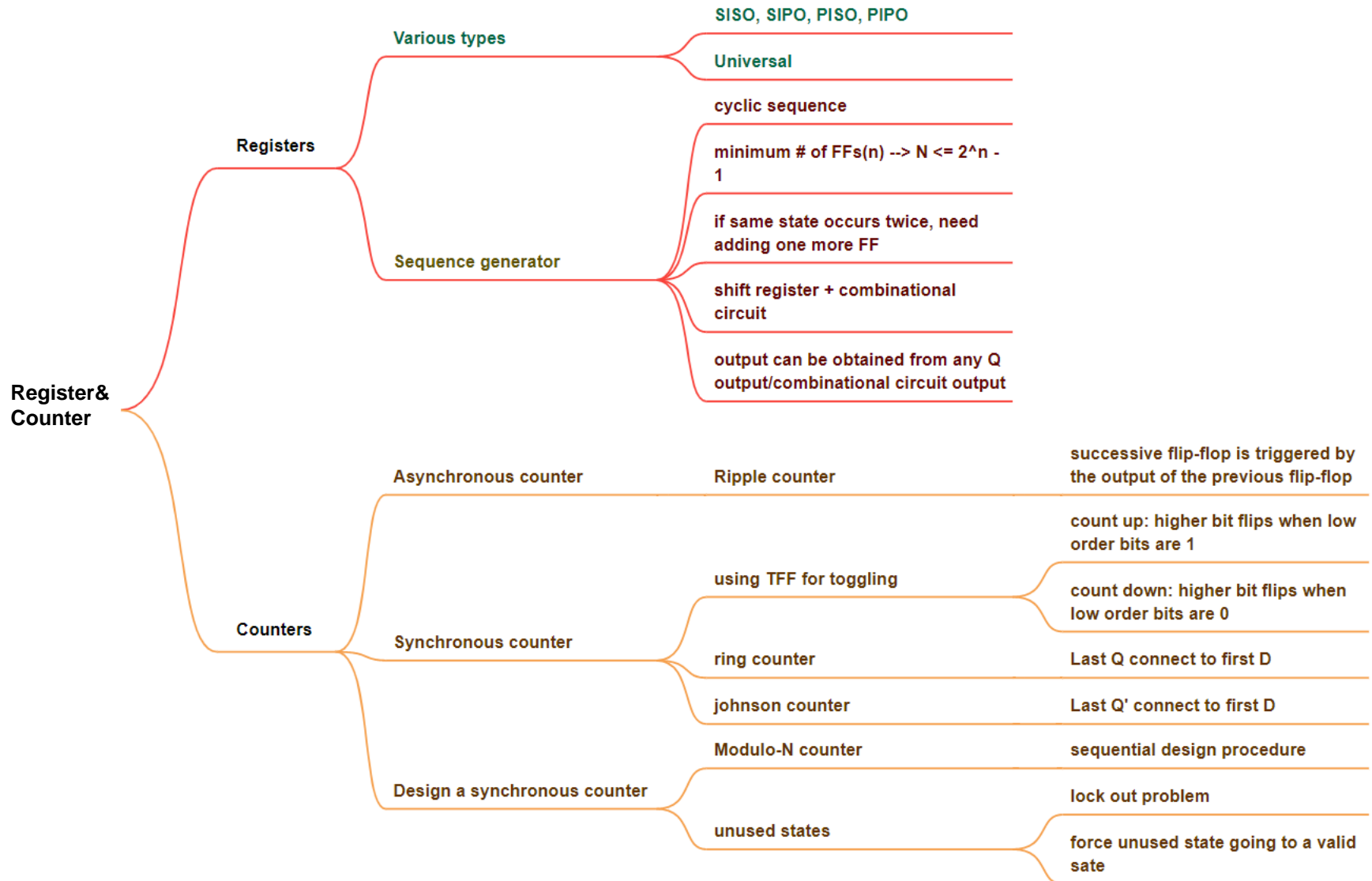# DIGITAL LOGIC

Chapter 4 part3:
Arithmetic Circuit

2023 Fall

Today's Agenda

- Recap
- Context
  - Binary Adder-Subtractor
  - Decimal Adder
  - Binary Multiplier
- Reading: Textbook, Chapter 4.5-4.7

Recap

Register & Counter

- **Registers**
  - **Various types**
    - SISO, SIPO, PISO, PIPO
    - Universal
  - **Sequence generator**
    - cyclic sequence
    - minimum # of FFs(n) --> N <= 2^n - 1
    - if same state occurs twice, need adding one more FF
    - shift register + combinational circuit
    - output can be obtained from any Q output/combinational circuit output

- **Counters**
  - **Asynchronous counter**
    - **Ripple counter**
      - successive flip-flop is triggered by the output of the previous flip-flop
  - **Synchronous counter**
    - **using TFF for toggling**
      - count up: higher bit flips when low order bits are 1
      - count down: higher bit flips when low order bits are 0
    - **ring counter** — Last Q connect to first D
    - **johnson counter** — Last Q' connect to first D
  - **Design a synchronous counter**
    - **Modulo-N counter** — sequential design procedure
    - **unused states**
      - lock out problem
      - force unused state going to a valid sate

Outline

- **Binary Adder**
- Binary Subtractor
- Decimal Adder (BCD)
- Binary Multiplier
- Other Arithmetic Functions

Binary Add

- Similar to the addition operation of decimal numbers.
- 0+0 = 0, 0+1 = 1, 1+0 = 1, 1+1 = **1**0 ← The higher significant bit is called a **carry** (进位).
- A combinational circuit that performs the addition of two bits as described above is called a *half-adder*.
- The addition operation involves three bits — the *augend bit*, *addend bit*, and the *carry bit* and produces a sum result as well as carry.
- The combinational circuit performing this type of addition operation is called a *full-adder*.

```
  11    carry(进位)
1011    augend(被加数)
0001    addend(加数)
————————————————
1100    sum
```

Recall: Design Procedure

1. Specification: From the specifications, determine the inputs, outputs, and their symbols.

2. Formulation: Derive the truth table (functions) from the relationship between the inputs and outputs

3. Optimization: Derive the simplified Boolean functions for each output function. Draw a logic diagram for the resulting circuits using AND, OR, and inverters.

4. Technology Mapping: Transform the logic diagram to a new diagram using the available implementation technology.

• Verification: Verify the design.

Half-adder

- 1. Spec
  - Inputs: x, y
  - Outputs: C(carry), S(sum)

- 2. Truth table

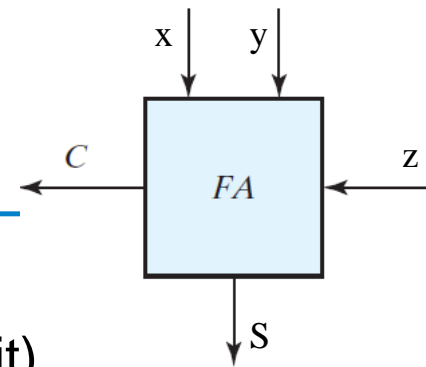|		2^1	2^0
x	**y**	**C**	**S**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- 3. Boolean function

  $S = x'y + xy' = x \oplus y$

  $C = xy$

- 4. Block diagram

or
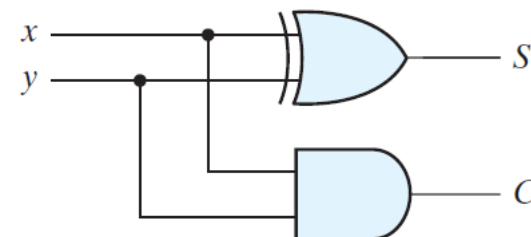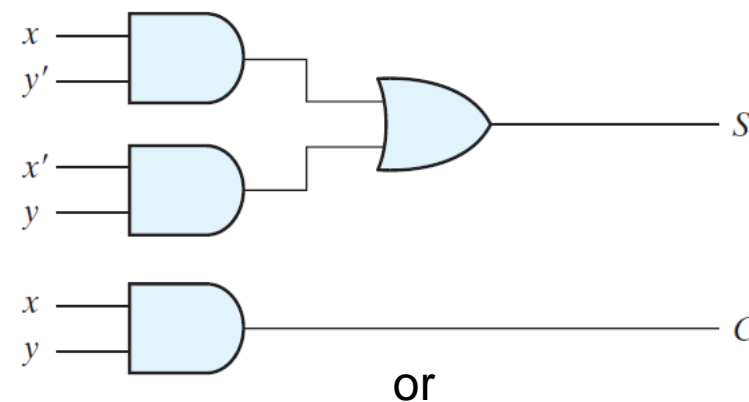
Full-adder

