

Design Verification and Test of Digital VLSI Designs
Dr. Santosh Biswas
Dr. Jatindra Kumar Deka
Indian Institute of Technology, Guwahati

Module - 3
Logic Optimization and Synthesis
Lecture - 1
Two level Boolean Logic Synthesis – 1

So, good morning and welcome to module 3 of the course of the design part of the course. So, as you know that in the design part of the course, we have seen that we start in case of VLSI digital design, we start of design specification, and finally we and finally we have to reach for the stage where we can go and layout the chip. So, in module 1 of this design part of the course, we have seen that that the basic design, and then we have seen that we start with high level design synthesis that is we take a specification and then we go for a RTL level design.

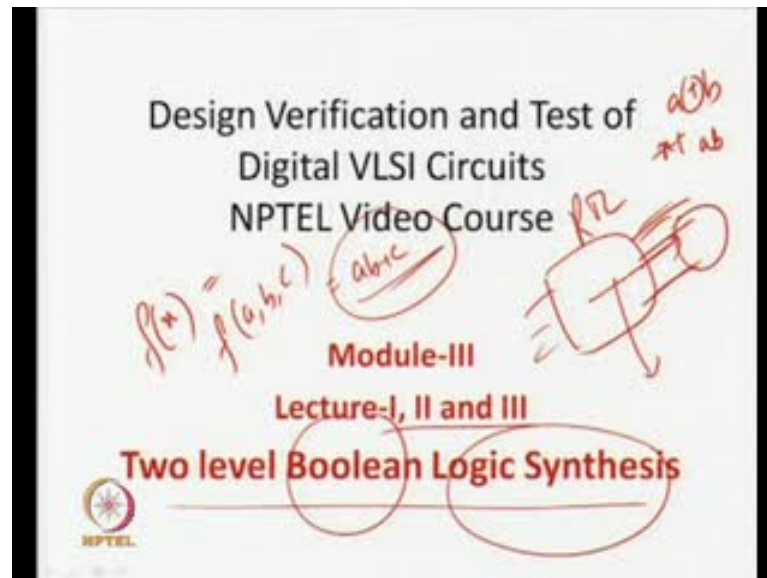
In the second module, we have seen that what are the different type automatic algorithms, which can solve the different parts of high level synthesis. That is we can go for we know that high level synthesis comprises scheduling allocation and binding. So, we have seen algorithms, which can solve the scheduling problem, which can solve the allocation and which can solve the binding problems.

Then, after the end of high level synthesis or after the high level synthesis procedural completes, then what we get? We get a RTL level design, that is a register level transfer design or you can say a basic architectural level design. Now, the next step of the design flow as discussed in module 1 or the introductory lecture of the course. We know we have seen that after the high level design is obtained or the RTL design is obtained that is some kind of black box design is obtained, then our idea is to go for a gate level design.

So, once the gate level design is obtained, then only we can place it, route it and go for the further down these steps. So, high level from the high level design, we need to have a largely gate level design because everything ultimately has to be implemented using in terms of gates and flops. Then, finally it can be I mean and actually in the hardware library, we have nothing but the gates and they actually comprise the transistors.

So, gates are a unit block, you can say the building block for digital designs and then there will be actually placed routers and will be going for the backend design. So, very important is that we have to convert the high level designs to a gate level design.

(Refer Slide Time: 02:03)



So, now in this case, what is what we are going to see, we are now in the second module, second module number 3, we in this lectures 3 lectures series, what you will see? We will see two level Boolean synthesis logic synthesis. So, what you mean? We will see what you mean by two level Boolean logic synthesis. So, first what you are going to see is what do you mean by Boolean logic synthesis.

So, actually the idea is that whatever I mean RTL design you have like we can say RTL design or black box design. So, they can all, they are all nothing but they represent some Boolean functions. So, how they represent some digital functions? So, you can easily represent them as Boolean logic or Boolean all binary functions sorry Boolean functions, which can have some binary values or in other words, every even RTL design will have some inputs and have some outputs. So, all the outputs actually are digital outputs and the inputs are obviously digital inputs because we are talking about digital design.

So, all the outputs can be represented in terms of the inputs and also in terms of input outputs also. I mean if you have what you call of sequential circuit, then the output will be depending on input and also on the state. If it is purely combinational circuit, then the

output will be depending on the input. So, in other words, all the outputs of the circuits can be represented in terms of some kind of Boolean functions.

Now, what happens? So, that is what all the functions all the outputs, if you can say can be represented in terms of some Boolean functions or they have some binary output values, then what is the idea? Then, we can convert each of these functions to logic gates that is say for example, we have a function say $f(x)$ is an output for one of the output of the RTL design. Then we know that if you say it will be a function of say a, b, c where a and b and c are the inputs.

So, $f(x)$ is equal to some function of actually a, b, c . So, that is the output, x will be depend on some input values of a, b, c . So, it can be like $a + b + c$ or something like that you can have some any Boolean, you can have any Boolean values or any binary, you will have x will have a binary value and you can have any Boolean, it can be represent any Boolean function.

Now, so that is the idea like for example, if it is an adder circuit, so you will have this adder is equal to $a + b$ or $a \oplus b$. Similarly, if it is a carried part of half adder, so the carried part will be $a \cdot b$. So, that means, output of all the all the outputs of the RTL design or the high level design can be represented by some Boolean functions.

So, that now these Boolean functions have to be converted into logic gates. So, Boolean as if they are Boolean functions will take they will have a binary values. So, we can easily represent them represent them in terms of by using some binary gates sorry in terms of some logic gates. So, that procedural is actually called logic synthesis.

So, in this 3 module, I mean module 3 in the 3 lecture series, we will see about two level Boolean synthesis or we will be trying to implement our that is Boolean functions by what do you say two level by using of Boolean by using logic gates, which should be having two levels. So, now we will see in details what you mean by two levels, what you mean by multiple levels, so forth.

(Refer Slide Time: 04:55)

Introduction

- In the last two modules, we discussed that in case of digital VLSI design we start with high-level system specifications, which are transformed into optimal Register Transfer Level (RTL) circuits using High Level Synthesis (HLS) algorithms.
- Once the RTL circuit is available, we need to transform it to gate level design, which can then be processed by backend algorithm; this process is called Logic Synthesis. Formally speaking, Boolean logic synthesis is a process by which an abstract form of desired circuit behavior, typically RTL, is transformed into a design implementation in terms of logic gates and flip-flops. This module is dedicated to logic synthesis of combinational and sequential circuits.
- It may be noted that two levels of logic are minimum required to implement an arbitrary Boolean function. Generally, we assume that the primitives are AND and OR gates and inverters. AND gates are used at the first level and OR gates are used in the second level. Inverters may be present at some inputs of the gates of the first level, but it is not considered as an additional level.

So, this is the introduction. What I have said that we have seen that we get the RTL design from high level synthesis. After the high level synthesis, we get the RTL designs. Then, now the RTL circuit is available, you have to transfer into gate level design. Now, that is actually called logic synthesis. So, if you are having the RTL design, so formally speaking, so if the RTL design you want to represent them using logic gates, so this process is called logic synthesis because all the outputs that as I mentioned, all the outputs of this RTL circuits will be represented by some Boolean functions.

Anything that is represented by some Boolean function can be always represented by some logic gates or some gates because logic gates are also nothing but they also represent some Boolean functions like for AND gate, it is a dot b having two inputs AND gate. So, the output, so the function of and two input AND gate is you can say that a dot b.

Similarly, you can read every function also, they will all be represented using Boolean I mean sorry logic gates. So, that is why it is called logic synthesis. So, now we are going to see this module, I mean this module is entirely ridiculous logic synthesis. So, module 3 and in the first three lectures, we will see how we can about two level logic synthesis. Now, we will slowly see what you mean by two level, what you mean by multiple level, so we will be coming into picture.

(Refer Slide Time: 07:17)

Introduction

- In the last two modules, we discussed that in case of digital VLSI design we start with high-level system specifications, which are transformed into optimal Register Transfer Level (RTL) circuits using High Level Synthesis (HLS) algorithms.
- Once the RTL circuit is available, we need to transform it to gate level design, which can then be processed by backend algorithm; this process is called Logic Synthesis. Formally speaking, Boolean logic synthesis is a process by which an abstract form of desired circuit behavior, typically RTL, is transformed into a design implementation in terms of logic gates and flip-flops. This module is dedicated to logic synthesis of combinational and sequential circuits.
- It may be noted that two levels of logic are minimum required to implement an arbitrary Boolean function. Generally, we assume that the primitives are AND and OR gates and inverters. AND gates are used at the first level and OR gates are used in the second level. Inverters may be present at some inputs of the gates of the first level, but it is not considered as an additional level.

NPTEL

Handwritten notes: a, b, c, d; AND; OR; SOP = (a.b) + (c.d)

So, whenever you say logic synthesis, so we are actually using logic gates if it is a combinational circuit and both logic gates and flip flop is a sequential circuit. So, now coming to the question of what you mean by sorry two level logic. So, what is a two level logic? In two level logic, we are using only AND gates, OR gates and inverters. So, inverters means a prime is if we inputs are not inverted, so if you see assume that inputs are a, b, c, d, so if you want to get a prime, b prime or c prime that is the inversion.

So, you have to use i inverter. So, use AND gates, OR gates and inverters. So, in two levels, AND gates are used at the first level and OR gates are used as the second level. We will see what do that mean. So, that means, I think in your digital design undergraduate course, we have seen that some function like f of x, which I have told can be represented as a b plus c. So, what does that mean? It means, you have an AND gate for a b, this is a b and then there will be a OR gate at the second level. So, it will be nothing but c. So, it is c a dot b plus c. So, that means, what in the first level, you have put an AND gate and the second level, you have put an OR gate. So, we will see with the bigger examples. So, that is the basic idea.

So, in sum of product form, what we know that in our under graduate design lectures, so in sum of product means a b plus c d plus e f that is you are summing all the products. So, that is that is all the product terms is they are actually represented by AND gates and we are actually of summing them the sum of products. So, your products, all the

but if I write x plus x prime; so this is nothing but a 1. So, this one can be represented this way, x can be 0 can be represented in this way. In a very similar way, I can write xyz plus x . So, this is nothing but this is nothing but actually only.

(Refer Slide Time: 09:21)

Introduction

- Many other choices are also possible namely, using OR gates at first level and AND gates in the second, using NOR and NAND gates etc.
- It is also possible to implement a circuit in more than two levels, however, it is more complex procedure.
- In this triple lecture, we will first discuss two level logic synthesis procedures. Latter in this module, we will discuss multilevel synthesis.
- There are two main reasons why we may want to implement a circuit in two levels, rather than multiple levels namely, speed of operation and simplicity of the algorithms. However, in practical cases two level implementation may not be possible. Reducing the number of levels increase the fanin and fanout counts of gates. Gates having high fanins and fanouts are slow. Therefore, design libraries do not generally have gates with more than four fanins; this requires multiple level synthesis.

NPTEL


$xyz + x = x$

You can say that this is nothing but x into xy plus 1, so this is nothing but x . So, this whole thing can be represented by an x . So, that means, what all the functions whatever the functions can be written, so same function like 1 0 or whatever you given like example like xyz plus x is nothing but equivalent to x . So, the idea is here you will be requiring a 3 input AND gate and a OR gate kind of a thing but it is equivalent to x will not require a gate.

(Refer Slide Time: 09:52)

Introduction

- Many other choices are also possible namely, using OR gates at first level and AND gates in the second, using NOR and NAND gates etc.
- It is also possible to implement a circuit in more than two levels, however, it is more complex procedure.
- In this triple lecture, we will first discuss two level logic synthesis procedures. Latter in this module, we will discuss multilevel synthesis.
- There are two main reasons why we may want to implement a circuit in two levels, rather than multiple levels namely, speed of operation and simplicity of the algorithms. However, in practical cases two level implementation may not be possible. Reducing the number of levels increase the fanin and fanout counts of gates. Gates having high fanins and fanouts are slow. Therefore, design libraries do not generally have gates with more than four fanins; this requires multiple level synthesis.

 $x^2 + x = x$

So, idea is that so any of the function which will be represented any of what you call these Boolean functions, which will be using to represent the outputs of your RTL circuits, they can be represented in many different ways. The same function can be represented in many different ways like just like $xyz + x$ is equivalent to x . So, the idea in other words, same function can be represented in different ways, but the number of gates required or number of hardware required will be different. So, our idea here will be you have given a function, so you have to represent it in such a way so that it is, it can be represented by the minimum number of gates.

Again, we will see that this problem is $n p$ hard and $n p$ comp, then it will be difficult to solve in a way to get the most minimum value is very difficult to do it in a polynomial time or a less amount of transfusion times. So, again we will find out heuristics to do that. So, this is the basic agenda.

