ECE 2050 Digital Logic and Systems

Chapter 6 : Combinational Building Blocks

Instructor: Yue ZHENG, Ph.D.

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

Last Week

- Basic Combinational Logic Circuits
  - AND-OR Logic
  - AND-OR-Invert Logic
  - Exclusive-OR Logic
  - Exclusive-NOR Logic
- The Universal Property of NAND Gates
  - Combinations of NAND gates can function as NOT, AND, OR, and NOR
  - Combinations of NOR gates can function as NOT, AND, OR, and NAND.
- Dual Symbols :

$$\overline{AB} = \bar{A} + \bar{B} \qquad \overline{A + B} = \bar{A}\bar{B}$$

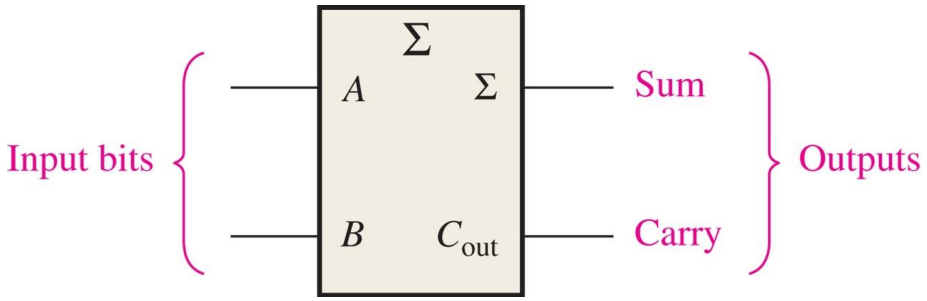
  - NAND Logic Diagrams Using Dual Symbols : NAND and Negative-OR;
  - NOR Logic Diagrams Using Dual Symbols : NOR and Negative-AND

Adders

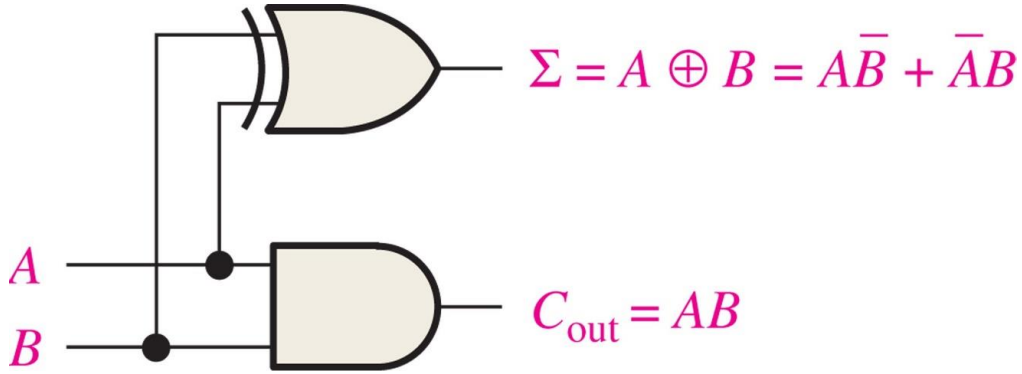
Half Adders

- Input: two binary digits
- Output: two binary digits — a sum bit & a carry bit

Logic symbol



Logic diagram



Half-adder truth table.

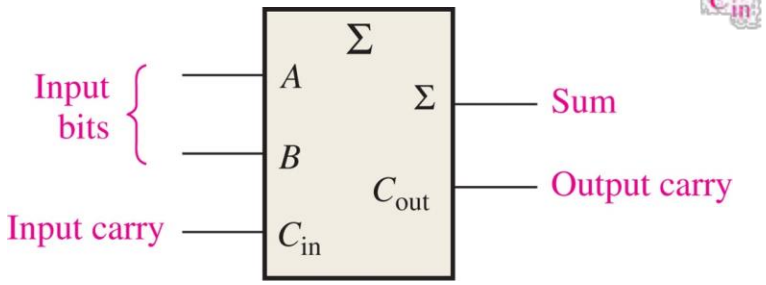
A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Σ = sum
C_{out} = output carry
A and B = input variables (operands)

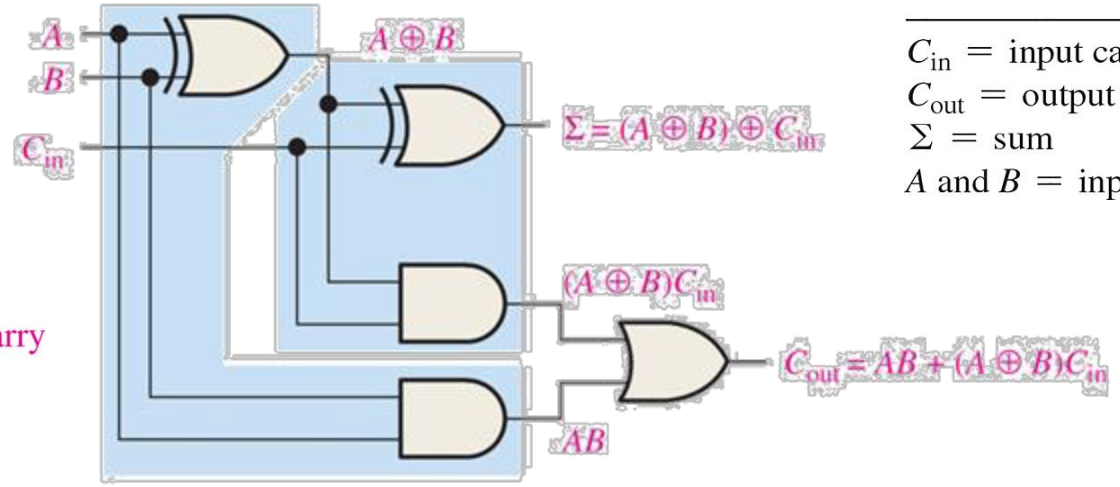
Full Adders

- Input: two binary digits & an input carry
- Output: two binary digits — a sum bit & a carry bit.

Logic symbol



Logic diagram



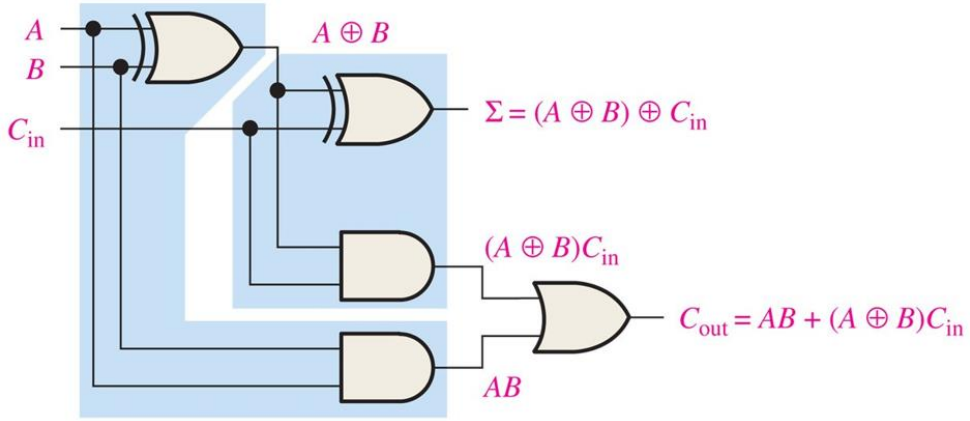
Two half-adders

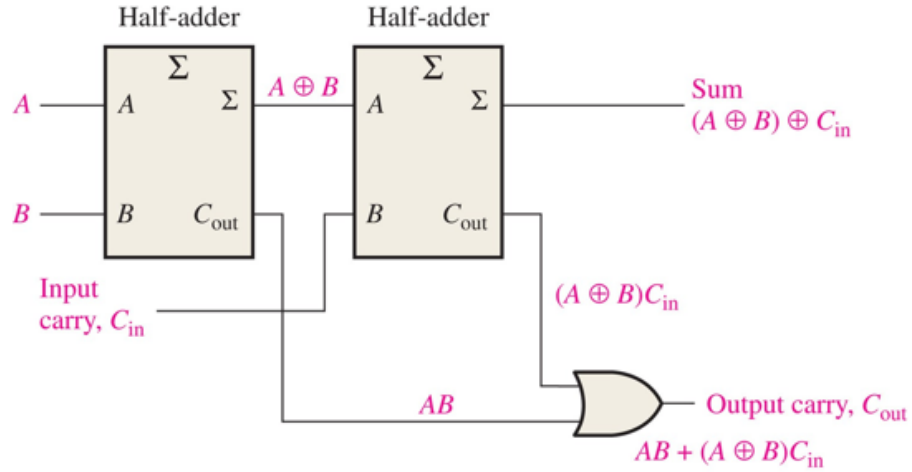
Full-adder truth table.

A	B	C_{in}	C_{out}	Σ
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

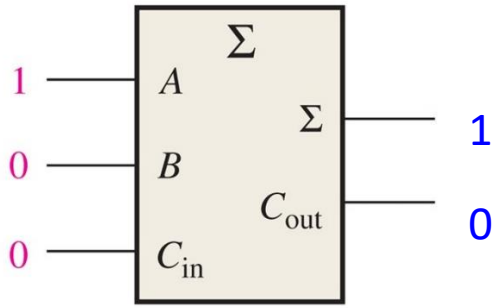
C_{in} = input carry, sometimes designated as CI
C_{out} = output carry, sometimes designated as CO
Σ = sum
A and B = input variables (operands)

Full-Adder Implemented with Half-adders.

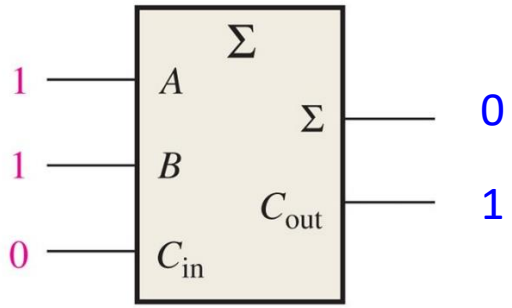




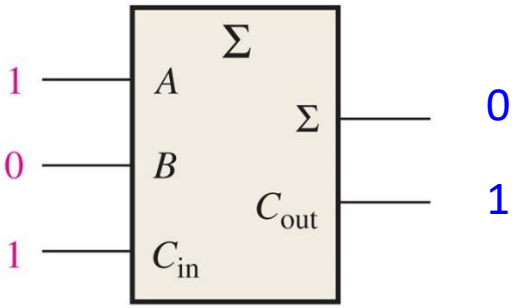
Examples



(a)



(b)

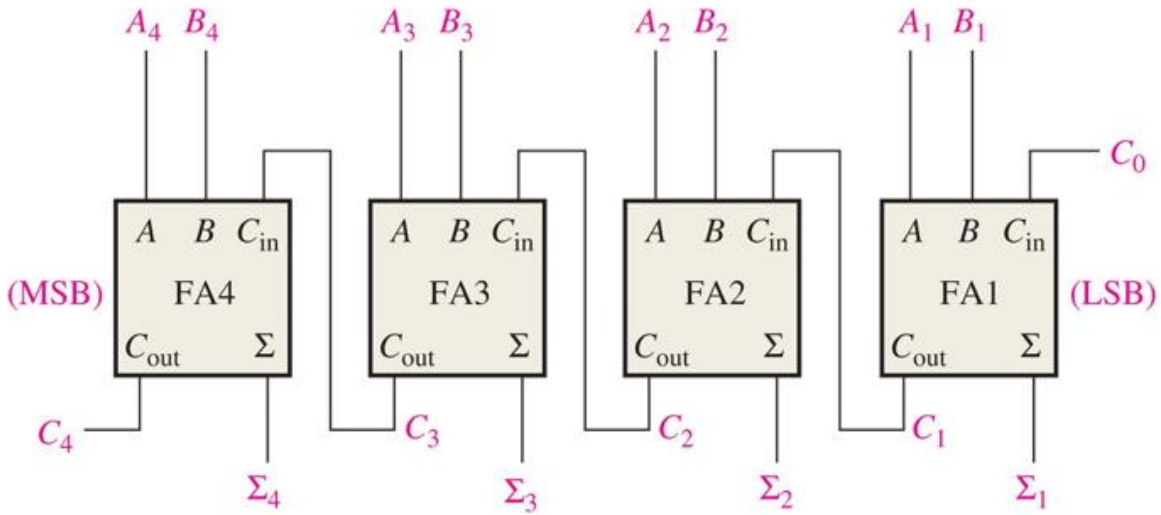


(c)

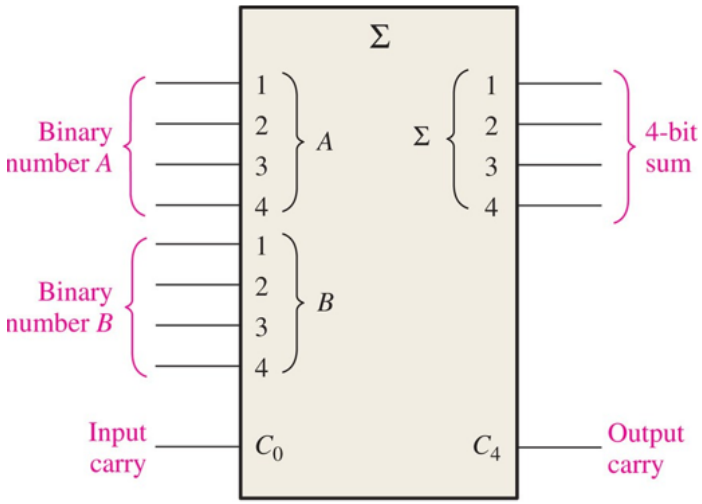
Parallel Binary Adders

- Two or more full-adders are connected to form parallel binary adders.
- Example:

