Chapter 5 Designing Combinational Systems (I)

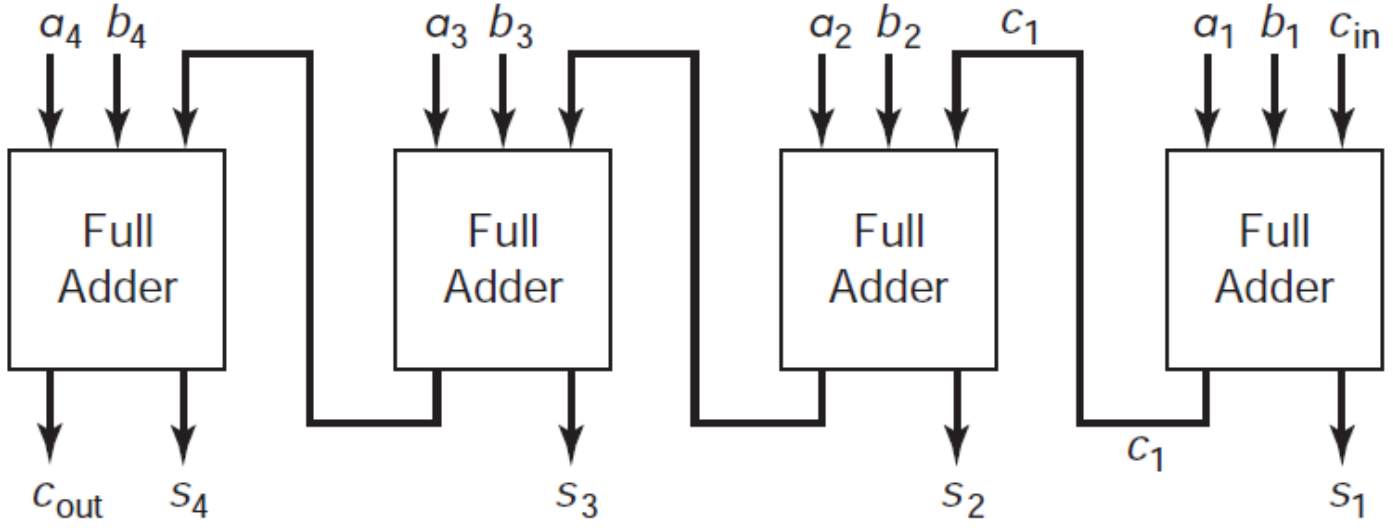
Overview (1/1)

- Large systems are usually designed by breaking them up into smaller subsystems
- We will first look at systems that consist of a number of identical blocks. (These are sometimes referred to as *iterative systems*.) Adders and other arithmetic functions are examples of this type of system
- Gate arrays are commonly available in three forms: *read only memory* (ROM), *programmable logic array* (PLA), and *programmable array logic* (PAL)

Iterative Systems (1/1)

- When we add two numbers by hand, we add the two least significant bits (plus possibly a carry-in) to produce one bit of the sum and a carry to the next bit
- If we wish to build an *n*-bit adder, we need only connect *n* of these. A 4-bit version is shown in Fig. 5.1

