Digital Circuits and Systems

Prof. S. Srinivasan
Department of Electrical Engineering
Indian Institute of Technology Madras

Lecture - 8

Karnaugh Map Minimization using Maxterms

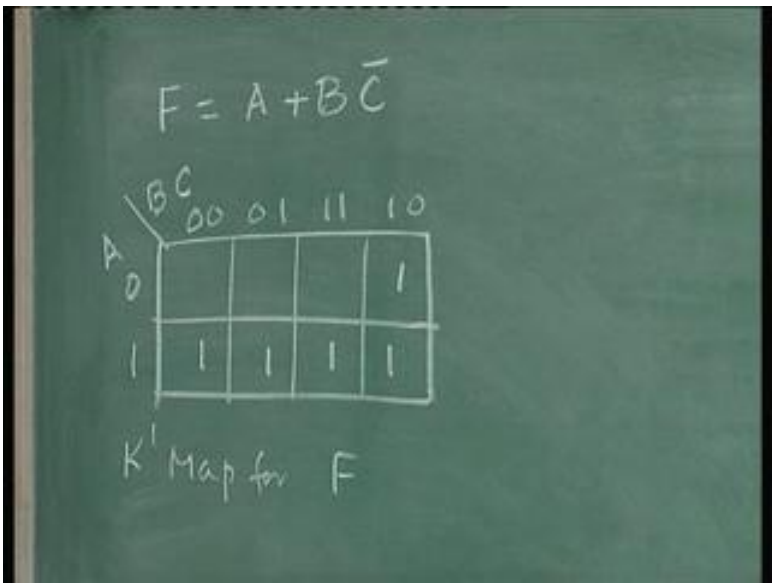
We have been taking about minimization of Boolean functions first by using Boolean algebra and then by map method by drawing a graphical representation of the truth table and then identifying the patterns of 1s and group them and reduce the variables in that process, number of variables as well as number of terms.

If you remember we also talked about min terms and max terms. That is we can write an expression Boolean algebra either as sum of products or product of sums that is AND terms combined with an OR gate and OR terms combined with an AND gate. So one is called sum of products and the other is called product of sums because analogous to the algebra 1 looks like a series of product terms and sum them, the other looks like a series of sum terms you multiply these sum terms as a product.

Sometimes what happens is when you draw a Karnaugh Map to simplify you find that there are more 1s than 0s. The idea is to minimize we said. The total number of terms should be as small as possible and the number of literals in each of these terms should be as few as possible. So when you have large groups of 1s it results in many prime implicants and essential prime implicants. If there are more 1s than 0s is it possible to use the 0s and get F bar complement of F expression. A 1 in the truth table says F is true the function is true and the 0 in the truth table says the function is false. So when you group 0s we can get an F bar expression just as we get F expression by grouping 1s and once I have F bar I have De Morgan's Theorem to get an F. So, that is one approach some people use. That also depends on the type of gates you want. It all depends on whether you want a sum of product as the minimum expression or the product of sum as the minimum expression. So let us today see an example where we will use 0s to simplify the logic function.

We will use our same familiar example from the last few classes. We will be using the function F is equal to A plus BC bar this has been our example. So when you do the Karnaugh Map for this if you remember (Refer Slide Time: 5:40) this is the K map where there is a one here one here one here one here one here if you look at a truth table you had one entry in the third row and last four rows. The first row second and fourth rows had 0 in the output and all other rows have 1 in the output. And if you map it this is what we got.

(Refer Slide Time 6:48)



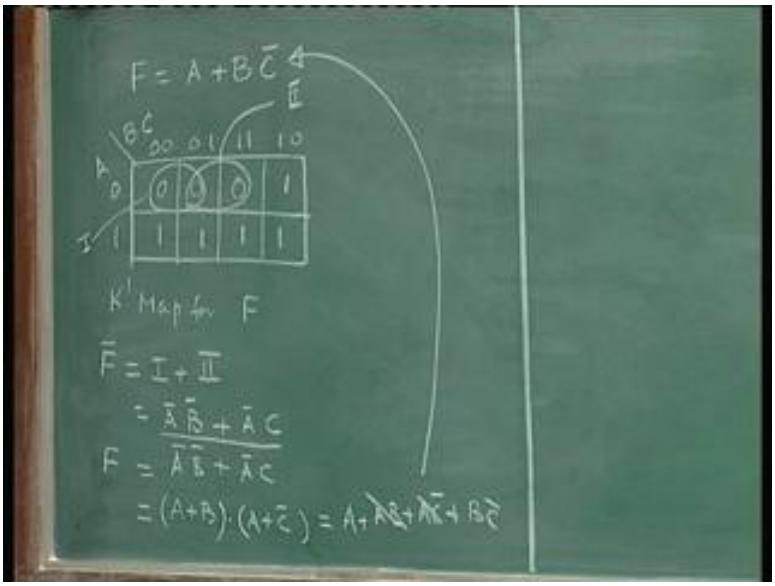
What we did last time was to simplify this using 1 here and then here, of course this is a trivial example, even though there are large number of 1s if the pattern is good large groups of 1s together it will give you a very small sum of product expression because larger and larger groups give you smaller and smaller product terms, that way we had a nice example. In this example we had A for this and BC bar. But just to give you the concept of using 0s to simplify the expression I am going to feel these entries of the truth table which had output false with 0s I am going to use 0s to simplify my logic function. So if I group these 1s and write prime implicants I will get F expression, expression for the output F.

