

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

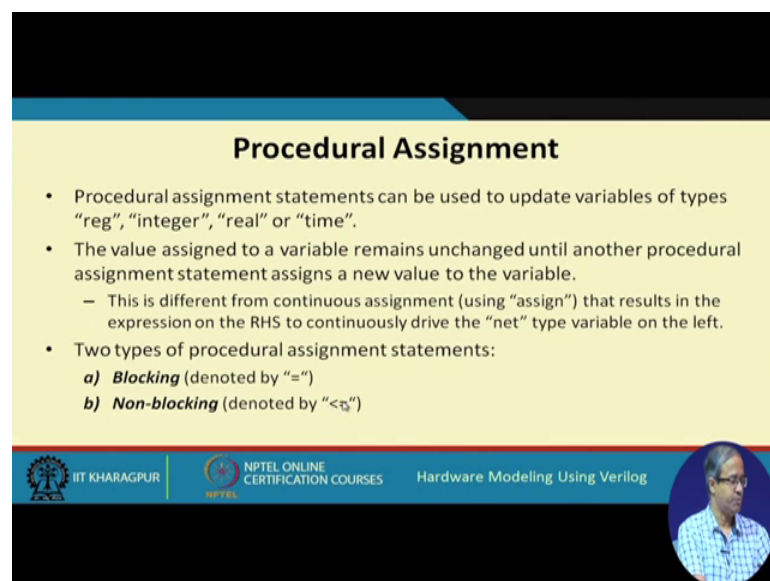
Lecture - 16
Blocking / Non-Blocking Assignments (Part 1)

So, we have been talking about various kinds of assignment statements in verilog. So, recall we talked about the continuous assignment statement using assign. Then we talked about the procedural assignment statements, we looked at several examples where inside an always block or inside an initial block we can use various kinds of assignments to variables. These are called procedural kind of assignments.

Now broadly speaking these procedural kind of assignments come in 2 different flavors. So, in this lecture we shall be starting our discussion to clearly distinguish and try to find out when and under what conditions should we use which of these 2 options or alternatives.

So, the topic of our lecture today's Blocking and Non-Blocking Assignments. These are the 2 types that I was talking about fine.

(Refer Slide Time: 01:23)



Procedural Assignment

- Procedural assignment statements can be used to update variables of types "reg", "integer", "real" or "time".
- The value assigned to a variable remains unchanged until another procedural assignment statement assigns a new value to the variable.
 - This is different from continuous assignment (using "assign") that results in the expression on the RHS to continuously drive the "net" type variable on the left.
- Two types of procedural assignment statements:
 - a) **Blocking** (denoted by "=")
 - b) **Non-blocking** (denoted by "$=$")

The slide footer contains the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and the text 'Hardware Modeling Using Verilog'. A small circular inset image of Prof. Indranil Sengupta is visible in the bottom right corner of the slide.

So, these assignment statements fall under the broad category of procedural assignments. So, what do you mean by procedural assignment? Procedural assignment is an

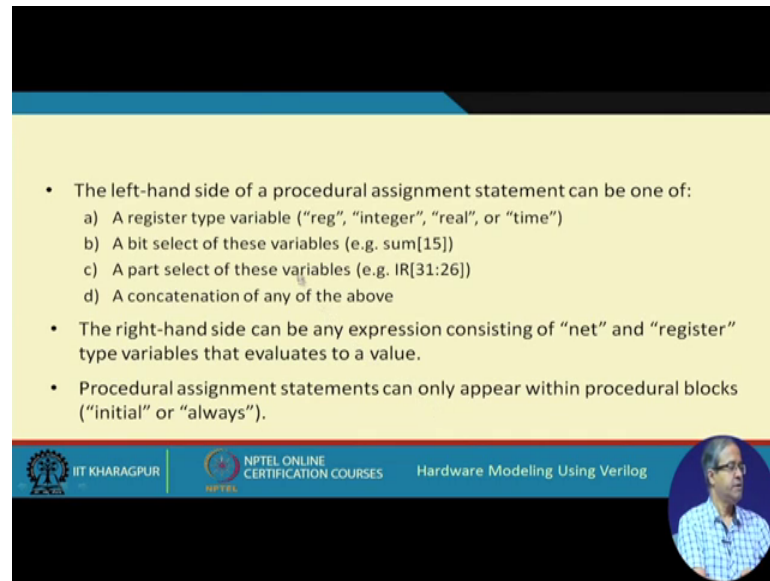
assignment and expression which assigns some value to a variable, which lies or figures inside a procedural block, in verilog a procedural block can be either initial or always ok.

So, these kind of procedural assignment statements can be used to update variables only of these types reg, integer, real or if you have a time variable which keeps track of time. You see you cannot have a net type variable on the left hand side of a procedural assignment. This is the restriction. So, if you are using some variable to assign a value inside a procedural block, they have to be one of these data types. Now you see there is also very clear difference from the continuous assignment type. Say in continuous assignment which we had seen earlier using assign, let us say we write assign a equal to b plus c.

Here whenever the value of b plus c changes, the value of a will be immediately changing it will be directly driving this net type variable on the left, and this net type variable on the left of this assign statement do not have any facility or capability of storing this value. That is why we call this as continuously driven. As b and c changes a will be continuously changing right, but in contrast for a procedural assignment there is an important difference to be remembered. The value that we assigned to a variable this remains unchanged, until you assign some other value to that same variable again. So, there is some notion of storage or memory associated with the variables that you use here.

