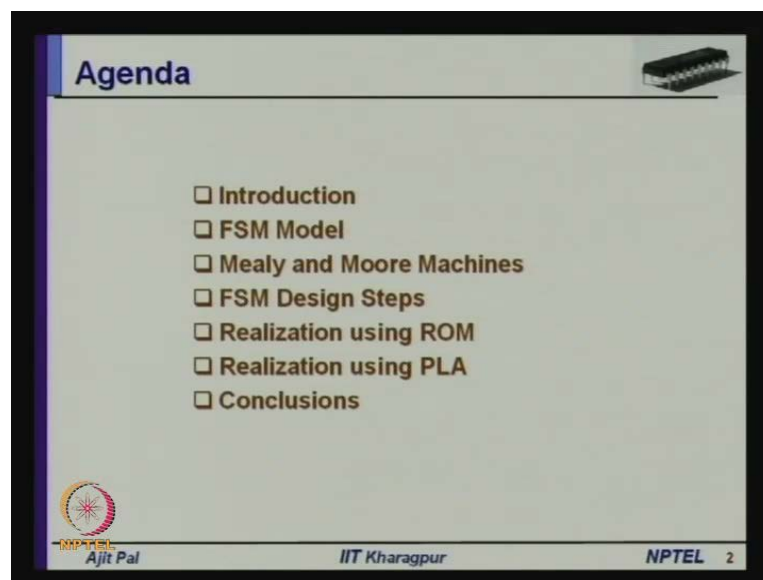


Low Power VLSI Circuits and Systems
Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Module No. # 01
Lecture No. # 17
Finite State Machines

(Refer Slide Time: 00:23)



Hello and welcome to today's lecture on finite state machines. Here is the outline of today's talk; I shall give a brief introduction about what is finite state machine, why it is required. Then, discuss a finite state machine model, particularly this is important in the context of implementation, there I shall talk about two different types of finite state machines Mealy and Moore machines. Then, I shall discuss finite state machines design steps, through a number of design steps, the specification is transformed into implementation in terms of gates and flip-flops, and how that has been done that I shall discuss in detail. Then I shall discuss realization of finite state machines using read only memory, as you know read only memory can be used to realize communication circuit. Similarly, we shall use programmable logic array for the realization of finite state

machines, I shall discuss about them. Then, I shall conclude my lecture with some concluding talks concluding reworks.

(Refer Slide Time: 01:33)

Introduction

- > Any digital system will have some control part
- > Finite state machines (FSMs) are commonly used to realize control parts
- > FSM provides an organized structure for capturing control sequences and operation
- > An FSM can be diagrammatically represented by a state-transition diagram
- > Labeled nodes represent states and labeled directed edges represent transitions among the states
- > Example: Consider a sequence detector that produces '1' when five one's appear sequentially at the input

The diagram shows a state transition diagram with five states: S1, S2, S3, S4, and S5. S1 is the start state. Transitions are as follows: S1 to S1 (0/0), S1 to S2 (1/0), S2 to S2 (1/0), S2 to S1 (0/0), S2 to S3 (1/0), S3 to S3 (1/0), S3 to S2 (0/0), S3 to S4 (1/0), S4 to S4 (1/0), S4 to S3 (0/0), S4 to S5 (1/0), S5 to S5 (1/0), S5 to S1 (0/0, 1/1), S5 to S2 (0/0), S5 to S3 (0/0), and S5 to S4 (0/0).

NPTEL
Ajit Pal
IIT Kharagpur
NPTEL 3

Any digital system will have some control part, why digital system, every system will have two parts hard and soft; like hardware and software, body and soul. Similarly, any system, any digital system will have two paths, one is known as data path and second is known as control path. For example, say data path and control path, for example, let me take off the example of general purpose computer, say CPU central processing unit. As you know, it consists of arithmetical logic unit, a set of registers, and these ALU and registers are controlled with the help of a control unit. Then, the whole thing is known as central processing unit, and of course you will require some interface for connecting to the outside world, and it communicates with the outside world through a bus known as system bus. As you know it comprises of at this lines data lines and control lines, and this is that interface. Now, this control unit controls ALU and registers, which is inside the CPU, and not only has that it controlled what is connected to the CPU through this interface, through the system bus.

So, you can see here the control unit is performing the role of controller of all these components which are present inside the CPU, and also which is connected to the CPU through external bus, and essentially control path is realized by the control unit and data path constitutes ALU registers and other things. So, you can see you require a control

part, and finite state machines are commonly used to realize control parts. So, how do you realize the control part, it can be realized with the help of the finite state machine. For example, this control unit of the CPU can be realized with the help of the finite state machine, and essentially FSM provides the finite state machines, provides an organized structure for capturing control sequences and operation. As you know a controller controls certain things through a sequence of events, through a number of steps and it will require several functional components. So, this is being nicely captured with the help of a finite state machine.

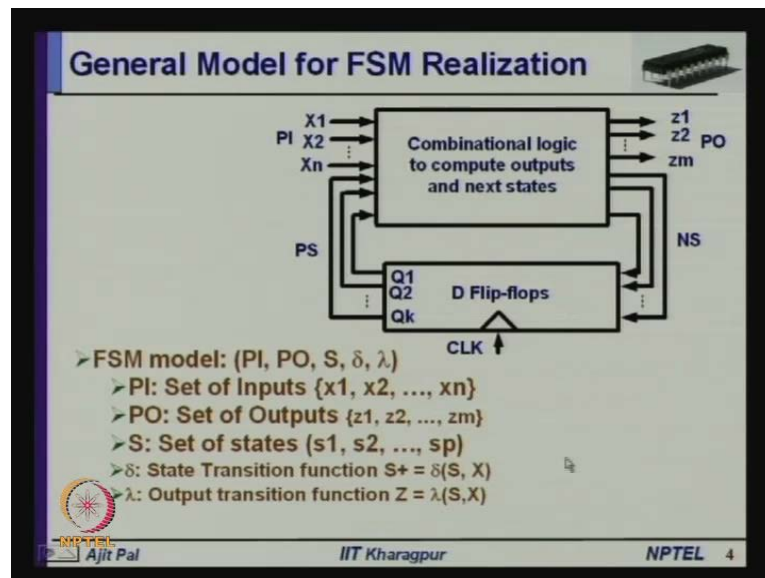
And a finite state machine can be diagrammatically represented by a state-transition diagram, question naturally arises how do you really represent a finite state machine. There are various ways of representing finite state machine, for example the specification usually given in terms of simple English language or may be with the help of some programming language C, or may be with the help of some hardware description language; like very long and so on. Whatever it may be that can be transformed into a more elegant representation with the help of what is known as state-transition diagram. And it nicely captures the functionality as well as the control sequences. And in a state-transition diagram we have a labeled nodes, I mean there are label nodes that represents states a finite state machine as the name specifies it will have finite numbers of states, and those are represented with the help of labeled nodes; usually circle, and label directed edges represent transitions among this states; that means, the finite state machine will go from 1 state to the another state, that transition is represented with help of edges, and on the edges you will specify on what condition the transition is taking place and what is the outcome of that particular transition.

