System Design Through Verilog
Dr. Shaik Rafi Ahamed
Department of Electronics and Electrical Engineering
Indian Institute of Technology, Guwahati

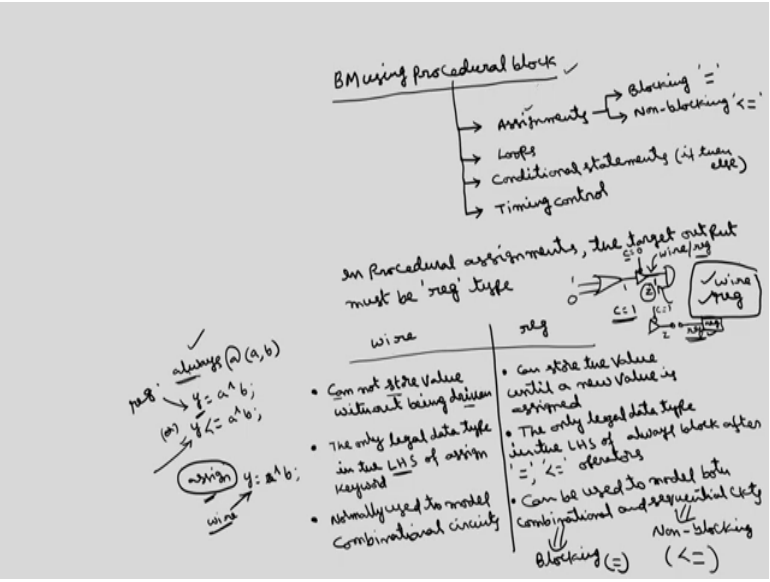
Dataflow and behavioral modeling
Lecture - 17
Examples of behavioral modeling

In the last lecture, so we have discussed about the basics of Behavioral Modelling and then we have discussed some examples AOI gate, AND-OR invert gate, with and without tri-state buffer, then we have discussed 2 by 1 multiplexer and 4 by 1 multiplexer. But, in these examples we have used continuous assignment.

So, we know that in the basics of VLSI Verilog; so we have discussed about the two types of assignments one is continuous time, another is a procedural assignment. So, one of the very important feature of this behavioral modelling is it enables these procedural assignments by means of a procedural block.

So, today we will discuss about how to implement this same multiplexers using procedural assignments.

(Refer Slide Time: 01:36)



So, behavioral modelling using procedural block. So, what is this procedural block? This contains assignments, loops, conditional statements such as if then else, this is similar to the C-language and we can have some timing control also. So, this is one of the powerful block in behavioral modelling.

So, this allows the assignments, loops, conditional statements, timing control. Again in the assignments we have two types of procedural assignments which we have discussed in the basics of Verilog; this can be blocking or non-blocking.

So, in blocking normally we will use = operator, in non-blocking we will use <= operator. So, in case of blocking as the name implies. So, the execution of one statement blocks the execution of the other statement whereas, in case of non-blocking we can execute all the statements simultaneously.

So, we will use in these assignments i.e., in the procedural assignment. So, one of the important feature is the target output must be register type. So, we know that we have different types of the data types; wire, register. So, normally this wire will be used in continuous assignments whereas, this register will be used in procedural assignments.

If you have a look into the differences between the wire and register data types. So, this wire cannot store the data without being driven. Whereas, a register can store the data or value until a new value is assigned. This is one of the important difference between the wire and register. If we take say for example, some gate whose output if I connected through this tri-state buffer to some other gate, if we define this as wire.

Suppose initially if it holds a value of 1, when this control signal of this buffer is 0. So, the output of this buffer is driving this wire; so that as long as this control signal is 0, output of this buffer is 1. Suppose if one of the input is 1 here, this is 1. So, control signal is 1 or 0 depends upon the logic. If I use the negative logic control signal is 0, output is equal to input control signal is 1, output will enter into the tri-state buffer.

