ELECTRICAL
Engineering KMITNB

Digital Circuits and Logic Design

Lecture6-2 :Number Representation and Arithmetic Circuits

Note: Half-Adder (HA)

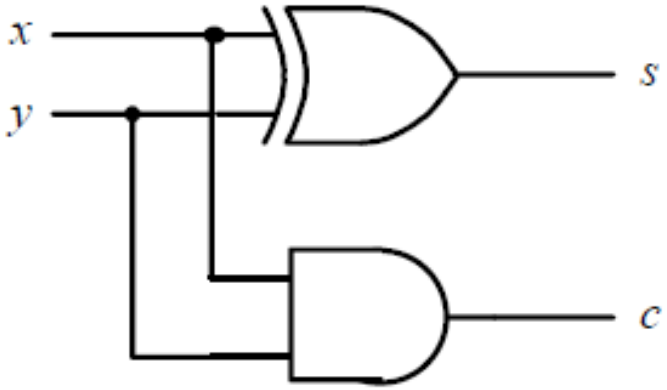
x	0	0	1	1
$+ y$	$+ 0$	$+ 1$	$+ 0$	$+ 1$
$c\ s$	0 0	0 1	0 1	1 0

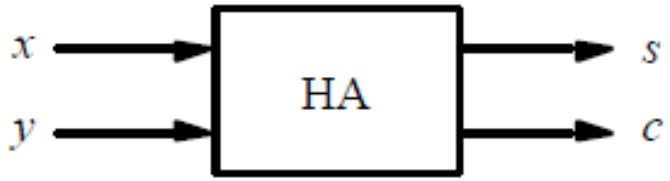
Carry Sum

(a) The four possible cases

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

(b) Truth table



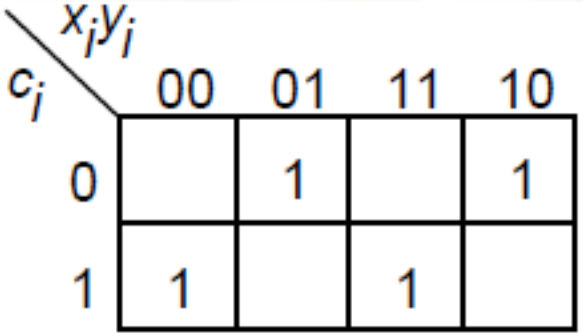


Note: Full-Adder (FA)

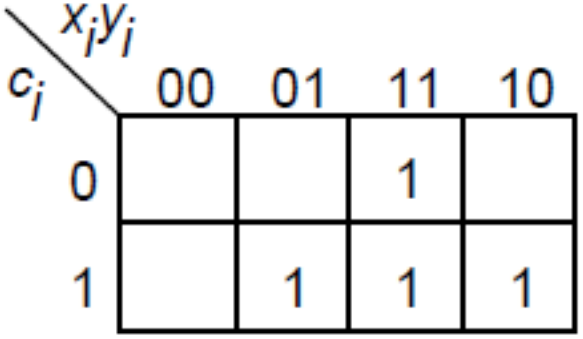
Addition at the i-th bit:

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



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



$$c_{i+1} = x_iy_i + x_ic_i + y_ic_i$$

(b) Karnaugh maps

Note: Full-Adder (FA)

