

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

**Lecture-28**  
**FSM Issues 5**

So, welcome to this lecture on advance digital system design in the course digital system design with PLDs and FPGAs. The last few lectures we have been discussing some issues in the controller FSM issues. And the last lecture we have looked at the problem of state assignment we have found that it is an intractable problem to it systematically.

So, we have try some heuristic approach to minimise in the area. Then we have looked at the problem of unused states what problems it can create and how to handle it for a fold tolerant, finite state machine design okay. So, when I say fold tolerant it depends on the application. But, nowadays in general it is not a big problem to take care of that because we are not limited by too much by the area.

So, it is better to take safe approach, we have not completed that maybe the last few slides were kind of remaining, so in this we are going to review that part. So, at that time I will complete and continue with some more issues in the finite state machine I am hoping to complete that today. And look at the issue of synchronisation briefly, because we cannot kind of take an extensive analysis.

And all that in this course we do not have time it takes some quite of few lectures to complete it. But then in a gist you know what is the problem what need to be done at least something which can be done routinely, that we will see, but there are advance techniques which you may have to refer to the literature. So, let us go into the last week slides last lecture slides.

**(Refer Slide Time: 02:29)**

- Number of states =  $s$
- Number of flip-flops =  $n = \lceil \log_2 s \rceil$
- Number of possible ways to do the state assignment?  
 $P(2^n, s)$
- e.g.  $s=17$   $n=5$  (Minimize Area)  
 $P(2^n, s) = 32! / (32-17)!$   
 $= 32 \times 31 \times \dots \times 18 \times 17 \times 16 = 2.5 \dots \times 10^{44}$
- NSL Minimization
- PL Minimization



So, this is what we have seen we have looked at the problem of state assignment. Suppose a state diagram or a state machine has number of states as  $s$ . Then the number of flip-flops are  $n$  okay. So, which is  $\log s$  to the base 2, the ceiling of  $\log s$  to the base 2 okay to make it an integer and now the moment you have  $n$  flip-flops  $2^n$  possible states are there. The assignment talks about assigning this  $2^n$  to  $s$  okay.

First of all how many possible ways to do the state assignment it is a problem of permutation because states are unique and this  $2^n$  possible states are also unique okay. So, it is a permutation, so it is permutation of  $2^n$ ,  $s$  and for even a reasonable size FSM. It is a medium level FSM we can say like states is 17 and  $n$  is 5, we will end up with a huge number of possible assignments.

And suppose our aim is to minimise the logic area of next state logic and output logic. Then this is an intractable problem. So, we look for the heuristic solution to this 2 problems.

**(Refer Slide Time: 03:57)**

## NSL Optimization

16

- Our aim is to do the state assignment such that NSL is minimized (minimum area)
- As we have seen a search through all possible assignments to find the assignment which results in the minimum area is almost impossible in terms of computation time
- Hence, we look for some Heuristic solution, where we follow some sensible rules, that would result in near optimal solution



NPTEL  
JEE

Kuruvilla Varghese



And definitely if you look we are as for as say you take the next state logic optimisation wherein we are looking at the next state as a function of the present state and input. So, when you minimise we are trying to play with the midterms which is formed of the input condition and the present state and what we have control is the state, because we are assigning the states.

**(Refer Slide Time: 04:30)**

## NSL Minimization

18

Inputs			Present State			Next State		
$I_1$	$I_2$	$I_3$	$Q_2$	$Q_1$	$Q_0$	$D_2$	$D_1$	$D_0$
1	1	0	0	1	0	1	0	1
1	1	0	0	1	1	1	0	1



NPTEL  
JEE

Kuruvilla Varghese

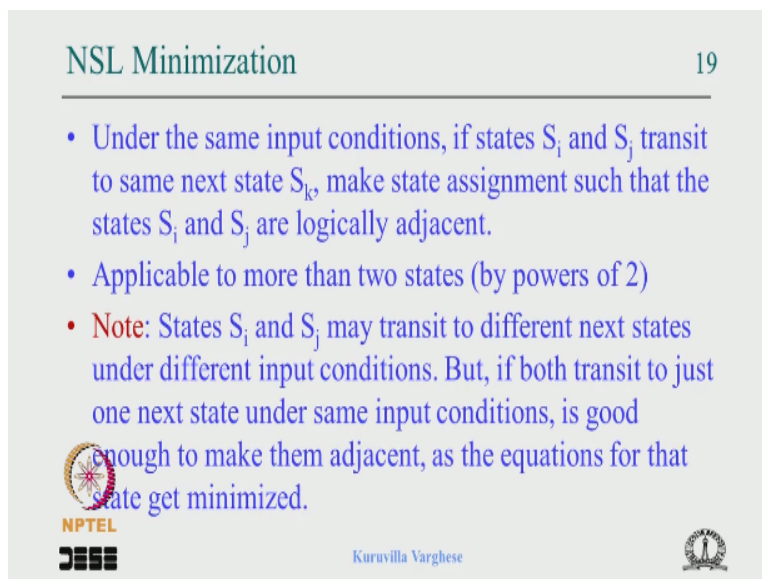


So, that tells that if we look in the next state table and this is what we have control you know I have written the assign the state. But assume that this is  $S_i$  and this is  $S_j$ , we are trying to find and this is  $S_k$ , you know both are  $S_k$  okay. Now I put kind of numerical values, but assume that this state is  $S_i$ , this is  $S_j$ , this is  $S_k$  and  $S_k$ . So, if  $S_i$  and  $S_j$  transit to same next state  $S_k$  under the input same input condition.

Then we will end up with say when you form equation for D2, D1 and D0 see D2 has 2 ones and the mid terms are like that. And you know that it differs since I have assigned this as 01, 00, 11 it is logically adjacent. And when you group it all rest are same and when you play with logic adjacent see theorem. Those Q0 literal will go off from the resulting kind of 5 literal product term okay.

So, that is how it happen, so this can be applied to the more number of states. But there is no fun in trying to do that with the 3 states should be n powers of 2. Because we can eliminate if there are 4 kind of logically adjacent assignment. Then we can remove 2 literals, then if it is 8, then we can remove 3 literals and so on. Btu these are as you go higher the chances are less. But definitely this like 2 states producing the same next state for same input condition is a possibility. So, that is what is shown here.

**(Refer Slide Time: 06:17)**



The slide is titled "NSL Minimization" and is numbered "19". It contains a list of three bullet points:

- Under the same input conditions, if states  $S_i$  and  $S_j$  transit to same next state  $S_k$ , make state assignment such that the states  $S_i$  and  $S_j$  are logically adjacent.
- Applicable to more than two states (by powers of 2)
- **Note:** States  $S_i$  and  $S_j$  may transit to different next states under different input conditions. But, if both transit to just one next state under same input conditions, is good enough to make them adjacent, as the equations for that state get minimized.

