TSN1101
Computer Architecture and Organization

Section A (Digital Logic Design)

Lecture 04

[redacted]

Standard Combinational Logic Circuit

TOPIC COVERAGE IN THE LECTURE - Part 1 of 3

- **Design of Half Adder**
  - Using AND-OR gates
  - Using XOR-AND gate
  - Using only NAND/ only NOR gates
- **Design of Full Adder**
  - Using AND-OR gates
  - Using two half adders
  - Using only NAND/ only NOR gates
- **Construction of 4-bit Parallel Binary Adder**
- **Construction of 4-bit Parallel Binary Adder/Subtractor**
- **BCD Addition – Procedure and Example**

TOPIC COVERAGE IN THE LECTURE - Part 2 of 3

- **Decoders**
  - 1-to-2-Line Decoder
  - 2-to-4-Line Decoder
  - 3-to-8-Line Decoder
  - Construction of higher order decoder using lower order decoders – An example
  - Implementation of combinational logic circuit using decoder
- **Encoders**
  - 8-to-3-Line Encoder
  - Design of 4-to-2-Line Priority Encoder
- **Multiplexers**
  - 2-to-1-Line MUX
  - 4-to-1-Line MUX
  - Implementation of Boolean Function using MUX
- **Demultiplexers**
  - 1-to-4-Line DMUX

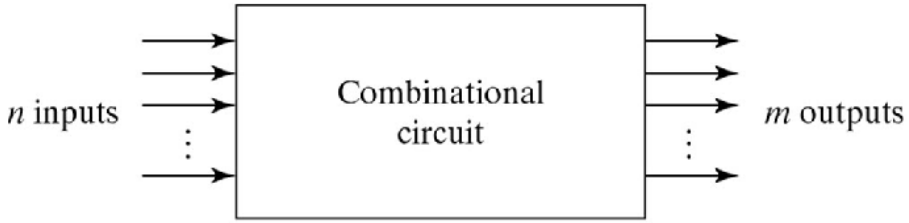
TOPIC COVERAGE IN THE LECTURE - Part 3 of 3

- **Comparators**
  - 2-bit magnitude comparator
  - 4-bit magnitude comparator
  - 8-bit magnitude comparator using two 4-bit comparators
- **Code Converters**
  - Binary to Gray code converter
  - Gray code to Binary converter
  - BCD to XS3 code converter
- **Parity Generators/Checkers**
  - Construction of Parity Generator and Parity Checker circuits

Combinational Logic Circuits

- A combinational logic circuit consists of logic gates whose outputs at any time are determined from the present combination of inputs.

Block Diagram of a Combinational Logic Circuit



- It consists of input variables, logic gates, and output variables.
- The logic gates accept signals (binary) from the n inputs and generate signals (binary) to m outputs.
- It can be specified with a truth table that lists the output values (unique) for each combination of input variables.
- It can also be described by m Boolean functions (expressed in terms of n input variables), one for each output variable.

Standard Combinational Logic Circuits – Part 1 of 3 (Arithmetic Circuits)

- Half Adder/Full Adder
- Parallel binary adder/subtractor
- BCD addition

Design of Half-Adder using

* AND-OR Gates
* XOR-AND Gates
* Only NAND Gates
* Only NOR Gates

Half-Adder

- This circuit needs 2 binary inputs and 2 binary outputs.
- The input variables are designated as augend (X) and addend (Y).
- The output variables are designated as the sum (S) and carry (C).

Truth Table:

INPUTS		OUTPUTS	
Augend (x)	Addend (y)	Carry (C)	Sum (S)
0	0	0	0
0	1	0	1 √
1	0	0	1 √
1	1	1 √	0

Boolean expressions:

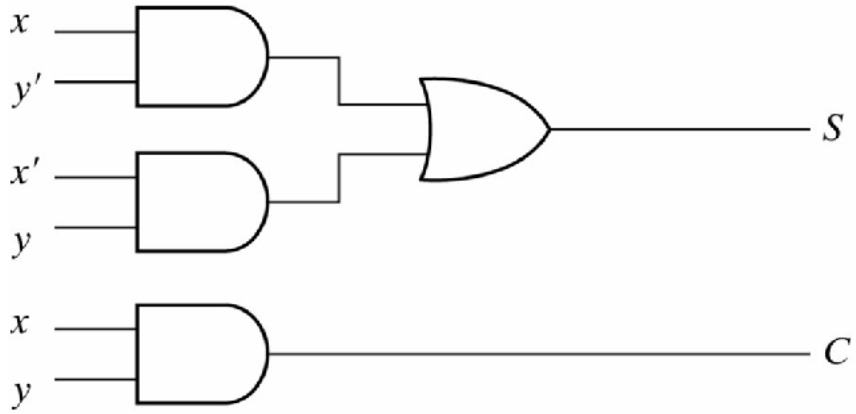
(a) $S = xy' + x'y$
$C = xy$

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

Half Adder
- Two different implementations

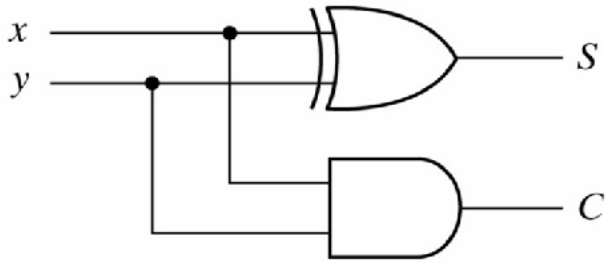
Logic Diagrams:

Implementation of Half Adder using AND and OR Gates



(a) $S = xy' + x'y$
$C = xy$

Implementation of Half Adder using XOR and AND Gates



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

Half Adder

- Implementation using **only NAND** gates

BOOLEAN EXPRESSIONS :

$$S = \bar{x}y + x\bar{y}$$

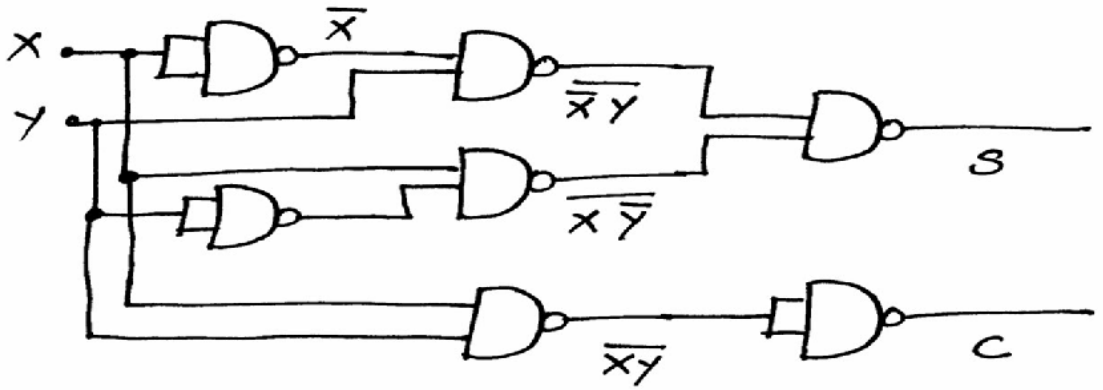
$$= \overline{\overline{\bar{x}y + x\bar{y}}}$$

$$S = \overline{\overline{\bar{x}y} \cdot \overline{x\bar{y}}}$$

$$C = xy$$

$$C = \overline{\overline{xy}}$$

LOGIC DIAGRAM :



Half Adder

- Implementation using **only NOR** gates

BOOLEAN EXPRESSIONS :

$$S = (X+Y)\cdot(\overline{X}+\overline{Y})$$

$$= \overline{\overline{(X+Y)\cdot(\overline{X}+\overline{Y})}}$$

$$S = \overline{(X+Y)} + \overline{(\overline{X}+\overline{Y})}$$

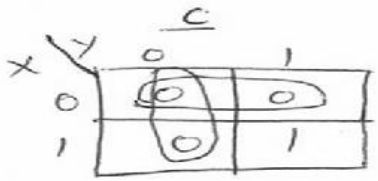
$$C = X \cdot Y$$

$$= \overline{\overline{X \cdot Y}}$$

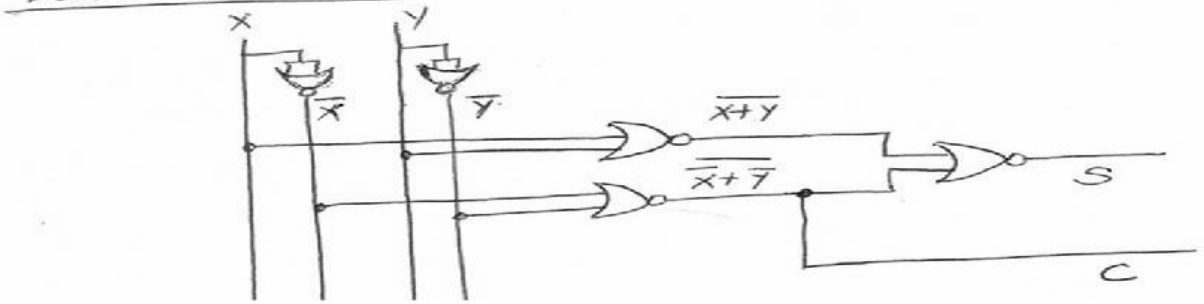
$$C = \overline{\overline{X}+\overline{Y}}$$

TRUTH TABLE

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



LOGIC DIAGRAM :



Design of Full-Adder using

* AND-OR Gates
* Two half adders
* Only NAND Gates
* Only NOR Gates

Full-Adder

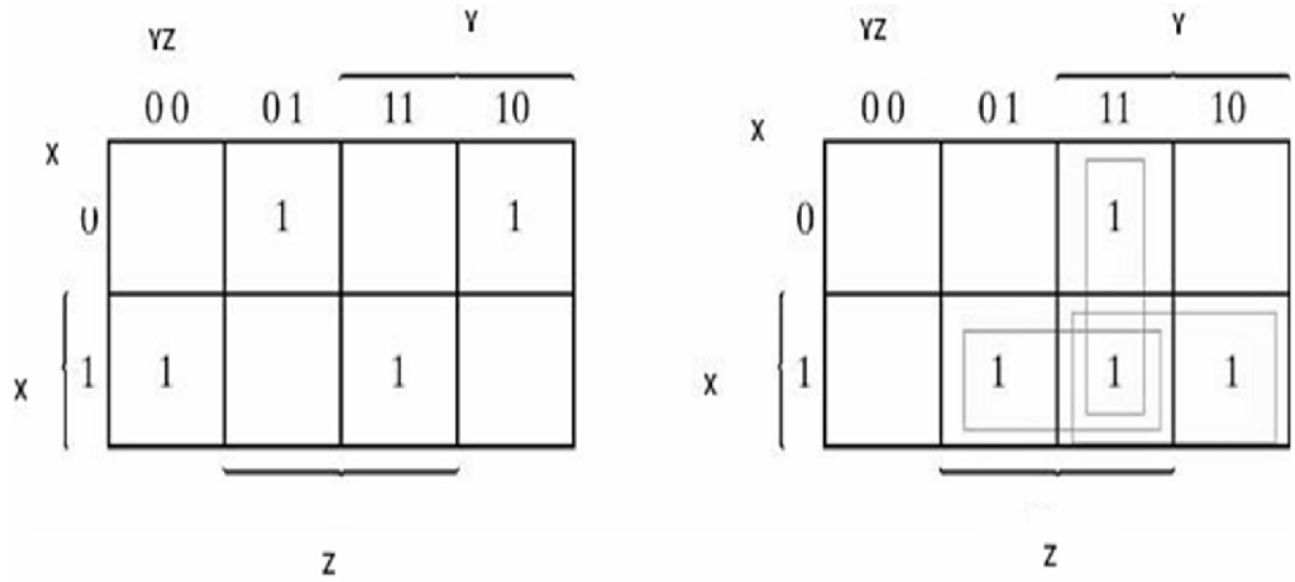
- Full Adder is a combinational circuit that forms the arithmetic sum of 3 bits.
- Consists of 3 inputs and 2 outputs.
- The outputs S and C are 0
  - when all input bits are 0
- The output S is equal to 1
  - when only one input is equal to 1 or
  - when all 3 inputs are equal to 1.
    - The output S is Odd Function – can be implemented with 3 input XOR gate
- The output C has a carry of 1
  - if 2 or 3 inputs are equal to 1.

Full Adder
- Truth Table

INPUTS			OUTPUTS	
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

Full Adder

- Simplification of Boolean expressions



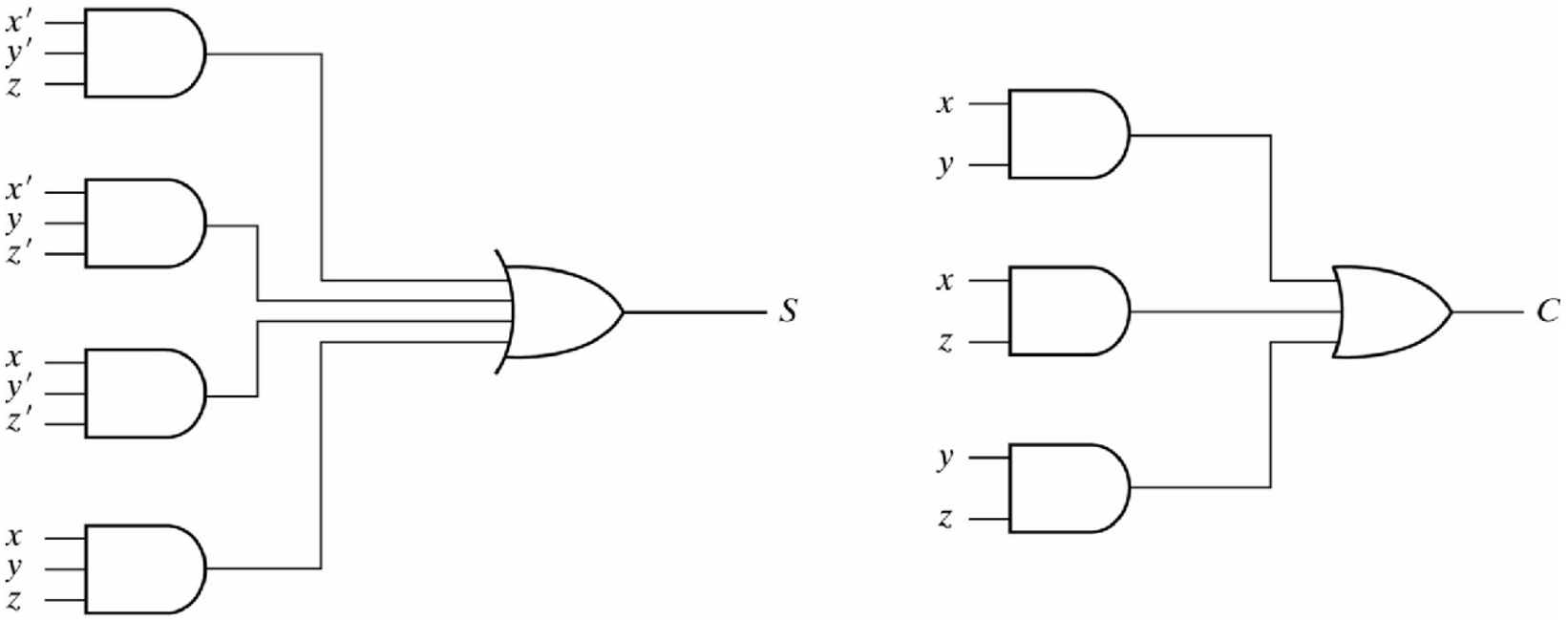
Inputs			Outputs	
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	**1**
0	1	0	0	**1**
0	1	1	**1**	0
1	0	0	0	**1**
1	0	1	**1**	0
1	1	0	**1**	0
1	1	1	**1**	**1**

