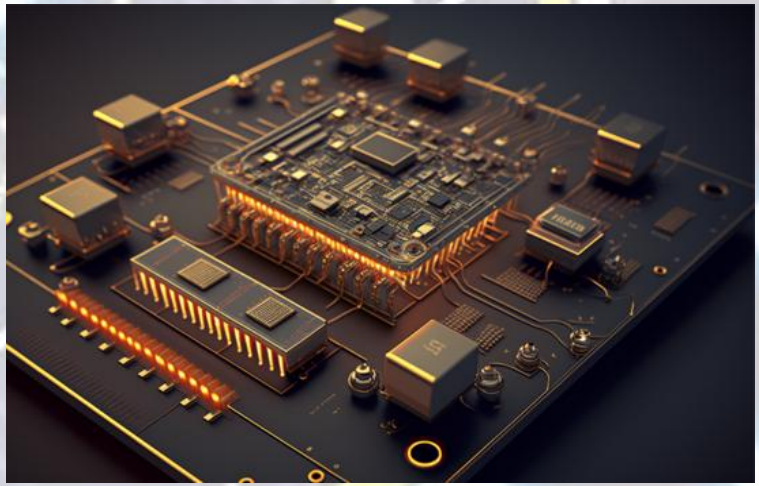
University of Global Village (UGV), Barishal

Dept. of Electrical and Electronic Engineering (EEE)

Digital Electronics (EEE 0714-2101)

Digital Logic Design (CSE 0611-1201)

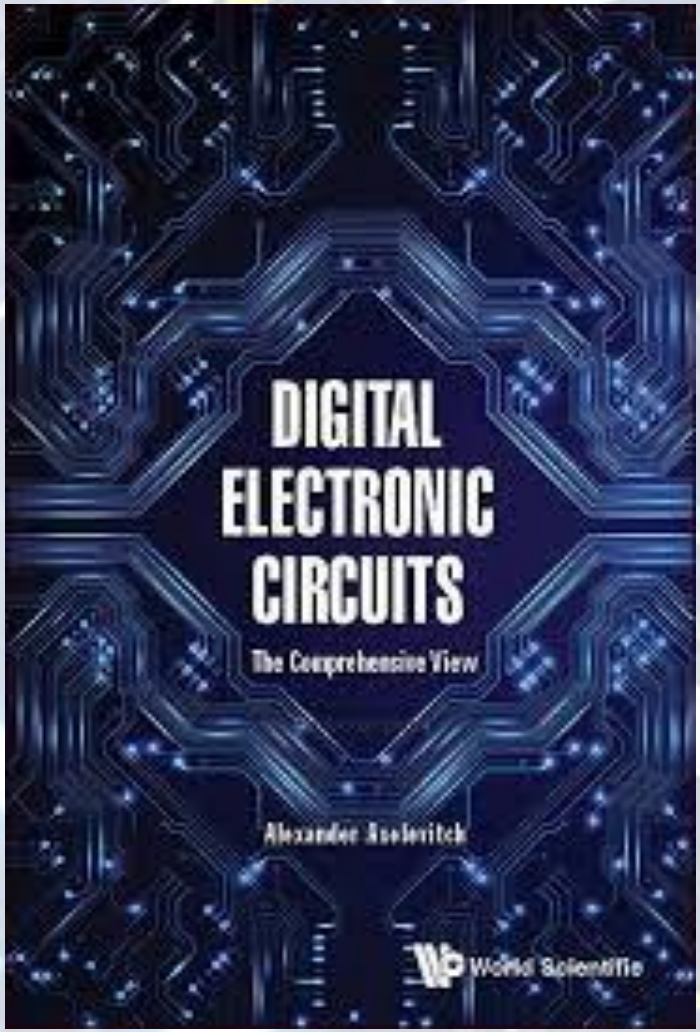
Noor Md Shahriar
Senior Lecturer, Deputy Head of Dept.
Dept. of Electrical & Electronic Engineering
University of Global Village, (UGV), Barishal
Contact: 01743-500587
E-mail: noor.shahriar1@gmail.com



'Imagination is more important than knowledge

- Albert Einstein

Basic Course Information



Course Title	Digital Electronics/ Digital Logic Design
Course Code	EEE- 0714-3107
Credits	03
CIE Marks	90
SEE Marks	60
Exam Hours	2 hours (Mid Exam) 3 hours (Semester Final Exam)
Level	4th Semester
Academic Session	Summer 2025

Digital Electronics **(EEE-0714-2109)**

3 Credit Course

Class:	**17 weeks (2 classes per week)** **Total Class Duration: 1 hrs.** **Total=34 Hours**
Preparation Leave (PL):	02 weeks
Exam:	04 weeks
Results:	02 weeks
Total:	**25 Weeks**

Attendance:

Students with more than or equal to 70% attendance in this course will be eligible to sit for the Semester End Examination (SEE). SEE is mandatory for all students.

Continuous Assessment Strategy

Quizzes

Altogether 4 quizzes may be taken during the semester, 2 quizzes will be taken for midterm and 2 quizzes will be taken for final term.

Assignment

Altogether 2 assignments may be taken during the semester, 1 assignments will be taken for midterm and 1 assignments will be taken for final term.

Presentation

The students will have to form a group of maximum 3 members. The topic of the presentation will be given to each group and students will have to do the group presentation on the given topic.

ASSESSMENT PATTERN

CIE- Continuous Internal Evaluation (90 Marks)

Bloom's Category Marks (out of 90)	Tests (45)	Quizzes (15)	External Participation in Curricular/Co-Curricular Activities (15)
Remember	08	08	Bloom's Affective Domain: (Attitude or will) Attendance: 15 Copy or attempt to copy: −10 Late Assignment: -10
Understand	08	07	
Apply	08		
Analyze	08		
Evaluate	08		
Create	05		

SEE- Semester End Examination (60 Marks)

Bloom's Category	Tests
Remember	**10**
Understand	**10**
Apply	**10**
Analyze	**10**
Evaluate	**10**
Create	**10**

Course Learning Outcome (CLO)

Serial No.	Course Learning Outcome (CLO)
CLO-1	**Understand** and **recall** the process of minimization through K-mapping and tabular method.
CLO-2	**Analyze** and **construct** combinational circuits and sequential circuits using Logic Gates.
CLO-3	**Explain** and **Examine** memory elements using circuits.
CLO-4	**Construct** combinational and sequential circuits through VHDL by understanding dataflow, behavioral and structural modeling, synthesis and simulation of both circuits.

SYNOPSIS / RATIONALE

- The **Digital Electronics** course provides EEE students with fundamental knowledge of digital systems, covering topics like Boolean algebra, logic gates, combinational and sequential circuits, and flip-flops. It equips students with the skills to analyze, design, and optimize digital circuits, which are essential in modern computing, communication systems, automation, and embedded technologies. By bridging theoretical concepts with practical applications, the course prepares students for advanced studies in microprocessors, VLSI, and digital signal processing, ensuring they are ready to tackle industry challenges and innovate in emerging technologies.

Course objectives

- To introduce the fundamental concepts of digital logic and Boolean algebra.
- To enable students to analyze and design combinational circuits using logic gates.
- To provide an understanding of sequential circuits, flip-flops, and their applications.
- To develop the ability to implement and troubleshoot digital circuits in practical scenarios.
- To prepare students for advanced topics like microprocessors, VLSI, and digital signal processing.
- To enhance problem-solving skills for designing optimized digital systems used in modern electronics and communication technologies.

Digital Logic Design | **Lectures: 3 hours/week**

Digital Electronics | **Credits: 3**

Serial No.	Content of Course	Hours	CLOs
1	**Analysis and Synthesis of Digital Logic Circuits**: Number system, codes, and conversion. Boolean algebra, De Morgan's law, logic gates and truth tables, combinational logic design, minimization techniques, implementation of basic static logic gates in CMOS and BiCMOS.	9	CLO-1, CLO-2
2	Arithmetic and data handling logic circuits, decoders and encoders, multiplexers and combinational circuit design.	8	CLO-2, CLO-3
3	**Programmable Logic Devices**: Logic arrays, Field Programmable Logic Arrays, and Programmable Read Only Memory.	8	CLO-3, CLO-4
4	**Sequential Circuits**: Different types of latches, flip-flops and their design using ASM approach, timing analysis, and power optimization of sequential circuits.	9	CLO-3, CLO-4
5	Modular sequential logic circuit design: Shift registers, counters and their applications.	8	CLO-4

Course Schedule

Course plan specifying content, CLOs, teaching learning and assessment strategy mapped with CLOs

Week	Content of Course	ASG/ Quiz/ Pr	Teaching-Learning Strategy	Assessment Strategy	Corres-ponding CLOs
1	Introduction to Digital Electronics, Basic idea about Analog and Digital signals. Details about various types of number systems.		Lecture, Discussion	Written Exam, Class Participation	CLO-1
	Converting base of integer and fractional numbers from one number system to another.		Lecture, Group Examples	Classwork, Problem Solving	CLO-1
2	Data Representation and Complements.	Quiz-1	Lecture, Visual Aids, Group Discussion	Quiz, Written Exam	CLO-1
3	Addition and Subtraction operation of Binary, Octal & Hexadecimal Numbers.	ASG	Lecture, Practice Problems	Assignment, Problem Solving	CLO-2
4	Negative binary number representation in various methods and basic idea about complements. Subtraction of other number systems using Radix and Diminished Radix complement.		Lecture, Group Problem Solving	Written Exam, Group Discussion	CLO-2

Course Schedule (Contd.)

5	Introduction to different types of binary codes. Weighted codes, Gray code, ASCII code, and error-detecting code.	Quiz-2	Lecture, Case Studies, Problem Practice	Quiz, Problem-Solving	CLO-3
6	Definition of Boolean algebra, Boolean theorems, and De-Morgan's theorem. Simplification using theorems.	Assign ment	Lecture, Hands-on Examples, Practical Work	Assignment, Oral Presentation	CLO-3
7	Simplification of Boolean Algebra, Properties & K-Map Method		Lecture, Board Work, Practical Examples	Problem Solving, Classwork	CLO-4
8	Binary Logic, AND, OR, NOT, NAND, NOR, X-OR, and X-NOR gates. Formation of Boolean algebra using universal gates	Quiz-3	Lecture, Case Studies, Practical Work	Quiz, Problem Solving	CLO-3
9	Gate Level Minimization, Boolean Functions, Truth Table, Canonical Forms		Lecture, Group Activities, Hands-on Examples	Written Exam, Practical Tasks	CLO-

Course Schedule (Contd.)

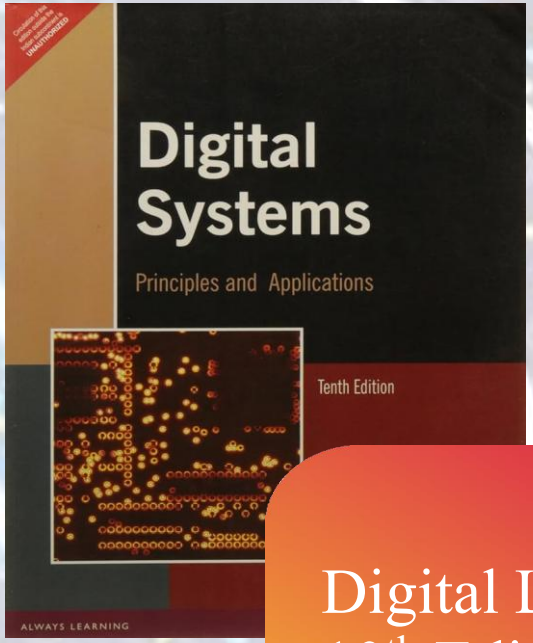
Week	Content of Course	ASG /Quiz/Pr	Teaching-Learning Strategy	Assessment Strategy	Corres-pondin g CLOs
10	Combinational Logic Analysis, K-Map with Don't Care Conditions	Mid-Term Exam	Lecture, Board Examples, Group Practice	Mid-Term Exam	CLO-4
11	BCD to excess 3 and Seven Segment Decoder		Lecture, Problem-Solving Activities	Written Exam, Problem Solving	CLO-5
12	Half adder, full adder, and subtractor design, encode, decoder	ASG	Lecture, Visual Aids, Practical Examples	Assignment, Oral Presentation	CLO-5
13	Design of Muxtiplexer , demultiplexer		Lecture, Group Examples, Case Studies	Written Exam, Group Problem Solving	CLO-5

Course Schedule (Contd.)

14	SR, D, JK and T flipflops ,Master-Slave flip-flop and Edge Triggered circuits. Conversion of Flip-flops.		Lecture, Practical Examples, Group Discussion	Problem Solving, Assignment	CLO-5
15	State Table, State Diagram, Mealy and Moore machines.	Quiz-4	Lecture, Case Studies, Group Problem Solving	Quiz, Written Exam	CLO-5
16	Counters: Asynchronous and Synchronous Counters, Up/Down Counters.		Lecture, Practical Examples, Visual Aids	Assignment, Written Exam	CLO-6
17	Ring Counter, Johnson Counter, Design of Sequential Circuits.		Lecture, Group Activities, Hands-on Examples	Problem Solving, Practical Tasks	CLO-6

REFERENCE BOOK

Digital Design 6th Edition By Morris Mano



Digital Logic Design 10th Edition By Tocci

YouTube

Video Lecture Playlist

https://youtube.com/playlist?list=PLbfLO9aEfT5fpYgbTlQKxo4jiwbEADayG&si=1ay1sarJsBxnPnqV

Week-1

Slide (16-35)

Basic Definition

✅ **Digital Logic Design** is the study and implementation of electronic circuits that process information using **binary digits (0 and 1)**, known as **digital signals**.
It involves:

- Designing circuits using **logic gates** (AND, OR, NOT, etc.).
- Understanding **Boolean algebra.**
- Constructing systems like **adders**, **multiplexers**, **counters**, **memory units**, and **finite state machines.**

Why study this subject?

Skill / Knowledge	Enabled By Digital Logic
System-level understanding	Learn how code runs on hardware
Problem-solving and design	Logic circuit design mirrors algorithm development
Embedded & IoT development	Essential for microcontroller interfacing
Hardware programming	Verilog/FPGA, VLSI, HDL-based design
Secure system design	Digital lock, authentication, cryptographic hardware
Performance optimization	Custom logic for acceleration (e.g., ML chips, DSP blocks)

Applications of Digital Electronics / Logic Design

- **Consumer Electronics**
  - Digital watches and clocks
  - Washing machines, microwave ovens, remote controls
  - Smart TVs, audio systems
- **Computers and Embedded Systems**
  - CPU and GPU architecture
  - Memory (RAM/ROM) management
  - Instruction decoders and control units
- **Communication Systems**
  - Modulation/demodulation logic (e.g., QAM, FSK)
  - Error detection and correction circuits (parity, Hamming code)
  - Multiplexers in channel selection

Applications of Digital Electronics / Logic Design

- **Automotive Systems**
  - Engine control units (ECUs)
  - Parking sensors and collision avoidance systems
  - Digital dashboards and infotainment units
- **Medical Devices**
  - Digital thermometers and ECG machines
  - Patient monitoring systems
  - Diagnostic imaging control circuits
- **Industrial Automation**
  - Programmable Logic Controllers (PLCs)
  - Robotic control logic
  - Conveyor belt and sorting system control

Applications of Digital Electronics / Logic Design

- **Security Systems**
  - Digital locks and access control
  - Motion detection and alarm logic
  - Biometric interface logic (FPGA-based)
- **Aerospace and Defense**
  - Flight control systems (redundant FSM-based designs)
  - Radar signal processing
  - Secure communication protocols
- **Networking and Data Centers**
  - Packet routing and switching logic
  - Data buffering and FIFO/LIFO logic circuits
  - Traffic prioritization algorithms (digital logic level)

Applications of Digital Electronics / Logic Design

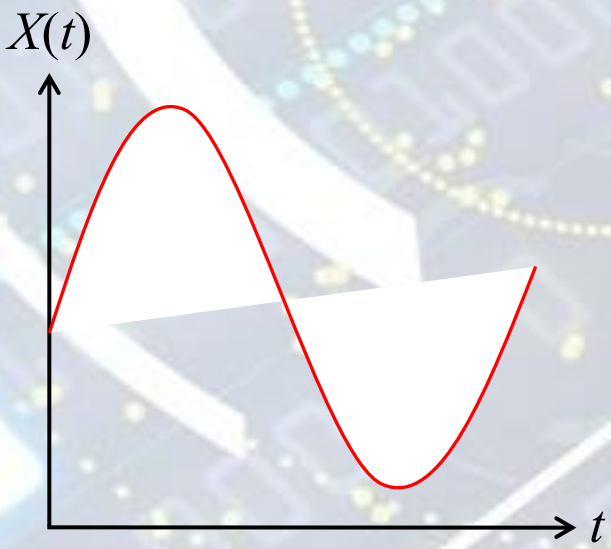
- **Signal and Image Processing**
  - Digital filters and convolution logic
  - Real-time image edge detection
  - Compression algorithms (DCT, Huffman coding)
- **Internet of Things (IoT)**
  - Sensor interfacing and signal conditioning
  - Power-efficient logic control
  - Edge computing controllers
- **Gaming and Entertainment**
  - Gamepad/button logic
  - Display controllers (LCD/LED driving logic)
  - Audio synthesis and timing circuits

Outline of Chapter 1

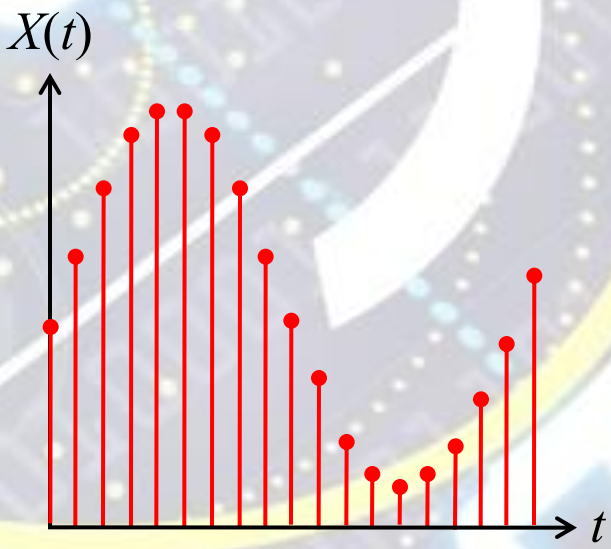
- 1.1 Digital Systems
- 1.2 Binary Numbers
- 1.3 Number-base Conversions
- 1.4 Octal and Hexadecimal Numbers
- 1.5 Complements
- 1.6 Signed Binary Numbers
- 1.7 Binary Codes
- 1.8 Binary Storage and Registers
- 1.9 Binary Logic

Analog and Digital Signal

- Analog system
  - The physical quantities or signals may vary continuously over a specified range.
- Digital system
  - The physical quantities or signals can assume only discrete values.
  - Greater accuracy



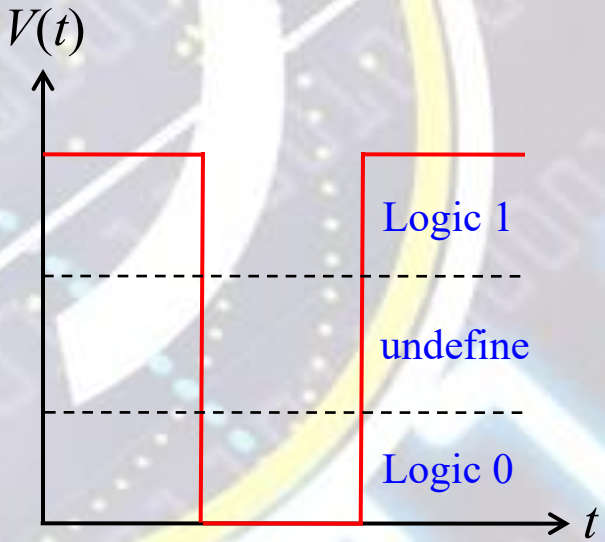
Analog signal



Digital signal

Binary Digital Signal

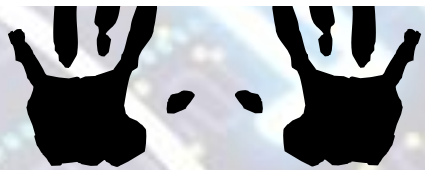
- An information variable represented by physical quantity.
- For digital systems, the variable takes on discrete values.
  - Two level, or binary values are the most prevalent values.
- Binary values are represented abstractly by:
  - Digits 0 and 1
  - Words (symbols) False (F) and True (T)
  - Words (symbols) Low (L) and High (H)
  - And words On and Off
- Binary values are represented by values or ranges of values of physical quantities.
- Why Digital
  - Digital circuits are inexpensive
  - Easy to reduce noise
  - Great flexibility in the design.

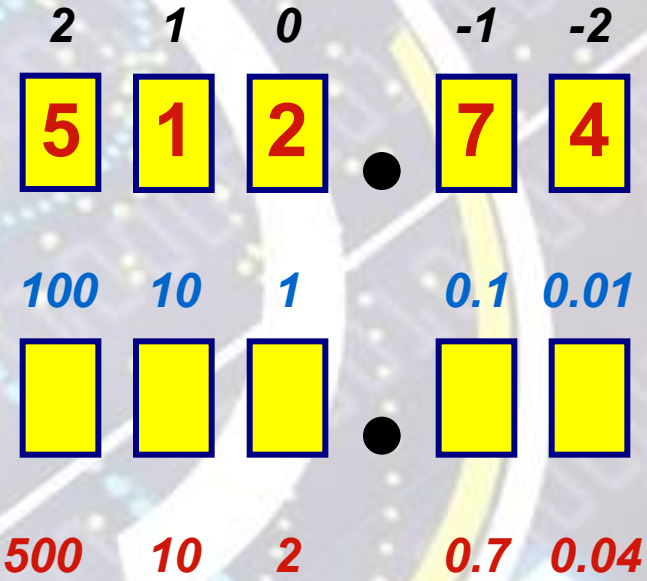


Binary digital signal

Decimal Number System

- Base (also called radix) = 10
  - 10 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
- Digit Position
  - Integer & fraction
- Digit Weight
  - Weight = $(Base)^{Position}$
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation



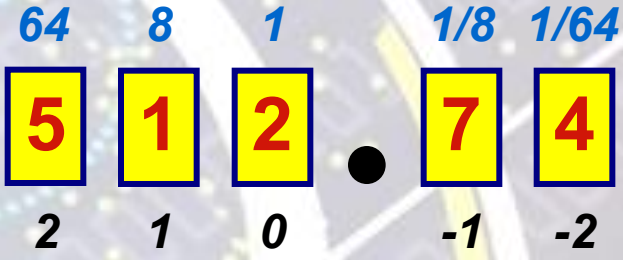


$$d_2*B^2+d_1*B^1+d_0*B^0+d_{-1}*B^{-1}+d_{-2}*B^{-2}$$

$$(512.74)_{10}$$

Octal Number System

- Base = 8
  - 8 digits { 0, 1, 2, 3, 4, 5, 6, 7 }
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation



$$5*8^{2}+1*8^{1}+2*8^{0}+7*8^{-1}+4*8^{-2}$$

$$=(330.9375)_{10}$$

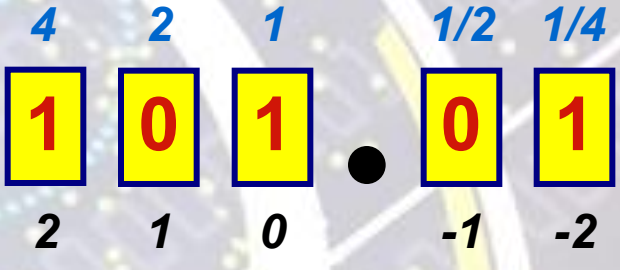
$$(512.74)_{8}$$

Binary Number System

- Base = 2
  - 2 digits { 0, 1 }, called *b*inary dig*its* or "*bits*"
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Bit* x *Weight*"
- Formal Notation
- Groups of bits

4 bits = *Nibble*

8 bits = *Byte*



$1_2 * 2^2 + 0 * 2^1 + 1 * 2^0 + 0 * 2^{-1} + 1 * 2^{-}$

$= (5.25)_{10}$

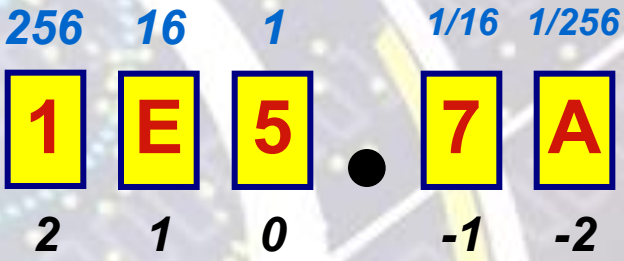
$(101.01)_2$

1 0 1 1

1 1 0 0 0 1 0 1

Hexadecimal Number System

- Base = 16
  - 16 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation



$1*16^{2}+14*16^{1}+5*16^{0}+7*16^{-1}+10*16^{-2}$

$=(485.4765625)_{10}$

$(1E5.7A)_{16}$

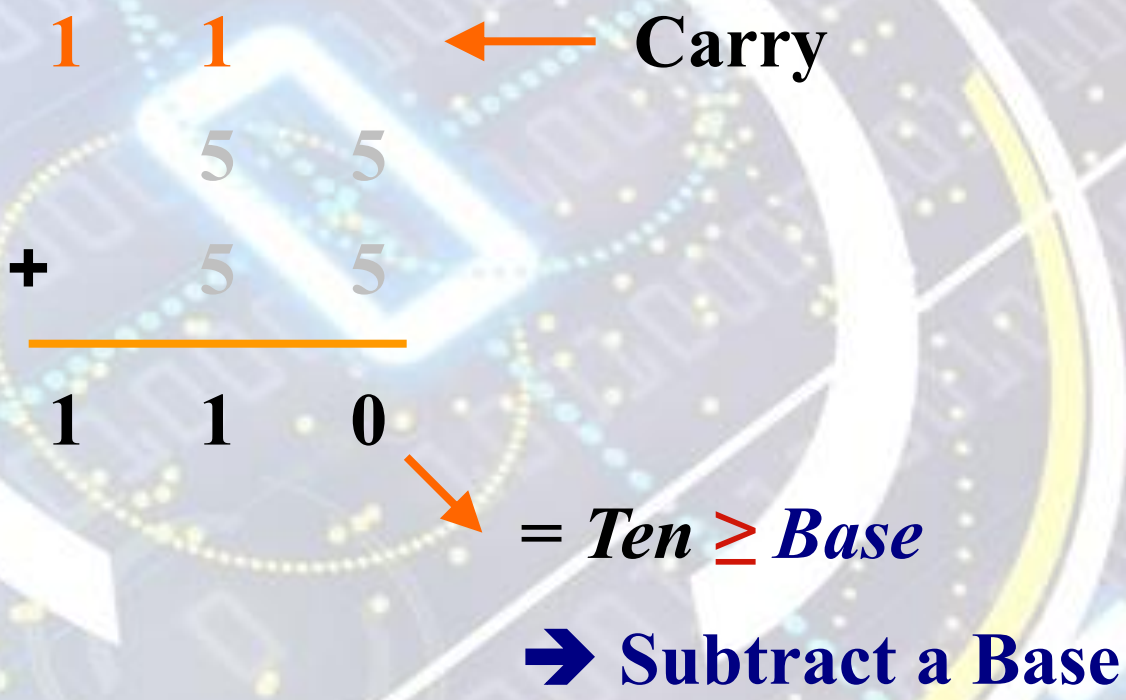
The Power of 2

n	2^n
0	$2^0=1$
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
4	$2^4=16$
5	$2^5=32$
6	$2^6=64$
7	$2^7=128$

n	2^n	
8	$2^8=256$	
9	$2^9=512$	
10	$2^{10}=1024$	Kilo
11	$2^{11}=2048$	
12	$2^{12}=4096$	
20	$2^{20}=1M$	Mega
30	$2^{30}=1G$	Giga
40	$2^{40}=1T$	Tera

