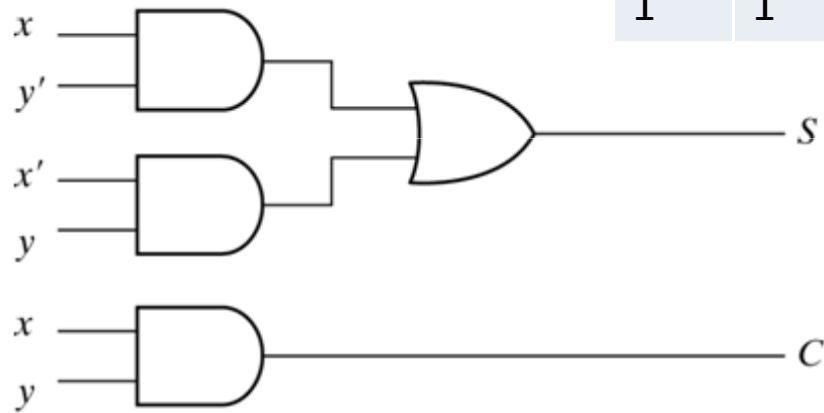


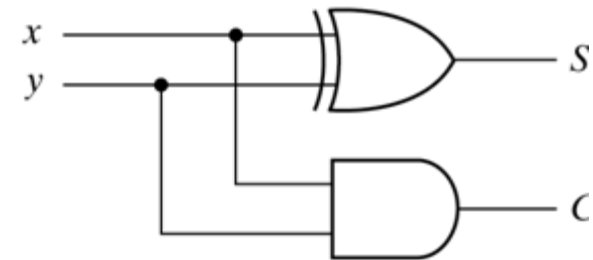
Adder

Half Adder

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



(a) $S = xy' + x'y$
 $C = xy$



(b) $S = x \oplus y$
 $C = xy$

Full Adder

- Truth table:

X	Y	Z	C	S
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

Note:

Z - carry in (to the current position)

C - carry out (to the next position)

$$S = \sum m(1,2,4,7)$$

$$C = \sum m(3,5,6,7)$$

- Using K-map, simplified SOP form is:

