Digital Circuits

ECS 371

Dr. Prapun Suksompong

prapun@siit.tu.ac.th

Lecture 2-3-4

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

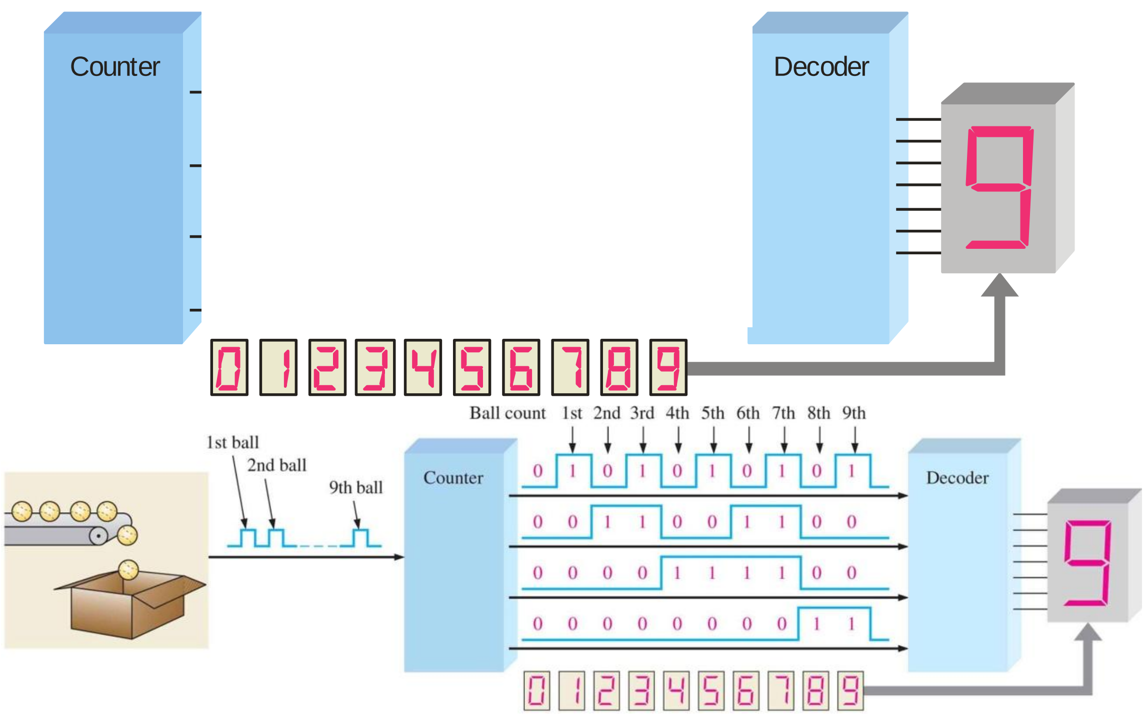
ECS371.PRAPUN.COM

Binary Counting

A binary counting sequence for numbers from zero to fifteen is shown.

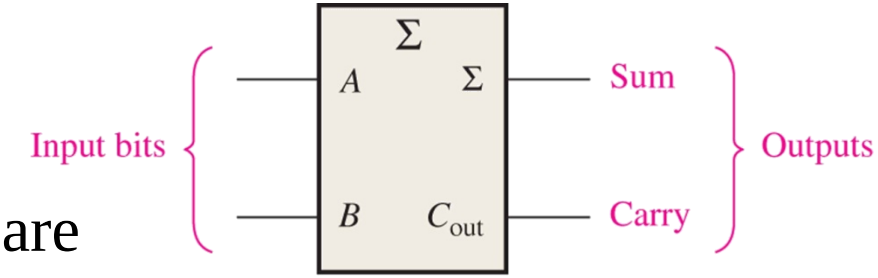
Notice the pattern of zeros and ones in each column.

Digital counters frequently have this same pattern of digits:



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
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

Binary Addition

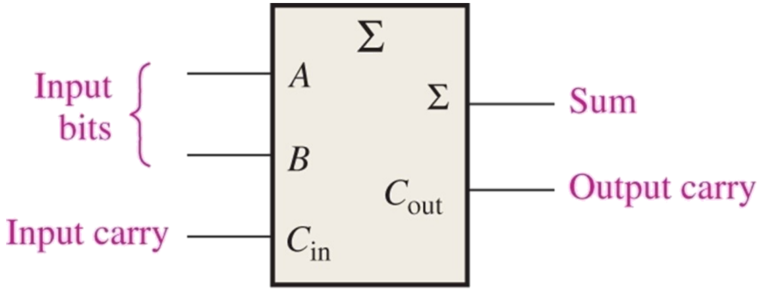


The rules for binary addition are

0 + 0 = 0	Sum = 0, carry = 0
0 + 1 = 1	Sum = 1, carry = 0
1 + 0 = 1	Sum = 1, carry = 0
1 + 1 = 10	Sum = 0, carry = 1

When an input carry = 1 due to a previous result, the rules are

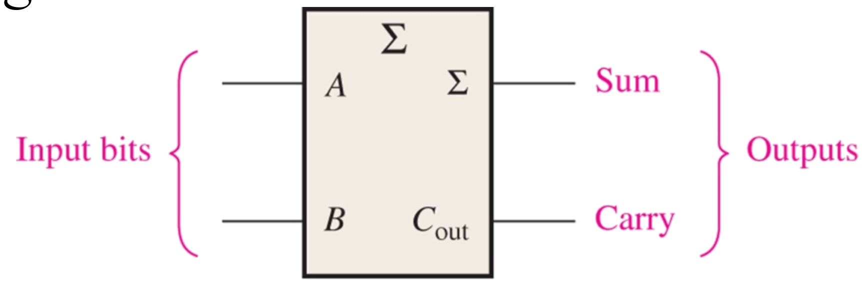
1 + 0 + 0 = 01	Sum = 1, carry = 0
1 + 0 + 1 = 10	Sum = 0, carry = 1
1 + 1 + 0 = 10	Sum = 0, carry = 1
1 + 1 + 1 = 11	Sum = 1, carry = 1

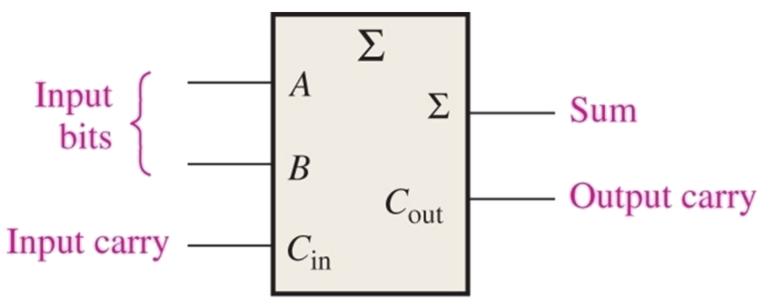


Truth Table

- **Truth table**: A table showing the inputs and corresponding output(s) of a logic circuit.