Suppose, if I make this C = 1, the output will goes into tri-state buffer this will not hold the previous value 1. So, this can store the values only. So, when it is driven, so it cannot store

the value without being driven. Instead of wire if I define using register even if you make this C = 1, the value of register remains the previous value 1.

Even this is floating, so if this C = 1, then it will enter into the high impedance state which is almost like floating. So, here again this register will store the previous value this is something like register it will hold the previous value, that is the difference. So, register can store the value until a new value is assigned whereas, the wire cannot store the value without being driven.

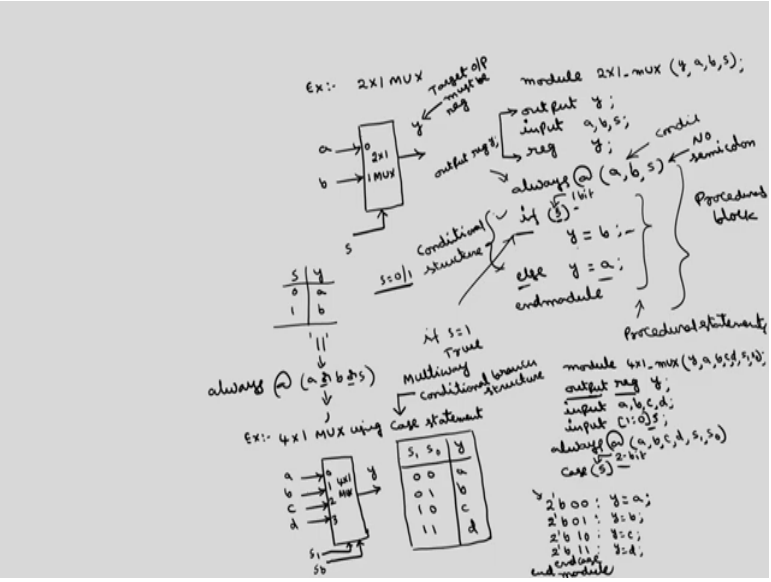
And, the second difference between the wire and register is wire is normally used in the LHS of the only legal data type in the LHS of assign keyword; in case of continuous assignment. If you have something like assign you can have some y = a exclusive OR b. Here this can be wire; in case of assign statement the LHS of this assign keyword is a wire, this is only legal type.

Whereas, register is the only legal data type in the LHS of always; normally this always will be used in the procedural statements, block after either = operator or <= operator. So, example is if I write always @ (a or b) y = a exclusive OR b; or you can have y this is non-blocking, this is non-blocking a exclusive OR b. So, in this target output y must be a register whereas, here this is wire.

So, in always LHS should be register, in assign LHS should be a wire, this is another difference. So, wire is normally used for modelling combinational circuits whereas, register can be used to model both combinational and sequential circuits. So, normally in case of combinational circuits we will use blocking statements; means we will use the operator =, whereas in sequential circuit normally we will use non-blocking operator <=. So, these are the main differences between the wires and registers.

Now, in the last lecture we have written the Verilog code for the multiplexer using wire concept that is continuous assignment. Now, we will see how we can write the Verilog code if you use the procedural block or procedural assignment; using always keyword.

(Refer Slide Time: 12:24)



So, this is 2 by 1 MUX. You call this signal as select signal s and two inputs you call as a, b output is y. So, we know that the operation is s is 0, output should be a if I connect to 0 and this is going to 1, s is 1, output should be b. So, module 2by1MUX output is y, inputs are a, b and selection signal; output y; input a, b, s. And, because we are going to use procedural assignments so the target output must be register. So, you have to define register y.

So, these two together can be written in a single statement. We can use output followed by register, then y. Then you have to write always @ followed by some conditional statement. So, what is the condition for the output to change? Either a changes, b changes or s changes.

You can write or also here instead of that we can write the bracket also. Here we can write this statement as always @ (a or b or s). Instead of using this OR operator we can use |; or we can simply use simply ‘,’. So, in the latest versions and all ‘,’ will be used, in the older versions | is going to be used. So, always at a, b, c.