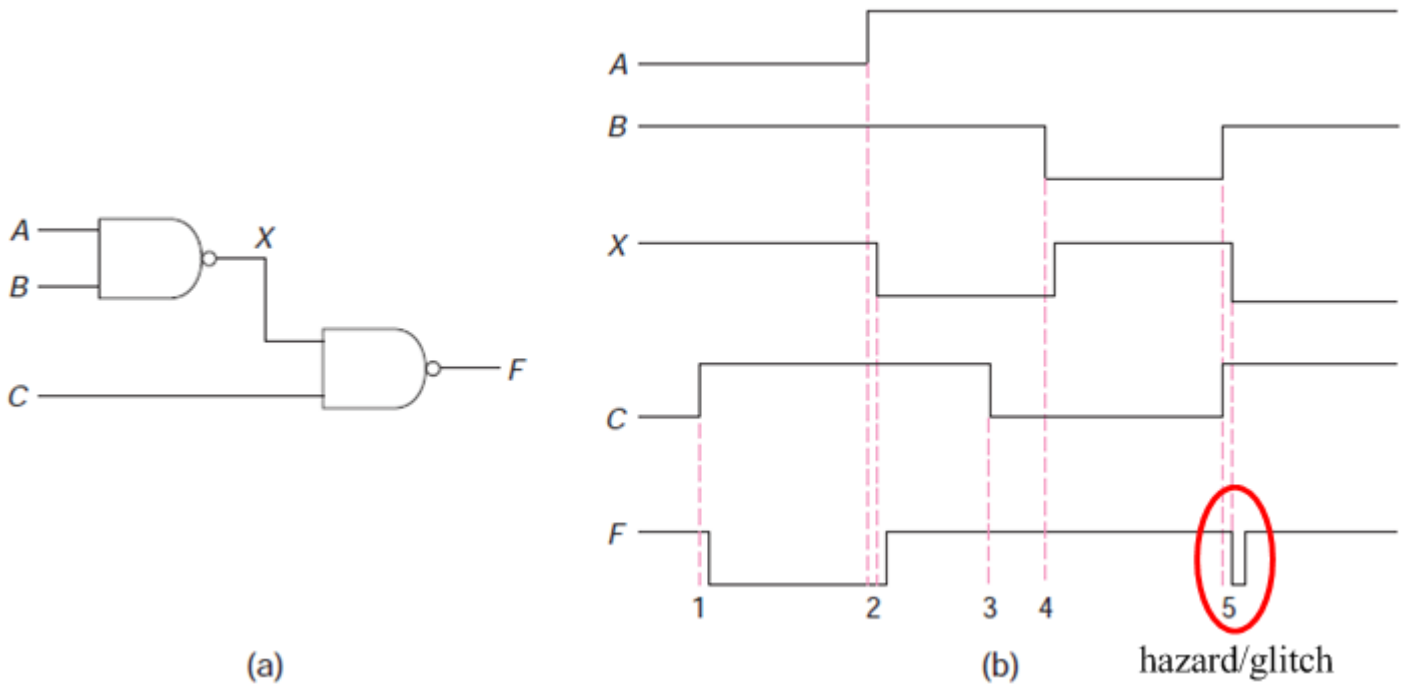
Figure 5.1 A 4-bit adder.



Delay in Combinational Logic Circuits (1/4)

- When the input to a gate changes, the output of that gate does not change instantaneously, but there is a small delay, Δ. If the output of one gate is used as the input to another, the delays add

Figure 5.2 Illustration of gate delay.



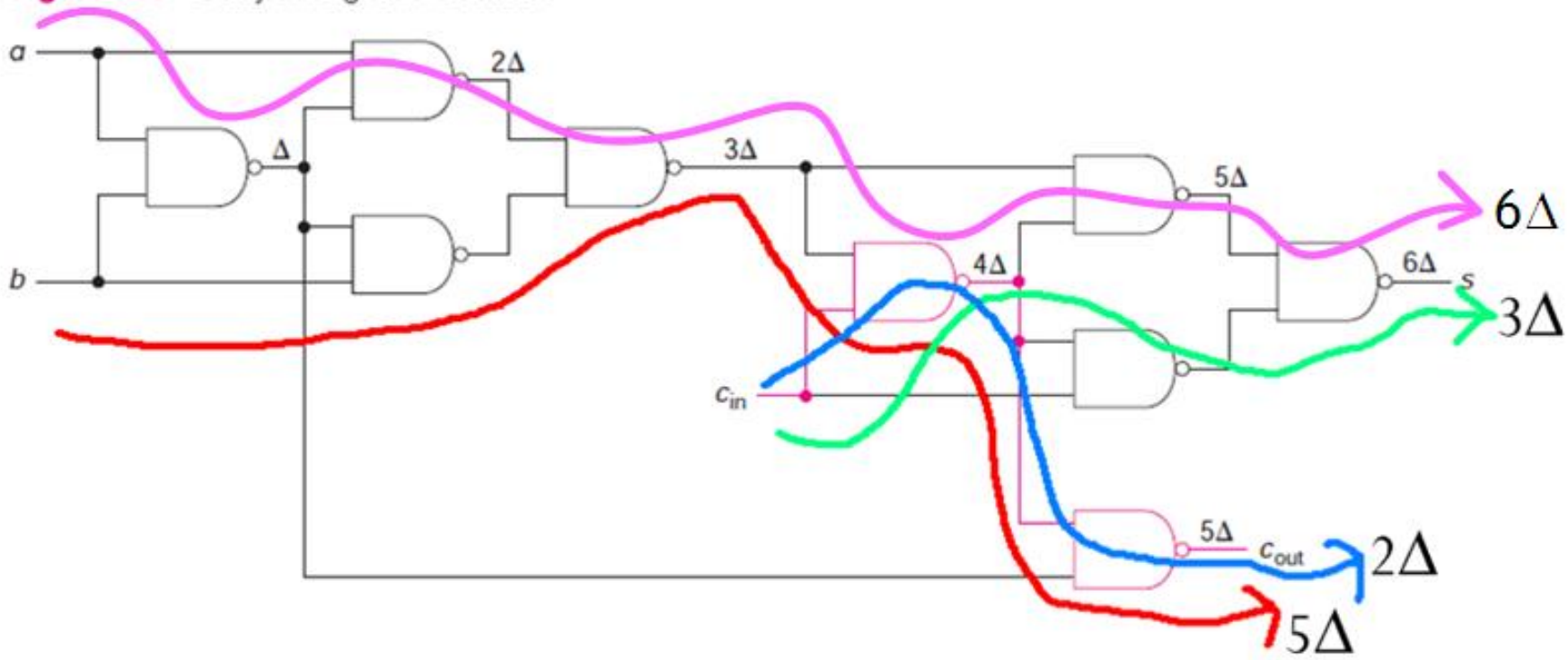
Delay in Combinational Logic Circuits (2/4)

