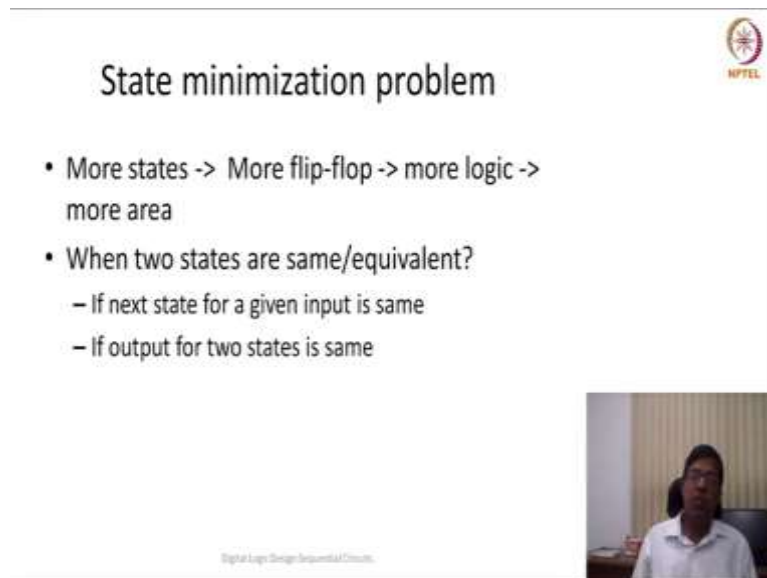


**Digital System Design**  
**Professor. Neeraj Goel**  
**Department of Computer Science Engineering**  
**Indian Institute of Technology, Ropar**  
**Lecture No. 54**  
**State Machine Reduction**

Hello everyone, in our previous lecture, we have seen different kinds of state machines, and how to design those patterns using state machines. How to detect patterns using state machines? Now, one problem which we observed in during one of the examples that what if we keep, we design a state machine, which has redundant states.

So, to avoid a redundant states one method could be that we design it very systematically we think of all the combinations and then the solution, whatever we come up with is already a minimized and optimized one. Now, is there a possibility that, if we come up with a method, if we come with the states, which are un-optimized and can we can we optimize them later?

(Refer Slide Time: 01:17)



The slide is titled "State minimization problem" and features the NPTEL logo in the top right corner. It contains the following bullet points:

- More states -> More flip-flop -> more logic -> more area
- When two states are same/equivalent?
  - If next state for a given input is same
  - If output for two states is same

In the bottom right corner of the slide, there is a small video inset showing a person speaking. At the bottom center of the slide, the text "Digital Logic Design Department, IIT Ropar" is visible.

So, then the first question also comes in our mind that, why do we would like to optimize or why we would like to reduce the number of states. So, reduction in states is directly related to implementation. So, if you see that the more number of states are there, more number of flip flops are required to store that state. And also if more number of states are there the next state logic depends on the previous state. If the number of states are more, then the logic will also be more so, that means more gates would be required.

And if more gates are required, then of course, more area would be required. And further, there is a good possibility that it would also have an impact on latency and delays. So, it is


always a good idea that if we can reduce, if we can optimize or minimize the number of states, then it is always good. But how to do that? So, if we want to reduce the number of state, then we have to say that some states are definitely there are some states, which are actually equivalent to some of the existing states.

How do we know that two states are same or actually represent, could be represented in one state? So, one idea could be that if a particular state, given any input, given any input, the two states will have the same transitions, and his same outputs. So, let us say, yeah, if they have same transition, they have same output. So, that means those two states are equivalent, and which also means that they are one of them is redundant.

So, let us try to understand this particular statement with one of the examples which you had done yesterday.