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





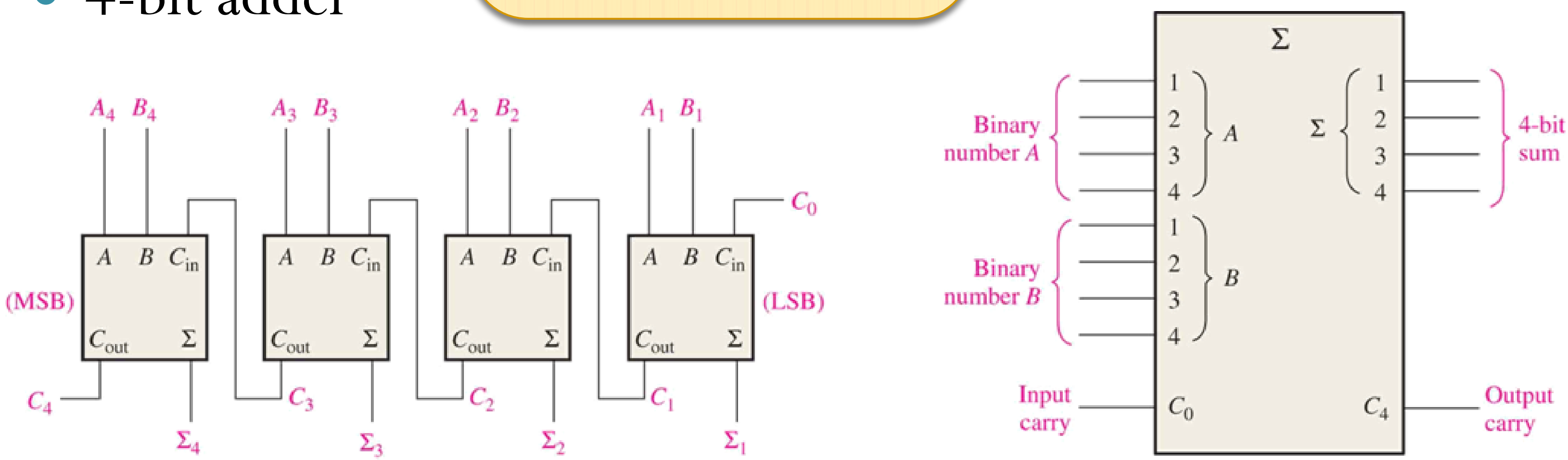
Inputs			Outputs	
A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Binary Addition (Con't)

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

```
 1111
  0111      7
+ 1101   + 13
------   ----
 10100  =  20
```

- 4-bit adder



(a) Block diagram (b) Logic symbol

Representation of Negative Numbers

- Digital Logic represents numbers as n-bit binary numbers, with fixed n.
- Some important operations:
  1. **1's complement**: Change all 1s to 0s and all 0s to 1s.
  2. **2's complement**: Add 1 to the LSB of the 1's complement.
     - If the addition produces a result that requires more than n digits, we throw away the extra digit(s).
  - If a number D is complemented twice, the result is D.
- An alternative method of finding the 2's complement: Change all bits to the left of the least significant 1.
  1. Start at the right with the LSB and write the bits as they are up to and including the first 1.
  2. Take the 1's complements of the remaining bits.

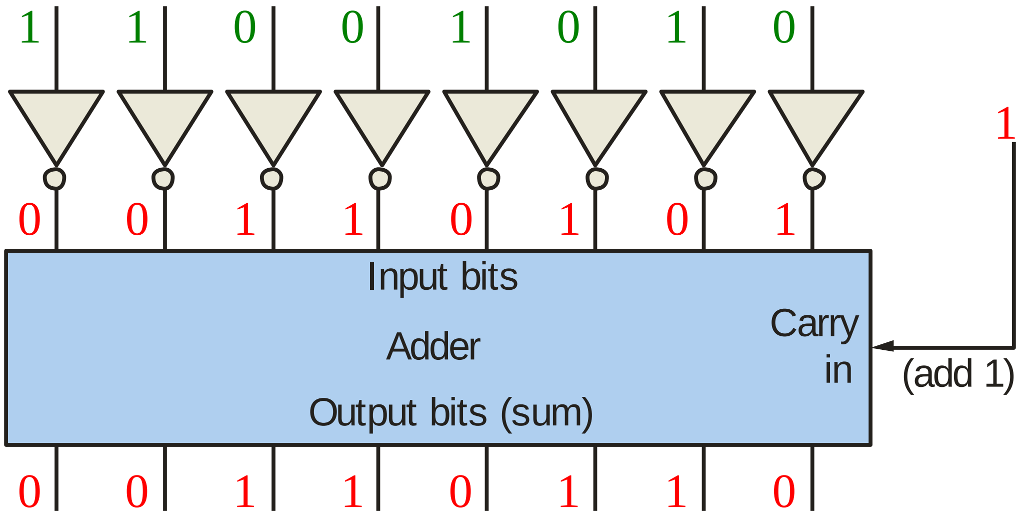
Example

The 1's complement of 11001010 is

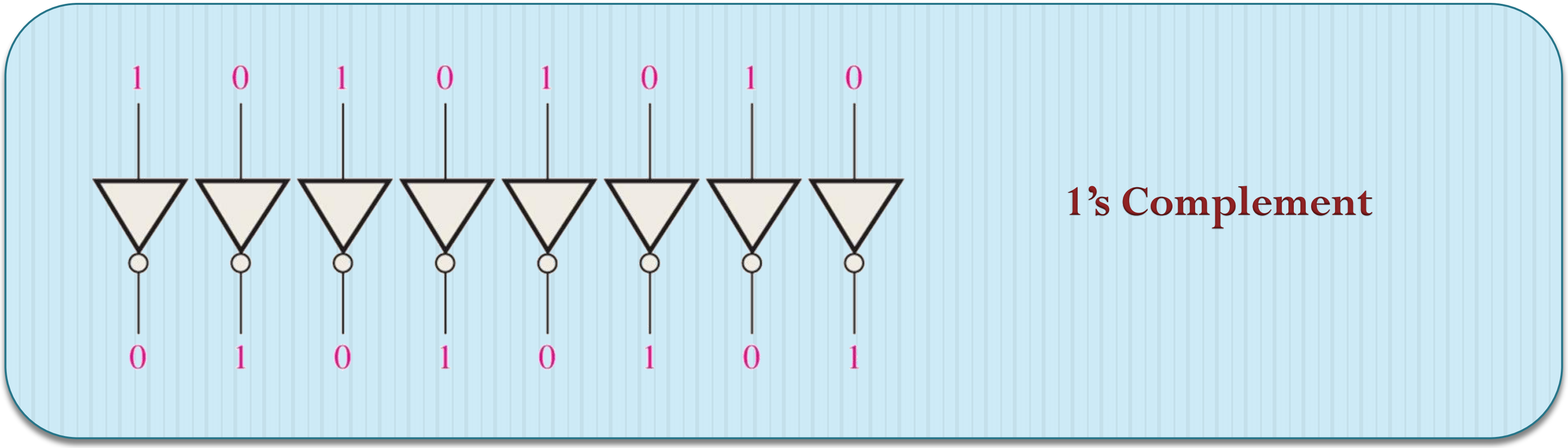
00110101 (1's complement)

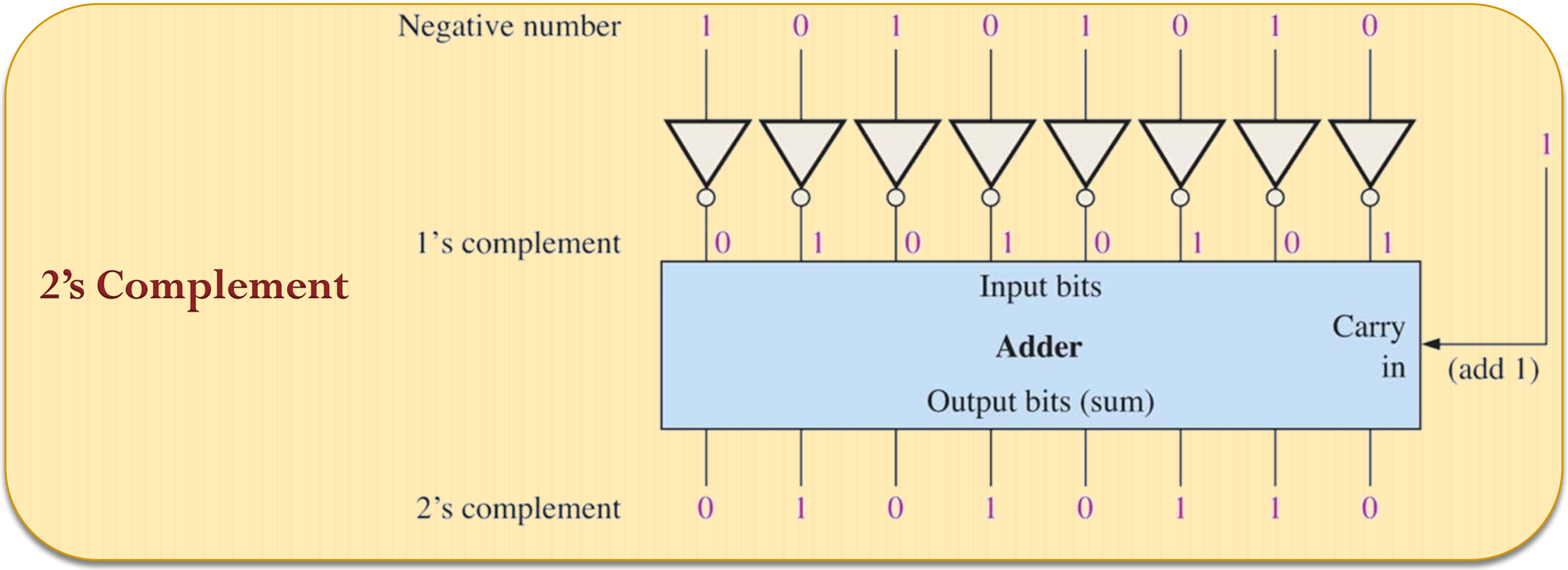
To form the 2's complement, add 1: +1

00110110 (2's complement)



