

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 36
Finite State Machine
(Contd.)

So this is the next example that we consider of Finite State Machine. So, in this case we are considering a lock that has that has got a digital input to unlock it and the input is 1 0 1.

(Refer Slide Time: 00:22)

Example - 101 lock

Combination lock with 101 being the combination

B	H ₀	L ₀	H _n	L _n	X
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	0	1	0

B is input signal to the lock, X is output signal to unlock

$|01 \leftarrow 101$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, that is there is a single input to this lock. So, conceptually you can think of it as a lock like this. And, it has got a B input and this B input is serial like. So, if it sees the sequence 1 0 1, then the lock will be opened. And, once so, every 1 0 1 sequence is different like say if I press this sequence 1 0 1, then the lock is open.

But, after this lock is opened so, it will go to a lock close mode. And, after that it will require another explicit 1 0 1 sequence for turning it on or opening the lock. So, we will see that. So, the state transition diagram that we have is something like this. So, initially we are at in this the first state, where you see that we can say we have seen a 0. The as long as we see 0 we stay in this state. So, we will we let us call the call the name of the

state as 0 seen and then when it is in the state 0 seen the output is 0. So, lock is not opened.

So, as long as we are getting B equal to 0 so, we continue in this state. Now, B equal to 1. So, this come it takes us to a state which we name as one seen, that is our lock unlocking pattern is 1 0 1. So, with a consider that we have seen the first one in that sequence. So, then also the output remains 0 that is the lock is not yet opened. And as long as B remains equal to 1 so, we continue to be in this state.

So, you see that if the sequence may be like this somebody may be entering values the 1 1 1 then 0 1.

(Refer Slide Time: 02:06)

Example - 101 lock

Combination lock with 101 being the combination

B	H ₀	L ₀	H _n	L _n	X
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	0	1	0

B is input signal to the lock,
X is output signal to unlock

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

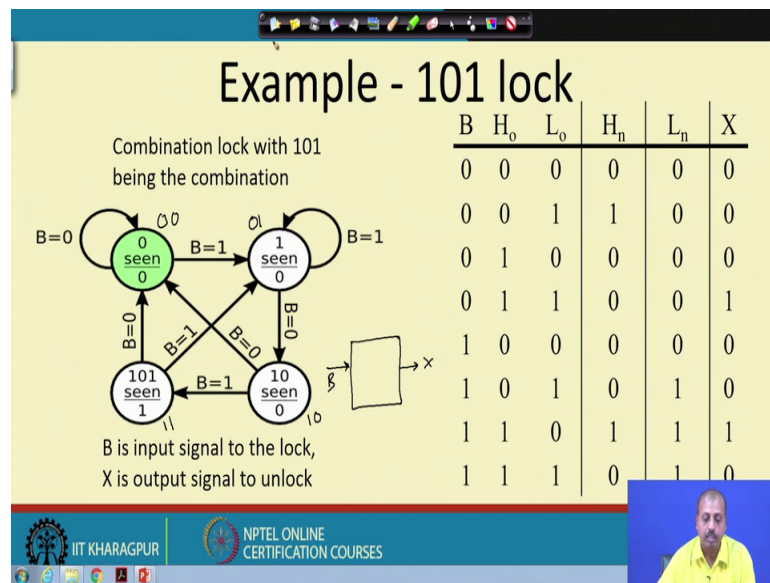
So, as long as these ones are going so, it will remain in this state 1 seen. And, then after that when this and the output remains 0. Now, if a B if the B input becomes 0 after that so, it comes to a state which we call 1 0 seen. So, in this 1 0 seen state also the output remains at 0 the lock is closed and then if we get a 1 then it is 1 0 1 seen.

And, when the 1 0 1 seen then the lock output is 1 that is lock is opened. Then, after if you if we get a 0 then it goes to 0 seen state that is lock becomes closed and we are we are in the state as if we have seen a 0 and if B equal to 1. So, this is a new unlocking sequence that I was talking about. So, B equal to 1. So, it takes as to this state.

So, again you need another explicit 1 0 1 for unlocking the lock. So, now if you press B equal to 1 or B equal to 0 the lock becomes closed. So, lock is; so, we can say that the as the lock is closed. And then this again another 1 0 1 input sequence is necessary for unlocking the lock.

So, B is the input signal to the lock and x is the output signal of the lock.

(Refer Slide Time: 03:27)



So, you can say that box that we that I had drawn the lock box. So, B is the input and X is the output. So, whenever we see 1 0 1 in B the lock is turned open. So, we can say that the state transition can be like this. They see if we have if we name these states first state as a 0, the code of the state is or if we make that the code of the state is 0 0 only.

