Digital Control in Switched Mode Power Converters and FPGA - based Prototyping

Prof. Santanu Kapat
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur

Module - 07
Introduction to Verilog and Simulation Using Xilinx Webpack
Lecture - 64
Behavioral Modeling in Verilog HDL for Sequential Digital Circuits

Welcome back. So, in this lecture, we are going to talk a little bit about Behavioral Modeling in Verilog HDL for Sequential Circuit Design.

(Refer Slide Time: 00:33)

So, here we will first you know we have already discussed a little bit about Verilog behavioral modeling and we will take some examples. And, here we want to give some guidelines about Verilog behavioral modeling.

(Refer Slide Time: 00:45)

Behavioural Modelling

- Behavioral model helps in evaluating trades offs of various architectures before going to the hardware
- Architecture evaluation occurs in an algorithmic level without going to dataflow or gate level
- It describes the behaviour of the circuit without knowing the logic gates implementation
- It is similar to writing program in C language

So, behavioral modeling will help. You know the trade-off between; that means if the algorithm becomes more complex and if you have a timing synchronism is required. So, in this circuit suppose in the last example we talked about a D flip flop, and to implement a D flip flop, we have gone to know the level of the logic diagram and those actual things are difficult to even sometime remember.

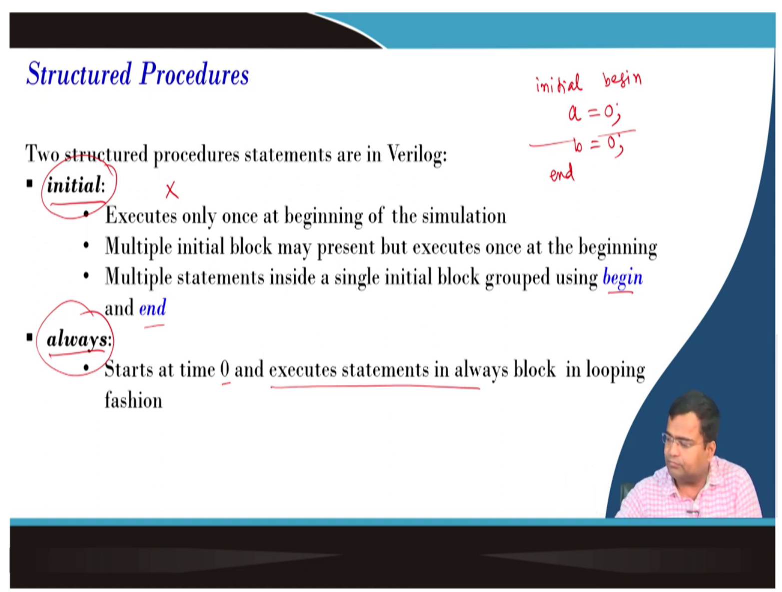
Even you know one may not remember what is the internal circuit of a D flip flop, but D flip flop is a very common flip flop. So, can we just implement a D flip-flop writing you know in terms of a clock edge and something that will be simple? But, Verilog will identify that it is a D flip flop and accordingly it will synthesize.

In this way, the behavioral model helps and you know their architecture evaluation occur is like an algorithm level. And, it will not go into more, we are not going to consider more data flow or gate level. It is more like a little bit of an algorithm level. But, again we have to be careful, we should not be should not go too much into the algorithm level.

So, it will be hard for the simulator or synthesis tool to optimize a circuit. And in fact, when you want to customize or when you want to test point by point you know the timing diagram, it is very important to understand the circuit inside. So, we should mix between some aspects of structural as well as some aspects of data flow behavioral modeling.

So, now, we want to describe the behavior of the circuit and then without knowing the logic implementation; that means, we do not know need to know the inside logic of a D flip flop. But, we still want to realize a D flip flop using its behavior. So, it is somewhat similar to C language, but we are not going too much into this ok.

(Refer Slide Time: 02:42)



So, two structure procedures are there in Verilog. One is we have to write initialize; that means if you want to initialize something; generally, this initialization block only works in simulation. It does not work in hardware because, in hardware, you cannot initialize any register value, whatever is stored has to be taken. But, in the simulation we can customize, we can initialize.

