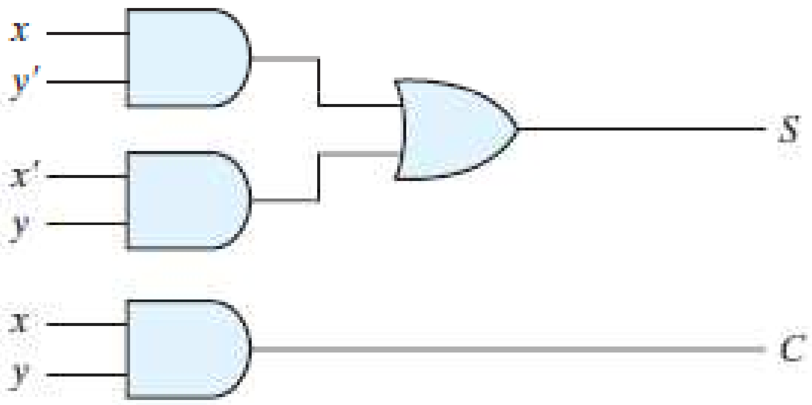
BINARY ADDER–SUBTRACTOR

Half Adder

- Circuit needs two binary inputs and two binary outputs

Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

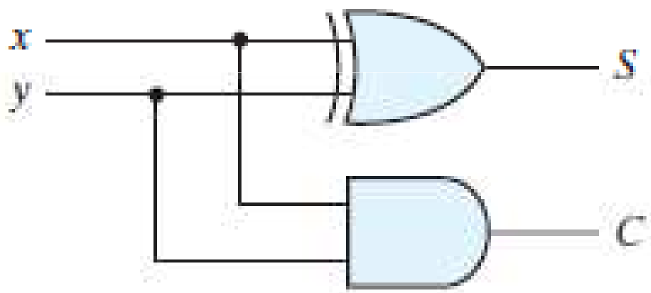


(a) $S = xy' + x'y$
$C = xy$

Half Adder

- Two half adders can be used to construct a full adder
- S=x'y+xy' = x⊕y

$$S = x'y + xy'$$
$$C = xy$$



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

Full Adder

- *Addition of n-bit binary numbers requires the use of a full adder*
- *Addition proceeds on a bit-by-bit basis*
- *Right to left, beginning with the least significant bit*
- *Must consider a possible carry out from the previous position*

Full Adder

- *A full adder add three bits and produces two output*
- *Input variables, denoted by x and y , represent the two significant bits to be added*
- *The third input, z , represents the carry from the previous lower significant position*
- *Output bits are Sum and carry*

