Combinational Logic Building Blocks

Chapter 6

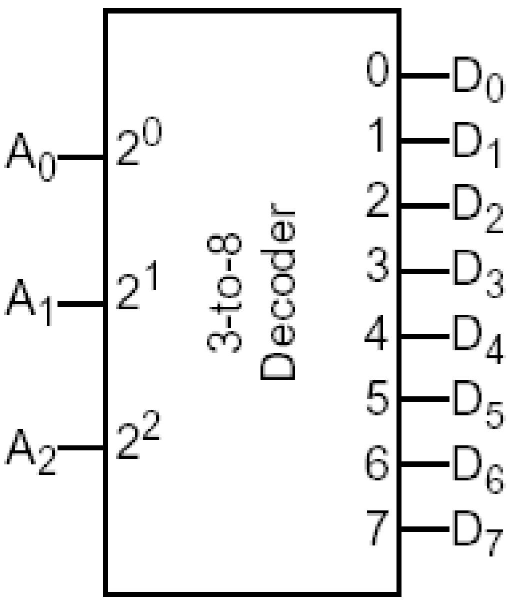
Combinational Logic
Introduction

- Combinational logic
  - Output at any time is determined completely by the current input.
    - We will later consider circuits where the output is determined by the input and the current state (memory) of the system.
  - In this chapter we will consider some useful building blocks that can be pieced together and used in larger designs. This will include:
    - Multiplexers (selectors) and demultiplexers (distributors)
    - Encoders, priority encoders, decoders
    - Adders (full and half)
    - Parity generators and parity checkers
    - Shifters and rotators
    - Comparators

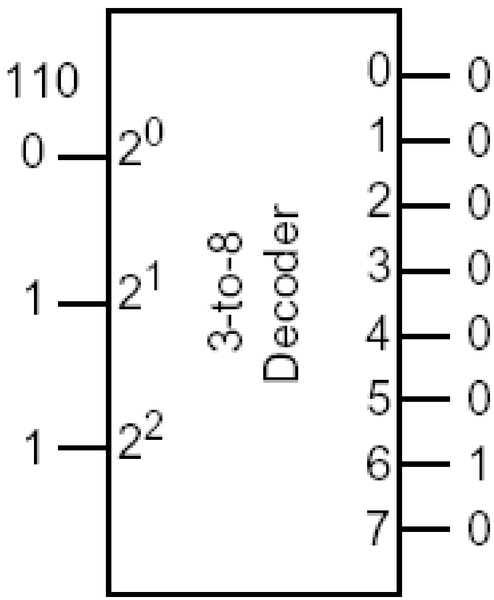
Decoders

Basic Decoder

- Standard decoder is an n-to-m-line decoder, where $m \leq 2^n$.
  - Example: 3-to-8-line decoder



Example:
Input of 110



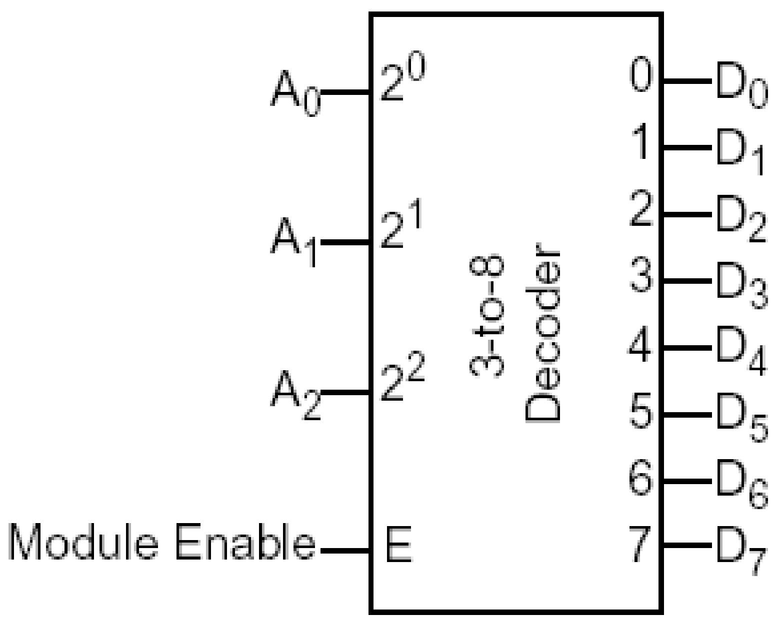
- All outputs D_m are low except for the one corresponding to the binary value of the input $A_n \ldots A_1 A_0$.

Decoders

Decoders With Enable

- Often, combinational logic building blocks will also have an enable line that turns on outputs or leaves them off.

3-to-8 Decoder
with Enable



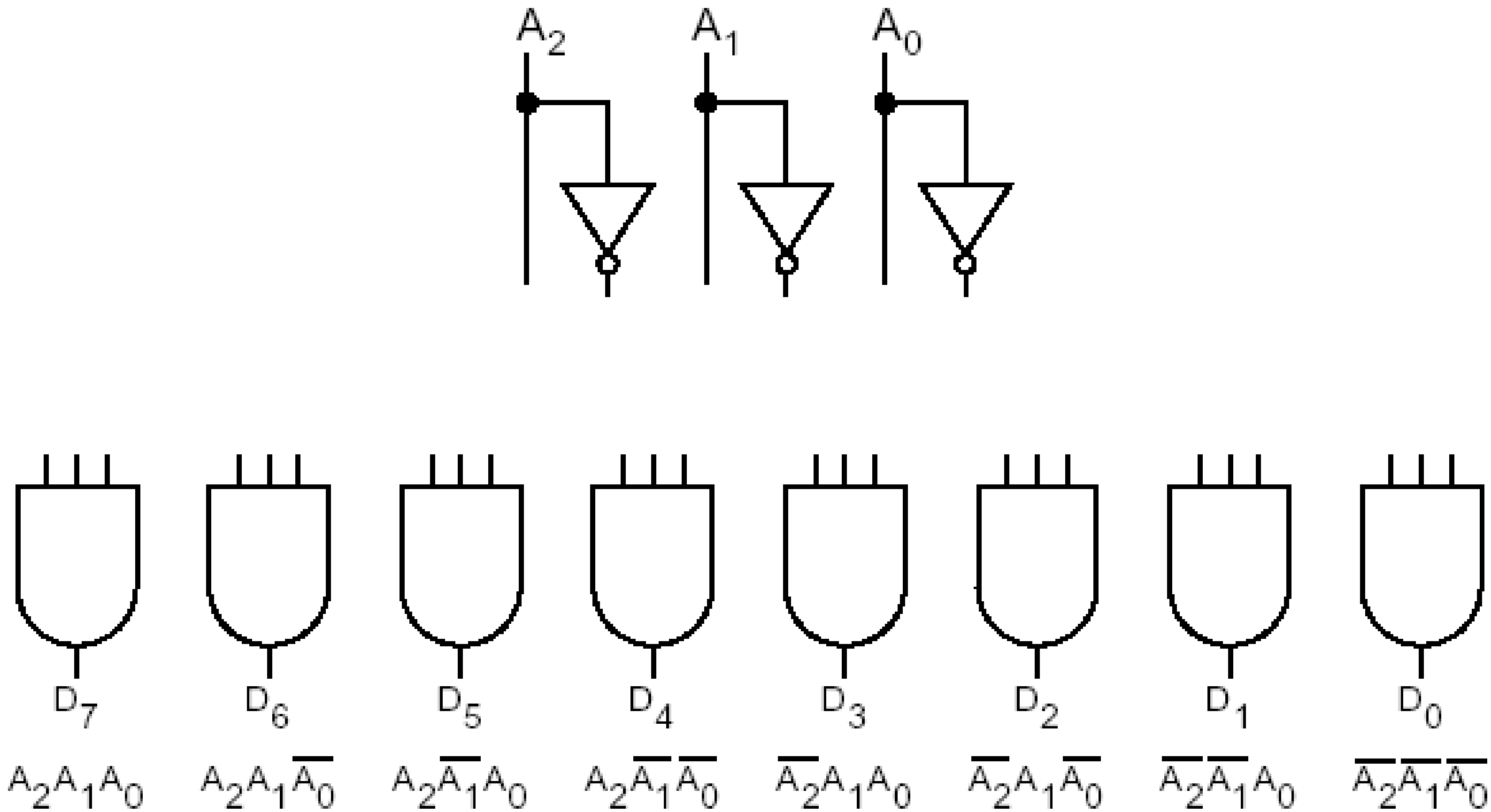
Decoders
Truth Tables

- Truth table for a **3-to-8-line decoder**:

Inputs				Outputs							
A_2	A_1	A_0	E	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
X	**X**	**X**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
0	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
1	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
1	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
1	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
1	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Decoders
Implementation

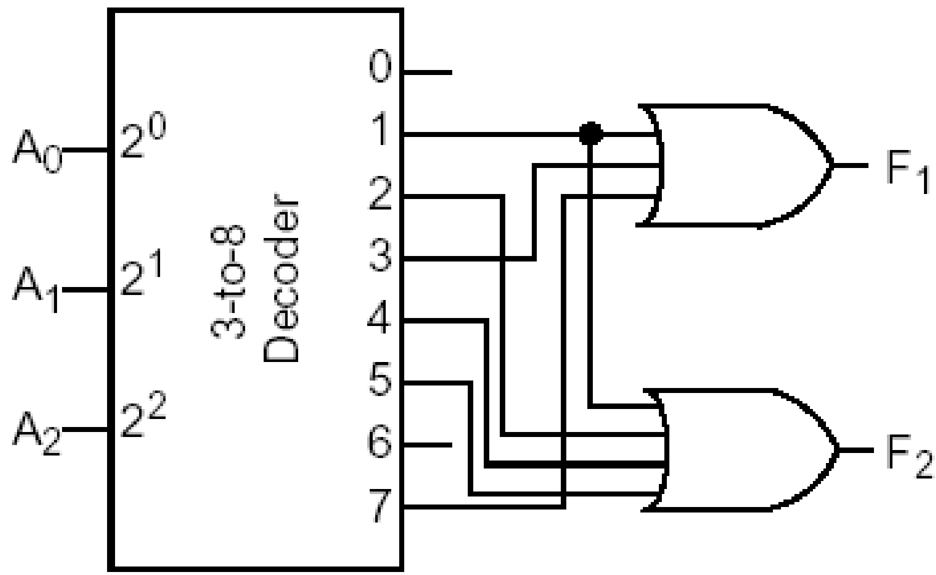
- How can a decoder be implemented? Fill in the circuit!



Decoders
Designing with Decoders

•Any Boolean function can be implemented using a **decoder** and **OR** gates by ORing together the function's **minterms**.

Inputs			Outputs	
A_2	A_1	A_0	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0



Decoders

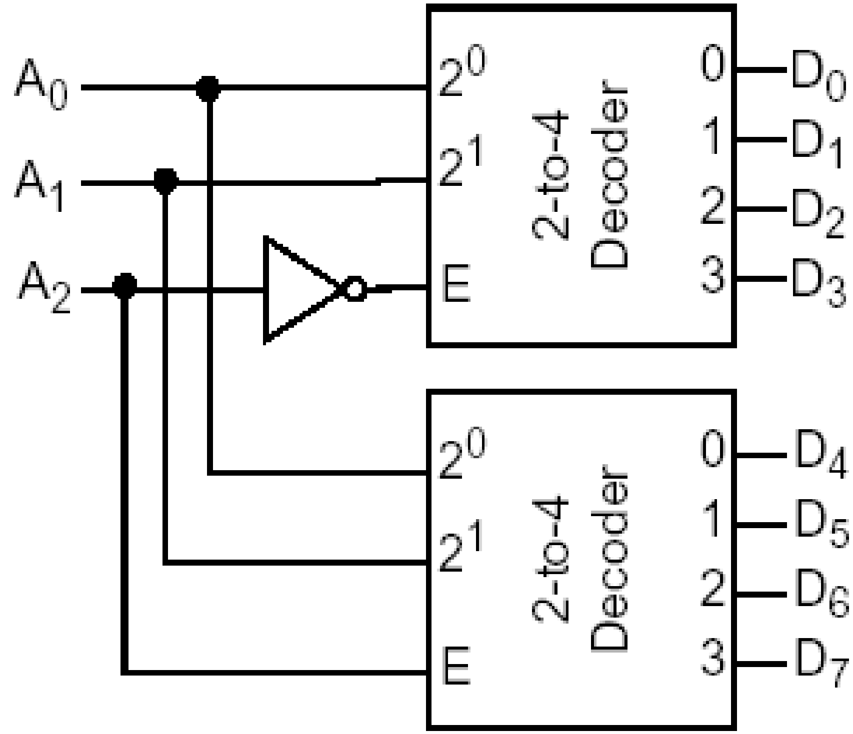
Decoder Networks

- We can also use multiple decoders to form a larger decoder.

A_2 used with enable input to control which decoder will output the **1**.

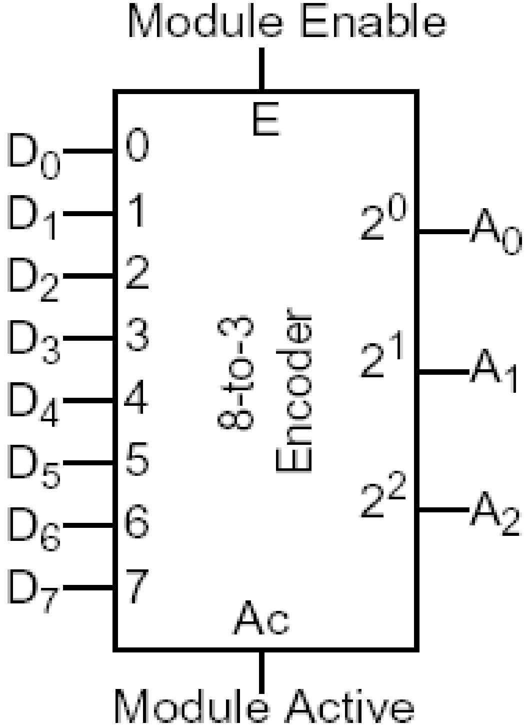
A_1 and **A_0** used to select which output on specific decoder will output **1**.

3-to-8 Decoder Implemented with two 2-to-4 Decoders

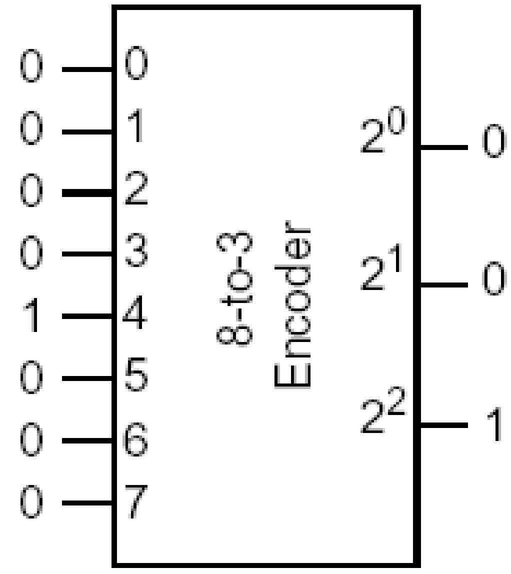


Encoders
Basic Encoder

- Standard binary encoder is an m-to-n-line encoder, where $m \le 2^n$.
  - Example: 8-to-3-line encoder



Example:
Input of 00010000



Encoders

Encoder Truth Table

- Truth table for an **8-to-3-line encoder**:

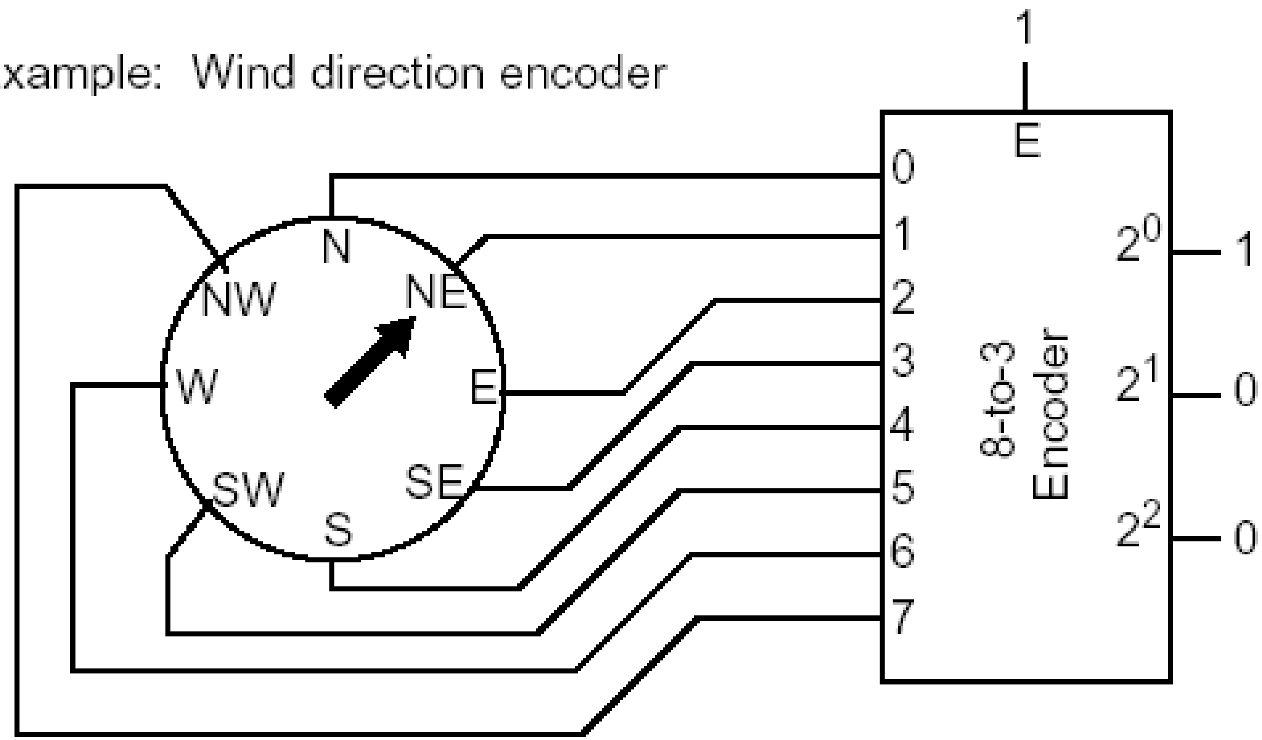
Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**
0	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**
0	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**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**

- Assumed that only one input is **1**. What happens if more then one is **1**?

Encoders
Designing with Encoders

- Encoders are useful when the occurrence of one of several disjoint events needs to be represented by an integer identifying the event.

Example: Wind direction encoder



Encoders
Priority Encoders

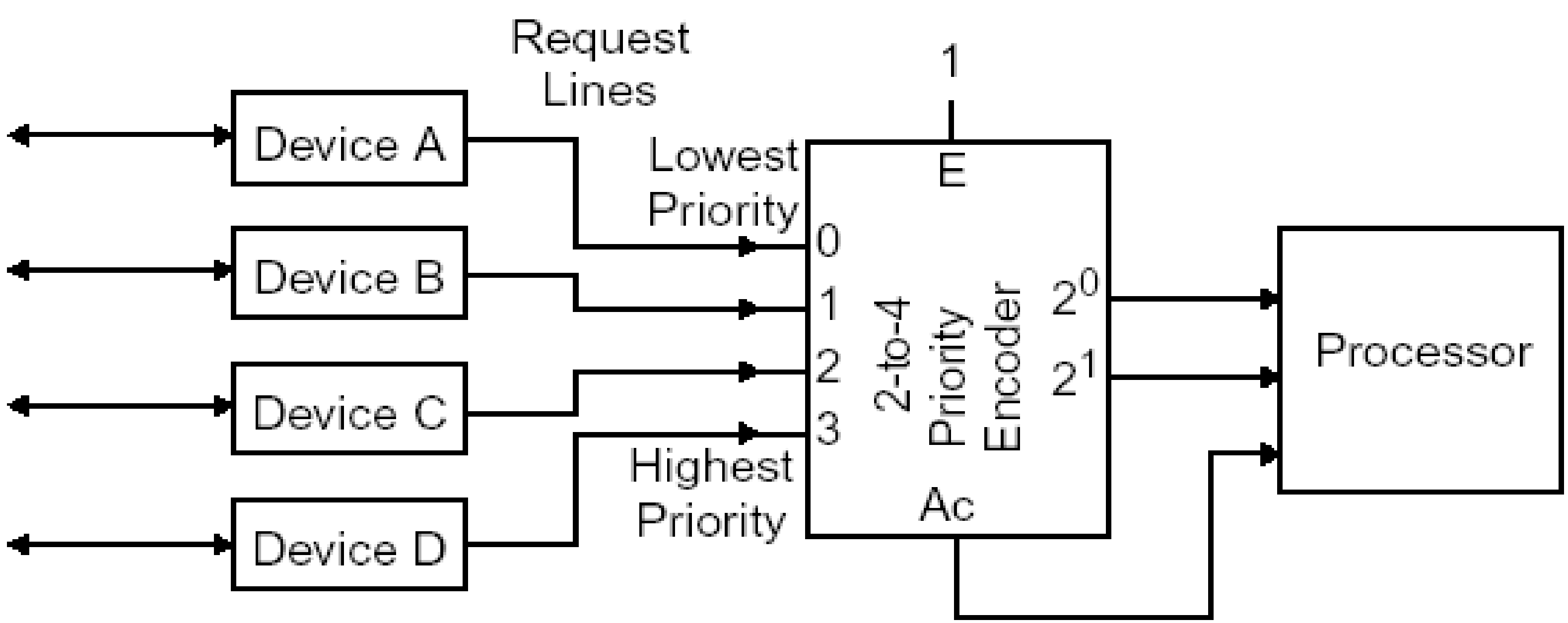
- A priority encoder takes the input of 1 with the highest index and translates that index to the output.

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**	**X**	**0**	**0**	**1**
0	**0**	**0**	**0**	**0**	**1**	**X**	**X**	**0**	**1**	**0**
0	**0**	**0**	**0**	**1**	**X**	**X**	**X**	**0**	**1**	**1**
0	**0**	**0**	**1**	**X**	**X**	**X**	**X**	**1**	**0**	**0**
0	**0**	**1**	**X**	**X**	**X**	**X**	**X**	**1**	**0**	**1**
0	**1**	**X**	**X**	**X**	**X**	**X**	**X**	**1**	**1**	**0**
1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**1**	**1**	**1**

Encoders
Design With P-Encoder

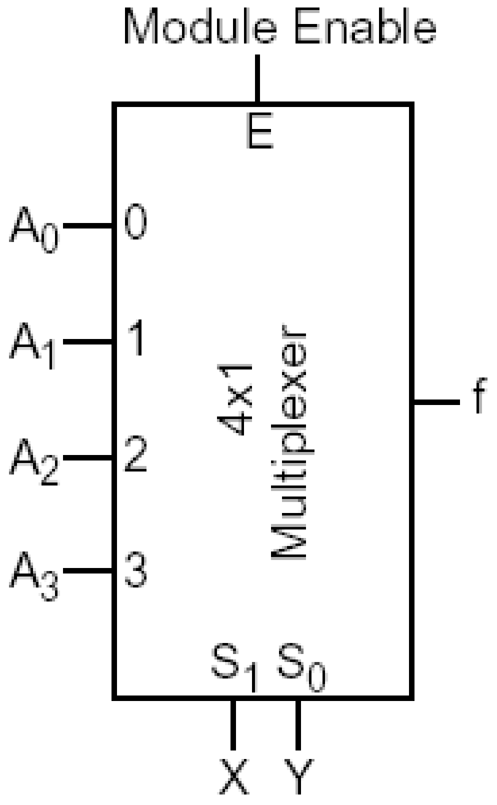
- Priority encoders are useful when inputs have a predefined priority and we wish to select the input with the highest priority.

