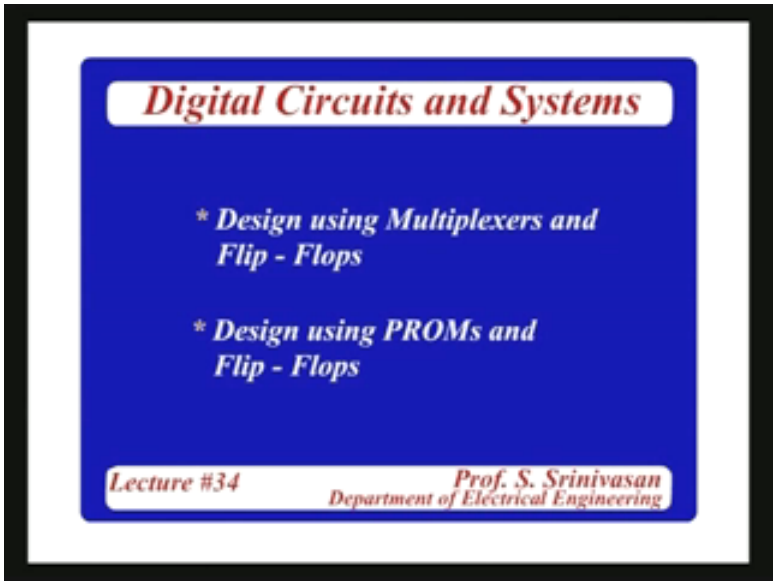
Digital Circuits and Systems

Prof. S. Srinivasan
Department of Electrical Engineering
Indian Institute of Technology Madras

Lecture - 34

MSI and LSI based Implementation of Sequential Circuits

(Refer Slide Time: 00:01:33)



We have been discussing the MSI and LSI circuits and implementing combinational logic using these circuits. We talked about multiplexers, decoders, programmable logic devices, prom, Programmable Logic Array PLA and PAL Programmable Array Logic. But as you know that most circuits are sequential in nature and even the combinational logic we discussed are useful in sharing the sequential circuits through the various states as per the state diagram state graph.

So basically when you design a circuit or a sub system or a system or whatever you want to call it, whatever level you want to deal with, it is a sequential circuit. When you say sequential it doesn't exclude combinational but when you say combinational it excludes sequential. That's why when you say sequential you do not have to say combinational and sequential, I don't have to say that. The sequential circuit will have to have a combinational logic which will be the logic for determining the next states and the outputs.

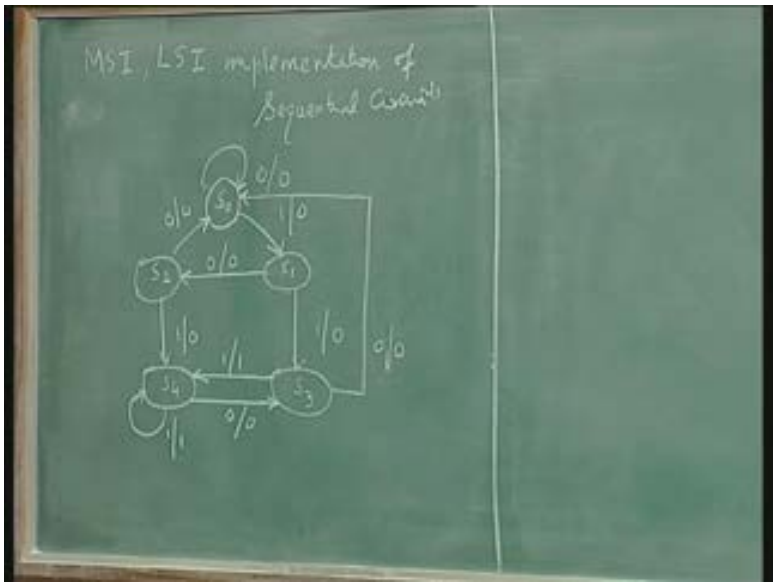
Therefore, with that in my mind we will now look at the implementation of sequential circuits using the same MSI LSI logic. I will emphasize this point once more (Refer Slide Time: 3:44) that when I say sequential circuit it has a combinational and a sequential component so the combinational part will use all these MSI LSI we discussed. Of course

we can also do sequential for example Programmable Array Logic which had its own flip-flops, this is what we discussed last time and those can be utilized for the sequential part of the circuit, steering logic, that is to implement the states. So it is an integrated effort.

We will also see later on how some of the sequential MSI like; shift register can also be used for implementation of circuits, state graphs, we will see that after this. Thus, right now we will use the same multiplex decoder etc and take a state graph which you know how to implement otherwise using gates, you will take such a circuit and see how to implement did using some of those MSI LSI you learnt in the last few lectures. So, for that we will take the example of a state graph. You know what this interpretation of graph means? One input one output, and these are the five states and is this a Melee machine or Moore machine? It is a Melee machine.

This is a machine with five states S_0 to S_4, we will have to make sure which of these states have two arrows going out, one is for 0 input and the other is for one input so S_0 with 0 goes back into this, with one it goes here, S_1 with 0 goes here, and with 1 it goes here, S_2 it is 0 it goes here (Refer Slide Time: 7:23), 1 it goes here, S_3 0 goes back, 1 it goes here, S_4 0 goes here and 1 goes back here to the same state. Therefore as I said it is an arbitrary state graph. It is a sample state diagram.