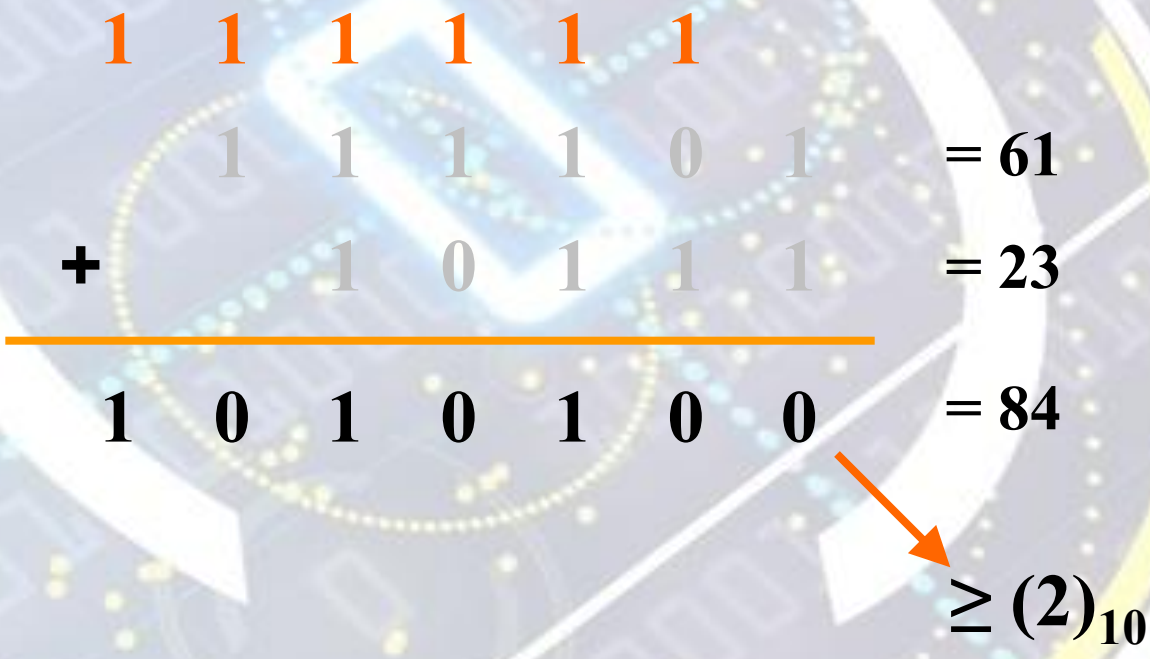
Addition

- Decimal Addition



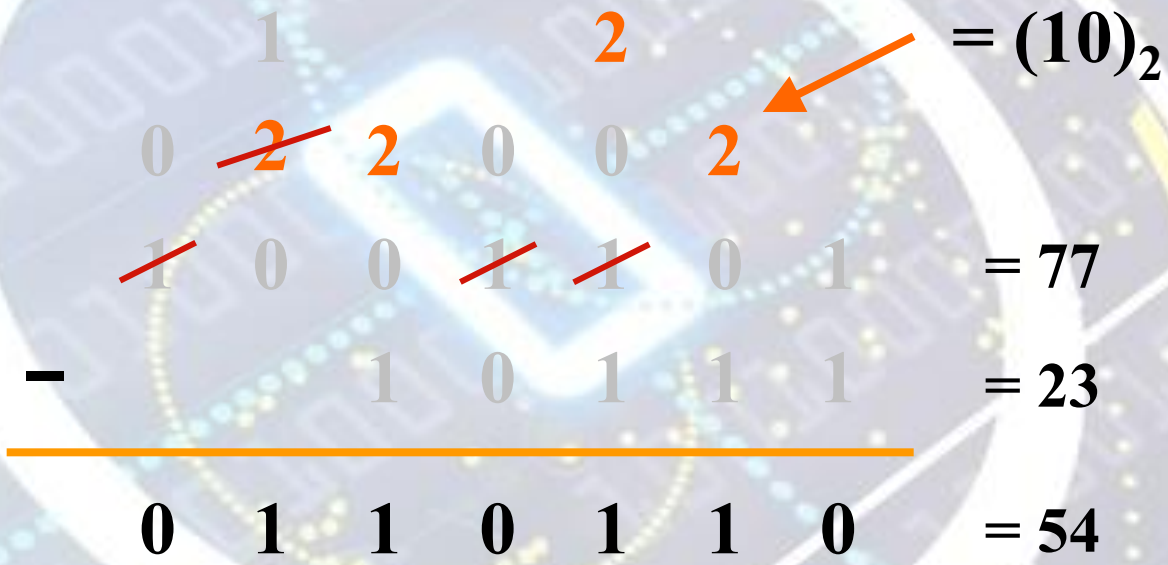
Binary Addition

- Column Addition



Binary Subtraction

- Borrow a "Base" when needed



Binary Multiplication

- Bit by bit

			1	0	1	1	1
x				1	0	1	0
			0	0	0	0	0
		1	0	1	1	1	
	0	0	0	0	0		
1	0	1	1	1			
1	1	1	0	0	1	1	0

Binary Division

- Bit by bit

```
                ) 1  1  0  0  0  ( 1 1
÷                  -  1  1  1
-------------------------------------
                   0  1  0  1  0
                   -  1  1  1
-------------------------------------
                   0  0  0  1  1
```

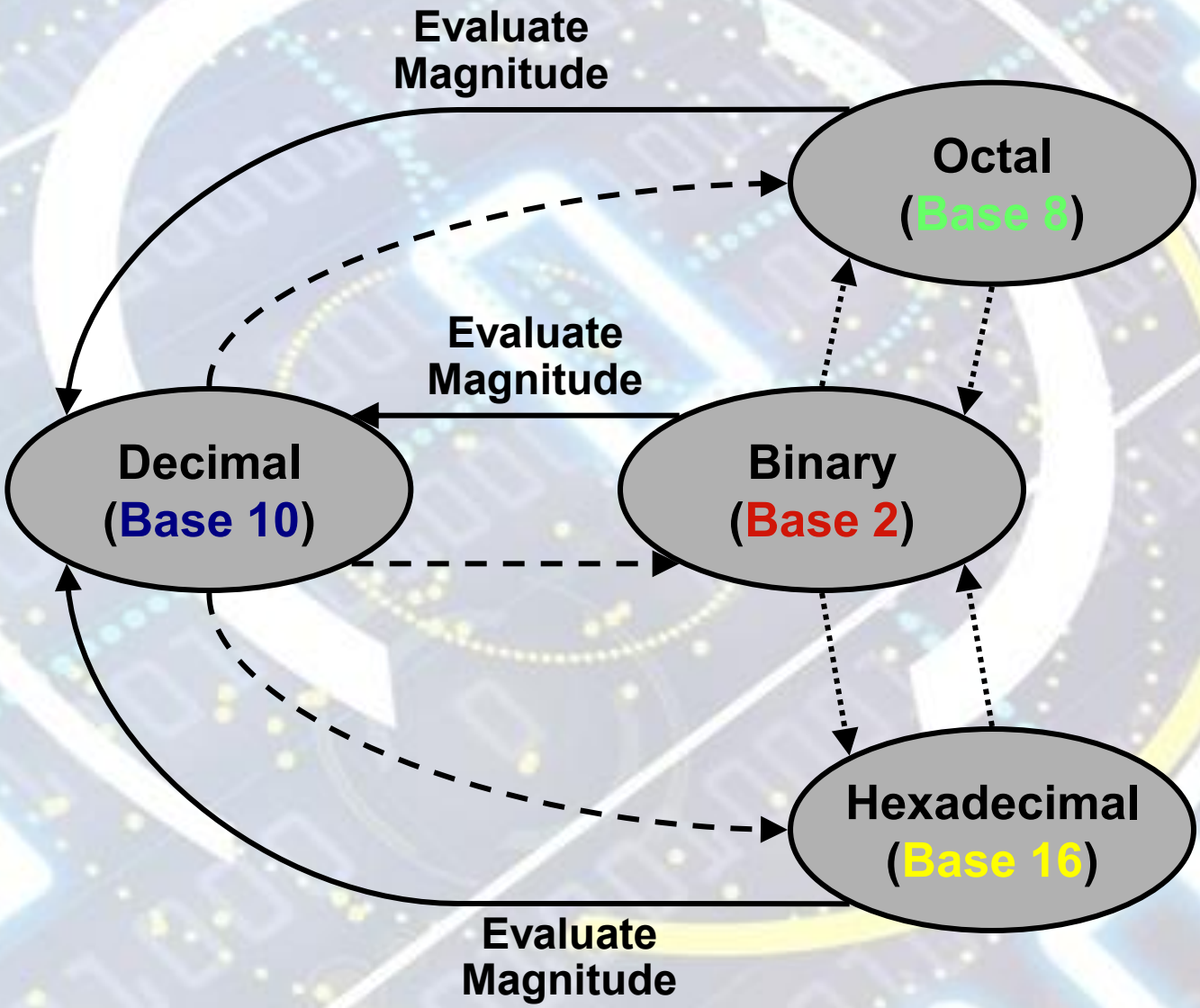
Reminder

Quotient value

Q= 1 1

R= 1 1

Number Base Conversions



Week -2

Page(37-45)

Decimal (*Integer*) to Binary Conversion

- Divide the number by the 'Base' (=2)
- Take the remainder (either 0 or 1) as a coefficient
- Take the quotient and repeat the division

Example: $(13)_{10}$

	Quotient	Remainder	Coefficient
13 / 2 =	6	1	$a_0 = 1$
6 / 2 =	3	0	$a_1 = 0$
3 / 2 =	1	1	$a_2 = 1$
1 / 2 =	0	1	$a_3 = 1$

Answer: $(13)_{10} = (a_3\, a_2\, a_1\, a_0)_2 = (1101)_2$

MSB LSB

Decimal (*Fraction*) to Binary Conversion

- Multiply the number by the 'Base' (=2)
- Take the integer (either 0 or 1) as a coefficient
- Take the resultant fraction and repeat the division

Example: $(0.625)_{10}$

		Integer		Fraction	Coefficient
0.625	* 2 =	1	.	25	$a_{-1} = 1$
0.25	* 2 =	0	.	5	$a_{-2} = 0$
0.5	* 2 =	1	.	0	$a_{-3} = 1$

Answer: $(0.625)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_2 = (0.101)_2$

MSB LSB

Decimal to Octal Conversion

Example: $(175)_{10}$

	Quotient	Remainder	Coefficient
175 / 8 =	21	7	$a_0 = 7$
21 / 8 =	2	5	$a_1 = 5$
2 / 8 =	0	2	$a_2 = 2$

Answer: $(175)_{10} = (a_2\, a_1\, a_0)_8 = (257)_8$

Example: $(0.3125)_{10}$

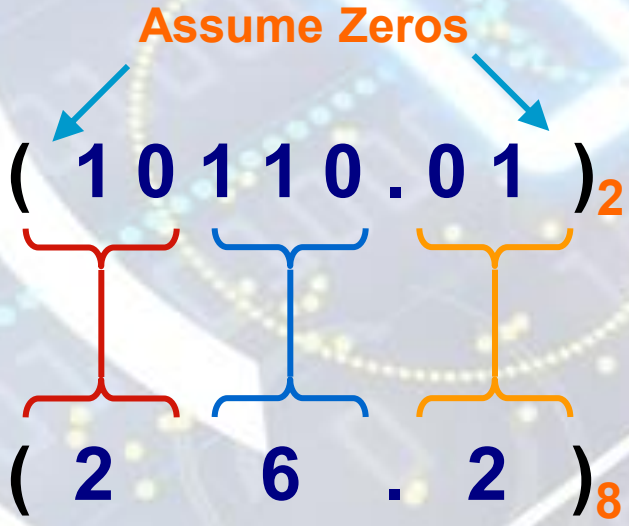
	Integer		Fraction	Coefficient
0.3125 * 8 =	2	.	5	$a_{-1} = 2$
0.5 * 8 =	4	.	0	$a_{-2} = 4$

Answer: $(0.3125)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_8 = (0.24)_8$

Binary – Octal Conversion

- $8 = 2^3$
- Each group of 3 bits represents an octal digit

Example:



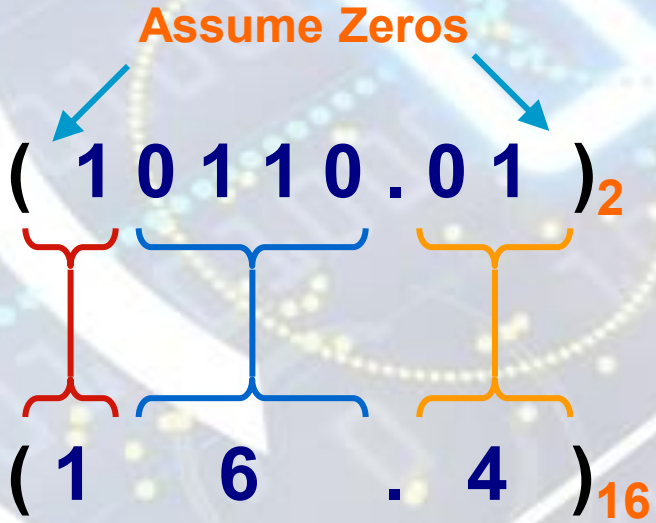
Octal	Binary
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Works both ways (*Binary* to *Octal* & *Octal* to *Binary*)

Binary – Hexadecimal Conversion

- $16 = 2^4$
- Each group of 4 bits represents a hexadecimal digit

Example:



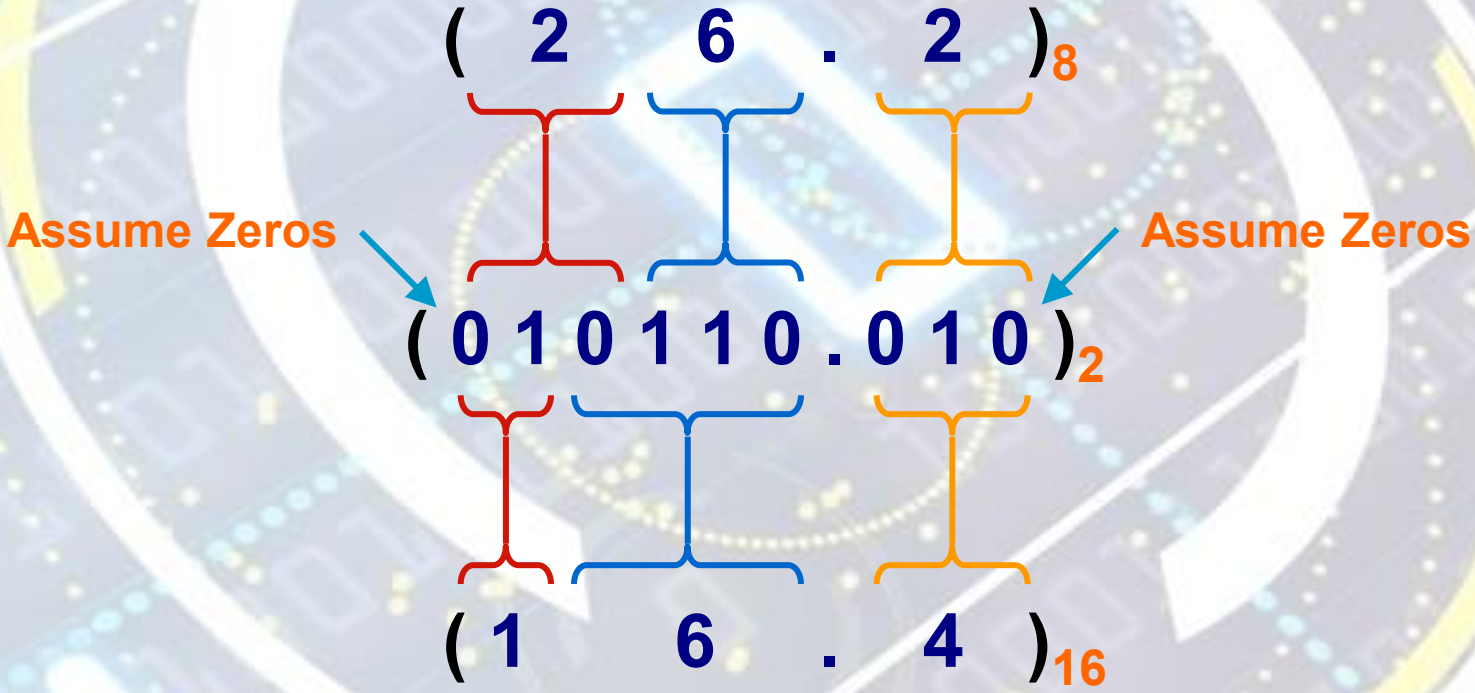
Hex	Binary
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

Works both ways (*Binary* to *Hex* & *Hex* to *Binary*)

Octal – Hexadecimal Conversion

- Convert to Binary as an intermediate step

Example:



Works both ways (*Octal* to *Hex* & *Hex* to *Octal*)

Decimal, Binary, Octal and Hexadecimal

Decimal	Binary	Octal	Hex
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1.5 Complements

- There are two types of complements for each base-*r* system: the radix complement and diminished radix complement.
- **Diminished Radix Complement - (r-1)'s Complement**
  - Given a number *N* in base *r* having *n* digits, the (*r*–*1*)'s complement *of N* is defined as:

$$(r^n - 1) - N$$

- **Example for 6-digit decimal numbers**:
  - 9's complement is $(r^n - 1) - N = (10^6 - 1) - N = 999999 - N$
  - 9's complement of 546700 is 999999–546700 = 453299
- **Example for 7-digit binary numbers:**
  - 1's complement is $(r^n - 1) - N = (2^7 - 1) - N = 1111111 - N$
  - 1's complement of 1011000 is 1111111–1011000 = 0100111
- **Observation:**
  - Subtraction from $(r^n - 1)$ will never require a borrow
  - Diminished radix complement can be computed digit-by-digit
  - For binary: 1 – 0 = 1 and 1 – 1 = 0

Complements

- 1's Complement (*Diminished Radix* Complement)
  - All '0's become '1's
  - All '1's become '0's

  Example $(10110000)_2$

  ⇨ $(01001111)_2$

  If you add a number and its 1's complement …

$$
\begin{array}{r}
1\ 0\ 1\ 1\ 0\ 0\ 0\ 0 \\
+\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1 \\
\hline
1\ 1\ 1\ 1\ 1\ 1\ 1\ 1
\end{array}
$$

Week -3

Page(47-56)

Complements

- Radix Complement

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and as 0 for $N = 0$. Comparing with the $(r - 1)$'s complement, we note that the r's complement is obtained by adding 1 to the $(r - 1)$'s complement, since $r^n - N = [(r^n - 1) - N] + 1$.

- Example: Base-10

The 10's complement of 012398 is 987602
The 10's complement of 246700 is 753300

- Example: Base-2

The 2's complement of 1101100 is 0010100
The 2's complement of 0110111 is 1001001

Complements

- 2's Complement (*Radix* Complement)
  - Take 1's complement then add 1

OR

  - Toggle all bits to the left of the first '1' from the right

Example:

Number: **1 0 1 1 0 0 0 0**

1's Comp.: **0 1 0 0 1 1 1 1**

+ 1

---

0 1 0 1 0 0 0 0

1 0 1 1 0 0 0 0

0 1 0 1 0 0 0 0

Complements

- Subtraction with Complements
  - The subtraction of two n-digit unsigned numbers $M - N$ in base r can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. Mathematically, $M + (r^n - N) = M - N + r^n$.
2. If $M \geqq N$, the sum will produce and end carry r^n, which can be discarded; what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

Complements

- Example 1.5
  - Using 10's complement, subtract 72532 – 3250.

	$M =$	72532
10's complement of	$N =$	+96750
	Sum =	169282
	Discard end carry 10^5 =	–100000
	Answer =	69282

- Example 1.6
  - Using 10's complement, subtract 3250 – 72532.

	M =	03250
10's complement of	N =	+27468
	Sum =	30718

→ There is no end carry.

→ Therefore, the answer is – (10's complement of 30718) = – 69282.

Complements

- Example 1.7
  - Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$; and (b) $Y - X$, by using 2's complement.

(a)		
	X =	1010100
	2's complement of Y =	+0111101
	Sum =	10010001
	Discard end carry 2^7 =	−10000000
	Answer. X − Y =	0010001

(b)		
	Y =	1000011
	2's complement of X =	+ 0101100
	Sum =	1101111

There is no end carry. Therefore, the answer is Y − X = − (2's complement of 1101111) = − 0010001.

Complements

- Subtraction of unsigned numbers can also be done by means of the $(r-1)$'s complement. Remember that the $(r-1)$'s complement is one less then the r's complement.
- Example 1.8
  - Repeat Example 1.7, but this time using 1's complement.

(a) $X - Y = 1010100 - 1000011$

$X =$	1010100
1's complement of $Y =$	± 0111100
Sum =	10010000
End-around carry =	+ 1
Answer. $X - Y =$	0010001

(b) $Y - X = 1000011 - 1010100$

$Y =$	1000011
1's complement of $X =$	+ 0101011
Sum =	1101110

There is no end carry, Therefore, the answer is Y – X = – (1's complement of 1101110) = – 0010001.

1.6 Signed Binary Numbers

- To represent negative integers, we need a notation for negative values.
- It is customary to represent the sign with a bit placed in the leftmost position of the number since binary digits.
- The convention is to make the sign bit 0 for positive and 1 for negative.
- Example:

Signed-magnitude representation:	10001001
Signed-1's-complement representation:	11110110
Signed-2's-complement representation:	11110111

- Table 1.3 lists all possible four-bit signed binary numbers in the three representations.

Signed Binary Numbers

Table 1.3
Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

Signed Binary Numbers

- Arithmetic addition
  - The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic. **If the signs are the same**, we add the two magnitudes and give the sum the common sign. **If the signs are different**, we subtract the smaller magnitude from the larger and give the difference the sign if the larger magnitude.
  - The addition of two signed binary numbers with negative numbers represented in signed-2's-complement form is obtained from the addition of the two numbers, including their sign bits.
  - A carry out of the sign-bit position is discarded.
- Example:

+ 6	00000110	− 6	11111010
+13	00001101	+13	00001101
+ 19	00010011	+ 7	00000111
+ 6	00000110	− 6	11111010
−13	11110011	−13	11110011
− 7	11111001	− 19	11101101

Signed Binary Numbers

- Arithmetic Subtraction
  - In 2's-complement form:

1. Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including sign bit).
2. A carry out of sign-bit position is discarded.

$$(\pm A)-(+B)=(\pm A)+(-B)$$
$$(\pm A)-(-B)=(\pm A)+(+B)$$

- Example:

$(-6)-(-13)$ → $(11111010-11110011)$

→ $(11111010+00001101)$

→ $00000111\ (+7)$

Week -4

Page(58-80)

1.7 Binary Codes

- BCD Code
  - A number with k decimal digits will require 4k bits in BCD.
  - Decimal 396 is represented in BCD with 12bits as 0011 1001 0110, with each group of 4 bits representing one decimal digit.
  - A decimal number in BCD is the same as its equivalent binary number only when the number is between 0 and 9.
  - The binary combinations 1010 through 1111 are not used and have no meaning in BCD.

Table 1.4
Binary-Coded Decimal (BCD)

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

Binary Code

- Example:
  - Consider decimal 185 and its corresponding value in BCD and binary:

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$$

- BCD addition

4	0100	4	0100	8	1000
+5	+0101	+8	+1000	+9	+1001
9	1001	12	1100	17	10001
			+0110		+0110
			10010		10111

Binary Code

- Example:
  - Consider the addition of 184 + 576 = 760 in BCD:

BCD	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
Binary sum	0111	10000	1010	
Add 6		0110	0110	
BCD sum	0111	0110	0000	760

- Decimal Arithmetic: (+375) + (-240) = +135

0	375
+9	760
0	135

Hint 6: using 10's of BCD

Binary Codes

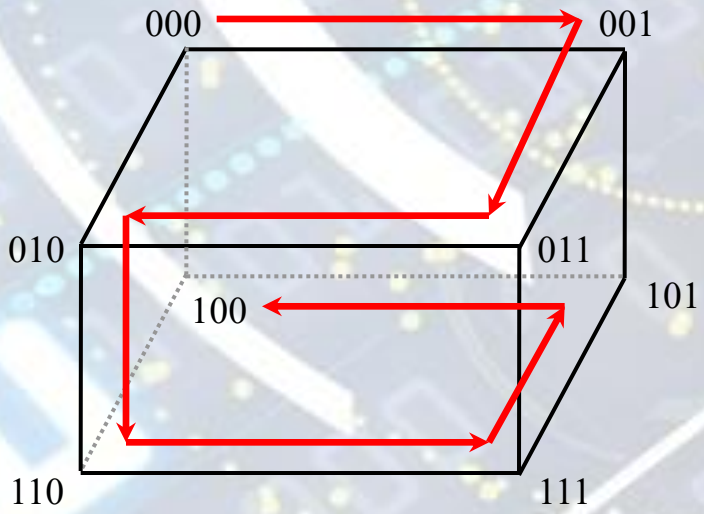
- Other Decimal Codes

Table 1.5
Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
	1010	0101	0000	0001
Unused	1011	0110	0001	0010
bit	1100	0111	0010	0011
combi-	1101	1000	1101	1100
nations	1110	1001	1110	1101
	1111	1010	1111	1110

Binary Codes)

- Gray Code
  - The advantage is that only bit in the code group changes in going from one number to the next.
    - » Error detection.
    - » Representation of analog data.
    - » Low power design.



1-1 and onto!!

Table 1.6
Gray Code

Gray Code	Decimal Equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

Binary Codes

- American Standard Code for Information Interchange (ASCII) Character Code

Table 1.7
American Standard Code for Information Interchange (ASCII)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	−	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	−	o	DEL

Binary Codes

- ASCII Character Code

Control characters

NUL	Null	DLE	Data-link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End-of-transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

ASCII Character Codes

- American Standard Code for Information Interchange (Refer to Table 1.7)
- A popular code used to represent information sent as character-based data.
- It uses 7-bits to represent:
  - 94 Graphic printing characters.
  - 34 Non-printing characters.
- Some non-printing characters are used for text format (e.g. BS = Backspace, CR = carriage return).
- Other non-printing characters are used for record marking and flow control (e.g. STX and ETX start and end text areas).

ASCII Properties

- ASCII has some interesting properties:
  - Digits 0 to 9 span Hexadecimal values 30_{16} to 39_{16}
  - Upper case A-Z span 41_{16} to $5A_{16}$
  - Lower case a-z span 61_{16} to $7A_{16}$
    - » Lower to upper case translation (and vice versa) occurs by flipping bit 6.

Binary Codes

- Error-Detecting Code
  - To detect errors in data communication and processing, an Eighth bit is sometimes added to the ASCII character to indicate its parity.
  - A parity bit is an extra bit included with a message to make the total number of 1's either even or odd.
- Example:
  - Consider the following two characters and their even and odd parity:

	With even parity	With odd parity
ASCII A = 1000001	01000001	11000001
ASCII T = 1010100	11010100	01010100

Binary Codes

- Error-Detecting Code
  - Redundancy (e.g. extra information), in the form of extra bits, can be incorporated into binary code words to detect and correct errors.
  - A simple form of redundancy is parity, an extra bit appended onto the code word to make the number of 1's odd or even. Parity can detect all single-bit errors and some multiple-bit errors.
  - A code word has even parity if the number of 1's in the code word is even.
  - A code word has odd parity if the number of 1's in the code word is odd.
  - Example:

Message A: 100010011 (even parity)

Message B: 100010010 (odd parity)

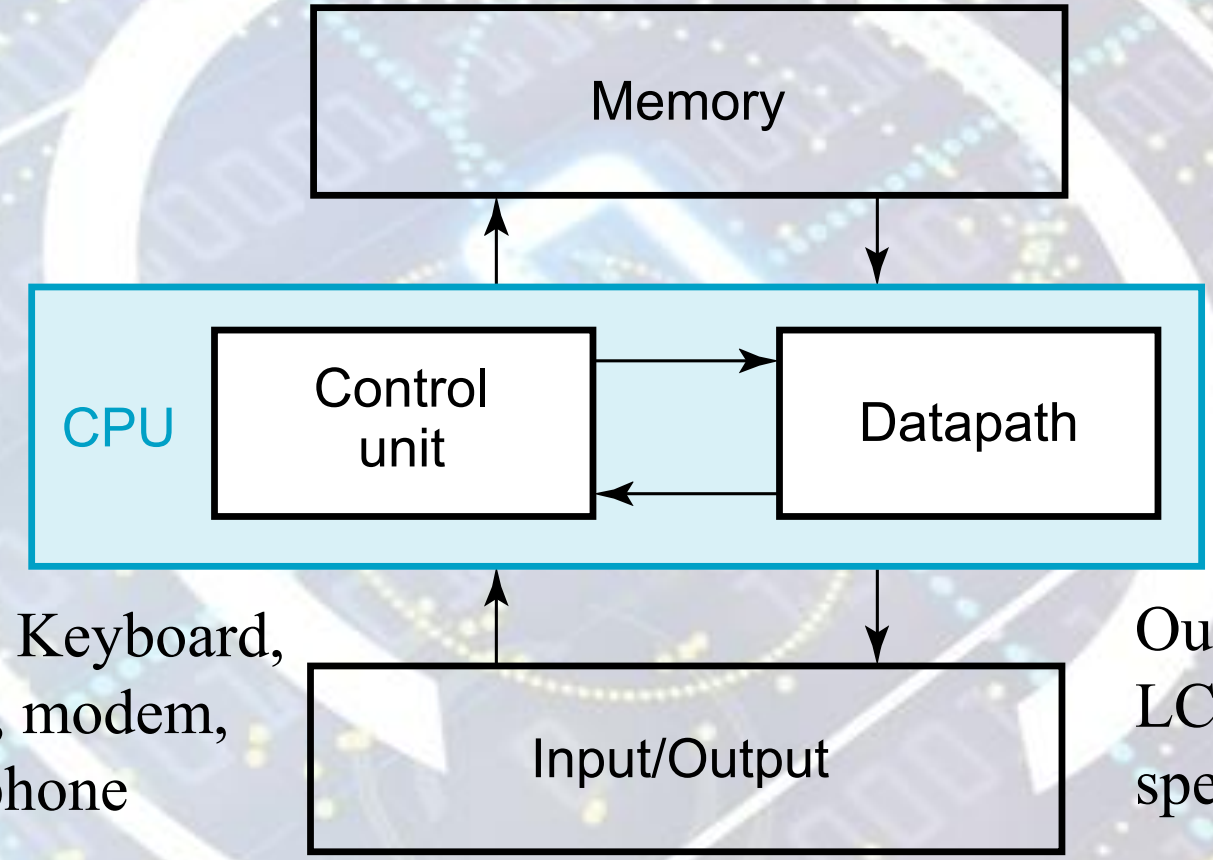
1.8 Binary Storage and Registers

- Registers
  - A binary cell is a device that possesses two stable states and is capable of storing one of the two states.
  - A register is a group of binary cells. A register with *n* cells can store any discrete quantity of information that contains *n* bits.

n cells → 2^n possible states

- A binary cell
  - Two stable state
  - Store one bit of information
  - Examples: flip-flop circuits, ferrite cores, capacitor
- A register
  - A group of binary cells
  - AX in x86 CPU
- Register Transfer
  - A transfer of the information stored in one register to another.
  - One of the major operations in digital system.
  - An example in next slides.

A Digital Computer Example



Inputs: Keyboard, mouse, modem, microphone

Outputs: CRT, LCD, modem, speakers

Synchronous or Asynchronous?