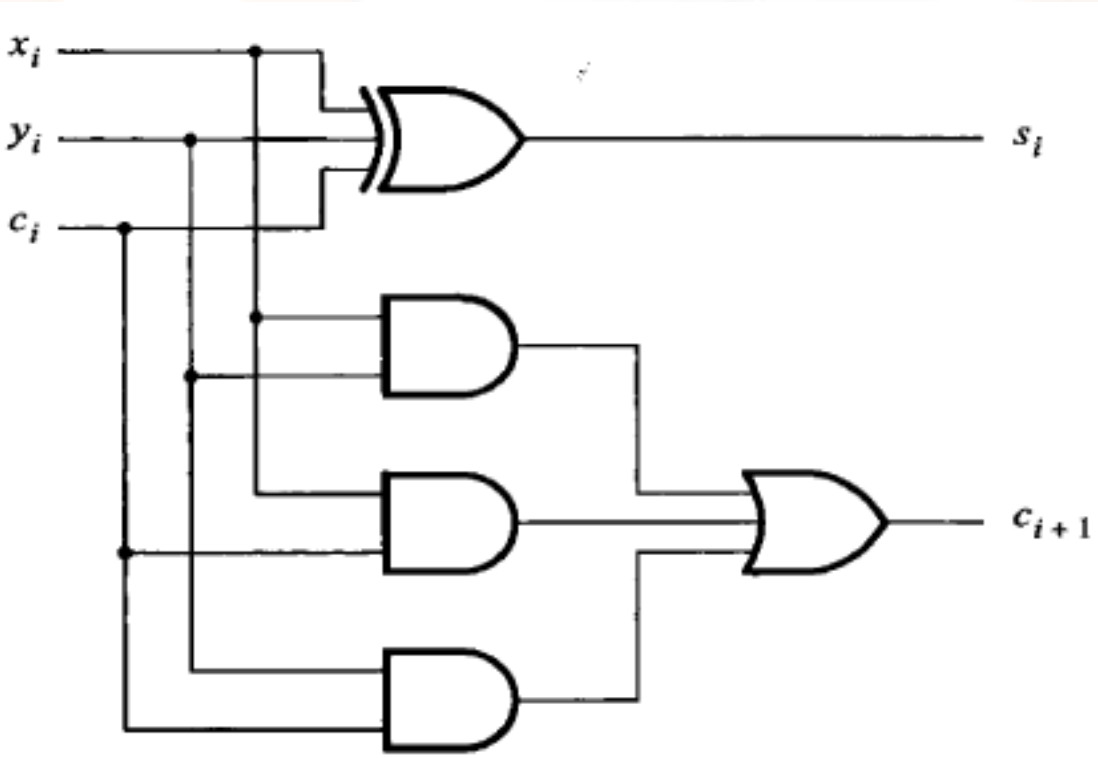
c_i \ x_iy_i	00	01	11	10
0		1		1
1	1		1	

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

c_i \ x_iy_i	00	01	11	10
0			1	
1		1	1	1

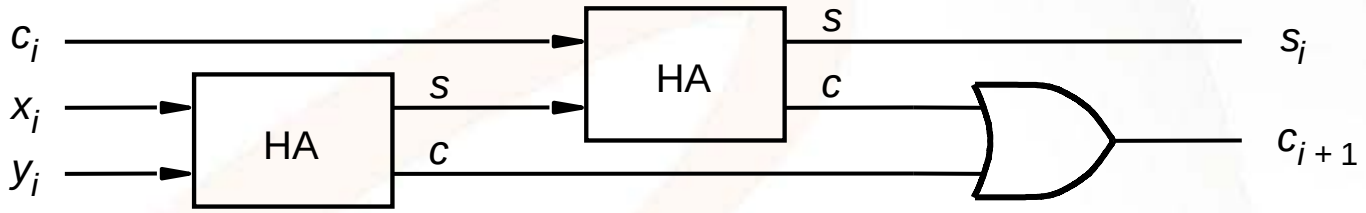
$$c_{i+1} = x_iy_i + x_ic_i + y_ic_i$$

(b) Karnaugh maps

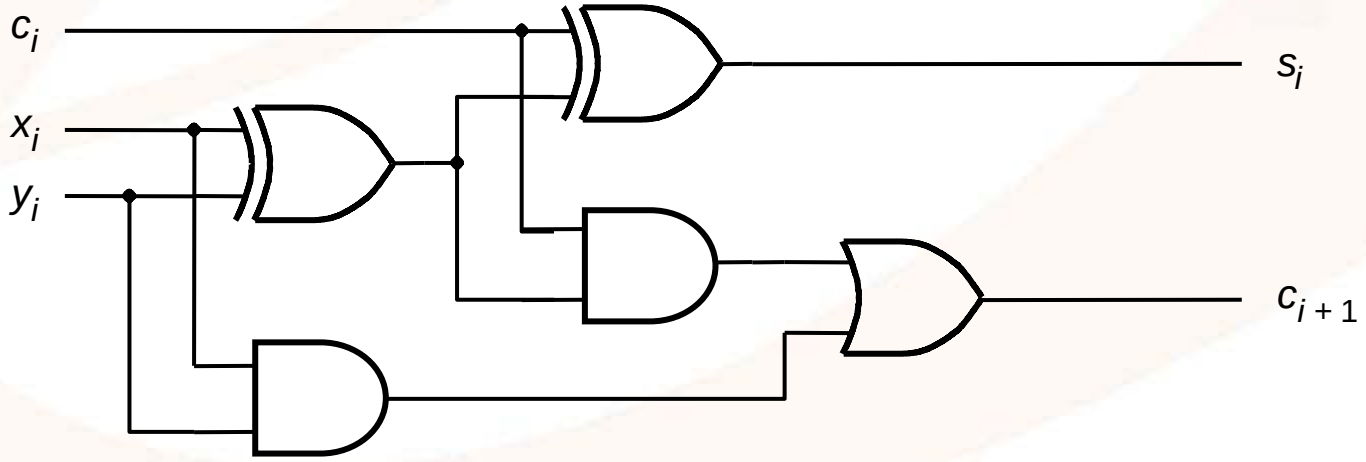


(c) Circuit

Note: Alternative Implementation of Adder



(a) Block diagram



(b) Detailed diagram

Figure 5.5. A decomposed implementation of the full-adder circuit.

