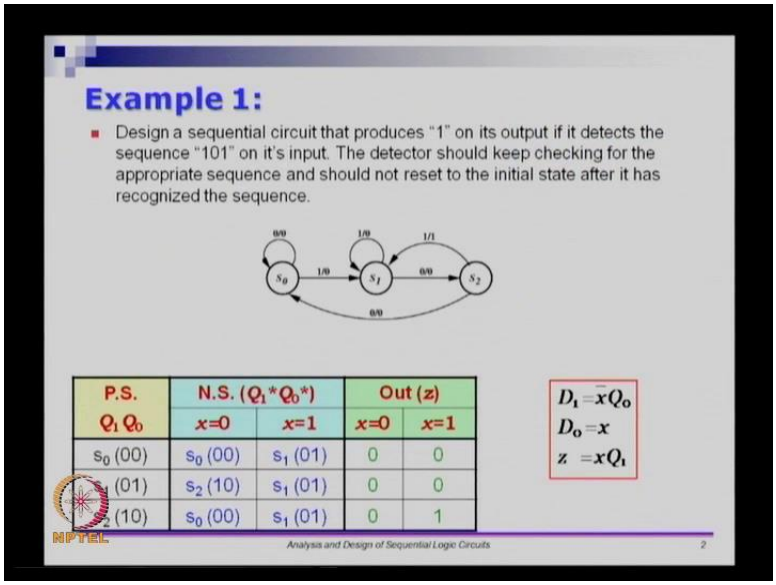
Digital Circuits and Systems
Prof. Shankar Balachandran
Department of Electrical Engineering
Indian Institute of Technology, Bombay
And
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 30
State Machines 2: Design Problems

Welcome to module 30. In this module, we are going to look at several design problems. I am going to state problems in English and I am going to show you how to Designs State Machine for this. So, in the process we will also learn a lot of small tricks and techniques, so that when you design problems, if you are given problems on your own and if you are asked to design circuits for that, you will be able to do those things yourself.

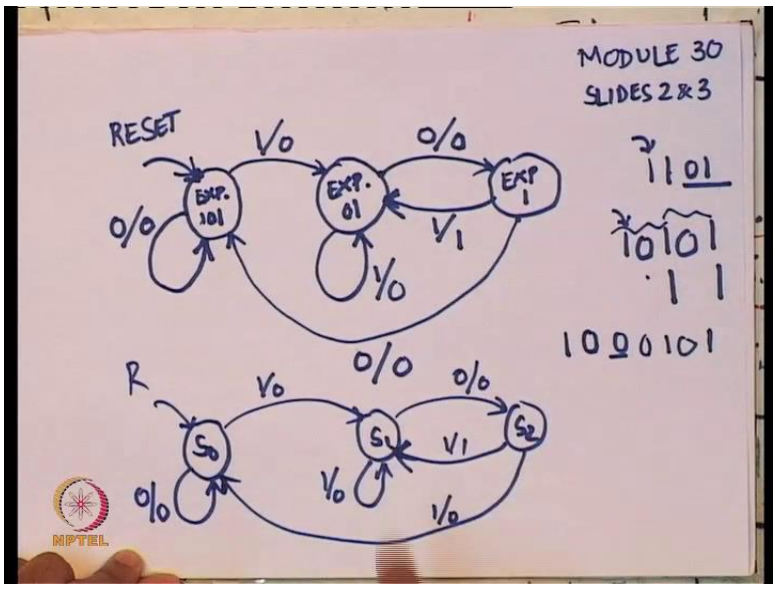
(Refer Slide Time: 00:47)



So, let us start with the example problem, so the example problem seems to be wordy, but let us see what it is. So, it says design a sequential circuit that produces 1 on it is output, if it detects the sequence 1 0 1. So, we have already done something similar earlier, but on seeing a 1 0 1 it should produce a 1 and the detector should keep checking for the appropriate sequence, the sequence 1 0 1 and should not reset the initial state, after it has recognize the sequence.

So, as in when there is 1 0 1 that is coming in, it is supposed to check it and whenever there is a 1 0 1, it is supposed to give a 1. So, for instance if you are input sequence was 1 0 1 0 1, then at the third cycle you see a 1 0 1. So, the output should be 1 and again at the fifth cycle the output should be 1, because there is this over lap of 1 0 1, so you have to recognize that also. So, this is the problem description and I want to now think about this in terms of a state machine, so I want to show how to do that. So, let us assume that I am going to build a state machine.

(Refer Slide Time: 01:55)



So, if I have not seen any input at all, so that is usually call the reset state. So, I have reset and usually we mark it like this, so that this is the starting state. So, in the starting state what we have is, we do not have anything, we have not seen even a single bits, so far. So, what I can then remark here is, I can say that I am going to expect 1 0 1, I am going to name it like this, I am expecting 1 0 1.

So, once I know that it is 1 0 1, I will then go and mark it. So, now there are two possibilities, if the machine is starting with this state. So, I will do a reset to the circuit the machine is going to start in this state. Now, there are two possibilities, the possibilities are that I can get a 0 as the input here or 1 as the input here. So, now if I start with a 0, let us say I get a 0 actually then... So, I am expecting 1 0 1, but I am getting an input as a 0.

So, if I get a input as 0, I am not getting a 1, so I can actually stay in the state itself. So, you see this, so I am expecting 1 0 1, but I am getting a 0. So, I cannot start the string 1 0 1, if it start with 0, so if it 0 it is as could as I am going to expect 1 0 1 itself, I will stay here itself. But, if I get a 1, the output should be 0, because I have not seen 1 0 1, I have just seen 1, but I can go to a new state which says I am expecting 0 1.

So, we have seen 1 here and you are expecting 0 1 here, so we have now two states. So, in this state now again there can be 0 as an input or 1 as an input. So, if I get 0 as an input, then this is possibly a good thing, because if I get 0 as an input. So, I know that I am expecting 1 0 1, I have already came to this state, it is expecting a 0 1 and now it is expecting it got a 0, so it has still not detect the sequence 1 0 1. So, I can go to a state which says expecting 1 just 1.

However, if I now get a 1, then there are choices where should I go. So, the easiest thing to do is, so this one is expecting a 1 0 1. So, from this the whole sequence is 1 0 1, from here we are expecting 0 1 anyway. So, if I get a 1, I could is look back to itself, this not going to change anything. So, if I get 1 1 as an input, so it is possible that I get 1 1 as an input and after that I may get a 0 1.

