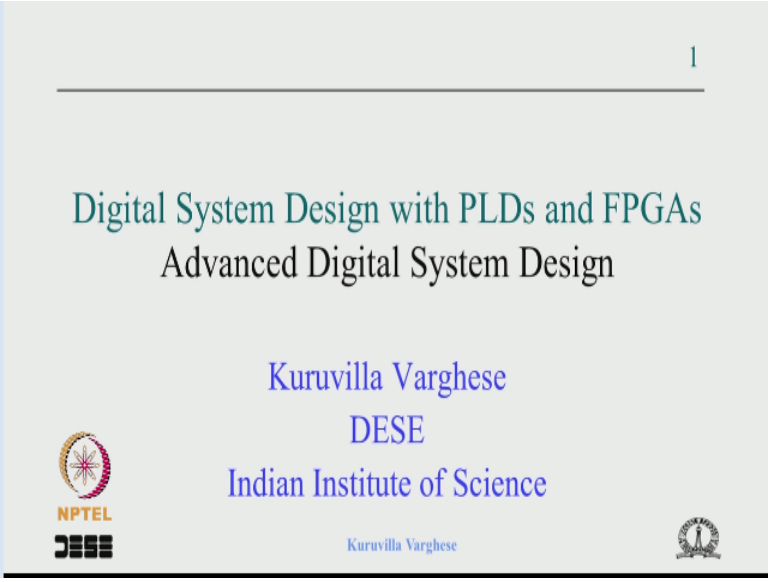
Digital Systems Design with PLDs and FPGAs
Kuruvilla Varghese
Department of Electronic Systems Engineering
Indian Institute of Science-Bangalore

Lecture-25
FSM issues 3

Welcome to this lecture on digital system design and the course digital system design with PLDs and FPGAs.

(Refer Slide Time: 00:28)

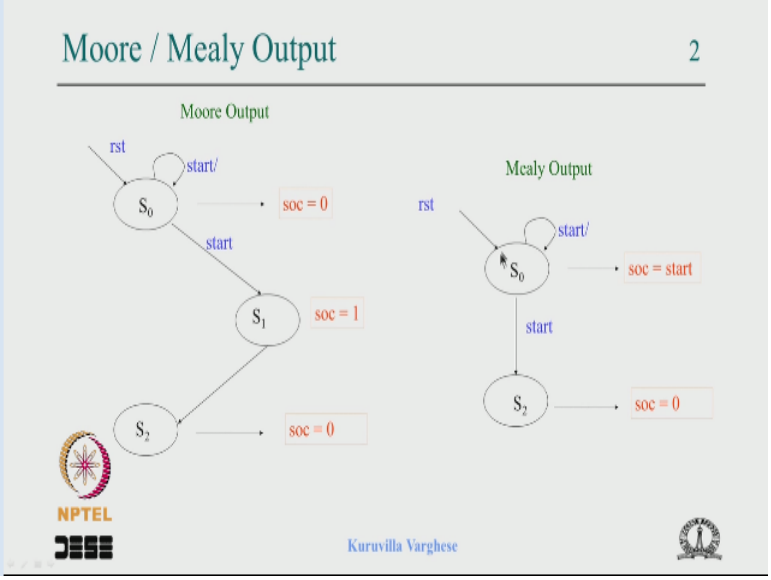


In the last lecture we have looked what are the feature of Mealy and Moore output when can be used the Mealy output and then we looked at state machine controlling the data path through clock gating and re-circulating mugs, it is advantages, disadvantages, how is the power dissipation and a solution for it and associated VHDL coding and so on**.**

Today we will address some more issues of state machine, when you design some things are related to kind of optimisation which sometime the tool does not support or tool does not support to a degree where it can be kind of used you have to check whether your tool supports. Otherwise you can try implement this manually which maybe little difficult in some cases.

But at least to be aware of these issues will be useful when you are kind of in a difficult situation. So, let us run through slides of the last lecture before we get on to today's portion. So, let us move on to the slide.

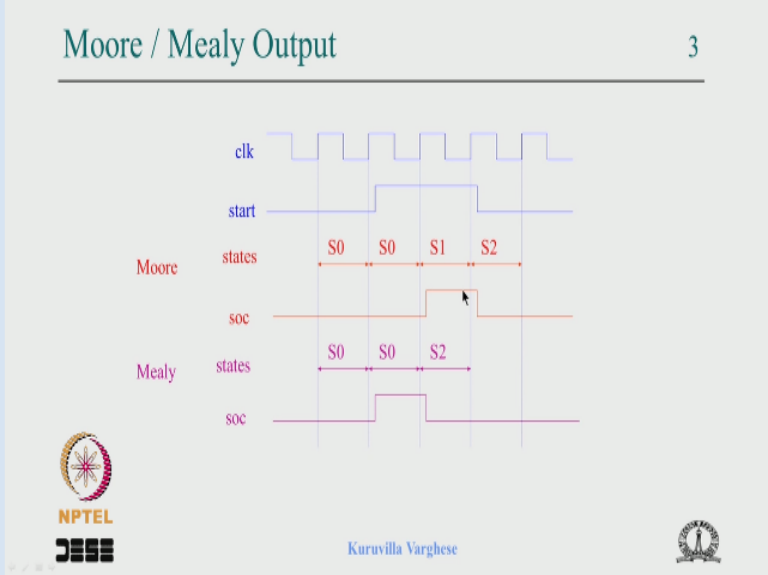
(Refer Slide Time: 01:58)



And we have said that we have taken the example of our ADC controller. Here in the Moore output when the start comes the machine transit to a particular state s1. And this start up conversion was made 1 and then it is made 0. So, it is of 1 clock duration during s1 and since this is a function of start signal, what we try to do was that we try to convert that Mealy type of output.

And you know by making the SOC as a function of start then we say that in this particular state s0 SOC is 1 in start is 1 and when the start is 1 upon the next clock edge it is transited to the next state s2. Because s1 is not required just 2 kind of make it equivalent to compare that is why I called this s2, because calling this s1 will kind of make it similar to that.

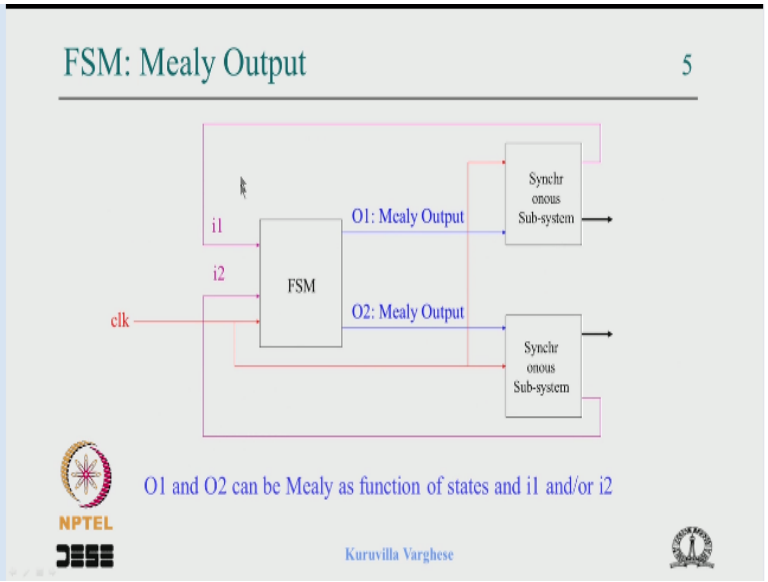
(Refer Slide Time: 03:04)



And we have looked at timing and we have form that for the Moore kind of output. The SOC comes during the state s1. But for Mealy it comes during s0 and also we find that 1 state is less. So, that we said in general, you have many outputs which are kind of Mealy outputs. Then maybe the number of states in the state diagram will be less. So, the state machine area will be less and also the output comes earlier to Moore in the case of Mealy.

One other thing we have found the problem was that if this particular input is not synchronous. Then it can change anytime and if it changes very close to the clock edge, you get a glitch at the output and that can create problem. And but that happens only when input is not synchronous.

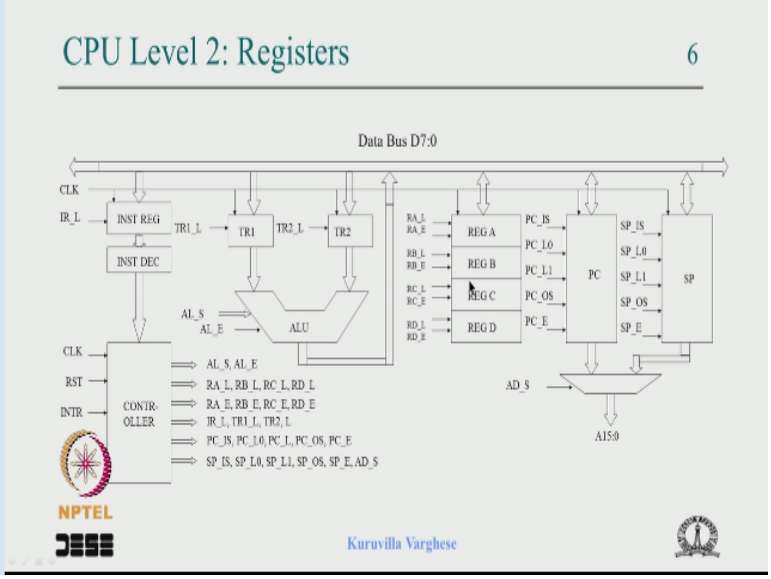
(Refer Slide Time: 04:12)



And when the input is synchronous we could happily use the suppose in this case I have we have showing the FSM 2 synchronous subsystem as I said it could be registers, counters and all working in same clock. Then some output is coming and we can now generate this particular output of the FSM as function of the synchronous input. So, wherever you have synchronous input you can generate.

