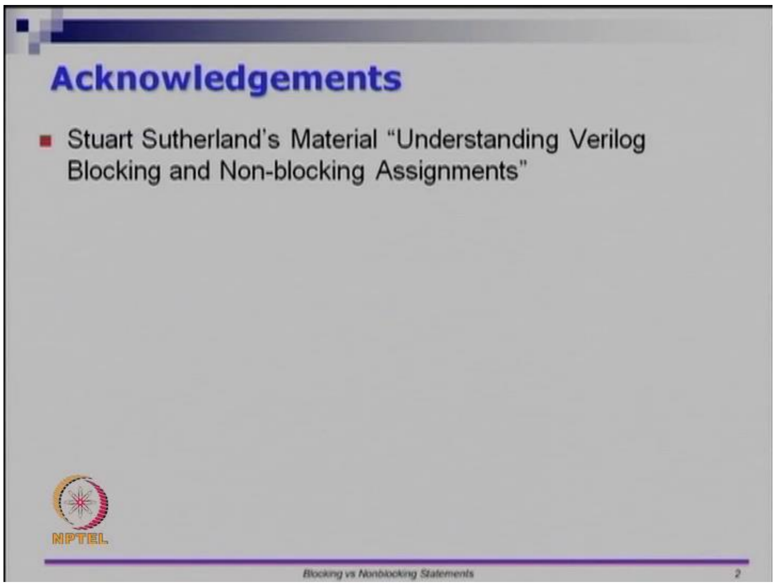
Digital Circuits and Systems

Prof. Shankar Balachandran
Department of Electrical Engineering
Indian Institute of Technology, Bombay
And
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 46
Verilog Modeling (Assignment Statements)

We are in module 46 now, in this module what we are going to look at is, we are going to look at different kind of assignments that is there in Verilog. So, I am taking this slide d 2 and in module 47, I will connect pipelining with Verilog. So, we have been doing some bit of black magic so all along.

(Refer Slide Time: 00:36)



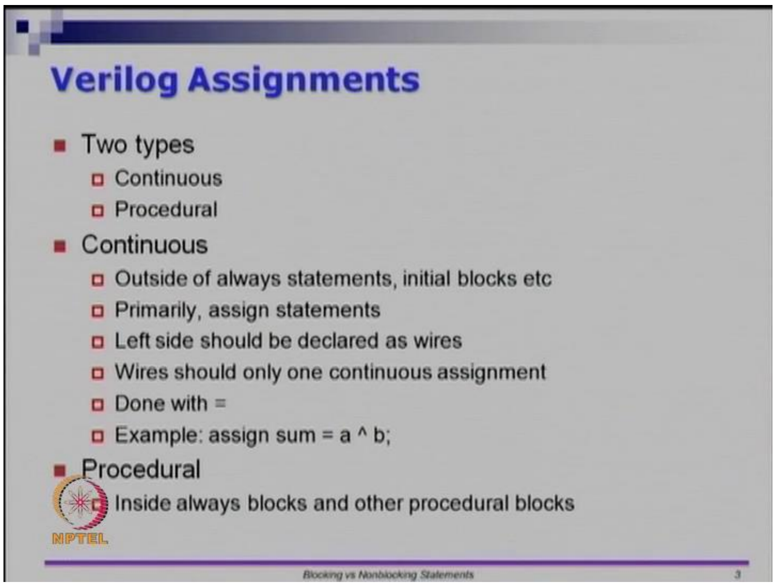
Acknowledgements

- Stuart Sutherland's Material "Understanding Verilog Blocking and Non-blocking Assignments"

Blocking vs Nonblocking Statements 2

So, the material that I am going to present here is partly based on Stuart Sutherland's material on understanding Verilog blocking and non blocking assignments, some of the examples are taken from this material and some of the examples are made otherwise.

(Refer Slide Time: 00:51)



Verilog Assignments

- Two types
  - Continuous
  - Procedural
- Continuous
  - Outside of always statements, initial blocks etc
  - Primarily, assign statements
  - Left side should be declared as wires
  - Wires should only one continuous assignment
  - Done with =
  - Example: assign sum = a ^ b;
- Procedural
  - Inside always blocks and other procedural blocks

Blocking vs Nonblocking Statements 3

So, in Verilog we have been doing a bit of this black magic, so there were multiple assignments and I was using equal to versus this token less than or equal to or this less than symbol followed by equal to at different places without explaining what they are actually doing. So, in this video I will explain what these different statements are and the implications of these statements.

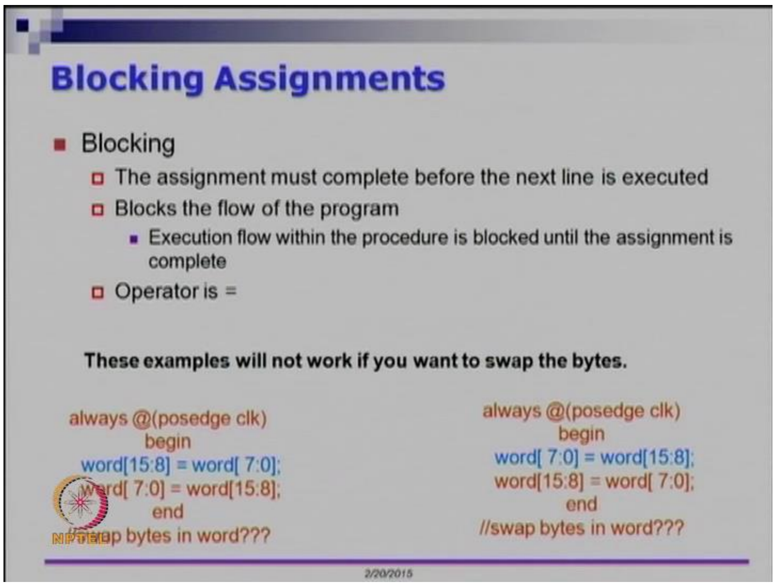
So, first of all assignments in Verilog are of two different kinds, there is something called continuous assignment, there is something called procedural assignment. The continuous assignment is something that happens outside of always statements and outside of initial blocks. So, primarily all the assignments that you do with assign statements are actually continuous assignments or anything that you do with instantiation of logic, which you connect other wires is also a continuous assignment.

So, the left side of a continuous assignment is usually declared as a wire and wires can have only one continuous assignment, it is usually done with equal to. So, you know all of these statements, assign a equals b and c. So, when you do that, assign is the key word, so a is the variable on the left side, a should be declared as a wire. And you can have only one assign a equals a plus b or a and b, you cannot have assign a equals a and b and also another assign a equals a and c you cannot do that.

Because what that would mean is there are two different gates that are driving the same wire that is not acceptable. So, wires can have only one continuous assignment that is assigning to it, the assignment is done with equal to, you cannot use less than and equal to token for this. So, example is assign sum equals a XOR b, so sum is supposed to be declared as a wire, this is the continuous assignments statement, this is the continuous assignment operator and there is an expression on the right side, so this is called continuous assignment.

Whereas, procedural assignments are those that happen inside procedural blocks, the procedural blocks that we have seen so far are initial statements which is not very useful other than test benches, but always blocks are procedural statements. So, this is equivalent to procedures as we know in sequential programming languages. So, let me take a small d 2 and look at procedural assignments statements. So, we are talking about procedural assignment statements there are two kinds of procedural assignment statements.

(Refer Slide Time: 03:41)



Blocking Assignments

- Blocking
  - The assignment must complete before the next line is executed
  - Blocks the flow of the program
    - Execution flow within the procedure is blocked until the assignment is complete
  - Operator is =

These examples will not work if you want to swap the bytes.

```
always @(posedge clk)
        begin
  word[15:8] = word[ 7:0];
  word[ 7:0] = word[15:8];
        end
//swap bytes in word???
```

```
always @(posedge clk)
        begin
  word[ 7:0] = word[15:8];
  word[15:8] = word[ 7:0];
        end
//swap bytes in word???
```

2/20/2015

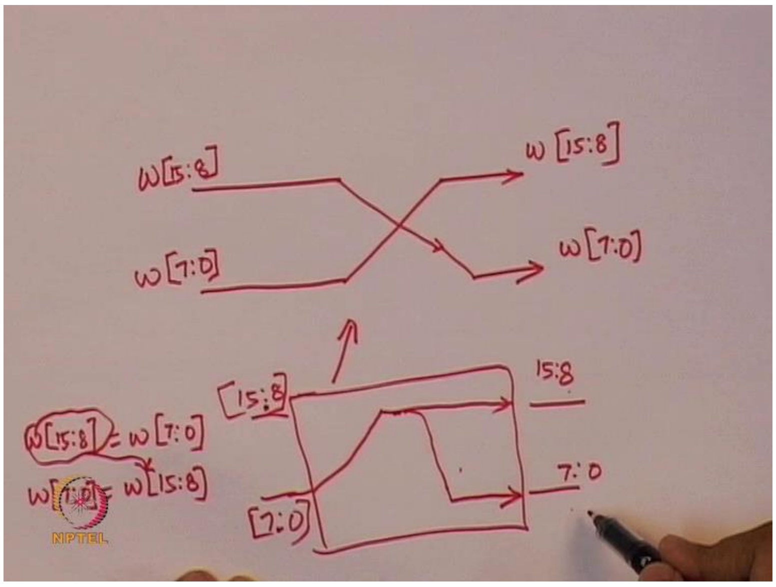
The first kind is called blocking, the second kind is called non blocking, so the first kind blocking is essentially done with equality symbol. So, you can see that here in the examples that we come to in a little while. So, the first thing is blocking assignment is an assignment that happens within a procedural block. So, always is a procedural block and assignment that you do with equal to inside an always block is called a blocking assignments.