At the bottom of the slide, there are three logos: NPTEL (National Programme on Technology Enhanced Learning), JEE (Joint Entrance Examination), and a circular logo with a book and a lamp.

And now throws up a, and that is what formally stated and I have told that even if like there is a situation. Suppose these are not same for D2 if both are 1, it like if you have exhausted all possibilities and nothing is working out. Even in this case suppose these are different even D2 is 1 and if we can logically adjacent kind of assignment at least for the D2 the equation will be kind of area will be minimise.



Because you should know that we are not talking a single kind of logical circuit we are talking about a logic circuit for D2, D1 and D0 which comprises together it comprises the next state logic circuit okay.

**(Refer Slide Time: 07:09)**

OL Minimization 20

---

Inputs			Present State			Outputs		
$I_1$	$I_2$	$I_3$	$Q_2$	$Q_1$	$Q_0$	$O_2$	$O_1$	$O_0$
1	1	0	0	1	0	1	0	1
1	1	0	0	1	1	1	0	1


Kuruvilla Varghese


Now this kind of proms as how to the output logic minimisation heuristic which say for a Moore kind of output. We say, if there are  $S_i$ ,  $S_j$  transit to same  $S_k$  sorry. It produces same output okay, because the output is decode of the present state, then this being same, if you can make this  $S_i$  and  $S_j$  logically adjacent, which is an example as shown here 010, 011.

So, when you work out minimisation for  $O_2$  then we see that the  $Q_0$  is removed okay. So, and this can be applied to the powers of 2 like 2, 4, 8s and so on. And in the worst case even if it is okay for 1 output it works okay. So, that can be done and I am not sure whether the tools do it. But then you can try it even manually, but at least if the tools offers such a thing you can choose some kind of options while synthesising or write some attributes in the VHDL code to be able to do that by the tool okay.

**(Refer Slide Time: 08:22)**

- (Under the same input conditions), if states  $S_i$  and  $S_j$  produces the same outputs, make state assignment such that the states  $S_i$  and  $S_j$  are logically adjacent.
- Applicable to more than two states (by powers of 2)
- **Note:** States  $S_i$  and  $S_j$  may produce different outputs (under different input conditions). But, if both produces one output same (under same input conditions), is good enough to make them adjacent, as the equations for that output get minimized.



So, that is a formal statement.

**(Refer Slide Time: 08:24)**

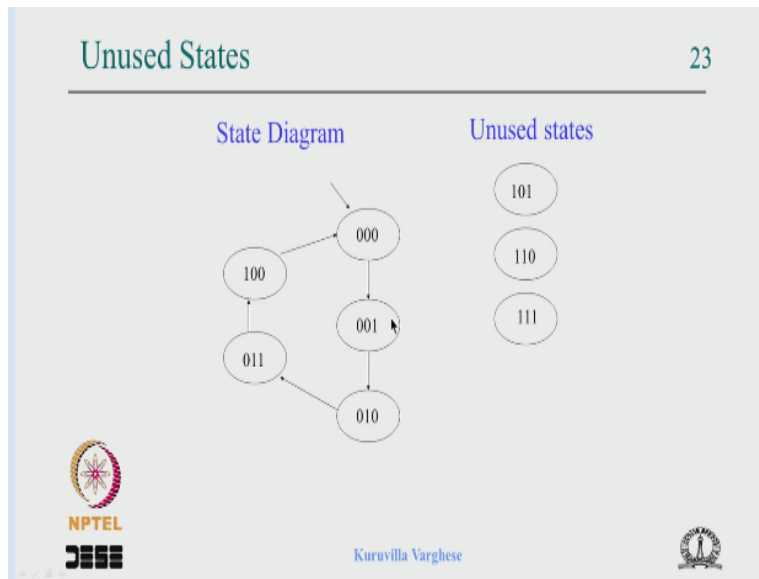
- Number of states =  $s$
- Number of flip-flops =  $n = \lceil \log_2 s \rceil$
- Unused states  $2^n - s$
- $s = 5, n = 3$
- Unused states =  $2^3 - 5 = 3$



Then you have looked at this the problem of unused state like you have  $S$  states and  $n$  flip-flops okay. And now when you take  $n$  the  $\log s$  to the base 2 could be a fraction. And then we take a ceiling, so there are some kind of states which is kind of left which not required here. Because we has taking a ceiling we are taking highest integer from this. And so we have to raise  $n - s$  number of states which is unused.

And example is shown here, like, if number of states have 5, then we will have to use 3 flip-flops for binary encoding. Then there are  $2^3 - 5$ , 3 unused state.

(Refer Slide Time: 09:10)



And then we said that suppose this is the state assignment we have use 0, 1, 2, 3 and 4. There are unused state 5, 6 and 7, now how to handle this unused what to do with unused state.

(Refer Slide Time: 09:25)

Unused States 24

- What happens if FSM get in to these states ?
  - It could get stuck there
  - It could loop through some or all of unused states
  - It could get back to a valid/used state.
- If these states produces some outputs ?
- On what conditions above happens ?

NPTEL  
JEE

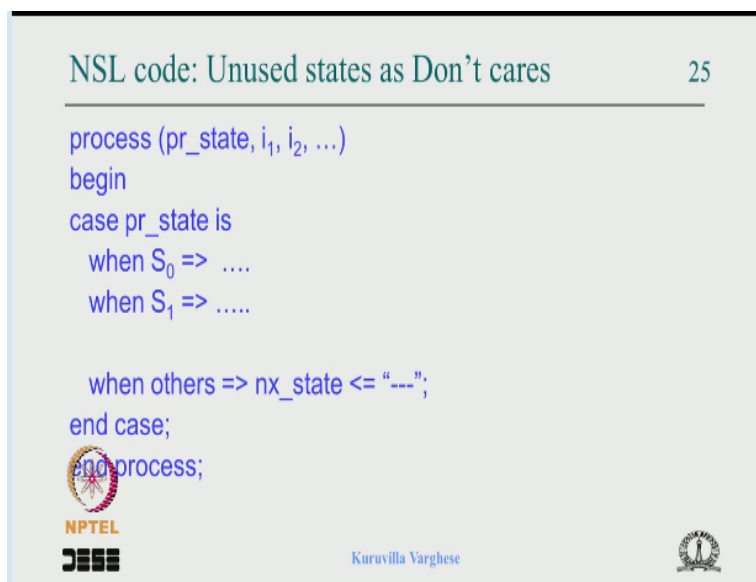
Kuruvilla Varghese

And what we said that by chance if some flip-flop change state say here we took an example. Suppose the state machine was in this state and it by like Q2 as flipped, it is bit by noise, then it gets in here what happens. So, at least like we can list out all possibilities one thing is that we can hope there it get back somewhere okay, maybe it is 101, it get packed 100 by a flip or whatever.