So, this 1 has already taken we to this state, this 1 will keep me in this state itself. So, that when I see the next 0 1, I can make other transitions. So, that is what you have now and now let us see I come to this state, in this state now again there are two choices. So, I could get a 1 or I could get a 0, so if I get a 1 let us see what happens. So, I have a 1 0 and if it is a 1 I want to be allowing cases like these 1 0 1 0 1, if I have a case like this I want the output to be 1 here, as well as 1 here.

So, the first 1 will... So, the reset state is here, the first 1 would have taken me here. The 0 would have taken me here and with this 1 if I go back here with the output of 1. What this we do is, if I have a sequence like 1 0 1 0 1 then I can do 1 0 1 0 1 and so on. So, this state is anyway expecting a 0 1, so if there is an overlap between 1 sequence 1 0 1 and another sequence 1 0 1, this state can capture that.

So, we can go there what if we get a 0 here, if we get a 0 here then we have the prefix. So, we came with someone and then 0 you are hoping that this will be a 1, but we got a 0 here. So, by if you get a 0 from this state, it means you already saw 1 0 before and you are seeing the second 0 now, this cannot potentially start a 1 0 1 sequence. So, I have to

now expect of complete 1 0 1 sequence and not a partial 0 1 sequence, which means I have to go back here.

So, on a 0 I will output a 0 and what I will do is. So, this state means I am expecting a 1, but if I get a 0, then if I go all the way back here, I can handle cases like this 1 0 0 0 1 0 1. Let us say I have an input sequence like this 1 0 0 0 1 0 1 and so you can handle such cases. So, whenever there is more than two 0s it cannot be a valid prefix for the 1 0 1 sequence. So, immediately you can say that you can go back to this state, so that you can continue from here.

So, this is a state machine, so what I have done is I have given some meaningful descriptions inside this state, I am expecting 1 0 1 here, I am expecting 0 1 here, I am expecting 1 here. So, the moment I see a 1, I know that I have to produce a 1, but if I get a 0 I need to figure out which of these states I have to go back too. Now, I can simplify this and instead write it as s naught, s 1 and s 2.

So, instead of having names which are expecting 1 0 1 and so on, I can use these things s naught, s 1 and s 2 and I can draw the same transitions. So, 1 0 1, so this takes care of what we want from the circuit, then there is reset and all other cases that we do not want. So, we can mark all of them and all these are not what we are looking for, you can mark all of them. So, this what you see in the bottom is a valid state machine.

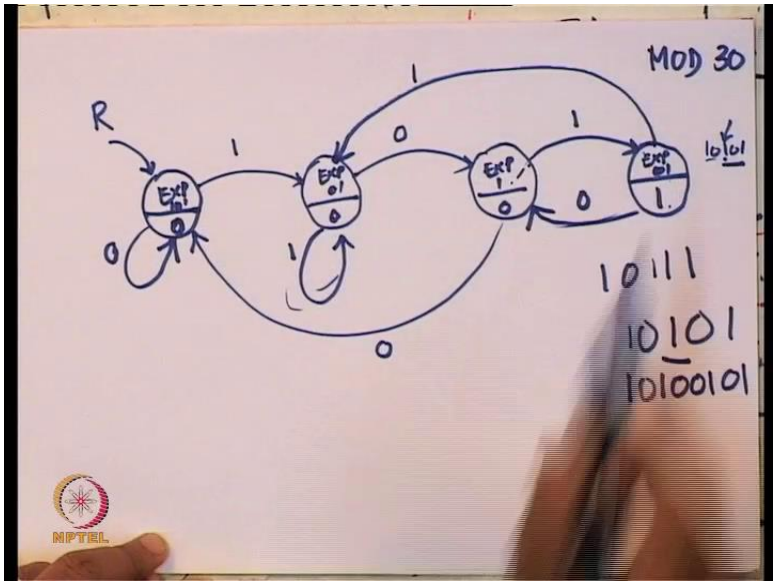
So, now this is a Mealy machine, because the state along with the transition dictates the output. So, for example, from this state, on this transition you get a 1, on this transition you get a 0. So, depending on the state as well as the input, the output is decided, so this is called a Mealy machine and what we have here is a mealy machine ((Refer Time: 08:37)). So, this the mealy machine that I did, so you have the same mealy machine describe here with state s naught, s 1 and s 2 and this you can draw the state table.

So, for s naught let us say I call this with 0 0, because there are three states I will need 2 state registers. So, I will call them Q 1 and Q naught, so s naught is 0 0, s 1 is 0 1, s 2 is 1 0. So, assigning these numbers to this state is called state assignment problem and from the current state based on x equals 0 or x equals 1, you have these as the next states and these as the outputs and you can derive logical equations for it.

So, if I want to implement this with a D flip flop, then D 1 for this assignment happens to be x bar Q naught and D naught happens to be x itself and z which is the output seems to be x and Q 1. So, you know how to do this, but mean I taught you how to do this in the last video, I will suggest that you actually go and try this now and you can see that these equations are the same.

So, if you use the sequence 0 0 0 1 and 1 0 for these three states, you will get these equations. So, this is the Mealy way of doing it, let me also show you the Moore way of doing it. So, I am going to do the same circuit, but I am going to do it as a Moore machine. So, in a Moore machine remember the states uniquely determine what the outputs must be.

(Refer Slide Time: 10:15)



So, again I will start with this one I am expecting 1 0 1, but if I am expecting 1 0 1, I cannot produce an output 1, because I am not seen any sequence yet. So, I will mark the output as 0 below, so within the state itself you will mark the output in Moore machines and this is the reset state. So, if I am in the state expect 1 0 1, the output must be 0. Now, if I get a 1 then I will go to the state expecting 0 1 and the output is still 0, from here if I have a 0, then I will go to a state which says expecting 1, the output is still 0.

And now on a 1, I can go to the state, which is actually should produce an output 1 first of all. Because, you are seen 1, 0 and a 1, so at this point you can produce an output of 1, but if I want a handle cases like 1 0 1 0 1. So, this is a partial sequence 1 0 1 is ending

this sequence, but this 1 is also starting the next sequence. So, I will come and name this in a little while. So, this takes care of various transitions, but it does not take care of all the transitions.

So, what happens if we get a 0, so you will still be expecting 1 0 1, so you can put 0 here, if I get a 1 from here, you have going to still expect 0 1. So, I can put a 1 here, so notice that these transitions do not have the outputs, they have only the inputs, because the states are supposed to tell you what the outputs are. So, on the transitions you do not have the outputs, so on a 1 you can come back here.

