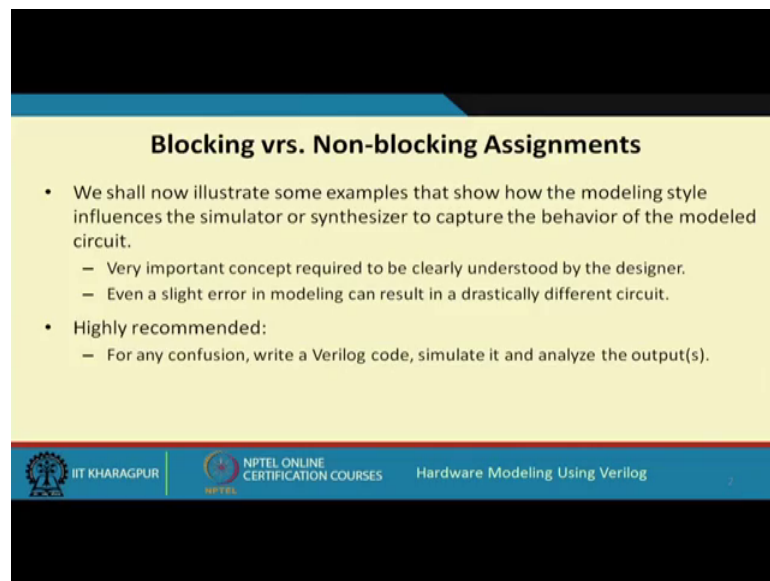


**Hardware Modeling using Verilog**  
**Prof. Indranil Sengupta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 18**  
**Blocking / Non-Blocking Assignments (Part 3)**



So, we continue with our discussion on the Blocking and Non-Blocking Assignments. So, in this lecture, we shall looking at some more differences in the way they behave and the way they synthesis or the simulation tool will interpret or handle the description that you have given or provided in either blocking or non-blocking kind of assignments. So, we shall be illustrating these points through a number of examples that will be the best way to just explain.

(Refer Slide Time: 00:56)



**Blocking vrs. Non-blocking Assignments**

- We shall now illustrate some examples that show how the modeling style influences the simulator or synthesizer to capture the behavior of the modeled circuit.
  - Very important concept required to be clearly understood by the designer.
  - Even a slight error in modeling can result in a drastically different circuit.
- Highly recommended:
  - For any confusion, write a Verilog code, simulate it and analyze the output(s).

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

So, actually through the examples, we shall be showing you some modeling styles; some modeling styles using both blocking assignment and also using non-blocking assignments.

And there we will try to understand by looking at this semantics. So, the meaning of the statements, how this simulator or the synthesis tool might be influenced to capture the behavior of a model in a particular way; that whether it is actually capturing the behavior in the way you are intending or it is doing something else. So, as these are something which is extremely important from the point of view of the designer because you have to

understand this blocking and non-blocking statements very clearly because if you do not model the behavior of a circuit or system that you want in the proper way.

Then even a very small error in the model which might easily be overlooked that might result in a drastically different output or a drastically different circuit depending on whether you are simulating or synthesizing. So, we shall be trying to see some examples. Now there is another thing, I just mean, I mentioned repeatedly that when you are studying some codes when you are learning some constructs of a language. So, it is always recommended that you actually run the code say for Verilog, you simulate it and see how the outputs are coming change the code and see; what are the changes that are happening.

(Refer Slide Time: 02:50)

**Example 1**

```
begin
  a = #5 b;
  c = #5 a;
end
```

- The value of "b" will be assigned to "c" 10 time units after the "begin ... end" block starts.

```
begin
  a <= #5 b;
  c <= #5 a;
end
```

- "a" is scheduled to get the value of "b" 5 time units into the future.
- "c" is also scheduled to get the value of "a" 5 time units into the future.

*Value of "c" will be different for the two cases*

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

So, the first example that I take is a simple timed assignment using blocking and non-blocking a very simple segment of code. So, inside a block begin end block this can be inside either always or initial whatever some procedural block I write blocking assignment after delay of 5 b c equal to after delay of 5 a and here similar, but non-blocking. So, let us try to understand what will happen here blocking statements will be executing one after the other, first statement will say that after a time delay of 5 b will be assigned to a second statement says after this is finished, only then, it will come to the second statement. So, after another time delay of 5 that value of a will be copied to c. So, the value of b which was there it first goes to a and then it goes to c.

