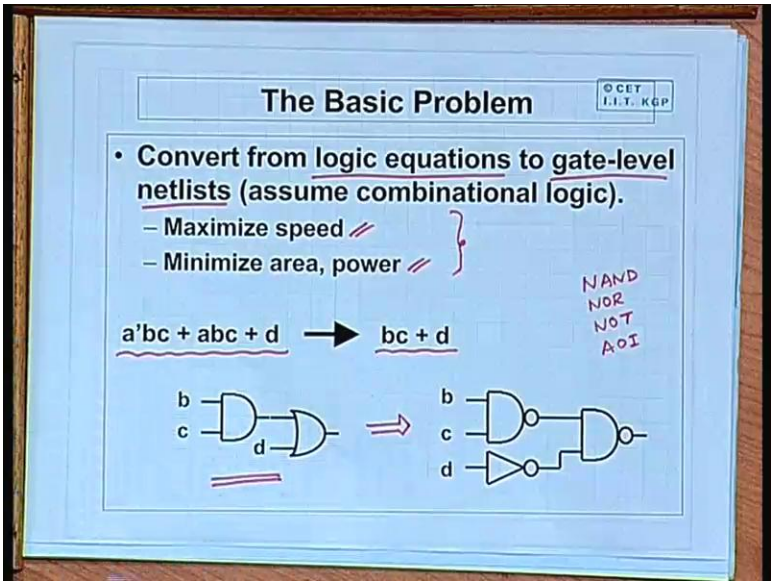
Electronic Design Automation
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 9
Synthesis: Part 2

So, continuing with our discussion on synthesis, today we would be talking in some more detail about the process of logic synthesis. Synthesis at the level of logic.

(Refer Slide Time: 01:20)

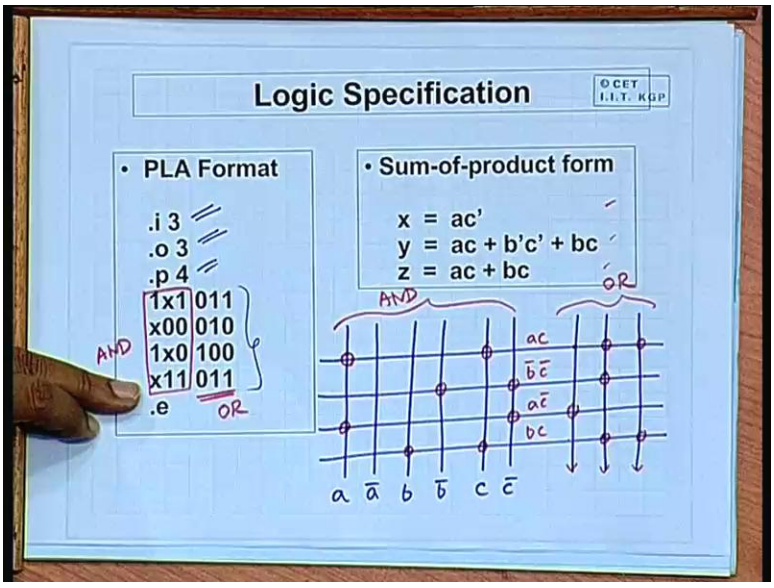


So, first let us try to understand what is the basic problem of logic design or logic synthesis? To start with, we assume that we are only considering combinational logic. The reason is that if we have finite state machine, there are very well-defined ways through state assignment tabular techniques to convert that into equivalent specifications for combinational part and sequential part. So that is a very well understood approach. So, we would be concentrating, right? Now, on the combinational part of the problem; that means how to synthesize or optimize the combinational specification part of the design specification. So the basic problem here is to convert from logic behavior, may be in terms of logic equations to gate level net lists. Now again I reiterate that requirements, may be conflictive, you may need to maximize speed minimize area

power. But everything you cannot get together. So if you maximize speed, possibly you will be paying for more area and more power.

And more over there can be other considerations like, suppose you have a logic equation like, this a simple example. So if you take b c common, from here you can see immediately that this gets minimized to this expression, say in terms of the straight implementation terms of AND and OR, you get this. But if your technology library, if your component library says that you can use only negative gates; negative gate means NAND, NOR, NOT, AND, OR, invert. Then you will have to redesign or translate this design net list into an equivalent net list like this which consist of NAND and NOT. As you can see here number of gates has increased with respect to this, so these are some of the constraints that we need to full fill. So here we cannot do much about this. We do not have AND and OR gates available in our library. We must redesign them using NAND and NOT. Now the next thing is that how do we specify the logic behavior which you want to synthesize?

(Refer Slide Time: 04:14)

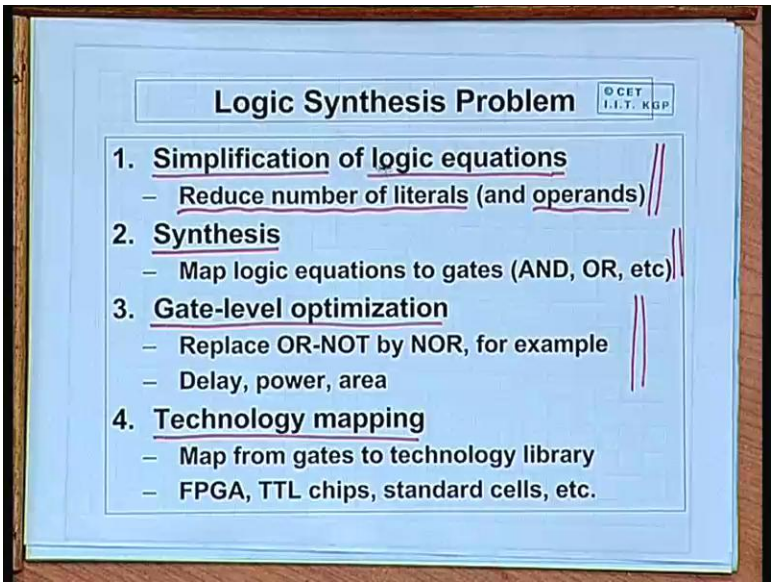


Now, there are many methods. I am showing here two of the more popular methods. One is the so called PLA format and the other is well known sum of products form. Well, sum of product

forms did not require any explanation. It is the standard Boolean expression form. If there are multiple outputs, we specify one equation for each. Now, this PLA format, you know that a programmable logic array can be represented by a so called personality matrix. Now, this particular specification behavior of the PLA says that there are three inputs, three outputs and four product terms. Now, in terms of the PLA layout if I draw, there are three inputs 1, 2, 3, say this is a, a bar, b, b bar, then c, c bar. There are three outputs this, this and this, and there are four product terms 1, 2, 3, 4, and this portion of it specifies the interconnection. See 1 x 1 this part is the AND plane this part is your so called AND plane and this part is the so called OR plane of the PLA. In this portion represents the AND plane and this portion represents the OR plane.

