EECS 1010 01
Combinational Logic

黃稚存

國立清華大學
資訊工程學系

Lecture 04-1

Outline

- Introduction
  - Combinational Circuits
  - Analysis Procedure
  - Design Procedure
- Binary Adder-Subtractor
- Decimal Adder
- Binary Multiplier
- Magnitude Comparator
- Decoder and Encoder
- Multiplexer
- Three-State Gates

Objectives

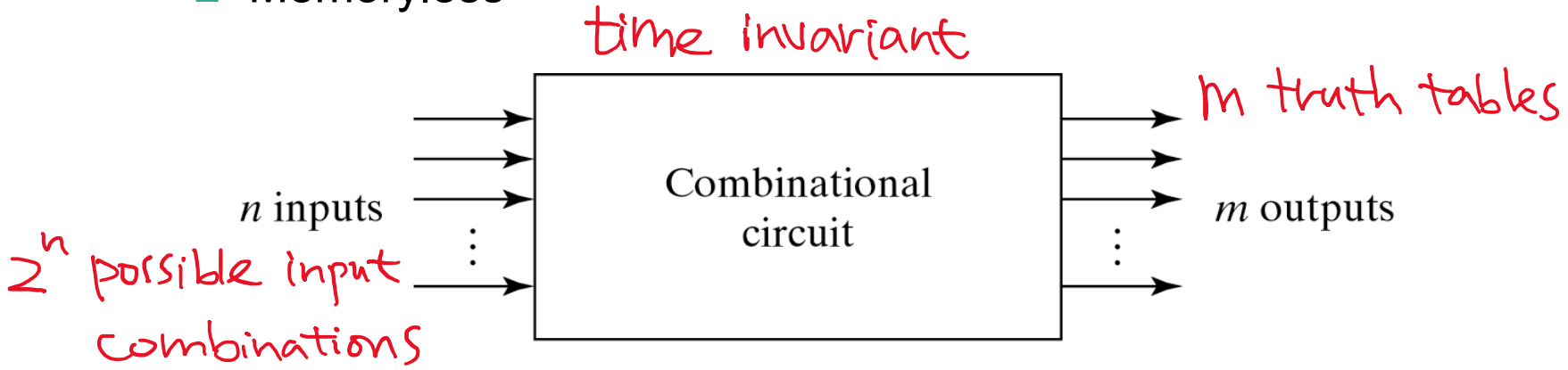
- Know how to analyze a combinational logic circuit, given its logic diagram
- Understand the half-adder and full-adder
- Understand the overflow and underflow
- Understand the implementation of a binary adder, BCD adder, and binary multiplier
- Understand fundamental combinational logic circuits: decoder, encoder, priority encoder, multiplexer, and three-state gate

Introduction

- Combinational Circuits
- Analysis Procedure
- Design Procedure

Logic Circuits for the Digital System

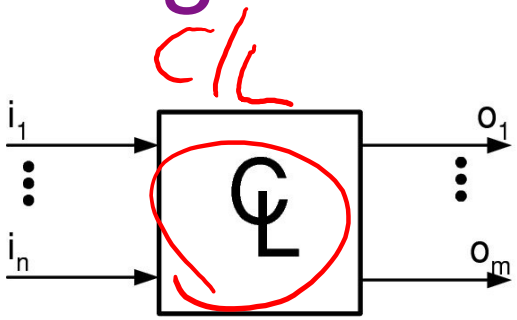
- Combinational circuits
  - Logic circuits whose outputs at any time are determined *directly* and *only* from the present input combination
    - Memoryless

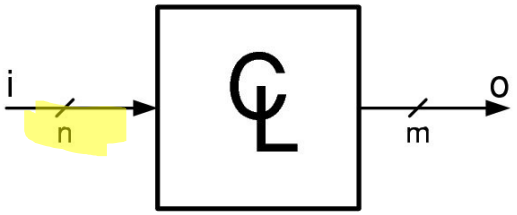


- Sequential circuits (Chapter 5)
  - Circuits that employ memory elements + (combinational) logic gates
  - Outputs are determined from the present input combination as well as the state of the memory cells

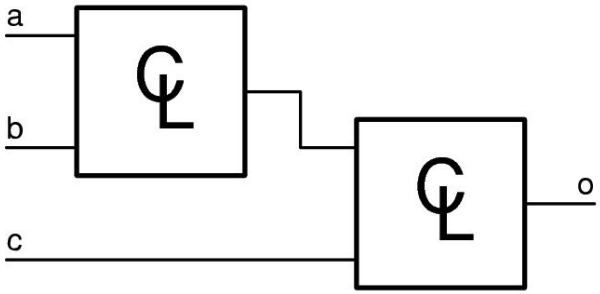
Combinational Logic Circuits

- Memoryless: o=f(i)
  - Used for control, arithmetic, and data steering

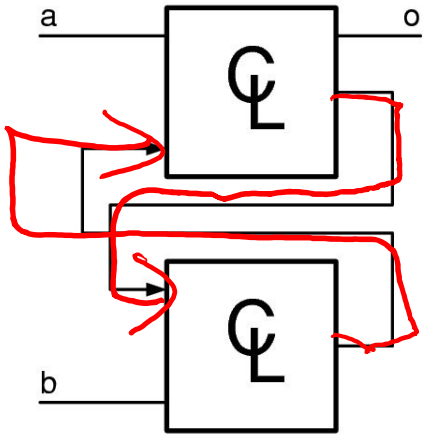




- Combinational logic circuits are closed under acyclic composition
- Cyclic composition of two combinational logic circuits
  - The feedback variable can remember the history of the circuits
  - Sequential logic circuit



