

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

**Lecture - 08**  
**Verilog Language Features (Part 3)**

So, you know earlier lectures, we have seen some examples where you have tried to illustrate some structural design constructs by using instantiation of gates. So, in the present lecture, we shall first see what are the various types of gates that are available as part of the Verilog language which you can directly use and instantiate in our design.

(Refer Slide Time: 00:50)





(Refer Slide Time: 00:53)

### Predefined Logic Gates in Verilog

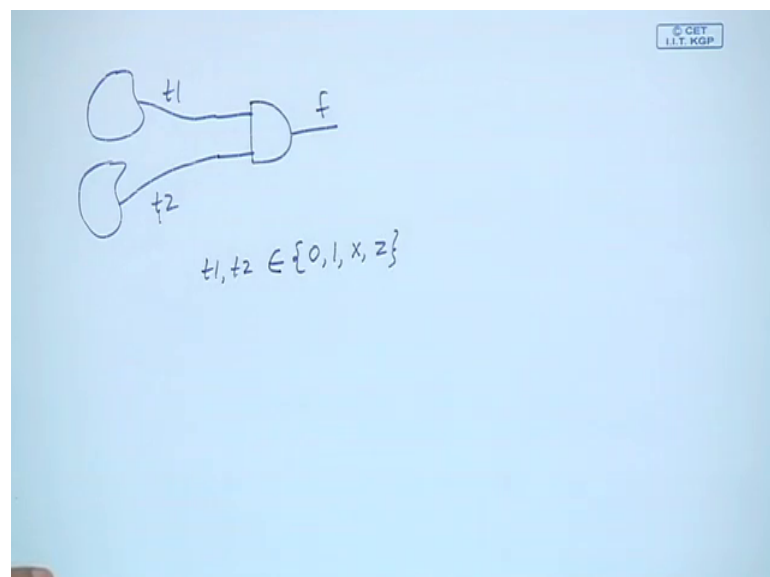
- Verilog provides a set of predefined logic gates.
  - Can be instantiated within a module to create a structured design.
  - The gates respond to logic values (0, 1, x or z) in a logical way.

2-input AND	2-input OR	2-input EXOR
$0 \& 0 = 0$	$0   0 = 0$	$0 \wedge 0 = 0$
$0 \& 1 = 0$	$0   1 = 1$	$0 \wedge 1 = 1$
$1 \& 1 = 1$	$1   1 = 1$	$1 \wedge 1 = 0$
$1 \& x = x$	$1   x = 1$	$1 \wedge x = x$
$0 \& x = 0$	$0   x = x$	$0 \wedge x = x$
$1 \& z = x$	$1   z = x$	$1 \wedge z = x$
$z \& x = x$	$z   x = x$	$z \wedge x = x$

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

So, this is the title of our lecture and we start with our discussion on predefined logic gates in Verilog. So, we have already seen some of these gates AND, OR, NOT, NAND, NOR, EXOR. Verilog provides predefined logic gates as I said and we have also talked about earlier that in Verilog will support 4 logic values step; 0, 1, undefined and high impedance.

(Refer Slide Time: 01:34)



Now when you talk about gates for example, I use a NAND gate. So, the inputs of the NAND gates are coming from say to other hardware blocks; let us call this  $t_1$ ; let us call this  $t_2$ . Now this  $t_1$  and  $t_2$ ; they will be belonging to this set 0, 1, X or z.

So, now when I say that this is an and gate, I also have to define; what is the behavior of this and gate, if some arbitrary input combinations are coming on  $t_1$  and  $t_2$ ; what should be the value of  $f$ . This actually follows from intuition; let us see; let us look at an and gate from the definition of an and gate 0 and 0 is 0 well if 1 of the input is 0, other is 1; it is also 0 when both the inputs are 1, then it is 1.

Now let us look at the behavior with X 1 of the input at X and the other input is 0 or 1. So, if 1 input is X and the other is 1 for and operation you cannot say what is the output, but if the one input is 0, then you can definitely say that the output will be 0. Similarly for the high impedance state, some other things see if one of them is one other is the high impedance state, then the output can be an indeterminate level, I call it X and if one is z other is X output is also x.

