Digital Circuits

ECS 371

Dr. Prapun Suksompong

prapun@siit.tu.ac.th

Lecture 13

Office Hours:
BKD 3601-7
Monday 9:00-10:30, 1:30-3:30
Tuesday 10:30-11:30

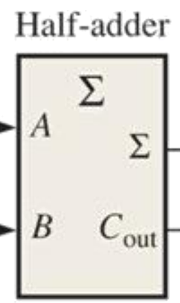
ECS371.PRAPUN.COM

Adder

Half-adder: A digital circuit that adds two bits and produces a sum and an output carry. It cannot handle input carries.

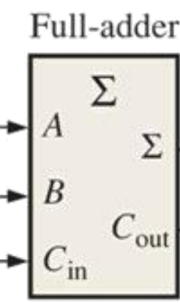
Full-adder: A digital circuit that adds two bits and an input carry to produce a sum and an output carry.

Half-adder



INPUTS A	INPUTS B	CARRY OUT C_{out}	SUM Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full-adder

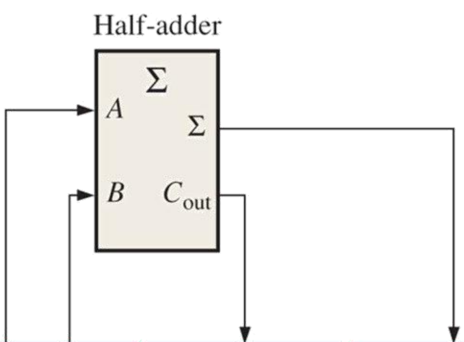


INPUTS A	INPUTS B	CARRY IN C_{in}	CARRY OUT C_{out}	SUM Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

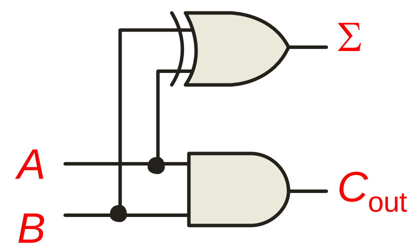
$1+1+0 = 2_{10} = 10_2$

1-bit Adder: Half Adder

- The basic difference between a **full-adder** and a **half-adder** is that the full-adder accepts an input carry.
- Half-Adder:



INPUTS A	INPUTS B	CARRY OUT C_{out}	SUM Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

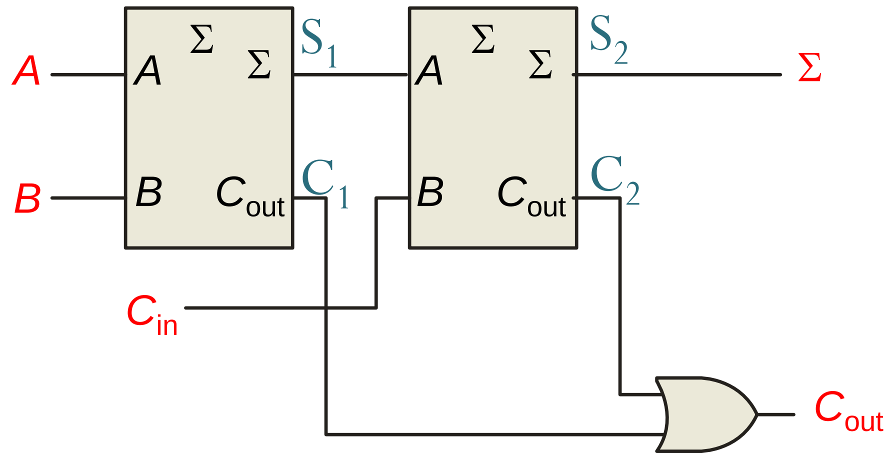


$$\Sigma = A \oplus B$$

$$C_{out} = AB$$

1-bit Adder: Full-Adder

- We will construct a full adder by first adding A and B using a half-adder.

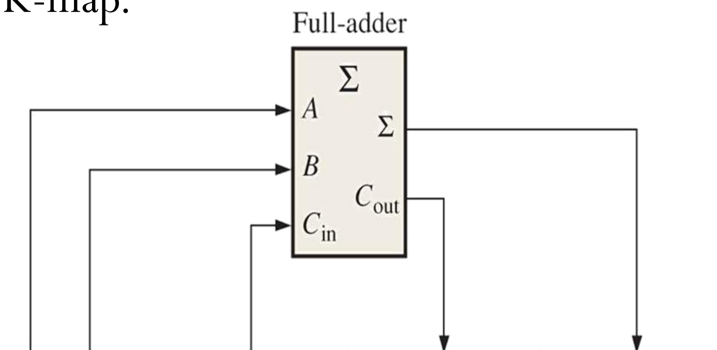


- Then, we use a second half-adder to add C_{in} to the result of the first half-adder.

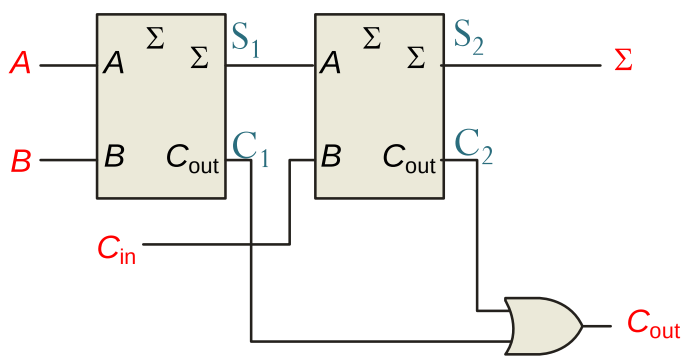
The Output Carry of Full Adder

$$
\begin{aligned}
C_{out} &= \boxed{AB + BC_{in} + AC_{in}} \\
&= AB + (A + B)C_{in} \\
&= AB + (A \oplus B + AB)C_{in} \\
&= AB + (A \oplus B)C_{in} + ABC_{in} \\
&= AB + (A \oplus B)C_{in} \\
&= C_1 + S_1 C_{in} = C_1 + C_2
\end{aligned}
$$

This is from K-map.



INPUTS A	B	CARRY IN C_{in}	CARRY OUT C_{out}	SUM 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



Multiple-bit Addition

- When one (multiple-bit) binary number is added to another, each column generates a sum bit and a 1 or 0 carry bit to the next column to the left.

$$
\begin{array}{lrr}
 & 1111 & \\
A: & 0111 & 7 \\
B: & +\ 1101 & +\ 13 \\
\hline
 & 10100 = & 20
\end{array}
$$

- To add binary numbers with more than one bit, we must use additional full-adders.
  - For 2-bit numbers, two adders are needed;
  - for 4-bit numbers, four adders are used;
  - and so on.

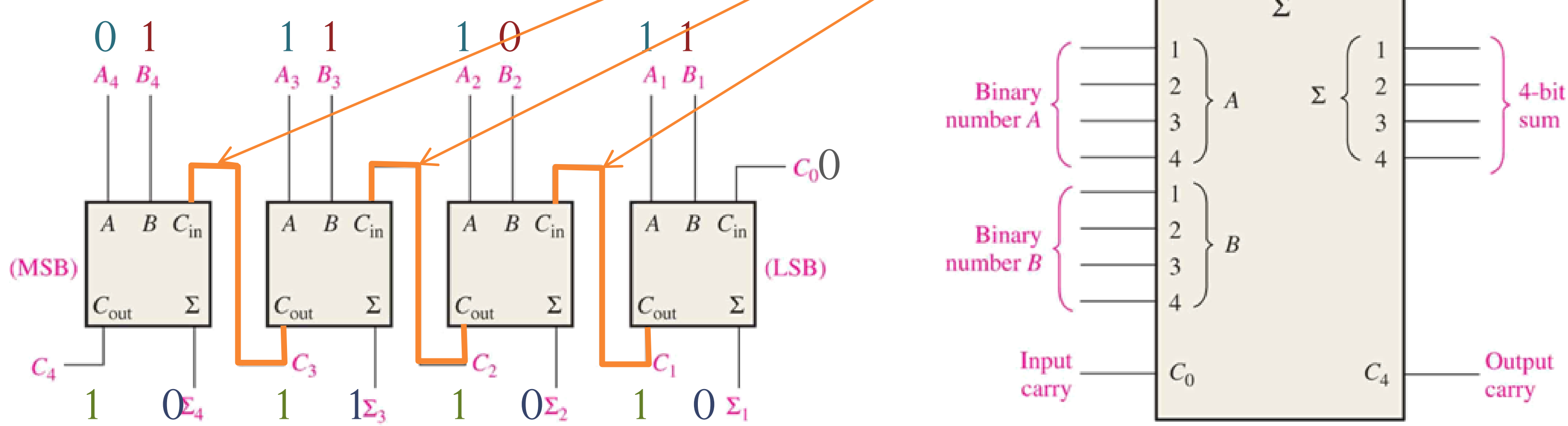
Recall: Binary Addition

- Example: Add the binary numbers 0111 and 1101 and show the equivalent decimal addition.