(Refer Slide Time: 7:45)

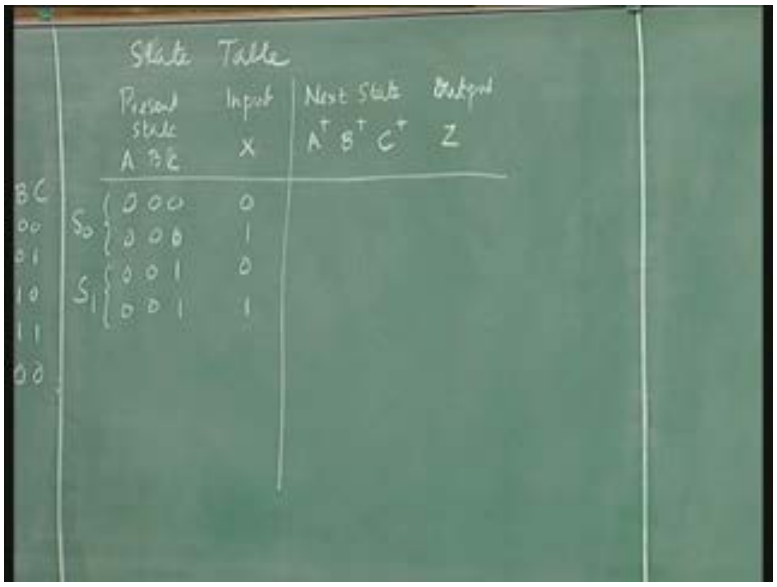


I want to use this diagram to illustrate the point of implement this using multiplexers, proms, pals and PLAs if necessary which will be used as steering logic and in case of PLAs the flip-flops will also be used in the same IC. Again we do not know the system, it is an arbitrary graph, whereas we assume this as an input and output so we will call the input X and output z, X is here so this is equal to the output and there are five states so we will call the states S_0 S_1 S_2 S_3 S_4 so as arbitrarily for S_0 it is 000, S_1 it is 001, S_2 it is 010, S_3 it is 011 and for S_4 it is 100, these are the assignment rules. We should also have

names for the state variables, so we will call the three state variables as ABC (Refer Slide Time: 9:06), there are five states so ABC goes from a 000 through 100.

Let us use a D flip-flop to make the implementation simple in the sense we do not have to go through the state graph state table and then a transition table. The state table and transition table are same for a D flip-flop implementation; you know that. So we will take this and write the state table which is also the transition table for this diagram. Therefore for the present state what are the various things you will write in a state table? The present state input x, the present state variables are ABC, input value is X and the next state is ABC. Now, to distinguish between this A B C and this A B C we will put a plus sign here meaning the values of ABC have to be clockwise and then the output Z.

(Refer Slide Time: 11:14)

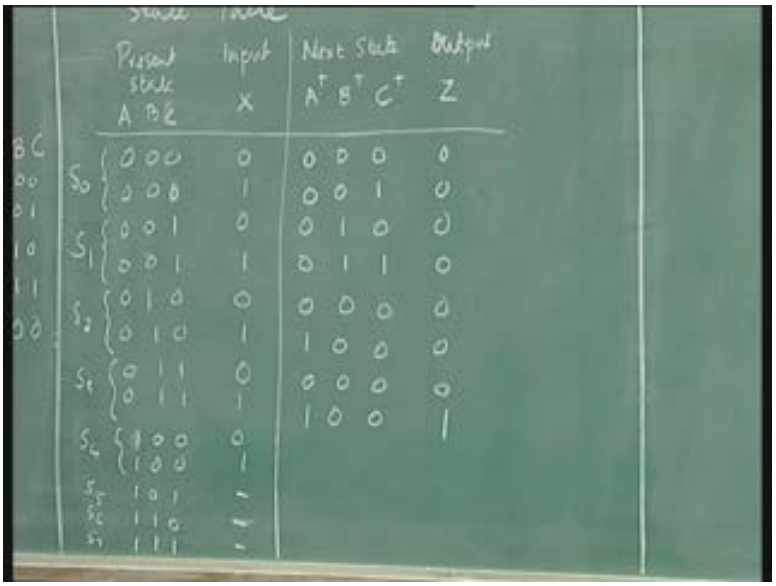


Now from S_0 X can be 0 or 1, then from S_1 it can be 0 or 1, at S_2 again it can be 0 or 1, S_3 can be 0 or 1, S_4 S_5 S_6 S_7 may not exist, now this is 1 0 1 1 1 0 1 1 1. We do not have these sates so there is no need for considering the input X is 0 input is 1 for these states.

Can somebody quickly tell me so that we will not make mistakes?

Where does S_0 go when X is 0? It is to the same state 0 0 0 and output is 0. Next, 0 0 1 0 (Refer Slide Time: 12:40) then 0 1 0 0 0 1 1 0 that is when the cycle is in S_1 the input is 1, it goes to S_3 which is 0 1 1, output 0. From S_2 it is 0 0 0 the output 0 or 1 0 0 with output 0, then from S_3 if it is 0 it goes back to reset state S_0, so 1 goes to S_4 with an output of 1.

