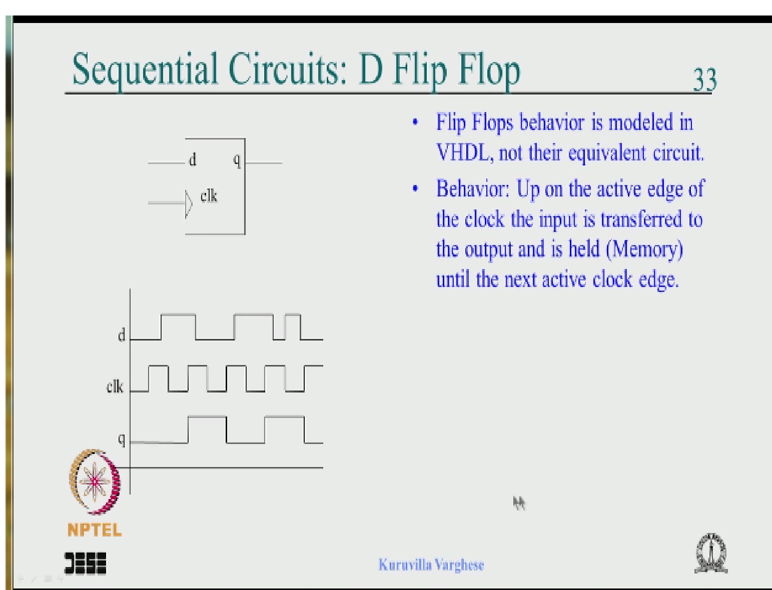
Digital Systems Design with PLDs and FPGAs
Kuruvilla Varghese
Department of Electronic Systems Engineering
Indian Institute of Science - Bangalore

Lecture-17
Synthesis of Sequential circuits

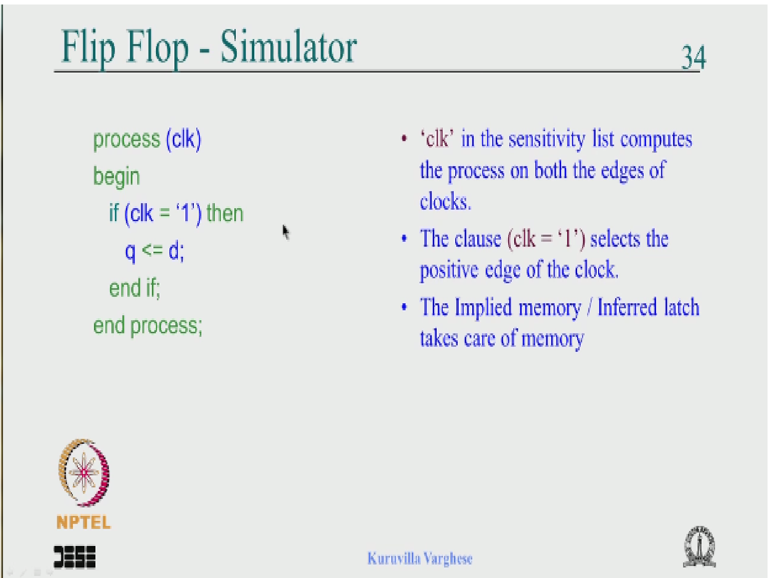
So, welcome to this lecture on VHDL in the course, digital system designed with PLDs and FPGA. We have looking at the VHDL modelling according of the sequential element, sequential circuit like flip-flops, registers and sequential circuit or data path using the sequential element. So, let us look through the starting with flip-flops. So, that things are clear. We will go little back and let us look at the slides.

(Refer Slide Time: 00:58)



So, I said the flip-flop is model by it is behaviour not by what is composed of, so behaviour is upon the clock edge, d equals to q, that is a behaviour.

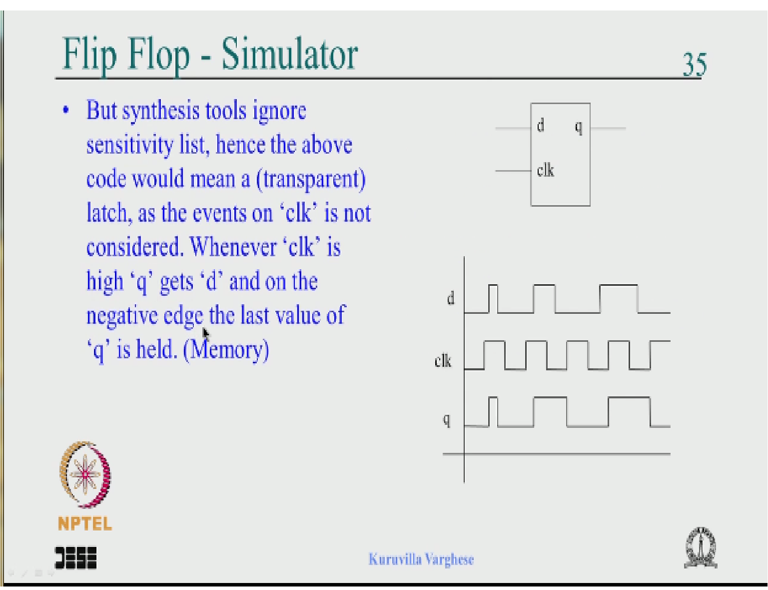
(Refer Slide Time: 01:13)



And we have seen if you write a code like this process clock. And begin if clock is 1 q gets d, end if, end process. Then as for as simulator is concerned there is an event on the clock say a raising edge, then it comes from goes from top to bottom. Then it checks if clock is 1, then q is assigned d. That means this event and clock is equal to 1 makes it is a raising edge. And q gets d, we say end if, that is a memory, so it memorises and it works properly.

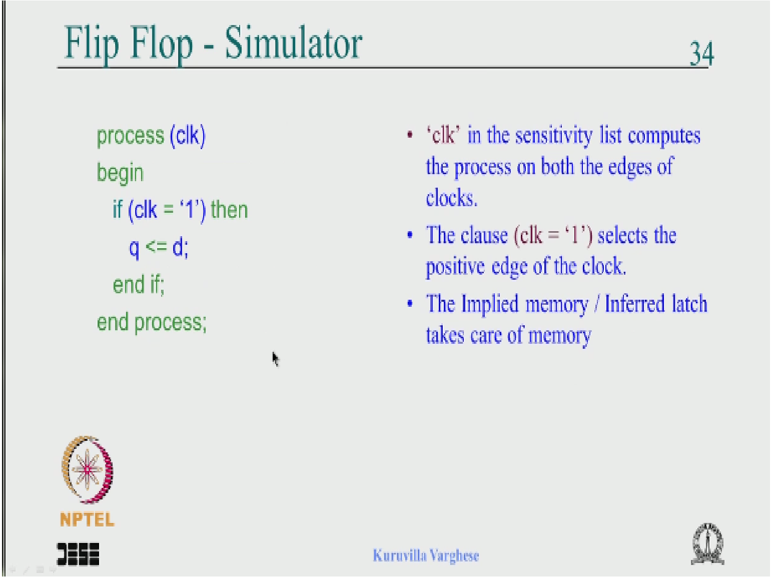
But the problem is synthesis tool is that is not going to worry about that kind of real time behaviour and all that, it looks at the code the code say if clock is 1, q gets d. As long as clock is 1, q gets d, end if. That means when clock goes 0 it you know it is a memory. So, that represent a transparent latch, then a flip-flop.

(Refer Slide Time: 02:15)



As for as the synthesis tool is concern. So, whenever there is clock is high the d comes to q. Whenever it goes low, the value at that point is kind of is latch memorise similarly here. You can see when the clock is 1 q reflects the change in b, when it goes low that is latch till it is enable.