So, once you assign some value to a variable, the value will remain unchanged until you again assign it to some other value. So, this is the difference from assign which I just now talked about. So, broadly speaking this assignment statements inside procedural blocks can be blocking or non-blocking. And they are denoted by the assignment operator. Blocking is denoted by equal to non-blocking is denoted by the arrow symbol less than equal to this symbol this means arrow fine.

(Refer Slide Time: 04:21)



The slide contains a list of rules for procedural assignment statements. The background is yellow with a blue header and footer. The footer includes the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and the course title 'Hardware Modeling Using Verilog'. A small circular portrait of a man is visible in the bottom right corner of the slide.

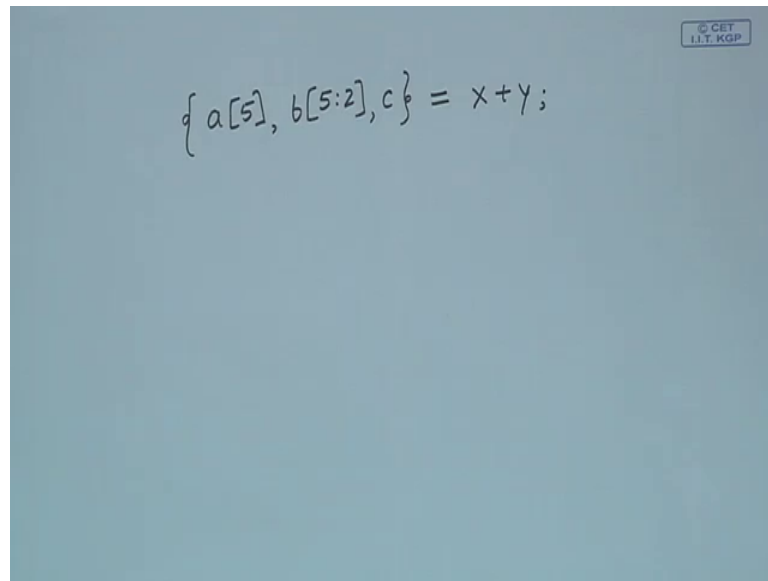
- The left-hand side of a procedural assignment statement can be one of:
 - a) A register type variable ("reg", "integer", "real", or "time")
 - b) A bit select of these variables (e.g. sum[15])
 - c) A part select of these variables (e.g. IR[31:26])
 - d) A concatenation of any of the above
- The right-hand side can be any expression consisting of "net" and "register" type variables that evaluates to a value.
- Procedural assignment statements can only appear within procedural blocks ("initial" or "always").

So, the rules for a procedural assignment statement are as follows.

So, I already talked about that the variable on the left hand side should be one of reg integer real or time this of course, are constrained. Not only this on the left hand side we can also have a bit select of one of the variables of this type like you can write some 15 equal to something right. This is one particular bit of a data of this type. Or you can also select a part by specifying the index values like IR bit numbers 26 to 31 equal to something. This also you can mention, or you can concatenate one or more of this by enclosing them with thee within the curly braces as I mentioned earlier.

So, you can also use the concatenation operation like for example, you can write concatenation.

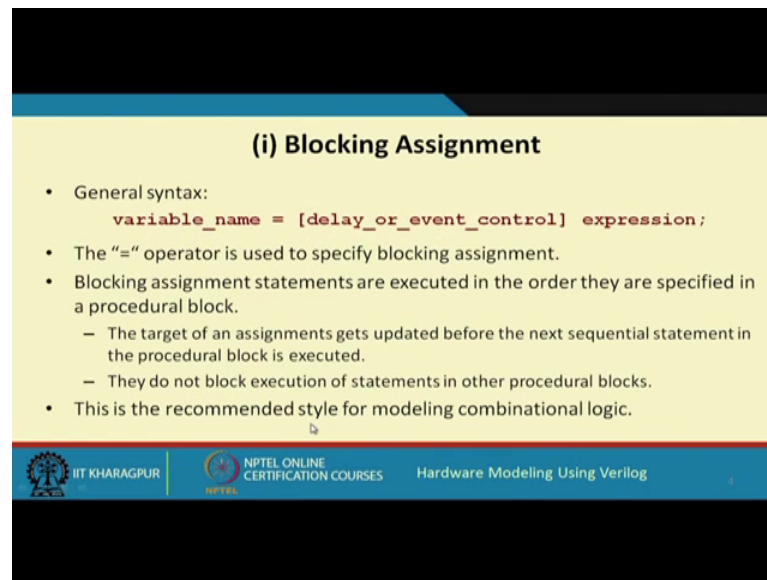
(Refer Slide Time: 05:20)



Let us say a variable a 5 comma a variable b you can have a section defined or an entire variable c. So, so you can define a concatenation like this equal to you can have any expression like x plus y. So, these kinds of expressions are allowed provided a b c these variables are one of these 4 types. Reg, integer, real or time, but the right hand side of the expression can be anything you can have any combination of net type variable and also, register type variables. And these assignments as I mentioned can appear only within an initial block or within an always block.

So, just outside these blocks you cannot use these assignments right.