	1111	
A:	0111 +	7 +
B:	1101	13
	10100 =	20

The carry output of each adder is connected to the carry input of the next higher-order adder. These are called **internal carries**.

- 4-bit adder



(a) Block diagram

(b) Logic symbol

Recall: Binary Addition

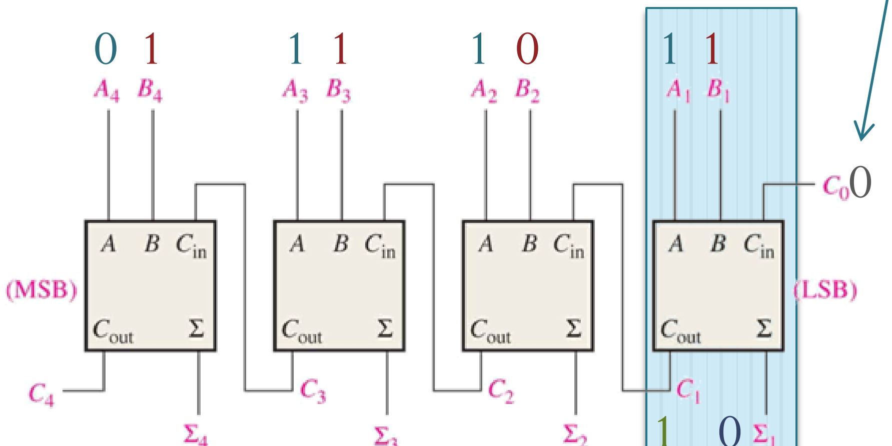
- Example: Add the binary numbers 0111 and 1101 and show the equivalent decimal addition.

Input Carry

	10
A:	0111 +
B:	1101
	0

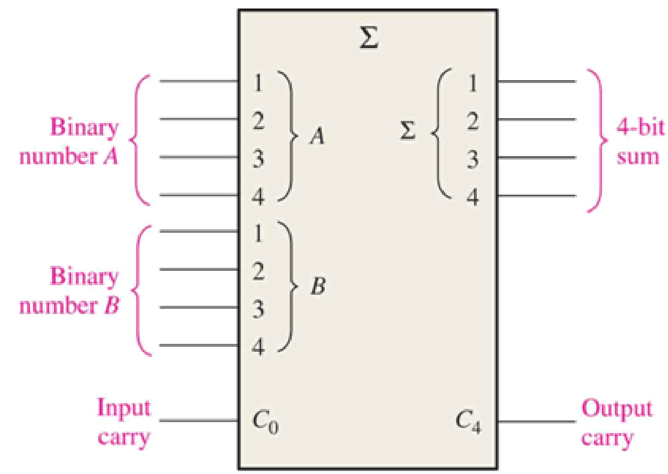
If there is no input carry to the LSB, then either a half-adder can be used or the carry input of a full-adder can be made 0 (grounded)

- 4-bit adder



(a) Block diagram

$1+1+0 = 2_{10} = 10_2$



(b) Logic symbol

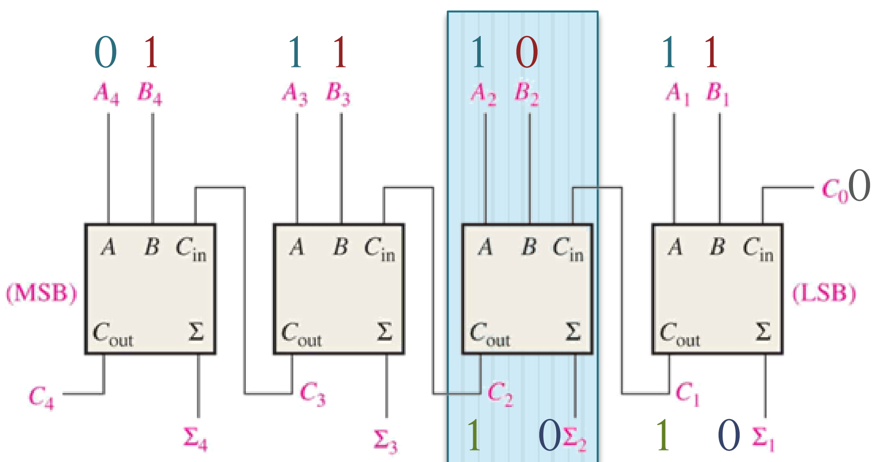
Recall: Binary Addition

- Example: Add the binary numbers 0111 and 1101 and show the equivalent decimal addition.

Carry bit from the right column

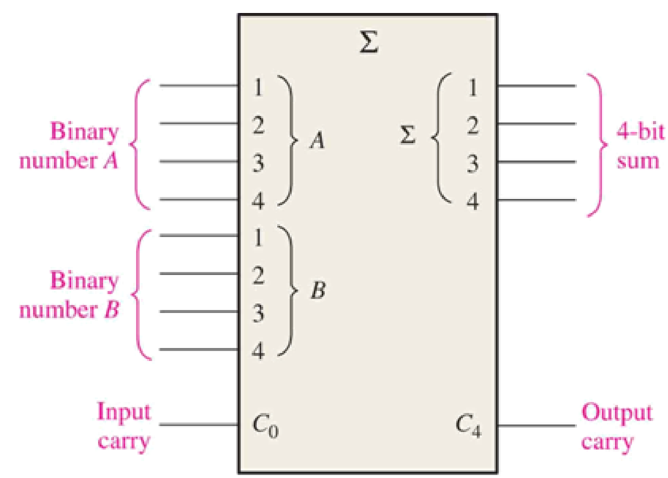
```
        1 1
A:    0 1 1 1   +
B:    1 1 0 1
    ---------
          0 0
```

- 4-bit adder



(a) Block diagram

$1+0+1 = 2_{10} = 10_2$



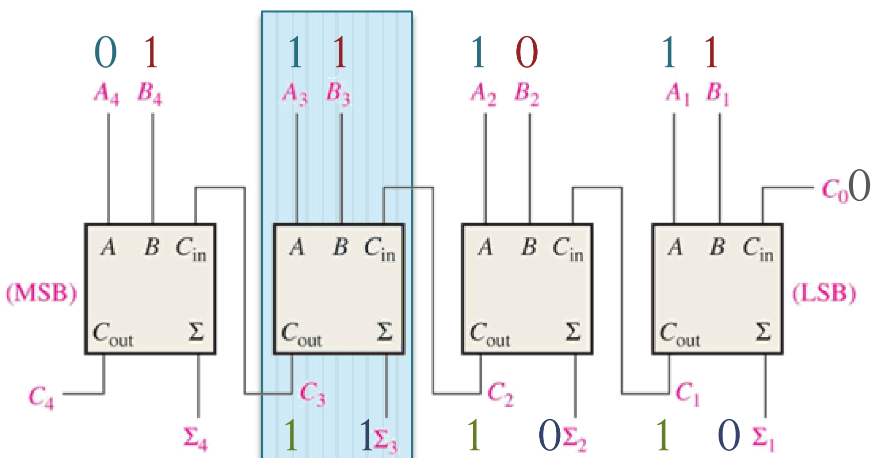
(b) Logic symbol

Recall: Binary Addition

- Example: Add the binary numbers 0111 and 1101 and show the equivalent decimal addition.

