Digital Circuits and Systems

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

Lecture - 36

Design of Circuits using MSI Sequential Blocks

(Refer Slide Time: 00:01:53)

In the last few lectures we have been seeing how to design combinational logic circuits using MSI and LSI building blocks. Some examples for building blocks we used were the multiplexers, decoders, programmable logic devices, especially prom and PLA Programmable Logic Array. Of course PAL also can be used but then PAL also has a sequential part the registers inside. Likewise there are MSI and LSI building blocks in sequential circuits and the 4-bit counter is an ideal example.

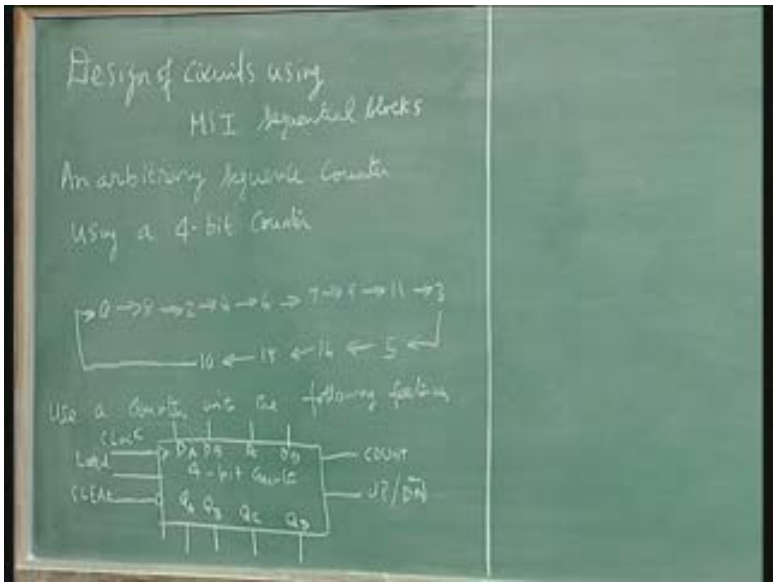
There are shift registers in 4-bit 8-bit. These are the examples which come under the category of MSI because of the number of equivalent gate functions. So like reducing the number of gates, the motivation behind using MSI circuits in combinational design was to reduce the number of gates and make the circuit compact.

Same motivation exists for reducing the sequential building blocks also. So whenever you have a sequential circuit with flip-flops whether a JK flip-flop or D flip-flop or whatever there are individual flip-flops individual ICs which has to be connected together with all the problems of interconnectivity, space etc so all those problems we talked about in combinational logic exist even in sequential circuits.

So to that extent we would like to minimize the number of ICs and the number of interconnections between these ICs even for the sequential building blocks. That's why instead of using flip-flops we can use a register or a counter which has the same function. A register as we know is a bench of flip-flops, counter has flip-flops. And finally when you make a circuit it will have a combinational part with a very few ICs and sequential part with very few one or two ICs and then the whole circuit becomes very compact and if you go one step further of course you will have circuit in which both combinational and sequential like this, and examples are Programmable Array Logic in which registers also can be pushed in so we can have a single IC which will implement the function.

So in that direction we talked about mapping a given function from a truth table on to an IC given rather than minimizing for gates. How do you map a given Karnaugh Map for a multiplexer based solution, for a rom based solution that's what we saw. Likewise today we will see how to map a given state table into a register a counter.

(Refer Slide Time: 5:10)



We call this design of circuits using sequential building blocks MSI sequential building blocks. So, as an example we take an arbitrary sequence counter using a 4-bit counter, four bits counter is available and we want to get an arbitrary sequence counter. When you say 4-bit sequential counter for IC for 4-bit counter it will count sequentially from 0 0 0 0 to 1 1 1 1 or opposite.

