Digital Electronics & Logic Design

(EC 207)

Dr. Vivek Garg

Department of Electronics Engineering

S. V. National Institute of Technology (SVNIT)
Surat

email: vivekg@eced.svnit.ac.in; vivekgarg0101@gmail.com

Course Outline

- **PN DIODE AND TRANSITOR** **(04 Hours)**

  PN Diode Theory, PN Characteristic and Breakdown Region, PN Diode Application as Rectifier, Zener Diode Theory, Zener Voltage Regulator, Diode as Clamper and Clipper, Photodiode Theory, LED Theory, 7 Segment LED Circuit Diagram and Multi Colour LED, LASER Diode Theory and Applications, Bipolar Junction Transistor Theory, Transistor Symbols And Terminals, Common Collector, Emitter and Base Configurations, Different Biasing Techniques, Concept of Transistor Amplifier, Introduction to FET Transistor And Its Feature.

- **WAVESHAPING CIRCUITS AND OPERATIONAL AMPLIFIER** **(06 Hours)**

  Linear Wave Shaping Circuits, RC High Pass and Low Pass Circuits, RC Integrator and Differentiator Circuits, Nonlinear Wave Shaping Circuits, Two Level Diode Clipper Circuits, Clamping Circuits, Operational Amplifier OP-AMP with Block Diagram, Schematic Symbol of OP-AMP, The 741 Package Style and Pinouts, Specifications of Op-Amp, Inverting and Non-Inverting Amplifier, Voltage Follower Circuit, Multistage OP-AMP Circuit, OP-AMP Averaging Amplifier, OP-AMP Subtractor.

- **BOOLEAN ALGEBRA AND SWITCHING FUNCTIONS** **(04 Hours)**

  Basic Logic Operation and Logic Gates, Truth Table, Basic Postulates and Fundamental Theorems of Boolean Algebra, Standard Representations of Logic Functions- SOP and POS Forms, Simplification of Switching Functions-K-Map and Quine-Mccluskey Tabular Methods, Synthesis of Combinational Logic Circuits.

- **COMBINATIONAL LOGIC CIRCUIT USING MSI INTEGRATED CIRCUITS** **(07 Hours)**

Course Outline

Binary Parallel Adder; BCD Adder; Encoder, Priority Encoder, Decoder; Multiplexer and Demultiplexer Circuits; Implementation of Boolean Functions Using Decoder and Multiplexer; Arithmetic and Logic Unit; BCD to 7-Segment Decoder; Common Anode and Common Cathode 7-Segment Displays; Random Access Memory, Read Only Memory And Erasable Programmable ROMS; Programmable Logic Array (PLA) and Programmable Array Logic (PAL).

- **INTRODUCTION TO SEQUENTIAL LOGIC CIRCUITS** **(04 Hours)**

  Basic Concepts of Sequential Circuits; Cross Coupled SR Flip-Flop Using NAND or NOR Gates; JK Flip-Flop Rise Condition; Clocked Flip-Flop; D-Type and Toggle Flip-Flops; Truth Tables and Excitation Tables for Flip-Flops; Master Slave Configuration; Edge Triggered and Level Triggered Flip-Flops; Elimination of Switch Bounce using Flip-Flops; Flip-Flops with Preset and Clear.

- **SEQUENTIAL LOGIC CIRCUIT DESIGN** **(06 Hours)**

  Basic Concepts of Counters and Registers; Binary Counters; BCD Counters; Up Down Counter; Johnson Counter, Module-N Counter; Design of Counter Using State Diagrams and Table; Sequence Generators; Shift Left and Right Register; Registers With Parallel Load; Serial-In-Parallel-Out (SIPO) And Parallel-In-Serial-Out(PISO); Register using Different Type of Flip-Flop.

- **REGISTER TRANSFER LOGIC** **(04 Hours)**

  Arithmetic, Logic and Shift Micro-Operation; Conditional Control Statements; Fixed-Point and Floating-Point Data; Arithmetic Shifts; Instruction Code and Design Of Simple Computer.

- **PROCESSOR LOGIC DESIGN** **(03 Hours)**

  Processor Organization; Design of Arithmetic Logic Unit; Design of Accumulator.

- **CONTROL LOGIC DESIGN** **(04 Hours)**

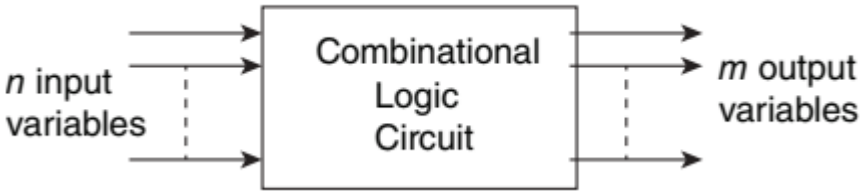
  Control Organization; Hard-Wired Control; Micro Program Control; Control Of Processor Unit; PLA Control.

Course Text and Materials

1. Schilling Donald L. and Belove E., "Electronics Circuits- Discrete and Integrated", 3rd Ed., McGraw-Hill, 1989, Reprint 2008.
2. Millman Jacob, Halkias Christos C. and Parikh C., "Integrated Electronics", 2nd Ed., McGraw-Hill, 2009.
3. Taub H. and Mothibi Suryaprakash, Millman J., "Pulse, Digital and Switching Waveforms", 2nd Ed., McGraw-Hill, 2007.
4. Mano Morris, “Digital Logic and Computer Design”, 5th Ed., Pearson Education, 2005.
5. Lee Samual, “Digital Circuits and Logic Design”, 1st Ed., PHI, 1998.

Digital Logic Circuits

COMBINATIONAL CIRCUITS	SEQUENTIAL CIRCUITS
Output depends only on the present value of the inputs.	Output depends on both the present and previous state values of the inputs
These circuits will not have any memory as their outputs change with the change in the input value.	Sequential circuits have some sort of memory as their output changes according to the previous and present values.
There are no feedbacks involved.	In a sequential circuit the outputs are connected to it as a feedback path.
Used in basic Boolean operations.	Used in the designing of memory devices.
Implemented in: Half adder circuit, full adder circuit, multiplexers, de-multiplexers, decoders and encoders.	Implemented in: RAM, Registers, counters and other state retaining machines.



Design Process

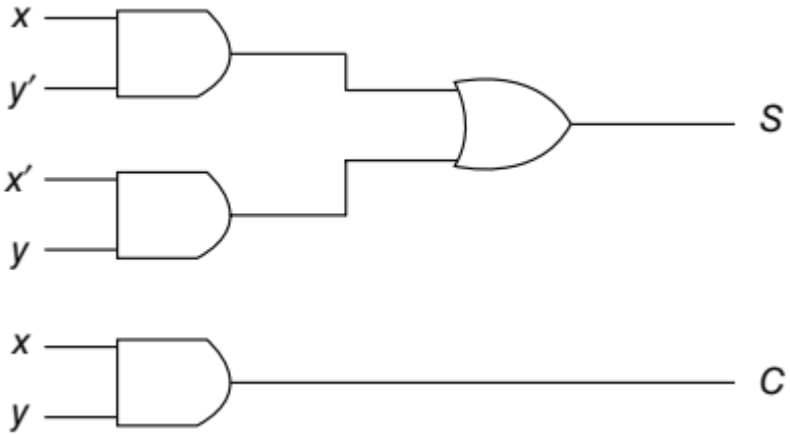
1. The problem is stated.
2. The number of available input variables and required output variables is determined.
3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationships between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.
6. The logic diagram is drawn.

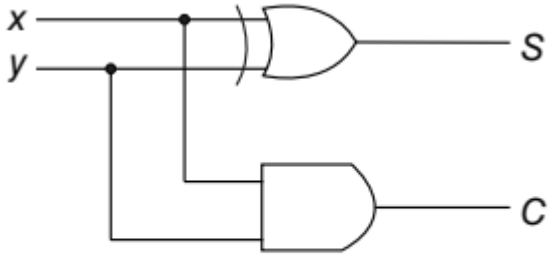
Adders

Half-Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = x'y + xy'$$
$$C = xy$$

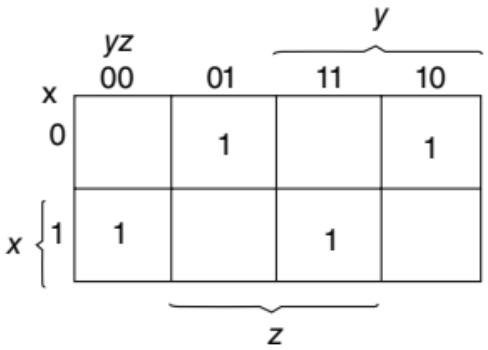




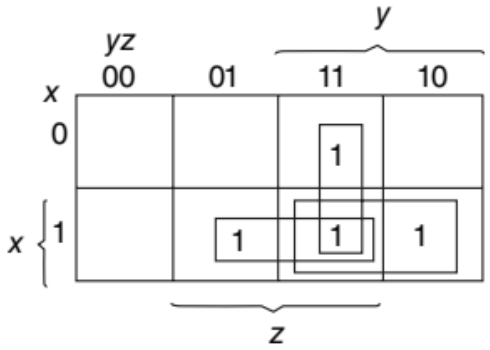
(e) $S = x \oplus y$
$C = xy$

Full-Adder

x	y	z	C	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



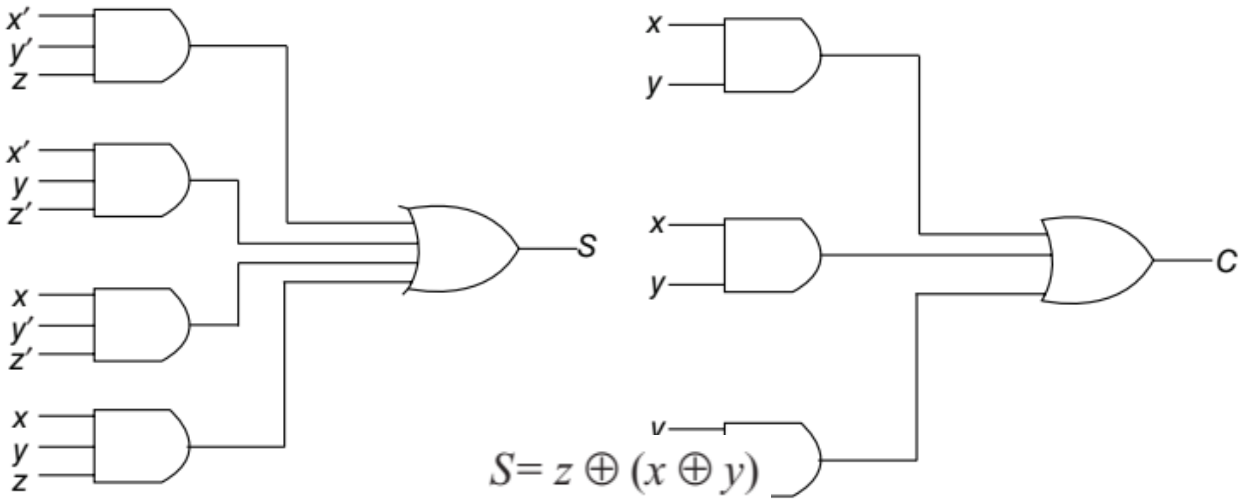
$S = x'y'z + x'yz' + xy'z' + xyz$



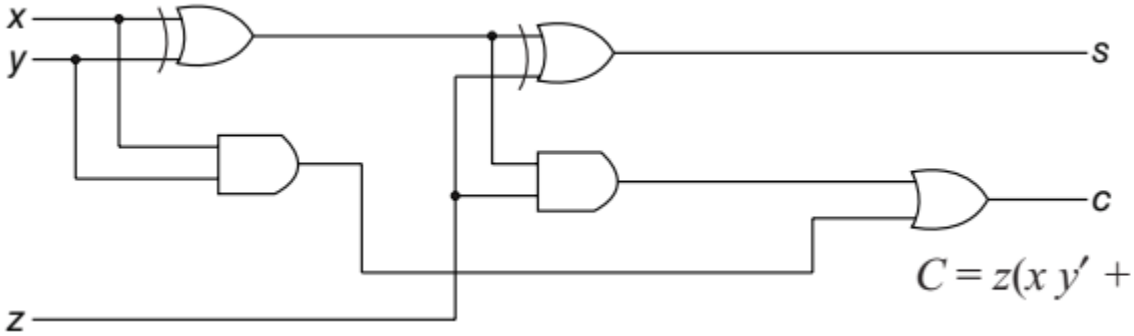
$C = xy + xz + yz$

$$
\begin{aligned}
S &= z \oplus (x \oplus y) \\
&= z'(x\,y' + x'y) + z(x\,y' + x'y)' \\
&= z'(x\,y' + x'y) + z(x\,y + x'y') \\
&= x\,y'z' + x'yz' + x\,yz + x'y'z
\end{aligned}
$$

$$C = z(x\,y' + x'y) + x\,y = x\,y'z + x'yz + x\,y$$



$$S = z \oplus (x \oplus y)$$



$$S = z \oplus (x \oplus y)$$

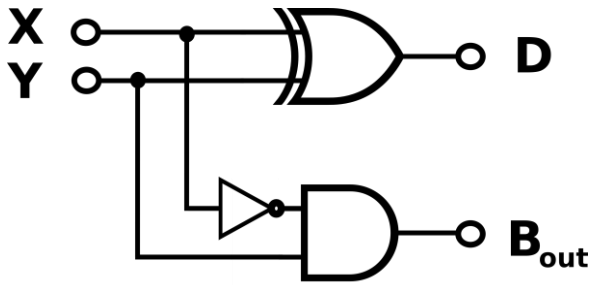
$$C = z(x\,y' + x'y) + x\,y = x\,y'z + x'yz + x\,y$$

Subtractors

Half-Subtractor

x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

$$D = x'y + x\,y'$$
$$B = x'y$$

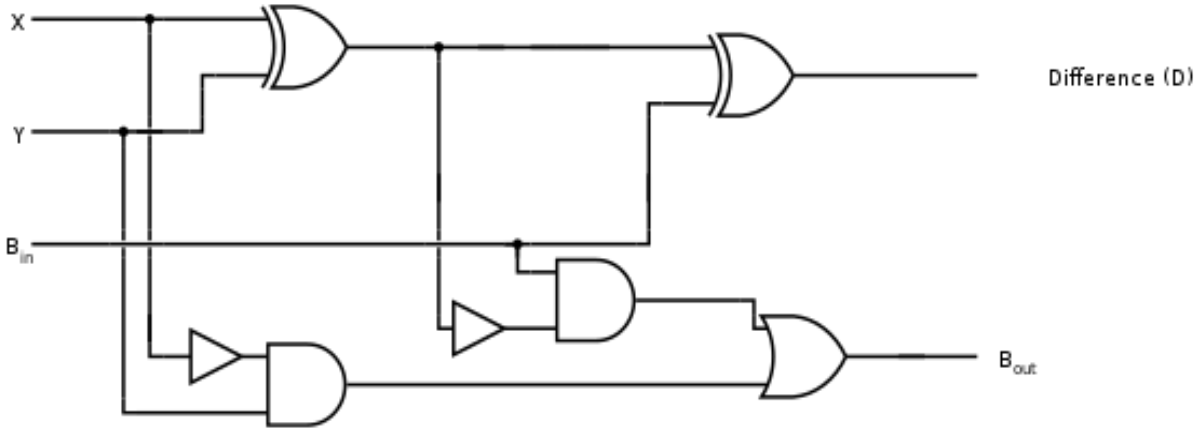


Full-Subtractor

x	y	z	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

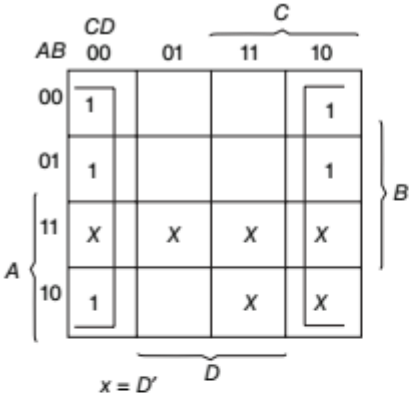
$$D = x'y'z + x'yz' + x\,y'z' + x\,yz$$

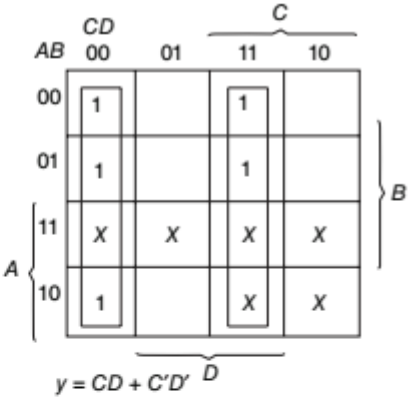
$$B = x'y + x'z + yz$$

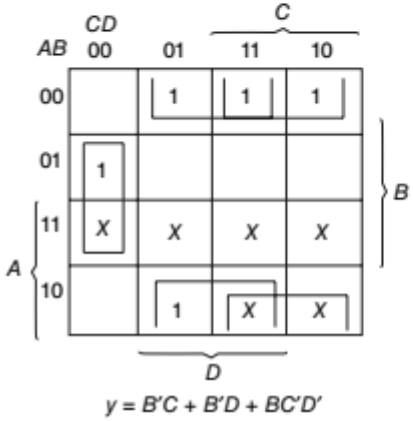


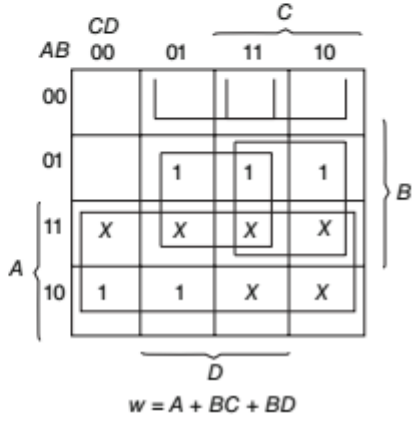
Code Conversion (BCD to Excess 3 code convertor

Input BCD				Output Excess-3 code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
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



