Principles of Mechanical Measurement

Dr. Dipankar N. Basu
Department of Mechanical Engineering
Indian Institute of Technology, Guwahati

Module – 03
Digital Techniques in Measurement
Lecture – 2
Binary logic gates & binary codes

Good morning friends, we are back with the second lecture of our third week on a fine nice school morning at Guwahati, the weather is quite brilliant outside today. But of course, we cannot feel that sitting inside this small studio; here what I have left with is only this monitor and the camera through which we are having this all interactions.

Now in the previous lecture we started talking about the adoption of Digital Techniques in Measurement. We have discussed about the importance of digitalization that is how digitalizing an output signal or sometimes even input signal helps us in storing large amount of data and also the post processing part.

We have briefly seen the method or I should say the principle of digitising a continuous signal. And then we have discussed about the number system; we have discussed about different number bases where decimal is the most common base. We have talked about binary octal and hexadecimals particularly binary which is most commonly used in digital instruments like computers.

And I am sure that you now clearly have idea that why binary is the one that is preferred among all possible kind of number bases simply because there are just two levels; one is 0 other is 1. And this 0 and 1 can be visualised in terms of several contrasting features; like we were just think about as one as off other is on. Like 0 is off and 1 is on or you can think about the 0 is low and 1 is high or 0 is small, 1 is large; something like that.

And that way if you think properly then any digital instruments requires electricity as a input. And therefore, it is quite easy for us to give the binary form of signals or to deal with binary form of signals because if you stop the signal by say switching off something switching off the line, then it represent the absence of something which is 0. Whereas, when you switch on the line then you have a current flowing through the circuit and that can be resembling 1.

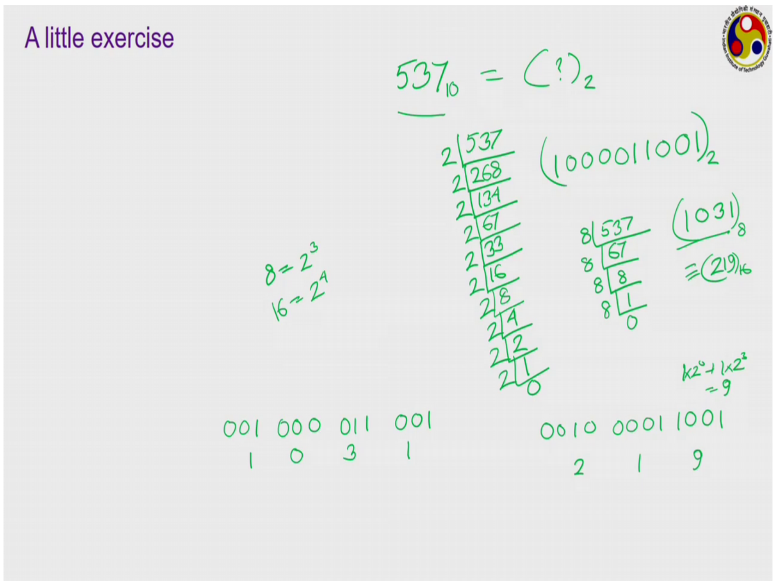
Something like when a light is switched off and the light is switched on; whereas, if you want to use some other kind of number systems then we need to consider several possible combinations. In a binary system we can have just two combination, one is off or two possible scenarios one is off other is on and so the corresponding electrical design is very very simple.

But if we want to think about say a decimal system adoption of a decimal system. Suppose think about we have a 10 v; 10 volt source 10 volt voltage source, now if we want to use a binary form then we can easily setup 2 limits. Like something like say between 0 to 5 any voltage coming which is small that is ranging between 0 to 5; we shall be treating that is 0, anything coming between 5 to 10 that is 1.

So, they are very clear distinction between the 2 and there is a large range to work with for each of them. However, if we want to apply the same instrument with digital representation then you have to make 10 different voltages or 10 different ranges I should say. Like 0 to 1 corresponding to 0 or maybe just something 0.5 and less than that correspond to 0 then 0.5 to 1.5 corresponding to 1 or something in that range.

So, much more difficult to controlled; much more difficult to use in instruments; that is a main reason we go for binary in all these kinds of instruments. But one problem with binary is that we give much longer number changes; like a very small decimal numbers can lead to a quite big binary numbers.

(Refer Slide Time: 04:17)



A little exercise

$537_{10} = (?)_2$

$$\begin{array}{r|l} 2 & 537 \\ 2 & 268 \\ 2 & 134 \\ 2 & 67 \\ 2 & 33 \\ 2 & 16 \\ 2 & 8 \\ 2 & 4 \\ 2 & 2 \\ 2 & 1 \\ & 0 \end{array}$$

$(1000011001)_2$

$$\begin{array}{r|l} 8 & 537 \\ 8 & 67 \\ 8 & 8 \\ 8 & 1 \\ & 0 \end{array}$$

$(1031)_8$

$=(219)_{16}$

$8 = 2^3$

$16 = 2^4$

$1\times2^0 + 1\times2^3 = 9$

001 000 011 001

1 0 3 1

0010 0001 1001

2 1 9

Let us do one small exercise; a couple of small exercises to start our day. Let us pick up any random decimal number. So, let us say we are taking 537 as a decimal number; what this 10 signifies? This 10 refers to the base which we are using and our objective is to get the corresponding binary representation.

. So, what is a method of converting something a decimal to binary? So, whenever we are having a decimal number and we want to convert this to a number of other base, then we have to divide that by the base of that number. So, if we divide this then we have 13 and what is the remainder here? That is 1, usually noting the 1. Now 268 we divide it by 2 again it gives us 134 this is being an even number remainder is 0. So, that remainder 0 we have to note down now which side we shall be noting now.

The first one that you have got that is the least significant bit; so this is 0. Now if you divide by 2 again, then integers we are going to get 6 7 and another 0 because this is again even number; we divide by 2 again this again odd number. So, that leaves us a remainder of 1 and we are getting 33, we divide by 2 again another odd number, so we are getting remainder as 1 and 16; now it is an even number. If we divide by 2, there is no remainder left and we are getting 8 divided by 2 again; again no remainder left we left with 4, divide by 2 again no remainder left.

So, we are left with 0 and 2 here and finally, if we divide by 2; then again no remainder left left with 1 final divide it by 2, division result is 0 and we are left with 1. So, it is a quite long chain of numbers. Now how many digits we have? 1 2 3 4 5 6 7 8 9 10; that means, we started with a 3 digit decimal number and we have ended up with a 10 digit binary number ah; I hope I have done it correctly if there is any mistake you please try on your own. How many digits we should have; how many divisions we have done 1 2 3 4 5 6 7 8 9 10 11 and corresponding to each of them we are getting 1 reminder.

So, this way we can convert any decimal number to corresponding binary representation, but binary always gives us; like in this particular case is very very long scale of long chain of numbers long stream of numbers and therefore, sometimes we prefer going for octal or hexadecimal basis. Like if we want to get the octal version of the same quantity this 537; then we have to divide it by 8.

So, if we divide it by 8 we are going to get how much? 6; 48 and 57; so 7 leaving us 1 as remainder. Now if we divide by 8 again here 8 and we are getting 3; yes and now you divide by sorry; not 2 if we divide by 8 again, we are getting 1 and 0 as remainder and finally, if we are divide it by 8 again; again result is 0 it is 1.

So, this is the corresponding octal, but we have also learnt there is once we have got the binary equivalent for a decimal or any number as a matter of fact, then there is no point going for such kind of calculations to identify its octal or hexadecimal versions because we can easily represent the binary in its octal or hexadecimal formation; formation.

What is the octal formatted binary; where octal refers to a base of 8 which is 2 cube. So, whenever we are trying to get the octal representation the binary number needs to be separated into several groups; each group comprising of 3 digits starting from the right most side or the least significant digit. So, if we write that binary starting from the least significant bit we have 1 0 0 that is the first group then we have 1 1 0 the second group, then we have 0 0 0 ia a third group and then finally, we have just 1. So, let us add couple of 0s before that to complete the group.

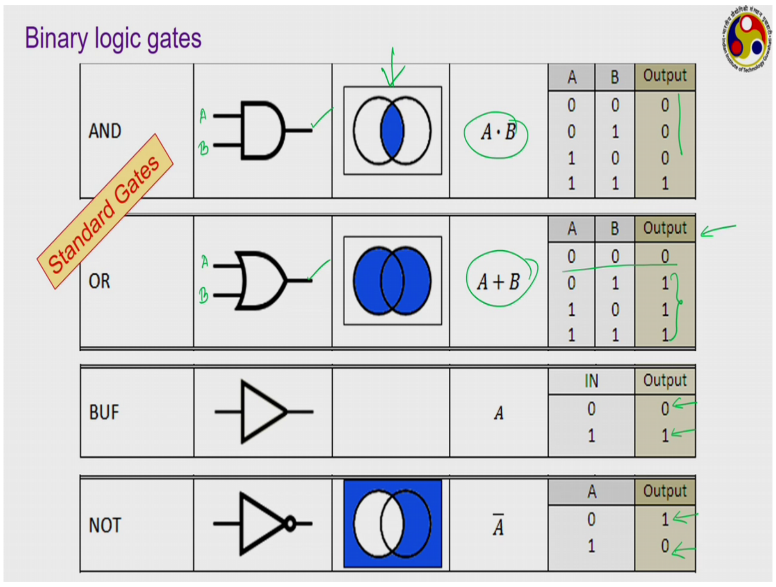
So, there are 4 groups here and each of the group then from that group try to identify what is the octal equivalent of that. Like the first group is having 1 octal equipment is 1, this is 0 octal equivalent is 0, then this is 1 1 octal equivalent is 3; this is 1 octal equivalent is 1. So, we are getting 1 0 3 1, this is 1 0 3 1; the octal equivalence.

Similarly, if we want to get the hexadecimal equivalence it is 16. So, which refers to 2 to the power 4; that means, we have to represent the number in groups several groups with each group comprising of 4 digits. So, again starting from the LSB we have 1001; then we have 1000 and in the last one; we have 0 1 then we adding couple of 0 before that. Now what 1001 may refer to? In case of hexadecimal case; we have used a table earlier if you think about that properly then or let us try to identify this 1001 in hexadecimal format. What it will be? We are going to get 1 into 2 to the power 0 plus 1 into 2 cube will gives us 9; so this gives 9.

Then this gives 1 and this gives 2. So, the hexadecimal equivalent of the same thing will be 2 1 9 in 16 bit system; which we could have obtained by dividing this original 537 by 16 in for in the same procedure. So, this way we can convert any binary number to or any decimal number either to its binary octal or hexadecimal version and the opposite also. We can do that is a number from any other number system can go back to binary.

So, the binary numbers are it is quite straight forward and well established method of converting across one number system or across different number systems; so, from one number system to another. Now once we have got the binary numbers then what should we do; how should we use that for processing of signals or for combining different signals and that question brings us to this concept of binary logic gates.

(Refer Slide Time: 11:25)



Binary logic gates

Standard Gates

AND — $A \cdot B$

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR — $A + B$

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

BUF — A

IN	Output
0	0
1	1

NOT — $\bar{A}$

A	Output
0	1
1	0

Logic gates refers to certain kind of electrical circuitry which perform some kind of mathematical operation on binary digits or binary inputs. The first basic logic gate that we have is the AND gate; it generally commonly a 2 input gate, but we may have that more inputs also this kind of diagram is often referred as a Venn diagram.