$$C = XY + XZ + YZ$$

$$S = X'Y'Z + X'YZ' + XY'Z' + XYZ$$

Sum

YZ \ X	0	1
	0	1
00	0 ₀	1 ₄
01	1 ₁	0 ₅
11	0 ₃	1 ₇
10	1 ₂	0 ₆

Carry

YZ \ X	0	1
	0	1
00	0 ₀	0 ₄
01	0 ₁	1 ₅
11	1 ₃	1 ₇
10	0 ₂	1 ₆

Z

Full Adder Circuit

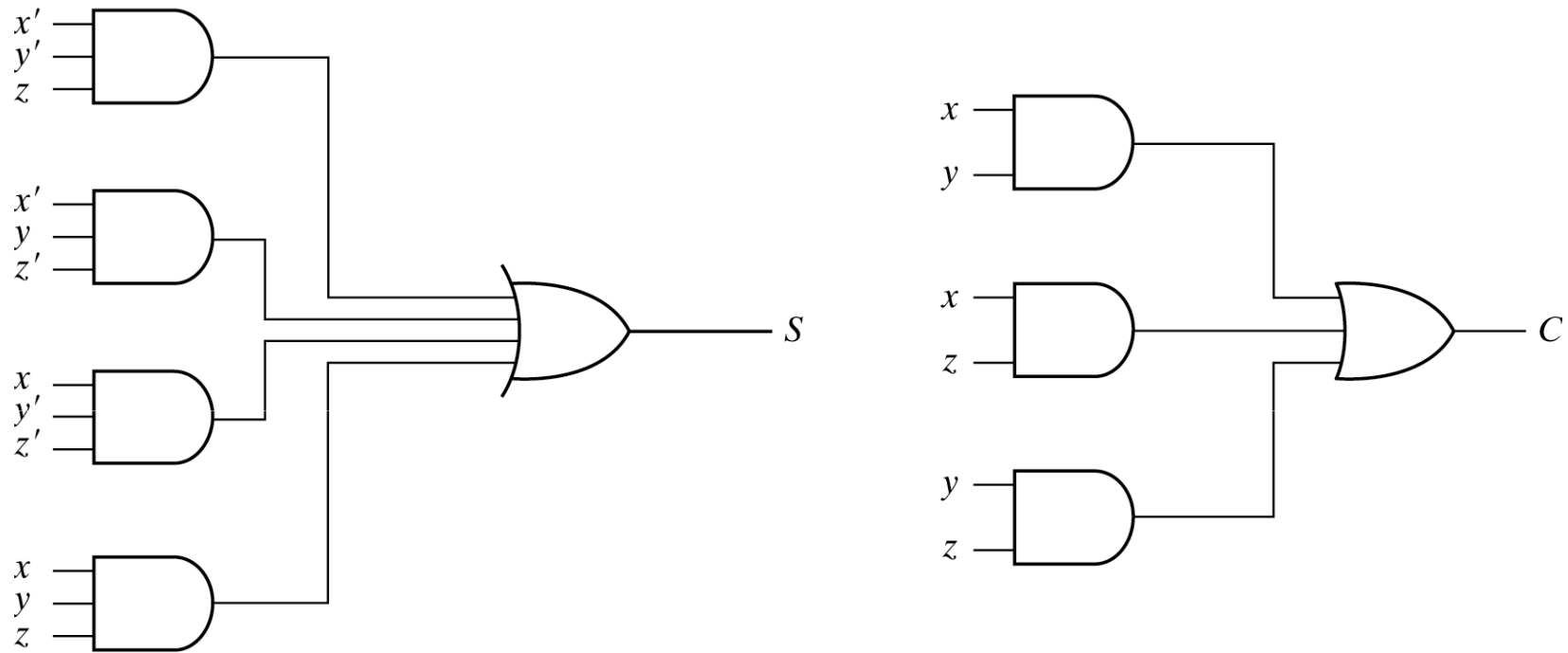


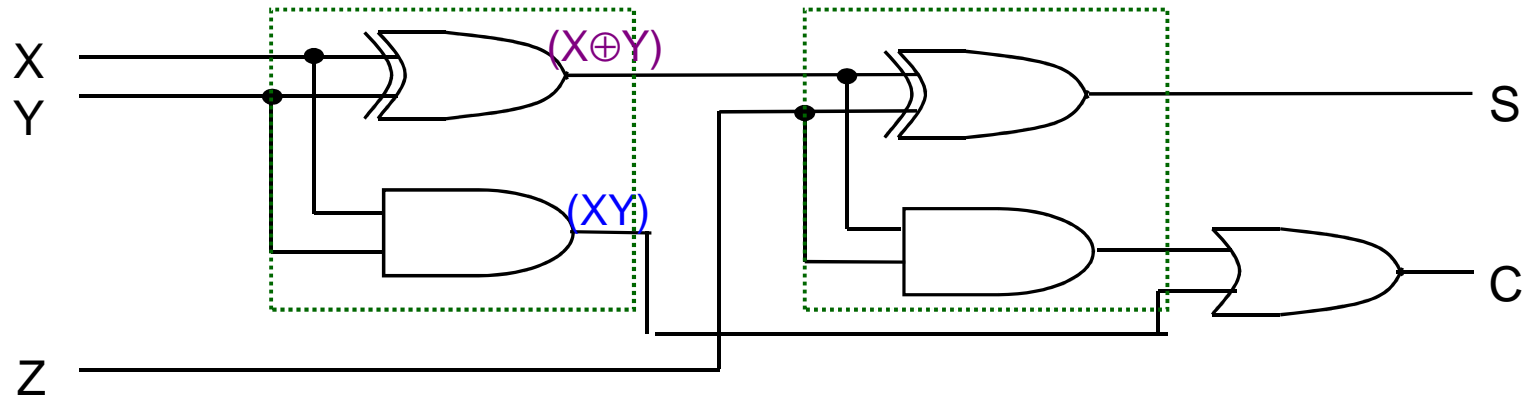
Fig. 4-7 Implementation of Full Adder in Sum of Products

Gate-level Design: Full Adder

- Circuit for above formulae:

$$C = XY + (X \oplus Y)Z$$

$$S = X \oplus Y \oplus Z$$



Full Adder made from two Half-Adders (+ OR gate).

4-bit Adder Circuit

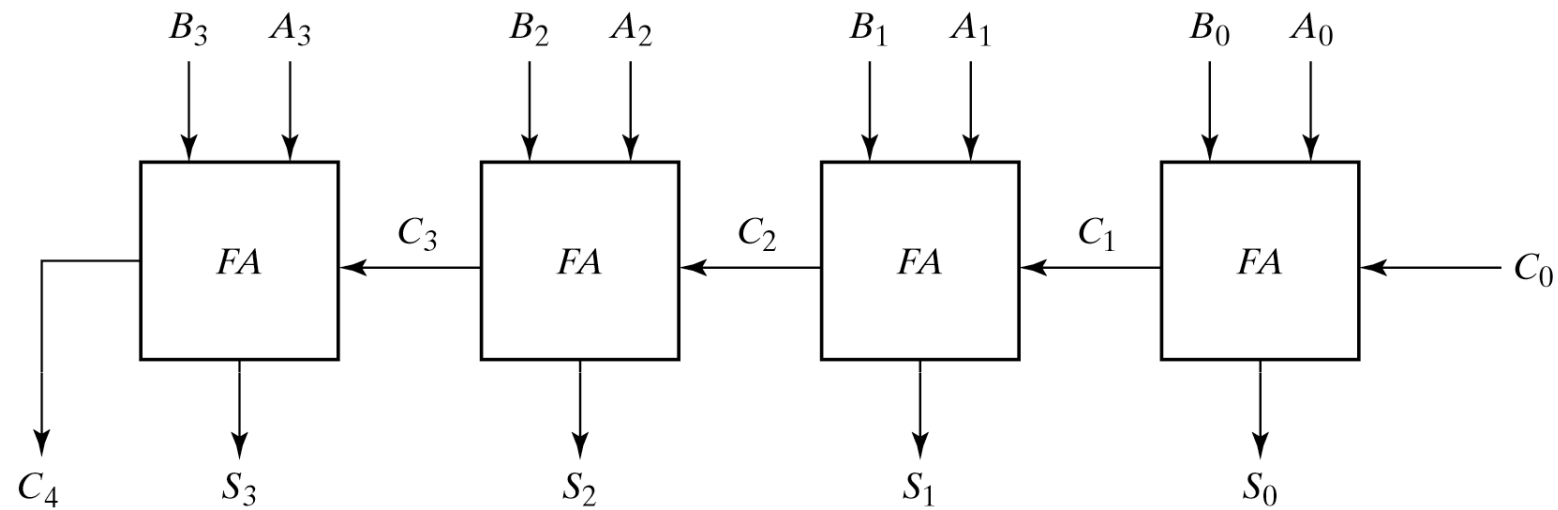
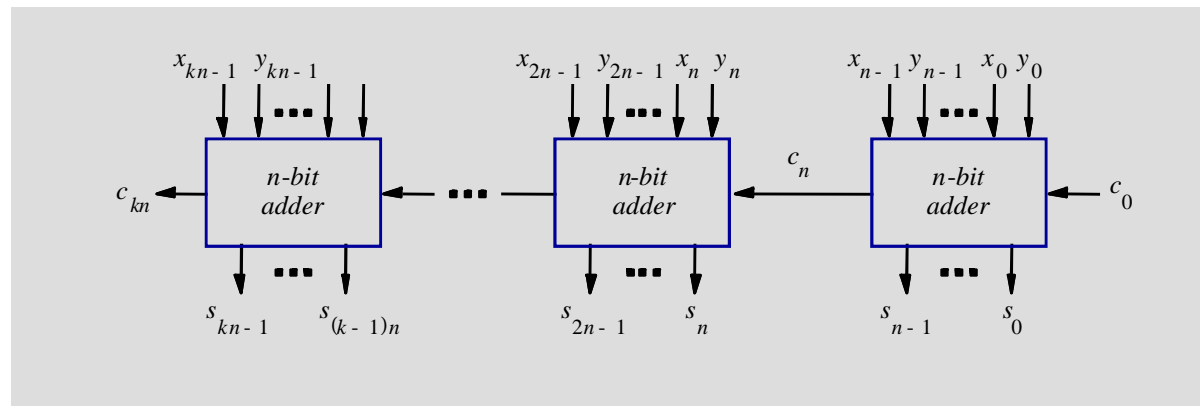