If I now group these 0s and identify prime implicants or essential prime implicants I will get F bar is it not, whenever the function is not true there is a 0 entry in the truth table as well as in the Karnaugh Map. So now I am going to group these two 0s and these two 0s so I will call this one, prime implicant one, prime implicant two, F bar is sum of prime implicant 1 and sum of prime implicant 2, a combination of prime implicant 1 and prime implicant 2 when you say sum you mean really OR operation which is 1. This is A bar B bar correct and this is A bar C (Refer Slide time: 8:32), this is only F bar but we want F because the problem definition or the truth table given to you is to implement a function F which has the output asserted as true for the given combinations but what we got is F bar output not asserted or output asserted as 0 so we do not want this but we want a complement of this and you know how to get the complement.

It is by applying DeMorgans theorem on both sides we get F which is nothing but A OR B because the variable gets changed into complement where AND becomes OR, OR becomes AND, variable gets complemented and you can simplify this to the original F which is A plus BC bar because now if I do this this would become A plus AB plus AC bar plus BC bar and the view of the identity in the Boolean algebra if you remember A OR AB is A so this is redundant similarly A OR AC bar is also A, this compared with

this gives you A so these two terms get knocked off so the result is A plus BC bar and this is what this is.

(Refer Slide Time 10:33)



There are a couple of points I want to make really; one is, smaller number of 1s grouping is easier only if they are randomly situated actually. When 1s are nicely grouped we may find a simplified expression of a minimum form. But if we had 1s and 0s spread in the random fashion as more 0s than 1s and more 1s than 0s then may be it's a good idea to go for simplification using 0s.

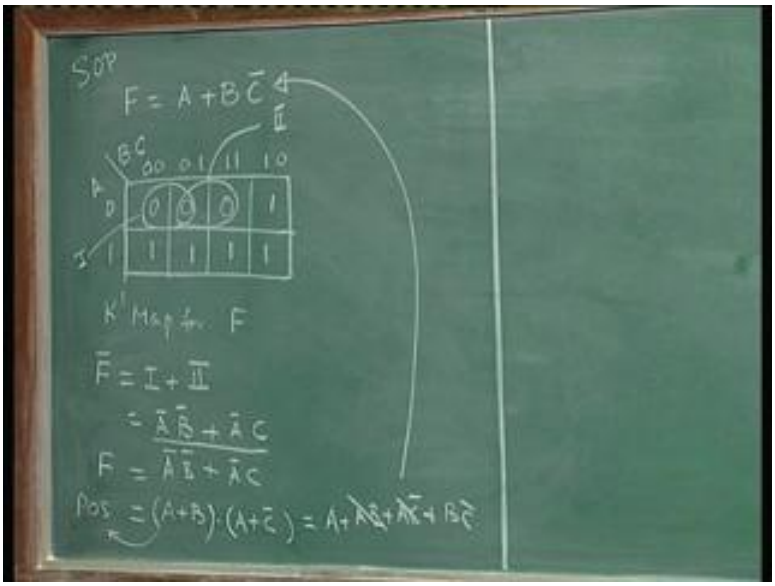
The second thing is this is the expression for F without this simplification. This simplification I need to prove that this is same as this, I can stop here. This is the minimum product of sum. This is the sum of product expression (Refer Slide Time: 11:24) and this is the product of sum expression. So if you wanted the solution in product of sum expression it is here, I would use this expression the minimum product of sum this is a minimum sum of product.

Just as you get a minimum sum of product expression using Karnaugh Map I will also get a minimum product of sum expression using Karnaugh Map using 0s. Where will I use that situation? I will use the situation where I want to have OR gates feeding into an AND gates because here each product term is an AND gate and the sum term represent an OR gate. Sum is equal to OR, sum is an OR operator because you put a plus we call it sum plus it is an OR operation and dot is an AND operation. Hence the sum term represents an OR gate and product term represents an AND gate. So if you want to have fewer OR gates and larger number of AND gates I will use the sum of product expression or if I have fewer AND gates and larger number of OR gates I will use this.

