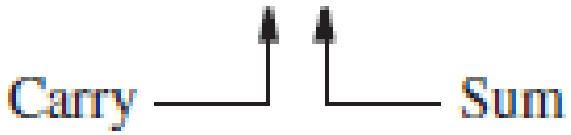
ARITHMETIC CIRCUITS

Addition of Unsigned Numbers

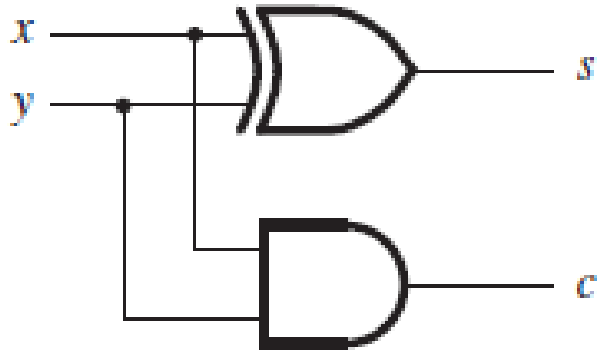
The one-bit addition entails four possible combinations,

x	0	0	1	1
$+\ y$	$+\ 0$	$+\ 1$	$+\ 0$	$+\ 1$
$c\ s$	0 0	0 1	0 1	1 0



		Carry	Sum
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(b) Truth table

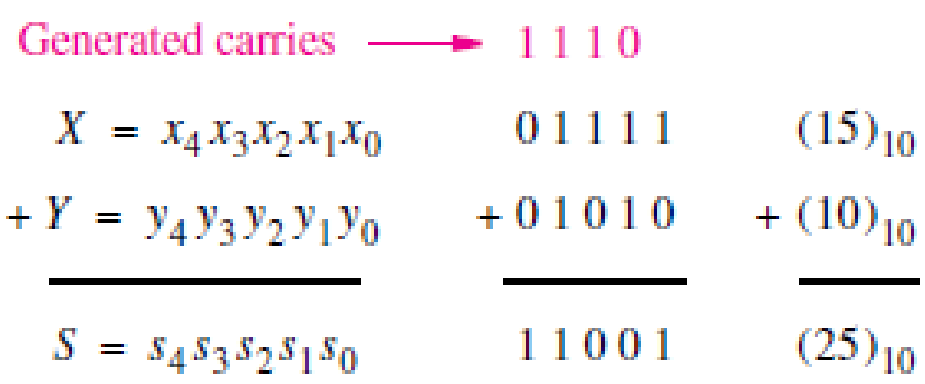


(c) Circuit

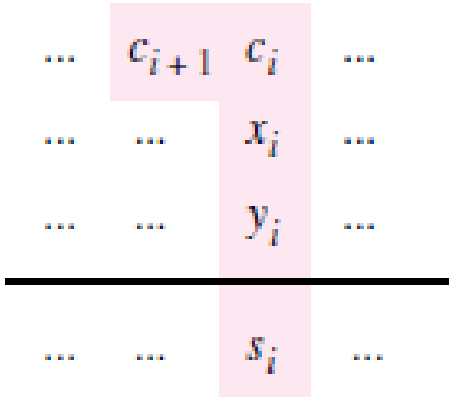
(d) Graphical symbol

The sum bit *s* is the XOR function. The carry *c* is the AND function of inputs *x* and *y*. A circuit realization of these functions is shown in Fig. *c*. This circuit, which implements the addition of only two bits, is called a *half-adder*.

Multibit Addition



(a) An example of addition



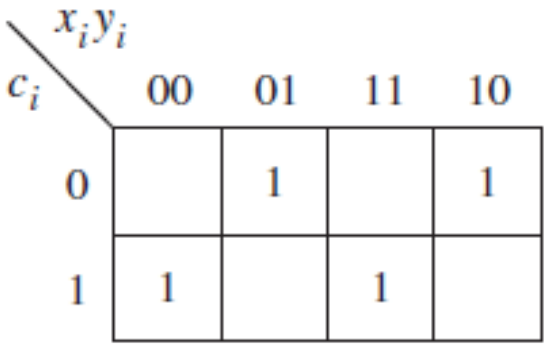
(b) Bit position i

for each bit position i, the addition operation may include a *carry-in* from bit position $i - 1$.

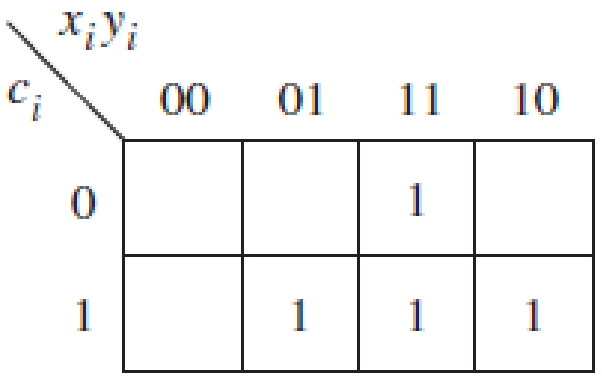
This observation leads to the design of a logic circuit that has three inputs *xi* , *yi* , and *ci* , and produces the two outputs *si* and *ci*+1.

