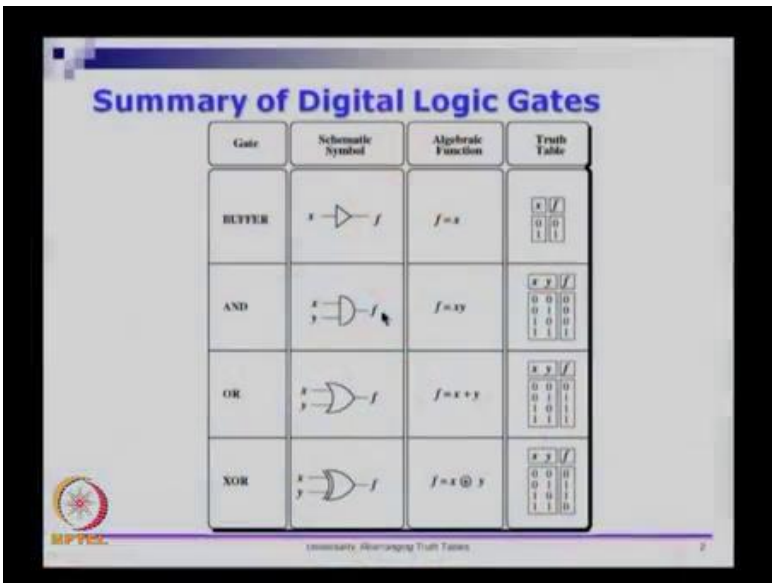
Digital Circuits and Systems
Prof. Shankar Balachandran
Department of Electrical Engineering
Indian Institute of Technology, Bombay
And
Department of Computer Science and Engineering
Indian Institute of Technology, Chennai

Module - 07
Universality, Rearranging Truth Tables

Hello all welcome to the 2nd week, I hope you enjoyed the lecture of the first week and I also hope that you have been doing the assignments. So, it is a small assignment for the 1st week, 14 questions are shown. So, take that seriously; however simple they are it is good that for you to go and try it and get it working and actually put your solutions back online, so that you can when you go to the 2nd week and so on, it will be better for you.

So, in this week we are going to see a few more topics are related to taking a Boolean function, how to specify it, how to minimize it and so on. The first of these modules is about universality and in the 2nd part of this module we will go and look at truth tables and what the truth tables really mean. So, let us start with the first one, so I want to give a quick summary of what the basic digital logic gates are.

(Refer Slide Time: 01:13)



So, what you are seeing here is the symbols that you are already aware of AND, OR and XOR and the dot is implicit, so that is removed here, the x plus y is OR and for XOR we use the symbol, where we have a plus and a circle around it. So, that is called XOR. So,

what we are seeing here is, you are seeing the name, the symbol that is used commonly, what is the algebraic function for it and the truth table for it. So, it is just a one slide summary of what we have seen already.

Now, this one what we are going to have is the same set of logic functions that we had last time in the previous slide, except that this is also an inversion at the output. So, if you notice this, this is the NOT gate, this is called the NAND gate. So, in the NAND gate we have AND followed by an inverter that is called NAND, OR followed by an inverter is called NOR and the XOR followed by inverter is called XNOR.

So, instead of having a symbol which is AND followed by the inverter, typically we use the short cut where we put a bubble. So, you see the circle here, this circle here is attached to the output of the gate. So, such AND gate followed by a small bubble which is interpreted as NAND or the logical complement of AND. Similarly, OR gate followed by a bubble the seen as complement of OR and bubble after XOR is XNOR. So, this is x XOR y, the complement and there is a short cut notation for XNOR also which is a dot and a circle around it.

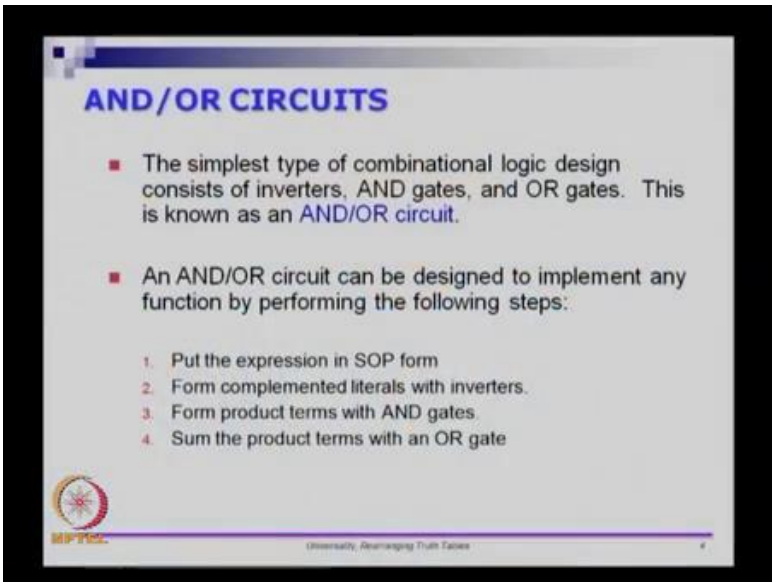
So, a plus and a circle around it, it is called XOR and a dot and a circle around it is called XNOR. We can see that XNOR is a Boolean function which actually checks equivalents of 2 inputs. So, you can see the truth table on the right side, if both the inputs are the same, so 0 0 and 1 1 then the output is 1, if the inputs are different then the output is 0. So, because of this XNOR is also, this is essentially checking equivalents of 2 input bits. So, x and y if they are equal then f is 1, if they are not equal then the output is 0.

