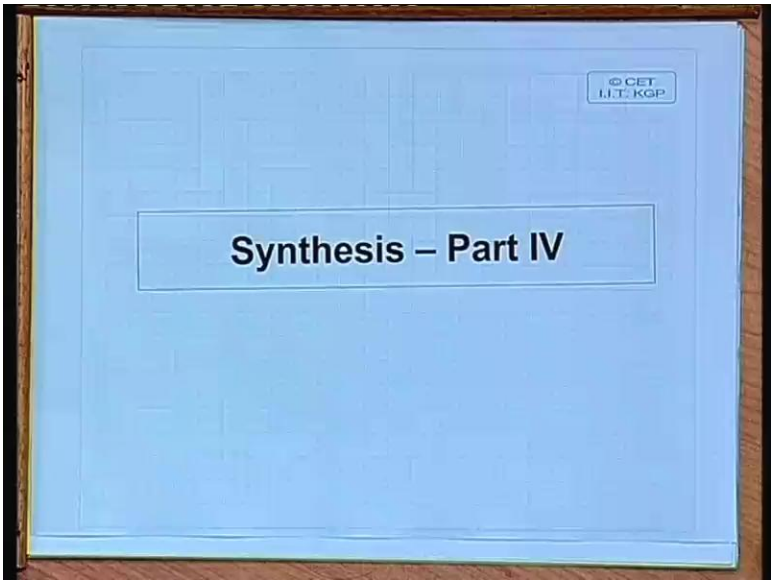
Electronic Design Automation
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture No #11
Synthesis: Part 4

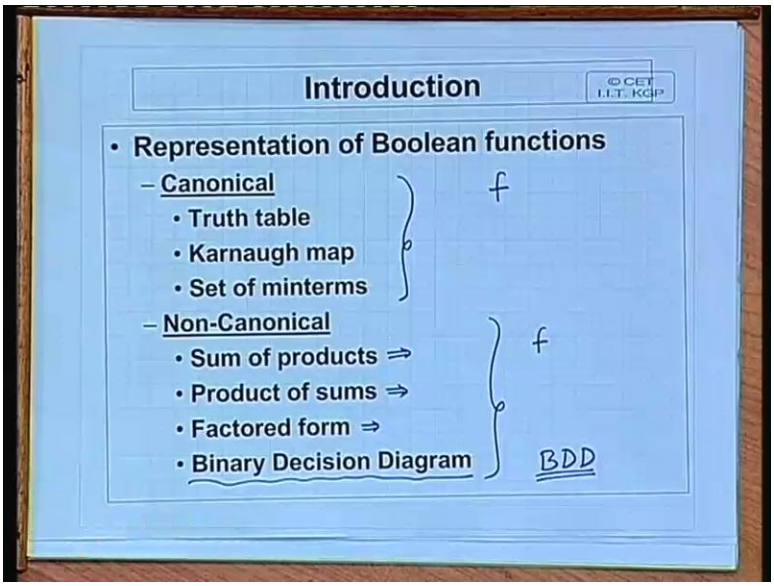
Today in this lecture we continue with our discussion on logic synthesis. Now if you recall in our last lecture, we had looked at the several schemes for carrying out logic synthesis for both 2 level and multilevel realizations. We talked about the essential idea behind methods like Socrates or MIS, M I S. Now in this lecture today, we shall be looking at an alternative method using which you can also perform logic synthesis.

(Refer Slide Time: 01:43)



In fact this method is more general than powerful not necessarily used or applicable to logic synthesis alone. I have much wider range of applicability.

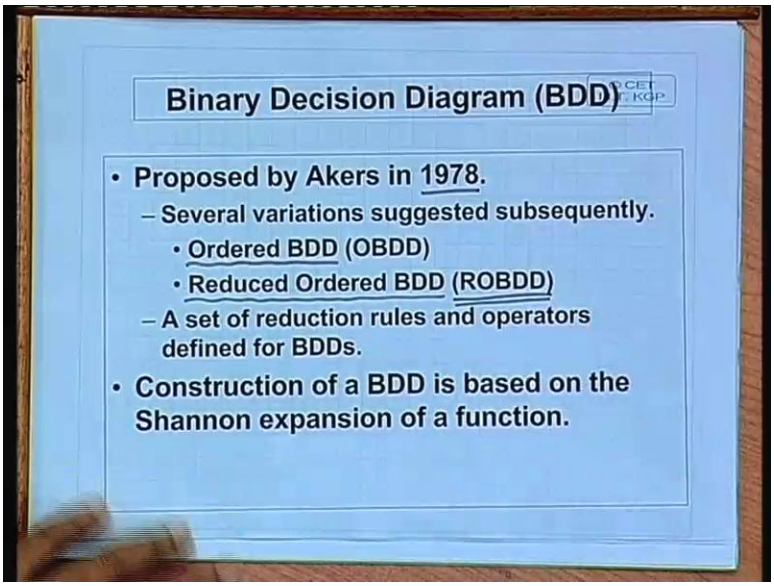
(Refer Slide Time: 01:59)



Before going into the method first let us look at or talk about some alternate representations of Boolean functions. Well when we are talking about representing a Boolean function there are broadly 2 ways of doing canonical and non-canonical. Canonical means these representations are unique given a particular Boolean function f. You can have a unique truth table or unique karnaugh map or a unique set of min terms okay. That is why we call this as the canonical representation. But however there are again several representations of a function f where you can have several different ways of doing. So for example, sum of products given a set of min terms.

You can combine some of them into bigger cubes to perform minimization and after minimization it is possible to have several alternate sum of products form for the same function. Similarly for product of sums and again a given function f can be factored in a number of different ways. You can have several different factored forms. Now here we are interested in the representation called Binary Decision Diagram which is called BDD in short. Now you will see BDD is also not a canonical representation in the sense that the BDD the nature and the look of this diagram can vary depending on certain parameters. So here we will be looking at the binary decision diagrams.

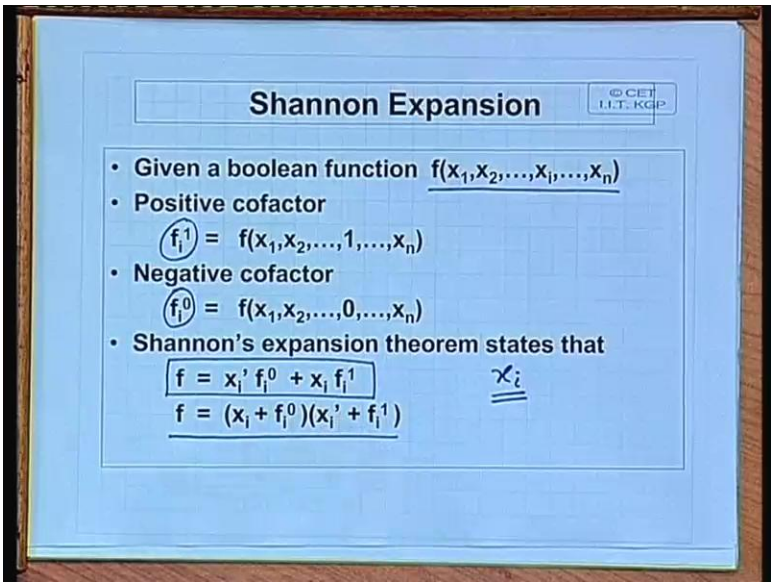
(Refer Slide Time: 03:55)



BDD or the binary decision diagram was initially proposed by Akers as early as in 1978. Now it was primarily. Now, the original definition of BDD were subsequently defined and several variations were suggested. Well 1 of the ordered binary decision diagrams where kind of ordering is imposed we shall be seeing it and after you get this ordered binary decision diagram, there are some operators which are defined using which you can carry out some minimization or reduction on that diagram is called reduced ordered BROBD.