Now, here for AND gate though we are calling it AND, we represent it as A dot B just like shown here. And the idea is that when both the inputs are 1 then only you are going to get an 1 as the output otherwise the all the for all other 3 possible combinations output will be 0. That means, say this is your input A and this is your input B. So, if both A and B are on, then only you are going to get a on signal as a output, if any one of them is off then it will not allow the current to flows through the circuit; both of them has to be on. And if any if both are 0 or both are off or even one is on and other is off; you are not going to get any output from this giving a 0 output 0 representation.

The other version of this one is OR; OR is written as A plus B. And it refers that like I am going back to the Venn diagram for AND, you can see this blue portion signifies the output. Now when you are going to get output? When the portion or the zone where both of them which is common to both input signals and then only all you are going to get output. This portion and this portion here like if we talk about say; this portion first; here one input is on, but other is not available.

Similarly in this portion one input is on the second input, but first one is not available; this the only portion where both the inputs are available and that is why you are going to get some output only in this blue portion. But in case of or whenever if any single input is available, we are going to get some output. That is if A and B both are on or even if one is on; you are going to get something there; it will not give you output that is it will switch off your circuit only if both A and B are off.

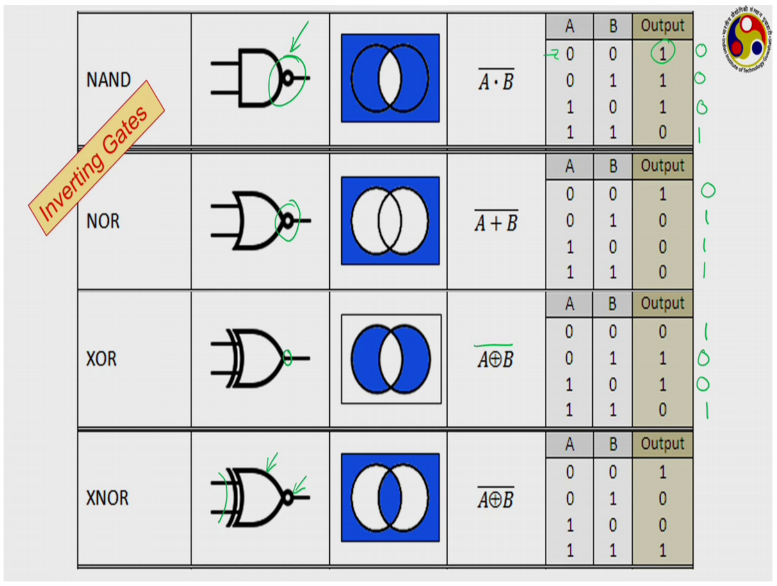
Correspondingly, we get this particular table this kind of tables are often called the truth table. When both are 0 then only we are getting a 0 at the output otherwise you are always getting a 1; that is an on condition as the output. I repeat in case of AND gate you need to have both inputs to be in switched on condition to get an on output or some signal at the output; even if one of the signal is off there will be no output. On the contrary, the OR gate whenever even a single input is on, it is going to give you an output.

That these two are called the standard gates; the most basic gates these are. Then we can have single input gates this is the buffer sometimes also called the hex; it is just whatever comes in that goes through. Means if the input is switched off it is switched off if the input is switched on it will remain switched on. The opposite of that is a NOT gate sometimes called the inverter here whatever comes in you get the opposite of that. Like if 0 is coming in it is off, it will switch on the circuit whereas, if the input is on it will switch off the circuit.

Just look at a, what we have here in this Venn diagram? Whenever we are having both inputs of means the blue portion, then only you are having the signal output signal. So, this these two are called single input gates this NOT gate is particularly useful if it is also called the inverter because its action is just to invert whatever input signal is coming in. How these are done? Through electrical circuitry we shall be seeing briefly later on. Next we have a few combination of these basic gates particularly AND, OR or NOT gate.

First is NAND; NAND is NOT plus AND; if we just go back to the previous slide these are very standard these are the representations of the circuits; in the circuit diagram we shall be using this symbols. So, AND is represented by something like a semicircle whereas, OR is having this representation and NOT signifies the triangle and this particular 0 actually the triangle refers to the hex on the bof B u f, but this particular circle at the outlet portion or the exit portion signifies a NOT operation.

(Refer Slide Time: 16:13)


Inverting Gates

		A	B	Output	
NAND	$\overline{A \cdot B}$	0	0	1	0
		0	1	1	0
		1	0	1	0
		1	1	0	1
NOR	$\overline{A + B}$	0	0	1	0
		0	1	0	1
		1	0	0	1
		1	1	0	1
XOR	$\overline{A \oplus B}$	0	0	0	1
		0	1	1	0
		1	0	1	0
		1	1	0	1
XNOR	$\overline{A \oplus B}$	0	0	1	
		0	1	0	
		1	0	0	
		1	1	1	

So, NAND refers to NOT plus AND; to represent that we have to combine the signal for both AND and NOT, that is why we are having the semicircular representation with the circle at the exit. Its object or its output is just opposite to AND, like when both inputs are 0; it is going to give a 1.

Similarly, when either of the inputs are 0 it is still giving a 1; only if both inputs are 1, it is giving you 0. If we compare the output of AND single with this; for and when both are 0, we get a 0.

When 1 is 0, other is 1; we still get a 0; only when both are 1, we get a 1; here it is does the opposite simply because of this NOT function. We are having an AND gate and a NOT gate together to give a NAND. The representation again its a combination of AND and NOT because in case of AND, we write it as A dot B and in case of NOT; we write as A bar; so A dot B over what gives you a NAND operation. So, the objective of NAND as it suggests is just opposite to AND like in case of AND, we want to switch on the circuit only when both inputs are available.

In case of NOT, we want to switch off the circuit when both inputs are available. Next is NOR again it is a combination of OR and NOT; as the symbol suggest we have the OR symbol and then there is a small circle representing a NOT operation. So, it is again the opposite operation to OR; in case of what we get? When both inputs are 0, then only we get a 0 output; like for OR we know when both are 0; we get a 0 for every other situation we get 1. Here we are getting just the opposite when both are 0 then only you are getting a 1; otherwise all as 0.

Means it will switch off the circuit when both are absent; even if a single one is present or the both are present, still the output will keep on switched off; only when both outputs are both inputs are switched off, then only it will switch on the circuit. There are several scenarios where kind of operations are required.

So, these two are often called the inverting gates because they invert the operation of the basic gates standard gates that is AND and NOR. And then we have couple of exclusive gates XOR is also called the exclusive OR. Here this curved line this one represents the exclusive portion; there is a you can see there is an OR symbol and is an exclusive symbol.

This is a quite interesting the way mathematically is represent A plus B refers to OR and now this circle this plus symbol has been enclosed by a circle to represent this XOR. In case of XOR the output portion is quite interesting when both the input signal of the same kind, it is giving a 0 when both the inputs are of different kind it is giving a 1. Like when both inputs are switched off or when both inputs are switched on, you are getting no signal from the output side.

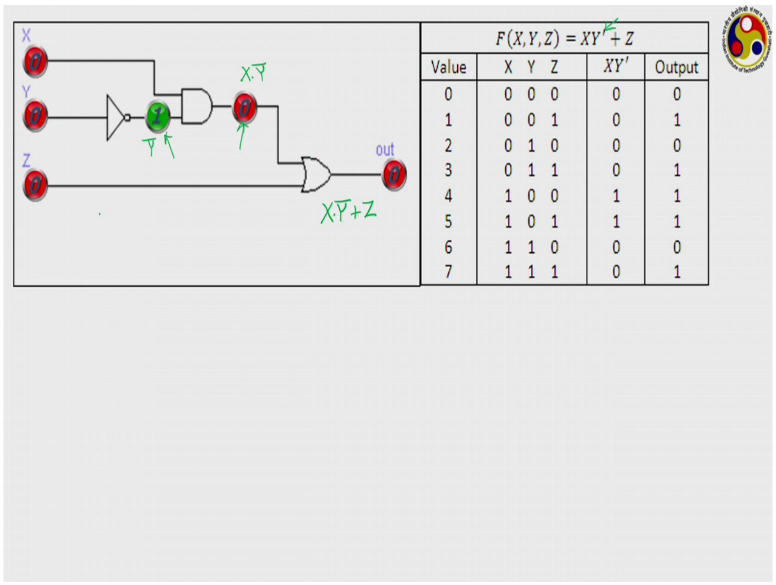
However when one of the input is switched on and other is switched off; you are getting an output your output gets switched off switched on should say. And we can also negate this operation like this a XOR; so if we add a NOT gate here, we are adding this operation by using this symbol; then that is a NOT version of this; how? That we should represent mathematically symbol wise, you have the symbol for XOR if we put an over bar that represent negation of that.

And what should be the output for an X; for such XOR plus NOT operation that should be the exactly opposite of whatever we are getting here. So, you should get 1 and 1 when both inputs are similar and 0 and 0 when both inputs are opposite that is exactly we get in XNOR. XNOR is exclusive NOT, exclusive OR plus a NOT.

So, you can see in the symbol this is a OR symbol then this is the exclusive part and then this is the NOT symbol and a mathematical symbol as given this. So, when both inputs are of the same kind; XOR cuts the circuit of whereas, XNOR create drives the circuit or keeps the circuit open; whereas, when both inputs are dissimilar XOR switch on the circuit, but XNOR does not give any signal output.

These 2 are called the exclusive gates. So, these are the fundamental kinds of gates that we get in digital operation and using the combination of these gates; we can perform any kind of binary operations also often it called the Boolean operations or Boolean algebra. Whenever we are having some binary signals available with us in the form of 0 or 1; we can use those signals or we can direct those signals to this gates; combination of these gates to get any kind of output that we want to have.

(Refer Slide Time: 21:11)



$F(X,Y,Z) = XY' + Z$

Value	X Y Z	XY'	Output
0	0 0 0	0	0
1	0 0 1	0	1
2	0 1 0	0	0
3	0 1 1	0	1
4	1 0 0	1	1
5	1 0 1	1	1
6	1 1 0	0	0
7	1 1 1	0	1

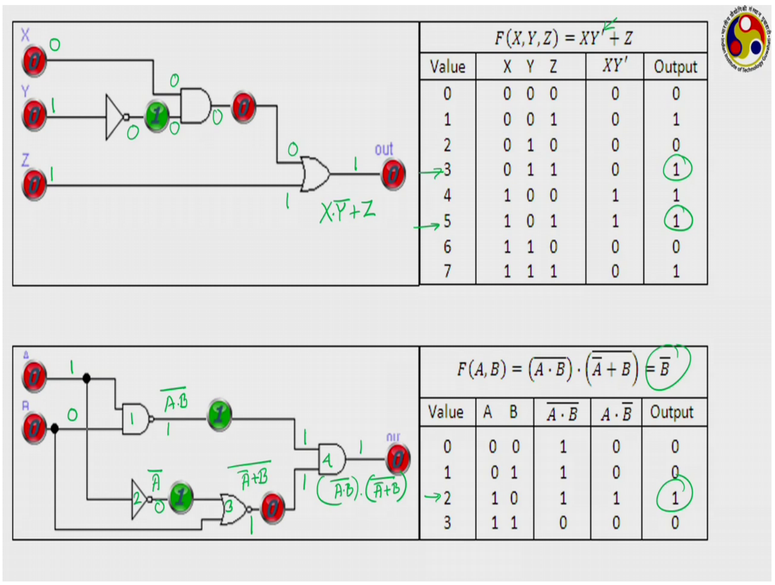
Like this is an example of a possible scenario; here you can see there are 3 different gates we are combining. Firstly, the output for rather the input signal from Y is first supplied to a NOT operation. So, whatever is coming Y you are getting a Y bar after this.