Implementation





Signed Binary Number

- Fix the number of bits.
- A signed binary number consists of both sign and magnitude information.
  - The **sign** indicates whether a number is positive or negative
    - In a signed binary number, the left-most bit (MSB) is the **sign bit**.
    - 0 indicates a positive number, and
      1 indicates a negative number
  - The **magnitude** is the value of the number.
- There are three forms in which signed integer (whole) numbers can be represented in binary:
  1. sign-magnitude,
  2. l's complement,
  3. and 2's complement.
- Of these, the 2's complement is the most important

Signed Binary Number

(1) Sign-Magnitude Form

- The magnitude bits are in true (uncomplemented) binary for both positive and negative numbers.
- Negate a number by changing its sign.

(2) 1's Complement Form

- A negative number is the 1's complement of the corresponding positive number.

There are two possible representations of zero, "+0" and "-0", but both have the same value.

	Sign-Magnitude	1's Complement	2's Complement
000	0	0	0
001	1	1	1
010	2	2	2
011	3	3	3
100	-0	-3	-4
101	-1	-2	-3
110	-2	-1	-2
111	-3	-0	-1

Signed Binary Number (2)

(3) 2's Complement Form

- A negative number is the 2's complement of the corresponding positive number.
- Has only one representation of zero.
- Zero is considered positive because its sign bit is 0.
- The weight of the sign bit is given a negative value.
- Decimal values are determined by summing the weights in all bit positions where there are 1s and ignoring those positions where there are zeros.

	Sign-Magnitude	1's Complement	2's Complement
000	0	0	0
001	1	1	1
010	2	2	2
011	3	3	3
100	-0	-3	-4
101	-1	-2	-3
110	-2	-1	-2
111	-3	-0	-1

Example

The positive number 58 is written using 8-bits as 00111010 (true form).

Sign bit   Magnitude bits

The negative number −58 is written as:

−58 = 11000110 (complement form)

Sign bit   Magnitude bits

An easy way to read a signed number that uses this notation is to assign the sign bit a negative weight (−128 for an 8-bit number). Then add the column weights for the 1's.

Weights:	−128	64	32	16	8	4	2	1.	
	1	1	0	0	0	1	1	0	
	−128	+64				+4	+2		= −58

2's Complement (con't)

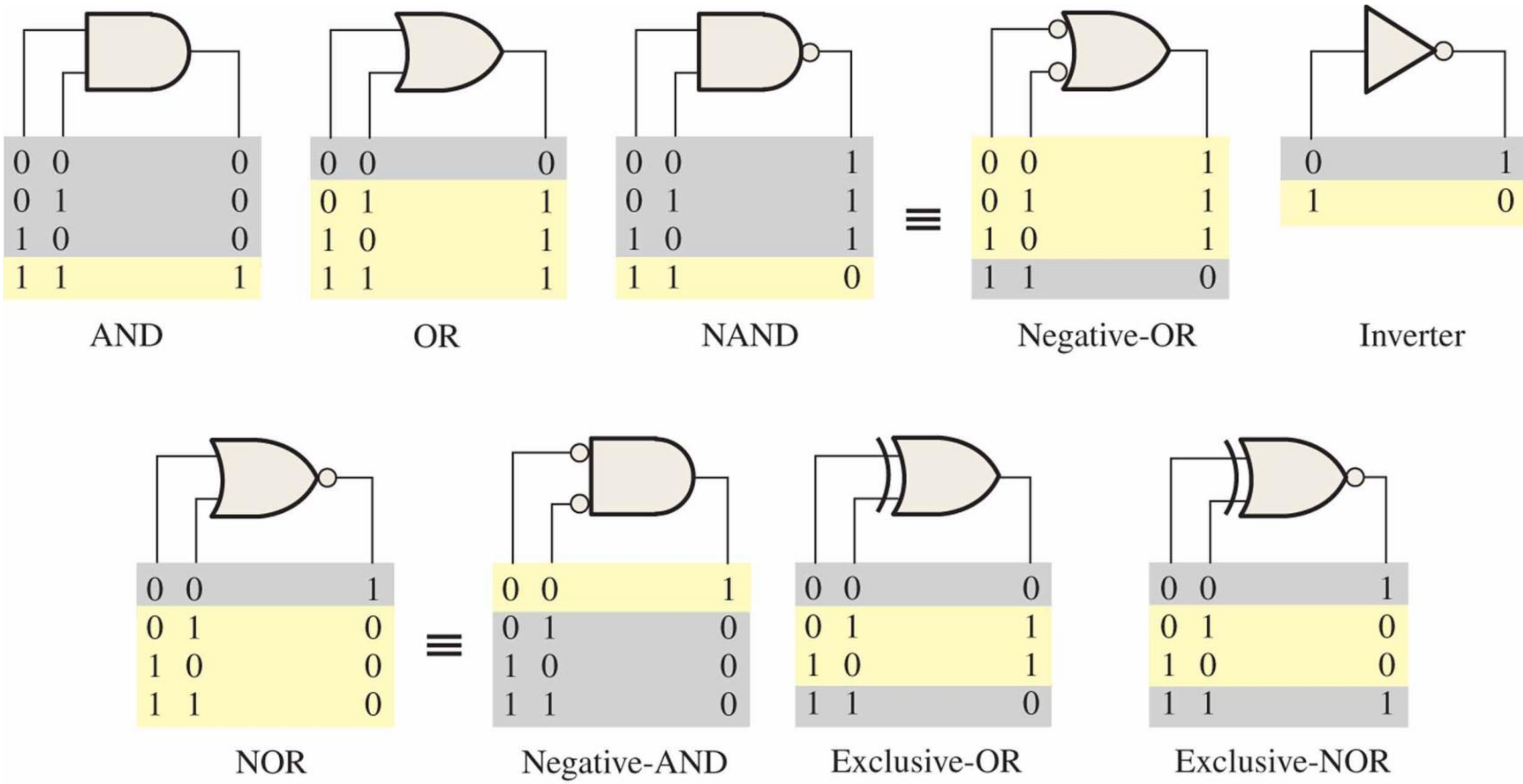
- The number of different combinations of n bits is 2^n
- For n bit 2's complement signed numbers, the range is

$$-\left(2^{n-1}\right) \text{ to } +\left(2^{n-1}-1\right)$$

- Has one extra negative number
  - This number does not have a positive counterpart.
- To convert n-bit 2's complement number into m-bit one:
  - If $m > n$, append m-n copies of the sign bit.
    - This is called ***sign extension***.
  - If $m < n$, discard n-m leftmost bits
    - The result is valid only if all of the discarded bits are the same as the sign bit of the result.

