SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF Computer Science and Engineering

COURSE MATERIAL

Subject Name: Fundamentals of Digital Systems **UNIT III** **Subject Code: SCSA1201**

Design Procedure - Adder - Subtracter - Code Conversion - Analysis Procedure - Multilevel NAND/NOR circuits - Exclusive OR functions - Binary adder and subtractor- Decimal adder - BCD adder - Magnitude Comparator - Decoders - Demultiplexer - Encoder – Multiplexers.

1. Design Procedure - Adder - Subtracter - Code Conversion

Half Adder

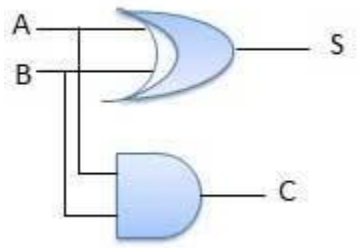
Half adder is a combinational logic circuit with two inputs and two outputs. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two **single** bit numbers. This circuit has two outputs **carry** and **sum**.

Block diagram

Truth Table

Inputs		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

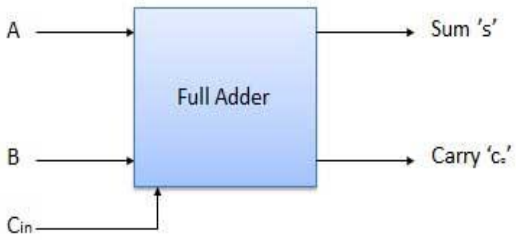
Circuit Diagram



Full Adder

Full adder is developed to overcome the drawback of Half Adder circuit. It can add two one-bit numbers A and B, and carry c. The full adder is a three input and two output combinational circuit.

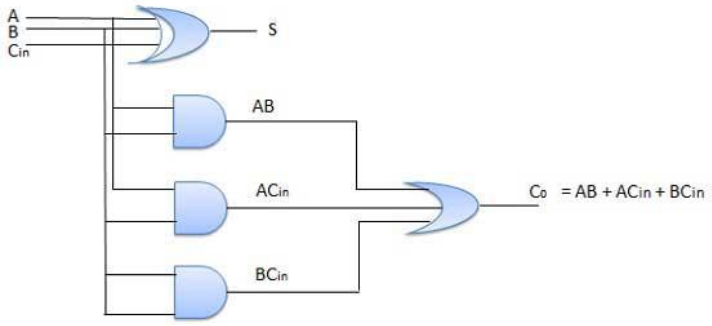
Block diagram



Truth Table

Inputs			Output	
A	B	Cin	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Circuit Diagram



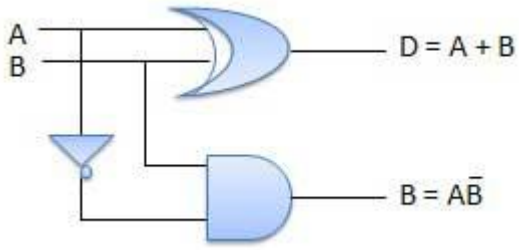
Half Subtractors

Half subtractor is a combination circuit with two inputs and two outputs (difference and borrow). It produces the difference between the two binary bits at the input and also produces an output (Borrow) to indicate if a 1 has been borrowed. In the subtraction (A-B), A is called as Minuend bit and B is called as Subtrahend bit.

Truth Table

Inputs		Output	
A	B	(A – B)	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Circuit Diagram

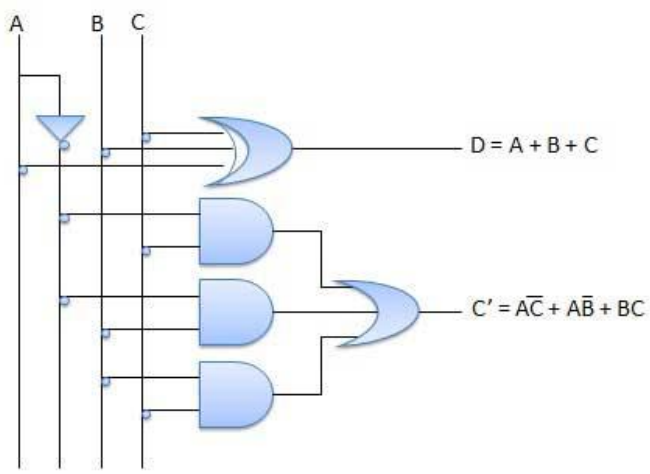


Full Subtractors

The disadvantage of a half subtractor is overcome by full subtractor. The full subtractor is a combinational circuit with three inputs A,B,C and two output D and C'. A is the 'minuend', B is 'subtrahend', C is the 'borrow' produced by the previous stage, D is the difference output and C' is the borrow output.

Truth Table **Circuit Diagram**

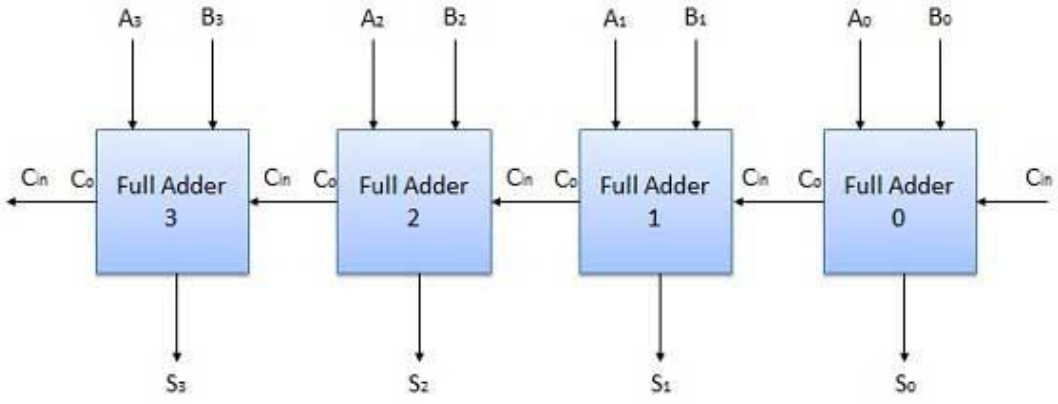
Inputs			Output	
A	B	C	(A-B-C)	C'
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



4 Bit Parallel Adder

In the block diagram, A_0 and B_0 represent the LSB of the four bit words A and B. Hence Full Adder-0 is the lowest stage. Hence its C_{in} has been permanently made 0. The rest of the connections are exactly same as those of n-bit parallel adder is shown in fig. The four bit parallel adder is a very common logic circuit.

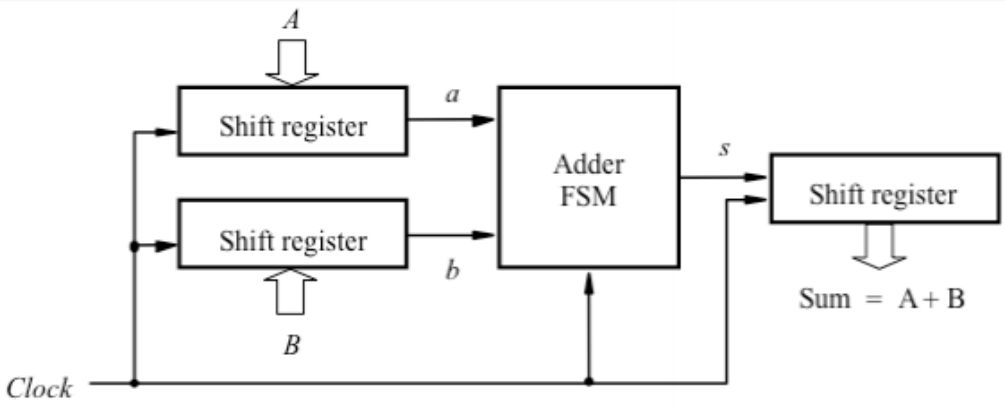
Block diagram



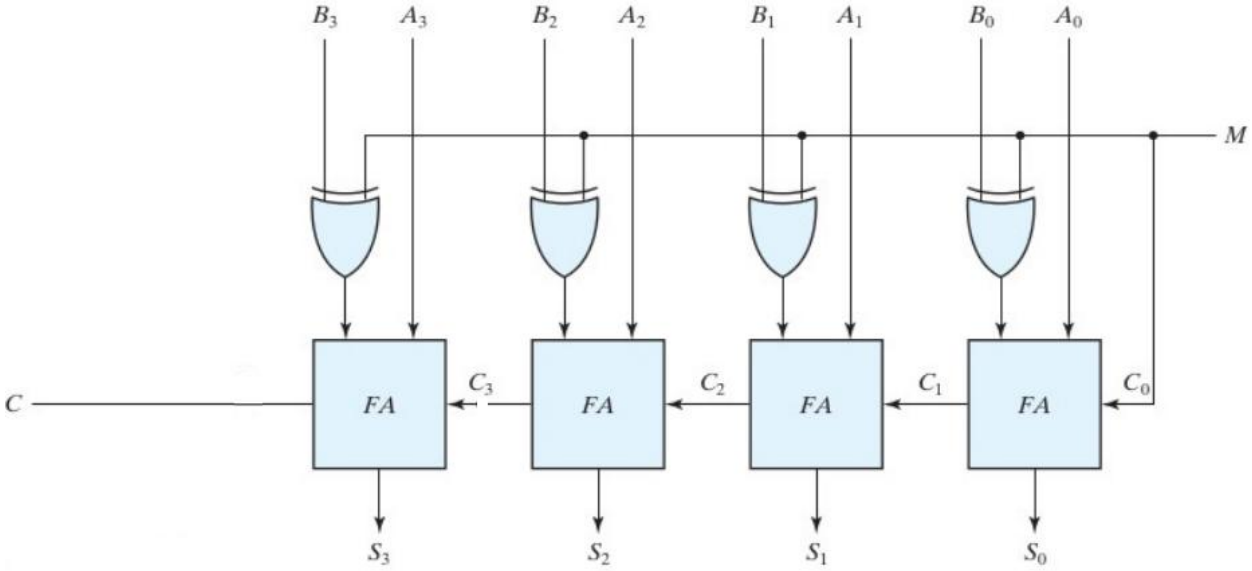
Serial Adder

If speed is not of great importance, a cost-effective option is to use a serial adder