Now I want to use that but I have my own sequence of states, that's the problem. So this is the sequence I want; 0 8 2 4 6 etc it is a 4-bit counter so from 0 to 15. I am going to use a counter which is following specifications. We will have a clock of course, any counter should have a clock edge trigger. There is a loading feature that means this counter will have inputs and outputs parallel inputs and parallel outputs so when I put a pattern here DA DB DC DD DE DF sequence of 1 and 0 I can load that into this by applying a signal

in the load input, the values in the DA DB DC DD can be stored in the counter. And the 'clear feature' as all of us know will clear the counter that means it will make it 0 0 0 0.

Count feature is the count up or count down. So when there is a signal given here with every clock cycle it will go by one count and whether it is up or down is decided by this signal. So if this is 1 and this is 1 then it will count up by 1. That means it was 1 0 0 0 let us say as an example and a clock arrives and let us say this is 1 and this is 1 it will go to 1 0 0 1. On the other hand, if there is a 1 0 0 0 then you have to count 1 and then in down this 0 so after the next clock it will become one less than 1 0 0 0 which is 0 1 1 1. So I can count up count down, I can load, I can clear all four bits so inputs are available and outputs are available. Therefore, using this how do I design a circuit which will go through this sequence.

Today I will concentrate only on the sequential part. That means we will draw the state table and how to design it? But once you identify the combinational logic which will steer after again…… see that all problems are basic problems we have set of flip-flops which stores the state variables, which are steered to the next state by means of combinational logic, there is no change in that. Here also there has to be a combinational logic with will steer this counter from state to state as per this requirement so that combinational part can again be implemented using muxes or prom.

Since I don't want to complicate, not exactly complicate but the problem becomes bigger to handle I will assume for this discussion alone that we will still use the gate solution for the combinational logic because I want to concentrate on the register steering, how the register will be counted and steered from state to state, what type of inputs will steer it from state to state as per this requirement and what combinational logic is required and that combinational logic we will produce in terms of gates today but you know how to do it in terms of multiplexers or prom. So finally we can do it together as a single compact solution.

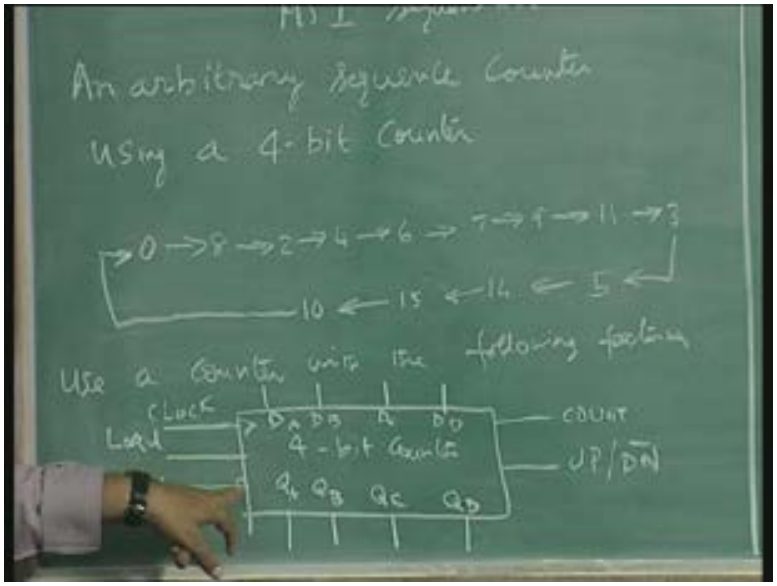
Now when you have all these signals you have to define some priority. What will happen if load and clear come on the same time? What happens if count and load comes here? If count and load comes will it count or will it load, so all these values will come in. Therefore let us define a priority of operation because these are all in the data book. I am assuming all these in an arbitrary fashion.

Once we decide which counter to use all you have to do is to go to the data book and look at all those features and see how these features are interconnected with each other and how are they dependent on each other. So since I have to give you an example I have to assume certain things before proceeding so I am going to assume that clear has the highest priority.

