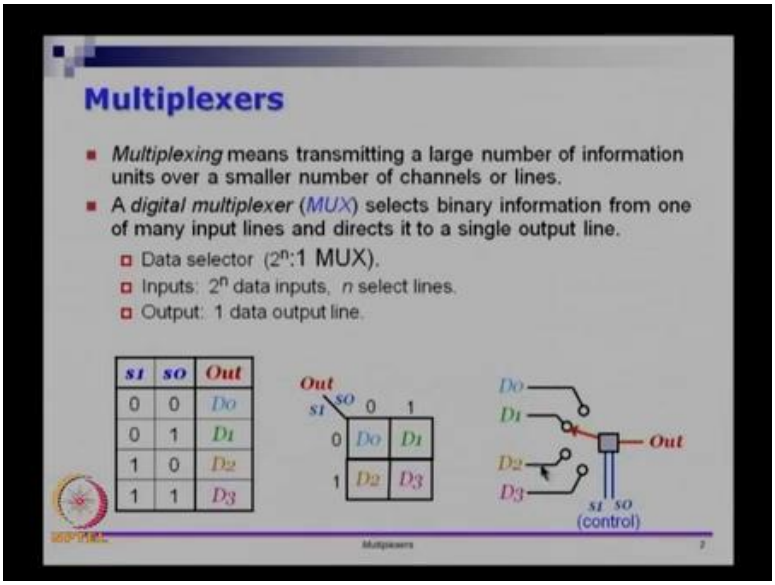
Digital Circuits and Systems
Prof. Shankar Balachandran
Department of Electrical Engineering
Indian Institute of Technology, Bombay
And
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 14
Multiplexers

Welcome to module 14. In module 13, I talked about basic blocks. So, I said there are 4 kinds of basic blocks in combination circuits namely decoders, encoders, multiplexers, and de multiplexers. In this module, we will see what Multiplexers are.

(Refer Slide Time: 00:35)



So, multiplexer in English means you are sharing something, so you might have seen movie multiplexers for instance, movie theaters which have multiplexers. So, the word multiplex means, many in one, that is what it essentially means. So, what will have when we say multiplexing is, you want to transmit a large number of information units on a small number of lines, this multiplexing is there in telephone lines and several other things.

In fact, I introduce a term multiplexer in a previous module where I said there are symbols, Boolean equations, and so on, I briefly mentioned this term multiplexer. A digital multiplexer selects binary information from one of the many input lines and it is

re directed to a single output line. So, imagine there are three or four of us in a neighborhood and there this only one telephone line that is going out of the neighborhood.

So, four of us if you want to connect to the telephone line, we will put what is called a multiplexer there, and based on what is given us a control to the multiplexer, exactly one of the telephone lines will be connected to the output. So, you can see a picture below, so imagine these are the 4 houses in the neighborhood and this is a multiplexer sitting there.

If an appropriate control is given, one of these 4 will be connect to the output and as long as the control is held at that, only one input is given to the output, the other inputs cannot be connected. If you want the other inputs to be connected, you have to change the control to something else. So, in this example, if s 1 is on and s naught is on, let us assume that this switch will switch to D 3, which means the output will start tracking D 3.

If s 1 is 1 and if s naught is 0, then output will start tracking D 2, this switch will come to this position and output will start tracking D 2 and so on. So, this is want we want from a multiplexer and if you want to put this in a table, this is the way to do it. So, you have s 1 and s naught, these are the control lines, so that there are 4 different control values that are possible namely 0 0, 0 1, 1 0, 1 1, and the output should follow either D naught, D 1, D 2 or D 3 depending on the input combinations.

So, remember D naught or D 1, D 2, D 3 themselves can be 0 or 1, we do not really care about whether D naught is 0 or 1, D 1 is 0 or 1 and so on. As long as the control is 0 0 in a single bit multiplexer, the output is a single bit, the inputs are all single bits at a time, there are 4 inputs in this example, but each one of them is a single bit and we do not care whether the bit is 0 or 1. All we care about is, based on the control line which the output should be connected to which input and which one should be detach from.

So, you connect to 1 and detach from 3 and if you notice at all points of time, the switch will be connected to one of them at least. In fact, exactly one of them, you cannot say I will not connect to any of them, so a multiplexer is not there. Because, you have only 4 combinations, of the 4 combinations multiplexer demands that one of the inputs is connected always and outside the 4 combinations, there is nothing else.