(Refer Slide Time: 13:50)



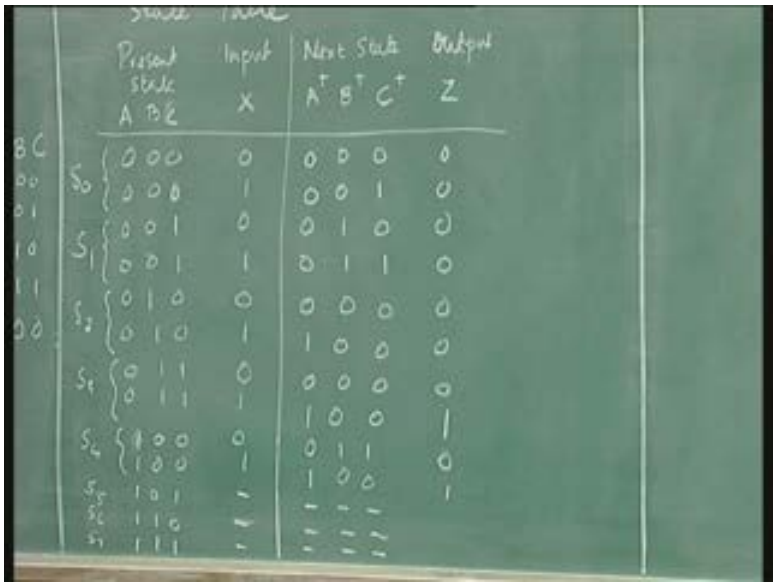
In S_4 it is going back to 0 1 1, output of 0 and it remains in that state and here it is an output of 1. The state table will also act as a transition table because we are going to use the D flip-flops. Now these states are not going to appear so the next states are also not defined 'don't cares' but the outputs are always 0. Whenever the states are not defined we will make it 0. This is also possible in advance, in the elementary level of treatment. Thus, for a slightly more advanced level we can say, when the state transition has to happen there also….. so the don't care state occurs between two states whose output is 0 or between two states whose outputs are 1, by making it a 0 we are causing a glitch.

Do you understand what I am saying?

Suppose I had a state and the next edition a don't care and then another state both these states define the state output as 1 and the input in the don't care state the output is……because we say all don't cares may be the output is 0. Let me now force an output of 1 to become 0 and then back to 1 if I can retain that 1 as 1 then I will avoid a glitch. That is a special technique. As I said we are keeping this discussion elementary because this is an introductory course, I would rather teach you concepts and some other practical things.

Of course I have been mentioning here and there like optimizing the state graph, assignment rules, reductional state graph and so on, assigning output states properly. These are the topics which I thought you should know, that which exists, it is not that people do not know these things but some of those things which you can learn later once you familiar with the basic things, these are the things which will come in practical design.

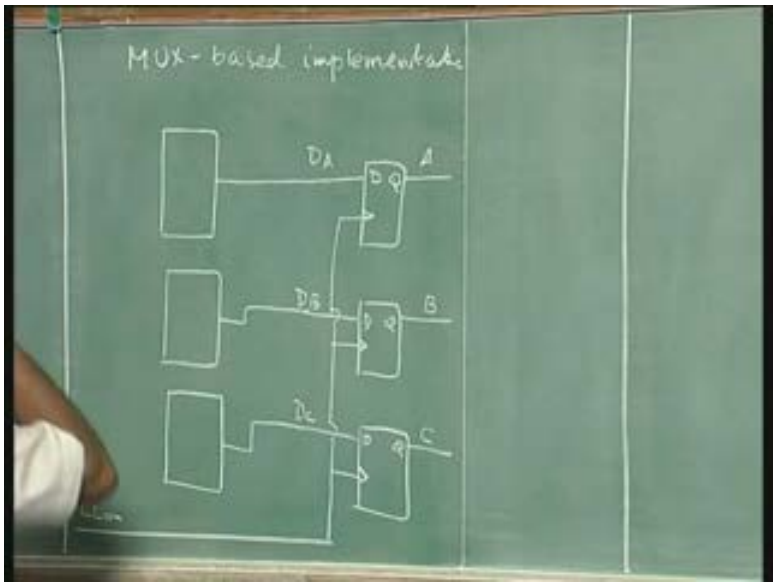
(Refer Slide Time: 14:52)



Anyway coming back to this input all output we will put all 0s to start with. As of now the standard rule we have been following that all outputs of undefined states we make them as 0s. So the question is now I do not want to go through the Karnaugh map, Karnaugh map minimization of….. these are the A power plus B power plus C power plus inputs to D flip-flops. In the case of D flip-flops implementation this becomes the D flip-flop input for A, D flip-flop input for B, D flip-flop input for C and these are the outputs so we have to draw the Karnaugh map for each one them and simplify them and implement by using gates. This is the procedure we are familiar with.

Now I am going to introduce multiplexers. In place of the combinational logic which we have earlier done using gates let us try to do a multiplexer based implementation. I still need D flip-flops because multiplexers can only give me the steering logic, multiplexers are combinational logic blocks, remember that, multiplexers are combinational building blocks and combinational building blocks are only a logic which controls the state variations, the state transitions of the flip-flops, so that I am going to replace by multiplexers but it does not mean that I am going to eliminate the D flip-flops.

(Refer Slide Time: 17:25)



The circuit is going to look like this (Refer Slide Time: 17:50) a Mux based implementation. We are going to have three flip-flops; D flip-flops and these three flip-flops are going to give the state variables A B C, there is no change in that, that is what we have been doing, we have been following this rule will continue with that and then these will be driven by……[18:41]…..

My aim is to reduce the number of ICs used in combinational logic, it has got a variety of gates of different types can I use multiplexers and get this, that is my question. So the DA which is the input to the A flip-flop DB which is the input to B flip-flops and DC to the C flip-flop will now come from multiplexers instead of gates that is earlier we have done the same problem not the same state graph but we might have similar to this, we know how to get DA DB DC by drawing Karnaugh maps and simplifying them, by drawing a gate implementation.