- The output is stable after the longest delay path. We are not usually interested in the output until it is stable
- Consider the *full adder*. It adds two 1-bit numbers and a carry input from the next less significant digit and produces a sum bit and a carry out to the next more significant digit
- We assume that all inputs are available at the same time and the processing delay of a gate is Δ. If two inputs to a gate change at different times, the output may change as late as Δ after the last input changes

Delay in Combinational Logic Circuits (3/4)

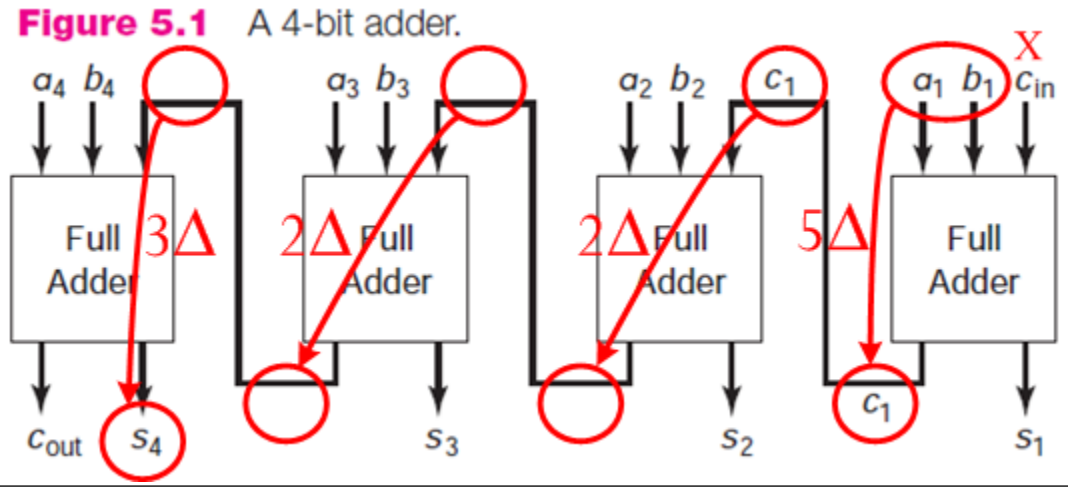
- Figure 5.3 shows four critical paths and their time delays in terms of Δ

Figure 5.3 Delay through a 1-bit adder.



Delay in Combinational Logic Circuits (4/4)

- The total time delay required for an n-bit adder (carry-ripple adder) is calculated as the delay from the inputs to c_{out} (for the least significant bit), plus n-2 times the delay from c_{in} to c_{out} (for the middle full adders), plus the longer of the delay from c_{in} to s (for the most significant bit). That equals $5\Delta + (n-2)\times 2\Delta + 3\Delta = (2n+4)\Delta$
- For a 64-bit adder, the delay would be 132Δ



Adders (1/4)

- One approach to building an n-bit adder is to connect together n 1-bit adders. This is referred to as a carry-ripple adder. The time for the output of the adder to become stable is as large as $(2n+4)\Delta$
- To speed this up, several approaches have been proposed. One approach is to implement a multi-bit adder with an SOP expression
- The truth table for a 2-bit adder is shown in Table 5.1, where a_1 and b_1 are the low-order bits

Adders (2/4)