$A_4A_3A_2A_1 = 1100$ and $B_4B_3B_2B_1 = 1100$



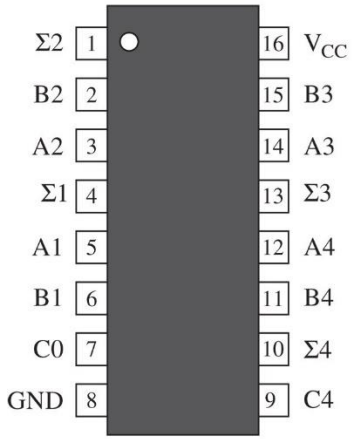
Block diagram



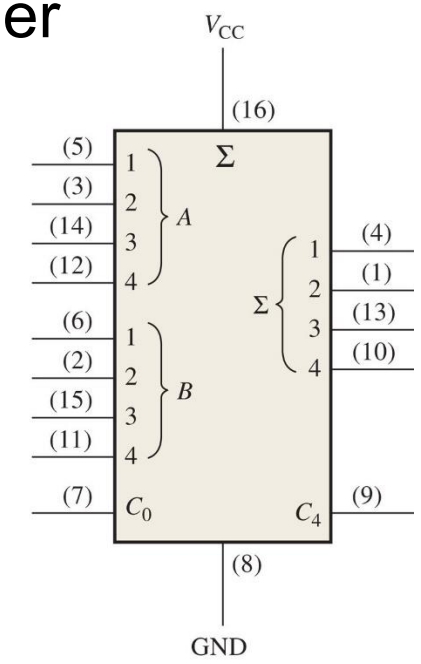
Logic symbol

Parallel Adders : 74HC283 / 74LS283

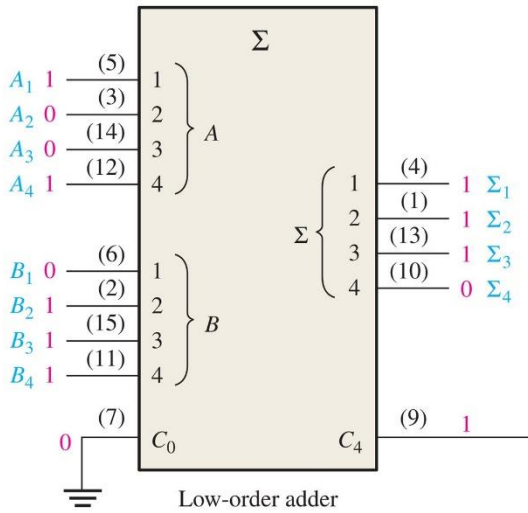
- 74HC283/74LS283: 4-bit parallel adders
- Cascading two 4-bit parallel adders to handle the addition of two 8-bit numbers
- The carry input of the low-order adder is connected to ground
- The carry output of the low-order adder is connected to the carry input of the high-order adder



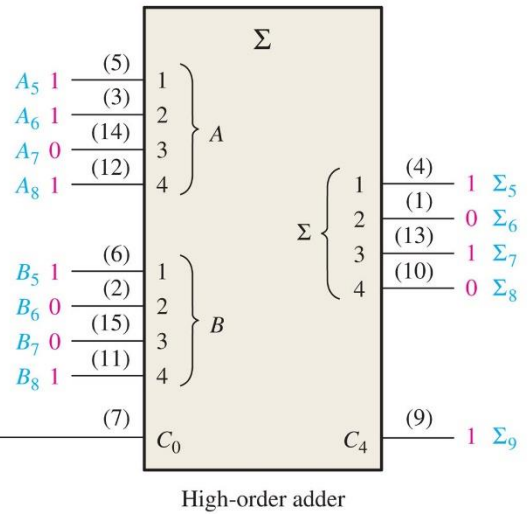
(a) Pin diagram



(b) Logic symbol



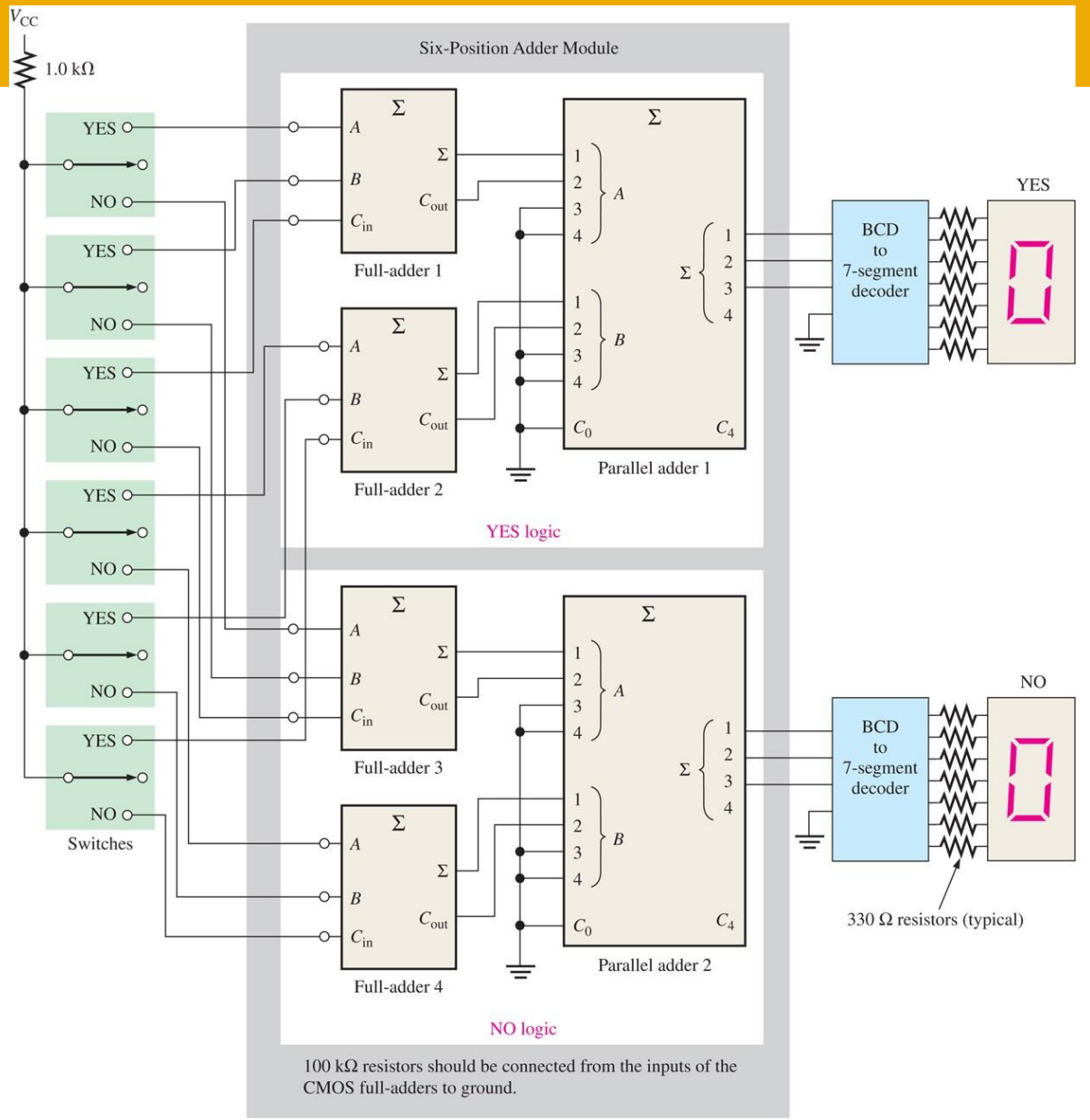
Low-order adder



High-order adder

Applications

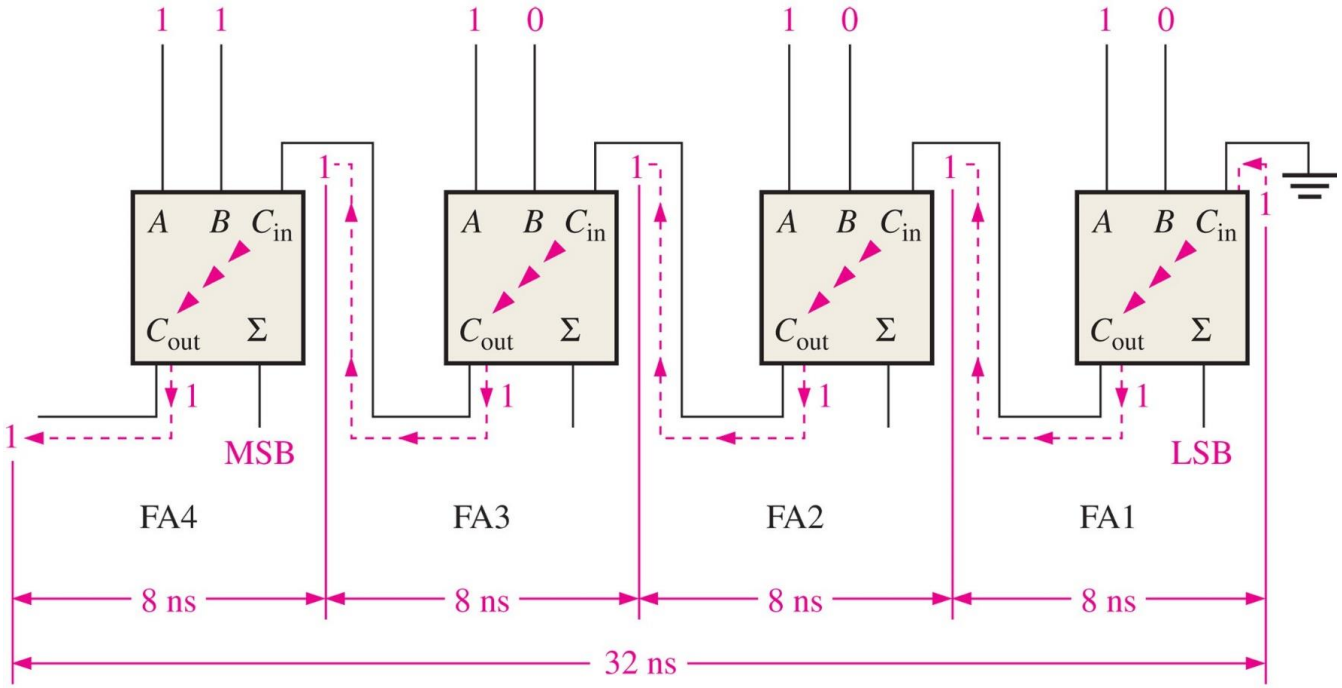
- A voting system that simultaneously counts # of “yes” and # of “no” votes.
- Assuming 6-position setup for simplicity
- Each input is LOW when the switch is in the neutral
- When a switch is moved to the “yes” or “no” position, a HIGH level (VCC) is applied to the associated full adder input.



The Ripple Carry Adder

- The carry output of each full-adder is connected to the carry input of the next higher-order stage (a stage is one full-adder);
- The sum and the output carry of any stage cannot be produced until the input carry occurs, which causes a time delay in the addition process.

The input carry to the least significant stage has to **ripple** through all the adders before a final sum is produced.



Worst-case carry propagation

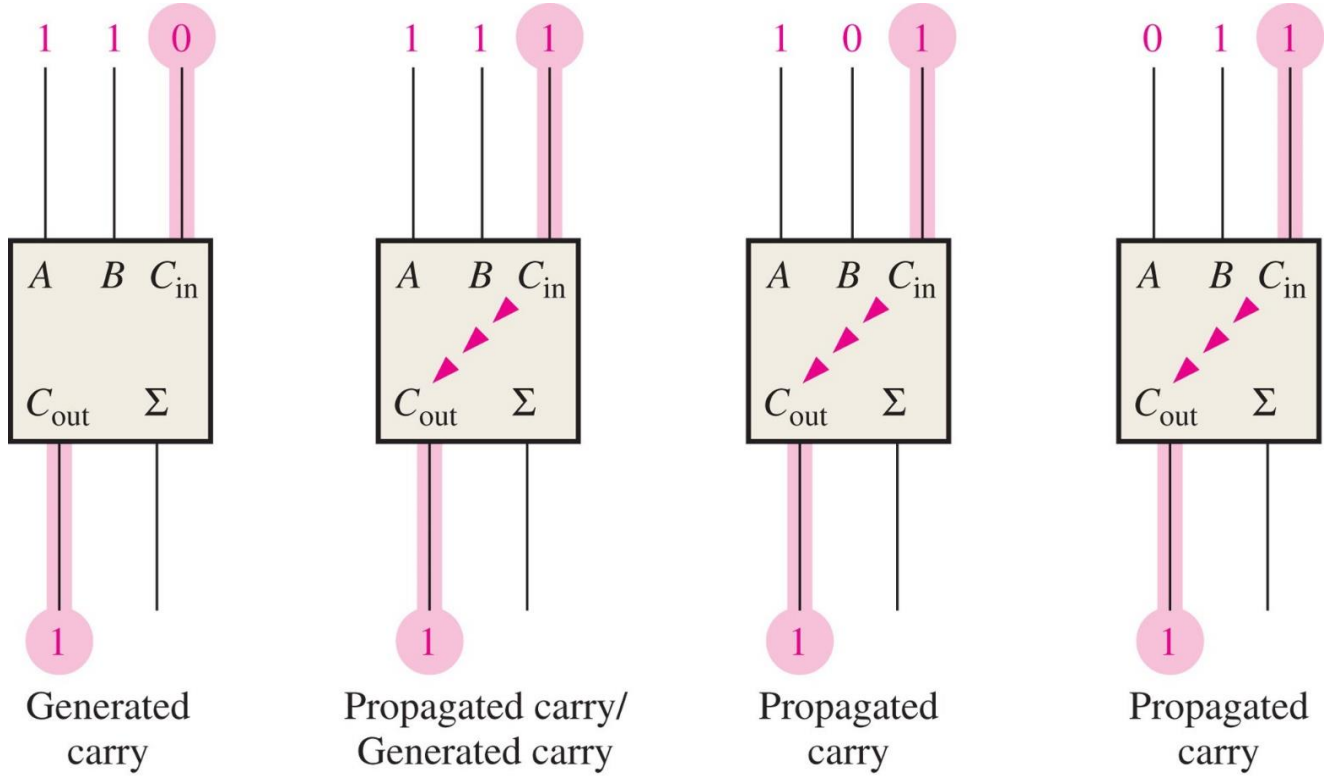
The Look-Ahead Carry Adder

- To speed up the addition process by eliminating the ripple carry delay
- Carry generation: $C_g = AB$
- Carry propagation:

Given the input carry is 1,

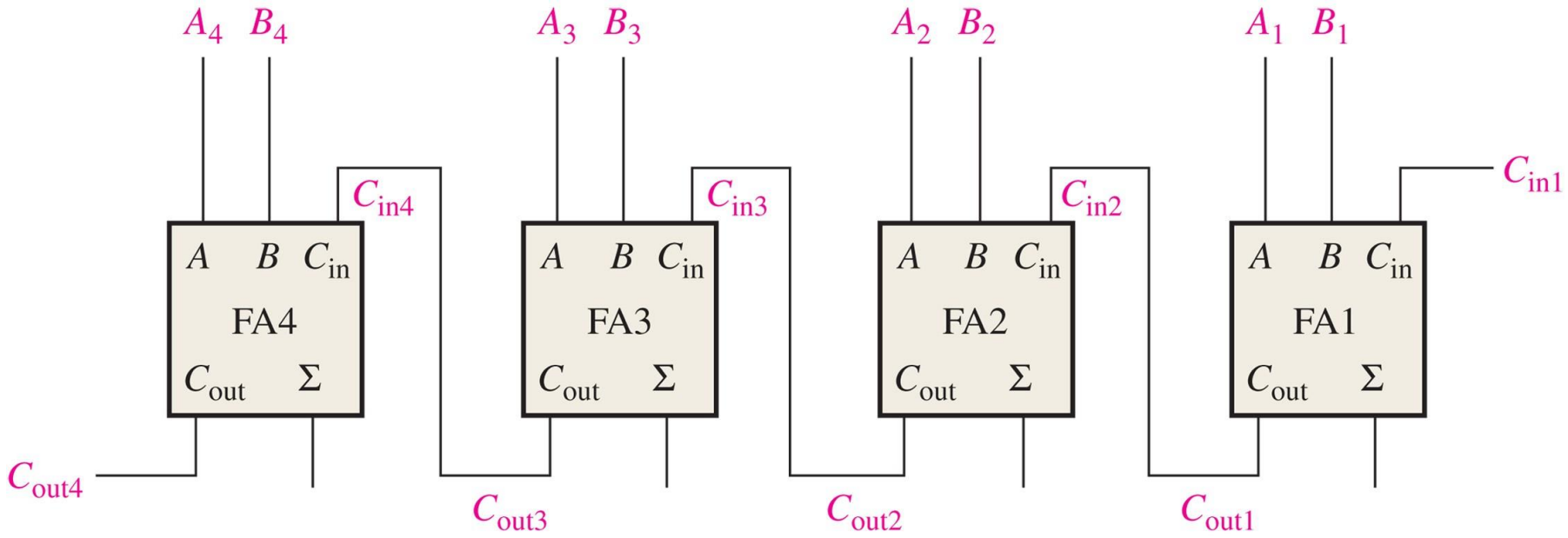
$C_p = A + B$

- Thus, the output carry is given by $C_{out} = C_g + C_p C_{in}$



A 4-Bit Look-Ahead Carry Adder (I)

- Adding $A_4A_3A_2A_1$ and $B_4B_3B_2B_1$



Immediately available

Full-adder 4	Full-adder 3	Full-adder 2	Full-adder 1
$C_{g4} = A_4B_4$	$C_{g3} = A_3B_3$	$C_{g2} = A_2B_2$	$C_{g1} = A_1B_1$
$C_{p4} = A_4 + B_4$	$C_{p3} = A_3 + B_3$	$C_{p2} = A_2 + B_2$	$C_{p1} = A_1 + B_1$

A 4-Bit Look-Ahead Carry Adder (II)

- Express C_{out} in terms of C_p, C_g and C_{in1}

Full-adder 1:

$$C_{out1} = C_{g1} + C_{p1}C_{in1}$$

Full-adder 2:

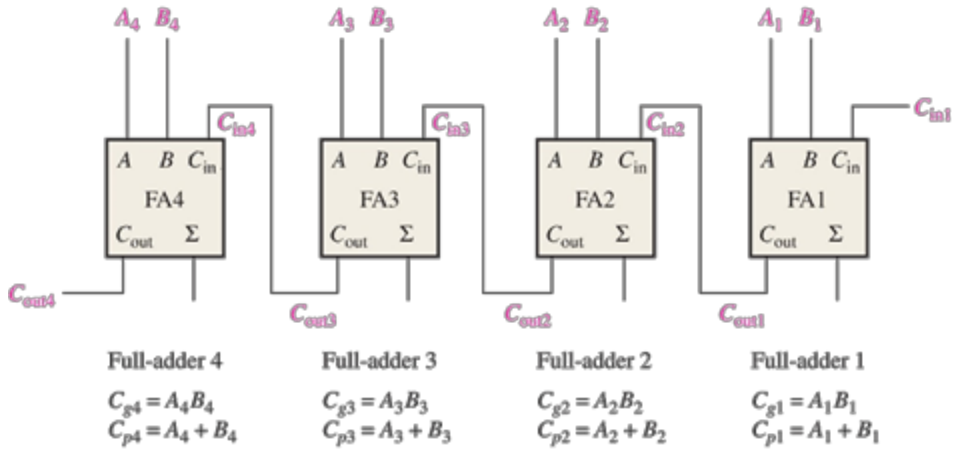
$$C_{in2} = C_{out1}$$
$$C_{out2} = C_{g2} + C_{p2}C_{in2} = C_{g2} + C_{p2}C_{out1} = C_{g2} + C_{p2}(C_{g1} + C_{p1}C_{in1})$$
$$= C_{g2} + C_{p2}C_{g1} + C_{p2}C_{p1}C_{in1}$$

Full-adder 3:

$$C_{in3} = C_{out2}$$
$$C_{out3} = C_{g3} + C_{p3}C_{in3} = C_{g3} + C_{p3}C_{out2} = C_{g3} + C_{p3}(C_{g2} + C_{p2}C_{g1} + C_{p2}C_{p1}C_{in1})$$
$$= C_{g3} + C_{p3}C_{g2} + C_{p3}C_{p2}C_{g1} + C_{p3}C_{p2}C_{p1}C_{in1}$$

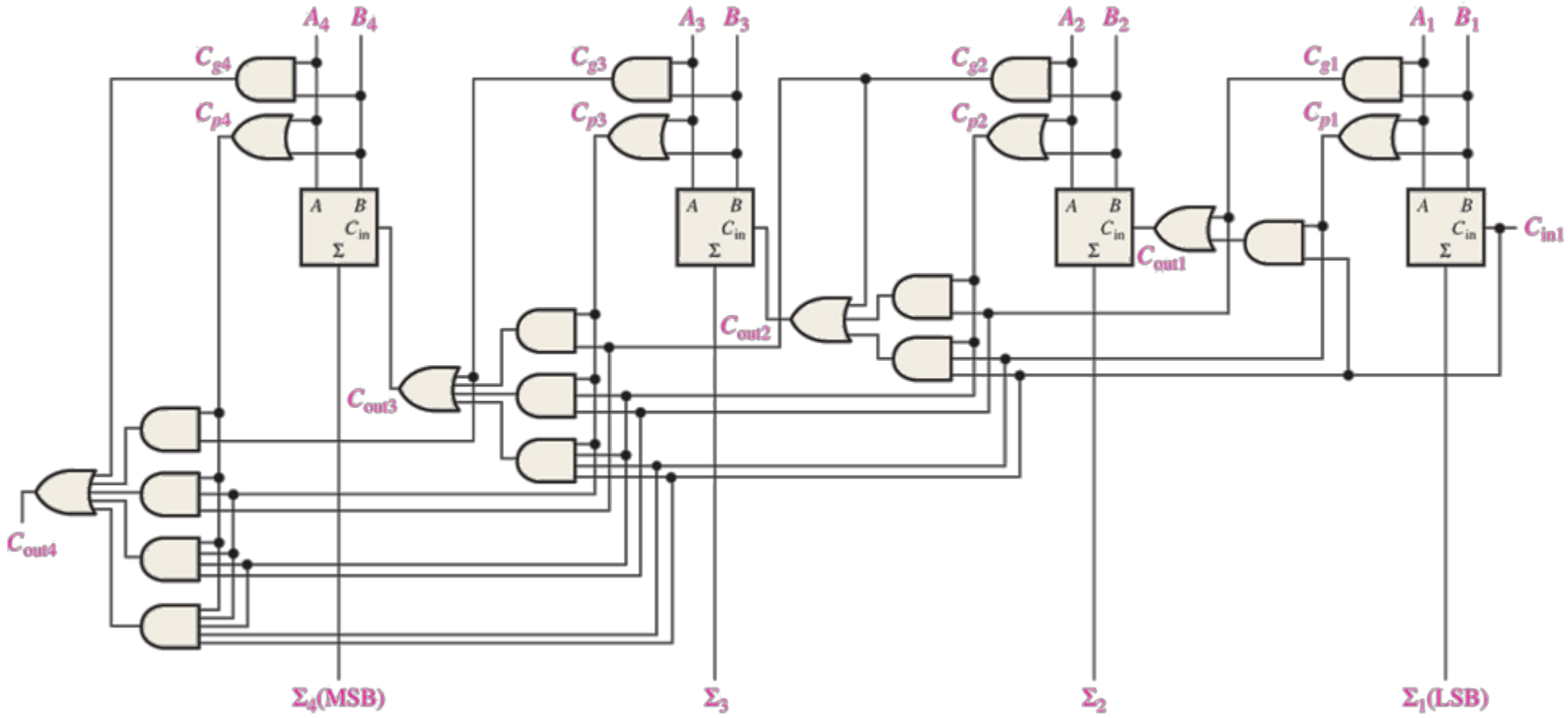
Full-adder 4:

$$C_{in4} = C_{out3}$$
$$C_{out4} = C_{g4} + C_{p4}C_{in4} = C_{g4} + C_{p4}C_{out3}$$
$$= C_{g4} + C_{p4}(C_{g3} + C_{p3}C_{g2} + C_{p3}C_{p2}C_{g1} + C_{p3}C_{p2}C_{p1}C_{in1})$$
$$= C_{g4} + C_{p4}C_{g3} + C_{p4}C_{p3}C_{g2} + C_{p4}C_{p3}C_{p2}C_{g1} + C_{p4}C_{p3}C_{p2}C_{p1}C_{in1}$$



A 4-Bit Look-Ahead Carry Adder (III)

- Logic diagram for a 4-stage look-ahead carry adder



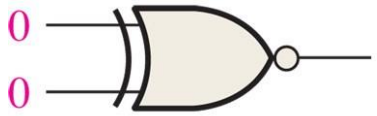
Combine Look-Ahead and Ripple Carry Adders

- Cascading two look-ahead carry adders by connecting the output carry of one adder to the input carry of the next adder.

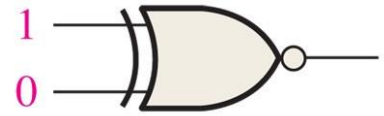
Comparators

Comparator: Equality

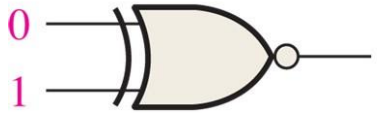
- Exclusive-NOR gate can be used as a basic comparator



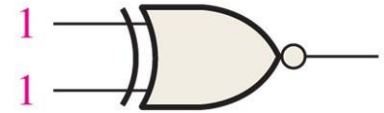
The input bits are equal.



The input bits are not equal.



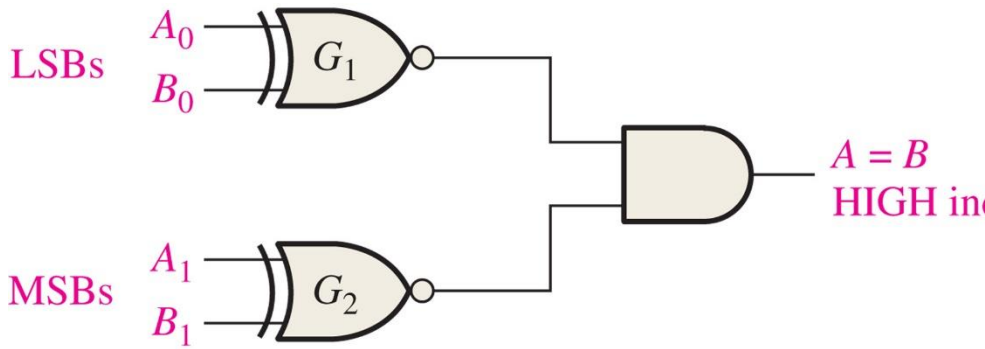
The input bits are not equal.



The input bits are equal.

Comparator: Equality

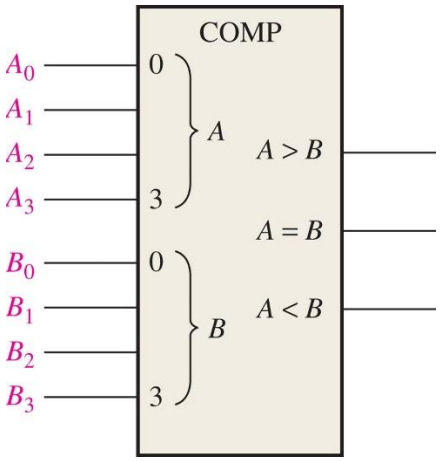
- Equality comparison of two 2-bit numbers



General format: Binary number $A \rightarrow A_1A_0$
Binary number $B \rightarrow B_1B_0$

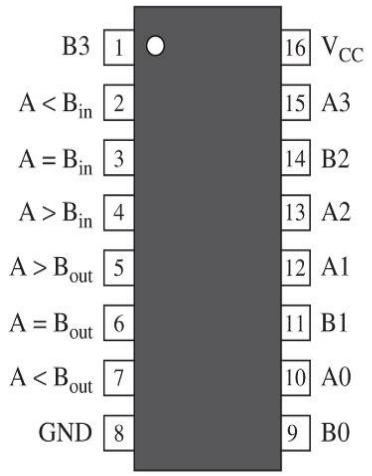
Comparator: Inequality

- Inequality comparison with three outputs indicating
  - Number A is greater than number B (A>B)
  - Number A is less than number B (A< B)
  - Number A is equal to number B (A = B).
- General procedures
  - To check for an inequality in a bit position, starting with the highest-order bits (MSBs).
  - When such an inequality is found, the relationship of the two numbers is established, and any other inequalities in lower-order bit positions must be ignored
- Using a 4-bit comparator as an example
  - If $A_3 = 1$ and $B_3 = 0$, number A is greater than number B.
  - If $A_3 = 0$ and $B_3 = 1$, number A is less than number B.
  - If $A_3 = B_3$, then examine the next lower bit position.