Let us consider a simple example to crystallize the idea, consider a sequence detector that produces one, when five one's appear sequentially at the input; that means, you are designing a sequence detector; that means, a series of input are coming, series of one's and zero, that whenever there are five consecutive one's you want to produce one, that is the function of the controller. And let us see how the state transition diagram captures the functionality and the transaction sequences, as you can see the finite state machine of this sequence detector, that produces one when five one's appear sequentially at the input, can be represented with the help of five states s_1 s_2 s_3 s_4 and s_5 , and s_1 we may call it is as initial state. So, if a zero comes it will remain in that state; that means, here

actually that 0 slash 0 represents the first 1 is the input, and the second 1 is the output. So, input and output for that transition is specified by the side of this edge. So, whenever the machine is in state s 1 if a 0 comes, it will remain in s 1 and zero will be the output. Then if a 1 comes it will go from s 1 to s 2, but still it will produce zero.

See again if, whenever it is in s 2 if 0 comes again it will come back to s 1, by outputting zero, but if a 1 comes it will go to state s 3 by producing 0, and then when it is in s 3 it will again produce output zero both for input zero and input one; however, when the input is zero it will go back to state S 1 and when the input is 1 it will go to state s 4, and if another 1 comes from s 4 to s 5 still producing 0, because not yet five one's has occurred in the sequence, and if zero comes again it goes back to state s 1. Finally, whenever it is in s 5 if a 1 comes, we have already completed 5 one's 1 2 3 4 and 5. So, five one's has come again. So, it will go back to state s 1 and producing 1, or it will go back to s 1 if the input is 0 producing zero; that means, in that case you have got only 4 one's at the input not five one's. So, this is how the function of the sequence detector, which detects a 1 whenever there are five consecutive one's can be diagrammatically represented with the help of a state-transition diagram.

(Refer Slide Time: 09:32)

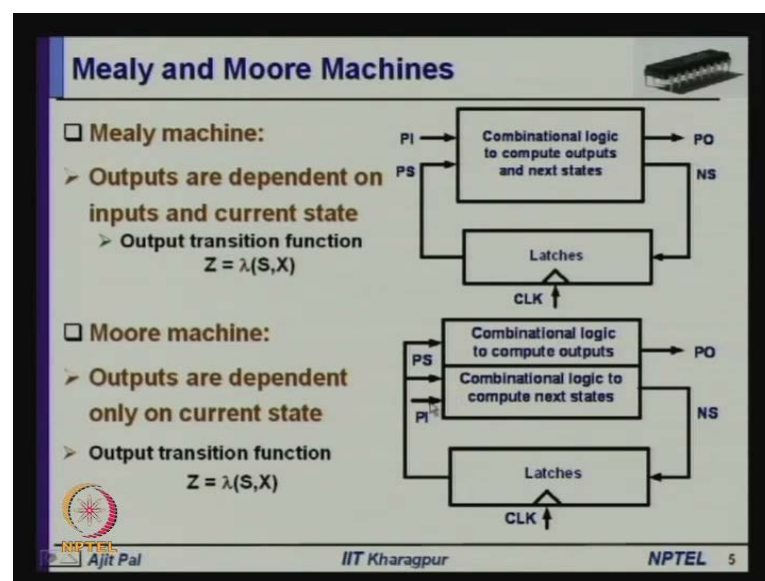


So, this is shown with the help of this example. Now, comes the question of realization, this realization is related, I mean can be represented with the help of a generalized model. You can see this is the generalized model for finite state machine realization. It

can be represented by 5 couple P I is a set of inputs, set of inputs, it will have set of primary inputs $x_1 x_2$ up to x_n , n primary inputs, it will have a set of outputs $z_1 z_2$ up to z_m . So, m primary outputs $z_1 z_2 z_m$ which is produced with the help of this combinational logic circuit. And in addition to that, it will have a set of states which are represented by a set of flip-flops, here I have written d flip-flops, but it is not really necessary to use d flip-flops, you can use any type of flip-flops, s_r flip-flop, j_k flip-flop, only thing that excitation function for these flip-flops should be different, but d flip-flops is most commonly used and the simplest, and so the state are represented with the help of this flip-flops and.

So, these states are there are p states mention here $s_1 s_2$ up to s_p . So, if these are p states the number of flip-flops required will be root p , sorry 2 to the power k ; that means, you have got k flip-flops will be equal to p . And delta the state transition function s_{plus} is represented by $\delta S X$; that means, the next state is dependent on the present input x_1 to x_n this input combination, and the past history which is represented by the state of deviation; that means, it is a function of state and the present input. Similarly, output transition function whatever output it will produce, here Z capital Z is essentially combination of z_1 to z_n , that is a function of again states as you can see the present state and the primary inputs. So, this is the generalized model.

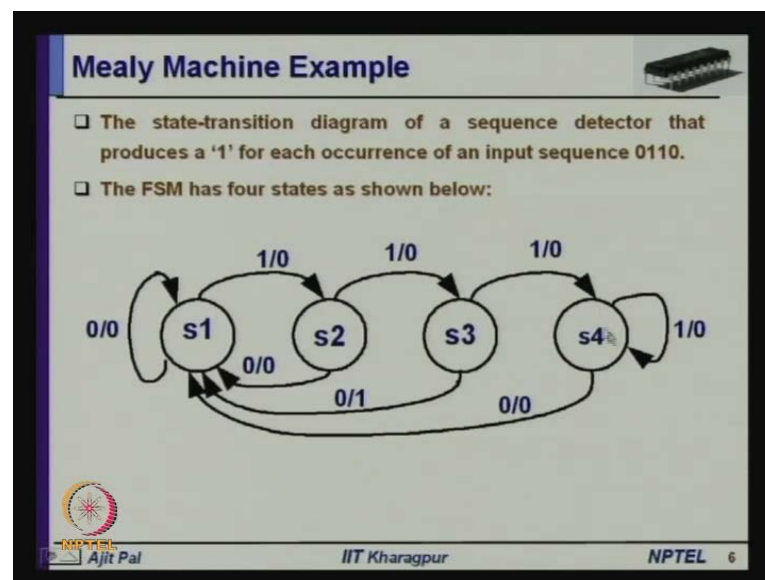
(Refer Slide Time: 12:02)



However, there are 2 classes of machines which are known as Mealy and Moore

machines. So, Mealy machine, in case of Mealy machine outputs are dependent on inputs and current state; that means, Z is equal to $\lambda S X$, as it is represented with the help of this diagram. Here as you can see the primary output is a function of both the primary input as well as the present state S . So, it is the function of both, and which is that is why it is known as Mealy machine. On the other hand the Moore machine, where the output is dependent, outputs are dependent only on current state, **not on the not on the not on the x** here there is a mistake, so actually for Moore machine, if it is a Moore machine, then your Z will be equal to λ into s not x , it will not depend on a input, it will depends only on this state; that means, the in case of a Moore machine output is dependent only on the present state not on the input. So, if there are n states, we have seen there are p states. So, p states mean p different outputs are possible in case of a Moore machine. We shall illustrate with example as you can see here, the output is dependent only on the present state p s not on the primary input. However, the next state function ah for both deviations is dependent on the primary input as well as on the present state, both here and as well as here, both for Mealy and Moore machine.