(Refer Slide Time: 02:45)

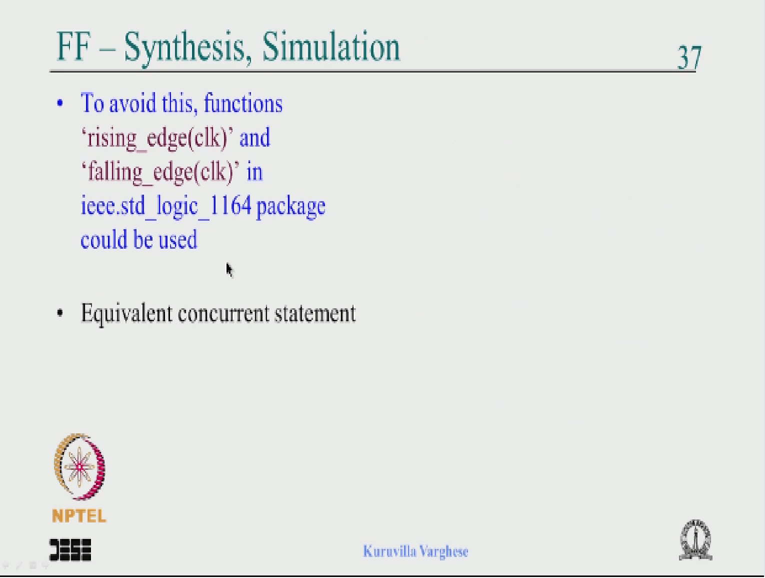


So, the question is that as for as synthesis tool is concern, we have to bring that event part here in the code because that is what is missing from the code. So, we have added an attribute called clock tic event, and clock is equal to 1. Then that represent a raising edge you are hoping okay**.** And q gets d, end if, as for as simulator is concern there is a kind of redundantancy, duplicate statement.

That means this start computing whenever there is an event. Again we say an event. But it does not matter, but the question here is that event does not mean 0 to 1 and 1 to 0 like a it can be u at the beginning of simulation, then you can go to 1. So, as I said this can create some issues like at the beginning of simulation and all that. And that may not reflect in the real life circuit.

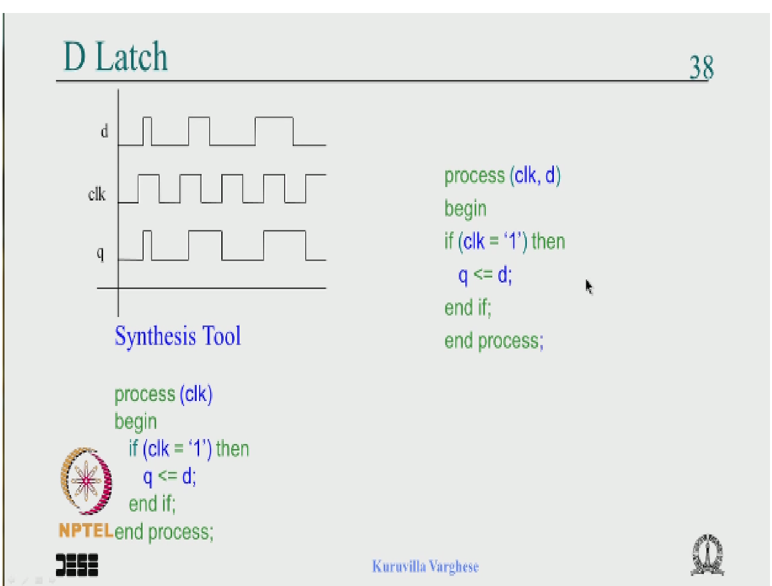
So, you could simulate something and you would think that a different behaviour, and you will get something different in the real life circuit. So, that can be avoided if there is a if the correct raising edge is caught.

(Refer Slide Time: 04:01)



So, there are 2 functions in ieee standard logic 1164 package, which is called rising edge clock and falling edge clock which will correctly you know return through if there is a 0 t o 1 or 1 to 0 transition only thing is that it calls another additional function. It might take a fraction of time extra which is not very significant, you can have an equivalent concurrent statement for this the flip-flop. Say you say q gets d when clock tic event and clock is equal to 1, and you do not mention else. So, that is memory, otherwise q is q itself, you know that that is the meaning of it.

(Refer Slide Time: 04:51)

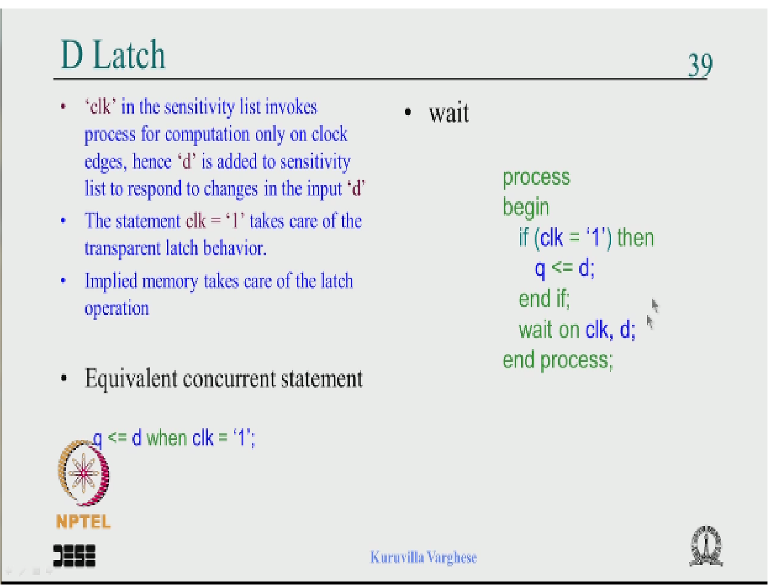


And when we look at the d latch, as for as synthesis tool is concern, if this code is enough if clock is equal to 1, q gets d, end if , that gives a latch for the synthesis tool. But for the simulator there is an issue the moment you write clock that catches a event on the clock and that becomes a flip-flop okay not a latch. So, it means that if the d is changing when the clock is 1**.**

This process is not computed as for as simulation, simulated tool is concern, simulator is concern. So, the solution is to put d in the sensitivity less. So, that when the clock is 1 and there is an event on d that will trigger a computation and it comes here and which say clock is 1, q gets d and that is reflected here. So, for the d latch the code for simulator and synthesis tool is process clock comma d, begin.

If clock is equal to 1, then q gets d, end if end process. So, this works for both simulator and synthesis tool. Synthesis tool looks at this code and synthesise it and simulate, a you know the looks at the sensitivity less. And correctly simulate the behaviour of the d latch.

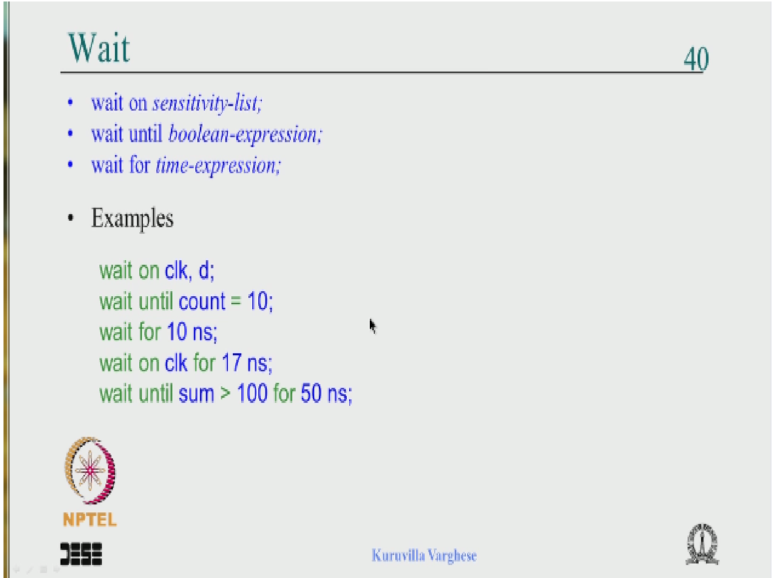
(Refer Slide Time: 06:20)



And you can have the equivalent concurrent statement. Here also you can say q gets d when clock is equal to 1 and else you say nothing okay. And we have also said that it is possible to use a syntax called wait on whatever that was warned in the sensitivity less. In that case you do not write the sensitivity less at the end this cannot be put at the beginning.

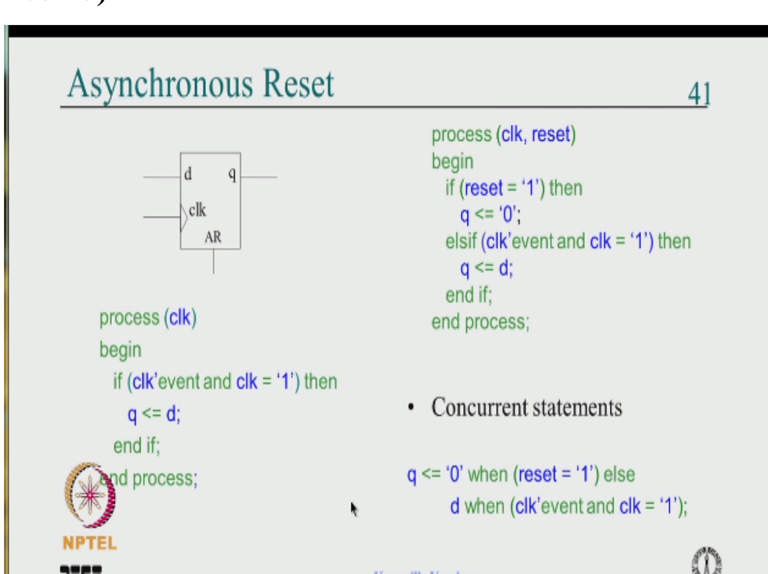
Because that has to be computed then you wait on clock end. And even in process, normal process when you write some process of beginning it is computed once okay. So, that is what is this behaviour, you have wait on clock, comma d in that case you should not write it here. Either of this you can use it.