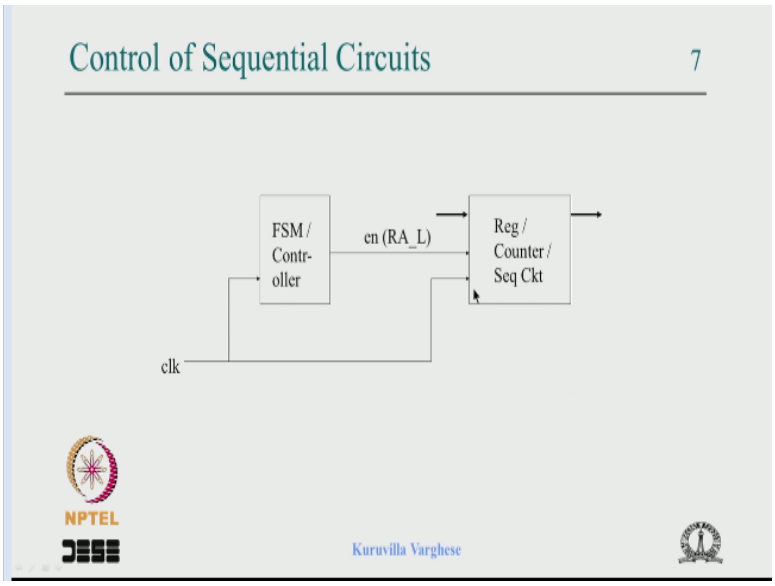
The output as Mealy type output which will have better timing it comes earlier and it has number of states less and so on.

(Refer Slide Time: 04:48)



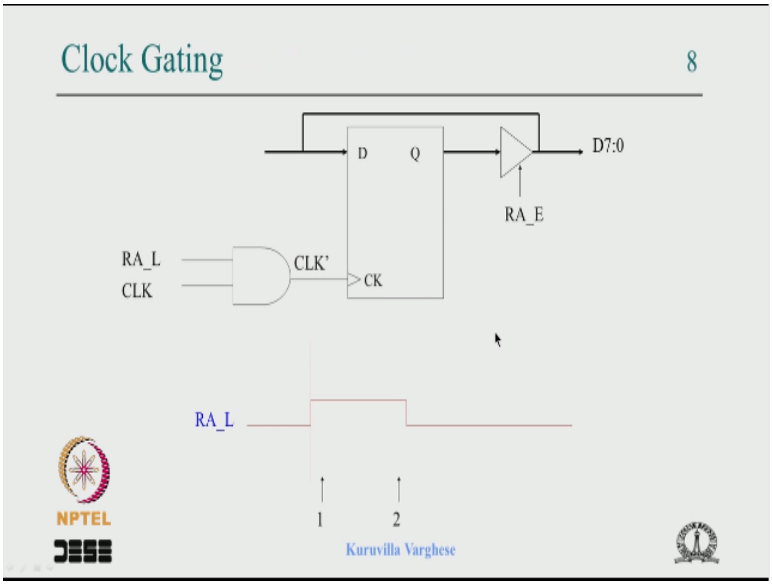
And we have also seen how the state machine generate some enable signal in particular way of looked at the register how this latch enables the latching the data from the input side.

(Refer Slide Time: 05:01)



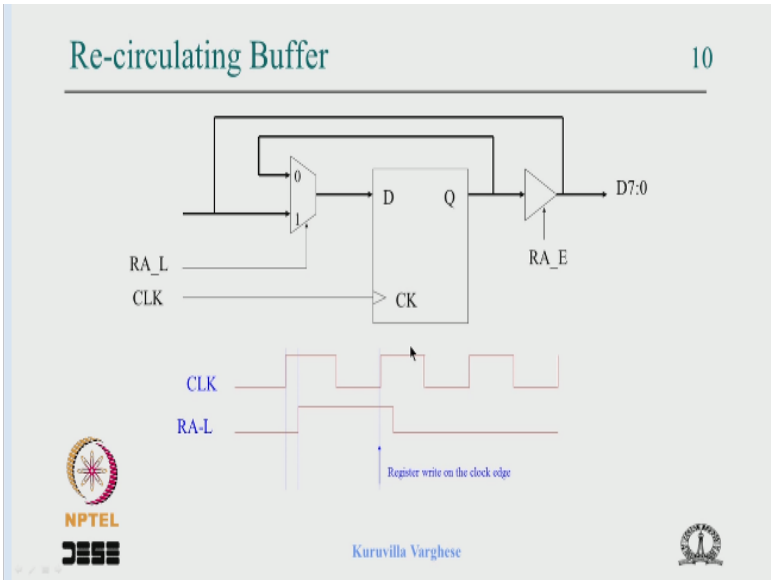
So, that is what we have shown we have a register and FSM working at the same clock and this latch signal is decoded from a state which is controlling the register. So, when the latch is high and the clock comes the data gets latch.

(Refer Slide Time: 05:16)



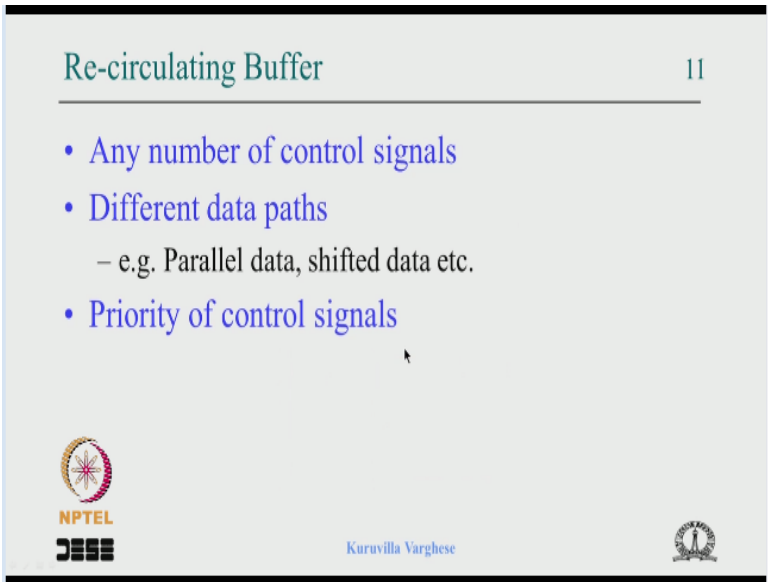
And a possible implementation was this and we have found a problem with it that it has 2 kind of active edges come here. The first one may not meet the kind of the timing requirement for the data. Second one could create timing problem and if this is used I said to clock a counter it might kind of count twice and so on.

(Refer Slide Time: 05:42



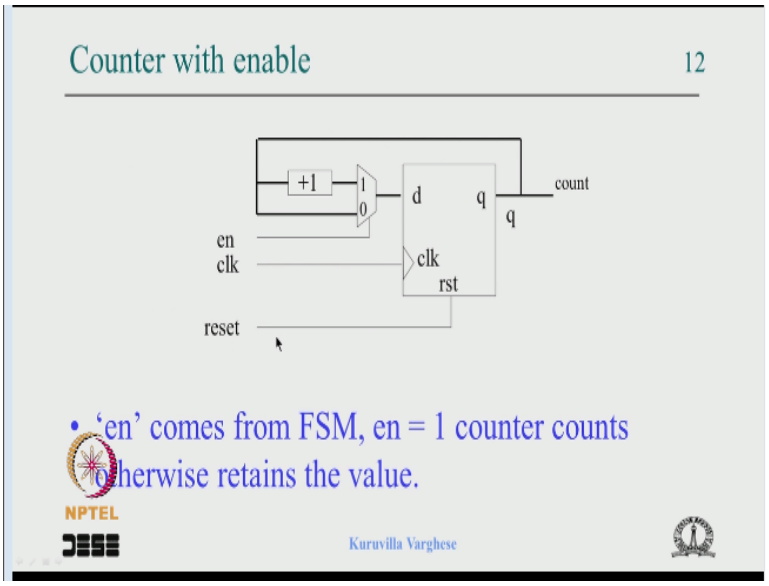
And the solution is a re-circulating mugs where we moved the latch signal or enable signal in the data path. And that works properly, because that is enabling the input to the input of the register and neatly in the next clock edge, it is latch enough time is there. The only issue with this project it is clocking all the time. Either it is connecting the input to the register or it is re-circulating.

(Refer Slide Time: 06:10)



And that and this idea can be applied to kind of any number of inputs Not only just enable.

