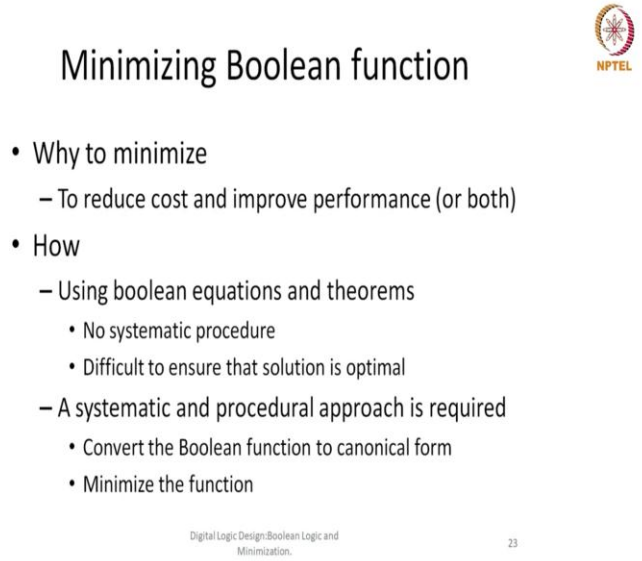
Digital System Design
Professor Neeraj Goel
Department of Computer Science Engineering,
Indian Institute of Technology Ropar
Canonical Form

Hello everybody. In our previous lecture we have seen that how Boolean algebra, its theorems can be used to minimize or to simplify a Boolean expression. And we also understand that simplifying a Boolean expression is very important from the cost perspective. So and we have also seen that if we use Boolean algebra, so we have observed that there is no systematic procedure. So for example, if we want to reduce number of terms so there were four steps suggested, we can use uniting theorem, you can use consensus theorem to remove literals, to remove terms.

And along with that some other methods were suggested that we can use, we can add additional terms, redundant terms so that we can reduce some other terms. So which steps need to be performed and in which order, this is a complex problem and there is no known solution. There is no working solution which can tell you that okay you have to first apply this step, then go to this step and then go to this step. And unfortunately what happens that if you go with a different order of steps, sometime different optimized expression comes out.

(Refer Slide Time: 01:40)



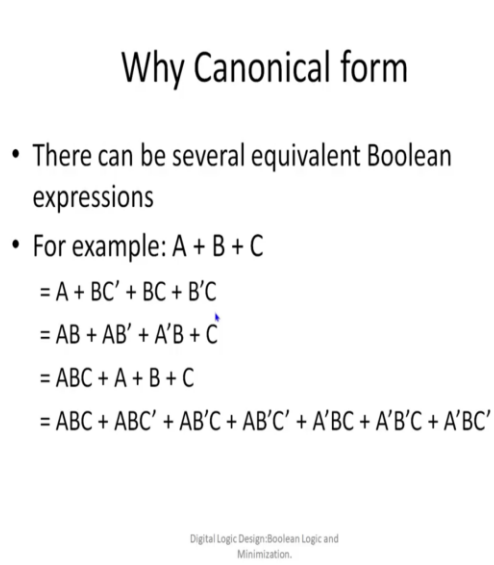
So what we need if we want to minimize a Boolean expression? Then we require a systematic or a procedural approach. Systematic means that there should be only one solution and procedural

means that we can follow some steps, step number 1, step number 2, step number 3 and then finally we will come up with a definitive solution. So this systematic and procedural solution, we will in this lecture, we will start understanding how that systematic and procedural start, procedural approach will work.

So the first step in that procedural stop, procedural approach is first we convert the Boolean function or Boolean expression into canonical form or standard form and then we go ahead and minimize that expression. So let us again understand why minimization is important. So we have seen that if we have less number of terms that certainly reduces the gates, number of gates that means it would reduce the cost.

And also if we reduce the number of literals in each term that would reduce the number of inputs in each gate and that would also affect in reducing the cost as well as the performance will also improve. Performance means the overall latency of that function, overall delay of that function will also reduce. So with that goal we will first try to see this systematic approach. In this systematic approach first we will understand what is a canonical or a standard form.

(Refer Slide Time: 03:40)



So to see this step, to see this reason or like why do we require a standard or canonical form. So in last two lectures we have seen there are several equivalent Boolean expressions. So that essentially means that one Boolean expression, let us say A plus b plus C. There could be many

different forms in which we can represent the same expression and unless we know that what that expression, whether that expression are equivalent to this original or given equation.