So, as I told you in the beginning of the course or in many lectures of the course that most of the problems in VLSI design are all $n p$ hard and $n p$ comp that is you do not have a very simple or a computationally lower low complex algorithm to solve the problems. That means, you do not have a polynomial time algorithm to solve the problem of this Boolean function minimization.

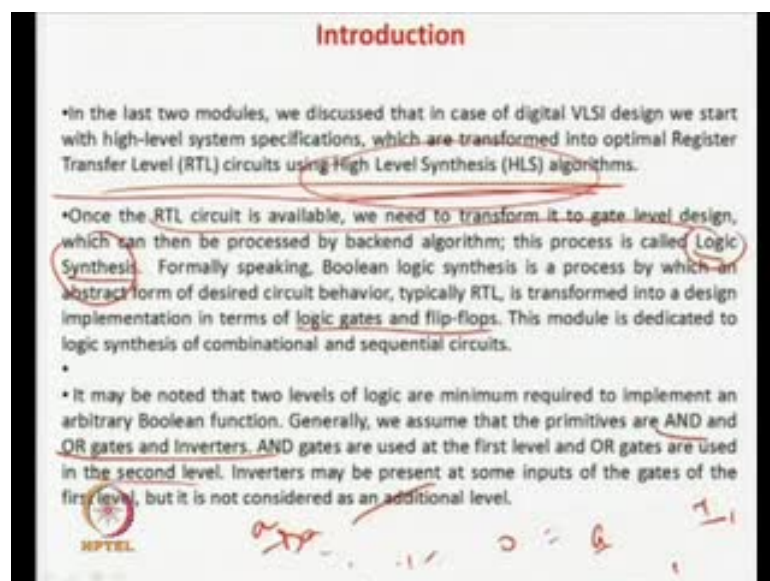
Now, why you cannot represent it? This is because you will find out that if the number of variables is x , the amount of time required to spend is about the order of 2 to the power n

plus. So, therefore, I mean if you need to have a 100 function 100 variable function, so to find out the minimum number of gate representation for that, it will be order of 2 to the power of 100, which is feasible in terms of complexity time, so time complexity.

So, what you have to do? You have to take some heuristics to find out some representation, so which is very near the minimal as well as the execution time will be lower. So, this is actually true for all the things like for different high level synthesis algorithms you have seen like your zero one ILP for high level scheduling.

Then, also we have seen that if you are going for quick partitioning based finding and so forth and they are all difficult problems, then if you have found out heuristics, which are required to solve the problem for same thing, you will also doing for the Boolean two level Boolean synthesis. So, we will see that multi level is much more complex problem than the two level problem.

(Refer Slide Time: 11:55)



So, we will all slowly come into that, but now for the time being, you just assume that from the discussion, just you can understand that the problem like also this two level Boolean synthesis, two level binary, so two level logic synthesis is also a very difficult problem.

(Refer Slide Time: 12:17)

Introduction

- Many other choices are also possible namely, using OR gates at first level and AND gates in the second, using NOR and NAND gates etc.
- It is also possible to implement a circuit in more than two levels, however, it is more complex procedure.
- In this triple lecture, we will first discuss two level logic synthesis procedures. Latter in this module, we will discuss multilevel synthesis.
- There are two main reasons why we may want to implement a circuit in two levels, rather than multiple levels namely, speed of operation and simplicity of the algorithms. However, in practical cases two level implementation may not be possible. Reducing the number of levels increase the fanin and fanout counts of gates. Gates having high fanins and fanouts are slow. Therefore, design libraries do not generally have gates with more than four fanins; this requires multiple level synthesis.

NPTEL

$x y z = x$

n

What is the problem? The problem given a function here, we represent it with a minimum number of gates.

(Refer Slide Time: 12:47)

Introduction

- Many other choices are also possible namely, using OR gates at first level and AND gates in the second, using NOR and NAND gates etc.
- It is also possible to implement a circuit in more than two levels, however, it is more complex procedure.
- In this triple lecture, we will first discuss two level logic synthesis procedures. Latter in this module, we will discuss multilevel synthesis.
- There are two main reasons why we may want to implement a circuit in two levels, rather than multiple levels namely, speed of operation and simplicity of the algorithms. However, in practical cases two level implementation may not be possible. Reducing the number of levels increase the fanin and fanout counts of gates. Gates having high fanins and fanouts are slow. Therefore, design libraries do not generally have gates with more than four fanins; this requires multiple level synthesis.

NPTEL

$A \rightarrow B \rightarrow C$

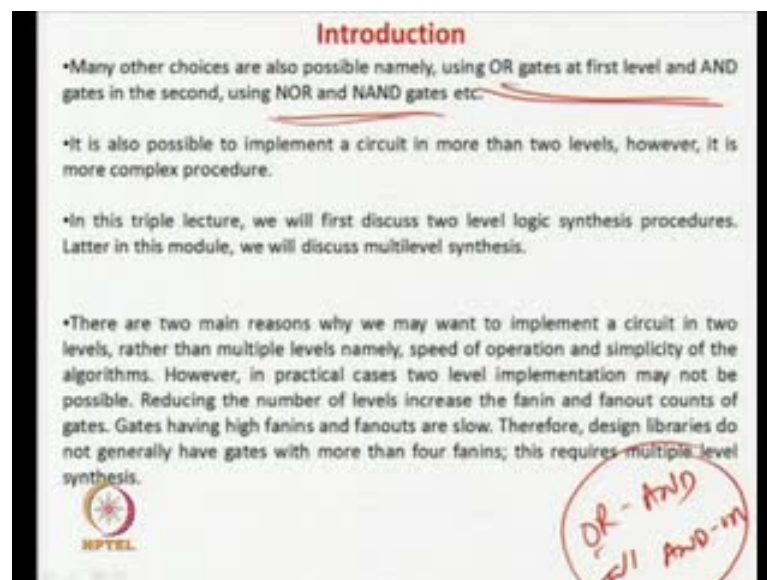
$(POS) (A+B)(C)$

So, that is actually the definition of the register transfer level to high level synthesis, which is given, the RTL designs using which are obtained by high level synthesis. So, you have to represent them using gates, so which is called logic synthesis and that number of gates should be minimum, otherwise I mean as I gave an example similar function, which is actually a similar function, same function here we represent it in very

different ways. They require a different number of gates to solve the problem like xyz plus x here you require 3 input AND gate and a OR gate, but for x same function equivalent function x this is actually equivalent to this.

Here, you do not require any gates. So, therefore, not only that you represent the Boolean functions by some logic gates or also that you have to do it in the minimum number of gates like so that is two input logic synthesis I mean two level logic synthesis like AND OR and AND OR. In this case, there are other choices also like you can use OR gates at the first level, AND gate in the second level, so what is that? It is actually called product of sums like A plus B dot C. So, what is this?

(Refer Slide Time: 13:37)



It is called the POS form that is you are producing the sums, so it will be what? It will A or B or B.

(Refer Slide Time: 13:58)

Introduction

- Many other choices are also possible namely, using OR gates at first level and AND gates in the second, using NOR and NAND gates etc.
- It is also possible to implement a circuit in more than two levels, however, it is more complex procedure:
- In this triple lecture, we will first discuss two level logic synthesis procedures. Latter in this module, we will discuss multilevel synthesis.
- There are two main reasons why we may want to implement a circuit in two levels, rather than multiple levels namely, speed of operation and simplicity of the algorithms. However, in practical cases two level implementation may not be possible. Reducing the number of levels increase the fanin and fanout counts of gates. Gates having high fanins and fanouts are slow. Therefore, design libraries do not generally have gates with more than four fanins; this requires multiple level synthesis.

NPTEL

(Handwritten diagram showing a sequence of gates: an OR gate followed by two AND gates connected in series.)

Then, finally, you have to put an AND gate and make it, this is will be C. So, this is POS form. So, you already know from digital design undergraduate course that two times we represent thermal product and product of sum. So, it is a product of sum. So, if you are using a product of sum, then the first level will be your OR gate and the second level will be your AND gates. So, you can also make it using NAND and NOR gates.


So, I will think it is possible right now, also inverters also will be equal if you are using. If you are not using inverting logic like if you are using NOR, OR, AND gates, so their inverters are already coming by I mean actually because you know AND and the OR gates. But, if you are using AND and the OR gates, so OR and AND gates like OR gates and AND gates are two level sorry AND level and OR level.

So, if you are using this, you require inverters, because you are not using any of the inverters in the OR AND or and gates. But if you are using NAND and NOR logic gates to represent, result is automatically, the inverters are taken care of this one as we will see that it is also possible to represent the circuit in multiple number of levels. So, here these two levels, so will see why it is very much required to represent in multiple levels because two level synthesis is a very hypothetical situation.

(Refer Slide Time: 14:43)

Introduction

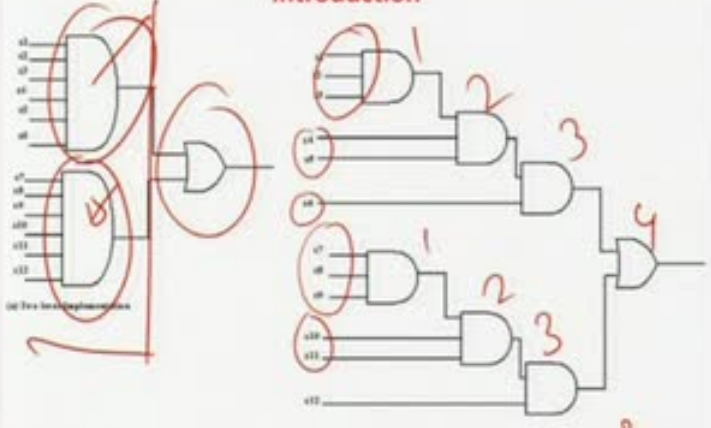
- Many other choices are also possible namely, using OR gates at first level and AND gates in the second, using NOR and NAND gates etc.
- It is also possible to implement a circuit in more than two levels, however, it is more complex procedure.
- In this triple lecture, we will first discuss two level logic synthesis procedures. Latter in this module, we will discuss multilevel synthesis.
- There are two main reasons why we may want to implement a circuit in two levels, rather than multiple levels namely, speed of operation and simplicity of the algorithms. However, in practical cases two level implementation may not be possible. Reducing the number of levels increase the fanin and fanout counts of gates. Gates having high fanins and fanouts are slow. Therefore, design libraries do not generally have gates with more than four fanins; this requires multiple level synthesis.




We slowly see, but from the representation point of view or circuit synthesis point of view, you cannot go on using two level synthesis. You require multilevel implementation.

(Refer Slide Time: 14:52)

Introduction



we have a Boolean function as $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12})$



Multilevel means it should be something like this. More than two levels will be there and in the end, you can have an OR gate. So, it will be 1, 2, 3, 4, four levels. So, you can slowly see that or can also have some, I mean you can have intuition and you can find out this much more difficult problem to solve as we will see it later. So, in the last lecture

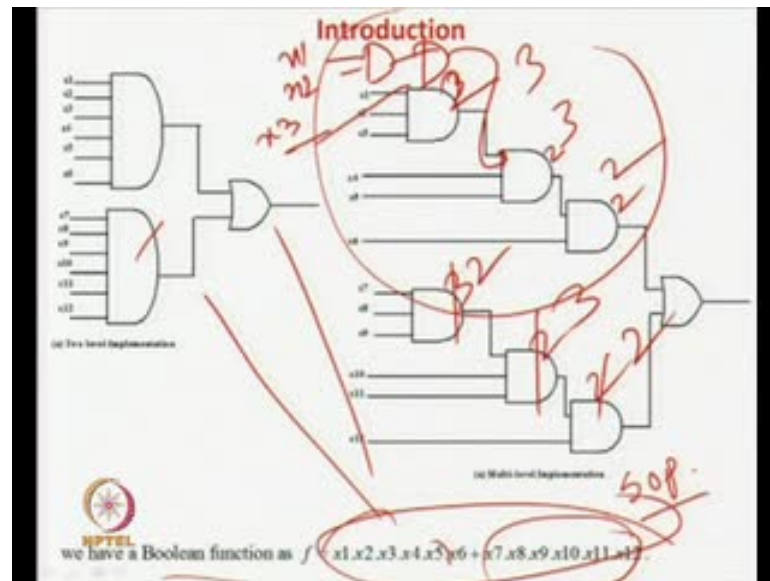
of this module, we will see how to go for multilevel synthesis. Then, it will be very clear to you in formal terms that why multilevel synthesis is such a difficult problem. Now, we will see that why we require a circuit in more than two levels. So, slowly we will come to that.

