

**Digital Circuits**  
**Prof. Santanu Chattopadhyay**  
**Department of Electronics and Electrical Communication Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 43**  
**Memory (Contd.)**

(Refer Slide Time: 00:18)

**Programming the ROM**

In Table, 0 → no connection  
1 → connection

Address 3 = 10110010 is permanent storage using fuse link

X : means connection

1 Programming the ROM According to Tabl 3

So, this programming of this ROM so, suppose we follow this convention that a 0 will be represented by the situation when there is no connection, and one will be represented by the situation when there is a connection. So, this address location 3 so, you see this crosses are there so, that means, these points are connected,. So, they are so, this one is connection so, this if you look into this crosses. So, they are representing the pattern 1 0 1 1 0 0 1 0.

So, when this particular line is selected, since this is a decoder so, at A 1 point of time only one of these 32 locations we will get selected. So, when this if the input pattern applied here is corresponding to the address 3, then this line will get selected, and this line will be since the crosses are here. So, so this is one so, this A 7 will be equal to 1, A 6 will be equal to 0, A 5 will be equal to 1.

So, at the output will we will get the pattern that is stored or at the location 3. So, this way we can program the ROM so that we can. So, for some permanent content there so,

we can the permanent storage. So now, even if the power goes so, this fused this fuse is remaining, so, these contacts remain.

So, as a result you will you will be getting the content as it is when you put the ROM into the circuit, next time so, the value does not change.

(Refer Slide Time: 01:43)

The slide is titled "Combinational circuit implementation". It contains two bullet points: "The internal operation of a ROM can be interpreted in two way: First, a memory unit that contains a fixed pattern of stored words. Second, implements a combinational circuit." and "Previous figure may be considered as a combinational circuit with eight outputs, each being a function of the five input variables." Below the text is the equation  $A_7(I_4, I_3, I_2, I_1, I_0) = \Sigma(0,2,3,\dots,29)$  with "Sum of minterms" written below it. An arrow points from the text "In Table, output A<sub>7</sub>" to the equation. The slide footer includes the IIT KHARAGPUR logo, NPTEL ONLINE CERTIFICATION COURSES logo, and a small video inset of a man.

So, you can use the ROM for realizing some combinational circuit, ok. So, is a internal operation of a ROM can be interpreted in 2 way, first as a memory unit that contains a fixed pattern or stored words that we have already seen. The second is for implementing a combinational circuit.

Let us take an example, suppose we want to we want to store drop in the previous example that I have taken. So, this we can be it can be considered as a combinational circuit with 8 outputs each being a function of 5 input variables. So, for example, here this A 7 is a 5 input function, A 6 is another 5 input function, A 5 is another 5 input function, A 0 is another 5.

So, I can simultaneously realize 8 such 5 input functions by programming the ROM. Why? Because if I apply a particular pattern here, then this A 7 to A 0. So, they will be containing the content of this so, I can ideally visualize it like this that if I 0 to I is I 4. So, they represent some input for the function A 7, and since this A 7 this is equal to 1 so,

this function outputs A 1 here, but for the function A 6. So, this is not the case so, this is equal to 0. So, A 6 will be output will be 0.


So, if I say that this function A 7 is like this. So, this function A 7 so, it has got the mean terms 0, 2, 3 some other mean terms are there and 29 is there suppose this is the A 7.

(Refer Slide Time: 03:35)

**Example**




- Design a combinational circuit using a ROM. The circuit accepts a 3-bit number and generates an output binary number equal to the square of the input number.

Derive truth table first



*Truth Table for Circuit*

Inputs			Outputs					Decimal	
A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	B <sub>5</sub>	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>		B <sub>0</sub>
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

Then we can it can be realized like this. So, if you look back so, this so for this is. So, A 7. So, 0, 2, 3 and 29 so, if you look into this table then you see that 0 for the 0 row. So, this cross is there, one we do not have the cross 2, I have the cross 0, 2, 3 and 29. So, at these points I have got the cross.

So, if this input pattern is such that this rows 0 to 3 or 29 gate selected, then the A 7 output will be equal to 1, if the pattern is such that some other output is getting selected from the decoder, then the output of A 7 will be 0. So, in some sense I can say that this A 7 is realizing the combinational function whose mean terms are 0 to 3 and 29. So, this is what is shown here.

So, let us take another example so, we want to design a combinational circuit the circuit will accept a 3-bit number, and generate an output binary number equal to square of the input number. So, I have got this 3 bit input so, A 0, A 1, A sorry, sorry they should be A 2 this should be A 2 not A 1. So, A 0, A 1, A 2 and I want the square of that.

So, this number is 0 so, square of that is 0. So, basically I am looking for a circuit where I have got these 3 inputs  $A_0 A_1 A_2$ . And it outputs 6 bits so,  $B_0, B_1, B_2, B_3, B_4, B_5$  so,  $B_0$  to  $B_5$ . So, these are the outputs, ok, I am trying to design this circuit such that this circuit it makes the square of these 3-bit number.