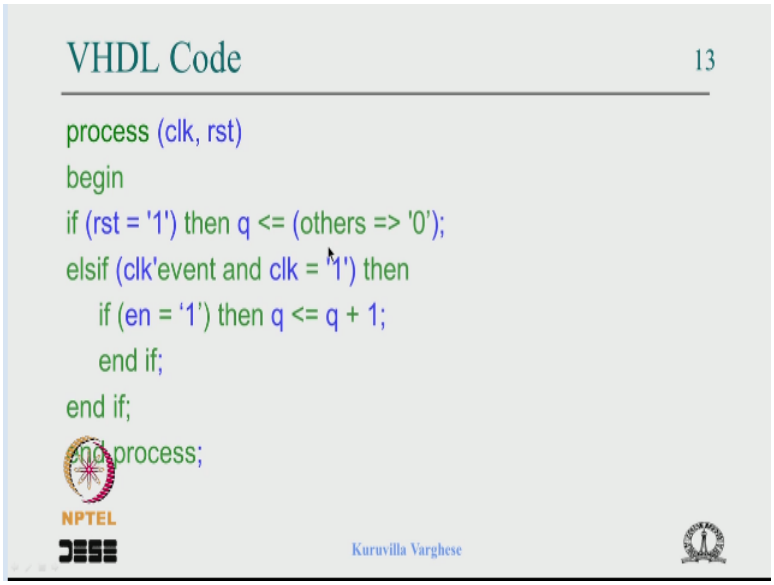
(Refer Slide Time: 06:14)



So, we are showing this as a enable of a counter, when the enable is 1 which has to be synchronous with the clock. And then the counter is incremented, then that is very useful kind of situation that we are keeping track of this state machine is keeping track of some event and event as happen and then we enable the event counter. So, that the counter can be incremented, because say we have working on some communication issue.

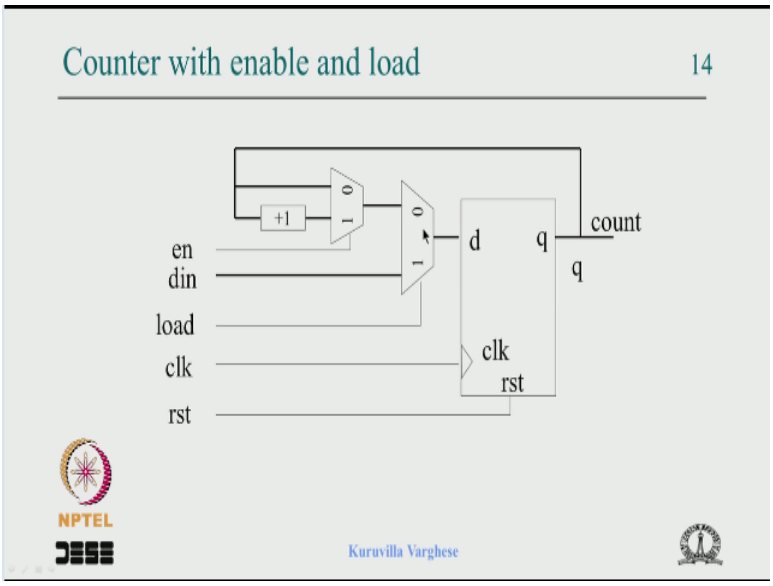
And we are counting the number of bits, so when we have kind of done all the processing then we can increment the bit, bit counter and so on. There are lot of application can be use.

(Refer Slide Time: 06:58)



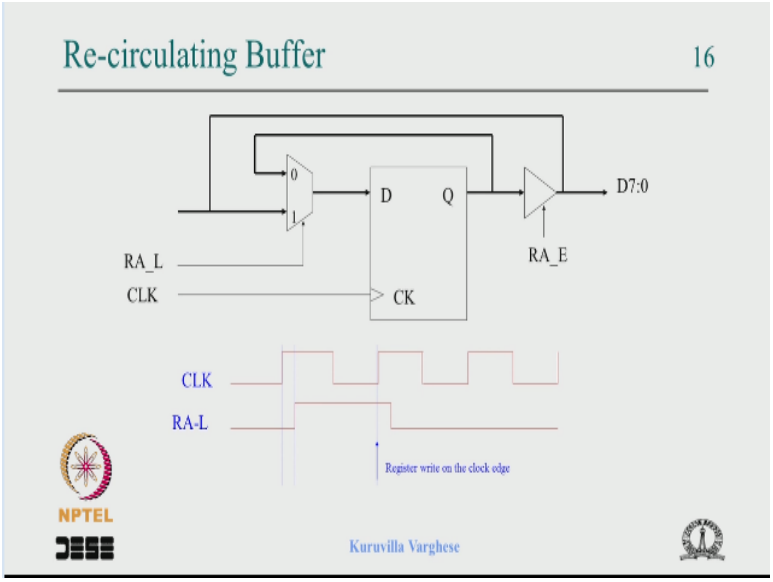
And we have seen the corresponding code.

(Refer Slide Time: 07:01)



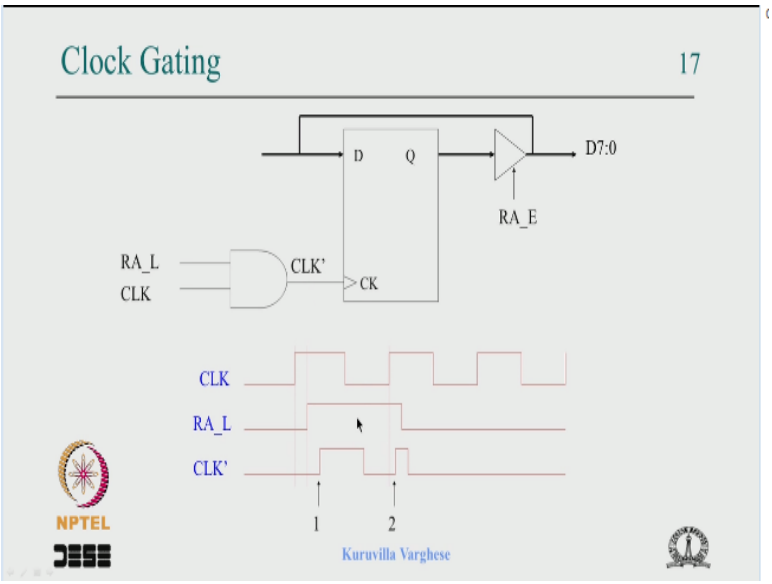
And we have also looked at we have a limited it to enable we have another load signal then another mugs come which has priority. So, it comes close to the d input of flip-flops**.** So, then we have seen the equivalent code.

(Refer Slide Time: 07:19)



And we said that the problem with this re-circulating mux is that lot of power dispassion.

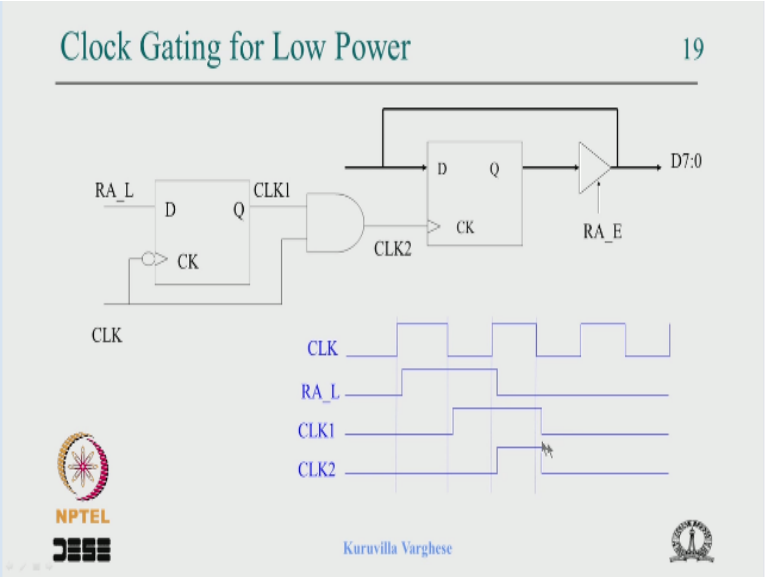
(Refer Slide Time: 07:26)



And this actually as much better in the power dissipation only because when it is switches it dissipates power, the issue is that it has kind of 2 edges. So, if you move this to the negative edge side then if you and it then you get a single pulse okay. correct location. So, we decided to resynchronise this enable signal with the negative edge off the clock.

Then this will move here like that and if you and it you get a neat pulse and that is what we have done in the next scheme.

(Refer Slide Time: 08:06)

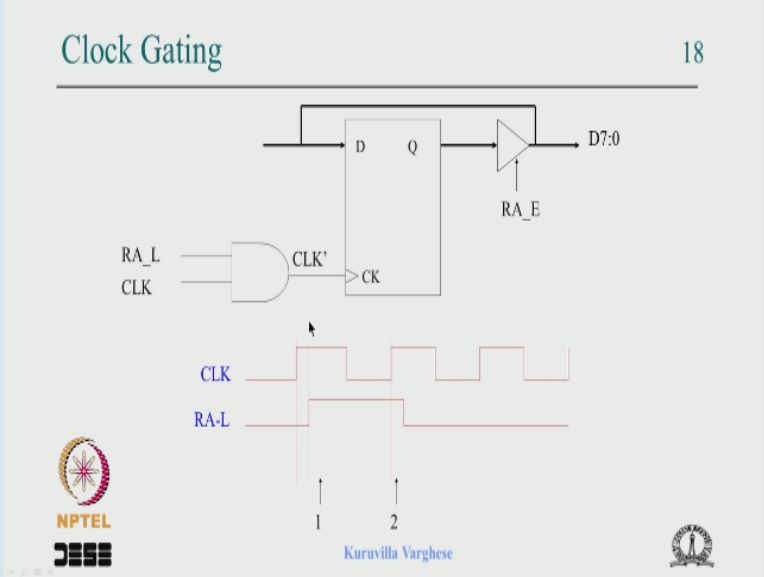


Here so, clock this a negative edge to get flip-flop. And the enable signal is given to that, so it is re-synchronise to negative edge clock 1. That is anded with the clock and that is use for clocking

the register. So, that is what is shown here. This was the original enable signal and it is given to the input of this D flip-flop. So, since it is negative edge to get at this edge this is latch.

