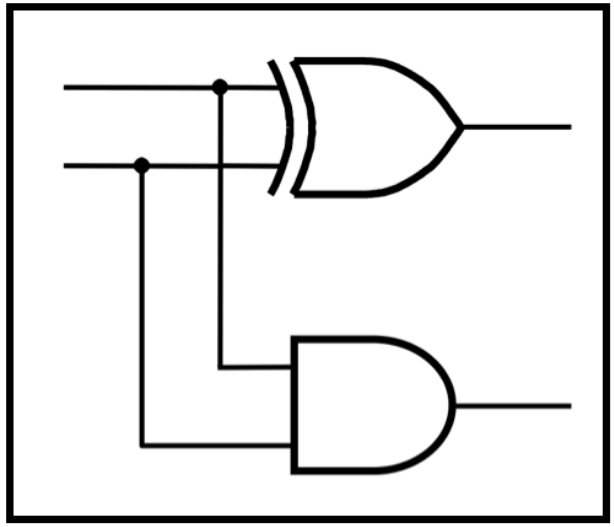


CprE 281: Digital Logic

Instructor: Alexander Stoytchev

http://www.ece.iastate.edu/~alexs/classes/

Multiplication

CprE 281: Digital Logic
Iowa State University, Ames, IA

Administrative Stuff

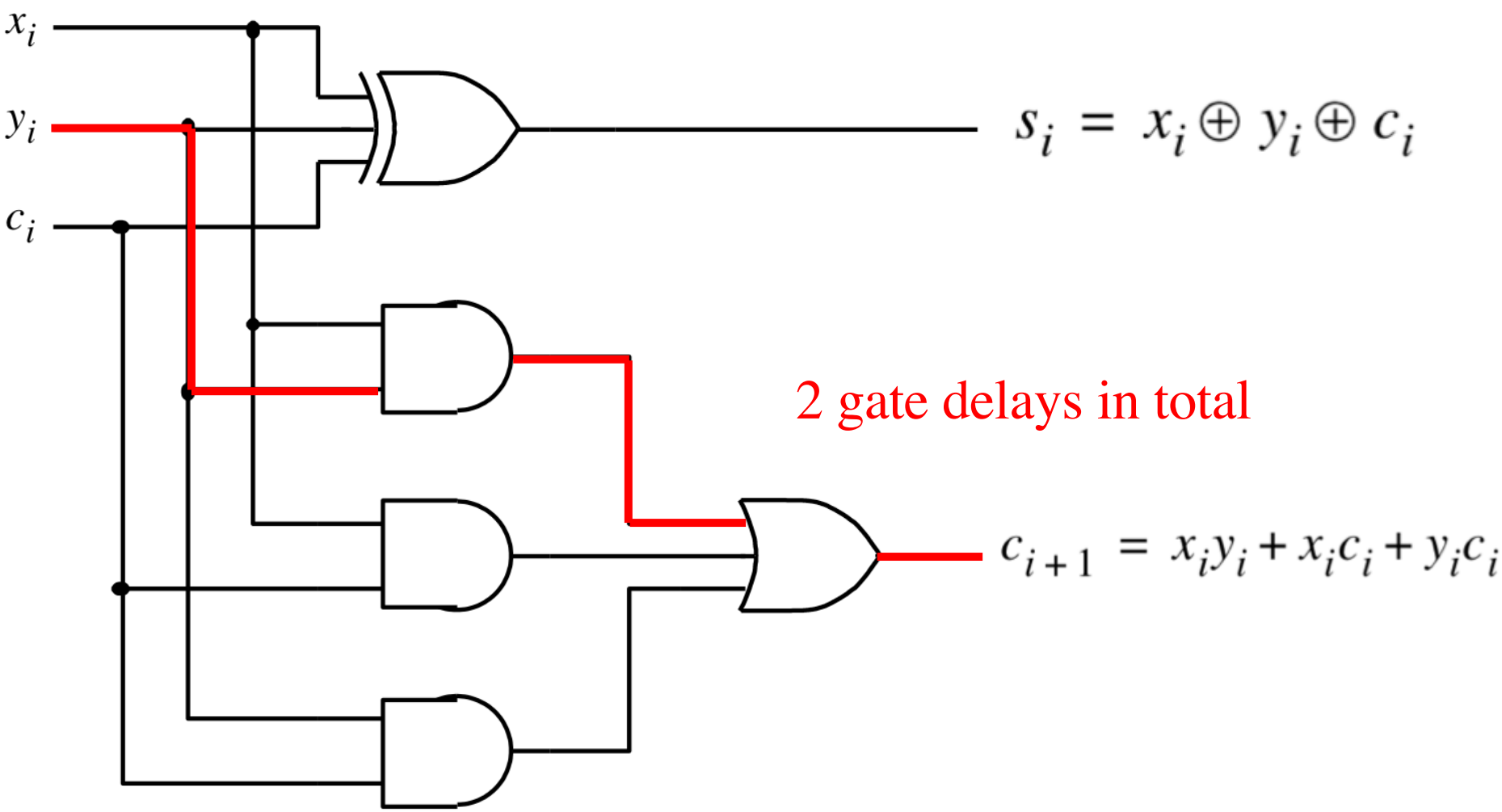
- **No HW is due today**
- **HW 6 will be due on Monday Oct. 11.**

Administrative Stuff

- **Labs this week**
- **Mini-Project**
- **This is worth 3% of your grade (x2 labs)**
- **https://www.ece.iastate.edu/~alexs/classes/2021_Fall_281/labs/Project-Mini/**

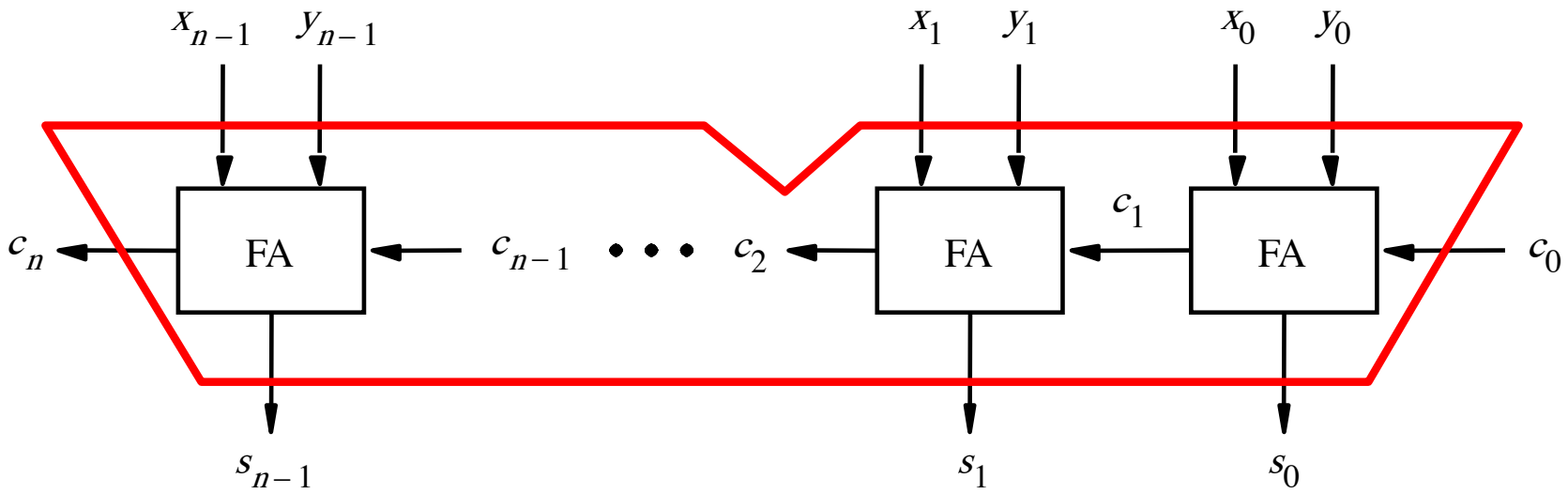
Quick Review

Delays through the Full-Adder circuit



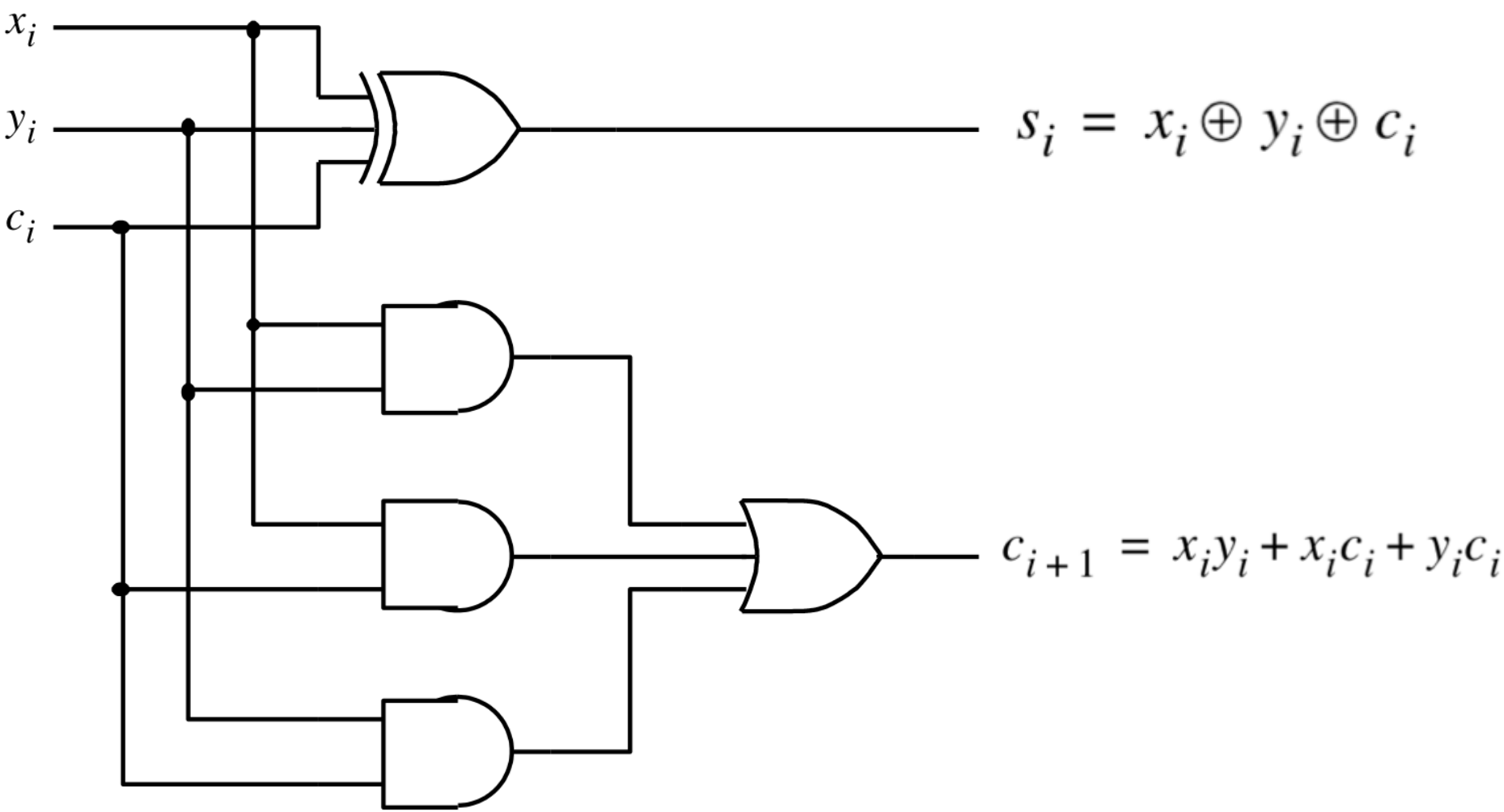
[Figure 3.3c from the textbook]

How long does it take to compute all sum bits and all carry bits?



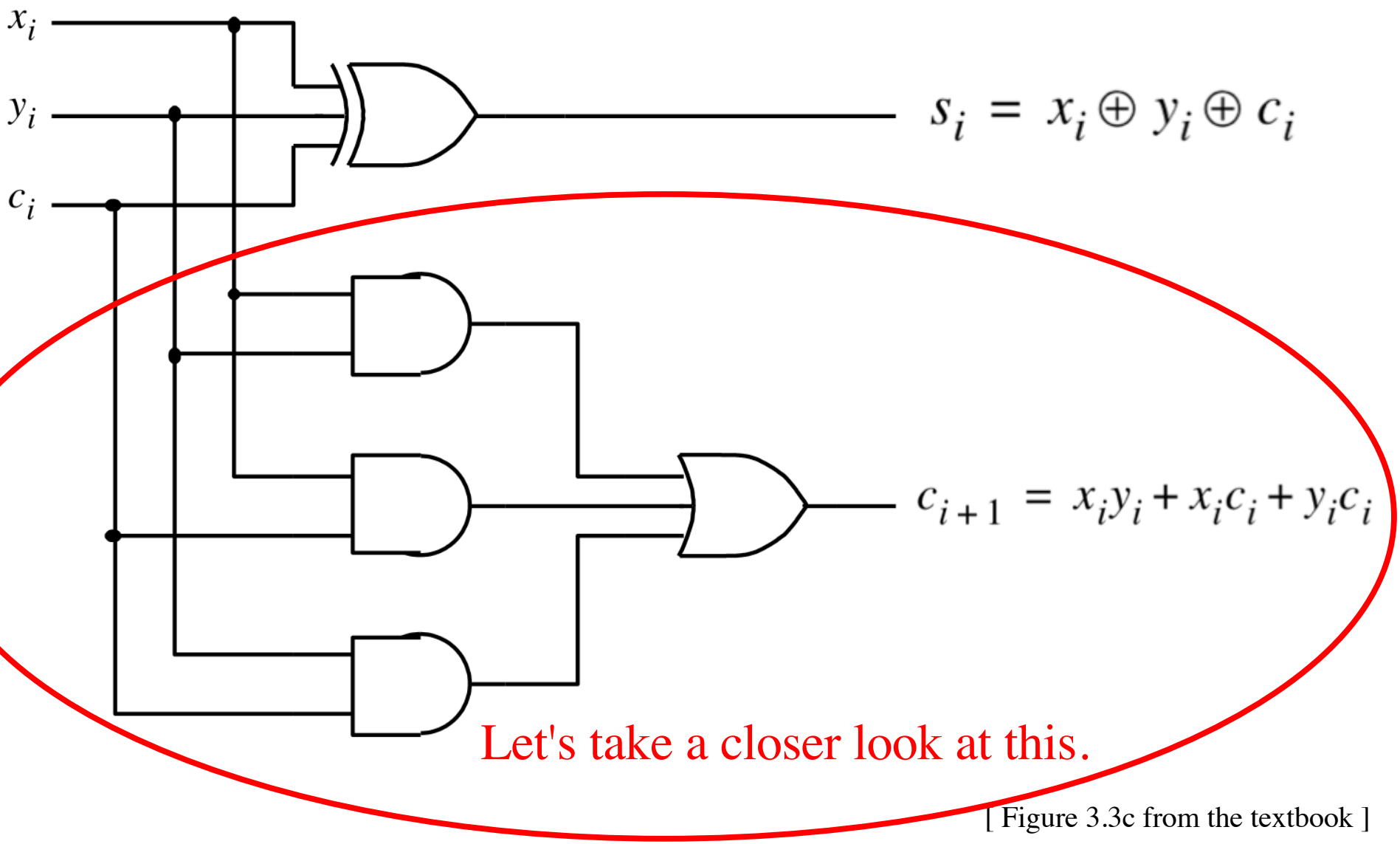
It takes 2n gate delays using a ripple-carry adder?

The Full-Adder Circuit



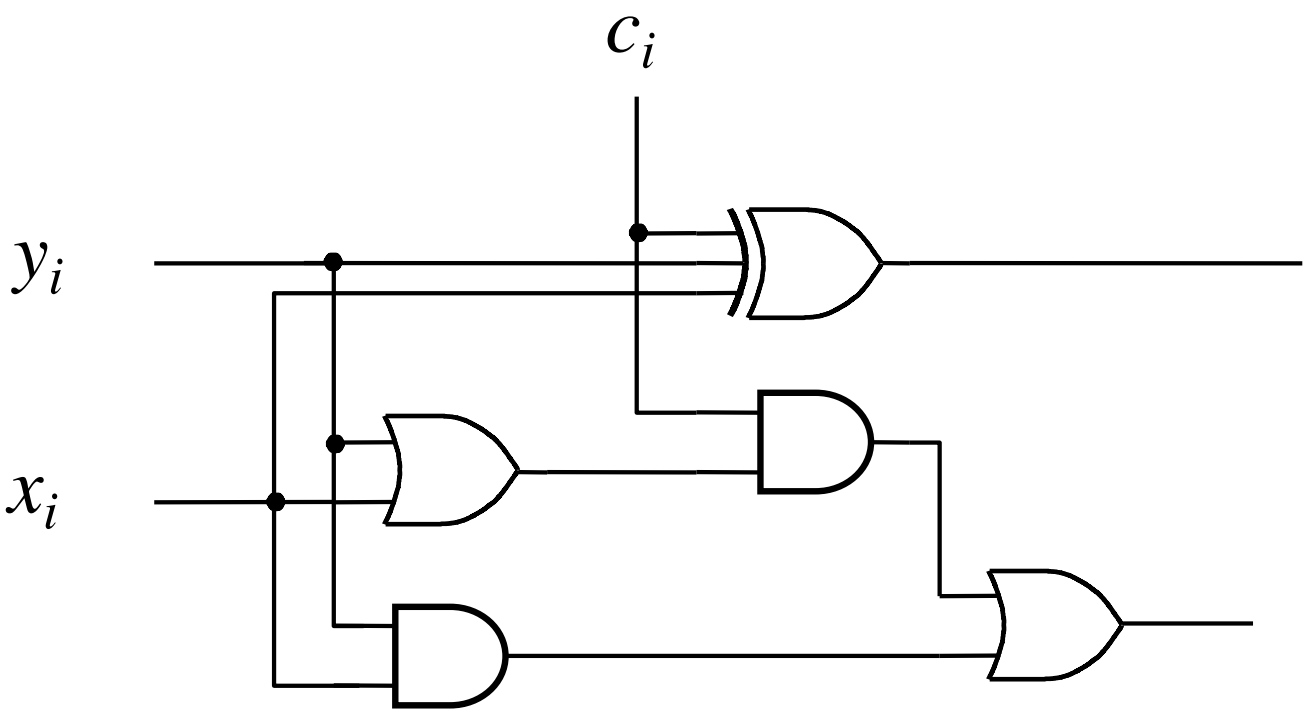
[Figure 3.3c from the textbook]

The Full-Adder Circuit



[Figure 3.3c from the textbook]

Another Way to Draw the Full-Adder Circuit



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

$$c_{i+1} = x_i\, y_i + (x_i + y_i)c_i$$

Decomposing the Carry Expression

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

Decomposing the Carry Expression

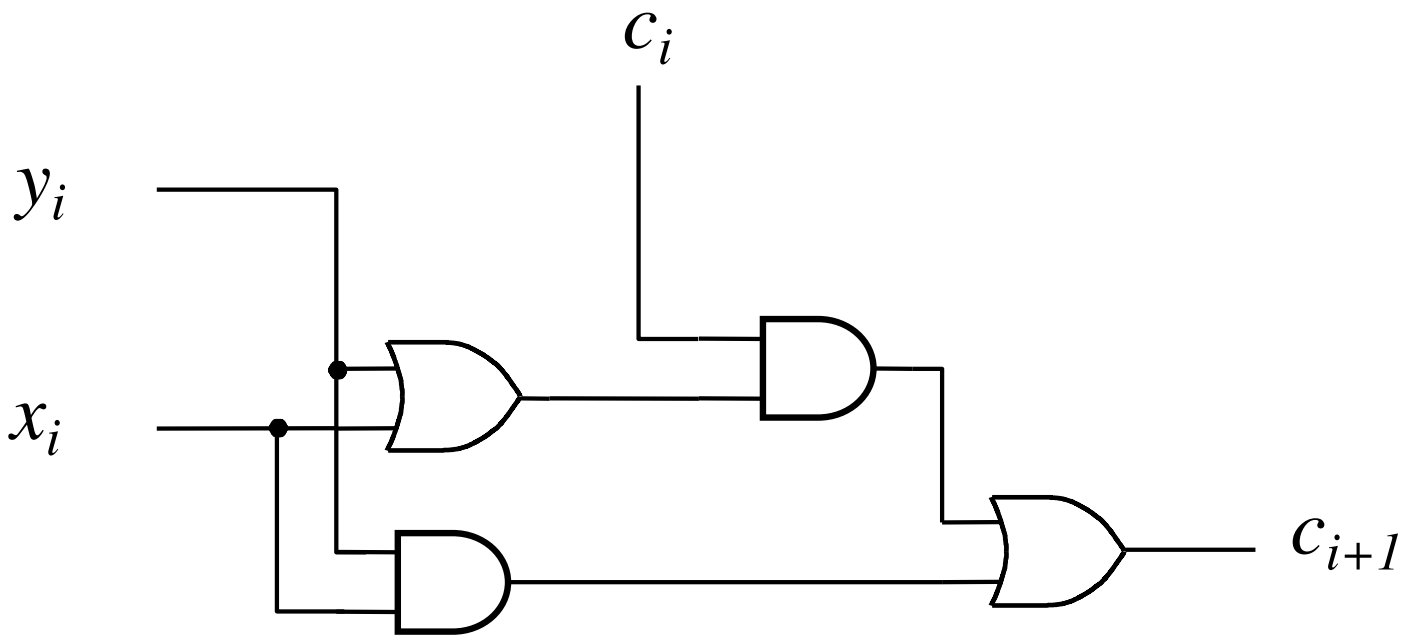
$$c_{i+1} = x_i\, y_i + x_i\, c_i + y_i\, c_i$$

$$c_{i+1} = x_i\, y_i + (x_i + y_i)c_i$$

Decomposing the Carry Expression

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

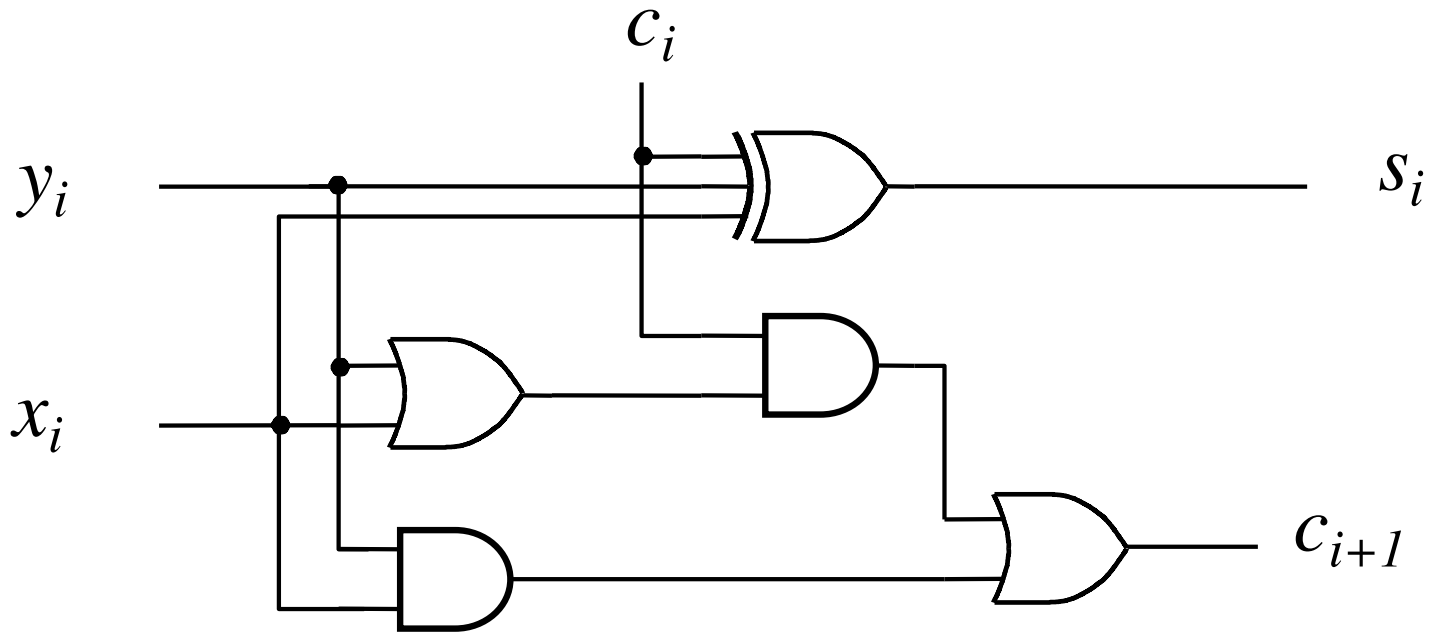
$$c_{i+1} = x_i\, y_i + (x_i + y_i)\, c_i$$



Another Way to Draw the Full-Adder Circuit

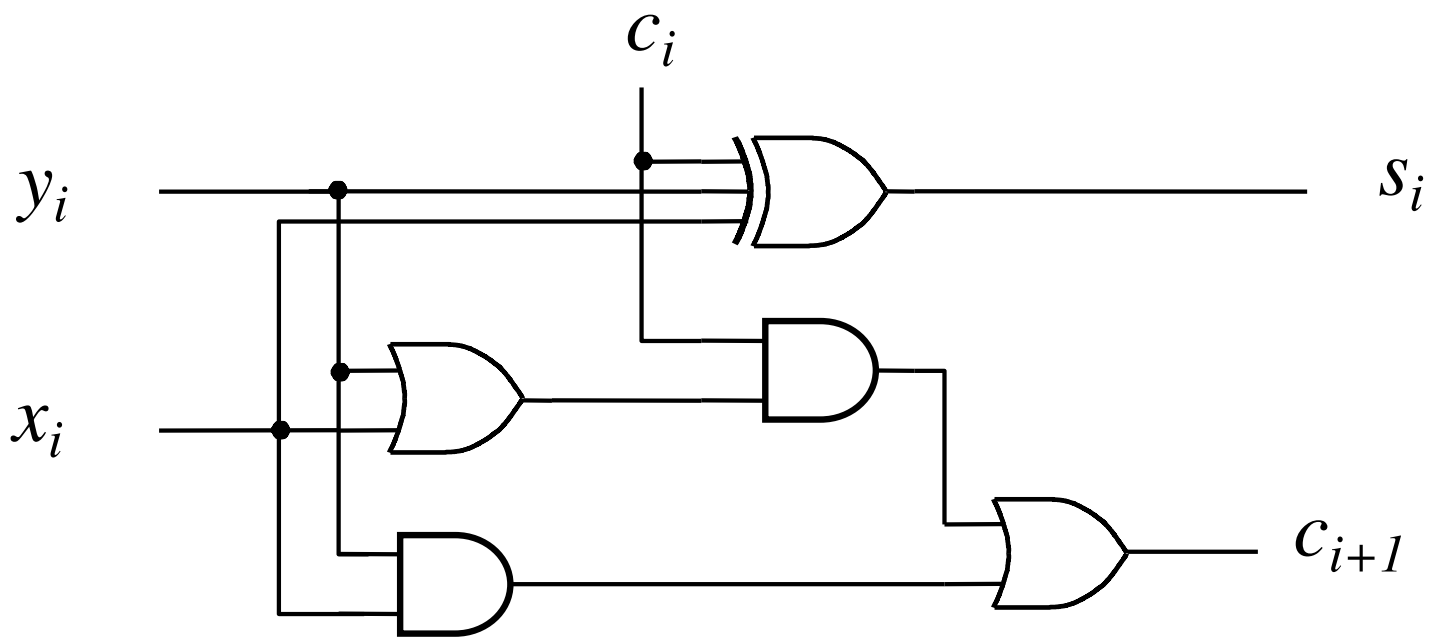
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

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



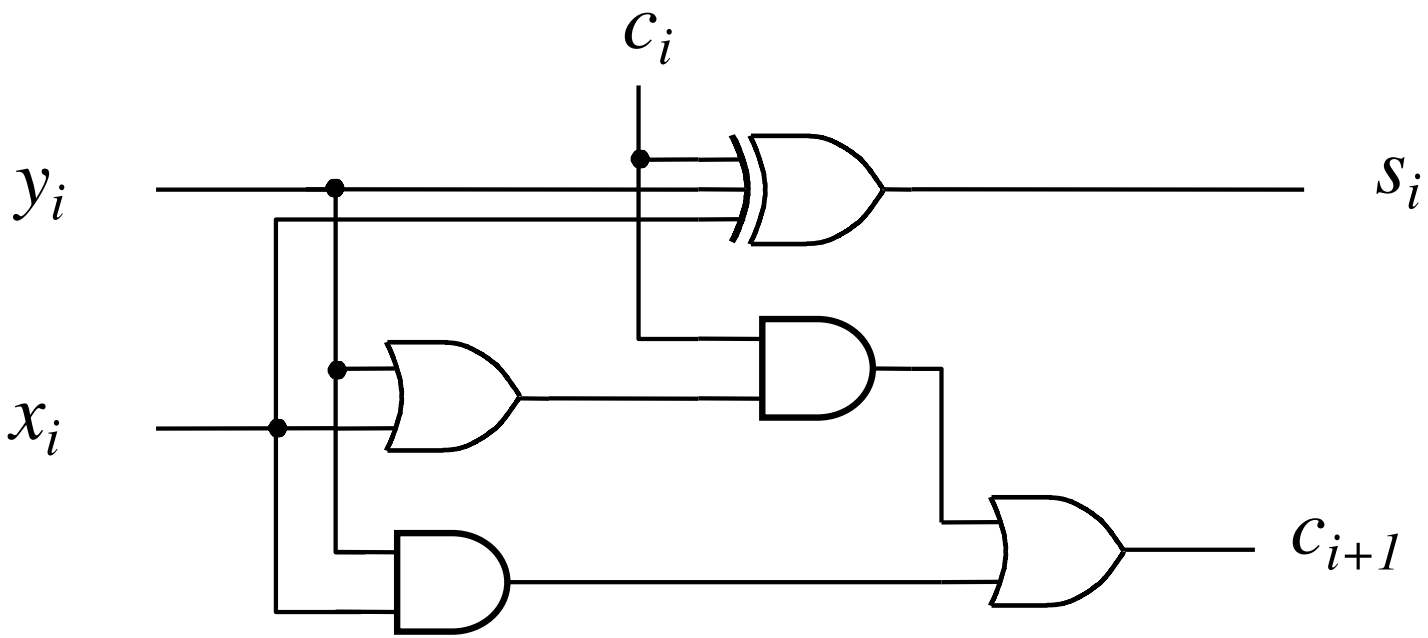
Another Way to Draw the Full-Adder Circuit