So, first of all let me take an example, so then we will again come back to this point that why multilevel and why two level? Then, in this case, let us see the function like is a Boolean function f of $x_1 x_2 x_3 x_4 x_5 x_6$ plus $x_7 x_8 x_9$ and this one. So, this is a sum of product form, this is one product term, this is one product term and making a sum two level implementation is what it is? x_1 dot x_2 dot up to x_6 , so you have AND gate and x_7 to x_{12} is that again as a dot product, and then again finally OR gate.

Now, these are two levels implementation because the first level is AND, the second level is OR and these are sum of product form. So, these are two level implementation. Like if we want to go for a multilevel implementation, so what is a multiple level implementation? They will have more than one level. So, we are actually splitting up this AND gates.

So, you see first that $x_1 x_2 x_3$ in one, x_4 and x_5 in second level and this IS the case and x_6 has taken a third level. So, there is level one level, two level, three again in this case also. It has taken $x_7 x_8 x_9$ in one gate AND gate x_{10} and x_{11} in the other and x_{12} in another. So it is like again 1, 2, 3 and finally, it is a four level implementation. Now, if you just look at this, so what is the difference.

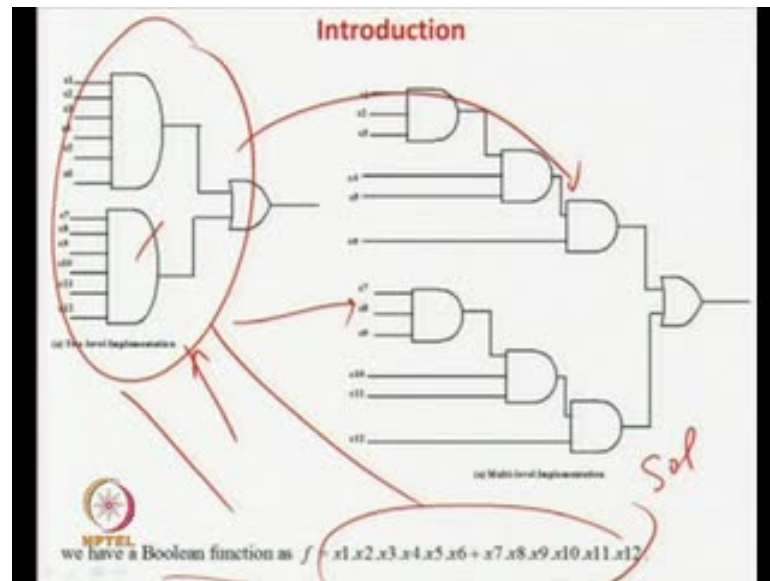
(Refer Slide Time: 16:03)



So, you can understand that in this case, it is very simple actually. Thus, whatever, function so SOP you have, you can directly map to this one. The sum product terms is there and sum terms is there and here you have to do lot of things like here you may get three input AND gate, three input AND gate, again a two input AND gate, you can do with another. This can be also implemented by other two inputs AND gate like you can have $x_1 x_2$ here and then again you can have another gate called x_3 . Here, make another AND gate over here and then you can connect it over here.

So, it will be two input, two input, two input AND gate. Similarly, you can three input, three input and you can have together three input AND gate. So, a lot of options will be available to do it. So, more the number of options, more difficult is a problem to solve. Here, there is one option to do it. So, the algorithm what the automatic techniques to convert the SOP tool to this equivalent implement, which will be very simple, but here there are lot of options available like. These are three input, two input, two input say again you have put making a two input and AND gate 2, 3 input AND gate and this again made a two input AND gate. So, lots of options are available and you have to take the best option.

(Refer Slide Time: 17:18)

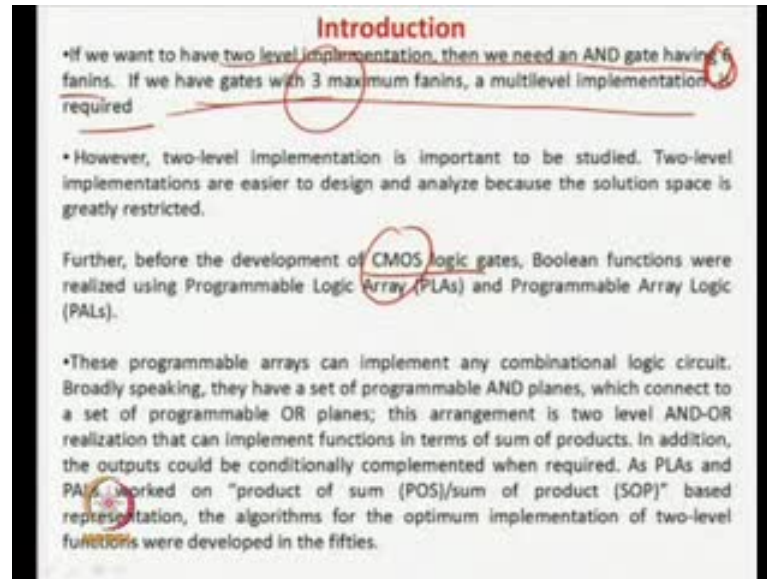


So, if there are lot of options available, so the algorithm will be very, very difficult because you have to find out the solution from among all the options, but still, why? As I told you that we first go on, we can say that this also you can think that this is a simple procedural that is you have to convert into two level implementation and multilevel implementation is having a lot of options.

So, the algorithm will be difficult, but still as I told you that I am in these two level implementation is not the very feasible or you can say that this is the circuit directly, go and fabricate it this is not a very feasible one. We have to convert this to multilevel implementation, and then only it can be fabricated. Now, we will see why it is so? But, then and there we will see, but still why you need to first go for two level implementation?

So, but I am saying is an actual that it is very given a SOP form from a product term very, very easily, you can go for this circuit implementation, two level circuit implementation, but two level circuit implementation cannot be directly fabricated. You have to convert it into a multilevel circuit implementation, and then only fabrication will possible. Now, we will see why and secondly why you do not directly go for multilevel circuit? We directly go for two level and from there, we actually convert. Now, we will get logic, we will try to break.

(Refer Slide Time: 18:02)



So, if you want to have a two level implementation, then we need an AND gate is having five fanins. We have a, if you have a gate with three fanins, then multilevel implementation is required. Now, you see what it is. If we have two level implementation, then we require an AND gate, which has five inputs 1, 2, 3, 4, 5, 6 sorry in this case it is 6 sorry 2 here and it is it is a six level fanins.

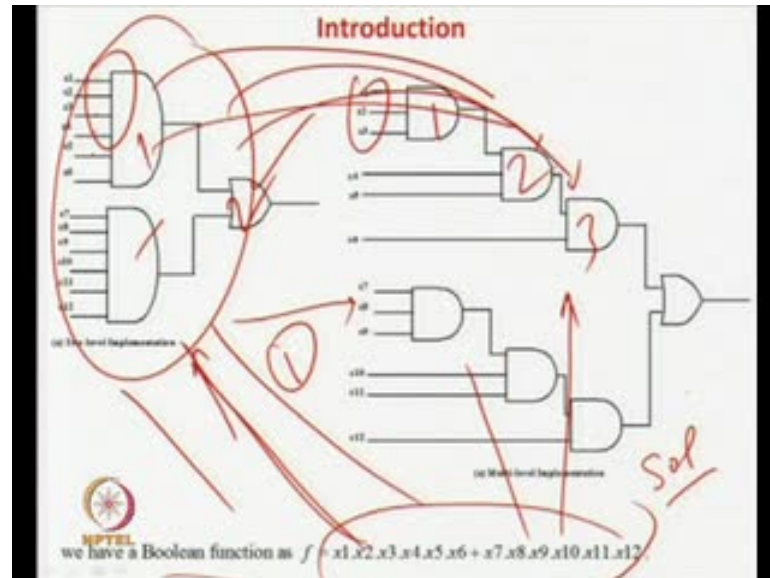
In this case, what is the maximum fanin? Fanin number of inputs for this gate for this implementation it is 3. So, for a multiple level implementation, you have three and for a two level two level. I mean implementation it is a 6. Now, you see why it is a problem? Just to show you can here come back. So, the idea here is that now a days, so that thing is making more clear slowly going down the lecture.

So, now days, actually you are using CMOS implementation. What is a CMOS implementation? So, in CMOS implementation in this diagram and why the easily you will get to you for the time being, you just take it from my side. And take it that granted for the time being that in a CMOS implementation, it is very difficult to fabricate a gate or even if you have a gate it is very expensive or it very slow.

If you have a gate, which is having more than four inputs, so if you have 5 inputs AND gate, 10 inputs AND gate, 100 inputs AND gate, theoretically it is possible, but if I want to do such a implementation in CMOS, now that is all the circuits are CMOS circuits. So,

if you have that gates will be extremely slow and the other or if you very expensive in terms of power and in the fabrication cost.

(Refer Slide Time: 18:14)



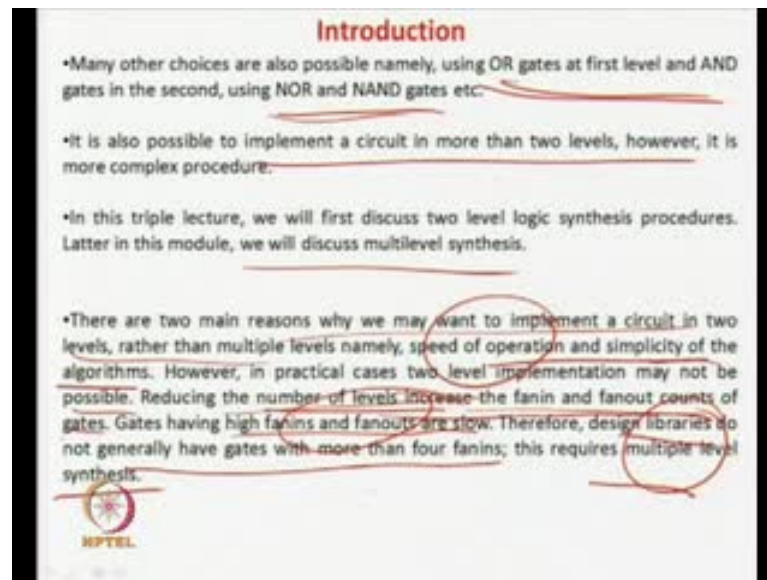
So, in case of CMOS, you can take thumb rule that most, almost all the gates are having four inputs or less. So, if that is the case, then we are in a big problem. Then, whenever we have a two level implementation, some of the gates have more than four inputs, then you have split it into multiple level inputs multiple level circuits. So, that is the idea, but directly if you want to convert SOP form to multiple level implementations to the algorithm will be horribly complex. So, that will be totally towards we are going to the end of the third module.

So, for time being, you take the two points from the as granted. The first point is into they are also into that if you directly go for this multiple level implicants as there are many available. So, this functions base is last. So, that is why the algorithm will be very, very complex, but here solution space is very limited because it is one way we can do it.

It is quite simple to go for a two level implantation from a SOP. So, you go that and then as in this thermo technology, it is very expensive or the gate should be very much having high deal play or low frequency or high voltage a lot of power. If you actually serving with more than four inputs, so that is why we have to convert this into this at the into, if we number of inputs in the multiple two level indentation is more than 4, then we have to convert it into break doubt, so that the inputs for each gate remains four or less.

That is why it is a two step process. First, we go for this two level implementation and then we convert it into a multiple level implementation, so satisfying the things to themselves simply as well as none of the gates will have be having more than four inputs. So, that is what is the idea.

(Refer Slide Time: 21:35)



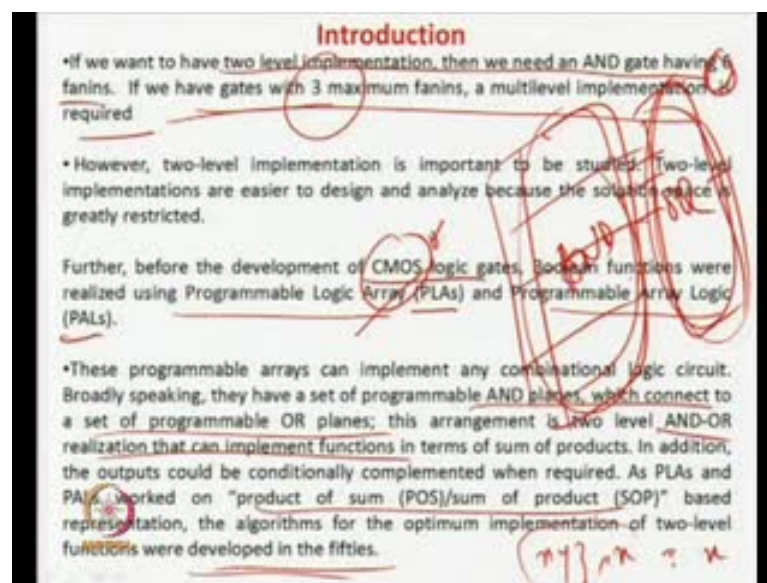
Now, so again, you can see that if somehow if somehow you could have a fabricated very good quality and it is so obviously this implementation would have been faster because for high pathetic calculation, it is no very easy to calculate 100 input AND gates or something like that having this possible, there is only two levels. So, circuit implementation would have been faster, but in this you see 1, 2, 3 and so far so many levels will be there, circuits will be slower. Therefore, we say that there are two main reasons. Why?