Let us assume all are synchronous. That means whatever you give to the clear signal, whatever you give to the load signal, whatever you give to up down, whatever you give to the count the action will be only at the next clock edge you assume that. Therefore, all are synchronous operations. But when all the signals are present in a confusing way the

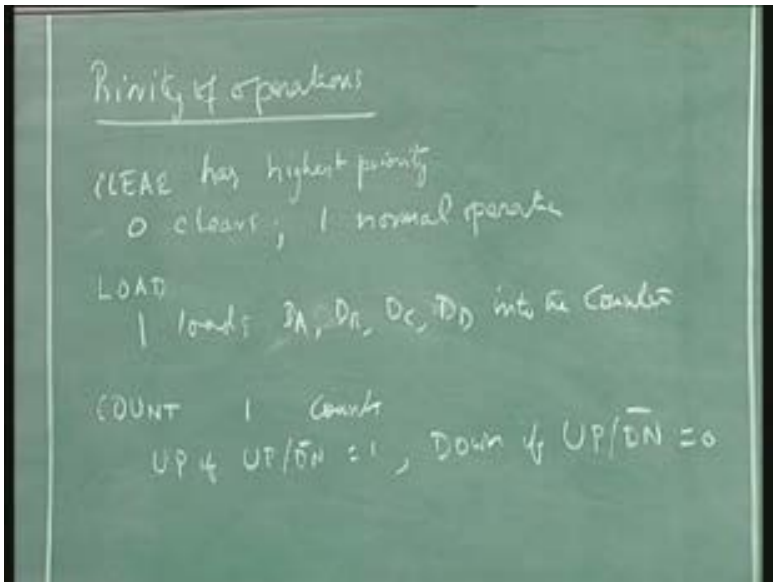
counter will look at the clear signal and if it is zero it will clear because I put a bubble here (Refer Slide Time: 10:20). It is clear 0, clear Z, clear 1……. So that it can go through the rest of the operations.

(Refer Slide Time: 10:30)



Then comes the load, 0 clear, 1 normal operation then comes the next priority; if clear is 1 that means the normal operation then if the load is 1 and the clock pulse occurs it will take the values of DA DB DC DD DE and put in on the counter so that QA QB QC will be same as DA DB DC DD after the clock pulse. So loading is the next feature, it's a 1 loads, 0 normal operation. Now, 1 load is DA DB DC DD into the counter, 0 means does not load. Then only the importance of count comes.

(Refer Slide Time: 12:10)



If there is no clear that means if this is 1 and this is 0 then count becomes significant. So the next priority is count 1 which counts up if up down is 1 down if up down is 0 and that is why you put down bar here which means…….

Now you have to decide whether you want up or down otherwise it does not know to count up or down. That means there are different operations possible, I can load a new value, I can clear the existing value or I can let it go up by 1 or go down by 1. These are the features available to me for me to implement this sequence using this particular 4-bit counter. So what connection should I make to load, to clear, to count, for up and down for each case and then we have to draw state table which then becomes a transition table.

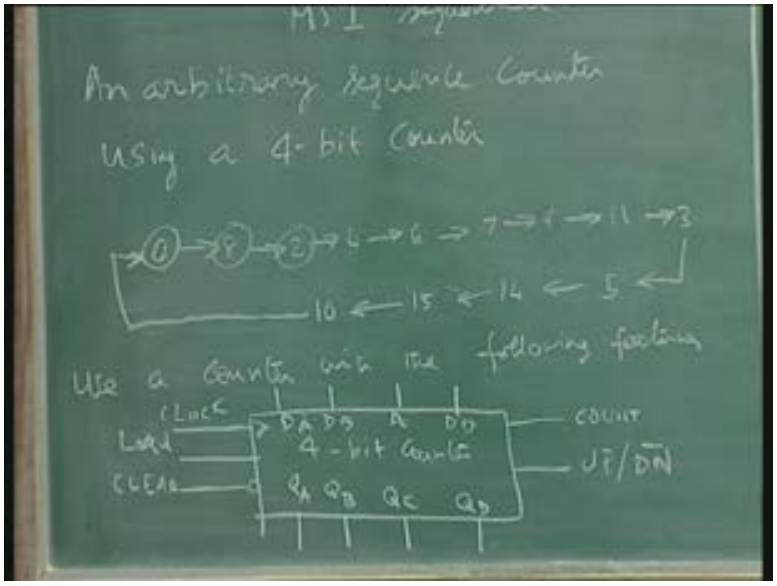
So present state, the next state is the same as that as a state table but then all when all these extra inputs come it becomes a transition table and you simplify the transition table to get the combinational logic.