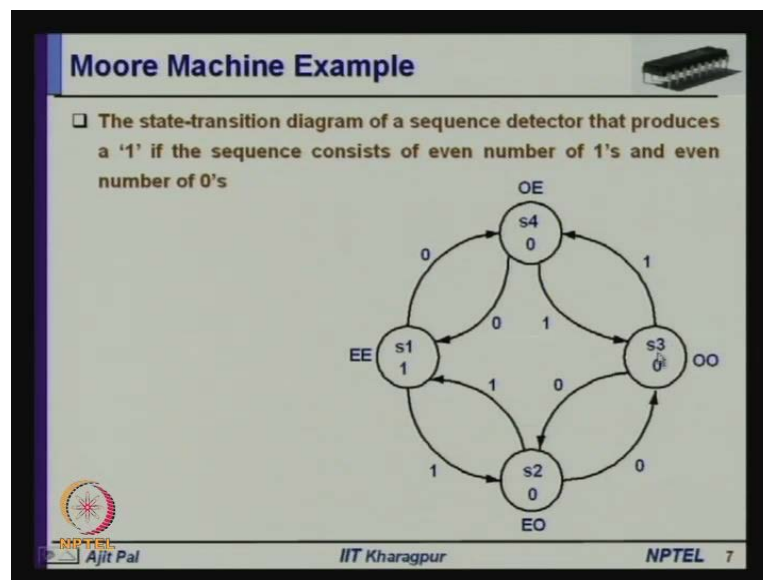
(Refer Slide Time: 13:56)



Now, let me a example to crystallize the idea of Mealy machine, here the state-transition diagram of a sequence detector that produces 1 for each occurrence of an input sequence 0 1 1 0 is shown here, in this particular case. Again it is a sequence detector, because sequence detector forms the simplest examples of controllers. Then here as you can see the finite state machine has 4 states s_1 s_2 s_3 and s_4 , and s_1 is the initial state again as

you can see, whenever it is in state s 1 if a 0 comes it remains in 0, and the state s 1 then if a 1 comes it goes to our state s 2 by producing 0; however, if a 0 comes it comes back to state s 1, again if 0 another 1 comes it goes to state s 3 by producing 0, whenever it is in s 3 it come comes back to state s 1 by producing 1, because here you can see it has fulfilled the requirement of producing 1, 0 when it was here 0 1 1 0, that is our specification, it should produce 1 when the input sequence is 0 1 1 0. However, instead of this, if it produces say 3 one's successively. So, in that case it goes from s 3 to s 4 by producing 0, and if it continuous to receive one's after that it remains in it is state 4 as long as one's are inputted; however, as soon as a 0 comes, it goes back to state s one. So, this is the specification of a finite state machine, that realizes the sequence detector for the input sequence 0 1 1 0. So, this is the machine.

(Refer Slide Time: 15:55)



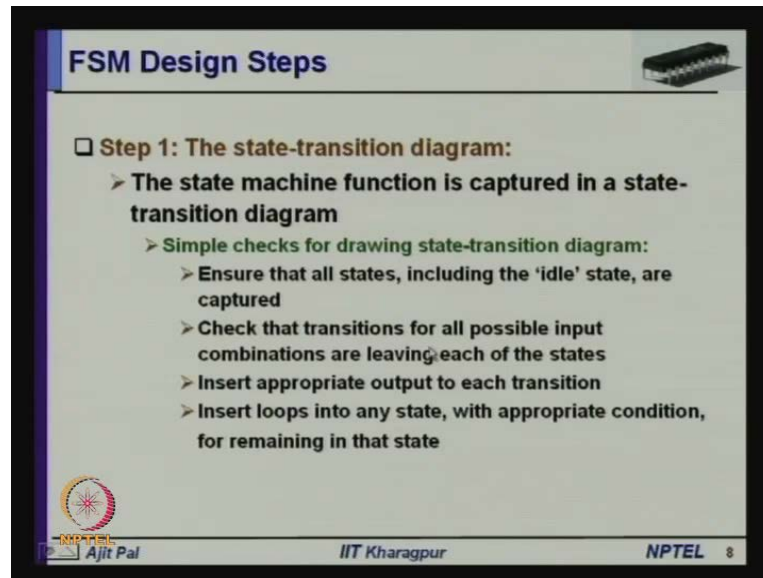
Now, let us consider a Moore machine, the state-transition diagram of a sequence detector again I have taken example of sequence detector, that produces a 1, if the sequence consist of even number of one's or even number of 0; that means, suppose you have inputted a sequence of zeros and ones, and you have finish the input sequence, and at the end if the number of ones and number of zero's both are even, then it produces a 1. On the other hand, if the input sequence has got odd number of 0 or odd number of 1 or odd number of zero as an one's, then it produces 0, that is the functionality to realize, and this can be represented with the help of the state-transition diagram. It has got again 4 states starting with states s 1, where both are 0 that is no input, I mean assuming that

both the one's and zero's are zero.

So, in this case even means 0 0 or it can be initially it is definitely 0 0. So, it is in state one. And then from here it will go to whenever a 0 comes, it will go to o e or 0 even one and if a 0 comes again it comes back to s 1, because it has now got even number of zeros. So, it produces a 1 if that is the end; however, in this way it goes from s 1 to s 3 s 2, if a 1 comes, and it is here you can see it is even or odd, odd means numbers of one's is now odd. If another one comes the number of one's become even. Then again it goes back to this ah s 1 state s 1 where it produces a 1. Similarly, it is in state s 3 when both are odd that occurs whenever if it starts with 0 1 0 and another 1. So, it goes to state s three, but again if whenever it is in s 3 if 1 comes goes back to odd even; that means, the number of one's became even 1 and 1.

Similarly, it goes back to if a 0 comes it goes back to state s 2. So, that is how state-transition occurs, and at the end it may have a number of zeros and ones like this, say for example suppose the numbers of inputs are six. So, it may have 0 0 1 0 1 0 let us assume this is a sequence. So, here you can find the numbers of zero's is four. So, it is even and number of one's is two, so it is also even so; that means, it will produce 1. On the other hand if the input sequence is say 0 1 1 0 1 0 then the number of one's is odd, I mean 0 is odd and 1 is also odd. So, then it will produce 0 so; that means, here it is o o, state is o o. So, it will produce zero. So, in this case as you can see depending on whether, the final state is even even, odd even, even odd or odd odd, it will produce 1 or 0 only when the final state is even even, even and even both the number of 0 and number of one's are even, then it produces one. So, this is the example of a Moore machine.