So, this first state it is coded as 0 0, if B equal to 0 the next state is also 0 0 and X remains equal to 0. If, B equal to 0 and the current state is say 0 1, then the next state becomes 1 0 ok. So, if we if we are at state 1 so, this is this we code as 1. So, we will come to this coding part later. So, there are 4 states. So, in this particular case it is assumed that this state is coded as 0 0 this state is named as 0 1 this is 1 0 and this is as 1 1.

So, if the input state bits are 0 0 and the input is also 0 0, the next states are also 0 0. So, this is the first transition second transition says that if the current state is 0 1 that is this 1 and the input is 0, then it remains in this state only. So, sorry it remains in this state only.

So, it sorry no it goes to the state 2 so, 1 0. So, if H, if this is 0 1 and B is 0, then it is going to the state 1 0 and the output remains 0. So, that way we can make this full table. And, once this table has been made so, we can come we can do a simplification and get the next the combinational logic for doing this thing.

(Refer Slide Time: 05:12)

101 combination lock

$$X = H_0 L_0$$

$$H_n = B' H_0' L_0 + B H_0 L_0'$$

$$L_n = B H_0' L_0 + B H_0 L_0' + B H_0 L_0$$

$$= B H_0' L_0 + B H_0 L_0 + B H_0 L_0' + B H_0 L_0$$

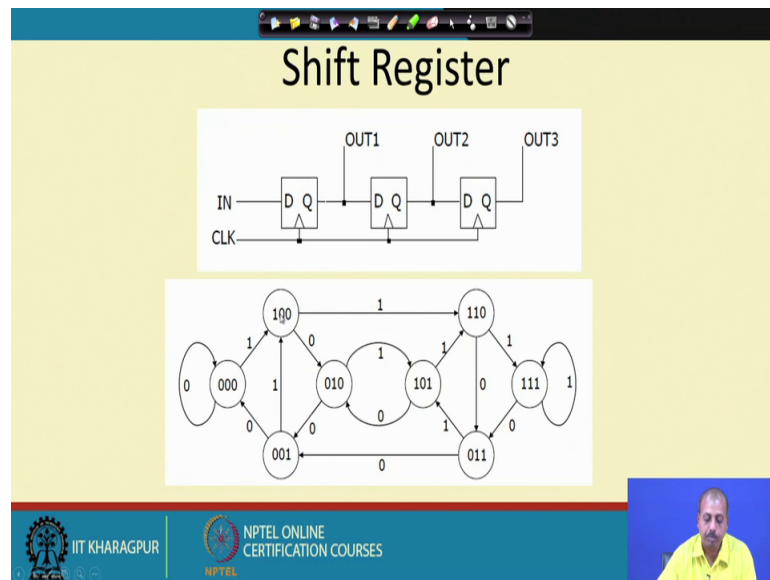
$$= B L_0 + B H_0$$

The diagram shows a logic circuit with three inputs: B, H₀, and L₀. The output is X. The circuit includes a 3-input AND gate for X, and two OR gates for H_n and L_n. The H_n OR gate has inputs B'H₀'L₀ and BH₀L₀'. The L_n OR gate has inputs BH₀'L₀, BH₀L₀, and BH₀L₀'. The circuit also features two flip-flops, one for H and one for L, which are fed back into the logic.

So, this X is given by H 0 1 0 and this or X H 0 1 0 some H n is given by this function and L n is given by this function. So, we can just make a combinational logic and we can say that we can we can put this combinational logic into some part of the circuit. So, this is the combinational logic. So, it has got B as a 1 input, then H 0 and L 0 as a other 2 inputs and then we have got these 2 flip flops, which holds the H and L which are L H and L values. So, this h next and L next so, they come here and they are fed to let us name this as a H 0 and this as a L 0.

So, this L 0 so, H 0 is fed here and L 0 is fed from this point. And, then this is our X output. So, in this part so, we realize the function for X equal to, then H n equal to, and L n equal to this individual functions are realized in this part. So, I am not drawing the combinational part there, but it can be realized using that.

(Refer Slide Time: 06:35)



Next we will be looking into another example, which is a shift register. So, this is the standard shift register that we have. So, as you know that with the input of every clock pulse this input bit gets shifted through this D flip flop chain.

Now, once this input pattern is known. So, you can try to draw the corresponding finite state machine. Now, we really do not know like what is the initial state of this 3 flip flops. So, it can be any of the states 0 to 7. So, if we draw so, if we assume that the initial state is 0 0 0, then if the on the arrival of the next clock with that at that point of time is in equal to 0. So, it remains in this state, if in equal to 1 it goes to this state 1 0 0.