Today I am going to do the gate solution as I told you because otherwise it will take some more time.

So I am going to put one condition that all these signals are synchronized, all these control signals are synchronized. What you mean by synchronous? Synchronous means at the next clock edge so the register to the clock at the next clock edge. I may put 1 here but it does not mean that immediately this value will go in, it will only go in at the next clock edge. If I put a 0 here it will not clear immediately but it will clear only in the next clock edge. I can put a 1 and 1 here it will not count up immediately and if it counts up immediately then there is no point, I need to do it one at a time. Therefore, with this background let us draw the state table and the transition table.

Of course this is the state table really, state graph except that I did not put any circle (Refer Slide Time: 14:25).

(Refer Slide Time: 14:26)



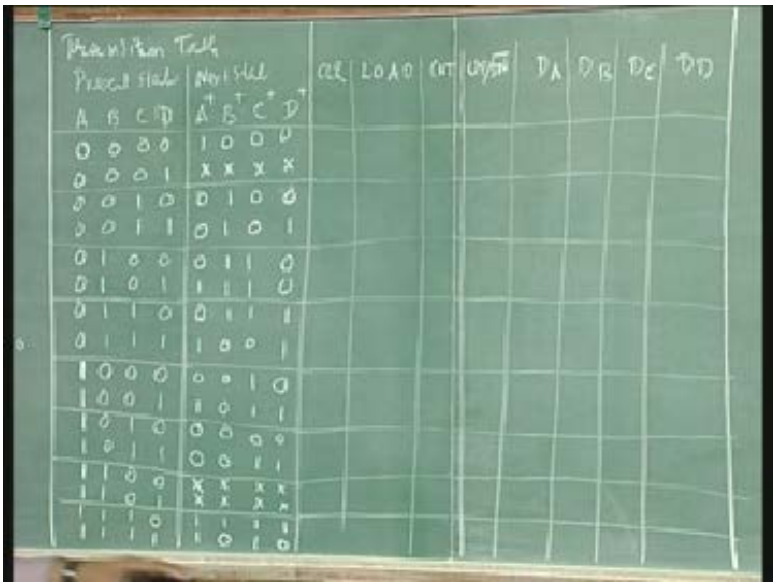
There is no other inputs so where is the question of drawing a separate state graph. So transition table will have the present count present state which are A B C D, the next state A B C D again and you want to make a distinction here by calling it A plus signal B plus signal and so on.

Then what are the various things you need to give for this transition to happen. So what should be the clear, we put will this in the same order of priority, so it is clear what should be load, what should be count, if 1 or 0, and what should be up down should be 0 or 1, I am going to move this line here.

If I had to load here I should know what to load so what should be value DA DB DC DD every time want to load, I can't load arbitrary things, I have to load correct things so I should also write DA DB DC DD. So let us write a few things. Just draw four so that we may not make a mistake of……..

These lines I require so that I keep them in a straight line.