So, for OR operation, it is again similar 0, 0 is 0, 0, 1 is 1, 1, 1 is 1 and for do not care if 1 of the input is 1, then the output is definitely 1, but for 0, you cannot say for this case is again X for X or similarly 0, 0 is 0, 0, 1 is 1, 1, 1 is 0 and for other cases you really cannot say for XOR function, if one of the input is X or z the output will be undefined. So, for all the gates you can define the behavior in terms of the 4 valued logic 0 1 X z like this.

(Refer Slide Time: 03:54)

**List of Primitive Gates**

<code>and G (out, in1, in2);</code>	<code>bufif1 G (out, in, ctrl);</code>
<code>nand G (out, in1, in2);</code>	<code>bufif0 G (out, in, ctrl);</code>
<code>or G (out, in1, in2);</code>	<code>notif0 G (out, in, ctrl);</code>
<code>nor G (out, in1, in2);</code>	<code>notif1 G (out, in, ctrl);</code>
<code>xor G (out, in1, in2);</code>	
<code>xnor G (out, in1, in2);</code>	
<code>not G (out, in);</code>	
<code>buf G (out, in);</code>	

There are gates with tristate controls

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

Now, talking about the list of gates this is the complete list the basic gates are this and, nand, or, nor, xor, xnor, not, there are buffers and there are some couple of more not gates. So, I will explain this for the normal and, nand, or, nor, xor, or, xnor, the number of inputs can be arbitrary like for example, for and I can write.

(Refer Slide Time: 04:35)

`and G (a, b, c, d, e, f);`  
output: a  
inputs: b, c, d, e, f

`not G (out, in);`

`bufif1 G (out, in, ctrl);`  
if `ctrl=1`, `out=in`  
if `ctrl=0`, `out=Z`

`bufif0 G (out, in, ctrl);`  
if `ctrl=0`, `out=in`  
if `ctrl=1`, `out=Z`

`notif1 G (out, in, ctrl);`

`notif0 G (out, in, ctrl);`

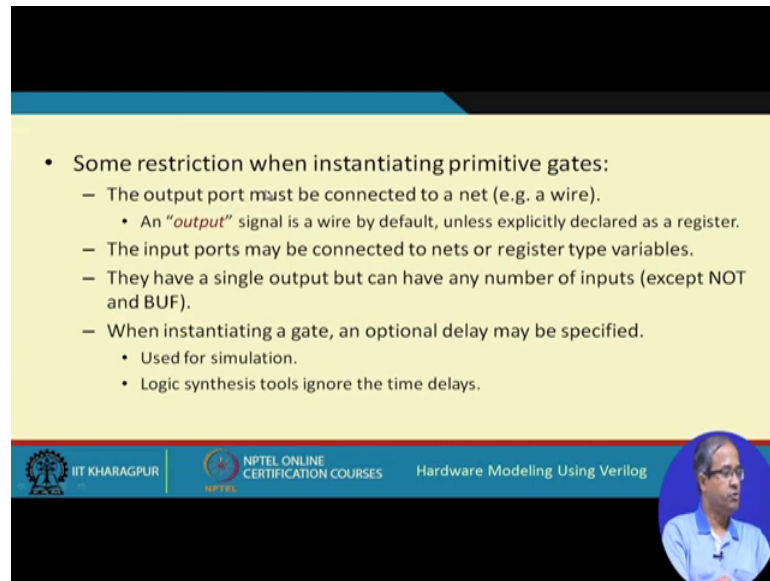
An instantiation and gate is G. So, I can write like this a, b, c, d, e, f, this will mean that the first variable that I am using here this will be my gate output and all the remaining variable there are 5 here this will be my inputs. So, actually I am defining a 5 input and

gate well a is the output and b, c, d, e and f are the inputs. So, this way depends on how many variables I am using in the parameter. So, the number of inputs of the gate will be defined accordingly, right.

So, there is a flexibility not is simple not will have a single input and a single output. So, the input is in output is out come to buf; buf is just a buffer sometimes you need to means isolate signals this out equal to in the logic value does not change, but the signal values are restored, this is a buffer now there are some tri state version of buffers and not buf if 1; let us say let us see this first buf, if one is something like this it is a buffer which is selected by a control signal like this. So, this is c t r l this is in this is out. So, the behavior is if your control is one then out will be equal to in if control is 0 then output will be tri state it will be between the high impedance state right this is your buf if 1 this gate.