Similarly, if the if the current state is 1 0 0 and the next input is 0, then it comes to the state 0 1 0 because this one gets shifted here and the 0 from the input comes to the first flip flop. So, this way you can complete this total finite state machine which we will so, from the circuit. So, this is this is the other way. So, from the circuit we can come to the finite state machine. So, here you know other combinational logic will be necessary, because these if you simplify this particular this particular machine for the state transition function, and output function you will find that it gives rise to the function like say this is $Q_2 Q_2 = D_2 = Q_1 D_1 D_3 = Q_2$ like that. So, it will come in a simple fashion like that.

So, that is why we do not show the circuit separately, but this is nothing, but this connection.

(Refer Slide Time: 08:15)

FSM design procedure

- Describe FSM behavior, e.g. state diagram
 - Inputs and Outputs
 - States (symbolic)
 - State transitions
- State diagram to state transition table, i.e. truth table
 - Inputs: inputs and current state
 - Outputs: outputs and next state
- State encoding
 - decide on representation of states
 - lots of choices
- Implementation
 - flip-flop for each state bit
 - synthesize combinational logic from encoded state table

Handwritten notes on the slide:

- Diagram of a flip-flop with inputs I_1, I_2 , clock CL , and output Q .
- State transition table:

	S_0	S_1	S_2	S_3
\rightarrow	00	01	10	11
\rightarrow	01	11	00	01
- 2-bit coding
- 2^2
- N no. of states
- min. no. of bits = $\lceil \log_2 N \rceil = n$
- 2^n

Now, how do you design an FSM finite state machine? So, first thing that we have by this time we know is the description of the finite state machine behavior. So, which is done in terms of state diagram? So, we can have some specification of a system in some English like language, and then we convert that description that English language description into a finite state machine. And, that finite state machine has got some inputs and outputs there will be some symbolic steps and there will be state transitions.

So, we can check whether this description that the finite state machine that we have drawn, it really reflects the a finite state machine behavior that was given as a input and that check can be manual that check can be based on some formal methods. So, which is beyond the scope of this course, but that verification can be done? Now, once we are satisfied that our state transition diagram whatever we have drawn, truly reflects the behavior that that has been asked for. So, we go to the next step which is a state diagram to state transition table.

So, you where inputs are the current state and the primary input so, they constitute the inputs and the outputs are the primary outputs and the next state. Now, once this from state diagram to state transition table we have drawn. So, next thing is the state encoding. So, every state is given some binary code ok. So, we naturally we need to decide like if there are say 4 binary were 4 states. So, if there are say 4 states say symbolically we represent it as $S_0 S_1 S_2$ and S_3 .

Now, for 4 states I will need 2 bit code to the at least a code of at least 2 bit 2 bit length to distinguish them. So, somebody may code it like this that the state S 0 will S 0 0 S 1 is 0 1 S 2 is 1 0 or and S 3 is 1 1. Now, there are other choices also like you see that I somebody may code it like this 1 as 0 1 this is this is 1 1 this is 0 0 and this is 0 1.

So, there is nothing there is no restriction regarding the choice that you make for the states. Somebody may even do this coding of higher number of bits like in this case so, if I do a 2 bit coding; so if I do a 2 bit coding then you see so, I can have 4 choices. So, it is so, the total number of alternatives that we have like this is alternative 1 this is alternative 2. So, then the number of alternatives that we have is 2 power 2 factorial of that, because this is I can just this 4 values.

So, I can arrange in any order. So, any permutation of those the 4 states that is good enough. So, it can give 4 factorial alternatives. And, if there are say N number of states. So, N is the number of states then the minimum number of bits. So, minimum number of bits needed is going to be log N to the base 2 so, these and the ceiling of that.

So, this is the value of N that we are talking about so, this is the so, this is the small n number of state bits needed. And so, if I do so, so, this is the minimum number. So, there is no restriction on the upper limit. So, we can go for any number of bits which is more than small n ok, but if I choose this smaller n number of bits then the number of alternatives that I have is 2 power n factorial. So, you see this is a huge search space that is there and we for each of these combinations. So, it is very much likely that the corresponding circuit that you get will be different.

So, so, basically if we say that my so, this is the finite state machine and we have got these flip flops ok. So, this is that combinational logic that we have so, that takes the primary input as in primary inputs and prime it produces the primary outputs, plus it gets lines from these flip flops the next state bits or the present state bits. It gets the present state bits from here and it produces the next state bits sorry let me draw a clearer diagram.