	2's Complement
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Logic Gates



AND

0	0	0
0	1	0
1	0	0
1	1	1

OR

0	0	0
0	1	1
1	0	1
1	1	1

NAND

0	0	1
0	1	1
1	0	1
1	1	0

Negative-OR

0	0	1
0	1	1
1	0	1
1	1	0

Inverter

0	1
1	0

NOR

0	0	1
0	1	0
1	0	0
1	1	0

Negative-AND

0	0	1
0	1	0
1	0	0
1	1	0

Exclusive-OR

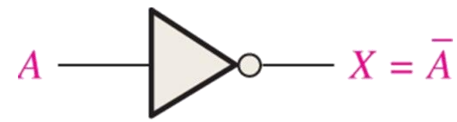
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR

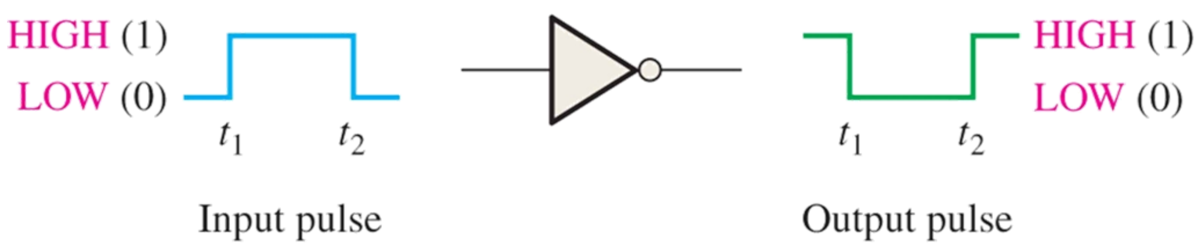
0	0	1
0	1	0
1	0	0
1	1	1

Note: Active states are shown in yellow.

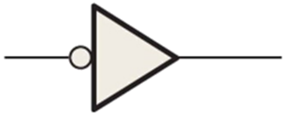
NOT Gate



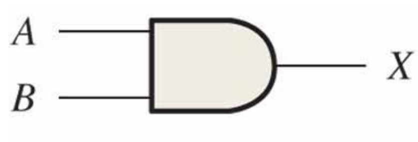
- The **inverter** (NOT circuit) performs the operation called **inversion** or **complementation**.
  - **Complement** :The inverse or opposite of a number.
    - LOW is the complement of HIGH
    - 0 is the complement of 1.



- The **negation indicator** is a "bubble" (o) that indicates inversion or complementation
  - Later, we will "play" with this bubble.

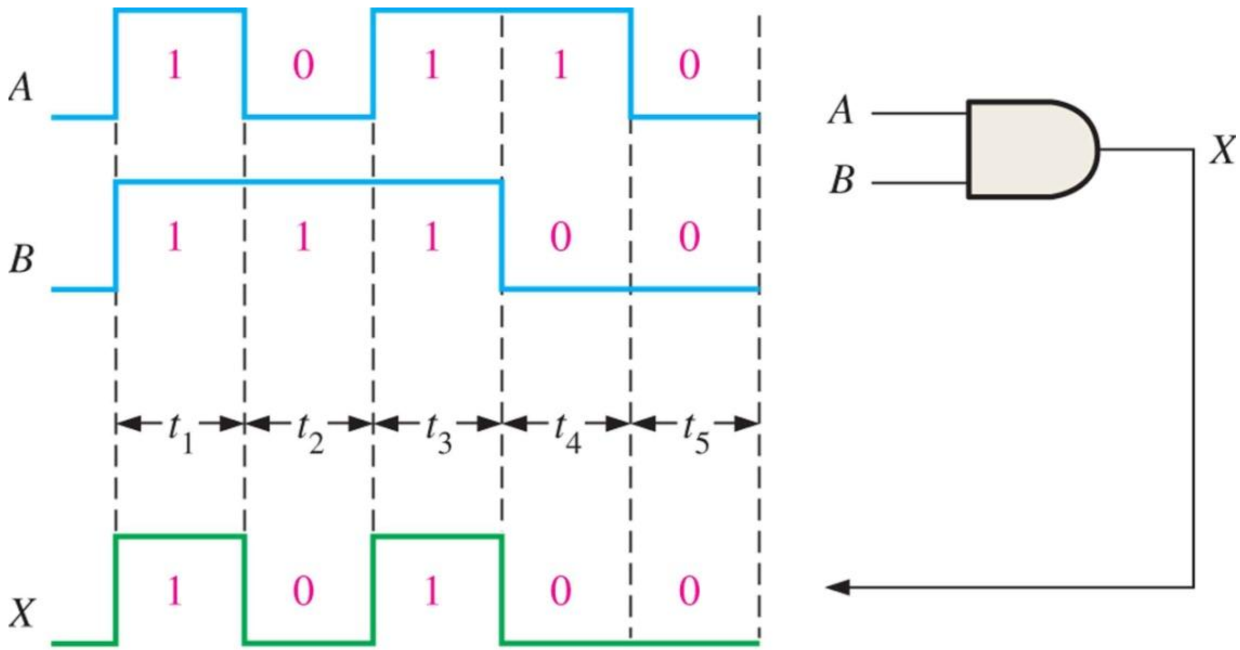


AND Gate

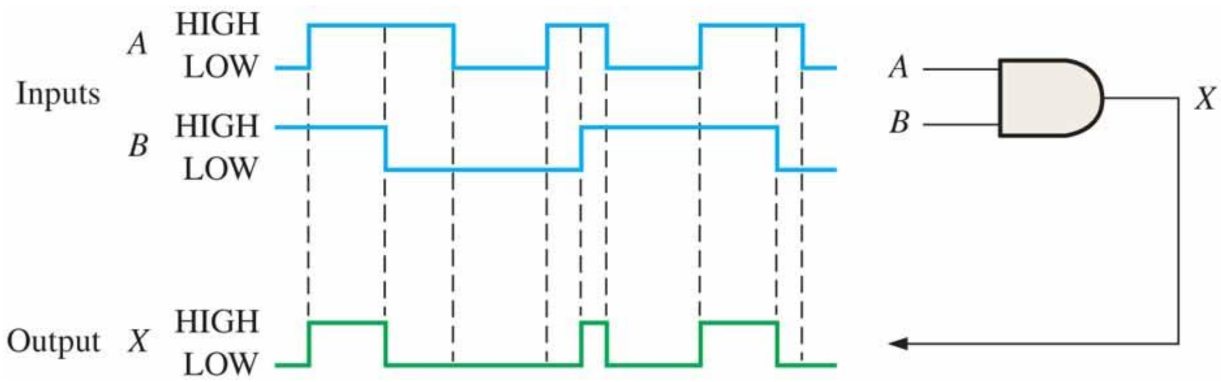


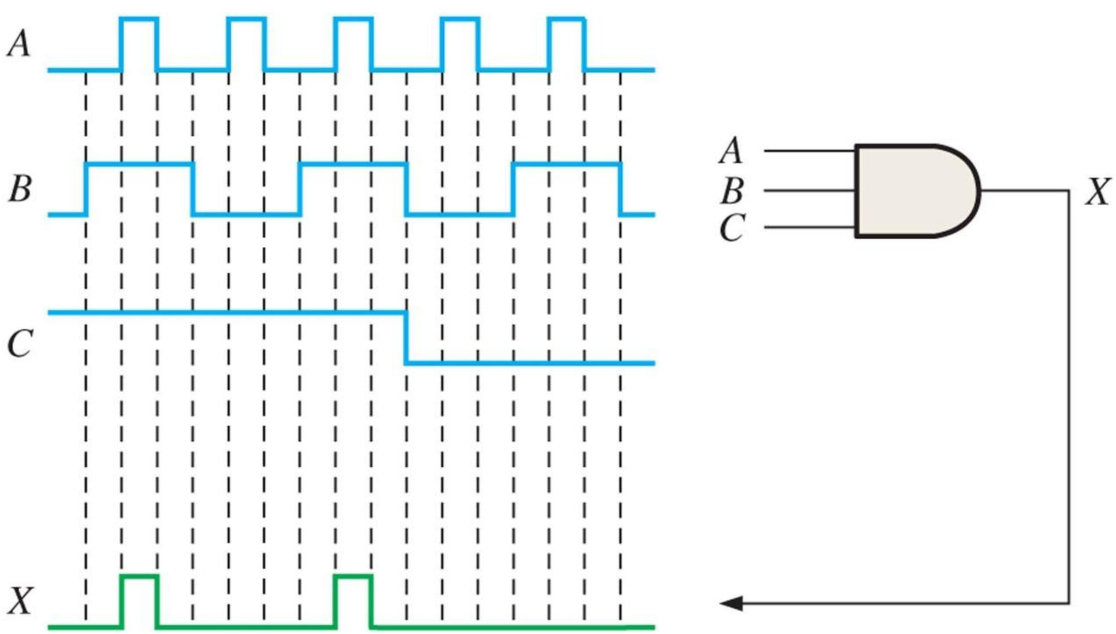
- Logic Expressions: The **AND** operation is usually shown with a dot between the variables but it may be implied (no dot).
- $X = A \cdot B$ or $X = AB$.