You can you know you can think of most probable case in a 1 bit you know flipping and getting by to 100 or 001, if it get back to 001 all the more better or it could gets struck there, it could loop through all are some of the states and so on. But the dangerous thing is that suppose it is looping through some states here. And it produces some random output which suppose we have not taken care in the design.

Then it could be very pretty disasters and that is, so, let us talk about the fault tolerance. And we have seen an example where it can happens.

**(Refer Slide Time: 10:45)**



```
NSL code: Unused states as Don't cares 25
process (pr_state, i1, i2, ...)
begin
case pr_state is
when S0 => ....
when S1 => .....

when others => nx_state <= "---";
end case;
end process;
```

The slide contains VHDL code for a Next State Logic (NSL) process. The code defines a process with inputs pr\_state, i<sub>1</sub>, and i<sub>2</sub>. It uses a case statement to handle different states: S<sub>0</sub> and S<sub>1</sub>. For any other state (others), the next state (nx\_state) is set to a don't care value ("---"). The slide also features the NPTEL logo, the name Kuruvilla Varghese, and a small circular logo in the bottom right corner.

Suppose we write the next state logic VHDL code like that. And we say case present state is and say when S<sub>0</sub>, when S<sub>1</sub> we say up to S<sub>4</sub> up to fifth state and for when others we assume that you know let us minimise the next state logic kind of area. And we said next state is do not care okay. This is the do not care of the standard logic. And then naturally the synthesis tool is going to consider se states you know.

Like 5, 6 and 7 the way it once you know it will be treated as, as for as the next state logic is concern, this could be treated as 1 or 0 or 1 like that you know. And the way the synthesis tool take it 0 or 1 decide what happens once it reaches there okay.

**(Refer Slide Time: 11:53)**



Inputs			Present State			Next State		
I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	x	x	0	0	0	0	0	1
x	x	1	1	0	0	0	0	0
x	x	x	1	0	1	X(1)	X(0)	X(1)



So, we have taken an example the next state. Suppose the synthesis tool is working at this problem and 0, 1 up to 5 is define properly and 5 sorry 0 up to 4, and 5, 6, 7 is not define and assume that 101 we said irrespective of the input condition we have not specified, it is a do not care. So, suppose for D2 it is treated as 1, D1, 0, D0 1, then what happens is that by mistake like by some kind of error the state machine get it to this state. The next state as by this table is 101, so it will get stuck there okay.

**(Refer Slide Time: 12:40)**

Inputs			Present State			Next State		
I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	x	x	0	0	0	0	0	1
x	x	1	1	0	0	0	0	0
x	x	x	1	0	1	X(1)	X(1)	X(0)
x	x	x	1	1	0	X(1)	X(1)	X(1)
x	x	x	1	1	1	X(1)	X(0)	X(1)



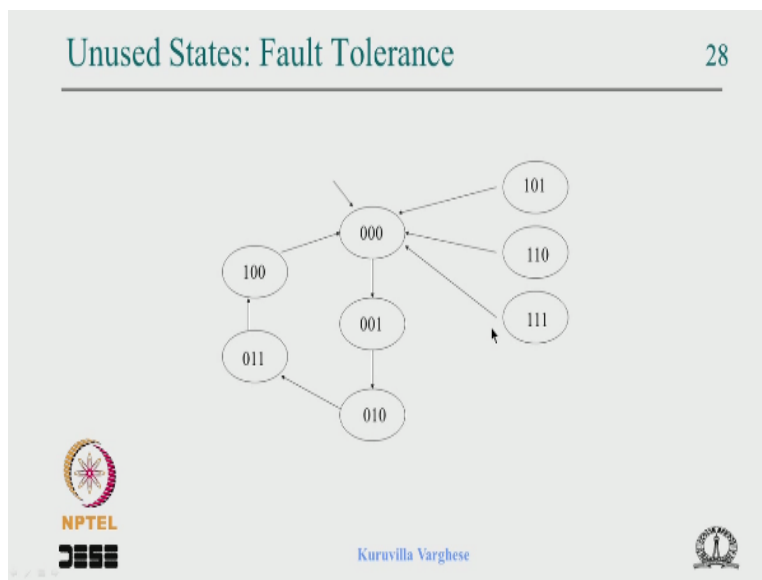
Similarly in the case of take this case you have states 5, 6, 7 we said do not care. And the synthesis tool treat it as 110 for this. And 110 is 111 and 111 is 101, then you can see that it goes to 101, then next state is 110. So, it comes here, then next state is 111 it comes here. Next state is

101 so, it looks through that and we are not even look at what is the output okay, output the way you write. You know, you might say that during this state.

The output is some state 1 when others you say it is 0 so, for all that may be during this 101110111 all the output may be 0 most of the time it will end up like that. And that could be kind of it the way the active logic is kind of defined it could be dangerous. Because there could be active low signal and which could like you have a may be reset bar or enable bar or write bar. Then you will find that signal is asserted.

And something happens okay and depending on the application it could be diastases so, we said the solution. So, it is not a good idea to leave it as a do not care or not specify.

**(Refer Slide Time: 14:04)**



We have to bring if it gets into unused state, we have to bring it back to some valued state you know that is important now when I say valued state most of the time at least in we have to in the picture, I have to show a some state I cannot there is no it is very illogical form it show the arrows here or here so, I show it at the beginning, this need not be true, because this is a kind of the power on initialisation stage.

And blindly we cannot bring it here okay so, that you should remember unused state should be brought back to as safe state. It could be the state.

(Refer Slide Time: 14:53)



NSL Block coding: Fault tolerance 29

---


```
process (pr_state, i1, i2, ...)  
begin  
case pr_state is  
  when S0 => ....  
  when S1 => .....
```

when others => nx\_state <= S<sub>0</sub>;

```
end case;  
end process;
```



Kuruvilla Varghese





But need not be okay so, like in the code when you say when others you should say next state is something some safe state.

(Refer Slide Time: 15:03)


Unused states 30

---

- Introduce transitions from unused states to a Safe state.
- Safe state could be Init state.
- The safe state depends on the application, could be something other than Init state.



Kuruvilla Varghese



It need not be any state you have to look at the application okay, do not like the problem with all these as that you by habit you put an arrow like that. And for hour you will be putting an arrow like something habit bills and people put an arrow happily without ever thinking about it. And or just write when others next state is starting state and you think you have says I mean it can be disasters again depending on the application.

Because there could be this could be a very long drawn control of a some huge machinery which is moving say I am may be exaggerating but I always whenever I see this picture I see these space division they have a launching rocket you know, And normally it gets assemble in one place, and to the launching pad it moves you know kind of over a trolley we takes may be half a day or one day to reach there it goes all the way slowly.

Now as I said I am exaggerating assume that this is a state machine controlling that movement, and by say it has moved 80% percent and 20% is remaining say half a day is remaining. And that state machine gets into this unused state and you just bring it back to the original. And that poor trolley which took a may be almost a day to reach here goes all the back there. You can imagine much dangerous example like this is definitely an exaggeration.