So, what about if I get a 0 from here, so if I get a 0 now, it means that you got a 1 followed by a 0. So, we got one or more 1's followed by a 0 and now you got another 0. So, essentially you got 1 0 0 or 1, so many 1s and a 0 and that is not a valid, so and if I get another 0 now it is not going to start a 1 0 1 sequence. So, I could as well go back here and of fresh I have to expect 1 0 1, so I have to go back here. Now, let us see from this state, from this state there are two choices, I can get a 0 or I can get a 1.

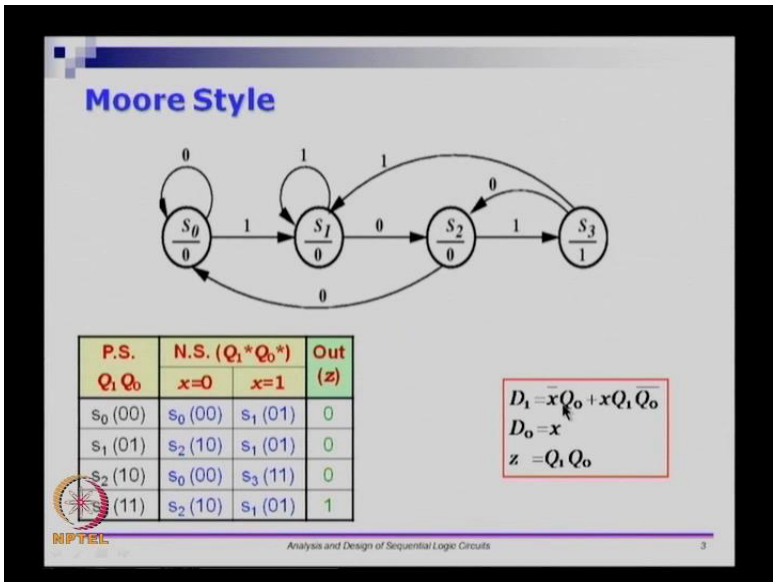
So, if I get a 0 from here, then I came in with the 1 if I get a 0 from here and if I go back to this one, what it would mean is. So, this is 1 from here I can again expect a 0 1, so if I expect a 0 1. So, I will write that now expect a 0 1, so if I get a 0 1 then again I can keep toggling between these two states and again keep producing the output 1. So, on a 0 I will go back here; however, if it is a 1, if I get a 1 from here, then I came in with the 1 and I am seeing another 1 now.

So, the only way to come into this state is the previous bit must have been a 1, if I see another 1 now, it is going to have a 1 1 sequence, this is not a valid prefix for 1 0 1. So, if I get a 1 now, then I have to expect a 0 1. So, but I have to go to this state, because I cannot stay in the same state, because this keep producing 1 as the output. So, if I get a sequence of 1s, then let us say I got 1 0 1 1 1 you will come here will produce a 1.

If you stay in the same state you will keep producing 1 as the output, but we do not one that every time I see a 1 0 1 only then the output should go to 1. So, on a 1 if go here that takes care of the fact that this is… So, both these states are expecting 0 1, but this state is different from this state, because this state produces 0 is the output, this state produces 1 as the output. So, if I have a partial 1 which is shade across two patterns is pattern like 1 0 1 0 1, you will come to this state and you will keep coming back to this state itself.

But, if you have disjoint 1's, if you have disjoint things like this, then you would come back to this state and that will give you the 1 0 1. So, this is the general description of the problem and again I can substitute s naught, s 1, s 2 and s 3 here and that will give me a state machine. So, that is what you are seeing in the next slide.

(Refer Slide Time: 14:48)



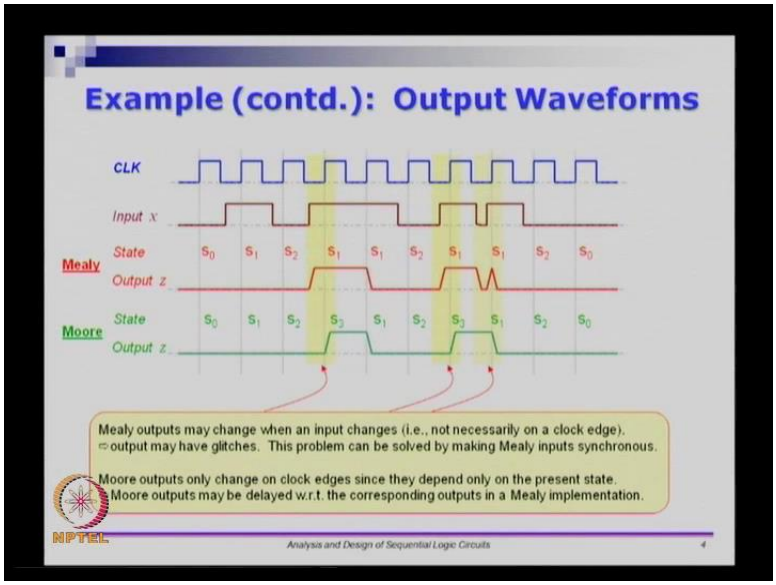
So, the Moore machine is essentially that, so it is a same thing that I did in the picture, there is a reset state here s naught, s 1, s 2, s 3 and s 3 on a 0 goes here, s 3 on a 1 goes here, s 1 and s 3 are both expect 0 1, except that one of them is producing 0 is the output and another one is producing 1 as the output at that is in important distinction. So, because states dictate the outputs, these two states have to be different, then we have done some assignment here s naught is 0 0, s 1 is 0 1 and so on and the output bit is given here, we can now go on derive the equations.

So, D 1 is x bar Q naught plus x Q 1 Q naught bar D naught is x and z is Q 1 Q naught. So, what has happened is you have 4 states, this still requires only 2 flip flops, so if you already had 4 states in the mealy machine, the Moore machine may have more than 4 states. So, there is a difference between Mealy and Moore, so Moore machines may have more state registers than the Mealy machines. Because, Moore style may require more states to begin with therefore, more state registers than a Mealy machine.

So, this is the set of equations here again I want to go back and check this and another thing that you can do is, go and change the state assignment. So, instead of calling s

naught as 0 0, s 1 as 0 1 and so on what if you put some other assignment here, go and see what the equations must be. So, now let us move to other design problems.