Note:Ripple Carry Adder

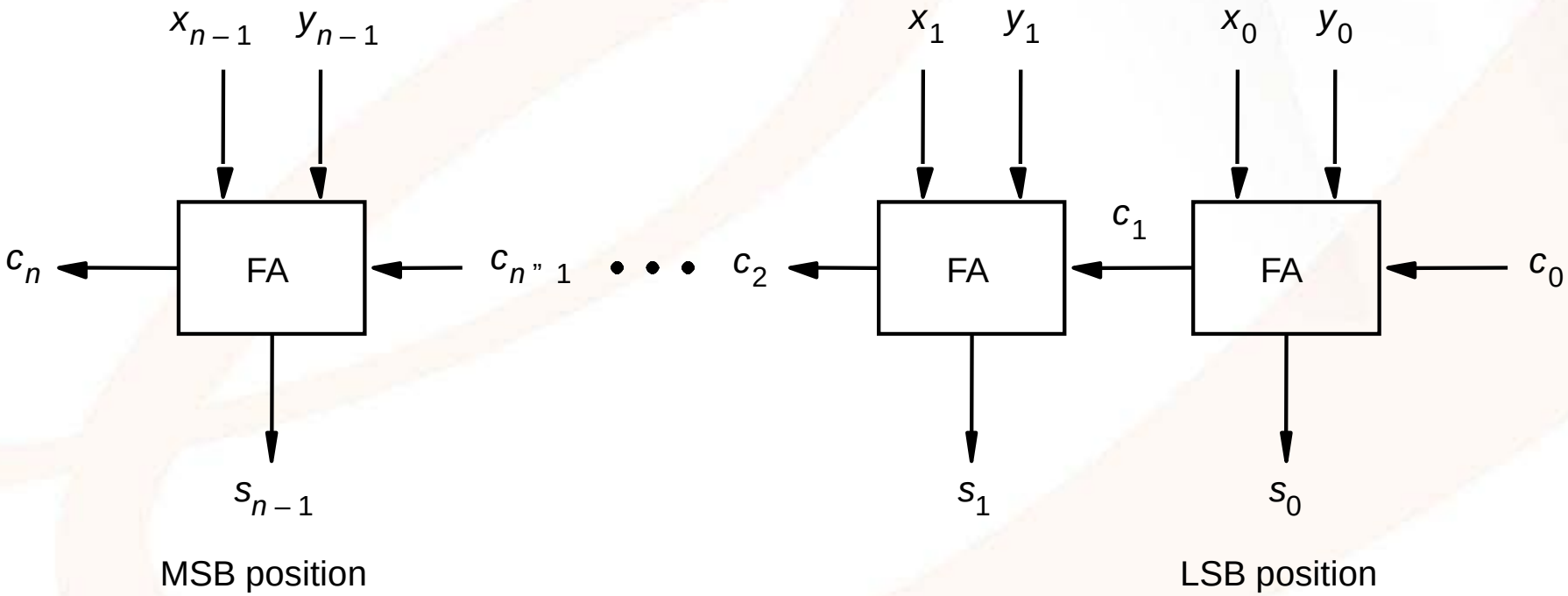
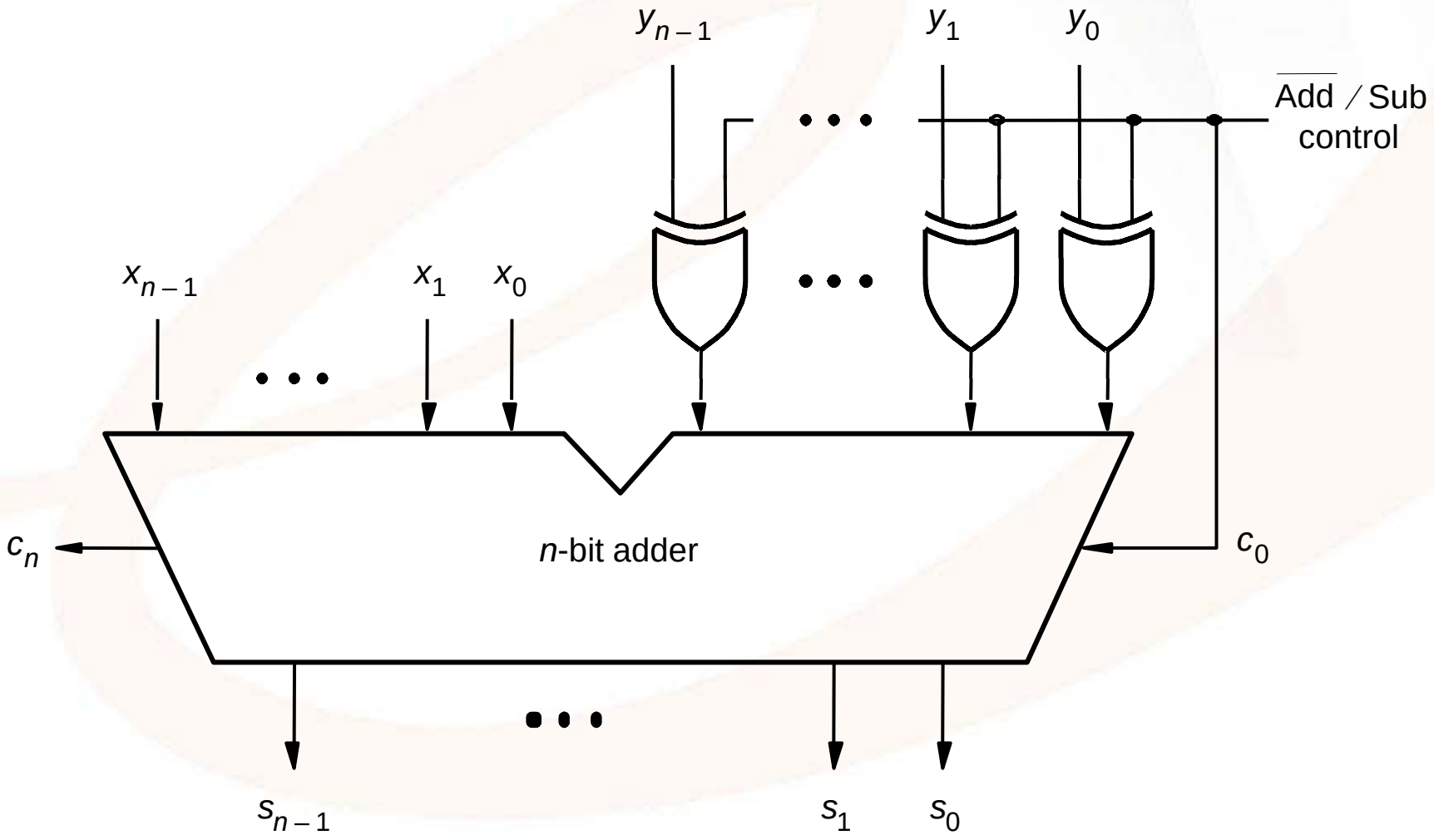


Figure 5.6. An *n*-bit ripple-carry adder.

Note:Adder/subtractor unit.



Carry-lookahead adder

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

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

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

Recall the carry-out function for stage I can be realized as

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

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

g_i **generate** p_i **propagate**

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

Carry-lookahead adder

- The function $g_i=1$ when both x_i and y_i are 1, regardless of the incoming carry c_i
  - Since in this case, stage *i* is guaranteed to generate a carry-out, *g* is called the ***generate*** function

$$g_i = x_i y_i$$

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

$$1 = 1 + d$$

generate

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

Carry-Out =1 if (g=1)

Carry-lookahead adder

- The function $p_i=1$ when either x_i and y_i are 1
  A carry-out is produced if $c_i=1$
  - The effect is that the carry-in of 1 is propagated through stage i; p is called the ***propagate*** function

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

$$p_i = x_i + y_i$$

$c_i = 1$

1 = d + 1

propagate

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

Carry-Out <= (Carry-In of 1) if (p=1)

Carry-lookahead adder

Why Generate and Propagate?

Why generate and propagate. If you look at the Boolean expressions for $\mathbf{p_i}$ and $\mathbf{g_i}$, you will see that they both use only $\mathbf{x_i}$ and $\mathbf{y_i}$. Neither depend on the carry. Since $\mathbf{x_i}$ and $\mathbf{y_i}$ are available immediately, this gives us hope that we can avoid waiting for carries.