(Refer Slide Time: 06:17)



(i) Blocking Assignment

- General syntax:
`variable_name = [delay_or_event_control] expression;`
- The “=” operator is used to specify blocking assignment.
- Blocking assignment statements are executed in the order they are specified in a procedural block.
 - The target of an assignments gets updated before the next sequential statement in the procedural block is executed.
 - They do not block execution of statements in other procedural blocks.
- This is the recommended style for modeling combinational logic.

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

So let us start with blocking assignment let us try to understand what it really means. Now the general syntax is on the left hand side you have the variable name of the types one of the types I mentioned, equal to well you can mention some kind of triggering event like you can mention a delay or you can also mention an event like at some clock edge or something that also you can mention; variable name equal to some expression. So, for the time being let us ignore this. Variable name equal to some expression this is the general syntax, and this equal to symbol is used to indicate blocking assignment.

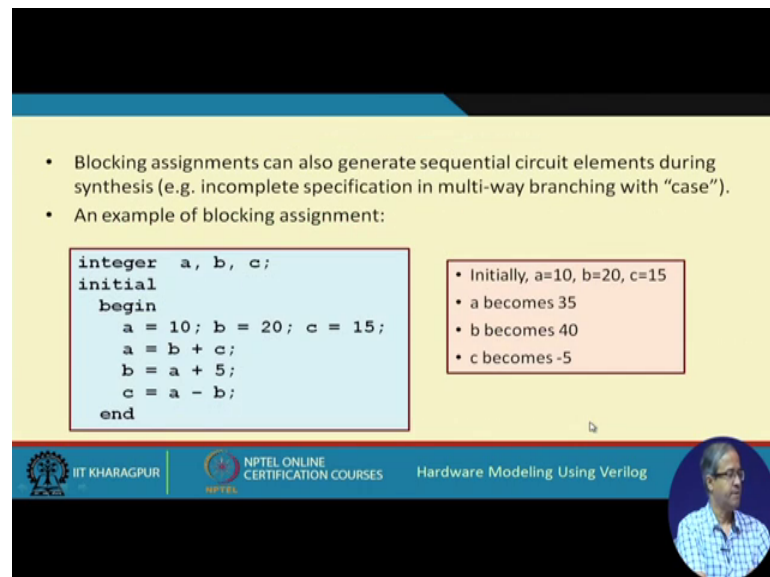
So, what is the meaning of blocking assignment? You see, so if inside a block there are several statements, the statements will be executed 1 by one in the order they are specified. Like suppose inside a procedural block begin end I have 4 statements, this statement will be executed one after the other in sequence. Suppose the first statement is a equal to b plus c. So, some value will be assigned to a. Second statement is let us say d equal to a plus 5. So, that value of a which was assigned that value of a will be used in the next expression. So, in that way 1 by one sequentially the instructions will get executed.

This is the interpretation of blocking assignment; that means, an instruction which is executing or an assignment statement which is executing it blocks the execution of all the statements which follow. So, unless it is finished the next instruction will not start, this is the meaning of blocking assignment. So, the target of an assignment statement will

get updated before the next statement is executed right, but if we have several blocks suppose I have 2 always blocks. Then a blocking type statement in one of the blocks will not block any statements in the other block. It only applied to the statements inside the same block ok.

So, this point is mentioned they do not block execution of statements in other procedural blocks. Now this blocking assignment style can be used to model sequential circuits also, but it is recommended that we use this assignment style to model combination logic we shall see a lot of examples later and try to find out why this is so.

(Refer Slide Time: 09:06)



The slide contains the following text:

- Blocking assignments can also generate sequential circuit elements during synthesis (e.g. incomplete specification in multi-way branching with “case”).
- An example of blocking assignment:

```
integer a, b, c;
initial
begin
  a = 10; b = 20; c = 15;
  a = b + c;
  b = a + 5;
  c = a - b;
end
```

- Initially, a=10, b=20, c=15
- a becomes 35
- b becomes 40
- c becomes -5

At the bottom of the slide, there are logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and the text 'Hardware Modeling Using Verilog'. A small circular portrait of a man is visible in the bottom right corner.

This is what I mentioned. So, using blocking assignment you can also generate sequential circuit elements is means we had seen some examples earlier you recall those examples where you had a multi way branching using case, where if all the values of the case variable were not specified suppose I am doing a case on a variable a, which can take on the value 0 1 2 and 3, but I specify what will happen if the values are 0 1 and 3, but for 2 I am not specifying. So, if the value is 2 what will happen? So, all the variables which appear on the left hand side they should not change, which means they should remember or memorize their value. So, the synthesis tool what it will do? It will be generating a latch a storage element for such variables, but if in the case statement you mention all the conditions without an ambiguity, then it will generate a pure

combinational circuit, because earlier we took some examples to explain these things fine.

So, an example of blocking assignment is shown here, this is not the complete verilog module just a segment of the module. Let us say a b c are 3 variables of type integer. So, inside a initial block I have this begin end. So, see the statements a 10 b 20 c 15 these are the 3 statements 1 by one they will get executed. So, initially this a b c will be getting the values 10 then b 20 then c 15.