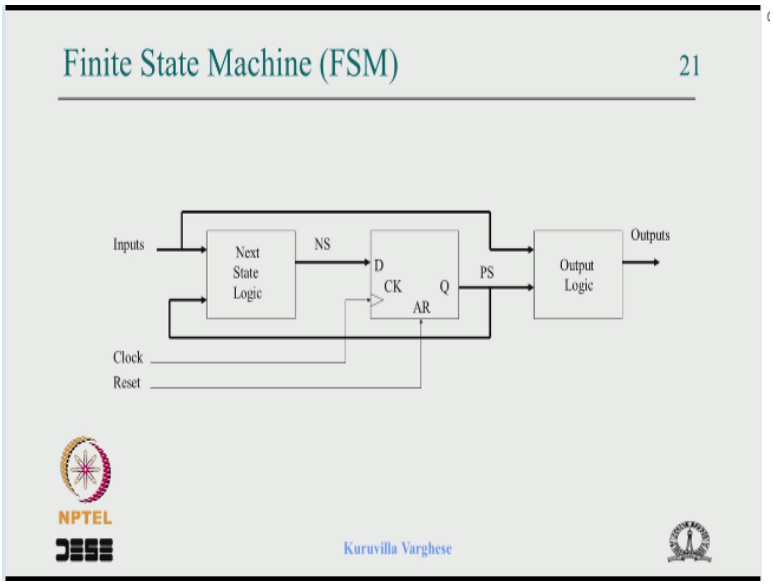
So, there is a TCQ delay it appears here and this edge it is kind of sorry, this edge that is the 0 is transferred here. So, you get it here, you with a clock this clock 1. That is what we do and the clock 2 is a neat pulse and the single pulse you get. So, this is a kind of clocking scheme for low power.

(Refer Slide Time: 09:07)



And as I said whether you write a VHDL code for this or VHDL code for this many a times it can convert this into kind of this particular circuit. At the VHDL code itself it can be you know converted. It is very easy to do that and the tools can do it automatically. So, that is where we have kind of stop, then the last lecture and let us move on with a some more issues of state machine.

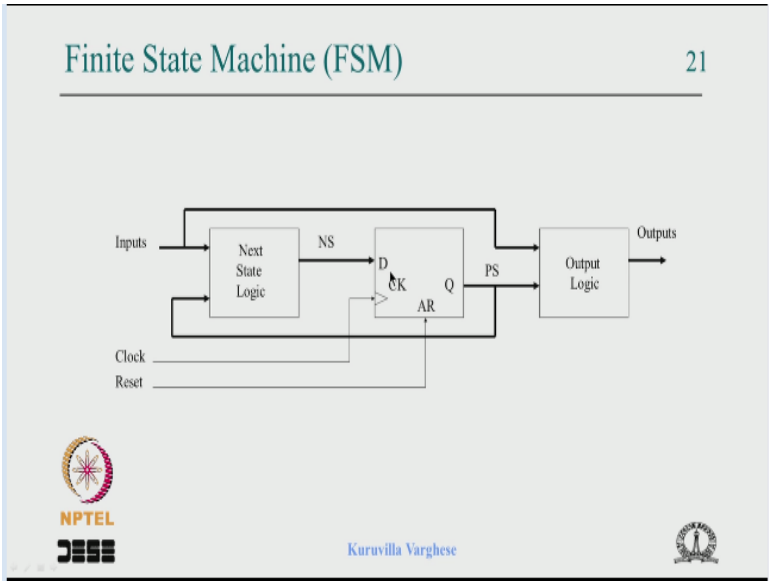
(Refer Slide Time: 09:40)



And once again before going to that just a reminder the state machine structure is like this you have state flip-flops keep track of the state. The output is a present state which is combined with the input and generate the next state which is latched on the next clock. So, this is kind of one path and the next path is that the present state along with the input as decoded for output.

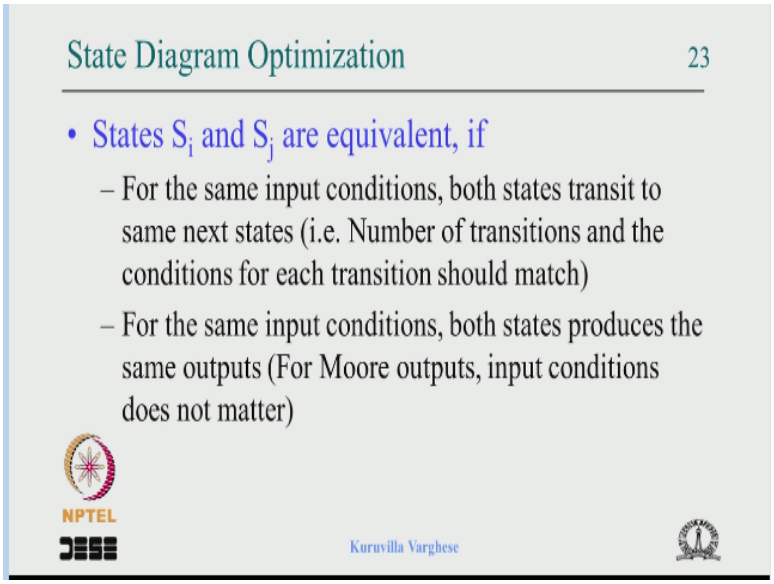
And it is Moore output it is only present state which is decoded. It is a Mealy output both input and the present state is decoded. And we said if you look at this logic and this logic both receive the present state and input. And so we can combine this logic in 1 block. Only thing is that this output is shown from there.

(Refer Slide Time: 10:27)



So, that is what is shown here. This logic comprises of next state logic and output logic. The next state logic will give next state, the output logic will give the output both are useful for analysing the writing VHDL code and so on. So, please keep both in your mind the some textbook show this and some textbook in your basic courses show this, it does not matter. But then it is better to keep both we use in your mind. So, let us take up this issue.

(Refer Slide Time: 11:03)



The state diagram optimization, so when you kind of come out with a control algorithm like we have seen that you problem specification then we said that you draw a waveform of the input and output. And looking at the waveform you can write a verbal description of what we are intended do many a times and that is algorithm you know. Formally port it that verbal description can be turned into a proper algorithm with iteration and all that.

And you write and many a times it can be complex and at least at the first stage when you do that the control algorithm is little elaborate complex and you draw the state diagram. It may happen that there are identical states in different part. Different parts means assume that you are in a particular state depending on the input you branch to different kind of sections and do something okay.

So, the it may happen that in one state in 1 section is similar to another state in another section and because it is many a times you draw it on a paper it is very difficult to kind of identify that

both are kind of identical okay. So, this the tool can do it, tool can detect kind of identical or we can even call equivalent you know identical is not a good word technically.

But we can say equivalent states okay. Now the question arises what are the equivalent states okay definitely from our your kind of experience with the state diagram you can say 2 states si and sj are equivalent. At least we can say they produces the same output okay. If it is Moore then it just produces same output or we say if it is Mealy kind of output, then we say 2 state for the same input condition produces a same output.

They are we cannot say they are equivalent at least there half, half the job is done with regard to output. Now we have to look at the transitions also okay. now if you look at the transition there are 2 stages, there could be 2 states which are transiting from the same previous state. And there could 2 states from it is the transition occurs to the same next state okay. So, for an equivalent state which one is consider is the question, but it is very easy.

We know that the next state is a function of the present state okay. So, if 2 states are has to be equivalent they should produce a same next state, because next state is a function of the present state okay. So, if 2 present state produces a same next state. Then in the that next state Boolean equation this present state appears. So, if they are identical I mean they in the next state is identical then we can say these states are equivalent.

So, let us put whatever I have told formally, so let us go to the slide. So, we say states si and sj are equivalent. If for the same input condition both state transit to the same next states okay. now when you when I say same next state si and sj can transit to sk under the same input condition. That means like we discuss earlier the maybe an input signal called start and si and sj will remain in their states when the start is low, when the start is high si transit to sk, sj transit to sk say.

Assume that these are different parts of state diagram okay not that do not assume that both are kind of concurrently working when the state machine in si and the start comes it goes to sk, when the state machine is an sj and the start comes it go to sk okay. now that is just this simple case

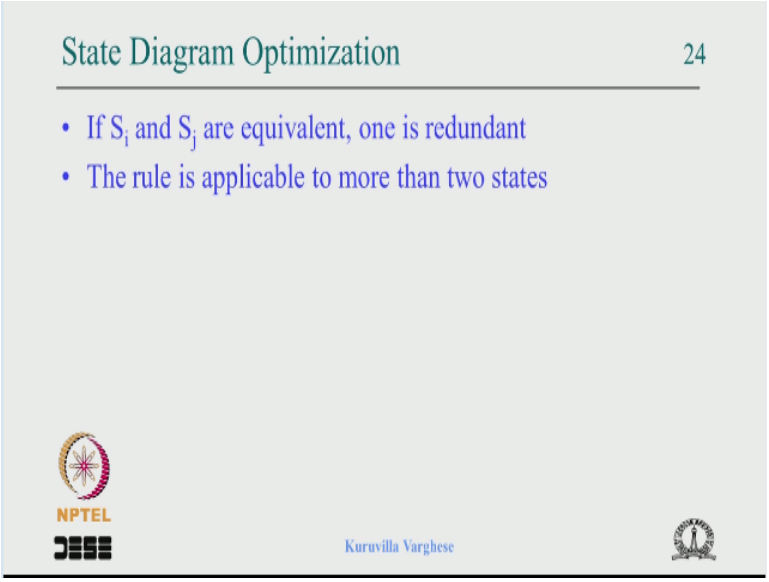
maybe si transit to not only sj on some other input condition is might transit to sorry si transit to sk on some condition, sl in some other condition.