So, the value b will be finally, assigned to c after a delay of 10 time units, right, this will happen here. Now if you use non-blocking assignment statement, what will this mean? This will mean here the time delay does not mean that you first finish this after time 5, then again after time 5 do this, no, it will mean that you evaluate the right hand sides see a scheduled to get the value of b 5 time units into the future right hand sides are evaluated together when you say hash 5 and hash 5. It does not mean like this at first this 5 and then this 5, this is a 10 not this, both of these are waiting for time 5 and they are assigning they are executed concurrently right. So, here b will be assigned after time 5 to a; a will be also assigned after time 5 to c.

So, you see in the first case here the value of b was assigned to c, but here finally, the value of a will get assigned to c, right. So, the final value of c will be different for these 2 cases, this is a very simple example, but you will have to understand the difference very clearly what this hash 5 here and what this hash 5 here means. So, when you are using a blocking assignment, I am repeating if you are using those delays because in a blocking assignment statement, this statements will be executed one after the other their delays will get accumulated if you give 5, 5, 5, it will 5, 10, 15, 20 that way the delays will get accumulated, but in a non-blocking case, it is not.

So, there the delay means you evaluate all the right hand sides together and after how much time you will be assigning it to the left hand side, if you give all then hash 5 hash 5 hash 5 means after delay of 5 all of them will be assigned together it is not that the delay will be adding up not like that. So, because of that for this 2 cases the final value of c was coming to be different let us take another example.

(Refer Slide Time: 06:35)

**Example 2**

```
always @(posedge clock)
begin
  q1 = a;
  q2 = q1;
end
```

*Parallel Flip-flops*

The diagram illustrates the hardware implementation of the Verilog code. It shows two D flip-flops connected in series. The first flip-flop has its D input connected to signal 'a' and its Q output connected to signal 'q1'. The second flip-flop has its D input connected to signal 'q1' and its Q output connected to signal 'q2'. Both flip-flops are clocked by a common 'clock' signal. The text 'Parallel Flip-flops' is written above the diagram.

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

This is procedural block with a clock always posedge clock, here I am using a blocking assignment. So, what does this block mean this block means that whenever there is a positive edge of the clock you activate or start executing this what is this saying that you first assign a to q 1, then assign q 1 to q 2, right.

So, in terms of the hardware, if you give it to the synthesis tool; what synthesis tool it will generate? You see a is assigned to q 1, just think a flip flop a will be assigned to q 1 you. So, whenever the positive edge of the clock comes, a will be assigned to q and it will be stored in the d flip flop the first statement goes and after this is done this q 1 will be assigned to q 2, because you see this is not that q 1 will be connected to this, because I mean you will have to understand; what is the meaning of non-blocking the synthesis tool and also the simulation tool they know what is the meaning of the blocking assignments blocking assignments means these statements are supposed to be executed one after the other after the first statement executes that value will be used to execute the second statement one by one sequentially.

So, if you keep that sequential thing in mind then you see the first statement says q 1 equal to a, this is fine, it will be mapped to this flip flop q 1 will be a, but after this is completed, only then, the second one will start. So, this q 1 already has received the value of a. So, that a is supposed to go to q 2. So, the synthesis tool will understand that meaning of this blocking assignment. So, instead of connecting q and it will connect a

directly to this second flip flop also to generate q. So, it will be 2 parallel flip flops after the clock finally, both q 1 and q 2 will be getting the same value a; a will be stored here as well as a will be stored here.

As this meaning shows, this a will go to q 1; q 1 will go to q 2, same a will come, right. So, in terms of the hardware it with 2 parallel flip flops, let us look at a slightly different you see this was my description I simply interchange the order.

(Refer Slide Time: 09:30)

**Example 3**

```
always @(posedge clock)
begin
  q2 = q1;
  q1 = a;
end

always @(posedge clock)
begin
  q1 <= a;
  q2 <= q1;
end
```