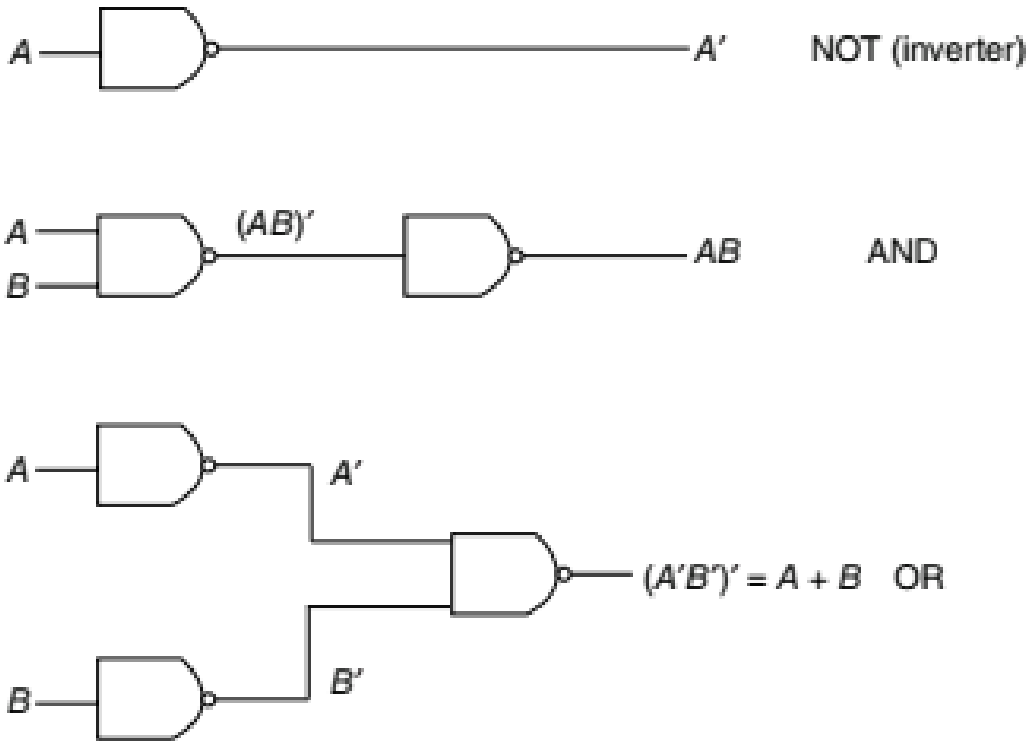






Universal Gates

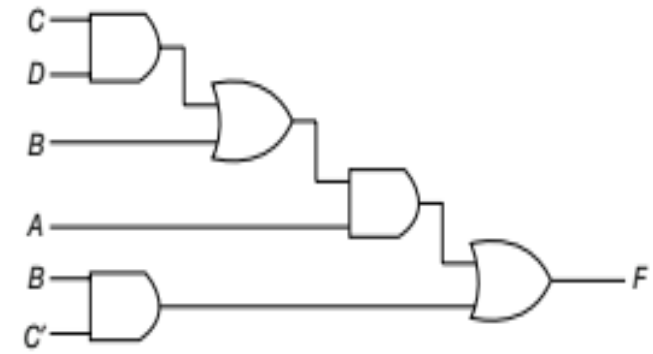
NAND Gate

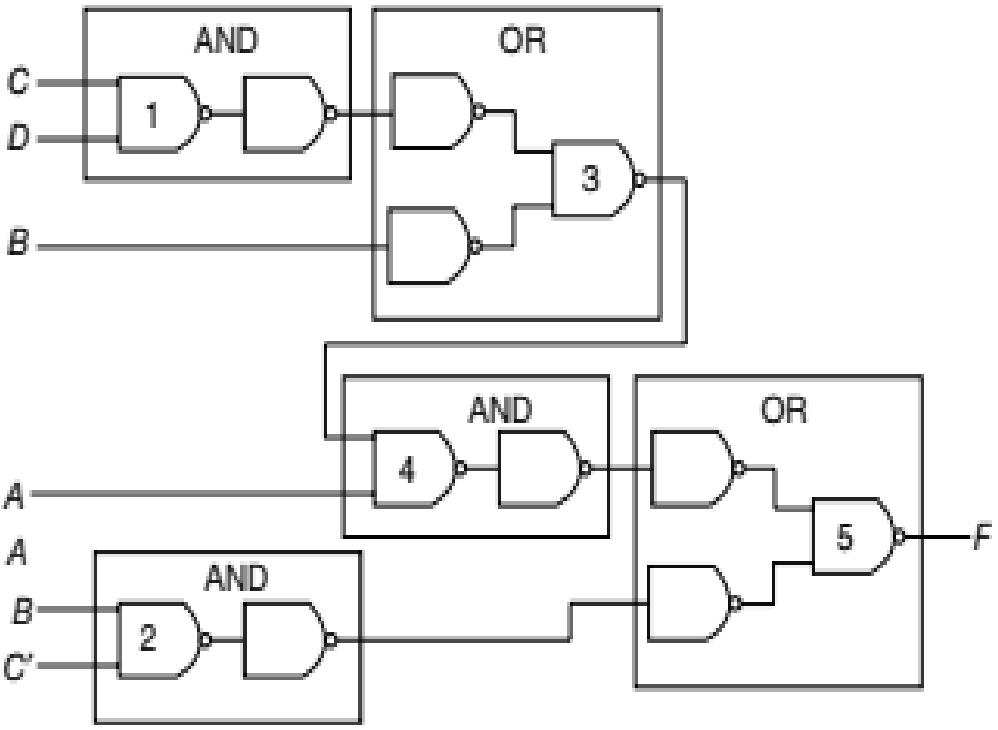


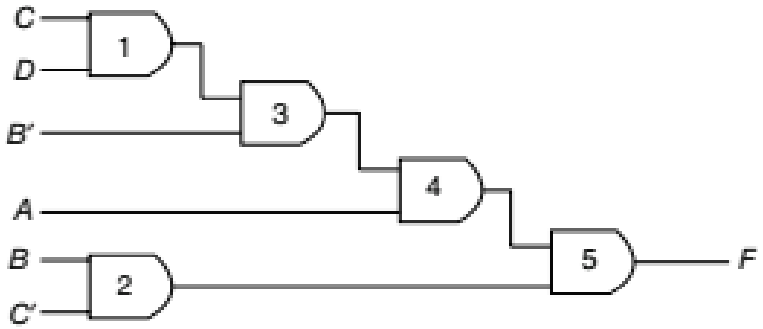
Universal Gates

AND-OR to NAND

$F = A(B + CD) + BC'$

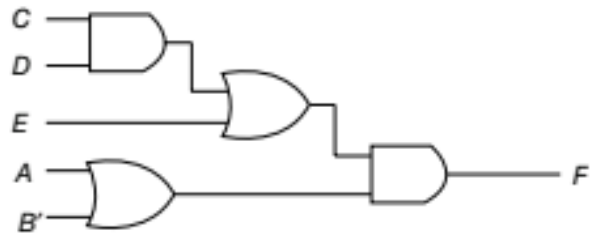


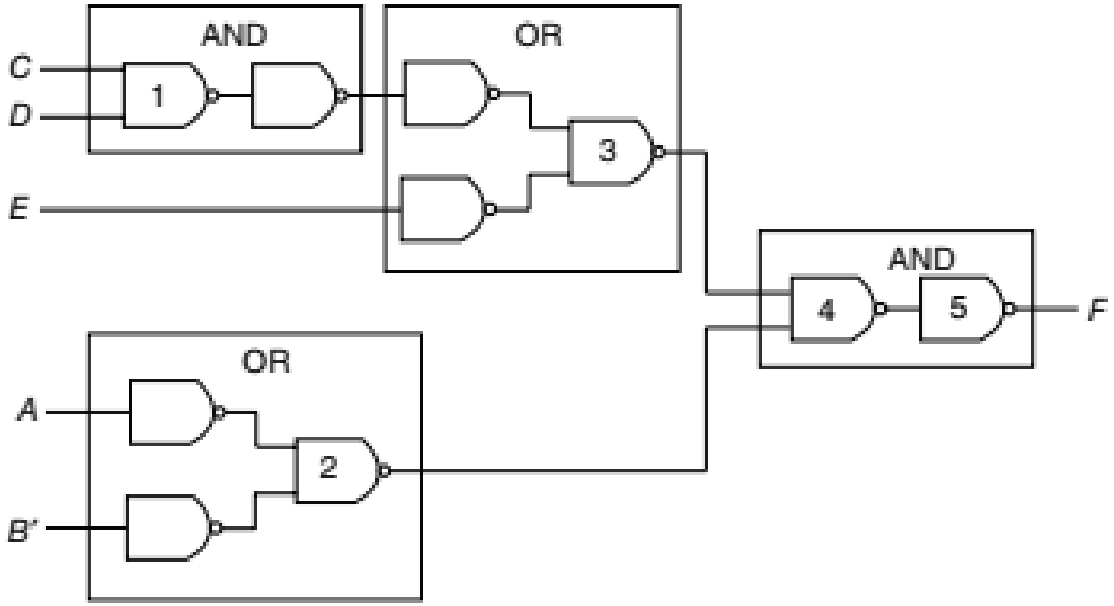


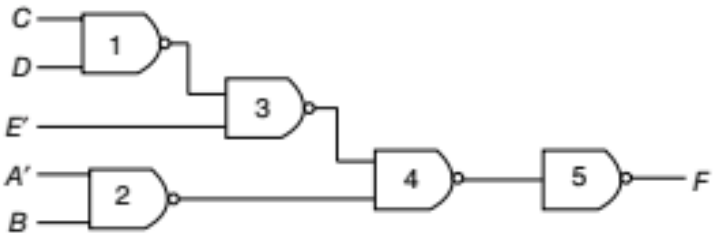


Universal Gates

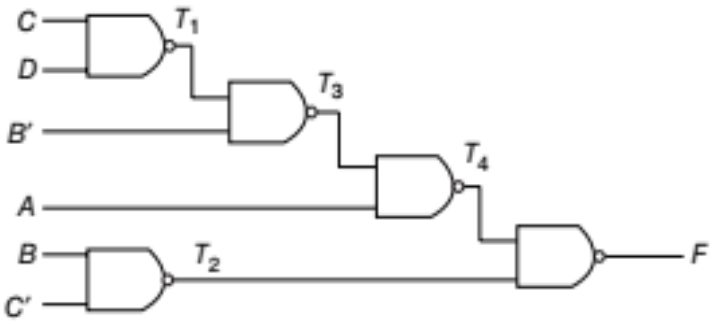
$$F = (A + B')(CD + E)$$







Universal Gates



$T_1 = (CD)' = C' + D'$

$T_2 = (BC)' = B' + C$

$T_3 = (B'T_1)' = (B'C' + B'D')'$

$= (B + C)(B + D) = B + CD$

$T_4 = (AT_3)' = [A(B + CD)]'$

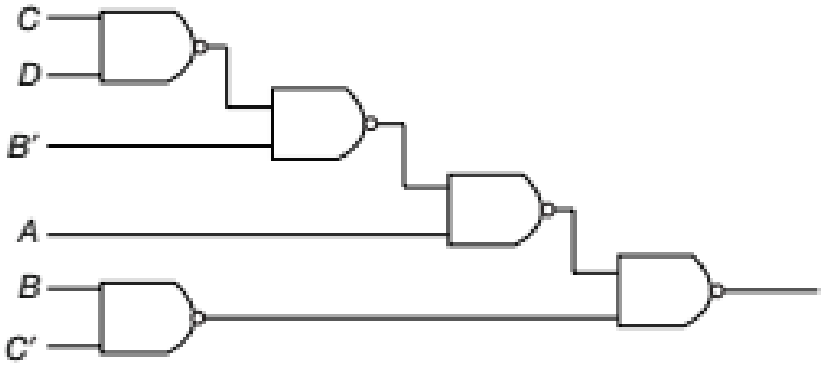
$F = (T_2T_4)' = \{(BC')'[A(B + CD)]'\}'$

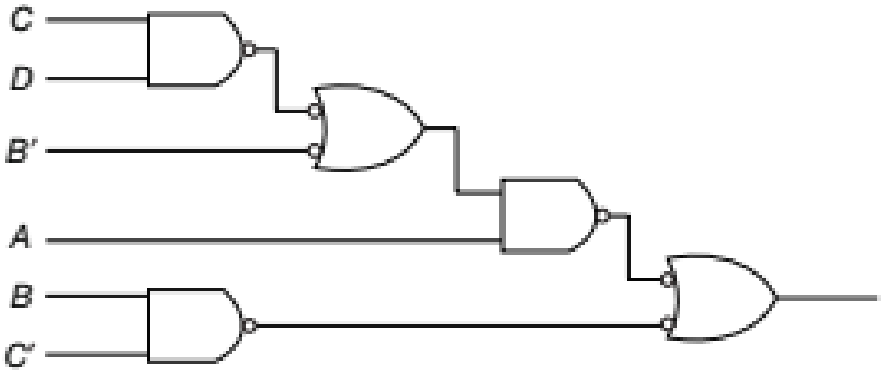
$= BC' + A(B + CD)$

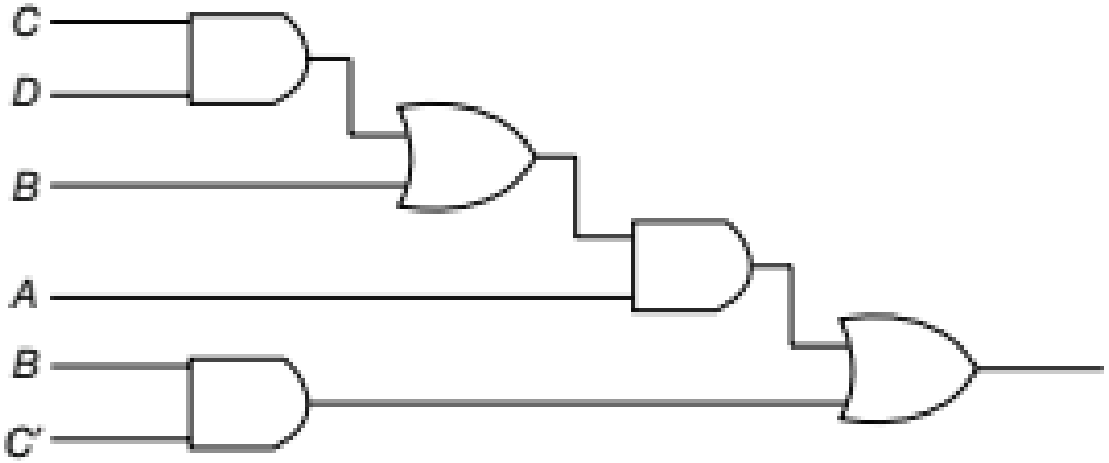
A	B	C	D	T_1	T_2	T_3	T_4	F
0	0	0	0	1	1	0	1	0
0	0	0	1	1	1	0	1	0
0	0	1	0	1	1	0	1	0
0	0	1	1	0	1	1	1	0
0	1	0	0	1	0	1	1	1
0	1	0	1	1	0	1	1	1
0	1	1	0	1	1	1	1	0
0	1	1	1	0	1	1	1	0
1	0	0	0	1	1	0	1	0
1	0	0	1	1	1	0	1	0
1	0	1	0	1	1	0	1	0
1	0	1	1	0	1	1	0	1
1	1	0	0	1	0	1	0	1
1	1	0	1	1	0	1	0	1
1	1	1	0	1	1	1	0	1
1	1	1	1	0	1	1	0	1