Now I am going to use multiplexers to get DA DB DC the output to the multiplexers should be connected to DA, the output of another multiplexer to DB, output of another multiplexer to dc that means we need three multiplexers, so it is a very simple design. This is the output of the first multiplexer (Refer Slide Time: 19:55), this is the output to the second multiplexer, and this is the output to the third multiplexer. For each of this it is a 8 to 1, now I will tell you why it is 8 to 1. It is 8 to 1 because there are eight different states in which the circuit exists, of course only five we use and the other three are don't cares. Now when you are in the present state we want know what the next state is. This is all in the steering logic.

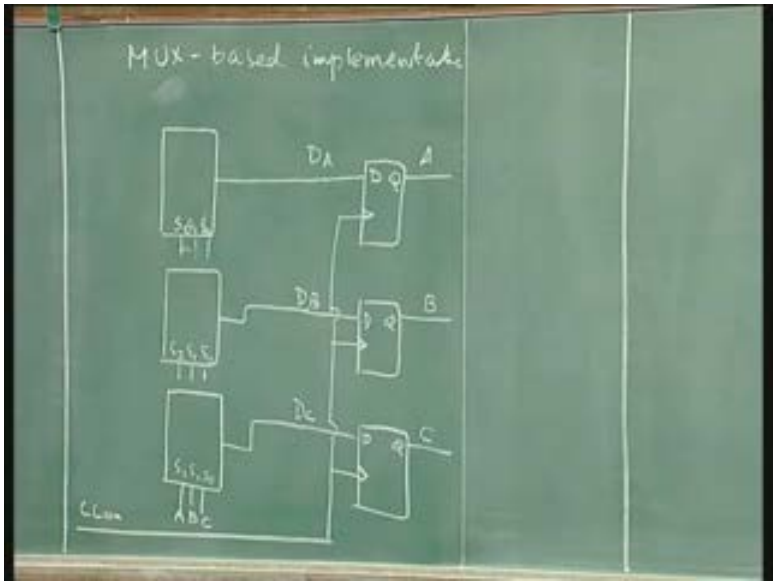
What is the purpose of the steering logic?

Given the conditions at the present state and the input what should be the next state, the next state is nothing but the DA DB DC values for the next state. That means I give the

present state values along with inputs and find out what should be the next state. Therefore, eight different present states are possible and for each of the present state what should be the next state. So the multiplexer is a selector mechanism. I can have the selected signals at the present state and then the corresponding input will be connected to the output.

Suppose you have a 4 to 1 mux two selector and four inputs so the selectors are 0 0 0, first input will be connected to the output, the selectors are 0 1 second one will be connected so 1 1 and the last one will be connected. We will you use the same argument here and say, I am going to give the values of the present state the a b c which says available here these are the selector values S_2 S_1 S_0 S_2 S_1 S_0 S_2 S_1 S_0, we will connect this A B C so it is going to be very messy if I draw that so we can just assume that this is A B C which are coming from here (Refer Slide Time: 21:50) because these are the present state values which is fed back.

(Refer Slide Time: 22:00)



You remember that over all block diagram, all that sequential circuit implementation? There we have this steering logic, the output which is the next state variable which is kept in the flip-flop set back as the present state along with the input then you get the next state. Imagine that, visualize that block diagram so A B C has to go back into the combinational logic and that is what this A B C is.

Now all I have to do is to look at the state table, and corresponding to the present state 0 0 0 what should be DA, DB and DC, if X is equal to 0 then X is equal to 1, if I can put that information in the input of the multiplexer A B C being selected here now 0 0 0 that will be connected to DA and so we will go to the next state which requires this.

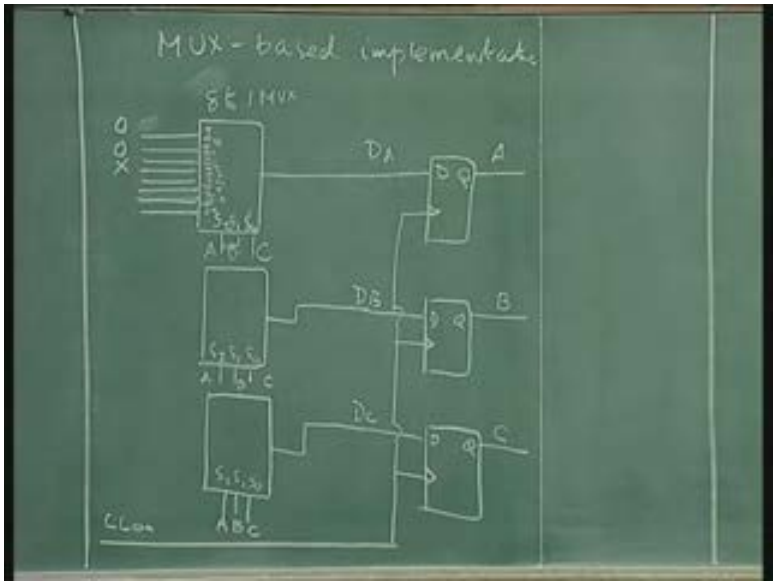
Now as an example we will take the first row, so present state is 0 0 0 X is 0 then next state of the flip-flop A should be 0. If the present state is 0 0 0 and input X is 1 then also

the next state of the flip-flop is 0. So the next state of the flip-flop A should be 0 when the present state of the circuit is 0 0 0. when the circuit is in S_0 state that is 0 0 0 state the next value for A is 0 irrespective of the value of X so because of that if I can put 0 here at the first input, so how many inputs are there, there are eight multiplexers so there will be eight so we will call this I_0 to I_7, this is I_1 I_2 I_3 I_4 I_5 I_6 I_7 and that is why it is an 8 to 1 multiplexer that is why each of these is an 8 to 1 multiplexer.

