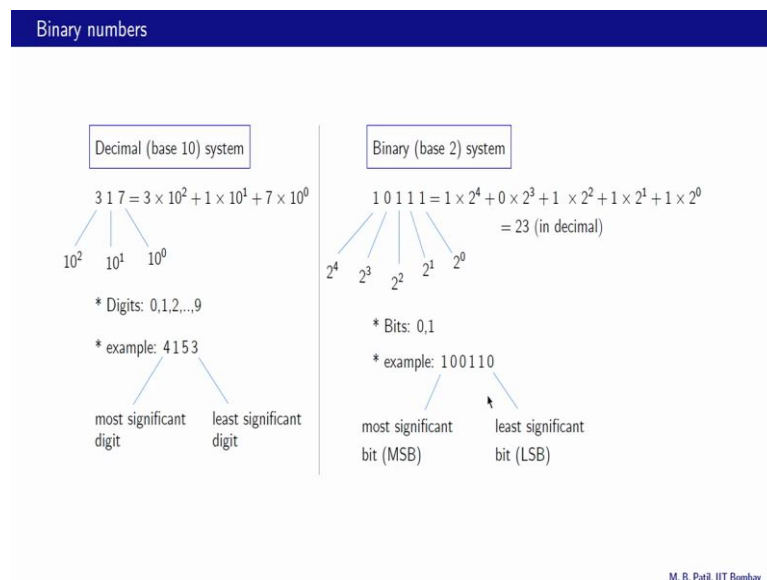


Basic Electronics
Prof. Mahesh Patil
Department of Electrical Engineering
Indian Institute of Technology, Bombay

Lecture – 58
Combinatorial circuits

Welcome back to Basic Electronics. In this lecture we will start with the binary or base to number system, we will see how addition of binary numbers can be implemented with logic gates. Finally, we will look at implementation of a logical function using only NAND or only NOT gates. So, let us begin.

(Refer Slide Time: 00:40)



We now want to discuss binary numbers that is numbers in the base 2 system, but before we do that let us look at our decimal system and here is an example 317 and let us ask this question why is the value of this number 317. And the answer is this 3 gets A weight of 10 raise to 2 this 1 gets A weight of 10 raised to 1 and this 7 gets the weight of 10 raise to 0. So, that is 3 times 100 plus 1 times 10 plus 7 times 1 and that is how it becomes 317. And these numbers are called the digits and in the base 10 system the digits of course can vary from 0 to 9, and the leftmost digit is called the most significant digit and the rightmost digit is called the least significant digit. Here is an example 4153 here the most significant digit is 4 and the least significant digit is 3.

Now, let us look at the binary system. So, here is an example of the binary or base 2 system and we notice that in each position we have only 1 or 0 and the weight that is assigned to each position is a power of 2 and this number for example, is 1 times 2 raise to 4 plus 0 times 2 raise to 3 and so on and if we calculate that it turns out to be the number 23 in our decimal system.

So, in the decimal system we had digits, here we have bits the digits could go from 0 to 9, the bits can only have 2 values 0 or 1 here is an example of A binary number. The leftmost bit is called the most significant bit or the MSB in this case it is 1 and the rightmost bit is called the least significant bit or LSB in this case it is 0. Why is this called the most significant bit because it gets the largest weight, similarly the LSB gets the smallest weight.

(Refer Slide Time: 03:18)

Addition of binary numbers

Decimal (base 10) system						Binary (base 2) system							
	10^4	10^3	10^2	10^1	10^0	weight		2^4	2^3	2^2	2^1	2^0	weight
		3	1	7	9	first number		1	0	1	1	1	first number (dec. 11)
+		8	0	1	5	second number	+	1	1	1	0	0	second number (dec. 14)
						carry		1	1	1			carry
	1			1				1	1	0	0	1	sum (dec. 25)
	1	1	1	9	4	sum							

