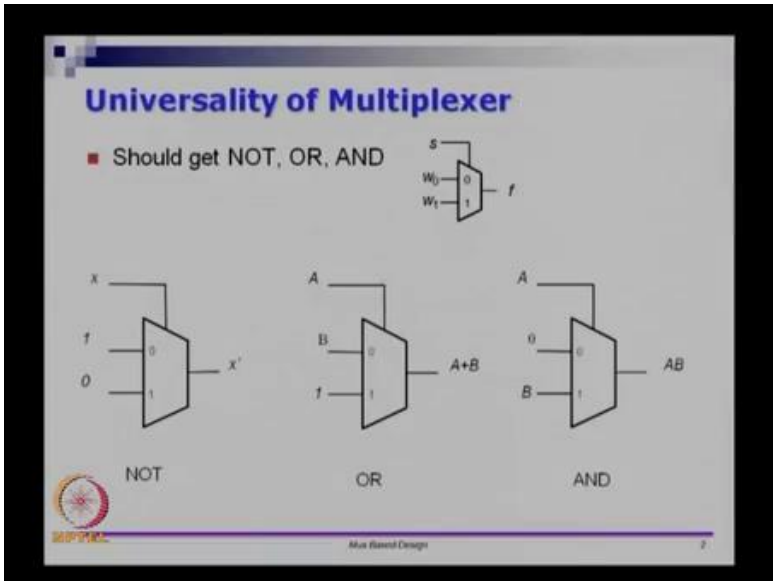
Digital Circuits and Systems
Prof. Shankar Balachandran
Department of Electrical Engineering
Indian Institute of Technology, Bombay
And
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 15
Multiplexer Based Design

Welcome to module 15. In this module we will look at how to use multiplexers to do circuit design. So, we have seen what a multiplexer is in the previous module, we will see how to use multiplexers. So, remember I mention that multiplexer is a basic blog for designing digital circuits. So, we will see how to use a multiplexer for dual circuit design. So, the first thing that I want to establish and this is not something that you see in many textbooks, that the multiplexer is actually a universal gate.

(Refer Slide Time: 00:48)



So, if it is a universal gate remember that we should be able to get the three basic operations - NOT, OR and AND buy using the multiplexer and nothing else. So, we earlier showed how NAND is a universal gate. So, in this slide, I want to show you, how multiplexer is also universal gate. So, essentially if you are given in an infinite supply of multiplexers that is enough to implement basic combinational circuits.

Let us see how, let us start with the first one, let us say NOT. I want to see, there are

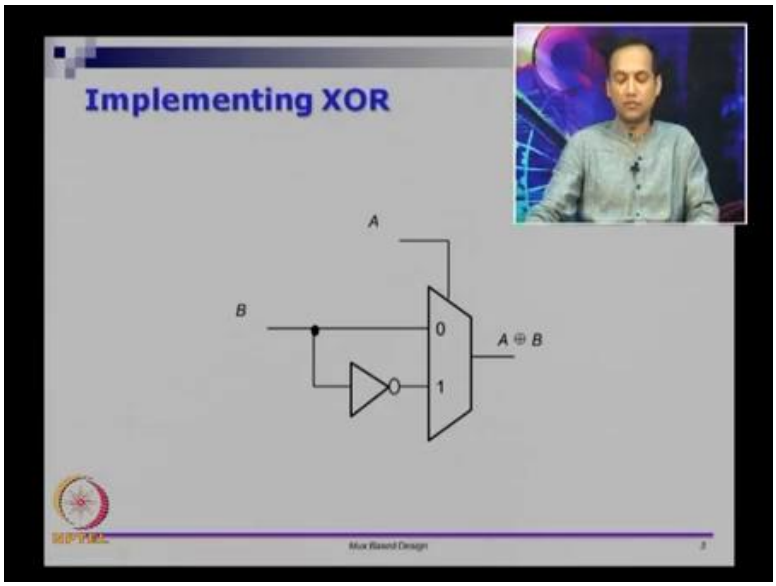
three inputs; however, NOT gate will have only one input. We have to somehow control the inputs in such a way that the function F that we get is just a logical complement of the input. So, one way in which this can be done, there is as follows. So, whatever input that you want, let us say x, that has been complemented, you give that as the select line.

And when x is 0, you put 1 at the input, for x equal to 1, you put 0 at the input and this will give you x bar at the output. So, you can see why... So, the multiplexers function is x bar and 1 plus x and 0. So, if x is 0, the output will be 1, if x is 1, the output will be 0. So, that is what we want in the NOT gate. So, we got the NOT ready, now let us see how to do OR. To do logical OR, you place let us say I want OR of A and B. So, you place one of the inputs as the select line and in the one pin, you put 1.

So, when A is 1 without checking B, you can say that A plus B should 1. So, when A is 1, the output should be 1, when A is 0 the output should be B. So, this is the logical functionality for our OR gate and finally for AND, when A is 0 without checking B, you can say that the output must be 0 and if A is 1, the output should follow B. So, this is the logical circuit for an AND gate.

So, you can see that if you are able to implement all the basic three basic operations, NOT, OR, and AND, so therefore we can call a multiplexer also has a universal gate. So, the notion of the gate need not be just the basic circuits, sometimes gate is also used for slightly more complicated blogs like these. So, multiplexer is a universal circuit which can do all the three basic operations.

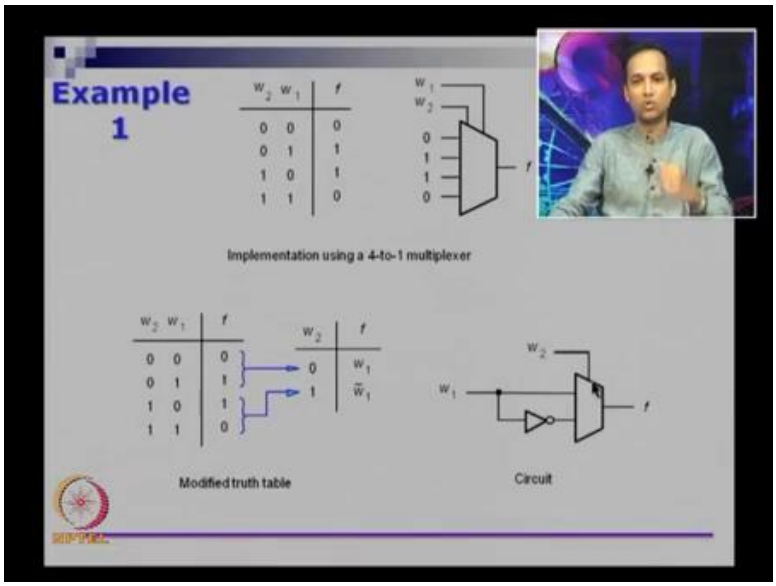
(Refer Slide Time: 03:33)



So, another useful thing to know is, how to implement XOR gate using multiplexer. So, this requires slightly more circuitry. In this case, let us say A and B are the two inputs. So, we know that if A is 1, then we want to give B bar as the output, if A is 0, we want to give B as the input, that is the definition of XOR, so A XOR B is A B bar plus A bar B. So, when it is A bar, which is indicate by the 0 here, when it is A bar give B and when it is A give B bar. So, this is the way in which you can implement XOR.