There are eight possibilities and for each of these possibilities you select the next value. Now let us finish flip-flop A before going to flip-flop B, we can exhaust this column (Refer Slide Time: 24:45) then we will go this column then to this. We can do it either way, doesn't matter. I can draw parallely A B C and then go to next state doesn't matter. It is more convenient to look through this and then this and then this. So if the present state is 0 0 1 S_1 state again irrespective of the value of X the next state is 0.

Next state is 0 irrespective of the value of X if the present state of the flip-flops are 0 0 1 so again put 0 here. Now you know how to do this? It is a simple game they are playing, that's all. So, I am giving you a procedure where implementation is the minimum thing. So all you have to do is to concentrate on the state graph, again I am emphasizing on this point. As an engineer you are responsible for a proper state graph, most efficient state graph, after the procedure, whether it is manual or software and then there is hardware which can map and so on.

(Refer Slide Time: 26:50)



What I am saying is for a design concept for a designer, a company hires you for designing digital systems, the maximum contribution comes in the, interpretation is a problem into a state graph, rest is a routine procedure I am saying it should not be done, it should be done, we should build it and test and if it doesn't work you have to change it and all that type of thing. So go again, S_2 it follows X so if X is 0 the next state is 0; if X is 1 the next state is 1. Then if the current state is S_2 the next state is of flip-flop A is 0 if

X is 0, 1 if X is 1 that means I need to connect X to the third input, this X is not a don't care, this X input (Refer Slide Time: 26:54).

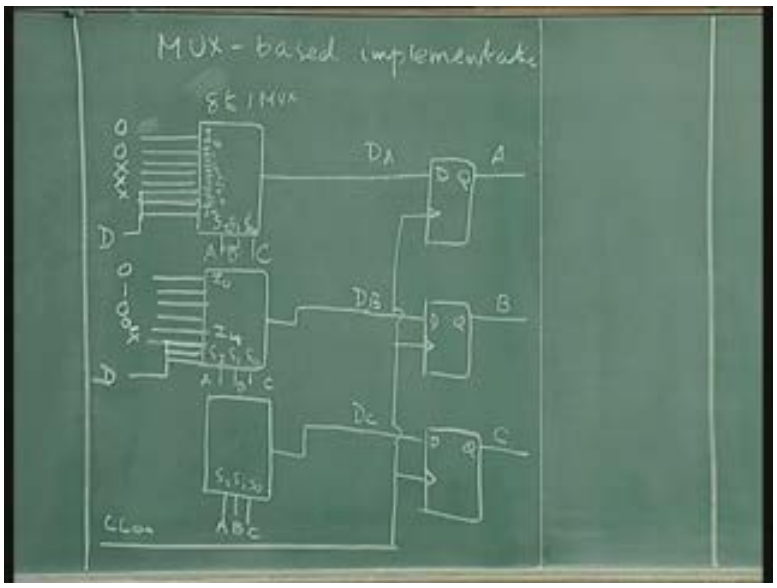
For S_3 it is the same, it follows X next state variable follows the input that means it is X again; for S_4 it is again the same thing, it so happens, I just arbitrarily took it, circuit is in state 1 0 0, X is 0 the next state is 0 for A, X is 1 next state is 1 so it is again X. So the first two 0s, next three Xs and these three are don't cares.

Do not confuse with this (Refer Slide Time: 27:47) I will put D here or phi. D is fine. Therefore, the first two inputs are tied to 0, next three inputs should be tied to input X and the last three inputs can be tied to anything, if you want another 0.

Do this for the next one, so I want to concentrate on 0 1 2 3 4 and 5 6 7 are anyway don't cares, what is I0 to I four?

Now you will have to look at the second column flip-flop B column. Present state is 0 0 0 next state is 0 irrespective of the value of X where X is 0 next set 0 X one X is one and next state is 0. second state, the circuit is in one state and you want to find out the next state of the flip-flop B, it is always 1, next state of the flip-flop is 1 from the present state it is 1 whether or not X is present or not, X is present so that is a 1. So what should be the next one? It is 0. The circuit is in S_2 irrespective of the value of X the next state of the flip-flop is 0. S_3 is the same, S4 is X bar. The circuit is in S4 if the value of X is 0 then the next state of the flip-flop B is 1. The value of X is 1 then the next state of flip-flop B is 0, it is complement of X so X bar.

(Refer Slide Time: 30:10)



So tie the first 1 3 4 to 0, second 1 to 1 and fifth one to X bar. So the design is so mechanical that mean this is common for whatever. If the state graph changes the only thing that will change is this. As long as there are up to eight states in my circuit in a state

diagram there is a common circuit diagram, the circuit diagram is common up to eight states, the only thing that will be different from design to design is the set of input variables, inputs that you need to…[…….. 31:00].