So, a blocking assignment happens in this form, the assignment must complete before the next line is executed. So, let us look at this procedural block, if I look at this statement, there is something on the left side we are saying it is equal to something on the right side, what that means is, until I execute the blue line, this red line will not get executed. So, this is similar to how your sequential programming languages work. If I write a 10 line program, the line 1 must execute before line 2 and line 2 must execute before line 3 and so on.

So, if I put this equality symbol, what that means is, I have to execute this line before I execute this red line, the same thing is true here also. I have to execute the red blue line before I execute the red line and more importantly, always this red lines execution follows the blue lines execution. So, if I have 5 lines of code, line 1 will get executed before line 2, so this is the notion of a procedural assignment. In a procedure, if I have 5 lines, then line 1 should get executed before line 2, line 2 before line 3, line 3 before line 4 and so on.

And the assignments, if they are using blocking assignment which is done using equal to, then the execution of the flow is blocked until the assignment is complete. So, now, let us take a look at this example, I will explain what this implication of a blocking assignment is. Let us say I want to take a 16 bit bus and I want to swap the contents of the bus. What I want to do is, I want to take the top half of the bus, put it in the lower half and take the lower half and put it in the upper half. So, let me draw a picture of what I want to do, so I have a bus that is coming in.

(Refer Slide Time: 06:07)



So, let us look at this picture here, so I have a bus that is coming in, so this is a bus that has 15 to 8 and this is the bus which has 7 to 0, what I want to do the problem statement is, I want to swap the contents of the bus, this is what I want to do. So, this is till 15 colon 8, this is still 7 colon 0, but what I want is when the clock pulse comes I want 7 to 0 to appear in 15 to 8 and 15 to 8 to appear in 7 to 0, so this is the problem statement.

For this problem statement, let us look at this piece of code that I have written ((Refer Time: 06:53)). So, in this piece of code I have always at posedge of clock, you know that if I do always at posedge of clock, when the clock pulse comes in, these actions are supposed to be run. How are these actions going to be executed? First of all it is a procedure, so the first line will get executed before the second line, so the blue line will get executed before the second line, the red line.

So, let us look at the blue line, in the blue line according to... So, the assignment that you have is a blocking assignment. What that means is, you take word 7 colon 0, you assign it to 15 colon 8, until that assignment is over you cannot go to the red line, which means word of 7 colon 0 is copied on to 15 colon 8. So, I want things to be swap, so word of 7 colon 0 got copied on to 15 colon 8. Now, when I refer to ((Refer Time: 07:50)) 7 colon 0 it is already taken a new value, because of that this word of 7 colon 0 is taking the new value of word of 15 colon 8.

So, whenever you see a equal to write of this kind, you think of this side as the old one and this side as the new one. So, this is old value of 7 colon 0, this is equality symbol and this is value of word 15 colon 8. So, the old value of 7 colon 0 becomes the new value of 15 colon 8, but because it is a blocking assignment the new value of 15 colon 8 will appear here. Because, unless this statements assignment is over, the red line cannot execute, which means this 15 colon 8 will appear here as 15 colon 8, the new value appears here and that gets copied to 7 colon 0, so let see the implication of it.

So, what I want is let us look at this picture again, we have this what we wanted as a sequence what we did was word of 15 colon 8, we said it is equal to word of 7 colon 0 and we said word of 7 colon 0 is equal to word of 15 colon 8, let us see the implication of this. So, this one is a blocking assignment, so what it does is, so you want this is the output 15 colon 8 and this the output 7 colon 0 and these are the 2 inputs, it is 15 colon 8 and 17 colon 0 and we are designing a black box, we are designing a circuit for this.

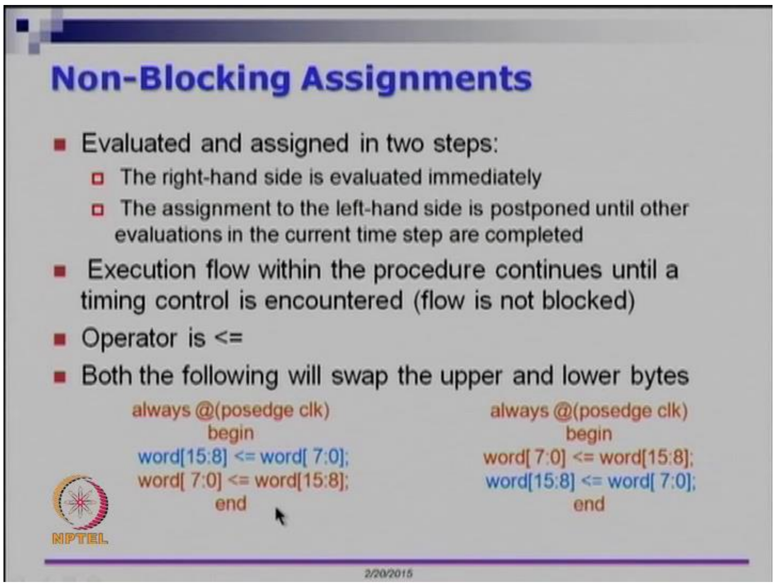
So, what this does is 7 colon 0 gets copied to 15 colon 8, so 7 colon 0 is ready to copy to 15 colon 8, so that is what this assignment does. But because it is blocking, this value 15 colon 8 is immediately available for the next statement and what is this going to be... So, this is going to take whatever value of 15 colon 8 is and what is the value of 15 colon 8 now, it is this value it takes this value and gives it to 7 colon 0.

So, what this as essentially done is, it is not swapping 15 colon 8 and 7 colon 0, it did not do that, instead the 7 colon 0 is copied on both the half of the buses. So, again if we go back to the slides ((Refer Time: 10:29)), this left half this always block is incorrect. Because, at the end of this clock pulse both 7 colon 0 and 15 colon 8 will have the old value of 7 colon 0, before the clock pulse came whatever was there in 7 colon 0 that will appear in both the top half and the bottom half.

If I change the order around that is what I have done in this picture, so in the right side I have done that. So, I have put word of 7 colon 0 line before the word of 15 colon 8 line. So, this is not going to be any different, what it is going to do is, whatever was there before the clock pulse came in at 15 colon 8 those 8 bits it will take that and put it in both 7 colon 0 and 15 colon 8. So, this is the meaning of blocking assignment, a blocking assignment is one in which, first of all it is inside a procedural block like an always block.

So, an equality symbol that is in the assignment statement, assign statement is not a blocking assignment, it is a continuous assignment and equality symbol within an always block is a blocking assignment. The moment you see a blocking assignment, the way you should think about that is, whatever you is in the right side of it will be immediately available to the left side and any subsequent use in the following lines will use that new value that was assigned. So, that was the problem here and this is not helping us swap, I still want to be able to swap the contents of the top half and the bottom half.

(Refer Slide Time: 12:07)



Non-Blocking Assignments

- Evaluated and assigned in two steps:
  - The right-hand side is evaluated immediately
  - The assignment to the left-hand side is postponed until other evaluations in the current time step are completed
- Execution flow within the procedure continues until a timing control is encountered (flow is not blocked)
- Operator is <=
- Both the following will swap the upper and lower bytes

```
always @(posedge clk)
      begin
word[15:8] <= word[ 7:0];
word[ 7:0] <= word[15:8];
      end
```

```
always @(posedge clk)
      begin
word[ 7:0] <= word[15:8];
word[15:8] <= word[ 7:0];
      end
```

2/20/2015

For that I can do something called non blocking assignment, the first thing I want to show is, what is in the bottom. Let us look at the always block that is written up, so instead of equal symbol token we have put less than equal to. So, that is less than followed by equal to, we have put two tokens, less than equal to in both these places and my claim is that, this always block as well as this always block will actually swap the contents of the top half and the bottom half of the buses.

So, let us see how this is done, first of all any non blocking assignments, so the word non blocking means, this assignment should not block the assignments that are following the assignment. So, what that means is, this is a non blocking assignment, this blue line is a non blocking assignment, but this non blocking assignment should not itself block the execution of the red line. So, the blue line being above the red line should not block the execution of the red line that is the meaning of non blocking assignment.