Next statement is a equal to b plus c. So, b and c will be added the value will go to a. So, it will be 35. So, the new value of a will become 35 the next statement is b equal to a plus 5. This new value of a will be used and 5 will be added to it. So, b will become 40. And the last statement is c equal to a minus b. So, the latest value of a and the latest value of b they are subtracted and the result will be minus 5.

So, this is the interpretation of the blocking assignment, statements execute in the order they appear just like what we see in a high level programming language like c or java fine.

(Refer Slide Time: 11:36)

```
module blocking_example;
  reg X, Y, Z;
  reg [31:0] A, B;   integer sum;

  initial
  begin
    X = 1'b0; Y = 1'b0; Z = 1'b1; // At time = 0
    sum = 1; // At time = 0
    A = 31'b0; B = 31'hbababab; // At time = 0
    #5 A[5] = 1'b1; // At time = 5
    #10 B[31:29] = {X, Y, Z}; // At time = 15
    sum = sum + 5; // At time = 15
  end
endmodule
```

There is one example here, where various kinds of assignments are demonstrated are shown. Here we are defining 3 variables x y z and 2 vectors of size 32 a b. This is not a meaningful code just an illustration. So, you say at the beginning we give some

assignments, say x equal to 0, y equal to 0, z equal to 1. These are all single bit variables and sum equal to 1 sum is an integer. Because we have not mentioned any time delay these assignments all will take place at time t equal to 0.

Similar is the case for these 2 assignments, a 31 bit all 0 and b 31 bit hexadecimal we initialize it with the hexadecimal number a b, a b, a b, a b; this all they happen at time t equal to 0. Now let us say I write at time delay 5 hash 5 a 5 equal to 1. So, the fifth bit of a will be changed to one at time 5, then I give another delay 10. So, I define a segment of b these 3 bits 29 30 and 31 they will be assigned a value which will be the concatenation of x y and z. And this will happen after delay of 15 which means at time 15 and lastly sum we are incrementing by 5. So, again because there is no delay here, this will also happen at time 15 ok.

This is just an example showing how the times are kept track off in blocking assignments.

(Refer Slide Time: 13:31)

Simulation of an Example

```
module blocking_assgn;
integer a, b, c, d;
always @ (*)
repeat (4)
begin
#5 a = b + c;
#5 d = a - 3;
#5 b = d + 10;
#5 c = c + 1;
end
endmodule

initial
begin
$monitor ($time, "a=%4d, b=%4d,
c=%4d, d=%4d", a, b, c, d);
a = 30; b = 20; c = 15; d = 5;
#100 $finish;
end
endmodule
```

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

Now, here we take an example of a blocking assignment, and we shall show the results through simulation because there are a few things to understand here. This is the example that we take for variables defined a b c d, this is a always block. Always at the rate star means any variable changes, and inside this block we are saying repeat this block 4 times. So, I am saying this begin end block has to be executed 4 times. And each of them will be having a delay of 5. And you see we have written a test bench which monitors the

values the time then a b c and d. These values will be printed whenever some changes of these variables take place.

So, we start by initializing a 2 30 b 20 c 15 and d 5 and we continue the simulation, till time 100 at 100 we call finish. So, let us see when you simulate this code with this test bench what is the output ok.

(Refer Slide Time: 14:44)

0	a=	30	, b=	20	, c=	15	, d=	5
5	a=	35	, b=	20	, c=	15	, d=	5
10	a=	35	, b=	20	, c=	15	, d=	32
15	a=	35	, b=	42	, c=	15	, d=	32
20	a=	35	, b=	42	, c=	16	, d=	32
25	a=	58	, b=	42	, c=	16	, d=	32
30	a=	58	, b=	42	, c=	16	, d=	55
35	a=	58	, b=	65	, c=	16	, d=	55
40	a=	58	, b=	65	, c=	17	, d=	55
45	a=	82	, b=	65	, c=	17	, d=	55
50	a=	82	, b=	65	, c=	17	, d=	79
55	a=	82	, b=	89	, c=	17	, d=	79
60	a=	82	, b=	89	, c=	18	, d=	79
65	a=	107	, b=	89	, c=	18	, d=	79
70	a=	107	, b=	89	, c=	18	, d=	104
75	a=	107	, b=	114	, c=	18	, d=	104
80	a=	107	, b=	114	, c=	19	, d=	104

```

Simulation
Results

Initially:
a=30, b=20, c=15, d=5
always @ (*)
repeat (4)
begin
#5 a = b + c;
#5 d = a - 3;
#5 b = d + 10;
#5 c = c + 1;
end