Fig. 4-9 4-Bit Adder

But this is slow...

K n -bit adder

K n -bit numbers can be added by cascading k n -bit adders.



Each n -bit adder forms a block, so this is cascading of blocks.

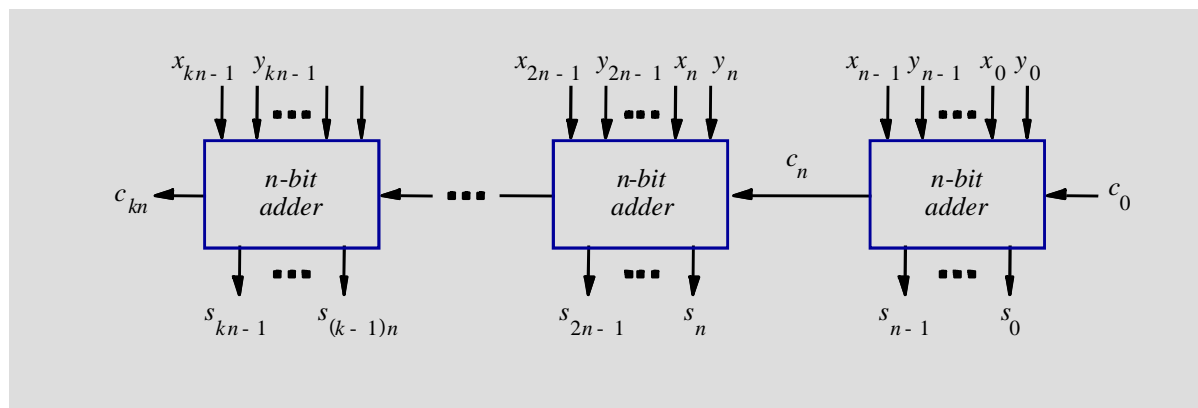
Carries ripple or propagate through blocks, [Blocked Ripple Carry Adder](#)

N bit Ripple carry adder

- Straight-forward design
- Simple circuit structure
- Easy to understand
- Most power efficient
- Slowest (too long critical path)

Delays in the ripple carry adder

- This is called a **ripple carry** adder, because the inputs A_0 , B_0 and C_i “ripple” leftwards until C_o and S_3 are produced.
- Ripple carry adders are slow!
 - For an n -bit ripple carry adder
 - Carry takes $2n$ gate delays
 - Sum takes $(2n-1)$ gate delays
 - Imagine a 64-bit adder. The longest path would have 128 gates!