So, let us take this a little slowly, it is non blocking assignments are evaluated and assigned in 2 steps. In blocking, as soon as the evaluation is over, you assign it to the left side whereas, in non blocking the evaluation happens independently and the assignment happens independently. The first thing is the right side is evaluated immediately. So, if you go and look at this part, the right side is word 7 colon 0 and word 15 colon 8. Evaluating that means, in some sense you get a copy of those two, you will do this for all the non blocking assignments that are on the right side.

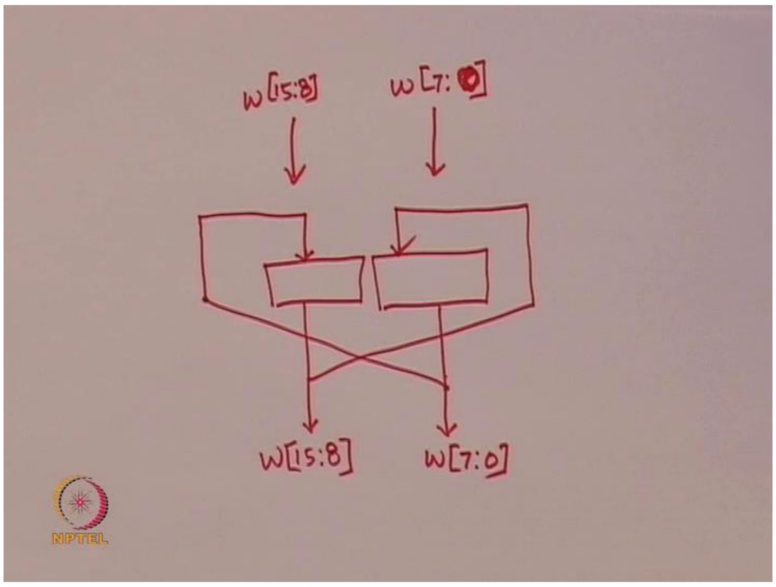
So, when you look at this sequence of statements, the right side is evaluated immediately and the assignment to the left side is postponed. So, that is the keyword it is postponed, until all the other evaluations in the current time step are completed. So, when the clock pulse comes in that is the current time step. What are all the different evaluations that have to be there? I need to evaluate 7 comma 0 and I have to evaluate 15 colon 8.

What are the assignments that I have to do? I have to do assignments to 15 colon 8, I have to do assignments to 7 colon 0. So, all the right hand evaluations happen immediately and the evaluations are all done in one go and finally, all the assignments happen in one go. So, it is equivalent to evaluating all the right hand side in one place, fine if finish the evaluation and then in one go do the assignments to the left side.

So, the operator that you use is called is less than or equal to token and the execution of the flow will happen, until either the flow itself is over which means... So, I will go and look at this, I will remember the word 7 colon 0 I will go to the next line that is the meaning of this. I will remember what word of 7 colon 0 had, I will go to the next line that is the meaning of the non blocking statement.

Again I will remember what 15 colon 8 had, I will go to the next line. At this point I am at the end of this, so but I have remember, what 7 colon 0 and 15 colon 8 are. Now, I go and do all the assignments for 15 colon 8, I should take whatever I remembered in 7 colon 0. So, and for 7 colon 0 I should take whatever I remembered in 15 colon 8, so that is the meaning of this. So, if you think about this the hardware equivalent is like this.

(Refer Slide Time: 15:53)



So, the hardware equivalent is, there is a bus that is coming in word 15 colon 0 and word 7 colon 8. Because, this is inside the posedge of clock, there is a register for both of them 7 colon 0 as well as 15 colon 8. So, what this does is, at every posedge of clock, so this is my actual projected output 15 colon, this is supposed to be 15 colon 8 and 7 colon 0 is 15 colon 8 and this is the projected output 7 colon 0.

What this non blocking assignment does is, it takes 15 colon 8 gives it as input to 7 colon 0 and it takes 7 colon 0 and gives it as an input to 15 colon 8. So, this is equivalent to swapping the contents of 15 colon 8 and 7 colon 0 and this every clock pulse, this will keep swapping the contents from lower half to the upper half and upper half to the lower half. Let us go back to the picture again, what we have is, we have the values ((Refer Time: 17:09)) that are there in 7 colon 0 and 15 colon 8, we remember those values and then we assign them to the left side.

So, in the picture that I did it is the outputs of the flip flops, let us look at the picture again, the outputs of these flip flops or these registers are the once that I am remembering, I am going to place them as inputs here. But these inputs will get committed only when the next clock pulse comes in. So, what you see in this slide is this one ((Refer Time: 17:40)) I evaluate all of them all the things there on the right side and in one go, I assign all of these things on the left side.

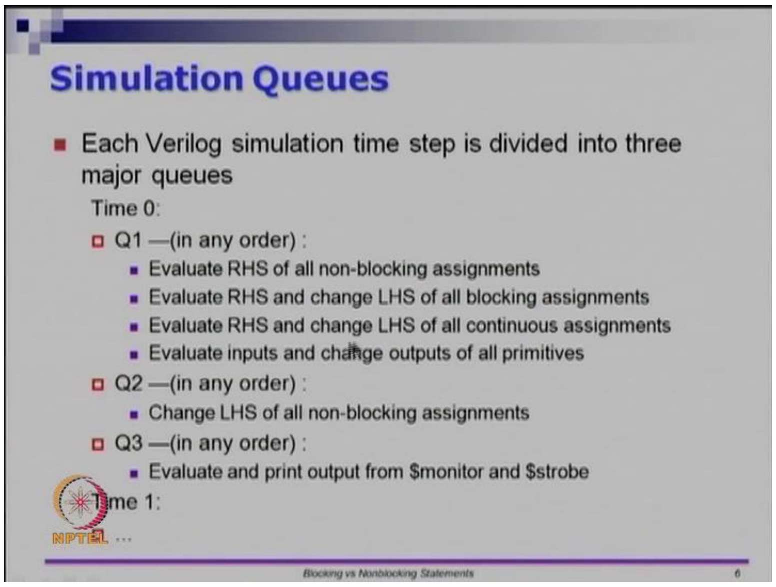
So, because of that there is something that is interesting that is happening. So, let us look at this always block and this always block, what I have done is I have taken these two statements and I swap the order in which these statements are. So, earlier I showed that if I swap the order with a blocking assignment, then either we get a copy of 15 colon 8 or we get a copy of 7 colon 0. Now, I establish the fact that the left side is actually swapping, I want to now show that the right side is also actually swapping.

So, let us see the logically what the right side is supposed to do. In this always block on the right side, let us look at the assignment statements which are inside there, they are again both are non blocking assignments. So, the way in which non blocking assignments have to be dealt with this, whenever a clock pulse comes in, I am going to remember 15 colon 8 and 7 colon 0. Once, I remember it I am now at the end of the block, I start from line 1 I look at line 2, I am end of the block.

I have evaluating everything on the right side, I go and do all the assignments that are required for the left side. So, the requirement I required for the left side is 7 colon 0 needs the previous value of 15 colon 8 and 15 colon 8 needs the previous value of 7 colon 0. Because, that is the meaning of this non blocking assignment, I want the previous value that was there when the block was entered. So, which means this will also do swapping, so if this does not get, if you are not able to understand this immediately, so go to the beginning of this slide and look at the argument that I made for, why this one is swapping, the same argument actually applies for this one.

So, the key thing is the order of non blocking assignments inside an always statement does not matter. So, I have two non blocking assignments here and I have the same two non blocking assignments, the order is reversed. The logic that is implied from here and the logic that is implied from here are actually the same. Whereas, if I put it as blocking assignments, the values where the logic is different.

(Refer Slide Time: 20:04)



Simulation Queues

- Each Verilog simulation time step is divided into three major queues

  Time 0:

  - Q1 —(in any order) :
    - Evaluate RHS of all non-blocking assignments
    - Evaluate RHS and change LHS of all blocking assignments
    - Evaluate RHS and change LHS of all continuous assignments
    - Evaluate inputs and change outputs of all primitives
  - Q2 —(in any order) :
    - Change LHS of all non-blocking assignments
  - Q3 —(in any order) :
    - Evaluate and print output from $monitor and $strobe

  Time 1:

  ...

Blocking vs Nonblocking Statements 6

So, let us see how a simulation queue works, so every Verilog simulation time step is divided into technically actually four major queues, but more importantly the three major queues, the first queue and this can happen in any order. So, you have various blocks, various always statements, various assign statements, various instantiations and what not all of them can executed in any arbitrary order, this is what is giving you the modeling of concurrency in a digital circuit.