(Refer Slide Time: 16:20)



So, before going to the other design, this is one thing that I want to show. So, let us see this, there is a clock input that is periodic and I have an input wave form that is changing at whatever time it is changing. I want to see if there is a 1 0 1 sequence. So, let us see this mealy output, so mealy output with go from… So, it will start with s naught which is the reset state and a 1 it goes to s 1, on a 0 it goes to s 2 and on a 1 it comes back to s 1 itself and when it is in s 2 it produces the one transition, may goes from s 2 to s 1 it produces a one translation.

And similarly when it goes from s 2 to s 1 it makes a one transition and one thing you can notice is, there is a glitch in the input here. So, from one it went to 0 and came back to 1, but because it is a synchronize machine this 0 is not observed by the flip flops. Because, only on the raising edge of this clock comes in, it is going to all check the value or sample the value of input. So, this state does not change; however, what has happened is, because of this glitch here the output may see a glitch.

So, in a mealy machine output can be directly controlled by the input, so a momentary change in input can actually change the output. In this case, the state registers are ok; however, the output has a glitch. So, it should have been at 1 and then it should go to a clean 0 and stay at 0, it can momentarily go to 1 and come back, this can happen in a

mealy machine. However, in a Moore machine, so we had 4 states s naught s 1, s 2 on s 3 the produce 1 and this will stay up clean 1.

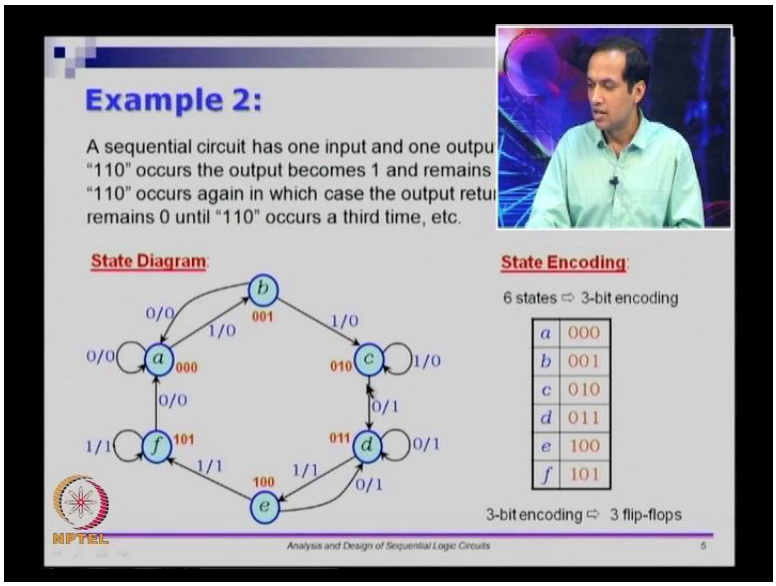
Because, the output is depended only on the state and the state registers are kept for one whole cycle, they are not going to change the value. So, on the rising edge the state will change and it will remain the same for one whole cycle. So, you will see that the outputs coming from a Moore machine are stable for one full cycle as compare to the mealy machine, mealy machine may have multiple transitions even within one cycle.

So, in summary mealy outputs can change when an input changes, so it need not necessarily happen on a clock edge, and therefore outputs may have glitches. So; however, if you make all the inputs themselves synchronous to the clock, then you will not have a problem. So, if the input is asynchronous for example, this change from 1 to 0 to 1 was not synchronized to the clock, therefore, you had this problem.

But, if the inputs were all synchronized, then this case will not happen or if you do not have any control over the input, from the external world you may be getting the input, you do not have control over whether it is synchronized or not, then in that case you employ a Moore machine. So, Moore machine will have a study output, because the state dictates the current output. So, and you going to keep the state in the same value for one whole cycle.

One thing that can happen is the output may get delayed by one cycle. So, here you can see that the third clock cycle, just as you are finishing in the third clock cycle itself the output went to 1, where as here only in the fourth clock cycle the output went to 1. So, that may be a delay in producing the output in a mealy in a Moore machine. However, Moore machine will have stable outputs, so just remember this.

(Refer Slide Time: 19:54)



So, let us solve other problems now, so let us look at another problem description, a sequential circuit has one input and one output, when the input sequence 1 1 0 occurs, the output should become 1 and it should remain 1 until the sequence 1 1 0 occurs again. So, this is the slightly more completed problem, so the previous problem you could have solved with a shift register, this one is not. So, if you get a 1 1 0 you produce a 1 and then it should remain at 1 till the next 1 1 0 occurs again, at which point it goes to 0 and you keep going back.

So, if I see the first 1 1 0, it should go to 1, the second 1 1 0 should take it to 0, the third 1 1 0 should take it to a 1 and so on. So, this is what I want as a description. Let us see how to design a circuit like this, to do this we need to first of all do a state machine. Let us see the state diagram, I start with the state a, let us assume that there is a reset state. So, on seeing a 0 I will remain in this state itself, because I am supposed to be looking at 1 1 0.

So, on a 1 I go to state b and I will produce an output 0 and b on a 0 can come back to a because… So, from here you are expecting a 1 0 sequence, so on is if I get a 0 I can go back and expect a 1 1 0. So, I can go back here and if I get a 1 I move to the new state called c, in c if I get a 0 now it means you are actually expecting a 0, so getting a 0 is good. So, I will go to the state d, but I will produce an output of 1, if I get a 1; however, I will stay in the same state itself.

Because, you got 1 1 and if I keep that in a serious of 1s at some point it can switch to a 0, so I can stay in this state itself. So, this state means I got 1 1 and I am expecting a 0, so you can if I get another 1 it is only shifted by one more position. So, it is still got 1 1 you are still expecting a 0, so you can stay back here. Now, you are produce an output of one for this transition; however, from d, d is now equivalent to a and some sense, because from here you expecting the second set of 1 1 0 two come through.

