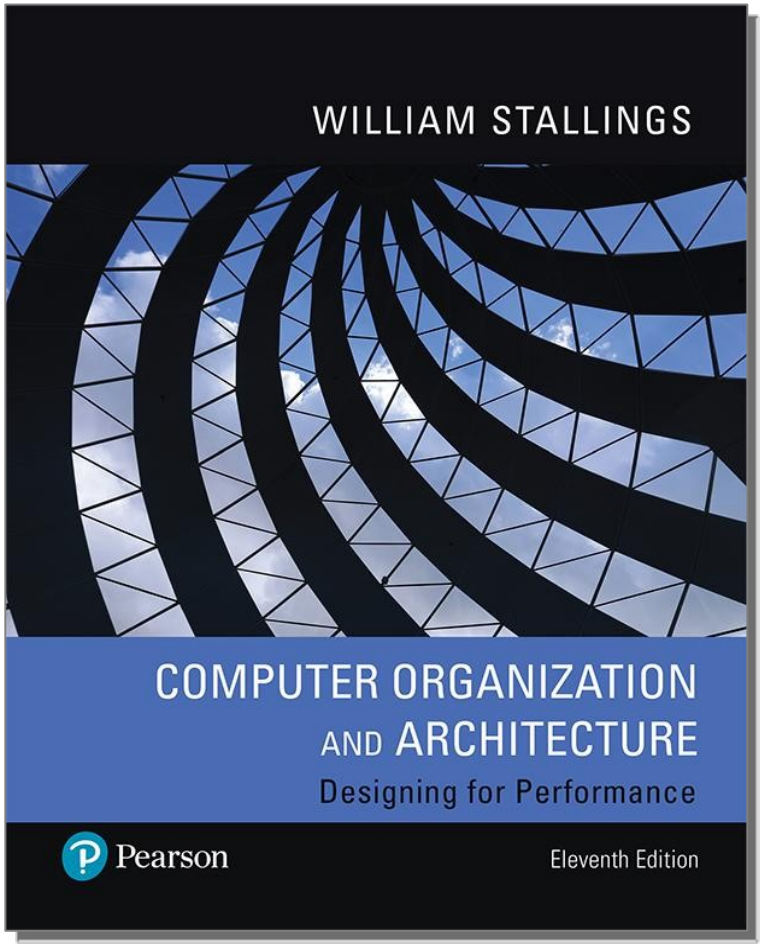
Computer Organization and Architecture
Designing for Performance

11th Edition



Chapter 12

Digital Logic

Boolean Algebra

- Mathematical discipline used to design and analyze the behavior of the digital circuitry in digital computers and other digital systems
- Named after George Boole
  - English mathematician
  - Proposed basic principles of the algebra in 1854
- Claude Shannon suggested Boolean algebra could be used to solve problems in relay-switching circuit design
- Is a convenient tool:
  - Analysis
    - It is an economical way of describing the function of digital circuitry
  - Design
    - Given a desired function, Boolean algebra can be applied to develop a simplified implementation of that function

Table 12.2
Correspondence Between Boolean Algebra and Operations on Sets

Boolean		Sets	
Function	**Description**	**Function**	**Description**
A AND B	1 if and only if A and B are 1	$A \cap B$	Set of elements that belong to both A and B (intersection)
A OR B	1 if A or B or both are 1; 0 if both A and B are 0	$A \cup B$	Set of elements that belong to A or B or both (union)
A OR B	1 if and only if A is 0	$\bar{A}$	Set of elements not in A (complement of A)

Boolean Variables and Operations

- Makes use of variables and operations
  - Are logical
  - A variable may take on the value 1 (TRUE) or 0 (FALSE)
  - Basic logical operations are AND, OR, and NOT
- AND
  - Yields true (binary value 1) if and only if both of its operands are true
  - In the absence of parentheses the AND operation takes precedence over the OR operation
  - When no ambiguity will occur the AND operation is represented by simple concatenation instead of the dot operator
- OR
  - Yields true if either or both of its operands are true
- NOT
  - Inverts the value of its operand

Table 12.1
Boolean Operators

(a) Boolean Operators of Two Input Variables

A	B	NOT A ($\overline{A}$)	A AND B ($A \cdot B$)	A OR B ($A + B$)	A NAND B ($\overline{A \cdot B}$)	A NOR B ($\overline{A + B}$)	A XOR B ($A \oplus B$)
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0

(b) Boolean Operators Extended to More than Two Inputs (A, B, . . .)

Operation	Expression	Output = 1 if
AND	$A \cdot B \cdot \ldots$	All of the set {A, B, …} are 1.
OR	$\overline{A + B + \ldots}$	Any of the set {A, B, …} are 1.
NAND	$\overline{A \cdot B \cdot \ldots}$	Any of the set {A, B, …} are 0.
NOR	$A + B + \ldots$	All of the set {A, B, …} are 0.
XOR	$A \oplus B \oplus \ldots$	The set {A, B, …} contains an odd number of ones.

Figure 12.1
Basic Boolean Functions of Two Variables

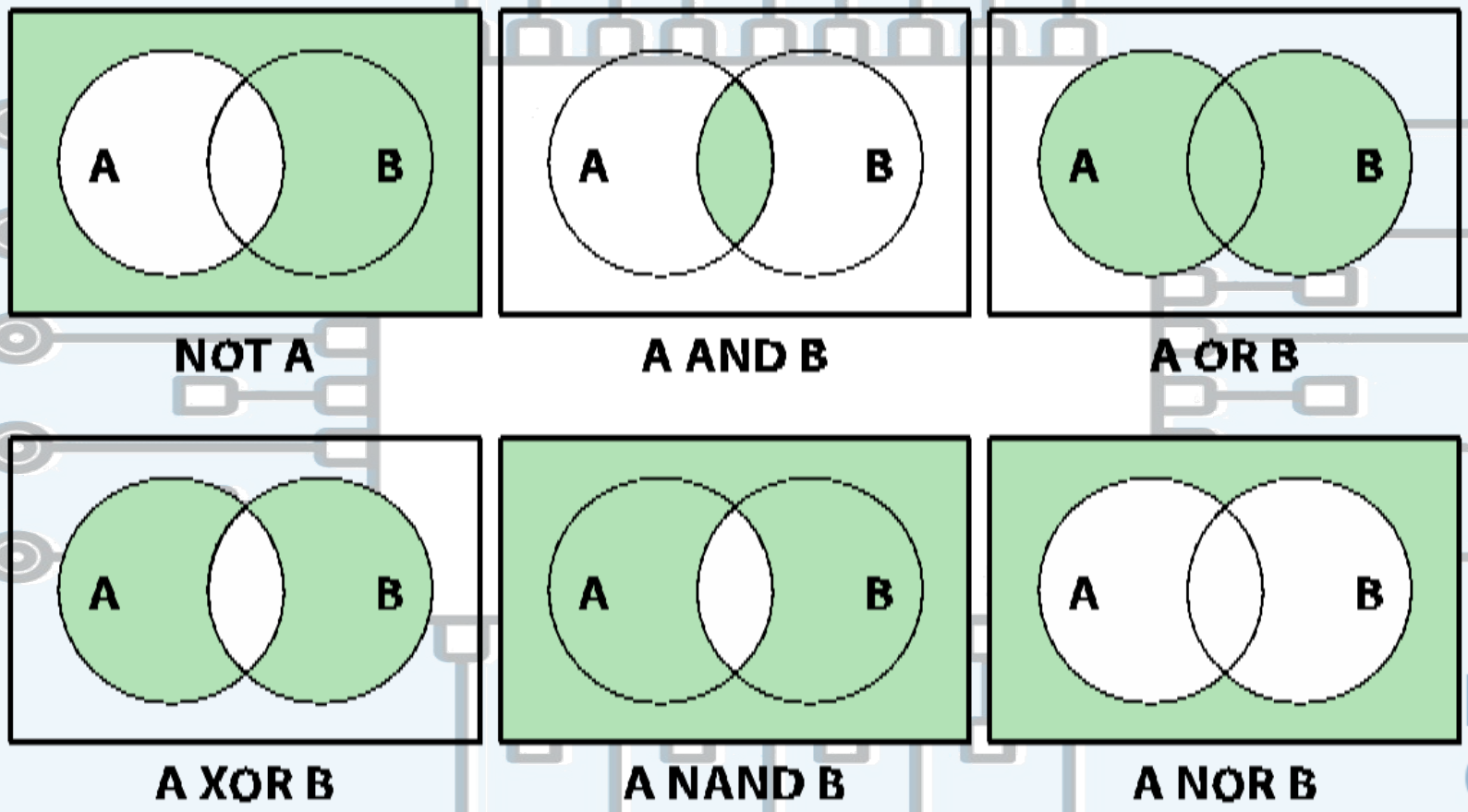


Figure 12.2
Venn Diagram for Three Boolean Variables

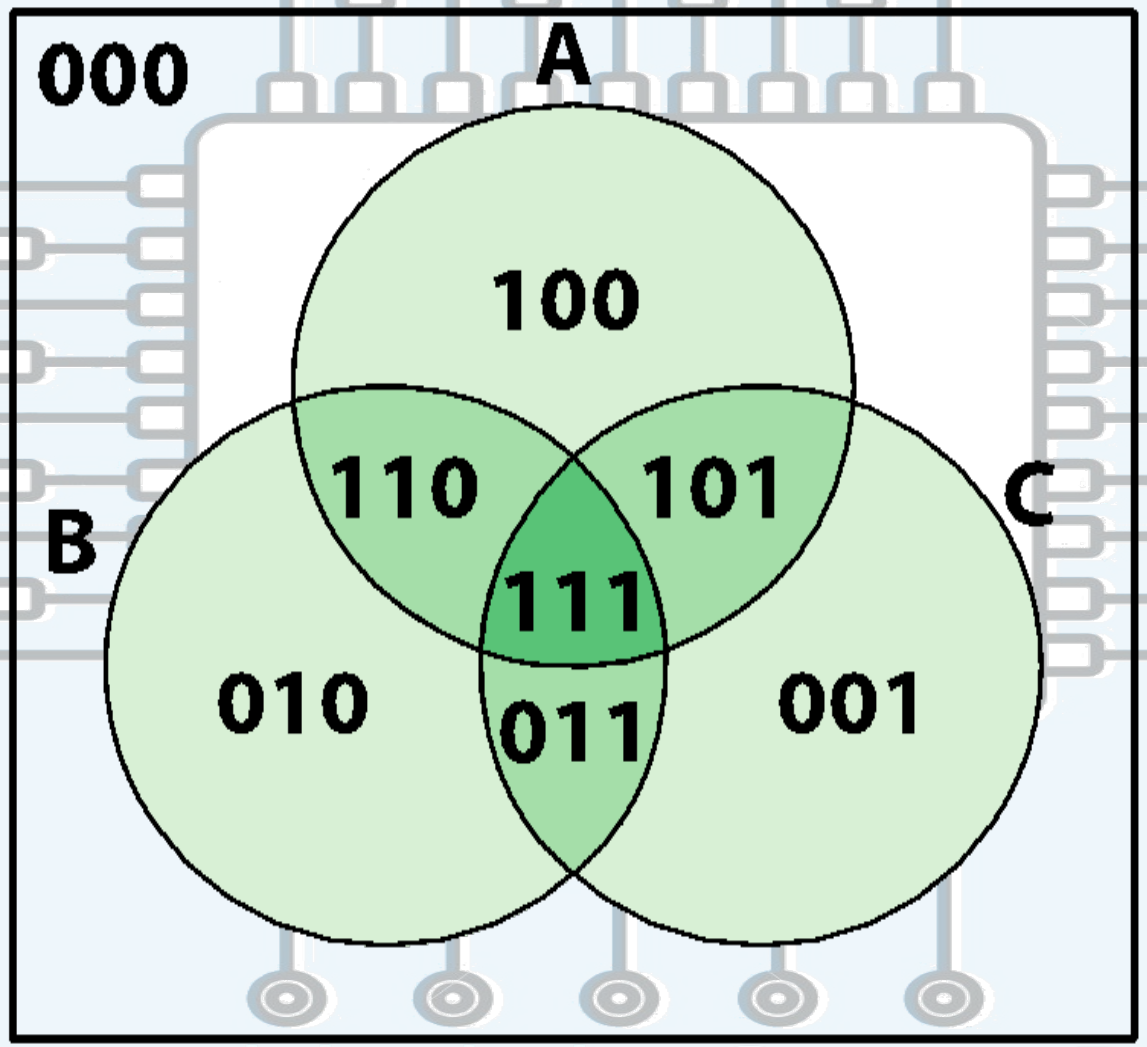


Table 12.3
Basic Identities of Boolean Algebra

Basic Postulates		
		Commutative Laws Distributive Laws Identity Elements Inverse Elements
Other Identities		
		Associative Laws DeMorgan's Theorem

Figure 12.3
Basic Logic Gates

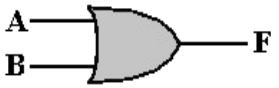
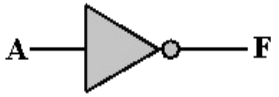
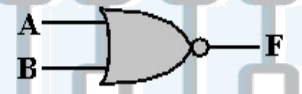
Name	Graphical Symbol	Algebraic Function	Truth Table
AND		$F = A \cdot B$ or $F = AB$	A B F 0 0 0 0 1 0 1 0 0 1 1 1
OR		$F = A + B$	A B F 0 0 0 0 1 1 1 0 1 1 1 1
NOT		$F = \overline{A}$ or $F = A'$	A F 0 1 1 0
NAND		$F = \overline{AB}$	A B F 0 0 1 0 1 1 1 0 1 1 1 0
NOR		$F = \overline{A + B}$	A B F 0 0 1 0 1 0 1 0 0 1 1 0
XOR		$F = A \oplus B$	A B F 0 0 0 0 1 1 1 0 1 1 1 0