Example: Resolving interrupt requests



Multiplexers
Basic Multiplexer (MUX)

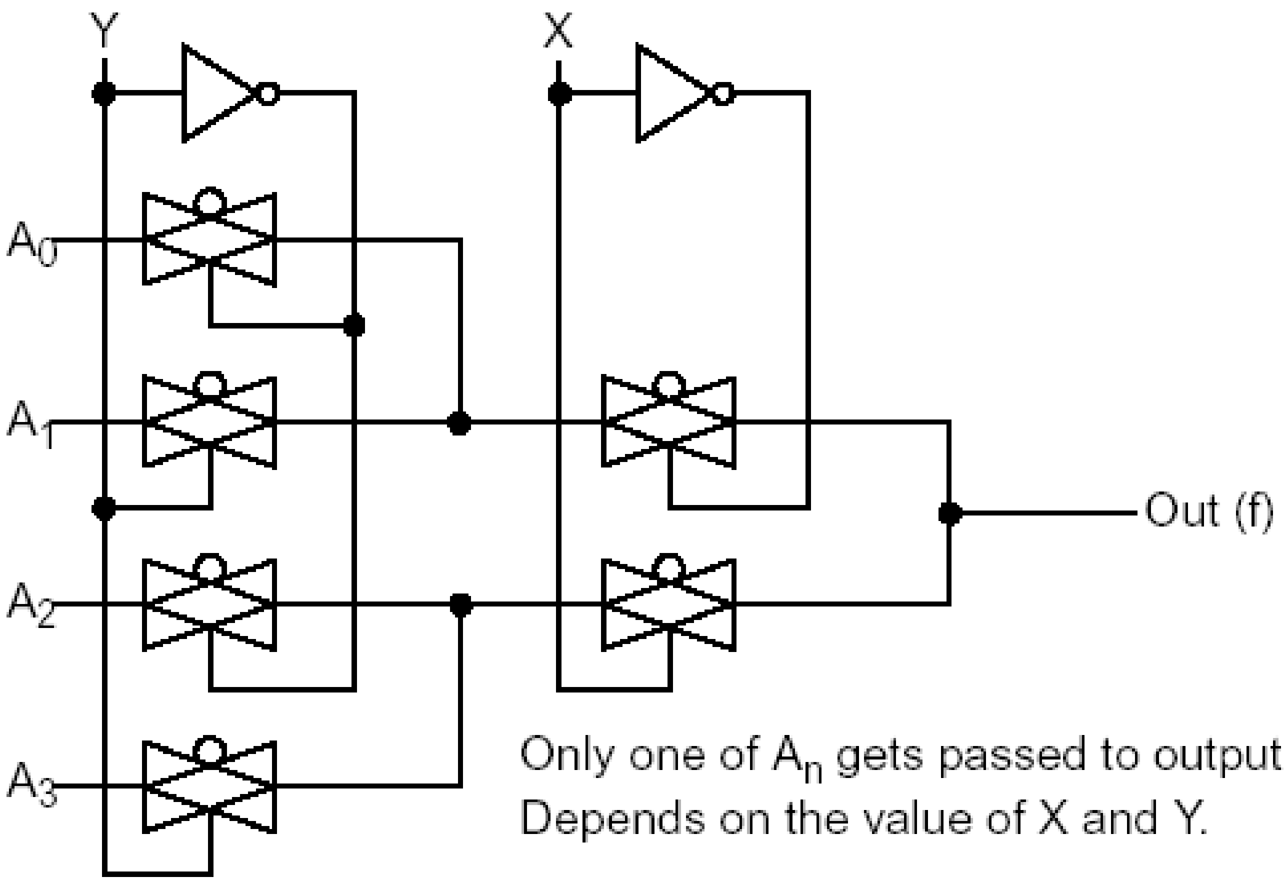
- Selects one of many inputs to be directed to an output.



		Inputs				Output
X	Y	A_3	A_2	A_1	A_0	F
0	**0**	**X**	**X**	**X**	**0**	A_0=**0**
0	**1**	**X**	**X**	**0**	**X**	A_1=**0**
1	**0**	**X**	**0**	**X**	**X**	A_2=**0**
1	**1**	**0**	**X**	**X**	**X**	A_3=**0**
0	**0**	**X**	**X**	**X**	**1**	A_0=**1**
0	**1**	**X**	**X**	**1**	**X**	A_1=**1**
1	**0**	**X**	**1**	**X**	**X**	A_2=**1**
1	**1**	**1**	**X**	**X**	**X**	A_3=**1**

Multiplexers
Using Pass Gates

- The **4x1 mux** can be implemented with **pass gates** as follows.

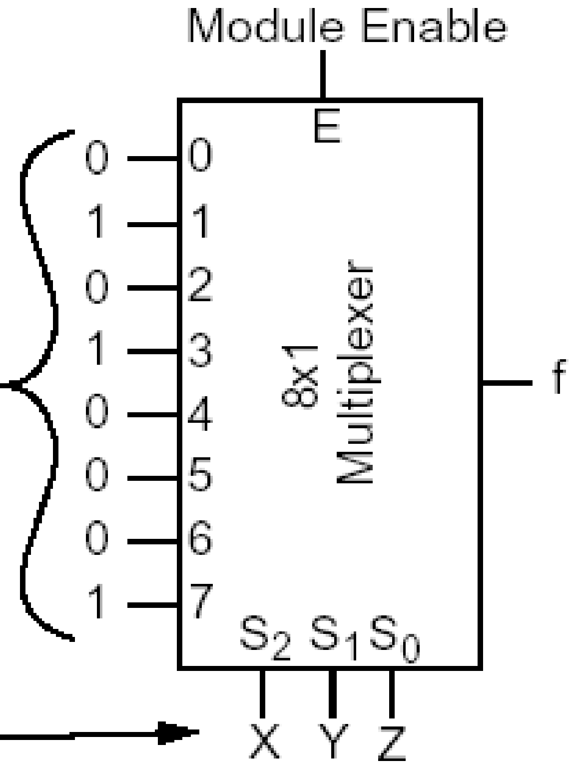


Multiplexers
Design with Multiplexers

- Any Boolean function can be implemented by setting the inputs corresponding to the function and the selectors as the variables.

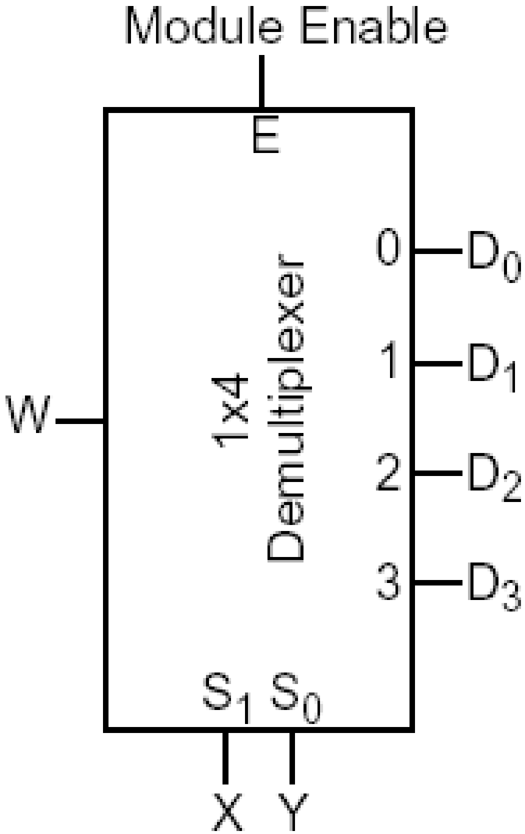
Example:

X	Y	Z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



Demultiplexers
Basic Demultiplexer

- Takes one input and selects one of many outputs to direct the input.



