

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

**Lecture – 14**  
**Procedural Assignment (Contd.)**

So, in the last we were discussing some of the procedural assignment statements are the procedural sequential statements in Verilog. So, we talked about statements like the begin end block and if else, nested if else, and the multi way branching using case. So, we continue with our discussion today on procedural assignments.

(Refer Slide Time: 00:49)

**(d) "while" loop**

```
while (<expression>)
    sequential_statement;
```

- The "while" loop executes until the expression is *not true*.
- The sequential\_statement can be a single statement or a group of statements within "begin ... end".

**Example:**

```
integer mycount;
initial
begin
    while (mycount <= 255)
    begin
        $display ("My count:%d", mycount);
        mycount = mycount + 1;
    end
end
```

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

And the first construct that we talk about today is the so called while loop. Well, this while statement is variable is available is standard high level languages like C or C plus plus. So, the syntax in verilog is very similar. So, the general syntax is shown here while within first bracket some logical expression which can evaluate to true or false. So, while this expression is true, this sequential statement or a block of statements within begin end will execute. So, this is a repeated repetitive execution construct that is why we call it a loop. So, as I had said this loop comprising of the sequential statement will execute until the expression is not true. So, as long as it is true, it will continue the execution. And the sequential statement as in the other constructs, it can be either a single statements or it can also be a group of statements within begin end block.

So, just one simple example using while: this is part of a test bench. So, here we are defining an integer called mycount; and in the initial block we are using a while loop. This is just for the sake of demonstration and illustration. So, while the value of mycount is less than or equal to 255 you going to this loop you display the value of the count and then you increase or increment the value of the count. Now here you may argue that well the first time when the programs starts execute it enters the begin block mycount is undefined. So, the first time when while checks the condition what will be the value true or false, this is not well defined.

So, in order to remove this ambiguity, before this while and after this begin you can introduce one statement actually where you can initialize mycount to 0. Like here, you can add a statement where mycount value can be set to 0. So, if you do this then this ambiguity will be removed. So, the first time when it comes into the while loop the mycount value will be 0, so it is definitely less than equal to 255. So, this loop will continue executing, it will continue till mycount crosses 255. So, it will be displaying the value of count my count starting from 0, 0, 1, 2, 3, 4 up to 255, this is just an example, fine.

(Refer Slide Time: 03:50)

**(e) "for" loop**

```
for (expr1; expr2; expr3)
    sequential_statement;
```

- The "for" loop executes as long as the expression expr2 is true.
- The sequential\_statement can be a single statement or a group of statements within "begin ... end".

- The "for" loop consists of three parts:
  - a) An initial condition (expr1).
  - b) A check to see if the terminating condition is true (expr2).
  - c) A procedural assignment to change the value of the control variable (expr3).
- The "for" loop can be conveniently used to initialize an array or memory.

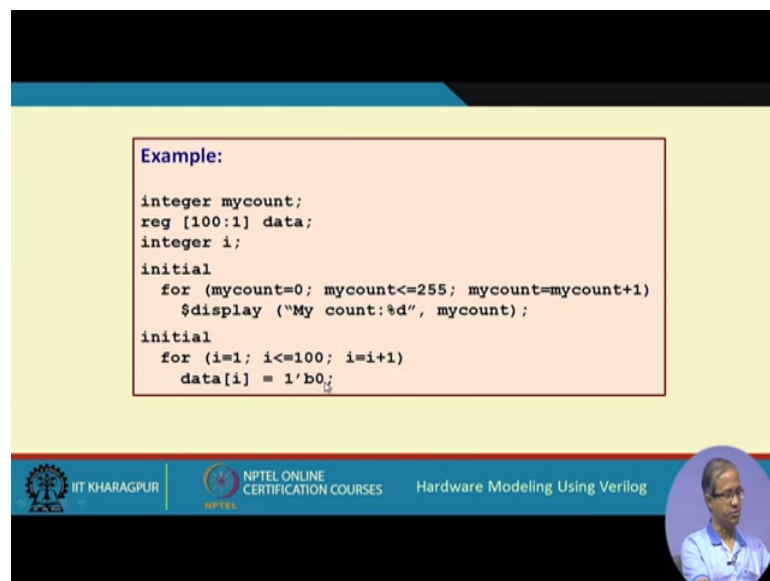
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

The next looping construct is again very similar to the construct available in the language C or C plus plus this is the for loop. Just like in C, so in the for loop the syntax consist of three parts, there are three expressions you can say among them expression one and

expression three, they are actually assignment statements. This expression one basically specifies an initial condition or an initialization some variables get initialized here; and expression three specifies an assignment where some control variable that controls how many times the loop will go on that control variable is updated that is expression three. And expression two is a logical expression which checks or it specifies a termination condition. So, as long as this condition is true the loop will go on.