Now, in the AND plane if there is a 1, it indicates that there is a connection with the uncomplemented letter a 0. Means there is a connection with the complemented letter x. Means there is no connection with that letter. So 1 x 1 means for the first one, 1 means it is connected with here, there is a transistor here, b is not connected, 1 means again c is connected and complemented. 0 1 1 means here there is no connection and in the OR plane 0. Means no connection 1 means connection 0 1 1. Similarly for the next one x is not connected 0 0. So b bar is connected c bar is connected 0 1 0 0 1 0 1 x 0 1, means a, is connected x not connected 0 means c bar is connected 1 0 0 1 0 0 x 1 1 x 1 1 0 1 0 1 1. So this is the PLA so there are 3 outputs you can see. So the product term realizes what the first product term realizes a c, the second realizes b bar c bar, the third realizes a c bar and the fourth one, b c. So the first output a c bar second output a c or b bar c bar or b c third output a c or b c you see it is the same specification as this. So these are the 2 alternate ways of specifying the same thing. Sometimes we specify it by PLA; sometimes we specify it by equations.

(Refer Slide Time: 08:12)

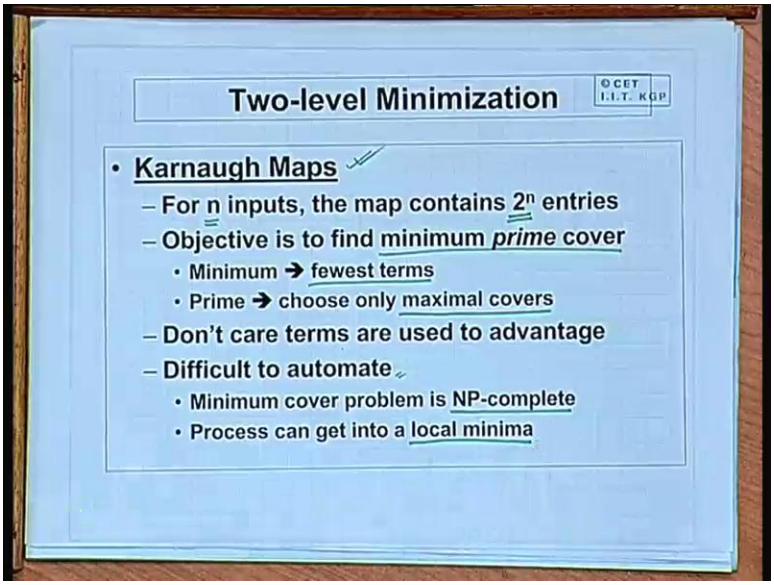


Now the logic synthesis problem, given the input specification in one of these two forms; mainly we have to do or carry out a process of simplifying the logic equations. So in the process of simplification, we are primarily looking at reduction in the number of literals and also operands. This is the first step that we want to minimize the logic equations minimization is one thing. Then we will have to synthesize like all the gates. That means all the expressions in the minimized form may not have equivalent gates at the library. So you will have to map the logic equations to gates. See first one is a purely mathematical step. You are doing minimization of Boolean expressions. Second one you are mapping the expressions to available gates, third one after you have finished mapping them into gates you do some sort of gate level optimization. Like if you have a OR gate followed by a NOT, replace it by NOR or if we have some special considerations for delay or power or area.

You can do some optimization based on the context of the design if you are wanting to minimize delay. You try to reduce the number of levels by combining gates moving gates here and there and so on; and of course after we have finished the gate level optimization. Finally you will have to do this technology mapping step because, ultimately this technology mapping step is essential, so that you pick the correct cells which are available in the library; in the technology library, so

that you can ultimately go into the final physical implementation of the design. Now, this logic synthesis problem has been studied for a long time. There are many classical approaches or methods which are available in the books I mean in the literature. So very quickly we will be looking at a couple of them.

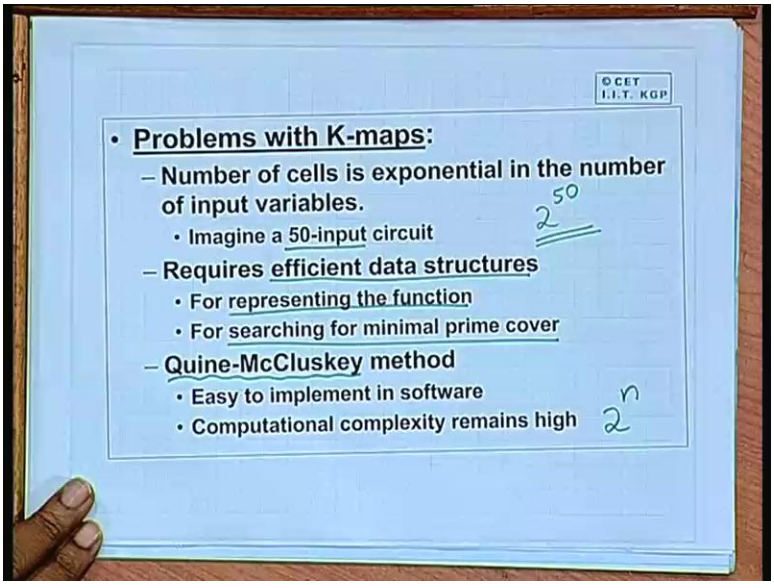
(Refer Slide Time: 10:38)



We will start with two levels minimization. And when we talk about two level minimization, the first method that comes to our mind is the classical karnaugh map method. Now, recall in the karnaugh map method, we construct a table or some kind of a chart. And on that chart all the true min terms are represented by 1 and the 'don't care' min terms are represented by a x. And we try to cover them using as few and as large as possible. That is how we do or carry out the minimization and it is more of a. You can say a visual process that we do. We look at it pictorially and you try to identify the bigger cubes. Now it is not very easy to automate the karnaugh map method by implementing it on a computer for obvious reasons. And another big problem is that, well, as you know the karnaugh map, you can use the maximum five or possibly six variables, not more than that. So in general if there are n inputs in the circuit the karnaugh map will contain 2 to the power n entries. So, the numbers of entries grow exponentially in the number of inputs. This is one big problem that is the size of the problem increases very rapidly.