Thus depending on the topology as they call it I will have either a sum of product expression or product of sum expression even if number of 0s more or less even without

considering the number of 0s and 1s in the truth table or regardless of 0s or 1s distribution in the truth table or the Karnaugh Map we may sometime want to write an expression as a product of sum expression. In that case I will use this approach. I will group 0s together and get F bar and then F but you can do one more thing, you can skip this by reading judiciously because A OR B is corresponding to this you because this I can do it mentally. Writing A bar and B bar and taking its complement knocking of the bars here and removing this AND and putting an OR also can be done in my mind and I can see that whenever two 0s are together this is A bar and this is B bar I can write this as instead of A bar I will use A, I will use an OR symbol using B. That means I will read the map as if I would read a product term.

(Refer Slide Time 14:49)



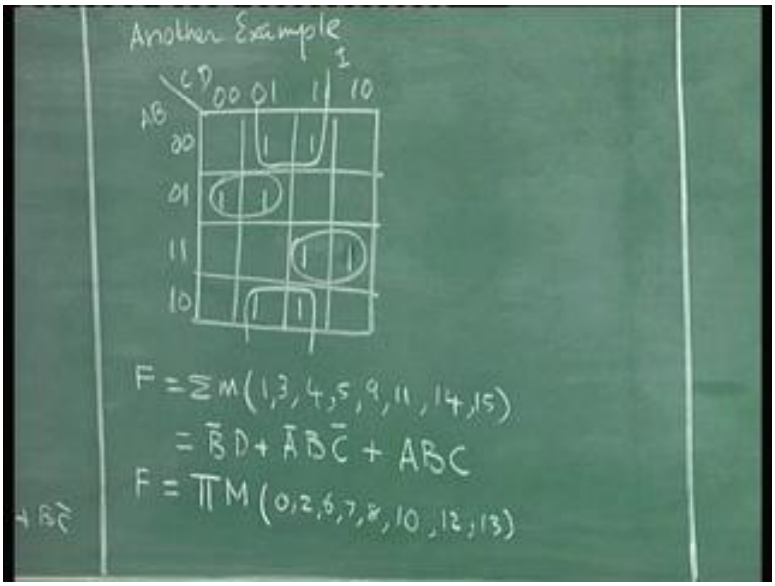
I will read a sum term as I would read a product term except that I will remember to complement each variable and remember to complement each operation. The AND operation will be written as OR operation and variables will be written as its complement. So without going through these two steps which is of course necessary but I am not doing something which is wrong but I am just trying to simplify my procedure visualize the procedure such that these two things are mentally done in your mind so you directly read the map as this. Let us take one more example to prove this point. This time we will have a four variable map sigma m number of min terms will form the sum 1, 3, 4, 5, 9, 11, 14, 15.

This is the map for F (Refer Slide Time: 15:47). I can use any variable I will call them ABCD so the variable for which the min term min terms for which the output is 1 or true or 1 which is this, 3 which is this, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15 this is the distribution. So this is prime implicant one essential prime implicant isthese two, this one cannot be combined in any other way, this is the most efficient way of combining because this one has to be combined so this has to be there, this one has to be combined, I

could have combined this in this way but this is not necessary because I have already taken this one into account and these four 1s form a prime implicant.

Now this first term will be this which is B bar D OR this one A bar B C bar then this one ABC. Simple, you have done this earlier. There is no ambiguity here in terms of non essential prime implicants and all that. On the other hand, if I try other terms have to be put in this so you represent this as sum of product this is a product of sum so you will put a product which is pi, capital M for max terms and small m for min terms so what are the max terms that will be there? Whatever term is not here that will be a max term so there will be a 0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13.

(Refer Slide Time 19:29)