Figure 12.4
Some Uses of NAND Gates

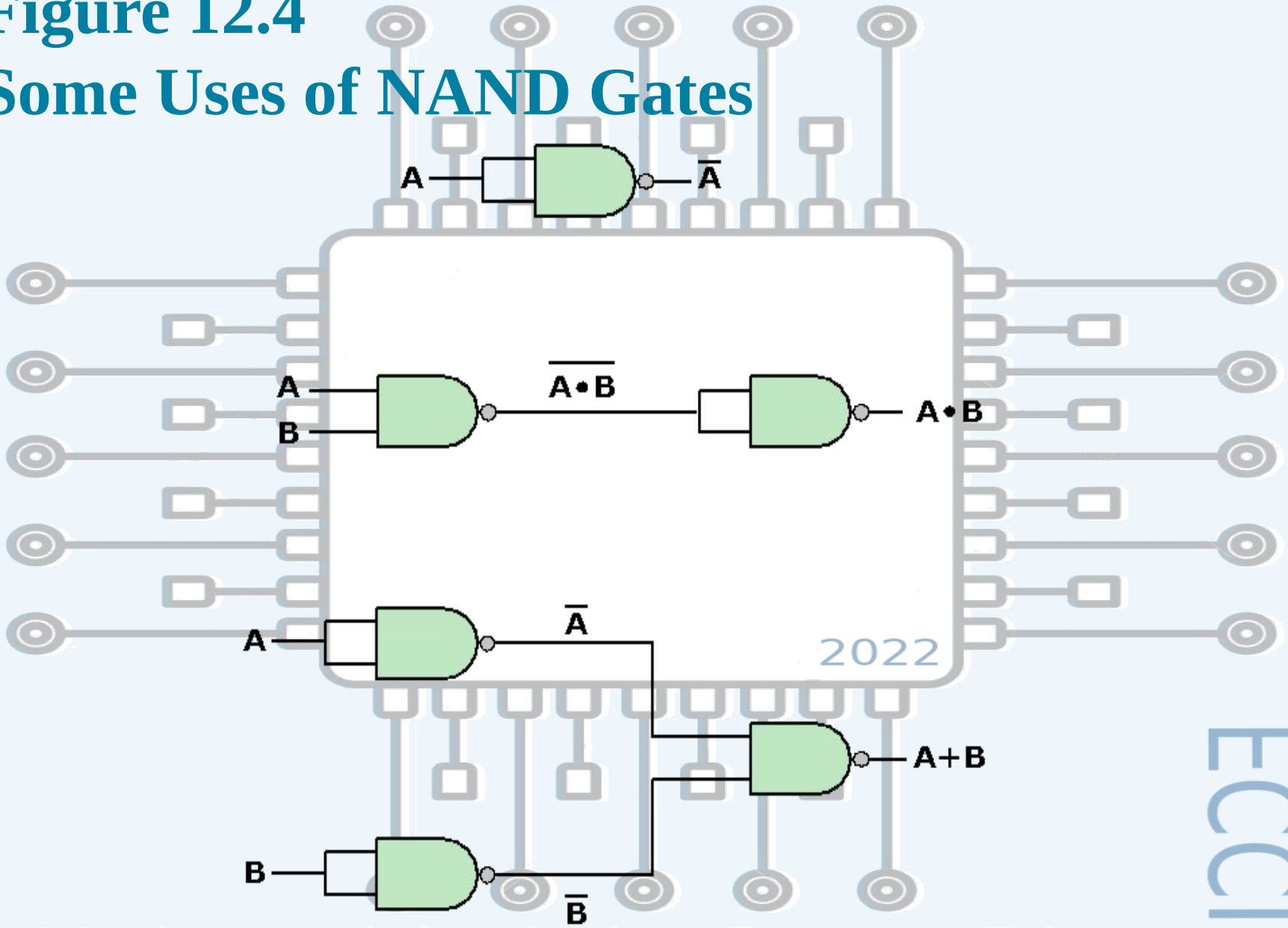
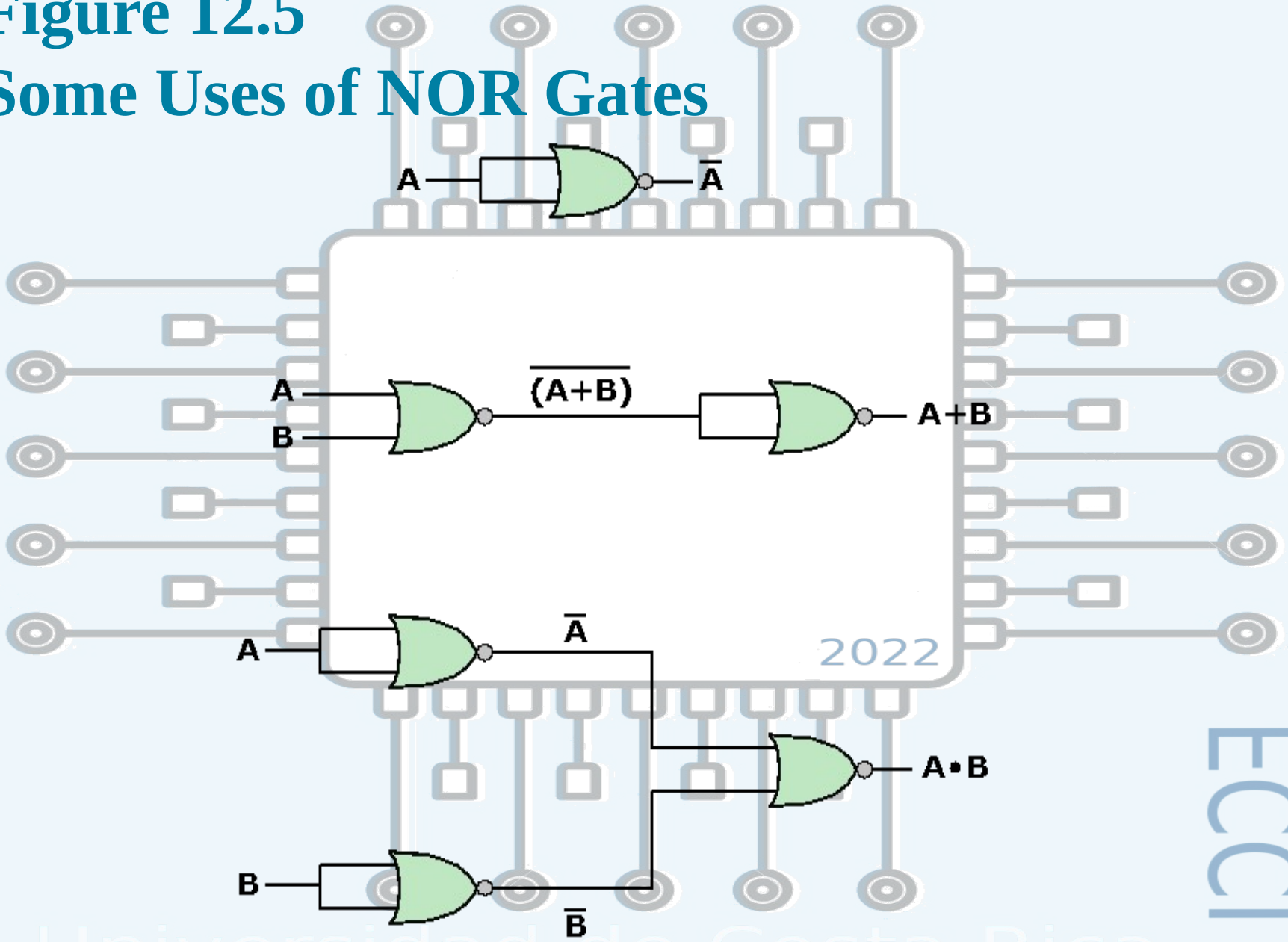
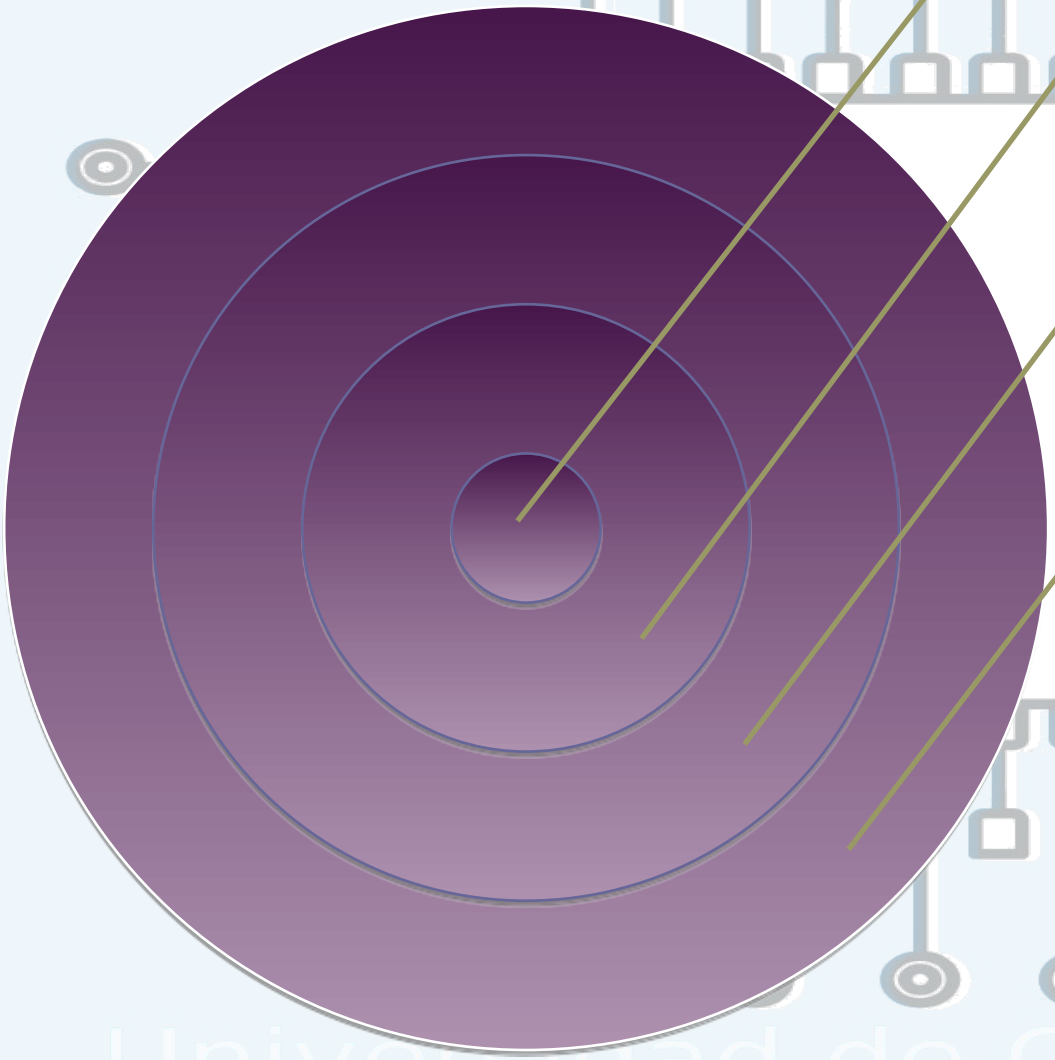


Figure 12.5
Some Uses of NOR Gates



Combinational Circuit



An interconnected set of gates whose output at any time is a function only of the input at that time

The appearance of the input is followed almost immediately by the appearance of the output, with only gate delays

Consists of *n* binary inputs and *m* binary outputs

Can be defined in three ways:

- Truth table
  - For each of the 2^n possible combinations of input signals, the binary value of each of the *m* output signals is listed
- Graphical symbols
  - The interconnected layout of gates is depicted
- Boolean equations
  - Each output signal is expressed as a Boolean function of its input signals

Table 12.4
A Boolean Function of Three Variables

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Figure 12.6 Sum-of-Products Implementation of Table 12.4