But like you can take medical where human safety is involve , and I suggest that you be very kind of serious about as an engineers very serious about what you are designing okay, many people drawn kind of do not think too much about it. But then I suggest that you are working in a team, you gather enough information according to the policy of company. There may be you have working on a confidential thing which the company does not want everybody to know every detail of it.

But at least know what critical application you have working on and it take enough precaution. Because even for senior group leaders it is wise to disseminate the required information to the engineers. So, that they can make a judgement in such scenarios some time the designers only know these issues in detail not the team manager some time. So, you have to take care of that.

**(Refer Slide Time: 17:53)**

- One can think of redundancy for present state, but, that if there is a mismatch to decide on which one is correct may require majority voting
- Another way is to use error correcting codes (hamming codes) for states, and use maximum likelihood decoding for deciding on correct states, in case of wrong transition



So, that is what is written here and definitely there could be you might ask why not suppose the state machine was in this state and by noise it has gone there is there possibility that it can it come back to the same state okay. So, now that is a tricky question you know that it is at least from without extra information. It is not possible because this state is something which is kind of memorise till the transition.

Once it transit we are not storing the last state okay we are kind of we are deriving the next state from the current state okay. But we are not like you cannot travel back you know we have no logic to work backwards okay, otherwise you should keep a copy of like the next thing to think is that you keep another register you make a copy. But then the issue comes by noise you know it gets here how are you sure, whether copy is wrong or this is wrong okay.

So, it prompts that if you are adding redundancy then you have to at least 2 copies, and you have to take a majority vote like you have the state machine. So, you have 3 kind of state transition happening, and if something goes wrong you take 2 out of 3 voting and do it there is another interesting possibility that you can use error correcting code for the state assignment that means that.

You have to add the redundancy to the state it means that in this case suppose we add 2 extra bit, parity bits. And assume that there is a distance between these valid state okay like a if you have

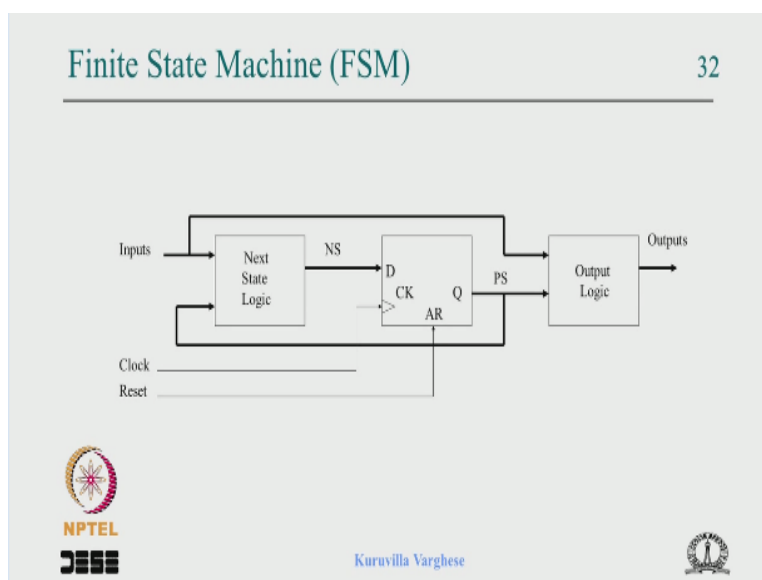
learned about error correcting codes, the hamming code and hamming distance and all that at least say 2 bit distance. And if the 1 bit flips, then you can decode the maximum likely would decoding which is a nearest neighbour decoding and all that.

If you are learn some communication you will be clear about it, but this could be done. But this is at the cost of kind of extra parity bits, extra bits and in the case of error doing the decoding and all that, you know that extra hardware is required so, which essentially talk about the error probability. So, you have to really kind of determine what is probability of error, what is the mean time between failure of the state machine with regards to the unused state.

And if it is not a very large number, then you have to take a kind of call on implementing this error correcting code and all that, it definitely should follow some analysis some reasoning blindly. You should not kind of add this redundancy wasting area you know may be increasing the delay and so on. So, that is my kind of colon to fault tolerance, but it is always safe. When you say when others next state is in a int state.

At least for this do not blindly write the init state you know analyse application is there another possibility, another probable state where it can be brought back should be thought about. So, that is my advice on that.

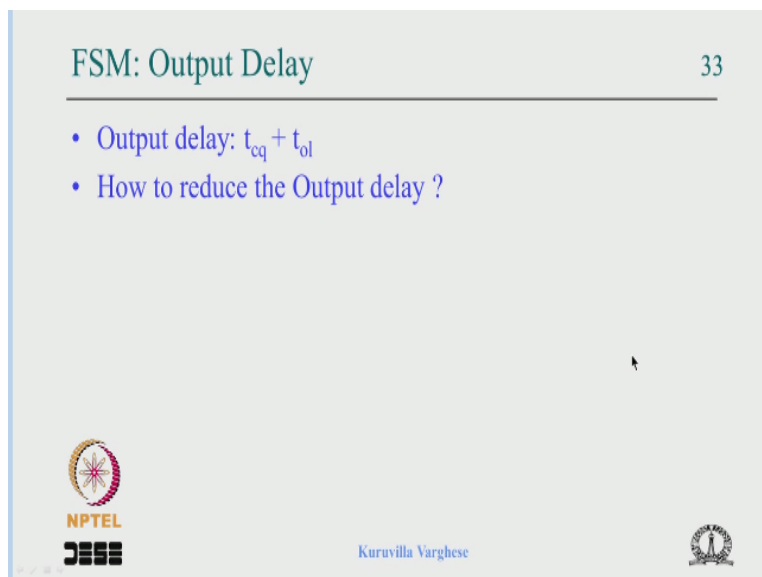
**(Refer Slide Time: 21:44)**



So, let us move back to back to the state machine again so, here you see the output logic let us concentrate on this for a while many a times you find that say it may happen that this is quite fast next state logic is fast. That means it is able to kind of make the state transition quickly, but this delay is huge compared with this that means that you are slowing to down to the clock to match this output logic delay.

Because output has to reach at the output so, or it may happen that there is an event and with regard to the event there is a state change. And you have producing the output when the state change happens you know that there is a  $t_{cq}$  delay from here to here. Then there is a  $T$  output logic delay. So, in response to some input, it might take a 1 clock period for the state to change. Then  $t_{cq}+P$  output logic delay for the output to come. So, at least in some cases this output delay may not be kind of tolerable I mean that maybe high you know.

**(Refer Slide Time: 23:05)**



FSM: Output Delay 33

- Output delay:  $t_{cq} + t_{ol}$
- How to reduce the Output delay ?