Carry-lookahead adder

First, let's write the $\mathbf{c_1}$ which is the carry out for the adding bit 0 of **x** and **y**.

$$c_1 = g_0 + p_0 c_0$$

Now, we write it for $\mathbf{c_2}$.

$$c_2 = g_1 + p_1 c_1$$

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

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

Carry-lookahead adder

Let's go one more step further:

$$c_3 = g_2 + p_2c_2$$
$$= g_2 + p_2(g_1 + p_1g_0 + p_1p_0c_0)$$
$$= g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0$$

Already, you should be able to detect a pattern.
By following the same pattern, you'd expect:

$$\boldsymbol{c_4 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0c_0}$$

Carry-lookahead (CLA) adder

carry generated in stage n-2, and propagated through remaining stages

carry generated in stage 0, and propagated through remaining stages

$$c_n = g_{n-1} + p_{n-1}g_{n-2} + p_{n-1}p_{n-2}g_{n-3} + \ldots + p_{n-1}p_{n-2}\ldots p_1 g_0 + p_{n-1}p_{n-2}\ldots p_0 c_0$$

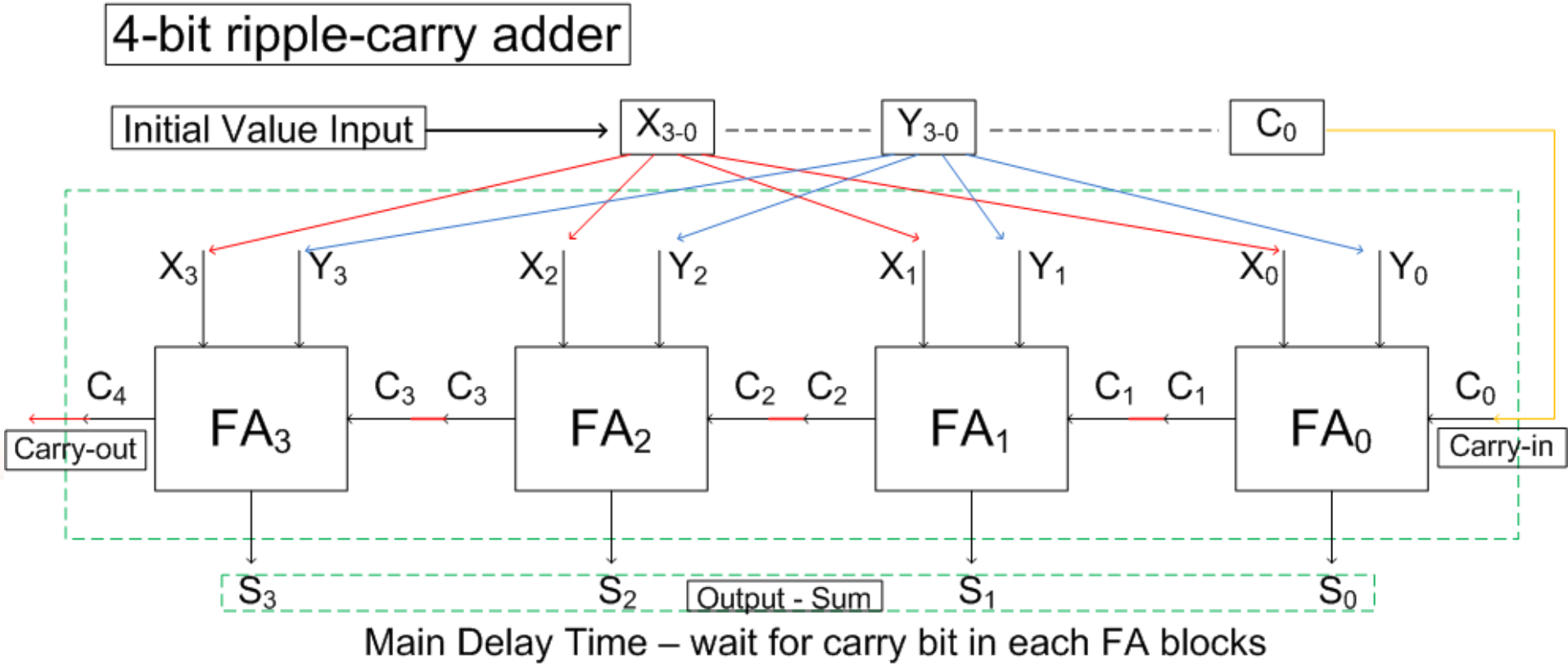
carry generated in last stage

carry generated in stage n-3, and propagated through remaining stages

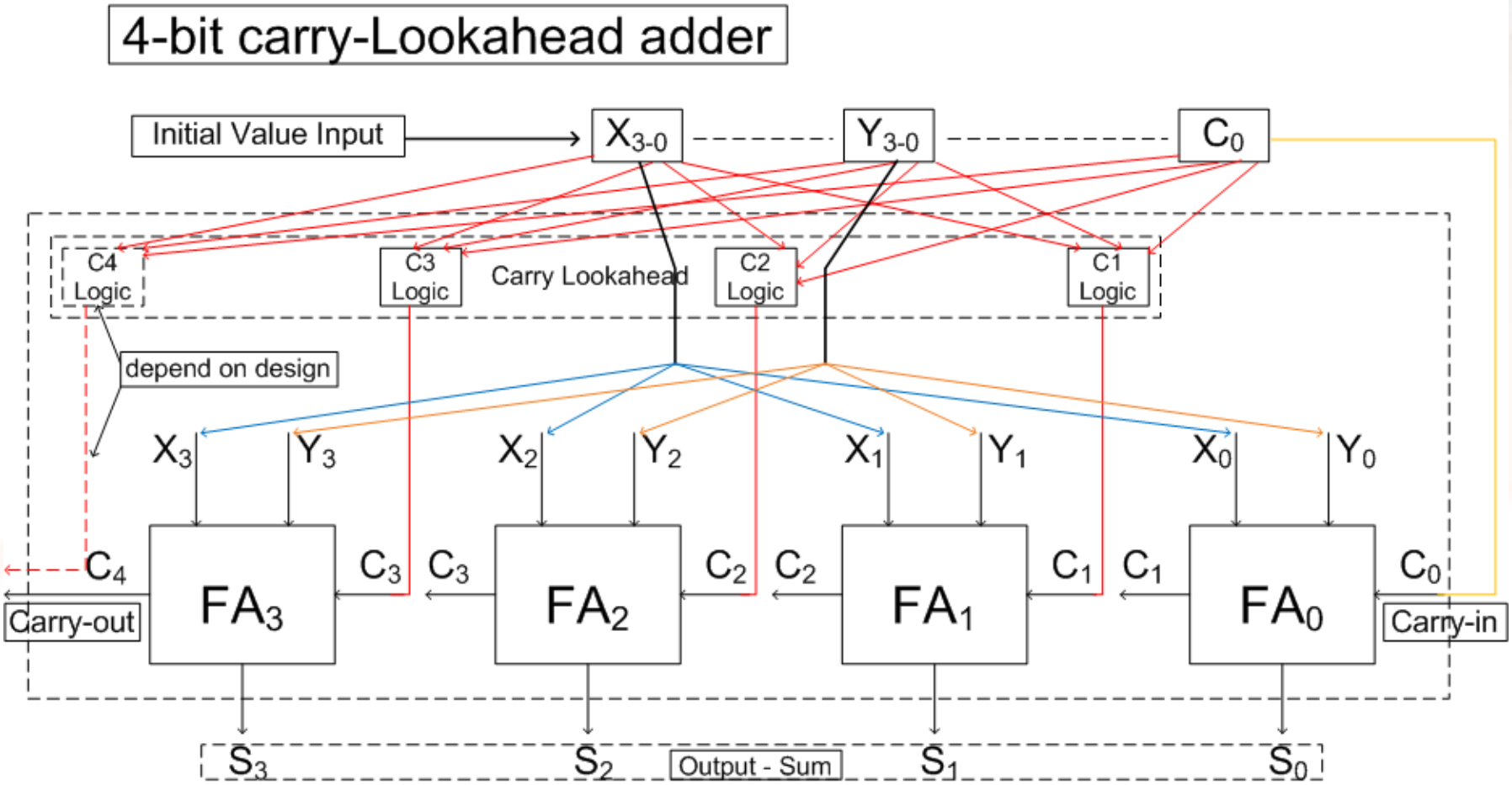
Input carry c_0 propagated through all stages

n-bit Adder

Ripple-Carry adder



Carry-lookahead (CLA) adder



Main Delay Time Decrease – We don't have to wait for carry bit in each FA blocks. But we're able to compute the carries in terms of xi , yi , and c0 .