Similarly, you have another gate called buf if 0; buf if 0 is similar the only difference is that the polarity of the control is different there is a negation here. So, it is just the reverse if control equal to 0 then out equal to in if control equal to 1 out equal to high impedance state this is buf if 0. Similarly there are equivalent versions for not so very similar, I am just showing the diagram and not writing expression instead of the buffer this is be the not. So, the output will be in bar and there will be a control this is not if one and there is another version not with the control inverse this is not if 0. So, this kind of tri state control gates are also available if you want to use it in a design right.


(Refer Slide Time: 08:31)



• Some restriction when instantiating primitive gates:

- The output port must be connected to a net (e.g. a wire).
  - An “output” signal is a wire by default, unless explicitly declared as a register.
- The input ports may be connected to nets or register type variables.
- They have a single output but can have any number of inputs (except NOT and BUF).
- When instantiating a gate, an optional delay may be specified.
  - Used for simulation.
  - Logic synthesis tools ignore the time delays.

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

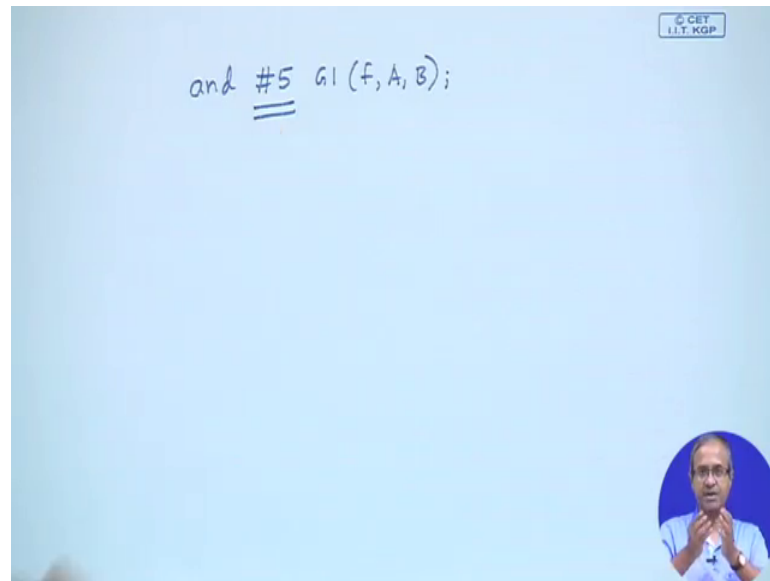


When you are instantiating these kind of primary gates or primitive gates there are some restrictions you need to remember like for example, here the outputs the first argument or the parameter is the output the output port must always be connected to a net it cannot be connected to a register.

So, an output signal in a module is a wire by default. So, you can directly use it if you want now the input ports of the gate like for example, this n 1 and n 2 these can be anything they can be connected to either nets or register type variables this I have already said that they have a single output, but can have any number of inputs the basic gates except not and buffer well and I have seen in the examples earlier that when you instantiate a gate; you can specify an optional delay like suppose you specify an and you can write and hash 5 G 1 f a b like this.

Now this delay is again only for simulation purpose for synthesis purpose this delay does not make any sense right because it is ultimately the hardware and gate how fast or how slow it is that will determine the delay. So, as a designer I have no idea what the delay will be how many picoseconds or how many nanoseconds, right.

(Refer Slide Time: 09:45)



So, this optional delay this is used only for simulation and then logic synthesis tool will ignore these delays.

(Refer Slide Time: 10:29)