We may want to implement in two and then multiple levels only speed of implementation, operation and simplicity of the algorithms, speed of operation we get hypothetical ways because nobody can fabricate very fast. 100 input AND gates are taken, but simplicity of the algorithm is there, directly nothing you can find out. However, in practical cases, two level implementation may not be possible because reducing the number of levels including the fan outs and this one.

So, if you have the, so if you are having two level implementation, then what happens? If you have two level implementations there, if you see the number of fanins, the number of

inputs, so in the case will be higher. If you are reducing the number, if you reduce the number of fanins, so fan outs should also be higher. So, gates having high fanins and fanouts are slow that is what if you have gates with 10 inputs or 10 inputs are extremely slow. So, what we have to do? So, designs libraries do not have generally have gates with more than 4 fanins. This requires multiple level synthesis. If a design library is not supporting any gates these having more than 4 inputs kind of a thing, so you have, so what we have to do? We have to go for high; you have to go for multiple level synthesis so that you can handle this case.

(Refer Slide Time: 22:45)



So, I have already explained this one. Now, so in this case, if we in this example, so here the number of findings is six, this case will become three and the name problem is with CMOS. Now, actually that is what again if you see the in the if you see the early before the CMOS statement was there, these some of line operation which is a more of more of weakaton ethnology and implementation technology is not related to digital CAD. Before that, CMOS technology CMOS technology was not there.

So, you used to use I mean some kind of a other stuff like all use programmable logic arrays or PALs or PALs, this is another way of representing. Now, what is we represent CMOS gates representation or Boolean function, but sometime but we have also used to use programmable logic array PLA or programmable logic array.

So, they are nothing but AND or PLAs, but speaking this arrays implementation of combination of circuits, we can implement the combination of circuits. These arrays and they actually have a programmable arrays connected to set of OR planes, this is the AND or realization of the functions. So, we are not going into difficult of implementation of PLA and PLAs. So, what is that is not much that is not much used mainly. In today's science, we mainly use what do you call CMOS gates.

So, in case of PLA and PLAs understand that they are nothing but there will be a plane, there will be a lot of AND gates and lot of chain of blocks of AND gates so over here. After there is a block of OR gates, so this is a AND gates and block of OR gates.

Now, so there only kind of architectures, so you can use some of the AND gates as function. Finally, you can use some of the OR gates plane. Then, we then finally, we can implement your sum of product form, you can directly represent the sum of product and product of sum. If sum of product will be represented as by this way and if you use the product of some actually OR plane or AND planes, thus the positions will be and then we can represent what we can called this, then we can represent in the binary form sorry Boolean function.

So, the and obviously, there are two level implementation already because it is a one set of AND planes and there is one set of OR planes. There is no exactly gate level implementation because assume that there was a set of analogs and lot of AND gate array and there is a lot of OR gate arrays you can implement your sum of product and product of sum from directly.

So, I mean, so these things were actually, they used to be developed some CMOS technology or CMOS logic gates were not that popular. So, what people used to do? So, they used to minimize the representation in two level like for example, as I told you so like this one is the case like $x y z$ plus x .

So, if I want to represent it in a two level, then what is my idea? Since, represent it in such a way two level only that is sum of product and product of sum. So, only as you represent because you have to represent in two level form, so it will be either sum of product and product of sum. We do not have to think about multiple level, but still I have to find out that what can be the minimum number of gates to represent. So, in this case, it is equivalent to x , so just a single connection.

(Refer Slide Time: 25:46)

Introduction

- With the introduction of CMOS based standard cell and semi-custom design methodologies, there was a decline in the popularity of PLAs and PALs.
- When implementing a circuit with standard cells, it is customary to use multi-level implementation because generally a CMOS gate has a maximum of 4 fanins.
- The cost in terms of area or speed of a multi-level implementation is not directly related to the cost of an equivalent two-level circuit. However, the role of the two-level techniques is still important, because optimization of multilevel logic involves a network whose nodes represent functions, which are represented as two-level circuits.
- Therefore, in this (triple) lecture we discuss two-level Boolean logic synthesis (i.e., optimized two level implementation of a circuit for a given Boolean function). Following that, latter in this module we will also discuss multilevel logic synthesis.

NPTEL

So, we see some other techniques of minimizing this. So, the main idea here was so PLA and PLAs are developed long back. So, two level minimization was developed in the early 50s. So, those algorithms are quite matured, so with the introduction of CMOS, I mean that these things were come declining in the popularity of PLA and PLAs, but PLA and PLAs were continued to be used since long time. So, lot of algorithms for minimizing the two level numbers of gates, if we have a two level implementation, so PLA and PLAs were the most popular ones.

So, the algorithms have been quite good algorithms available or it has been these are saturation given a Boolean function. So, you can optimize in such a way so that they can be represented by the minimum number of gates in a two level fashion. So, if your architecture is two level, so the algorithm will give you Boolean function representation taking the minimum number of gates in two level representation. So, those algorithms are very strong. So, we will see those algorithms. So, if even, so we go in a two state algorithm basis.

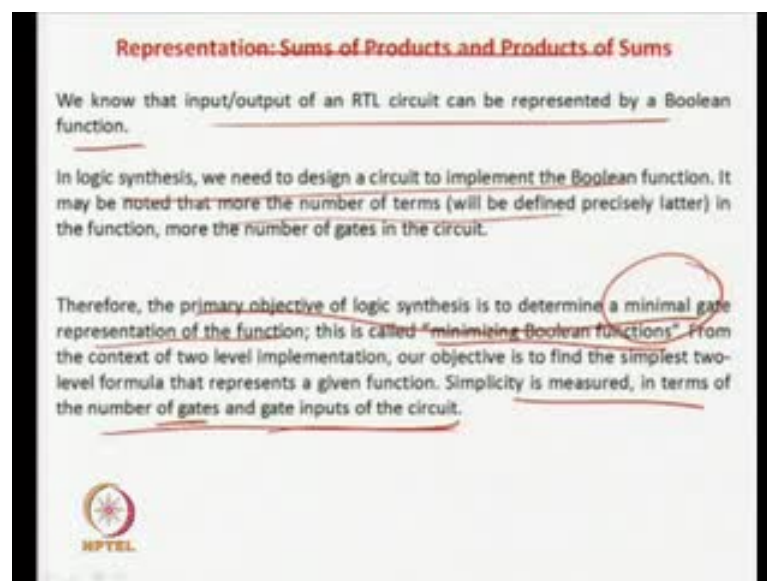
So, given any number of Boolean function, so what do you do? First go for a two level implementation and we minimize our circuits. So, minimize the number of gates, minimize the number of Boolean functions such so that they require the minimum number of gates to module level implementation and with this now arrival or not CMOS is there is become more popular PLA and PLAs. So, what do we do? Now, again from

two level to multiple level implementation, we do as a second step, so and because of CMOS maximum 4 input AND gates 4 input gates are already possible in CMOS.

So, what do you do? So, so what in this triple level, so we will see two level Boolean two level logic synthesis. Further, in the module we will see how we can convert into multiple logic synthesis. The basic idea because the two level logic synthesis algorithms are very powerful, they still in first phase two round. If you say that my final aim is the multiple level synthesis, since the process is two level, first we go for two levels.

First we go for a two level synthesis and then we minimize the function in such a way that the minimum number of gates is equivalent to implement the two level implementation. Then, from two level, we will go for multiple level implementation because wherever you find the number of inputs in a gate, it is more than 4, you have to break it up that is basically the idea.

(Refer Slide Time: 28:00)



So, that is I told about in this point because optimization of multilevel logic involves a network whose nodes represent functions, which are represented two level circuits. That is the basic idea because even if you go for multi level implementation, one of the important minterm representation is two level implementation. That is the idea.

So, we will see we already know the so, from now on, this triple lecture, we are going to progress on optimization of the circuits in two level form. So, assume that our circuit can

be implemented in two level logic, and then we will be minimizing the function, so that the numbers of gates are equal. So, we know that the input, output of the R T L can be represented by a Boolean function that is very very well known that in logic synthesis.

We will implement in actually in number of gates will be implementing in two level gates format that is AND, OR gate or AND plane depending on the sum of product or product of sum from. Our main idea will be that we have to minimize the number of gates to for this implementation. So, this is actually called minimizing Boolean function. So, whenever we say Boolean logic synthesis or Boolean synthesis or whatever, then we have to our main goal is we have to minimize the number of gates equivalent to this simplicity is measured in terms of the number of gates and gate inputs of the circuit.

So, obviously complexity is number of gates as well as the number inputs to a gate. So, if you are using 1, 2 input and AND gates, so obviously, two input AND gate is lower in cost and the 3 input AND gates because we are assumption is that we have, we can fabricate n input AND gates. If we n can be as large as possible, so as for the current time, so for the current triple lectures or assumption is that we are going for two level implementation.

So, our cost of implementation of a function is number of gates as well as the number of inputs to a gate because how we assume that gates n were n can be any large or any arbitrary order and also be fabricated.

(Refer Slide Time: 29:45)

Representation: Sums of Products and Products of Sums

Now we formally define a two-level formulae. Formula consists of constants, variables, parentheses and operators. A letter is a constant or a variable. A literal is a letter or its complement. For example, for $0, 1, x, y$ are letters and $0, 1, x, x', y$ are literals; $0, 1$ are constant literals and x', y' variable literals. The following definitions are introduced.

Definition 1: A product term is a formula of one of the following forms:

1. 1 x y x' $x'y$
2. a variable literal
3. a conjunction of variable literals where no letter appears more than once. $x'y'z$

Definition 2: A sum term is a formula of one of the following forms:

1. 0
2. a variable literal x, y, z
3. a disjunction of variable literals where no letter appears more than once. $x+y+z$

So, that is the primary objective of logic synthesis is to determine a minimal representation of the function. So, what do you mean by minimal gate representation? So, minimal or optimization gates is measured in terms of number of gates and the number of inputs to the gates. That is what we are going to do. So, we all know that this thing, so we can represent our stuff what do you can say that represent our Boolean functions in sum of products or products of sum forms. Now, let us quickly see some definitions. So, what is a product term?

(Refer Slide Time: 30:04)

Representation: Sums of Products and Products of Sums

For example, $x.y'$ is a product term, $x + y'$ is a sum term and x' is both. On the other hand, $x.x$ is neither product terms nor sum term, because the letter x appears twice and the term reduces to 0.

Definition 3: A Sum of Products (SOP) formula is one of the following:
 a product term;
 a disjunction of product terms.

Example, $f = x.y' + yz$

The cost of a SOP formula is determined by the number of product terms and the number of literals. Broadly, speaking, number of product terms determine the number of AND gates and number of literals determine the number of inputs of a gate.

$f = x.y' + yz$ has two product terms and four literals.

Handwritten notes: AND-OR, $xy + x = x$

So actually first there is something called a letter, ok , so what is the letter? So letter is a constant or a variable. Let us now see literal; literal is a letter or its complement. For example, zero, one, x and y are letters; zero, one, x, x prime, y are literals. In this case, zero, one are constant literals; x and y prime are variable literals. Ok This is the definition of literals and variable ah literals and letters. Now we see what is the product term, so one is the product term, we have to standard definition, a variable literal so x is the product term, y is the product term, x prime is the product term. A conjunction of variable literals where no letter appears more than once is also a product term, like x y z is the product term, by x prime y is also the product term. But x prime x is not a product term, because the variable x is there is twice.

But a sum term, sum term is a formula where zero is the sum term; a variable literal like x y z, individual x comma y comma z, individualized term sum term. A disjunction of variable literals where no letter appears more than once, like x plus y plus z this is

actually a what you can say $x + y + z$ or $x + y$ or $x + y + z$ plus appears on like this are nothing but sum term.

So now we can see some example $x \cdot y'$ is the product term, $x + y$ is the sum term, because you are actually have conjunction over here disjunction over here is sum term and x is both, right. You have a x is nothing but a is variable literal, ok so it is both a sum term and a product term. On the other hand, $x \cdot x$ is neither a product term nor sum term, because the letter x appears twice and the term reduces to zero, ok so that is what is the example and the definitions. Some definition is sum of product term so what is the sum of product term is called zero is the sum of product term; a product term like $x \cdot y \cdot z$ is a sum of product term. And a disjunction of product terms like $x \cdot y \cdot z + x \cdot y + x + \text{something like that}$ you can say y' or something like that.