(Refer Slide Time: 07:09)



And we said that there are different wait syntax wait on, wait until and wait for. Wait for is definitely used for simulation and you have a mix of that and depending on the situation**.** These both are for simulation anyway you cannot use it for synthesis. Because a time is specified, but wait until count is equal to 10, you can think how a sensible circuit can be designed for synthesis**.**

(Refer Slide Time: 07:45)



We have discussed about the synchronous reset a flip-flop with an asynchronous reset. Resets whenever this is active, this is asserted the irrespective of the clock that means it is that is why it is called asynchronous. If there is anything is synchronous then upon the clock that happens. But here asynchronous mean that anytime you make it 1. In this case then the q gets 0**.**

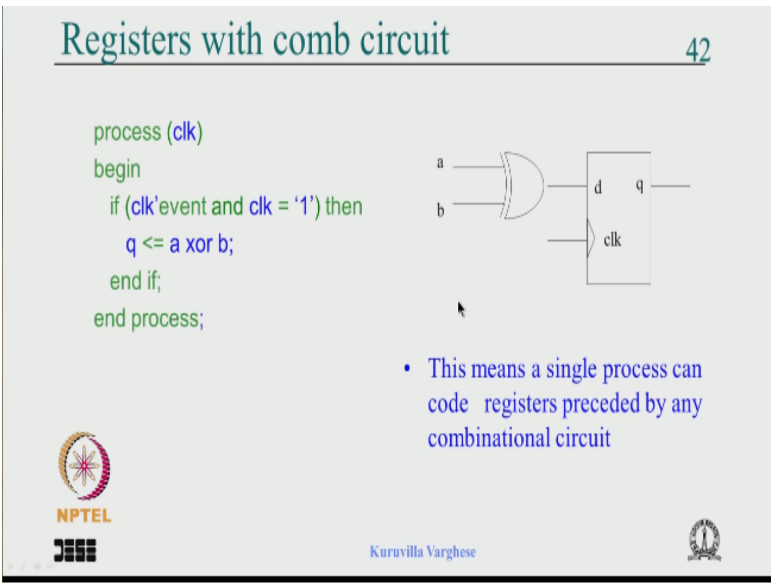
Whatever maybe the clock whether it is high low or transiting though there is a I mentioned there is some kind of relation you have to maintain for the proper operation of the flip-flop. As for as reset is concern not a good time to discuss that here maybe towards the end of the course we will discuss that. But for the time being we assume that it is anytime it can come and it will reset.

And it has priority okay, so we will see how that can be coded. First of all that means that the process has to be sensitive to that one. So, we write the reset here in the sensitivity less then it has priority. So, we will write before if clock event is and clock is equal to 1. The reset the code is shown, if the clock and reset is in the sensitivity less, and we write the code if reset is 1, then q gets 0, else if clock tic event clock is equal to 1.

Then q gets d, end if okay, we do not specify the else and you are also mind you else if not the else okay. We cannot be else, because else only we catch the reset equal to 0. So, else if clock event, clock is equal to 1, then end if. That represent a memory, so this gives you a asynchronous reset and if it is a register of you know multiple input and multiple output. Then you can say others 0.

And this become a vector and this become a vector you get the same asynchronous reset for all the flip-flop element in the register okay. Now you can have a concurrent statement which say that q gets 0, when reset is 1, else d, else else d when clock event and clock is equal to 1, then you do not say else okay. So, here q gets 0 when reset is 1, else d, when clock tic event and clock is equal to 1, and we do not say else, then you get the same thing.

(Refer Slide Time: 10:30)

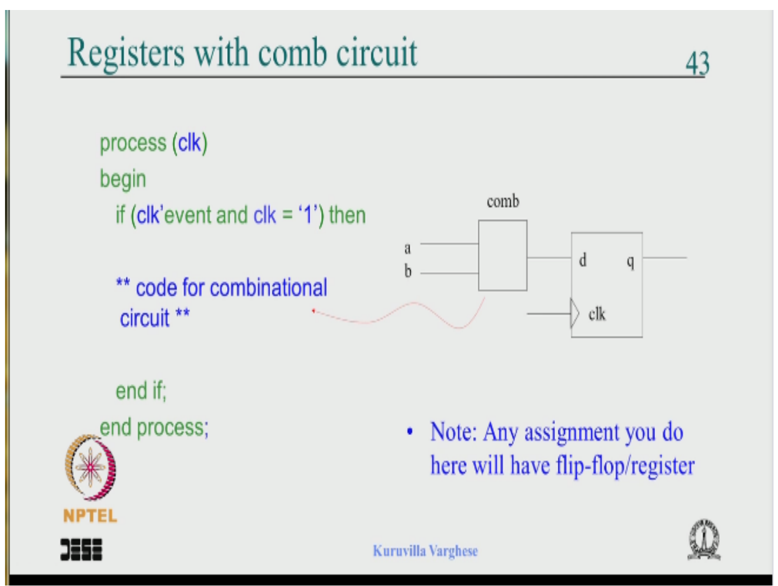


We have looked at this code the last class. Here the difference is that instead of q gets d. We say q gets a xor b okay. And as I said you know that any flip-flop upon the clock d is transfer to q. So, in this case if you are say upon the clock if q is getting a combinational circuit output would mean that you have to connect that exclusive or gate at the input of d. Because the only input which is transferred to the q is d.

So, naturally you have to connect your xor gate here, then upon the clock a xor b will come to the q. So, that is what the synthesis tool is going to do. The exclusive OR gate with a and b as input is connected to the d and upon the clock the q gets d okay**.** So, it means that you can write a single process which can combine a flip-flop and the combinational circuit proceeding it okay, or you can say if it is a register with the vector input and vector output.

You could say a register with the proceeding logic which can be very complex can be coded using a single process okay. So, anytime you see a register and some logic behind it, that can be coded using a single process.

(Refer Slide Time: 12:12)



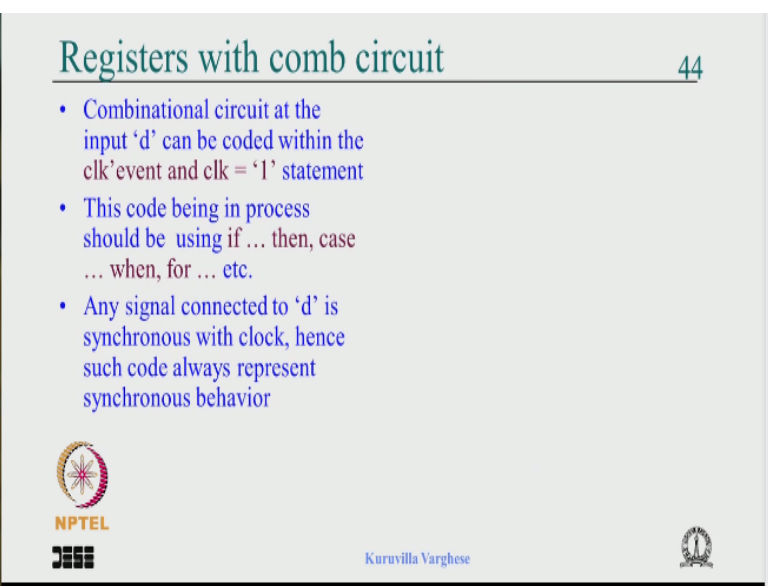
And so, it means that you say if clock tic event and clock is equal to 1and end if and within that you write some combina code for combinational circuit, which can be complex using you can say k is some input is when for each value for each of that cases you can say. If some input and additional inputs and so on, it can be complex code here. But what happens is that whatever you write here comes at the input of the d.

So, this combinational circuit the code of that appears here okay. That is a cracks of the matter and mind you as I said anything you write here okay. You say suppose z get something that z become a flip-flop okay. Any assignment you do under clock tic event clock is equal to 1. You

will get a flip-flop or if it is a vector you will get a register and this should be kept in mind. Otherwise you will carelessly code something.