* $0 + 1 = 1 + 0 = 1 \rightarrow S = 1, C = 0$

* $1 + 1 = 10 \text{ (dec. 2)} \rightarrow S = 0, C = 1$

* $1 + 1 + 1 = 11 \text{ (dec. 3)} \rightarrow S = 1, C = 1$

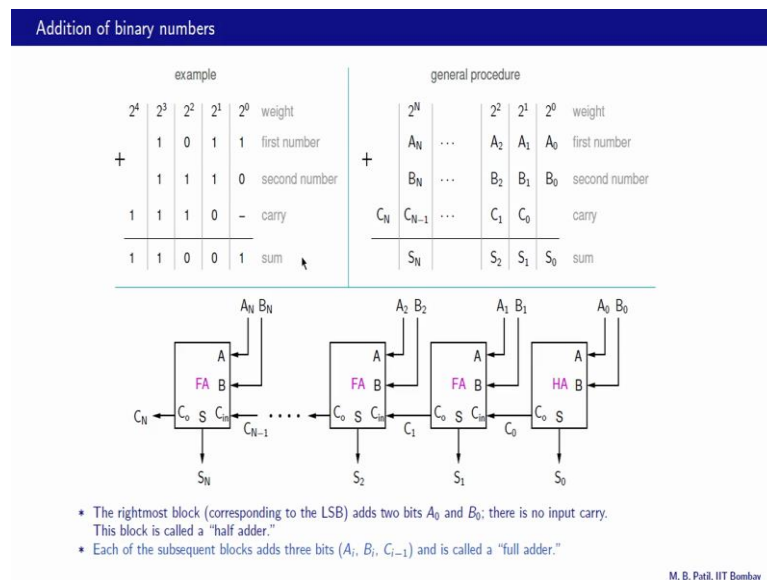
M. B. Patil, IIT Bombay

We will now discuss addition of binary numbers, but let us start with the decimal or base 10 systems first, because we are already familiar with addition of numbers in this system and then we will see the parallels between the base 10 system and base 2 system. So, here is an example 3 1 7 9 plus 8 0 1 5, how do we go about adding these two numbers? We start with the least significant digits that is this column here we add this 9 and 5 now each one of these gets the rate of 10 raise to 0. So, the sum here is 14 we take 1 as carry now this 1 has a special significance, it is weight is 10 raise to 1 which is 10 times higher than this weight.

Next, we add these 2 and we also add the carry and so on in this case the carry 0, so we only need to add 1 and 0 etcetera and that is how we end up with this sum. Let us now look at the binary or base 2 system and before we look at addition of 2 numbers let us look at these additions of bits here. If you are adding 0 and 1 what do we get the result is one; that means, the sum bit is 1 and the carry bit is 0. If we are adding 1 and 1 then the result is decimal 2, decimal 2 is the same as binary 1 0 the right bit here is the sum bit which is 0 and the left bit here is the carry bit that is 1. What about 1 plus 1 plus 1? That is decimal 3 which is binary 11. So, in this case the sum bit is 1, S and the carry bit is also 1 that is C is equal to 1. So, using these let us now try to understand this addition example.

So, we want to add two numbers 1 0 1 1 which is decimal 11 and 1 1 1 0 which is decimal 14, where do we begin? We begin with the least significant bit. So, we add 1 and 0 we get a sum bit 1 and no carry bit, then we add this 1 and 1, and 1 and 1 gives us 1 0, so sum bit 0 carry bit 1, sum bit 0 carry bit 1. Again we add 1 and 1 here sum bit 0 carry bit 1, now we add 1 1 and 1. So, we get decimal 3 and that is sum bit 1 and carry bit 1 and now of course, in this column there is no bit over there so only the carry. So, that is 1 here and if we evaluate this number that turns out to be decimal 25 as we would expect.

(Refer Slide Time: 06:53)



Let us now generalize the binary addition that we saw in the last slide here is our example and that will help us to see what will happen in the general case, all right. So,