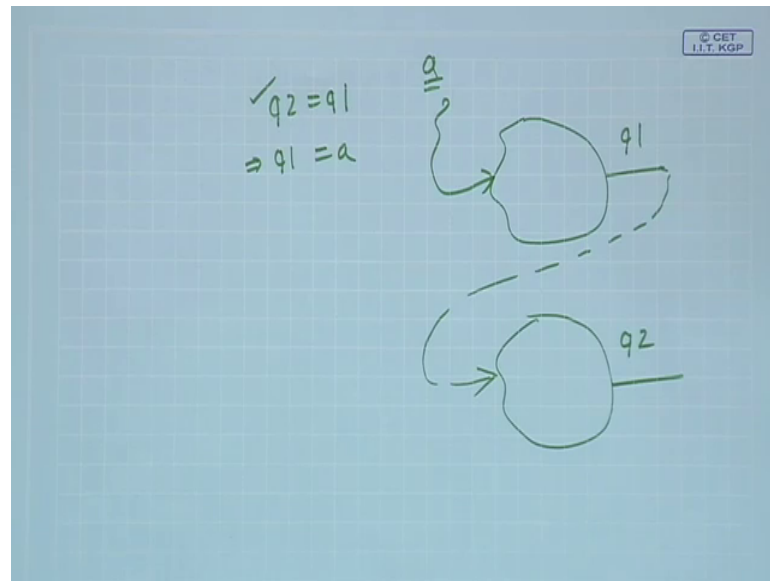
*Shift Register*

The diagram shows two D flip-flops connected in series. The first flip-flop's D input is connected to signal 'a', and its Q output is connected to signal 'q1'. The second flip-flop's D input is connected to signal 'q1', and its Q output is connected to signal 'q2'. Both flip-flops have a clock input connected to a common 'clock' signal.

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

I just write it like this and below I write an description same description using non-blocking. Non-blocking assignment here let us try to understand; what will happen; here we have written q 2 equal to q 1 and q 1 equal to a; what does this mean?

(Refer Slide Time: 09:57)



You see you can imagine, there were 2 flip flops,  $q_1$  will be the output of 1 of the flip flops,  $q_2$  will be the output of the other flip flop, first one you are saying that  $q_1$  will be going to  $q_2$  which means this  $q_1$  has to be connected to the input of this  $q_2$  will have to go to  $q_2$ .

Secondary we saying some other vary because after this is completed only then you will come here  $a$  will go to  $q_1$ . So, some other value  $a$ ;  $a$  will go to  $q_1$ . So, the old value of  $q_1$  will go to  $q_2$ . This  $a$  will be going to  $q_1$ , you see what we are trying to say is nothing, but a shift register if you write it in the other way around by interchanging the order; what you are saying is just a 2 bit shift register  $q_1$  will be shifting to  $q_2$  and then this  $a$  will be shifting into  $q_1$ , right. So, the same thing will happen for the non-blocking case or so, non-blocking means the same thing  $q_1$  will go to  $q_2$   $a$  will go to  $q_1$  the order is not important because in whatever we just order you mentioned.

So, the hardware that will be synthesized well be a 2 bit shift register  $q_1$  will go to  $q_2$   $a$  will go to  $q_1$ . So, here also  $a$  goes to  $q_1$   $q_1$  goes to  $q_2$   $q_1$  goes to  $q_2$ . So, the previous values of  $a$  and  $q_1$  previous value of  $a$  previous value of  $q_1$ , they will be assigned to  $q_1$  and  $q_2$ . This will be assigned to  $q_1$  previous value of  $q_1$  will be assigned to  $q_2$ . So, both of these descriptions model a shift register; so, one thing you understand for a blocking case it is very peculiar. So, if you write the 2 statements in a

particular order then it will generate 2 flip flops in parallel, but if you simply interchange them it will become a shift register.

So, this you will be able to understand only if you know; what is the meaning of the blocking assignments, if there are some statements what exactly is the meaning and how they will execute and what will be the final result, right.

(Refer Slide Time: 12:28)

**Example 4**

```
always @(posedge clock)
  q2 <= q1;
always @(posedge clock)
  q1 <= a;
```

*Shift Register*

The diagram shows two D flip-flops connected in series. The first flip-flop's D input is connected to signal 'a', and its Q output is 'q1'. The second flip-flop's D input is connected to 'q1', and its Q output is 'q2'. Both flip-flops share a common clock input labeled 'clock'. The text 'Shift Register' is written above the circuit.

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

So, let us continue, let us take another example here there are 2 blocks to procedural blocks both on positive edge here you are saying at the positive edge q 1 will go to q 2 also the positive edge a will go to q 1, just if you move back in the previous slide you see here we used a single procedural block, but you are doing the same thing at the positive edge a was going to q 1 q one is going to q 2.