(Refer Slide Time: 12:50)

FSM design procedure

- Describe FSM behavior, e.g. state diagram
 - Inputs and Outputs
 - States (symbolic)
 - State transitions
- State diagram to state transition table, i.e. truth table
 - Inputs: inputs and current state
 - Outputs: outputs and next state
- State encoding
 - decide on representation of states
 - lots of choices ✓✓
- Implementation
 - flip-flop for each state bit
 - synthesize combinational logic from encoded state table

So, this is the combinational logic and we have got the flip flops fine. So, the primary inputs are connected to this combinational logic. Similarly, it produces the primary output and these flip flops content. So, they constitute the present state. So, they constitute the present state and it compute this combinational logic will compute the next state like this. So, this is a standard technique that we have seen. Now, you see that depending upon the coding like individual states what code we give this combinational logic is going to vary.

So, what is the best possible state assignment in terms of maybe the number of gates in the combinational logic or the power consumption of the combinational logic and these or maybe some other factor. So, that is a difficult problem. So, we there are lots of research works that have come up on this particular problem, which is known as the state encoding problem, which you come back to this later. So, I just I wanted to emphasize on this point that there are lots of choices in the state encoding process.

The next part is the implementation part, now these flip flops that I have said. So, I did not explicitly mention which type of flip flop we are using. So, normally while we are doing the design we use d type of flip flop. So, this is the d type of flip flop D and Q. So, D is getting this next state bit from the combinational logic and giving the present state output to the combinational logic.

Now, it is not necessary that we have to use D flip flop only somebody may say I will use a jk flip flop. So, I will use a J K flip flop, I accordingly 2 lines should come from the combinational logic to determine the J input and K input for the flip flops. Now, not just so, that is 1 reason that a why J K is generally not used, because of this reason that when you need to compute this combinational logic that we have so, they need to compute 2 functions per flip flop, one for the J input one for the K input, whereas if the flip flop is D, then the next state logic needs to compute only 1 function for a flip flop.

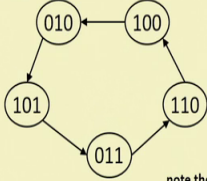
So, as a result it is, it may be easier that we do the realization using D, but that is not always true it because J K type of flip flop they offer other transition probabilities or the transition properties. So, that way may be for some finite state machine J K is doing better or maybe T type of flip flop may be used instead of this d flip flop.

However, in general in whatever designs that we see so, they use D flip flop because of it is simplicity you can say. And the because of this combinational logic design and the cad tools that have been designed. So, they specifically target it as a d flip flop. So, that is why d flip flop has remained as the standard one. Then after this is done so, if after these flip flops have been chosen. So, we synthesize combinational logic from encoded state table. So, that way we do the full synthesis. So, this is the full synthesis process. So, we will take an example and explain like how this is taking place?

(Refer Slide Time: 16:21)




Synthesis Example

- Implement simple count sequence: ~~000, 010, 011, 101, 110~~
 $100 \rightarrow 010 \rightarrow 101 \rightarrow 011 \rightarrow 110$
- Derive the state transition table from the state transition diagram



Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	x	x	x
0	0	1	x	x	x
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	x	x	x

note the don't care conditions that arise from the unused state codes

Suppose, we want to implement a counter that counts in this sequence 0 0 0 0 1 0 0 1 1 1 0 1 1 1 0 and after that it comes back to 000. So, the first point is to derive the state transition table from the state first a first step is basically the state transition diagram. So, in this case it is obvious. So, it starts at awaited sorry this diagram is something this sequence is something wrong I am sorry. So, this sequence is not correct the sequence, that it is doing is basically if you start at 1 0 0 then it goes to 0 1 0, then it goes to 1 0 1 then 0 1 1 0 1 1 and then 1 1 0 and then it is coming back to 100.

So, that is the sequence that it is doing. So, this is so, derivation of the state transition function is like this that if the present state is 0 0 0, then the next state is a do not care. So, the next state for 0 0 0 thus next state is do not care, similarly 0 0 1 also the next state is do not care. So, what happens if the counter reaches any of these states? So, that is not an issue with the designer. So, they are that is kept as do not do not care.

If you start at 0 1 0 then the next state should be 1 0 1. So, this way the present state next state relationship is done. So, do not care conditions that arise from unused state codes. So, that is basically ignored. So, this way we can do this we can do this state transition table part for this example.

(Refer Slide Time: 18:19)

Don't cares in FSMs (cont'd)

- Synthesize logic for next state functions derive input equations for flip-flops

		C			
		0	1	1	0
A	0	X	1	1	0
	1	X	1	X	0

		C			
		0	0	1	1
A	0	X	0	0	1
	1	X	1	X	1



		C			
		0	1	0	0
A	0	X	1	0	0
	1	X	0	X	1

C⁺ = B

B⁺ = A + B' C

A⁺ = A' C' + AC

(01 ← 101)

Now, after that is done. So, we can synthesize the logic for next state functions and derive the input equations for flip flops. So, that way you make the truth table from this

you make the truth table, from this graph, from this from this next state table the state transition table.