S=x′y′z +x′yz′ + xy′z′ + xyz

C=xy+xz+yz

Full Adder

- Implementation using **AND-OR** Gate Network



Full Adder

- Implementation using **two Half Adders** …

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

$= z'(xy' + x'y) + z(xy + x'y')$

$= z'(xy' + x'y) + z(xy' + x'y)'$

$= z'(x \oplus y) + z(x \oplus y)'$

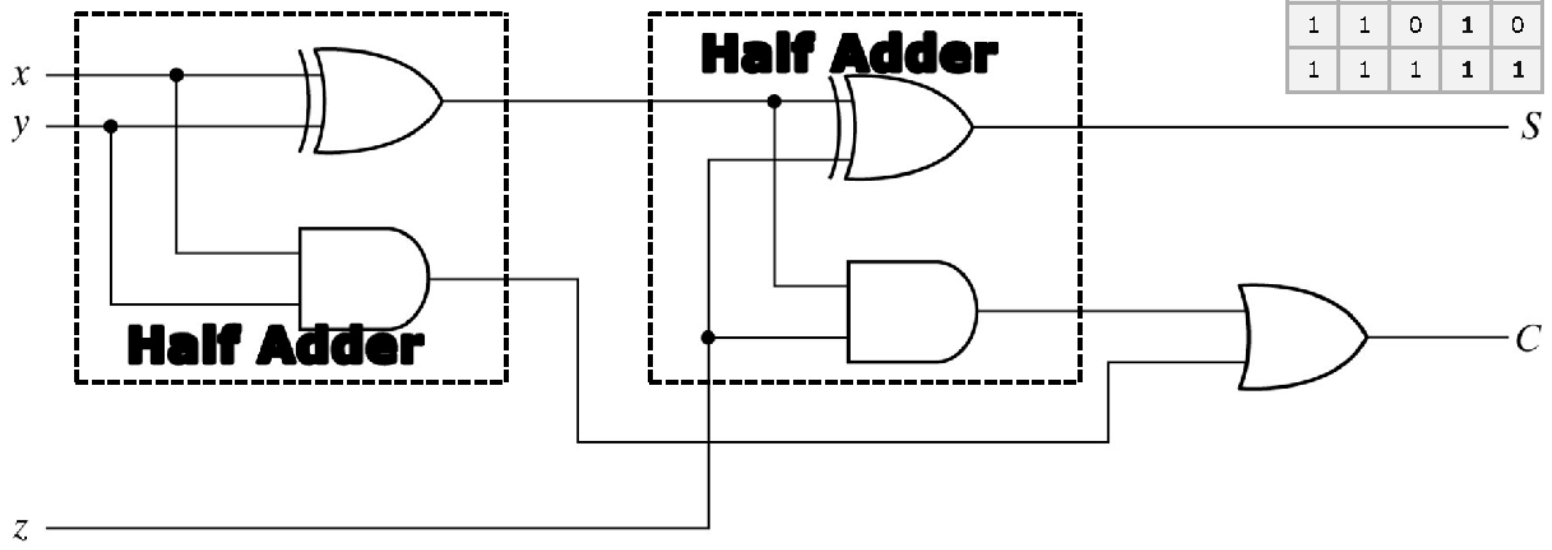
$S = z \oplus (x \oplus y)$

$C = xy'z + x'yz + xy$

$= z(xy' + x'y) + xy$

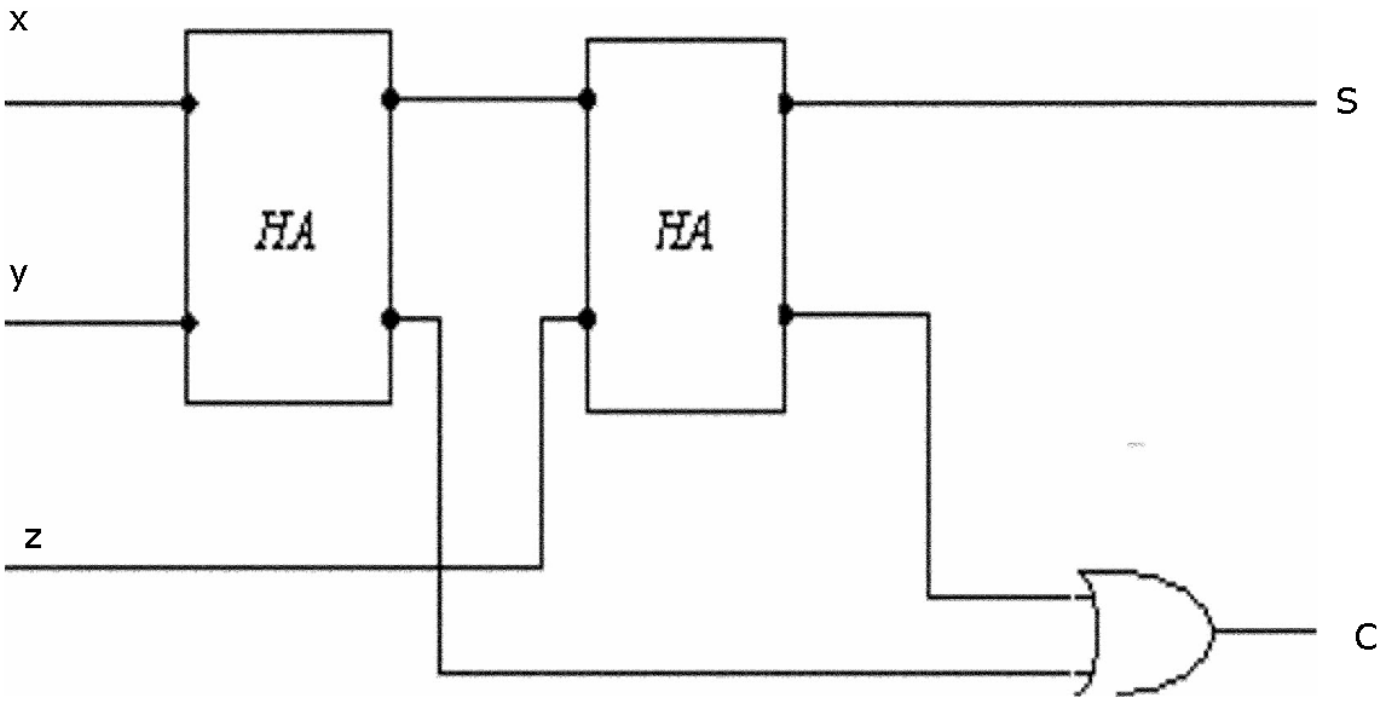
$C = z(x \oplus y) + xy$

Inputs			Outputs	
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**



Full Adder

- Implementation using **two Half Adders**



Full Adder

- Implementation using **only NAND** gates

Boolean expressions in NAND form

$$S = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$$

$$= \overline{\bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz}$$

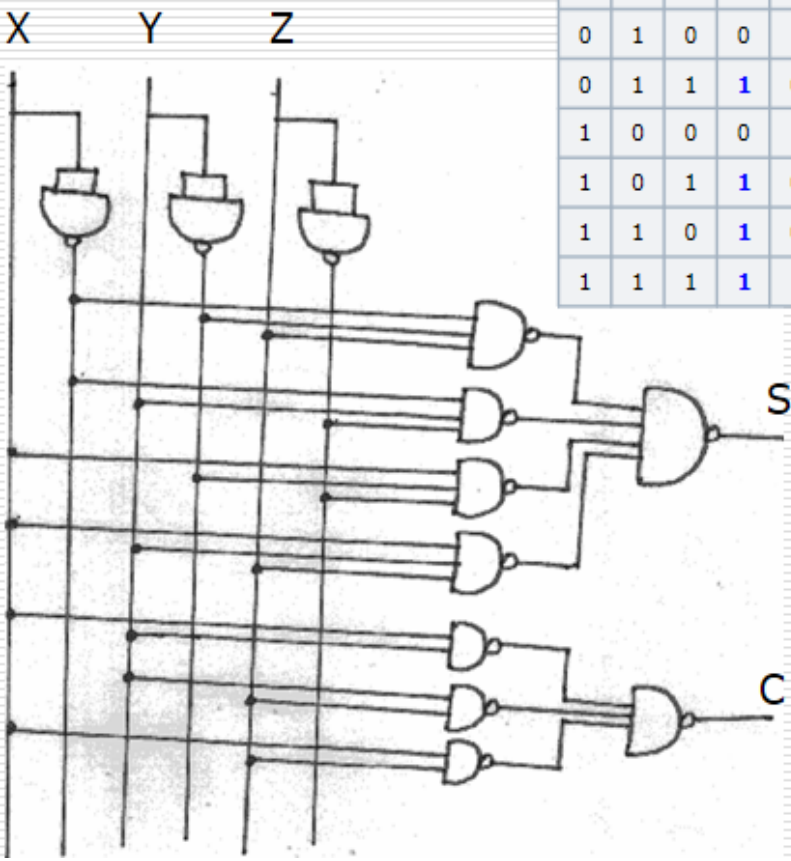
$$S = \overline{(\bar{x}\bar{y}z)} \cdot \overline{(\bar{x}y\bar{z})} \cdot \overline{(x\bar{y}\bar{z})} \cdot \overline{(xyz)}$$

$$C = xy + yz + xz$$

$$= \overline{\overline{xy + yz + xz}}$$

$$C = \overline{(\overline{xy})} \cdot \overline{(\overline{yz})} \cdot \overline{(\overline{xz})}$$

Logic Diagram using only NAND gates



Inputs			Outputs	
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

Full Adder

- Implementation using **only NOR** gates

Inputs			Outputs	
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

Boolean expressions in NOR form

$$S = (x+y+z)\cdot(x+\bar{y}+\bar{z})\cdot(\bar{x}+y+\bar{z})\cdot(\bar{x}+\bar{y}+z)$$

$$= \overline{(x+y+z)\cdot(x+\bar{y}+\bar{z})\cdot(\bar{x}+y+\bar{z})\cdot(\bar{x}+\bar{y}+z)}$$

$$S = \overline{(x+y+z)} + \overline{(x+\bar{y}+\bar{z})} + \overline{(\bar{x}+y+\bar{z})} + \overline{(\bar{x}+\bar{y}+z)}$$

CARRY

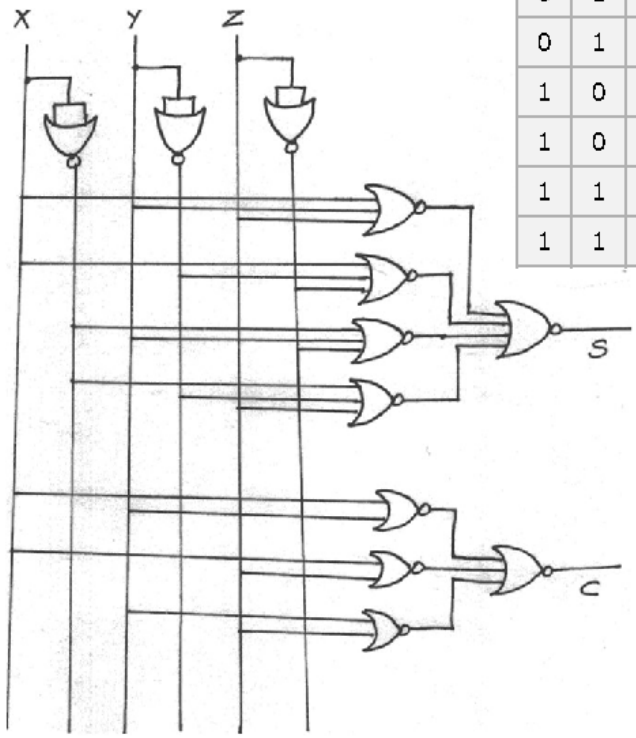
x \ yz	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$C = (x+y)\cdot(x+z)\cdot(y+z)$$

$$= \overline{(x+y)\cdot(x+z)\cdot(y+z)}$$

$$C = \overline{(x+y)} + \overline{(x+z)} + \overline{(y+z)}$$

Logic Diagram using only NOR gates



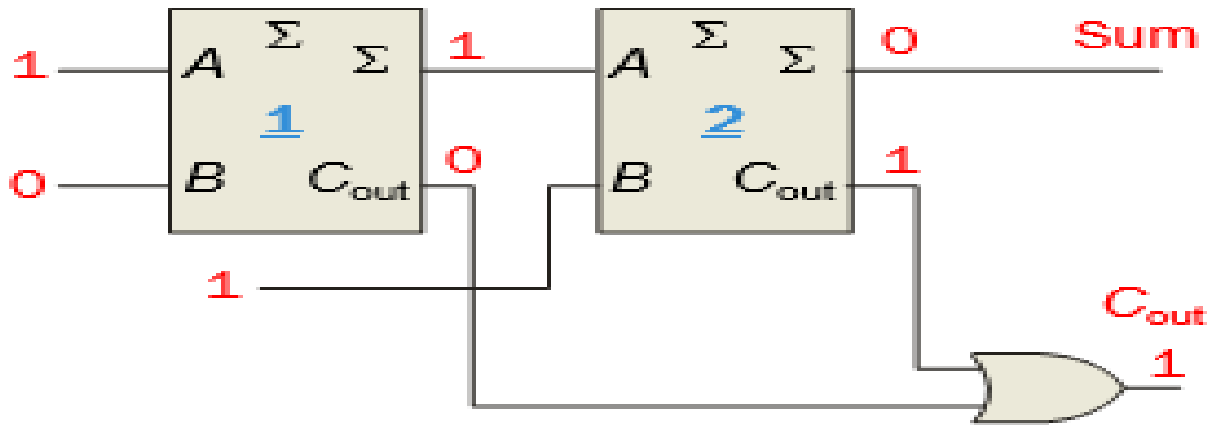
Construction of

* 4-bit Parallel Binary Adder
* 4-bit Parallel Binary Adder/Subtractor

Parallel Binary Adder

- Parallel Adder is a digital circuit that produces the arithmetic sum of 2 binary numbers.
- Constructed with full adders connected in cascade, with output carry from each full adder connected to the input carry of next full adder in the chain
- The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 1 denoting the least significant bit
- The carries are connected in a chain through the full adders.

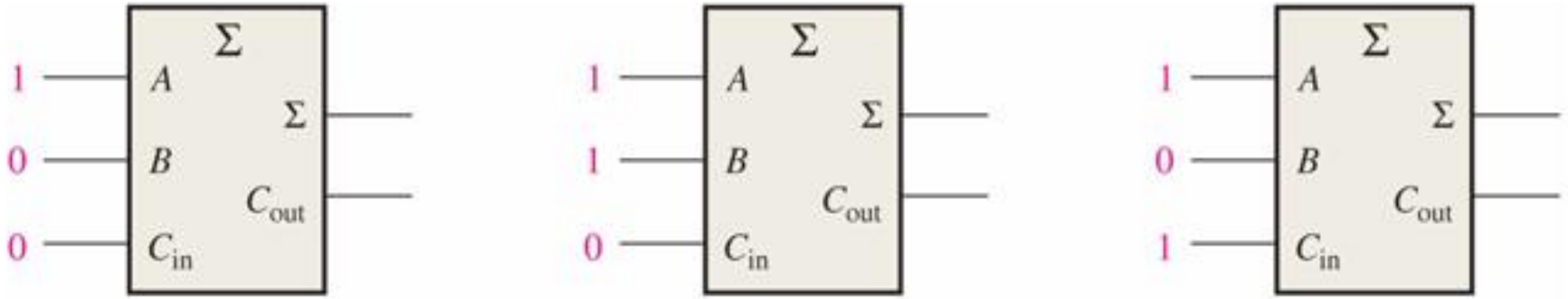
How the FA works with 1+1



- The first half-adder has inputs of 1 and 0; therefore the Sum =1 and the Carry out = 0
- The second half-adder has inputs of 1 and 1; therefore the Sum = 0 and the Carry out = 1
- The OR gate has inputs of 1 and 0, therefore the final carry out = 1

Exercise

- For each of the full-adders below, determine the outputs for the inputs shown



(a) The input bits are $A = 1$, $B = 0$, and $C_{in} = 0$.

$1 + 0 + 0 = 1$ with no carry

Therefore, $\Sigma = 1$ and $C_{out} = 0$.

(b) The input bits are $A = 1$, $B = 1$, and $C_{in} = 0$.

$1 + 1 + 0 = 0$ with a carry of 1

Therefore, $\Sigma = 0$ and $C_{out} = 1$.

(c) The input bits are $A = 1$, $B = 0$, and $C_{in} = 1$.