But here we are meaning the same thing maybe we have using 2 different blocks, but both are activating in positive edge and the assignment will take place in a very similar way. So, here also you will be generating the same shift register, right. So, this is fairly simple. So, if you just put them in this order then also you can generate a shift register, but if you put them the same code instead of non-blocking; if you put blocking; what will happen?

(Refer Slide Time: 13:27)

**Example 5**

```
always @(posedge clock)
  q1 = a;
always @(posedge clock)
  q2 = q1;
```

*q2 will be indeterminate*

Hardware Modeling Using Verilog

Let us understand; here there are 2 different always blocks. So, in the earlier case, we said that the 2 statements are inside a single always block which means they were executing one after the other.

But here I am saying; there are 2 different always blocks, they are executing concurrently maybe there are blocking assignments, but  $q1 = a$  and  $q2 = q1$ ; both will be executing together whenever posedge of the clock comes. So, now, you see; what will happen; well,  $q1$  will go to  $q2$  is fine, but  $a$  will be going to  $q1$  now in terms of the hardware just see. So, what we are trying to say is this first it will say  $a$  will go to  $q1$ ; that means,  $a$  will go to  $q1$  second statement says  $q1$  will go to  $q2$ . So, some  $q1$  will go to  $q2$  both at the positive edge, but the value of  $q2$  will be indeterminate.

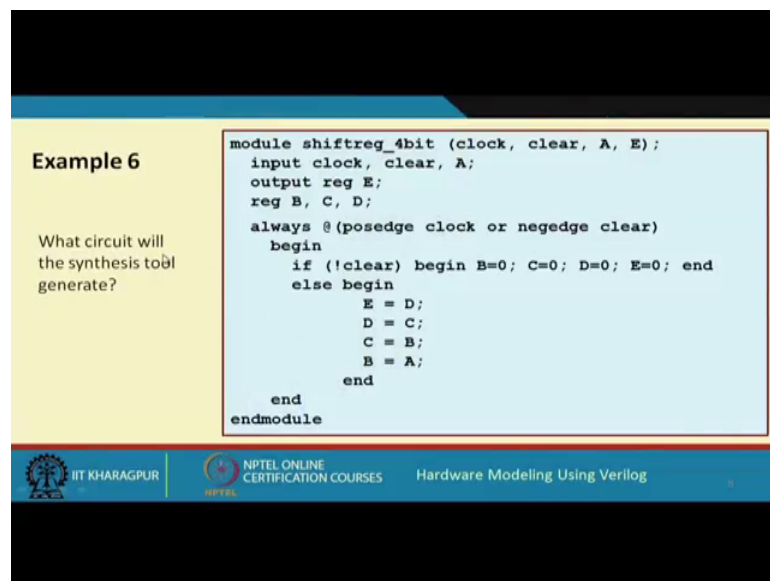
Because what will be this  $q1$ , here will it be the old value of  $q1$  or this new value of  $q1$  which you have just now loaded into  $q1$  from  $a$ , this description is a little ambiguous if you use blocking statements here, this description is ambiguous and you should avoid this because simulator may also confuse; this  $q2$  will be  $q2$  will become indeterminate because it is simulator first execute this. So, whatever is the old value of  $q1$  it will be assigned to  $q2$ , then  $a$  will be assigned to  $q1$ , but if its executed this first, then the new value of  $a$  will be assigned to  $q1$  and that  $q1$  will be assigned to  $q2$ .

So, it will act as shift register. So, the interpretation of the meaning may not be unique depending on the order of the execution that will be scheduled by this simulator the



synthesis tool will also give you a warning that there is a confusion and the output value will be indeterminate. So, you should. So, the crux is that you should try to avoid clock triggered assignment on blocking statement as far as possible whenever you want to carry out assignment with clock triggering better to use non-blocking then all these problems will not occur, fine, let us look at another example which is slightly more complex.