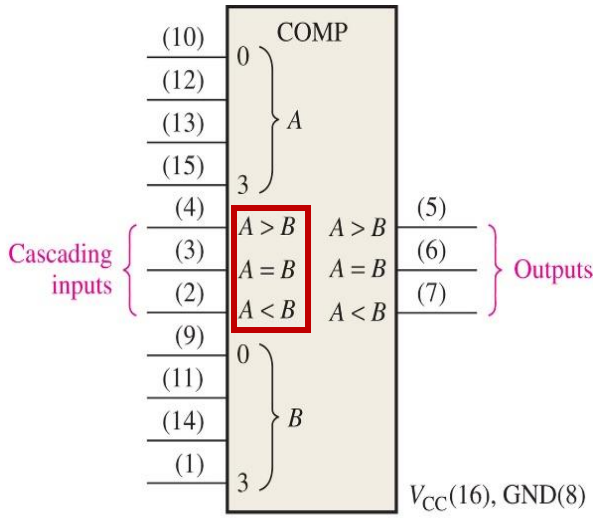


Comparators: 74HC85/74LS85

- The 74HC85/74LS85 pin diagram and logic symbol



(a) Pin diagram



(b) Logic symbol

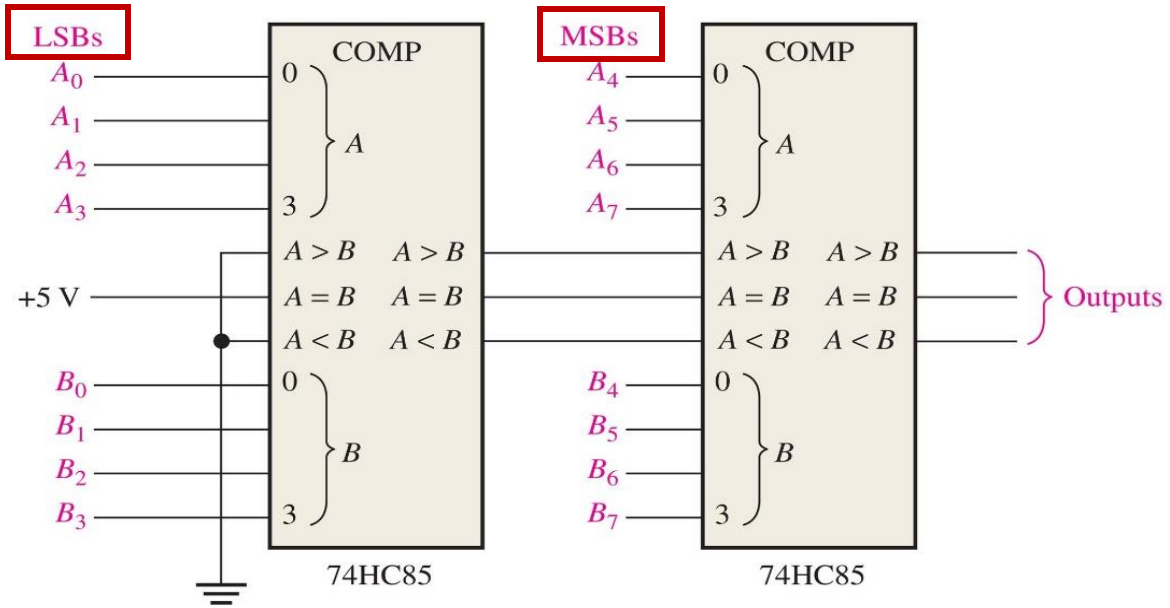
H = HIGH voltage level; L = LOW voltage level; X = don't care.

Comparing inputs				Cascading inputs			Outputs		
A3, B3	A2, B2	A1, B1	A0, B0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$Q_{A>B}$	$Q_{A<B}$	$Q_{A=B}$
A3 > B3	X	X	X	X	X	X	H	L	L
A3 < B3	X	X	X	X	X	X	L	H	L
A3 = B3	A2 > B2	X	X	X	X	X	H	L	L
A3 = B3	A2 < B2	X	X	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 > B1	X	X	X	X	H	L	L
A3 = B3	A2 = B2	A1 < B1	X	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 > B0	X	X	X	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 < B0	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	H	L	L	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	H	L	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	L	H	L	L	H

Ref: https://assets.nexperia.com/documents/data-sheet/74HC_HCT85.pdf

Cascading of Comparators

- Several comparators can be cascaded for number of more bits



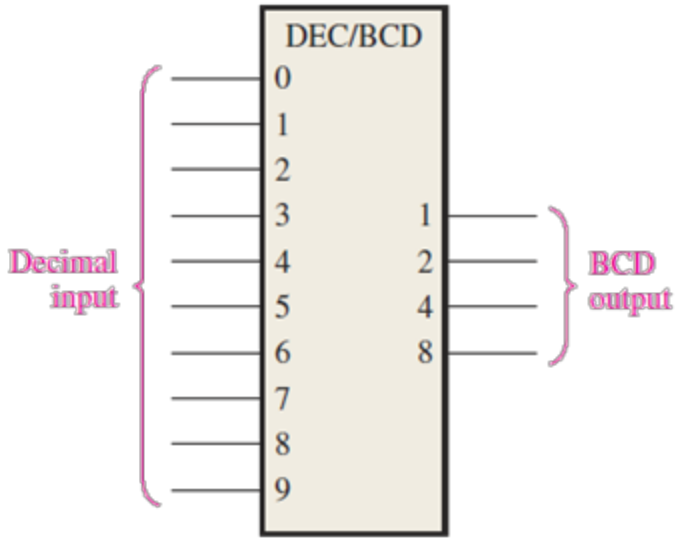
Note the lowest-order comparator must have

- A HIGH on the *A* = *B* input and
- LOWs on the *A*>*B* and *A*<*B* inputs.

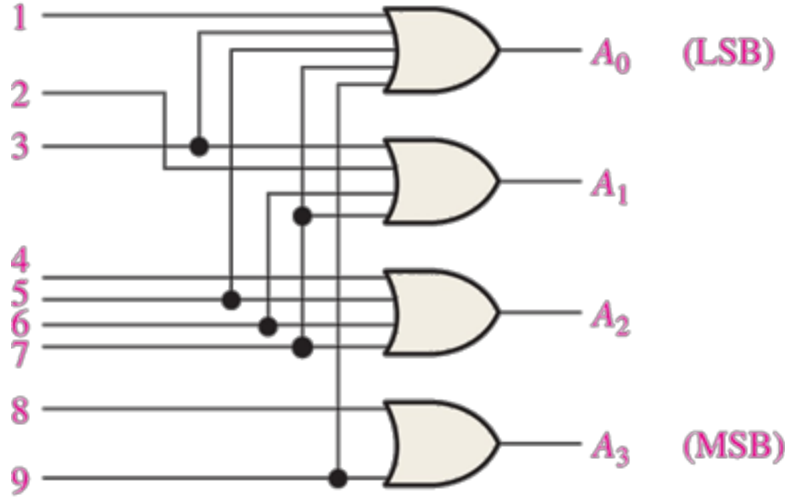
Encoder & Decoder

Encoder

- Input: Accepts an active level on one of its multiple input lines
- Output: multiple *output* lines
- Example: a 10-line-to-4-line encoder



Decimal Digit	BCD Code A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

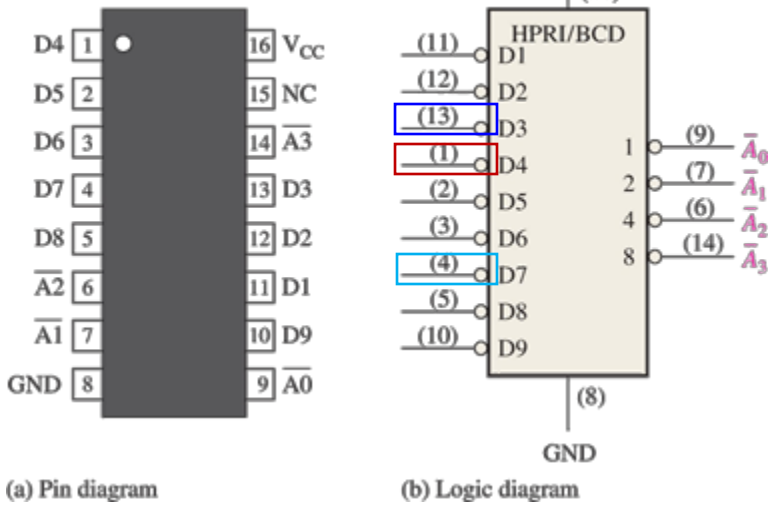


Decimal-to-BCD Priority Encoder

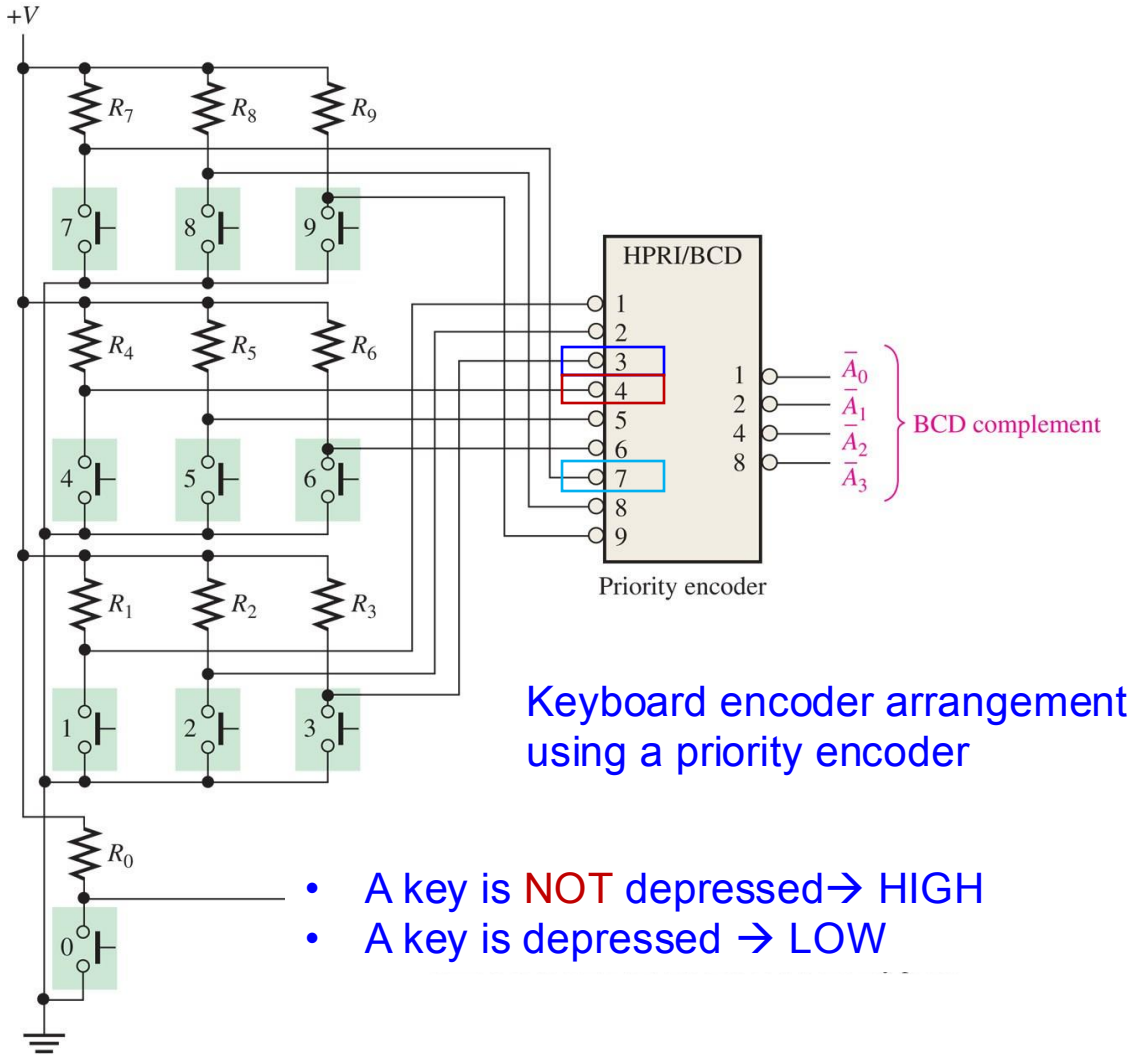
- Produce a BCD output corresponding to the highest-order decimal digit input that is active and will ignore any other lower-order active inputs

Example

- If LOW levels appear on pins, 1, 4, and 13 of the 74HC147 → Pin 4 is the highest-order decimal digit input having a LOW level and represents decimal 7. Therefore, the output levels indicate the BCD code for decimal 7



(a) Pin diagram (b) Logic diagram

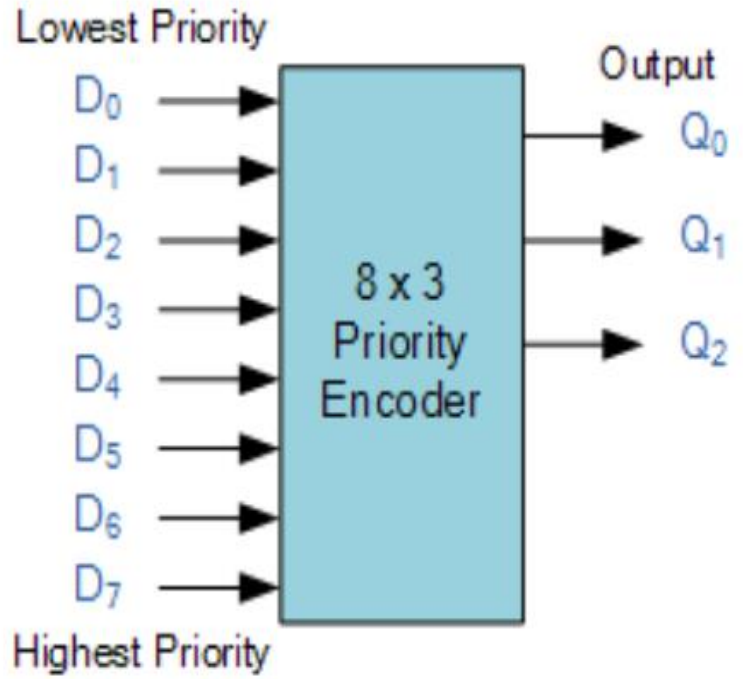


Keyboard encoder arrangement using a priority encoder

- A key is NOT depressed→ HIGH
- A key is depressed → LOW

Priority Encoder

- Use “Don’t Care” to indicate priority by design

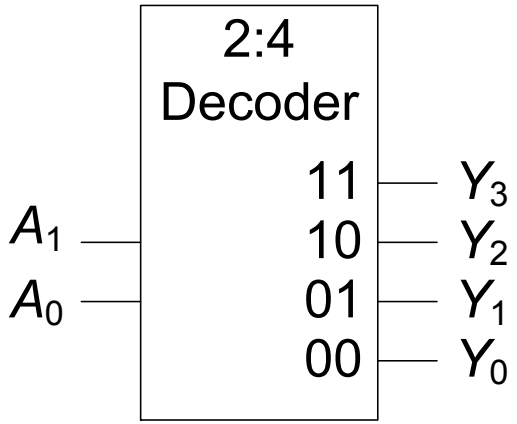


Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = don't care

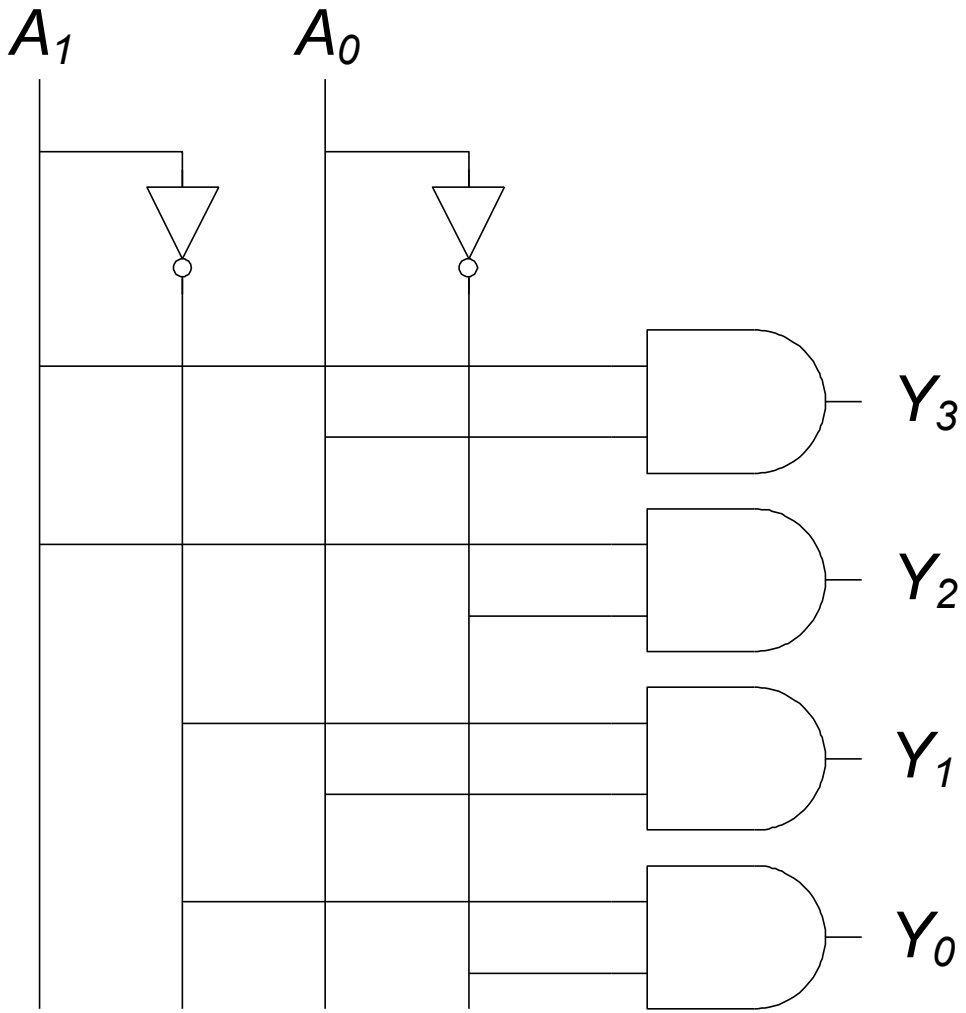
Decoders

- N inputs, 2^N outputs
- **One-hot outputs:** only one output **Active** at once