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



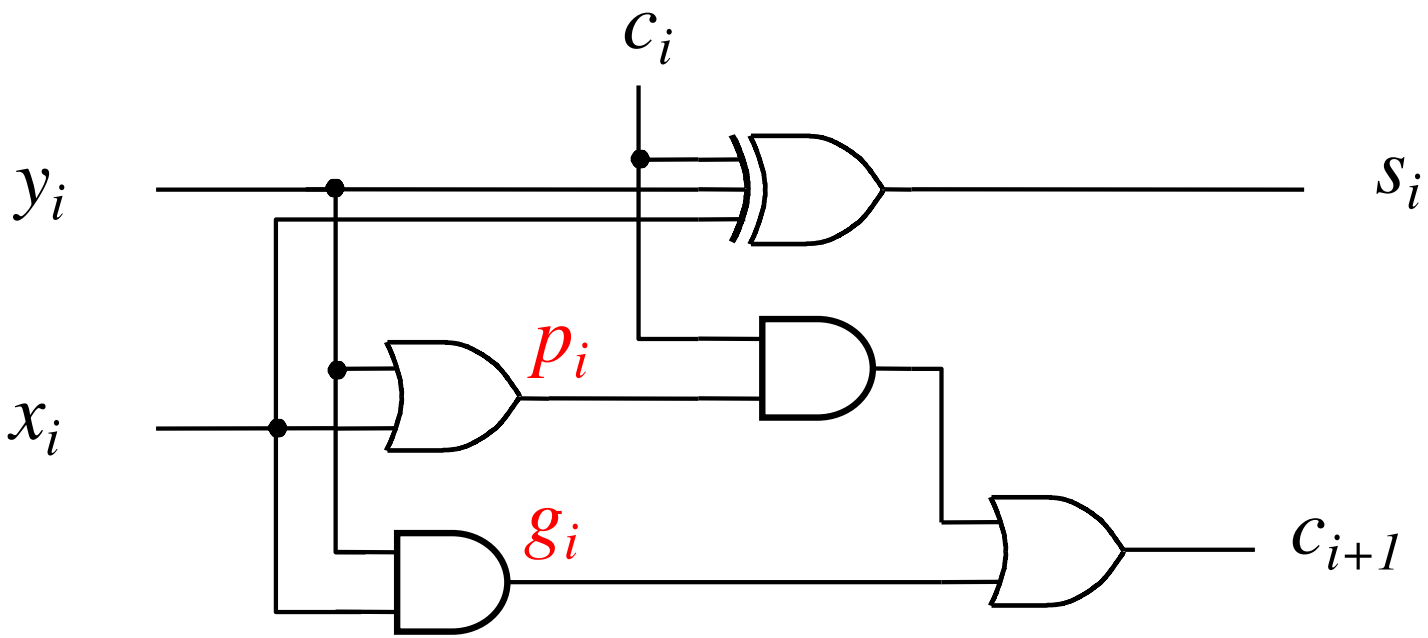
Another Way to Draw the Full-Adder Circuit

$$c_{i+1} = \underbrace{x_i y_i}_{g_i} + \underbrace{(x_i + y_i)}_{p_i} c_i$$

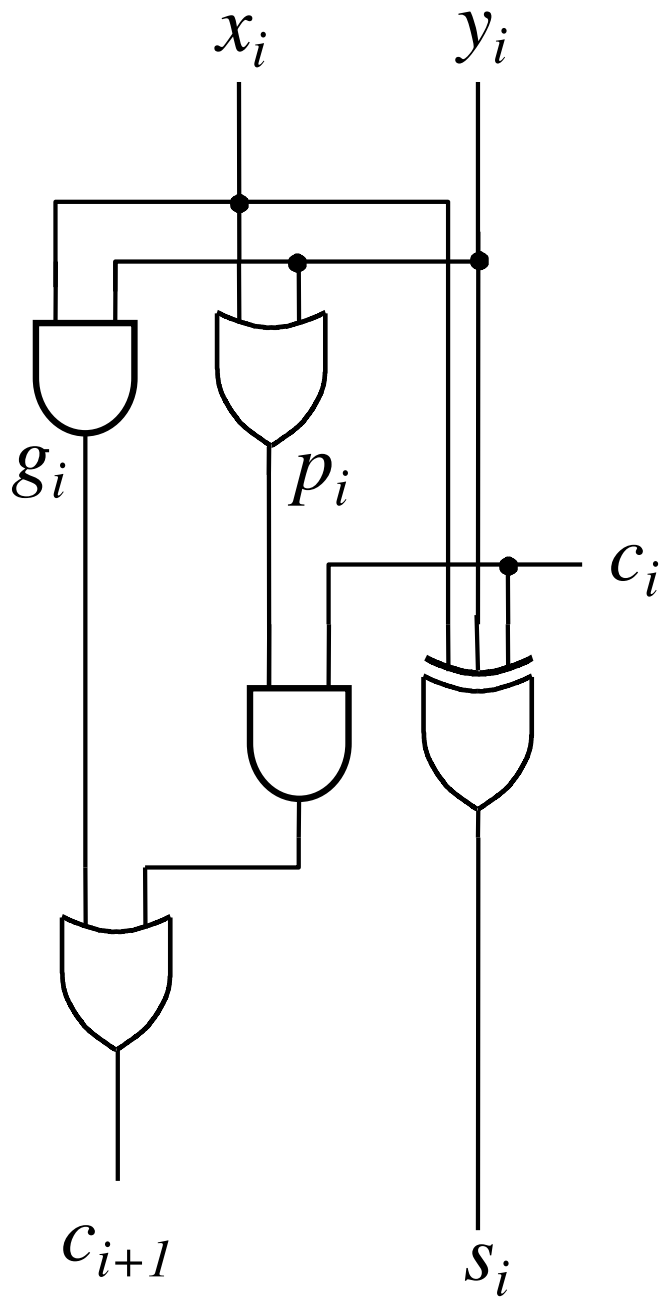


Another Way to Draw the Full-Adder Circuit

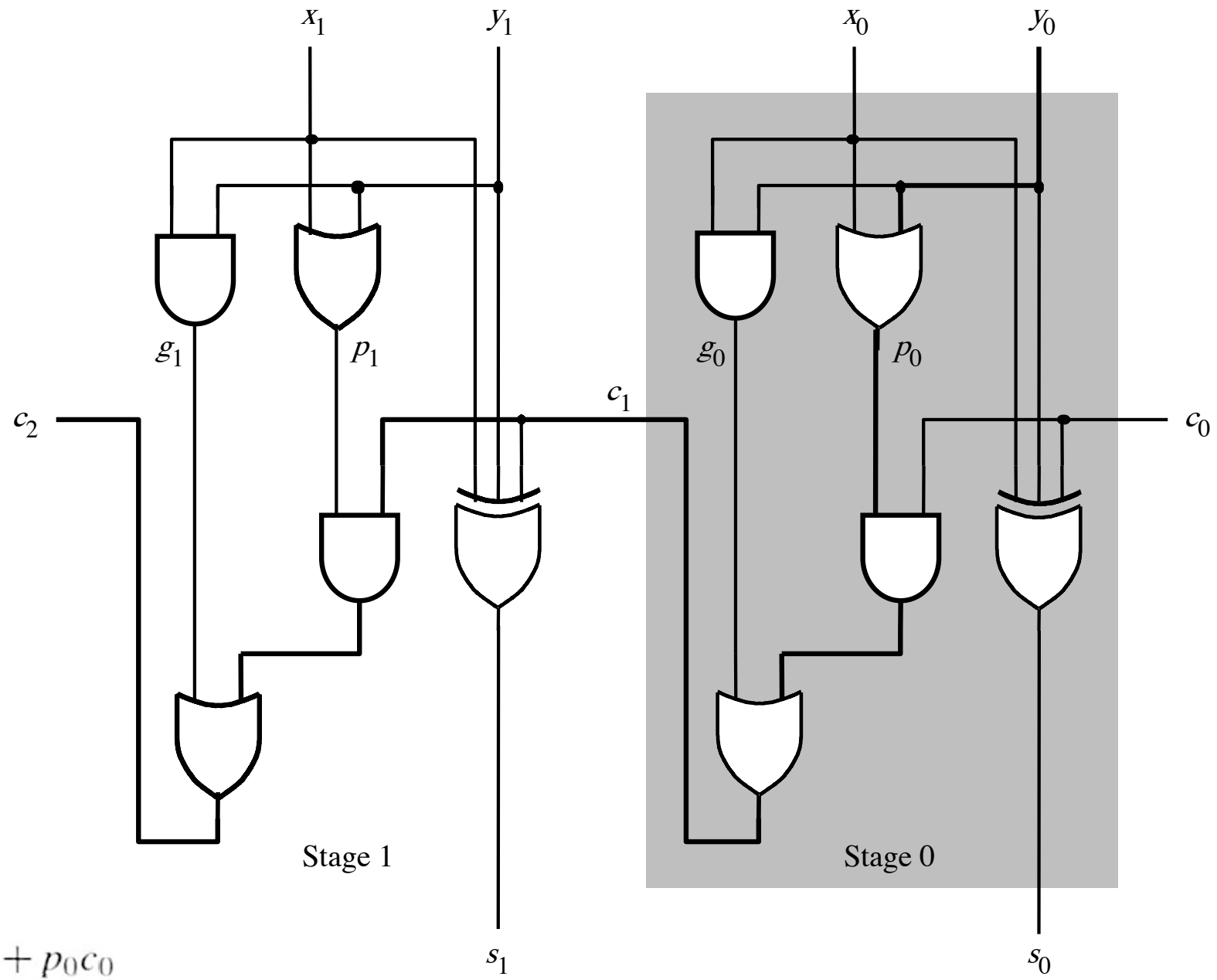
$$c_{i+1} = \underbrace{x_i y_i}_{g_i} + \underbrace{(x_i + y_i)}_{p_i} c_i$$



Yet Another Way to Draw It (Just Rotate It)



Now we can Build a Ripple-Carry Adder

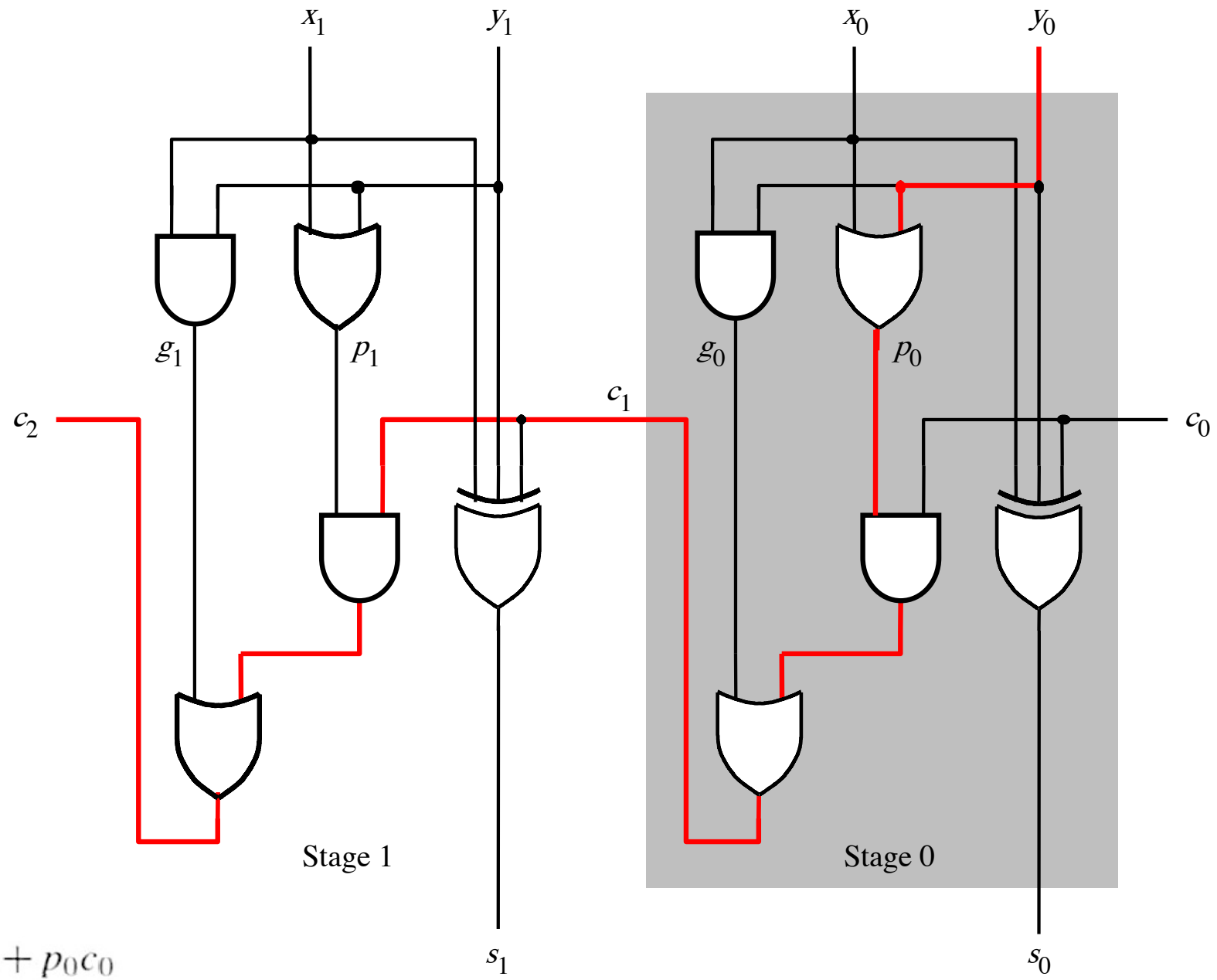


$$c_1 = g_0 + p_0 c_0$$

$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

[Figure 3.14 from the textbook]

Now we can Build a Ripple-Carry Adder

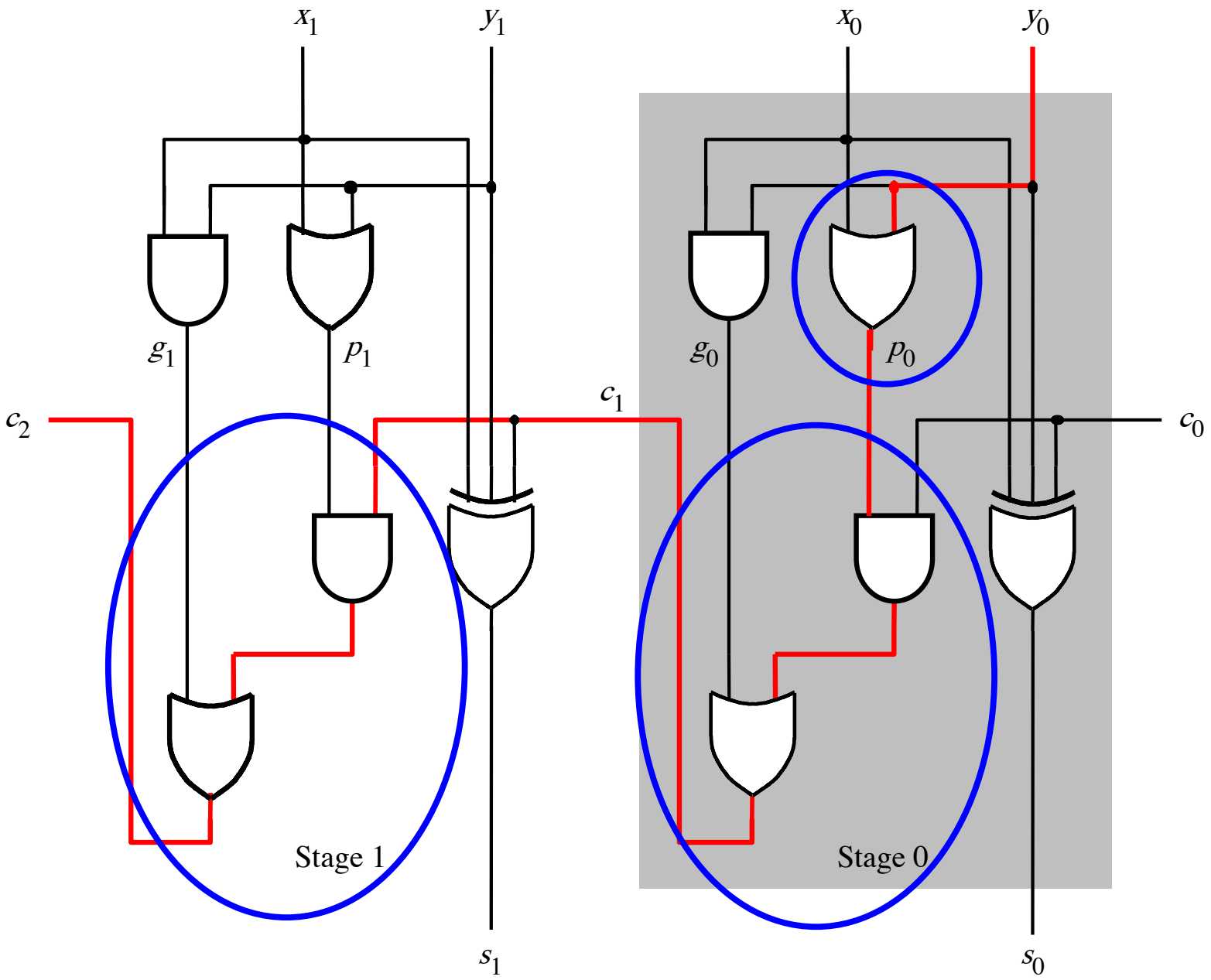


$$c_1 = g_0 + p_0 c_0$$

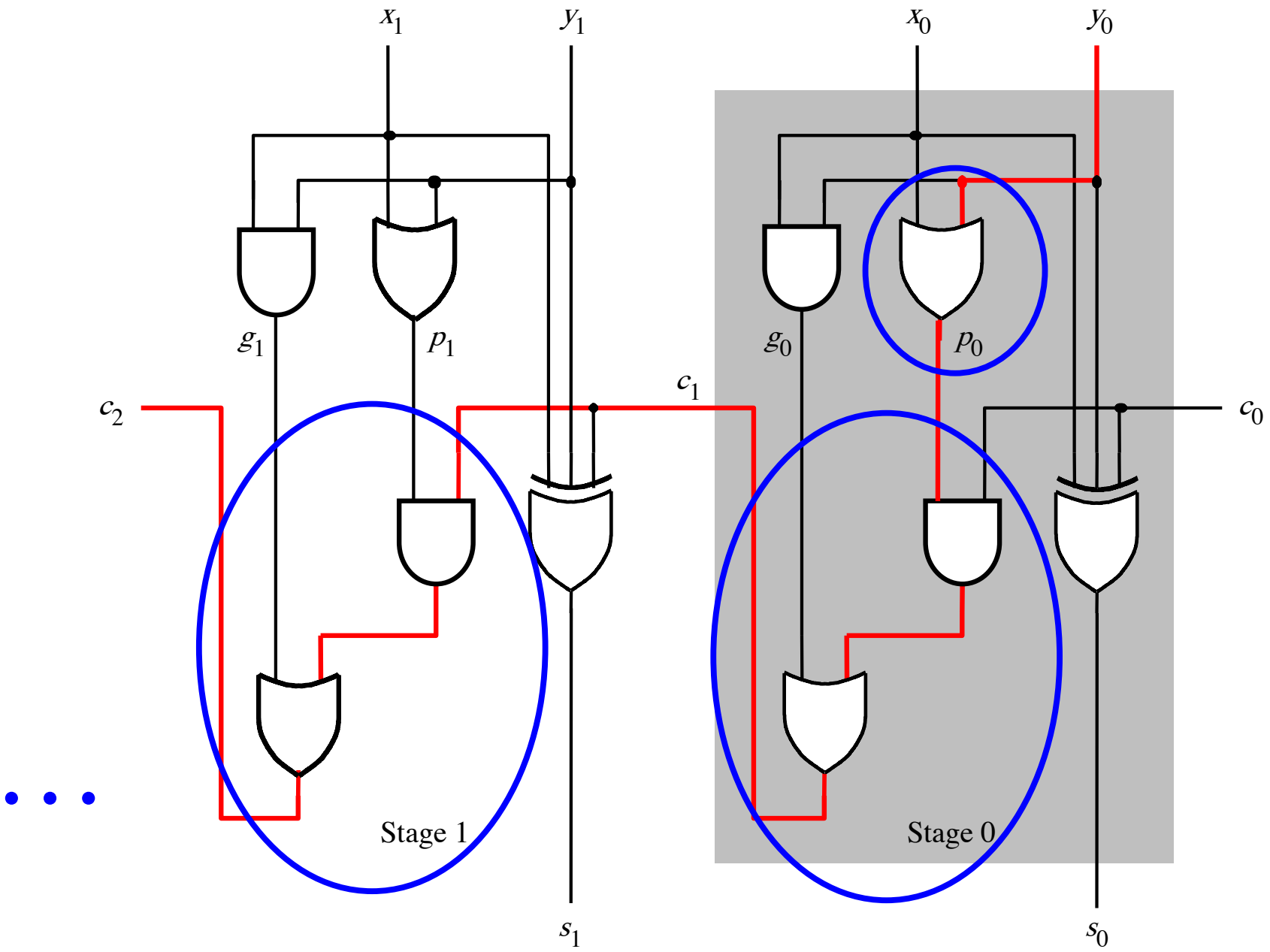
$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

[Figure 3.14 from the textbook]

The delay is 5 gates (1+2+2)



n-bit ripple-carry adder: 2n+1 gate delays



Decomposing the Carry Expression

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

$$c_{i+1} = \underbrace{x_i y_i}_{g_i} + \underbrace{(x_i + y_i)}_{p_i} c_i$$

$$c_{i+1} = g_i + p_i c_i$$

$$c_{i+1} = g_i + p_i (g_{i-1} + p_{i-1} c_{i-1})$$

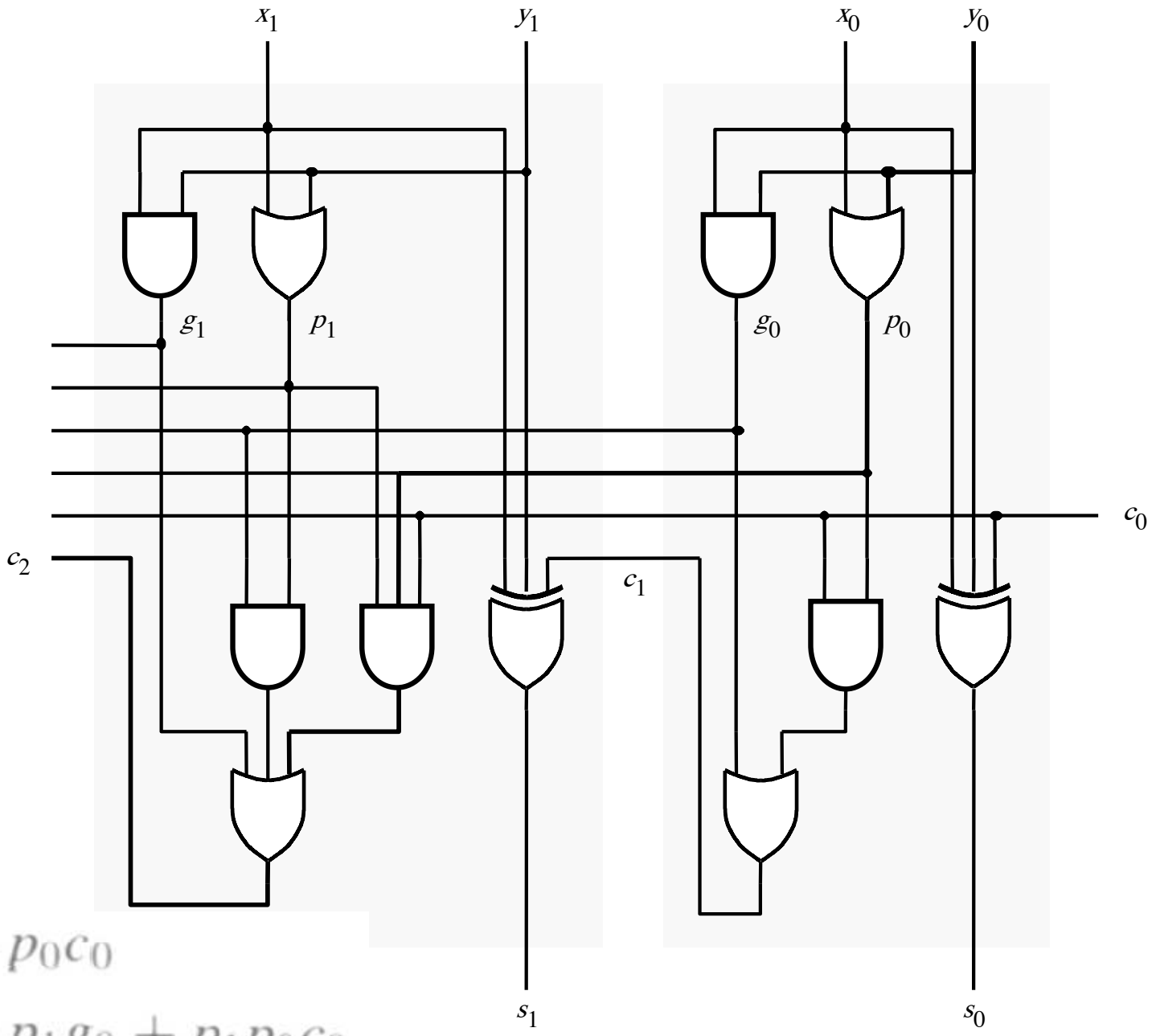
$$= g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-1}$$

Carry for the first two stages

$$c_1 = g_0 + p_0 c_0$$

$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

The first two stages of a carry-lookahead adder

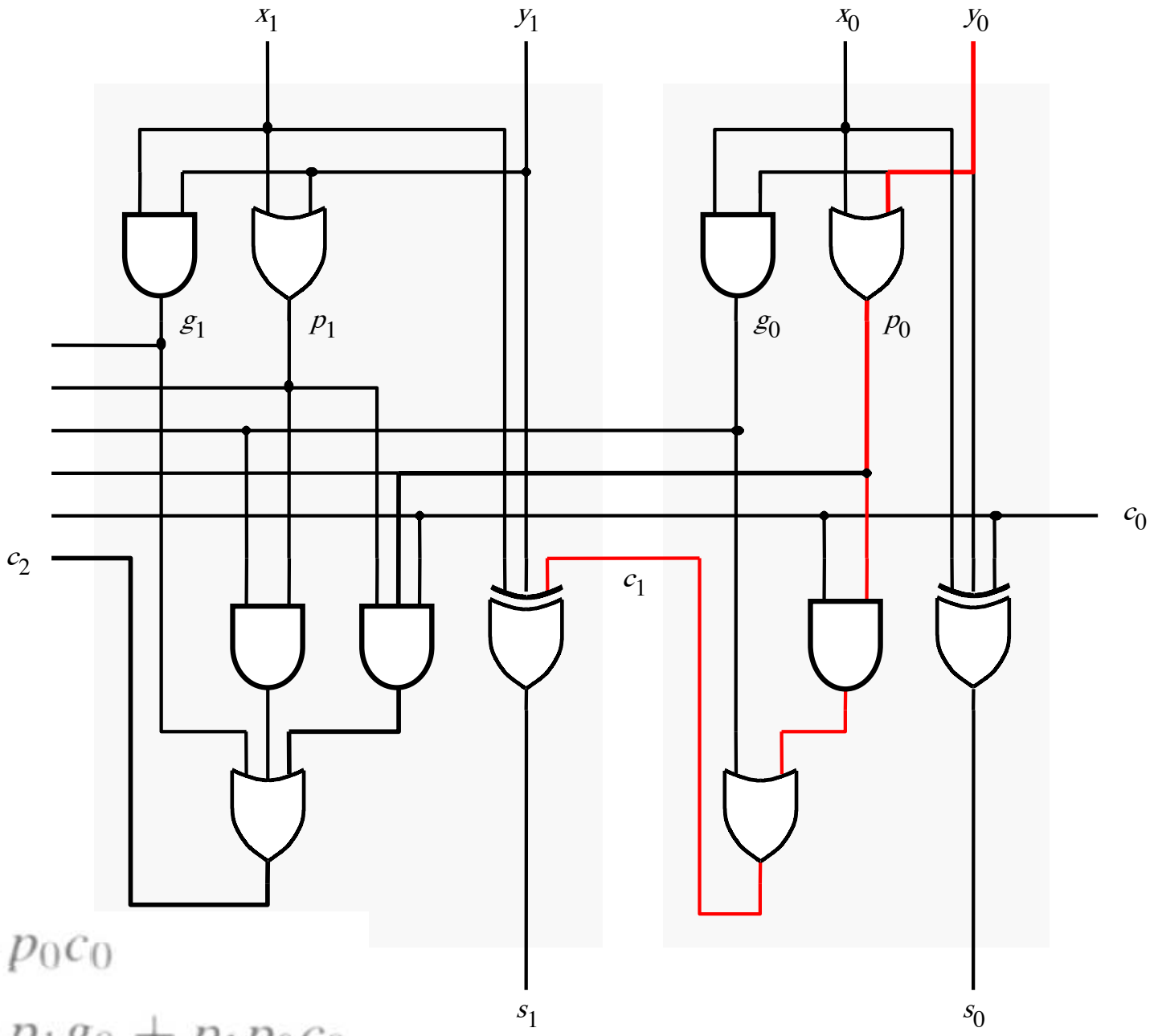


$$c_1 = g_0 + p_0 c_0$$

$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

[Figure 3.15 from the textbook]