So, in a digital circuit all the gates are simultaneously looking at what there inputs are and they are output will fire. So, Verilog is essentially modeling that, so in time step 1 in some order of all the blocks these things are done within each and every block evaluate the RHS or the Right Hand Side of all the non blocking assignments. Evaluate the RHS and change the LHS of all the blocking assignments, evaluate the RHS and change the LHS of all continuous assignments and evaluate inputs and change outputs of all primitives.

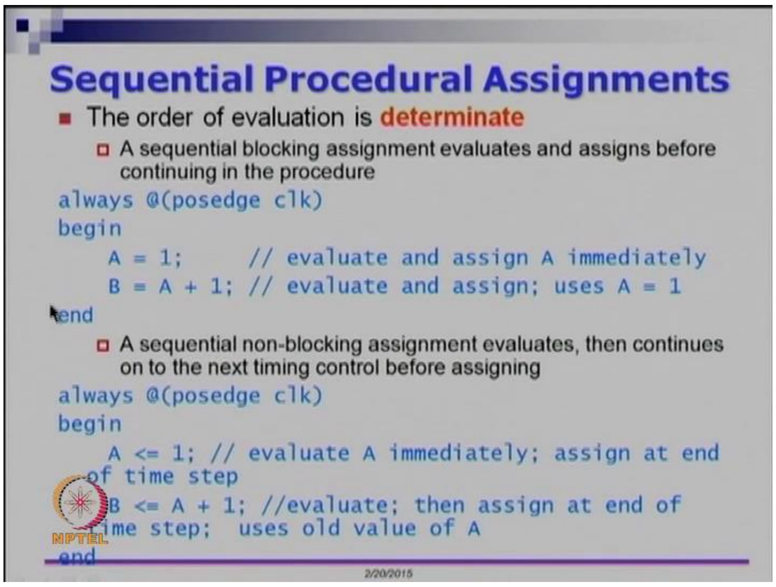
This is the order in which various concurrent statements are going to get executed. So, let us see this is the very crucial thing, so if I am going in time step. So, let us say I am a Verilog time step 5 and some inputs are changed I am going to see what is going to be the set of changes at time steps 6 between 5 and 6 we have this so called micro steps that is what this micro steps are. So, at Q 1 is a micro step what it does is evaluate the right hand side of all the non blocking assignments first. So, which means go and take all the values that are on the right side keep them assign, because this may get destroyed later within the procedure itself it may get destroyed. Keep it aside and evaluate the right hand side in change the LHS of all the blocking assignments.

Because, blocking says I do not have to wait this statement evaluates the right side and until this is over the following statements cannot execute that is the meaning of blocking, which means you have to immediately assign to the LHS. Evaluate they are right hand side and change the left hand side of all the continuous assignments, evaluate the inputs and change the outputs of all the primitives.

Once this is all over, then you go to Q 2 which is the second micro step at this point all the non blocking assignments will get there left side changed. And finally, any display or monitor or stroke statements will print the values, so this is only for printing Q 3 is only for printing the two major steps are these two, Q 1 and Q 2, Q 1 is all the non blocking assignments are evaluated and set aside.

And the actual assignments happen only in Q 2, but before doing that it is done in these things are done, evaluate the right hand side and change the LHS, evaluate the right hand side it change LHS of all continuous assignments, evaluate the inputs and change the outputs of all primitives.

(Refer Slide Time: 23:17)



Sequential Procedural Assignments

- The order of evaluation is **determinate**
  - A sequential blocking assignment evaluates and assigns before continuing in the procedure

```
always @(posedge clk)
begin
     A = 1;      // evaluate and assign A immediately
     B = A + 1; // evaluate and assign; uses A = 1
end
```

  - A sequential non-blocking assignment evaluates, then continues on to the next timing control before assigning

```
always @(posedge clk)
begin
     A <= 1; // evaluate A immediately; assign at end
of time step
     B <= A + 1; //evaluate; then assign at end of
time step;  uses old value of A
end
```

2/20/2015

So, let us see the implication of all of these things, I have several examples if from this slide onwards. Let us look at this always block, always at posedge of clock A equals 1 and B equals 1, if you go by the procedure that I explain before, first of all these two are blocking assignments. So, because this is the blocking assignment it the meaning of that is evaluate 1 and assign it A immediately. So, A is assigned immediately, once is assigned immediately when it is blocking.

So, 1s is assigned only then you can go to the second line at this point the value of A being 1 is already available. So, A will become 1 now 1 plus 1 is 2, so B will get 2, so that is the meaning of this procedural block. So, what we have is at time step 0 this procedural block gets executed at that point this is 1, it gets immediate assign to this. And since that value is immediately available here it goes here.

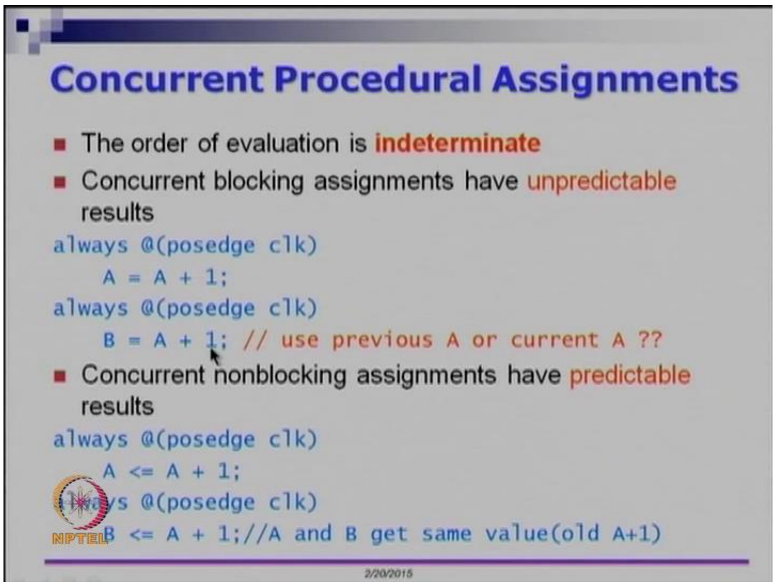
So, this equality symbol that you are seeing is like a software programming languages when you say equal to right, whatever you have in 5th line that value should be available in the 6th line, whatever computation do the 6th line that value must be available in the 7th line and so on. So, it is essentially that block all the software programming languages the equality that you see there are all blocking.

Whereas, if you go and look at this one, A is 1 and B is A plus 1 let us see what the meaning of this is. So, because both of them are non blocking assignments, the first thing that happens is whatever is on the right side that is evaluated first. So, there are two non blocking assignments which means one gets evaluated there is really no evaluation that is necessary. So, I will have to remember that A should get a value of 1, then A plus 1 also gets evaluated in the same time, whatever the current value of A is you add 1 to it and that should go to B.

So, if the current value of A is 5, 5 plus 1 should go to B I will remember that. And now I come to the end at this point I am ready to make the assignments, what are the assignments I am going to make to A I remember that I should be assigning 1. So, I will put 1 to B I remember that I should assign the value 5 plus 1 which is 6, so I will assign 6. So, you can see the difference between here and here.

Here this block will always give you A equals 1 and B equals 2 whereas, here you will get A equals 1 and B equals a previous value of A plus 1. So, that is the meaning of the sequential procedural assignment.

(Refer Slide Time: 26:19)



Concurrent Procedural Assignments

- The order of evaluation is indeterminate
- Concurrent blocking assignments have unpredictable results

```
always @(posedge clk)
    A = A + 1;
always @(posedge clk)
    B = A + 1; // use previous A or current A ??
```

- Concurrent nonblocking assignments have predictable results

```
always @(posedge clk)
    A <= A + 1;
always @(posedge clk)
    B <= A + 1;//A and B get same value(old A+1)
```

2/20/2015

Let us look at concurrent procedural assignments, in concurrent procedural assignments what we have is. So, in the sequential assignment I have two assignments that are still within a single block within a single always block. Now, we are going to reason about what is going to happen across the multiple always blocks, remember 1 always block here, this is suppose to model some circuit, this is also suppose to model 1 some circuit.

If I have a statement like this A equals A plus 1 within one always block and B equals A plus 1 in another always block, let us see what the meaning of this is at time step 0 the clock pulse comes in and I go to the each and every procedure. And the order in which the procedure is executed is not suggested by the Verilog standard, Verilog standard does not say anything about if this always block appears in the file before this always block, then do this assignment before this does not say anything.

So, the order of evaluation in concurrent procedural assignments in determinate, which means there is some fixed order. But that order is not known to you different compilers could do different things, different simulators could do different things and so on. So, let us look at this the order in which these always block will get evaluated has no relationship to this always block at time step 0 you go here, let us say this block gets evaluated first. Then, whatever the current value of A is you add 1 and because is the blocking assignment you assign it A immediately.