Now, this 0 0 0 square of that is 0, 0 0 1 squared is 1, 0 1 0 square is 4 so, this is 4. 0 1 1 that is 3 square is 9. So, 0 0 1, 0 0 1 this way I make this circuit, ok. I have to I have this is this is that functional table that I have. Now the knowledge that we have gathered in our combinational circuit design it says that, you have to design 5 different Karnaugh maps or 6 different Karnaugh maps and do minimizations, and then you have to draw the circuit using logic gates to realize the function.

So, this ROM based realization is somewhat simpler.

(Refer Slide Time: 06:08)

### Example

(a) Block diagram

$A_2$	$A_1$	$A_0$	$B_5$	$B_4$	$B_3$	$B_2$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

(b) ROM truth table

ROM Implementation

Because what we do? We take one 8 by 4 ROM and these address lines we feed  $A_0, A_1, A_2$  and this in this data line so, we look into this thing more carefully, if you look into this part very carefully you see that  $B_1$  is always 0. So, I do not need any logic to realize  $B_1$ . So,  $B_1$  is always 0, and  $B_0$  is always equal to  $A_0$ . So,  $B_0$  and  $A_0$  are same, if  $A_0$  is 0,  $B_0$  is 0. So, this that is no point storing this  $B_0$  and  $B_1$  lines.

So, I can directly take it  $B_0 B_1$  equal to 0, and  $B_0$  equal to  $A_0$ . But for  $B_2, B_3, B_4$  and  $B_5$ , they are dependent on the values of this  $A_0, A_1$  and  $A_2$ . So, what we do ? We

take 1 A so, this is the final truth table that I need to store. So, I need to store the truth table for this B 2, B 3, B 4 and B 5, because I know B 0 is B 1 is equal to 0, and B 0 is always equal to A 0, B 0 is always equal to A 0.

So, from this table so, if we just draw this one so, it says that for 0 0 0, the output should be 0 0 0 0, 0 0 1 it is 0 0 0 0 0 like that so, this is the output. So, what I do? I take one 8 by 4 ROM so, it has got A 0, A 1, A 2 as the output so, the if I take the 8 locations of this ROM.

In the first location, it is a 8 by 4 ROM so, the there are 8 location the first location there are 4 bits so, I stored 0 0 0 0. Second also I store 0 0 0 0, third I stored 0 0 0 1, then 0 0 1 0, then 0 1 0 0, then 0 1 1 0, then 1 0 0 1 and 1 1 1 0.

So, this first bit so, that is this bit is transferred to B 2, similarly the second bit is transfer second bit is transferred to B 3, third bit is transferred to B 4 and this last bit will be transferred to B 5. So, you will be getting this function realize now, suppose A 0, A 1, A 2 comes as say 1 0 0, suppose it gets 1 0 0. So, if it gets 1 0 0, then my pattern is sorry this is slightly wrong. So, this is this is not connected here, rather this bit is connected to B 5, and this bit is connected to B 2 as it is so, you see that B 5 is the most significant bit. So, B 5 is connected so, B if the first bit that I have stored here should be connected to B 5 and this bit should be connected to B 2.

So, if I apply the pattern 1 0 0 then the square of that is so, this is 4 so, that is 16,. So, that is so, that is that is 16. So, 16 is the pattern is given as 1 0 0 0 0, ok. So, if I just look into this table if I look into this table so, this is the so, say 1 0 0. So, 1 0 0 the output is 0 1 0, 0 0 0 so, that should be the output. So, B 4 is one and rest of the thing should be all 0.

So, in this circuit in this case what is what will happen? If I apply 1 0 0 then this B 5 will be equal to 0. So, in this I have stored this pattern 0 1 0 0 here. So, for I have stored the pattern 0 1 0 0 here 0 1 0 0 at that location at location 4.

(Refer Slide Time: 10:32)



### Example

(a) Block diagram

$A_2$	$A_1$	$A_0$	$B_5$	$B_4$	$B_3$	$B_2$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

(b) ROM truth table

ROM Implementation





So, when I give the pattern 1 0 0 here so, this location gets selected, and as a result this 0 comes to B 5, this one goes to B 4, this one goes to B 3 and this one goes to B 2 ok.

So, you will be getting like so, 0 1 0 0 and so, this is always all these are always 0. So, which is basically equal to 16, ok. So, this way you can use some ROM to get the combinational functions realized.

(Refer Slide Time: 11:11)

### Types of ROMs

- The required paths in a ROM may be programmed in four different ways.
  - Mask programming:** fabrication process
  - Read-only memory or PROM:** blown fuse /fuse intact
  - Erasable PROM or EPROM:** placed under a special **ultraviolet light** for a given period of time will **erase the pattern in ROM**.
  - Electrically-erasable PROM (EEPROM):** erased with an **electrical signal** instead of ultraviolet light.