Full Adder

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

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

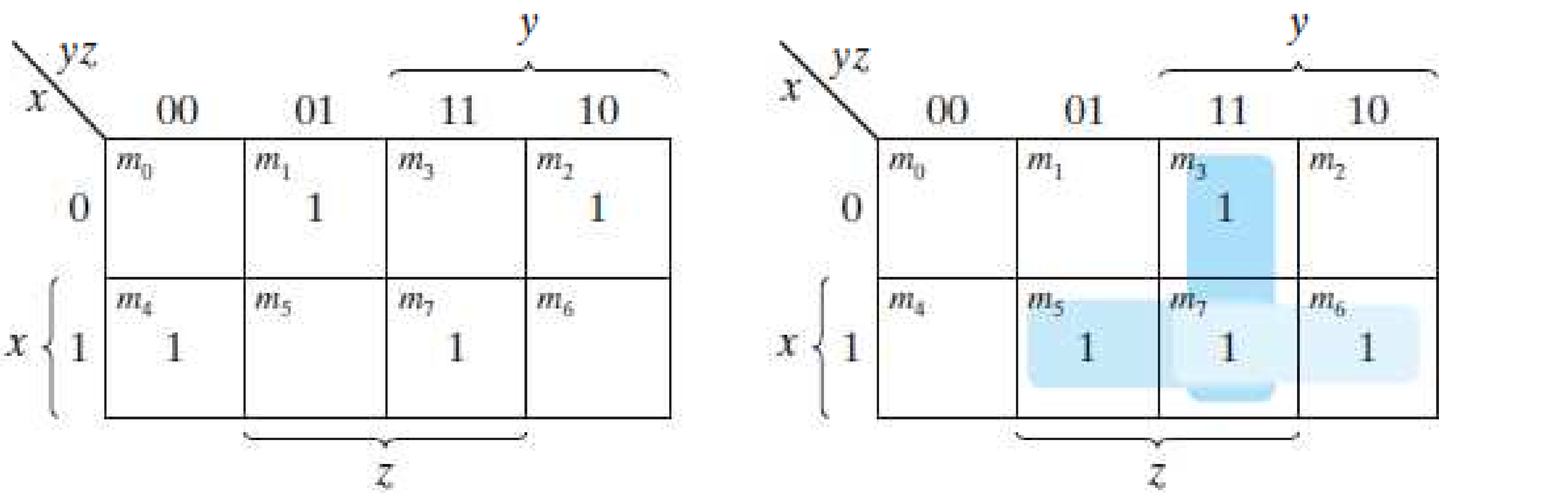
Full Adder

Full Adder

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

$S = x'y'z + x'yz' + xy'z' + xyz$

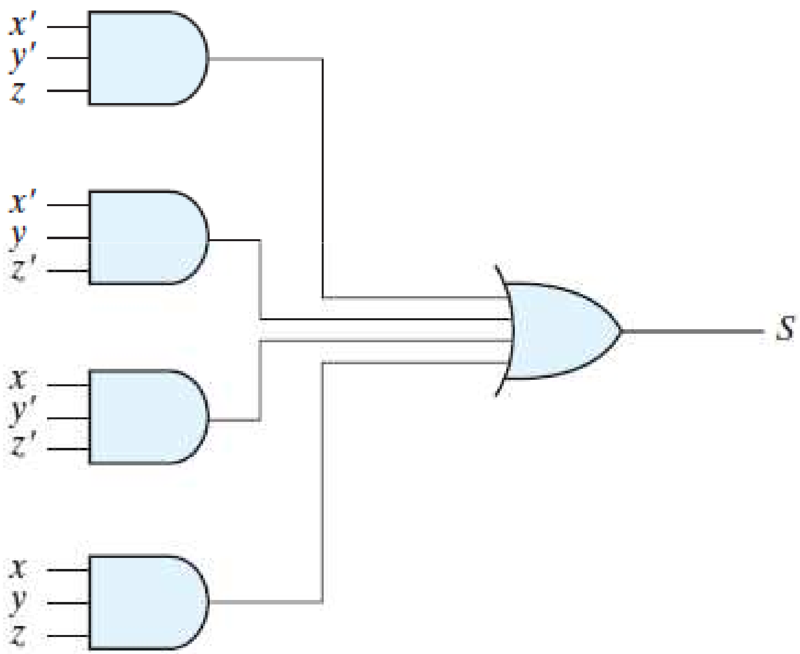
$C = xy + xz + yz$

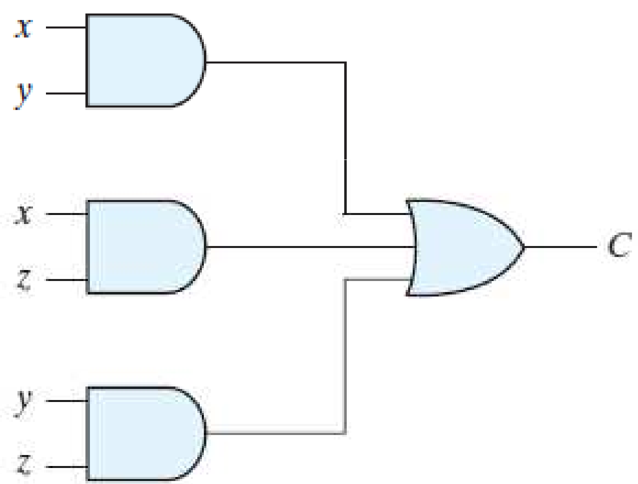


(a) $S = x'y'z + x'yz' + xy'z' + xyz$ (b) $C = xy + xz + yz$

Full Adder

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$





Full Adder

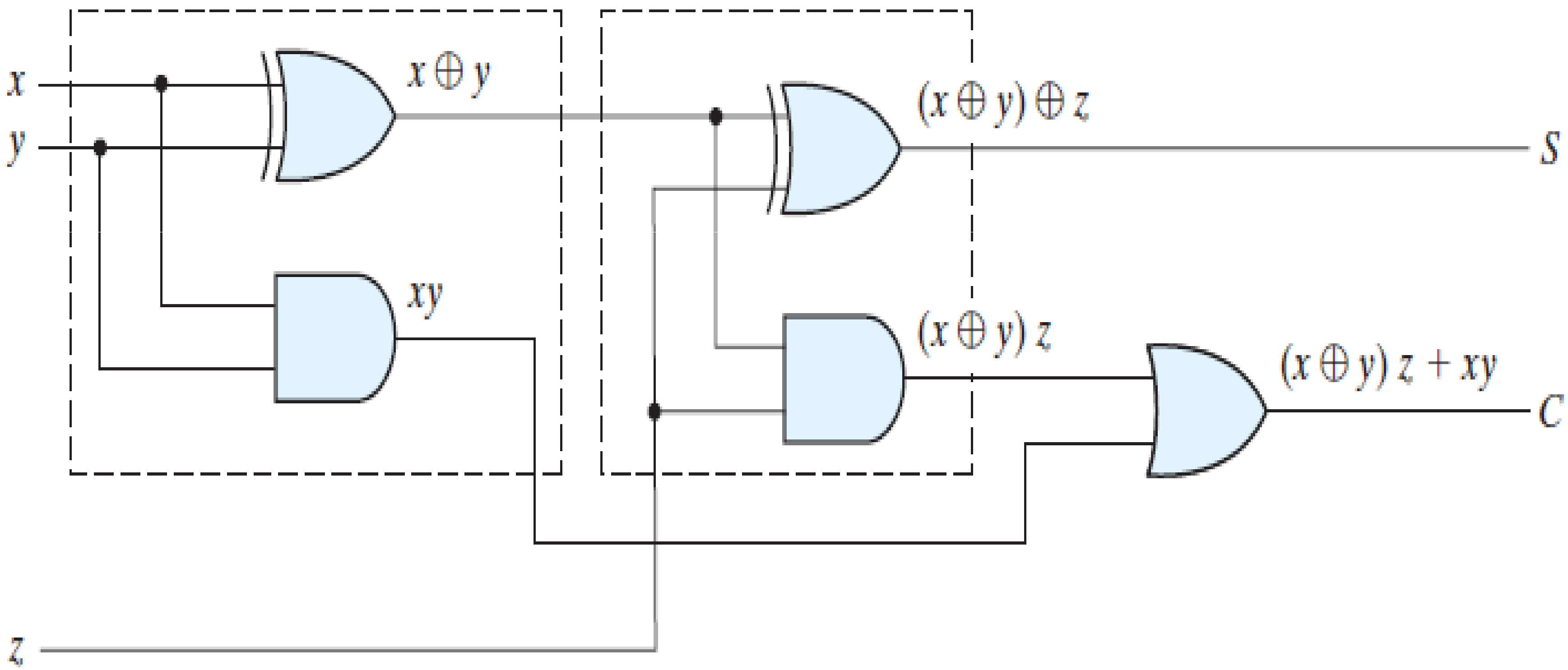
$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

$$\begin{aligned} S &= z \oplus (x \oplus y) \\ &= z'(xy' + x'y) + z(xy' + x'y)' \\ &= z'(xy' + x'y) + z(xy + x'y') \\ &= xy'z' + x'yz' + xyz + x'y'z \end{aligned}$$

$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

$$(x \oplus y)z + xy$$

Full Adder



Binary Adder

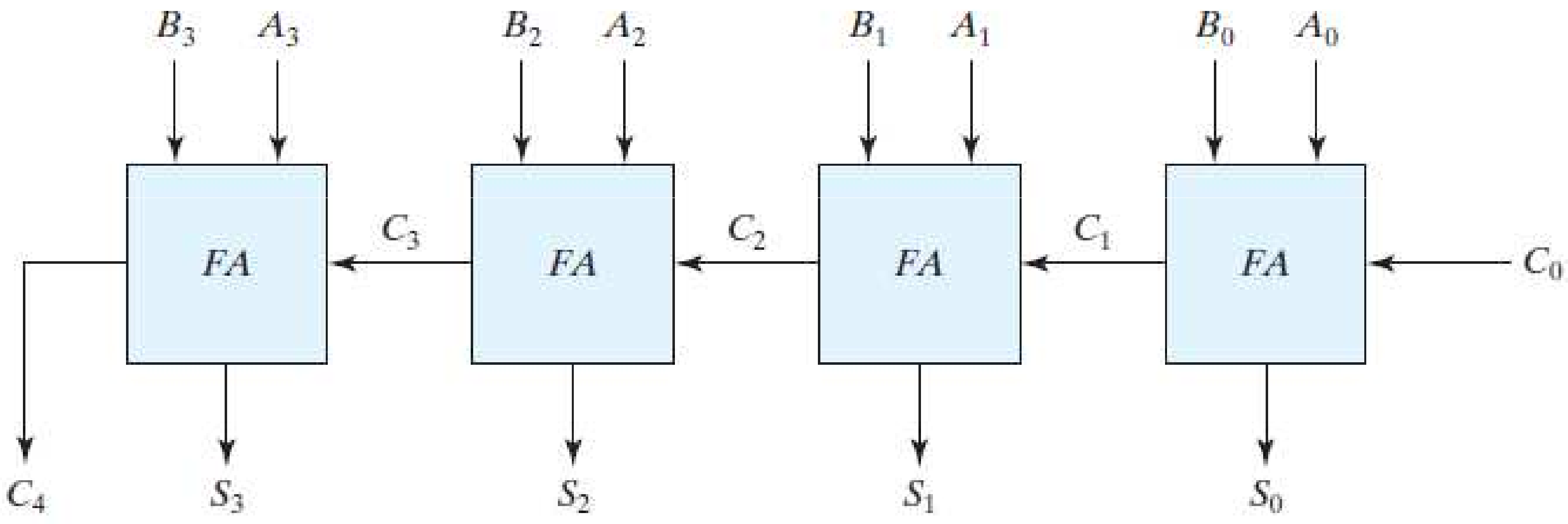
- Constructed with full adders connected in cascade
- Output carry from each full adder connected to the input carry of the next full adder in the chain
- Addition of *n-bit numbers requires a chain of n full adders*