So already we know that this is nothing but a sum of product term, this we already studied in our under graduate course. So this is actually a sum of product terms. So the cost of a sum of product term is determined by the number of product terms and the number of literals. And again we have said that we have to minimize something, so have to minimize this function, because if I want to minimize these function is we have to minimize in such a way, so that the modified, so the modified presentation is equivalent to s as well as it is, it will take less number of gates to implement it. So, what do you meant by the cost of the function, it is the number of product term like here we have two product terms. So if you have two product terms that means what you require one AND gate sorry one OR gate to do this. This is one OR gate.

If you have two product terms here, that means what you requires two AND gates ok and the number of literals, $x \cdot y$ and $y \cdot z$. So the number of literals is x is one and y term you have two and the you have two, that means you required two inputs get over and two input I get over here. So obviously so if you look at the circuit for this one quickly, it will be nothing but $x \cdot y'$ is an inverted and then now y and then z then the AND gate and that we have OR gate in this case of SOP. So OR gate over here, so you have a OR gate over here, this is the OR gate. This is $x \cdot y$, x sorry $x \cdot y'$ y and so, there are two literals and two literals, so you got two input AND gates, two input AND gates. And there are two product terms, so you have OR gates. So that when we are talking about the cost of a function or that is means how many gates will be required to implement the circuit, directly depends on the number of product terms and number of literals.

So broadly, speaking, the number of product terms determines the number of AND gates and the number of literals determine the number of inputs of the required gates. This one has two product terms and two literal terms. So in this case, we have already seen this one. Ok now a product of sum terms, in this case one is product of the sum term and sum term. So what is sum term, $x + y$ the sum term. And a conjunction of sum terms like $x + y$ $x' + z$ this is actually a POS form, right so this is actually a sum term, this is another sum term.

So, this is actually a sum term, these are the sum terms conjunctive, so it is a product form. So, again in this case also what are the cost of this determined by the number of sum terms and the sum terms are there any number of OR gates. These are the same way SOP formula. So, normally speaking this number of OR gates number of OR case. So, how many sum terms will be there; in this case two sum gates and two OR gates, so the number of intervals determined the number of intervals. So in case we have two sum terms, so there will be two OR gate: one OR gate will be here, and one OR gate will be there.

So, here and two inputs OR gates will decode, and this will be corresponding to one gate. So, now two level formula can be either sum of product term or product of sum terms. So, in this lecture we will discuss both of them because we know both of them are dual of each other or De-Morgan's theorem, so for this we are not going to discuss both sum of product or product of sum there, SOP form which is more popular. So, if you look at this form to actually here AND OR architecture we will be looking into this one.

Now, we will see, we are now slowly going towards the case stand we have a function, function like this. How we can represent using another equivalent function like $10x + yz + x'$? So, these are sum of the product form. This is equivalent to nothing but equivalent to x . Our main goal is that how can you minimize this function that is how you can represent the function another equivalent to the number of literals as well as the number of sums is less.

(Refer Slide Time: 35:30)

Prime Implicants

There are two basic steps for minimizing Boolean functions namely, determining prime implicants and then finding subset such implicants that cover all product terms of a function. We introduce the concept of prime implicants and schemes to determine prime implicants.

Definition 5: An implicant of a function is a product term that is included in the function.

For instance, xyz is an implicant of $f(x,y,z) = xy + xyz + xyz'$.

Definition 6: A prime implicant of a function is an implicant that is not included in any other implicant of the function.

For instance, xyz is not a prime implicant of $f(x,y,z) = xy + x'y'z$ because xyz is contained in xy . xy is a prime implicant of $f(x,y,z)$ because it is not contained in any other implicant.

So, if an implicant is not prime, then it is possible to obtain prime implicant of by removing some literals from it.

So, for that, we are going to look at some difference. The two basic steps are minimizing the Boolean functions namely, determining the prime implicants and finding the subset of such implicants, which cover all the terms in the product we saw the this is what for the time being find implicants subsets covering etcetera may not be slowly we are looking at the definitions things will start becoming more clear.

So, what is an implicant of a function? Implicant of a function is a product term that is included in the function. That is the definition. That means, for example, $x y$ is an implicant of $x y$. So, if I say f of $x y$ is a function, this is actually $x y$. So, what is $x y$? $x y$ represents the $x y z$ and $x y z$ prime because f is a function of 3 input variables say $x y$ and z . So, if you are representing a is equal to $x y$ that means x the said actually is the $x y z$ and $x y z$ prime. So, obviously, this is an implicant of a function $x y$ because it is a part of this function.

Now, definition six, so definition six it says that a prime implicant of a function is an implicant that is not included in any other implicant of the function. That means, in this case, let us see by an example. So, prime implicant is an implicant of a special type of implicant, so that it is not included in any other implicant of the function. So, function can have lot of implicants.

So, this is one implicant, this is one implicant and this is one implicant. So, in this case, you can say that the two implicants of the function, so you can say when a function

implicant is a prime implicant, if a prime implicant, if and only if this is not included in any other implicant.

So, we will see, for example, this $x y z$ is not a prime implicant of this one. Why? It is a new function? It is a new function like this. What is the new function? The new function is say that f of $x y z$ is equal to f of $x y$ plus $x y$ prime z prime because so this one is not a prime implicant of this one because it can be expanded into $x y z$ and it will become primes of $x y$ and z prime.

Now, this term, this product term can be represented. It comprises of two other implicants, this $x y z$ and $x y z$ prime. Now, we can say that, so $x y z$ is the implicant of this function, but $x y z$ is not a prime implicant of this one because $x y z$ is contained of this one because $x y z$ is contained in this one. We saw if I say that $x y z$ is a implicant of this function because this is the function.

Now, $x y z$ is an implicant of the function because it is comprised in this function because $x y$ can be represented as this one. So, obviously, this is included in any other function, it is not a prime implicant, because $x y z$ is included in the implicant $x y$ of the function. So, it is not a prime implicant, but $x y$ is prime implicant of the function. Now, you see $x y$ is, $x y$ is also a implicant because $x y$ is included in the function.

So, obviously, $x y$ is included in the function. This is obviously, its term, its product term is also an implicant of a function that is very obvious because $x y$ is included in the function. This is a prime implicant because $x y$ is not contained in any other implicant.

So, if you think that what is the function of three implicants $x y z$ and $x y z$ prime, sum of the implicant function, so you can easily obvious that $x y$ is not contained in this one. So, in this term, so obviously, $x y$ is a prime implicant because $x y$ is not contained in any other implicant. So, if I implicant this not prime and it is possible to obtain a prime implicant by removing some literals out of it, that means, what is not prime implicant?

Now, if I remove x from this and still obviously, if you are removing some literals out of this should belong to the function. So, in this case, if I remove x , so obviously $x y$ belongs to the function, it will become a prime implicant. So, that is what being told in this definition.

(Refer Slide Time: 38:50)

Prime Implicants

Definition 7: If a prime implicant includes a minterm that is not included in any other prime implicant, then that prime implicant is essential.

For example, $f(x,y,z) = xy + x'y'z'$ has two prime implicants namely, xy and $x'y'z'$. Prime implicant xy is essential because xy contains xyz and xyz' which are not contained in any other prime implicant (i.e., $x'y'z'$). In another function $f(x,y,z) = xy + xy' + xz'$, prime implicants are xy , xy' and xz' . Among them, xz' is not essential because $xz' = \{xy'z', xz'\}$ and $xy'z'$ is in xy' and xz' is in xy .

So, next one is essential prime implicant. If a prime implicant includes a minterm that is not included in any other prime implicant, then that prime implicant is essential. So, what does that mean? It means that a prime implicant is what it is saying that a prime implicant like in this case, in this function, $x y$ plus x prime $y z$ prime.

(Refer Slide Time: 40:56)

Prime Implicants

Definition 7: If a prime implicant includes a minterm that is not included in any other prime implicant, then that prime implicant is essential.

For example, $f(x,y,z) = xy + x'y'z'$ has two prime implicants namely, xy and $x'y'z'$. Prime implicant xy is essential because xy contains xyz and xyz' which are not contained in any other prime implicant (i.e., $x'y'z'$). In another function $f(x,y,z) = xy + xy' + xz'$, prime implicants are xy , xy' and xz' . Among them, xz' is not essential because $xz' = \{xy'z', xz'\}$ and $xy'z'$ is in xy' and xz' is in xy .

So, why we call it as a prime implicant? It is because it is a prime implicant because this is not included in any other implicants. So, that means, it is a prime implicant. Now, $x y$

z is also implicant of this function or of this function, but we call but $x y z$ is not prime implicant why? This is because this is comprised in one implicant of this function.

So, this is not a prime implicant. Now, we see what you mean by essential prime implicant. So, the prime implicant that is included, some minterm that is not included in any other prime implicant, then that prime implicant is called as essential prime implicant. So, we will explain by an example. So, we say that for example, this is having two prime implicants namely this and this obviously, there are two prime implicants because neither included in this nor this one is included in this.

So, both of them are prime implicants because $x y$ is not included in x prime, y prime, z prime and vice versa. The prime implicant x is essential because it contains this, which are not included in any other prime implicant. That means what this is actually having two implicants that is $x y$ say $n x$ prime sorry $x y z$ and $x y$ and z prime. Now, you see both of them are not included in this prime implicant.

So, that means, $x y$ is have a minterm. So, that is actually, these are actually called minterm. So, if you remember the digital undergraduate course, so all these product terms are actually called as minterms in some product form. So, these terms are not included in any other prime implicants. So, it becomes essential prime implicants. So, why do you call essential prime implicants because this prime implicant has some what do you call minterm or some product term not included in any other prime implicant.

So, to represent this function, always this prime implicant has to be included, but if there is a prime implicant, which is not essential, it may be dropped like we will see in another function like this one like $x y$ plus $x y$ plus $x z$ prime, the prime implicants are this, this and this equation is usually verified, because this is neither included in here nor included here and vice versa.

(Refer Slide Time: 42:18)

Prime Implicants

Definition 7: If a prime implicant includes a minterm that is not included in any other prime implicant, then that prime implicant is essential.

For example, $f(x,y,z) = xy + x'y'z'$ has two prime implicants namely, xy and $x'y'z'$. Prime implicant xy is essential because xy contains xyz and xyz' which are not contained in any other prime implicant (i.e., $x'y'z'$). In another function $f(x,y,z) = xy + xz'$, prime implicants are xy , xy' and xz' . Among them xz' is not essential because $xz' = \{xy'z', xyz'\}$ and $xy'z'$ is in xy' and xyz' is in xy .

So, none of the terms can be included in one another. So, it is all these are prime implicants, but some of them may not be essential prime implicants because xz' prime, if you represent see xz' prime it is nothing but xz' prime z' prime and xzy' prime. If you can open this term, now you can see xzy' prime z' prime is in xzy' prime. So, what is xzy' prime? xzy' prime is nothing but xzy' prime z' and xzy' prime z' prime.

So, if I open the term, I will get this. If I open the term, I will get xzy' and $xzy'z'$ prime. If I open this now, you see xz' , I am breaking into xzy' prime, $xzy'z'$ prime, you can see xzy' prime z' prime is actually included in this guy xzy' prime z' prime, this one xzy' prime z' prime, this guy is already included in this and xzy' prime z' prime is actually included in xzy' .

So, this guy is having two literals, what do you call? What do you say two minterms, we one is included in this and one is included in this. So, this prime implicant is not an essential prime implicant because I can be dropped because this. So, in other words, what do I say? What they mean to say is that among them, this is not an essential prime implicant. Why? This is because its components are evaluated in some other prime implicants.

(Refer Slide Time: 42:50)

Determining Prime Implicants

Theorem 1: If cost of a Boolean function depends on literals then a minimal SOP must always consist of a sum of prime implicants.

Proof: Let us assume that f is an SOP which is minimal and one (product) term is non-prime. Another SOP formula $f_1 = f$ exists, that can be obtained by replacing the non-prime implicant by a prime implicant that contains it. The cost does not increase and the formula is equivalent to the original one. So, we reach a contradiction i.e., f is an SOP which is minimal and all terms are prime implicants.

For example, let SOP representation of $f(x,y)$ be $x + x'y$. A prime implicant of f is y . The SOP representation can be changed as $x + y$, where we have replaced $x'y$ with y ; y includes $x'y$. So the new SOP is rewritten by saving one literal.

MPTEL

So, it becomes a small essential prime implicant and why these are essential implicants? This is because they have some minterms or some product term, which is not included in any other product term any other prime implicants. That means, what we represent the function. So, we have to obviously have the minimum, we should have, you should have the essential prime implicants essential prime implicants will have some terms, which is not included in any other term.