So, this is what is called a variable entered map, this is like a K map, only that we did not put D naught, D 1, D 2, D 3 as 4 difference columns and wrote output based on that.

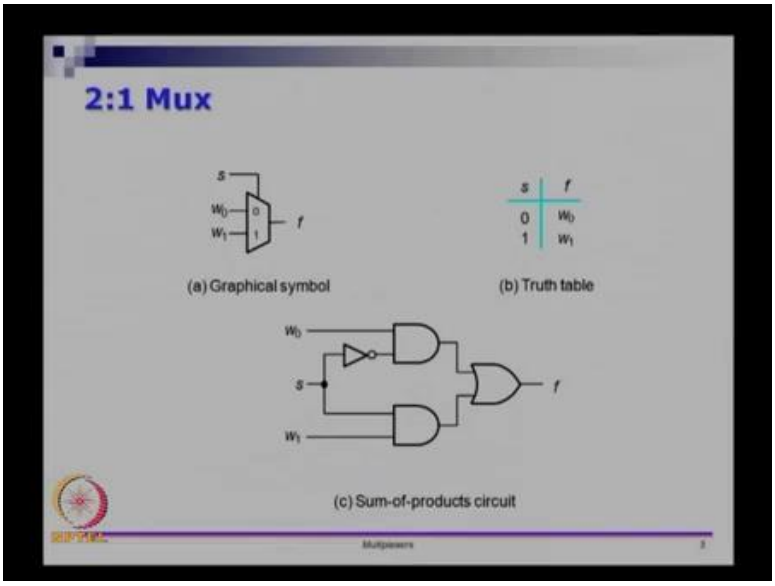
Since, we do not really care about, what value D naught or D 1 takes, we just put D naught here directly. So, a way to do that in a K map is as follows. So, when 0 0 is... So, this is the way I should interpret this K map. If s 1 is 0 and s naught is 0, output is the same as D naught.

If s 1 is 0 and if s naught is 1, output should follow D 1 and so on, that is the way we should read this table. So, we do not care about 0's or 1's within the K map here, so this is a 4 to 1 multiplexer, it has 4 inputs it has 1 output.

In some sense, this is resembling like what we had in an encoder. It had 2 power n inputs, but then an encoder will have, if you have 2 power n inputs, it will have n outputs. In a multiplexer, you have 2 power n inputs, you have only one output. So, that is the difference between encoder and a multiplexer. So, it has only one output in this one, in multiplexers. So, we will call this a data selector, there are 2 power n data inputs and we have n select lines, as before we will not count the control lines, when we say it is a multiplexer.

So, if I say 4 to 1 multiplexer, it is 4 different inputs connected to 1 output. The control lines, the count of the control lines is not taken into account there. So, there are 4 inputs, n select lines and 1 output line, we will call this a 2 power n is to 1 multiplexer.

(Refer Slide Time: 05:48)



So, let us see the simplest of the multiplexers, the simplest of them is a 2 to 1 multiplexer. So, here we have a 2 to 1 multiplexer, this is the symbol I showed the symbol earlier also. This is the symbol, we are going to use for multiplexer. So, the

control line will come on this edge of the trapezoid. So, you see that there is a long edge, there is a short edge, which are parallel and there are these two crooked edges.

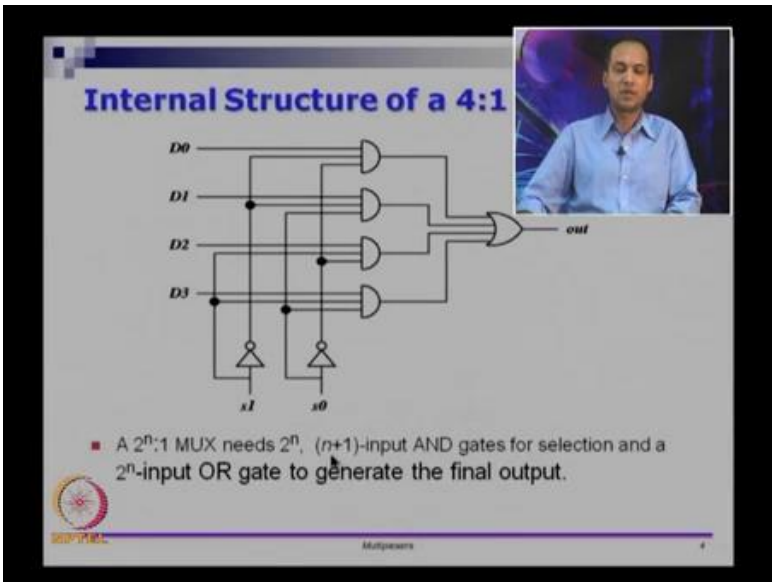
So, a control line will come to one of these slanted edges, the longer side is supposed to be corresponding to the inputs and the shortest side of the trapezoid is supposed to correspond to the output. So, we can see that this if you imagine this as a black box, it is 3 inputs w naught, w 1 and s and there is one output f. And what we want is, if the select line is 0, we want w naught to be transfer to f, if the select line is 1, we want w 1 to be transfer to f.