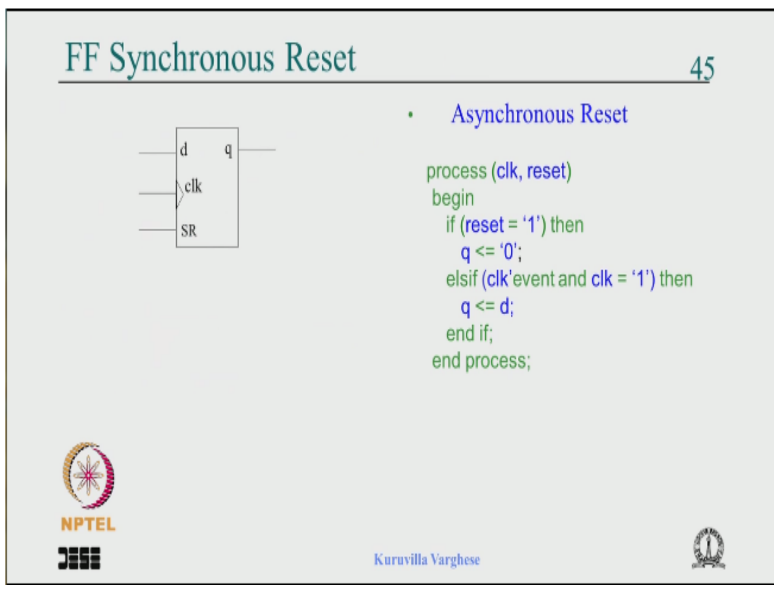
And you will find that everything is delayed by 1 clock or 2 clock, just because you are forgotten that under this clock tic event clock is equal to 1. Everything is synchronous it means a flip-flop at the assignment you know. Whichever output you get a flip-flop that should be kept mind. So, that is what I am saying that this a register or a flip-flop with a proceeding combinational circuit can be coded as a single process like that okay. So, that should be kept in mind.

(Refer Slide Time: 13:56)



And now let us move to the synchronous reset okay.

(Refer Slide Time: 13:59)



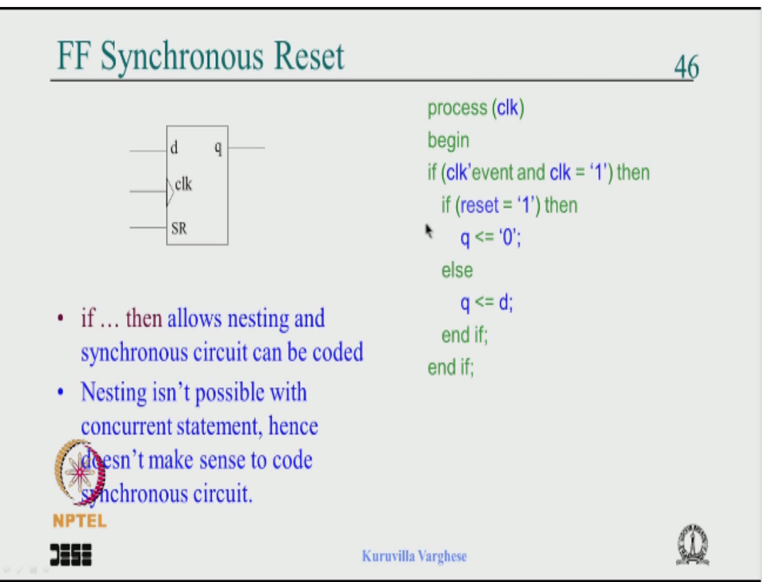
Now as a name suggest the synchronous reset would mean that if it goes active, the q will not go 0 immediately upon the next positive clock edge, the q will become 0 okay. So, you assert it anytime. But the q, the flip-flop is reset, when the next clock comes that is the meaning of synchronous reset. And we have seen the code for asynchronous reset, which has priority over the clock. So, we wrote it you know the reset we wrote in the sensitivity less.

We wrote the reset before, because it has priority if reset is 1, then q gets 0, else if clock event clock is equal to 1, q gets d okay. But here first of all this is synchronous with the clock it has no priority. So, the first thing do is that the reset has to be removed from the sensitivity less. Because any time reset changes it does not respond. Only when the clock comes it check if reset is 1, then the q gets 0, as for as the coding is concern.

I am not talking about the real circuit part of it, that we will worry a wild later. But I am talking about how to code it you know. When the reset when the clock comes if reset is 1, then q gets 0, else q gets d. So, once the clock comes we have priority for reset over the d okay, that has to be kept in mind. So, that shows that code has to come under here, because it is synchronous to the clock.

So, we are not going to write reset in the sensitivity less, going to remove this we have putting back. If and we say if clock tic event clock is equal to 1, then we say if reset is 1, then q gets 0, else q gets d okay. So, that is the code for the synchronous reset.

(Refer Slide Time: 16:15)



We need not **we** w should not the reset in the sensitivity less, so process, clock, begin, if clock tic event and clock is equals to 1. Then underneath we say because that is synchronous reset if reset is 1. That has priority q gets 0, else q gets d end if okay. And this is end if, because that represent the memory. This if is this end if is for the outer if and this is the inner if okay, this is a nested if.

Now we are able to write a synchronous reset using the process using if syntax or if construct it is because if can be nested okay. Now if you look in the concurrent statement we told the previous slide we put the concurrent statement and in the concurrent statement there is no way to nested. So, it is very difficult to write synchronous reset in a meaningful way it is not that you know.

You could maybe you can write q gets 0 when reset is 1 and clock tic event clock is equal to 1. But it does not kind of convey the right meaning it looks as if you are you know gaiting the clock with the reset and so on. So, it does not convey but may be if the problem is that if the people start writing statement like that the tool vendors will start incorporating such a thing.

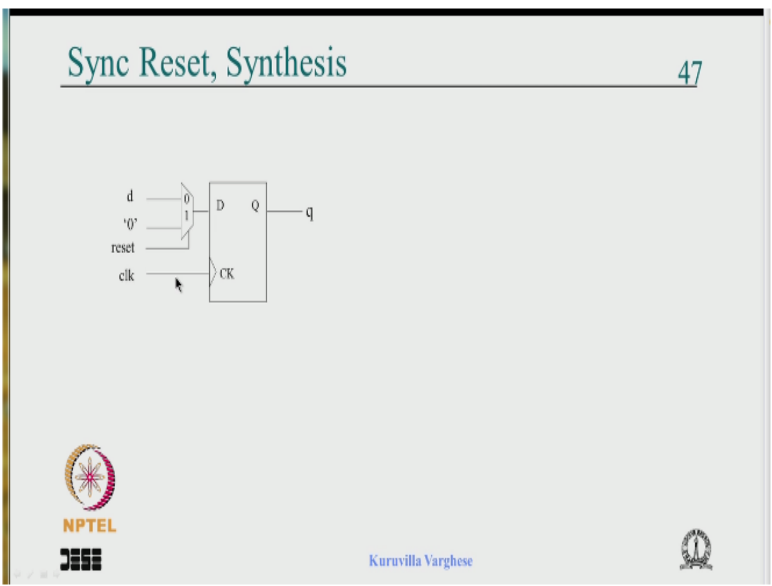
So, I am not really worried were some synthesis tool will kind of go and synthesise such a statement. But what I am saying that meaning wise the code wise it does not make much sense. So, we should not probably try you know coding synchronous reset using the concurrent statement okay. So, that should be kept in mind. So, this is what is stated here, if then allows nesting and synchronous circuit can be coded using synchronous reset can be coded here.

In concurrent statement is not possible it is kind of does not make sense to write a code for synchronous reset. Also look at this statement it say, if reset is 1, q gets 0, else q gets d okay, so if you carefully look at the code. Because I am now going to talk about the synthesis tool, now we talked about the simulator, synthesis tool will look at this code okay. Which say that if clock tic event clock is equal to 1, then if reset is 1, q gets 0, else q gets d.

So, we know that the moment you have if clock tic event clock is equal to 1, end if, that is a flip-flop okay. There is there is a d there is a q that is it. But we are saying additional if, which say that if something is 1, q gets 0, else q gets d. So, if you think what structure is, this then you can think it is a kind of multiplexer 2 to 1 multiplexer, where the select line is reset okay. Which say that if reset is 0, reset is 1, q gets 0, else q gets d.

So, it is a 2 to 1 mugs okay. Now since this is synchronous that 2 to 1 mugs should come should be connected to the d. Because earlier we have seen we say q is a xor b, then the a xor b should come at the d. Naturally this 2 to 1 mugs should come at the input d input of the flip-flop.

