UNIT III

Designing Combinatorial Circuits

The design of a combinational circuit starts from the verbal outline of the problem and ends with a logic circuit diagram or a set of Boolean functions from which the Boolean function can be easily obtained. The procedure involves the following steps:

- The problem is stated
- The number of available input variables and required output variables is determined.
- The input and output variable are assigned their letter symbol
- The truth table that defines the required relationship between the inputs and the outputs is derived.
- The simplified Boolean function for each output is obtained
- The logic diagram is drawn.

Example of combinational circuit

Adders

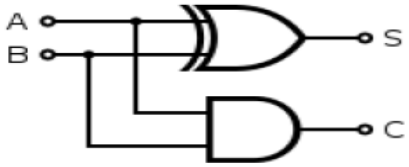
In electronics, an adder or summer is a digital circuit that performs addition of numbers. In modern computers adders reside in the arithmetic logic unit (ALU) where other operations are performed. Although adders can be constructed for many numerical representations, such as Binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where twos complement or ones complement is being used to represent negative numbers, it is trivial to modify an adder into an adder-subtracter. Other signed number representations require a more complex adder. **-Half Adder** A half adder is a logical circuit that performs an addition operation on two binary digits. The half adder produces a sum and a carry value which are both binary digits. A half adder has two inputs, generally labelled A and B, and two outputs, the sum S and carry C. S is the two-bit XOR of A and B, and C is the AND of A and B. Essentially the output of a half adder is the sum of two one-bit numbers, with C being the most significant of these two outputs. The drawback of this circuit is that in case of a multibit addition, it cannot include a carry.

Following is the truth table for a half adder:

A	*B*	*Carry*	*Sum*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Equation of the Sum and Carry. Sum=A'B+AB' Carry=AB

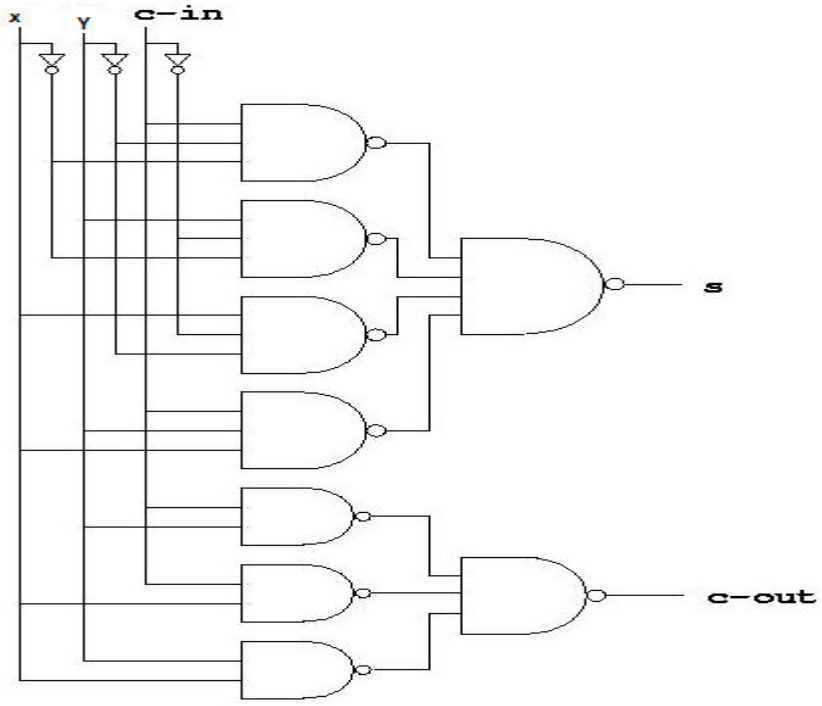
One can see that Sum can also be implemented using XOR gate as $A \oplus B$



Full Adder. A full adder has three inputs *A*, *B*, and a carry in *C*, such that multiple adders can be used to add larger numbers. To remove ambiguity between the input and output carry lines, the carry in is labelled *Ci* or *Cin* while the carry out is labelled *Co* or *Cout*. A full adder is a logical circuit that performs an addition operation on three binary digits. The full adder produces a sum and carry value, which are both binary digits. It can be combined with other full adders or work on its own.