And if this always block gets evaluated next then A is already change, let us and you are adding one to that again this a blocking assignment that will go and change B, this is happen if this always block is executed before this always block. But because Verilog does not say anything about concurrent procedural assignments, it is possible that this always block gets executed before this always block, if that happen then B gets the previous value of A plus 1 and A gets the previous value of A plus 1.

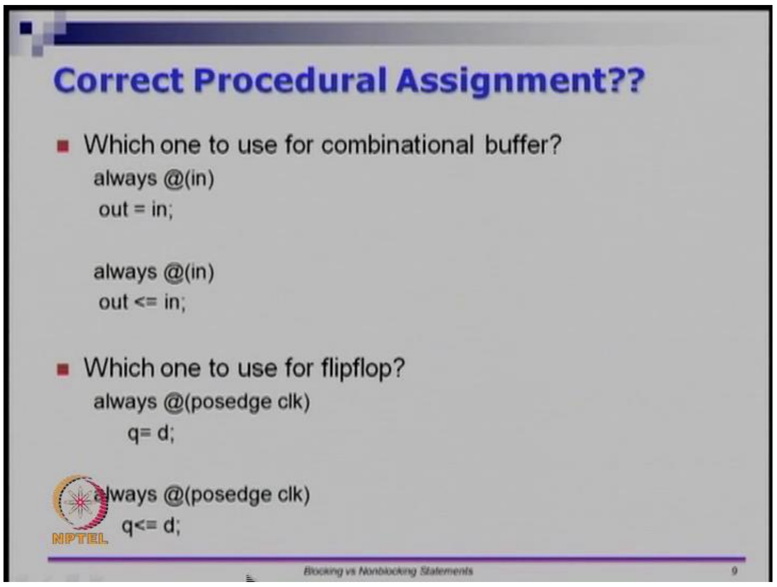
So, both B and A all are the same value whereas, if this happen first and if this happen next, then A will get the previous value of A plus 1 assign to it and B will get previous value of A before the always blocks are executed plus 2. So, it is not clear whether B should gives the previous A or the current A, so it is not clear. So, in such cases when you have multiple assignments you have to be very careful is blocking assignment is not good in the concurrent node.

However, if you do this, in this case what I have done is A equals A plus 1, B equals A plus 1. The model for non blocking statements is a across all the always blocks look at what is on the right side of non blocking statements evaluate them first. So, across all of them if I evaluate A plus 1 it takes whatever is the previous value add 1 to it, this one is also saying take the previous value add 1 to it. So, this A plus 1 and this A plus 1 are both consulting the same value of A add 1 to it and keep it aside, we have that we have the old value of A plus 1 which is kept aside.

Now, both the always blocks are coming to the end of their respective blocks at that point you can make the assignments to the left side. So, this is our Q 2 if you look at this picture ((Refer Time: 30:09)), this is our Q 2 change the left hand side of all the non blocking assignments, at this point the left hand side a here is A suppose to get 2, A suppose to get previous value of A plus 1, B will always get previous value of A plus 1. So, A and B are both getting the same value, both of them get the old value A plus 1, so this is the meaning of what this is doing.

So, this is indeterminate at the end of this we do not know whether B got pervious A or current A and added 1 to it. So, if it is previous A then we would have added only 1 to it, if it is a current A it is actually the previous A plus 2. So, we do not know whether B is previous A plus 2 or previous A plus 1, whereas in this one both A and A have previous A plus 1.

(Refer Slide Time: 31:01)



Correct Procedural Assignment??

- Which one to use for combinational buffer?

```
always @(in)
 out = in;

always @(in)
 out <= in;
```

- Which one to use for flipflop?

```
always @(posedge clk)
     q= d;

always @(posedge clk)
     q<= d;
```

Blocking vs Nonblocking Statements 9

So, let us look at some of the implications, so I am going to ask this question. So, I have the first bullet is suppose to be modeling a combinational buffer. A combinational buffer is one in which the output is suppose to follow the input immediately and the second bullet is about flip flop. So, what is the correct way to model things, so the first one is combinational, let say there are two ways to do this. So, always at in out equals in always at in out this is non blocking equal to in.

So, of course, you can always argue that I could have used assign statement out equal to in, but if we... So, this is just an illustrate of example, because this may arise in several other places, if you want to model combinational circuits how do you want to do it. So, the way to model combinational circuit is, if I put out equals in, then the because of the nature of the assign statement this in immediately gets assign to out, which means any change it in gets propagate the output.

And if out is connected to something else that will go and immediately trigger the other once also. Whereas, if I did this one if we did out less than equal to or the non block equal to followed by in, this would mean I will evaluate in, but I keep it aside I will wait for all the other evaluations of the right hand side to get over and then I will do the left hand side assignments.

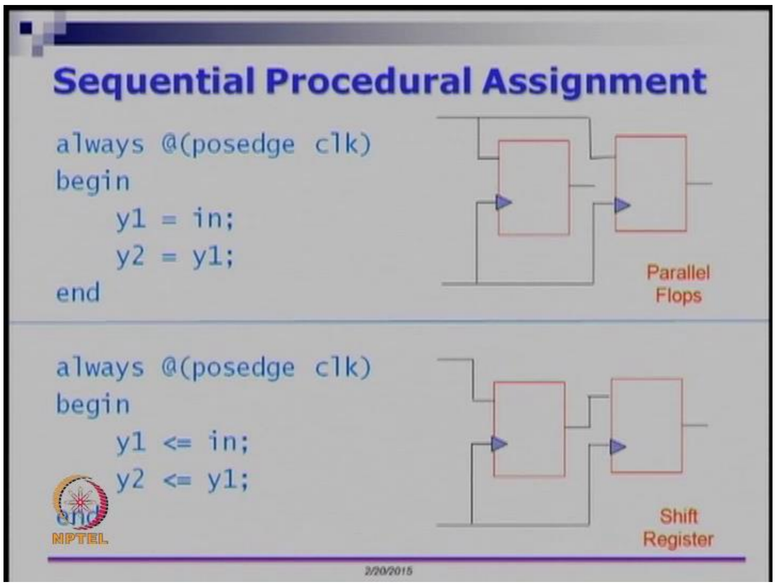
So, for modeling combinational logic this is the correct way to do it, now let us look at flip flops. So, in flip flops always at posedge of clock q equals to d as suppose to q non blocking equal to d. So, again if I use the same argument the change in d should not appear in q immediately, the change in d is suppose to change as q in the next clock cycle, if that is the case then changing q immediately is not correct. Therefore, this non blocking assignment is the correct way to do it.

So, anything that is uses q on the right side should be using the current value of q, not the new value whatever is there. So, let say always at posedge of clock q equal to d and I did something like b equals q plus 5, q plus 1 let say I had another statement which as b equals q plus 1. Then, is it is suppose to take the current d and change it immediately no. So, whatever value of d comes it gets assign to q that value of q is not suppose to trigger anything else yet.

Because, this is clock statement only when the clock assignment is all evaluated finally, you go and make the assignment. So, this q non blocking equals d essentially takes care of the fact that I am sampling d I am sampling d and; however, the assignment to q I will do it once all the always blocks are evaluated and when I am ready to assign to the left hand side of the non blocking assignments.

So, the correct way to combinational do combinational logic is, this one if you are using always statements, the correct way to do combinational logic is this one and the correct way to do sequential logic is this one.

(Refer Slide Time: 34:22)



Sequential Procedural Assignment

```
always @(posedge clk)
begin
    y1 = in;
    y2 = y1;
end
```

```
always @(posedge clk)
begin
    y1 <= in;
    y2 <= y1;
end
```

Let see some of the implications of sequential procedural assignments. So, in this picture we have two different assignments, the statement seem to be the same except that both assignments are blocking here and both assignments are non blocking here, let see what happens because of this. So, always at posedge of clock which means, first of all the this any assignment that happens inside the posedge of clock, means there is a flip flop that is going to be infer.

We are talking about a flip flop y 1 is not continuously getting in, y 2 is not continuously getting y, this is not continuous assignment, this is the sequential procedural assignment. So, if that is the case only when the clock pulse is come in y 1 and y 2 will change which means we are modeling with flip flops. So, again go back to the Verilog models to the talk about flip flops I said at posedge of clock any assignment that you do inside is suppose to be of flip flop.

So, let us look at this one now y 1 equals in, so what this means is take in whatever is there sample it. But because of this equality sign what it says is whatever is in assign it to y 1 immediately and because is this blocking until this y 1 copies in the second line will not get executed, which means the current whatever value y 1 got here is available and the right hand side here. And because of this is also blocking assignment, which means this value that you put in goes to y 1 and from there it goes to y 2 in some sense y 2 and y 1 are both getting the same values in.

So, if you think about did it looks like in is given to y 1 and y 1 is given it is value to y 2, but if you go and look at the values that y 1 and y 2 are getting both of them are getting the value in. So, the circuit that you are going to get is essentially a set of parallel flops. So, the way to thing about this is each and every assignment gets a flip flop, so y 1 is an assignment y 2 is an assignment both of them get a flip flop and what is the value that this flip flop y 1 is getting in.

