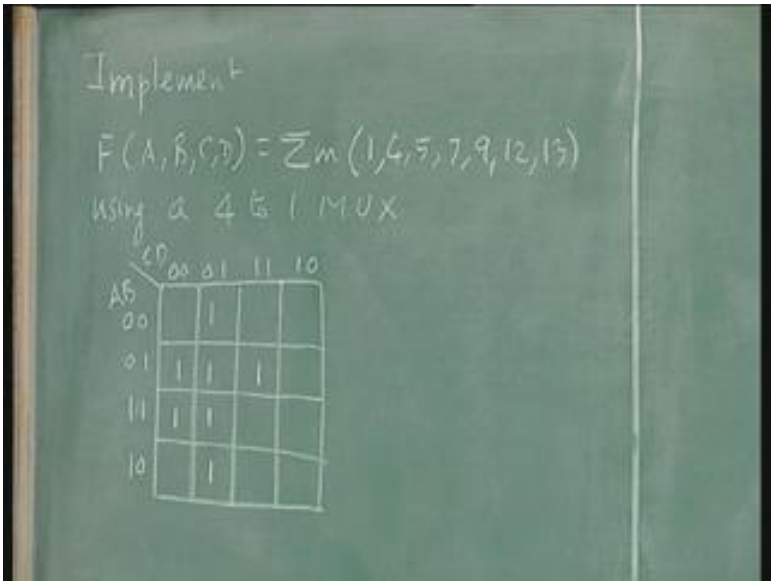
Digital Circuits and Systems
Prof. S. Srinivasan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Lecture - 29
Multiplexer Based Design

(Refer Slide Time: 1:48)

Yesterday we talked about multiplexers, talked about MSI based design, combinational logic and took multiplexers as the example of the MSI logic. Today we will see one more example of the multiplexer based design. So let us take a Karnaugh Map with four variables and see how we can implement it using a multiplexer.

Implement F(A, B C, D) is equal to min terms (1, 4, 5, 7, 9, 12, 13) using 4 to 1 MUX. So we can draw the Karnaugh Map for this, AB and CD are the variables in these two directions (Refer Slide Time: 3:23) you map this function on this map and it becomes 1 4 5 7 9 12 13, 1 2 3 4 5 6 7 8 9 10 11 12 13.

(Refer Slide Time: 4:02)



Now since we wanted to use only 4 to 1 MUX I have only four inputs and one output and there will be two selector variables you have to decide two of these four variables A B C D as selector variables and then get the inputs I_0 I_1 I_2 I_3 as functions of the other two variables. So I can have a 4 to 1 MUX let us say a b arbitrarily. Actually you can make a sort of systematic analysis and find out which two variables if you take will have a minimum gate count. You can do that if you want to.

For example; you don't have to always take a b as the selector variables, I can take any two of the four variables as selector variables and the other two variables become the functions of the inputs, inputs will be the functions of the other two variables I_0 I_1 I_2 I_3 I_4 and I_0 I_1 I_2 I_3 will be the functions of the other two variables and you need to have some gate combinations to connect these two variables into different inputs and certain combinations will require less gates than certain other combinations so I can do an exercise of identifying the minimum hardware requirement by choosing different variables as input variables. It can be done as an exercise but as I said its not a very major exercise because the number of gates saving may be one or two whichever way you do it. But if you want to really do it then it is possible to do it. You can always say use the minimum possible minimum extra hardware. if you have given a condition constraint design this function using this 4 to 1 MUX with minimum additional hardware then I will have to do that exercise, will have to choose selectors different combinations and then find out the gates required and compare them to find out the one with a minimum combination the minimum gate requirement and use that selector combination as the selector inputs.

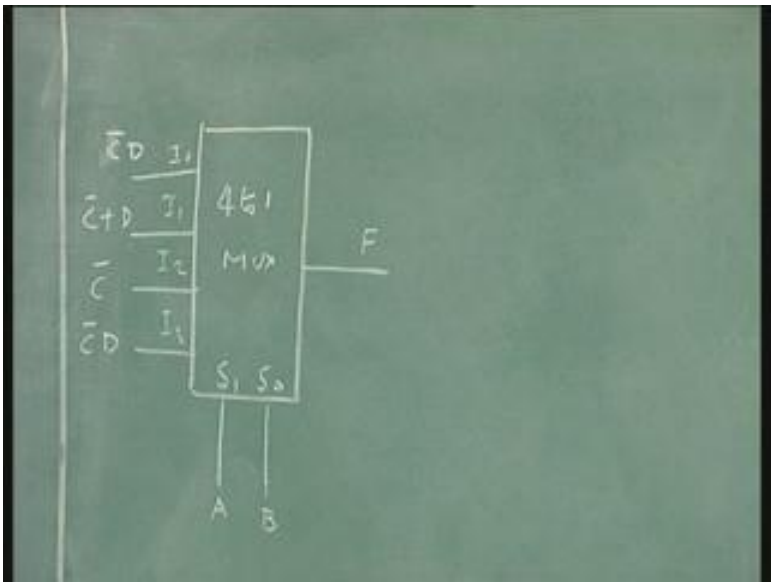
You can also do it as an exercise may be as an assignment later.

Select all other possibilities of the selector variables find out the gate requirement and find out which two selectors give you the least requirement in terms of the extra gates. So

if you take AB as the selector variables the other functions I_0 I_1 I_2 I_3 which will be the inputs to the multiplexer it will be functions of C and d and this will be my output function (Refer Slide Time: 7:16) and that you can very easily see from here using this map because AB 0 0 is this whole row, the row one is AB 0 0 so since we are only considering the first input I_0 it will be connected to the output. When AB is 0 0 that means for this first combination of I_0 the input would be depending on this first row, for the second combination I_1 which is A_0 B_1 this is A_0 B_1 so this is the slice or the row of the Karnaugh Map which will decide the input to the I_1, this will decide the input I_2 and this will decide the input I_3. So let us take AB 0 0 and when AB is 0 0 there is only one entry which is C bar D.