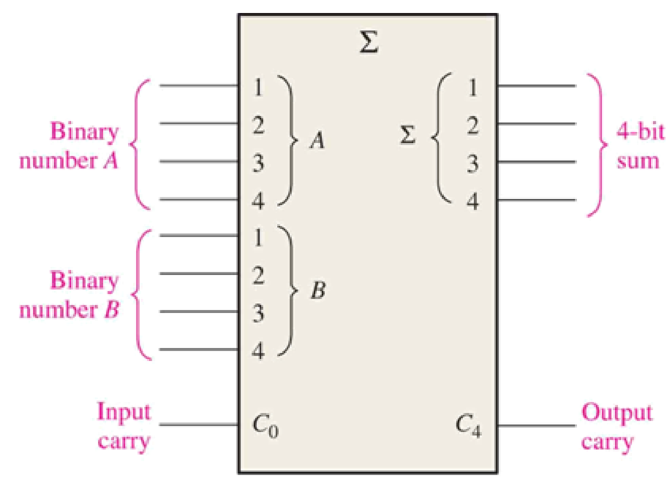
	111
A:	0111
B:	1101 +
	100

- 4-bit adder



(a) Block diagra[m]

$1+1+1 = 3_{10} = 11_2$



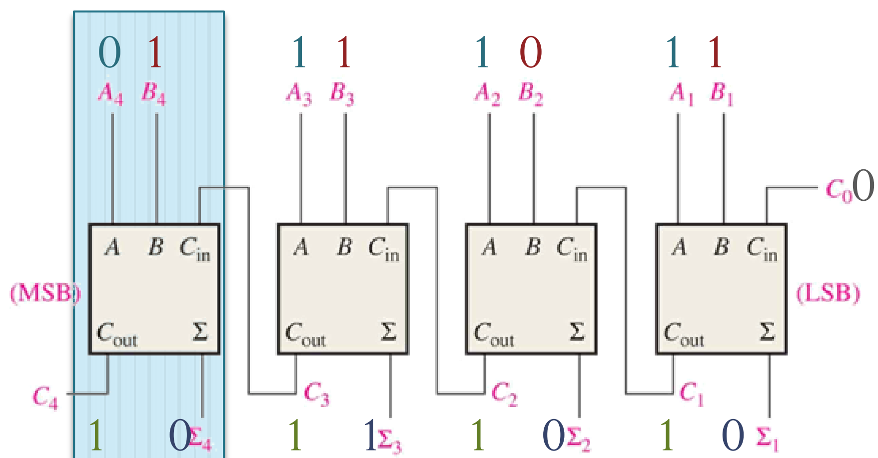
(b) Logic symbol

Recall: Binary Addition

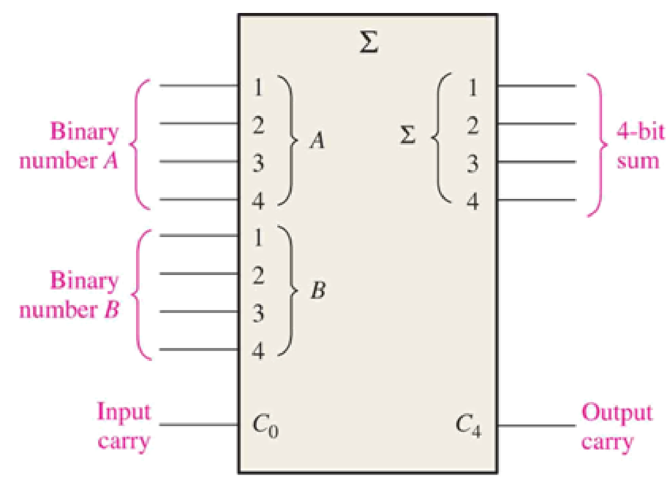
- Example: Add the binary numbers 0111 and 1101 and show the equivalent decimal addition.

$$
\begin{array}{lr}
 & 1111 \\
\text{A:} & 0111 \\
\text{B:} & +\ 1101 \\
\hline
 & 0100
\end{array}
$$

- 4-bit adder



$0+1+1 = 2_{10} = 10_2$



(b) Logic symbol

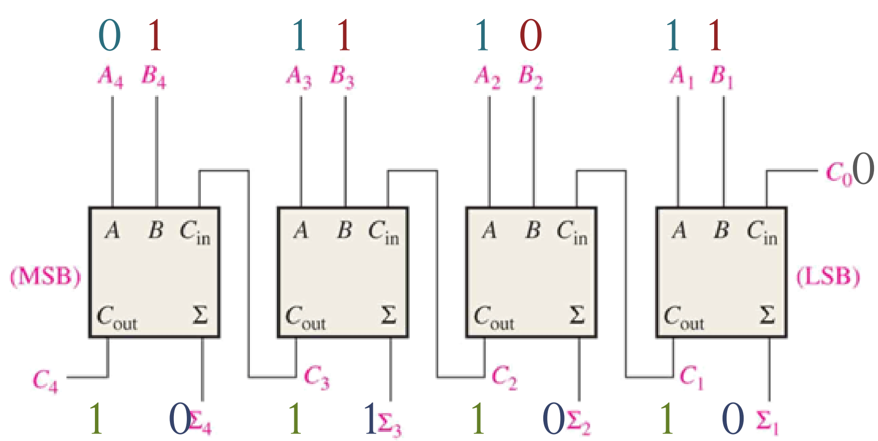
Recall: Binary Addition

- Example: Add the binary numbers 0111 and 1101 and show the equivalent decimal addition.

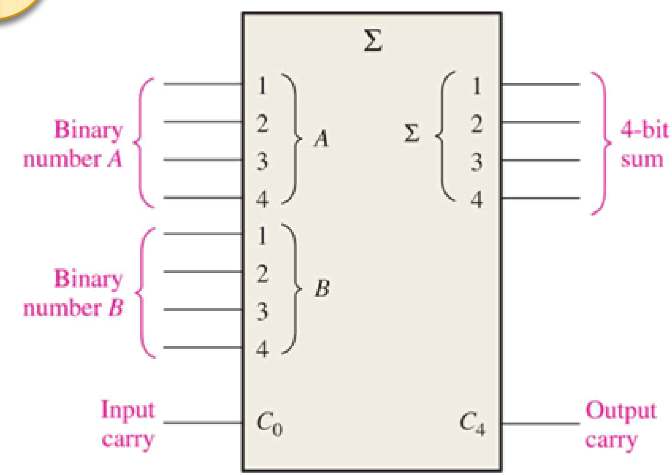
$$
\begin{array}{lrcr}
 & 1111 & & \\
A: & 0111 & + & 7 \\
B: & 1101 & + & 13 \\
\hline
 & 10100 & = & 20
\end{array}
$$