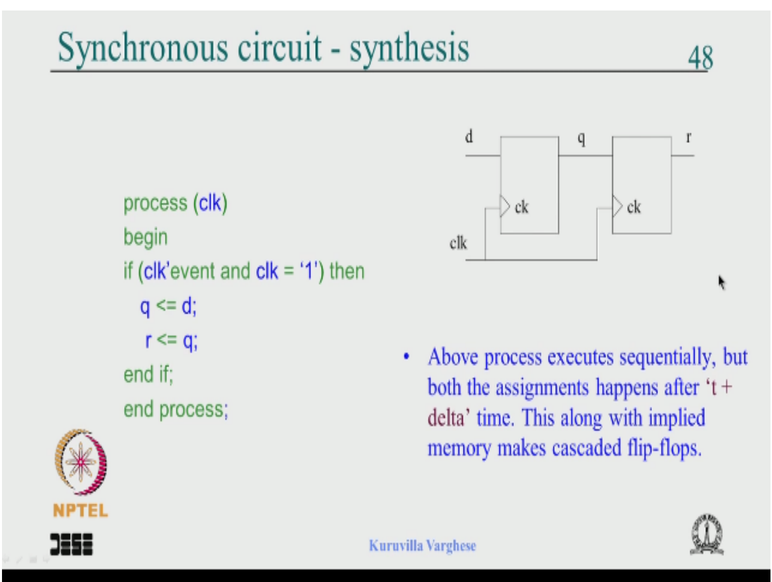
(Refer Slide Time: 20:26)



So, that is what is shown here, so there is a 2 to 1 mugs at the input of the flip-flop at the select line is connected to the synchronous reset and you see when it is 0 and 1. When it is 1 a 0 is connected when it is 0. The real d is real d input is connected. So, if reset is 1, this 0 goes there upon the next clock edge. If it is 0, then whatever is the input directly goes here again upon the clock edge okay.

So, you see that this is how the synthesis tool make sense of the code you write it is natural if clock tic event clock is equal to 1 end if will give you this flip flop. And underneath they anything within that you write will be connected to the d, and there you saying, if reset is 1q is 0, else q is d, that is it 221 mugs. So, that is how the synthesis tool is synthesising okay using then known components okay. We said that that register transfer logic RPL coding is based on the registers and the higher level blocks we have studied. So that is what is shown here.

(Refer Slide Time: 21:54)



So, let us so, that should be clear in your mind. So, let us look at another code okay please have a look at this particular code okay, I have not shown the reset, but mind you in all the code, you please write the reset just because we are learning I did not want to write reset, and you know at to the complications. I want to it concentrate on the real the code so, I remote the reset, but then please whenever you code any register, any flip flop always at reset it okay.

This is just for learning so, I have not written the reset. So, which says now the process clock begin, if clock tic event and clock is equal to1, q gets d and r gets q end if end process. So, it in a very kind of trivial way, if you look at it so, q gets d, r gets q so, you can say r gets d something like that. But mind you by definition this assignment as some delta cycle significance. It is not an immediate assignment.

So, if there is an event on clock at 3 nanosecond, then we know that when the meaning is that this q gets assigned t + delta. So, there is an event whatever may be the value at the d will be assigned q of d +delta. And when come when it come s to the r statement, whatever was the value of q at time t is assigned to r at t + delta okay. So, r will get the previous value of the q and the q will get the new value of the current value of the d after d +delta okay.

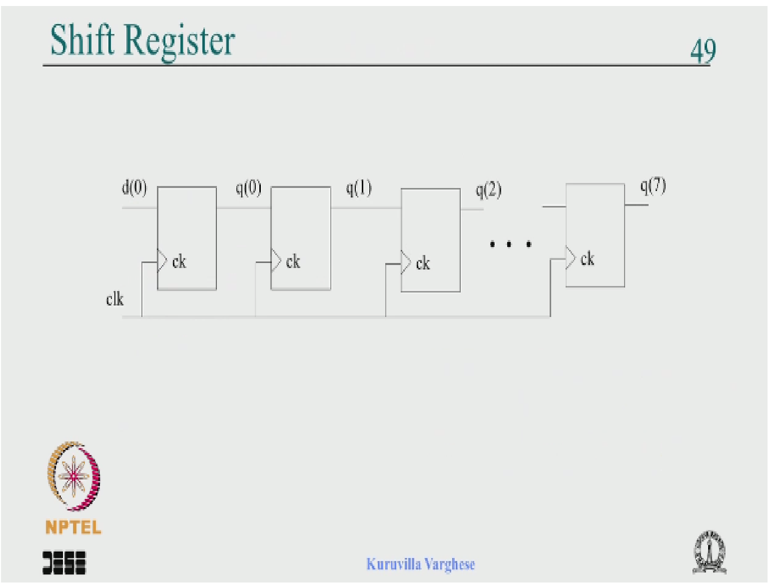
So, you see that it is not at the d the r gets d after d + delta. So, you that the event as a happened at t so, whatever was the value of d at the timing sense, t is assigned at q at t + delta. Now

whatever was the value of q is assigned to r you know, whatever was the value of q at time t is assigned to r at t+ delta, now the game is over. So, the q the r gets previous value of q, q gets a previous value of d. So it is like a chain of flip flops.

It is not a single flip flop, and when the simulation time is moved now to d + delta, then the t + delta value of d will go to t +2delta value of q and so on okay. So, that is how it proceed, but the essence is that you get something like this okay. You will get 2 flip flops, the first flip flop will give you the q, the second flip flop will give you the r okay. So, if you do and that is consistent with whatever we have told you.

You write any assignment, you get a flip flop so, the q gets a flip flop, r gets a flip flop. So, that is what it goes up that gives you an idea, how to code a shift register okay.

(Refer Slide Time: 25:29)



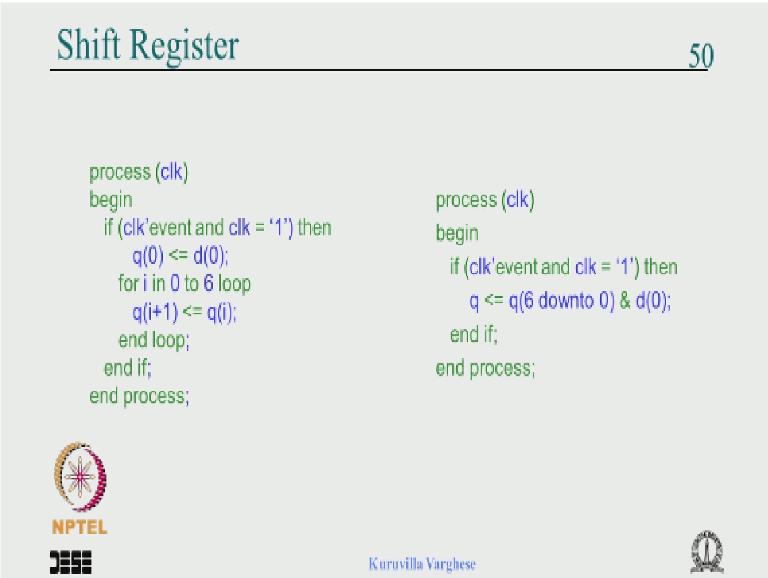
So, let us look at the this kind of a 8 bit shift register okay. I should have shown it kind of instead of left to right, right to left because, I treat q7 as a MSP, but and I hope you just get it we treat q7 as a MSP. So, normally, we write MSP on the left hand side, but you instead of flipping at I did not get time to flip it out. So, assume that the q7 is MSP, so, when the clock comes d0 go to q0.

And q0 go to q1 and so on, q1 goes to q2, and the q6 go to q7. This is just the continuation I have not shown what is in between so, this go code gives you a clue what to write. So, which say that

if clock tic event clock is equal to 1, q0 will get d0, then q1 gets d1, q2 gets d2 and so on. You have to write all the way, now all the way have to q7 okay. Now we know that another way of writing instead of writing it you know each assignment distinctly.

We can write a loop we can say for i in initially, we will make an assignment like q0 get d0. Then we will say for i in like 0 to 6 loop qi+ 1 is qi, so, like that we can say so, that is what I am saying here clock begin.