So, again there is a sequential statement which can be a single statement or it can be a group of statement inside begin end. So, this for loop will start by initializing by executing expression one then it will continue as long as expression two is true; and at the end of every iteration expression three will be executed once to update the control variable. This is just like what is available in standard high-level languages like C.

(Refer Slide Time: 05:29)



**Example:**

```
integer mycount;
reg [100:1] data;
integer i;
initial
  for (mycount=0; mycount<=255; mycount=mycount+1)
    $display ("My count:%d", mycount);
initial
  for (i=1; i<=100; i=i+1)
    data[i] = 1'b0;
```

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

Just a simple example using for this again is part of a test bench, where here I am showing two examples one by one. So, let us say we have defined an integer variable called my count and a vector of bits of type reg. So, it is a vector of size 100, where the index values reg from 1 to 100 called data and an integer i. So, in the first example, there can be multiple initial box I told you. Suppose, in the first initial block, we are just doing exactly what we did earlier using the while loop. Using for loop we are displaying the count values starting from 0 up to 255. So, this will be the initial condition before starting the loop mycount will be initialized to 0; and at the end of the loop, so you

update your control variable mycount equal to mycount plus 1 execute this, it will be incremented then you check whether this condition is still true or not. So, as long as this is true, this loop will go on. So, this display will be executing repeatedly.

Now, in the second example this is again another initial block. Here we are using for loop to initialize this vector to all zeros. So, here as you can see we are using the for loop to run through the index values 1 to 100, starting with 1 and up to 100, and at the end of every iteration we increase i by 1 data i equal to 0. So, these are some examples for using for loops.

(Refer Slide Time: 07:16)

**(f) "repeat" loop**

```
repeat (<expression>
    sequential_statement;
```

- The "repeat" construct executes the loop a fixed number of times.
- It cannot be used to loop on a general logical expression like "while".
- The expression in the "repeat" construct can be a constant, a variable or a signal value.
  - If it is a variable or a signal value, it is evaluated only when the loop starts and not during execution of the loop.
- The sequential\_statement can be a single statement or a group of statements within "begin ... end".

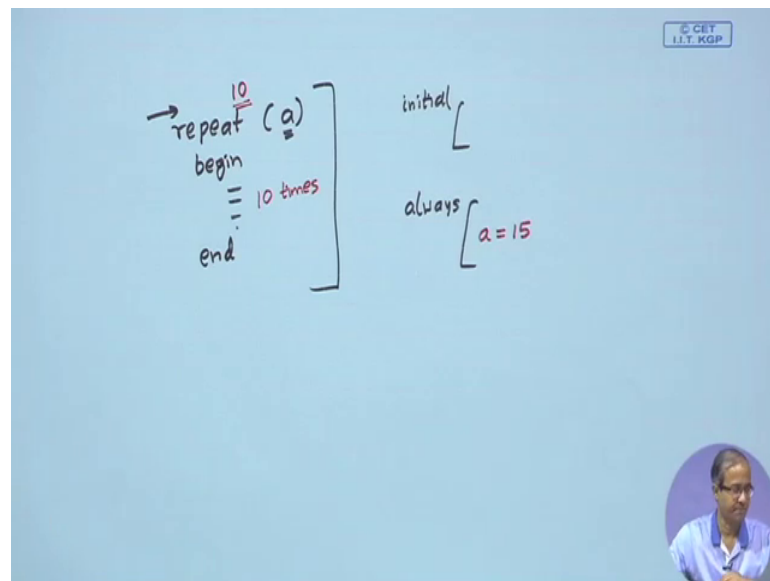
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

Now, the third kinds of loop which is there is repeat. Now, see the main difference is like this for while or for the loop we specify a condition. So, we do not say directly that how many times the loop should go on. So, in the while loop, we specify a condition and as long as that condition is true I continue executing the loop. Similarly, for the for, there is a logical expression we have specified in expression two you recall, there we specify a condition again, where as long as that condition is true we continue with the execution. But in contrast in this repeat statement we do not do like that we specify exactly how many times we want to run the loop.

So, in this repeat statement, this expression whatever is there this actually tells how many times this loop has to execute. So, this construct is used to execute the loop some fixed number of times. And it does not use a logical expression which indicates that

when to stop and when to continue with the loop. So, some constraints on this expression that you can give here, well this expression can either be a constraint, it can be a variable or it can be some signal value, signal value means which is continuously driven. Now, if it is a constant, it is fine you know how many times to loop, but if it is a variable or a continuously driven signal then the rule is that these value is evaluated only when the loops starts.