Acyclic



Cyclic

Analysis Procedure

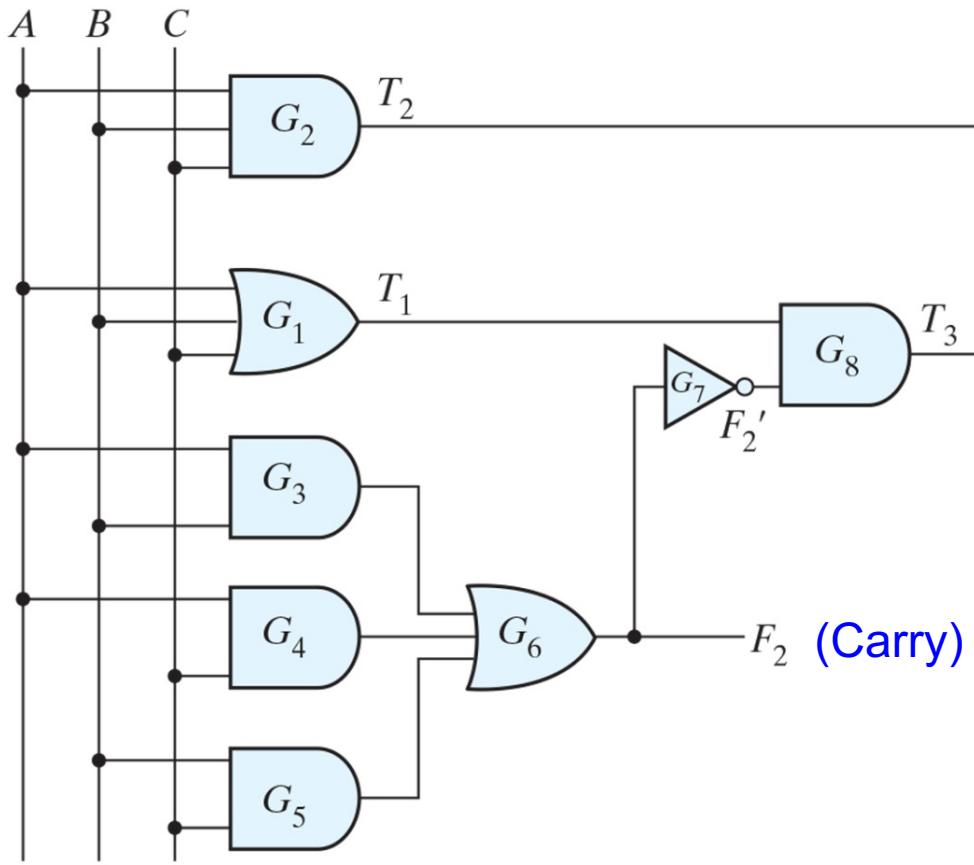
- Analysis for an available logic diagram
  - Make sure the given circuit is combinational
    - No *feedback path* or *memory element*
  - Derive the corresponding *Boolean functions*
  - Derive the corresponding *truth table*
  - Verify and analyze the design
    - Logic simulation (waveforms)
  - Explain the function

Derivation of Boolean Functions (1/2)

- Label all gate outputs that are functions of the input variables only. Determine the functions.
- Label all gate outputs that are functions of the input variables and previously labeled gate outputs. And find the functions.
- Repeat previous step until all the primary outputs are obtained.

Derivation of Boolean Functions (2/2)

- Full adder example:



- T_2=ABC
- T_1=A+B+C
- T_3=$F_2'T_1$
- F_2=AB+AC+BC (Carry)
- F_1=T_3+T_2 (Sum)
  =$F_2'T_1$+ABC
  =(AB+AC+BC)'(A+B+C)+ABC
  =A'BC'+A'B'C+AB'C'+ABC

A⊕B⊕C

Derivation of Truth Table (1/2)

- For *n* input variables
  - List all the 2^n input combinations from 0 to $2^n - 1$
  - Partition the circuit into small single-output blocks and label the output of each block
  - Obtain the truth table of the blocks depending on the input variables only
  - Proceed to obtain the truth tables for other blocks that depend on previously defined truth tables

Derivation of Truth Table (2/2)

- Full Adder Example

AB+BC+AC A+B+C ABC $F_2'T_1$ T_3+T_2

A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Design Procedure

① Specification

- From the specifications, determine the inputs, outputs, and their symbols

② Formulation

- Derive the truth tables (functions) from the relationship between the inputs and outputs

③ Optimization

- Derive the simplified Boolean functions for each output function

④ Technology mapping

- Derive the logic diagram based on the implementation technology

⑤ Verification

- Verify the design

Code Conversion Example (1/3)