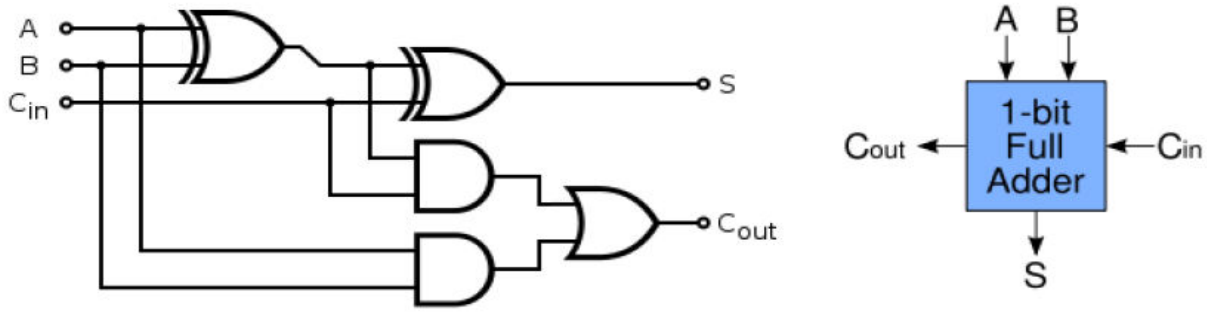
Input			Output	
A	B	C_i	C_o	*S*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Co=A'BCi+AB'Ci+ABCi'+ABCi S=A'B'Ci +A'BCi'+ABCi'+ABCi A full adder can be trivially built using our ordinary design methods for combinatorial circuits. Here is the resulting



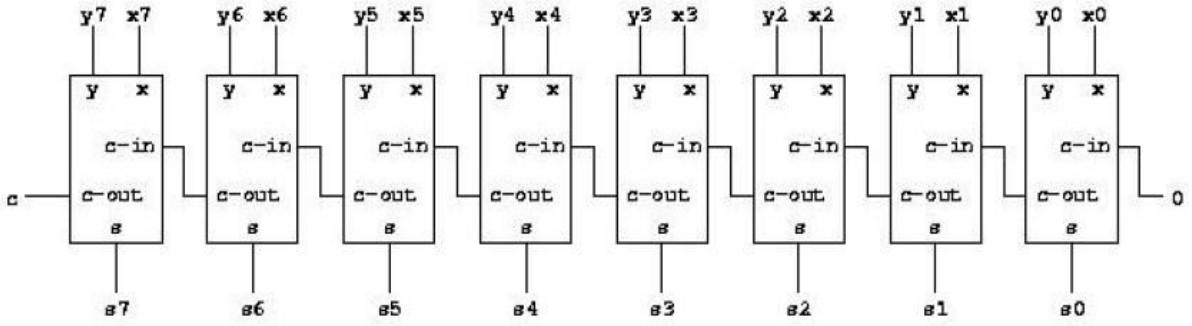
circuit diagram using NAND gates only:

Co=A’BCi+AB’Ci+ABCi’+ABCi by manipulating Co, we can see thatCo= Ci(AB)+AB
S=A’B’Ci +A’BCi’+ABCi’+ABCi By manipulating S, we can see that S=Ci(AB) Note that the final OR gate before the carry-out output may be replaced by an XOR gate without altering the resulting logic. This is because the only discrepancy between OR and XOR gates occurs when both inputs are 1; for the adder shown here, this is never possible. Using only two types of gates is convenient if one desires to implement the adder directly using common IC chips. A full adder can be constructed from two half adders by connecting A and B to the input of one half adder, connecting the sum from that to an input to the second adder, connecting Ci to the other input and OR the two carry outputs. Equivalently, S could be made the three-bit xor of A, B, and Ci and Co could be made the three-bit majority function of A, B, and Ci. The output of the full adder is the two-bit arithmetic sum of three one-bit numbers.



Ripple carry adder

It is possible to create a logical circuit using multiple full adders to add *N*-bit numbers. Each full adder inputs a *Cin*, which is the *Cout* of the previous adder. This kind of adder is a *ripple carry adder*, since each carry bit "ripples" to the next full adder. Note that the first (and only the first) full adder may be replaced by a half adder.