And of course this is the objective of karnaugh map. I have told you to find minimum prime cover. That means, there should be fewest terms and we have to choose the maximum covers and you try to utilize the 'don't care' terms judiciously so that the cover becomes bigger and the number of terms becomes fewer. Now, as I had mentioned correctly earlier that the karnaugh map method of optimization is difficult to automate primarily because of two methods: one is that it is more of a graphical method which you do through visual inspection and secondly the minimum cover problem has been shown to NP complete problem which is computationally difficult. So, if you apply any kind of a greedy approach, there is always a likely hood that you get into a solution which corresponds to a local minima. But that is not the overall best solution. You get into a local solution. That is not the best possible one. So, finding the best possible solution is not a very easy task for large values of n.

(Refer Slide Time: 13:29)

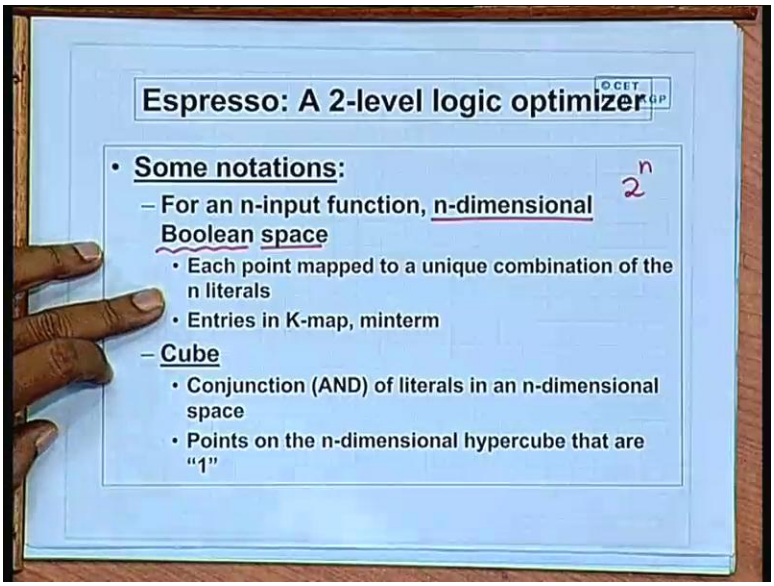


And of course again looking at the problems of k map, once again the number of cells you have that grow exponentially in the number of input variables. So a 50 input circuit is a very practical one. So just imagine you will be having 2 to the power 50. So many cells, this is simply impractical. So if you want to have something similar to a k map implementer, in a computer you

will not implement it just as a map. As a picture you will have to have very efficient data structure for representing the functions and also for searching for minimal prime cover. So, one such approach is the Quine Mccluskey method. See Quine Mccluskey method is essentially a systematic way to approach the problem. But it is not really much different than k map method. In k map we are doing it visually in Quine Mccluskey method. We are trying to formulate a systematic approach. This is this method. Since it is systematic it is easy to implement in software. But easy to implement does not mean that it is efficient computational complexity still remains high.

So if there are n numbers of variables, the total time complexity can easily go up to 2 to the power n. This is one thing which we cannot eliminate in Quine Mccluskey method. So what is the solution? Now, the solution is that this karnaugh map or the Quine Mccluskey method, the basic you can say the premise of this method was that I want to find the best solution. So, let us try to find out a systematic method to do that. But practical considerations say that since the minimum prime cover problem itself is n p complete. So it is futile to search for the best solution always; particularly if the problem size is big is large. So, rather it will be much better if we have a method which can give us very good solutions in reasonable amount of time that very good solution does not mean that it is the best possible. But it is very close to the best. So, one such method which is in fact a very popular method used for two level optimizer is a software package which is called Espresso.

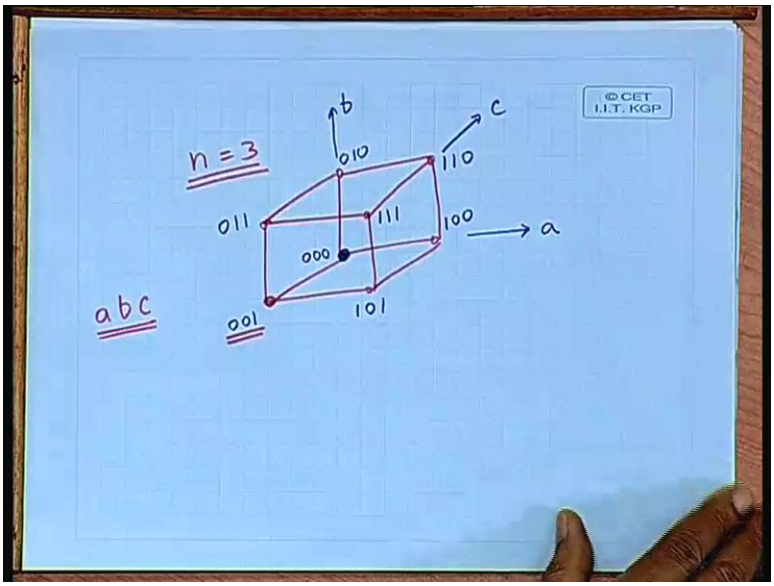
(Refer Slide Time: 16:18)



This uses an approach like that you will see that, in this method there are some steps which may appear to be funny. Because those steps are not anything through which we are trying to improve a solution. But we are expecting that if we do this in the future, we may lead to a solution which is better this is more like a random search kind of approach. You are following with the expectation that we will be landing up in better solutions in the future. So and again in these algorithms, there is nothing like this is the end of the algorithm you reach here and you finish this is the last.

Say as long as your computer time budget permits you can go on running the algorithm, you will be getting better and better fine. Now before trying to explain the algorithm let us introduce some notations for an n input function I had mentioned that, there are two to the power n possible min terms. This can be represented by an n dimensional space. This is a Boolean space because along each of this n dimensions values can be either 0 or 1. And each point in the n dimensional space corresponds to a unique combination of n literals.