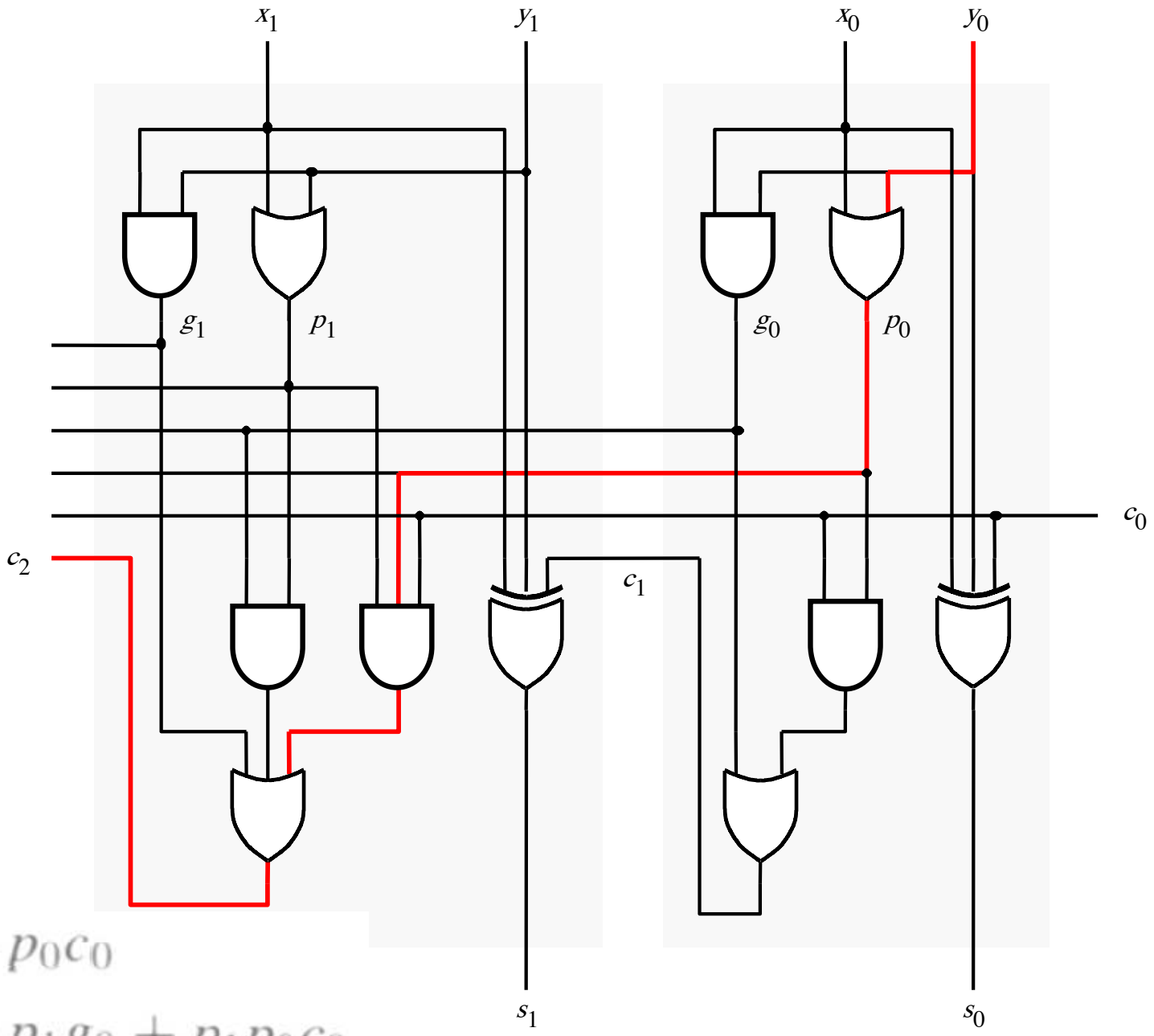
It takes 3 gate delays to generate c_1



$$c_1 = g_0 + p_0 c_0$$

$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

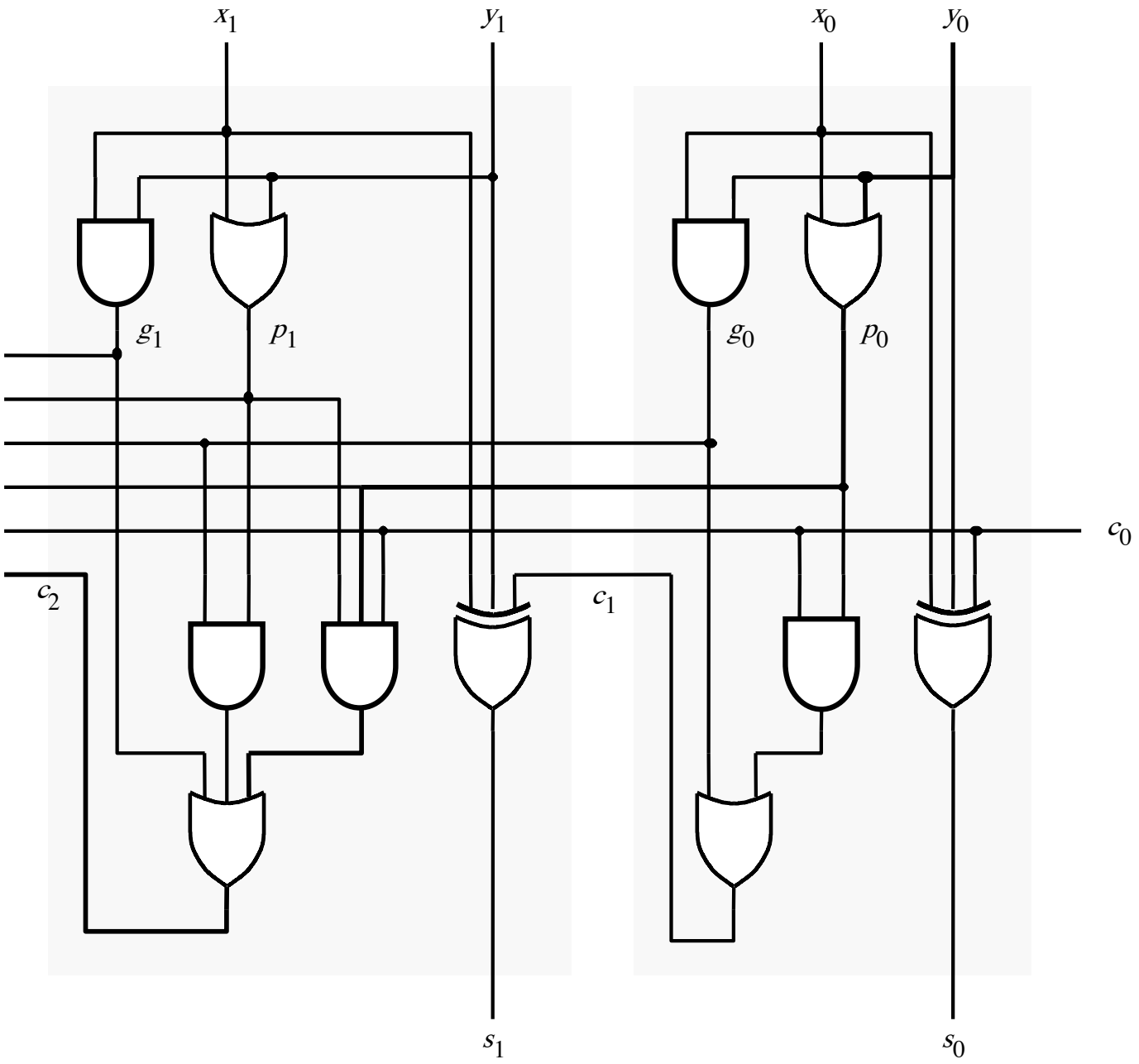
It takes 3 gate delays to generate c_2



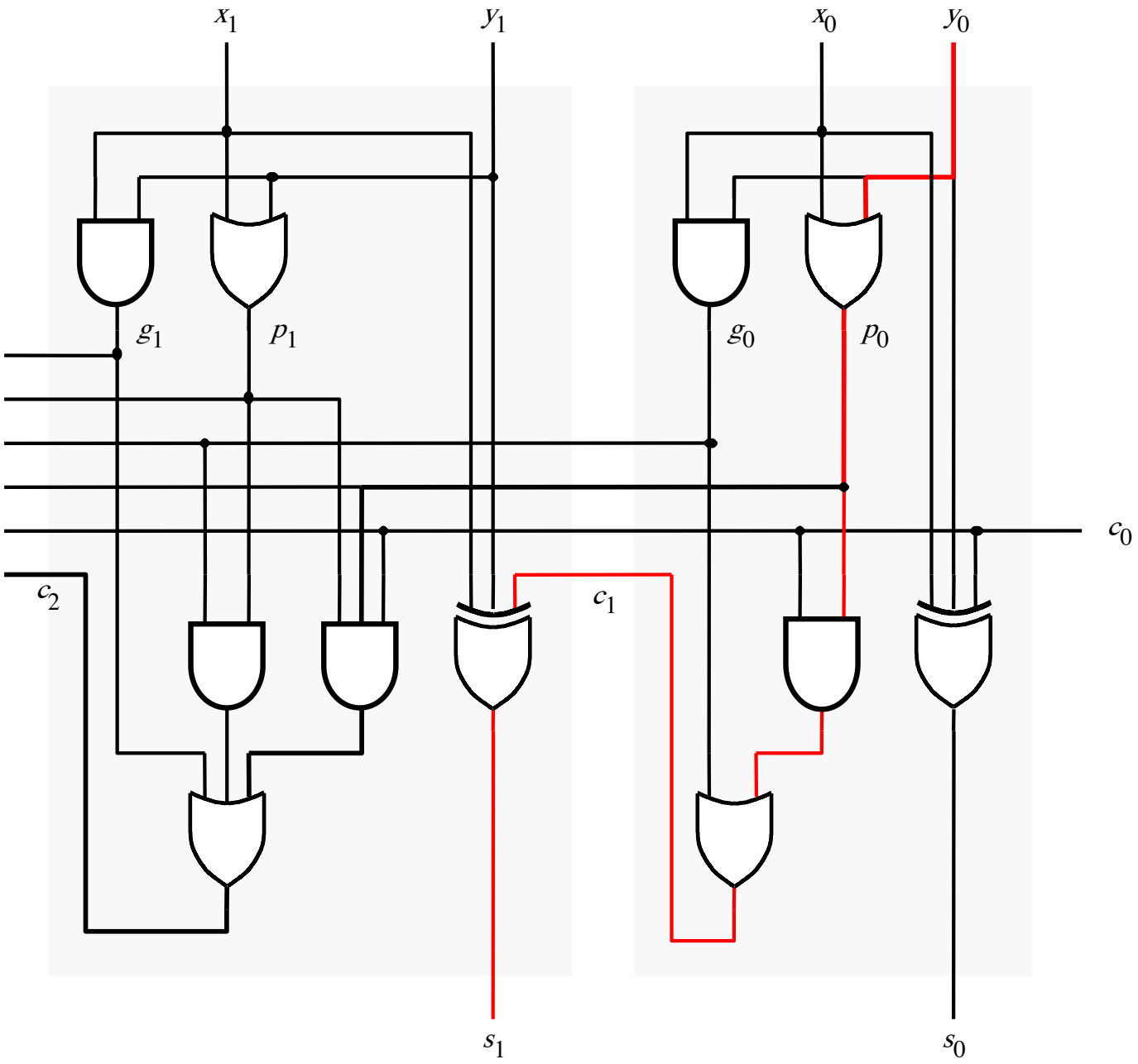
$$c_1 = g_0 + p_0 c_0$$

$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

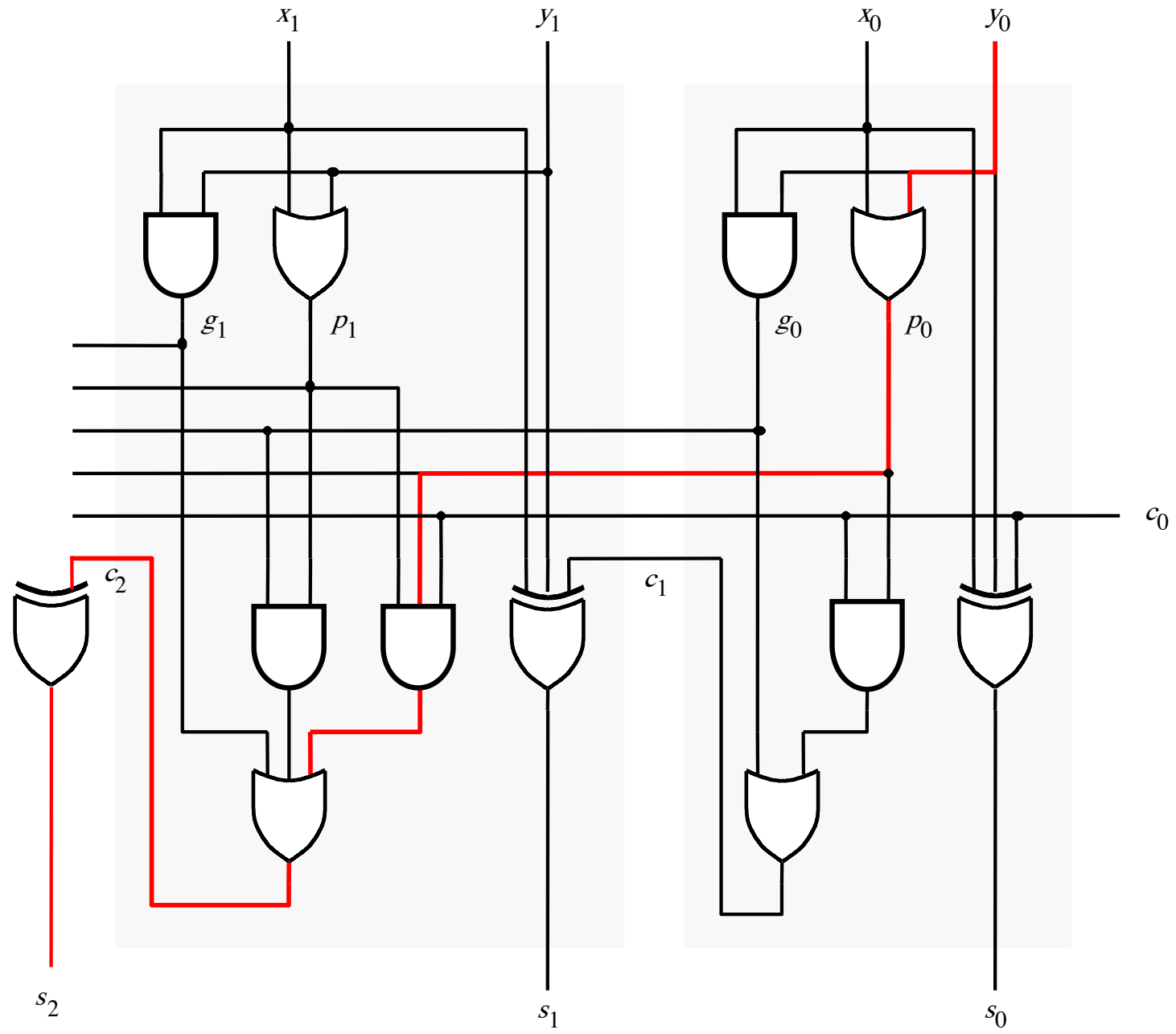
The first two stages of a carry-lookahead adder



It takes 4 gate delays to generate s_1



It takes 4 gate delays to generate s_2



N-bit Carry-Lookahead Adder

- **It takes 3 gate delays to generate all carry signals**
- **It takes 1 more gate delay to generate all sum bits**
- **Thus, the total delay through an n-bit carry-lookahead adder is only 4 gate delays!**

Expanding the Carry Expression

$$c_{i+1} = g_i + p_i c_i$$

$$c_1 = g_0 + p_0 c_0$$

$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

$$c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$

$$\cdots$$

$$c_8 = g_7 + p_7 g_6 + p_7 p_6 g_5 + p_7 p_6 p_5 g_4 + p_7 p_6 p_5 p_4 g_3 + p_7 p_6 p_5 p_4 p_3 g_2 + p_7 p_6 p_5 p_4 p_3 p_2 g_1 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 g_0 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0 c_0$$

Expanding the Carry Expression

$$c_{i+1} = g_i + p_i c_i$$

$$c_1 = g_0 + p_0 c_0$$

$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

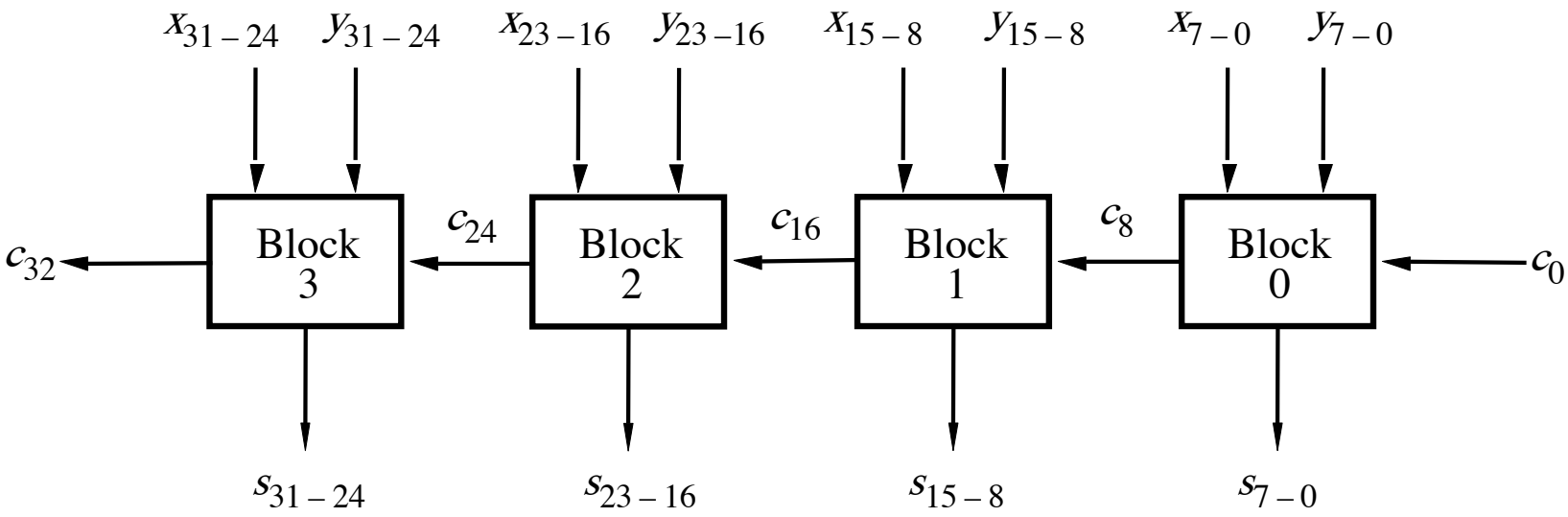
$$c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$

$$\cdots$$

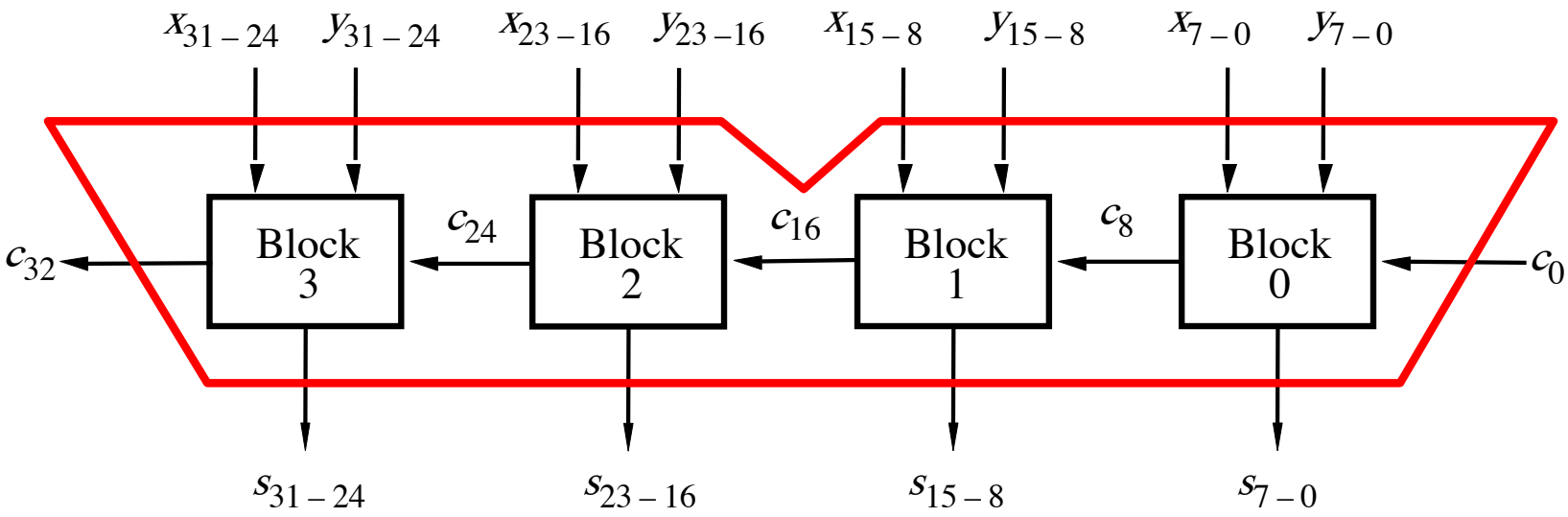
$$c_8 = g_7 + p_7 g_6 + p_7 p_6 g_5 + p_7 p_6 p_5 g_4 + p_7 p_6 p_5 p_4 g_3 + p_7 p_6 p_5 p_4 p_3 g_2 + p_7 p_6 p_5 p_4 p_3 p_2 g_1 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 g_0 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0 c_0$$

Even this takes only 3 gate delays

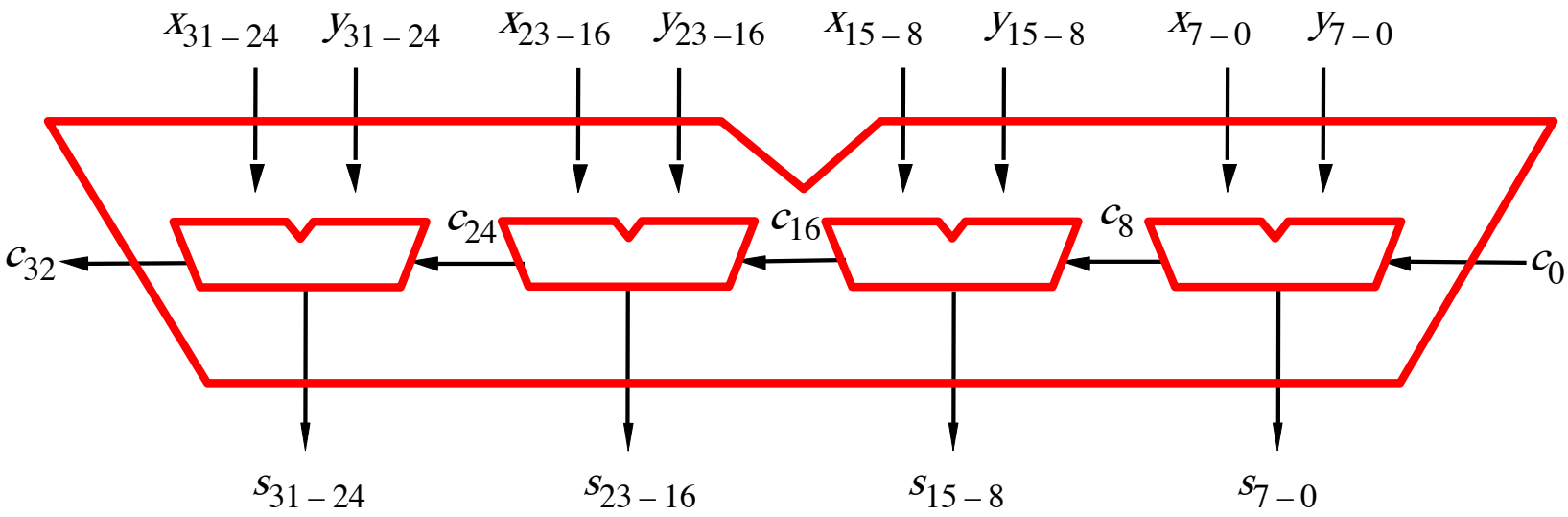
A hierarchical carry-lookahead adder with ripple-carry between blocks



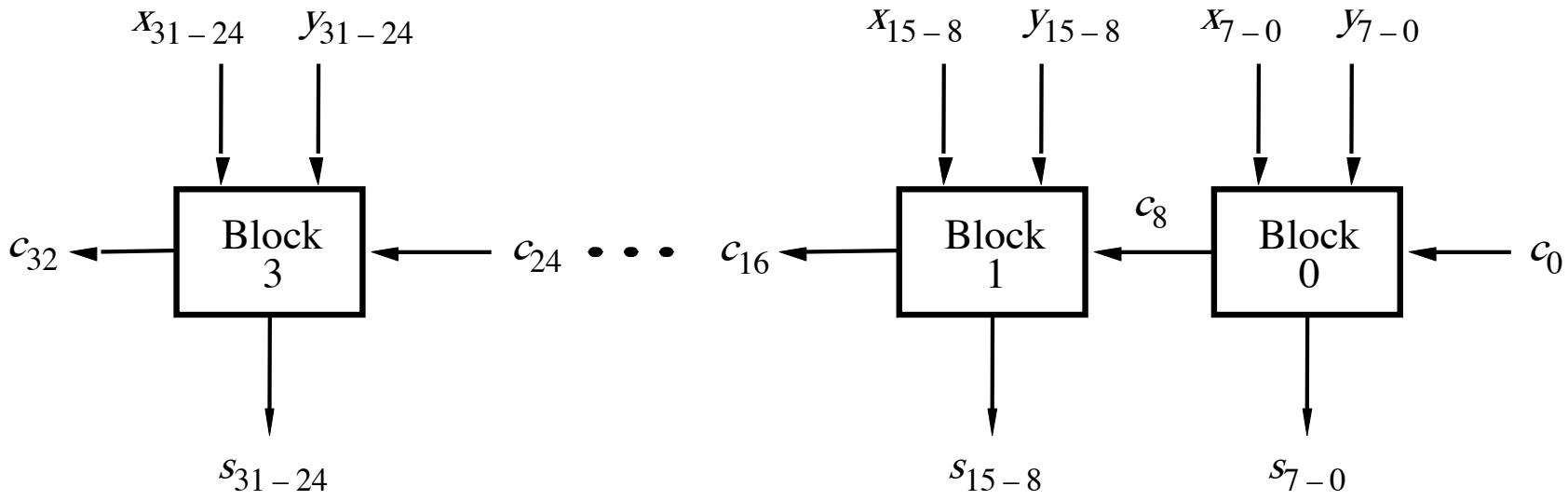
A hierarchical carry-lookahead adder with ripple-carry between blocks



A hierarchical carry-lookahead adder with ripple-carry between blocks

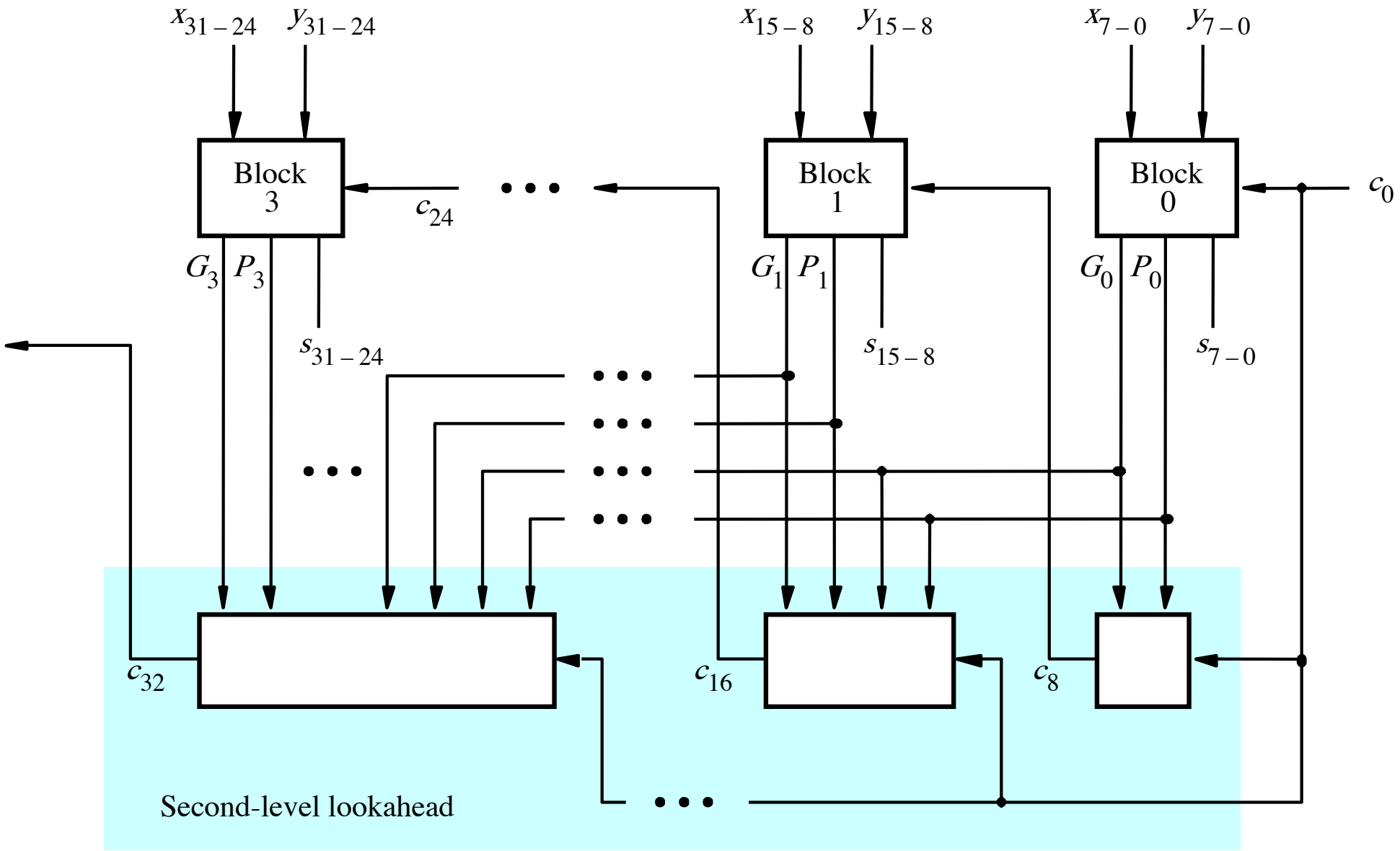


A hierarchical carry-lookahead adder with ripple-carry between blocks



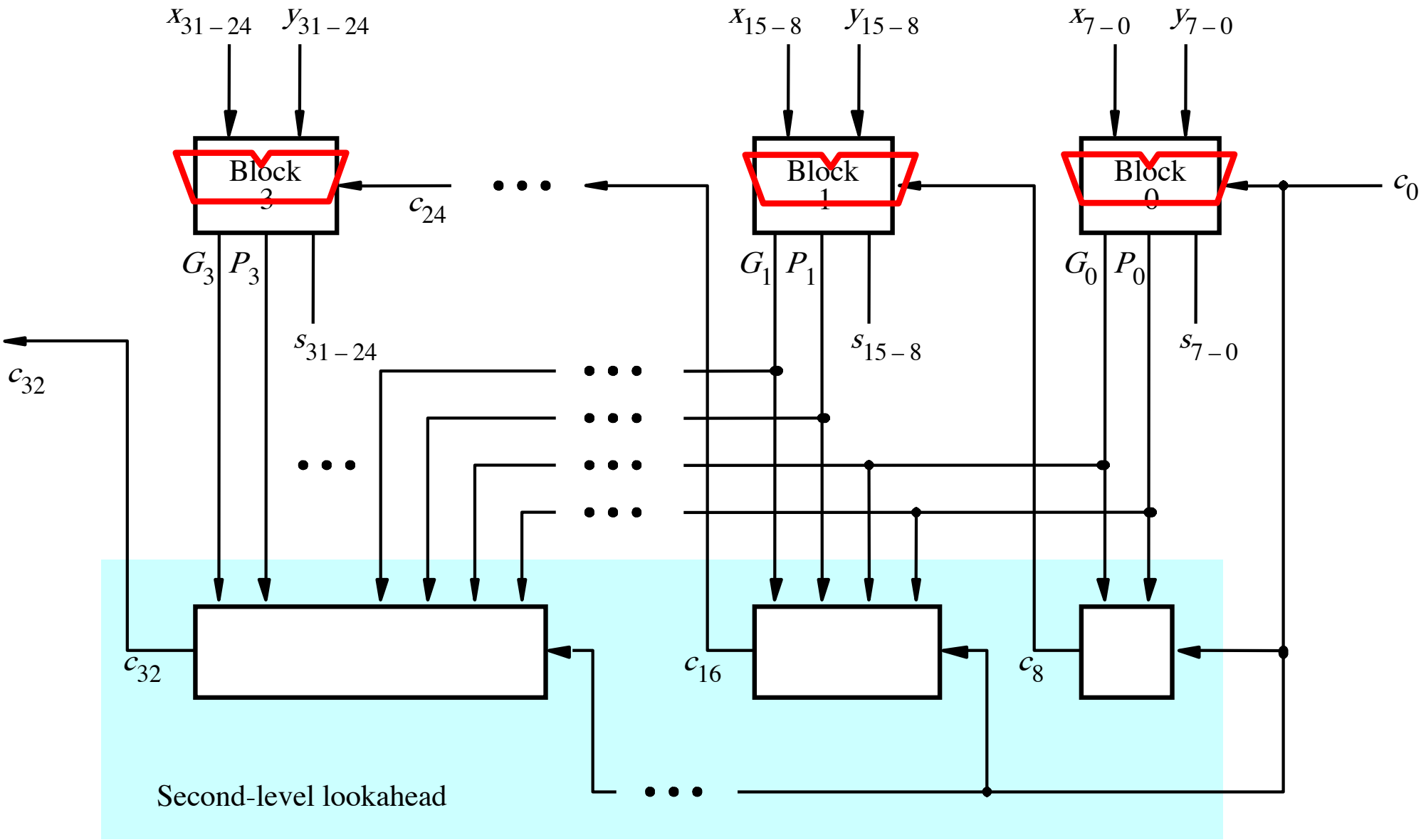
[Figure 3.16 from the textbook]

A hierarchical carry-lookahead adder

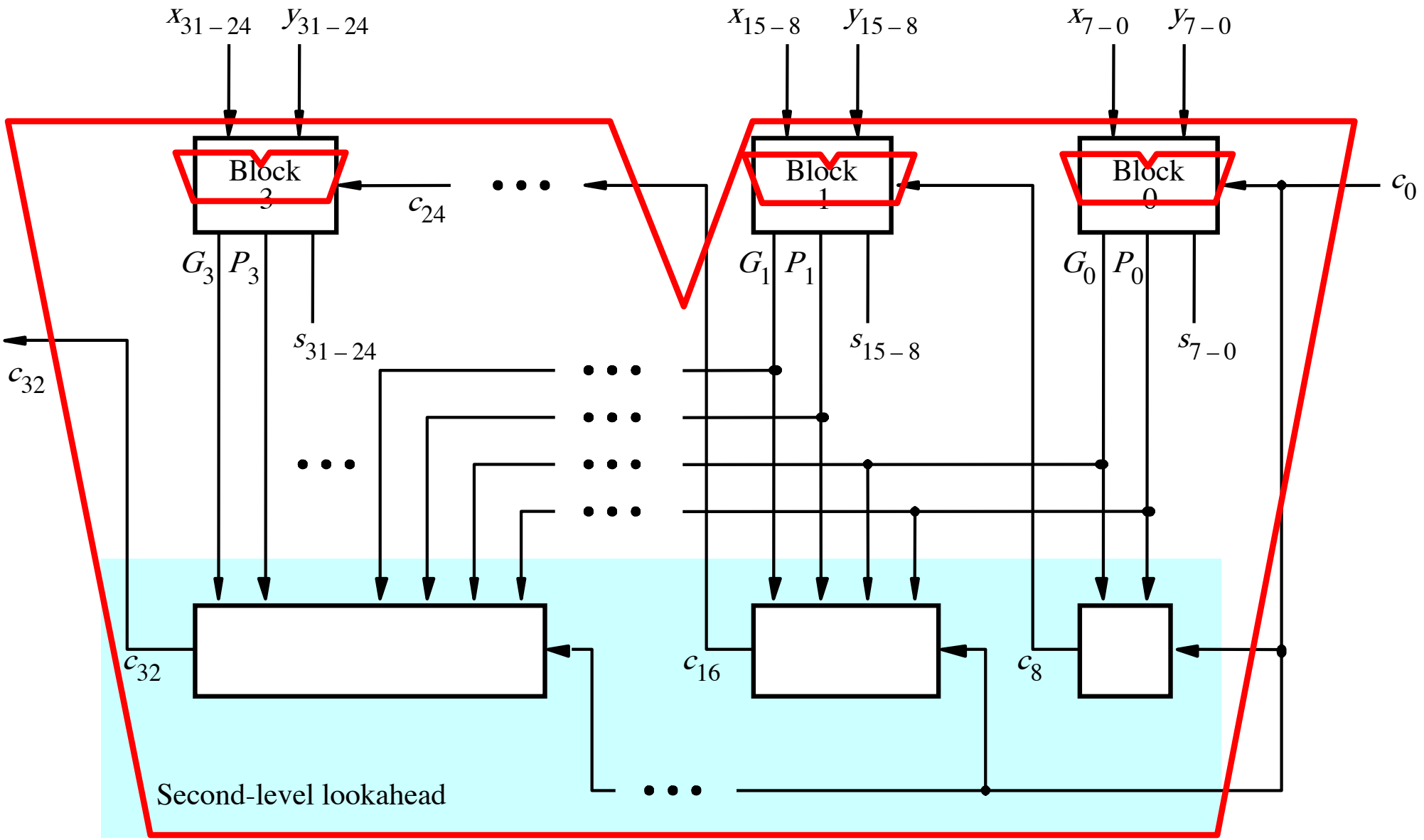


[Figure 3.17 from the textbook]

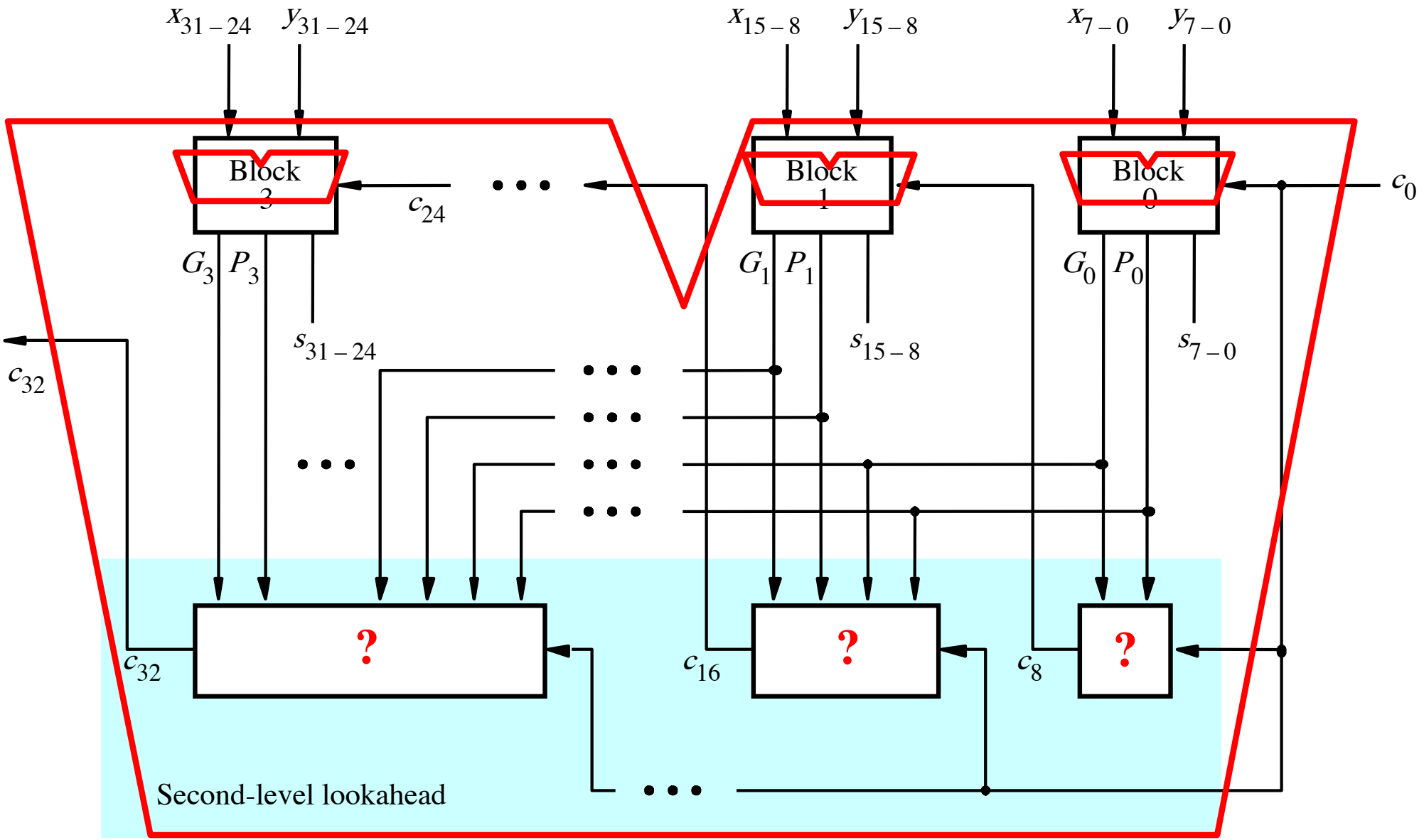
A hierarchical carry-lookahead adder



A hierarchical carry-lookahead adder



A hierarchical carry-lookahead adder



The Hierarchical Carry Expression

$$
\begin{aligned}
c_8 = \; & g_7 + p_7 g_6 + p_7 p_6 g_5 + p_7 p_6 p_5 g_4 \\
& + p_7 p_6 p_5 p_4 g_3 + p_7 p_6 p_5 p_4 p_3 g_2 \\
& + p_7 p_6 p_5 p_4 p_3 p_2 g_1 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 g_0 \\
& + p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0 c_0
\end{aligned}
$$

