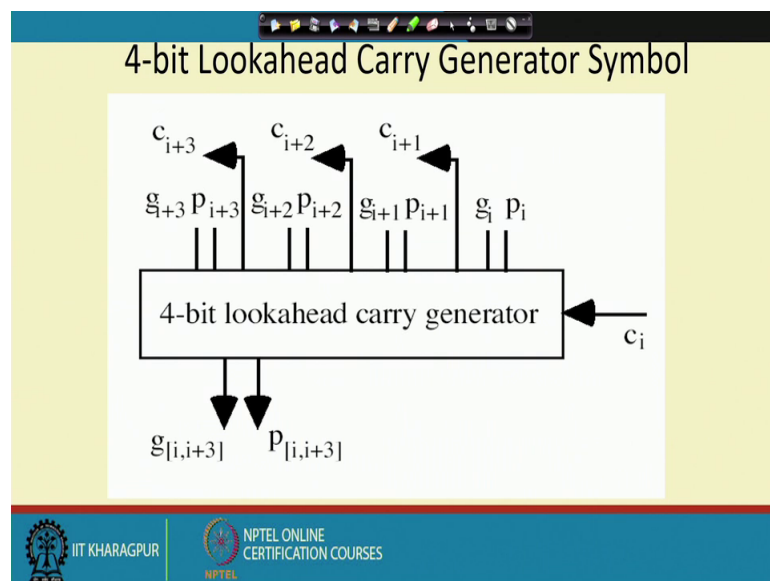


Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 21
Arithmetic Circuits (Contd.)

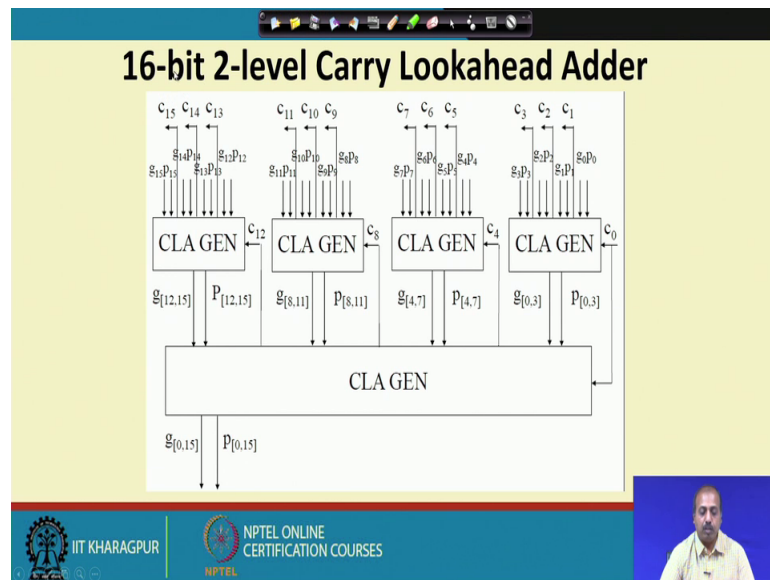
So, this look ahead carry generation part this portion; so, we can make it a design which is again an iterative design.

(Refer Slide Time: 00:18)



Suppose I have got this 4-bit look ahead carry generator circuitry; so, that has got c_i as input and it generates of course, this a_i and b_i those x_i and y_i those bits are there. So, it is not shown here explicitly. So, this g_i and p_i ; so, they are coming here in terms of that x_i and y_i . So, that part is not shown here and it generates this c_{i+1} c_{i+2} c_{i+3} and it generates this a_{p_i} plus $i+2$ plus 3 and g_{i+2} plus 3 ; so, those portions.

(Refer Slide Time: 01:00)



Now, if you have got that this carry look ahead generator 4-bit versions. So, now, you can use it for 16 bit carry look ahead circuitry; so, we have got this 4 such carry look ahead blocks connected. And these carry look ahead blocks; so, they are getting say this say this say $g_0, p_0, g_1, p_1, g_2, p_2$ etcetera and it is generating 2 generate one generate signal and one propagate signal. So, this generate signal will be will be fed to this CLA generator circuit.

So, this is actually this block is similar to this block; so, whatever be the number of the inputs here similar number of inputs are there in this CLA generator block also and we are just using some sort of hierarchical structure ok. So, this is using the same CLA generator block, but it is this of this g inputs are coming from this the corresponding CLA generator block of the previous level of hierarchy.

And it is generating g_0 to 15 and p_0 to 15 that is the if you take a 16 bit block then this will be the generate signal from the 15 bit block and this will be the propagate signal from the 16 bit block and this is the propagate signal from the 16 bit block.

(Refer Slide Time: 02:27)

Operation of 16-bit 2-level Carry Lookahead Adder		
Signals computed	Formulas	Delay
g_i, p_i $i=0..15$	$g_i = X_i Y_i$ $p_i = X_i \oplus Y_i$	1 gate delay
$g_{[i..i+3]}, p_{[i..i+3]}$ $i=0, 4, 8, 12$	$g_{[i..i+3]} = g_{i+3} + g_{i+2} p_{i+3} + g_{i+1} p_{i+2} p_{i+3} + g_i p_{i+1} p_{i+2} p_{i+3}$ $p_{[i..i+3]} = p_i p_{i+1} p_{i+2} p_{i+3}$	2 gate delays

So, you can cascade another such block here ah; so, to get a 32 bit block ok; so, that way we can go ahead. So, we can we will looking to an example; so, these are simple formulas that are we have got this delay calculations and all like this $g_i p_i$. So, the formula is g_i equal to $x_i y_i$; so, that is that will 1 1 gate delay and this p_i computation is $x_i x$ or y_i ; so, that is also 1 gate delay.

Now, this g_{i+2} plus 3 and p_{i+2} plus 3 they follow this particular formula and here if I am if I am assuming to 2 level realization of the circuit; then that there is one set of and gates that will be realizing this g_i plus into p_i plus 3 similarly g_i plus 1 into and p_i plus 2 and p_i plus 3. So, this way it will be realizing the AND terms and after that there will be an OR term. So, there will be 2 gate delay in the generate g_{i+2} plus 3 and p_{i+2} plus 3.

(Refer Slide Time: 03:21)

16-bit 2-level Carry Lookahead Adder

Signals computed	Formulas	Delay
c_4, c_8, c_{12} $g_{[0..15]}, P_{[0..15]}$		2 gate delays
	$c_4 = g_{[0..3]} + c_0 P_{[0..3]}$ $c_8 = g_{[4..7]} + g_{[0..3]} P_{[4..7]} + c_0 P_{[0..3]} P_{[4..7]}$	
	$c_{12} = g_{[8..11]} + g_{[4..7]} P_{[8..11]} + g_{[0..3]} P_{[4..7]} P_{[8..11]} + c_0 P_{[0..3]} P_{[4..7]} P_{[8..11]}$	
	$g_{[0..15]} = g_{[12..15]} + g_{[8..11]} P_{[12..15]} + g_{[4..7]} P_{[8..11]} P_{[12..15]} + g_{[0..3]} P_{[4..7]} P_{[8..11]} P_{[12..15]}$	
	$P_{[0..15]} = P_{[0..3]} P_{[4..7]} P_{[8..11]} P_{[12..15]}$	

So, for this c_4, c_8 and c_{12} ; so we will have 2 gate delays for g_0 to g_{15} and p_0 to p_{15} that will also have 2 gate delays.

(Refer Slide Time: 03:33)

16-bit 2-level Carry Lookahead Adder

Signals computed	Formulas	Delay
$c_{i+1}, c_{i+2}, c_{i+3}$ $i = 4, 8, 12$		2 gate delays
<i>i.e.,</i> $c_5, c_6, c_7, c_9, c_{10}, c_{11}, c_{13}, c_{14}, c_{15}$		
	$c_{i+3} = g_{i+2} + g_{i+1} P_{i+2} + g_i P_{i+1} P_{i+2} + c_i P_i P_{i+1} P_{i+2}$	
	$c_{i+2} = g_{i+1} + g_i P_{i+1} + c_i P_i P_{i+1}$	
	$c_{i+1} = g_i + c_i P_i$	

Now for this $c_{i+1}, c_{i+2}, c_{i+3}$ for $i = 4, 8, 12$; they will see 2 gate delays; so, because of this expression if you looking into this expression; so you will see how many gate delays are necessary.