So, in goes to y 1, so this is y 1 and what is the value that y 2 is getting, y 2 if you trace through this it is actually getting in itself. So, if you thing about that y 2 is also getting in, so it is as could as taking in and giving it to this flip flop y 2. So, this is y 1, this is y 2, if you notice is flip flop 1 and flip flop 2, this is producing y 1, this is producing y 2 they are actually ditto the same there exactly in the same, because that is what the implications.

So, the circuit correlates to whatever you rode as a procedure and whatever you rode as a procedure correlates to the circuit. In fact, if you give it to tools, the tools may realize that these two flip flops are taking the same input and giving the same logical value. So, if you want optimize the hardware, if you are tapping y 1 to do something and y 2 to do something, you can as well just tap y 1 and get rid of this flip flop to all together that is what tools would do.

Whereas, let us look at this one y 1 is non blocking equal to in and y 2 is non blocking assign y 1. Now, let us look at the implication of it, first of all y 1 and y 2 are assignments. So, they required two registers 2 flip flops y and y 2 or 2 flip flops, what is the semantic are meaning of this non blocking equals. So, this statement says sample the current value of in give it to y 1 at the end of the clock cycle. Similarly, it is this one says take the current value of y 1 and assign it to y 2 in the future.

So, this is sample it currently give it in the future, this is also sample it currently and give it in the future. So, equivalent circuit is as follows, it is actually a shift register, so the meaning of this one is whatever value is in y 1 give it to y 2 and whatever values in give it to y 1. But the window in which it is going to operators control by the positive edge of the clock, this is going to happen only at the positive edge of the clock, we already know that as a hardware diagram, this is the shift register.

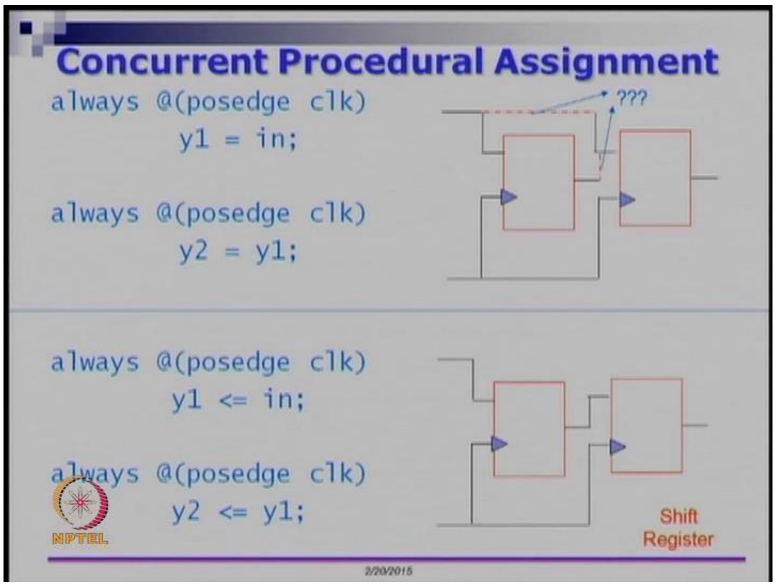
So, this value will be let us say this value is stable and this value is stable and the clock pulse comes in this value will get here and whatever stable value was here that will get here and then the closing window is that raising edge itself, it closes no for the changes are propagated. So, the same thing is suppose to be here, so if you want a model the shift register you have to do this.

Whereas, if you are modeling combinational logic which is getting registered somewhere you do this. So, in all these cases right as an exam question is the different thing, but as a hardware designer you have a certain intent and you want a express the intent as a program. So, if your intent is to want a shift register, then you are modeling should also reflect that.

The shift register means, all the flip flops are sampling the respective inputs at the same time and they copy to their outputs at the same time. So, if you have this respective inputs and to their corresponding outputs, then you are actually you should think about non blocking assignments. Whereas, if I want a sequence of changes that should result in a change in the output, then you can have a sequence of statements, this like using temporary variables.

In this sense, if you see look this picture here, you can as well treat y 1 as though it is temporary variables in gets to y 1 and y 1 gets to y 2 I can as well get rid of y 1 and say y 2 equals in, because it is blocking assignments. So, whenever you want some temporary things, if you have very large expression on the right side, you want to break it down into smaller ones, then this is good thing to do.

(Refer Slide time: 40:37)



Concurrent Procedural Assignment

```
always @(posedge clk)
        y1 = in;

always @(posedge clk)
        y2 = y1;
```

```
always @(posedge clk)
        y1 <= in;

always @(posedge clk)
        y2 <= y1;
```

So, let us look at if the same thing is done concurrently here what we have is y 1 is in and y 2 is y 1 we have into two different statements and this is done concurrently which means it is not in the same always block, there are two different always blocks. So, the key thing is if I do this always at posedge of clock y 1 equals to in; that means, in for a flip flop for y 1. So, that is this flip flop, this y 1 is suppose to follow in always at posedge of clock y 2 equals y 1.

So; that means, this flip flop y 2 is suppose to be coping y 1, but because of this semantic of this always blocks and the blocking assignment. So, if this block got executed before this block, then whatever value was in would have propagated to y 1 and that value of y 1 will get propagated to y 2 which means the current input will go to y 2 also. So; that means, it is equivalent to this parallel flop case, so the input is going to both y 1 and y 2 or if this flip gets executed first, then y 2 makes a copy of y 1 and then y 1 changes to in that would be equivalent to doing the shift register.

So, y to takes the value of y 1 and y 1 takes the value of in, so this is crazy, if you put blocking assignments. Because, the order in which the always blocks are going to be executed is not guaranteed then there is the race condition on the input of this flip flop y 2, it could either be in or it could be y 1 and you do not know which, a single complier will always give you the same result, but if you take it to a different complier it may actually give a different result.

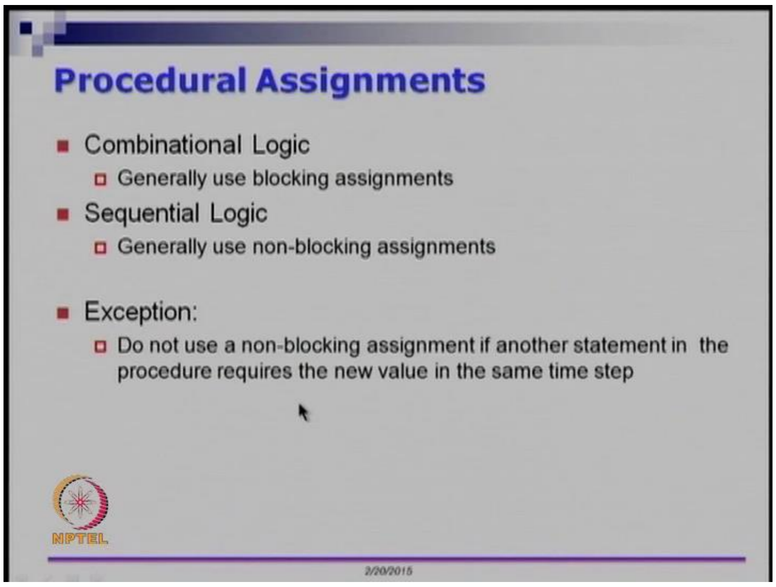
So, this is the serious problem whereas, if you had this statement these are concurrent procedural assignments again, because they are two always statements we have both of them with non blocking assigns. So, no matter the order in which done will have to wait for all the left hand side to be evaluated first and once all the evaluations are over only then you will do the assignments. So, whether it is this always block or this always block, you do the evaluations first.

So, which means take the current value of in keep it aside, take the current value of y 1 keep it aside. Whether, this execute first or this execute first both of them have to execute and once they are done, once they come to end only then you go to the left side, the left side it says whatever value of in you remember put it in y 1 whatever value of in y 1 you remember put it in y 2, so that is equivalent to the shift register.

So, the shift register could either be model like this or model like this whereas, if you do this, this is actually still a correct circuit, there is no problem only that both y 1 and y 2 will get the same value. However, this circuit is incorrect in two ways, one there is a race condition and two the intent of the programmer is not going to captured.

So, if the intent of the programmer is made 2 flip flops y 1 and y 2 which are both suppose to follow input that is not captured, the intent of the designer is I actually need a shift register even that is not captured. So, this may actually depend on the tools that you are using for simulation, so this is the terrible think. So, this is the reason why you do not use this blocking assignments statements for sequential logic, whenever you want a flip flop you always use non blocking assignments.