(Refer Slide Time: 03:01)

**Example**



Input sequence	Present state	Next state x=0	Next state x=1	Output x=0	Output x=1
Reset	A	B	C	0	0
0	B	D	E	0	0
1	C	F	G	0	0
00	D	H	I	0	0
01	E	J	K	0	0
10	F	L	M	0	0
11	G	N	P	0	0
000	H	A	A	0	0
001	I	A	A	0	0
010	J	A	A	0	1
011	K	A	A	0	0
100	L	A	A	0	1
101	M	A	A	0	0
110	N	A	A	0	0
111	P	A	A	0	0

Sequence Detector  
4 disjoint bits  
Pattern:  
1001 and  
1010



So, yesterday in our previous lecture, we had done one of the sequence detector with 4 which was taking 4 disjoint bits as a input, so basically group of 4, which are known overlapping as well as disjoint. And the pattern, which we were detecting were 1001 and 1010. So, if the pattern is 1001, then the output is 1 or the pattern is 1010, then the output is 1. And incidentally, the way we have derived it was an optimal state machine, and it was already reduced.

So, let us say, if we try to create in a NAV method, where we are not having any optimization, but we are trying to create all the states, all possible states. So, then this state table would look something like this. So, I will briefly explain this state table so, you

remember that, initially we will start with the reset state. So, a reset state let us say is A. So if present state is A and input was 0, then we will come to B state and if input is 1, then in next state is going to be C, output is always going to be 0.

Now, in case of B, we are assuming that input sequence 0 has already arrived. And so the state, present state is B. Now if the next input is 0, then the state is D and if next input is 1, then the state is E. Similarly, for state number C this is actually, C. So, this is C, so from C, I will correct it. So, if the current state is C, then the next state, if the next input is 0, current input is 0, then it will become F, and otherwise, if input is one that it will become G.

So, in case of state number D, the input pattern 00 has already arrived, and in case of E, 01 pattern has already arrived. So, in case of F, 10 has already arrived and in case of G, 11 has already arrived. And now, from the D, let us say if input is 0, then we can have H state and if input is 1 then I state. Similarly, J, L, M, N and P, so all these possible states are there.

So, what is the significance of H state? H state means 000 has already arrived. Now, if next is 0, then we are going to reset state, and if 1 then also we are going to reset state, but the output in both cases are going to be 0. So similarly, for I state 001 has already arrived. Now, if another 0 will come we will again, we have to go to A state. And similarly, if next is input is 1 then also we have to go to A state, but the output is individual. So, output 1 would be there in case of 0101. So, basically, if the previous information is 010, and the current X equal to 1, then the output is going to be 1.

So, similarly, the other output which is going to be 1 is 1001. So when the present sequence or the past sequence is 100 and the next, the current value is 1, then the output is going to be 1. In all other cases, so this this represent all the 16 scenarios. So, in all other cases, all the 8 scenarios, so in all the 8 scenarios, the output is going to be 0. So it is going to be 1 only in these two cases.

So, this represent an exhaustive state machine, which means that we have written each and every possible state, which could be possible. So, if we can optimize this, then possibly the same method we can use to reduce any number of state. Now, let us see, how do we reduce? So, let us remove this input sequence part, which was showing the significance of each and every state.

(Refer Slide Time: 07:29)

**Example**


Present state	Next state x=0	Next state x=1	Output X=0	Output X=1
A	B	C	0	0
B	D	E	0	0
C	<del>F</del> E	<del>G</del> D	0	0
D	H	+H	0	0
E	<del>I</del> L	<del>K</del> H	0	0
<del>F</del>	L	MH	0	0
<del>G</del>	<del>N</del> H	<del>P</del> H	0	0
H	A	A	0	0
<del>I</del>	A	A	0	0
<del>J</del>	A	A	0	1
<del>K</del>	A	A	0	0
<del>L</del>	A	A	0	1
<del>M</del>	A	A	0	0
<del>N</del>	A	A	0	0
<del>P</del>	A	A	0	0

H=I=K=M=N=P

J=L

E=F

D=G



So, then this this table would look like something like this. So now, if we see this table, then try to see that, what is the difference between state number P and state number N? So, in N state, if input is input is 0, then the next state is A, if input is 1, then also next state is A. And output in both cases input 0, as well as input 1 is 0. So similarly, for the P, if the input is 0, then the next state is A, and if input is 1, then also next year is A. And output for what the case is 0. So, that means P and N should be could be, should we actually the same state. We do not see any difference between these two rows.