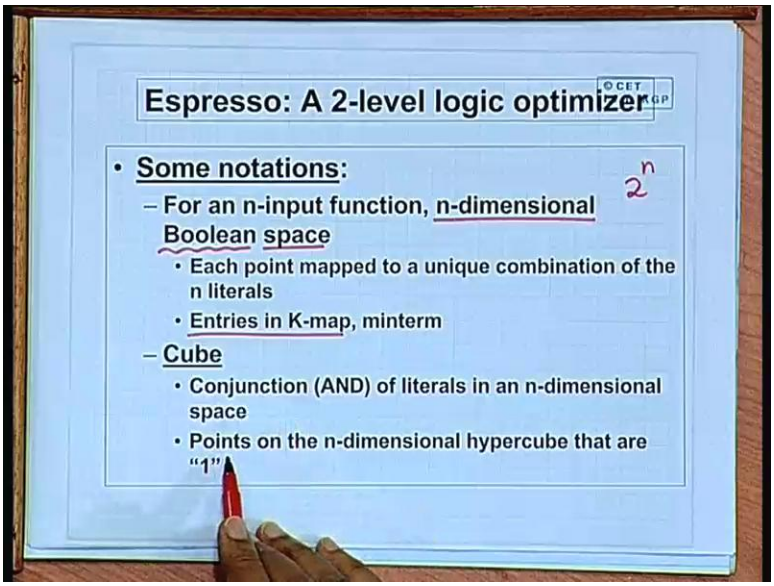
(Refer Slide Time: 17:50)



Let us take an example. Suppose we take n equal to 3. So for n equal to 3 the solution space will look like this a cube. Now, in this diagram, suppose the three input variables a, b and c and we assume that this is our origin. The origin means, this represents the point where the a b c values are 0 0 0. Suppose, this is the dimension corresponding to a, this is the dimension corresponding to b and this is the dimension corresponding to c. So whenever we are moving along a edge in a particular direction the value of the corresponding variable changes state.

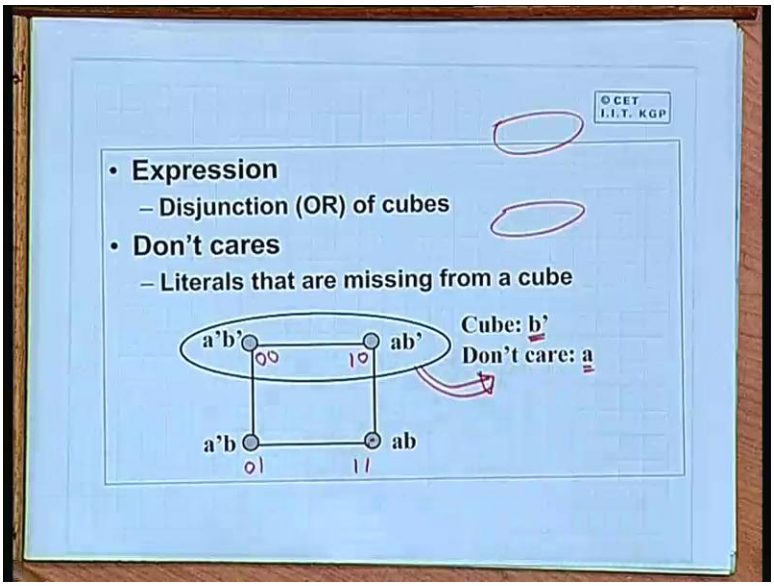
For example here it will be 1 0 0, here it will be 0 1 0, this is the c direction. Here it will be 0 0 1. This is again direction of a this should be 1 0 1. This will be direction of b. So 0 1 1 direction of a will be 1 1 1 direction of c should be 1 1 0. So this is what we are trying to say that this, we will look at for n input function n dimensional Boolean space. Each point mapped to a unique combination of the n literals. So each point of the n dimensional space is mapped for example this point is mapped to this combination 0 0 1 a equal to 0 b equal to 0 c equal to one. So it is the same for all other point fine.

(Refer Slide Time: 19:58)



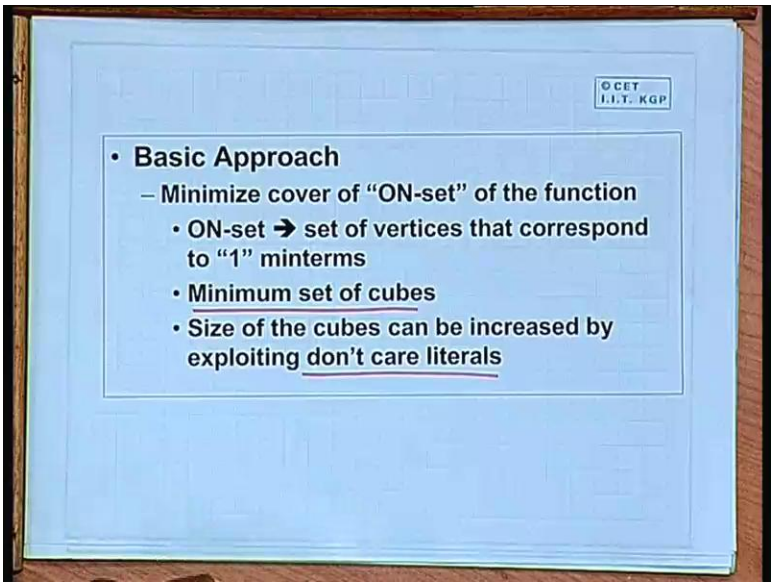
So, this each point in the space is with respect to the karnaugh map it represents the entries. Each entry represents a min term and in this n dimensional space if you take a conjunction of some literals. For example, in this n dimensional space if you take a combination of these two literals, say 0 0 1 and 1 0 1 together. This we call as a cube. This definition of the cube is very similar to that for a k map. So, the cube is a conjunction of literals and that each of the literals are the points that are true. We will take only those points which are the which correspond to the true min terms.