- The minimum SOP expressions are

$$c_{out} = a_2b_2 + a_1b_1a_2 + a_1b_1b_2 + c_{in}b_1b_2 + c_{in}b_1a_2 + c_{in}a_1b_2 + c_{in}a_1a_2$$

$$s_2 = a_1b_1a_2'b_2' + a_1b_1a_2b_2 + c_{in}'a_1'a_2'b_2 + c_{in}'a_1'a_2b_2' + c_{in}'b_1'a_2'b_2 + c_{in}'b_1'a_2b_2' + a_1'b_1'a_2b_2' + a_1'b_1'a_2'b_2 + c_{in}b_1a_2'b_2' + c_{in}b_1a_2b_2 + c_{in}a_1a_2'b_2' + c_{in}a_1a_2b_2$$

$$s_1 = c_{in}'a_1'b_1 + c_{in}'a_1b_1' + c_{in}a_1'b_1' + c_{in}a_1b_1$$

Table 5.1 Two-bit adder truth table.

a_2	b_2	a_1	b_1	c_{in}	c_{out}	s_2	s_1
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	0	0	0	1
0	0	1	0	1	0	1	0
			≀≀				
1	1	1	0	1	1	1	0
1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	1

Adders (3/4)

- This two-level solution requires a 12-input gate for s_2. Clearly, we could repeat this process for a 3-bit or 4-bit adder, but the algebra gets very complex, and the number of terms increases drastically
- Another problem that we would encounter in this implementation is there is a limitation on the number of inputs (called *fan-in*) for a gate. Gates with 12 inputs may not be practical or may encounter delays of greater than Δ

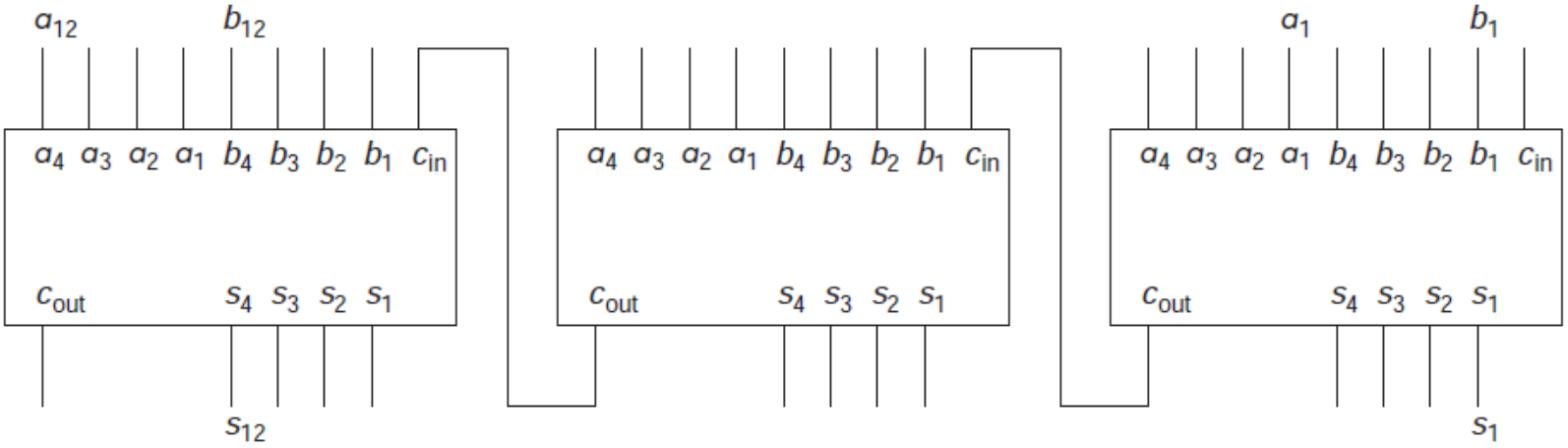
Adders (4/4)

- There are 4-bit adders: the 7483, 7483A, and 74283. Each is implemented differently, with a three-level circuit for the carry-out. The total delay for an n-bit adder reaches

  $(3/4n+1)\Delta$

- When larger adders are needed, these 4-bit adders can be *cascaded*. For example, a 12-bit adder, using three 4-bit adders

Figure 5.4 Cascading 4-bit adders.