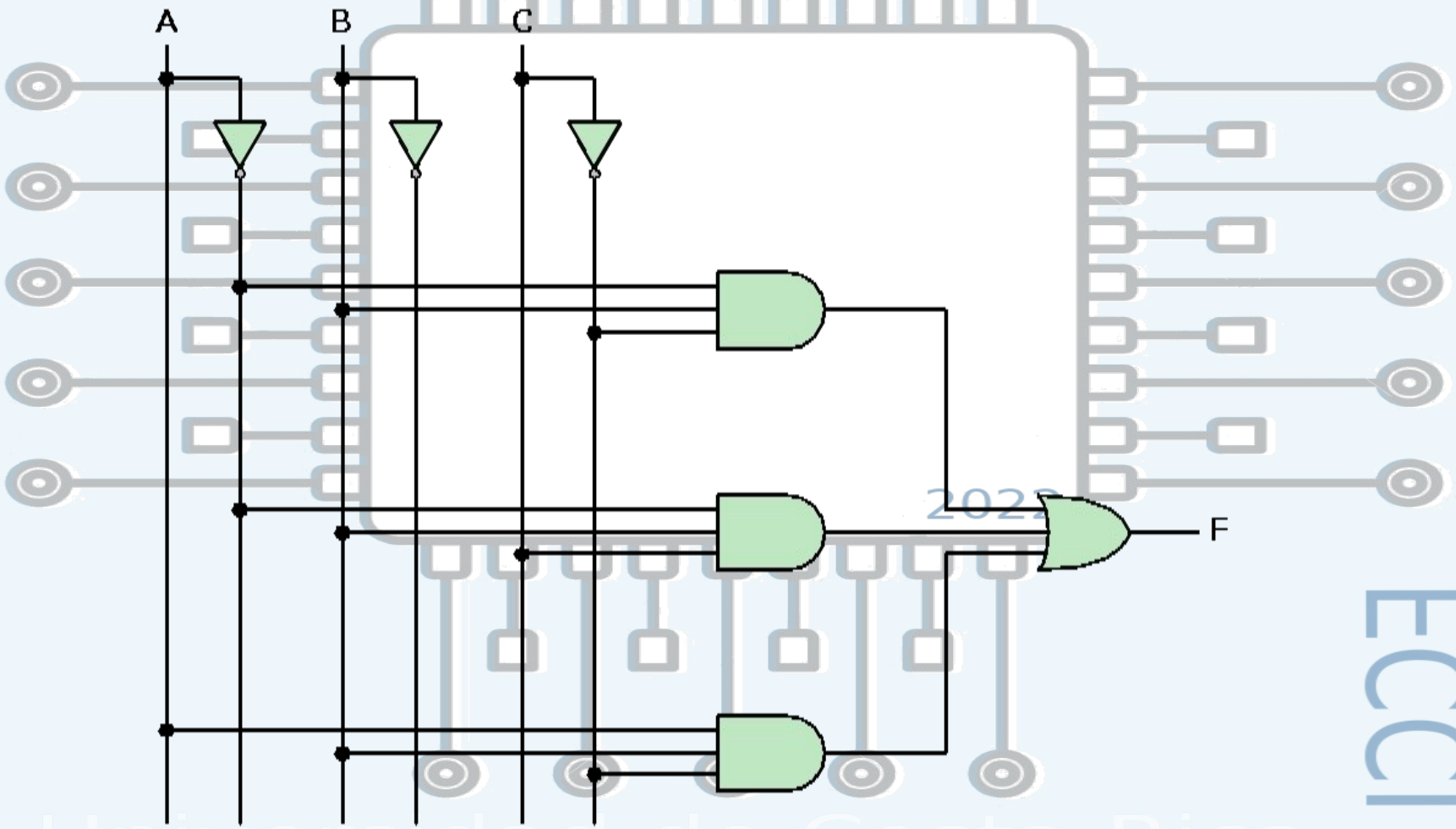


Figure 12.7 Product-of-Sums Implementation of Table 12.4

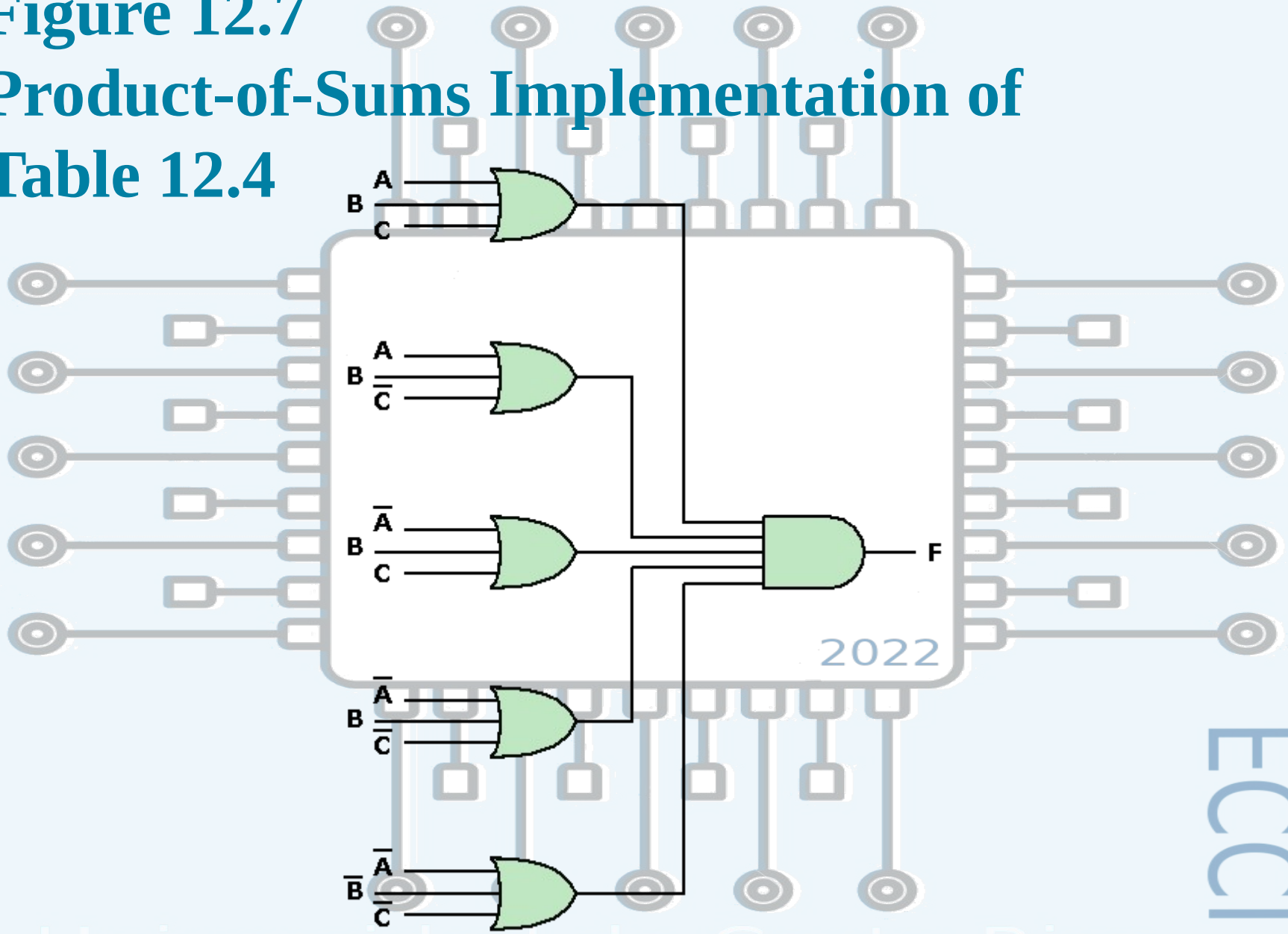


Figure 12.8
Simplified Implementation of Table 12.4

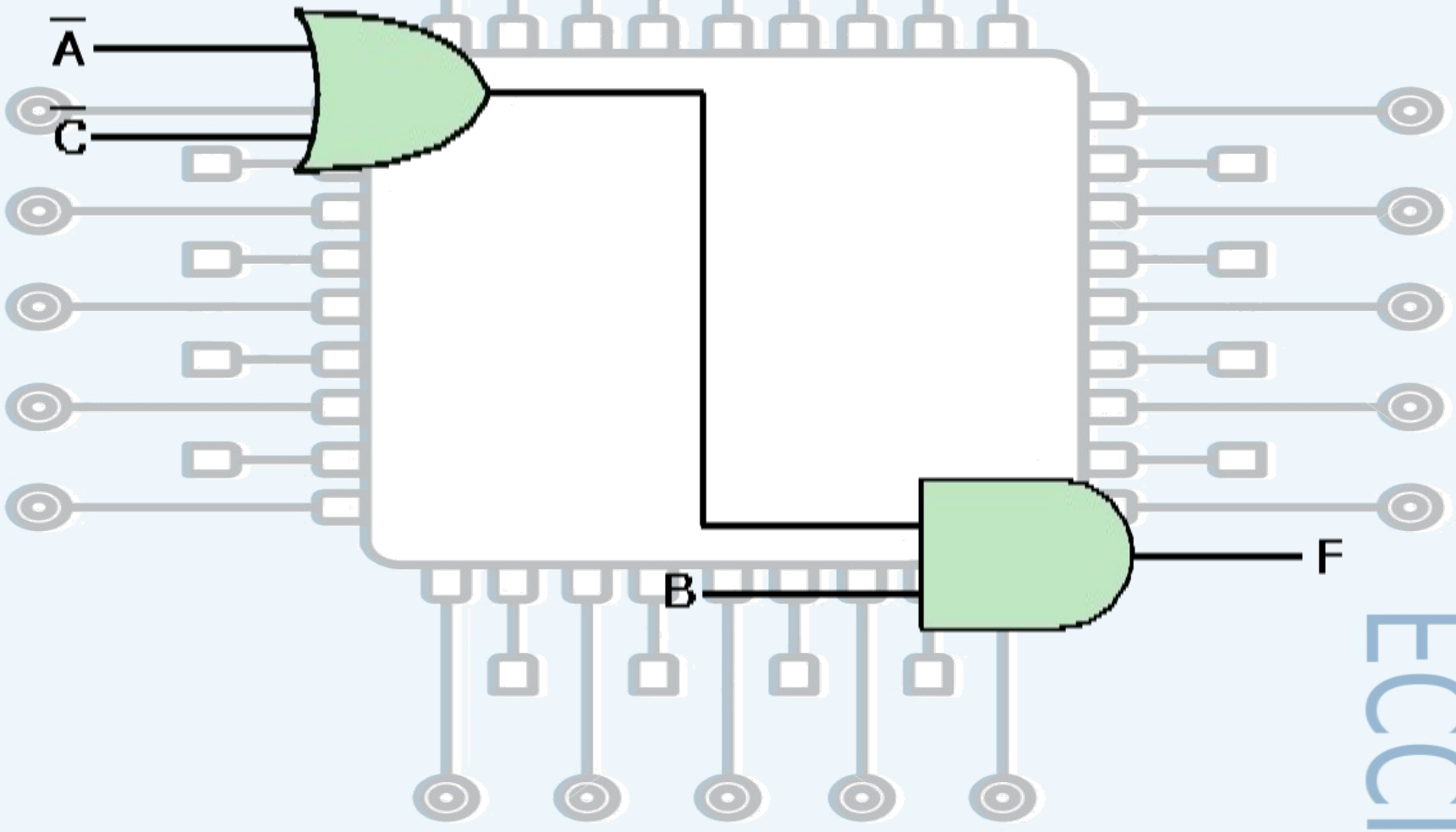


Figure 12.9
The Use of Karnaugh Maps to Represent Boolean Functions

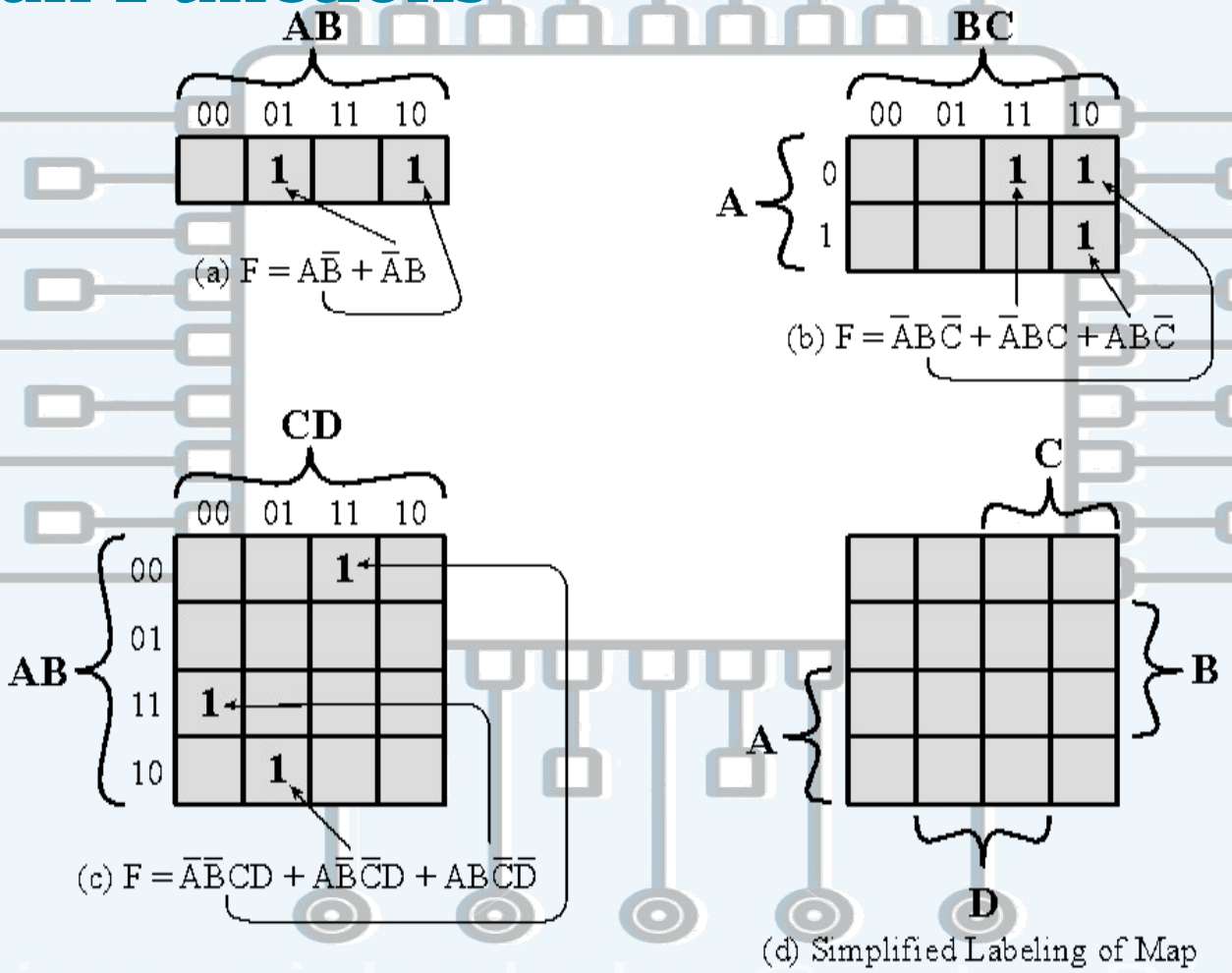


Figure 12.10
The Use of Karnaugh Maps

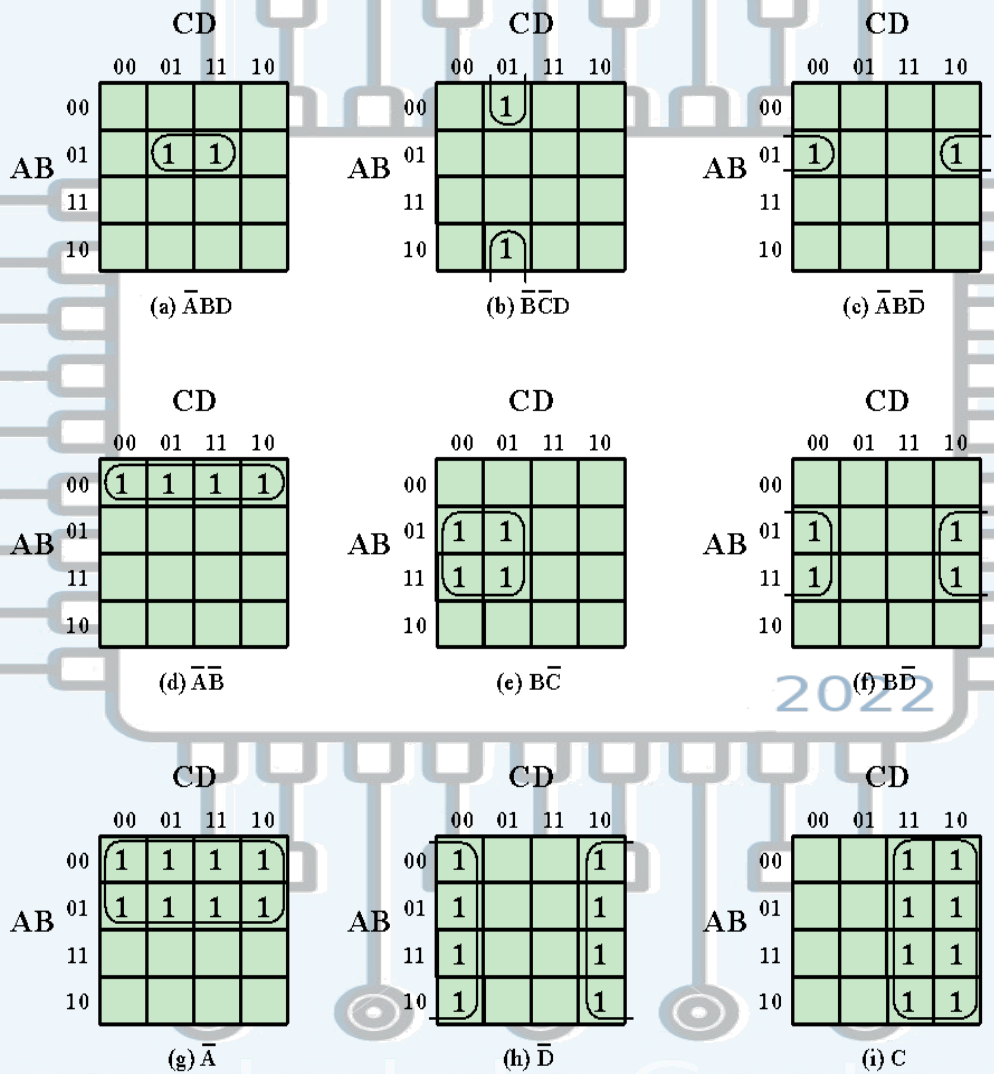


Figure 12.11 Overlapping Groups

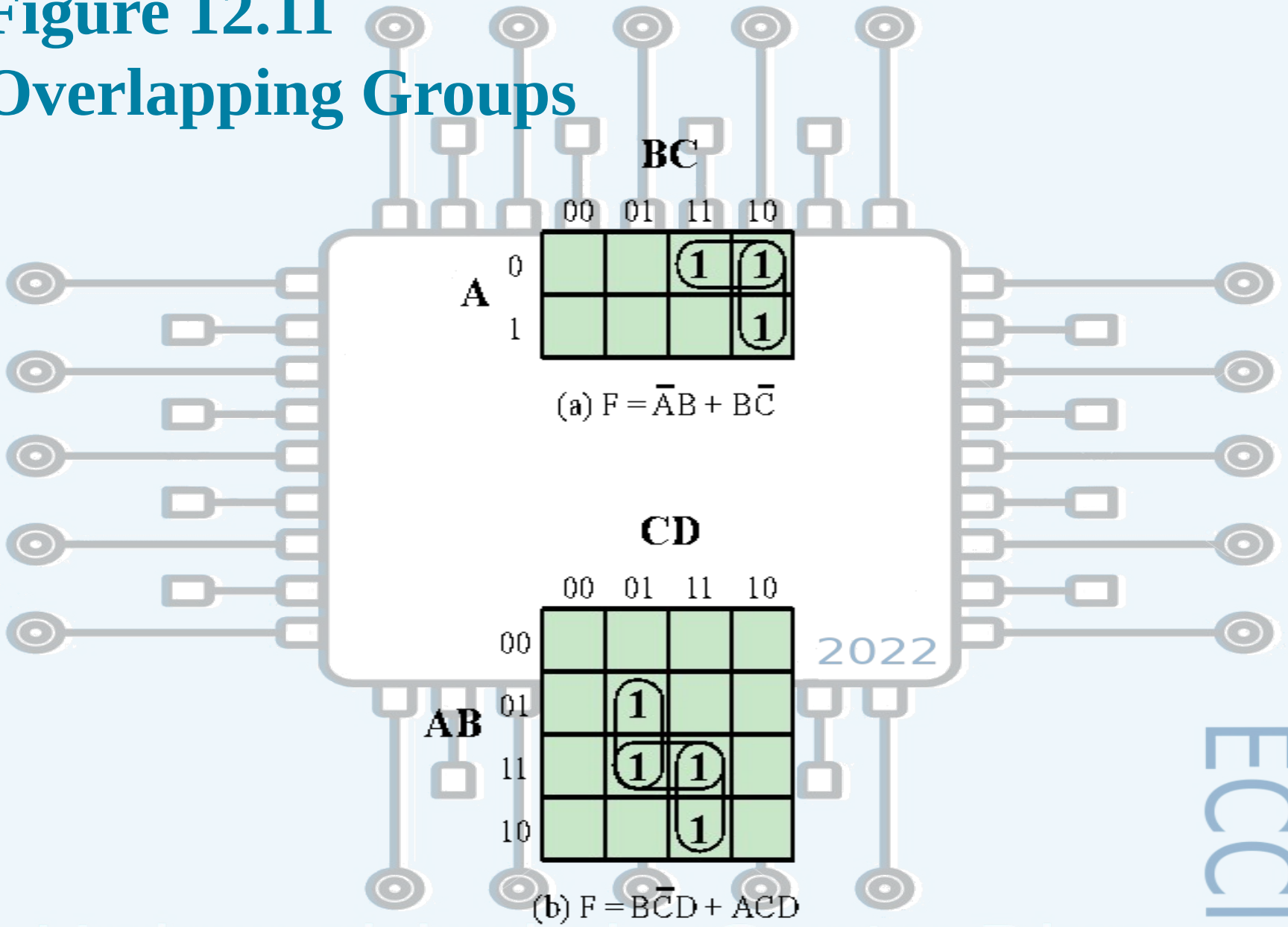


Table 12.5
Truth Table for the One-Digit Packed Decimal Incrementer

Number	Input				Number	Output			
	A	B	C	D		W	X	Y	Z
0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	2	0	0	1	0
2	0	0	1	0	3	0	0	1	1
3	0	0	1	1	4	0	1	0	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	6	0	1	1	0
6	0	1	1	0	7	0	1	1	1
7	0	1	1	1	8	1	0	0	0
8	0	0	0	0	9	1	0	0	1
9	1	0	0	1	0	0	0	0	0
Don't care condition	1	0	1	0		d	d	d	d
	1	0	1	1		d	d	d	d
	1	1	0	0		d	d	d	d
	1	1	0	1		d	d	d	d
	1	1	1	0		d	d	d	d
	1	1	1	1		d	d	d	d

Figure 12.12
Karnaugh Maps for the Incrementer

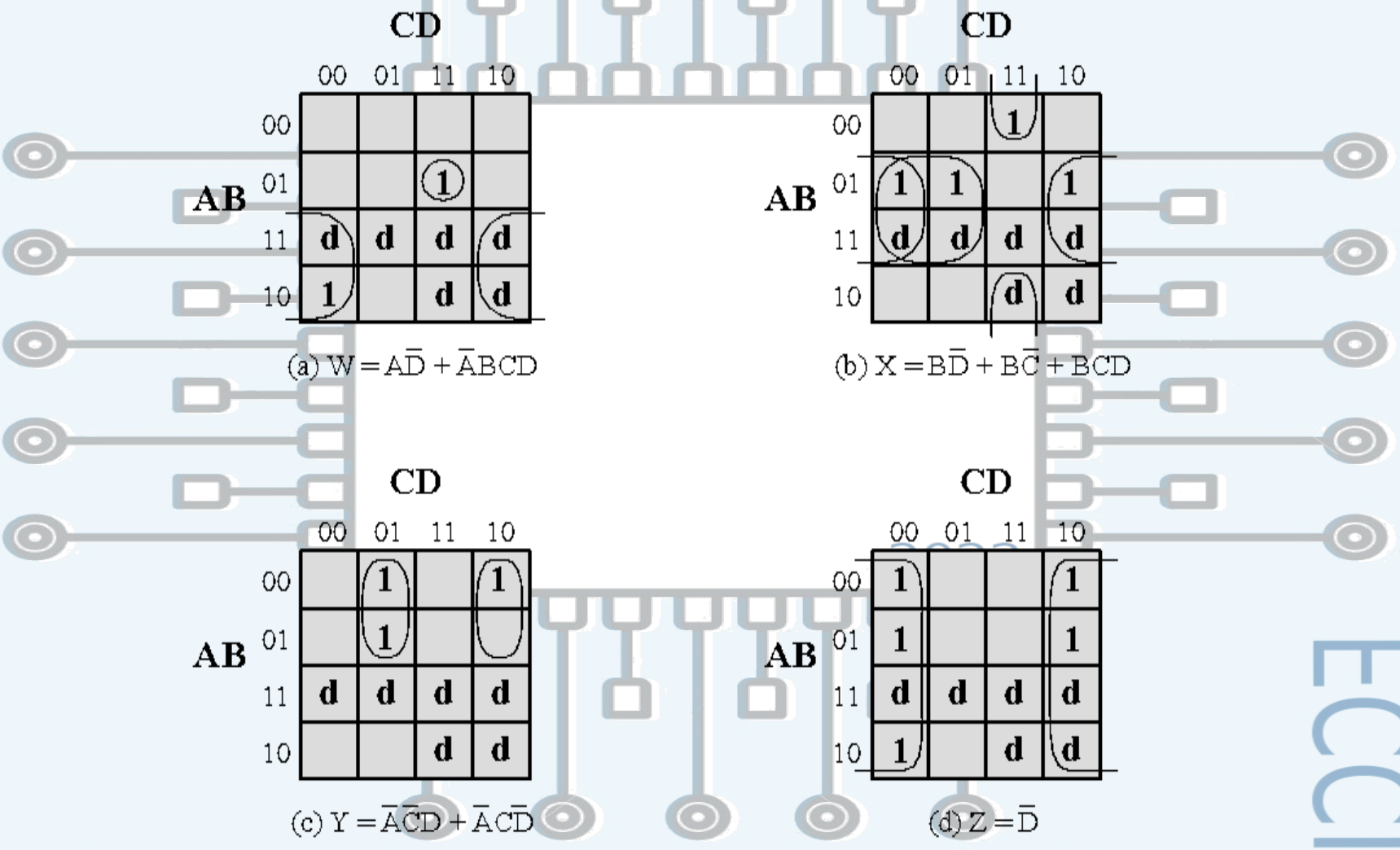


Table 12.6
First Stage of Quine–McCluskey Method

(for $F = ABCD + AB\bar{C}D + AB\bar{C}\bar{D} + A\bar{B}CD + \bar{A}BCD + \bar{A}BC\bar{D} + \bar{A}B\bar{C}D + \bar{A}\bar{B}\bar{C}D$)