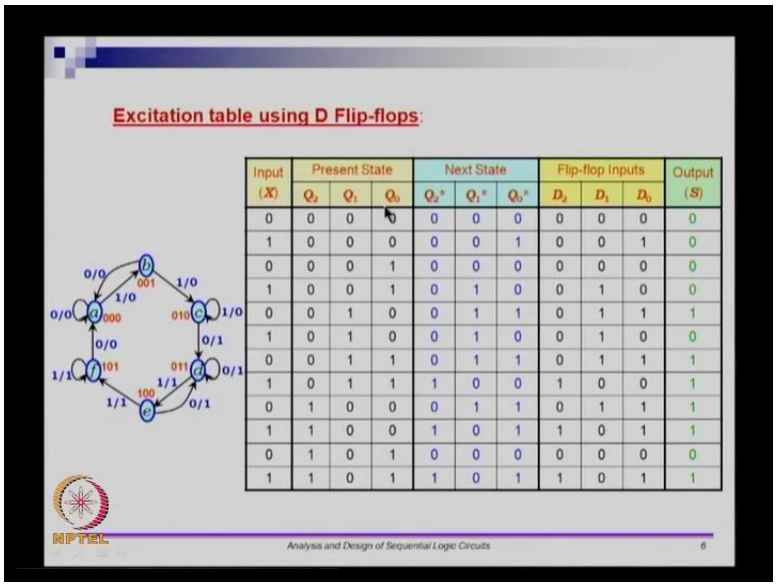
So, in the second set of 1 1 0 comes in… So, d will be similar to a on 0 you look back, but the output will be a 1, because the problem description says, the output should remain 1 until the second 1 1 0 occurs. So, we have that; however, on a 1 you go to state e, so e on a 0 you can go back to d itself, because it is again expecting a 1 1 1 0 sequence, on a 1 you go to state f and we now have to stitch f and a together.

So, f from a 0 can go to a and now the output can go to 0, so this output went to 0, because you got the first 1 1 0 will end up in d, the second 1 1 0 will take you to a. But, that point you have to make the output 0; however, if we get a 1 from there, you keep looking back at f itself. So, a, b and c are equivalent to d, e and f except that the outgoing transitions from a, b and c are all 0s and the transition from d, e, f are all 1s.

And there is a transition from c to d which produces output 1 and from here actually produces an output 0. So, it is you can see the symmetry between the top half of this state machine versus the bottom half of the state machine. Now, I can do state encoding there are 6 states, 6 states will require 3 flip flops. So, in general n states will require logarithm n base 2 flip flops and the sealing off that the integer which is larger than log n base 2.

So, 6 states log of 6 is 2 point something and the sealing of that is 3 bits, so I will need 3 bits to encode that and let say I have arbitrarily assigned it has 0 0 0, 0 0 1, 0 1 0, and so on. So, I have given some increasing sequence for all of these.

(Refer Slide Time: 24:11)

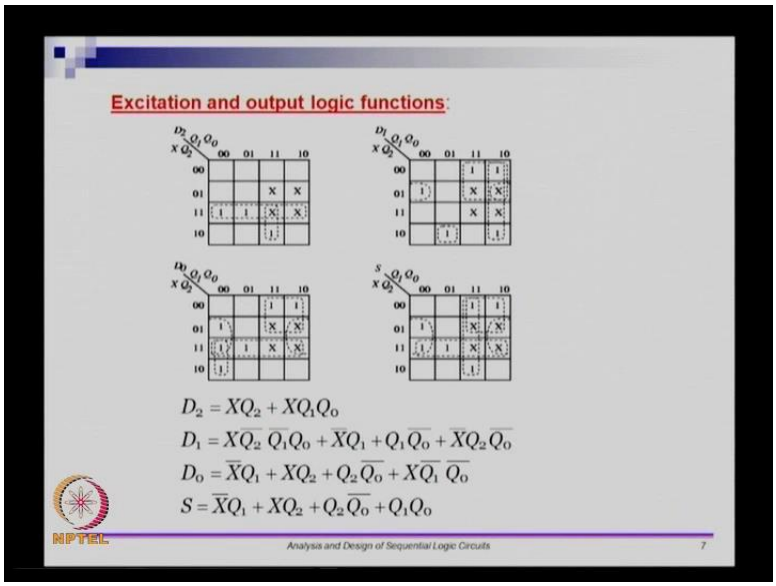


Input (X)	Present State			Next State			Flip-flop Inputs			Output (S)
	Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	D_2	D_1	D_0	
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	1	0	0
0	0	1	0	0	1	1	0	1	1	1
1	0	1	0	0	1	0	0	1	0	0
0	0	1	1	0	1	1	0	1	1	1
1	0	1	1	1	0	0	1	0	0	1
0	1	0	0	0	1	1	0	1	1	1
1	1	0	0	1	0	1	1	0	1	1
0	1	0	1	0	0	0	0	0	0	0
1	1	0	1	1	0	1	1	0	1	1

Now, if I want to get a circuit out of this then you need do something more. So, you start with 3 flip flops Q 2, Q 1, Q naught which indicate the present state and we have an input, you look at 4 vectors 4, 4 bits and there are 16 combinations of these. So, you enumerate all the 16 combinations, for each combination you decide what the next state must be. So, from the state diagram you can see what the next state is and what the output is, you can do this column and these three columns mark blue and this column mark green, you can get that from the state diagram.

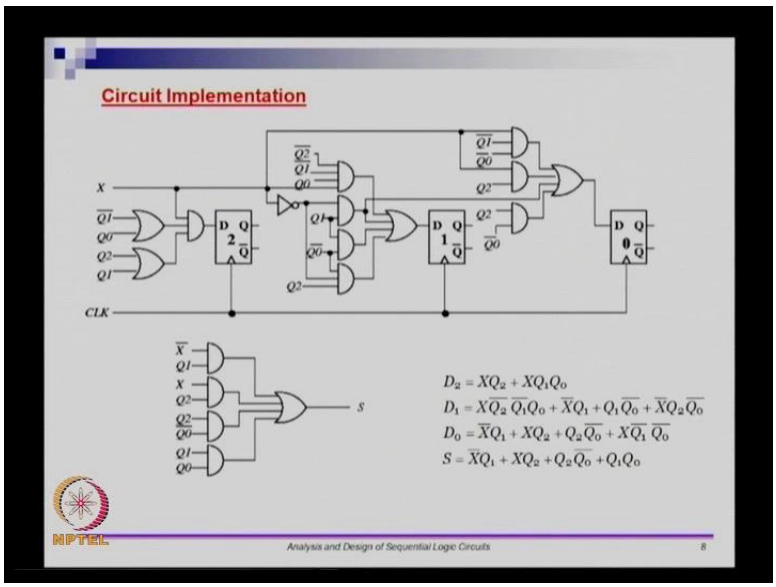
Now, if I say use the D flip flops, then I taught you how to infer the D inputs from the present state and next state. So, whatever you want as the next state, you place them in the D inputs that is how D flip flop works. So, if I want 1 0 1 for the next state I have to place 1 0 1 in the D input in the current state, only then it will become 1 0 1 in the next state. So, now, I can get rid of this next state I do not need that I will go and describe D 2, D 1 and D naught and s in terms of x Q 2, Q 1 and Q naught that is takes care of the circuit. So, I will suggest that you go and do that yourself.