So, this is the short cut notation, so just keep in mind that NAND is just not of AND, so you will see AND then an inversion or a small bubble, NOR is not of OR, so OR gate followed by a bubble and so on. Instead of drawing the OR gate, AND and an inverter explicitly will have a short cut notation where there is NOR followed by this. In fact, physically if you go and realize these things, sometimes we can implement these gates directly, the uncomplemented version namely the AND and OR actually needs an inverter after the NOR.

So, physical implementation in CMOS for instance it is a technology used to make chips. In CMOS technology, NAND and NOR or something that you realize with the basic transistors with 4 transistors for 2 inputs and if you want AND, you actually put an inverter following NOR or following NAND to get OR and NAND. Anyway, those are

details that we get in to later, as of now just remember the symbols. So, NOR is a gate and NAND is a gate and it is the inversion of the basic gates, that we have seen already.

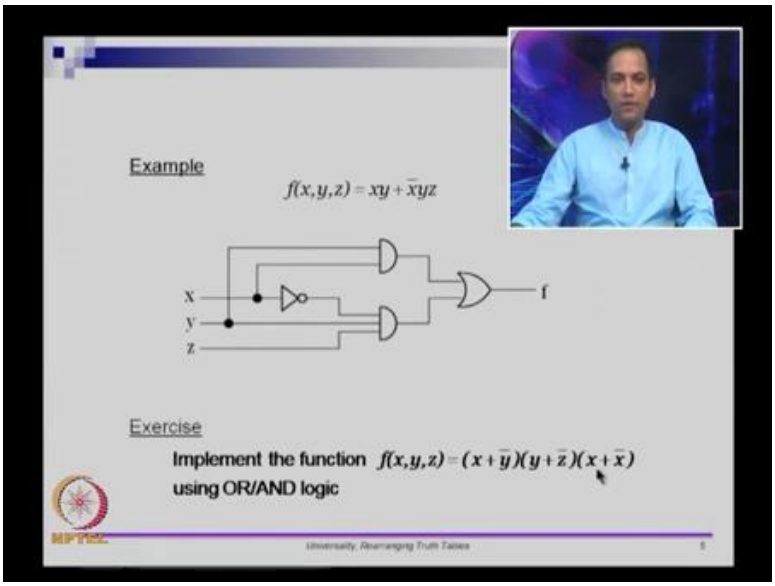
(Refer Slide Time: 04:50)



So, let us look at what is an AND/OR circuit. It is the basic kind of a circuit. It is a simplest combinational circuit that you can design. So, if you want to design a circuit using AND/OR an inverter, all you have to do is, you take the Boolean function and express that in the SOP form, let us say the Sum of the Product form and you may want complemented inputs also. So, if you want complemented inputs you take those literals by taking the variables and putting an inverter to get the complemented literals.

Then, you form the product terms using AND gates, because each is a product then you can actually use an AND gate to get the product. And the logical sum that you need for all of those is achieved using OR gate that is why it is called as AND OR circuit. So, the process is you start with an SOP, SOP has various product terms which may have complemented and uncomplemented literals. Wherever you have complemented literals use inverters and connect them using AND gate and you have various product terms now, connect them using an OR gate you have a AND OR circuit.