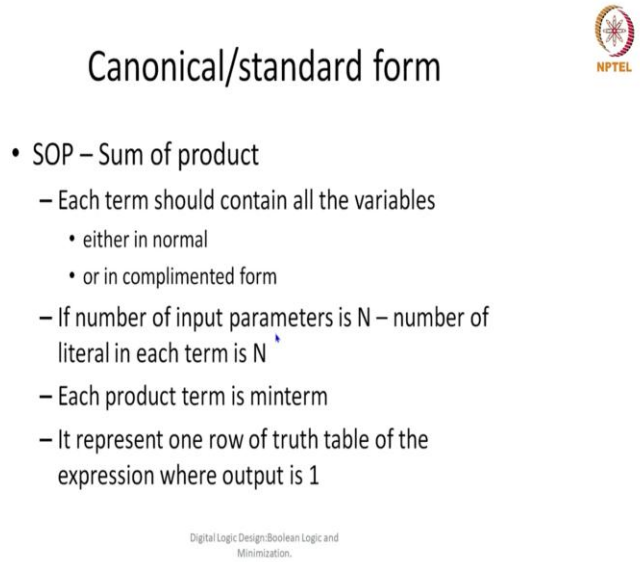
So whether they are same or not it again is effort to know whether two expressions are same or not. The only method which will always work when we are comparing two expressions, whether they are same or not is to write truth table of both these expressions and then match each and every row of the truth table if they are matching then we can say that they are same, otherwise we cannot say.

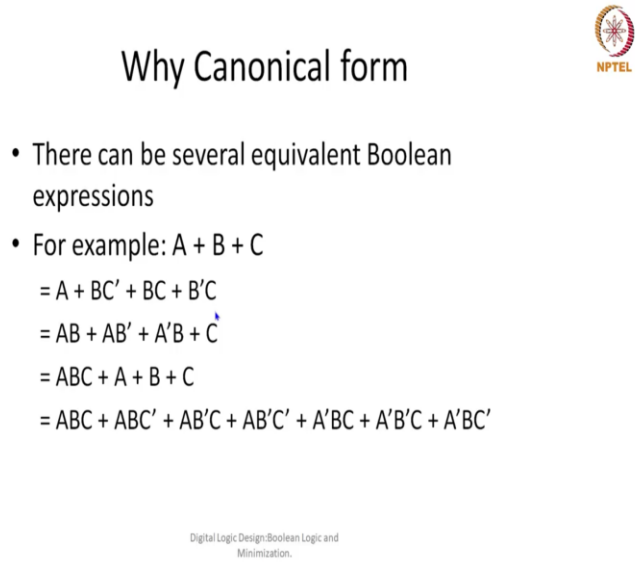
So because even matching two expressions whether they are equivalent or not is a tedious process. So that is why we would require one canonical or a standard expression or standard form in which a function can be represented, so this example is given like A plus b plus C. You can see that I can also write this A plus b plus c equal to A plus BC dash plus BC plus b dash c or I can also write it like ab plus ab dash plus A dash b plus AC plus C.

So there are various other and you can take it as a exercise that whether all of these are equivalent or not you will finally see that yes, they are actually and using the, using some of the theorems which you have studied, we can use those theorems to understand to see whether there we can check equivalency based on those theorems.

However sometimes those theorems are not sufficient because as I said earlier that you have to perform steps in certain order and that order is a sort of undefined. So that is why if there is some canonical or a standard representation that means that given an expression, first we go ahead and represent that expression in a canonical form and from that canonical form because that is going to be only one canonical form from there we start and then we minimize.

(Refer Slide Time: 06:19)





So there are two canonical forms which we have seen, which we will see in this lecture. One is called sum of product and another is product of sum, so canonical sum of product form. So here each term, so you see how many terms would be there? There would be product of sum of product so that means one term is a product term.

So in each product term if there are n variables, all the variables should be present in each term either in the normal form or in the complemented form. So in other words if there are n parameters or n variables in that function then each term will have n literals. Literal mean either

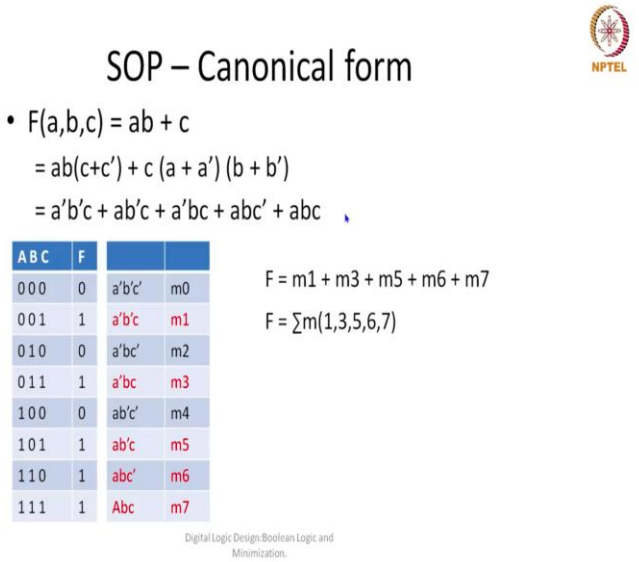
that would be present in the normal or original form or in the complemented form. So there has to be n literals in each product term.

So for example, in this case I see that there are 3 variables A plus b plus c that means there has to be at least 3 variables. So in this, this is also a product, sum of product, so in sum of product this product term has only one literal and this product term has only 2 literals. So here also, here the literals are variable. So these are 2 literals in each of the product term and this particular term has only one literal.