And another important thing is here after always there will be no semicolon. If (s) output y = b, else output y = a; end module. This is called conditional statements. As I have told procedural block, this entire thing is procedural block. This will consist of always keyword followed by some condition, followed by the procedural statements. These are called procedural statements

So, these procedural statements can contain the conditional structure or it can have a loop. So, we will discuss in the coming classes the examples and the loops or it can have timing control.

So, here this statement if else s, if this s is true then this will be executed; true in the sense 1. So, in this if s = 1 that is true. So, then this particular statement is executed otherwise else this statement is executed. So, this is the format of if else structure, this structure is exactly similar to that of C language.

So, this is the Verilog code using procedural assignment. And we have used blocking assignment because this is a combinational circuit; normally in case of sequential circuit we will use non-blocking. Now, we will take another example of 4 by 1 multiplexer. Here we will introduce another concept called case.

So, we know that 4 by 1 multiplexer will have four inputs let us call the inputs as a, b, c, d this is connected to 0 1 2 3 inputs and only one output. So, this target output you have to use as a registered data type and here this is s1, s0, two selections will be there. So, the operation is s1 s0 output y; 0 0 it should be a; 01, b; 10, c; 11, d. This is the truth table of 4 by 1 multiplexer.

Now, we are going to write this Verilog code using case statement. This case will acts as a multi way conditional statement; multi way conditional branch structure, if else is single way. Now, if we want to have the multiple here you have only; s can be 0 or 1, only one possibility is there whereas, here now we have two selection signals. So, there are four possibilities. So, this is called multi way conditional branch structure.

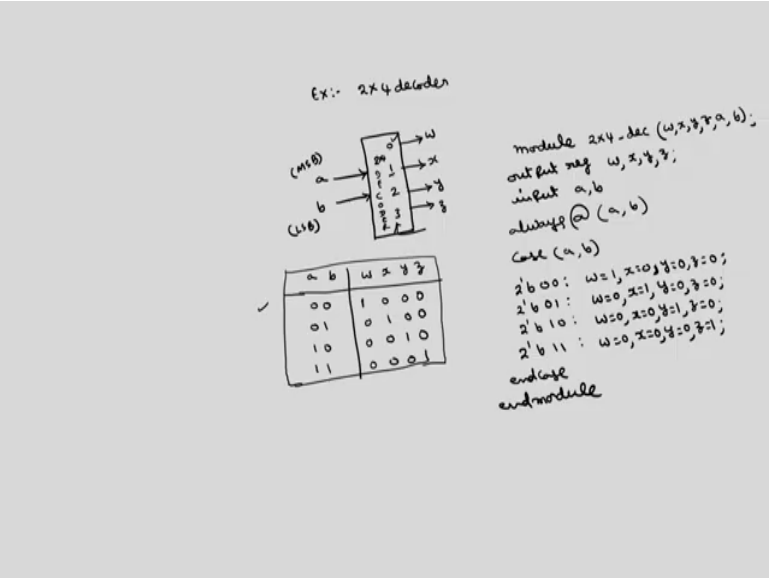
So, module 4 by 1 MUX output is y, inputs are a, b, c, d, s1 s0, output I will include register y. So, these two are two key words separated by tab. Then input a, b, c, d and another input is this is vector we can call this as [1:0] s; this is s1, s0.

Then always using procedural statement @ either a changes, b changes, c changes, d changes, s 1, s 0. Case s, here this s is now 2-bit; whereas in the previous example this s is 1-bit. Then what are the four possible values of these two bits? 2-bit this is a representation of the numbers in Verilog; this can be 0 0, 2 binary, this stands for 2 bit, b stands for binary 0 1 ; 2'b

1 0 colon 2'b 1 1 ;. So, if this is 0 0, what is output y; y is = a. So, 0 means 0 input is connected to a semicolon. Here output y = b; output y = c ; y is = d ; end case end module

So, this is how we can use a multi way conditional branch structure by using case statement.