Next will be looking into other types of ROMs so, there are different types of ROMs like the very simplistic type of ROM their known as mask ROM. So, they are at the

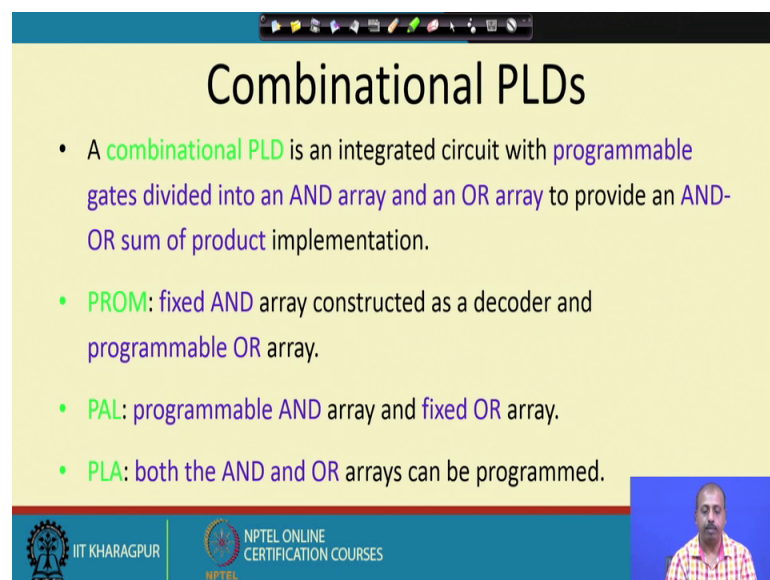
fabrication time itself so, this is fixed then there are some read only memory or PROM programmable read only memory. So, they are so, if what is so, here normally we have got a programmer available is there it is a hardware circuitry.

So, you can put the ROM there in a in a slot, and apply some high voltage accordingly the ROM will get programs. So, so the different locations of the ROM, it will get different values, then there is erasable PROM or EPROM. So, that is so, you can program it by means of applying high voltage, but for you can also erase the content in case of PROM. So, you cannot erase the content so that is one time programmable. Whereas, this erasable PROM or EPROM.

So, you can erase the content also by exposing the chip to the ultraviolet light for some period of time and this electrically erasable PROM or EPROM. So, they can be erased with an electrical signal instead of ultraviolet light.

So, that way so, you can you need not take the chip out of the board. So, if the chip can be there in the circuit board itself, only we apply a high voltage so, the chip content will be erased. Whereas, for EPROM you have to take the chip out and expose it to ultraviolet light separately for erasing it. So, these are the different types of ROMs that we have.

(Refer Slide Time: 12:44)



The slide is titled "Combinational PLDs" and contains the following bullet points:

- A **combinational PLD** is an integrated circuit with **programmable gates** divided into an **AND array** and an **OR array** to provide an **AND-OR sum of product** implementation.
- **PROM**: **fixed AND** array constructed as a decoder and **programmable OR** array.
- **PAL**: **programmable AND** array and **fixed OR** array.
- **PLA**: **both the AND and OR** arrays can be programmed.

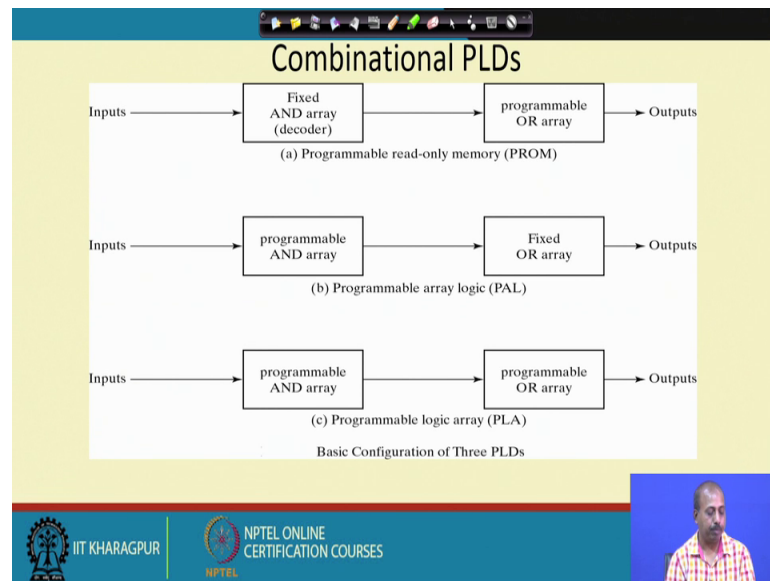
The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a speaker in the bottom right corner.

There are some other programmable logic devices. So, they come under the broad heading of combinational PLD's or programmable logic devices. A PLD is an integrated

circuit with programmable gates divided into an AND array and an OR array to provide and or sum of product type of realizations.

So, in case of PROM we have got fixed AND array constructed as a decoder and a programmable or array. We have got PAL or programmable array logic. So, you programmable AND array and fixed OR array and the PLA where the AND OR both are programmable.