```

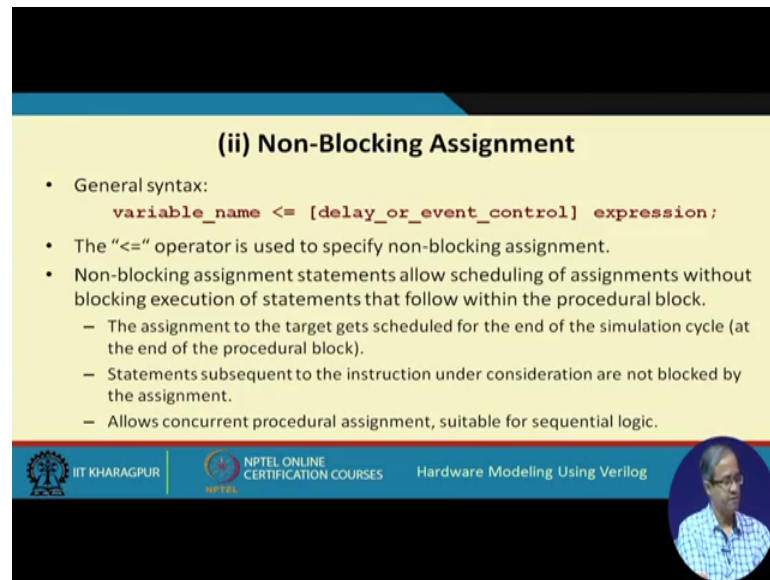
So, I am showing here this code side by sides. So, that you can understand what is happening, you see in the test bench we initialized a 30 b 20 c 15 d 5. So, I am showing the initial values here. And this is the simulation output which you got. And on color I have shown the values which are changing ok.

So, initially at time t equal to 0, a 30 b 20 c 15 d 5 was printed. Then at time 5 this a equal to b plus c happened. So, b and c was added and a become 35 others did not change. Then again after time 5 d equal to a minus 3. So, a was 32 d become 32 35 minus 3 32. Then again 5 b equal to d plus 10 d plus 10 b becomes 42. Then lastly c equal to c plus 1; C was 15 it becomes 16. Now this will repeat 4 times. So, again a equal to b plus c, b is 42 c is 16. So, a becomes 50 8 d, equal to a minus 3 d becomes 55 b equal to d plus 10 it becomes 60 5 and c equal to c plus 17.

So, this thing repeats 4th time third time and a 4th time. So, at the end you get results like this. So, just by simulating this code you get results like this for the variables change

like this. So, the point to note as note there is that every time this loop is executed, you execute them statement by statement. One statement is finished then only the next statements statement starts.


(Refer Slide Time: 16:37)



(ii) Non-Blocking Assignment

- General syntax:
`variable_name <= [delay_or_event_control] expression;`
- The "<=" operator is used to specify non-blocking assignment.
- Non-blocking assignment statements allow scheduling of assignments without blocking execution of statements that follow within the procedural block.
 - The assignment to the target gets scheduled for the end of the simulation cycle (at the end of the procedural block).
 - Statements subsequent to the instruction under consideration are not blocked by the assignment.
 - Allows concurrent procedural assignment, suitable for sequential logic.

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



Fine now let us look at the second type of assignment statement namely the non-blocking assignment statement. Now non-blocking assignment is different from blocking in the way they work, and they are more suitable to be used for specifying sequential circuit behavior as you will see through many examples later.

So, first the general syntax; syntax is very similar variable name and expression, but instead of equal to we use this arrow assignment less than equal to this symbol. This operator specifies non-blocking assignment. Now as the name implies a non-blocking assignment statement, does not block the execution of the statements which follow inside that block. The statements inside the block they can execute concurrently, this is a concept here. It is not that after a statement finishes only then next one will start not that it is not blocking the other statements. So, all these statements can execute together ok.

So, the idea is this. So, if you have a non-blocking assignment, then the assignment to the target which takes place that will happen at the end of the simulation cycle of that block; that means, at the end of the block. So, if you see here the assignment to the target will get scheduled for the end of the simulation cycle means that one run of the loop at the end of the block. So, here we will explain with an example.

So, because it is non-blocking statements after the instruction currently we are looking at executing will not be blocked, they will all be executing together which will allow something called concurrent procedural assignment which is very suitable for modeling sequential logic we shall see these things.

(Refer Slide Time: 18:46)

• This is the recommended style for modeling sequential logic.

- Several “reg” type variables can be assigned synchronously, under the control of a common clock.

```
integer a, b, c;
initial
begin
a = 10; b = 20; c = 15;
end
initial
begin
a <= #5 b + c;
b <= #5 a + 5;
c <= #5 a - b;
end
```

- Initially, a=10, b=20, c=15
- a becomes 35 at time = 5
- b becomes 15 at time = 5
- c becomes -10 at time = 5

All the right hand side expressions are evaluated together based on the previous values of “a”, “b” and “c”. They are assigned together at time 5.

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

So, let us take an example. As I had mentioned this is the recommended style for modeling sequential logic, where a number of reg type variables can be assigned synchronously under the control of a common clock. But here it is just a simple example I am taking, here this is just a test bench I am writing there is no clock. There were 3 variables a b and c. So, in one initial block I am initializing them to 10 20 and 15. And in the second initial initialize block I am using non-blocking statement with some delay identifier. So, the idea is that when there are non-blocking statement inside a procedural block.

Then all these 3 statements are executing concurrently, meaning the right hand side expressions they will be evaluating together using the current values of a b and c. And after the delay which is specified they will all be concurrently assigned to the variables on the left. So, in this example let us say a is 10 b is 20 c is 15. So, on the right hand side b plus c a plus 5 and a minus b will be evaluated first, what is b? See b plus is 35. What is a plus 5? It is 15, what is a minus b? It is minus 10. So, they will be assigned all together at time 5. So, a will become 35 b will become 15 c will become minus 10.

So, you see the final value that a b c gets here is quite different from what it got when you used a blocking assignment statement. So, when you use blocking and when you use non-blocking the results are different. So, you should be careful when you are using it to model some behavior of a system or a circuit. So, as I said that all the right hand side expressions are evaluated together based on the previous values of a b c not these latest values. And they are assigned all together concurrently at time 5 right.