$1 + 0 + 1 = 0$ with a carry of 1

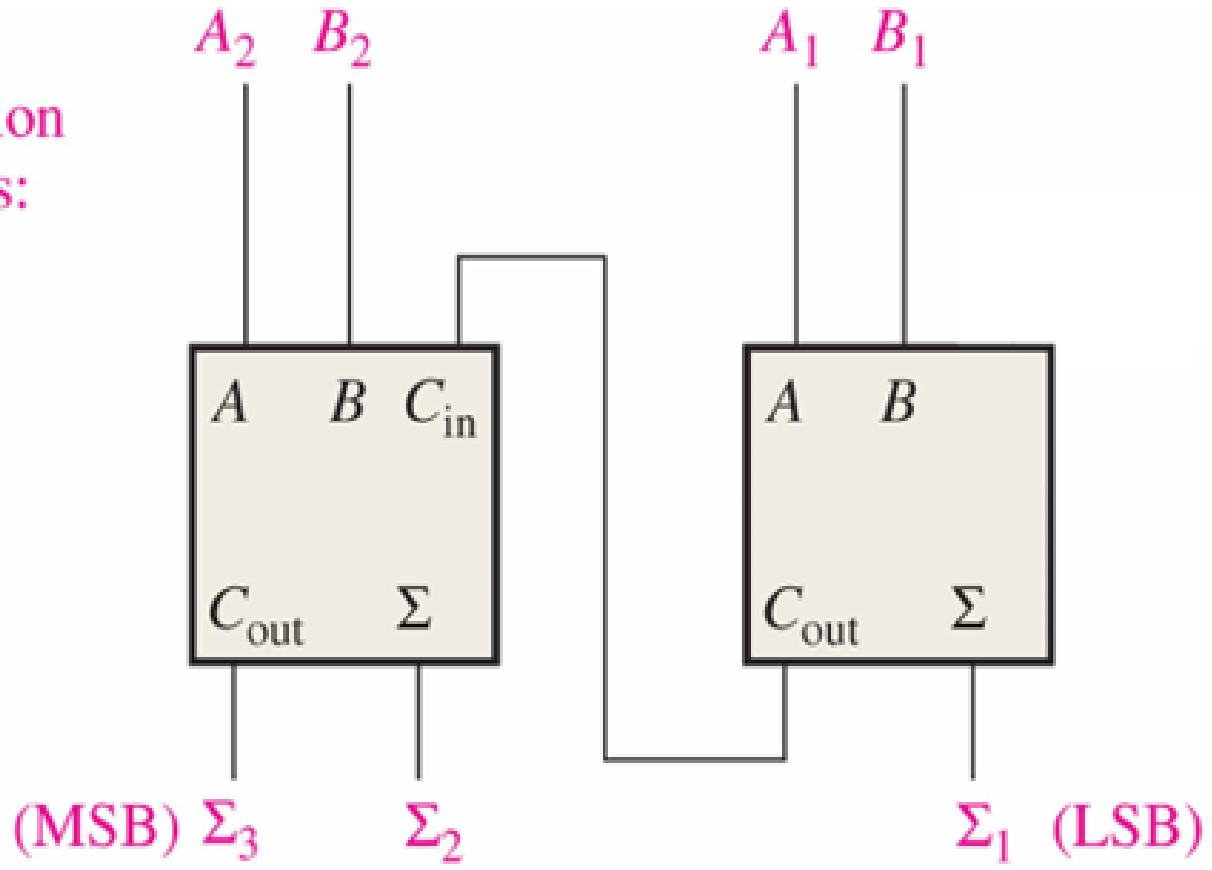
Therefore, $\Sigma = 0$ and $C_{out} = 1$.

2-bit parallel adder (I)

- 2-bit parallel adder logic circuit using half- and full-adders

General format, addition of two 2-bit numbers:

$$\begin{array}{r} A_2A_1 \\ +\ B_2B_1 \\ \hline \Sigma_3\Sigma_2\Sigma_1 \end{array}$$

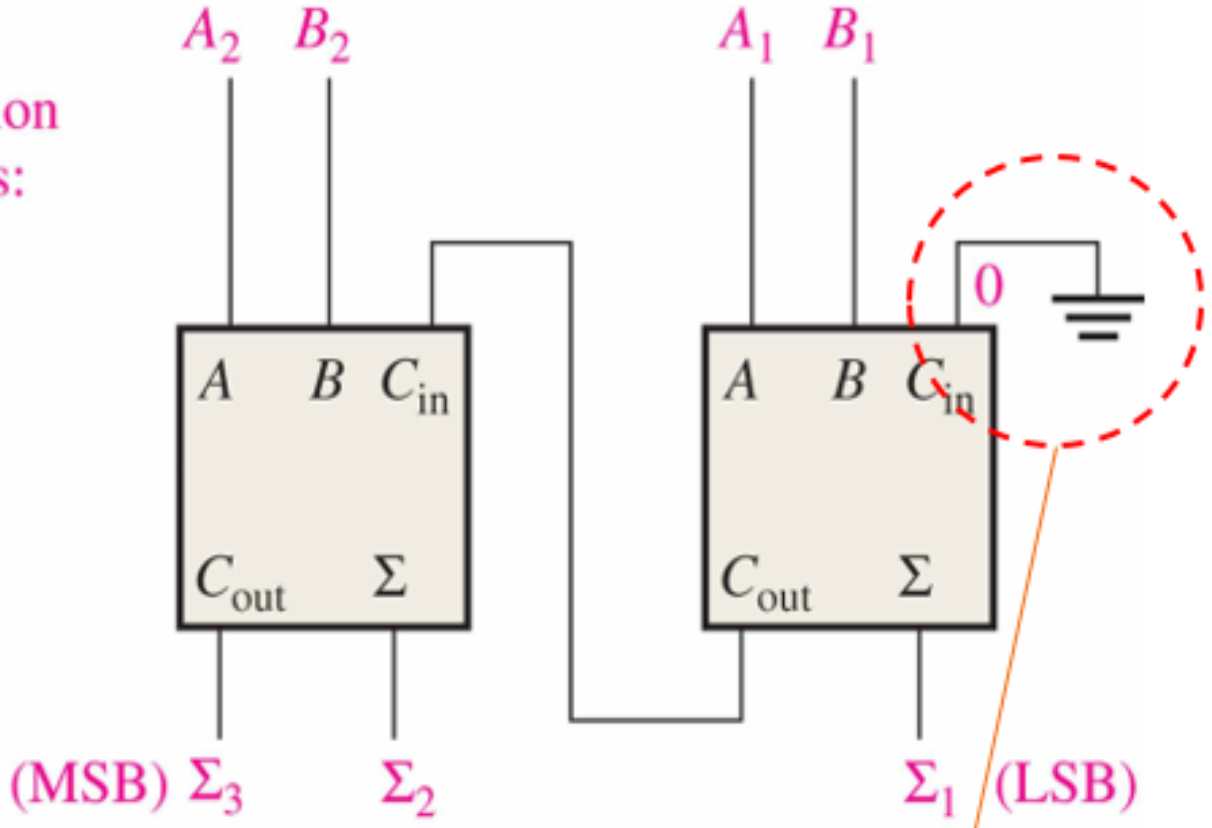


2-bit parallel adder (II)

- 2-bit parallel adder logic circuit using ***only full-adders***

General format, addition of two 2-bit numbers:

$$\begin{array}{r} A_2A_1 \\ +\,B_2B_1 \\ \hline \Sigma_3\Sigma_2\Sigma_1 \end{array}$$



Note how the first FA carry in is grounded since LSB addition has no use for the carry in

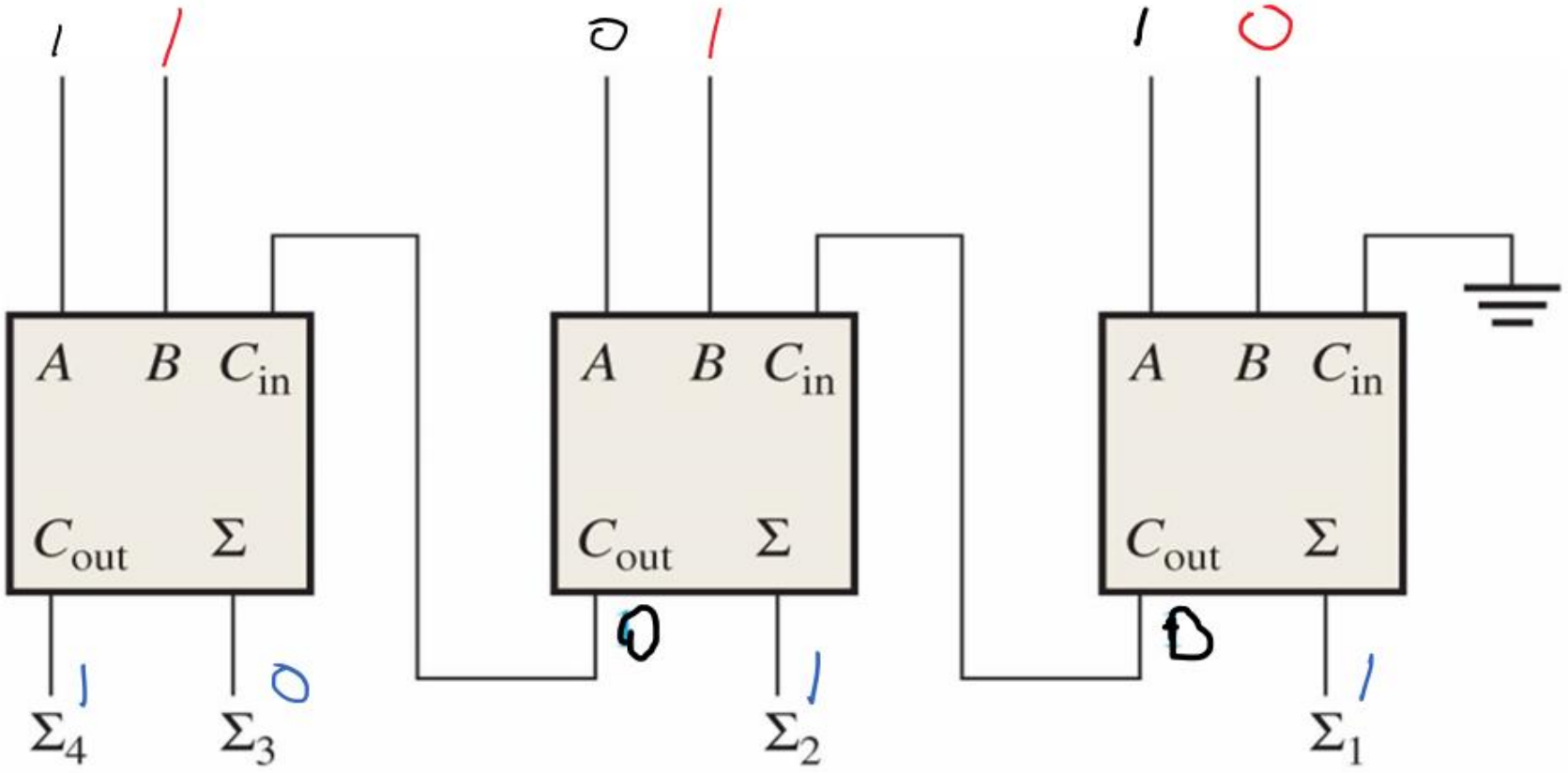
Exercise

- Determine the sum generated by the 3-bit adder below and show the intermediate carries
- What is the result when 101 and 110 are added together?

101_2 $A_3A_2A_1$

$+\ 110_2$ $+B_3B_2B_1$

1011_2 $\Sigma_4\Sigma_3\Sigma_2\Sigma_1$

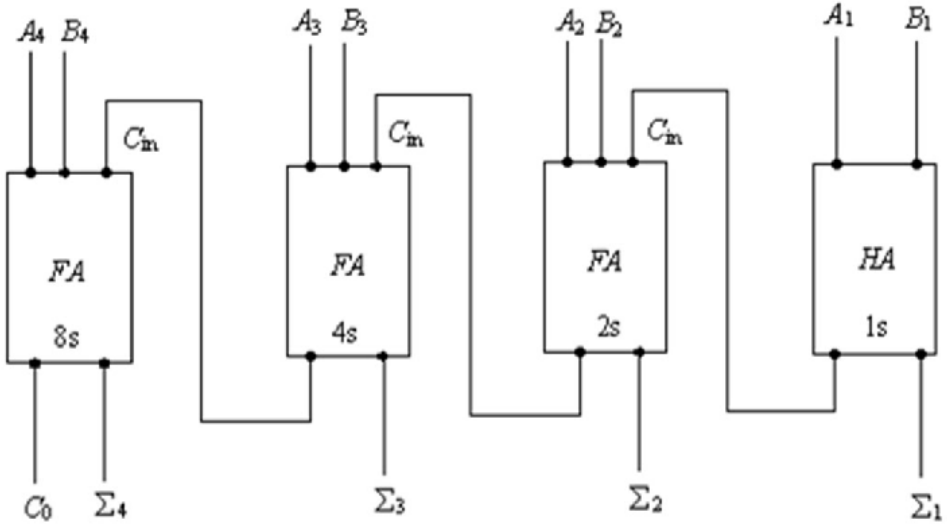


Parallel Binary Adder - Four bit

4-bit Parallel Binary Addition - An example

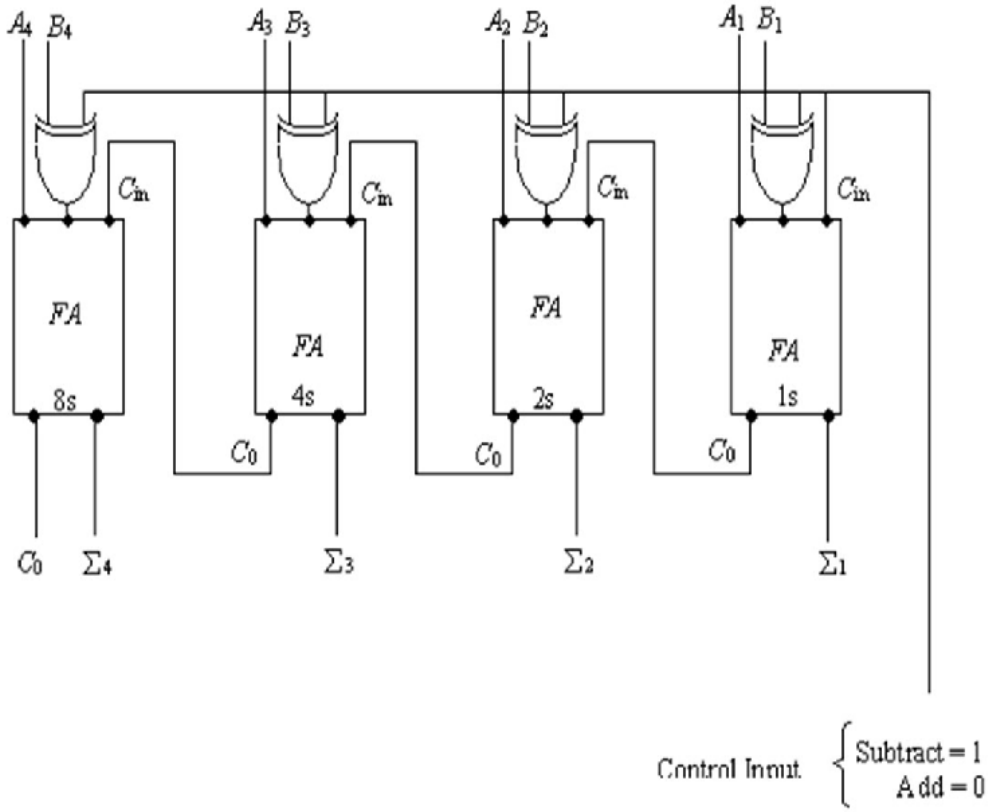
Subscript I:	**4**	**3**	**2**	**1**	
Input carry	**0**	**1**	**1**	**0**	C_{in}
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	$\sum_i$
Output carry	**0**	**0**	**1**	**1**	C_0

4-bit Parallel Binary Adder- Construction diagram



- An n-bit parallel adder requires (n-1) full adders and one half adder with each carry output connected to the input carry of the next higher-order full adder.
- A four bit parallel adder using 3 *Full Adders* and 1 *Half Adder* is shown.
- Half–adder adds the 1^s column A_1 and B_1.
- The 2^s, 4^s and 8^s columns being added by three *Full Adders.*

Parallel Binary Adder/Subtractor - Four bit



4 Bit parallel adder / subtractor

- 4 bit parallel adder / subtractor Circuit can be constructed using Full Adders and XOR gates as shown.
- The logic circuit has an additional input called the control input, which determines whether to perform the addition or subtraction.
- If this control input is logic 0, all four XOR gates have no effect on the data on the input line *B*.

  IF C_{in} of the first Full Adder is held low, it works as a **four bit parallel adder**
- When the control input is at logic 1 the XOR gates work as a inverter and the circuit performs as **four bit parallel subtractor**

BCD Addition

* Procedure
* Example

BCD ADDITION
- Procedure

- Start from Least Significant 4-bit group, add the two BCD digits, using the rules for binary addition
- If a 4-bit sum is equal to or less than 9, it is a valid BCD number.
- If a 4-bit sum is greater than 9, or if a carry out of the 4-bit group is generated, it is an invalid result.
  - ✓ Add 6 (0110) to the 4-bit sum in order to skip the six invalid states.
  - ✓ If a carry results when 6 is added, simply add the carry to the next 4-bit group.