(Refer Slide Time: 13:25)



So, this is the PROM type of PLD that we have, where the AND part is fixed. So, you apply the input so, one of the outputs will get selected. So, that is the fixed and then we have got this programmable OR so, you connect some of these devices.

So, while discussing decoder based combinational circuit design, you have seen this type of structures where we have got this PL name of this decoder. And some of the decoder outputs are odd together. So, for whatever a min terms, the circuit output should be one. So, we have odd those outputs of the decoder to get the programmable output.



So, the same thing is here so, if you are using a decoder and then the AND part becomes fixed and you have got the programmable or part to get the PROM. So, we have got this PLD or PSO this PLA or programmable array logic where the AND part is programmable, but AND OR part is fixed. And we have got this PLA where the both AND plane and OR plane they are programmable.



(Refer Slide Time: 14:37)

### Programmable Logic Array

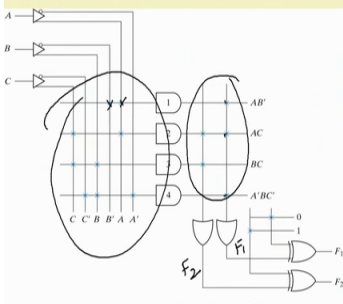
- The decoder in PROM example can be replaced by an array of AND gates that can be programmed to generate any product term of the input variables.
- The product terms are then connected to OR gates to provide the sum of products for the required Boolean functions.
- The output is inverted when the XOR input is connected to 1 (since  $x \oplus 1 = x'$ ). The output doesn't change and connect to 0 (since  $x \oplus 0 = x$ ).



So, we will be looking into this PLA programmable logic array. So, in case of programmable logic array, the decoder from decoder in the PROM example can be replaced by an array of and gates and that can be programmed to generate any term any product term of the input variables. Think there is an example here.



(Refer Slide Time: 14:56)

### PLA

$$F1 = AB' + AC + A'BC'$$
$$F2 = (AC + BC)'$$


PLA with 3 Inputs, 4 Product Terms, and 2 Outputs

Product Term	Inputs			Outputs	
	A	B	C	(T) $F_1$	(C) $F_2$
AB'	1	0	-	1	-
AC	1	-	1	1	1
BC	-	1	1	-	1
A'BC'	0	1	0	1	-



So, suppose I have to get I have to realize these 2 functions. So, F 1 equal to this and F 2 equal to this.

Now, so, also if first we will look into the product lines that I need. So, I need the line  $AB\bar{C}$ , then  $AC\bar{A}B\bar{C}A$ , then again  $ACBC$  and this whole bar so, what is done? So, this  $ABAC\bar{A}BC$  and  $BC$ . So, these are the 4 product terms that I need. So, this is so, this is the AND plane the programmable and plane so, this switches that I have that I shown here.

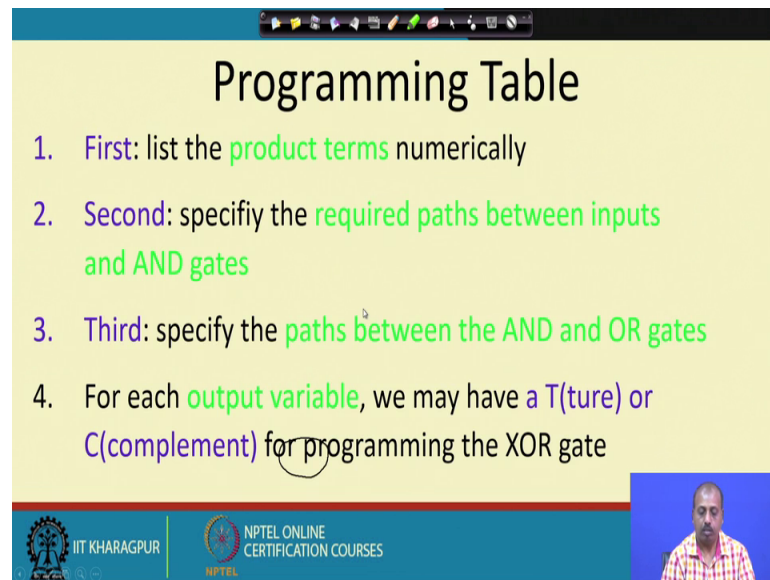
So, they all these points are programmable, out of that so, these connections have been made, as a result on this line you will get  $AB\bar{C}$ , on this line you will get  $AC$ , this line you get  $BC$  and this line you will get  $A\bar{B}C\bar{C}$ , and then that this so that is about the so, this is the and plane, and this side is the or plane. In the OR plane, I am connecting the product terms product lines in such a fashion, that they are or will give me the individual function.

So, this gives me  $F_1$  and this gives me sorry, this gives this gives us  $F_1$  this gives us  $F_2$ . So, this  $ABAC\bar{A}BC$  sorry  $AB\bar{C}AC$  and  $A\bar{B}C\bar{C}$  so, these 3 points so, they are connected to this or gate. So, their connections are fused so, you get this term here. And then there is an XOR gate so that we can do some inversion if needed.

