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

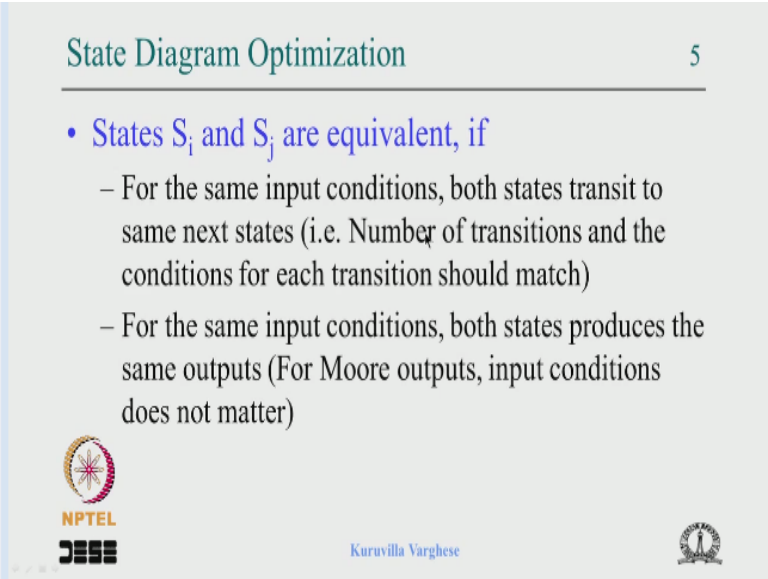
Lecture-26
VHDL coding of FSM

So, welcome to this lecture on VHDL and the course digital system design with PLDs and FPGAs, the last lecture we have seen some issues with finite state machine basically we have seen how to optimise state diagram. Basically the algorithm, then we have looked at the output races and what is solution to that problem. Then something about the selection of flip-flops and you know kind of advantages in choosing a particular type of application.

Actually we have more some more issues to be handled in the case of finite state machine**.** But I think to discuss that it is better we look at how to do the VHDL coding of finite state machine that will bring clarity to that those portion not that lot of coding is involved. But whatever little that can be shown there will you know bring clarity and you know put things together.

So, for a at least 1 lecture or 2, we will concentrate on the VHDL coding. So, before that we will have a quick look at the last lectures slide. So, let us go to the slide**,** basically this is what we have seen when you have a state diagram there could be equivalent states.

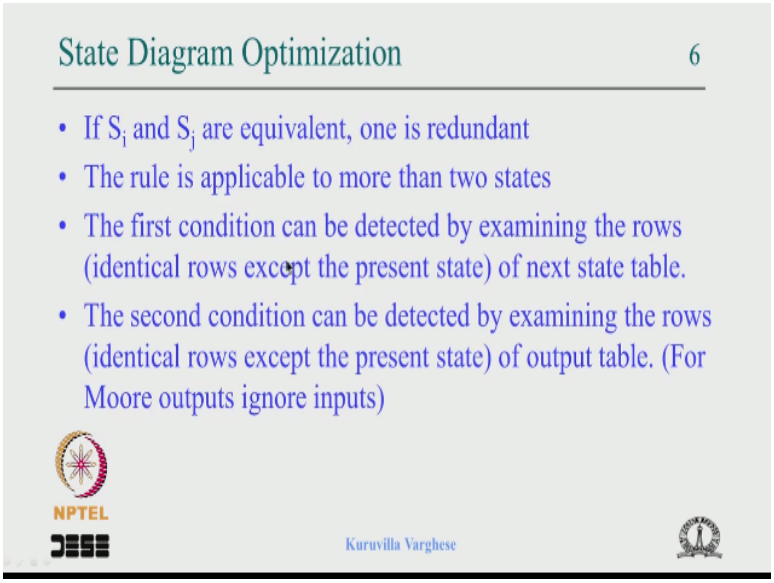
(Refer Slide Time: 01:52)



And there are multiple equivalents state you can retain 1 and remove all the other, because others are redundant. And the equivalent states produces a same output under the same input condition for mealy kind of output for Moore it is only just state matters. And it also transit the equivalent state transit to the same next state for the same condition.

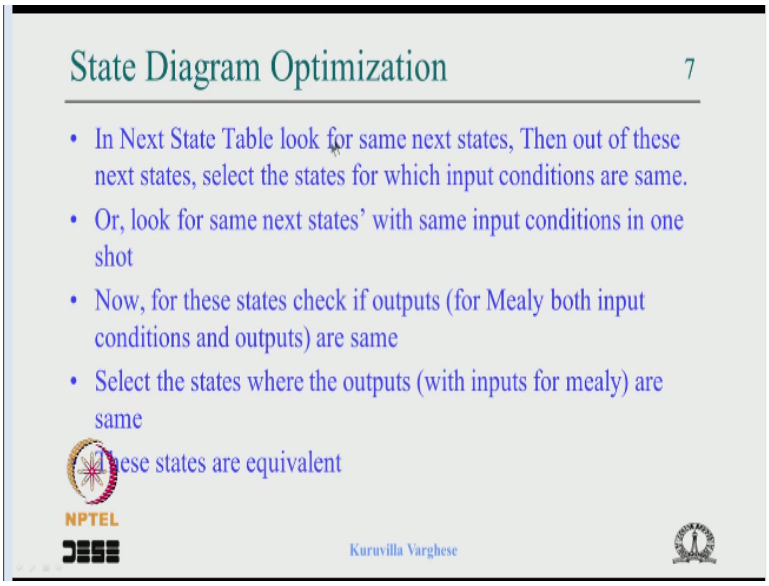
That means number of transition and number of conditions on those transition should match. Then they are equivalent and you can retain one that was a idea and we have discussed how to detect this in a next state table.

(Refer Slide Time: 02:39)



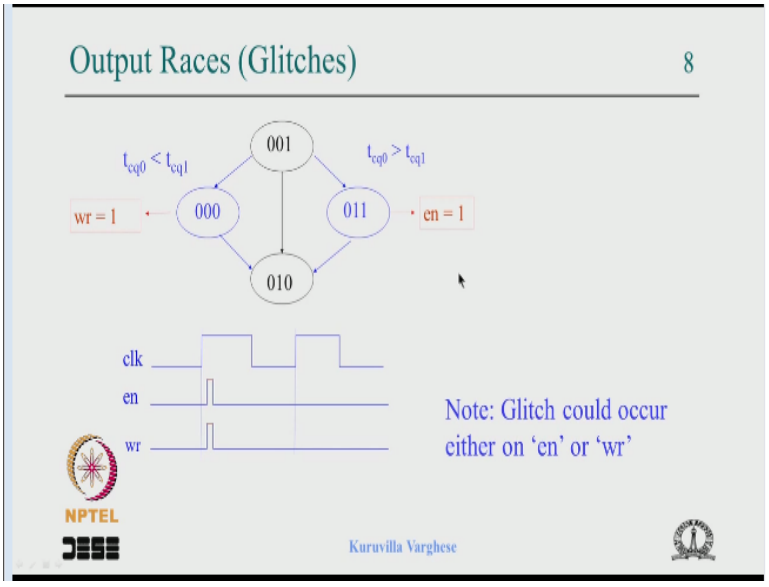
So, if you look at the next state table. So, you detect the rows which are identical except the present state. Those are candidates for kind of equivalent state. Then with this you can go to the next state table, these states next state table and find kind of the rows with identical rows except the rows present state. Then an intersection of those states should give you the equivalent state and 1 can be retained.