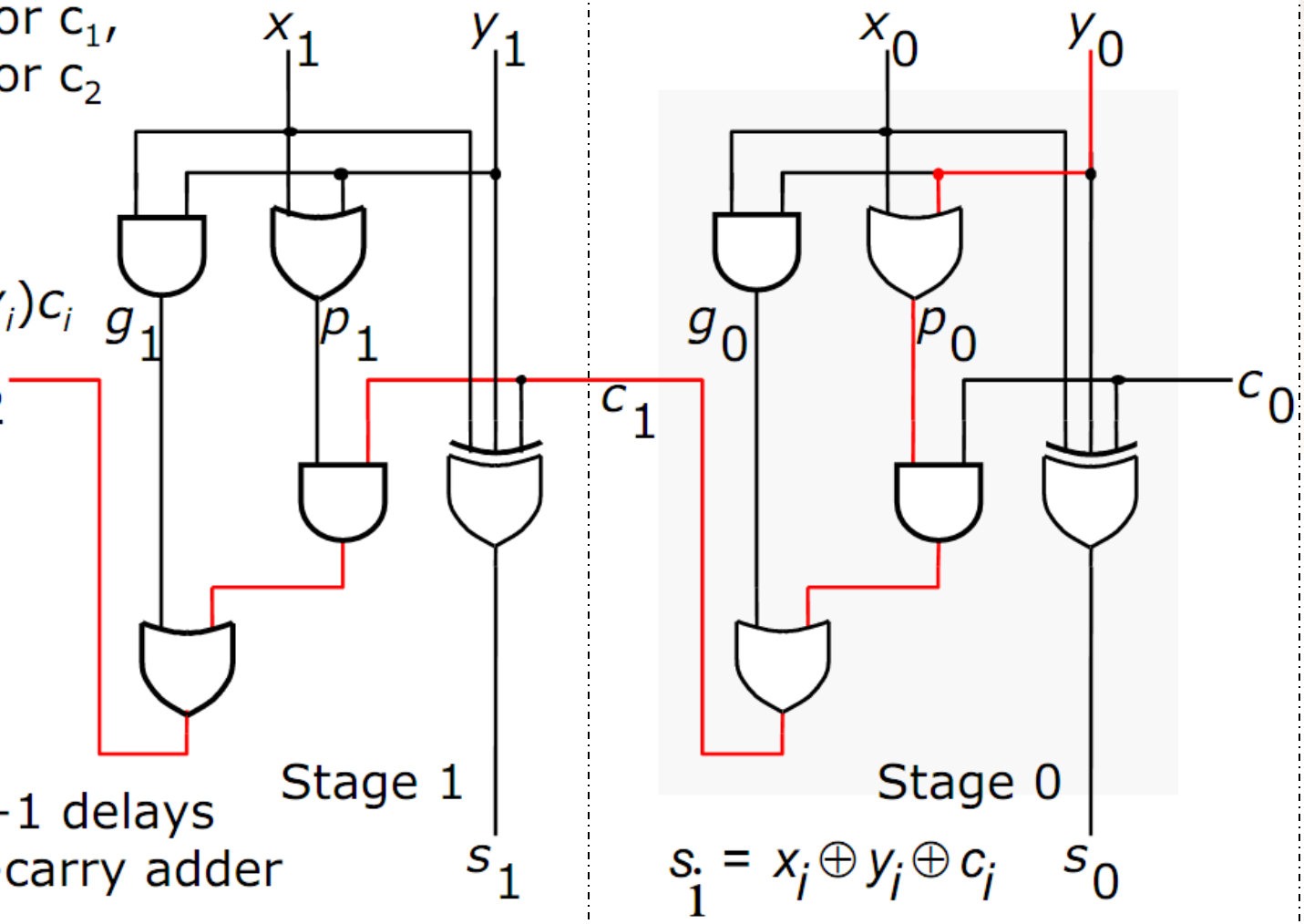
Ripple-carry adder critical path

3 gate delays for c_1,
5 gate delays for c_2

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

$g_i=x_iy_i$

$p_i=x_i+y_i$



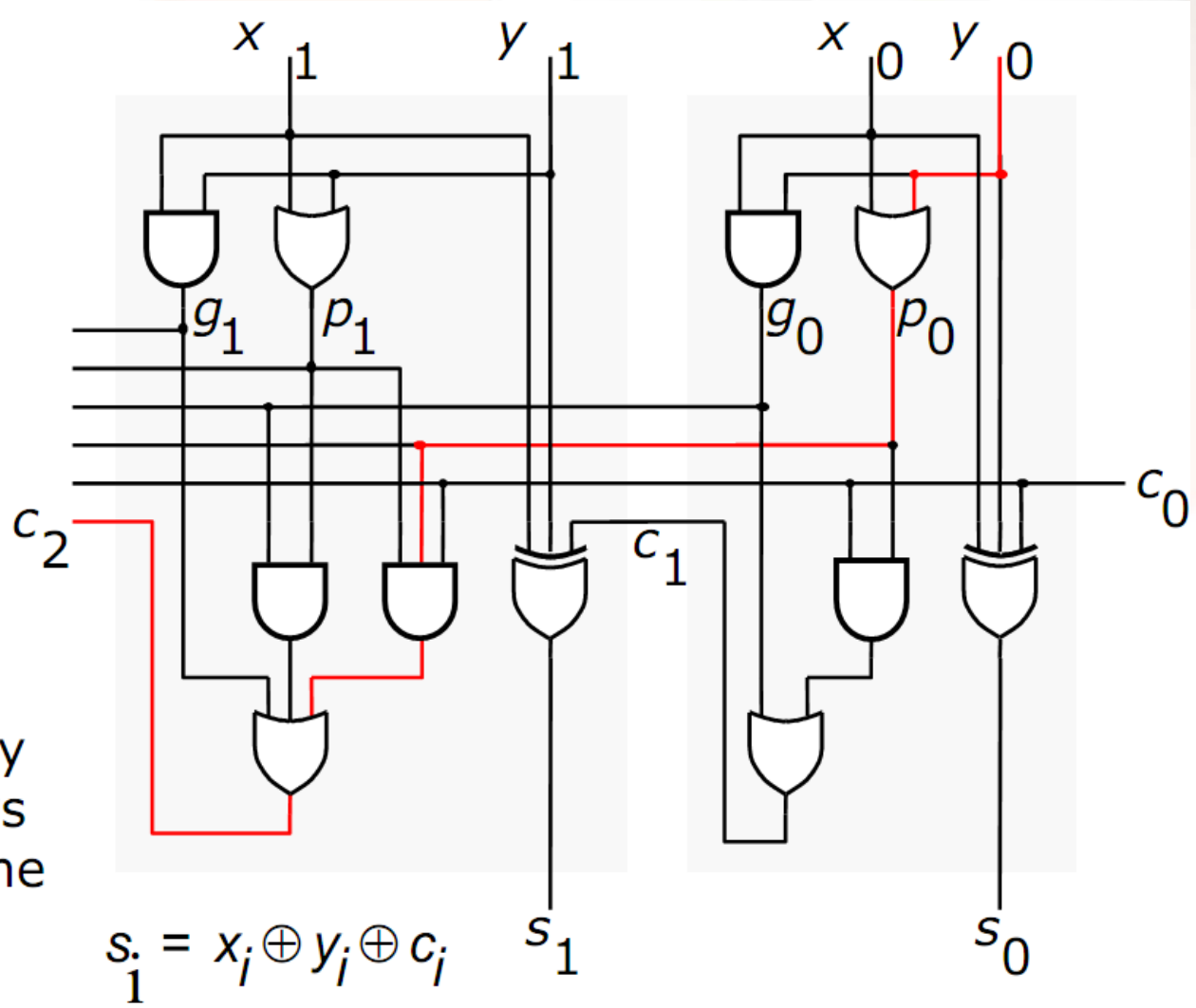
In general, $2n+1$ delays
for n-bit ripple-carry adder

Carry-lookahead critical path

3 gate delays for c_1,
3 gate delays for c_2
3 gate delays for c_n

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

Total delay for n-bit CLA adder is 4 gate delays
All g_i and p_i, one delay
All c_i, two more delays
One more delay for the sums s_i