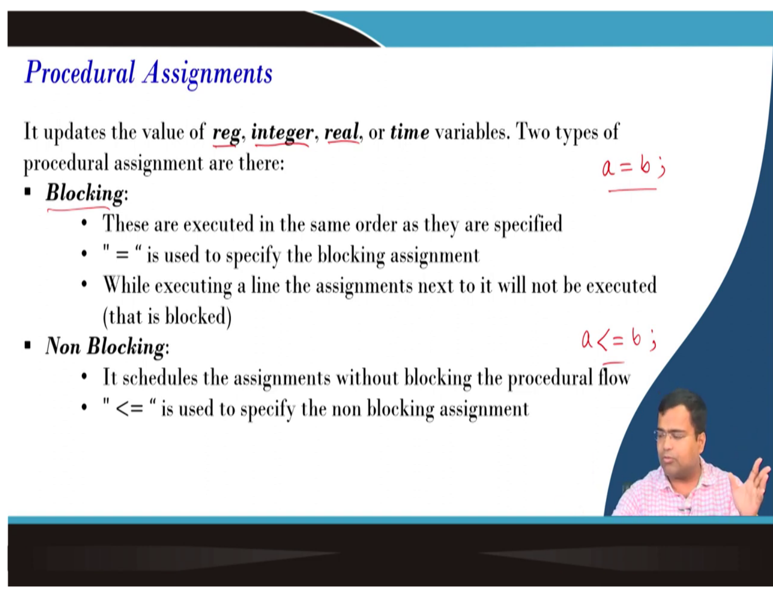
Similarly, if we are using multiple initializations like multiple registers you are initializing, then you have to write inside begin and end statements. More than one; that means, suppose we have a register a which you are initialized like we are writing initial an equal to 0, something like that. If it is a single statement then no problem. But, suppose you want to initialize b equal to some 1, if it is a binary number or it is a decimal number you have to specify b equal to also 0, then you have to write begin and end.

So, more than one step, another thing is always. So, one is initial which only works in simulation, but always is the one which works for simulation and hardware also. The always is executed, it starts with the time 0; whenever you start the hardware or the simulation. And it executes the statement in the loop fashion, depending upon what is the trigger signal.

That means, always will be executed all the time, but whether it will go inside the loop, always there will be a loop fashion whether the loop will be executed all the time or not, that depends on the trigger signal, that is kept associated with the always block. If you do not put anything, then it will always loop and will be executed all the time, but we do not want that.

We want that because it is a synchronous circuit, we want there should be an edge concerning which, or maybe the edge of multiple signals, will decide whether to enter into the loop. And, if it enters into the loop what are the changes that are required and that will be demonstrated using some algorithm?

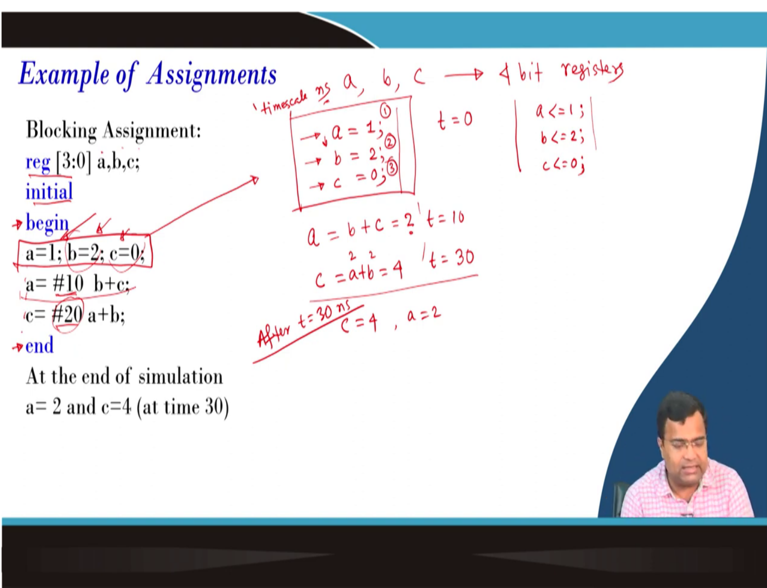
(Refer Slide Time: 04:54)



So, for the procedural assignment, first of all, we need to define a register that can be an integer or we can also it can take the value of real. Then, are two types of one are the blocking statement and the non-blocking statement. And, we will discuss for example blocking and non-blocking. So, only the symbols are different. For blocking, we will write an equal to b and for non-blocking, we will write an equal to b.