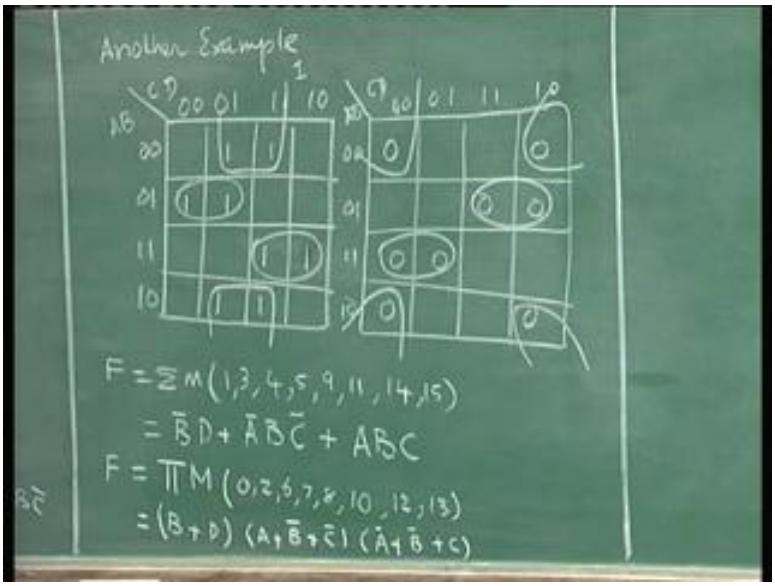
It so happened that I equally chose just not that, I wanted to have a large number of 1s and large number of 0s so it happened when I took this example that exactly there are eight 1s and eight 0s. So you could have proceeded either way proceeded using the sum of product way or product of sum way. If you had a specific reason to go for a product of sum because you wanted to use AND gate fitting to OR gate I will have do it using sum of products and if you are going to have OR gates fitting to AND gates I should go for product of sums. But if you are not given a choice like that if you are not given a condition constraint like that.

In this case there are eight 1s and eight 0s I can go either way. So I will now put the 0s here and I will draw another map here same map but in order to avoid cluttering I will redo it here 0 0 so I can group these 0s into groups of two 0s or four 0s or eight 0s as the case may be based on the case may be based on the adjacency rule and then try to read them. So these four 0s form a group, these two 0s form a group, prime implicant, these two form a group so there are only three terms and I can write F bar and then take a complement of that using De Morgan's identity or De Morgan's theorem and then write the product of sum form bar, as I told you just now a while ago I can directly read the

map for product of sum expression by treating a variable as its complement so when you read it you read it as a complement.

Any variable is read as its complement and an AND operator is read as an OR operator and vice versa. So these four 1s would be B bar D bar if you want to write it as a sum of product. Since I want to write it as a product of sum this will be B OR D. so this will be B OR D. you look at these four as you would do for 1. The only difference is this would be read as B D if you are grouping 1s. When you are grouping 0s B would be read as B bar, D would be read as D bar then AND would be read as an OR. Hence instead of B bar AND D bar I will read it as B OR D that's it. That is a simple trick if you want to call it. It is not a trick really it is a procedure which is bypassing some of the logical steps. This is not as if it is a new concept or anything or it is not a derivation.

(Refer Slide Time 24:04)



If these two were 1s I would read it as B I would read it as A bar B C, read it as A OR B bar OR C bar and finally these two 1s if it is one then it will be read as ABC bar where it will be read as A bar B bar C, this is your minimum product of sum, this is min terms, this is max terms, this is 1s, this is 0s, this will use AND gate fitting into an OR gate for the final output and this will use OR gate fitting into an AND gate for the final output if both are identical. Your choice of using this or that or if you are given a constraint as use many AND gates and only one OR gate you will use this.

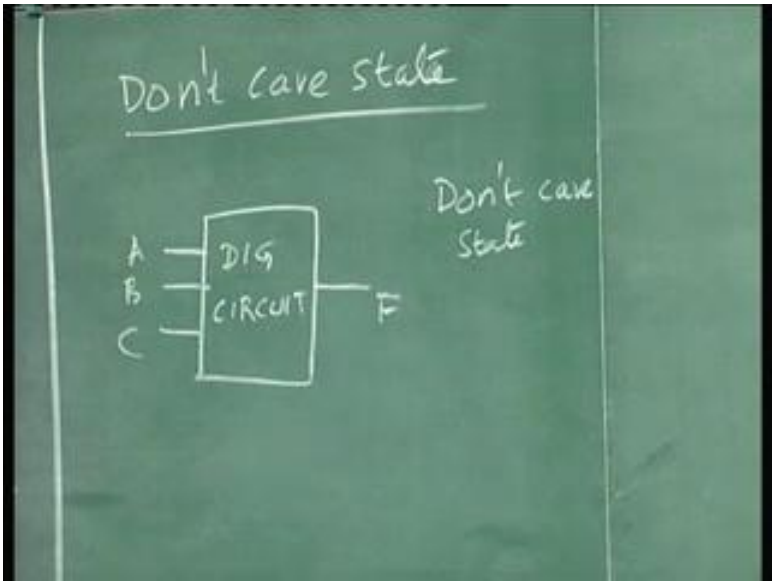
If you are asked to do many OR gates and only one AND gate you will use this (Refer Slide Time: 24:47), if we can use many AND gates and only one OR gate then we will use this. Sometimes these restrictions are designed because of the availability of parts because of the matching of the other part of the circuitry you are designing because you may want to have an inventory of types of parts of the same type but in different designs or different implementations. There may be several reasons why you want to go for this and that. I also told you the other day how to do a four variable, five variable map, six