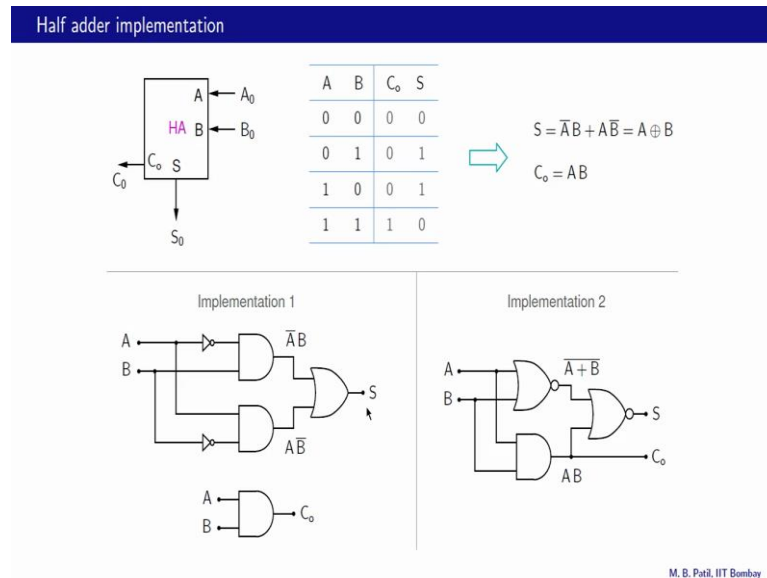
here is the problem we have two numbers - first number and second number binary numbers and we want to add these two number. There are $N + 1$ bits in each number and we have A_0 as the least significant bit in the first number and A_N has the MSB, and the weights go from 2^0 to 2^N . These are the carries this C_0 is coming from column number 0, C_1 is coming from column number 1 and so on, and this is our final sum S_0 gets the weight of 2^0 , S_1 gets a weight of 2^1 and so on.

Here is the block diagram for binary addition and this first block performs this operation; that means, it takes A_0 and B_0 adds the two and gives us a sum bit S_0 and the carryout bit C_0 here and we have called that S_0 so this is S_0 and we have called the carryout as C_0 so that is C_0 .

Now, this block is called half adder because it does not have a carry here and all other blocks all of these are called full adders because they also include a carry. What about this full adder? Let us take this as an example. So, we have 2 bits coming from the 2 numbers we have a carry in coming from the previous stage and as output this full adder must produce a sum bit and the carryout bit. So, that is what we see over here, these are the 2 bits coming from the numbers this is the carry in, this is the output sum bit and this is the carryout

So, in this case for example, we have A_1 and B_1 , $A_1 B_1$ there. C_0 is coming as the carry in from the previous stage and the sum bit is S_1 , S_1 and the carryout is C_1 , C_1 there and so on. So, let us summarize the rightmost block corresponding to the LSB adds 2 bits A_0 and B_0 there, there is no input carry this block is called a half adder and each of the subsequent blocks adds 3 bits A_i , B_i and C_{i-1} coming from the previous stage and these blocks are called full adders.

(Refer Slide Time: 10:03)

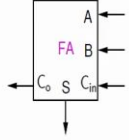


Let us now look at the implementation of a half adder with gates. So, let us start with the truth table. So, there are 2 tables. In fact, here one for the sum bit and one for the carryout bit when A and B are both 0 then the sum bit is 0 carryout is also 0, when A B are 0 1 or 1 0 then the sum bit is 1 carryout is 0 and when A B are both 1 then the sum bit is 0 carryout is 1, all right. So, these are the functions that we need to implement.

So, this S turns out to be A bar B plus A B bar, this term is A bar B and this term is A B bar and that is essentially the XOR operation and the carryout function is one only for this condition, therefore, that is A and B. So, here is how the S and C o functions can be implemented. C o of course, is just and of A and B as shown here and to get S we need to generate A bar and B bar then A bar B and A B bar and then finally, put them through OR gate to get S there is A better implementation and that requires A smaller number of gates and you should take this as homework and verify that S is indeed A bar B plus A B bar in this case.