Subtractors and Adder (1/2)

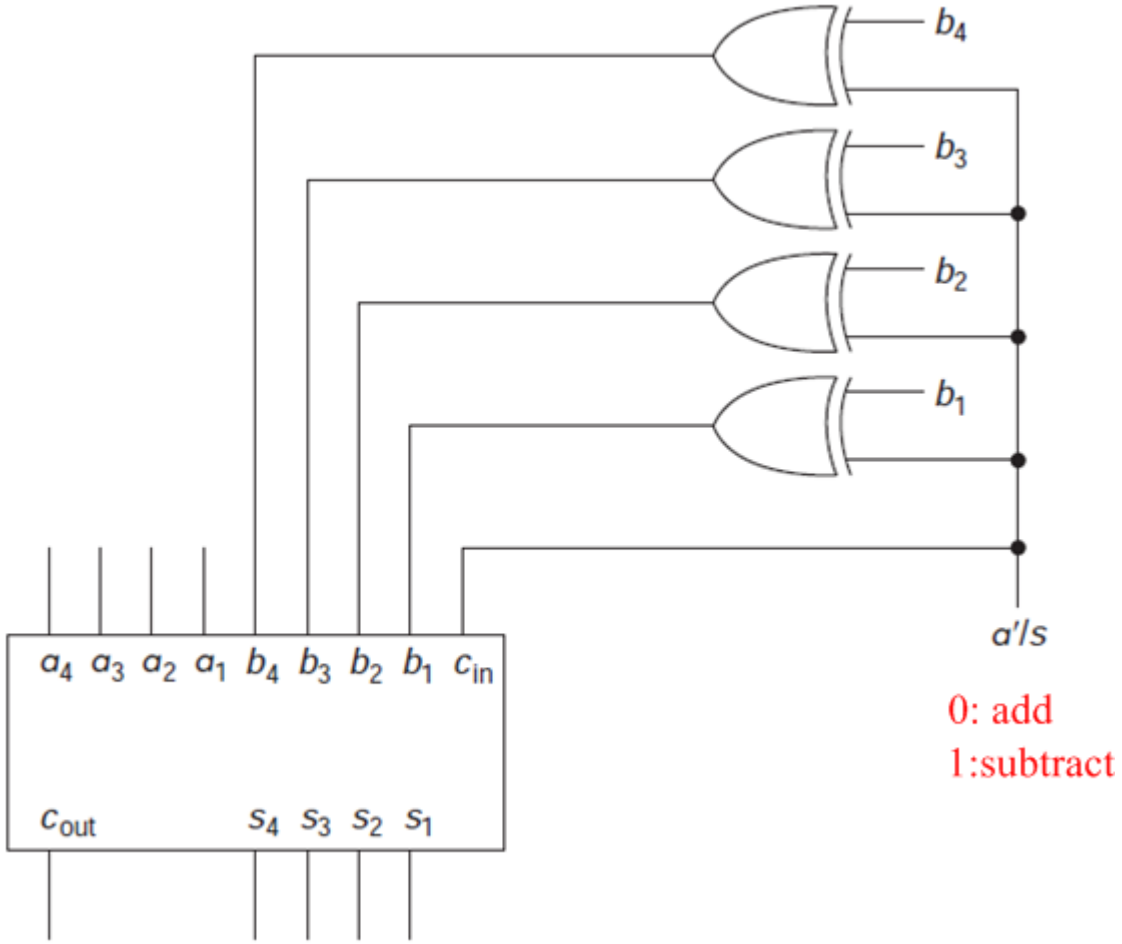
- To build an adder/subtractor, we need a signal line that is 0 for addition and 1 for subtraction. We call that *a*'/*s* (short for add'/subtractor)
- Remember that

$$1 \oplus x = x' \quad \text{and} \quad 0 \oplus x = x$$

- In Fig. 5-5, the input bits b_i, i=1,2,3,4, are connected through Exclusive-OR gates enabled by *a*'/*s*. The numbering format is 2's complement

Subtractors and Adder (2/2)