(Refer Slide Time: 22:20)



Now, we will discuss a 2 to 4 decoder. Another example, 2 to 4 decoder using the behavioral model and that to the procedural assignment. So, this we have already discussed while discussing about the gate level. So, in gate level you have discussed about the internal circuitry also whereas, here now we will use only the block diagram approach. We do not bother about the internal circuitry of this 2 by 4 decoder. We are now interested only in the function of this circuit.

So, 2 by 4 decoder as the name implies two inputs will be there this is a, b you call this as MSB and this is LSB and four outputs will be there. This you connect to 0, 1, 2, 3 we call w, x, y, z. So, I am not considering the enable signal for this decoder. And what is the operation? We are interested here in behavioral modelling only the operation.

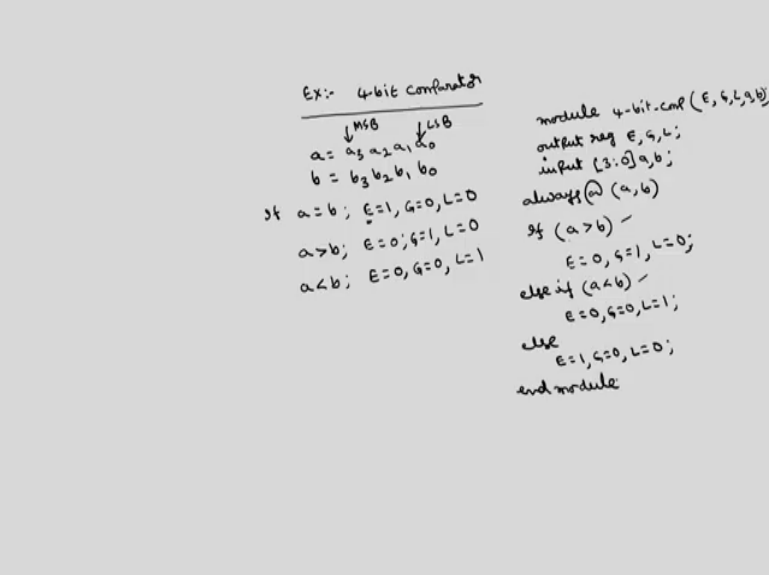
So, a, b; w, x, y, z; this is 0 0, 0 output will be selected; if I use the positive logic selected means 1, not selected means 0. So, 1, w is selected, these are all not selected. 0 1, x is selected because x is connected to 1 input; 1 0, y is selected; 1 1, z is selected. This is the

truth table of 2 to 4 decoder without enable signal and uses positive logic. So, if I know this operation it is enough to write the Verilog code. Whereas, in case of gate level or structural modelling we should know the internal circuitry of this circuit this block.

Module 2by4decoder outputs are w, x, y, z; inputs are a, b; output and all the target outputs we have to declare as a register data type. So, I am using two key words output is one another is register. So, w, x, y, z; input a, b; here also I will use the case statement because we have four possibilities. Always @ (either a changes or b changes) here we are going to use case. 2-bit binary 0 0 output will be w = 1, x = 0, y = 0, z = 0; These are all commas, not semicolons.

Finally, here we have semicolon. 2'b 0 1; this is colon w = 0 , x= 1, y= 0 , z = 0 ; Similarly, 2'b 1 0; w= 0, x = 0, y = 1, z = 0. 2'b 1 1; w = 0, x = 0, y= 0, z = 1; end case end module. This is about 2 to 4 decoder using case statement, similarly you can have comparator also.

(Refer Slide Time: 27:18)



Another example is a 4-bit comparator. This also we have discussed in gate-level modelling. So, in case of gate-level modelling, 4-bit comparator is very complicated circuit. So, the Verilog code takes lot of statements whereas, here we can very much simplify the Verilog code correspond to 4-bit comparator.