(Refer Slide Time: 11:46)

Full adder implementation



A	B	C _{in}	C _o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

C _{in} \ AB	00	01	11	10
S:	0	0	1	0
	1	1	0	1

$$S = \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + \bar{A}\bar{B}C_{in} + ABC_{in}$$

C _{in} \ AB	00	01	11	10
C _o :	0	0	1	0
	1	0	1	1

$$C_o = AB + BC_{in} + AC_{in}$$

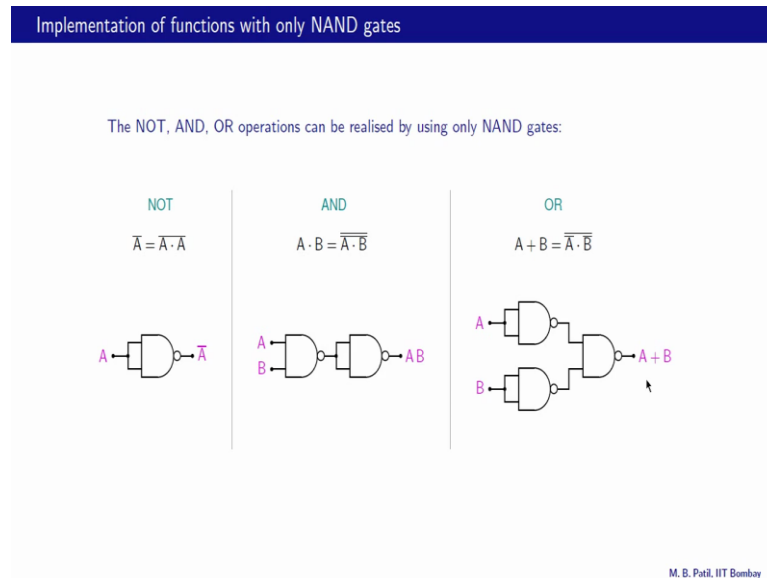
M. B. Patil, IIT Bombay

Let us now look at how a full adder can be implemented. It has 3 inputs A B and C in the carry in coming from the previous stage and it has 2 outputs the sum bit and the carryout or C o bit. So, let us construct the truth tables for S and C o, how do we do that? All we need to do is to look at the number of 1's in each row in this case there is no 1 so therefore, the addition of these 3 will give us the binary number 0 0 that is what we have here. In this case and this case and this case we have only 1 1, so the addition of these 3 that is A and B and C in will give us the binary number 0 1 and that is what we see over here.

In this row we have two 1's, in this row also we have two 1's and in this row also we have two 1's and the addition of A B and C in that case will be binary 1 0 that is decimal 2. So, that is what we see over here. In this last case we have A equal to 1, B equal to 1 and C in equal to 1 and when we add these we get decimal 3 or binary 11 and that is what we see over here. So, now, we have the tables for S and C o and let us now see how they can be implemented.

So, here is the k map for the sum bit and unfortunately all of these 1's are isolated they cannot be combined with any neighbors and therefore, we have 4 minterms given by this expression here, what about C o? That is the k map for the carryout bit, in this case there is some minimization possible we have four 1's and we can cover these four 1's with 3 rectangles and therefore, we have 3 terms each term having 2 variables.

(Refer Slide Time: 14:11)



Let us now look at implementation of logical functions with only NAND gates. It turns out that the NOT, AND, OR operations can be realized by using only NAND gates and let us look at these one by one starting with the NOT operation. So, here is the implementation we have a NAND gate here, we connect the 2 inputs together and connect our input variable A over there. So, what do we have at the output? We have A dot A bar and since A dot A is nothing, but A we get A bar at the output.

Here is the implementation of the AND operation and let us see how it works. We have one NAND gate here. So, at this point we have A B bar and then we have connected this inverter. So, therefore, we get at the output A B bar bar of that and that is nothing but A B so that is the AND operation. And here is the OR operation this is an inverter, the same as this one. So, we have A bar here, B bar here and then we have A bar and B bar and the NOT of that like that and that by De Morgan's theorem is nothing but A plus B. So, that is our OR implementation.

(Refer Slide Time: 15:43)

Implementation of functions with only NAND gates

Implement $Y = AB + BC\bar{D} + \bar{A}D$ using only NAND gates.

$$Y = \overline{\overline{AB} \cdot \overline{BC\bar{D}} \cdot \overline{\bar{A}D}}$$

$\bar{A} = \overline{A \cdot A}$
 $A \cdot B = \overline{\overline{A \cdot B}}$
 $A + B = \overline{\bar{A} \cdot \bar{B}}$

M. B. Patil, IIT Bombay

Let us take up this exercise implement Y equal to $A B$ plus $B C D$ bar plus A bar D using only NAND gates and as we have seen in the last slide this is how we can implement the NOT, AND and OR operations with NAND gates.