The output carry from the left-most full-adder becomes the MSB in the sum

- 4-bit adder



(a) Block diagram



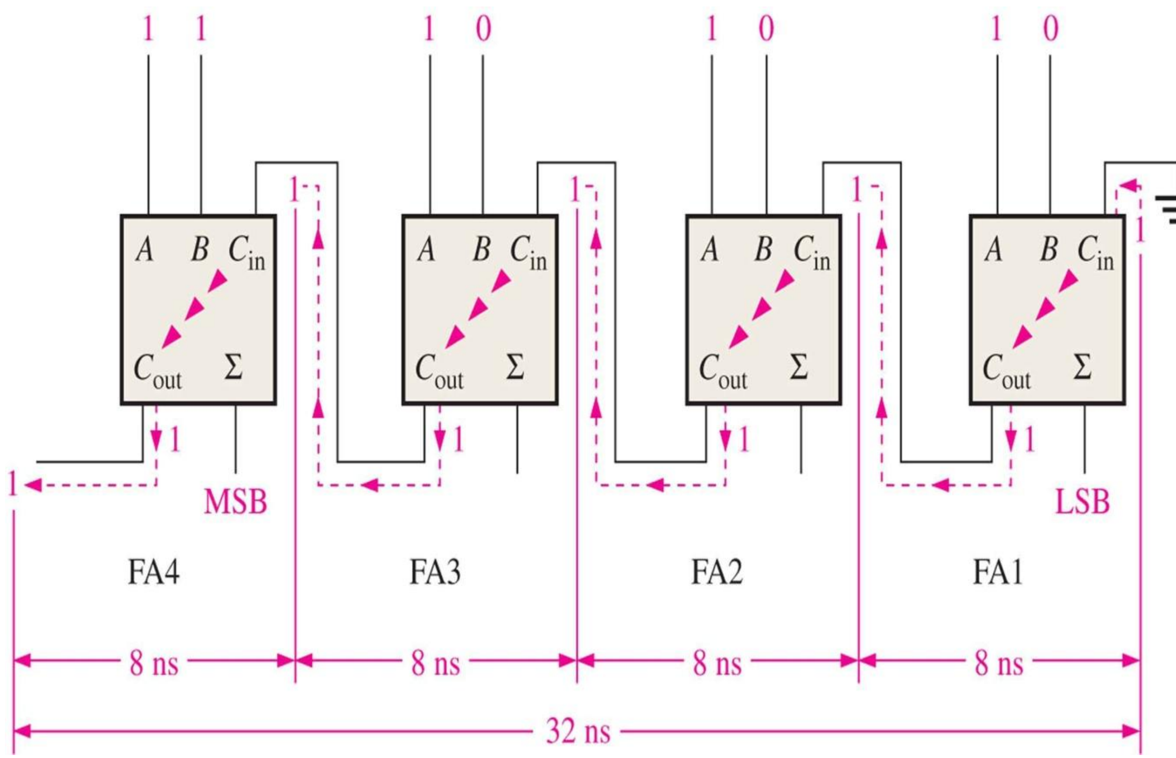
(b) Logic symbol

Parallel Adder

- Two categories (based on the way in which internal carries from stage to stage are handled)
  1. Ripple carry (The adder we have studied is a ripple-carry adder.)
  2. Look-ahead carry
- Externally, both types of adders are the same in terms of inputs and outputs.
- The difference is the **speed** at which they can add numbers.
  - The look-ahead carry adder is much faster than the ripple-carry adder.
- The speed with which an addition can be performed is limited by the time required for the carries to propagate, or ripple, through all the stages of a parallel adder.

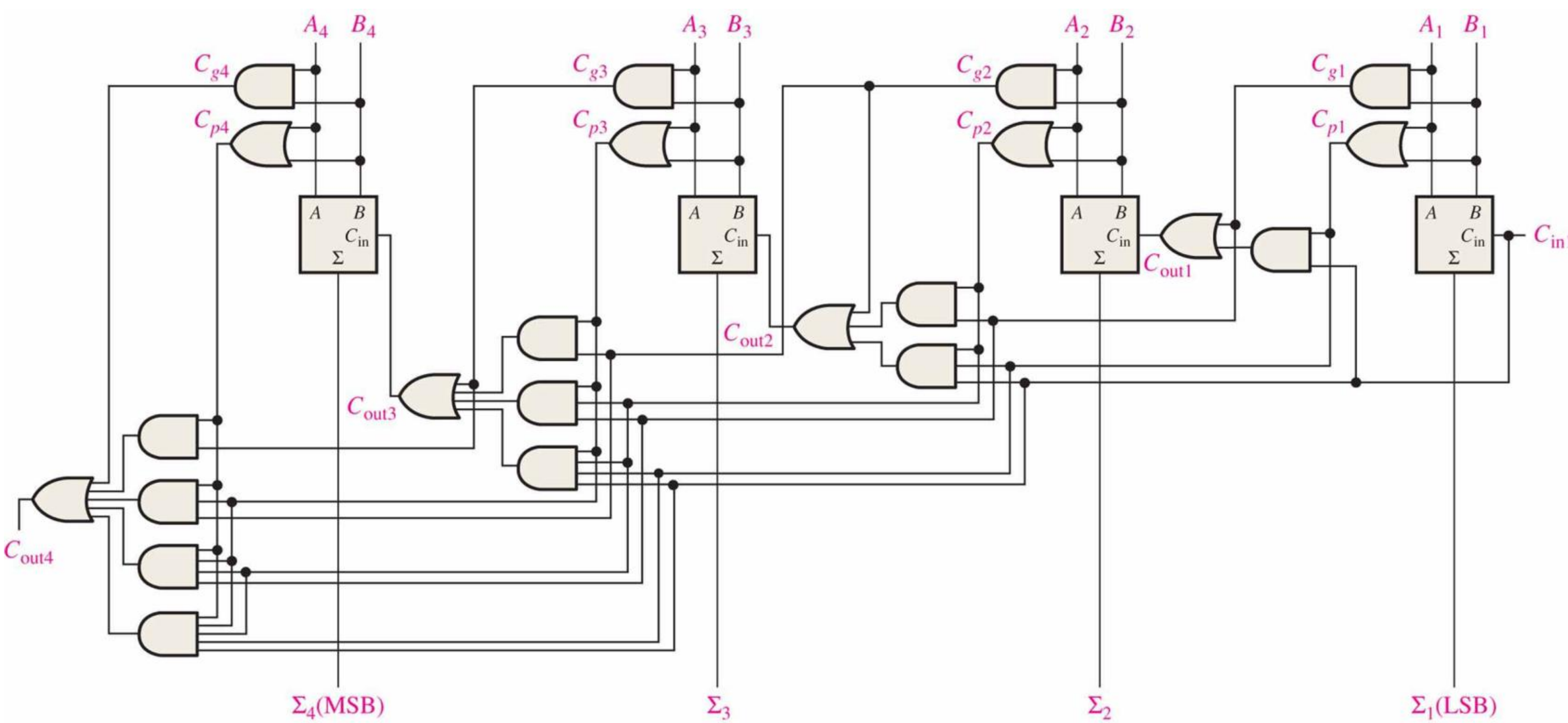
Ripple Carry Adder

- A **ripple carry adder** is one in which 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).
- Practical consideration: Real devices/gates have propagation time.
- The sum and the output carry of any stage cannot be produced until the input carry occurs.
- This causes a time delay in the addition process

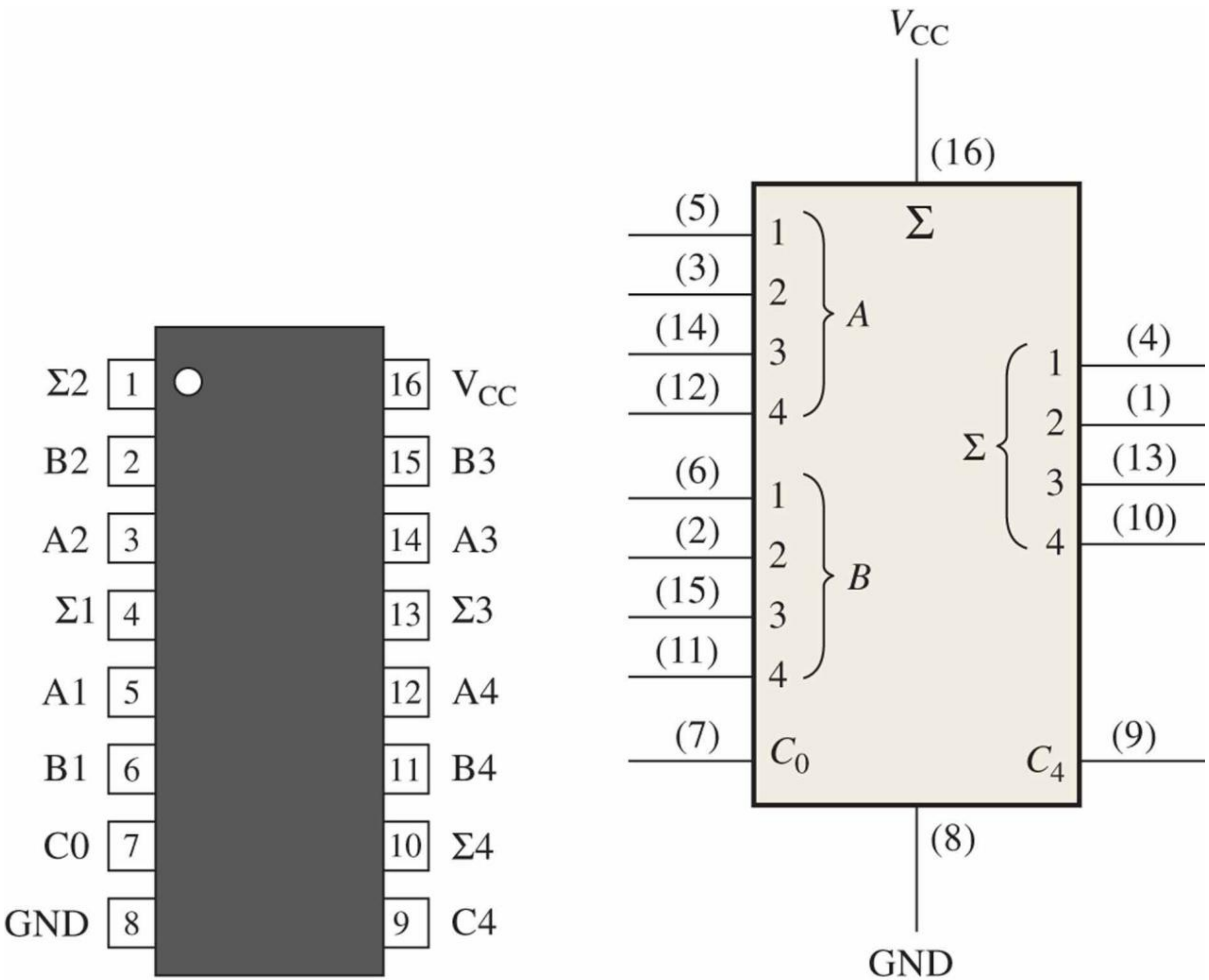


Look-Ahead Carry Adder

- Speedup the addition process by eliminating ripple carry delay.
- Anticipate the output carry of each stage.