- BCD-to-excess-3 code converter

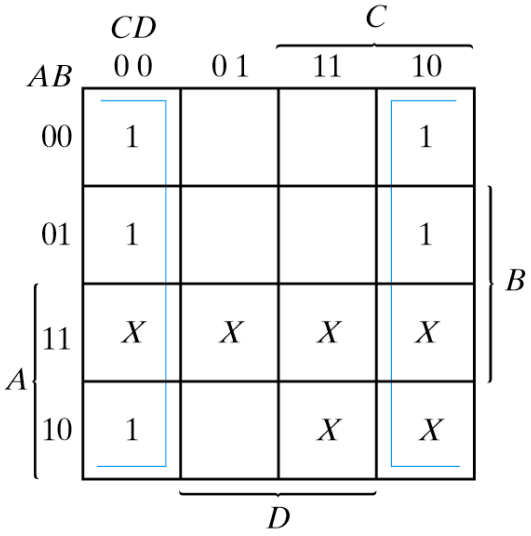
(1) Define the spec and IOs
(2) Derive truth table

① Spec
- input (ABCD), output (wxyz) (MSB to LSB)
- ABCD: 0000~1001 (0~9)

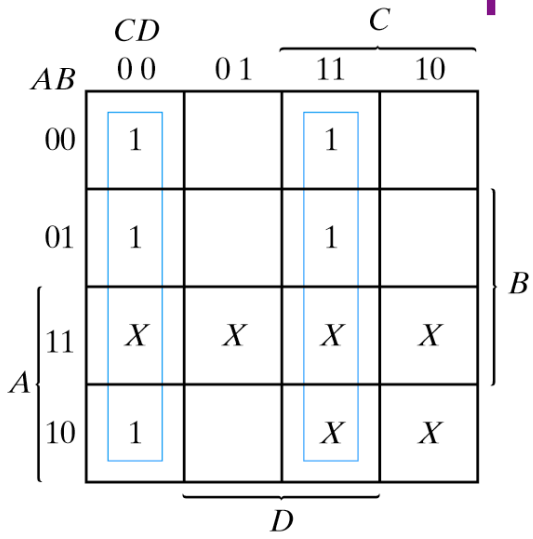
② Formulation
- wxyz = ABCD+0011

Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Code Conversion Example (2/3)



$z = D'$



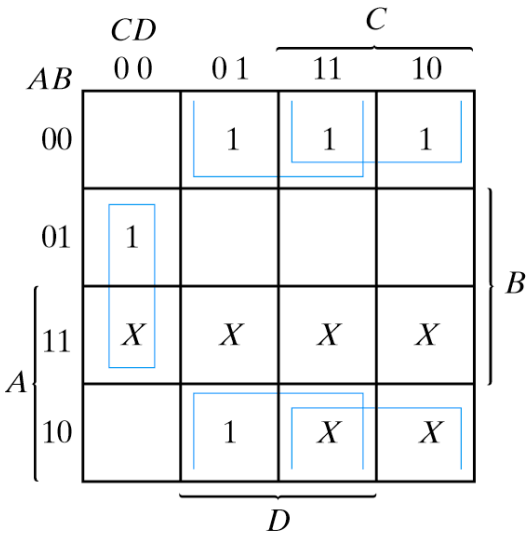
$y = CD + C'D'$

③ Optimization:

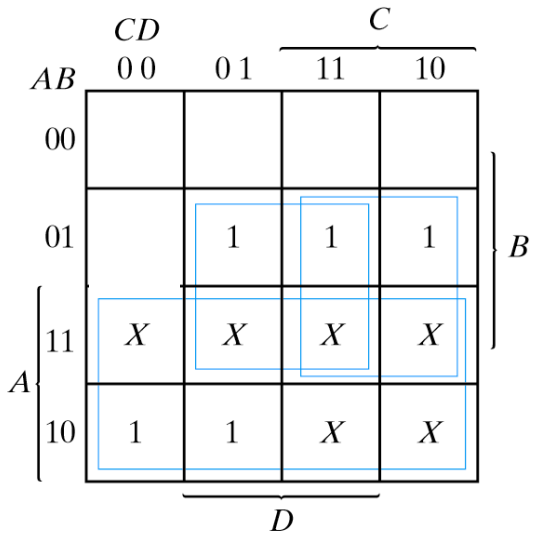
- Determine simplified Boolean function

- $z = D'$ $(C+D)'$
- $y = CD + C'D'$
- $x = B'C + B'D + BC'D'$
- $w = A + BC + BD$

7 ANDs, 3 ORs, 3 INVs



$X = B'C + B'D + BC'D'$



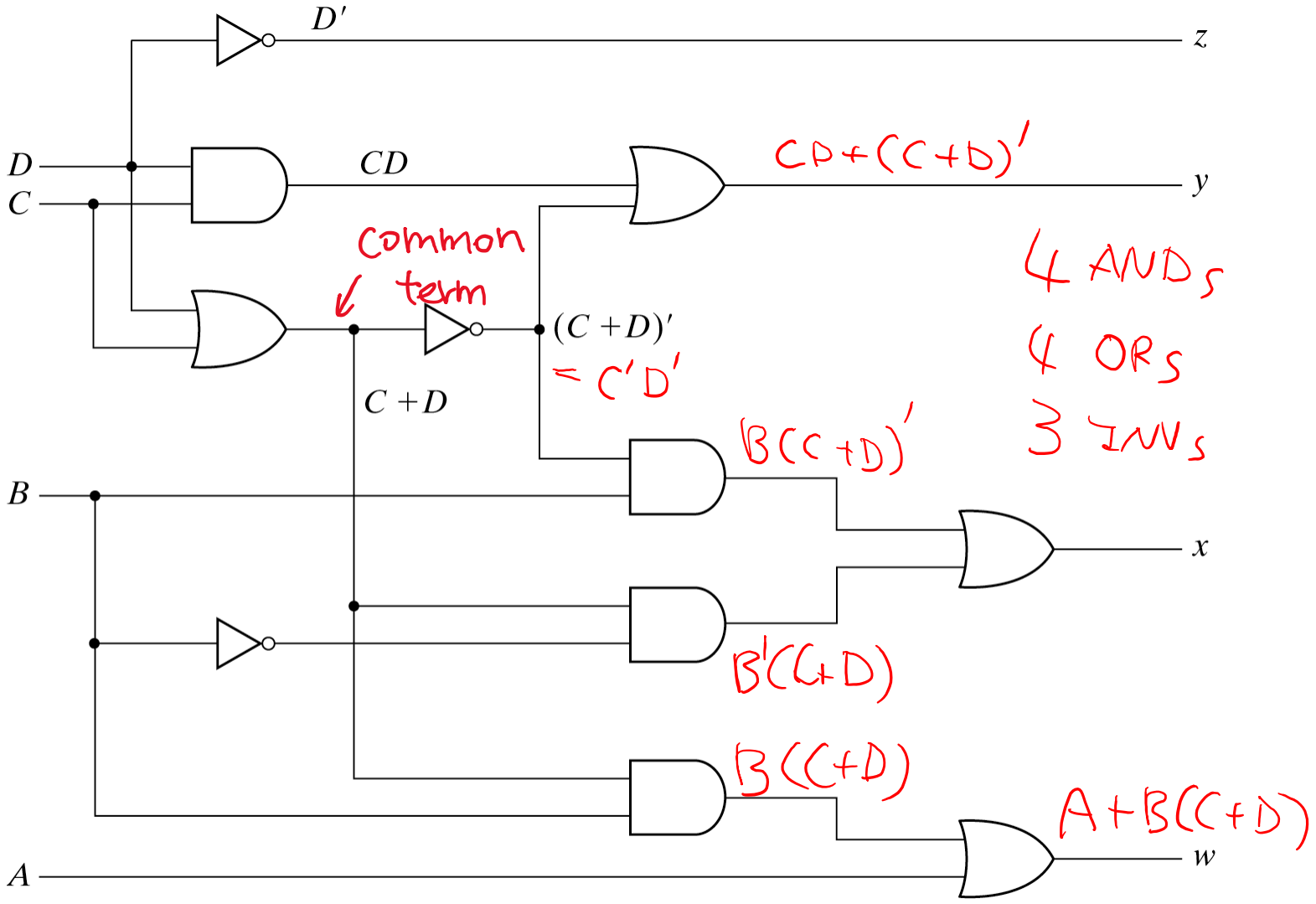
$w = A + BC + BD$

➔ multi-level, non-standard

- $z = D'$
- $y = CD + (C + D)'$
- $x = B'(C + D) + B(C + D)'$
- $w = A + B(C + D)$

Code Conversion Example (3/3)

④ Logic diagram



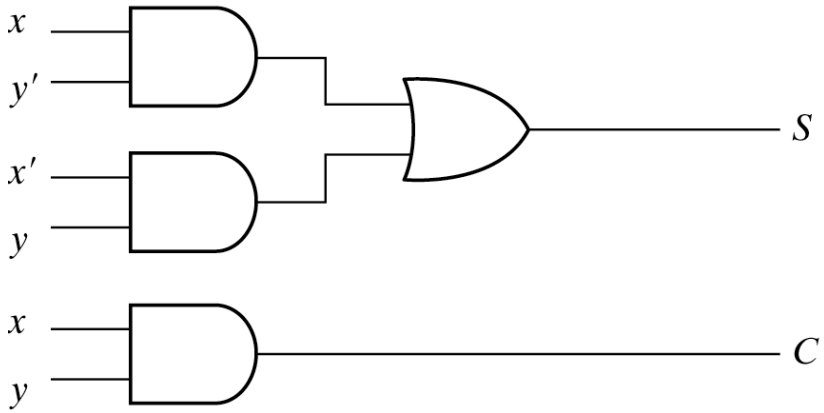
Binary Adder-Subtractor

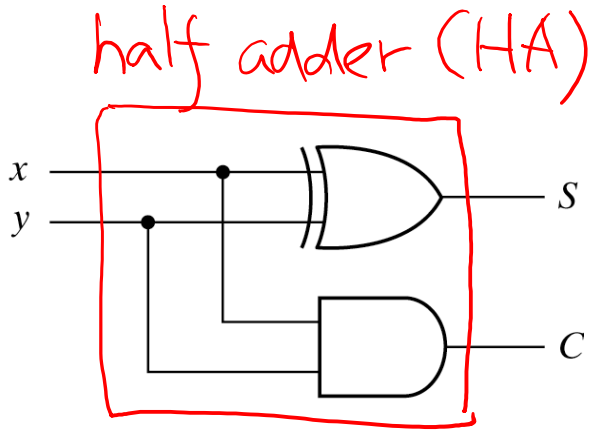
Binary Half Adder (HA)

$$\begin{array}{r} x \\ +\ y \\ \hline C\ S \end{array}$$

- IOs
  - Input: x, y
  - Outputs: C (carry), S (sum)
- Truth table and functions
  - $S = x'y + xy' = x \oplus y$
  - $C = xy$
- Logic diagram