Transfer of information

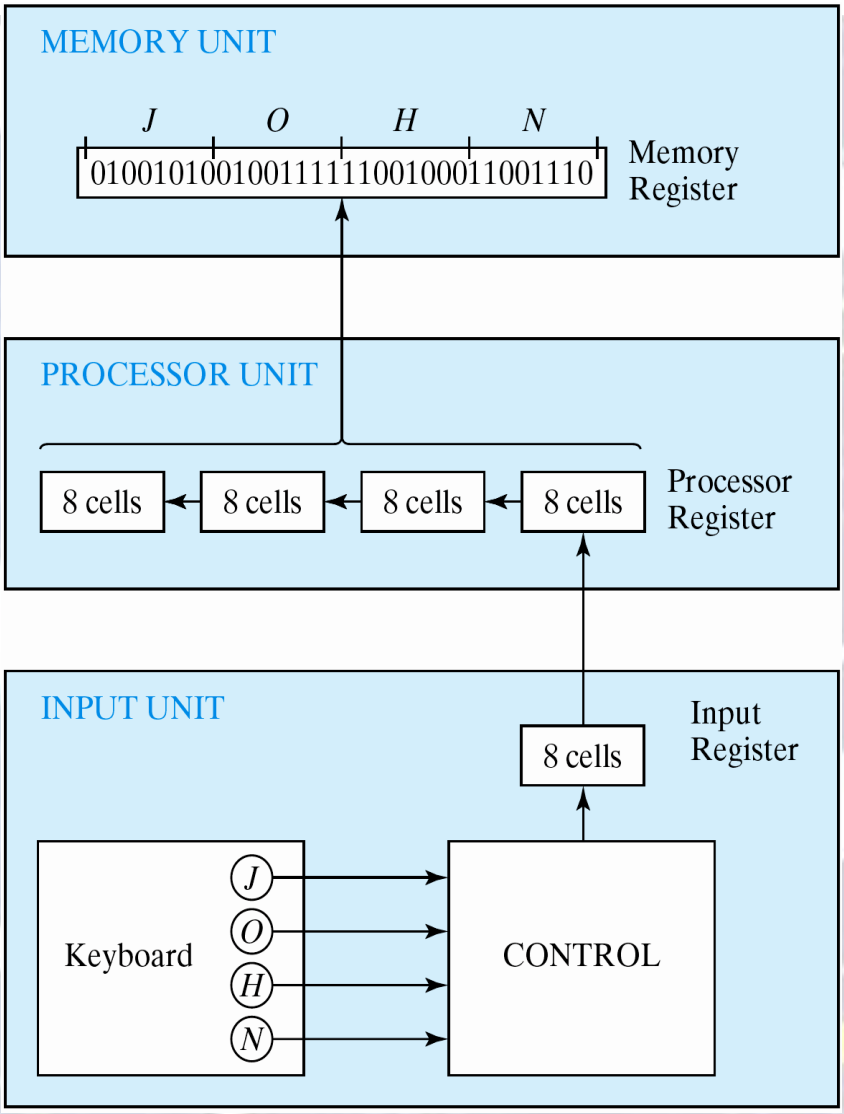


Figure 1.1 Transfer of information among register

Transfer of information

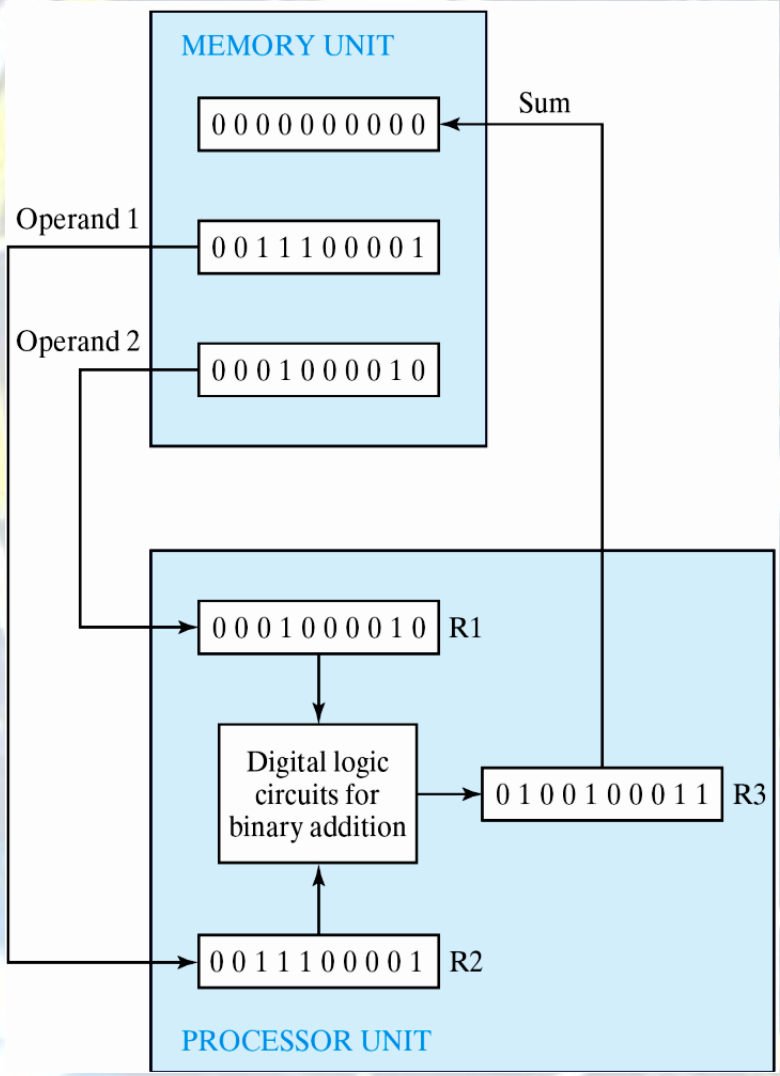


Figure 1.2 Example of binary information processing

- The other major component of a digital system
  - Circuit elements to manipulate individual bits of information
  - Load-store machine

```
LD      R1;
LD      R2;
ADD     R2, R1;
SD      R3;
```

1.9 Binary Logic

- Definition of Binary Logic
  - Binary logic consists of binary variables and a set of logical operations.
  - The variables are designated by letters of the alphabet, such as A, B, C, x, y, z, etc, with each variable having two and only two distinct possible values: 1 and 0,
  - Three basic logical operations: AND, OR, and NOT.

1. AND: This operation is represented by a dot or by the absence of an operator. For example, $x \cdot y = z$ or $xy = z$ is read "x AND y is equal to z," The logical operation AND is interpreted to mean that $z = 1$ if only $x = 1$ and $y = 1$; otherwise $z = 0$. (Remember that x, y, and z are binary variables and can be equal either to 1 or 0, and nothing else.)
2. OR: This operation is represented by a plus sign. For example, $x + y = z$ is read "x OR y is equal to z," meaning that $z = 1$ if $x = 1$ or $y = 1$ or if both $x = 1$ and $y = 1$. If both $x = 0$ and $y = 0$, then $z = 0$.
3. NOT: This operation is represented by a prime (sometimes by an overbar). For example, x' = z (or $\bar{x} = z$) is read "not x is equal to z," meaning that z is what z is not. In other words, if $x = 1$, then $z = 0$, but if $x = 0$, then $z = 1$, The NOT operation is also referred to as the complement operation, since it changes a 1 to 0 and a 0 to 1.

Binary Logic

- Truth Tables, Boolean Expressions, and Logic Gates

AND

x	*y*	*z*
0	0	0
0	1	0
1	0	0
1	1	1

$z = x \cdot y = x\,y$

OR

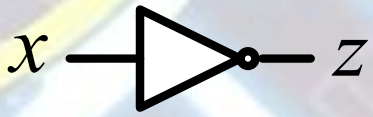
x	*y*	*z*
0	0	0
0	1	1
1	0	1
1	1	1

$z = x + y$

NOT

x	*z*
0	1
1	0

$z = \overline{x} = x'$



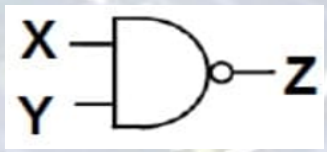
Binary Logic

- Truth Tables, Boolean Expressions, and Logic Gates

NAND

x	*y*	*z*
0	0	1
0	1	1
1	0	1
1	1	0

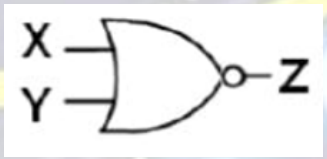
$$z = \overline{(x.y)}$$



NOR

x	*y*	*z*
0	0	1
0	1	0
1	0	0
1	1	0

$$z = \overline{(x + y)}$$



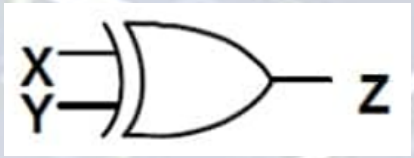
Binary Logic

- Truth Tables, Boolean Expressions, and Logic Gates

XOR

x	*y*	*z*
0	0	0
0	1	1
1	0	1
1	1	0

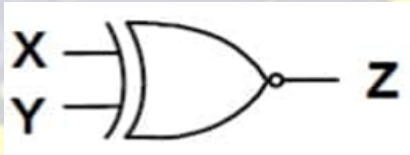
$z = \overline{x}.y + x.\overline{y}$



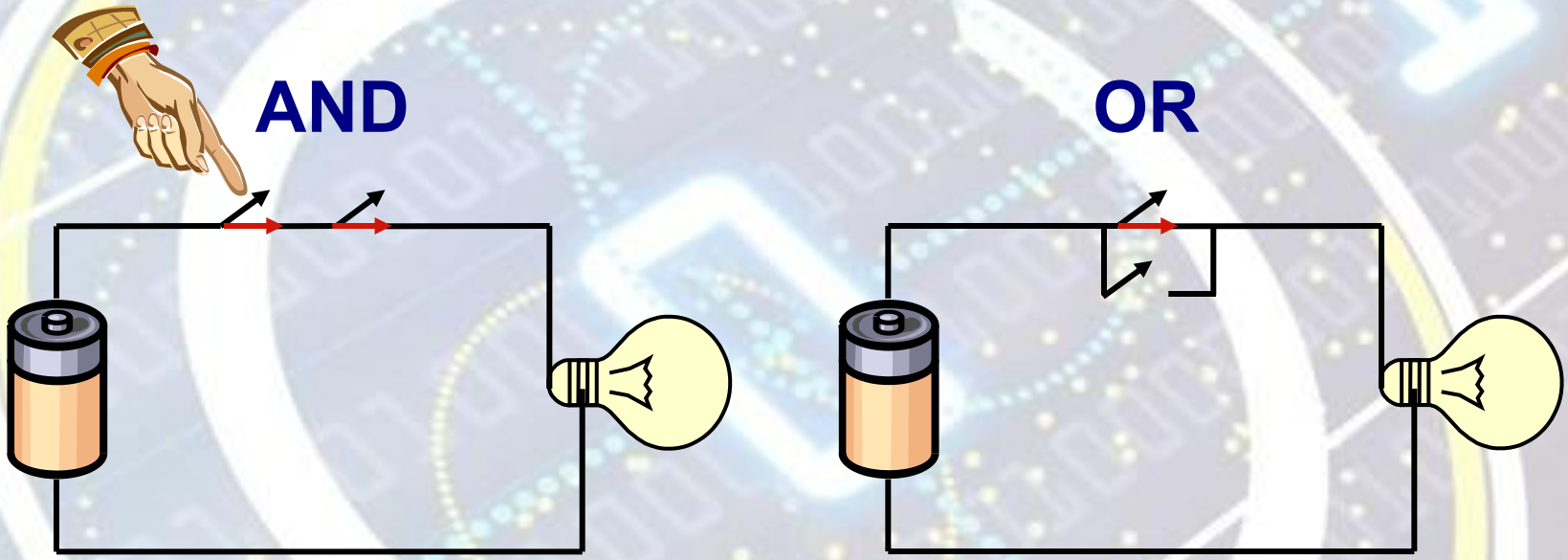
XNOR

x	*y*	*z*
0	0	1
0	1	0
1	0	0
1	1	1

$z = x.y + \overline{x.y}$

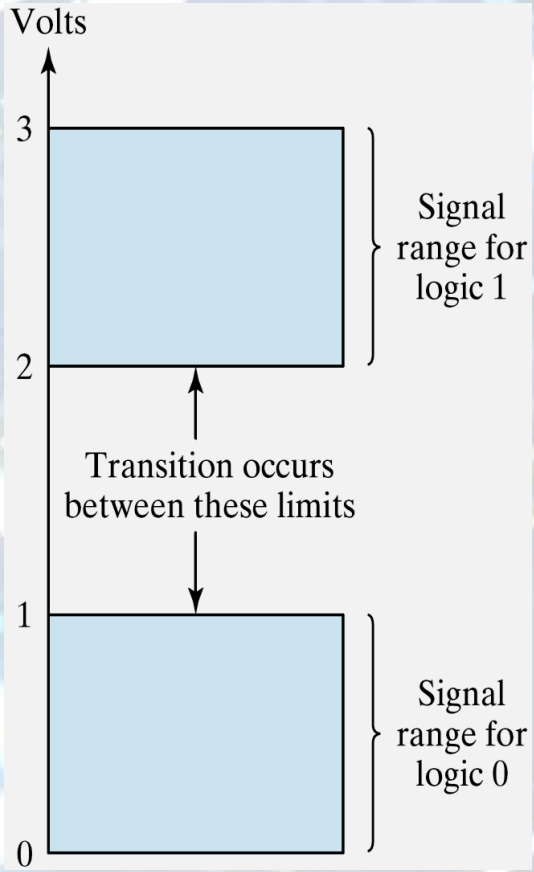


Switching Circuits



Binary Logic

- Logic gates
  - Example of binary signals



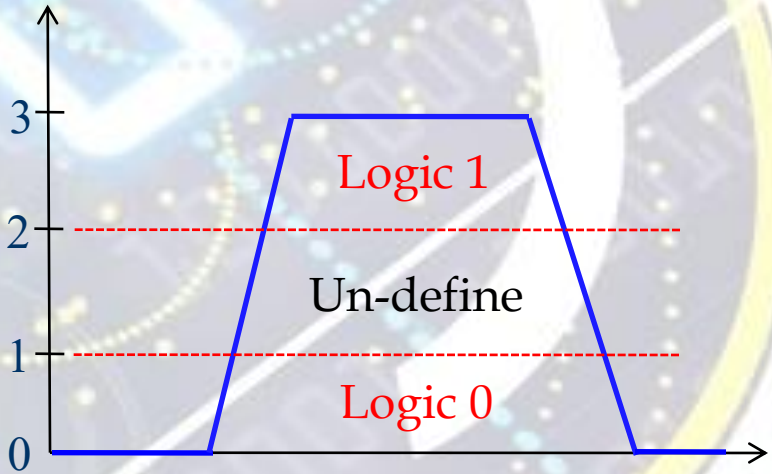
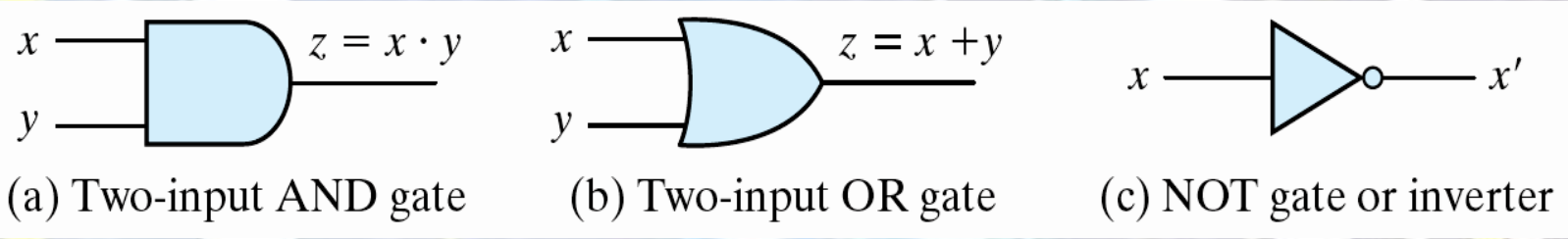


Figure 1.3 Example of binary signals

Binary Logic

- Logic gates
  - Graphic Symbols and Input-Output Signals for Logic gates:



(a) Two-input AND gate (b) Two-input OR gate (c) NOT gate or inverter

Fig. 1.4 Symbols for digital logic circuits

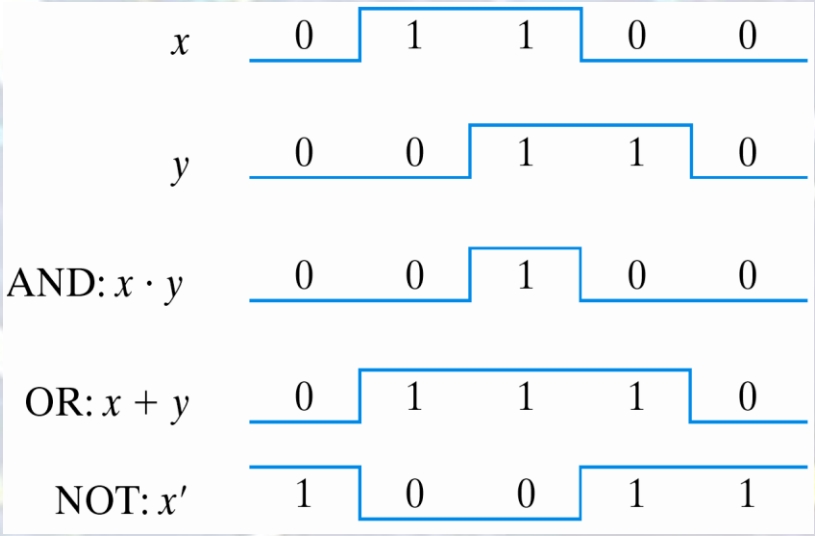


Fig. 1.5 Input-Output signals for gates

Binary Logic

- Logic gates
  - Graphic Symbols and Input-Output Signals for Logic gates:

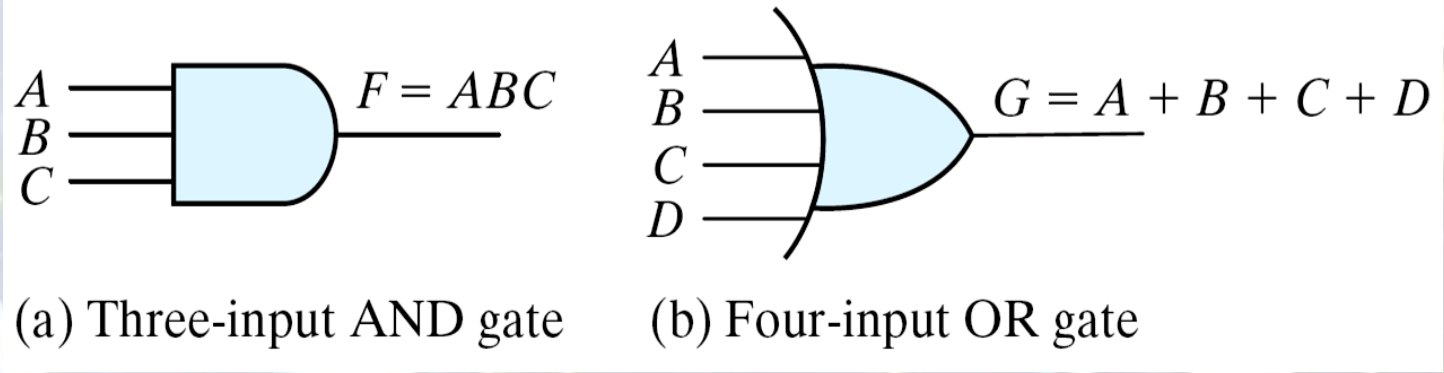


Fig. 1.6 Gates with multiple inputs

Week -6

Page(82-104)

Algebras

- What is an algebra?
  - Mathematical system consisting of
    - » Set of elements
    - » Set of operators
    - » Axioms or postulates
- Why is it important?
  - Defines rules of "calculations"
- Example: arithmetic on natural numbers
  - Set of elements: $N = \{1,2,3,4,\ldots\}$
  - Operator: +, –, *
  - Axioms: associativity, distributivity, closure, identity elements, etc.
- Note: operators with two inputs are called *binary*
  - Does not mean they are restricted to binary numbers!
  - Operator(s) with one input are called *unary*

BASIC DEFINITIONS

- A set is collection of having the same property.
  - S: set, x and y: element or event
  - For example: $S = \{1, 2, 3, 4\}$
    - » If $x = 2$, then $x \in S$.
    - » If $y = 5$, then $y \notin S$.
- A *binary operator* defines on a set S of elements is a rule that assigns, to each pair of elements from S, a unique element from S.
  - For example: given a set S, consider $a*b = c$ and * is a binary operator.
  - We say that, * is a binary operator if it is specifies a rule for finding c from the pair (a, b) and also if $a, b, c \in S$.
  - On the other hand, * is not a binary operator if $a, b \in S$,while the rule finds $c \notin S$.

BASIC DEFINITIONS

- The most common postulates used to formulate various algebraic structures are as follows:

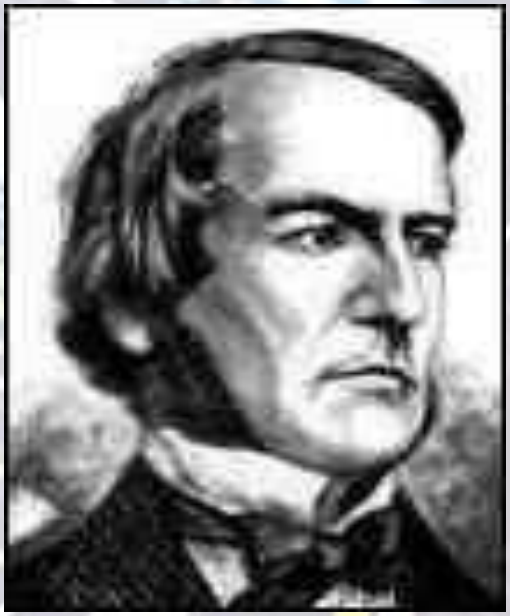
1. Closure: a set S is closed with respect to a binary operator if, for every pair of elements of S, the binary operator specifies a rule for obtaining a unique element of S.
   - *For example, natural numbers $N=\{1,2,3,...\}$ is closed w.r.t. the binary operator + by the rule of arithmetic addition, since, for any $a, b \in N$, there is a unique $c \in N$ such that*
     - $a+b=c$
     - But operator – is not closed for N, because 2-3 = -1 and 2, 3 $\in N$, but $(-1) \notin N$.
2. Associative law: a binary operator * on a set S is said to be associative whenever
   - $(x * y) * z = x * (y * z)$ for all $x, y, z \in S$
     - $(x+y)+z = x+(y+z)$
3. Commutative law: a binary operator * on a set S is said to be commutative whenever
   - $x * y = y * x$ for all $x, y \in S$
     - $x+y = y+x$

BASIC DEFINITIONS

4. *Identity element*: a set S is said to have an identity element with respect to a binary operation * on S if there exists an element e$\in S$ with the property that
   - $e * x = x * e = x$ for every $x \in S$
     - $0+x = x+0 = x$ for every $x \in I$. $I = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$.
     - $1*x = x*1 = x$ for every $x \in I$. $I = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$.
5. *Inverse*: a set having the identity element e with respect to the binary operator to have an inverse whenever, for every $x \in S$, there exists an element $y \in S$ such that
   - $x * y = e$
     - The operator + over I, with $e = 0$, the inverse of an element a is $(-a)$, since $a+(-a) = 0$.
6. *Distributive law*: if * and $\cdot$ are two binary operators on a set S, * is said to be distributive over . whenever
   - $x * (y \cdot z) = (x * y) \cdot (x * z)$

George Boole

- Father of Boolean algebra
- He came up with a type of linguistic algebra, the three most basic operations of which were (and still are) AND, OR and NOT. It was these three functions that formed the basis of his premise, and were the only operations necessary to perform comparisons or basic mathematical functions.
- Boole's system (detailed in his 'An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities', 1854) was based on a binary approach, processing only two objects - the yes-no, true-false, on-off, zero-one approach.
- Surprisingly, given his standing in the academic community, Boole's idea was either criticized or completely ignored by the majority of his peers.
- Eventually, one bright student, Claude Shannon (1916-2001), picked up the idea and ran with it



George Boole (1815 - 1864)

Axiomatic Definition of Boolean Algebra

- We need to define algebra for binary values
  - Developed by George Boole in 1854
- Huntington postulates for Boolean algebra (1904):
- $B = \{0, 1\}$ and two binary operations, + and ·
  - Closure with respect to operator + and operator ·
  - Identity element 0 for operator + and 1 for operator ·
  - Commutativity with respect to + and ·

$$x+y = y+x, \quad x \cdot y = y \cdot x$$

  - Distributivity of · over +, and + over ·

$x \cdot (y+z) = (x \cdot y)+(x \cdot z)$ and $x+(y \cdot z) = (x+y) \cdot (x+z)$

    - Complement for every element x is x' with $x+x'=1$, $x \cdot x'=0$
  - There are at least two elements $x, y \in B$ such that $x \neq y$

Boolean Algebra

- Terminology:
  - *Literal:* A variable or its complement
  - *Product term:* literals connected by •
  - *Sum term:* literals connected by +

Postulates of Two-Valued Boolean Algebra

- $B = \{0, 1\}$ and two binary operations, + and ·
- The rules of operations: AND、OR and NOT.

AND

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT

x	x'
0	1
1	0

1. Closure (+ and·)
2. The identity elements
   (1) +: 0
   (2) · : 1

Postulates of Two-Valued Boolean Algebra

3. The commutative laws
4. The distributive laws

x	y	z	$y+z$	$x \cdot (y+z)$	$x \cdot y$	$x \cdot z$	$(x \cdot y)+(x \cdot z)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Postulates of Two-Valued Boolean Algebra

5. Complement
   - $x+x'=1 \rightarrow 0+0'=0+1=1;\ 1+1'=1+0=1$
   - $x \cdot x'=0 \rightarrow 0 \cdot 0'=0 \cdot 1=0;\ 1 \cdot 1'=1 \cdot 0=0$
6. Has two distinct elements 1 and 0, with $0 \neq 1$

- Note
  - A set of two elements
  - + : OR operation; ・: AND operation
  - A complement operator: NOT operation
  - Binary logic is a two-valued Boolean algebra

Duality

- The principle of *duality* is an important concept. This says that if an expression is valid in Boolean algebra, the dual of that expression is also valid.
- To form the dual of an expression, replace all + operators with . operators, all . operators with + operators, all ones with zeros, and all zeros with ones.
- Form the dual of the expression

  $a + (b.c) = (a + b).(a + c)$
- Following the replacement rules…

  $a.(b + c) = a.b + a.c$
- Take care not to alter the location of the parentheses if they are present.

Basic Theorems

Table 2.1
Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

Boolean Theorems

- Huntington's postulates define some rules

Post. 1: closure
Post. 2: (a) x+0=x, (b) x·1=x
Post. 3: (a) x+y=y+x, (b) x·y=y·x
Post. 4: (a) x(y+z) = xy+xz,
(b) x+yz = (x+y)(x+z)
Post. 5: (a) x+x'=1, (b) x·x'=0

- Need more rules to modify algebraic expressions
  - Theorems that are derived from postulates
- What is a theorem?
  - A formula or statement that is derived from postulates (or other proven theorems)
- Basic theorems of Boolean algebra
  - Theorem 1 (a): $x + x = x$ (b): $x \cdot x = x$
  - Looks straightforward, but needs to be proven !

Proof of $x+x=x$

- We can only use Huntington postulates:

Huntington postulates:

Post. 2: *(a) x+0=x, (b) x·1=x*
Post. 3: *(a) x+y=y+x, (b) x·y=y·x*
Post. 4: *(a) x(y+z) = xy+xz,*
(b) x+yz = (x+y)(x+z)
Post. 5: *(a) x+x'=1, (b) x·x'=0*

- Show that $x+x=x$.

$x+x$	$= (x+x)\cdot 1$	by 2(b)
	$= (x+x)(x+x')$	by 5(a)
	$= x+xx'$	by 4(b)
	$= x+0$	by 5(b)
	$= x$	by 2(a)

Q.E.D.

- We can now use Theorem 1(a) in future proofs

Proof of $x \cdot x = x$

- Similar to previous proof

Huntington postulates:

Post. 2: *(a) x+0=x, (b) x·1=x*
Post. 3: *(a) x+y=y+x, (b) x·y=y·x*
Post. 4: *(a) x(y+z) = xy+xz,*
(b) x+yz = (x+y)(x+z)
Post. 5: *(a) x+x'=1, (b) x·x'=0*
Th. 1: *(a) x+x=x*

- Show that $x \cdot x = x$.

$x \cdot x = xx + 0$ by 2(a)
$= xx + xx'$ by 5(b)
$= x(x + x')$ by 4(a)
$= x \cdot 1$ by 5(a)
$= x$ by 2(b)
Q.E.D.

Proof of $x+1=1$

- Theorem 2(a): $x + 1 = 1$

$$
\begin{aligned}
x + 1 &= 1 \cdot (x + 1) && \text{by 2(b)} \\
&= (x + x')(x + 1) && 5(a) \\
&= x + x' 1 && 4(b) \\
&= x + x' && 2(b) \\
&= 1 && 5(a)
\end{aligned}
$$

Huntington postulates:

Post. 2: *(a) x+0=x, (b) x·1=x*
Post. 3: *(a) x+y=y+x, (b) x·y=y·x*
Post. 4: *(a) x(y+z) = xy+xz,*
(b) x+yz = (x+y)(x+z)
Post. 5: *(a) x+x′=1, (b) x·x′=0*
Th. 1: *(a) x+x=x*

- Theorem 2(b): $x \cdot 0 = 0$ by duality
- Theorem 3: $(x')' = x$
  - Postulate 5 defines the complement of x, $x + x' = 1$ and $x\,x' = 0$
  - The complement of x' is x is also $(x')'$

Absorption Property (Covering)

- Theorem 6(a): $x + xy = x$
  - $x + xy = x \cdot 1 + xy$ *by* 2(b)
    $= x\,(1 + y)$ 4(a)
    $= x\,(y + 1)$ 3(a)
    $= x \cdot 1$ Th 2(a)
    $= x$ 2(b)

Huntington postulates:

Post. 2: *(a) x+0=x, (b) x·1=x*
Post. 3: *(a) x+y=y+x, (b) x·y=y·x*
Post. 4: *(a) x(y+z) = xy+xz,*
(b) x+yz = (x+y)(x+z)
Post. 5: *(a) x+x′=1, (b) x·x′=0*
Th. 1: *(a) x+x=x*

- Theorem 6(b): $x\,(x + y) = x$ by duality
- By means of truth table (another way to proof)

x	*y*	*xy*	*x+xy*
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

DeMorgan's Theorem

- Theorem 5(a): $(x + y)' = x'y'$
- Theorem 5(b): $(xy)' = x' + y'$
- By means of truth table

x	*y*	*x'*	*y'*	*x+y*	*(x+y)'*	*x'y'*	*xy*	*x'+y'*	*(xy)'*
0	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	0	0	1	1
1	0	0	1	1	0	0	0	1	1
1	1	0	0	1	0	0	1	0	0

Consensus Theorem

1. $x.y + x'.z + y.z = x.y + x'.z$
2. $(x+y)\cdot(x'+z)\cdot(y+z) = (x+y)\cdot(x'+z)$ -- (dual)

- **Proof:**

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

QED (2 true by duality).