(Refer Slide Time: 03:47)

Signals computed	Formulas	Delay
$S_{i+1}, S_{i+2}, S_{i+3}$ $i = 4, 8, 12$ <i>i.e.,</i> $S_5, S_6, S_7, S_9, S_{10}, S_{11}, S_{13}, S_{14}, S_{15}$	$S_i = p_i \oplus c_i$	1 gate delay

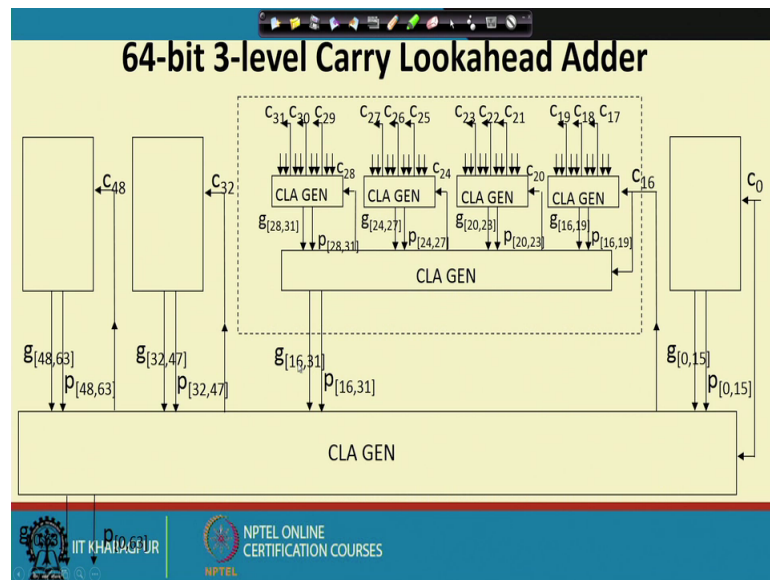
Total: 8 gate levels in the CLA adder vs. 32 gate levels in the ripple carry adder

So, this way it can total 8 gate levels in this carry look ahead adders versus 32 gate levels in the ripple carry adders. So, if I am if I am had having a 16 bit adder then there will be 32 gate level in the ripple carry adder because this individual stage it will have that sum and carry. So, that will be the carry come calculation will have 2 stages ok. So, that way it will have total 32 stage, but this if you are using this carry look ahead mechanism; so, that will have only 8 stage.

That is why this carry look ahead adder; so they are going to be much faster than this ripple carry adder and in many application; so where we need fast addition, subtraction. So, we go for this carry look ahead adder instead our ripple carry.

Of course the penalty that we pay is the number of gates that are needed becomes very high. So, this is the thing that I was talking about.

(Refer Slide Time: 04:45)



So, this is suppose I have called this 16 bit carry look ahead blocks ok. Now from there; so, I can I can have some 32 bit block and from there I can have some 64 bit block. So, here it is 64 bit block that is generated that is shown here in terms of this cascading of this clock generator circuit.

(Refer Slide Time: 05:00)

Level	Signals computed	Delay
PRE	g_i, P_i $i=0..63$	1 gate delay
1	$g_{[i..i+3]}, P_{[i..i+3]}$ $i=0, 4, 8, 12, \dots, 56, 60$	2 gate delays
2	$g_{[i..i+15]}, P_{[i..i+15]}$ $i=0, 16, 32, 48$	2 gate delays
3	$g_{[0..63]}, P_{[0..63]}$ C_{16}, C_{32}, C_{48}	2 gate delays
2	$C_{20}, C_{24}, C_{28}, C_{36}, C_{40}, C_{44}, C_{52}, C_{56}, C_{60}$	2 gate delays
1	$C_{21}, C_{22}, C_{23}, C_{25}, C_{26}, C_{27}, \dots, C_{61}, C_{62}, C_{63}$	2 gate delays
POST	$S_{21}, S_{22}, S_{23}, S_{25}, S_{26}, S_{27}, \dots, S_{61}, S_{62}, S_{63}$	1 gate delay



And these are different gate delays calculation that we have; so it is similar to the previous calculation.

(Refer Slide Time: 05:06)

Delay of a k-bit Carry-Lookahead Adder

$$T_{\text{lookahead-adder}} = 4 \lceil \log_4 k \rceil$$

k	$T_{\text{lookahead-adder}}$	$T_{\text{ripple-carry-adder}}$
4	4	8
16	8	32
32	12	64
64	12	128
128	16	256
256	16	512



So, in that in summary you can say that a look ahead adder; so, now total it has a carry bit k bit carry look ahead then the total look ahead adder delay is 4 into log of k. So, that gives the advantage like k equal to say if I have got the adder size has a 32, then look ahead adder delay will be 12 stages and this ripple carry adder will have 64 stages. So, that ways for 256 bit add addition; so, look ahead adder will have got a delay of 16 stages and carry look ahead ripple carry will have 512 stages; so that is pretty high.



(Refer Slide Time: 05:46)

Decimal Adders

8421 weighted coding scheme or BCD Code

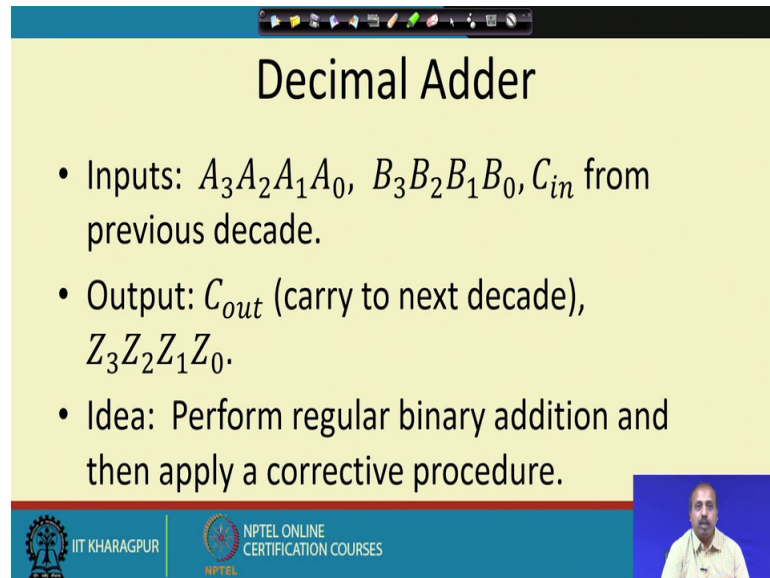
Decimal Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Forbidden codes: 1010, 1011, 1100, 1101, 1110, 1111



Next will be looking into decimal adders; so, decimal adder means that I have got the binary coded decimal digits BCD and then I want to add them. So, in case here; so here I have got the digits important for 0 to 9 only; so rest are not important. So, they are coded like this; so, 9 is coded as 1001 and rest of that 4-bit combinations are not valid for the BCD addition; so, these are the forbidden codes 1010, 1011 and all.

(Refer Slide Time: 06:19)



The slide is titled "Decimal Adder" and contains the following information:

- Inputs: $A_3A_2A_1A_0$, $B_3B_2B_1B_0$, C_{in} from previous decade.
- Output: C_{out} (carry to next decade), $Z_3Z_2Z_1Z_0$.
- Idea: Perform regular binary addition and then apply a corrective procedure.

At the bottom of the slide, there are logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with a small video inset of a man speaking.