So, you can make the truth table for $C \text{ plus } B \text{ plus } A$. And, this is the functionality that we get $C \text{ plus } B \text{ plus } A$ like that and then we can go for realizing the circuit.

(Refer Slide Time: 18:52)

The slide is titled "Self-starting FSMs" and contains the following content:

- Start-up states
 - at power-up, FSM may be in an used or invalid state
 - design must guarantee that it (eventually) enters a valid state
- Self-starting solution
 - design FSM so that all the invalid states eventually transition to a valid state may limit exploitation of don't cares

The diagram shows a state transition graph with states 000, 001, 010, 011, 100, 101, 110, and 111. State 111 has a self-loop. Transitions are: 000 to 011, 001 to 010, 010 to 100, 011 to 110, 100 to 110, 101 to 110, and 110 to 110. A note says "implementation on previous slide".

Logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES are at the bottom.

Sometimes we are looking for some self-starting FSMs. So, self-starting FSMs so, they have got some startup states so, at power up. So, FSM maybe in an used or invalid state, where design must guarantee that it eventually enters into a valid state.

So, this is self-starting FSMs design the FSM. So, that all invalid states they eventually make a transition to the valid state and this way this limits the exploitation of do not care like say. So, we previously we had say this while we are writing this is 000. So, we took the next state as do not care, but here that we cannot take so, 000 so, we put it as if you if you start the machine at 0 0 0 it will come to 0 1 1.

Similarly, if you start at 0 0 1 so, it will come to the state 0 1 0. So, that way it is restricted, but it is not arbitrary like in a previous example that we have seen. So, there if you start at some state which is invalid it is unknown. So, what will be the next state? So, that is not known, but in this case it is completely specified. So, this self-starting FSM

so, they will always either keep you in this cycle always all one or it will take you to one of these 5 states ok, after some time.

(Refer Slide Time: 20:15)

Self-starting FSMs

Deriving state transition table from don't care assignment

	C		
	0	1	0
A	0	1	0
	B		

	C		
	1	0	1
A	1	1	1
	B		

	C		
	1	1	0
A	0	0	1
	B		

Present State			Next State		
C	B	A	C ⁺	B ⁺	A ⁺
0	0	0	0	1	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	X

And, during by the going by the same mechanism that we have drawn so, you can draw the as state transition table like say from 0 0 0 it is going to 0 1 1. So, 0 0 0 it is going to 0 1 1. Similarly, 0 0 1 it is going to 0 1 0. So, rest from 1 1 1 it remains it is 1 1 do not care. So, from 1 1 1, it it can either remain in 1 1 1 or it can come to 1 1 0. So, that way it is left to the designer the specification does not tell what will happen to 1 1 1 clearly.

But, it can go to either it can remain either in this state or it can come to this state. So, if that is specified then it has to be 1 1 X, but if we truly if we want to follow this diagram truly then for this 1 1 1 the next state should also be 1 1 1. Anyway so, once we had done with this state transition table so, we can we can make the corresponding circuit and get it done.

(Refer Slide Time: 21:20)

Comparison of Mealy and Moore machines

- Mealy machines tend to have fewer states
 - different outputs on arcs ($i*n$) rather than states (n)
- Mealy machines react faster to inputs
 - react in same cycle – don't need to wait for clock
 - delay to output depends on arrival of input
- Moore machines are generally safer to use
 - outputs change at clock edge (always one cycle later)
 - in Mealy machines, input change can cause output change as soon as logic is done – a big problem when two machines are interconnected – asynchronous feedback

The slide includes several diagrams: a Mealy machine state transition diagram with an output on the transition arc, a Moore machine state transition diagram with an output associated with a state, and two examples of asynchronous feedback loops. One example shows a state transition labeled $a=0/b=1$ leading to an output z . The other shows a state transition labeled $a=0$ leading to an output z .

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we make a so, look into a comparison between Mealy machine and Moore machine. So, mealy machines they tend to have fewer states, because the output is a function of state present state and input. So, as a result so, we many of the Moore machine states so, they are copy they are they can be combined in a mealy in a mealy machine state. Because of the reason that for Moore machine for every even the other otherwise the 2 states being same only the output being different. So, they are going to constitute 2 different states.

Whereas for mealy machine. So, they can be clubbed into one state and we can represent the transitions by means of these inputs also. The input of the inputs primary input they can also be considered as some constraint on the primary output. Second point is the mealy machines react faster to inputs, because of this reason that in case of mealy machine what is happening is that suppose this is the current state and there is a transition like this to this state.