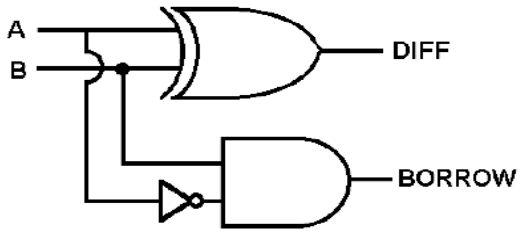


The layout of ripple carry adder is simple, which allows for fast design time; however, the ripple carry adder is relatively slow, since each full adder must wait for the carry bit to be calculated from the previous full adder. The gate delay can easily be calculated by inspection of the full adder circuit. Following the path from *Cin* to *Cout* shows 2 gates that must be passed through. Therefore, a 32-bit adder requires 31 carry computations and the final sum calculation for a total of 31 * 2 + 1 = 63 gate delays.

Subtractor

In electronics, a subtractor can be designed using the same approach as that of an adder. The binary subtraction process is summarized below. As with an adder, in the general case of calculations on multi-bit numbers, three bits are involved in performing the subtraction for each bit: the minuend (Xi), subtrahend (Yi), and a borrow in from the previous (less significant) bit order position (Bi). The outputs are the difference bit (Di) and borrow bit Bi + 1. **Half subtractor** The half-subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, X (minuend) and Y (subtrahend) and two outputs D (difference) and B (borrow). Such a circuit is called a half-subtractor because it enables a borrow out of the current arithmetic operation but no borrow in from a previous arithmetic operation. The truth table for the half subtractor is given below.

X	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



$D=X'Y+XY'$ or $D= X \oplus Y$

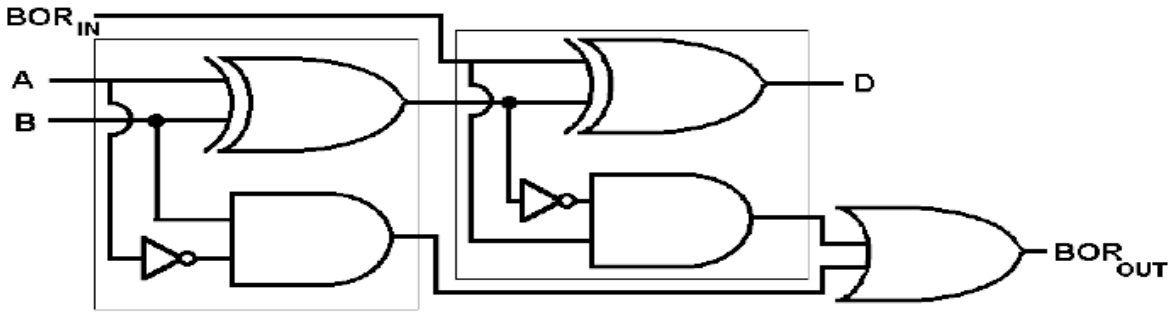
$B=X'Y$

Full Subtractor

As in the case of the addition using logic gates , a *full subtractor* is made by combining two half-subtractors and an additional OR-gate. A full subtractor has the borrow in capability (denoted as **BORIN** in the diagram below) and so allows *cascading* which results in the possibility of **multi-bit subtraction**. The final truth table for a full subtractor looks like:

A	B	BOR_{IN}	D	BOR_{OUT}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

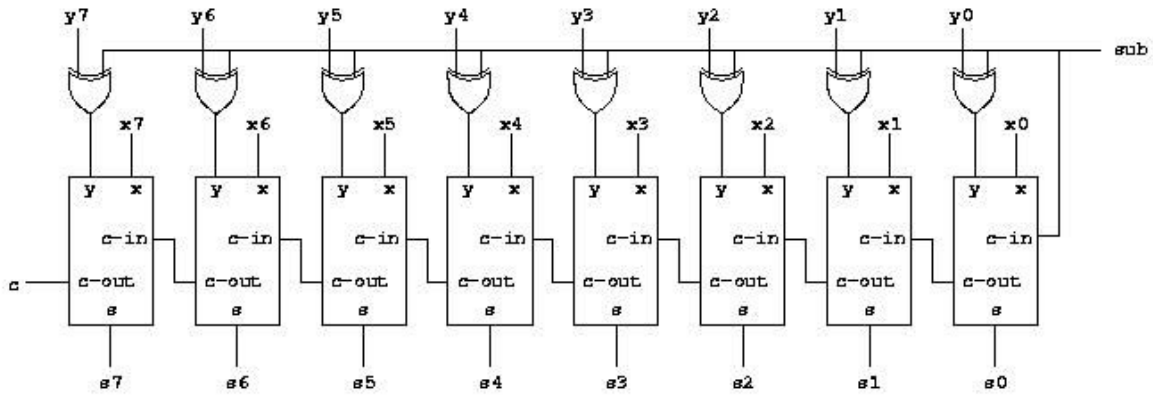
Find out the equations of the borrow and the difference The circuit diagram for a full subtractor is given below.



For a wide range of operations many circuit elements will be required. A neater solution will be to use subtraction via addition using *complementing* as was discussed in the binary arithmetic topic. In this case only adders are needed as shown bellow.

Binary subtraction using adders

Our binary adder can already handle negative numbers as indicated in the section on binary arithmetic But we have not discussed how we can get it to handle subtraction. To see how this can be done, notice that in order to compute the expression *x - y*, we can compute the expression *x + -y* instead. We know from the section on binary arithmetic how to negate a number by inverting all the bits and adding 1. Thus, we can compute the expression as *x + inv(y) + 1*. It suffices to invert all the inputs of the second operand before they reach the adder, but how do we add the 1. That seems to require another adder just for that. Luckily, we have an unused carry-in signal to position 0 that we can use. Giving a 1 on this input in effect adds one to the result. The complete circuit with addition and subtraction looks like this:



Medium Scale integration component

The purpose of circuit minimization is to obtain an algebraic expression that, when implemented results in a low cost circuit. Digital circuit are constructed with integrated circuit(IC). An IC is a small silicon semiconductor crystal called chip containing the electronic component for digital gates. The various gates are interconnected inside the chip to form the required circuit. Digital IC are categorized according to their circuit complexity as measured by the number of logic gates in a single packages.

- Small scale integration (SSI). SSi devices contain fewer than 10 gates. The input and output of the gates are connected directly to the pins in the package.

- Medium Scale Integration. MSI devices have the complexity of approximately 10 to 100 gates in a single package
- Large Scale Integration (LSI). LSI devices contain between 100 and a few thousand gates in a single package
- Very Large Scale Integration(VLSI). VLSI devices contain thousand of gates within a single package. VLSI devices have revolutionized the computer system design technology giving the designer the capabilities to create structures that previously were uneconomical.