There is a fundamental difference between both taking an equal to b, but there is a fundamental difference in terms of the level of. If it is only one level there is no problem, but if there are multiple statements then it makes a huge difference between blocking and non-blocking.

(Refer Slide Time: 05:36)



How does it work? So, let us say we take a blocking example. We are setting a register or 4-bit register a, b, c; all three are registers; that means, a, b, c all are 4-bit values stored in a register that means, 4-bit register. I would say they are all 4-bit registers, where we can store some value. Now, again we are initializing. So, initializing with begin because we have more than one line, even for each line there are multiple you know this subpart.

So, it will begin and end. It should be within this beginning and end. First, we are assigning an equal to 1, b equal to 2, and c equal to 0; that means, it will first write an equal to 1, b equal to 2, and c equal to 0. This will start when the simulation starts at 0 times. At this time, it will first execute a; that means, a will be loaded with a, b will be loaded with 2 and c will be loaded with 0. This will happen one by one.

So, 1st it will be executed, 2nd it will be executed and 3rd it will be executed. But, if the same thing if you write an equal to 1, b equal to 2, and c equal to 0, this will also take an equal to a, a will become 1, b will become 2, and c will become 0. But, this value will actu at the same instant, they will take instantaneously. So, it is a parallel operation or concurrent. But, here it is sequential one by one ok.

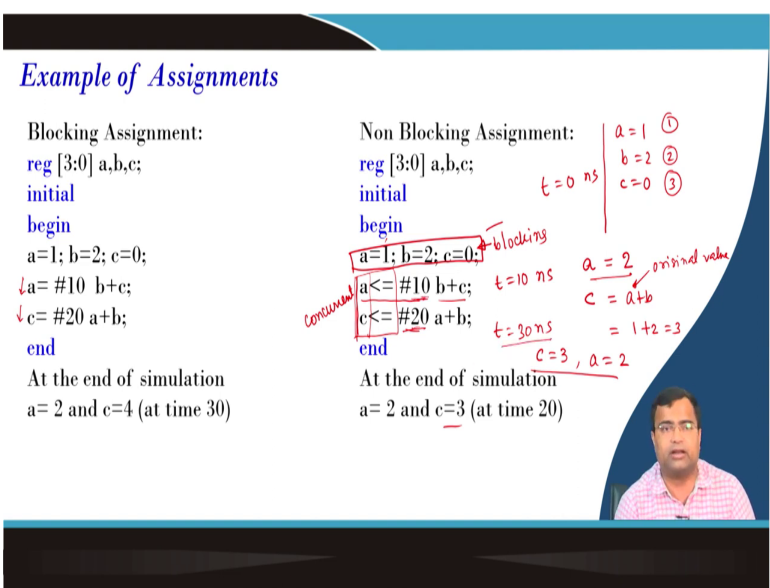
So, now, once this takes then it goes this is the first statement that we have described in two possible cases. So, talking about blocking statement; that means, we are talking about this scenario. Next, there is a 10 unit time; that means, an after 10 unit time; that means, when;

that means, I am telling that at t equal to 10, then a is stored as b plus c. Then, what will be a now? b plus c which is nothing, but equal to 2 right at t second.

Then, after then another 20 units, that means it is after since it starts from 0, 10 units is the interval. So, it is t equal to 10. Next, after 20 units after this time; that means, it is t equal to 30; that means, 20 is the time increment after 20 nanosecond duration executing this line after that. So, t is equal to 30, and c is equal to a plus b. Now, at this time what is a? a is 2. What is b? It is 2. So, it will be 4, because this is 2, this is 2, 2 it will be 4. So, let me add this end of the cycle, our c will be stored as 4 OKs and a will remain at 2.