(Refer Slide Time: 03:16)



And we have discussed some kind of decent algorithm. So, look for the same next state and find them out then look at the input condition that will give a subset of that then you go to the next state table look for all these kind of states whether the outputs are same and you choose thus those states. In the case of mealy you have to check the input condition also. Then you are with the equivalent states then you can retain 1

(Refer Slide Time: 03:51)



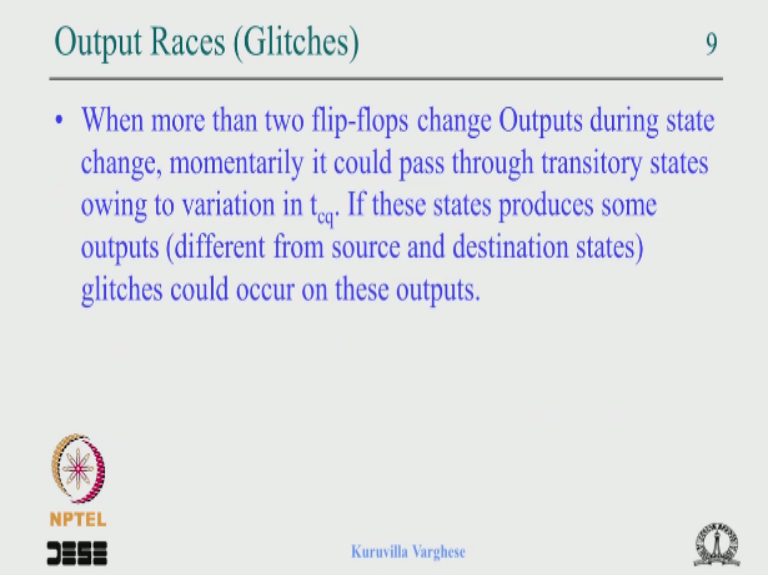
We have looked at this scenario when there are multiple flip-flop change the state and if there is variation in propagation delay, there could be transitory states and if that produces a output. And there could be glitches in the output for a brief duration at the beginning during transition and

that can affect if it is used depends how these control signals are used. This is used in a synchronous way to enable some mugs and all that no harm.

Because clocking will happen on the next state. But if you use it asynchronously to clock something to basically to clock something then you are in trouble because say this is a right signal to a memory then the location can get corrupted depending on the pulse width okay. even if the data bus is tri stated some whatever the values, whatever those floating values will get kind of return into the it is a location.

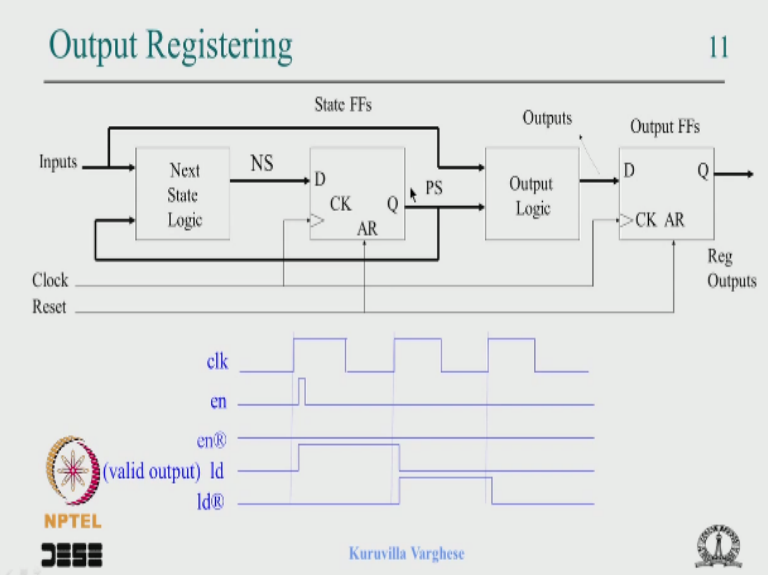
So, 1 into b a careful and to avoid it if we can avoid somehow the multiple bits changing this scenario can be avoided so, that means you have to go for gray coding.

(Refer Slide Time: 05:17)



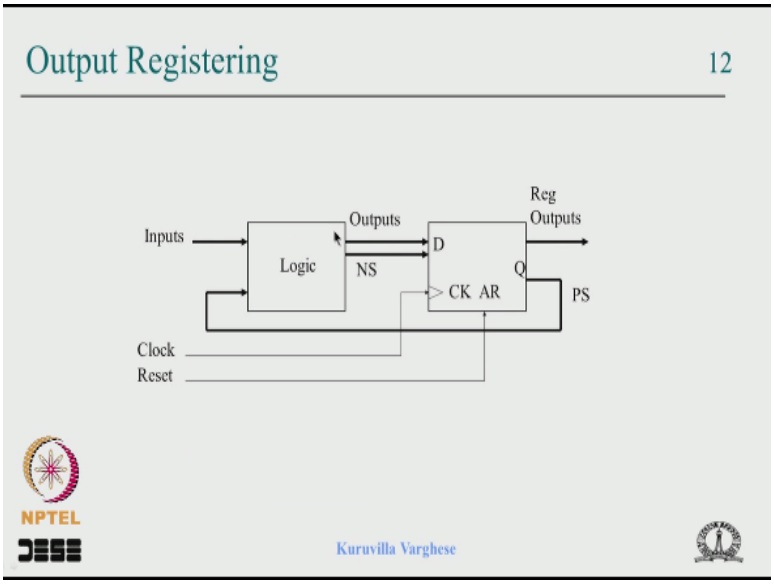
But that is not possible in all cases then we go for a output registering.

(Refer Slide Time: 05:20)



The idea is put a register at the output of the logic output logic so, these transitory kind of glitches. I will not be cot in the next clock edge, but the valid outputs will come will be registered here everything appears 1 clock period late. But that is okay so, that is how the output is registering is you know is done. And we have kind of simplified because this logic can be club here, then this flip-flop can be club there.

(Refer Slide Time: 05:55)

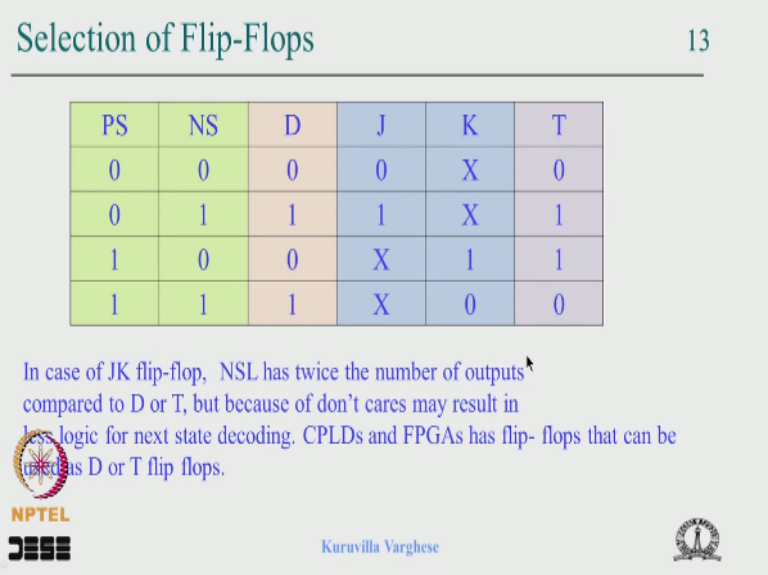


And you get a picture like that which is this comprises of next state logic and output logic. This comprises of output registers and state flip-flops so, you can see that out of this logic the next state come the output comes. And here the state flip-flop will give the present state and the

output flip-flop will give the registered output. The advantage of this view is that we know that a single process can do you know in a single process.

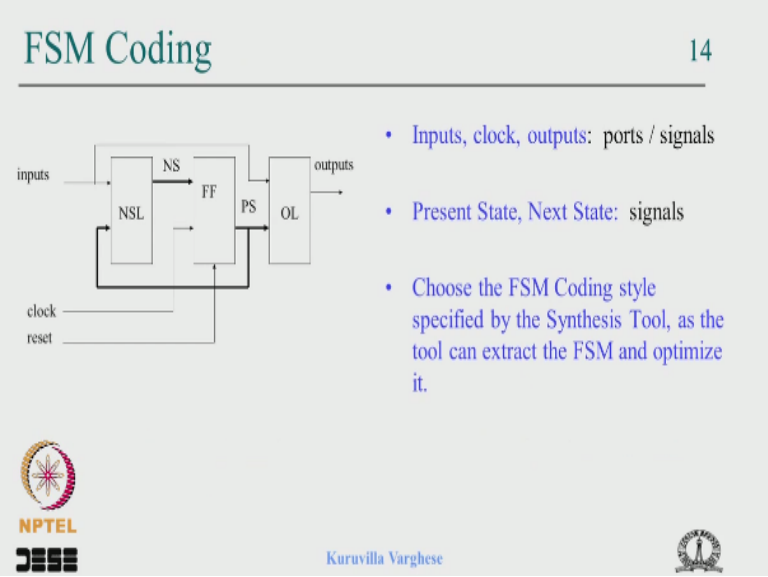
We can code a register with the proceeding logic so, this can be very easily you know coded using a single process in VHDL that is 1 advantage of that.