Universal Gates

NAND to AND-OR

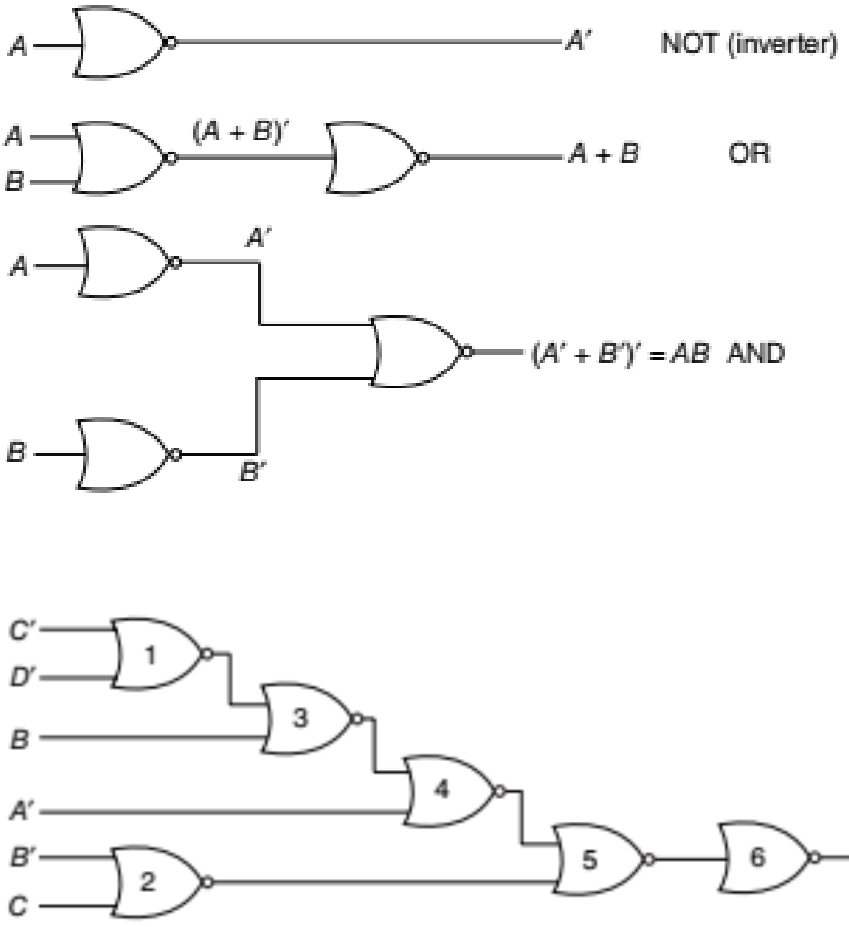


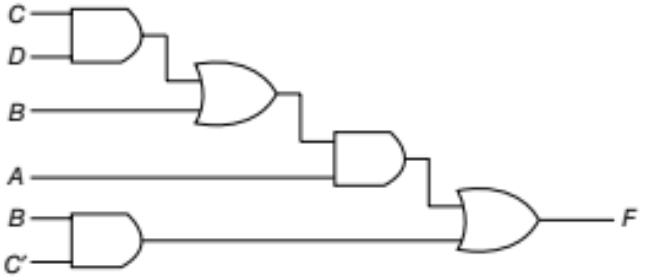


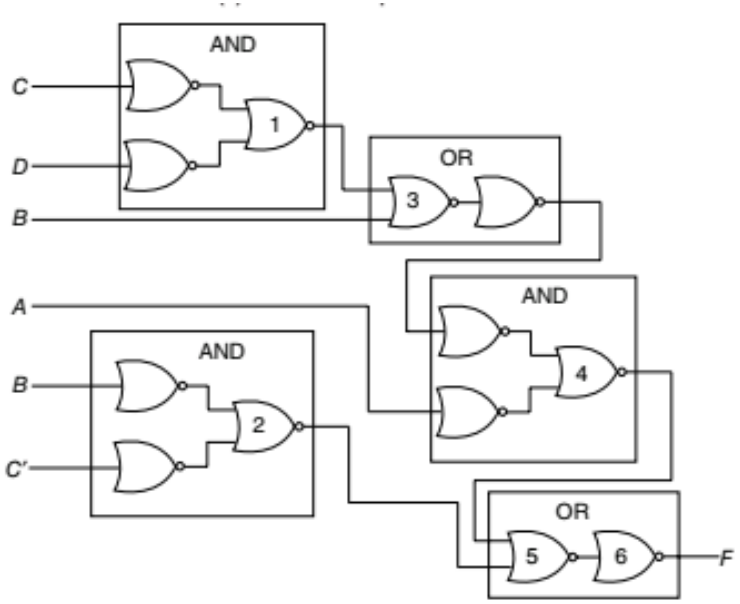


Universal Gates

NOR Gate

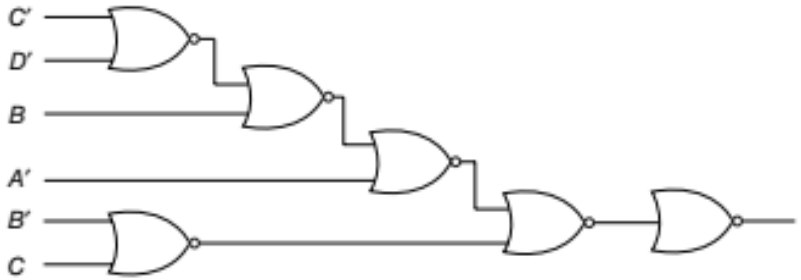


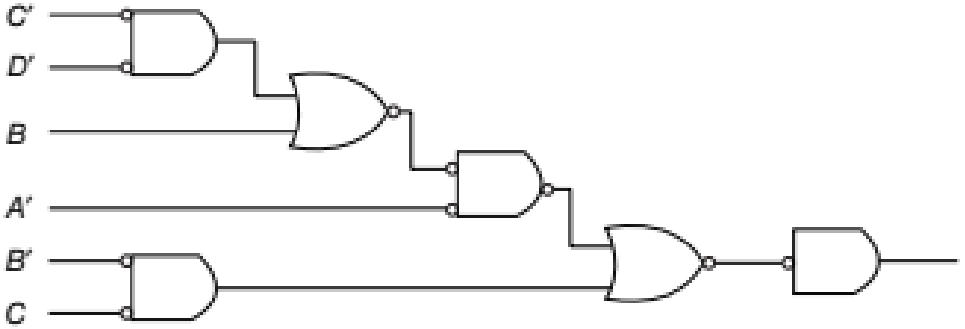


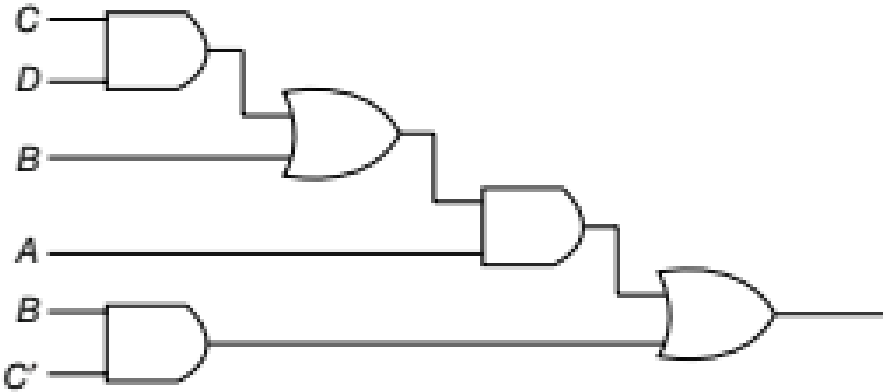


Universal Gates

NOR to AND-OR

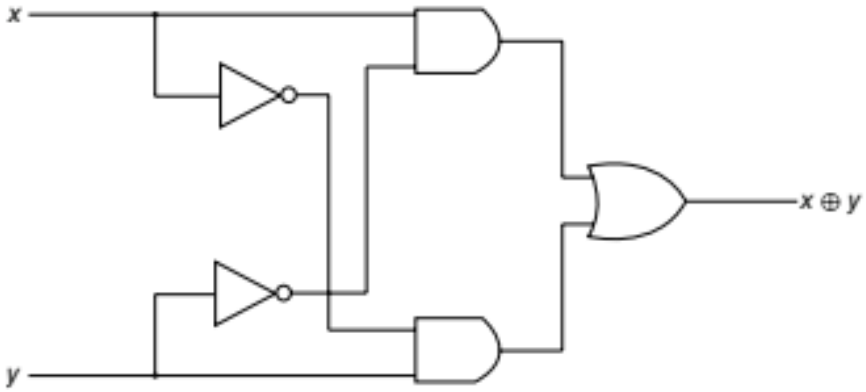


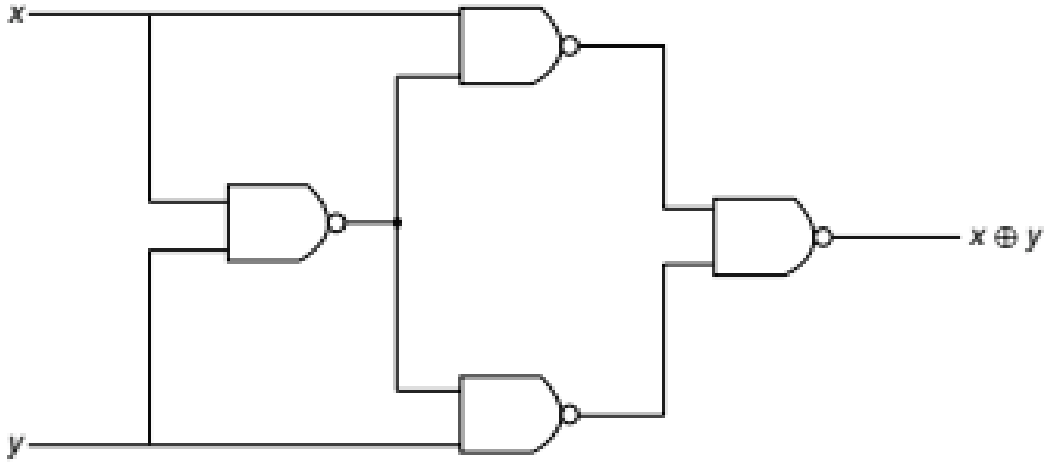




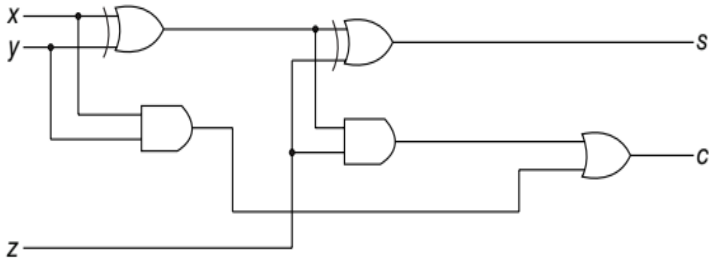
Universal Gates

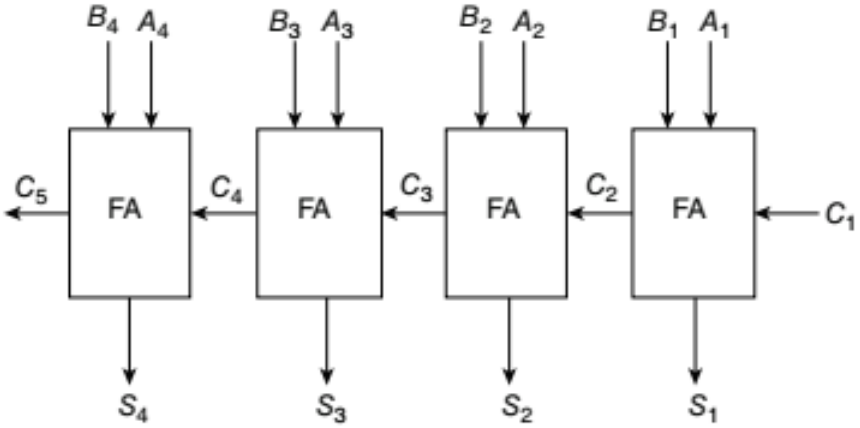
XOR Gate





Binary Parallel Adder





Subscript i	4 3 2 1		*Full-adder*
Input carry	0 1 1 0	C_i	z
Augend	1 0 1 1	A_i	x
Addend	0 0 1 1	B_i	y
Sum	1 1 1 0	S_i	S
Output carry	0 0 1 1	C_{i+1}	C

Combinational Circuits

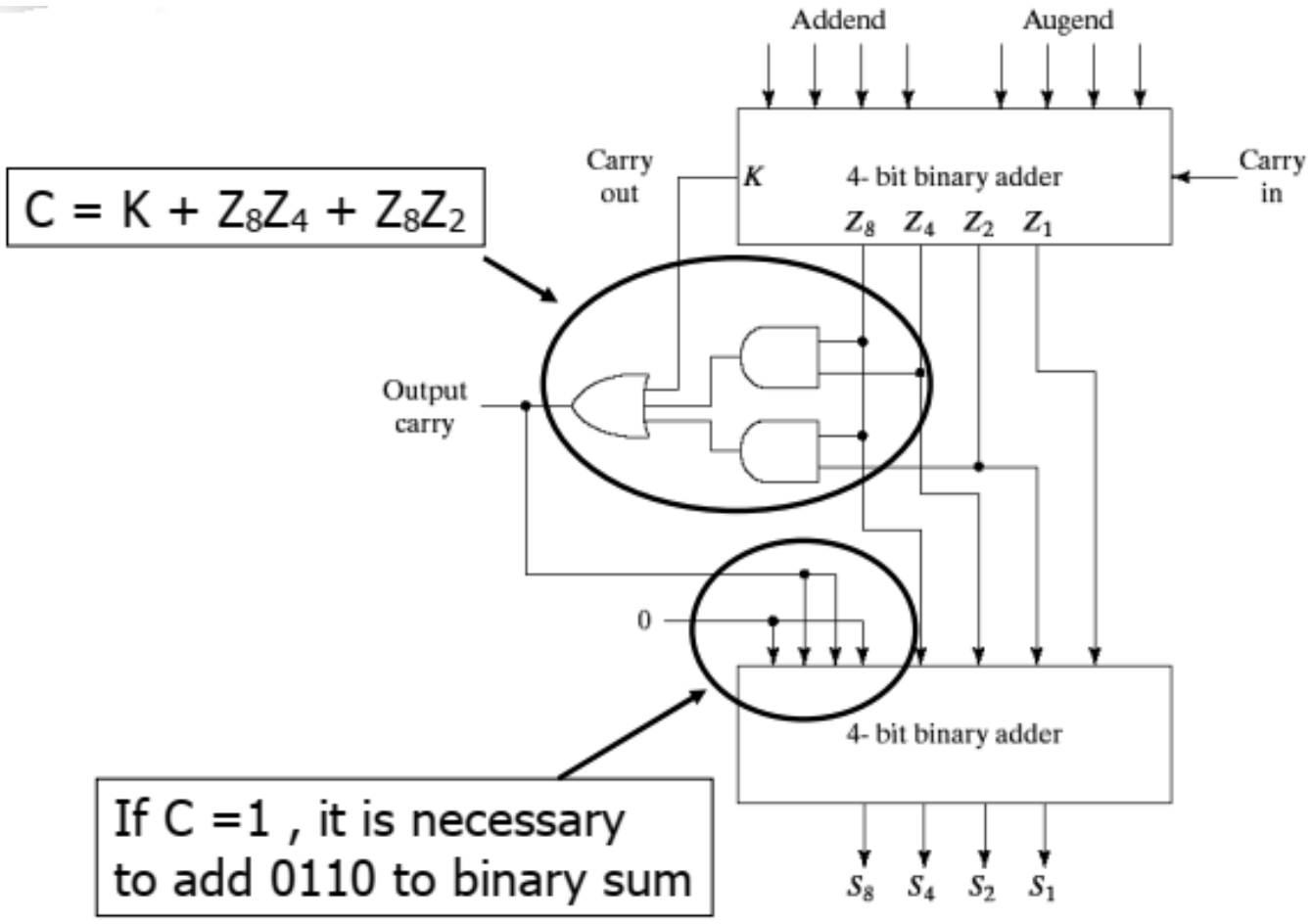
BCD Adder

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	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
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

$$C = K + Z_8Z_4 + Z_8Z_2$$

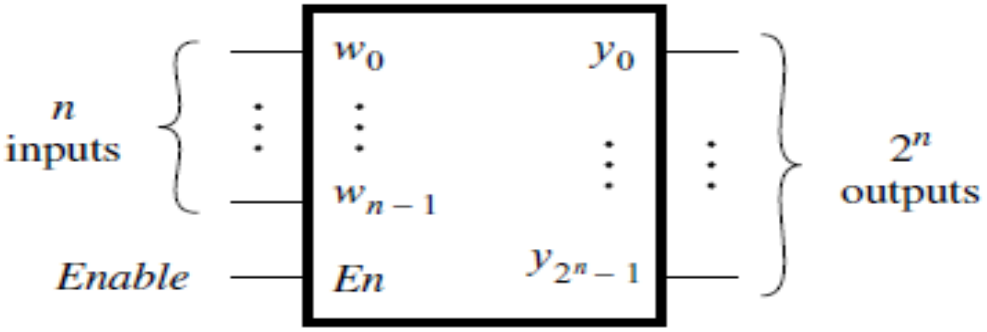
- When the binary sum is equal to or less than 1001_b
  - BCD Sum = Binary Sum
  - C = 0
- When the binary sum is greater than 1001_b
  - BCD Sum = Binary Sum + 0110_b
  - C = 1

BCD Adder



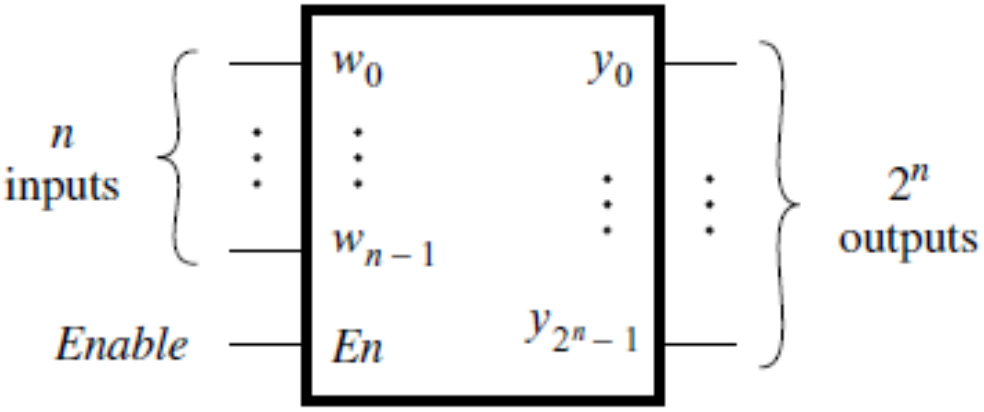
$$C = K + Z_8Z_4 + Z_8Z_2$$

Decoder



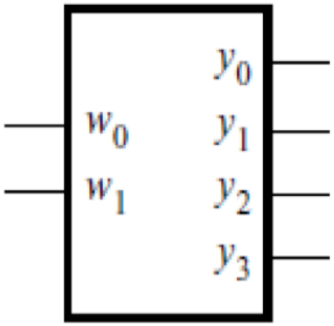
- A circuit that coverts binary information from n input lines to a maximum of 2^n unique output lines
  - May have fewer than 2^n outputs
- A n-to-m-line decoder ($m \leq 2^n$):
  - Generate the m minterns of n input variables
- For each possible input combination, there is only one output that is equal to 1
  - The output whose value is equal to 1 represents the minterm equivalent of the binary number presently available in the input lines