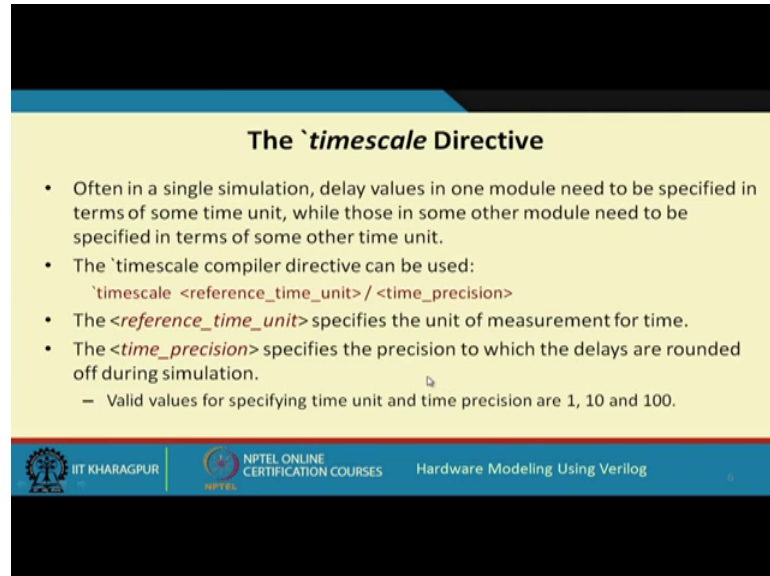
```
timescale 10ns/1ns
module exclusive_or (f, a, b);
input a, b;
output f;
wire t1, t2, t3;
nand #5 m1 (t1, a, b);
and #5 m2 (t2, a, t1);
and #5 m3 (t3, t1, b);
nor #5 m4 (f, t2, t3);
endmodule
```

The diagram illustrates the implementation of an exclusive-or function using four gates: a NAND gate (m1) with inputs a and b, followed by two AND gates (m2 and m3) and a NOR gate (m4). The intermediate signals are t1, t2, and t3, and the final output is f.

So, let us take an exam which shows these delays. So, here we have a structural representation, well note the first line for the time being, I will explain this what does this mean this is an exclusive or function realization using nand gates; nand and nor gates. So, using 4 gates, I can implement this exclusive here there are 3 wires t 1 t 2 and t 3. So, we have used this, this M 1 is a nand, M 2 is an and M 3 and M 4. Here we have

all specified delays as 5 as I said this will be used only for simulation purposes now the first line time scale.

(Refer Slide Time: 11:26)



The slide is titled "The `timescale Directive" and contains the following text:

- Often in a single simulation, delay values in one module need to be specified in terms of some time unit, while those in some other module need to be specified in terms of some other time unit.
- The `timescale compiler directive can be used:  
``timescale <reference_time_unit> / <time_precision>`
- The <reference\_time\_unit> specifies the unit of measurement for time.
- The <time\_precision> specifies the precision to which the delays are rounded off during simulation.
  - Valid values for specifying time unit and time precision are 1, 10 and 100.

The slide footer includes the IIT KHARAGPUR logo, NPTEL ONLINE CERTIFICATION COURSES logo, and the text "Hardware Modeling Using Verilog".

Time scale actually tells you what these numbers 5 mean you see roughly speaking the first number here ten nanosecond this gives you the basic unit of time. So, when I write 5 it will actually mean 50 nanoseconds and 1 nanosecond will be the precision of simulation. So, these are explained here in some detail. So, when you use the time scale directive you can specify these 2 times now why you need this you need this for simulation purposes that is the first thing. Now with respect to some module you may have to specify some delay values, but some, but for some other module the deliver may be different.

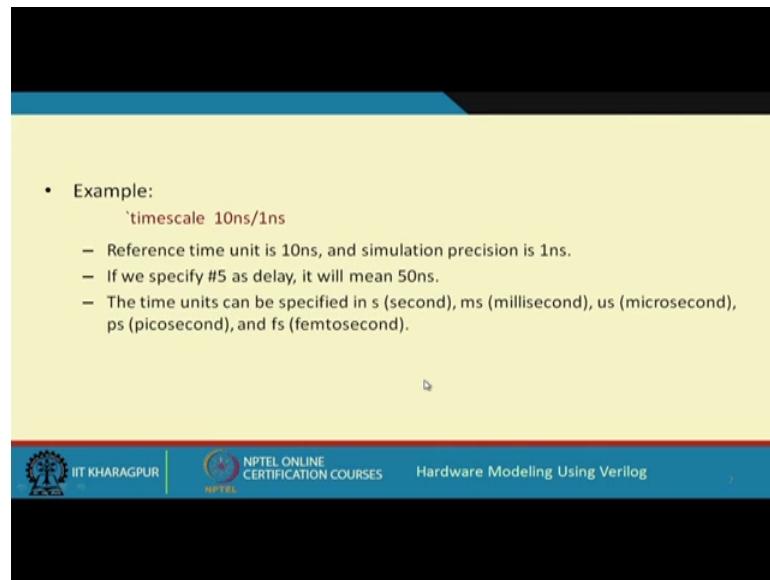
So, it is always good to have this kind of a declaration time scale declaration at the beginning of every module. So, if the time scale values are different from one module to the other you can have different declarations like that. So, the syntax of the time scale command or directive is this reverse scope time scale you specify reference time units slash time precision the reference time unit is actually used to specify the unit of measurement of time as I had said. So, whatever here you specify 5 this 10 nanosecond is the unit 5 in to 10, right.