Although in the last one you see each of the product term has 3 literals or essentially all the 3 input variables are present in each product term. So ABC dash, ab dash C, ab dash c dash, A dash BC, A dash b dash C, A dash BC dash. So all the product terms have all the 3 variables either in the complemented form or the normal form. So this would be the canonical form of the sum of the product expression.

So this is in other words, there is something called standard S.O.P standard sum of the product form of that particular function. So now this is clear that in a canonical form in each product term there has to be all the literals or all the variables present. And the second thing is that all of these product terms are also called minterms. And if we see this is sort of a representation of the truth table. So for example, in your truth table wherever you have a 1 as a output, you 1 product term would be corresponding to that particular one. And so let us understand this minterm thing and also this truth table with help of an example.

(Refer Slide Time: 09:16)



So let us say we have this function ab plus c, it has 3 variables a, b and c. Now if I want to have this canonical representation so then each product term has to have all the variables. So what I will do is I will do, I will multiply ab with 1 and 1 also means c plus c dash. So I am multiplying ab with c plus c dash and I am multiplying c with a plus a dash and b plus b dash.

So the overall idea is so that I can have all the variables in each product term. Now I can multiply this ab with c and ab with c dash and similarly I will multiply this whole expression. Then the expression which will come out is, the expression which will come out would be like this a, I have started with this a dash. Yeah, so I have first opened up this one. So this is a dash b dash c, ab dash c, a dash bc, abc dash, abc.

So there were two times this abc was generated, so I have removed one of that abc. So essentially there would be 5 product terms. So then if I write the truth table of this expression a b plus c it will come out to be something like this that first I will enumerate all the options of abc 000 001 up to 111. And then for each input sequence, I will find out the value of F by applying that ab plus c. So essentially a and b, both of them has to be 1 or c has to be 1, then F would be 1.

So F is 1 because c is 1 here and if c is 1 then it is 1 and c is 1 so it is 1 but both a and b were not 1 so that is why it was not 1 there, it was all 0. So when a and b both are 1 but c was 0 then also it is 1 because at least a and b both are 1. So similarly this is the truth table. So if we try to find the correspondence of this sum of product term with this table of truth tables so you will see that

each of this term corresponds to a row of this truth table wherever output is 1. So for example this here this I can say a when a dash.

So when a is 0 that means a dash is 1 and b is 0 means b dash is 1 and c is 1 then only this particular output would be 1. So this is corresponding to 1 minterm which is a dash b dash c. So in other words this particular minterm a dash b dash c would mean that only this particular term, so let us say in a function only this term is there a dash b dash c. So that means in my output only this particular for this particular input combination, output would be 1. And for all other cases, output has to be 0. So this is also the reason it is called minterm.

So this is the minimum terms that has to be, minimum term to represent this particular function, so if only 1 value is 1. So now the another way. Now another way to look at this whole thing is that wherever 1 is there, if corresponding minterm corresponding value of a and b, ab and c is written here so that would I can use all of those 1 terms and say that if any of those 1 terms any of those minterms is 1 then my function F would be 1.

So this is another way of looking at sum of product expression. So here let us see the minterms corresponding to all of these A, B, C. So for example this row corresponds to a dash, b dash, c dash. So if this particular minterm is 1, so that means my a dash, my value of a is 0, b is 0, c is 0 then only expression will give me 1. So then this particular minterm has to be 1 or has to be present in my canonical form.

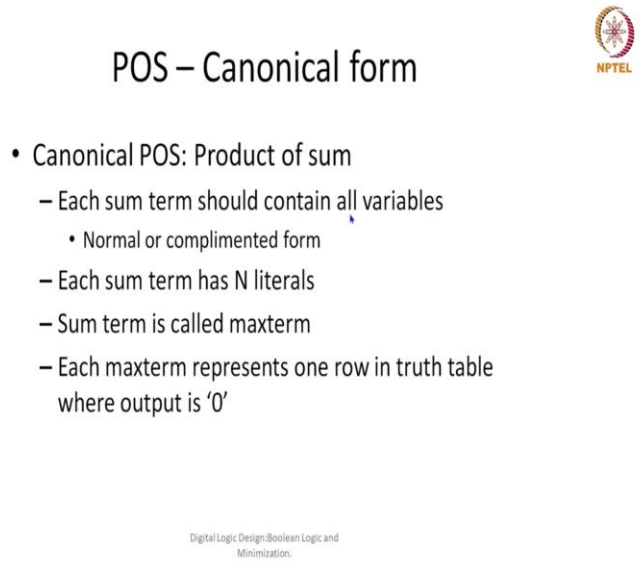
So here a dash, b dash c is present so I am making it red and calling this minterm as m1. So these are represented using decimal numbers. So they are not represented by using binary numbers but decimal numbers. This represent 0 because 000 is 0 and 001 is 1 in decimal. So we are writing it as small m means minterm 1. So similarly a dash bc this particular term. So a dash bc is another minterm which is present.

a dash means a is when value of a0, when value of b is 1, c is 1 then this particular minterm will become 1. So similarly we can see all other minterms which are present here. So the overall sense we have to make out of this minterm as well as the correspondence with truth table that when any of these minterm is 1 then my expression is true. So each minterm represents to 1 in the truth table. So the whole expression that is why in a canonical form we can also write it like a function, this F is equal to minterm 1 plus minterm 3 plus minterm 5 plus minterm six plus

minterm 7. So rather than writing in this long form we can write directly the corresponding decimal number for that minterm.