(Refer Slide Time: 19:43)



FSM Design Steps

- **Step 1: The state-transition diagram:**
 - **The state machine function is captured in a state-transition diagram**
 - **Simple checks for drawing state-transition diagram:**
 - **Ensure that all states, including the 'idle' state, are captured**
 - **Check that transitions for all possible input combinations are leaving each of the states**
 - **Insert appropriate output to each transition**
 - **Insert loops into any state, with appropriate condition, for remaining in that state**

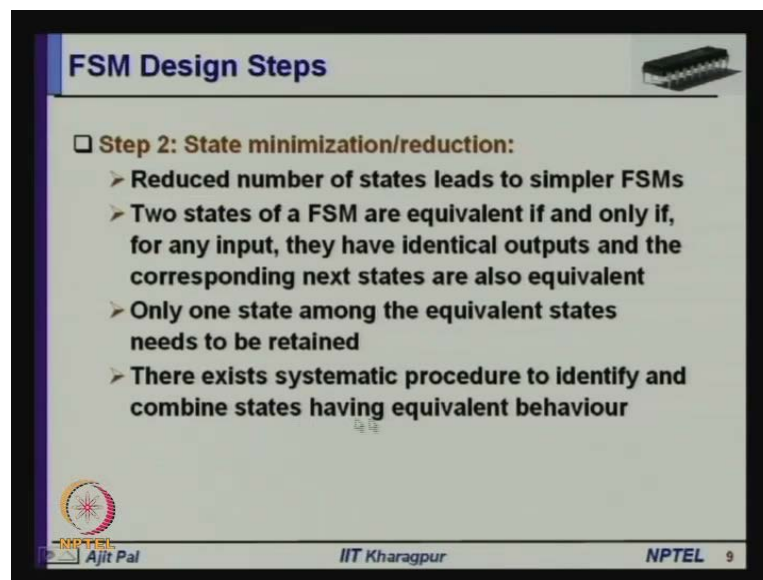
NPTTEL Ajit Pal IIT Kharagpur NPTTEL 8

Now, coming to the finite state machine design steps, how do you proceed. So, normally the specification may be given in terms of, as I said simple English text, or it can be specified with the help of the program C program, or we may specified it now a days with the help of some hardware description language like very long or VHDL. So, whatever it may be from that, you may create a state-transition diagram which I have already shown. Now, why you are actually realizing state-transition diagram, because the state-transition diagram represent, captures the testate functionality and the different transitions very nicely. Whenever you create the state-transition diagram you have to do some simple checks, you have to ensure that all states including the ideal states are captured; number two is you have to check that transitions for all possible input combinations are leaving each of the states. For example, in this case, in this particular example there is only one input so the number of possible input combinations is 0 or one.

So, from each state two edges will emerge; one is 0 for 0 another is for 1. Similarly, if there are two inputs then there will be 4 possible outputs 0 0 0 1 1 0 1 1. So, that you have to check that. So, that all possible transitions are captured by that state-transition function. Then insert appropriate output to each transition. So, you have to provide the necessary output based on the behavior, this given in the form of English text or whatever it maybe, you have to insert the appropriate output as I have already shown, that is necessary in the context of bilivation. However, in case of Moore machine the outputs are specified as part of the circle of the state, within the circle of the state. And

then insert loops into any state, with appropriate conditions for remaining in that state as I have shown, there are self loops, these loops, this type of loops, whenever continuous sequence of 0 comes or continuous sequences 1 comes, you have to give this kind of loops.

(Refer Slide Time: 22:42)



The slide is titled "FSM Design Steps" and features a small image of a circuit board in the top right corner. The main content is a list of points under the heading "Step 2: State minimization/reduction:". The points are:

- Reduced number of states leads to simpler FSMs
- Two states of a FSM are equivalent if and only if, for any input, they have identical outputs and the corresponding next states are also equivalent
- Only one state among the equivalent states needs to be retained
- There exists systematic procedure to identify and combine states having equivalent behaviour

At the bottom of the slide, there is a logo on the left, the name "Ajit Pal" in the center, and "IIT Kharagpur" and "NPTEL 9" on the right.

So, this is how we will create the state-transition diagram that is the step 1. Step 2 is state minimization and reduction. This plays a very important role, because reduced number of states leads to simpler finite state machines, what may happen, you may end off with a finite state machine with more number of states, than actually required. So, there will be redundancy and that redundancy is not desirable from the view point of implementation, because more the number of states, more hardware you will require to implement it. So, one important step is state minimization, and how it is being done. Actually two states of a finite state machine are equivalent, if and only if for any input they have identical outputs and the corresponding next states are also equivalent. So, what you have to do, you have to identify the set of equivalent states, and then only 1 of the equivalent states will be present in the final state transition diagram. So, it is important minimization step, and there is a systematic technique, systematic procedure to identify and combine states having equivalent behavior. We are not going to details how state minimization and reduction can be done, but this is available in any text book on switching circuits and logic design, including the book which I have studied in my student days; the book by G. Kowahi. So, I am not going to do the details of state minimization and reduction.

(Refer Slide Time: 24:28)

FSM Design Steps

Step 3: FSM State Encoding:

- > In the state assignment state, each state is given a **unique code**
- > States assignment strongly influences the **complexity** of its combinational logic part
- > Traditionally state assignment has been used to **optimize the area and/or delay** of the circuit

The diagram shows a state transition graph with five states: S1, S2, S3, S4, and S5. Transitions are labeled with binary inputs: 000, 001, 100, and 010,111.