c_i	x_i	y_i	c_{i+1}	s_i
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

(a) Truth table



$$s_i = x_i \oplus y_i \oplus c_i$$



$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

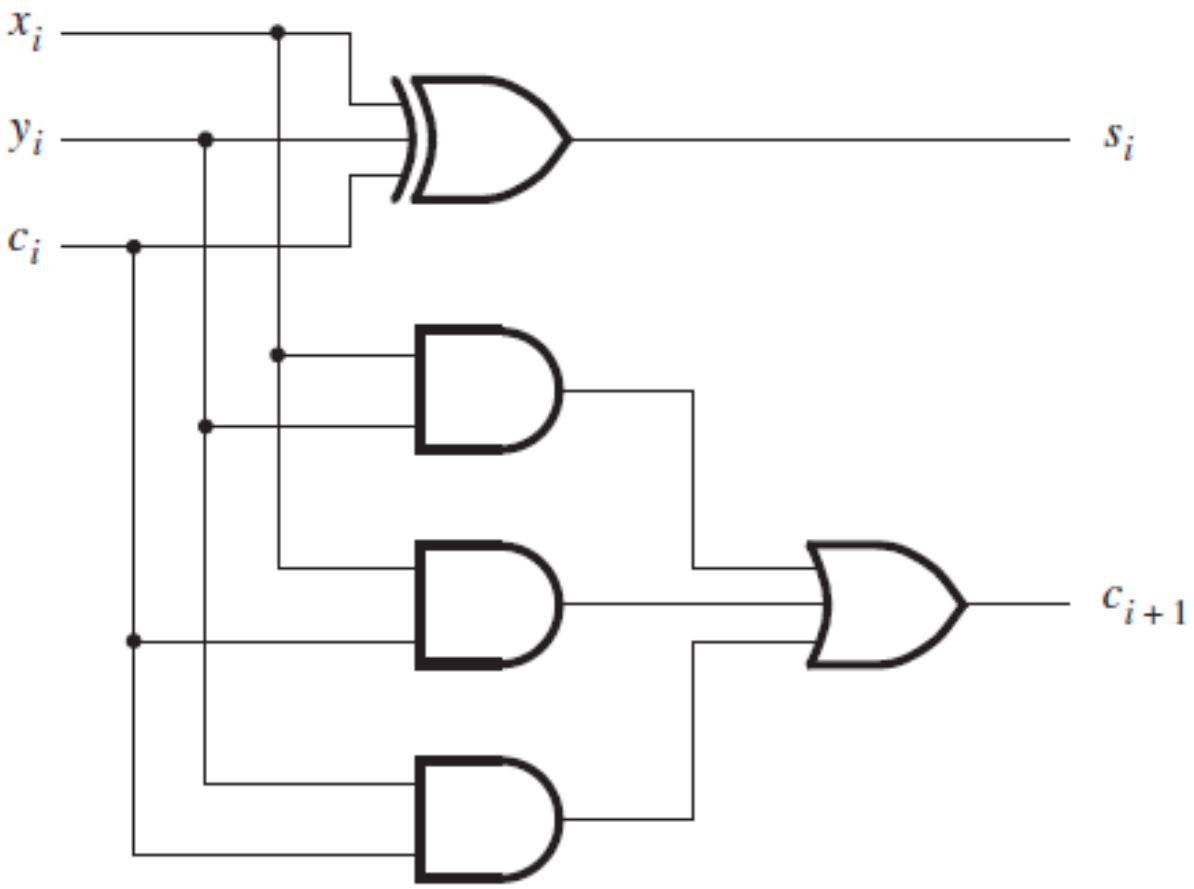
$$s_i = \overline{x}_i y_i \overline{c}_i + x_i \overline{y}_i \overline{c}_i + \overline{x}_i \overline{y}_i c_i + x_i y_i c_i$$

$$s_i = (\overline{x}_i y_i + x_i \overline{y}_i)\overline{c}_i + (\overline{x}_i \overline{y}_i + x_i y_i)c_i$$