Product Term	Index	A	B	C	D	
	1	0	0	0	1	
	5	0	1	0	1	
	6	0	1	1	0	
	12	1	1	0	0	
	7	0	1	1	1	
	11	1	0	1	1	
	13	1	1	0	1	
A B C D	15	1	1	1	1	

Table 12.7
Last Stage of Quine–McCluskey Method

(for $F = ABCD + AB\bar{C}D + AB\bar{C}\bar{D} + A\bar{B}CD + \bar{A}BCD + \bar{A}BC\bar{D} + \bar{A}B\bar{C}D + \bar{A}\bar{B}\bar{C}D$)

	ABCD							
BD	X	X			X		X	
							[X]	⊗
					[X]	⊗		
		[X]	⊗					
A C D	[X]			⊗				

Figure 12.13
NAND Implementation of Table 12.4

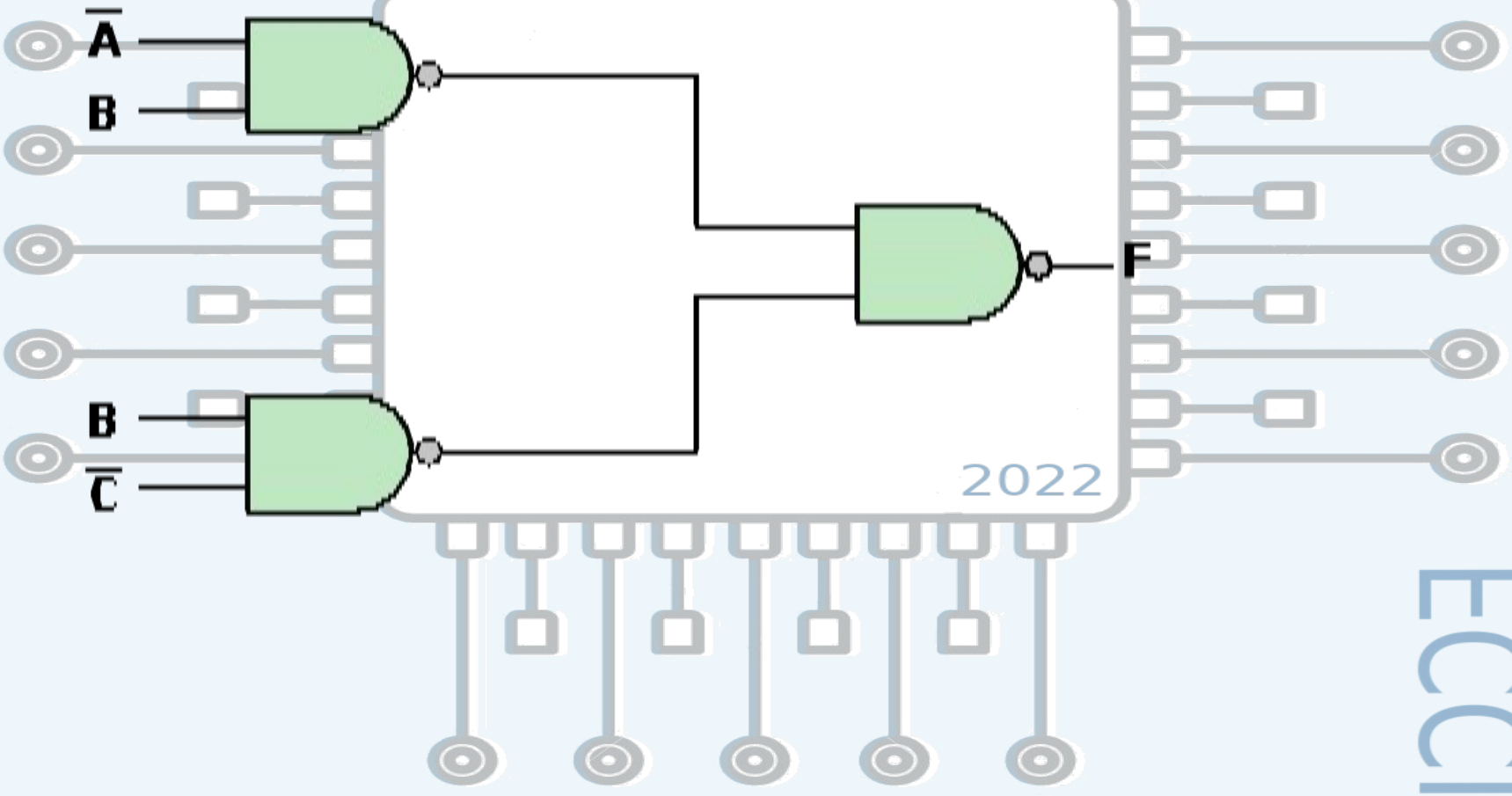


Figure 12.14
4-to-1 Multiplexer Representation

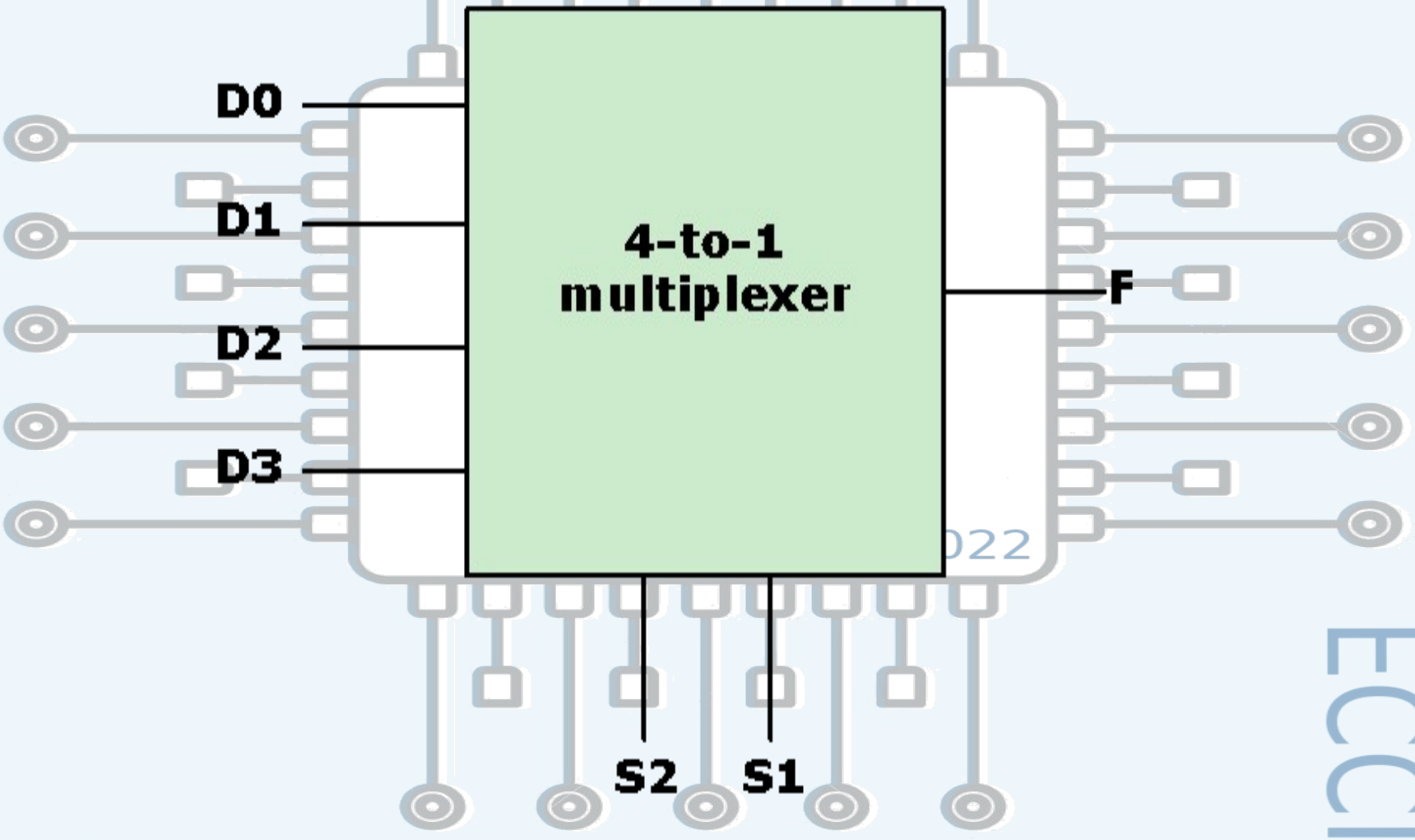


Table 12.8
4-to-1 Multiplexer Truth Table

S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Figure 12.15
Multiplexer Implementation