So, for this function what is our step number 1? Step number 1 is to implement this operation and let us assume that we have gates available to us, NAND gates available to us with 2 or 3 inputs. So, this is similar to this A plus B operation here except we now have A plus B plus C because we have 3 terms and that would be A bar dot B bar dot C bar and the bar of the whole thing, all right. So, let us write this in this format like that. So, this $A B$ can be treated as A and since we have A bar here we have $A B$ bar and the same thing applies to these terms as well so that is our first step and that is the implementation of the first step. So, we have $A B$ bar here $B C D$ bar bar of that and A bar D bar of that and all of those are applied to this 3 input NAND gate that should give us our output Y and now we need to figure out how to generate each of these.

How do we obtain $A B$ bar? That is in fact nothing but the NAND operation itself and therefore, all we need to do is to connect 2 input NAND gates here with inputs A and B like that, what about this term? We have $B C D$ bar and the bar of that. So, this can be achieved using a 3 input NAND gate with inputs $B C$ and D bar and D bar can be obtained by using this equation. So, we use a NAND gate with inputs D and D .

So, that is B C and D bar and D bar has been obtained with this NAND gate by connecting these 2 inputs together and then connecting that to D and this term can also be implemented using a NAND gate, 2 input NAND gate with inputs A bar and D like that. So, this is A bar and this is D and again A bar has been obtained from A using this NAND gate. So, this is our overall implementation of the logical function Y using only NAND gates.

(Refer Slide Time: 18:42)

Implementation of functions with only NAND gates

Implement $Y = A + B + C$ using only 2-input NAND gates.

$$Y = (A + B) + C$$

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

$$= \overline{\overline{A + B} \cdot \overline{C}}$$

$\overline{A} = \overline{A \cdot A}$
 $A \cdot B = \overline{\overline{A \cdot B}}$
 $A + B = \overline{\overline{A} \cdot \overline{B}}$

M. B. Patil, IIT Bombay

Next example implement Y equal to A plus B plus C using only 2 input NAND gates. So, now, we have this restriction we can use NAND gates with 2 inputs only all right and here are our equations once again for implementation of the NOT, AND and OR operations using NAND gates. Here is our step number 1 and what we have done here is we have grouped this A plus B together. So, we are treating A plus B as 1 entity now and let us call that x for example. So, we have now Y equal to X plus C and how do we implement X plus C using NAND gates we can refer to this equation here. So, X plus C would be X bar dot C bar and bar of that. So, that is how we can implement this first step. So, C bar and then A plus B bar and the NAND of that.

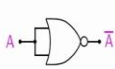
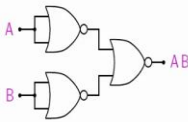
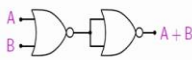
Now, let us see how to generate each one of these C bar is easy to generate we just need to use this equation. So, we use a NAND gate connect the 2 inputs together and connect that to C to get C bar. What about this operation? This is the NOR operation and it is not there in this list. So, what we will do is we will generate A plus B bar from A plus B

using the not operation like that. So, we have a NAND gate with inputs tied together we apply A plus B here and then we get A plus B bar over here at the output and now we know how to generate A plus B using this last equation and that completes our implementation.

(Refer Slide Time: 20:52)

Implementation of functions with only NOR gates

The NOT, AND, OR operations can be realised by using only NOR gates:

<p>NOT</p> $\bar{A} = \overline{A + A}$ 	<p>AND</p> $A \cdot B = \overline{\overline{A} + \overline{B}}$ 	<p>OR</p> $A + B = \overline{\overline{A + B}}$ 
--	--	---

Implementation of functions with only NOR (or only NAND) gates is more than a theoretical curiosity. There are chips which provide a "sea of gates" (say, NOR gates) which can be configured by the user (through programming) to implement functions. ↵

M. B. Patil, IIT Bombay

It turns out that we can also use NOR gates to implement logical functions without using any other gates. So, the NOT and OR operations can be realized using only NOR gates and let us look at these one by one. Here is the implementation of the NOT operation using a NOR gate. So, we have a NOR gate here, its inputs are connected together and that is connected to the input A. So, at the output we have A plus A bar like that and A plus A is nothing, but A. So, therefore, we have A bar. So, that is the implementation of the NOT operation.