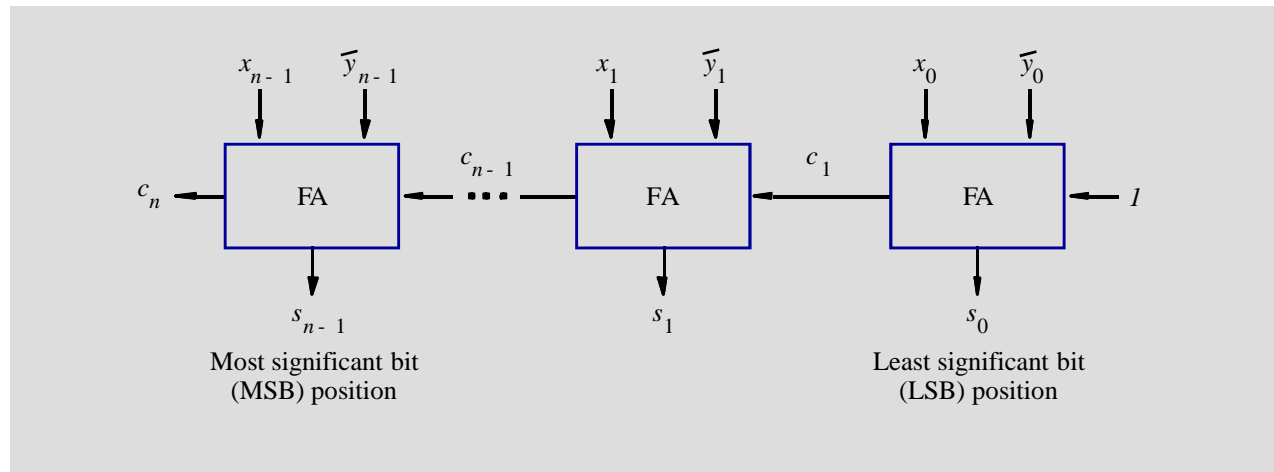


Binary Subtractor

- Subtraction is done by using complements
- A's 2's Complement = $A' + 1$
- $A - B = A + B' + 1$

n -bit subtractor

- Recall $X - Y$ is equivalent to adding 2's complement of Y to X .
- 2's complement is equivalent to 1's complement + 1.
- $X - Y = X + \overline{Y} + 1$
- 2's complement of positive and negative numbers is computed similarly.



4-bit adder subtractor

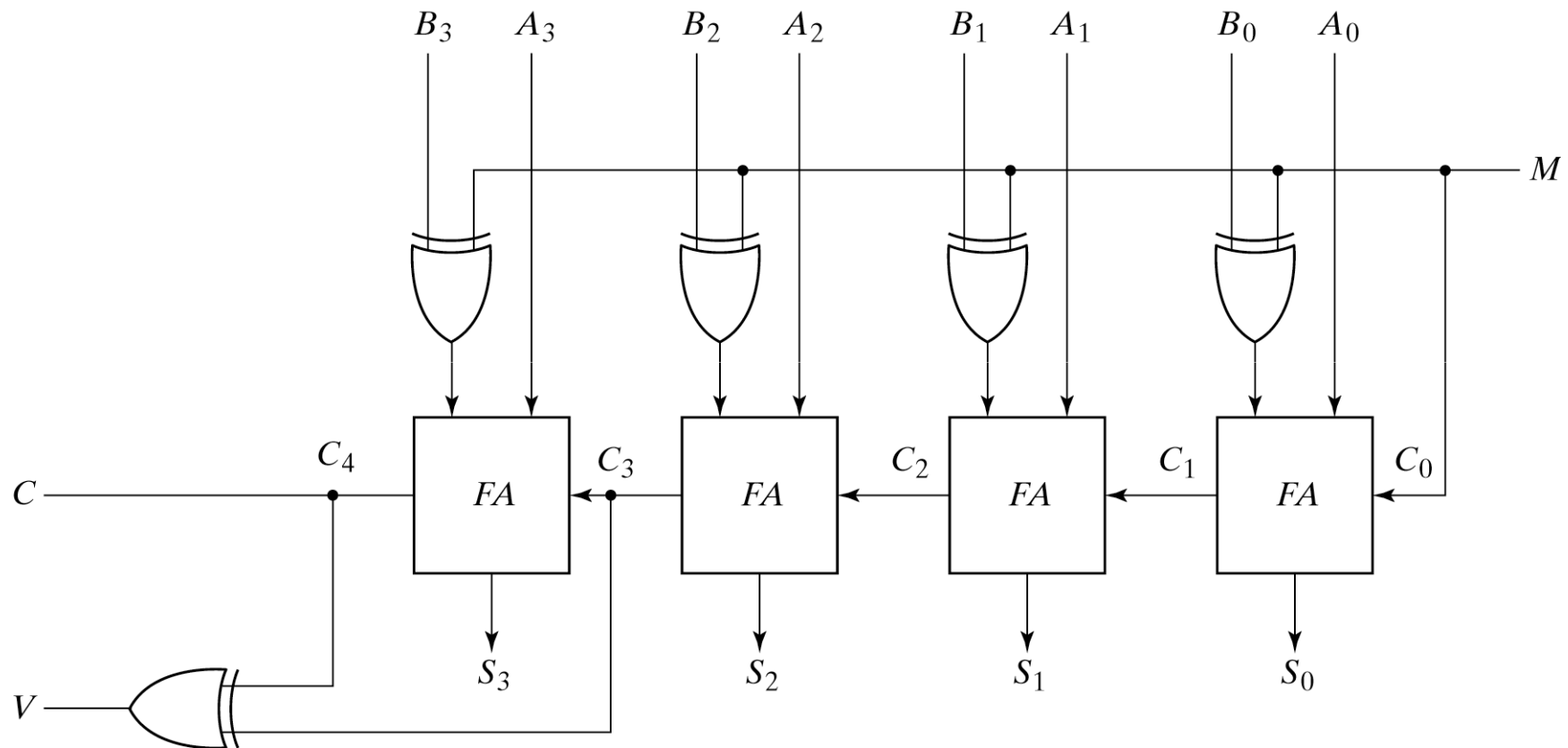


Fig. 4-13 4-Bit Adder Subtractor

Carry Lookahead adder

A faster way to compute carry outs

- Instead of waiting for the carry out from all the previous stages, we could compute it directly with a two-level circuit, thus minimizing the delay.
- First we define two functions.
 - The “generate” function g_i produces 1 when there *must* be a carry out from position i (i.e., when A_i and B_i are both 1).