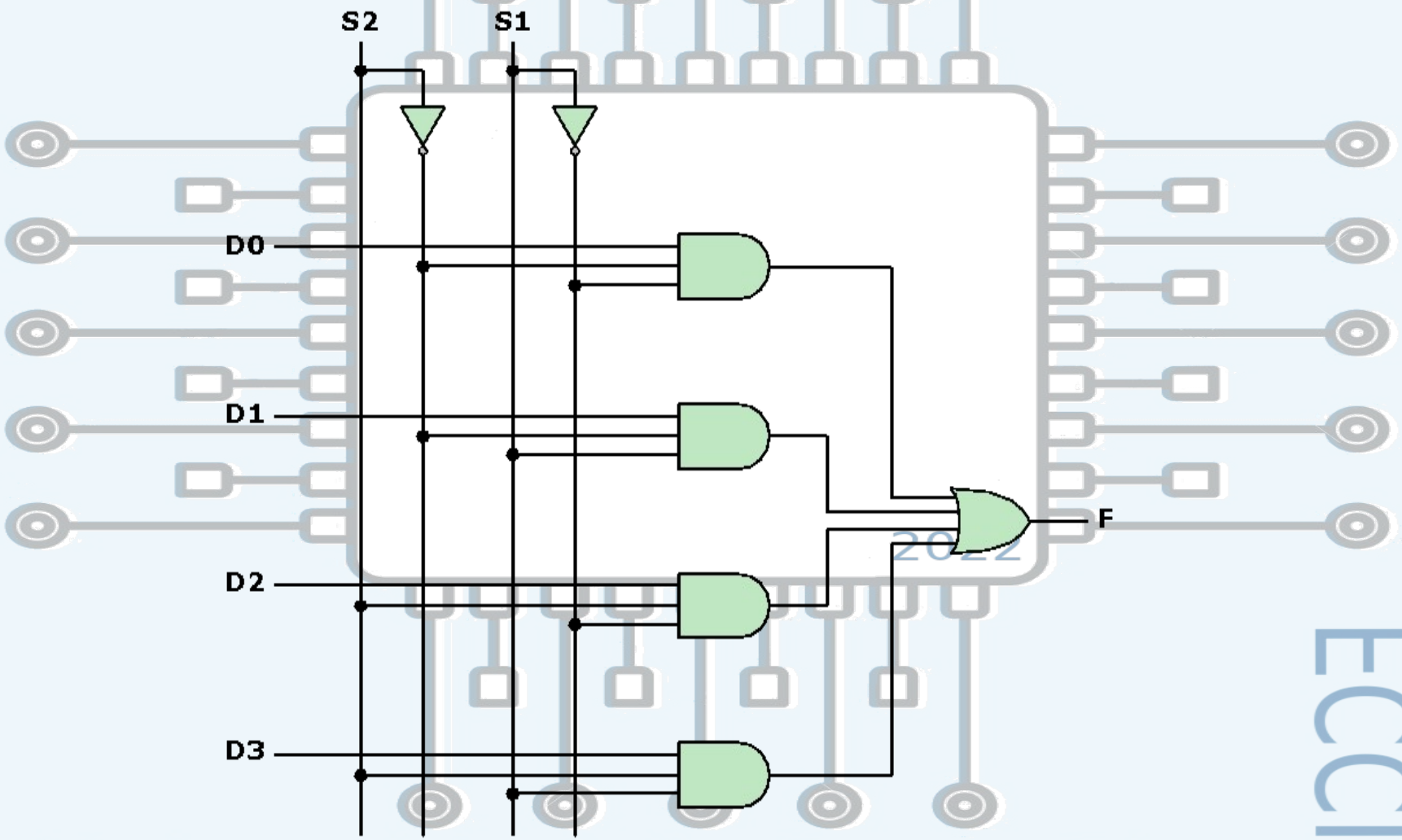


Figure 12.16
Multiplexer Input to Program Counter

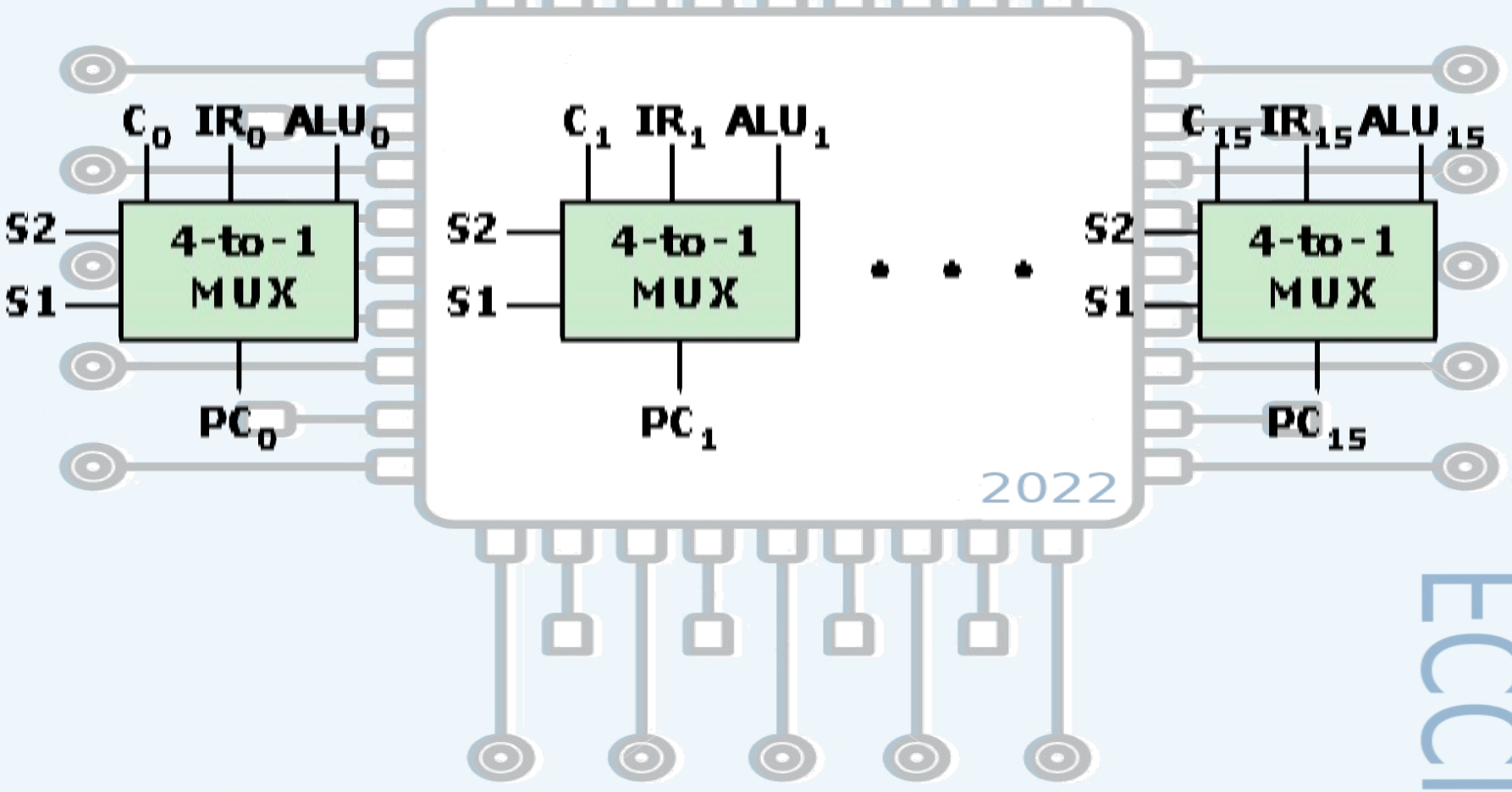


Figure 12.17
Decoder with 3 Inputs and $2^3 = 8$ Outputs

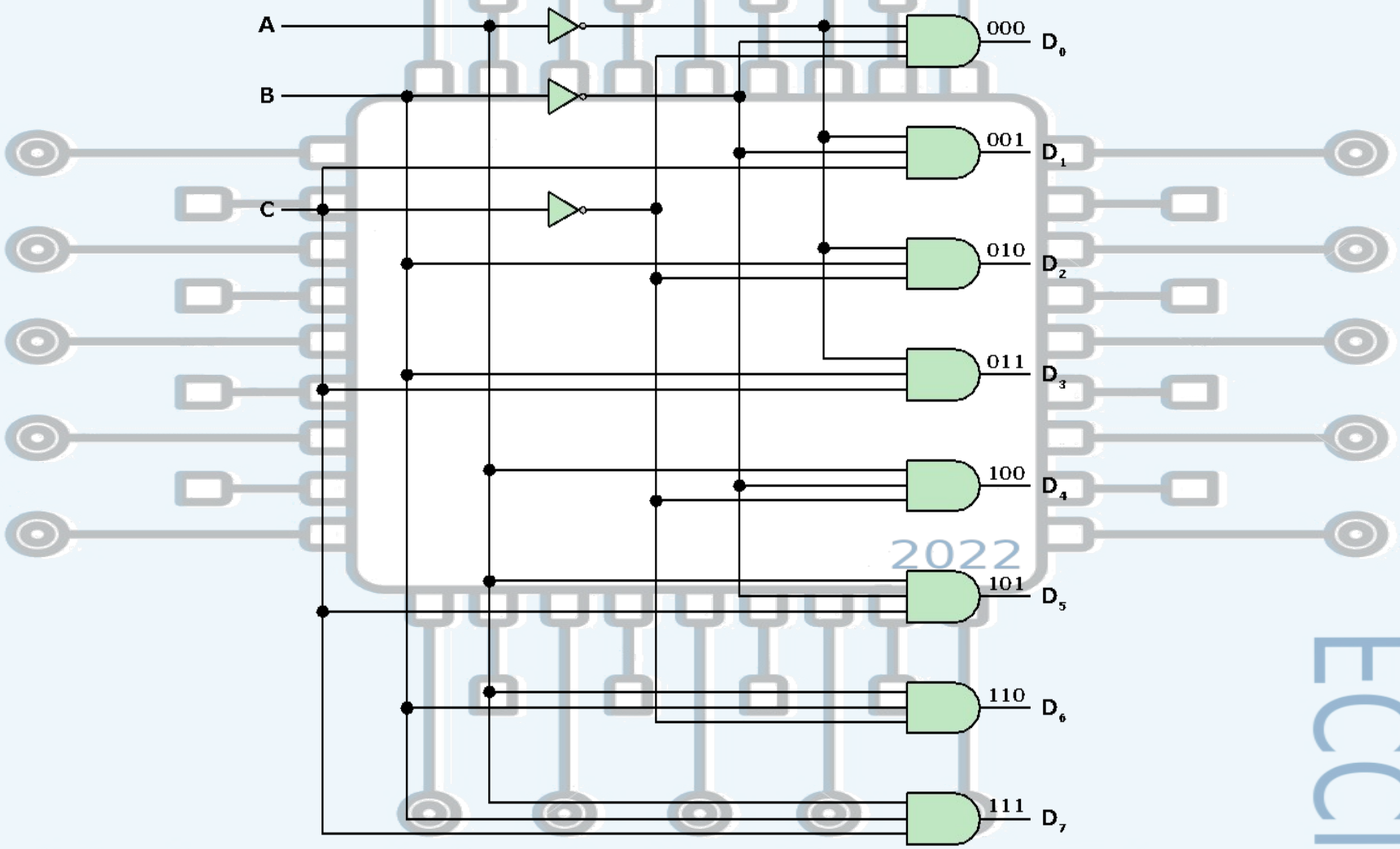


Figure 12.18
Address Decoding

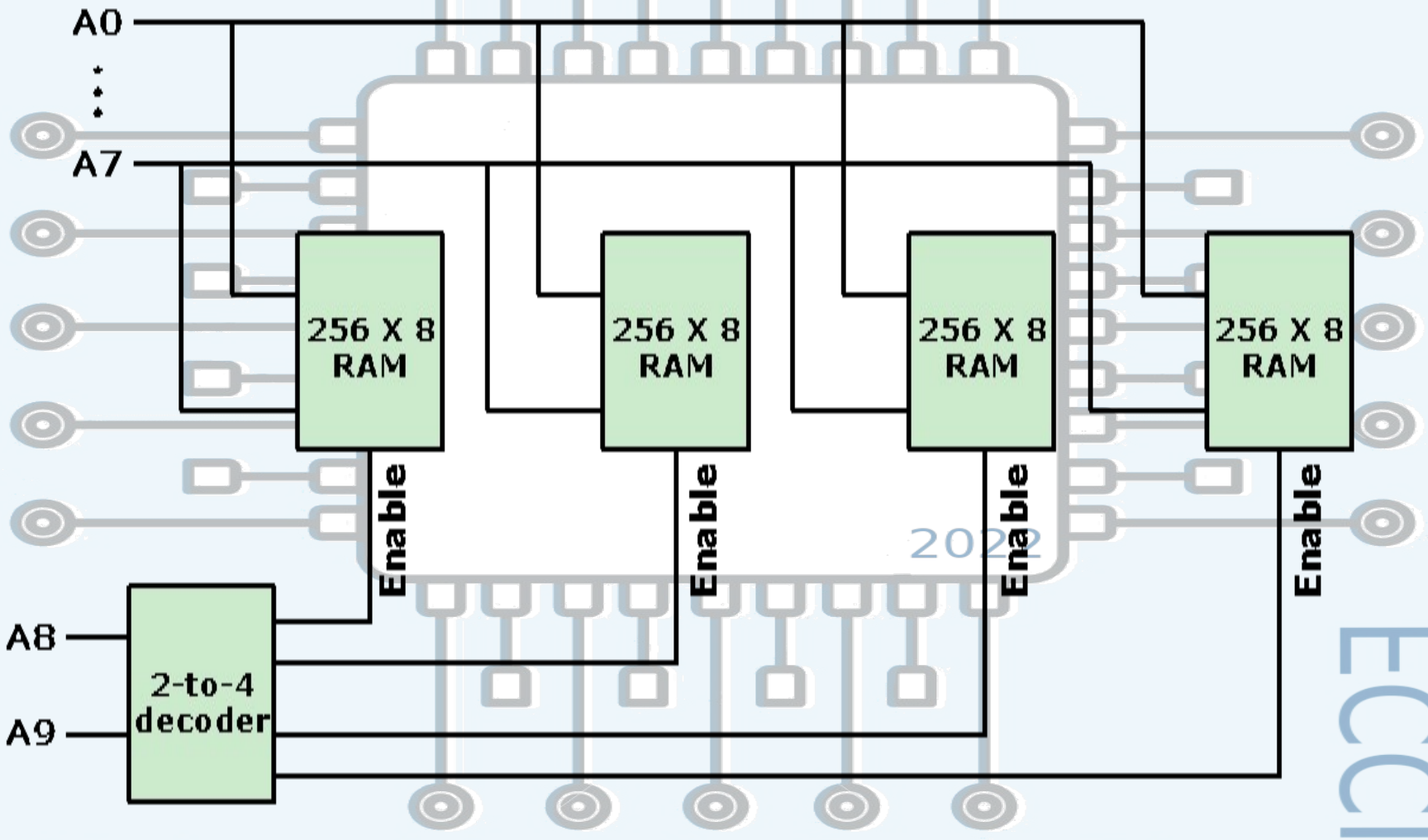
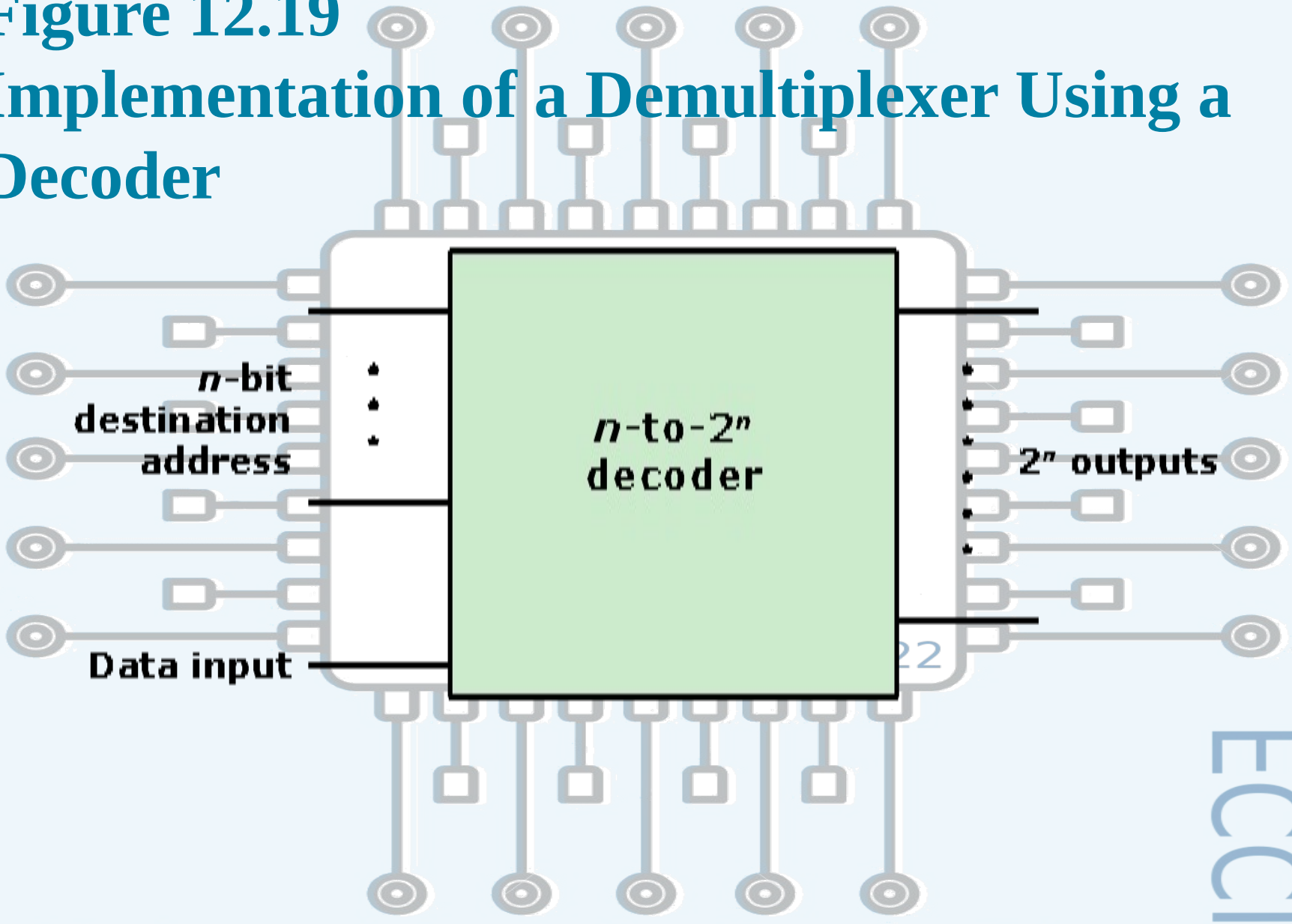


Figure 12.19 Implementation of a Demultiplexer Using a Decoder



Read-Only Memory (ROM)

- Memory that is implemented with combinational circuits
  - Combinational circuits are often referred to as “memoryless” circuits because their output depends only on their current input and no history of prior inputs is retained
- Memory unit that performs only the read operation
  - Binary information stored in a ROM is permanent and is created during the fabrication process
  - A given input to the ROM (address lines) always produces the same output (data lines)
  - Because the outputs are a function only of the present inputs, ROM is a combinational circuit

Table 12.9
Truth Table for a ROM

Input				Output			
X_1	X_2	X_3	X_4	Z_1	Z_2	Z_3	Z_4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	1
1	0	1	0	1	0	1	1
1	0	1	1	1	0	1	0
1	1	0	0	1	1	1	0
1	1	0	1	1	1	1	1
1	1	1	0	1	1	0	1
1	1	1	1	1	1	0	0

Figure 12.20
A 64-Bit ROM

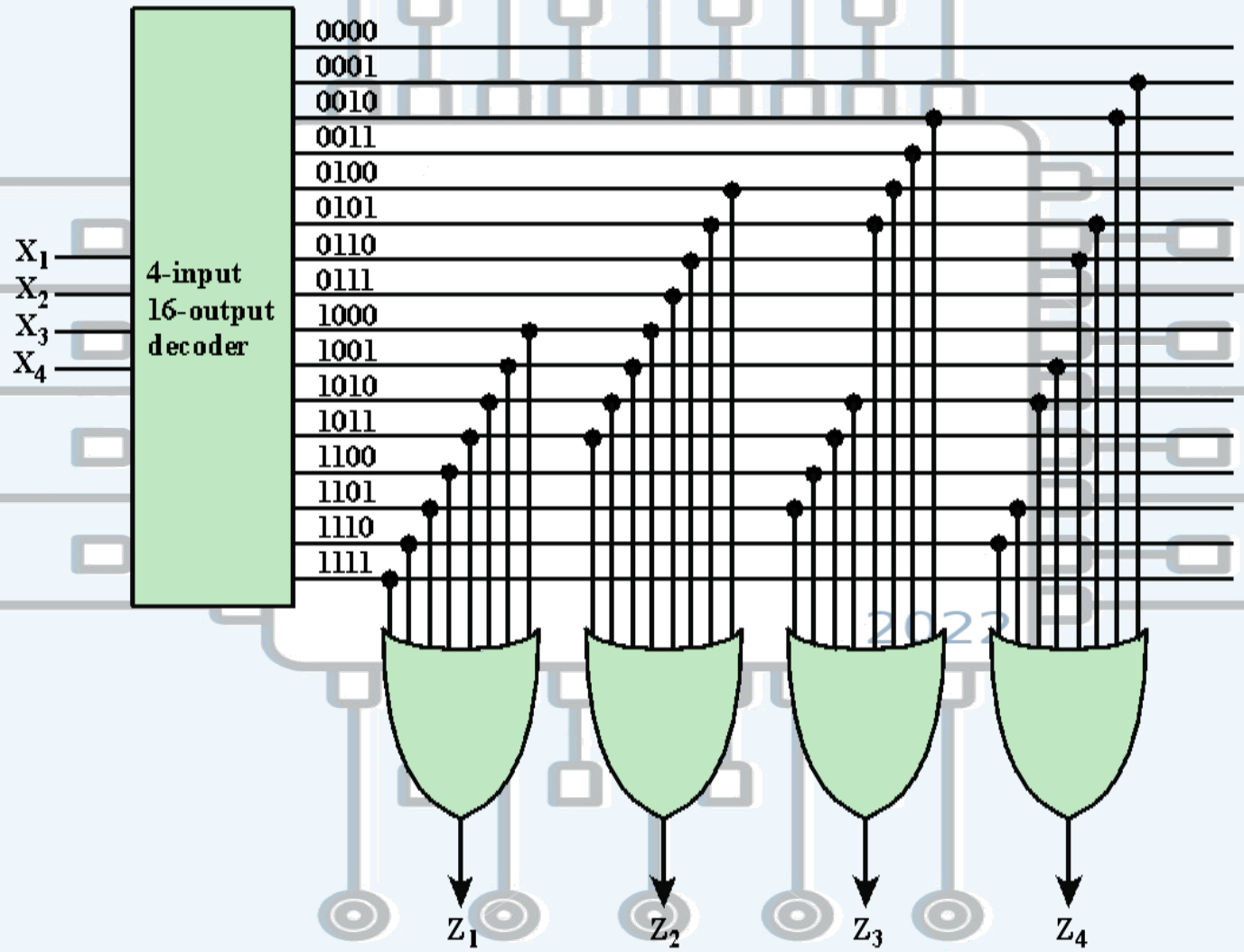


Table 12.10
Binary Addition Truth Tables

(a) Single- Bit Addition

A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b) Addition with Carry Input

C_{in}	A	B	SUM	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 12.21
4-Bit Adder