The Hierarchical Carry Expression

$$c_8 = g_7 + p_7 g_6 + p_7 p_6 g_5 + p_7 p_6 p_5 g_4 + p_7 p_6 p_5 p_4 g_3 + p_7 p_6 p_5 p_4 p_3 g_2 + p_7 p_6 p_5 p_4 p_3 p_2 g_1 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 g_0 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0 c_0$$

The Hierarchical Carry Expression

$$c_8 = g_7 + p_7 g_6 + p_7 p_6 g_5 + p_7 p_6 p_5 g_4 + p_7 p_6 p_5 p_4 g_3 + p_7 p_6 p_5 p_4 p_3 g_2 + p_7 p_6 p_5 p_4 p_3 p_2 g_1 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 g_0 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0 c_0$$

G_0

P_0

The Hierarchical Carry Expression

$$c_8 = g_7 + p_7 g_6 + p_7 p_6 g_5 + p_7 p_6 p_5 g_4$$
$$+ p_7 p_6 p_5 p_4 g_3 + p_7 p_6 p_5 p_4 p_3 g_2$$
$$+ p_7 p_6 p_5 p_4 p_3 p_2 g_1 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 g_0$$
$$+ p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0 c_0$$

G_0

P_0

$$c_8 = G_0 + P_0 c_0$$

The Hierarchical Carry Expression

$$c_8 = g_7 + p_7g_6 + p_7p_6g_5 + p_7p_6p_5g_4 + p_7p_6p_5p_4g_3 + p_7p_6p_5p_4p_3g_2 + p_7p_6p_5p_4p_3p_2g_1 + p_7p_6p_5p_4p_3p_2p_1g_0 + p_7p_6p_5p_4p_3p_2p_1p_0c_0$$

$$c_{16} = g_{15} + p_{15}g_{14} + p_{15}p_{14}g_{13} + p_{15}p_{14}p_{13}g_{12} + p_{15}p_{14}p_{13}p_{12}g_{11} + p_{15}p_{14}p_{13}p_{12}p_{11}g_{10} + p_{15}p_{14}p_{13}p_{12}p_{11}p_{10}g_9 + p_{15}p_{14}p_{13}p_{12}p_{11}p_{10}p_9g_8 + p_{15}p_{14}p_{13}p_{12}p_{11}p_{10}p_9p_8c_8$$

The Hierarchical Carry Expression

$$c_8 = g_7 + p_7g_6 + p_7p_6g_5 + p_7p_6p_5g_4$$
$$+ p_7p_6p_5p_4g_3 + p_7p_6p_5p_4p_3g_2$$
$$+ p_7p_6p_5p_4p_3p_2g_1 + p_7p_6p_5p_4p_3p_2p_1g_0$$
$$+ p_7p_6p_5p_4p_3p_2p_1p_0c_0$$

The same expression, just add 8 to all subscripts

$$c_{16} = g_{15} + p_{15}g_{14} + p_{15}p_{14}g_{13} + p_{15}p_{14}p_{13}g_{12}$$
$$+ p_{15}p_{14}p_{13}p_{12}g_{11} + p_{15}p_{14}p_{13}p_{12}p_{11}g_{10}$$
$$+ p_{15}p_{14}p_{13}p_{12}p_{11}p_{10}g_9 + p_{15}p_{14}p_{13}p_{12}p_{11}p_{10}p_9g_8$$
$$+ p_{15}p_{14}p_{13}p_{12}p_{11}p_{10}p_9p_8c_8$$

The Hierarchical Carry Expression

3-gate delays

$$c_8 = g_7 + p_7g_6 + p_7p_6g_5 + p_7p_6p_5g_4$$
$$+ p_7p_6p_5p_4g_3 + p_7p_6p_5p_4p_3g_2$$
$$+ p_7p_6p_5p_4p_3p_2g_1 + p_7p_6p_5p_4p_3p_2p_1g_0$$
$$+ p_7p_6p_5p_4p_3p_2p_1p_0c_0$$

G_0

P_0

2-gate delays

$$c_{16} = g_{15} + p_{15}g_{14} + p_{15}p_{14}g_{13} + p_{15}p_{14}p_{13}g_{12}$$
$$+ p_{15}p_{14}p_{13}p_{12}g_{11} + p_{15}p_{14}p_{13}p_{12}p_{11}g_{10}$$
$$+ p_{15}p_{14}p_{13}p_{12}p_{11}p_{10}g_9 + p_{15}p_{14}p_{13}p_{12}p_{11}p_{10}p_9g_8$$
$$+ p_{15}p_{14}p_{13}p_{12}p_{11}p_{10}p_9p_8c_8$$

The Hierarchical Carry Expression

$$c_8 = g_7 + p_7g_6 + p_7p_6g_5 + p_7p_6p_5g_4 + p_7p_6p_5p_4g_3 + p_7p_6p_5p_4p_3g_2 + p_7p_6p_5p_4p_3p_2g_1 + p_7p_6p_5p_4p_3p_2p_1g_0 + p_7p_6p_5p_4p_3p_2p_1p_0c_0$$

3-gate delays

$$c_{16} = g_{15} + p_{15}g_{14} + p_{15}p_{14}g_{13} + p_{15}p_{14}p_{13}g_{12} + p_{15}p_{14}p_{13}p_{12}g_{11} + p_{15}p_{14}p_{13}p_{12}p_{11}g_{10} + p_{15}p_{14}p_{13}p_{12}p_{11}p_{10}g_9 + p_{15}p_{14}p_{13}p_{12}p_{11}p_{10}p_9g_8 + p_{15}p_{14}p_{13}p_{12}p_{11}p_{10}p_9p_8c_8$$

G_1

P_1

2-gate delays

The Hierarchical Carry Expression

$$c_8 = G_0 + P_0 c_0$$

$$\begin{aligned} c_{16} &= G_1 + P_1 c_8 \\ &= G_1 + P_1 G_0 + P_1 P_0 c_0 \end{aligned}$$

$$c_{24} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$$

$$c_{32} = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0$$

The Hierarchical Carry Expression

$$c_8 = G_0 + P_0 c_0$$

4-gate delays

$$c_{16} = G_1 + P_1 c_8$$
$$= G_1 + P_1 G_0 + P_1 P_0 c_0$$

5-gate delays

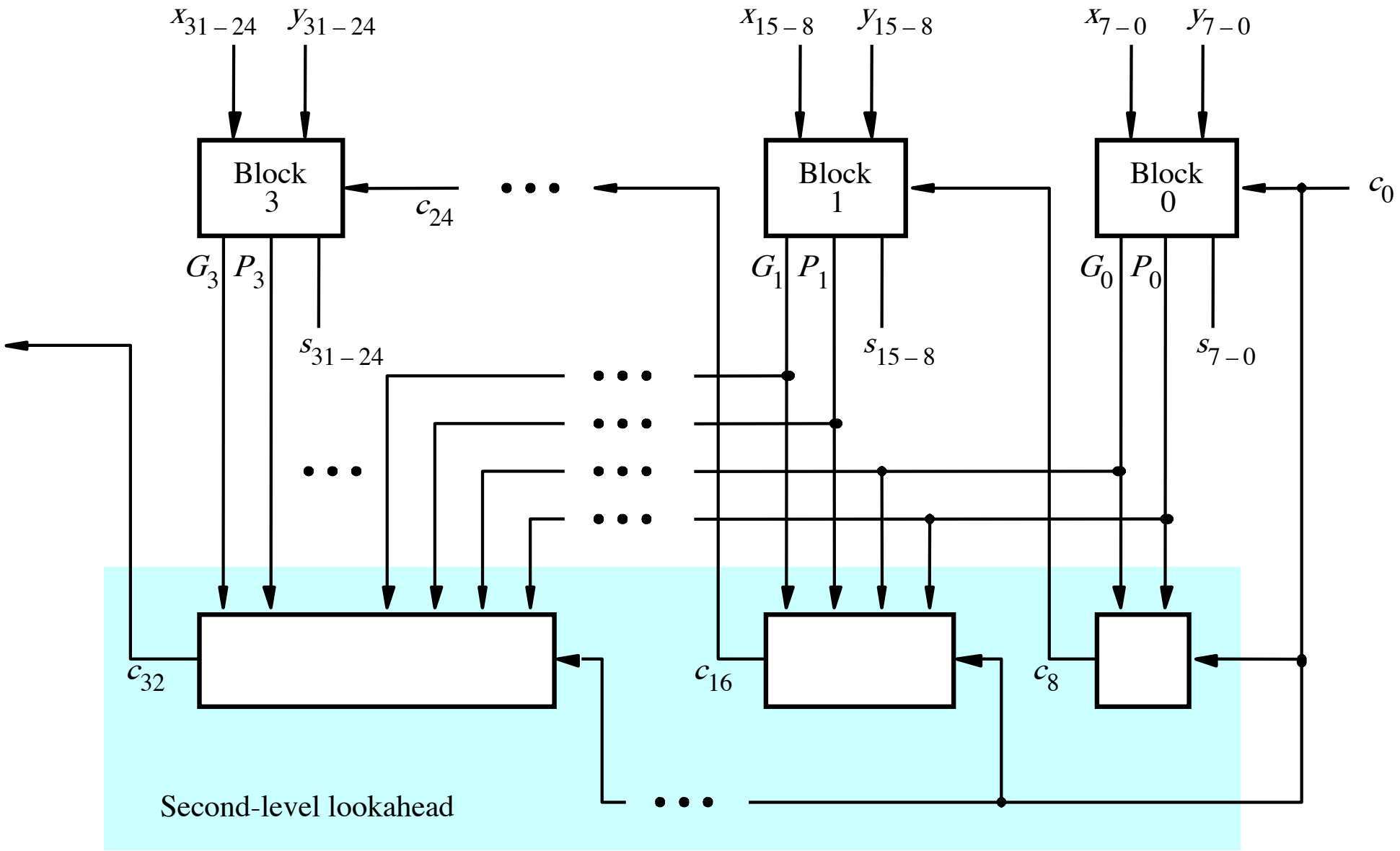
$$c_{24} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$$

5-gate delays

$$c_{32} = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0$$

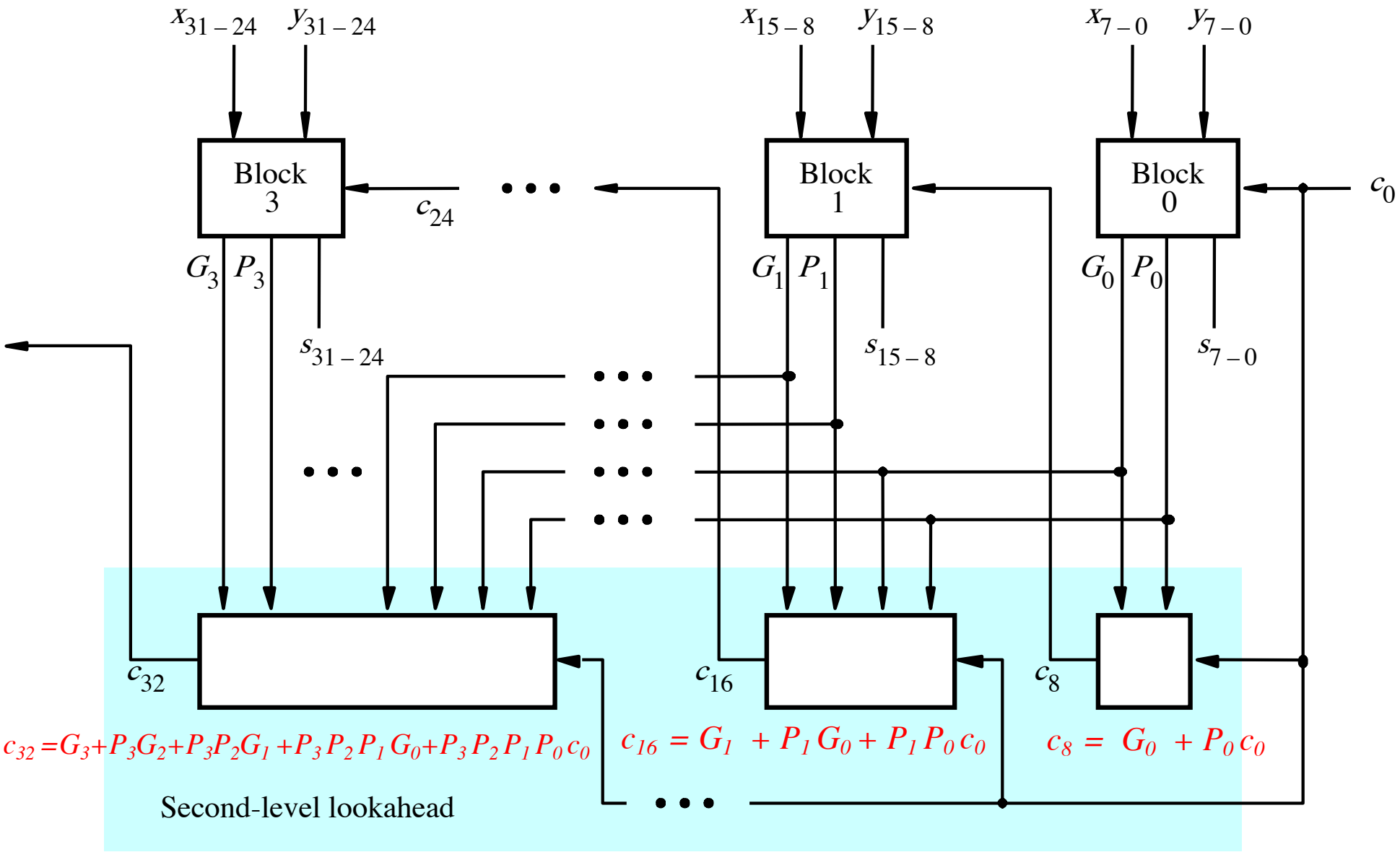
5-gate delays

A hierarchical carry-lookahead adder



[Figure 3.17 from the textbook]

A hierarchical carry-lookahead adder

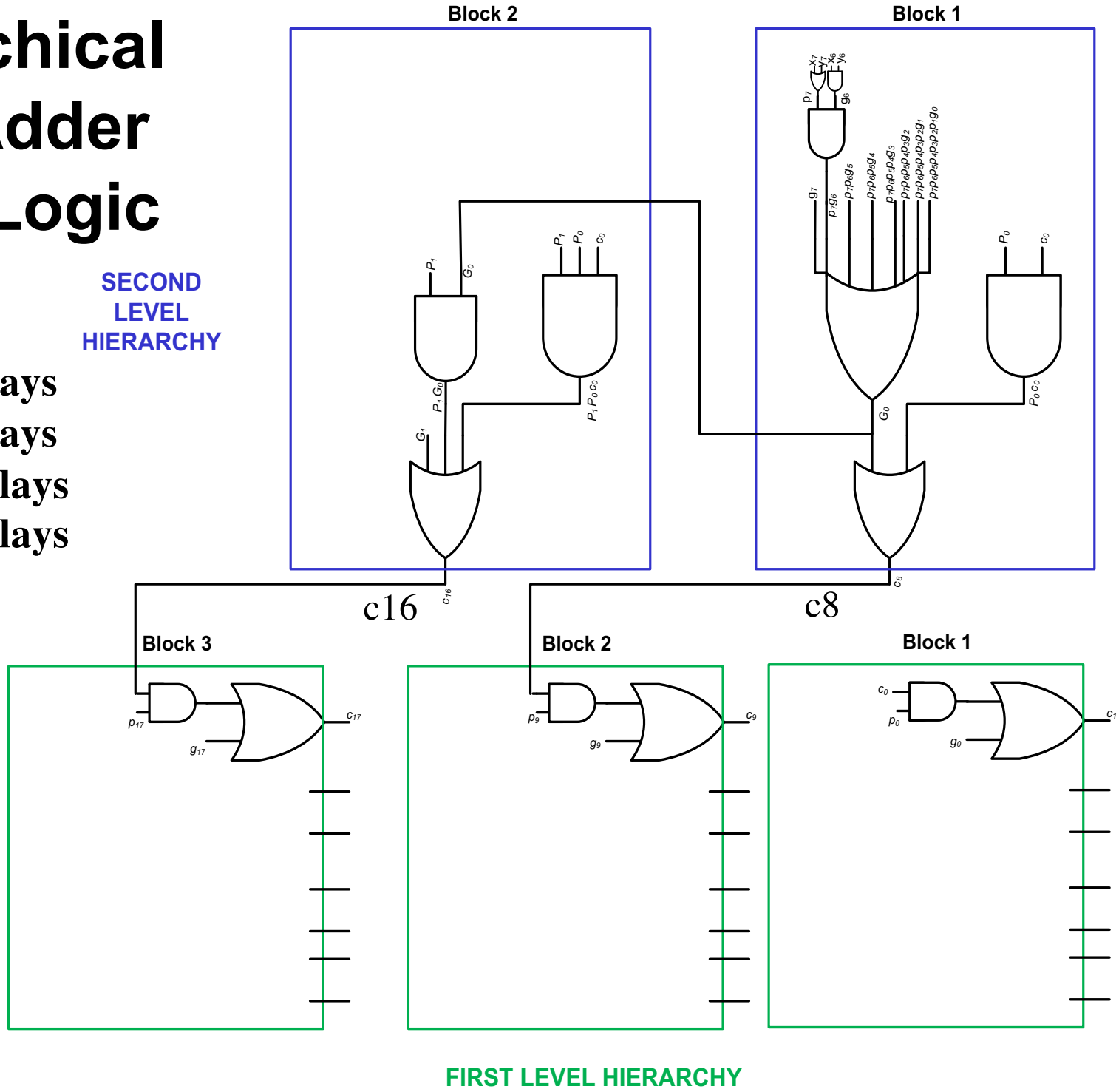


[Figure 3.17 from the textbook]

Hierarchical CLA Adder Carry Logic

SECOND LEVEL HIERARCHY

C8 – 4 gate delays
C16 – 5 gate delays
C24 – 5 Gate delays
C32 – 5 Gate delays



FIRST LEVEL HIERARCHY

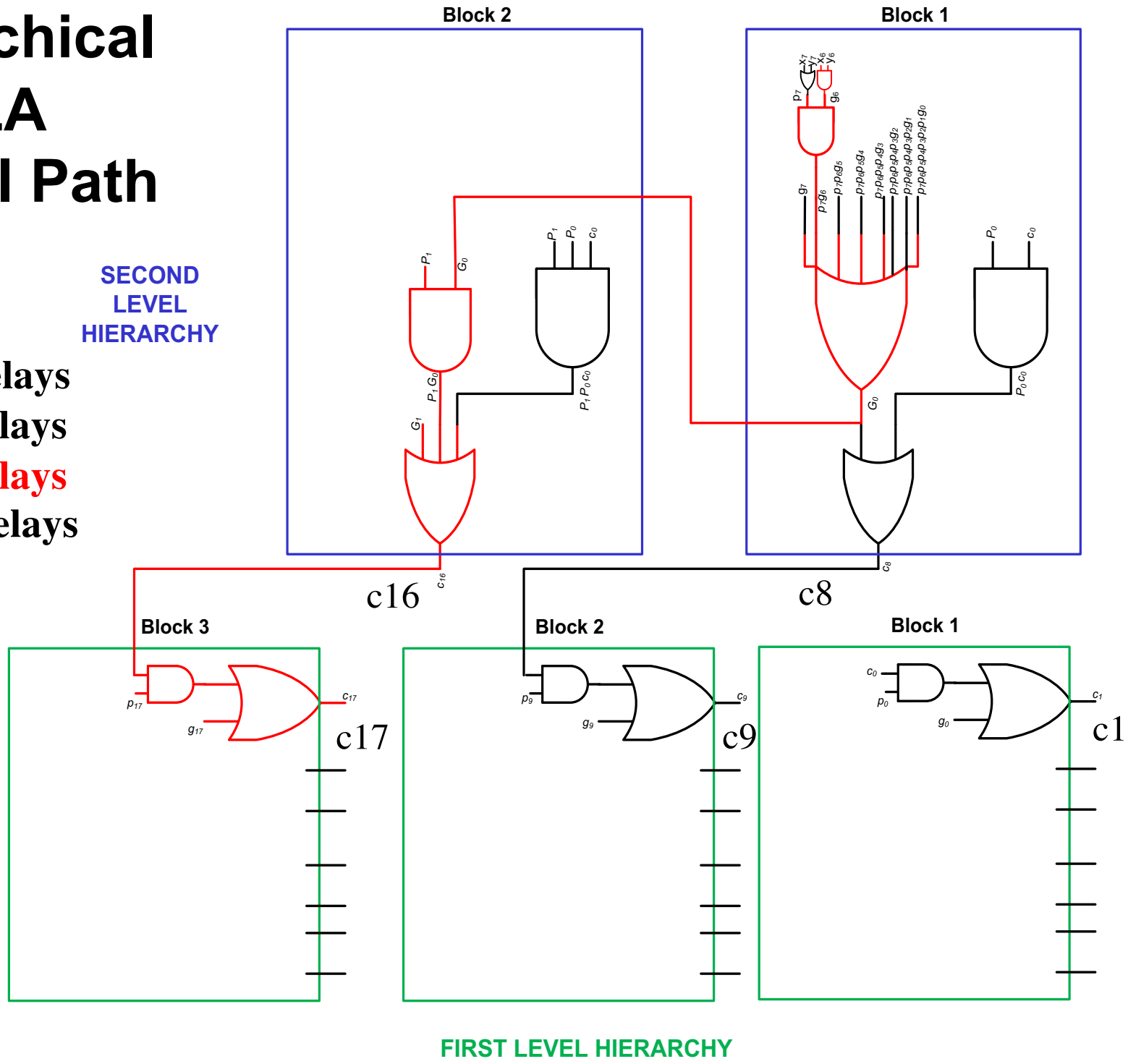
Hierarchical CLA Critical Path

C1 - 2 gate delays

C9 – 6 gate delays

C17 – 7 gate delays

C25 – 7 Gate delays



Total Gate Delay Through a Hierarchical Carry-Lookahead Adder

- **The total delay is 8 gates:**
  - **3 to generate all Gj and Pj signals**
  - **+2 to generate c8, c16, c24, and c32**
  - **+2 to generate internal carries in the blocks**
  - **+1 to generate the sum bits (one extra XOR)**

Decimal Multiplication by 10

What happens when we multiply a number by 10?

4 x 10 = ?

542 x 10 = ?

1245 x 10 = ?

Decimal Multiplication by 10

What happens when we multiply a number by 10?

4 x 10 = 40

542 x 10 = 5420

1245 x 10 = 12450

Decimal Multiplication by 10

What happens when we multiply a number by 10?

4 x 10 = 40

542 x 10 = 5420

1245 x 10 = 12450

You simply add a zero as the rightmost number

Decimal Division by 10

What happens when we divide a number by 10?

14 / 10 = ?

540 / 10 = ?

1240 / 10 = ?

Decimal Division by 10

What happens when we divide a number by 10?

14 / 10 = 1 //integer division

540 / 10 = 54

1240 / 10 = 124

You simply delete the rightmost number

Binary Multiplication by 2

What happens when we multiply a number by 2?

011 times 2 = ?

101 times 2 = ?

110011 times 2 = ?

Binary Multiplication by 2

What happens when we multiply a number by 2?

011 times 2 = 0110

101 times 2 = 1010

110011 times 2 = 1100110

You simply add a zero as the rightmost number

Binary Multiplication by 4

What happens when we multiply a number by 4?

011 times 4 = ?

101 times 4 = ?

110011 times 4 = ?

Binary Multiplication by 4

What happens when we multiply a number by 4?

011 times 4 = 01100

101 times 4 = 10100

110011 times 4 = 11001100

add two zeros in the last two bits and shift everything else to the left

Binary Multiplication by 2^N

What happens when we multiply a number by 2^N?

011 times 2^N = 011 00…0 // add N zeros

101 times 4 = 101 00…0 // add N zeros

110011 times 4 = 110011 00…0 // add N zeros

Binary Division by 2

What happens when we divide a number by 2?

0110 divided by 2 = ?

1010 divides by 2 = ?

110011 divides by 2 = ?

Binary Division by 2

What happens when we divide a number by 2?

0110 divided by 2 = 011

1010 divides by 2 = 101

110011 divides by 2 = 11001

You simply delete the rightmost number

Decimal Multiplication By Hand

$$
\begin{array}{r}
5127 \\
\times\ 4265 \\
\hline
25635 \\
307620 \\
1025400 \\
20508000 \\
\hline
21866655
\end{array}
$$

[http://www.ducksters.com/kidsmath/long_multiplication.php]

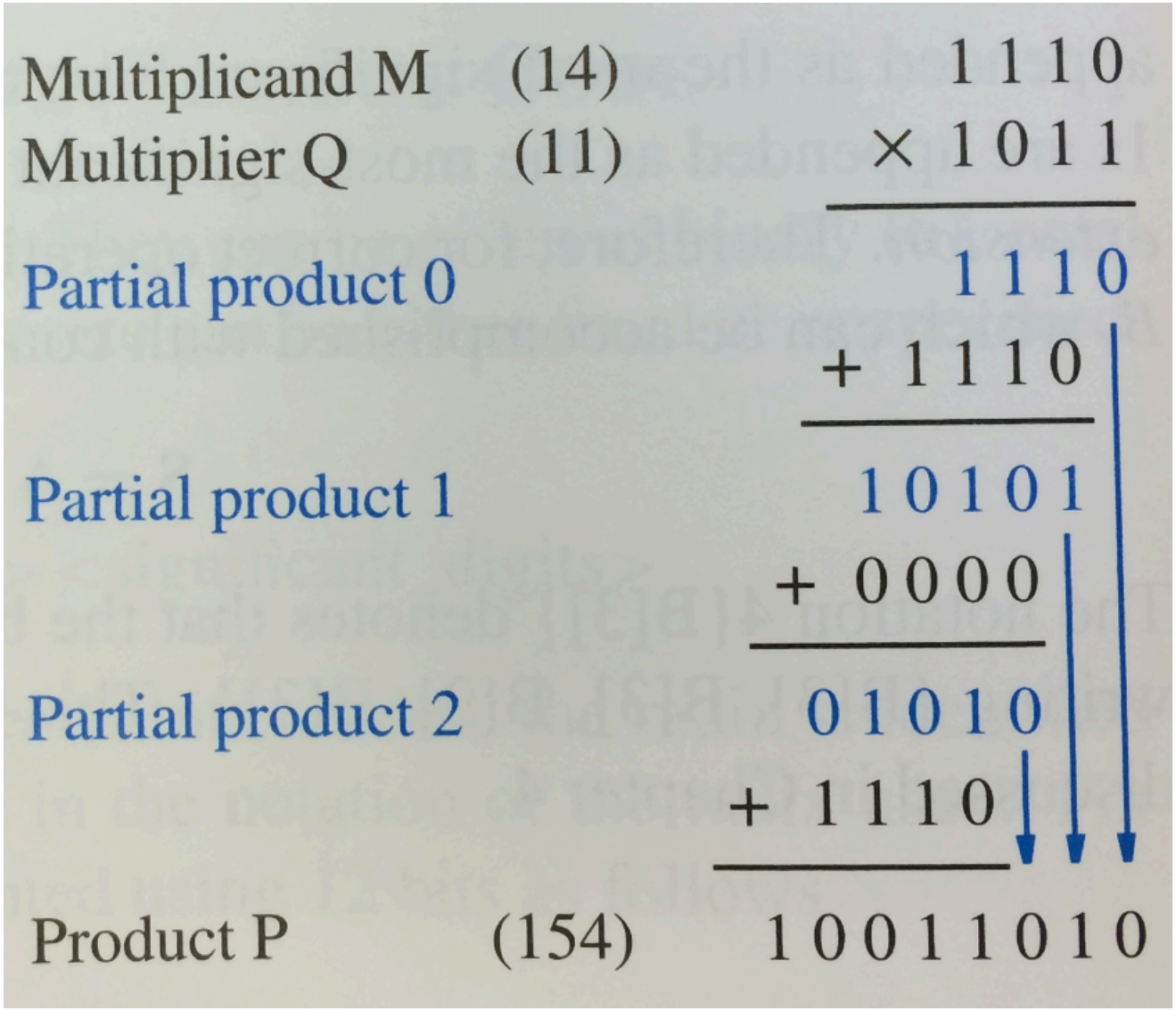
Multiplication of two unsigned binary numbers

Binary Multiplication By Hand

Multiplicand M	(14)	1 1 1 0
Multiplier Q	(11)	x 1 0 1 1
		1 1 1 0
		1 1 1 0
		0 0 0 0
		1 1 1 0
Product P	(154)	1 0 0 1 1 0 1 0

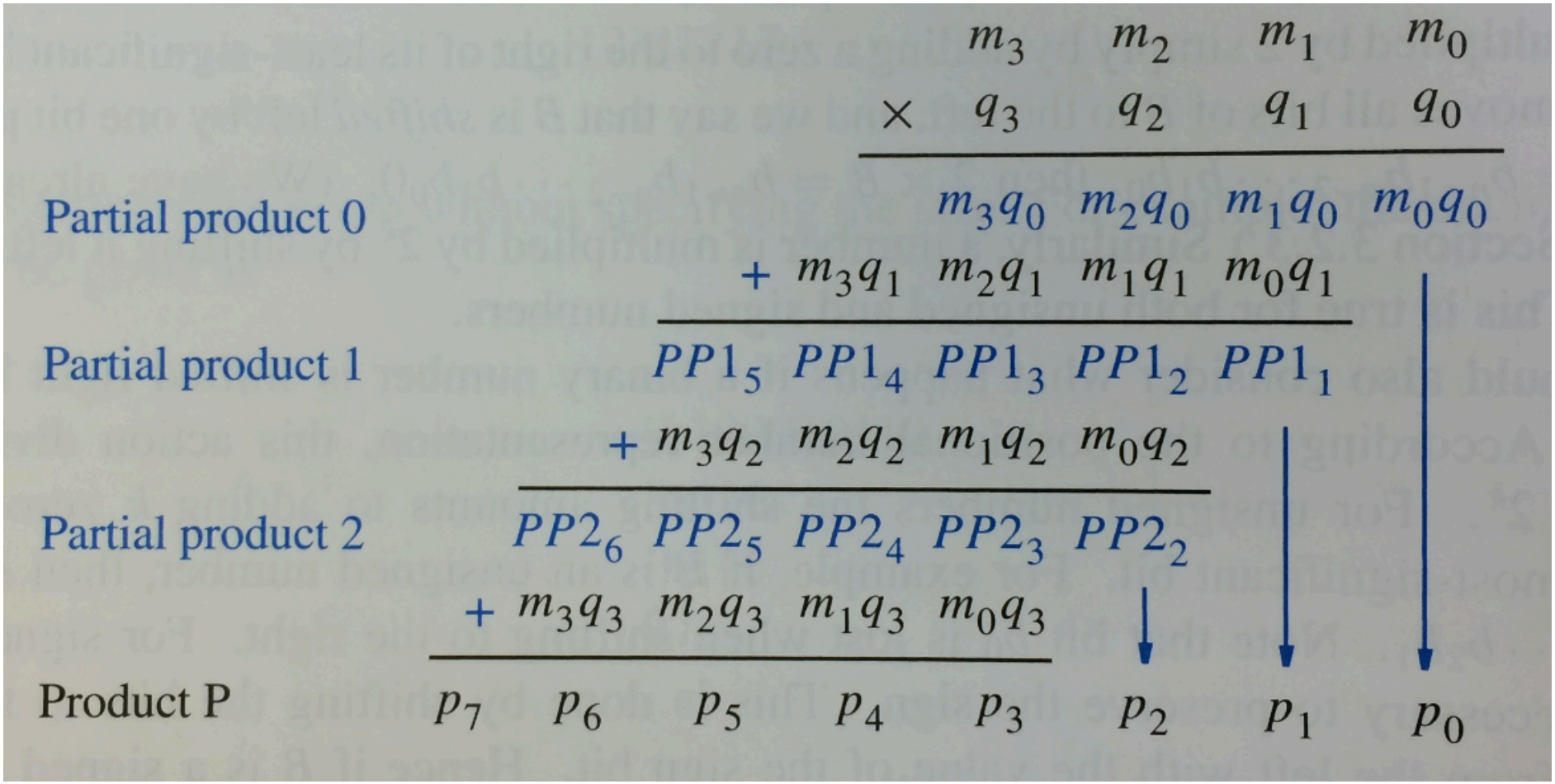
[Figure 3.34a from the textbook]

Binary Multiplication By Hand



[Figure 3.34b from the textbook]

Binary Multiplication By Hand



[Figure 3.34c from the textbook]

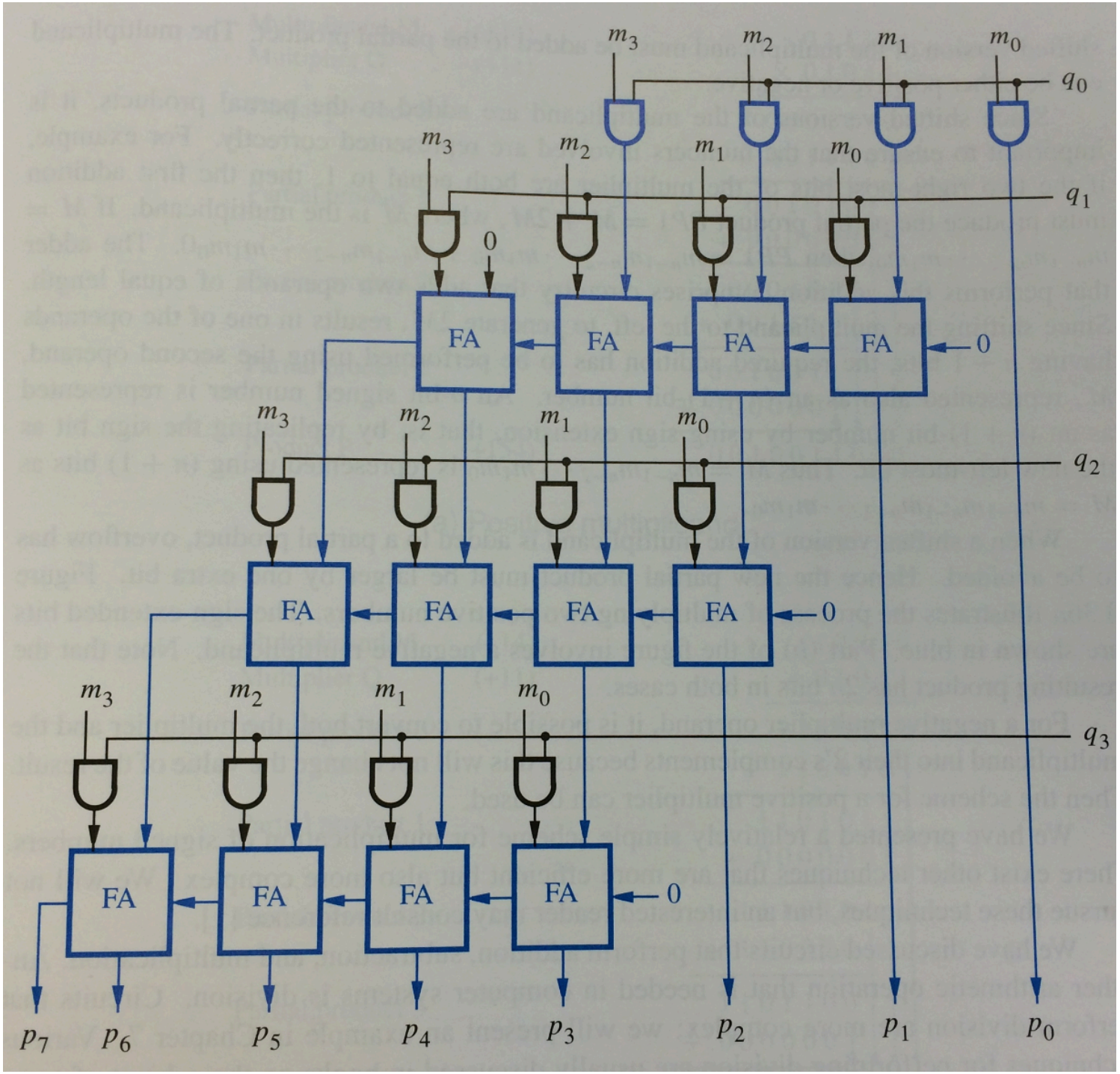


Figure 3.35. A 4x4 multiplier circuit.