Binary Adder

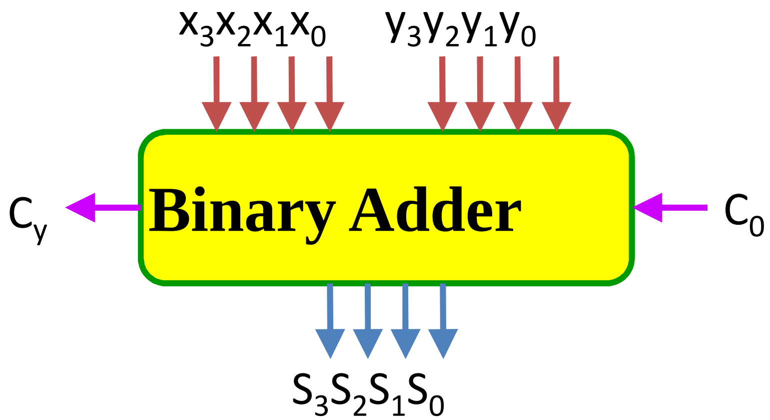
Subscript *i*:	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

Binary Adder

Ripple Carry Adder

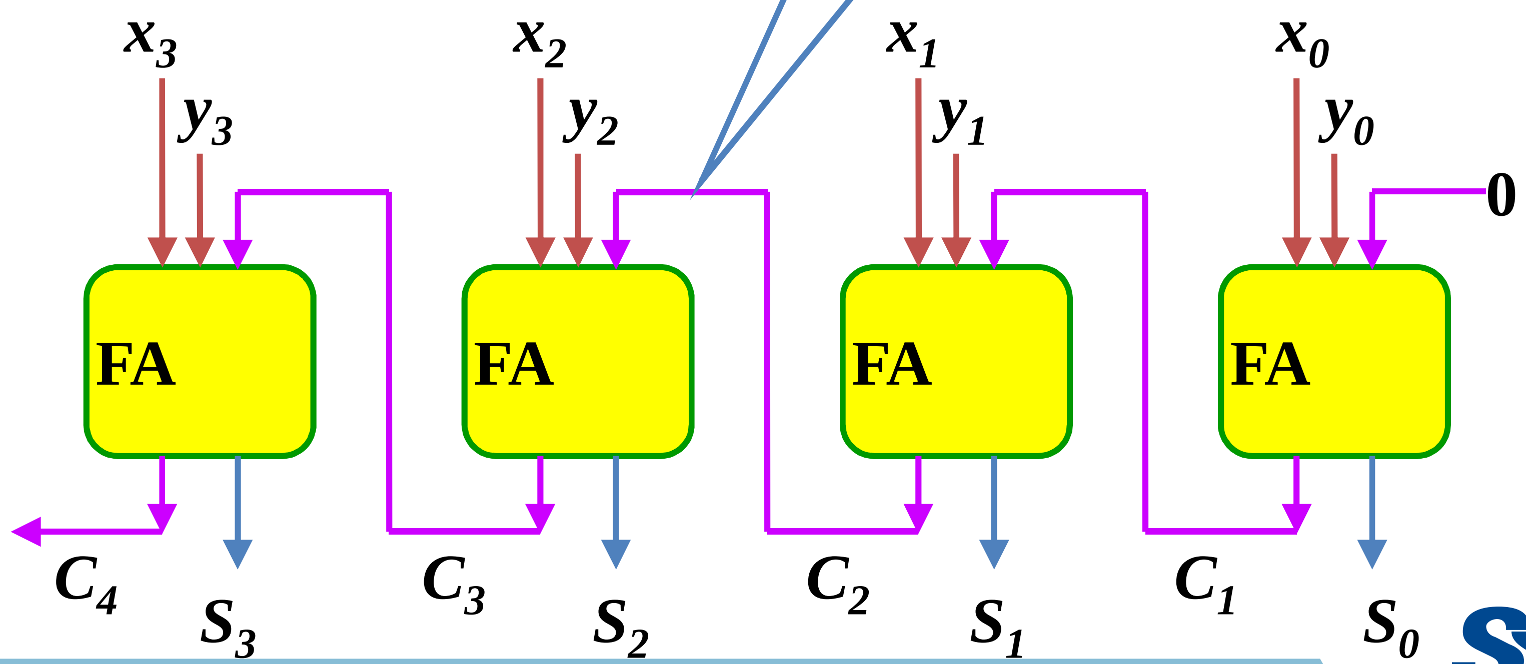


Binary Adder



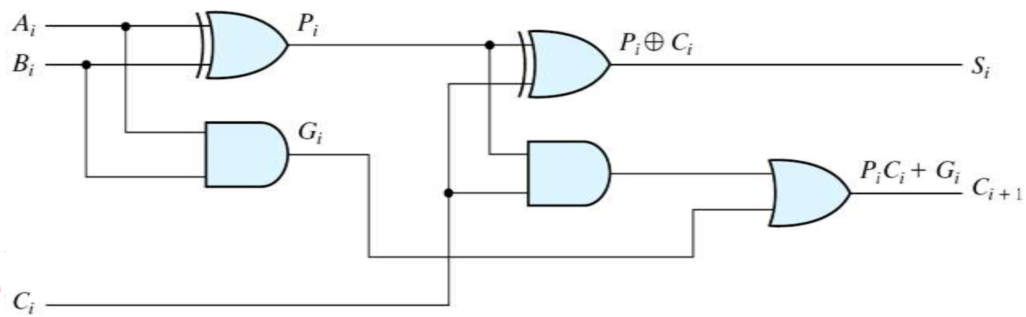
Carry Propagate Addition
n bit need 2*n gate delay

$$\begin{array}{ccccc} & c_3 & c_2 & c_1 & \\ + & x_3 & x_2 & x_1 & x_0 \\ + & y_3 & y_2 & y_1 & y_0 \\ \hline Cy & S_3 & S_2 & S_1 & S_0 \end{array}$$



Parallel Adders

- Reduce the carry propagation delay
  - Employ faster gates
  - Look-ahead carry (more complex mechanism, yet faster)
  - Carry propagate: $P_i = A_i \oplus B_i$
  - Carry generate: $G_i = A_i B_i$
  - Sum: $S_i = P_i \oplus C_i$
  - Carry: $C_{i+1} = G_i + P_i C_i$
  - C_0 = Input carry
  - $C_1 = G_0 + P_0 C_0$
  - $C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0$
  - $C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