(Refer Slide Time: 06:38)



And we have the last thing we have looked at was the selection of flip-flop if you use D or T OR JK what ere the transition table. And we said JK you have twice in a number of output coming of the next state logic. But then because of the do not care maybe the area can be lesser than D and T and the FPGAs and CPLDs have many a times this choice of between the D and T flip-flops same flip-flop can be used as D or T .So, that was a last lecture.

(Refer Slide Time: 07:17)



So, let us look at today's part we are looking at how to do the VHDL coding of a finite state machine so, this is block diagram of finite state machine. You have a next state logic looking at the input and the present state decoding the next state which flip-flops will upon the clock will transfer the next state to the present state output logic will decode the present state for mealy output along with input to produce the output.

Now you know that this is flip-flop you know how to setup flip-flops, you know how to do the VHDL coding of that. It is very simple and this next state logic is a combination circuit, and we discuss how to write using when else or with select or case when if then all that. So, there is nothing great about it but also you know this is also logic so, you can use any constructs in principle there is nothing new about it.

But there are since there are various like we will see how many ways different ways you can attack this problem that should be kept in mind. So, naturally so, if you take this inputs there could be multiple external inputs then the clock reset all that will be ports or signal while coding. Because in the state machine is controlling the data path so, many a times the clock and reset can be external all the inputs sometime can need not be ports it can be signals.

Because you we have a data paths along with the state machine so, many a times some out of the data path. When you code it together, this can be signals output can be signal because which in

goes to the data path to control something. But at least for our kind of classroom study sometime, we make some FSM simple FSM with some assumption of on input and output in that case will be ports.

But definitely the present state and the next state are internal and so, they will be signal okay, now at the beginning itself I will say this there are different ways of coding it and we are going to soon what all possible ways you can code the FSM okay say 1 thing I can say, you can write a process for this, a process for this, a process for this it is okay. I mean or you can write concurrent statement for this output logic.

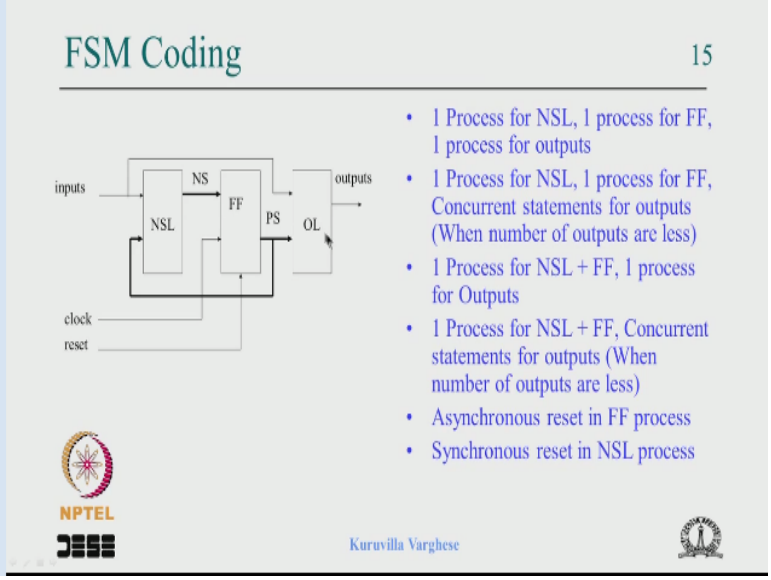
But in the important thing is that you know that if the tool the synthesis tool has to detect that this is an FSM finite state machine. So, if you do the coding say this is a combinational circuit, this as flip-flop and this as a combinational circuit the tool may not know that all the 3 comprises a state machine okay. So, that will be like you will get the desired result, because you are coding it correctly.

But all together put it together the tool should know it is a state machine, then only it can do the state machine specific optimisation like we talked about the state diagram optimisation. Unless that is known the tool may not be able do the various kind of optimisation attack various issues and so on. So, it is very important that when you work with a tool, refer to the tool manual.

And find which is a coding style which is specified by synthesis tool and stick with it. And do not try to code it in any other way. So, that is very important to kind of understand. And now a days many a times people read less and less manuals. But even sometime it is trivial I suggest that when you use a tool however the simple the tool is even if you buy a new cell phone.

It is worthwhile to read the manual spending sometime so, that lot of time can be saved. Otherwise the initial things will be fast then you get stuck you can waste lot of time, so that is my advice.

(Refer Slide Time: 12:01)



So, let us go into this coding part so, let us see how many different ways you can code it. So, as I said you could write a process for next state logic 1 process for flip-flop, 1 process for output logic okay. We will see that how the process can be written but at least no we are kind of enumerating all possible ways of you know writing this a FSM in VHDL okay. Now sometime what happens is that the number of outputs are less okay.

So, number of states are more may be say you have 8 states and only 2 outputs okay. Now if you write a process for 8 states and 2 outputs, what we have to do is that when you write a process you can use only case or if, and you will say case present state is when s0. You will say what are the output there are only 2 outputs. Then you say when s1, when s2 and so on okay up to s7. But this can be avoided if the outputs are coded in concurrent statement.

Then we can say suppose o1 and o2 are the outputs then you can say with present state select okay o1 gets 1, when you know s0 or s5 or s6 something like that else 0 okay. So, in 1 statement 2 statement, 2 concurrent statement the output can be returned so, if you see that the number of outputs are much less compare to the number of states. Then it is better to write the concurrent statement this so output logic.

So, that is another way of you know looking at it there is a third way of that you know that a we know that the registers with the proceeding logic can be written in a single process okay. So, we

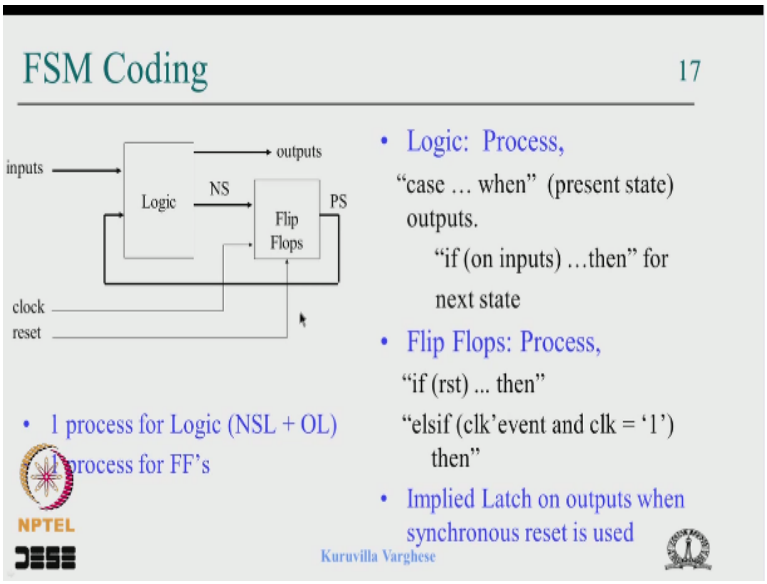
can do that we can write 1 process for this flip-flop and the next state logic. And you can write a 1 process for output and that always again bring this issue up, if the number of outputs are less same 1 process for next state logic and flip-flop.

But you can write concurrent statement for output okay so, these are the kind of 4 different ways you can code by looking at this picture. You know we can kind of look at different scenario with these logic combine okay we will come to that and we have to also look at the reset. The reset can be asynchronous or synchronous if the flip-flop as asynchronous reset then we will use that. So, asynchronous reset in the flip-flop process.

So, if you a if we have a we are using asynchronous reset where ever the flip-flop is there we will write include the asynchronous reset if or you can include a synchronous reset in along with the next state logic wit h that we have discuss earlier how to introduce the reset and we just talked about the circuit. That so, naturally we will have to say that has a priority we have to say we if reset is 1, then the next state is starting state and so on okay.

So, these are the possible you know different possible ways of doing the VHDL code now we have seen a 2 block view of the FSM where in we said anyway this output logic and the next state logic receive the same kind of inputs that is the present state and the input. So, why not combine this and show the output as if from here and call this logic which comprises of next state logic and output logic.