Decoder

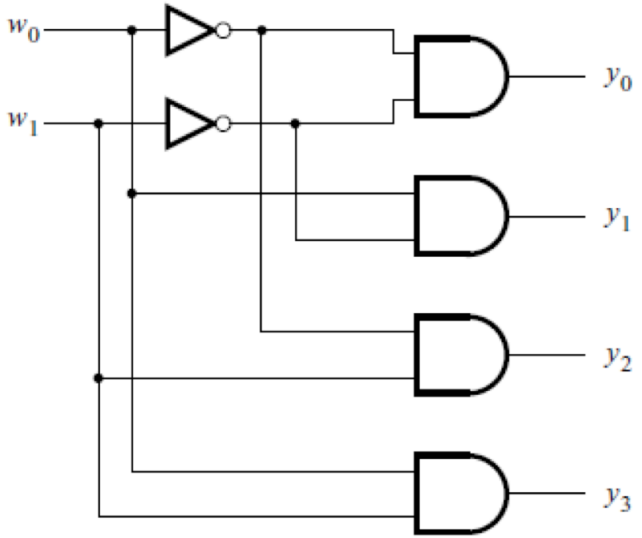


w_1	w_0	y_0	y_1	y_2	y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

(a) Truth table

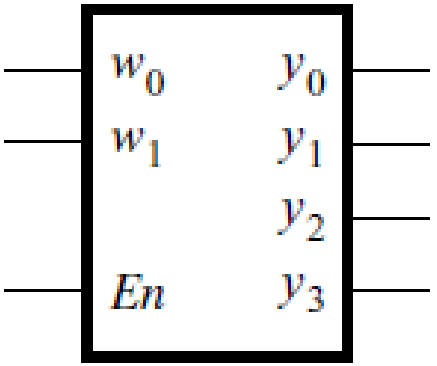


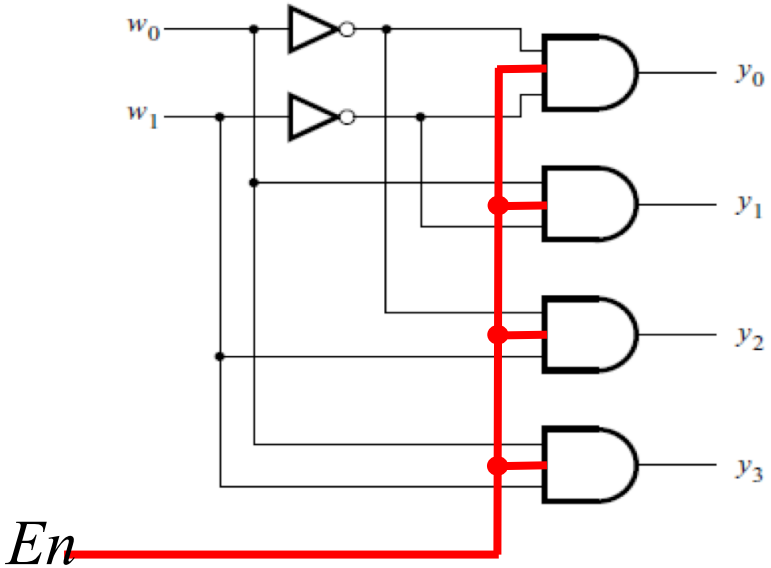
(b) Graphical symbol



Decoder

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

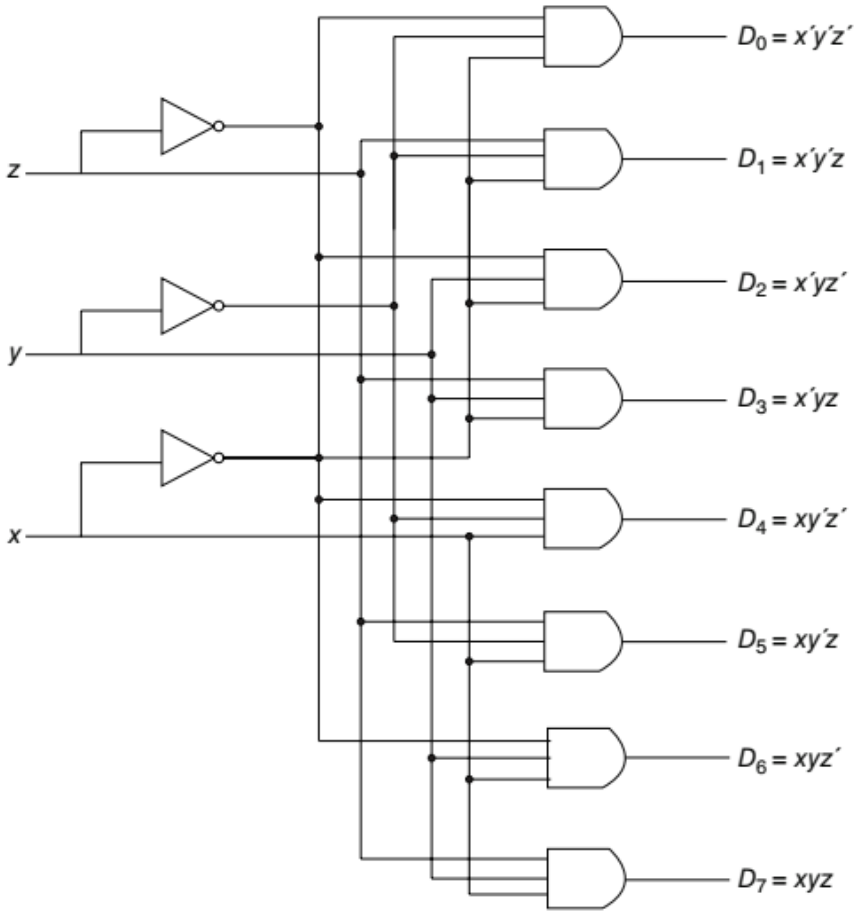




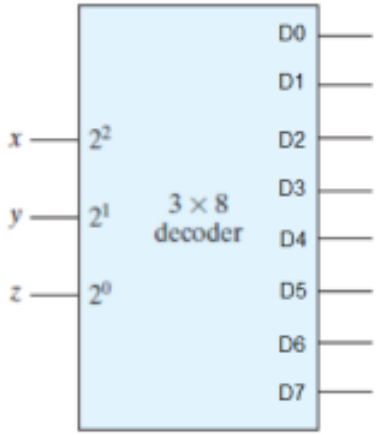
Decoder

- **3-to-8 ?**
- **4-to-16?**
- **5-to-??**

Decoder

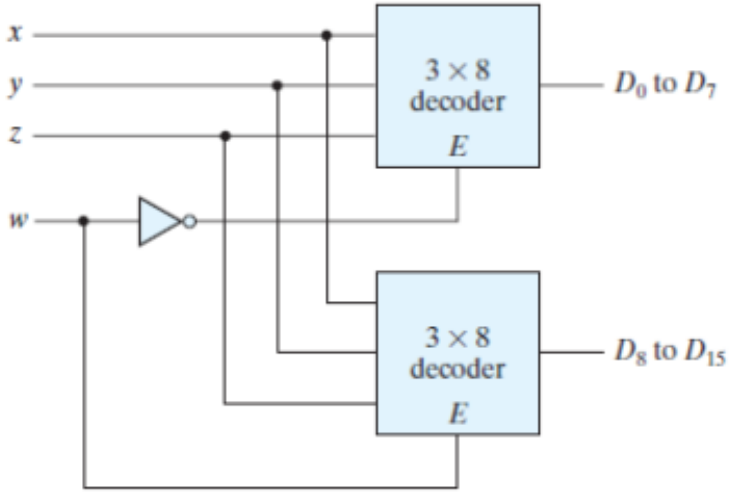


Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Decoder

- 4 × 16 decoder can be constructed using two 3 × 8 decoders.
- Here, we are using *active-high* enable, meaning when E=1 the outputs of the decoder will be valid. The outputs are shown in *positive logic*, meaning the signal on the selected output line is 1 and all others are 0.

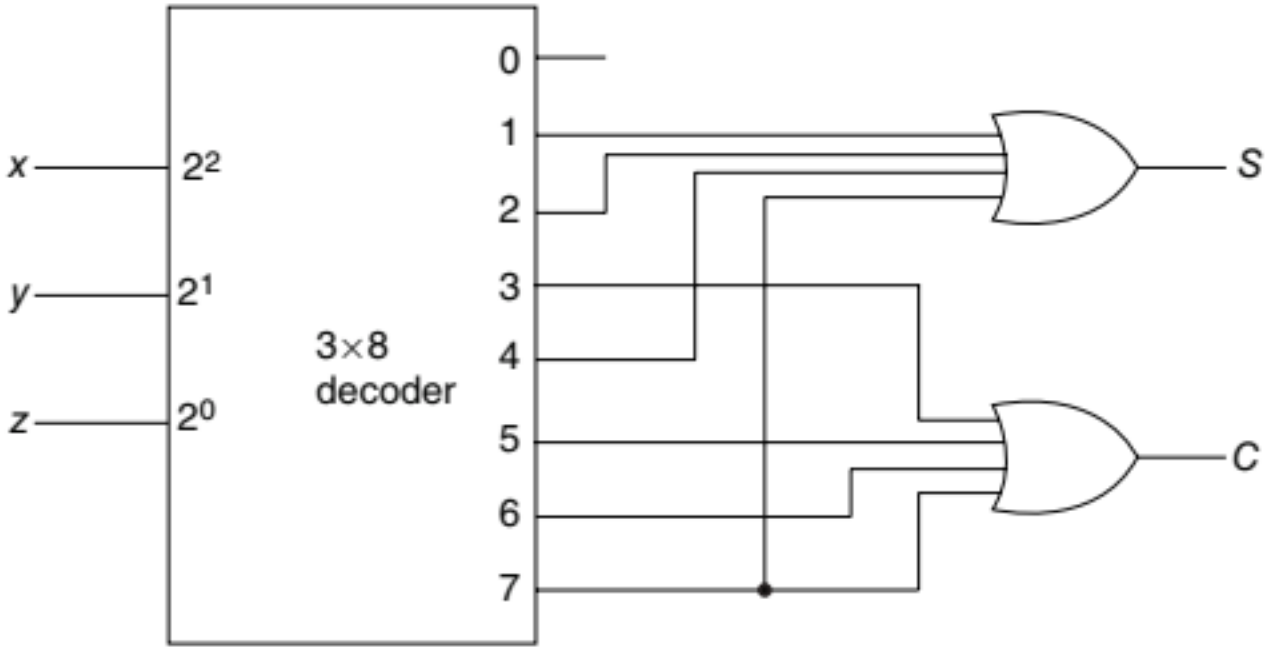


w	x	y	z	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**

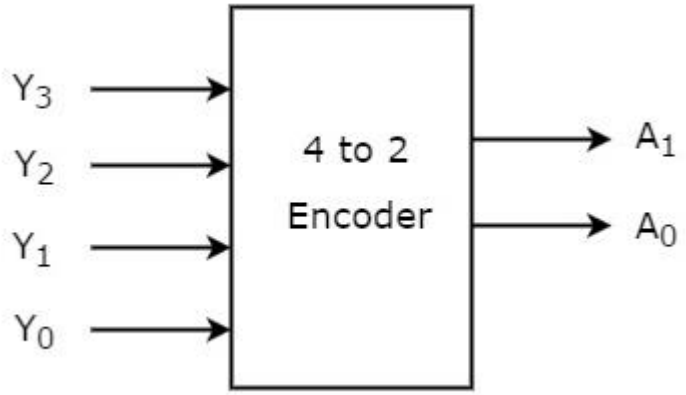
Decoder: Full adder Implementation

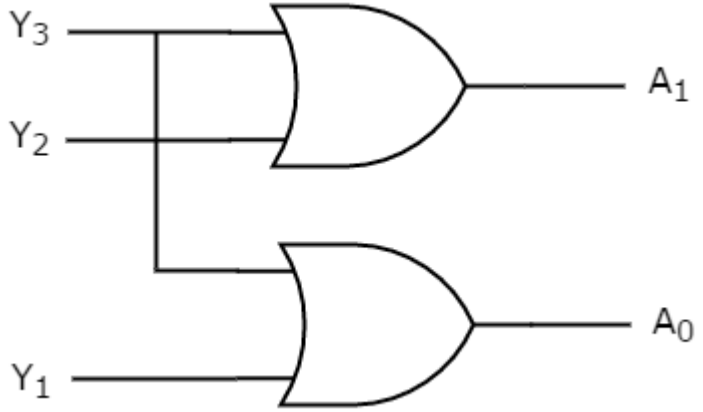
$$S(x, y, z) = \Sigma(1, 2, 4, 7)$$

$$C(x, y, z) = \Sigma(3, 5, 6, 7)$$



Encoder





Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_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

$A_1 = Y_3 + Y_2$

$A_0 = Y_3 + Y_1$

Encoder

- Inverse operation of a decoder.
- Encoder has 2^n or fewer input lines and n output lines.

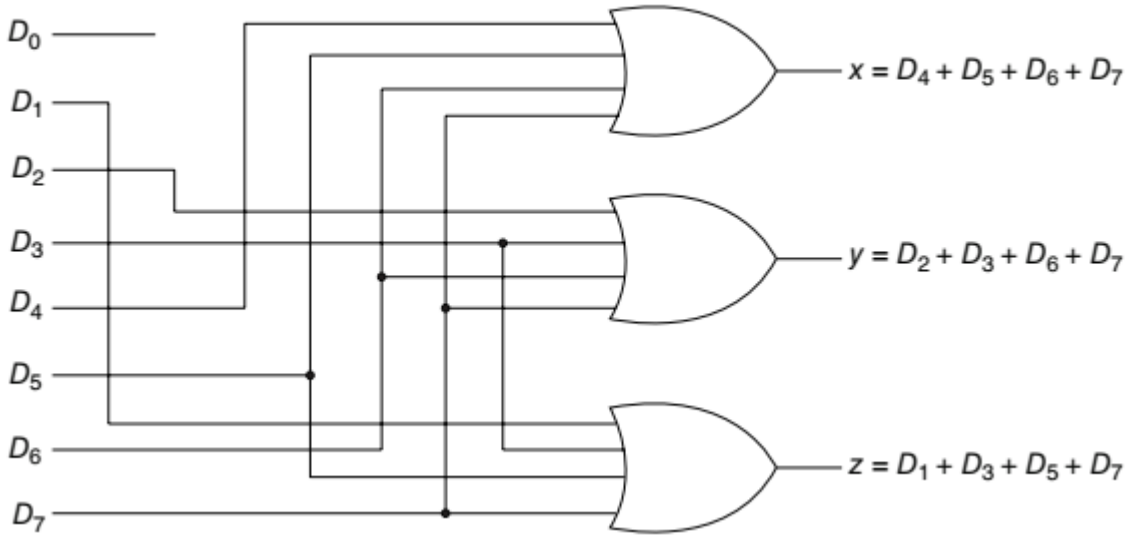
Truth Table of an Octal-to-Binary Encoder

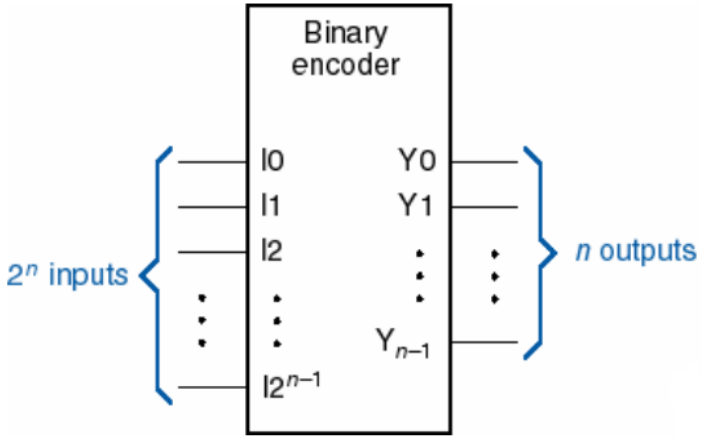
Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

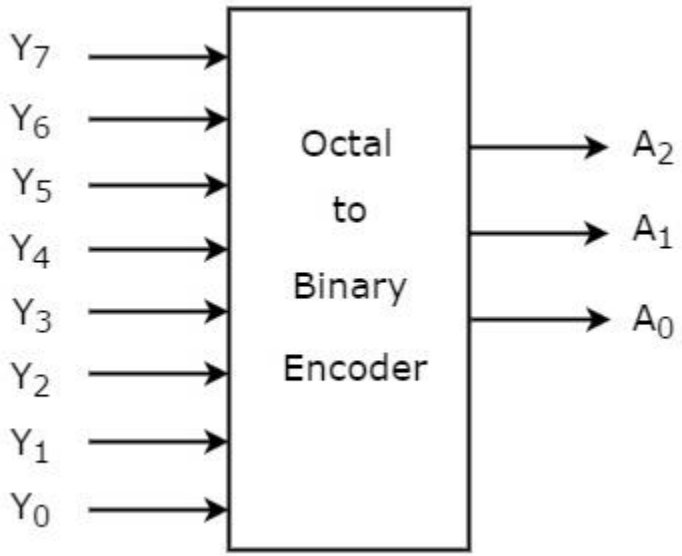
- If two or more inputs are active simultaneously, the output will produce an undefined combination.

Encoder


D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
$x = D_4 + D_5 + D_6 + D_7$
$y = D_2 + D_3 + D_6 + D_7$
$z = D_1 + D_3 + D_5 + D_7$


Binary encoder
I0
I1
I2
$I2^{n-1}$
Y0
Y1
Y_{n-1}
2^n inputs
n outputs

Encoder



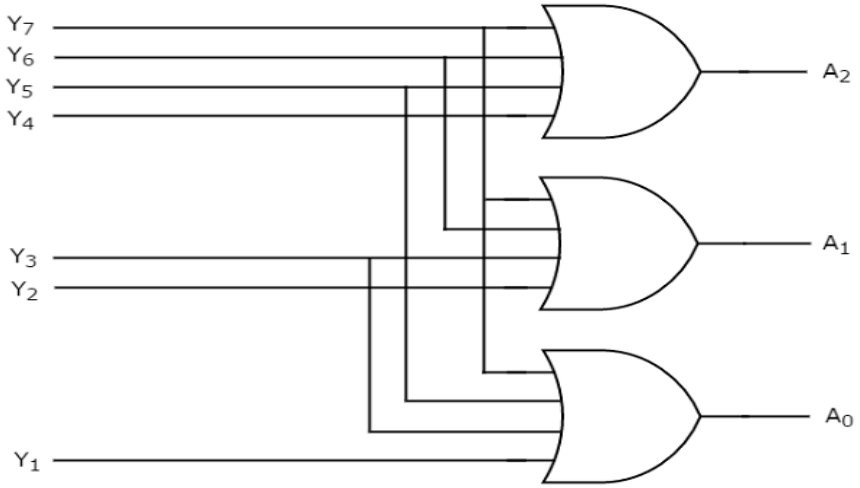
Inputs								Outputs		
Y7	**Y6**	**Y5**	**Y4**	**Y3**	**Y2**	**Y1**	**Y0**	**A2**	**A1**	**A0**
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$$A_2 = Y_7 + Y_6 + Y_5 + Y_4$$

$$A_1 = Y_7 + Y_6 + Y_3 + Y_2$$

$$A_0 = Y_7 + Y_5 + Y_3 + Y_1$$

Encoder



Drawbacks of Encoder

Following are the drawbacks of normal encoder.

• There is an ambiguity, when all outputs of encoder are equal to zero. Because, it could be the code corresponding to the inputs, when only least significant input is one or when all inputs are zero.

• If more than one input is active High, then the encoder produces an output, which may not be the correct code.

Priority Encoder

- Priorities are assigned to each input of encoder.
- Output of encoder will be the binary code corresponding to the active high inputs, which has higher priority.