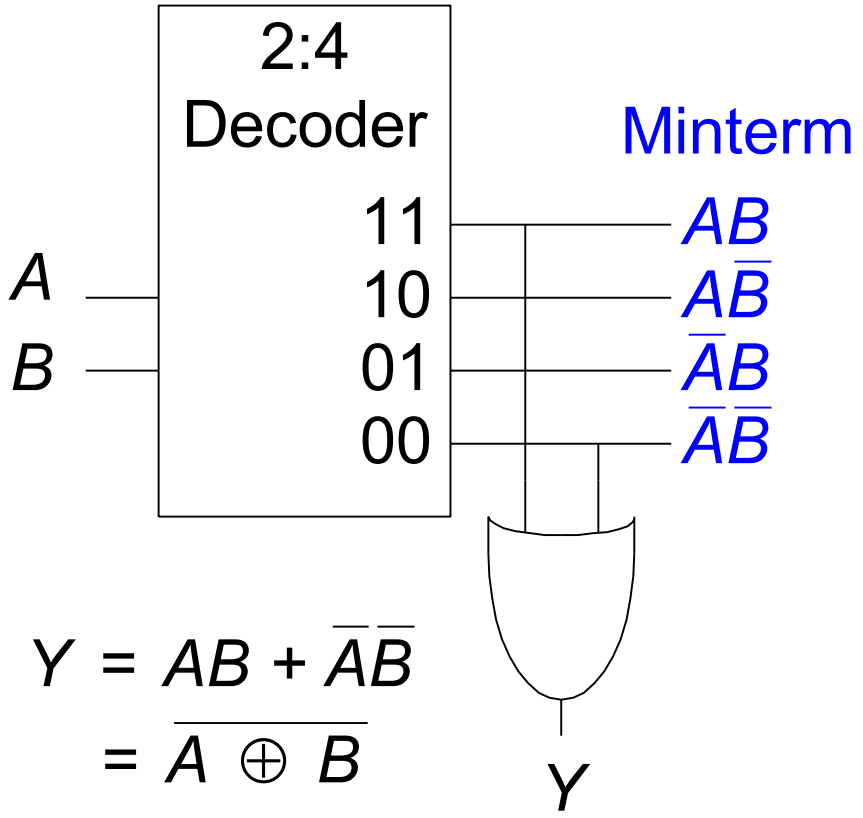
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	**1**
0	1	0	0	**1**	0
1	0	0	**1**	0	0
1	1	**1**	0	0	0

Decoder Implementation



Logic Using Decoders

- OR the minterms:

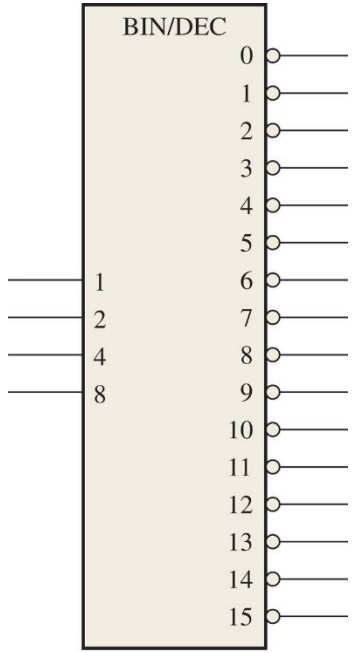


Example: 4-bit Decoder

Decoding functions and truth table for a 4-line-to-16-line (1-of-16) decoder with active-LOW outputs.

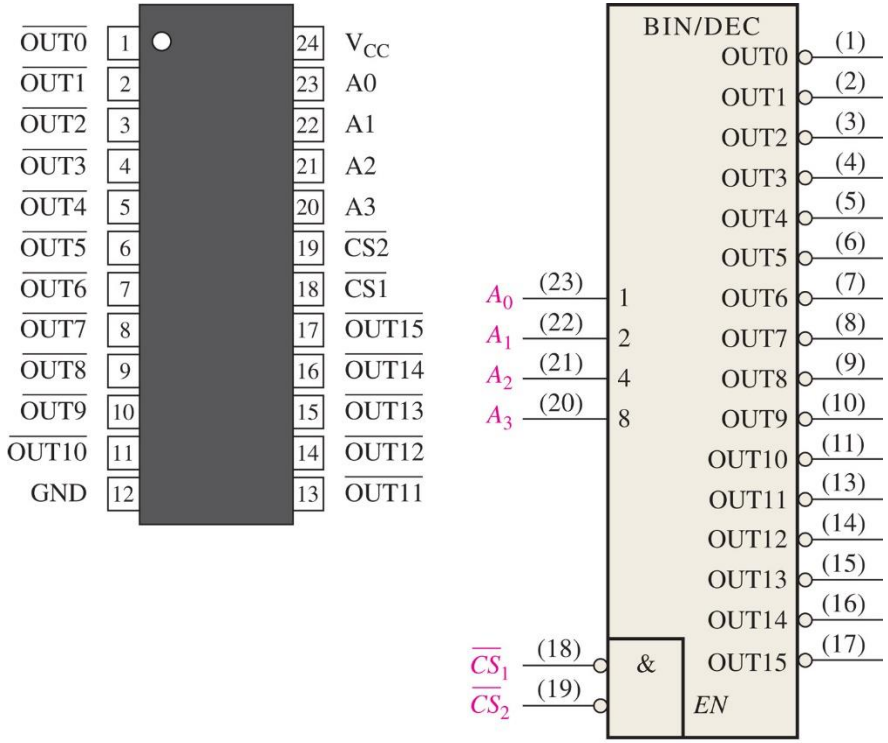
Decimal Digit	Binary Inputs A_3	A_2	A_1	A_0	Decoding Function	Outputs 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1	0	1	0	$A_3\overline{A}_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1	0	1	1	$A_3\overline{A}_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
12	1	1	0	0	$A_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
13	1	1	0	1	$A_3A_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1	1	1	0	$A_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

16 NAND Gates



Decoders: 74HC154

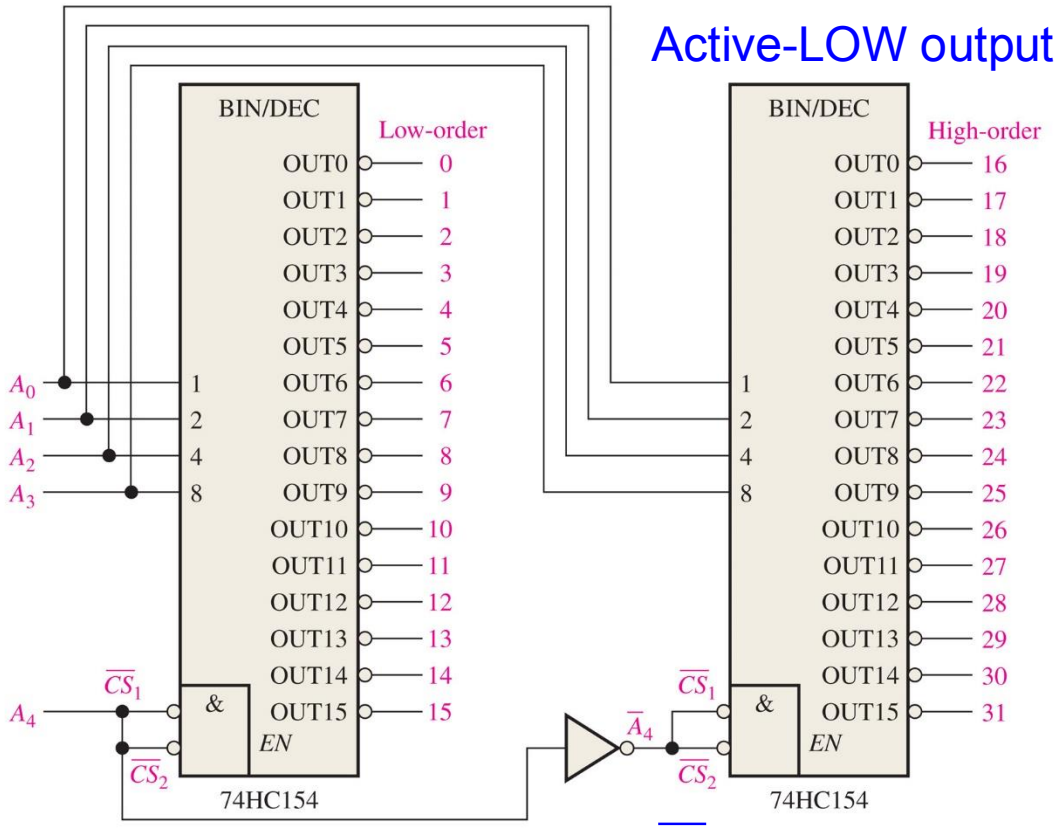
- EN: $\overline{CS_1}$ and $\overline{CS_2}$ both LOW → Output HIGH
- EN is connected to an input of each NAND
- EN: HIGH to enable the decoder; otherwise all decoder outputs will be HIGH



Example

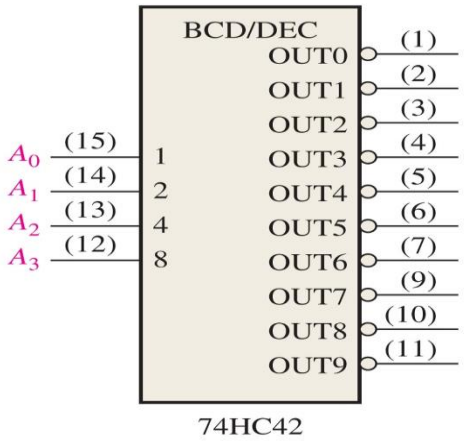
- Implement a 5-bit decoder using two 4-bit decoders (74HC154)

- EN is connected to an input of each NAND
- EN: HIGH to enable the decoder; otherwise all decoder outputs will be HIGH



Decoders: BCD-to-Decimal Decoder

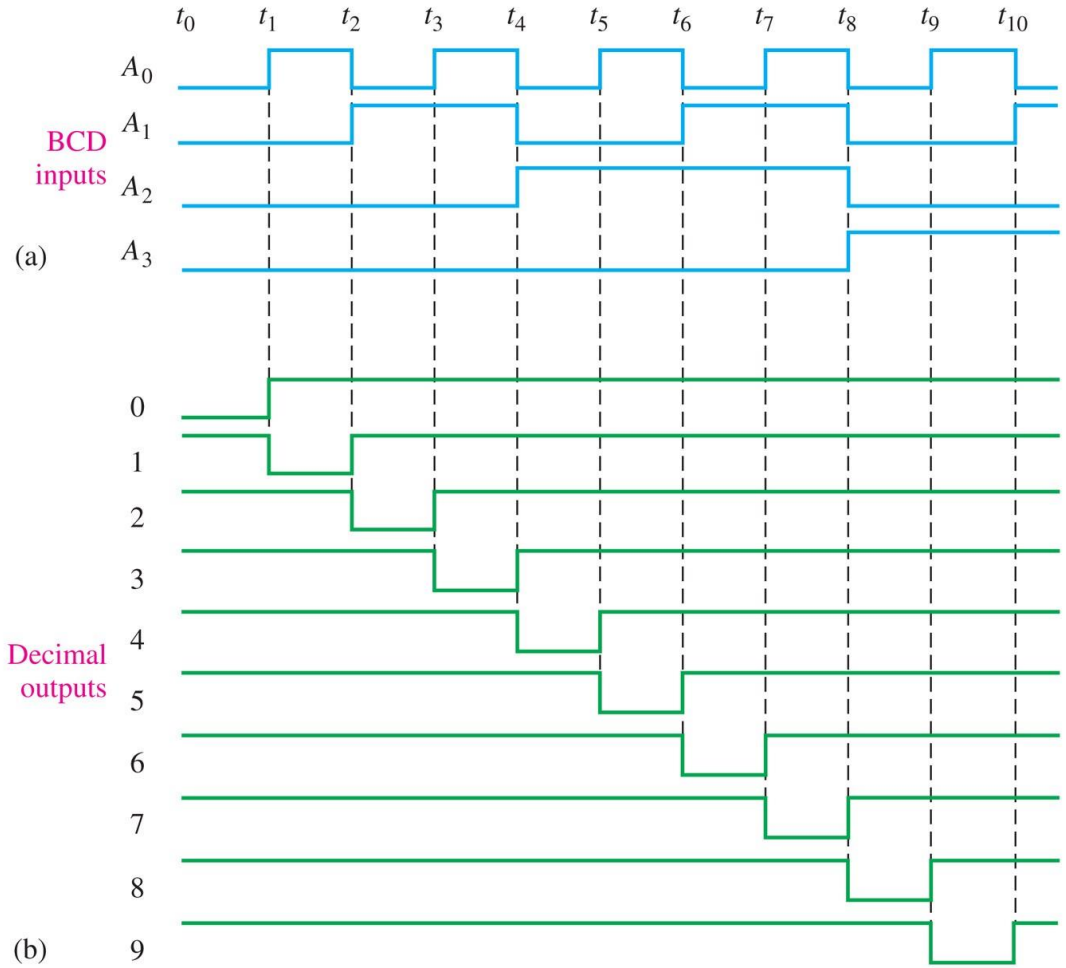
- 4-line-to-10-line decoder: only the ten decimal digits 0 through 9.