So, I would recommend that you go and think about, whether you can do this without this extra inverter, because I am using another gate here an inverter. Can you try and see, if you can implement an XOR gate without using inverters by using only 1, 2 to 1 mux. Just can you look at connections that you can do to A, B and so on, to see if you can get A XOR B here, you should not use any extra gates, NOR any extra multiplexers, see if that is possible.

(Refer Slide Time: 04:35)



So, now let us do a few design examples using multiplexers, let us say I am given this Boolean function here. So, w 2 and w 1 and I have 0 1 1 0 1. So, you can recognize that this is actually a multiplexer. So, when 0 and 1 are different, then you have 1, when they are same, the output is 0. So, the implementation let us say I ask you to do it using a 4 to 1 multiplexer. So, there are 4 inputs and there is 1 output. So, there are 4 input combinations possible and there is 1 output, let us see how to do this using a 4 to 1 mux.

So, one way to do that is take your, so remember a 4 to 1 multiplexer has 4 input lines and 2 select lines. So, in the two select lines, I am going to give w 2 and w 1 as the select inputs and I am going to take the truth table from here and plug it in here directly, 0 1 1 0 I am plugging it in directly. So, this top most is for the 0 0 pin, there is 0 1 pin, 1 0 pin and 1 1 pin. So, there are 4 pins, input pins that the multiplexers will have a 4 to 1 multiplexer.

So, I take the truth table and plug it in directly. So, this is supposed to realize this XNOR function here is XOR function here directly. Now, you can also realize this using smaller muxes, so I have already showed that example earlier. So, let us do this little bit, so what I am going to do is, I am going to take the truth table, so this truth table is the same as this, this truth table here, only that it here we used a 4 to 1 mux to do this.

I want to see, if we can you use a 2 to 1 mux, because 2 to 1 mux is actually cheaper in terms of the number of basic transistors in it than a 4 to 1 mux. So, let us see how to do

that, so if you take this 1, you look at the combination here. So, you look at 0 and 1 here, if you go and look at w 2 and w 1, if you do not care about w 2 for now, then you can look at as when w 2 is 0, f is w 1, you can see that here, this 0 1 is the same as the 0 1 here.

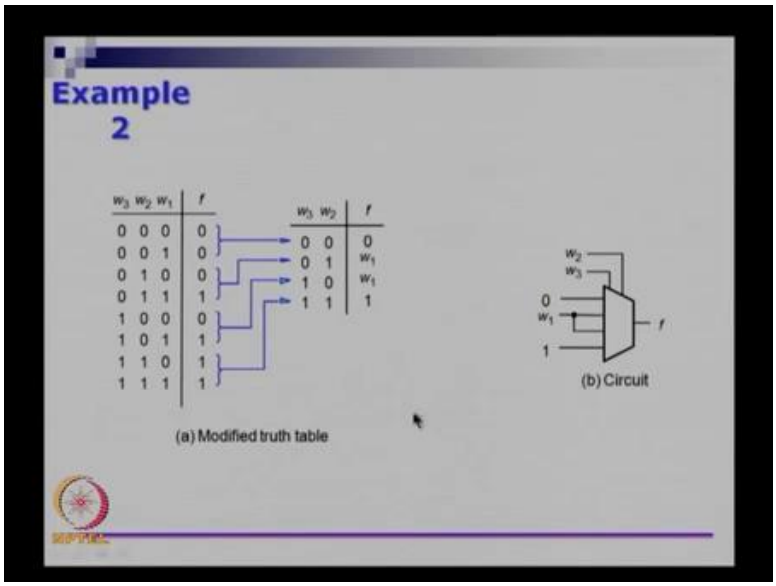
So, when w 2 is 0 which is the top half of the table, then f is w 1, when w 2 is 1, which is the bottom half of the table, f is w 1 complement. So, you can take this and write it in a different form. So, when w 2 is 0, f is following w 1, when w 2 is 1, f must follow w 1 complement. So, the moment it comes to this, now you can go and write this as a 2 to 1 mux, so this is a modified truth table.

So, in a truth table you have only 0 and 1 entries, whereas in this truth table, you actually have variables. So, this is also called a variable entered map, where you do not put all the variables only on the input side, even the function is written in terms of functions of other variables. So, w 2 is 1 variable, w 1 is other input to the same circuit. But, f is not just a vector of 0's and 1's, f is some function of w 1 also. So, this is called a variable entered map and this is what we had earlier.

So, we had w 2 here in the previous slide I showed it as A and B, here it is w 2 and w 1. So, we give one of the inputs as the select line, for 0 you give w 1 as the input, for 1 when w 2 is 1, you get w 1 bar as the input and that gives you f. So, this is how I came up with the circuit for the XOR gate in the previous slide. So, we took the table and modified it to a variable entered map, once you get the variable entered map, there are only two values here.

These two values, you can just put it as a select line, select line this only one line which can take two values, so you we have that. However, if you had this, if you have a four entry K map, so you have four rows here, these four rows cannot be expressed only by one combination of w 2 for that we needed w 2 and w 1. That is what we have in the picture in the top.

(Refer Slide Time: 08:45)



So, now let us look at another example, so in this example, we have a truth table which is on 3 inputs. So, we have 3 inputs w 3, w 2 and w 1 and there is a truth table that is given here. So, let us not worry about, what the truth table is suppose to be doing is just look at this idea now. So, what we are going to do now is, we are going to see, if we can use a multiplexer which is smaller in size.

So, if you directly take this vector and give it as an input to a 8 to 1 mux with select three lines w 3, w 2 and w 1 and give this vector as the input at the different pins, then you can realize this using a 8 to 1 mux. So, already mention this, a function on n variables can be implemented directly using a 2 power n to 1 mux. Now, I am going to do something slightly different. So, what I am going to do is, I am going to take two rows at a time. So, if you go and look at the first two rows, we can see that w 3 and w 2 are the same, only w 1 is changing.