(Refer Slide Time: 21:06)

The slide features a light yellow background with a blue header and footer. In the center, there is a light blue box containing Verilog code. To the right of the code is a purple box with text. Below the code is a pink box with text. The footer contains logos for IIT Kharagpur and NPTEL, along with the course title 'Hardware Modeling Using Verilog' and the slide number '11'.

```
always @(posedge clk)
begin
  a <= b & c;
  b <= a ^ d;
  c <= a | b;
end
```

Recommended style for modeling synchronous circuits, where assignments take place in synchronism with clock.

All assignments take place synchronously at the rising edge of the clock.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 11

So this is a more common way where we use this kind of non-blocking assignment with a clock this statement. Means whenever there is a positive edge of a clock you compute these expressions.

And assign them to these variables concurrently. So, all assignments take place synchronously at the rising edge of a clock you see. This is quite natural from hardware point of view because in any hardware register there is some kind of a clock signal. So, whenever there is a clock and there is a load the value input value gets loaded. So, here also something similar can take place the right hand side can be evaluated and the values can be made available to the input of the registers, as soon as the clock comes all those input values will get loaded in the registers that will happen exactly when the clock comes ok.

So, that feature can be modeled using this posedge clock facility. And as I said this is the recommended style for modeling synchronous sequential circuits where there is a clock,

and you are wanting assignments to take place in synchronism with the clock right. This is the recommended style.

(Refer Slide Time: 22:20)

Swapping values of two variables "a" and "b"

```
always @(posedge clk)
  a = b;
always @(posedge clk)
  b = a;
```

- Either a=b will execute before b=a, or vice versa, depending on simulator implementation.
- Both registers will get the same value (either "a" or "b").
 - Race condition.

```
always @(posedge clk)
  a <= b;
always @(posedge clk)
  b <= a;
```

- Here the variables are correctly swapped.
- All RHS variables are read first, and assigned to LHS variables at the positive clock edge.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 13

Now let us take on one example, suppose we want to swap the value of 2 variables a and b. Swapping means if a is 10 and b is 20. So, we want to make a 20 and b 10 interchange their values. Let us look at various styles of doing it, and let us see that whether swapping actually takes place or not ok.

This is the first example I am taking using non-blocking assignment with 2 always blocks. Both are activated at the posedge of clock, there is no delay. So, at time t equal to 0 just immediately after the clock edge comes this will be executed. So, you see because these are concurrent blocks a equal to b, and b equal to a they are actually executing concurrently. But you see when you are simulating this, depending on the simulator simulated will be either doing this assignment first and then this or the reverse. So, if a equal to b is done first, then b will be going to a and that value of b will again be copied to b. So, both a and b will be getting the value of b that was there. But if this statement executes first, then the value of a is get copied to b and that same value remains in a.

So, both the registers will finally, get this same value, but it can be either a or b you do not know. It depends on the simulator how this simulator actually schedules these 2 operations executed execution. And this is called a race condition. Depending on the relative order of execution the final result can be either a or it can be b. Let us look

another example. Same one instead of non-blocking instead of blocking I am using non-blocking assignments. Say here there is no problem, because these assignments will take place only when positive edge of clock comes. And they will take place together. So, the right hand side will be evaluated.

So, right hand side b right hand side a that values will be taken, and they will be assigned to a and b at the posedge of the clock. So, there is no question of indeterminism like in blocking assignment that was happening here. Because here there is nothing that says that whenever the clock comes only then to be assign, blocking says it will be done one by one. It can be either this or this or this first this or this depending on the order of execution. So, if you use non-blocking statement then the variables will be correctly swapped, because the right hand side variables are read first let us say equal to 10 b equal to 20 they will be read first.

And whenever the positive clock edge comes 20 will be assigned to a and 10 will be assigned to b. So, swapping will actually take place.

(Refer Slide Time: 25:30)

Trying to swap using blocking assignment

```
always @(posedge clk)
begin
  a = b;
  b = a;
end
```

- Both "a" and "b" will be getting the value previously stored in "b".

```
always @(posedge clk)
begin
  ta = a;
  tb = b;
  a = tb;
  b = ta;
end
```

- Correct swapping will occur, but we need two temporary variables "ta" and "tb"

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

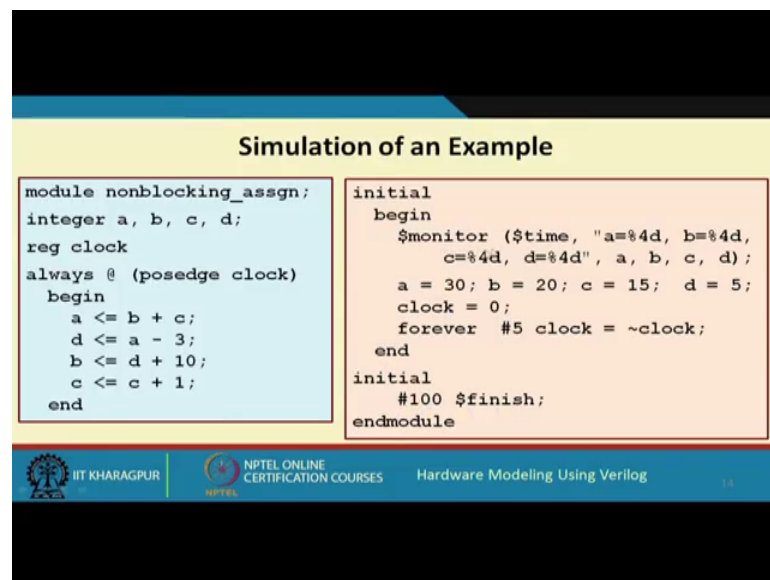
Let us take another example. Suppose we are using blocking statement to try to swap. Suppose I write a equal to b, b equal to a what will happen? Inside a same block, just try to see what will happen here. First this statement will be executing b will be copied to a, and then this will be executing a will be copied to b. So, both a and b will be finally,