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

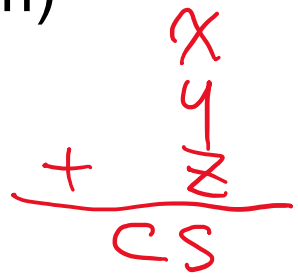




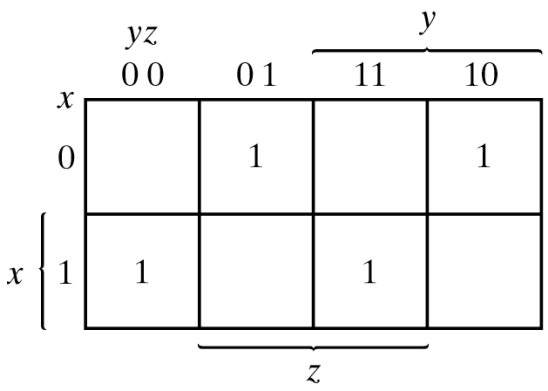
Binary Full Adder (FA) (1/3)

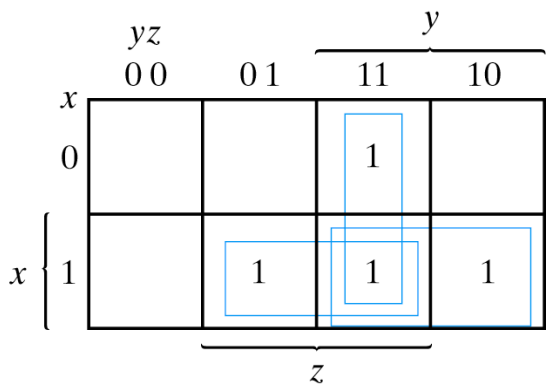
- IOs
  - Input: x, y, z (carry from previous lower significant bit)
  - Outputs: C (carry), S (sum)
- Truth table

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



$$C = \sum(3,5,6,7) = xy + yz + xz$$





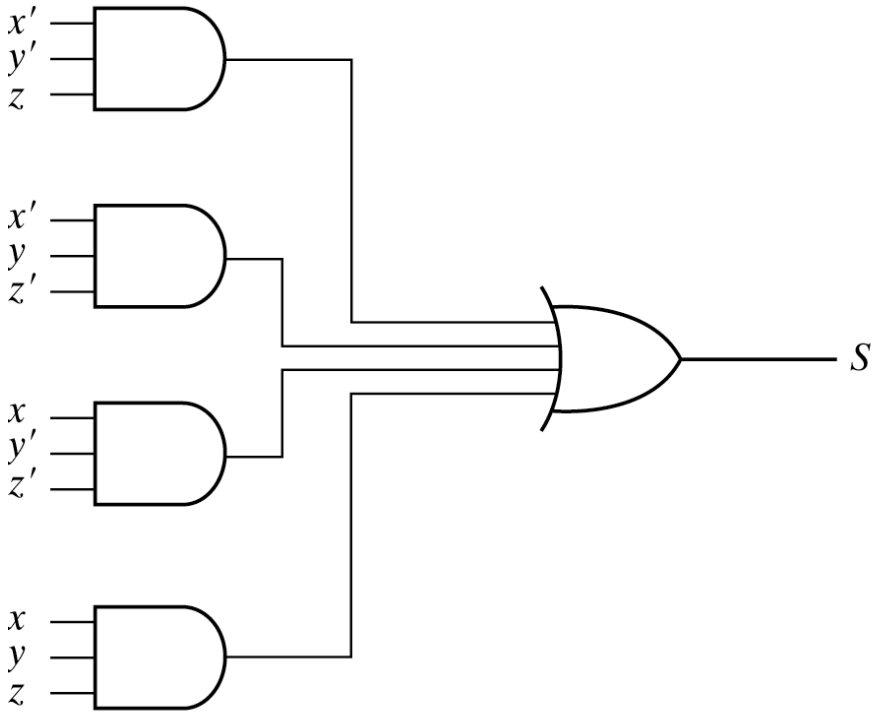
$$S = \sum(1,2,4,7) = x'y'z + x'yz' + xy'z' + xyz = x \oplus y \oplus z$$

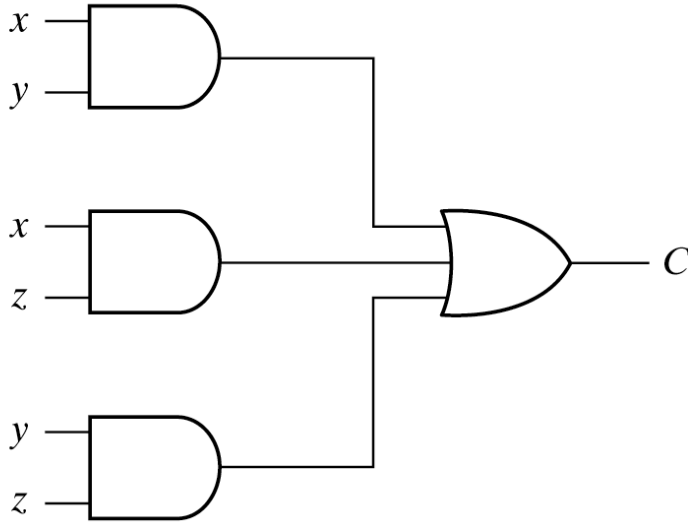
Binary Full Adder (FA) (2/3)