States	Codes
S1	000
S2	001
S3	010
S4	011
S5	100

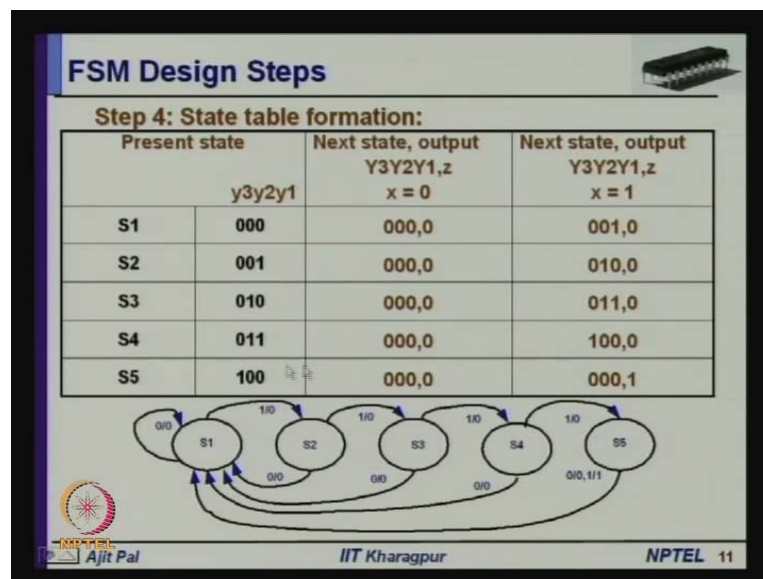
NPTEL Ajit Pal IIT Kharagpur NPTEL 10

Let us now consider another step, that is finite state encoding, what do you really mean by encoding. You see suppose if machine have got p states as I have already told, general finite state machine, we represent it by symbolic names $s_1 s_2 s_3$ up to s_p , but you have to realize the machine. So, each state has to be given a binary code, just the symbolic name is not enough. So, that is being done in this important step known as finite state encoding, finite state machine state encoding. So, in the state assignment state each state is given a unique code. So, each state will be given a code and that should be unique. So, that it does not, I mean there should not be a common code for 2 states, and why state assignment or state encoding actually, it is sometimes called state encoding, sometime it is called state assignment both are same thing. So, state assignment strongly influences the complexity of the combination logic part. So, depending on what state assignment you do, it influences the hardware; that means, realization of the finite state machine; that means, area require by the machine by the combinational part, or the delay of that circuit and later on we shall see it will even influences the power dissipations, is that is where we have the scope for minimizing power, later on we shall discuss about it.

So, traditionally state assignment has been used to optimize the area, and delay as I have already told, and recently this is we also used to minimize the power dissipation. And from the example that I have mention in the beginning; that means, it detects 5 one's in a sequence, for that machine the state assignment is shown here. It has got 5 states $s_1 s_2 s_3 s_4$ and s_5 , and you can see the codes I have just given serially; that means, s_1 is given

the code 0 0 0, s 2 is given the code 0 0 1, s 3 is given the code 0 1 0, s 4 is given the code 0 1 1, s 5 is given the code 1 0 0. Since, there are five states; you will require 3 bits; that means, with the help of 3 bits you can represent up to eight states. So, as long as the number of states is two, only 1 bit is sufficient. If the number of states is within four, 2 bits are sufficient so on; that means, if you will require k bits to have 2 to the power, sorry you will require if you have got k bits, then the number the total numbers of states, maximum number of states will be equal to p, where p is equal to 2 to the power k.

(Refer Slide Time: 27:43)



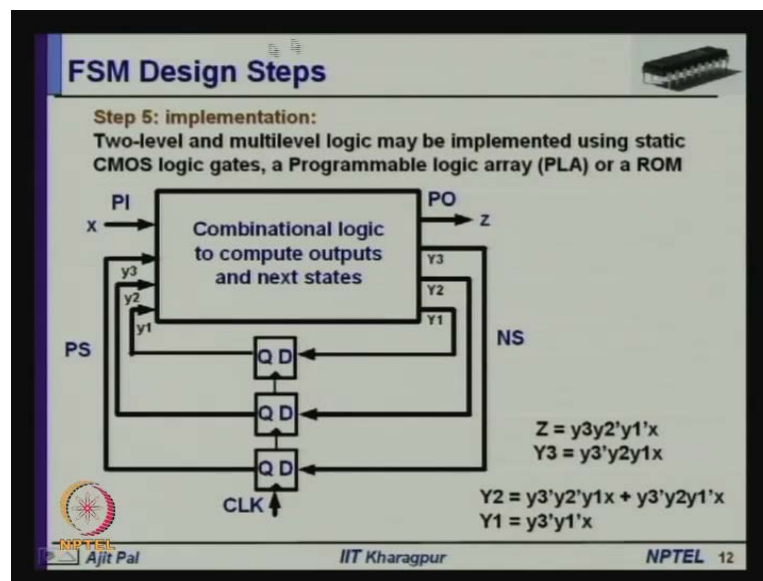
So, this is the second step and after that step we will start the fourth, I mean there will be fourth step, known as state table formation. So, here you can see, we have the 5 states s 1 s 2 s 3 s 4 and this s 5, and the corresponding states are given here s 1 s, the code for the different states and the corresponding states variables are y 3 y 2 and y 1. So, these are the states variables y 3 y 2 and y 1, and the corresponding codes are 0 0 0 0 0 1 0 1 0 0 1 one and 1 0 0. Now, you can easily form a table, from the state-transition diagram known as state table. So, this step is known as state table formation, how are you doing it. You can see suppose this is machine is in state s 1, this is the present state and if the input is 0 x 1 is 0, then what it does, it remains in state s 1 as it is shown 0 0 0 is the state s 1, and also it produces 0. So, you get you enter 0 here.

Similarly, when 1 comes as you input is 1 as you can see here, if it is input is 1 it goes to state s 1 and the corresponding code is 0 0 1 and output is also 0, as it is listed here zero.

So, in this way you can complete the entire table. For example, if it is in s 3, it goes to s four. So, s 3 is the, it goes to s 4 when the input is 1. So, the corresponding code is 0 1 1, the code for s 4 is 0 1 1 as you can see; however, the output is 0. Finally, as you can see when the machine is in state 5, it goes to state s 1, when the input is 0 by producing zero, so 0 0. So, here you can see there are 2 entries. So, 0 0 0 s 0 it goes to state s 1, and producing output 0, or it goes to state s 0 by producing one, so both are shown here. So, that means, when input is one, it produces one going to states one.

So, you can see this table is very useful, and this table somewhat is somewhat similar to truth table, and as you are all familiar with realization of Boolean function, from the truth table. So, once you have a truth table representation, you can very easily realize different functions, what functions you will realize. You will realize the next state functions and the output functions. We have already know the general representation on a finite stated machine. We have a set about outputs, primary outputs, and also you have to or generate the output for the next states.

(Refer Slide Time: 30:59)



So, next states and output of realized with the help of communicational circuit, and obviously, this will from our next step implementation. So, here what you are doing, you can implement the finite state machine with the help of some circuit; that means, combinational logic part, you will realize with the help of a circuit. So, there are several

alternatives in this case, the combinational logic as you know can be realized in a number of ways.

(Refer Slide Time: 31:28)

Realization of the combinational ckt. © CET I.I.T. KGP

1. Two-level or multilevel realization using gates (NAND, NOR, INVERTER)

$$z = y_3 y_2' y_1' x$$

$$Y_3 = y_3' y_2 y_1 x$$

$$Y_2 = y_3' y_2' y_1 x + y_3' y_2 y_1' x$$

$$Y_1 = y_3' y_2 y_1' x + y_3' y_2' y_1' x = y_3' y_1' x$$

The diagram shows a 4-input OR gate with inputs labeled y_3 , y_2 , y_1 , and x . The output is labeled z . The inputs y_3 and y_1 are shown with arrows pointing to the gate, while y_2 and x are shown with lines connecting to the gate.