If a prime implicant is not essential, then we can drop because its components are not available in some other terms. Now, you see how long, now you can see that if you have to about to determining a minimal representative minimum Boolean representation function of sum of product form. Then, we have to find out the prime implicants. So, we have to find first prime implicants. Then, we have to find out what are the essential prime implicants.

So, if you find out the essential prime implicants essential prime implicants, they have to be present. If you if you drop an essential prime implicants, then that means we are losing something.

That means, essential prime implicants some term we product of which is not available any other prime implicants, obviously that has to come into picture the implicants. Then, we can find out the essential prime implicants, and then more or less your job is done. So, because then you are finding out some terms, which have or some implicants, which

have product terms, which are not available in any other. So, they have to be included and some terms like this one subprime implicants.

So, in this case, what happens? In this case, this is a prime implicant, but still it is available in two other terms part by part. So, you can drop it. So, our step is in two ways. First, you have to find out what are the prime implicants. What are the prime implicants? Prime implicants are the terms, which are not included in any other. See there is a term, which is totally included in other like example, we are taking $x y z$ plus x .

(Refer Slide Time: 44:37)

Determining Prime Implicants

Theorem 1: If cost of a Boolean function depends on literals then a minimal SOP must always consist of a sum of prime implicants.

Proof: Let us assume that f is an SOP which is minimal and one (product) term is non-prime. Another SOP formula $f_1 = f$ exists, that can be obtained by replacing the non-prime implicant by a prime implicant that contains it. The cost does not increase and the formula is equivalent to the original one. So, we reach a contradiction i.e., f is an SOP which is minimal and all terms are prime implicants.

For example, let SOP representation of $f(x,y)$ be $x + x'y$. A prime implicant of f is y . The SOP representation can be changed as $x + y$, where we have replaced $x'y$ with y ; y includes $x'y$. So the new SOP is rewritten by saving one literal.

Handwritten notes: 18, x'y, y

So, in this case, this is not at all a prime implicant. Why? This is because this is totally embedded in x because x can be represented as $x y x$ prime z prime and $x y z$ prime and $x y$ prime z and then $x y z$, if you bring it up, it can be open up into four terms. So, obviously, this is included in this one. So, this is not a prime implicant.

So, you can directly drop it. Then, you remain with only the prime implicants, and then we find out what are the essential prime implicants? This is because essential prime implicants are the ones, which has a term, which is not included in any other term. So, you obviously you have to have the essential prime implicants. So, once you can go about this, then we get the very minimum representation of your circuit.

So, what is the theorem? It says that if the code of the Boolean function depends on literals, then a minimal SOP must always consist of a sum of prime implicants. That is

the proof is very obvious. Let us assume that if it is a SOP, which is minimal and one product term is non prime. That means, what will we say that if you are going for minimal representation of SOP form.

So, you should not have any kind of non prime implicants because non prime implicant is what is already including. Now, so if you prove it in a formal way, so we will go for a formal way counted logic proof contra positivity, proof kind of a thing, let us assume that which is minimal and one product term is non prime.

So, another SOP will obviously exist, which is equivalent that can be obviously obtained by representing non prime implicant because another implicant actually another prime implicant will comprise the non prime implicant, which is equivalent. So, obviously you can remove the non prime implicant. So, the cost does not increase and the formula is equivalent sorry the formula is equivalent always, so it is a contradiction. So, SOP is minimal on there in terms of prime implicants.

So, how the proof goes? The proof says that if you ever the minimal representation of SOP terms, all the product terms should be prime. So, assume that there is a product term on prime. Now, obviously, this function is extra term, which is a non prime. Now, you can obviously remove the non prime term and should be still equivalent because how the non prime implicant is included in some other prime implicant. So, obviously, you can remove the non prime implicant, the function will become smaller in size. So, it will be less complex and it will be easy to equivalent.

So, obviously, if you assume that SOP, if it is an S O P, which is minimal and yet one product term is non prime is false. Let us see the example. So, let this be the explanation the prime implicant of f is y for a prime implicant of f is y because there is no other term, which there is no other term, which will there is no other I mean term in this case, which will actually comprise of y. So, the SOP can be represented to x plus y x plus sorry the expression is x plus x plus x prime y. So, you can be represented it in equivalently in x plus y.

(Refer Slide Time: 46:43)

Determining Prime Implicants

Theorem 1: If cost of a Boolean function depends on literals then a minimal SOP must always consist of a sum of prime implicants.

Proof: Let us assume that f is a SOP which is minimal and one (product) term is non-prime. Another SOP formula $f_1 = f$ exists, that can be obtained by replacing the non-prime implicant by a prime implicant that contains it. The cost does not increase and the formula is equivalent to the original one. So, we reach a contradiction i.e., f is an SOP which is minimal and all terms are prime implicants.

For example, let SOP representation of $f(x,y)$ be $x + x'y$. A prime implicant of f is y . The SOP representation can be changed as $x + y$, where we have replaced $x'y$ with y ; y includes $x'y$. So the new SOP is rewritten by saving one literal.

NPTEL

Why? This is because you can see that x prime y is actually included in this. So, x prime y is not a prime implicant because y is a prime implicant because y cannot be included in any other implicant. So, that is why it becomes a prime implicant. Now, x prime y does not it is not a prime implicant because x prime y can be included. So, y can be represented like this one. So, we have represented this one in this.

So, the new prime will be by this one. So, we are removing one literal like x prime has been removed, so some expression becomes simpler. Now, it has two prime implicants x plus y . That is what the idea is. So, that is I mean this is a definition obviously, which is also simple or to understand that minimize your circuits.


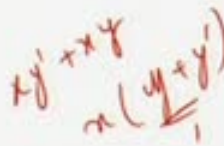
So, if you want to minimize your representation, then what we have to do? You have to find out the prime implicant, you have to take them. So, if n term is non prime, so that actually will not form a minimal representation. So, that is what being what is being understand.

(Refer Slide Time: 47:36)

Determining Prime Implicants by Tabular Method

Definition 8: Given a Boolean function in SOP form $\overline{X}y + Xy'$, where X is a product term not having variable y , then consensus can be applied on the two terms $\overline{X}y, Xy'$ to generate $X(\overline{X}y + Xy')$; X is the consensus term containing both $\overline{X}y, Xy'$. This is also called distance-1 merging.

In other words, pairs of terms that differ in exactly one letter, which must appear complemented in one term and un-complemented in the other, are used for consensus.



Now, we have to find out some algorithm or some automated way where you can do it or how can you find out; given a function, you have to find out the prime implicants. Then, we can find out, how we can find out the essential prime implicants.


Now, we will go step wise. So, first thing we have seen is we have to eliminate out all the non prime implicants whose having a prime implicants, which is already proved that be problem be headed because I already having some terms already present in another prime implicants. So, better remove all the prime implicants, then the question arises can you do it? There are different ways of doing it.

(Refer Slide Time: 48:47)

Determining Prime Implicants by Tabular Method

Given a Boolean formula in SOP form we need to determine the prime implicants. The steps of tabular method are given below.

1. Express the function in minterm canonical form
2. Consider all pairs of adjacent terms, i.e., the pairs of terms to which consensus can be applied. The consensus terms are implicants of the function though not necessarily prime. All terms that form these new consensus terms are included in the new terms, and hence they are not prime. We mark the old terms as non-prime and are not used for further consensus.
3. The new terms are added to the SOP and steps 1,2 are repeated till no more consensus terms can be found

 All terms that are absorbed (or contained) by the new terms are marked as non-primes. Finally, the terms that are not marked constitute all the prime implicants of the function.

So, today we will see the tabular method. What is the tabular method? The tabular method what is like this, so if you have a SOP form $x y$ prime and x sorry $x y x y$ prime where the x is a product term not having the variable one y . Then, it can be then can be a consensus to generate this one.

(Refer Slide Time: 48:55)

Determining Prime Implicants by Tabular Method

Now we illustrate the scheme using an example.
Consider the SOP: $f(w, x, y, z) = x'y'z + wxy' + x'yz'$.

The function in minterm canonical form is as follows:

$$w'x'y'z' + w'x'y'z + wx'y'z' + wx'y'z + wxyz' + wxyz + w'x'yz' + wx'yz'$$

Handwritten notes: $x'yz'$, wxy' , $wxyz'$, $wxyz$, $w'x'yz'$, $wx'yz'$

- Now we construct a table, where the minterms appearing in the canonical form are entered.
- The column is divided into four parts based on number of complemented letters of the terms.
- The first group consists of minterms with no un-complemented literals.
- In general, some groups may be empty. So, for consensus, we need to compare terms of immediately successive groups.
- It may be noted that we need not consider the minterms in the immediately preceding group, because this would only cause us to repeat comparisons.

So, we can have, so we say that $x y$ prime plus $x y$, so does not have the term y . Then, obviously, it will become a x common y plus y prime change to one. So, finally, we will

have this one. So, it is also called on distance merging because we have we cancel out the y and y prime and we have this one. Also, this is called consensus.

(Refer Slide Time: 49:46)

Determining Prime Implicants by Tabular Method

Now we illustrate the scheme using an example.

Consider the SOP: $f(w, x, y, z) = x'y' + wxy + x'yz'$

The function in minterm canonical form is as follows:

$$w'x'y'z' + w'x'y'z + wx'y'z' + wx'y'z + wxyz' + wxyz + w'x'yz' + wx'yz'$$

- Now we construct a table, where the minterms appearing in the canonical form are entered.
- The column is divided into four parts based on number of complemented letters of the terms.
- The first group consists of minterms with no un-complemented literals.
- In general, some groups may be empty. So, for consensus, we need to compare terms of immediately successive groups.
- It may be noted that we need not consider the minterms in the immediately preceding group, because this would only cause us to repeat comparisons.

Using this consensus, we will try to see how we can go about the finding out the primes. So, let us see there is we will have a look at the algorithm. So, we first represent the SOP form in canonical form. So, what is a canonical form? We have already seen in undergraduate course. So, if you have a function like $x y' + x' y z$, so this is a function of four variables term $x y z$. So, somebody has written it $x' y' + w x y + x' y z'$. So, we will have $w' z'$ and $x y x' y'$ will be common, then it will be $w' z'$. Then, also we can have $w' x' y' z$, then you will have $w' x' y' z$. So, this $x' y'$ is common. You can have $w' z' w z'$ and $w z$. So, this is how you can open up. So, if you are opening, this actually forms the canonical representation.

So, what do you mean by canonical way? So, this actually represents by $x' y'$, which can be if you have to open the term canonical means opening the term that $x' y'$, then $x' y'$.

So, we will have $w' z'$ and $x y x' y'$ will be common, then it will be $w' z'$. Then, also we can have $w' x' y' z$, then you will have $w' x' y' z$. So, this $x' y'$ is common. You can have $w' z' w z'$ and $w z$. So, this is how you can open up. So, if you are opening, this actually forms the canonical representation.

So, if you just look at this function, this can be represented in canonical way in this function. So, we can look at these terms. So, we will see $x' y' + w' x' y' z$

prime z prime, then z prime w prime x prime y prime z, then w x prime this on and this one this four terms actually coming from this. So, this will be giving these two terms and so forth.

(Refer Slide Time: 50:11)

Determining Prime Implicants by Tabular Method

Given a Boolean formula in SOP form we need to determine the prime implicants. The steps of tabular method are given below.

1. Express the function in minterm canonical form
2. Consider all pairs of adjacent terms, i.e., the pairs of terms to which consensus can be applied. The consensus terms are implicants of the function though not necessarily prime. All terms that form these new consensus terms are included in the new terms, and hence they are not prime. We mark the old terms as non-prime and are not used for further consensus.
3. The new terms are added to the SOP and steps 1,2 are repeated till no more consensus terms can be found

All terms that are absorbed (or contained) by the new terms are marked as non-primes. Finally, the terms that are not marked constitute all the prime implicants of the function.

Note: The slide contains handwritten red annotations. A circle highlights the second step, and another circle highlights the third step. There are also some scribbles and lines drawn over the text.

So, these are we are actually representing in a canonical way. So, this is the first step now, what will we do? We see all there adjacent terms that is the pairs of terms to which consensus can be applied, we will see with an example. Then, the idea will be clear. So, we take two terms of this consensus or two terms of the canonical form and then we can find out if there is a consensus. So, what is the consensus? Consensus means $x y$ prime plus $x y$. So, you can cut this x and sorry cut the y , so will become a x and then we find out if there is a consensus.