Inputs		Output
A	*B*	*X*
0	0	0
0	1	0
1	0	0
1	1	1



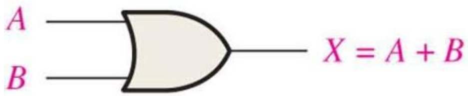
AND Gate (Con't)

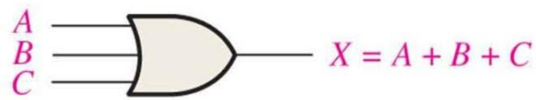


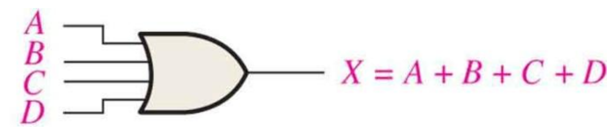


OR Gate

- **OR gate** produces a HIGH output when one or more inputs are HIGH.

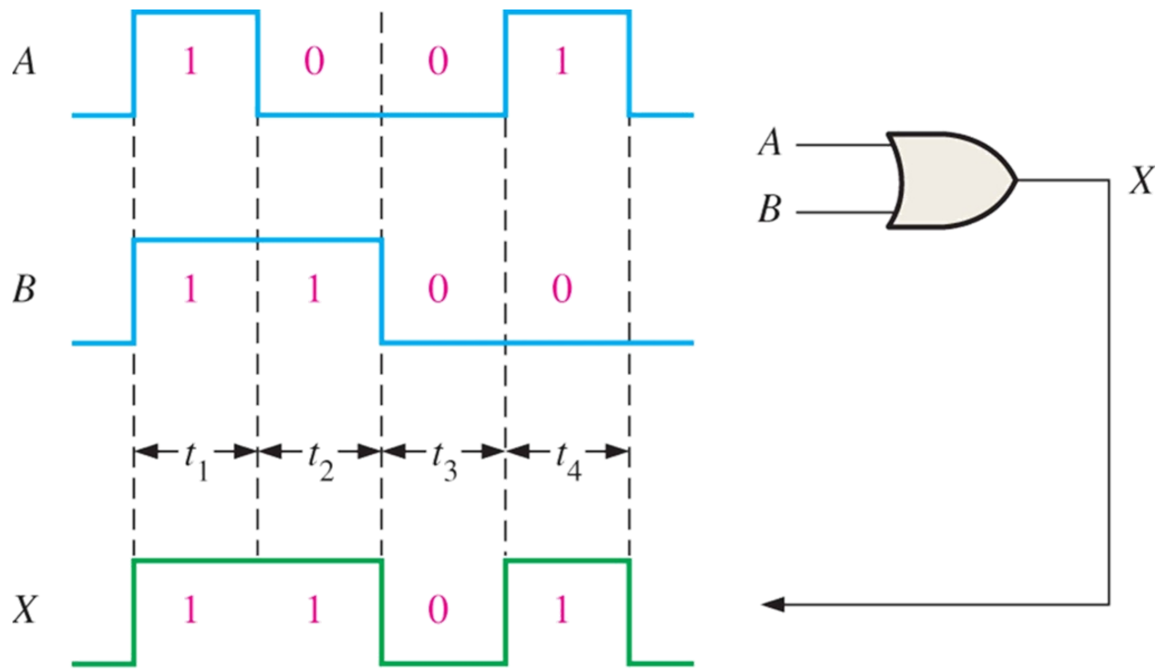




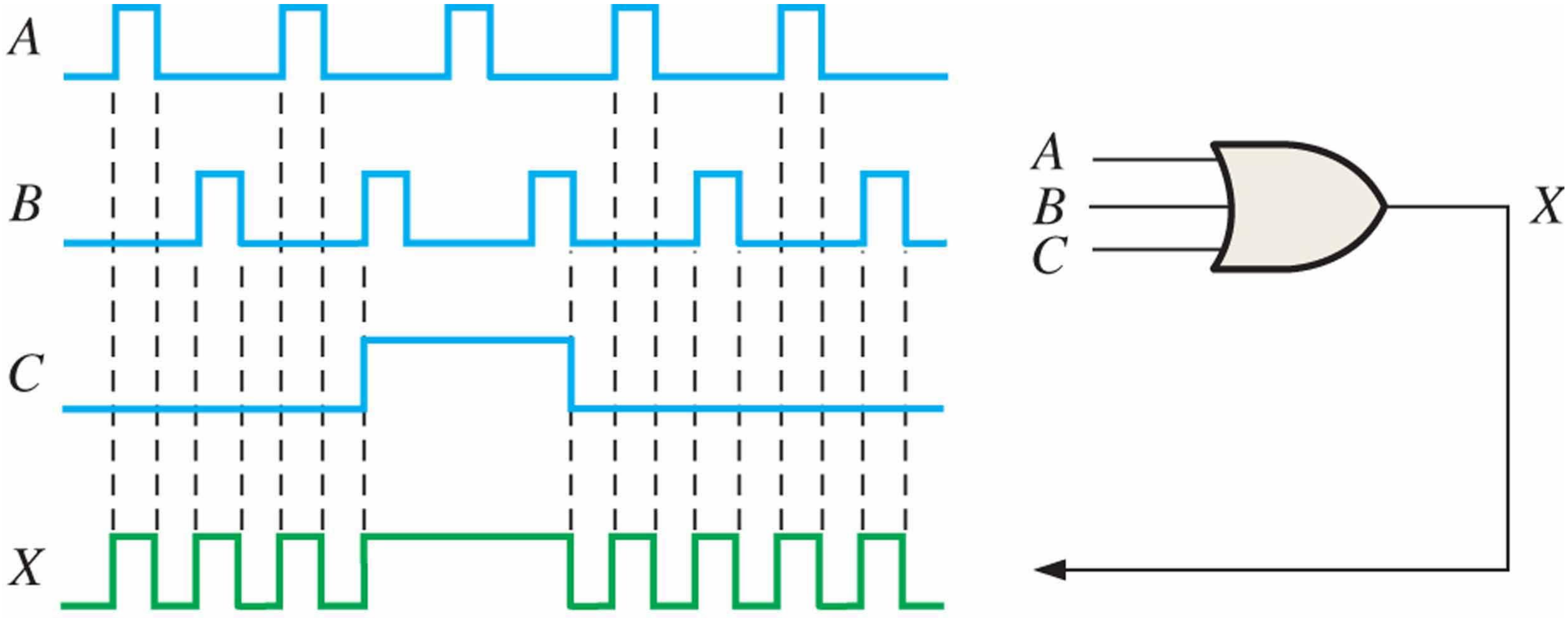


- Expression: Use plus sign (+) between the variables.
  - $X = A + B$.