So actually for practical purpose we will be using ROBDD for and manipulation. So in order to get this ROBDD starting from a BDD, so a set of reduction rules and operators have been defined which can be applied successfully on. Now the way a BDD is constructed, you see it is essentially based on the so called Shannon expansion of a given function. So first let us try to understand what this Shannon expansion is and then we shall be explaining through examples how a binary decision diagram can be constructed okay.

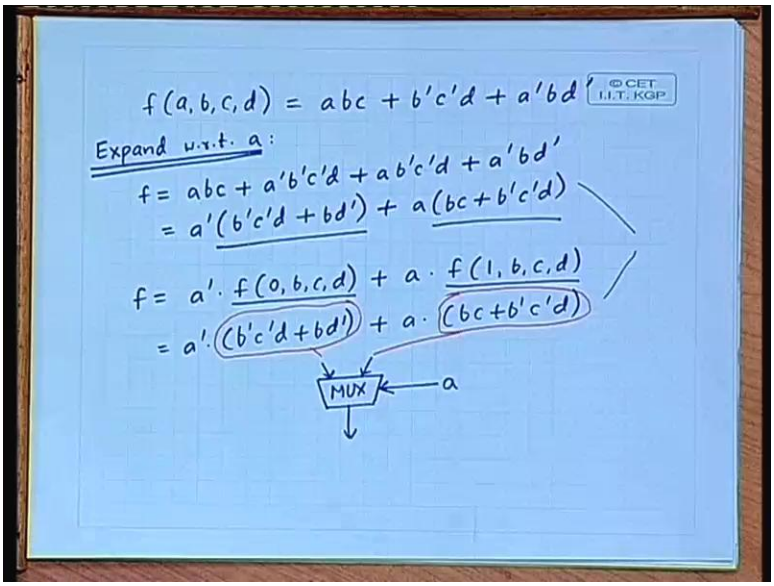
(Refer Slide Time: 05:38)



So Shannon expansion for a function well the definition can be given like this. Suppose we have an n variable f is the function of x1, x2, etcetera xi. Now we define positive and negative cofactors as false. A positive cofactor x suffix i 1 is the same function where the input variable x i replaced by a1. So, when i input is replaced by a constant 1, then the resulting function is called the positive co factor is denoted like this. Similarly if the i th variable is replaced by a constant 0, then we call it negative co factor represented like this. Now according to Shannon's expansion a function f when expanded with respect to a input variable x i can be written like this equal to si prime negative cofactor is with respect to i or x i versus cofactor 1.

So this is the expression which we will be making use of when we will be constructing binary decision diagram. So essentially it means that i take out x i prime and i can replace ith variable by 0, i can take out x i which is the ith variable by 1. Now it can be proved that this and if you are more interested in the product of sum representation it can be alternate form x i or cofactor x i prime. So in this way you can represent or expand a Boolean function with respect to any input variable. So let us take a simple example to just illustrate.

(Refer Slide Time: 07:54)



Suppose I have a function of 4 variables say a b c d then the function is a b c or b prime c prime d prime or a prime b prime okay. Suppose we want to expand this function with respect to input variables. Well first let us try to see how we do this normally. Normally this function can be like this well a b c is already there the second term is independent of a. So we can write like this a prime, b prime, c prime, d or a b prime, c prime, d. Actually we are introducing a, a prime and a similarly this contains a no problem.

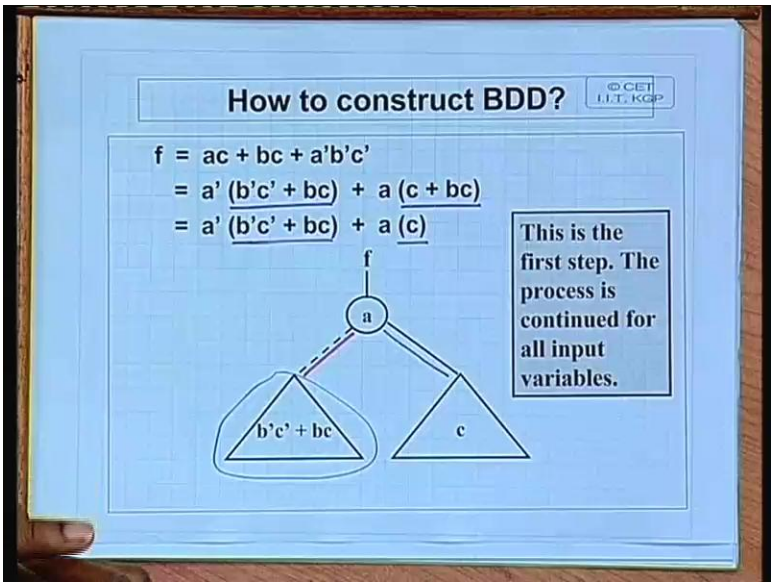
So by doing this what you have achieved is that each of the product term contains some term of a either complemented or uncomplemented. So after that we can simply take a prime common b prime c prime d or b d prime you can take a common b c or b prime c prime d. So this is the so called Shannon's expansion with respect to a. Now you can check even with respect to the formula just we have mentioned this one. If we apply the same rule, what happens see, same thing if you do with respect to the negative and positive cofactors you see the formula once more. It is x i prime multiplied by the negative cofactor. So what we can write as for the formula a prime will be negative cofactor that means the function f by a, is replaced by 0 or a the function f a is replaced by 1. Now you can find out the values of this negative and positive cofactors from this original expression. Let us see a prime. If you replace a by 0, just put a equal to 0 here this

first term goes away you have b prime, c prime, d and this becomes only b d prime. Similarly for the second 1 replace a by 1, this term goes away. This becomes b c or this.