(Refer Slide Time: 05:50)

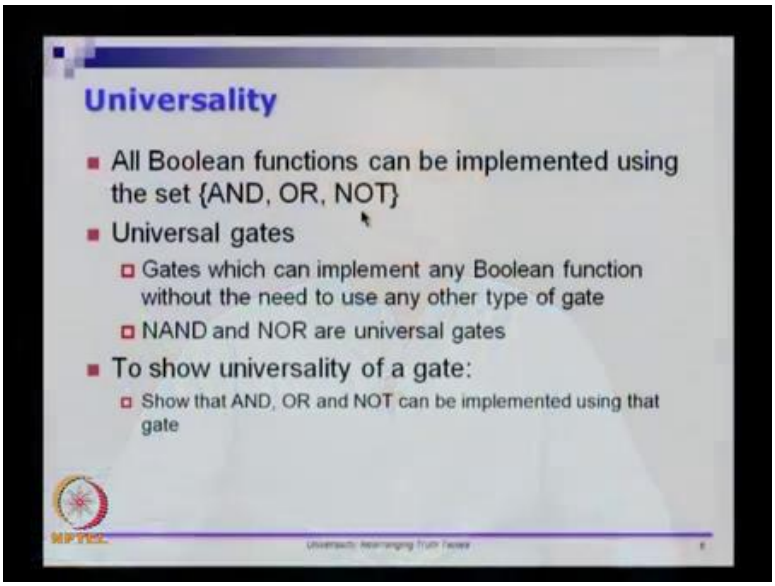


So, let us look at this small example here f of x, y, z is x y plus x bar into y z and this is already in the sum of products form. So, this is the product term, this is the product term and we need a sum for that it is already in the sum of product form. Then, if you want to realize it you can do it in this way. So, you look at this AND gate, this is doing x and y and if you look at this AND gate, it is taking x and complementing it. So, this line is x bar then you have y and z, so this is x bar y z. So, you have all the product terms ready by now.

And finally, there is a single OR gate which is doing this logical sum, so you have that here, so this is a basic AND OR circuit. So, I would like you to go and implement the function f of x, y, z as x plus y complement and y plus z complement and x plus x complement using OR AND logic. So, in OR AND logic what you do is these are all product terms. So, this is a product of sum terms, you take each of these sums and implement that using OR.

So, you may also want to use inverters before you do the logical OR and finally, you have 3 sum terms, you want a product of that you use the AND logic for that. So, go and try this out, it is very similar to doing this except that you will have first, you will have a set of OR gates and then you will have a AND gate. So, in this circuit what we have is a direct implementation of x y plus x bar y z. We did not try to simplify it or do anything else, this is directly writing it in this form. So, it is just to show you that AND OR circuit is easy to arrival, so now let us get into this concept of universality.

(Refer Slide Time: 07:50)

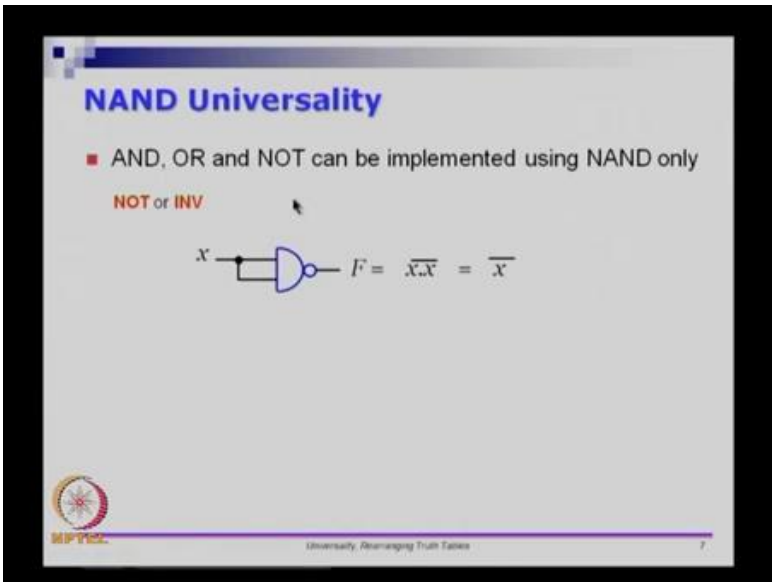


So, sometimes what happens is, let us say you know that basic AND, OR and NOT are the logical operations that are there in Boolean logic. So, therefore if you have only AND, OR and NOT gates, it is enough to implement any circuit technically. So, if I give you an infinite supply of AND gates, OR gates and NOT gates, you can implement any Boolean function, with only these things.

So, there is a notion of universality, where I call a gate as universal if it can implement any Boolean function without the need of any other type, any other type of the gate. So, if I look at AND gate alone, AND gate by itself is not universal, because you still you cannot implement OR operations and NOT operation by only using an NAND gate. However, NAND and NOR are called universal gates.

So, these are called universal, because even if you do not have any other type of gate, as long as you have just the NAND gate or just the NOR gates it is enough to implement any Boolean function. So, in general if you want to show if a gate is universal or not, all we have to show is using NAND alone, let us say we should be able to get AND, OR and NOT operations implemented. Similarly, just using NOR alone if you are able to get AND, OR and NOT operation implemented, then these two gates can be called as universal gates. So, again to retreat universal gate is one in which you can implement the three basic Boolean functions namely AND, OR and NOT without using any other type of gates.