getting the value of b. Because this is the first statement to execute, the value of value of b if it is 20 20 will be copied to a.

So, a will also become 20. So, 20 will also be copied to b again. So, both a and b will be 20 the previous value of b. So, using blocking statement if you want to really do a swap you can do it like this, by using 2 additional temporary variables t a and t b. You first this assign a to the temporary variable t a, then b 2 t b then t b to a and t a to b. In this way swapping will correctly take place, but of course, you see you have to require you use 2 additional variables. So, this example shows you that using non-blocking assignment for certain application can be very convenient, as this swapping example is one of them it shows you fine.

So, let us similarly simulate one example using non-blocking assignment; same example that is showed for blocking, but here I use non-blocking assignment. So, my test bench is also similar, but since here we use a clock. So, I am applying a clock here. So, what you are doing?

(Refer Slide Time: 27:26)



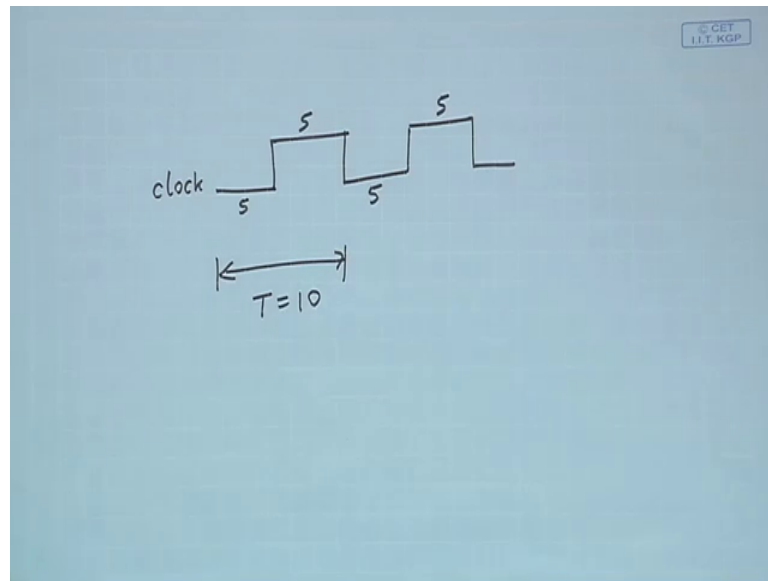
The slide is titled "Simulation of an Example" and contains two code blocks. The first block is a Verilog module named 'nonblocking_assgn' with variables 'a', 'b', 'c', 'd' and a register 'clock'. It uses non-blocking assignments to update 'a', 'b', 'c', and 'd' based on the current values of 'b', 'a', 'd', and 'c' respectively, triggered by the rising edge of 'clock'. The second block is a test bench that initializes 'a=30', 'b=20', 'c=15', and 'd=5', sets 'clock=0', and uses a 'forever' loop to toggle 'clock' every 5 time units. A '\$monitor' statement is used to print the values of 'a', 'b', 'c', and 'd' at each clock edge. The slide footer includes the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and the text 'Hardware Modeling Using Verilog'.

```
module nonblocking_assgn;
integer a, b, c, d;
reg clock
always @ (posedge clock)
begin
a <= b + c;
d <= a - 3;
b <= d + 10;
c <= c + 1;
end
endmodule

initial
begin
$monitor ($time, "a=%4d, b=%4d,
c=%4d, d=%4d", a, b, c, d);
a = 30; b = 20; c = 15; d = 5;
clock = 0;
forever #5 clock = ~clock;
end
initial
#100 $finish;
endmodule
```

We are using the monitor to just mention what are the variables to print. Then we initialize the variables, clock also is initialized to 0. And in a far away loop we are generating the clock signals. Forever is a loop which goes on indefinitely we say that after a delay of 5 you do clock equal to not clock. So, the clock signal will be generating like this.

(Refer Slide Time: 27:53)





Clock was initially 0.

So, after a delay of 5 it becomes 1, after delay of 5 it becomes 0. After delay of 5 it again becomes one like this. So, a clock will be generated with a time period of 10, right. So, this forever statement generates this clock and we continue simulation till 100 there is another initial block for you say at 100 you finish. Let us see the simulation output for this example.