(Refer Slide Time: 25:24)



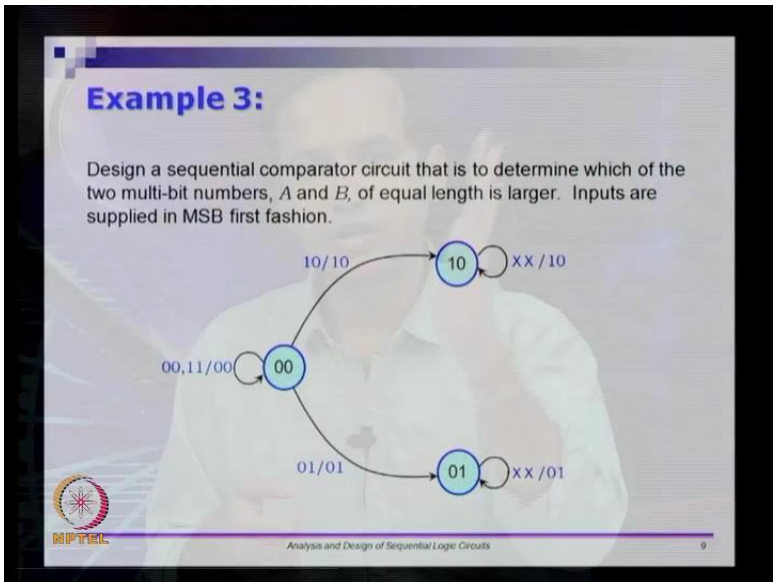
The equations are shown here, so D 2, D 1 and D naught have some equations, s has an equation in terms of Q 2, Q 1, Q naught and x and you can draw circuit for that.

(Refer Slide Time: 25:35)



So, this is a circuit that can be use for this, so I suggest that you have actually try and do this yourself and convince yourself that the circuit is current.

(Refer Slide Time: 25:45)



So, now let move to the next example, design a sequential compare at a circuit that is to determine which of the two multi bit numbers A and B of equal length is larger, inputs are supplied in MSB first fashion. So, let us first understand what the problem is, you are ask to design a comparative, comparative will take two numbers is inputs and compare the numbers to with each other and say whether A is greater than B or A is less than B or it could be that A is equal to B, in fact.

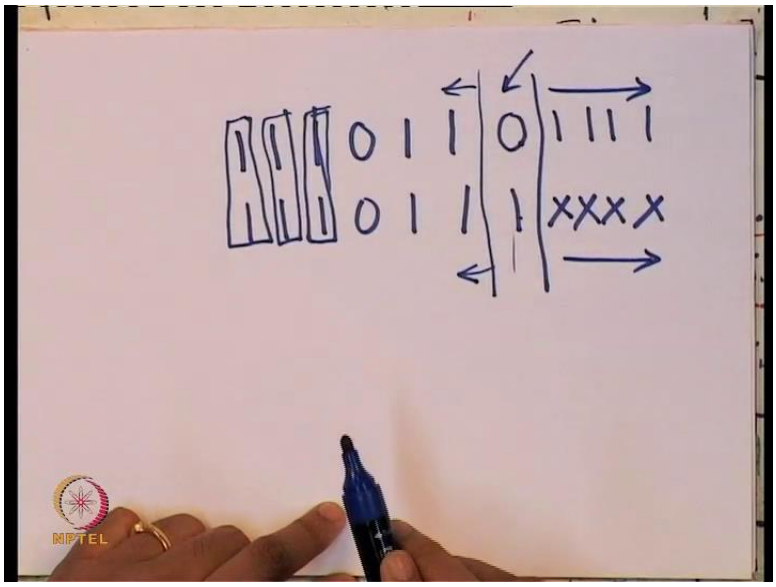
So, in this case… So, this is something that we do in real world also, let if I give you two numbers how would you compare, if one number has more digits than the other, then automatically you will say that the one with the more digits is larger than the number with the fewer digits. But, what if both the numbers have the same number of digits, I give a 6 digit number, I give you another 6 digit number how would you compare these numbers.

You will start from the left side, so you will start from the left side, if the leftmost digit is the same, you cannot resolve whether A is greater than B yet will go to the next digit, you will again compare those two if they are still the same, you will go to the next digit and so on. So, the same thing if you are given bits if I give you 2 n bit numbers and I ask you to compare, you can actually do a state machine for that let see how that works.

So, let say I have state called 0 0 and I am going to get two inputs, one from A and one from B, but starting from the MSB first I am getting a most significant bit first and I

know that A and B are equal length. So, when I get A and B of equal length, so if both of them in the most significant position have a 0, it means the very first bit is not enough for me I need one more the I probably need to go and look at the bit which is in the second from the left or if I have both of them as 1 I still cannot distribution say A is greater than B. So, either case A and B I do not have a conclusion, so I will output 0 0 and remain in the same state. So, let me do this on a piece of paper.

(Refer Slide Time: 28:01)



So, if I have an input like this and if I give you another input which is also like this, you will first compare these two numbers, you cannot distinguish between them, you go to the next digit, you compare these two bits, again they are the same, then you compare these two bits and so on. And if you exhaust all the 6 digits, you will stay the same state 0 0 itself and produce 0 0.

Instead, let say at this point you have 0 1, you have a 7 bit number up till here these two numbers are the same, but now you get on a you get a 0 and b you get a 1, if that happens… So, A you got a 0 and B you got a 1, if that happens then you go to the state called 0 1, in this state the output… So, you also make an output 0 1 with says A is… So, which one is greater. So, if this is 1, then A is greater of the second output is 1 B is greater, but if both are 0 then both are equivalence that the meaning here.