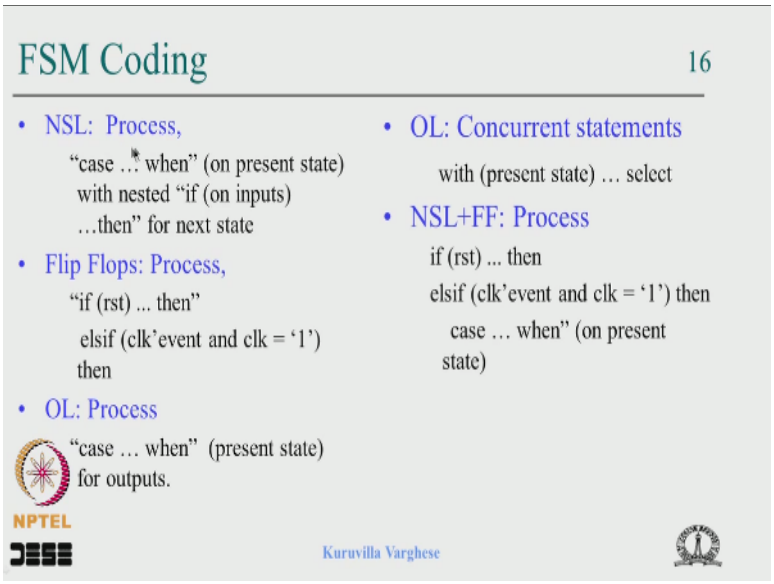
(Refer Slide Time: 16:19)



So, let us kind of look at that so, if you do that then we have a logic section which is comprising of next state logic and flip-flops. So, you can write 1 process for this next state logic and output logic and 1 process for flip-flop okay. And here you could even separate the output logic with the concurrent as concurrent statement you can do that. But the fact that we are looking at such a block diagram which shows that may be this is the appropriate thing with respect to this diagram okay.

So, these are the kind of 2 scenarios we have looked at as for as the coding is concern so, let us look at these kinds of scenario. So, where the next state logic is process flip-flop is process, output logic is process or concurrent statement or these 2 together in a process.

(Refer Slide Time: 17:18)



So, we will see how that can be coded suppose you want to write next state logic as a process then we know that we have 2 kinds of input, 1 is the present state okay which is multiple bits okay. And number of inputs lot of inputs okay so, we write a process within the sensitivity less just the present state and various inputs may be i1 comma i2 comma like that so, that is the process. And now we have 2 inputs to handle 1 is the present state and next is a input.

And when you look at the state diagram for a particular state all the inputs are not valid you know the like you are in a particular state. Then you say if a particular input is 1 go to the next state and so on. So, and this has to be happen for all the states so, very normal in duty of things to do as for as the coding is concern is a case present state is and you say when s0 okay. Now for that state s0 you say if start is low.

Then remind there or if start is 1 else if start is 1 then go to the next state. So, this is 1 because you have to say for a all the states so, case is a better thing but for a particular state only certain inputs are valid. So, we use if a nested if under the case for each choices we use a if for to specify the transition okay. Now when it comes to this particular bock very simple we say of reset is 1, present state is the starting state may be s0, else if clock event clock is equal to 1.

We say present sate get the next state that is all okay so, that is what is shown here if reset then starting state else if clock event clock is equal to 1 the present state get next state. And output

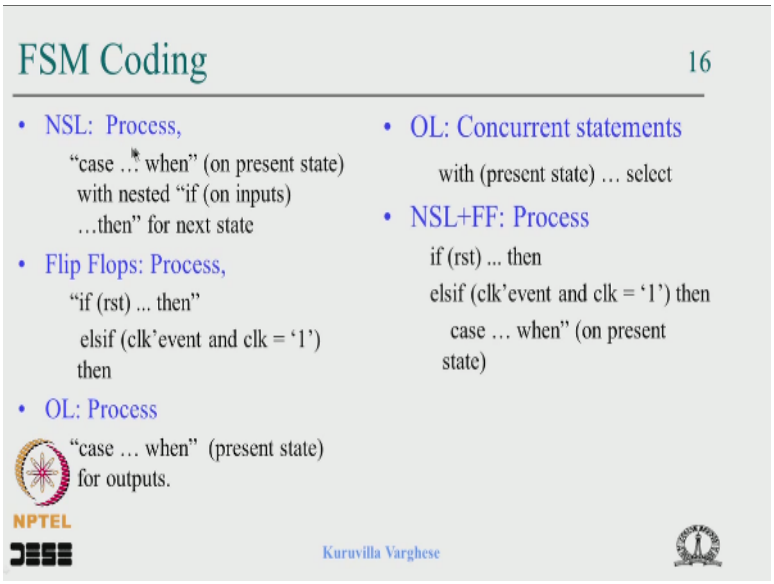
logic if you are using only the Moore kind of output then we can say case present state is when s0. You say suppose there are 3 outputs you say what are the output values, when s1 what are the output values and so on okay. So, that is what is shown here.

And if the output logic is in concurrent statement we say because that depends on the present state. We can say with present state you say when s0 and what are the outputs, and if it is say mealy kind of output you can say if under that particular state you can say if kind of input is 1. Then the output is 1 and so on and then we can combine these 2 together that can be coded like if reset is then you know what is the beginning.

Then you say else if clock event clock is equal to 1, because we are putting these 2 together and under thus because this will now come at the d input of the flip-flop. So, this logic will come within this clock event clock is equal to 1, we have discuss that and you say case present state is when then all the if you have to write. So, this is little bit kind of complex coding but it is very convenient because looking at the state diagram.

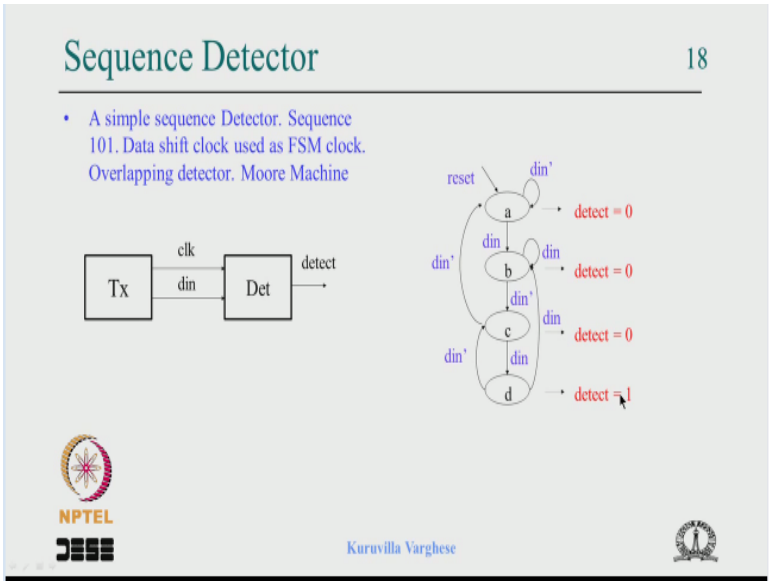
We are able to write in 1 short otherwise you have to look at the state diagram to write this process. Again go back from the beginning you have to look at the state diagram to write this. But if you have combining it from top to bottom you can write it together**.** So, the it is less error prone and rechecking or debugging is very easy if you put it together okay.

(Refer Slide Time: 21:21)



So, now this is scenario where we are combining both together. So, we said it is case when on present state you say case, present state is when s0, when s1 for each of the state you say if on a particular input then you transit to next state. And the flip-flop is simple, we have discussed that and when synchronous reset is use there is a chance of implied latch. I will maybe I will not discuss this now when we go to the coding we will look at it.

(Refer Slide Time: 21:59)



Now what we are going to do is that we are going to take an example, very simple example maybe when we discuss the case study. We will see a reasonably complex state machine to control the data path. Now the intent is to you know show the styles of coding for that I have to kind of use less space. So, I am kind of bringing a very simple example as a state machine.

Basically to illustrate the coding all I do not want to show only part of the code. I want to show everything, so I have chosen a very simple example it is a sequence detector**.** Basically in communication when a transmitter is transmitting something to the receiver it can be synchronous or asynchronous many times the clock is send along with the data in communication.