- 1. Spec
  - Inputs: x, y, z(carry from previous lower significant bit)
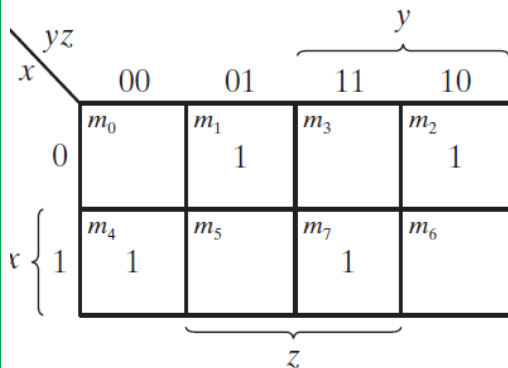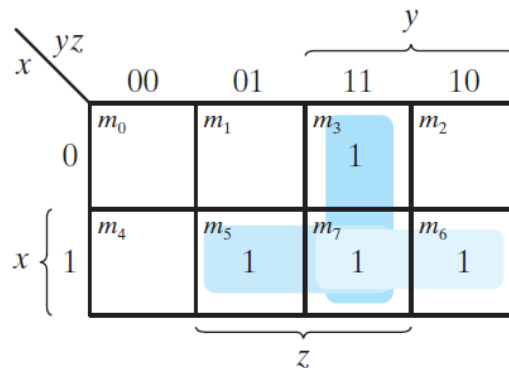 - Outputs: C(carry), S(sum)
- 2. Truth table
- 3. Boolean function

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

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

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

(b) $C = xy + xz + yz$

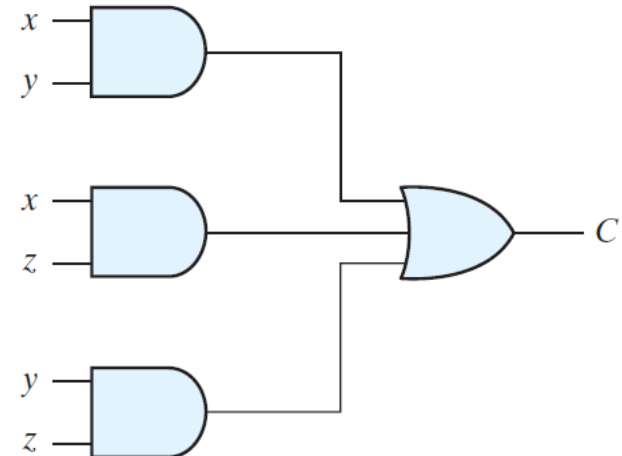xor gate, odd number of 1 → sum = 1

if >= two inputs are 1 → carry = 1

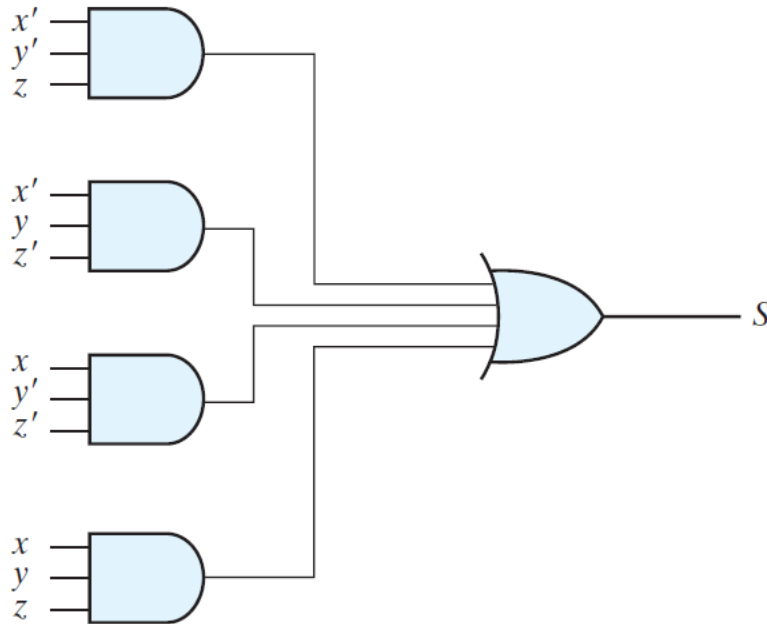# Full-adder

- 3. Boolean function
  S = x'y'z + x'yz' + xy'z' + xyz = x⊕y⊕z
  C = xy + xz + yz

- 4. Block diagram

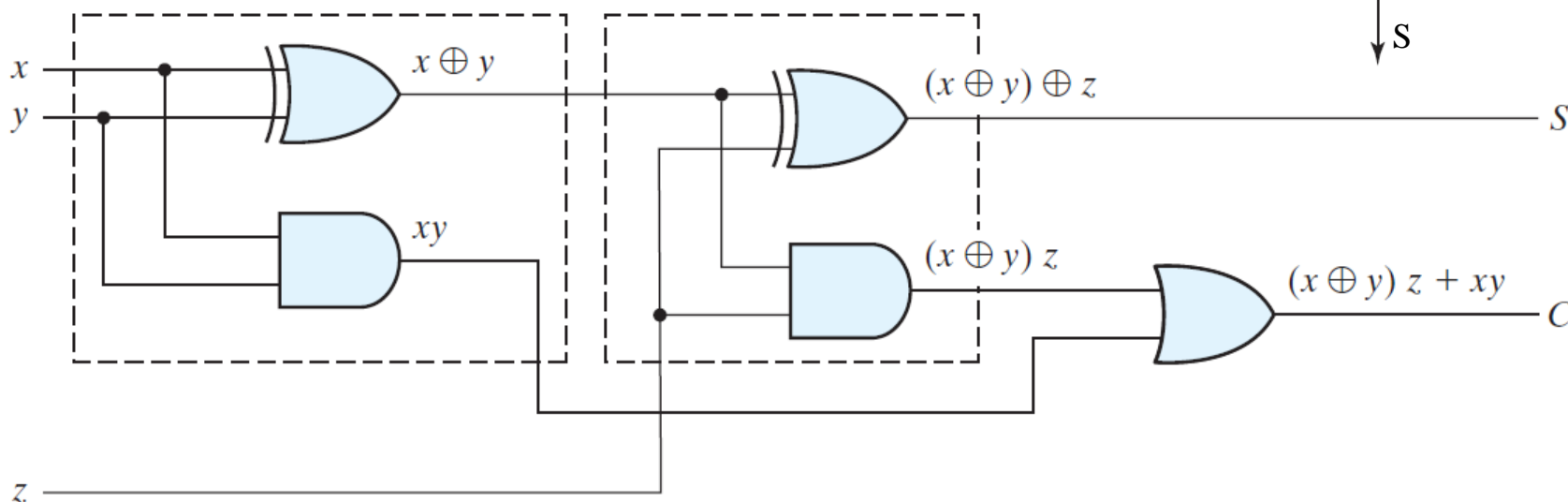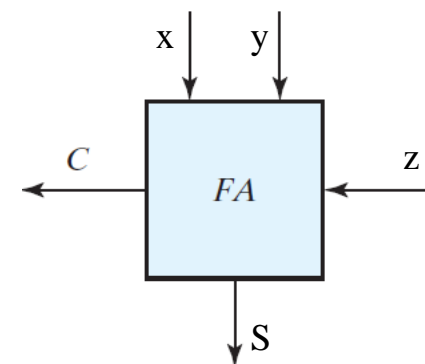# Full Adder Implemented with Half Adders

- Full adder implemented with: Two half adders and one OR gate

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

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

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

$= z \oplus (x \oplus y)$

$C = z(xy' + x'y) + xy = z(x \oplus y) + xy$

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

Ripple-Carry Adder

- unsigned addition
- $(C_n S_{n-1} S_{n-2}...S_0)=(A_{n-1}A_{n-2}...A_0)+(B_{n-1}B_{n-2}...B_0)$
- eg. S=A+B, $A=A_3 A_2 A_1 A_0$, $B=B_3 B_2 B_1 B_0$, $S=S_3 S_2 S_1 S_0$

The computation time of a ripple-carry adder grows linearly with word length n

T=O(n) due to carry chain

	C_{i+1}			C_i	
C	1	0	1	1	0
A		1	0	1	1
B		1	0	0	1
S	1	0	1	0	0

A_i
B_i

B[3:0]   A[3:0]