So, for decimal adder; so, we have got the inputs A_3 , A_2 , A_1 , A_0 , B_3 , B_2 , B_1 , B_0 and C_{in} from the previous cascade state. And output was C_{out} and Z_3 , Z_2 , Z_1 , Z_0 and so what we do for decimal adder is that we perform regular binary addition and then we do some corrective procedure.

(Refer Slide Time: 06:40)

Comparing Binary and BCD Sums

Decimal Sum	K	P ₃	P ₂	P ₁	P ₀	Same C _{out}	Z ₃	Z ₂	Z ₁	Z ₀
0-9										
10	0	1	0	1	0	1	0	0	0	0
11	0	1	0	1	1	1	0	0	0	1
12	0	1	1	0	0	1	0	0	1	0
13	0	1	1	0	1	1	0	0	1	1
14	0	1	1	1	0	1	0	1	0	0
15	0	1	1	1	1	1	0	1	0	1
16	1	0	0	0	0	1	0	1	1	0
17	1	0	0	0	1	1	0	1	1	1

C_{out} is set to 0

So, for example here we have got this number 0 to 9 for this part it is same ok. So, actually this likely shifted; so, this is for 0 to 9 if the decimal sum is equal to 0 to 9 then it is same; however, if the decimal sum is 10; 1010. So, it will be represented in the BCD forms; so they will be they will be represented has carry bit being 1 and the Z₃, Z₂, Z₁, Z₀ being 0.

Similarly, for 11 if the decimal sum is 11 then the C out will be equal to 1 and the Z₀ will be equal to one and the rest of them will remain as 0 12 will be like that. So, basically this C out bit will be set to 1 whenever the sum becomes greater than 9 for the BCD addition ok. So, this way I can ah; so, C out is 0 for this part and for other part; so, it will be equal to 1.

(Refer Slide Time: 07:36)

Decimal Adder

- No correction needed when the decimal sum is between 0-9.
- Must apply a correction when the sum is between 10-19.
- Case 1:
 - 16-19: K is set to 1. Add binary quantity 0110 to $P_3P_2P_1P_0$.
 - 10-15: $KP_3P_2P_1P_0$ are set to 01010, 01011, ..., 01111. Need to add 6. Use a K-map to obtain a Boolean expression to detect these six binary combinations.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, no correction is needed when the decimal sum is between 0 and 9 and we must apply a correction, when the sum is between 10 and 19. So, at most we can add 9 and 9; so that is at most the value can be 18. So, this one a value the between 10 and 19; so, you have to do some correction.

(Refer Slide Time: 08:00)

Rules of BCD adder

- When the binary sum is greater than 1001, we obtain a non-valid BCD representation.
- The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.
- To distinguish them from binary 1000 and 1001, which also have a 1 in position Z_8 , we specify further that either Z_4 or Z_2 must have a 1.

$$C = K + Z_8Z_4 + Z_8Z_2$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this; so, these are the rules for BCD addition. So, when the binary sum is greater than 1 0 0 1 we obtain a non valid BCD representation. And the binary the addition of binary 6 to the binary sum; it will convert the number to the correct BCD form.

So, whenever it becomes more than 1001; so, you just add this 6 to the binary to the whatever sum you have got and that gives the corrective action. So, to distinguish between 1000; that is binary 1001 and 1001; so, which will also have one in position Z₈. So, we specify that either Z₄ and Z₂ must have a value 1. So, C is equal to K plus Z₈, Z₄ plus Z₈ Z₂.

(Refer Slide Time: 08:46)

Implementation of BCD adder

- A decimal parallel adder that adds n decimal digits needs n BCD adder stages.
- The output carry from one stage must be connected to the input carry of the next higher-order stage.

Block Diagram of a BCD Adder

So, this is the BCD sum part; so, this is 0100; so, this will be added if this Z₈, so, that if Z₈ K 4.

(Refer Slide Time: 09:00)

Rules of BCD adder

- When the binary sum is greater than 1001, we obtain a non-valid BCD representation.
- The addition of binary 6(0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.
- To distinguish them from binary 1000 and 1001, which also have a 1 in position Z₈, we specify further that either Z₄ or Z₂ must have a 1.

$$C = K + Z_8 Z_4 + Z_8 Z_2$$

So, if you look into this expression is K plus $Z_8 Z_4$ plus $Z_8 Z_2$. So, if this is situation then only the carry is generated.

(Refer Slide Time: 09:07)

Implementation of BCD adder

- A decimal parallel adder that adds n decimal digits needs n BCD adder stages.
- The output carry from one stage must be connected to the input carry of the next higher-order stage.

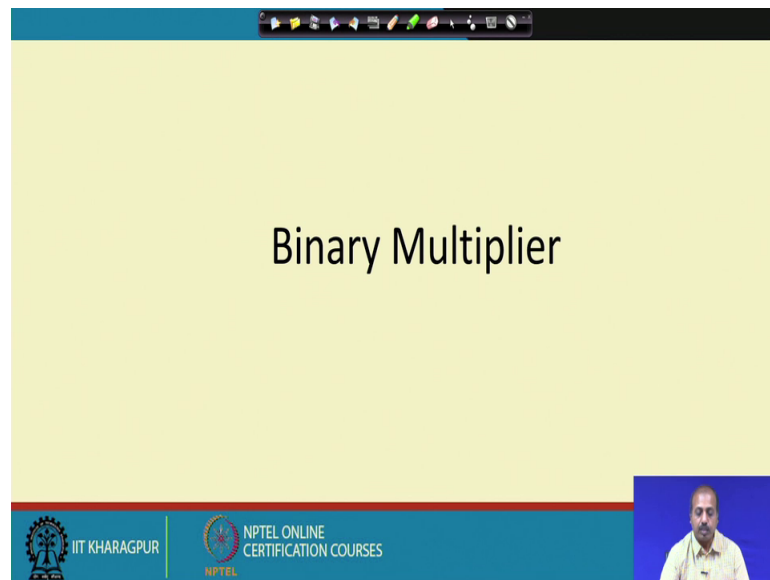
Block Diagram of a BCD Adder

So, what we are doing? For with this 4-bit addition result; so, you are trying to add this 6, but that is conditional conditional subject to the case that there was a carry or this Z_8, Z_4 was equal to 1 or $Z_8 Z_2$ was equal to 1.

So, if you draw the corresponding truth table then will find that under these conditions; so, we need to do the correction of adding 6 to the binary addition result to get the BCD addition. So, a decimal parallel adder that adds n decimal digits needs a n bit; n BCD adder stages. And the output carry from one stage must be connected to the input carry of the next higher stage; so, that is obvious.

So, if you need more number of decimal digits to be added. So, then this entire block has to be repeated one after the other to get the BCD addition for higher number of digits.

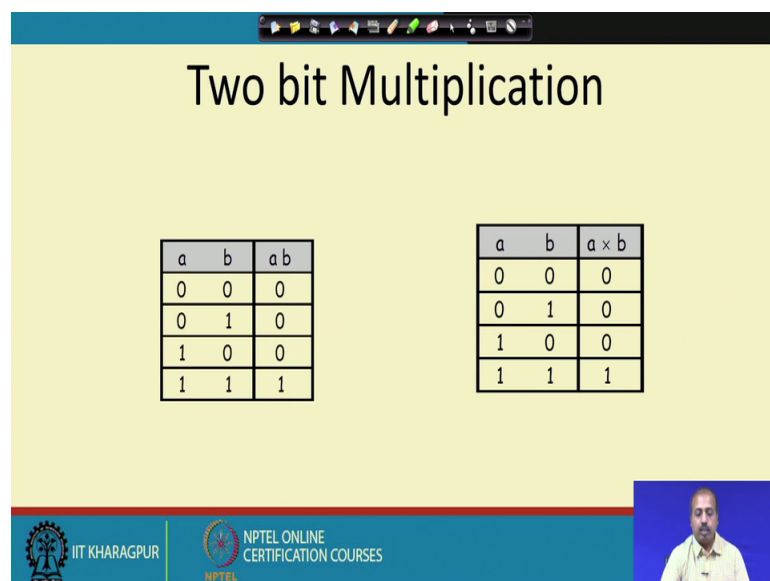
(Refer Slide Time: 10:00)



Binary Multiplier

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

(Refer Slide Time: 10:08)



Two bit Multiplication

a	b	a b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a × b
0	0	0
0	1	0
1	0	0
1	1	1

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES


Next we look into the multiplier; now you see that if I have got 2 bits a and b has input then the multiplication is like this a b a and b it is 0 1 0 0 1. So, if you compare with the AND operation you see that you are getting the same thing. So, this AND is some sort of 2 bit binary multiplication; so, this 2 bit binary multiplication is same has the AND of the 2 bits.