And which some encoding you know there are different type of the clock encoding can be use maybe you have heard about Manchester encoding differential Manchester encoding and all that okay so, that allows to embed the clock in the data. It is retrieve and synchronise with the PLL so, we are kind of and the receiver should know when the transmission start.

So, many a times are the beginning of the transmission or a beginning of a frame depending on whether it is synchronous transmission or asynchronous transmission. There will be a pattern and which is a quite a long pattern okay, it is not a very simple pattern and the receiver is looking continuously every bit. It has to look for that pattern okay, when the pattern comes it is kind of you know sure that transmission is starting.

And the synchronisation happens like that so, we are taking a very simple example transmitter and we are assuming the clock is available that is the scenario. Though it is a separate clock line is not sent from the transmitter to receiver so, we are kind of coming out with the state machine to detect a pattern 101 send by the transmitter. That means this detector is continuously looking at this data in din and that is synchronise with the clock.

So, every positive edge the data is send and this is continuously looking at it and whenever it receives a 101 it make the detect high okay. Now we are not you know designing the receiver, we are just detecting designing the detector sequence detector. And we are trying to implement the detector as a finite state machine which need not be the case in a real sequence detector which is used in communication.

So, mind you but then it is a nice exercise so, to work with it basically sequence detection so, it gives some mental exercise as well as it clarify our idea of the state machine the coding and all that. So, that is the basic idea but mind you this is in over lapping detector okay that means suppose transmitter is sending something like say 101. Then like if it is in known over lapping detector say 101 it is detection.

Then say it is an 00 not detected again 1011 is sent again it is detected suppose the transmitter is sending 101 again 101 it is a known over lapping detector. That means every 3 bit boundary this pattern is searched okay. So, but when we say over lapping detector it means that a the receiver looks for the particular sequence every bit boundary not the boundary is not limited by the length of the sequence every bit sequence is checked.

So, it may happen that the transmitter is sending a pattern 10101 5 bits so, the first bit first 3 bits 101 detector will detect as a the particular sequence. Then from the third bit, third bit is 10101 so, the third bit is 1, fourth bit is 0, fifth bit is 1 again that over lapping the 1 to 3 is a sequence 3, 4, 5 is also the 101 sequence that is detected. So, a you know I mean this is a exercise may be a mathematical exercise than a circuit is design.

But then it is good to discuss that so, let us look at how this cn be done so, let us turned to the block diagram, the block diagram of the state machine is simple a clock input, a data input and a 1 output which is detect. Whenever it receive 101 it is this goes high and as I said if there is 0 1 here, then this third bit fourth bit fifth bit will constitute a sequence again that will also be detected. We have to design that in a such a way.

And very important thing to remember that the clock and the data is synchronise so, we cannot afford to miss any clock okay. So, in the FSM we have discussed we have assumed that the clock is bit kind of flexible. And the clock can be very high frequency and we have you know we can wait for some input to in a state for something to happen and all that here we cannot miss any clock you know we cannot eternally wait every state a data comes.

That should be kept in mind so, let us try to detect this 101 so, let us draw a state diagram for it. So, basically we have to look for say at the beginning if we come to a state okay like at the reset we come to state a let us call it is a. And the output has to be because we have 1 output definitely it is 0, the output is 0, detect is 0 okay. Now we are looking fo a101 so, if the input is 0 we are nothing to do it remain there.

You know because we are come to starting state the sequence of not started it is 0 coming so, remain there. So, if it is din is bar dash or din is 0, then remain there detect is there, but if din is 1 we are received the first bit okay 1, and we come to sate b we are received the first bit, detect is 0, but here the game start, if it is 0 we got the next bit. Because here we are received the first bit if din is 0, din dash, we can go to the next state.

But if din is 1 you do not have to go back because we have already got 1 and if further once are coming it is a starting bit of sequence and we can remind here you know. As long as 1 is coming remain there. So, that is what is shown here. Because anytime the sequence can start, so if further ones are coming remain there, detect is 0. If it is din is 0, din dash go to next state. That means that we got 1 and 0.

Now if the 1 comes, we go to state d and we can say detect is 1 okay. and but if it is 0 din at this state we have got 1 and 0. If 0 comes everything is destroyed we have to go back okay. so, that is the detect is 0, so if the 0 comes you go back reset nothing can be done look for new sequence. But if it is 1 then we have got this sequence okay 101 and the detect is 1 okay. we say detect is 1, now the game start okay.

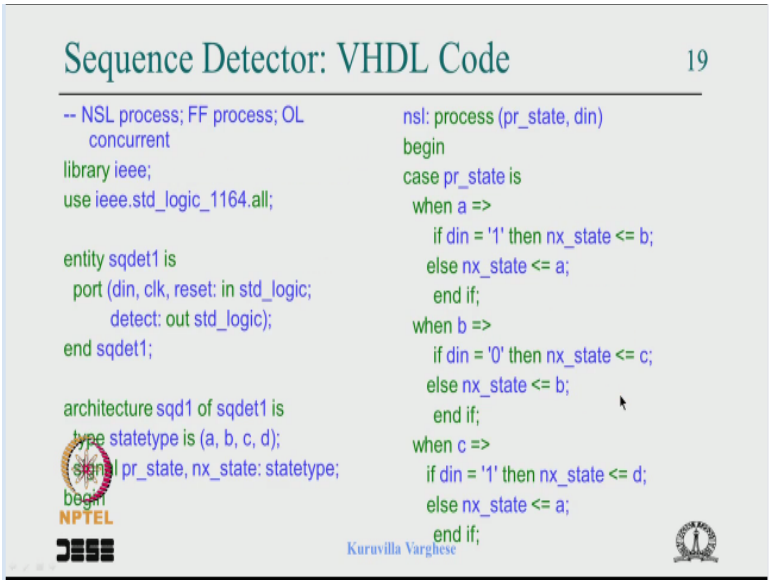
If it is 0 no lock you go back okay sorry**.** If it is 0 mind you we have got 1, 0, 1 okay. Now 0 you do not need to go back, you should not go back, because this 1 could be start of a new sequence. So, if it is 1, 0 you go back, so you have you are in the new sequence 10 and 1 comes you come back here. So, if it is 0 you go back here if it is still 1 it could be start of an new sequence.

So, you go back here okay, so that is it. So, if din is 0 this is a new sequence and 10, if it is 0 you go back, if it is 1 you get it back again. If it is 1 still it could be the starting state. So, that is state

diagram. Once again because we have to write the code, so remember this at the beginning come to state a detect is 0, 0 remain there, 1 go to next state, 1 remain there detect is 0, 0 you come here.

Because you have got 1 and 0, and detect is 0, if it is 0 go back, if it is 1 come to d, detect is 1, if it is 1 go back to the state b. Because it is a new sequence could be starting. If it is 0 this is the second bit of probable second bit of a new sequence. So, 10 and that is done, so we have 4 states. So, a, b, c, d, 1 input din and 1 output okay, so let us look at VHDL coding of this and let us look at this is a Moore kind of output okay.

(Refer Slide Time: 33:03)



So, let us look at this we will write like this next state logic as a process, flip-flop as a process output logic, we will write concurrent. Because there is only 1 output, there is no point in writing a process, because there 4 states we have to say K is present state is when a, when b and all that. That can avoid if you write as a concurrent statement. So, let us look at it this is the library close ieee library the package.

This is the entity as I said since we are writing this kind of standalone all our input din, clock, reset or input standard logic detect is output standard logic and the entity name is sqdet 1 and we say architecture some of that. Now comes thing we have to kind of remember with most tool support. See we have 4 states with binary encoding of we will end up with 2 flip-flops okay.

So, the present state and the next state are 2 bits. Because there are 2 flip-flops, so this we have to define a standard logic down to 0 okay both of this. But this is kind of combusum what happens is that if we come and make some changes. Suppose we find that this state diagram you have drawn some problem. And we have inserted 1 state in between, then it becomes 5 state.

And whatever was 2 bits has become 3 bits. And you have to do lot of changes it is a big problem. So, most tools allow you to define the present state and next state as an enumerated type okay**.** So, that the you can easily change state diagram, so how that is done is that you define a new type okay you say type state type and this is a name you know it can be anything, it need to be call state type.