Multiplication of two signed binary numbers

Sign extension for positive numbers

- **If we want to represent the same positive number with more bits, we simply pad it on the left with zeros.**

- **For example:**

0110	**(+6 with 4-bits)**
00110	**(+6 with 5-bits)**
000110	**(+6 with 6-bits)**

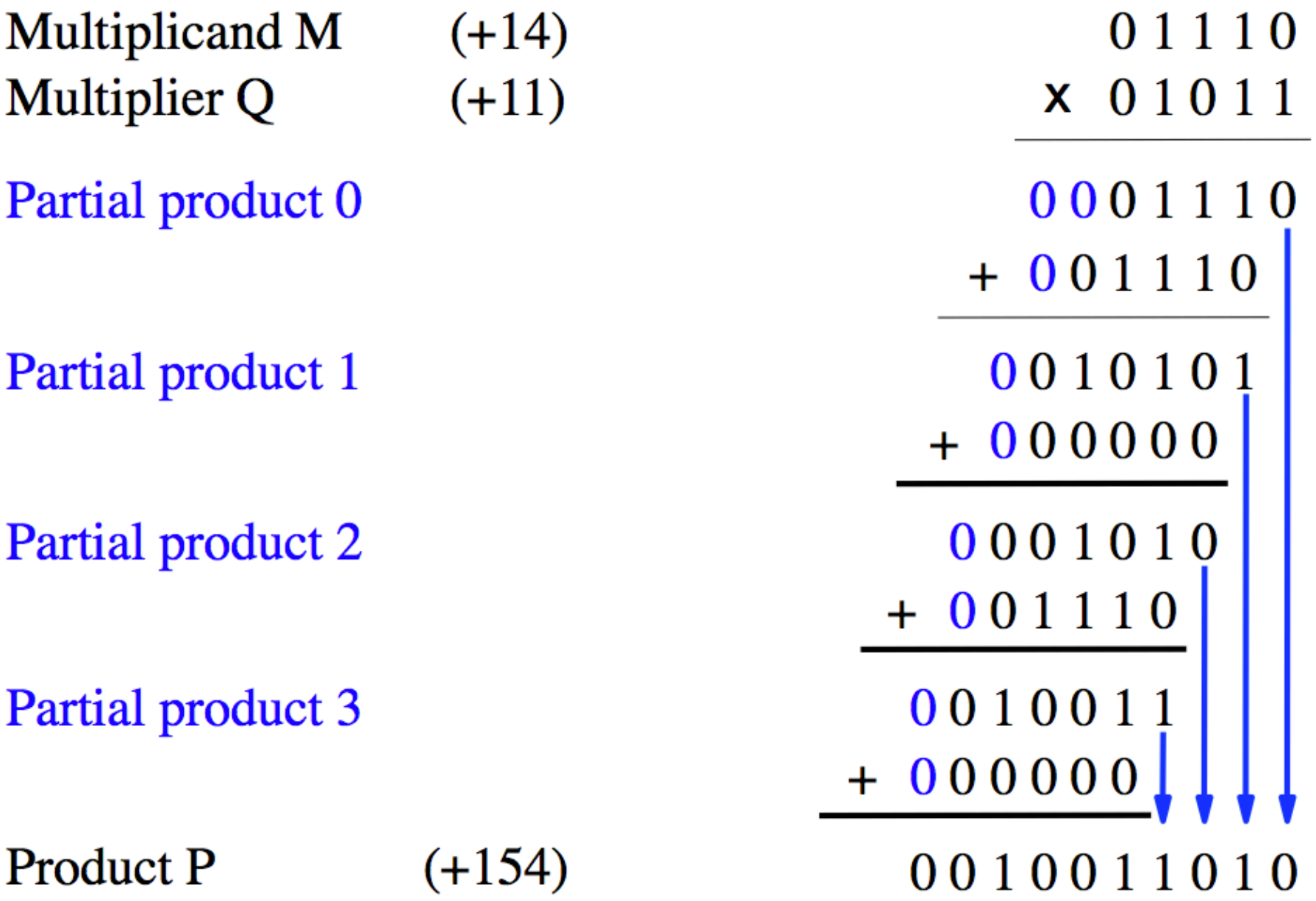
Sign extension for negative numbers

- **If we want to represent the same negative number with more bits, we simply pad it on the left with ones.**

- **For example:**

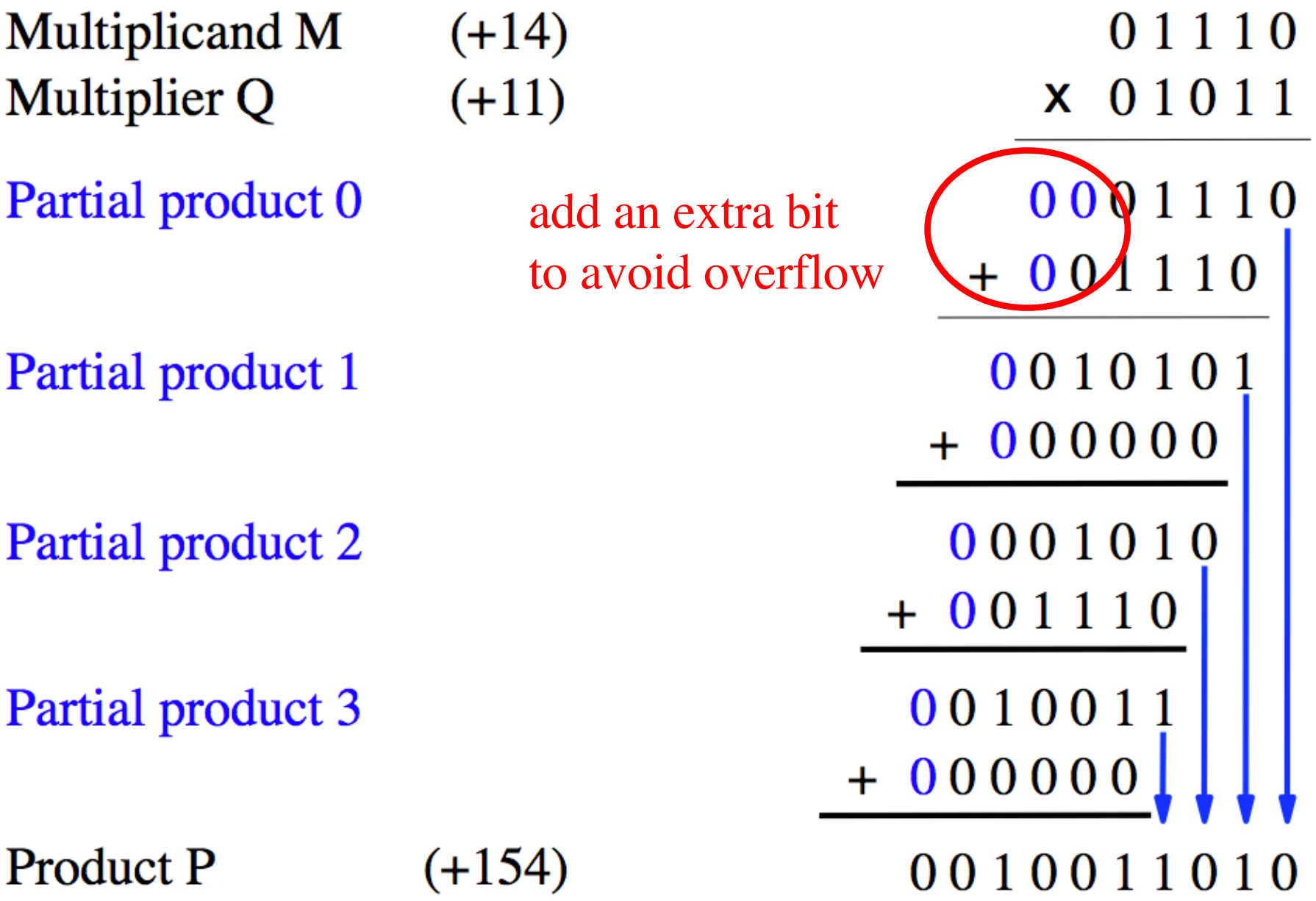
1011	**(-5 with 4-bits)**
11011	**(-5 with 5-bits)**
111011	**(-5 with 6-bits)**

Positive Multiplicand Example



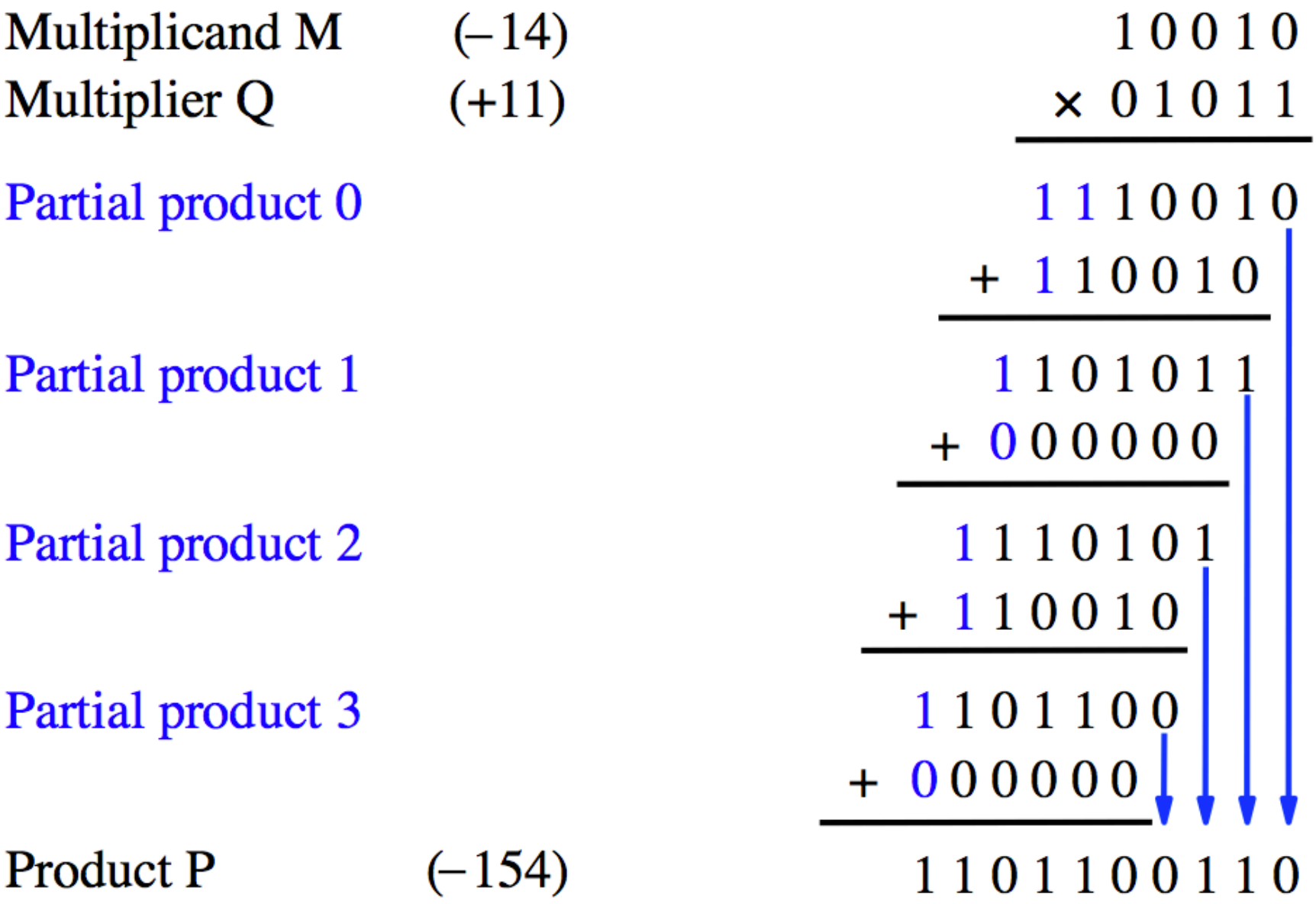
[Figure 3.36a in the textbook]

Positive Multiplicand Example



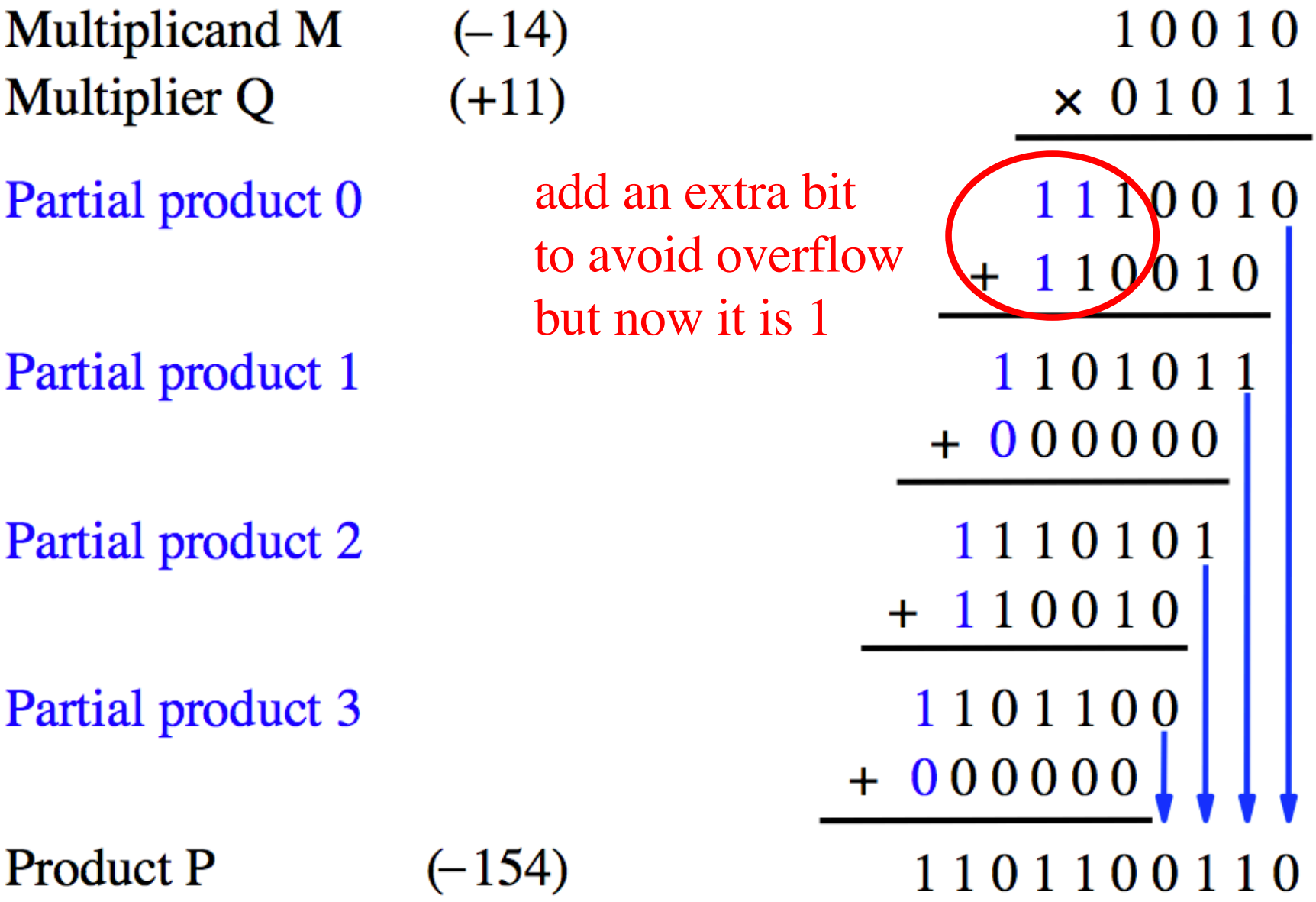
[Figure 3.36a in the textbook]

Negative Multiplicand Example



[Figure 3.36b in the textbook]

Negative Multiplicand Example



[Figure 3.36b in the textbook]

What if the Multiplier is Negative?

- **Negate both numbers.**
- **This will make the multiplier positive.**
- **Then proceed as normal.**
- **This will not affect the result.**
- **Example: 5*(- 4) = (- 5)*(4)= - 20**

Arithmetic Comparison Circuits

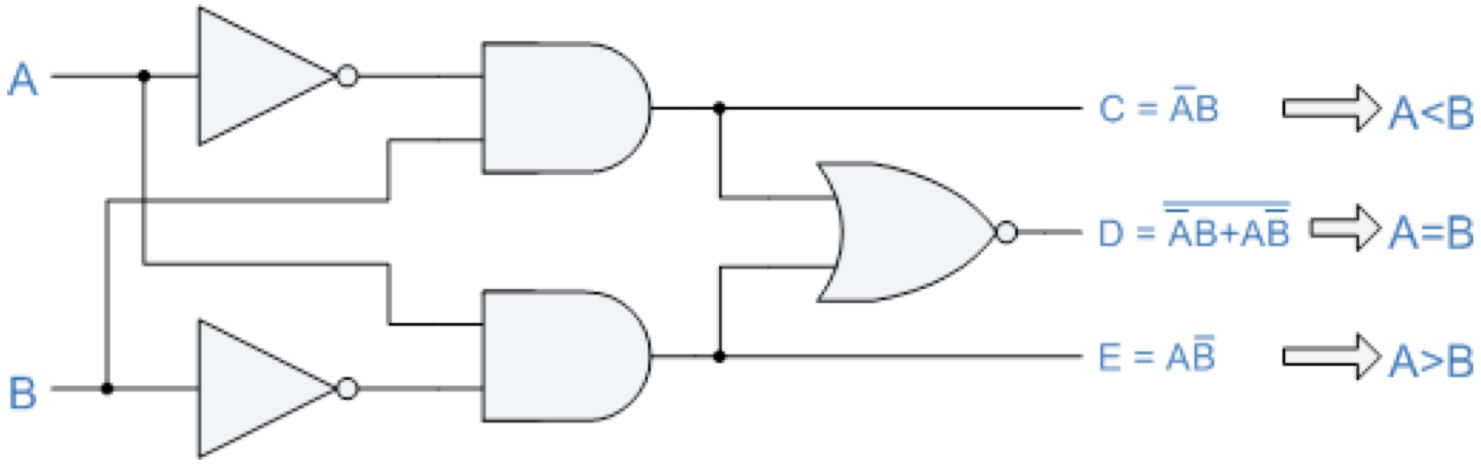
Truth table for a one-bit digital comparator

Inputs		Outputs		
A	B	$A > B$	$A = B$	$A < B$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

[http://en.wikipedia.org/wiki/Digital_comparator]

A one-bit digital comparator circuit

Inputs		Outputs		
A	B	$A > B$	$A = B$	$A < B$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0



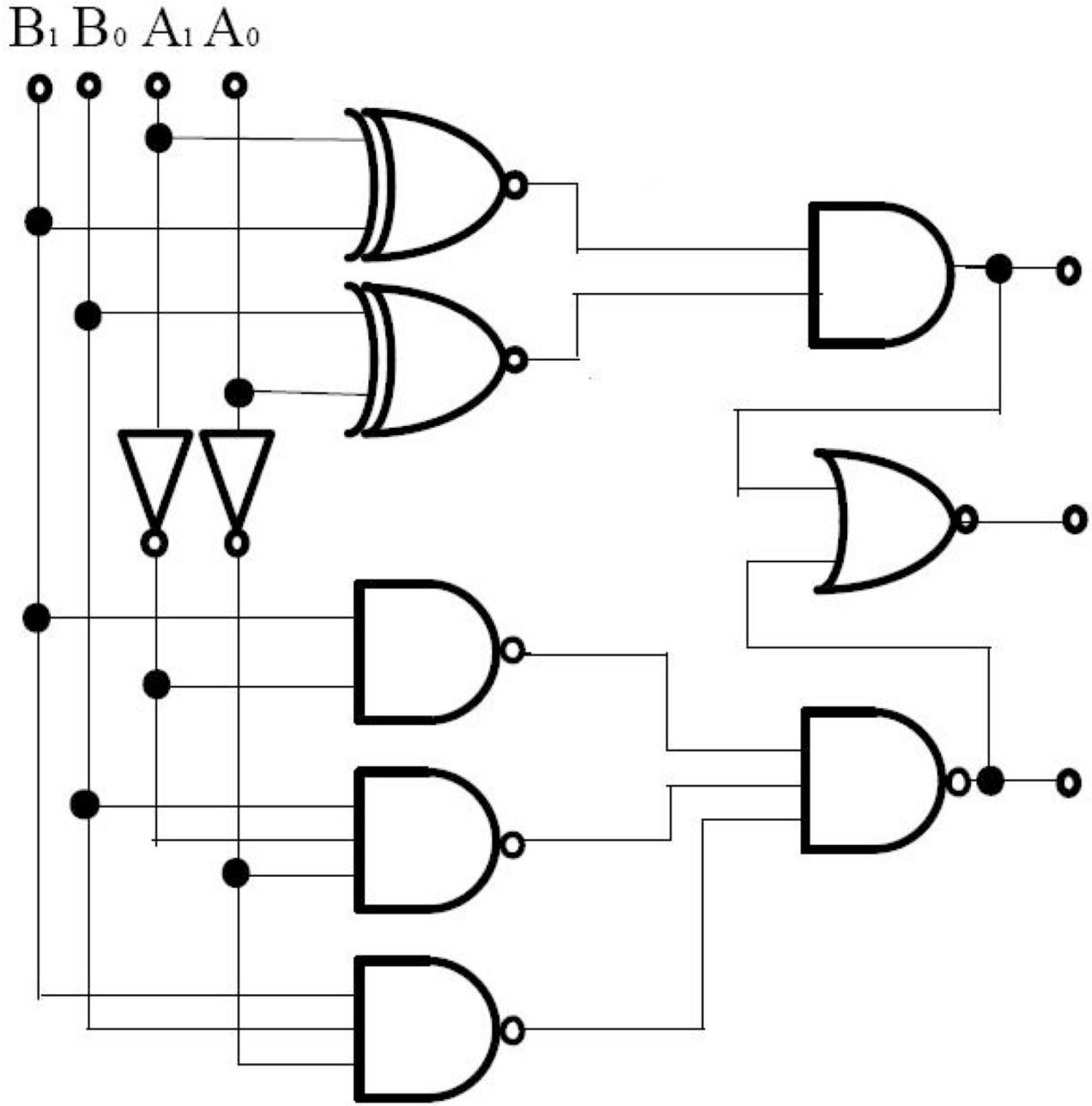
[http://www.personal.psu.edu/users///c/w/cwb5096/Old%20Site/index_files/cmpenlab4.htm]

Truth table for a two-bit digital comparator

Inputs				Outputs		
A_1	A_0	B_1	B_0	$A < B$	$A = B$	$A > B$
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

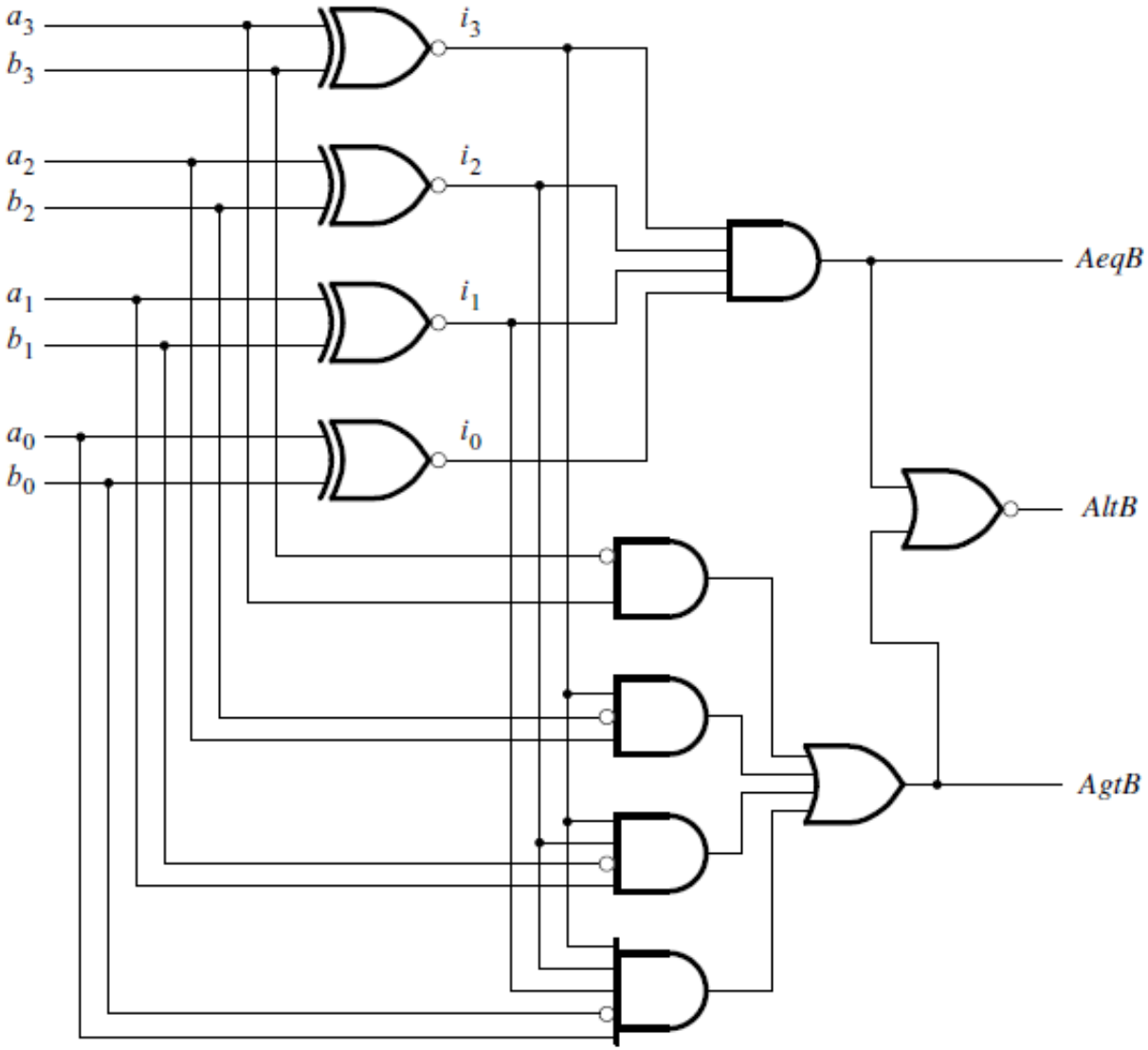
[http://en.wikipedia.org/wiki/Digital_comparator]

A two-bit digital comparator circuit



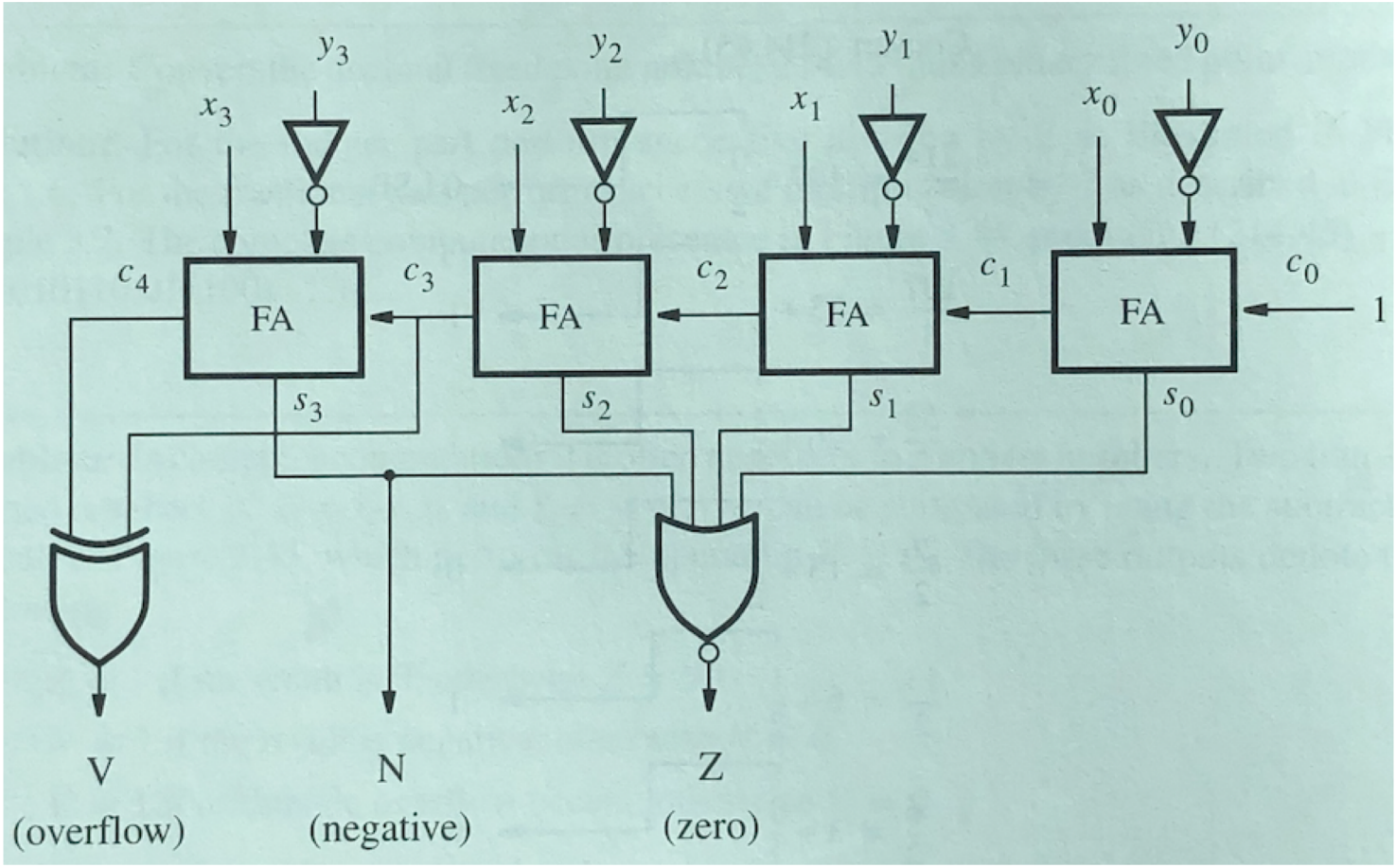
[http://forum.allaboutcircuits.com/showthread.php?t=10561]