Inputs				Outputs		
Y_3	Y_2	Y_1	Y_0	A_1	A_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

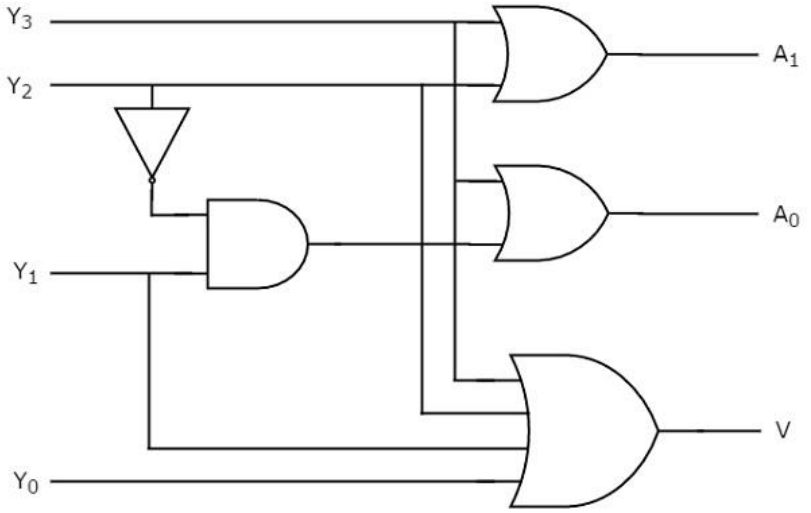
Priority Encoder

K-Map for A_1

Y_3Y_2 \ Y_1Y_0	00	01	11	10	
00					
01	1	1	1	1	Y_2
11	1	1	1	1	
10	1	1	1	1	Y_3

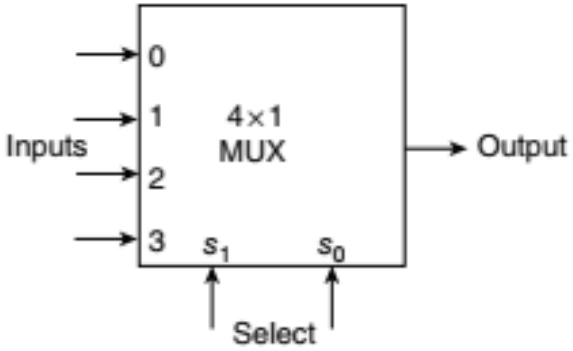
K-Map for A_0

Y_3Y_2 \ Y_1Y_0	00	01	11	10	
00			1	1	$Y_2'Y_1$
01					
11	1	1	1	1	Y_3
10	1	1	1	1	

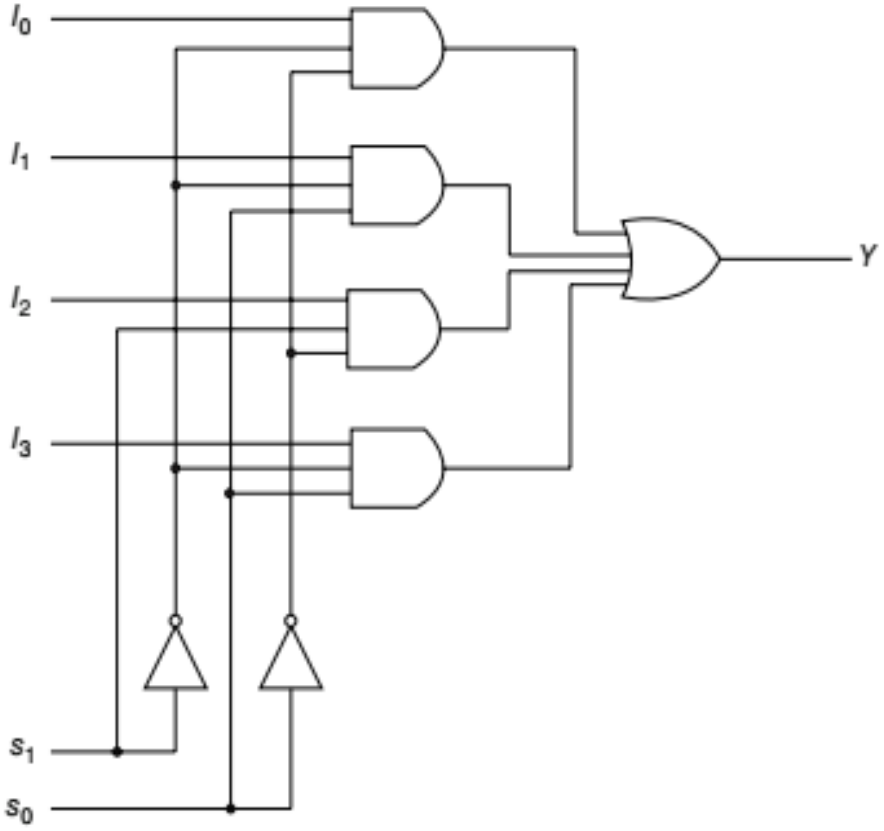


Multiplexers

- *A multiplexer* is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line



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

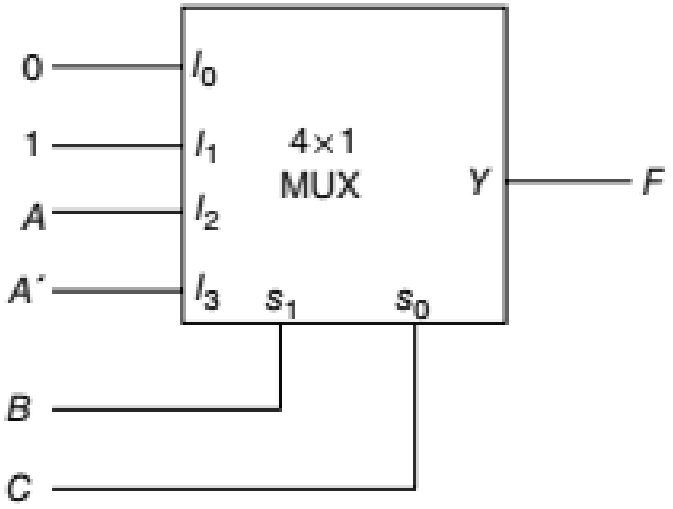


Multiplexers

$F(A, B, C) = \sum(1,3,5,6)$

Minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

	I_0	I_1	I_2	I_3
A'	0	(1)	2	(3)
A	4	(5)	(6)	7
	0	1	A	A'

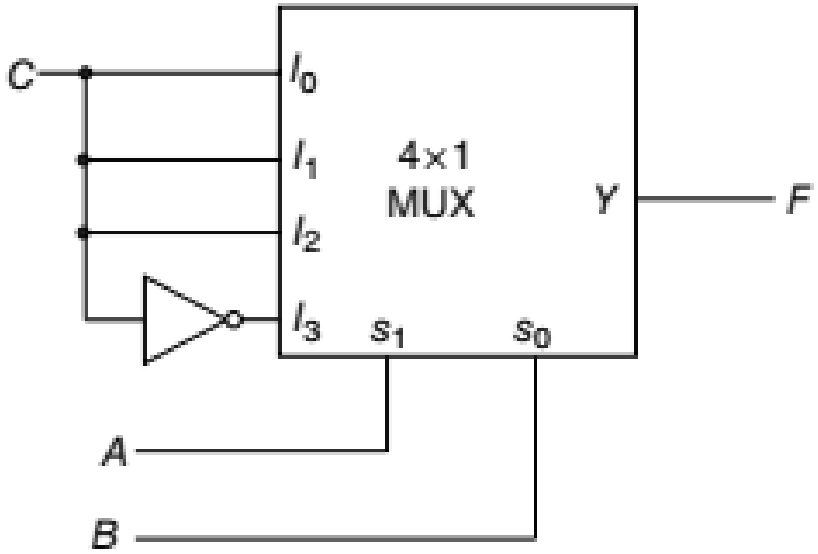


Multiplexers

$F(A, B, C) = \sum(1,3,5,6)$

Minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

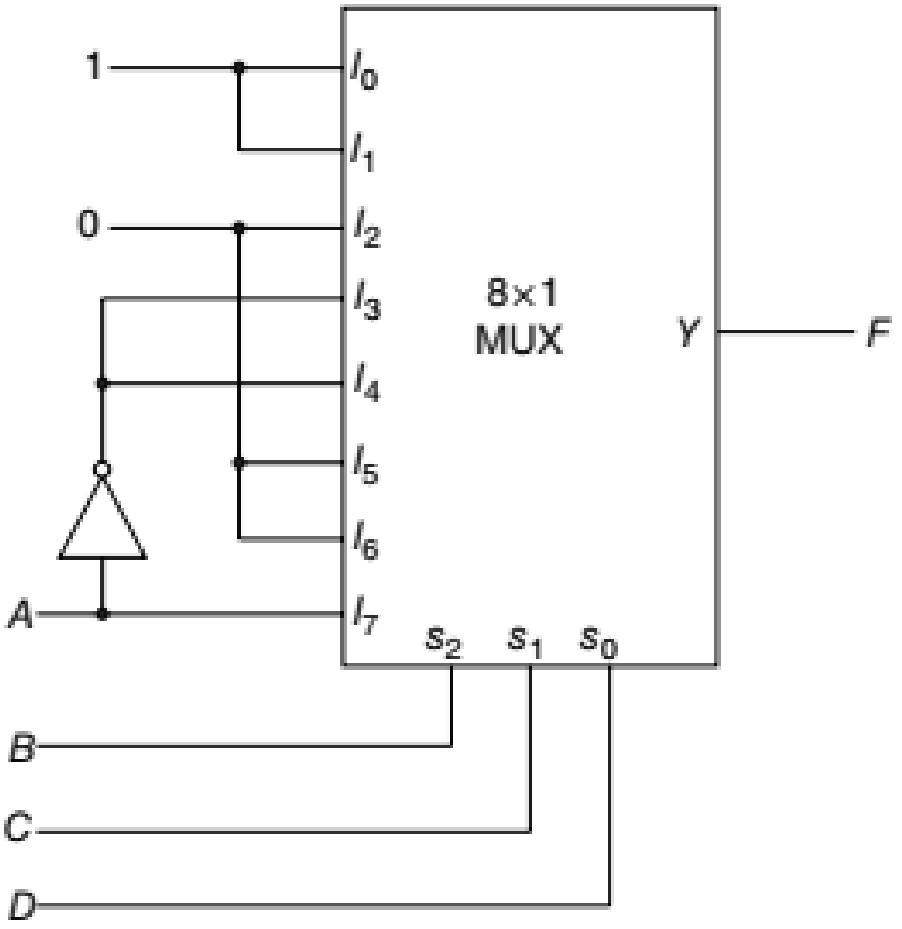
	I_0	I_1	I_2	I_3
C'	0	2	4	(6)
C	(1)	(3)	(5)	7
	C	C	C	C'



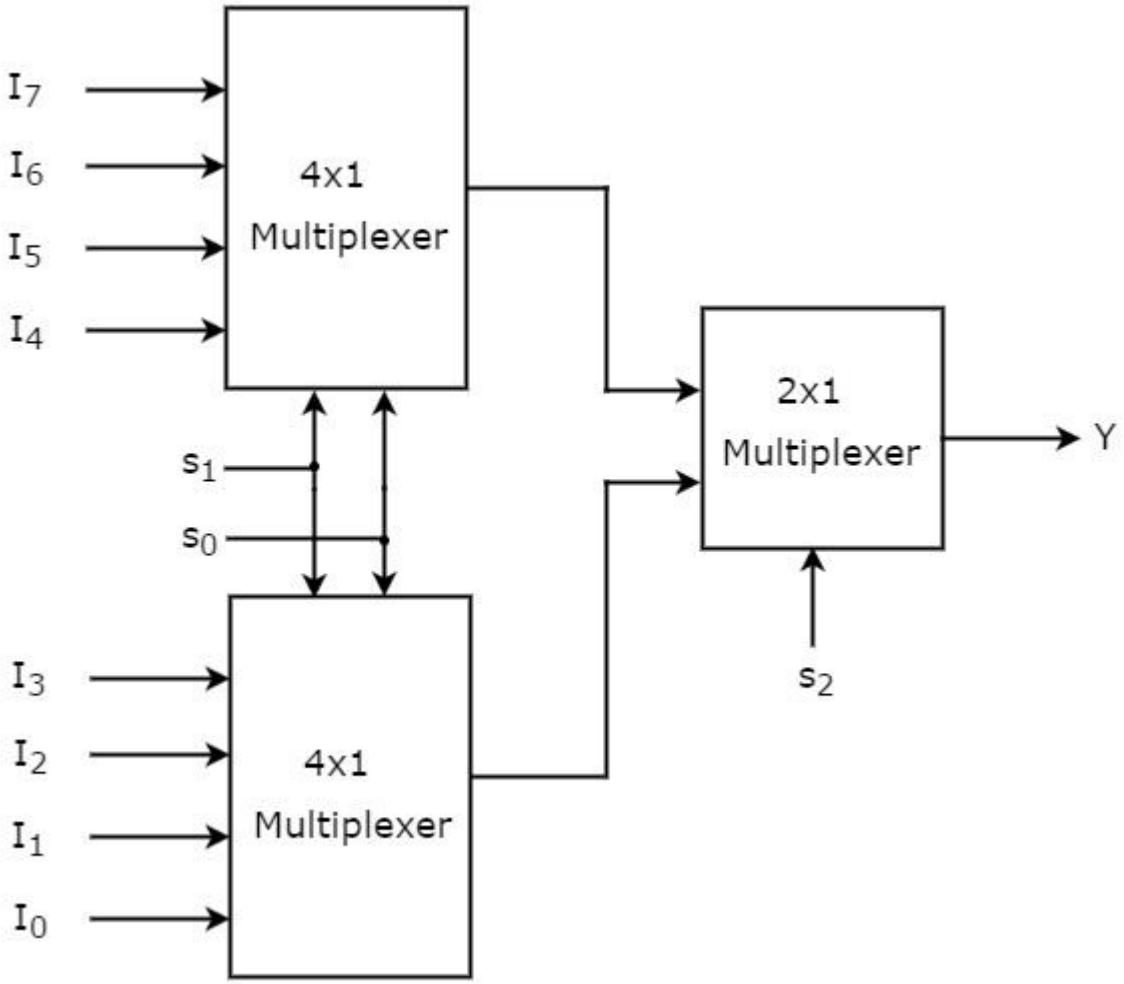
Multiplexers

$F(A, B, C, D) = \sum(0, 1, 3, 4, 8, 9, 15)$

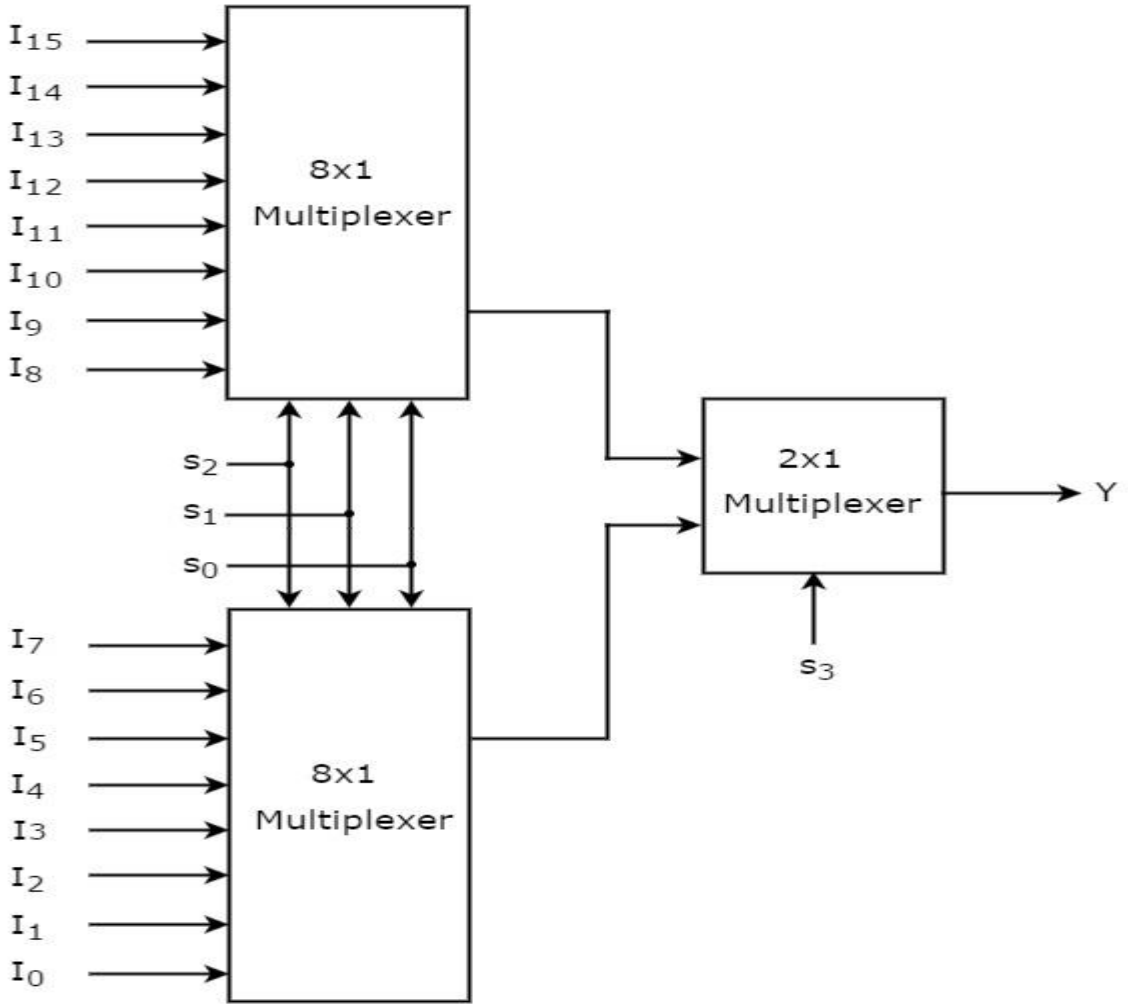
	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A'	(0)	(1)	2	(3)	(4)	5	6	7
A	(8)	(9)	10	11	12	13	14	(15)
	1	1	0	A'	A'	0	0	A