(Refer Slide Time: 44:34)



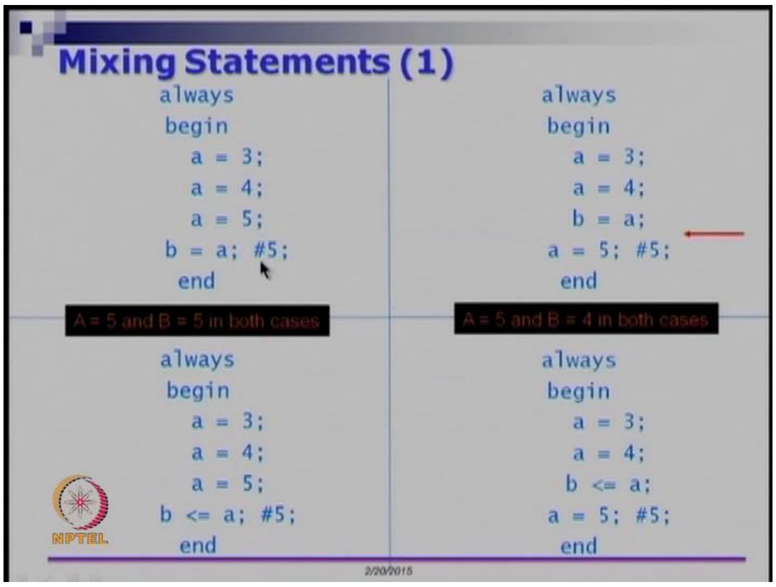
Procedural Assignments

- Combinational Logic
  - Generally use blocking assignments
- Sequential Logic
  - Generally use non-blocking assignments
- Exception:
  - Do not use a non-blocking assignment if another statement in the procedure requires the new value in the same time step

2/20/2015

So, combinational logic generally use blocking assignments, sequential logic generally use non blocking assignments, the only exception for this case is do not use a non blocking assignment, if another assignment in the procedure requires a new value in the same step. What I mean by that is, if you are using something for a temporary calculation use blocking assignments and not non blocking assignments, so let see examples.

(Refer Slide Time: 45:02)



Mixing Statements (1)

```
always
begin
  a = 3;
  a = 4;
  a = 5;
b = a; #5;
 end
```

```
always
begin
  a = 3;
  a = 4;
  b = a;
a = 5; #5;
 end
```

A = 5 and B = 5 in both cases

A = 5 and B = 4 in both cases

```
always
begin
  a = 3;
  a = 4;
  a = 5;
b <= a; #5;
 end
```

```
always
begin
  a = 3;
  a = 4;
 b <= a;
a = 5; #5;
 end
```

2/20/2015

So, I am going to mix different kinds of statements from now on and the rest of the video is about this. So, let say this I have assign a equals 3 it is all in always statement, blocking assign a 3, blocking assign a 4, blocking assign a 5, blocking assign b equals a and I say now wait for 5 time units that is what the meaning of hash 5 is, let see what the meaning of this part is first.

So, it is says a should immediately get the value of 3, the next line says a should get immediately value 4, next line says a should get the value immediately 5 and b equal to a says whatever is the values of a immediately I should give it to b. So, at this point this a equal to 3 and a equal to 4 actually do not do anything, you destroying the value of a making it 3, you destroying that value at making it 4, you destroying that value and making it 5. So, b will be 5, a will also be 5 at the end of time 5 units that is for this case.

Let see what happens here a equals 3, a equals 4, b equals a and a equals 5. So, if I go and do this in sequential programming language, you would except that b gets the current value of a. What is the current value of a? According to this statement the most reason one is the assignment of a is 4. So, b will get 4 and a there is another assignment later, this is assigning 5 to a. So, at the end of this assign statement this always block b will get 4 and a will get 5.

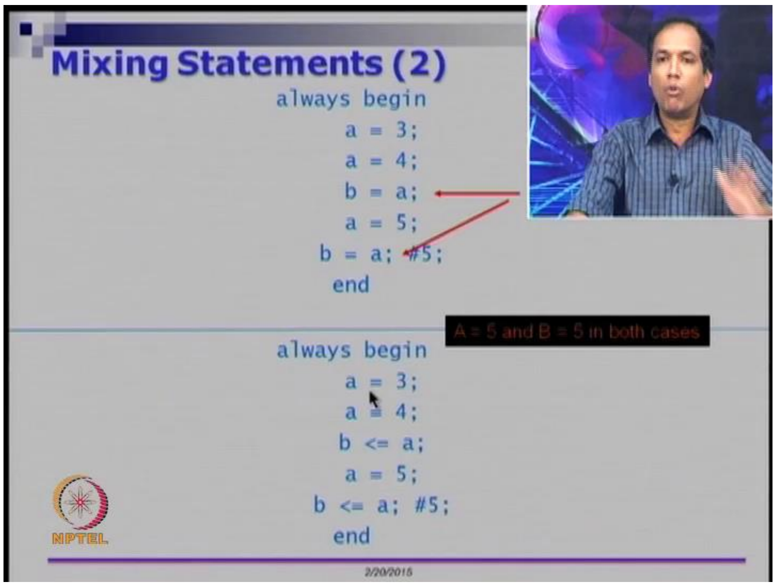
So, the changing the order of these assignments has actually change the values that a and b will get, now let see this part. So, what we have done here is, this last statement is replaced by non blocking assignment and this b equal to a is replaced by a non blocking assignment. So, let see the implication of that in this case 3 gets assign to a and that comes here at this and the line the control goes to next line, there you do not care about 3 you assign 4, when you come here you do not care about 4 also you now assign 5 and that current value of 5 is assign to b.

So, b gets the value 5 and a gets the value 5 also, but a has it immediately and the value 5 for b the schedule for the future, which means this 5 is remember and if there are other non blocking assignments all of them should get evaluated and only then you will assign it to b, let see what happens on the right side here. So, this 3 and 4, this 4 destroys 3, so that is straight forward 4 destroys 3, then this value of 4 is suppose to be copy to b, but not immediately sometime in the future.

So, b is going to get the value 4 sometime in the future, but a is assign 5, so the key question is this b suppose to get 5 or this b suppose to get 4. So, at the time of execution of b we said remember the value 4, this is going to be assign b in the future. So, even if a changes we said remember the value 4 for b, so at the end of this always block we will get 4 and a will get 5. So, for the left side a equals 5 and b equals 5 at for both the cases, for the right side a equals 5 and b equals to 4 for both the cases.

So, this is just to show give you some more understanding of blocking and non blocking, in this case we are mixing blocking and non blocking assignments and here there is no implication of a flip flop or anything, because there is no always at posedge of clock, it suppose to be a combinational block with some delay is 5 units of delay.

(Refer Slide Time: 49:09)



Mixing Statements (2)

```
always begin
     a = 3;
     a = 4;
     b = a;
     a = 5;
   b = a; #5;
    end
```

A = 5 and B = 5 in both cases

```
always begin
     a = 3;
     a = 4;
     b <= a;
     a = 5;
   b <= a; #5;
    end
```

Let us look at these two always blocks, so a equals 3, a equals 4, b equals a, so it is clear that b is suppose to get a value of 4, then a equals 5. So, at this point a becomes 5 and b equals a at this point b also becomes 5. So, this b equal to a is an immediately assign here, so this is a blocking, so b gets 5 here immediately a also gets 5 here immediately at this point when you come here a equals to 5. In fact, this whole thing is as good as these three statements ((Refer Time: 09:51)) exist.

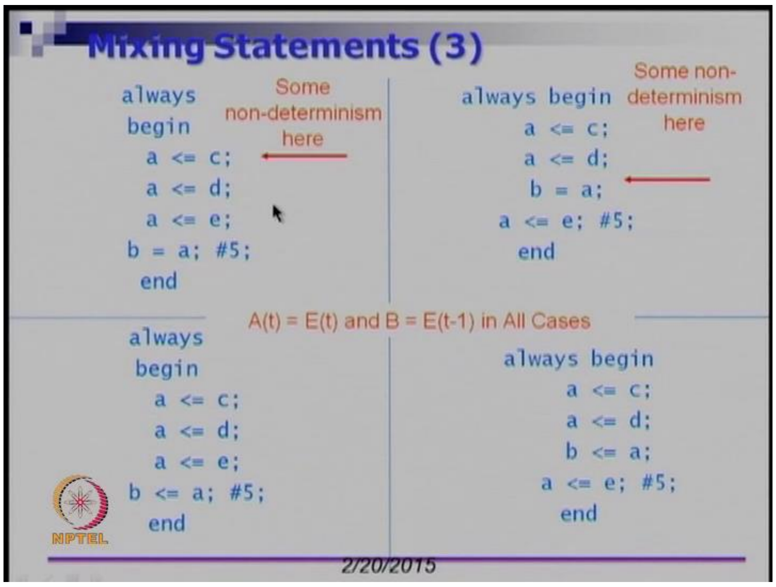
Because, this is the last assignment to a and we are taking the last assignment of a and assigning it to b immediately which means all these three statements even if they were not there the meaning as the same, let us look at the piece of logic below this. So, a equals 3, a equals 4, b equals a. So, at this point b is suppose to remember the value 4 we are still within the same procedure a gets 5.