Like see here  $F_2$  you see, that in  $F_2$  we said it should be  $AC$  plus  $BC$  whole bar. So, after getting  $AC$  plus  $BC$  I need to do a complement. And that complementing is done by this additional XOR gate that I have, and here there is a problem there are 2 programmable points or 4 yeah.

So, you can select either the 0 to come to the XOR gate or the 1 to come to the XOR gate. Since for  $F_1$ , we do not need the complemented output direct  $F_1$  is necessary. So, this is connected to 0, for  $F_2$  I need this complemented output so, this is connected to 1 so, you get this  $F_1$  and  $F_2$ , so, in the programmable PLA programming table. So, we first find out what are the product terms, what are the inputs and I could what are the outputs. So, this is known as the PLA programming table.

(Refer Slide Time: 17:21)



## Programming Table

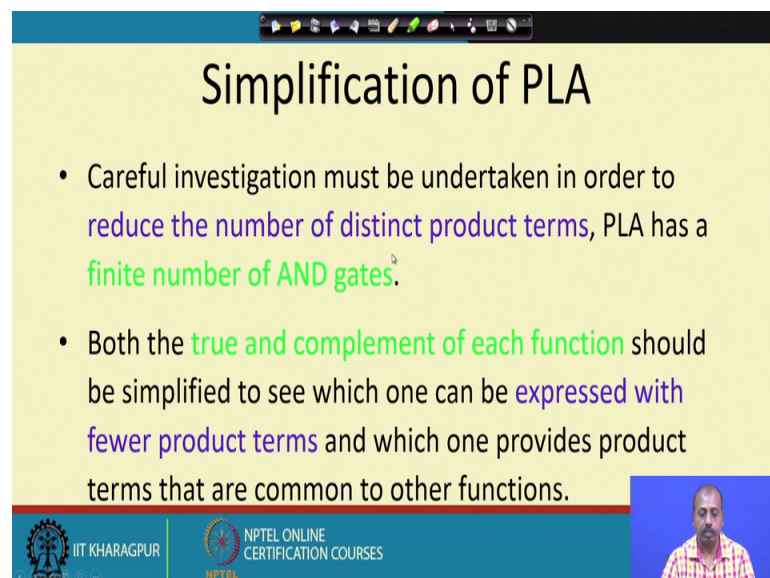
1. **First:** list the **product terms** numerically
2. **Second:** specify the **required paths between inputs and AND gates**
3. **Third:** specify the **paths between the AND and OR gates**
4. For each **output variable**, we may have a **T(ure)** or **C(omplement)** for programming the XOR gate

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And once this is done, we can we can go for the actual realization. So, for this programming table formulation, first we list down the product terms numerically, then we specify the required paths between inputs AND gates. Then in the third space we specify the paths between and gates and or gates.

And then for each output variable we may have a true output or a complemented output for the programming the XOR gate. So, that is how this is done for the programming table part.

(Refer Slide Time: 17:53)



## Simplification of PLA

- Careful investigation must be undertaken in order to **reduce the number of distinct product terms**, PLA has a **finite number of AND gates**.
- Both the **true and complement of each function** should be simplified to see which one can be **expressed with fewer product terms** and which one provides product terms that are common to other functions.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Sometimes it is better that we do a more careful investigation to see whether I can reduce the PLA size further. So, it may so happen that if I compliment one of the output then there will be there may be more commonality between the product terms of different functions. Since the AND plain is realizing all the product terms so, if I can get a situation where this product terms are less so, I will be getting a better realization.

(Refer Slide Time: 17:25)

**Example**

Implement the following two Boolean functions with a PLA:

$$F_1(A, B, C) = \sum(0, 1, 2, 4)$$

$$F_2(A, B, C) = \sum(0, 5, 6, 7)$$

The two functions are simplified in the maps shown.

		B			
		00	01	11	10
A	0	1	1	0	1
	1	1	0	0	0



		B			
		00	01	11	10
A	0	1	0	0	0
	1	0	1	1	1

1 elements  $F_1 = A'B' + A'C' + B'C'$

0 elements  $F_1 = \overline{AB + AC + BC}$

$F_2 = \overline{AB + AC} + A'B'C'$

$F_2 = (A'C + A'B + AB'C)'$

So, this, what is done? Is, it is like this suppose I have to realize these 2 function, F 1 ABC which is which has got the mean terms 0, 1, 2, 4 in it and F 2 has got ABC which is 0, 5, 6, 7. The 2 functions are simplified in the maps as shown here, now while doing the simplification, we try for both F 1 and F 1 dash, ok.

So, if our F 1 so, if you if you would group with A 0, A 1 element so, you will get A bar B bar like if you if you combine these 2. So, you will get A bar B bar, then this A bar C bar. So, combining this one and this one you will get A bar C bar, and you will get B bar C bar by combining these 2.