So, sj also should transit upon the same condition to sk and sl. So, all the transition, all the input condition should match okay. that is n we have to state the output for the same input condition both states produces a same output okay . For Moore output you can ignore the input condition does not matter which a Moore output we do not decode inputs.

So, only the present state matter, so that is then we can say si and sj are equivalent. That means for both states transit to same next state for the same condition okay. All the transition all the condition should match. These 2 should produces a same output for exactly same for the same input condition if it is Mealy, if it is Moore just same output. Then si and sj are equivalent.

Then we can replace sj with the si you know that is a game you know simply wherever sj is there rub it off and you know bring those you know arrows you know whichever was going from sj anyway it goes from si and the output also is taken care there is no problem it works properly okay.

(Refer Slide Time: 17:37)



So, that is how, so if si and sj are equivalent one is redundant you can remove it and the rule is applicable to more than 2 states when we formally states we say about 2 states. But then it is easy

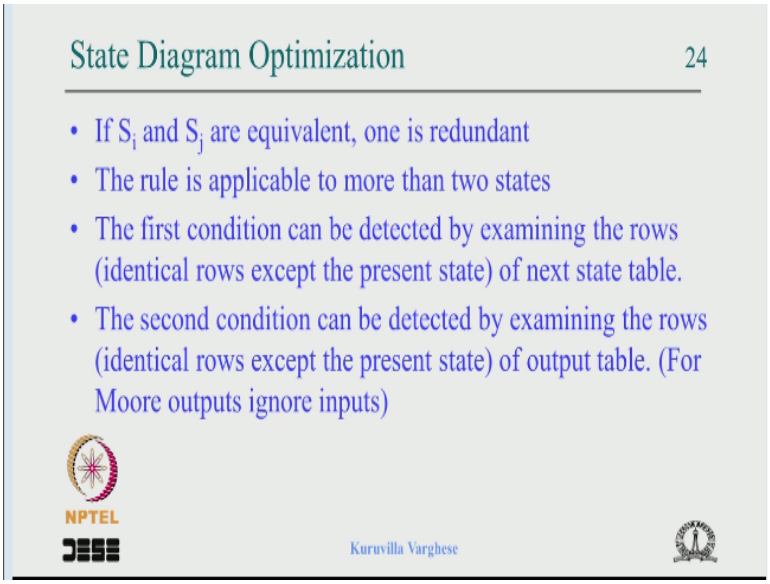
to kind of formally stated, but if 3 states can be identical under the same kind of condition. Now how to detect this you know question is that find we say like in a state diagram like it you know that if there are 32 states spread across.

And a3 heat okay, you have drawn in a big sheet and how do you look in you know. You put a finger on 1 state and keep the other finger on all the other state and looking for and you have to do all the combinations. It is a very luborise job, so on a state diagram it is very difficult to do that even if you kind of that does not help a computer algorithm.

Because ultimately if you want to do that in a tool. We have to formally state the algorithm. So, which is a best place to look, so we know that the first one talks about the next state. So, the ideal place to look for that is in the next state table okay. so, what are we looking we are looking the for 2 states at least. Let us take 2 states, the 2 state for the same input condition produces a same output condition. So, you go to next state table pick up the rows okay.

And wherever you find there are same next state and same input condition and you pickup present state, y are kind of the ideal candidates as for as the next state is concern. So, that is what I state here, the first condition can be detected by examining the rows.

(Refer Slide Time: 19:37)

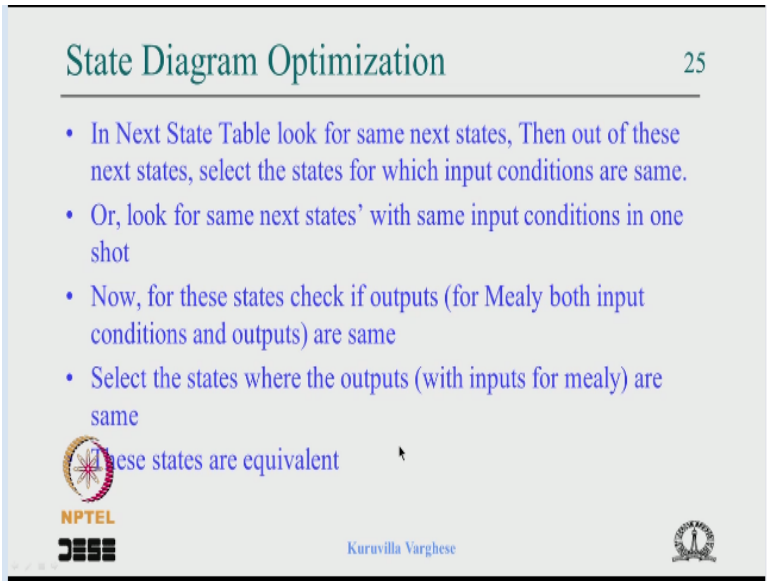


Identical rows except the present state, because the next state is same and the input conditions are same. So, you pickup all that then you get candidate for the present state. Because we have to now detect whether the check whether the output is same. And that can be easily detected in the output logic table output table. So, the second condition can be detected by examine the rows, identical rows except the present state.

So, leave the present state look at the outputs, if output are same and the input conditions are same. Then you can kind of for Moore output you can just look for the identical output. Then you know then you can pickup those states okay. Now there we have 2 kind of make an intersection of 2 state. Because this at the end of it you get you know some stages if you look at it you get some state.

And both are should be handed or intersected then you get correct thing. So, this can be systematically done in an algorithm.

(Refer Slide Time: 20:56)



So, let us take that, so in the next state table look for the same next state you know you look where are the next states are similar. Then out of this next states select the state which have the input condition same. So, if you state an algorithm look for the next state which are all similar pick up all that, look at the input condition. Wherever out of this input conditions are same then you pickup the present state.

Then go to output table check if the outputs are same for this next state. So, wherever the outputs are same pickup those states and then we have for Mealy we have to look for the inputs also you know we have to match the input for Moore you match the output. Then these states where the outputs are same they are equivalent you know. So, systematically you can do that.

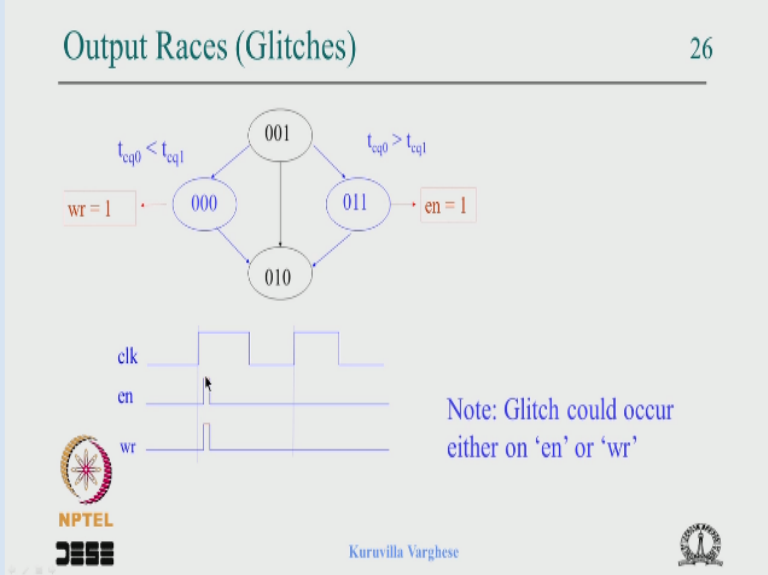
So, the tools can do that you know first look for the identical next state then go for look at in these states the inputs condition the next state table. If there identical pick up those go to the output kind of output table, check for these states where the outputs and input conditions are same, those states will be the equivalent once. And you can remove keep one and remove all the rest.

Then the state diagram is optimise you have to check whether your tool does it maybe when we come to the tool we cannot I am waiting for some more portions to be covered. So, that we can introduce it tools so that once for all we can try quite a few things also that you know that the tools are changing all the time. So, I am in that part of the lecture maybe less useful.

Because after few years if at all this course is used by someone, then the tool part is kind of there is no guarantee that you know it make sense because by then the whole interface and then everything would have change. So, people will how still is useful to see design methodology the steps and output and same. I am hoping that it will be useful, but I am delaying it so that we pick up maximum information input.

So, that you understand everything when I demonstrate the tools, boards and things like that okay. So, let us move on to another issue this we talked about a state diagram optimisation we have in a state diagram how to identify 2 or more states are identical. So, that we can optimise the state diagram. We can kind of remove the redundant state and keep minimal state diagram. So, that the area occupied by the fine 8 state machine is less that is our aim.

(Refer Slide Time: 24:02)



So, let us move on to another issue which is called output racing or a output glitches, so whatever. So, forget about the name, so let us look at the issue okay. So, take a state machine 3 flip-flops okay. So, let us call it the MSP Q2, Q1 and Q0. So, I am just showing a part of a state diagram. So, there assume there is a big state diagram which kind of at least it has more than 4 states.

We do not know whether it occupies 8 states. But more than 4 state which forces to use 3 flip-flop to implemented. And there are we call it Q2, Q1 and Q0 okay. Assume there is a transition from 001 to 010 okay. So, in this when there is no input condition maybe a clock comes this transit to this. So, Q1 is going, Q0 is going from 1 to 0. And Q1 is going from 0 to 1 okay. Now assume that see we have 3 flip-flops.