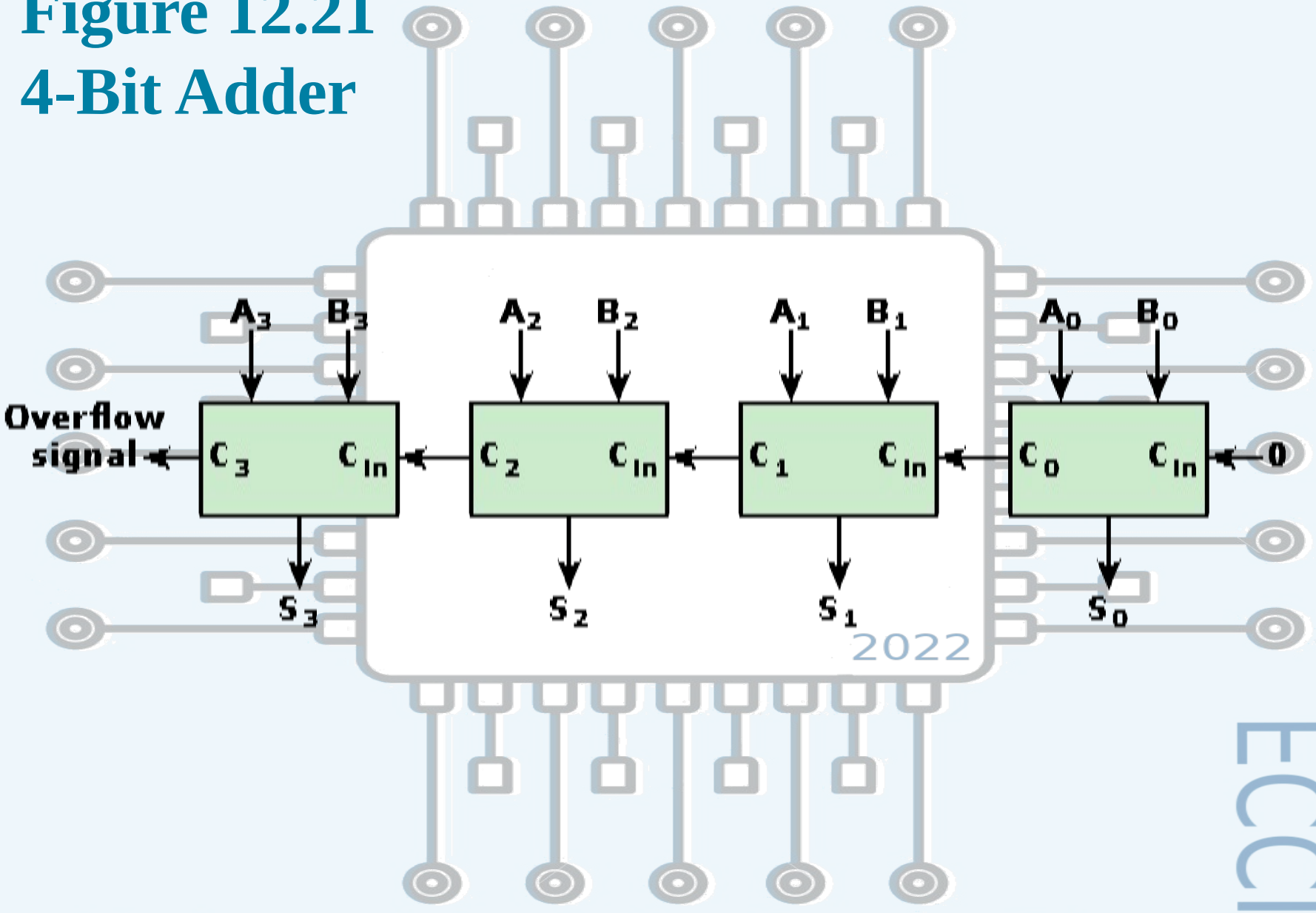


Figure 12.22 Implementation of an Adder

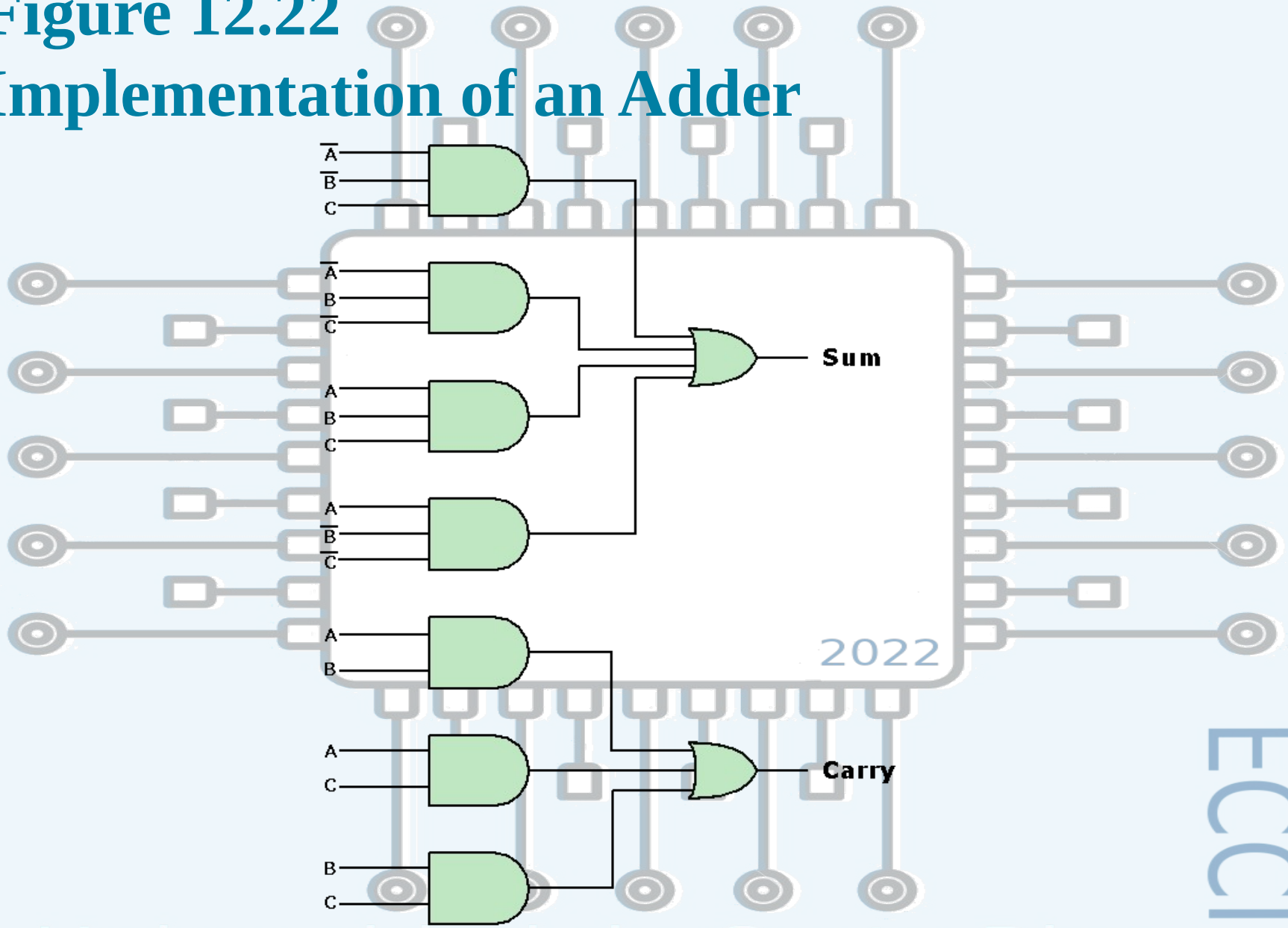
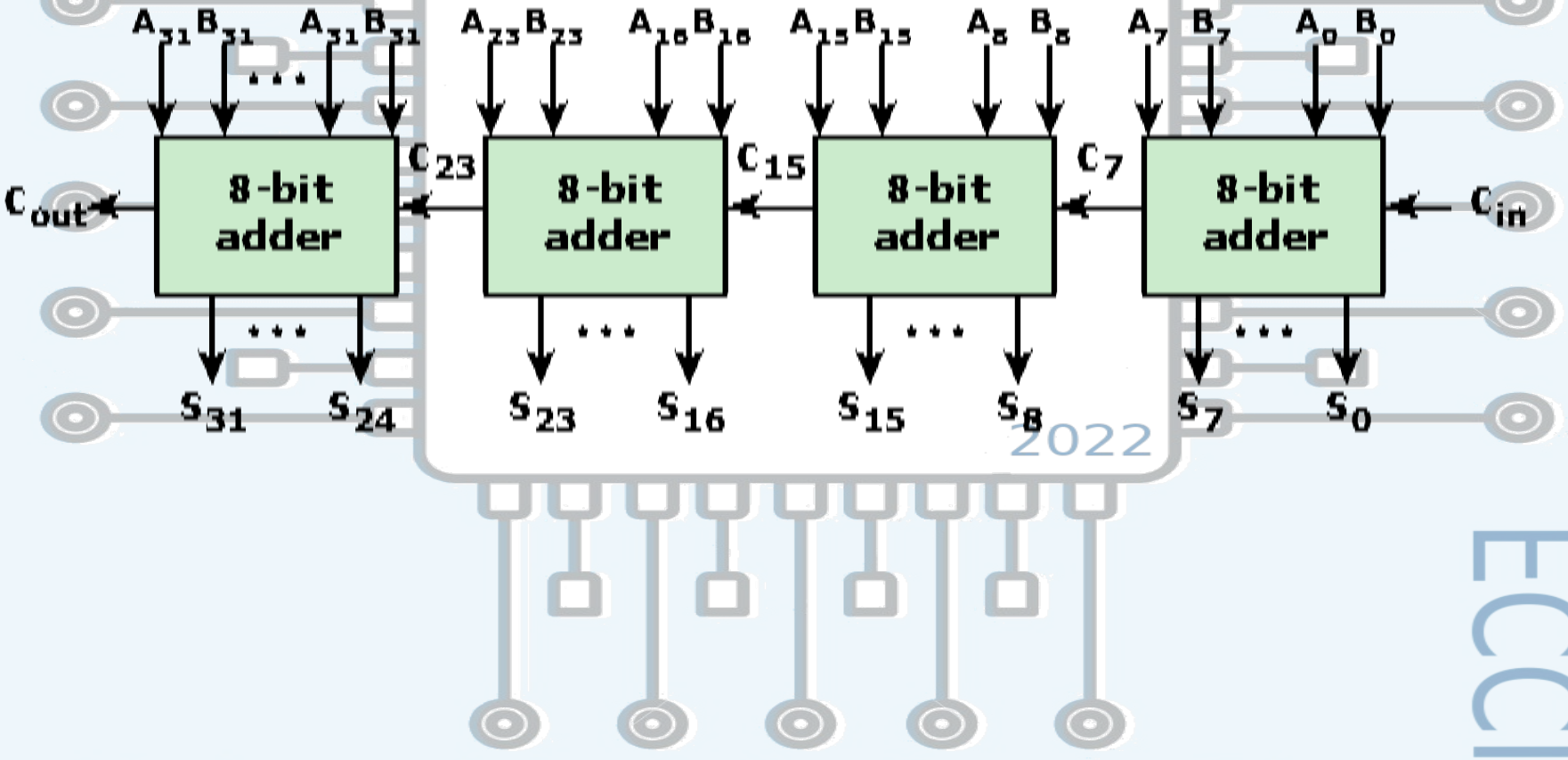
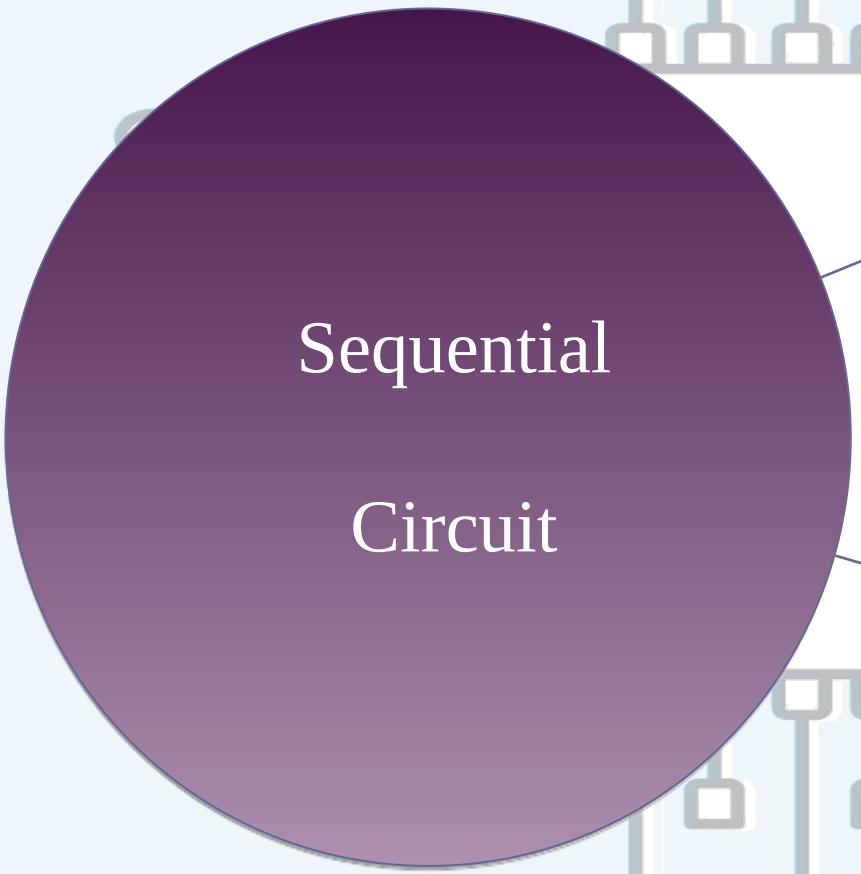


Figure 12.23 Construction of a 32-Bit Adder Using 8-Bit Adders



Sequential Circuit



Current output depends not only on the current input, but also on the past history of inputs

Makes use of combinational circuits

Flip-Flops

- Simplest form of sequential circuit
- There are a variety of flip-flops, all of which share two properties:

1. The flip-flop is a bistable device. It exists in one of two states and, in the absence of input, remains in that state. Thus, the flip-flop can function as a 1-bit memory.
2. The flip-flop has two outputs, which are always the complements of each other.

Figure 12.24
The S–R Latch Implemented with NOR Gates

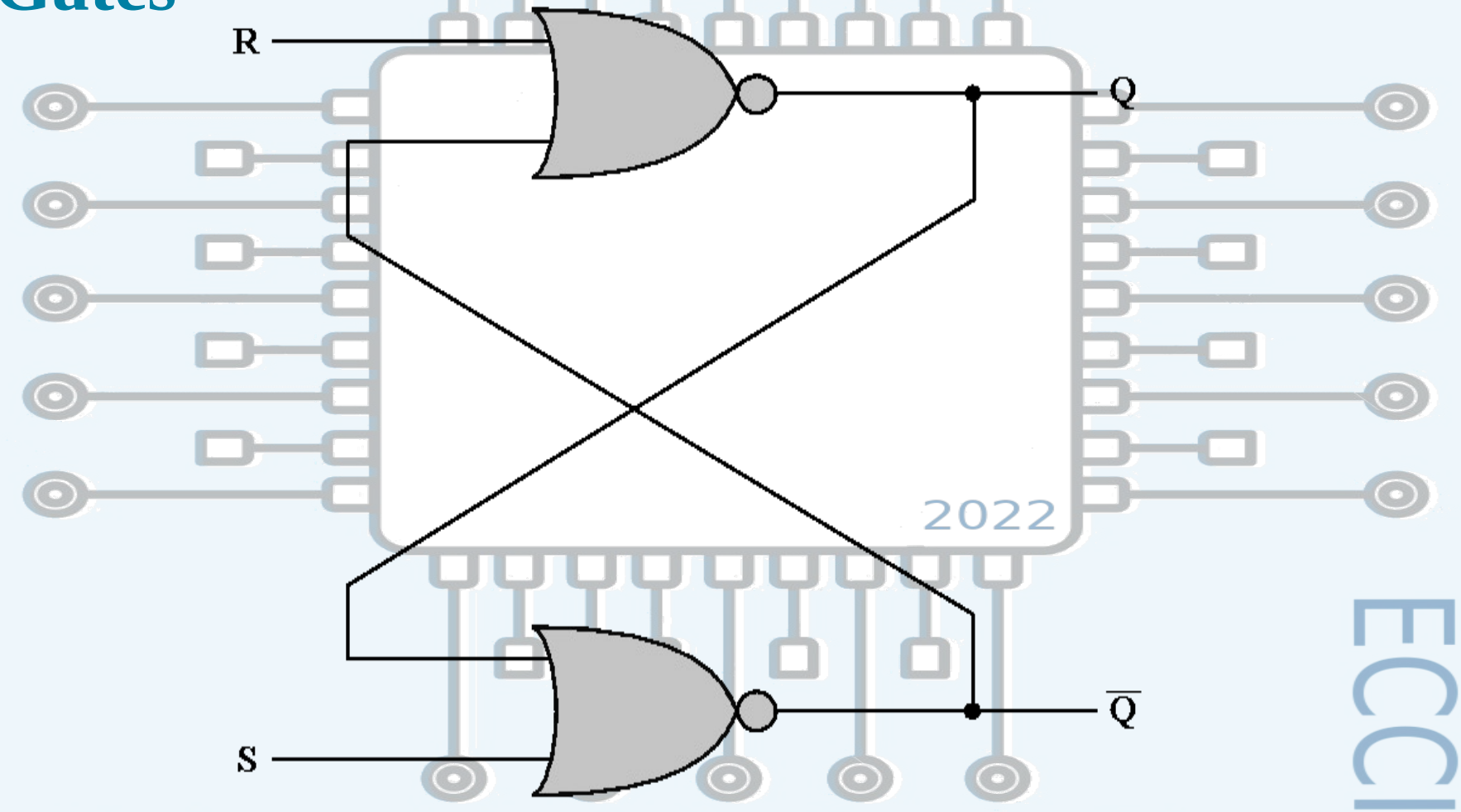


Figure 12.25
NOR S–R Latch Timing Diagram

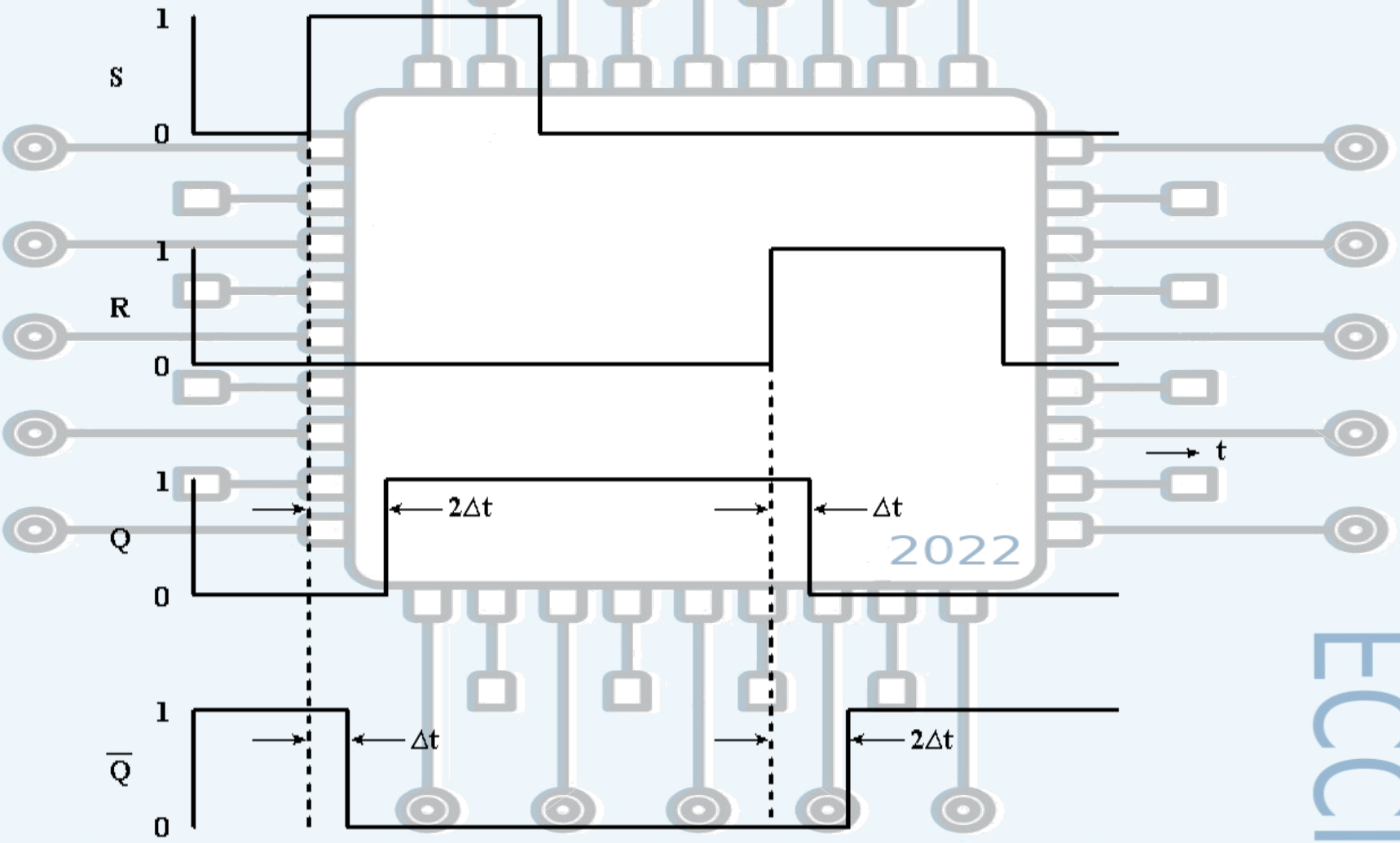


Table 12.12
The S–R Latch

(a) Characteristic Table

Current Inputs	Current State	Next State
SR	Q_n	Q_{n+1}
00	0	0
00	1	1
01	0	0
01	1	0
10	0	1
10	1	1
11	0	–
11	1	–