(Refer Slide Time: 19:55)



Can you just read it out?
The present state is 0 and the next is 8? (Refer Slide Time: 17:50) if the present is 1 what is the next state? Not defined, 2 next state is 4 and here 0 0 1 1 five, and 4 next state is…… five next state is…. fourteen is 1 1 1 0, six next state….. seven next state, eight next state……., nine next state, eleven next state…. ten next state…. 0….. eleven…. three, twelve not defined, thirteen….. fourteen fifteen, fifteen this is the next state, so up to this it is called the state table and if you give inputs also it becomes a transition table.

Now if the present state is 0 next state is eight I cannot do it by clearing, I cannot do it by up count, I cannot do it by down count so I need to load. Now clear is my highest priority so I have to make sure it does not clear that means I have to put a 1 in this (Refer Slide Time: 20:32). This cannot be don't care. If it is a don't care but supposing if it takes a value 0 then it will clear and…[…20:41] because clear is my highest priority because I put it in the order of priority so it is a loading feature we want so we put 1 in the loading, count is don't care because since this has a higher priority I can put a don't care on this. Count again you have don't care because load is of higher priority whereas for clear I cannot put a don't care because it has higher priority than load and if we put a 0 there it will clear or not load. So because of that there is no need for any values for DA DB DC DD.

Now I have to give the load value, what is value of load? It is 1 0 0 0 that is the new value. Thus, what you are going to load now is the next state. Next don't care, it can be anything because even if it is clear it is no problem, if it counts up no problem, counts down no problem because if that state happens you have to make sure that you go properly.

Hence, this is a don't care condition and I can put all of them as don't cares. It could clear for example because don't care happens to be zero, it could load some value because load

can be 1 so some arbitrary values can get loaded but since it is not a condition we are going to have, if it is an arbitrary value which is the condition we have as a part of the count then automatically the next state will be defined based on that.

Therefore, I do not have to worry about the 'don't care' state, I can give all of them don't cares. Now in this case whatever can happen; it can get cleared, it can load the new value, it can count up, count down but since this state is not defined whatever is the next state it assumes and from that state if we proceed further there is no harm in that. Now 2 to 4 again is only a loading feature, 3 to 5 is a loading feature, 4 to 6 also is a loading feature and mostly it is a loading feature, I did not choose anything, it is an arbitrary sequence I got.

(Refer Slide Time: 23:23)

This is 5, this is 7, this is 14 so that is again a ……. feature, 6, this is 7, count up, of course you can do the whole thing by the loading feature also it does not matter. This is just a circuit which I took as an example just to show you how to control various inputs that I am going to make as a count of features I can do a load no problem. Therefore, now I have to be very clear that it cannot be clear, it cannot load because count up comes as a lower priority than load so I have to make sure that it does not load by putting it as a zero, I cannot put a don't care here (Refer Slide Time: 24:23) I want to put load, I want to put a count up which means count up 1 1 and inputs are no meaning because it is going to count up.

Here do you know why it should be 1? I do not want to clear it, I do not want to load it, I could have loaded it, there are two ways of doing this; one is to load a value of 7, in this case I would have done it my away the previous things, there is 1 here, don't care, don't care, 7..

I want to show you how to use other options, the other options being count up. So if this value is 6 and the next value is 7 I can do a count up. and count up I have to do by putting a 1 here and 1 here and make sure that it does not load so I put 0 here and when I am going to count up it does not matter what I can do to the inputs.

Here (Refer Slide Time: 25:40) this is 7 to 9 load, 8 to 2 load, 9 to 11 load, and for 10 I want to use the clear option again. I can load 0 0 0 0 nobody prevents me but I want to show you the other feature, I have never used the clear feature so now I want to use the clear feature. I wanted to use the count up feature I did it so now let me use the clear feature. so when clear is there I will have to make it clear and if it is clear it does not matter what the others are because clear is the higher priority input and now it doesn't matter what is connected to the load, what is connected to this, what is connected to this and so on.