(Refer Slide Time: 10:34)


Three bit Multiplication

- Partial products are: 101×1 , 101×1 , and 101×0
- Note that the partial product summation for n digit, base 2 numbers requires adding up to n digits (with carries) in a column.
- Note also $n \times m$ digit generates up to an $m + n$ digit result (same as decimal).


$$\begin{array}{r} 101 \\ \times 011 \\ \hline 101 \\ 1010 \\ 0000 \\ \hline 001101 \end{array}$$




IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



NPTEL



For 3 bit multiplication; so suppose; so we have will generate this partial product terms, so 101101 ; so, this 101 multiplied by 1 . So, that generates this partial term 101 then this 101 multiplied by the second one that generates this partial sum and then multiplied by 0 that generates this partial sum and then we have then we add all this partial sum to get the result.




So, we have the partial products and this partial product summation for n digit base 2 numbers requires adding up the up to n digits in a column. So, that way we are going to add these digits. So, that if I am multiplying and n by one n bit number by an m bit number, then we have to generate it will generate and m plus n digit number ok.

(Refer Slide Time: 11:25)

Multiplier Boolean Equations

- We can also make an $n \times m$ "block" multiplier and use that to form partial products.
- Example: 2×2 – The logic equations for each partial-product binary digit are shown below:
- We need to "add" the columns to get the product bits P_0 , P_1 , P_2 , and P_3 .
- Note that some columns may generate carries.

		b_1	b_0
	\times	a_1	a_0
		—————	—————
		$(a_0 \cdot b_1)$	$(a_0 \cdot b_0)$
$+$		$(a_1 \cdot b_1)$	$(a_1 \cdot b_0)$
	—————	—————	—————
	P_3	P_2	P_1
			P_0

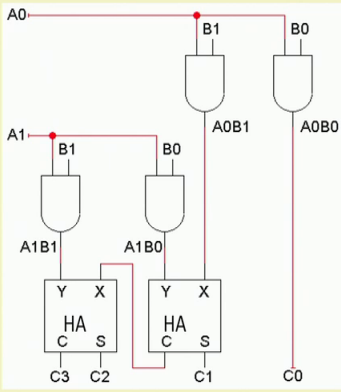
So, that is they of they are from multiplication. So, Boolean equation for multiplication; so, it will be like this. So, if I am taking as a 2 bit multiplication for example, then this generates a 0 b 0 and a 0, b 1 as the this partial sum. And then this a 1, b 0 and a 1, B 1 as another partial sum then this partial sums are added the P_0 , P_1 , P_2 , P_3 that generates the product terms ok.



(Refer Slide Time: 11:52)

A 2x2 binary multiplier

- The AND gates produce the partial products.
- For a 2-bit by 2-bit multiplier, we can just use two half adders to sum the partial products. In general, though, we'll need full adders.
- Here C_3 - C_0 are the product, not carries!

		B_1	B_0
	\times	A_1	A_0
		—————	—————
		A_0B_1	A_0B_0
$+$		A_1B_1	A_1B_0
	—————	—————	—————
	C_3	C_2	C_1
			C_0



So, this is the circuitry for that; so, this generates A_0, B_0 ; then this generates A_0, B_1 A_1, B_0 and A_1, B_1 and then this A_0, B_0 directly gives; it is not added with anything.

So, that that come directly comes at the multiplication the result output C 0; then this A 0, B 1 and A 1 B 0 are to be added.

So, I can use a half adder here because there is no carry. So, I can just use a half adder to do that and then from this some carry may be generated and that carry has to be added with A 1, B 1. So, this A 1, B 1 and that carry comes and that will generate the sum C 2 and the carry C 3.

So, this way I can use this circuitry to get a 2 by 2 binary multiplier.

(Refer Slide Time: 12:43)

Multiplication: a special case

- In decimal, an easy way to multiply by 10 is to shift all the digits to the left, and tack a 0 to the right end.
$$128 \times 10 = 1280$$
- We can do the same thing in binary. Shifting left is equivalent to multiplying by 2:
$$11 \times 10 = 110 \quad (\text{in decimal, } 3 \times 2 = 6)$$
- Shifting left twice is equivalent to multiplying by 4:
$$11 \times 100 = 1100 \quad (\text{in decimal, } 3 \times 4 = 12)$$
- As an aside, shifting to the *right* is equivalent to *dividing* by 2.
$$110 \div 10 = 11 \quad (\text{in decimal, } 6 \div 2 = 3)$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, some special cases; so, in decimal an easy way to multiply by 10 is to shift all digits to the left and put a 0 to the right end. So, 1 to 8 multiplied by 10; so, we know the result is 1 2 8 and a 0. So, you shift the numbers digits by to the left and put a 0 at the right end. So, for the same thing we can do with binary one.

So, so we can shift left is equivalent to multiplying by 2. So, a shifting left is multiplying by 2; so, for example, this 1 1 multiplied by 1 0 is in decimal system it is 3 into 2 which is equal to 6. So, if you converted into binary notation; so, it is 1 1 0.

So, what has happened is that this 1 1; so, it has been left shifted and this less significant bit position we have put a 0 ok. So, that gives the binary shifting or this multiplication by 2 and if you shift it twice; shift left twice then it is a multiplication by 4. So, for example,

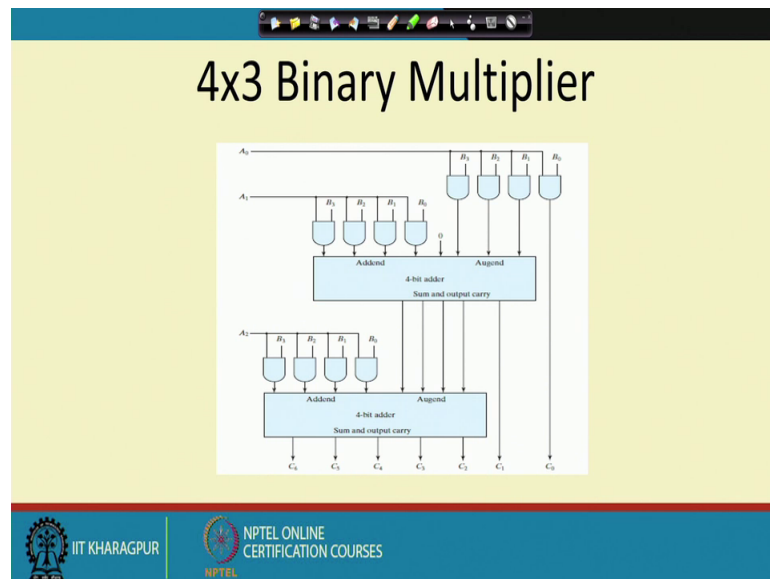
this 1 1; so, if I left shift twice then I will get 1100 and this number is nothing, but 12. So, this 3 into 4 is 12; so, this is basically left shifting by 2 bits.

So, shifting to the right is equivalent to dividing by 2. So, if you are ah; so for example, this 110 that is 6. So, if you are shifting it by right position; so, you are shifting it right means this 0 will go. So, 1 1 will remain; so, that is 1 1 the result is 3 this is nothing, but 6 divided by 2 is 3.

So, this multiplication and multiplication by 2; so, this is basically left shifting the number by one position, multiplication by 2 power i is left shifting the number by i positions and similarly division by 2 is right shifting the number by one position and division by 2 to the power i is right shifting the number by i positions.