Serial adder: bits are added a pair at a time (in one clock cycle)



4 Bit Adder/ Subtractor



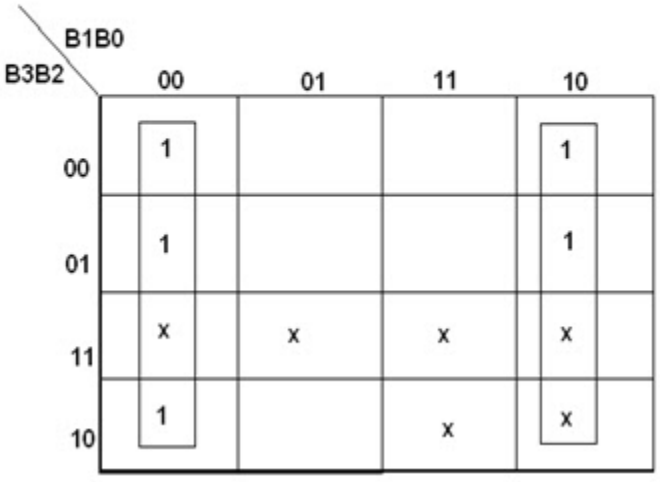
The circuit for subtracting A - B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The input carry C0 must be equal to 1 when subtraction is performed. The operation thus performed becomes A, plus the 1's complement of B, plus 1. This is equal to A plus the 2's complement of B.

Code conversion

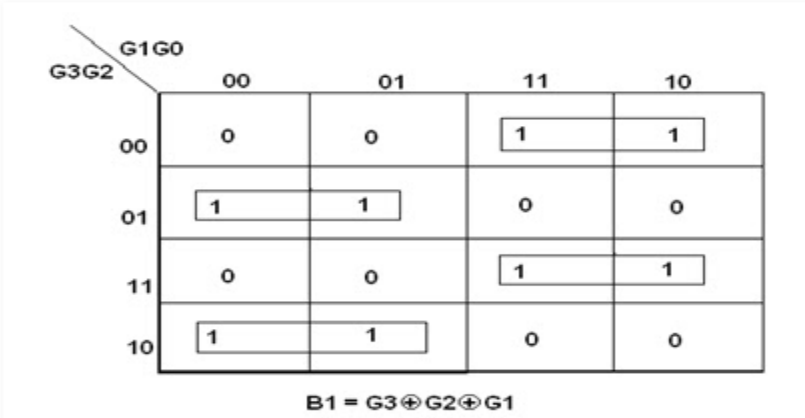
BCD Input				Excess-3 Output			
B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

K-Map for E0:-

E0=Bar (B0)



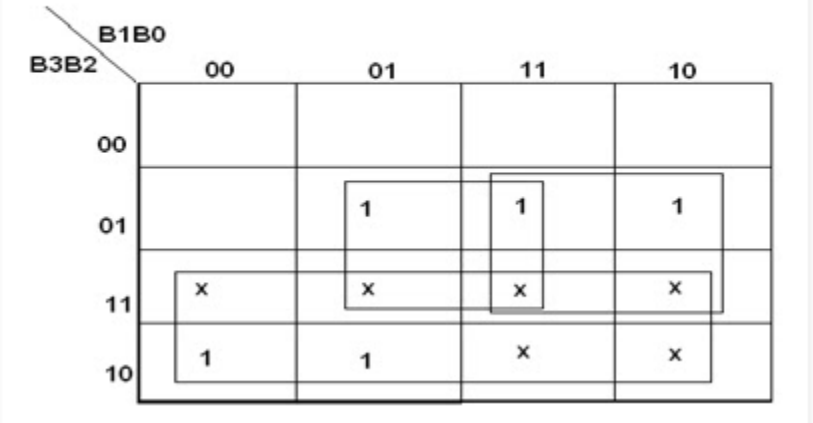
K-Map for E1:-



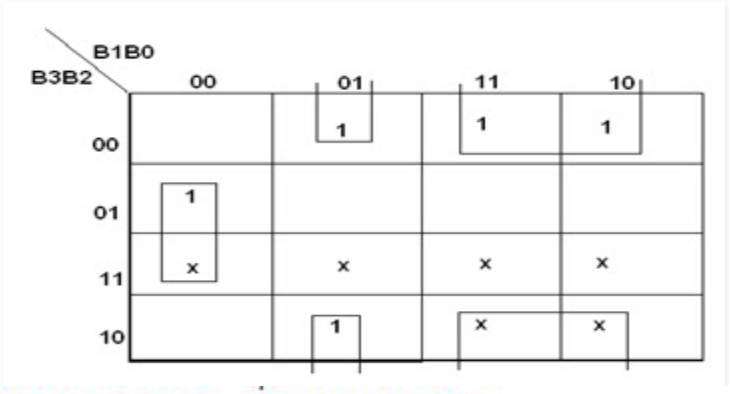
B1 = G3⊕G2⊕G1

K-Map for E3:-

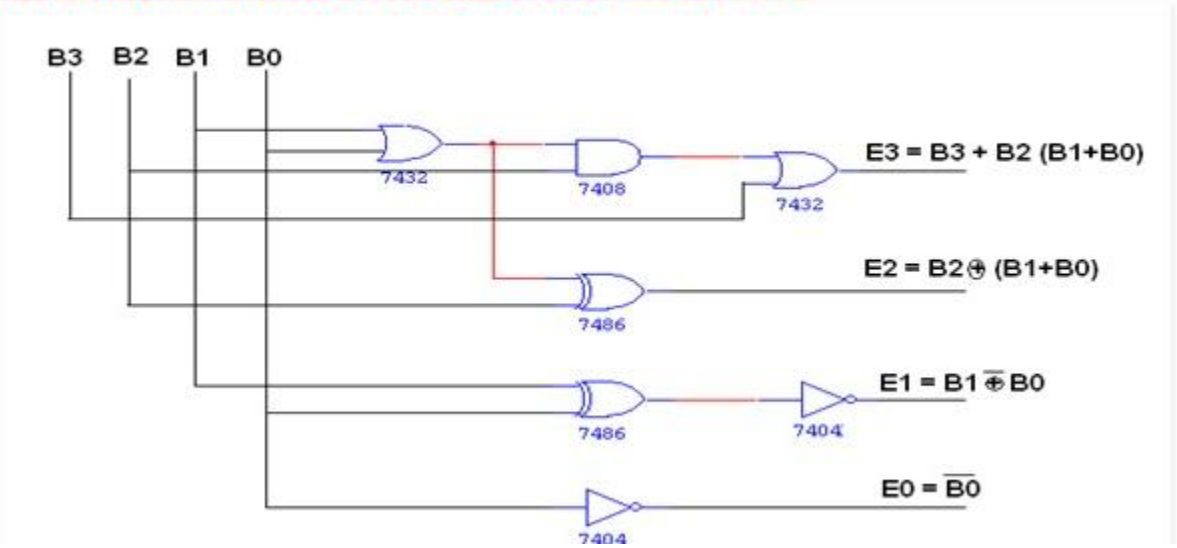
E3 = B3 + B2 (B0 + B1)



K-Map for E2:-



Logic Diagram for BCD to Excess-3 Converter:-

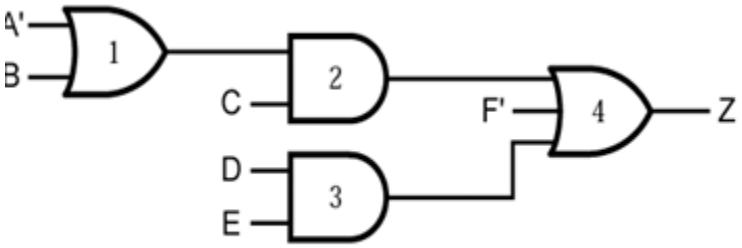


2. Analysis Procedure - Multilevel NAND/NOR circuits

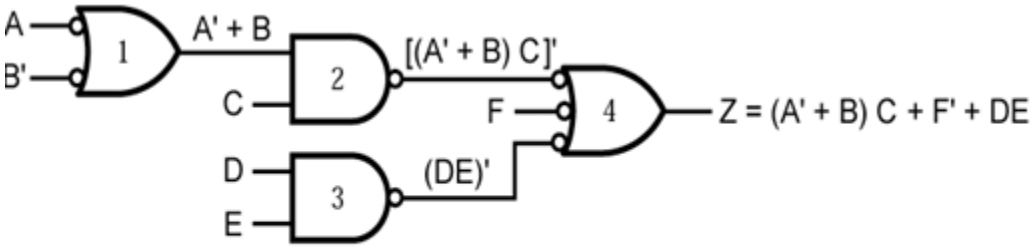
Multi-Level Gate Circuits

Two-level circuits consisting of AND and OR gates can easily be converted to networks that can be realized only NAND and NOR gates – A two-level AND-OR (SOP) circuit can be realized (directly) as a two-level NAND-NAND circuit – A two-level OR-AND (POS) circuit can be realized (directly) as a two-level NOR-NOR circuit . The same approach can be used for multilevel networks.