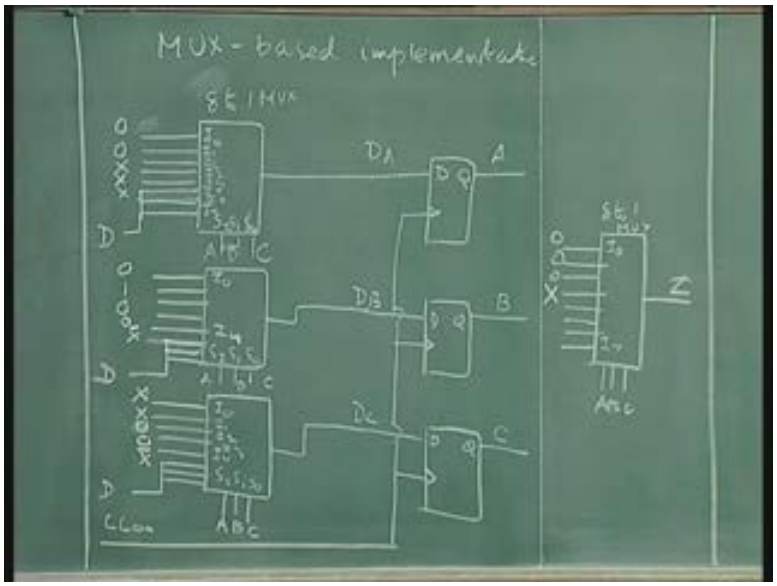
There again, elegance is there, this is a very elegant circuit, implementation and number of ICs, only three muxes, there are whole lot of gates even though these muxes are functionally more complex, as I said, cost wise and everything else they are comparable so we don't have to worry about having to use multiplexers instead of gates. Now the last three is again don't care so I_0 I_4 and 1 2 3. The present state is 0, next state value is the same as input value, the present state is 0 0 0, next state of C is 0 and X is 0 and if this is 1 X is also 1 so this has to be X again.

Next is X again because of this, the X value is reflected, C plus follows X when the circuit is in state S_1 so it is X and the next is 0 and the next is 0 and finally it is X bar. If the circuit is in S_4 state the input is 0 and output is 1, I mean when X is 0 the next state is 1 for C and if the input is 1 the next state is 0 for C so it is X bar.

I need to do horizontal mapping from here to here to here. Now we have finished this circuit implementation and then the output is not completed. The output is again you draw a Karnaugh map for the output because there are only four entries of 1 which is less than the zero in number. Now I can draw a Karnaugh map to make a simple gate circuit but if you want to use any multiplexer then it's fine, I can give a multiplexer based output also.

Now if you give the same input variables A B C, this is an 8 to 1 mux (Refer Slide Time 34:00) and the output of the multiplexer can be the output of my C but again it depends on what present state the output is, the output is determined by the present state and the input variable. In S_0 state the X is 0 the output is 0, X is 1 the output is 1 and then 0 so it goes like that. so I can use this same technique of drawing the multiplexer based output so there will I_0 to I_7 that is 1 2 3 4 5 6 7 and it is 0 for this state, 0 for this state so for the first three states the output is 0 anyway. For the first three states whether or not X is 0 or 1 whether or not X is 0, whether or not X is present the output is 0.

(Refer Slide Time: 34:50)



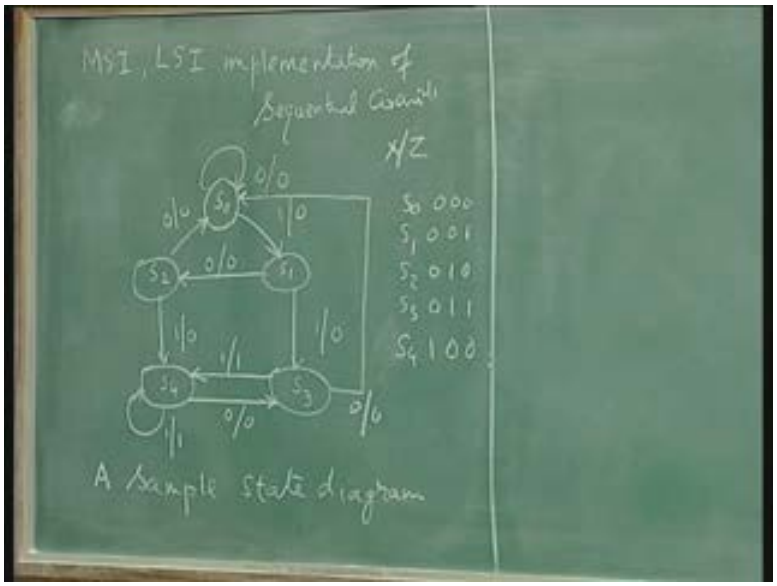
For the fourth state that means the S_3 state if the input is 0 the output is 0 and if the input is 1 the output is 1 so I need to put a X here (Refer Slide Time: 35:10). In the fifth state again as the fourth state it is the same thing where the input is 0 output is 0 and if the input is 1 the output is 1 X and we assume that the last three will be don't cares so we will put zeros anyway.

Because you have a state and the moment really going to the state if you have the output as 1 and if you do not require an output then some glitch may be causing and triggering something but this is only a practical tip, muxes instead of flip-flops driven by gates, this is how I used a MSI, it is an implementer.

According to the state graph the output is defined by the present state and the input. This is what the next state is, (Refer Slide Time: 36:26) when the circuit is in S_3 and if the input is 0, the next state is S_0. And this arrow means that from S_3 it goes to S_0 but not now but in the next clock edge. But now in S_3 the output is defined as 0 if X is 0 and 1 if X is 1 in this output corresponding to this state.

Because with the difference in Melee and Moore what did you say, in Moore machine the output is defined for the state, in S_3 state the output is defined, if it has a Moore machine I will put a 0 here let us say this is also 0, this also is 0 and this also is 0 so put 0 here. In Melee machine we are saying that the present state alone does not determine the output but the present state and the input determines the output so the output corresponds to the present state but not to the next states. This is the confusion sometimes people have. Some of the books do not explain this very well. This is the present state output we are defining.