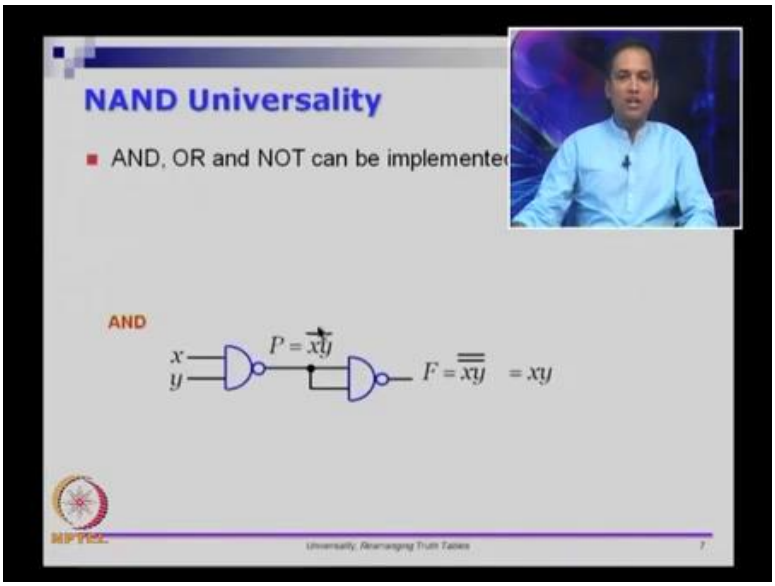
(Refer Slide Time: 09:20)



So, let us see how this can be done, will I will show you the example for how to do, how to show universality of the NAND gate. So, I want to show that AND, OR and NOT can all be implemented using only NAND. So, let us start with NOT, if you want to do a NOT, so this is an implementation where I use a single NAND gate and get complement of input. So, I have x which is given to both the inputs and what is the logical operation in terms of NAND, it takes the 2 inputs and AND set and complements the resulting product.

So, we have the product, in this case x is connected to both the inputs x and x the complement of that x and x is x itself. So, you can see this is x complement, so what this circuit is doing with this connection here, there is actually implementing inverter.

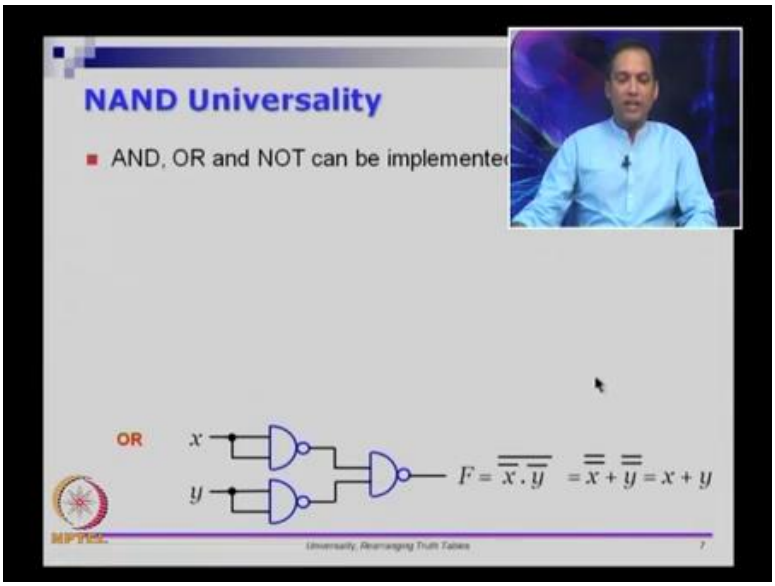
(Refer Slide Time: 10:14)



Then, let us look at AND, so AND we know is not of NAND, so NAND is not of AND, so essentially AND is the NOT of NAND also. So, if you want to get x and y, what we can then do is take NAND that gives me x and y complement. Now, you have the complemented version of the output that we are expecting, send the two inverter. So, f is the inputs complement, because remember this is the circuit for a NOT gate, we just saw it a while ago, this is a circuit for a NOT gate.

So, it takes whatever input is coming in and complements it and what is the input that is coming in for f, it is P which is x y complement. So, x y complements complement is... So, we know that a bar bar is a itself, so x y complement complement is x y itself. So, we have shown that NOT and AND can be done using only NAND.

(Refer Slide Time: 11:15)



Finally, if we want to show OR operation it is slightly more involved, but here it actually uses Demorgan's theorem. So, what it does is, you take x and y as inputs, you complement the inputs, if we complement the inputs and send it to an NAND gate, it gives you x plus y let us see how. So, you take this line, this is x followed by an inverter here, this is an inversion operation, so this line will be x bar and this line will be y bar and what is the NAND doing, it will do x bar and y bar the complement of that.