BCD ADDITION
- An Example

	1	1	
478	**0100**	**0111**	**1000**
+137	**0001**	**0011**	**0111**
	---	---	---
	0110	1011	1111
	0000	0110	0110
	---	---	---
615	**0110**	**0001**	**0101**
	---	---	---

Standard Combinational Logic Circuits – Part 2 of 3

- Decoders
- Encoders
- Priority Encoders
- Multiplexers
- Demultiplexers

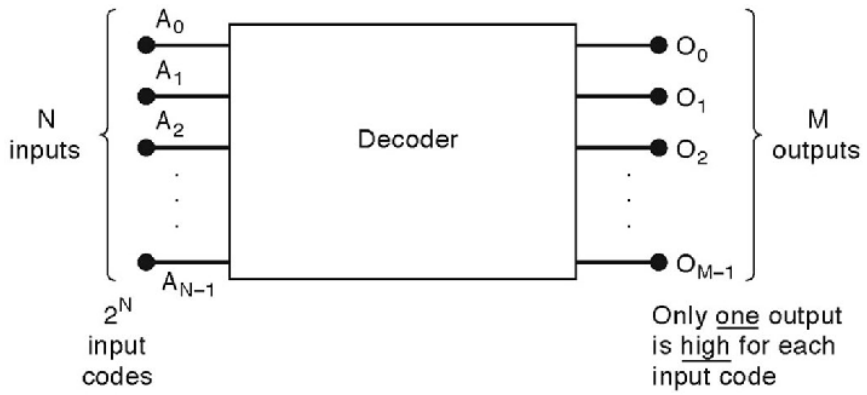
Decoders

* 1-to-2-Line Decoder
* 2-to-4-Line Decoder
* 3-to-8-Line Decoder
* Construction of higher order decoder using lower order decoders – An example
* Implementation of combinational logic circuit using decoder

DECODER
- Introduction

A decoder is a combinational logic circuit that accepts a set of inputs that represents a binary number and activates only the output that corresponds to the input number; all other outputs remain inactive.

A decoder can be considered as a combinational circuit that converts binary information from N input lines to a maximum of 2^N unique output lines.



This can be called as
N-to-M-line Decoders, where $M \leq 2^N$

The purpose is to generate 2^N or fewer minterms of N input variables.

DECODER - Examples

1-to-2-Line Decoder (Active HIGH output)

- For N=1, M=2, we get 1-to-2-line decoding function.

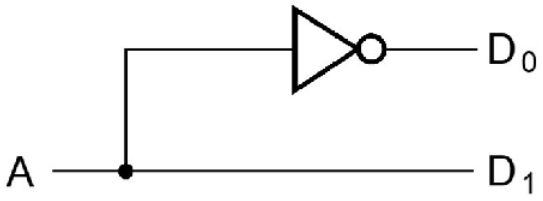
Truth Table:

Input	Outputs	
A	D_0	D_1
0	1	0
1	0	1

Boolean Functions:

$$D_0=(\overline{A})$$
$$D_1=(A)$$

Logic Diagram:



DECODER - Examples

2-to-4-Line Decoder (Active HIGH output)

- For N=2, M=4, we get 2-to-4-line decoding function.

Truth Table:

Inputs		Outputs			
A_1	A_0	D_0	D_1	D_2	D_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

Boolean Functions:

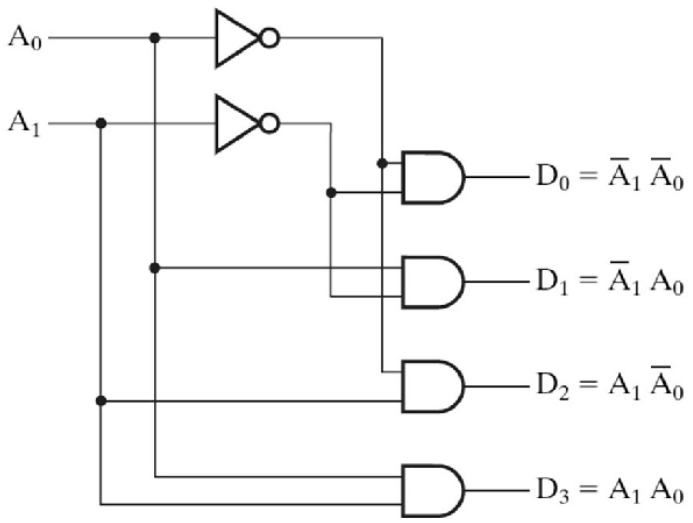
$$D_0 = \overline{A}_1 \overline{A}_0$$

$$D_1 = \overline{A}_1 A_0$$

$$D_2 = A_1 \overline{A}_0$$

$$D_3 = A_1 A_0$$

Logic Diagram:



DECODER
- Active HIGH and Active LOW outputs

If an active-LOW output (Example: 74138 IC, one of the output will low and the rest will be high) is required for each decoded number, the entire decoder can be implemented with

- NAND gates
- Inverters

If an active-HIGH output (Example: In 74139 IC, one of the output will high and the rest will be low) is required for each decoded number, the entire decoder can be implemented with

- AND gates
- Inverters

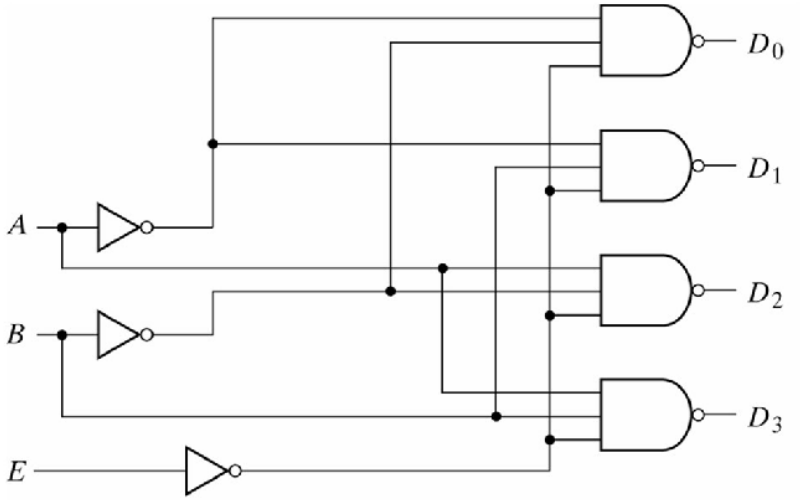
DECODER - Examples

2-to-4-Line Decoder with enable input (Active LOW output)

Truth Table:

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

Logic Diagram:



- The circuit operates with complemented outputs and a complement enable input. The decoder is enabled when E is equal to 0.
- Only one output can be equal to 0 at any given time, all other outputs are equal to 1.
- The output whose value is equal to 0 represents the minterm selected by inputs A and B
- The circuit is disabled when E is equal to 1.

DECODER - Examples

3-to-8-Line Decoder (Active HIGH output)

➢For N=3, M=8, we get 3-to-8-line decoding function.

Truth Table:

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

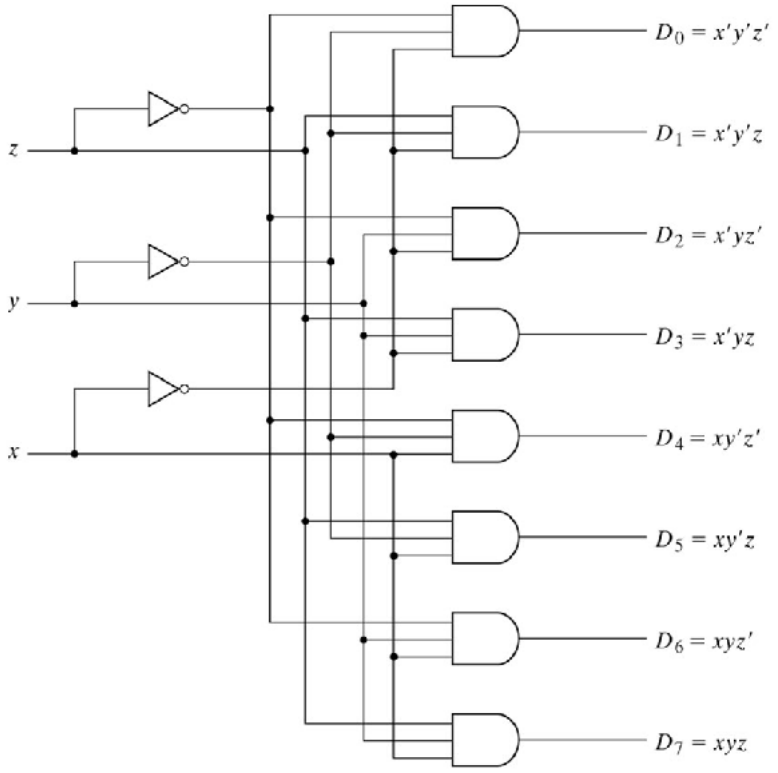
This decoder can be called in several ways.

It can be called as a **3-line-to- 8-line decoder**, because it has three input lines and eight output lines.

It can also be called as a **binary-octal decoder or converter** because it takes a three bit binary input code and activates the one of the eight outputs corresponding to that code.

It is also referred to as a **1-of-8 decoder**, because only 1 of the 8 outputs is activated at one time.

Logic Diagram:



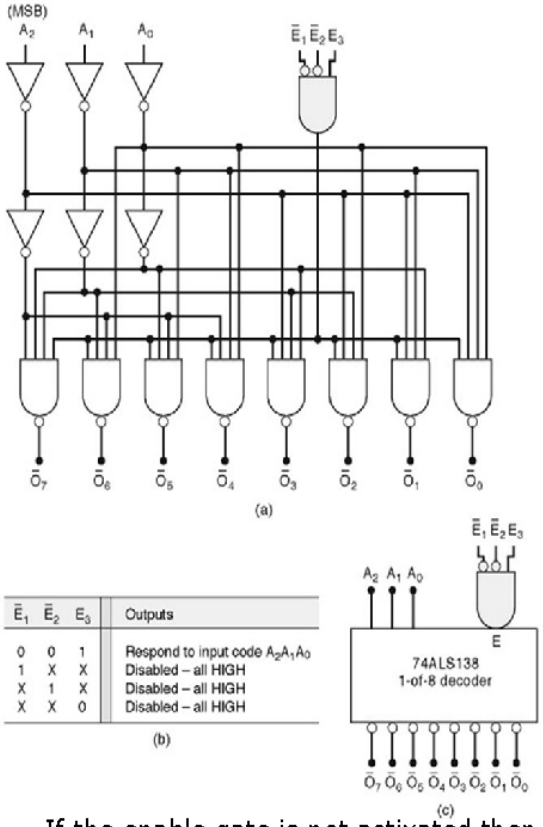
DECODER - Examples

3-to-8-Line Decoder (Active LOW output)

Truth Table (74138 IC):

Inputs						Outputs							
Enables			2^2	2^1	2^0	Active–*LOW*							
E_3	$\overline{E}_1$	$\overline{E}_2$	A_2	A_1	A_0	$\overline{O}_7$	$\overline{O}_6$	$\overline{O}_5$	$\overline{O}_4$	$\overline{O}_3$	$\overline{O}_2$	$\overline{O}_1$	$\overline{O}_0$
X	X	H	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H	**L**
H	L	L	L	L	H	H	H	H	H	H	H	**L**	H
H	L	L	L	H	L	H	H	H	H	H	**L**	H	H
H	L	L	L	H	H	H	H	H	H	**L**	H	H	H
H	L	L	H	L	L	H	H	H	**L**	H	H	H	H
H	L	L	H	L	H	H	H	**L**	H	H	H	H	H
H	L	L	H	H	L	H	**L**	H	H	H	H	H	H
H	L	L	H	H	H	**L**	H	H	H	H	H	H	H

Logic Diagram (74138 IC):

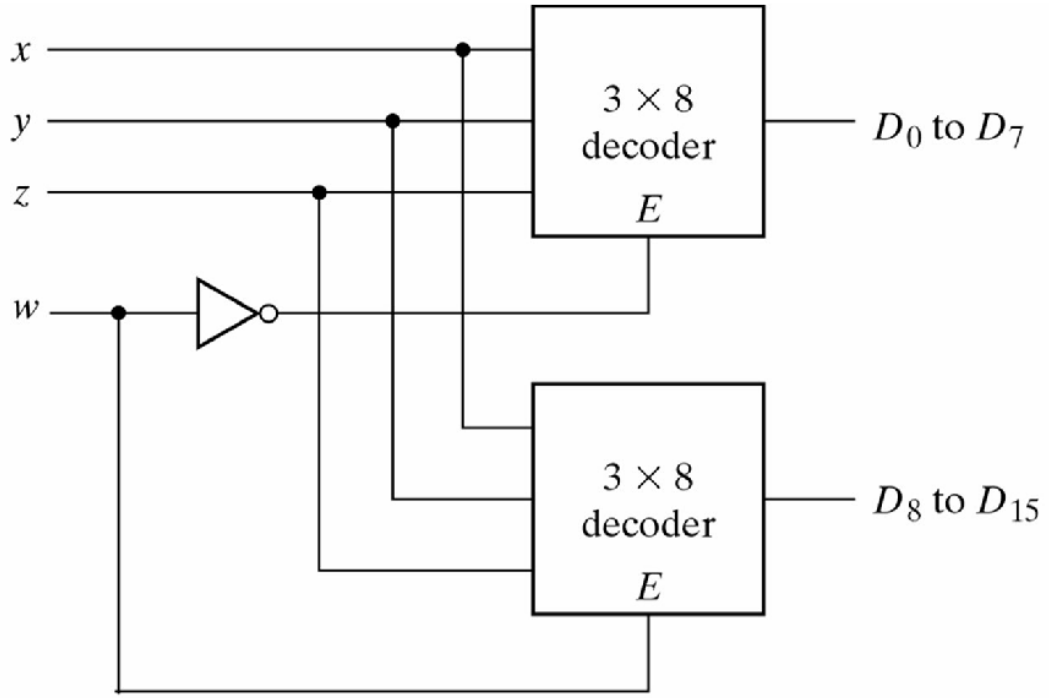


- There is an enable function on this device, a *LOW* level on each input E'_1, and E'_2, and a *HIGH* level on input E_3, is required in order to make the enable gate output *HIGH*.
- The enable gate output is connected to an input of each NAND gate in the decoder, so it must be *HIGH* for the NAND gate to be enabled.
- If the enable gate is not activated then all eight decoder outputs will be *HIGH* regardless of the states of the three input variables A_0, A_1, and A_2.

DECODER

- Construction of higher order decoder with lower order decoders

Example: Construction of 4-line-to-16 line Decoder with two 3-line-to-8 line decoders



- When w=0, the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's , and the top eight outputs generate minterms 0000 to 0111.
- When w=1, the enable conditions are reversed. The bottom decoder outputs generate minterms 1000 to 1111, while the outputs of the top decoder are all 0's.

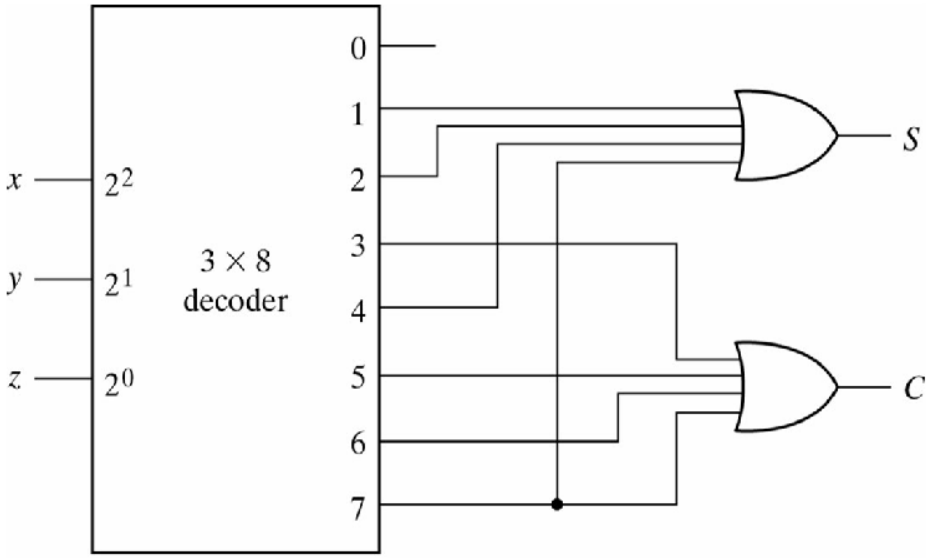
Implementation of combinational Logic circuit - Using Decoder with logic gates

- Any combinational logic circuit with n inputs and m outputs can be implemented with an n-to-2^n-line decoder and m OR gates.
- Procedure:
  - Express the given boolean function in sum of minterms
  - Choose a decoder to generate all the minterms of the input variables.
  - Select the inputs to each OR gate from the decoder outputs according to the list of minterms for each function.

Implementation of combinational Logic circuit - Using Decoder with logic gates (An example)

Inputs			Outputs	
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**

Implementation of a Full Adder with a Decoder



- From the truth table of the full adder, the functions can be expressed in sum of minterms.