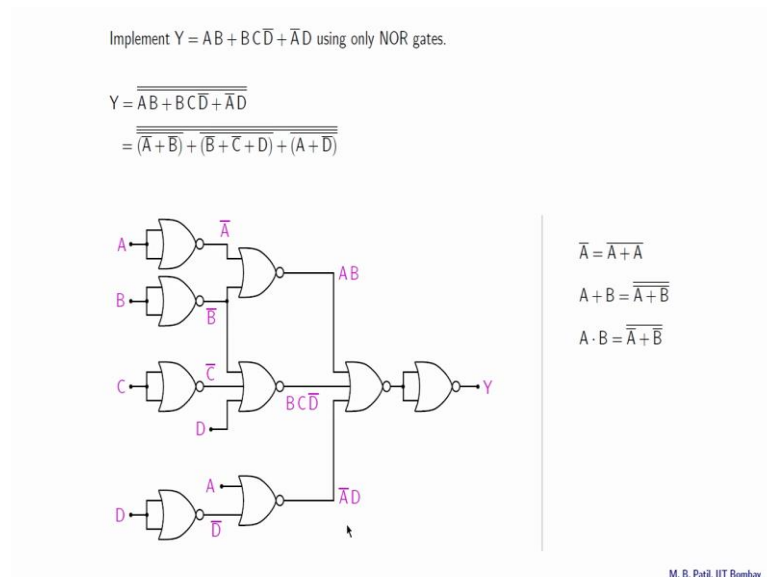
To implement the AND operation we can use De Morgan's theorem that is A and B is A bar plus B bar and the bar of the whole thing all right. So, what do we need to do? We need to generate A bar and B bar using nor gates and we already know how to do that this is the inverter that we can use. So, from A we generate A bar, from B we generate B bar and then that goes to this third NOR gate; and then we get A B at the output.

What about the OR operation? A OR B is nothing but A OR B bar and bar again. So, this is the first NOR gate that gives us A plus B bar and the bar of that again can be obtained using this inverter to get A plus B at the output. So, at this point we have looked at

implementation of logical functions with only NAND gates and also with only NOR gates.

Now, the question that comes up is are we doing this for fun or is there some practical implication of all of this. The answer is that these are actually practically very relevant and let us see why. Implementation of functions with only NOR or only NAND gates is more than a theoretical curiosity. There are chips which provide a “sea of gates” that is a large number of gates of only 1 type say NOR gates or NAND gates and these gates can be configured by the user through programming to implement functions and that is where this implementation with only NOR or only NAND is very useful.

(Refer Slide Time: 23:42)



Let us implement this function Y equal to $A B$ plus $B C D$ bar plus A bar D using only NOR gates and these are the equations we have looked at for implementation of NOT, OR, AND operations with only NOR gates. So, what is our first step? We have the OR operation here and here is how we can implement the OR operation, we have to do A plus B bar and then NOT of that like that and that can be implemented like this here.

So, this not operation the bar on top is provided by this NOR gate and $A B$ plus $B C D$ bar plus A bar D bar is given by this first NOR gate. So, what are the inputs to this first NOR gate? $A B$, $B C D$ bar and A bar D , like that and now we need to see how to generate each of these. How can we get $A B$? Here is the equation, first we can generate A bar from A using this equation and also B bar from B and then we can use a NOR gate

to get $A \cdot B$ like that. So, this NOR gate gives us A bar; this NOR gate gives us B bar and then finally, this NOR gate gives us $A \cdot B$. Now, these terms can also be similarly implemented like that and you can check that out. So, that completes our implementation of the logical function Y using only NOR gates.

To summarize we have seen the meaning of the terms half adder and full adder and how they can be used to add 2 binary numbers we have also seen how a logical function can be implemented using only NAND gates or only NOR gates. In the next class we will start looking at more complex elements such as multiplexer. So, see you next time.