So, if AB is 0 0 I_0 is the output and since I_0 is C bar D if you connect C bar D here that means if I have a gate which will generate C bar D then AB 0 0 output will be C bar D which is what we want from this map. Continuing this exercise for AB 0 1 these three 1s are there which is nothing but C bar plus D, this is C bar or D because this is AND, this is 1 for all entries except the last one which is CD bar so this is C bar or D so I have one OR gate whose inputs are C bar and D and you will get the second output.

(Refer Slide Time: 9:37)



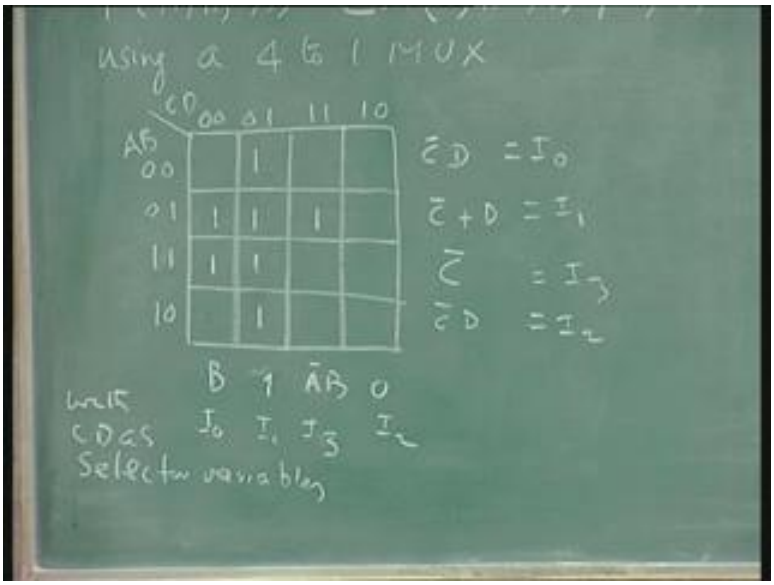
The third one is 1 0 you will have to be careful here because here the map will give you 1 1 this is AB whereas the third input I_2 corresponds to 1 0 A_1 B_0 and this will be 1 1 so this is 1 0 and this is equal to I_0 and this is equal to I_1 and I_2 would be this which is again C bar D. And this is 1 1, when 1 1 is the input to the selector variables, the function is 1 1 here which is equal to C bar. Probably this is one of the lowest gate requirements because this gate is same as this gate this only an inverter and there is anyway an inverter required for this so we don't need an extra inverter here so there is one inverter, one AND gate and this C bar is also there so C bar is common to all of these so we have one inverter, one AND gate and one OR gate probably this is one of the simplest I don't know I didn't do this exercise, I just took this arbitrary thing, the map was arbitrary, the function was

arbitrary. The selection was arbitrary but it so happened that this is one of the simplest. The other ones may not be very different with an extra gate here and an extra gate there. So I_2 would be C bar D, I3 will be C bar. So with 1 MUX, one inverter, one AND gate and one OR gate I will get my function F implemented.

Student conversation (11:30)

As I said I didn't try to do this optimization. What she is saying is if we had taken CD as the selector variables this should be my I_0 so with CD as selector variables, this will be I_0, I_0 would be b, I_1 would be 1, I_2 will be A bar B, this would be 0 so I need only one gate and one inverter one inverter and one AND gate and here I need one AND gate and one OR gate. Of course there is one extra gate. As I said even if you get for this type of simple problem a gate here and a gate there this becomes significant when your design is large and you want to save the number of gates. The saving of the number of gates is significant only at certain circumstances as I mentioned several times in this course.

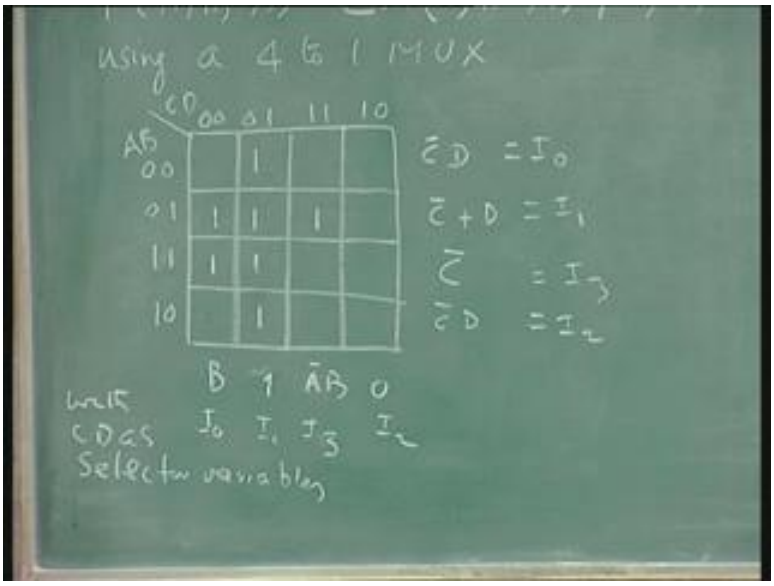
(Refer Slide Time: 13:40)