(Refer Slide Time: 28:31)

Simulation Results				
0	a = 30,	b = 20,	c = 15,	d = 5
5	a = 35,	b = 15,	c = 16,	d = 27
15	a = 31,	b = 37,	c = 17,	d = 32
25	a = 54,	b = 42,	c = 18,	d = 28
35	a = 60,	b = 38,	c = 19,	d = 51
45	a = 57,	b = 61,	c = 20,	d = 57
55	a = 81,	b = 67,	c = 21,	d = 54
65	a = 88,	b = 64,	c = 22,	d = 78
75	a = 86,	b = 88,	c = 23,	d = 85
85	a = 111,	b = 95,	c = 24,	d = 83
95	a = 119,	b = 93,	c = 25,	d = 108

```
Initially:  
a=30, b=20, c=15, d=5  
always @ (posedge clock)  
begin  
  a <= b + c;  
  d <= a - 3;  
  b <= d + 10;  
  c <= c + 1;  
end
```

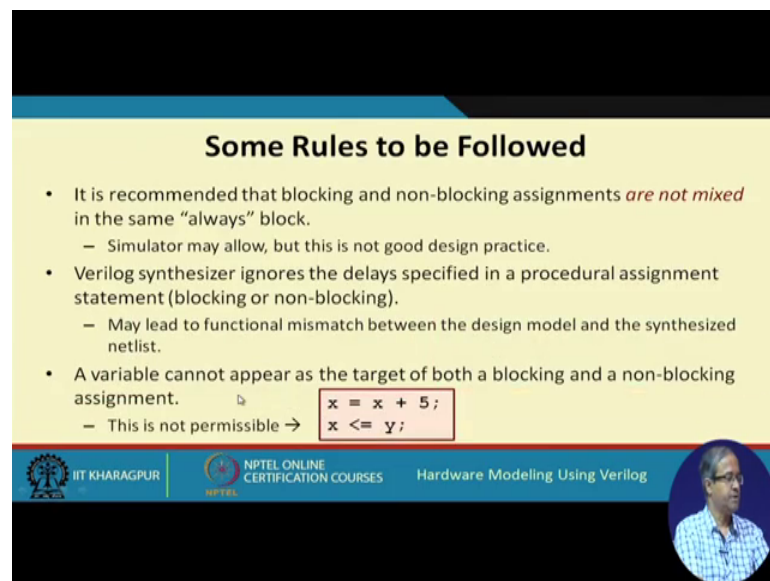
  Hardware Modeling Using Verilog

So, I have shown this side by side for convenience, initially a 30 b 20 c 15 d 5 that is the first value which is printed at time t equal to 0. Then just this is the statement positive edge of the clock is coming after time 5 clock was 0. So, after time 5 the first clock edge comes right.

So, at time 5 the first change will take place, the positive edge comes at time 5. So, b plus c will be assigned to a a minus 3 will be assigned to d, d plus 10 to b and c plus 1 to c, what is b plus c? B plus is 35. A minus 3 is 27. D plus 10 is 15. C plus 1 is 16. So, all the values are assigned together you see 35 15 16 27. Now next clock positive edge will be coming after the time period 10. So, next one change will happen at 15.

So, again you do this with these new values, b plus c 15 plus 16 31; A minus 3 32; D plus 10 37; C plus 17. You see all these values are assigned parallelly 31, 37 17 32 and these repeats up to the time 100. So, 95 is the clock edge that you get d for 100 last change will take place at 95. So, this is how this simulation result shows.

(Refer Slide Time: 30:10)



Some Rules to be Followed

- It is recommended that blocking and non-blocking assignments *are not mixed* in the same “always” block.
 - Simulator may allow, but this is not good design practice.
- Verilog synthesizer ignores the delays specified in a procedural assignment statement (blocking or non-blocking).
 - May lead to functional mismatch between the design model and the synthesized netlist.
- A variable cannot appear as the target of both a blocking and a non-blocking assignment.
 - This is not permissible →

```
x = x + 5;  
x <= y;
```

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

So, there are a few rules which you should remember when you are using this kind of blocking and non-blocking statements in a procedural block.

And these are summarized here you see, the first thing I told you is that blocking statement is the recommended style for combinational circuits for specifying combinational logic. And non-blocking statement is a recommended style for specifying

synchronous sequential circuit. So, the first thing to remember is that you should not mix blocking and non-blocking assignments together in the same block. Well, it is not that it is not allowed I shall show you some examples later where you will show that what will happen if you mix them. But it can make the things complicated. For you simulator or the synthesizer may allow, but this is certainly not a good design practice you should avoid this. You use either blocking or non-blocking, inside a particular procedural block.

The second thing you remember is that this delays that you show or you specify these are only for simulation. So, when you are doing synthesis. So, all these delays will be ignored. So, the design which is working perfectly in simulation may not work exactly exactly as you just want them to work in the final synthesized hardware. So, that is why you should keep these things in mind. Suddenly say a particular variable you cannot you cannot have it as the target of a blocking as well as non-blocking assignment together in 2 blocks in the same module. If you do it the simulator or the synthesizer will give you a warning on error message. So, with this we come to the end of this lecture, where we had looked at the mainly the differences between the blocking and the non-blocking assignment styles.

So, we shall be continuing our discussion on these 2 styles over the next few lectures, because this is very important in modeling. And unless you understand clearly the differences between the 2 and what will happen if various types of combinations use it will be difficult for you to understand the interpretation of the semantics of different structures. So, we shall be continuing with this in the next lecture as well.

Thank you.