So, input slash some say inputs slash some output. So, if in this state if the FSM is in this state then as soon as this input condition is satisfied. For example, if I say that from this state it is going to this state and the condition is a equal to 0. And in that case it will make B equal to 1. So, in if the machine is in this state then whenever a is equal to 0 B will be made equal to 1. So, that is possible for mealy machine, but because some Moore machine what is happening is that this output is a part of the state itself.

So, it will be like this say a equal to 0 a equal to 0. So, it will take to this state where it will make B equal to 1. Now, this transition of state cannot occur until the next clock pulse arrives, because the next state function that it will that the combinational logic computes. So, it is put into this flip flop and this, but this flip flop output will come to this point only when this clock has arrived to this flip otherwise this value is not available at the output of the flip flop.

So, as a result even if a equal to 0 till the clock pulse comes thus the machine does not transit from this state to that state. So, B does not become 1, till the transition has take place or we can say that till the next clock pulse has arrived. So, there is a delay to the output depends on arrival of so, this react in some react in some in same cycle do not need to wait for clock and delay to output depends on arrival of input. Whereas, Moore machines so, they are generally safer to use because the output will change at the clock edge.

So, always one cycle later; so it can be a bit safe in our design in mealy machine input a change can cause output change as soon as logic is done. So, sometimes it is a problem particularly if we have got 2 machines that are interconnected with an asynchronous feedback. So, one machine making a transition that can affect another machines operation; so if it is a mealy machine type of if it is a Moore machine type of realization then both these machines will transit at the next clock pulse.

So, they are going to behave in a better fashion compared to a mealy machine where one of the machine may makes some transition in between, because the change in the input combination and as a result the output may become that is the interconnection that the interface may become problematic sometimes this is seen. So, though there is no hard and fast rule like which one will be used, but in general mealy machine has got less number of states. So, for a large machine; so, we will go for mealy type of realization.

(Refer Slide Time: 25:14)

Implementing an FSM

- 1. Perform state assignment
 - different assignments may give very different results
 - no really good heuristics
 - using an extra bit or two for state, works well
 - FPGAs often use a 1-hot encoding
- 2. Convert state diagram to state table
 - equivalent representation
 - mechanical
- 3. State table gives truth table for next state and output functions
 - synthesize into logic circuit
 - e.g. 2-level logic implementation

Handwritten assignments:
 $S_0 \rightarrow 0001$
 $S_1 \rightarrow 0010$
 $S_2 \rightarrow \cancel{00}0100$
 $S_3 \rightarrow 1000$

Handwritten diagram: $0 \rightarrow 0$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, the state steps for coming back to the steps for implementing a finite state machine. So, first point is to perform state assignment. So, we can make different assignments that can give different results, there is a as I said that it is an NP hard problem. So, there is we cannot solve the problem optimally with a polynomial time algorithm. So, as a result there is no really good heuristics that exists.

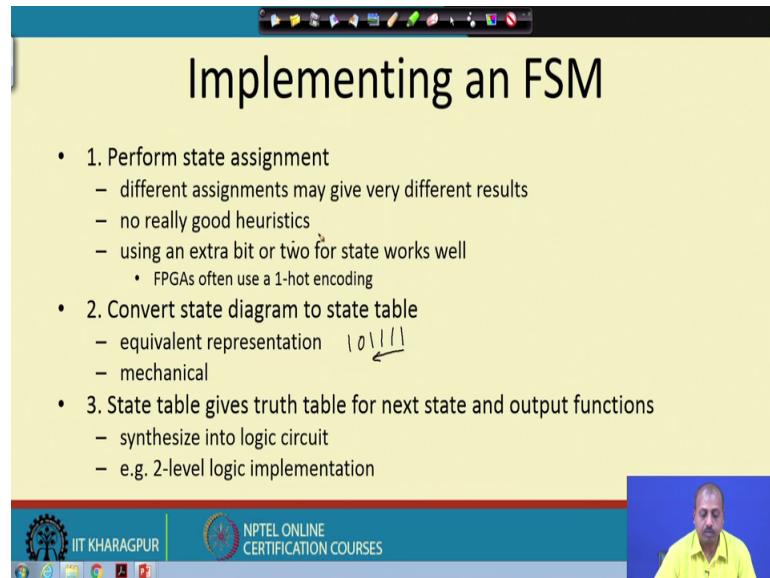
Then sometimes we use some extra bit 1 bit or 2 bit extra that for that makes the combinational logic simpler and the extreme point is with FPGAs where we do one hot encoding. So, one hot encoding is like this like if I have got say 4 states S_0 S_1 S_2 and S_3 , then I can say that for S_0 I have a code like this, 0 0 0 1 S_1 is a code the code is like this, S_2 is a code like 0 0 sorry 0 1 0 0 and for S_3 the code is 1 0 0 0.