Even it is for the size, or the power requirement or you want to reduce the number of gates or because you have an IC with so many gates and you want to fit in your design into that if you cannot do that you will have to go for a bigger IC as I said thousand gate IC and two thousand gate IC so if I am somewhere near 990 I would rather try to finish it in this rather than adding another 30 gates which will be 1020. If it may not fit into these thousand gates then I may have to go for that. So except for those few special conditions gates saving is not a big deal today in today's technology so let us say cost effective it's inexpensive so- you don't have to really worry too much about it. We are able to get one gate less in this option than in this option. So this is my I_0. If I had used CD here instead of AB I would have used B as I_0, I would have connected B to I_0 1 to I_1 A bar B to I_3 and 0 to I_2. That is the essence of the multiplexer design. These are all simple straight forward

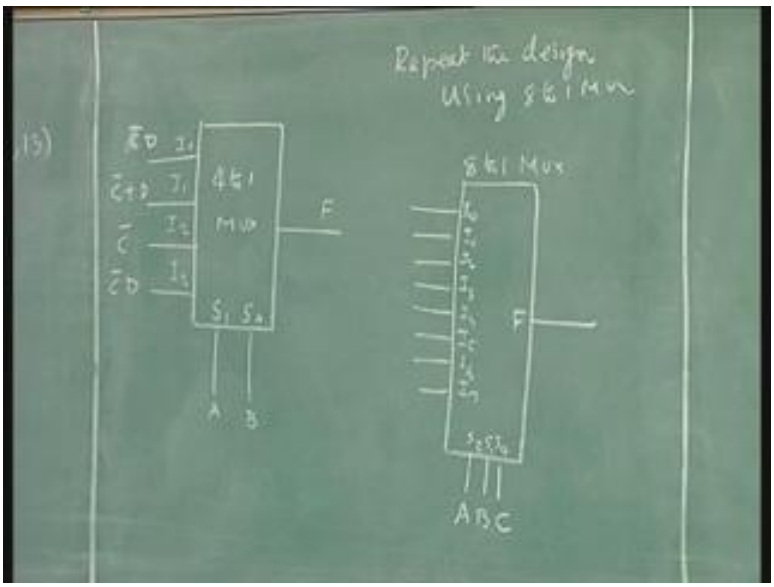
design. There is also elegance in this. The design is so clean and the circuit is also small and not messy.

(Refer Slide Time: 14:11)



You want an eight to 1 multiplexer then I can do this, I can do 8 to 1 multiplexer. So, repeat the design using 8 to 1 MUX. Suppose I have 8 to 1 MUX I have eight inputs, three selectors I_0 I_1 I_2 I_3 I_4 I_5 I_6 I_7 are eight inputs and one output F so this is 8 to 1 MUX.

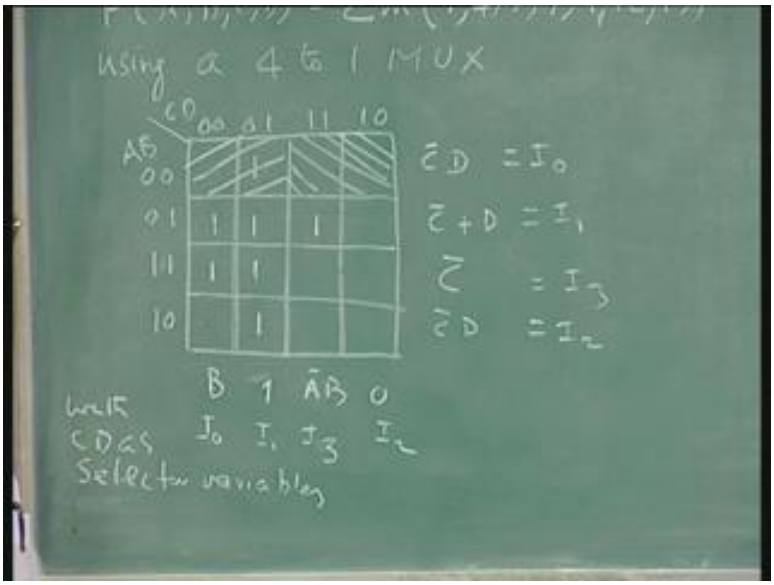
(Refer Slide Time: 15:00)



Now each of these inputs will be decided by two cells in this map. Here because there are two variables and there are four slices or four cells each by each of the variable combinations in the selector there is one row of four cells decided the input. Here there are three variables ABC or whatever then two cells will decide the combination. Here I don't have to about the gates because either it is either going to be 0 or 1, if I have ABC then the only other variable which is going to determine the inputs is the D variable. If I choose ABC as the input D is only other variable so all the inputs are functions of D so either it will be D or D bar 0 or 1 so again there is nothing but to choose from unless you want to be specific and see there is no inverter.

Suppose I had a combination of ABC there and some other combination BCD I may not even require an inverter, one is an inverter and other is a non inverter requirement. So, on all these things you can do a little nitty-gritty and little extra bonus type of design but conceptually it is the same. Therefore I am going to keep ABC as the inputs so when ABC is the input the I_0 will be decided by these two squares 17:10), these two cells decide the I_0. so this is my I_0 because a b c are taken care of so in this cell d is 0 and in this cell D is 1 so if the one is there in this cell then you have a D bar, you have 1 in this cell you have a D and if you have one in both cells then you have 1, if you have 0 in both cells it's a 0 that's all. So, if there is a 0 or 1 or D or D bar in this case it is D.