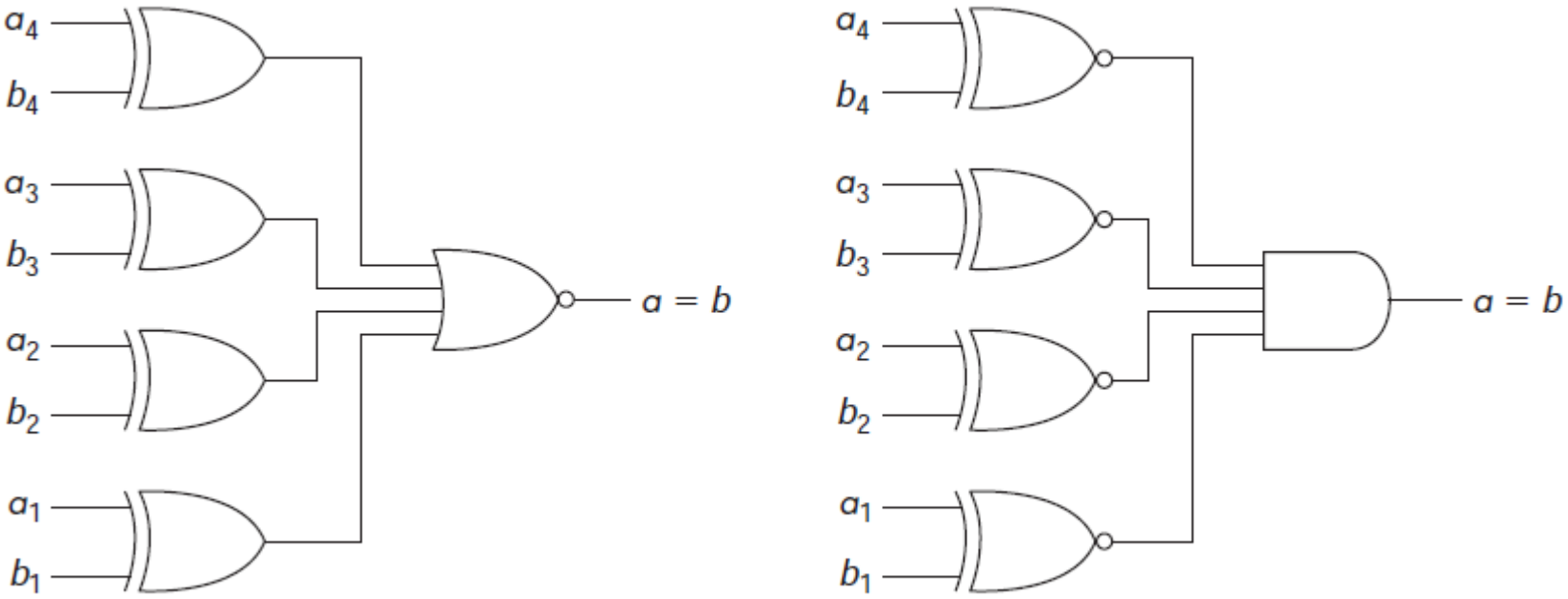
Figure 5.5 A 4-bit adder/subtractor.



Comparators (1/2)

- The Exclusive-OR produces a 1 if the two inputs are unequal and a 0, otherwise. Mutli-bit numbers are unequal if any of the input pairs are unequal.

Figure 5.6 Two 4-bit comparators.



(a) Exclusive-OR

(b) Exclusive-NOR

Comparators (2/2)

- To build a 4-bit comparator that will indicate greater than and less than, as well as equal to (for unsigned numbers), we recognize that, starting at the most significant bit (a_4 and b_4)
- $a > b$ if $a_4 > b_4$ or $(a_4 = b_4$ and $a_3 > b_3)$ or $(a_4 = b_4$ and $a_3 = b_3$ and $a_2 > b_2)$ or $(a_4 = b_4$ and $a_3 = b_3$ and $a_2 = b_2$ and $a_1 > b_1)$
- $a < b$ if $a_4 < b_4$ or $(a_4 = b_4$ and $a_3 < b_3)$ or $(a_4 = b_4$ and $a_3 = b_3$ and $a_2 < b_2)$ or $(a_4 = b_4$ and $a_3 = b_3$ and $a_2 = b_2$ and $a_1 < b_1)$
- $a = b$ if $a_4 = b_4$ and $a_3 = b_3$ and $a_2 = b_2$ and $a_1 = b_1$

Binary Decoders (1/7)

- A *binary decoder* is a device that, when activated, selects one of several output lines, based on a coded input signal. Most commonly, the input is an n-bit binary number, and there are up to 2^n output lines
- Fig. 5.8a is an active high decoder, while Fig. 5.8b is an active low decoder

Binary Decoders (2/7)

Figure 5.8a An active high decoder.