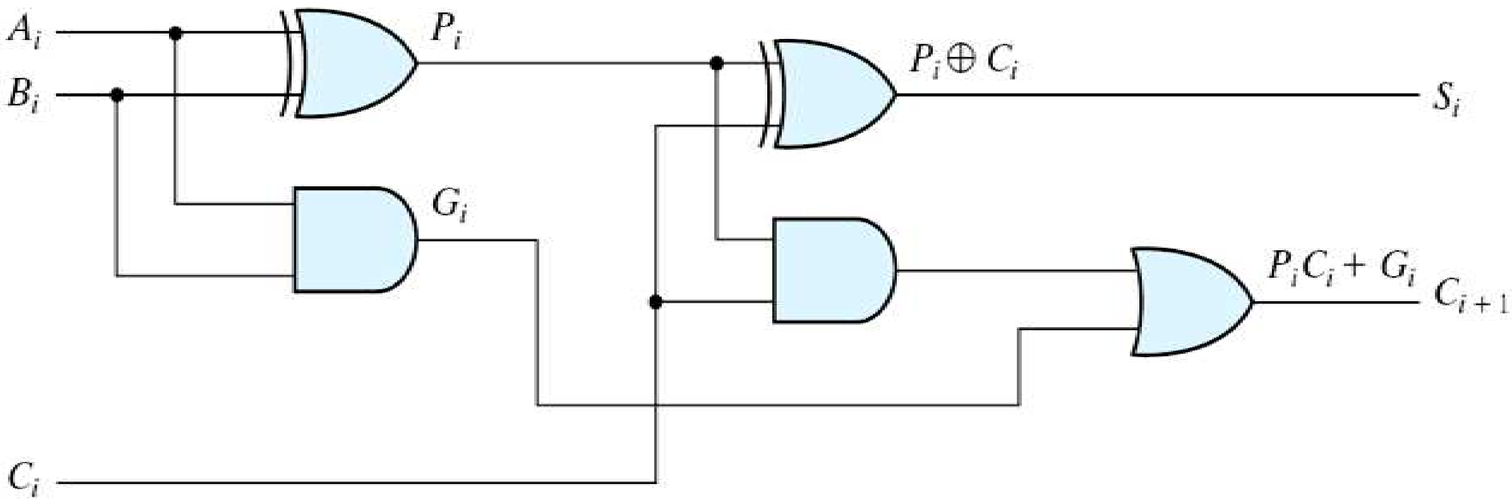


Carry Propagation

P_i is the carry Propagator
G_i is the carry Generator

$$P_i = A_i \oplus B_i$$
$$G_i = A_iB_i$$

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_iC_i$$



Carry Propagation

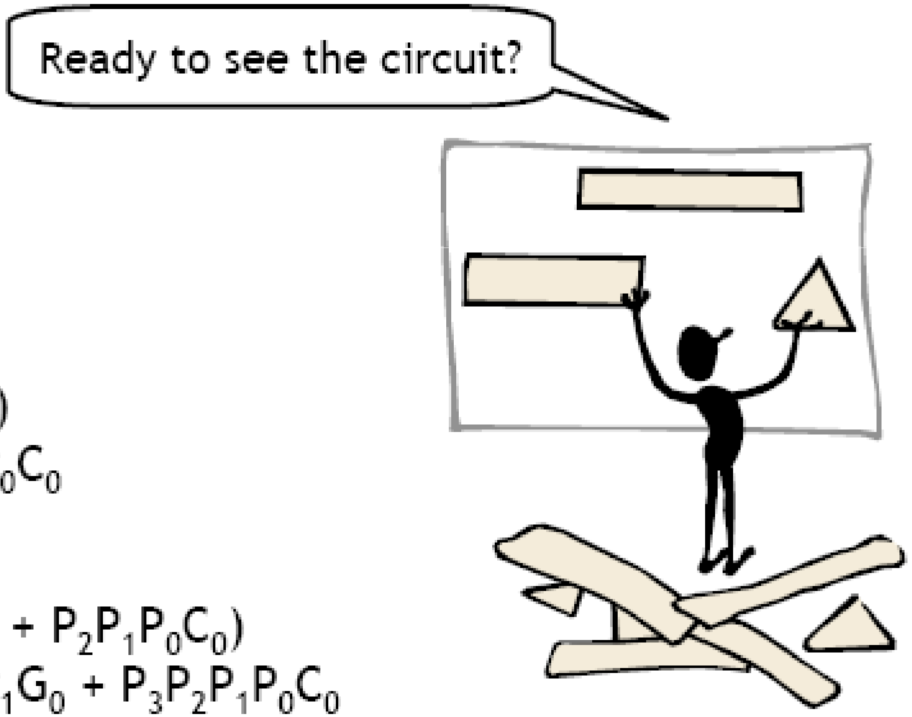
- Let's look at the carry out equations for specific bits, using the general equation from the previous page $C_{i+1} = G_i + P_iC_i$.

$$C_1 = G_0 + P_0C_0$$

$$\begin{aligned} C_2 &= G_1 + P_1C_1 \\ &= G_1 + P_1(G_0 + P_0C_0) \\ &= G_1 + P_1G_0 + P_1P_0C_0 \end{aligned}$$

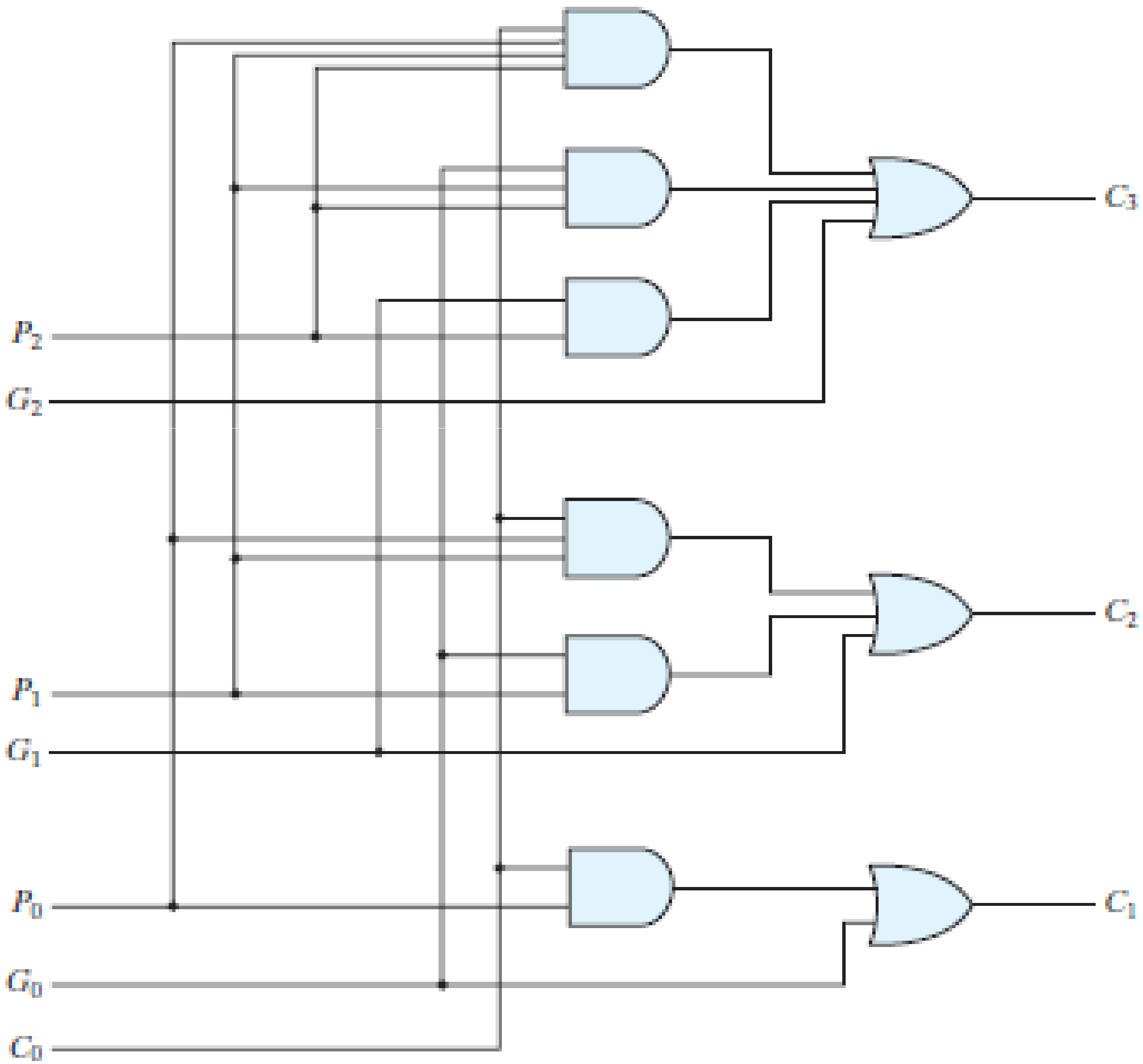
$$\begin{aligned} C_3 &= G_2 + P_2C_2 \\ &= G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0) \\ &= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \end{aligned}$$