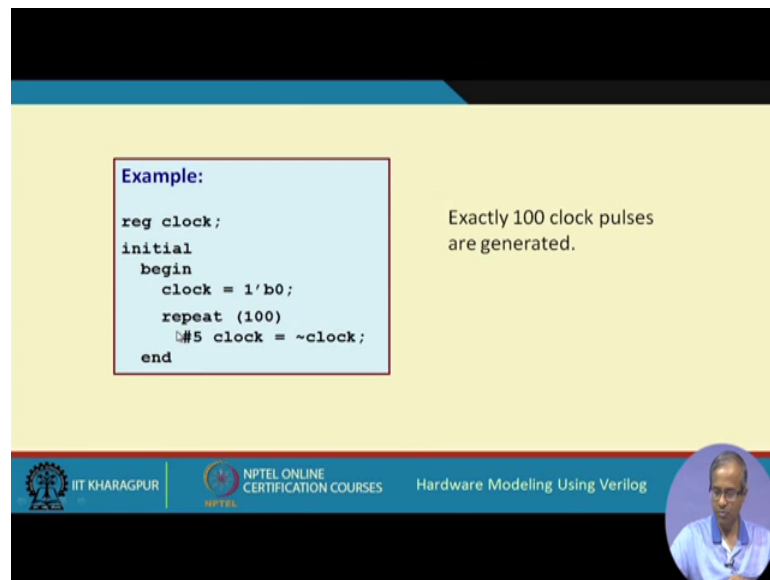
(Refer Slide Time: 09:18)



Suppose, at the beginning of the loop you see let say I have given a statement like this repeat some variable a then within begin end, there are some statements. This a might be this might be updated by some other initial or always block that can be an initial block or there can be an always block, where this variable a might get updated. But here the rule says that when this loop starts, you take the value of a which was there at that point in time.

Suppose the value of a was 10, so this loop will be executing ten times. But even if during execution of the loop suppose in this block the value of a is changing to 15, but still because the value of a was 10 at the beginning of the loop, this loop will be executing ten times only right, this is the rule we have to remember, fine that is the rule.

(Refer Slide Time: 10:25)



The slide features a light yellow background with a blue header and footer. A light blue box on the left contains Verilog code. To the right of the code, a note states 'Exactly 100 clock pulses are generated.' The footer includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with the course title 'Hardware Modeling Using Verilog' and a circular portrait of a man in a blue shirt.

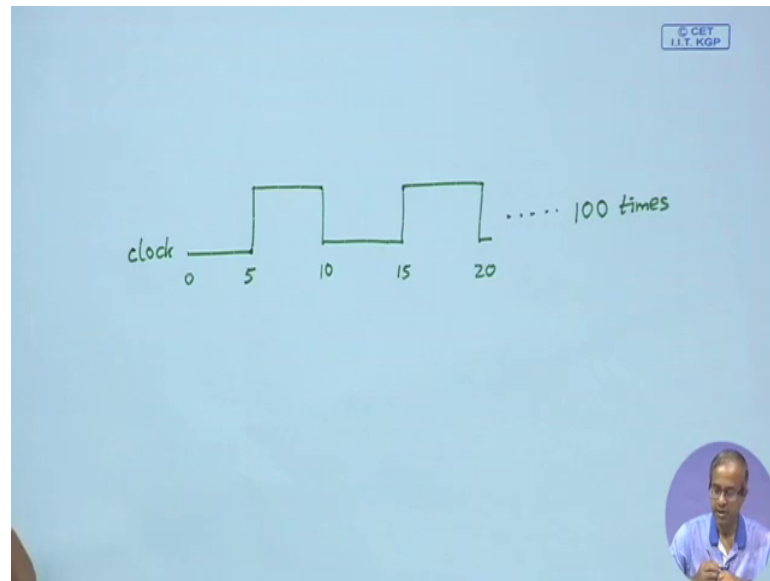
```
Example:  
reg clock;  
initial  
begin  
    clock = 1'b0;  
    repeat (100)  
        #5 clock = ~clock;  
    end
```

Exactly 100 clock pulses  
are generated.

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

So, one simple example here we are actually generating a fix number of clock pulses in a test bench let say. Here we are defining a signal clock as of type reg. Well, I am repeating you will have to define a variable of type reg if it appears on the left hand side inside an initial or an always block. Here because it is appearing on the left hand side, you have to use reg. So, what we are doing we are initializing the clock to zero then there is a repeat loop repeat hundred times. So, what we are doing repeat hundred times we are giving a delay of five and we are changing the value of clock, the clock was zero, so not clock will be one, one will be assigned to clock. So, again after delay of five, not of one zero will be there that zero will be assigned and this will repeat hundred times.