So, we will put the symbol 0 and 1 to indicate which one is supposed to be for s equal to 0 and which one is supposed to be for s equal to 1. This table makes it clear, when s is 0, we want w naught to be transferred to f and then s is 1, we want w 1 to be transferred to f. So, these two symbols here tell us, for which value of s this one should be passed to the output. So, this is the graphical symbol and this is the truth table and this is the implementation in sum of products form.

So, let us interpret this in an English sentence. If s is off, then you can see that w naught, if s is off, then the inverter will have it 1 here, 1 and w naught will be w naught here. If s is 1, this will be 0, this will be 1 and you will have 1 into w 1 here. So, if s is 1, you will have w 1 here, if s is 0, you will have w naught here and your logically OR in these two.

So, as an expression what we have is, s you can see that here also. So, s we can write f in terms of s w naught and w 1, f function f is s bar into w naught plus s into w 1. So, I will again repeat this, f is s bar into w naught plus s into w naught. We saw this equation earlier also, we will call this s as a select line and w naught and w 1 are the data lines.

(Refer Slide Time: 08:24)



So, we will see a 4 to 1 mux here, the internal structure of a 4 to 1 mux looks like this. So, we have 2 inputs s 1 and s naught which are control inputs and we have the 4 data inputs D naught, D 1, D 2 and D 3 and what we are going to do is, we are going to take all the 4 combinations of s 1 and s naught. So, if you put the inverters here, if you put the two inverters here, at least s 1, s naught you will get 4 different combinations and you have one AND gate for each one of the data lines.

So, if you look at these two, they are coming from s 1 complement and s naught complement. If you look at these two lines here in the second AND gate, the two lines in the bottom of the second AND gate, you will see that s 1 is coming from off and s naught is coming from 1, you will see that here, see it is coming from 1. Then, for the last line, it is start from s 1 be on and s naught being on and this AND gate is adding with D naught, D 1, D 2 or D 3 and 4 of these are connected to us single 4 input OR gate and then you have a output.

So, essentially what this is doing is, you are if you go and look at the output, so it has 4 inputs here and 1 output here, it is a 4 to 1 multiplexer. So, a 4 to 1 multiplexer will have 2 select lines, 4 data lines and 1 output and because we have the select lines here, exactly one of these AND gates will be on, the other ones will give you 0. And whichever is on will pass on D naught to out, you can find that out by looking at...

So, try input s 1 equals 0 and s naught equals 1, see which of the gates is turning on, you will see that exactly one of the AND gates will turn on. And that AND gate will also

copy the corresponding input of the OR gate. So, OR gate will see three 0's and the data input one of these data inputs here. So, the data input are in the 0's is the data input itself that you will see in the output. So, this is the internal structure of a 4 to 1 mux.