NPTEL  
Kuruvilla Varghese

So, the output delay is  $t_{cq}$  or  $t_{cq}+t_{ol}$  you know, that is what is written here. The question is that how can we reduce this output delay if required okay, if required is not that you should try to reduce it. If you have implemented you have looked at the various delays and if it is okay with you go ahead there is no problem okay you do not need to reduce it is but if there is a requirement to reduce it how to reduce is the question.

And let us ponder you can think about it okay. now you are a will to go ahead you know you can when I ask such questions I mean do not fast forward the slide and look at kind of slide when I ask such a question maybe you should pause and think about it okay. So, let us come back to state machine diagram. So, we have a problem here, when the clock comes it takes TCQ time + T output logic time say so much time for the output to be valid okay.

And valid output comes after so much time okay. Now the question is that can we reduce it okay. Now there is there are 2 ways kind of if you look at this block diagram very carefully or you say if you kind of contemplate on this block diagram as some would like put it. So, here you know that the look at the present state, present state is nothing but like upon a clock this next state becomes the present state okay.

So, what happens like you have an input the next state logic is decoded and is ready here, then a clock comes it is transferred here. Then it goes through the output decoding and output is available. So, let us ask this question why not anyway the next state if it is ready instead of waiting for the clock to come and for that to appear here. Let us take the next state and put this output logic here.

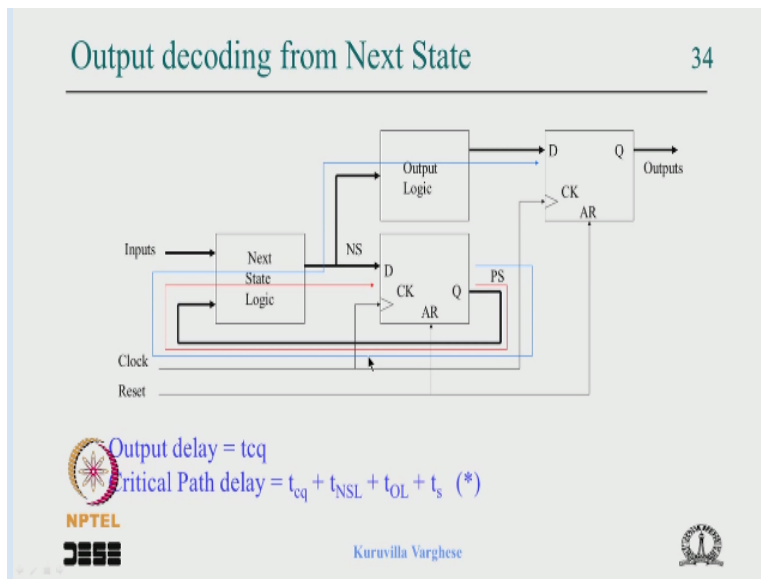
And take the output, we cannot take the output because that we are expecting after the clock. So, let us put 1 flip-flop set of registers at the output, if there are 3 outputs we will put 3 flip-flops clock by the same thing. So, that when the clock comes the state change and output also change and we need the state you know now do not think that the state is not required. Because state is required to kind of decode the next state.

So, this is very much required, but instead of waiting for the next state to appear at the present state and decode. We will put the output logic here, and then we have to put a set of flip-flops at the output, because the state change and the output change okay. So, that is one possibility, So keep that in mind okay. The next possibility is that is 1 good possibility the next possibility is that think that are say 5 states.



So, we have 3 flip-flops okay, so Q2, Q1, Q0 assume there are 3 outputs maybe some right bar enable latch or something like that can we encode this output in Q2, Q1, Q0 directly okay. That means whether we can take Q2 as enable Q1 as right bar Q0 as latch or something like that okay. Then it is a great thing, because then we need any logic this is removed and the delay is TCQ okay. Now the earlier approach to also is the delay is TCQ because upon the clock. This output comes out of the register okay. So, these are the 2 probable method.

**(Refer Slide Time: 27:29)**



And now let us look at the diagram so, this I have written, so this decoding from the next state looks like this you have initially we use to put the output logic here. Now we put output logic here, next at the output of the next state logic and we put the registers for the output, and which is clock by this same clock. Because it has to the output change has to synchronise with state change okay.

When this next state for which the output is decoded becomes the present state. That output with regard to that next state is available at the output okay. Now there is an issue look at the register to register path. Because to decide on this clock frequency we need to consider now 2 path one is this, this red line. That is from the state flip-flop Q2, Q1, Q0 to D2, D1, D0 and there are 3 into 3, 9 possibilities like Q2 to D2, Q2 to D1 and so on okay.

So, 9 paths are there, so we have to say  $TCQ + t_{\text{next state logic}} + t_{\text{setup}}$  is a clock period, but you look at the other path where from this state flip-flop through the next state logic through the output logic it reaches this output flip-flops. So, you will have  $TCQ, T_{\text{next state logic}}, T_{\text{output logic}}$  and setup time okay. So, this is like you will be asking okay fine the output delay is registered.

I mean **re** reduce but what about this clock period, now the clock period might take a pattern, because it could be more. But mind you there could be some kind of conciliation, because next state logic is a combination circuit, the output logic is a combination circuit. Now it is possible to put it together and minimise it okay, now instead of decoding from the input and the present state, the next state and then decoding the output logic.

We could what I am mean this that you can think of this is a single circuit where the output is decoded directly from the input and present state. So, if the tool is able to flatten it you know put as a contiguous kind of circuit and minimise it, maybe this next state logic + output logic delay might be comparable to this original next state logic delay. Then we have done it nothing is **is** lost and the output delay is reduce okay.

So, that is kind of one way of dealing with output delay. Now mind you what with regard to VHDL coding is required is that we use to write at the beginning like suppose if you are writing a process we will write case present state is when S0, what are the outputs which are like 1 and 0 we write. So, here instead of saying case present state is, because it the output logic was here.

We say case next state is and then that will generate the output and then you can register it or we can write this register and this logic together in a process. Because we have seen that a registers with the proceeding combinational logic can be return in kind of process okay. So, you can say if reset is 1, then like all the outputs are 0 one by one. Then you say else if rising edge of clock.

Then you say case next state is, so this will be taken care okay. So, that is with regard to coding it not a very difficult thing to code straight forward. So, it is the matter of coding it, some change in

the coding then you get it not much of a detail it is a minor detail okay. So, the critical path delay is  $t_{CQ} + t_{NSL} + t_s + t_{OL}$ . Now if you look at both the delays there is no much difference.



Like we have to consider from the previous clock delay okay, the output okay there to compare equally, that means that like in this a clock comes then  $t_{CQ}$ , T next state logic, then T setup then again  $t_{CQ}$  and T output logic. I mean that is how because we are now decoding the output from the next state. So, we have to work from previous clock period. Then only it can be compared.