So, if it is a 0 1 b is greater and the moment I decide the B is greater, all the bits there going come later that is not matter. So, even I if have any bits here after this and any bit

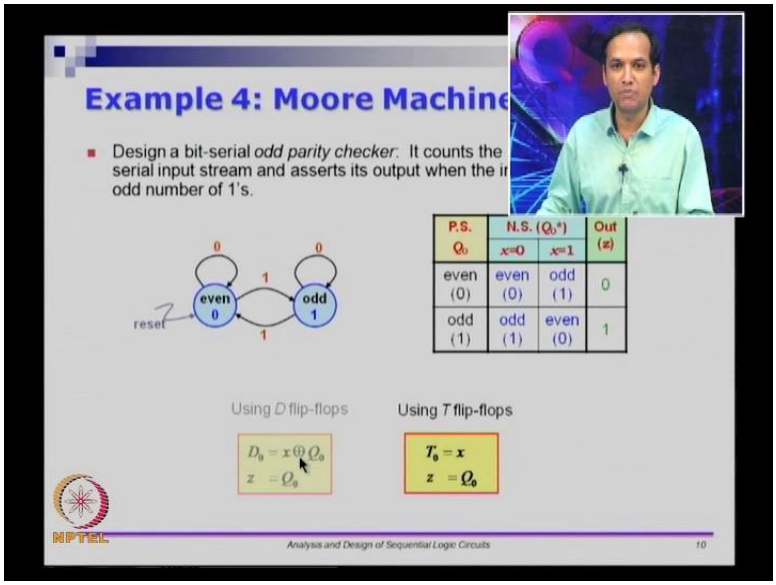
here after this it does not matter, it could be any sequence of bits after this and after this, this bit this position tells me that A is less than B. So, that is what is explaining the loop here.

Once you have decided that A is actually less than B, then that is not going to change if you get more bits which have lesser weight any way. So, you will just stay in that, so the inputs become a do not care, you will always produce the output 0 1. And similarly, if you have decided A is greater than B there is one position where you discriminated A and B and A is, in fact greater than B to moment you have decided that any number of bits that come in they do not matter anymore, so you can just stay in that is state itself.

So, this is the very simple comparator circuit, it compares one bit a time starting from the left most side, if you started from the right most side this is the very, very hard problem. Because, I see a 1 and 0 on the right most on A and B I cannot say that A is greater than B after then go and look at the next bit, because that may actually change the weights and so on.

So, if I start with the MSB side, then this a very straight forward state machine, it is a very simple state machine and this is a very nice example, because you get 1 bit as input at a time for each of the inputs A and B and with that you are able to say whether A is greater than B or not.

(Refer Slide Time: 30:50)



Then, let us design a few other circuits, design a bit serial odd parity checker. So, it counts the number of 1s in a bit serial input stream and it asserts the output when the input stream contains odd number of 1s. So, if there is odd number of 1s produce a 1, if there is even number of 1s produce a 0. So, that is what an odd parity checker is, so parity stands for the number of 1s and if it is a odd parity, the number of 1s is odd.

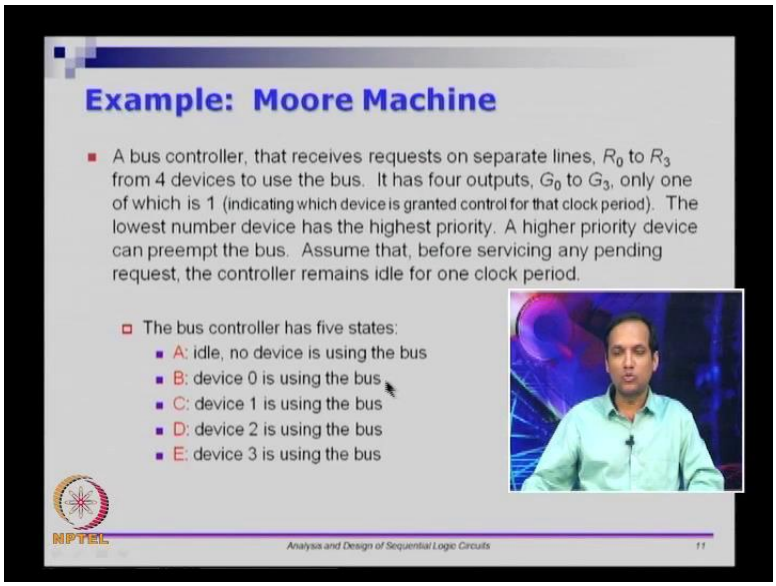
So, that is a very simple state machine actually, so you start with you… So, if you do not have any inputs at all then there are even number of 1s, because there is 0 1s. So, that is even number of 1s, you start from there, if you keep getting 0 it is not changing the number of 1s. So, you stay in that state itself, the moment you see a 1 you have seen odd number of 1s and if you keep seeing 0 you will just stay in even number of 1s itself.

In odd number of 1s if you see more 0 it is not changing the parity till state there itself. But, the moment if you another 1, then you go to even number of 1s, so this even and odd even though the states are marks 0 and 1 they are actually looking at even number of 1s and odd number of 1s. So, this state is for even number of 1s, this state is called is s 0 and this is for odd number of 1s, the state is called s 1. So, this blue number 0 and 1 or state s 0 and state s 1 respectively, this is not even number of 0s and odd number of 1s, so do not mistake them.

And you can now go and do a state machine for that, so I assign 0 to this state 1 to this state and I can get a state table out of it and the equations for that also. So, D naught happens to be x XOR Q naught and z which is the output that is going to come from it is Q naught itself. So, there is one only single flip flop, the single flip flop can say whether the, if the flip flop has 0 it means it is odd it is even parity, so for or if the number of 1s is even, where if it is 1 then the number of 1s is odd so far.

So, it is just a single bit state machine, if you want to do T flip flop, the T flip flop is slightly simpler. So, t naught is just x itself and z is Q naught itself. So, I suggest that you go and try this and convince yourself that using the T flip flop the logic become simpler. So, this is an example where using a different flip flop seems to actually reduce the gate count, here you would an XOR gate, hear you do not need anything at all you actually use one flip flop in both the cases, but you do not have any gates here, you have some gate here. So, the kind of flip flop you use can actually control the number of inputs that you need.