Now further for expressing this or for writing this we also sometimes write the expression this is the symbol of sum and m we are using this parenthesis and using these parenthesis means this is m1, m3, m5, m6 and m7. And because of sum where sum symbol we are adding all of these minterm. So this function F in a canonical or a standard S.O.P form can be represented like this which is also equivalent to expressing it in terms of variables like this. So this is one of the canonical form which is sum of the product.

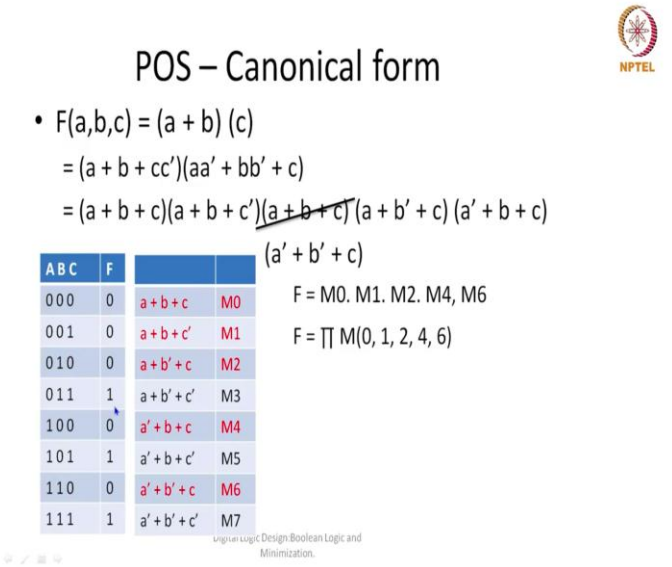
(Refer Slide Time: 16:54)



Similarly, the other canonical form is called product of sum. So it is the dual nature of Boolean expression that anything which can be represented as sum can also be represented as a product. So here we have a product of sum means each sum would be considered as 1 term and because it is a canonical in nature or standard in nature.

So each sundown should have all the variables present either in the normal form or in the complemented form. And like sum of product here also each sum term should have n literals. And these sum terms are called max terms and each of these max term represent one row of the truth table wherever output is 0. So let us see again with an example.

(Refer Slide Time: 17:51)



So I am taking a very similar example, let us say a function abc is represented in sum of in form of a product of sum where a plus b is 1 sum term and c is another sum term. So if we want to create a canonical form then the idea would be that, first of all I have to add, I have to add like I, there should be all the variables present in each of the sum term.

So here c was not present so I have added I created a plus b plus 0 and 0 can also be written as c, c dash because I wanted in each of the sum term there has to be c and c dash both the terms. And similarly here in this sum term a and b both has to be there so I added a, a dash plus b, b dash. Both of them are effectively 0. So by expanding this expression we can show we can see what would be there. So using the distributive law, a plus b plus c and a plus b plus c dash. And similarly, if we expand this whole form then it will be a plus b plus c, a plus b dash plus c, a dash plus b plus c, a dash plus b dash plus c.

Now we see abc appears twice here so we can remove one of the, which is redundant. So again if I would create the truth table for this particular expression then truth table will come out something like this. So my output is 1 when both either like both this expression a plus b is 1 and c is 1. So whenever c is 1 and either of a and b is 1 then the output is 1. Similarly, either of a and b is 1 and c is 1 then the expression is 1. Here also when both a and, either a and b is 1 and c is 1 then the expression is 1.

So a max term actually would represent a 0 in each of this expression in each row of the truth table. So for example, if I have a plus b plus c, so I will write it like this, if corresponding to this row if we have this particular term, this particular sum term A plus b plus c, what is the result of that? That means that only in this particular row it would be 0. In rest of the rows, it would be 1.

So similarly, if a plus b plus c dash, is the sum term that means it would be 0 here and would be 1 in rest of the rows. So similarly we can find out the sum expression corresponding to each row and then we can also represent it using decimal number 0, 1, 2, 3, 4, 5, 6, 7. Now so we have created a sum term corresponding to each row. So this particular sum term corresponding to each row is called max term because in that case only 0 is present in that row and in rest of the cases, in rest of the rows, 1 is there as a output, so it is also called max term.

So it is a just terminology you can see it that way. So the whole expression can be written as that if, it is a product of all 0 terms. So that means when any of those term is 0. The whole expression would be 0 or in other words all these sum expressions has to be false to make the whole F as 1. So I will repeat. So let us say I have a product of sum. So each of the sum term, when each of the sum term, any of the sum, any of the sum term is present that means that particular sum term will make the whole expression as 0 or to make that function 1, none of the sum term has to be, none of the sum term has to be 0.