And OR may not be kind of has a same propagation delay, assume that the Q1 is faster than Q0, that mean TCQ 0 is less than TCQ1 oaky. The propagation delay of first flip-flop is less okay sorry. first flip-flop TCQ1 is less than TCQ0 or TCQ 0 is greater than TCQ1. So, what happens is that you see that it was 001, if this is faster this 0 will become 1 before this 1 become 0.

Then ultimately 1 will move to 0, because the Q0 will change state after a while. So, what happens is that in between while changing the state from 001 to 010 for brief period a transitory state will come which is 011 okay. so, that is what is shown in the picture I have shown in the

picture and saying that the propagation delay of Q0 is lower. I mean it is larger than the TCQ1 or the Q1 is faster.

So, Q1 becomes 1 and for sometime Q0 remind 1 then transit to 0. So, 001 becomes 011 and ultimately become 010 very brief period it becomes you know and it happens you know that it is because you know do not worry about it. We talked about in combinational circuit when there are multiple output, there is difference in the path delay it happens. So, similarly when there are multiple flip-flops.

Because of the variation in the propagation delay of TCQ flip-flop this happens and it is very natural okay. there is nothing wrong with this. Now you can take the other condition if Q0 is faster than Q1. So, take the other condition Q0 is faster than Q1 means this 1 will become 0 before the 0 becomes 1. So, you will get a situation where for a brief period you have all 000. Then it transit to 001.

So, you have the other condition Q0 is faster. So, 1 become 0 then the Q1 chains straight 010. So, either of this condition if the delay does not match exactly one is greater than the other there are 2 condition there could be a transitory state 000 or 011 and the fact is that we do not know whether these are kind of valid state maybe this is part of the state diagram, some other state.

So, you might ask the question to ask is that is there any concern to worry about okay. it does not matter for a brief period let this happen it is only for a very few maybe say 2% of the time or 1% of the time it just you know transit it does not like it settles anyway okay . It looks little (()) (28:41) but assume that these states produce some output okay. Because, this could be valid state in the state diagram or part of our control algorithm.

Assume that there are 2 outputs called enable and write and that is 1 in this state which is valid. So, what happens is that when the clock comes and the present state changes from 001 to 0**1**10. These outputs will briefly go active okay. so, you see such a situation here the state machine in this state the clock comes. So, for a and ultimately this the state here is 0, before was 001 and here it is 010. But brief in between it may go-through 000 or 011.

So, if it goes through 011 you get a right pulse briefly going high glitch. And an enable pulse which is briefly going high if it happens this way okay. now once again is that a cost for a concern you know is that create some problem now it depends, how this particular output is used. If this is used like to enable a counter we know that it does not matter.

Because we know that when you enable something using clock gating or re-circulating mugs nothing is going to happen. Because we see unless the enable structures all over here then only this clock will kind of become active, enable the re-circulating mugs. So, may not in a synchronous case it may not kind of create problem. But we do not know how this is used maybe this is use asynchronously to increment a counter or this right is going to an asynchronous memory as a right pulse.

Then what happens for a brief duration the right pulse go high and if the access time is enough maybe the address line is floating or a some values there data line is floating. But then something get latched into some address and which can corrupt some location okay. So, depending on like how fast is the memory, how fast how you know sort is the access time and so on.

So, this could create problem so, that you should keep in mind. Because this is called output racing, because these 2 outputs to Q1 and Q0 is racing each other and one is faster than the other, that can produce briefly the transitory state it can produce the output, this may or may not cause problem. And when there are critical applications we have to worry about it like you have making a kind of medical equipment which is use for some kind of radiation and you have designing a controller for it you have to be very careful you know.

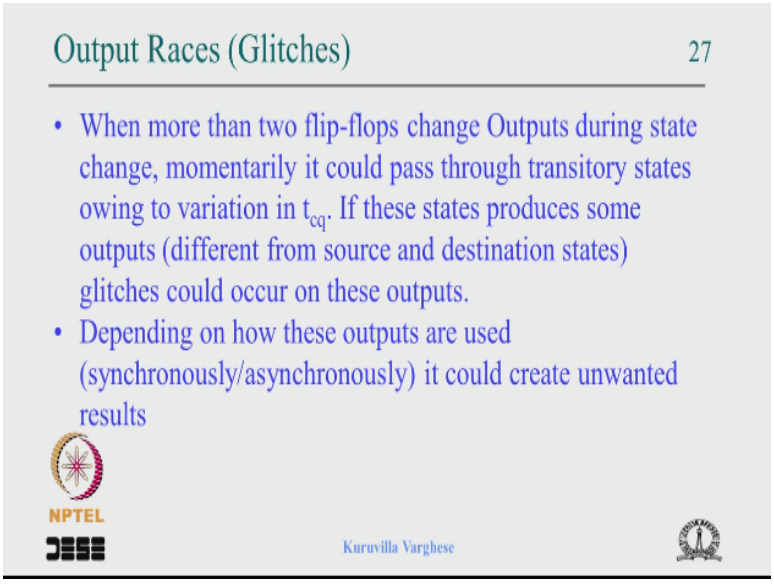
But if you are doing something very not that kind of a there such a critical application is not there. It may not matter you know, some user intervention user is pressing a key if it is not detected maybe the user can press it again may not matter. So, it depends on the application the criticality of the application you have to break your head about it okay.

Always when you are designing something maybe you are in a huge team okay. you are doing a part of the design maybe you are doing front end design you are doing the back end design and you do not know like some people sometime you do the back end design of a kind of verification. And if you ask what is this use for they may not even aware which chip there working on our they widely know.

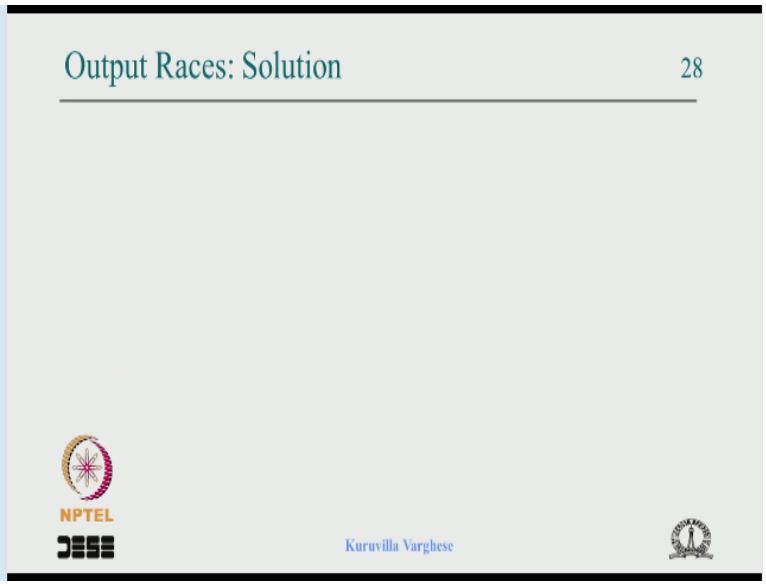
And they may not even know which is a application it is very kind of very bad situation you have working on a design on a big team you should know what are you working and what application it is used. So, that you can kind of judicially you know choose the design practices you know which is how the reliability cost, flexibility all that can be balanced properly depending on the application.

Very important like you are sending something to the space you know and a space vehicle is being controlled very critical nothing can be corrected on the way sometime from the ground. So, you have to make it very critical, so it depends where you are on a aircraft you have to be very careful. So, let us go ahead, so this what I have stated earlier.

(Refer Slide Time: 33:44)



(Refer Slide Time: 33:45)



Now the question is that what is a solution you know there is fine there is an output race what is the solution okay. Now when you have a problem okay when you have such a problem and when you in general okay maybe you are used to always learning from the textbook there is some problem, the textbook tell you some answers. But mind you this textbooks get the answer from the research somebody has done you know.

At the beginning when somebody has propose the state machine Moore or Mealy all these issues would not have been thought about it you know. That implementation kind of detail, but when you have a problem you should know to solve it in the first place you should know what cause the problem okay. Then you can attack find a solution say i have heard of a story.

There was a highway and there was a sharp turn in the highway okay you know highway is very you know very high speed highway is there where vehicles are going at very high speed above say 100 kilo meter per hour and there is a sharp 90 degree turn for whatever reason there is a turn and at night there used to be lot of accidents and the residence are kind of worked out you know.

They have to call the ambulance service, the police and so on. So, the residence decided to do something about it and they have met together and somebody suggested that let us station an ambulance you know at the turning. So, that when the accidents happen people can be picked up

and everybody like the idea you know it is a great thing, because whenever the problem happens you used to call an ambulance.

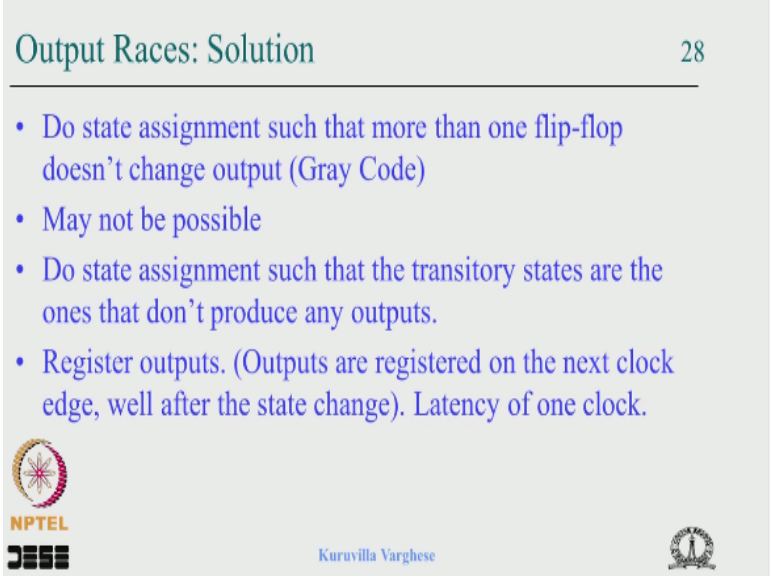
So, if then ambulance is ready nearby, this can be attended to okay. so, everyone agreed and they were going for kind of an collection to buy the ambulance and some wise man said see the problem has occurred, because there is a sharp turn in the highway, the to solve you have to straighten the turn you know reduce the sharp turn make it smooth, then there will be less problem you know and everyone agreed when about doing that.