variable map. At some length we have done a four variable map, and conceptually that is good enough. We talked about implicants, prime implicants, essential prime implicants, non essential prime implicants, how to use a non essential prime implicant properly and all that. We will have to now go to some examples of systems small 1s, we will design and think of some real examples. so far I have been giving you min terms and max term lists without any physical relationship, any physical correlation or anything, it is just arbitrary sum of products, arbitrary product of sums, arbitrary 0s and 1s in the expression or arbitrary 1s and 0s in the truth table.

Before we move on to some of the real life examples I want to give you one more concept which is a very simple concept it is called "don't care condition". This is called "don't care state". What is a "don't care state"? I have a truth table here, the eight rows or a map with eight cells, a map with sixteen cells or a truth table with sixteen rows and we defined for each of these rows or cells an output to be true or false asserted to be 1 or 0, that means you are very clear what you want the output to be for each of the input combinations because based on that condition you are designing a system or a circuit.

On the other hand I may have a system a circuit, I will define the output to be true for certain combinations of the inputs when it has to be false for certain other combinations of the input and there may be some combination which are not covered in both the list.
I can give you a list of combinations in input, how many combinations are possible with three inputs? Eight combinations are possible, out of these eight combinations I will say definitely for certain combinations let us say for three combinations the output has to be true and for another three may be the output has to be false and then there are two combinations which are not defined means you may not define it for many reasons, these combinations may not occur in your system or these combinations may occur but you don't worry too much about that it can be 1 or 0 the output can be true or false because it is not going to effect your process in anyway.

(Refer Slide Time 29:50)



Hence there are certain combinations in any system any circuit, certain combinations input for which you will not define the inputs, the inputs are undefined, it need not be define for practical reasons, such a combination of inputs may not occur in practice or the practical reasons could be such a combination of inputs if it occurs I really don't know I don't really care what it is because that is not going to effect my processing in anyway. Whatever is the reason if you want to say there are certain combinations of inputs for which I do not care that is way it is called don't care state, I do not care what the output is so such cases are called "don't care states".

I will give you a simple example. Suppose I want to count 4 using one hand where in one hand I will use only four fingers all the time and the fifth finger will not be used so I will say, supposing I show a 3 like this you know it is 3 or I put let us say how much I show like this, this is also 2 because these four are the fingers you will have to look at, I am not showing any sign or whatever.

But supposing I put three and this thumb also out I will always tell you to look at my four fingers and determine the condition like 3 or 2 or 1, and this finger you don't even look at but I am showing this thumb you don't even look at but I may put it like this, like this but it doesn't matter because this is not going to effect you because this is 1 or this is also 1, this is also 2 and like that I may have four inputs otherwise three inputs and certain combinations by four inputs is going to affect my output so I would be worried about those, if it happens I have to take some action and certain other combinations if it happens I don't care that thing is going to happen I don't have to take any action. So we can always say 0 we are only interested in certain combinations of the input giving a logical output 1. When that is the case and when any other combination is coming the output can be 0 then you are very safe. I don't want a false output whenever I have a condition for which the output is required to be 1 I need an output to be 1. If there is a

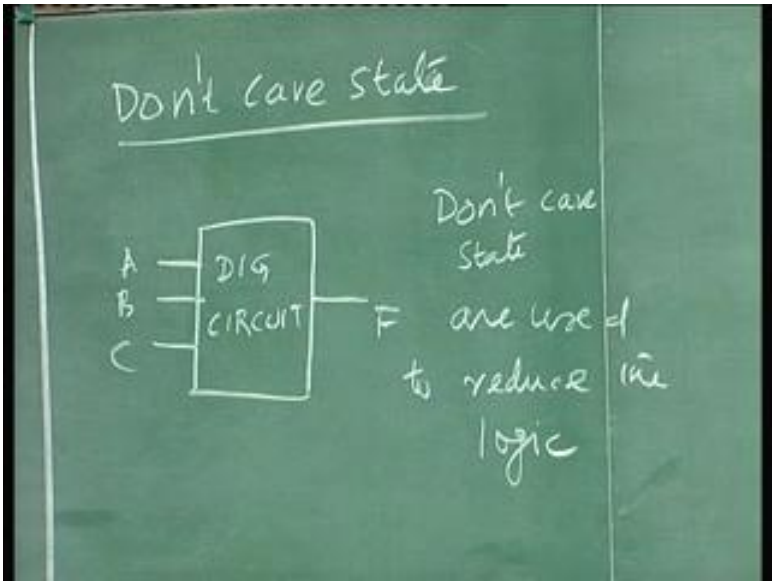
mistake in that I will be worried but for all other conditions I want the output to be 0 whether they occur or not that is one way of looking at it which is fair enough.