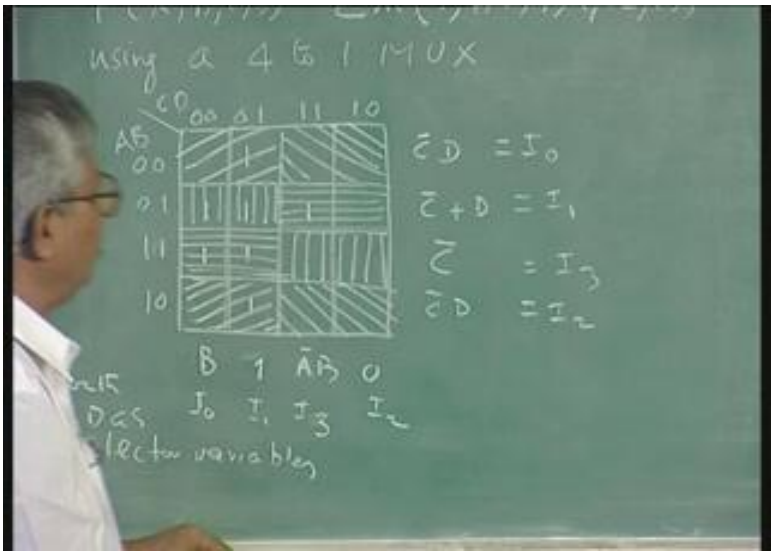
(Refer Slide Time: 17:40)



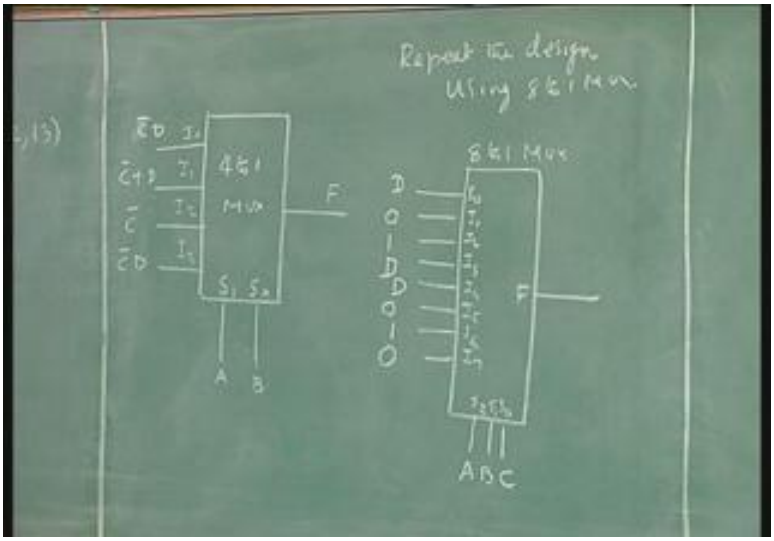
Second is this, it is no inputs no entry. That means corresponding to I_2 which is 0 0 1 for ABC corresponding to these two cells there is no entry so the input I_1 is 0. Input I_1 is 0 and 0 means ground and then this is I_2 and I_2 would be 1 because both these cells are 1, 1 is the power supply voltage V_{cc} as we call it, if it's a bipolar transistor V_{dd} they call it, if it is a CMOS transistor if it's a 5V, a 5V, 3.3 V, 1.2V whatever is the voltage you choose for your operation so that is what that 1 is.

Then I_2 is this, I_0, I_1, I_2 and this is I_3, I_3 as an entry of 1 in D so you have D this is my I_3 (19:12) you have to remember to make sure that you take I_4 as this and not this because again it is a combination of 1 0 0 and 1 0 1 then only comes 1 1 0 1 1 1. So 1 0 4 is I_4 so this is this, this is I_4 again it is D and 1 0 1 is I_5 which is 0. I_6 is 1 1 0 A is 1, B is 1 C is 0 so these two cells so 1, I7 is these two cells 1 1 1 A is 1, B is 1, C is 1 so that is this so 0.

(Refer Slide Time: 20:30)



(Refer Slide Time: 20:35)



We didn't do it purposely but it so happened that there is no inverter needed ABC are connected here and the inputs are either 0, 1 or D there is no D bar not even an inverter that means in a single 8 to 1 MUX you can get a design without any inverter. All you have to do is to connect your 0, 1 or D as per this design. As I said 4 to 1 MUXes are

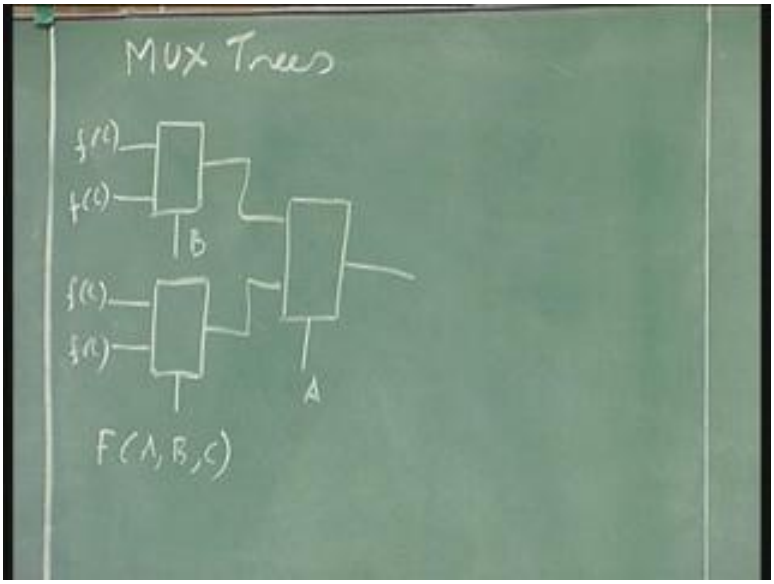
available as dual packages, 8 to 1 MUXes are available as a single package, a single 8 to 1 MUX is 1 IC and with one IC I can get this job done. In this case there is no point in doing this because this is anyway 2 to 1 MUX, 4 to 1 MUX is available in a dual package, I am using only one of them and instead of wasting one IC one MUX of a dual 4 to 1 and then using extra gates I buy a dual 4 to 1 MUX use only half of it and in addition put some gates whereas here I buy a MUX which is 8 to 1 with no other gates so naturally I will prefer this design to this design. Of course this 8 to 1 MUX may be little more expensive than 4 to 1 MUX it doesn't matter because as I said the extra cost increment as a function of the extra complexity is not very much but probably a better design than this from point of view of hardware so you can go on, I can have any number of circuits.