Some type you know some name is then you say the state names a, b, c, d, so these are the 4 values this state type can take and we know that internally take 0, 1, 2, 3 and there are 4 states. So, it will be encoded in 2 bits and internally it maybe assigned 00, 01, 10 and 11. Then we define the present state and the next state by this kind of enumerated data type. We say signal present state and next state is state type okay.

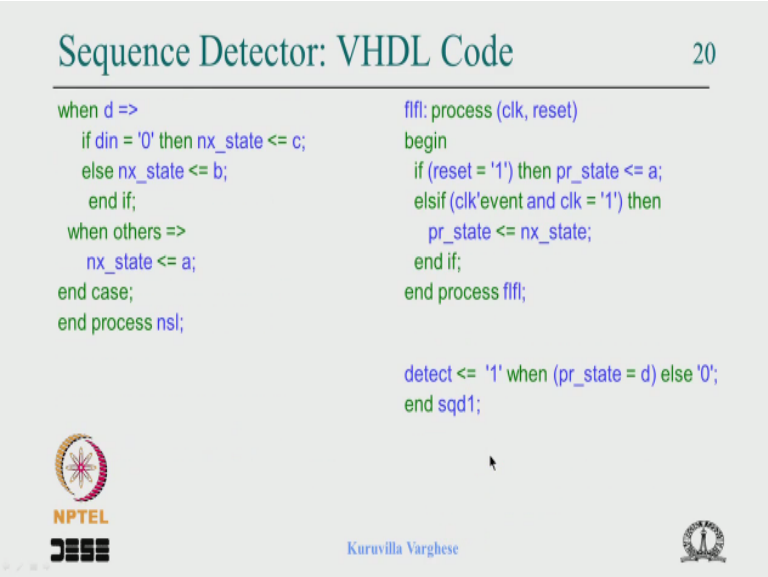
This is 1 advantage the tools give it is very convenient and internally it is converted into the standard logic vector. And so, this is but you please check with the tool you are using whether it support that. Now let us write the next state logic. So, let us write the next state logic looking at the state diagram. Here next state logic a label, process the input to the next state logic is present state we call it is you see here the PR state, the pr state, all the inputs you know.

We have written all the inputs, but we have only one input. So, we write that din you say begin okay. Now for various state we have to write the code, so we say K is present state is okay. Now we have 4 states a, b, c, d, so we say when a a in the first state, now we specify the transitions okay say if din is 1, then we got 1 is next state is b, else next state is a, that means if din is 0 remind there you know that is what is shown here, din 0 remain there, din is 1 go to b okay.

And mind you do not ever write if din is 1, next state is b, end if okay. If you write like that there will be a implied latch you know people will think that anyway you are in a. So, you just say if din is 1, next state is b and end if you say the when a is there, there will be a feedback do not ever do that just state the complete if else you know you have to complete it always there will be implied latch oaky.

And when b in b we say as din is 0, then go to next state. Otherwise stay there, so if din 0 and next state is c, else remain there, when c we know that if 101, if din is 1, then next state is d, else next state is a going back all the way. If it is 1 go to d, else go back there. And in the d if it is 1 go back to b, else go back to c.

(Refer Slide Time: 38:30)



So, that is what is shown here, if din is 0 then next state is c 10, else next state is b okay, end if. When others, because we have to enumerate all the condition as for as simulator is concern next state is a okay, it does not matter we are only kind of taking possibility for simulator. So, end case, end process, so that is how we code the next state logic in a process using K is and nested if.

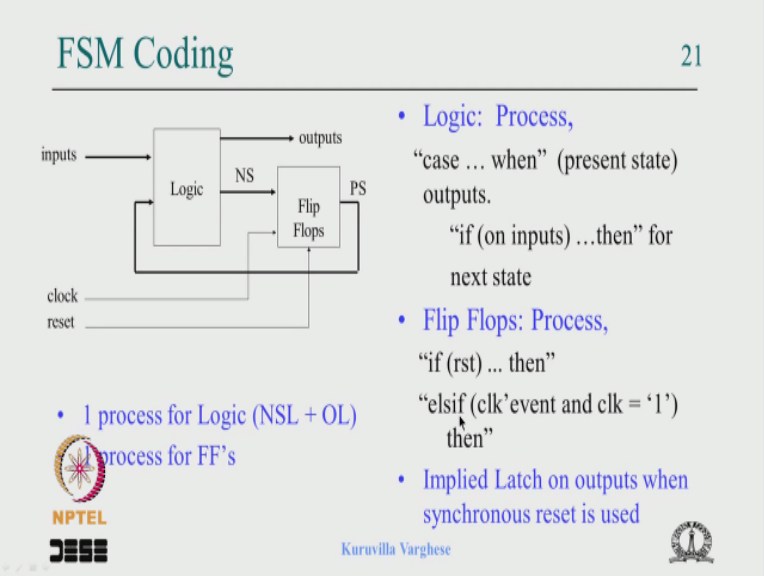
And as for as the flip-flop is concern it is simple flip-flop the process, clock and reset in the sensitivity less begin, Asynchronous reset if reset is 1, then the present state is a, because we

have to make the present state the starting state you know. So, upon the reset this will become a, then everything happens properly and it will transit. So, at power on it comes to a.

So, that is what is written here like if reset is 1, then present state is a, else if clock event clock is equal to 1, then present state get next state, end if. Because there is a memory and end process that is it okay and the concurrent statement you know that detect is 1, when the present state is d. So, that is what is written as a concurrent statement, detect it gets 1, when present state equal to d, else 0 and end sequence detect 1.

So, this is completed we have the entity, architecture. We have written a process for the next state logic with case is on present state for each state we write the transition using the if for state transition and we write flip-flop in a process with asynchronous reset and the output in a using a concurrent statement being Moore, it is very easy okay, so, that is it.

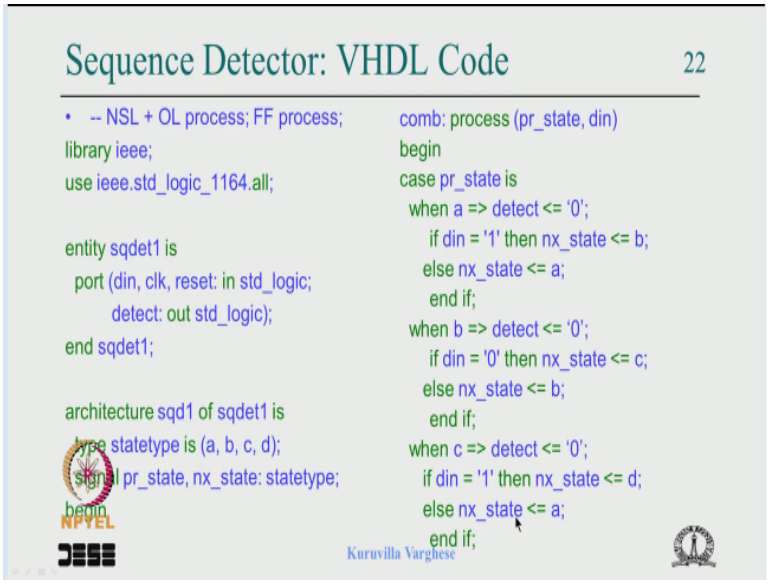
(Refer Slide Time: 40:37)



And now we look at this kind of scenario, where the next state logic and output logic are combine together, so it is simple like we say here K is present state is and you say when a at the beginning we say what is output detect is 0 without if and we say if condition and you repeated. And that is a idea, so that is what we are going to see case on present state and you state what other outputs for each choice.

And in each choice you write if on input for all the transition okay and as usual if reset is in the in the flip-flop process okay. And as we have written and when synchronous reset is use there will be an implied latch I will tell you I will show you soon how that happens okay.