(b) Simplified Characteristic Table

A	B	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	–

(c) Response to Series of Inputs

t	0	1	2	3	4	5	6	7	8	9
S	1	0	0	0	0	0	0	0	1	0
R	0	0	0	1	0	0	1	0	0	0
Q_{n+1}	1	1	1	0	0	0	0	0	1	1

Figure 12.26
Clocked S–R Flip-Flop

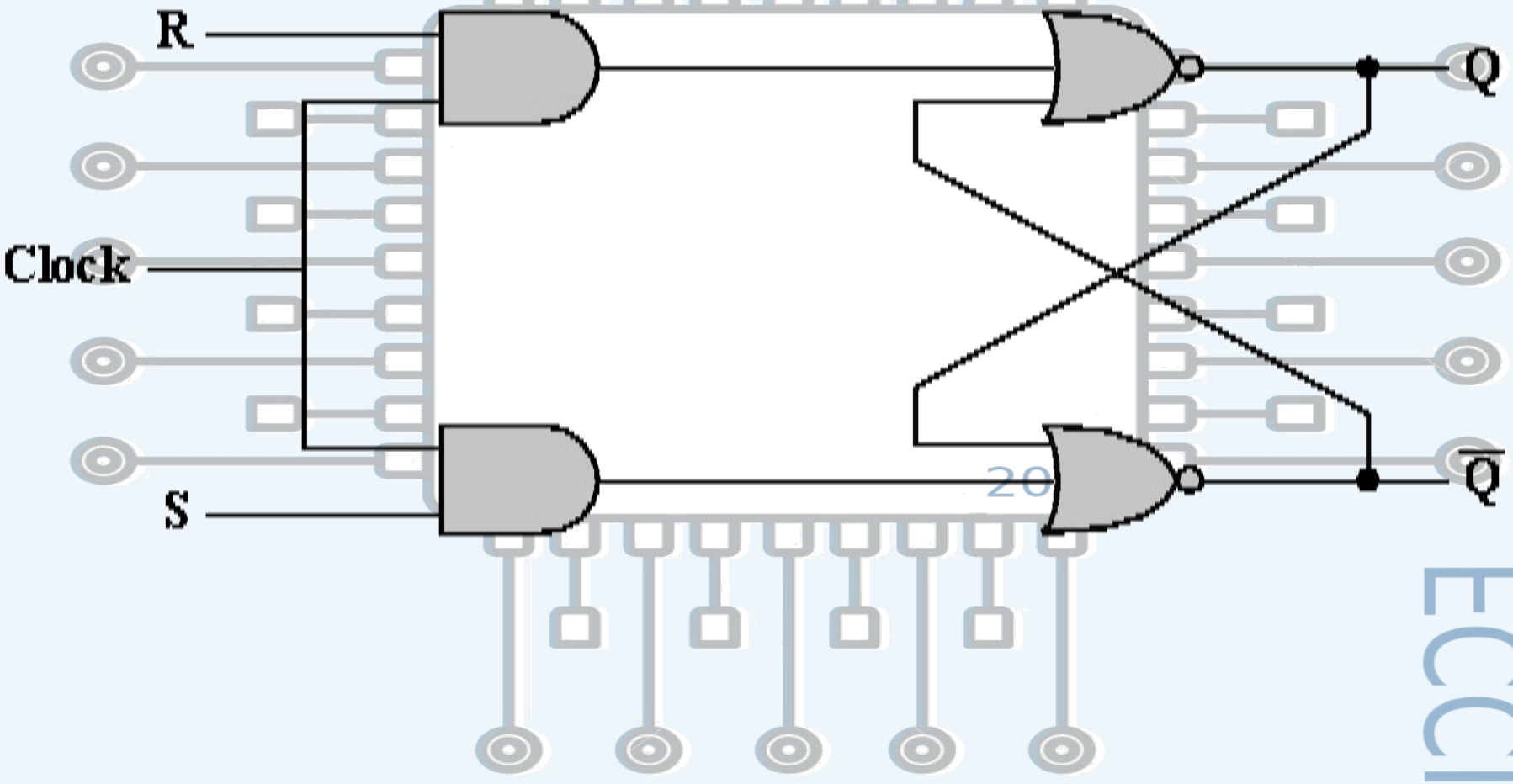


Figure 12.27
D Flip-Flop

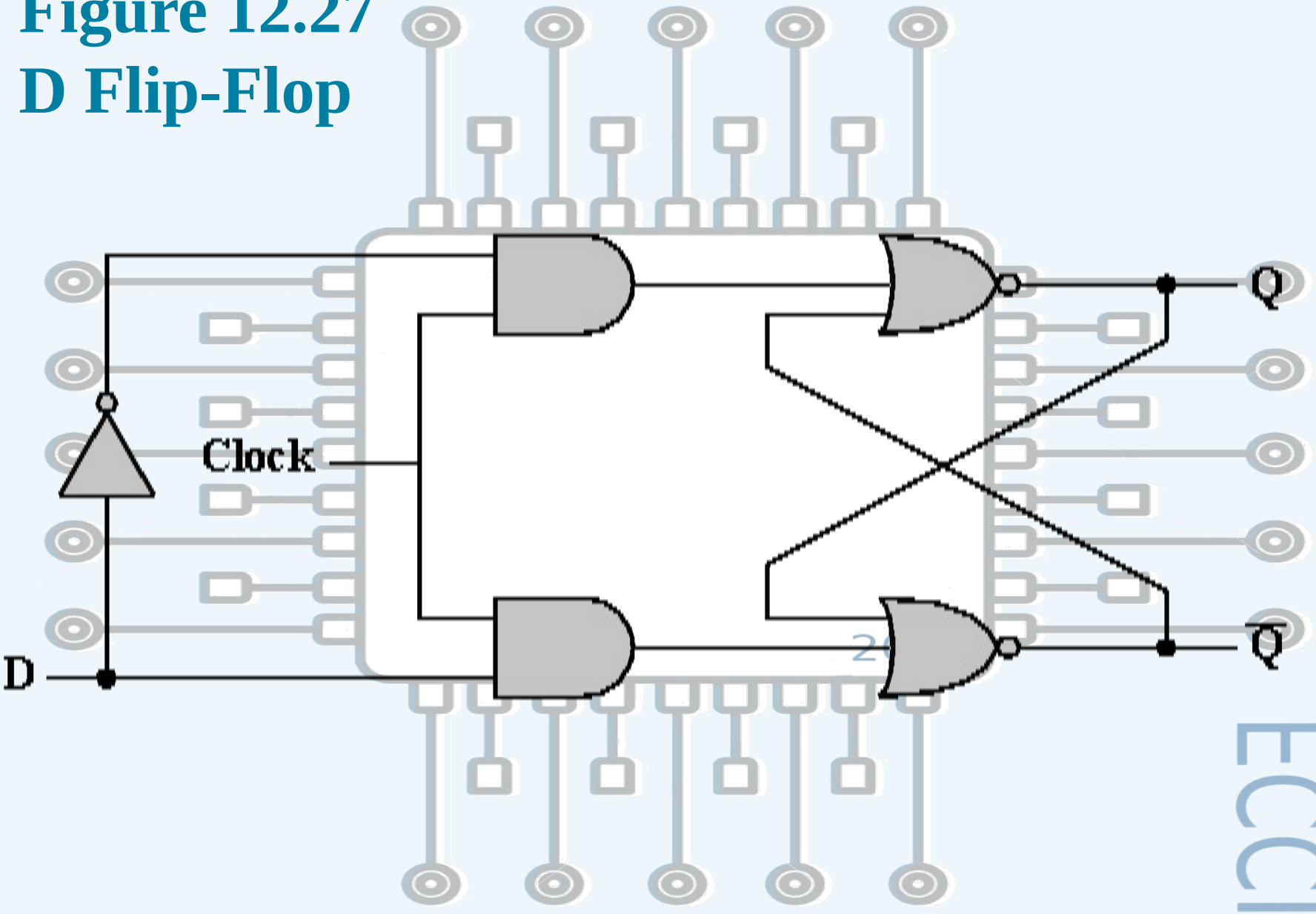


Figure 12.28
J–K Flip-Flop

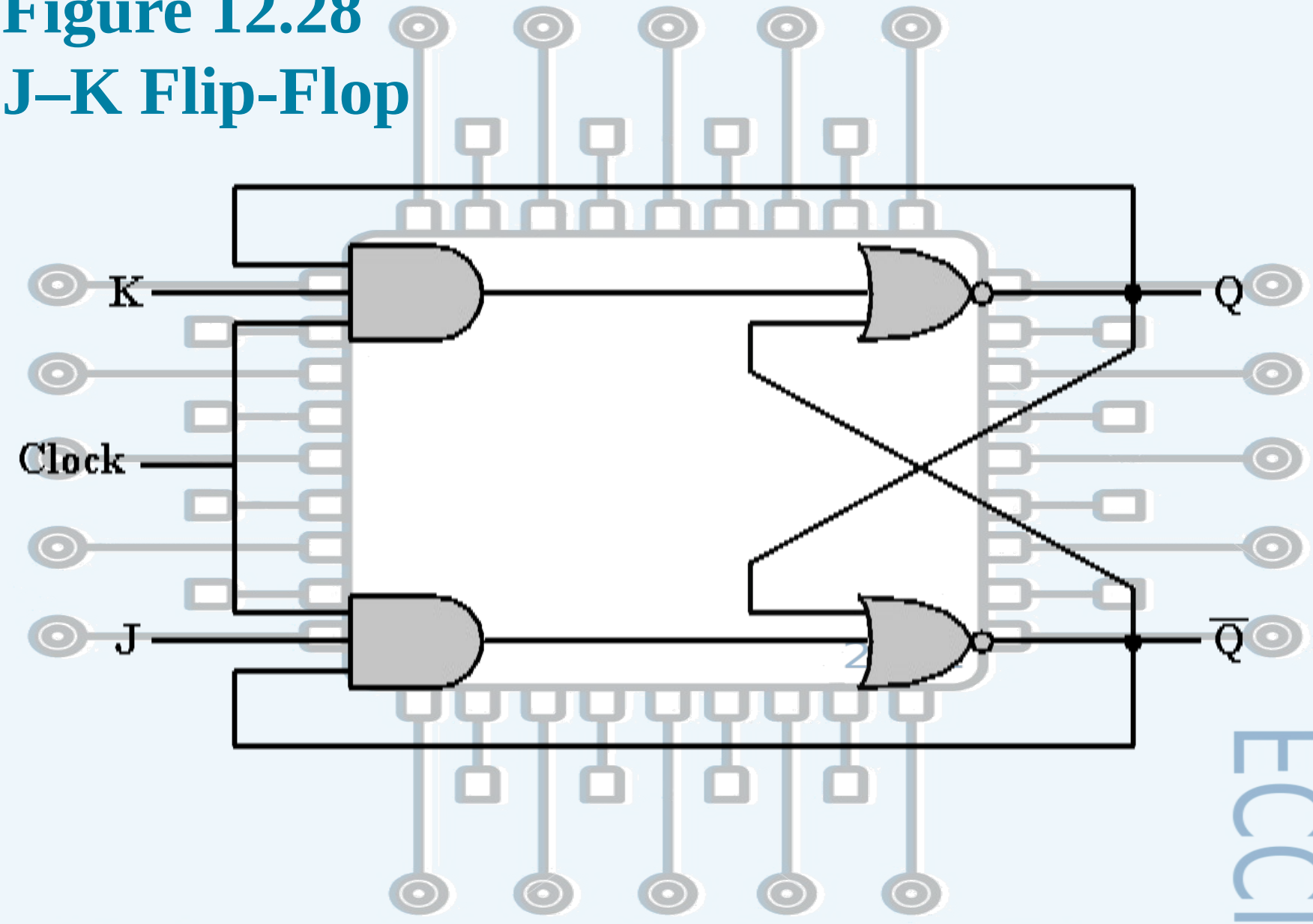


Figure 12.29
Basic Flip-Flops

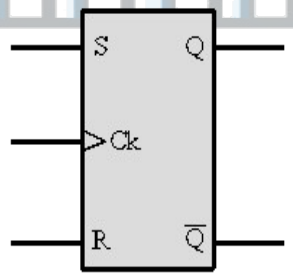
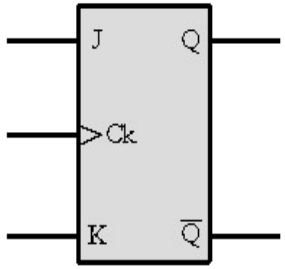
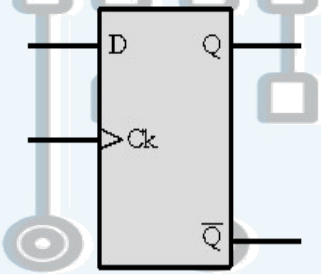
Name	Graphical Symbol	Truth Table
S-R		S R Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 –
J-K		J K Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 $\overline{Q_n}$
D		D Q_{n+1} 0 0 1 1

Figure 12.30
8-Bit Parallel Register

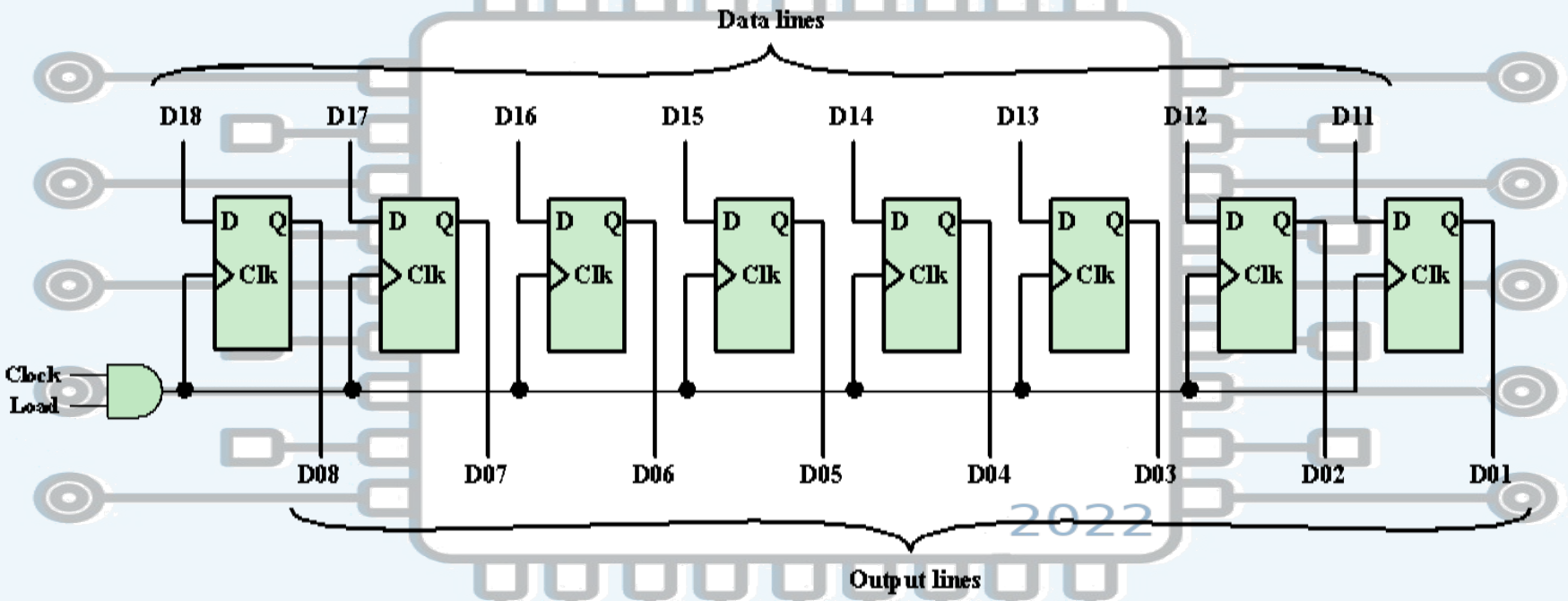
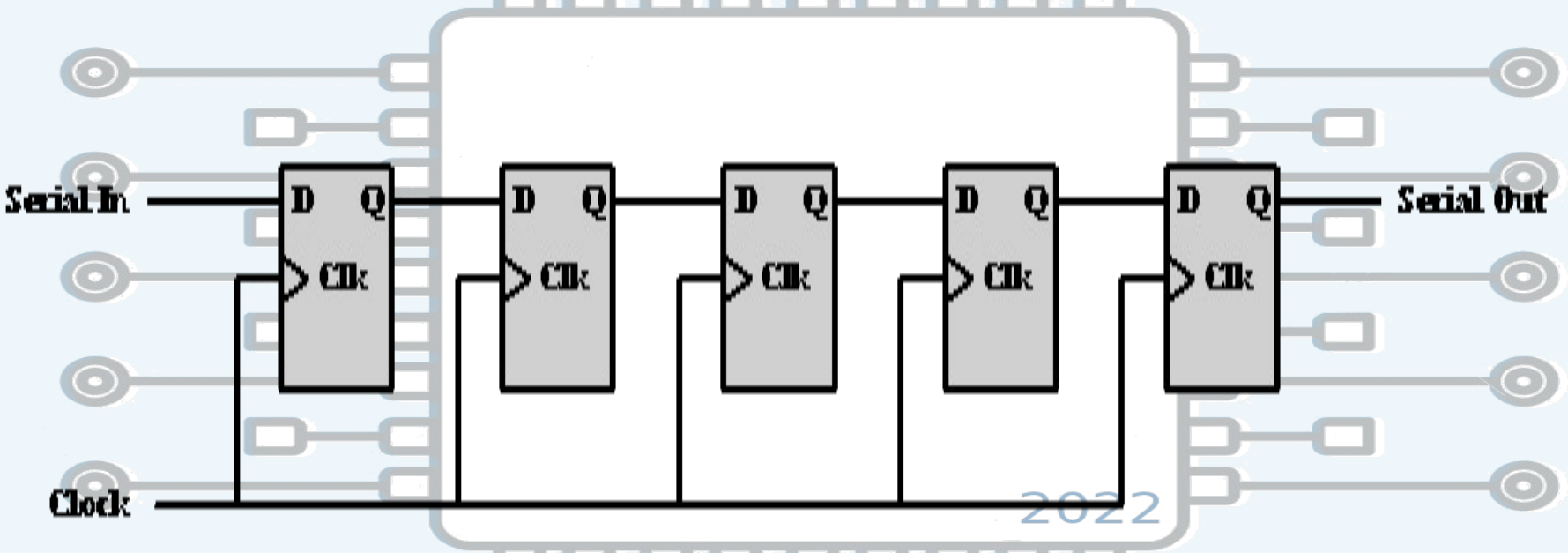