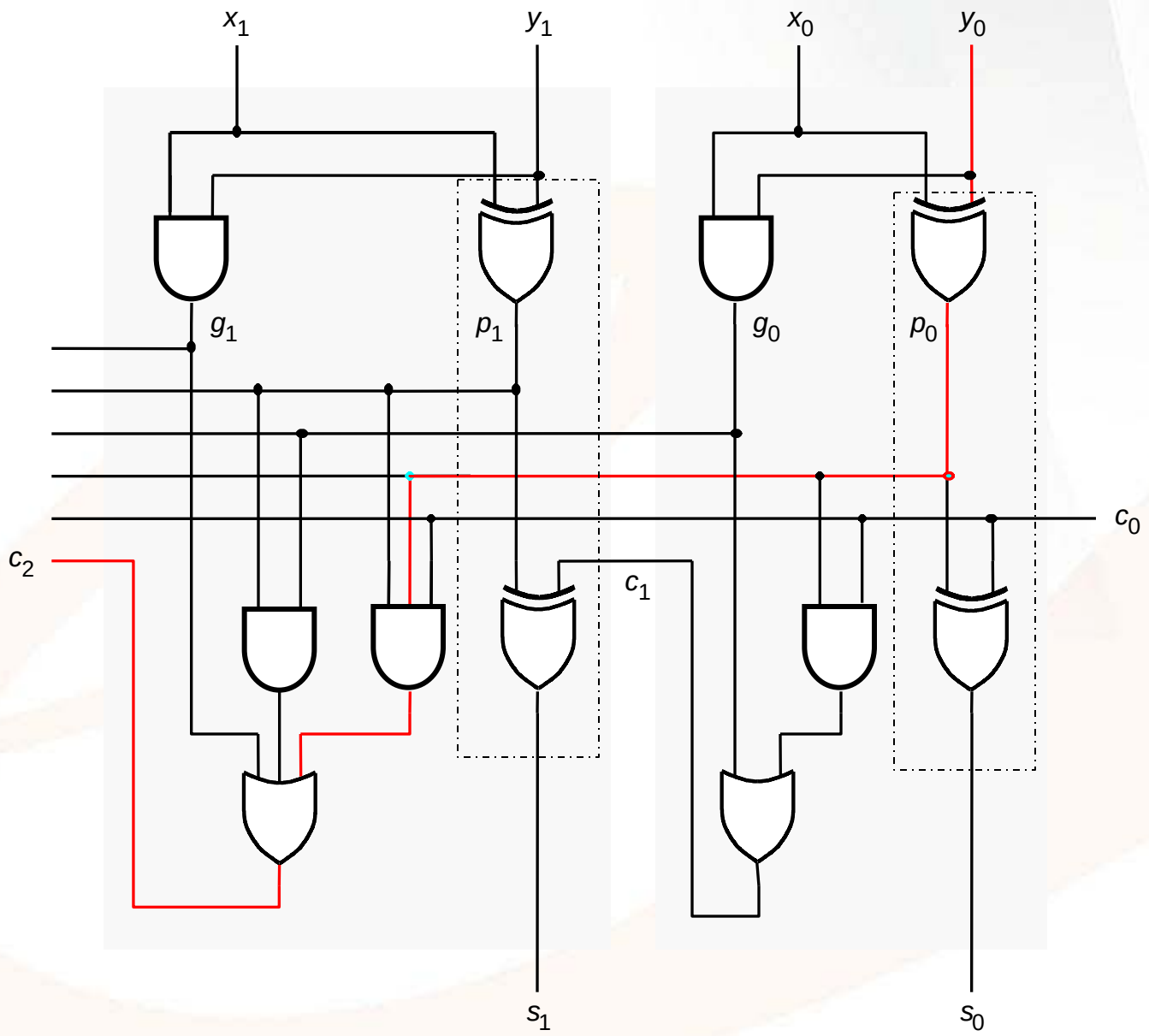


Figure 5.19. An alternative design for a carry-lookahead adder.

Carry-lookahead limitations

- The expression for carry in a CLA adder

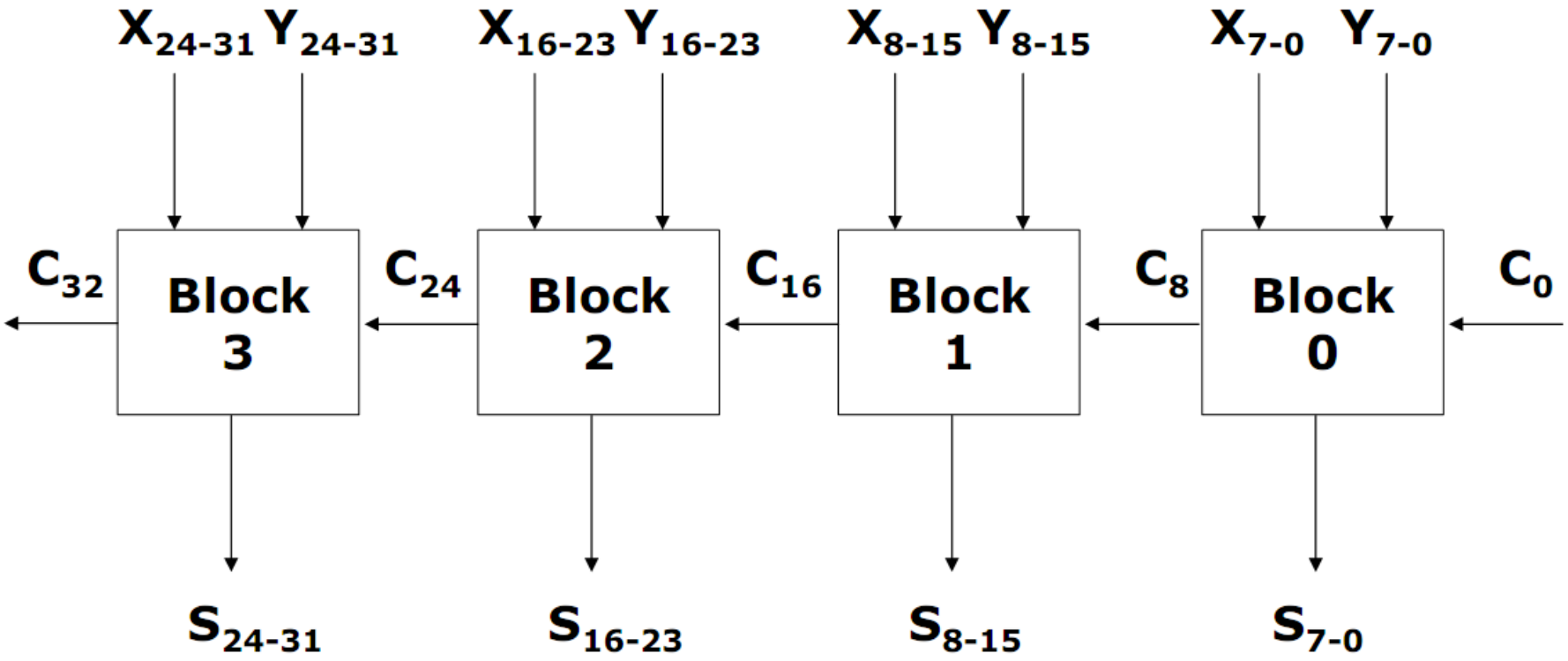
  $c_n = g_{n-1} + p_{n-1}g_{n-2} + p_{n-1}p_{n-2}g_{n-3} + \ldots + p_{n-1}p_{n-2}\ldots p_1 g_0 + p_{n-1}p_{n-2}\ldots p_0 c_0$