(Refer Slide Time: 11:33)



So, it will be something like this. This variable clock this will be initialized to 0. And at time five this is at time 0, at time 5, it will be toggled, it will become 1. Again at time 10, it will remain at 1; at time 10, it will again come back to 0. At time fifteen, it will again become 1; at time 20, it will again become 0, and this will continue exactly 100 times right.

So, if you want in some application to generate a clock which will be free running continuously without any constraint then you do not give this kind of restriction like 100 pulses then you can do something always or there is another construct will see later called forever which can implement an infinite loop, a loop which never stops. But for repeat you have to specify exactly how many times you are expected to carry on or continue with the loop.


(Refer Slide Time: 12:40)

**(g) "forever" loop**

```
forever  
  sequential_statement;
```

- The "forever" loop is typically used along with timing specifier.
  - If delay is not specified, the simulator would execute this statement indefinitely without advancing \$time.
  - Rest of design will never be executed.
- The "forever" construct does not use any expression and executes forever until \$finish is encountered in the test bench.
  - Equivalent to a "while" loop for which the expression is always true.
- The sequential\_statement can be a single statement or a group of statements within "begin ... end".

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



Next is with forever loop as I had said, this is a version of a loop where you execute the sequential statement infinite number of times forever. In the forever construct, you do not use any expression or any count value how many times to loop. So, you will be continuing with this loop forever, now with respect to the test bench if some block executes finish then only this will be terminated; otherwise this will go on continuously. So, you can say that this forever construct is like a while loop, where the expression is always true. So, in place of the expression, you give something which is always to like you can write something as follows.


(Refer Slide Time: 13:35)

© CET  
I.I.T. KGP

while (1 < 2)  
 begin  
 ==  
 end

always true

====  
forever  
 begin  
 ==  
 end






Let say just if you write while 1 less than 2 do something begin a block of statements end. Now, clearly this 1 less than 2 is a constant expression this one which is always true. So, this while loop is equivalent to a forever construct where you may use forever and again the same set of statements inside begin end. These two will be equivalent. So, this sequential statement again it can be single statement or it can be a group of statement within begin end.

Now, there is one issue here you have to be careful about this statement forever is a kind of a statement which is unconditional, it does not check anything and continuously go on executing. So, it is very important to specify a delay using some hash, hash 5, hash 10 something like that. What might happen is that if you do not specify the delay then the simulator will start this forever loop and it will never execute and because there is no delay, the time will also be not advancing. So, simulator would execute this statement indefinitely infinitely without advancing time.


Now, because time is not advancing, the rest of the design the following statements are other blocks required there is a delay specified, it will never be executed. Because you see it may so happen that you have started to execute this forever statement at a time t equal to 5 let say when dollar time is equal to 5. Now, there is another initial block, which says that at time 10, you have to do something you have to print or you have to assign something, but because this forever has started and it has not finished it is in an infinite loop, so time will never advance. And the other statement which is supposed to be activated at time 10, it will never be activated. So, whenever you use forever be sure to use a timing specify there.