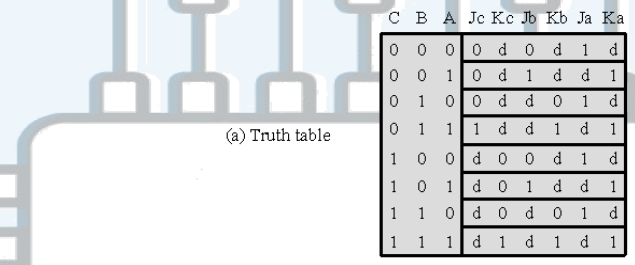
Figure 12.31
5-Bit Shift Register



Counter

- A register whose value is easily incremented by 1 modulo the capacity of the register
- After the maximum value is achieved the next increment sets the counter value to 0
- An example of a counter in the CPU is the program counter
- Can be designated as:
  - Asynchronous
    - Relatively slow because the output of one flip-flop triggers a change in the status of the next flip-flop
  - Synchronous
    - All of the flip-flops change state at the same time
    - Because it is faster it is the kind used in CPUs

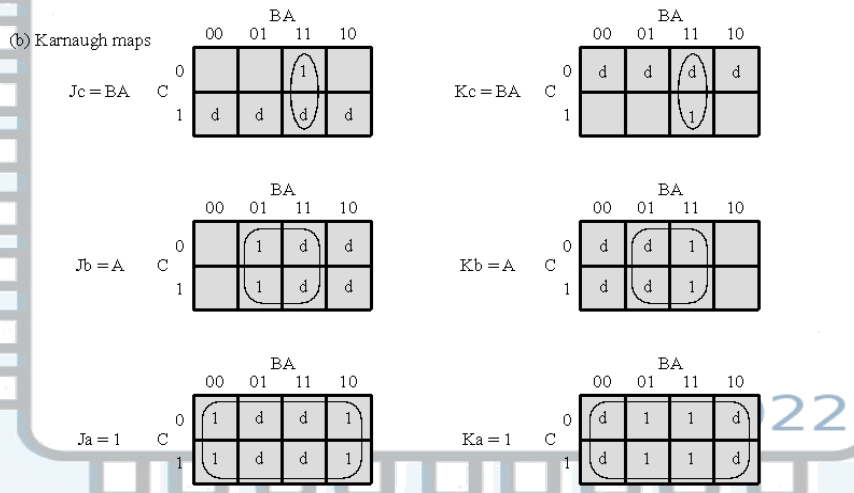
Figure 12.33
Design of a Synchronous Counter



(a) Truth table

C	B	A	Jc	Kc	Jb	Kb	Ja	Ka
0	0	0	0	d	0	d	1	d
0	0	1	0	d	1	d	d	1
0	1	0	0	d	d	0	1	d
0	1	1	1	d	d	1	d	1
1	0	0	d	0	0	d	1	d
1	0	1	d	0	1	d	d	1
1	1	0	d	0	d	0	1	d
1	1	1	d	1	d	1	d	1

(b) Karnaugh maps



(c) Logic diagram

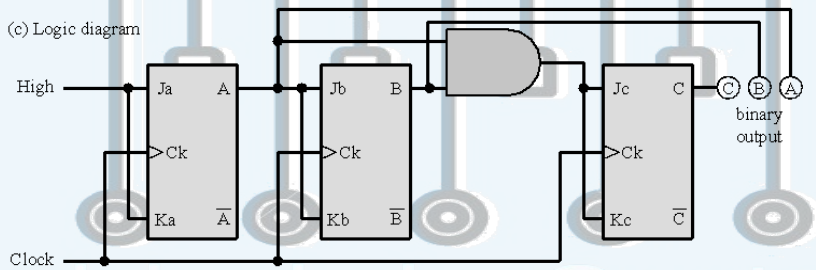


Table 12.13
PLD Terminology

Programmable Logic Device (PLD)
A general term that refers to any type of integrated circuit used for implementing digital hardware, where the chip can be configured by the end user to realize different designs. Programming of such a device often involves placing the chip into a special programming unit, but some chips can also be configured "in-system." Also referred to as a field- programmable device (FPD).

Programmable Logic Array (PLA)
A relatively small PLD that contains two levels of logic, an AND-plane and an OR- plane, where both levels are programmable.

Programmable Array Logic (PAL)
A relatively small PLD that has a programmable AND-plane followed by a fixed OR-plane.

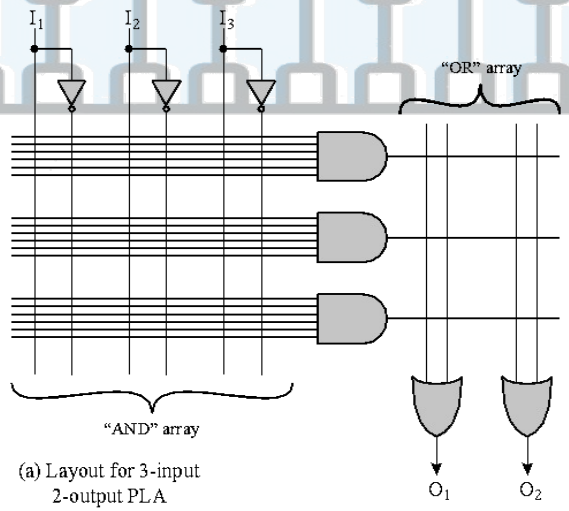
Simple PLD (SPLD)
A PLA or PAL.

Complex PLD (CPLD)
A more complex PLD that consists of an arrangement of multiple SPLD-like blocks on a single chip.

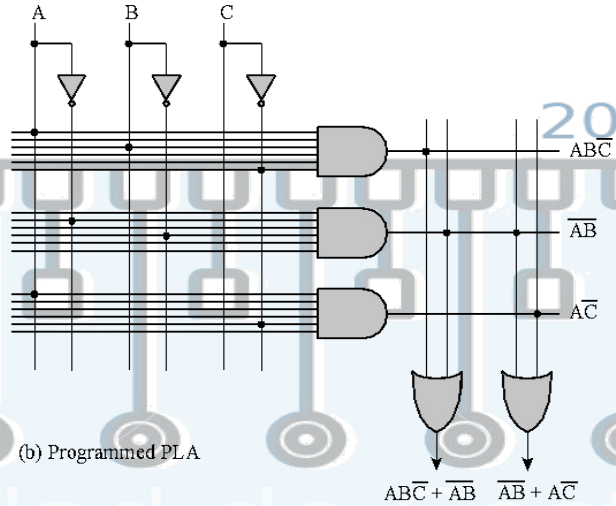
Field- Programmable Gate Array (FPGA)
A PLD featuring a general structure that allows very high logic capacity. Whereas CPLDs feature logic resources with a wide number of inputs (AND planes), FPGAs offer more narrow logic resources. FPGAs also offer a higher ratio of flip- flops to logic resources than do CPLDs.

Logic Block
A relatively small circuit block that is replicated in an array in an FPD. When a circuit is implemented in an FPD, it is first decomposed into smaller subcircuits that can each be mapped into a logic block. The term *logic block* is mostly used in the context of FPGAs, but it could also refer to a block of circuitry in a CPLD.

Figure 12.34 An Example of a Programmable Logic Array (PLA)



(a) Layout for 3-input 2-output PLA



(b) Programmed PLA

Figure 12.35
Structure of an FPGA

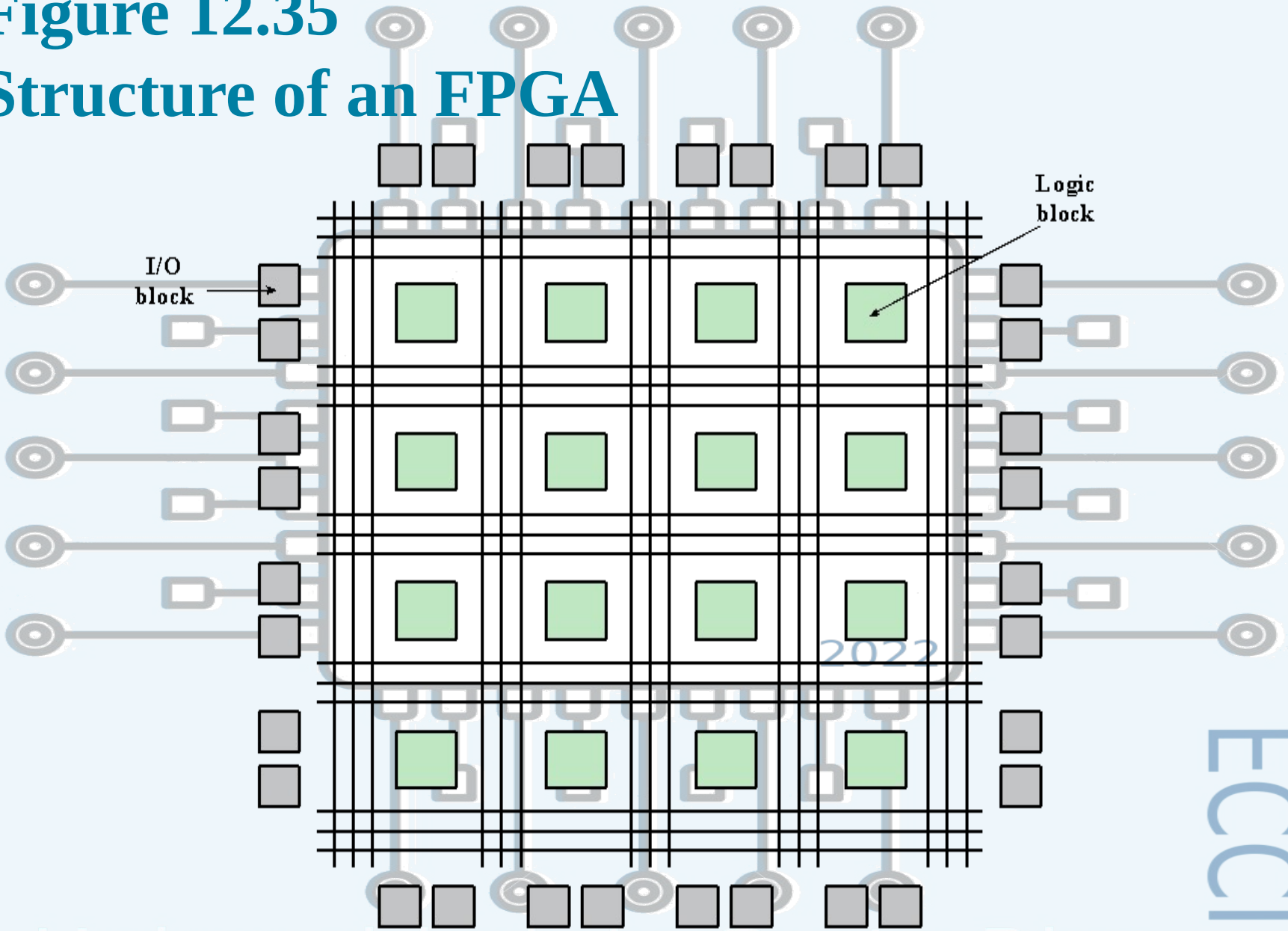
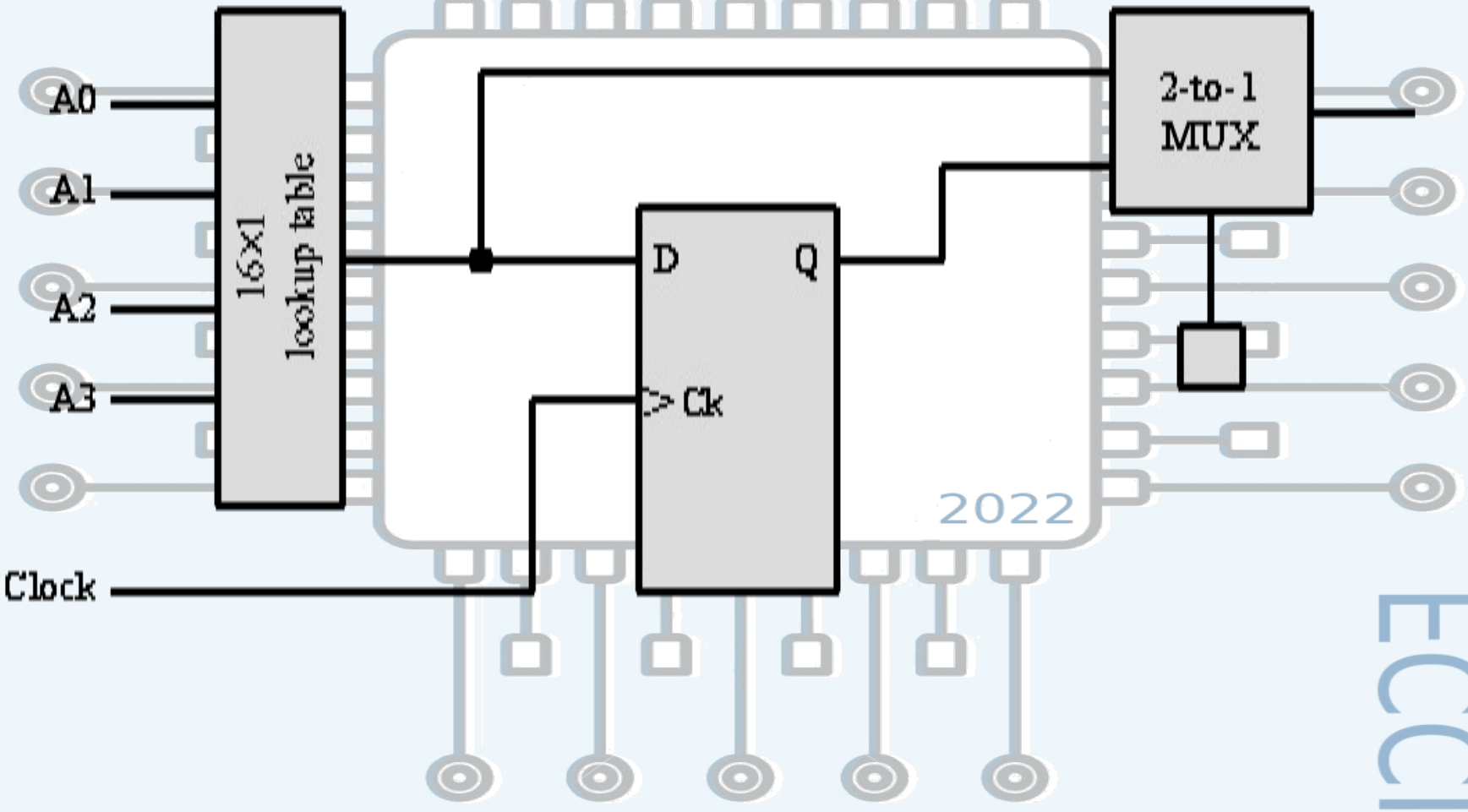


Figure 12.36
A Simple FPGA Logic Block



Summary

Chapter 12

Digital Logic

- Boolean Algebra
- Gates
- Combinational Circuits
  - Implementation of Boolean Functions
  - Multiplexers
  - Decoders
  - Read-Only-Memory
  - Adders
- Sequential Circuits
  - Flip-Flops
  - Registers
  - Counters
- Programmable Logic Devices
  - Programmable Logic Array
  - Field-Programmable Gate Array

2022

ECCI

Universidad de Costa Rica

Copyright