Now this Y bar and X are directed to an AND gate. So, what will be your output? That will be your X dot Y bar that is the output that we are getting from this operation or from this AND operation. Now this output from this AND gate is directed to this OR gate and also it is receiving input as Z. So, what will be your output from this OR gate? It will be X dot Y bar.

Let me write it once more; it will be X dot Y bar plus Z; this is what written as this prime is also another way of representing this bar. I shall be preferring to write Y bar, but sometimes in literature in books or even internets also you will find it is written as Y prime sometimes; just to denote the NOT notation. I have used this notation just to let you know that this another notation for not is also available.

So, you can have the truth table now for combining all 3. So, when X all 3 are 0 then what we are having there? All 3 are 0 then Y because of this NOT operation is becoming 1 here and then 0 and 1; is going to this giving a 0 output is OR and this Z is also coming as 0 giving 0 is a final output.

(Refer Slide Time: 22:57)


X
Y
Z
out
$X\cdot\overline{Y}+Z$
A
B
$\overline{A\cdot B}$
$\overline{A}$
$\overline{\overline{A}+B}$
$(\overline{A\cdot B})\cdot(\overline{\overline{A}+B})$

$F(X,Y,Z) = XY' + Z$

Value	X Y Z	XY'	Output
0	0 0 0	0	0
1	0 0 1	0	1
2	0 1 0	0	0
3	0 1 1	0	1
4	1 0 0	1	1
5	1 0 1	1	1
6	1 1 0	0	0
7	1 1 1	0	1

$F(A,B) = (\overline{A\cdot B})\cdot(\overline{\overline{A}+B}) = \overline{B}$

Value	A B	$\overline{A\cdot B}$	$A\cdot\overline{B}$	Output
0	0 0	1	0	0
1	0 1	1	0	0
2	1 0	1	1	1
3	1 1	0	0	0

However when we are having say we take a scenario where X is 1, Y is 0 and Z is 1; then what you are going to have? After the NOT operation Y becomes 1, then both 1 going to the AND operation giving another 1 as the output and now this Z is also 1. So, both are 1 to the OR gate; you are getting an 1 as the output; 1 0 1 is designious scenario; you get 1 as the output.

Let us try another example; let me erase this let us say when X is 0, Y is 1 and Z is 1. What you are going to get? Why because of this NOT operation becomes 0 here. So, this AND receives 0 and 0 giving a 0 as the output, now OR receiving 0 from the earlier AND gate and 1 from the Z: so, it will give an 1. 0 1 1 is this scenario you get a 1 as the output; yes.

This way we can combine 3 signals in through 3 different gates another example to deal with. Here we have 4 different gates, but just 2 signals first look at the diagram carefully. You have a NAND gate here be careful not to miss this; it is an AND plus this NOT; so it is an NAND operation and also you have a NOR operation here; sorry you have a NOT operation here. And finally, you have a NOR operation here and everything finally, directed to this end.

So, what will be your output? One case is shown here the truth table is also known, but later search through just try to see or properly. So, A and B both are coming to let us give some names. So, let us say this is number 1, this is our number 2, this is number 3 and this is number 4 this are our gate numbers.

So, after gate number 1 what we are going to see? A and B both going through a NAND operations. So, your output will be A dot B over bar same A and B are also directed to a sorry; A is directed to a NOT operation. So, the output of this NOT operation is A bar and now this A bar and B are directed to a NOR operation.

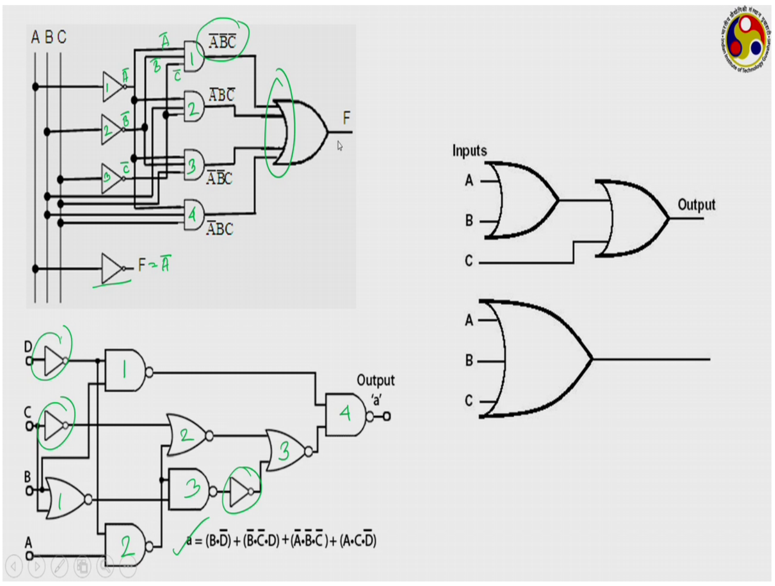
So, the output from this one will be A bar plus B whole over bar because of the NOR operation; this is the NOR part. And finally, this entire thing is going to a AND, so your final output will be A dot B over bar dot A bar plus B whole over bar. So, combining all this you can get the final output. I have not gone into the detail of Boolean algebra, it can you shown that this exact thing leads to B bar actually; this entire thing. We can see in terms of a few examples; let us say one example is already worked out here instead of that let us take both in say A is 1, B is 0.

Then what will be the output from the NAND operation? A 1, B 0 had it been an AND gate; your output should be 0, but is a NAND. So, output will be 1; now after the NOT operation, A leads to 0. And because that is A bar now this 0 and 0 from B; they both go to NOR and as both are 0 had been OR operation we would have got 0, but is a NOT; so we are getting an 1. So, this AND is receiving 1 from the lower side gate, this gate number 3 and another and another 1 from the NAND operation. So, this a standard and gate; so we are getting 1 as the final output.

So, 1 0 separation this the 1 final output; this the only cause you are getting 1 in this particular scenario for other all others are leading to 0. This way logic gates can be combined and we can get any kinds of outputs. Look at this particular example specifically here we are still deliver just 2 signals, but we want our circuit to operate such that only when A is 1 and B is 0; we want to get an output signal; in all other cases it should given a signal.

So, by combining these 3 we are getting an output signal only for this combination of A equal to 1, B equal to 0; for all other cases your outputs are 0. And this how we can combine any number of inputs through any number of circuits to get any kind of output variation.

(Refer Slide Time: 27:35)


A B C
$\bar{A}\bar{B}\bar{C}$
$\bar{A}B\bar{C}$
$\bar{A}\bar{B}C$
$\bar{A}BC$
F
$F = \bar{A}$
Inputs
A
B
C
Output
D
Output 'a'
$a = (B \cdot \bar{D}) + (\bar{B} \cdot \bar{C} \cdot D) + (\bar{A} \cdot \bar{B} \cdot \bar{C}) + (A \cdot C \cdot \bar{D})$

This are certain example like here you have an example of 3 inputs A B and C which has been directed through several operations to get possible outputs. Here particularly when this particular one this is just a plane NOT operation this and this is going to give you this is look at the input it is connected only to A.

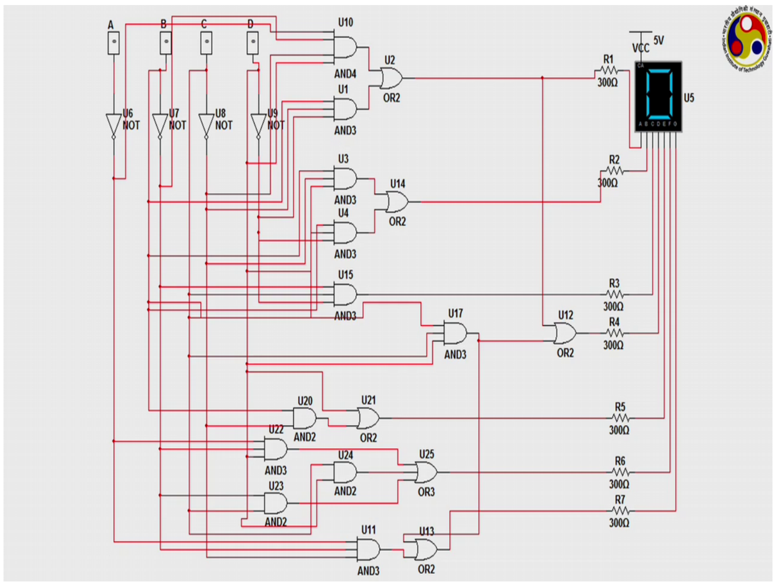
So, this is only going to give you A bar, but this one there are several operations there are 3 NOT operations. So, each of them what this NOT operators are giving? This say this the 1 2 and 3. So, 1 is going to give you output as A bar, 2 is going to give you output as B bar because that is connected to B, 3 is connected to C that is going to give you output as C bar. So, now we have 4 AND operations 1, 2, 3 and 4 and finally, outputs of all this 4 going to a single OR operation. Then what are the inputs to this first one? You can see the first one is having 3 inputs; the first input is coming from coming as A bar what is the second input? If you stretch the second input, it is leading you to B bar.

And if you stretch the third input; it is leading to the C bar. So, if you combine all of them you are getting A bar dot B bar dot C bar. Similarly, others also can be combined and we can get any kind of operations from this and there are several possible scenarios that we can deal with. This is another example where again we have 4 inputs and they have been combined with so many different number of gates. What are how many gates that you can identify here? Of course, there are several gates that you can count, but can you identify an AND gate here?

I cannot see any AND gate; is there any NAND gate? There are NAND gate like you can have 1 2, you can have 3, you can have 4; 4 different NAND gates are there. There is any OR gate? There is no OR gate also, but there are NOR gates. You have 1 2 and 3 NOR operation and also there are a few NOT operation; like one here, another one here, another one here NOT operation going on and by combining this we can get the final output a single output.