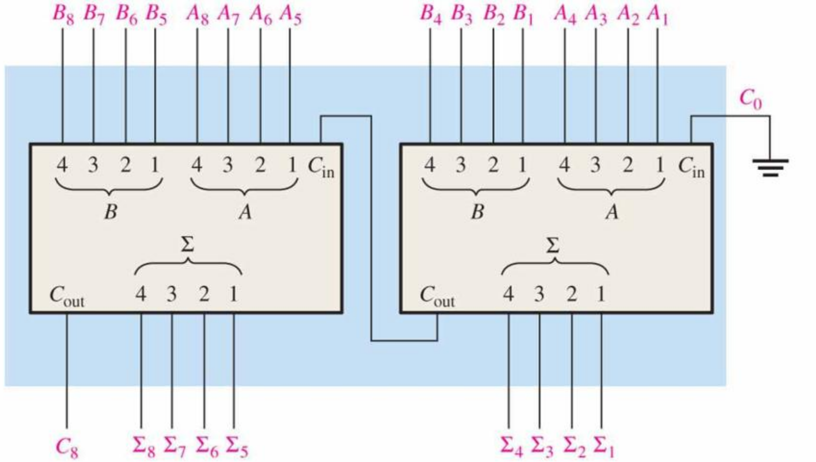


74x283: 4-bit Parallel Adder

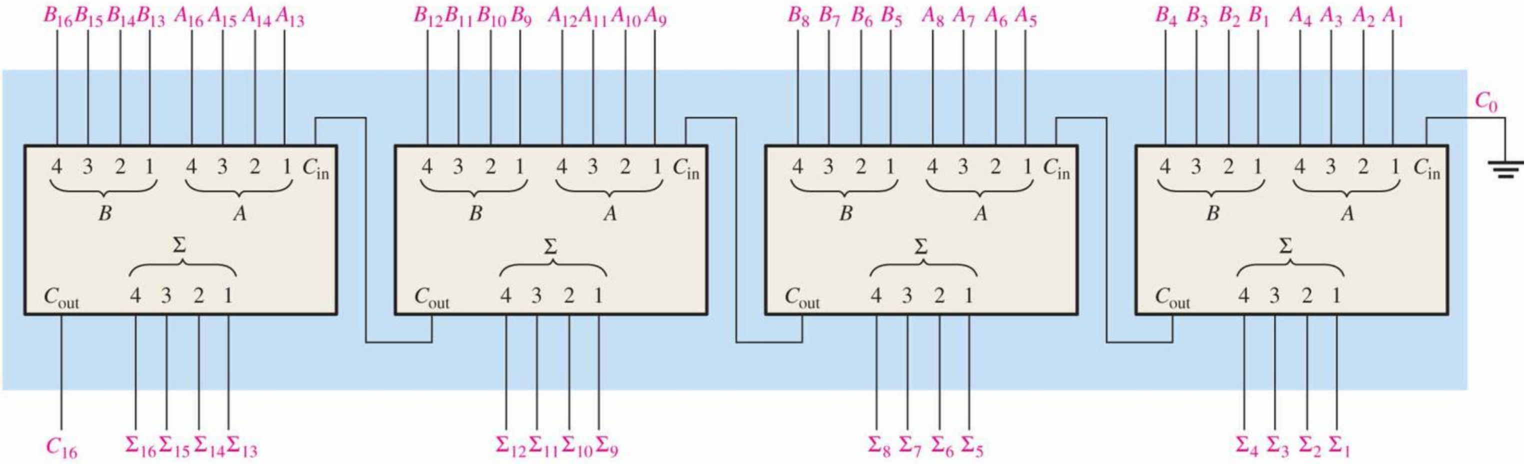


(a) Pin diagram of 74LS283 (b) 74LS283 logic symbol

Adder Expansion



(a) Cascading of two 4-bit adders to form an 8-bit adder



(b) Cascading of four 4-bit adders to form a 16-bit adder

Arithmetic Operations with Signed Numbers

- Using the signed number notation with negative numbers in 2's complement form simplifies addition and subtraction of signed numbers.
- Rules for **addition**: Add the two signed numbers (as if they are unsigned number). Discard any final carries. The result is in signed form.

Examples:

$$
\begin{array}{rl}
 & 00011110 = +30 \\
+ & 00001111 = +15 \\
\hline
 & 00101101 = +45
\end{array}
\qquad
\begin{array}{rl}
 & 00001110 = +14 \\
+ & 11101111 = -17 \\
\hline
 & 11111101 = -3
\end{array}
\qquad
\begin{array}{rl}
 & 11111111 = -1 \\
+ & 11111000 = -8 \\
\hline
\cancel{1} & 11110111 = -9
\end{array}
$$

Discard carry

Error (Overflow)

- Note that if the number of bits required for the answer is exceeded, error will occur. This occurs only if both numbers have the same sign.
- The error will be indicated by an incorrect sign bit.
- Some textbooks use the word "overflow" to denote this error.

$$
\begin{array}{l}
01000000 = +128 \\
\underline{01000001 = +129} \\
10000001 = \cancel{-126}
\end{array}
$$

Discard carry

$$
\begin{array}{l}
10000001 = -127 \\
\underline{10000001 = -127} \\
\cancel{1}00000010 = \cancel{+2}
\end{array}
$$

Wrong! The answer is incorrect and the sign bit has changed.

Some textbooks denote this case by "underflow".

Adding two positive numbers produces an overflow if the sign of the result is negative.

Adding two negative numbers produces an underflow if the sign of the result is positive

Subtraction

- Rules for **subtraction**: 2's complement the subtrahend and add the numbers. Discard any final carries. The result is in signed form.

Example: Repeat the examples done previously, but subtract:

$$\begin{array}{rl} & 00011110 \quad (+30) \\ - & \underline{00001111} \quad -(+15) \end{array} \qquad \begin{array}{rl} & 00001110 \quad (+14) \\ - & \underline{11101111} \quad -(-17) \end{array} \qquad \begin{array}{rl} & 11111111 \quad (-1) \\ - & \underline{11111000} \quad -(-8) \end{array}$$

2's complement subtrahend and add:

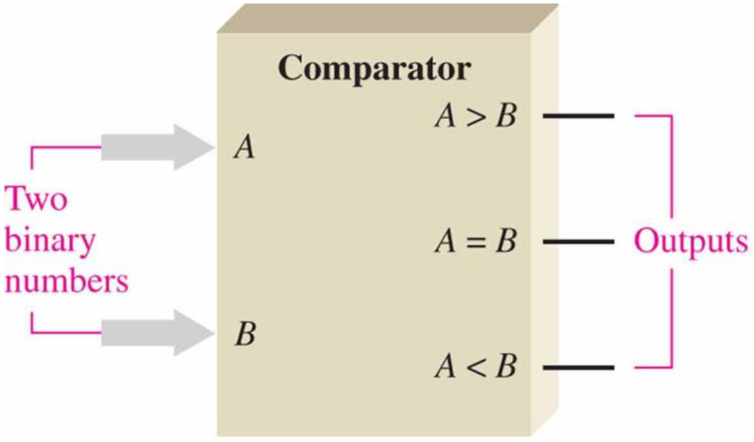
$$\begin{array}{r} 00011110 = +30 \\ \underline{11110001 = -15} \\ \cancel{1}00001111 = +15 \end{array} \qquad \begin{array}{r} 00001110 = +14 \\ \underline{00010001 = +17} \\ 00011111 = +31 \end{array} \qquad \begin{array}{r} 11111111 = -1 \\ \underline{00001000 = +8} \\ \cancel{1}00000111 = +7 \end{array}$$