(Refer Slide Time: 33:33)



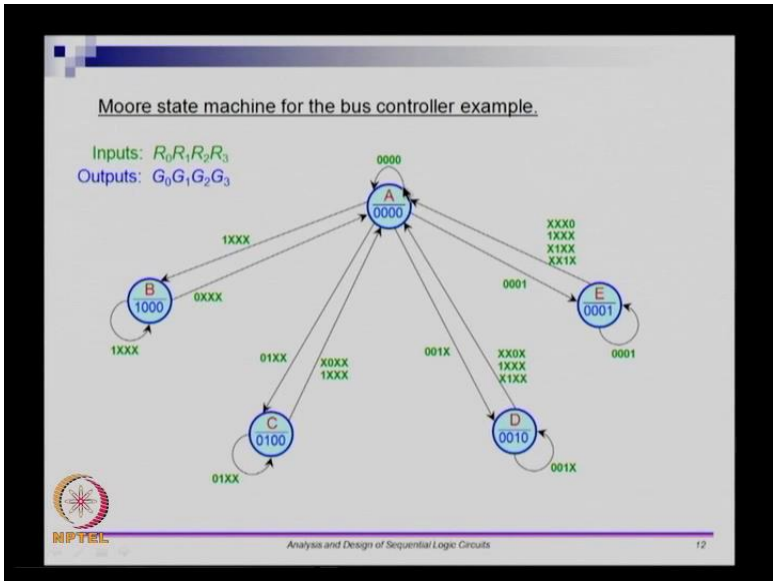
So, as a final example I want to give this, there is a bus controller that receives request from R naught to R 3 these are 4 devices. And there are 4 outputs G naught to G 3 and only one of which should be 1 indicating which devices granted control. So, imagine there are 4 devices contenting may be there are 4 computers trying to print on to the same printer. So, one could be let say the boss of the company, then 1 computer could be a manager, the 3rd computer could be an assistant manager and the 4th computer could be a regular employee.

So, all 4 of them if there want to print to a printer, so you want the boss of the company to have higher priority over the other employees of the company. So, you want to see who should get grant, there are four requests that are coming in you want to grant permission to only one of them. So, what we want is there are 4 outputs G naught to G 3 and only one of them must be 1, because you do not want a printer to print multiple jobs from different users at the same time.

So, you want only one of them to be a 1, the lowest number device has the highest priority. So, if R naught makes a request then G naught must turn on, if R 1 makes a request you go and look whether R naught as made a request or not if R naught did not make a request then you make G 1 to be on and so on. So, that is what this one is and assume that the before servicing any pending request, the controller will remain idle for one clock period.

So, this is a very wordy problem, let see how to design this, so since we have an idle state, we need to have at least one state with says idle. So, if A is idle, no devices using the bus, then I will use the state B to say that device 0 is using the bus, state C for device one is using the bus, D for 2 is using the bus and E for 3 is using the bus. So, the bus here means who is getting access to the printer let say.

(Refer Slide Time: 35:43)



Now, let see how this state machine itself would look like, so this A is the idle state and on A you do not give grant to anyone. So, you produce 0 0 0 as the output, as long as nobody makes any request, which means if all the inputs are 0 then you stay in A itself. However, the moment are not makes a request which means R naught is 1, no matter what the other employees have this the boss, the left most is the boss, if the boss makes a request to the printer even if other employees have request, you want to give access to the boss.

So, from A you go to the state B, where the access is granted to G naught, so G naught becomes 1, G 1, G 2, G 3 stay at 0, as long as the boss has the request for the printer, the boss will get access to it. And only when the boss releases the request, once the boss releases the request that is made by making R naught 0, no matter what the other inputs are we set we will remain idle for one clock cycle. So, we come back to the idle state itself, now let see this transition.

So, let say that the boss is not making a request, the manager has a request and it is also possible that the assistant manager and the clerk also want access to the printer. So, as long as the boss is not making a request and the manager has a request, no matter what the assistant manager and the clerk has we want to give access to the manager. So, the manager gets a 1 the boss is not getting access, so that is a 0 and also the assistant manager and the clerk do not get access.

And this will remain at this state itself, as long as the boss is not making the request and as long as the manager wants the access to the printer. So, that is why you have 0 1 x x, the moment the boss makes a request, let say the printer is being used by the manager, the moment boss makes a request that is from 1 x x x, you want to go to the idle state and the boss should be serviced or the manager is done with printing the manager removes 0.

So, removes the request which means he makes R 1 0, in that case also you want to go to the idle state. So, the other two states are equivalent, so let us look at the last state, so you come to the state only when you know that the boss the manager assistant manager none of the made a request and the clerk has a request for the printer, only in that case you come and you will stay in that state as long as you are the only one, the clerk is the only one took make the request.

The moment the assistant manager makes a request or the manager makes a request or the boss makes the request, you will go back to the idle state and you will have to service one of these three. In fact, if none of one makes the request, let say the E is also done, the clerk is also done with the work, then the clerk makes the request 0 in that case it goes to the idle state at that point will remain in the idle state itself till somebody makes the request.

So, this is a very simple state machine, but is a very useful state machine, because it shows you the notion of priority, the notion of priority can be handle in a very straight forward manner here by looking at what are the inputs and what are the outputs. So, what I will suggest you to do is, you go and design a circuit for doing this. So, there are 5 states, these 5 states will require at least 3 flip flops, these 3 flip flops will take inputs which are R naught, R 1, R 2 and R 3 and the circuit should produce outputs G naught, G 1, G 2, G 3.

So, go and think about how you will design such a circuit and in this case the outputs are completely dependent only on the state itself. So, if you are in state A nobody gets grand, if you are in state D the assistant manager gets grand and so on. So, this is the more machine, so go and check whether a output logic that you get is actually a function only of the state registers and not a function of the input directly, go and verify that yourself.

So, what we have done in this module is, we have taken several design examples, we started with the problem description and I showed you how to do is state machine, how to mark the state transitions and in some cases we also saw how to show the actual circuit itself. So, the general process is for synthesis of a state machine, you start from the English description, you go and draw state machine for that you do what is called a state assignment you put…

So, you find out the number of state registers, you do is state assignments with says, this state get 0 0 as the label, this state gets 1 0 as the label and so on. And once you do that you pick up the flip flop that you want your derive the excitation tables for that use the excitation table, get the excitation for your circuit and draw the circuit. So, this is the 5 step process, so this is the slide that I showed you in the last video I suggest that you go back and look at the slid once more, this is the sequence for synthesis.

So, there are 5 steps that you need to do for the synthesis, so this brings me to the end of this module. In the next module we will do a few thinks related to the state machine itself, there are two more modules where I am going to talk about state machines in a little more detail. So, at this point I suggest that you go and do all these little piece of work, before you go to the next video I suggest that you go on look at the contents of these two videos and ensure that you have done all the little exercises. And finally, workout these exercises convince yourself that you understand the ideas and then go to module 31. So, this brings me to the end of this module and I will see you in a little while the next module.

Thank you.