NPTEL

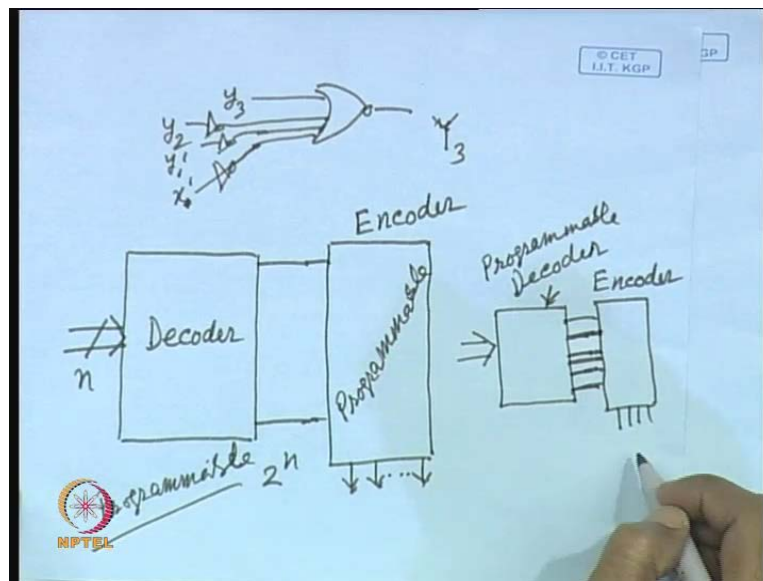
What are the different ways let me list out; number one is realization of the combinational circuit path, number one is you can realize by using either two level or multilevel with a realization, using gates, discrete gates like NAND, NOR, and INVERTER. So, we have already discuss, how you can realized functions by using this type of NAND, NOR, and INVERTER. In this particular case, from this truth table that I have shown; that means, the truth table of the finite state machine or state table of the finite state machine usually, you know for a combinational circuit we call a truth table, and for a sequential circuit with call it state table. So, usually they actually represent the same thing. So, in case of state table for this state table you have to realized y_3 y_2 y_1 and z . So, these are the 4 Boolean functions you have to realize, and here the inputs are I mean lower case y_3 , lower case y_2 lower case y_1 . On other hand you are realizing upper case y_3 , upper case y_2 , upper case y_1 and z .

So, this y_3 y_2 y_1 and z in this particular case are functions of small y_3 small y_2 small y_1 and x , and so it is essentially a 4 input 4 output realizations. You have got 4 binary input and it will produce 4 binary outputs, and the functions that you have to realized is given here. For example, z is equal to $y_3 y_2' y_1' x$, and y_3 capital Y_3 is equal to $y_3' y_2 y_1 x$, and y_2 is equal to $y_3' y_2' y_1 x$ plus $y_3' y_2 y_1' x$ $y_1' x$,

and y_1 is equal to $y_3 \bar{y}_2$, sorry $y_2 \bar{y}_1 \bar{x}$ plus $y_3 \bar{y}_2 \bar{y}_1 \bar{x}$, and here you can do little bit of minimization, other function cannot be minimized, but this can be minimized. So, you get y_3 combining this 2 you get $y_3 \bar{y}_2$, this y_2 will vanish, because here you have got y_2 and here it is $y_2 \bar{y}_1$. So, it will be $y_1 \bar{x}$. So, this is how you can realize the Boolean function y_1 capital Y 1.

So, once you have got these Boolean expressions, you can realized with the help of NAND, NOR, and INVERTER as you know this can be realized by single NOR gate. So, how it will realized you can say z will realize by a NOR gate, there are 4 inputs, what will be the inputs as you know the complement of the network from the n MOS path, here it will be here the inputs will be $y_3 \bar{y}_2$. So, you will require a inverter $y_3 \bar{y}_2$ then y_2 then y_1 and $x \bar{y}_1$. So, this will represent z , they realized z in terms of gates. So, it has it has been realized with the help of NAND, NOR, and INVERTER as you can see. So, this realization is done, using NAND, NOR, and INVERTER. Similarly, you can realize y_3 , y_3 can be realized in this way.

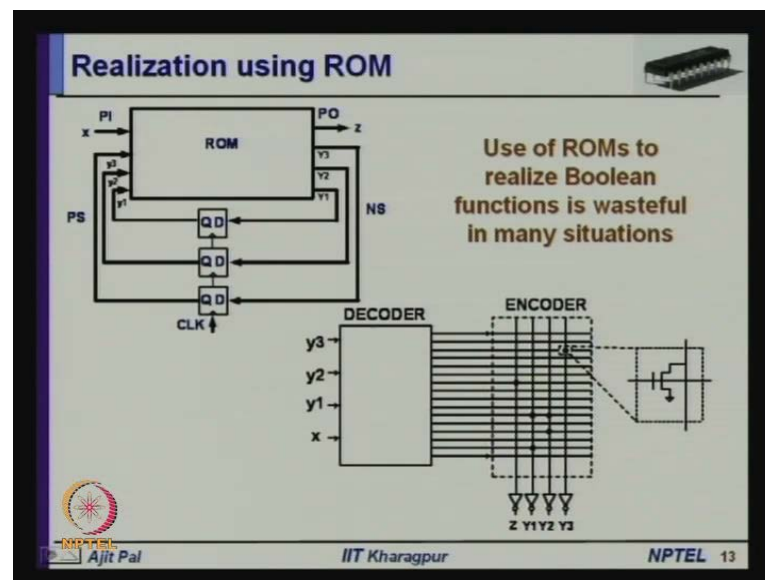
(Refer Slide Time: 36:37)



Again you will realized y_3 with the help of a NOR gate, and in this case you will require y_3 then $y_2 \bar{y}_1$, then you will required $y_1 \bar{x}$ and you will require $x \bar{y}_1$, sorry $x \bar{y}_1$. So, you'll require inverter. So, this is how you'll require the realized y_3 . On the other hand y_2 can be realized by using, because you require two level gates, that you can do and you can realized this by using again by using this simple NOR gate. So, here you

will have required y_3 , y_2 and y_1 and inverter. And this will require you know two more levels; that means, NOR at the final gate, and another NOR in the initial gate. So, you will require two levels of NOR and NOR here. So, this is given an exercise to you, you can try it out.

(Refer Slide Time: 38:07)



So, this is how you can realize functions by using NAND, NOR and gates, but that is not the only way by which you can realize functions. You can realize by using ROM read only memory. Sometimes you have to realize very complex circuit, so in that case the number of inputs and number of outputs can be quite large. So, realization of the combinational circuit by using read only memory is shown here, I have already explained that ROM can be used to realize combinational circuit, and here it is shown how it is being done. So, this x , y_3 , y_2 and y_1 , these are given as inputs to the ROM and it produces the output z , y_3 , y_2 and y_1 , how it does. You can see the structure of the ROM is given here, read only memory is given here, as you know ROM is nothing, but it consist of a decoder and an encoder. Decoder function as you know, it will produce 16 outputs for 4 inputs, because it is a decoder, and for any input combination the output will be one. So, it produces a one here on the bottom line, if all the inputs are 0. And it produces a 1 on the top line, if all the inputs are 1. And for all other input combinations, it will produce one of the remaining lines and that is a function of a decoder.