(Refer Slide Time: 16:24)



**Example 6**

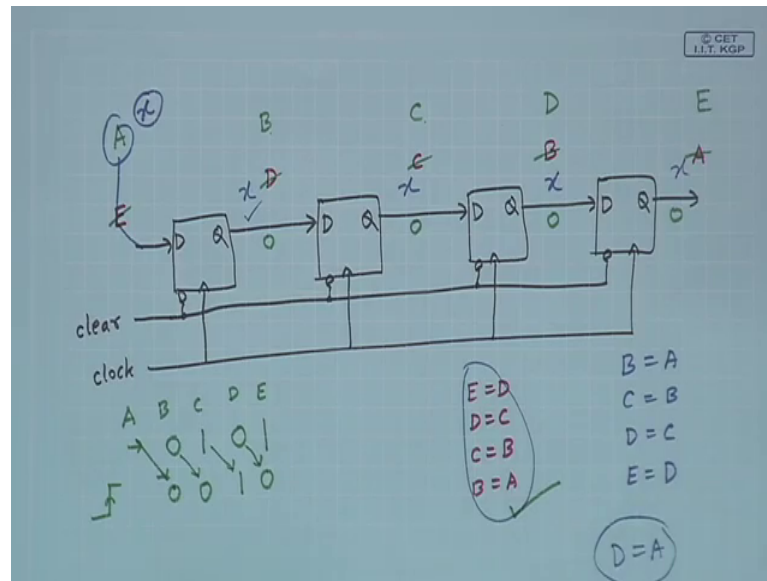
What circuit will the synthesis tool generate?

```
module shiftreg_4bit (clock, clear, A, E);
input clock, clear, A;
output reg E;
reg B, C, D;
always @(posedge clock or negedge clear)
begin
if (!clear) begin B=0; C=0; D=0; E=0; end
else begin
E = D;
D = C;
C = B;
B = A;
end
end
endmodule
```

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

And it will be bigger; well here our objective was to design a 4 bit shift register, suppose this was what we started to design.

(Refer Slide Time: 16:41)



So, we wanted to design a 4 bit shift register like this; there 4 flip flops; the output of one flip flop will go to the input of the second flip flop. So, this is what we are trying to model let us say and of course, there will be a clock; there will be a clock signal which will be feeding in parallel to all the flip flops and of course, there will also be a clear input clear will be active 0. So, whenever it is 0 it is activated. So, I am showing it as a bubble; suppose, this is what I was trying to model and let us say we came up with a Verilog code like this.

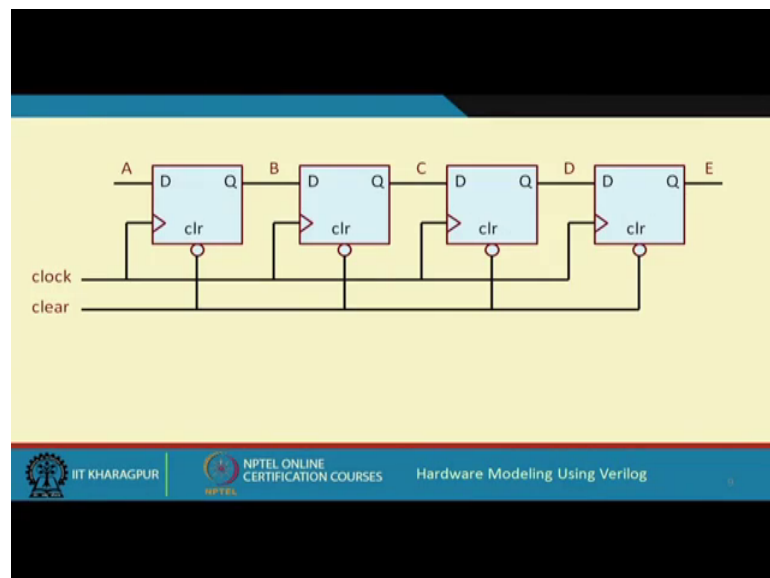
Let us see whether this Verilog code means actually models are shift register like this or not, right, let us see here as I said our parameters are clock clear A and E. So, what is our A and what is our E? This E is our input and A is our output you see A, the input and E is the output fine E is the input and A is the output. So, actually what we want here is that let us say we want to shift register when we wanted to call these as E, we wanted to call this A and the intermediate point; let us call this as B, C and D. So, you can either do it like this or you can do it in the previous way that will be easier I think let us keep it in the previous way that will be easier. Let us call this E, let us call this A, then this will be B, this will be C, this will be D.

So, it really does not matter. So, let us just referred back in the previous one; let us see. So, here the intermediate points B, C, D; we are declaring as reg. Now in the block; what we are doing? We are saying always that positive edge clock or negative edge clear. So,

whenever either there is a clock going high or a clear is becoming 0, you enter this block if clear is 0, if not clear in this begin end block you clear B, C, D, E to 0s. So, there are 2 here there are flip flops B, C, D, E you initialize them to all 0s 0 0 0 0. So, whenever clear is 1, clear is 0, not clear is 1.