So see actually it is the same thing whether you do it using logical manipulation or you directly apply Shannon's formula, it is actually giving you the same formula. So you understand 1 thing starting from the given function you can do or carry out the expansion with respect to an input variable a or b or c or d depending on which variable you are using for the expansion. Well you will be getting different forms of the function. Now the another thing let us also understand in briefly that why we are doing or carrying out this kind of an expansion when we are saying that our objective is to synthesize. See when we expand the function with respect to a, you can think it like this that we have a multiplexer, we have a multiplexer which we control with the input variable a, now if the input variable a is 0, then some input is selected if it is 1 some other input is like this. Now somehow what we will be doing, we will be feeding the first input with the circuit realizing the first sub function. We will be feeding the second input by a circuit realizing the second sub function then our purpose is served. So in this way we can have a very synthesis procedure where the circuit can be implemented by a network of multiplexers.

Similarly these functions are there which are independent of a, they may be expanded by some other variable say b or c or d and you can repeat this procedure. We will be having a tree of multiplexer's fine. But before going into that first let us see what a, binary decision diagram is now. And how do we construct it okay fine.

(Refer Slide Time: 12:55)

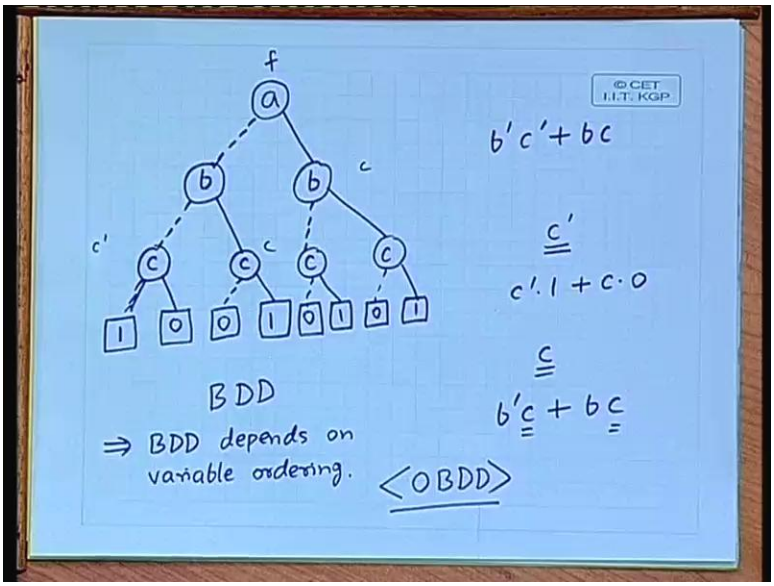


So actually binary decision diagram follows very closely from the Shannon's expansion we have just mentioned. Suppose we have a given function f like this and through Shannon's expansion we can follow the same procedure. We can write the function like this a prime. This will be the term with it and a, this will be the term with it. This you can very easily say because the middle term you can write a prime b c or a b c a prime b c will come with this and a b c will come with this and c or b c can be minimized which is only c. So this is the ultimate. Now in a binary decision diagram the way we show this that initially the function f was there.

We are expanding the function with respect to the input variable a. Now this we show like this by a diagram. Well a, is the node of the diagram node of the figure. This is actually a tree the root is the function f it feeds the first node a, and you see that a, has 2 arcs pointing downward 1 is dotted other is solid. Well you can also show them by different colors if you want that is a matter of convention. Say suppose we show the dotted ones by red and the other 1 by blue. This means the dotted 1 means that if the value of this variable a, is false. I mean 0, then you will be following this path. If the value of a is true, then you will be following this path. So according to this formula the left path this block will correspond to b prime c prime or b c and this block will correspond to simply c. This you can repeat. Similarly this sub block you can expand with

respect to b and we will be getting a diagram. So we will take an example okay. This example we can take no problem.

(Refer Slide Time: 15:14)



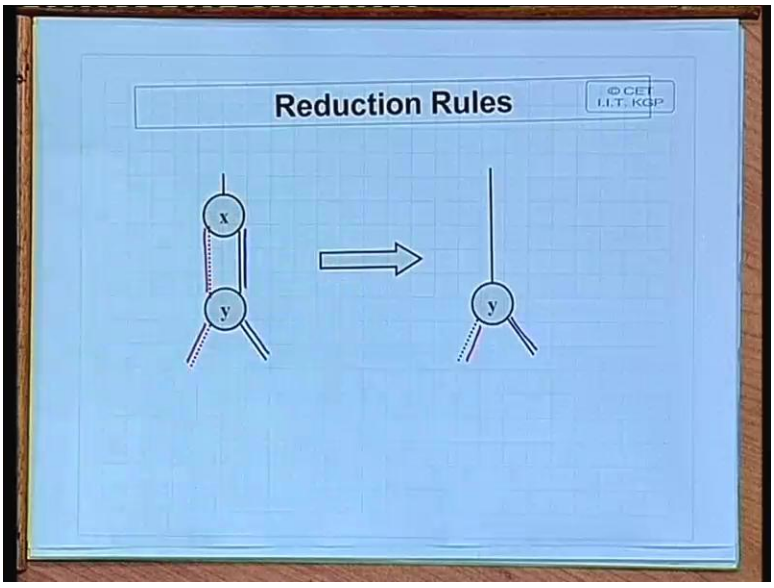
We had this a we had the dotted arrow and the solid arrow if we recall the left side was b prime c prime or b c; b prime c prime or b c. Suppose we choose to expand this with respect to b so we will be getting a node b out here. So if b is prime c prime comes with it. This side we will get c prime and this side we will get only c. So remember this side c prime this side c. The next step here we will be expanding with respect to c. Now we have only c prime here okay so this can be written as c prime into 1 or c into 0. So we will be getting final term c dot final terminal nodes 1 solid 1 this 0. Similarly the other side, so if you expand with (( )) (16:24) and here similarly this side was only c. So c can be written as (( )) (16:50) both side (()) (17:03).

So this is the complete binary decision diagram we have constructed. So you see that you are given a function f from the function f we go on expanding using Shannon's theorem the function successfully with respect to some input variable say here we have taken first a then b then c. So ultimately as you go down the 3 finally we will be landing up into nodes which corresponds to

constant zero's and one's; that is the end of the story. So the tree will be constructed till we reach the final level where the leaf nodes are all the constants 0s and 1s.

Now I already mentioned that this binary decision diagram is not a canonical representation means it is not unique because the same function if you have tried to draw the BDD by using a different variable ordering, like you have taken b first, then a, then c, then your diagram would have been different right. So the point you notice is that the binary decision diagram BDD depends on variable ordering. So if for a particular case you fix up the ordering like I will take a first then b then c if the ordering is fixed then you say you have got something called ordered binary decision diagram OBDD. So actually this is a case of OBDD provided you have fixed the ordering a b c. Now once we have obtained a OBDD there are several steps you can follow to minimize or reduce the size of the tree. So first let us look at the reduction rules then we will take an example to illustrate.