Now, what will have been present in the encoder part, I have already discuss about the realization of encoder part, by using MOS transistors, so that means, wherever there is a 1, I mean that particular four eight is there, you will have a transistor, in all other places transistors are not present. So, we have seen there are 5 terms $1\ 2\ 3\ 4$, 6 terms. So, as we have already seen in the function $1\ 2\ 3\ 4\ 5$, 5 terms; 5 terms are there, actually this has to be expanded into two, so there are 6 terms, because this capital Y 1, I have combined 2 mean terms to get this minimized form. So, essentially it has got 6 mean terms to realized the functions, and those 6 mean terms are shown here, with the help of 6 dots; that means, whenever a that particular a input combination is 1 for example, in this case it is $y_3\ \bar{y}_2\ \bar{y}_1$ and x , this is 1 this particular y_1 is 1. So, y_1 is 1, we have seen y_1 is 1, when $y_3\ \bar{y}_2\ \bar{y}_1\ \bar{y}_2$ and x . So, that realizes here. However, when this is 1 this line will be grounded. So, instead of 1 it will be produced 0. So, that is why you will require an inverter to really produce y_1 . So, you can see we have put an inverter at the output, so that it produces 1 instead of 0.

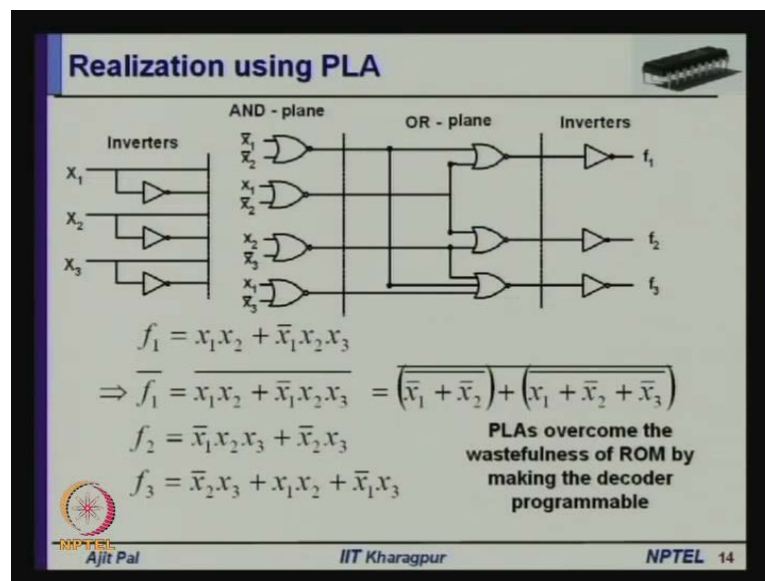
So, this is how you can use several most, I mean this is the implementation by using marks programmable ROM, that mean at the time of fabrication the marks is fabricated, the ROM is fabricated at be the marks stage; that means, the fabrication house does it. there are other alterative in discuss, where you can have instead of this type of marks programmable ROM. you can have electrically erasable ROM, in that case each particular node point will have a that floating gate transistor, and the floating gate transistors can be programmed with the help of different techniques, which I have elaborated in detail in my earlier lectures. So, let us not go into the details of that, but the basic idea is, you can realized this ROM function either by using marks programmable ROM or you can read by using electrically erasable ROM, or you can do it erasable programmable ROM, or you can have those flash memory. So, in that case also you will use those floating gates.

So, I am not going into the details of that, but this is one very interesting and useful way of realizing us finite stated machine. So, whenever you realize the finite state machine with the help of a ROM, one very important point you will noticed that, use of ROMs to realize Boolean functions is wasteful in many situations, why it is wasteful. Actually, when the number of input variables is very large, then total number of combinations is very large. Say there are ten inputs variables, I mean total number of inputs is ten

including the states variables and the inputs variables. So, it will require 2 to the power 10 different combinations.

So, out of those 2 to the power 10 combinations, usually a very small fraction is used to realize the function; that means, output is 1, only for a very small fraction of the total possible combinations. For example, in this case, out of 16 different combinations only 6 are required to produce the outputs z , y_3 , y_1 , y_2 and y_3 . So, that means, ten different combinations you can see that there is no dot. So, for ten different combinations there has no dots; that means, they are not required to realize these functions z , y , 1 , 2 and y_3 . However, you have to realize, I mean they will be generated by this decoder circuit, and unnecessarily you have made the decoder complicated to realize those outputs, to get those outputs. So, that is why I am calling it wasteful. Is there any way by which you can reduce this waste, or minimize this waste. Actually, another technique is there to realize a combinational circuit that is done with the help of what is known as programmable logic array PLA.

(Refer Slide Time: 44:49)



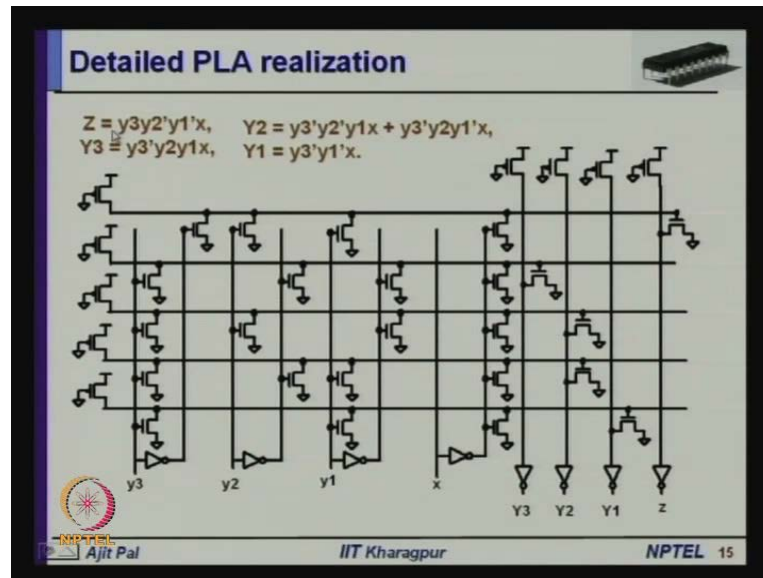
What is PLA. So, in PLA what has been done, we have seen a ROM comprises two parts; decoder if you have got n inputs, it will produce 2 to the power n outputs. And then there is an encoder, this encoder part is programmable, you can program it to produce different outputs. So, here decoder is not programmable, encoder is programmable, you can say programmable. Now, instead of this if we do it in a, if we incorporate little more

probability, what you can do. we can make the decoder probable; that means, we can apply the inputs, then we can have, instead of decoder we can have 2 levels, and it will produce only the desired outputs; say six, so it will produce the 6 outputs and which will go to the encoder. So, this is the encoder, but the decoder is then programmable. So, this is the probable decoder you can say, this one is programmable decoder. Only, the desired combination for which you require output one are generated here, not the redundant or unnecessary products. So, and then of course, it will produce the necessary outputs. So, that is the basic philosophy behind the programmable logic array.