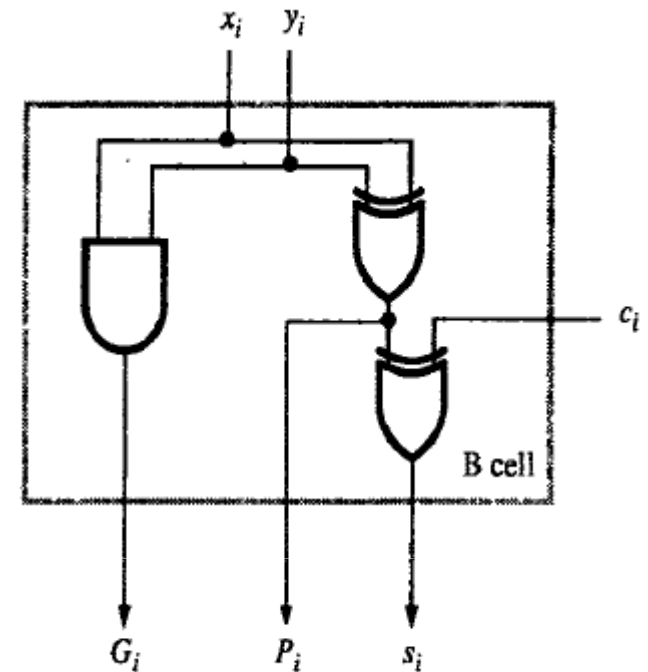
$$g_i = A_i B_i$$

- The “propagate” function p_i is true when, if there is an incoming carry, it is propagated (i.e., when $A_i=1$ or $B_i=1$, but not both).

$$p_i = A_i \oplus B_i$$

- Then we can rewrite the carry out function:

$$C_{i+1} = g_i + p_i C_i$$



A_i	B_i	C_i	C_{i+1}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A Note On Propagation

- We could have defined propagation as $A + B$ instead of $A \oplus B$
 - As defined, it captures the case when we propagate but don't generate
 - I.e., propagation and generation are mutually exclusive
 - There is no reason that they need to be mutually exclusive
 - However, if we use \oplus to define propagation, then we can share the XOR gate between the production of the sum bit and the production of the propagation bit

Algebraic carry out

- Let's look at the carry out equations for specific bits, using the general equation from the previous page $c_{i+1} = g_i + p_i c_i$:

$$c_1 = g_0 + p_0 c_0$$

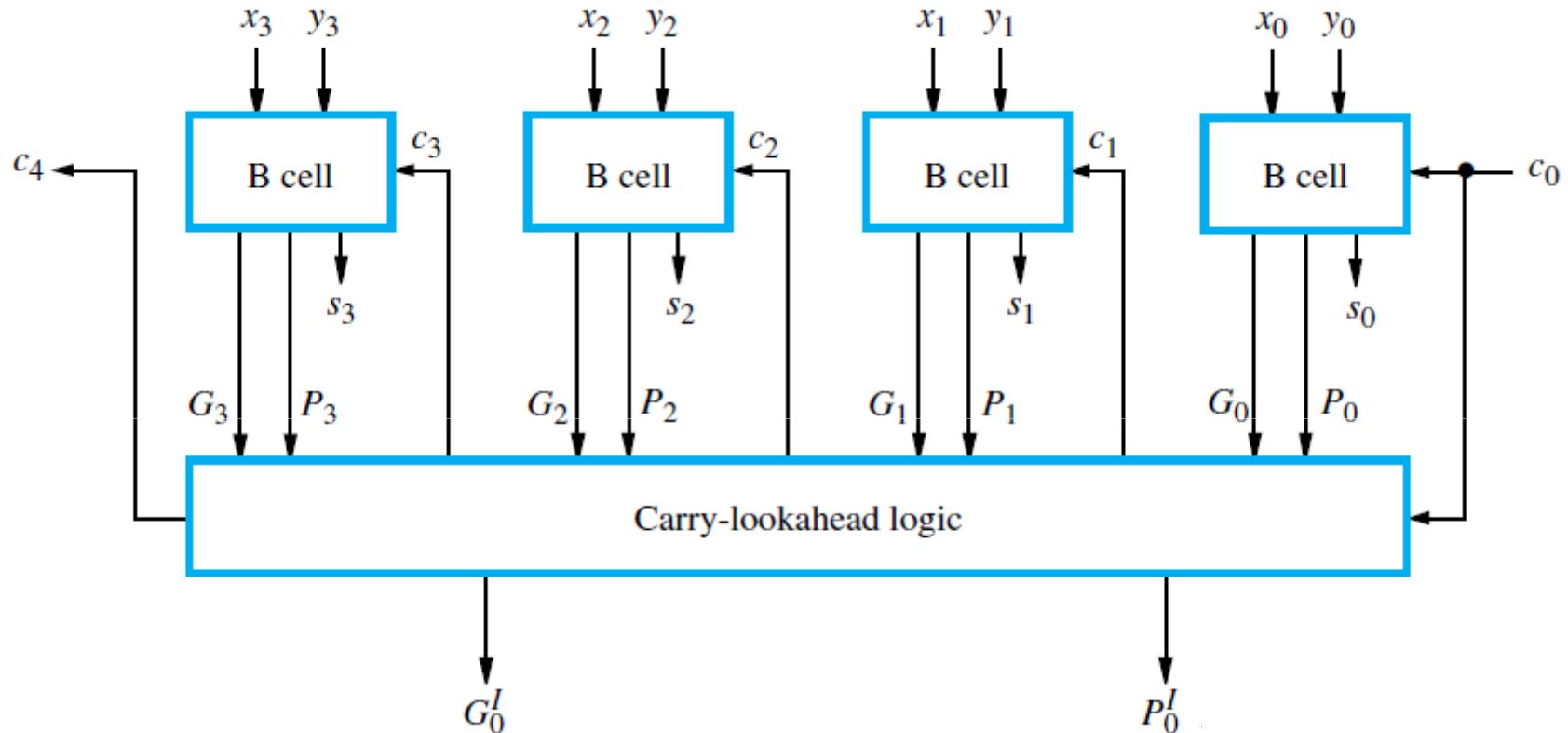
$$\begin{aligned} c_2 &= g_1 + p_1 c_1 \\ &= g_1 + p_1 (g_0 + p_0 c_0) \\ &= g_1 + p_1 g_0 + p_1 p_0 c_0 \end{aligned}$$