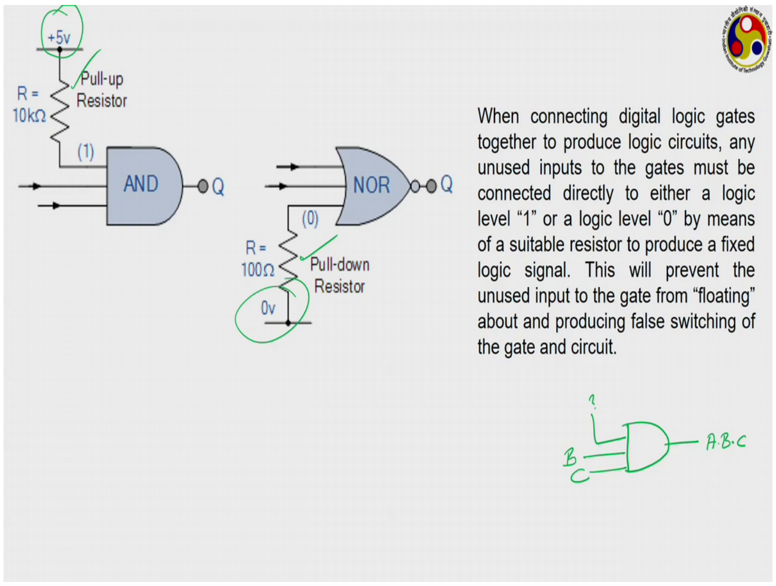
The a form for the final output is given and you please check this particular form or rather try to derive this by analysing this logic circuitry that will be a good exercise for you. The same circuit can sometimes take like that already have seen the examples; it can take commonly we talk about 2 inputs, but the same principle can be extended when you are dealing with multiple inputs like 3 or 4 to the same one.

(Refer Slide Time: 30:33)



Like in this case this particular gate is actually having 4 inputs; there can be much more number of gates combined in a practical circuit. Like this a very very complicated diagram for a commercial circuitry like this and as the number of circuit keep on increasing, we go towards the integration of this into the IC or integrated chips.

(Refer Slide Time: 30:45)



When connecting digital logic gates together to produce logic circuits, any unused inputs to the gates must be connected directly to either a logic level “1” or a logic level “0” by means of a suitable resistor to produce a fixed logic signal. This will prevent the unused input to the gate from “floating” about and producing false switching of the gate and circuit.

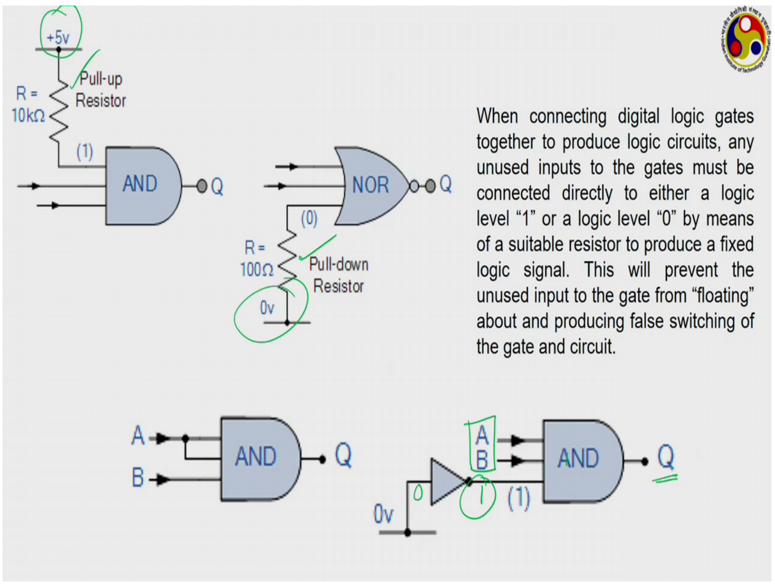
But now there is a one thing that we have to be careful when we are dealing with a gate; any kind of gate which is having multiple inputs 2 or more than that. Now sometimes it may be possible during operation that one of the inputs remains idle.

Like suppose you are dealing with the AND operation that is shown here, this is your AND gate which is going to give a single output, but that is having 3 possible inputs A, B and C. Finally, your output will be A into B into C or A dot B dot C, but in a certain scenario; your A is not having any output because you want to use this same gate for a 2 input system.

So, your one option is idle then what to do in this? If you keep it idle then it will lead to some destruction in the output and sometimes even a wrong prediction of the output. That is why we use the concept of this pull up resistor; pull up resister is depending upon a what default output that we want to get, we can connect it to a high voltage or a low voltage.

Like if we want if we want the output to be equal to 1 in the default scenario; then we shall be connecting this to a high voltage like this plus 5 volt here whereas, if we want to get the output to be 0; then we shall be connecting this one to a pull down resistor like shown here.

(Refer Slide Time: 32:29)



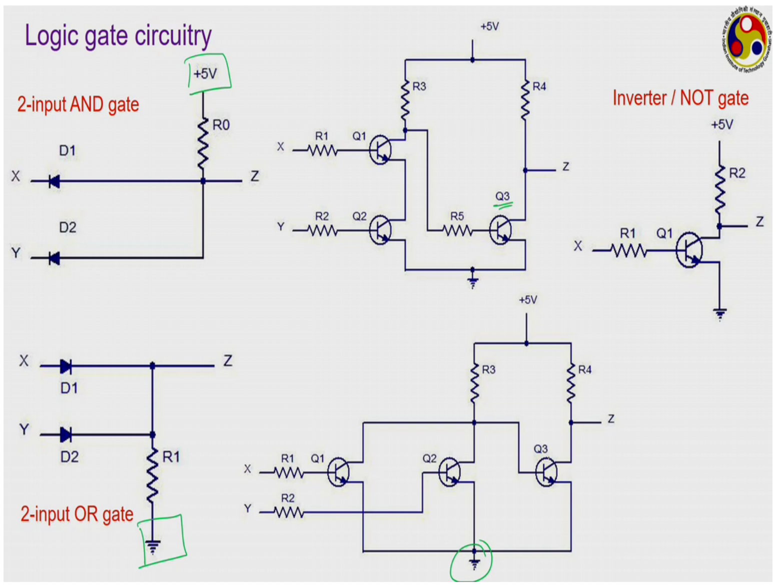
When connecting digital logic gates together to produce logic circuits, any unused inputs to the gates must be connected directly to either a logic level “1” or a logic level “0” by means of a suitable resistor to produce a fixed logic signal. This will prevent the unused input to the gate from “floating” about and producing false switching of the gate and circuit.

So, depending upon what logic levels that we want in a default scenario means when there is no input given to the instrument; then accordingly we can set up a value of pulled up or pull down resistors or we can go for a pull up resistor or a pull down resistor.

This is another example of the same; here there is a you can see the NOT operation has been performed we have provide is 0 volt. So, where is NOT; you are supplying 0, you are getting 1 from there and now depending upon the value of A and B you are going to get something. Like if, so this one is always being supplied to this AND gate independent of the nature of A and B.

There why you will always be; you are never expected to get any random output on this side this is the way we use the pull up or pull down resistor.

(Refer Slide Time: 33:07)



Now, how you are doing all these operation? Exactly what you have inside this logic gate? This course is directed more towards mechanical measurement that is why I shall not be going to the details of any electronics. This is just to give you some idea what kind of things that we do inside a such kind of logic gate operation done.

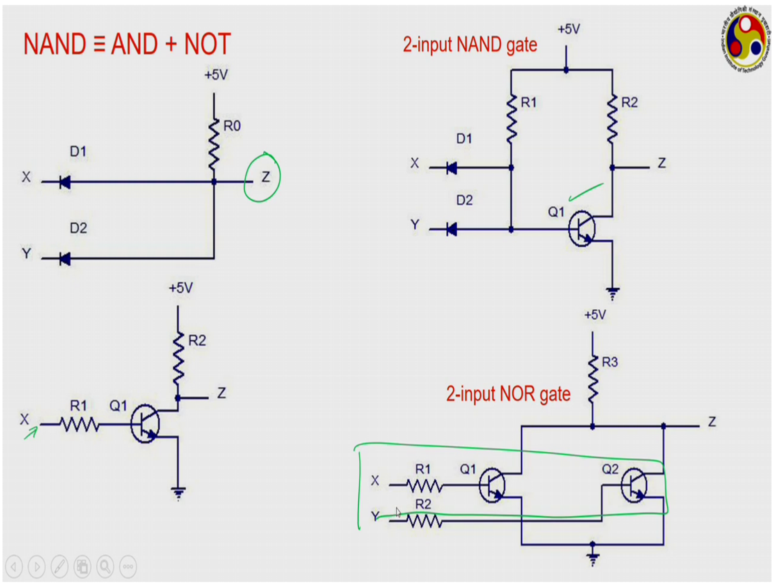
Like this a simple 2 input AND gate; you can see there are 2 diodes, there are several possible designs for the AND gate, this is one of the simpler designs when there are 2 diodes. And when the in the both the diodes are connected each of the diode is connected to one of the inputs and then their exits are connected to a common resister and a 5 volt system.

Now, only the scenario you are going to get an output here when both are going to have both X and Y are switched on or that is something is supplied there. If one of them is switched off or both are switched off then you are not going to get anything here. The same operation you are going to get if you are using instead of diode; if you are using transistors. Like shown here Q 1 and Q 2 are 2 transistors and another transistor Q 3 is used to form the output.

Similar operation we can get with OR gate as well, in case of a OR; we have diode based or we have quite similar operation to the AND. Only in this case here like here we provide a default 5 volt here the default is 0 because this is earth and accordingly we are going to get the output from this. Same we can have a transistor base formulation where instead of the transistor again on this Q 2; the second transistor has been grounded on one side of that.

Then the standard NOT operation through the diode we can again have a standard NOT operation; a single input device, we need single diode to get that. There are several other types of design as well and if any of you are interested in electronics; you can go into the detailed of this and find lots more materials in any standard electronics book or any standard instrumentation book.

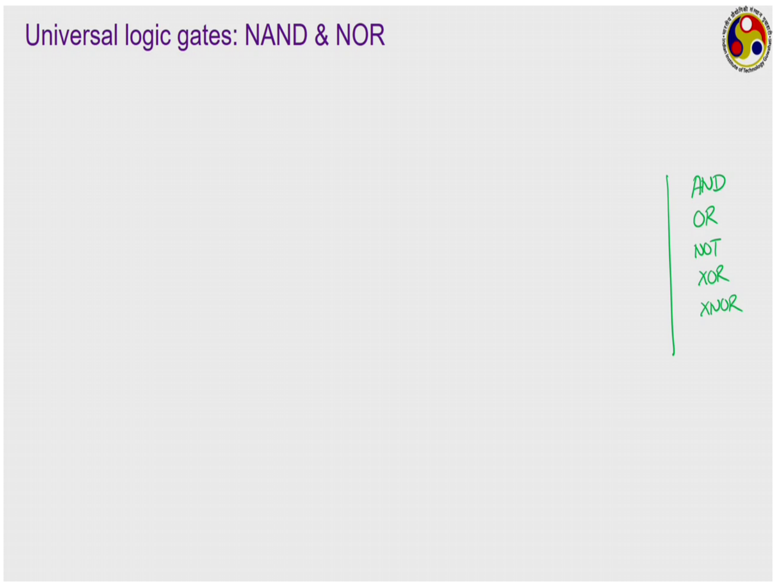
(Refer Slide Time: 35:05)



Now, NAND; NAND is AND and NOT; so, logical operation you have already seen what about the physical circuit? This is the top one here is for AND, the bottom on is for NOT. So, like in logical operation we have to combine AND and NOT; here also we can do the same that is we have to combine the circuit for AND then the circuit for NOT.