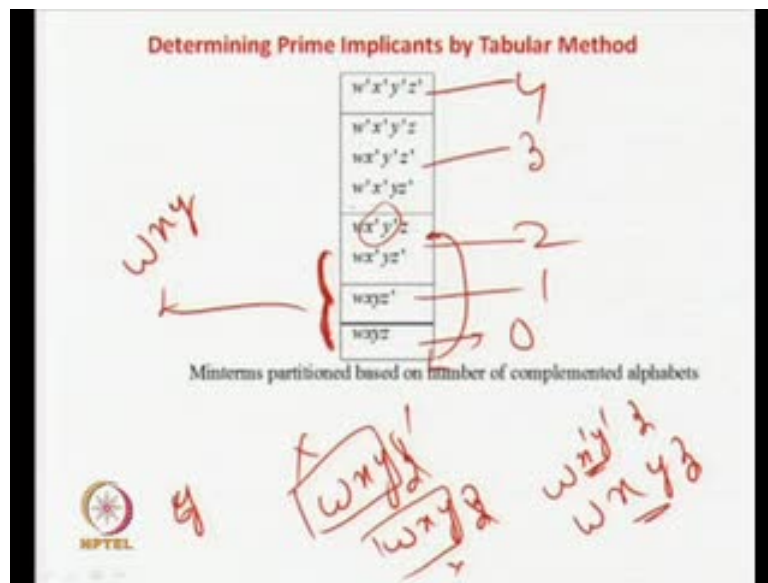
So, the consensus is the implicants to the prime. See, about that this is the next step and then finally, we repeat this steps till no more consensus can be found. So, the idea is that we may first canonical form. Then, we may take terms at a time or in a step, we try to find out the consensus and we keep on doing it till no more consensus can be found out.

So, all the terms that form the consensus are included in the old terms as they as they are not prime will mark the old terms. As they are not prime, we will mark the old terms and non prime not usual for the consensus. So, we consider this example in more clear it says that if there are two terms from which are the later consensus like for example, $x y$ prime plus $x y$.

So, you cut out this X and finally, it will remain as capital X. So, obviously this capital X will comprise $x y$ prime and $x y$. So, this will now become non prime and x will become a prime that is the idea because this $x y$ prime and $x y$ are comprised in x . So, obviously these things we can determine because no longer will remain plus only $x y$. That is what is been called we mark the old terms as non prime and are not used for further consensus that is the idea and we keep on repeating it till no more consensus be there.

Finally, you do all terms that are absorbed by the new terms are marked as non pairs. This the terms that are marked constitute all the prime implicants of the function, which ever terms are consumed consensus are removed and finally keep on doing it. Finally, the terms which could not found out, find the consensus and they become prime implicants because they are included in any other term. So, let us take with an example.

(Refer Slide Time: 52:04)



So, this is your example we saw. What we do is that first we divide, we make with the table and it into four parts. So, here there is no primes $w x y z$, none of the terms is having a prime here. What you are doing? Here only one term is having a prime like $w x y z$.

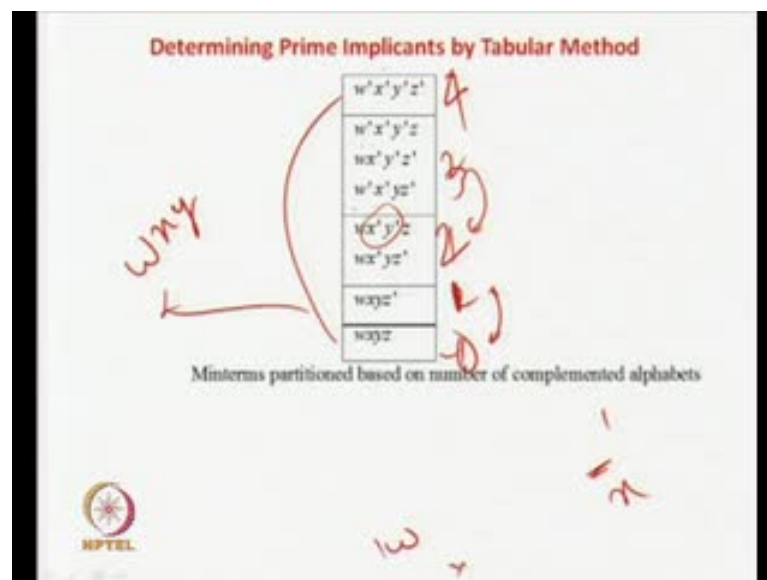
So, only one term is having a prime, here two terms are having a prime it is $w x$ prime y prime z . So, two terms sorry two what do you call alphabets or literals are having a prime here $w x$ prime z prime. So, here two literals are having a prime here three literals are

having a prime that is $w'x'y'z'$ $w'x'y'z$ $w'x'yz'$ $w'xyz'$ and here $w'x'y'z'$ $w'x'y'z$ $w'x'yz'$ $w'xyz'$.

So, three literals are having a prime here four literals. So, we are actually dividing it into $w'x'y'z'$ $w'x'y'z$ $w'x'yz'$ $w'xyz'$. We are dividing it because you see in if it does, see those terms $w'x'y'z'$ and $w'x'y'z$, so in this case $w'x'y'z'$ and $w'x'y'z$. So, you consider this guy as capital X and this guy as capital X.

So, it will be this $w'x'y'z'$ $w'x'y'z$ will be cut off and finally, will, this two will make a consensus term will be $w'x'y'$. So, that is why we are actually in this, we are dividing the table in such a way so that if you just compare these elements from this to, you get a consensus. Similarly, if you compare these two, we will get a consensus, but you cannot because here none of the term is complimented, here two terms are complimented. So, if you compare this one and this one, we never get any consensus because you are having $w'x'y'z'$ and here we are having say $w'x'y'z$, so two terms in which there is a difference in any terms of compliments.

(Refer Slide Time: 53:43)



So, there and here, you are having some say $w'x'y'z'$ $w'x'y'z$. So, you see there are two terms in which there is a difference in the terms of compliments. So, there cannot be any consensus.

(Refer Slide Time: 54:03)

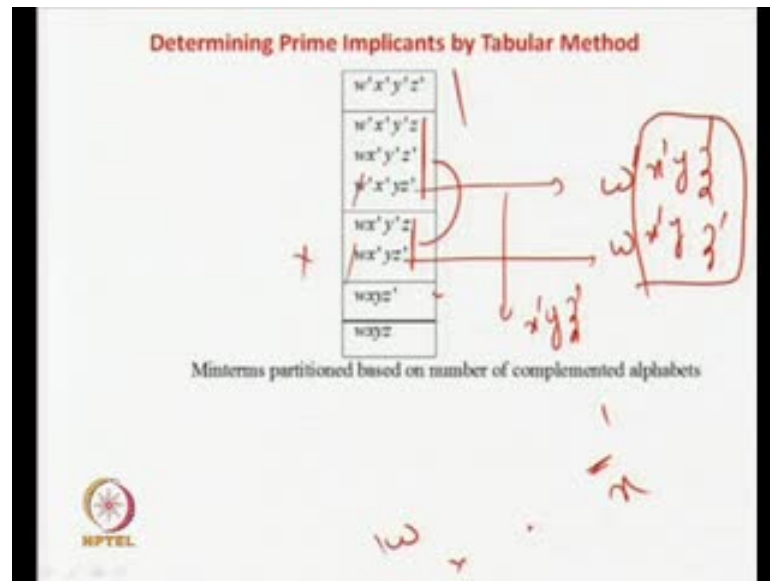
Determining Prime Implicants by Tabular Method

$w'x'y'z'$
$w'x'y'z$
$wx'y'z'$
$w'x'yz'$
$wx'y'z$
$wx'yz'$
$wxyz'$
$wxyz$

Miniterms partitioned based on number of complemented alphabets

So, that is why the table is made in very intelligent way so that this table is divided in such a way, so here we are having one compliment sorry zero compliment, two compliment, one, two, four and four, four compliment zero, one, two, three and four compliment. So, these two guys terms can have consensus. Similarly, this two and this two, so we cannot have consensus from here up to here because here there are no compliments, there are four compliments. So, what do you have to do that is let us find out whether there are any consensus between this two terms or obviously, so it will generate $x y z$ prime. Similarly, you can have to compare this one with both of this. So, if you compare with both of this, all we will get, so we have done $w x$ prime y prime z and here it is $w x y z$ prime.

(Refer Slide Time: 54:49)



So, these two guys will not have any consensus, but if you compare this two guys see, so you have a $w'xy'z$ prime, which is this one and here it is $w'xyz$ prime y is not having this and z prime is present. So, this one is not there. So, just if I take this term over here, now you can easily see that I do not have a consensus over this. So, these two terms are not having any consensus. Now, this two do not have any consensus. So, what we can do is we can try out with terms for these two terms. All these two terms will have to be compared with all the three terms.

So, you can very easily see that if I take this, so it will be $w'xy'z$ prime x prime y z prime x prime y z prime and this guy is having $w'xyz$ prime. So, this is the guy I am taking. You can see that in this case, if you see very clearly, this part is actually x and w can be eliminated.


(Refer Slide Time: 55:29)

Determining Prime Implicants by Tabular Method

In the example, if we compare the term from the first row ($w'x'y'z'$) with the top most term of second row ($w'x'y'z$), their consensus term is $w'x'y'$.

Both $w'x'y'z'$ and $w'x'y'z$ are marked as non-prime because there exists another implicant $w'x'y'$ that contains them.

A second column is created in the table where in the first row the consensus term $w'x'y'$ is placed.




So, if you take this, so this w prime and this w can be eliminated because x prime y z prime is also x prime y prime z. Finally, this will give you a consensus term of x prime y and z prime. So, this how about we know about consensus, so this is what I have told you. So, if you compare this with this, so you get the consensus term this. Similarly, this are marked as non prime because now this two will be non prime, this will be non prime and this one will be prime because his two guys are included over here. This second column is created where this new term will be placed.

(Refer Slide Time: 55:50)

Determining Prime Implicants by Tabular Method

$w'x'y'z'$	(not prime)	$w'x'y'$ P
$w'x'y'z$	(not prime)	
$wx'y'z'$		
$w'x'yz'$		
$wx'y'z$		
$wx'yz'$		
$wxyz'$		wxy P
$wxyz$		

Entry after consensus of $w'x'y'z'$ and $w'x'y'z$




So, if you look at this table sorry if you look the table, you will understand the terms. So, what do we have done? So, we can we can start with this two.

(Refer Slide Time: 56:27)

Determining Prime Implicants by Tabular Method

$w'x'y'z'$ (not prime)	$w'x'y'$	3
$w'x'y'z$ (not prime)	$x'y'z'$	
$w'x'y'z'$ (not prime)	$w'x'z'$	
$w'x'y'z$ (not prime)	$x'y'z$	2
$wx'y'z'$ (not prime)	$wx'y'$	
$w'x'yz'$ (not prime)	$wx'z'$	
$wx'y'z$ (not prime)	$x'yz'$	1
$wx'yz'$ (not prime)	wyz'	
$wxyz'$ (not prime)	wxy	
$wxyz$ (not prime)		0

Entry after consensus of all terms of first table



So, this two will get as y. So, w x y z prime will be cut. So, this will be one consensus stuff. So, now this will be a non prime this will be a non prime and this will be a prime over here. Now, in this case also, we can see if you are having trying this two, so it will be w prime x prime y prime. So, this will actually become your x power and this z and this prime will get cut off. So, this is the new term available for w prime x prime and z prime, this is the consensus stuff. Now, this will be actually your prime, this will be prime and this we mark as non-prime. So, if you try out for whole thing, you will find out a table like this. So, you can find out that we can have a consensus. You can just see and try out, we get this entire thing.

(Refer Slide Time: 57:03)

Determining Prime Implicants by Tabular Method

$w'x'y'z'$ (not prime)	$w'x'y'$ (not prime)	$x'y'$
$w'x'y'z$ (not prime)	$x'y'z'$ (not prime)	$x'z'$
$w'x'y'z'$ (not prime)	$w'x'z'$ (not prime)	
$w'x'yz'$ (not prime)	$x'y'z$ (not prime)	
$wx'y'z'$ (not prime)	$wx'y'$ (not prime)	
$wx'yz'$ (not prime)	$wx'z'$ (not prime)	
$wxyz'$ (not prime)	$x'yz'$ (not prime)	
$wxyz$ (not prime)	wyz'	
	wyz	
	$wxyz'$ (not prime)	
	$wxyz$ (not prime)	

Entry after consensus of all terms of column 2

So here, we have four prime implicants as wyz' , wyz , $x'y'$, $x'z'$.

So, we can find out all of these terms like this one with this one or this one with this one or you put a find out the consensus. This is your new table, the next part of the table. So, we can find out this term very included in at least one of the consensus. So, none of them are prime and here everything is a prime for you. So, here also, you can see zero number of negations or compliment, this is one, this is two and this is three.

So, again you have to repeat this schedule by taking this terms from this two rows, then again terms from this two rows, then again we check from this two rows. Finally, you are going to get as this two, you will get by you will find out this two terms could not have a consensus because it is $w x y$ and this $w x z$ prime, this two stuffs could not have any consensus.