Operator Precedence

- The operator precedence for evaluating Boolean Expression is
  - Parentheses
  - NOT
  - AND
  - OR
- Examples
  - $x\,y' + z$
  - $(x\,y + z)'$

Boolean Functions

- A Boolean function
  - Binary variables
  - Binary operators OR and AND
  - Unary operator NOT
  - Parentheses
- Examples
  - $F_1 = x\,y\,z'$
  - $F_2 = x + y'z$
  - $F_3 = x'y'z + x'y\,z + x\,y'$
  - $F_4 = x\,y' + x'z$

Boolean Functions

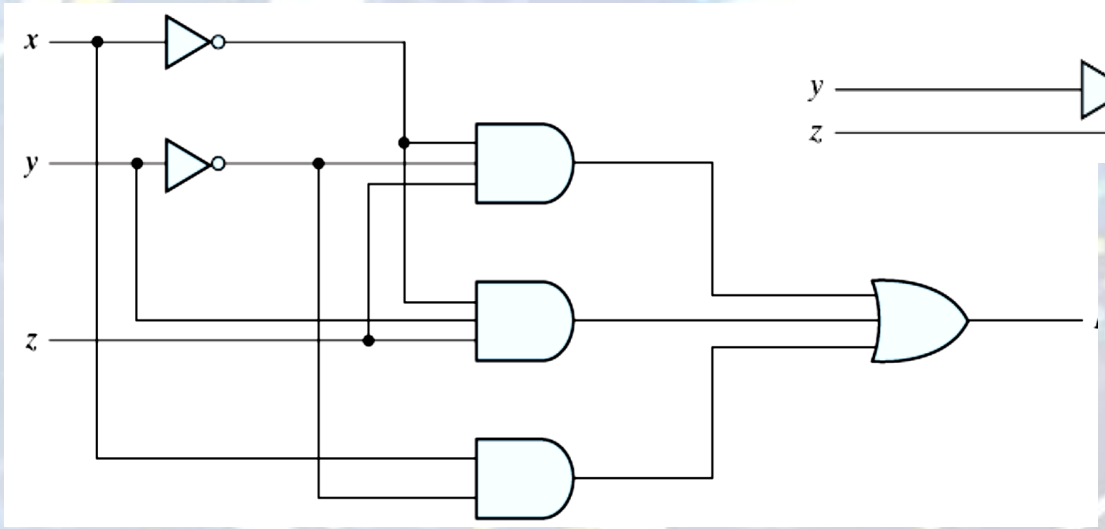
- The truth table of 2^n entries

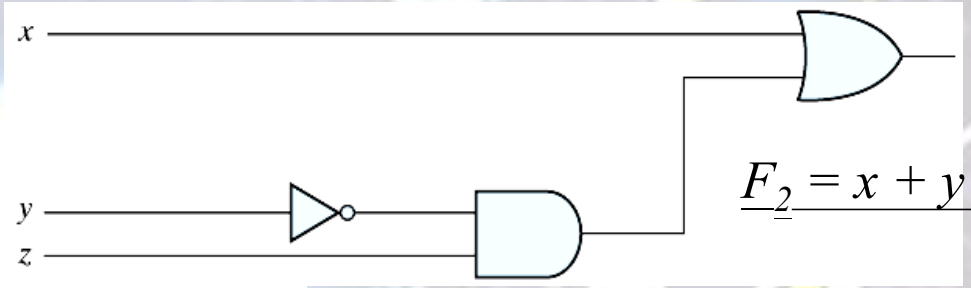
x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

- Two Boolean expressions may specify the same function
  - $F_3 = F_4$

Boolean Functions

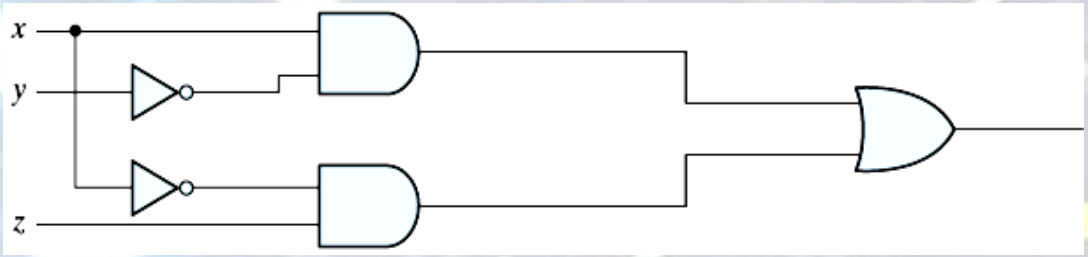
- Implementation with logic gates
  - F_4 is more economical





$F_2 = x + y'z$

$F_3 = x'y'z + x'yz + xy'$



$F_4 = xy' + x'z$

Week -7

Page(106-120)

Algebraic Manipulation

- To minimize Boolean expressions
  - Literal: a primed or unprimed variable (an input to a gate)
  - Term: an implementation with a gate
  - The minimization of the number of literals and the number of terms → a circuit with less equipment
  - It is a hard problem (no specific rules to follow)
- Example 2.1
  1. $x(x'+y) = xx' + xy = 0+xy = xy$
  2. $x+x'y = (x+x')(x+y) = 1\ (x+y) = x+y$
  3. $(x+y)(x+y') = x+xy+xy'+yy' = x(1+y+y') = x$
  4. $xy + x'z + yz = xy + x'z + yz(x+x') = xy + x'z + yzx + yzx' = xy(1+z) + x'z(1+y) = xy + x'z$
  5. $(x+y)(x'+z)(y+z) = (x+y)(x'+z)$, by duality from function 4. (*consensus theorem* with duality)

Complement of a Function

- An interchange of 0's for 1's and 1's for 0's in the value of F
  - By DeMorgan's theorem
  - $(A+B+C)' = (A+X)'$ let $B+C = X$
    $= A'X'$ by theorem 5(a) (DeMorgan's)
    $= A'(B+C)'$ substitute $B+C = X$
    $= A'(B'C')$ by theorem 5(a) (DeMorgan's)
    $= A'B'C'$ by theorem 4(b) (associative)
- *Generalizations*: a function is obtained by interchanging AND and OR operators and complementing each literal.
  - $(A+B+C+D+ \ldots +F)' = A'B'C'D' \ldots F'$
  - $(ABCD \ldots F)' = A'+ B'+C'+D' \ldots +F'$

Examples

- Example 2.2
  - $F_1' = (x'yz' + x'y'z)' = (x'yz')'(x'y'z)' = (x+y'+z)(x+y+z')$
  - $F_2' = [x(y'z'+yz)]' = x' + (y'z'+yz)' = x' + (y'z')'(yz)'$
    $= x' + (y+z)(y'+z')$
    $= x' + yz' + y'z$
- Example 2.3: a simpler procedure
  - Take the dual of the function and complement each literal

1. $F_1 = x'yz' + x'y'z$.

   The dual of F_1 is $(x'+y+z')(x'+y'+z)$.

   Complement each literal: $(x+y'+z)(x+y+z') = F_1'$

2. $F_2 = x(y'z' + yz)$.

   The dual of F_2 is $x+(y'+z')(y+z)$.

   Complement each literal: $x'+(y+z)(y'+z') = F_2'$

2.6 Canonical and Standard Forms

Minterms and Maxterms

- A minterm (standard product): an AND term consists of all literals in their normal form or in their complement form.
  - For example, two binary variables x and y,
    - $xy, xy', x'y, x'y'$
  - It is also called a standard product.
  - n variables con be combined to form 2^n minterms.
- A maxterm (standard sums): an OR term
  - It is also call a standard sum.
  - 2^n maxterms.

Minterms and Maxterms

- Each *maxterm* is the complement of its corresponding *minterm*, and vice versa.

Table 2.3
Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Minterms and Maxterms

- An Boolean function can be expressed by
  - A truth table
  - Sum of minterms
  - $f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$ (Minterms)
  - $f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$ (Minterms)

Table 2.4
Functions of Three Variables

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	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

Minterms and Maxterms

- The complement of a Boolean function
  - The minterms that produce a 0
  - $f_1' = m_0 + m_2 + m_3 + m_5 + m_6 = x'y'z' + x'yz' + x'yz + xy'z + xyz'$
  - $f_1 = (f_1')'$
    $= (x+y+z)(x+y'+z)\,(x+y'+z')\,(x'+y+z')(x'+y'+z) = M_0\,M_2\,M_3\,M_5\,M_6$
  - $f_2 = (x+y+z)(x+y+z')(x+y'+z)(x'+y+z) = M_0 M_1 M_2 M_4$
- Any Boolean function can be expressed as
  - A sum of minterms ("sum" meaning the ORing of terms).
  - A product of maxterms ("product" meaning the ANDing of terms).
  - Both boolean functions are said to be in Canonical form.

Sum of Minterms

- Sum of minterms: there are 2^n minterms and 2^{2n} combinations of function with n Boolean variables.
- Example 2.4: express $F = A+BC'$ as a sum of minterms.
  - $F = A+B'C = A\,(B+B') + B'C = AB + AB' + B'C = AB(C+C') + AB'(C+C') + (A+A')B'C = ABC+ABC'+AB'C+AB'C'+A'B'C$
  - $F = A'B'C + AB'C' + AB'C + ABC' + ABC = m_1 + m_4 + m_5 + m_6 + m_7$
  - $F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$
  - or, built the truth table firs

Table 2.5
Truth Table for F = A + B′C

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Product of Maxterms

- Product of maxterms: using distributive law to expand.
  - $x + yz = (x + y)(x + z) = (x+y+zz')(x+z+yy') = (x+y+z)(x+y+z')(x+y'+z)$
- Example 2.5: express $F = xy + x'z$ as a product of maxterms.
  - $F = xy + x'z = (xy + x')(xy + z) = (x+x')(y+x')(x+z)(y+z) = (x'+y)(x+z)(y+z)$
  - $x'+y = x' + y + zz' = (x'+y+z)(x'+y+z')$
  - $F = (x+y+z)(x+y'+z)(x'+y+z)(x'+y+z') = M_0M_2M_4M_5$
  - $F(x, y, z) = \Pi(0, 2, 4, 5)$

Conversion between Canonical Forms

- The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.
  - $F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$
  - Thus, $F'(A, B, C) = \Sigma(0, 2, 3)$
  - By DeMorgan's theorem

    $F(A, B, C) = \Pi(0, 2, 3)$

    $F'(A, B, C) = \Pi(1, 4, 5, 6, 7)$
  - $m_j' = M_j$
  - Sum of minterms = product of maxterms
  - Interchange the symbols Σ and Π and list those numbers missing from the original form
    - Σ of 1's
    - Π of 0's

- Example
  - $F = xy + x'z$
  - $F(x, y, z) = \Sigma(1, 3, 6, 7)$
  - $F(x, y, z) = \Pi(0, 2, 4, 6)$

Table 2.6
Truth Table for F = xy + x′z

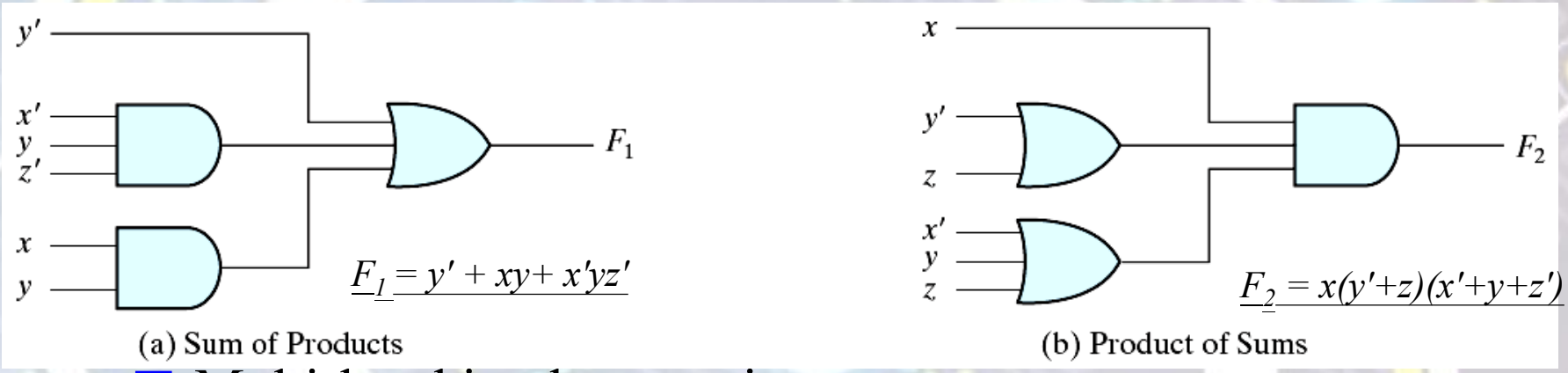
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Standard Forms

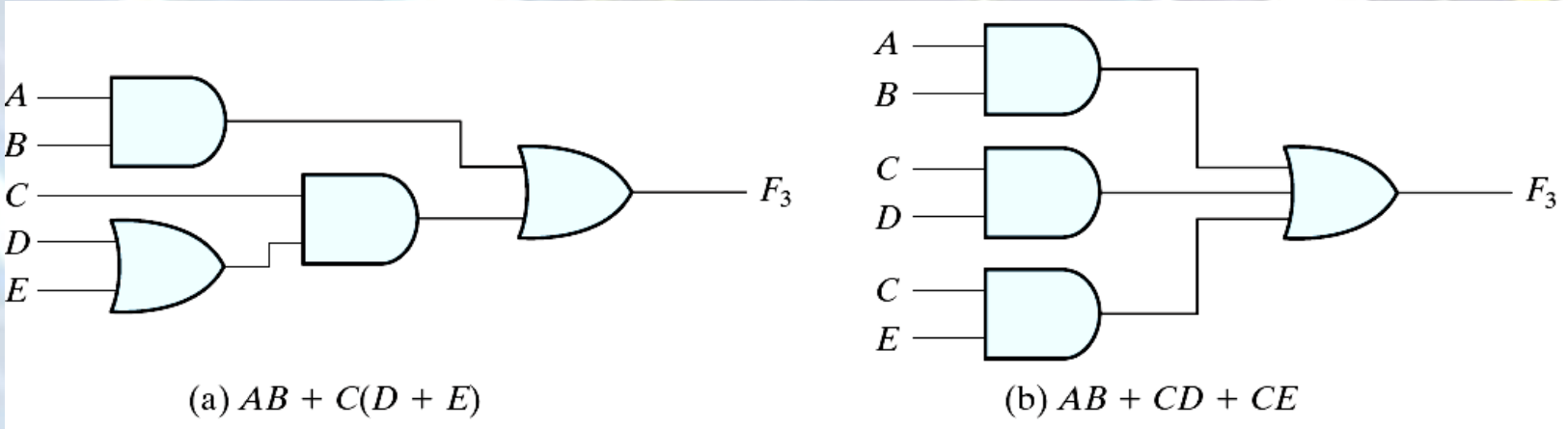
- Canonical forms are very seldom the ones with the least number of literals.
- Standard forms: the terms that form the function may obtain one, two, or any number of literals.
  - Sum of products: $F_1 = y' + xy + x'yz'$
  - Product of sums: $F_2 = x(y'+z)(x'+y+z')$
  - $F_3 = A'B'CD + ABC'D'$

Implementation

- Two-level implementation



- Multi-level implementation



2.7 Other Logic Operations (

- 2^n rows in the truth table of n binary variables.
- 2^{2^n} functions for n binary variables.
- 16 functions of two binary variables.

Table 2.7
Truth Tables for the 16 Functions of Two Binary Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- All the new symbols except for the exclusive-OR symbol are not in common use by digital designers.

Boolean Expressions

Table 2.8
Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Week -8

Page(122-136)

2.8 Digital Logic Gates

- Boolean expression: AND, OR and NOT operations
- Constructing gates of other logic operations
  - The feasibility and economy;
  - The possibility of extending gate's inputs;
  - The basic properties of the binary operations (commutative and associative);
  - The ability of the gate to implement Boolean functions.

Standard Gates

- Consider the 16 functions in Table 2.8 (slide 33)
  - Two are equal to a constant (F_0 and F_{15}).
  - Four are repeated twice (F_4, F_5, F_{10} and F_{11}).
  - Inhibition (F_2) and implication (F_{13}) are not commutative or associative.
  - The other eight: complement (F_{12}), transfer (F_3), AND (F_1), OR (F_7), NAND (F_{14}), NOR (F_8), XOR (F_6), and equivalence (XNOR) (F_9) are used as standard gates.
  - Complement: inverter.
  - Transfer: buffer (increasing drive strength).
  - Equivalence: XNOR.

Summary of Logic Gates

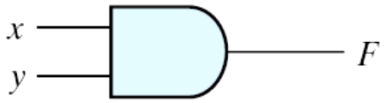
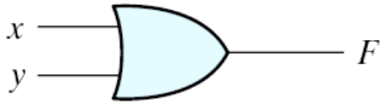
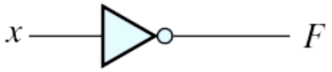
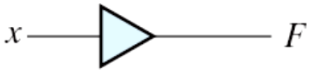
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = xy$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1

Figure 2.5 Digital logic gates

Summary of Logic Gates

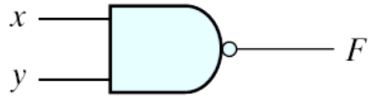
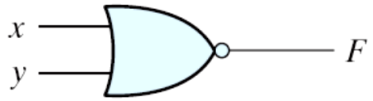
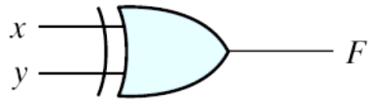
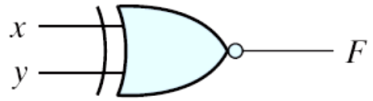
NAND		$F = (xy)'$	x	y	F
			0	0	1
			0	1	1
			1	0	1
			1	1	0
NOR		$F = (x + y)'$	x	y	F
			0	0	1
			0	1	0
			1	0	0
			1	1	0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x	y	F
			0	0	0
			0	1	1
			1	0	1
			1	1	0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	x	y	F
			0	0	1
			0	1	0
			1	0	0
			1	1	1

Figure 2.5 Digital logic gates

Multiple Inputs

- Extension to multiple inputs
  - A gate can be extended to multiple inputs.
    - » If its binary operation is commutative and associative.
  - AND and OR are commutative and associative.
    - » OR
      - $x+y = y+x$
      - $(x+y)+z = x+(y+z) = x+y+z$
    - » AND
      - $xy = yx$
      - $(x\ y)z = x(y\ z) = x\ y\ z$

Multiple Inputs

- NAND and NOR are commutative but not associative → they are not extendable.

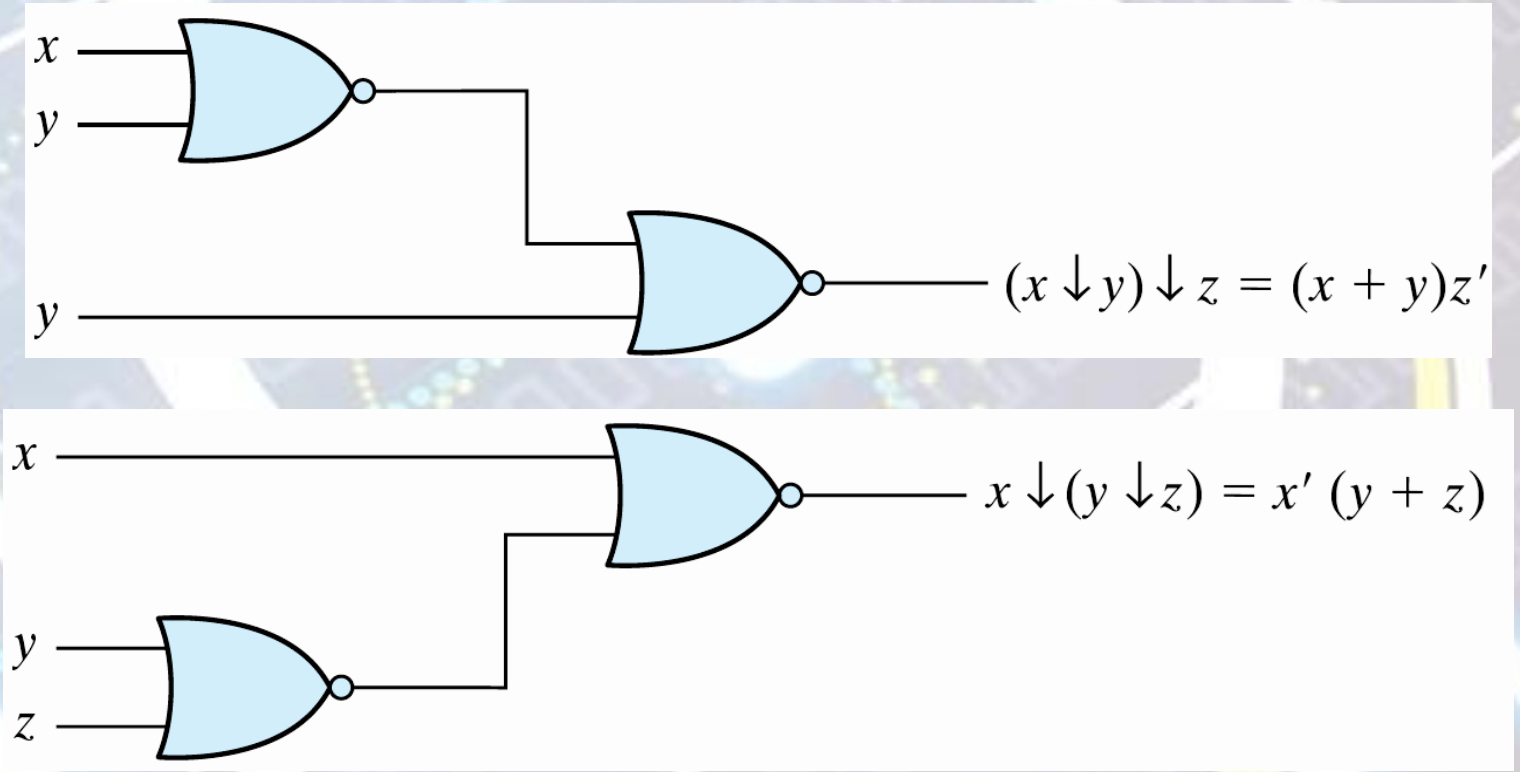


Figure 2.6 Demonstrating the nonassociativity of the NOR operator; $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$

Multiple Inputs

- Multiple NOR = a complement of OR gate, Multiple NAND = a complement of AND.
- The cascaded NAND operations = sum of products.
- The cascaded NOR operations = product of sums.

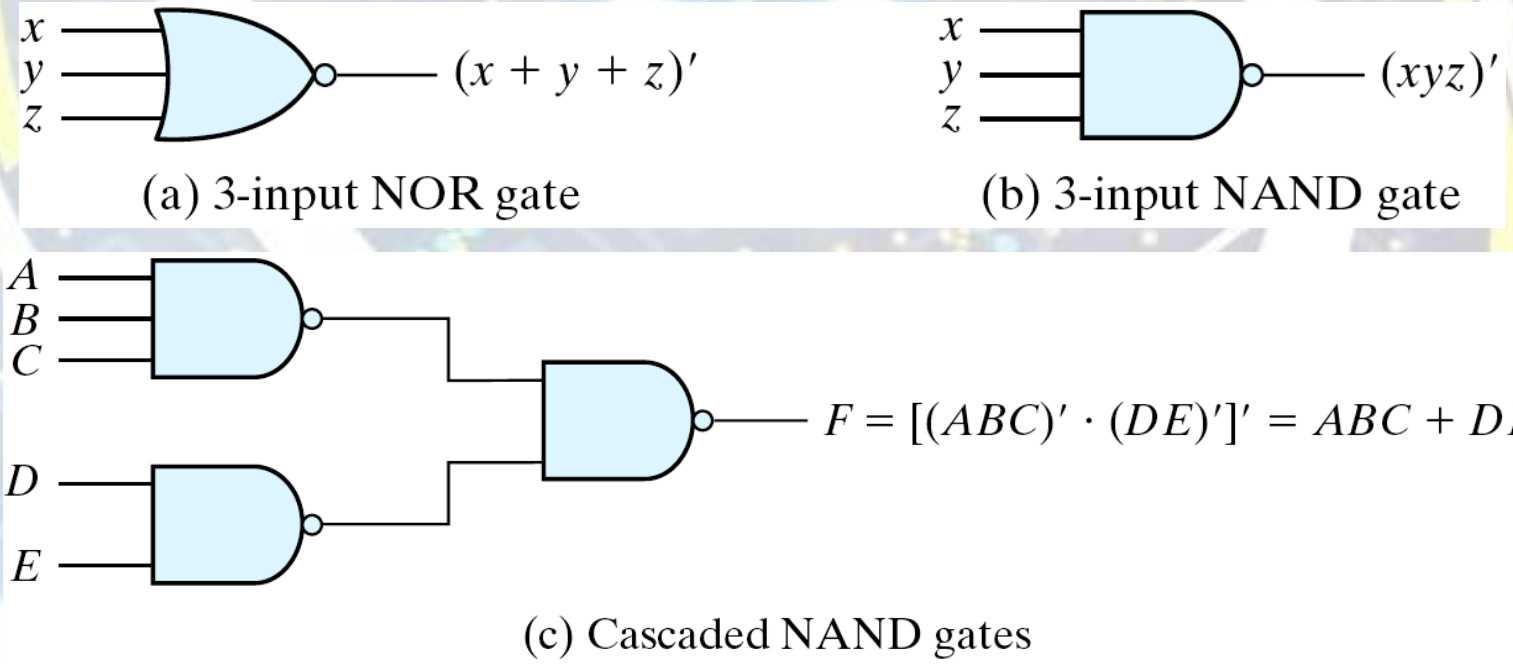
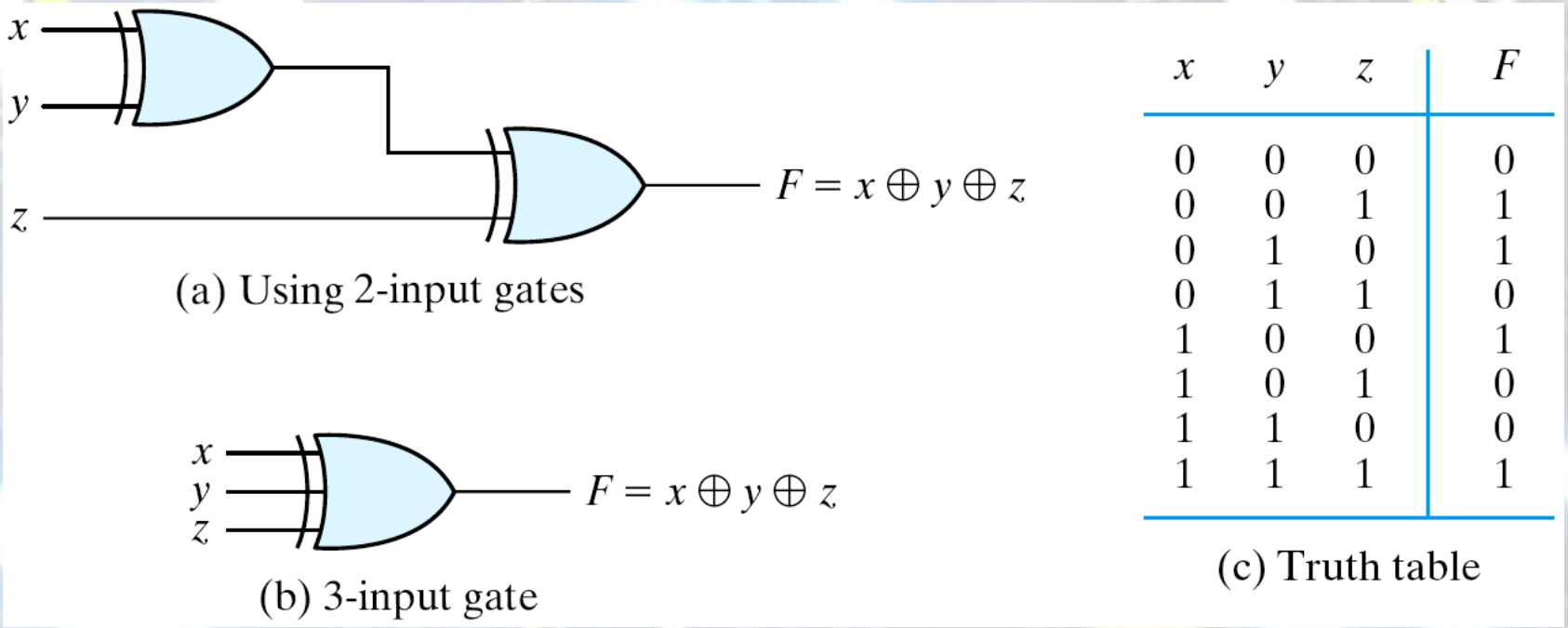


Figure 2.7 Multiple-input and cascated NOR and NAND gates

Multiple Inputs

- The XOR and XNOR gates are commutative and associative.
- Multiple-input XOR gates are uncommon?
- XOR is an odd function: it is equal to 1 if the inputs variables have an odd number of 1's.