So, whenever we are going to get the output from this AND operation like here there itself we have to connect this particular point of NOT operation and that is what is done. You can clearly see the output Z has been connected to this NOT operation and we are going to get a NAND gate working. Similarly we can also get a NOR gate where this portion is related to the OR operation or I should say this portion was associated with the OR operation and then we can connect the NOT gate to get the NAND; sorry to get the NOR.

(Refer Slide Time: 36:07)



Universal logic gates: NAND & NOR

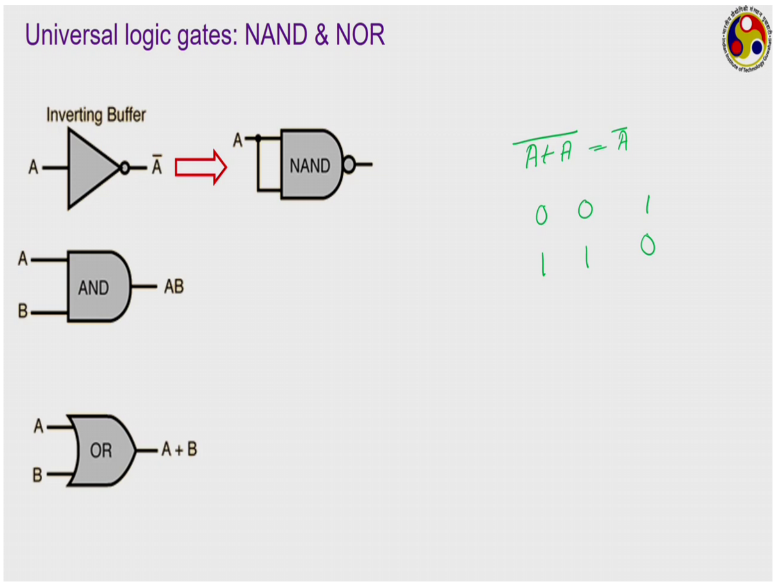
AND
OR
NOT
XOR
XNOR

NAND and NOR are actually quite interesting because they are often called the universal gate, universal logic gates. We have mentioned about 8 different types of logic gates out of them let us exclude X; we just allowed the signal to pass through in the form that it comes in.

So, there are 7 other gates; out of the 7 if we exclude the NAND and NOR, then we have AND, we have OR, we have NOT, we have XOR and we have XNOR. Interestingly, all this 5 operation of each of this 5 can be achieved by solely using NAND gate or solely using NOR gate. And that is why either these 2 gates are very very commonly preferred in circuit designs or instrumentation designs.

Because you if you do not have a AND gate available with you by combining a few NAND gates; you can get the AND operation done and the same with the others that I have just mentioned.

(Refer Slide Time: 37:07)



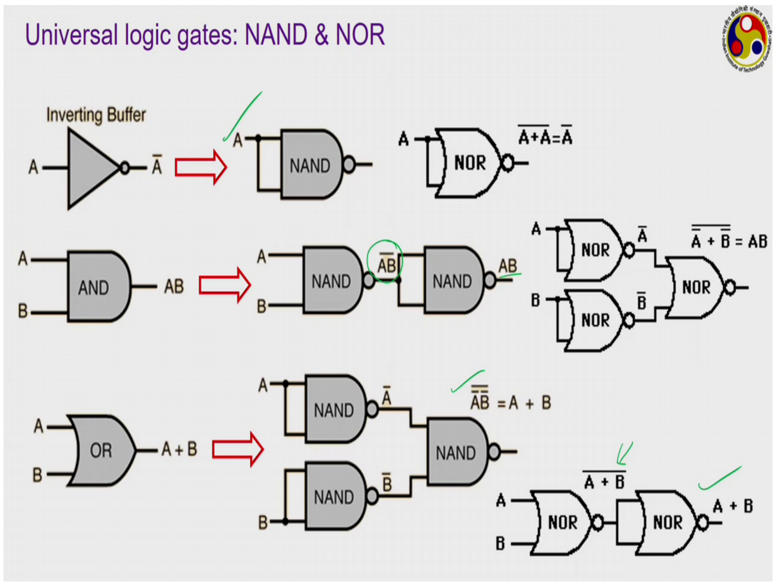
Universal logic gates: NAND & NOR

How you do this? Let us see these are the 3 standard operation; you have the NOT, you have the AND and you have the OR. How we get NOT operation using a NAND? NAND is AND and NOT; so here both inputs will be coming from the same source; then it will just act like a NOT operation.

Because here you are supplying A as the output; sorry A as input and both inputs; we are going to get A plus A and now you are having a NAND over bar. So, what is going to be the result of this? It has to be A bar only unlike if A 0; then you are having 0 and 0 and so your output is going to be 1.

Whereas if A is 1; you are going to have 1 and 1; supply to the NAND, your output is going to be 0 that is what your NAND is giving. Now let us see how we get the AND operation.

(Refer Slide Time: 38:01)



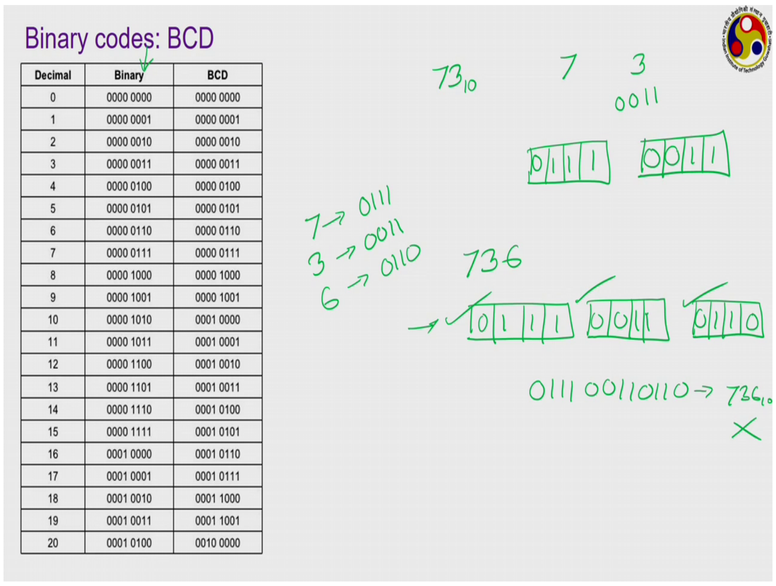
This the AND operation; we are supplying AND is NOT plus and sorry NAND is NOT plus AND, so we shall be doing the reverse that is we shall be perform the NAND operation or general NAND operation and then we shall be using a NOT operation done by your NAND gate; just what you have done here the same thing we shall be doing. So, we has input is A and B, output you are getting A dot B and the now is the NAND operation; you are getting A B as the output.

Like suppose your A is 1 and B is 0, so where the NAND operation you are getting 1 as the output and then we are reversing that getting a 0; so done the AND operation done. Similarly we can get a OR operation done combining 3 different NAND gates. The first one is take having A as the input in both inputs giving A bar, second one B is connected to both the input lines of this second one giving B bar. Now A bar and B bar supplied to NAND and that is leading to your OR operation; both A bar and B bar is combined in a NAND operation with over bar that gives you an OR operation.

To get the same operation done; you can we could have also use the NOR gate. Like this is the NOR to get the NOT operation or inversion operation; this is the version of AND operation. Here again there are 2 NOR gates; the first one is receiving A as both inputs, second one is receiving B as both inputs; they are be giving A bar and B bar and which are supplied to a another NOR, output will be plane AND operation.

And the NOR operation both A and B are supplied to one NOR gate ah; output will be A plus B over bar. And if they are we are supplying that as both inputs to another NOR; then the output will plane OR. That is why NAND and NOR are often called universal logic gates because using any of them; we can have all other operations I have not showing XOR or XNOR operations, but you can think about how we can achieve that the same way combining a few NAND gates or a few NOR gates.

(Refer Slide Time: 40:03)



Binary codes: BCD

Decimal	Binary	BCD
0	0000 0000	0000 0000
1	0000 0001	0000 0001
2	0000 0010	0000 0010
3	0000 0011	0000 0011
4	0000 0100	0000 0100
5	0000 0101	0000 0101
6	0000 0110	0000 0110
7	0000 0111	0000 0111
8	0000 1000	0000 1000
9	0000 1001	0000 1001
10	0000 1010	0001 0000
11	0000 1011	0001 0001
12	0000 1100	0001 0010
13	0000 1101	0001 0011
14	0000 1110	0001 0100
15	0000 1111	0001 0101
16	0001 0000	0001 0110
17	0001 0001	0001 0111
18	0001 0010	0001 1000
19	0001 0011	0001 1001
20	0001 0100	0010 0000

Next we come to the binary code; while decimal numbers are supplied to an instrument, the instrument converge that is to binary. How that conversion physically is done? That we shall be discussing later on, but instead of before that we let us discuss about how it stores the number. And there comes this binary codes the first one we have is called BCD or Binary Coded Decimal; Binary Coded Decimal.

Here the idea is each digit of a decimal number is treated separately; like we are showing here the charts of decimal digits from 0 to 20 and corresponding binary numbers are also here. I am sure you can calculate this binary, but the third column you have the BCD representation. Here what we are going to do every number is going to be treated separately. Let us take a random; let us say our input is 73 in decimal, we can always get a binary equivalent of this, but here in BCD our objective is not to get the binary equivalent; rather we shall be treating 7 and 3 separately and each of these decimal digits will be supplied to a 4 bit counter 4 and corresponding binary version of each of these digits will be stored in that 4 bit.

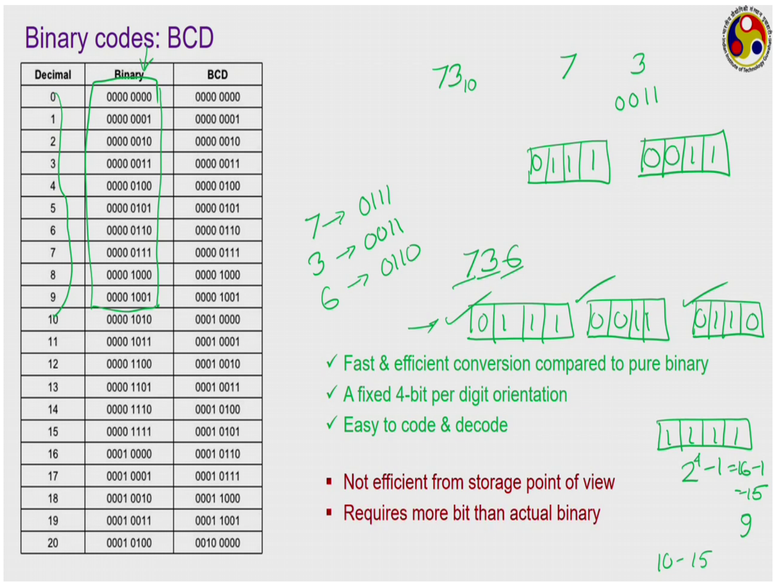
What I mean? Like here you have 3, now what is the binary equivalent of 3? Binary equivalent of 3 is 1 1 let us add couple of leading zeros. So, here we shall be providing a register; a 4 bit register which going to store this 4 digits 0 0 1 1, that is the binary equivalent of 3. And another 4 bit register we shall be coding we shall be storing the binary equivalent of 7; what is the binary equivalent of 7? It is 1 1 1; so a 0 in the first box. If suppose we are dealing with a number say 736 there are 3 digits for each of the digits a 4 bit register will be provided.

