Design Verification and Test of Digital VLSI Designs

Dr. Santosh Biswas
Dr. Jatindra Kumar Deka
Indian Institute of Technology, Guwahati

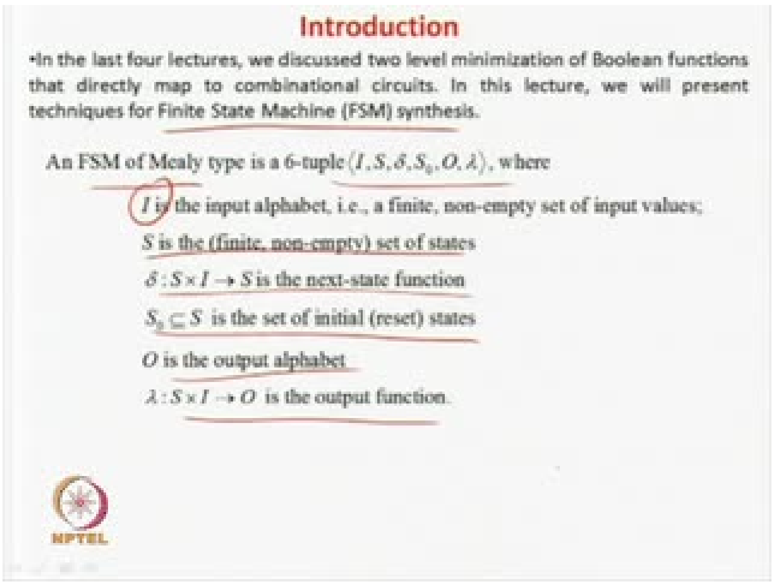
Module -3
Lecture - 5
Finite State Machine Synthesis

So, welcome to the fifth lecture of the third module which is one finite state of synthesis. So, in the last lecture or in the series of lecture or in the design module of the course; so, what we have seen, we have seen the starts from design specification and from there we go on to high levels synthesis and then we go for 2 level gate levels synthesis that is the Boolean circuits. And for the functions we have inform which we output, we obtained from the high level synthesis implementation in terms of the gates; logic gates. So, we go for 2 levels of logic gates synthesis and so for.

So, in other words transforming inputs specification to high level synthesis and from high level synthesis is the output is the r t l base block level diagrams, block level design, in terms of resistance transfer level blocks. Following that we go for logic gate level synthesis and then we keep on going steps of sub steps. So, in the last few lectures on the design module we have seen that how can we can get starting of specification and how can we get the 2 level gate, level implementation. But, our discussion you might be noting, we might have noted that was restricted to combinational circuits.

In the mainly, mainly in the area of combinational circuits that is we started with the design specification and we had some architectural diagram. Then we went for gate level synthesis we mainly restricted or sub to combinational circuits. So, we have seen that the some of the exact algorithms like selecting a prime implicates and then going for branch and bound algorithm. To implement a minimize 2 level implementation in terms of logic gates and in the last lecture, we have seen how we can go on for heuristic like express so. Which can actually go for a 2 level synthesis but, the I mean cost of execution is much less or is computationally much less compare to branch and bound base algorithm. But, whatever was the case our main discussion was restricted for combinational circuits.

(Refer Slide Time: 02:34)



Introduction

•In the last four lectures, we discussed two level minimization of Boolean functions that directly map to combinational circuits. In this lecture, we will present techniques for Finite State Machine (FSM) synthesis.

An FSM of Mealy type is a 6-tuple $\langle I, S, \delta, S_0, O, \lambda \rangle$, where

I is the input alphabet, i.e., a finite, non-empty set of input values;

S is the (finite, non-empty) set of states

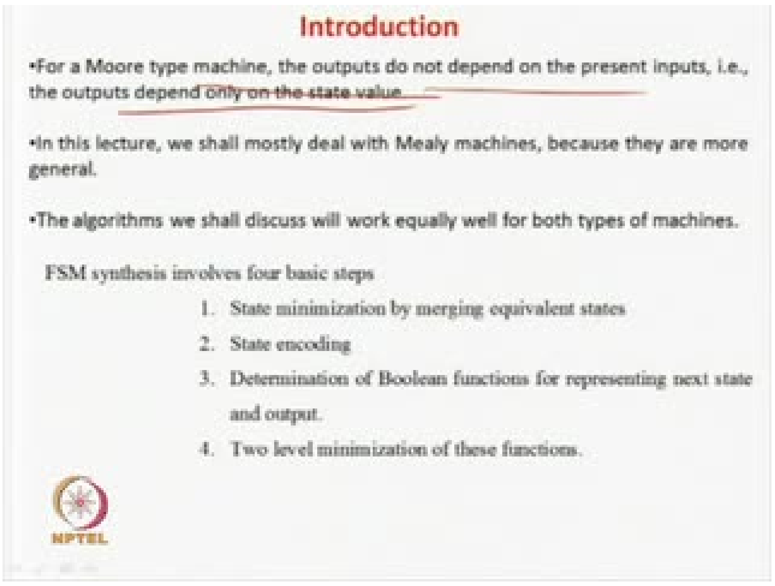
$\delta : S \times I \to S$ is the next-state function

$S_0 \subseteq S$ is the set of initial (reset) states

O is the output alphabet

$\lambda : S \times I \to O$ is the output function.

(Refer Slide Time: 03:41)



Introduction

•For a Moore type machine, the outputs do not depend on the present inputs, i.e., the outputs depend only on the state value.

•In this lecture, we shall mostly deal with Mealy machines, because they are more general.

•The algorithms we shall discuss will work equally well for both types of machines.

FSM synthesis involves four basic steps

1. State minimization by merging equivalent states
2. State encoding
3. Determination of Boolean functions for representing next state and output.
4. Two level minimization of these functions.

Now in today lectures, what we are going to do we are taking to do Finite State Machine base synthesis which is nothing but, actually 2 level implementation when your circuits are sequential. That is we have some flip flop, so we requires some memory circuits is of sequentially in nature so, how we can do that. So, we will realize that the almost the same philosophy which we have used in the 2 level synthesis will be applied over here, but also in addition we have to be take care of the states.

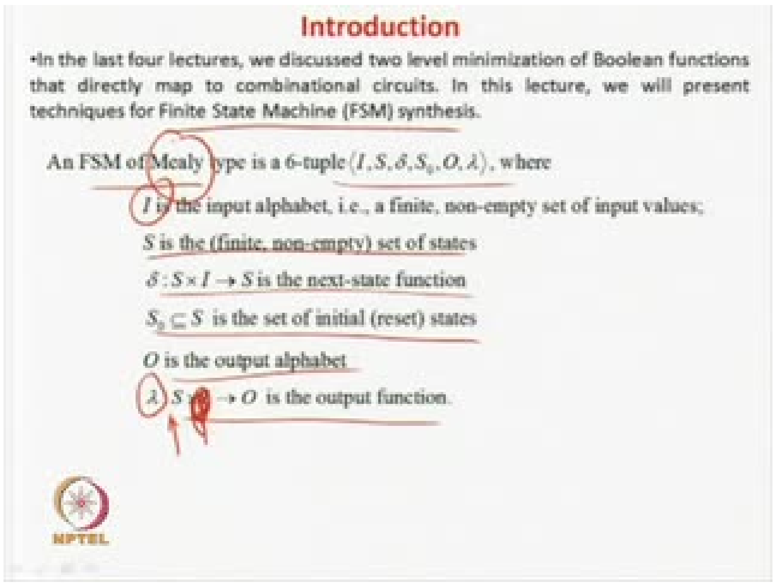
So, a whenever we talk about this finite state of machine sorry whenever we talk about this sequential circuits, we know that they can model in terms of Finite State Machine. Because this is from our undergraduate course on flat there is automation theory. We all know that given a circuit which is having sequential in nature they can be implemented in terms of Finite State Machine. So, Finite State Machine can be 2 types Moore or Mealy.

So, we will see one by one; so the basic idea we already we know from our undergraduate's studies then even a sequential circuit we require a Finite State Machine to modulate. So, what is Finite State Machine of Mealy type and what is Moore type. But, lets us first see what the Finite State Machines are. So, we set a 6 topple, so we consists of input alphabets in the case of digital inputs are primary inputs and S is the Finite State Machines of states this is the next state function.

That will tell you that given an input and connection what will be the next state the initial state or the reset state that when we will start. Is the output alphabet that is the output corresponding to the each state and there is an output function. So, this is the basic definition of what we called Mealy type of Finite State Machine in fact it will take some inputs and based on the present state it will go to a next state and also it will generates the output that is the basic idea.

And of course this is what is required in the sequential circuits. So, sequential circuits can easily modulate in terms of Finite State Machine. So there are some 2 type: there is one is the Moore type and another one is the Mealy type. So, this was the Mealy type machine, so what is the Moore type? So, Moore type machine is the case; so outputs are not depends on the presents inputs and it is only depends on the state value.

(Refer Slide Time: 03:58)


Introduction

•In the last four lectures, we discussed two level minimization of Boolean functions that directly map to combinational circuits. In this lecture, we will present techniques for Finite State Machine (FSM) synthesis.

An FSM of Mealy type is a 6-tuple $\langle I, S, \delta, S_0, O, \lambda \rangle$, where

I is the input alphabet, i.e., a finite, non-empty set of input values;

S is the (finite, non-empty) set of states

$\delta : S \times I \rightarrow S$ is the next-state function

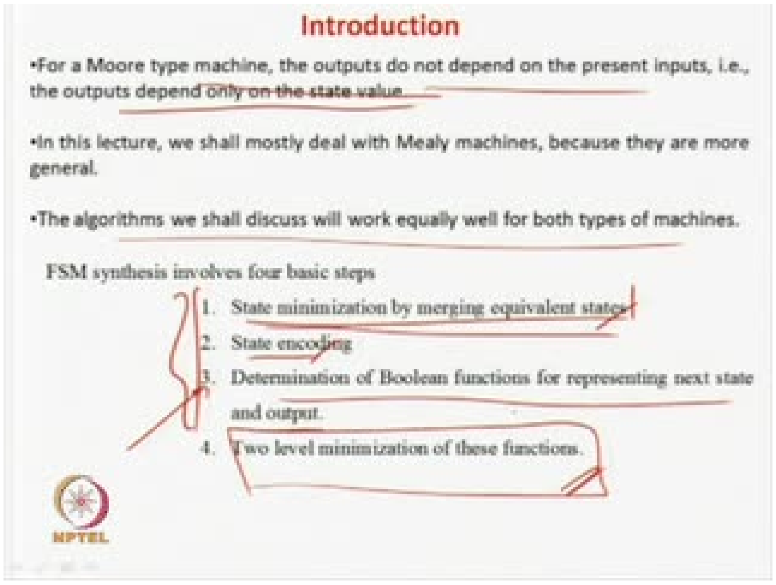
$S_0 \subseteq S$ is the set of initial (reset) states

O is the output alphabet

$\lambda : S \times I \rightarrow O$ is the output function.

So, what is the case, so in this case the output function here you can see that is dependent on the both the primary inputs and the states. So, is the Mealy machine but, if I remove inputs then actually s all the states will correspond to some output then it become Moore machine. So of course Mealy machine is the more generalized form of what we can say Finite State Machine. So, here in this lecture, we can consider mainly on Mealy machine because Moore machine can be consider as a sub class of this one.

(Refer Slide Time: 04:24)


Introduction

•For a Moore type machine, the outputs do not depend on the present inputs, i.e., the outputs depend only on the state value.

•In this lecture, we shall mostly deal with Mealy machines, because they are more general.

•The algorithms we shall discuss will work equally well for both types of machines.

FSM synthesis involves four basic steps

1. State minimization by merging equivalent states
2. State encoding
3. Determination of Boolean functions for representing next state and output.
4. Two level minimization of these functions.

So the algorithm, I mean, so in this case we will find out algorithms which can synthesis Mealy machine and Moore machines, of course it will be synthesis using them. And mainly what we will see so how we can synthesize or how we can implement synthesis means, if we talk corresponding the previous lecture how we can have a 2 level implementation, for a sequential circuit that is a given a sequential circuit or given sequential basics behaviors circuits basically.

We modeling in terms of Finite State Machine and form Finite State Machine we have to generate 2 level implementation 2 level circuit implementation and for that Finite State Machine. So, what are the basic steps what will be involved a first we have to minimize the number of states. So, what are the basically and finally, we have to use our own express algorithm or your branch and bound algorithm for the minimization.

But, before that as this circuits is not sequential we have to go for some what we can say call pre define steps or preliminary steps pre processing steps so that it can be broad down so the level where it can be express a algorithm or we can say branch and bound algorithm its can be applied. So, if your circuit is the purely combinational that you have input and output behavior given to you that is some of the product from is given to you and what you can do.

You can go for either branch and bound base algorithms or you can go for express to minimize that but, in this case of sequential circuit along with the input output behavior you also stage. We have to do some pre possessing so that it can be broad to the format of a combinational circuit kind of a thing so that can be directly express so or other synthesis algorithms can be applied. So, what are the steps of pre processing we actually go for sate minimization because of some of the state will be equivalent we will seen the lecture number 1. What is mean by equivalent sates? So, of the states will be equivalent to the so mass those equivalents states then we have to go for State Encoding.

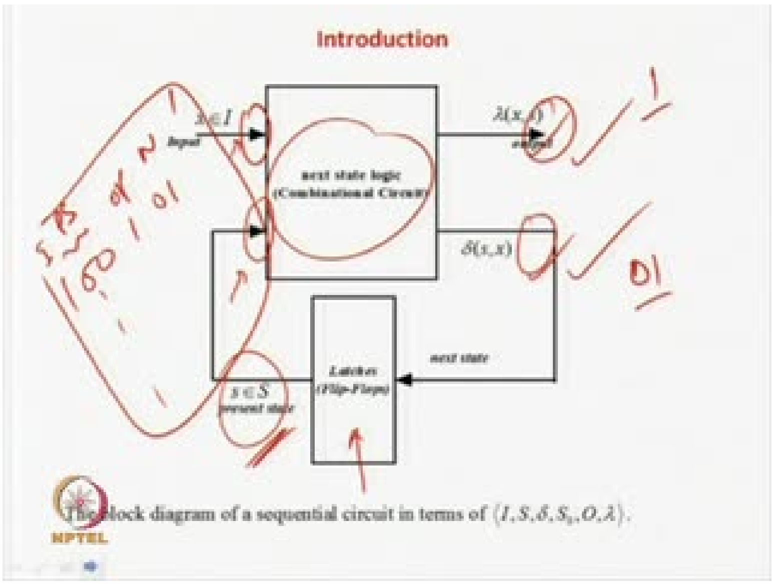
They because they are circuit may have ten state after say equivalent margin we have five states. So, all these states you have to encode with 0 0 0 1 1 0 1 1. So, if there 4 states we required to bits to encode if there are 3 states you required, if there more than 4 states and about less than 8 states then you required 3 bits to encoding.

So, for I think you know that is log of 2 in the number of states upper bound can be giving you to the number of bits required to encode this for the undergraduate's states.

Then, you finite Boolean functions representing in the next state and the output. That we will see and the new go for 2 level minimization of this function. Basically this is nothing but, that convert sequential circuit to the combinational behavior and from there you can use actually the it heuristics, which we have discussed heuristics and branch and bound algorithm which we have discussed in the last two lectures.

So, basically there are the three pre processing steps minimization of the equivalence states and states encoding and determine of Boolean functions for determining the next state and the output. So, we will see with the example for the mean, once we do that you are ready for a 2 level minimization using the algorithm solving us already discussed.

(Refer Slide Time: 07:01)



Introduction

The block diagram of a sequential circuit in terms of $\langle I, S, \delta, S_0, O, \lambda \rangle$.

So, if you look at this is the architecture of are these the architecture of a sequential circuit and represented in terms of Finite State Machine. So, this is your input primary input this is the output lambda is the output function. So, you have this you can see the output will be the value of presents states as well as, the input with the Mealy machine.

And actually we should also have the next state. So, next state is dependent on the again x and s and represents as the input so this is your present state, so this is your next state. So, after one clock pulse the next state will become present state. These are all you are your undergraduate studies on digital circuits and this is the secret your present state.

So this is our combinational circuit which is determining what will be the output and what will be the next state based on the input and the present state. So if it becomes a Moore machine we do not have considered the input so output will be dependent only on the present state. So, this is what is the basic architecture of a Finite State Machine which is to be implemented in terms of circuit kind of a thing. Now our idea is that we have to synthesize this part.

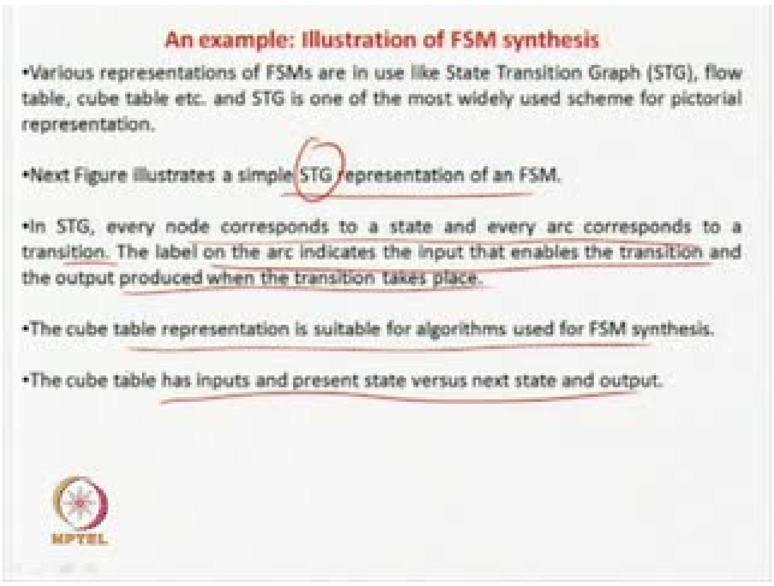
So, this is also already flows nothing there required to synthesize. So, what do we required we required for what values of inputs and what is the present state and what is the actually this one and this one so. You can consider the this one is the combinational circuits. So based on some primary inputs and the present state balance we have to generate the outputs and generally next states.

So once we get for what input and present state combinations what are the outputs and what are the next states? So you can easily thing that it is a synthesis problem for a combinational circuits. So, you can easily go about using or old algorithm like branch and bound next processes we solving the problem. What here actually we require the present state which is not present in the combinational circuit synthesis that is one thing which is required. So, you have to go for state equivalence merging as well as state encoding.

So, once you encode this state so you know that is the present state is 0 0 the input is 1 the output is 1 and the next state is say 0 1 the next is 0 1. So you know that in 1say the input is 1 present state is 0 0, the output has to be 1 and the next state is 0 1. So, it is actually output is 1 and next state is say 0 1. So, immediately in the like dot dot dot you can make up a table like this or you can make up a table sorry this one is not there you make up a table.

This is the our output next state this is your output and this is your present state. Then you can easily make up a table; and go for combinational circuit synthesis. So, that is the very simple idea of how you can go about synthesizing a sequential circuits.

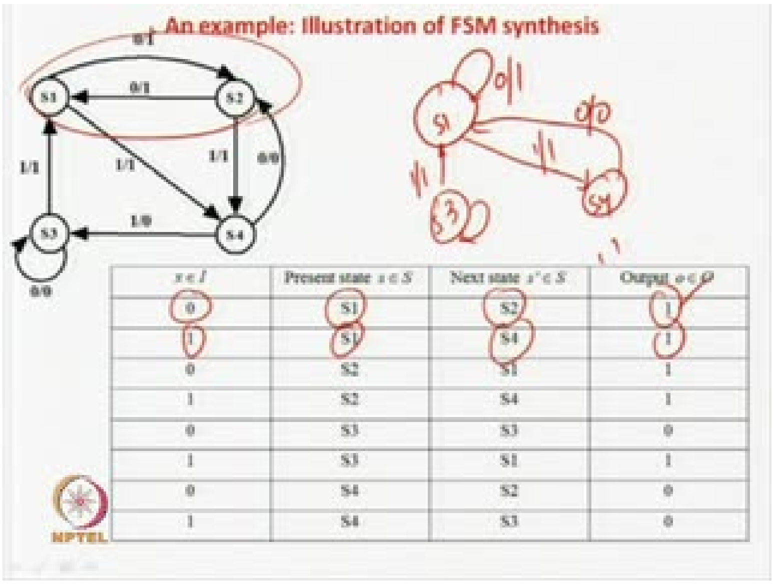
(Refer Slide Time: 09:16)



An example: Illustration of FSM synthesis

- Various representations of FSMs are in use like State Transition Graph (STG), flow table, cube table etc. and STG is one of the most widely used scheme for pictorial representation.
- Next Figure illustrates a simple STG representation of an FSM.
- In STG, every node corresponds to a state and every arc corresponds to a transition. The label on the arc indicates the input that enables the transition and the output produced when the transition takes place.
- The cube table representation is suitable for algorithms used for FSM synthesis.
- The cube table has inputs and present state versus next state and output.