Multiplexer

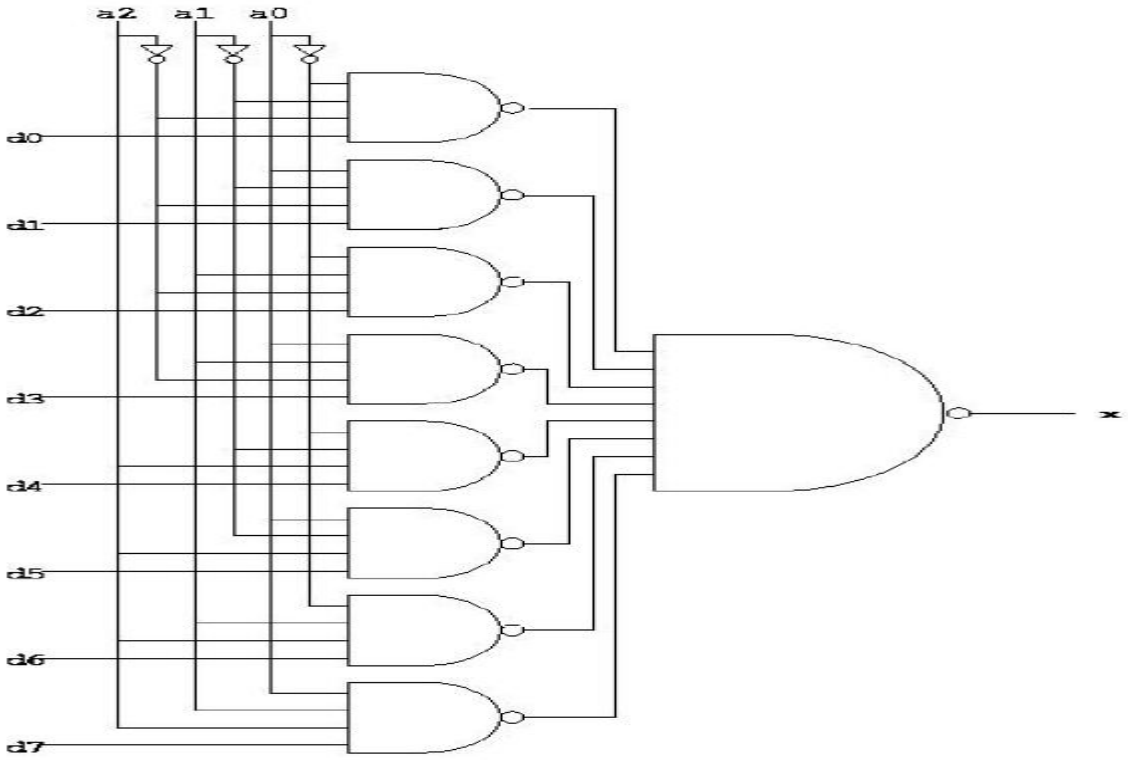
A multiplexer is a combinatorial circuit that is given a certain number (usually a power of two) *data inputs*, let us say 2n, and n *address inputs* used as a binary number to select one of the data inputs. The multiplexer has a single output, which has the same value as the selected data input. In other words, the multiplexer works like the input selector of a home music system. Only one input is selected at a time, and the selected input is transmitted to the single output. While on the music system, the selection of the input is made manually, the multiplexer chooses its input based on a binary number, the address input. The truth table for a multiplexer is huge for all but the smallest values of n. We therefore use an abbreviated version of the truth table in which some inputs are replaced by `-' to indicate that the input value does not matter. Here is such an abbreviated truth table for n = 3. The full truth table would have 2(3 + 23) = 2048 rows.

SELECT			INPUT								
a_2	a_1	a_0	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	x
0	0	0	-	-	-	-	-	-	-	0	0
0	0	0	-	-	-	-	-	-	-	1	1
0	0	1	-	-	-	-	-	-	0	-	0
0	0	1	-	-	-	-	-	-	1	-	1
0	1	0	-	-	-	-	-	0	-	-	0
0	1	0	-	-	-	-	-	1	-	-	1
0	1	1	-	-	-	-	0	-	-	-	0
0	1	1	-	-	-	-	1	-	-	-	1
1	0	0	-	-	-	0	-	-	-	-	0
1	0	0	-	-	-	1	-	-	-	-	1
1	0	1	-	-	0	-	-	-	-	-	0
1	0	1	-	-	1	-	-	-	-	-	1
1	1	0	-	0	-	-	-	-	-	-	0
1	1	0	-	1	-	-	-	-	-	-	1
1	1	1	0	-	-	-	-	-	-	-	0
1	1	1	1	-	-	-	-	-	-	-	1

We can abbreviate this table even more by using a letter to indicate the value of the selected input, like this:

a_2	a_1	a_0	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	x
0	0	0	-	-	-	-	-	-	-	c	c
0	0	1	-	-	-	-	-	-	c	-	c
0	1	0	-	-	-	-	-	c	-	-	c
0	1	1	-	-	-	-	c	-	-	-	c
1	0	0	-	-	-	c	-	-	-	-	c
1	0	1	-	-	c	-	-	-	-	-	c
1	1	0	-	c	-	-	-	-	-	-	c
1	1	1	c	-	-	-	-	-	-	-	c