(Refer Slide Time: 27:06)



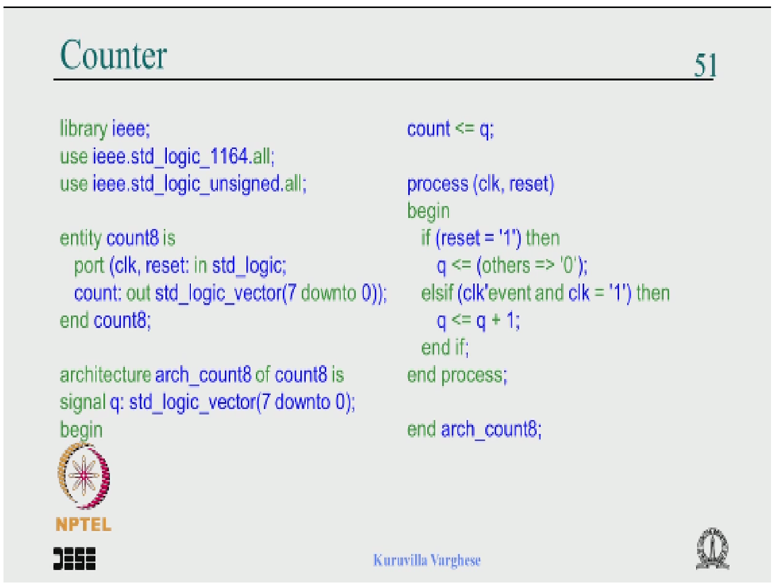
If clock tic event and clock is equal to 1, then q0 get d0 so, that makes a first assignment out of the loop then for i in 0 to 6 loop qi + 1 is qi. So, if the index is 0 then q1 is q0 when index is 1, then q2 is gets q1. When the index is 6 q7 get q0 sorry q6. And you could definitely we have made a probably naming this as d0. If we had named you know like q0, q1 like that. We could have written everything in a loop.

That depends on you naming convention but, there is another way of looking at it instead of looking at kind of bit by bit. We can treat it as a vector okay. So, if you instead of looking at bit by bit, let us look at it as a vector. So, if you look at the vector what happens is that basically q7 to q0 that is q7, q6, q1, q0 is going to get q6. Because q7 is getting q6, q6 down to 0 and d0 so, we can say if you look at it as a vector.

Then you can say q7 down to 0, which is nothing but q as we have defined then will get q6 down to 0, and d0. So, we are viewing instead of kind of bit by bit we are viewing it as a vector. So, that is a more convenient way then writing a loop so, you can say in one short, if clock tic event and clock is equal to 1. Then q which is 7 down to 0 gets q6 down to 0 and d0, and that is if you write 7 down to 0, it becomes very clear the 7 gets 6 and 0 gets d0.

And everything in between in the same model okay so, this is a better way of coding, then this an in one short it is done. It is easy to look at the picture and code it, and it is kind of consistent with the diagrams we do. Because I said, I will not be drawing you know kind of individual flip flop. So, in principle you can show you know 8 flip flops trigger, the with thick lines here which says q is here. And here we say q6 down to 0 and d0, then, in one short it becomes of a flip flop very kind of concise way of representing it. So that is the code for shift register.

(Refer Slide Time: 30:12)



So, now let us look at a code for counter a simple binary counter which is kind of 8 bit so, this counts from 0 to 255. So, you see here the main code I am going to show first so, here we have declared a signal this is kind of it shows how to use signal q standard logic vector 7 down to 0. And we have in the entity clock and reset is input. You see here process clock reset okay, if reset is 1.

Then q gets others 0, because it is an 8 bit output else, if clock tic event clock is equal to 1, q gets q + 1. It says that if reset is there q is initialise 0, otherwise q gets incremented value of the q, that means q gets whatever was the previous value of q +1, because this is at t + delta this t + 1 okay. Now you realise actual output in the entity is called count, which is of try out which is standard logic vector 7 down to 0.

We will not be in a position to write count here, because if you say count others 0 is fine. But here, if you say count gets count + 1 cannot be done. Because count is an out kind of mode output, and output should come on the left hand side of the assignment not on the right hand side. So the trick we do is that, we define a signal internally which is exactly saying data type as this output port. So, we define that signal q standard logic vector 7 down to 0.

And this is not signal has both, you know you can there is no direction like output or input, one side is input, the other side is output. So, we can happily say q gets q + 1, and we say in the before the process are in anywhere in the architecture declaration region. We say count gets q okay so, that is how we use we avoid the use of buffer, because buffer as restriction so, we use signals to circumvents the scenario of using something some output gets output + 1.

So we declare signal and the signal is assigned the real output okay that is the meaning of it. And this + which is now you look the entity clock and reset is standard logic input count is standard logic vector 7 down to 0. So is the q now we are saying that a standard logic vector gets standard logic vector + 1, and 1 is an integer okay. I am not writing 1 within the codes okay so, this +, is not this standard +, the standard +, is for basically for integer.

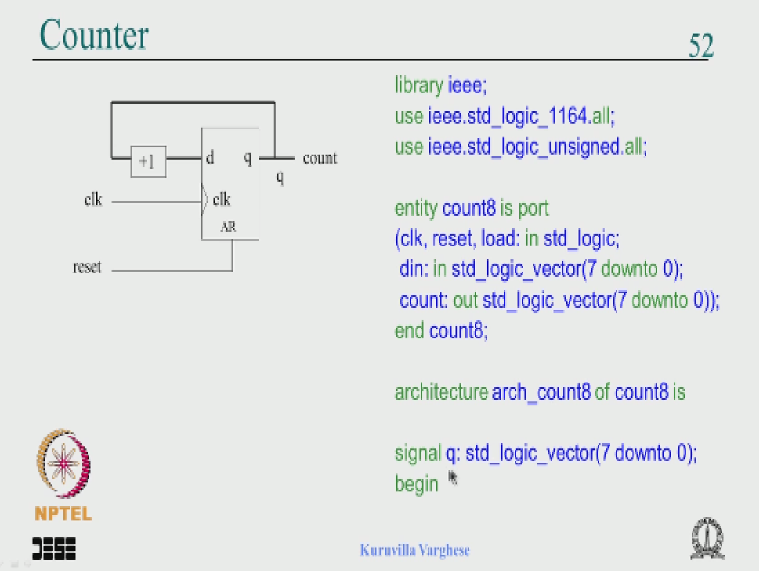
Here one side is standard logic vector, one side is integer so, this operator is over loaded in this particular package use say standard logic underscore unsigned package that is why we say here use ieee standard logic unsigned dot all okay. So, that is the code for the counter once again we have library. We have use ieee standard logic 1164, that is one which is defining the standard logic. We use standard logic unsigned for this + to be use.

And we have clock and reset is input, port as a output since we have somewhere going to say q gets something gets something + 1. We declare in the architecture declaration region we say architecture in name of this particular count it is signal, we say q standard logic vector 7 down to 0. Then begin in the first thing we do is that not to forget is that count gets q, because that is a same thing we circumvent the problem of the right hand side.

You know the output for right hand side by declaring a signal, and assigning the signal to the output. We say process clock reset begin if reset is 1, q gets other 0 else if clock tic event, and clock is equal to 1, q gets q + 1 end if, end process okay. That is how we get a the counter now with the previous synchronous reset, if you look at the code then which say that look at this code which say q gets q + 1 upon the clock okay.

So, this is asynchronous reset, but when the clock comes the q is q+ 1 so, that means that the output as you know taken back to the d, and an incremental comes here and give it to the d that is the meaning of it.

(Refer Slide Time: 35:42)



So, you get such a circuit you have 8 flip flops all the clocks are tight together connected to the clock. All the resets are tight together and connected to the reset. So, upon the reset this is reset and upon the clock q is taken, and given to an incremental and the output is given to the d. So,

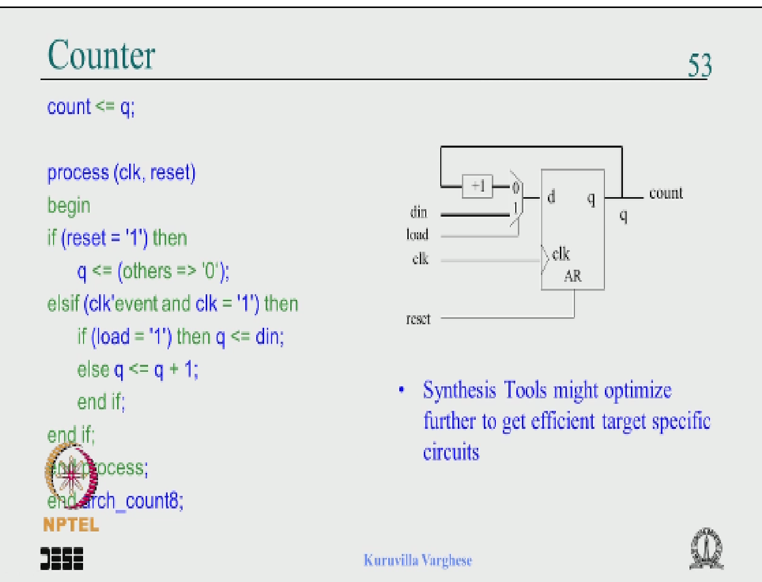
that is the meaning when the clock if clock1, and clock is equal to 1, q gets q+ 1 so, when the clock comes this q gets q + 1 that is how it is synthesise simple.