(Refer Slide Time: 09:55)



An example: Illustration of FSM synthesis

$x \in I$	Present state $s \in S$	Next state $s' \in S$	Output $o \in O$
0	S1	S2	1
1	S1	S4	1
0	S2	S1	1
1	S2	S4	1
0	S3	S3	0
1	S3	S1	1
0	S4	S2	0
1	S4	S3	0

So next will now we will take some examples and explain all these things. So, in the next figure will represent what is the saying as Single Transition Graph will be used to represent in F S M for when you are going to synthesis the circuit. So, that is one standard way of representing F S M one will be using circuits synthesis for them.

So, in S T G every node corresponds to states and every arc corresponds to a transition and the label on the arc states which inputs enable the transition and the output produce is also given in the transition. Then we will see I mean the cube table representation is suitable for a F S M synthesis is the cube table has inputs present state values and outputs and let us see all these things what i have told you using an example. So, this is your S T G of finite state machine. So, what you say in state S 1 if you apply a 0 inputs the output is 1 and now go to state S 2. Now see the present state is S 1 the input is 0 the output is 1 and the next state is S 2. Similarly, we can easily see let us take another example from the state itself from state 1 if you apply a 1 from state 1 if you apply 1.

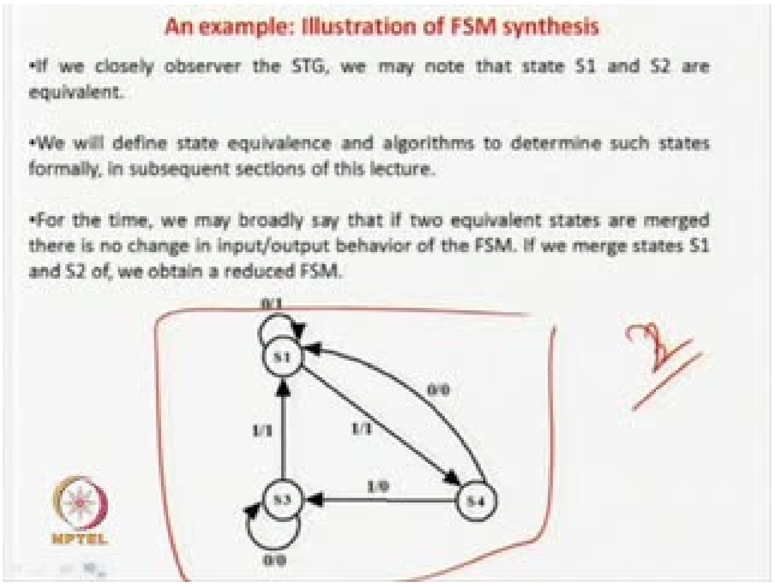
Then you go to state S 4 you go to state is S 4 and the output is a 1. So, in this way the hold table can be made from the S T G of S 1. Now what we have to do now these are your inputs and this part is your inputs so based on the present inputs and the present state values. What is the next state value and what are the outputs? So for that this is the actually we will design your and I mean inputs we can call the these inputs and we will be using 2 level implementation synthesis rules to find out the implement logic for this one and combinational circuits for this one.

But, we will see but, one more thing we have to do in this case you see S 1 S 2 S 3 and S 4 and now in terms of some alphabets or some names. So, you have to give some encoding for this one. So now you can see that you have some 4 states so 2 bits are enough to encode it. So you can say they S 0 S 0 0 S 2 is 0 1 1 0 and 1 1 kind of a thing. So, you have to give some values to this and then we can go about the synthesis. So, all this things you will see. This is the example shows how a table can be converted. So now, once again in the next state the first step was equivalent state we will merge them the state. So, we will find in the next few sides we will give formula and definition of the equivalence and all. So, for the time being let us see this just sorry this is for the time we assume that. So, S 1 and S 2 are equivalent so how can equivalent can so we will see.

So in this case, we can see that formal definition coming up shortly first for the time being just assumed that is S 1 and S 2 are equivalent. So if S 1 and S 2 are equivalent so there will be merge and then you can see if you apply 0 from S 1 and S 2. If we apply 0 from S 2 you go to S 1 and output will both the cases are 1. So, you will have s 1 the input is 0 the output is 1 that way you can design and now S 1 if we apply 1, 1 this will go to S 4. So, this actually 1 and 1 and

Now obviously from S 2 also you have to apply 1 then you go for S 1 that is why it is possible from S 1 if we apply 1. We get the output 1 go to the statement from S 2 also same thing happens. So, for the combine state if you can go to S 4 in terms of this one and then also you can see from S 3 just you can have self loop if you apply 1 and 1 output is 1 you get apply a 1 output we get as 1 you go to the state S 1. So, for S 1 we will go over here correct, all these other things remain same kind of a thing from near obviously from S 4, if you apply a 0 you go to state 0 and get 0 state S 2 but, S 1 and S 2 are merge so you will have starting like conditions like this similarly; the whole graph actually will look like something like this.

(Refer Slide Time: 12:44)

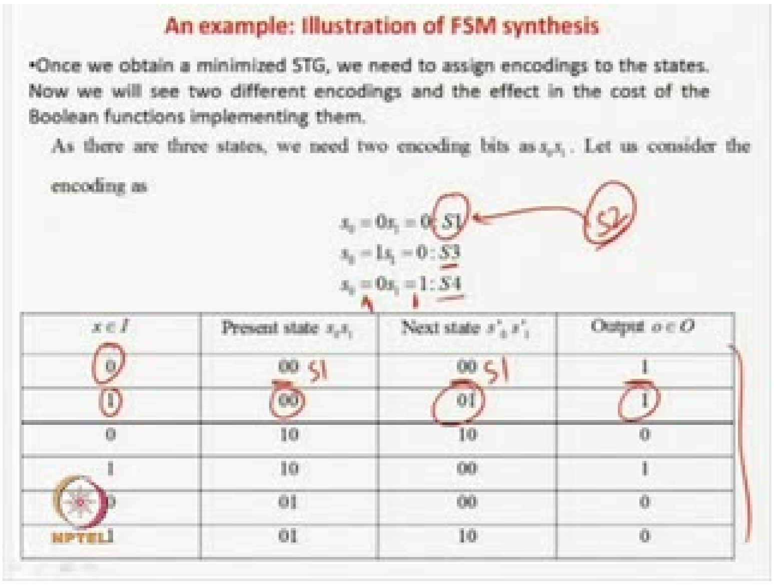


An example: Illustration of FSM synthesis

- If we closely observer the STG, we may note that state S1 and S2 are equivalent.
- We will define state equivalence and algorithms to determine such states formally, in subsequent sections of this lecture.
- For the time, we may broadly say that if two equivalent states are merged there is no change in input/output behavior of the FSM. If we merge states S1 and S2 of, we obtain a reduced FSM.

Now in this case what we have to done S 1 and S 2 has to be merge. Now we can actually get this as a Finite State Machine. But, how we can say formally this one is equivalent all those things will be looking like this in a few movements. Let us just complete the example and then we will see how formally we can define that. So, this is now your Finite State Machine where S 1 and S 2 are merge. Now we have 3 states, so again 2 bits will be require to encode this one ok.

(Refer Slide Time: 13:10)



An example: Illustration of FSM synthesis

•Once we obtain a minimized STG, we need to assign encodings to the states. Now we will see two different encodings and the effect in the cost of the Boolean functions implementing them.

As there are three states, we need two encoding bits as s_0s_1. Let us consider the encoding as

$s_0 = 0 s_1 = 0 : S1$ ← S2

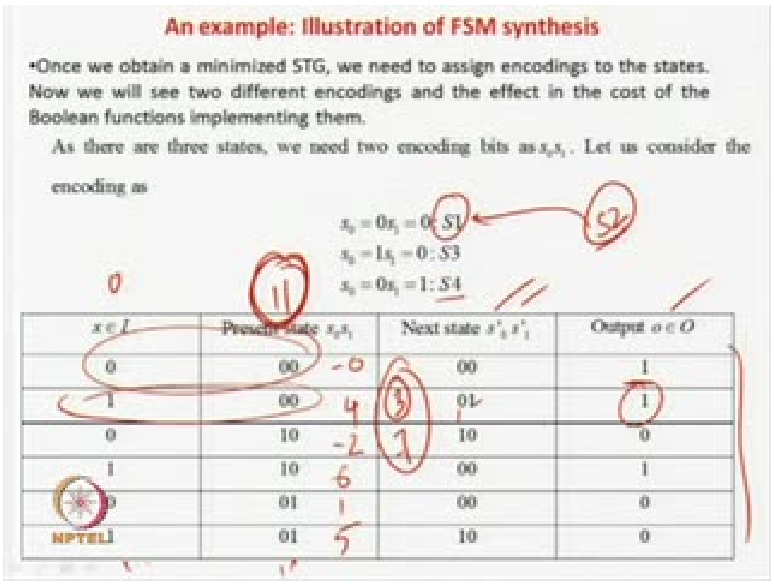
$s_0 = 1 s_1 = 0 : S3$

$s_0 = 0 s_1 = 1 : S4$

$x \in I$	Present state s_0s_1	Next state $s'_0 s'_1$	Output $o \in O$
0	00 S1	00 S1	1
1	00	01	1
0	10	10	0
1	10	00	1
0	01	00	0
1	01	10	0

So, let us see, we can take some encoding scheme. So in this case if there are saying that is 1 is encoded as 0 0, S 3 is encoded as 1 0 and S 4 is encoded as 0 1. So, what is the case now, so you can see the whole table be drawing but now in this case S 2 is no longer there because S 2 is merge with S 1 but, now we can just see what happens. So, if apply a s 1 if we apply 0 we get as 1 and go back to S 1. We can see this is nothing but, S 1 and the output is 1. So, if you apply a 0 1 here in the present state is S 1. That is 0, 0 is encoded as 0, 0 in the next state is nothing but, again S 1 is set to the output is 0, 0 and the

(Refer Slide Time: 14:19)



An example: Illustration of FSM synthesis

•Once we obtain a minimized STG, we need to assign encodings to the states. Now we will see two different encodings and the effect in the cost of the Boolean functions implementing them.

As there are three states, we need two encoding bits as s_0s_1. Let us consider the encoding as

$s_0 = 0 s_1 = 0 : S1$ ← S2

$s_0 = 1 s_1 = 0 : S3$

$s_0 = 0 s_1 = 1 : S4$

0 11

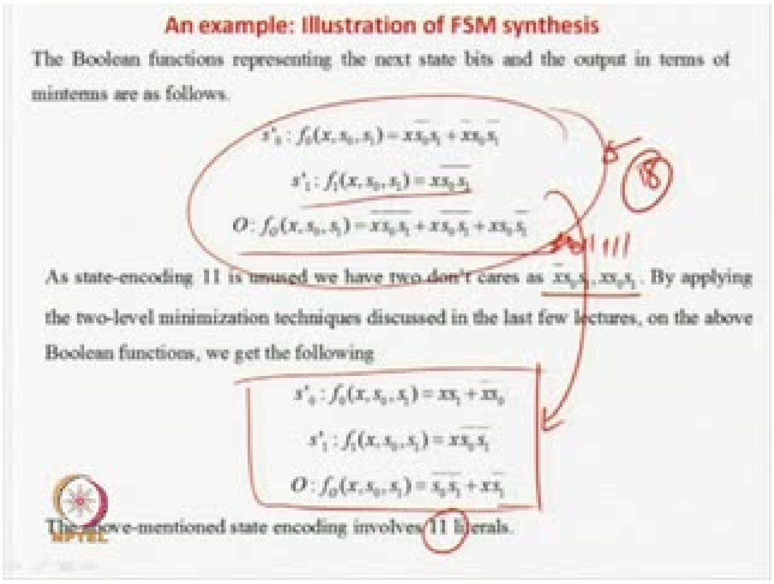
$x \in I$	Present state s_0s_1	Next state $s'_0 s'_1$	Output $o \in O$
0	00 -0	00	1
1	00 4	01 3	1
0	10 -2	10 7	0
1	10 6	00	1
0	01 1	00	0
1	01 5	10	0

Output that is the output function generated that is output is 1. So, we discuss like 0 0 present state input is 0 in the next state again 0, 0 self loop and the output is a 1. From here from S 1 if we apply 1 the output is 1, we go to S 4. So, we can say if we apply 1 over here presence state is 0 0 and the next state is S 4, S 4 is encode as 0 1 and output is as 0 1. Similarly, the whole table can be drawn.

Now after draw of this table what we see, now what we have to do, so you should combination circuit for 0 prime S 1 prime and output. So, if you see that 0 0 0 0 0 0 0 0 is the input that this one, so this the main term in this case the output 0 and also for S 0 prime S 1 is also 0 and the output is 1. So, if the input is 0 1 0 0 1 0 the S 0 prime is 0 S 1 prime is 1 and the output is 1 this output is 0 0 0. So, this is corresponded to 0 this corresponded to 4 is 0 1 to 4 that is S 2 0 0 1 0 1 1 0 it is 6 and 1 0 1 is 1 and 1 0 1 is 5. So you have 0 1 2 4 5 some do not case have objectives will come into picture. So, in this case we can think 0 1 is there 2 is there and there 3 is not there. So, 3 is do not care 4 is there and 5 is there and 6 is there and 7; so we do not care says 1 1 1 and we go about do not care is 0 1 1. This is the case because we do not have state encoded as 1 1.

So, 1 1 as represents the state next state whatever you consider from state is 1 1, that is presents state is 1, 1 input is 0 or 1 what are they do not care conditions. So while minimizing this, you can use this 3 and 7 that is 0 1 1 and 1 1 1 as the do not care.

(Refer Slide Time: 15:39)



An example: Illustration of FSM synthesis

The Boolean functions representing the next state bits and the output in terms of minterms are as follows.

$$s'_0 : f_0(x, s_0, s_1) = x\bar{s}_0 s_1 + \bar{x} s_0 \bar{s}_1$$

$$s'_1 : f_1(x, s_0, s_1) = x\overline{s_0 s_1}$$

$$O : f_O(x, s_0, s_1) = \overline{x s_0} \bar{s}_1 + \overline{x s_0} s_1 + x s_0 \bar{s}_1$$

As state-encoding 11 is unused we have two don't cares as $\bar{x} s_0 s_1, x s_0 s_1$. By applying the two-level minimization techniques discussed in the last few lectures, on the above Boolean functions, we get the following

$$s'_0 : f_0(x, s_0, s_1) = x s_1 + \bar{x} s_0$$

$$s'_1 : f_1(x, s_0, s_1) = x\overline{s_0 s_1}$$

$$O : f_O(x, s_0, s_1) = \bar{s}_0 \bar{s}_1 + x\bar{s}_1$$

The above-mentioned state encoding involves 11 literals.

So, now let us see what this is synthesis picture look like so you can find out just if you look at these terms. So, in this case you can see this we have only two 1s over here. So, 1 0 1 and 0 0 1 0 that is 0 1 0 that is 2 and this is 5 these are the two cases only in this case 0 prime is a 1. So, in this case you can see that is 0 1 1 sorry S 0 prime is 1 0 1 that is 5. So, that is 1 0 1 that is 5 and 2 that is 0 1 0 so these are the mean terms corresponding to S naught prime similarly you will find out that the mean term for S 1 prime is this one and the mean term is this one. Now you have to do not care sets 1 sorry 0 1 1 and 1 1 1.

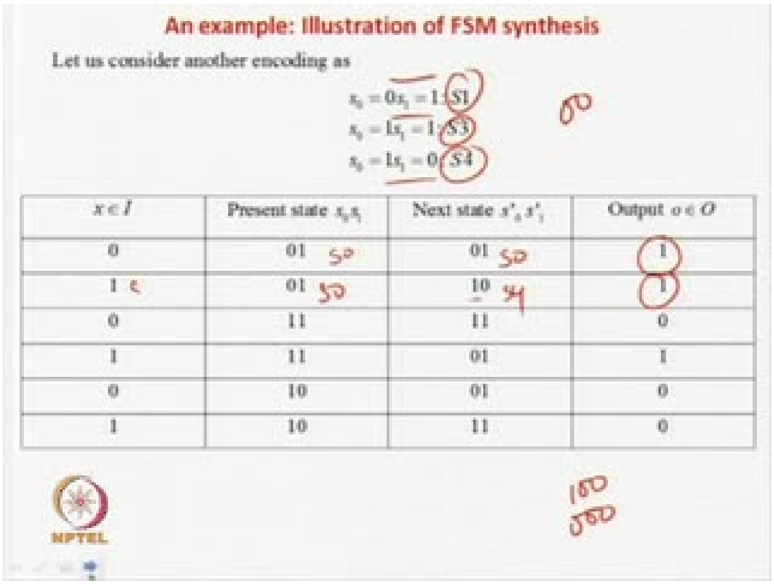
Because 1 1 is not use an encoding but now, now so these are your equations. So now you can easily find out that this has been now converted into minimization by 2 level minimization functions. We using any of these schemes we have discussed in the last lecture like last three lectures that is your branch and bound or express so.

So, if you minimize this we will find out that just you can apply at your home and you can find out this you can do some a calculation using some or express or you can use your exact minimizes then you we can find out that this can be reduces to something like this. And here we have 1 2 3 4 5 6 7 8 9 10 11. So, this is have now 11 literals. So now from initially how 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 from 18 literals by minimization we have got our 11 literals. So, this is how we actually convert a sequential circuit synthesis to what we can call combinational circuit's synthesis. What we do we have circuit level final state machine, like this on table like this. So, this is called an S T G.

So, you merge equivalence states so after merging the equivalent state you have something like this then you go for state encoding which is in this case is now S 1 is represents 0 0 S 3 is 1 0 and S 4 is 0 1. You go for encoding then you make this table, so once you have this table this in the now no longer sequential circuit problem it becomes combinational circuit synthesize you synthesize that and you get final you minimize 2 level implementation which is having some 11 literals in this present case. So these how, how sequential circuit synthesis is goes.

So, basically sequential circuit synthesize nothing but a combinational circuit synthesis but, in this some pre-processing steps like merging equivalent state and finding out state encoding.

(Refer Slide Time: 18:00)



An example: Illustration of FSM synthesis

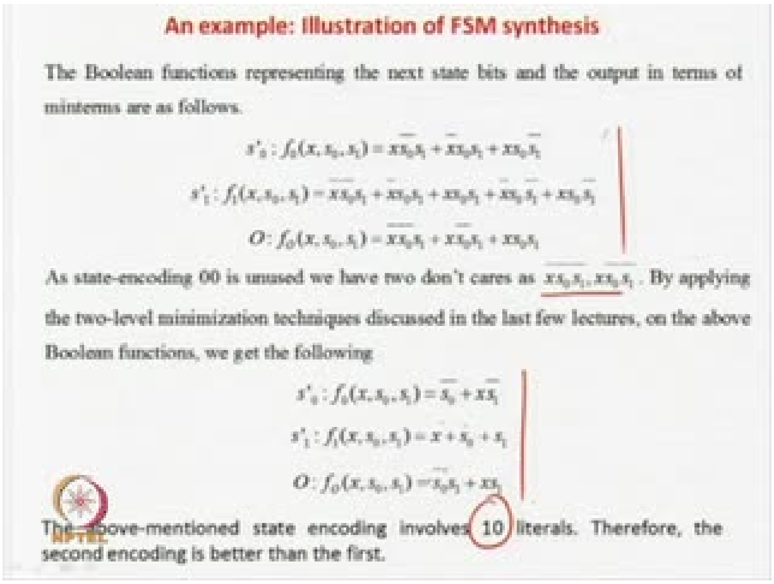
Let us consider another encoding as

$$s_0 = 0\, s_1 = 1\; \boxed{S1}$$
$$s_0 = 1\, s_1 = 1\; \boxed{S3}$$
$$s_0 = 1\, s_1 = 0\; \boxed{S4}$$

$x \in I$	Present state $s_0 s_1$	Next state s'_0, s'_1	Output $o \in O$
0	01	01	1
1	01	10	1
0	11	11	0
1	11	01	1
0	10	01	0
1	10	11	0

00
S0
S0
S0
S0
S4
100
000

(Refer Slide Time: 19:59)



An example: Illustration of FSM synthesis

The Boolean functions representing the next state bits and the output in terms of minterms are as follows.

$$s'_0 : f_0(x, s_0, s_1) = x\bar{s}_0 s_1 + \bar{x} s_0 s_1 + x s_0 \bar{s}_1$$
$$s'_1 : f_1(x, s_0, s_1) = \bar{x}\bar{s}_0 s_1 + \bar{x} s_0 s_1 + x s_0 s_1 + \bar{x} s_0 \bar{s}_1 + x s_0 \bar{s}_1$$
$$O : f_O(x, s_0, s_1) = \bar{x}\bar{s}_0 s_1 + x\bar{s}_0 s_1 + x s_0 s_1$$

As state-encoding 00 is unused we have two don't cares as $\bar{x}\bar{s}_0\bar{s}_1, x\bar{s}_0\bar{s}_1$. By applying the two-level minimization techniques discussed in the last few lectures, on the above Boolean functions, we get the following

$$s'_0 : f_0(x, s_0, s_1) = \bar{s}_0 + x\bar{s}_1$$
$$s'_1 : f_1(x, s_0, s_1) = \bar{x} + s_0 + s_1$$
$$O : f_O(x, s_0, s_1) = \bar{s}_0 s_1 + x s_1$$

The above-mentioned state encoding involves $\boxed{10}$ literals. Therefore, the second encoding is better than the first.