So, what I am going to do is, I am going to write a variable entered map, where f is either 0 1 or a function of w 1. So, what are the possible functions of w 1? You can have w 1 itself or w 1 complement. So, what we will have is in this column, we will have 0, 1 w 1 or w 1 bar. So, I am going to take two rows at a time, so let us take these two rows, in these two rows w 3 and w 2 are 0.

So, that is what we have here, w 3 and w 2 are 0 and if you look at f, f is 0 independent of w 1 and w 1 is 0, f is 0, when w 1 is 1, f is still 0. So, no matter what w 1 is f seems to

be 0, I will be write that here, f is 0, so this is independent of w 1, when the input is 0 0. Then, you look at the next two rows 0 1 0 and 0 0 1; in this case, w 3 is 0 and w 2 is 1. So, at least these two are remaining same, but if you go and look at f, f is not 0 like here, it is actually changing with respect to w 1.

So, f takes the value of w 1 in these two rows, so you can go and write that as w 1 here. Similarly, f takes the value of w 1 for these two rows, because we are keeping w 3 and w 2 constant, if you change w 1 from 0 to 1, f will change from 0 to 1. So, that we represent here, if w 3 and w 2 are 1 0, then f is w 1 and if w 3, w 2 are both 1's, then without looking at w 1, the output is 1.

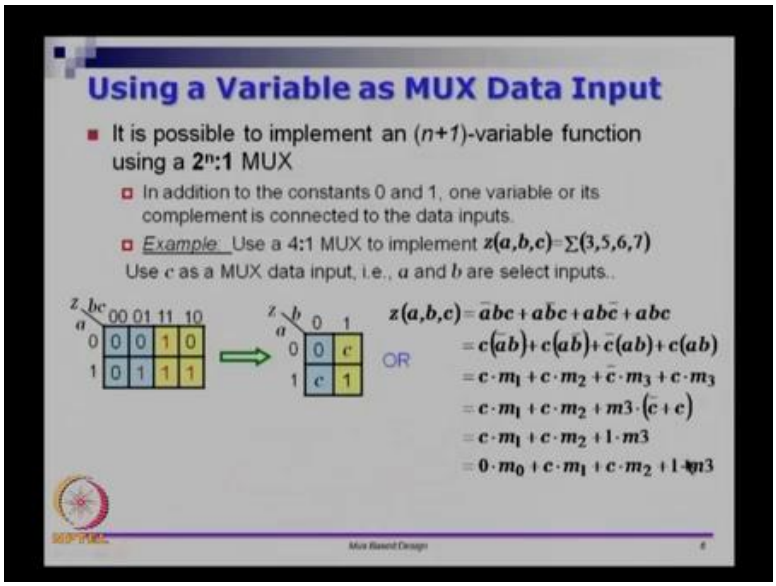
So, even if w 1 is 0 or 1 it does not a matter output is 1, we can write that tag. Once, you get something like this, now you go back and look at this. So, we have now put that in a 4 to 1 mux, what we have done is, now this has four rows, so I can use a 4 to 1 mux and implement this, a 4 to 1 mux has two select lines. So, I will give w 3 as this pin and w 2 as the right pin.

So, we are assuming that the most significant bit is in the left most side and the least significant bit is in the right most side and in the vector here, we have 0 w 1 which is connecting to both the 0 1 input and the 1 0 input and finally, 1. So, it is this truth table entries here directly plugged in. So, now, if you want to check, what the value of f is, when let us say w 3 and w 2 are both 1, then you go and look at w 3 w 2 both 1; that is this pin, then the output is 1.

You can see that in the table, if w 3 and w 2 are both 1, so the output is 1. So, you can check, whether the circuit is correct by plugging in different values of w 1, w 2 and w 3 and go back to the table and check, whether it is correct or not. So, you have to keep doing some of these things, till you get comfortable with the whole idea of translation and so on.

So, with experience, you will get it, but always it is good to once you get a circuit plug in the values and check, whatever you wanted in the table is it coming through the circuit or not. So, now there are a few other things that you can do, I will show you how to use a variable has a mux data input.

(Refer Slide Time: 12:59)



So, in general, you can implement n plus 1 variable function using a 2 power n is to 1 mux. So, in the previous slide I said with n inputs, you need a 2 power n is to 1 mux to implemented, but actually, you do not to need a 2 power n is to 1 mux. So, for m plus 1 variable, 2 power n is to 1 mux is enough.

So, you take n of these variables and give them as select lines and in the inputs, you put 1 0 and the function of the other variable that you have eliminated. So, when you go from n plus 1 variable to a 2 power n is to 1 mux, the input vector can be just function of one of the variables, we will see a quick example later. So, in addition to the constant 0 and 1, you will use one variable either in the complemented or the uncomplemented form.

So, let us see this example z of a, b, c is 3, 5, 6, 7 and what I am going to do is, I am going to use c as a mux data input and a and b are select inputs and if I do that I should be able to implement this using of 4 is to 1 mux. If I directly do this, I need to 8 is to 1 mux, because it is on 3 inputs, I will need 8 different input lines, let us see how to do this. So, let us look at various combinations, this is the truth table.

So, we have the truth table from a on one side and b, c on the other side and you can see that min term 3, min term 5, min term 6 and min term 7 are all 1's, we have that. Once, we had this, let us see how to get c out of this, so you go and look at combination at a time, you look at these two cells here. So, in this case, c is changing from 0 to 1, but the output is not changing, the output is remaining at 0.

Even the c changes from 0 to 1, the output is 0, so you put the 0 there directly. Here, the input if c changes from 1 to 0, f changes from 1 to 0, which means f is changing as c is changing, so you put c there also. In these two columns, when c changes from 0 to 1, f changes from 0 to 1, which means f is following c and in these two end cells, when c is changing from 1 to 0, f is remaining at 1, which means f is independent of c, so you have one here. So, this is again a variable entered map for 3 inputs.