Then if it is 5 to 1 what will you do? If it is 6 to 1 what will you do, so it is similar to Karnaugh Maps. So I can make a maximum of four input design. So, in a five input design what I can do is I can have three as selector variables and these will be functional in two other variables so totally four variables then input are the functions of the fourth variable and three variables are chosen as selectors.

If the function is a five variable function F(A, B, C, D, E) I will choose let us say a b c are selectors and these I_0 to I_7 will be functions of D and E or if you do not want that type of thing I can even have a 16 to 1 MUX it's possible sometimes 16 to 1 MUX are available but otherwise you can make a 8 to 1 MUX and then get a 16 to 1 MUX. A 16 to 1 MUX may be available but I am sure 32 to 1 MUX is not available as a single IC but you have to make it. But it is easy to make MUX of any complexity by using lower order MUXes by a tree structure.

For example, if I connect a, I will call this MUX trees, I have two MUXes of 2 to 1 feeding into another 2 to 1 MUX this is a 4 to 1 MUX (Refer Slide Time: 23:50) because suppose you have a function of two variables I will put one variable here or even three variables F(A, B, C) will be implemented using this. Now I should only 2 to 1 MUX and I should not use any other higher order MUX. If I am given that restriction then what I will do is to use this with one control variable these two with another control variable and these will be functions of C.

(Refer Slide Time: 24:40)

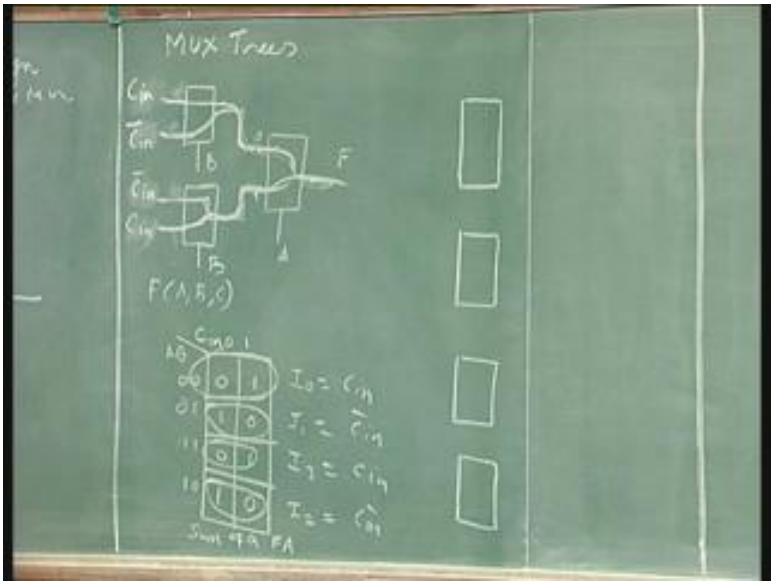


With 2 to 1 MUX feeding to another 2 to 1 MUX so these two together to another 2 to 1 MUX and this is the output. What I mean by this is, let us take the same example of what we did in the last lecture on the full adder. Let us say this is a b c, this is the sum of a full adder (Refer Slide Time: 25:27) and C is a carry in actually. AB are the two bits, I want to implement it using this scheme so how will I do this. This is A_0, this is A_1, this is B_0 B_1 this is B_0 B_1 so this is 0 0 so what I give here would be corresponding to this first row, this branch is 0 0 branch so when A is 0 B is 0 this is my I_0 which is C_{in} so I connect C_{in} to this and this path is nothing but A is 0 B is 1, 0, 1 is this path and the first path was this, second path is this a is 0 b is 1 and A is 0 B is 1 corresponds to I_1 and I1 is C_{in} bar. So whatever is the complexity or condition given to you we should be able to implement that circuit using MUXes only. This is for A is equal to 0 (Refer Slide Time: 27:25) so the first two rows correspond to this input A because A is 0 here and the second two inputs correspond to A is equal to 1 that means these correspond to these two rows and these two rows corresponds to B is equal to 0 and this will be the input and for B is 1 this will be the input.

Therefore the next path is a one b 0 that is this. This corresponds to I_2 that would be C_{in} bar. Finally for this A is 1 and B is 1 the path is this, this is I_3 which is C_{in}. So I have three 2 to 1 MUXes, to the first MUX I give B as the selector and choose first that for the most significant MUX I give A is equal to 0 and A as input so I select this half of the map or this half of the map. This half of the map is implemented when A is equal to 0 and this half of the map is represented, A is 1 corresponds to this half of the map. So, this half of the map will be represented by the input at this point which will be the output of this MUX. This half of the map will be represented by input here which is equal to the output of this. So having decided that this half of the map will be decided by this output which is this MUX there I have B as the selector variables and when b is 0 what happens and when B is 1 what happens? When B is 0 it is C_{in} and when B is 1 it is C_{in} bar.