Similarly, if we see that there is no difference between P, N, M, and K, and I and H, all of these rows they are all same. So, if they are same, then we can possibly say that all of them are same. So, wherever I K, M, N, P all of them are written, we can replace them with probably H, and we can remove these rows because they are all redundant. So, for removing I am putting cross, so all of these rows would be removed.

Now, similarly, we see that J and L, these two are also looking same, because their next state is same and their outputs are also same. So, I can remove that. So, now I can either choose J or L. So, I incidentally choose L now J. So, we can do the other way also. Now, the next step would be that wherever J is there I have to replace it with L, and wherever I, K, M, N and P is there I have replaced them with H. So, that mean,s wherever I found I, K, M and N, I would replace them with H and similarly if J is there, I will replace it with L.

So, now, if we look it further, then it appears like this, these two rows E and F these two rows are also same. Now E's next state is L when input is 0, and when input is 1 the next state is

H, output in both cases 0. Similarly, for F also if input is 0, then next state is L, and when input is 1 next H. So, that means, these two rows are actually same if they are same, so, that means we can remove one of them.


And similarly, if you see, so we can remove let us say F 1. Now, let us say the other thing is also there. If you see D and G. For D, the next eight is H if input is 0, and if input is 1, then also it is H output in both cases is 0. Similarly, for G also the next series H in both the cases if the input is 0 or input is 1 and output for both the case is 0. So, that means we can remove one of them, let us say we remove G.

So, wherever G was there, wherever F was there, now, we have to replace them with E. So, we have replaced them with E also. And now, so with this, we can find. What is the final table?

(Refer Slide Time: 11:31)

**Final reduced state machine**


Present state	Next state x=0	Next state x=1	Output x=0	Output x=1
A	B	C	0	0
B	D	E	0	0
C	E	D	0	0
D	H	H	0	0
E	L	H	0	0
H	A	A	0	0
L	A	A	0	1



Digital Logic Design (Sequential Circuits)


The final table looks something like this, which has only A B C. So, only seven states are there A B C D E, D E F, F is not there, D E H L. So, these are the only states. So, that means we can reduce what what is he learning from this this whole exercise that what we have done, we see if two rows in a state table is same that means those two states are same.

(Refer Slide Time: 12:08)



## Summary of the approach


- Two states are same: if their next state is same and their outputs are same
- Application of the above rule recursively till no more equivalent states are found



Digital Logic Design: Sequential Circuits


So, can we apply and the second thing, which we have learned that we can apply this particular information, this particular knowledge recursively till there is no further equivalent state found or there is no further removal of state there is no further rows which are same could be found. So, this looks like a simplistic method. And we can find equivalent states and we can reduce our state machines and does not take too much of time also. So, let us try to apply that on some of the example.

(Refer Slide Time: 12:46)



## Implication method

Pr State	NS X=0	NS X=1	Out
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1



Digital Logic Design: Sequential Circuits

So, this is the another example. Now, let us try that can we reduce any of the states? So, if we see here, these two rows are different. Even if you spend couple of more time you can pause

the video for some time and then look at these rows carefully, you will not be able to find any opportunity. So, can we still say that this is an optimized state machine?

So, when we have such a state machine where there is no obvious there is no obvious reason to merge or two or obvious reason to say that these two states are same, and what should we do, so then the method is a little sophisticated, but it is trying to understand what are we trying to say.

We are seeing that these two states A and B, these two states could be same only and only if, if D and F are same C and H are same because outputs are same. Outputs are same, if these two next states are same, then we can say these A and B are also same. So, we can apply this kind of a fundamental relation, and then see the possibility that when two states could be equivalent.

So, if we are able to resolve and then we can, if two states are not same, then we can remove that particular relationship and say that because these two states are not same, so they're actually not same. So, then because of that some other relationship may also be false, and which would keep on distinguishing that if two states are same or not. Whatever is left would mean that these two things are same. So, let us try to understand what I am saying. So, I understand that it is slightly tricky to put in words. So, let us see with the example.