Active-LOW output

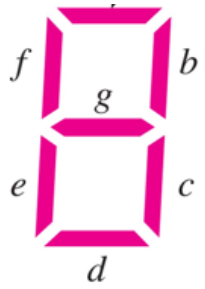
BCD decoding functions.

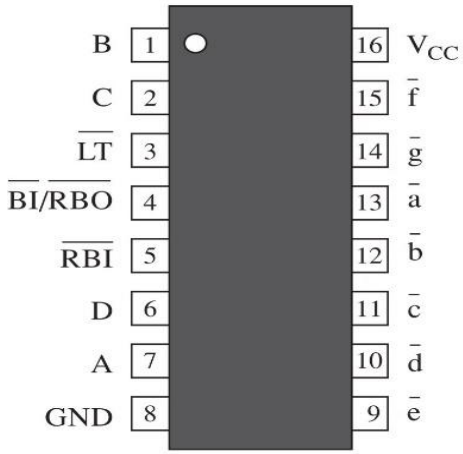
Decimal Digit	BCD Code A_3	A_2	A_1	A_0	Decoding Function
0	0	0	0	0	$\bar{A}_3\bar{A}_2\bar{A}_1\bar{A}_0$
1	0	0	0	1	$\bar{A}_3\bar{A}_2\bar{A}_1A_0$
2	0	0	1	0	$\bar{A}_3\bar{A}_2A_1\bar{A}_0$
3	0	0	1	1	$\bar{A}_3\bar{A}_2A_1A_0$
4	0	1	0	0	$\bar{A}_3A_2\bar{A}_1\bar{A}_0$
5	0	1	0	1	$\bar{A}_3A_2\bar{A}_1A_0$
6	0	1	1	0	$\bar{A}_3A_2A_1\bar{A}_0$
7	0	1	1	1	$\bar{A}_3A_2A_1A_0$
8	1	0	0	0	$A_3\bar{A}_2\bar{A}_1\bar{A}_0$
9	1	0	0	1	$A_3\bar{A}_2\bar{A}_1A_0$

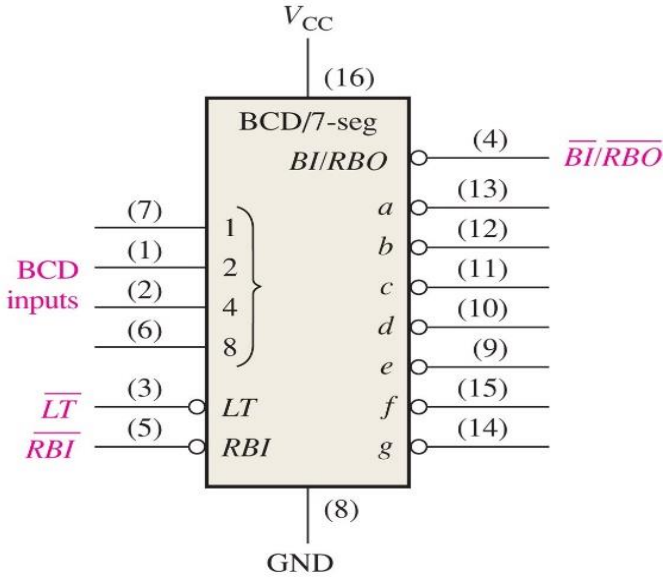


Decoders: The BCD-to-7-Segment Decoder

- Inequality comparison with three outputs indicating
  - LT : Lamp test
  - RBI : Ripple blanking input
  - BI /RBO : either Blanking input or Ripple blanking output
- Lamp test: $\overline{LT}$: LOW, and $\overline{BI}$ / $\overline{RBO}$: HIGH, all segments are turned on



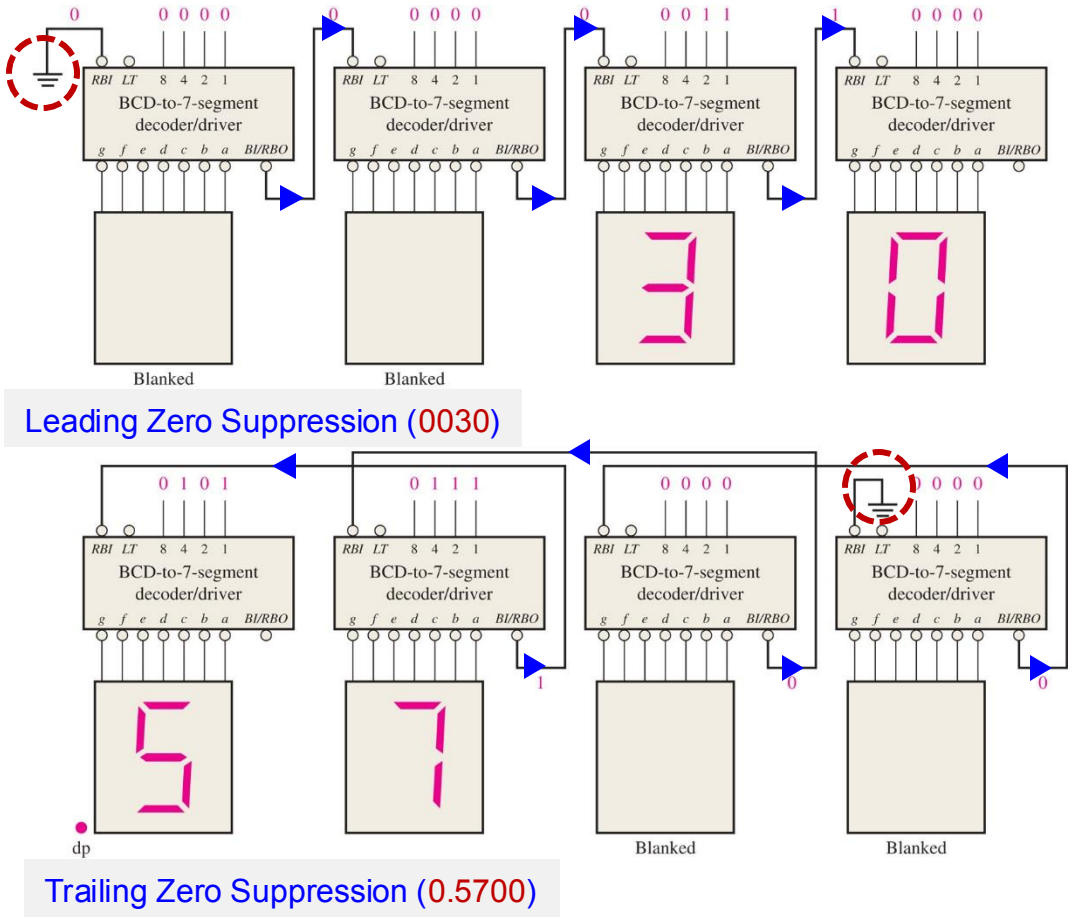




Decoders: The BCD-to-7-Segment Decoder

- BCD inputs = 0000 and $\overline{RBI}$ is LOW, → decoder outputs HIGH, display blanked & $\overline{RBO}$ LOW
- Zero suppression:

  0030.0800 will be displayed as 30.08

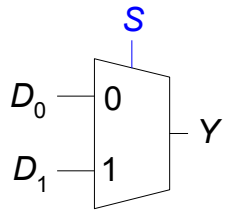


Multiplexers

Multiplexer (Mux): Data Selector

- Selects between one of *N* inputs to connect to output
- **Select** input is **$\log_2 N$ bits** – control input
- **Example:**

2:1 Mux



S	D_1	D_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	0
1	1	0	1
1	1	1	1

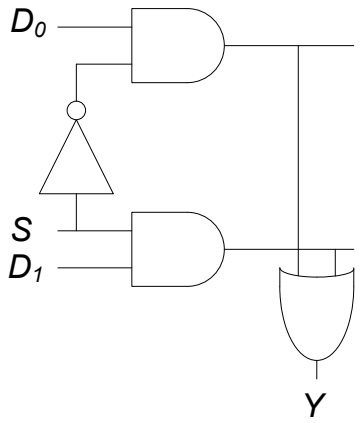
S	Y
0	D_0
1	D_1

2:1 Multiplexer Implementations

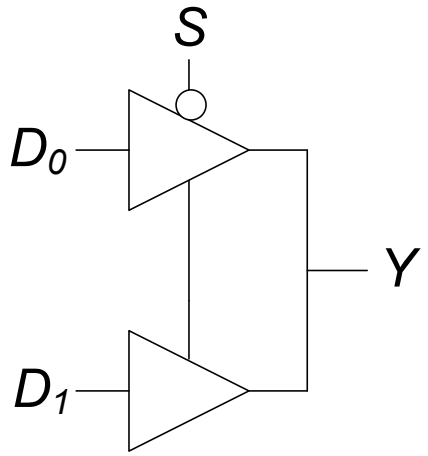
- **Logic gates**
  - Sum-of-products form

Y	D_0D_1 = 00	01	11	10
S = 0	0	0	1	1
S = 1	0	1	1	0

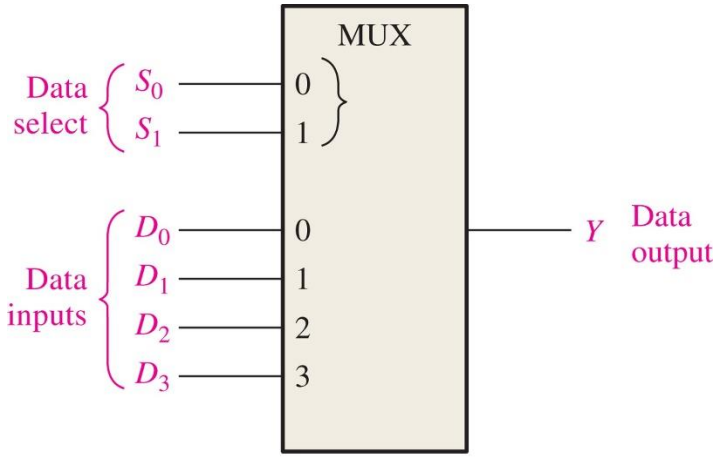
$$Y = D_0\overline{S} + D_1S$$



- **Tristates**
  - Two tristates
  - Turn on exactly one to select the appropriate input

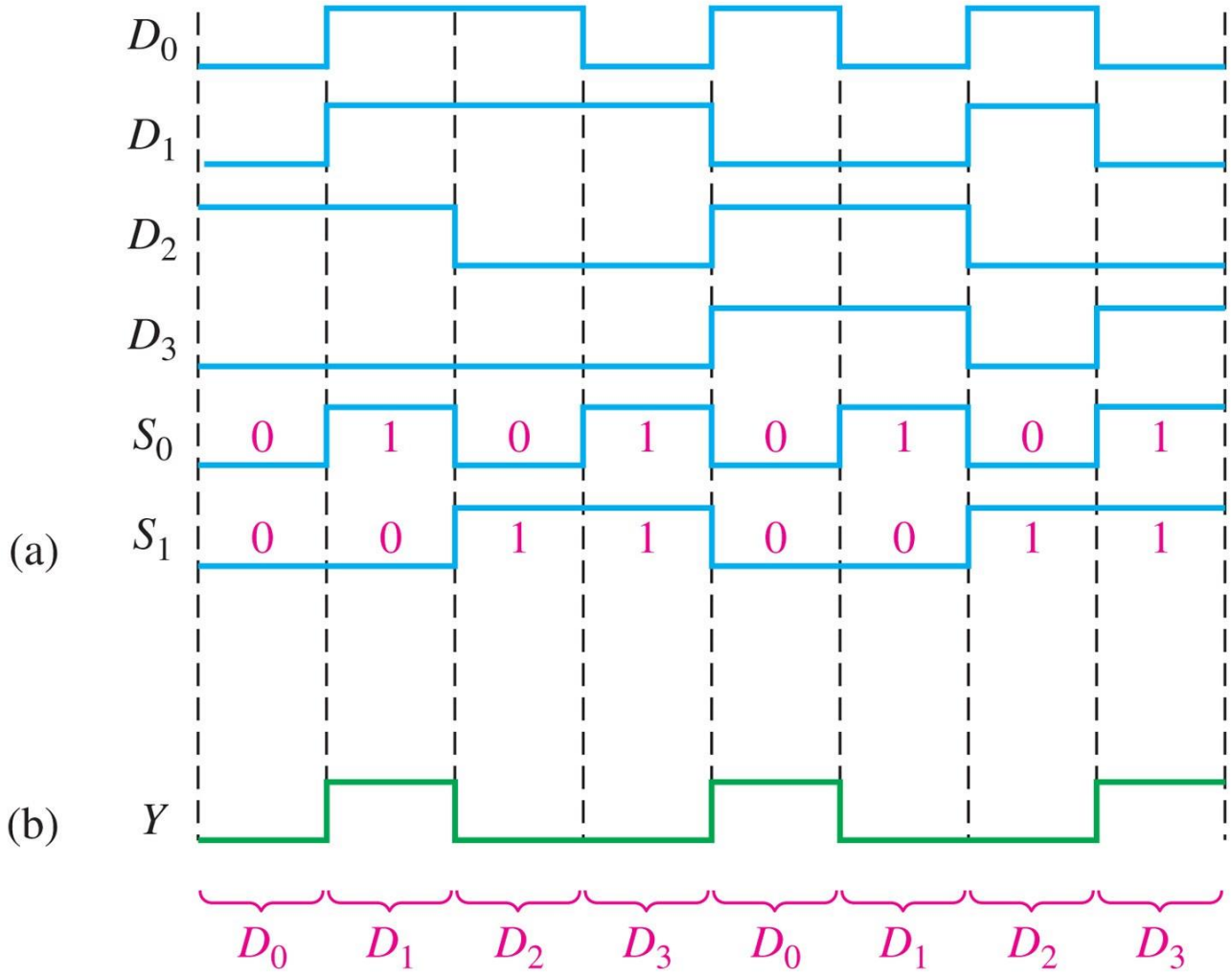


Example: 4:1 Mux



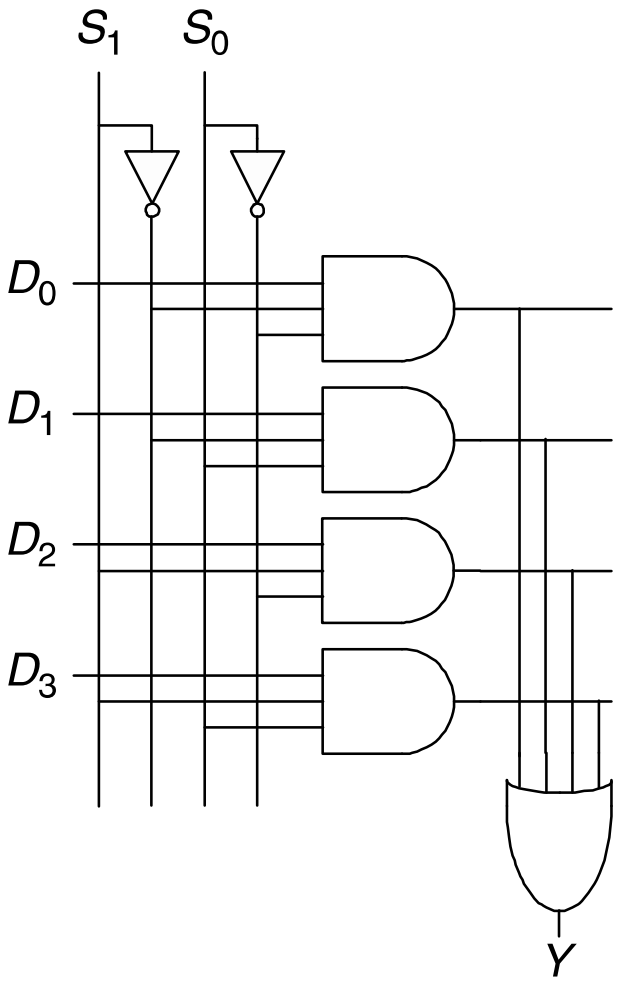
Data selection for a 1-of-4-multiplexer.