Else clock has come you are trying to do something you are writing 4 things you are writing E equal to D, you are writing D equal to C, you are writing C equal to B and you are writing B equal to A. So, if you write like the; what is meaning D equal to E means whatever is D; you are copying to E, D equal to C. So, whatever is C, you are copying to D, C equal to B; whatever in B, you are copying to C whatever B; B you are A; you are copying to B, right.

(Refer Slide Time: 20:48)



So, this is like the shift register thing. So, this is the shift register which will be trying to model like this, right and this is the description equal to D means.

So, in this order, the statements will be executed, these are all these are all blocking assignments; first D will be assigned to E. So, in a shift register; what happens suppose the shift register my bits were 0, 1, 0, 1. Now if I apply a clock; what will happen? A clock comes; this will be shifted, right. So, this 0 will come here; this one will come here, this is 0 will come here, this one will come out and a new 0 will come in. So, you see D equal to E, this is E, this is D, this is C, this is B; D equal to in whatever was D; this goes to E, this is the first step, then C goes to D; whatever is C, it goes to D, then B

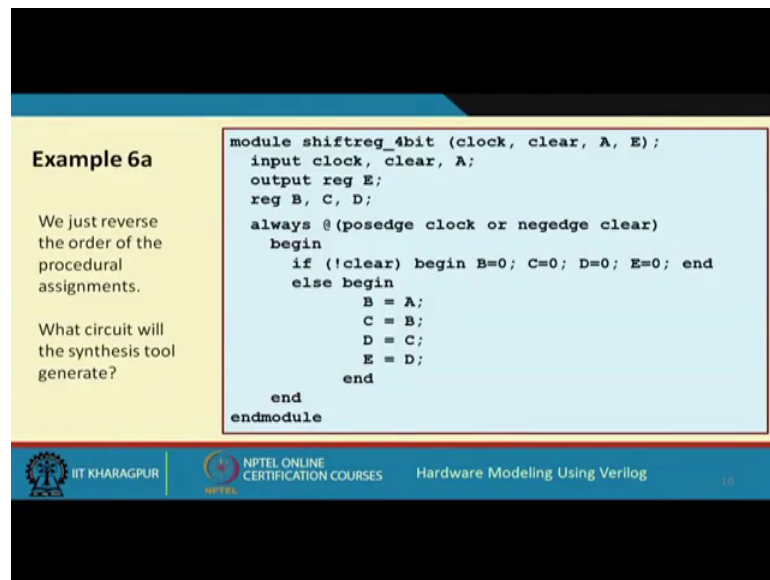
goes to C; whatever is B; it goes to C; then A is the input; A is the input from outside A goes to B.

So, A goes to B. So, it is actually shift register, first, we are shifting this, then you are shifting this, then you are shifting this, then you are shifting this; this is correct behavior of a shift register. So, when you are actually trying to model a shift register, this order is correct. So, I think here let us correct this. It is actually A should be input and this E should be output. This is correct description. This A is input and E is output. This is correct, fine. So, this is a correct way of modeling a shift register like this. Now suppose we make a small change; just this 4-4 assignments which are there, we simply reverse their order, we just write B equal to A first, C equal to B next, D equal to C next and E equal to D last.

So, we just reverse the order of the procedural assignments. The remaining thing is unchanged; just these 4 statements; we reversed; C means ultimately you may think that well, we are actuated to a shift register. So, whether you specify that first and then this or this first and then this should be the same, it should not matter, but suppose you do it like this, then what will happen? Let us see, well, this A is coming from outside, right, A is the external input which is coming here. Now you first say; you have saying A equal B equal to A. So, whatever is A that is first coming to B. So, A suppose the value of A was x let us say.

So, x will come here, then this statement will execute B is going to C. So, B has already got x that x will go to C, this x will go to C; then D equal to C, C is got x; x will go to D; x will go to D; E equal to D that x will go to E. So, whatever was the value of A is directly going to E. So, it is not exactly like a shift register, it is working because the earlier case it was not.

(Refer Slide Time: 23:46)



**Example 6a**

We just reverse the order of the procedural assignments.