And time precision the second one here in case we mentioned 1 nanosecond, it specifies the precision with which we round off the delays during simulation; that means, how



accurate the simulation is this simulation will be accurate up to 1 nanosecond that is why you have given a one nanosecond and for this times the valid values you can give only 1 10 and 100 nothing, you cannot write 2 3 5 here this values can be 1, 10 and 100 only and this units can be of course, different.

(Refer Slide Time: 13:51)



• Example:

```
`timescale 10ns/1ns
```

- Reference time unit is 10ns, and simulation precision is 1ns.
- If we specify #5 as delay, it will mean 50ns.
- The time units can be specified in s (second), ms (millisecond), us (microsecond), ps (picosecond), and fs (femtosecond).

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

This has I said this was the time scale we give in that example. So, when we write hash 5 it means a delay of 50 nanosecond. Now instead of nanosecond, I can use other units also I can use s meaning second M s millisecond, then I can write u s microsecond p s picosecond or f s femtosecond this can be specified, but again these are all for simulation this has nothing to do with the actual delay of the circuits these are just very rough estimates which the designer has given for the purpose of simulation.

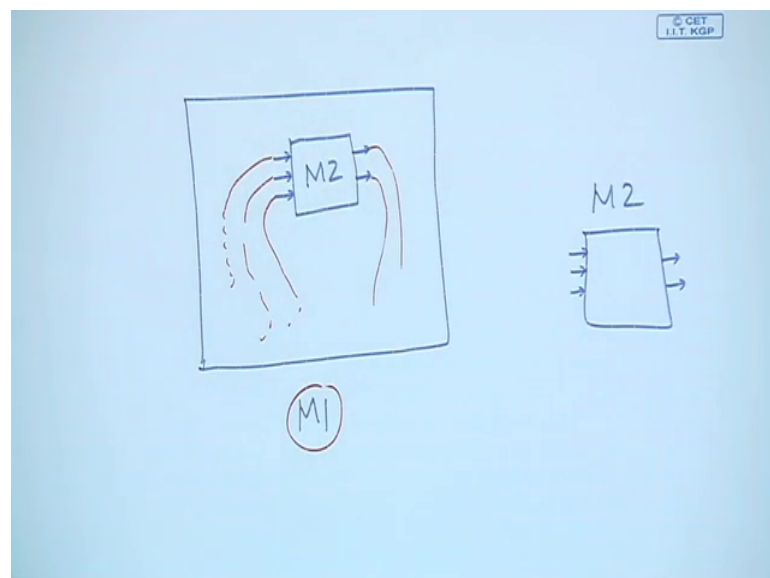
(Refer Slide Time: 14:38)

### Specifying Connectivity during Instantiation

- When a module is instantiated within another module, there are two ways to specify the connectivity of the signal lines between the two modules.
  - Positional association
    - The parameters of the module being instantiated are listed in the same order as in the original module description.
  - Explicit association
    - The parameters of the module being instantiated are listed in arbitrary order.
    - Chance of errors is less.

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

(Refer Slide Time: 14:49)





Now, the question comes how to specify connectivity that when you are instantiating a module within another module like what I mean is that suppose I have a module, this is M 1 and I have another module M 2 and I am instantiating this module M 2 inside here. So, M 2 had some input signals M 2 had some output signals. So, here also the same input and output signals will be there. So, I will have to suitably connect this input and output signal from the other means hardware components that are there in M 1 so that this instantiation will work in a proper way, right.

So, there are 2 ways to specify this connectivity how these input and output lines are connected. The first is called positional association where the parameters of the modules that is being instantiated are listed in the same order as in the original module description. So, in the original module description let us say we are defining a full adder. So, if it was sum carry a, b, c, then when you are instantiating we have to specify in the same order sum carry a, b, c.