C_4 4-bit binary adder C_0

S[3:0]
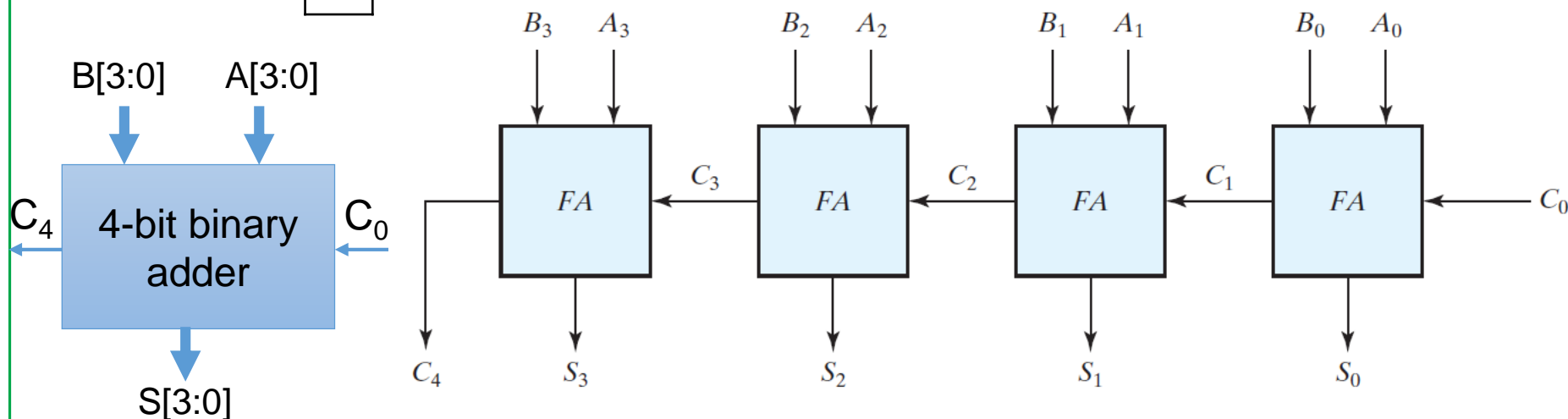
Ripple-Carry Adder

Ai \ BiCi	00	01	11	10
0	m_0	m_1 1	m_3	m_2 1
1	m_4 1	m_5	m_7 1	m_6

$$S_i = A_i \oplus B_i \oplus C_i)$$

Ai \ BiCi	00	01	11	10
0	m_0	m_1	m_3 1	m_2
1	m_4	m_5 1	m_7 1	m_6 1

$$C_{i+1} = A_iB_i + C_i(A_i \oplus B_i$$

Ai	Bi	Ci	Ci+1	Si
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
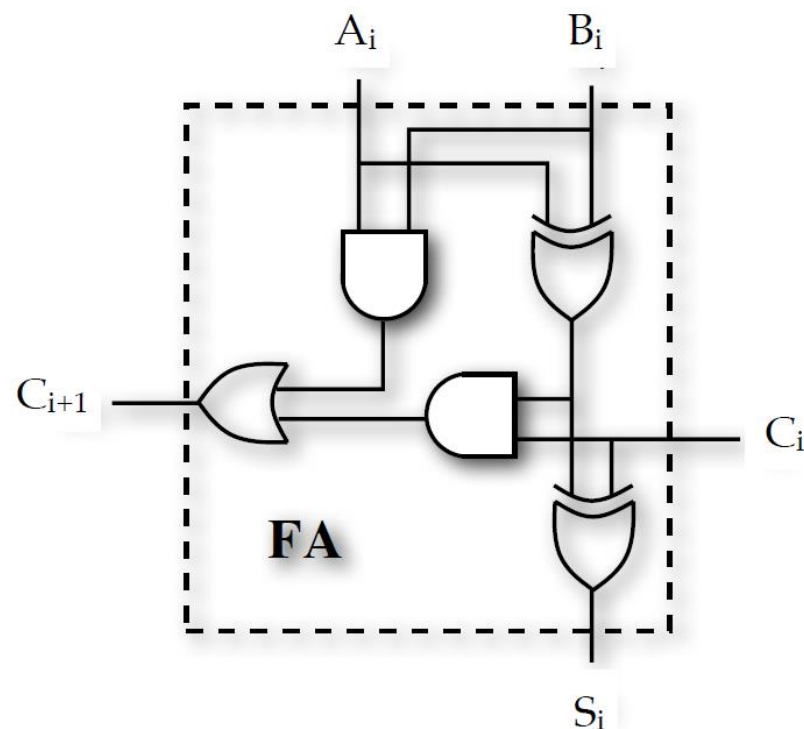
Carry Lookahead Adder

- For a full adder, define what happens to carry
  - Carry-generate: $C_{out}=1$ independent of C_{in}
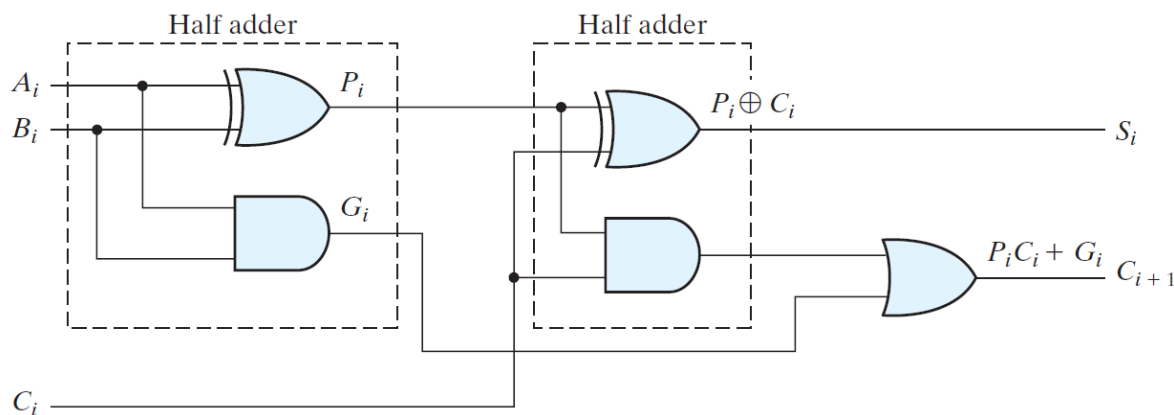   - $G_i = A_i \cdot B_i$
  - Carry-propagate: $C_{out}=C_{in}$
    - $P_i = A_i \oplus B_i$ or $P_i = A_i + B_i$
- Use the above G_i and P_i
  - $C_{i+1} = A_iB_i + B_iC_i + A_iC_i = A_iB_i + (A_i + B_i)C_i = \boxed{G_i + P_iC_i}$
  - $S_i = A_i \oplus B_i \oplus C_i = \boxed{P_i \oplus C_i}$

A_i	B_i	G_i	P_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	x

Carry Lookahead Adder

- Do not have to wait for C_i to compute C_{i+1}
  - $C_{i+1} = G_i + P_i C_i$
  - $C_{i+2} = G_{i+1} + P_{i+1} C_{i+1} = G_{i+1} + P_{i+1} G_i + P_{i+1} P_i C_i$
  - $C_{i+3} = G_{i+2} + P_{i+2} C_{i+2} = G_{i+2} + P_{i+2} G_{i+1} + P_{i+2} P_{i+1} G_i + P_{i+2} P_{i+1} P_i C_i$
  - $C_{i+4} = G_{i+3} + P_{i+3} C_{i+3} = G_{i+3} + P_{i+3} G_{i+2} + P_{i+3} P_{i+2} G_{i+1} + P_{i+3} P_{i+2} P_{i+1} G_i + P_{i+3} P_{i+2} P_{i+1} P_i C_i$

- Fixed delay time for each carry (but not the same for every gate!)