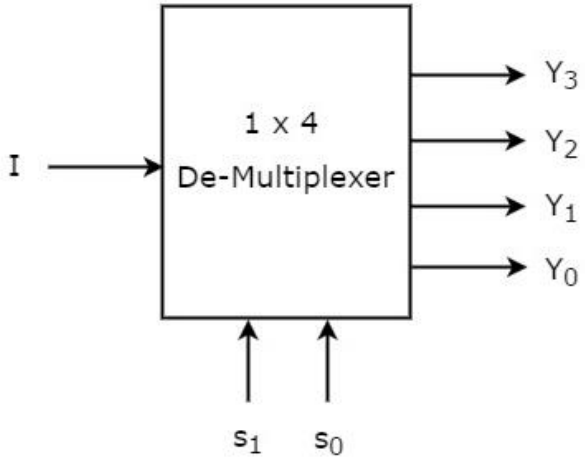
Multiplexers

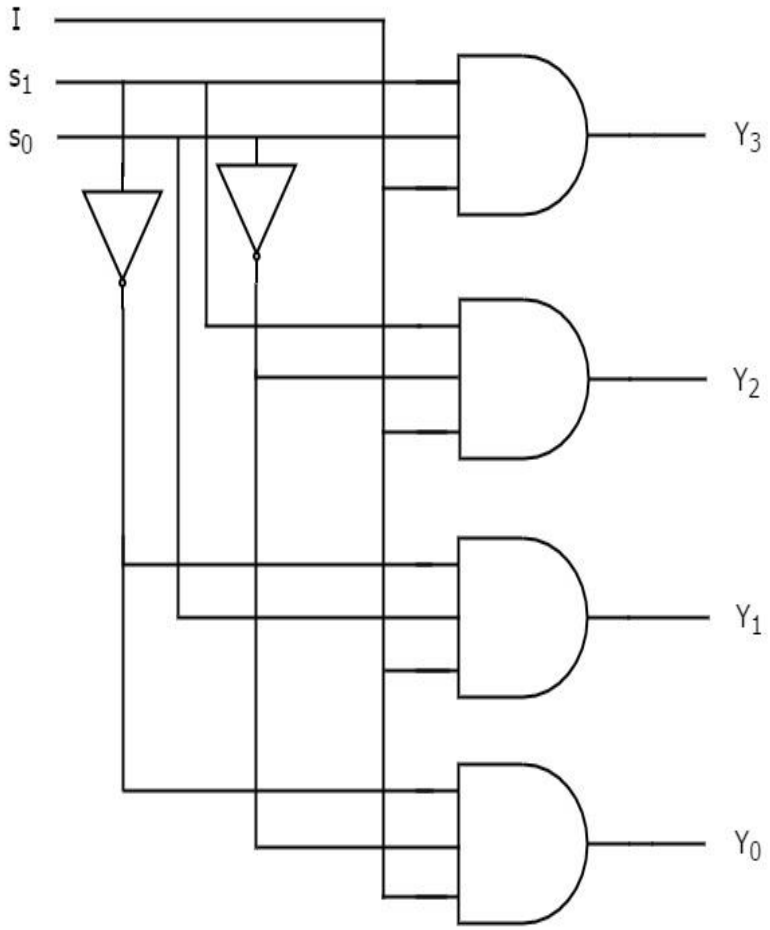


Multiplexers



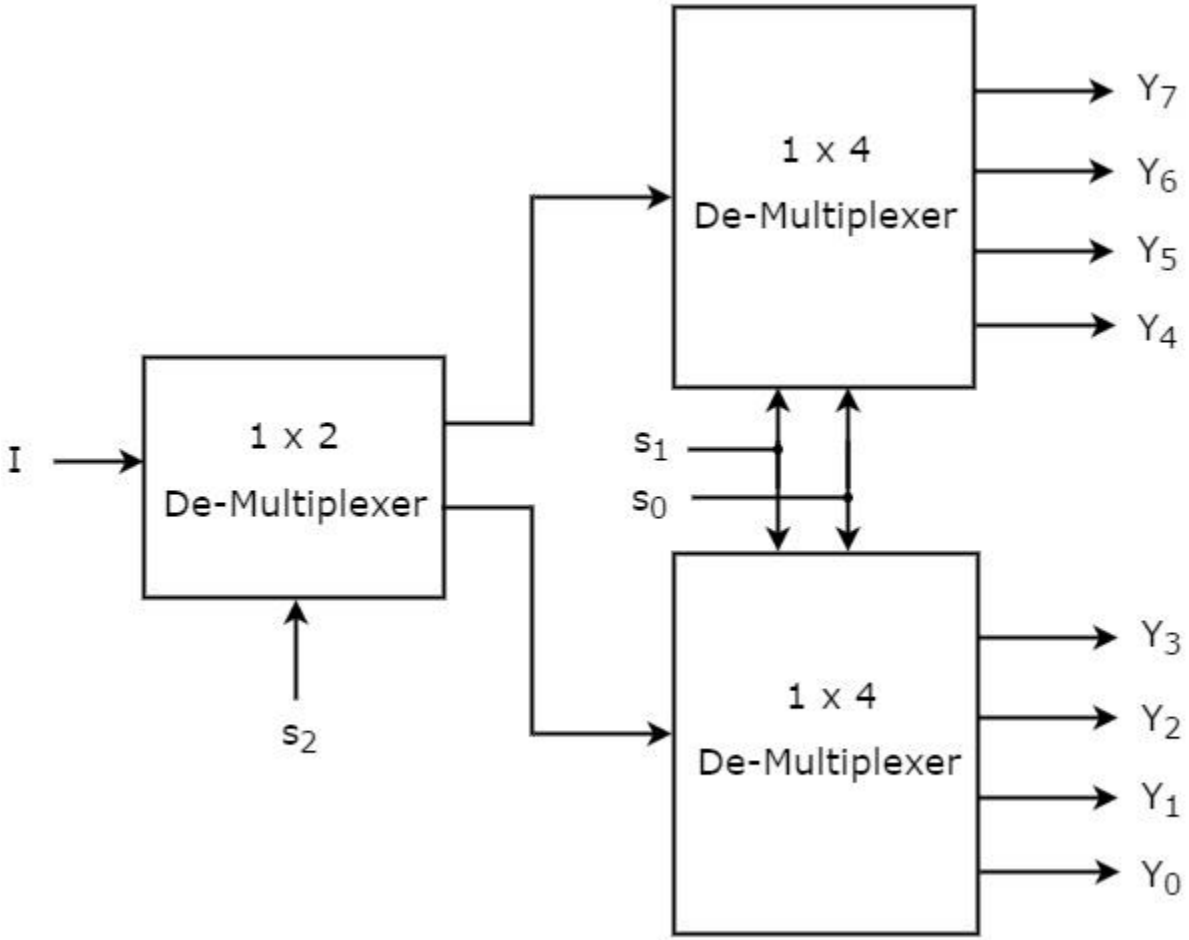
Demultiplexers



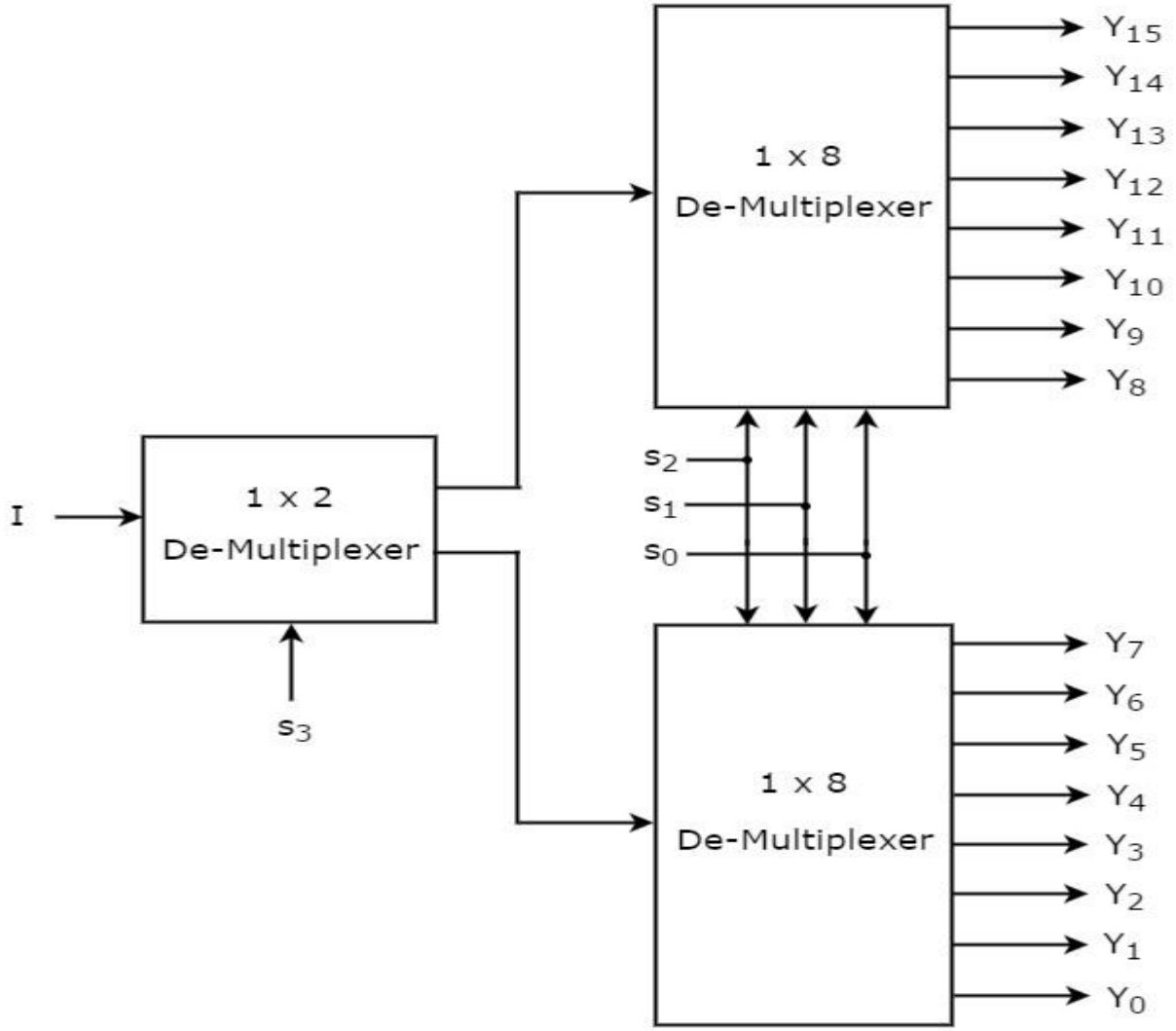


Selection Inputs		Outputs			
S_1	**S_0**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	0	0	0	0	**I**
0	1	0	0	**I**	0
1	0	0	**I**	0	0
1	1	**I**	0	0	0

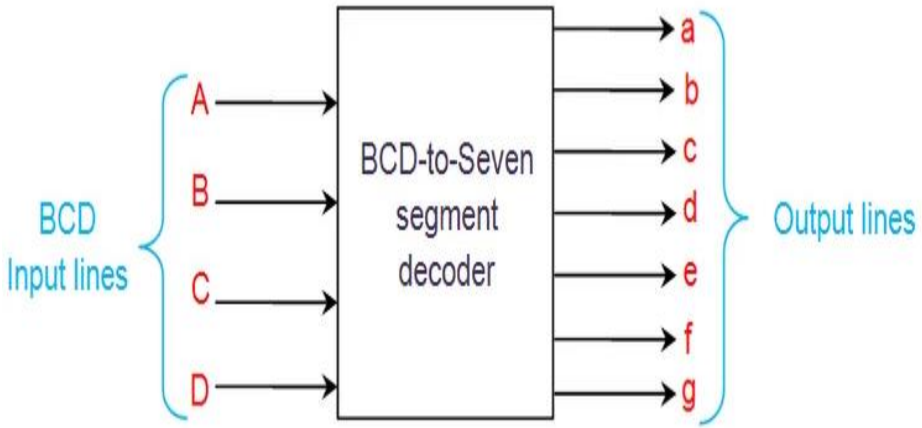
Demultiplexers

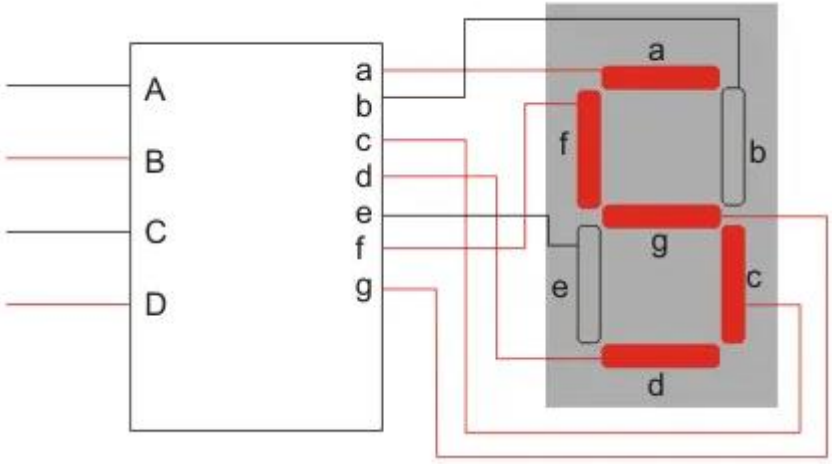


Demultiplexers



7-Segment Display

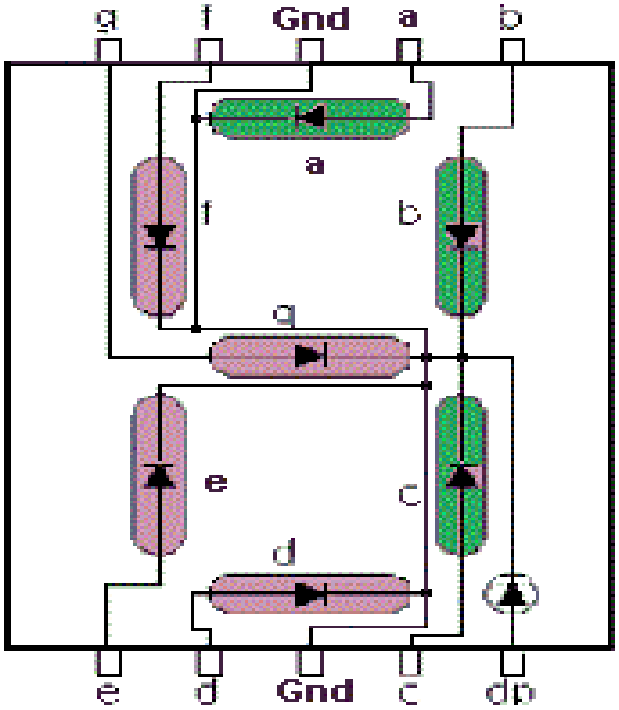




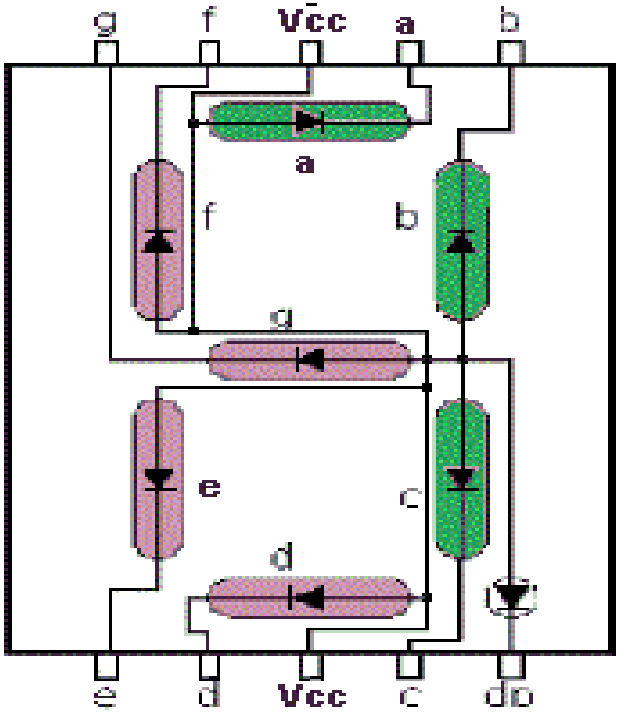
Decimal Digit	Input lines				Output lines							Display pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	
1	0	0	0	1	0	1	1	0	0	0	0	
2	0	0	1	0	1	1	0	1	1	0	1	
3	0	0	1	1	1	1	1	1	0	0	1	
4	0	1	0	0	0	1	1	0	0	1	1	
5	0	1	0	1	1	0	1	1	0	1	1	
6	0	1	1	0	1	0	1	1	1	1	1	
7	0	1	1	1	1	1	1	0	0	0	0	
8	1	0	0	0	1	1	1	1	1	1	1	
9	1	0	0	1	1	1	1	1	0	1	1	

7-Segment Display

Common cathode:
Note the common
Gnd with each led



Common Anode:
Note the common
Vcc with each led



Read Only Memory

- Only read operation can be done.
- Non-volatile memory.
- **Types:**

  a) Masked ROM

  b) Programmable Read Only Memory (PROM)

  c) Erasable and Programable Read Only Memory (EPROM)

  d) Electrically Erasable and Programable Read Only Memory (EEPROM)