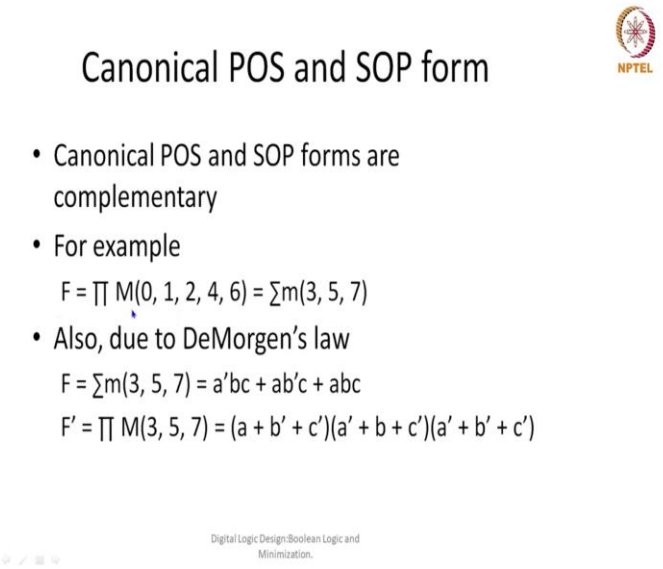
So that means the only remaining part, so the terms which are not present here, any of them is there then only this function F will become 1. So this is how product of sum expression will form a canonical form. And now for this particular function a b plus a plus b into c, we see the, these max terms being present m0, m1, m2, m4 and m6. So we can say that my function F can be written as product of m0, m1, m2, m4 and m6. So what does this mean?

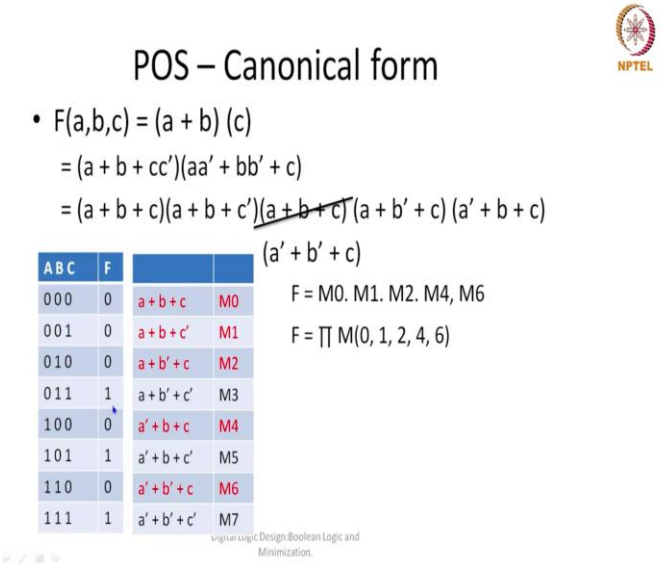
This mean that when any of these terms are present then my output would be when any of these m terms are present then my output would be 0 else my output is going to be 1. So because it is a product term, so I can write this as a product and then M capital M means max terms. So a max term 0, 1, 2, 4 and 6 are being multiplied so that I can get this expression F.

So this is how a canonical form of product of sum can be written or can be derived out of any expression. So now there is also a couple of relations. So there is some relation between product

of sum and sum of product form. So looking at this also we can say that because all 0's are represented by a product of sum while each 1 means it is a minterm.

(Refer Slide Time: 24:45)





So we can say that whenever, so they are complementary in nature. So that means if a function F is represented like this 0, 1, 2, 4, 6. Actually this function 0, 1, 2, 4, 6, then this can also be represented using sum of product, product term 3, product term 5 and product term 7. So I can say that these two are equivalent expression. And the other relation between this minterm and max term is that because of De Morgan's law. De Morgan's law means if I want to complement

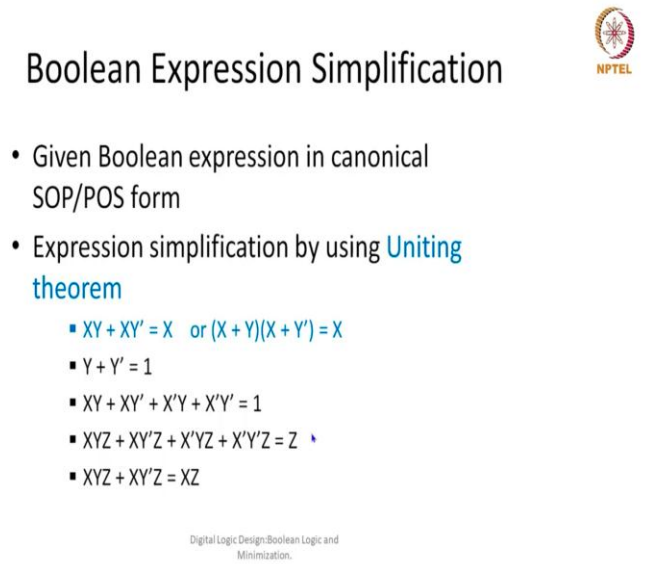
a function, so let us say F is a function where this is a F is a function of minterm, sum of minterms 3, 5 and 7.