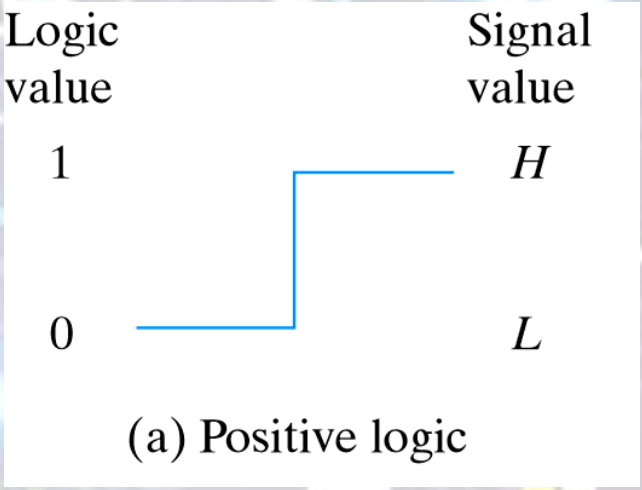
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(c) Truth table

Figure 2.8 3-input XOR gate

Positive and Negative Logic

- Positive and Negative Logic
  - Two signal values <=> two logic values
  - Positive logic: H=1; L=0
  - Negative logic: H=0; L=1
- Consider a TTL gate
  - A positive logic AND gate
  - A negative logic OR gate
  - The positive logic is used in this book



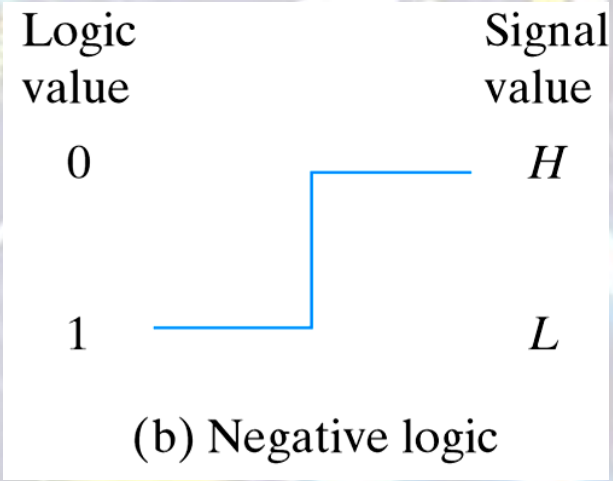
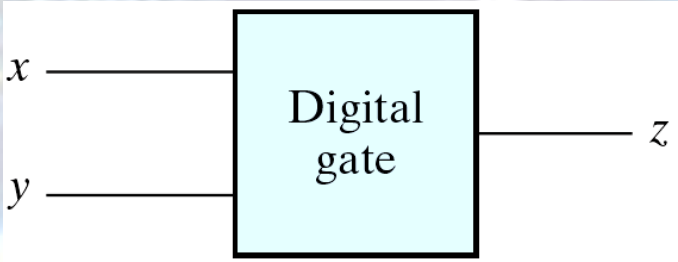


Figure 2.9 Signal assignment and logic polarity

Positive and Negative Logic

x	y	z
L	L	L
L	H	L
H	L	L
H	H	H

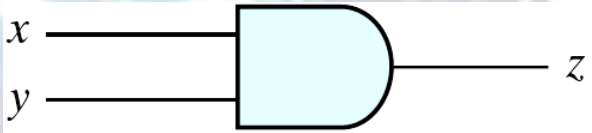
(a) Truth table with H and L



(b) Gate block diagram

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

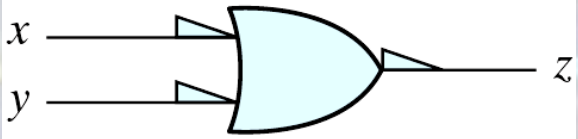
(c) Truth table for positive logic



(d) Positive logic AND gate

x	y	z
1	1	1
1	0	1
0	1	1
0	0	0

(e) Truth table for negative logic



(f) Negative logic OR gate

Figure 2.10 Demonstration of positive and negative logic

2.9 Integrated Circuits

Level of Integration

- An IC (a chip)
- Examples:
  - Small-scale Integration (SSI): < 10 gates
  - Medium-scale Integration (MSI): 10 ~ 100 gates
  - Large-scale Integration (LSI): 100 ~ xk gates
  - Very Large-scale Integration (VLSI): > xk gates
- VLSI
  - Small size (compact size)
  - Low cost
  - Low power consumption
  - High reliability
  - High speed

Digital Logic Families

- Digital logic families: circuit technology
  - TTL: transistor-transistor logic (dying?)
  - ECL: emitter-coupled logic (high speed, high power consumption)
  - MOS: metal-oxide semiconductor (NMOS, high density)
  - CMOS: complementary MOS (low power)
  - BiCMOS: high speed, high density

Digital Logic Families

- The characteristics of digital logic families
  - Fan-out: the number of standard loads that the output of a typical gate can drive.
  - Power dissipation.
  - Propagation delay: the average transition delay time for the signal to propagate from input to output.
  - Noise margin: the minimum of external noise voltage that caused an undesirable change in the circuit output.

CAD

- CAD – Computer-Aided Design
  - Millions of transistors
  - Computer-based representation and aid
  - Automatic the design process
  - Design entry
    - » Schematic capture
    - » HDL – Hardware Description Language
      - Verilog, VHDL
  - Simulation
  - Physical realization
    - » ASIC, FPGA, PLD

Chip Design

- Why is it better to have more gates on a single chip?
  - Easier to build systems
  - Lower power consumption
  - Higher clock frequencies
- What are the drawbacks of large circuits?
  - Complex to design
  - Chips have design constraints
  - Hard to test
- Need tools to help develop integrated circuits
  - Computer Aided Design (CAD) tools
  - Automate tedious steps of design process
  - Hardware description language (HDL) describe circuits
  - VHDL (see the lab) is one such system

Week -9

Page(138-159)

3-1 Introduction

- Gate-level minimization refers to the design task of finding an optimal gate-level implementation of Boolean functions describing a digital circuit.

3-2 The Map Method

- The complexity of the digital logic gates
  - The complexity of the algebraic expression
- Logic minimization
  - Algebraic approaches: lack specific rules
  - The Karnaugh map
    - » A simple straight forward procedure
    - » A pictorial form of a truth table
    - » Applicable if the # of variables < 7
- A diagram made up of squares
  - Each square represents one minterm

Review of Boolean Function

- Boolean function
  - Sum of minterms
  - Sum of products (or product of sum) in the simplest form
  - A minimum number of terms
  - A minimum number of literals
  - The simplified expression may not be unique

Two-Variable Map

- A two-variable map
  - Four minterms
  - x' = row 0; x = row 1
  - y' = column 0; y = column 1
  - A truth table in square diagram
  - Fig. 3.2(a): $xy = m_3$
  - Fig. 3.2(b): $x+y = x'y+xy'+xy = m_1+m_2+m_3$

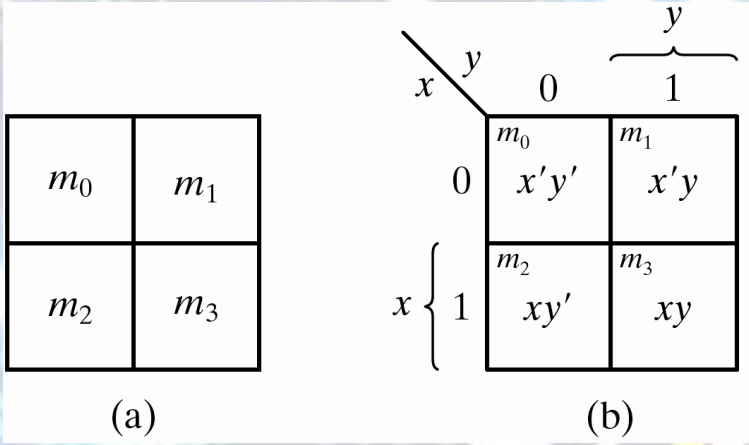


Figure 3.1 Two-variable Map

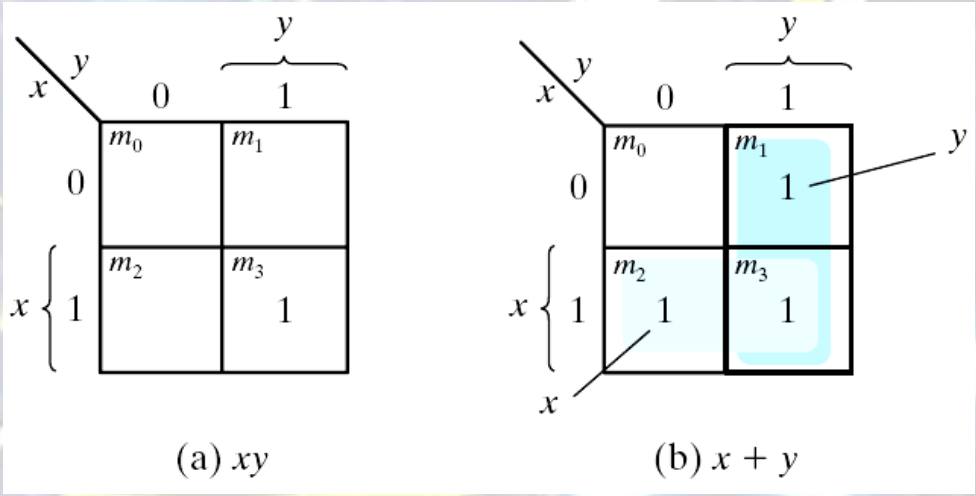


Figure 3.2 Representation of functions in the map

A Three-variable Map

- A three-variable map
  - Eight minterms
  - The Gray code sequence
  - Any two adjacent squares in the map differ by only on variable
    - » Primed in one square and unprimed in the other
    - » e.g., m_5 and m_7 can be simplified
    - » $m_5 + m_7 = xy'z + xyz = xz(y'+y) = xz$

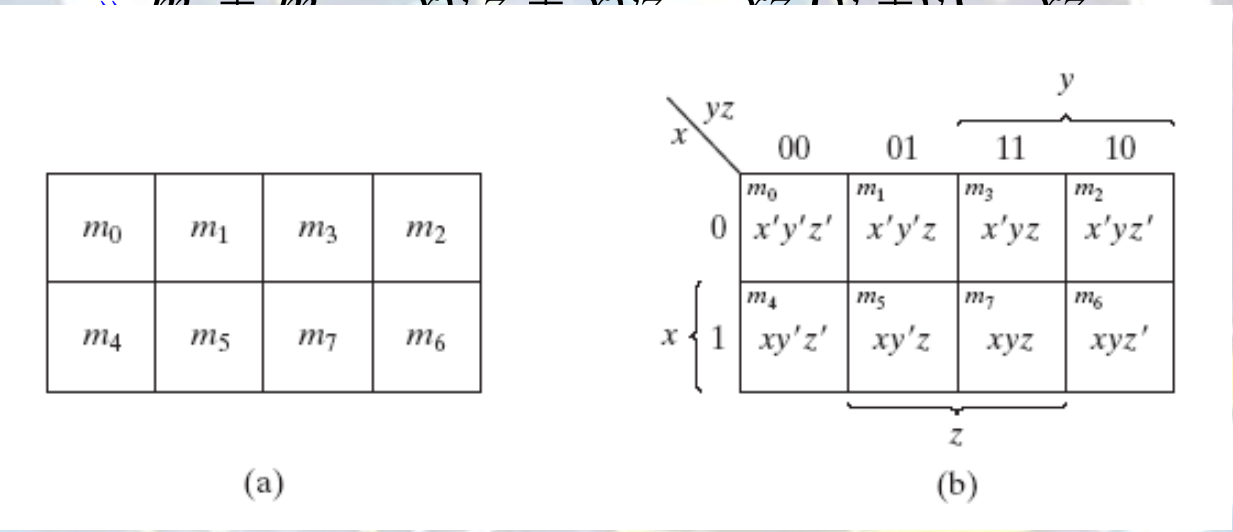


Figure 3.3 Three-variable Map

A Three-variable Map

- m_0 and m_2 (m_4 and m_6) are adjacent
- $m_0 + m_2 = x'y'z' + x'yz' = x'z'(y'+y) = x'z'$
- $m_4 + m_6 = xy'z' + xyz' = xz'(y'+y) = xz'$

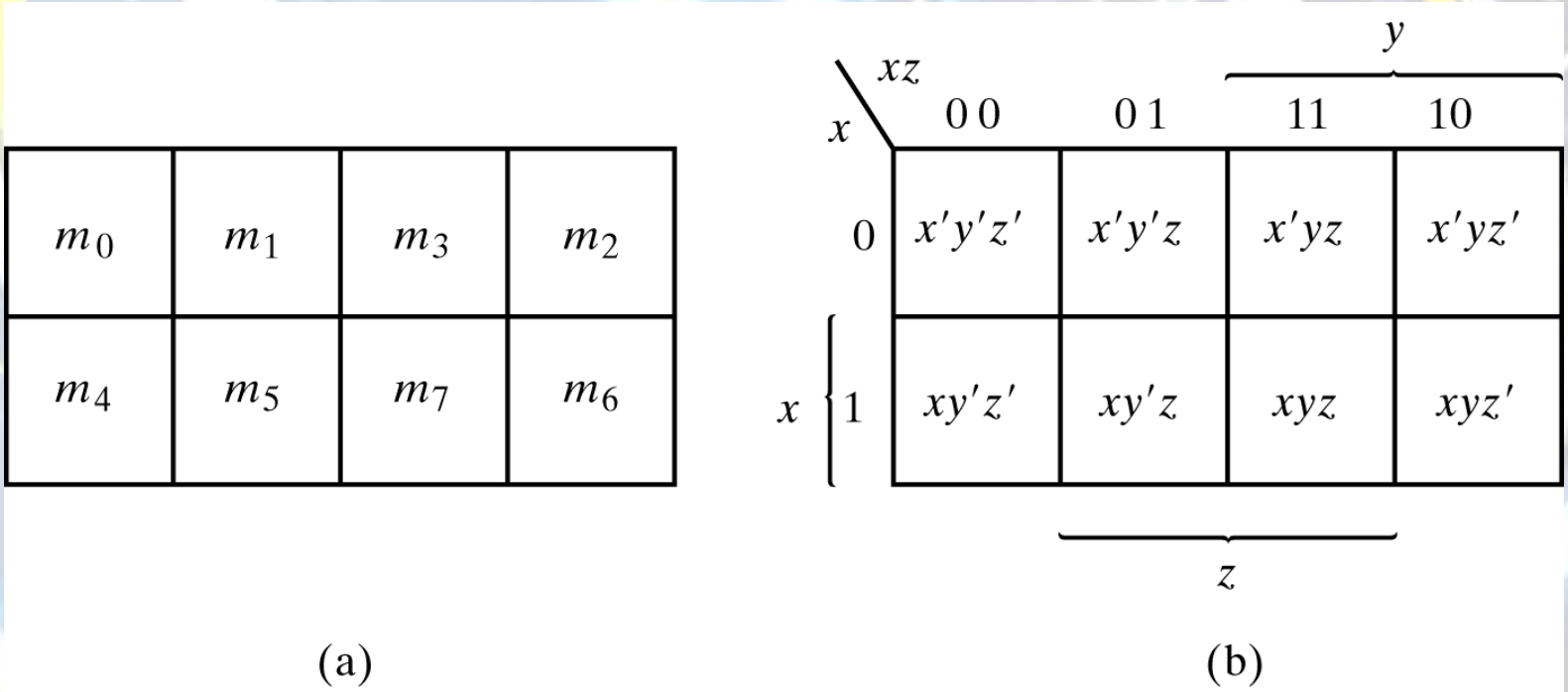


Fig. 3-3 Three-variable Map

Example 3.1

- Example 3.1: simplify the Boolean function $F(x, y, z) = \Sigma(2, 3, 4, 5)$
  - $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$

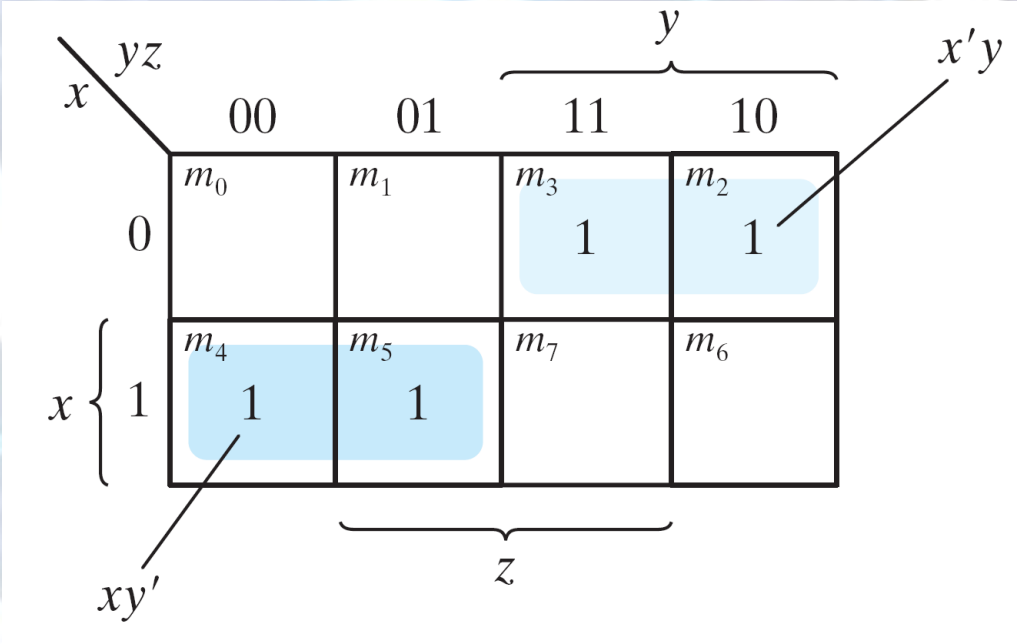


Figure 3.4 Map for Example 3.1, F(x, y, z) = Σ(2, 3, 4, 5) = x'y + xy'

Example 3.2

- Example 3.2: simplify $F(x, y, z) = \Sigma(3, 4, 6, 7)$
  - $F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$

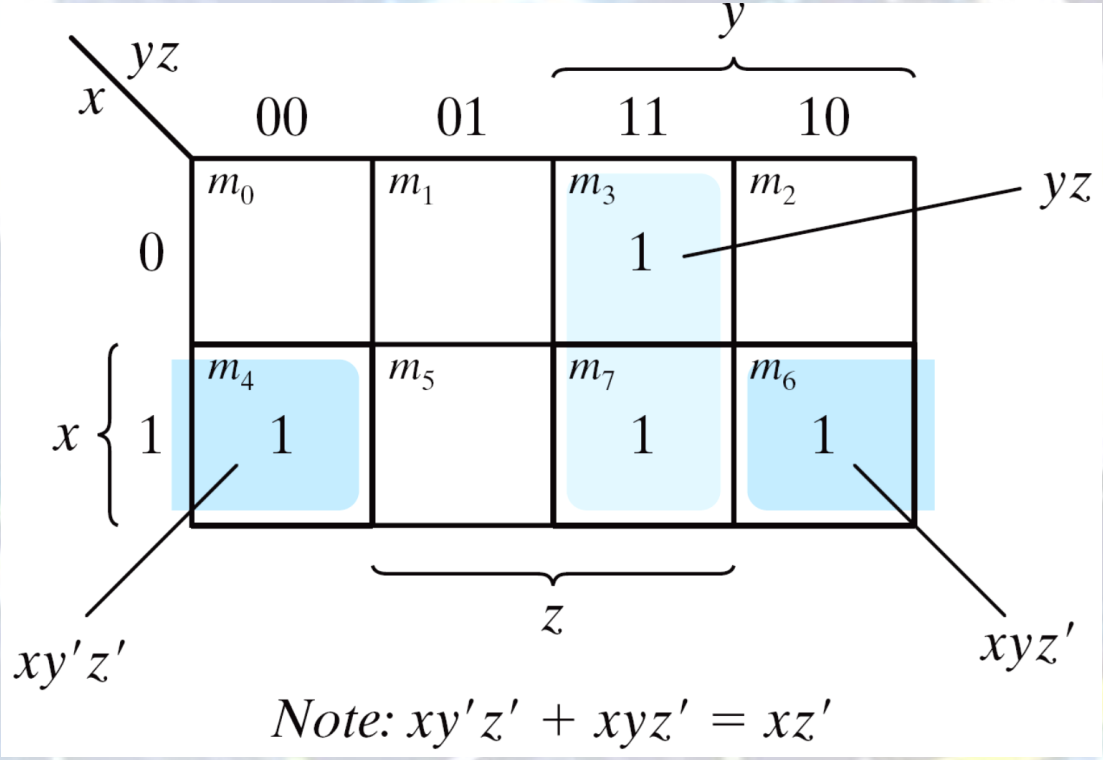


Figure 3.5 Map for Example 3-2; F(x, y, z) = Σ(3, 4, 6, 7) = yz + xz'

Four adjacent Squares

- Consider four adjacent squares
  - 2, 4, and 8 squares
  - $m_0+m_2+m_4+m_6 = x'y'z'+x'yz'+xy'z'+xyz' = x'z'(y'+y) +xz'(y'+y) = x'z' + xz' = z'$
  - $m_1+m_3+m_5+m_7 = x'y'z+x'yz+xy'z+xyz = x'z(y'+y) + xz(y'+y) = x'z + xz = z$

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

x \ yz	00	01	11	10
0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
1	$xy'z'$	$xy'z$	xyz	xyz'

y (11, 10); x (0, 1); z (01, 11)

(b)

Figure 3.3 Three-variable Map

Example 3.3

- Example 3.3: simplify $F(x, y, z) = \Sigma(0, 2, 4, 5, 6)$
  - $F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$

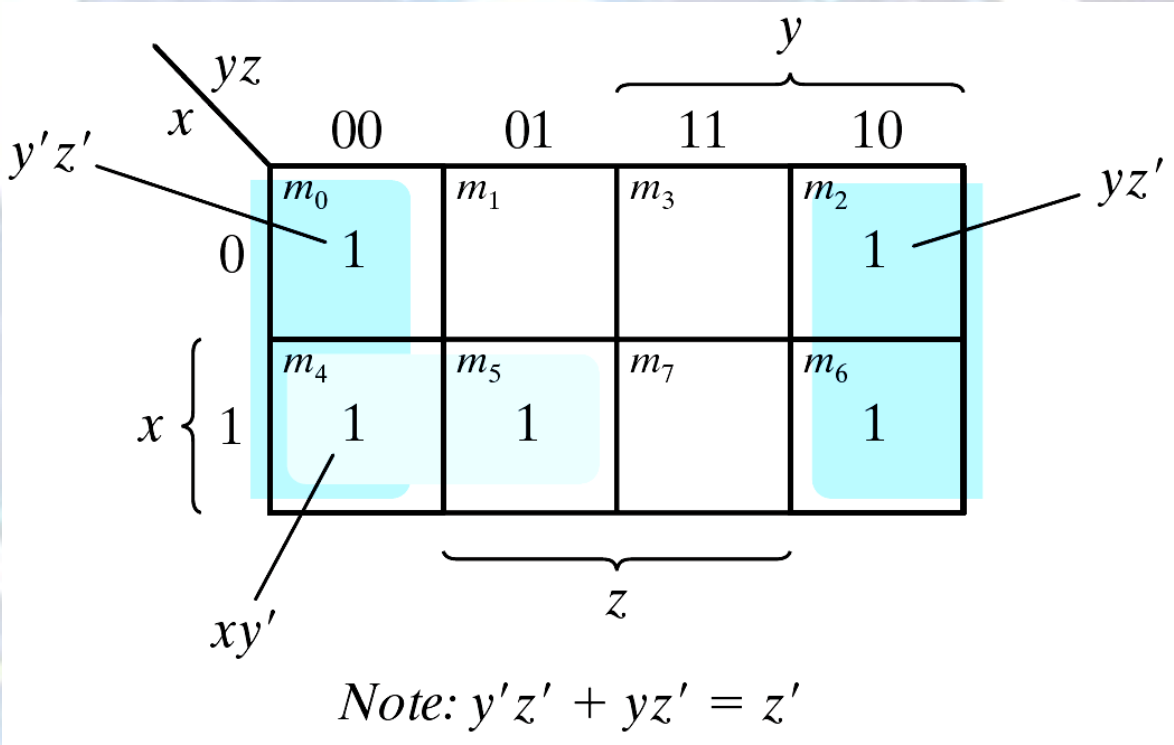


Figure 3.6 Map for Example 3-3, F(x, y, z) = Σ(0, 2, 4, 5, 6) = z' +xy'

Example 3.4

- Example 3.4: let $F = A'C + A'B + AB'C + BC$
  a) Express it in sum of minterms.
  b) Find the minimal sum of products expression.

  Ans:

  $F(A, B, C) = \Sigma(1, 2, 3, 5, 7) = C + A'B$

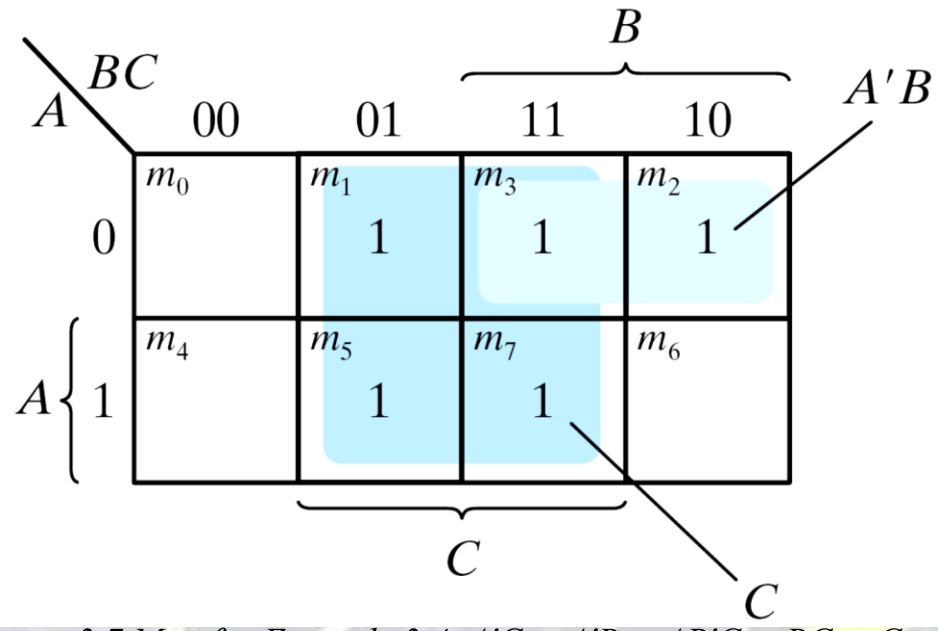


Figure 3.7 Map for Example 3.4, A'C + A'B + AB'C + BC = C + A'B

3.3 Four-Variable Map

- The map
  - 16 minterms
  - Combinations of 2, 4, 8, and 16 adjacent squares

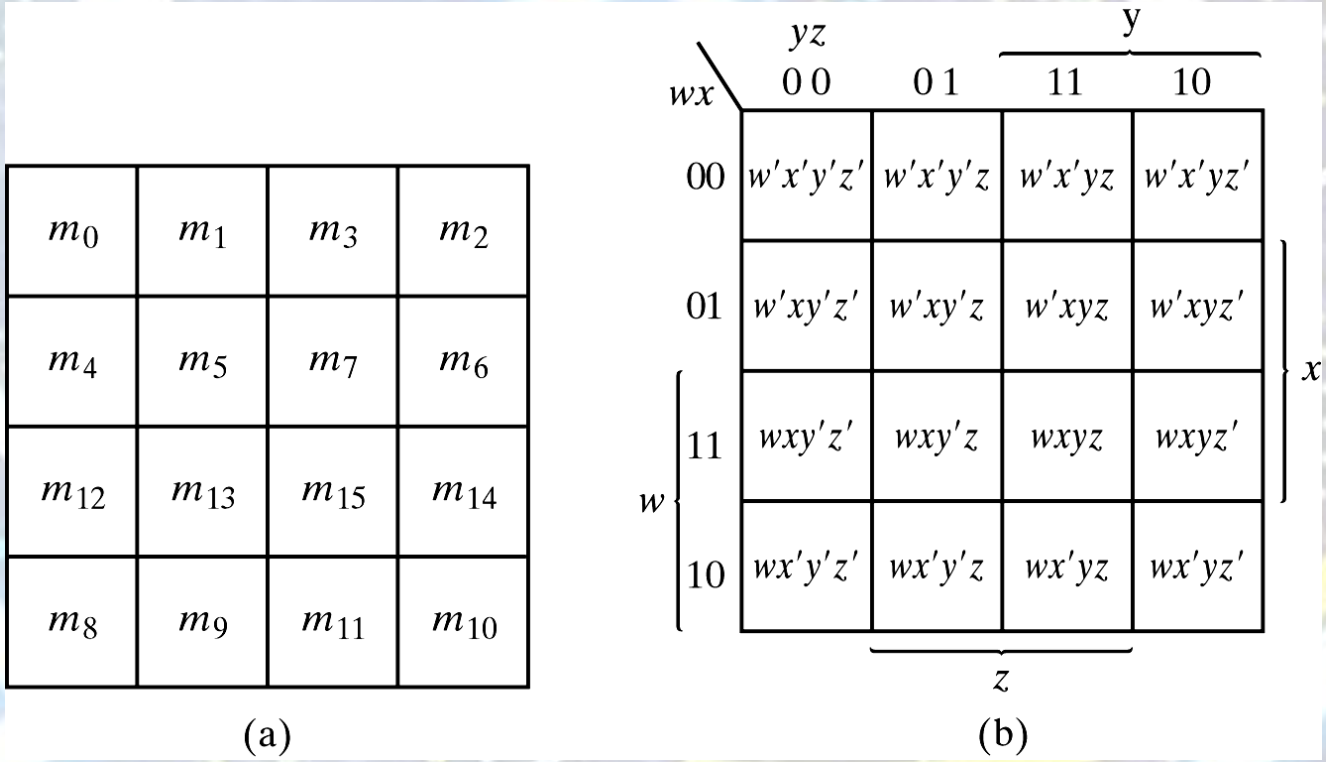
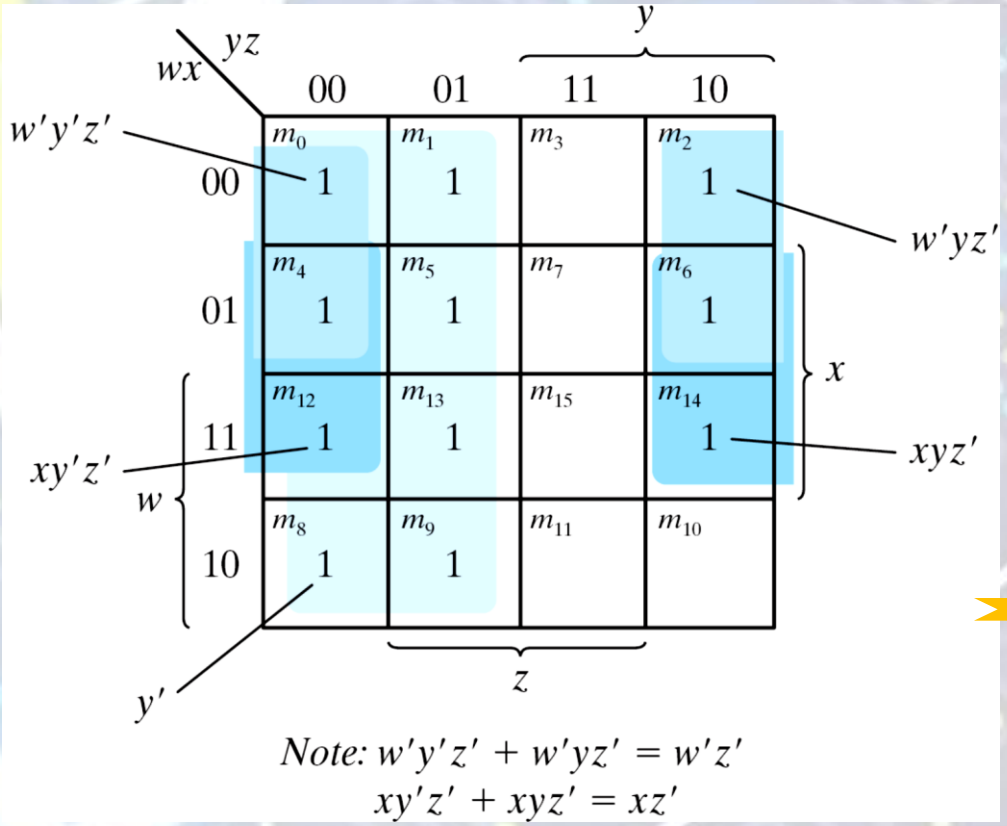