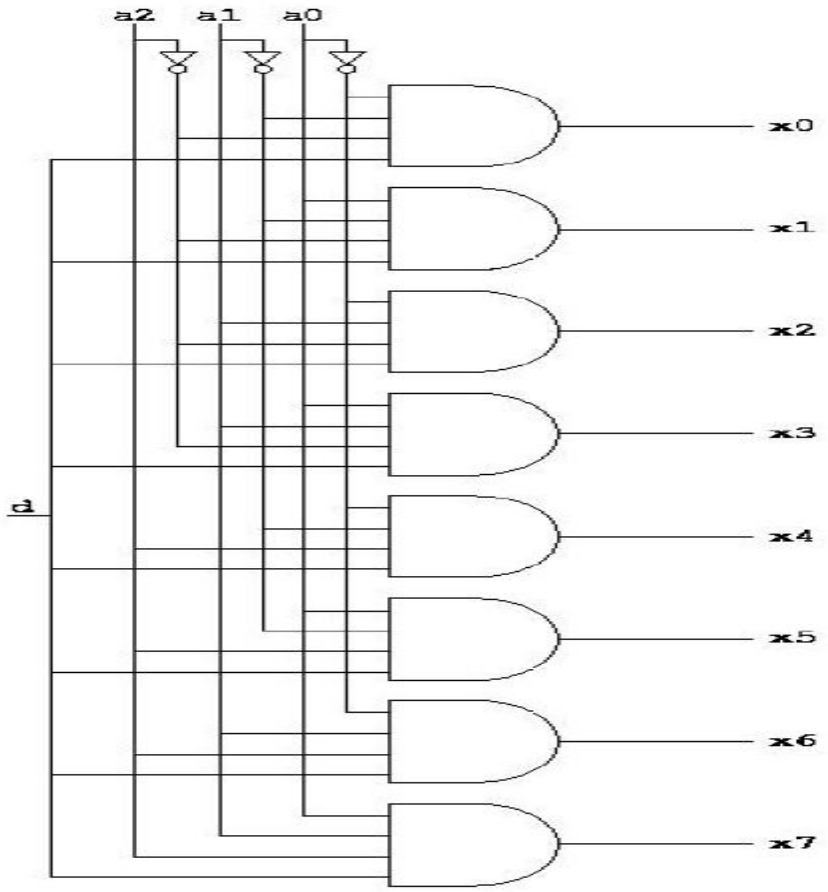
The same way we can simplify the truth table for the multiplexer, we can also simplify the corresponding circuit. Indeed, our simple design method would yield a very large circuit. The simplified circuit looks like this:



Demultiplexer

The demultiplexer is the inverse of the multiplexer, in that it takes a single data input and n address inputs. It has 2n outputs. The address input determine which data output is going to have the same value as the data input. The other data outputs will have the value 0. Here is an abbreviated truth table for the demultiplexer. We could have given the full table since it has only 16 rows, but we will use the same convention as for the multiplexer where we abbreviated the values of the data inputs.

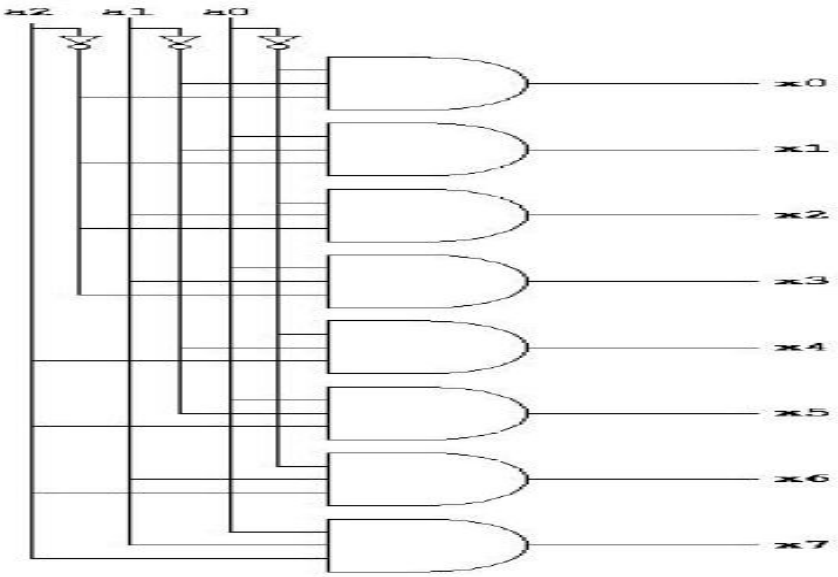
a2	a1	a0	d	x7	x6	x5	x4	x3	x2	x1	x0
0	0	0	c	0	0	0	0	0	0	0	c
0	0	1	c	0	0	0	0	0	0	c	0
0	1	0	c	0	0	0	0	0	c	0	0
0	1	1	c	0	0	0	0	c	0	0	0
1	0	0	c	0	0	0	c	0	0	0	0
1	0	1	c	0	0	c	0	0	0	0	0
1	1	0	c	0	c	0	0	0	0	0	0
1	1	1	c	c	0	0	0	0	0	0	0