So, a will be also 2. This is execution after, this is after t equal to 30 nanoseconds because, we have not given anything, but we I have discussed that in Verilog by default whenever you set the simulation time scale, if it is in nanoseconds then it will be 30 nanoseconds. If it is said microsecond, it will take 30 microseconds, and so on. So, these are the time unit and the actual whether it is nanosecond or picosecond, that you have to specify at the top.

(Refer Slide Time: 09:44)



Now, imagine the second option. Now, we are the second option here you see again this will show at t equal to 0. The first statement will be equal to 1, b equal to 2, and c equal to 0. So, this will execute 1st, 2nd, and 3rd because this is non, it is a blocking statement. Here, it is blocking for this particular block. It will go one by one initial value. So, it will take the value, and load it with the value.

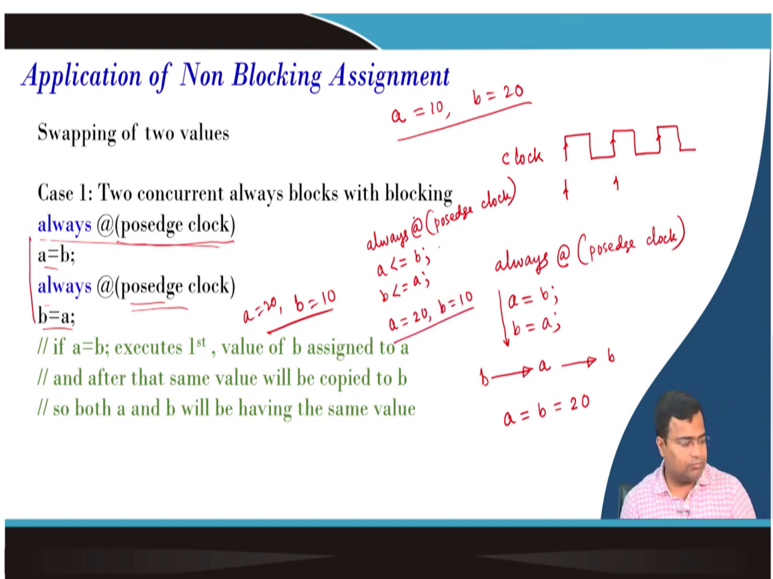
And, there is no conflict because it is just taking constant value. But, at t equal to 10 nanoseconds, 0 nanoseconds what will happen? a will take b plus c ok; that means, what will be my a? It will take b plus c. So, b plus c means it is 2 right? It is 2, it is happening concurrently. Then after 20 nanoseconds, you see while we are showing that this waveform we are showing after 10, 20 nanoseconds.

But these lines are getting executed concurrently; that means, it will whenever it executes all lines are executed the same type. So that means, concurrent they are concurrent. In this case, a will take b plus c at the same time c will be taking a plus b. So, it will not wait for a to fill up with 2. What was a? Our original a was 1. So, it will take the original value of a; that means, it will be a plus b. So, it will take the original value, the original value. What is that? It will be 1 plus 2. So, it is nothing, but 3.

Even though we are showing 20 nanosenanoseconds this time; that means, at t equal to 30 nanoseconds you will see c remain 3 and a will become 2; that means, the difference will be c because these lines are executed simultaneously. But, these lines are executed one by one. As a result, these values of c will be different, though they will be shown after 20 seconds, we are not waiting for 20 nanoseconds more time to execute because they are executing at the same time.

This is a concurrent operation. Only the display will happen after another 20 nanoseconds after this.

(Refer Slide Time: 12:47)