$$= (x_i \oplus y_i)\overline{c}_i + \overline{(x_i \oplus y_i)}c_i$$

$$= (x_i \oplus y_i) \oplus c_i$$

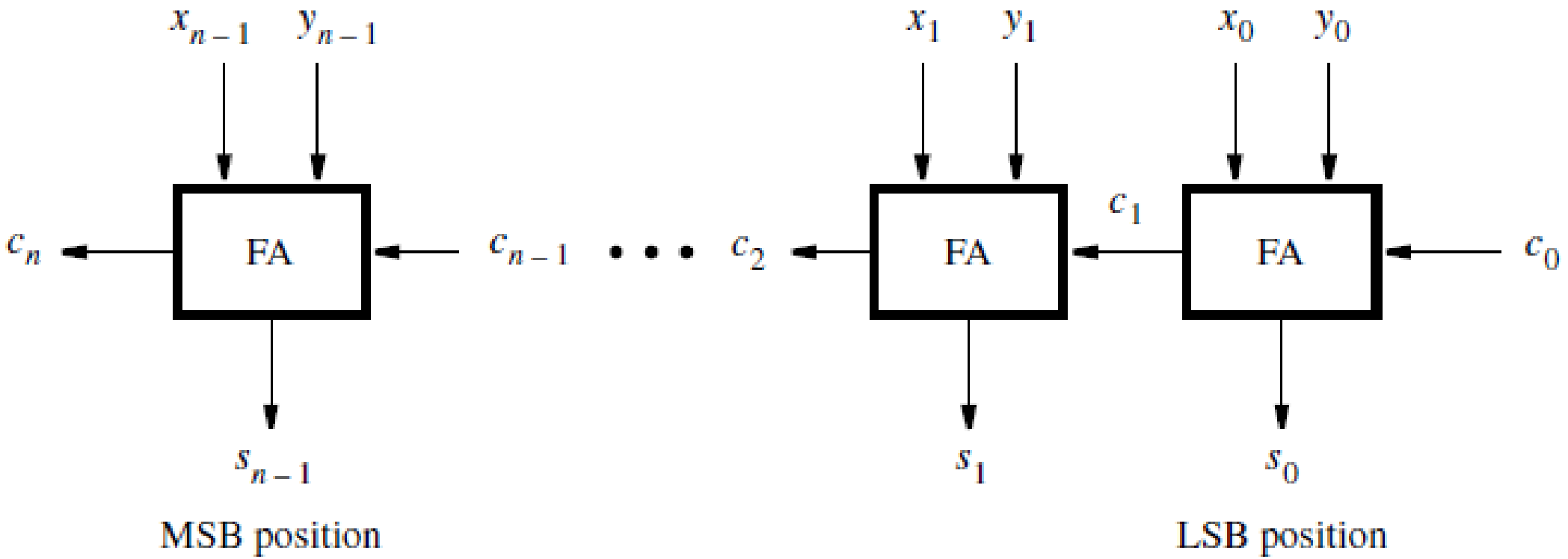
$$s_i = x_i \oplus y_i \oplus c_i$$



(c) Circuit

This circuit is known as a *full-adder*.

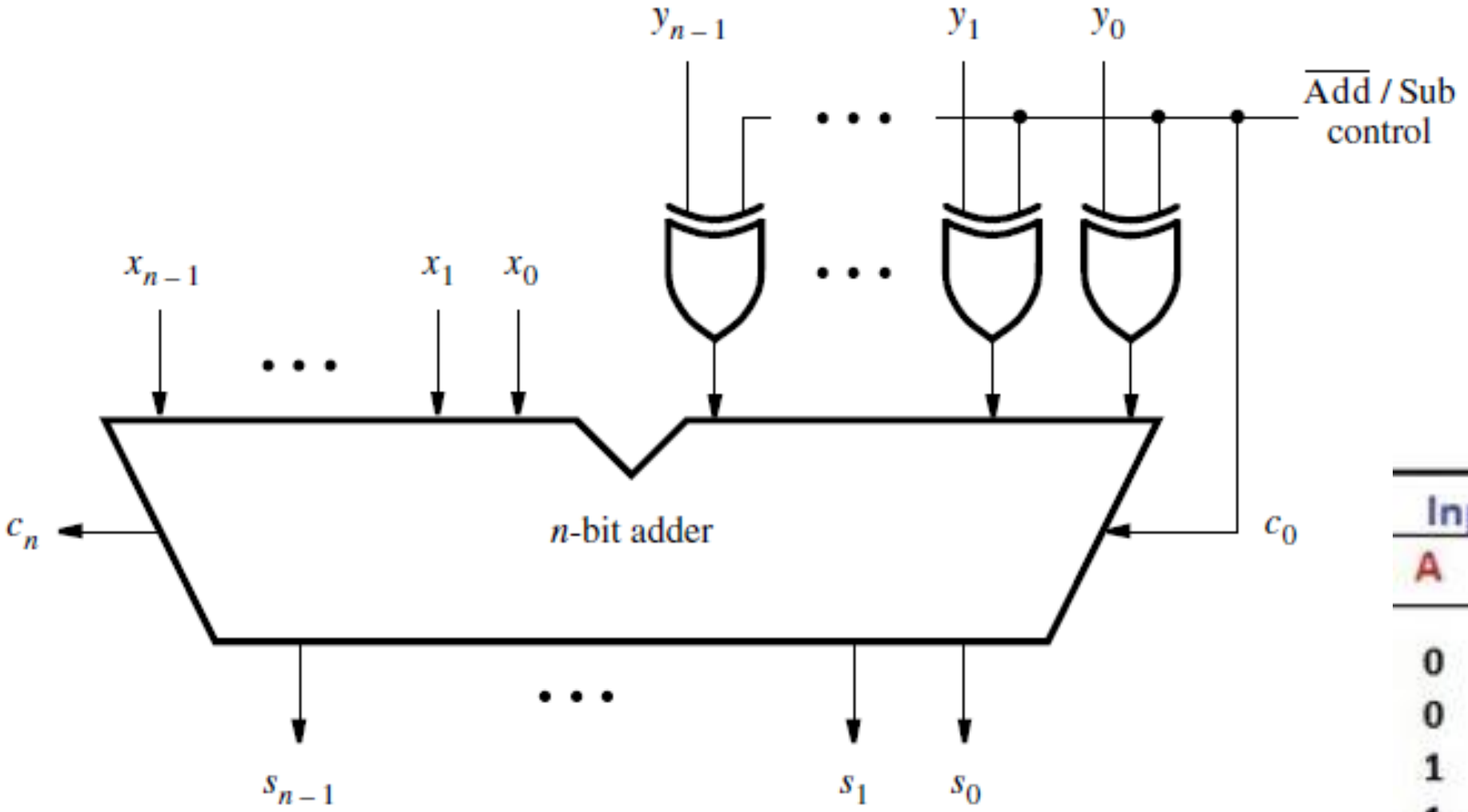
Ripple-Carry Adder



For each bit position we can use a full-adder circuit, connected as shown in Figure

- When the operands X and Y are applied as inputs to the adder, it takes some time before the output sum, S, is valid. Each full-adder introduces a certain delay before its si and $ci+1$ outputs are valid. Let this delay be denoted as Δt.
- Thus the carry-out from the first stage, $c1$, arrives at the second stage Δt after the application of the $x0$ and $y0$ inputs. The carry-out from the second stage, $c2$, arrives at the third stage with a $2\Delta t$ delay, and so on.
- The signal $cn-1$ is valid after a delay of $(n - 1)\ \Delta t$, which means that the complete sum is available after a delay of $n\Delta t$. Because of the way the carry signals "ripple" through the full-adder stages, the circuit is called a ripple-carry adder.