A four-bit comparator circuit



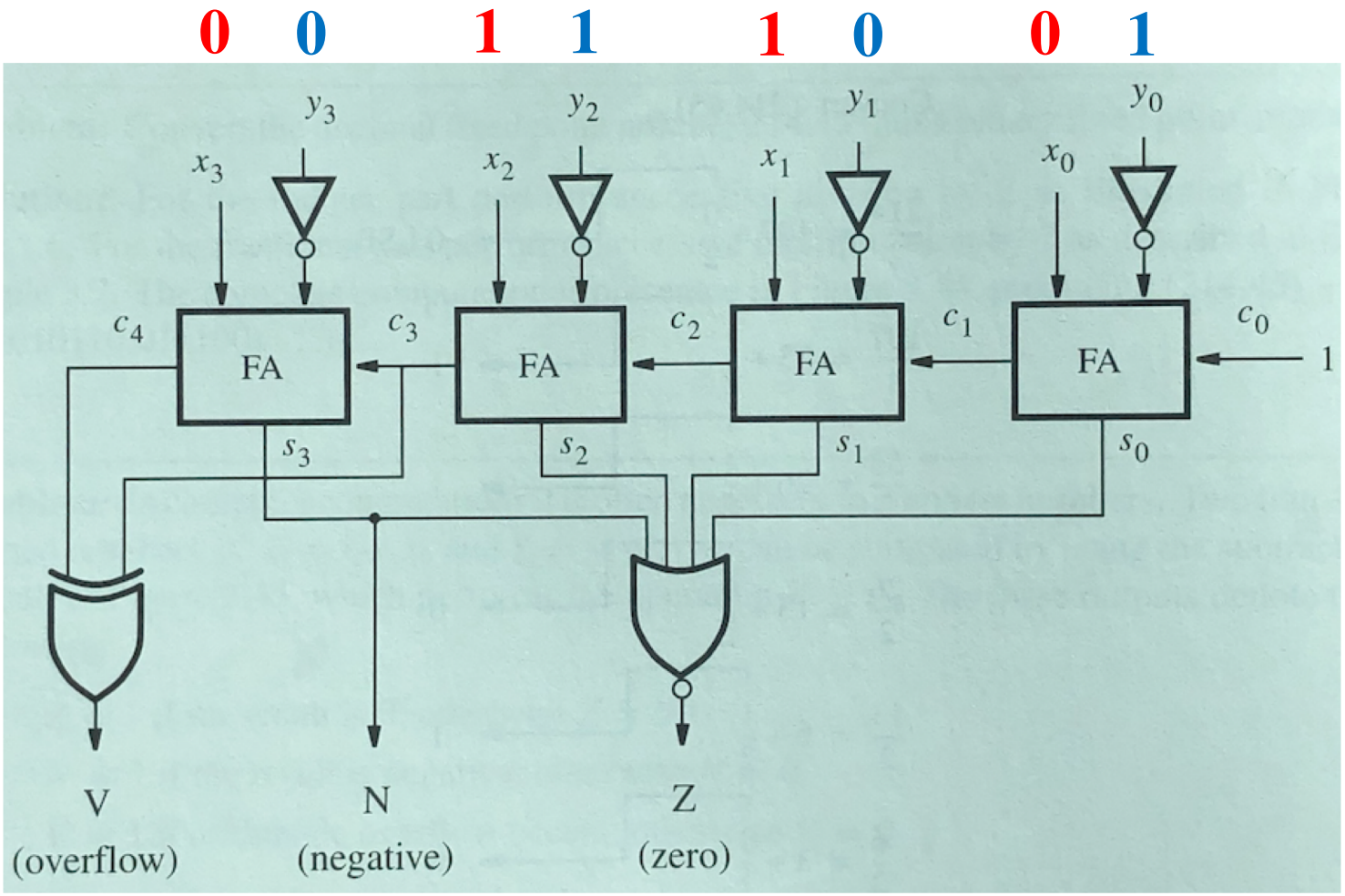
[Figure 4.22 from the textbook]

Another four-bit comparator circuit



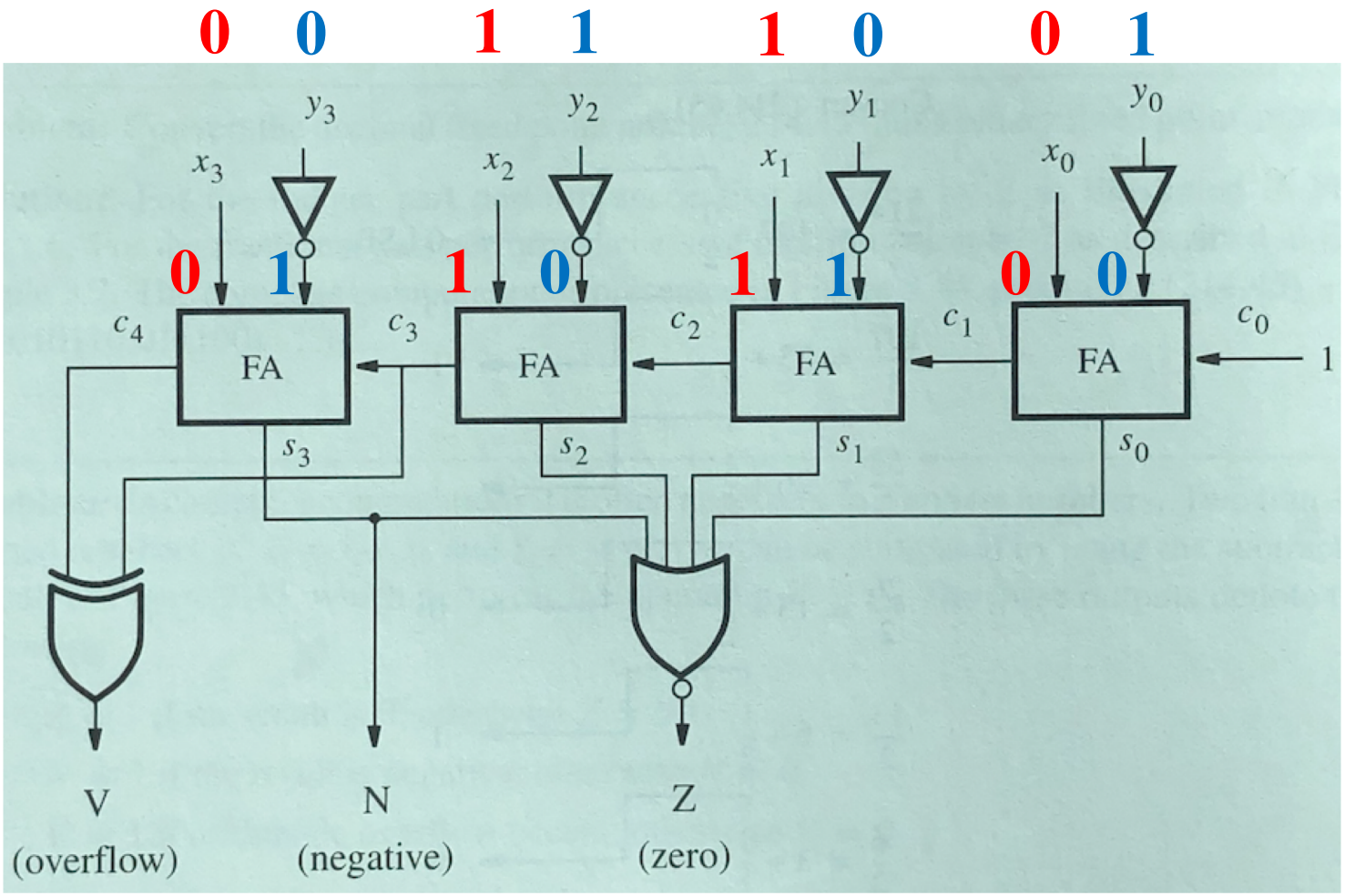
[Figure 3.45 from the textbook]

Another four-bit comparator circuit

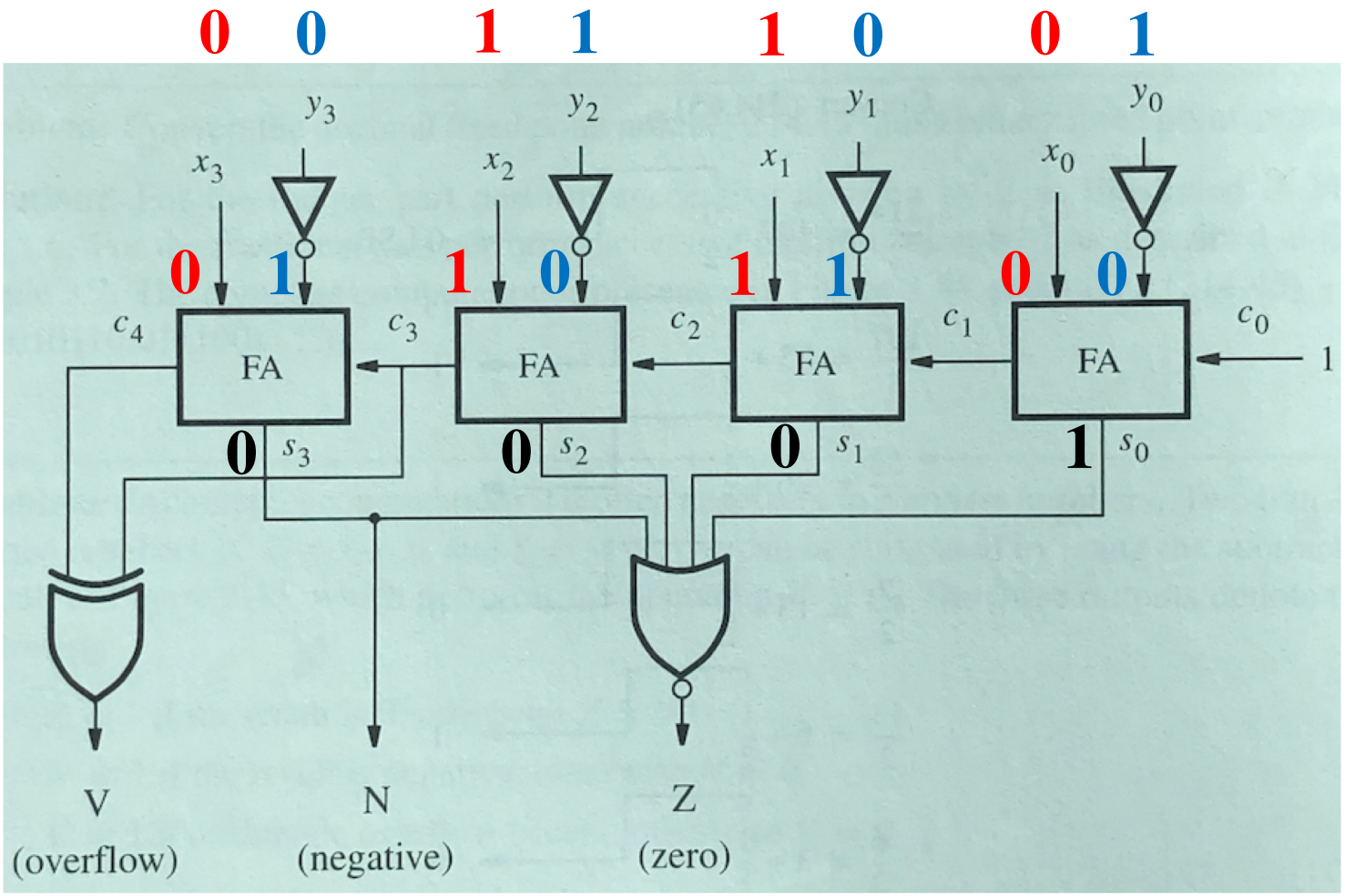


Compare 6 with 5 by subtraction (6-5).

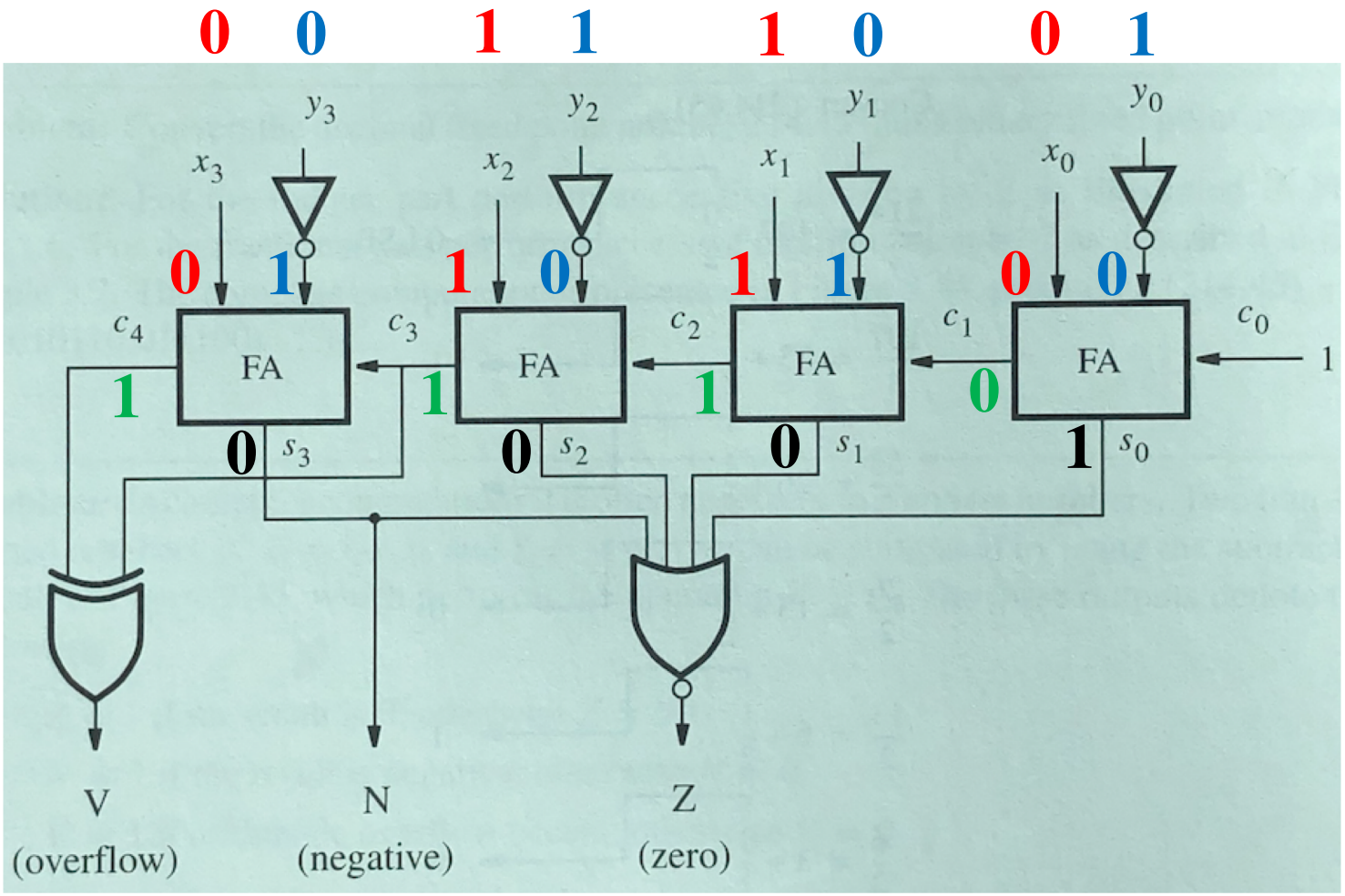
Another four-bit comparator circuit



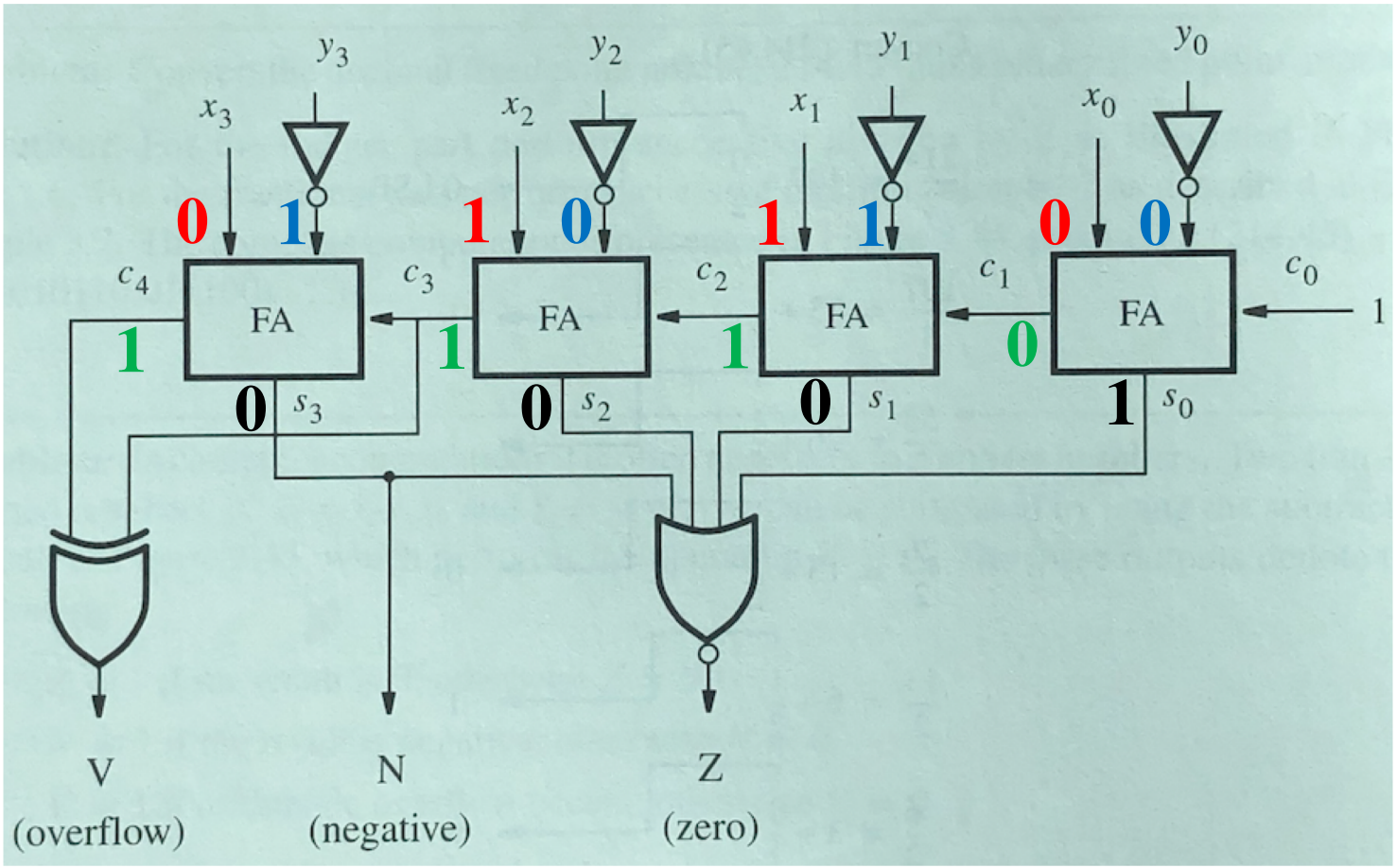
Another four-bit comparator circuit



Another four-bit comparator circuit



Another four-bit comparator circuit



Binary Coded Decimal (BCD)

Table of Binary-Coded Decimal Digits

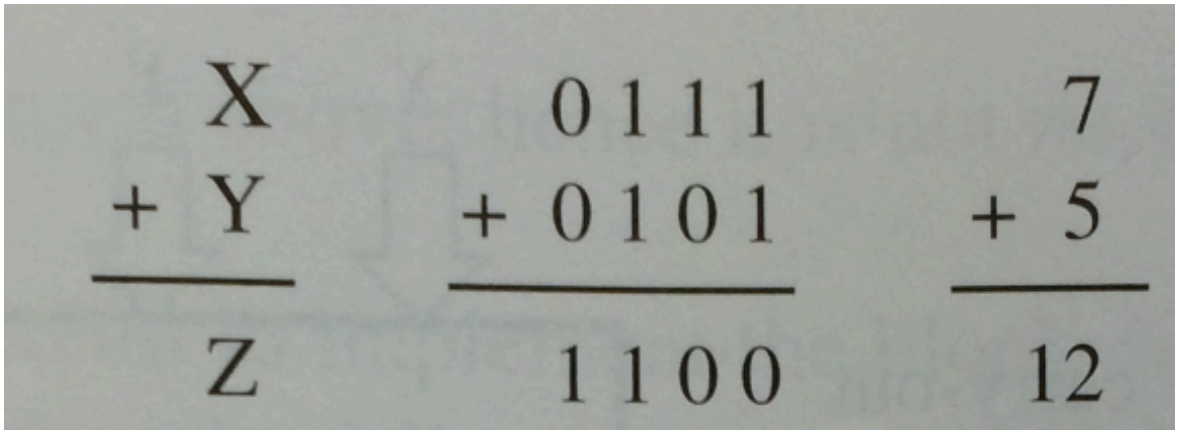
Decimal digit	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Addition of BCD digits

X	0111	7
+ Y	+ 0101	+ 5
Z	1100	12

[Figure 3.38a in the textbook]

Addition of BCD digits

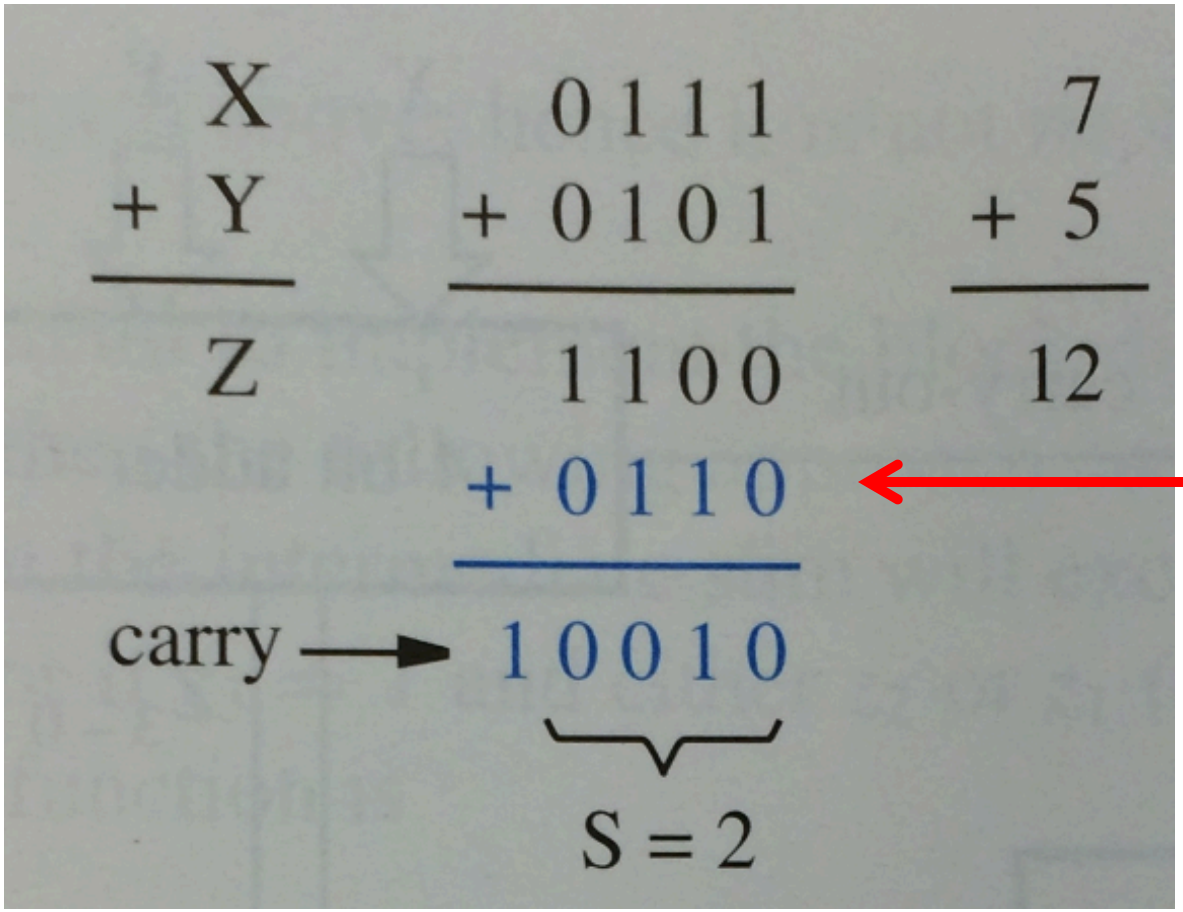


X	0 1 1 1	7
+ Y	+ 0 1 0 1	+ 5
Z	1 1 0 0	12

The result is greater than 9, which is not a valid BCD number

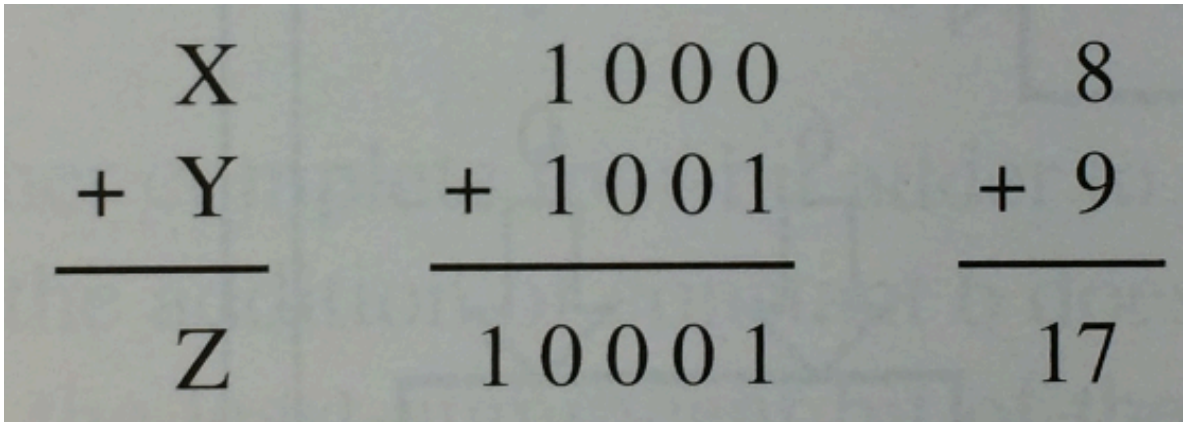
[Figure 3.38a in the textbook]

Addition of BCD digits



[Figure 3.38a in the textbook]

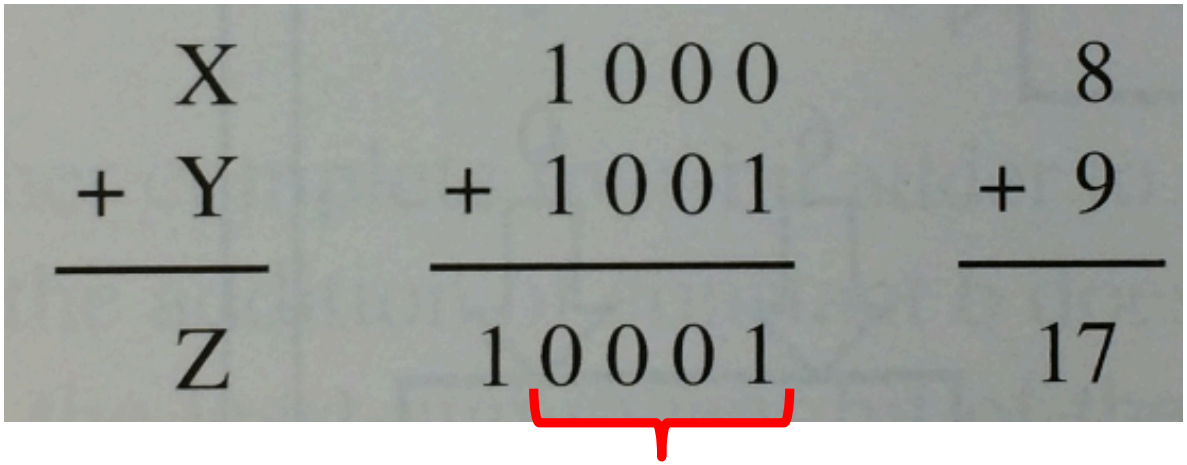
Addition of BCD digits



X	1000	8
+ Y	+ 1001	+ 9
Z	10001	17

[Figure 3.38b in the textbook]

Addition of BCD digits

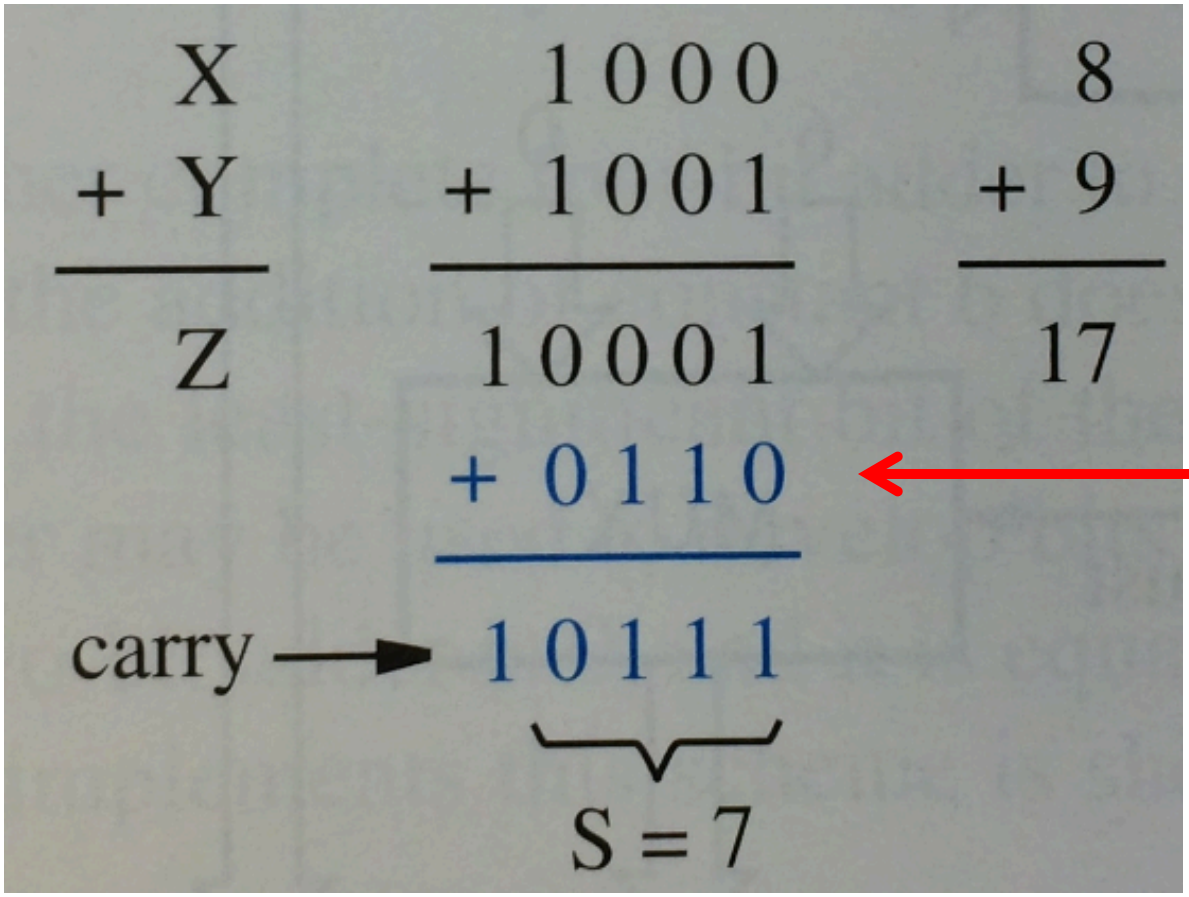


X	1000	8
+ Y	+ 1001	+ 9
Z	10001	17

The result is 1, but it should be 7

[Figure 3.38b in the textbook]

Addition of BCD digits



[Figure 3.38b in the textbook]

Why add 6?

- **Think of BCD addition as a mod 16 operation**
- **Decimal addition is mod 10 operation**

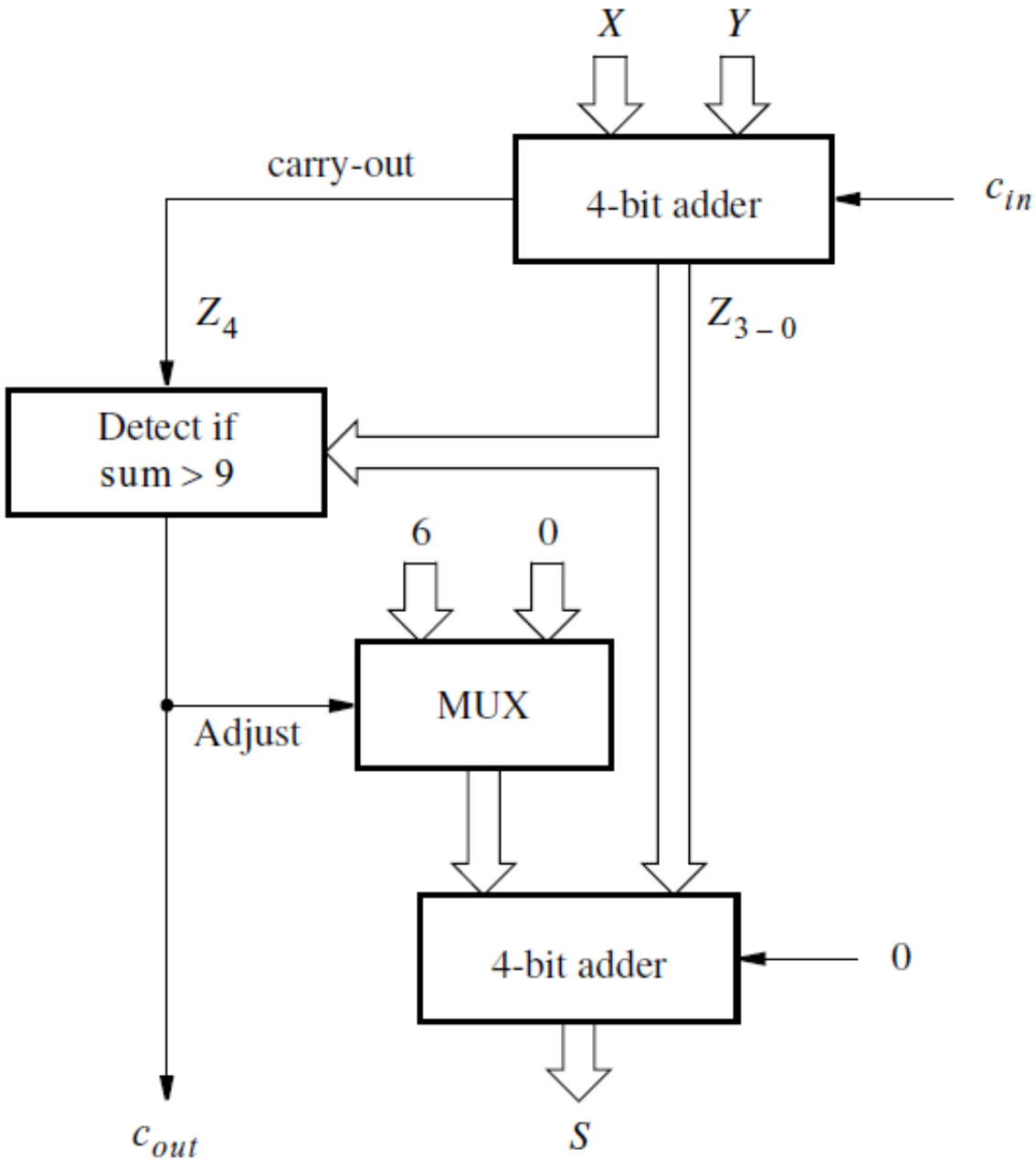
BCD Arithmetic Rules

Z = X + Y

If Z <= 9, then S=Z and carry-out = 0

If Z > 9, then S=Z+6 and carry-out =1

Block diagram for a one-digit BCD adder



[Figure 3.39 in the textbook]

How to check if the number is > 9?