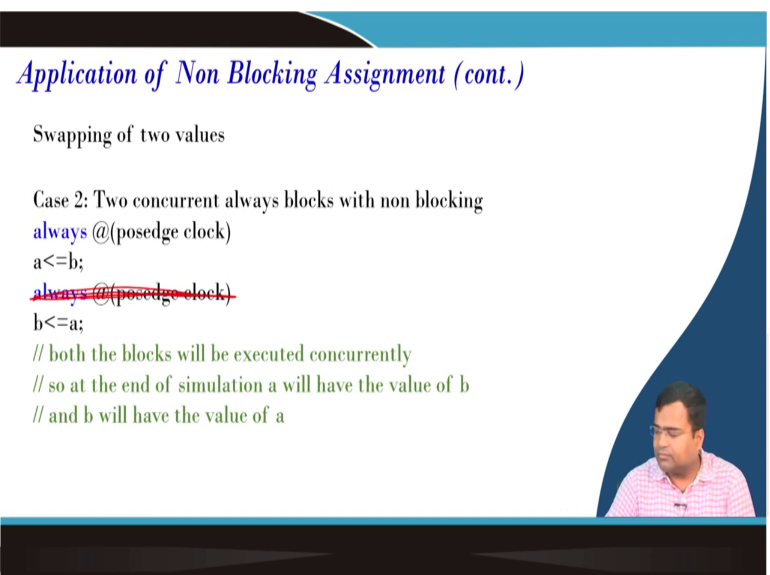
So that means, effectively it will show at 30 nanoseconds. So that means, the application of non-blocking if you want to swap; that means, a and b always at the edge clock; that means, suppose I have a clock, clock which is coming like this. So, typically the clocks are taken with a 50 percent duty ratio in general. So, I will take it like this. So, add pauses because we are talking about the pauses.

So, this is the positive edge, these are edges. So, at this edge, an equal to b, and b is equal to a. So that means, we are swapping; that means, whatever content b has, it will load it will get loaded into a and whatever a had earlier will be loaded into b. So that means if originally a was 10 and b was 20, after this executing this line your b a will become 20 and b will become 10 and this will happen. So, here; that means, the same since we are using always block, that is why they are happening.

But, suppose we use under the same always at the rate posedge clock. Now, we are writing an equal to b, and b equal to a. In that case first; that means, first b will be loaded to a and then a will be loaded to b; that means, whatever value b had earlier will remain the same. That means, in this case, if originally this was the case, what will happen after executing these two lines?

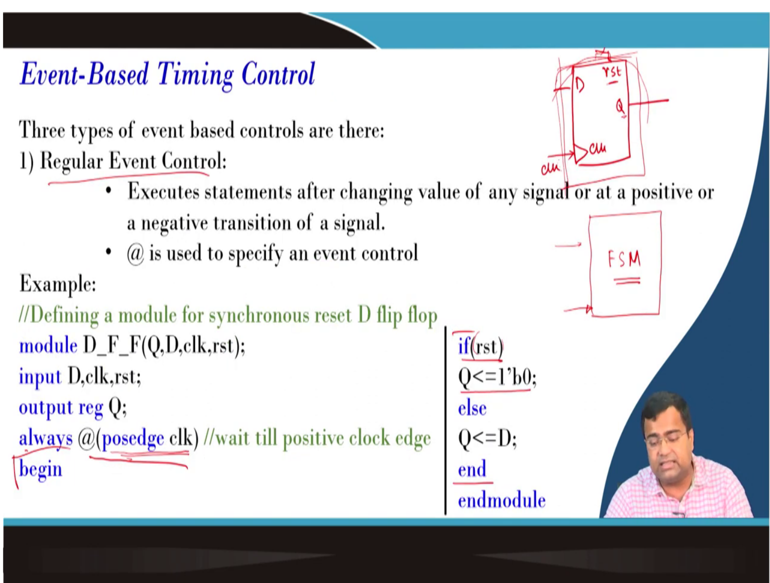
You will get equal to b equal to 20. But, suppose if we write always at the rate the posedge clock, then if we write a less than equal to b and b equal to a, after the execution you will get an equal to 20 and b equal to 10. So, they will simply interchange swap ok.

(Refer Slide Time: 15:22)



So that means, swapping to value you can do concurrently. So, it has two concurrent blocks that can be used with blocking, it is possible that case 1 or using here it should not have another. So, one should be enough, two should have been enough; that means you can use just one block.