For this half of the map again B is 0 or B is 1 when B is 0 input is C_{in} bar and b is 1 input is C_{in}. So this is a given restriction that only 2 to 1 MUX is used. This also can be used for higher order MUX. Suppose I want a 16 to 1 MUX or 32 to 1 MUX using 4 to 1 or 8 to 1 MUXes I can use four 8 to 1 MUXes feed it to a 4 to 1 MUX and get a 32 to 1 MUX so this also is another way you can do this so this is let us say MUX tree. This is also a MUX tree.

(Refer Slide Time: 30:10)



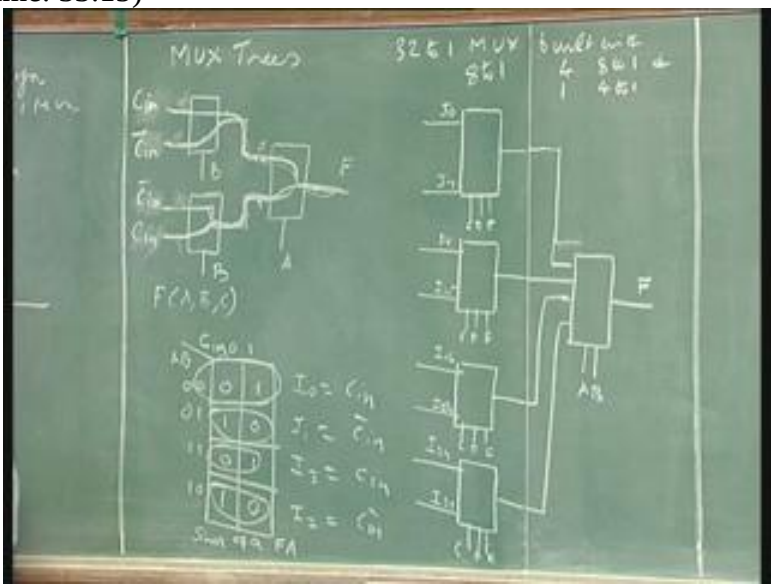
Let us say each of them say 8 to 1 MUX, i am not showing individually, they are identical 8 to 1 MUXes. Let us feed them all into a 4 to 1 MUX so this is my final output. So this output will be selected from 32 inputs which are eight here, eight here, eight here, and eight here so there are 32 inputs and one output so the total thing will be 32 to 1 MUX built with 4 into 4 8 to 1 MUXes and one 4 to 1 MUX. Now each of these I will call them I_0 I_1 I_2 I_3 to I, this will be I8 to I15 and I16 to I32, I16 to I23 and I24 to I31. So, as if there is a 32 to one MUX available to you I can do the Karnaugh Map and mapping and connect this as functions of a b c d e whatever I want to choose.

Supposing I need three inputs here for each one of them two for this so the function is (A, B, C, D, E) five variable function there is no problem. Usually the two most significant variables are used here and three less significant. So what is normally done is this is AB and this is CDE (Refer Slide Time: 32:23). So, 32 to 1 function can be implemented with 0s and 1s here.

Suppose I want to implement a 64 to 1 function this will be (A, B, C, D, E, F) six variables. A 64 variable function will have six variables and A 64 entry 64 cell or sixty-four min terms function will have six variables (A, B, C, D, E, F) so this can be A, B, C and this can be C, D, E, F which will all be functions and I_0 to I_{31} will be functions of variable F. I can even have a seven variable map implemented by using (A, B, C, D, E)

then I_0 to I31 will be the functions of the last two variables F and G and so forth. So I have a choice of using any type of MUXes and any number of multiplexers I want in my design. I can use identical ones, I can use a mix of 8, 4 and 2 or if I am given only 2 to 1 I do this type of design or if I am given one multiplexer I do this type of design so this is a whole variety whole flexibility for you to do.

(Refer Slide Time: 33:15)



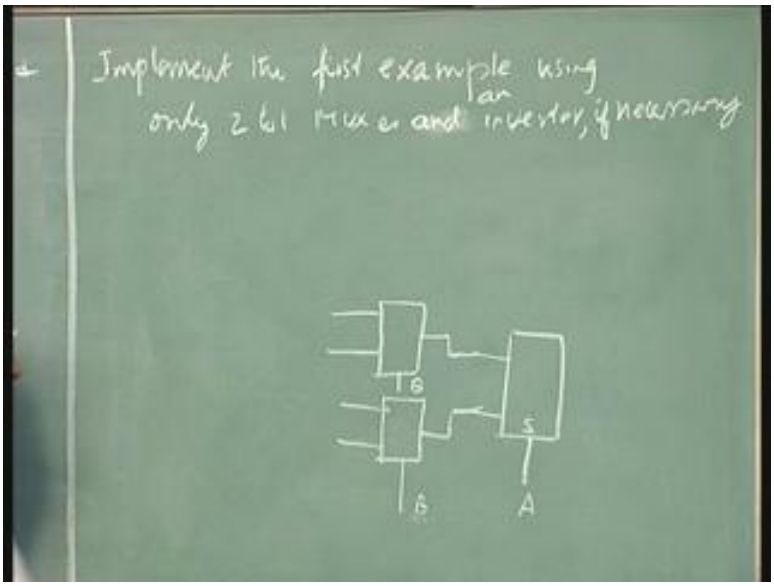
So, if I now go back and do this exercise on only 2 to 1 MUXes, I have only 2 to 1 MUXes so a four variable map I am asked to do this exercise using only two variable MUX how will I do this. I have again a choice of three 2 to 1 MUXes will do because a three 2 to 1 MUXes will have, and I don't want to use any extra gates, if the additional thing is given to me no extra gates are required but then I should have 16 to 1 MUX so I will have to build it. Thus, when additional gate is used and only inverters are allowed then there is one particular solution. If even AND and OR gates are allowed then that will give you another solution. So, depending on what restriction you put you can design. Now let us do this with one variable.