Doing it this way I also have more don't cares in my table and more don't cares is better for my logic. This is (Refer Slide Time: 26:58) is 11 to 3 load, 12 to clear so again this is all don't care. nothing in going to happen when you use don't care to all these because it might get cleared in the worst case, it might be loaded with a new value, it can count up or it can count down, what if it all happens? Therefore, whatever happens we are going to go to a state which is already available and from there we know how to proceed.

Now this state (Refer Slide Time: 27:32) is not going to occur and that's why it is a don't care but if it does occur because of some transitional switching or anything, based on the condition of clear or whatever it is at that time it will go to another state which is a legal state and from there it will proceed.

This is 13 and for 14 to 15 I would like to do a count up, but actually I would have done a count down which would have been nice just to show you the difference. So can I make it different? If I make a change here then I should make a corresponding change here also…. leave it. So 14 to 15 is the count up. Count up means it is no clear, no load, this is the count of feature 14 to 15, and this is the loading feature 15 to 10.

(Refer Slide Time: 28:40)

Now I can do a rom solution for this, it is an ideal [can….28:50] for rom solution. Because after all, all I need is a prom, all I need is a……. size of the rom is how much? It is 2 power 4, there are only four inputs namely the present states A B C D and the next states information is clear, load etc. Here you do not need these (Refer Slide Time: 29:15) I just need to store the clear value for each of the present states, I need to know what is clear, load, what it this etc, so how many? There are 1 2 3 4 5 6 7 8 so 2 power 4 times 8 is the simplest solution.

So I can have a prom solution, prom solution is the simplest, take it as an exercise, prom solution for combinational logic, size is 2 power 4 times 8 so address will be 0 to f and content you read as whatever. We learn to assume that all don't cares are 0, otherwise we will make some error.

Of course you can have anything but generally we will assume that all don't cares are zeros so first bite would be 1 1 0 0 1 0 0 0 that means C8. The first word of the prom, first address of the prom should contain the content of the clear value, load value, count, up down for that particular load value DA DB DC DD for that particular input combination. If the present state is 0 0 0 0 I want clear to be 1, load to be 1, count should be don't care, up and down should be don't care, DA should be 1, DB should be 1, DC should be 0, DD should be 0 and DE should be 0. I have to put it inside the prom, you say hardware truth table, prom is nothing but a hardware truth table.

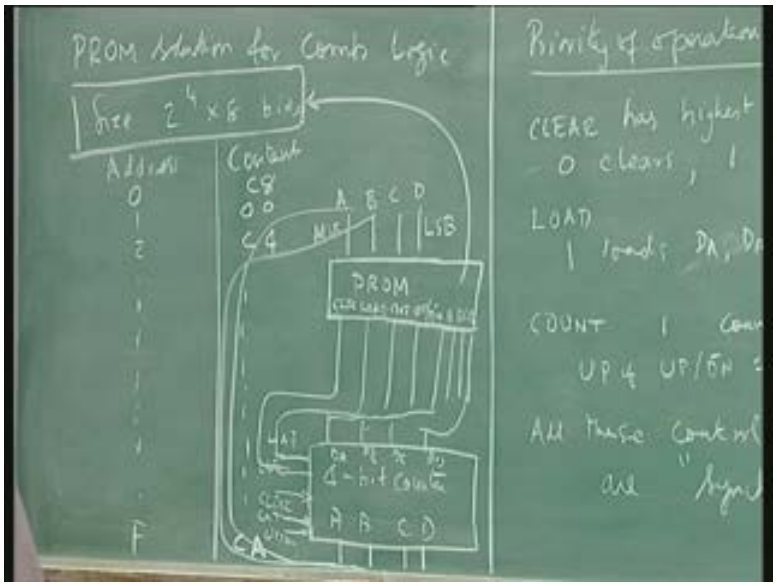
So burn as I even said, it is like etching on silicon. That means I need to put those values. So I can't put a don't care value, I have to put a 0 value. I can put anything but let us be uniform. When you try to understand the table I would like to have it uniform. so for this purpose of the prom implementation I will assume that all don't cares should be zeros so the content 0 0 0 0 is the address and the content correspondent to that address is 1 1 0 0 1 0 0 0 eight bits 1 1 0 0 1 0 0 0, it is a eight bit concatenated into hexadecimal form. The

first four bits is one hexadecimal and the second four bits is another hexadecimal digit so the first hexadecimal is the 1 1 0 0 which is C, and the second four hexadecimal value is 1 0 0 0 which is eight which is what C8 is.