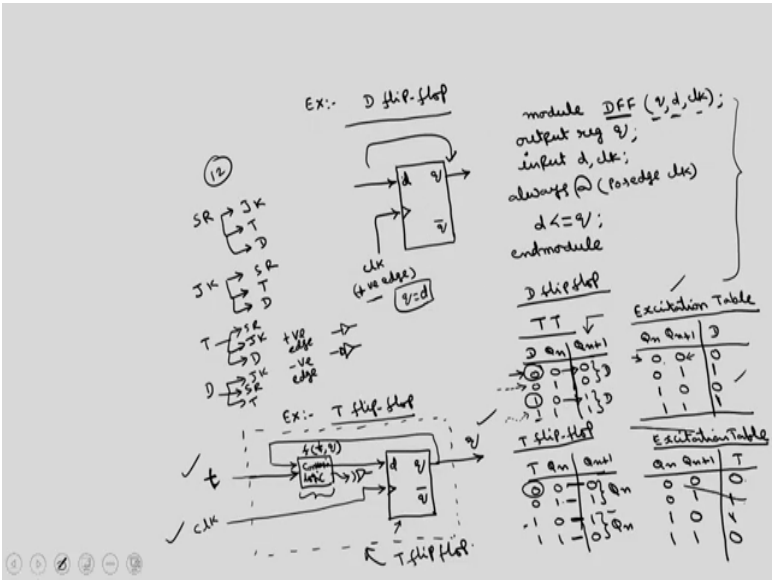
Here, basically we are taking two 4-bit numbers a and b; this is a 3 a 2 a 1 a 0, this is MSB and this is LSB, b is b 3 b 2 b 1 b 0. If a = b we are taking three outputs, equality is 1, greater than is 0, less than is 0. If equal equality output will be 1. Similarly, a >b, equality is 0, greater than is 1, less than is 0. If a < b, equality is 0, greater than is 0, less than is 1. So, we have two 4-bit input numbers and outputs are we have three outcomes one is equality, another is greater and next one is lesser.

Here also we can use this always statement and case statements. Module 4-bit comparator we can use the output as E, G, L; inputs are a, b. Here I will write single bit only, later I am going to define this. Output as well as register E, G, L; input this is vector [3:0] a, b. Then always @ (a , b), if a or b changes; so, this a stands for if any of the bit changes a 3 a 2 a 1 a 0, similarly b means any of the bits b 3 b 2 b 1 b 0 changes.

If a > b, then E = 0, G = 0, L= 0, G = 1; Else if a< b; this a less than or a greater than this will take the 4-bit as a whole instead of taking as individual bits. So, E = 0, G = 0, L =1. Else, the only possibility is if these two are false, then a = b. E = 1, G = 0, L= 0; end module.

So, this is very much simplified and if you take this structural level or gate level modelling, so it will contains many statements and the Verilog code is very much complicated. So, there are a few combinational circuits. Now, we can use the sequential circuits also.

(Refer Slide Time: 32:04)



If I use another example of a D flip-flop, sequential circuit. Here also we need only the overall operation of the D flip-flop, we do not need the internal circuitry which we have discussed in the earlier classes to describe the structural or gate level modelling.

So, this is D flip-flop with d input, q output, q bar output of course, and there is a clock. This can be at the level trigger or edge triggered if I assume that this is positive edge triggered. As I have told here if there is no bubble we will use > symbol for positive edge triggered, o> symbol for negative edge triggered. Whereas, a D flip-flop can be a level triggered also there is no problem with race around in case of D flip-flop because in D we between J and K we will be having inverter.

So, JK can be 0 1 or 1 0. So, in 0 1 combination of JK and 1 0 combination of JK there is no race around condition. The race around condition occurs only when J = K= 1 whereas, in D flip-flop that condition never occurs. So, we have seen that this output q=d simply.

So, at every positive edge. So, this output q becomes simply d whereas at other three portions of the clock which is a negative edge, positive level, negative level the output remains in the previous state; at only positive level whatever the input d is transferred to the output.

