apply both 1 1. So, the same state remains no change, then we apply 1 and 0. So, Q becomes 1 and Q bar becomes 0, then 1 1 same state no change then we apply S equal to 0 R equal to 0 which is 1 and 1, but after that we again 1 1, but the problem is that the simulator got confused and the simulator hangs; it continuously try to deduce that what will happen when S equal to R equal to 1, but because of the race condition the output never converges. So, simulator assumes that the 2 gates have exactly equal delays.

So, when the outputs are 1 and 1 and we apply 1; 1 outputs will become 0 and 0 next state; it will again become 1 on 1 next state, it will again become 0 and 0. So, both the outputs will oscillate in definitely, it will never converge and the simulator will go on trying to compute the output value indefinitely. So, this was exactly what was happening here, right. So, crux is that you should not apply such invalid inputs to an S R flip flop in actual circuit operation.

So, with this, we come to the end of this lecture; where we have actually looked at some of the modeling styles. In particularly; looked at the assign, again we took some examples of assign; earlier also just work out a number of examples. So, we again saw

assign here and we saw that not only combination circuits, we can also model sequential circuit elements whatever circuits I can draw using gates I can also implement that using assigned statements like you know; how to design flip flops I just show the example of a cross coupled NAND gates you can design a j k flip flop, S R flip flop, t flip flop, all using gates NAND gates NOR gates, then H triggered flip flop. So, once you have a gate level, circuit diagram using assign statement or using instantiation of the gates, you can create those designs. So, those are also some ways to create sequential circuit elements in Verilog.

So, in the next lecture, we shall be continuing with some more Verilog constructs and examples.

Thank you.