Adder and Subtractor Unit



Inputs		Output
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

When Add/Sub = 0, it performs **X + Y**.

When Add/Sub = 1, it performs

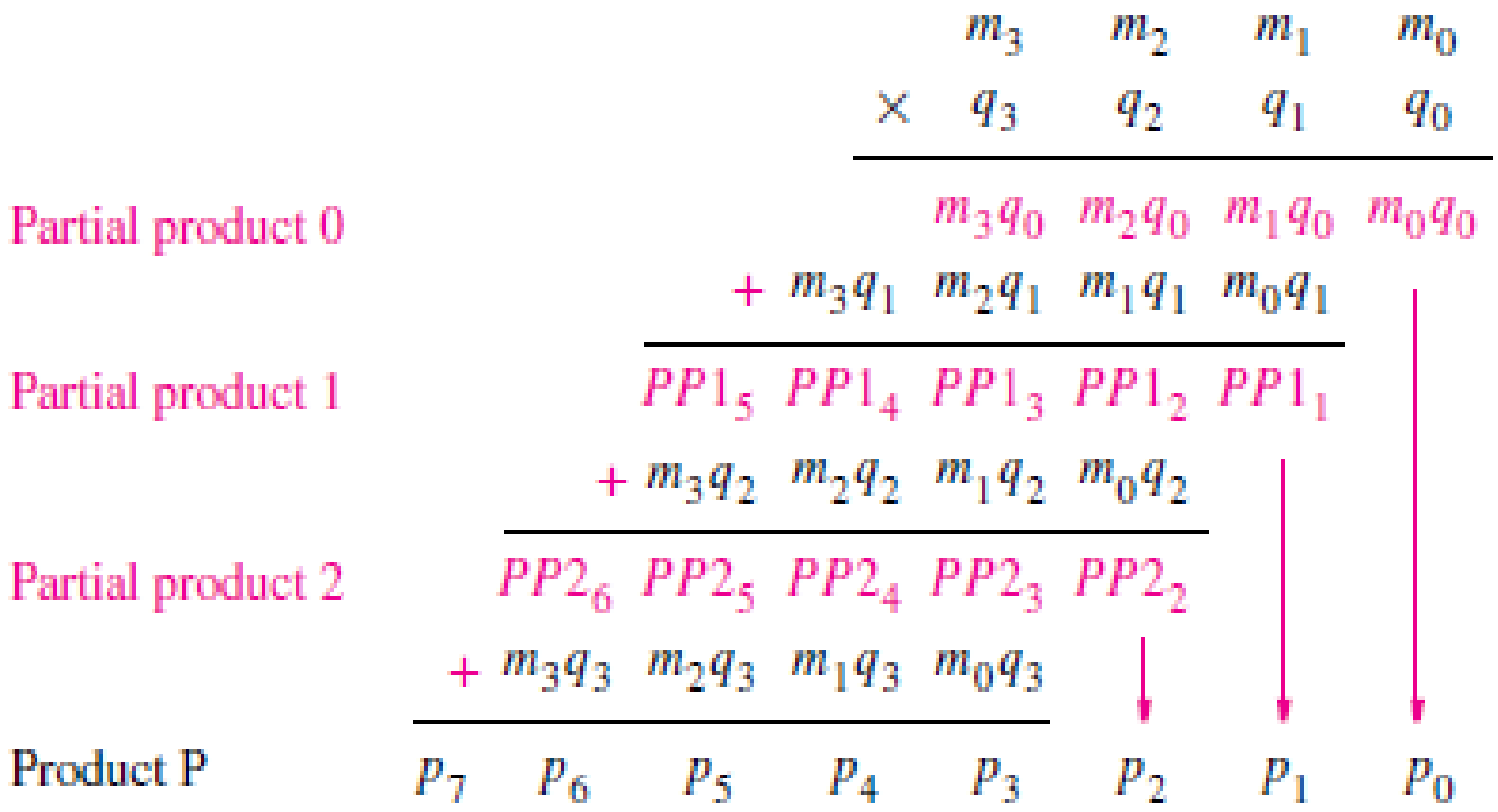
X + Y’ + 1