- 2-level logic diagram

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

$$C = xy + yz + xz$$





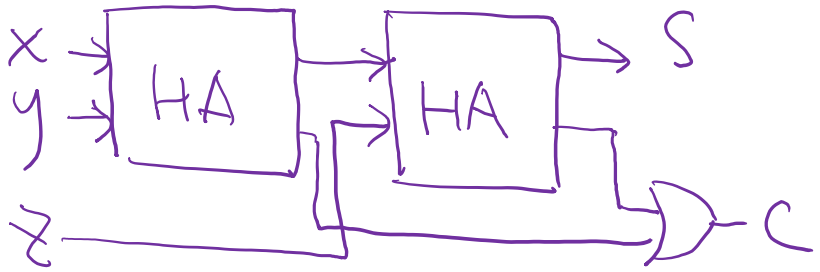
Binary Full Adder (FA) (3/3)

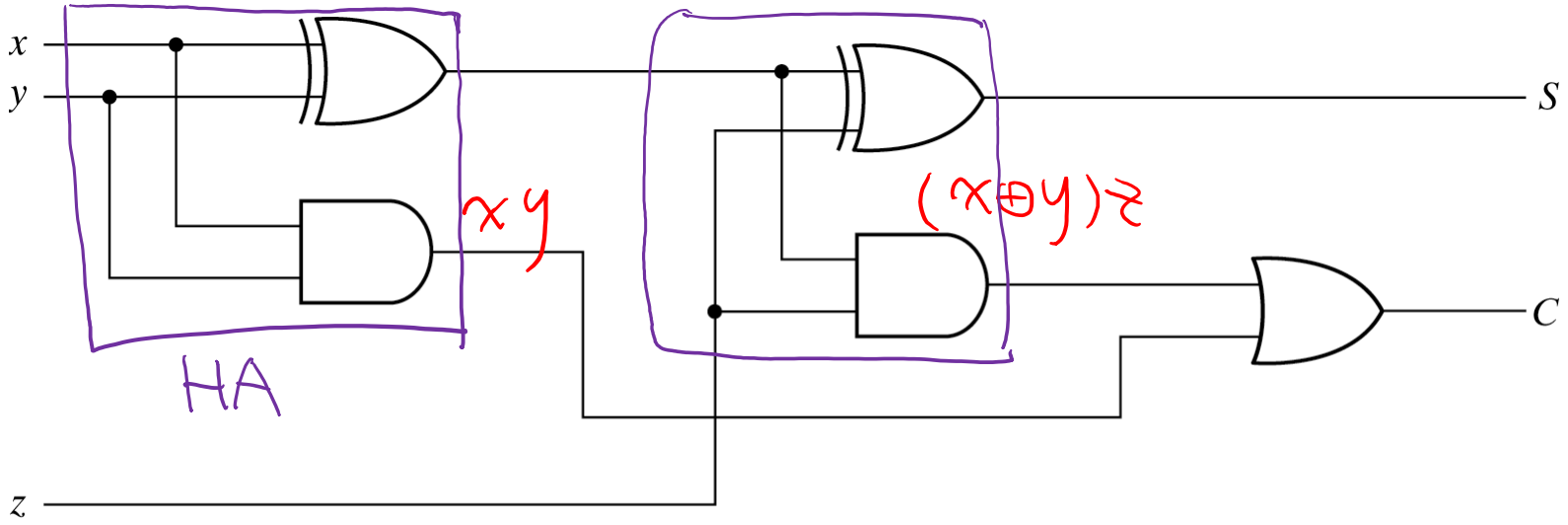
- Full adder implemented with half adders
  - Two half adders and one OR gate