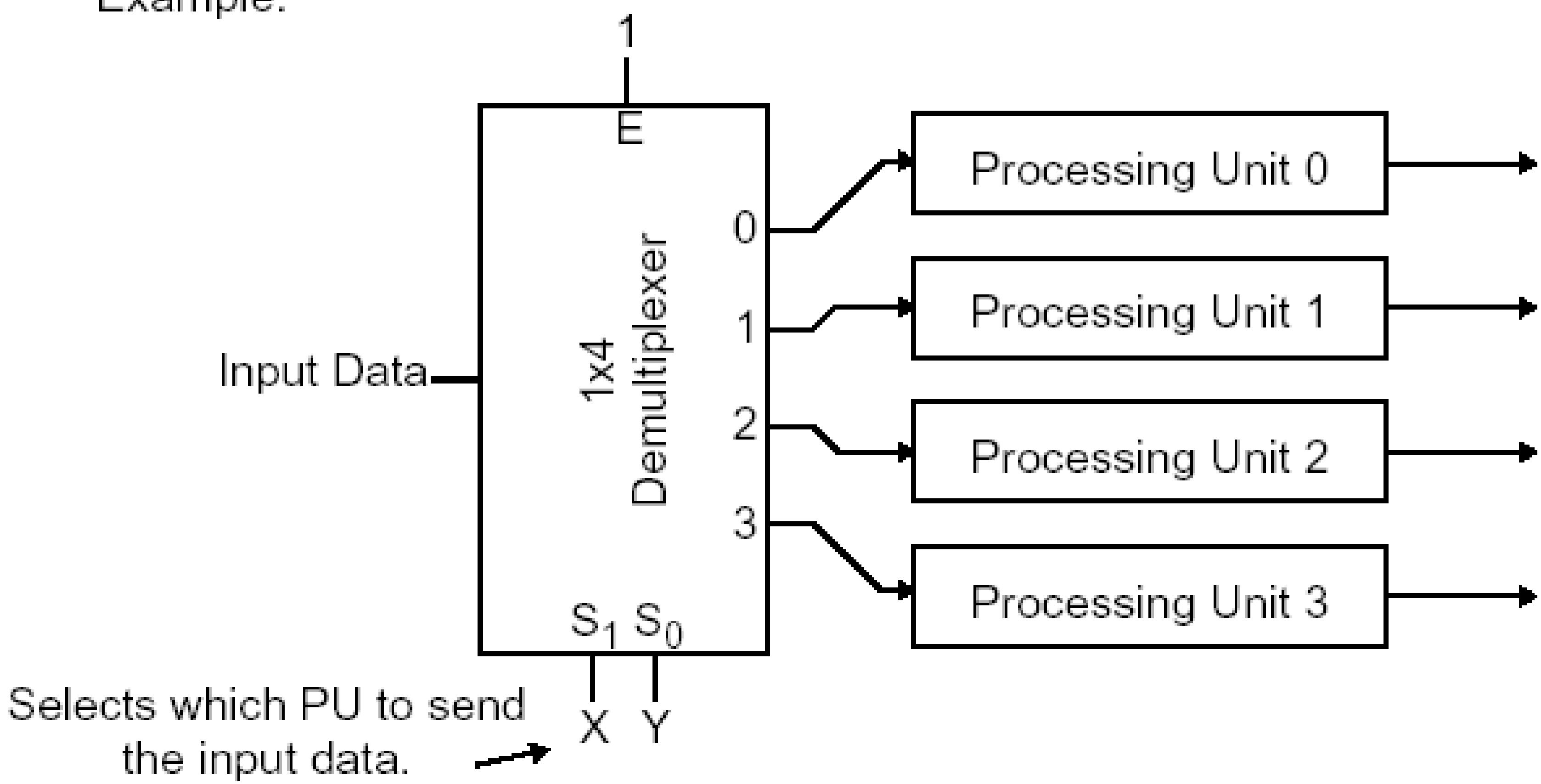
Inputs			Outputs			
X	Y	W	D_3	D_2	D_1	D_0
0	**0**	**0**	**0**	**0**	**0**	W=**0**
0	**1**	**0**	**0**	**0**	W=**0**	**0**
1	**0**	**0**	**0**	W=**0**	**0**	**0**
1	**1**	**0**	W=**0**	**0**	**0**	**0**
0	**0**	**1**	**0**	**0**	**0**	W=**1**
0	**1**	**1**	**0**	**0**	W=**1**	**0**
1	**0**	**1**	**0**	W=**1**	**0**	**0**
1	**1**	**1**	W=**1**	**0**	**0**	**0**

Demultiplexers

Design W/ Demultiplexers

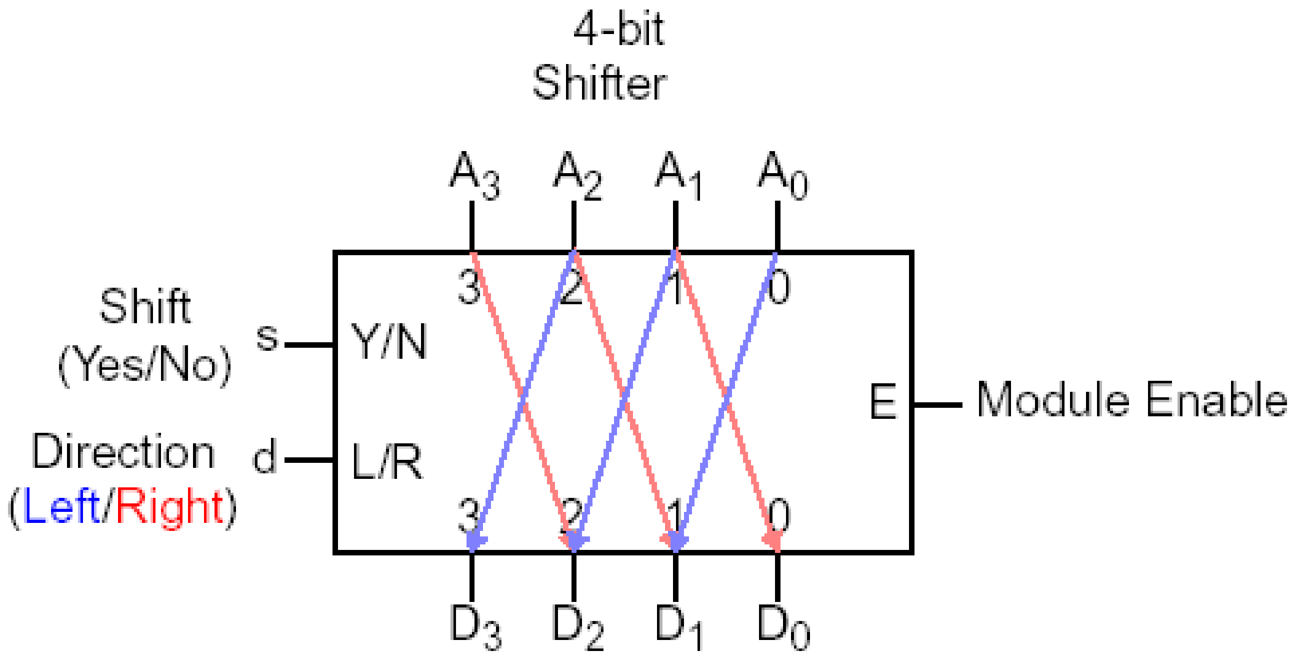
- A demultiplexer is useful for routing an input to a desired location.