So, this way we can have the special cases for multiplication by multiplication and division by powers of 2.

(Refer Slide Time: 14:50)



So, for 4-bit multiplication; so 4 by 3 binary multiplier; so, we have got one 4-bit input B 0, B 1, B 2, B 3 and we have got 3 bit input A 0, A 1, A 2. So, what we can do? So, we can generate these partial sums in this fashion ok. So, then they can be passed through some adders ok; so, two 4-bit adders will be necessary and we can passed them through the adder and ultimately we can get this sum which is consisting of 7 bits. So, 8 C 0 to C

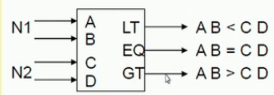
6 plus there will be an another carry; so, 4 plus 3; 7, the result may be 8 bit also. So, that way there will be another carry generated ok.

So, this way we can have this partial sums generated and we can do the multiplication. Of course, this multiplication process that we have talked about is pretty slow and in case of integrated circuit chips particularly in VLSI domain. So, even find many other efficient multiplication algorithm that have come up. So, this is the basic multiplication algorithm that we have discussed and the idea is to see how can we do it using logic gates and all.

Next we will be looking into another very important arithmetic module which is known as magnitude comparator. So, I have got 2 inputs I need to tell whether the one of them is equal to the other greater than the other or less than the other.

(Refer Slide Time: 16:21)



Two-Bit Comparator



block diagram
and
truth table

A	B	C	D	LT	EQ	GT
0	0	0	0	0	1	0
		0	1	1	0	0
		1	0	1	0	0
		1	1	1	0	0
0	1	0	0	0	0	1
		0	1	0	1	0
		1	0	1	0	0
		1	1	1	0	0
1	0	0	0	0	0	1
		0	1	0	0	1
		1	0	0	1	0
		1	1	1	0	0
1	1	0	0	0	0	1
		0	1	0	0	1
		1	0	0	0	1
		1	1	0	1	0

we'll need a 4-variable Karnaugh map
for each of the 3 output functions

So, if I have got a two-bit value; so, it is like this. So, see suppose AB is the first number and CD is the second number. So, what is what is happening is that this AB is AB is 0 0; then if CD is 0 then it less than is 0, equal is 1 and greater than is 0 that is these 2 numbers are equal. So, this EQ output is 1 and this less than and greater than these 2 outputs are 0.

Similarly, if AB is 0 0 and CD is 0 1; so, this less than output is 1 equal output is 0 and this greater than output is 0. Then this is 1 0 again the same thing; so, this is this AB is 0

0. So, whatever be the CD apart from 0 0 this less than will be 1, others will be 0. So, this way I can draw a truth table for I can make a truth table for writing the entire behavior of this block of the this comparator block.

So, for doing the minimization; so, we will need 4 variable Karnaugh map for each of this 3 output functions ok. So, it 4 variable Karnaugh map and accordingly you can do the minimization, get the circuitry and ultimately we can say it is less than equal to and greater than these 3 outputs we can see we can I will draw the corresponding logic circuit.

(Refer Slide Time: 17:44)

Two-Bit Comparator (cont'd)

	A				
	0	0	0	0	
	1	0	0	0	D
C	1	1	0	1	
	1	1	0	0	
	B				

K-map for LT

	A				
	1	0	0	0	
	0	1	0	0	D
C	0	0	1	0	
	0	0	0	1	
	B				

K-map for EQ

	A				
	0	1	1	1	
	0	0	1	1	D
C	0	0	0	0	
	0	0	1	0	
	B				

K-map for GT

LT = $A'B'D + A'C + B'CD$

EQ = $A'B'C'D + A'BC'D + ABCD + AB'CD' = (A \text{ xnor } C) \cdot (B \text{ xnor } D)$

GT = $B'C'D + AC + ABD'$

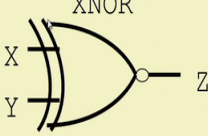
So, this is the thing; so K map for less than is $A' B' D + A' C + B' C D$ EQ is this one ok. So, this $A' B' C' D + A' B C' D + A B C D + A B' C D'$ that is either A and B actually the logic is that A and B both are 0 C and B should also be 0. A is 0, B is 1; so C should be 0 and D should be 1. So, this way I can do it.

So, this equality; so if you if you look it look into it more carefully; so, we can find that the this can also be written has $A \text{ xnor } C$ and $B \text{ xnor } D$ ok; that way it can be there.

(Refer Slide Time: 18:22)


Equality Comparator

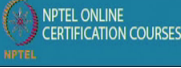

XNOR



$Z = X \text{ XNOR } Y$

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

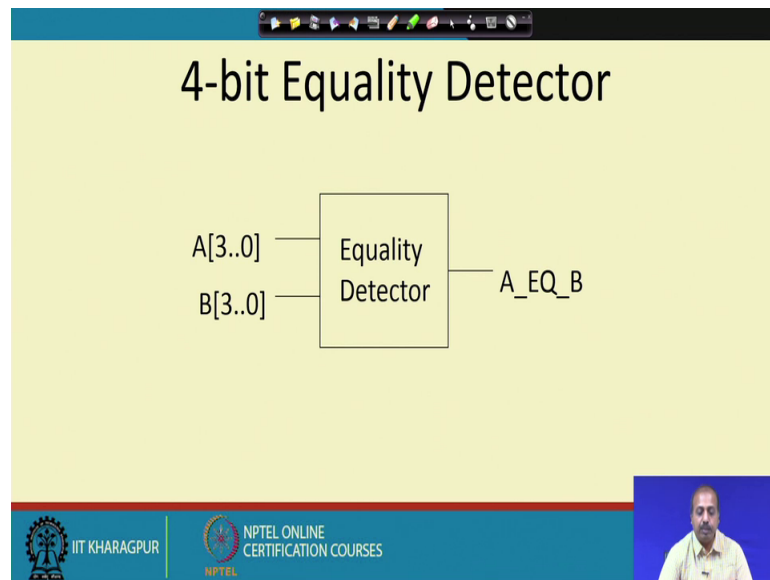




So, this is the; so,. So, that way we can draw the corresponding truth table and from there I can get the logic functions realized in terms of basic logic gates. Now let us look into to this equality comparator more carefully; suppose I have got only 2 bits to be compared and I have say whether the bits are equal or not. So, in terms of truth table; so, if it is 0 0 then the output is 1, if it is 0 1; it is 0 1 0 is 0 and 1 1 is also 1, for these 2 cases it is 1.

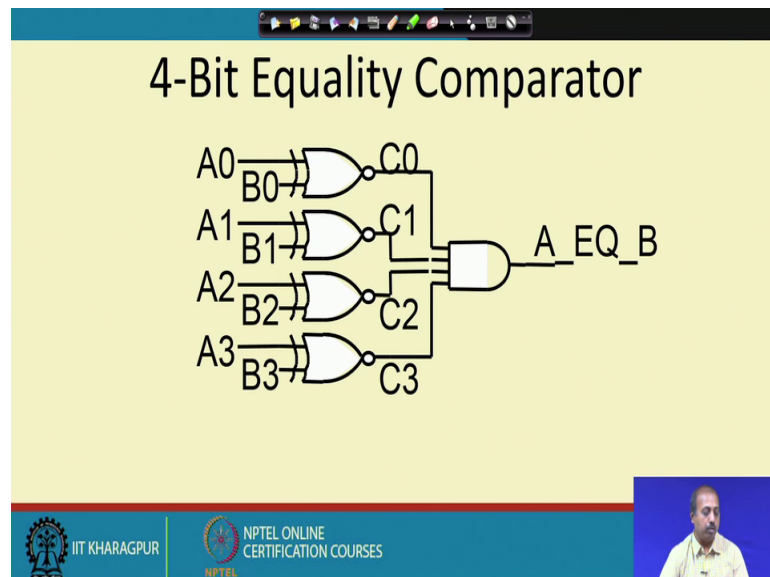
Now, you see the this truth table correspond to simple XNOR logic. So, it is a 2 input XNOR gate; so, this is what is happening here. So, Z is equal to X XNOR Y; so, this XNOR gate is an equality comparator.