Figure 3.8 Four-variable Map

Example 3.5

- Example 3.5: simplify $F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

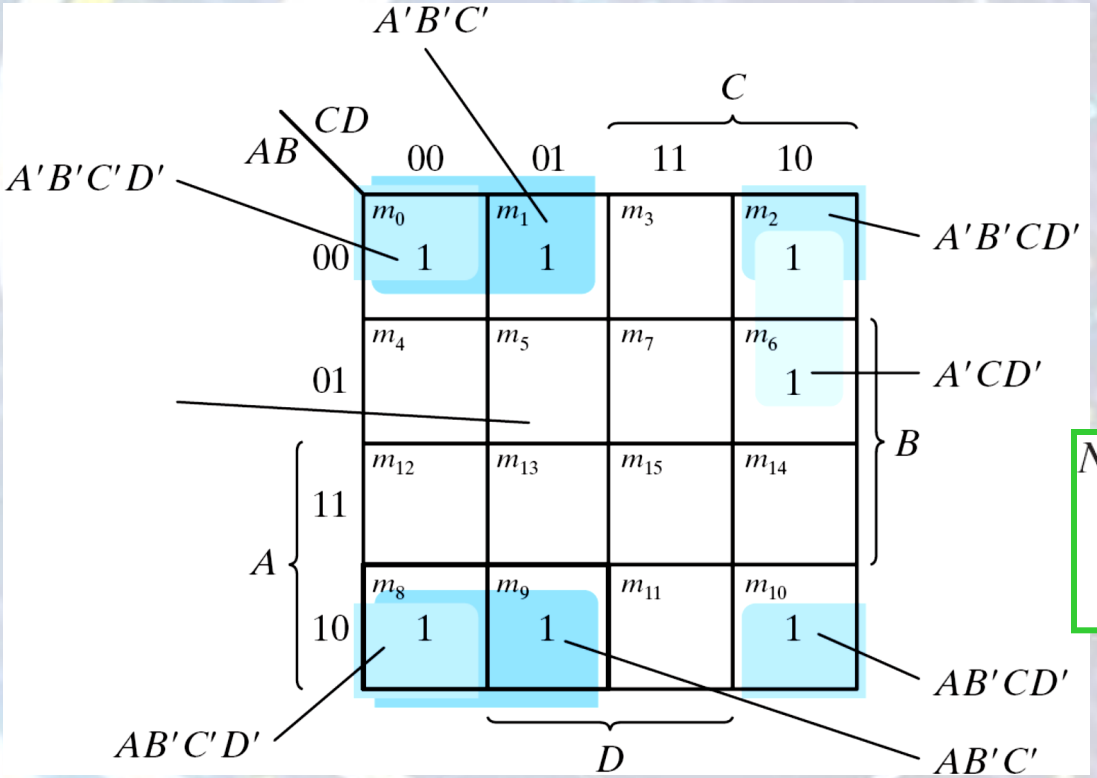


$F = y' + w'z' + xz'$

Figure 3.9 Map for Example 3-5; $F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = y' + w'z' + xz'$

Example 3.6

- Example 3-6: simplify $F = A'B'C' + B'CD' + A'B'C'D' + AB'C'$



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$
$A'B'C' + AB'C' = B'C'$

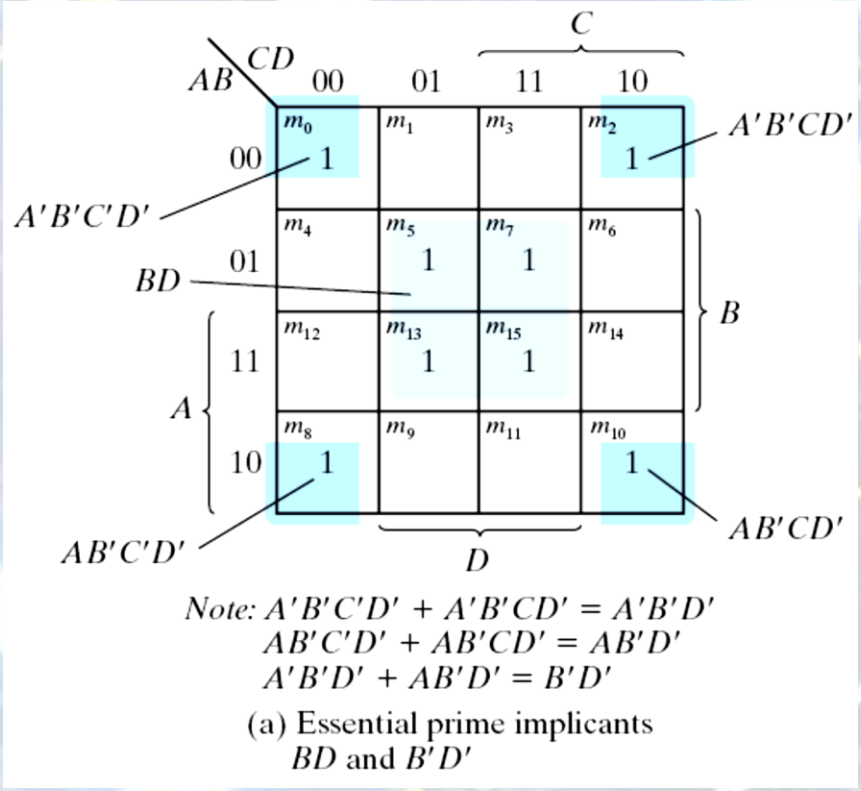
Figure 3.9 Map for Example 3-6; $A'B'C' + B'CD' + A'B'C'D' + AB'C' = B'D' + B'C' + A'CD'$

Prime Implicants

- Prime Implicants
  - All the minterms are covered.
  - Minimize the number of terms.
  - A prime implicant: a product term obtained by combining the maximum possible number of adjacent squares (combining all possible maximum numbers of squares).
  - Essential P.I.: a minterm is covered by only one prime implicant.
  - The essential P.I. must be included.

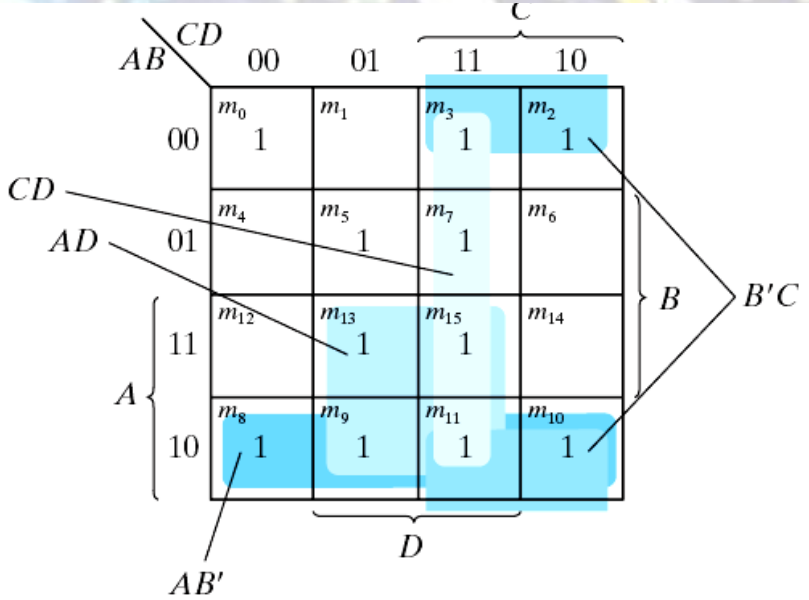
Prime Implicants

- Consider $F(A, B, C, D) = \Sigma(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$
  - The simplified expression may not be unique
  - $F = BD+B'D'+CD+AD = BD+B'D'+CD+AB'$
    $= BD+B'D'+B'C+AD = BD+B'D'+B'C+AB'$



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$

(a) Essential prime implicants BD and $B'D'$



(b) Prime implicants CD, $B'C$, AD, and AB'

Figure 3.11 Simplification Using Prime Implicants

3.4 Five-Variable Map

- Map for more than four variables becomes complicated
  - Five-variable map: two four-variable map (one on the top of the other).

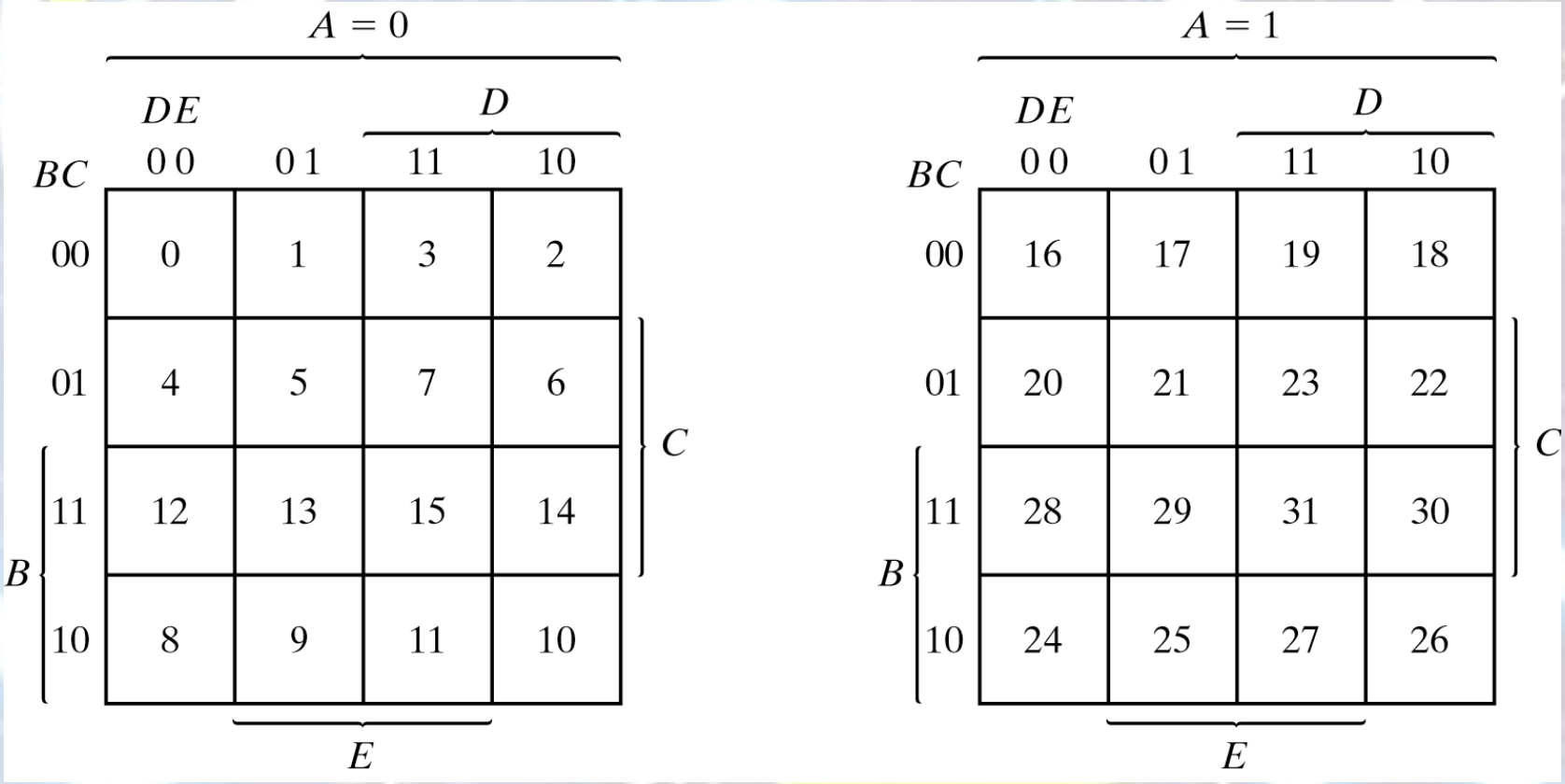


Figure 3.12 Five-variable Map

Example 3.7

- Example 3.7: simplify $F = \Sigma(0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31)$

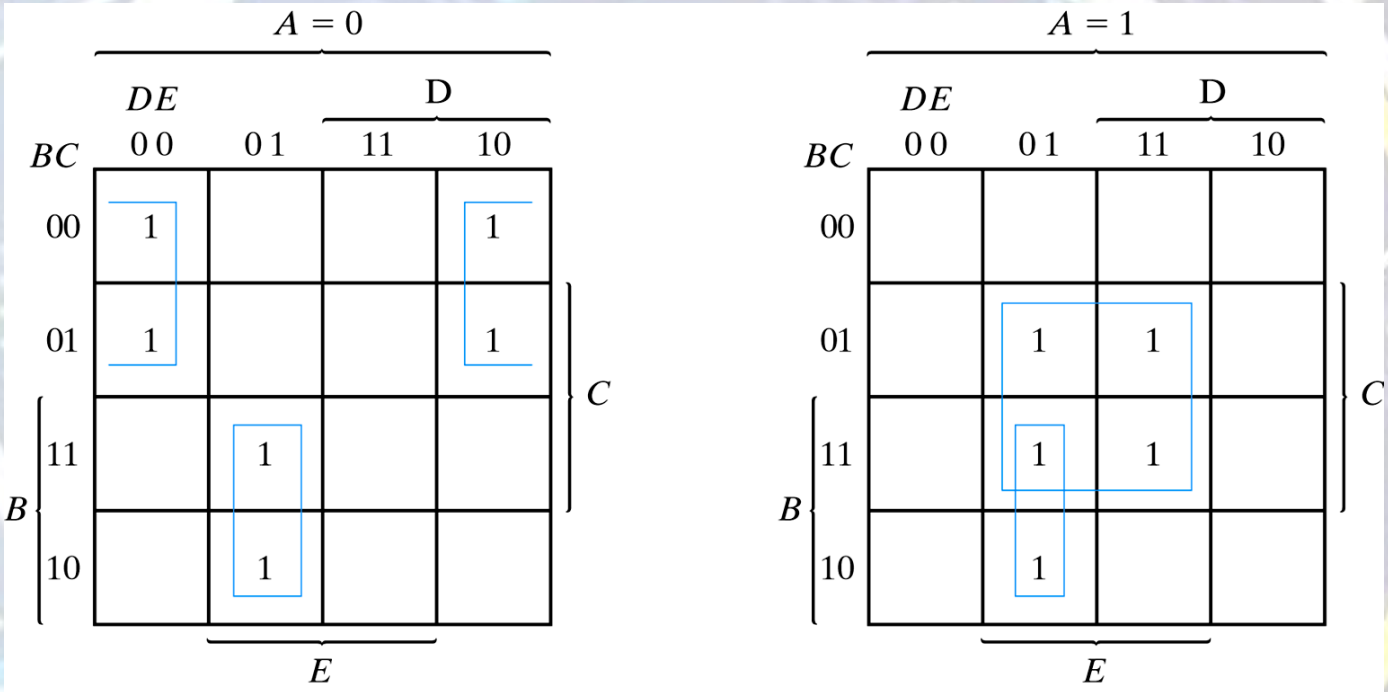


Fig. 3-13 Map for Example 3-7; $F = A'B'E' + BD'E + ACE$

$F = A'B'E' + BD'E + ACE$

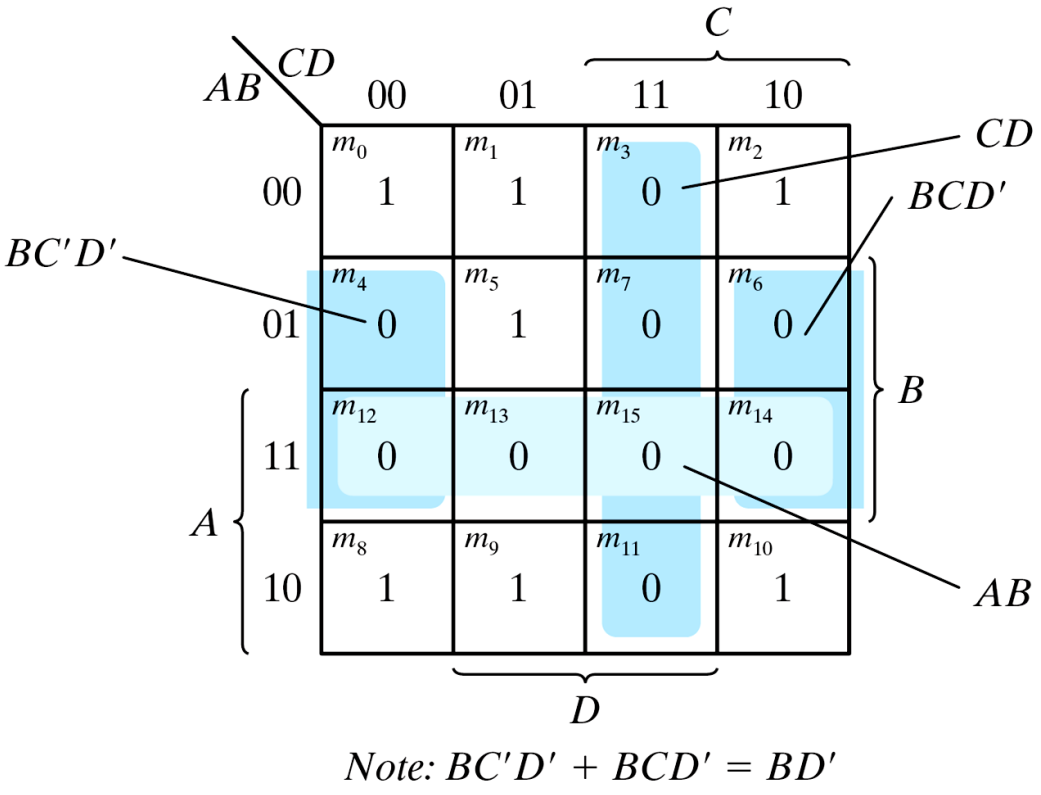
3-5 Product of Sums Simplification

- Approach #1
  - Simplified F' in the form of sum of products
  - Apply DeMorgan's theorem $F = (F')'$
  - F': sum of products → F: product of sums
- Approach #2: duality
  - Combinations of maxterms (it was minterms)
  - $M_0M_1 = (A+B+C+D)(A+B+C+D') = (A+B+C)+(DD') = A+B+C$

AB \ CD	00	01	11	10
00	M_0	M_1	M_3	M_2
01	M_4	M_5	M_7	M_6
11	M_{12}	M_{13}	M_{15}	M_{14}
10	M_8	M_9	M_{11}	M_{10}

Example 3.8

- Example 3.8: simplify $F = \Sigma(0, 1, 2, 5, 8, 9, 10)$ into (a) sum-of-products form, and (b) product-of-sums form:



a) $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10) = B'D' + B'C' + A'C'D$

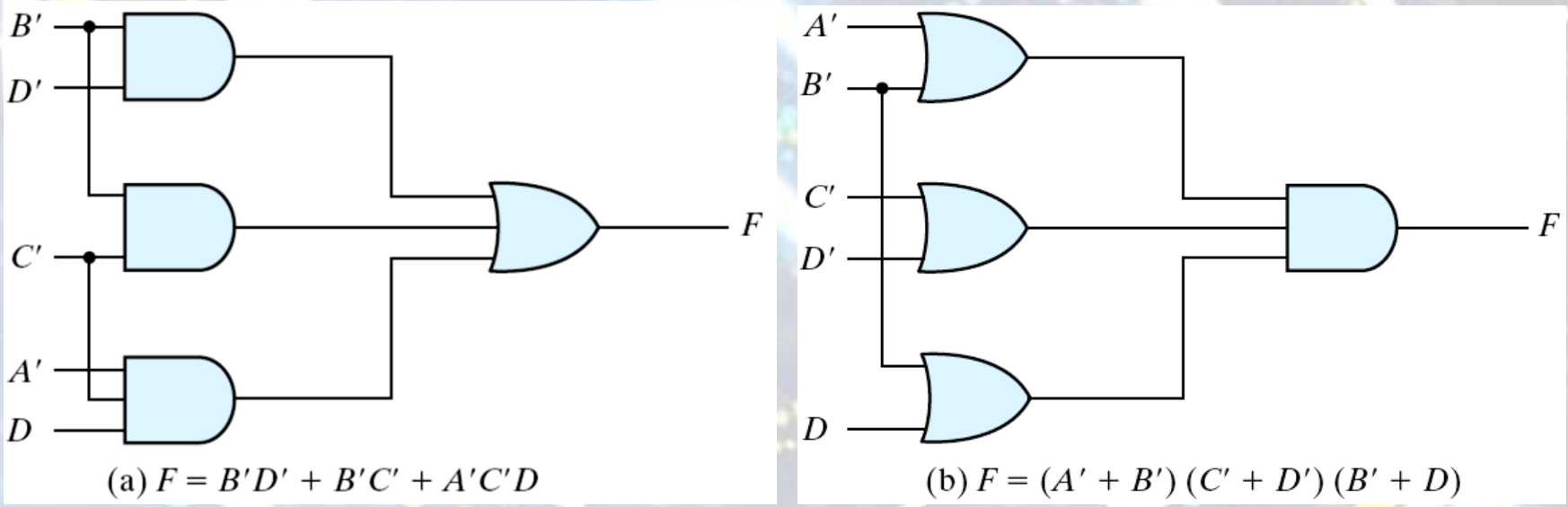
b) $F' = AB + CD + BD'$

- *Apply DeMorgan's theorem; F=(A'+B')(C'+D')(B'+D)*
- *Or think in terms of maxterms*

Figure 3.14 Map for Example 3.8, F(A, B, C, D)= Σ(0, 1, 2, 5, 8, 9, 10) = B'D'+B'C'+A'C'D

Example 3.8 (cont.)

- Gate implementation of the function of Example 3.8



Sum-of products form

Product-of sums form

Figure 3.15 Gate Implementation of the Function of Example 3.8

Sum-of-Minterm Procedure

- Consider the function defined in Table 3.2.
  - In sum-of-minterm:

$$F(x, y, z) = \sum(1, 3, 4, 6)$$

  - In sum-of-maxterm:

$$F'(x, y, z) = \Pi(0, 2, 5, 7)$$

  - Taking the complement of F

$$F(x, y, z) = (x' + z')(x + z)$$

Table 3.2
Truth Table of Function F

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Week -10

Page(161-184)

3-6 Don't-Care Conditions

- The value of a function is not specified for certain combinations of variables
  - BCD; 1010-1111: don't care
- The don't-care conditions can be utilized in logic minimization
  - Can be implemented as 0 or 1
- Example 3.9: simplify $F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$ which has the don't-care conditions $d(w, x, y, z) = \Sigma(0, 2, 5)$.

Example 3.9 (cont.)

- $F = yz + w'x'$; $F = yz + w'z$
- $F = \Sigma(1, 3, 7, 11, 15)$; $F = d\,(1, 3, 5, 7, 11, 15)$
- Either expression is acceptable

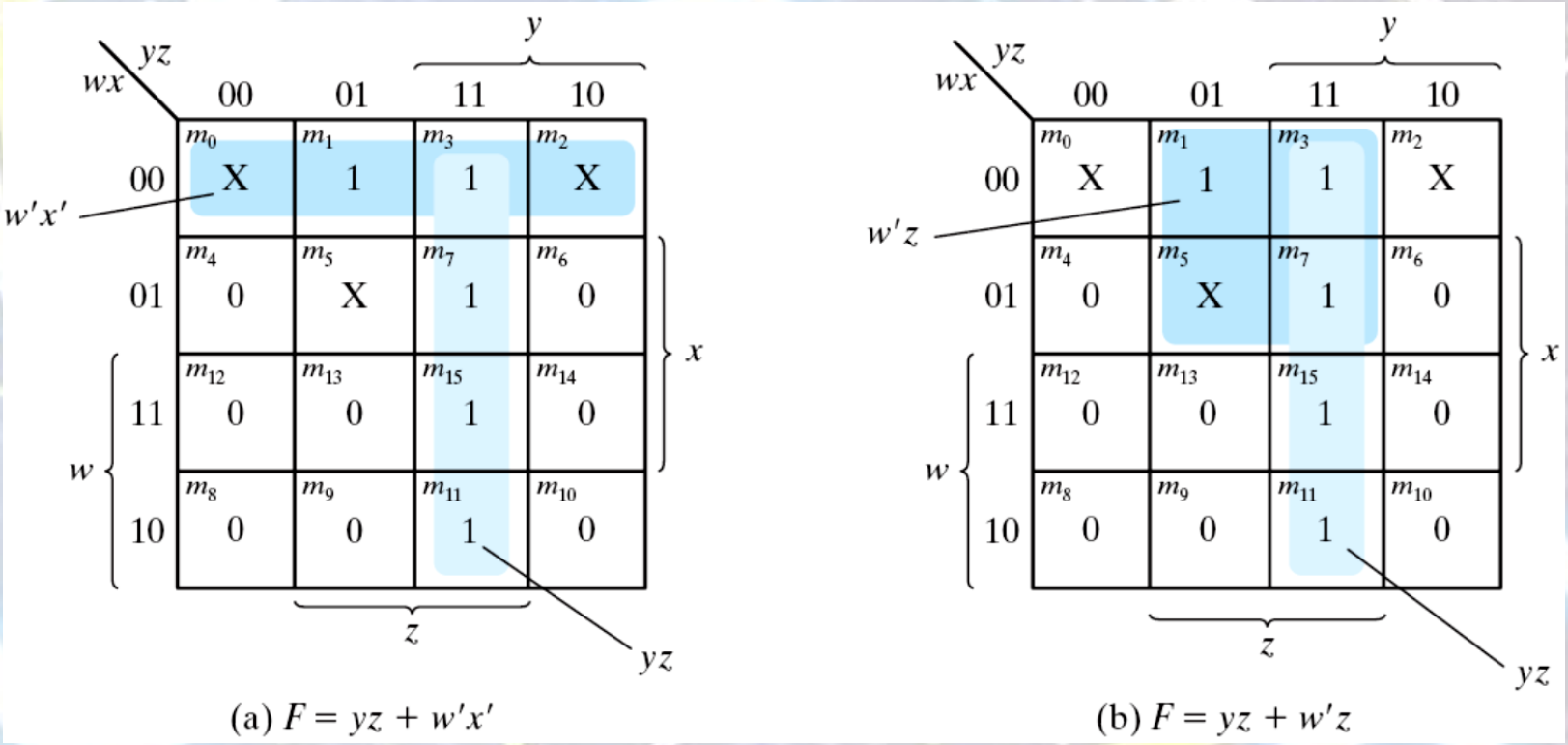


Figure 3.17 Example with don't-care Conditions

3-7 NAND and NOR Implementation

- NAND gate is a universal gate
  - Can implement any digital system

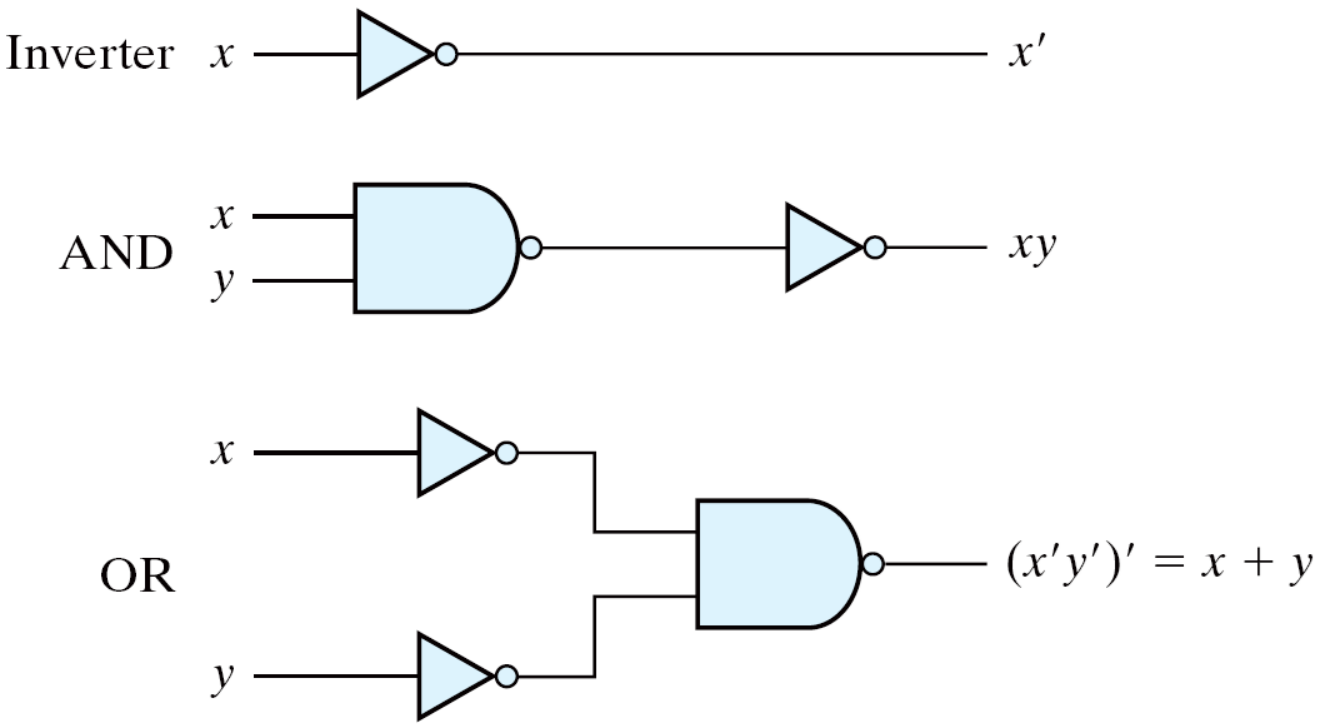


Figure 3.18 Logic Operations with NAND Gates

NAND Gate

- Two graphic symbols for a NAND gate

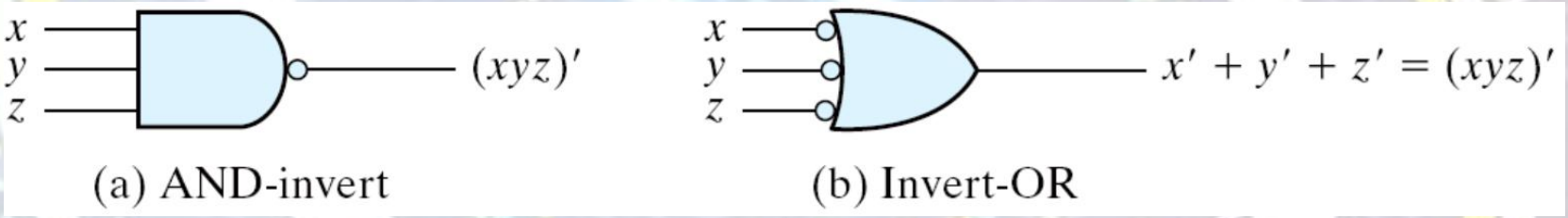
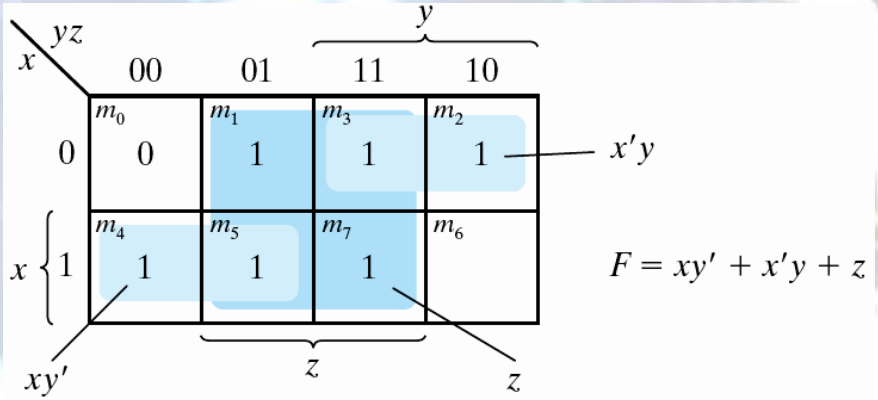