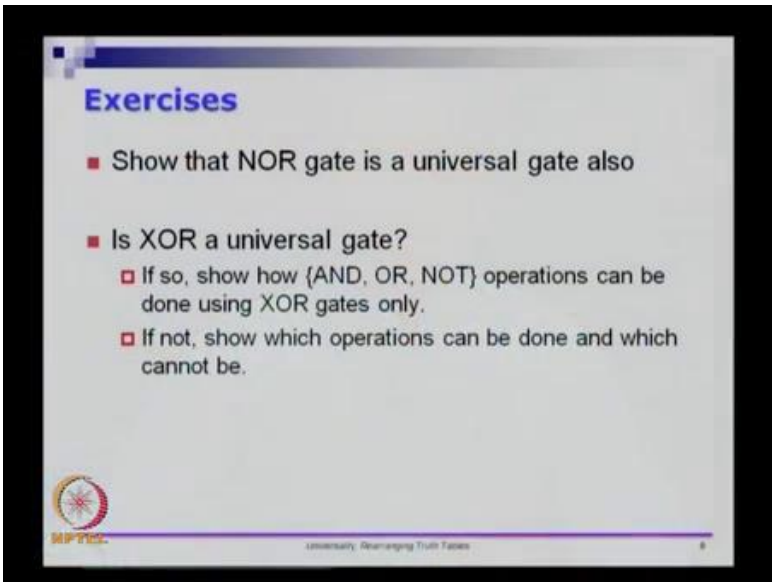
So, if you take Demorgan's theorem this is complement of the product, this is the same as sum of the complements. So, you take x complement is the term here, y complement is the term here. So, that is what you have here and this complement and this dot using Demorgan's theorem will essentially come as the logical OR here and the complements to the inputs to that gate, which is x bar and y bar. And we know that x bar bar is x, y bar bar is y and this essentially gives us as a black box if you look at this boundary. So, if I draw boundary here with x and y as input and f as output, this will give me a OR gate.

So, the reason y this universality is interesting is that if I am given just NAND gates you can implement this circuit, as long as I can do AND OR and NOT I will be able to implement AD circuit. So, what I would like you to do is, go and think about whether NOR is a universal gate and if it is, so show thus. So, show that it is actually universal gate.

So, you have to use a process which is very similar to what I did earlier, so in fact I will give this to you NOR gate is actually universal, I would like you go and show that it is

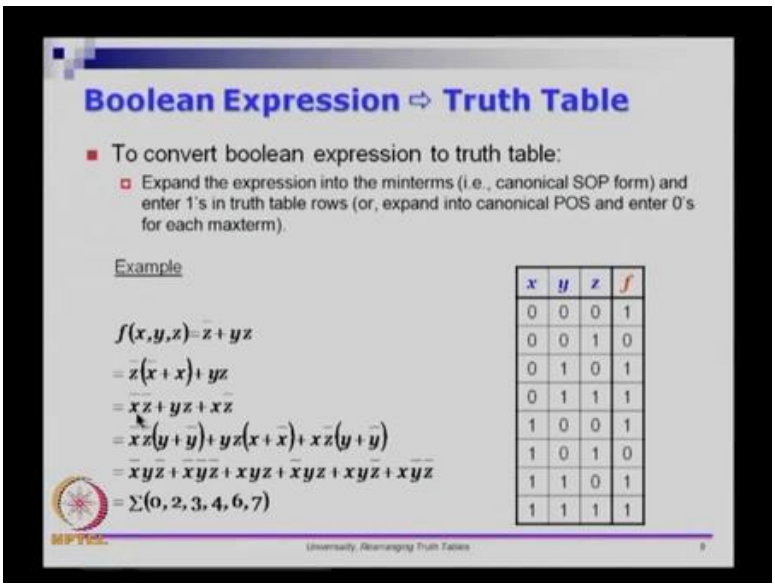
universal by appropriately combining the Boolean functions, see if you can get AND OR and NOR operation only using NOR gates.

(Refer Slide Time: 13:16)



So, the other thing that I want you to think about is XOR a universal gate, so if XOR is a universal gate it should be implement OR NOT and AND only using XOR gates, nothing else. So, if you think that it is a universal gate, show how each of these operations can be done using XOR gates only. If you think it is not at least think about which operations can be done using XOR only and which cannot be. Go on the in something think about why if the second one is the case, see y it happens also, if first one is the case you want to give a circuit which actually shows this, so do this as your mental exercise.