Now we will see as a interesting thing. Now let us change the state encoding, so in this case we have given set encoding as a 0 0 1 0 1 1 so who has given this how it came from here? So, we have put have also started with 1 1 0 1 and 1 1 1 0 something like this. So, any state encoding you put have used so let us see with I must state encoding what is the case. So in this case if there S 1 represented as 0 1 is S 3 is represented as 1 1 and S 4 represented as 1 0.

So, this is the case now you will find out that, so present state is S 0 S 0. So you apply as the 0 the present state is 0 1. Now 0 is not encoded as 0 0 it is encoded as 0 1 so this is a in change the output is 1. Similarly, from a S 0 if we apply 1. So, next state 1 0 so 1 0 is nothing but S 4. So, you can see the figure so you should apply 1 from S 1, it will go to the S 4 and the output is the 1 so that is being stated.

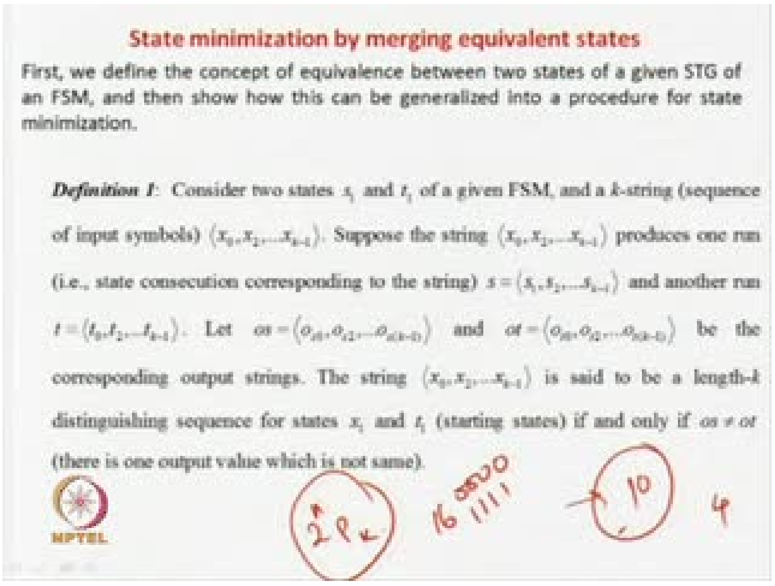
So, now the S 4 is encoded as 1 0. So, from S 0 if we applied 1, so S 0 0 1 then you go to S 4 whose encoding is 1 0 in the output is 1. So, if we just represented in terms of mean terms and some S 4 is form. So, we will get this expression from S 0 prime and S 1 prime encode. So here again we have do not care now we have do not care terms will be different because in this case represent 0 0 encoding is not used. So, 0 0 encoding is not used so your do not care terms will be 1 0 0 and 0 0 0.

So, you have these two do not cares if you just see so you have these two do not cares like this is 0 0 0 and 1 0 these are the two do not care terms and this is your expression, you can easily find out that these are your mean terms. So, expression in terms of S 0 S 1 S 0 prime S 1 prime and you can be found out. Now again by using these binary minimization techniques like express so or whatever you can use, so you can find out the this will be the final expression. So, here you are having 1 2 3 4 5 6 7 8 9 10; now here 10 literals.

So, you see now what is the interesting thing as happened is so with this type of state encoding like 0 0 1 0 and 0 1 and the number of literals required was 11. But, with this new state encoding the number of literals required are table. So, not only the pre-processing steps you can take arbitrarily but, you will be very careful that the state encoding has the great impact on the area of the circuit.

So, previous from lectures we have seen combinational circuit synthesis if we are going for the exact algorithm. So, we can get an optimal are representation as 2 level implementation representation we have but, then we can have followed by long amount of time. So, we had time so we had find out one person of heuristic and sometimes you get sub optimal implementation and in terms of number of gates so we may land of you taking some gate or resource then it could have been done in the most optimal case. So, similarly in this case now the state input also becoming a problem here. So, you say for example, if you have say 100 states.

(Refer Slide Time: 20:49)



State minimization by merging equivalent states

First, we define the concept of equivalence between two states of a given STG of an FSM, and then show how this can be generalized into a procedure for state minimization.

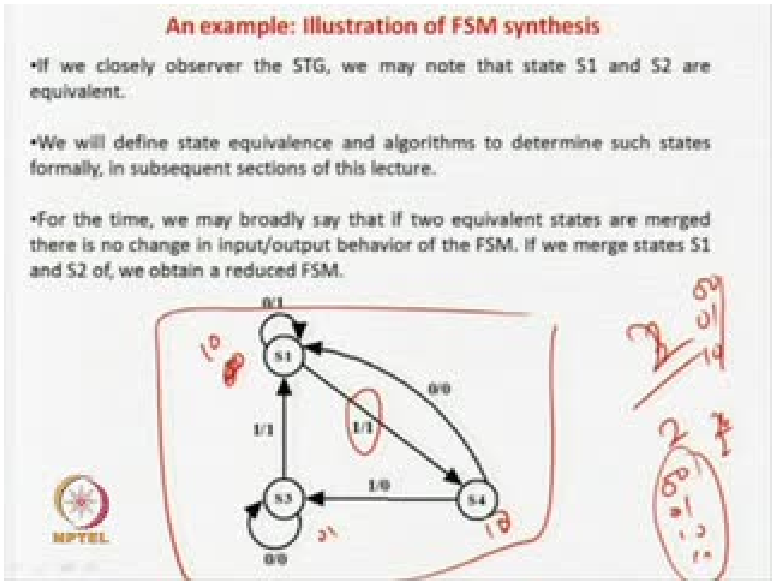
Definition 1: Consider two states s_i and t_i of a given FSM, and a k-string (sequence of input symbols) $\langle x_0, x_1, \ldots x_{k-1} \rangle$. Suppose the string $\langle x_0, x_1, \ldots x_{k-1} \rangle$ produces one run (i.e., state consecution corresponding to the string) $s = \langle s_i, s_1, \ldots s_{k-1} \rangle$ and another run $t = \langle t_0, t_1, \ldots t_{k-1} \rangle$. Let $os = \langle o_{s0}, o_{s1}, \ldots o_{s(k-1)} \rangle$ and $ot = \langle o_{t0}, o_{t1}, \ldots o_{t(k-1)} \rangle$ be the corresponding output strings. The string $\langle x_0, x_1, \ldots x_{k-1} \rangle$ is said to be a length-k distinguishing sequence for states s_i and t_i (starting states) if and only if $os \neq ot$ (there is one output value which is not same).

Ok also as say for example, we can say we have say 7 states let us take of numbers in 10 states. So, if you have 10 states that mean we require 4 bits to representing 4 bit means 16 values you can have 0 0 from 0 0 0 0 0 2 1 1 1 1 these are the range you have now you have only 10 states.

So, among the this 16 that will use 0 0 0 to dot dot dot 1 1 1 1 which of them you will be using for this encoding the 10 states and what state you will be encoding by 1. We again actually very very exponential problem because, here also sub spaces is very high. So, what is the sub space if you have n bites to encode, so this is 2 to the power and perform combinations available. Now say you have only case states, so this permutation keeps. So, among then you have to take any k among the among the if there n bits which are represented to encoding so 2 to the power n possible values and encoding values will be there if their case states to represented to be represents.

So, you can use any of this 2 to the power of n patterns available in this. So, this solution space is 2 to the power of n permutation you can have combination among them. So, keep so 2 can understand this and we can have 200 inputs are available 200 bits may be available for encoding the number states are say about 2 to the power of 5 or something like this. We can final this solution very high and depending on these selections of this encoding which values will greatly impact the circuit area. So, not only this synthesis in case of means what we can say in case of this final state machine synthesis. This choosing proper state and encoding and exp1ntial problem we know this solution space is very high. Because, that if there any n bits represented is using encoding like if you have states but, still we required 2 bits encoding because 3 states required either 0 0 1 1 1 0 and 1 1.

(Refer Slide Time: 22:36)



An example: Illustration of FSM synthesis

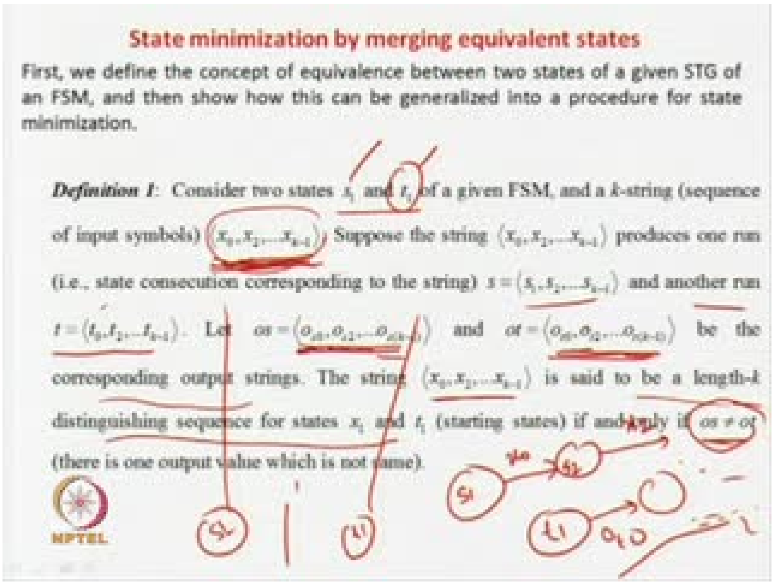
- If we closely observer the STG, we may note that state S1 and S2 are equivalent.
- We will define state equivalence and algorithms to determine such states formally, in subsequent sections of this lecture.
- For the time, we may broadly say that if two equivalent states are merged there is no change in input/output behavior of the FSM. If we merge states S1 and S2 of, we obtain a reduced FSM.

So, 1 in the like the previous example like we have three states. So, we have three states here but number of bites required will be two. So, if there two bits means 0 0 0 1 1 0 1 1. So, 2 to the power 2, that equal to 4 combination of inputs are available but among this four any three you can take and if you can use them any again in permutated with the you can use 0 0 0 1 and 1 0 for encoding 1 1 you need. Now this 0 0 0 and 1 0 can be again permuted among this. So, this can be 0 0, this can be 0 1, this can be 1 0 but, again you can also change this is as 1 0 this is 0 0 and so fore.

So, this we actually may exist solution space very very high and choosing the most optimal what do I say most of the optimal state encoding is again big problem. So, here in this lecture again will see some heuristic to solve the problem. So now in the sequential circuit synthesis is actually has many pre processing steps. The first pre processing step is merging of equivalence state. So, we will an exact algorithm for thus because that algorithm is at very complex algorithm that is to find out the Equivalence States. So, we will see as exact algorithm for this. Next is find out good state encoding so again good state encoding as the solution space is very high the problem is exponential and you will see a heuristics for this. So, the heuristics will try to give you some kind of an optimal not an optimal but some kind of optimal or near optimum solution for state encoding that will try to give you the minimal hardware representation or minimum gate level implementation. So, once that is done so you convert into the table and once the table is done and then you go for binary sorry Boolean circuit synthesis which we are already seen.

So, basically in this lecture so now we will see two methods for solving two problems one is finally equivalence states are merging them because unnecessarily if you have too many states in the circuits which are equivalence unnecessarily you will be having large amount of the area. So, we will see how states can be encoded and we will see that the problem with simple and so we can use that exact algorithm for this. And next will be State Encoding. So as you already seen the state space is very very high. So, if the state space is very very high and then what happens that means solution space is very very high. So if you want to find out the not optimal sorry most optimum point or the based point for finding out the solution is the based point in terms of state encoding which is going to the minimum get implementation is a very difficult problem and on complexity will take you very long time. So, we will see heuristic algorithm for that. so let up first see combine what you mean by equivalent state. So, we still we have discussed that equivalence states can be must.

(Refer Slide Time: 24:55)



State minimization by merging equivalent states

First, we define the concept of equivalence between two states of a given STG of an FSM, and then show how this can be generalized into a procedure for state minimization.

Definition 1: Consider two states s_i and t_i of a given FSM, and a *k*-string (sequence of input symbols) $\langle x_0, x_1, \ldots x_{k-1}\rangle$. Suppose the string $\langle x_0, x_1, \ldots x_{k-1}\rangle$ produces one run (i.e., state consecution corresponding to the string) $s = \langle s_i, s_1, \ldots s_{k-1}\rangle$ and another run $t = \langle t_0, t_1, \ldots t_{k-1}\rangle$. Let $os = \langle o_{s0}, o_{s1}, \ldots o_{s(k-1)}\rangle$ and $ot = \langle o_{t0}, o_{t1}, \ldots o_{t(k-1)}\rangle$ be the corresponding output strings. The string $\langle x_0, x_1, \ldots x_{k-1}\rangle$ is said to be a length-*k* distinguishing sequence for states s_i and t_i (starting states) if and only if $os \neq ot$ (there is one output value which is not same).

So, now how do you define state equivalence? So, state equivalence are defining the following manner. So, there are two states s 1 and t 1 in a circuit. So, you say that s 1 is a state t 1 is the state. Now you are applying some input symbol. This is x 0 x 2 x k minus 1, so you are applying some states say input say x 0. And then you go for somewhere then you apply x 2 and dot dot dot. And suppose we string there is the x 0 x 2 dot dot produce one run that is some states you are going to get that is s 1, s 2 dot dot dot and s k plus 1. And another run this one that is from s 1 before applying input x 0 x 2 dot dot dot say you are getting one set of string which is nothing one set of what you can, one set of outputs you are going to get, which is nothing but, say this s 1, s 2, s 3 and this one. Starting from s 1 if you apply this input so you are getting these states.

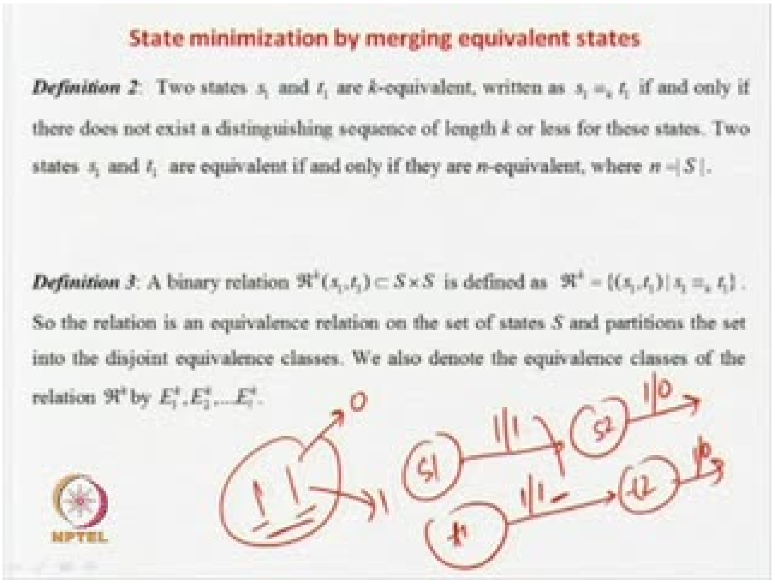
Another set of states if you are applying from t 1, if you apply this same inputs from state t 1 so you are going to get as t 0, t 1, t 2 dot dot dot. And let the output corresponding to this be s 0, s 2, s k minus 1 that is if you are starting from s 1 if you apply x 0 then the output is output s 0, corresponding to this one. That is now you are having contested two, so the output will be this one; again dot dot dot the output will this one. And now again the another run from t 1 then t 2 something like this, this is the another run with the same inputs from state t 1 and the sting output generated is o t 0 o t 2 and this one be the corresponding states that is from state s 1 state t 1 where two states, you are applying this inputs, this is the inputs sequence.

And the output obtained a this one if you starts from S 1 and the output obtained in this one if you start from t 1. So, that is the state s 1 you are applying some inputs x 0, x 1 and you are getting some outputs there nothing but, this one. Now from state t 1 we are applying the same sequence of inputs and you are getting some output they are nothing but this one, they are the output strings.

Now the string this one that is this input is said to be k-length in distinguishable that is there is this sequence of inputs where is in distinguishable for state s 1 and t 1 k-length this obviously you assume that k-length if all the outputs are same; that is, if these outputs and these outputs are always same then you can say that this inputs string cannot distinguish s 1 and t 1. So, in fact what is happening I have one state s 1 and one state t 1 and applying x 0, x 2 dot dot dot. And the outputs I am getting from s 1 are o s 0 o s 2 and dot dot dot. And from t 1 if I apply the same input the output received is o t 2 sorry o t 0, o t 2 and so fore and all these are equivalent. This one is equivalent, this one this one is equivalent to this one and so fore.

Then actually this, this inputs string cannot distinguish between s1 and t 1. It is saying that and now if you take the reverse way then you can say that a k-length string, k-length distinguishing sequence that is if this length I considered as k can distinguish between state s 1 and t 1 if and only if some output is different. Like for example I have say that this is state s 1 and this is state t 1. You apply same this is s 1 and this is t 1. You apply same inputs which is x 0 dot dot dot x k minus 1; same thing and at all these states you travels to if there is at least one state output which is different then you can say that, that inputs string can differentiate s 1 and t 1 but, if the if there is no outputs difference corresponding to the states if you starts from t 1 or s 1for these inputs then you say that this string cannot distinguish between these two states.

(Refer Slide Time: 28:46)



State minimization by merging equivalent states

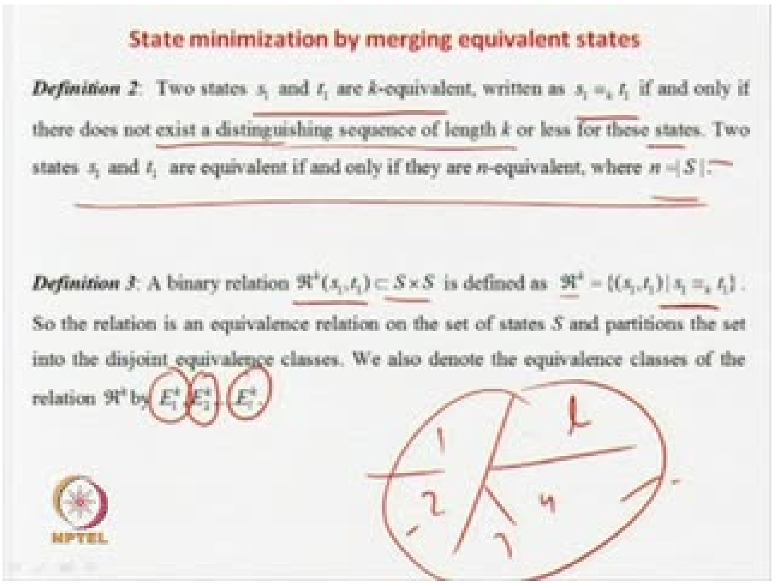
Definition 2: Two states s_1 and t_1 are *k*-equivalent, written as $s_1 \equiv_k t_1$ if and only if there does not exist a distinguishing sequence of length *k* or less for these states. Two states s_1 and t_1 are equivalent if and only if they are *n*-equivalent, where $n = |S|$.