$$\begin{aligned} c_3 &= g_2 + p_2 c_2 \\ &= g_2 + p_2 (g_1 + p_1 g_0 + p_1 p_0 c_0) \\ &= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \end{aligned}$$

$$\begin{aligned} c_4 &= g_3 + p_3 c_3 \\ &= g_3 + p_3 (g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0) \\ &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0 \end{aligned}$$

- These expressions are all sums of products, so we can use them to make a circuit with only a two-level delay.

4 bit Carry lookahead adder

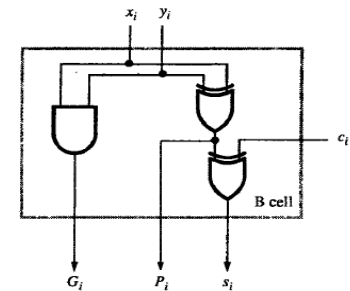


Gate Delays 4 bit CLA

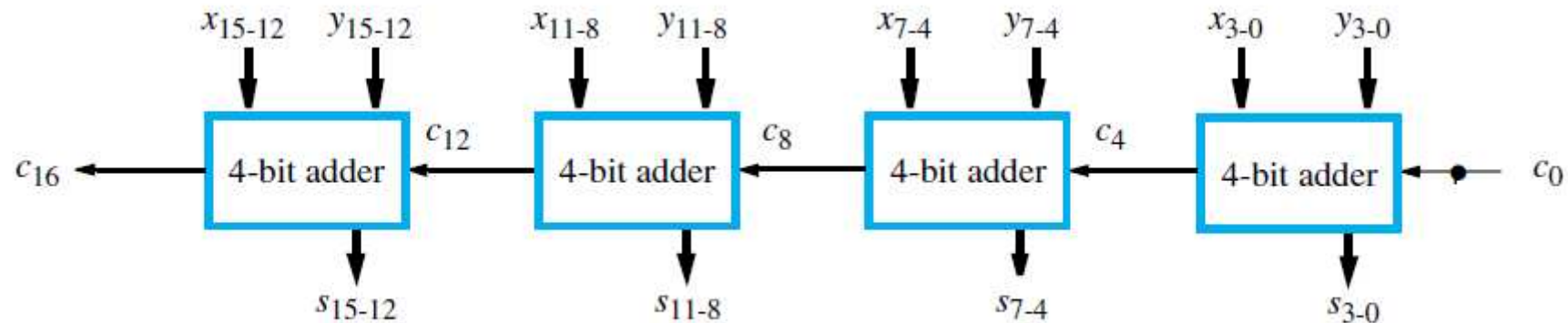
$G_i, P_i = 1$ gate delay
 $C_i = 3^{\text{rd}}$ gate delay
 $S_i = 4^{\text{th}}$ gate delay