So, similarly when you have a problem always think what cost the problem there could be a at least some solution to it, and let us see here the output raise what cost of problem is that we can say that there are there is a like a look likes that there is a discrepancy among the delay you know in the flip-flop. There is a kind of variation in the propagation delay okay. Now if you once again and this is not in our control

I am mean you fabricated and everybody fabricate try to fabricated on the same dive very close together but yet when you do the processing the like electrochemical processing. There could be kind of variation but if you think again deeply, you see the problem has come, because there are 2 flip-flops changing the state that is why the variation in delay is affecting. Suppose the state assignment was 001 to 011. So, 1 is not changing only the q1 was changing the state.

So, it shows that if the state transition can be logically adjacent okay or you say some time the grey coding if the adjacent stage like when a state transit from one state to another those stage are logically adjacent. They change only by one bit then this problem will not raise so, at least like from working out from the fundamental cost.

(Refer Slide Time: 37:46)



We can say the solution is that do state assignment such that more than one flip-flop does not change output okay well and it is good you know it works at least in simple cases. You have a state transit to next state then it transit to the next state and so on, it comes back then you can do say 000001011 you know the grey code and you can do that okay. But what about all kinds of transition, you have a state then it transit to 3 state from there.

It goes in a loop from here it comes back to something may not be able to do the you know logically adjacent coding in all cases. You take a simple case you have 1 state here it transit to next state it transit to another state and come back. So, 3 stages are adjacent to each other. You cannot do a grey coding no because if you remember corn of map where we do the logical adjacent see for corn of map.

We know the 3 states cannot be logically adjacent to each other for states can be okay maybe you will say you insert 1 dummy state in between which does nothing. It can be done but then in a complex state diagram. And you find there are lot of you know kind of adjacent problem, you introduce the dummy state then it becomes equivalent I mean complex and lot of various wasted around that and that not very bright solution.

So, if this is possible if grey code is possible do that okay, it may not be possible and then what we do is that maybe we can say another issue in the original diagram is that. It is producing

output maybe there are some transitory state which does not produce it does not change the output then do that the state assignment such a way. But very if a state does not produce any different output what is the use of such a state.

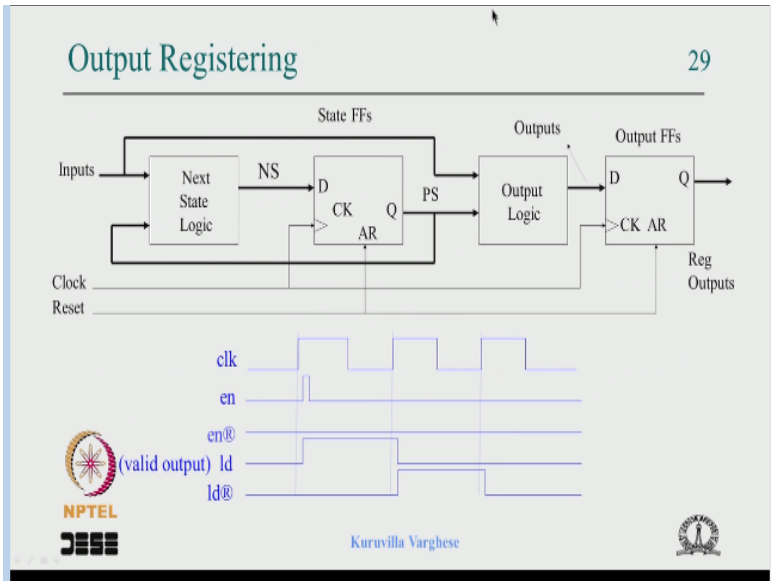
So, though it looks very sound very nicely some solution sound very nicely, but it may not make much sense, you know many a times when people talk the what they talk sound very good. But it may not be practically as so relevance you have to very careful, when you make statement so, this statement sound good, but practically does not happen if there is a transitory state which produces the output same as before or same as after.

Then you may ask what is the doing okay it may not do any useful part process suppose now assume that you cannot do anything what is the solution now you are struck with this problem. You have a glitch okay then and it creates some problem and you want to solve it, what is these solution so, now then let us look at the ideal again what we look at is that how does a good output looks like, if enable is a valued output.

It would have started from here it would have gone all the way up to here. This is a transitory output so, it appears as a glitch okay, it is a valued output it extend all the way at least for a 1 clock period, So, why not we put at the output logic 1 flip-flop and give a clock to it so, that this if valued output is a registered and get a registered output. This glitch want be registered because it appears briefly for a short duration.

So that is the kind of the most kind of permanent solution you output the register the output you know and output register in the next clock edge, well after the states state change. But the output will appear after 1 clock period okay.

(Refer Slide Time: 41:45)



That is what I have shown here what we do is that you have the output logic you put a suppose there are 3 output 3 flip flops for 3 output give the same clock and the reset and this is the registered output. Earlier we used to take the output here, now you look at it. Suppose we have an enable output which is glitching okay and which is coming here there is a load output which is a valid output then you see the difference like.

Since when the clock comes in this clock edge nothing happens enable is not high, the next clock edge when the clock comes here that this glitch would have disappeared. So, the enable register remind 0 throughout all the time**.** But if it is a valid output which starts at this point goes all the way to the next clock period. At least is the worst case 1 clock period it can remain high does not matter.

So, in the worst in the critical case it reminds there for 1 clock period. But when this clock edge comes it is high. So, the registered version of that output comes at the output with a 1 clock cycle delay. So, that means that all the glitches go off. But all the outputs kind of lack by 1 clock cycle you know lack with respect to the state change okay. but it is less of a problem you know that .

In the state diagram these come to the starting state when it starts interacting or start initialising, everything is delayed by 1 clock period as for as this particular output is concern not a problem

not a very serious problem. But you should keep that in mind you know there can be side effect if you do not remember that you know, that a little care has to be taken with respect to the timing.

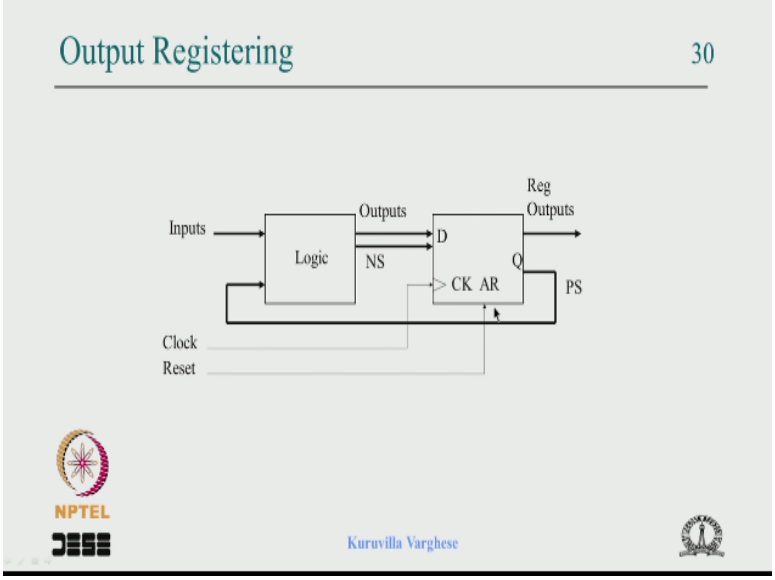
You have to analyse the timing what happens because you see that this output is going to the data path and data path is giving some input. And this latency can reflect sometime in the control algorithm. So, you have to take care, you have to think do not take things just because you skate you know solution is there you put the output register and you know assume that everything works you please analyse the timing**.**

There you can think of the situation work this makes a difference to the controller algorithm, something need to be tweet for proper work. You have to keep that in mind you know. If you are clever you can find a solution maybe you can solve a very critical problem if you understand the issue. But now look at it you know that is a permanent solution for output racing or output glitch.

But one thing I want to kind of bring out from this diagram. The earlier we said that this output logic and the next state logic gets a input, the same input you know the present state is here and input here. The real inputs from the data path is input to this state, input to the state and we combine this state here. So, now if you look at this diagram see this state and this 2 state look identical. So, if you can take this output logic put it here and call logic.

Now that this flip-flops and this flip-flops are same you know you get the clock and asynchronous reset same. So, we can combine this here. Only thing is that combines output registering as well as the state flip-flops.

(Refer Slide Time: 45:44)



So, that is what is shown here. Now the output registering the diagram looks much more simplified you have a set of flip-flops which combines a state variable. That means if there are 5 states there are 3 flip-flops. There are 3, 4 the state variables Q2, Q1, Q0. There are 3 outputs then there are 3 flip-flops for output. And this is the logic which 1 part is the next state logic which decode the present state and the input.