(Refer Slide Time: 16:17)

```
// Clock generation using "forever" construct
reg clk;
initial
begin
  clk = 1'b0;
  forever #5 clk = ~clk; // Clock period of 10 units
end
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES


Hardware Modeling Using Verilog

Like let us take an example, a simple example that uses a clock signal indefinitely. So, inside the initial block, we again say initialize a signal clk to 0 then we give a forward loop with a timing delay forever delay of 5 clock equal not clock. So, this will generate a clock with a time period of 10 units 5 time units which will remain 1, 5 time units it remain 0, and this will go on indefinitely because it is a forever loop. But now the time is advancing 5, 10, 15, 20 it is advancing, so you are not only up any other statements in the other initializer always blocks fine.


(Refer Slide Time: 17:05)

### Other Constructs Available

# (time_value)	<ul style="list-style-type: none"><li>• Makes a block suspend for "time_value" units of time.</li><li>• The time unit can be specified using the `timescale command.</li></ul>
@ (event_expression)	<ul style="list-style-type: none"><li>• Makes a block suspend until "event_expression" triggers.</li><li>• Various keywords associated with "event_expression" shall be discussed with examples..</li></ul>




IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

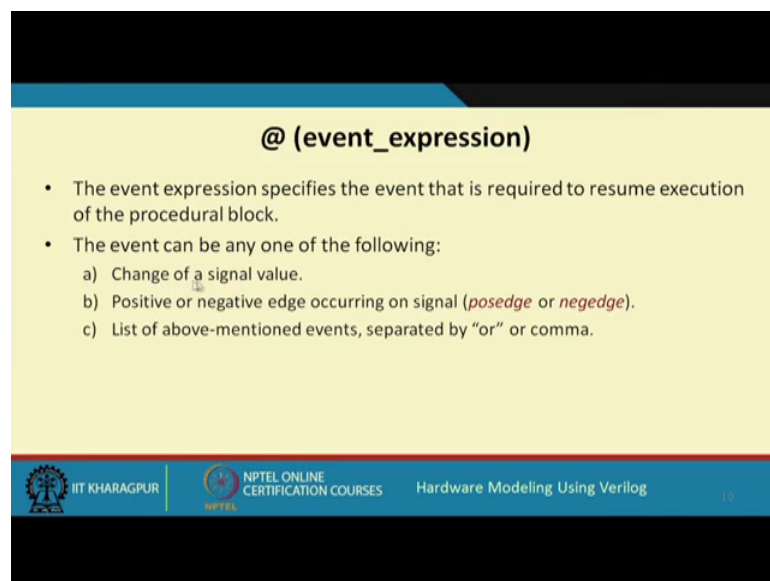
Hardware Modeling Using Verilog



So, the other constructs that are available in verilog one of course, you have used this time specifier hash followed by some time value. So, the meaning of this statement is a block that has this kind of timing delay specified at the beginning of it the block will be suspended for this much amount of time before it gets executed. And the time unit can be specified as you have mentioned earlier using the time scale command. So, it can some unit nanosecond, picoseconds, second, millisecond anything fine.

So, the other important construct which you have already seen through some examples which you were particular in the always block you use this, this is at the rate with in bracket some event expression. Here what you are doing, here we are making a block suspend its execution until this event expression triggers. So, this event expression can be specified in various ways we will see, there are some keywords using which you can specify this event expressions.

(Refer Slide Time: 18:21)



**@ (event\_expression)**

- The event expression specifies the event that is required to resume execution of the procedural block.
- The event can be any one of the following:
  - a) Change of a signal value.
  - b) Positive or negative edge occurring on signal (*posedge* or *negedge*).
  - c) List of above-mentioned events, separated by "or" or comma.