(Refer Slide Time: 13:58)



So, now let us which to something else, so we have to going to looked at how to go from Boolean expressions to truth tables and how to go from truth tables back to Boolean expressions. So, if you want to convert a Boolean expression to truth table, it is relatively easy. So, what we want, so there are several ways in which you can do it, I will show you one way which would can be done. So, in this example f of x, y, z is z bar plus y z if we are given something like this and let us say I want to implement this function.

Remember, this is z bar plus y z it is not a function of x, if you notice it is not a function of x. In fact, if this can be simplified little further, but let start with Boolean function now. Let say I gave you this function, I want you to fill up the truth table for it and I am asking the truth table to be a 3 inputs, which means you need 8 rows and 4 columns. So, that is what we have here, we have a function x, y, z 3 inputs. So, there are 8 different input combinations and f is a function of x, y, z.

So, I am given this function I want to fill up this truth table, let see how to do it, so first we take z bar. So, what that means, is, so this is the sum of products, so I will take each of the product terms. So, the first product term is just the simple literal z bar and if I want to see what is the contribution of this to the truth table, what it says is whenever z is 0 this expression is 0 complement plus y z, which is same as 1 plus y z which is the same as 1 which means if z is 0 f becomes 1.

So, what we can then do is where ever you see z equal 0 put f equals 1, because that is what this function says if z is 0, 0 complement plus y z is 1 plus y z which is same as 1. So, where ever z is 0 put a 1. So, there are 4 places where z will be a 0, this row, this row, this row and this row. So, that takes care of z bar, now let us go and look at y z, so y z if this term becomes 1, then z bar plus 1 will be 1. So, if both y and z are 1, f will become 1.

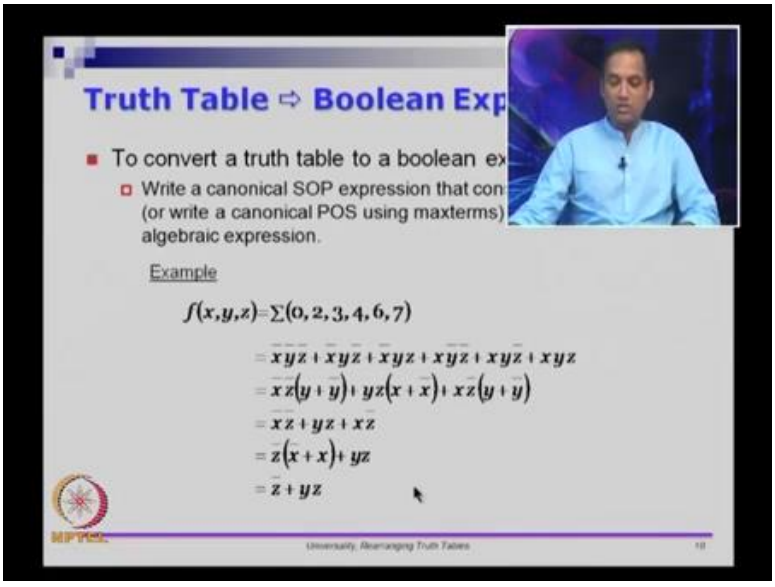
Let us go and look at the rows where both y and z are 1, so in this row both y and z are 1 and in the last row y and z are 1 so, you have marked that. Now, we have exhausted all the product terms which means all the entries that we need to put in or already taking care of we do not need to do anything more. So, whatever is remaining must be 0’s, so these two are essentially 0’s, so this is one way to fill up the truth table. Another way to do that which is slightly more explicit is, get the min terms that are involved in it.

So, the way to do that is, you take z bar we have seen these expressions before, so this is missing a x term. So, express in terms of x, so z bar into x bar plus x plus y z, then this is

essentially a rearrangement x bar z bar plus y z plus x z bar. Now, if you look at each of these terms, this term is missing a y. So, we introduce y this term is missing a x, so we introduce something in terms of x this is missing a y so, we introduce in terms of y. That gives us these 6 terms and if you look at these 6 terms, so x bar y z bar, so this is 0 1 0 that is the same as to 2.

Then you have 0 0 0 that is 0, then 1 1 1 is 7, then 0 1 1 is 3 and so on, so this gives as 6 min terms 0, 2, 3, 4, 6 and 7. If you go and look at this truth table here, the row containing 0 0 0, the row containing 0 1 0, the row containing 0 1 1 and so on will be 1's the row containing min term 1 and min term 5 should contain 0, we can verify that. So, the row containing min term 1 is this row and row containing min term 5 is this row these have 0's and 1.