I showed you eight combinations in which for three combinations the output has to be 1 I am concerned about this, and for three other combinations I say 0 the you don't worry, in case I can combine all these 3 plus 2 is equal to 5 and say for all other five combinations I can have them and when I don't care about the output what does it matter to you if the output is 0. So I can say for all other combinations other than these three I want for which I want 1 the output is 0 I can do that. That means I will have more 0s and less 1s. Or I can say since I don't care about the output those three combinations for which I want the output to be 1 and those two combinations for which I don't care what the output is all the five combinations always produce an output of 1 and I will make sure that the three combinations for which the output should not be 1 for those combinations the output will always be 0 that is also possible. It is a reasonable argument, either both are reasonable arguments. But if you are doing so I fix this don't care condition the states for which you really do not worry about the outputs the condition for which you do not worry about the outputs you are fixing it as a 1 or a 0 and when you put a 1 and a map you have to cover it you have to enclose it write a term.

On the other hand if you put a 0 my number of 1s in the map gets reduced and when the number of 1s in the map gets reduced my expression becomes more difficult my expression becomes more complex. Fewer once is longer, each term will have more literals if the number of 1s is more. So I would like to use these conditions for which I don't worry about the output, it is an advantage. Whenever I like I will use them as 1 in order to simplify the hardware but I don't have to produce a 1 so whenever it is not convenient to me I will ditch it and use it as a 0. The advantage of this is I can use this don't care condition to simplify my hardware. For example, I have a group of three 1s you put an extra one and make a nice term so there is a "don't care state" and in one of those cells I will use it as a 1 to my advantage to reduce my hardware to reduce my term.

On the other hand if I already completed all my 1s and there are some don't cares strewn around I will not bother to cover them and write terms for it. So use it to your advantage. The don't care states are used to reduce the logic, it is not necessary to use every one of them.

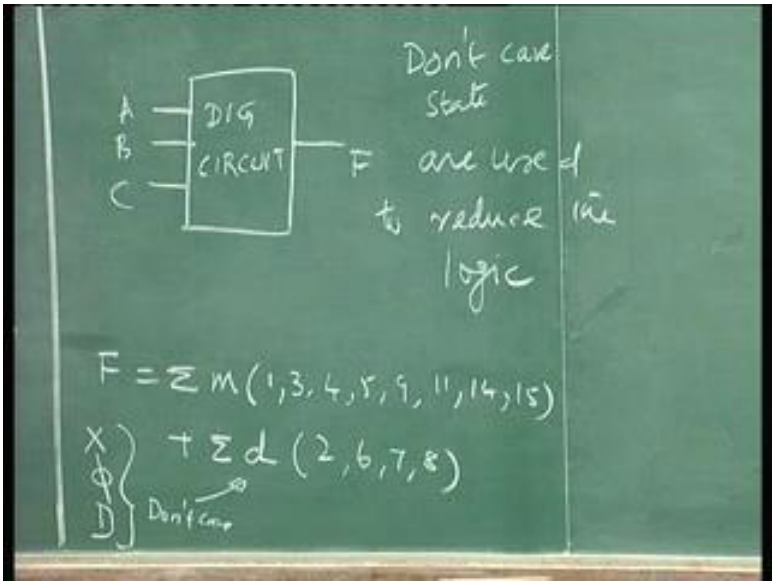
(Refer Slide Time 35:20)



Whichever don't care states are useful to introduction further either the number of terms or in a given term if you can reduce the number of variables we call them literals. I said ABC, AB bar, A bar B bar these are all literals, A bar B bar are all literals, each variable's inner term is called a literal and each is a term. There are two terms with two literals each, three terms with two literals, three literals, three literals. So by using don't care if you can reduce it further either the number of terms or a literal in any of those terms then I will use it to my advantage use it as a 1. Otherwise I will not purposely go and introduce an extra term to cover a don't care. I don't want to introduce a term in order to include my "don't care condition", it is a waste. So the idea is always to reduce the hardware.