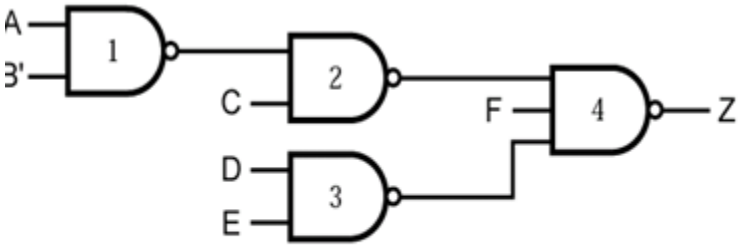
AND – OR to NAND- NAND example



(c) Equivalent AND-OR network

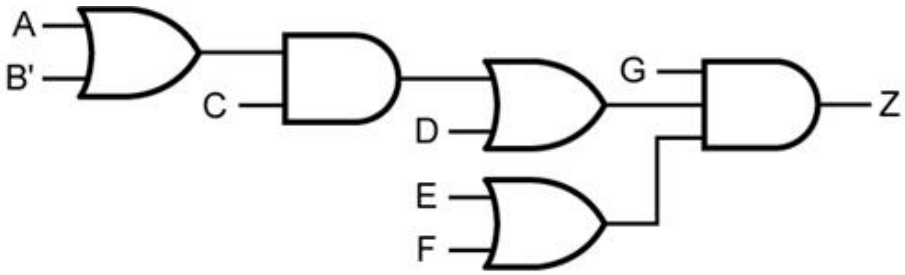


(b) Alternate form for NAND gate network

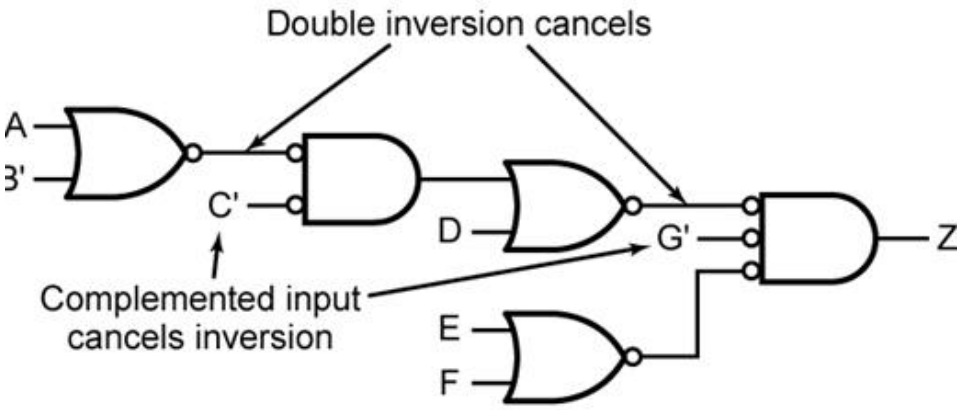


(a) NAND gate network

OR –AND to NAND-NAND example



(a) Circuit with OR and AND gates

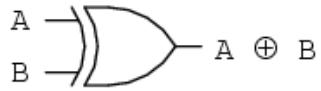


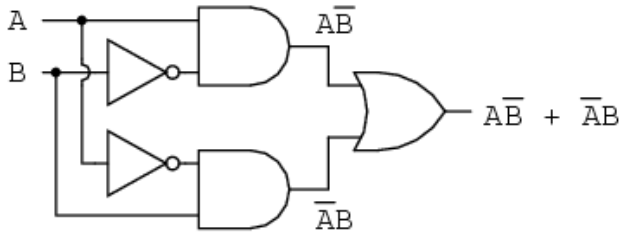
(b) Equivalent circuit with NOR gates

3. Exclusive OR functions - Binary adder and subtractor- Decimal adder - BCD adder

Exclusive OR functions:

The XOR function operates such that when both inputs are the same the output is zero. The output is only positive if one of the inputs is on. As a Boolean equivalency, this rule may be helpful in simplifying some Boolean expressions. Any expression following the AB’ + A’B form (two AND gates and an OR gate) may be replaced by a single Exclusive-OR gate.





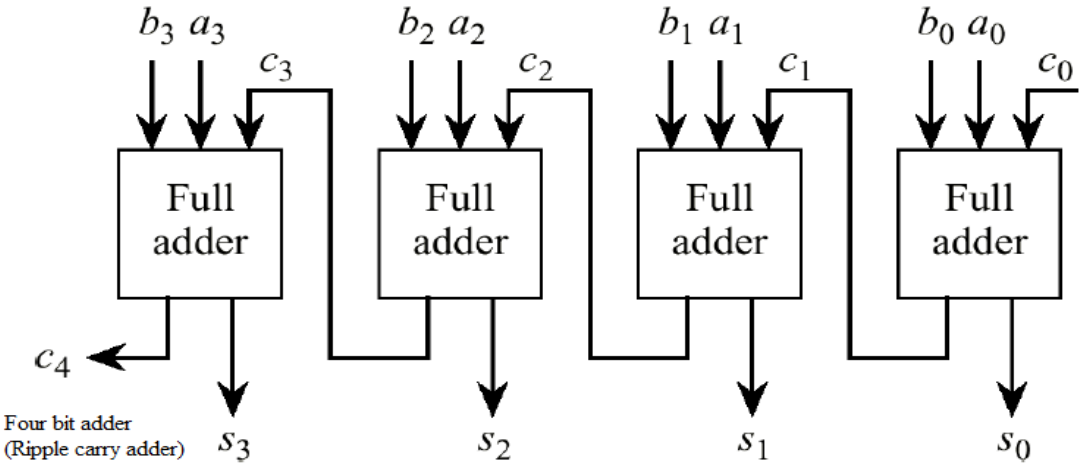
Truth table for XOR Gate

INPUTS		OUTPUTS
A	B	Y = A⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

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

Binary Adder (Asynchronous Ripple-Carry Adder):

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. A binary adder can be constructed with full adders connected in cascade with the output carry form each full adder connected to the input carry of the next full adder in the chain. The four-bit adder is a typical example of a standard component .It can be used in many application involving arithmetic operations.



The input carry to the adder is c_0 and it ripples through the full adders to the output carry c_4.

n -bit binary adder requires n full adders.

Subscript i	3	2	1	0	
Input carry	0	1	1	0	C_i
A	1	0	1	1	A_i
+					
B	0	0	1	1	B_i
SUM	1	1	1	0	S_i
Output Carry	0	0	1	1	C_{i+1}

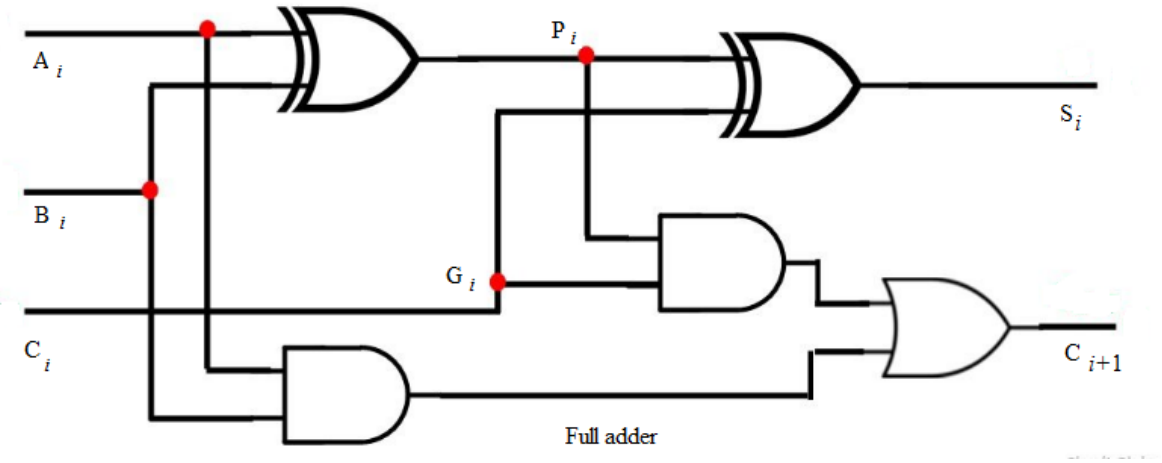
A= 1011

B= 0011

$C_0 = 0$

Carry Propagation

The addition of A+B binary numbers in parallel implies that all the bits of A and B are available for computation at the same time. As in any combinational circuit, the signal must propagate through the gates before the correct output sum is available. The output will not be correct unless the signals are given enough time to propagate through the gates connected form the input to the output. The longest **propagation delay time** in an adder is the time it takes the carry to propagate through the full adders.



Full adder

The signal form the carry input C_i to the output carry C_{i+1} propagates through an ***AND*** gate and an ***OR*** gate, which equals ***2 gate levels***.

If there are ***4*** full adders in the binary adder, the output carry C_4would have ***2×4=8 gate levels***, form C_0 to C_4

For an n-bit adder, 2n gate levels for the carry to propagate form input to output are required.

The ***carry propagation time*** is an important attribute of the adder because it limits the speed with which two numbers are added.

To reduce the carry propagation delay time:

1) Employ faster gates with reduced delays.

2) Employ the principle of Carry Lookahead Logic

Proof: ***(using carry lookahead logic)***

$P_i = A_i \oplus B_i$

$G_i = A_i B_i$

The output sum and carry are:
$S_i = P_i \oplus C_i$
$C_{i+1} = G_i + P_i C_i$
G_i is known as the ***carry generate***, and it produces a carry of ***1*** when both A_i and B_i are ***1.***

P_i called a ***carry propagate***, it determines whether a carry into stage *i* will propagate into stage *i*+1 **.**

Computing the values of P_i and G_i only depend on the input operand bits (A_i & B_i) as clear from the Figure and equations.

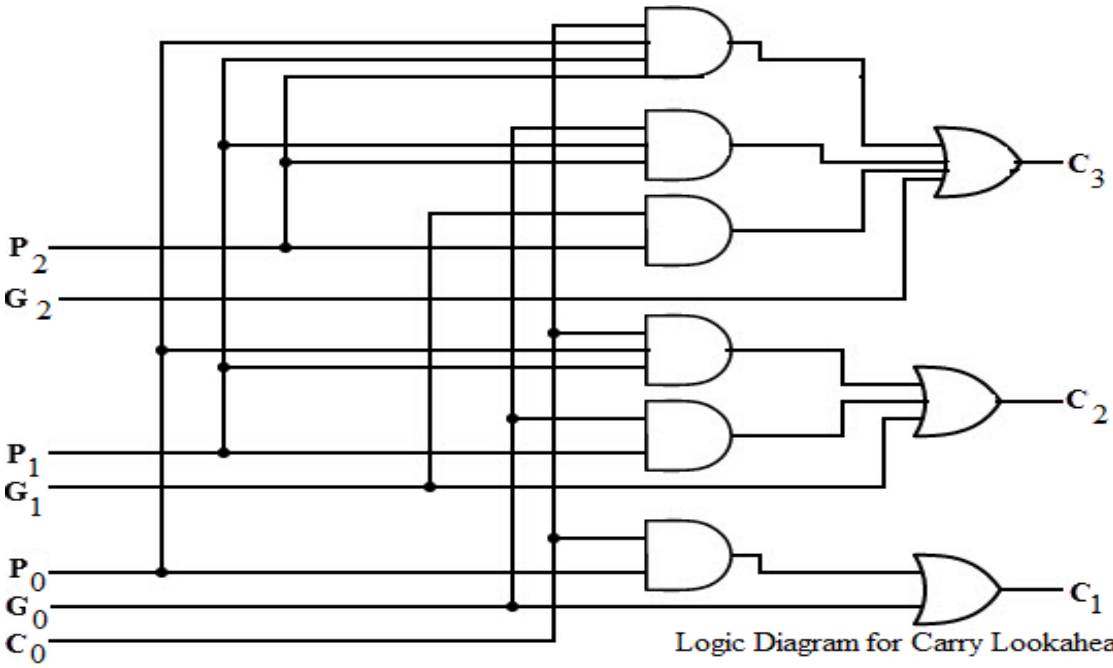
Thus, these signals settle to their **steady-state value** after the propagation through their respective gates.