7 - 0111

8 - 1000

9 - 1001

10 - 1010

11 - 1011

12 - 1100

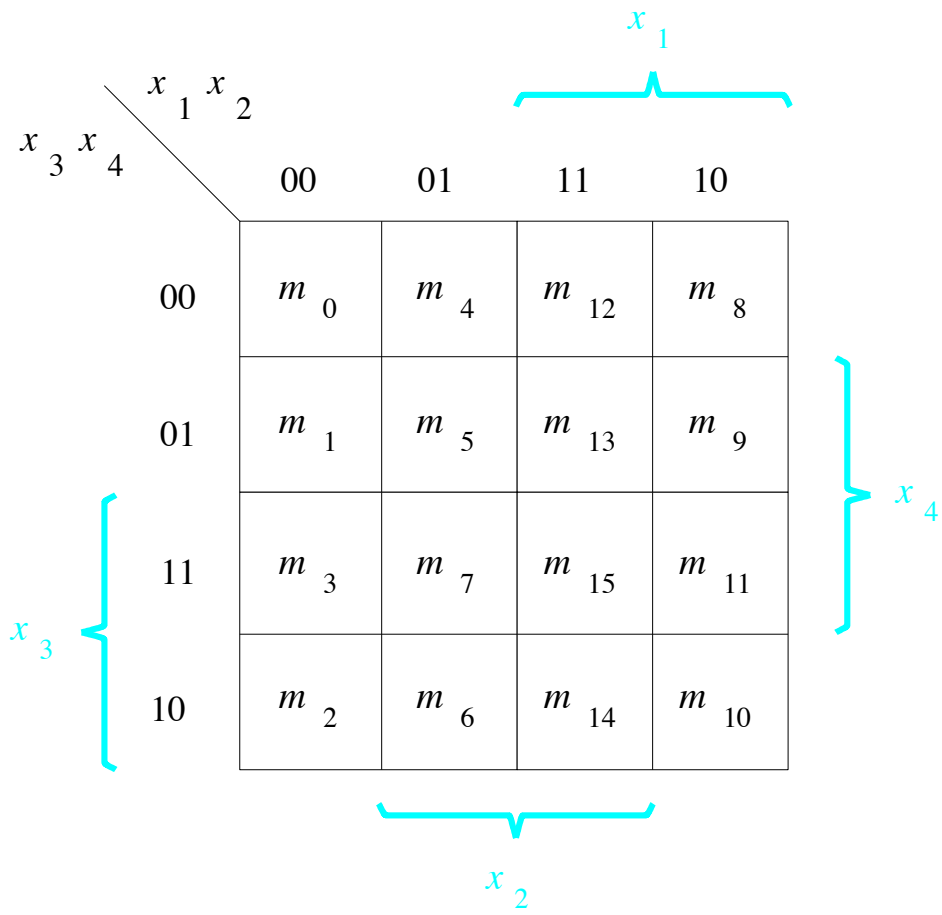
13 - 1101

14 - 1110

15 - 1111

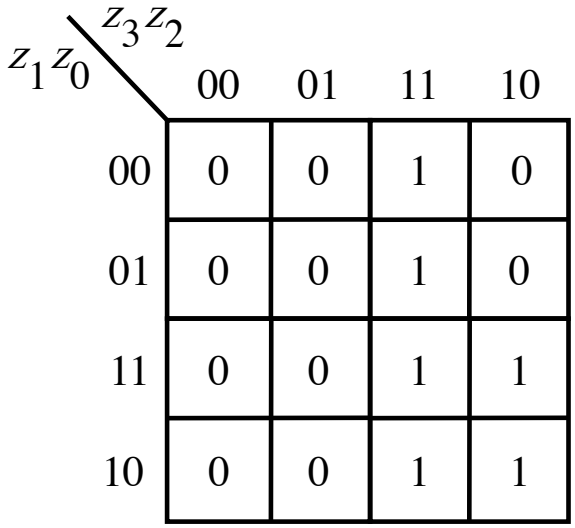
A four-variable Karnaugh map

x1	x2	x3	x4		
0	0	0	0	m0	0
0	0	0	1	m1	0
0	0	1	0	m2	0
0	0	1	1	m3	0
0	1	0	0	m4	0
0	1	0	1	m5	0
0	1	1	0	m6	0
0	1	1	1	m7	0
1	0	0	0	m8	0
1	0	0	1	m9	0
1	0	1	0	m10	1
1	0	1	1	m11	1
1	1	0	0	m12	1
1	1	0	1	m13	1
1	1	1	0	m14	1
1	1	1	1	m15	1



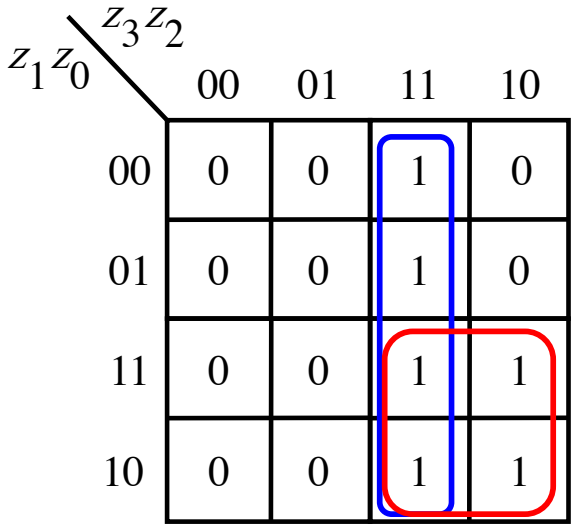
How to check if the number is > 9?

z3	z2	z1	z0		
0	0	0	0	m0	0
0	0	0	1	m1	0
0	0	1	0	m2	0
0	0	1	1	m3	0
0	1	0	0	m4	0
0	1	0	1	m5	0
0	1	1	0	m6	0
0	1	1	1	m7	0
1	0	0	0	m8	0
1	0	0	1	m9	0
1	0	1	0	m10	1
1	0	1	1	m11	1
1	1	0	0	m12	1
1	1	0	1	m13	1
1	1	1	0	m14	1
1	1	1	1	m15	1



How to check if the number is > 9?

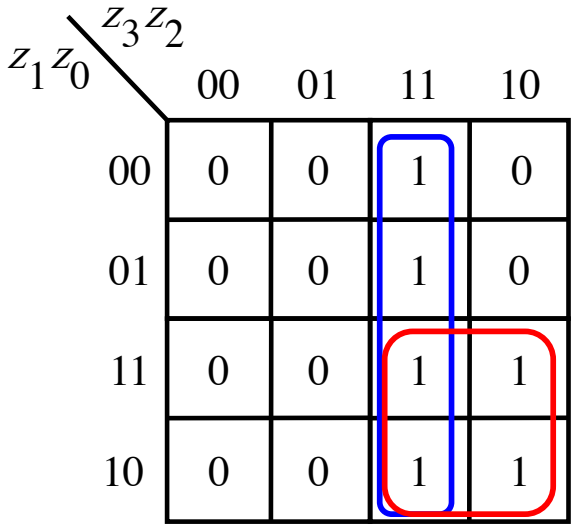
z3	z2	z1	z0		
0	0	0	0	m0	0
0	0	0	1	m1	0
0	0	1	0	m2	0
0	0	1	1	m3	0
0	1	0	0	m4	0
0	1	0	1	m5	0
0	1	1	0	m6	0
0	1	1	1	m7	0
1	0	0	0	m8	0
1	0	0	1	m9	0
1	0	1	0	m10	1
1	0	1	1	m11	1
1	1	0	0	m12	1
1	1	0	1	m13	1
1	1	1	0	m14	1
1	1	1	1	m15	1



$$f = z_3z_2 + z_3z_1$$

How to check if the number is > 9?

z3	z2	z1	z0		
0	0	0	0	m0	0
0	0	0	1	m1	0
0	0	1	0	m2	0
0	0	1	1	m3	0
0	1	0	0	m4	0
0	1	0	1	m5	0
0	1	1	0	m6	0
0	1	1	1	m7	0
1	0	0	0	m8	0
1	0	0	1	m9	0
1	0	1	0	m10	1
1	0	1	1	m11	1
1	1	0	0	m12	1
1	1	0	1	m13	1
1	1	1	0	m14	1
1	1	1	1	m15	1



$$f = z_3z_2 + z_3z_1$$

In addition, also check if there was a carry

$$f = \text{carry-out} + z_3z_2 + z_3z_1$$

Verilog code for a one-digit BCD adder

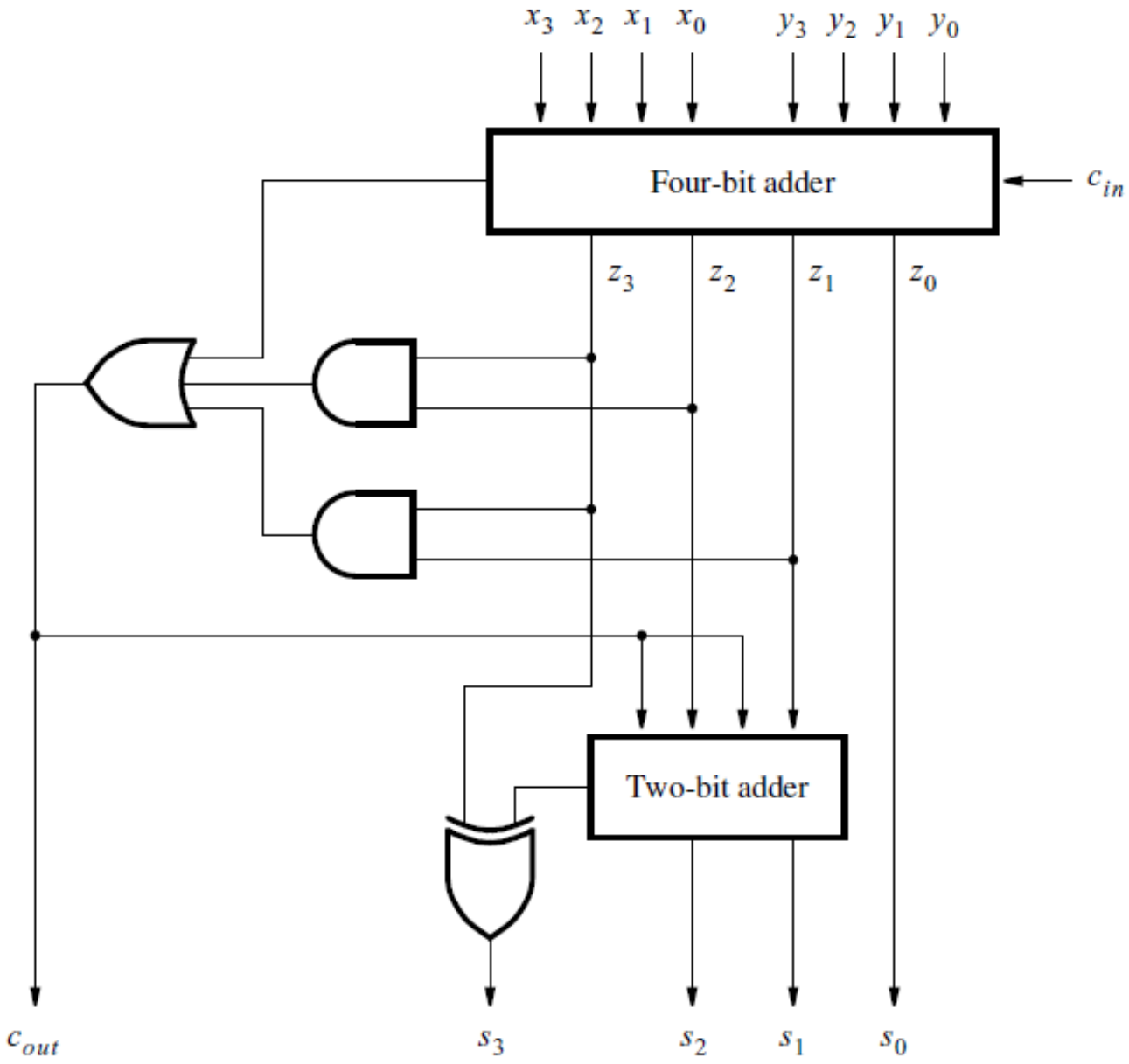
```
module bcdadd(Cin, X, Y, S, Cout);
    input Cin;
    input [3:0] X,Y;
    output reg [3:0] S;
    output reg Cout;
    reg [4:0] Z;

    always@ (X, Y, Cin)
    begin
        Z = X + Y + Cin;
        if (Z < 10)
            {Cout,S} = Z;
        else
            {Cout, S} = Z + 6;
    end

endmodule
```

[Figure 3.40 in the textbook]

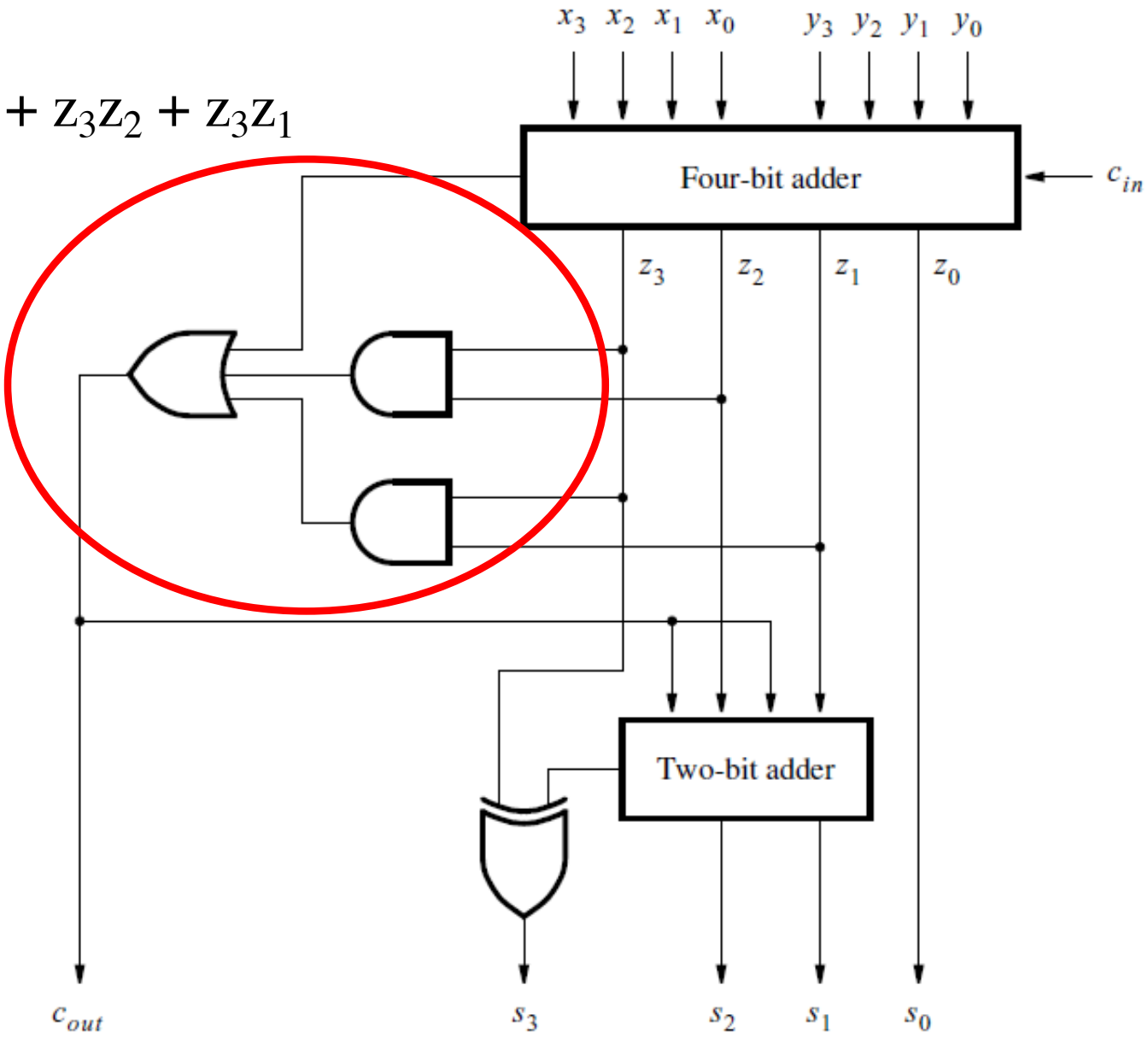
Circuit for a one-digit BCD adder



[Figure 3.41 in the textbook]

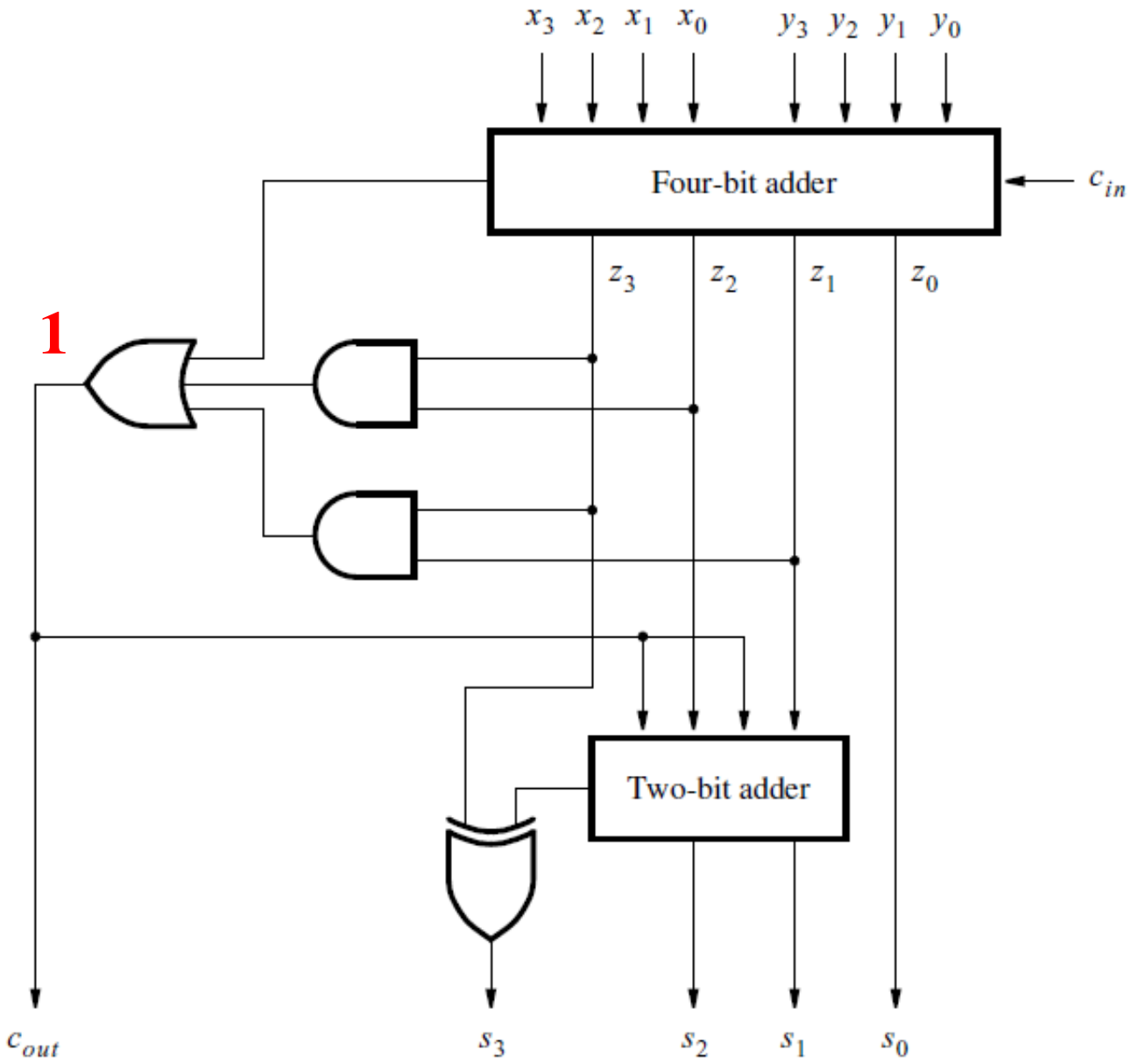
Circuit for a one-digit BCD adder

carry-out + z_3z_2 + z_3z_1



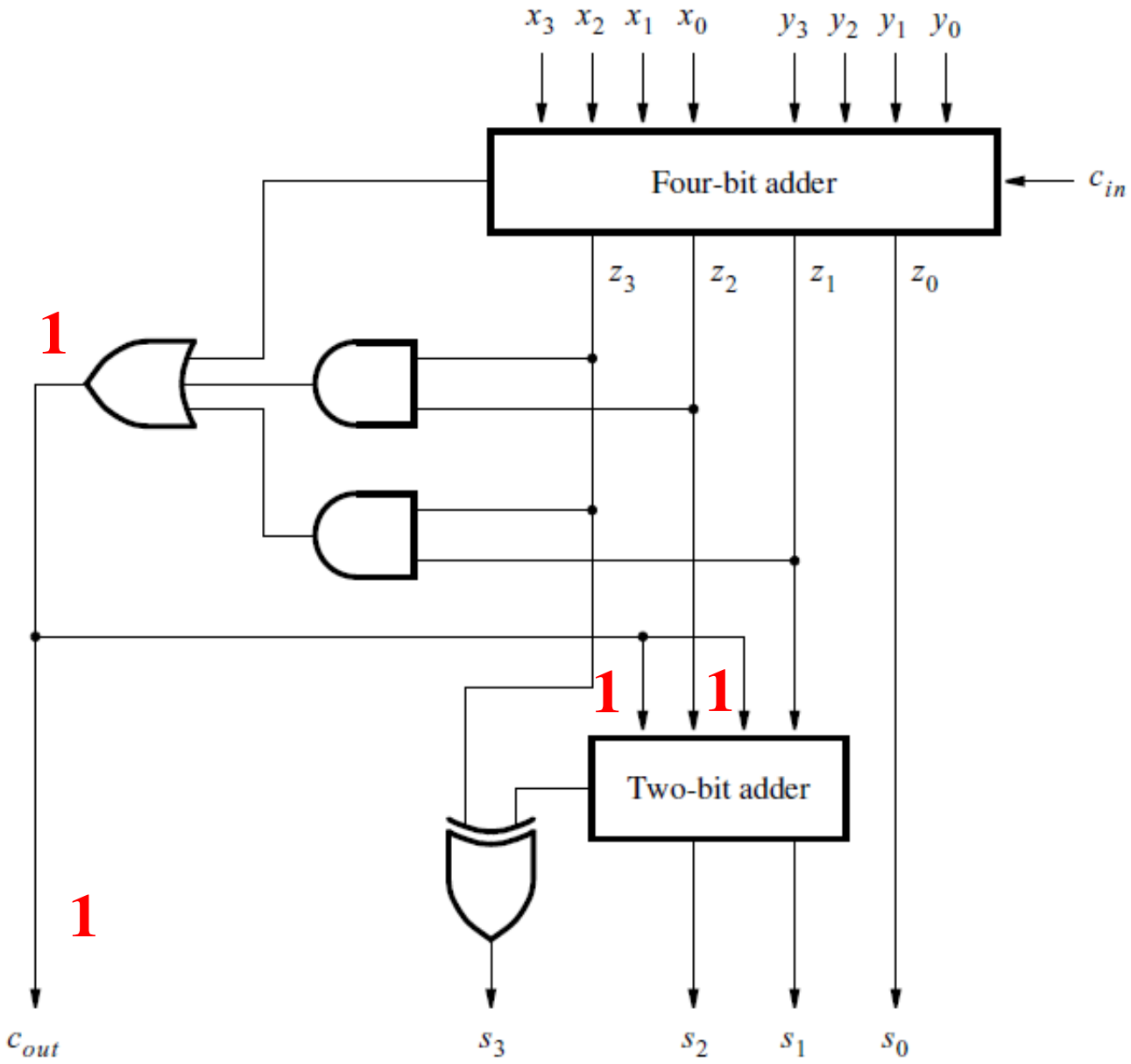
[Figure 3.41 in the textbook]

Circuit for a one-digit BCD adder



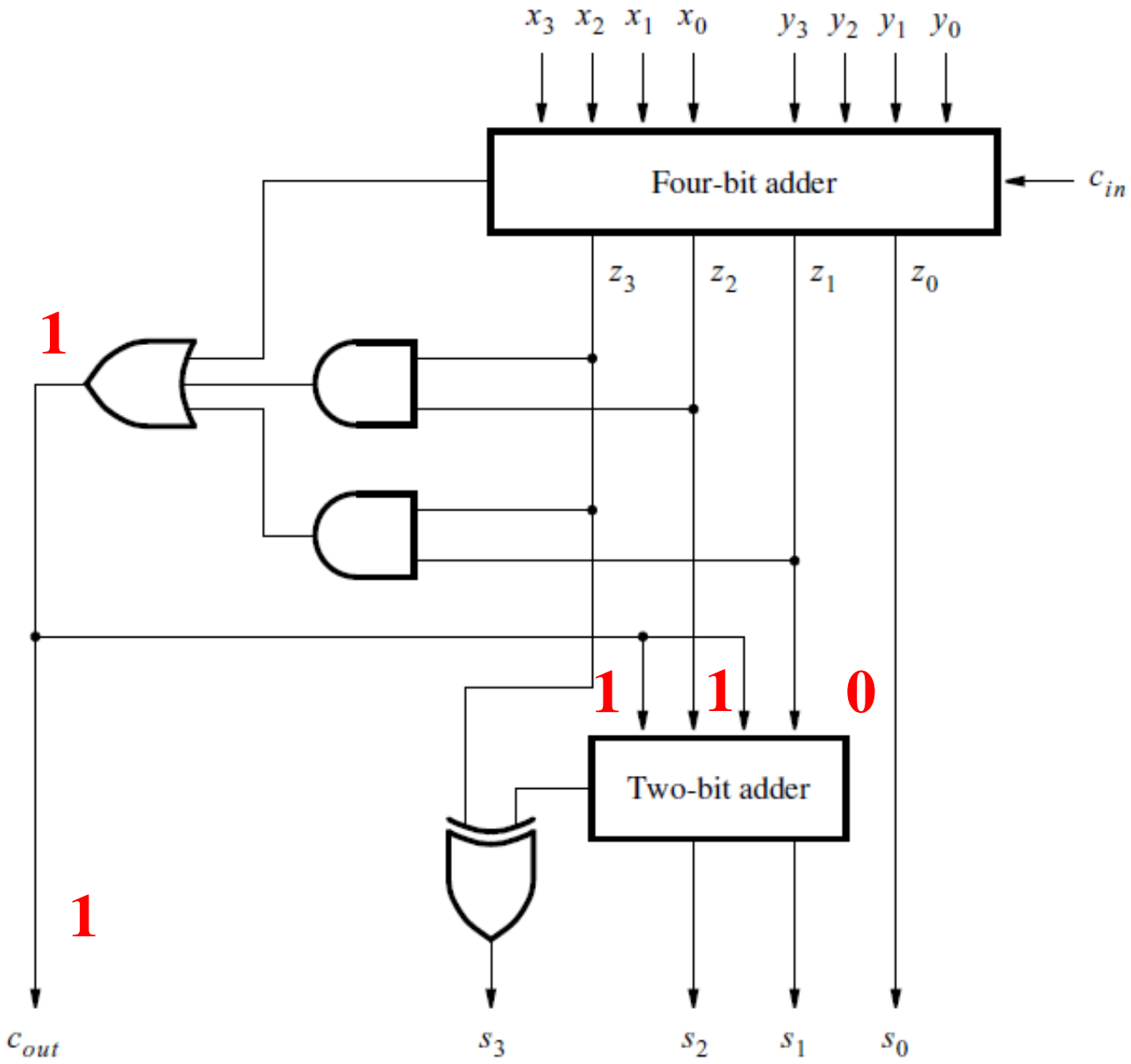
[Figure 3.41 in the textbook]

Circuit for a one-digit BCD adder



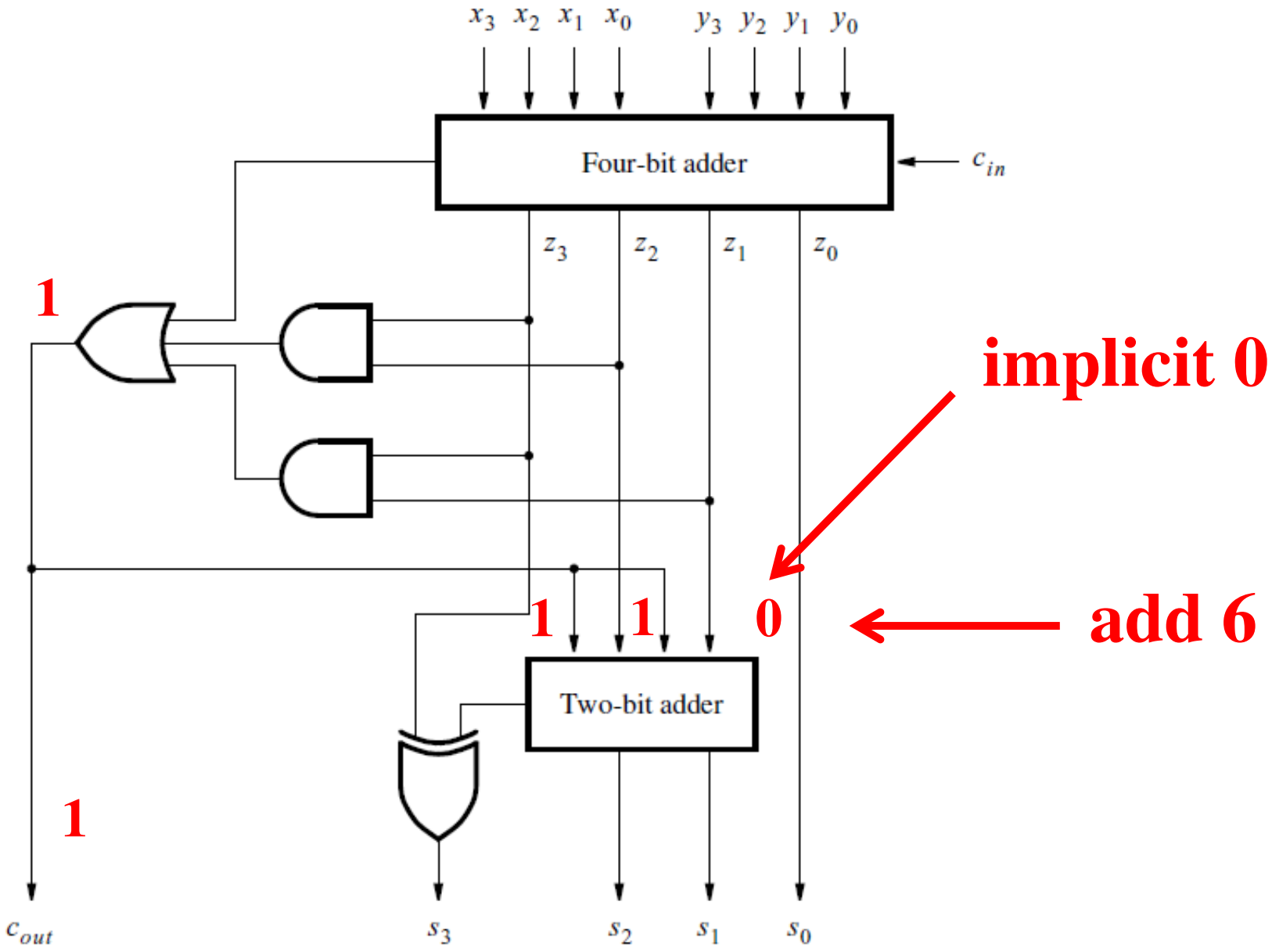
[Figure 3.41 in the textbook]