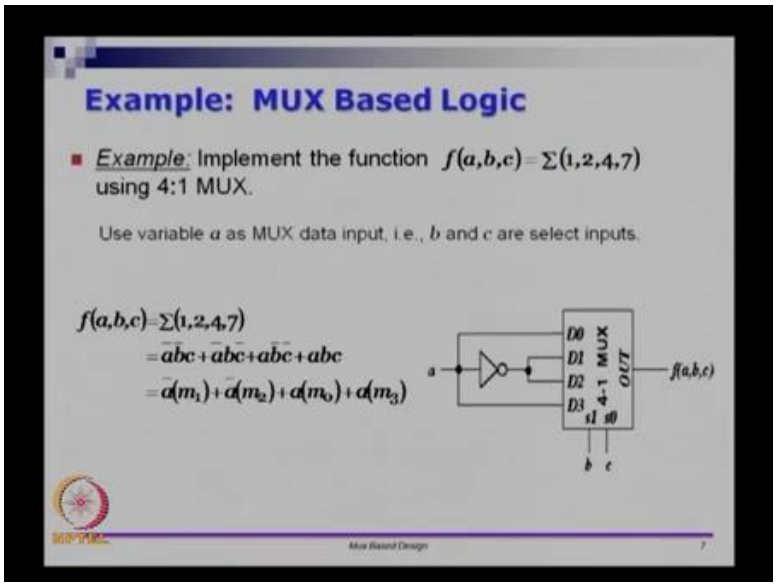
And once you have something like this, you can go and implement that using a truth table, using a mux. So, you can do this or you could have done something slightly more elaborate. So, I did it directly in the map here, you can also do this from basic Boolean functions. So, you go and look at this, we have a bar b c plus a b bar c plus a b c bar plus a b c, these are the terms 3, 5, 6 and 7 respectively. In each one of them, remember, because it is some min term, they are all in normal form.

So, you can take c out, you then c a bar b, a b bar, a b and so this must have been, so this is a b bar which is 1 0, then you have a b and a b. So, this is from 6 and this is from 7, then A bar b in terms of a 2 input set up, you can thing about this as min term 1, a b bar would be min term 2 and a b would be min term 3. Then, you go and simplify it further, we can write as c into m 1 plus c into m 2, then we have c bar plus c into m 3, which is same as 1 into m 3.

So, we have the function c and m 1, c and m 2, 1 and m 3, you can rewrite that as 0 and m naught, c and m 1, c and m 2, 1 and m 3. So, now, if you go and take this and put it into a 4 to 1 mux, so your A and B will be the inputs to the select lines and the 4 input bit is to contain 0 c c and 1, this is exactly what we saw on the previous thing. So, we saw 0, w 1, w 1 and 1, so these are my a b c instead of w 3, w 2 and w 1; this is a b c.

So, what I just showed is just a way of showing doing this whole thing using a 4 is to 1. So, you give 0 c c 1 as inputs and you a and b as the select lines and you get z as a function of a a b c. So, I am trying to show the same thing done in various different ways.

(Refer Slide Time: 17:50)

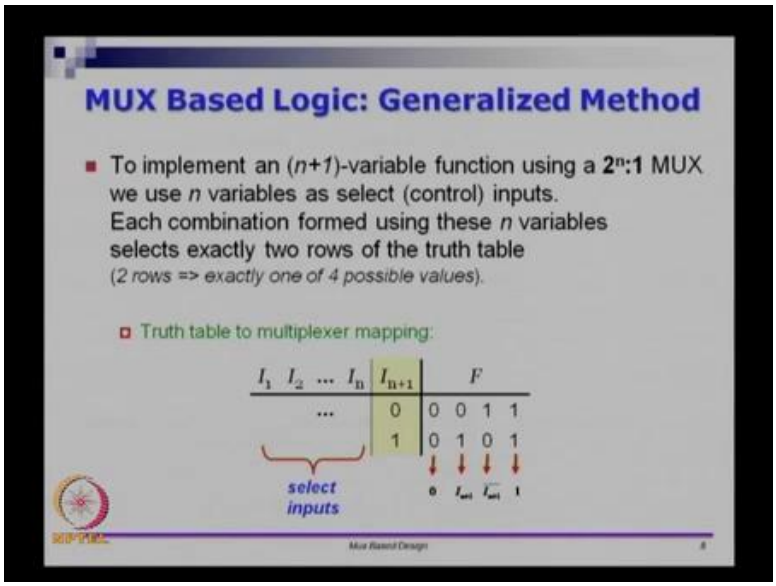


Let see another example, f of a b c is sigma of 1, 2, 4, 7; I want to do this using the 4 is to 1 mux. So, in this case, I am going to use variable a as the mux input and b and c will be the select inputs. So, I write it as sigma of 1, 2, 4, 7; these are the terms and I can take things out a bar into m 1. So, b bar c is equivalent to m 1 of a two variable function, b c bar is m 2 as a variable function, b bar c bar is m 0 of a two variable function, b c is m 3 of a two variable function.

So, now, you have in terms of two variable things m 1, m 2, m naught m 3 and there are inputs a, a bar and so on. Now, you can implement this using a 4 is to 1 mux, you give b and c as the inputs, for input 0 and input 3, a should go un complemented. So, the zeroth min and the third pin get un complemented versions of a, for minterm 1 and 2, suppose to get a bar, so you put a inverter here.

So, a bar goes to d 1 and d 2 inputs. So, you can go and check, whether that this is implementing the Boolean function that we needed. So, it is relatively straight forward. So, this way of doing, we will see something called Shannon's expansion at the end of the module, what we are doing is, what is called Shannon's expansion, we will see that the end of this module.

(Refer Slide Time: 19:12)

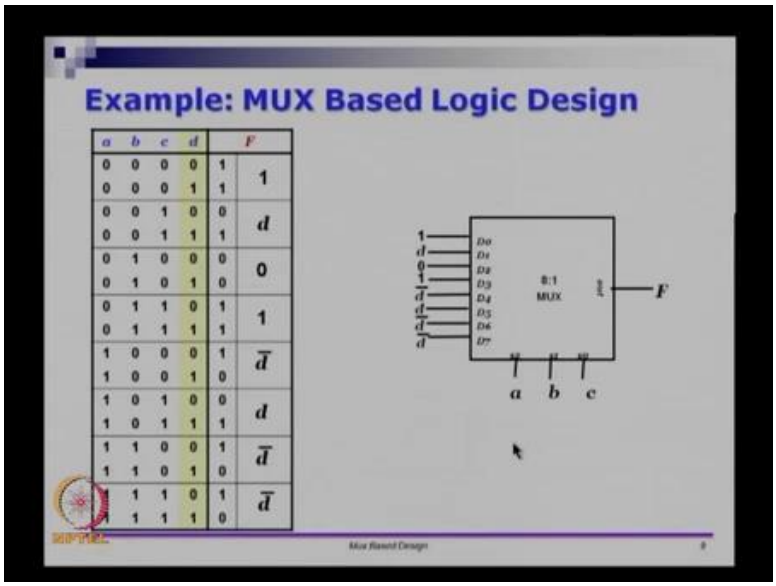


So, in general, what you have to do is, if you are given a n plus 1 variable function and if you are last implement that is using a 2 power n is to 1 mux of the n plus 1 inputs, you take n of them and give a select lines. And each combination form will contain some of these things will either have 0 1 or the one variable that you are eliminated or the un complemented version of the variable that you eliminated. So, each combination of these n variables will select exactly two rows from the truth table.