Computed values of **all** the P_i's are valid one XOR-gate delay after the operands A and B are made valid.
Computed values of **all** the G_i's are valid one AND-gate delay after the operands A and B are made valid.

The ***Boolean function*** for the carry outputs of each stage and substitute the value of each C*i* from the previous equations:

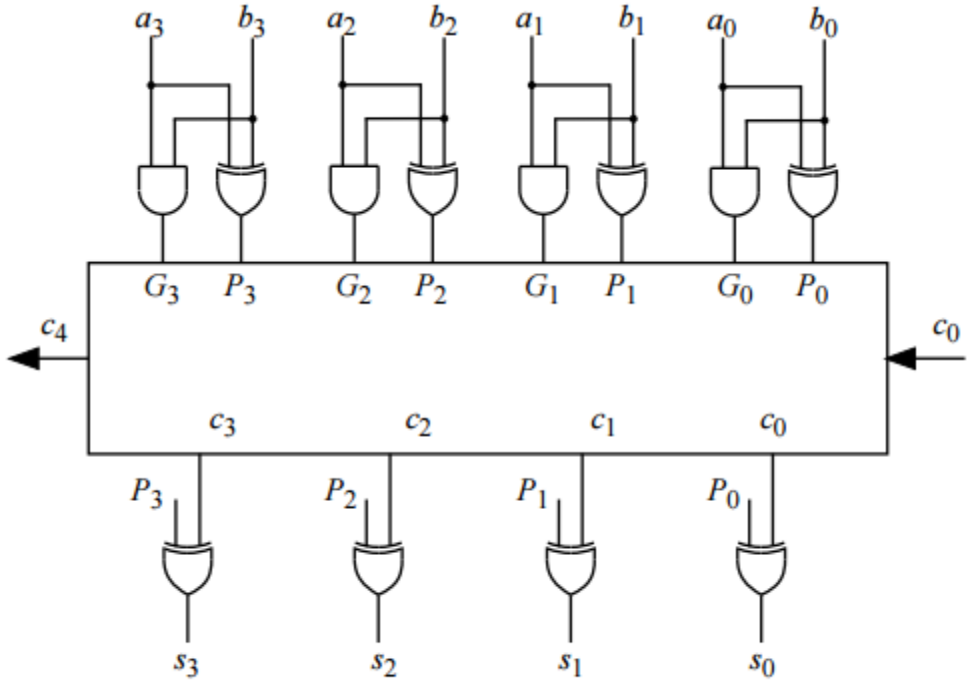
C_0= input carry
$C_1 = G_0 + P_0C_0$
$C_2 = G_1 + P_1C_1 = G_1 + P_1 (G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$
$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

Each carry signal is expressed as a direct SOP function of $\mathbf{C_0}$ rather than its preceding carry signal.
Since the Boolean expression for each output carry is expressed in SOP form, it can be implemented in two-level circuits.
The 2-level implementation of the carry signals has a propagation delay of 2 gates, i.e., 2τ.



Logic Diagram for Carry Lookahead adder

The 4-bit carry look-ahead (CLA) adder consists of 3 levels of logic:



4-Bit carry lookahead adder implementation detail.

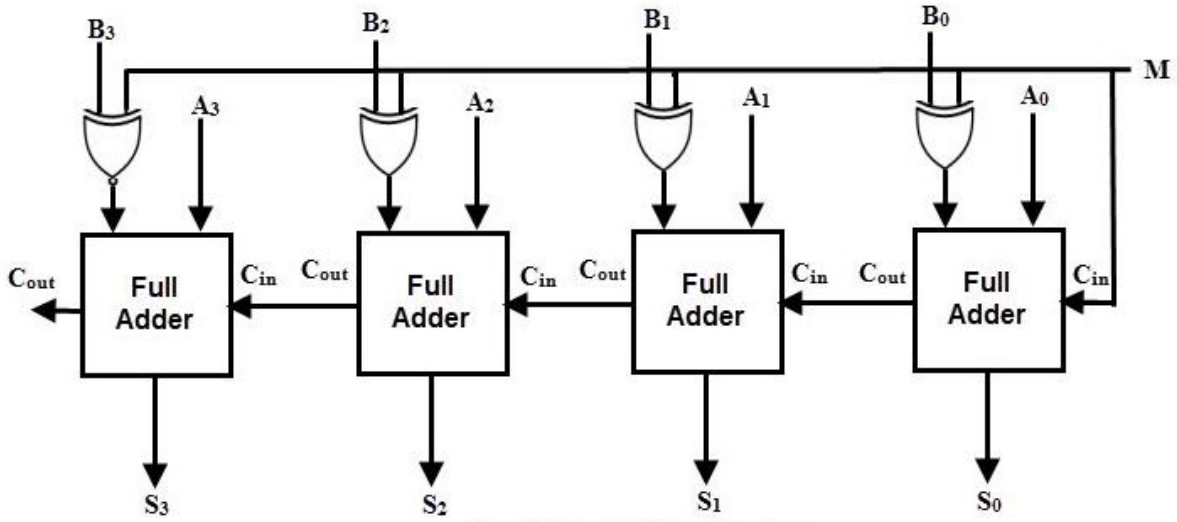
First level: Generates all the P & G signals. Four sets of P & G logic (each consists of an XOR gate and an AND gate). Output signals of this level (P's & G's) will be valid after 1τ.
Second level: The Carry Look-Ahead (CLA) logic block which consists of four 2-level implementation logic circuits. It generates the carry signals (C_0, C_1, C_2, and C_3) as defined by the above expressions. Output signals of this level (C_0, C_1, C_2, and C_3) will be valid after 3τ.
Third level: Four XOR gates which generate the sum signals (S_i) ($S_i = P_i \oplus C_i$). Output signals of this level (S_0, S_1, S_2, and S_3) will be valid after 4τ.
Thus, the 4 Sum signals (S_0, S_1, S_2 & S_3) will all be valid after a total delay of 4τ compared to a delay of (2n+1)τ for Ripple Carry adders.
For a 4-bit adder (n = 4), the Ripple Carry adder delay is 9τ.
The disadvantage of the CLA adders is that the carry expressions (and hence logic) become quite complex for more than 4 bits.
Thus, CLA adders are usually implemented as 4-bit modules that are used to build larger size adders.

Binary Subtractor

To perform the subtraction, we can use the *2's complements*, so the subtraction can be converted to addition.
2's complement can be obtained by talking the *1's complement* and adding 1 to the LSD bit.
1) 1's complement can be implemented with inverters.
2) 1 can be added to the sum through the input carry.
The circuit for subtracting A-B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The input carry C_0 must be equal to 1.

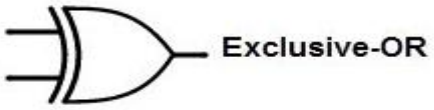
Binary Parallel Adder/Subtractor:

The addition and subtraction operations can be done using an Adder-Subtractor circuit. The figure shows the logic diagram of a 4-bit Adder-Subtractor circuit.



Four bit binary adder-subtractor

The circuit has a mode control signal M which determines if the circuit is to operate as an adder or a subtractor.
Each XOR gate receives input M and one of the inputs of B, i.e., B_i. To understand the behavior of XOR gate consider its truth table given below.



A	B	Z	
0	0	**0**	
0	1	**1**	$\bar{A}B$
1	0	**1**	$A\bar{B}$
1	1	**0**	

$Z = \bar{A}B + A\bar{B}$

If one input of XOR gate is **zero** then the output of XOR will be **same** as the second input. While if one input of XOR gate is **one** then the output of XOR will be **complement** of the second input.

So when **M = 0**, the output of XOR gate will be $B_i \oplus 0 = B_i$. If the full adders receive the value of B, and the input carry C_0 is 0, the circuit performs **A plus B**.

When **M = 1**, the output of XOR gate will be $B_i \oplus 1 = B_i'$. If the full adders receive the value of B', and the input carry C_0 is 1, the circuit performs A plus 1's complement of B plus 1, which is equal to **A minus B**.

BCD ADDER:

Computers or calculators that perform arithmetic operations directly in the decimal number system represent decimal numbers in binary coded form.

An adder for such a computer must employ arithmetic circuits that accept coded decimal numbers and present results in the same code. For binary addition, it is sufficient to consider a pair of significant bits together with a

previous carry. A decimal adder requires a minimum of nine inputs and five outputs, since four bits are required to code each decimal digit and the circuit must have an input and output carry

(1 digit requires 4-bit

Input: 2 digits + 1-bit carry

Output: 1 digit + 1-bit carry)

Since each input digit does not exceed 9, the output sum cannot be greater than 9 + 9 + 1 = 19, the 1 in the sum being an input carry.

Suppose we apply two BCD digits to a four-bit binary adder. The adder will form the sum in binary and produce a result that ranges from 0 through 19.

These binary numbers are labeled by symbols K, Z8, Z4, Z2, and Z1. K is the carry, and the subscripts under the letter Z represent the weights 8, 4, 2, and 1 that can be assigned to the four bits in the BCD code.