So, we cannot change the order, but there is an alternate method called explicit association where we can change the order we can specify the parameters in an arbitrary order, but we shall see how I mean arbitrary order the advantage is that here the chance of error is less because suppose, when you wrote the module for the full adder you give the first 2 parameters sum and carry, but when you are instantiating by mistake you have written carry and sum. So, in simulation you will see that result is not coming correctly. So, you will have to find out where the error is there you have to debug it, but if you use this explicit association then this kind of errors the chance of occurring will be much less we shall see how we can use this.

(Refer Slide Time: 17:12)

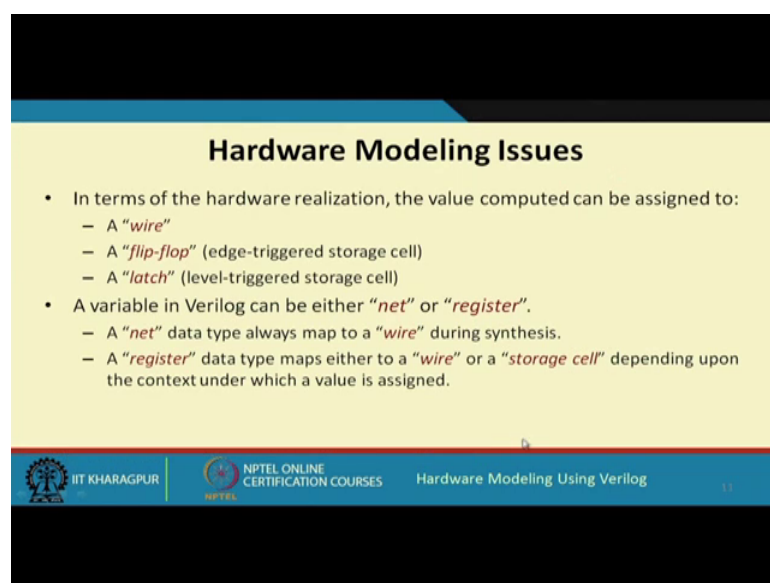
<pre> module testbench; reg X1,X2,X3,X4,X5,X6; wire OUT; example DUT(X1,X2,X3,X4,X5,X6,OUT);  initial begin \$monitor (\$time," X1=%b, X2=%b, X3=%b, X4=%b, X5=%b, X6=%b, OUT=%b", X1,X2,X3,X4,X5,X6,OUT); #5 X1=1;X2=0; X3=0; X4=1; X5=0; X6=0; #5 X1=0; X3=1; #5 X1=1; X3=0; #5 X6=1; #5 \$finish; end endmodule </pre>	<p style="text-align: center;"><b>Positional Association</b></p> <pre> module example (A,B,C,D,E,F,Y); wire t1, t2, t3, Y; nand #1 G1 (t1,A,B); and #2 G2 (t2,C,-B,D); nor #1 G3 (t3,E,F); nand #1 G4 (Y,t1,t2,t3); endmodule </pre>
<p style="text-align: center;">  IIT KHARAGPUR     NPTEL ONLINE CERTIFICATION COURSES    Hardware Modeling Using Verilog </p>	

So, we shall be explaining with an example first positional association. So, we take a simple example let us say we have an example some module name is example and there are 7 parameters A, B, C, D, E, F, Y. So, Y is the output and A, B, C, D, E, F are the input.

Now, inside a test bench test bench is also module. So, we are instantiating this example we are calling a DUT and these are the variable names you are giving there are total seven parameters you are giving X 1, X 2, X 3, X 4, X 5, X 6 OUT, but we are specifying them in the same order the first six inputs we are connecting X 1, X 2, X 3, X 4, X 5, X 6; these you are applying and the last one is the output that is out and out will be observing how do are printing monitor right this is positional association this same order in which we have defined the same order will be using in the instantiation, but in the other one explicit association what you do is something like this in the original example it was A, B, C, D, E, F. So, the output was last suppose when I instantiate; I want output to be coming first.