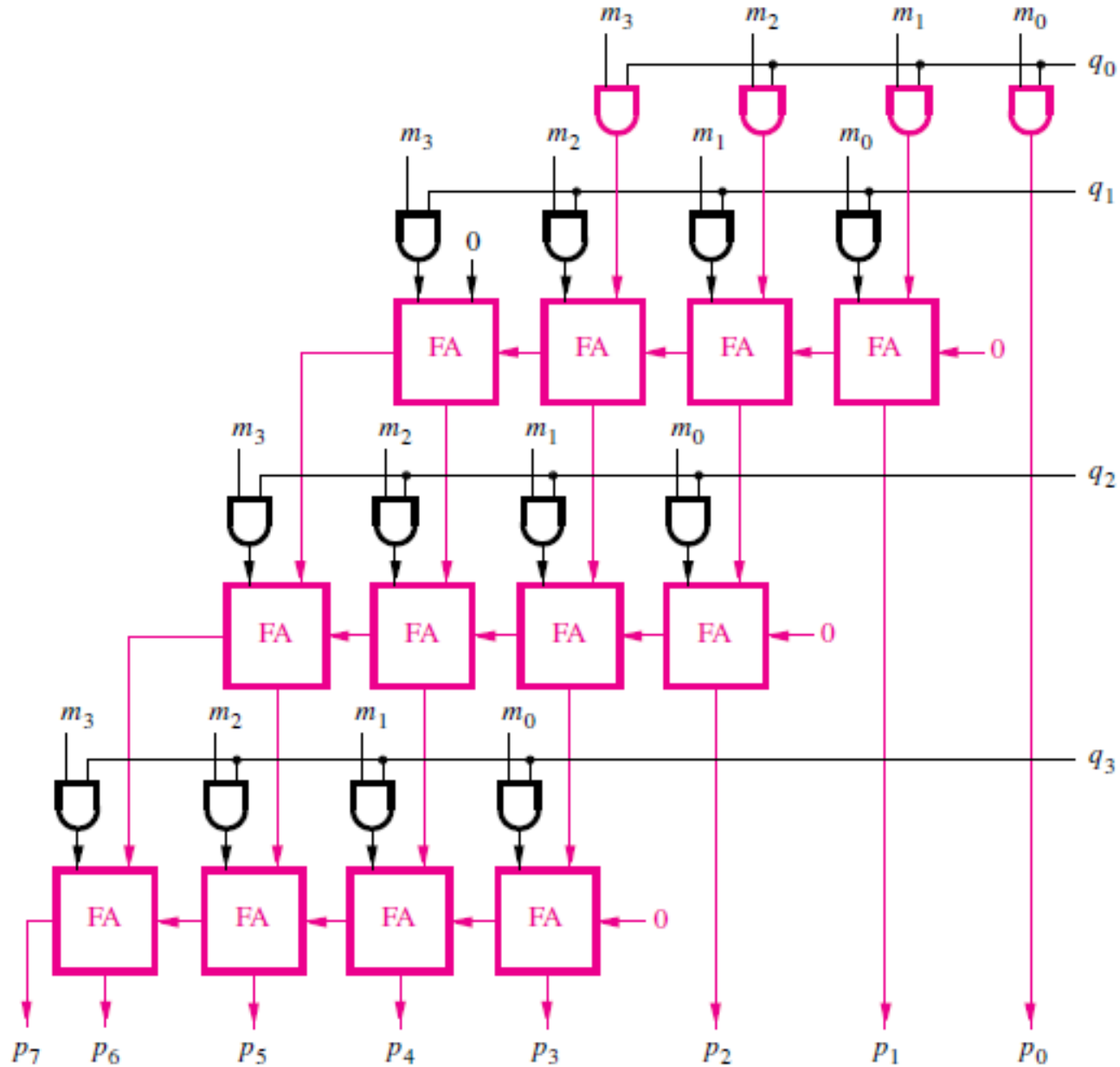
= **X + 2’s complement of Y**

= **X - Y**

Multiplication

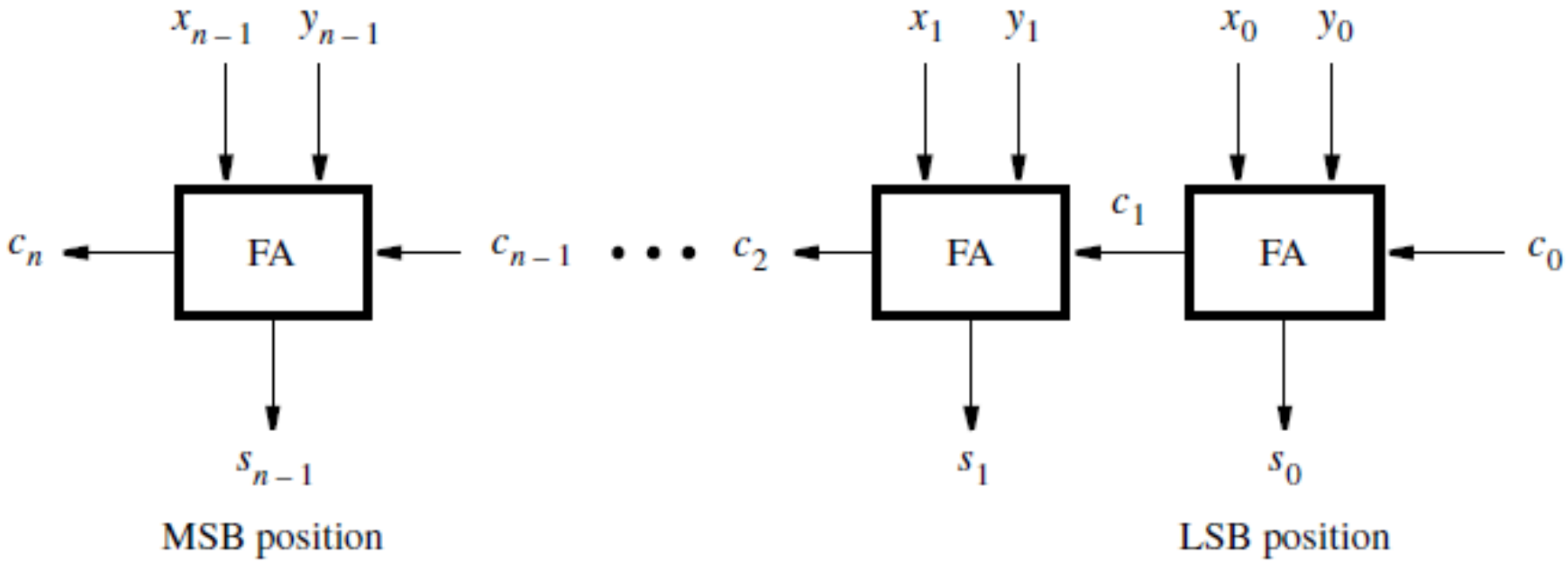
Two binary numbers can be multiplied using the same method as we use for decimal numbers.