(Refer Slide Time: 19:20)



So, if I have if I need a 4-bit equality comparator equality detector; then this may be the block diagram of that. So, I have got this A 4-bit input A_3, A_2, A_1, A_0 and similarly B_3, B_2, B_1, B_0 . So, this is the this is the B 4-bit B input and it answers A equal to B. So, this output is 1 only when this A and B input this A and B input are equal to each other.

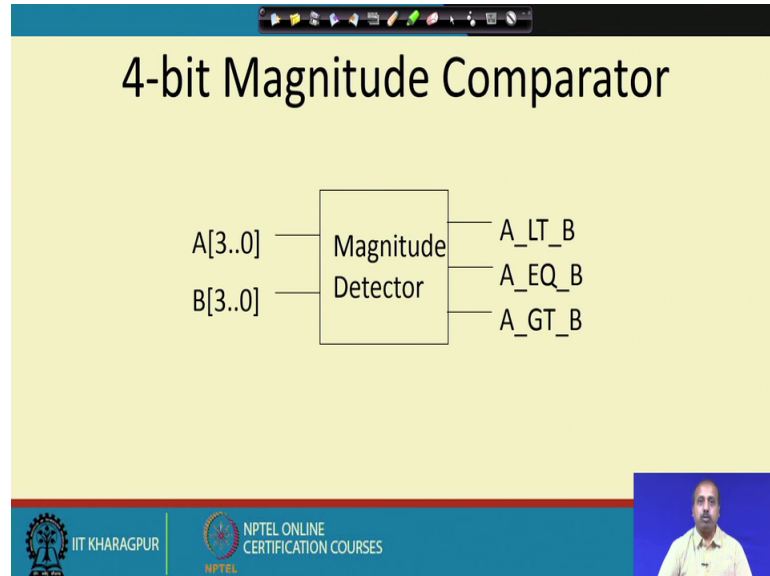
(Refer Slide Time: 19:37)



So, I can take a very simple circuit consisting of 4 XNOR gates this A_0, A_0 and B_0 fed to the first XNOR A_1, B_1 fed to the next X NOR like that. And now this $C_0, C_1, C_2,$

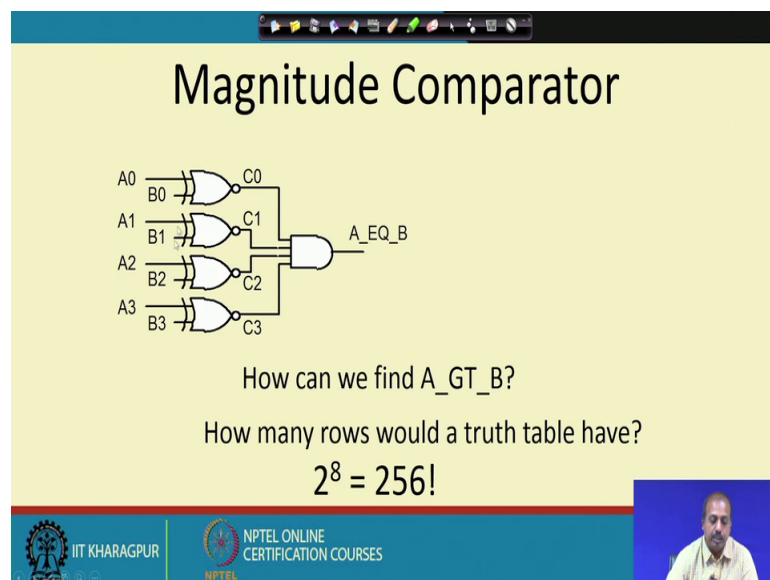
C 3; so, they are; they are to be anded. So, this AND gate is not; so, they are to be anded to get A equal to B ok; so, that can be a 4-bit equality comparator.

(Refer Slide Time: 20:00)



But what about magnitude comparison like; so not only equality, I also want less than and greater than output. So, the A is less than B or A is greater than B.

(Refer Slide Time: 20:15)



So, equality is definitely fine; so how can we get A greater than B. So, how many rows in the truth table have? So, if you try to answer this question then since there are 8 inputs A

0 to; since we have got 8 inputs A₀ to A₃ and B₀ to B₃; total number 8 input say if I am trying to draw a truth table; so, there will be 256 entries.

So, if I am trying to draw a Karnaugh map; then it is an 8 variable Karnaugh map that we have to minimize. So, that is a very clumsy way that is way very; so for in our course we have seen that we can draw K map up to 6 variables not more than that. So, that makes it difficult; so we have to do some something else for getting the minimized form.

(Refer Slide Time: 21:00)

Magnitude Comparator

Find A_GT_B
 Because $A_3 > B_3$
 i.e. $A_3 \cdot B_3' = 1$

If $A = 1001$ and
 $B = 0111$
 is $A > B$?
 Why?

Therefore, one term in the logic equation for A_GT_B is $A_3 \cdot B_3'$

The slide contains a logic diagram for a magnitude comparator. It shows four XOR gates labeled C0, C1, C2, and C3. The inputs are A0, B0, A1, B1, A2, B2, A3, and B3. The outputs of C0, C1, and C2 are connected to a single 3-input AND gate, which produces the output A_EQ_B. The output of C3 is also shown. The slide also includes a logic equation for A_GT_B and a numerical example where A=1001 and B=0111, asking if A > B and why. The answer provided is that because A3 > B3, the condition is A3 · B3' = 1.

.So, if you see that if A equal to 1001 and B equal to 0111 then A is greater than B; why? This is because this most significant bit A₃ is equal to 1 here and B₃ B₃ is 0 here; so A is greater than B. Now because A₃ greater than B; so A₃; so the condition is A₃, B₃ dash or A₃, B₃ bar is equal to 1.

(Refer Slide Time: 21:36)

Magnitude Comparator

$A_GT_B = A_3 \cdot B_3'$
 $+ \dots$
 Because $A_3 = B_3$ and
 $A_2 > B_2$
 i.e. $C_3 = 1$ and
 $A_2 \cdot B_2' = 1$

If $A = 1101$ and
 $B = 1011$
 is $A > B$?
 Why?

Therefore, the next term in the logic equation for A_GT_B is $C_3 \cdot A_2 \cdot B_2'$

Now, one term in the logic equation for A greater than B is $A_3 \cdot B_3'$. Now what about this one? Say if I have got A equal to 1101 and B as 1011. So, here also A is greater than B because the first 2 bits A_3 and B_3 were same and A_2 was 1 and B_2 was 0. So, I can say that A_3 is equal to B_3 and A_2 greater than B_2 that is why I have got C_3 equal to 1. So, C_3 equal to one if $A_2 \cdot B_2'$ equal to 1 and A_3 equal to B_3 .

So, we have got ah; so, so C_3 was A_3 equal to B_3 ok. So, the C_3 was a 3 equal to B_3 ; so, the conditions becomes C_3 and A_2 and B_2 bar B_2 .

(Refer Slide Time: 22:22)

Magnitude Comparator

$A_GT_B = A_3 \cdot B_3'$
 $+ C_3 \cdot A_2 \cdot B_2'$
 $+ \dots$
 Because $A_3 = B_3$ and
 $A_2 = B_2$ and
 $A_1 > B_1$
 i.e. $C_3 = 1$ and $C_2 = 1$ and
 $A_1 \cdot B_1' = 1$

If $A = 1010$ and
 $B = 1001$
 is $A > B$?
 Why?