$$\begin{aligned} C_4 &= G_3 + P_3C_3 \\ &= G_3 + P_3(G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0) \\ &= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0 \end{aligned}$$

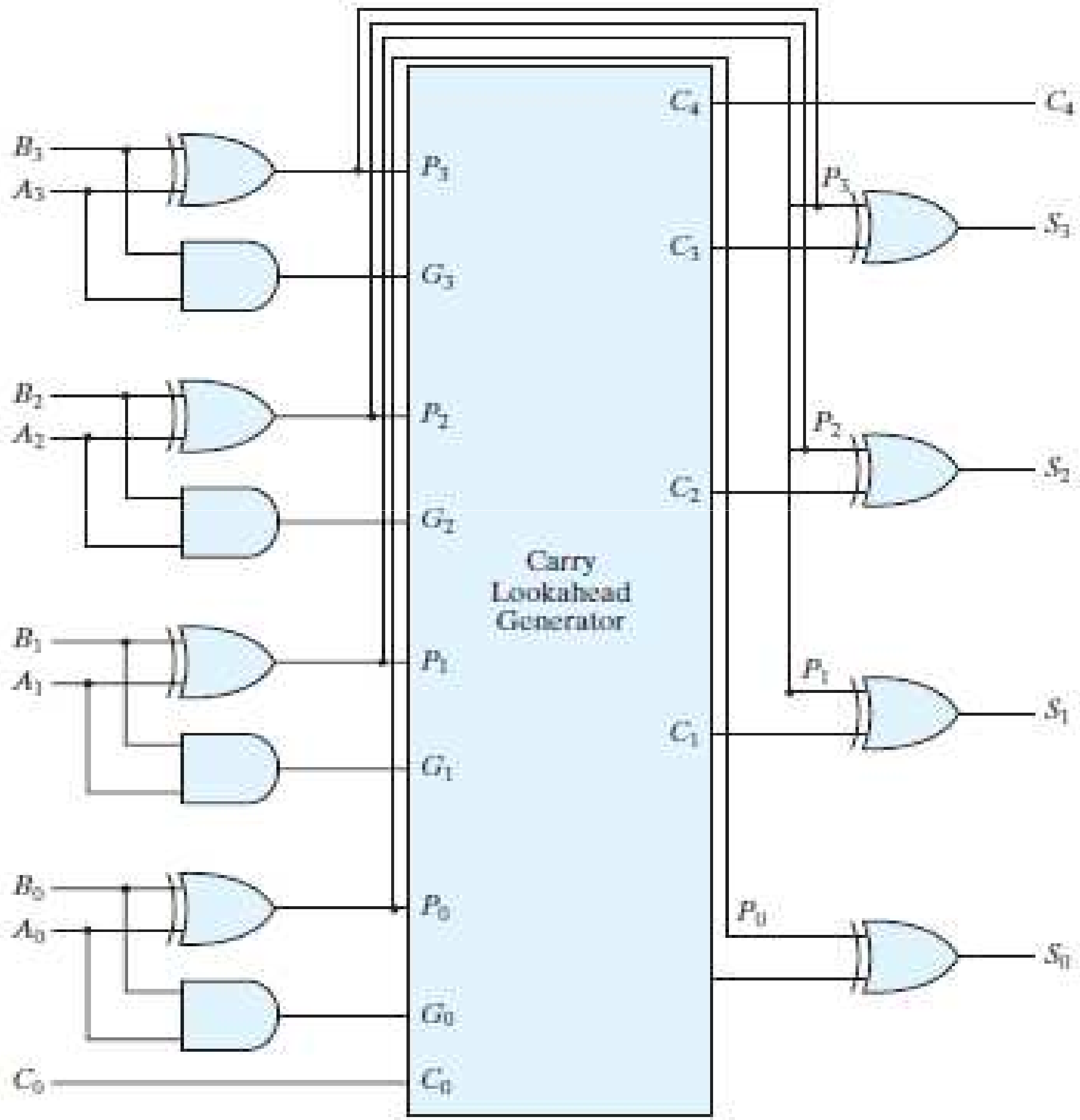


- These expressions are all sums of products, so we can use them to make a circuit with only a two-level delay.