On the other hand, if you group in terms of 0's so, which you are getting AB like say here, this is a equal to 1 and B equal to 1. So, these 2 term you can get. So, if you are so, this AB AB will be coming like so, this is basically the complement of this. So, if you complement it, then this function will come so, this is A 0 elements are taken into consideration then you will be you will be getting this like here AB plus AC plus BC whole bar, that is the function we will get.

And similarly here for F 2 for F 2 also you will get AB plus AC, and if you are doing it taking F 2 bar that is then you will be getting the other function A bar C plus A bar B plus AB bar C bar whole bar. Now what we do is that we compare all these functions and try to figure out the common terms. And we see that if I choose this complemented form of F 1, and true form of F 2 then this AB and AC so, these terms are becoming common.

So, I will so, they were as these terms will become common so, I can just take them together and get it realized. So, the less number of product terms will be sufficient for realizing this thing.

(Refer Slide Time: 20:31)

PLA table by simplifying the function




- Both the **true** and **complement** of the functions are simplified in **sum of products**.
- We can find the same terms from the group terms of the functions of  $F_1, F_1', F_2$  and  $F_2'$  which will make the minimum terms.

$$F_1 = (AB + AC + BC)'$$

$$F_2 = AB + AC + A'B'C'$$

PLA programming table

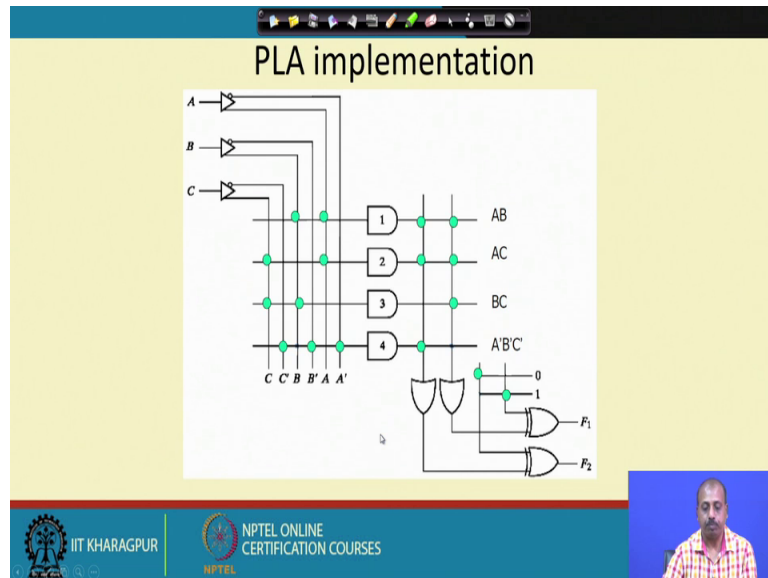
Product term	Inputs			Outputs	
	A	B	C	(C) $F_1$	(T) $F_2$
$AB$	1	1	-	1	1
$AC$	2	1	-	1	1
$BC$	3	-	1	1	-
$A'B'C'$	4	0	0	-	1

So, we do it that and the both true and compliment of the functions are simplified in sum of product form, and we can find the same terms from the group from the group terms of the functions  $F_1, F_1', F_2$  and  $F_2'$  which will make the minimum number of terms.

So, we find that F 1 should be taken in complemented form and F 2 it should be taken in true form to get the minimum sized minimum number of product terms. So, that is done here.

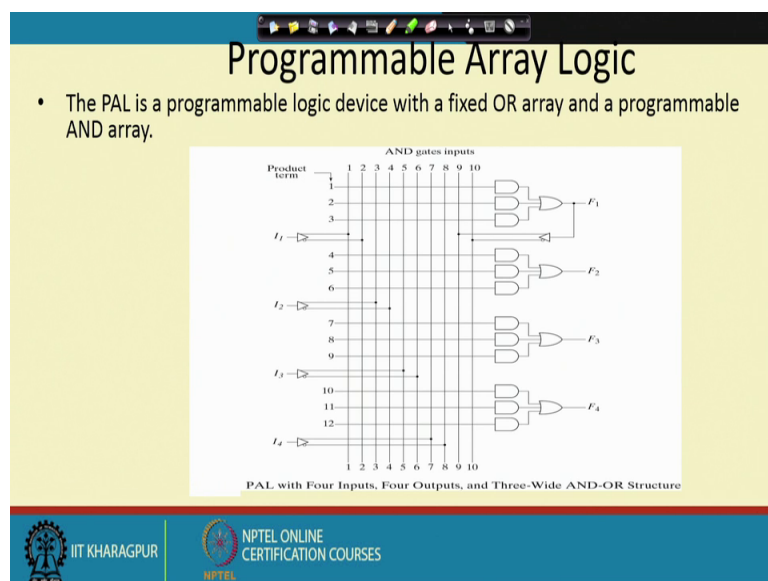
(Refer Slide Time: 20:59)



And final PLA implementation turns out to be like this; that we take this terms so, that this is simple. So, this term realizes the product term AB. So, this A and B so, those 2 lines are switches are bond, so, this AB comes here.