So, this 5 is not suppose to reflect in b and here this a equal to 5 is again blocked here, so this is the blocking assignment a equals to 5. So, this b equal to a is actually looking at the right side which got immediately evaluated to 5. So, let us look at the sequence of events here, so all the blocking assignments are done one after the other. So, this blocking assignment is 3, this blocking assignment is 4, you hit a non blocking assignment at this point what we do is, will schedule b to get a value of 4 in the future and we continue.

That is the sequence, a get's 3 immediately, a gets 4 immediately, b gets a value of 4 to be scheduled in the future. I continue running this a get's a value 5 and this one says schedule the value of 5 for b in the future. So, in some sense this one says assign 4 in the future and this one says assign says 5 in the future, which means of the two events whether 4 is in the future or 5 is in the future, the program order it says 5 in the future is what I want.

Because, 4 in the future is coming before 5 in the future at the end of this statement we will also get the value 5. So, this one and this one will actually get the same values, so this is not good for the modeling hardware, but again I am showing this for that you understand the notion of blocking and non blocking and the problems with mixing. So, a and b both of them get 5 in both the cases.

(Refer Slide Time: 52:08)



Mixing Statements (3)

```
always
 begin
   a <= c;
   a <= d;
   a <= e;
 b = a; #5;
  end
```

Some non-determinism here

```
always begin
    a <= c;
    a <= d;
     b = a;
  a <= e; #5;
   end
```

Some non-determinism here

A(t) = E(t) and B = E(t-1) in All Cases

```
always
 begin
   a <= c;
   a <= d;
   a <= e;
 b <= a; #5;
   end
```

```
always begin
    a <= c;
    a <= d;
    b <= a;
  a <= e; #5;
   end
```

2/20/2015

Let us look at these statements now, a equals c, a equals d, a equals e. So, I have three non blocking statements, first one is trying to assign c to a , the second one is assign d to a and third one is assigning e to a. So, in time step 0 we are saying that schedule c for a in the future, again a time step 0 we are still evaluating the right side, we are saying assign d 2 a in the future, again a time step 0 we are saying evaluate e and assign to a in the future.

So, what this means is, there are three different assignments that we are trying to do it a in the future, the last assignment a revise which means a will get the value of e and this one it says b equals a, what this does is b is trying to get the... So, you go and look at the evaluation of a, when you evaluated a all these events for a are schedule in the future. So, whatever is the previous value of a take that immediately copy that to b that is the meaning of this statement.

So, the meaning of this whole thing is take c d and e, this assignments c and d do not matter, the assignment e should be give to a in the future. However, whatever is the current value of a assign it immediately to b that is the meaning of this block, let see what happens here, there are c and d here and you are sampling a, what is the a that you will be sampling here, a that you are sampling is whatever you entered the block width. So, is b equal to a does not a really change anything is b equal to a still sampling the old value of a, because the other assignments to a are all non blocking.

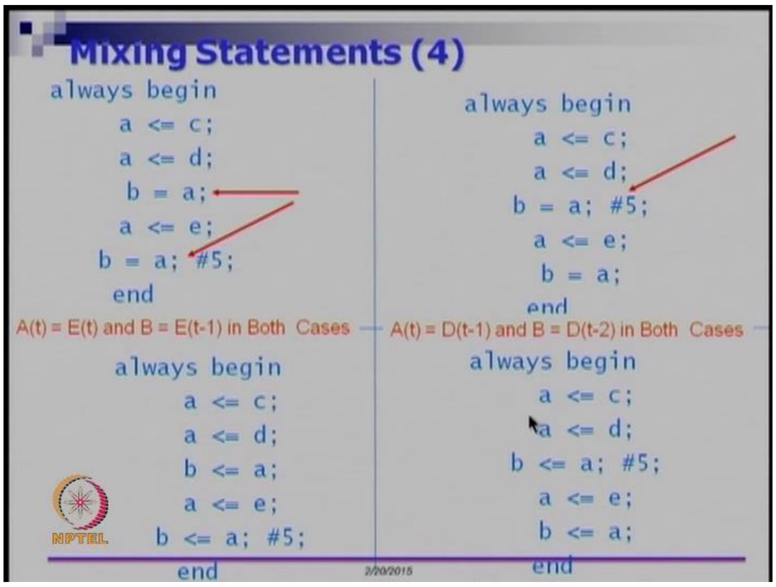
So, if is all non blocking you can as well go and execute this first, when execute this first a is taking the value whatever the previous value is you schedule that for b immediately and this a equal to e is again scheduling something in the future. So, this a equal to e over rights this a equal to d and a equal to c. So, a will get the value of e and b will get the previous value of a, so that is true for both these things.

Let us look at what happens here, so a equals c, a equals d and a equals e and this means whatever is the current value of e assign into a in the future and whatever is the current value of a assign into b in the future that is the meaning of this. And the meaning of this one is c and d is the latest assignment. So, since all of these are non blocking assignments, you take the last assignment for each of these once.

So, last assignment of a is take e, the last assignment of b is take a, so whatever the current value of a is give it to b, whatever the current value of e is give it to a. So, all of them are actually doing the same thing, so a of t will become e of t and b will become e of t minus 1 in all cases, because b is coping a and a is coping e. So, b will be lagging behind a in one step, so a of t will be e of t whatever the current value of e is, you will get it immediately to a.

But b is copying the current value of a, which means is the previous value of b, so a of t is e of t and b of t is e of t minus 1 in all these four cases. So, clearly you can see that mixing the statements is equal to and non blocking equal to, it is already getting very confusion, you have to remember so many details above what each of these things are and so on. So, there is some non determinism that happens with respect to how these things are going to execute, but that does not matter all of them give the same results.

(Refer Slide Time: 55:16)



Mixing Statements (4)

```
always begin
    a <= c;
    a <= d;
    b = a;
    a <= e;
  b = a; #5;
  end
```

A(t) = E(t) and B = E(t-1) in Both Cases

```
always begin
    a <= c;
    a <= d;
  b = a; #5;
    a <= e;
    b = a;
  end
```

A(t) = D(t-1) and B = D(t-2) in Both Cases

```
always begin
    a <= c;
    a <= d;
    b <= a;
    a <= e;
  b <= a; #5;
  end
```

```
always begin
    a <= c;
    a <= d;
  b <= a; #5;
    a <= e;
    b <= a;
  end
```

2/20/2015

Let us do one final round of mixing statements, so you have a equals c and a equals d, then you have b equals a, a equals e and then we have one more assignment to b here. So, there are two assignments to b in this previous slide we had only one assignment to b and three assignments to a, in this slide we have three assignments to a and two assignments to b let see how to reason this.

So, a equals c a equals d which means a equal to d over write the a equal to c and this b equal to a this is actually going to look at the old value of a. So, this is not going look at the new value, so this is the old value of a, this e is also scheduling something for a in the future. So, this a equal to e is not really doing much, so this c d and e a equal to e overwrites d and c. So, this a equal to e is also schedule for the future.

So, which means I can this is non blocking I can come to a, so it is again saying b equal to a. So, in some sense b equal to a and b equal to a are the two statements that we have and which means b is going to copy a and a in the future is going to take the current value of e that is the meaning of this always blank. So, I recommend that you go and reason what should be the assignments for each one of them, I will give you what it final assignment must be...

So, a of b must be equal to e of t and b of t must be equal to e of t minus 1 for this side and for the right side a of t must be equal to d of t minus 1 and b must be equal to d of t minus 2 in these cases. So, now it have some understanding of blocking versus non blocking, I showed you what happens to the left side, I want you do go and think about what happens to this one, this one and this one.

So, clearly all this mixing and matching as a programming technique I can do all these dirty things, but as a designer this is an night man. So, if an engineer whose working under me wrote anything like this, I would immediate to fire engineer, because this is to learn what non blocking and blocking assignments are, but this is not capturing the intent of the designer neatly and anyone else who is going to look at is going to get confused.

So, the thumb rules are that use blocking assignments only for combinational logic, use non blocking assignments only for sequential logic and as much as possible do not mix non blocking and blocking assignments. So, in general what you do is, you keep the combinational block aside, you keep the sequential block aside and you combine them you have them as two different always blocks.

This exactly what we did in our state machine, if you go back and look at the state machines Verilog code, you would see that the sequential logic was one single assignment and it will used non blocking assignment and the combinational logic had where two different blobs and the combinational blobs are separated from a sequential blobs, they were different and these were different.

So, this brings me to end of module 46, in the next module what we will do is, we will learn the notion of pipelining. So, in pipelining, so we will all the concepts that we learnt about the blocking and non blocking assignments, we will take that and see what is the implication of pipelines. So, this brings me to end of the module 46 and I will see a module 47.

Thank you and see you soon.