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
		.....					.....			.....
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Derivation of BCD adder

When the binary sum is equal to or less than 1001_b

BCD Sum = Binary Sum

C = 0 ;

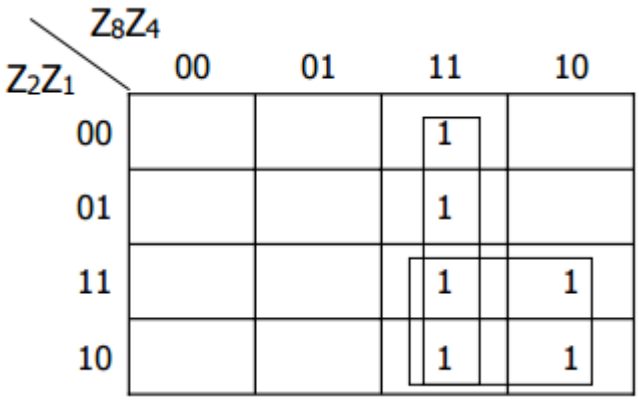
When the binary sum is greater than 1001_b

BCD Sum = Binary Sum + 0110_b

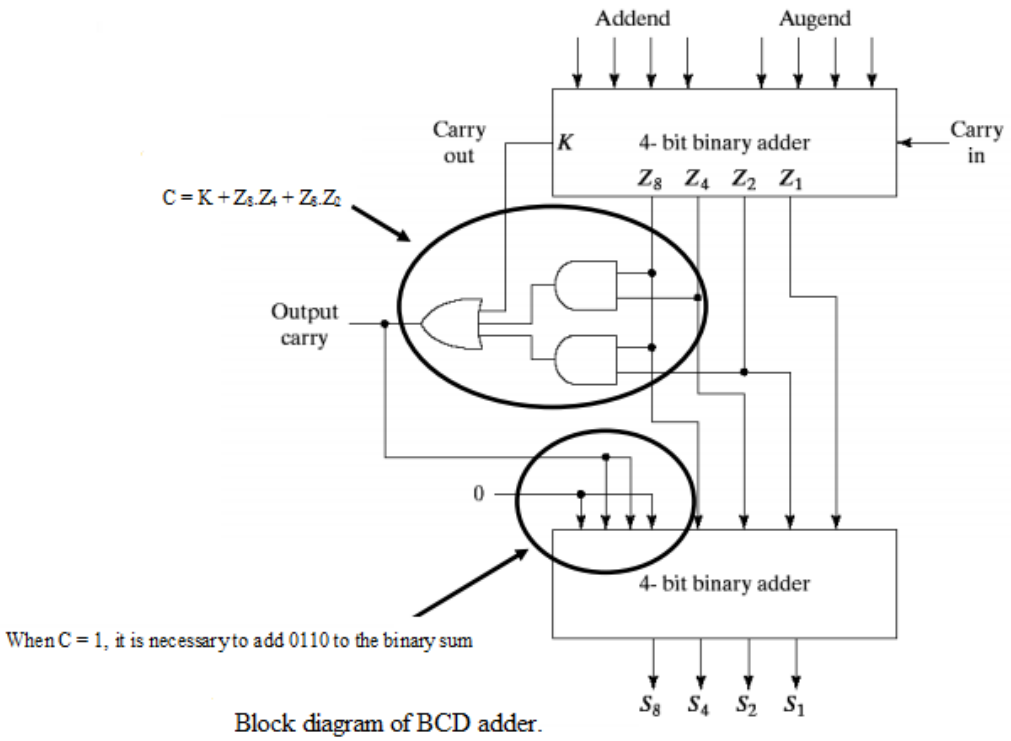
C = 1

The condition for a correction and an output carry can be expressed by the Boolean function

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



When C = 1, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage.



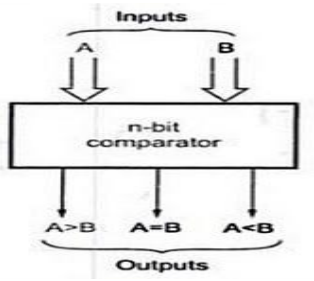
Block diagram of BCD adder.

4. Magnitude Comparator - Decoders - Demultiplexer - Encoder – Multiplexers.

MAGNITUDE COMPARATOR:

A digital **comparator** or **magnitude comparator** is a hardware electronic device that takes two numbers as input in binary form and determines whether one number is greater than, less than or equal to the other number. **Comparators** are used in central processing unit s (CPUs) and microcontrollers (MCUs).

Magnitude Comparator is a combinational circuit capable of comparing the relative magnitude of two binary numbers. It is one of the two types of digital comparator.



Block diagram of n-bit comparator

Figure(a) shows the block diagram of n-bit magnitude comparator. It accepts two n-bit binary numbers, say A and B as inputs and produces one of the outputs: A>B, A=B and A<B.

One of the outputs will be high depending upon the relative magnitude. That is, output A>B will be high if A is greater than B, output A=B will be high if A and B are equal, and output A<B will be high if A is less than B.

Its logic behaviour is same as adder. It does not return sum or carry.

Magnitude comparators are used in central processing units and microcontrollers.

This basic circuit for a magnitude comparator can be extended for any number of bits.

Four bit magnitude comparators are very popular circuits and are commercially available.

Examples: 74HC85 and CMOS 4063. These are four bit magnitude comparators.

XNOR

\

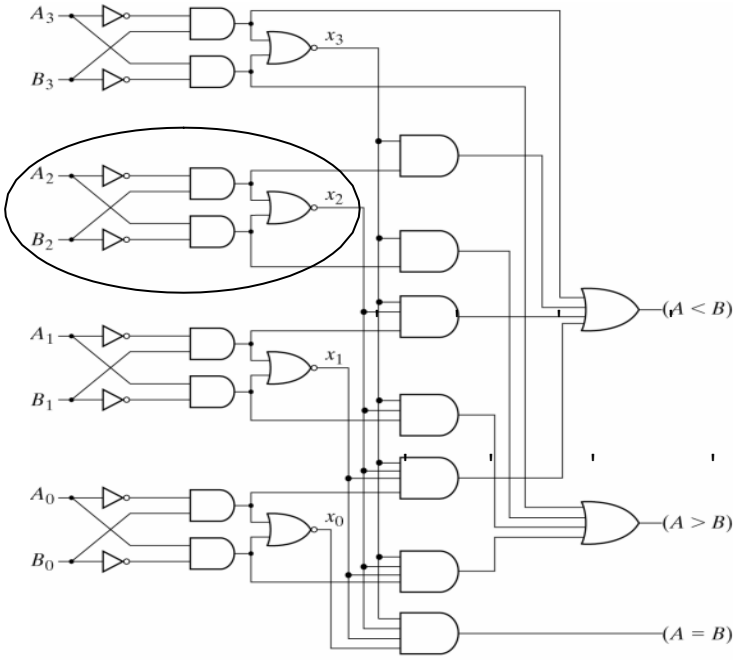


Fig. 4-17 4-Bit Magnitude Comparator

DECODERS

A decoder is a combinational circuit that converts binary information from *n* input lines to an 2^n unique output lines.

Some Applications:

- Microprocessor memory system: selecting different banks of memory.
- Microprocessor I/O: Selecting different devices.
- Memory: Decoding memory addresses (e.g. in ROM).
- In our lab… decoding the binary input to activate the LED segments so

that the decimal number can be displayed.

3-to-8-line DECODER

Binary Inputs			Outputs							
			D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

If the input corresponds to minterm m_i then the decoder ouput$_i$ will be the single asserted output.

3-to-8-line DECODER

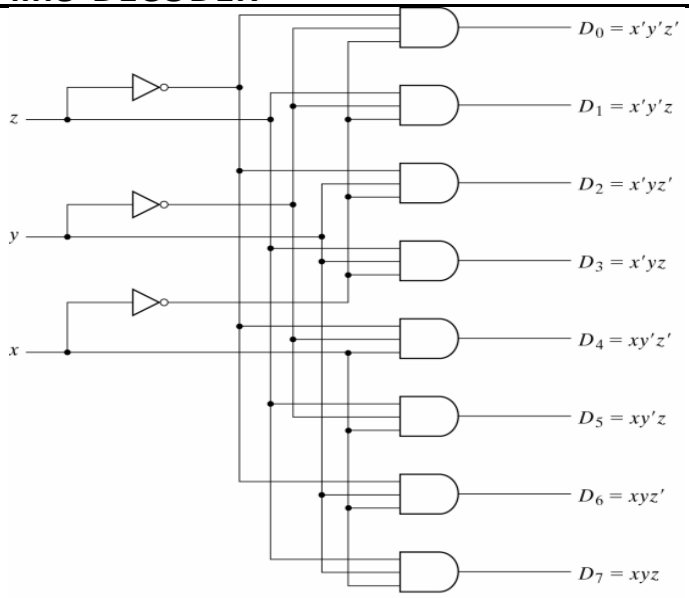


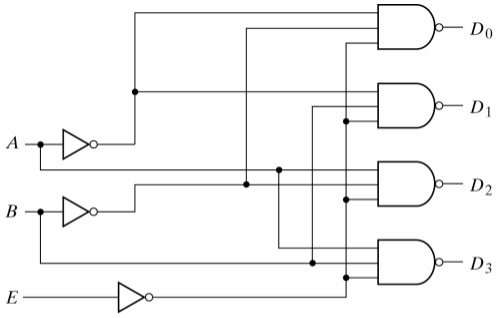
Fig. 4-18 3-to-8-Line Decoder

2-to-4-line DECODER with Enable

The decoder is enabled when E = 0. The output whose value = 0 represents the minterm is selected by inputs A and B.

The decoder is inactive when E = 1€ D_0 … D_3 = 1

A Decoder with enable input is called a decoder/ demultiplexer. Demultiplexer receives information from a single line and directs it to the output lines.