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

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

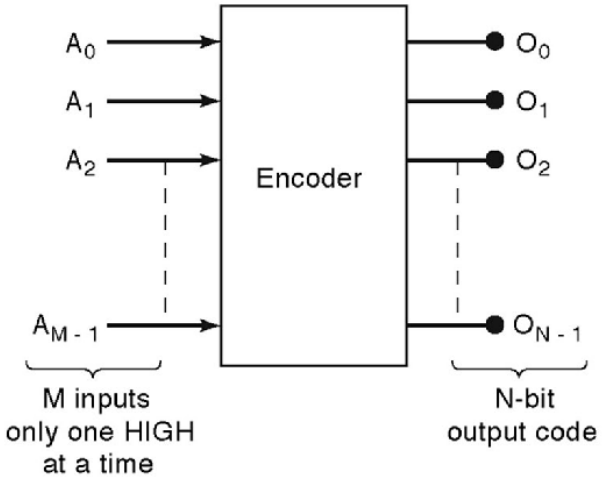
  where Σ indicates sum, m indicates minterm and the number in brackets indicate the decimal equivalent
- Since there are three inputs and a total of eight minterms, we need a 3-to-8 line decoder.
- The decoder generates the eight minterms for x,y,z
- The OR gate for output S forms the logical sum of minterms 1,2,4, and 7.
- The OR gates for output C forms the logical sum of minterms 3,5,6, and 7

Encoders

* 8-to-3-Line Encoder
* Design of 4-to-2-Line priority Encoder

ENCODER
- Introduction

- An encoder is a combinational logic circuit that essentially performs a "reverse" of decoder function.
- An encoder accepts an active level on one of its inputs, representing digit, such as a decimal or octal digits, and converts it to a coded output such as BCD or binary.
- Encoders can also be devised to encode various symbols and alphabetic characters.
- The process of converting from familiar symbols or numbers to a coded format is called encoding.



This can be called as
M-to-N-line Encoders, where $M \leq 2^N$

An encoder has 2^N or fewer input lines and N output lines.

The output lines generate the binary code corresponding to the input value.

ENCODER - Example

8-to-3-Line Encoder (Octal-to-binary encoder)

Truth Table:

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
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

Boolean functions

$$A_0 = D_1 + D_3 + D_5 + D_7$$
$$A_1 = D_2 + D_3 + D_6 + D_7$$
$$A_2 = D_4 + D_5 + D_6 + D_7$$

This encoder has eight inputs (one for each of the octal digits) and three outputs that generate the corresponding binary number.

This encoder can be implemented with OR gates whose inputs are determined directly from the truth table.

Output A_0 is equal to 1 when the input octal digit is 1,3,5, or 7. Output A_1 is 1 for octal digits 2,3,6, or 7 and output A_2 is 1 for digits 4,5, 6 or 7.

ENCODER - Example

8-to-3-Line Encoder (Octal-to-binary encoder)

Limitations:

- Only one input can be active at any given time.

If two inputs are active simultaneously, the output produces an undefined combination.

Example: If D_3 and D_6 are 1 simultaneously, the output of the encoder will be 111 because all three outputs are equal to 1. This does not represent either binary 3 or binary 6.

Remedy: Establish priority over inputs. This is to ensure that only one input is encoded.

- Output with all 0's is generated when all the inputs are 0. This output is same as when D_0 is equal to 1.

Remedy: Provide one more output to indicate that at least one input is equal to 1

PRIORITY ENCODER - Example

Design of 4-to-2-Line Priority Encoder (4-input priority encoder)

- A priority encoder is an encoder that includes the priority function
- If two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.
- Truth Table of a 4-input Priority Encoder:

INPUTS				OUTPUTS		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

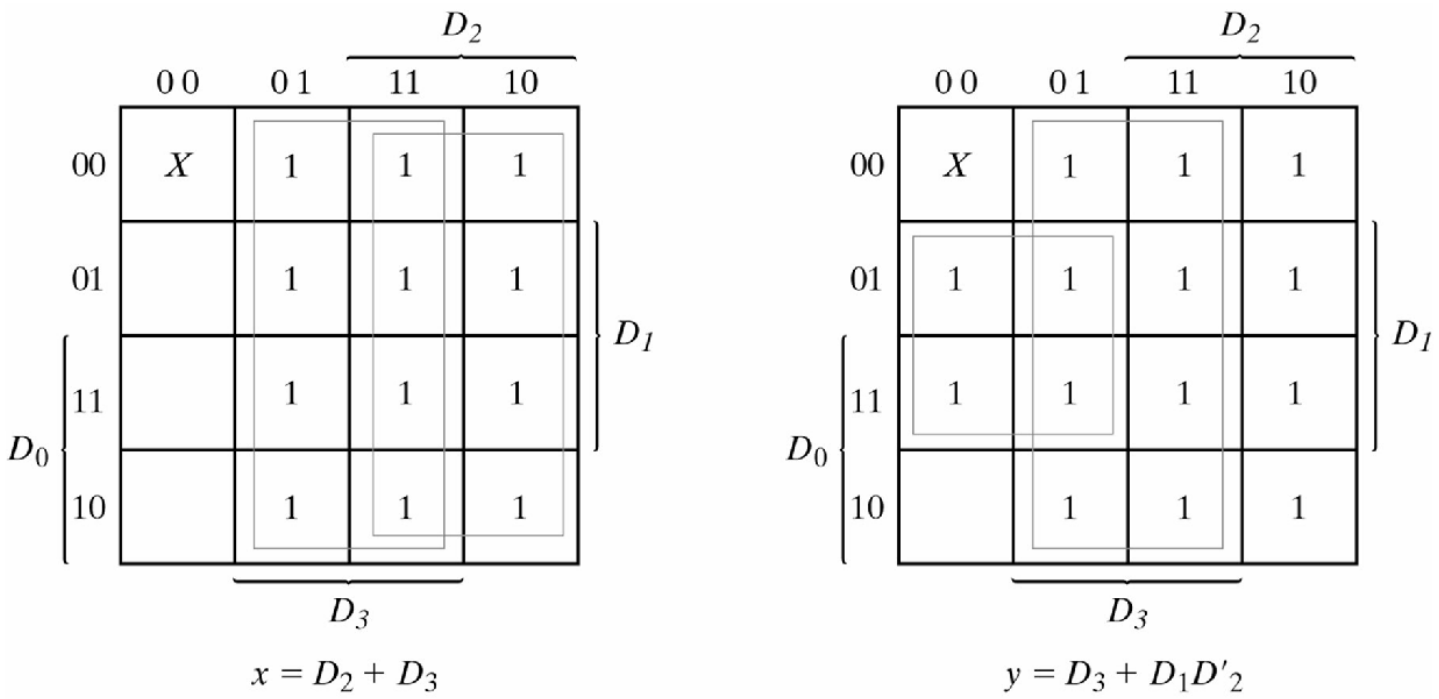
PRIORITY ENCODER - Example

Design of 4-to-2-Line Priority Encoder (4-input priority encoder)

- In addition to two outputs x, and y, the truth table has a third output designated by V, which is a valid bit indicator that is set 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0.
- X's in the output column indicate don't care conditions, the X's in the input columns are useful for representing a truth table in condensed form.
- The higher the subscript number, the higher the priority of the input. Input D_3 has the highest priority, so regardless of the values of the other inputs, when this input is 1, the output for xy is 11 (binary 3)

PRIORITY ENCODER - Example

Design of 4-to-2-Line Priority Encoder (4-input priority encoder)

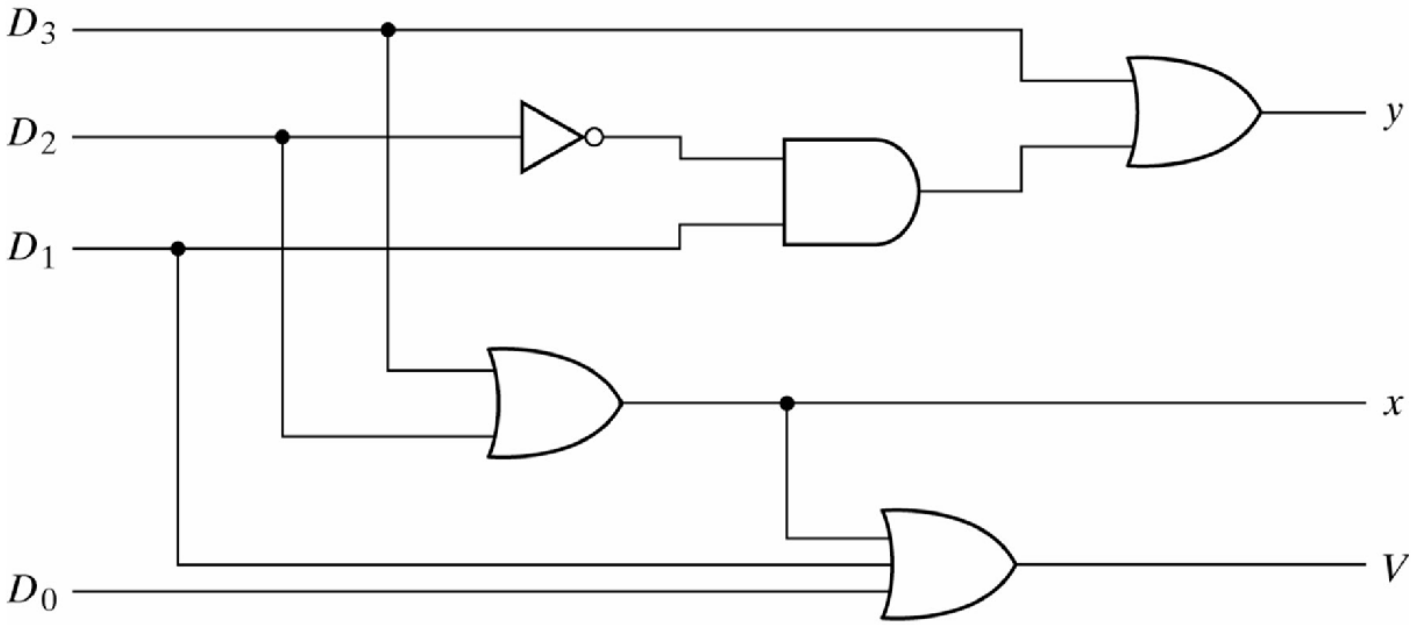


$V = D_0 + D_1 + D_2 + D_3$

K-Maps for 4-input Priority Encoder

PRIORITY ENCODER - Example

Design of 4-to-2-Line Priority Encoder (4-input priority encoder)



Logic Diagram for 4-input priority encoder

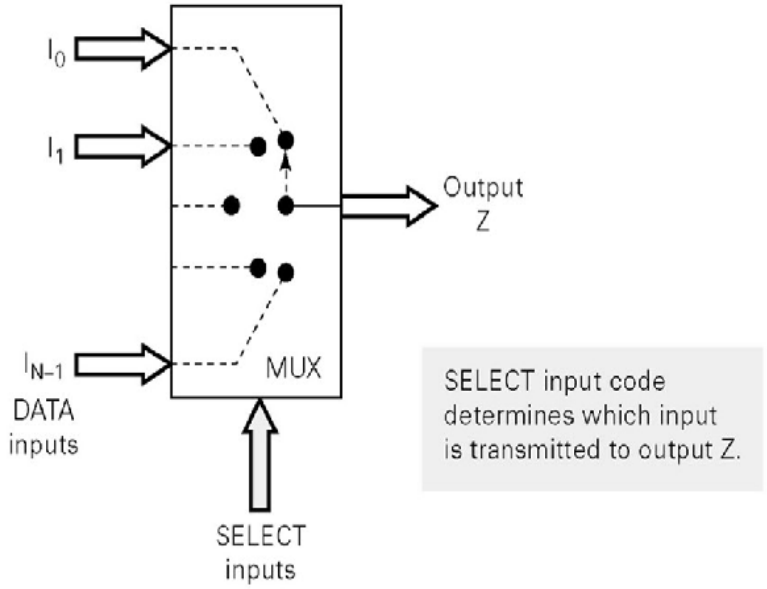
Multiplexers (MUX)

* 2-to-1-Line MUX
* 4-to-1-Line MUX
* Implementation of Boolean Function using MUX
  - Two methods

Multiplexer (Data Selector)
- Introduction

A multiplexer (MUX) is a combinational logic circuit that selects binary information from one of many input lines and directs it to a single output line.

It can be considered as a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination.



A multiplexer has 2^N data input lines and N selection lines, whose bit combination determine which input is selected and routed to a single output line.

Multiplexer (Data Selector) - Examples

2-to-1- line MUX (2 input MUX)

Truth Table:

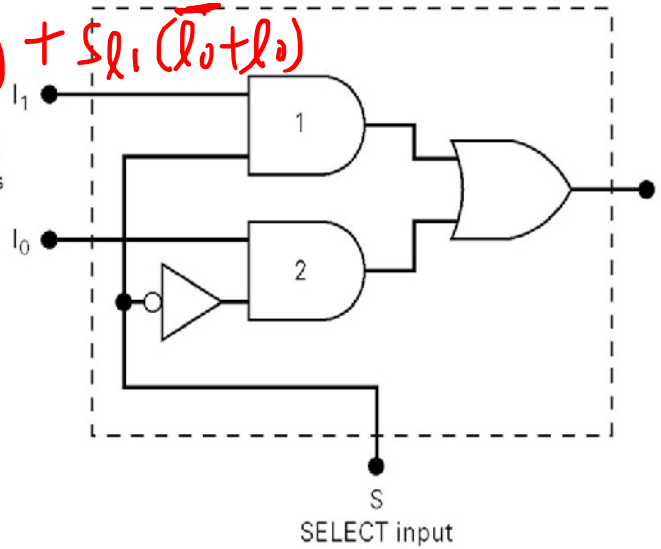
S	I_0	I_1	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$Y = \bar{S} I_0 \bar{I}_1 + \bar{S} I_0 I_1 + S \bar{I}_0 I_1 + S I_0 I_1$

$= \bar{S} I_0 (\bar{I}_1 + I_1) + S I_1 (\bar{I}_0 + I_0)$

$= \bar{S} I_0 + S I_1$

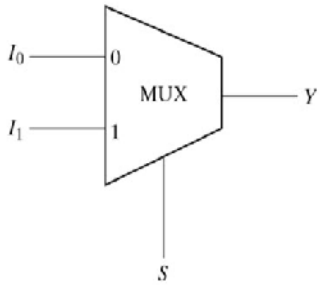
Logic Diagram and Function Table:



$Z = I_0 \cdot \bar{S} + I_1 \cdot S$

S	Output
0	$Z = I_0$
1	$Z = I_1$

Block diagram:

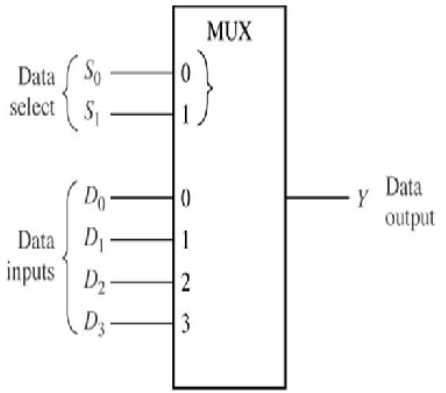


Multiplexer (Data Selector) - Examples

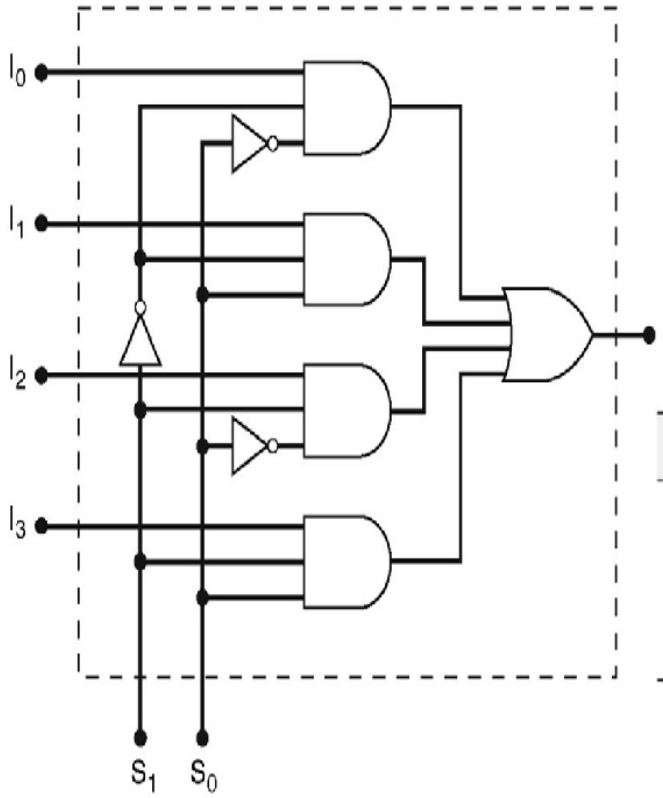
4-to-1- line MUX (4 input MUX)

Logic Diagram and Function Table:

Logic Symbol:



Input	Select Lines	Output Lines
I	S_1 S_0	D_0 D_1 D_2 D_3
I	0 0	1 0 0 0
I	0 1	0 1 0 0
I	1 0	0 0 1 0
I	1 1	0 0 0 1