Similarly, AC comes here, BC comes here, and then after doing the XOR after doing the OR plain here, then we select this x or programming point so that for F 1 I get F 1 bar and F 2 comes directly. So, this way we can have PLA implementation of this circuit.

(Refer Slide Time: 21:34)



So, then we will this programmable array logic so, this is another programmable device, where we have got this. And plane fixed so, this and whether sorry, the OR plain is fixed. And the after and plane is programmable so, then the in the OR plain we have we can you can get or of these 3 and gates.

So, so, this so, you cannot choose any other one whereas, while drawing this and gates so, you can program all these points to select the inputs that should come for the f for the first and gate. So, this OR gate is fixed so, this is the or of these 3 inputs, these 3 and gates. Similarly, second OR gate is the is the or of these 3. Now you can select so this first 2 lines they correspond to  $I_1$  and  $I_1$  bar. Second 2 lines correspond to  $I_2$  and  $I_2$  bar.

So, sometimes we take this  $F_1$  back, and this  $F_1$  and  $F_1$  bar that is 2 additional inputs that comes back here. So, that is sometimes helpful, if there is a sharing of function between  $F_1$  and something else, then you can pass it through these lines and use it on the on the lines 9 and 10 and that can be used for AND gate connection so, we will see that.

So, essentially in the in the PAL device so, this or part is fixed and the and part is programmable.

(Refer Slide Time: 23:00)

**PAL**

- When designing with a PAL, the Boolean functions must be simplified to fit into each section.
- Unlike the PLA, a product term cannot be shared among two or more OR gates. Therefore, each function can be simplified by itself without regard to common product terms.
- The output terminals are sometimes driven by three-state buffers or inverters.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, Boolean functions must be simplified to fit into each section. So, you see that if you in a case so, if you if you need  $F_1$  is as a function which requires more than 3 product

terms, then I cannot fit it. Similarly, if F 2 requires more than 3 product terms, I cannot fit it standalone.

Of course, what you can do possibly if you is that in a for realizing one function if you require more than more product terms, then you can break it down into 3 product term cases and then additionally use some OR gate at the output of those 2 and or them together. But that is costly that is some additional gate will be necessary.

So, that way it is restrictive unlike PAL product term cannot be shared among 2 or more or gates. Therefore, each function can be simplified by itself without regard to common product terms. So, this is this form from the simplification point of view. So, this is easy so we do not need to compare between the product into between the functions to see whether there is a commonality of product terms between the functions. So, we can simply take individual functions separately, and we can minimize them so that they go to one of the alternative.

And the output terminals are sometimes driven by 3 state buffers or inverter so, that is also done. So, that we can get the inverted output.

(Refer Slide Time: 24:24)

**Example**

$$w(A, B, C, D) = \sum(2, 12, 13)$$
$$x(A, B, C, D) = \sum(7, 8, 9, 10, 11, 12, 13, 14, 15)$$
$$y(A, B, C, D) = \sum(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$
$$z(A, B, C, D) = \sum(1, 2, 8, 12, 13)$$

Simplifying the four functions as following Boolean functions:

$$w = ABC' + A'B'CD'$$
$$x = A + BCD$$
$$y = A'B + CD + B'D'$$
$$z = ABC' + A'B'CD' + AC'D' + A'B'C'D = w + AC'D' + A'B'C'D$$

The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a man speaking.

So, let us take an example suppose w ABCD is this one so, w x y z so, these are 4 functions. So, we try to simplify the 4 functions like this w equal to ABC bar plus A bar B bar C D bar. X equal to a plus BCD, y equal to this and z equal to this. Then after



doing this, we find that first 2 terms of z are same as w. So, these we rewrite it as w plus  $\overline{A}\overline{C}\overline{D}$  plus  $\overline{A}\overline{B}\overline{C}\overline{D}$ .

So, after doing this rewriting now a so now, we are satisfied because individual individual functions they are having at most 3 product terms, ok. So, previous so, the this particular expansion of z so, that was problematic, because it was requiring 4 product terms. So, but here it is requiring 3 product terms only. So, if it was rickys if some line is requiring more than 3 product terms, then we can take a dummy output, and the dummy output can be fed as input for the next one so, I will come to that.



(Refer Slide Time: 25:31)

### PAL Table

- z has four product terms, and we can replace by w with two product terms, this will reduce the number of terms for z from four to three.