- Fanout of G_i & P_i also affect the overall delay $\rightarrow$ usually be limited to 4 bits

C_0 =input carry,
C_1 =G_0 +$P_0 C_0$,
C_2 =G_1 +$P_1 C_1$ = G_1 +$P_1 G_0$ +$P_1 P_0 C_0$,
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 Lookahead Adder

C_0 = input carry,

$C_1 = G_0 + P_0 C_0$,

$G_i = A_i \cdot B_i$

$P_i = A_i \oplus B_i$ or $P_i = A_i + B_i$
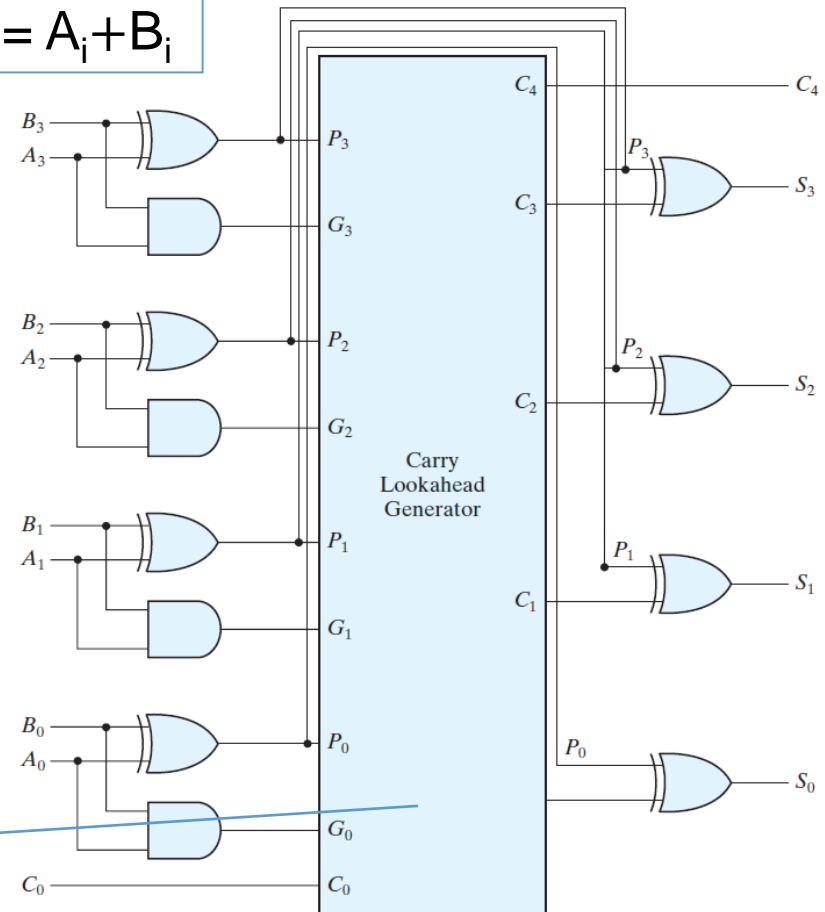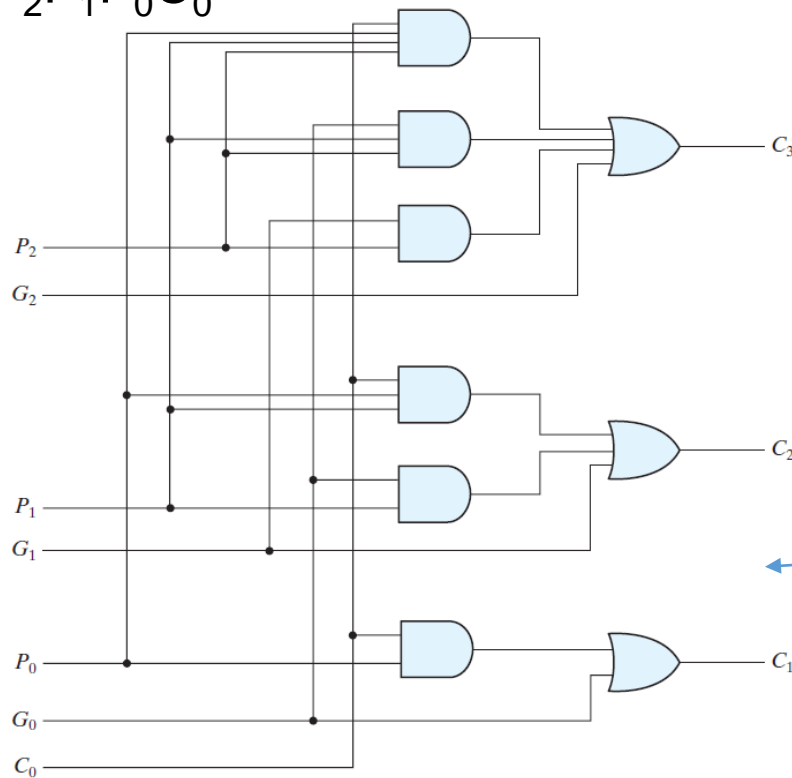
$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0$,

$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

Outline

- Binary Adder
- **Binary Subtractor**
- Decimal Adder (BCD)
- Binary Multiplier
- Other Arithmetic Functions

Binary Adders/Subtractors

- Binary subtraction normally is performed by adding the minuend to the 2's complement of the subtrahend.

M	Function	Comments
0	S=A+B	addition
1	S=A+B'+1	subtraction

overflow detection for signed 2's complement addition/subtraction
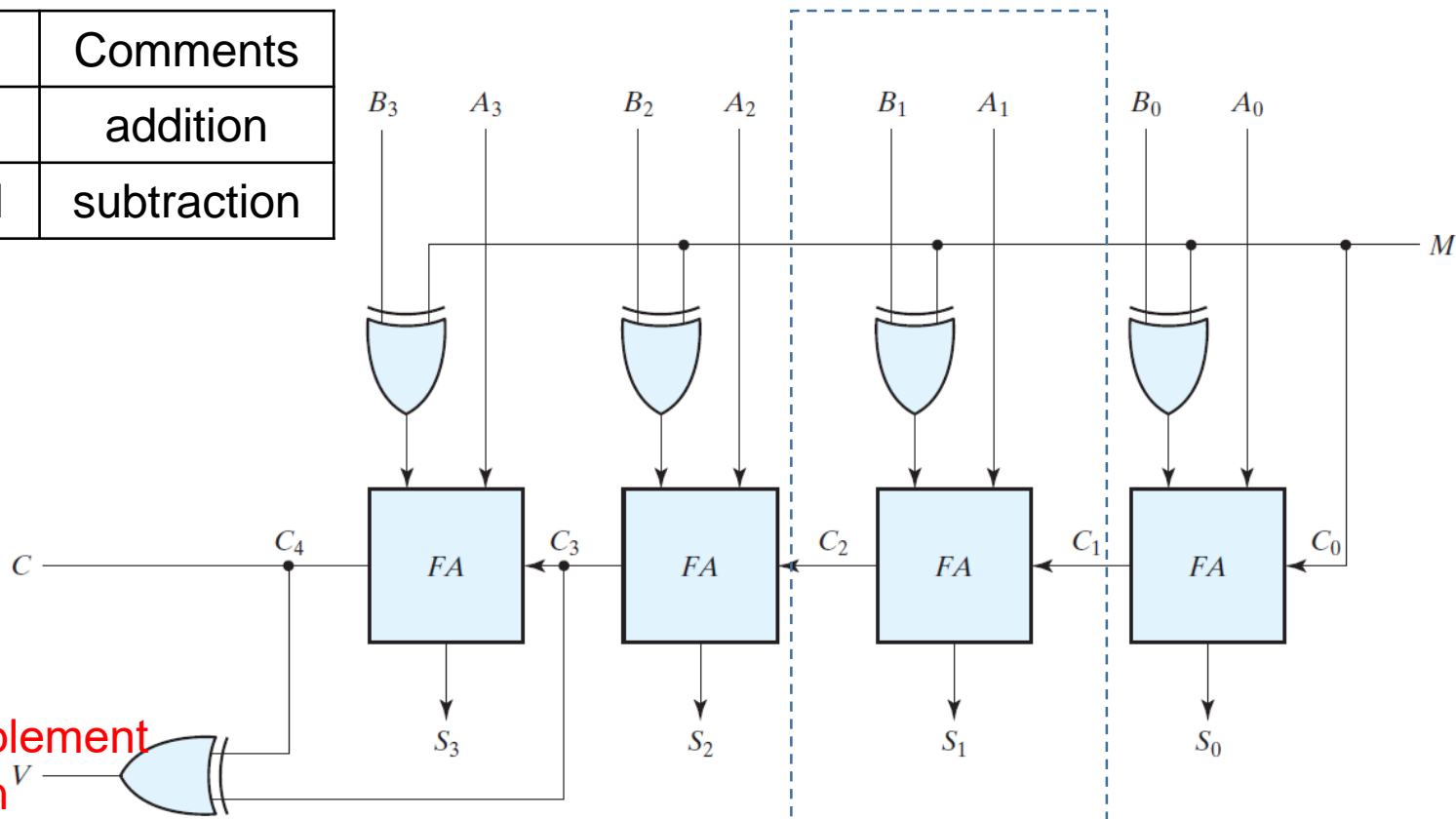
Overflow

- When n-digits addition with sum occupying n+1 digits, we say that an overflow (溢出) occurred.