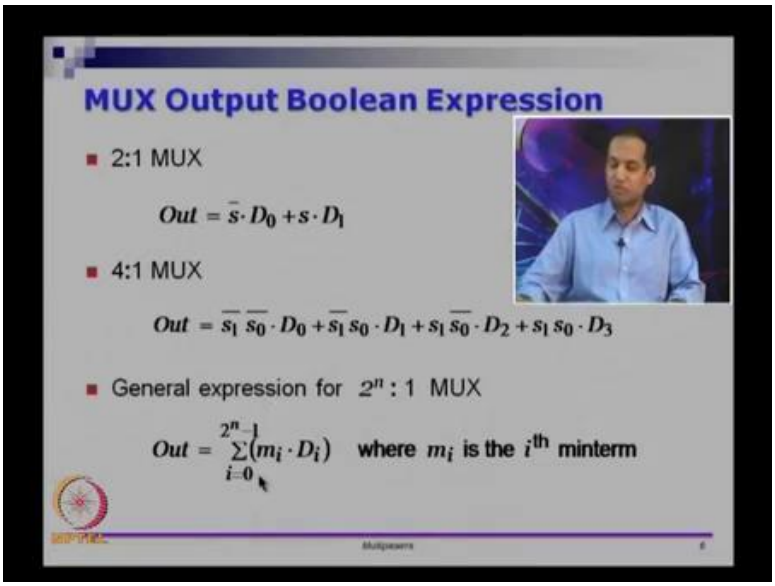
So, one thing that this circuit has is, it has a remarkable uncanny resemblance to the decoder that we saw earlier. So, we will come to that in a little while. So, a 2 power n to 1 mux needs 2 power n, n plus 1 input AND gates. So, we have... So, this is a 4 to 1 multiplexer, it requires n plus 1 input AND gates. How many of... So, it requires 2 power n is the count and n plus 1 is the number of inputs, it requires 4, 2 plus 1 or 3 input AND gates for selection and a single 2 power n input OR gate to generate the final output.

So, this is the description of a mux. Like I said, this circuit has a remarkable closeness to the decoders. So, this is the circuit for a mux and I am going to convert this to a decoder, let us see how to do this. The first thing I am going to do is, instead of the 4 data lines that we had earlier, I am going to connect all of them together and I am going to call that enable. And if I get rid of the OR gate, then I will have 4 output lines and this is actually a picture of a decoder.

A decoder is one in which you have a single enable line and you have 4 output lines for 2 input combinations, within the 2 input combinations we will have 4 outputs, this actually a decoder. So, if you want to get a decoder from a mux, you have to connect all the inputs together and somehow if you are able to remove the OR gate and keep the 4 input lines to OR gate as it is, then you will get a decoder.

The reverse if you want to get a mux from a decoder, then you put a single OR gate here and do not connect the same line to all the inputs, instead leave them as 4 separate inputs. So, that will give you a mux, so you can see the picture. So, when you remove the OR gate, this is a decoder with 4 separate input lines and with the OR gate, it is a multiplexer. So, they are very closely connected. So, let us look at how to use muxes or what are the expressions for muxes. So, I went I said that earlier, now I will say explicitly.

(Refer Slide Time: 13:03)



A 2 to 1 mux has this expression, if I have 1 select line call s, if s is off, then you will see that this expression is... So, if s is off, then s is 0, then 0 bar is 1, 1 into D naught; however, this term will become 0. So, 1 and D naught is D naught, if s is 1 this term will be 0, this term will be D 1, so that is the expression for a 2 to 1 mux. So, now I want you to go and think a little bit about, what the expression for a 4 to 1 muxes. I am going to show that in a little while, but you go and think about it.

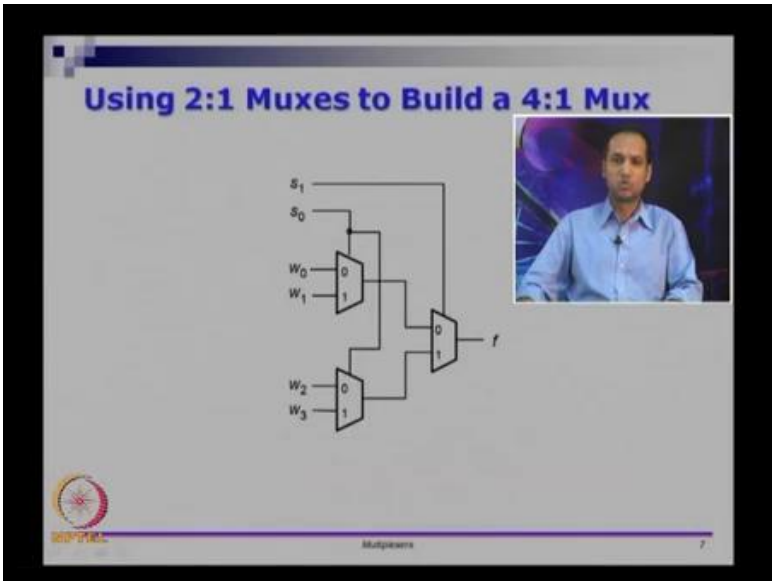
So, it should have 2 select lines, s 1 and s naught, 4 data lines D naught, D 1, D 2, D 3 and 1 single output line called out. We take a while to think about what the expression is going to be, so the expression is as follows. If s 1 bar s naught bar corresponds to both of them being off, if both are off pass on D naught, if s 1 is off, s naught is on pass on D 1, if s 1 is on, s naught is off pass on D 2 and if both of them are on, pass on D 3.