(Refer Slide Time: 15:45)



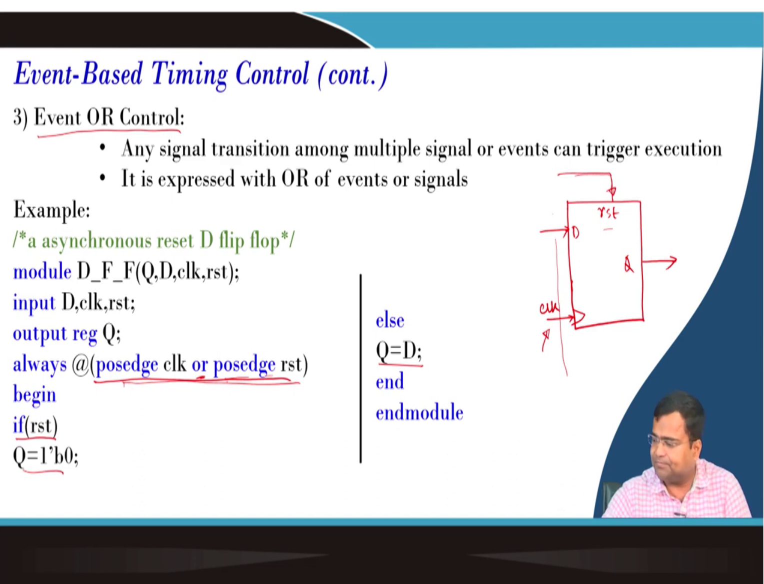
Now, even though there are three types of event regular event control; that means, what we are using. Always at the rate. So, this is a regular event and if we want to implement a D flip-flop; that means, it is just a D flip-flop. What will do? In the D flip-flop, what we will have? We will have you know if we consider a D flip flop, we will have a D block, and we will have a clock. So, this is our clock and then we can have a reset and Q.

So, if it is reset dominated; that means, when the reset is high then the output will be 0, but if the reset is low then at the edge of the clock Q will become D. And, if you write always at the rate clock begin if it is safe; that means, the reset is activated Q will be 1, else Q will be D because it will only enter the clock edge so; that means, clock edge. So, at every clock edge will first see whether the reset is high or not.

But, we have not given any reset edge here, as it reset it is all clock synchronized reset; that means, it is not asynchronous, it is synchronized concerning the clock. Whatever, reset take unless the clock edge comes, it will not go into the block, and it will not check the status. Even, if the reset goes high when the clock is not there, it will not make it 0 because it is only synchronized with respect to the clock.

But if we want to incorporate; that means, this is a D operation you can see the if-else statement and you should start with the begin with the if else statement.

(Refer Slide Time: 17:30)



Now, if we want to let us say consider the second case. This is the case; that means, we have again this D flip flop where this is our reset, this is our D and this is our clock ok and this is Q. In this case if the reset edge comes even though the clock does not come, then it will enter into the loop because there is a or operation. So, either of the edges of this loop will be executed.

And when it goes high, if the reset is high, it will set high; that means, if the reset comes edge come before the clock or after the clock edge, then also the Q will set to 0. And, then if the reset is low the clock comes, then only it will take Q equal to D. So, it is; that means, this is called event or control; that means, it is not just a regular event only the clock synchronizes. You are also synchronizing the clock as well as the same.

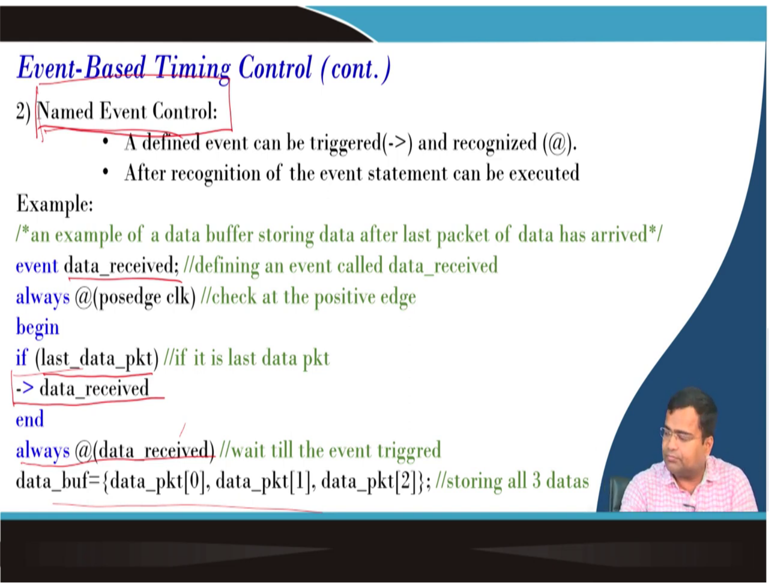
So, when you go to you know if you want to synthesize this circuit, you need to take care about another asynchronous clock because it is not only it is synchronized with respect to this, and the asynchronous clock may come so, we have to reset. So, such a reset is difficult to design automatically. So, we need to customize this because there is an asynchronous clock. But, in the first case what we took is synchronized.