Gate Delays 4 bit ripple carry

$C_i = 8$ gate delay
 $S_i = 7$ gate delay



16 bit ripple carry adder using 4 bit Carry lookahead adder



Gate Delays for 16 bit

1. Ripple Carry Adder

Sum 31

Carry 32

2. CLA

$G, P = 1^{\text{st}}$ gate delay

$C_4 = 3^{\text{rd}}$ gate delay,

$C_8 = 5^{\text{th}}$ gate delay

$C_{12} = 7^{\text{th}}$ gate delay

$C_{16} = 9^{\text{th}}$ gate delay

$S_{15} = 10^{\text{th}}$ gate delay

Gate Delays for 32 bit

1. Ripple Carry Adder

Sum 63

Carry 64

2. CLA

$G, P = 1^{\text{st}}$ gate delay

$C_4 = 3^{\text{rd}}$ gate delay,

$C_8 = 5^{\text{th}}$ gate delay

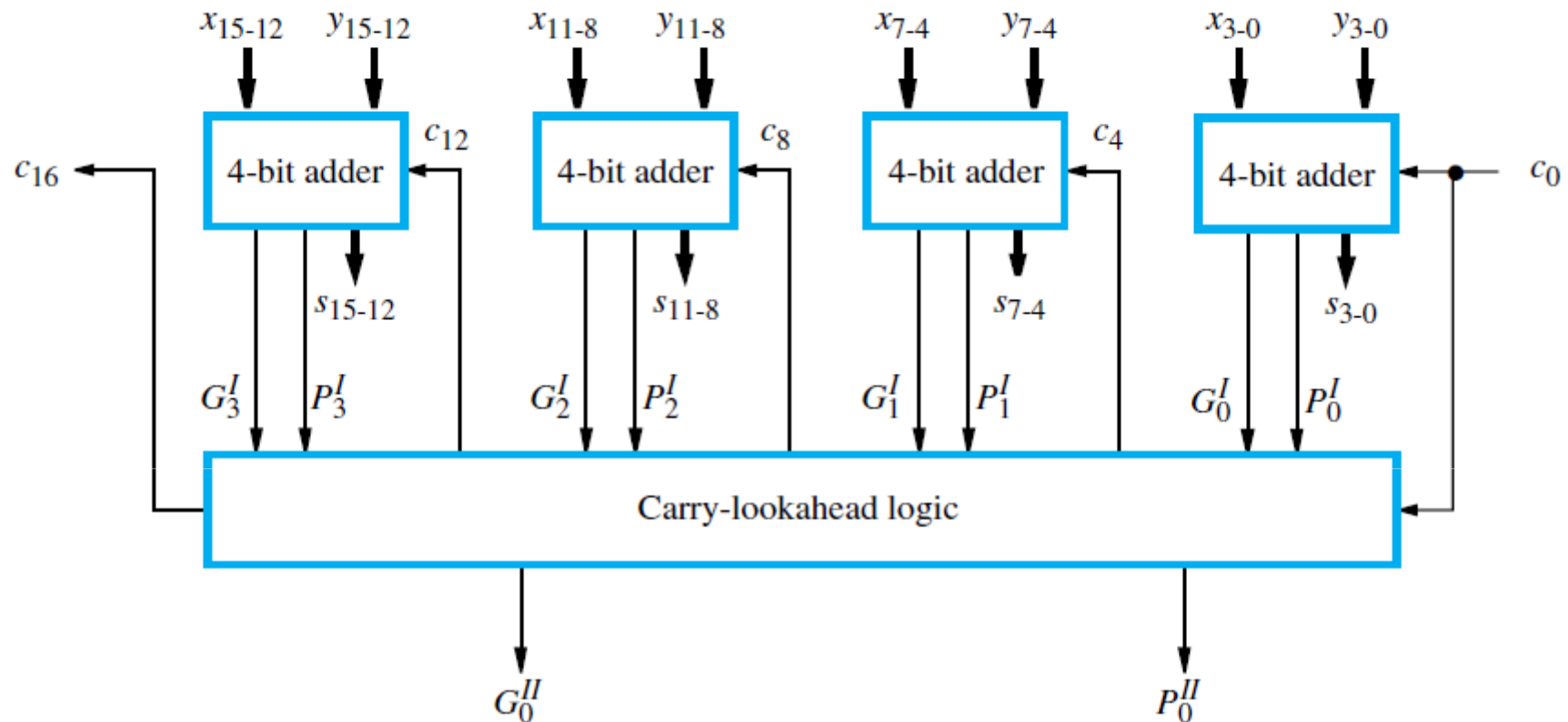
$C_{12} = 7^{\text{th}}$ gate delay

$C_{32} = 17^{\text{th}}$ gate delay

(1 for g,p + 2*8 for each 4 bit adder)