Carry Lookahead Logic

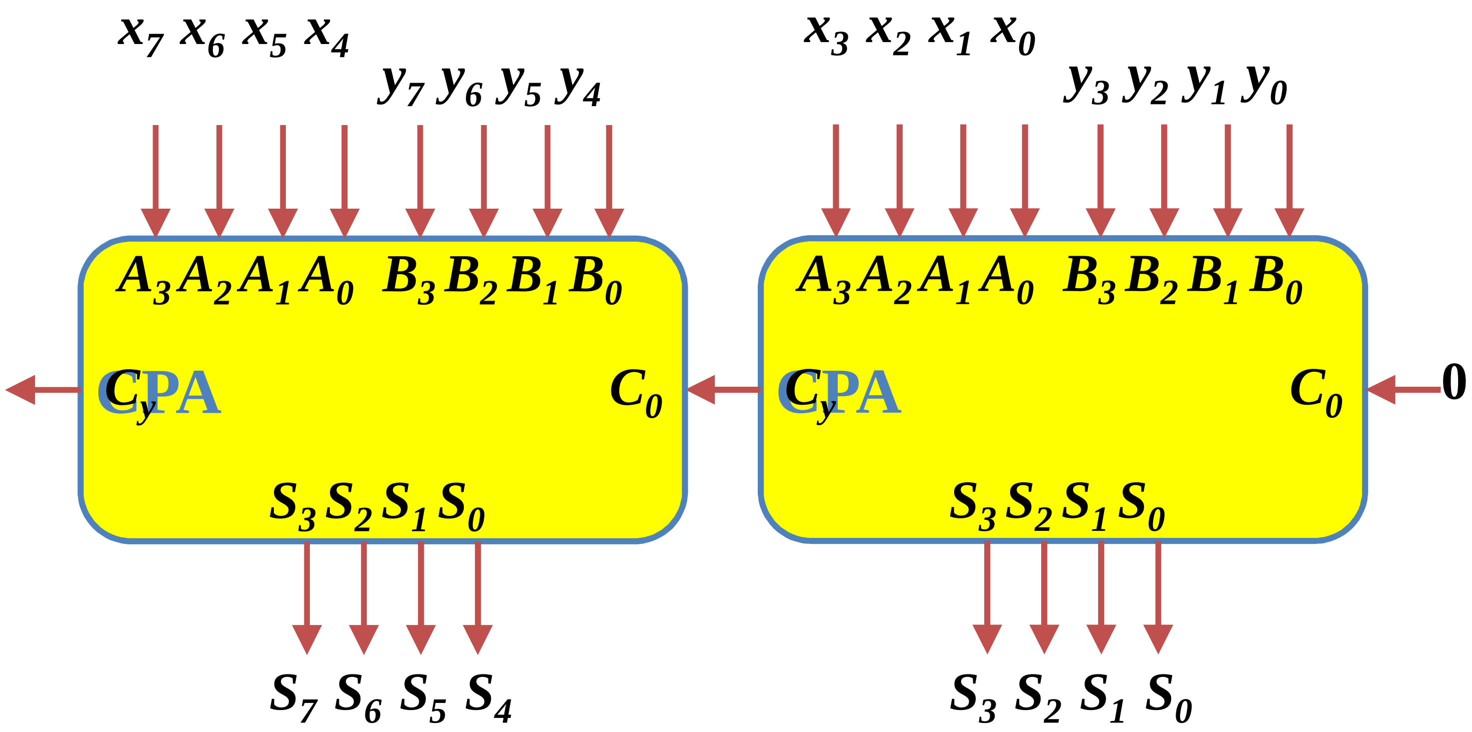


Carry Lookahead adder



Binary Adder

- Carry Propagate Adder

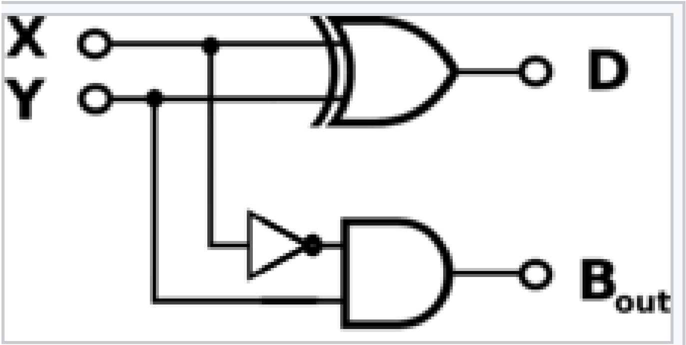


Half Subtractor

Inputs		Outputs	
X	Y	D	B_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$D = X \oplus Y$$

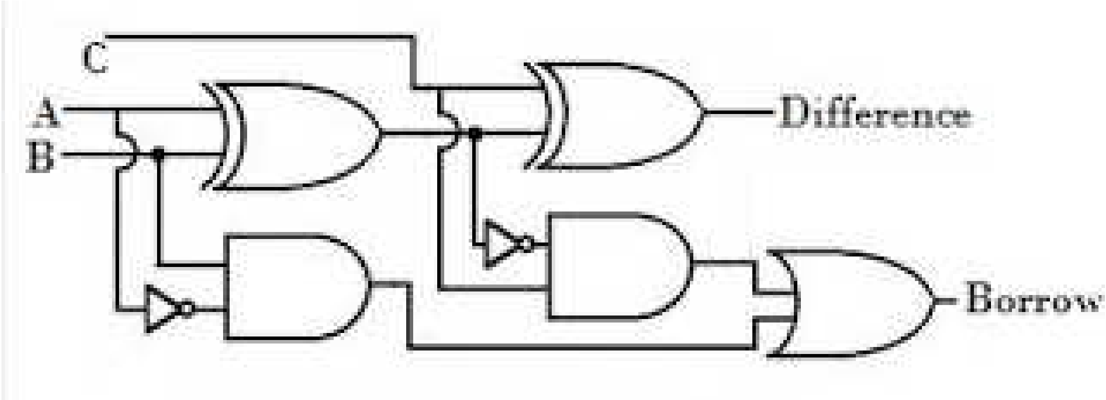
$$B_{\text{out}} = \overline{X} \cdot Y.$$