Discard carry

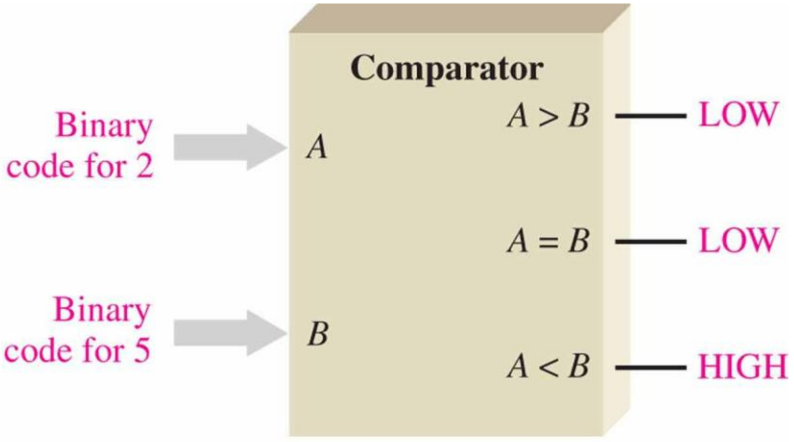
Discard carry

Comparator

- A comparator compares two quantities and indicates whether or not they are equal.

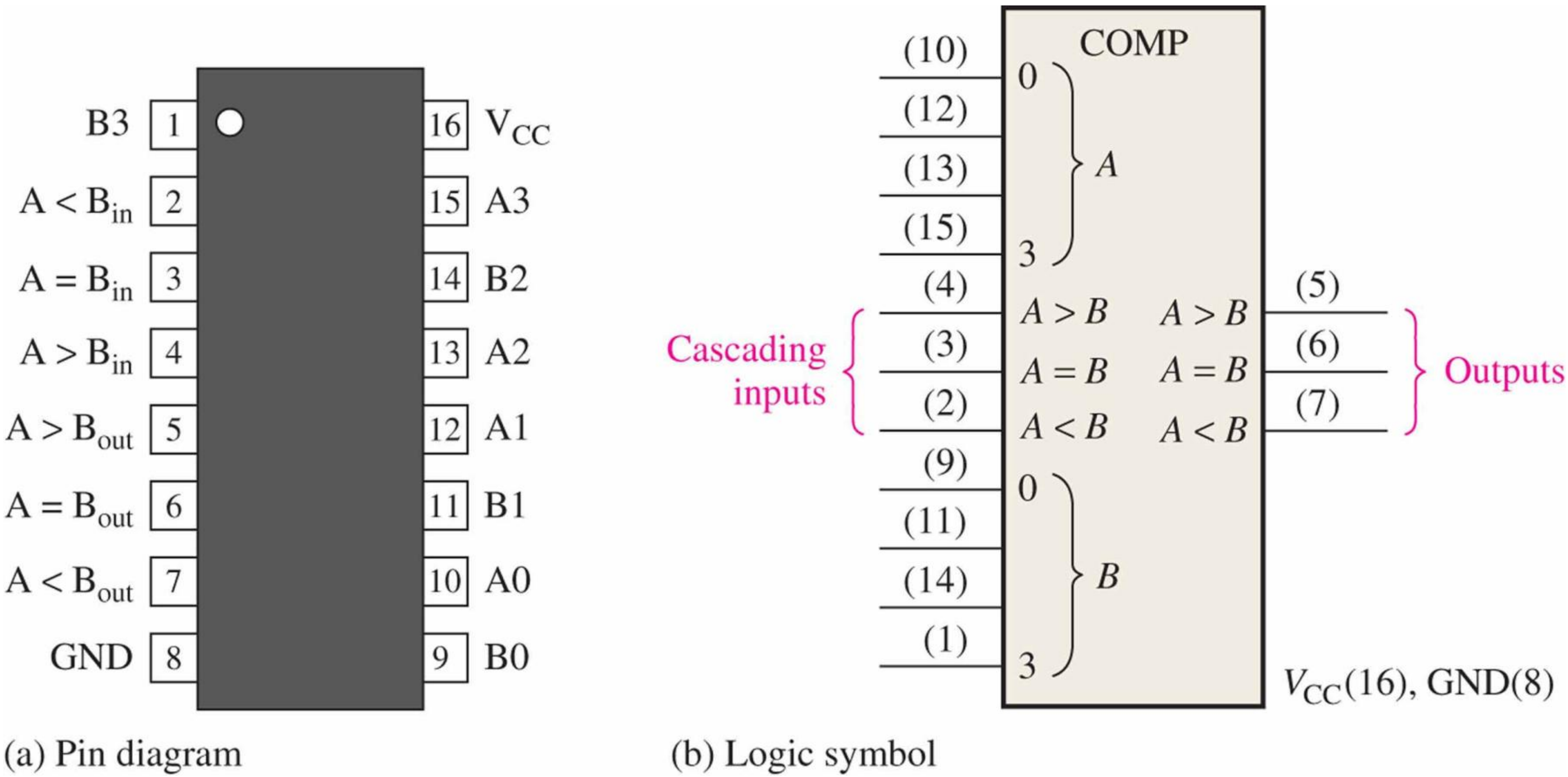


(a) Basic magnitude comparator



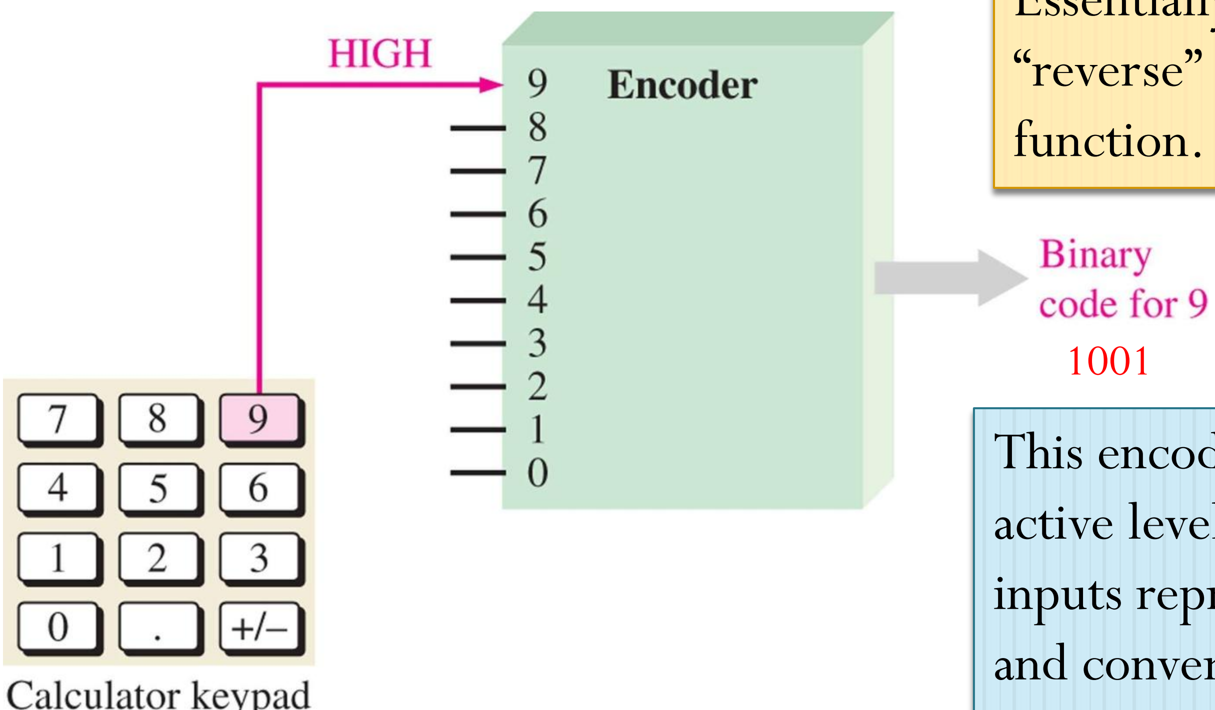
(b) Example: A is less than B ($2 < 5$) as indicated by the HIGH output ($A < B$)

74x85: 4-bit Magnitude Comparator



Encoder

In general, the encoder converts information, such as a decimal number or an alphabetic character, into some coded form.