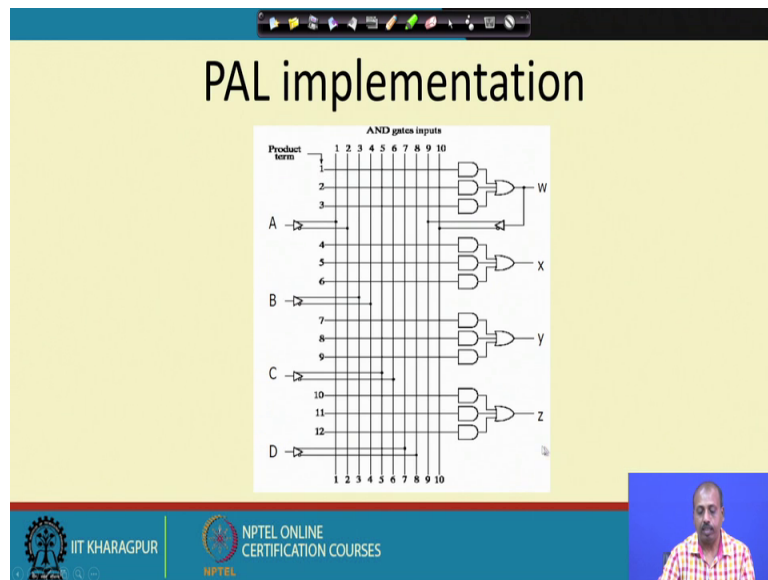
*PAL Programming Table*

Product Term	AND Inputs					Outputs
	A	B	C	D	W	
1	1	1	0	-	-	$w = ABC'$ $+ A'B'CD'$
2	0	0	1	0	-	
3	-	-	-	-	-	
4	1	-	-	-	-	$x = A$ $+ BCD$
5	-	1	1	1	-	
6	-	-	-	-	-	
7	0	1	-	-	-	$y = A'B$ $+ CD$ $+ B'D'$
8	-	-	1	1	-	
9	-	0	-	0	-	
10	-	-	-	-	1	$z = w$ $+ AC'D'$ $+ \dots$
11	1	-	0	0	-	
12	0	0	0	1	-	

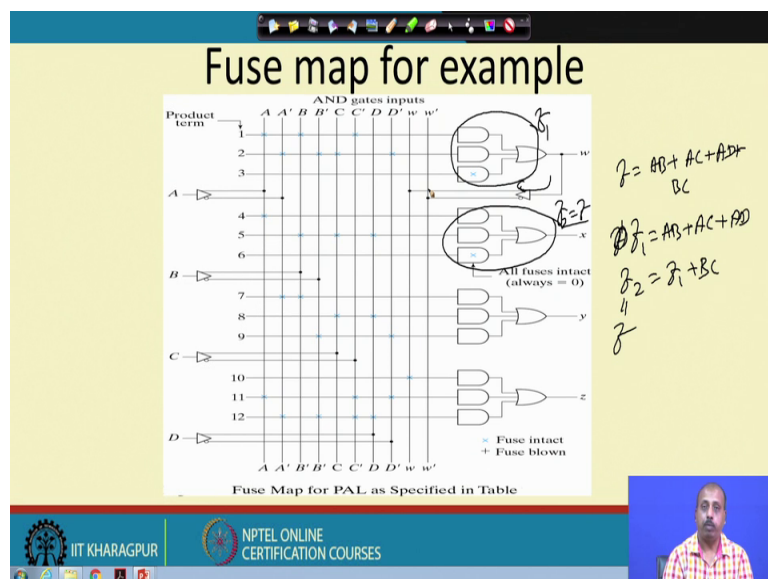
First of all, so, this realization so, z has 4 product terms and we can replace by w so, here like this. So, this way we can make it simplified.

(Refer Slide Time: 25:41)



And then the realization becomes I so, w line is connected back. And it is connected to this this 9 and 10 lines they carry w and w bar lines, and they can be used for connecting to z.

(Refer Slide Time: 25:56)



So, this way we can have a PLA implementation. So, this fuse map that tells like how are you going to connect. So, this product line so, this is AB and C bar. So, that is connected to this and gate. Similarly, this A bar B bar C so, that is connected to this and gate, and

this AND gate is not used at all. So, this is it had only this function had only 2 product lines. So, this AND gate is not necessary so, this is not program.

So, all these connections are opened. Then for the second one again the same thing I have got connections where third AND gate is not necessary, all fuses are intact. So, it is always equal to 0. So, this way we can do the connection and the w output so, it is fed back to this w and w dashed lines. And this w and w dash, this w line is used here by the z output as another and input to this or gate. So, that way we can do this realization.

Now, if there are if you get some function which is the more having more number of product terms, then what you can do is you can take a dummy output, like suppose I have got a function f consisting of the product terms like AB plus AC plus say AD plus BC, say something like this. Then what you can do ? You can have A 1 such you can take a dummy output. So, f dash, let us call it f 1. So, f 1 is say AB plus AC plus AD, and then f 2 is f 1 plus BC.

So, you can break it down like this and use this type of feedback connection for realizing this. So, for this first by this first set you realize f 1, and this f 1 is fed back here and then by the second set you realize f 2, that is equal that is that is equal to f. So, f 2 is equal to f and this f 1 plus BC is realized by that. So, this way you can utilize this back connections.

So, that you can have functions with larger number of product terms realized, but of course, it is restrictive in the sense that the device that you are using may not have a large number of such connection back connections available. So, you may not be able to realize very large functions with this type of logic.