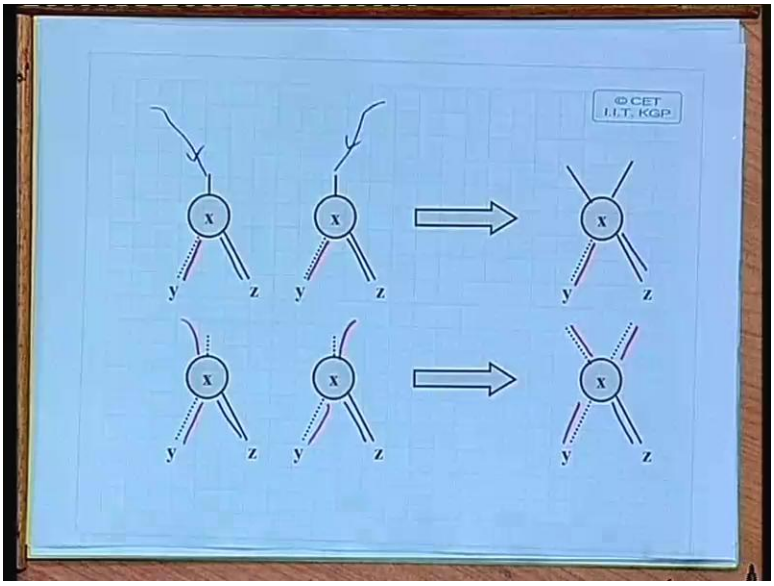
(Refer Slide Time: 19:41)



The first reduction rule is this. It says that if in the graph you have 2 nodes like this where these are the negative edges or the dotted lines these are the positive edges, then if from a node both negative and the positive edge is leading to the same node in the second level. Then you can omit

this x because this does not depend on x whatever is the value of x will come to here. So after reduction you can omit x and you can have this only node y fine. So this is 1 reduction rule. If from a node both the odds lead to the same successor you can omit that node.

(Refer Slide Time: 20:39)



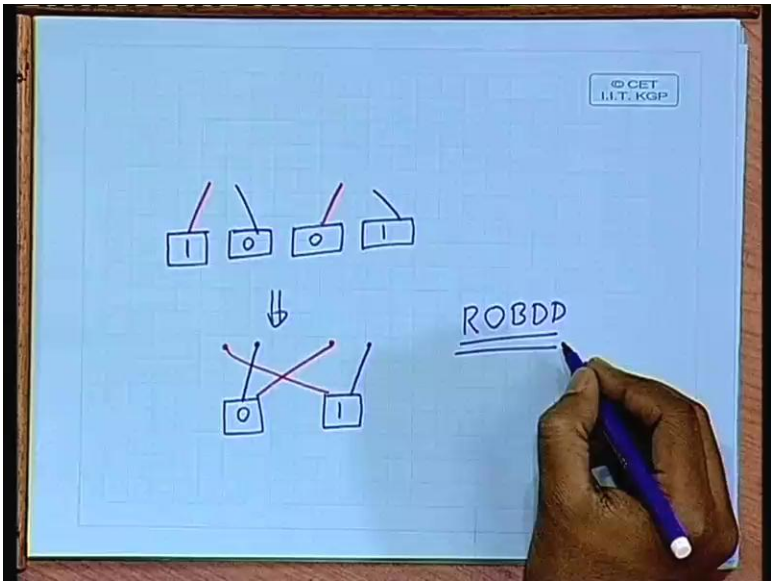
And the second set of reduction rule say that if you have a scenario like this where there are 2 different nodes, both leveled with x whose successors are same say y and z both are y and z. Then say the sources may be different this may be coming from 1 place. This may be coming from some another place these are 2 sub graphs in your BDD. Now here what you do you combine these 2 nodes into 1 the successor links are merged together. So instead of having 2 different arcs to y, you have a single arced y. Similarly single arced z and these 2 sources from where arcs are coming to here and here you put them both into the same node. Yes.

Student: The 2 sub nodes happen both at the same level?

Not necessarily, so actually when you are carrying out reduction you will see that the 2 sub nodes need not be necessarily at the same level. But if they are labeled with the same x they have to be at the same level initially. Because the order in which you are expanding a b c. So all a's

are at the same level, b's are in same level, c's are in the same level okay. Well and the second set of rules is identical the only thing is that the arcs which are coming in they are negative arcs. But the rest are the same these 2 are negative links these 2 are positive links. So you can merge these 2 and this is the positive link. So if we have nodes like this where the successors are the same you can merge them together provided the sources are also of the same type. But if 1 was dotted other was solid you could not have combined okay. And 1 last way of reduction.

(Refer Slide Time: 22:52)

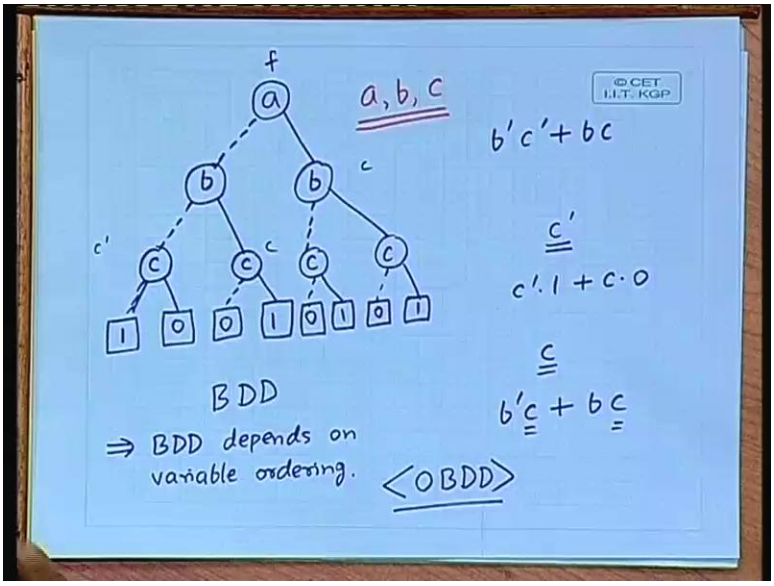


That is of course trivial. See at the lowest level I am taking the smallest example at the lowest level, you have only constants 1 and 0 say 1 0 0 1. Say suppose you had this kind of thing but there was arc coming from top level to here then to here then to here then to here say these were the 4 arcs coming. Now what you do what i am saying that instead of having so many terminal nodes leveled with either 0 or 1 you have a single 0 and a single 1 merge them together and there are 4 different sources okay. 1 here, 1 here, 1 here and 1 here. Now this source was coming to 1 so take it to 1 this was coming to zero.