Decoder

In both the multiplexer and the demultiplexer, part of the circuits *decode* the address inputs, i.e. it translates a binary number of n digits to 2n outputs, one of which (the one that corresponds to the value of the binary number) is 1 and the others of which are 0. It is sometimes advantageous to separate this function from the rest of the circuit, since it is useful in many other applications. Thus, we obtain a new combinatorial circuit that we call the *decoder*. It has the following truth table (for n = 3)

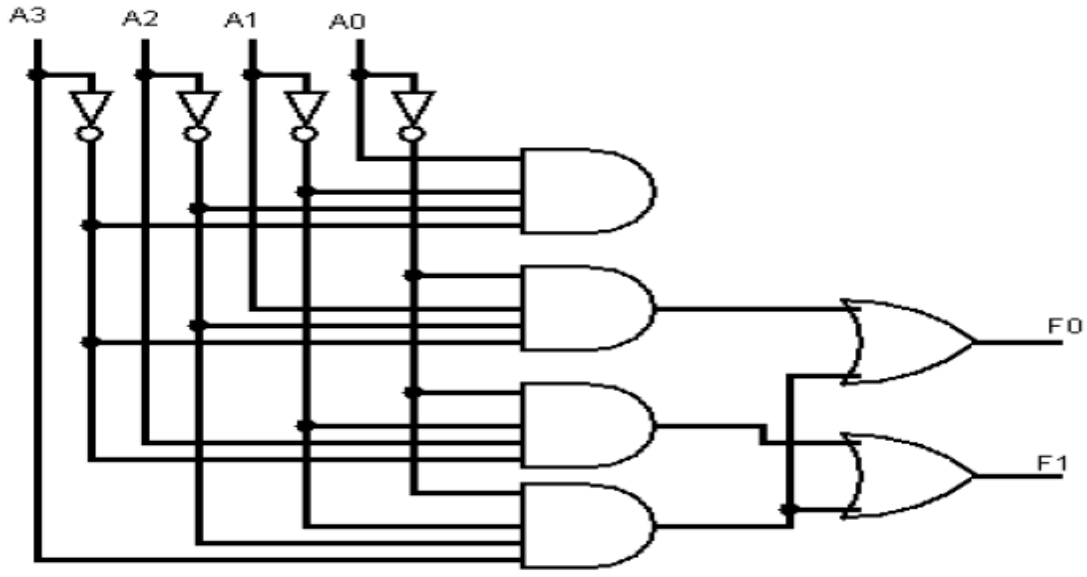
a2	a1	a0	x7	x6	x5	x4	x3	x2	x1	x0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0



Encoder An encoder has 2*n* input lines and *n* output lines. The output lines generate a binary code corresponding to the input value. For example a single bit 4 to 2 encoder takes in 4 bits and outputs 2 bits. It is assumed that there are only 4 types of input signals these are : 0001, 0010, 0100, 1000.

I_3	I_2	I_1	I_0	F_1	F_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

4 to 2 encoder



The encoder has the limitation that only one input can be active at any given time. If two inputs are simultaneously active, the output produces an undefined combination. To prevent this we make use of the priority encoder. A priority encoder is such that if two or more inputs are given at the same time, the input having the highest priority will take precedence. An example of a single bit 4 to 2 encoder is shown.

I_3	I_2	I_1	I_0	F_1	F_0
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

4 to 2 priority encoder The X's designate the don't care condition designating that fact that the binary value may be equal either to 0 or 1. For example, the input I3has the highest priority so regarded the value of other inputs, if the value of I3 is 1, the output for F1F0=11(binary 3)