But, here if the clock is very fast then it is ok; that means you might have some delay with you know taking the reset action into the action. Because if the reset is enabled, but the clock is yet to come then you have to wait till the clock edge comes, and still the reset is high, then only Q will be high. So, such may introduce some delay, but it will make use of your automated synthesized tool to design. That means, is all about in Verilog we can customize to an asynchronous level with no problem.

But, if you go to ASIC implementation, you have to separate the block which is a clock synchronized with a master clock. Then, we can write a finite state machine and we can synthesize the automated tool and the tool will do it automatically, we do not have to do it. But, suppose some clocks are asynchronous then we have to segregate, we have to define this block and that block has to be semi-customized.

We have to do it manually; that means, we have to design that CMOS block separately which the automated tool cannot do. So, that is; that means, one regular event and the regular event can be easily know synthesized by an automated tool.

(Refer Slide Time: 20:18)



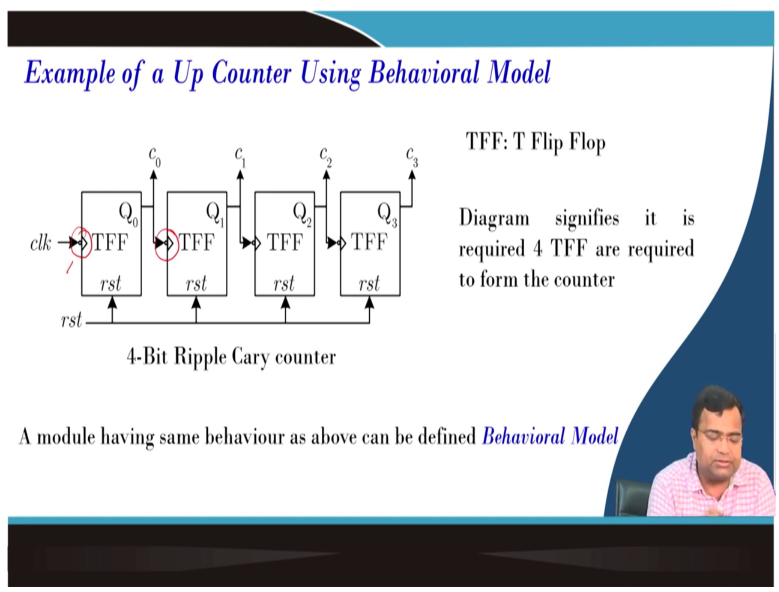
And, if the name event control we can create an event. Suppose, here the event is that data is received. If the data is received; that means, the last packet data is high then we are creating a trigger, the data received trigger. And, that trigger can be used under the always block; that means, if the data received clock goes high, then you can take the necessary action under that

loop. So means, this is one example we can create a custocustomizedt based on the requirement.

And, that event can be because we will be we will require this event whenever we will go for the gate signal generation with the dead time, we will need this name event control also. And, the third one is the event or control, where we have we do not only have a clock that we are using with another signal. And, if we made more and more, it will not be possible to design synthetic circuits with an automated tool.