Now, I am putting this table in a corner of my slide, so that we can create this kind of an implication method or a peer-based method. So, in this peer-based method what we are seeing is, we are for each peer for each. Let us say the peer A and B we are trying to see when they could be equivalent. So, in this peer method, we try to have all possibilities. So, for example, for A and B, A is here B is here. So, we see that A and B could be equivalent only if D and F are equivalent and C and H are equivalent. If any of them is not equivalent that means A and B is not equivalent.

Now, similarly, we try to put all the possibilities all the peers and see that whether they could be equivalent or not. Because A is equal to B if B is also equivalent to A so that means that this particular matrix will become lower triangular matrix. There will be not A to A basically this particular cell is not there where we are comparing A to A because A is actually equivalent to A. So, similarly, the upper half of the this matrix is also not there.

Now, let us quickly look at all the cells of this matrix. You see, A and C, if we see A and C, we do not care whether the next states are same or not, but because outputs are different so A

and C is bound to be different. So, that is why we have already put across here that A and C are not same. Now, similarly, we can see for A and D. For A and D, they could be same if A and D are same, and C and E are same. Now, this condition, if A and D are same, and this, this additional condition A and D are same, so we can remove this condition. They would be equivalent if C and E are same.

So, let us again look at A and E. So, if we see A and E then, because their outputs are different, so they are not going to be equivalent. So, similarly, we can see A and F also have different outputs, so they can never be same. Similarly, A and G could be same if D and B are same and C and H are same. So, we have created all the peers. And for every two states, whatever peers are possible we are writing it here.


So let us look at the one, which has little exceptional rows. So, for example, C E you see. So, if you see C E, C, and E, C and E could be same if E and C are same, so that means this particular relation E and C is also redundant, we can remove that, and C and E are same if A and D are same.

Now, let us look at G and D also. So, if we say G and B, F and F are already same, so G and B would be same, G and B would be same. G and A, G and A, G, what will be G and B. So, G and B would be same if F and B are same, because H is already there, which is same. So, that is why B and F is written over here, the second term is not written because that is already found to be same.

So, this may we are yes, of course, it is little pain painstakingly process, but because it is a tabular method, we can do it using computer, so we can simply put things into computer. We can say that whether in first loop, we can find out all the dependencies or all the peers, which if those peers are saying that we for example, D and E are both found to be same, then that means and C and H are both are the same that means A and B could be same and reverse is also true. If any of them is not same that means there is no possibility that A and B could be same.



(Refer Slide Time: 19:31)




## Implication method

b	<del>d-f</del> <del>c-h</del>						
c	x	x					
d	<del>a-d</del> <del>c-e</del>	<del>a-f</del> <del>a-h</del>	x				
e	x	x	<del>c-e</del> <del>a-d</del>	x			
f	x	x	<del>e-f</del> <del>d-g</del>	x	<del>c-f</del> <del>a-g</del>		
g	<del>b-d</del> <del>a-h</del>	<del>b-f</del> <del>a-h</del>	x	<del>a-b</del> <del>e-h</del>	x	x	
h	x	x	<del>d-g</del> <del>d-g</del>	x	<del>a-g</del> <del>a-g</del>	<del>c-f</del> <del>b-g</del>	x
	a	b	c	d	e	f	g

Thus,  $a \equiv d$  and  $c \equiv e$

Digital Logic Design: Sequential Circuits



So, now, let us look at the second possibility and try to see So, I have removed this table so that we can focus on this matrix, and I have also removed A and D because this peer is this particular cell is for comparing A and D and similarly I removed C and D also.

So now, in the first iteration, so first we have created this lower triangular matrix where for each comparison we have found that what could be the equivalence requirement. So, if these L. I am again repeating if these two are same, then only these two peers are equivalent then only this A and B could be equivalent, reverse is also true.