Implement example one, I will call this first example. This is the first example today's is the first example not what you did yesterday. Implement the first example using only 2 to 1 MUXes and inverters if necessary and an inverter if necessary. Of course I can always do various combinations of the variables and then even eliminate that inverter. But as I said that exercise can be done that is not the point but the point is to tell you the design methodology procedure the concept involved ((behind)) multiplexer by design. I am not doing that exercise now.

Let us arbitrarily assume D as the input variable and A B C as the selector variables. And since 2 to 1 MUX is there each can have only one selector so first level you should have a selector A, second level you should have a selector B and the third level should be selector C so that decides my architecture that decides my configuration. Because the last step is A should be the selector this will feed into two, the two inputs of this will feed

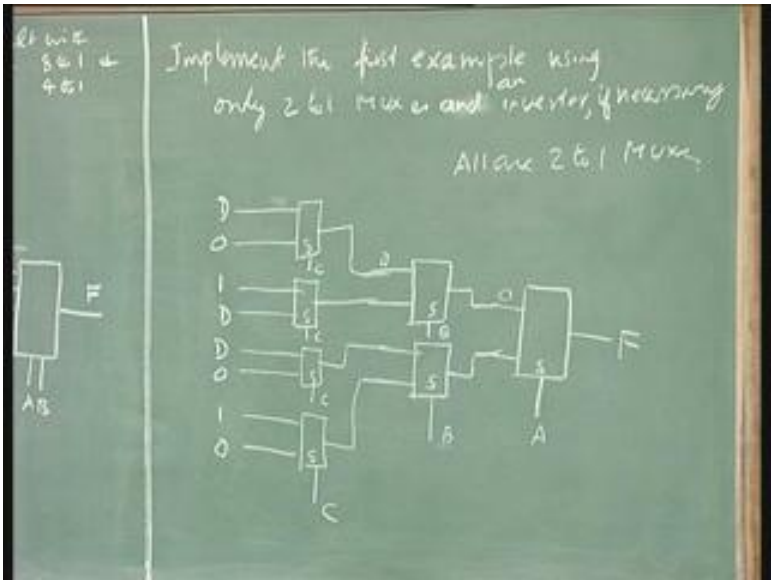
from two 2 to 1 MUXes again. So naturally when I have chosen A here I will have to use B here the next variable (Refer Slide Time: 36:47) and each one of them will feed from a 2 to 1 MUX, four of them and this C will be the selector variable for this.

(Refer Slide Time: 37:15)



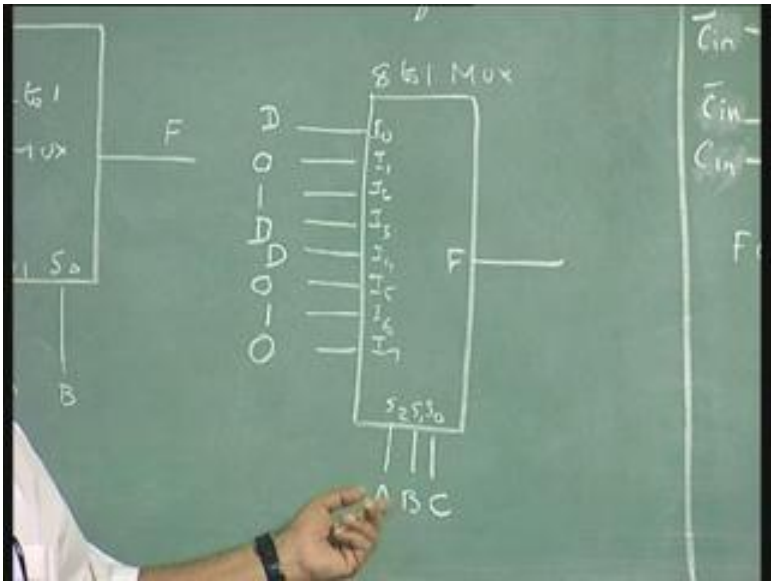
All are 2 to 1 MUXes. So I have two inputs for this, two input for this, two input for this and two input for this and I use ABC as the selector variables in the order of significance this is the highest significant….. So the map is given to the two halves if A is equal to 0 A is equal to 1 and that is sub divided into two halves if B is equal to 0 B is equal to 1 and that is further subdivided into two halves if C is equal to 0, C is equal to 1.

(Refer Slide Time: 38:55)



So now as far as I am concerned everything is inside I don't have to do any routing because if I take this then this is exactly what it is, I have A B C as selector variables and the first is 0 0 0 as I said, this is the 0 of A, this is 0 of b and 0 of C. So if I have 0 0 0 that will be the first entry of this, this will be 0 of A and 0 of B (Refer Slide Time: 38:20) and 1 of C means it is 0 0 1.

(Refer Slide Time: 38:40)



So all I have to do is to copy this into that and that will be D 0 1 D D 0 1 0 (Refer Slide Time: 38:54) it is a multiplexer tree. Now I have done the same example in three different ways. this example I have done using 4 to 1 MUX, I have used 8 to 1 MUX, I have used C of 2 to 1 MUX and if I wanted I could have used 4 to 1 MUX and 2 to 1 MUX combination two 4 to 1 MUX feeding into a 2 to 1 MUX I can do that so whatever way I want I can get it done.