Figure 3.19 Two Graphic Symbols for NAND Gate

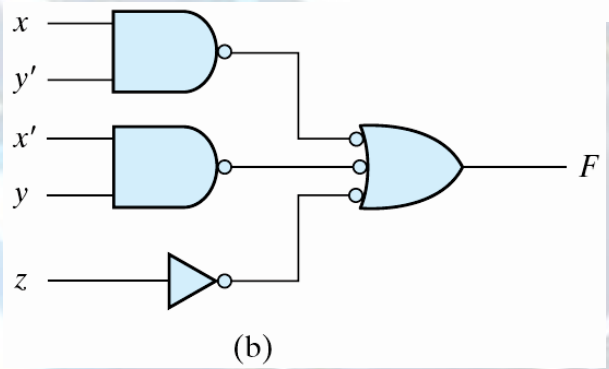
Example 3.10

- Example 3-10: implement $F(x, y, z) =$

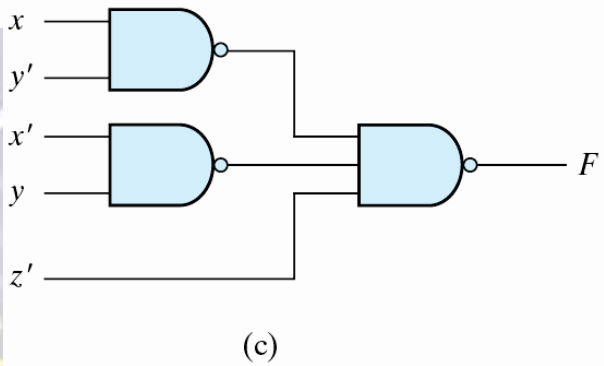
$$F(x,y,z) = \sum(1,2,3,4,5,7) \quad \Rightarrow \quad F(x,y,z) = xy' + x'y + z$$



(a)



(b)



(c)

Figure 3.21 Solution to Example 3-10

Multilevel NAND Circuits

- Boolean function implementation
  - AND-OR logic → NAND-NAND logic
    - » AND → AND + inverter
    - » OR: inverter + OR = NAND
    - » For every bubble that is not compensated by another inverter.

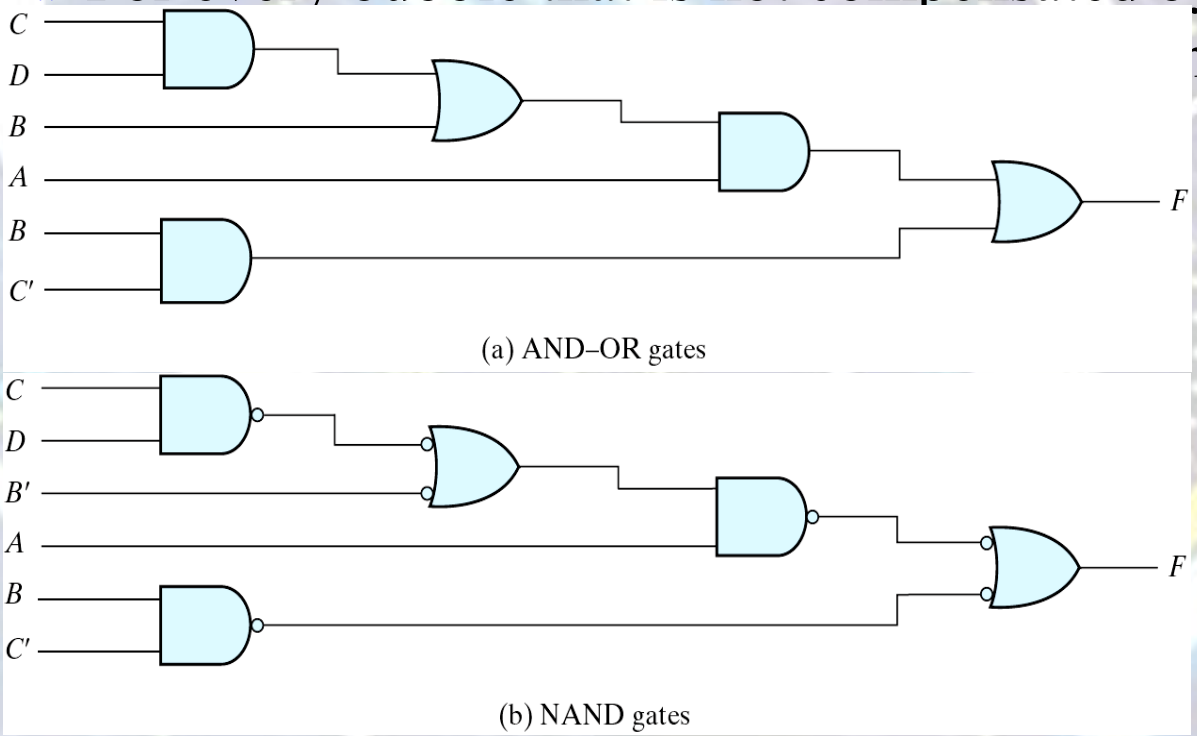
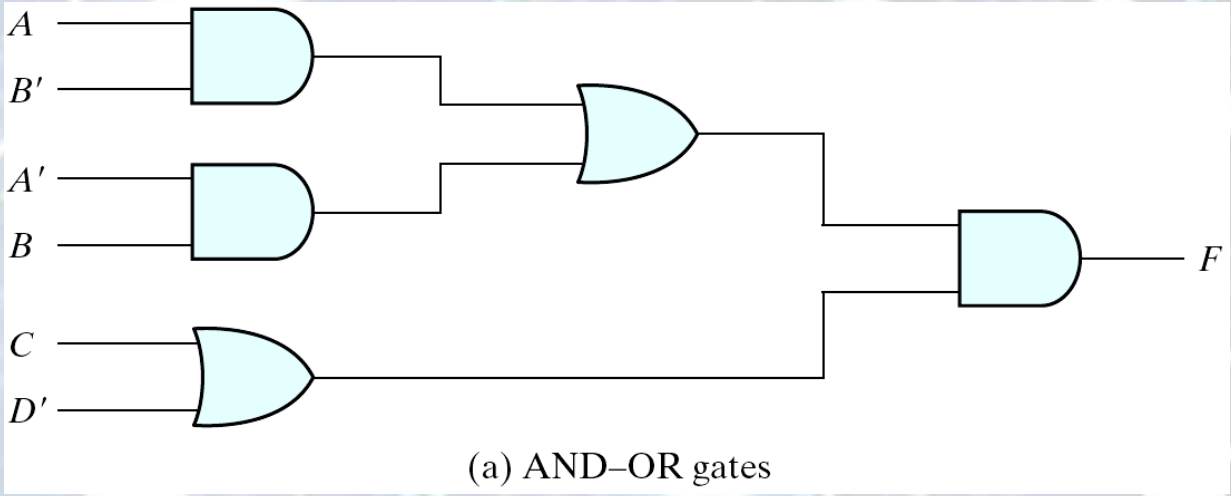
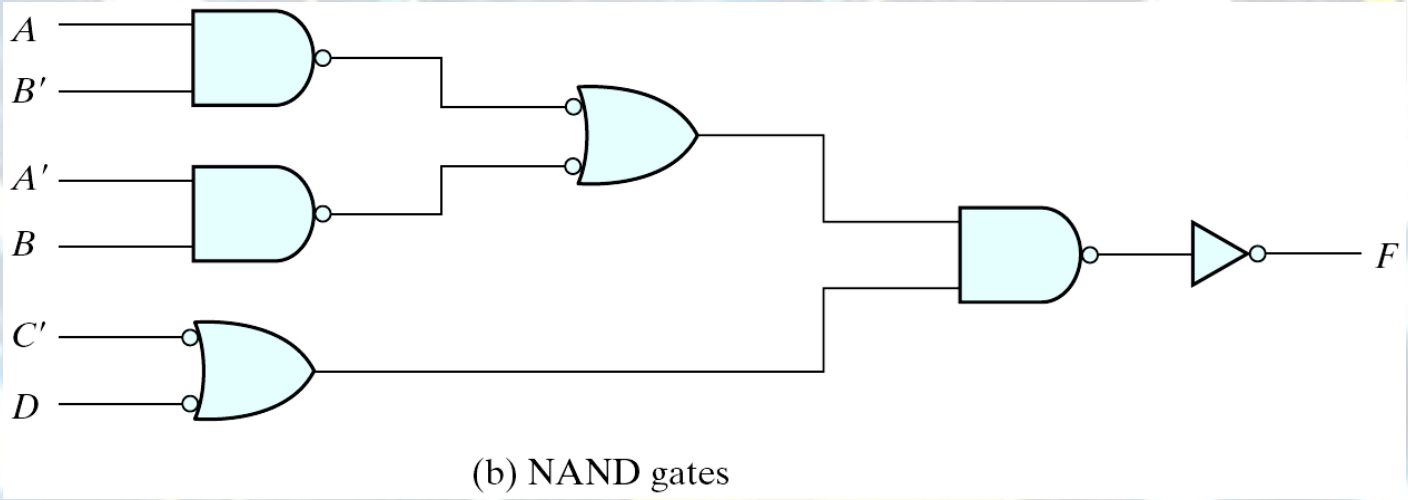


Figure 3.22 Implementing F = A(CD + B) + BC′

NAND Implementation



(a) AND–OR gates



(b) NAND gates

Figure 3.23 Implementing F = (AB′ +A′B)(C+ D′)

NOR Implementation

- NOR function is the dual of NAND function.
- The NOR gate is also universal.

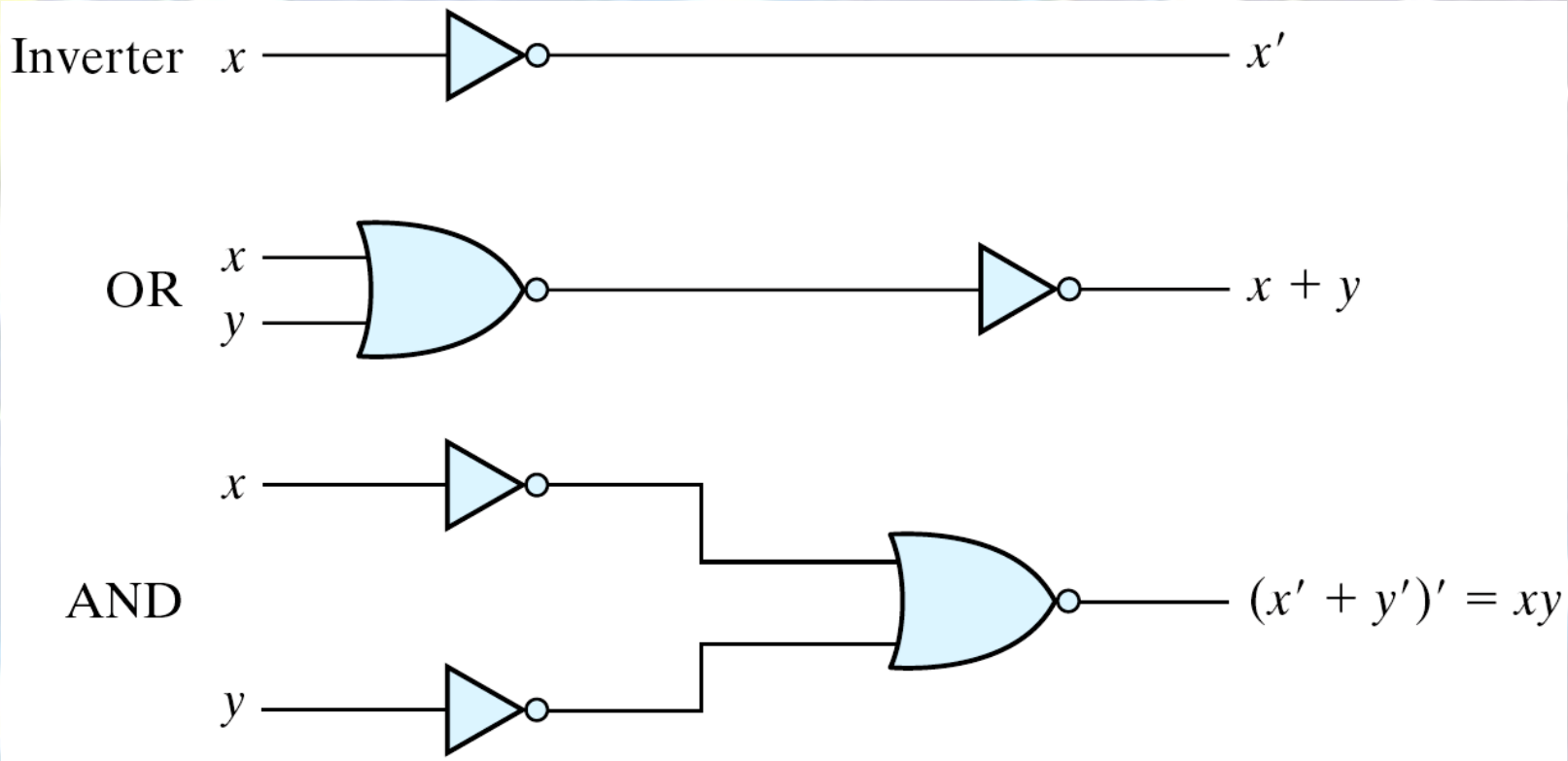


Figure 3.24 Logic Operation with NOR Gates

Two Graphic Symbols for a NOR Gate

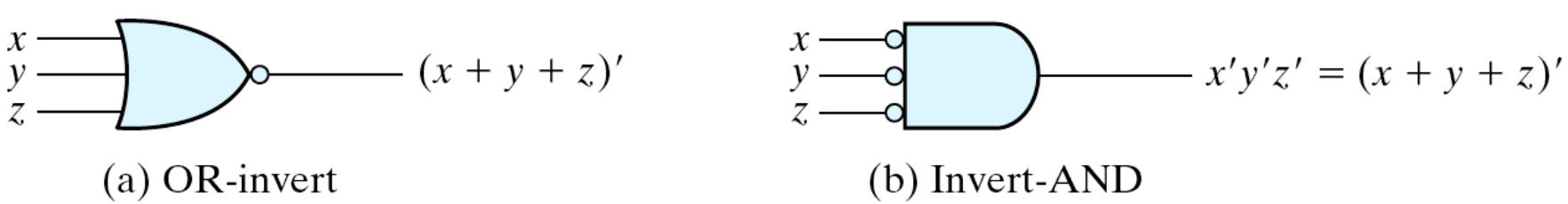


Figure 3.25 Two Graphic Symbols for NOR Gate

Example: F = (A + B)(C + D)E

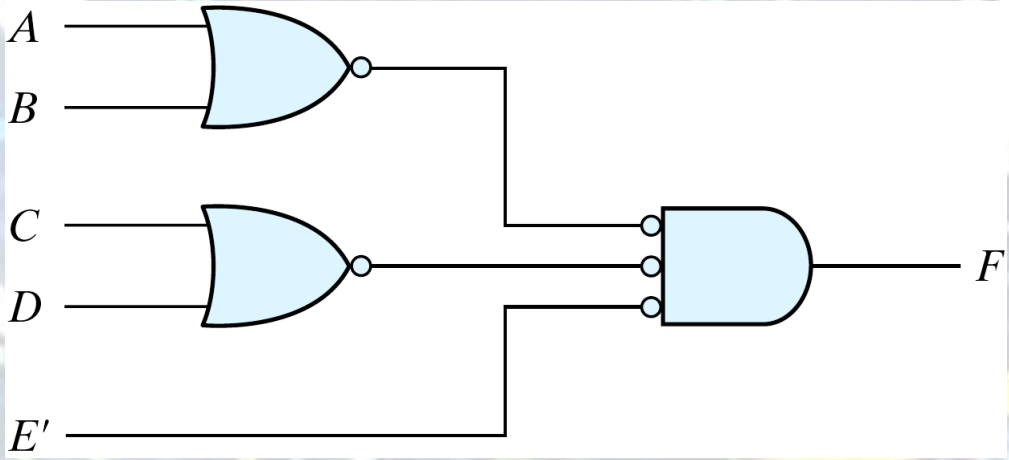


Figure 3.26 Implementing F = (A + B)(C + D)E

Example

Example: F = (AB′ +A′B)(C + D′)

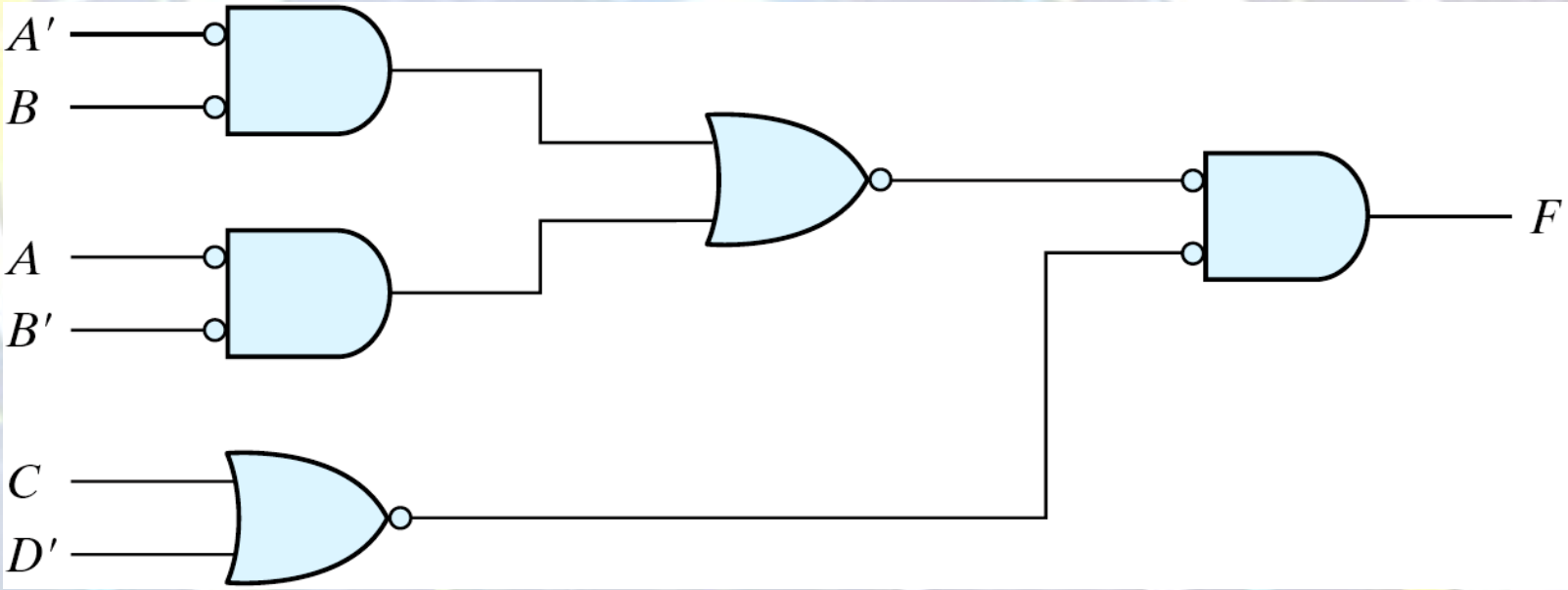


Figure 3.27 Implementing F = (AB′ +A′B)(C + D′) with NOR gates

3-8 Other Two-level Implementations (

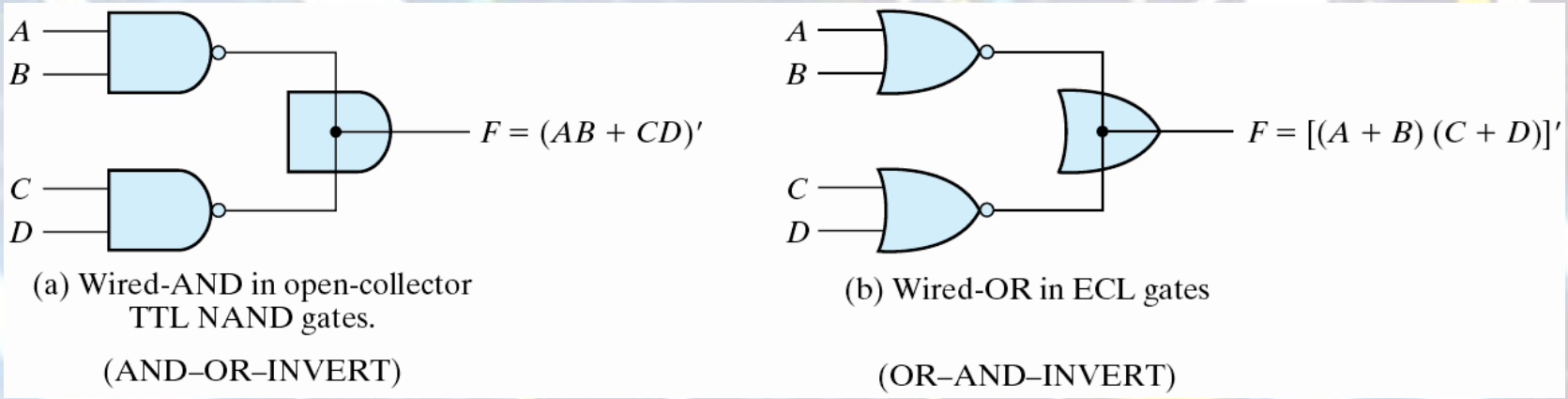
- Wired logic
  - A wire connection between the outputs of two gates
  - Open-collector TTL NAND gates: wired-AND logic
  - The NOR output of ECL gates: wired-OR logic

$$F = (AB)' \cdot (CD)' = (AB + CD)' = (A' + B')(C' + D')$$

$$F = (A + B)' + (C + D)' = [(A + B)(C + D)]'$$

AND-OR-INVERT function

OR-AND-INVERT function



(a) Wired-AND in open-collector TTL NAND gates.

(AND–OR–INVERT)

(b) Wired-OR in ECL gates

(OR–AND–INVERT)

Figure 3.28 Wired Logic

AND-OR-Invert Implementation

- AND-OR-INVERT (AOI) Implementation
  - NAND-AND = AND-NOR = AOI
  - $F = (AB+CD+E)'$
  - $F' = AB+CD+E$ (sum of products)

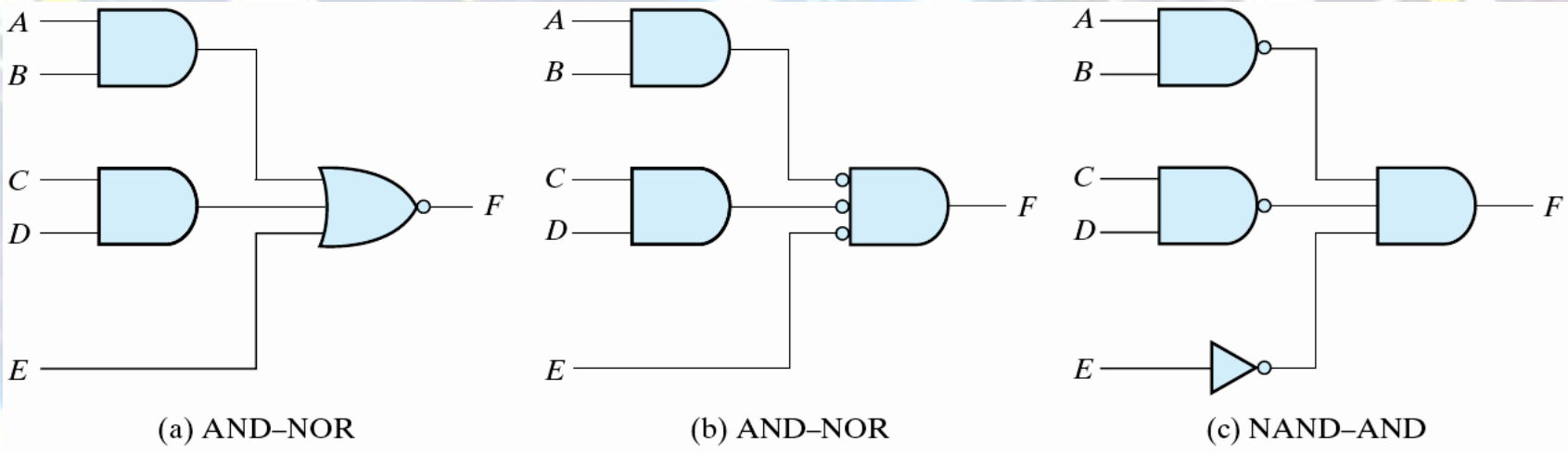


Figure 3.29 AND-OR-INVERT circuits, F = (AB +CD +E)′

OR-AND-Invert Implementation

- OR-AND-INVERT (OAI) Implementation
  - OR-NAND = NOR-OR = OAI
  - $F = ((A+B)(C+D)E)'$
  - $F' = (A+B)(C+D)E$ (product of sums)

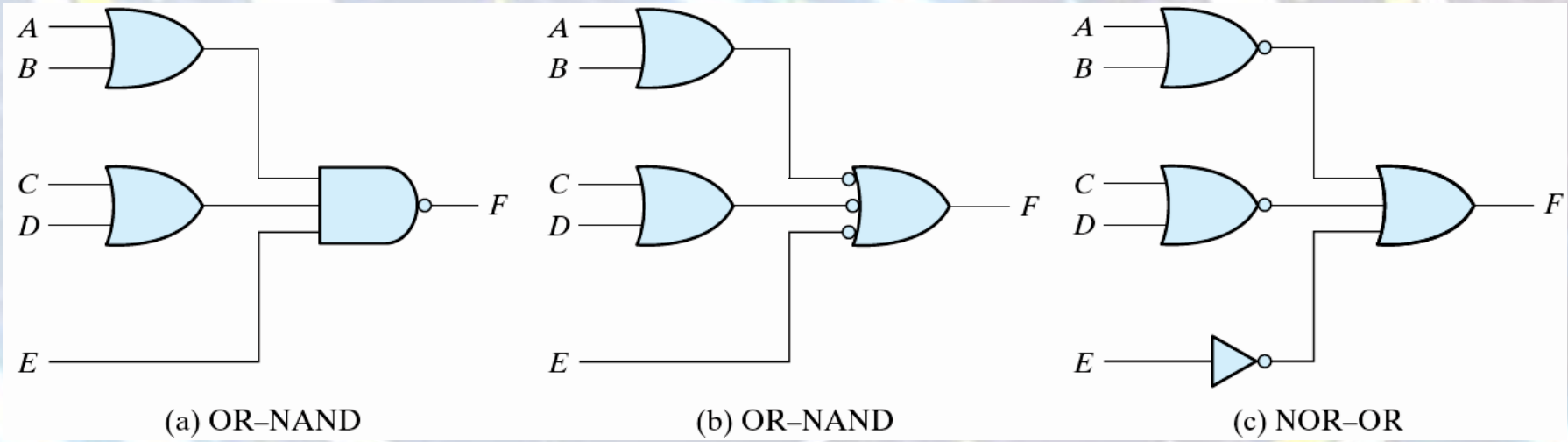
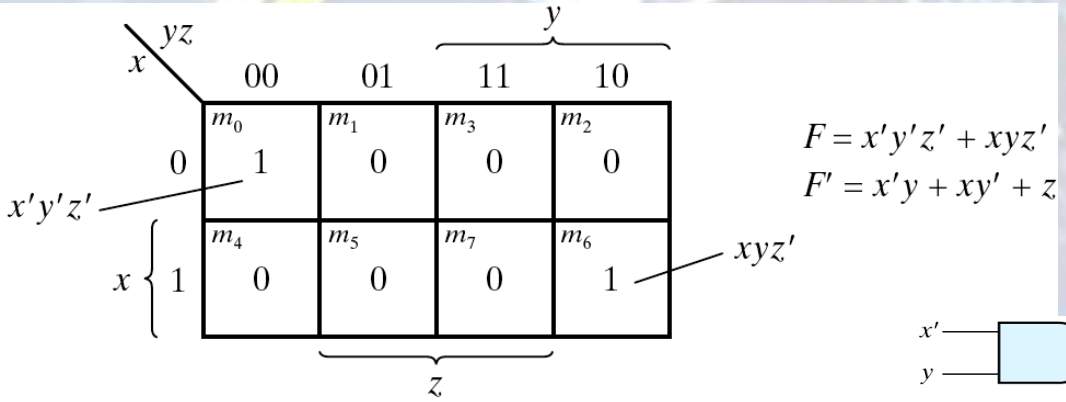