  Example:



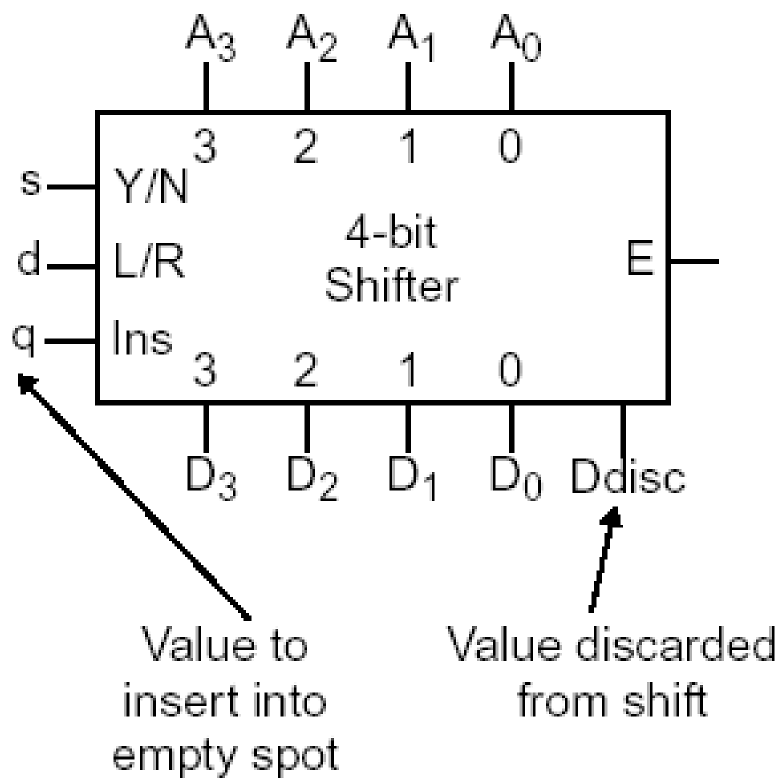
Shifters
Basic Shifter

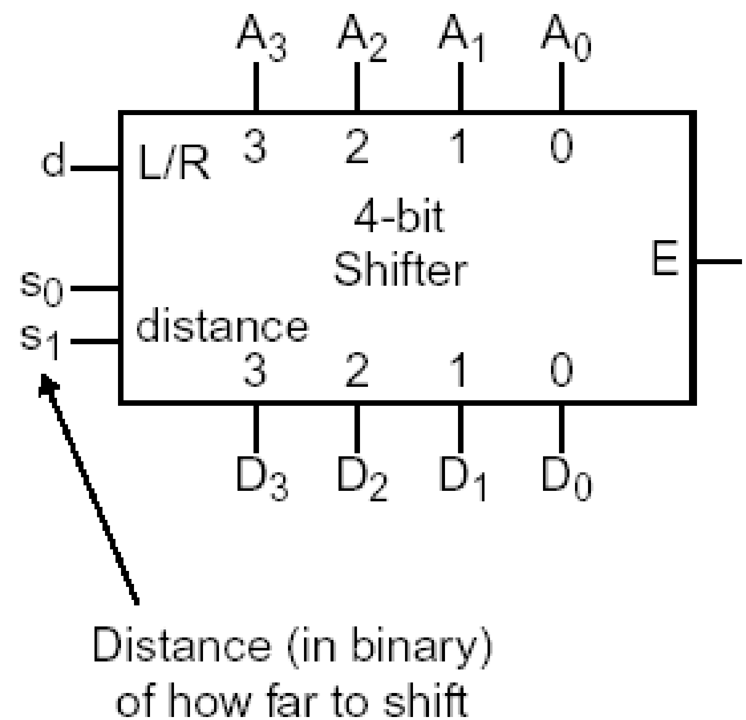
- A shifter takes a set of inputs and shifts it to the right or left.



Shifters
Common Shifters

- A whole variety of other shifters are possible such as

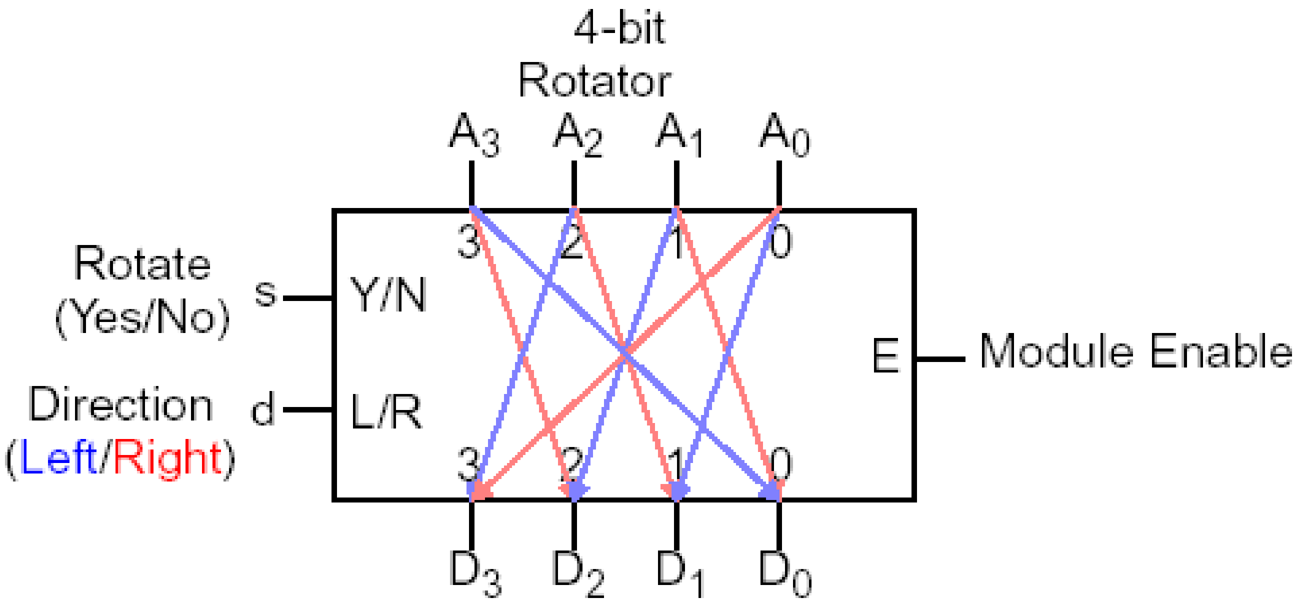




Rotators

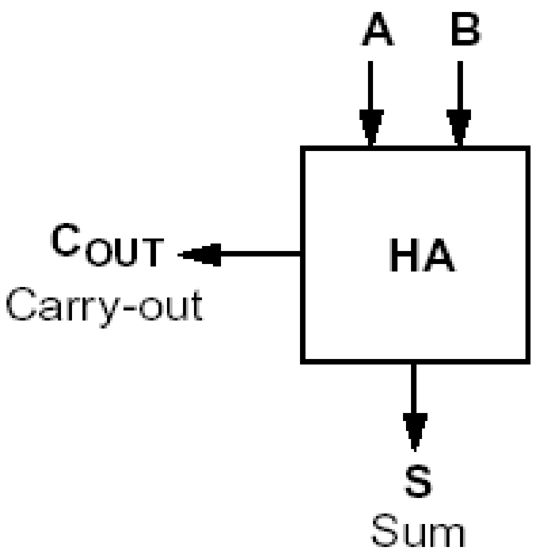
Basic Rotator

- A rotator takes a set of inputs and rotates it to the right or left.
  - This is similar to a shifter except that instead of dropping the bit off of the end in the shifting, the normally discarded bit is taken and moved to fill the empty spot on the end.