So, then you do a truth table to multiplexer mapping. So, what you do is, let us assume that we have the inputs I 1, I 2, I n up to I n plus 1 and the function is f. Then, what you can do is, you go and look at two rows at a time, you take these as select inputs and you take two consecutive rows of I n plus 1. So, whenever 0 1, if it is 0 0, then f is 0 it iself, if you say that is the 0 1, this is 0 1, then f is the same as I n plus 1. If it gets to 1 0, then it is I n plus 1 complement or if it remains at 1 1, which at 1.

So, I am not saying that this is the 4 output function, if f is the single output function and if it takes the value 0 0 for these two rows, then you put a 0. If you 0 1, you put I n plus 1, if it is 1 0, you put I n plus 1 complement, if it is 1 1, you put 1 and it do this for every pair of two rows. You take two rows at a time starting from the top and you do this for every pair of two rows at a time and you will have to do this 2 power n times. So, you will have 2 power n vector, the output which is in terms of 0 1 and I n plus 1, that is the general way to do this.

(Refer Slide Time: 20:57)



So, let us do that as a specific example now, I have given you some function here f and you want to implement this using a 8 to 1 mux. So, which a 4 input function which means we should be able to implement this using a 2 power 3 is to 1 mux or a 8 input mux. What we have going to do, we going to keep the left column as it is, a b c; we are going to keep it as it is.

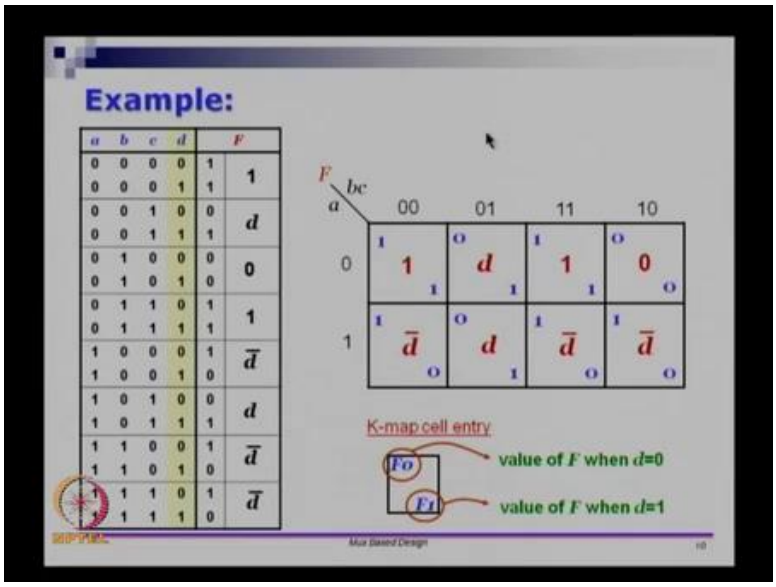
And I am going to take two rows at a time and see what the function F is. So, when d is 0 or 1, it does not matter, f is remaining at 1. So, I can put 1 here, then this is 0 1 and this is 0 1, so this must be the same as d. Then, you have 0 1 and 0 0 here, so if a these two rows, even if d is which as f is at 0, so I will put 0 here, 1 1 will make a 1, 1 0 will make a de complement, 0 1 will make d and so on and we have a vector f which is in terms of d norm.

Now, you can get rid of this column d and get rid of this vector here, what you have is a b c and f in terms of 1 d d bar and 0. So, you take this 1 d 0 1, d bar d d bar d d bar, you plug that here directly, 1 d 0 1 d bar d d bar d you put that in the appropriate pins here. So, d naught input should correspond to 1, the d 1 input should correspond to d and so on and the three select lines are a b and c, you give them in the same order from the highest bit to the lowest bit, that is the order that we are taken.

So, this is one way which we can do circuit is using muxes. So, I am going to show now how to do this in a k map itself. So, let see this simplified form here I have used 1 d d d

bar and 0, let see another way to think about the same problem.

(Refer Slide Time: 22:40)



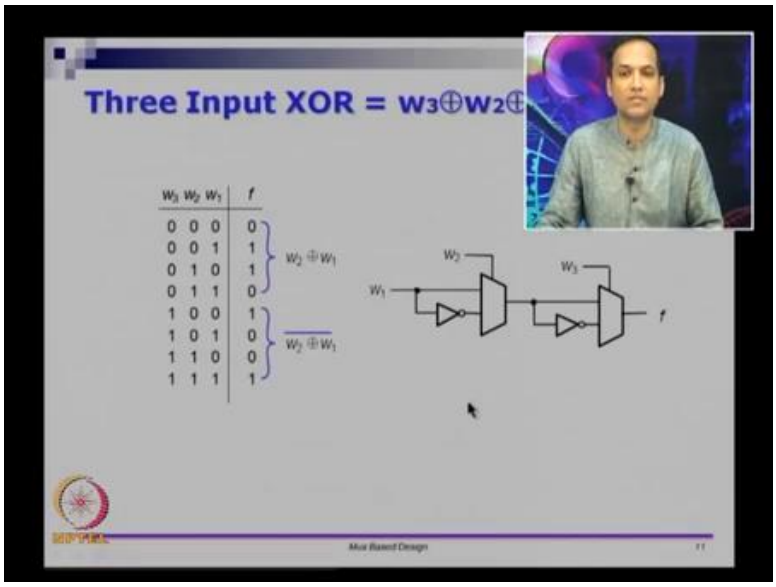
So, what are going to is, instead of a four variable k map, I am going to use a three variable k map. So, the three variable k map, I am using only a b and c, I am not going to use d as an input and what I am going to do is, take two rows at a time and I am going do something different. So, in the left top corner of every cell, I am going to put the value of F, when d equal to 0 for that cell and in the bottom, I am going to put the value of F, when d equal to 1, but corresponding to the combination a b c.