So, you can see that the structure is very similar to what you had here. So, here you had 2 values or 2 combinations for a single input line s, here you have 4 possible combinations s 1, s naught both complemented and only one of them complemented and both not complemented. So, in the general expression is out, as a Boolean expression is the logical OR over 2 power n terms. Starting from 0 to 2 power n minus 1, m i which is the min term i and data i which is the ith line of data.

So, m i is the ith min term, you take the ith min term, hand it with data input i and you do it for all the 2 power n combinations, you put a single OR gate with 2 power n inputs, that will give you the output. This is the general expression for a 2 to 1 mux. So, go and

think about what the expression for a 3 to 1 mux is for instance, go and write it down and check whether it is correct or not. Now, let us see, how to do various things with a muxes. The first thing I am going to show as I did earlier with decoders is, I am going to use a smaller mux namely a 2 to 1 mux to build a larger mux, namely a 4 to 1 mux.

(Refer Slide Time: 15:21)



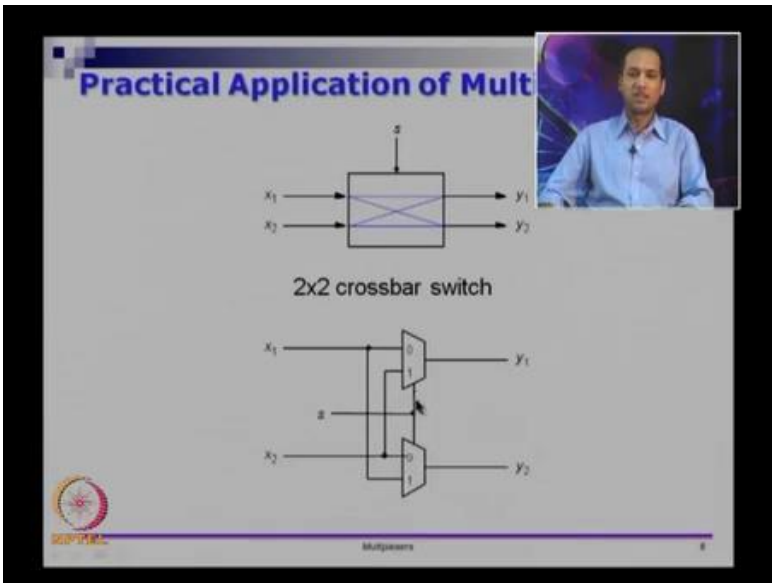
So, let us takes talk of the situation first. What does a 4 to 1 mux need? A 4 to 1 mux needs 2 control inputs, 4 data inputs and 1 output. So, if you imagine a box around this now, this circuit will take s 1 and s naught which are the 2 control inputs and it takes 4 data inputs w naught, w 1, w 2, w 3 and there is a single output f. So, you can think of it as a block box around it and we want to design a circuit like this.

So, to be able to do that, so this is a circuit which does it, so let us see how it is so. So, now, let us imagine that s 1 is on, if s 1 is on, whatever is in this line will be passed f and whatever is not in this line will not be. So, if s 1 is on, this line this mux is output, the one in the bottom it is output will be passed on to f. Now, let us see, what are the conditions under which these inputs are going to come?

So, where is this going to come, if s naught is 0 then w 2 will be on this line, if s naught is 1, then w 3 will be on this line that is the meaning of these 2 to 1 mux. It is taking s naught as a select line, if s naught is 0, then w 2 will be here, if s naught is 1 w 3 will be here. And if you want this to be passed on to here, s 1 should be 1. So, you can go from the left side to right side and see the conditions under which f gets various terms, if you want w naught to be passed on to f, we want s naught to be 0.