(Refer Slide Time: 18:17)

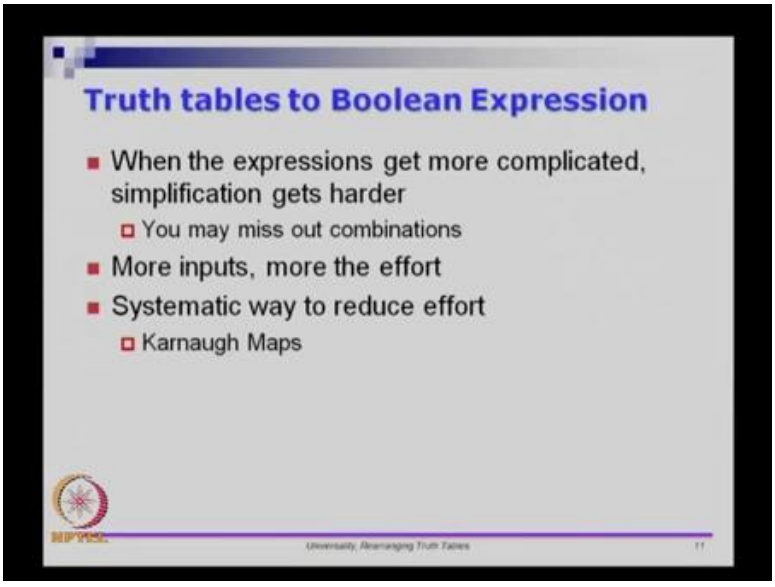


So, if you want to go from other way round given a truth table given get to a Boolean expression, this is slightly more complicated. So, what you should usually do is write a canonical sum of product expression and simplify the expression. For example, if you giving something like this, this is same thing as what we had earlier 0, 2, 3, 4, 6, 7 you go and write, so this is the canonical sum of product it has min terms.

Now, you go and write it as an algebraic expression involving x, y and z, so each one of these terms corresponds to one term from here in the same order. Then, you start applying some simplification logic whatever rules at we have seen, so for you apply that sequence of simplifications, you end up this something like this z bar plus y z. So,

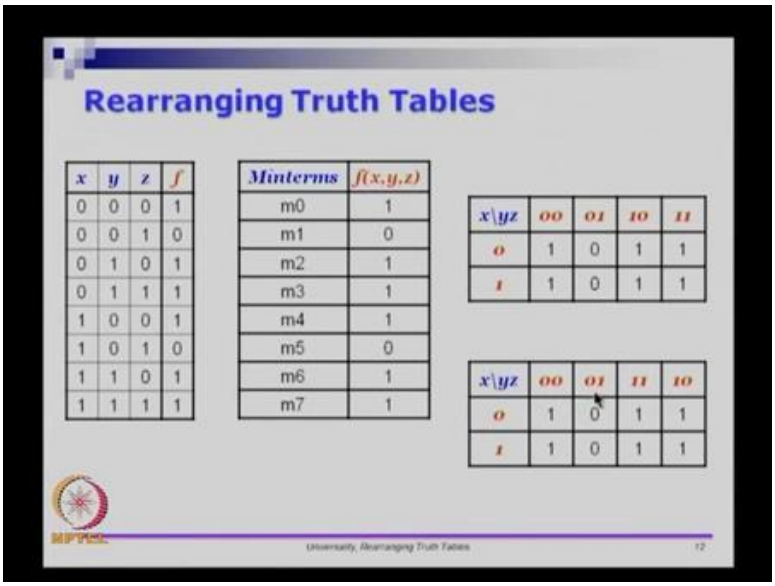
starting from an expression, so starting from a CSOP Canonical Sum of Product, so this is essentially a truth table. So, it says the 0th row, the 2nd row and 3rd row and so on, the row containing min term 0, min term 2, min term 3 and so on are all ones. So, we have taken that derive the much simpler Boolean expression. So, this is the process that we have seen in the last week and let us quite complicated because you have to remember all the rules and so on it gets a bit messy.

(Refer Slide Time: 19:35)



Let see there is way to simplify that. So, when the expression get more complicated this Boolean simplification gets very hard and give me actually miss out some of these things and you may end up with something which is complicated. Also more the number of inputs that is given you have to put in more and more effort. So, we want to have a systematic way of reducing this effort and the next module will go and look at what are called Karnaugh maps which to exactly that.

(Refer Slide Time: 20:04)



So, before we get it Karnaugh maps let see what these truth tables are, so let start with same function here x, y and z and this is my function f that is necessary. So, the first thing is I am going to do is, instead of saying x, y and z I am going to say these are the min terms and I preserve the order in the output f of x, y, z. So, I know that order of the input is x, y and z, because that is the order in the left to right.

So, I know the min terms now, but what about the min terms, we know that 0 0 0 corresponds to min term 0, 1 1 1 corresponds to min term 7 and so on. So, essentially for min term 0, if min term 0 if this product term becomes 1 f becomes 1, for if this product term becomes 1 the min output to still 0 and so on.