So, when a is 0, b is 0 and c is 0, for d equal to 0, whatever it is, I am going to put it the left top, when d equal to 1, I am going to put it at the right bottom. So, that will be 1 and 1, so these are the two values. So, this is for d equal to 0, this is for d equal to 1, I am go and do the same thing for the other cells. So, when a is 0, b is 0 and c is 1, it is 0 and 1 here, I put that in the diagonal here, then for 1 0, it is 0 0.

So, I put that in the diagonal here and for 1 1 for is for this combination for a equal to 0, b equal to 1 and c equal to 1, that is this set of rows and I take 1 1 here put that in the diagonal, I can do that for all the other cells also. Now, what we can do is we go and look at this here, wherever you have 1 1, you put a 1 and wherever you have a 0, 1 you put at d, wherever you have a 0 0, you put a 0. And so 1 1 is 1, wherever you have a 1 0, you put a d bar and you do this for all of them.

This table is just a rearrangement of the table that we had here, if he did not have the d column and that this vector here, it would have been a two variable or three variable k map, three variable k map would end up like this. So, this is another way in which you can do this.

(Refer Slide Time: 24:45)



So, now, I am going to take you to a through another other few examples. So, at this point, I want you to acquire some one basic skill. So, one basic skill that is needed for you to do a lot of logic design is this. If you are given a k map of 2 inputs, you should be able to recognize, whether the truth table given us for AND, OR, XOR and XNOR immediately.

So, if you tell you that, it is a two variable k map, if I draw a two variable truth table for you, by just looking at the truth table, you should try and guess, whether it is AND, OR, XOR or XNOR, these four you for able to write it is, it will be come in quite handy several times. So, I am going to use one of those tricks for this one. So, go and acquired this skill, you keeps staring at it is, it is very easy to actually look at two variable AND, two variable OR, two variable XOR and XNOR, you can recognize them immediately, actually it is very easy to pick up.

So, now, I am going to do something similar are this truth table here. So, I am going to do something for a three input XOR, the three input XOR function is defined as w 3 XOR w 2, XOR w 1. We already know what the definition of XOR is, we are taking 3

inputs and we have XOR operators between them. So, I am going to do something slightly different what I did so far.

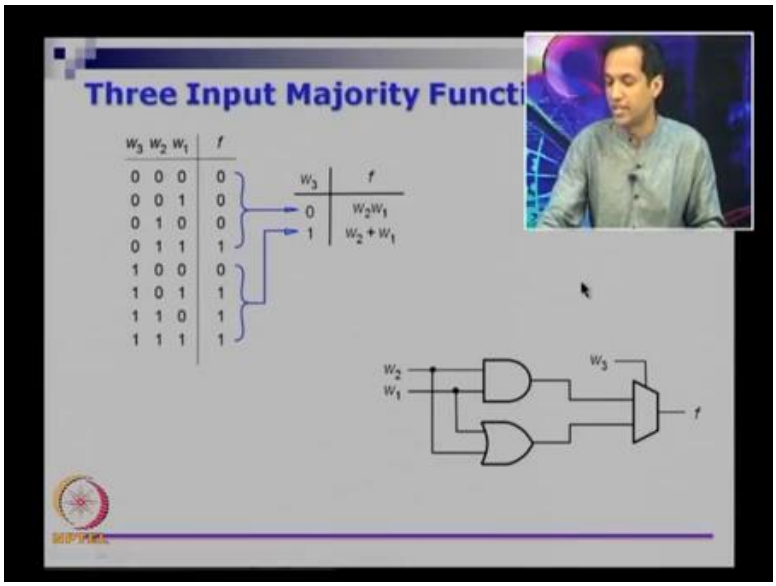
So, I am going to take four rows at a time, instead of two rows at a time, I would I can take four rows take at a time and if you go look at these top four rows, you go and see what is remaining constant, w 3 is held at 0. Whereas, w 2 and w 1 are all changing, there are different combinations of w 2 and w 1, we have here. So, what I am going to do is, I am going to write f in terms of w 2 and w 1.

So, if you go and look at the vector here 0 1 1 0 of 4 inputs, you can immediately say that, it is actually XOR of which is an XOR truth table, 0 1 1 0 is the XOR of two inputs and what are the two inputs here in this case, w 2, w 1 are the ones at a changing. So, these four terms correspond to w 2 XOR w 1, except that, it is for w 3 and w 3 complement.

So, essentially these four entries are w 3 bar into OR, AND w 2 XOR w 1. If you go and look at these four rows here, again w 3 is held constant w 2 and w 1 are so if you look at only this part, that is like a two input truth table. This truth table you have 1 0 0 1; that is actually for XNOR. So, this is w 2 XOR, w 1 complement. Now, we can take this and write it up, so what I am going to do is, I am going to take w 3. So, look at this, this part, look at this, this part.

So, when w 3 is 0, I want w 2 XOR w 1 and w 3 is 1, I want w 2 XOR w 1 complement. So, if I have a mechanism by which can realize w 2 XOR w 1, I can then do it from backwards. When, w 3 is 1, I will given input which is w 2 XOR w 1, otherwise, I ill give it is complement. So, this part is taking care of the truth table. However, we still need w 2 XOR w 1 and we know that, this is the way in which we can do w 2 XOR w 1, showed this earlier. For XOR of 2 inputs, you give w 2 XOR w 1. So, this circuit as a hole is doing w 3 XOR w 2, XOR w 1.

(Refer Slide Time: 28:45)



Let us take another function here; I am going to call this the three input majority function. So, we have 3 inputs and what f is doing is taking the majority of the inputs. So, you look at the inputs here, if all the inputs are 0, the majority is 0, if all the inputs are 1, the majority is 1. Otherwise, of the 3, let say one of them is 1 value and 2 of them are the different value, whatever is majority that will be the output.

So, you take this row here 0 1 1, there are a majority 1's and minority 0. So, the output is a 1 and if you have in this row, there are 2 0's and a 1. So, majority is 0 here and so on, this is called a three input majority function. So, to see, how we can design this using a mux, you go and take these top four rows, the top four rows that we combine, we can see that is actually a truth table for AND. Only the last one is 1, the other three are 0, so you can write this as w 2 and w 1 and we take these four rows, if you group them, this is a truth table for a 2input OR, so this case w 2 or w 1.