m_3 m_2 m_1 m_0 q_0
m_3 m_2 m_1 m_0 q_1
0
FA FA FA FA 0
m_3 m_2 m_1 m_0 q_2
FA FA FA FA 0
m_3 m_2 m_1 m_0 q_3
FA FA FA FA 0
p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0

Hierarchical Verilog Code

For larger designs, it is often convenient to create a hierarchical structure in the Verilog code, in which there is a *top-level* module that includes multiple instances of *lower-level* modules.


x_{n-1} y_{n-1}
x_1 y_1
x_0 y_0
c_n
FA
c_{n-1} • • • c_2
FA
c_1
FA
c_0
s_{n-1}
s_1
s_0
MSB position
LSB position

Let n=4

```
module adder4 (carryin, x3, x2, x1, x0, y3, y2, y1, y0, s3, s2, s1, s0, carryout);
    input carryin, x3, x2, x1, x0, y3, y2, y1, y0;
    output s3, s2, s1, s0, carryout;
    fulladd stage0 (carryin, x0, y0, s0, c1);
    fulladd stage1 (c1, x1, y1, s1, c2);
    fulladd stage2 (c2, x2, y2, s2, c3);
    fulladd stage3 (c3, x3, y3, s3, carryout);
endmodule

module fulladd (Cin, x, y, s, Cout);
    input Cin, x, y;
    output s, Cout;
    assign s = x ∧ y ∧ Cin;
    assign Cout = (x & y) | (x & Cin) | (y & Cin);
endmodule
```

- A Verilog module can be included as a subcircuit in another module.
- The general form of a module instantiation statement is given below.

```
module_name  instance_name (.port_name ( [expression] )
{, .port_name ( [expression] )} );
```

- The ***instance_name*** can be any legal Verilog identifier and the port connections specify how the module is connected to the rest of the circuit.
- The same module can be instantiated multiple times in a given design provided that each instance name is unique.
- Each port_name is the name of a port in the subcircuit, and each expression specifies a connection to that port.

- **Named port connections** -The syntax .port_name is provided so that the order of signals listed in the instantiation statement does not have to be the same as the order of the ports given in the module statement of the subcircuit.
- **Ordered port connections**-If the port connections are given in the same order as in the subcircuit, then .port_name is not needed.

Using Vectored Signals

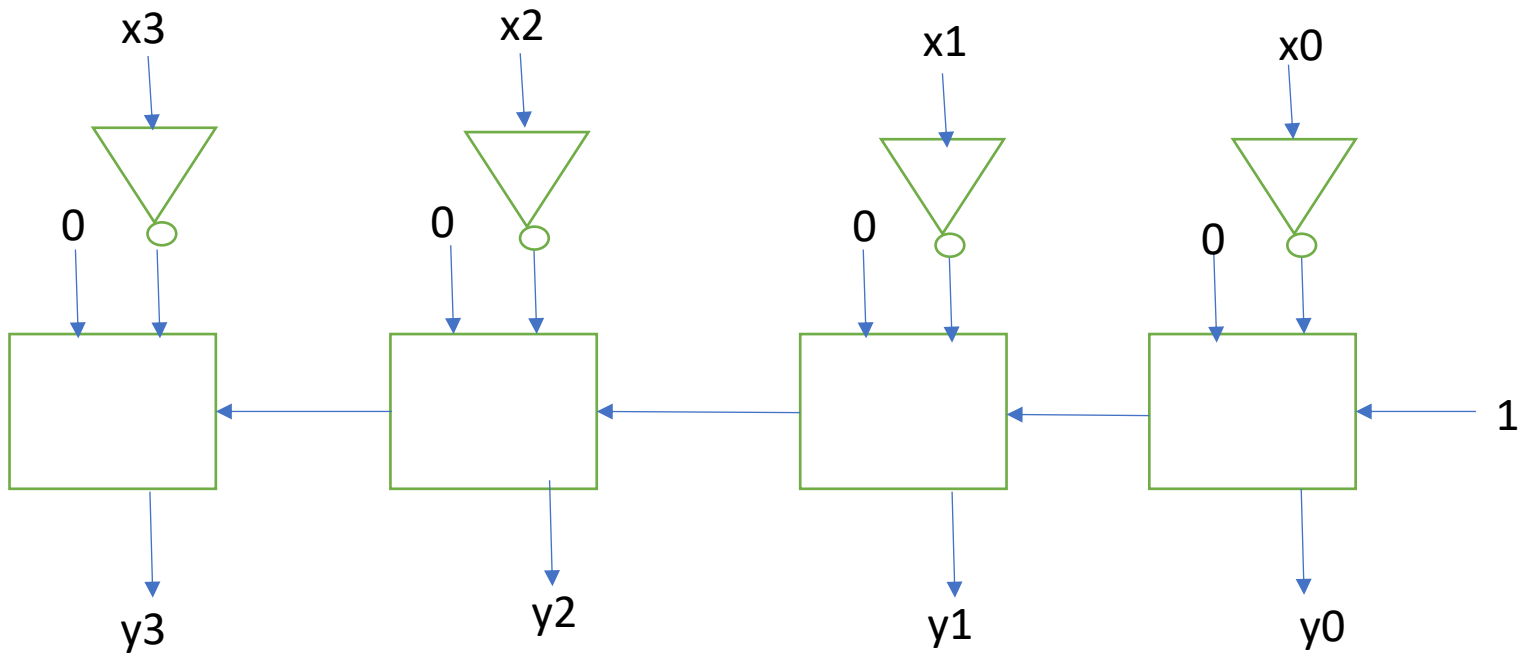
- Multibit signals are called *vectors*.
- An example of an input vector is