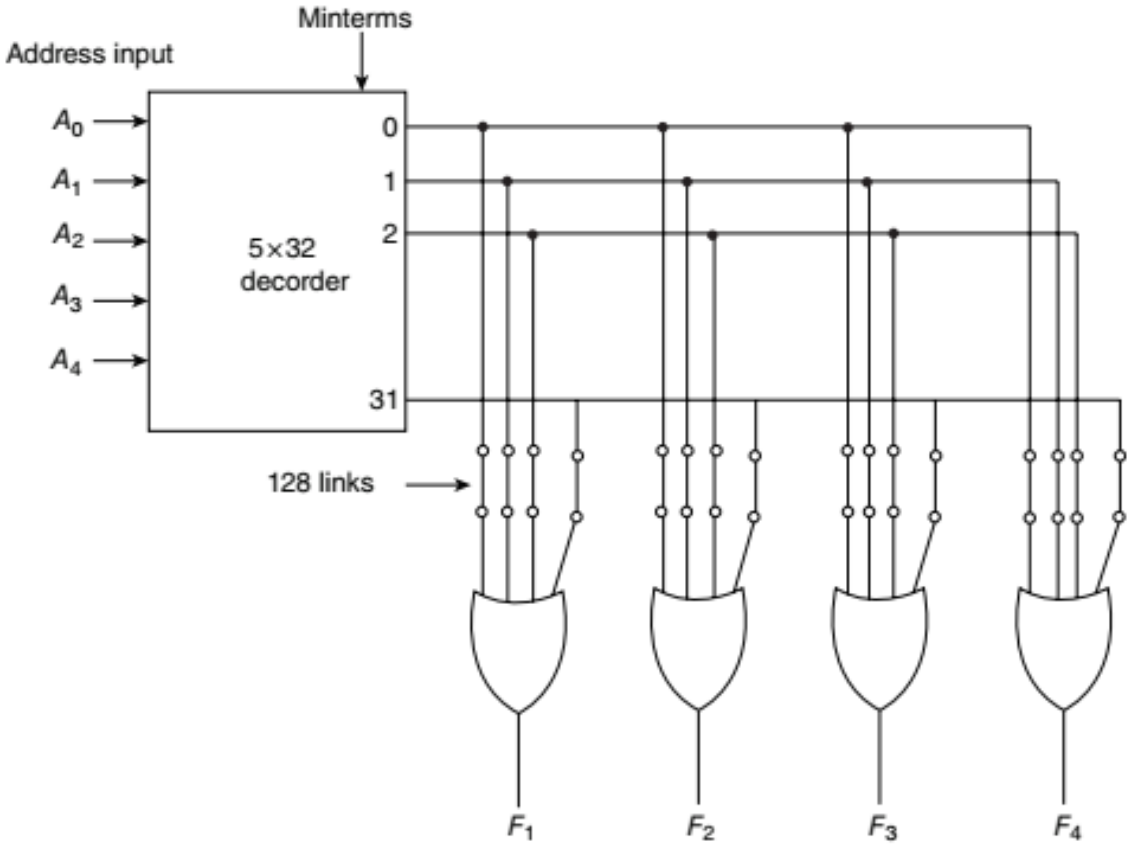
Read Only Memory

Internal Architecture:

- Implemented using Decoder and OR gates.
- Connection between the output of decoder and OR gate is defined using programming.
- It stores fixed set of binary information.
- Links are formed by fusing the wire or breaking the link.
- **n**: Input Lines, **m**: Output lines
- Each bit combinations of input variables are called as **address**.
- Each bit combination that comes out of the output lines is called a ***word***.
- Number of bits per word is equal to number of output lines

Read Only Memory

32 × 4 ROM

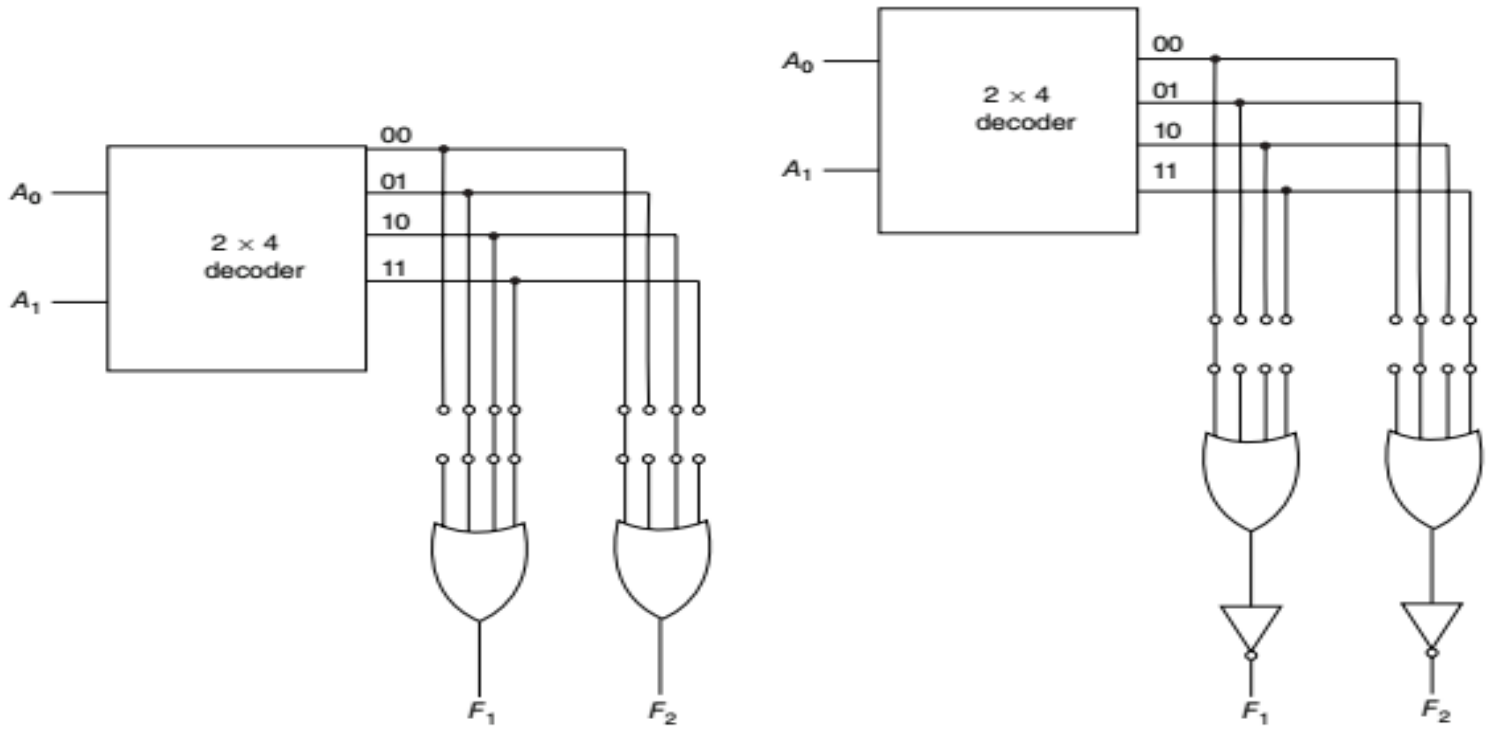


Read Only Memory

Combinational Logic Implementation

A_1	A_0	F_1	F_2
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

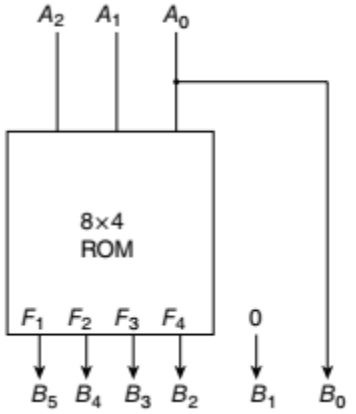
(a) Truth table



Read Only Memory

Design a combinational circuit using a ROM. The circuit accepts a 3-bit number and generates an output binary number equal to the square of the input number

Inputs			Outputs						Decimal
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49



A_2	A_1	A_0	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

Read Only Memory

a) Masked ROM

b) Programmable Read Only Memory (PROM)

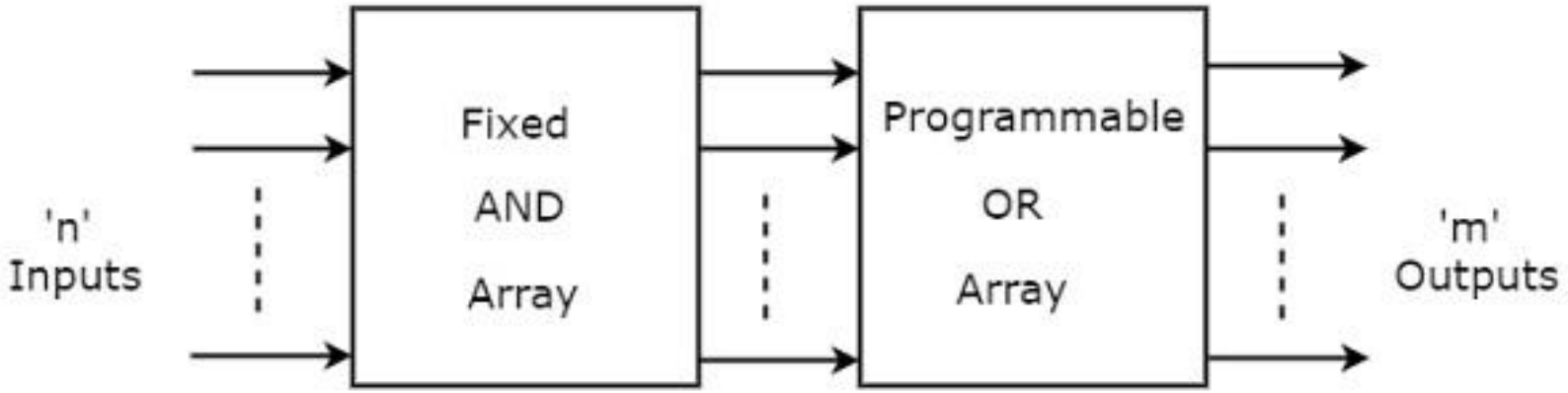
c) Erasable and Programable Read Only Memory (EPROM) :

UV Rays

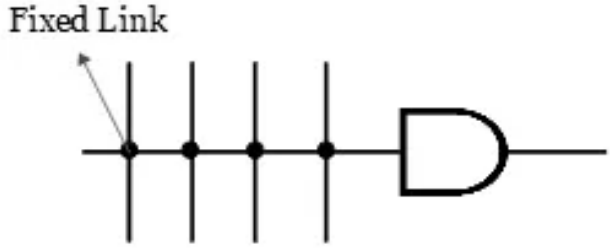
d) Electrically Erasable and Programable Read Only Memory (EEPROM):

High Electrical Voltage

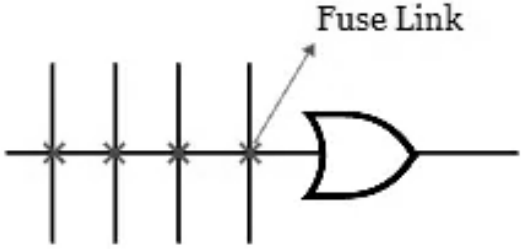
Programmable Read Only Memory



Conventional Gate



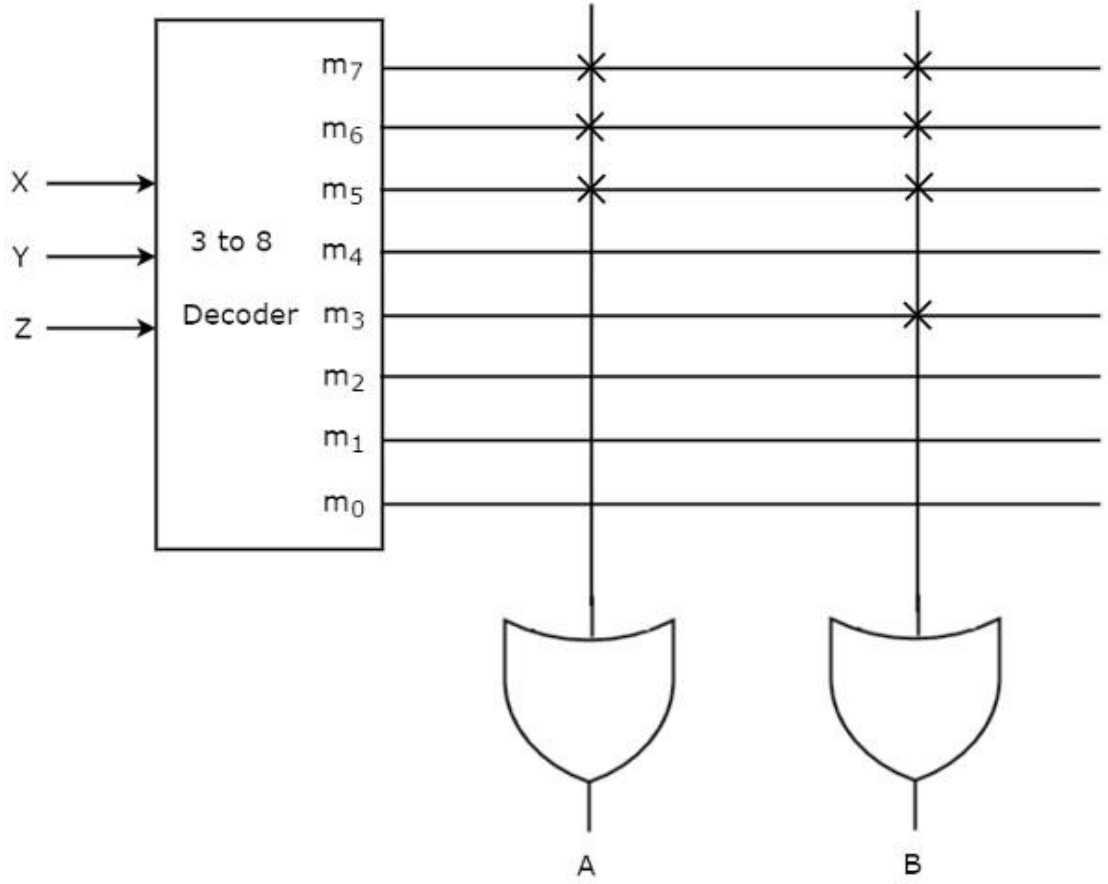
Fixed AND gate array



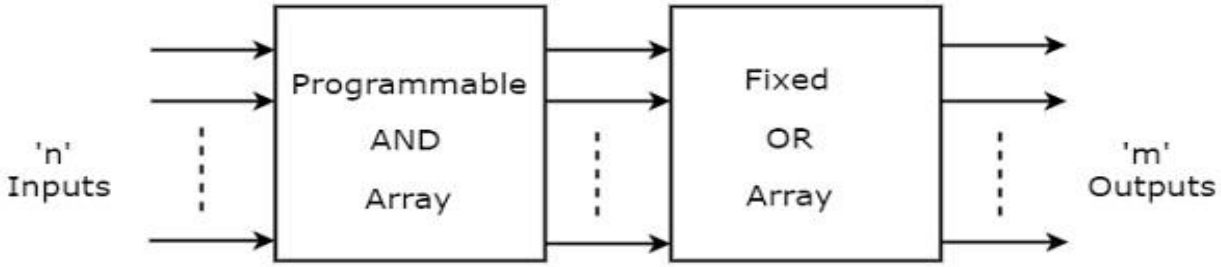
Programmable OR gate array

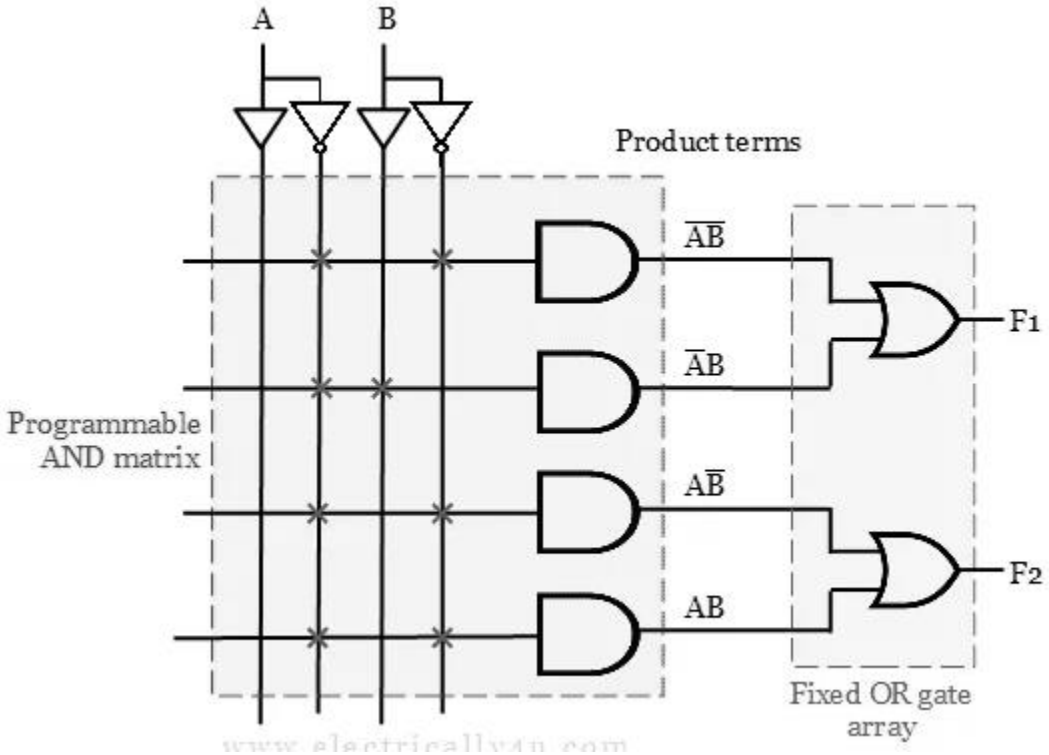
Programmable Read Only Memory

$A(X,Y,Z)=\sum m(5,6,7)A(X,Y,Z)=\sum m(5,6,7)$
$B(X,Y,Z)=\sum m(3,5,6,7)$



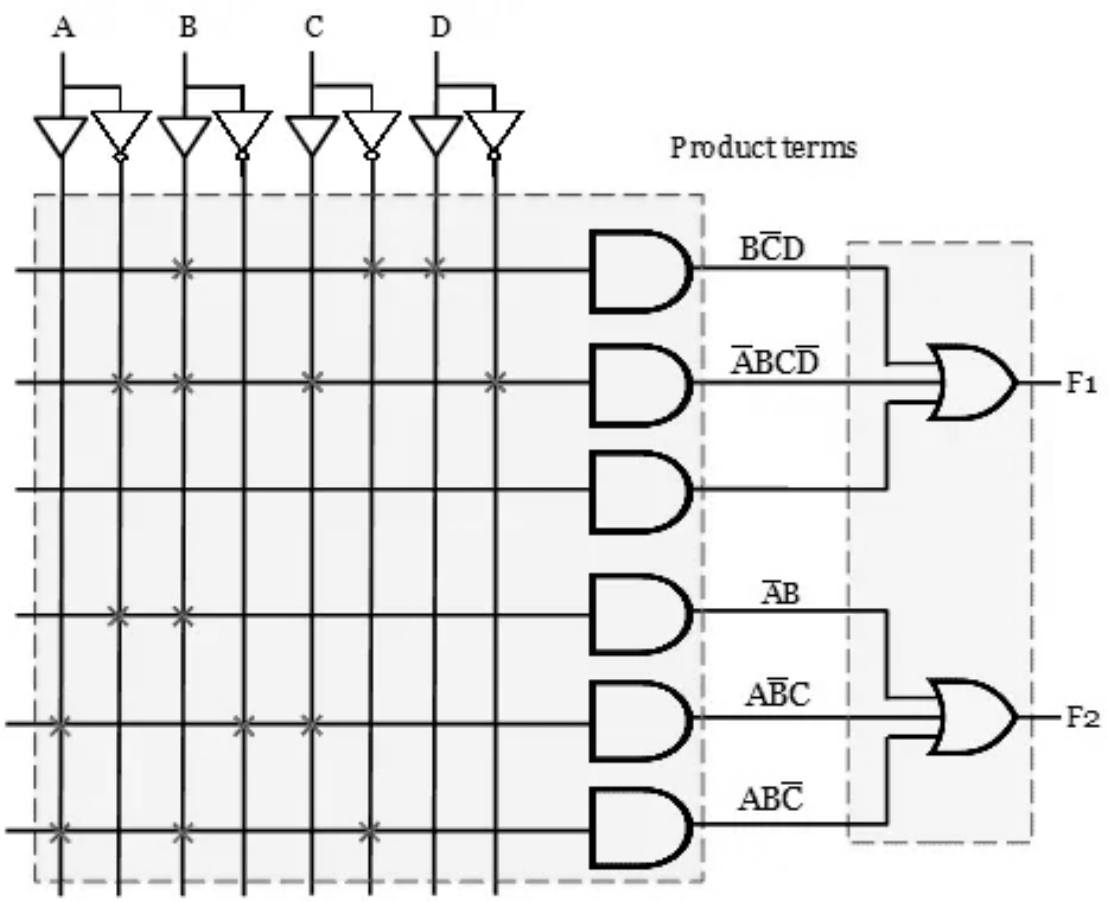
Programmable Array Logic (PAL)