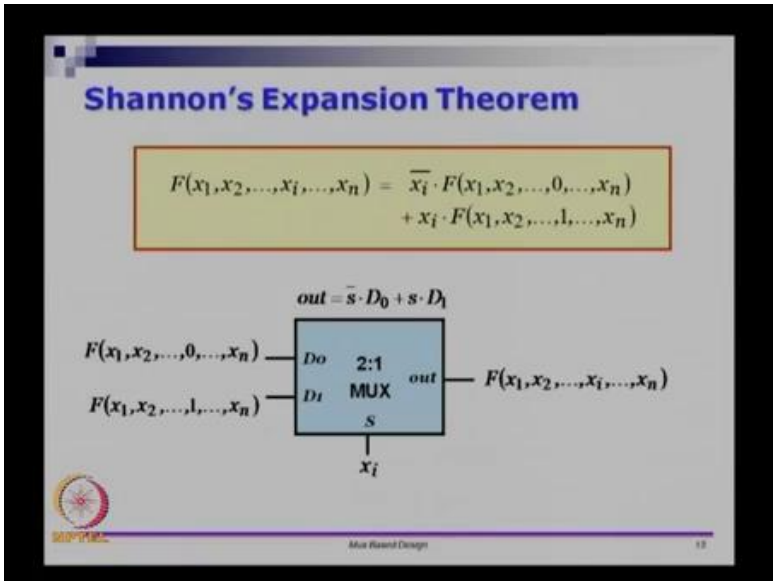
So, when w 3 is 0, f is w 2 and w 1 and w 3 is 1, f is w 2 or w 1. So, let see how to implements this using a mux. So, in this case, w 3 is enumerated, so I will implement that using a 2 to 1 mux. So, I give w 3 as the select line for a 2 to 1 mux, when it is 0, I want to give w 2 and w 1 as the input, you can see that here w 2 and w 1 is going as the input here and w 3 is 1, you want to give w 1 or w 3 as the input. So, this function f is using a mux and two discrete gates and implement circuit.

So, the these exercise is I have been telling implement using only NAND and

implements using only mux and so on, there is nothing really holly about using only mux and what not. So, depends on what gates you have to disposal. So, if you have 2 to 1 muxes, you can implement the AND using 2 to 1 mux, OR using 2 to 1 mux and so on. But, what if you give one AND gate, one OR gate and one multiplexer, so this is an engineering problem.

So, you have to somehow take the problem decompose that in such a way that you can use the resources that are available and realize the circuit. So, what I am trying to show here is, you do not have to use 2 to 1 muxes always to do this. This example if I had NAND or and the 2 to 1 mux, it is enough to design this circuit. I do not have to ask for 3 2 to 1 muxes or 1 4 is to 1 muxes and so on, I do not need it.

(Refer Slide Time: 31:19)



So, now, I will bring you to this famous expansion theorem call the Shannon's expansion theorem. So, Shannon's expansion theorem is very useful. So, in all the things that I have done so far, I actually use Shannon's expansion without explicitly saying that you. So, let us look at what the theorem says, f of n variables as it is any logical function on n variables x 1, x 2 so on up to x i up to x n, so this is the function of n variables.

Then, I should be able to plug x y, x i out, it which can be any of the x n's. So, in this case we have pick some x i out. So, it could be x 1, x 2, x 3, x 5, x 100, it could be any of them you can write f as a composition of two functions. So, you have x i complement and f of x 1 to x n plus x i and f of x 1, x 2 up to x n; except that wherever it is x i as a

variable you substitute the constants 0 and 1.

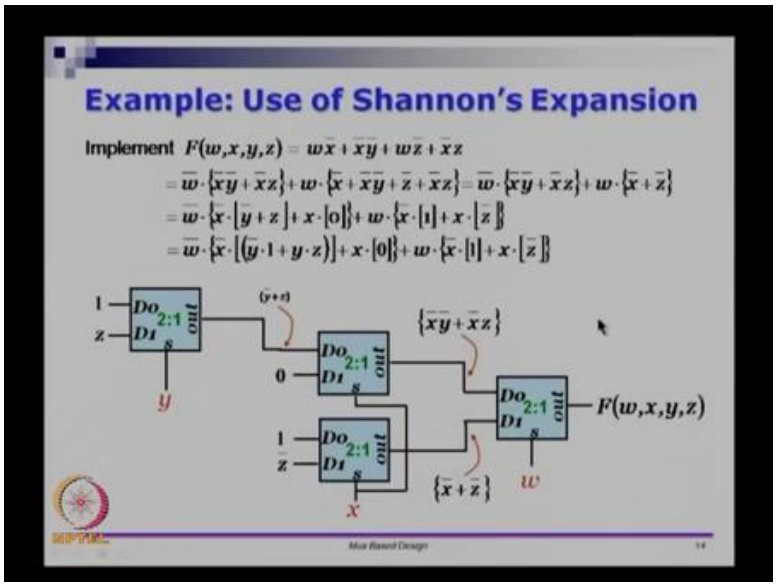
So, you can see, it is a very shuffle thing, but watch out. So, here it is a function on n variables x 1 to x n, this term has not a function on n variables, it is a function on n minus 1 variables and a constant 0. Similarly, this 1 is a function on n minus 1 variables and a constant 1. So, we call this the 0 cofactor of f and this is the one cofactor of f with respect to x i.

So, you take this 0 cofactor of x, f with respect to x i and it with the x i complement, you take the one cofactor of f with respect to x i and with it x i; logically or these two Shannon's expansion says, you can write this in this is this is form. Why is this interesting? So, one way to think about this is, if you do not think in terms of a mux, if I have select bar into something plus select into something, if you can see the structure, it is very similar.

This is, if exercise the select line, then select bar in to sum f of sum n minus 1 input function plus select in to f of sum n minus 1 input function. So, that is why, it is interesting. So, if you if you write it in this form, then you can write the output as select bar into d not plus select into d 1; that is a 2 to 1 mux, what you can do is, you put x i as the select line and it put f of x 1, x 2 so on with the 0 substitute for x i in x n.