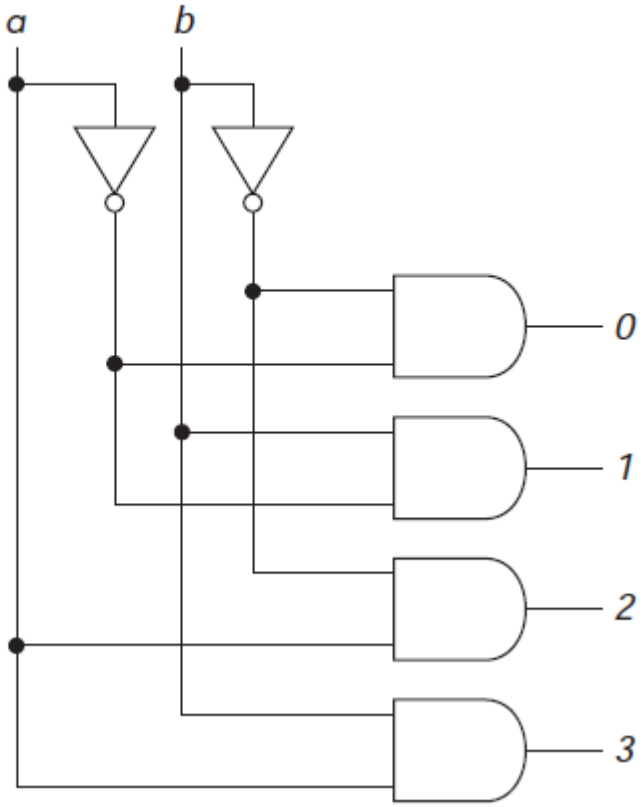


Table 5.2a An active high decoder.

a	*b*	*0*	*1*	*2*	*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

Coded inputs

Binary Decoders (3/7)

Figure 5.8b An active low decoder.

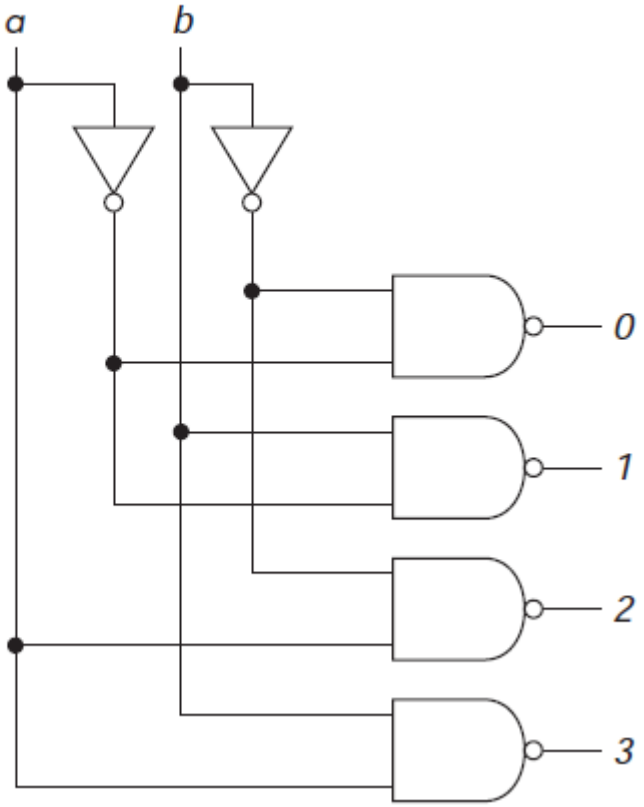


Table 5.2b An active low decoder.

a	*b*	*0*	*1*	*2*	*3*
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Coded inputs

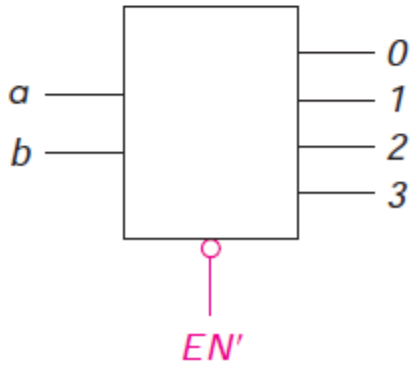
Binary Decoders (4/7)