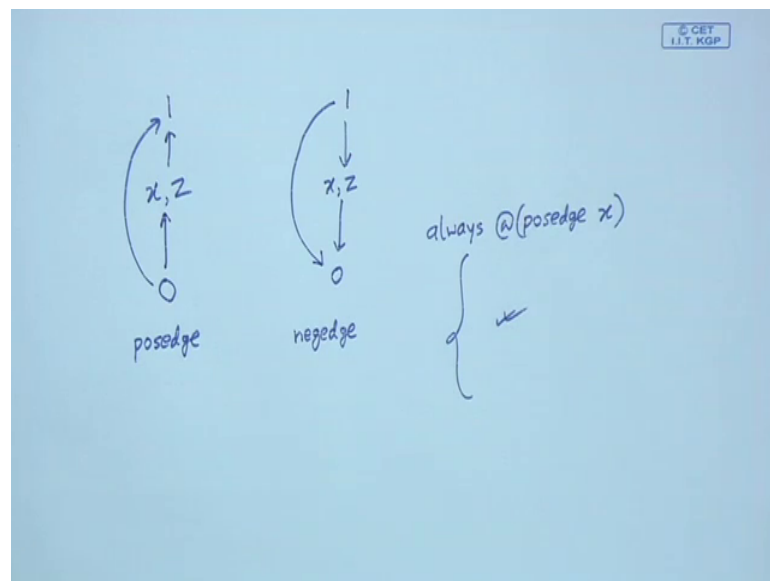
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 10

Let us see how. So, just to recall, so in the event expression where actually specifying some event, and when that event occurs whatever is in the procedural block inside that that will be executed or it will be resuming its execution. Because you see we mention that in an always block what you do inside always there is a group of statements within begin end let us say, this always block is supposed to execute this block of statement repeatedly, but when it will not do so continuously. Whenever the event expression is true, then it will be executing once; then it will go back and wait till the event expression

gets true again. If it happens it will execute a second time, then it will again go back again it will check whether the event expression is true. And if it is not true, it will wait till it becomes true, it will be executing it third time in this way it will go on. So, event expression will tell you exactly when to execute the block the next type.

So, the event actually can be in general one of the following. So, either the change of a signal value, so you must specify whenever a particular signal value or any of the values change, then do this or you can specify either a positive or negative edge of a signal. This is particularly used for sequential circuit synchronous sequential circuit specifications using the keyword posedge or negedge. Or you can have a combination of several of these which is separated by a keyword or, or you can also use a comma. So, by definition in verilog whenever we talk about a positive edge, well a posedge does not necessarily mean a transition from 0 to 1; actually a posedge is defined as any transition where a signal changes from either 0, x or z to 1; or from 0 to z, or x.

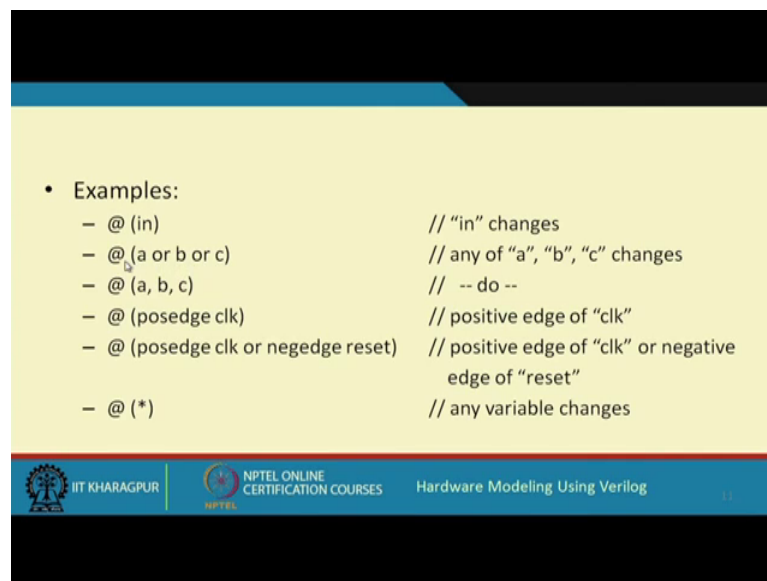
(Refer Slide Time: 20:53)



So, the idea is that in terms of the hierarchy zero is considered to be lowest one is considered to be highest, an x and z are considered to be in between. So, you define a posedge I mean if either you have a transition from 0 to x, z, from x, z to 1, or from 0 to 1 in this way you define posedge. Similarly, you can define a negative edge as transition like this. So, where again the idea is similar, so you again have a hierarchy where x then you have x, z then you have 0.

So, any transition where you are either moving from 1 to 0, or 1 to x or z, or x or z to 0, this is defined as negedge; so in verilog whenever the value of a variable is like you specify like this always at the rate posedge of some variable let say x. So, whenever the value of a variable x changes from a value which is from 0 to 1, or 0 to z, 0 to x, x to 1 means x means do not care here do not confuse this x with this x, then a positive edge will be defined and the block which is inside will get executed.