  input [3:0] W;
- This statement defines *W* to be a four-bit vector. Its individual bits can be referred to using an index value in square brackets.
- The most-significant bit (MSB) is referred to as *W*[3] and the least-significant bit (LSB) is *W*[0].
- **input** [0:3] W; Here *W*[0] is MSB and *W*[3] LSB

```
module adder4 (carryin, X, Y, S, carryout);
input carryin;
input [3:0] X, Y;
output [3:0] S;
output carryout;
wire [3:1] C;
fulladd stage0 (carryin, X[0], Y[0], S[0], C[1]);
fulladd stage1 (C[1], X[1], Y[1], S[1], C[2]);
fulladd stage2 (C[2], X[2], Y[2], S[2], C[3]);
fulladd stage3 (.Cout(carryout), .s(S[3]), .y(Y[3]), .x(X[3]), .Cin(C[3]));
endmodule
```

Exercise 1

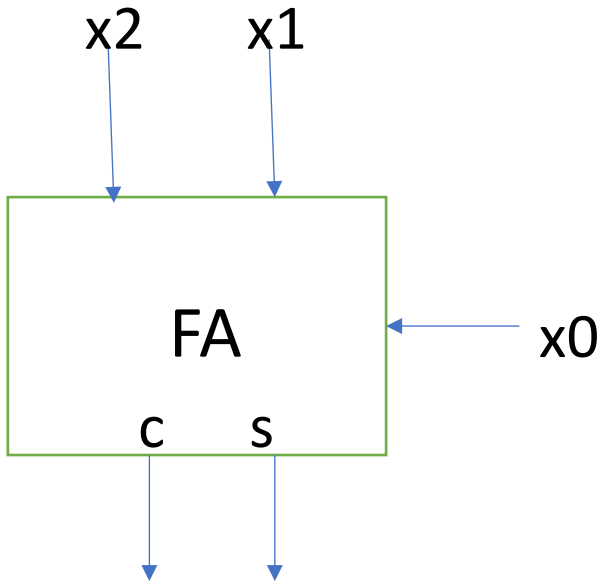
Design a circuit using full adders to generate 2's complement of a 4 bit number X



y3y2y1y0 is the 2's complement of X

Exercise 2

Design a circuit using full adders to find the no. of 1's in a three bit unsigned number



CS is the result in binary.

Binary-Coded-Decimal Representation

Table 3.3 Binary-coded decimal digits.

Decimal digit	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

BCD Addition

X	0 1 1 1	7
+ Y	+ 0 1 0 1	+ 5
Z	1 1 0 0	12
	+ 0 1 1 0	
carry →	1 0 0 1 0	
	S = 2	

X	1 0 0 0	8
+ Y	+ 1 0 0 1	+ 9
Z	1 0 0 0 1	17
	+ 0 1 1 0	
carry →	1 0 1 1 1	
	S = 7	

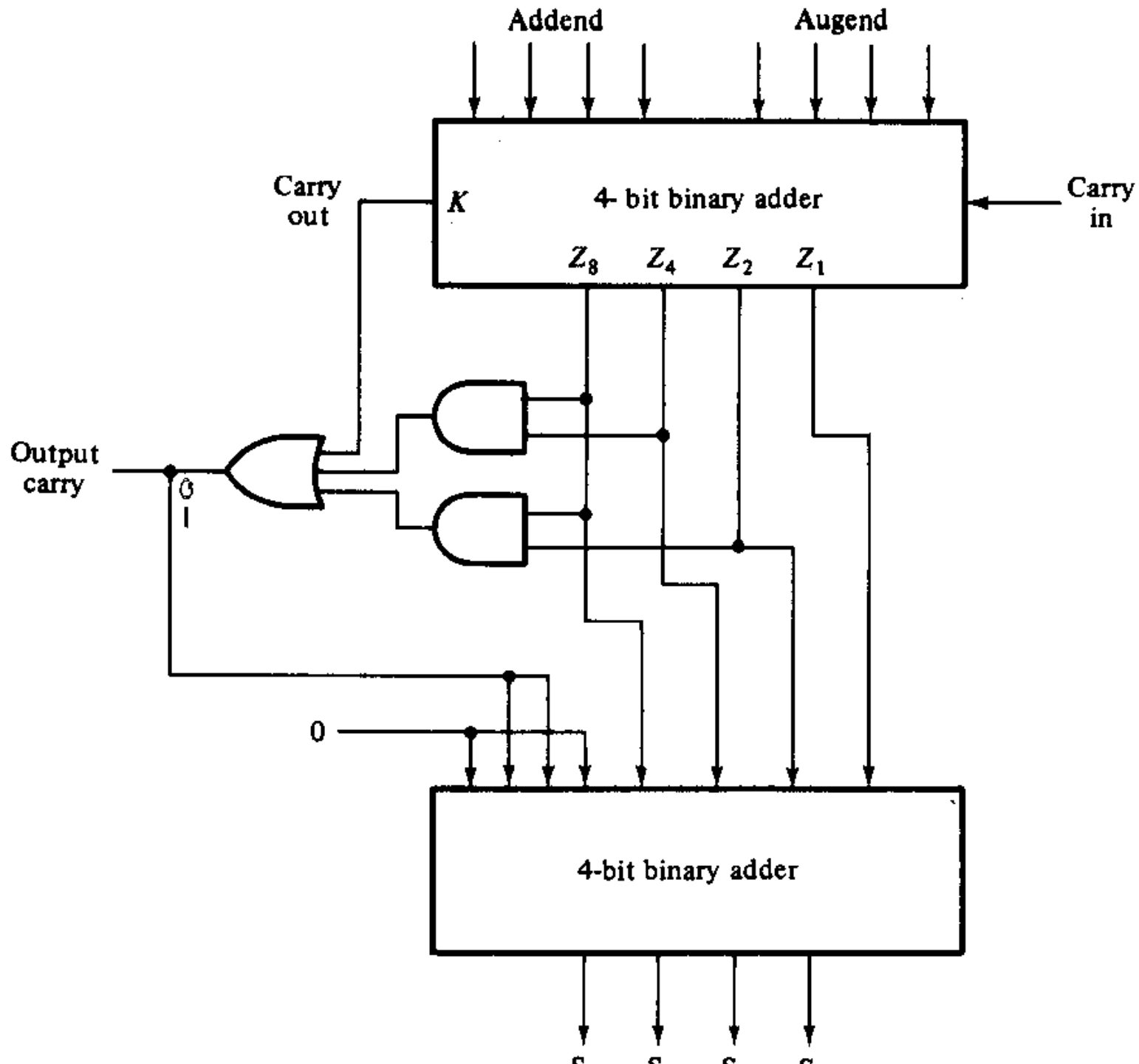
Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