So take it to 0 this was coming to 0 take it to 0 and this was to 1 take it to one. So this way you complete the edges. But this 1s and 0s you merge together. So once you apply all these steps

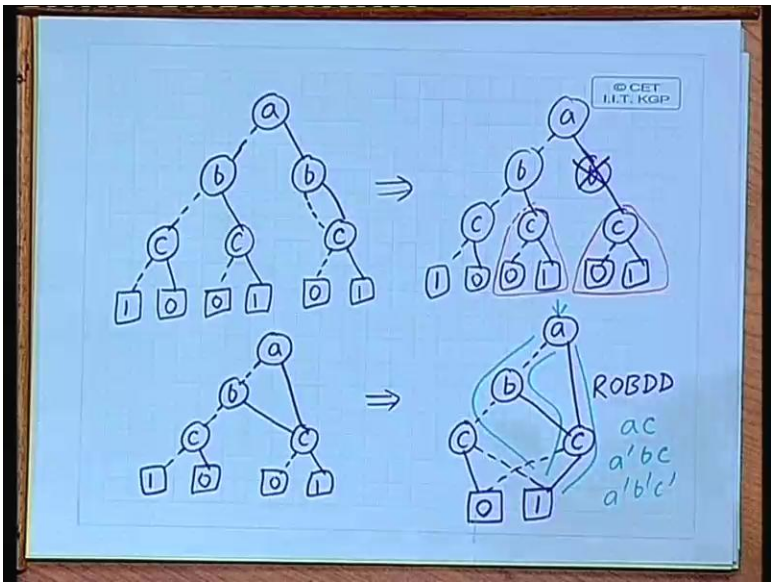
repeatedly you get a diagram which is smaller in size it is reduced and this is what is called reduced ordered binary decision diagram ROBDD. Now let us try to illustrate ROBDD with that same example i had shown you earlier so let me take that same example.

(Refer Slide Time: 24:41)



This was the original BDD which I had drawn this was drawn with respect to a specific variable ordering first a then b then c. So starting from this diagram let us now try to systematically see that what are the rules, we can apply and we can minimize. 1 thing we can see, see these 2 nodes they are identical. You see these nodes and their child they are identical. So what we do I am drawing the modified diagram. So from this diagram I am just showing the modified one.

(Refer Slide Time: 25:46)



So initially you had a then you had b. Now on the left side you had 1 out here 0 out here 0 out here and 1 out here. But on the right side you just see that on the right side you had something like this you had under b c 0 1 c 0 1 the same thing. So what you do instead of having 2 copies of them we merge them into a single copy. So we have c 0 1 and from b you see there was 1 dotted arc coming and 1 solid arc coming. So we add both dotted arc and also a solid arc. This diagram captures same information that from b if you have a dotted arc you come here. If you have a solid arc also here because since the sub trees are identical we can keep 1 copy and you can point just all the arcs together. Now after doing this, now you find that you get a situation where both the complemented and uncomplemented arcs are pointing the same node. So you can omit b.

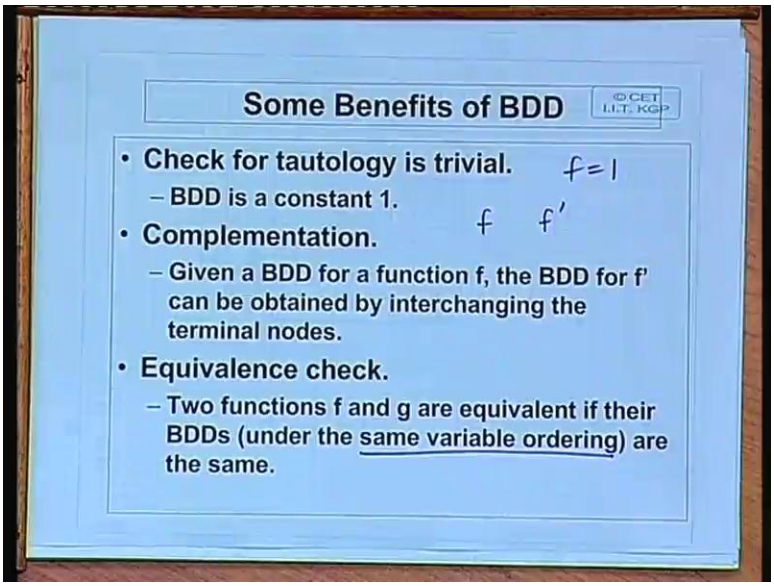
So in the next step you get a diagram left side still remains same. But this side from this b is no longer there b we have omitted from a you straight away come to c. From c you have 0 and 1 so this 1 node you have omitted. Now again you observe that in this diagram there are some sub graphs which are identical this 1 and this 1 c with the dotted arc pointing to 0 and this solid arc pointing to 1 both are the same. So again following the same rule we will be only using 1 copy of this and will be pointing to 2 edges the there. So after this modification the graph will look like this, this part remains same. But here you have only 1 c 0 on the left side 1 on the right side. You

have arcs from a coming here like this also arc coming from b so there also arc coming like this. So you see as you carry out the transformation the tree becomes a graph it is no longer a tree now okay this becomes the graph. So you inspect that if you can do any other transformation. You can see that it cannot be anything else as far as those rules only other than margin 0s and 0s you cannot do anything else. So in the final step you merge this 0s and 1s into a single 0 node and a single 1 node. So you have 1 c out here and a c out here. So at the lowest level you have 1 0 node and 1 1 node.

So from c dotted from 1 solid to 0 and from here dotted to 0 solid to 1 this is your final ROBDD. Now here the point you notice is that the BDD representation is not canonical. I told you but OBDD is once you have fixed up the ordering the BDD will look the same and if you apply the reduction rules the ROBDD will also be canonical against a given variable ordering. The variable ordering are fixed up then this ROBDD is also a canonical representation. Say the same function if we have 2 different ways of representing it which means we have 2 sum of products form. But if you construct the ROBDD with respect to the same variable ordering, we will finally land up in the same diagram okay fine. OBDD will be the same because what is OBDD on I am expanding the variables if this is true, it will go this side. If it is false, it will go to the other side and other thing if you look at this diagram again. You will also observe another thing that in this diagram you see this is the route and finally you are landing up into the node 0s and 1s.

See if you finally land up in one, it means that for this particular input combination the function is true 0 means it is false. Now you try to find out what are the different paths from the route up to one. Now in this case you will see that the paths are several there is 1 path like this. There is another path like this there is another path like I think that is all these 3 parts are there. So from this you can straight away write down the function also. For this it will be a c both are solid arcs the first min term will be a c means the first product term will be a c the second 1 will be this is dotted a bar solid b c a bar b c, this 1 a bar b bar c bar. So just by scanning the BDD or the ROBDD you can also find out that under what input combinations you will be getting the function as true. And you can do some really for the false under what combinations it is zero.