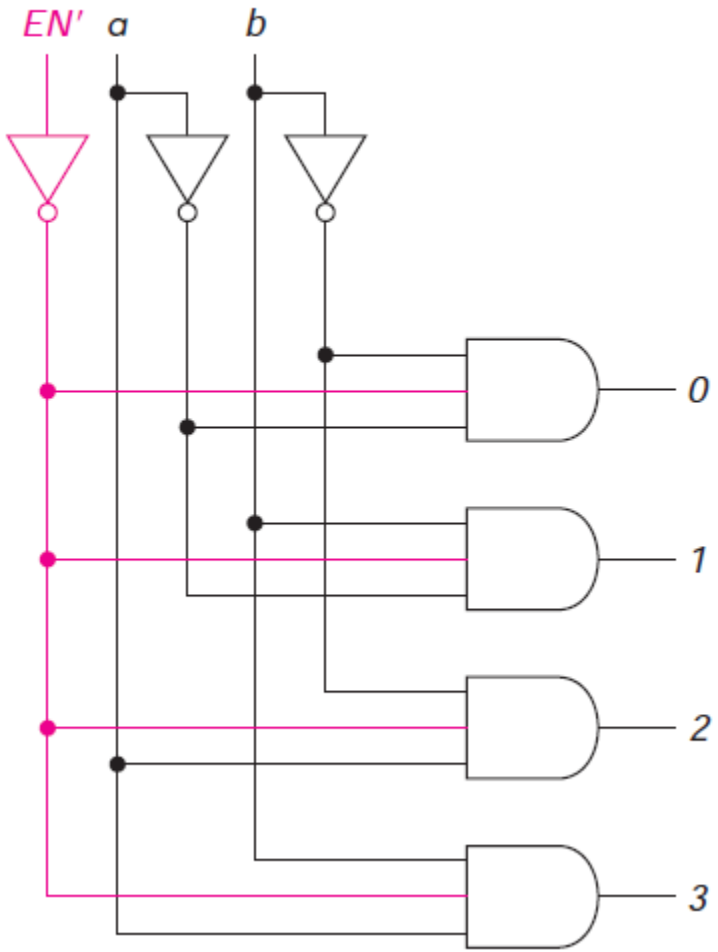
- The circuit for an active high output decoder with an active low enable input is shown in Fig. 5.9. Note the enable input is inverted and connected to each AND gate. When $EN'=1$, a 0 is on the input to each AND gate, and, thus, all of the AND gate outputs are 0
- In most commercial literature, such signals are labeled with an overbar $\overline{EN}$, rather than as EN'

Binary Decoders (5/7)

Figure 5.9 Decoder with enable.

EN'	*a*	*b*	*0*	*1*	*2*	*3*
1	X	X	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1

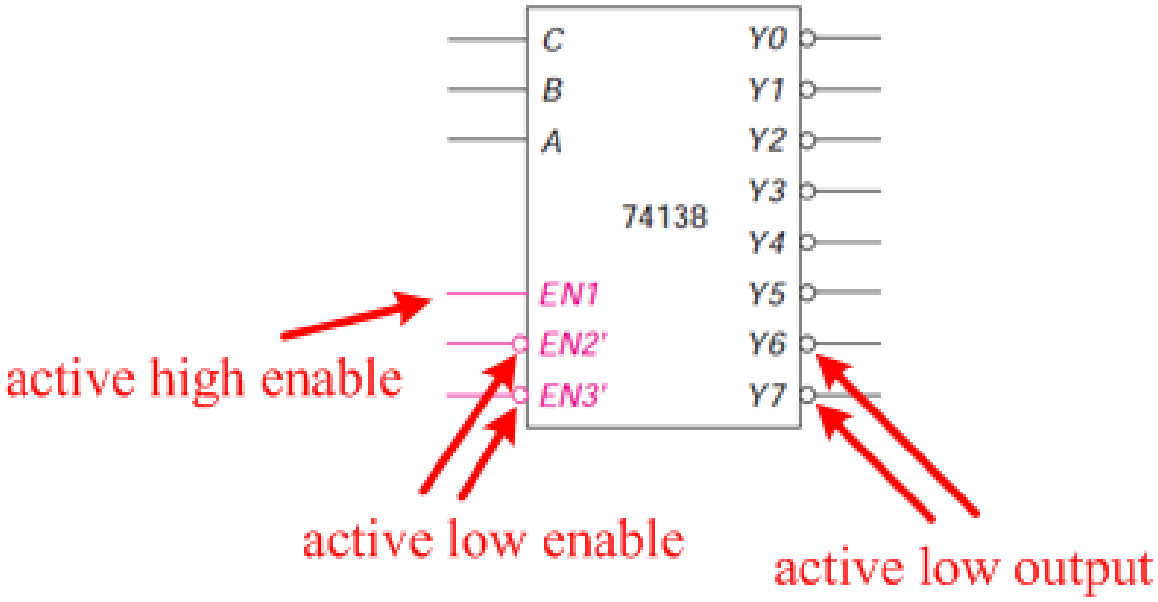




Binary Decoders (6/7)

- A truth table for the 74138 is shown in Table 5.3 and the block diagram is shown in Fig. 5.10

Figure 5.10 The 74138 decoder.



Binary Decoders (7/7)

Table 5.3 The 74138 decoder.

Enables			Inputs			Outputs							
EN1	*EN2′*	*EN3′*	*C*	*B*	*A*	*Y0*	*Y1*	*Y2*	*Y3*	*Y4*	*Y5*	*Y6*	*Y7*
0	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	1	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0