So, module I will give the name as DFF because I am going to instantiate this in the later examples. So, there is two inputs and one output q, d, clock; output and also we have to declare as a registered data type q, input d, clock. Then always @, this is positive edge, the keyword is posedge; posedge of clock. So, we have to assign d <= q. Here we are going to use the non-blocking statement <= then end module.

Now, finally, I want to construct a counter, so for that we require T flip-flops. So, this is the code corresponding to D flip-flops. Now, using this D flip-flop I will first construct the T flip-flop, then using four such a T flip-flops I will implement a 4-bit ripple counter.

Now, I want to implement a T flip-flop by instantiating the D flip-flop. So, given a D flip-flop, this is d, q, q bar with clock. So, we have to use some conversion logic here with input is T over all. So, output should be d, this can be a function of q also. This combinational logic is basically a function of T and q; we can use lowercase letter it is up to you.

Now, over all we will be having this clock here is of course, common this clock will be clock for even this T flip-flop also. So, this is the clock clk. Now, this will be your T flip-flop inside this block we have T flip-flop, D flip-flop and a conversion logic.

So finally, this is the output of T flip-flop also. For the overall T flip-flop, we have one input T, another input clock, output is q, but inside this we have a D flip-flop. We can instantiate this D flip-flop, but the only thing is we require this conversion logic.

To obtain this conversion logic we should know these excitation tables of D flip-flop and T flip-flop. So, this excitation tables can be derived from the truth tables. Say if I take the D flip-flop, what is the truth table? D, Qn present state next state is Qn+1. 0 regardless of the previous state Qn+1 is 0; 0 even in previous state is 1, it is 0 simply D. If D is 1 regardless of the previous state this will be 1.

Then excitation table is one of the important table which will be used for deriving the conversion logics as well as design of synchronous counters. So, this truth table will give the present state as a function of the present input and previous output, because the flip-flop is a sequential circuit in which output not only depends upon the present inputs, but also on the past outputs.

Excitation table on the other hand will determine the input required for a given change of state. So, input will be a change of state Qn to Qn+1. So, there are four possible changes I want to retain in this 0 state only. Before application of the clock output was in 0 after the application of the clock also I want to retain in the 0 state only.

So, what is the D required? Or before application 0 after application of the clock I want to change the state as 1, what is the D required? Or before the application of the clock this is 1, after application of the clock I want the state as 0 what is the D required? Similarly, I want to retain the state as 1, what is the D required?

This you can easily obtain from here, you see here 0 to 0 transition is here. So, the corresponding input D is 0. So, this D is 0; 0 to 1 transition is here the corresponding input is 1, so this is 1. So, 1 to 0 is transition here 1 to 0. So, the corresponding input required is 0; 1

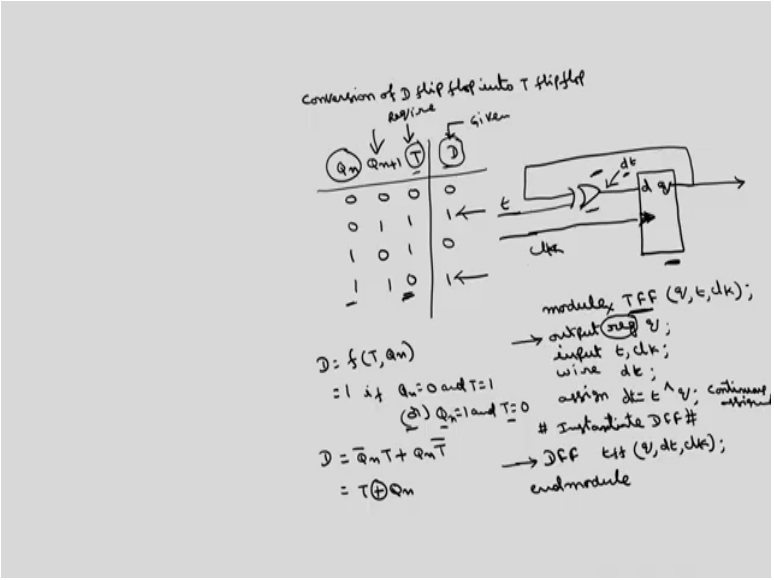
to 1 is this is 1. This is 1 to 1 is here, the input required is 1. So, this is the simply excitation table of D flip-flop.