The theme of this course as I said in the beginning is reduce hardware, hardware reduction will result in power saving, cost saving, size saving, space saving and everything. That is why we want efficient hardware but reliable hardware, we can't knock it of arbitrarily and say reduce hardware. Usually fifty percent of the terms I have reduced the hardware but you will not get fifty percent of marks you will get 0 marks that is the problem. I see it as a digital course so whether you get a 0 mark it is a 1 mark so it is right or wrong? So, reduction at what cost? It is the reduction without losing the reliability of the circuit. So let us take an example of that class I can use the same thing and put some don't cares. I will take for example the same sigma M, what is the output or 1, what are the terms for which the output is 1? It is 1, 3, 4, 5, 9, 11, 14, 15 I don't want to change any of these but instead of making all other terms as 0s I will say may be 6, 7, 8, and 10 are don't cares. How do I write it? It is d for don't care some books use x, this d will be replaced in some books by x, some books by phi, some books by capital D so all the same don't care.

(Refer Slide Time 38:11)



Don't care terms are, arbitrarily I put that why not I put these four? It is 2, 6, 7, 8 just to show the effect. I don't have an idea of what it is going to look like. From the list of 0s I am arbitrarily assuming these four terms need not to be really 0s they need not be 1s they need not be 0s of the output but they can be don't cares, you don't worry about whether the output is a 1 or 0 for these four conditions of input combinations. So my map gets modified with don't care, this is the modified map modified K map with don't care states.

What are the terms for which there is 1 to start with? These are 1s, these are 1s, these are 1s, and these are 1s. Now I am going to add 2, 6, 7, 8 when I do 2, 6, 7, 8 I don't mark a 1 there. Remember, if I mark a 1 there when I try to simplify this map reduce this map I will make an effort to include every possible 1 that will be unnecessary waste of effort. So what I will do is I will have to use another symbol. if I put a 0 again I may ignore them, if I put a 0 I will ignore it, if I put a 1 I will necessarily have to include it so what I will do is use another symbol I can use d or x or phi whatever is less. So let me use d, this is 2 (Refer Slide time: 40:12), 6, 7, 8. These four ds I have included to indicate the "don't care conditions" of those four combinations of the input. these four combinations of input in the acquired output need not be 1 it need not be 0 they can be anything, they can be either 1 or 0 because that does not effect my circuit performance in anyway because I may not have those combinations of input at all to start with or even if it does occur doesn't matter to me what happens.

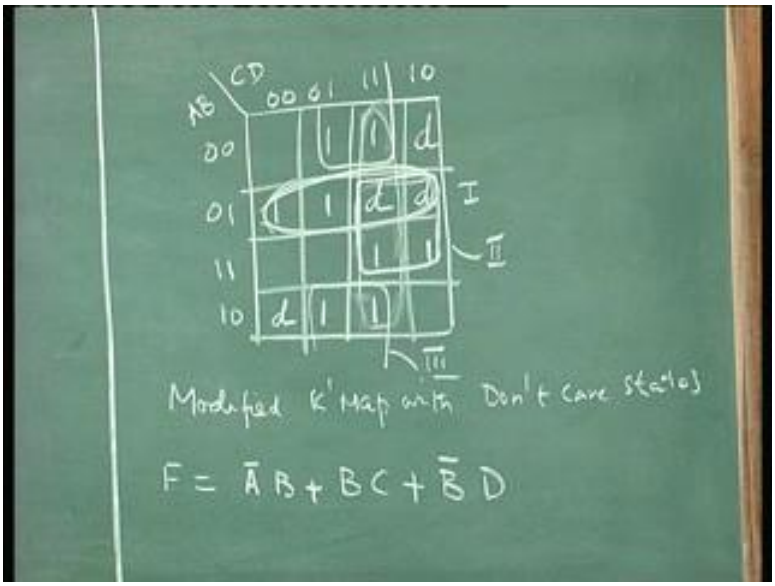
If that is the case then how do I simplify? Now I will have more options, I would probably do this plus 1, 2 or need not even have it I can have these four and these four (Refer Slide Time: 41:28). So this is my 1, this is my 2, this is my 3, prime implicant 1 essential EPI 1, 2 and 3 essential prime implicants.

Now remember, look at this and see that this d has not been included, this d is has not been included so it is not mandatory. It is not necessary to include all ds. At the same

time I use d to my advantage I use these two ds because they are advantageous and I am able to make a simpler grouping of these two ds. If I don't have these two ds I have to use this as one prime implicant which will have three literals. Now I have a prime implicant with has only two literals that means I am knocking one of the inputs of my gate AND gate that is an advantage, it is a hardware saving. So this is the concept of don't cares and how to use don't cares.

Now let us write the expression. Once you have don't cares then combine them we don't have to keep them separate any more, you can just write the expression as if, you write a normal expression.