Now what is the next table? What is the next content? It is 1, the 1 will have 0 0, address 2 will have 1 1 0 0 0 1 1 0 C4 correct and last one is f which is 1 1 0 0 1 0 1 0 Ca ……..and in between please fill.

Now let us get back to the rails problem. Prom solution is beautiful solution now I have, how it is going to be hardware wise? I have prom here, I have this 4-bit register, I have this 4-bit counter, I have prom full size is this, this fellow is here ad these values are this is a b c d and am assuming this as MSB and D is LSB. You have to be careful which bit is what, otherwise it will reverse. The values are the way in which I put; clear value, clear load, up count, up down DA DB DC DD clear load, count up down, A B C D 1 2 3 4 5 6 7 8 so this is the address, this is the thing and this goes into my 4-bit counter which gives me the load feature, clear feature, clock force and so on.

(Refer Slide Time: 36:10)



This is (Refer Slide Time: 36:15) clear, this is count up down DA DB DC DD so we have to match this, clear should be connected to this, load should be connected to this and this is A this is D etc so all these connections will be made. Then the output is A B C D E or this A B C D this is the present state which becomes the next state address so this A will be connected back to this, B will be connected back to this. What an elegant solution.

A 4-bit counter and a 2 power 4 times eight rom and I got the whole circuit implemented. So digital design is a very very elegant subject. Don't worry about the Boolean algebra and lots of these gates and exclusively OR NAND and all those, of course we need to learn these fundamentals but today in the hardware it's so simple to make.

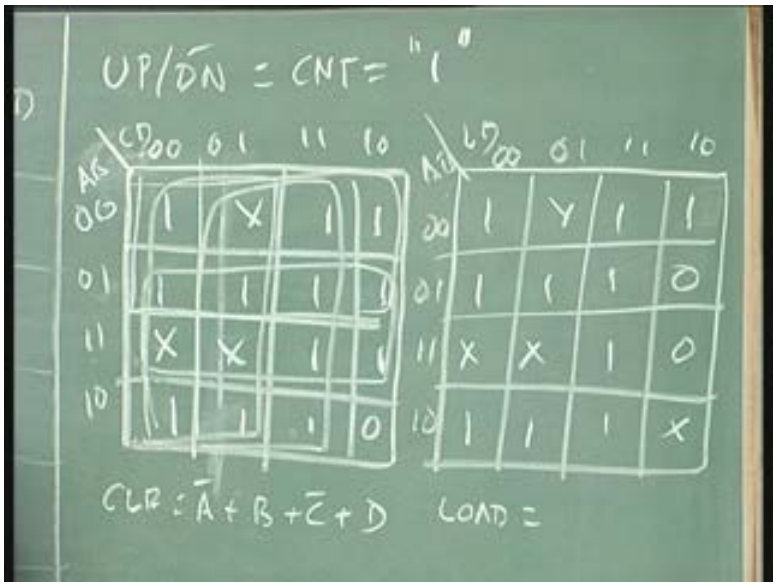
So let us get on with our design, let us do the gate based design for this. So for each of these outputs; clear, load, count, up down, DA DB DC DD we need the Karnaugh Map and then implement it using logic so we will do one. For example, up down is one always, all don't cares we assume 1 so these are don't cares. This is 1 so straight away I can write up down is same as count which is always one no logic no simplification.