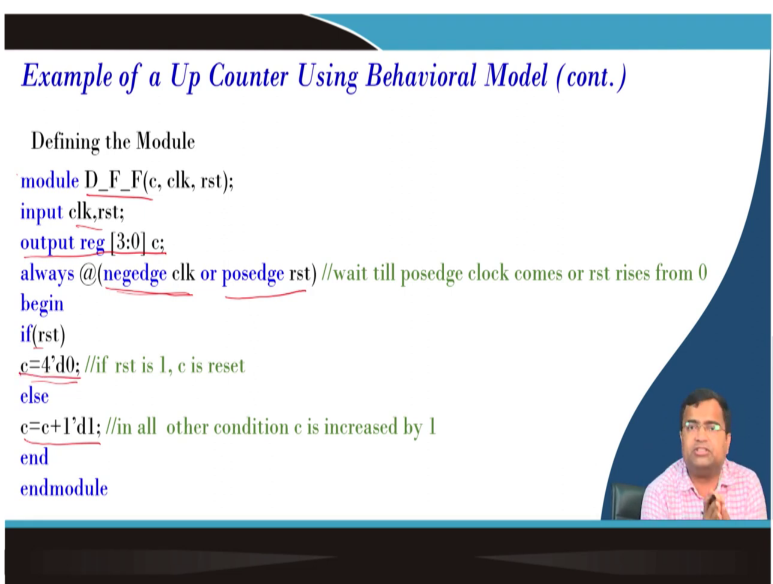
So, we have to customize semi-custom, and we have to manually synthesize this circuit. So, we need to be careful in a Verilog, when you do FPGA prototyping there is no problem because everything will be synthesized and it will be prototyping FPGA. But, I am talking about the ASIC implementation.

(Refer Slide Time: 21:31)



So, we can take a 4-bit you know counter; that means, it is just the clock, we will increment the counter and there is a 4-bit you know reset pass. If the reset is high, everything will be 0. So, this can be easily designed using the behavioral model.

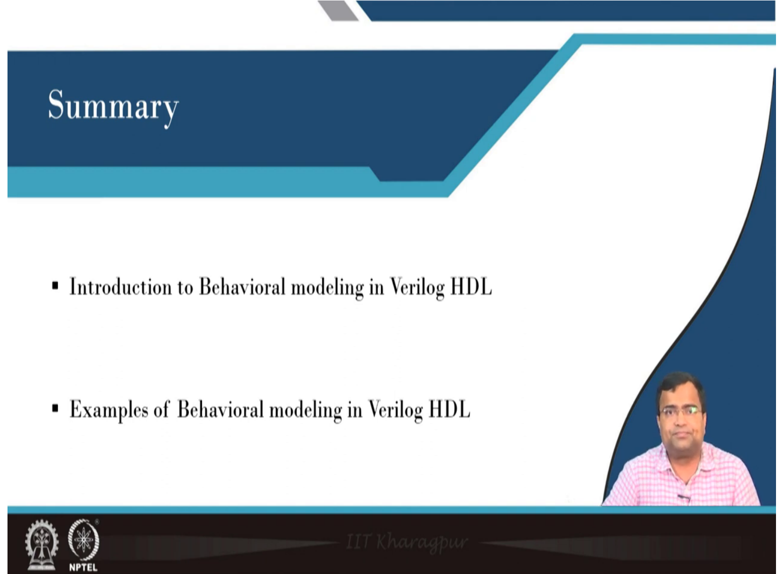
(Refer Slide Time: 21:47)



How? We can define a D flip flop, we have already discussed how to write a D flip flop. Then, under that module I mean this is a reset clock reset. So, we have a register c, always block if the negative edge clock or the positive edge reset curve because you are talking about the negedge of the clock. So, here the clock engagement is considered because there is it is an active low, engagement of the clock. If the negedge comes, then what it will do?

Reset, it will become the reset is high everything will reset otherwise it will keep on incrementing. So, this is the incrementing counter. So, every time the clock's negative edge comes, the counter is incremented by 1. And it is a modular counter; which means, it is a 4-bit. When all 4 bits become 1, then it will automatically get reset. But, if you separately reset then at any point you can 0, make it 0.

(Refer Slide Time: 22:47)



So, in summary, we have discussed behavioral modeling in Verilog HDL and we have taken, and we have considered some example case study case behavioral modeling in Verilog HDL. And, now we will be going to demonstrate how can we simulate using the ISE simulator, which we will be discussing in the next lecture, that is it for today.

Thank you very much.