(Refer Slide Time: 41:38)



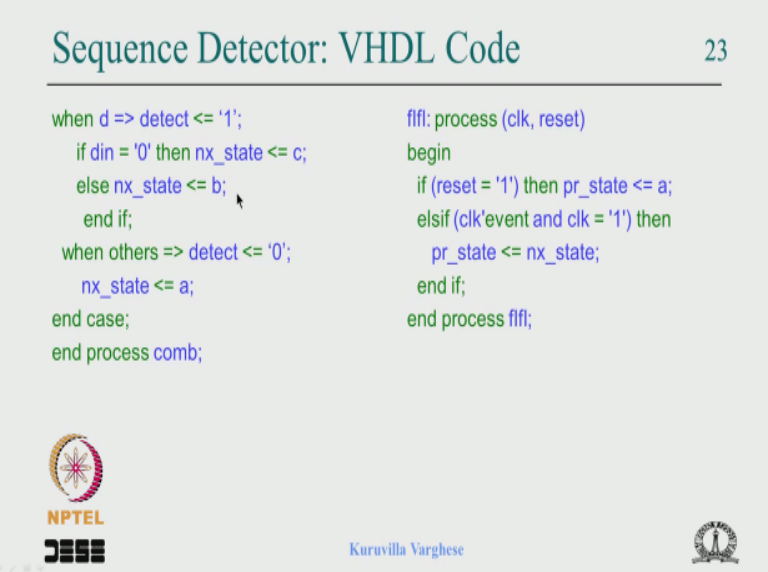
So, that is this is sequence detector VHDL code wherein the next state logic and output logic is combined in a single process and flip-flop is a process. So, as before we have the library and the package, we have the entity with the din clock reset is input and detect as output. The architecture declaration before the begin we have the enumerated data type, types, state type is a, b, c, d signal present state, next state is state type.

And these all are kind of standard, so you can cut and paste, copy paste from the previous thing. Only thing is that if you have a new state machine with more state you add that. And this is the useful template to keep and this is where we are writing a giving a label process where we are combining both the output logic and the next state logic the sensitivity less has the same as before present state and din.

The fact that the output logic is combined does not change this because the output logic was a get the same kind of inputs. So, present state and din and we say case present state is and we take the first state when a and we say detect gets 0. Then that is the we have specified the output for

particular state more output and now we specify the transition if din is 1 then next state is b, else next state is a end if and so on okay. So, we do the same for b the c and the d okay.

(Refer Slide Time: 43:26)

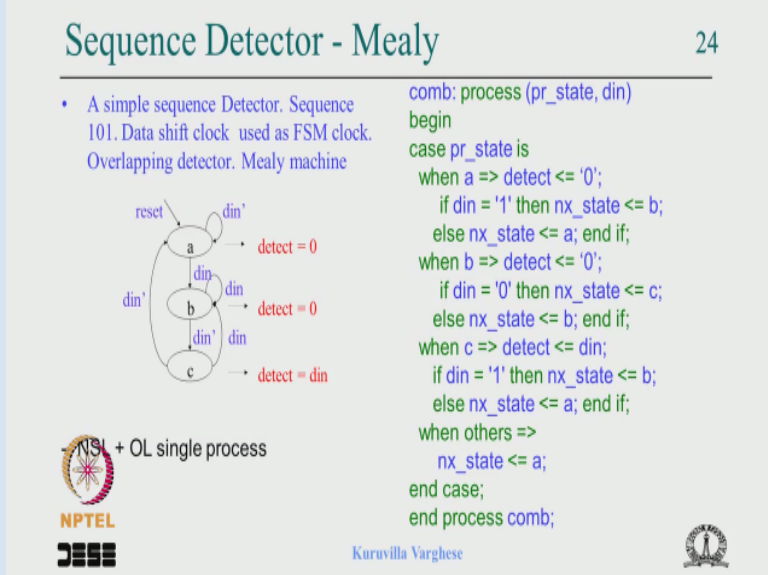


So, that is it only thing is that when you say when others you have to say detect is 0. If you do not say there is an implied latch so, where ever you say when you have to specify the output if there are multiple output in your case everything has to be told here, like if you have a detect and enable as a output everywhere it has to be specified not only the place where it changes that you should keep in mind.

Now the flip-flop as before same the clock reset begin upon the reset the present state is a, else if clock event clock is equal to 1, present state is next state. So, that is why I said this all can be written as the template you just copy paste this is same for all, you know irrespective of then the enumerated types maybe it is a state diagram with 5 state, 8 state this is not going to change it is all same.

Because we have declared this as a a state type which is enumerated so, we need not worry about the size of the number of bits which is used to encode the state that gets automatically you know covered.

(Refer Slide Time: 44:44)



And so, let us now earlier you know we have looked at a Moore machine for detecting 101. So, let us do a mealy machine just for the as I said just as the practice and you can already see what can happen in a Moore machine. We have said come to b 0 come to c, 1 come to d and say detect is 1. So, you can imagine what can happen, we say 10 now upon the 1 instead of going to d, we say in this state if the din is 1 the detect is 1 okay that is the game.

But we said know where to go like with that transition that is what we are going to see in that mealy machine so, let us look at these same thing it is a sequence detector same block diagram sequence is 101, it is a overlapping detector it is a mealy machine. So, at the power on reset you come to state a detect is a 0 as long as the din is 0 remind there. If din is 1 come to next state b 0 again we know that.

If din is 1 remind there, din is 0 come to next state c c din is 1 remind there. Because it could be a start of a new sequence and 0 transit to the next state, now here we are going to say detect is 1, if the din is 1. Because we have got 10 if detect is 1 if the din is 1 so, we can say detect is din, the detect is 0 if din is 0, but upon that 1 we have to transit. So, you got 10 and 1 back here because that 1 could be a beginning of a new sequence.

So, 101 again 01 you can go like that but if it is 0 you go back here okay so, that is what is shown here, detect is din, din happens go back here, din bar a in goes back there. So, this shows

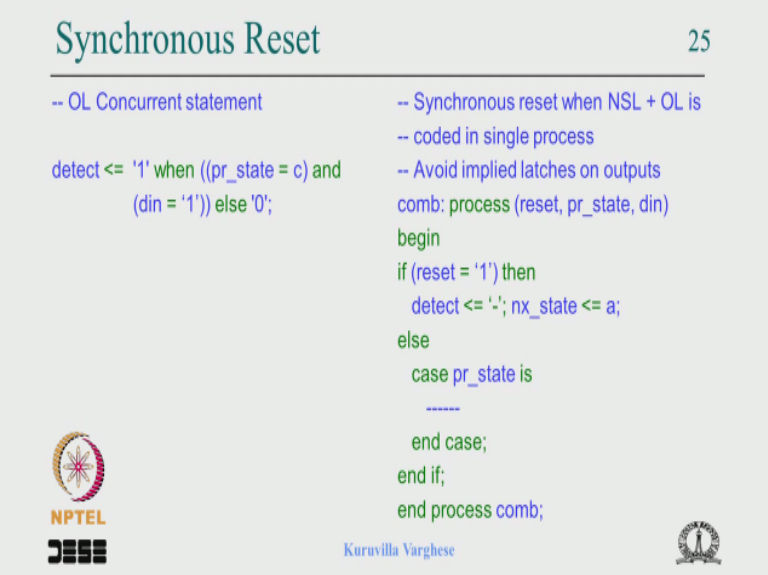
even in a like we have discuss what is the difference between Moore and mealy kind of output. You see here there were 4 stage now the 1 state is less for 1 output, and we also know that the detect comes early earlier it goes to the d state.

And the detect used to come so, whatever we have discuss before is true here also. So, let us look at this kind of coding and we are going to code the next state logic and the output logic together in 1 process so, that is in 1 process. So, we write a label process, present state and din, and when we say case present state is when a we say when a detect is 0, if din is 1 next state is b, then next state is b, else that means it is 0, and the next state is a end if.

And I said as I said do not ever say if din is 1 then next state is b end if, because implied latch is there come to b we say detect is 0 because detect is 0 if din is 0 go to c, if din is 0 next state is c, else next state is b you know remain there din. Now like when c we say detect is din, detect is 1 if din is 1, detect is 0 when din is 0. And now the transition is if din is 1 go to b, din is 1 then next state is b, else next state is a end if.

When others we say next state is a sorry I have to say some value for detect otherwise it will be implied latch remember that I have to specify detect some value it does not matter. But I definitely I have to say detect is 0 or something like that so, otherwise there will be implied latch I have say that so, that is the mealy machine with the output logic.