(Refer Slide Time: 20:55)



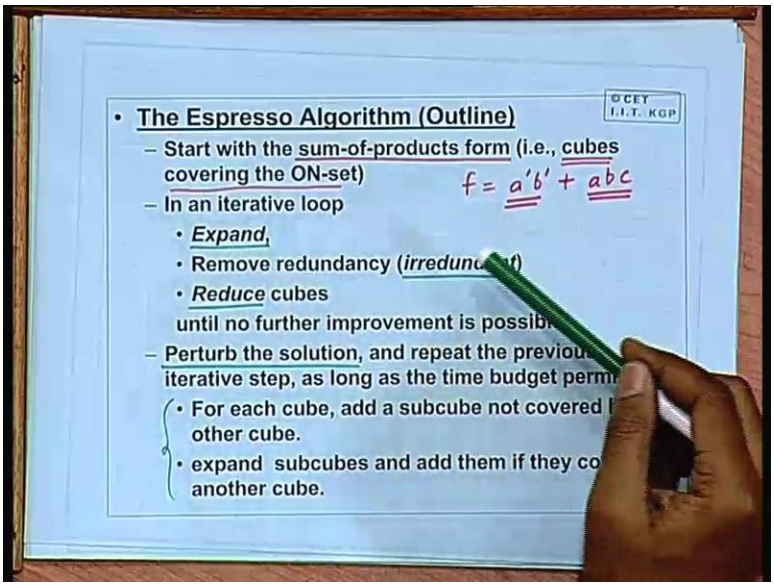
And when you say that there are expressions, so there will be a number of such cubes, we will be taking in the disjunction or OR of that. May be we have 1 cube like this another cube like this. So this or this well some of the combinations may be 'don't care'. Like, suppose for a 2 cube, there are only two variables a b. These may be the combinations say this is 0 1 1 1 1 0 and 0 0. This is a cube. Say now when you take a cube which consists of a bar b bar and a b bar they combine together. This becomes only b bar. So with respect to this cube a is missing. So with respect to this cube we say that 'don't care' literal is a, it is not that I am talking about, 'don't care' min terms I am talking about 'don't care' literals. So when you have a bigger cube some of the literals are missing those literals are said to be 'don't care' literals.

(Refer Slide Time: 22:08)



So the basic approach behind Espresso is to try and minimize cover of the onset of the function. See, on set means, the set of vertices that correspond to 1 like in, k map we take only those min terms which are true min terms and try to cover them. So the approach is very similar. We take only the two vertices which we call the on set and it tried to obtain a cover for them using minimum number of cubes and bigger cubes. Minimum set of cubes is of course one requirement and we can utilize 'don't care' literals like in k map again. So that is either the cubes can be made bigger all right. So, now let us see what the Espresso algorithm is essentially all about.

(Refer Slide Time: 23:09)



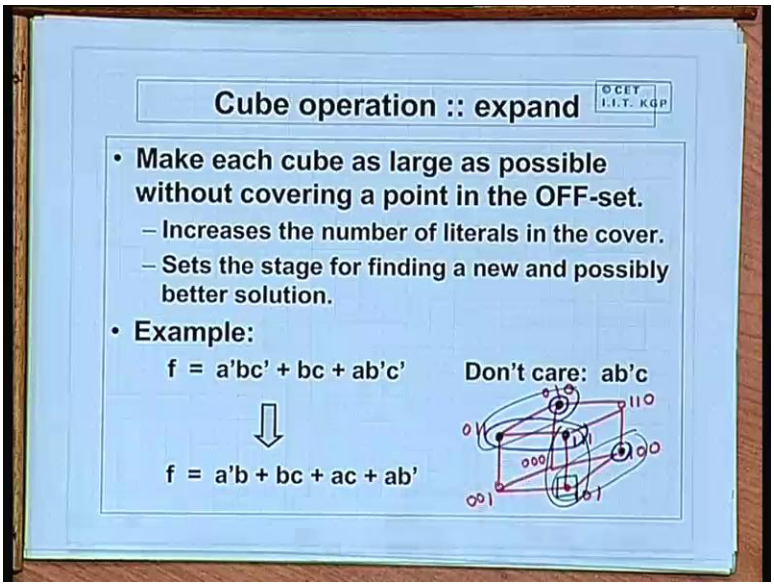
So, here you present the outline of the algorithm to start with we have the sum of products form of the expression. So, when you say it is sum of products form, so each of the product term corresponds to a cube which covers the on set like if you have a function like this f equal to a bar b bar a b c, a bar b bar corresponds to a cube and a b c corresponds to another cube. So it is a sum of products form. This is one cube. This is another cube. Now this algorithm repeats 3 steps in an iterative loop until no further improvement is possible or your time budget runs up. Now, these three steps; I will be explaining these three steps are called expand, irredundant and reduce. Well, as the name implies expand means you try to increase the size of the cube expand. Second step you try to reduce or remove some redundancy. Say get after the expand step you may be having some cubes which are not necessary which have become redundant.

So you try to remove them. The third step you try to reduce the size of the cubes. This may be, well it may be funny. But you really do this. You can reduce the size of the cube with the hope that well, right? Now we may be worsening the solution. But this worsening may lead to a betterment in the near future; that is the espresso. So these three steps are efficient and in the meantime in the say during this loop. You occasionally make some sudden changes to the solution make some perturbations. Well we will be explaining that two ways. You do

perturbations. I will explain these 2. Now the objective of this perturbations or to make sudden changes to the solution is essentially to try and see that we do not get locked into a local minima. Well, if you make sudden changes to the solution, it is likely that the solution comes out of the local minima and goes somewhere else. From there if you repeat those steps, we will be going towards a better solution.

So this is an approach which tries to do that. Now as you can see from the steps, that it is not a very deterministic algorithm that we start from somewhere. We stop at a point with the guarantee that whatever we have obtained is the best. No it is not like that. So we try to iterate we try to make random changes with the hope that we will be getting a better solution. But our hope may or may not be full filled in all cases. But it has been found that for most of the circuits we can get a good solution in a reasonable amount of time fast enough. So now let us try to explain what are these three steps expand irredundant and reduce.

(Refer Slide Time: 27:06)

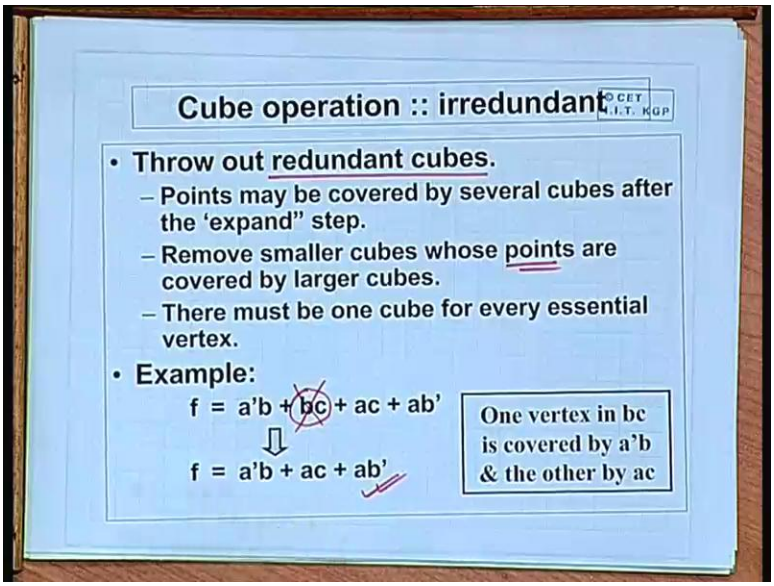


The cube operation expands says that you try to make each cube as large as possible without covering a point in the offset. Well so if you have a smaller cube we try to make it bigger you this. So here you see lets take a example second is draw that cube. So 0 0 0 0 0 1 0 1 1 0 1 0 1 1