Complemented outputs

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(a) Logic diagram (b) Truth table

Fig. 4-19 2-to-4-Line Decoder with Enable Input

A 4 x 16 DECODER

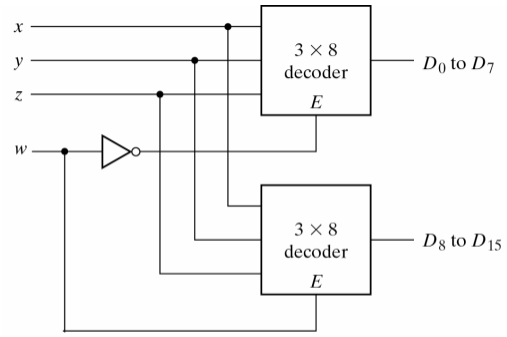


Fig. 4-20 4 × 16 Decoder Constructed with Two 3 × 8 Decoders

• When w = 0, the top decoder is enabled and the bottom is disabled. Top decoder generates 8 minterms 0000 to 0111, while the bottom decoder outputs are 0's.

• When w = 1, the top decoder is disabled and the bottom is enabled. Bottom decoder generates 8 minterms 1000 to 1111, while the top decoder outputs are 0's.

Full-Adder using Decoder

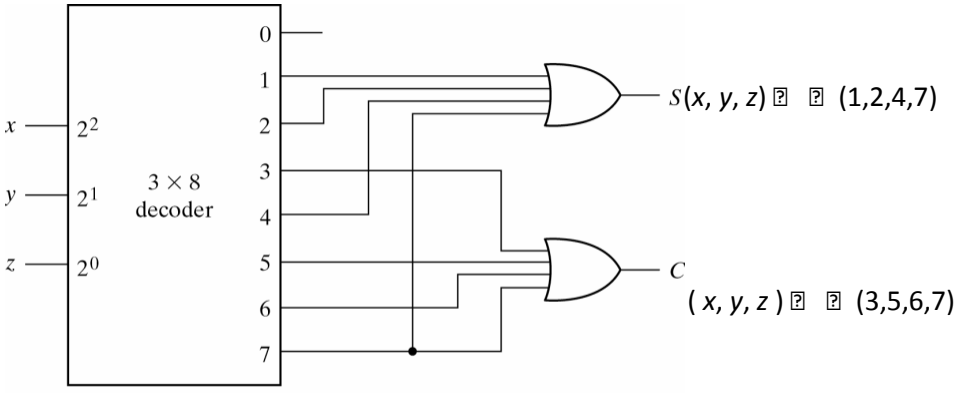


Fig. 4-21 Implementation of a Full Adder with a Decoder

MULTIPLEXERS/DATA SELECTORS

A multiplexer is a combinational circuit that selects one of many input lines (2^n) and steers it to its single output line. There are (2^n) and *n* selection lines whose bit combinations determine which input is selected.

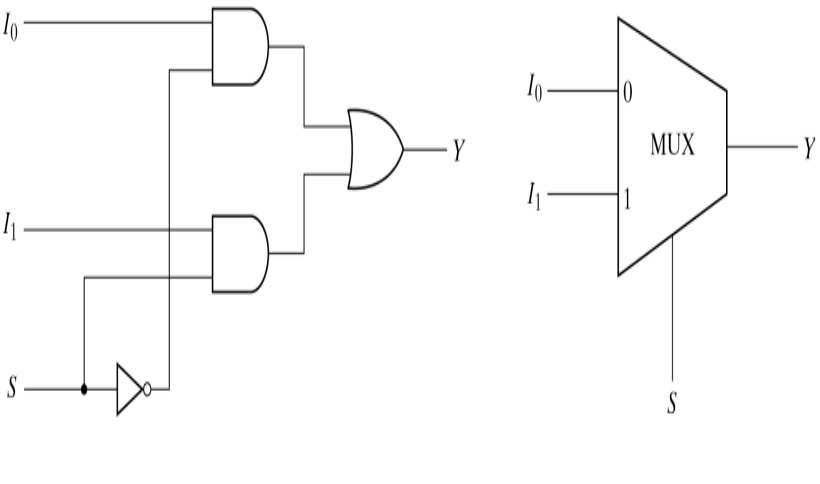
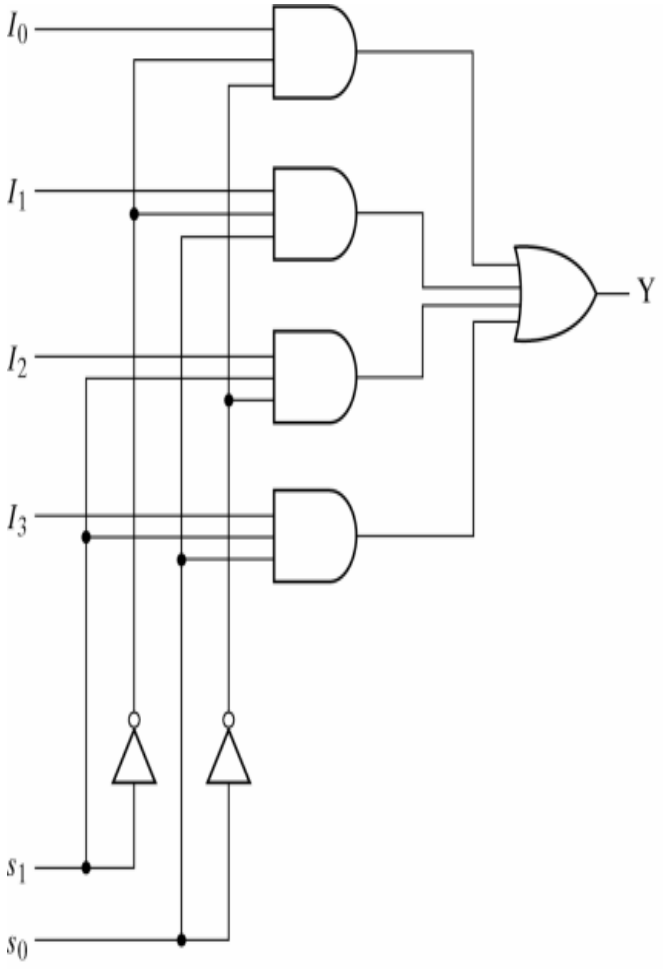


Fig. 4-24 2-to-1-Line Multiplexer

4-to-1LINE MULTIPLEXER DESIGN

In general, a 2^n–to–1- line multiplexer is constructed from an n–to 2^n decoder by adding to it 2^n lines, one to each AND gate.



(a) Logic diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

Fig. 4-25 4-to-1-Line Multiplexer

QUADRUPLE 4-to-1LINE MULTIPLEXER

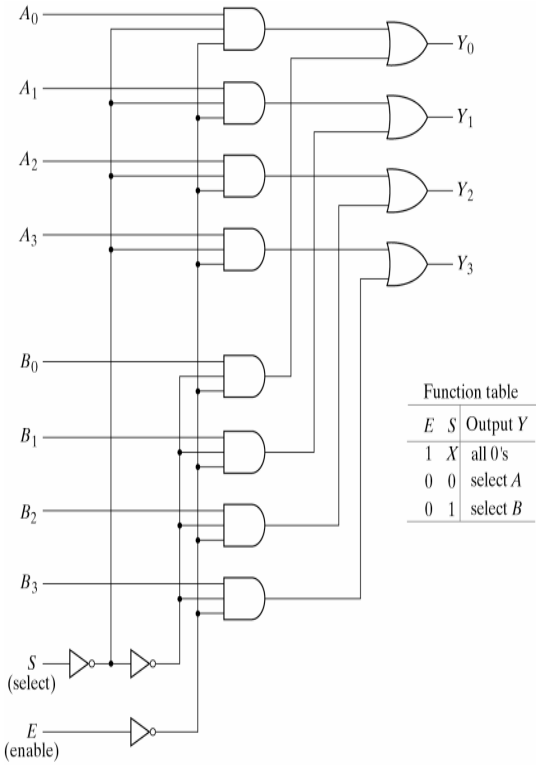


Fig. 4-26 Quadruple 2-to-1-Line Multiplexer

Function implementation using multiplexers

Function with n variables and multiplexer with $n - 1$ selection

$F(x, y, z) = \sum (1,2,6,7)$

Input variables x, y: Selection lines, S_1 and S_0
Variable z: Date line 0
Data lines 1,2,3: z', 0,1

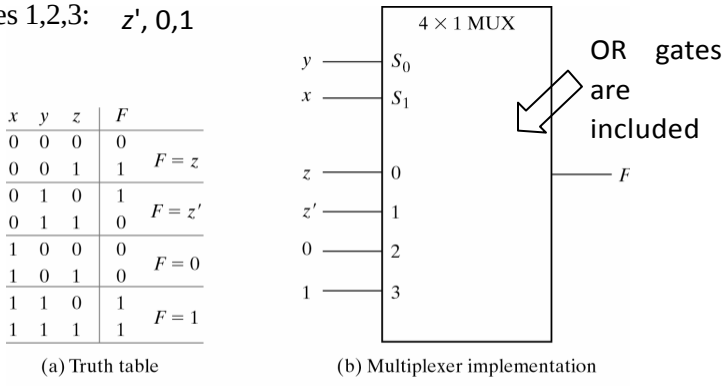
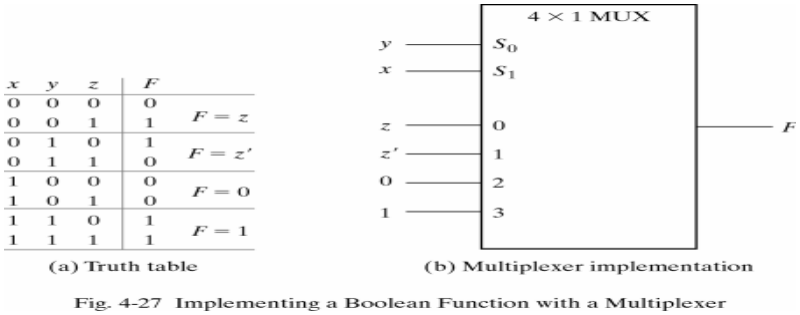


Fig. 4-27 Implementing a Boolean Function with a Multiplexer

Function implementation using 4x1multiplexer

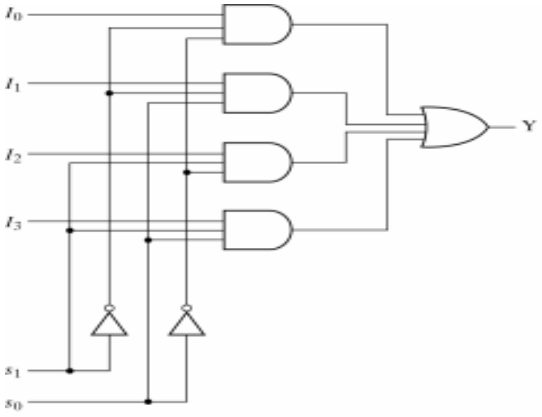


x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table

(b) Multiplexer implementation

Fig. 4-27 Implementing a Boolean Function with a Multiplexer



s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

(a) Logic diagram

Fig. 4-25 4-to-1-Line Multiplexer

Function implementation using 8x1multiplexer

$F(A, B, C, D) = \sum (1,3,4,11,12,13,14,15)$

1. Complete the truth table from the SOP.
2. The first $n - 1$ variables in the table are applied to the selection inputs of the multiplexer.
3. For each combination of the selection variables, we evaluate the output as a function of the last variable.
4. Apply these values to the data input in proper order.