Definition 3: A binary relation $\Re^k(s_1, t_1) \subset S \times S$ is defined as $\Re^k = \{(s_1, t_1) \mid s_1 \equiv_k t_1\}$. So the relation is an equivalence relation on the set of states *S* and partitions the set into the disjoint equivalence classes. We also denote the equivalence classes of the relation $\Re^k$ by $E_1^k, E_2^k, \ldots E_i^k$.

Will these are bit complex definition but when we take an example, you will understand. Very simple let me tell you that this is your say state s 1 and this is your state t 1. You apply 1; here you apply one you get the output 0 kind of a thing and here you apply 1. The output 0 of kind of theory then you says that S 2 and this said t 2. You apply one to get one sorry and here also you apply a one and you get one. Then I can say that if from state s 1 and t 1 if I applying one and a one then 2 length differentiation cannot be obtained 2 length differentiate cannot be obtained. What you mean by 2 length differentiation? That is if I apply one first input second input in sequence. Both the when the output is 0 in the first case and 1 in the second case and the 1 in the second case.

Now in the if you apply in the same thing that is 1 first and 1 second the outputs is again 0. Which is same as this then the output is again same as this. So you can that if the two length input is 1 1 cannot distinguish between these states s 1 and t 1. However, if you can say that, just take a different way, so if I say that if I apply a 1 the output is 1 again 1 1. Here apply a I put 1 this is 0 and apply 1 you get the answer is 1. Then you can say that 2 lengths strain can distinguish between s 1 and t 1. Because, if you apply a 1 the output is 1 if you apply 1 from t 1 output is 1 fine. But the next input if you are from s 1 if you are started your journey from s 1 from this for this second outputs is 0 and if you are started your journey from t 2 then the output is 1then there is a distinguishing. So this 1 1 length 2 length string 1 1 can be distinguish between s 1 and t 1 if this is the case. But, if this is the output corresponding to 1 then you can say that the input string length of 2 s 1 and s 2 that is actually the told by this complex definition which is being seen over here.

(Refer Slide Time: 30:49)



State minimization by merging equivalent states

Definition 2: Two states s_i and t_i are k-equivalent, written as $s_i =_k t_i$ if and only if there does not exist a distinguishing sequence of length k or less for these states. Two states s_i and t_i are equivalent if and only if they are n-equivalent, where $n = |S|$.

Definition 3: A binary relation $\Re^k(s_i, t_i) \subset S \times S$ is defined as $\Re^k = \{(s_i, t_i) \mid s_i =_k t_i\}$. So the relation is an equivalence relation on the set of states S and partitions the set into the disjoint equivalence classes. We also denote the equivalence classes of the relation $\Re^k$ by $E_1^k, E_2^k, \ldots E_r^k$.

So now let see what do you means by two states is s 1 and t 1 your k-length equivalent. So, we are writing this as this if and only if does not the distinguish sequence of k or less than this. That means two states s 1 and t 1 are k-length equivalent if there does not exist any inputs string which is less than or equal to k which can distinguish them. That is distinguish between s 1 and t 1 we require input length of more than k. Now two states are equivalent if and only if this n or the length is equal to the number of states.

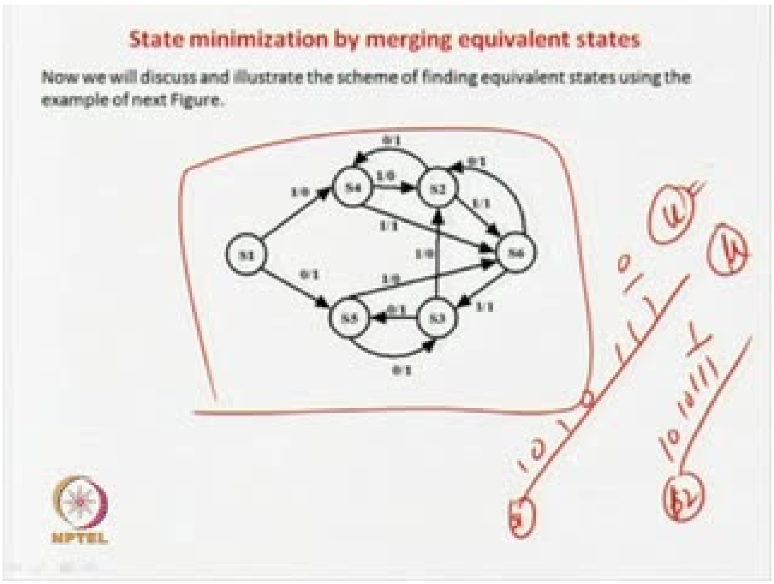
So, for example let us assume that your graph totally has ten states. And now if you can say that there are no string which are less than length. So your graph is state ten states, your graph has ten states; correct? Now you say that for states s 1 and t 1, t 1. All this there is no string whose length is less than or equal to ten which can distinguish this. That means, what the does not exist any string because ten is the maximum number of states in your circuit or strain in your final state machine or S T G; so that means, no way you can distinguish between s 1 and t 1 because why? Why this so? Because the number of states which are present in your system is 10 and all this strings which are length 10 or less cannot distinguish between s 1 and t 1. So, s 1 and t 1 are equivalent states.

Now we can define a binary relation R k s 1 t 1 belonging to s cross s the binary relation as defined as s 1 and t 1 if s 1 is k-equivalent to t 1. That is we can have some equivalent classes like E k 1 that means what you what you say, so what are the E k 1 means? It means all the equivalent states which are k-equivalent. These are all the sets which are another set of states which are k-equivalent. The another set of states which are k-equivalent that means actually given a set of state we can partition it into some l part l-equivalent partitions 1 2 3 4. Some l equivalent partitions with each partition will comprised its which are k-equivalent that means there is no strings of length k or less which can distinguish them.

So, if you make k is equal to s where s is the number of states in the system. So it will be this equivalent class will be nothing but, all these things equivalent states. In other words, so if you find that if you say that there are n numbers of states in a main machine in a Finite State Machine.

And for some state s 1 and s 2 that does not exists in any string of length s or less which can distinguish them that means they cannot distinguish your machine. So, they become your equivalent states if the total equivalent states some time we can define one equivalent two equivalent three equivalents. So, what do you mean by one equivalent? One equivalent means that these states cannot be distinguish by giving a single input. Two equivalent means this states cannot be distinguish by giving any input sequence of two bit input sequence and so fore. So, in the general way this is actually binary equivalent for up to k so it says that by giving an input of length k or less if you are cannot distinguish between states of class E 1 E 2 E 3 and so fore.

(Refer Slide Time: 33:39)



State minimization by merging equivalent states

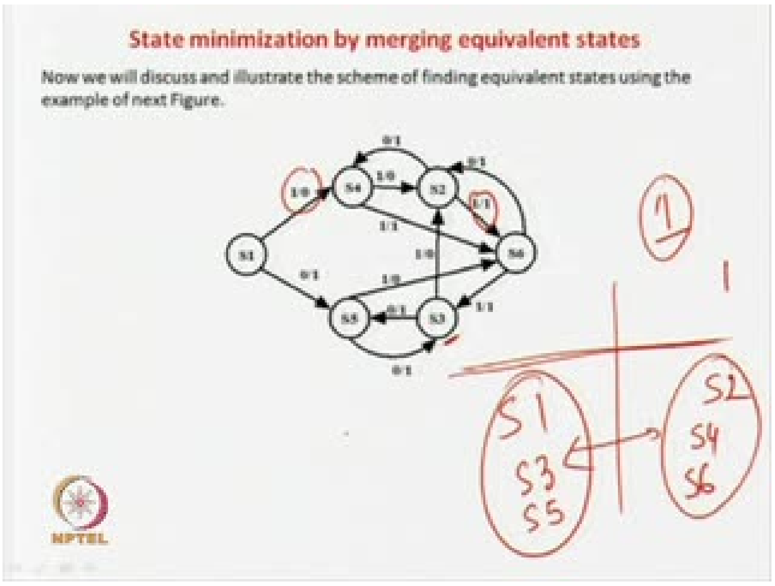
Now we will discuss and illustrate the scheme of finding equivalent states using the example of next Figure.

So, this was some complex mathematical definition, but we will try to give you an example to see what we exactly means then we will see what are these equivalent partitioning means. So again take an example. But, before going to the example just keep this small things in mind. So, what do you mean by equivalent state? Two states are equivalent if you cannot distinguish them. So, how can you distinguish states? We can distinguish states by giving some similarly say two states s 1 and s 2. How do you find out the difference. If you find out the different you have to give some same inputs to both the states.

You have to start from these two states; obviously you have to starts from these two states and this is say state s 1 and this is say state s 2. Then you have to give some similar inputs like 1 0 1 0 1 1 1. Say 1 0 1 0 1 1 1. So, you both are sequence of stage. If you at least find out one state where the output is 0 over here or 1 over here or vice versa. Then you can say that s 1 and s 2 can be distinguished or in other words you can say t 2 or general refer into so you can say that s 1 and t 2 can distinguish. But, if you find out that up to going by up to k-length inputs this number of input is 10 k then you find out that whatever output is given if my journey start form s 1 and the same outputs I am sorry whatever outputs saying getting if I start my journey from s 1.

Same kind of outputs I am getting if I start my journey from s 2 up to inputs of length k and in simply mean that k-length inputs I cannot distinct with this 2 states. And if this case is equal to the total number of states in your system then this two becomes equivalent state because, nobody can distinguish them. So, these are the two some key rules you have to keep in mind. Because, we can distinguish states only by the outputs for similar inputs different outputs for similar inputs. So, this is your circuits; so let me just say, let us say state S 1 and state say S 2.

(Refer Slide Time: 35:06)



State minimization by merging equivalent states

Now we will discuss and illustrate the scheme of finding equivalent states using the example of next Figure.

So now let us say what are that mean. So, if I say that state S 1 and S 2 what one bit different one sequence distinguishable how? So, from S 1 if I apply a one you get a 0. But from S 2 we apply 1 i get 1. So, very easily you can say that S 1 and S 2 are one bit or 1 string not equivalent. That is we have S 1 here and S 2 here. So, if you apply a 1 over here you get a 0 as answer; from S 2 if you apply a 1 you get a 1 and so S 1 and S 2 can be distinguished by one bit input that is 1.

Now very interestingly if you say S 1 and S 3. So, if you take S 1 and S 3 you see from S 1 if I apply a 1 we go to a 0 from S 1 also if I apply a 1, you get a 0 you can say this; from S 3 if I apply a 1 sorry from S 3 if I apply a sorry from S 3 if I apply 1 I get 1 and go somewhere the output is 0. So, from S 1 also if I apply a 1 the output is 0 and I go to state S 4. So, by applying from S 1, if I apply a 1 I get 0 if from S 3, if I also apply a 1 I get 0 ok 1. Now also we see the over way if I apply 1 is this S 1 and S 3 they cannot be distinguish. So now let us try with 0, so from S 1 if you apply a 0, we get the output 1 and from S 3 if I apply 0, we can also get output S 1. So, from S 1 if you apply a 0 you get a 1 and from S 3 also if you apply a 0 you a get 1.

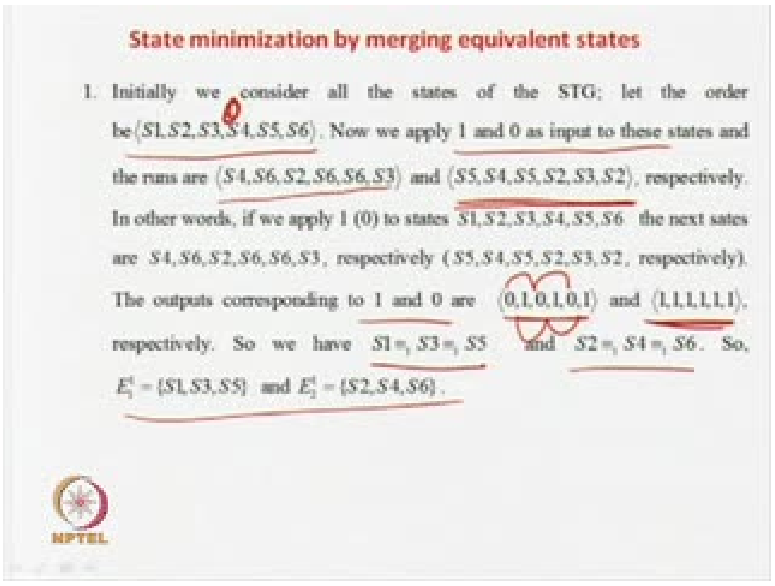
So, S 1 and S 2 cannot be distinguish by a single bit input. So you have to, you can put S 3 over here. Similarly you can also check for S 5. Now if you see S 5; so now we can check S 5. So if you take S 5; so, from S 5 if you apply 0 1 you get a 0. Same thing in case of this. Now if you apply a 0 from S 1 and also from S 5 if you apply a 0 you will get the output as 0. So, same way S 5 also cannot be distinguish by applying a one bit see one bit input sequence. So they are all. So, you can say that S 1 and S 3 and S 5 by equivalent this way you can say this is a equivalent relation if k is equal to 1.

So, you can say that S 1 is equivalent in 1 to S 3 equivalent to 1 to S 5; that means, what by applying one bit inputs sequence or one input sequence S 1 and S 3 and S 5 cannot be distinguish. So, they becomes a equivalent class for string input length of one so that is what is being told a binary relation this is the defines as this so here k is equal to 1.

So that is as an equivalent class on the set of states and the partition on the sets into disjoin equivalent classes for some given ks. So here k is equal to one. So if you consider k equal to 1 then we can say that S 1 S 3 S 5 make an equivalent class similarly you find that S 3 S 4 and S 6 will also make an equivalent class. I am not (( )) you can easily find out that. So, by inputs strings of length 1 S 1 S 3 and S 5 and S 2 S 4 and S 6 are actually equivalent classes.

So, they cannot be distinguished by giving an equivalent by from equivalent I mean S 1 and S 3 and S 5 and S 2 and S 4 and S 6. They cannot be differentiated I mean you cannot differentiate in between this three and also you cannot differentiate in between this three by applying a single bit input single length input but, off course by you can differentiated between two classes by applying any single bit inputs like as I told you S 1 can be easily differentiated from S 2 because, from S 1 if I apply a 1 I get the answers 0 and from S 2 if I apply a 1, you get the answer S 1 output as also 1 and a 0 as difference so very easily you can distinguish between this by applying a single sequence of inputs. So, you can distinguish between these two classes but, inside the cluster you cannot define using single bit inputs so that is what is the actually written over here.

(Refer Slide Time: 39:08)



State minimization by merging equivalent states

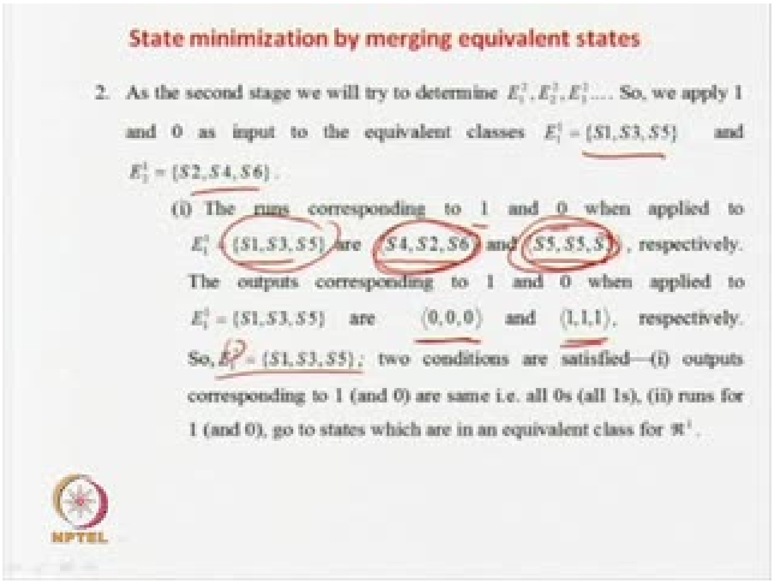
1. Initially we consider all the states of the STG; let the order be $\langle S1, S2, S3, S4, S5, S6 \rangle$. Now we apply 1 and 0 as input to these states and the runs are $\langle S4, S6, S2, S6, S6, S3 \rangle$ and $\langle S5, S4, S5, S2, S3, S2 \rangle$, respectively. In other words, if we apply 1 (0) to states $S1, S2, S3, S4, S5, S6$ the next sates are $S4, S6, S2, S6, S6, S3$, respectively ($S5, S4, S5, S2, S3, S2$, respectively). The outputs corresponding to 1 and 0 are $\langle 0,1,0,1,0,1 \rangle$ and $\langle 1,1,1,1,1,1 \rangle$, respectively. So we have $S1 \equiv_1 S3 \equiv_1 S5$ and $S2 \equiv_1 S4 \equiv_1 S6$. So, $E_1^1 = \{S1, S3, S5\}$ and $E_2^1 = \{S2, S4, S6\}$.

So, how do you start? Initially we consider all these states to be equivalent now we apply 0 and 1 inputs to all the states. So, if you apply 1 to these things you are from S 1 you will go to S 4 kind of a thing now if you apply initially what do you do initially you consider all the states to be equivalent S 1 S 2 S 3 S 4 S 5 S 6 all are equivalent. If you apply 1 and 0 to this, so you are going to next state will be S 4 S 6 S 2 S 6 S 6 S 3 corresponding to this if you apply a 1 and S 5 S 4 S 5 S 2 S 3 S 2 this is the next state if you apply 0 to all the state respectively.

So, in other words if you do this these the next states will be this and this respectively the output corresponding to this r mean for example if you apply a 1 to all this. This is your next state and if you apply a 0 to all this state the next state sequences this one and the outputs are nothing but, this and this. So, you can see that these three are that is S 0, S 3 and S 5 output is 0 0 0 they cannot be distinguish and 2 4 and 6 the outputs is 1 1 1 so here you cannot be distinguish. So, you have this one and this one as explain bellow. So, now this is your class S 1 S 3 S 5, S 2 S 4 and S 6.

So, if you apply a 1 to this state so you are going to state S 5 S 4 S 5 whatever for outputs are all 1. So, by applying a 1 you cannot generate distinguish between S 1 S 2 S 3 by applying 1 to the state in the first row you cannot distinguish between anything sorry; I am sorry if you apply a 0 over here the outputs are all 1 that you can easily verify. So, this one outputs the outputs corresponding to 1 and 0 over this. So, these are your outputs, so if you just see that if I apply a 0 over this. So, the outputs corresponding to this will be something like this and the even if you apply a 1 so this is the outputs here you can distinguish between the first position second position second third but, you cannot distinguish between first third and fifth second fourth and sixth, so this way you are getting this equivalent class.

(Refer Slide Time: 41:05)



State minimization by merging equivalent states

2. As the second stage we will try to determine E_1^2, E_2^2, E_3^2.... So, we apply 1 and 0 as input to the equivalent classes $E_1^1 = \{S1, S3, S5\}$ and $E_2^1 = \{S2, S4, S6\}$.

   (i) The runs corresponding to 1 and 0 when applied to E_1^1 $\{S1, S3, S5\}$ are $S4, S2, S6$ and $S5, S5, S3$, respectively. The outputs corresponding to 1 and 0 when applied to $E_1^1 = \{S1, S3, S5\}$ are $\langle 0,0,0 \rangle$ and $\langle 1,1,1 \rangle$, respectively. So, $E_1^2 = \{S1, S3, S5\}$; two conditions are satisfied—(i) outputs corresponding to 1 (and 0) are same i.e. all 0s (all 1s), (ii) runs for 1 (and 0), go to states which are in an equivalent class for $\Re^1$.