What circuit will the synthesis tool generate?

```
module shiftreg_4bit (clock, clear, A, E);
input clock, clear, A;
output reg E;
reg B, C, D;
always @(posedge clock or negedge clear)
begin
if (!clear) begin B=0; C=0; D=0; E=0; end
else begin
B = A;
C = B;
D = C;
E = D;
end
end
endmodule
```

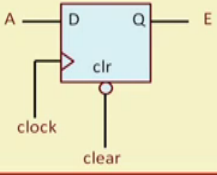
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 10

So, the previous value of D was going to E then C was going to D, then B was going to C, then A was going to B. So, A was not overwriting everything. So, if I change the order; whatever new value I have applied to A that will be overwriting B, then C, then D, then E; all of them will become x.

So, actually the synthesis tool will also do this analysis and it will find out well actually what you are meaning is D equal to A. See the synthesis tool whatever you specify, it will always try to do some optimization even if you specify a function, it will do some minimization and try to get the best circuit here also.

(Refer Slide Time: 25:48)

- The effect of the assignment made by the first statement ( $B = A$ ) is immediate.
- Thus,  $B$  changes, and the updated value is used in the second statement ( $C = B$ ).
- The updated value of  $C$  is used in the third statement ( $D = C$ ).
- The updated value of  $D$  is used in the fourth statement ( $E = D$ ).
- The statements execute sequentially.
  - But at the same time step of the simulator.
  - The four statements are equivalent to a single statement that assigns  $A$  to  $E$ .



The diagram shows a D flip-flop. The input  $A$  is connected to the  $D$  input. The output  $Q$  is connected to the output  $E$ . The flip-flop has a clock input and a clear input ( $clr$ ).

You have provided some specification, but the tool does some analysis and finds that well what you mean  $E$  is actually  $D$  equal to  $A$ , right. So, instead of a shift register, what it will be generating is nothing, but a single flip flop whose  $E$  equal to you know,  $D$  equal to  $E$  equal to final  $E$ . So, the  $A$  will be going directly to  $E$ .

So, the intermediate things are not generated. So, although you might have wanted a shift register to be generated, but the synthesis tool had applied some intelligence and found out that the way you have specified using blocking statements, it is not really a shift register specification, you are directly just assigning the input to the final output. So, it will be generating a single flip flop, right. So, whatever I have said is mentioned, here effect of assignment  $B$ ;  $A$  goes to  $B$ ,  $B$  goes to  $C$ ,  $C$  goes to  $D$ ,  $D$  goes to  $E$ , so because they are executed sequentially. The final result is  $A$  will be going to  $E$ . So, synthesis will be generating a single flip flop like this, right.

(Refer Slide Time: 26:50)

**Example 6b**

Recommended style for modeling sequential circuits.

- Statements can appear in any order.
- Chances of errors are less.

```
module shiftreg_4bit (clock, clear, A, E);
  input clock, clear, A;
  output reg E;
  reg B, C, D;
  always @(posedge clock or negedge clear)
  begin
    if (!clear) begin B<=0; C<=0; D<=0; E<=0; end
    else begin
      E <= D;
      D <= C;
      C <= B;
      B <= A;
    end
  end
endmodule
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 12

So, see if you had done the same thing using non-blocking assignments, then irrespective of the order in which you give the statements, it will still be a shift register; these 4 statements you can do any permutation; you can bring the first statement third last statement first; any order you specify, but the meaning is same all the right hand sides will be evaluated at the same time read and they will be assigned at the same time. So, shifting will be shifting; it will not matter depends on in which order you are just putting or writing the statements in the block. So, here if you are using non-blocking statements, the statements can appear in any order, still it will be a shift register.

So, as you can see; as I mentioned earlier that for this kind of clocked sequential circuits, non-blocking assignment is a much safer option because here is one big example that even if you just interchange these lines by mistake; still your final circuit will be a shift register. It will not change; it will be a right hand side expressions are evaluated all in parallel because they are evaluated in parallel. The order is not important. They will finally, go to E, C will finally, go to D and so on. The old values will go to this, it does not depend on the order in which put these statements. So, it will generate a shift register like this.

So, this is what I am emphasizing repeatedly that whenever you are modeling a synchronous sequential circuit, it is always a very good practice to use non-blocking assignments. So, that the errors like the one, I try to highlight can be avoided.

So, with this we come to the end of this lecture.

Thank you.