Then I can say that F dash, in F dash all these 3, 5 and 7 would be 0 and all rest of them would be 1. So I can also say that F dash is actually a product of max term 3, 5 and 7. So we can look at algebraically also. In algebraically, if I want to write 3, sum of 3, 5, 7 it would be written as a dash bc plus ab dash c plus abc. Now if I want to apply De Morgan's law what De Morgan's law says that every product will become sum and every sum will become product and the polarity of each variable would be inverted. So a dash becomes a, b become b dash, c becomes c dash and this all the product things will come sum.

So similar for all of these product term they become sum terms and this sum term, this product term also become this sum term. So if I see this, this corresponds to 011 that means 3. And this is 101 that means 5 and this is 111 that means 7. So algebraically also we can see that De Morgan's law can be applicable here for the minterms and max terms. And this also gives us some sort of a shortcut.

So we can see that if something can be represented efficiently in minterm we can use that particular approach or something which can be represented using max term that can be followed. So after arriving at the canonical form, the next step we wanted to see is that after having the whole expanded form where all the variables are present in each of the sum term or product term then how can we further minimize. So the further minimization can be done by using, yeah.

(Refer Slide Time: 27:45)



So it can be done by using uniting theorem. So in this uniting theorem, so there is only one theorem that could be sufficient here. We can, we just need to see that which terms, which variables we need to combine. So now the uniting theorem says that XY plus XY dash is equal to X and for product of sum it is X plus Y plus into X plus Y dash equal to X.

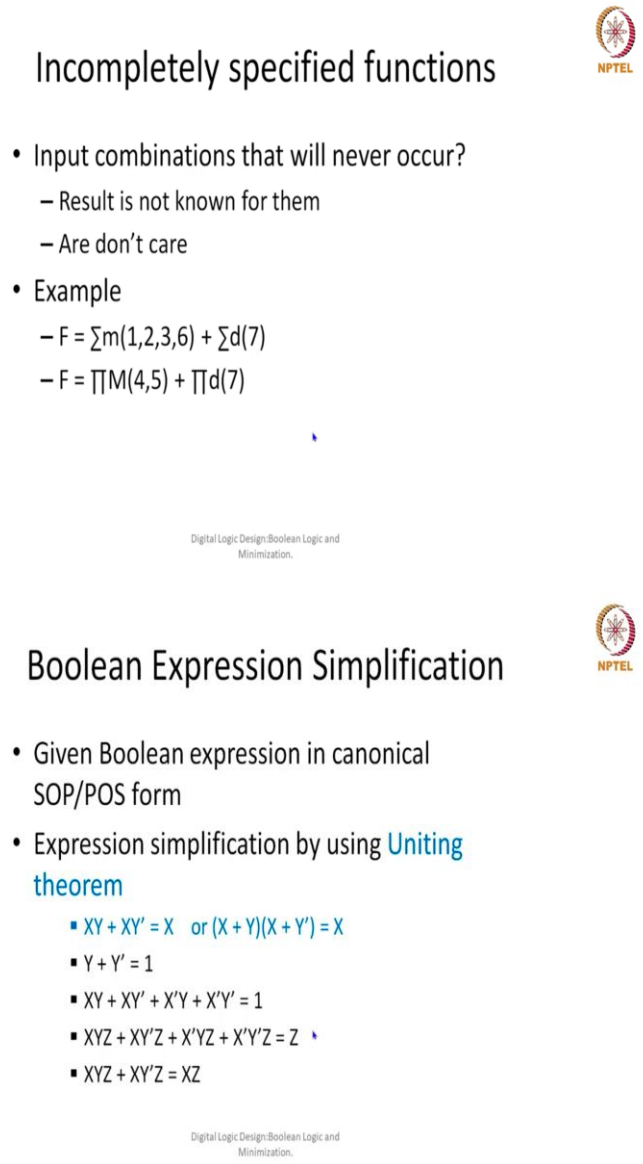
So this uniting theorem can be applied multiple times, can be applied again and again to reach to a final solutions. So only one theorem is sufficient to simplify the expression. So I am giving various examples of this uniting theorem. So let us say Y plus Y dash equal to 1 because both can be combined and it will become 1. So similarly XY, XY dash X dash Y plus X dash Y dash will become 1.

So essentially I can say that XY dash plus XY, this will become X and similarly X dash Y plus X dash Y dash will become X dash and X plus X dash will become 1 so we can apply that recursively. In other words, if all the four combinations of my truths table 00 01 sorry 11 10 01 and 00 if all of the four are present then I can say that the expression is always 1. So let us say if the expression is like this XYZ, XY dash XY dash Z, X dash YZ, X dash Y dash Z.

So I can take Z common and then I can, using this particular the above expression I can remove all the XY terms and the term final output would be Z. So this way only the uniting theorem is sufficient. It could be applied in different scenarios and different combinations that could help us

in receiving or getting what could be the minimum expressions for that particular combination. So one more question here that let us say a particular.