**(Refer Slide Time: 32:34)**

### Output decoding from Next State 36

---

- Compared to earlier case; from previous clock edge, total delay would be  $t_{CQ} + t_{NSL} + t_s + t_{CQ} + t_{OL}$
- In the case of decoding output from Next State, delay would be  $t_{CQ} + t_{NSL} + t_{OL} + t_s + t_{CQ}$
- But, Since NSL and OL are together, synthesis tool can optimize (minimize) them combined and may result in less delay than  $t_{NSL} + t_{OL}$

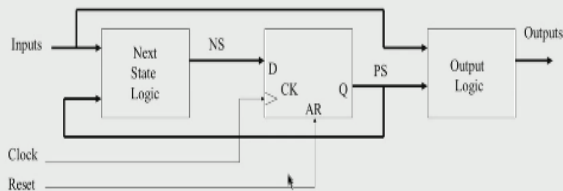

Kuruvilla Varghese


But if you compare that like a like you have in the in this case you have.



**(Refer Slide Time: 32:36)**

### Output decoding from Next State 35

---



- In the normal case, we would have chosen clock period as  $t_{CQ} + t_{NSL} + t_s$  with some margin

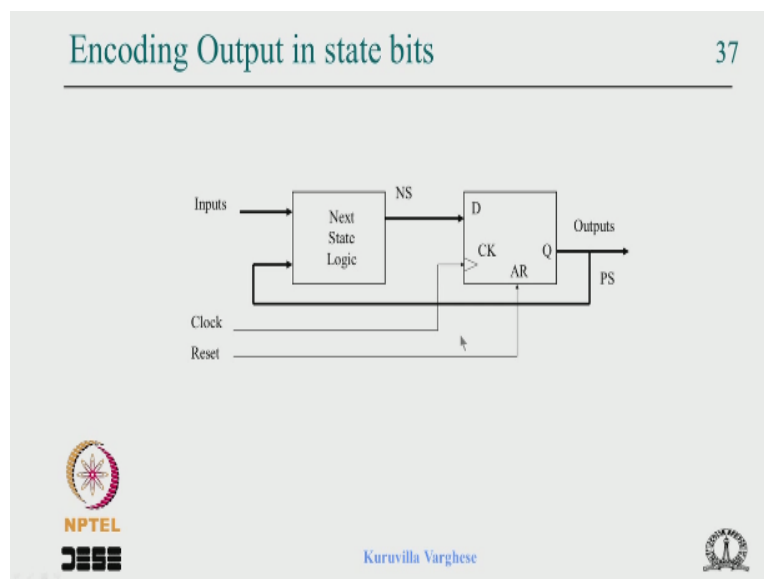

Kuruvilla Varghese


$t_{cq} + t_{NSL} + t_{setup} + t_{cq} + t_{OL}$  okay, that is a case, that what is written here, but if you look at this the same thing.  $t_{cq} + t_{NSL} + t_{OL} + t_{setup} + t_{cq}$ , so only the order is different from the previous clock edge you know the total delay is same. But after clock edge this will give you only just one  $t_{cq}$  you know. That is the like earlier the boundary was here. So, you like after clock edge you have get  $t_{cq} + t_{OL}$ .

But here after clock edge only  $t_{cq}$  comes because  $t_{OL}$  goes before that okay. So, but we our only hope is our consolation is that, when is both are minimise together. This  $t_{NSL} + t_{OL}$  could be greater than this  $t_{NSL} + t_{OL}$ , because that can get minimise okay. This cannot get minimise because there is a register boundary coming in between, there is a next state logic. Then there is a flip-flop here and the output logic.

But here if you look at we have moved the output logic before so, between one register and the next register. You have both together and that can minimise okay so, from this kind of expression itself for a clever student you can make out what is happening because I have written these terms in that particular order okay. So, that tells you what is the advantage.

**(Refer Slide Time: 34:09)**



So, that is about output decoding from next state so, just a summary we have started with a unused state and we said that for fault tolerance that you have to bring the state machine to a safe

state this is a good practice for irrespective of any design do bring from the unused state make a transition to a safe state. Then we have looked at the problem of reducing the output delay.

And we said there are two ways one is the decoding the output from the next state logic than the present state. But you have to register the output 2b kind of coincident handle with the state change. And we have seen that from a previous clock edge point of view. There is no difference we are playing with we are moving some logic which was after a register to the before the register but in a effect you have the output delay is reduced from  $t_{cq} + t_{OL}$  to  $t_{cq}$  alone.

But now the two combination circuit is in within between two registers and that can be optimise in we are hoping that the clock period may not be affected so much. You know that is cracks of it now less let us look at next technique where the output is encoded in the present state. And if you remember block diagram, if you remember the lecture.

We have started the seriously not the introduction not a course policy and all that when we started disclose the first lecture. We have learned about the synchronous circuit as a kind of structure of the sequential circuit. This is the kind of the structure of a synchronous counter okay. But the counter may not be counting in a strict sequence you know that you should remember.

So, you can imagine how we are going back to a counter which is a may be a crazy count it may go through it may go through all kinds of sequences depending on the input. But the count output all the state output itself is used as a output. And may be at the outset you might think that this is not going to happen. This is not possible okay mind you it is more probable than you think. Because the reason is that we have we are not worried about particular numerical value of the state.

You said the when we develop a state diagram, we work with symbolic state. We say  $s_0$ ,  $s_1$ ,  $s_2$  and so on. Then we assign it to reduce some area and all that so, the same technique can be applied, because we have some two output and two flip-flops say we like we have the freedom of choosing the state. So, looking at the output pattern requirement, if you go back and do the state assignment, then it is possible that this might work okay.

So, that is the moment I say that should kind of trigger some light in your brain okay. So, let us get back to the diagram so, here addition to that okay. Now let us look at it may be we take an example say take these example okay so, here the output delay know not out it is a tcq because as soon as the state changes the next state, present state gets next state.

And the output is same as present state o, it is valued so, the output delay is tcq exactly same as the previous technique.

**(Refer Slide Time: 38:34)**

**Encoding Output in state bits** 38

States	Outputs	
	WR/	EN
$S_0$	0	1
$S_1$	1	0
$S_2$	1	1
$S_3$	0	0
	$Q_1$	$Q_0$

Since output patterns are unique and equal to number of states, state variables can be used as outputs

Kuruvilla Varghese

And assume that in our case we are four states  $s_0, s_1, s_2, s_3$  and there are two outputs assume again now this is a definitely a you will acquire me of cooking it up yes I have kind of cooked it up. But this these are the kind of possibilities like you have two outputs right bar and enable, and if you care to look at it  $s_0$  outputs are 01. 1011 and 00 okay so, these are unique pattern 011011 and 00 so, why not to the state  $s_0$ .

We assign 01 like  $q_1$  gets 0,  $q_0$  get 1 and  $s_1$  10, and  $s_2$  11,  $s_3$  and 00 that means that straight away the  $q_1$  can be used as right bar. And  $q_0$  can be used as enable so, many a times you do not even look at the possibility. It may happen nowadays what happens is that you do such a kind of suppose say the pattern was 00011011 sequentially and we have use the numerated data type.