- **Carry** for unsigned numbers:
  - Carry indicates unsigned number overflow

```
  1 1 1
  1 0 1 1   … 11
+ 0 1 1 1   … _7
1 0 0 1 0   … 18 (needs 5bits)
```

- **Overflow** for Signed numbers: (2's Complement)
  - two -ve numbers are added and the obtained result is +ve
  - two +ve numbers are added and the obtained result is –ve

```
  1 0 0 1
    1 0 0 1   … –7
+   1 1 0 1   … –3
  1 0 1 1 0   …–10
          6
```
4 bits can not correctly represent -10

```
  0 1 1 1
    0 1 1 1   … 7
+   0 0 0 1   … 1
    1 0 0 0   … 8
          -8
```
4 bits can not correctly represent +8

Binary Adders/Subtractors

- Overflow happens when A and B are 2's complement signed value
- Example: In each case, determine the values of the four SUM outputs, the carry C , and overflow V
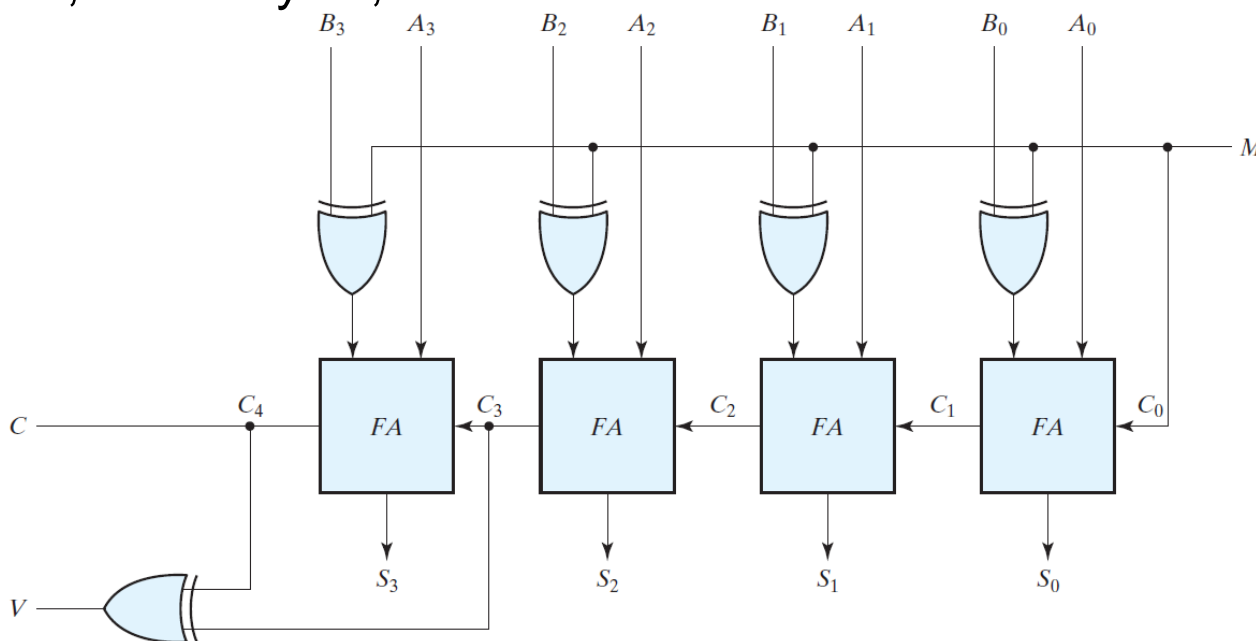
M	A	B	SUM	C	V
0	0111	0110	1101	0	1
1	1100	1000	0100	1	0

Serial Adder

- Initially, augend is in register A and addend is in register B

- Shift control enables/disable the clock for FF

- addition of two operands from LSB to MSB

- A new sum (S) bit is transferred to shift register A

- A carry-out (C) of the FA is transferred to Q as the z input of the next addition

- Finally, when the shift control is disabled, summation result is stored in shift register A
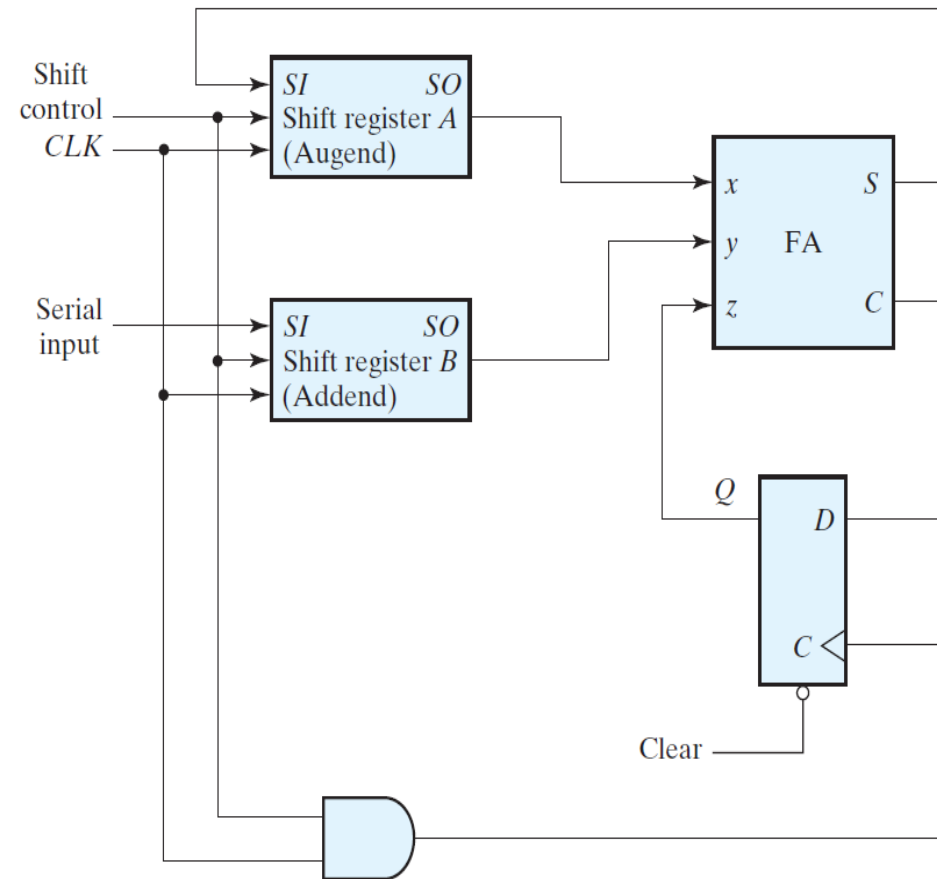
Timing Sequence of Serial Adder

- Serial addition of 0101 + 0111
  - Register A(Store Augend and Sum): 0101
  - Register B(Store Addend): 0111
  - More cycles required to initialize Register A and B

$$
\begin{array}{r}
1110 \\
0101 \\
+ \quad 0111 \\
\hline
1100
\end{array}
$$
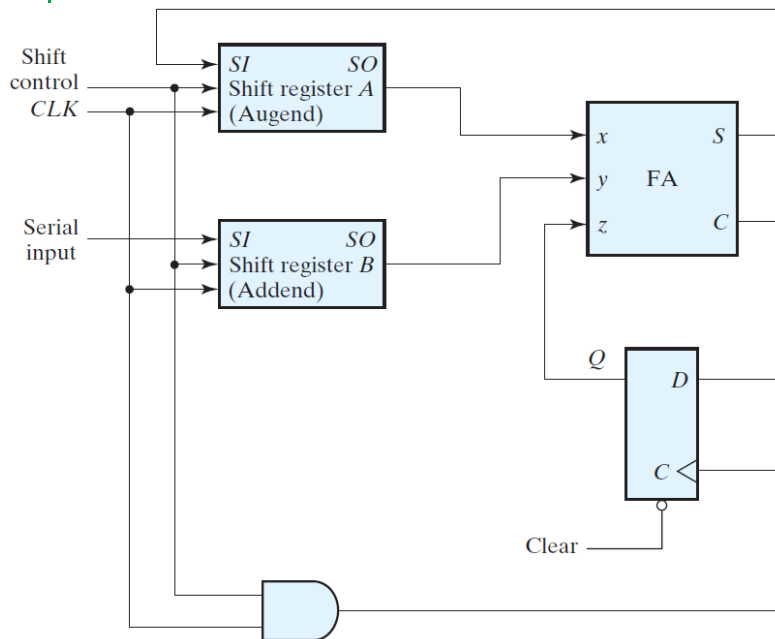
T	Register A	Register B	C	S
0	0101	0111	0	0
1	0010	0011	1	0
2	0001	0001	1	1
3	1000	0000	1	1
4	1100	0000	0	0

sum

Outline

- Binary Adder
- Binary Subtractor
- **Decimal Adder (BCD)**
- Binary Multiplier
- Other Arithmetic Functions

Decimal Adders

- Addition of 2 decimal digits in BCD
- $\{C_{out}, S\} = A + B + C_{in}$
- $S = S_8 S_4 S_2 S_1$, $A = A_8 A_4 A_2 A_1$, $B = B_8 B_4 B_2 B_1$
- A digit in BCD cannot exceed 9, add 6 (0110) for final correction.