Function implementation using 8x1 MUX

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

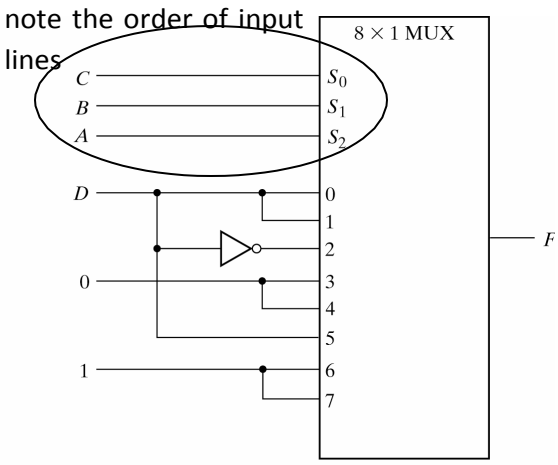


Fig. 4-28 Implementing a 4-Input Function with a Multiplexer

Three State Gates

A three-state gate is a digital circuit that exhibits three states: 0, 1 and a high- impedance (high *z* state). The high impedance state behaves as an open circuit.

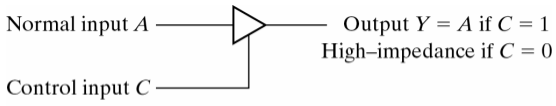
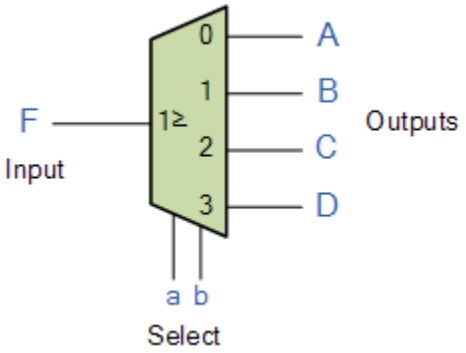


Fig. 4-29 Graphic Symbol for a Three-State Buffer

Because of this feature (high *z* state), a large number of three-state gate outputs can be connected to form a common line without endangering load effects.

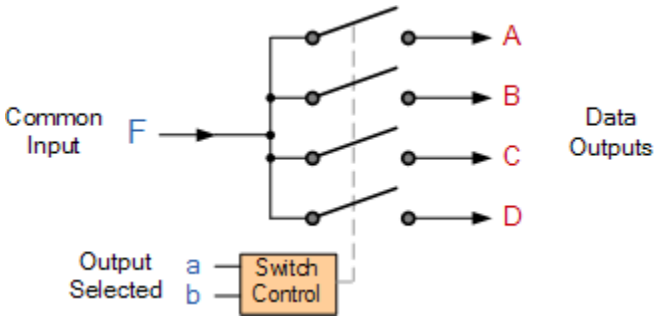
The data distributor, known more commonly as a **Demultiplexer** or "Demux" for short, is the exact opposite of the Multiplexer

Function implementation using 8x1 MUX



The demultiplexer takes one single input data line and then switches it to any one of a number of individual output lines one at a time. The **demultiplexer** converts a serial data signal at the input to a parallel data at its output lines as shown below.

1-to-4 Channel De-multiplexer



Output Select		Data Output Selected
b	a	
0	0	A
0	1	B

1	0	C
1	1	D

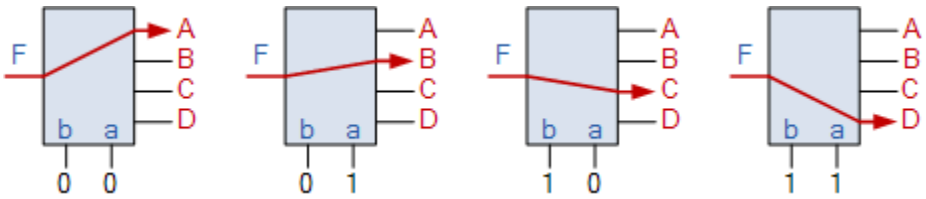
Function implementation using 8x1 MUX

The Boolean expression for this 1-to-4 **Demultiplexer** above with outputs A to D and data select lines a, b is given as:

F = abA + abB + abC + abD

The function of the **Demultiplexer** is to switch one common data input line to any one of the 4 output data lines A to D in our example above. As with the multiplexer the individual solid state switches are selected by the binary input address code on the output select pins "a" and "b" as shown.

Demultiplexer Output Line Selection

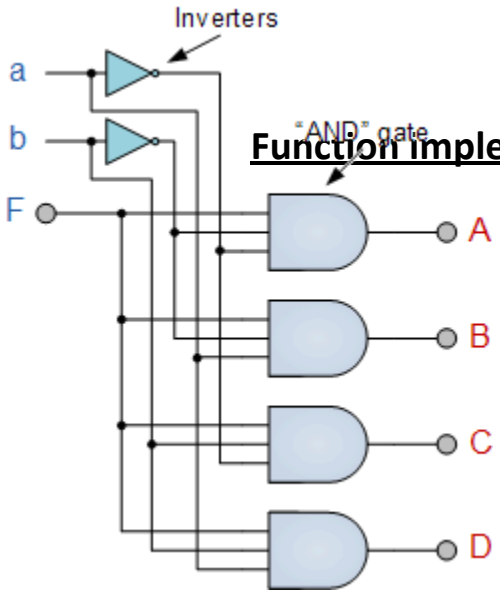


As with the previous **multiplexer circuit**, adding more address line inputs it is possible to switch more outputs giving a 1-to-2^n data line outputs.

Some standard demultiplexer IC´s also have an additional "enable output" pin which disables or prevents the input from being passed to the selected output. Also some have latches built into their outputs to maintain the output logic level after the address inputs have been changed. However, in standard decoder type circuits the address input will determine which single data output will have the same value as the data input with all other data outputs having the value of logic "0".

The implementation of the Boolean expression above using individual logic gates would require the use of six individual gates consisting of AND and NOT gates as shown.

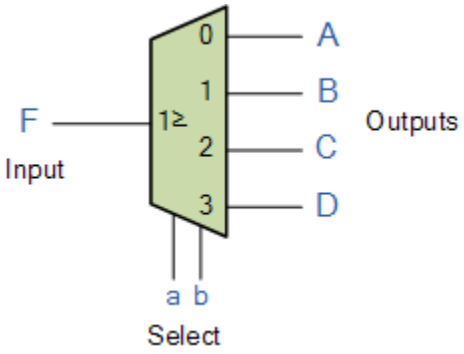
4 Channel Demultiplexer using Logic Gates



Function implementation using 8x1 MUX

The symbol used in logic diagrams to identify a demultiplexer is as follows.

The Demultiplexer Symbol



Again, as with the previous multiplexer example, we can also use the demultiplexer to digitally control the gain of an operational amplifier as shown.

Applications of Demultiplexer:

1. Demultiplexer is used to connect a single source to multiple destinations. The main application area of demultiplexer is communication system where multiplexer are used. Most of the communication system are bidirectional i.e. they function in both ways (transmitting and receiving signals). Hence, for most of the applications, the multiplexer and demultiplexer work in sync. Demultiplexer are also used for reconstruction of parallel data and ALU circuits.
2. **Communication System** – Communication system use multiplexer to carry multiple data like audio, video and other form of data using a single line for transmission. This process make the transmission easier. The demultiplexer receive the output signals of the multiplexer and converts them back to the original form of the

data at the receiving end. The multiplexer and demultiplexer work together to carry out the process of transmission and reception of data in communication system.

3. **ALU (Arithmetic Logic Unit)** – In an ALU circuit, the output of ALU can be stored in multiple registers or storage units with the help of demultiplexer. The output of ALU is fed as the data input to the demultiplexer. Each output of demultiplexer is connected to multiple register which can be stored in the registers.
4. **Serial to parallel converter** – A serial to parallel converter is used for reconstructing parallel data from incoming serial data stream. In this technique, serial data from the incoming serial data stream is given as data input to the demultiplexer at the regular intervals. A counter is attach to the control input of the demultiplexer. This counter directs the data signal to the output of the demultiplexer where these data signals are stored. When all data signals have been stored, the output of the demultiplexer can be retrieved and read out in parallel.

Encoder

An encoder is a circuit that changes a set of signals into a code. Let's begin making a 2-to-1 line encoder truth table by reversing the 1-to-2 decoder truth table.

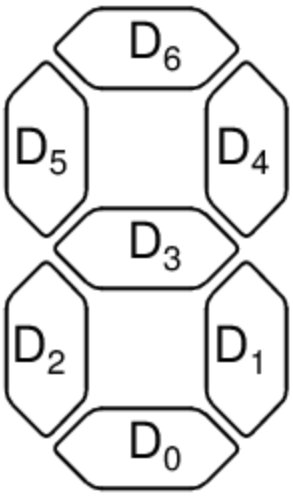
D_1	D_0	A
0	1	0
1	0	1

This truth table is a little short. A complete truth table would be

D_1	D_0	A
0	0	
0	1	0
1	0	1
1	1	

One question we need to answer is what to do with those other inputs? Do we ignore them? Do we have them generate an additional error output? In many circuits this problem is solved by adding sequential logic in order to know not just what input is active but also which order the inputs became active.