Then internally synthesis tool has done as sequential assignment we have seen that more synthesis tool will do a sequential assignment which is nothing but 00011011 and automatically match you will find that when the equation of the output is derived right bar will be you will find that that is Q1. But here it is not such a kind of nice kind of matching.

Like s0 is not 00, s0 is 01, so synthesis tool may not do though it is very simple, synthesis tool may not do that trick, but you we have looked at the kind of changing the state assignment by some attribute like state encoding attribute, enum encoding attribute and so on. So, you have used enum encoding and assign this state, then you get you know without any output logic to output.

So, this even compare to the previous technique, this is a very kind of possible technique which you should look at it okay. So, I have told you the reason why it works okay. Because we do not care about the state assignment, so we can do that state assignment maybe in may not minimise the next state logic or output logic, output logic definitely because there is no output logic at all okay.

Now this is quite neat still you will accuse me of kind of you know manipulating it to look very nice possibility yes I agree that. Now what we have done that there are 2 outputs, there are 2 rise to 2 possible patterns okay. And so 4 possible pattern and we have exactly 4 states and these each pattern is not repeating it is unique. So, everything works out very nicely for us what if like in this case what has what would have happen.

If say s2 the same 01 appears know, then this there is a repeat so, that no unique output pattern to distinguish the state okay. So, that is a possibility.

**(Refer Slide Time: 42:36)**

States	Outputs	
	WR/	EN
S <sub>0</sub>	0	1
S <sub>1</sub>	1	0
S <sub>2</sub>	1	1
S <sub>3</sub>	1	0
	Q <sub>1</sub>	Q <sub>0</sub>

For states S<sub>1</sub> and S<sub>3</sub> outputs are same and hence one extra bit is needed for state variables.



Like we have like this case you look at this case here the same scenario where the output pattern is different. In state s1 the output is 10, s0 it is 01, s2 is 11. But when it comes to s3 it is again 10 okay. So, there is no way now to use Q1 as right bar, because these 2 states are kind of identical. But we can do one thing, we can add an extra flip-flop okay. So, actually to encode 4 states we need only 2 flip-flops.

But for us to reduce output delay what we do is that we introduce one more flip-flop for the present state, no harm okay absolutely no harm. So, that is what we are going to do it.

**(Refer Slide Time: 43:27)**

States	Outputs		Extra bit
	WR/	EN	
S <sub>0</sub>	0	1	0
S <sub>1</sub>	1	0	0
S <sub>2</sub>	1	1	0
S <sub>3</sub>	1	0	1
	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>

Adding the extra bit makes unique pattern and state variables can be used as outputs.





We introduce like Q2 there were only 2 flip-flops, we introduce an additional flip-flop and where this pattern was repeating. We made a difference by making this extra 0 in 1 one of the state where the output is repeating and in the other state where the same output pattern is a we put it 1. So, now you look this is 100, this is 101 rest of the places you can make it 0 or 1 it does not matter, so we have made it 0 there no particular reason.

But yeah you can make it 1 also like this is something you should know kind of there is no great reason for to choose 0 or 1 like as a pure logical exercise, but there maybe some (()) (44:24) with the regard to the circuit. Because if you think of an NMOS circuit if the gate is 0, it is off, gate is 1 it is so. Like there could be some differences something is made active.

The current is sinking, so you should think about those scenario. But then what I am saying that we can safely assign the default bits as 0. Now what we do is that we say state 0 is 010, state 1 is 100, state 2 is 110, state 3 is 101 okay. Now like then we use Q2 as right bar, Q1 enable and Q0 we do not use it, Q0 is a part of the present state. Because we have added this to make this output unique like .

Like to for this repeated pattern to come in this 2 different state we have added extra bit. So, but you might ask now you have 3 flip-flops and there are 8 possibilities and we have only 4 valid states, there is unused state. That has to be brought back and that need some decoding all kinds of thing okay. And that may have some reprocreation. But if it is going to all 00 it may not add to the circuit so on there are.

But you **you** think about it what are possibilities. So, that is a game in encoding output in **in** state bits you do not have to pest like you do not have to really worry about like a number of outputs and number of state bits and how many patterns are repeating like what are the total possibilities which is greater which is less and all that . None of this serious analysis not required what we are looking is that, what is the repeating pattern okay.

So, definitely if number of like you have some 16 states only 2 outputs are there, the no way to kind of like encode all that you know the state bits using 2 patterns. So, you need extra bits, if the

number of outputs like 2, rise to number of outputs are kind of less than the possible state. Then you need anyway extra bit, whether it is repeating or not.

But in generally you can look for the maximum repeat and try to make a difference, because if there are more outputs and less states definitely there is going to be reputation okay. So, you should understand that.

**(Refer Slide Time: 47:05)**

**Encoding Output in state bits** 41

States	Outputs				Extra bits	
	Adr_1	Adr_0	WR/	EN		
S <sub>0</sub>	0	0	1	0	0	0
S <sub>1</sub>	0	1	0	1	0	0
S <sub>2</sub>	0	0	1	0	0	1
S <sub>3</sub>	0	1	0	1	0	1
S <sub>4</sub>	0	0	1	0	1	0
S <sub>5</sub>	1	0	0	1	0	0
S <sub>6</sub>	1	1	0	1	0	0
	Q <sub>5</sub>	Q <sub>4</sub>	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>

NPTEL  
Kuruvilla Varghese

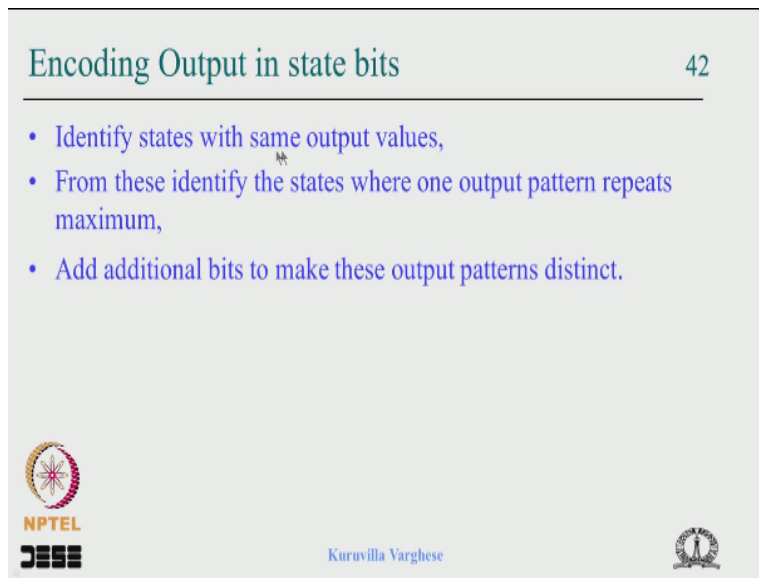
So, let us take an another example, where this is little more kind of complex. So, we have 7 states S<sub>0</sub>, S<sub>1</sub>, S<sub>2</sub>, S<sub>3</sub>, sorry S<sub>4</sub>, S<sub>5</sub> and S<sub>6</sub>, and we have 4 output okay. Now once again I have put 4 outputs, so possible patterns for 2 rise to for 16 which is greater than the number of states. So, like this should be enough okay to encode. But you see there is reputation okay.