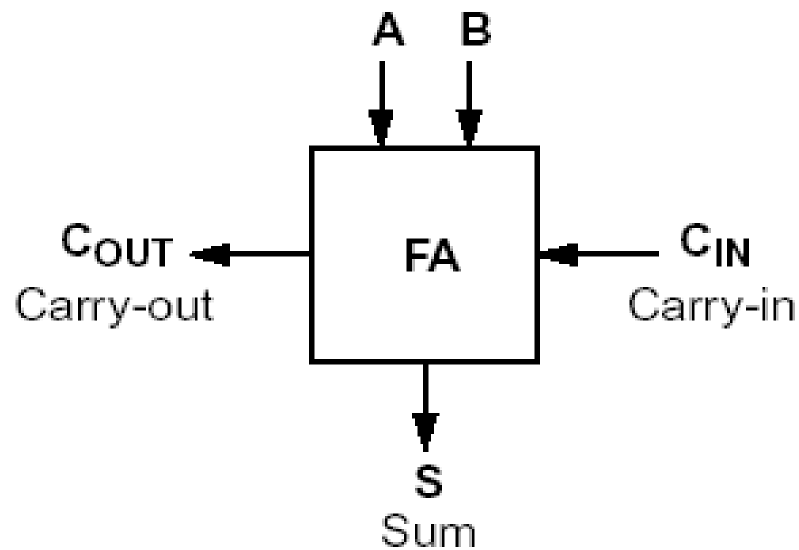


Adders
Half and Full Adder

- Two basic building blocks for arithmetic are half- and full-adders as depicted by the block diagrams below.



Half-adder



Full-adder

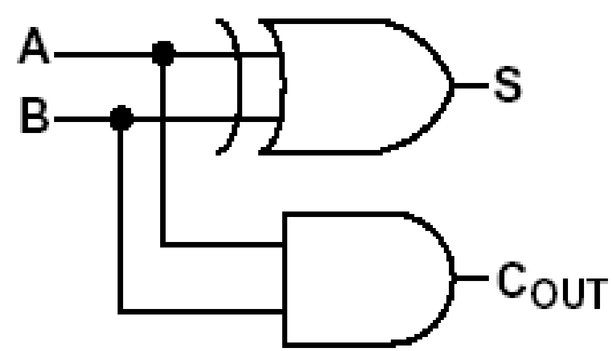
Adders
Half-Adder (HA)

- First of all, how do we add?
- 2's complement arithmetic allows us to add numbers normally.

Inputs A	Inputs B	Sum S	Carry-out C_{OUT}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

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

$$C_{OUT} = AB$$



Adders

Full-Adder (FA) (1)

- Half-adder missed a possible carry-in. A full-adder (FA) includes this additional carry-in.

Inputs		Carry-in	Sum	Carry-out
A	B	C_{IN}	S	C_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = (A \oplus B) \oplus C_{IN}$$

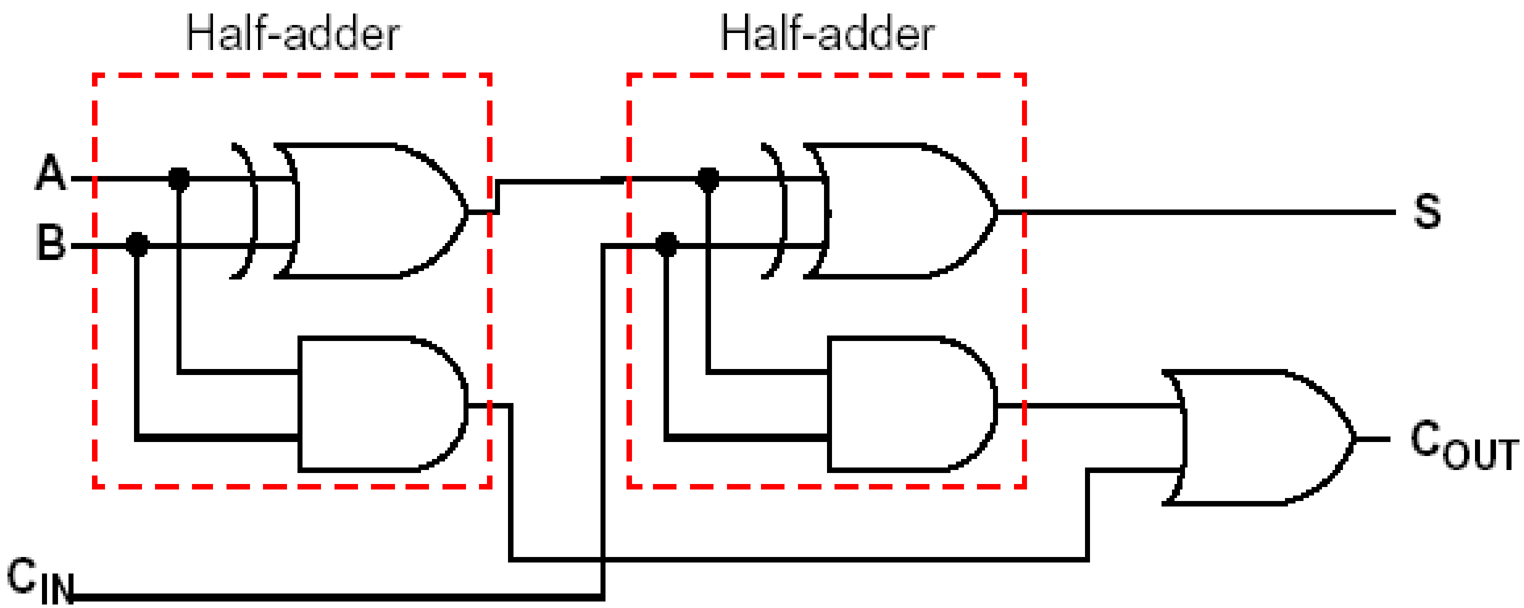
$$C_{OUT} = AB + C_{IN}(A \oplus B)$$

Adders

Full-Adder (FA) (2)