Therefore, the next term in the logic equation for A_GT_B is $C_3 \cdot C_2 \cdot A_1 \cdot B_1'$

That way if we just shift by one more position; so, you will get the condition one more condition there. So, that will be this is the thing A 3 equal to B 3, A 2 equal to B 2 and A 1 greater than B 1. So, we have got C 3 equal to 1, C 2 equal to 1 and A 1 B 1 dash equal to 1.

So, we see that the A 3 greater than A greater than B terms becomes C 3 and C 2 and A 1 and B 1 bar.

(Refer Slide Time: 22:51)

Magnitude Comparator

The diagram shows a 4-bit magnitude comparator circuit. It has four inputs: A0, A1, A2, A3 and B0, B1, B2, B3. The circuit consists of four XOR gates (C0, C1, C2, C3) and one AND gate (A_EQ_B). The XOR gates are connected as follows: C0 takes A0 and B0; C1 takes A1 and B1; C2 takes A2 and B2; C3 takes A3 and B3. The outputs of C0, C1, and C2 are connected to the inputs of the AND gate A_EQ_B. The output of the AND gate is A_EQ_B.

Logic equation for A_GT_B:

$$A_GT_B = A3 \cdot B3' + C3 \cdot A2 \cdot B2' + C3 \cdot C2 \cdot A1 \cdot B1' + \dots$$

Because A3 = B3 and A2 = B2 and A1 = B1 and A0 > B0
 i.e. C3 = 1 and C2 = 1 and C1 = 1 and A0 . B0' = 1

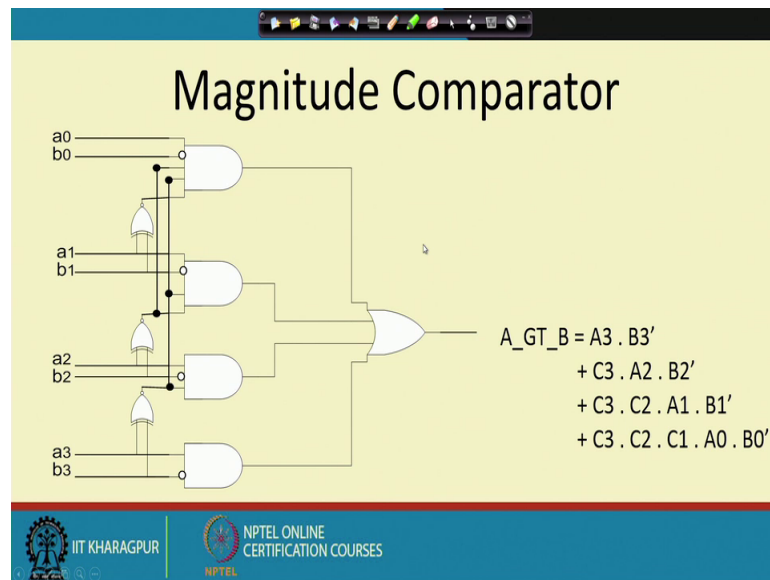
Therefore, the last term in the logic equation for A_GT_B is C3 . C2 . C1 . A0 . B0'

If A = 1011 and B = 1010 is A > B? Why?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

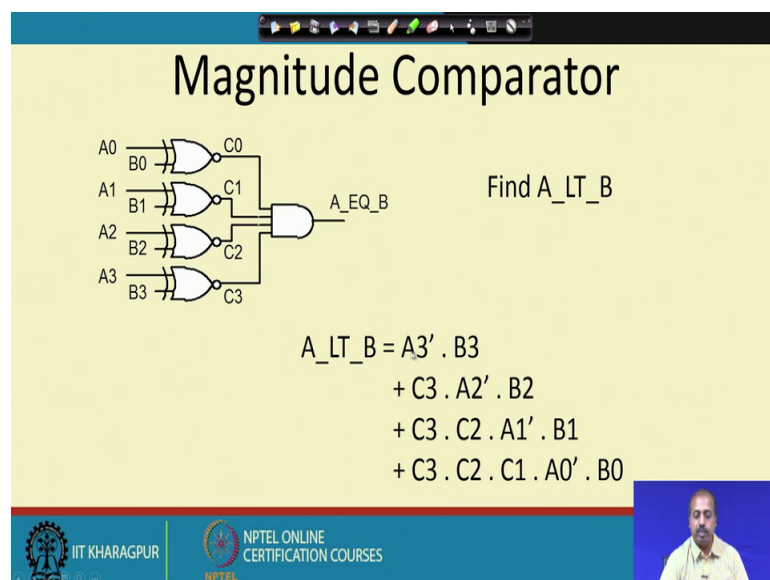
And naturally, I will have one more stage where it will be doing this B 0 also like only the B; B 0 A 0 and B 0 bits are differing. So, if ah; so, this is the situation the A 0, B 0 dash is equal to 1, C 3 equal to 1, C 2 equal to 1 and C 1 equal to 1. So, I will get the last term as C 3, C 2, C 1; A 0, B 0 dash.

(Refer Slide Time: 23:15)



So, the final expression becomes like this; so, A greater than B is $A_3 B_3 \text{ dash} + C_3 A_2 B_2 \text{ dash} + C_3 C_2 A_1 B_1 \text{ dash} + C_3 C_2 C_1 A_0 B_0 \text{ dash}$. And this can be realized by this circuit you see that we have got a very clean circuit here and we do not take help of the Karnaugh map and all. So, sometimes that makes that process makes it very difficult and we have to use some other logic to come to the circuit.

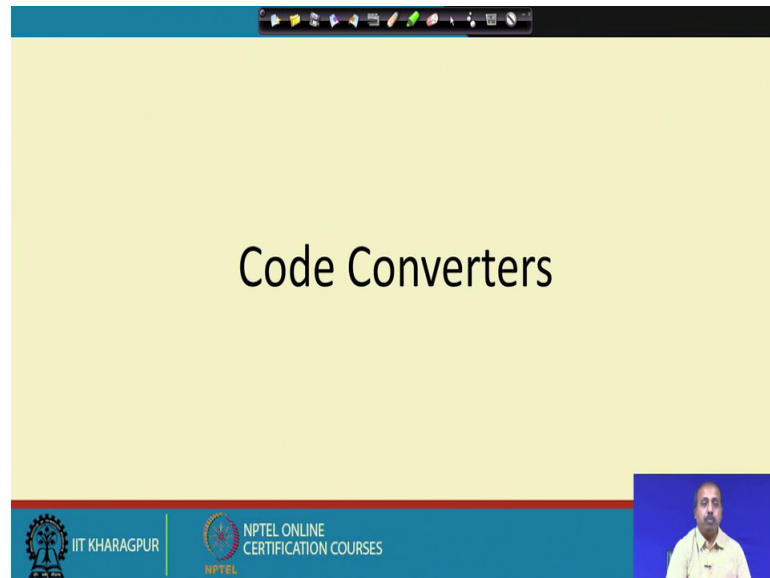
(Refer Slide Time: 23:54)



So, similarly you can say A less than B A less than B is given by this formula $A_3 \text{ dash} B_3$. So, A_3 is 0 and B_3 is 1 or $A_3 B_3$ are same detected by this condition C_3 and $A_2 \text{ bar}$

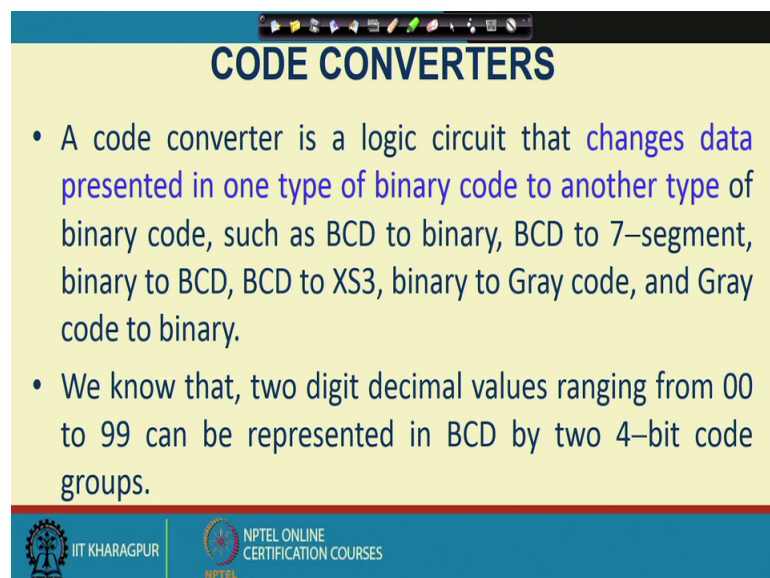
A₂ is 0 and B₂ is 1 or C₃, C₂ that is A₃, B₃ are same A₂, B₂ are same and A₁ is 0 and B₁ is 1 and this condition A₃, B₃ same A₂, B₂ same A₁ B₁ same and A₀ is 0 and B₀ is 1. So, that way I can write down the condition for A less than B.