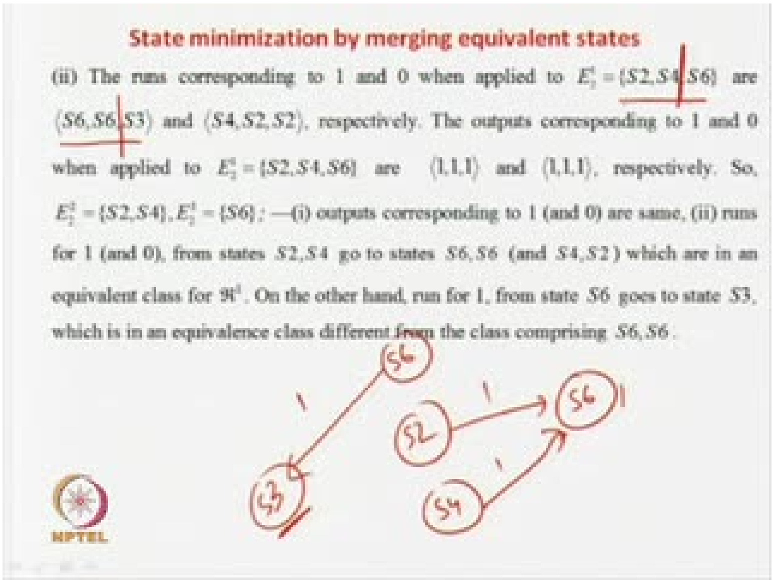
Now you have to keep on doing it, so now your classes are this and this. Now we are again try to repeat to this one. So now, again what do you do you? Now you take these two states and will now repeat this. So, if you the run corresponding to 1 and 0 when apply to this there S 1, S 3 and S 5 you apply 1. So, the next state you can find out from the figure are S 4 S 2 and S 6 and if you apply a 0, the outputs the next state corresponding to these are S 5 S 5 and S 3 and the output corresponding to these are this and this.

So now you can see, you can see that from if you apply a 1 over here, so from S 1you go to S 4 from S 3 you go to S 2 and from S 5 you go to S 6 and the outputs are all 0s and again if you apply a 0 from this you go to S 5, S 5, and S 3 and outputs are all 1s. So, again actually still by actually applying this second bit as input first time you have already seen that S 1, S 3 and S 5 cannot be distinguishing by a single bit inputs single sequence of inputs. Now you are applying 2 bits inputs 2 bits second sequence you are applying, but still the outputs set is S 4, S 2 and S 6 so this sets our states has been equivalent from the previous and already we have seen.

And if you are applying a 0 then the next sequence of states from this one and next set of states are S 5 S 5 and S 3. So they are again equivalent in themselves and the outputs are also similar. So, even by applying a two bit input S 1, S 3, and S 5 cannot be distinguished. So, you can write S 1 2 that is length of strings 2 cannot distinguish between S 1 and S 3 and S 5; so similarly, you can write for this one so that is what is the idea. So what we have seen? That is if you apply one bit input you cannot distinguish between S 1 S 3 and S 5.

But, you can distinguish between S 1 S 2 and S 1, S 4 something like this but, even I applying a second bit inputs or second sequence of input you cannot distinguish between S 1 S 3 and S 5 because the next state corresponding to 1 is this. The next state corresponding to E 1 1 when the input is 0 is this and again this clusters are also comprising of equivalent state and the outputs are similar like this case. So, they cannot be distinguish so you can easily write that S 1 and S 2 and S 5, they cannot distinguish among themselves when you are applying this second bit of inputs.

(Refer Slide Time: 43:08)



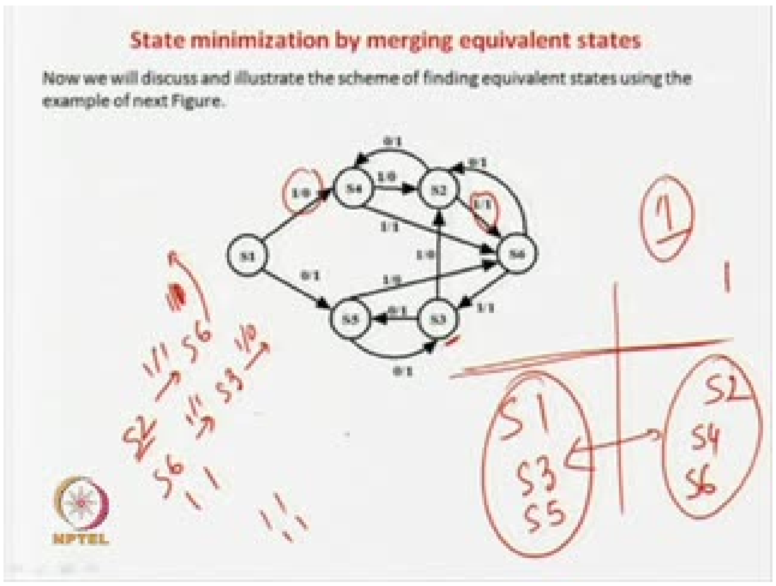
State minimization by merging equivalent states

(ii) The runs corresponding to 1 and 0 when applied to $E_2^1 = \{S2, S4, S6\}$ are $\langle S6, S6, S3 \rangle$ and $\langle S4, S2, S2 \rangle$, respectively. The outputs corresponding to 1 and 0 when applied to $E_2^1 = \{S2, S4, S6\}$ are $\langle 1,1,1 \rangle$ and $\langle 1,1,1 \rangle$, respectively. So, $E_2^2 = \{S2, S4\}, E_3^2 = \{S6\}$; —(i) outputs corresponding to 1 (and 0) are same, (ii) runs for 1 (and 0), from states $S2, S4$ go to states $S6, S6$ (and $S4, S2$) which are in an equivalent class for $\Re^1$. On the other hand, run for 1, from state $S6$ goes to state $S3$, which is in an equivalence class different from the class comprising $S6, S6$.

Now we will try for the second cluster we have. So, second cluster was S 2 S 4 and S 6. Now again we apply 1 and 0 to this. So if you apply a 1 so the next state is S 6 and S 3. So now you can see that from the previous plan we know that S 6 and S 3 can be extended. So, what a basically we have done from S 2 we had S 4 and we have S 6; 2 4 6; 2 4 6. So, the next state corresponding to S 2 is nothing but, S 6; from S 4 is next it is nothing but, S 6; but, S 6 is next state is S 3 and for that we have to apply a 1. So, if you apply a 1 so these are the next state. So, you know that S 3 can easily be distinguished from S 6 because S 1, S 6 and S 5 are equivalent and S 2, S 4 and S 6 are equivalent. So, S 3 and S 6 are equivalent not equivalent S 3 and S 6 are not equivalent.

So, by using two bit sequence distinguish between set S 2, S 4 and S 6. So, what you apply so what do you do? So, first you apply say for example, S 2 and sorry you can distinguish between these 2 because from S 2 and S 4 you are going to state S 6, which are equivalent because S 6 is equivalent with itself but, if you want to distinguish between S 2 and S 4 with S 6 it can be very easily done.

(Refer Slide Time: 44:23)



State minimization by merging equivalent states

Now we will discuss and illustrate the scheme of finding equivalent states using the example of next Figure.

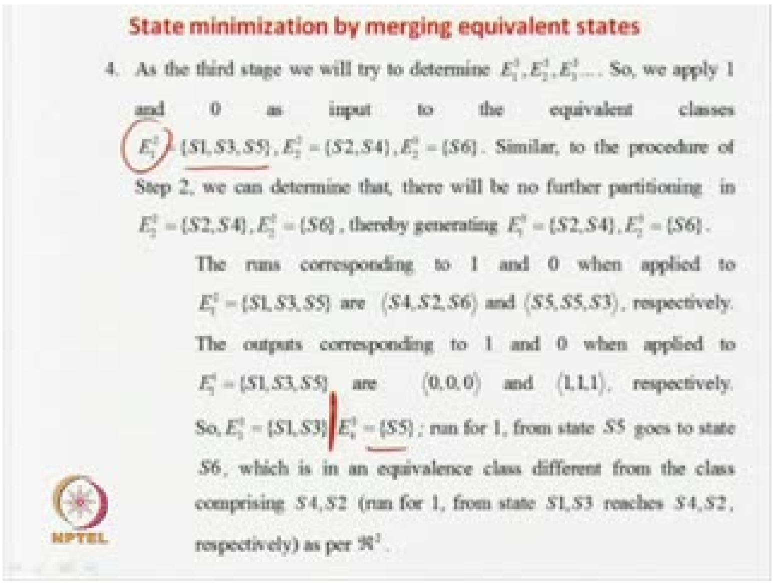
Because we can just see the figure so in this case if you want to apply from S 2 if you apply 1 you go to S 6. So now from S 6 if you apply a 1 you go to state S 3. So from S 6 S 2. If you apply a 1 you go to S 6 in the next will you apply go to S 3. Now from S 6 if you apply a 1, so you apply 1 you go to state S 3 and from S 3 if you apply a 1 you go to state S 2.

So, what happens basically is that by using two bit sequence. So, if you are that means if you apply the first bit sequence from S 2. So, you will be in either S 2, S 4 or S 6 only. So S 2 is applying a 1, you first bit input you go to S 6 or if you apply a 0, you go to S 4.

So from S 2 if you apply a 1 you go to state S 6 and from S 6 if you apply a 1, so you go to state S 3.

So the output is always 1 in this case into apply. So, from here if you apply a 1 the output is 1 from S 6 also if you apply the input is 1. Now from S 3 what do you do from S 3 you already know from S 2 you apply a 1; from S 3 you apply 1 the answer you get is 0. And from S 6 if you apply a 1 the answer is 1 sorry the answer is 1, 1 here. So, this is the distinguisher. So by from S 2 you apply a 1 you go to S 6 the output is 1 when you again if you apply 1 the answer is 1 but, if you starts from S 6 1 output is 1 and then again you apply a 1 the answer is a 1. But if you start from S 6 you apply a 1 you get a 1 go to S 3 again you apply a 1 here you get the answer as 0. So, this one and 0 is different, so by using the input sequence is 1 1, from S 2 you get the output as 1 1 but, if you apply the same sequence 1 1 from S 6 the answer you get output you get this 1 0 and so it can be distinguished.

(Refer Slide Time: 46:46)



State minimization by merging equivalent states

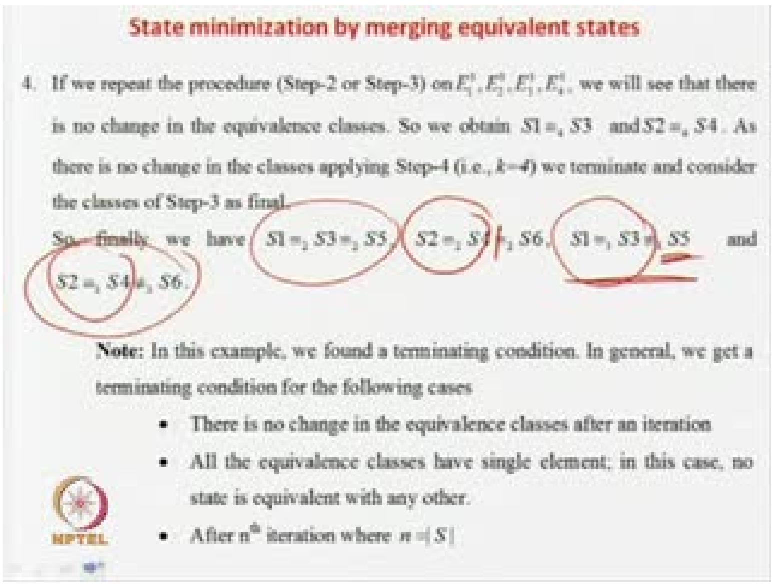
4. As the third stage we will try to determine $E_1^3, E_2^3, E_3^3 \ldots$ So, we apply 1 and 0 as input to the equivalent classes $E_1^2 = \{S1, S3, S5\}$, $E_2^2 = \{S2, S4\}$, $E_3^2 = \{S6\}$. Similar, to the procedure of Step 2, we can determine that, there will be no further partitioning in $E_2^2 = \{S2, S4\}$, $E_3^2 = \{S6\}$, thereby generating $E_2^3 = \{S2, S4\}$, $E_3^3 = \{S6\}$.

The runs corresponding to 1 and 0 when applied to $E_1^2 = \{S1, S3, S5\}$ are $(S4, S2, S6)$ and $(S5, S5, S3)$, respectively. The outputs corresponding to 1 and 0 when applied to $E_1^2 = \{S1, S3, S5\}$ are $(0,0,0)$ and $(1,1,1)$, respectively. So, $E_1^3 = \{S1, S3\}$ $E_4^3 = \{S5\}$; run for 1, from state $S5$ goes to state $S6$, which is in an equivalence class different from the class comprising $S4, S2$ (run for 1, from state $S1, S3$ reaches $S4, S2$, respectively) as per $\Re^2$.

So, you can see that by applying two bit sequence I can distinguish between states S 2, S 4 with S 6 that you can do. So, your next set becomes something like this S 2 S 4 is one set and S 6 is other set. So S 2 and S 4 and S 6 can be distinguish S 2 and S 4 cannot be distinguish with each other by two bit input but S 2 and S 4 can be distinguish with S 6 by using two bit input that is 1 1. And we have found out that by node to be inputs we can distinguish between S 1 S 3 and S 5 as similarly, with node two bit input sequence you can distinguish between S 1 and S 4. So this is how you can just look at this slide and in third stage, what we can do is that? We will again try in the third stage now initially second stage you had S 1, S 3 and S 5.

Now we can easily see that if apply the some inputs which we applied we can be actually distinguish between the S 1, S 3 and S 5 and what we can easily find out not going to in details how we can do that as similar case we have seen for this one. As similar case we have seen for this one how you are differentiated between S 2 and S 4 in one class and S 6 in another class. Similar, way we can find out that using sequence of 3 bit runs you can distinguish between S 1, S 3 in one case and S 5 in the another case that you can easily find out.

(Refer Slide Time: 47:30)



State minimization by merging equivalent states

4. If we repeat the procedure (Step-2 or Step-3) on $E_1^4, E_2^4, E_3^4, E_4^4$, we will see that there is no change in the equivalence classes. So we obtain $S1 \equiv_4 S3$ and $S2 \equiv_4 S4$. As there is no change in the classes applying Step-4 (i.e., $k=4$) we terminate and consider the classes of Step-3 as final.

   So, finally we have $S1 \equiv_2 S3 \equiv_2 S5$, $S2 \equiv_2 S4 \neq_2 S6$, $S1 \equiv_3 S3 \neq_3 S5$ and $S2 \equiv_3 S4 \neq_3 S6$.

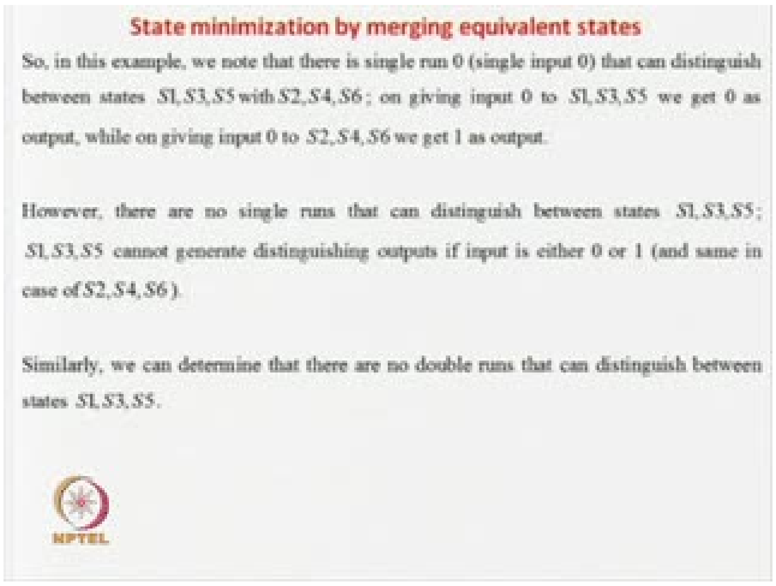
Note: In this example, we found a terminating condition. In general, we get a terminating condition for the following cases

- There is no change in the equivalence classes after an iteration
- All the equivalence classes have single element; in this case, no state is equivalent with any other.
- After n^th^ iteration where $n = |S|$

So, over all we will fine that with 2 sequence number we use that S 1 and S 3 and S 5 cannot be distinguish the, but with 3 bit sequence you can distinguish between S 1 and S 3 and S 1 and it can be distinguish between this but, again you will find out the S 2 and S 4 sorry this is not equivalent. So, we can find out the sorry about that, so in this case again will find out that S 2 and S 4 cannot be distinguish between input sequence but, S 1 and S 2 cannot be distinguish with S 6 in a two bit inputs sequence that already I have seen but, for distinguishing between S 1 and S 3 you required a 3 bit input sequence.

So, of this is again and very very obvious that S 2 and S 4 will be easily distinguish between S 6 with the 3 bit it example. So, just you have to repeat this step mean with already we have distinguish between S 1 and S 3 with S 5 in 3 steps in 3 input sequence, that what is being shown over here and we have distinguish between S 2 and S 4 in two bit inputs sequence. So, obviously we will also hold.

(Refer Slide Time: 48:38)



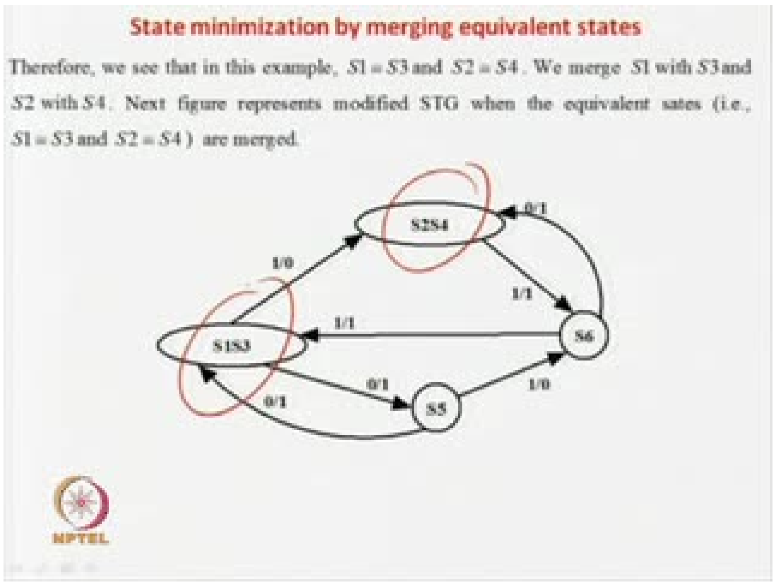
State minimization by merging equivalent states

So, in this example, we note that there is single run 0 (single input 0) that can distinguish between states $S1, S3, S5$ with $S2, S4, S6$; on giving input 0 to $S1, S3, S5$ we get 0 as output, while on giving input 0 to $S2, S4, S6$ we get 1 as output.

However, there are no single runs that can distinguish between states $S1, S3, S5$; $S1, S3, S5$ cannot generate distinguishing outputs if input is either 0 or 1 (and same in case of $S2, S4, S6$).

Similarly, we can determine that there are no double runs that can distinguish between states $S1, S3, S5$.

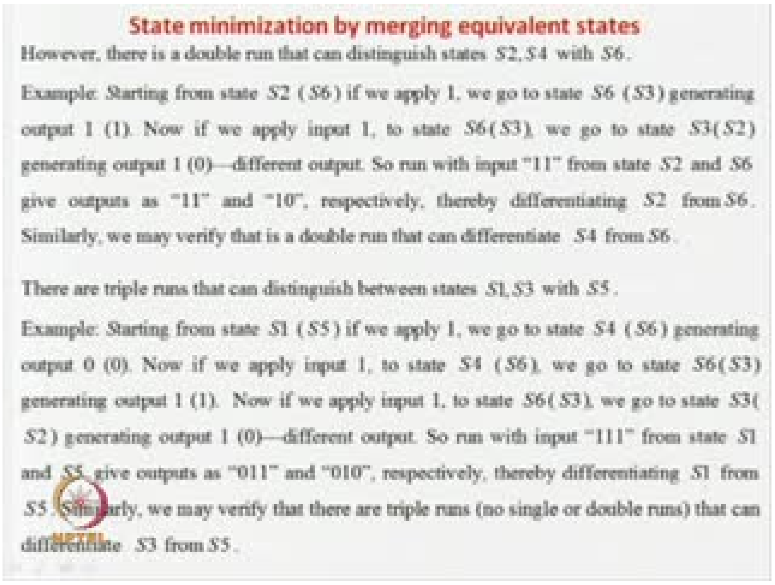
(Refer Slide Time: 48:50)



State minimization by merging equivalent states

Therefore, we see that in this example, $S1 = S3$ and $S2 = S4$. We merge $S1$ with $S3$ and $S2$ with $S4$. Next figure represents modified STG when the equivalent sates (i.e., $S1 = S3$ and $S2 = S4$) are merged.