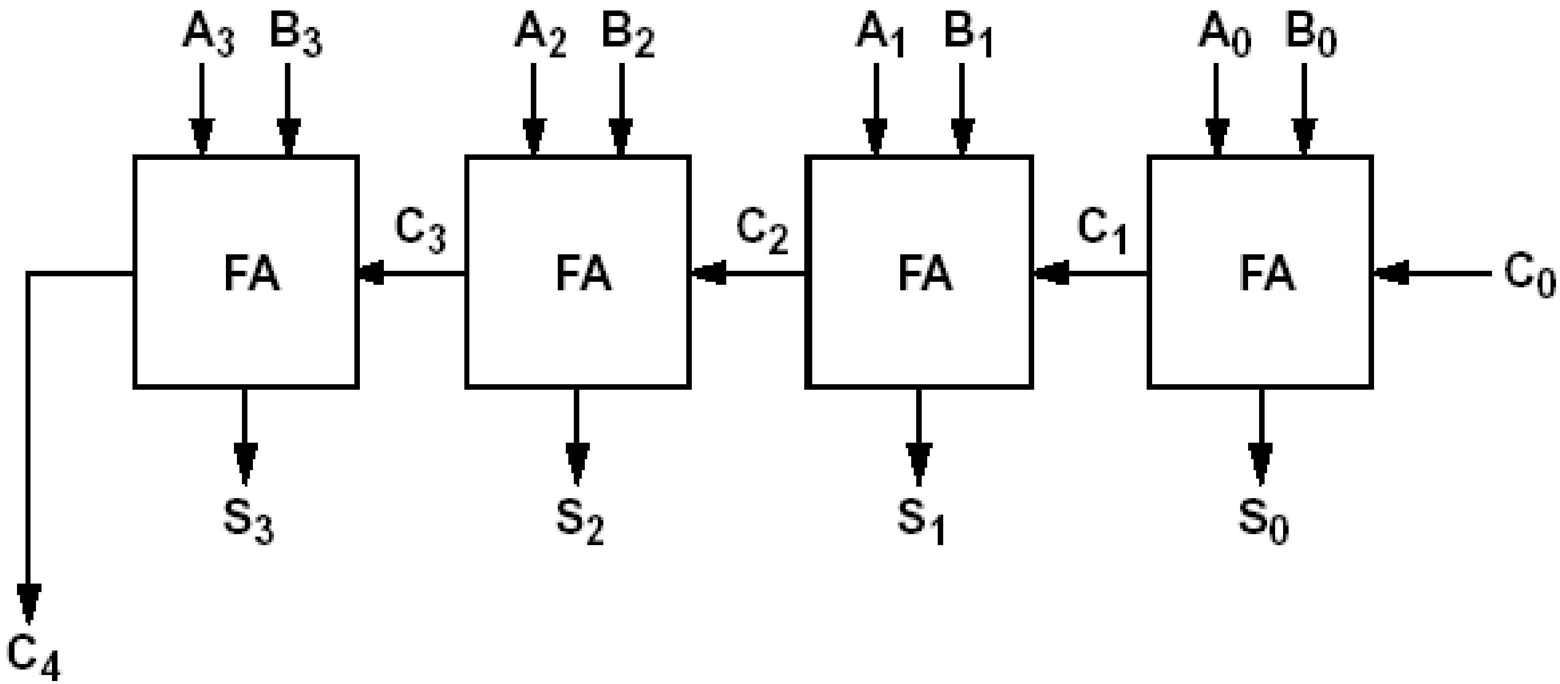
$$S = (A \oplus B) \oplus C_{IN}$$

$$C_{OUT} = AB + C_{IN}(A \oplus B)$$



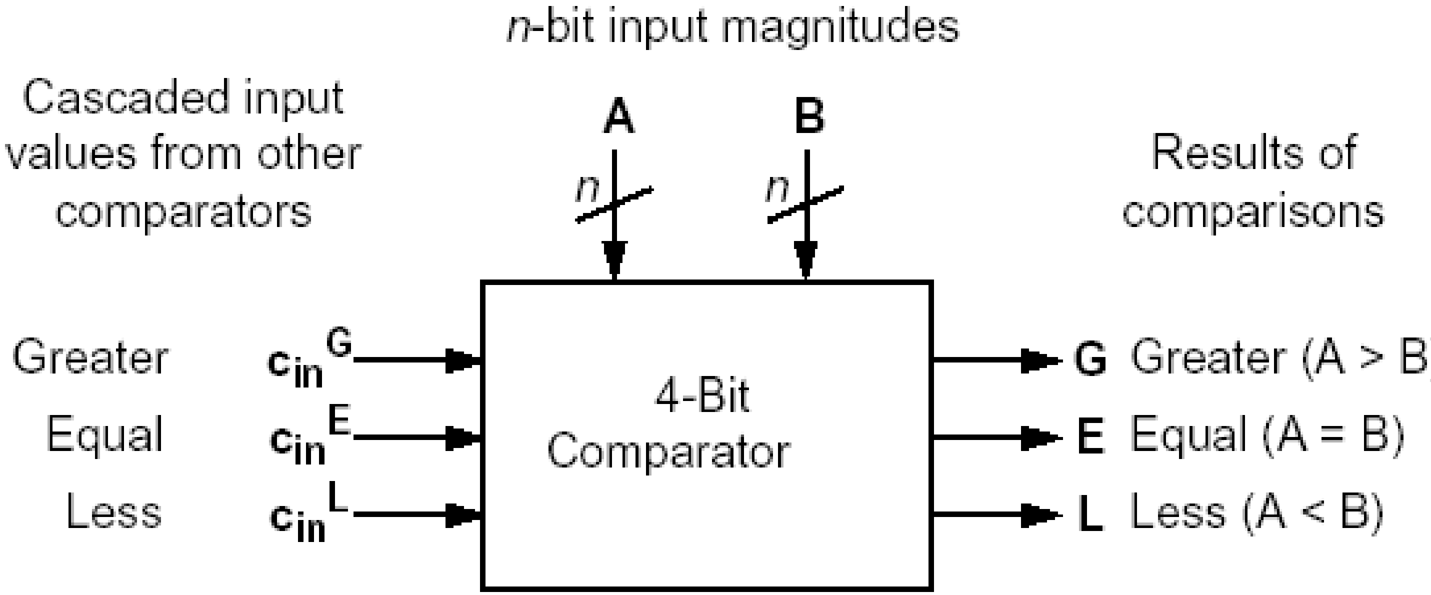
Adders
Binary Addition

- A 4-bit binary adder can be formed with four full-adders as follows.



Comparators
Magnitude Comparator

- Given two *n*-bit magnitudes, A and B, a comparator indicates whether
  - A = B, A > B, or A < B



Comparators
Magnitude Comparator

- The approach is to use the XNOR function (equivalence) on each of the *n*-bits as follows

$$x_i = \mathbf{A}_i\mathbf{B}_i + \overline{\mathbf{A}}_i\overline{\mathbf{B}}_i = \overline{\mathbf{A}_i \oplus \mathbf{B}_i}$$

- The Boolean functions for a 4-bit magnitude comparator is as follows
  - $(\mathbf{A} = \mathbf{B}) = x_3x_2x_1x_0$
  - $(\mathbf{A} > \mathbf{B}) = \mathbf{A}_3\overline{\mathbf{B}}_3 + x_3\mathbf{A}_2\overline{\mathbf{B}}_2 + x_3x_2\mathbf{A}_1\overline{\mathbf{B}}_1 + x_3x_2x_1\mathbf{A}_0\overline{\mathbf{B}}_0$
  - $(\mathbf{A} < \mathbf{B}) = \overline{\mathbf{A}}_3\mathbf{B}_3 + x_3\overline{\mathbf{A}}_2\mathbf{B}_2 + x_3x_2\overline{\mathbf{A}}_1\mathbf{B}_1 + x_3x_2x_1\overline{\mathbf{A}}_0\mathbf{B}_0$

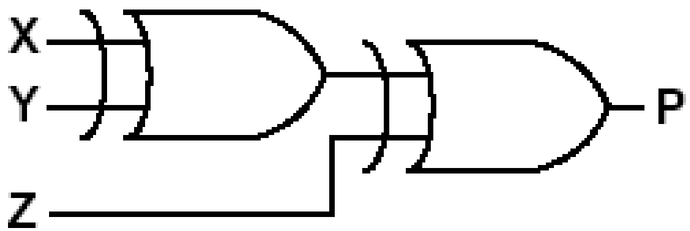
Note: $\mathbf{A}_i\overline{\mathbf{B}}_i$ indicates whether $\mathbf{A}_i > \mathbf{B}_i$, $\overline{\mathbf{A}}_i\mathbf{B}_i$ indicates whether $\mathbf{A}_i < \mathbf{B}_i$, and x_i indicates whether $\mathbf{A}_i = \mathbf{B}_i$.

Building Blocks
Parity Generators

- A parity generator/checker can detect a 1-bit error in a message.

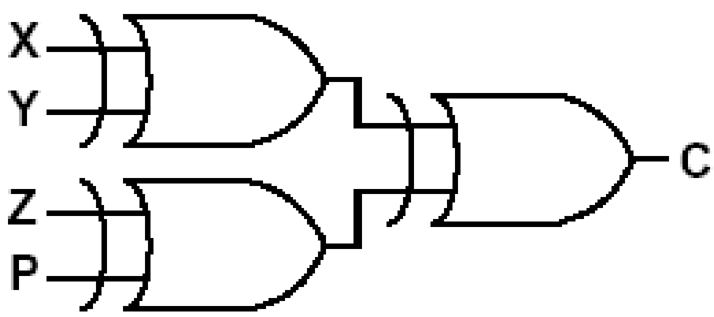
To generate an even parity bit

$P = X \oplus Y \oplus Z$



To check a even parity bit

$C = X \oplus Y \oplus Z \oplus P$



Message X	Y	Z	Even Parity Bit, **P**	C
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	0

If no errors detected, **C = 0**