Similarly, if I consider T flip-flop the truth table is T, Qn, Qn+1; 0 simply Qn+1 becomes Qn. So, this 0 is transferred, 1 is transferred. T is 1 toggles; previous state if it is 0 1, if it becomes 1 it is 0. This is complement of Qn.

So, what is the excitation table? This is truth table. Excitation table of T flip-flop is for a given change of state what is the input T required? 0 to 0 this is 0 to 0, input is 0; 0 to 1 is here 1; 1 to 0 is here 1; 1 to 1 is here 0. So, using these two excitation table I can convert any flip-flop into any other flip-flop. So, there are total 12 combinations of these conversions. There are four flip-flops SR, JK, T.

So, we can convert SR to JK, SR to T, SR to D. Similarly if I take JK; JK to SR, JK to T, JK to D. Similarly if I take T; T to SR, JK to D, D to JK, SR to T. So, totally we have 12 different conversions, but the procedure is same.

(Refer Slide Time: 42:44)



Conversion of D flip-flop into T flip-flop. So, we have to express D as a function of T. So, you take Qn, Qn+1 and you take T, this is D. This is given flip-flop you have to take at the

extreme end and this is required flip-flop. This is 0 to 0, 0 to 1, 1 to 0, 1 to 1. So, what is T and what is D? T is 0 1 1 0 whereas D is 0 1 0 1.

So, we have to express D as a function of T and Qn. This Qn+1 I am writing to just fill these entries of D and T and after that we can neglect this, I want to basically express this D as a function of this T and Qn. So, what is the expression? So, wherever D=1, you read the corresponding terms. So, D = 1, if Qn = 0 and T = 1 or here also 1, Qn is 1 and T is 0.

So, what is a Boolean expression for D? Therefore, so, this is Qn = 0, T = (Qnbar) T, OR is plus. Hence Qn is 1 equal to Qn (Tbar). This is exclusive OR between T and Qn. So, the conversion logic that is required here, this conversion logic is basically exclusive OR gate. So this is D flip-flop. Here I am writing the lowercase letters because normally in Verilog code we will write lowercase letters.

So, this d is nothing but exclusive OR this is d, this is t, this is qn and this clock is of course common to both this is t. This is the overall circuit diagram of T flip flop using D flip-flop, this is conversion logic.

Now, we can instantiate this D flip-flop to write the code for the T flip-flops. So, the Verilog code for the T flip-flop is module I am using T flip-flop is the name. So, what are the inputs and outputs? Input is t, clock; output is q. This is the overall output of the T flip-flop; q, t, clock; output reg q, input t, clock.

Now, here this bit is if I call as this as dt this we have to define as a wire, because this is combinational circuit. So, here we have defined this dt as a wire. Then how to obtain dt from t and q? This is a continuous assignment; assign d = t exclusive OR q, this is continuous assignment.

Once if I obtain this dt, then I can instantiate D flip-flop. So, D flip-flop whatever the name that is given in this one we have to take here as it is D flip-flop and I will give overall this one as some t flip-flop. This name I have to use if you want instantiate this D flip-flop. So, what are the outputs and inputs of this D flip-flop; dt is the input, output is q, inputs are the order in which I have given this input and outputs. Order is q, d, clock. If I follow the same order, in place of d we have dt, clk; end module.

In fact, here another important thing is here out this no need of this register here, because in this D flip-flop we have already defined that as a register. Here we have already defined registers. So, no need to define here this is about conversion of D flip-flop into T flip-flop. So, using this T flip-flop we can construct the counters. That we will discuss in the next lecture. Thank you.