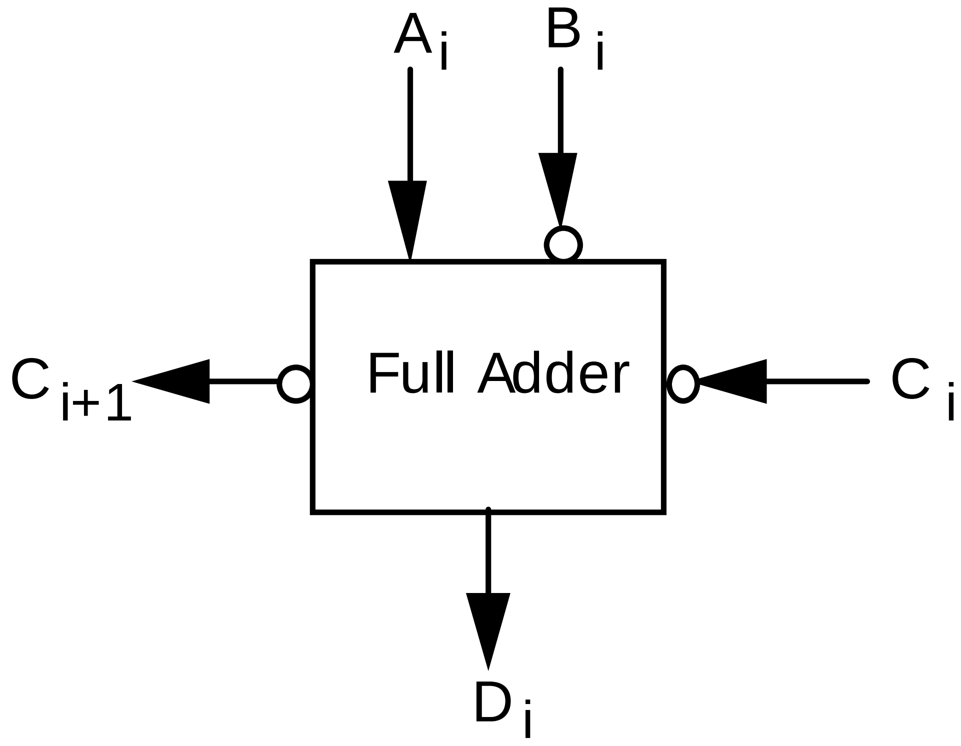
Full Subtractor

Inputs			Outputs	
X	Y	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

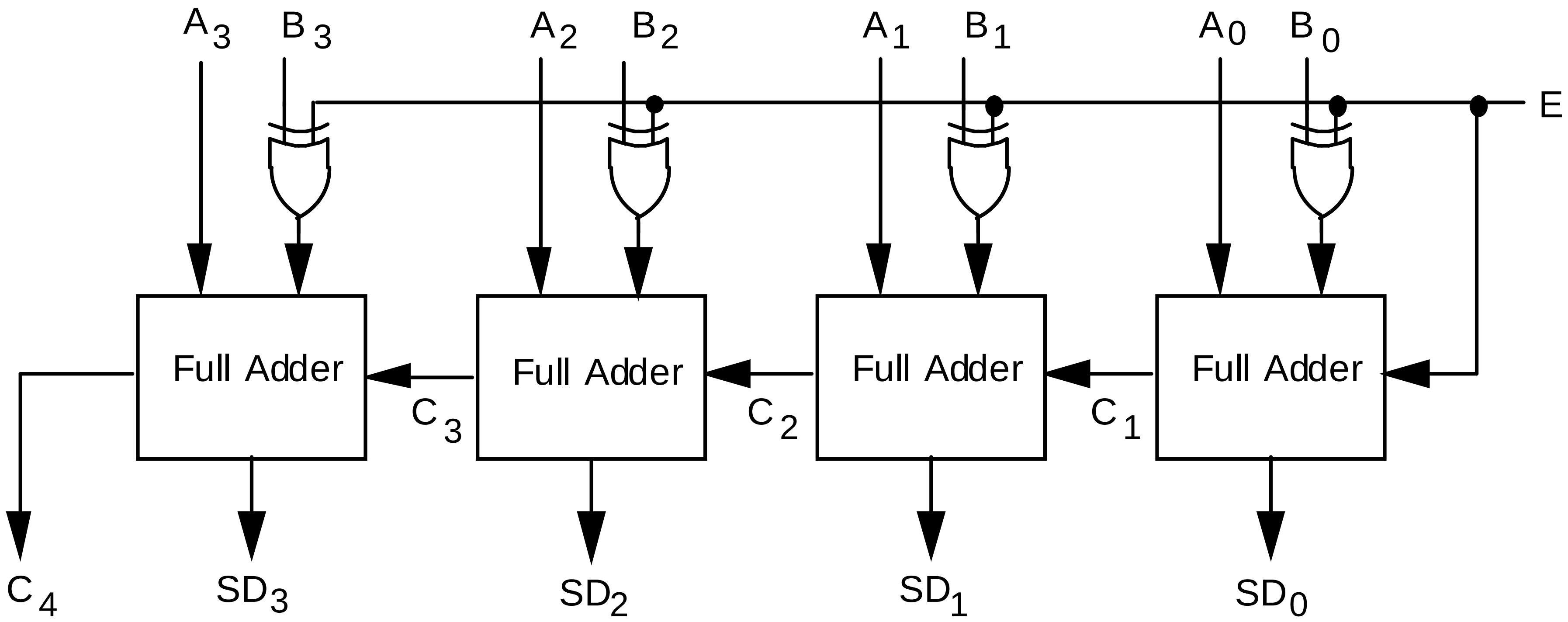
$D = A \oplus B \oplus B_{in}$ and $B_{out} = A'B_{in} + A'B + BB_{in}$