(Refer Slide Time: 24:24)



Next, we will be looking into another important topic which is known as code converters. So, many a times we want to convert one numbers coded in some number system into some other number system. So, that is that is known as code converters; so, they coded in different form.

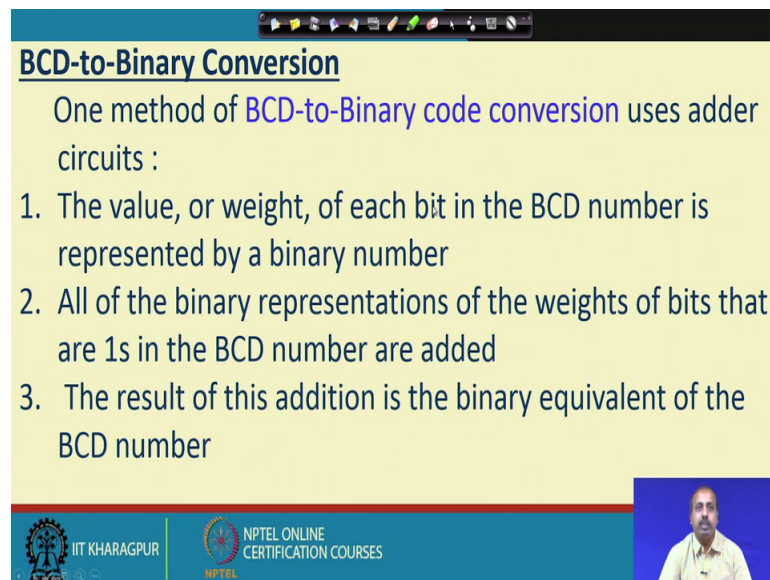
(Refer Slide Time: 24:45)



So, this is our code converter it is a logic circuit that changes data represented in one type of binary code to another type of binary code. For example, BCD code to binary code, binary code to BCD code BCD to 7 segment binary BCD to 7 segment display code, then binary to BCD; BCD to XS 3 code, so like that binary to gray code gray to binary code.

So, there are different types of codes that are there and many a times we need to interchange between this type this type of code. So, ok; so, how to do this code conversion? That we will see now; so, 2 digit, 2 digit decimal values ranging from 0 0 to 99 can be represented by BCD BCD by 2 4-bit code groups; so, that can be utilized for doing this conversion.

(Refer Slide Time: 25:36)



BCD-to-Binary Conversion

One method of **BCD-to-Binary code conversion** uses adder circuits :

1. The value, or weight, of each bit in the BCD number is represented by a binary number
2. All of the binary representations of the weights of bits that are 1s in the BCD number are added
3. The result of this addition is the binary equivalent of the BCD number

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So; first will be looking into BCD to binary conversion, so one method for BCD to binary conversion is by some adder circuit. So, value or weight of each bit in the BCD number is represented by a binary number and all of the binary representations of the weights of bits that are 1 in the BCD number; they will get added. So, this way we can have some adder circuitry and using that adder we can convert the BCD number to binary number.



(Refer Slide Time: 26:08)

Contd...

For example, 46_{10} is represented as

4	6	→ Decimal
0100	0110	

- The MSB has a weight of 10, and the LSB has a weight of 1.
- So the most significant 4-bit group represents 40, and the least significant 4-bit group represents 6 as in Table.



So, it is like this. So, 46; so 46 this is the 4 is this is the notation and 6 this is the thing. So, this is the decimal number and this is the BCD number. Now this most significant bit it has got a weight of 10 and the least significant bit has a weight of 1.

So, the most significant 4-bit group represents 40 and the least significant 4-bit represent group is 6 ok. So, because this is this weight is 10 and this weight is 1. So, this is 4 into 10 plus 6; so that we can do.

(Refer Slide Time: 26:44)

	Bit position							
	$10^1 = 10$				$10^0 = 1$			
Decimal weight								
Binary weight	2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
BCD bit Weight	$2^3 \times 10$	$2^2 \times 10$	$2^1 \times 10$	$2^0 \times 10$	$2^3 \times 1$	$2^2 \times 1$	$2^1 \times 1$	$2^0 \times 1$
	80	40	20	10	8	4	2	1
Bit designation	h	g	f	e	d	c	b	a
46_{10}	0	1	0	0	0	1	1	0
		40				4	2	

Weight Table

So, we can say that this bit position; so, this is decimal weight is bit position 1 and bit position 0. So 10 to the power 0 and 10 to the power 1 so, if the binary weight of that. So, if you look into that number at weight 10 and look into the corresponding. If you look into the corresponding binary number binary weights, this is 2 power 0 1 2 and 3. So, the BCD number; the BCD bit weight is 2 power 3 into 10, this is 2 power 2 into 10, this is 2 power 1 into 10 and 2 power 0 into 10.



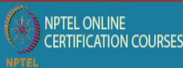

Now, so for example, this 46; so 46 it has got 0100. So, this 0 gives a contribution of 0 then this; so, this 4 this 4 is represented by the first 4 bits and the 6 is represented by the next 4 bits. And for the first 4 bits they have got a weight of 10 now this numbers; so, now this 0 multiplied by 10 that gives me 0; 1 1 is. So, this is 2 power 2; so, that is 4 multiplied by 10 that gives me 40, then this is 2 multiplied by 0 2 power 1 into that value is 0; so that is that is giving me 0.

So, that way; so this part gives me 10 and this part gives me 6.

(Refer Slide Time: 28:11)

The binary equivalent of each BCD bit is a binary number representing the BCD bit weight

BCD bit position	BCD bit weight	Binary representation
a	1	1
b	2	10
c	4	100
d	8	1000
e	10	1010
f	20	10100
g	40	101000
h	80	1010000

So, so binary equivalent of each BCD bit will be is a binary number and it is represented as a BCD weight bit. So, this is BCD position a, b, c, d, e, f, g, h as we have seen here this bit designation a, b, c, d, e, f, g, h. So, they have got this BCD weight values like this and corresponding binary representations like this ok.

(Refer Slide Time: 29:20)

Example :
Convert the BCD equivalent of 26 to binary.

Solution

2	6		
0010	0110		
		00000010	2
		00000100	4
		+ 00010100	20
		<u>00011010</u>	26

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, similarly suppose I have to convert this 26 into binary. So, again we do we do the same thing that; so, this part is all of them they will contribute to 10 and this this is 2 power 0 1, this is 2 power 1 2, 2 power 2 4 and this is 8.

So, this is this is 2; so, 2 into 10; so that gives me 20. So, I write down the binary representation for 20 then in thus in this part. So, this part is this weight is 10 power 0 for this all the 4 bits the weight is 10 power 0. So, this is this bit is 1, 2; so 2 into 10 power 0 that is 1; so that that gives me 2; so, I write down the binary representation for 2. Similarly, here I write down the binary representation for 4 and I sum them up; so, this gives me the binary 26.

So, in this way we can convert BCD numbers to equivalent binary numbers; of course, we have got other avenues. So, we can have a truth table and write down the corresponding Boolean functions for doing this conversion also; as well.