(Refer Slide Time: 37:45)



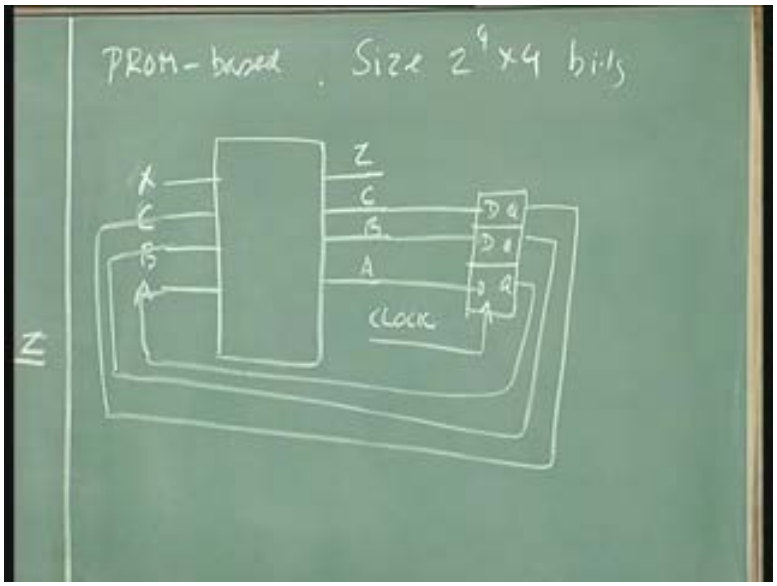
Had I now used a prom instead of a mux that is even easier. There are four outputs which are DA DB DC and C, so how many inputs I need? I need four, this can be considered as a truth table. For a Prom implementation, so what is the size of the prom? This is the address, the inputs to the prom is called as the address that means to tell you what has to be stored in each of these locations in the prom.

There are four addresses that means there are sixteen locations so I need a size of 2 power 4 times four, so 2 power 4 is 16 and 16 into 4 is 64 bits, the answer is right but not practical, if you buy a 64 bits prom it will have four inputs its four outputs I can't implement the circuits. So this is the size of the prom we need and the outputs are Z, this goes into the three flip-flops I will not draw separately, three D flip-flops and the outputs are, this is a standard block diagram of a sequential machine each is a flip-flop. You can also use a four bit register and use the first three bits that is also possible, it need not be three individual flip-flops.

For example, flip-flop come in dual, I told you 4 to 1 multiplexer comes as a dual chip, you cannot by a single 4 to 1 mux but you will have to buy a dual 4 to 1 mux an used one. If you want to by an inverter you will have to buy 6 a six pack we call it in US, if you want to go for a beer you will have to buy a six pack, the price is cheaper,……..

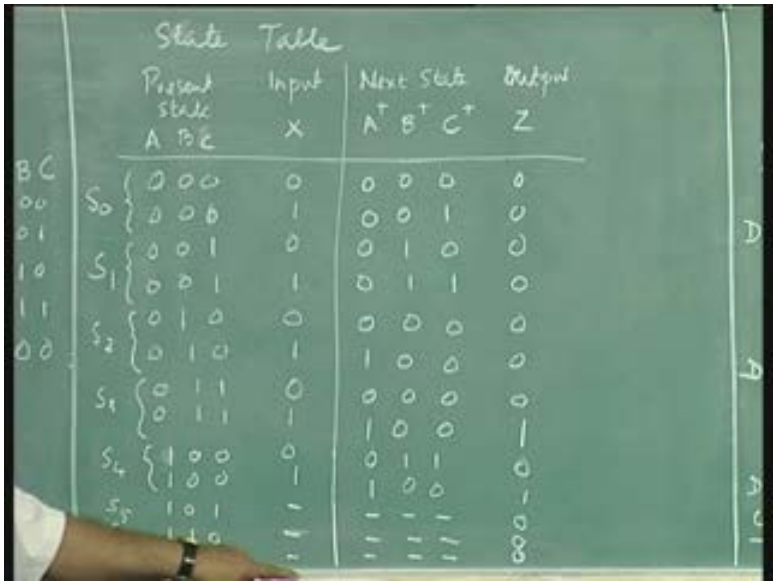
The idea is to commercialize, try to drink more, six bottles you buy you try to consume if faster than one at a time, that is the concept, commercial concept, driving the people to by more and more, so two in one. Similarly, muxes are 2 in 1, D flip-flops are 2 in 1. So anyway I have to buy two flip flps then a third flip-flop then I need to buy a two chip so you can as well buy a four bit register because the register has a common clock for all the flip-flops and four inputs and four outputs that I want. Therefore, sometimes you think you are saving by buying three flip-flops but you will end up saving if you buy a four bit register, I told this several times earlier.

(Refer Slide Time: 41:00)



Therefore, all I have to do is to program this prom as per this table. so present state the first address is 0 I told you ……[…….41:40] so the prom table or the rom content table as we call it, this is the size, the rom content table will be address content both in hexadecimal so I can write 0 1 2 3 there is no need to see because it goes from 0 to f; very first is 0 0 0 and the last is 1 1 1 f (Refer Slide Time: 42:19) so in the first word 0 0 0 the word to be stored is 0, second location I need to store 0 0 1 0 is hexadecimal 2, this is hexadecimal 4, this is hexadecimal 6.