(Refer Slide Time: 30:19)

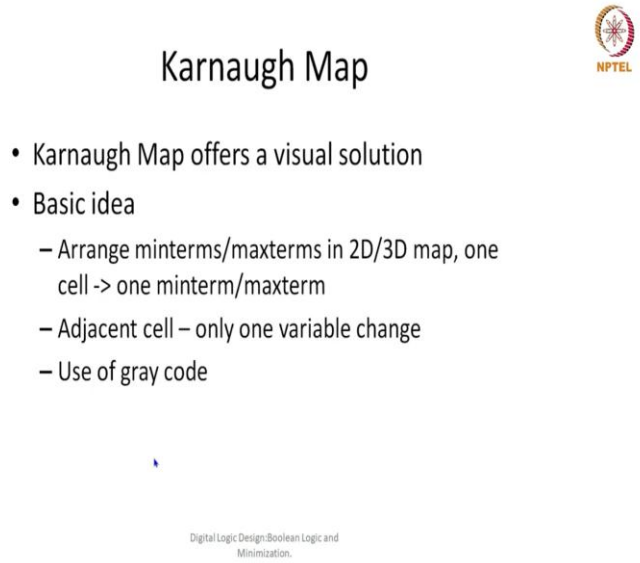


So all of this expressions all of this like minterms and max terms that depend on creating sort of a truth table, where we know for each input combination, what is the output. What if certain input combinations are not there or their result is not known or so the two possibilities, one that we do not know, what are the results. The second is that the those input combinations will never occur.

So in those cases it does not matter what are the output for those combinations, those input combinations. In such cases we call such functions or such expressions as incompletely specified functions. And those input combinations where output is not known or those input combinations which will never occur are called don't cares. So these don't cares are written like let us say in case of a sum of products it is called let us say m1, 2, 3 and 6. Then we can say that plus sum of don't cares 7 and the same expression you want to write as product of sum then it would be product term, max term 4, max term 5 plus don't care 7.

So this way these don't cares or this incompletely specified functions can also help us further to optimize so we can use these don't care terms to further simplify our expression. So we will see how these things can be done. So now we have already seen that how we can use this uniting theorem. Now the better or a systematic way because as a, as human beings rather than working with numbers, we are very, very comfortable seeing things visually, creating patterns and identifying patterns.

(Refer Slide Time: 32:31)

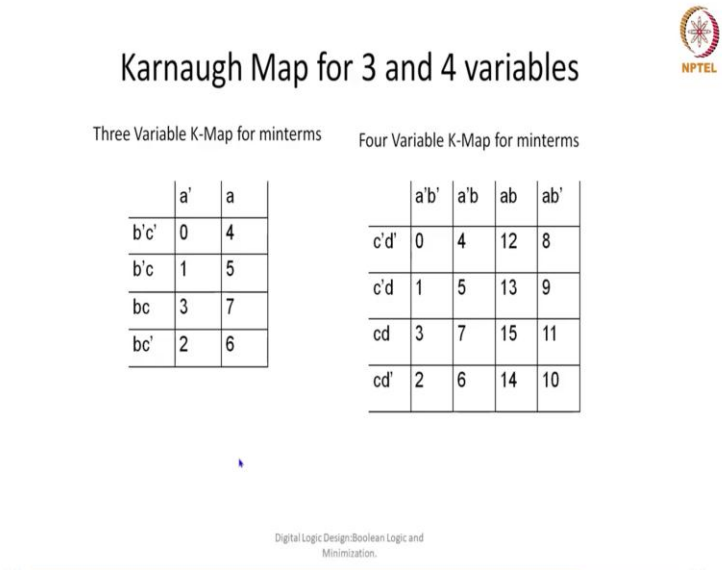


So that is why one popular approach for Boolean simplification or Boolean expression simplification is called Karnaugh map. So this Karnaugh map is essentially a visual solution where we represent all the minterms and max terms in form of a 2D or 3D diagram where there would be multiple cells. Each cell corresponds to one minterm or max term. So either it has a

minterm or max term. So let us say we are simplifying for sum of products, then we will only use minterms not max terms. So each cell in my Karnaugh map would represent one minterm.

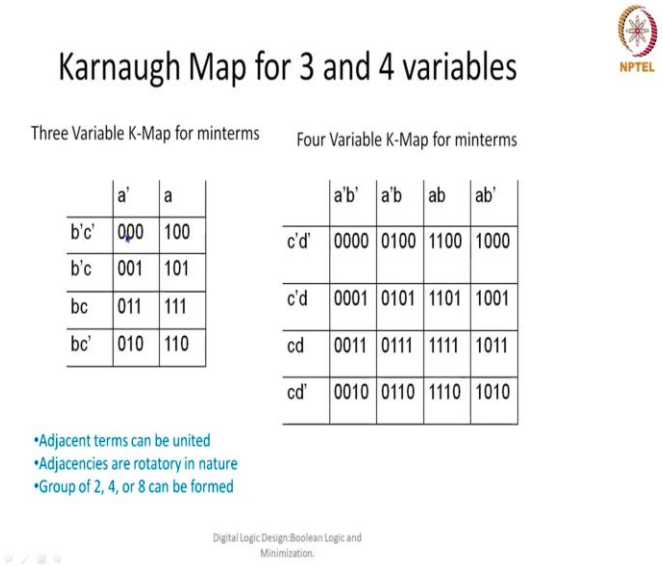
And the other property of this Karnaugh map is the adjacent cell vary or differ only in one variable. So for, let us say my Karnaugh map is for 3 variables, rest of the 2 variables would be same for each adjacent map, adjacent cell and only one variable will change. So because there is going to be only one variable change so the way these Karnaugh maps are represented their form of the representator rather than decimal got in a gray code encoding. So let us take a quick example of how these Karnaugh maps are represented.