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

10		1	0	0	0	0
8_{10}	A		1	0	0	0_2
9_{10}	B		1	0	0	1_2
	KZ	1	0	0	0	1_2	binary coded results
			0	1	1	0_2	if > 9, add 6
17_{10}	CS 0 0 0 1	0	1	1	1_2	BCD coded result

K: binary carry, Z: binary sum, C: BCD carry, S: BCD sum

Decimal Adders

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

$Z_8Z_4+Z_8Z_2$

K

- C = 1 when
  1. K=1
  2. or K = 0, but A+B > 9, which is $Z_8Z_4 + Z_8Z_2$
- C = 1 means need to add 0110

Z_8Z_4 \ Z_2Z_1	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

Decimal Adders

- $C_{out} = K + Z_8 Z_4 + Z_8 Z_2$
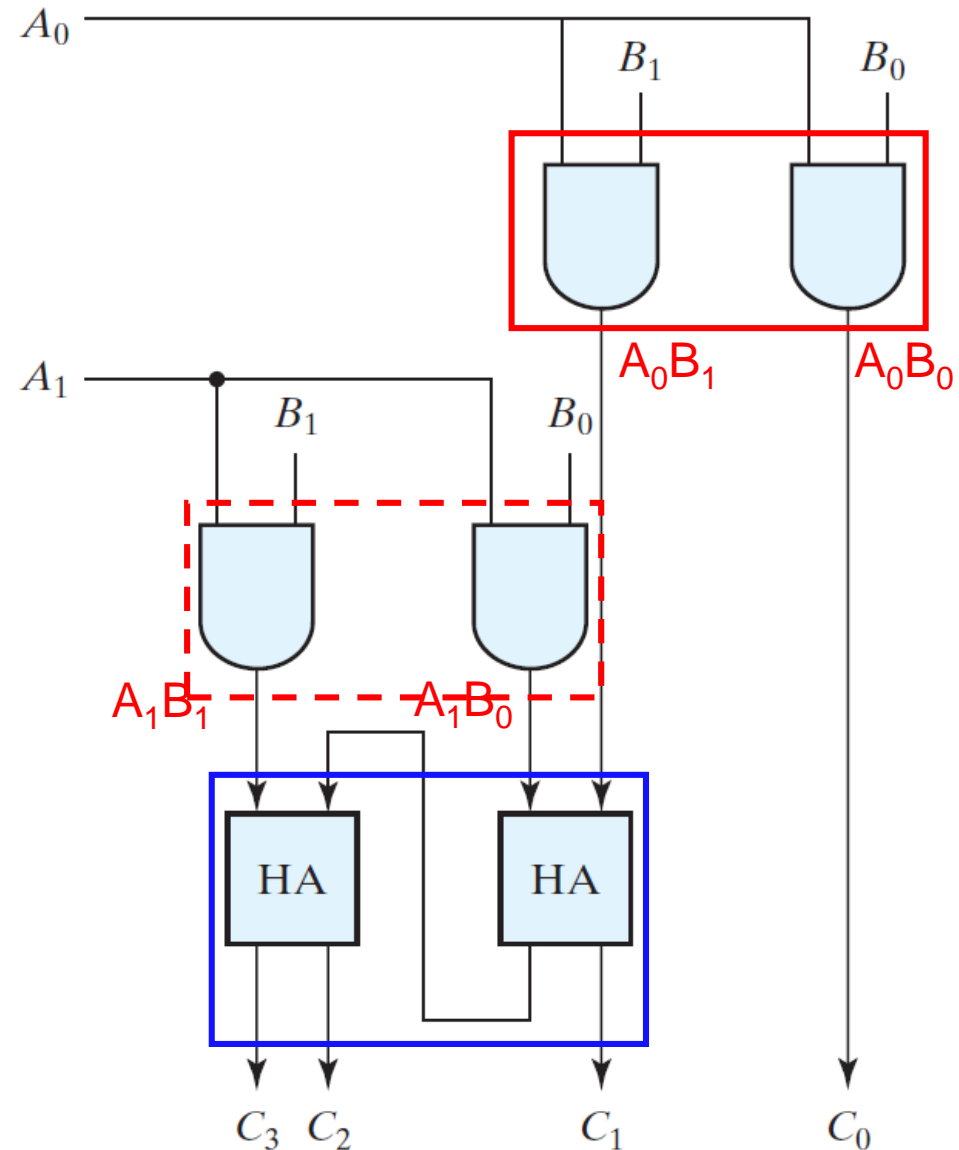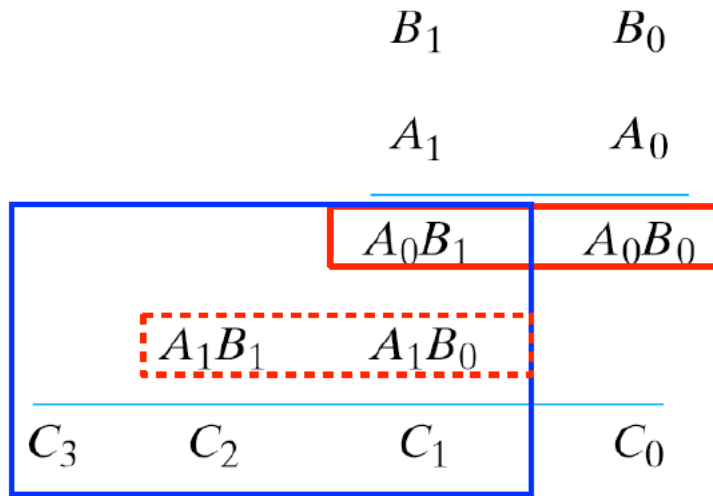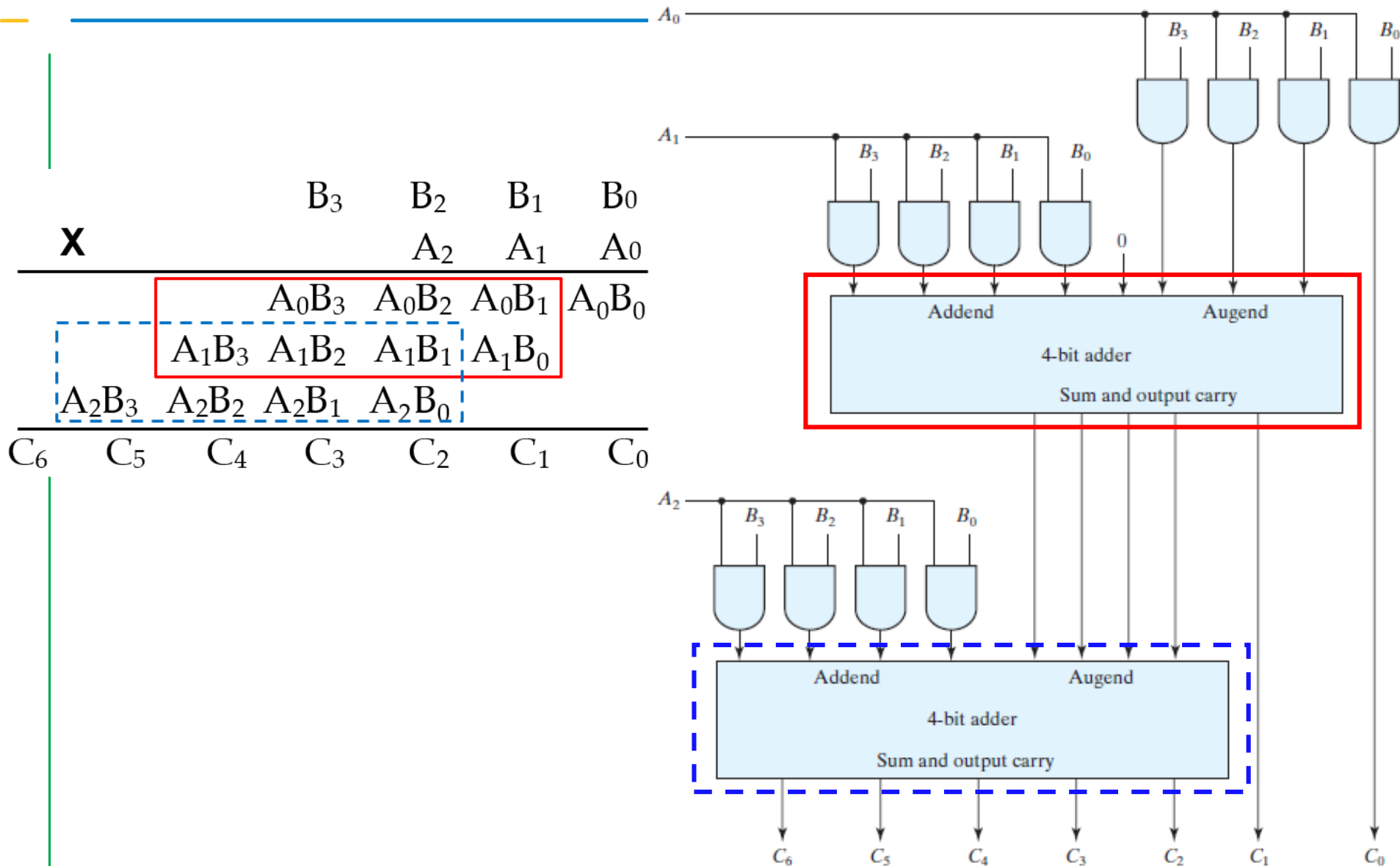- When C = 1, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage.
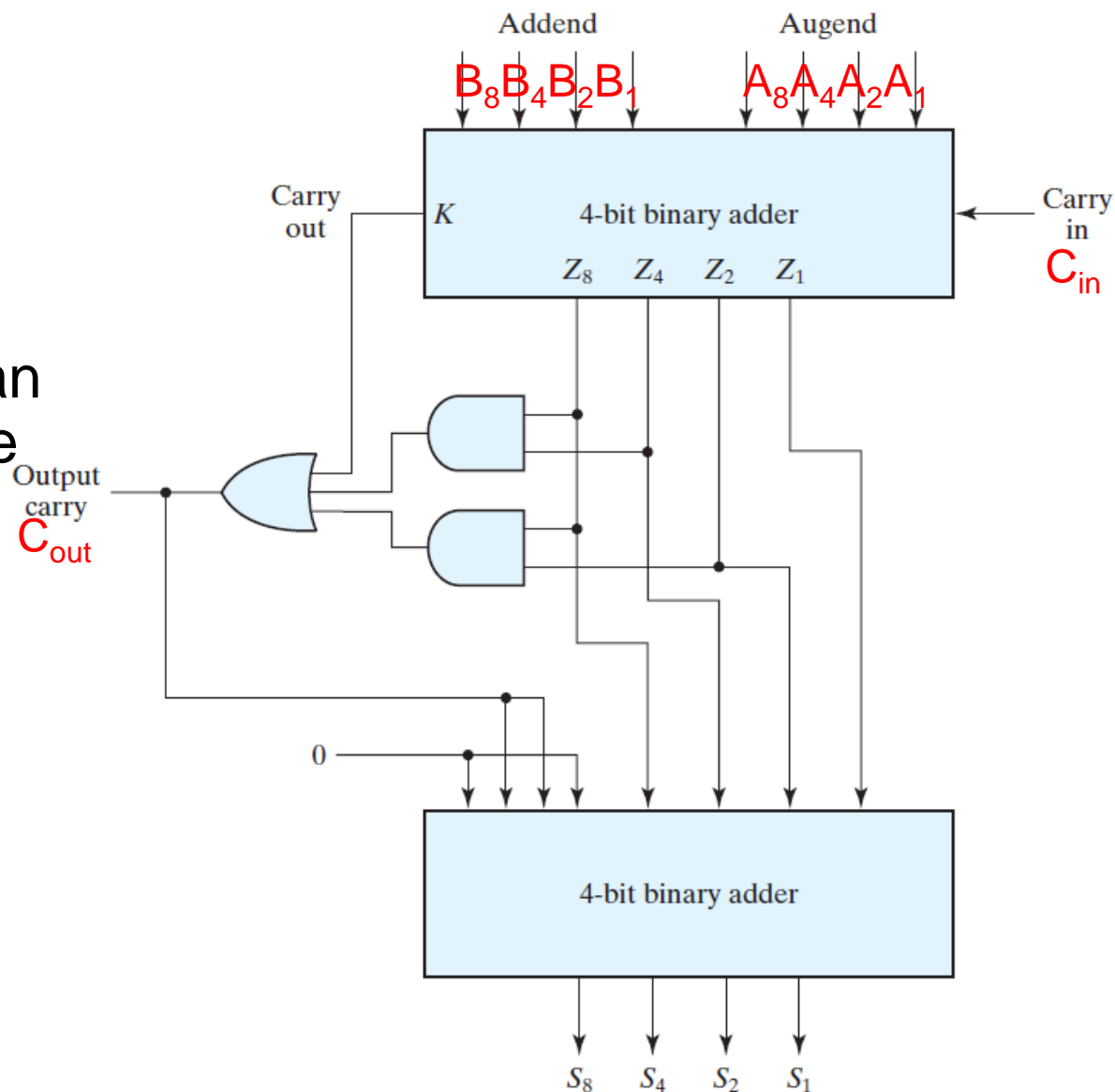
Outline