And there are several structures, the most common one is the AND-NOR realization. Here, since you'll require both complemented and un-complemented outputs. So, each input is complemented and those outputs are fed to a stage known as AND plane, as if you are doing NAND, NOR realization. So, you are realizing this in this form say $x_1 \times x_2 \times \bar{x}_1$ bar $x_2 \times x_3$ one, so NAND, NOR. So, this is realizing the NAND operation, and this will perform the NOR operation. However, after performing this NAND and NOR, it may not produce f_1 , it may produce \bar{f}_1 , because it will be realizing not by using NAND, NOR logic, but by using NOR structures. So, since you will be realizing by NOR in this manner. So, this one NOR. So, you can see f_1 is equal to $x_1 \times x_2$ plus $x_1 \text{ bar} \times x_2 \times x_3$ that can be realized in this form. So, this is a NOR $x_1 \text{ bar} \times x_1 \text{ bar}$ plus $x_2 \text{ bar} \text{ bar}$ is the. This can be realized by NOR function, this can also be realized by NOR function, and then final NOR will realize the other function, and which can be complemented to get f_1 .

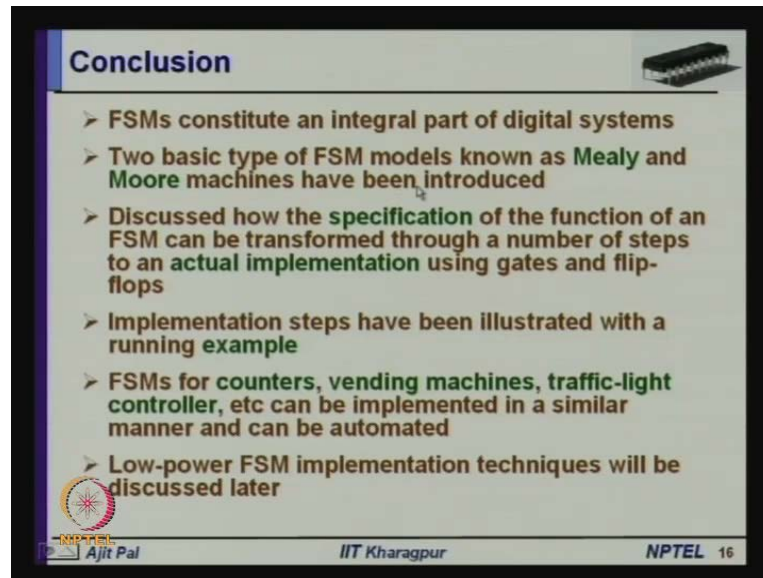
So, you can see f_1 is realized here, so to get f_1 you have to use inverter. So, this is the basic structure the most common type of PLA, and PLA as you can see, it is not producing unnecessary outputs. So, four outputs are required to realize f_1, f_2, f_3 , four combinations or product terms are required to produce to realize those functions. So, only those four are generated by this stage. So, you have made the decoder part programmable with the help of this AND plane, and then this OR plane is somewhat performing the function of that encoder of that ROM, and you are getting the necessary outputs, and of course, you will require inverter and you can get the functions.

(Refer Slide Time: 49:25)



So, the realization of the function that we have already done, in terms of transistors I have shown here. So, you can see this part, these are the three, four inputs x , y_1 , y_2 and y_3 these are the four inputs lower case y_3 , y_2 and y_1 and x and you have got inverters, which are applied to get, this is the NOR; five NOR function 1 2 3 4 5 we have seen there are five product terms. So, five NOR functions are generated, and these five transistors are essentially those pseudo and n MOS loads. So, this is the pseudo n MOS implementation. So, these are the loads, those pull-off transistors and this are the pull-down n MOS transistors for realizing different functions, different and plain functions. And then the NOR operation is performed by another NOR structure, which is shown here, and these are the loads or the pull-up devices of that NOR functions, the pseudo n MOS gates, and you can see these are the n MOS transistors. You require only five transistors here 1 2 3 4 5 to realize $y_3 y_2 y_1$ and $x z$. So, you see this is the detail PLA realization in terms of transistors. So, that basic structure that I have shown, I have elaborated it with the help of transistor this part, this n plain and NOR plain, how it is realized by using MOS transistors is shown here. This is for the, that five consecutive one sequence detector realization is shown here.

(Refer Slide Time: 51:20)



Conclusion

- FSMs constitute an integral part of digital systems
- Two basic type of FSM models known as Mealy and Moore machines have been introduced
- Discussed how the specification of the function of an FSM can be transformed through a number of steps to an actual implementation using gates and flip-flops
- Implementation steps have been illustrated with a running example
- FSMs for counters, vending machines, traffic-light controller, etc can be implemented in a similar manner and can be automated
- Low-power FSM implementation techniques will be discussed later

NPTEL
Ajit Pal
IIT Kharagpur
NPTEL 16

So, with this we shall come to the end of today's lecture, with these concluding remarks; first one is finite state machines constitute an integral part of digital systems as I have told, for any digital system you will require data path and control path, and control path can be realize using finite state machines. So, they will form an integral part of the digital system. Second is as I have told there are two basic types of FSM models, known as Mealy and Moore machines. I have introduced these two models, because any realization will use one of the two models, either the Moore machine or the Mealy machine. So, whenever it is a Moore machine if the output is dependent on only on the states, then it will be a Moore machine. On the other hand if the outputs are dependent on both inputs and states, then it will be a Mealy machines.

And we have discuss how the specification of the function of an finite state machine can be transformed, through a number of steps to actual implementation using gates and flip-flops, or even in terms of transistors as I have already explain, and various implementation steps have been illustrated with a running example. Now, finite state machines I have given example of only sequence detector. You may have the impression possibly finite state machines are used, only to realize sequence detector; that is not true. It can be used to realize any controller. For example, counter, counter is a kind of finite state machine, where without primary inputs.

Similarly, you can have shift register, ring counter or you can you may now a day's you will be finding vending machines in railway stations, airports even in IIT campus, you will find vending machines. The controller of the vending machines can be realized, say coffee vending machine or say newspaper vending machine, can be realized with the help of these finite state machines, or you can use realize traffic-light controller. So, these can be realized in a similar manner, and I have discussed everything by manual implementation, but when the machine is very complex. For example, you are realizing the control unit of a 32 bit micro processor; in that case manual design is not feasible. So, you have to use automated technique to realize a finite state machine, and fortunately automated tools and techniques are available. And later on we shall see low-power FSM implementation technique that I shall discuss later and for the time being thank you.