Essentially, it performs a "reverse" decoder function.

This encoder accepts an active level on one of its inputs representing a digit, and converts it to a binary output.

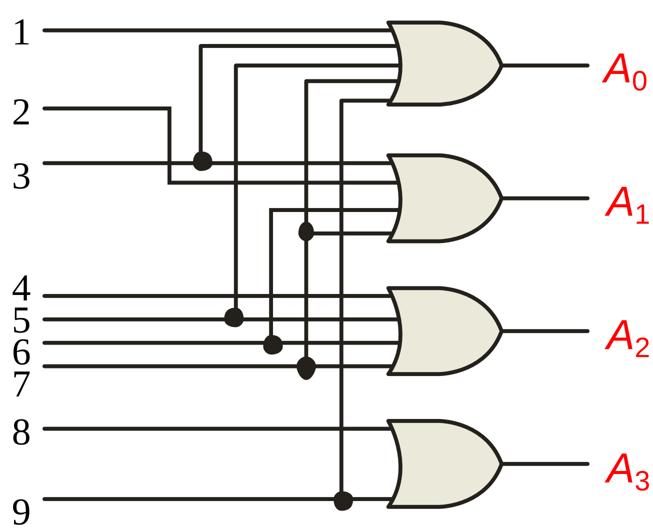
BCD

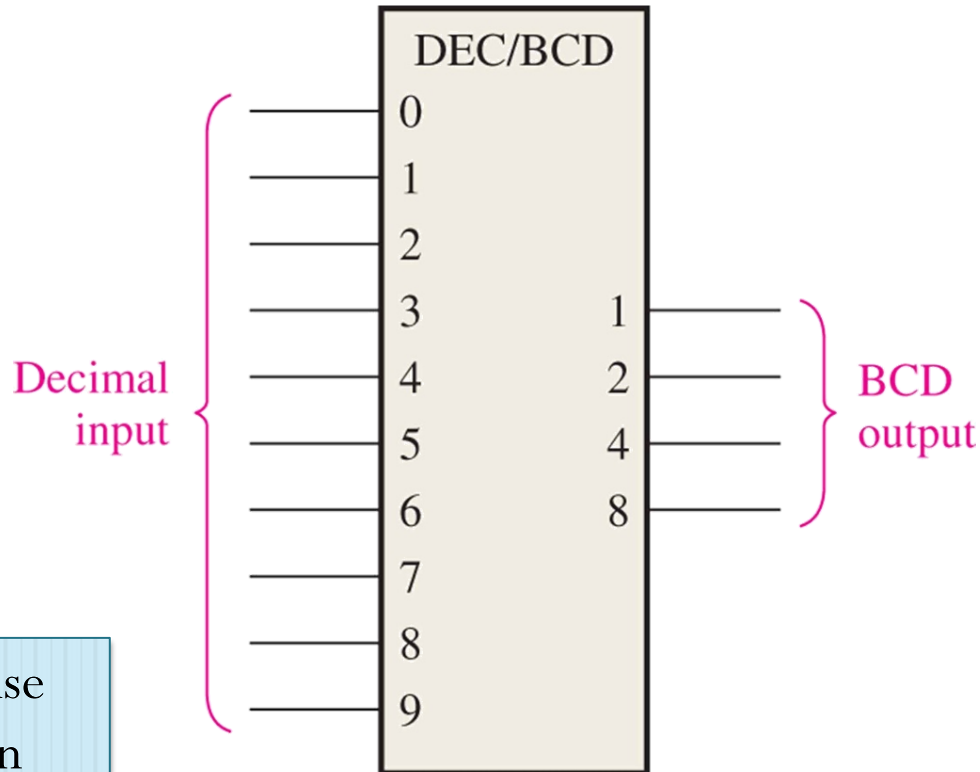
- Binary coded decimal (BCD) is a weighted code that is commonly used in digital systems when it is necessary to show decimal numbers such as in clock displays.
- Express each of the decimal digits with a binary code.

Decimal	Binary	BCD
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	00010000
11	1011	00010001
12	1100	00010010
13	1101	00010011
14	1110	00010100
15	1111	00010101

Decimal-to-BCD Encoder

The decimal-to-BCD is an encoder with an input for each of the ten decimal digits and four outputs that represent the BCD code for the active digit.

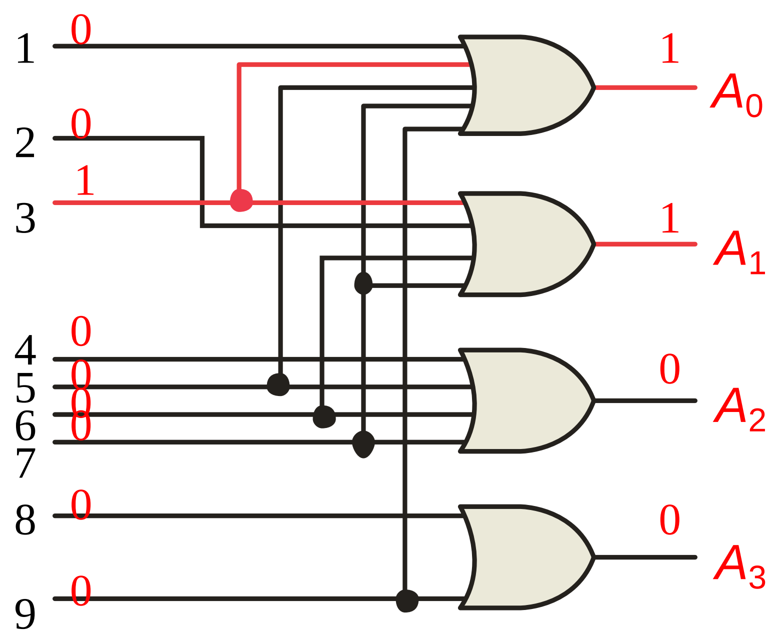




A 0-digit input is not needed because the BCD outputs are all LOW when there are no HIGH inputs.

Example

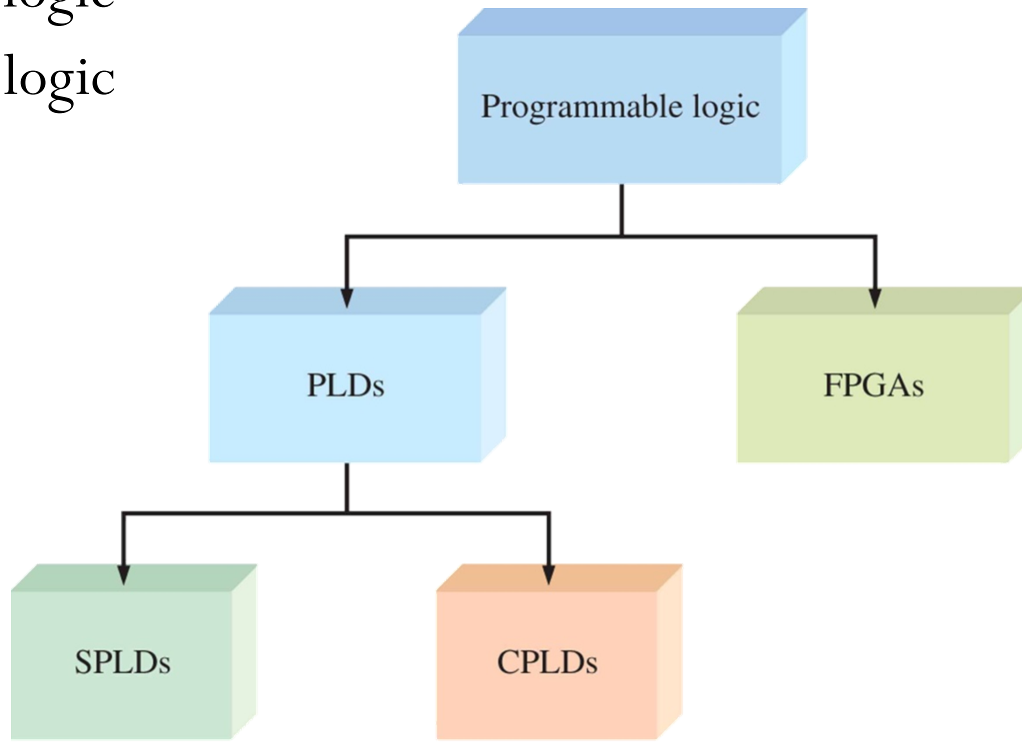
- Q: Show how the decimal-to-BCD encoder converts the decimal number 3 into a BCD 0011.
- A: The top two OR gates have ones as indicated with the red lines. Thus the output is 0011.