(Refer Slide Time: 41:40)

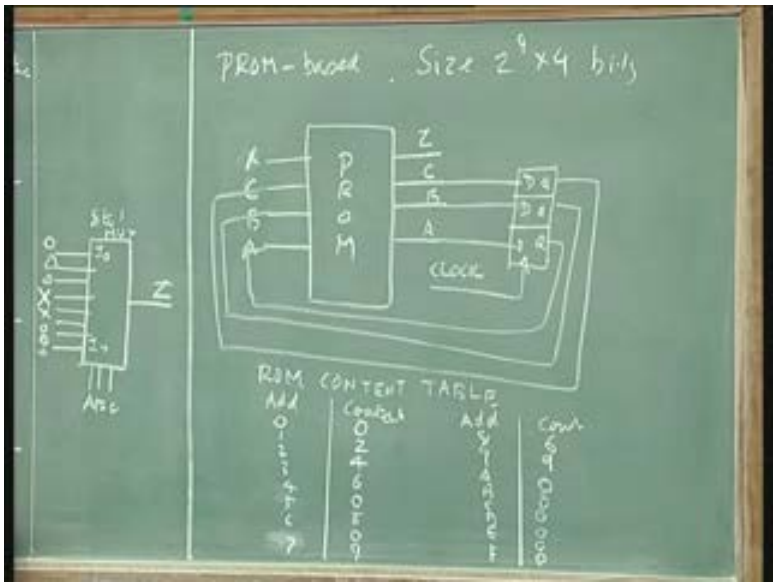


The content is 0 2 4 6 0 8 0 so up to this (Refer Slide Time: 43:04) we will read and then write one more column so after 6 8 0 it is 9 and here this is 4 5 6 7 so 0 to 7 so 8 0 and 9 is 7 so I will continue this table this side (Refer Slide Time: 43:45) continue so 8 9 A B C D E F so after 7 it is 9 and after 8 it is 6 then 9. Now how would you put a don't care in the rom? I cannot write a don't care in the rom because it is a physical address, I give the address and I get the output.

Suppose a state exist or not or whether I give the address or not the corresponding words should be read so I can make all of them same because whenever any of these sates occur which is not going to happen I can always say that the next state is 0 0 0 something like that I have to decide 1 1 1 may be. Thus, whenever a state occurs you put is back to the state which is existing in the circuit usually the reset states so for the rest of these don't cares I will put 0 0 0 and the output is also 0 so I will fill all the column all the rows from here to here.

Therefore, up to ten rows there are addresses which are valid, from tenth to fifteenth six rows there is no output so I have to put a 0. I can put 0 all through. This is called the rom content table. Therefore, take a rom of this size 2 power 4 times 4 get a rom rom programmer from somewhere prom programmer from somewhere insert this prom in your prom programmer and get each word and corresponding content; put 0 0 0; first put a 0 address put the content as 0, increment the address by 1 put the content as 2, increment the address by 1 put the content as 4 and finish the programming.

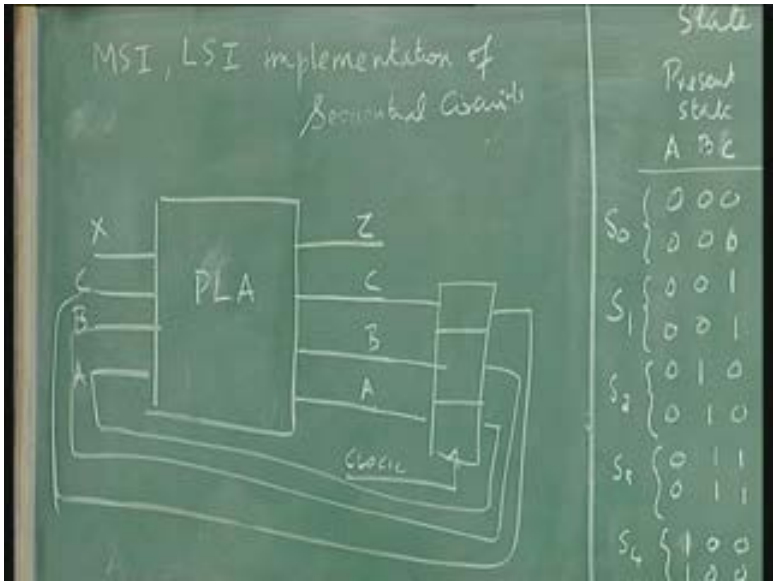
(Refer Slide Time: 45:50)



Then connect it to the flip-flops as shown here, the circuit works. This is again a MSI or LSI based implementation. LSI or MSI here because the rom size is too small I do not want to call in LSI. Usually roms and PLAs and PALs are classified as large scale Integrated Circuits but then the size here is so small, it does not matter actually because these definitions are not important, implementation scheme is more important.

How elegant it is?
One chip rom, we have the appropriate size, program it, there is one register with four bits and a clock, that's it, you give X and the output comes. Again convert this into Programmable Array Logic, programmability wise, so this prom is a programmable logic device, the next programmable logic device is PAL. For PAL I need to go little bit into the thing I am not going to do it, I am going to leave it to you as an exercise. What will be the size of the PAL, four inputs have to be given and four outputs need to come.

(Refer Slide Time: 47:35)



To this we will connect, I am not going to tell you what it would be, the only thing you do not know is how many minimum number of min terms or product terms we can have in reasonably. so determine the programmable logic PLA table that means we go for the minimization of each of these variables A plus B plus C plus and C so each of those four columns we have to minimize with as many common terms as possible then come out with a list of product terms we need to generate and generate those product terms and combine them in the OR gate. Then you can use a Programmable Array Logic also which we will do in the next lecture.