- Binary Adder
- Binary Subtractor
- Decimal Adder (BCD)
- **Binary Multiplier**
- Other Arithmetic Functions

Binary Multiplier

- Multiplication consists of
  - Generation of partial products
  - Accumulation of shifted partial products
- Binary multiplication equivalent to AND operation

$$
\begin{array}{r}
\mathbf{1\ 0\ 1\ 1\ 1} \quad \text{multiplicand} \\
\mathbf{x} \qquad \mathbf{1\ 0\ 1\ 0} \quad \text{multiplier} \\
\hline
\mathbf{0\ 0\ 0\ 0\ 0} \\
\mathbf{1\ 0\ 1\ 1\ 1} \\
\mathbf{0\ 0\ 0\ 0\ 0} \\
\mathbf{1\ 0\ 1\ 1\ 1} \\
\hline
\mathbf{1\ 1\ 1\ 0\ 0\ 1\ 1\ 0}
\end{array}
$$

Partial Product

$$
\begin{array}{r}
23 \\
x \quad 10 \\
\hline
230
\end{array}
$$

2-bit x 2-bit Binary Multiplier

4-bit x 3-bit Binary Multiplier

$$\begin{array}{ccccccc} & & B_3 & B_2 & B_1 & B_0 \\ X & & & A_2 & A_1 & A_0 \\ \hline & & A_0B_3 & A_0B_2 & A_0B_1 & A_0B_0 \\ & A_1B_3 & A_1B_2 & A_1B_1 & A_1B_0 \\ A_2B_3 & A_2B_2 & A_2B_1 & A_2B_0 \\ \hline C_6 & C_5 & C_4 & C_3 & C_2 & C_1 & C_0 \end{array}$$

Outline

- Binary Adder
- Binary Subtractor
- Decimal Adder (BCD)
- Binary Multiplier
- **Other Arithmetic Functions (optional)**

Other Arithmetic Functions

- It is convenient to design the functional blocks by **contraction**
  - Removal of redundancy from circuit to which input fixing has been applied
- Functions
  - Increment
  - Decrement
  - Multiplication by constant
  - Division by constant
 - Zero fill and extension

Design by Contraction

- Simplify the logic in a functional block to implement a different function
  - The new function must be realizable from the original function by applying basic functions to its inputs
  - Contraction is treated here only for application of 0s and 1s (not for X and X')
  - After application of 0s and 1s, equations or the logic diagram are simplified

Design by Contraction Example

- Contraction of a ripple carry adder to incrementer for n=1 $(A_2 A_1 A_0 + 001)$
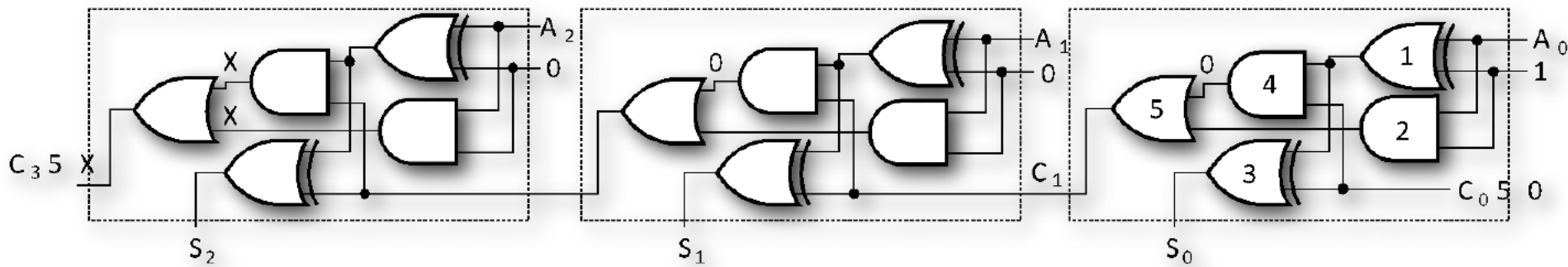