1 1 1 0 1 0 0 1 0 1. So in the original one, it says that a bar b c bar. That means 0 1 0 this 1 b c b c means these two things taken together a bar b c a b c a b bar c bar 1 0 0. This was the original cover. There are three cubes after the expands step and also there is a ‘don’t care’ literal a b bar c 1 0. Suppose this is ‘don’t care’. ‘Don’t care’ I am marking green, this is do not care. So after the expand step what you do? You try to expand all the cubes in whatever directions you can.

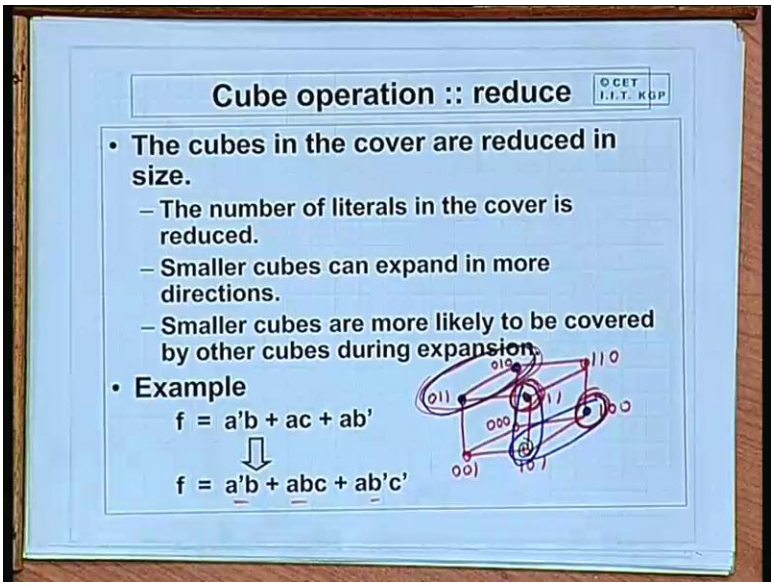
Well you see that you can expand in a number of different ways like this 0 1 0 which is there is a single cube in different color. Suppose you can combine this with neighboring, see these are the on states these are the true min terms. So you can make a cube like this. You can make a cube like this. You can make a cube like this. So in the all possible ways you can expand the size of the cube you do that and for doing that you can utilize the ‘don’t care’ literals. So, now here 1, 2, 3 and 4 cubes which correspond to a function like this. So this is essentially what this cube operation expands. See, here we are simply trying to expand the existing cubes in all possible direction it can. But we are not checking whether the cube cover we have obtained in the process is redundant or irredundant. For example in this one, suppose if you take this cover and this cover then this cover is not required. These two points are already covered by other two. So some of this, may be removed. But in the expand step we do not do that we simply try to make the cubes bigger and removing the redundancy that will leave to the next step.

(Refer Slide Time: 30:33)



The step is irredundant. So, the expand step will expose some redundant cubes. Now, in the second step irredundant step we try to remove or throw out the redundant cubes. So redundant cubes are those for which the points are already covered by other cubes. Those we can simply throw out. Now, in the earlier example well as I had mentioned we can throw out 1 of the cubes b c. We can throw out these we can retain this 1 this 2 and this 1 3. So starting from this you can identify that b c is redundant, we remove this we get this. So, after this step irredundant we get a solution where the redundancy is removed, right? So, you can expect that after expand and irredundant we get a solution, which is you can say to some extent minimized. Well we have explored the expansion of the cubes and we have removed the redundancy. But you understand we still are following a greedy approach. We are not looking at the problem globally. So maybe we have or we have reached a point which corresponds to a local minima. So the next step the reduce step it tries to explore other alternatives.

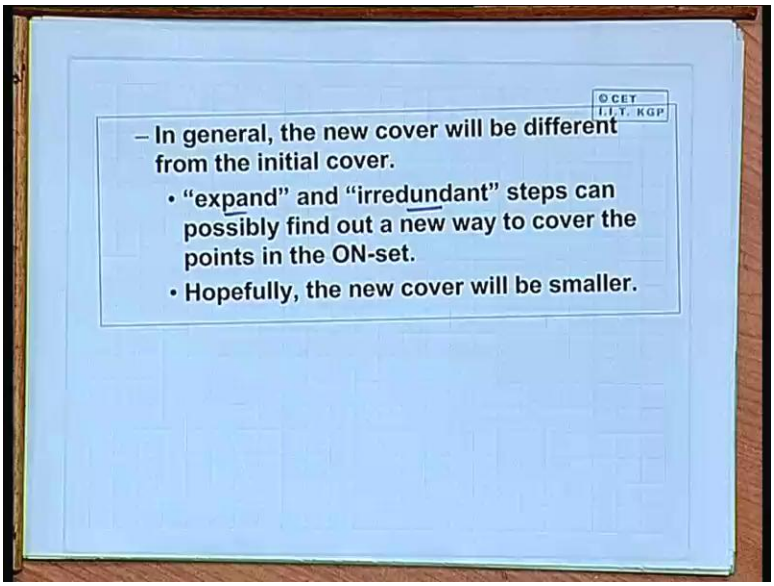
(Refer Slide Time: 32:18)



Reduce step says that the cubes are reduced in size. Reduced in size means we are again increasing the redundancy. So a cube which is bigger, I make them smaller like say again let me draw that cube. So here we had 0 1 1 0 1 0 1 1 0 1 0 0 1 1 1 0 0 0. So a bar b was this. These are the two main terms a c this was the 'Don't care'. This was the 'don't care' and a b bar this one. Now, after the reduce step what you can do is you can retain a bar b. This is retained and the other ones we can retract. This, a c I can simply make it smaller I make it only this a b bar. I can simply make it this. So, some of the cubes may be made smaller by removing the 'don't care' terms or by taking out some point which is covered by some other cube.