One incremental at the input d, and you see this is the signal q and this is the output port count. And we say q is coming all the way here, because q is q+ 1 and this is the count output so the q is assigned to the count that is the meaning of it. That is how the synthesis tool synthesise, and now let us make it little more complex. We will write a presentable counter, that means same thing, we have clock reset load okay which is input.

And we have a data input which is 8 bit vector, we have a count which is 8 bit vector output. So, the game is that when the load is 1, and when the clock is it. It is a synchronous load okay synchronous sleep resettable, when the load is 1, when the clock comes this data in goes to the count. And then if the load goes 0 whatever was the loaded value, the counts starts from there okay. So, you please set a value loader value to the counter and start counting from there.

That is the meaning of it so, that is it and we use we are going to use incremental + 1. So, that + come from here so, use ieee standard logic unsigned clock reset load is standard logic in din is again, the preset value in is standard logic vector 7 down to 0. The count is out standard logic vector 7 down to 0 by now, we have very clear that we have to say something + output gets output + 1. So, we declare a signal which is of saying type as count which is used insight.

(Refer Slide Time: 38:18)



And we assigned count gets q and now in the code you say process clock reset, if reset is 1 then q gets others 0 else, if clock tic event and clock is equal to 1. Now this synchronous load start okay now when you say else, if clock tic event and clock is equal to 1 end if that is for you get a flip flop. Now we say if load is 1, then q gets the din so, when the clock comes if load is 1, q gets d. The load is not 1 then q gets q+ 1 okay.

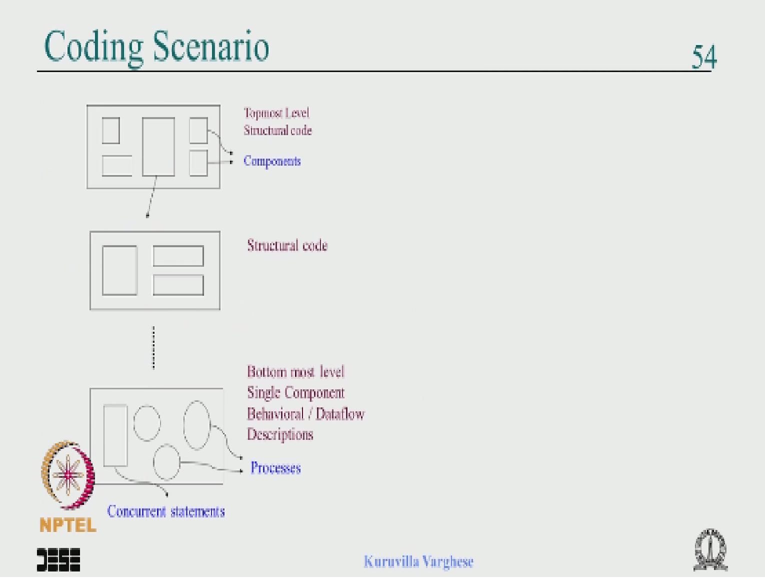
So, that give you so, the load as priority over incrementing the load is 1,and the clock comes does not increment it loads a new value. And if load is 0, then the q gets incremented and any way, we have assigning q to count in the count get incremented. Now once again look at this code this reset upon the clock load is 1, and q gets din, else q gets q +1. This shows a kind of 221 mugs so, with the select line as load.

So, when the load is 1 q that mugs will get din the one part the otherwise zeroth part will get q feedback through an incremental. So, that is what is the synthesis tool is going to do so, it put a mugs here. Because it has priority over the counting so, the load is a select signal of the 221 mugs. When it is 1 because it has priority din is connected to the d so, upon the clock this comes here and if it is load is 0.

Then whatever is here is incremented, and it is keep on incrementing as long as the load is 0. So, that is how the synthesis tool synthesise circuit in terms of flip flops, in terms of mugs, as in

terms of decoders the incremental adder subtractor and so on. So, you can practice some kind of exercise I will as we go long, I will show your real tool how to simulate this and we will see the simulation with the tool and so on.

(Refer Slide Time: 40:43)



So, that is the kind of this sequential circuit coding we have looked at basically today the asynchronous reset. So the reset is coded as a priority before the clock tic event, clock is equal to 1. Then we looked at the synchronous reset, then we looked at the combination circuit before the flip flop for registers, where under the clock tic event clock is equal to 1 any assignment.

You make become a combination circuit at the input of the flip flop or registers so, that is how to do it only thing is that, you are to be cautious so, anything you write, you get a flip flop and we are seeing how to write shift register like if you write two assignment and one get other, then you get it a chain of flip flops, chain of two flip flops. So, you can extend at to form a shift register.

And we have seen how to write a loop to implement a shift register or treat it as a vector, which is much more concise elegant easy to understand, otherwise, if you write a complex loop. Then somebody has to put the index work through the index and so on. Though, writing is as a vector

is much more much better way to code shift register. And we have looked at the counter the counter, how to write the counters we have at looked at how to declare a signal.

And write the code like q gets q +1 because q is a signal you cannot write the count is count + 1. Because count is an output which cannot come on the right hand side. We circumvent this problem by declaring a similar signal and assigning that signal to the output in the architecture statement region in the concurrent as a concurrent statement we also have seen how to write a synchronous reset. So, it becomes as for as synthesis tool is concern.

It becomes a 221 mugs. Because in the under clock tic event clock is equal to 1, we say if reset is 1, q gets 0, else q gets d that represent the structure of a 221 mugs. Then we have seen a similar a presentable counter a synchronous ably lot able counter. So, the load and din is input so, which is synchronous again under the clock tic event, clock is equal to 1. We say if load is 1 then whatever input value goes to the q, else q gets incremented.

So, it again comes as a 221 mugs at the input of the flip flop or the register so, when it is 0 the input goes directly, a when it is 1. Then the q gets incremented or other ways, whatever is the priority okay. So, let us come back to the does the coding scenario so, when we code it, when you have a top level code like a CPU at the top most level a complex circuit not simple circuit like adder and all that so, mostly at the top level.

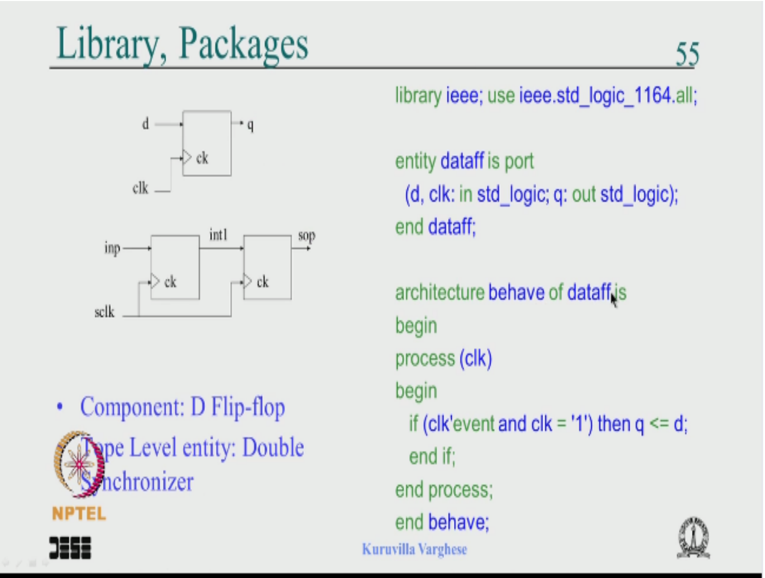
We will have a structural code which is a interconnection of various components okay. So, the case of CPU you have may be registers, program counter, ALU. So, all will be a structural coding mostly and you take ALU say and for that divide into pieces again this could be structural code composed of adder, subtractor logical unit and so on. But at this level when you take further divide you take the adder, then we do not write may be a structural coding may be.

We write some kind of a combination of various concurrent statement and processes at the lower most level okay. So, when you write the component you write it, using number of processes concurrent statement, which are all concurrent. But at the second level the next top level we use structural coding to interconnect and top most level is almost always structural coding.