Making a full subtractor from a full adder



Adder/Subtractor-2

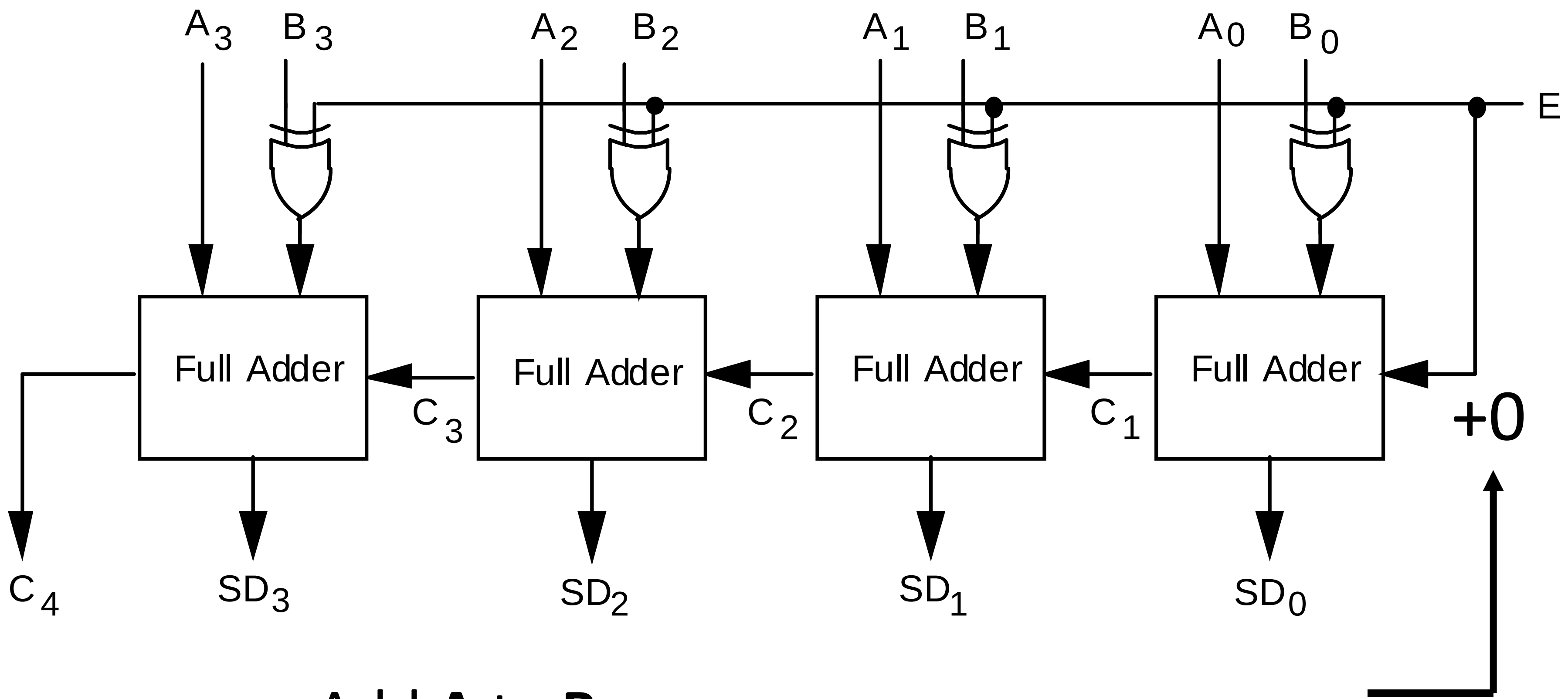


E = 0: 4-bit adder

E = 1: 4-bit subtractor

B	E	B xor E	
0	0	0	E=0 Add
1	0	1	
0	1	1	E=1 Sub
1	1	0	

4-bit Adder: E = 0

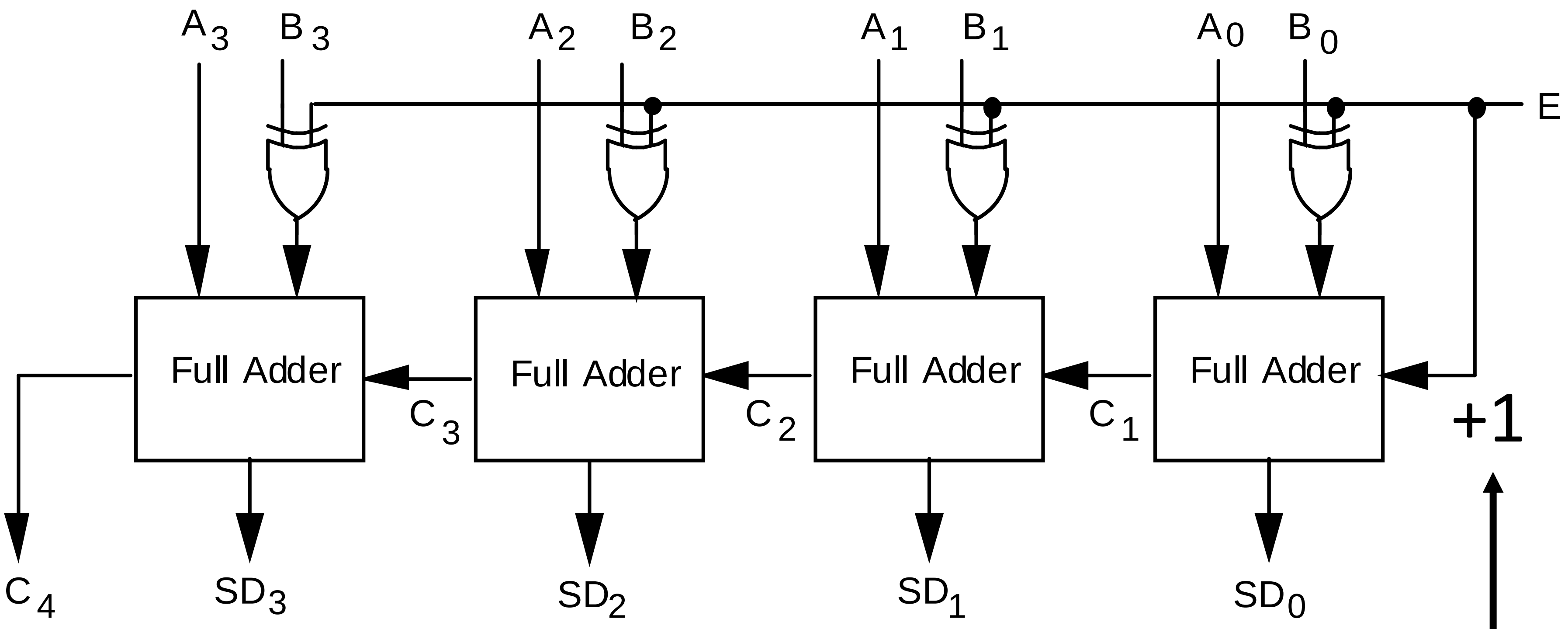


Add **A** to **B**

That is, add **A** to **B**

D = **A** + **B**

4-bit Subtractor: E = 1



Add **A** to **!B** (one's complement) plus 1

That is, add **A** to two's complement of **B**

D = **A** - **B**

Checking Overflow

- V detects overflow
- V=0 means no overflow
- V=1 means the result is wrong because of overflow
- Overflow - when adding two numbers of the same sign (both negative or positive)
- And result can not be shown with the available bits.

C3	C4	V= C3 xor C4
0	0	0
1	0	1 overflow
0	1	1 overflow
1	1	0

4-bit adder subtractor with overflow

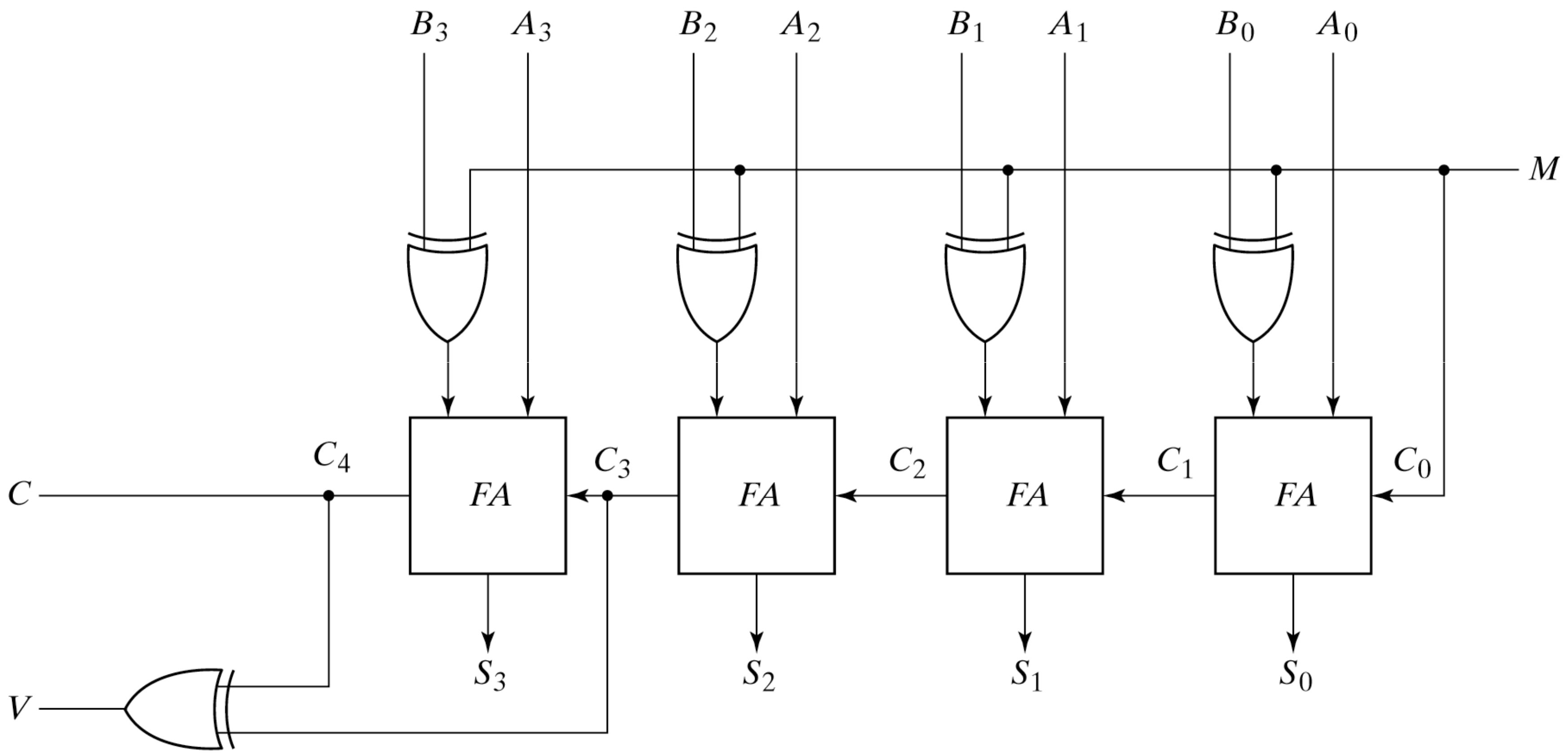


Fig. 4-13 4-Bit Adder Subtractor