(Refer Slide Time: 49:00)



State minimization by merging equivalent states

However, there is a double run that can distinguish states $S2, S4$ with $S6$.

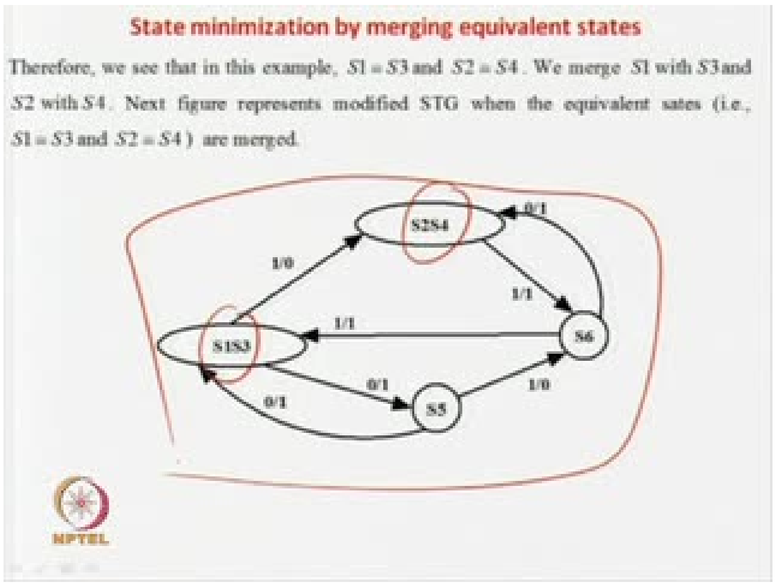
Example: Starting from state $S2$ ($S6$) if we apply 1, we go to state $S6$ ($S3$) generating output 1 (1). Now if we apply input 1, to state $S6$($S3$), we go to state $S3$($S2$) generating output 1 (0)—different output. So run with input "11" from state $S2$ and $S6$ give outputs as "11" and "10", respectively, thereby differentiating $S2$ from $S6$. Similarly, we may verify that is a double run that can differentiate $S4$ from $S6$.

There are triple runs that can distinguish between states $S1, S3$ with $S5$.

Example: Starting from state $S1$ ($S5$) if we apply 1, we go to state $S4$ ($S6$) generating output 0 (0). Now if we apply input 1, to state $S4$ ($S6$), we go to state $S6$($S3$) generating output 1 (1). Now if we apply input 1, to state $S6$($S3$), we go to state $S3$($S2$) generating output 1 (0)—different output. So run with input "111" from state $S1$ and $S5$ give outputs as "011" and "010", respectively, thereby differentiating $S1$ from $S5$. Similarly, we may verify that there are triple runs (no single or double runs) that can differentiate $S3$ from $S5$.

So similarly, you have to just extrapolate this manner and you will find out that we can just you can find out that I mean just have to repeat this steps, so which is being already shown I mean discussed in this case so just have to explore this lights and you can find out that S 3 and S 4 cannot be distinguish and S 1 and S 3 cannot be distinguish. Because, you will find out that the circuit initially this system initially had how many steps this system initially had 1, 2, 3, 4, 5, 6. So you can find out that by executing this steps which I have told you is written over here. So, say that by applying two bit inputs sequence we can distinguish S 2, S 4 together we can distinguish S 6 but, with any two bit sequence you cannot distinguish between S 1, S 3 and S 5 but, now applying a 3 bit input sequence.

(Refer Slide Time: 49:41)

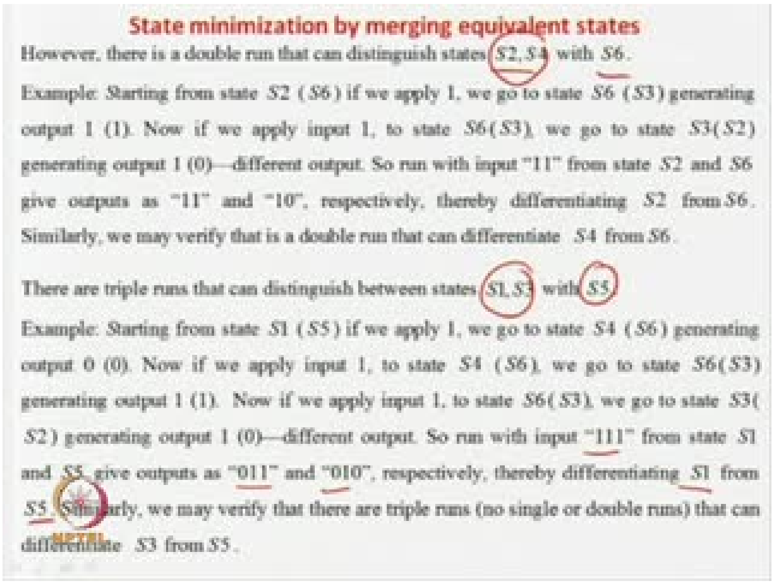


State minimization by merging equivalent states

Therefore, we see that in this example, $S1 \equiv S3$ and $S2 \equiv S4$. We merge $S1$ with $S3$ and $S2$ with $S4$. Next figure represents modified STG when the equivalent sates (i.e., $S1 \equiv S3$ and $S2 \equiv S4$) are merged.

You can find out that I can distinguish between S 1 and S 3 in 1 slide and S 3 in another side. But, if you apply this for 5 6 states, so 6 numbers of times, you can find out that still S 2 and S 4 will be equivalent and S 1 and S 3 will be equivalent. So, this will be your equivalent state they cannot be distinguish because now number steps are 6. So, we know 6 more no such strings of length 6 can distinguish between S 2 and S 4. Similarly, no strings of length 6 can be distinguished between S 1 and S 3. So similarly, these things will be remaining equivalent and others strings states will be distinguishing. Now you merge S 3 and S 4, S 1 and S 3 and the graph will look something like this. So that is what have been written over here.

(Refer Slide Time: 50:15)



State minimization by merging equivalent states

However, there is a double run that can distinguish states $S2, S4$ with $S6$.

Example: Starting from state $S2$ ($S6$) if we apply 1, we go to state $S6$ ($S3$) generating output 1 (1). Now if we apply input 1, to state $S6$($S3$), we go to state $S3$($S2$) generating output 1 (0)—different output. So run with input "11" from state $S2$ and $S6$ give outputs as "11" and "10", respectively, thereby differentiating $S2$ from $S6$. Similarly, we may verify that is a double run that can differentiate $S4$ from $S6$.

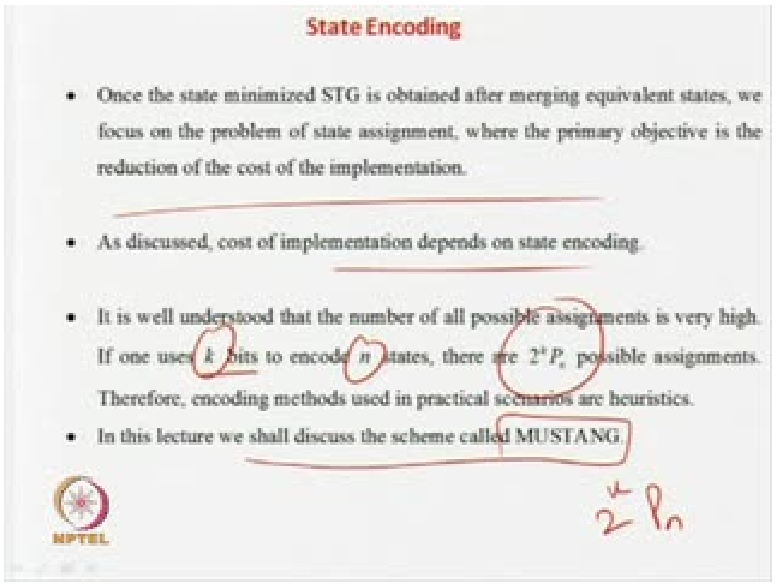
There are triple runs that can distinguish between states $S1, S3$ with $S5$.

Example: Starting from state $S1$ ($S5$) if we apply 1, we go to state $S4$ ($S6$) generating output 0 (0). Now if we apply input 1, to state $S4$ ($S6$), we go to state $S6$($S3$) generating output 1 (1). Now if we apply input 1, to state $S6$($S3$), we go to state $S3$($S2$) generating output 1 (0)—different output. So run with input "111" from state $S1$ and $S5$ give outputs as "011" and "010", respectively, thereby differentiating $S1$ from $S5$. Similarly, we may verify that there are triple runs (no single or double runs) that can differentiate $S3$ from $S5$.

So, there is the there is double run and that can distinguish between this and this. So, that is by using a string of length two you can distinguish between this but, any string of any link cannot distinguish between S 2 and S 4. That is cannot be possible. That is they are actually S 2 and S 4 are equivalent states. Now says that there are triple run that can be distinguish between this and this; so that also how it is done we can easily see that is run with 1 1 1 from state 1 and S 5 gives output this and this respectively S 1 from S 5.

So, actually they are saying that you are required 3 input strings and 3 bit input strings into distinguish between S 1 and S 3 with S 5 but, two bit input strings can be differentiate between S 2 and S 4 with S 6. But, any number of strings of any length cannot be distinguishing in between S 2 and S 4 and S 1 and S 3 they will recovery become equivalent states and finally you have this graph.

(Refer Slide Time: 51:05)



State Encoding

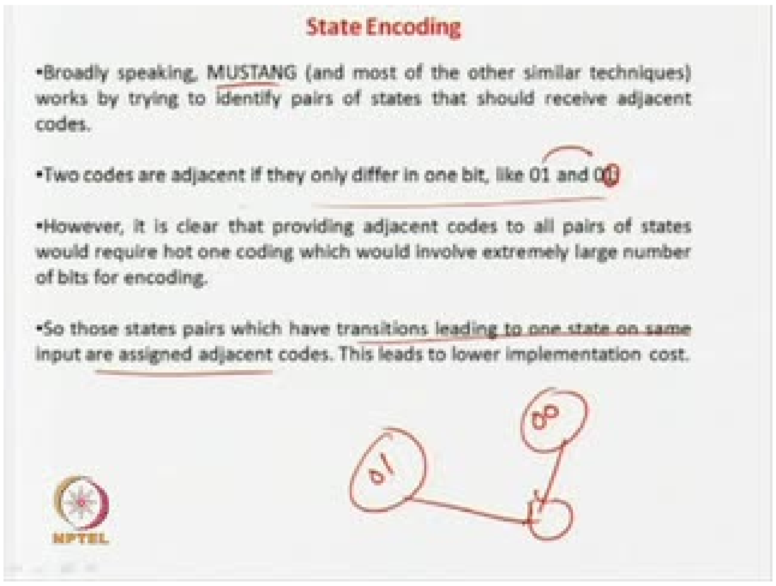
- Once the state minimized STG is obtained after merging equivalent states, we focus on the problem of state assignment, where the primary objective is the reduction of the cost of the implementation.
- As discussed, cost of implementation depends on state encoding.
- It is well understood that the number of all possible assignments is very high. If one uses k bits to encode n states, there are $2^k P_n$ possible assignments. Therefore, encoding methods used in practical scenarios are heuristics.
- In this lecture we shall discuss the scheme called $\boxed{\text{MUSTANG}}$.

$2^k P_n$

So, once we have this minimize so in this case only 4 states in two two states could be merged. So, your graph structure become similar simple then what do you have to do is that, then you have to merged this states to let you diagram will be become small? Then we have to go for state encoding. As I told you so if you required k this bits encode the n states then what is possibility. So, if they k bits are used to represents your states to states used k bits used to encode. So, the 2 to power of k possibilities will be there and if there n states so the permutation this number will be very very high. So, in this case what is the best State Encoding and which one is apply to which is the very difficult property to solve.

So, in this lecture discuss about the heuristics with call the MUSTANG heuristics; it is very simple heuristic which will tell you which state encoded to choose and apply to which but, again this will be a non-optimal solution may not be an optimum solution why because the idea is very simple because I mean the state space is so large and we will be trying by huge and we will try to find 1 of the somehow, it will and we already discussed in the last class is the problem with heuristics based of optimization, so it will be try to figure out some good part of solution means and we have stopping criteria kind of thing we stop which may not be your optimum situation. So, we will see the heuristic to do this because exploring this bit state space is very time consumed.

(Refer Slide Time: 52:23)



State Encoding

•Broadly speaking, MUSTANG (and most of the other similar techniques) works by trying to identify pairs of states that should receive adjacent codes.

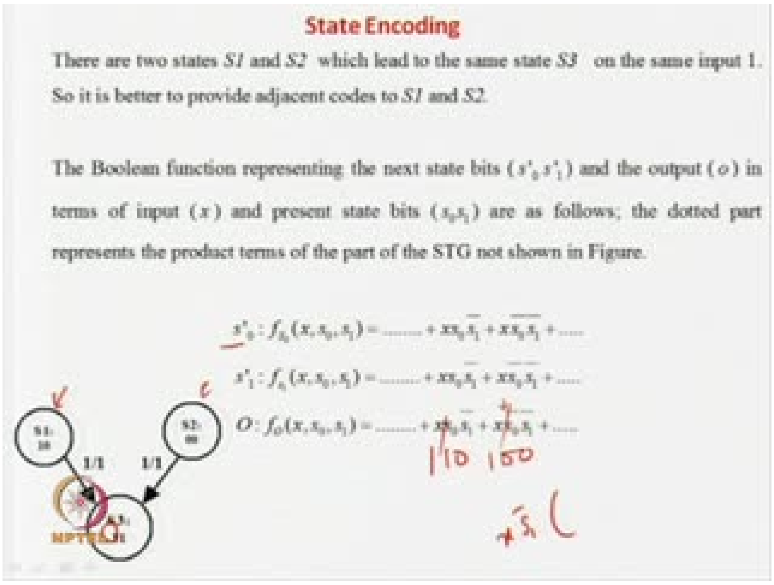
•Two codes are adjacent if they only differ in one bit, like 01 and 00

•However, it is clear that providing adjacent codes to all pairs of states would require hot one coding which would involve extremely large number of bits for encoding.

•So those states pairs which have transitions leading to one state on same input are assigned adjacent codes. This leads to lower implementation cost.

What is the basic idea of MUSTANG? So what it does? Actually if it is finds that there are two states which are leading to the same fanout states then it try to them that differentiate by one bit kind of a thing like say for example, if you the encoding 0 1 and encoding is 0 0 sorry it is 0 0 sorry. So, what is the idea is that, if 2 quotes are adjacent they are if that differentiate one bit so actually one bit are, we always know that if the any encoding or any states or whatever so if they are having single difference. That we know that they are comfortable like 0 1 and 0 0 like 1 0 and say 0 sorry 1 1 then again comfortable with 1; so they are actually for encode adjacent. Now the idea is here is that those states that we have transition leading to the same states are assign adjacent code like for example these two states are there they are going as a same fanout say you mean same I mean what you call adjacent code or comfortable codes to that.

(Refer Slide Time: 53:14)



State Encoding

There are two states *S1* and *S2* which lead to the same state *S3* on the same input 1. So it is better to provide adjacent codes to *S1* and *S2*.

The Boolean function representing the next state bits ($s'_0 s'_1$) and the output (o) in terms of input (x) and present state bits ($s_0 s_1$) are as follows; the dotted part represents the product terms of the part of the STG not shown in Figure.

$$s'_0 : f_{s_0}(x, s_0, s_1) = \ldots\ldots + x s_0 \bar{s}_1 + x \bar{s}_0 \bar{s}_1 + \ldots\ldots$$

$$s'_1 : f_{s_1}(x, s_0, s_1) = \ldots\ldots + x s_0 \bar{s}_1 + x \bar{s}_0 \bar{s}_1 + \ldots\ldots$$

$$O : f_O(x, s_0, s_1) = \ldots\ldots + x s_0 \bar{s}_1 + x \bar{s}_0 \bar{s}_1 + \ldots\ldots$$

So, how to that lead what is that lead to good idea less number of states. Let us take with example. So, MUSTANG simple light we will find out these states in your Finite State Machine which is having a common fanout. Say if these two states are having a common fanout you try to assign some State Encoding to this which are having code adjacent; that is, in case 0 1 sorry 1 0 and this 0 is the one bit inputs so try to do that. Now doing this how its advantages already, we know that if there are two states like this or any 2 bits which are comfortable very nearby others, so they can minimize.

So, this is only one bit difference; the same philosophy was used in tabular matter for finding the prime bit inputs. So, if you have 2 mean terms only one literal are different we can get easily optimization like say for example x bar y plus x y. So, there is one bit so difference is y prime so we can take y x same plus y prime plus. Why because, of 1 it will be easily get reduce. So, is if find out the cases in terms we have one bit literal difference they can be must to do this one. So, let up see this is the case; so, S 1 and S 2 so we are encoding at 1, 0 this one is encoding is that encoded as 0, 0 and the output is something like that.

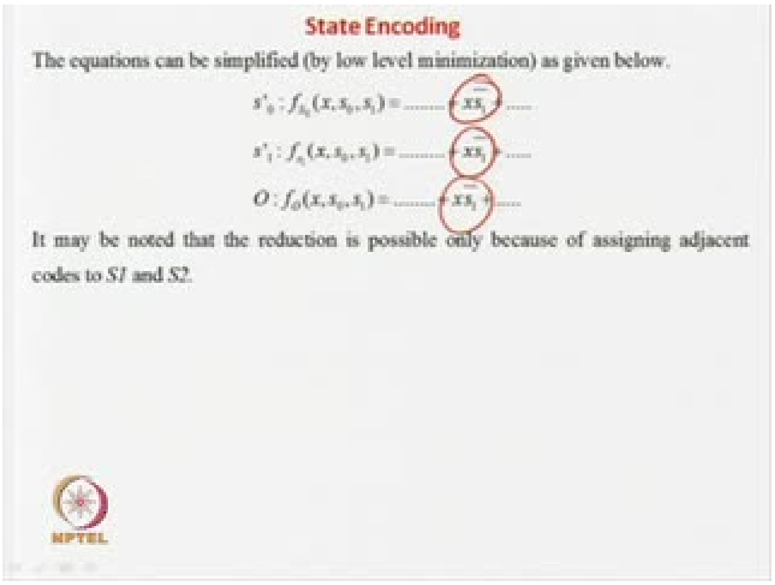
So, you can write this and lot of other states will be there the function of S 0 prime is this one. So, x the input is 1 of the state is 1 0 and again this state 0, 0. So, this is the input is x. Input is 1 present state is 1 0. So, present state input is 1 present state is 1, you get answer as 1. This is the this one. And again you can see over here set is the S 1 prime.

So, if the input is 1 input is 1 and the present state is 0 0, again output is 1, that is what is being reflected over here.

So, just actually we having the dotted line we can have other state to do this. So, but, what idea is showing is that just for if you want represent this small sub graph, so what it is saying, so it is saying that if your input is 1 that is input is 1 and your present state is 1 0. So, the next state this being that is S 0 prime is 1 also if it is say that present state is 0, 0 input is 1 then the next state is also 1. So, this is 1, 1, 0, so this one, 1, 0 corresponding to this term this is 1 this is some in terms and again is 1 0. So, this is 1, 0, and 0 so, this term is actually corresponding to this type and what else this corresponding to 1. 0 1 and this corresponds to 1, 1, 0 kind of input again.

Now you can see over here that sorry this is about the output string you can also check over here, so what is the output. So, the output is 1 in the case when the input is 1 and the state is 0 and the output is 1 in this case said. So, in both the cases the output is 1. Now we can easily say that they can minimize this because there is one bit difference over here. So, even we can easily minimize the output is x a x 1 prime and then this x is this will get cut off and similar that is being shown in the next line.

(Refer Slide Time: 56:23)



State Encoding

The equations can be simplified (by low level minimization) as given below.