Circuit for a one-digit BCD adder



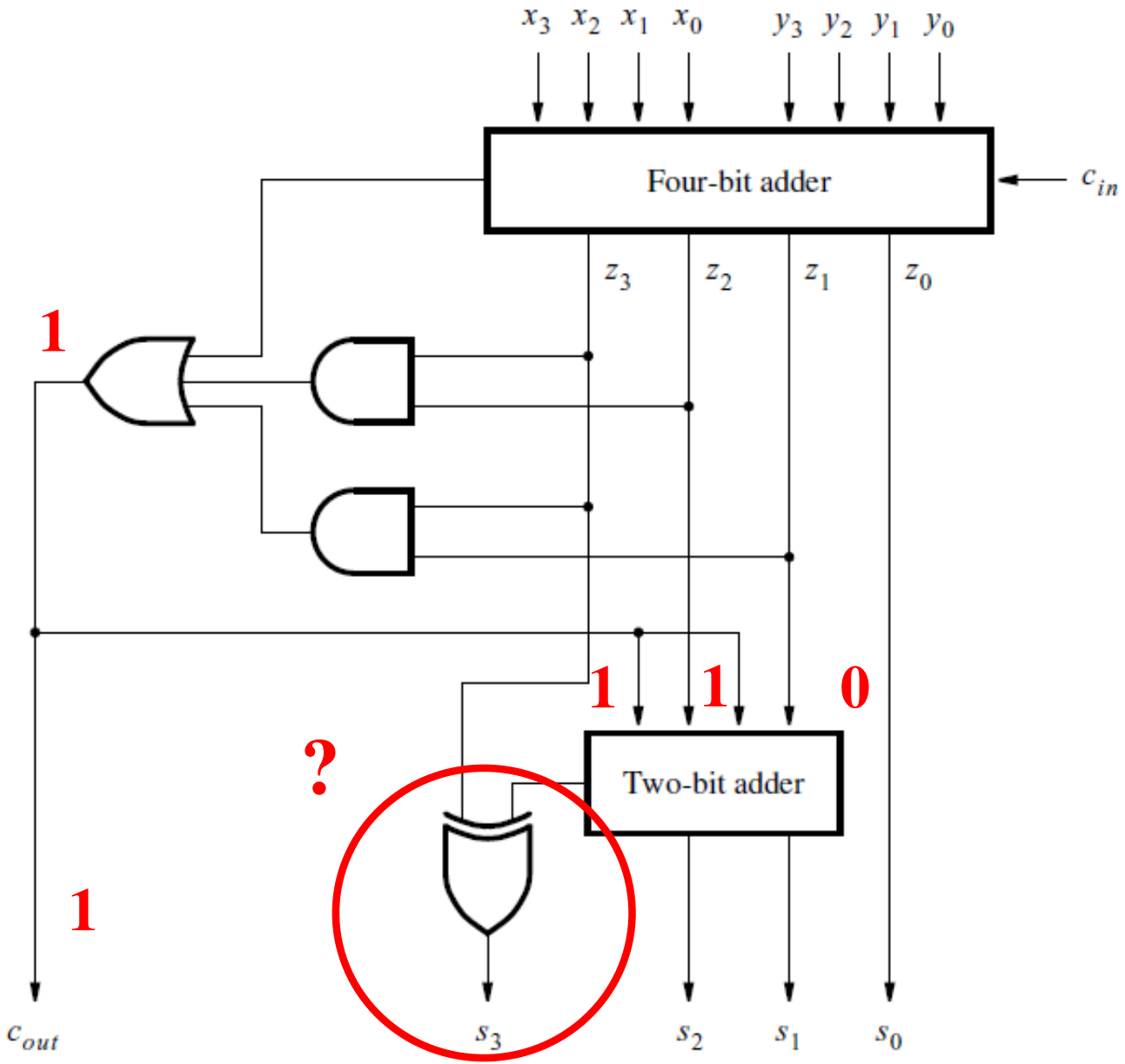
[Figure 3.41 in the textbook]

Circuit for a one-digit BCD adder



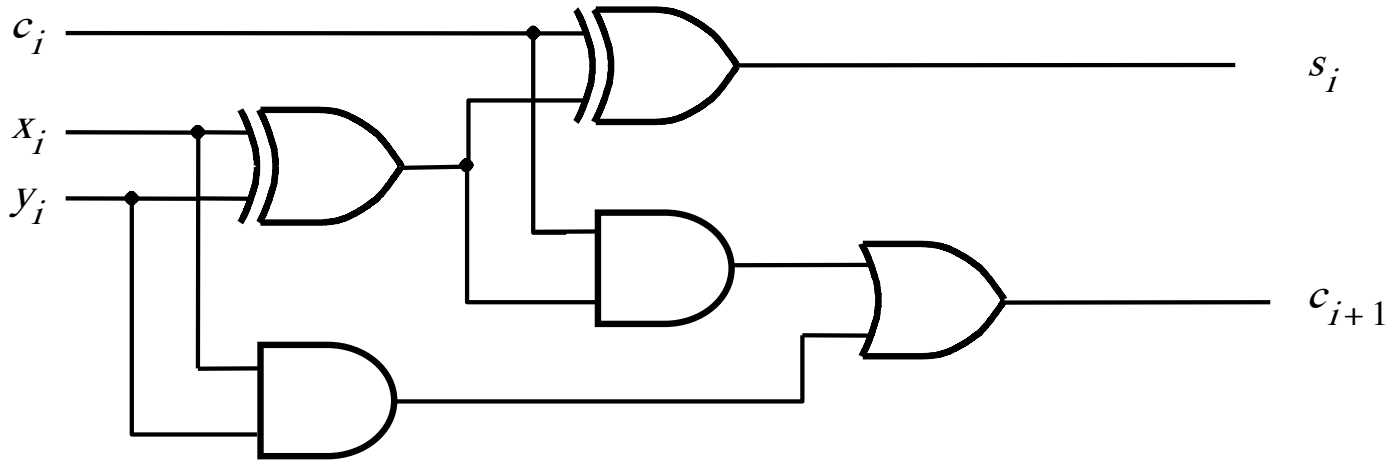
[Figure 3.41 in the textbook]

Circuit for a one-digit BCD adder



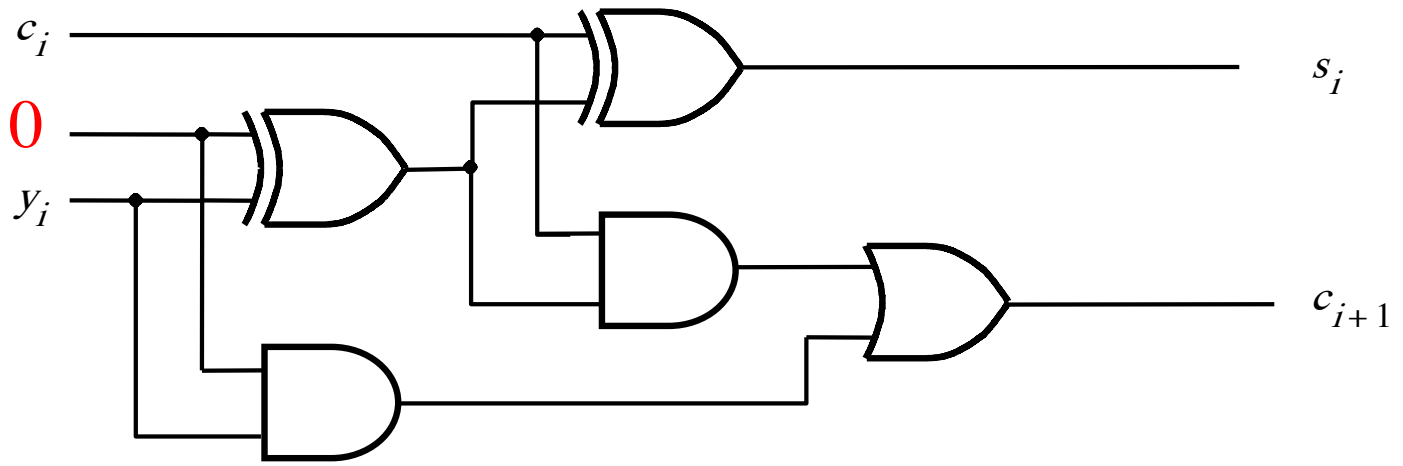
[Figure 3.41 in the textbook]

Simplification of the Full-Adder circuit when x_i=0

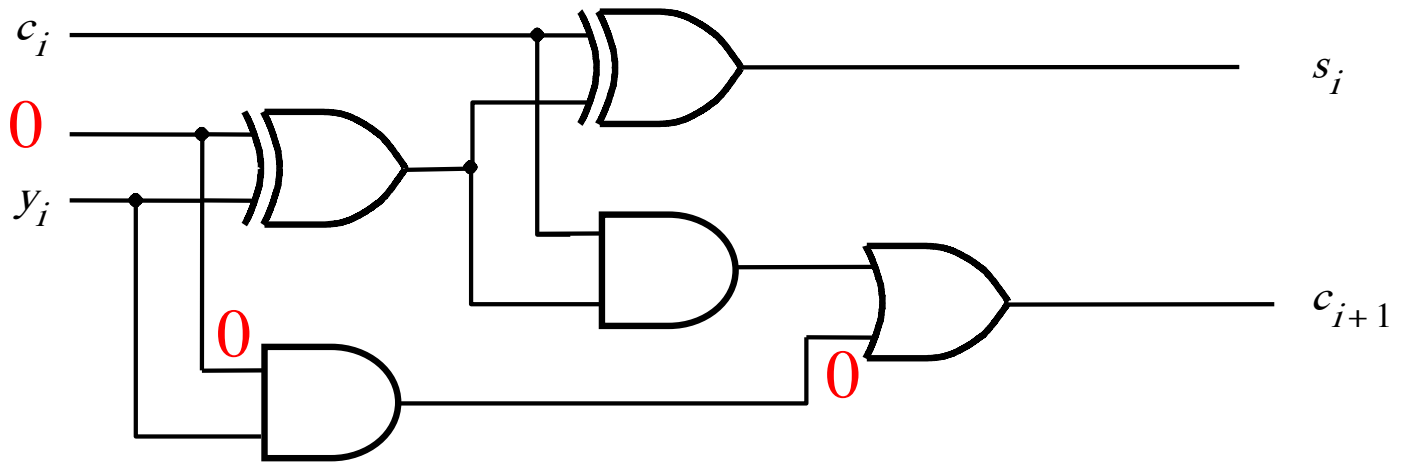


[Figure 3.4b from the textbook]

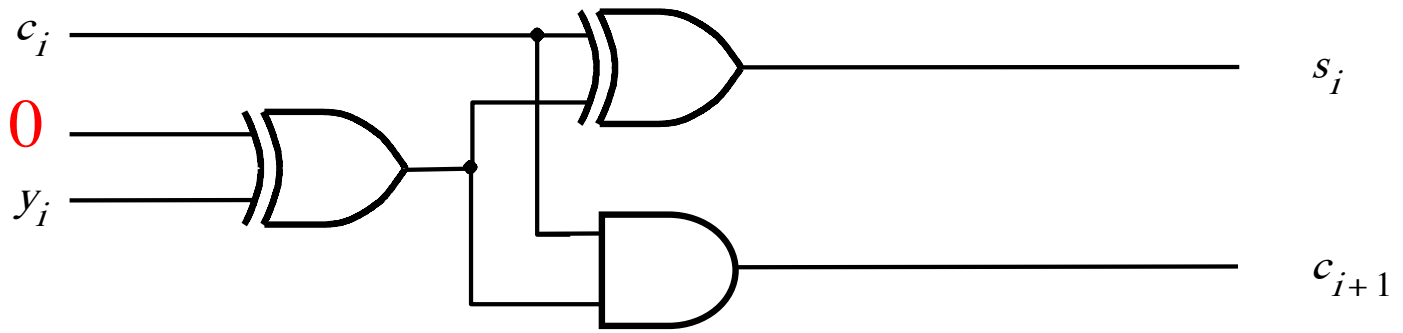
Simplification of the Full-Adder circuit when x_i=0



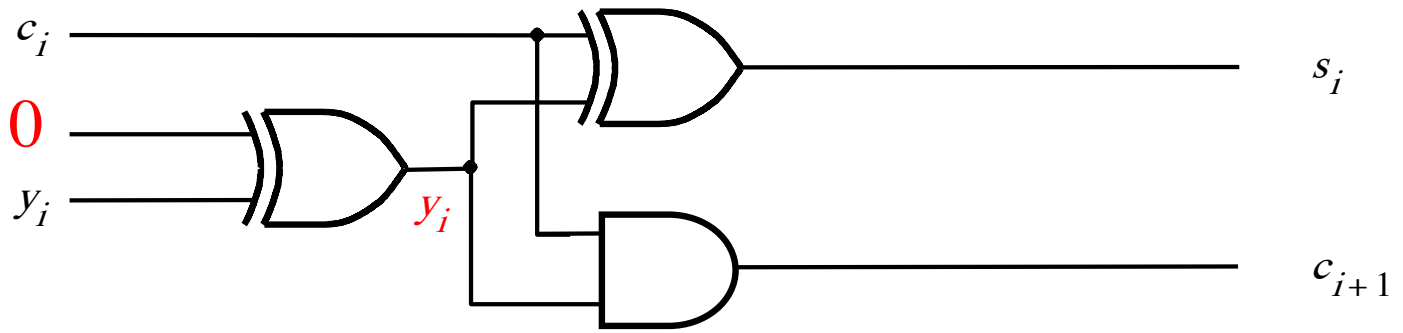
Simplification of the Full-Adder circuit when x_i=0



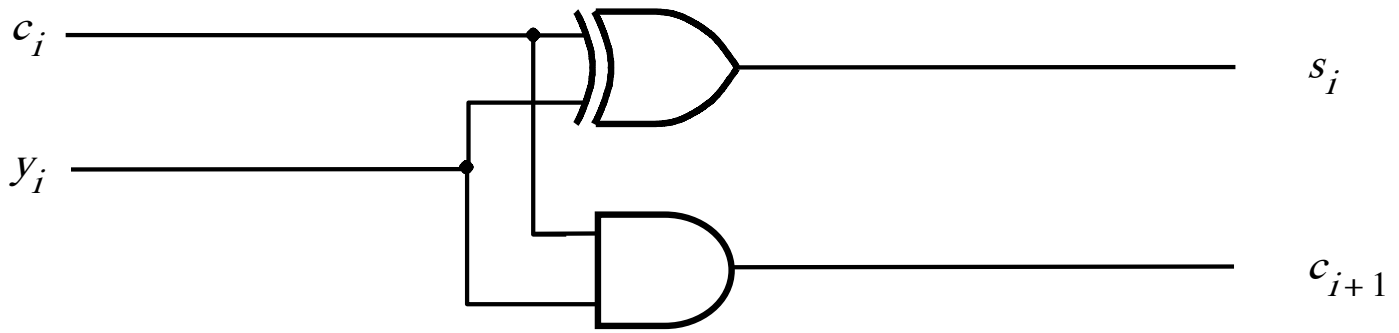
Simplification of the Full-Adder circuit when x_i=0



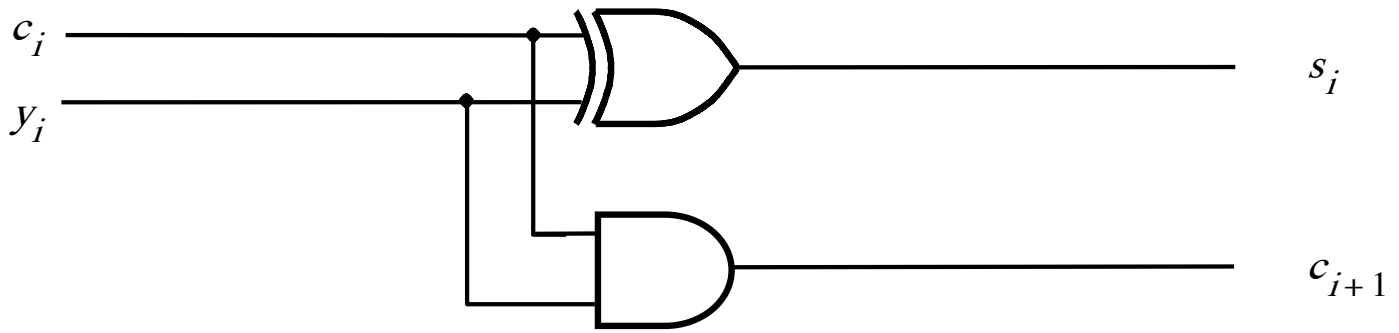
Simplification of the Full-Adder circuit when x_i=0



Simplification of the Full-Adder circuit when x_i=0

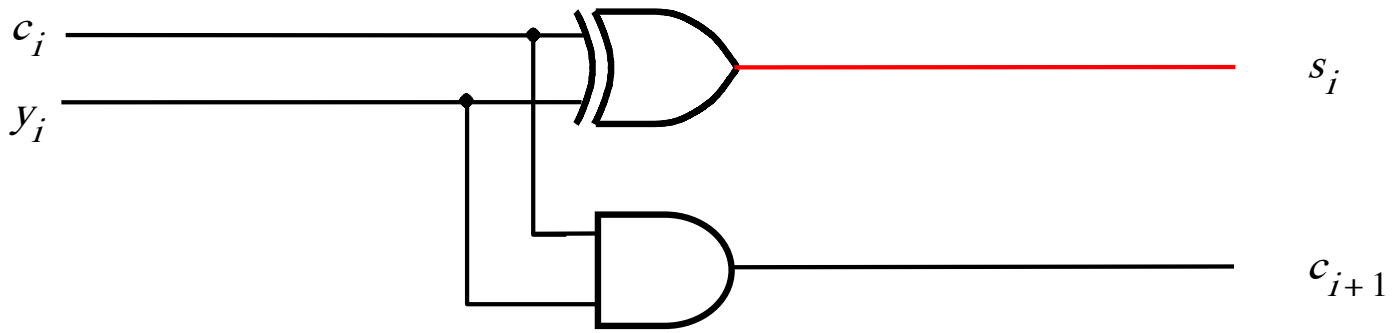


Simplification of the Full-Adder circuit when x_i=0



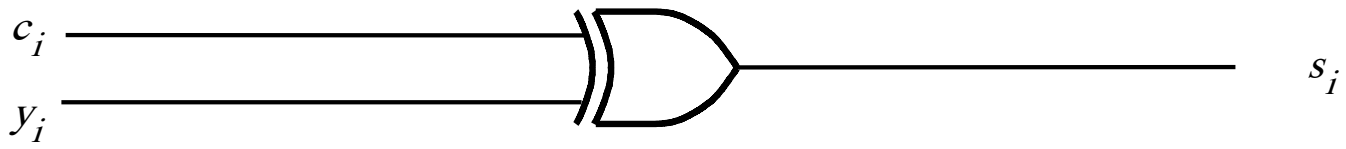
It reduces to a half-adder.

Simplification of the Full-Adder circuit when x_i=0



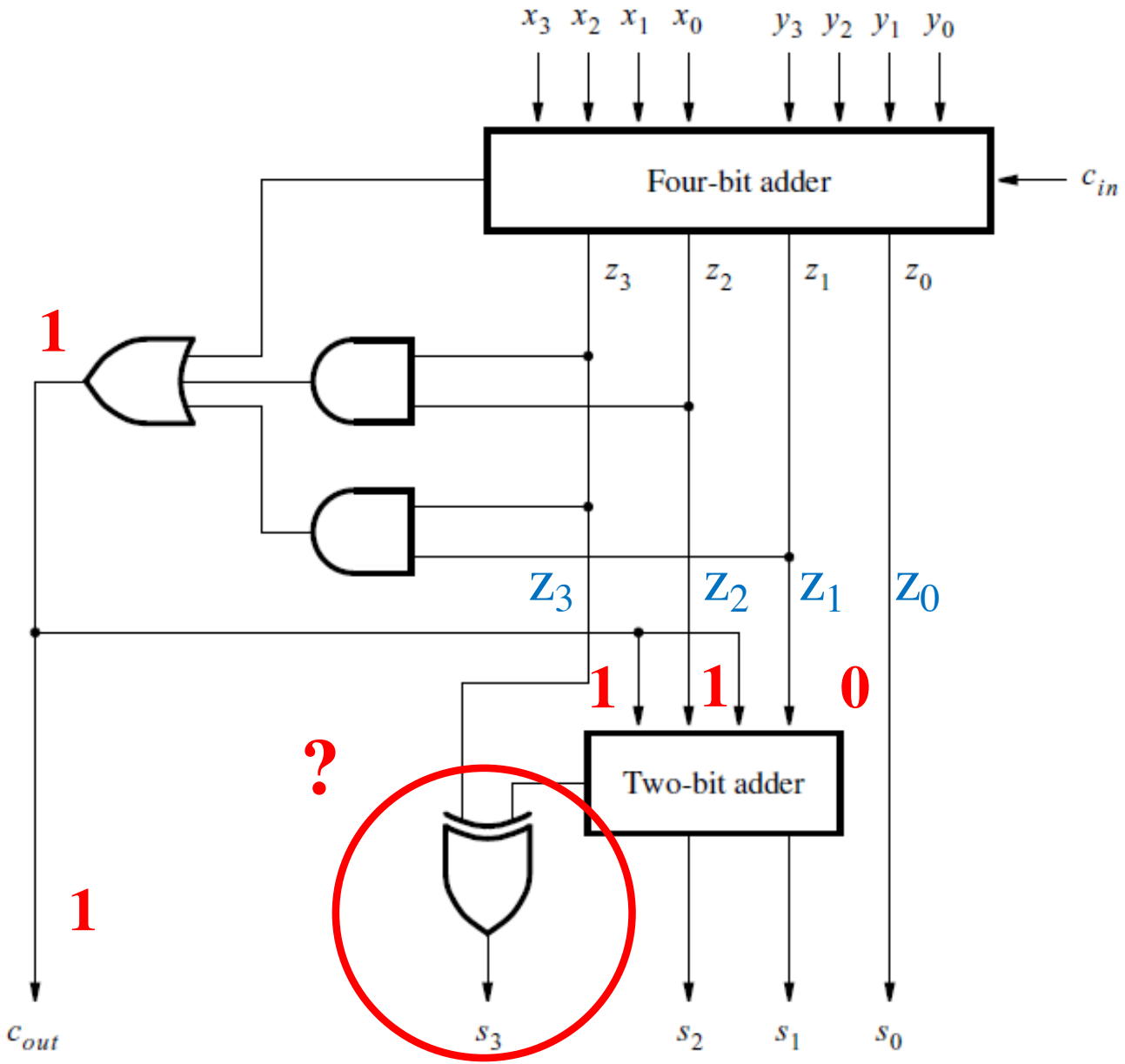
But if we only need the sum bit …

Simplification of the Full-Adder circuit when x_i=0



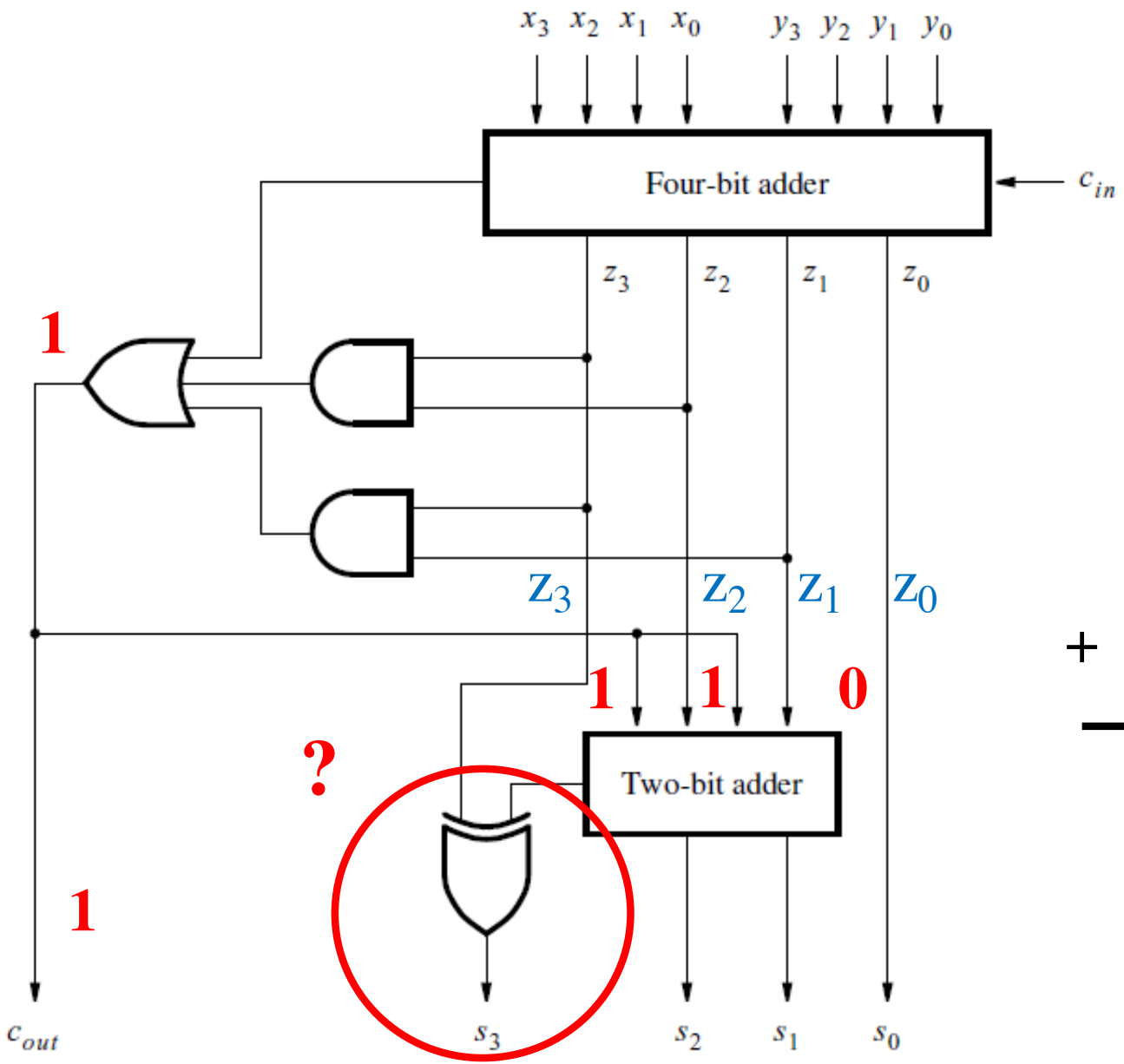
… it reduces to an XOR.

Circuit for a one-digit BCD adder



[Figure 3.41 in the textbook]

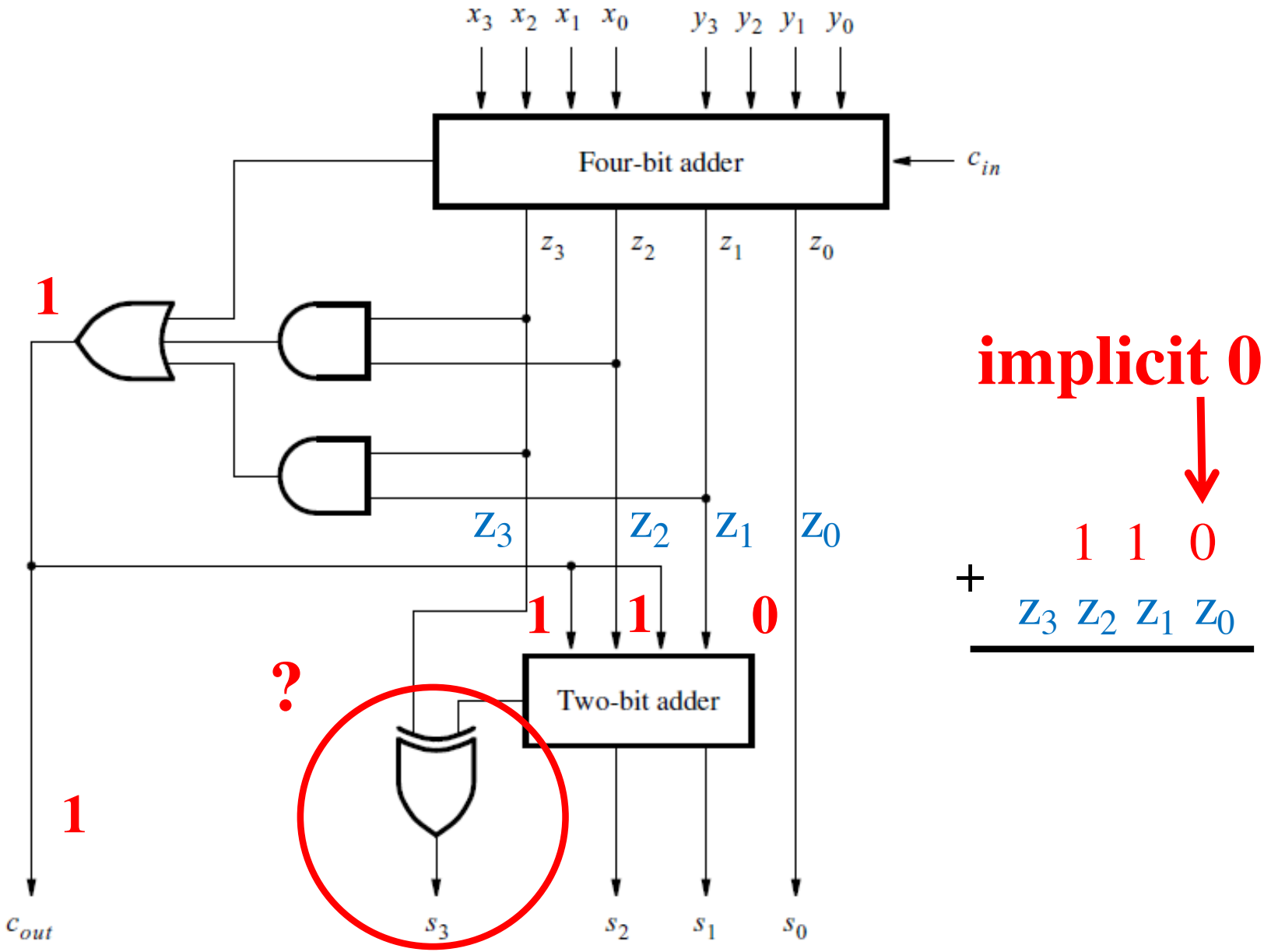
Circuit for a one-digit BCD adder



$$\begin{array}{r} 1\ 1\ 0 \\ +\ Z_3\ Z_2\ Z_1\ Z_0 \\ \hline \end{array}$$

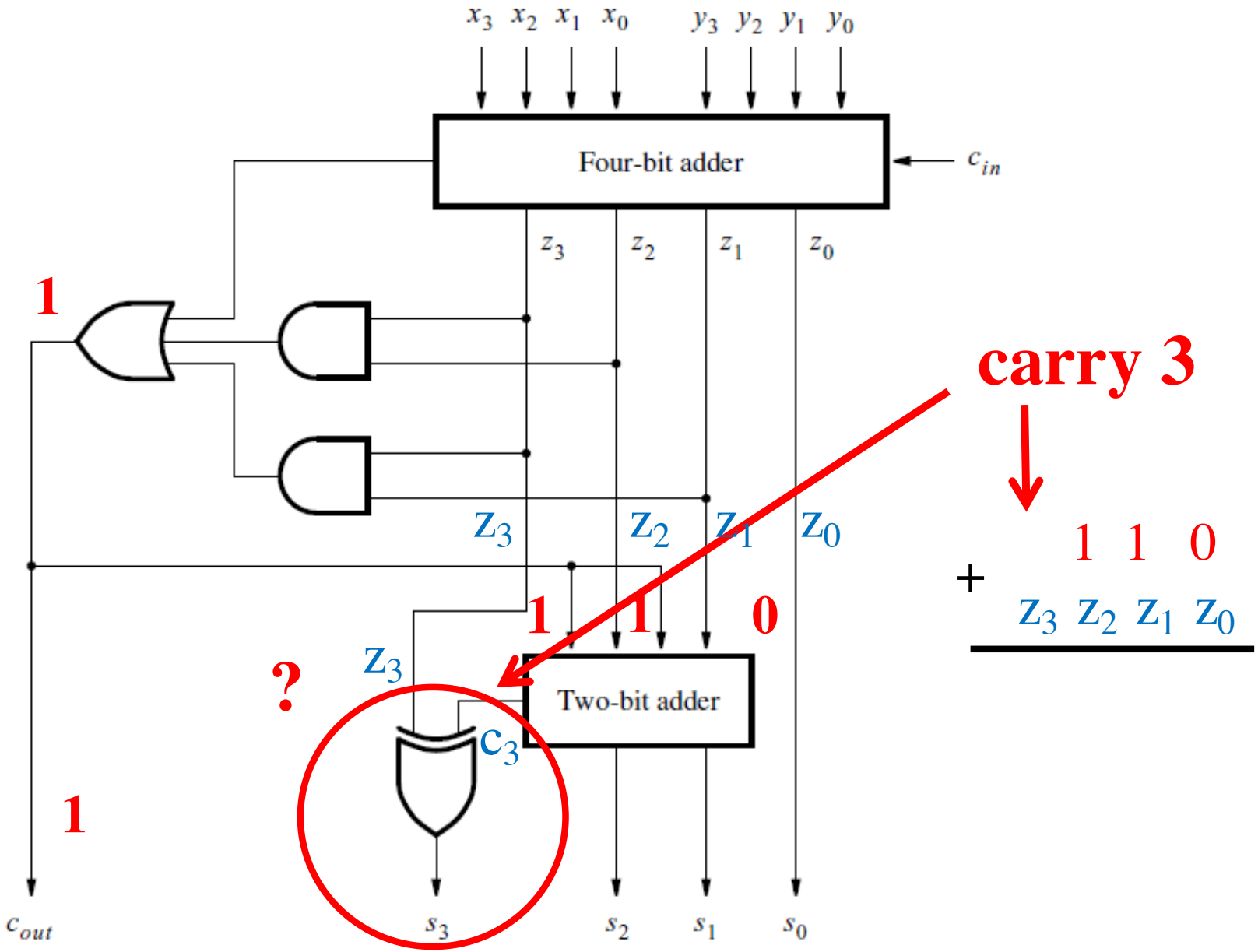
[Figure 3.41 in the textbook]

Circuit for a one-digit BCD adder



[Figure 3.41 in the textbook]

Circuit for a one-digit BCD adder



[Figure 3.41 in the textbook]

Questions?

THE END