So, that this line will take w naught and we want s 1 to be 0, so that this line will be going to the output. So, we want both these to be 0 for w naught to go to the output. If you want w 3 to go to the output, both of them should be 1 and if you want let say w 1 to go to the output, you can see that s 1 should be 1. So, the w 1 comes here and s naught should be 0, so that it comes here. So, the combination you need for w 1 to go to f is s 1 is 1 and s naught is 0. So, this is the 4 to 1 mux, but internally we have 2 to 1 mux is used for that.

(Refer Slide Time: 17:45)



So, a practical application for a multiplexer is actually what is called a cross bar switch. So, let say there are two people on the other side of the telephone line, so there are two people you want to make phone calls and there are two people who can receive the phone calls and over is call in should be able to call both of them, these what we want. So, there are two people trying to make phone calls and there are two people who can receive them, we want to be able to make any one of them connect to any one of them.

So, such a circuit is called a cross bar switch, so cross bar switch is 1 in which any of the inputs can connect to any of the outputs, based on what is the select. So, we this is the very useful application, because you will see this in telephone which was an inside circuits and so on. The internal implementation of a cross bar switch is as follows, so you use 2 mux on the top and mux in the bottom and it takes one external line call the control line which is select and there are 2 inputs x 1 and x 2.

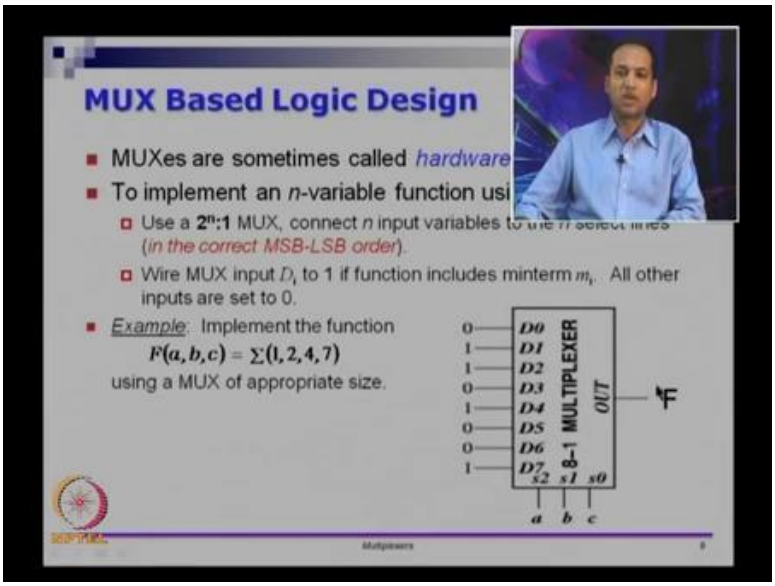
So, what we want is if s is 0, so let us look at what happens when s is 0, if s is 0, x 1 will

be passed on to y 1 and x 2 will be passed on to y 2. Because, if you put s equal to 0 look at which line is connected, x 2 is connected to the 0th pin and here s 1 is connected to the 0th pin of this mux. So, if s is 0, x 1 will be connected to y 1 and x 2 will be track by y 2; however, if s 1 is 1 then x 2 will be passed on to y 1 and x 1 will be passed on to y 2. So, this is simple circuit which actually does what is called a cross bar.

So, the name cross bar comes because of this cross connected fashion you have, you do not have lines going from x 1 to y 1 and x 2 to y 2. You also have this cross connection or x connection that is there, which is needed for this cross bar switch and this is a circuit which actually does the cross bar. So, if you want a cross bar like this, you need 2 muxes which can take care of it. So, one think I want you to go and think about this how to design a 4 cross 4 cross bar.

So, 4 cross 4 cross bar will have 4 inputs here and 4 outputs here, it should be able to connect any 4 to any 4 here and you should have an appropriate number of select lines. So, go and think about how many select lines you need for this, you also go and think about how to design that using a mux.

(Refer Slide Time: 20:27)



So, now let us move on to how to use mux as a logic circuit, so muxes are sometimes called hardware look up tables, we will see them why it is the carrying then I am called look up table, to implement a n variable function you can actually it take a 2 power n to 1 mux, which means it has a 2 power n data lines n select lines and 1 output that is a 2 power n is to 1 mux, you can implement any n variable function using a 2 to the n to 1

mux.