(a)
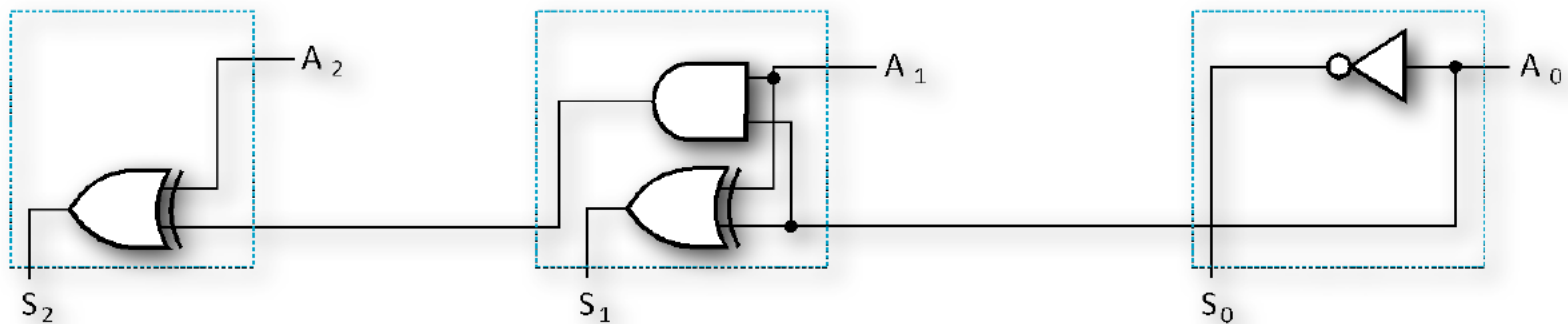
Incrementing and Decrementing

- Incrementing
  - Add a fixed value to an arithmetic variable
  - Fixed value is often 1, called counting up
    - A+1, B+4
  - Functional block is called incrementer
- Decrementing
  - Subtracting a fixed value from an arithmetic variable
  - Fixed value is often 1, called counting down
    - A-1, B-4
  - Functional block is called decrementer

Multiplication/Division by 2^n
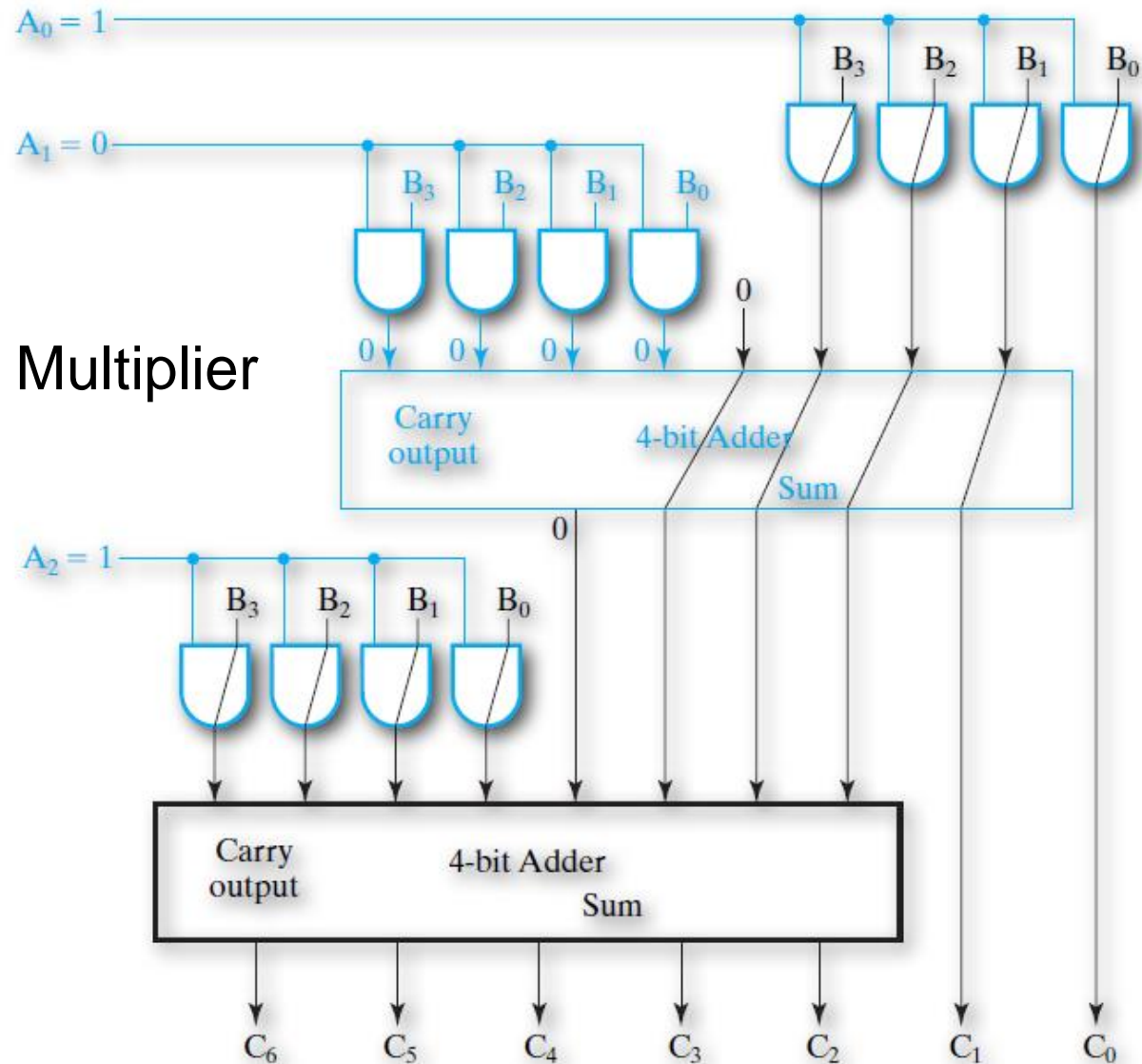
- Shift left (multiplication) or right (division)
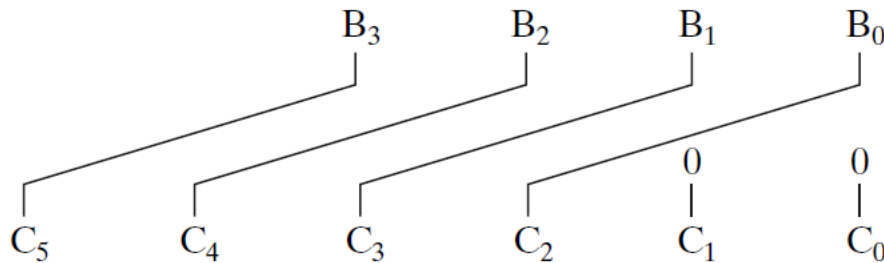
shift left by 2
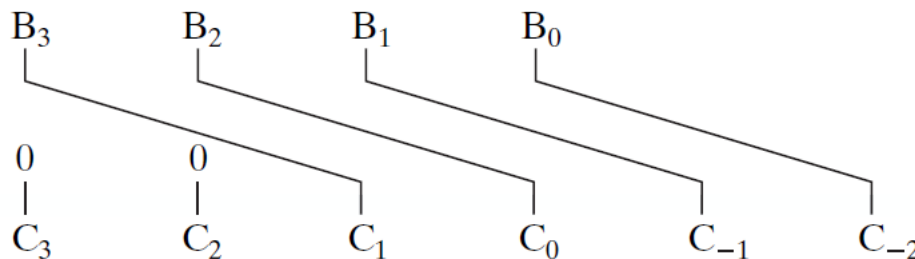
shift right by 2

Multiplication by a Constant

- 4-bit x 3-bit Binary Multiplier
- $B_3 B_2 B_1 B_0$ x 101

Zero/Sign Extension

- Fill an m-bit operand with 0s to become an n-bit operand with n > m
  - Filling usually is applied to the MSB end of the operand
- Zero Extension
  - 01110101 filled to 16 bits
    - 0000000001110101   {{8{0}}01110101}
  - 11110101 filled to 16 bits
    - 0000000011110101   {{8{0}}11110101}
- Sign Extension
  - Copies the MSB of the operand into the new positions
  - 01110101 extended to 16 bits
    - 0000000001110101   {{8{a7}}a71110101}
  - 11110101 extended to 16 bits
    - 1111111111110101   {{8{a7}}a71110101}