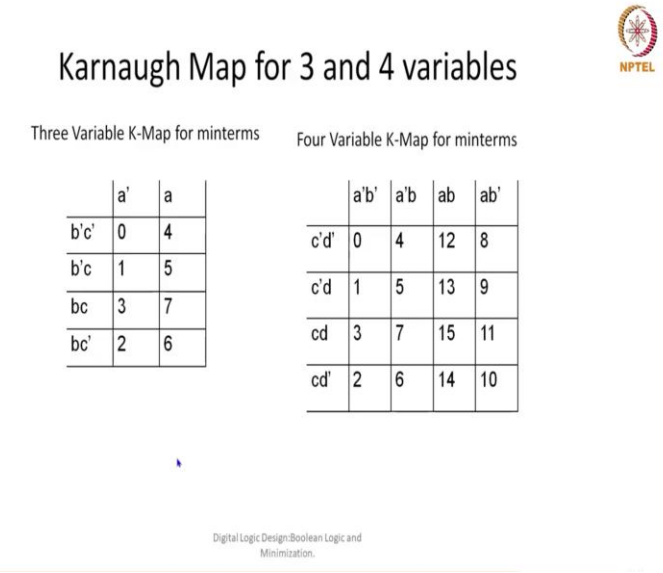
(Refer Slide Time: 33:49)



So let us say I have these 3 variables. So for 3 variables let us say variables are a, b and c and a being the, we are keeping a at the more significant place and c at the least significant place. So I can write it like 0, 1. So this is the, this is a, these are three code representation, 0, 1, 3, 2, 4, 5, 7, 6. So if it is a 4 variable so it would be like 0, 1, 3, 2, 4, 5, 7, 6 then 12, 13, 15, 14, 8, 9, 11, 10.

(Refer Slide Time: 34:38)





So I think the next slide will make it more clear. So here it would be written like this. So you see that this particular cell represents minterm 0 or it is also saying this is a Dash, b Dash and c dash and this particular cell says this is a, a means it is 1, and b dash c dash are in complimented form that means they represent 0 so this is 100 and this particular cell my value of a most significant is 0 and then b is also 0 and c it is 1.

So this represent minterm 1. Similarly, this represent minterm 2 and this represent minterm 3. So we can see this is actually a gray code where adjacent cells have only a bit flip at one place. So

here there is a bit flip at this second place and here we see the bit flip is only at the least significant bit. So similarly 4 variable K map can also be represented using this.

So I can have 2 variables on this axis x-axis and 2 variables on y-axis and they are also represented in a gray code manner. So a dash b dash a dash b ab and ab dash. So writing it this way, like writing a dash b dash or a dash b, so this help us in writing this K map in a more precise manner. But otherwise you can also remember this, these numbers if we want 0, 1, 3, 2, 4, 5, 7, 6 etc.

So once they are, so here you see there are couple of things. On that each of the cell represent one of the minterms. So whatever number was written here 0, 1, 3, 2 they are representing minterms and they can also represent max term. So in case of max term we have to remove this a dash b dash we have to write it like a plus b and this we have to write c plus d. So this can also be represented as max term, the numbers of max term will again remain same. But in that case my result would be in terms of product of sum.

So each cell represents one of the minterm or max term and adjacent terms, adjacent cells are differed only by one bit. So essentially all the adjacent cells can be united. We can use our uniting theorem to combine each adjacent cell. Now adjacent cell could be in x direction, it could be y direction. So basically it could be adjacent horizontally, it could be adjacent vertically.

It cannot be adjacent diagonally. So diagonal cells does not have any kind of a correlation but the last one, last row and the first row, first row and the last or the first column and this last column can also be combined. So you see in this first cell and this is the last cell, here also the difference of bit is only 1. So this cell is the last one. This is the first one. Here also the bit difference is only at the at 1 bit level.

So it is a kind of a rotatory in nature. So adjacency could be adjacent cell even if it is a like, it could be coming from the top or basically it could be a rotation based adjacency. The number of cells which could be combined it could be 2, it would be 4, it could be 8. So any power of 2 those if we find a group of them forming a rectangle or a square, we can use all of them to combine. So we will see all of these K maps with help of couple of examples and these examples we will I will use the writing pad so that we can you can also see that how to solve these examples.

So and that part would be seen in the next part, this lecture would be divided into two different parts. So one part is this where we have explained or we have summarized how to create a canonical form and how to create a K map. So the next part of this lecture we will take couple of examples where K map could be, various example of K map could be taken where optimizations or how we can form this group and how we can simplify using K map. Thank you very much.