Data-Select Inputs		
S_1	S_0	Input Selected
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

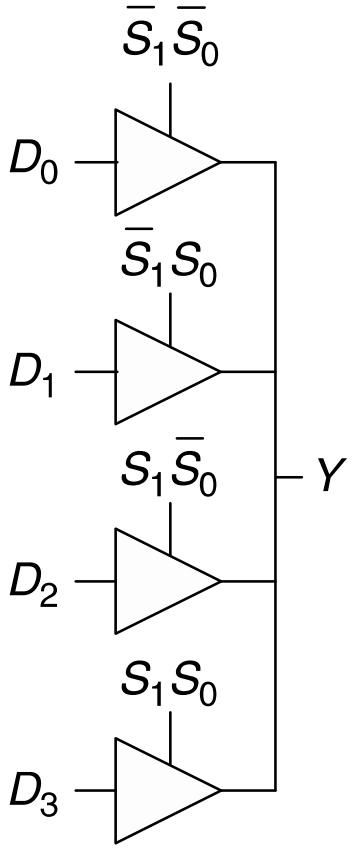


4:1 Multiplexer Implementations

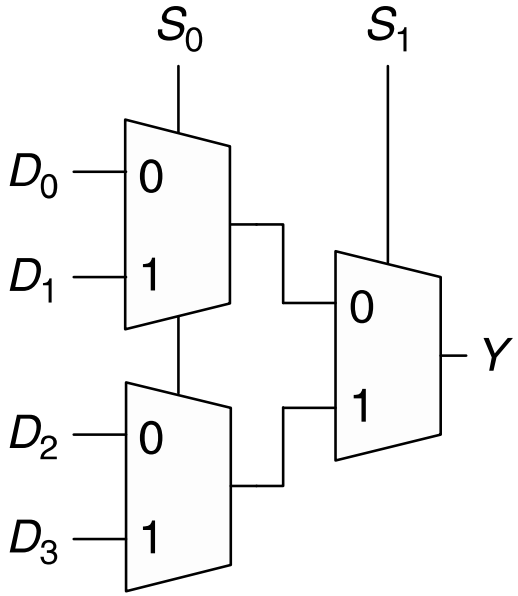
2-Level Logic



Tristates



4:1 Mux Symbol

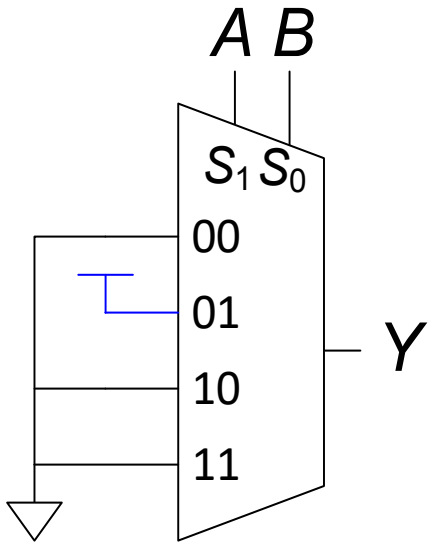


Logic using Multiplexers

- Using MUX as a **lookup table**

A	B	Y
0	0	0
0	1	1
1	0	0
1	1	0

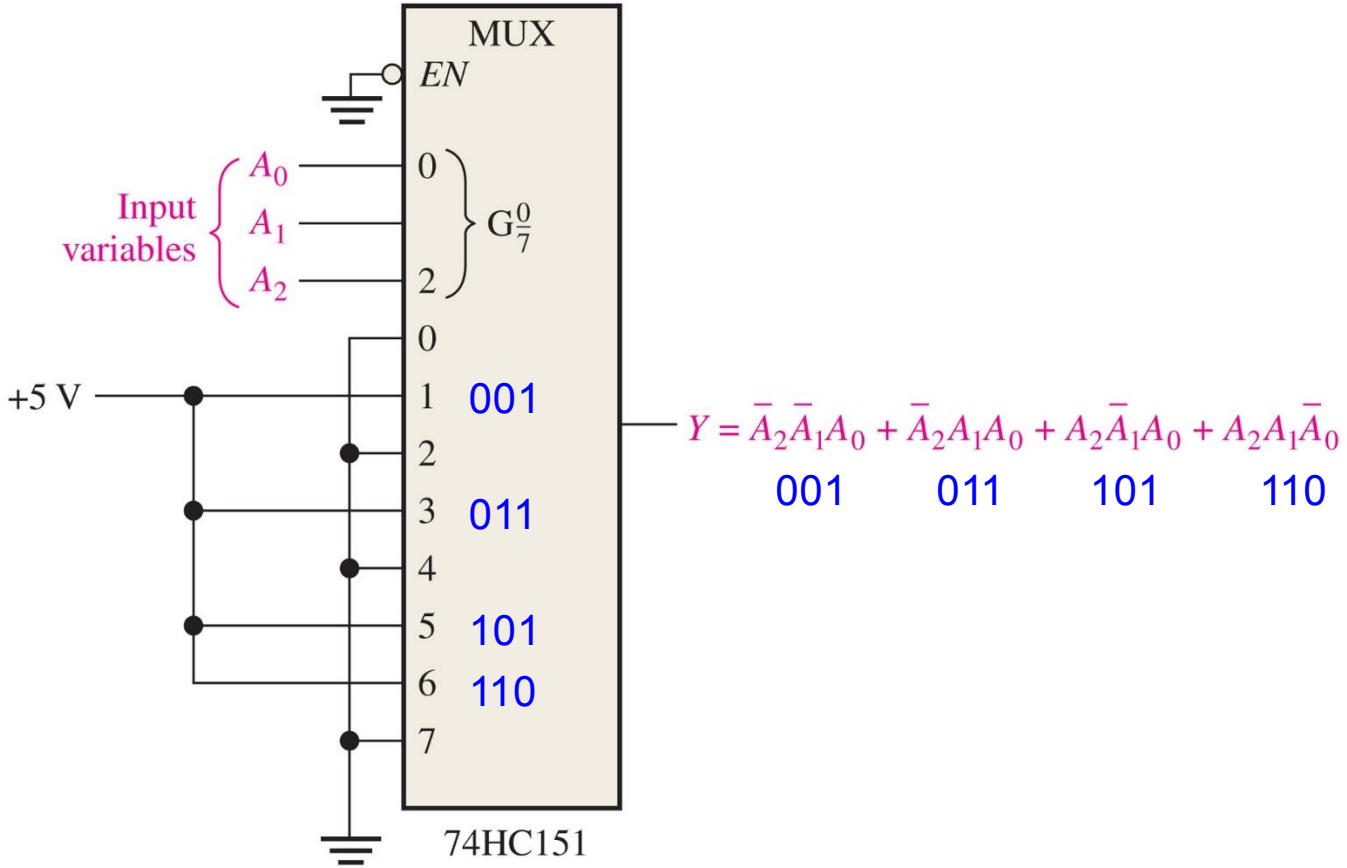
$Y = \overline{A}B$



Logic using Multiplexers

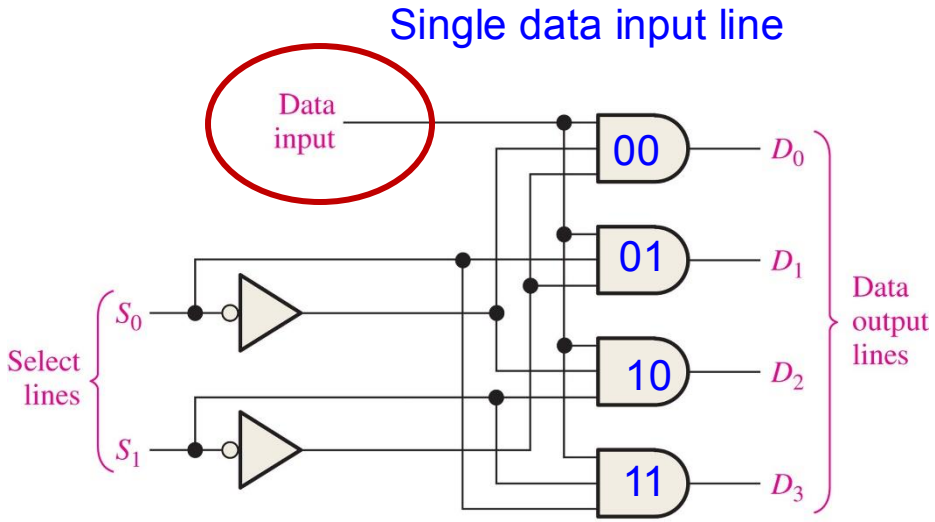
- Using MUX as a **logic function generator**

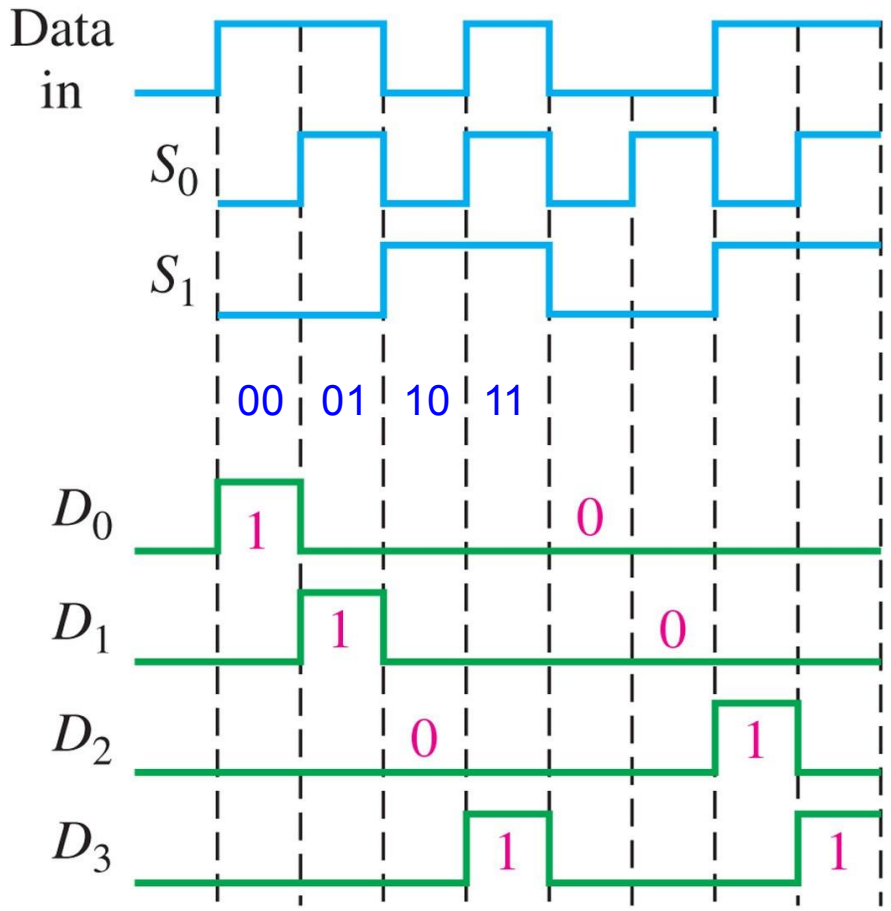
Inputs			Output
A_2	A_1	A_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



Demultiplexer: data distributor

- Aka. a data distributor that reverses the multiplexing function

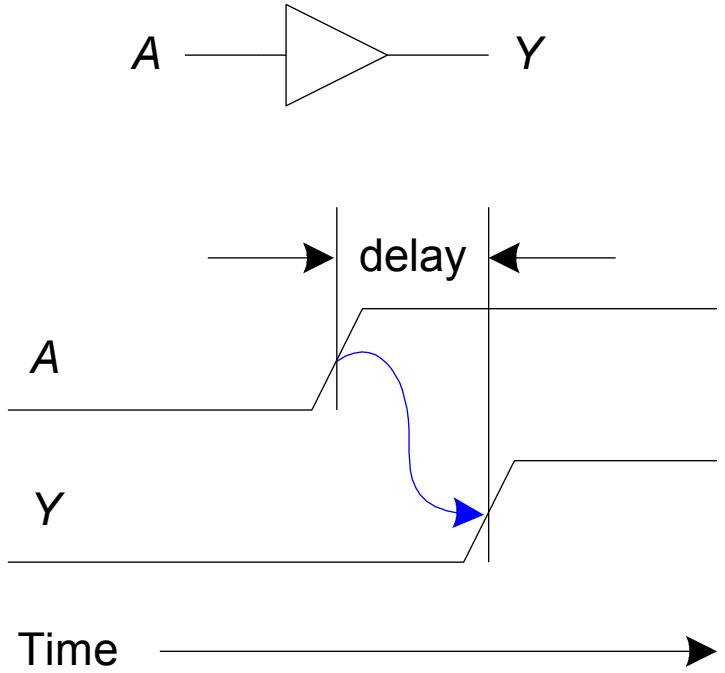




Timing

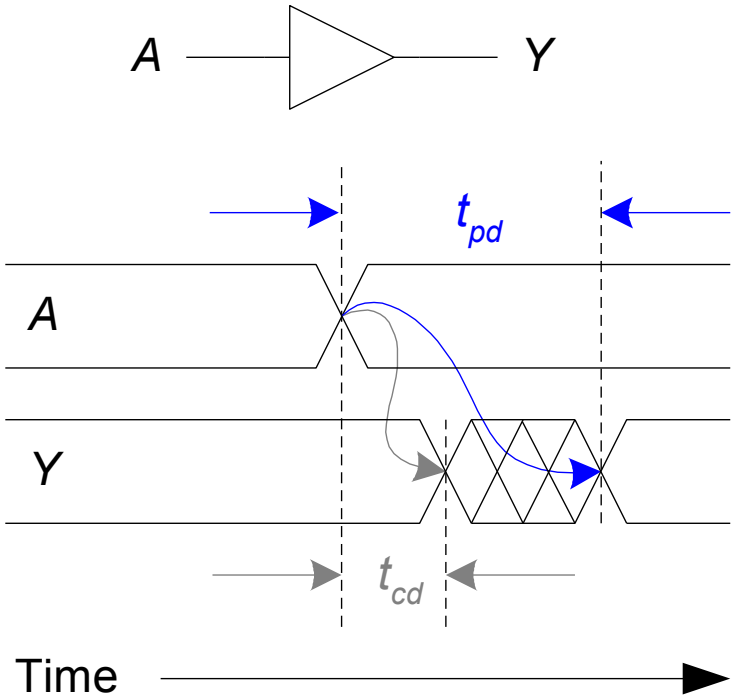
Timing

- Delay: time between input change and output changing
- How to build fast circuits?