Now, whenever there is a transition from one state to another state in the finite state machine you see only 2 bits are going to change ok. So, only 2 bits are going to change. So, that makes it interesting. So, that the combinational logic that we have. So, that becomes much simpler since only 2 bits will be changing. So, this type of coding is known as a 1 hot encoding the 1 hot means in the state code only 1 bit is 1 and all are the remaining bits are 0s.

And, this is particularly useful when we have got this flip flop rich architectures like FPGAs field programmable gate arrays; we will see that later. So, this 1 hot encoding is very popular. So, we convert state diagram to state table we can do equivalent

representation or we can do some it is a mechanical process. And, then we can get the state table truth table for the next state and output function and then go for a logic minimization logical implementation.

(Refer Slide Time: 27:15)

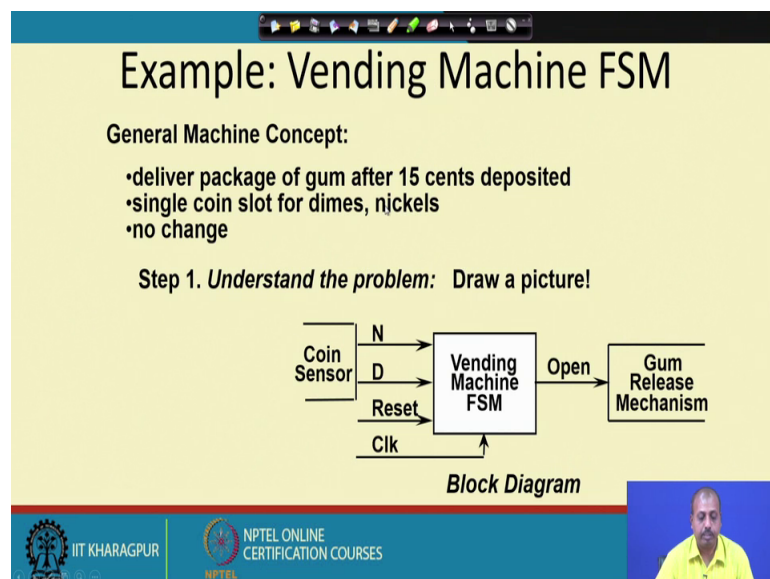


Implementing an FSM

- 1. Perform state assignment
 - different assignments may give very different results
 - no really good heuristics
 - using an extra bit or two for state works well
 - FPGAs often use a 1-hot encoding
- 2. Convert state diagram to state table
 - equivalent representation 10111 ←
 - mechanical
- 3. State table gives truth table for next state and output functions
 - synthesize into logic circuit
 - e.g. 2-level logic implementation

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

(Refer Slide Time: 27:19)



Example: Vending Machine FSM

General Machine Concept:

- deliver package of gum after 15 cents deposited
- single coin slot for dimes, nickels
- no change

Step 1. Understand the problem: Draw a picture!

Block Diagram

```
graph LR; subgraph Coin_Sensor [Coin Sensor]; N; D; Reset; Clk; end; Vending[Vending Machine FSM]; Gum[Gum Release Mechanism]; Clk --> Vending; N --> Vending; D --> Vending; Reset --> Vending; Vending -- Open --> Gum;
```

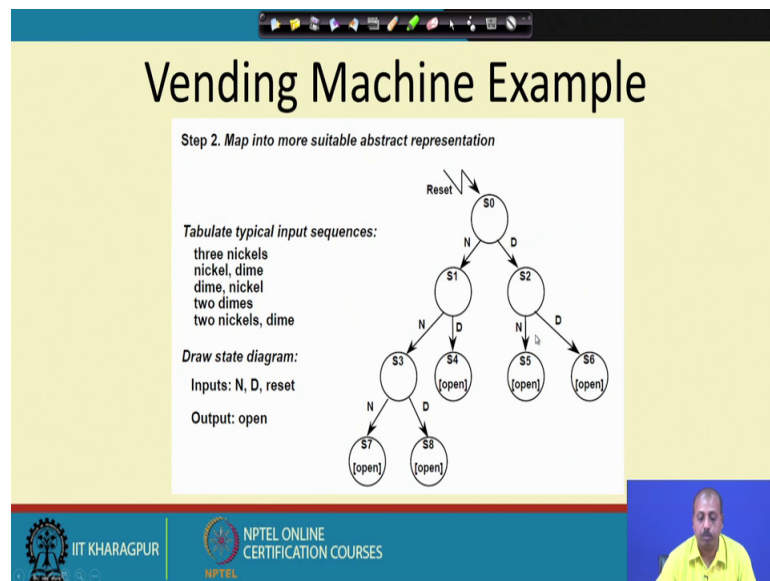
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if we look into an example like we are looking into your vending machine. So, it delivers a package of gum after 15 cents have been deposited. There is a single coin slot for dimes nickels and there is no change no change is given back. So, first stage is first step is to understand the problem. So, this is my vending state machine FSM there is a

coin sensor that senses nickel and dime and there is a reset that will reset the system and there is a clock signal.