(Refer Slide Time 43:44)



So one would be A bar B plus two would be BC plus three would be B bar, this is B bar and D. this is what we got now including don't care. Out of four don't cares I will really use only two and I have not used the other two because these two is an advantage, you exploit them to reduce the hardware. But at the same time don't feel obliged to include every don't care. It is going to be a burden to include every don't care because it is going to unnecessarily give you more terms which are not recommended. If I want to combine this d I have to put one more term which is not necessary. This one is already covered you put a d here I will have an extra term with three literals which is a waste one extra NAND gate, one extra AND gate with three inputs and that input has to fit into an OR gate that means OR gate input also has been increased by 1 so all those extra burden is not necessary, it is the same thing with these three.

Originally we had this expression, we now have that expression B bar D is same in both, these two terms have been introduced, each by one literal. These two AND gates will now have the modified form with only two inputs, here we have three inputs so that is the advantage in this hardware so I am having three AND gates fitting into an OR gate. I am not writing the values here you know that this is A bar so you have to come through A

and inverter B B C D inverter so now extra two inverters have to be also included. This is the story of the don't cares.

So now we have exhausted all possibilities of simplification. Of course I have shown you several things we saw in an arbitrary fashion. What I mean is sort of arbitrary, the examples we took. We didn't take any device except don't cares I tried to justify A plus B bar, A plus BC bar, I said can have a combination but justifying after we do this circuit. But now we have to do it the other way.

We have to have a system like any system you want to design what you want from that system for that system you write the truth table that is the design. you identify a physical device system that you want, how many inputs are there in the system, how many outputs are there in the system, what is the input output relationship you want and for the input output relationship you want you write the truth table and from the truth table you can either use Boolean algebra to simplify it or go to Karnaugh Map and simplify it and you can do it the sum of products way or the product of sum way depending on the type of hardware relationship you want to gain.

Therefore what you have to do is to take simple examples of logic functions, arithmetic functions. The one interesting thing is in this course we have always been talking about logic gates logic means decision, true or false or AND and OR etc. For example this function is true why this is called logic because this function is true if and only if both the inputs are true and for all other cases the output is false. So we think of it as a logical statement, logically we can make sense.

OR gate is a logical gate. If at least one of the inputs is true the output is true or an NOR gate you will say the other way. NOR is complement of OR, only if both the inputs are 0 the output is 1 and in all other cases the output is 0.

But then I said these are all digital systems, many times it is also computational intensive. You remember we talked about the computers as the basic building blocks anything can be thought of as a computer. So the basic thing in a computer is an arithmetic circuit. Logic also comes in occasionally but mostly arithmetic add, subtract, multiply, divide and so on.

We will see that later on in this course in subsequent lectures. We can also build arithmetic using logic. Actually they are called logic gates but we will use arithmetic gates like add circuit, subtract circuit and things like that using these logic blocks. Why is logic same as arithmetic because there are only two things. We are talking about binary variables a variable which has only two values 0 or 1 so it doesn't matter if it is arithmetic or logic because when you have two inputs we have to add these two inputs but only one output is possible, the adder will have two inputs and one output let us say we have to add two binary numbers. Two binary numbers you are going to add I have two inputs and one output and each of these inputs can have only two values and output can also have two values. So, if both the inputs are 0 the output is 0 the sum of two 0s is 0

but if one of the input is 1 the output has to be 1, if one is a 1 and the other is 0 the output is 1 the sum is a 1. So sum is 0 if both the inputs are 0, sum is 1 if one of the input is 1.

What will happen if sum is 1 in both the inputs? Then 1 1 is input both the inputs are 1 the sum of two 1s is 2 and I can't represent 2 but I can represent 0 or 1 so output has to be 0. And you should also remember to include another output called carry output which will know that when both the inputs are 1 output will be 0 even though the output is 0 sum is 0 it results in a carry so we will have to know how to handle the carry. But that is the arithmetic part. What I am trying to say is this is a logic function basically, I will have to look at the two values of the input to determine output. If the two values of the input gives me the output so my logic gates can be used for my arithmetic operations.

So what we have done is the basic introduction to the logic gates, introduction to Boolean algebra, we talked about min terms max terms, sum of products, product of sum, Karnaugh Map simplification, don't cares etc but then all of them will have to lead to the realization of things which we need.

Therefore in the subsequent classes we will take design example from combinational logic both logically and arithmetic how we can design and build. that means you have to first design and specify a circuit, specify the inputs and outputs and find what is the relationship between input and output, represent from my truth table and use the tools that you have learnt, Boolean algebra or Karnaugh Map with or without don't cares, sum of products, product of sum, etc then finally come up with the simplification and that simplification will be drawn in terms of true gates and that gate circuit will work as an arithmetic circuit so the logic circuit will become an arithmetic circuit. We will see in subsequent lectures.