And decide the next state which goes to the state flip-flops. And there is output logic which look at the present state and input produce the output which goes to 3 flip-flops to register the output. And that comes here out of the flip-flops come the present state and the registered output. But one thing you should remember that one good thing happens here. We said that a set of registers with respect to the with a preceding logic can be coded in a single process.

So, though it looks very kind of complex with 4 blocks, when it comes to VHDL coding we can write a single process to implement this state machine okay, whether the tool detects it we have to check it. But at least from what we have learned it is a very nice thing you can write a single process. And we will see that maybe in the next I realise that maybe it is wise to discuss the VHDL coding of the state machine soon.

Before we go ahead maybe in the next lecture we look at it how to, what are the possible ways to doing the VHDL coding. And this is very easy like you write a process to code this register along

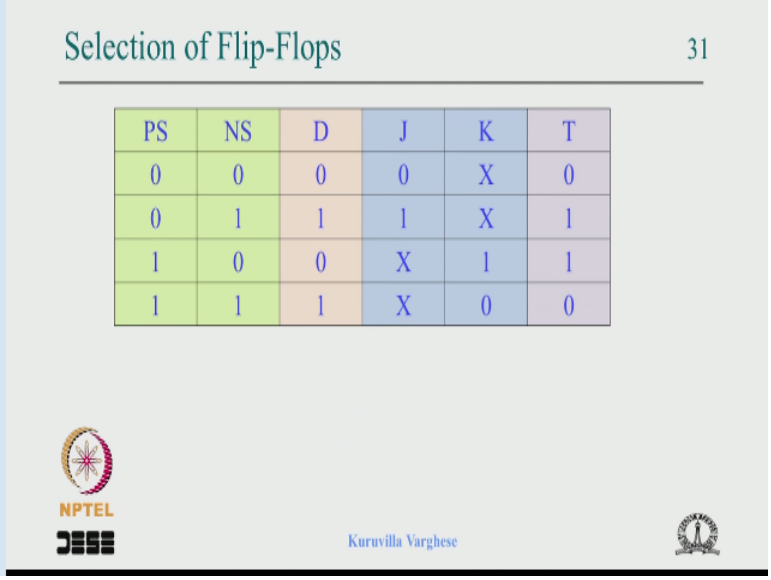
with the next state logic and output logic. So, it is a very simplified coding and you want to generate a state machine with registered output. The VHDL coding is very easy just put this state diagram and start writing the code you will get it.

So, that is what we have discussed now the output registering wherein when there are multiple flip-flops change state in the same transition. Then due to mismatch of variation in the propagation delay of flip-flops, there could be transitory state which can produce output you know glitches. And depending on how these outputs are use this, this can create problem in our circuit.

And to solve we said the problem because of the discrepancy or multiple flip-flops changing states. So, grey coding a solution and the ideal solution is to register the output in the next clock edge. So, that this glitches disappear the valid once come in the latency of 1 clock period. And we have seen that when we kind of do little rework on that structure, you get a nice structure which can be the VHDL coding of that becomes very easy okay.

So, let us move forward to another issue that is the selection of flip-flops okay. now in all our discussion to make things easy we assume that we are going to use the D flip-flops all the time we said fine I mean the assumption was that we use the D flip-flops but you could use.

(Refer Slide Time: 49:26)



Let us come to this you could use other flip-flops like J, K or toggle flip-flops and so on. And let us take this transition the present state to the next state okay. So, like there are these are the possible transition 0, 0 to 0, 0 to 1, 1 to 0 and 1 to 1 okay. Now in the state machine if you are using D flip-flop okay the sequential circuit used, D flip-flop to implement the state machine. Then we know that it is very easy.

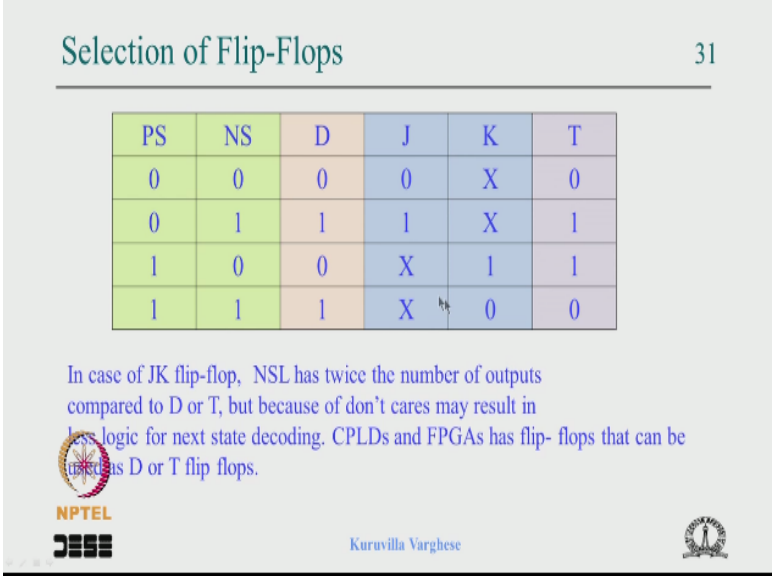
The input to the D should be same as next state you know it is very easy. Like we do not care what was the present state, if the next state is 0, the D has to be 0, next state is 1 D has to be 1 okay, but instead of D flip-flop if you are using a JK flip-flop okay. Then that means like in the state machine like what I am talking is that this particular flip-flops instead of D if you use JK then you have a J and K.

Then you should know that for each like earlier we had d1 say d2, d1, d0 coming out of the next state logic. But you have now J2K2, J1K1, J0K0. So, in the next state logic will twice the output for the JK flip-flop okay. so, if that happens if you are using JK flip-flop here. Then condition for that to decide when you work out the next state table it will be like that. If the present state is 0 next state is 0 then we should disable the set.

So, J has to be 0 and K is do not care, it does not matter okay K can be 1 then it is reset you know the next state is reset it does not matter. 0 to 1 J has to be 1, it is set operation kind of J is 1 and K can be 0 I mean K can be do not care, if K0 it is a set operation K is 1, then it is a toggle operation it does not matter whether it is 0 to 1 can be set or a toggle. Similarly 1 to 0 K has to be 1 it is a reset operation.

And J is do not care, J0 it is reset, J is 1 it is a toggle, because 1 to 0 it toggles, 1 to 1 then that it should not be kind of reset. So, K has to be 0 J is do not care if it is 00 no setting, if it is J is 1 then it is a setting operation does not matter.

(Refer Slide Time: 52:06)



So, that is what is the transition table for the JK and if it is a toggle flip-flop, then you know that 0 to 0 it should not toggle. So, toggle is 0, 0 to 1 it has to toggle 1, 1 to 0 then toggle is 1, 1 to 1 it is 0. So, toggle flip-flop by which I mean there is an input called toggle and if it is 0 the output 1 toggle, if it is 1 the output will toggle, that is a meaning of it. But looks in the case of JK

The next state logic will have to produce twice the output with respect to D flip-flop so, you will find that internal to the next state logic there will be a some logic for J2, some for K2, some for J1, some for K1 and things like that and at least for a (()) (53:02) might looks like the next state logic in the case of JK could be kind of little complex than the D or T. But mind you have do not cares in K and J in the case of JK.

So, that could kind of minimise the next state logic area we are not able to say that. So, though the JK K is different from D, you have the twice the number of output with respect state logic. Because of the do not care the area of the next state logic can be less than in the case of D and T you know that you should keep in mind. And when it comes to the FPGA normally you get D or T flip-flops okay.

JK flip-flops are not there I am not very I do not remember now at least in CPLD you have an option of D and T not because that in the state machine this is useful. But for counters you know

that the counter has toggling action like a like at least the binary counter you know it counts very sequentially 00,001,0010,11 then the next bit is toggle you know.

So, the toggle flip-flop owing to counter kind of sequence the binary sequence this is useful in the counting application. So, the CPLDs and FPGAs has flip-flops which can be use for the counting and wherever there is toggling application. So, maybe we have less time left for the next kind of issue to be taken. So, essentially we have looked at the 3 issues today.

One was the basically we have looked at the state diagram optimisation and wherein we said that basically the issue is to find identical state and that is identical if both produces a same output for the same input condition and they transit to same next state for the identical condition all the number of transition and number of next state should match.

Then there identical we have discuss how to kind of identify this from the next state table and the output table and somewhat formal steps we have stated. But it can be converted into a algorithms suitable for kind of software can be implemented. Some tools implement that you can check whether the tool implements it. And then we looked at the output racing and we have said the grey coding is a solution, output registering is a solution.

Then we have looked at selection of flip-flops and you can use D flip-flop, T flip-flops or JK flip-flops. In the case of JK flip-flop the next state logic has to produce twice the number of output. But then because of the do not care the area can be less, but practically the FPGAs and CPLDs does not have JK flip-flops mostly it has D and T flip-flops. Once again T flip-flops are given for kind of binary counting could be useful in the case of the state machine.

The again there are ways the tools can choose the ideal type of flip-flop or you can kind of tell the tool to choose a particular flip-flop. If you are sure of the kind of some kind of advantage you get out of that. So, maybe in the next lecture we will go look at the VHDL coding of the state machine. Since the state machine is an important part.

We will look at it not that we have learned all what is required, but formally putting it together some kind of something specific to state machine coding can be discussed there. So, please revise what we have looked at it go through the timing work out some make some practical cases think of when this is applied to the practical cases what are the kind of side effects it produces.

Try to solve it or if you have done similar work you think you connect with these lectures those are working in industry you can kind of relate to what you have experience what you have done in a project or in a work you are doing now. So, please revise it and I stop here, I wish you all the best and thank you.