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

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

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

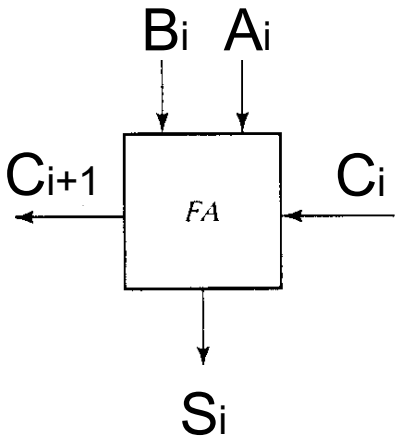


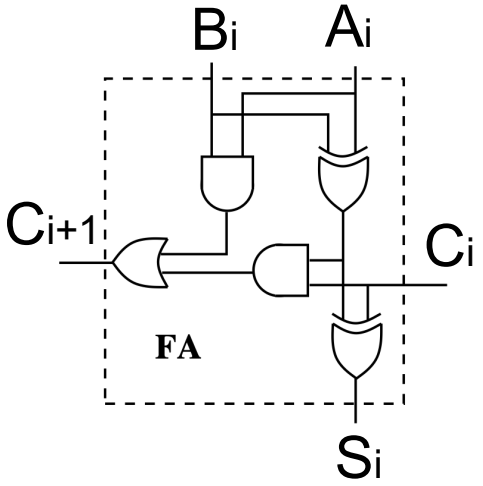


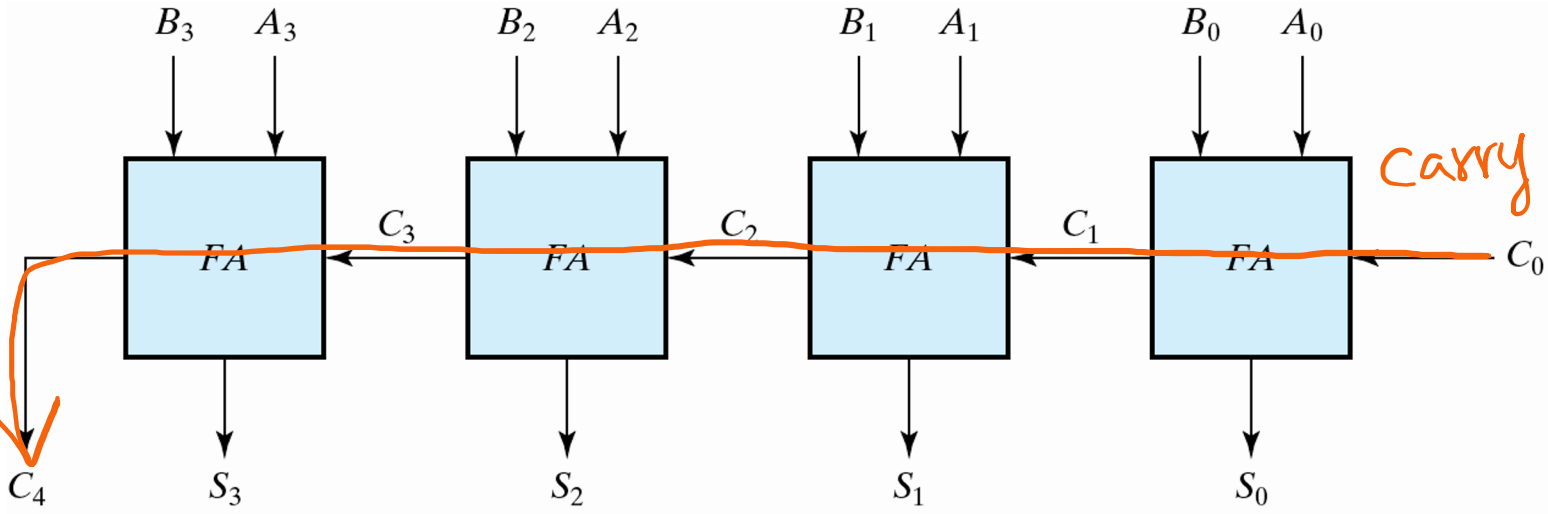
Binary (Ripple-Carry) Adder (1/2)