So, you will be having a sequence of 3 4 bit register is one, this is the second one for the second digit, this is the third one for the third digit. For 7 how it will be showing? 0 1 1 1, for the second one that is 3; 0 0 1 1 and for 6; 0 1 1 0; that means, as soon as you type one digit, say if you type 7 and immediately its binary equivalent, which is 0 1 1 1 will be stored in one register. Next as you type 3 its binary equivalent that is 0 0 1 1 will be stored in the next 4 bit register. And now when you type 6 its binary equivalent 0 1 1 0 will be stored in the next 4 bit register.

This way whenever you input one decimal digit; it will be converted to its binary version and stored in a 4 bit register. And this way whatever maybe the number of digits in your original decimal number same number of 4 bit registers will be provided to you. So, this is what we referred as BCD or Binary Coded Decimal; you have to be careful that this is different from that octal or hexadecimal representation or coding of binary. Here we are not at all storing the actual binary converted value of the original decimal; rather we are storing the binary conversion of each digit.

Like look at this example if we someone just reach read if some reading all these binary digits following the register then what he is having? He is having 0 1 1 1 0 0 1 1 0 1 1 0; now the binary equivalent for this one will be something else that is not going to be 736 in decimal. The decimal equivalent of this one is never going to be 736 because we have not at all converted 736 to its binary equivalent stored that, rather we have converted each of its digit this 7 this 3 and this 6 to their respective binary equivalents and each of them has been stored in a 4 bit register.

(Refer Slide Time: 44:23)



Binary codes: BCD

Decimal	Binary	BCD
0	0000 0000	0000 0000
1	0000 0001	0000 0001
2	0000 0010	0000 0010
3	0000 0011	0000 0011
4	0000 0100	0000 0100
5	0000 0101	0000 0101
6	0000 0110	0000 0110
7	0000 0111	0000 0111
8	0000 1000	0000 1000
9	0000 1001	0000 1001
10	0000 1010	0001 0000
11	0000 1011	0001 0001
12	0000 1100	0001 0010
13	0000 1101	0001 0011
14	0000 1110	0001 0100
15	0000 1111	0001 0101
16	0001 0000	0001 0110
17	0001 0001	0001 0111
18	0001 0010	0001 1000
19	0001 0011	0001 1001
20	0001 0100	0010 0000

- ✓ Fast & efficient conversion compared to pure binary
- ✓ A fixed 4-bit per digit orientation
- ✓ Easy to code & decode

- Not efficient from storage point of view
- Requires more bit than actual binary

So, this method of storing binary digits in BCD format is quite fast and efficient conversion compared to pure binary conversion. Because in pure binary conversion you have to go quite series of operation, but here we are dealing with only a few standard digits. And like 0 to 9 are the digits that are available to you and we know their binary equivalent.

So, our interest is only restricted to this particular zone and using know the binary equivalent for each of them; we can immediately identify the BCD version corresponding to each of the digits and so the final BCD version of the original decimal number. Then we can have a fixed 4 digit orientation a fixed 4 bit for digit orientation which is quite easy for the computer to deal with.

And we can also code and decode it easily like the coding we have also see[n]- already seen. Similarly when we are given with a set of BCDs suppose it is mentioned that your BCD code is 0 1 0 1 0 1 1 1 0 0 0 0 and 1 0 0 1. Then we know that we are having 4 registers each comprising of 4 bits and therefore, there are the digital sorry the decimal number that we are talking about that has 4 digits in it.

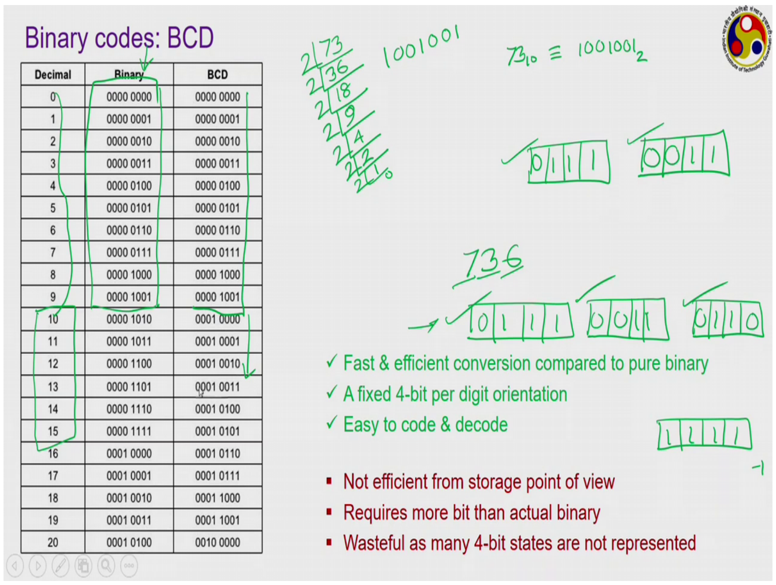
So, we have to identify the decimal equivalent for each of these registers. So, 0 1 0 1 refers to what? Refers to 5 0 1 1 1 refers to 7; 0 0 0 0 refers to 0 and 1 0 0 1 refers to 9. So, the corresponding decimal digit that we talking about is 5 7 0 9; whatever is the number of decimal number of digits in a decimal number; the same will be the number of 4 bit registers required here.

But it is not very efficient from storage point of view because remember in a 4 bit register what is the largest number that we can store? Largest binary number this is your 4 bit register then the largest binary you can store is 1 1 1 1; which is 15 or we can also calculate writing as 2 to the power 4 minus 1, which is 16 minus 1; that is 15.

So, the largest decimal number that we can store in a 4 bit register is 15, but here the largest what we are storing? That is only 9 because we are restricting our self to a single digit which is 0 to 9. So, that is all the numbers starting from 10 to 15; all these 6 numbers which could have been stored in a 4 bit register we are not at all doing that.

So, we are essentially losing some storage potential and we are not using the storage quite efficient. It requires definitely more bit than the actual binary like if we say the number that is started with 73; if we want to calculate the binary equivalent for 73, what we are going to get?

(Refer Slide Time: 47:17)



Binary codes: BCD

Decimal	Binary	BCD
0	0000 0000	0000 0000
1	0000 0001	0000 0001
2	0000 0010	0000 0010
3	0000 0011	0000 0011
4	0000 0100	0000 0100
5	0000 0101	0000 0101
6	0000 0110	0000 0110
7	0000 0111	0000 0111
8	0000 1000	0000 1000
9	0000 1001	0000 1001
10	0000 1010	0001 0000
11	0000 1011	0001 0001
12	0000 1100	0001 0010
13	0000 1101	0001 0011
14	0000 1110	0001 0100
15	0000 1111	0001 0101
16	0001 0000	0001 0110
17	0001 0001	0001 0111
18	0001 0010	0001 1000
19	0001 0011	0001 1001
20	0001 0100	0010 0000

✓ Fast & efficient conversion compared to pure binary
✓ A fixed 4-bit per digit orientation
✓ Easy to code & decode

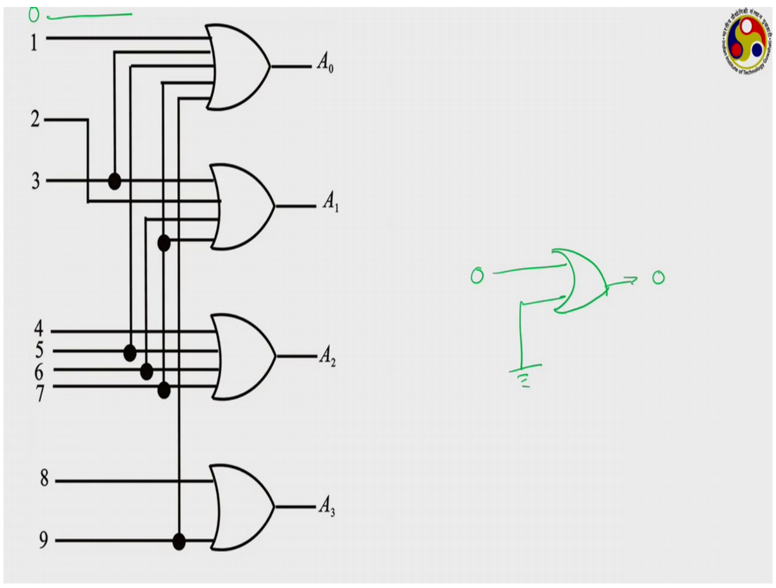
- Not efficient from storage point of view
- Requires more bit than actual binary
- Wasteful as many 4-bit states are not represented

So, divide by 2; let me make some space. So, dividing 73 by 2 we going to have 36 and 1 as remainder, divide 36 by 2 we are having 18, 0 is remainder; divide this by 2 we are getting 9 and again 0 is remainder. And now dividing 9 by 2, we are having 4; 1 as remainder; by 2 we are having 2 again 0 is remainder dividing by 2 we are having 1.

So, 0 and finally, dividing this 2 we do not we have 1. So, how many binary digits we have then; these are the binary equivalent that is this 73 in decimal is equivalent to 1 0 0 1 0 0 1 in binary, this is the exact representation and we need only 7 digits. However, in this BCD format we needed 4 digits for 7 and 4 digits for 3 total 8 bits. So, 4 plus 4 total 8 bits we require whereas, had we stored original binary would have needed only 7 it is 1 less.

So, we need more bits than the actual binary and ways full as many of the 4 bits states are not present represented as I mentioned 10 to 15 this entire block of 6 decimal numbers which could have been represented in the 4 bit registers are not possible; just check the BCD column. Up to this we are using only the only 1 register, but from this point onwards we have to use 2 registers. So, that way BCD may not be very efficient for storage point of view, but quite easy to code and decode and it has a fast conversion because we are dealing with each digit separately.

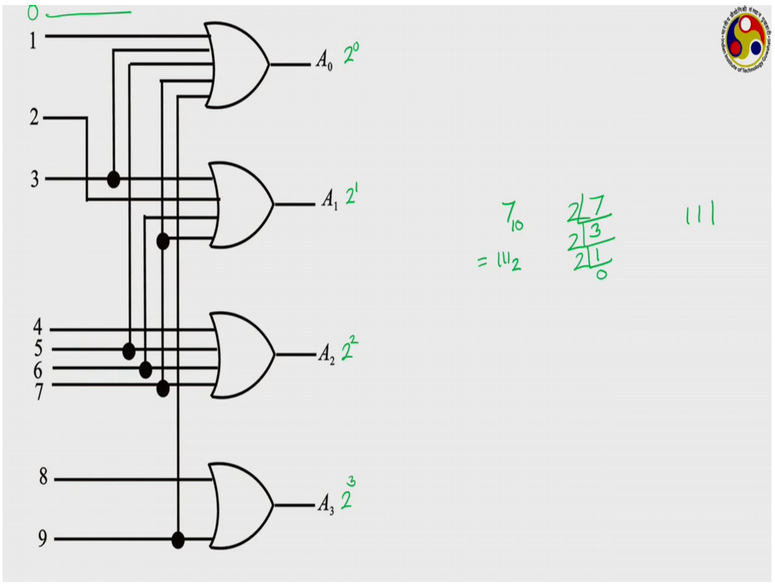
(Refer Slide Time: 49:03)



Now, how we do this in a circuit? That we can easily do by combining this 4 or gates; these are 4 all logical gates and we are having all this. All the inputs are the decimal 1 to 9; 0 is not shown here, but actually we have 10 input 0 should also be there. 0 when 0 is the input we do not need to connect that to anything else; like we can easily have an OR gate where which 0 is connected and the other side is grounded; so that which is also giving a 0 signal and hence its output is always going to be 0.