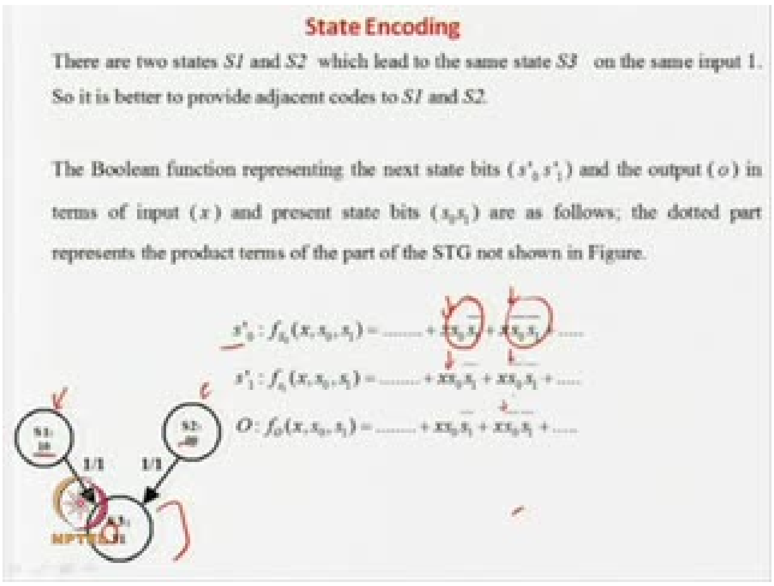
$$s'_0 : f_{s_0}(x, s_0, s_1) = \ldots\ldots + \bigcirc{x\bar{s}_1} + \ldots$$

$$s'_1 : f_{s_1}(x, s_0, s_1) = \ldots\ldots + \bigcirc{x\bar{s}_1} + \ldots$$

$$O : f_O(x, s_0, s_1) = \ldots\ldots + \bigcirc{x\bar{s}_1} + \ldots$$

It may be noted that the reduction is possible only because of assigning adjacent codes to *S1* and *S2*.

(Refer Slide Time: 56:26)



State Encoding

There are two states $S1$ and $S2$ which lead to the same state $S3$ on the same input 1. So it is better to provide adjacent codes to $S1$ and $S2$.

The Boolean function representing the next state bits ($s'_0 s'_1$) and the output (o) in terms of input (x) and present state bits ($s_0 s_1$) are as follows; the dotted part represents the product terms of the part of the STG not shown in Figure.

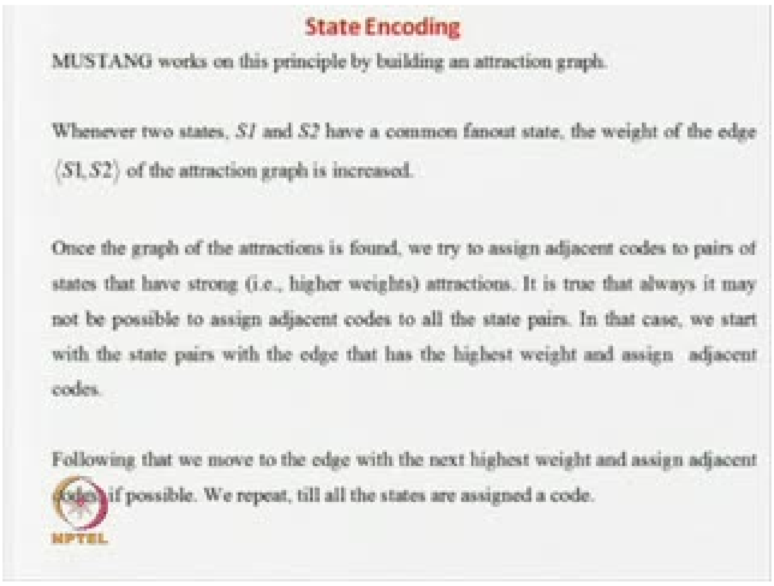
$$s'_0 : f_{s_0}(x, s_0, s_1) = \ldots\ldots + x s_0 \bar{s}_1 + x \bar{s}_0 \bar{s}_1 + \ldots\ldots$$

$$s'_1 : f_{s_1}(x, s_0, s_1) = \ldots\ldots + x \bar{s}_0 s_1 + x \bar{s}_0 s_1 + \ldots\ldots$$

$$O : f_O(x, s_0, s_1) = \ldots\ldots + x \bar{s}_0 s_1 + x \bar{s}_0 s_1 + \ldots\ldots$$

So, you can easily minimize this, this structure will get minimize because there is 1 bit difference. So, here one bit difference in this case also this is one bit difference and this case also one bit difference. So, there being a one bit difference, so this minimization can be possible; so in other words, whenever that is the idea of MUSTANG. So, if you have similar state and this two states leading to this you should have a one bit difference. So, if you have a one bit difference so the state representations will be quite simple in this case. So, because in this corresponds to 0, 1 and this corresponds to 0, 0 in the similarly in only cases, now we can have to able do a, what you say a minimization.

(Refer Slide Time: 57:02)



State Encoding

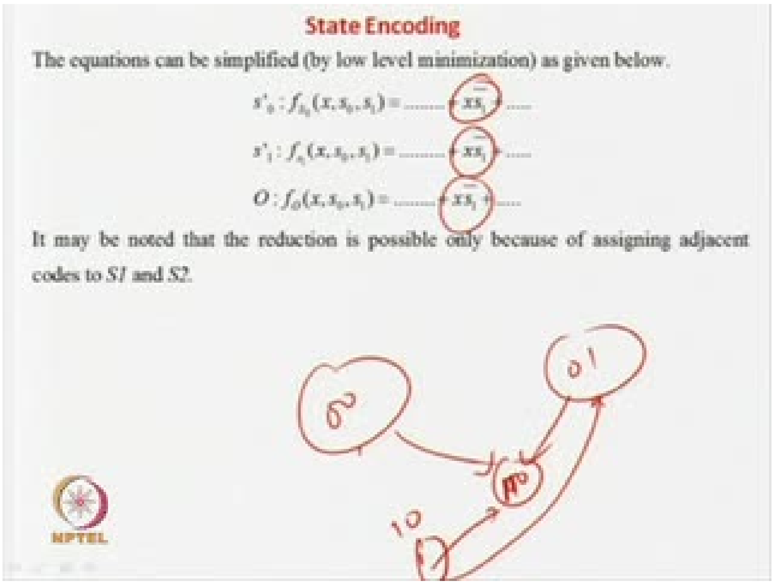
MUSTANG works on this principle by building an attraction graph.

Whenever two states, $S1$ and $S2$ have a common fanout state, the weight of the edge $\langle S1, S2\rangle$ of the attraction graph is increased.

Once the graph of the attractions is found, we try to assign adjacent codes to pairs of states that have strong (i.e., higher weights) attractions. It is true that always it may not be possible to assign adjacent codes to all the state pairs. In that case, we start with the state pairs with the edge that has the highest weight and assign adjacent codes.

Following that we move to the edge with the next highest weight and assign adjacent codes if possible. We repeat, till all the states are assigned a code.

(Refer Slide Time: 57:10)



State Encoding

The equations can be simplified (by low level minimization) as given below.

$$s'_0 : f_{s_0}(x, s_0, s_1) = \dots\dots + \boxed{x\bar{s}_1} + \dots\dots$$

$$s'_1 : f_{s_1}(x, s_0, s_1) = \dots\dots + \boxed{x\bar{s}_1} + \dots\dots$$

$$O : f_O(x, s_0, s_1) = \dots\dots + \boxed{x\bar{s}_1} + \dots\dots$$

It may be noted that the reduction is possible only because of assigning adjacent codes to *S1* and *S2*.

So, that is actually state minimization so how MUSTANG shown. Now you can see that now these are little bit problem if I directly use this philosophy. The philosophy is there always say for example I always one to try to have if some states are there. So, this is common fanout and also you can say that this is another state we have common fanout I should try to help this is 0, 0 this is 0 1. Now this one bit difference and say this is I want put it as say I have to put it as 1, 0, 0. Then we can say that if have to put it as 1 0.

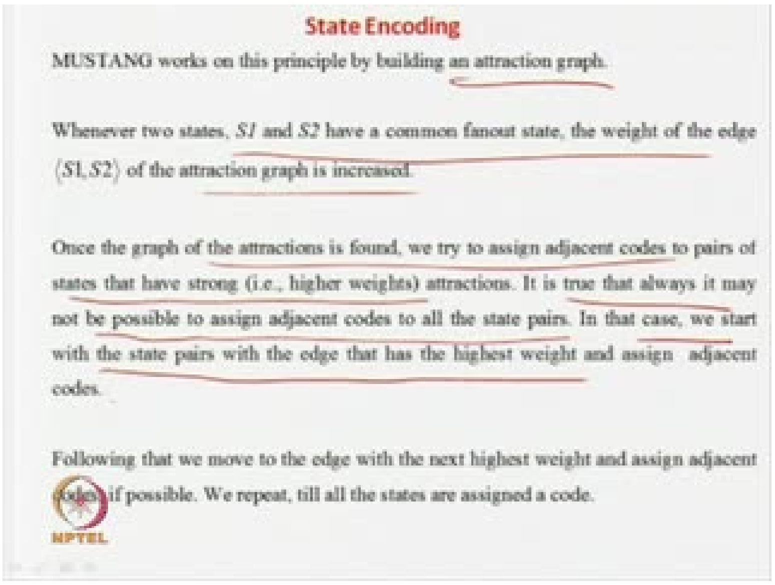
So, in this case we can see and again this is 0, 0, one bit difference one bit difference is there. Now we can see that now for example, I have also state transition from this to what happens? We can see that so in this is 0, 0, 1, 1 difference there 2 states we can see that this is 0, 0, 0. So, I want this difference now there is path from this to this state then there are two bit differences.

So, always it may not be possible to have this adjacent with length that we always way not possible for you to give what we are say that is one bit difference in all cases because then may it lead to a, what you called that encoded. So, in this case you may have given only 1 in for this state that is actually called hallow 1 encoding. We can say that 1, 2, 1, 2, 3, 4 states are there so i can say that there are big difference that is one.

So, in this case i may have put it as 0, 0, 1, and 0. So, this one be 0, 1, 0, 0 and this should be 1,0, 0. so, only one bit is high difference to this is State Encoding will take a huge number of to do that look at that will possible that 1 is only one bit is 1 in the encoding. We do that more so, we do not ate such use space for doing this State Encoding if you are using two bit encoding to do this. So here it is not possible this is 0, 0, 0, 1 you can put it as 1 0 as I told you 1 0 but, again say 0, 0. You can put sorry 1, 1 you can put it over here but, again now these are transition from here to a two bit difference so you may not be always possible to do this so ah.

We they actually have a applied an algorithm to do that; so in the MUSTANG now will actually will tell you that how to start you not all that this example shows that not all the states you can follow which is having adjacent ports. Then which one you will give preference weather you apply a 0, 0 here and you apply a 0 1 here or actually that is you are giving preference to this link or you are going to giving preference to this links or which links you are going to give as preference.

(Refer Slide Time: 59:41)



State Encoding

MUSTANG works on this principle by building an attraction graph.

Whenever two states, $S1$ and $S2$ have a common fanout state, the weight of the edge $\langle S1, S2 \rangle$ of the attraction graph is increased.

Once the graph of the attractions is found, we try to assign adjacent codes to pairs of states that have strong (i.e., higher weights) attractions. It is true that always it may not be possible to assign adjacent codes to all the state pairs. In that case, we start with the state pairs with the edge that has the highest weight and assign adjacent codes.

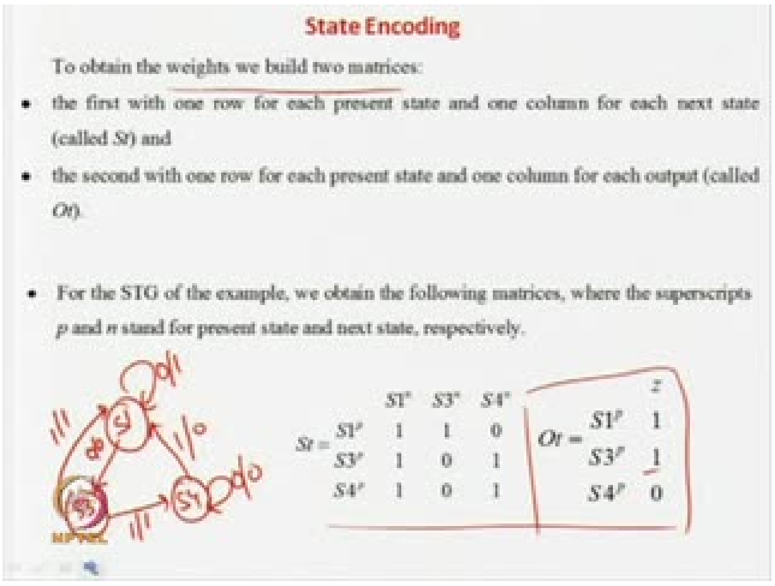
Following that we move to the edge with the next highest weight and assign adjacent codes if possible. We repeat, till all the states are assigned a code.

So, MUSTANG actually gives you an actually idea finding out an attraction graph and it will give some weights to the edges and this and depending on the edges you can give priority that means which is of the two nodes will have the common adjacent encoding. So, this either switches having the higher weights will be given and the nodes corresponding to the edges having will be giving first priority present adjacent codes. Adjacent having higher value for given for first priority. We keep on moving and now off course some of this step if they have what you call common fanout may not be ever to give then address and encoding that is the idea.

So, we actually a whenever states S 1 and S 2 common phenomenon these are the weight of the age is increased; that means, how it works the attraction graph. So, if two nodes S 1 and S 2 are common fanout state the weight of the age is increased. So, if 2 states say S 1 and S 2 have lot of common fanout. Then what do you do? So, if an age has if an age has a lot of common fanouts, if two nodes say S 1 and S 2 lot of common fanouts then the weights are increased starting and if that nodes S 3 and S 4 then only one or two nodes common fanouts then what do you do then it will get the weight. So, that means two nodes S 1 and S 2 which have lot of common fanouts nodes you will be given higher weights and you will try to give them adjacent codes compare to nodes which have a less number of fanouts, similar same fanouts.

Once your attraction graph is found we try to assign the adjacent code to pair of states having strong attraction; that means, will select those pair of nodes I mean, say S 1 and S 2 this is have a large number of common fanouts. Then correspond to the in the to the nodes it is do not have that many number of common fanout nodes. It is true that it is always true that it may not be able to possible to assign the adjacent code to all these states pairs. In that case we start with the state pairs that are the highly weight; in other words state pairs having a most numbers of common fanouts and assign adjacent code. We keep on doing a I mean a course some of the state pair lots have this one.

(Refer Slide Time: 61:40)



State Encoding

To obtain the weights we build two matrices:

- the first with one row for each present state and one column for each next state (called *St*) and
- the second with one row for each present state and one column for each output (called *Ot*).

- For the STG of the example, we obtain the following matrices, where the superscripts *p* and *n* stand for present state and next state, respectively.

$$St = \begin{array}{c|ccc} & S1^n & S3^n & S4^n \\ \hline S1^p & 1 & 1 & 0 \\ S3^p & 1 & 0 & 1 \\ S4^p & 1 & 0 & 1 \end{array} \qquad Ot = \begin{array}{c|c} & z \\ \hline S1^p & 1 \\ S3^p & 1 \\ S4^p & 0 \end{array}$$

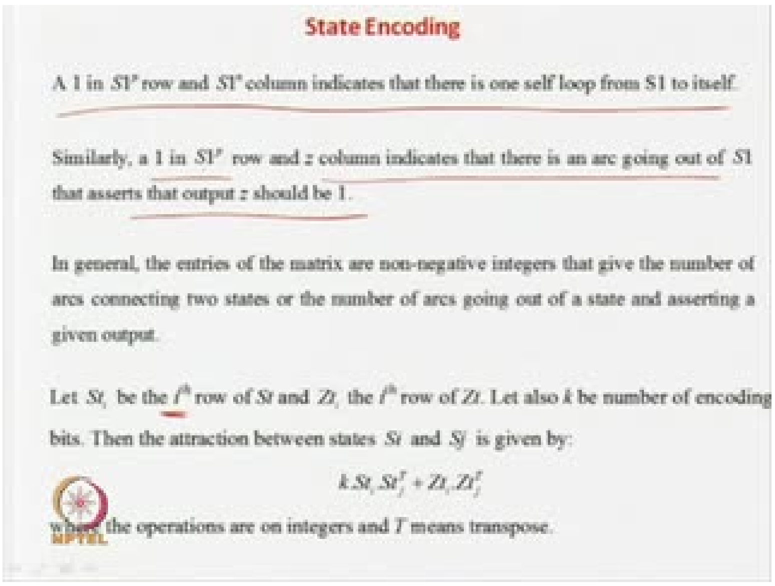
So we will just explain the idea some algorithm what we do find out all state pairs which are having some common fanouts. So, this number of state pairs more number of common fanouts will be having larger weights corresponding to the states what is having a I mean understand corresponding with this states which are may having less number of common fanouts. Now we give weights and we start with the nodes having the highest weight. You assign the some weights we the node pairs with highest number of weights assign some adjacent codes and you keep on moving now we have the algorithm.

So, that is simple once weight has given so what do you do? Once the weights have been final weight given you just select the node pair with the highest number of weights assign an adjacent codes, then we take the next the node pair having next highest weight. We keep on moving but, now we see how to obtain these weights. So, this is the, we represents the matrix by 2 graphs what you called 2 matrices. So, this corresponds to 3 nodes is the orbiter graph. So, S 1 this S 3 and this is S 4. So, there is a p representing the presence state and this one is the next state. So from S 1 present to S 1 is then S is the loop; that is been shown over here. And from S 1 to S 4; so, from S 1 to s 4, you can see from S 1 to S 4 you can see. So, S 1 to S 3 from S 1 to S 3 you can see there is an age.

So, is matrices is 1 and 1 h. So, S 1 to S 4 there's no 0. So, there S no S similarly, from S 3 through S 1 there is 1 from S 3 to S 1. You will have h then from S 3 to S 1 there S a node 0. So, node loop nothing no transition will be there from S 2 to S 4 there is transition as is 1 from S 4 to S 1 there is a transition. Because, of S nodes from S 1 to S 4 to S 4 itself there is a transition, you put a self assign. This is matrix representing S 3, S 3 matrix and this is another matrix that is output matrix. So, you says that S 3 1. So, S 1, 1 z equal to 1. So that means what that means there is an outgoing h from 1 is output is 1.

So, you can say that this is 1 this is 0 0 that. You can say that means what you can say that there at least 1 age which coming out of present state of S 1 the output is 1. Similarly, S 3 you can say that this is 1 1 and this is also 1, 1 that is from S there is an S in this case both ages are outgoing ages of S 3 output. That is 1 from S 4 both this are 0 that is what being seeing. You seeing that p 4 z 0 that is only ages of from S 4 the output is 0. So, this for this graph you actually draw this in simple matrix way.

(Refer Slide Time: 64:22)



State Encoding

A 1 in $S1^p$ row and $S1^n$ column indicates that there is one self loop from S1 to itself.

Similarly, a 1 in $S1^p$ row and z column indicates that there is an arc going out of $S1$ that asserts that output z should be 1.

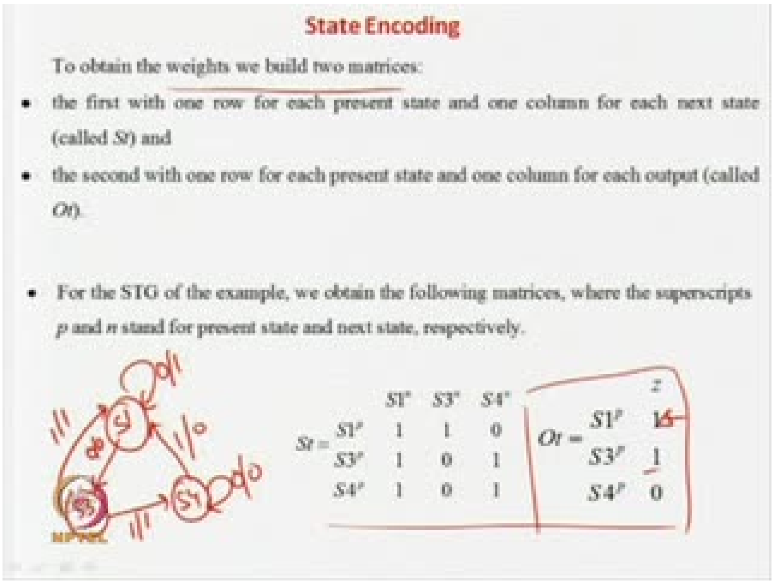
In general, the entries of the matrix are non-negative integers that give the number of arcs connecting two states or the number of arcs going out of a state and asserting a given output.