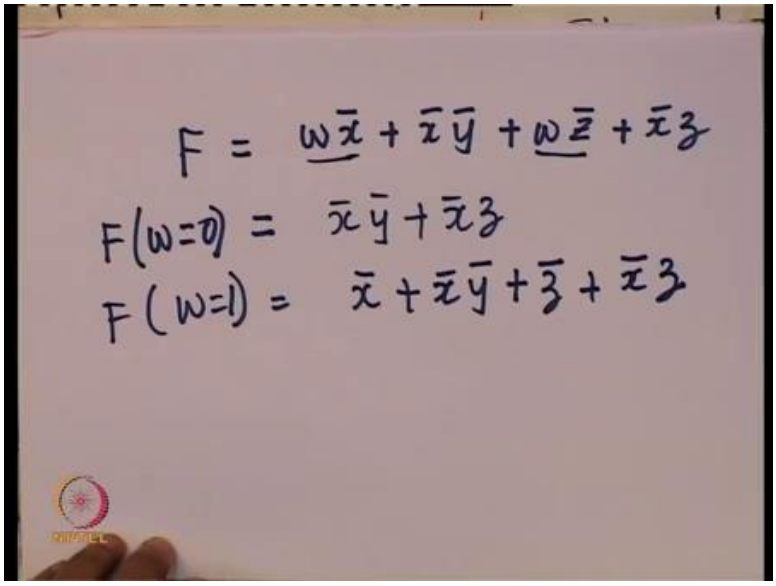
And f of x 1, x 2 so on up to and 1 substituting for x i and x n give that as your input d naught and d 1 and that will give you the function F of x 1, x 2 up to x i and x n. So, this is the very interesting thing and this is the very fundamental theorem, it is very useful in taking a circuit, taking a Boolean function and decomposing that in terms of multiplexers.

(Refer Slide Time: 34:11)



We will see a quick example, so let say I have a function w x y z; I want to write in terms of Shannon's expansion. The first thing I am going to do is, I am going to accumulate all the w bar terms together and all the w terms together, what do we have a w bar. So, you have this term x bar z is suppose to be a w bar x bar z plus w x bar z. So, you have that and this x bar y bar also is a w bar x bar y and the w x bar y, so you have that. So, you can see this, you put w equal to 0, so let me do this on paper here.

(Refer Slide Time: 34:50)



So, I have this function F, I have return that here, I will evaluate the function at w equal

to 0. So, this is my zeroth co factor, when w equal to 0. So, this term is 0 and this term is 0. So, it is x bar, y bar plus x bar z, when w equals 1, you put 1 here, so this is x bar, the second term will be there, it is it is x bar, y bar. Third term will be a z bar, 4th term will be x bar.

So, w bar in to x bar y bar plus x bar z, so this for when w equal to 0 and this is for when w equals 1, you can simplify both of them, x bar y bar plus x bar z remains as it is, this 1 can be simplified further as x bar plus z bar. So, now you have w bar into function when f when w is 0 plus w into the function, when w is 1. So, you can do then this form, you give w as the select line, you put x bar y bar at plus x bar z that the d naught input x bar plus z bar of the d 1 input.

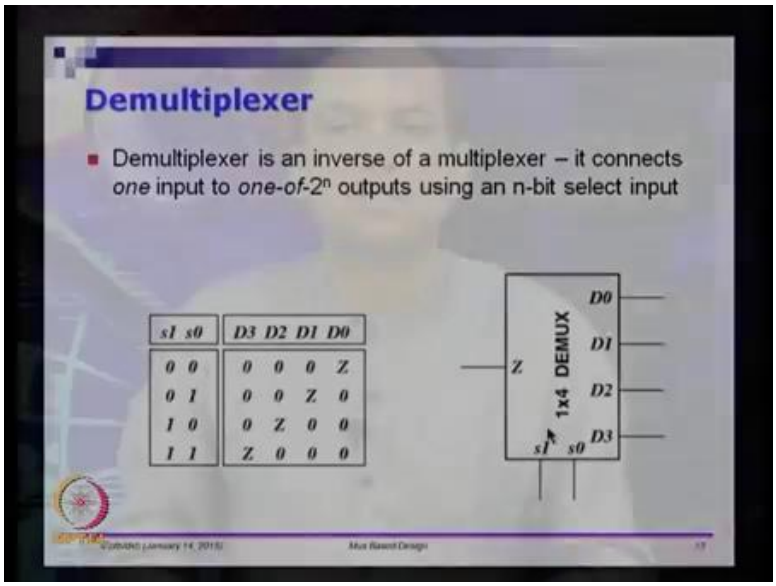
Now, you have to implement these functions also, no buddies going to give you these things directly, you want to implement this further. So, I want to break it down in terms a x and y and so on. So, this I am going to write it in terms of x first. So, I take x out, when x is 0, it is y bar plus z, when is 1, this is 0, so that is what we have here. And for this part again I am going to do the same thing, when x is 0, then the out this function is this function is 1, when x is 1 the function is z bar, so that is what we have here.

So, these terms are taking care of one more level, so we going to put x as the select lines for these. So, you can that here x is coming as a select line to two muxes, this mux is implementing x bar plus y bar plus z and this mux is implementing x bar plus z bar and now, here we have z bar 1 0. So, all these are okay, but we have a two input function here y bar plus z, we can now break it down further and get something very similar.

So, now, if we go and look at what is given to the circuit, I am giving 1 z, y, z bar x and w these are all basic variables are there complements and some constants. Now, this circuit is taking only 1 x y z bar x, w and 0 as inputs, all these are easily derive on it. So, we have use Shannon's expansion and what we have done is, first removed w from it, then we are removed x from it and this from we have removed y from it.

So, this will be only a function of z and 1, this will be a functions of only y z and 1 and 0, here you will have function of x y z and 1 and 0, if maybe in some examples you may also get 1 and 0 of it. So, this is the use for Shannon's expansion, it is a very useful thing.

(Refer Slide Time: 38:15)



So, the final a block that is important for or combination circuit is, what is called the de multiplexer, it is inverse of a multiplexer. So, based on the select lines 0 0 0 1 1 0 or 1 1 it will take one single input and place that on one of the 4 outputs. So, if z is whatever input you give here, if you put 0 0, z will be copied to d naught, otherwise and be copied to d 1 or to d 2 or to d 3, the primary difference is all the other ones will get a 0. So, if 0 0 is given us a select line, z is going to go to d naught all the others will be given in as 0.

Similarly, when this is 1 0, then z will be given to d 2, all the others will be kept us 0. So, you can see that it has similarity. So, mux as in de multiplexer, de multiplexer has some similarity to what we have seen so earlier. So, we have seen encoders and decoders and we are now seen multiplexers and de multiplexers, there all related somehow. So, I suggest that you go and think about, what is this de multiplexer closes to.

So, it has 1 input; it has 4 outputs and 2 extra lines, go and think about, what is the basic block of seen before which it resembles come most. So, this brings me to the end of module 15 of this week 3. And in the next module, we are going to look at verilog and I am going to use some examples, that if you are seen so far and show you, how to write verilog code, so see you in a little while.

Thank you.