(Refer Slide Time: 22:38)



• Examples:

- @ (in) // "in" changes
- @ (a or b or c) // any of "a", "b", "c" changes
- @ (a, b, c) // -- do --
- @ (posedge clk) // positive edge of "clk"
- @ (posedge clk or negedge reset) // positive edge of "clk" or negative edge of "reset"
- @ (\*) // any variable changes

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

So, let us take some examples of this event specification. So, I can right at the rate with in bracket the name of a variable. This actually means the event, which is defined as a change in the variable. So, whenever the variable in changes, this event is said to have occurred. Similarly, I can write a or b or c, this means whenever any of the variables either a or b or c or more than one changes then this condition is considered to be true and the block will be executed. Now, in place of or you can also write a comma, the meaning is the same. So, you can either write the variable separate by or, or the variables separated by commas. So, I mentioned posedge, so you can specify the posdge on some variable, clk is the variable I am referring to a clock here.

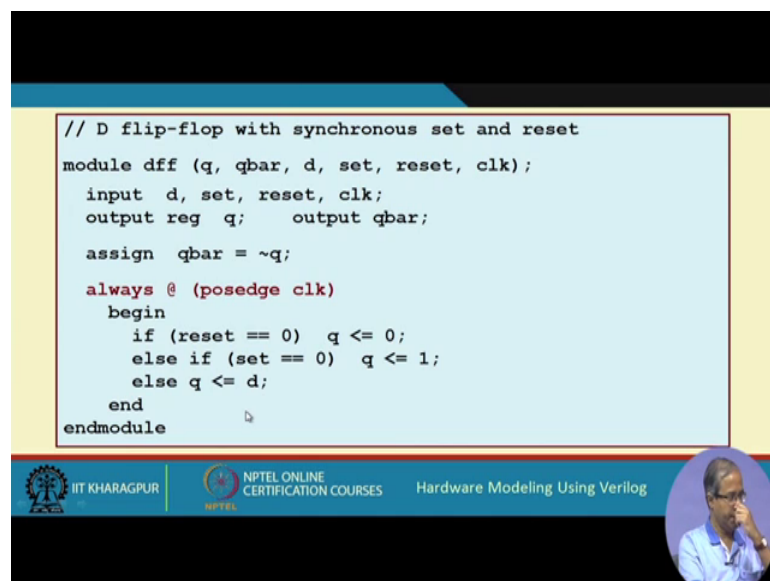
So, whenever there is a positive edge, I have defined what is posedge on this variable, this event is considered to be true. So, you can combine several of the events. You can say that at the rate either posedge clock or negedge of reset. So, whenever either the positive edge of clock comes or the negative edge of reset comes you execute the block

the event you can define a composite event by using or and in place of or you can also put comma like this both are allowed.

And the last one I have been telling here I am putting a star. You see here I have given at the rate a or b or c. So, if there are hundred variables in my verilog code, I can I may have to write all the hundred variables. Now, if I write a star, star means any if any of the variable changes execute the block. So, in some designs in some modules writing a star becomes more convenient because first thing is that you will not have to write. So, many variable names and suddenly you may have forgotten to list one of the variables.

So, you see that your simulation is not coming correctly, may be you have missed one of the where d was not there, but if you write star then there is no problem. So, you know that you have included this variables all of them, fine.

(Refer Slide Time: 25:21)



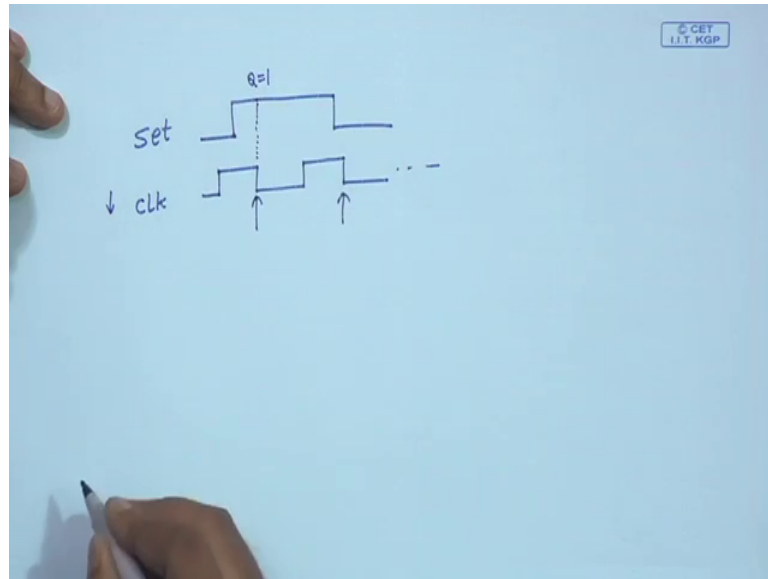
```
// D flip-flop with synchronous set and reset
module dff (q, qbar, d, set, reset, clk);
  input d, set, reset, clk;
  output reg q;   output qbar;