Let St_i be the i^{th} row of St and Zt_i the i^{th} row of Zt. Let also k be number of encoding bits. Then the attraction between states Si and Sj is given by:

$$k\, St_i\, St_j^T + Zt_i\, Zt_j^T$$

where the operations are on integers and T means transpose.

(Refer Slide Time: 64:50)



State Encoding

To obtain the weights we build two matrices:

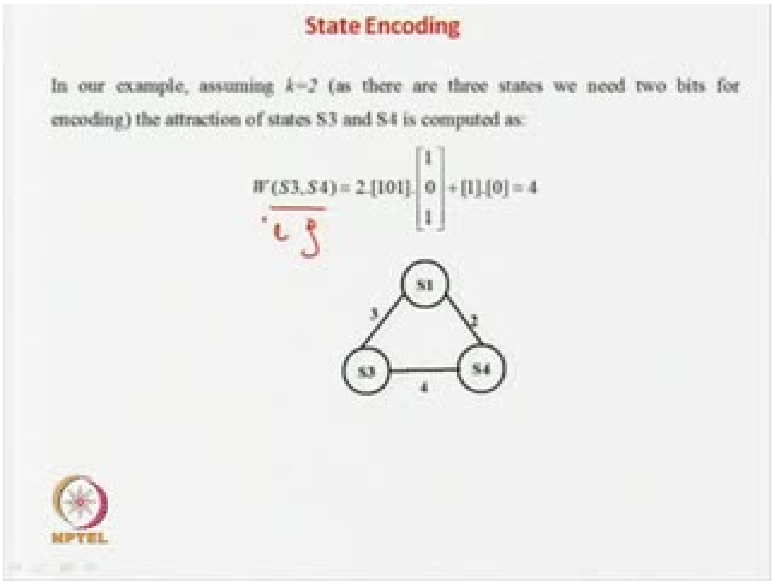
- the first with one row for each present state and one column for each next state (called St) and
- the second with one row for each present state and one column for each output (called Ot).

- For the STG of the example, we obtain the following matrices, where the superscripts p and n stand for present state and next state, respectively.

$$St = \begin{array}{c|ccc} & S1^n & S3^n & S4^n \\ \hline S1^p & 1 & 1 & 0 \\ S3^p & 1 & 0 & 1 \\ S4^p & 1 & 0 & 1 \end{array} \qquad Ot = \begin{array}{c|c} & z \\ \hline S1^p & 1 \\ S3^p & 1 \\ S4^p & 0 \end{array}$$

Now if you want to find out that what is the attraction between say two states let S t be the ith row ith row and of. So, this general thing this you have said 1 in row in this one indicate there is a self loop algorithm, you have seen. Similarly, 1 in s p 1 indicates that is outgoing h from 1 is output is 1 that is what you are seeing. Similarly, 1 s p 1 row and z column that is this one that is saying that means these outgoing age from S 1 is output and S 1 that is been said. So, whatever I have been discussed about the drawing matrix for this graph is the represents is being told in this slide.

(Refer Slide Time: 65:03)



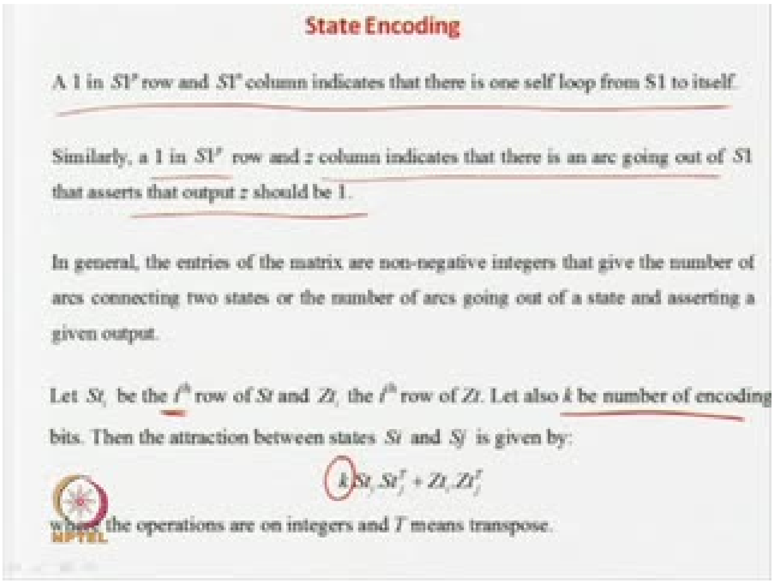
State Encoding

In our example, assuming $k=2$ (as there are three states we need two bits for encoding) the attraction of states S3 and S4 is computed as:

$$W\underline{(S3,S4)} = 2.[101].\begin{bmatrix}1\\0\\1\end{bmatrix} + [1].[0] = 4$$

$i \quad j$

(Refer Slide Time: 65:11)



State Encoding

A 1 in $S1^s$ row and $S1^s$ column indicates that there is one self loop from S1 to itself.

Similarly, a 1 in $S1^p$ row and z column indicates that there is an arc going out of $S1$ that asserts that output z should be 1.

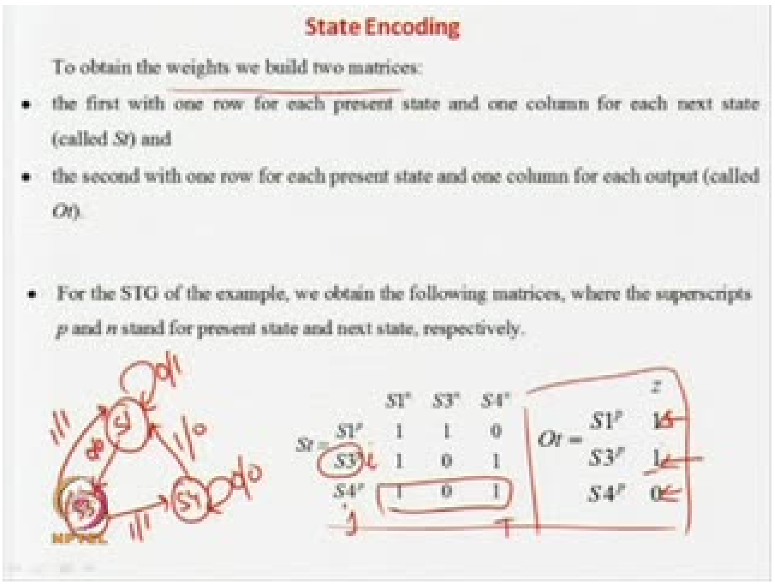
In general, the entries of the matrix are non-negative integers that give the number of arcs connecting two states or the number of arcs going out of a state and asserting a given output.

Let St_i be the i^{th} row of St and Zt_i the i^{th} row of Zt. Let also k be number of encoding bits. Then the attraction between states Si and Sj is given by:

$$k.St_i.St_j^T + Zt_i.Zt_j^T$$

where the operations are on integers and T means transpose.

(Refer Slide Time: 65:18)



State Encoding

To obtain the weights we build two matrices:

- the first with one row for each present state and one column for each next state (called St) and
- the second with one row for each present state and one column for each output (called Ot).

- For the STG of the example, we obtain the following matrices, where the superscripts p and n stand for present state and next state, respectively.

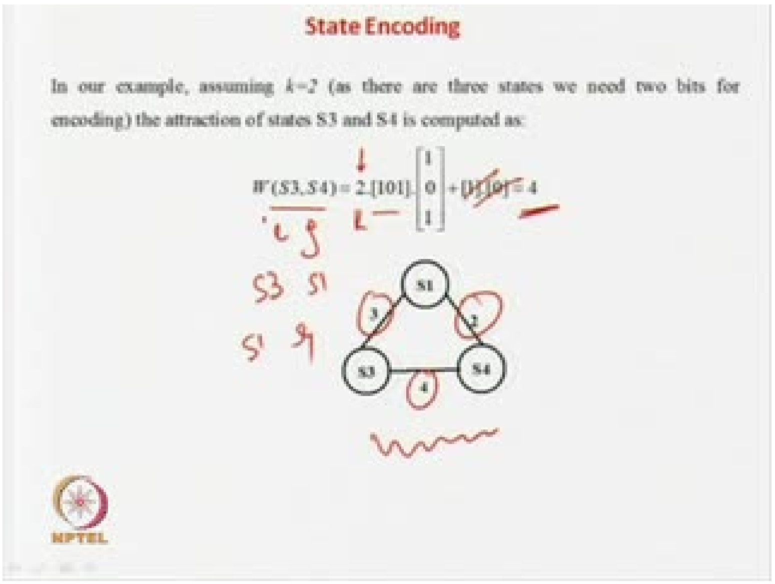
$$St = \begin{array}{c|ccc} & S1^n & S3^n & S4^n \\ S1^p & 1 & 1 & 0 \\ S3^p & 1 & 0 & 1 \\ S4^p & 1 & 0 & 1 \end{array} \qquad Ot = \begin{array}{c|c} & z \\ S1^p & 1 \\ S3^p & 1 \\ S4^p & 0 \end{array}$$

So, now you have to find out the attraction. So, you have to find out the attraction between say S 3 and S 4. That is i, we call this is call as i and this is j. So once you do that what the formula this is k. So, what is the k, k is the number of bits required for encoding. And then, S t i this is your matrix S t i so this is ith row you are taking say so for example you have to finding out the attraction between S 3 and S 4. So this is i; so s p 3 this row you have to take and S t j transports so 2 are considering say between 3 and 4. So this 4 will become your j and you have to take transports of this one and you have to do this multiplication plus i ith row and jth column transpose for this matrix.

So, if you are to find out i and jth of course if you are find out the attraction between node number 3 and 4 how do you do that. So, you have to take the ith row s this is the i t h row number of bits 3. So number of bits required encoding is to that is equal to k over here that s told over by the equation. So this equation is saying how can you find out the attraction between 2 nodes. So k is to in this case i is your finding out the between 3 node what is the row is 1 0 1 that is ith row you have to write 1 0 1 over here now that is fourth matrix that is the j is 4. So you have to finding attraction between 3 and 4.

So this is j; so in this case you can see j transpose. So what is the j over here so j is 1 over here but, you have to take this transpose. So this is actually, this is transpose, plus this or this. So what is this one? So i is and 4. So, this is 3 that is row z 3 and z 4. So, this is z 3 and this z 4. So, this one star 0; so this is nothing but 1 star 0. So if you do this is gone you will final the product the value is 4, so you can write down a 4 over here.

(Refer Slide Time: 66:57)

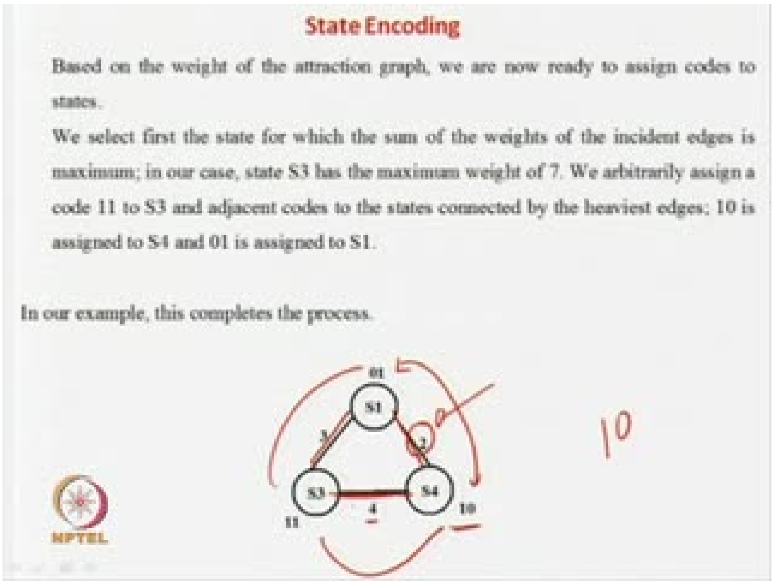


State Encoding

In our example, assuming $k=2$ (as there are three states we need two bits for encoding) the attraction of states S3 and S4 is computed as:

$$W(S3,S4) = 2.[101].\begin{bmatrix}1\\0\\1\end{bmatrix} + \cancel{[1][0]} = 4$$

(Refer Slide Time: 67:39)



State Encoding

Based on the weight of the attraction graph, we are now ready to assign codes to states.

We select first the state for which the sum of the weights of the incident edges is maximum; in our case, state S3 has the maximum weight of 7. We arbitrarily assign a code 11 to S3 and adjacent codes to the states connected by the heaviest edges; 10 is assigned to S4 and 01 is assigned to S1.

In our example, this completes the process.

Similarly, you have to find out the attraction between S 3 and S 1 so in this case S 3 and S 1 if you find out so this one will be S 3 and this one will be s 1. Similarly, matrix you can find out will be that will be 3 from S 1 and S 4 you want to find out then it will be S 1 and S 4 you to the computation it will be 2 and so fore. Now you have got the bits;

now you can find out that between S 3 and S 4 the weight is highest is 4. So, you take S 3 and S 4 and the give some encoding so this is the highest row.

So, you can see that so in this case you have apply 1 1 and 1 0. So, this is in this case is adjacent and 1 so it is very simple. Now the once the rows has been given so you can find out that which is the strong in a see there for you corresponding to first the bits already been defined now, when the one when the weights has been defined.

Now you can see for S 3 the weights is 3 plus 4 is 7. So, in the so this 2 nodes you can say sorry let us just find out what does that mean, so now you can you have to see taht this 2 pear this attraction between 2 this is 4 this is 3 and this 2 pear is 4 so 2. So, this 2 are highly attracted pear that means, the number of fanout requirement for this S 3 and S 4 is the highest and similar you can keep you think of keeping the for you start with 4 then you go for 3 and 2 and then you keep on think the adjacent encoding. So, in this case you take this weight and you put 1 1 and 1 0 so 1 bit adjacent is there now you go to S 3 so here also you have 1 0 is that.

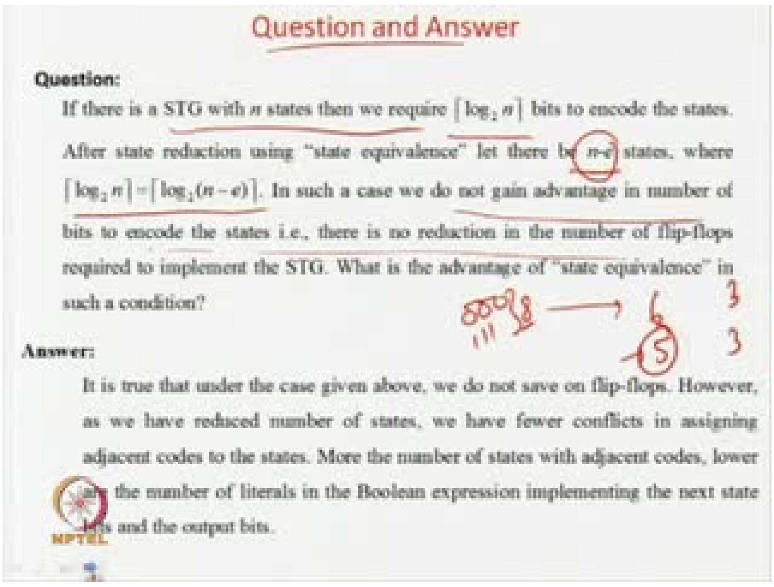
You put 0 1 is there so that adjacency is 0 1, but you can see that there this is not adjacent pair because is 1 0 and 1 not 2 be difference are there, but again it is the lowest weight so this is not get the priority. So, you have started with this one put some adjacent and weights then you go for this 1.

Second row you put some adjacent encode, but again when you are coming to last bit so you cannot do this because you can exhorted of the number of encoding bits. So, these are the complete the process. So, you have weight cells 1 1 1 0 and 0 1. So, if you look at our first example; when you are counting number of literals 11 and 10 so if you find out that if you are using this encoding the number of literals was 10 and if you using some other encoding in which you are as state encoding of 0 0 something like this. The number of literals was 11and this was not 2. So, you can find out that must and actually tells you the attraction between the nodes and then you can go about this.

So, once you have find out the first and final equivalent states by the theory I have told you, you minimize that when you use the MUSTANG algorithm is an heuristic. So, you find weight for which of the state pears so you take the highest weight put adjacent code and keep on doing it.

Finally, you may find out that you are going for 4 3 2 1 from highest weight to lowest weight you are going on than in the end you may find out that 1 state pear you may not be able to assign adjacent encode so fore. You have to keeps on doing this one I am then finally you follow this states will have adjacent encodes and some of them will not have. So, once the state adjacent matrix adjacent matrix some of the states with adjacent code some of the applied and some of the states encode could not be applied. So, finally you have a state encoding so once state encoding is that you go for this table and implement it using a 2 level binary minimization.

(Refer Slide Time: 69:56)



Question and Answer

Question:

If there is a STG with n states then we require $\lceil \log_2 n \rceil$ bits to encode the states. After state reduction using "state equivalence" let there be $n-e$ states, where $\lceil \log_2 n \rceil = \lceil \log_2 (n-e) \rceil$. In such a case we do not gain advantage in number of bits to encode the states i.e., there is no reduction in the number of flip-flops required to implement the STG. What is the advantage of "state equivalence" in such a condition?

$8 \longrightarrow 6$ 3

5 3

Answer:

It is true that under the case given above, we do not save on flip-flops. However, as we have reduced number of states, we have fewer conflicts in assigning adjacent codes to the states. More the number of states with adjacent codes, lower are the number of literals in the Boolean expression implementing the next state bits and the output bits.

So, this is how you actually come to a Finite State Machine Synthesis. This process is very simple, if you write in an actual so what is happening first you are finding out the equivalent states. So, how do are finding out the equivalent state two states, S 1 and S 2 are equivalent if there does not exist any length string. Which can distinguish is them? And what is the length of the string maximum the number of states in the graph that is first state. So, you do that and merge all the equivalent states. Next what you do next, you go for MUSTANG or some other heuristic algorithm to find out good state encoding. So, you try to find out states with large number of common fanout and apply them as adjacent codes some of the states you may not be case states may not able to do this because of the fact I told you.

Then give state encoding so once the good state encoding have been given you go for this state level representation binary representation binary minimization what you are already done. So, this is how you go for minimization of Finite State Machine and what actually a pre processing step that is state encoding machine state minimization states encoding converting this circuits to a combinational circuit kind of a thing. So, that the combinational synthesis algorithms can be easily applied.

Now you are going to the question and answers session so what is the question, the question is saying that if there is an s t g with n states. So, you know that lock 2 n upper say is required to the encode this things. Then you go to the state reduction and by equivalence of the number of states minimize is n minus e and then with reduction is say very very less in some case so is remain something like that.

But, maybe 1 or 2 states might has been reduced, but if you tell still take law then what happens actually this is still remains this seen. That is by reducing the number of states you do not you cannot reduce the number of bits require to encode like for example; initially you have 6 states so you require 3 bits to encode, but now first reducing this state by equivalent now if you have 5 states still you required this 3 bits to encode. Then you do not gain any thing by merging equivalent states, but still why do you do that? You do not gain anything means do not gain anything in the number of flip flops so the number of bits in state encoding, but still why you merge equivalent states? The answer is if you are merging equivalent states from say 6 states to 5 states still you required 3 bits to encode.

But, now initially you had 0 0 0 2 1 1 1. So, 8 encoding bits possible and you have to using this 8 encoding. You have to encode 6 states, but now merging equivalent states now you have 5 states. So, once set you have see number of bits remains are same that is not is weight state, but using this 8 encoding possible now you are using you can have 5 states. So, there is more flexible in the MUSTANG will succeed reduce that MUSTANG or any heuristic state encoding is succeed to give adjacent state adjacent state encoding if the number of states are less corresponding to a case compared to state compared to the case for number of states are more.

So, you always go for state encoding even if the number of state sorry you always go for state minimization even is the does not lead to any saving or do not lead to any kind of what you can what do I say even if they do not lead to any kind of gains in terms of bits but still you have flexibility in the encoding because now you have after state minimization now you have more state bids to encode state. So, any good heuristics to put adjacent encode will succeed so that is what is the reason for this one. So, with this come to the end of this lecture and the next lecture will see multi level implementation because, in all the lecture still now you have seen that 2 level implementation is good and you have seen lot of algorithm for that, but in practical scenario 2 level implementation are not possible because of the implementation are not possible because of the requirement of high fanout in the gates. So, you have to open multilevel implementation that will see the next lecture.

Thank you.