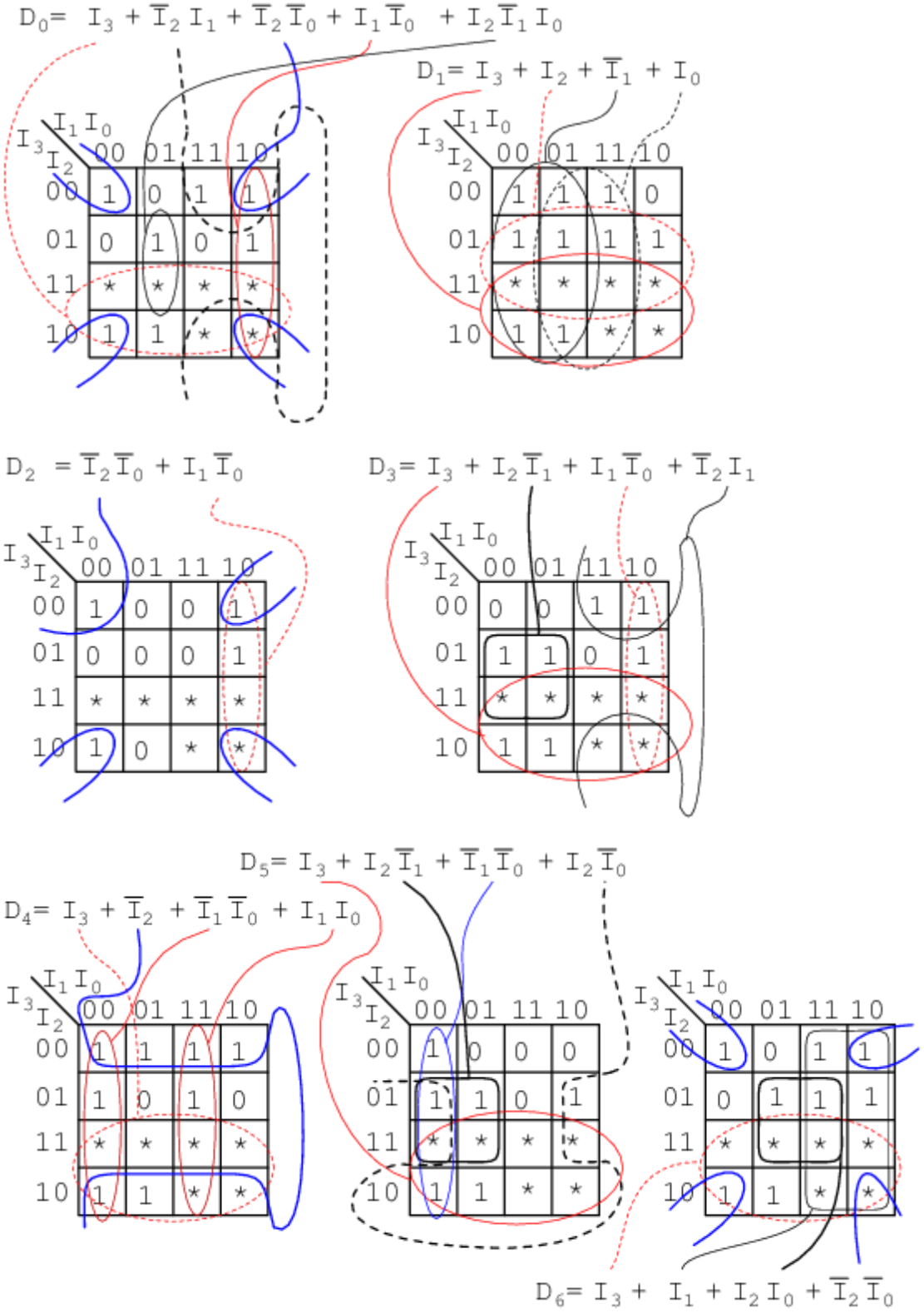
A more useful application of combinational encoder design is a binary to 7-segment encoder. The seven segments are given according



Our truth table is:

I_3	I_2	I_1	I_0	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	1	1	1	0	1	1	1
0	0	0	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0	1
0	0	1	1	1	0	1	1	0	1	1
0	1	0	0	0	1	1	1	0	1	0
0	1	0	1	1	1	0	1	0	1	1
0	1	1	0	1	1	0	1	1	1	1
0	1	1	1	1	0	1	0	0	1	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

Deciding what to do with the remaining six entries of the truth table is easier with this circuit. This circuit should not be expected to encode an undefined combination of inputs, so we can leave them as “don’t care” when we design the circuit. The equations were simplified with karnaugh maps.



The collection of equations is summarised here:

$$D_0 = I_3 + \overline{I}_2 I_1 + \overline{I}_2 \overline{I}_0 + I_1 \overline{I}_0 + I_2 \overline{I}_1 I_0$$

$$D_1 = I_3 + I_2 + \overline{I}_1 + I_0$$

$$D_2 = \overline{I}_2 \overline{I}_0 + I_1 \overline{I}_0$$

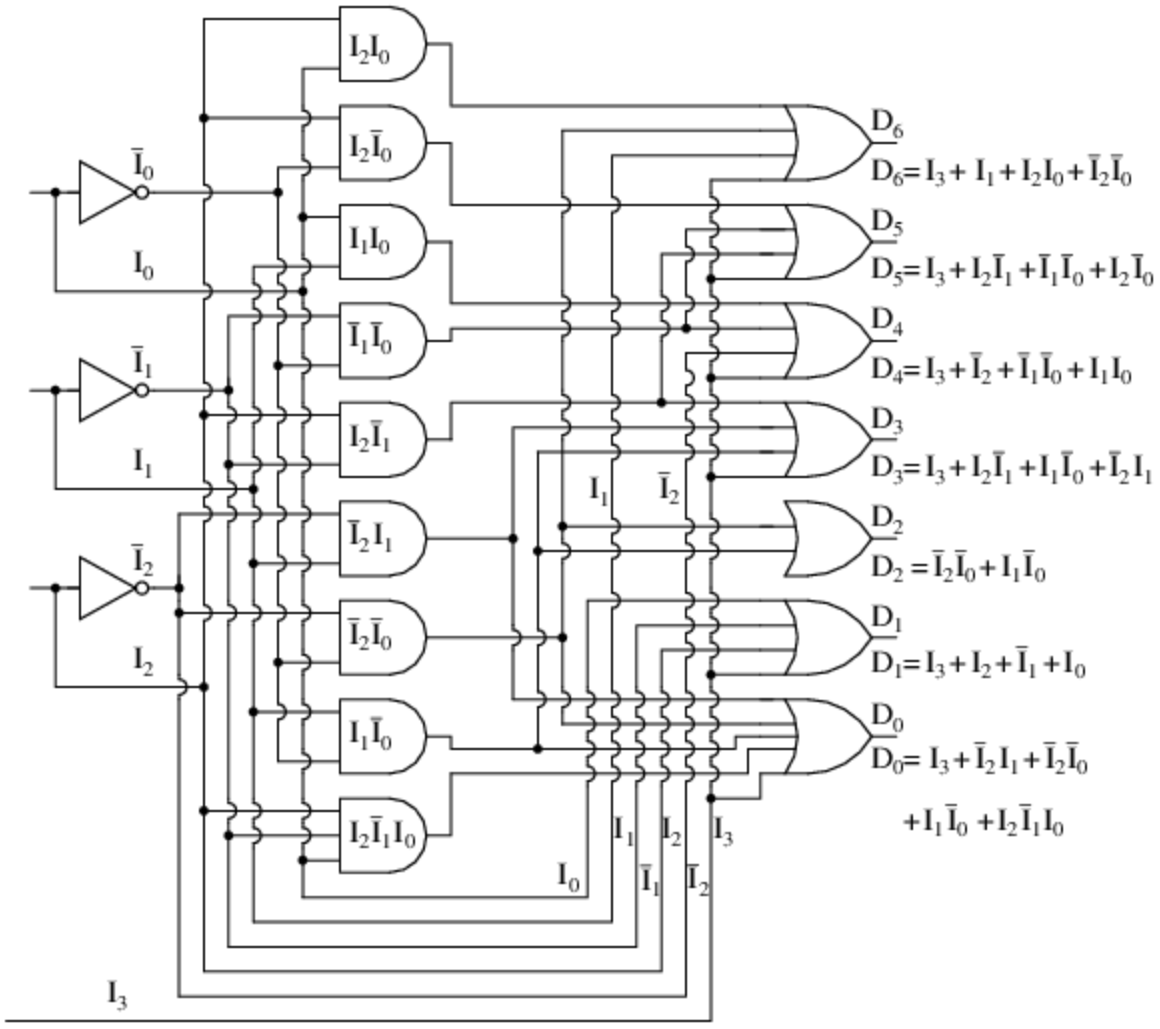
$$D_3 = I_3 + I_2 \overline{I}_1 + I_1 \overline{I}_0 + \overline{I}_2 I_1$$

$$D_4 = I_3 + \overline{I}_2 + \overline{I}_1 \overline{I}_0 + I_1 I_0$$

$$D_5 = I_3 + I_2 \overline{I}_1 + \overline{I}_1 \overline{I}_0 + I_2 \overline{I}_0$$

$$D_6 = I_3 + I_1 + I_2 I_0 + \overline{I}_2 \overline{I}_0$$

The circuit is:



Reference

1. http://www.tutorialspoint.com/computer_logical_organization/combinational_circuits.htm
2. "Digital Logic and Computer design" by M. Morris Mano
3. Textbook of Digital Fundamentals by Thomas L. Floyd (9th Edition)
4. Logic and Computer Design Fundamentals (4th Edition) 4th Edition by M. Morris R.
5. Mano , Charles R. Kime.
6. www.electronicsinourhands.blogspot.com/2012/10/**bcd-adder**.html
7. www.electronics-tutorials.ws › Combinational Logic