(Refer Slide Time: 33:02)

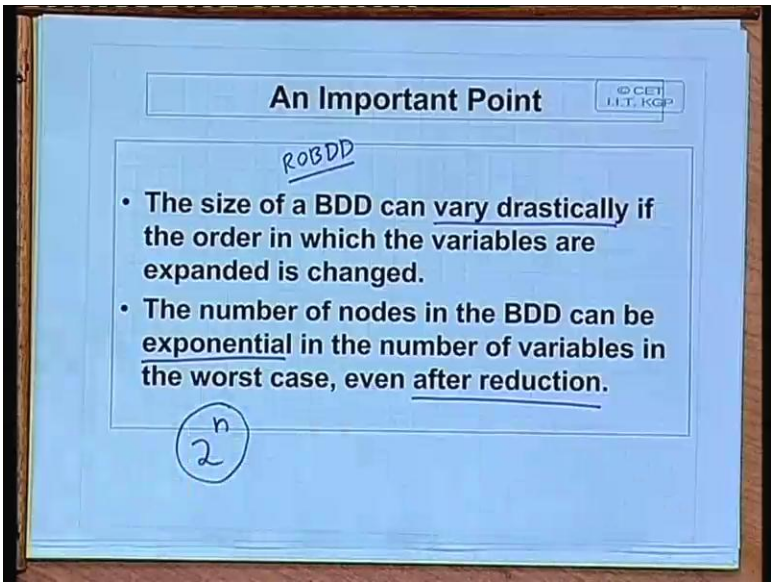


Now some advantages of BDD. First thing is that, well this. Yes. (( )) (33:11) While reducing a BDD we are converting a tree to a graph. Yes. (( )) (33:19) Yes very much true. I understand your question. You are saying that a tree was the data structure which we had initially for the OBDD. But after transforming that into OBDD we have destroyed the tree property. Now it is a graph and algorithms which apply to graph are more complex true. But you understand invariable function the size of the OBDD is of the order 2 to the power n. So for a big circuit n is large the amount of memory you will be requiring to store in the OBDD will be huge. But after reduction memory can be manageable.

That is why we go for ROBDD nothing else that is a compact way of representing the function. That is all. (( )) (34:11) See hardware it depends if you are see whether you are using ROBDD or OBDD, it does not really matter the way you process it. I will give you an example shortly that the ultimately you will be landing up in the same hardware. Because in ROBDD also we are capturing the same information well. So where ever there is similar things we are trying to combine them together that is all okay. Fine. Now some advantages or benefits of ROBDD.

Let us see first thing is that checking for tautology if the function is a true function f equal to 1 then the BDD will be a constant 1 no a c just a single node 1 its trivial. Obviously the simultaneous complementation, suppose I have a function f, I have a BDD for it I want to construct the BDD for the function f prime. So what I do? I simply exchange the 2 terminal nodes nothing else everything else is same. I make this 0 as 1 1 as 0. That is all and equivalence check is also easy provided you have the same variable ordering. Because as I told you both OBDD and ROBDD are unique under the same ordering, so if you can enforce that you are using the same variable ordering, then checking whether 2 functions are equivalent or not that also becomes easy fine.

(Refer Slide Time: 35:50)

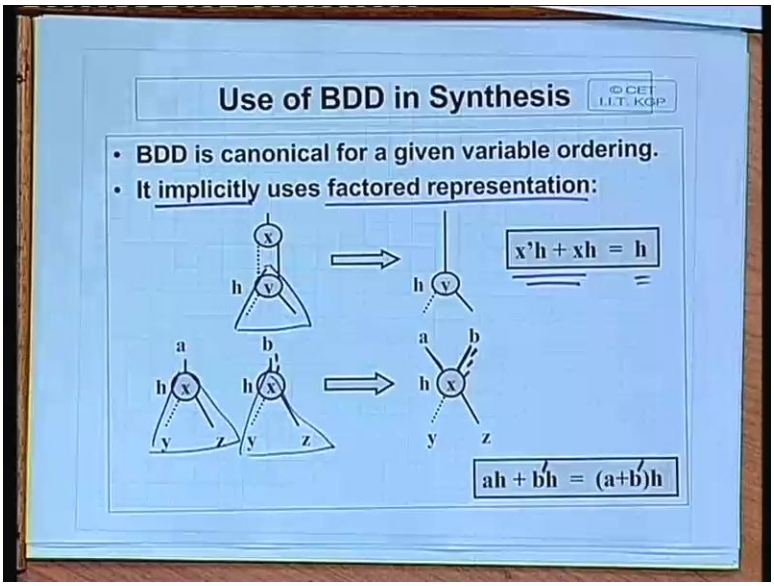


And point to note is that see from BDD we are drawing OBDD. But we really do not know which ordering of the variable would be better for us the size of the BDD after reduction. That means we are actually talking of ROBDD. The size of the final ROBDD can vary drastically depending on the order of the variables. So there is no deterministic algorithm which will tell you that, well this variable ordering is the best. But of course there are some heuristics which people use like you try to use a variable for breaking up or splitting up a function which approximately divides the number of true min terms into half half false and half true.

So you try to use a variable like that, well you can count the number of literals in the 2 sub functions and find out whether they are approximately equal. So if you can split it up into half at each stage that will be most probably the most efficient way of splitting those are heuristics. But it is not necessarily true that if you do that every time you will be getting the best. But for most of the time you will be getting good results and the worst case as I told you that the number of nodes can be exponential. And even in extreme cases after reduction also you cannot reduce much.

It will still remain exponential so the size complexity of a v d d in the worst case is 2 to the power n. But for most functions after reduction we get a drastic reduction. But you can always have a function like the exclusive OR function again that extreme example I am taking that BDD where it is difficult to reduce you will not be able to reduce. Fine. So this concept of reduced binary decision diagram this has been used by many researchers primarily for the representation of Boolean functions, for simulation, for verification, even for synthesis. Now in this context we will show you 1 method which has used where BDD was applied for synthesis.

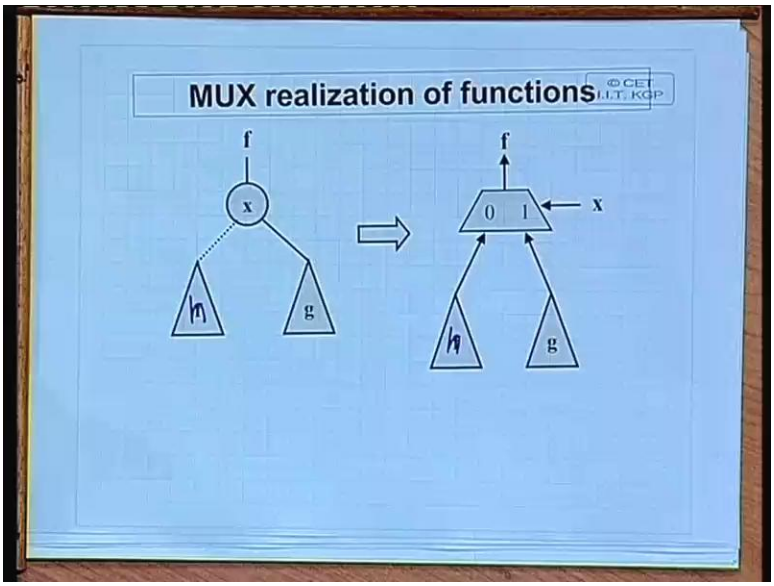
(Refer Slide Time: 38:15)



But first you observe well BDD is canonical for a given variable ordering I have already mentioned, but 1 thing you will just observe BDD it somehow implicitly uses factor representation. The transformations that we are carrying out that is implicitly doing that like when ever we are eliminating an node like x when it was leading to the same node at the next level. So in terms of the Boolean expression the left side which was indicating that if x is false come here if x is true come here. Suppose the sub function that is under this y lets call it h so this function is x prime h or x h this one. Now after reduction we are calling it simply h. So it is just a factor you just take h common or x prime is 1 you minimize it.