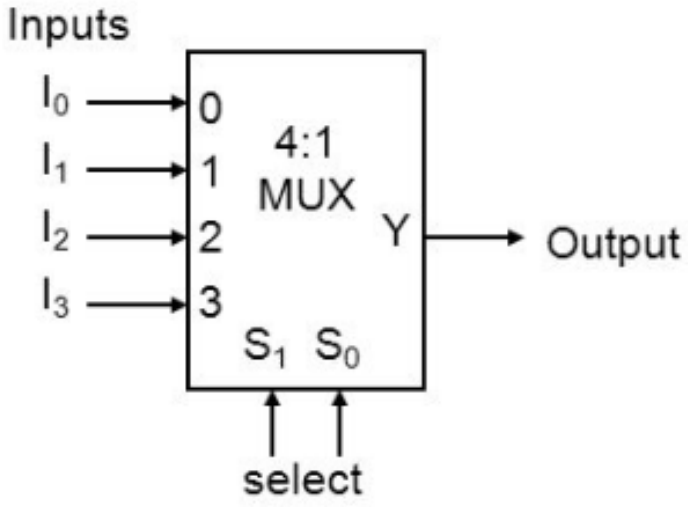
S_1	S_0	Output
0	0	$Z = I_0$
0	1	$Z = I_1$
1	0	$Z = I_2$
1	1	$Z = I_3$

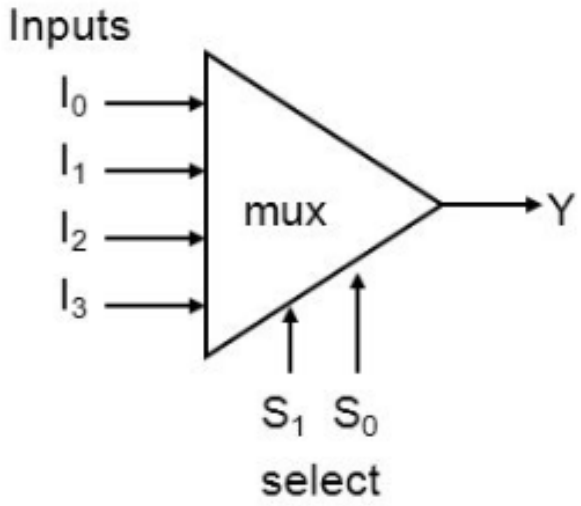
Multiplexer

- Truth table for a 4-to-1 multiplexer:

I_0	I_1	I_2	I_3	S_1	S_0	Y
d_0	d_1	d_2	d_3	0	0	d_0
d_0	d_1	d_2	d_3	0	1	d_1
d_0	d_1	d_2	d_3	1	0	d_2
d_0	d_1	d_2	d_3	1	1	d_3

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

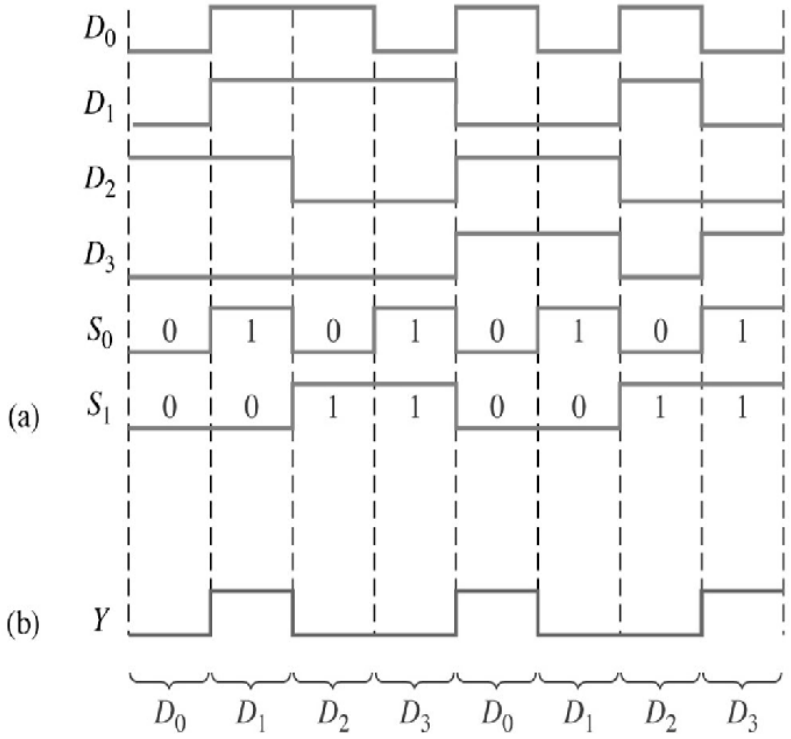




Multiplexer (Data Selector) - Examples

4-to-1- line MUX (4 input MUX)

Output Waveforms in relation with the Data-Input and Data-Select waveforms



The binary state of the data-select inputs during each interval determines which data input is selected.

Here the data-select inputs go through a repetitive binary sequence 00,01,10,11,00, and so on.

The resulting output waveform is shown.

Multiplexer (Data Selector)

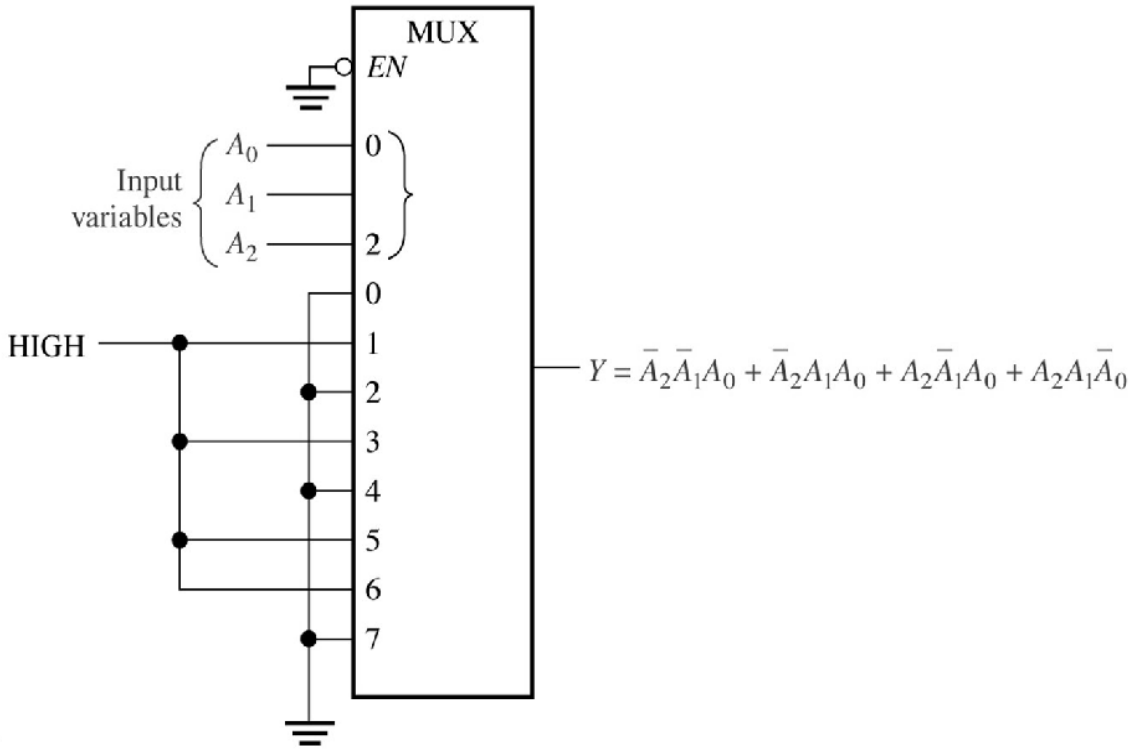
- Implementation of Boolean Function using MUX (Method I)

Problem 1:

Implement the logic circuit function specified in the table given below by using 74LS151 8-input data selector/multiplexer.

A_2	A_1	A_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Solution:



Multiplexer (Data Selector)

- Implementation of Boolean Function using MUX (Efficient Method)

- An efficient method for implementing a boolean function of n variables with a MUX that has n-1 selection inputs and 2^{n-1} data inputs is given below:
  - List the Boolean function in a truth table
  - Apply the first n-1 variables in the table to the selection inputs of the MUX.
  - For each combination of the selection variables, evaluate the output as a function of the last variable. This function can be 0,1, the variable, or the complement of the variable. Apply these values to the data inputs in the proper order.

Multiplexer (Data Selector)

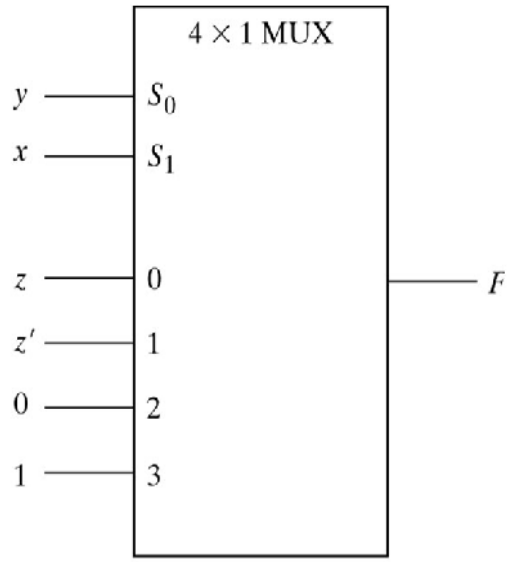
- Implementation of Boolean Function using MUX (Efficient Method)

Problem 2:
Implement the boolean function **F=x'y'z +x'yz' + xyz' + xyz** using a suitable MUX

Solution:

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

- The two variables x and y are applied to the selection lines in that order; x is connected to the S_1 input and y to the S_0 input.
- The values for the data input lines are determined from the truth table of the function
  - For example:

    when xy=00, output F is equal to z because F=0 when z=0 and F=1 when z=1.

    This requires that variable z is applied to the data input 0

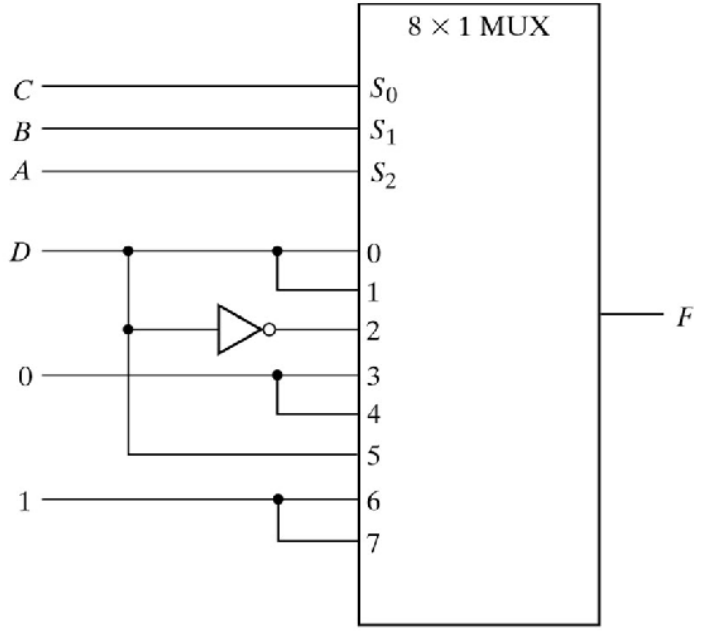
Multiplexer (Data Selector)

- Implementation of Boolean Function using MUX (Efficient Method)

Problem 3:
Implement the boolean function
F=A′B′C′D+A′B′CD+A′BC′D′+AB′CD+ABC′D′+ABC′D+ABCD′+ABCD using a suitable MUX

Solution:

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	



Demultiplexers (DMUX)

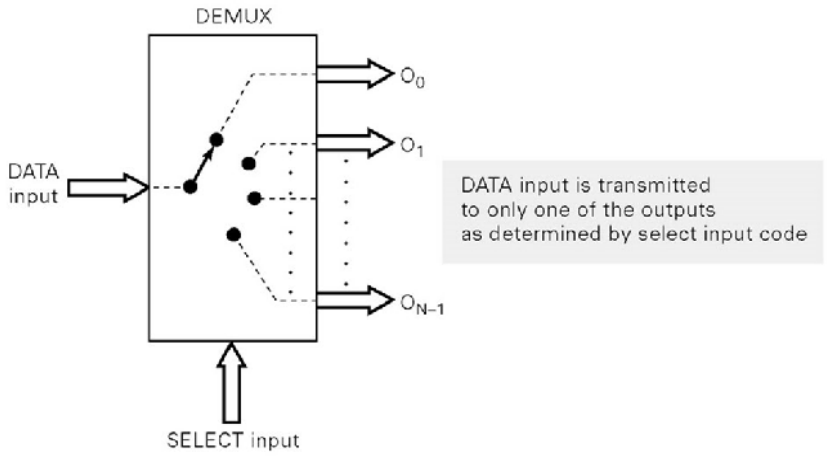
* 1-to-4-Line DMUX

Demultiplexer (Data Distributor) - Introduction

A Demultiplexer (DEMUX) basically reverses the multiplexing function.

It takes data from one line and distributes them to a given number of output lines.

For this reason, the demultiplexer is also known as a data distributor.

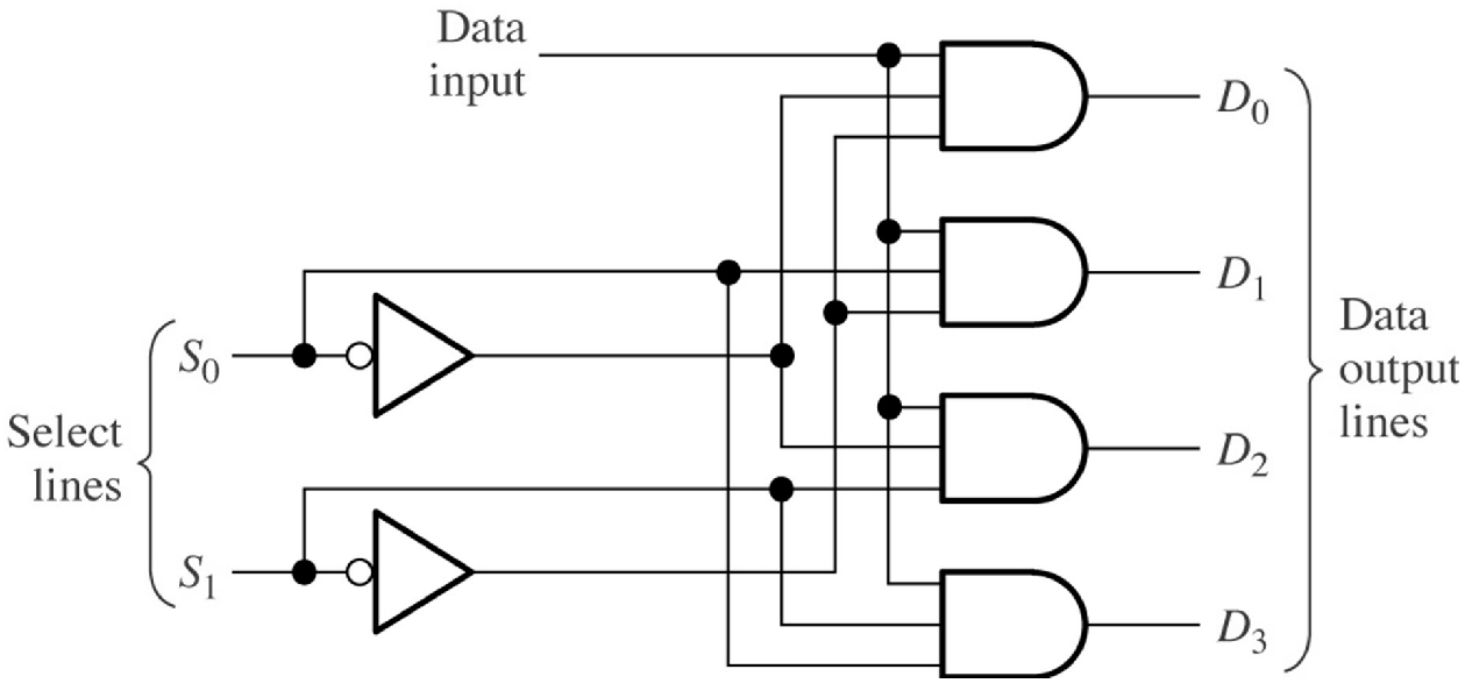


Demultiplexer takes one data input source and selectively distributes it to 1 of N output channels just like multiposition switch.