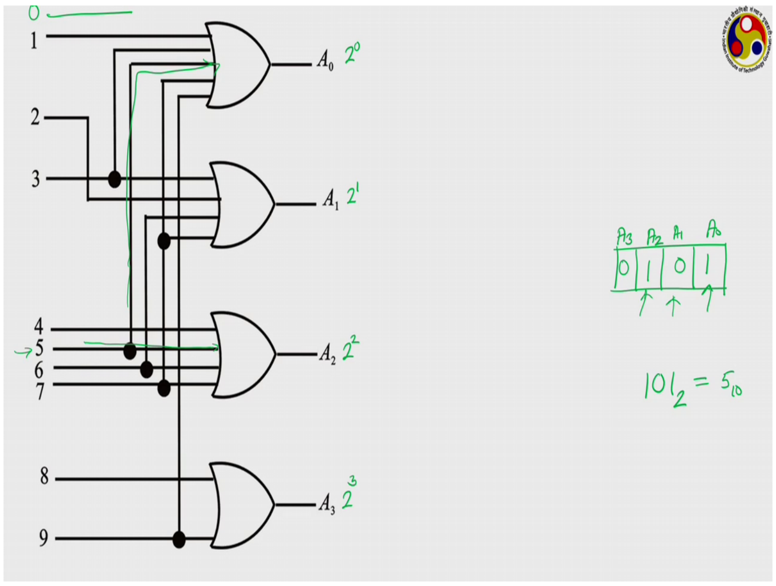
So, we do not need to talk about 0 anymore; 1 to 9 are the digits. Let us pick up any one of them; let us say here this on the output side let me tell first A naught to A 3 refers to the 4 digits. A naught is the least significant bit A 3 is the most significant bit.

(Refer Slide Time: 50:01)



Or in other way A naught corresponds 2 to the power 0 level, A 1 2 to the power 1 level, A 2 corresponds to 2 to the power 2 level and A 3 2 to the power 3 level. Let us pick up any number; let us say 7 in decimal what is its binary equivalent for 7? So, if we divide 7; we are left with 3 with 1 as remainder dividing by once more; 1 is reminder and dividing it by this here 1. So, the binary equivalent for 7 is 111; like this.

(Refer Slide Time: 50:41)



So, what we want? We want in the this is the 4 bit register that we have when we want to store 7 or we want to store the BCD equivalent of 7; then you this is your A naught, this is your A 1, A 2 and A 3. We need to have operations such that A naught gives you a 1, A 1 gives you 1, A 2 gives 1, but A 3 gives a 0; this what we have to store.

Here each of these OR gates are connected to one of the bits. So, what we are doing here; look at A naught, A naught has 5 different inputs and one of this input is 7; this is 7 follow this, this is being going to 1. So, whenever you are putting 7 a positive signal is going to this A naught and when others are 0, but still you are getting a 1 as the output giving you this 1. Then other end of 7; 7 actually where it is connected, another end is connected to this A 1; which gives you this A 1 to be 1 and the other is connected to this A 2; giving you A 2.

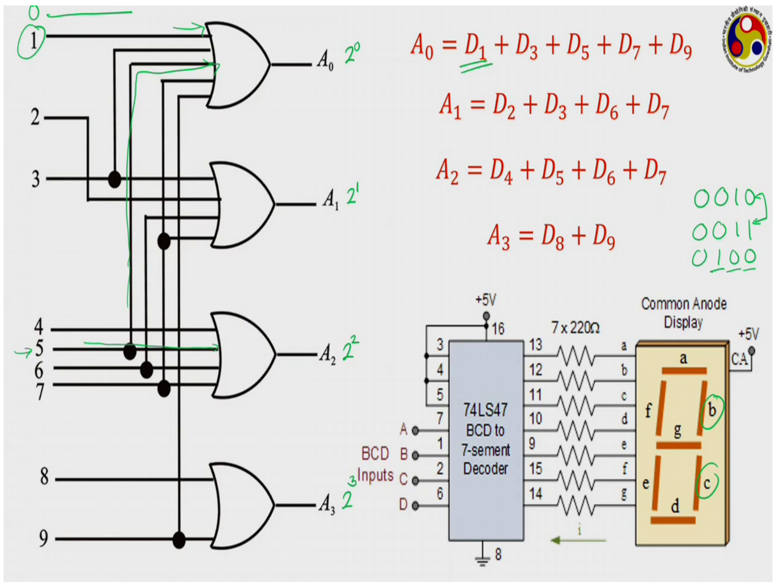
So, to represent 7; we want this 3 digits to be this 3 bits to be 1 and that is why this 7 is connected to this 3 OR, but not with A 3. Similarly suppose you want to get the digital representation of; you want to digitally represent 101 what is 101? This is in decimal this a binary how we can find the decimal equivalent of this?

When to get 1 into 2 to the power 0 plus 0 into 2 to the power 1 plus 1 into 2 to the power 2; giving you 5, its decimal equivalent is 5. Now if we want to store 5 in a register like this; so what you will it going to do? You need to have A 1 to be 0, A 3 to be 0 and A 1; sorry A naught and A 2 to be equal to 1; that means, the 5 input that you are giving us

5 that needs to be connected only to 2 of the; 2 of the or gates. Now where 5 is connected?

Let us see just this 5 state goes to 1; sorry A 2 and the other connection from 5 this is the one that is going sorry and the if you follow the other connection that is going to this A naught; there by giving you A 2 and A naught to be equal to 1 others to be equal to 0.

(Refer Slide Time: 53:25)



So, this way we can very easily get the BCD operation done or BCD conversion done for each of the digits using this 4 OR gates. This can be the mathematical equivalent A naught is connected to 5 different inputs D 1, D 3 here D 1 refers to the decimal input 1; D 3 refers to the decimal input 3.

So, A naught is connected to 1 3 5 7 and 9 inputs; you can think about your mobile phone or maybe a computer keyboard; whenever you are pressing the 1, that 1 actually is connected to the a some electrical line whenever you are pressing 1; it is switching on that line and an electrical signal is going in the form of this D 1. And accordingly it is a activated as this is A naught switch and it is giving A naught 1 in the least significant bit in the A naught position.

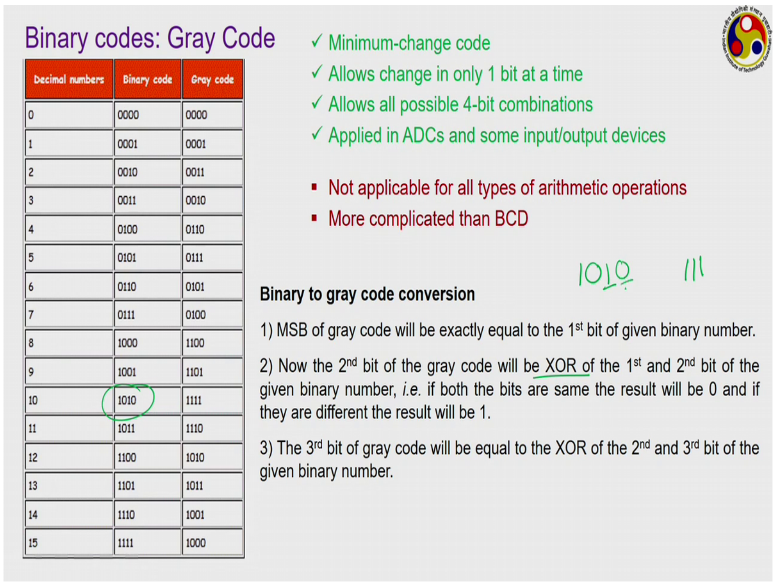
But this is not connected like follow this 1; it is connected only through this A naught because you need other 3 bits to be 0. So, you are having this 0 0 0 1 in the 4 bit register; A 1 is having this operation; D 2 it is connected to 2 3 6 and 7, A 2 is connected to 4 5 6 and 7 and A 3 is connected to 8 and 9. Very simple operation just 4 OR gates and they have; they are connected to corresponding decimal input points and you have the BCD converted conversion as the output in the 4 bit register.

This BCD conversion are used for several different purposes like this is one of the representation, where it is actually connected to a 7 segment decoder. Something like that digital display that we get on clocks here we can see there are 7 line on this a to f sorry a to g; there are 7 lines and depending upon what you want like when you want an 8 to be represented, then all these 7 signal should be on. So, accordingly we are getting the signals, but suppose if you want just one to be represented; then you need only b and c; you do not need others.

So, in that case only b and c will be getting some signals other will be switched off; according we can get this digital display done. Now binary coded decimal is a quite standard practice, but one problem here with binary coded decimal is that suppose you are dealing with 2; once you are storing 2, you are having 0 0 1 0. And now you want to store 3; you have 0 0 1 1 and next step when you want to store 4; then the third digit comes into play. So, you have 0 1 0 0 then between these 2; how many bits are changing? Just 1.

But in this case how many bits are changing? We are having all these 3 bits are changing that is simultaneously we have to do 3 operation; for just change of one digit. So, that sometimes can be alleviated by using the next one which is called the gray code.

(Refer Slide Time: 56:23)



Binary codes: Gray Code

Decimal numbers	Binary code	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

- ✓ Minimum-change code
- ✓ Allows change in only 1 bit at a time
- ✓ Allows all possible 4-bit combinations
- ✓ Applied in ADCs and some input/output devices

- Not applicable for all types of arithmetic operations
- More complicated than BCD

1010 111

Binary to gray code conversion

1) MSB of gray code will be exactly equal to the 1st bit of given binary number.

2) Now the 2nd bit of the gray code will be XOR of the 1st and 2nd bit of the given binary number, *i.e.* if both the bits are same the result will be 0 and if they are different the result will be 1.

3) The 3rd bit of gray code will be equal to the XOR of the 2nd and 3rd bit of the given binary number.

Grey code has a different representation; here you can see the binary equivalent in a (Refer Time: 56:34) we are talking about 4 bit register. So, 0 to 15 the decimal numbers which can be stored in a 4 bit register because 15 has a binary equivalent of 1 1 1 1. And logically we can store this is this has the largest number largest decimal number in the 4 bit register.

Think about BCD there the largest decimal we can store is this 9, but in case of gray scale we are storing all 0 to 15 that is all the 16 digits we are storing. But we are not at all following the binary format you are following a different format like shown in this table; I shall be coming back to this number format very shortly.

Here the concept is minimum change that is allows only change, change only 1 bit at a time. Like compare this between these 2; here there is only change in the second bit, I am talking about from the LSB side that is only one change in the second one; between these 2 there is only one change, change in the first one.

Between these 2 there is only change in the second bit that is between successive numbers that is only single change in the bit which is the advantage of gray scale you need to have less number of operations. And it also allows all possible 15; 4 bit combination going from 0 to 15 in decimal scale. It is found application in different ADCs means Analogue to Digital Converter; something we shall be talking in the next lecture and also in some input output devices.