So, you see that in state 1 these 4 outputs are 0101 like that and in state 0, 2 and 4 the output pattern is 0010 it is repeating okay. So, we have 3 reputation to make a difference we need 2 bit, extra bits. So, we added that and we have 0 00 here, 01 here, 10 here 2 kind of distinguish between these 3 and between these 2 week and make 00 and 101 rest can be 00. So, we now assign in like Q<sub>5</sub>, Q<sub>4</sub>, Q<sub>3</sub>, Q<sub>2</sub>, Q<sub>1</sub>, Q<sub>0</sub> and these are kind of the state bits.

Like 6 flip-flops frankly we require only 3 flip-flops, but then in this scheme we end up with 6 v 6 flip-flops and possible states are 64. But we get now the output directly encoded in the state bit

like you have Q5 as address 1 Q4 as address 0, Q3 as right bar and Q2 as enable you know. So, this is a good mechanism to reduce the output delay. So, the rule is simply stated.

**(Refer Slide Time: 49:01)**



Encoding Output in state bits 42

- Identify states with same output values,
- From these identify the states where one output pattern repeats maximum,
- Add additional bits to make these output patterns distinct.

NPTEL IIT Bombay Kuruvilla Varghese

Identify the state with same output value from this there could be multiple of such. From these identify the states where one output pattern repeat maximum okay. So, there could be like this case there are 2 states where output pattern is repeated. There are 3 states where the same output pattern is there pickup the 3 okay. Now check how many extra additional bits are required to make a difference make it.

And use the original bits where we started like Q5, Q4, Q3, Q2 as output then we are done. So, that is encoding output in state bits, so quickly running through it like this is the game we are kind of encoding the output in the present state. At the beginning it looks a kind of impossible kind of thing to do. But then we remember that this state assignment is something which we have the control flexibility.

So, why not do the state assignments at that such a match happens, and that is very easy when the possible patterns and the number of states matches. Then you can do straight away there is a reputation then you have take additional bits, and definitely as I said if the number of state bits know 2 to the power of number of state number of outputs are kind of less you need anyway additional bits.

Otherwise if it is greater than we have to look at reputation you can work out that kind of case and then you can do a proper state assignment do encode the output in state bit which reduces TCQ all together it throws the output logic. But definitely you should be asking this question now we are playing with the state assignment what happens if this increases the next state logic delay yes.

These kind of question should be ask analyse I do no think the tools do it. Because the tools many a times choose a kind of fix kind of state assignment with which this cannot be done. But then you can do a like in a at least in simple cases you can do an analysis and try to do this yourself maybe there are the tools, complex very good tools might try to do this you have to check tool manual.

So, that is what is maybe the last FSM issue that I am kind of discussing this lecture . Now we have another issue called synchronisation.

**(Refer Slide Time: 51:55)**

**Metastability in edge triggered Flip-Flop** 43

$t_s$ : Setup time: Minimum time input must be valid before the active clock edge  
 $t_h$ : Hold time: Minimum time input must be valid after the active clock edge  
 $t_{co}$ : Propagation delay for input to appear at the output from active clock edge

NPTEL  
JEE

Kuruvilla Varghese

So, that maybe we will complete that in the course of the next lecture and then we have to do we will maybe we will look at the devices called programmable logic devices, then we kind of play with a tool before getting into maybe FPGA we have to learn the test bench to be able to use a

tool. Because we have I think we have learned enough to write the VHDL code for data path and the state machine.

Because you know all about the coding the combinational circuit various syntax what it means how the simulation work how the concurrency simulated just for the your understanding. And we have looked at detail about the sequential element, sequential circuit, registers how to code using various construct what it means, and then we have looked at in particular about the finite state machine as a separate lecture we have handled it.

So, that we have good grip on that, so this is good enough for synthesis. But to be able to do a good development then we have to write the test bench. So, I am planning to cover the test bench maybe I will in the next few lectures I will try to cover the programmable logic devices. These are being used less and less now a days not too much application. But from an academic point of view maybe it is wise to look at it.

It is a nice architecture but it does not scale to accommodate kind of complex circuit and it also has some certain problems like lack of memory lack of registers and so on. But for certain application it is good as an academic exercise to look at it architecture is a good point. Because it tells how that architecture is evolved. And the thinking process behind it is good.

So, we will look at it. I do not teach that in my regular course because that is not being used very much. So, that I can avoid, but then the course name itself had PLD at the beginning when I gave the syllabus sometime back. So, I was stick with it, I will cover it maybe it will be definitely academically useful though maybe application wise it is not you may not kind of end up using it.

I do not know what happens after few years if you are kind of this is being recorded in 2013 what happens if you are listening to this lectures in 2015 I do not know whether that PLDs will be useful. But in anyway as I said I will be covering the kind of synchronisation in a one lecture or so. Then PLDs then the test benches, then we can look at the FPGAs and a case study and play with a tool to wind it upon.

That will be a kind of good conclusion. So, I think today we have looked at basically unused state transiting back to the safe state. Then we have looked at a that is for fault tolerance then we have looked at the techniques to reduce output delay. One was decoding the output from the next state and registering it. And we have seen that analysis wise there is no kind of difference in terms of the terms involved.

But there is that within the register boundary the output logic and next state logic is coming together the minimisation can happen maybe the total critical path delay can come down . Second thing was that the decoding the output in the present state though it looks kind of a less probable situation we have found that it is quite useful thing to do. And we have looked at the technique the algorithm for it.

So, in the next lecture I am going to handle problem of synchronisation very important thing. As I said I cannot do all analysis come out with various the expression for fault tolerance and all that. We cannot go in depth. I cannot go into the flip-flop. This is something do with the flip-flop. I cannot go it normally in the course I go into the flip-flop try to analyse bring out this issue why it happens a deep understanding from the that point of view.

But then if you accept that problem we will see at least how to handle it most simple and most used techniques we will look at it without that I feel the course would not be complete. So, we will look at it and, so I hope things are coming to the end of the course, please revise, otherwise when we kind of tying everything together at the end, you will be at lost.

So, have reach the FPGA, then we will be kind of putting everything together. We will be putting the digital system basics will be putting VHDL, the FPGAs, the tool maybe some board exercise everything together. So, unless you have good grip it will be difficult to follow please go back and revise learn it well. And I wish you all the best and thank you.