So, you can see the corresponds between the table here on the left and the table on the middle, now let see if this can be rearranged differently. So, the first thing I am going to do is, instead of writing it as 8 different rows and 2 columns I am going to rearrange it a little bit. So, I have 3 rows here and 5 columns, but the truth table is actually in the last 4 columns and the bottom 2 rows, where ever you see 0s and 1s, so let see what this means.

So, the first thing is let us go and look at what is here, it says x is the representative for the columns here and all combinations of y z or represented in this row here. So, there are 4 columns representing the 4 combinations of y and z and there are 2 rows which are representing 2 combinations of x. So, if you notice the order x can takes 0 or 1 and y z can takes 0 0 0 1 1 0 or 1 1. In some sense what we have done is, we have taken this and

folded it, folded this table into something like this.

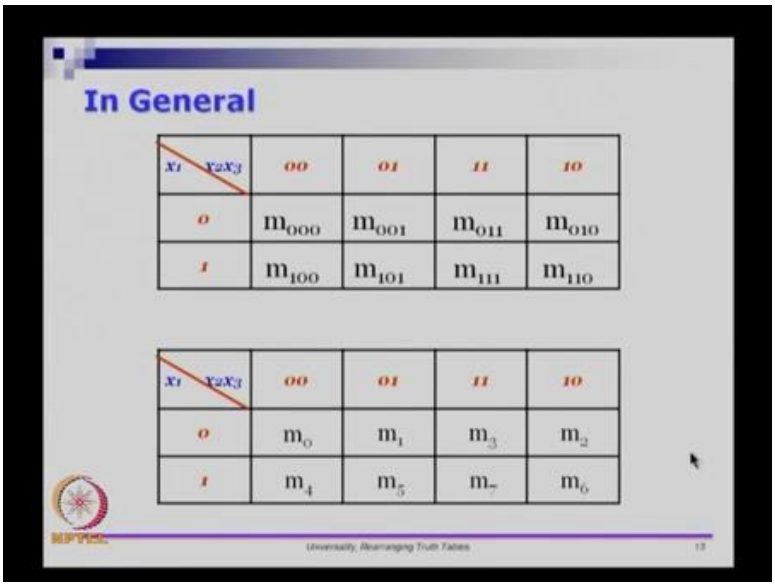
So, let see what the entries are. So, when x is 0 it is these top 4 rows here, these top 4 rows if you look at the entries here, let us 1 0 1 and 1 that is what you have here 1 0 1 and 1. Similarly when x is 1 here, we have y and z has 0 0 0 1 1 0 1 1 for those the output should be 1 0 1 1. So, that is what you have here 1 0 1 and 1, so if I pick this it corresponds to the term x equals 0, y equals 0 and z equal 1 are this is min term 1.

So, if I pick this term here, this corresponds to x equals 1, y equals 1 and z equal 0 or this is for min term 6. All I have done is I have taken this table here and folded that in a slightly different way. Now, I am going to take this and do something more, what I am going to do here is, I will keep the row order as it is, but I am going to shuffle the order of the columns. So, I have 1 1 here this column 1 1 contains 2 1, so I am going to bring that here, this column 1 0 contains 1 1 I am going to bring that here.

So, the ordering a slightly different here in x it can take value 0s and 1 and y z is going to take the value 0 0 or 0 1 or 1 1 or 1 0. We have not lost anything from the truth table here, you pick any term here you go back and check those entries in the rows here, they are still the same all we have done is taken this table folded it in a different form and then rearrange the columns and we are here now.

So, for 3 input functions you are going to rearrange the tables in this form, we will see the reason why we are doing this form as opposed to this or this. So, we get it to something interesting the moment you write the truth table in this form we will see that in the next lecture.

(Refer Slide Time: 24:02)



So, in general what we have is, we have let us say it is a function of 3 inputs that we want to look at the inputs let say are x 1, x 2 and x 3. Then, you rearrange them this form namely 0 0 0 1 1 1 and 1 0 let us clear that if we read from x 1 to x 2, x 3 this is 0 followed by 0 0 that 0 0 0 1 followed by 0 0 that 1 0 0 and so on. So, this will give us min term 0 this is supposed to be min term 4 m 0 0 is 1 min term 1 and this is m 5 and so on. So, this is min term 0, 1, 3 and 2 and this row contains min term 4, 5, 7 and 6.

So, given a truth table without going through this folding and other things you can directly come to this one, we will see why this is useful in the next module. So, this brings me to end of this module, what we have seen this module is universality of NAND and NOR gates and we also saw how to rearrange the tables and so on to set up the platform for what is called Karnaugh maps. We will do that in the next lecture.

Thank you.