Propagation & Contamination Delay

- **Propagation delay:** t_{pd} = **max** delay from input to output
- **Contamination delay:** t_{cd} = **min** delay from input to output

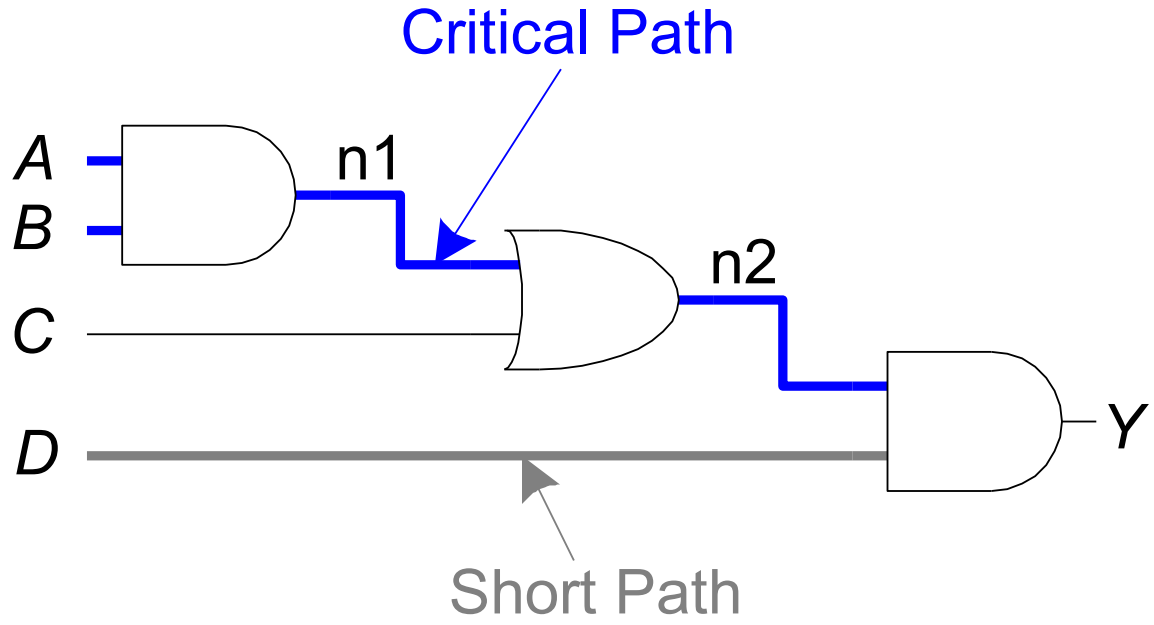


Propagation & Contamination Delay

- **Delay is caused by**
  - Capacitance and resistance in a circuit
  - Speed of light limitation

- **Reasons why t_{pd} and t_{cd} may be different:**
  - Different rising and falling delays
  - Multiple inputs and outputs, some of which are faster than others
  - Circuits slow down when hot and speed up when cold

Critical (Long) & Short Paths



Critical (Long) Path: $t_{pd} = 2t_{pd\_AND} + t_{pd\_OR}$ **(max delay)**

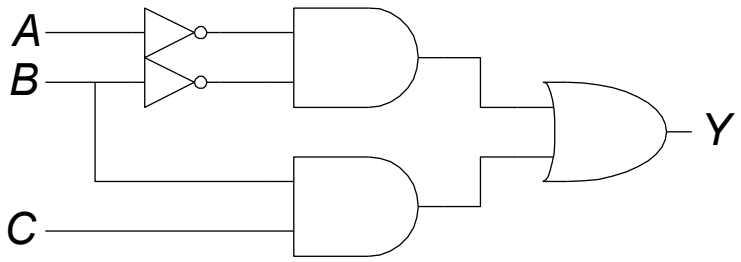
Short Path: $t_{cd} = t_{cd\_AND}$ **(min delay)**

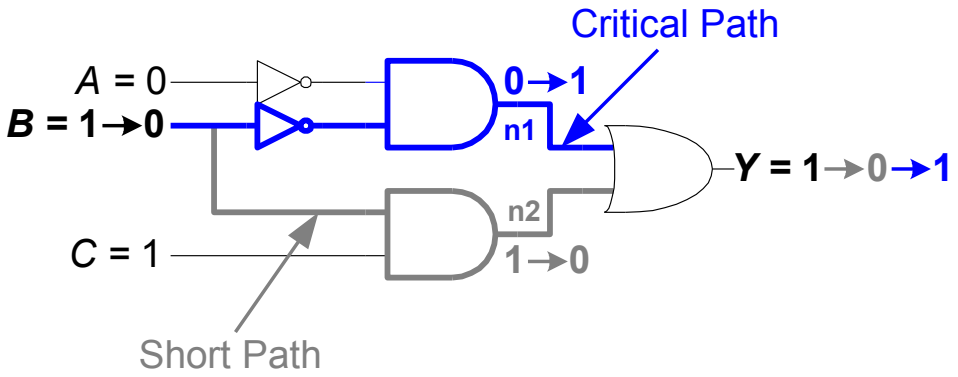
Glitches

- When a single input change causes an output to change multiple times

Glitch Example

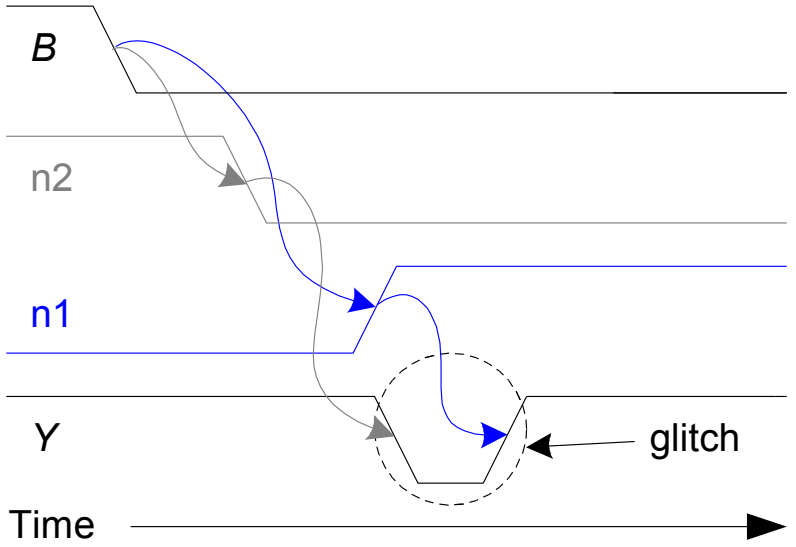
- What happens when A = 0, C = 1, B falls?



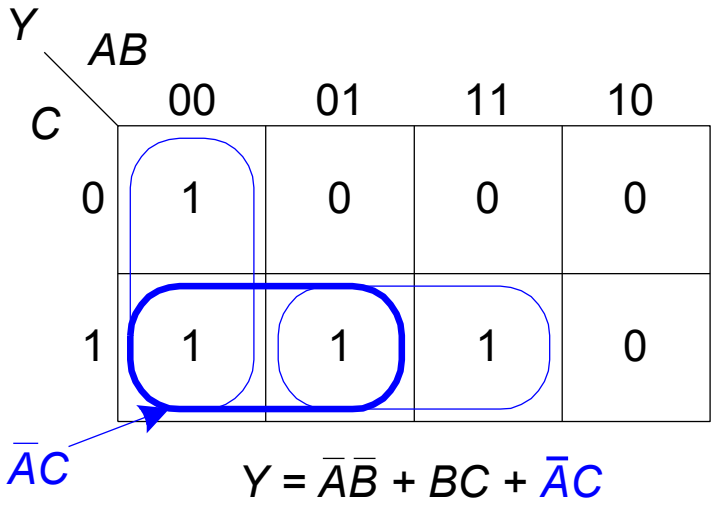


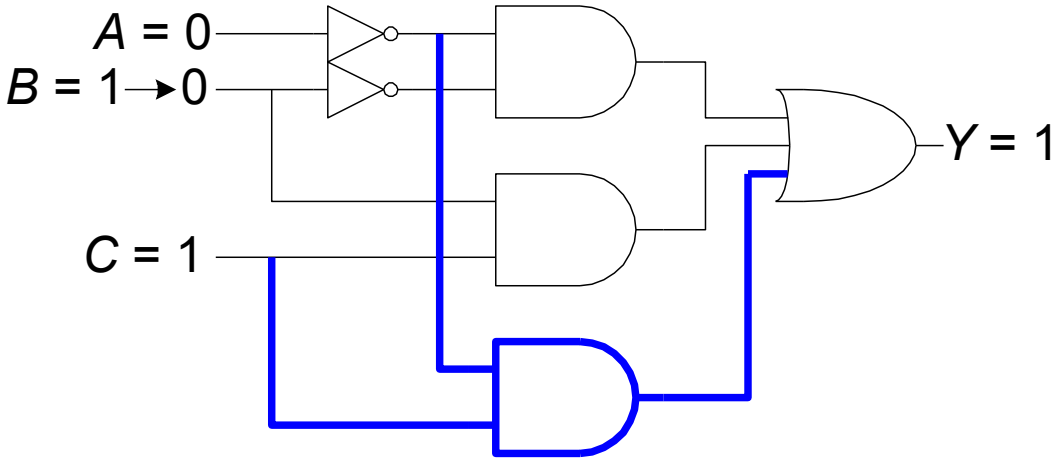
Y	AB = 00	01	11	10
C = 0	1	0	0	0
1	1	1	1	0

$Y = \bar{A}\bar{B} + BC$



Fixing the Glitch



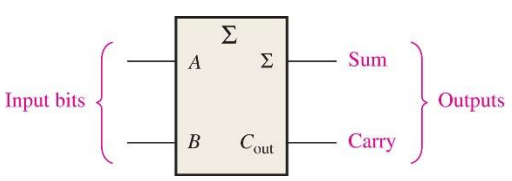


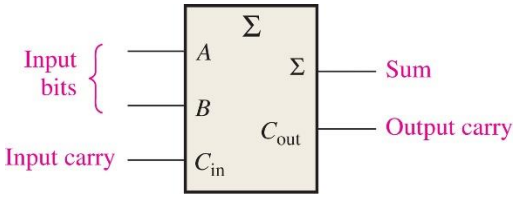
Why Understand Glitches?

- Because of **synchronous design** conventions (see Chapter 3), glitches don't cause problems.
- It's important to **recognize** a glitch: in simulations or on oscilloscope.
- We **can't get rid of all glitches** – simultaneous transitions on multiple inputs can also cause glitches.

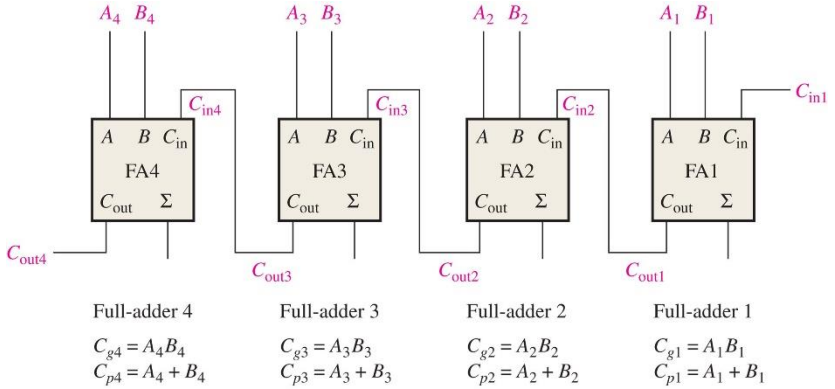
Chapter Review

- Half and Full Adders
- Parallel Binary Adders
- Ripple Carry and Look-Ahead Carry Adders
- Comparators
- Decoders (n input lines → max. 2^n output lines)
- Encoders (e.g. Decimal-to-BCD Priority Encoder)
- Code Converters (e.g. BCD-to-Binary Conversion)
- Multiplexers (To route several inputs onto a single output line)
- Demultiplexers
- Timing

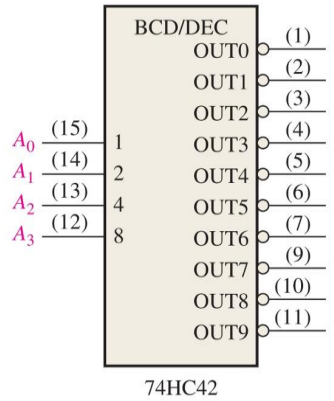


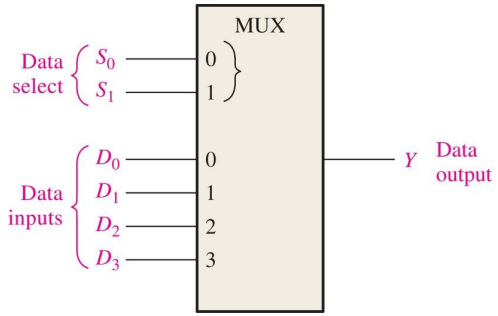


How to design a multiplier by FA/HA?



True/False Quiz





- ✓ A half-adder adds two binary bits.
- ✗ A half-adder has a carry output only.
- ✗ A full adder adds two bits and produces two outputs.
- ✗ A full-adder can be realized only by using 2-input XOR gates.
- ✓ When the input bits are both 1 and the input carry bit is 1, the sum output of a full adder is 1.
- ✗ The output of a comparator is 0 when the two binary inputs given are equal.
- ✓ A decoder detects the presence of a specified combination of input bits.
- ✗ The 4-line-to-10-line decoder and the 1-of-10 decoder are two different types.
- ✓ An encoder essentially performs a reverse decoder function.
- ✗ A multiplexer is a logic circuit that allows digital information from a single source to be routed onto several lines.