Demultiplexer (Distributor) - Example

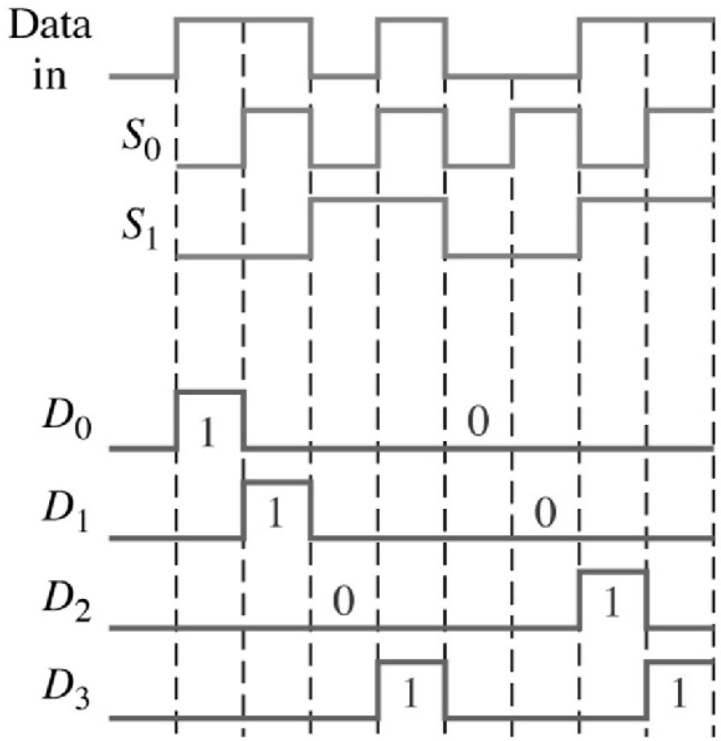
1-to-4- line DMUX

Logic Diagram :



Demultiplexer (Distributor) - Example
1-to-4- line DMUX

The serial data input waveform (Data in) and data select inputs (S_o and S_1) and the corresponding data output waveforms (D_0 through D_3) are shown below



The select lines go through a binary sequence so that each successive input bit is routed to D_0, D_1, D_2, and D_3 in sequence.

Standard Combinational Logic Circuits – Part 3 of 3

- Comparators
- Code Converters
- Parity Generators/Checkers

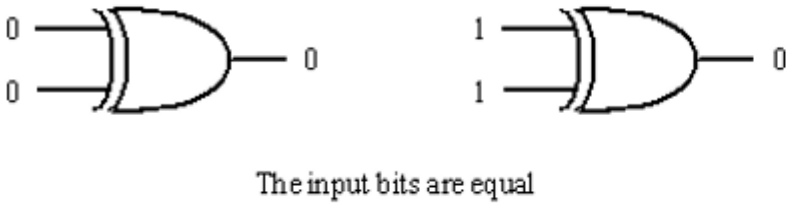
Comparators

* 2-bit magnitude comparator
* 4-bit magnitude comparator
* 8-bit magnitude comparator using two 4-bit comparators

COMPARATORS
- Introduction

Comparator is a combinational logic circuit that compares the magnitudes of two binary quantities to determine which one has the greater magnitude.

In other words, a comparator determines the relationship of two binary quantities.



Basic 2–bit comparator.

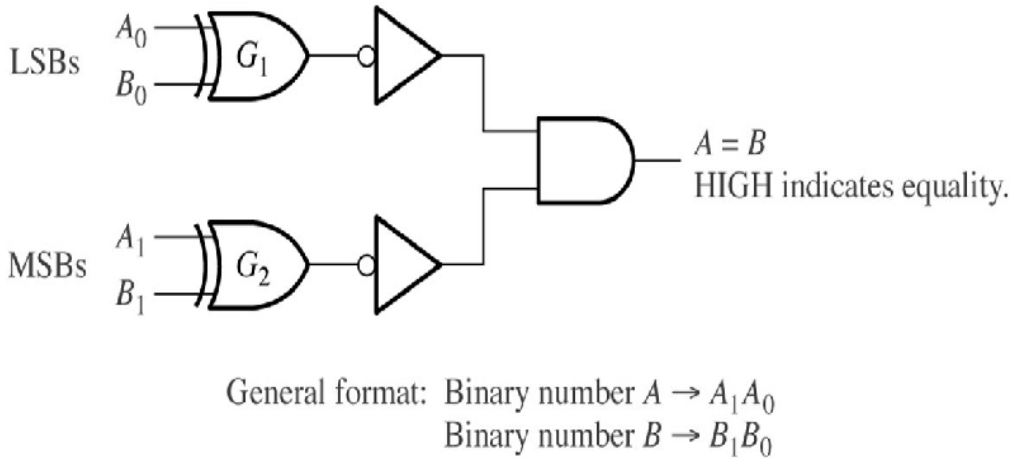
If two input bits are not equal, its output is a 1.

But if two input bits are equal, its output is a 0.

So exclusive–OR gate can be used as a 2–bit comparator.

COMPARATORS – An example
2-bit magnitude comparator

Logic diagram for equality comparison of two 2-bit numbers



XOR gate and inverter can be replaced by an XNOR symbol.

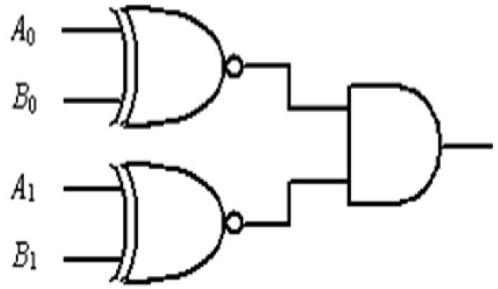
- In order to compare binary numbers containing two bits each, an additional XOR gate is necessary
- 2 LSB of two numbers are compared by gate G_1
- 2 MSB of two numbers are compared by gate G_2
- 2 Inverters and 1 AND gate can be used

COMPARATORS – An example
2-bit magnitude comparator

Problem:

Determine the output for the following sets of binary numbers to the comparator inputs in figure below.

(a) 10 and 10 (b) 11 and 10

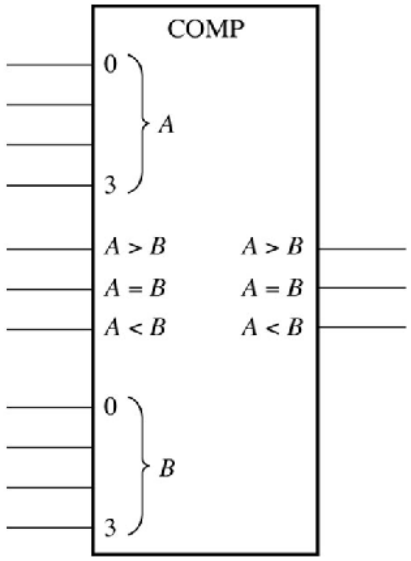


Solution

(a)The output is **1** (b) The output is **0**

COMPARATORS – An example
4-bit magnitude comparator

Logic symbol for 7485
4-bit magnitude comparator

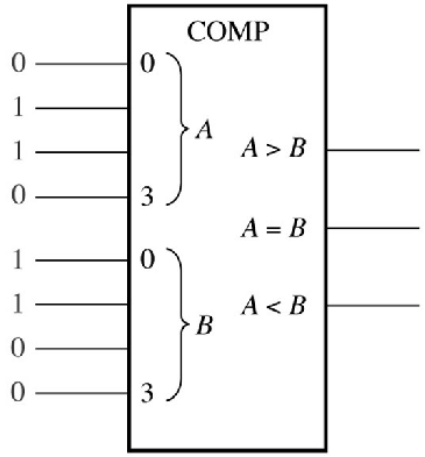


- Compares two unsigned 4-bit binary numbers , A_3, A_2, A_1, A_0 and B_3, B_2, B_1, B_0.
- It has three active-HIGH outputs.
- Start with most significant bit in each number to determine the inequality of 4-bit binary numbers A and B
- Output $A<B$ will be HIGH if $A_3=0$, and $B_3=1$
- Output $A>B$ will be HIGH if $A_3=1$, and $B_3=0$
- If $A_3=0$, and $B_3=0$ or $A_3=1$, and $B_3=1$, then examine the next lower order bit position for an inequality.
- Only when all bits of A=B, output A=B will be HIGH

- In addition, it also has three cascading inputs:
- These inputs provides a means for expanding the comparison operation by cascading two or more 4–bit comparators.
- To expand the comparator, the $A<B$, $A=B$, and $A>B$ outputs of the lower–order comparator are connected to the corresponding cascading inputs of the next higher–order comparator.

COMPARATORS – An example
4-bit magnitude comparator

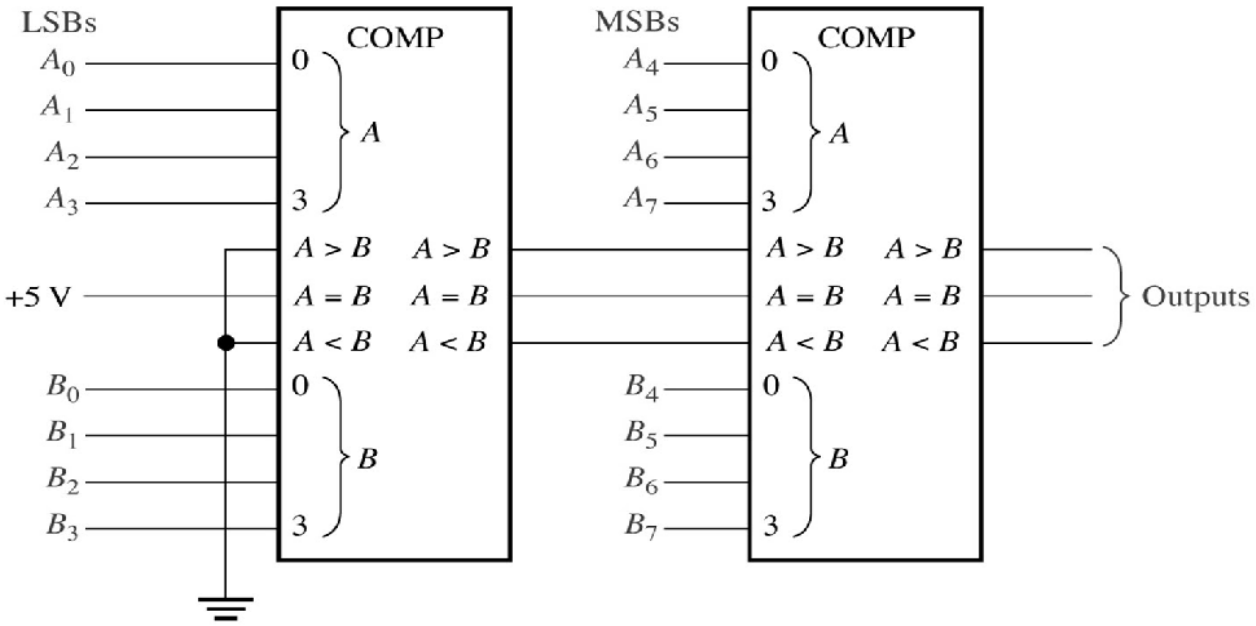
Problem: Determine the A=B, A>B, and A<B outputs for the input numbers shown on the 4-bit comparator as given below



Solution:
The number on the A inputs is 0110 and the number on the B inputs is 0011. The A > B output is HIGH and the other outputs (A=B and A<B) are LOW

COMPARATORS

- 8-bit magnitude comparator using two 4-bit magnitude comparators



Code Converters

* Binary to Gray code converter
* Gray code to Binary converter
* BCD to Excess 3 code converter

CODE CONVERTERS
- Introduction

- A code converter is a combinational logic circuit that changes data presented in one type of binary code to another type of binary code

Examples:

- BCD to binary
- BCD to 7–segment,
- binary to BCD,
- BCD to XS3,
- binary to Gray code
- Gray code to binary.

CODE CONVERTERS

Design of 4-Bit Binary to Gray Code Converter

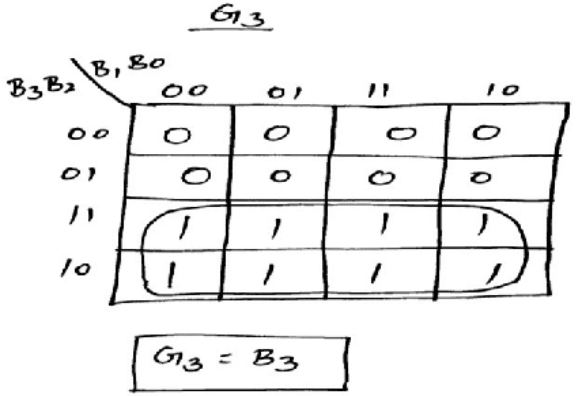
TRUTH TABLE:

INPUTS (BINARY)				OUTPUTS (GRAY CODE)			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

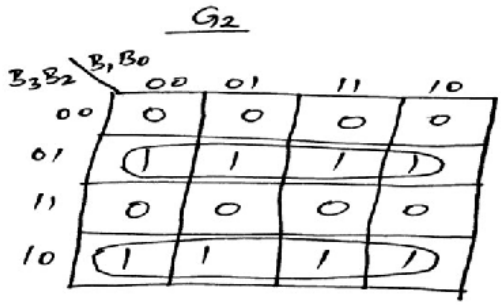
CODE CONVERTERS

Design of 4-Bit Binary to Gray Code Converter

Simplification using K-maps:

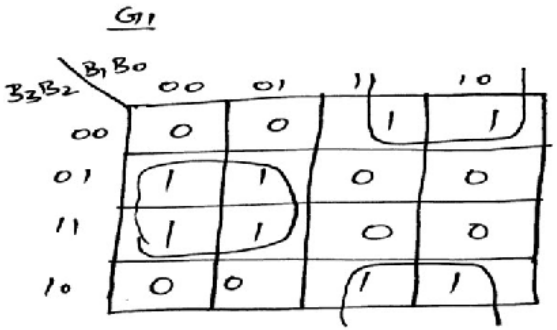


$G_{13} = B_3$



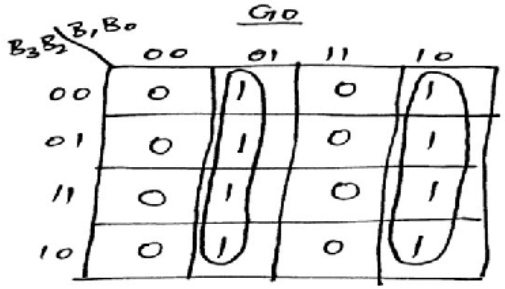
$G_2 = \overline{B_3}\, B_2 + B_3\, \overline{B_2}$

$G_2 = B_3 \oplus B_2$



$G_1 = B_2\, \overline{B_1} + \overline{B_2}\, B_1$

$G_1 = B_2 \oplus B_1$



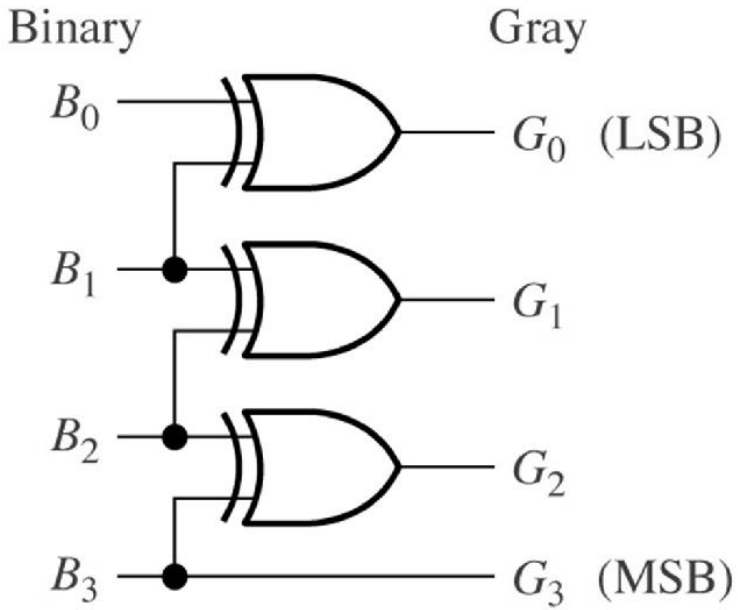
$G_0 = \overline{B_1}\, B_0 + B_1\, \overline{B_0}$

$G_0 = B_1 \oplus B_0$

CODE CONVERTERS

Design of 4-Bit Binary to Gray Code Converter

Logic Diagram:



CODE CONVERTERS

Design of 4-Bit Gray code to Binary Converter

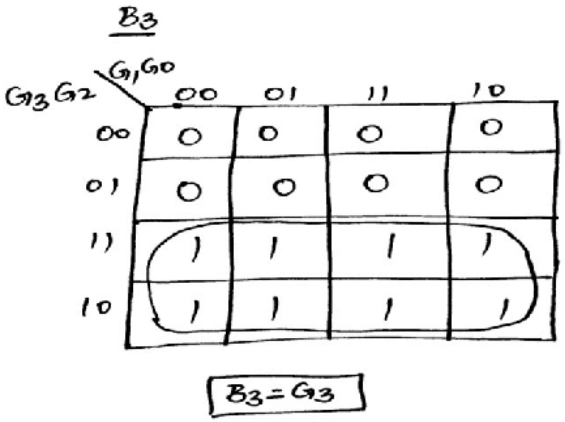
- Truth Table:

INPUTS (GRAY CODE)				OUTPUTS (BINARY)			
G_{I3}	G_{I2}	G_{I1}	G_{I0}	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

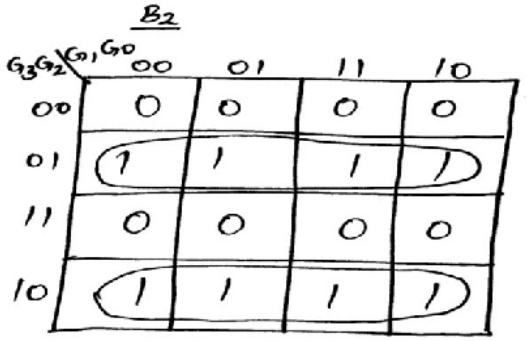
CODE CONVERTERS

Design of 4-Bit Gray code to Binary Converter

Simplification using K-Maps:

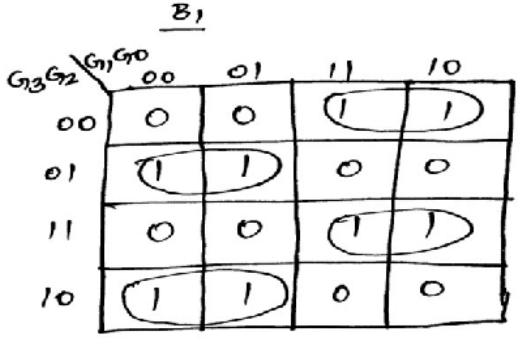


$B_3 = G_3$



$B_2 = \overline{G_3} G_2 + G_3 \overline{G_2}$

$B_2 = G_3 \oplus G_2$



$B_1 = \overline{G_3}\,\overline{G_2} G_1 + G_3 G_2 G_1 + \overline{G_3} G_2 \overline{G_1} + G_3 \overline{G_2}\,\overline{G_1}$

$= G_1 (\overline{G_3}\,\overline{G_2} + G_3 G_2) + \overline{G_1} (\overline{G_3} G_2 + G_3 \overline{G_2})$

$= G_1 (\overline{G_3 \oplus G_2}) + \overline{G_1} (G_3 \oplus G_2)$

$= G_1 \oplus G_3 \oplus G_2$

$B_1 = G_1 \oplus B_2$

CODE CONVERTERS

Design of 4-Bit Gray code to Binary Converter

Simplification using K-Maps:

B_0

G_3G_2 \ G_1G_0	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

$$B_0 = \bar{G}_3\bar{G}_2\bar{G}_1G_0 + \bar{G}_3\bar{G}_2G_1\bar{G}_0 + \bar{G}_3G_2\bar{G}_1\bar{G}_0 + \bar{G}_3G_2G_1G_0 + G_3G_2\bar{G}_1G_0 + G_3G_2G_1\bar{G}_0 + G_3\bar{G}_2\bar{G}_1\bar{G}_0 + G_3\bar{G}_2G_1G_0$$

$$= \bar{G}_3\bar{G}_2(\bar{G}_1G_0 + G_1\bar{G}_0) + \bar{G}_3G_2(\bar{G}_1\bar{G}_0 + G_1G_0) + G_3G_2(\bar{G}_1G_0 + G_1\bar{G}_0) + G_3\bar{G}_2(\bar{G}_1\bar{G}_0 + G_1G_0)$$

$$= \bar{G}_3\bar{G}_2(G_1 \oplus G_0) + \bar{G}_3G_2(\overline{G_1 \oplus G_0}) + G_3G_2(G_1 \oplus G_0) + G_3\bar{G}_2(\overline{G_1 \oplus G_0})$$

$$= (G_1 \oplus G_0)(\bar{G}_3\bar{G}_2 + G_3G_2) + (\overline{G_1 \oplus G_0})(\bar{G}_3G_2 + G_3\bar{G}_2)$$

$$= (G_1 \oplus G_0)(\overline{G_3 \oplus G_2}) + (\overline{G_1 \oplus G_0})(G_3 \oplus G_2)$$

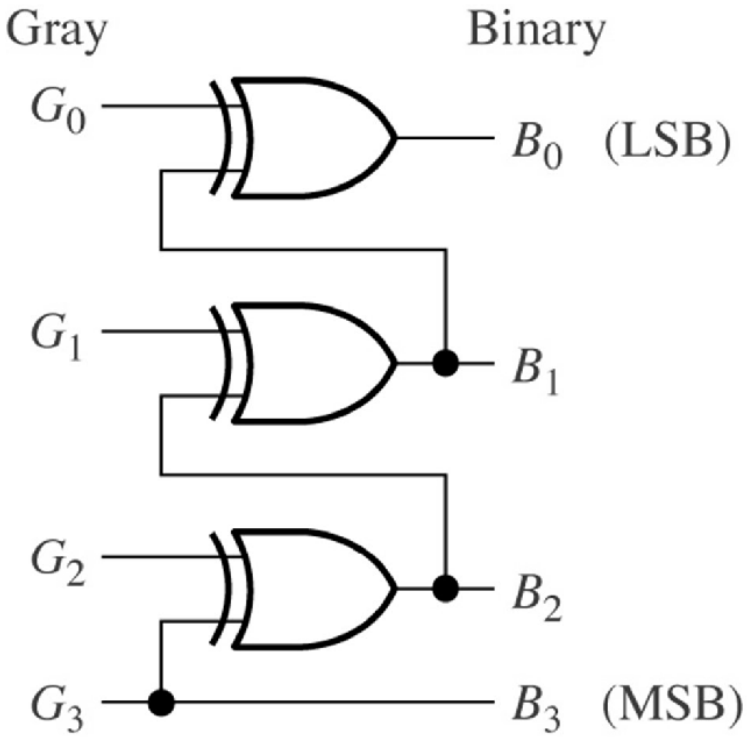
$$= G_0 \oplus G_1 \oplus G_2 \oplus G_3$$

$$\boxed{B_0 = G_0 \oplus B_1}$$

CODE CONVERTERS

Design of 4-Bit Gray code to Binary Converter

Logic Diagram:



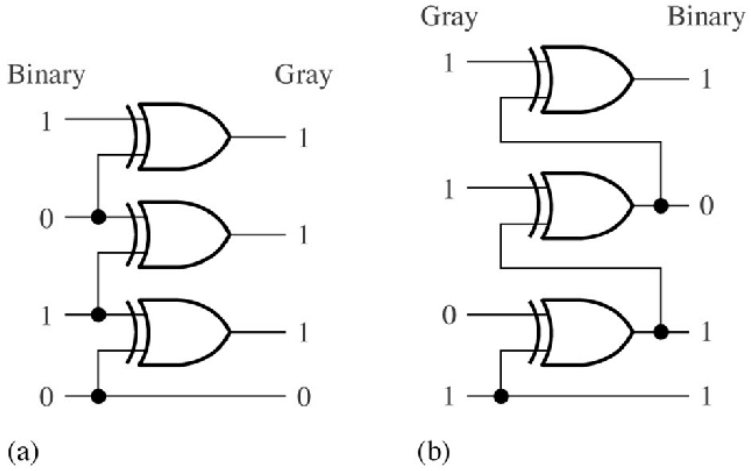
CODE CONVERTERS

Binary to Gray code and Gray code to Binary Converter

Problem:

1. Convert the binary number 0101 to Gray code with XOR gates
2. Convert the gray code 1011 to binary with XOR gates

Solution:



CODE CONVERTERS

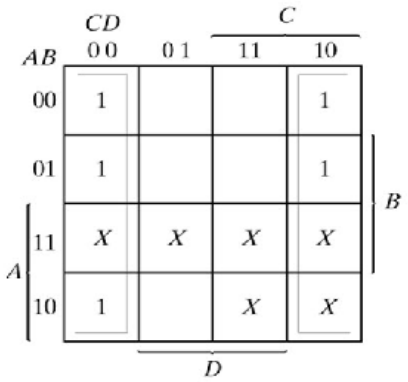
Design of BCD to XS3 Code Converter

TRUTH TABLE

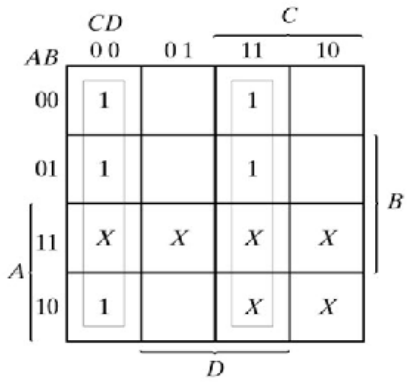
INPUTS (STANDARD BCD CODE)				OUTPUTS (XS3 CODE)			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

CODE CONVERTERS

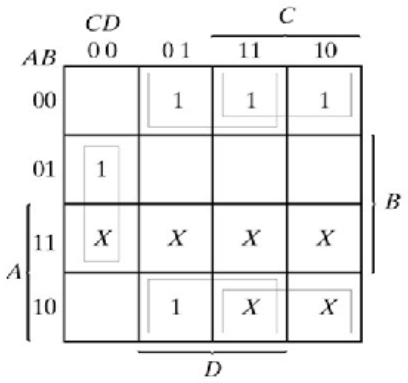
Design of BCD to XS3 Code Converter



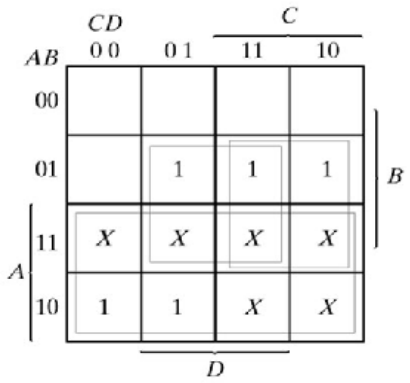
$z = D'$



$y = CD + C'D'$



$X = B'C + B'D + BC'D'$



$w = A + BC + BD$

K-maps for simplification and simplified boolean expressions

CODE CONVERTERS

Design of BCD to XS3 Code Converter

- After the manipulation of the boolean expressions for using common gates for two or more outputs, logic expressions can be given by

z=D′

y=CD+C′D′ = CD + (C+D)′

x= B′C + B′D + BC′D′ = B′(C+D) + BC′D′

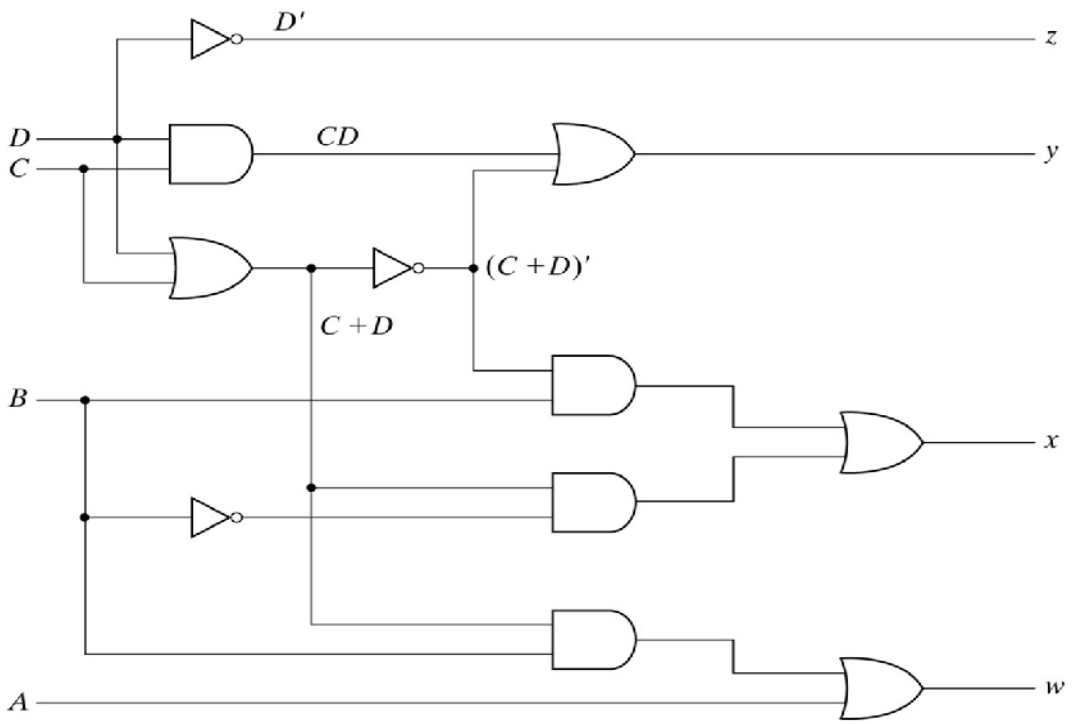
= B′(C+D) + B (C+D)′

w= A + BC + BD = A + B (C+D)

CODE CONVERTERS

Design of BCD to XS3 Code Converter

Logic Diagram



Parity Generators/Checkers

* Construction of 3-bit parity generator and 4-bit parity checker circuits

PARITY GENERATORS/CHECKERS
-Introduction

- Errors can occur as digital codes are being transferred from one point to another within a digital system or while codes are being transmitted from one system to another.
- The errors take the form of undesired changes in the bits that make up the coded information; that is a 1 can change to a 0, or a 0 to a 1 because of component malfunctions or electrical noise.

- A parity bit is used for the purpose of detecting errors during the transmission of binary information
- It is an extra bit included with a binary message to make the number of 1's either odd or even
- If the number of 1s in a code is even – called even parity and if the number of 1's in a code is odd – called odd parity

PARITY GENERATORS/CHECKERS

-Introduction

- The message, including the parity bit is transmitted and then checked at the receiving end for errors
- An error is detected if the checked parity does not correspond with the one transmitted.

- The circuit that generates the parity bit in the transmitter is called a parity generator.
- The circuit that checks the parity in the receiver is called a parity checker.

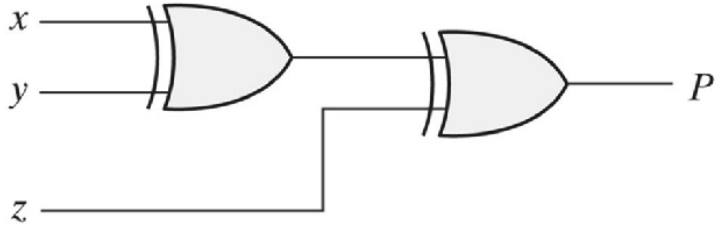
PARITY GENERATORS/CHECKERS

- Construction of 3- bit Even parity Generator

Truth Table:

Three-Bit Message			Parity Bit
x	*y*	*z*	*P*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Logic Diagram:



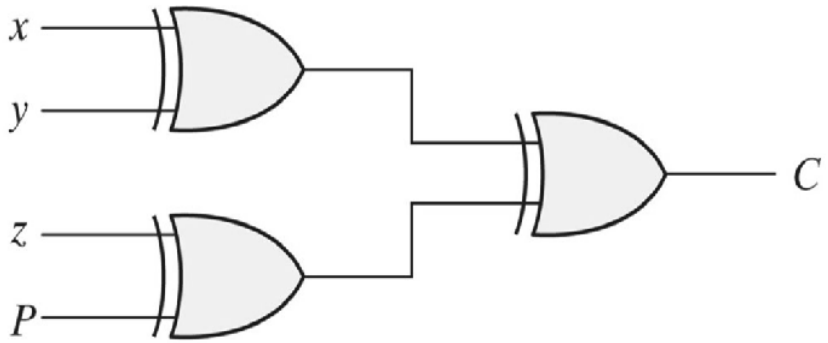
PARITY GENERATORS/CHECKERS

- Construction of 4- bit Even parity Checker

Truth Table:

Four Bits Received				Parity Error Check
x	*y*	*z*	*P*	*C*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Logic Diagram:



PARITY GENERATORS/CHECKERS

- Construction of 3- bit odd parity Generator and 4-bit odd parity Checker

Problem:

Construct 3-bit odd parity generator and 4-bit odd parity checker circuit along with the truth table.

References

Slides adopted from the books

1. Thomas L.Floyd, “Digital Fundamentals,” 11th Edition, Prentice Hall, 2014 (ISBN10:0132737965/ISBN13:9780132737968)

- M.Morris Mano and Michael D. Ciletti, " Digital Design," 5th Edition, Pearson Education International, 2012

- Ronald J.Tocci, Neal S.Widmer, and Gregory L.Moss, "Digital Systems- Principles and Application"- 11th Edition, Pearson Education International, 2011 (ISBN: 9780135103821)