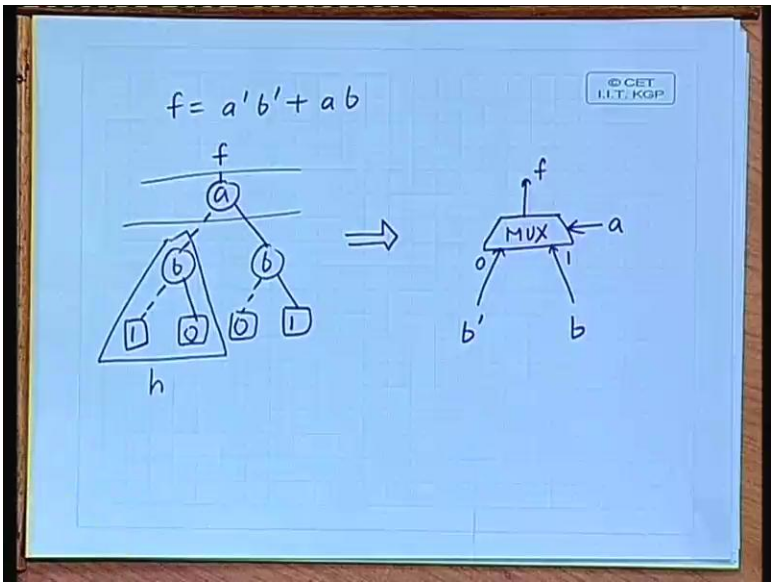
Similarly when you have something like this both a and b are coming here you put them in the same place. Now again this sub function is h this sub function is also h so the left hand side it was a h or b h when you do this as if we are combining them together a OR b h. So this reduction rules that we are carrying out while constructing ROBDD this implicitly captures the factor information and implicitly does some minimization for us. This is the idea. Yeah. (( )) (40:05) If b had a dotted line (( )) (40:12) it would have been different. Yeah, yeah. You could have could have. If this was a dotted line, then this was a dotted line, then this should be h OR b prime h. Yes you could have yes true fine. So now let us look at how BDD can be used for synthesis.

(Refer Slide Time: 40:42)



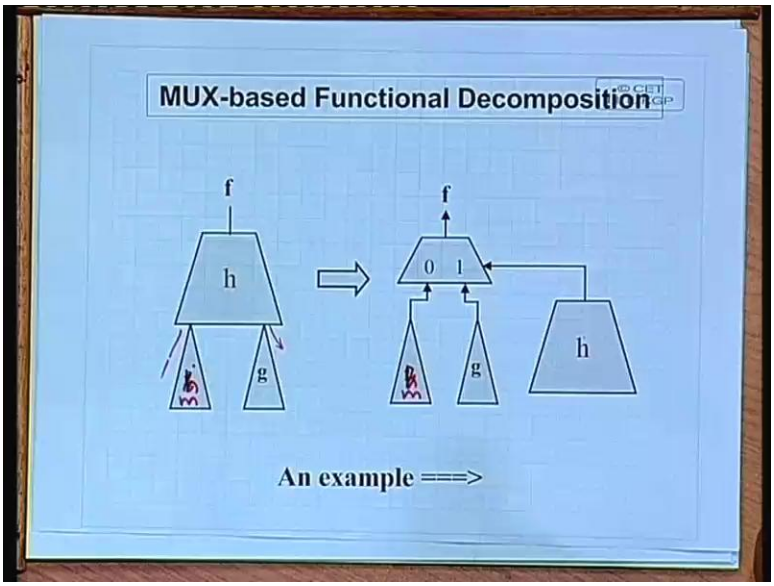
So I told you before that very naturally that why you are expanding the function while constructing BDD. There is some kind of multiplexer realization which immediately comes to your mind. Suppose you have a scenario like this a scenario where you have a node leveled x out here. If x is false you come to a sub tree out here, if is true you come to a sub tree out here and this sub tree realizes well this is f lets call it something else h realizes h this realizes g. Then this you can map into a partial hardware realization where this node x is modeled by a multiplexer this multiplexer is having a control line as x. The output of the multiplexer is the function and the inputs 1 of them is being fed by a sub set realizing h other by g. So as a very simple example.

(Refer Slide Time: 42:00)



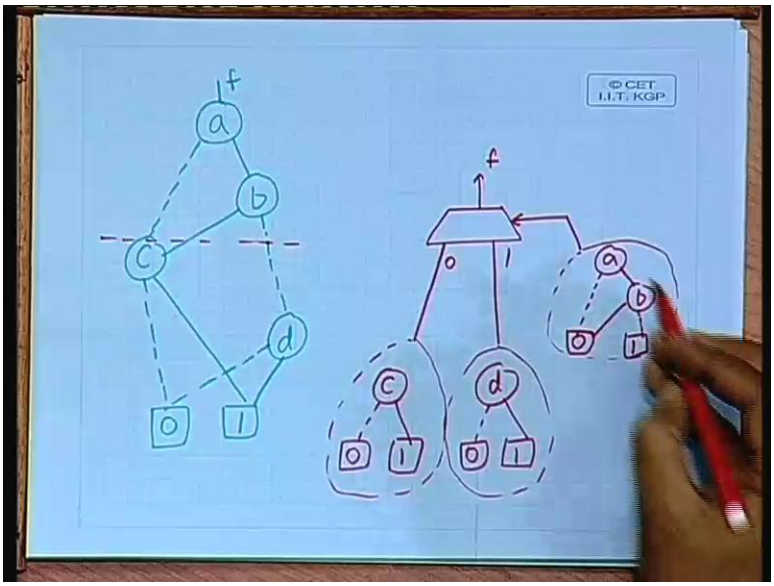
Let us take a very simple example say I have function a prime b prime or a b. So if I very quickly draw the BDD for this function, for the function f, we start with a 1 and 0 0 and 1. Unfortunately for this function you cannot do any further reduction this is already in the reduced form. Now this if you want to map into the hardware you will be having a multiplexer which control input is a I am talking about this level breaking up with respect to a the output is f the 2 inputs to the multiplexer. The left input and the right input. If it is false, I am coming to this. So I must feed this sub function which as calling h. Now well in this example this sub function is nothing but b prime if b is true, 1 if b is false, if b is false, 1 if b is true 0 b prime. So here you will have to feed b prime and here you will have to feed b. This is the idea basic concept.