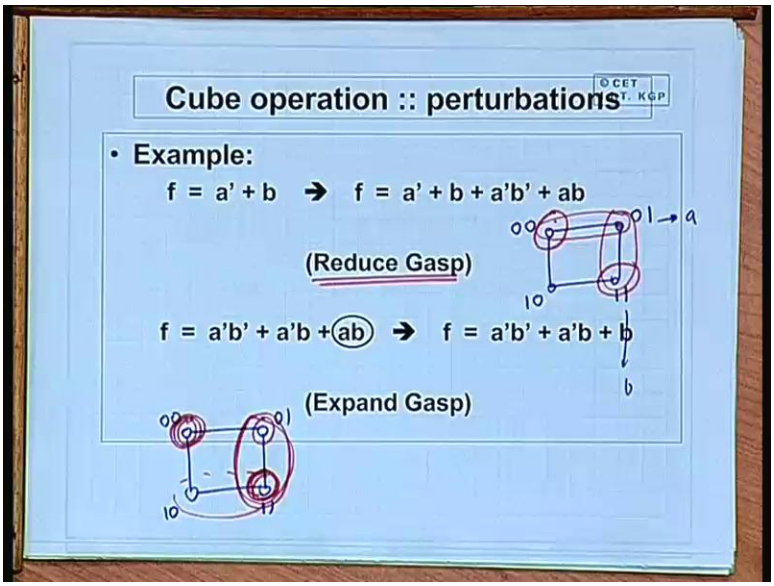
So in this way some of the cubes are making smaller. So now I have this 1 big cube and 2 smaller cubes these 3. Now you may argue that why we are doing this we are making the solution worse. Yes, we are making the solution worse. But our intension behind doing this is that, some of the cubes we have made smaller and in the process we have opened up possibly an avenue. So that this smaller cubes can expand in other directions in the future may be if this was also a 'don't care' term. Now, this cube can expand in this direction, right? So we are now open to other possibility that is why we are doing this.

(Refer Slide Time: 35:11)



So after this step, the new cover will be typically different from the initial cover. So as you, as I mentioned the expand and irredundant step after you do this. After you do this, reduce if you again apply expand and irredundant step. May be you will be moving to another solution which can be better. But there is no surety that it will be better. But you are exploring other alternatives. So as you repeat these three steps what happens is that you are moving from one area of the solution space to the other. Your total solution space is huge. It is not possible for you to explore everything. But you are trying to move from one part to other using these three steps and every time you always memorize that what is the best solution I have seen so far. So after certain number of steps or after certain time have elapsed you check or you see if what is the best solution you have seen so far and you report that to be the solution to be taken accepted. Now, as I mentioned apart from these three steps you also carry out some kind of perturbations or some kind of sudden jerks to the solution.

(Refer Slide Time: 36:48)

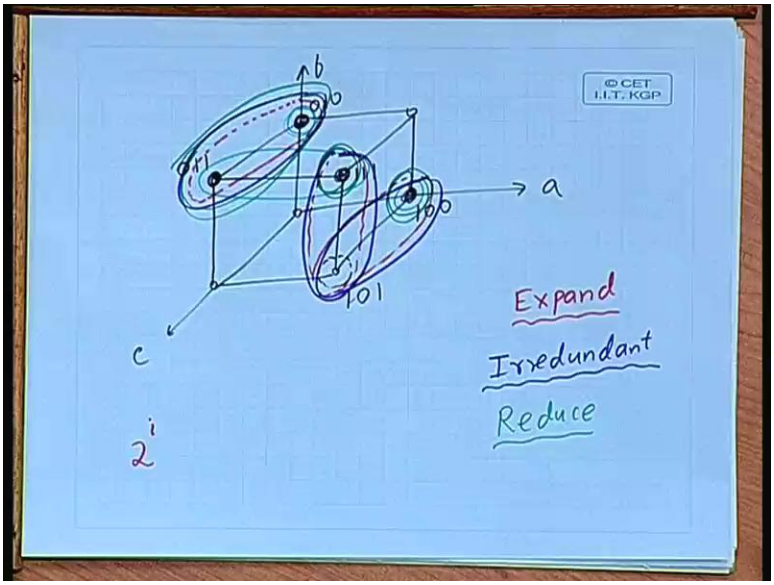


These are applied periodically in between, in order to well even move out from one part of the solution to the other part, so that the chance of falling or getting stuck into a small area of the solution space gets reduced. You can go out to some other parts of the solution and explore. Now here the idea is something like this. Well this is a very simple solution. Suppose I have a two variable cube a and b simple example say 0 0 0 1 1 0 1 1. This is the direction of a and this is the direction of b. Well, a bar is this. These two points b is this. Now, the first step, well this is not a step. This is one kind of perturbation. This is called reduced gasp, this is the name. It says that you retain these two cubes as it is, but also add some more which are not common like a bar b bar. Let this also be a cube a b, let this also now you have 1, 2, 3 and 4 cubes.

So one which is common to both ‘don’t’ ‘do’ that, but with the others you also add them as a cube. So now you allow them to expand in all other directions. So this is some what similar to reduce. But you are exploring all the possibilities together and expand gap is a similar step. Here also I am showing. Suppose I have this map and you had these three cubes. You have this a bar b. You have this a b. You have this. So now it says that a bar b bar you leave this as a cube, leave this, leave also this. But you take this one, the other one you combine either with this or with this. So now there will be three cubes 1 2 3, this is called expand gasp. Of course after expand

gasp you can have a thing where one cube becomes the proper subset of the other. But still you retain this so that during expand phase the small cube may possibly expand in other directions, right? But you can see these are not any step which is used to improve the solution consciously. But essentially you are trying to get a situation where you can explore steps which explore other steps other ways of expanding or moving out. So let us take a small example.