However it is not for applicable for all types of arithmetic operations is a operation operability because of the different versional binary code is a very standard representation; BCD is a very standard representation, very easy to code and decode gray scale code is not that. Just by seeing this number if we are talking about 0 1 0 0 in binary; we can easily get the corresponding decimal version, but if you get this particular number in gray code; then you have to spend some more time to get this that is why we cannot perform all possible arithmetic operation on this and it is more complicated than BCD.

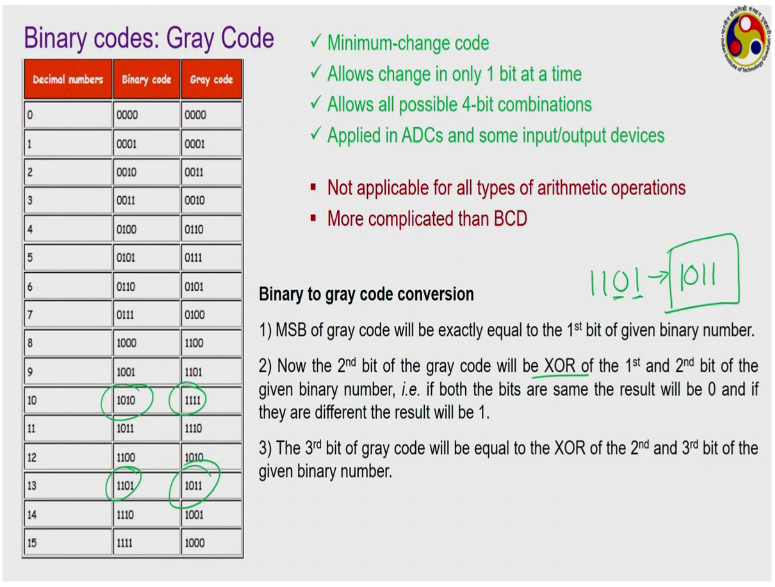
Now, the question is how we are converting this binary to gray code conversion? Let us take one example let us take say this one, 10 whose binary equivalent is 1 0 1 0 is binary equivalent records 4 bits; gray code also have to use all the 4 bits. First principle is that most significant bit of the gray code will be exactly equal to the first bit of the given binary. Like here the binary that we have selected is 1 0 0; sorry 1 0 1 0 most significant bit is the first one which is 1.

So, for the gray code also it will be 1; next principle here we have to compare the first and second bit of the binary and perform an XOR operation on this. In what your first and second bit; 1 and 0. So, 1 and 0 if we perform an OR operation, we are going to get 1, XOR operation means we wants to get a 0. We have to perform sorry sorry sorry we are talking about an XOR operation; not NOR operation sorry this an XOR operation. XOR means if both inputs are of the same kind it output will be 0; if both inputs are different, output will be 1; here both inputs are of different kind 1 and 0; your output will be 0.

I am making a mistake again; in XOR operation if both inputs are different then result will be 1, if both inputs are same your output will be 0. Here your inputs are 1 and 0; so they are different, so your output has to be 1. In the for finding the next one we have to now compare the second and third of the binary. So, 0 and 1 again perform the XOR operation; again they are of different kind which will lead to 1. Now though find and we have to continue this way let to find the fourth bit now; again we have to compare the third and fourth bit of the binary.

Again they are different. So, it will be 1 and this is the BCD equivalent if we take another example.

(Refer Slide Time: 60:41)



Binary codes: Gray Code

Decimal numbers	Binary code	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

- ✓ Minimum-change code
- ✓ Allows change in only 1 bit at a time
- ✓ Allows all possible 4-bit combinations
- ✓ Applied in ADCs and some input/output devices

- Not applicable for all types of arithmetic operations
- More complicated than BCD

Binary to gray code conversion $1101 \rightarrow \boxed{1011}$

1) MSB of gray code will be exactly equal to the 1st bit of given binary number.

2) Now the 2nd bit of the gray code will be XOR of the 1st and 2nd bit of the given binary number, *i.e.* if both the bits are same the result will be 0 and if they are different the result will be 1.

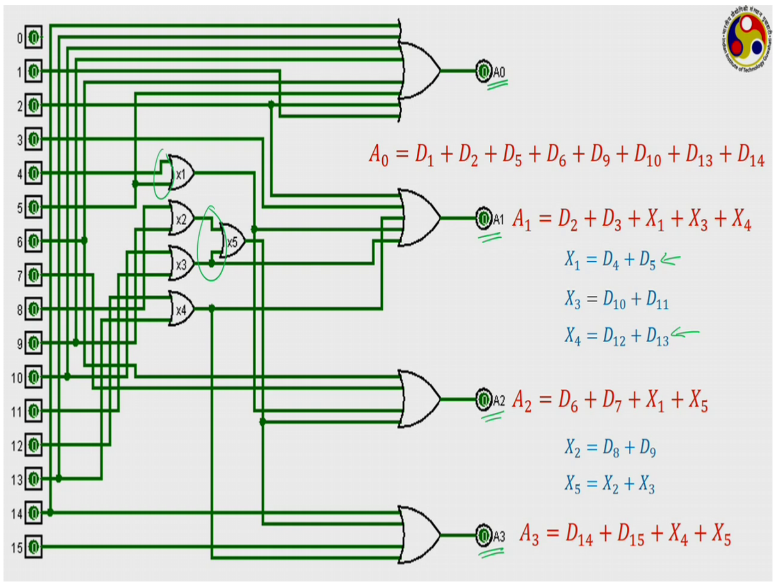
3) The 3rd bit of gray code will be equal to the XOR of the 2nd and 3rd bit of the given binary number.

Let us take 1 1 0 1 which is this one; 13. Now the first one in gray code will be what? Whatever is your most significant bit in binary, the same is your gray which is 1. To perform the second bit we have to perform the XOR operation between the first and second; both are 1 that is they are identical; so, your output will be 0.

Now to get the third bit we have to perform XOR bit in second and third, they are 1 and 0 they are different; so the output will be 1. And then between 0 and 1; you have to perform another XOR bit in 0 and 1 they are different; so output will be 1. So, this is the gray code version; this just shown here, this way we can convert binary to gray code. Yes more complicated, but if we think logically we can do it; remember the first bit on the gray code will be the same as the most significant bit of binary then for every subsequent bit we have to compare the we have to perform XOR operation.

That is to find the second one we have to second one in gray we have to find the XOR between first and second in binary. To find the third in gray we have to find the XOR of second and third in binary and to find the fourth in gray we have to do the XOR of third and fourth in binary. And XOR means something we has made a mistake while talking; XOR when both inputs are same your result will be 0, if both inputs are different result will be 1.

(Refer Slide Time: 62:07)


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
x1
x2
x3
x4
x5
A0
A1
A2
A3
$A_0 = D_1 + D_2 + D_5 + D_6 + D_9 + D_{10} + D_{13} + D_{14}$
$A_1 = D_2 + D_3 + X_1 + X_3 + X_4$
$X_1 = D_4 + D_5$
$X_3 = D_{10} + D_{11}$
$X_4 = D_{12} + D_{13}$
$A_2 = D_6 + D_7 + X_1 + X_5$
$X_2 = D_8 + D_9$
$X_5 = X_2 + X_3$
$A_3 = D_{14} + D_{15} + X_4 + X_5$

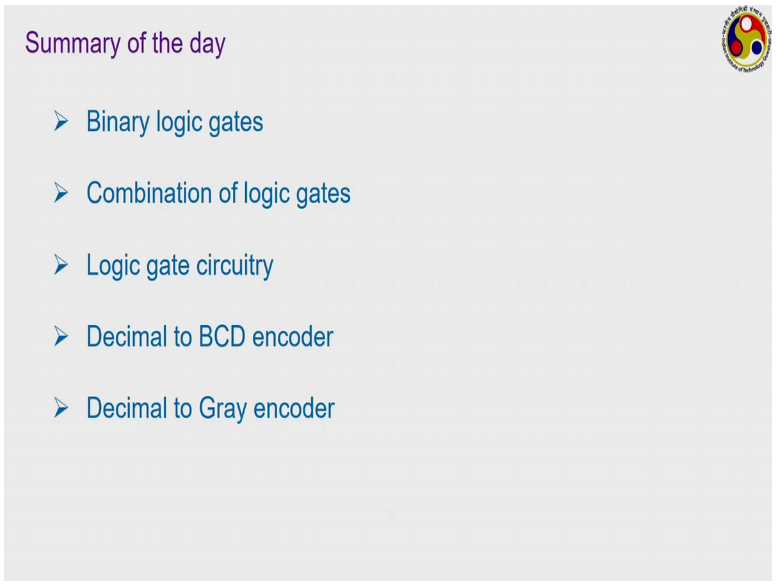
This is the possible circuit; much more complicated one you please check the circuit properly. There are 15 possible operations starting from 0 to 15, there are 15 possible inputs that you can give and we are combining them with 4 OR gates at the output level.

Here again A naught refers to the first bit that is the right most bit, A 1 refers to the next one, A 2 is the next one and A 3 refers to the leftmost bit which will correspond to the most significant bit of your binary equivalent and also there are 5 OR gates in between. So, A naught can be written as combination of this it is directly connected to 1 2 5 6 9 10 13 and 14. A 1 is connected to 2 3 and all series connected to this in between OR gates 1 3 and 4.

X 1 if we look at the diagram X 1 is having 2 inputs what are these? One input is 5 other input is 4. So, X 1 is D 4 plus D 5; X 3 is again connected to 2 your X 3 is having 2 inputs and your inputs are if you follow properly one it could 10 other is 11. And X 4 is again having 2 inputs again and the inputs are 12 and 13; so that is what we have here and we are combining them in A 1. A 2 is connected to 6 and 7 and also X 1 and X 5.

What is X 1? X 1 is having input as D 4 and D 5 and X 5 is having input as X 2 and X 3; it is taking input from this two. Now X 2 is having 2 inputs, X 3 is having 2 inputs and finally, the a 3 the leftmost bit it is connected to 14, 15 and also X 4 and X 5. This way we can quite similar to BCD we can form also the gray code version of this. So, this takes us to the end of today’s lecture.

(Refer Slide Time: 64:03)



Summary of the day

- Binary logic gates
- Combination of logic gates
- Logic gate circuitry
- Decimal to BCD encoder
- Decimal to Gray encoder

If we want to summarise we have discussed about the binary logic gates today. We have discussed in detail about each of them and also hopefully you understood how to do this operation; please practice some more examples.

We have discussed about how to combine different logic gates to get different kinds of inputs for any number of input different number of outputs for any number of input signals. Then we have briefly discussed about the circuitry for AND, OR and NOT gates and how to combine that to get NAND and NOR operation. Also we have talked about the universal gates that is NAND and NOR; then we have talked about the decimal to BDC encoder and gray encoder.

In the next lecture, we shall be seeing about alphanumeric coding and also we shall be discussing about the decimal to binary conversion or digital to analogue; analogue to digital conversion and also the opposite that is digital to analogue conversion. So, that is it for the day thanks for your attention; please try these problems before you start hearing the next lecture.

Thank you.