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

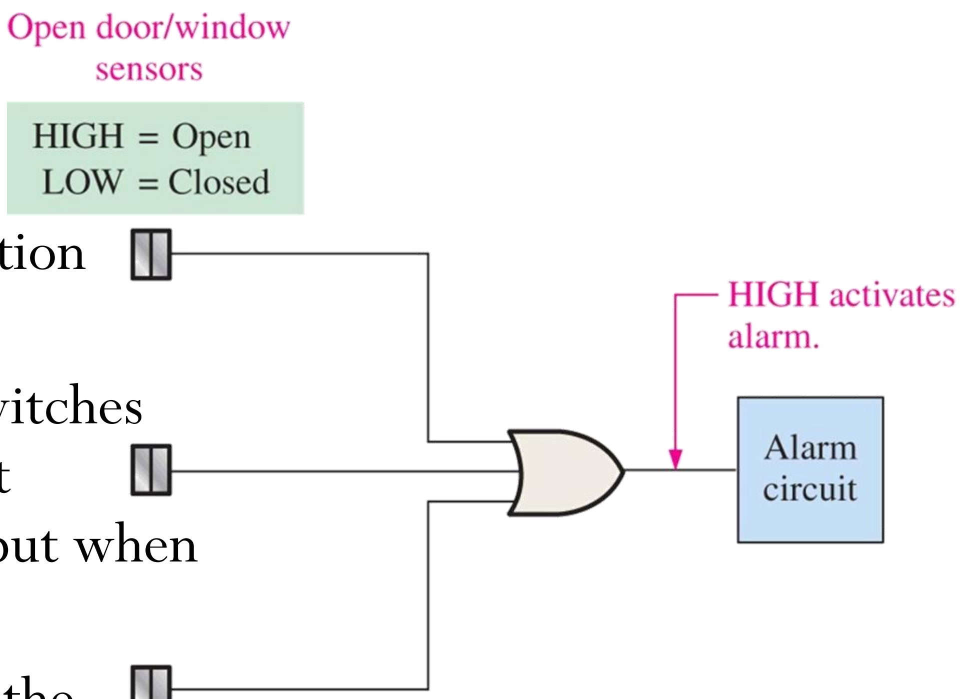


OR Gate (con't)



Application

- A simplified intrusion detection system using an OR gate.
- The sensors are magnetic switches that produce a HIGH output when open and a LOW output when closed.
- As long as the windows and the door are secured, the switches are closed and all three of the OR gate inputs are LOW.
- When one of the windows or the door is opened, a HIGH is produced on that input to the OR gate and the gate output goes HIGH.

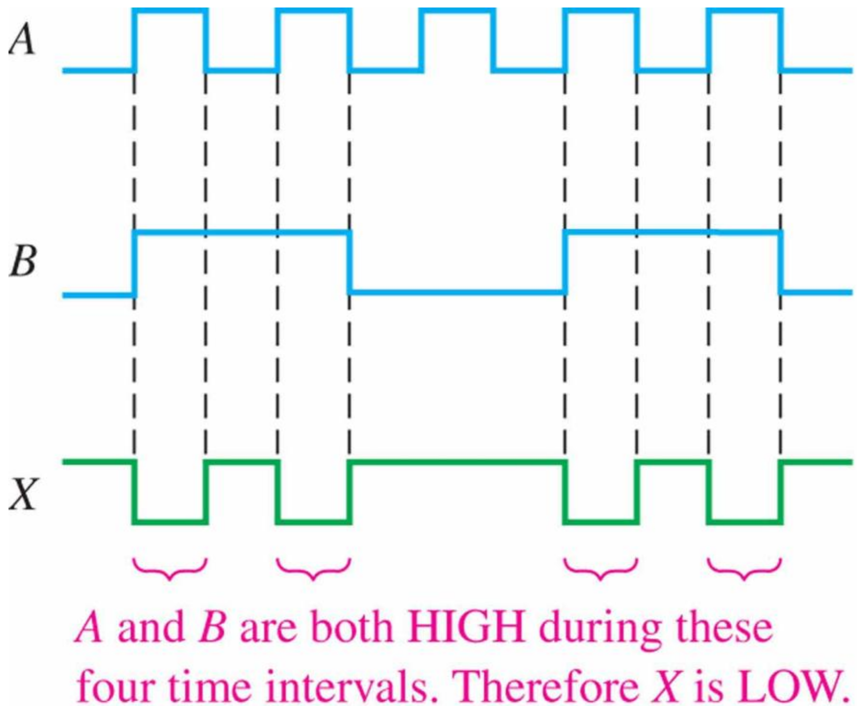


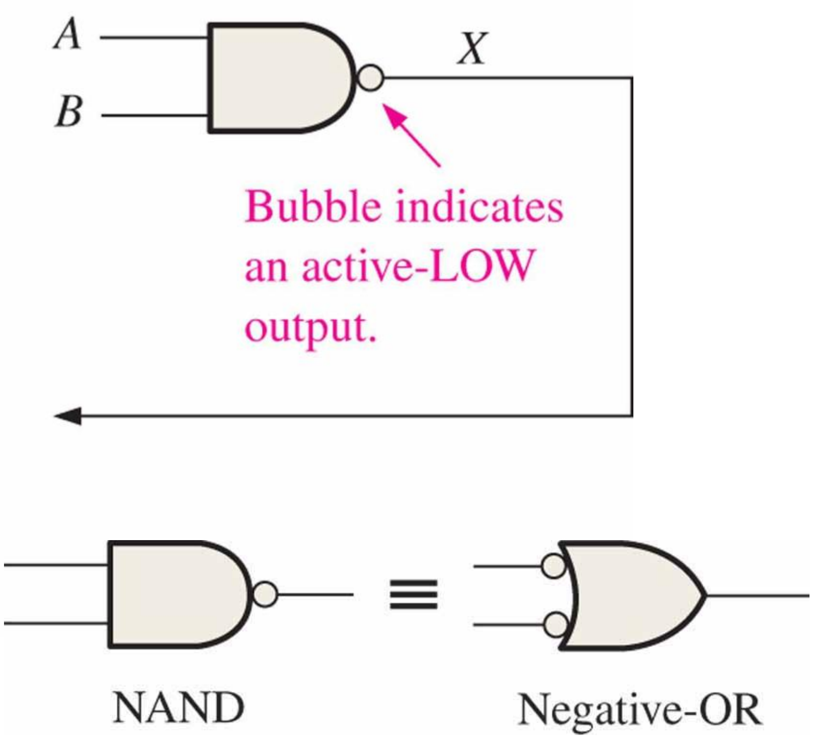
It then activates and latches an alarm circuit to warn of the intrusion.

NAND Gate

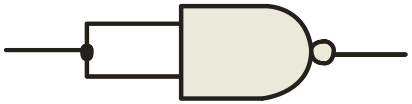
- **NAND gate** produces a LOW output only when all the inputs are HIGH.

Inputs		Output
A	*B*	*X*
0	0	1
0	1	1
1	0	1
1	1	0

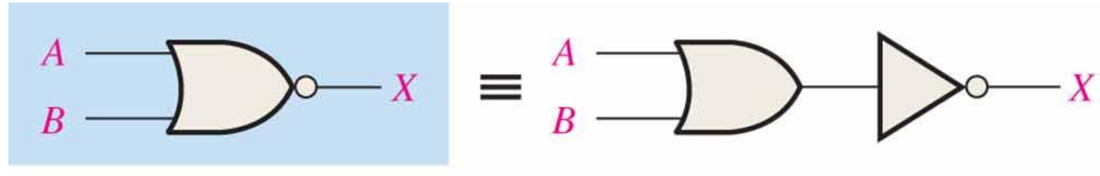




NAND Gate (Con't)

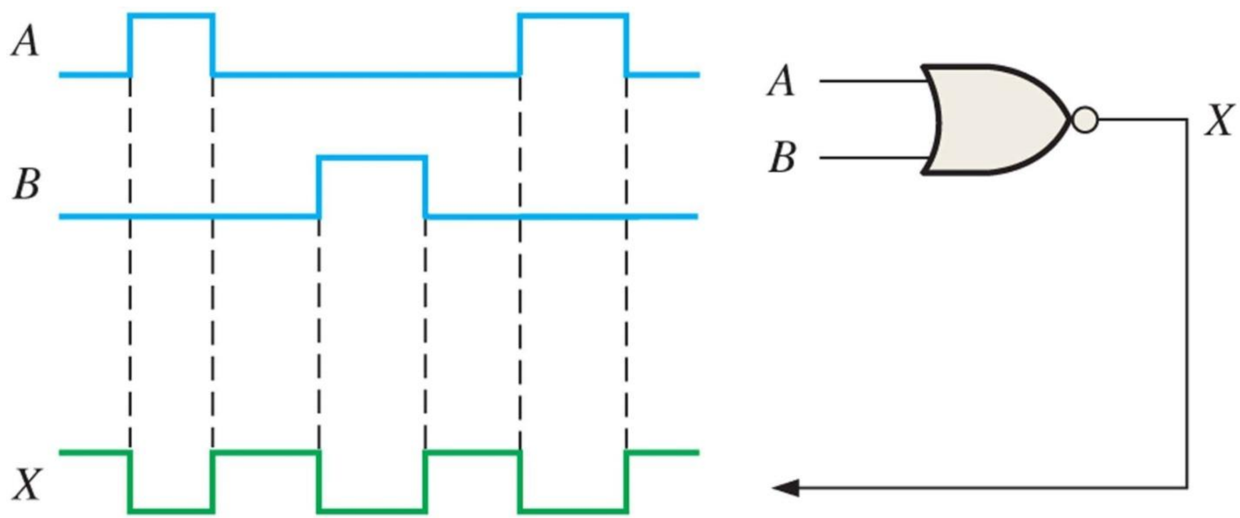
- The NAND gate is a popular logic element because it can be used as a universal gate.
  - NAND gates can be used in combination to perform the AND, OR, and inverter operations.
  - In fact, all other basic gates can be constructed from NAND gates.
  - Ex. Inverter 
  - We will revisit this property.
- The **NAND** operation is shown with a dot between the variables and an overbar covering them.
  - $X = \overline{A \cdot B}$ (Alternatively, $X = \overline{AB}$.)