(Refer Slide Time: 49:12)



And next state logic combine, but if you are writing the separate then you can say like this detect is 1 when the present state is c, and din is 1 else 0. So, that is what we want here say we say detect is 1 if the present state is c and the din is 1, that is what is written here okay. Now assume that we do not have like in the previous diagram the asynchronous reset. We are combining the synchronous reset okay.

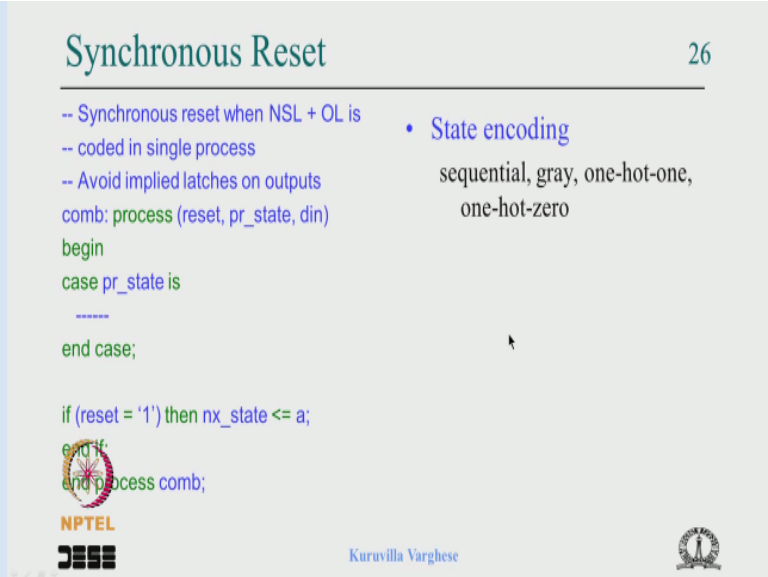
Now when you code that it looks very simple for coding we say if reset is 1, next state is the starting state, else if clock event clock else you can say case present state is okay. But there is a danger in there that is what I am going to show okay so, here what can happen is that you say a **a** process where the present state and in is a input and now a new input comes reset and you say if reset is 1, then assume that this detect is not there.

And you say next state is a okay, else you say case present state id that we know that in that case you are going to specify the next state as well as detect okay. So, in here for each state we are going to specify the next state and detect and if you do not specify the detect here, there will be a implied latch on detect. So, we say detect is do not care okay you can say anything it does not matter, detect is 0 really it does not matter.

But then to avoid that feedback that latch we say detect is 1 okay. So, this you have to be careful when you write 2 things when you combine the next state logic and output logic. And you

include the reset in this particulars use synchronous reset you can end up in this state and write all the outputs here. If there are 4 outputs, you are going to specify all that here. In this reset you write for output and say do not care okay, that is that you have to remember.

(Refer Slide Time: 51:31)



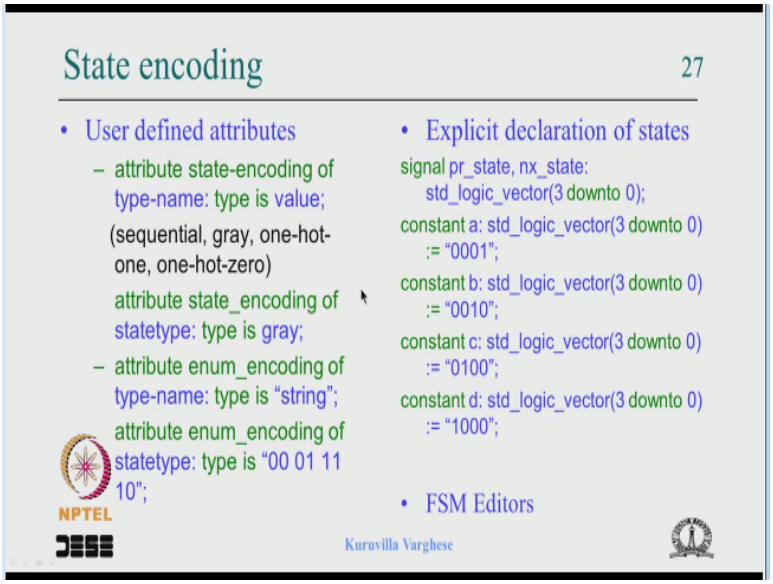
Otherwise you know there is another way of doing it you just say the reset separate, you do not say you write the case for the next state logic and output logic do not mix with a reset you say if reset is 1, then next state is a just say end if not a very good I am not very kind of comfortable with this style you know that this is much better than this one. But I suggest use a previous one.

Now when you write a code like that, we have no control many a times what is the state encoding okay like many a times a tool will assign 00, 01 and 10 to it though we have not discuss this state assignment what are the advantages of state assignment which we are going to see in the next lecture . We may have to give a different assignment, so that can be kind of specify by some state encoding attributes.

So, you can have a sequential coding, we have seen the gray know, if you do sequential coding you might end up with the output racing. So, you can use a gray coding and that is something called one-hot-one where one state is encoded in a 1 flip-flop we will see that where it is useful. But one-hot-zero is like one-hot-one, but you start with all 0s okay. In a one-hot-one if there are 4 state, then there are 4 flip-flops.

So, you have 001, 00, 10, 0100, 100 but in a one-hot-zero you start with all 0s, so that is a difference.

(Refer Slide Time: 53:15)



But that can be manipulated by state encoding using some attribute in VHDL coding, so this is the kind of syntax for it, you say attribute state encoding of whatever type name you have use. See here we have use the state type you know when we said at the beginning we said the state type oaky. So, we have to say here attributes state encoding of state type, then type is you can say sequential, gray, one-hot-one, one-hot-zero and things like that.

So, I am showing 1 example where attributes state encoding of state type, type is gray. There is another way you can say attribute, e num, encoding of type name state type say is type is string give the straight away. So, you can say take the assignment as 00, 01, 11, 10 okay. Here you have to choose between this 4. But here you can kind of you know tell the custom encoding okay.

Now you have to check the tool the vendor specification whether they support this, every tool vendor may not support this attribute may they may call it by different name. So, you have to you know check the documentation of the tool. Suppose that is not possible okay, like the tool does not give you a option of changing it, then you can explicitly instead of using the enumerated encoding.

You can instead of saying types, state type is a, b, c, d you can literally hot coded you say signal present state and next state is standard logic vector 3 down to 0. And you define constants you say constant a is standard logic vector 3 down to 0 equal to 001, constant b, constant c, constant d and so on, then you can use a, b, c, d only. So, even if the states are changing you have to at the **at** architecture declaration region you can come and change it in one place.

And do the change in the coding and mind you we have seen that the moment you have the state diagram the coding is exactly 1 to 1 okay. So, it is possible that you can automatically generate the code from the diagram not a graphical diagram. So, the tools let you draw state machine. So, the moment you put a bubble and give a name, they know that there is a state called second bubble and b they know that there is a b.

And if you draw an put an arrow they know that there is a transition. So, as in when you draw these are these things are tracked and ultimately a 1 to 1 code is generated. So, there are graphical FSM editors are available in the tool and that can be use. But at the beginning when you start I suggest you code it, it not a great effort you learn it well maybe then once you have practice, then you can use a FSM editor from the tool vendor.

So, I just want to kind of close that part of the VHSL here. So, that we can continue with state machine the next lecture. So, essentially we have looked at the various coding style we can adopt for doing the VHDL coding of the finite state machine we have seen all possible was okay like using process and next state logic and output logic combine, next state logic and flip-flop combine.

Output logic using concurrent statement we have taken an example a simple example of a sequence detector of an overlapping sequence detector which is detecting 101. We have seen a moore type detector mealy type you know output. We have seen the various coding style, we have seen there is an issue when you synchronous reset and combining the output and the next state logic and how to sort it out.

And we have briefly looked out how to change the state assignment using attributes. Once again it is tools specific, vendor specific you have to check the documentation. Ultimately vendors give an FSM editor which generates the code from what you draw, though it looks like a magic, it is very straight forward good computer students can take it as an kind of project and easily do that.

And so, that is what we have in the next lecture we will continue with issues of the finite state machine various other issues and this lecture will be useful there. So, I suggest you go back maybe practice some state machine maybe you can try a different sequence in the best thing is that different sequence write the code simulate and learn well. So, I wish you all the best and thank you.