(Refer Slide Time: 43:35)



And in general if you have a scenario like this, where you have a general BDD I am showing it in abstract form where you have a portion of the BDD this you are calling as h. This h is not a single node rather. This h is a portion of BDD and from h you have a point which is feeding to a sub function f. Well, let us call it h again that this, I used the same notation h. This h is already there. Let us call it say m. So then you can have a realization where there is a multiplexer with m and g below it. But the control input is fed not by a single variable but by another circuit which is realizing h. If h is true, I will follow the right path. If h is false, I will follow the left path. See a sub function with 2 sub function under this means that, this is the false path and this is the true path. So let us take an example to illustrate this. Suppose I have a function like this. I am just working this out. I show this diagram.

(Refer Slide Time: 44:58)

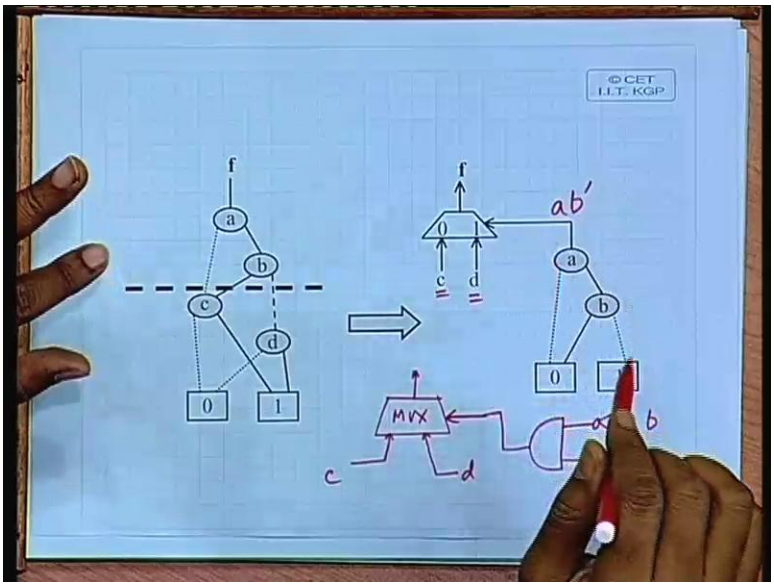


Suppose I have a function. Well I am drawing the BDD straight away suppose i have a reduced order BDD which looks like this. It is a 4 variable function this suppose i have BDD for this which realizes the function f. Now somehow I will have to cut this graph, so that the false path will point us to the left and the true path would point us to the right. See there are different ways of doing it. What you can do is that, we can cut this say at this level so after a and b we can cut this at this level and what you can have you can have a realization like this. This is a multiplexer whose output is f and the control input which is feeding I am showing it in abstract term. This is a circuit which implements the 2 inputs. Out here is the 0 input and 1 input this implements another circuit corresponding to the BDD.

And this implements another 1 which corresponds to say it is just a matter of convention. The left 1 I am treating as 0 the right 1 I am treating as one. See with respect to this the only thing that I want is that under some condition I will go left under some condition I will go right. Well here you really cannot tell that this is false. This is true but you can just tell this is left and this is right. So just based on that convention I am calling the left 1 as 0 right 1 as one. And if it is left then you have to come here c 0 1 and if it is right you have to come here d 0 1 right. So this is

your requirement of function and this 1 you see that this is simply c. this is simply d and this is well i am showing this diagram now.

(Refer Slide Time: 48:20)



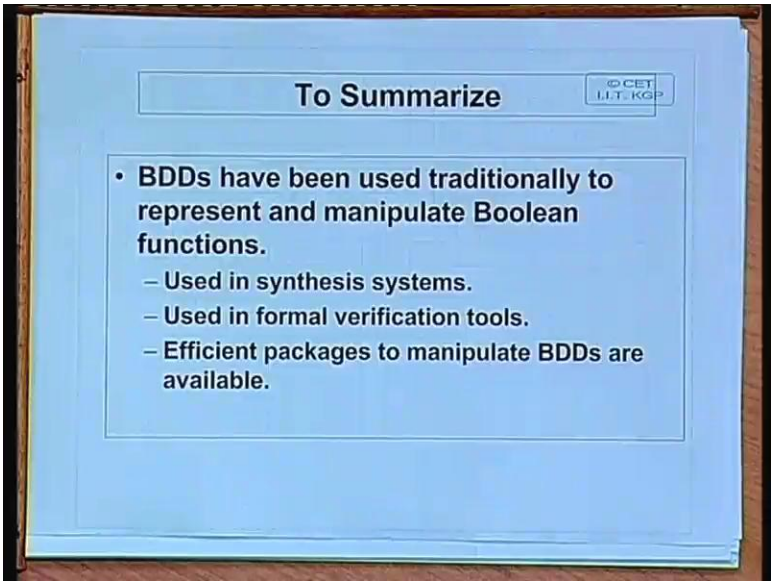
This is drawn in that compact form the left input is simply c right. Input is simply d but what is this function you can again follow that same rule. You trace all the paths through min term a b bar. So this actually is the function a b bar. So you see that this way of synthesizing a function is in a sense hybrid that means you are not getting a purely gate level realization. There are some multiplexers there are some small functions. You get which you will have to realize using gates like here your final implementation will be, you will be getting the multiplexer 1 multiplexer is needed and here you need some function for a b bar. Well if you are using NOT and AND gates. So you will have to generate b bar use an AND gate and feed it.

Here you get c here you get d so there are many synthesis tools which actually use a process similar to this. So it does not always synthesize the communication logic using gates only you will find that there are multiplexers in other block logic multiplexer decoders are also sometimes used for synthesizing a function. Now the advantage of using multiplexer is that if your way of representation or the way you are representing is true ROBDD then this multiplexer based

realization comes very naturally right. (( )) (50:07) So yeah you can cut this at any level you can cut this at any level provided, you can identify which is left which is right.

Say you can cut at the level of a also then you know that this part is left this part is your left and this whole part is your right. So you can cut at any level but you will have to remember this mapping. (( ))(50:46) Depend on this, so just you will have to do some kind of balancing again the best point to cut will be place where there is minimum overlap between the two. If you cut at a, you will see that the c node is common to both. So you will have to cut in 1 place where there is a minimum overlap between the two. And approximately they are of equal sizes. There are some heuristics which are used to that.

(Refer Slide Time: 51:20)



So to summarize, finally this binary decision diagrams, this was traditionally used to represent functions and to manipulate them. But today they also use this synthesis and also for formal verification. There are software packages which are already available in the public domain using which you can do or carry out number of different kinds of manipulations reductions on BDD. So any BDD based implementation whatever you are implementing you can use those packages to start with those are already there.

So you can simply use them and you can build up your system. So in our next class we would be starting to talk about synthesizing starting from a slightly higher level. We will be talking about high level synthesis say starting from a behavioral description a language like Verilog. So how do we go about doing it? What are the different steps of transformations? So how do you ultimately land up into a hardware implementation typically at the arc level register transfer level?