Why am I doing this is because when I have given a IC today like gate count as I said you are given a thousand gate IC but I said gate function only there they are only equivalent functions. When I say a thousand gates IC it does not mean that there are thousand gates connected to it. It is equivalent of a thousand gate function. Whatever you can achieve using thousand gates you can achieve using this IC. That means you can design a system which will probably require a thousand gates. But what they give really is a mix of everything. inside if you look at the hardware of the programmable ICs which are used for making digital systems there will be a bunch of gates of course Exclusive OR gates mainly because Exclusive OR gates is used for adder and all that. There will be a few multiplexers, there will be some D flip-flops, there will be some memory registers so they will give a variety of this. It’s a sort of a menu.

Therefore an IC of thousand gates equivalent will not contain thousand gates of one particular type say AND gate or OR gate but it will contain a few gates, it will contain a few multiplexers, it will contain a few flip-flops, it will contain some registers which are used in memory. And then you are asked to use this IC in the best possible way to implement a given function. So when you do that the mapping becomes even more sort of not straightforward. It's not that as if I have everything that I want is available as a source. So usually some of these things that mix up these different components in a IC which is available today for implementation many of those are two input multiplexers, 2 to 1 multiplexer.

Sometimes there are 4 to 1 MUXes present, and many of these programmable chips which are given to you as thousand gate function chips are three hundred gate function chips and they will contain MUXes which are 2 to 1. Sometimes they also give 4 to 1 but not 8 to 1 and all that. So I should be able to reduce a given function using a series of smaller MUXes and that is why I did this exercise for you. So if I am given an IC which only has 2 to 1 MUXes in it, so given any function I should be able to get that function implemented as long as I do not put the restriction on how many ICs we should use, do not use more than three two input ICs, three input MUXes and get rid if you say then my life might become difficult I may have to do some further simplification and it may or may not be possible. But if I don't have that restriction I am not putting any such restriction.

I didn't say how many you can use, I only said use 2 to 1 MUXes and if possible do not have inverters. So fortunately for us there are no inverters here. But I didn't do it as a deliberate exercise. If possible you get an exercise done this way and say that the inverter is possible or there is no way you can do an inverter less design in this particular case. Either way I can decide and then proceed.

Forget about the inverter for a minute. I am only giving 2 to 1 MUXes as available gates to you and you are supposed to design it and as long as I do not tell you do not more than so many then you are safe and I can do this exercise, the mapping of this into an IC which is a C of 2 to 1 MUXes given to you and a huge function given to you not this four input function but usually a large function. A large function with several inputs and several outputs are given to you along with a MUX based IC which will probably contain about three hundred 2 to 1 MUXes then map it. Of course manual mapping you don't do normally but you do an automatic mapping and automatic mapping is done by programmers who understand logic. So, if you don't know how this is done you cannot write an automatic mapping program.

Why a program is more efficient than another program is because the person who designs that program is more familiar with what is inside and how it is done. So the domain specific knowledge of design is very important if you want to optimize even automatically. The computer is not a Know-all stuff but it is something which is controlled by a programmer. The effectiveness of the computer is decided by the efficiency of the program that is written so if you want to do a best design you have to write best programs which will do it automatically. What you are avoiding is drudgery of

having to do this every time. But the program has to be efficient so that it will always give the best possible solution. With this we will conclude the multiplexer based design. This is only a flavor as I said you can do any number of these exercises of different size. I can do multiplexer gates like this, three multiplexer if you want. I want any type of multiplexer design or I want to do this way, there are other ICs available and one other IC is called decoder. We will talk about it in the next lecture.

A decoder is basically again a simple IC it is a MSI if you want to call it. This is complimentary to a coder. What is a coder? A coder is a circuit which codes inputs. Suppose I have eight inputs to represent and all these eight inputs can only take two values 0 or 1, or instead suppose I have eight inputs and only one of them will be active at a given time so instead of saying input number seven is active, input number three is active, input number 5 is active I can give the code of 0s or 5 or 3 that means eight inputs can be represented by three bits 0 0 0, 0 0 1, 0 1 0, 0 1 1 etc so I give eight inputs and get three outputs it is called eight to three and then this process is called the coding process.

Now in the reverse once you have a coding I need to get it back as a decoder. If the three inputs are given I should be able to get eight inputs back. Similar to multiplexer and de-multiplexer we have a coder and a decoder. So these are again treated as MSIs and this can also be used for implementing of logic functions. In the next lecture we will about briefly, and this not a big complex circuit.

We will talk about the decoders and see how it can be used to design combinational logic like adder for example again and then move on to more sort of involved ICs like LSIs, these are MSIs. The LSIs are large scale circuit in which more functionality is available to you. The problem is once you have more and more functions available to you then you have the responsibility of using it efficiently. That means I can always not get a mapping to my requirement to my choice the best possible way. The multiplexer is a limited function so I am able to map it efficiently.

Supposing it is the IC I told you about for example, few gates, few multiplexers, few flip-flops given to you there is a lot of variations here lot of variability here so how am I going to get an efficient design out of this. I am going to get an efficient design I D flip-flops out of it by doing an operation called programming. So that way I can use it for different applications so I will have to have a programmability built into this so this device is called programmable devices.

After medium scale integrated circuits we will talk about large scale integrated circuits but large scale integrated circuits are generally programmable devices in which the hardware that is given to you can be put to different uses by doing a little bit of manipulation. Programming is just not sitting in front of computer and punching in things but programming is manipulating the circuitry inside connecting the circuitry inside to our best advantage for the output required that is called programming.

From the next lecture we will talk about first on decoders and then on LSI the programmable devices.