So, what will do is, we will connect to 2 to 1 mux the n input variables, so we will plug the truth table in some sense of the inputs, we will plug the input variables to the select lines and the output will be the corresponding output. So, I will show the picture in a little while, so you use a 2 power n is to 1 mux connect the n input variables to the n select lines, in the what is called the most significant bit to the least significant bit order.

So, I always mention this f of x, y, z means x is the first left most column, y is the next column, z is the next column and so on. So, if I write f of x, y, z then x is the most significant bit and z is the least significant bit. So, from left to right when you look at it whatever is the left most is the more significant and whatever is a right most is the least significant. So, that is how we do it decimals also, if I tell you number 756, then 7 is the most significant it has higher weight it carries a weight of 100 and 6 is the least significant it has weight 1.

So, left to right is what we have as more significant to least significant, so what we will do is, we will wire the mux in such a way that we will, so show you the circuit let us I will let us like a small example I will come to the bullet in little while. So, if you want to implement a function f of a comma b comma c which is sigma of 1, 2, 4 and 7 let see how to do that. So, I am going to take de multiplexer, so the first thing we notice is the min term is going up to min term 7, a min term 7 will happen only for a 3 input circuit. So, we have 3 inputs here a, b and c.

So, we give a, b and c as a select lines s 2, s 1 and s naught, so that is what I said. So, a, b c is in the left to right order and the select lines s 2 is more significant, then s 1 is more significant, then s naught in this order. And what we are going to do is let see the min terms here, so min term 1 is here. So, you take d 1 input put 1 there, min term 2 we want, so take d 2 input put 1 there, we want min term 4 is here, so take d 4 input put a 1 and 7 take d 7 input put a 1.

So, if we go and look at this, this is the truth table for the circuit, so the truth table will actually have this combination 0 1 1 0, 1 0 0 1. So, you plug in the truth table in the 2 power n is to 1 multiplexer as the input lines various. So, these are all constants there are no variables here, these are all constant the actual a, b, c which are the input conditions you plug them here and the output is the expected output.

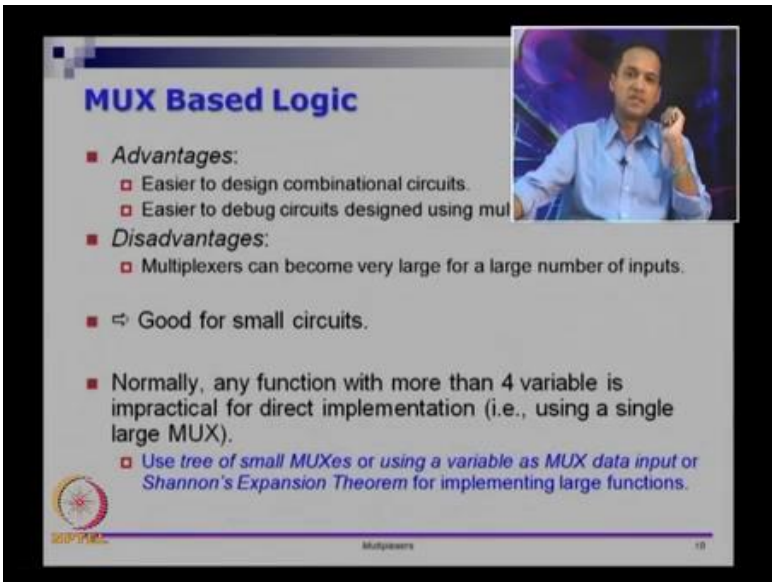
So, you can imagine how this is going to work, so what... So, let say I have a, b, c as 1 0

1, if a is 1, b is 0, c is 1 let suppose to pass on the d 5th line to the output, but what is the d 5th line have, it has 0 at the input. So, f will be 0, so you can see that 5 is not in the expression CSOP here. So, when this is 1, this is 0 and this is 1, D 5 which as input 0 will be pass to the output. If I want let say, if I put a equal to 0, b equal to 1 and c equal to 1, so that corresponds to condition 0 1 1 or 3.

So, D 3 should be passed on to output or 0 will be passed on to the output. So, this is the mux base circuit for a truth table. So, why the call this a look up table is as follows, you take any generic mux, you put the inputs in the order that you want, find out the truth table that you want for the function, plug it in the set off data inputs you are done that is it no more design, you do not have to be anything more. All you have to do is, take the 3 variables in this case and take the function that you want stick it is truth table at the inputs you are done.