So, what I have here is here with respect to this module the output first has to be connected to out I write dot out now within bracket the variable name of the original module description which is Y which is Y which means Y is being connecting to out similarly I write dot X 1 within bracket a; that means, X 1 is connecting to a then X 2 is connecting to B, you see here although we have to write more, but we can clearly see that which variable is getting connected to which parameter of the module that you are instantiating and this can be put in any order not necessary A, B, C, D, E, F, I can put f first, then D, then A then C. So, the order is also not important here I can put in any order this is one way to specify the parameters in instantiation.

(Refer Slide Time: 19:42)



**Hardware Modeling Issues**

- In terms of the hardware realization, the value computed can be assigned to:
  - A “*wire*”
  - A “*flip-flop*” (edge-triggered storage cell)
  - A “*latch*” (level-triggered storage cell)
- A variable in Verilog can be either “*net*” or “*register*”.
  - A “*net*” data type always map to a “*wire*” during synthesis.
  - A “*register*” data type maps either to a “*wire*” or a “*storage cell*” depending upon the context under which a value is assigned.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

Now, some of the hardware modeling issues well actually when you use the assignment statement various kind of assignment statements we have seen we have seen the assign statement we have seen the equal and the less than equal.

Now, depending on the type of assignment statement you are using the value that is computed will be assigned to either a net type variable typically a wire or a reg type variable which can be ultimately be a flip-flop or a latch a flip-flop is nothing, but an edge triggered storage cell where data will get stored whenever a clock edge comes and latch is level triggered whenever some enable signal is activated now a variable in Verilog as I have said can be either net or register a net during synthesis will always map to a wire this is a matter of fact similarly a registers which you declare in Verilog. This will map either to a wire or to a storage cell depending on how we have written the Verilog code. So, shall we seeing some examples later. So, we shall see this distinction sometimes the register can be mapped to a wire sometimes it can map to a storage itself.

(Refer Slide Time: 21:09)

```
module reg_maps_to_wire (A, B, C, f1, f2);
  input A, B, C;
  output f1, f2;
  wire A, B, C;
  reg f1, f2;
  always @(A or B or C)
  begin
    f1 = ~(A & B);
    f2 = f1 ^ C;
  end
endmodule
```

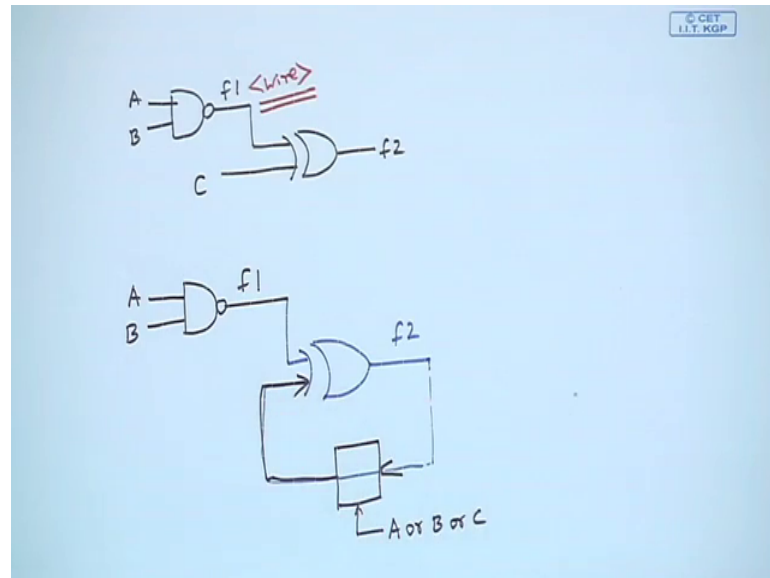
The synthesis system will generate a wire for f1.

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

Let us see some examples this is a module where there are 5 ports A, B, C are the inputs f1 f2 are the outputs say we see I am also declared them as wires this f1 f2 because they are appearing on the left hand side of the always I am in declaring them as a reg well here, I am not using a clock here I am writing always at the rate A or B or C what this statement means is that you execute the always block whenever either of A or B or C this signals are changing their state. So, whenever one or more of the signals are

changing execute the always block the meaning is this. So, here f 1 will be doing a nand and f 2 will be doing an xor with 1 and this; now here you see the first one will be generating a nand; next one will be generating an xor it will be like this.