Now, again if you try to see that if you have to find out that if you try this three elements and try to compare this four elements, we can compare will get this two consensus term that is x prime y prime and $x z$ prime. So, all this terms are actually included in these terms because we could find this terms by consensus having a consensus among this. For example, if this one says that it is nothing but w prime x prime and y prime, so if we have a term like w prime sorry x prime and y prime, so here you are having a term like this term. If you take I is actually by x prime y prime and w , so this w will be eliminated. Finally, we get the term as x prime y prime and x prime y prime this one actually this two is generating this one.

(Refer Slide Time: 58:24)

Determining Prime Implicants by Tabular Method

$w'x'y'z'$ (not prime)	$w'x'y'$ (not prime) $x'y'z'$ (not prime) $w'x'z'$ (not prime)	$x'y'$ $x'z'$
$w'x'y'z$ (not prime)	$x'y'z$ (not prime)	
$wx'y'z'$ (not prime)	$wx'y'$ (not prime)	
$w'x'yz'$ (not prime)	$wx'z'$ (not prime)	
	$x'yz'$ (not prime)	
$wx'y'z$ (not prime)	wyz'	
$wx'yz'$ (not prime)		
$wxyz$ (not prime)	wxy	
$wxyz$ (not prime)		

Entry after consensus of all terms of column 2

So here, we have four prime implicants as $wyz', wxy, x'y', x'z'$.

So, this we can find out that all these terms to take a consensus we will get the result the final terms. So, all both of them are included over here, but these two terms do not have any consensus anymore. So, they will still remain. So, this one is two prime implicants and this one is two prime implicants. So, we have four prime implicants like this one. So, you can see that now my function is, but the four prime implicants of these functions are thing, but wxz prime wxy prime yz prime and z prime. So, this is my all the prime implicants are here.

So, you can think that they are having at least some implicants or some mean terms or some product terms, which is not available in any other like this. I will have some terms this I will have some terms this guy will have some terms and this is having some terms, which are not available sorry I mean this are all prime implicants because this guy is not available here this guy cannot be included over and this guy totally marked over here. We are not actually finding out the essential prime, we are only finding out the prime implicants.

(Refer Slide Time: 59:46)

Determining Prime Implicants by Tabular Method

Now we illustrate the scheme using an example.

Consider the SOP: $f(w, x, y, z) = x^2y' + wxy + x^2z'$.

The function in minterm canonical form is as follows:

$w^2x^2y^2z' + w^2x^2y^2z + wx^2y^2z' + wx^2y^2z + wxyz' + wxyz + w^2x^2y^2z' + wx^2y^2z'$

- Now we construct a table, where the minterms appearing in the canonical form are entered.
- The column is divided into four parts based on number of complemented letters of the terms.
- The first group consists of minterms with no un-complemented literals.
- In general, some groups may be empty. So, for consensus, we need to compare terms of immediately successive groups.
- It may be noted that we need not consider the minterms in the immediately preceding group, because this would only cause us to repeat comparisons.

So, what are the prime implicants means? That means, this term cannot be totally included here for here and this true for all others. So, essential prime implicant means it will, it should have at least one term, which is not available in any other. So, that will come in the next stage. So, first what we have done? First we have eliminated all the non essential prime sorry we have eliminated all the terms which are non primes. Non primes means what like this is a term, so it is actually included in this term. So, $w x y z$ is included in $w x y$, so this is not a prime term.

So, it can be eliminated. As of now in the first step, what we have done? We have taken a function and by using the tabular method, what we have done? We have right to eliminate all the terms or we can call the product terms, which are non primes. That means, they can be included directly in any of the prime implicant. So, we are actually having this set of four prime implicants. So, they are actually minimal way, then the original function you see it was something like this. It was using canonical function that quite a large, it is quite a large function.

This one we have reduced it to something like this. Now, what will be next step? Next step, we have to find out what are the essential primes because if there are some primes, which are totally includes say if it not an essential prime, that means, it has not have any prime implicants, which not any implicant this is not there in any other. So, it can be

easily dropped. Now, here prime implicants, now to find out, which are the essential prime implicants?

So, essential prime implicants are what there actually we will have term, which is not available in any other prime implicants. So, they obviously, have to be there. So, then we have to find out what is the essential prime implicant, then we will remove some other prime if possible. Then, that will be the most minimal representation for finding what we have to do.

First step we have eliminated all those product terms we said actually not primes. that means, they can be totally included, they totally make SUM of the prime terms. So, that can be directly thrown away, so that is what we have done.

(Refer Slide Time: 01:00:43)

Determining Prime Implicants by Tabular Method

- It may be noted that the function considered in the example was completely specified, i.e., there was no don't care terms.
- If the function is incompletely specified then we follow the same procedure discussed above (for completely specified functions), however, we drop the minterms that contain only don't care minterms.
- Specifically, a product term is a prime implicant of an incompletely specified function if it is a prime implicant of the function and contains at least one minterm which is not don't care.
- Now we will consider the following example, which illustrates tabular method for incompletely specified functions.

NPTEL

So, this was about the, I mean determining prime implicants by tabular methods. So, what was some of the steps you should observe over here? So, you can see that in this case, the example was completely specified. There was no do not care term. So, what I was do not care. So, even if this incompletely specified, that means, some do not care terms are there.

(Refer Slide Time: 01:01:35)

Determining Prime Implicants by Tabular Method

Consider the function written in SOP form $f(x, y, z) = x'z' + xyz + d(xy'z' + xy'z)$.

The function in minterm canonical form is as follows
 $x'y'z' + x'yz' + xyz' + d(xy'z' + xy'z)$.

The table for consensus of the SOP is shown in next Table

$x'y'z'$ (not prime)	$x'z'$ (not prime)	z'
$x'yz'$ (not prime)	$y'z'$ (not prime)	
xyz' (not prime)	yz' (not prime)	
$xy'z'$ (not prime)	xz' (not prime)	
$xy'z$ (not prime)	$xy'z$ (not prime)	
$xy'z'$ (not prime)	$xy'z'$ (not prime)	

$x'z'$ (circled in red)
 z' (boxed in red)
 $xy'z$ (circled in red)
 $xy'z'$ (circled in red)

$x'z'$
 z'
 $xy'z$
 $xy'z'$

So, if the function is specified, so incompletely specified, then what we do so? So, we have to go for some procedure, but the function is incompletely specified. Then, what we do? So, basically you have to make it complete, so how to make it complete? Then, what do you do? You have to put some do not cares. So, now the same procedure will follow. Then, if we find out what is the modification, we have to do? So, if there is a prime implicant, we find out any only comprises of do not care. Then, you can directly throw away that.

So, that is what we are going to just see very small example like if you have a sum of product terms like xz' and $xy'z$, so this there with specified function, so this be told. Then, actually these are that say that do not care terms that is we do not know what is the I mean you already know the concept of do not care. So, it is incompletely specified say if say that for this obviously output is one and for this does not matter. So, how do you know about it?

So, if you just compare little specified completely specified function, so it is bit it is incompletely specified or if there is some do not cares, then how do you know about it? Take this example. So, in this case, so this is the case, this is the do not cares. So, how do you know about that we follow the same procedure like we actually make the canonical representation.

So, in this case, if you can see it is the prime do not care actually this one and this case. We have one $x y z$ prime to explain x prime is broken down into x prime y prime z and x prime $y z$ prime and this is actually directly brought out and this will already in the canonical representation.

So, we say canonical representation in this way again we may take up the tables. So, this 1, 2, 3, 4, 5 terms, so 1, 2, 3, 4, 5 terms are there. So, we are actually one negation here we have two and here we have three could not have any terms like $x y z$ that is not available here. Then, you could have a fourth column, but it is not there.

So, we have only the terms like one complement. So, here two compliment and there are three compliments, to do that, so you paste it over here and then you do that same way. You can try to find out if there is any consensus between them. So, in this case, I do not think consensus because it is $x x y z$ prime there is two terms, which is different consensus, but we can very easily find out that they will be a consensus between this two terms x prime y prime z prime and this $x y z$ prime. So, you can easily delete out this x prime and you can get $y z$ prime.

Similarly, then do about with the whole stuff, and then you can find out which are primes and which are not primes. So, there are same procedures you follow. So, you will find out that all the terms here are included over here. So, they all will become non-prime. So, only two stuffs will be remaining that is the final this one and one extra prime will also remain and that will find that it is not actually included in any other.

Now, question is should I directly if you have taken a completely specified function, then we could have said that $x y$ prime plus z prime or $x y$ prime, z , they both comprise a prime implicants, but it is completely incomplete specified function and there are some do not care. Now, what will be your case? Now, if you see that, what is this $x y$ prime?

So, it is actually, it can be represented that $x y$, $x y$ prime z an $x y$ prime and z prime. So, this is nothing but $x y$ prime and x . So, this is also a term, which is also a prime. So, this is a prime because it is not included in any other because we could not find the consensus for this.

So, this is one prime and this is one prime. So, you should take both, but now you have to do very clear observation that $x y$ prime actually it is a prime term, but it comprises only the do not care. So, do not include it.

So, if you remember out Karnaugh map procedure, so what you do? You include the do not care only. If we give some benefits, if we do not include the do not care, if they cause some problems like if you include on do not care, so number of do not care terms which will include the function size of the function, now let us forget it. So, in this case, there is one prime implicant, which comprises one term which exactly contains nothing but only your do not care. So, if you carry all this one, so you have a large size of the function.

(Refer Slide Time: 01:05:13)

Determining Prime Implicants by Tabular Method

Consider the function written in SOP form $f(x, y, z) = x'z' + xyz + d(xy'z' + xy'z)$.

The function in minterm canonical form is as follows
 $x'y'z' + x'yz' + xyz' + d(xy'z' + xy'z)$.

The table for consensus of the SOP is shown in next Table

$y'z'$ (not prime)	$x'z'$ (not prime)	z'
yz' (not prime)	$y'z$ (not prime)	
$xy'z'$ (not prime)	yz' (not prime)	
xyz' (not prime)	xz' (not prime)	
$xy'z$ (not prime)	$xy'z'$ (not prime)	

Handwritten annotations on the slide include circled terms like z' , $y'z'$, and $xy'z'$, and numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100. There are also handwritten numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100.

So, you will be including one more term comprising do not care. So, directly you can forget it. So, there is one prime implicant only that is z prime, which comprises also there is non do not care terms, so that you have to containing, so in other words, so what is the idea? So, we get one prime implicant. However, do not consider this because it only comprises do not cares, so know that in appropriate specified function. So, completely basic function you go about the same procedure, but one you have to do a small distinction what distinction is there? If you find out any prime implicant, which comprises only do not cares, then it is actually you are carrying dead stuff because that person is not useful. So, we can directly throw away in this one.

So, but if you find out a consensus term like for example, if you are having a consensus like $x y z$ prime like if you are finding a consensus, this one is actually $x y$ prime z and if you are having sorry let us take this two. So, if I have this one like this two like $x y z$ prime and this one $x y$ prime z . So, if you take this sorry if you take this two, you are having $x y$ prime z prime and $x y z$ prime if you take this two.

So, this one will be cut and this one will be cut and you are going to get y prime z prime. So, this is what is obtained come some this one and this one. Now, you see, we cannot throw away this one directly x prime and z prime why? This is because at least comprises one term, which is a , it is not a did not care and one term is a do not care. So, this we cannot throw because it at least contains a one term which is a not a do not care. So, we have to carry this do not care along with this, but this is a term which is having both of them, all the components as do not care. So, you can actually throw out this one. So, with this, we will close this lecture.

So, what we have essentially seen today that given a real circuit, so all the terms can be represented with some Boolean function. Now, you have to minimize represent the minimize value. So, what do you mean by minimal? Minimal means minimum number of literals and minimum number of terms. So, in this case, we have said that we have to first go for a prime representation of the function in terms of prime implicants because prime implicants are the ones, which cannot be included in any other prime. So, obviously, they are actually essential components of your function.

So, we have to find that of for finding the doubt, we are seen one example one procedure saw that the tabular method that will actually give you the prime implicants. Then, prime implicants we have to included in the function because they are actually none of them can be included any other terms.

So, there is what you can call good important components of the function or essential parts of your components of the circuit. Then, we have seen that if we have incomplete is specified function, same procedure you have followed, but you do not have to take some of the prime implicants. They comprise only the do not care term.

So, in the next class, what we will see? So, we will see now we are having the prime implicants, from that, you have to find out that how we can find out the essential prime implicants or among this prime implicants or subset of the prime implicants. So that fully

covers your function and still the number of terms is minimal that we will see, and also we will then go for the multilevel stuff.

Thank you.