Refer Slide Time: 40:14)

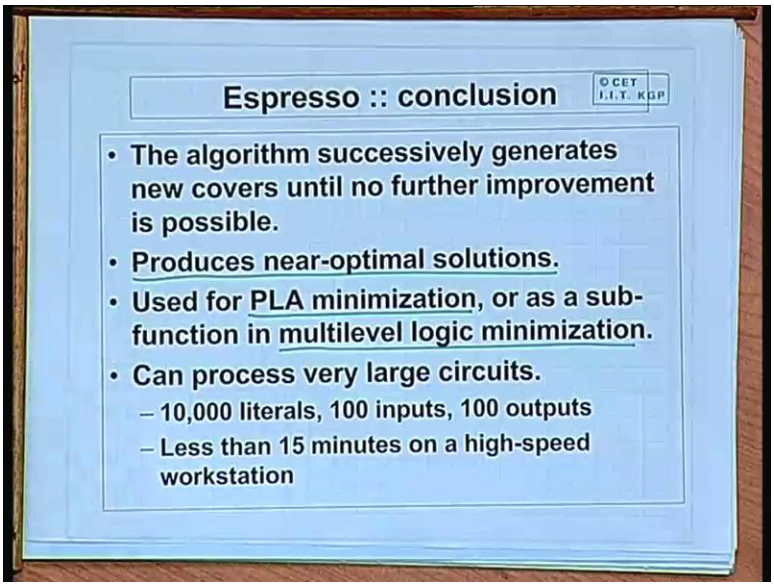


An illustrate, the different steps of this espresso algorithm. Let us illustrate with a 3 cube. So let this be the direction of a and this is b and this is c. Say, I am only showing the true min terms. This is a true min term this is a bar b c 0 1 1. This is true min term, this is another true min term 0 1 0. This is another min term, this is 1 1 1, this is another, this is 1 0 0. The same example and this one is a ‘don’t care’ I am showing it as dot 1 0 1. Now in this example, what I said just I am repeating just to show all these steps together that suppose to start with we have a cover where the one cover is this itself one. There is another cover where this one is alone there is another cover where you have these two. These are the three cubes. Now, the expand step what you can do is the first step is expand. I am just trying to summarize what we are doing in the expand step. What you really do is that you see that in which direction the existing cubes can expand like you look at this bigger cube.

You see this cube cannot expand any further because the cube can cover only 2 to the power certain number of nodes 1 2 4 8 in that way. But this was a single term. You can say another true never. So you can combine these 2 into 1. So in the expand step you can take these two together. You can also utilize the ‘don’t care’. This single one, well although you cannot expand this bigger cube. But this ‘don’t care’ one you can take along with this you can have this as 1. Similarly this one can combine with this do not care. So now, we have 1 2 3 and 4 this is expand. Now, after expand what I have said that you see that you know how all possible expansion. But you see there are many cubes which contain end vertices which are already contained in other cubes, right? So some of them you can eliminate. So for example, you can eliminate either this one or this one. But you cannot eliminate this because it contains this vertex or this. But for the other two, the end vertices are contained in the other. So you can eliminate one of them.

So the irredundant step tries to do that. So after irredundant step possibly you will be getting a solution where you have this, you have this and you have this. This one you remove you have these 3 cubes. Now after that as I have said that when you do that reduce reduce is just the reverse of expand. That will show expanding some of the cubes you contract. So the example I have said that you keep this you return this. But these two you remove and you make them smaller. Let this cube contract to this and let this cube contract to this. So, these are the different steps which you can use in the espresso and the espresso algorithm. As I said it works in this way and you can expect to get a good solution in a reasonable amount of time if you follow these steps.

(Refer Slide Time: 45:12)



Now, to summarize this espresso algorithm, what it does? This algorithm successively generates new covers. Well, until no further improvement is possible, this step you can use if you are not using the two perturbations. That means these two perturbations are mentioned reduce gasp and expand gasp. If you do not use them, then beyond the point you will see that you are not getting any further improvement. But if you use them, then you can drastically get away from the solution and you can explore newer portions of this search space. So essentially, what you do there is that, instead of saying that I repeat till no further the solution is possible, rather than that I will say that I will repeat till my time budget exhausts. I will run my program for 10 minutes. So within 10 minutes whatever best solution I get, I will take it and the good thing about espresso is that it had been.

It has been tested over very large number of problems and it has been found to produce near optimal solutions which are very much acceptable. Now since espresso is a two level optimizer, it produces the functions in sum of products form. So to talk about two level circuits, the first kind of circuit which comes to our mind is the PLA. So this is used either for PLA minimization or it is used in a different context. Even for a multilevel logic minimization sometimes we need two level optimizers. So this espresso can be used as a function which is called from within a

multilevel logic optimizer and statistically it has been found that this espresso can process extremely large circuits, say of the order of 10000 literals, hundreds of inputs and outputs. But even for circuits of this size it does not take more than 15 minutes on a typical high speed workstation.

So what this means is that, this espresso algorithm is an acceptable algorithm which many people uses in practice. And as you see for many other algorithm that we will be studying or looking at which later that in CAD in VLSI CAD electronic automation design, there are several different steps where the solution space is very huge. Solution space is huge and it is not possible to explore all portions of all portions of the solution together. So there we go for a compromise. So instead of trying to go for the best solution, we try to get a good solution, but within a reasonable amount of time. So we will see that, well the way espresso algorithm works will give you a hint of what we really do. See espresso algorithm there are some steps which you can appreciate. Well you are moving towards the correct solution. But there are some steps where we are moving away from the good solution. So that, is the crux of these classes of algorithms that sometimes we move towards the good solution. Sometimes we move away from a good solution. Our expectation is that in the process.

We will finally land up in another solution which is better than the one we have seen so far. Now here we have seen or we have looked at this method of two level optimizations. Now the question is that is this two level optimizer is sufficient or we need to have some methods where the number of levels needs to be more than two. Well, it is not very difficult to answer this question. See if you think of some classes of circuits. We will give an example in the next lecture. There is some classes of circuits which can be implemented in two level realizations no doubt. But the number of resources and the complexity, well, when I say number of resources and complexity, it may mean number of gates. It may also mean the size of the gates because I told you earlier that bigger the gate slower will be its speed. So there are circuits for which 2 level implementation will be very complex to implement. But if you go for a multilevel implementation you will see that there can be a drastic reduction in the amount of resources you require and in practical situations there are many circuits which are like that.

So in practice, people look for some methods or solutions which actually try to do an optimization taking not two, but more than two levels of logic. So of course delay is a parameter to be considered, but also there is another thing as I told you earlier, we will be repeating. You also have to look at the size of gates you are using. If you say, if you say that if you use more than one levels, more than two levels, but the sizes of the gates are less, then possibly overall on the whole the delay of the circuit will be less and not more. May be number of levels is more, but the gates are smaller and simpler. So in our next lecture we will be looking at some aspects of multilevel logic optimization. And we will see the basic approach which follow the problems involved and what are the main. You can say the directions in which the people approach the problem. Thank you.