NOR Gate

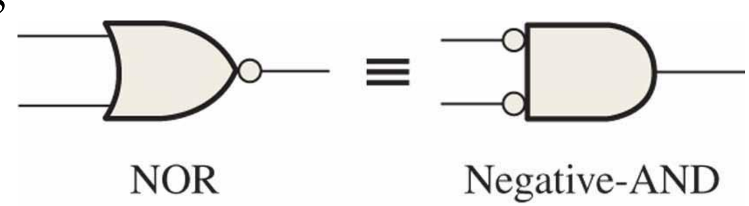


- **NOR gate**'s output is LOW when one or more of the inputs are HIGH.

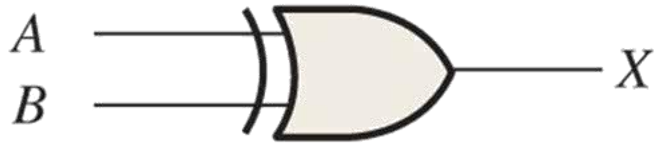
Inputs		Output
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	0



- Can also be used as a universal gate.

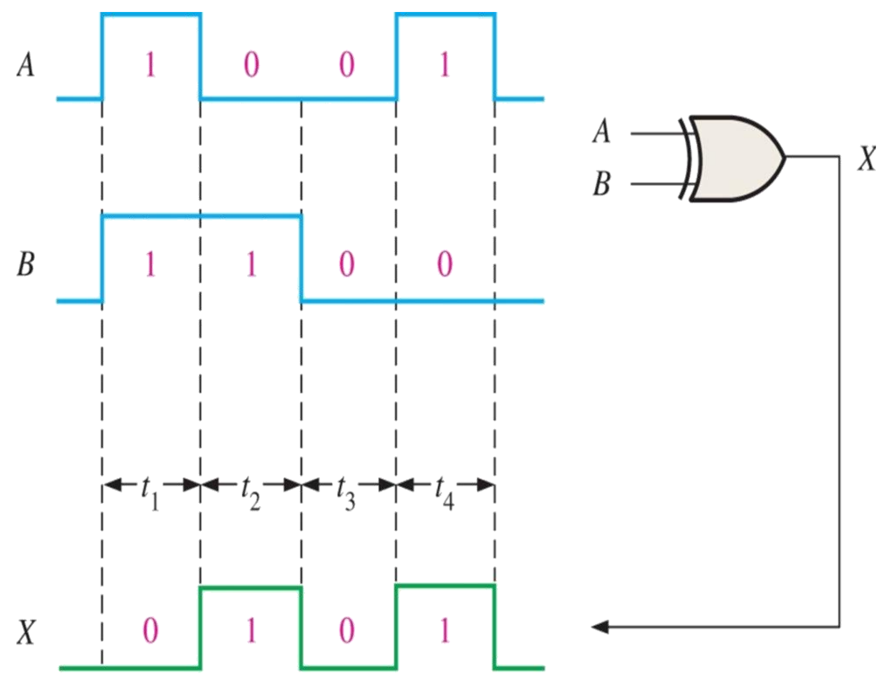


XOR



Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

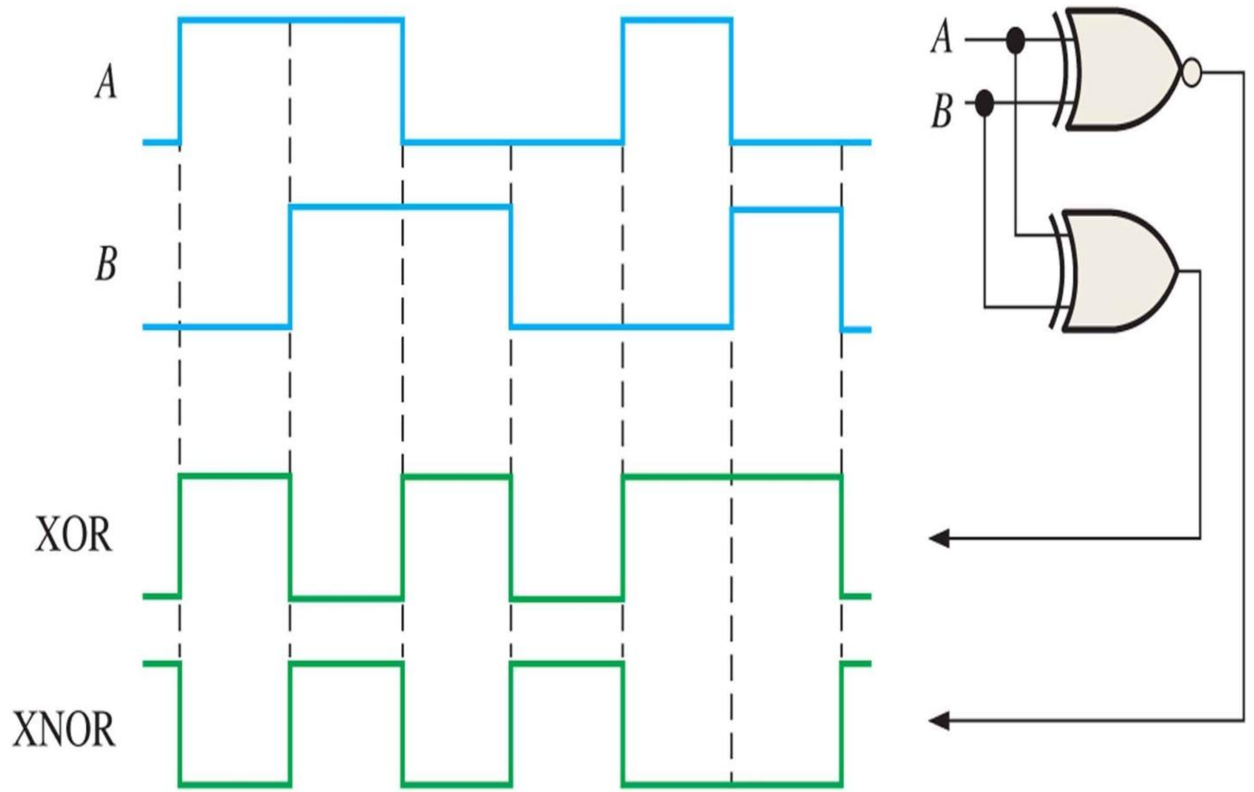
- **Exclusive-OR (XOR) gate** produces a HIGH output only when its two inputs are at opposite levels.
- XOR and XNOR gates are formed by a combination of other gates already discussed.
  - Because of their fundamental importance in many applications, these gates are often treated as basic logic elements with their own unique symbols.