- obviously results in a fast solution (since it is only a 2 level AND-OR function)
- Fan-in limitations may effectively limit the speed of a CLA adder
  - Devices with known fan-in limitations (such as an FPGA) often include dedicated circuitry for implementation of fast adders
- The complexity of an *n*-bit CLA adder increases rapidly as *n* becomes large
  - To reduce this complexity, we can use a *hierarchical* approach in designing large adders

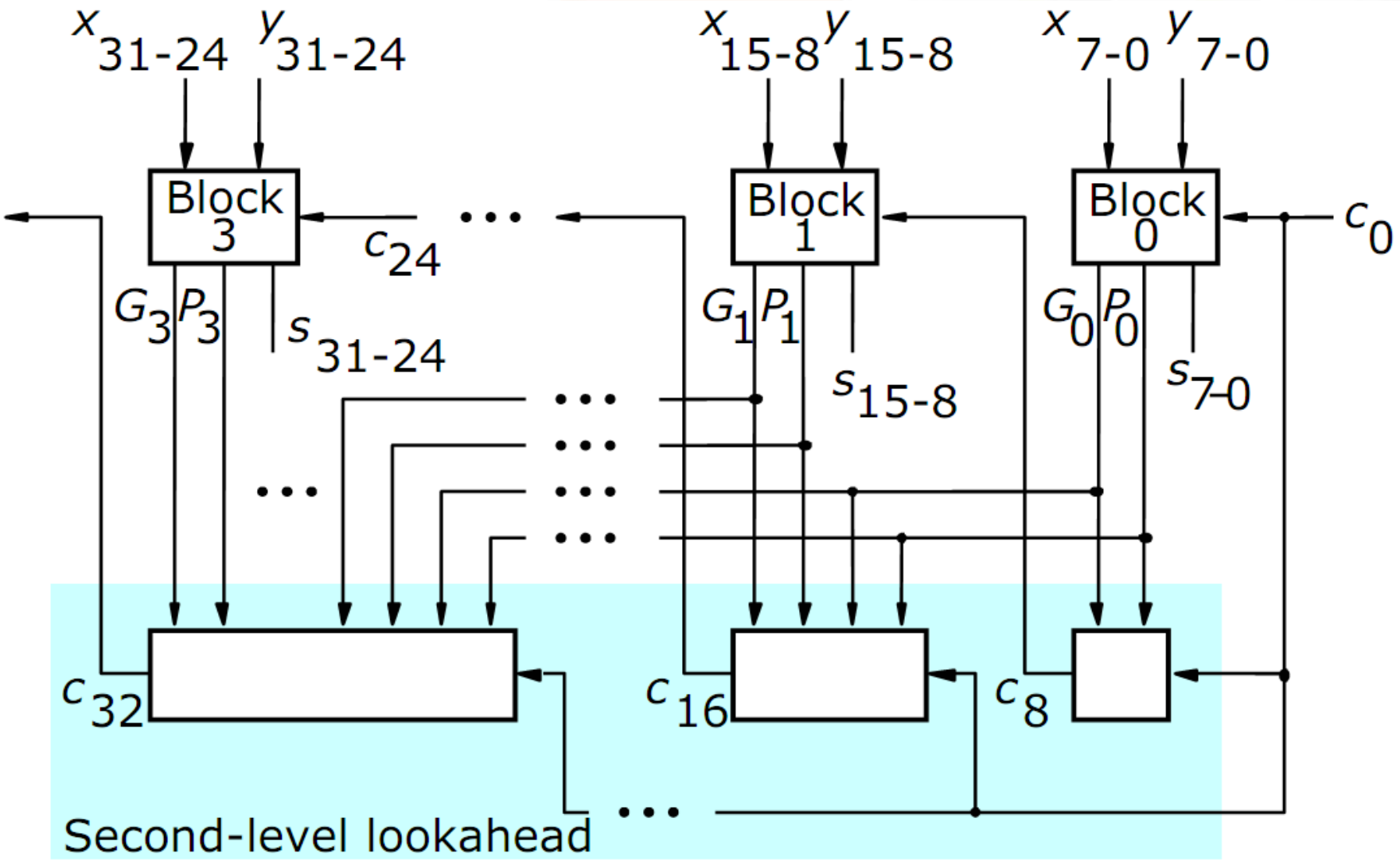
32-bit adder design

- Suppose we want to design a 32-bit adder
- Divide this adder into 4 blocks such that
  - *Bits $b_{7\text{-}0}$ are block 0*
  - *Bits $b_{15\text{-}8}$ are block 1*
  - *Bits $b_{23\text{-}16}$ are block 2*
  - *Bits $b_{31\text{-}24}$ are block 3*
- Each block can be constructed as an 8-bit CLA adder
  - *The carry-out signals from the four blocks are c_8, c_{16}, c_{24}, and c_{32}*
- There are 2 basic approaches for interconnecting these four blocks
  - *Ripple-carry between blocks*
  - *Second level carry-lookahead circuit*

Ripple-carry between blocks



Second level carry-lookahead circuit



Second level carry-lookahead circuit

- For the second level circuit:

$$P_0 = p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0$$

$$G_0 = g_7 + p_7 g_6 + p_7 p_6 g_5 + \ldots + p_7 p_6 p_5 p_4 p_3 p_2 p_1 g_0$$

$$c_8 = G_0 + P_0 c_0$$

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

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

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

Hierarchical CLA analysis

• Assuming a fan-in constraint of four inputs, the time to add two 32-bit numbers involves

G_j, and P_j require three gate delays, C_8, C_{16}, and C_{24} are available after ***five gate delays***. The time needed to add two 32-bit numbers involves these five gate delays plus ***two more*** to produce the internal carries in blocks 1, 2, and 3, plus ***one more gate delay*** (XOR) to generate each sum bit. ***This gives a total of eight gate delays***.