(Refer Slide Time: 22:26)

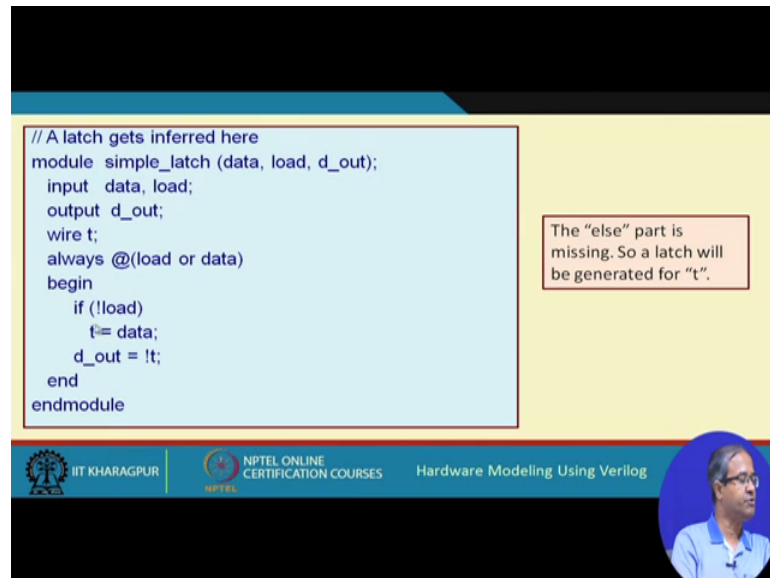


First gate will be an nand where the inputs will be a and b and the output will be f 1 second one would be exclusive or gate where one of the input will be f 1 and the other input will be C and the output will be f 2. So, this will be a combinational circuit which will be generated and for f 1. This will also be a wire this f 1 will also be a wire there is no need for any storage cell here. So, here although we have declared f 1 f 2 as reg both f 1 and f 2 will not require any storage cell either flip flop or latch. Let us take another example where we have changed this little bit f 2 equal to f 1 xor f 2 and this. So, what does this say similar declaration, but f 1 is fine for f 1 there is no problem f 1 equal to nand of A, B.

So, let us try to just also show this; what is happening. So, we are saying A, B this is f 1, but for the other one; I am writing f 2 equal to f 1 xor f 2. So, what does this mean what I am saying is that there is an xor. So, one of the input is f 1 the output is f 2 and we are saying that is output is as if the second input, right. So, you see this kind of a design is normally not recommended what will happen here is that. So, if you do this kind of a design for f 2 a storage cell will be generated, but for f 1 no need f 1 is a simple wire.

So, what will happen is that for f 2 here there will be a latch generated. So, the value of f 2 will be stored here and the stored value will be coming here and this latch will be enabled whenever either A or B or C is changing, this will be how the hardware will be implemented right this is the meaning. So, here f 2 is appearing both on the left hand side on the right hand side that is why a storage say will be generated.


(Refer Slide Time: 25:12)



```
// A latch gets inferred here
module simple_latch (data, load, d_out);
  input data, load;
  output d_out;
  wire t;
  always @(load or data)
  begin
    if (!load)
      t = data;
      d_out = !t;
    end
  endmodule
```

The "else" part is missing. So a latch will be generated for "t".

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



So, let us take another example this is actually the example of a latch where there are 3 ports data load and d out. So, what you are writing input data and load of the input d out is the output and t is a temporary wire t; we are saying always whenever load our data changes begin if load is 0 load 0 active, then data will go to t and d out will be not of t see here the this thing, I will explain in more detail later just one first look of this; in this if statement; there can also be a else if then else we have specified if load is 0, then do this, but we have not mentioned what will happen if load is 1. So, the meaning is if load is 1 the value of t should not change which means the value of t must be stored in a latch.

So, in case of this kind of incomplete if then and statement if you have that will also generate a latch for some of the variables. This one example illustrate, we shall be looking at some more examples later also. So, with this, we come to the end of this lecture where we have looked at again some of the Verilog constructs and we shall be continuing with the discussion there are a lot of other things to discuss also, we have looked at the different ways of assigning some values to variables one is using the assign

statement another is equal to the equal to statement another is less than equal to. So, what are the differences between these assign we have already seen, but what are the other 2. So, we shall be seeing this in our next lectures.

Thank you.