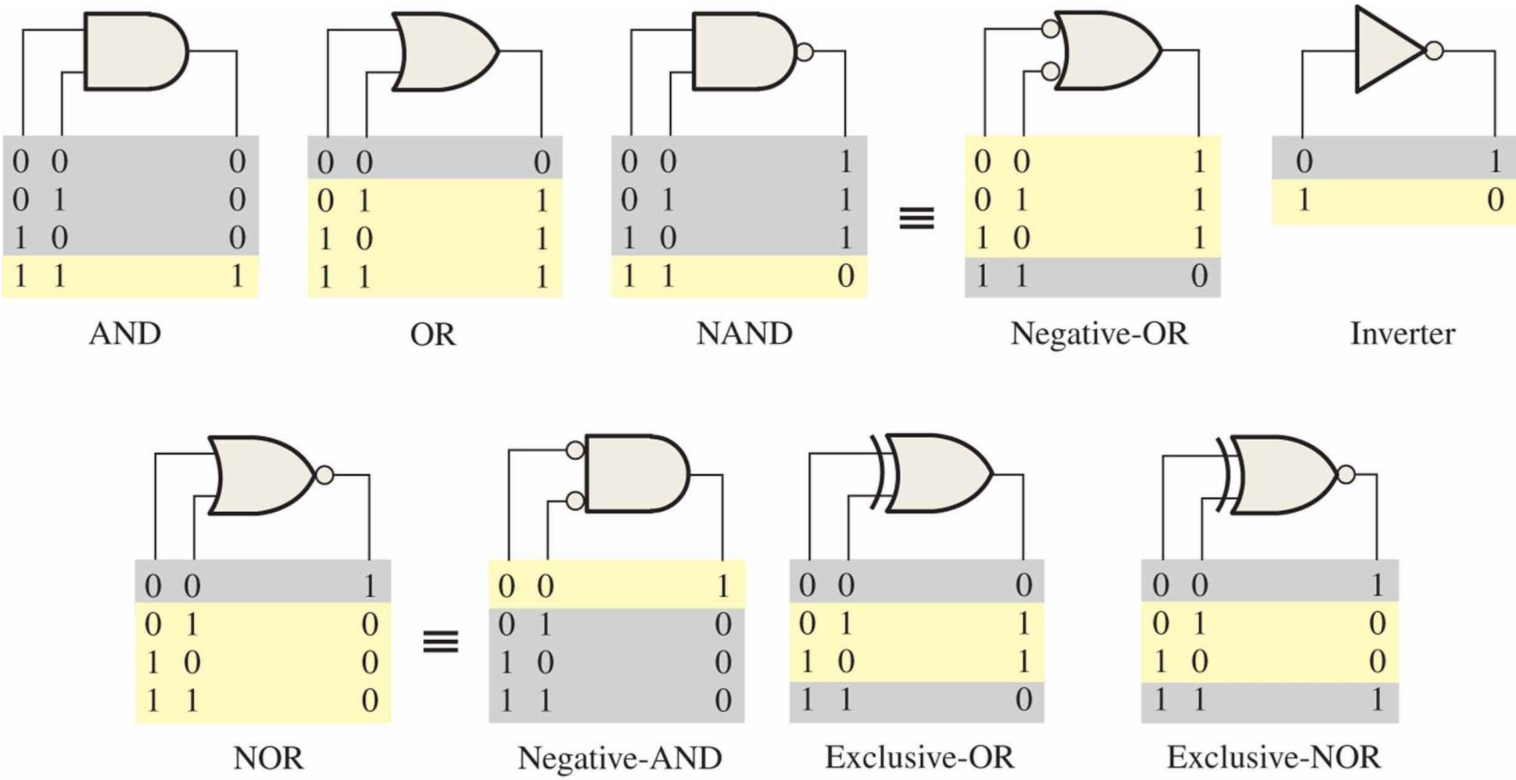
XOR and XNOR

- **Exclusive-NOR gate**: A logic gate that produces a LOW only when the two inputs are at opposite levels.

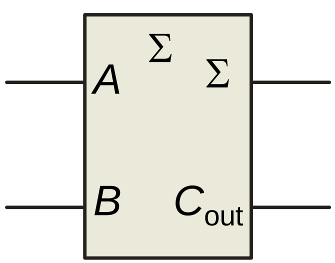
Inputs		Output
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	1



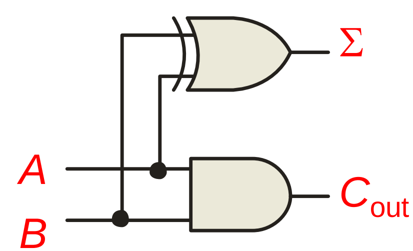
Logic Gates & Application



Note: Active states are shown in yellow.

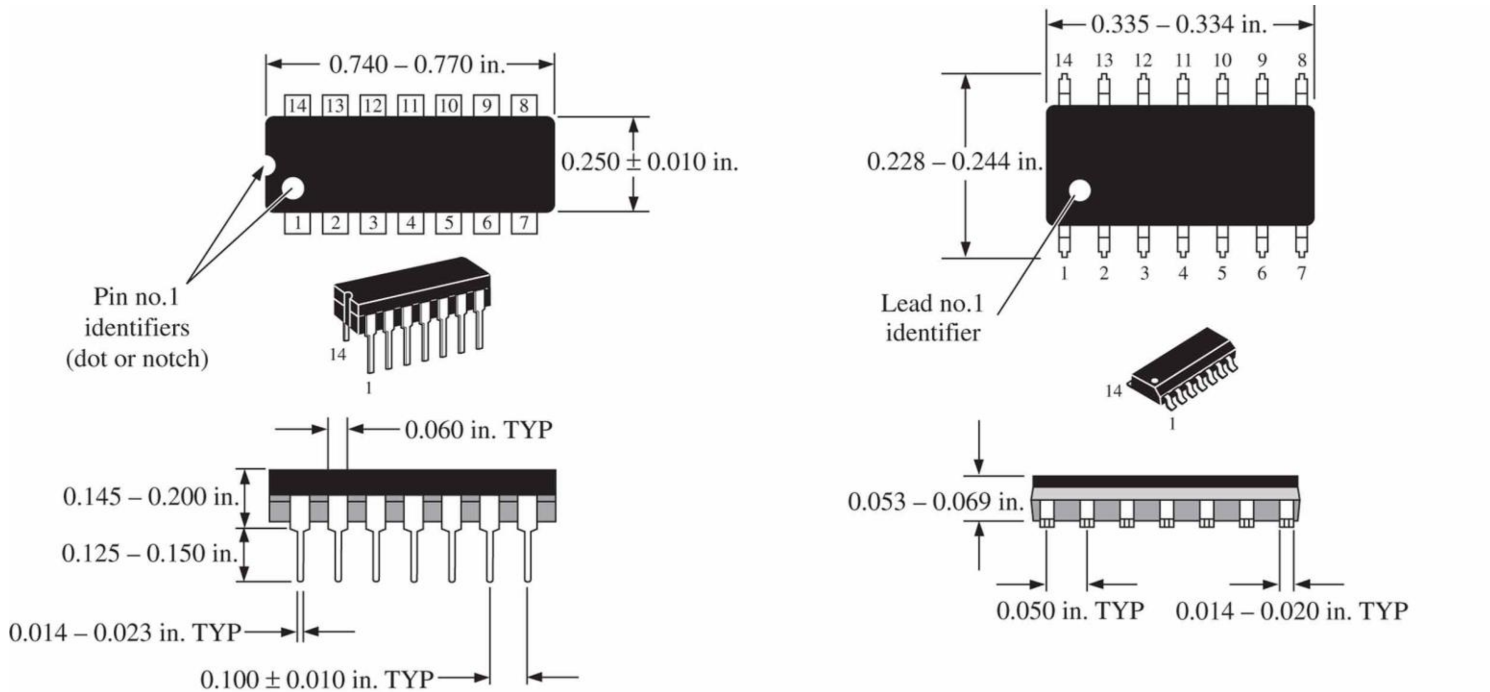


Inputs		Outputs	
A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Fixed Function Logic

- Typical dual in-line (DIP) and small-outline (SOIC) packages showing pin numbers and basic dimensions.



(a) 14-pin dual in-line package (DIP) for feedthrough mounting

(b) 14-pin small outline package (SOIC) for surface mounting

Pin configuration diagrams