So, the when appropriate change appropriate some 15 cents or a more has been given, then this slot opens and 1 gum is released. So, gum release mechanism is given the open signal. So, that is the specification of the problem.

(Refer Slide Time: 28:01)



So, you can say that we can go to 15 cents, how can we go to 15 cents? Ok. So, there can be 3 nickels 1 nickel 1 dime so, nickel is equivalent to 5 cents dime is equivalent to 10 cents. So, this one sequence may be a 3 nickels, nickel dime nickel 2 dimes and we do not give the change 2 nickels and 1 dime.

So, that way we do not give the change. So, whenever it becomes just higher than 15 cents the gum is released. So, it is you start at state S0. So, this sequence is in nickel nickel. So, going to the states S1 S2 S1 S0 S1 S3 and S7 and in the state S7 the open signal is equal to 1 or it may be dime dime going like S2 S6 states.

So, we draw this diagram after this diagram has been drawn. So, we look into the behavior and get the confidence that this really represents the behavior properly.

(Refer Slide Time: 28:56)

Vending Machine Example

Step 3: State Minimization

Present State	Inputs D	Inputs N	Next State	Output Open
0¢	0	0	0¢	0
	1	0	5¢	0
	1	1	10¢	0
5¢	0	0	X	X
	0	1	10¢	0
	1	0	15¢	0
10¢	1	1	X	X
	0	0	10¢	0
	0	1	15¢	0
15¢	1	0	15¢	0
	1	1	X	X
	X	X	15¢	1

reuse states whenever possible

Symbolic State Table

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Then we do some state minimization. So, all these states that we have drawn here they are not required and they can be minimized into this thing. So, on reset it enters into a state called 0 cent and then if they even if the nickel is placed given then it goes to 5 cent state as if it has seen 5 cent.

Another, nickel comes to the state called 10 cent and from 0 cent also if they are if a dime is given it comes to the 10 cent state. Similarly, for the 5 cent state if a dime is given it comes to 15 cent and from the 10 cent state if it comes to the state 15 cent. Of course, so, this is from 10 cent if you put a dime also though it is 20 cent, but we do not need to remember that as a separate state because no separate action is necessary. So, that is 15 cent only.

So, after that if we code the states like say the present state is 0 cent. So, 0 cent then if the input given is 0 0, then it remains a 0 state that is no input is given and output remains 0. Then you if a nickel is given it comes to the state 5 cent and the output remains 0. So, in this way you can draw the symbolic state table.

(Refer Slide Time: 30:12)

Vending Machine Example

Step 4: State Encoding

Present State		Inputs		Next State		Output
Q ₁	Q ₀	D	N	D ₁	D ₀	Open
0	0	0	0	0	0	0
		0	1	0	1	0
		1	0	1	0	0
		1	1	X	X	X
0	1	0	0	0	1	0
		0	1	1	0	0
		1	0	1	1	0
		1	1	X	X	X
1	0	0	0	1	0	0
		0	1	1	1	0
		1	0	1	1	0
		1	1	X	X	X
1	1	0	0	1	1	1
		0	1	1	1	1
		1	0	1	1	1
		1	1	X	X	X

Once you have drawn the symbolic state table the next part is the state encoding. So, this Q₀ present state. So, this 50 cent 5 cent 15 10 cent and 15 cent they are given the code 000110 and 11 accordingly, you get this next state function. Once this state of state encoding has been done so, we have got this truth table.

(Refer Slide Time: 30:31)

Vending Machine Example

Step 5. Choose FFs for implementation D FF easiest to use

K-map for D₁

K-map for D₀

K-map for Open

$D_1 = Q_1 + D + Q_0 N$
 $D_0 = N Q_0 + Q_0 \bar{N} + Q_1 N + Q_1 D$
 $OPEN = Q_1 Q_0$
8 Gates

So, this truth table you can now minimize using some Karnaugh map for individual D₁ and a D₁ B₀ and open for the 3 for these 3 3 functions you can get. So, this is the function that is obtained a D₁ equal to Q₁ plus D plus Q₀ N. So, like that you can find out that this is the function obtained. And, ultimately we can realize it by means of some combinational logic function like this ok. So, this is the vending machine example.

So, in this way any find given any finite state machine. So, you can start with the behavior come to the corresponding the state transition diagram, then from there the state table and then count encode the states using some codes. And then come to the combinational logic to be realized and then get the final function realized in terms of logic gates and flip flops.