$S_{32} = 18^{\text{th}}$ gate delay

High level Generator and Propagator function



4 CLA forms 16bit adder

Gate delays

$G, P = 1$

$G^I, P^I = 3^{\text{rd}}$ gate delay

$C_4, C_8, C_{12}, C_{16} = 5^{\text{th}}$ gate delay

C_{12} to $C_{15} = 7^{\text{th}}$ gate delay

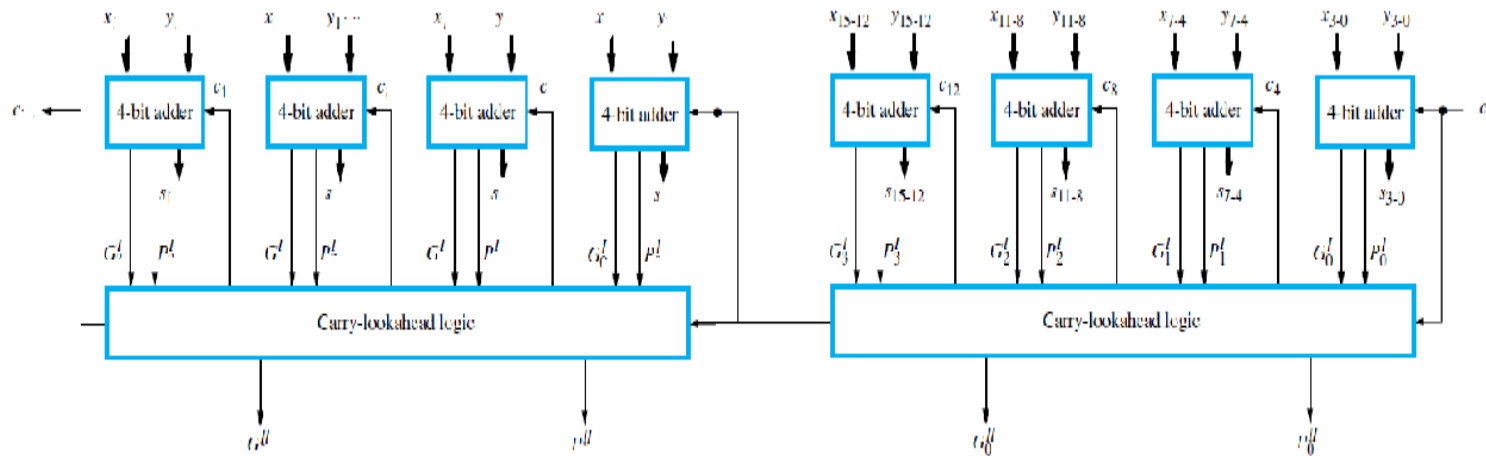
$S_{15} = 8^{\text{th}}$ gate delay

$$P_0^I = P_3 P_2 P_1 P_0$$

$$G_0^I = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$$

$$c_{16} = G_3^I + P_3^I G_2^I + P_3^I P_2^I G_1^I + P_3^I P_2^I P_1^I G_0^I + P_3^I P_2^I P_1^I P_0^I c_0$$

32 bit adder cascading two 16 bit Carry lookahead adder



Gate Delay 32 bit adder

$G, P = 1$

$G^I, P^I = 3^{\text{rd}}$ gate delay

$C_4, C_8, C_{12}, C_{16} = 5^{\text{th}}$ gate delay

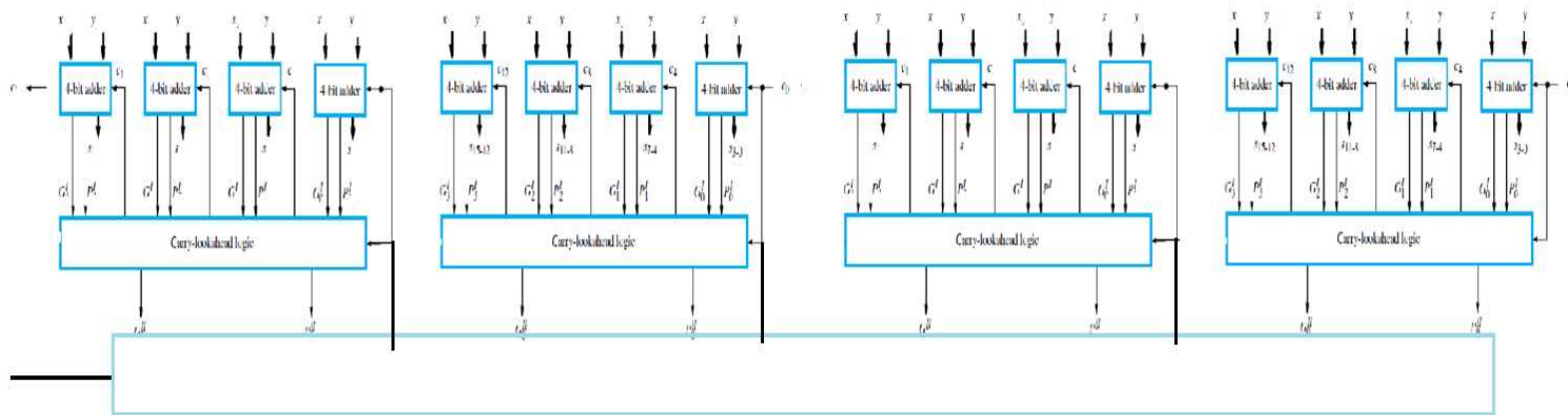
C_{12} to $C_{15} = 7^{\text{th}}$ gate delay

$G^I, P^I = 7^{\text{th}}$ gate delay

$C_{20}, C_{24}, C_{28}, C_{32} = 9^{\text{th}}$ gate delay

$S_{31} = 10^{\text{th}}$ gate delay

64 bit Carry lookahead adder



Gate Delay

$G, P = 1$

$G^I, P^I = 3^{\text{rd}}$ gate delay

$G^{II}, P^{II} = 5^{\text{th}}$ gate delay

$C_{16}, C_{32}, C_{48}, C_{64} = 7^{\text{th}}$ gate delay

$C_{52}, C_{56}, C_{60} = 9^{\text{th}}$ gate delay

$C_{61}, C_{62}, C_{63} = 11^{\text{th}}$ gate delay

$S_{63} = 12^{\text{th}}$ gate delay

Overflow conditions

Operation	Operand A	Operand B	Result indicating overflow
$A + B$	≥ 0	≥ 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	≥ 0	< 0	< 0
$A - B$	< 0	≥ 0	≥ 0

Overflow conditions

Sequence of MIPS instructions can discover overflow - signed addition

```
addu $t0, $t1, $t2 # $t0 = sum, but don't trap
xor  $t3, $t1, $t2 # Check if signs differ
slt  $t3, $t3, $zero # $t3 = 1 if signs differ
bne  $t3, $zero, No_overflow # $t1, $t2 signs ≠,
                                # so no overflow
xor  $t3, $t0, $t1 # signs =; sign of sum match too?
                                # $t3 negative if sum sign different
slt  $t3, $t3, $zero # $t3 = 1 if sum sign different
bne  $t3, $zero, Overflow # All 3 signs ≠; goto overflow
```

Overflow conditions

Sequence of MIPS instructions can discover overflow - unsigned addition

```
addu $t0, $t1, $t2    # $t0 = sum
nor $t3, $t1, $zero    # $t3 = NOT $t1
                        # (2's comp - 1:  $2^{32} - \$t1 - 1$ )
sltu $t3, $t3, $t2     #  $(2^{32} - \$t1 - 1) < \$t2$ 
                        #  $\Rightarrow 2^{32} - 1 < \$t1 + \$t2$ 
bne $t3, $zero, Overflow # if( $2^{32}-1 < \$t1+\$t2$ ) goto overflow
```