- Produces arithmetic sum of two binary numbers

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}

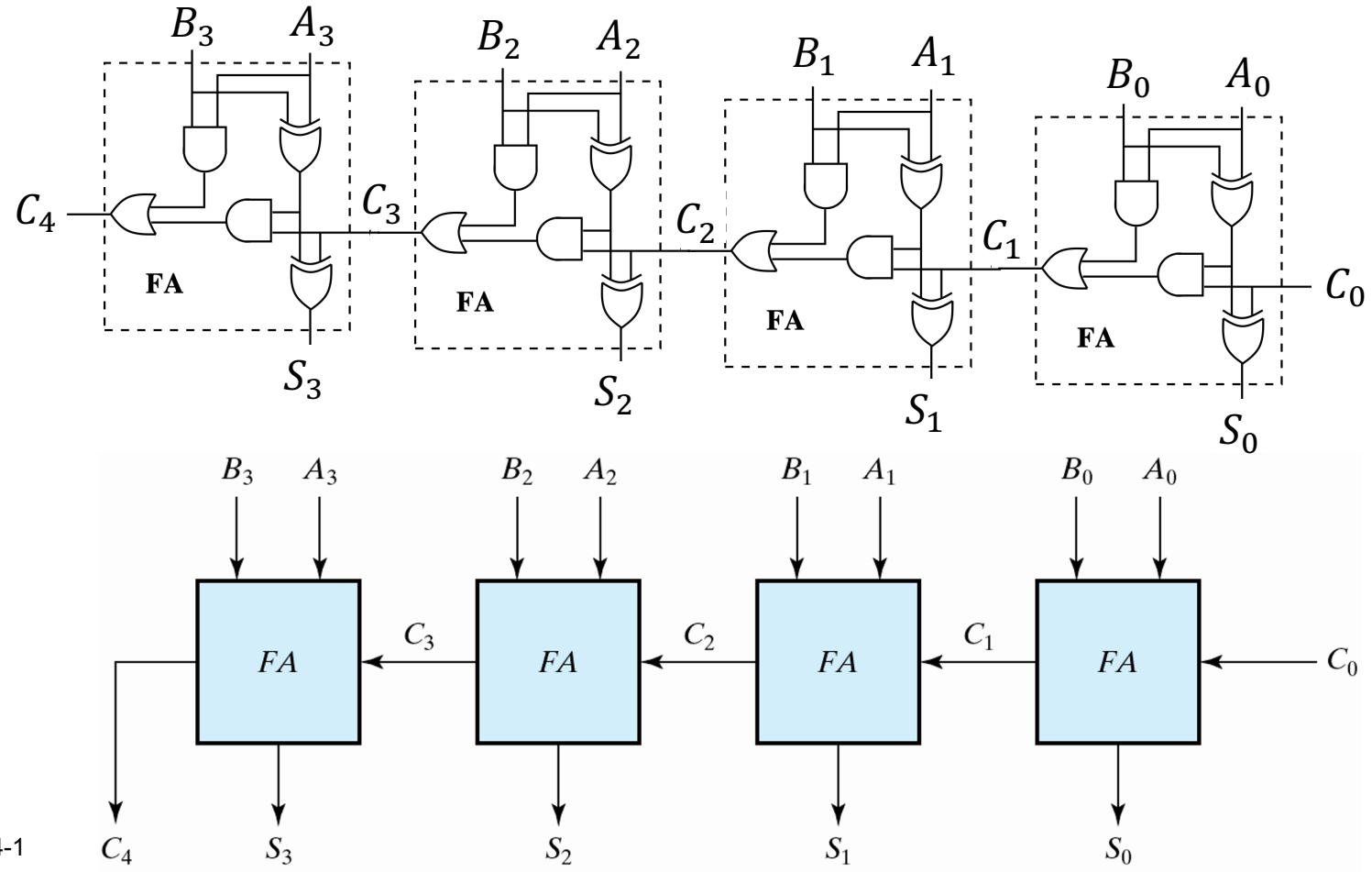






Binary (Ripple-Carry) Adder (2/2)

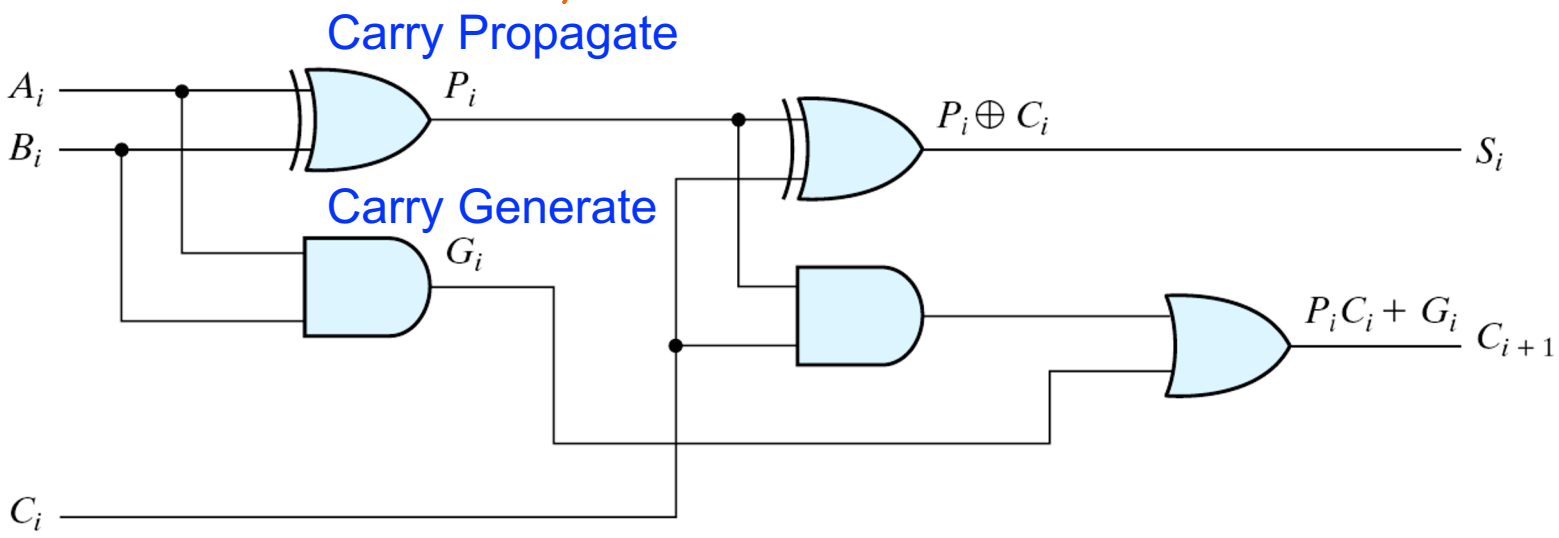
- The computation time of a ripple-carry adder grows linearly with word length *n*
  - T=O(n) due to carry chain



Carry Propagation

- For ripple carry adder
  - Longest propagation delay: $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow C_4$
  - 8 gate levels (for 4-bit adder)
  - 2*n* gate levels for the carry to propagate from input to output for an *n*-bit adder

2n gate delays ⇒ carry propagation



Carry Lookahead Adder (CLA) (1/5)

- Reduce the carry propagation delay
  - Using faster gates
  - Parallel adders, e.g., the *carry lookahead adder (CLA)*
- Carry Lookahead Adder (CLA)
  - 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: $C_{i+1} = G_i + P_i C_i$
  - $C_1 = G_0 + P_0 C_0$ $= A_0 B_0 + (A_0 \oplus B_0) C_0$
  - $C_2 = G_1 + P_1 C_1$ $= G_1 + P_1 (G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$
  - $C_3 = G_2 + P_2 C_2$ $= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$