Programmable Array Logic (PAL)

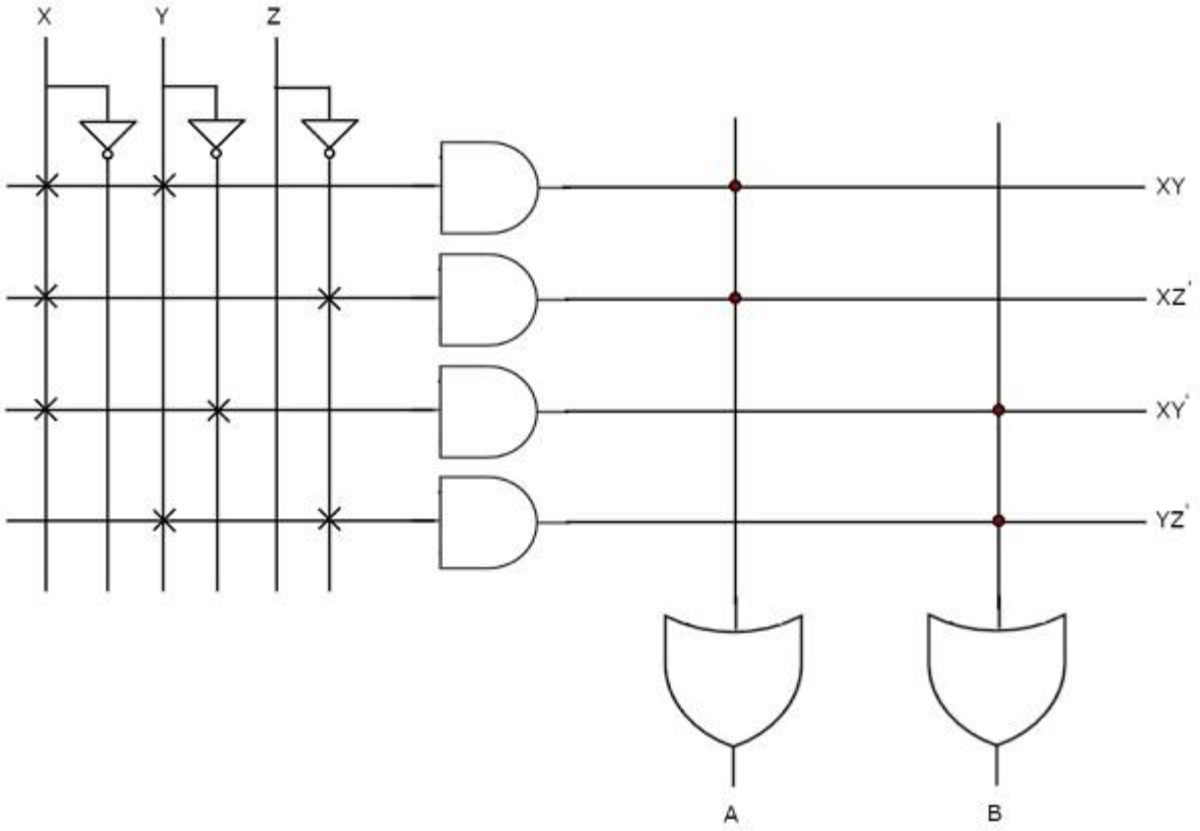
Implement the Boolean functions F1 = A'BC'D + A'BCD' + ABC'D and F2 = A'BC' + A'BC + AB'C + ABC' with PAL device.



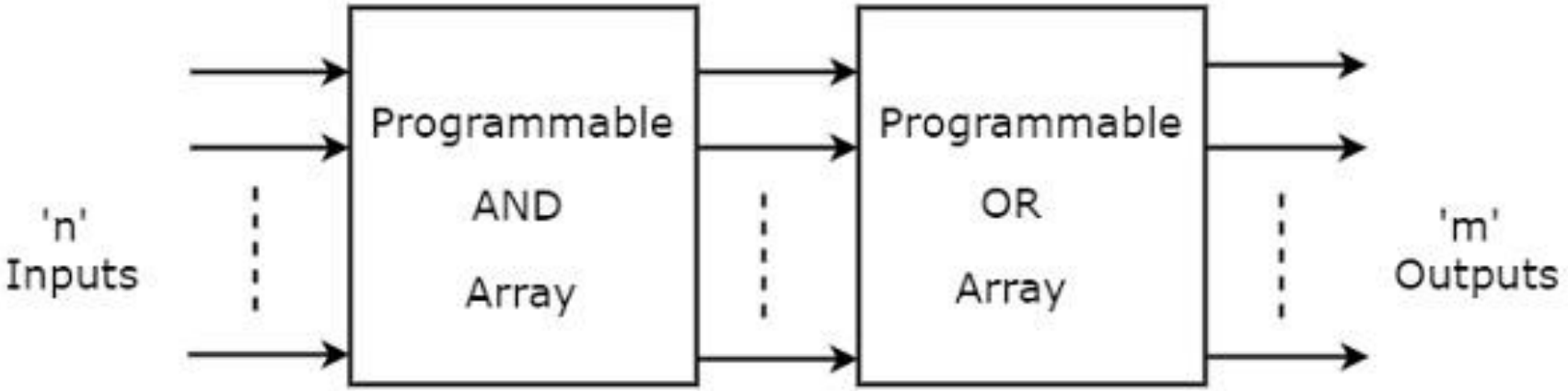
Programmable Array Logic (PAL)

′A=XY+XZ′

B=XY’+YZ’′



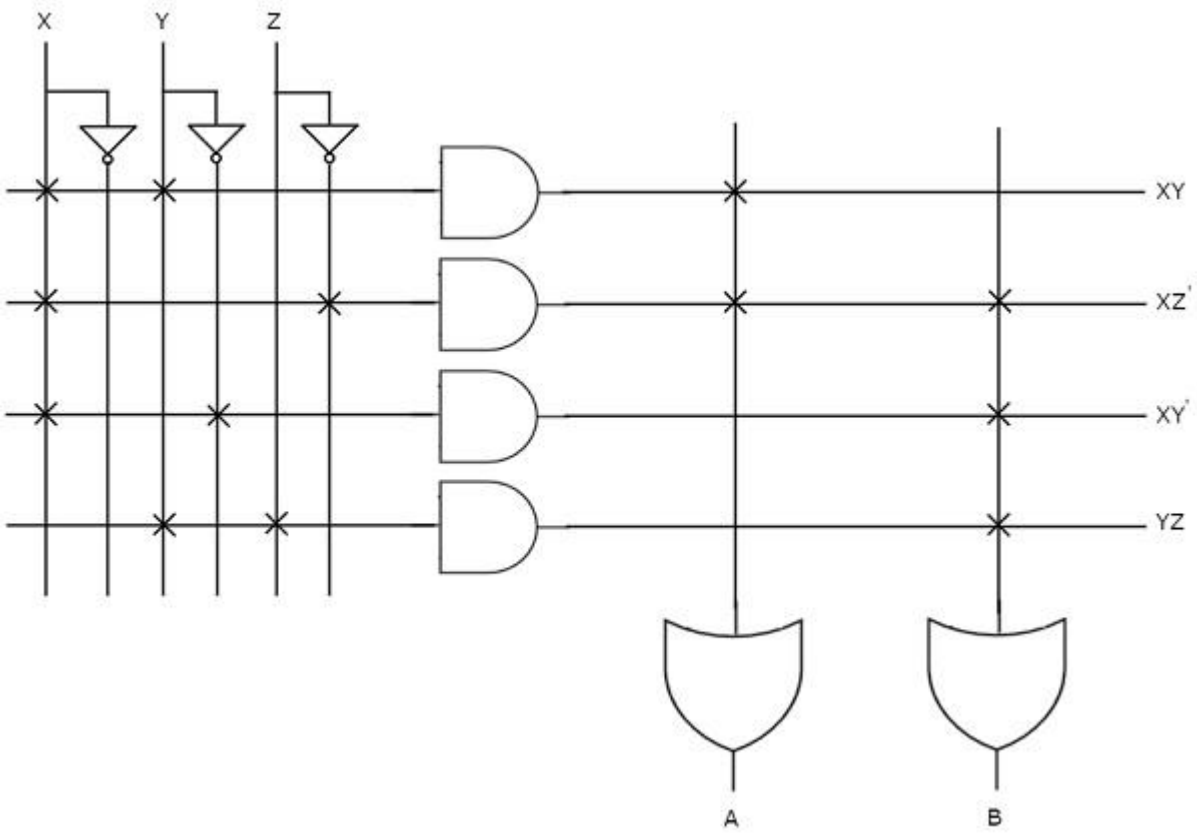
Programmable Logic Array (PLA)



Programmable Logic Array (PLA)

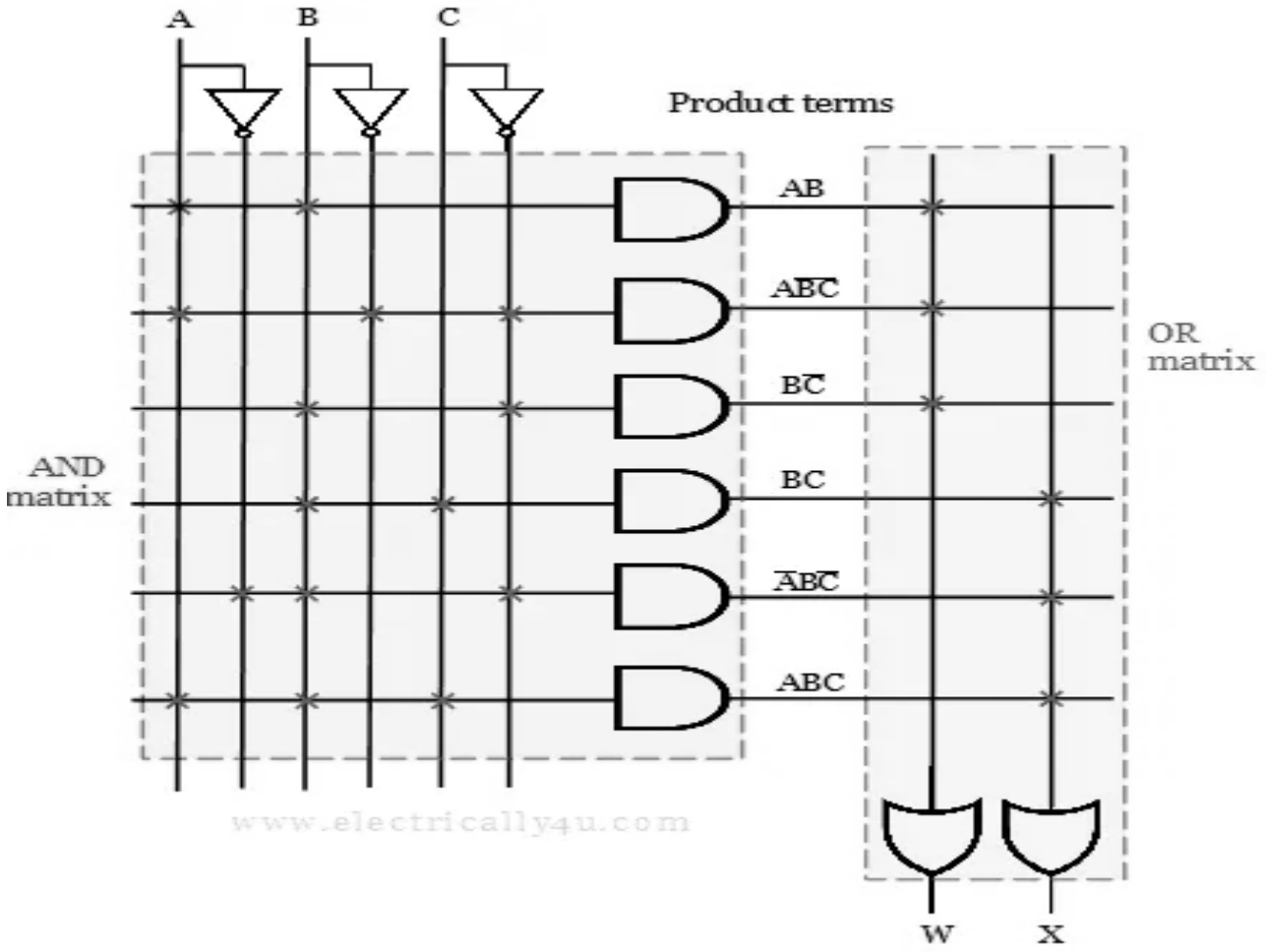
A=XY+XZ′A=XY+XZ′

B=XY′+YZ+XZ′



Programmable Logic Array (PLA)

How to realize the Boolean expression X = AB + AB'C' + BC' and Y = BC + A'BC' + ABC using Programmable Logic Array.



Programmable Logic Array (PLA)

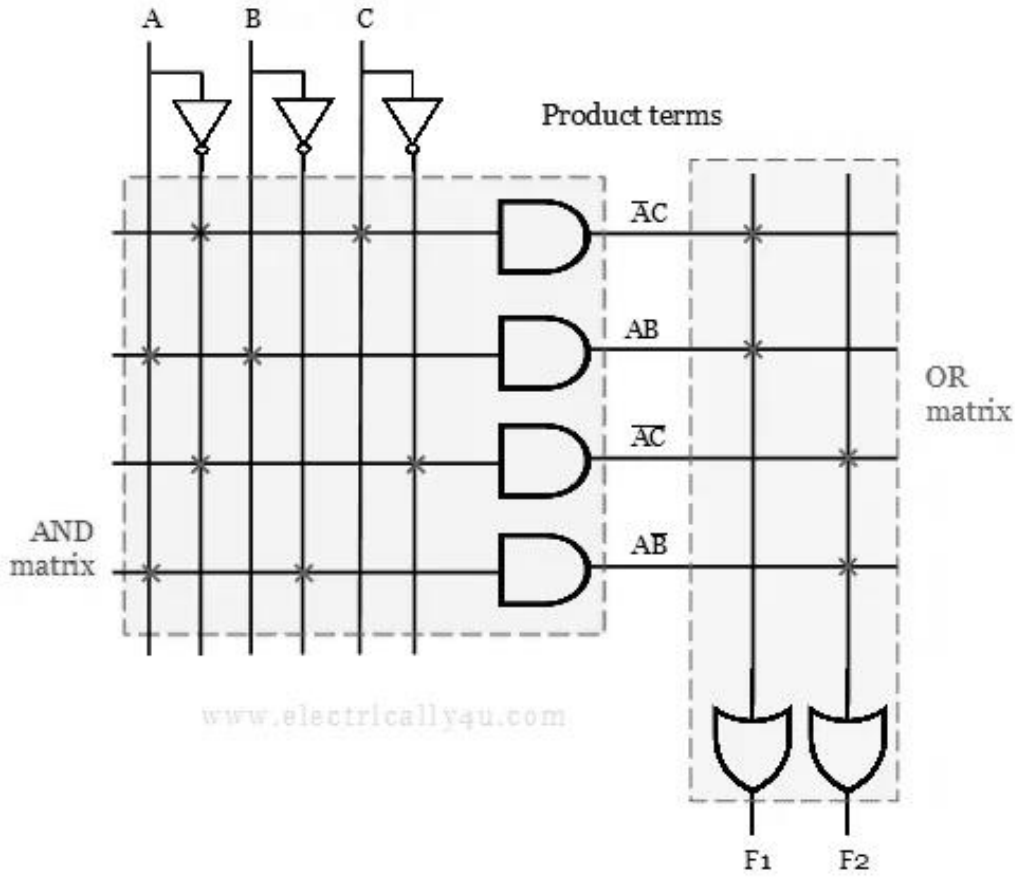
Realize a Boolean functions F1(A, B, C) = ∑ m(1, 3, 6, 7) and F2(A, B, C) = ∑ m(0, 2, 4, 5) using PLA.

A \ BC	00	01	11	10
0	0	1	1	0
1	0	0	1	1

$F_1 = \bar{A}C + A\bar{B}$

A \ BC	00	01	11	10
0	1	0	0	1
1	1	1	0	0

$F_2 = \bar{A}\bar{C} + A\bar{B}$



Random Access Memory (RAM)

RAM: the stored data is volatile

- ***DRAM***
  - » A capacitor to store data, and a transistor to access the capacitor
  - » **Need refresh operation**
  - » **Low cost**, and **high density** ⇒ it is used for main memory
- ***SRAM***
  - » Consists of a latch
  - » **Don't need the refresh operation**
  - » **High speed** and **low power consumption** ⇒it is mainly used for cache memory and memory in hand-held devices

Random Access Memory (RAM)

DRAM

- Capacitor to store the data
- Slower
- Cheap
- Main Memory
- Highly dense
- Simple
- One transistor

SRAM

- Transistors are used to store the data
- Faster
- Expensive
- Cache Memory
- Less Dense
- Complex
- 6 Transistors