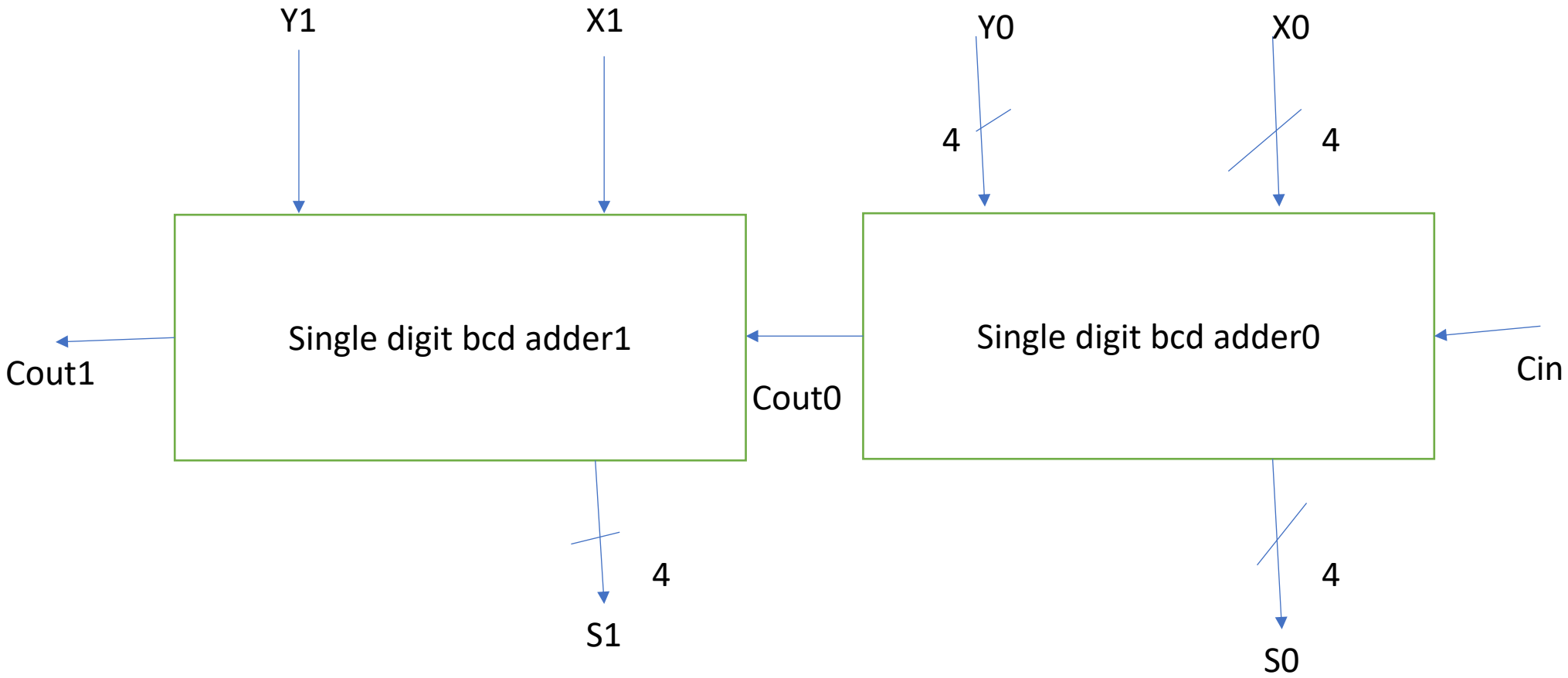
z8, z4 \ z2, z1	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

Correction should be done when the carry k = 1 or when the expression z8z4 + z8z2 evaluates to 1. Therefore the expression for the correction circuit is

$$C = K + Z_8 Z_4 + Z_8 Z_2$$

$$C = K + Z_8 Z_4 + Z_8 Z_2$$


Addend
Augend
Carry out
K
4- bit binary adder
Carry in
Z_8
Z_4
Z_2
Z_1
Output carry
0
1
0
4-bit binary adder


Y1
X1
Y0
X0
4
4
Single digit bcd adder1
Single digit bcd adder0
Cout1
Cout0
Cin
4
4
S1
S0