So, if I gave you the truth table let say I gave a truth table 1, 3, 5, 7 instead of 1, 2, 4, 7 let say I give you 1, 3, 5, 7 all you have to do is make D 1 on D 3 on D 5 on and D 7 on, the rest of them 0 that is does the circuit that you want it is as simple as that.

(Refer Slide Time: 25:24)

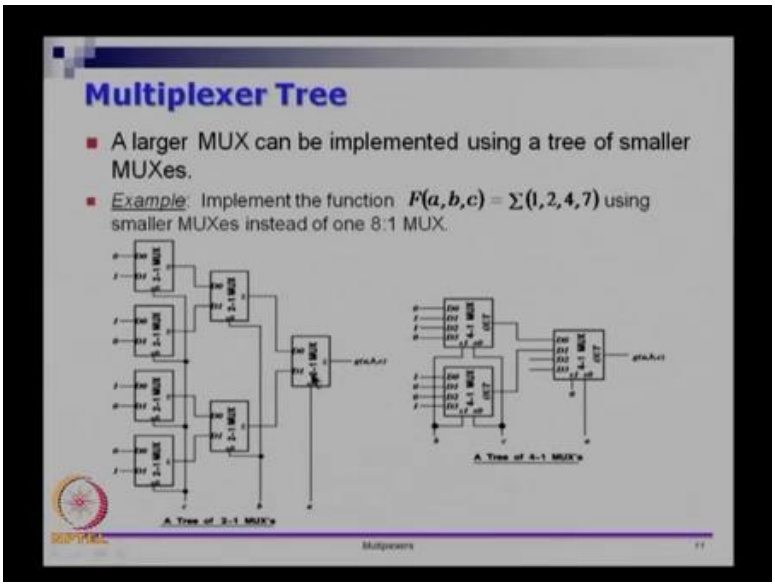


So, this is called mux based logic design, a mux based logic the advantage is that it is very easy to design the circuit. Because, all you have to do is find out the truth table and stick it in, it is easier to debug the circuits using multiplexers, if you want a find out of this any fault in at there is no minimization nothing, you have a circuit and if you put the truth table you are done. So, there is you will not go wrong too many times.

However it comes with a disadvantage, the problem is that muxes can become very, very large for a large number of inputs. So, in this case we did not do any minimization ((Refer Time: 25:56)). So, there is some structure to the problem here, we did not see any structure we did not minimize anything we just struck the truth table here. So, the circuit make it very complicated, we may actually spend more to do simple functions even.

So, mux space design is for very, very small circuits it is not good for larger circuits. So, normally if you have a function which as more than 4 variables, then you do not usually use a mux space think. So, you cannot use a single multiplexer for this, you can use a combination of multiplexers, but a single multiplexer becomes prohibitively expensive the movement the number of inputs goes to more than 4.

(Refer Slide Time: 26:35)



So, what it typically do is, you use what is called a multiplexer tree, so a multiplexer tree is one in which you have smaller muxes instead of the larger mux. So, I will show a pick picture for this circuit, so f of a comma b comma c actually demands an 8 to 1 mux, but instead of using a 8 to 1 mux, you can use either 2 to 1 mux is like this or 4 to 1 mux is like this to get the corresponding output.

So, what we have done is, we have arranged a tree of multiplexers here, so there are 4 2 to 1 mux is followed by 2 to 1 mux followed by 1 2 to 1 mux and we strict the truth table as we had for previous case. But, now these are all smaller multiplexers, rather than one huge monolithic 8 to 1 multiplexer. Similarly, we have 3 4 to 1 multiplexers as suppose to 1 huge 8 to 1 multiplexer.

So, I will not get into the details of this, it is possible to take smaller multiplexers and design larger multiplexers I have already showed you that. So, this one is essentially expanding on the idea for 8 inputs. So, this brings me to the end of this module, so what we have seen in this module is, this very basic circuit called a multiplexer and we saw what comes with a multiplexer, the multiplexer is a very useful logic block.

In the next module we will see some of the interesting property is of a multiplexer we will also see various base in which we can use a multiplexer. So, most important we in the next lecture, we will see the notion of universality of a multiplexer. So, you multiplexer is actually a universal gate, we will see why it is show in the next class, you will also see some important examples which show how to use a multiplexer and we end up with one important theorem called Shannon's theorem in the next module.

So, thank you for now and see you in a little while.