So, let us see if A and B are equivalent or not. So, let us see at D and F, D, we have to see here F and D, because D and F is not same, so that means, A and B can never be equivalent, so we have put a cross here. So, that means A and B cannot be same. So, now let us look at C and E. Around C and E, this is E, this is C, so it depends on A and D we do not know. So, now let us look at AF, EH, for A and F, A and F is also this is A this is F. So, this particular cell is corresponding to A F, which means that A and F are not same, because A and F are not same so that also means that D and B cannot be same. So, we will put a cross here. So, this we have already seen.

Now, let us look at EF, BD. S, whether E and F. For E will get F, we do not know what E and F. Can we say anything would B and D? For B and D, yes, B and D are not same that we have identified, so we had to mark this also as a cross. But let us do that in the second iteration. So, in first iteration, we try to see all the possibilities, which is there from the

original list. So, C, F, A, B. A, B, also we have identified that it is not equivalent so that means we can cross this also, but let us do that in the next iteration.

Now let us look at BD and CH, B, and D, yes, B and D is also cross, so that means this G and A will also be crossed. So now, look at B and F, B, and F. So, B and F is not equivalent so that means this particular cell is also not going to be equivalent. Now, look at A and B. So, this will, this is because of A and B, we will cross this in the next iteration. Now let us look at C and E. So, C and E, C, and E depends on A and D, we do not know but D and G, D and G, we do not know about D and G also. A and G, A and G that also we do not know.

So, C and F, B and G, C and F, C is here, and F is here, we do not know what C and F. And do we know anything would B and G? For B and G, but it will come in the next iteration. So, in the first iteration, we have found few, but for the next iteration, we have already seen that A and B could be crossed and E, this also could be crossed because of E and this could be crossed because of B and D.

So, B is here D because of B and D we can cross this also and similarly, this can be crossed because of A and B. A and B are not equivalent so this particular cell can also be not equivalent. Now, because of them, because of this newly found crosses so, we can cross this also. This is crossed because of B and G, which we have recently crossed. So, because B and G cannot be equivalent, so because of that H and F also cannot be equivalent.

So, similarly for B and G, we can cross this also. This is again because of B and D, this is B and D because of B and D this is also not. Because B and D is not equivalent, because of that A and D is also not equivalent. Now, because A and G is not equivalent, so this is also not equivalent. C and E, this is because of D and G, so now D and G is not equivalent so this is also not equivalent. And because we have removed A and G, A and G are not equivalent and so this E and H is also not equivalent.

So, this we are trying to do a recursively. So, as soon as one particular peer says that they cannot be equivalent because of that some other peer will also say they cannot be equivalent. So, if we remove all of them, what we have seen finally is the two peers which are left C and E and A and D.

So, if you see, this peer, A and B depends on C and E, and C and E peer depends on A and D. So, this appears that this A is equal to D and C is equal to E. So, if we do that, so let us say we replace here with C and E are equivalent, so that means A and D are equivalent if A and

D are equivalent that means C and E are equivalent so, these two conditions crosses with each other. And we can conclude that A is equivalent to D and C is equal to E.

(Refer Slide Time: 26:11)



### Final reduced state table

Present state	Next state X=0	Next state X=1	Output
a	a	c	0
b	f	h	0
c	c	a	1
<del>a</del>	<del>a</del>	<del>c</del>	<del>0</del>
<del>c</del>	<del>c</del>	<del>a</del>	<del>1</del>
f	f	b	1
g	b	h	0
h	c	g	1

Digital Logic Design: Sequential Circuits



So, in the next step, what we should do is we should replace the original table wherever D was there, we will replace them with A and wherever. So, wherever E was there we will replace it with C. So, when we replace them, then what do we see, we see that these two rows become redundant. So, AAC 0 and we see AAC 0 here also. Similarly, we see CC, we can remove this and now, we see CCA 1 and here also we see CCA 1 so, we can remove this also.

So, this would be the final, if we remove these two rows, this will be finally reduced row. So, using this whole exercise we are able to find two redundant sets. And then we can say that whatever state machine will come out of that it would be optimized one because we have removed all the possible redundant states. So, this is how we can reduce the state machines.