Programmable Logic Devices

- There are two broad categories of digital ICs.
  1. Fixed-function logic
  2. Programmable logic

We've already talked about many of these



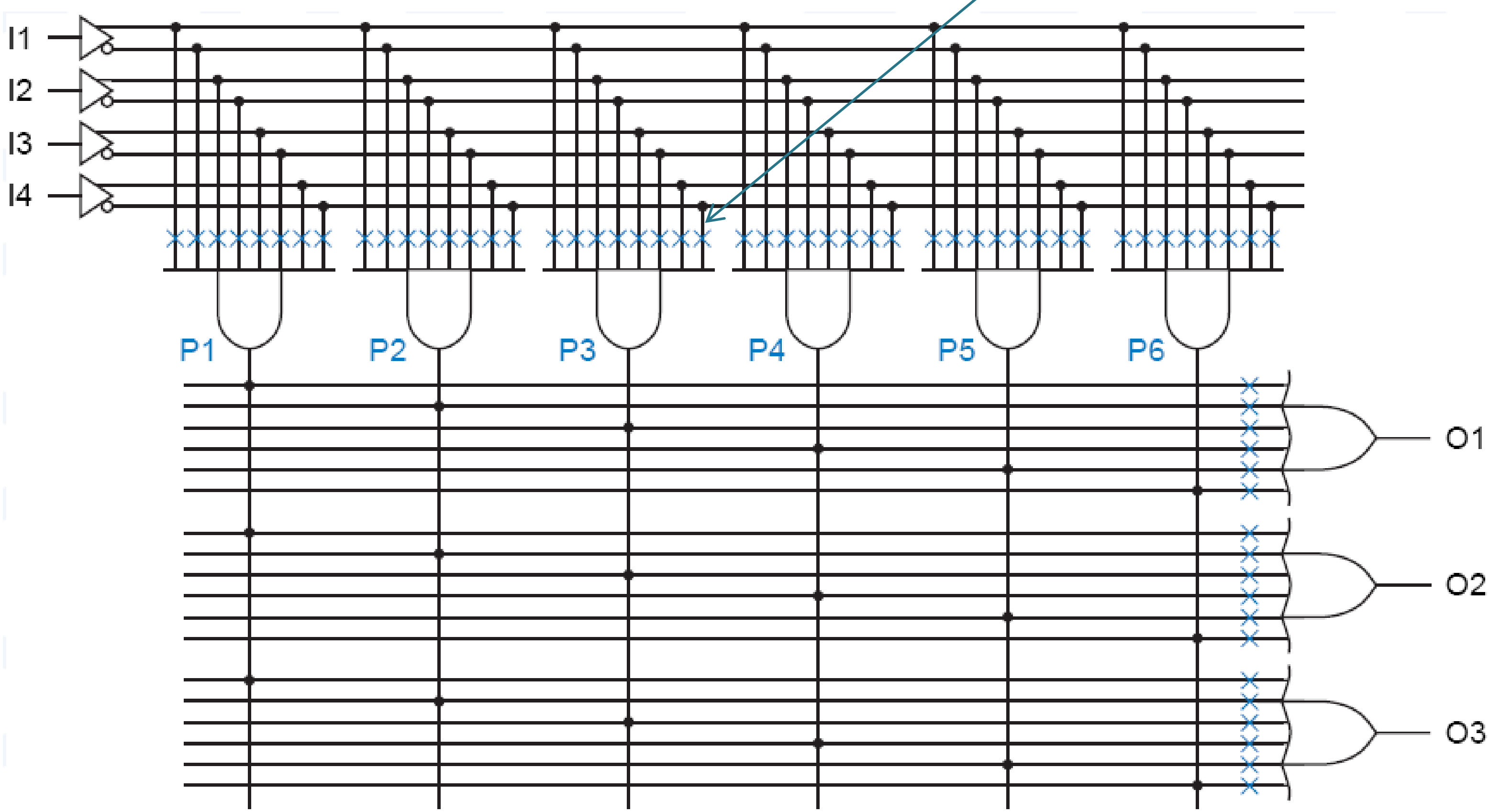
PLD (Programmable Logic Device)

- Historically, the first PLDs were programmable logic arrays (PLAs)
- A PLA is a combinational, two-level AND-OR device that can be programmed to realize any SOP logic expression.
  - Hence, it can also be used to implement minimal sum.
- Most PLDs also have a programmable inverter/buffer at the output of the AND-OR array.
  - Hence, it can also be used to implemented POS expression and minimal product.

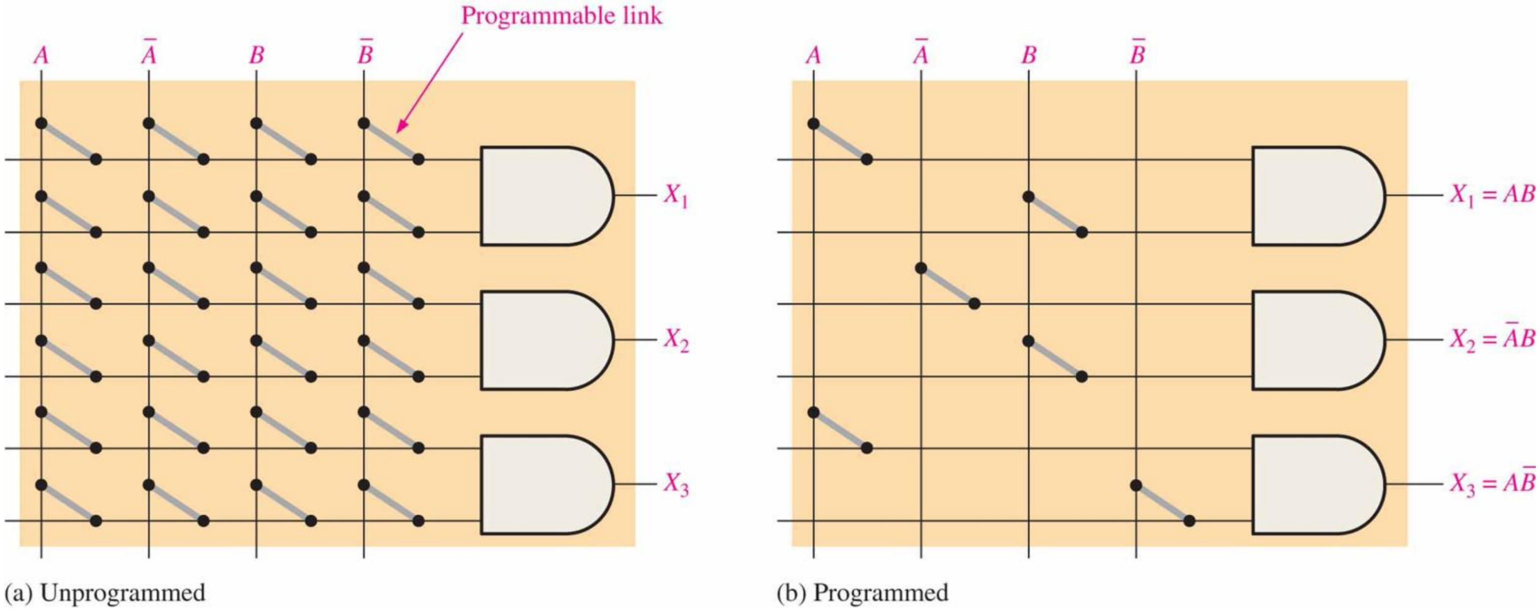
PLA

A 4 × 3 PLA with six product terms

Potential connections in the array are indicated by X's; the device is programmed by establishing only the connections that are actually needed.



Programmable link in PLDs



ECS371 Exam

- NOT to torture you.
- It's an opportunity for you to demonstrate what you have learned from this course.
- Aim for partial credit! If you know something, write that down.

Some Important Corrections

- These typos in the notes have already been corrected in class.
- However, for those who skipped class, here are some important ones:
  - Distributive law:
    - A+BC = (A+B)(A+C)
    - A(B+C) = AB+AC
    - Caution: AB + CD = (AB+C)(AB+D) = (A+CD)(B+CD)
      - **NOT** the same as ABC + ABD
  - K-Map for 4 variables