  assign qbar = ~q;

  always @ (posedge clk)
  begin
    if (reset == 0) q <= 0;
    else if (set == 0) q <= 1;
    else q <= d;
  end
endmodule
```

Let us take an example some illustrations of these always usage of always. So, here I am showing the module description of a D type flip-flop with synchronous set and reset. So, when I say a flip-flop has synchronous set and reset. It means whenever the clock the active clock edge comes setting and reset of the flip-flop takes place in synchronism with that clock edge this is meant by synchronous set or reset. Everything setting means setting the flip-flop to one; resetting means setting it to 0; this happens only when the next clock edge comes and the corresponding set or reset signals are active like let me give an example.

(Refer Slide Time: 26:17)



Suppose I have a set signal, let say I have set the signal to high and it goes high like this and this is my clock, my clock is going like suppose my clock is triggered by the falling edge let say my clock signal is going like this. So, I have one active edge here, I have one active edge here, and this will continue. So, whenever the falling edge of the clock comes, I check whether set is high or low. Suppose set is active high, so I find it high here. So, the flip-flop will be set the output, Q will be set to 1. So, the next blockage I find set is already written back to 0 so nothing will happen no change. Synchronous means changes take space at the active edge of the clock right setting and resetting similarly.

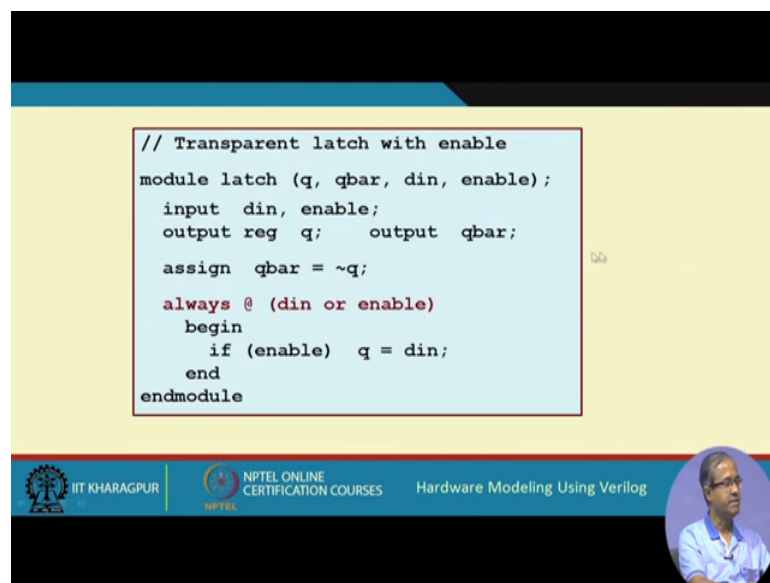
So, let us see the module we have q, q bar, d is the input and set reset and the clock. So, d, set, reset, and clock are the inputs; and this q is defined as an output which is also a reg. Well, we could have defined q bar also as reg, but this q bar we are generating directly from q through an assign statement, q bar equal to not q. Because inside this always block we are only initializing q, and q bar we are generating as the not of q; and inside this always block what we are doing always at the rate posedge clock. So, whenever the clock edge comes a positive edge these example shows 0 to 1, you do this.

You check if reset is zero then you set the output q to 0; else if set is 0, then set q to 1; and otherwise if neither set or reset is active. So, set and reset are active low here. Whenever they are 0, they are active; otherwise you store the input data in q. This is

synchronous because you are executing the block only when the active edge of the clock is coming. So, if the reset you have made 0 earlier, so it will not be reset immediately. Now, the meaning of this equal and less than equal we shall be explaining later, but for the time being you understand the meaning of these statements. This you can easily understand fine.

Let us take another small modification where we have made it asynchronous set and reset. So, in the earlier example here, we are only triggering this block at the posedge of the clock, but now you are saying at posedge of the clock or negedge of set or negedge of reset. That means, I am not only waiting for the clock to come, but if I have seen that the set signal has gone down to 0, so immediately I can execute this block; or the research signal has gone down to 0, I can immediately execute this block. So, this is called asynchronous set and reset where the set and reset operations do not depend on the clock signal.

(Refer Slide Time: 29:37)

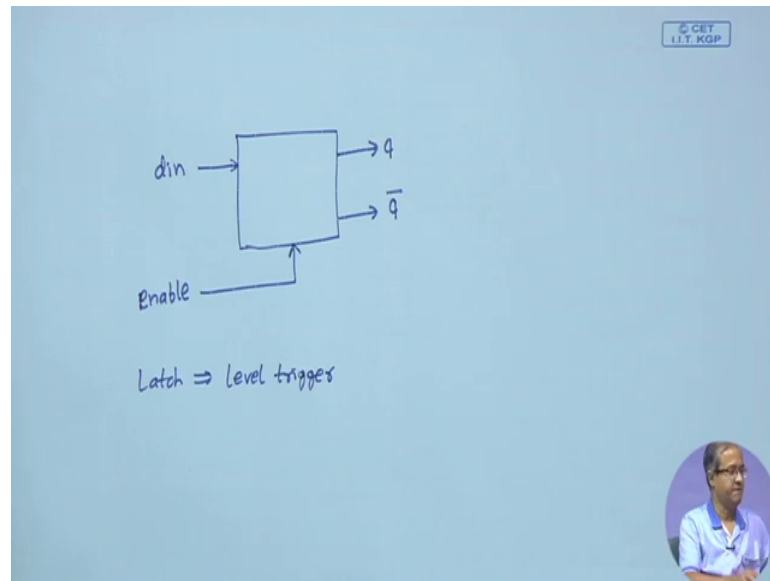


```
// Transparent latch with enable
module latch (q, qbar, din, enable);
input din, enable;
output reg q; output qbar;
assign qbar = ~q;
always @ (din or enable)
begin
if (enable) q = din;
end
endmodule
```

And in this example, we have a transparent latch with enable. Transparent latch mean that there is no clock, there is a latch q, q bar, there is a data in and there is an enable. So, it is a something like this.



(Refer Slide Time: 30:00)



There is a latch where there is a there is input called din there are outputs one is q, one is q bar, and there is an enable, this is not a clock. So, whenever this enable is active, if enable is one then d in will stored in the latch. So, this is level triggered, you recall a latch means this is level triggered this is not activated by the edge like in a flip-flop fine. So, the description is very simple. So, you declare din and enable as input q as reg and again this output q bar is an output which you can generate using assign not of q. And in this block there is no clock. So, in this always block, you are saying din or enable. So, whenever either the input changes or the enable changes, you execute this block, it says if enable is true which means enable is one, one means true then you assign d into q this is the description of a transparent latch.

So, with this we come to the end of this lecture. Now, in the next lecture, we shall be looking at some of the examples that uses some of the constructs that we have already seen. And with respect to synthesis, we shall also discussing a few of the very interesting concepts.

Thank you.