But this kept maybe combine the structural code and concurrent statement and things like that. But that is how the coding scenario is now we will go little more with the VHDL before we got to the digital design**.** So, maybe because there are some part to be able to do some exercise, we need something called test benches. So, we will complete this kind of sequential part to a decent level.

Then look at the test bench then come back to the maybe will have a tool demo, then we will come back to the digital design then continue with the PLDs, FPGA. Once again come back to the digital circuit and so on okay.

(Refer Slide Time: 45:50)



So, let us look at let us go ahead with libraries and packages these are mainly nothing very serious most of it is syntax. So, we can quickly run through that. But to illustrate the packages, I want to take a very simple example basically in the package, we have trying to write some component and instantiate in a top level code okay. So, let us take a very simple example.

Because if I take a complex example, then we have to understand the circuit the code become big it may not go in a kind of slide. So, I will limited this. So, let us take this is a component a simple d flip flop, one second I am not shown reset for gravity. But in real life always have reset the clock input d and the q. This shows a top level entity which is an instantiation of 2 flip-flops, basic component flip-flop.

And interconnected as a shift register we can say it is a 2 stage shift register, but this as a use which is called a double state synchroniser. Once again we will see what is a double state synchroniser towards the end of the course in timing issues, I will touch upon it because that is I won't have the time to go-through all the details of the synchronisation. But then at least I will state the problem, state the issue and a quick solution using some of the synchroniser okay.

So, this is a simple enough circuit which is a like 2 flip-flops in chain okay. So, this is the first flip-flop, it is the second flip-flop, the d for the first d is connected to the input the first q is going to an internal signal. That is going to the next flip-flop d. And the real output is called sop which is synchroniser okay. This is an internal signal, this is the real input and the clocks are tight together, it is called s clock which is synchronise clock okay.

Now we will see how in a simple code how to we construct this you should be familiar by now. Because we have seen a structural coding of a ripple order using the flip-flops sorry using the full order as a component. So, the same thing we will repeat it, then we will see how to write this component in a package and instantiate it from a package, that is our idea to start with.

So, this is the flip-flop code you say library ieee, use ieee standard logic 1164. Because that is for the standard logic entity data flip-flop, that is this one is port, d, clock is in standard logic, these are standard logic, q is out standard logic, end data flip-flop. Then you write the architecture, architecture behave of data flip-flop is begin process clock begin if clock tic event and clock is equal to 1, then q gets d, end if and process, end behave.

Now I think you have quiet familiar, so I am kind to save space by writing this in a single line. So, normally we used to write q gets d with an intending q gets d. So, that is 1 point you should remember when you write the code do the proper intending okay. So, the VHDL does not have any significance on the carriage written or a line feed. So, you like earlier we use to write library ieee.

Then in the new line we used to write use ieee as for as long as the semicolon is there, you could write it in one kind of line to save space I have written like that. But you do write in a decent form, so in principle you can start entity here. But then nobody will able to understand. So, please write entity separate you know I have given some blank line in between and I have also intended what is inside a bit inside.

So, you can see that in the process the begin, so the what is inside have intended and normally whatever is underneath I will intend it one or two lines, one or two characters. So, please do the proper intending, so that people can read and understand. Suppose if you write another if nested. Then definitely it has to come with a little 2, 3 characters inside.

So, that the people can understand like a in the earlier code here we here. If, else if, end if that is in one kind of line and underneath here if else if in one line. So, if I had written everything in line, then when it comes to end if you have no way to matching and very difficult to understand. So, please use proper indentation, but mind you some people it is better to give some 2, 3 characters do not press a tab.

Sometime because if you keep on intending then come to 3 level of nesting. Then it goes all the way. Out of the screen out of the paper when you take printout also some people have the habit of writing everything in capital letters please do not do that which quietly irritating there no need to write everything in capital. And some people have very bad at you know it looks everything is kind of in a single line that is quite horrible nobody will be able to understand it.

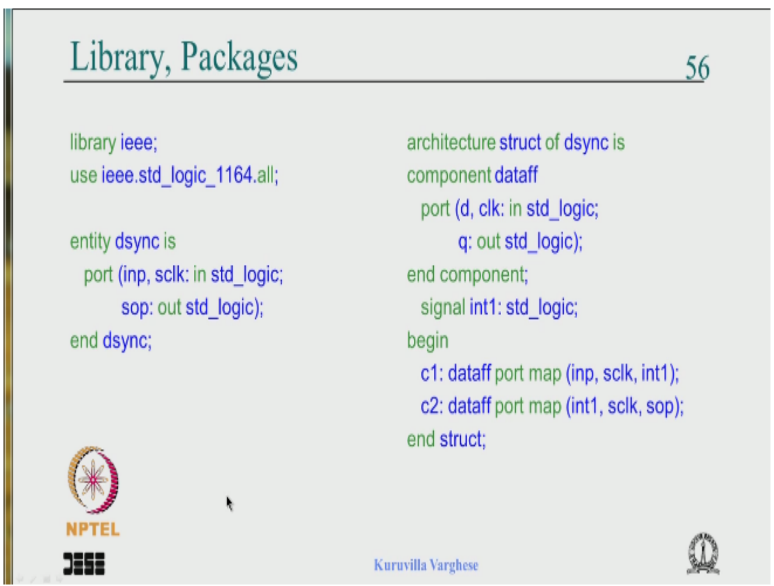
So, please use proper indentation, similarly when you use an operator gives some space around like equal, you could write in principle reset without space equal to 1. But in some places can confused the tool wherever there is there could be need of separation okay. So, please give I should have told this at the beginning, but then we have come to a decent stage of looking at the code you please intend it properly, please write it neatly.

So, that people can understand, because unless you cultivate that habit, if you pick up some bad habits and it proceed and like your code can become clear nobody will be understand that okay.

So, here this is the code for the flip-flop and now we will see that is the library, entity and the architecture okay. Now we are going to write a top level code where this particular component is instantiated.

Assume that this is written in the same project of course you may not know what is the project, I will demonstrate that. At least thing that this is written in the same file maybe and in the next kind of what comes next in the file is a top level entity which, is the double synchroniser, which is this one okay.

(Refer Slide Time: 53:16)



So, we say library use close, then entity double sync is port, input, s clock is a in, sop is out. So, here you have input s clock is a input and sop is a out. And that say that this is an internal signal, because this is now the top level component where this instantiated twice and this is an internal signal. So, we have to declare this particular component, declare this internal signal instantiated type twice okay.

So, we are going to the architecture, architecture some name of this double sync is then we say component data flip-flop. That is the component we have written here, you say port, d clock is in standard logic, q is out standard logic, so end component. So, that is the component declaration and we have to declare this particular signal. So, that is signal if 1 is standard logic. So, we declared after the entity we start the architecture before the begin.

We have declared the component, we have declared the signal. Now we are going to instantiate this 2 times and with this connection, input is connected to the d of the first flip-flop. The output of the first d flip-flop is int1, input of the second flip-flop is int1, the clock is saying and the output of the second flip-flop is sop okay. So, we say begin c1 the data flip flop port map, the input s clock int1.

So, d goes input goes to b, s clock is clock and int1 is q, c2 data flip-flop port1. So, that int1 output goes to the int1 input d, s clock is same, same clock and the output is sop. So, you get the kind of double state synchroniser with the d flip-flop as component and this is how we write normally. Now what we have going to see next is that how to put this in a package and compiled it into a library.

And we write a top level component of double sync synchroniser, with that component now coming from the library rather than from our own file, our own code okay. That is what we have going to see now, we have coming to the end of the lecture. So, I will take that up in the next lecture, so we have seen towards the last part basically how to write a we have going to like we have done the proprietary wall, how to write a component in a package.

And put it to library as an example, we have chosen the case of a top level entity a double sync synchroniser, which is nothing but a tool state shift register composed of d flip-flop. So, we saw how to write the d flip-flop code, how to instantiate in the top level component by declaring the component and declaring the internal signal. So, as I said in the next lecture we have going to see how to put this in a package.

And that package in a library and instantiate it from the library and we will proceed you know how to little more about the standard libraries and so on. So, today we have seen all of that sequential circuit basically the synchronous reset, how to code combinational circuit along with the registers, we have seen some example shift registers, counters, then we have proceeded to the packages and libraries how to write components.

And the at least the beginning part of it so, please go back and revise it try to understand that, if you already using a tool try to write some proper code see how to getting synthesise and so on. So, I wish you all the best and Thank you.