This is coming back to the gate solutions, that is the prom solution and I have finished it. This is the table with the prom, (Refer Slide Time: 38:23) the prom output controls the counter, counter output control the address, it is fed back to the address of the rom. Here is this example where again the same thing I want to do that is instead of the prom I want to put individual gate based solution because that is our traditional solution.

Let us do for two things now. Load and clear seems to be easy. This is for clear and for that you draw the map. The values are 1, don't care, 1, 1 and 1 1 0 1 …….. so this is the clear map. That means I can almost combine everything; this one, this one, I have to have three terms and in no other way I can reduce it. Three terms are required. I can make this one (Refer Slide Time: 40:15) this one and this one. And here the values are…….. So this is A bar plus, this is B bar plus and this is C bar sp this is one solution.

I do not have to do everything, this is there, this is A, this is B, so four terms are required, now clear is nothing but A bar plus B plus C bar plus D. Give me a value for load…. Everything is 1 except…. So it is a 1 don't care, 1 1 and 1 1 0 1, 1 1 don't care 1 and don't care don't care 0 1.

(Refer Slide Time: 41:10)



So here I need to put this as 1 and this as 1. So these two and one more, so three terms are required. Load is equal to B bar plus C bar likewise I have already written for up down, I have already written for count, so likewise you have to do maps for DA DB DC DD then your circuit it going to be again the same way except that it is going to be a little more

complex. so load is a clock and load will be connected from the combinational logic corresponding to load so three inputs or gates, I will just show one; B bar C bar D and then we have to have clear put the corresponding map for that, put the corresponding circuit for this, then do the corresponding things for up and down is 1, count is 1, up down is 1 and then we have DA DB DC DD and QA QB QC QD and the A B C D values you are talking about here are A B C D here.

(Refer Slide Time: 44:30)

The A B C D you are talking about in the maps are the outputs of the present state. The present state output combined with extra logic goes as inputs, load, clear, up down, count, DA DB DC DD you know how to do that. So please complete that prom table and do one more thing, instead of using 4-bit counter suppose you use a shift register then repeat the problem.

Thus, this exercise is completed, 2 it is mostly done, all I have to do is to do the DA DB DC DD map and here all I have to do is to complete the table that is for completing this part of problem. I want you to repeat this problem using a 4-bit shift register instead of a counter. You should know what are the features available for shift register; clock of course, load, a new value of this can be loaded, clear if you want to, clear for negative signal, load for positive signal, say shift and we will say left or right. If this is 1 then I will put left bar right, conventionally right is correct, right is 1 left is 0. If it is a 0 it shifts to the left just by one bit position, if it is 1 it goes to right. Again the same priority scheme will apply. First is clear, if the clear is 0 it is clear irrespective of the value of load.

Second priority will be load, third priority will be shift along with left and right. So do this same problem, it is an example of showing how to use sequential MSIs. Because no circuit can exist as combinational logic. Prom is good and multiplexer based solutions are good but unless we use them in a sequential circuit……of course we also saw how to

use combinational logic as part of sequential circuit steering logic but the sequential circuit itself can be a MSI counters and the shift registers and putting them together will be a nice combination of one or two chips for combinational logic mostly one chip for sequential logic and interconnection is very little and going one step further push these sequential logic flip-flops into the combinational logic and make it a single IC in which combinational building block and sequential building blocks are programmable, and array logic is an example.

The other things I talked about the CPLDs Complex Programmable Logic Devices and FPGS Field Programmable Gate Arrays is an example of these things. So we have a single IC solution for every complex digital system, that is how you see very simple systems.

In today's electronics or anything you open there will be so much inside but IC will be very very small somewhere inside. So example of computer chip, it contains lot of combinational lot of sequential registers and chips.

With this we will finish our discussion on all the building blocks, MSI LSI everything. So we need to go for system design. Of course we have done system design in many cases. We talked about pattern generators, pattern checkers and all that and we will also do some more practical systems like a traffic controller, how to design a traffic controller. So a few examples we will see based on the time available.