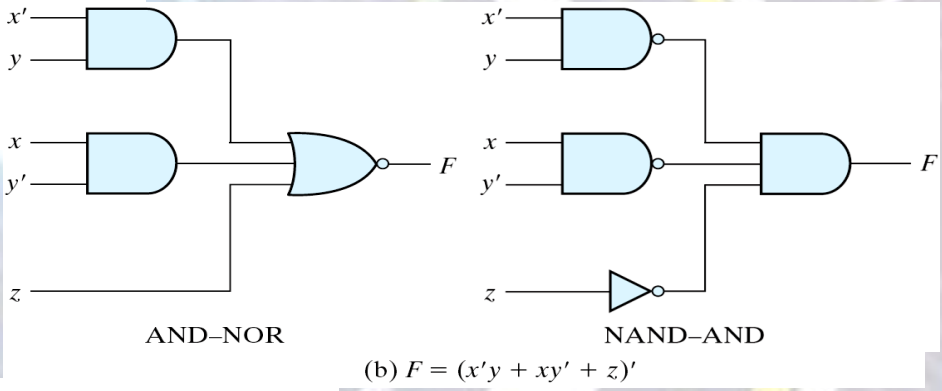
Figure 3.30 OR-AND-INVERT circuits, F = ((A+B)(C+D)E)'



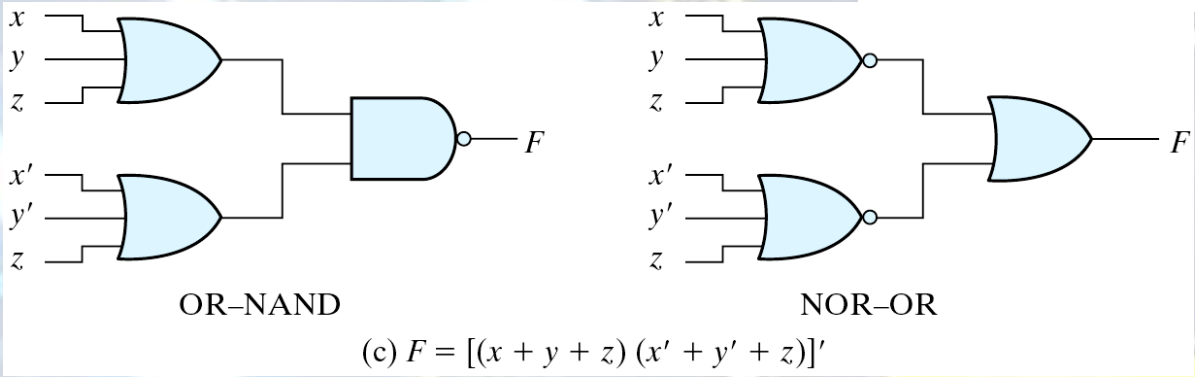
$F = x'y'z' + xyz'$

$F' = x'y + xy' + z$

(a) Map simplification in sum of products



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



(c) $F = [(x + y + z)(x' + y' + z)]'$

Figure 3.31 Other Two-level Implementations

Exclusive-OR Implementations

- Implementations
  - $(x'+y')x + (x'+y')y = xy'+x'y = x \oplus y$

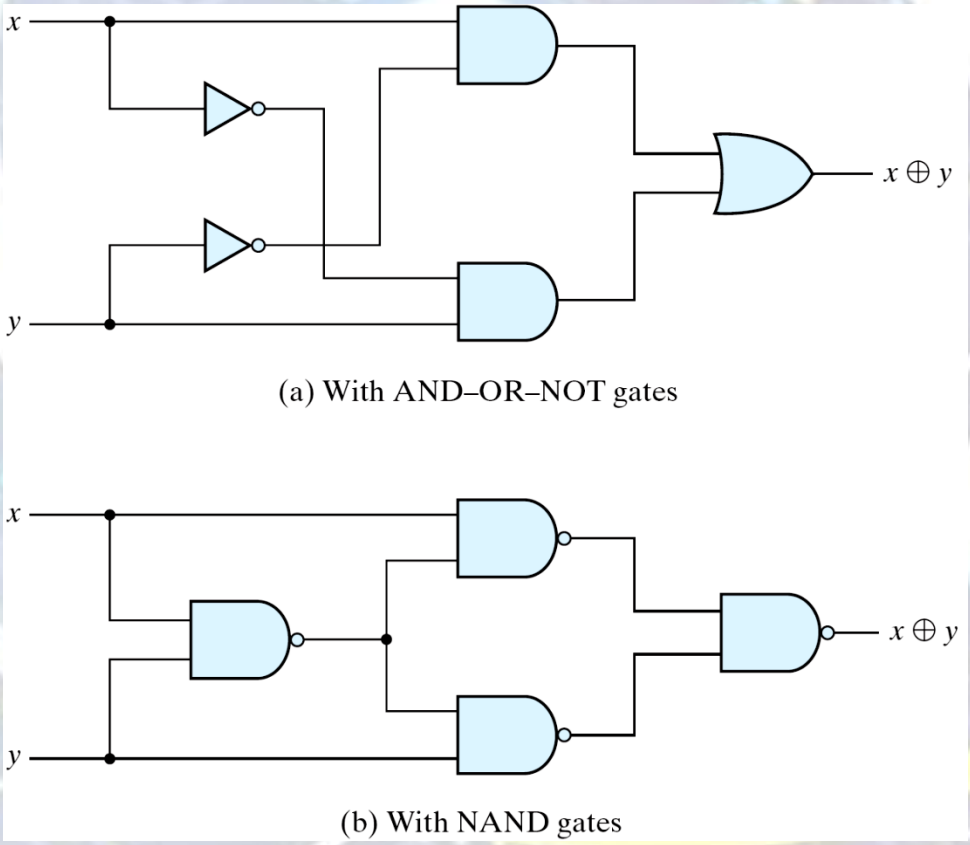


Figure 3.32 Exclusive-OR Implementations

Odd Function

- $A \oplus B \oplus C = (AB' + A'B)C' + (AB + A'B')C = AB'C' + A'BC' + ABC + A'B'C = \Sigma(1, 2, 4, 7)$
- XOR is a odd function → an odd number of 1's, then $F = 1$.
- XNOR is a even function → an even number of 1's, then $F = 1$.

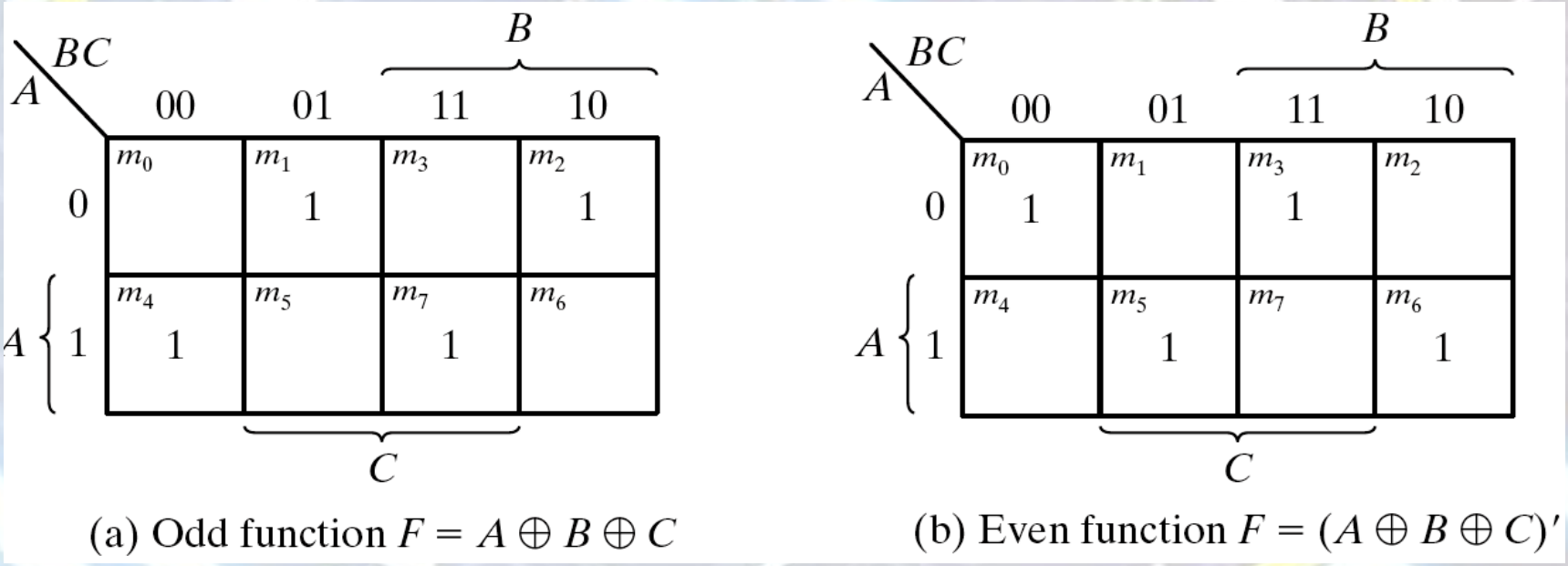


Figure 3.33 Map for a Three-variable Exclusive-OR Function

XOR and XNOR

- Logic diagram of odd and even functions

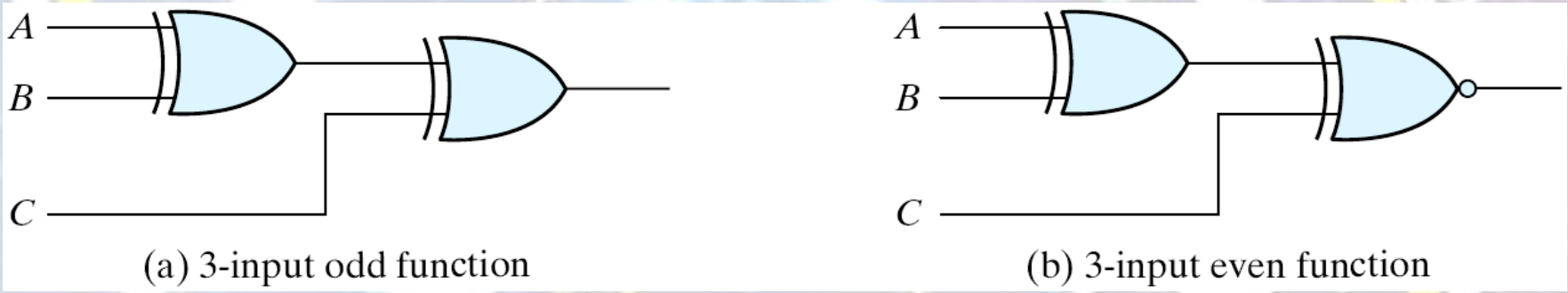


Figure 3.34 Logic Diagram of Odd and Even Functions

Parity Generation and Checking

Table 3.4
Even-Parity-Generator Truth Table

Three-Bit Message			Parity Bit
x	*y*	*z*	*P*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Parity Generation and Checking

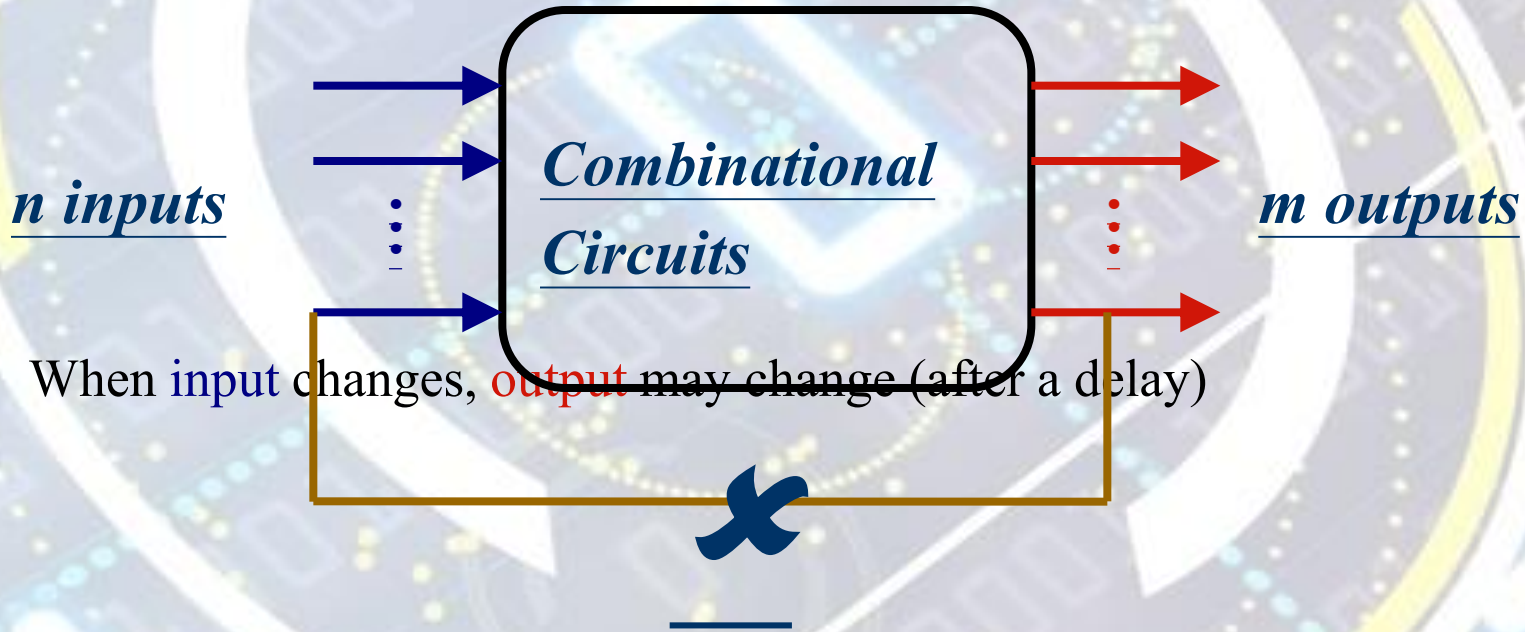
Table 3.5
Even-Parity-Checker Truth Table

Four Bits Received				Parity Error Check
x	*y*	*z*	*P*	*C*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Combinational Circuits

- Output is function of input only

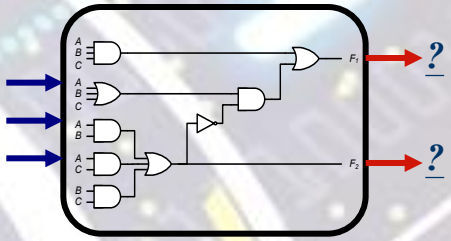
  i.e. no feedback



When input changes, output may change (after a delay)

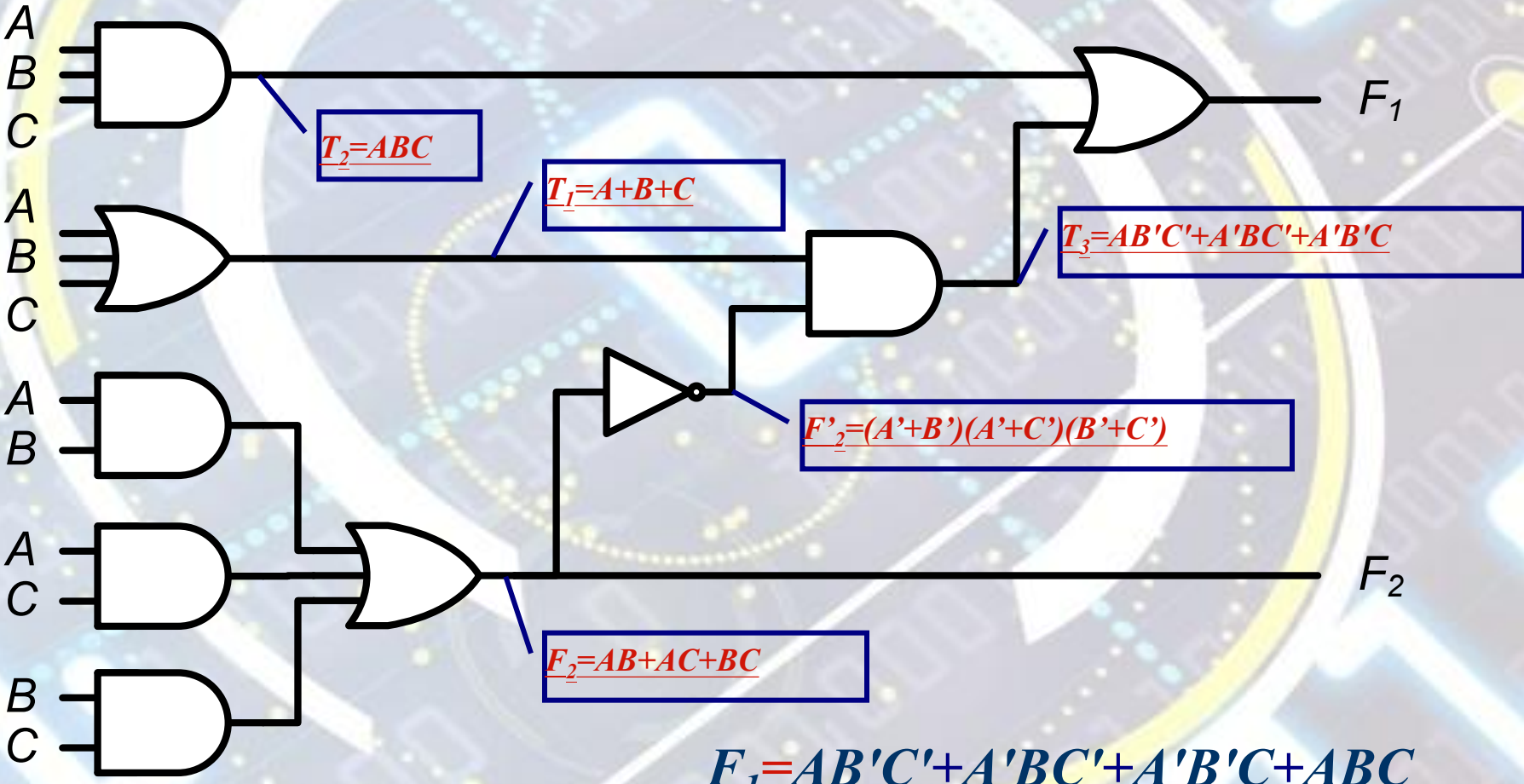
Combinational Circuits

- Analysis
  - Given a circuit, find out its *function*
  - Function may be expressed as:
    - » Boolean function
    - » Truth table
- Design
  - Given a desired function, determine its *circuit*
  - Function may be expressed as:
    - » Boolean function
    - » Truth table



Analysis Procedure

- Boolean Expression Approach

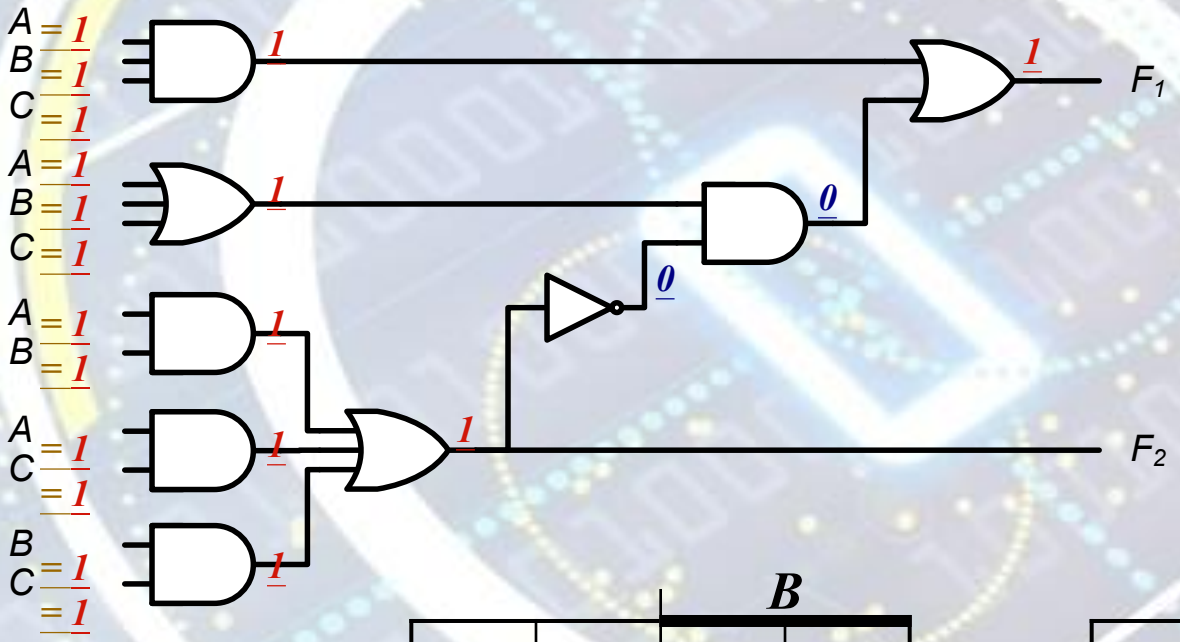


$$F_1=AB'C'+A'BC'+A'B'C+ABC$$

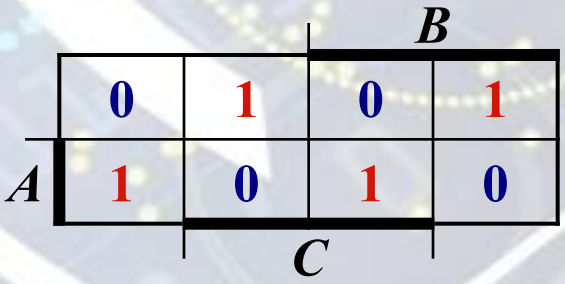
$$F_2=AB+AC+BC$$

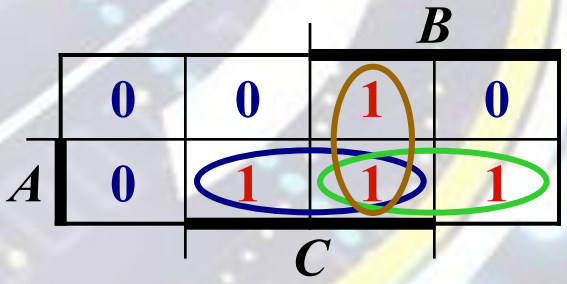
Analysis Procedure

- Truth Table Approach



A B C	F_1	F_2
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





$F_1 = AB'C' + A'BC' + A'B'C + ABC$

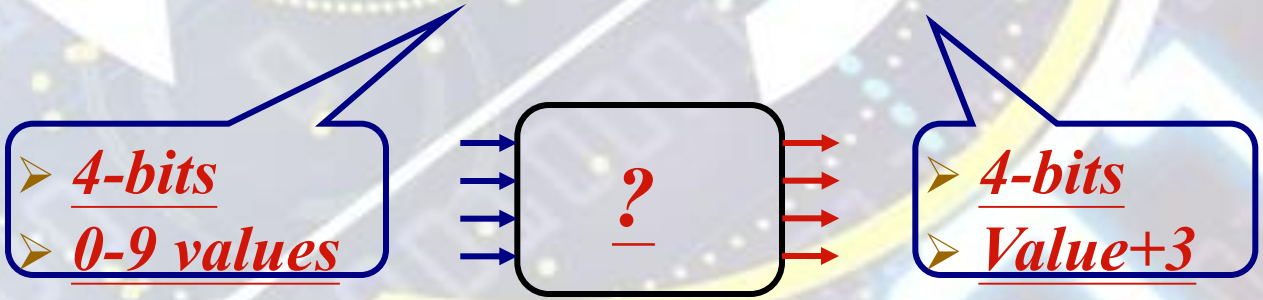
$F_2 = AB + AC + BC$

Design Procedure

- Given a problem statement:
  - Determine the number of *inputs* and *outputs*
  - Derive the truth table
  - Simplify the Boolean expression for each output
  - Produce the required circuit

Example:

Design a circuit to convert a “BCD” code to “Excess 3” code



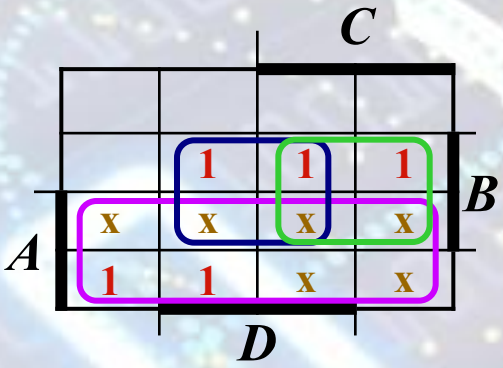
Week -11

Page(186-188)

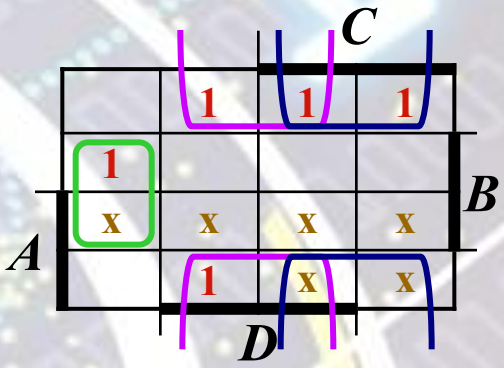
Design Procedure

- BCD-to-Excess 3 Converter

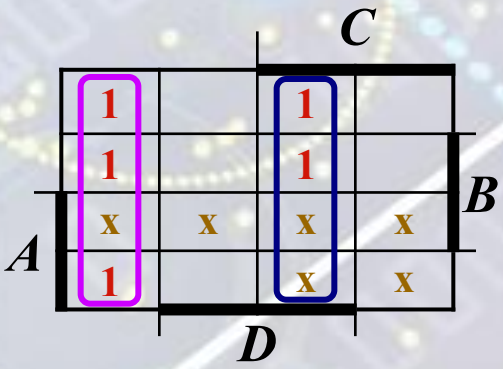
A B C D	w x y z
0 0 0 0	0 0 1 1
0 0 0 1	0 1 0 0
0 0 1 0	0 1 0 1
0 0 1 1	0 1 1 0
0 1 0 0	0 1 1 1
0 1 0 1	1 0 0 0
0 1 1 0	1 0 0 1
0 1 1 1	1 0 1 0
1 0 0 0	1 0 1 1
1 0 0 1	1 1 0 0
1 0 1 0	x x x x
1 0 1 1	x x x x
1 1 0 0	x x x x
1 1 0 1	x x x x
1 1 1 0	x x x x
1 1 1 1	x x x x



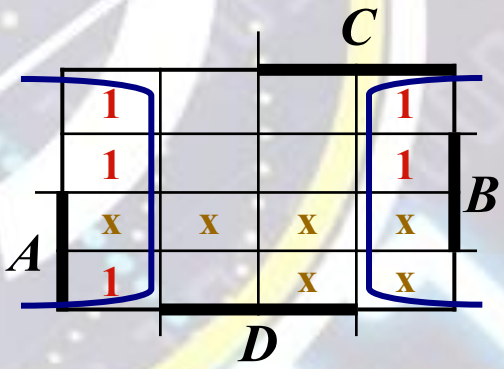
$w = A+BC+BD$



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



$y = C'D'+CD$

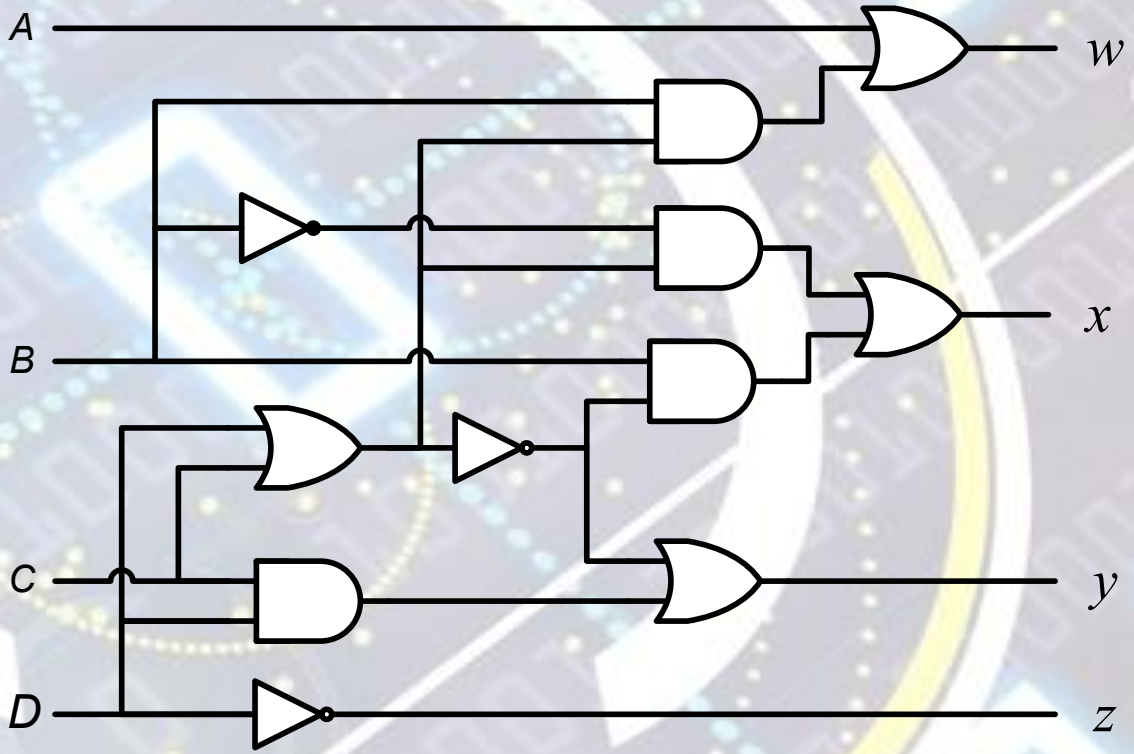


$z = D'$

Design Procedure

- BCD-to-Excess 3 Converter

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



$w = A + B(C+D)$

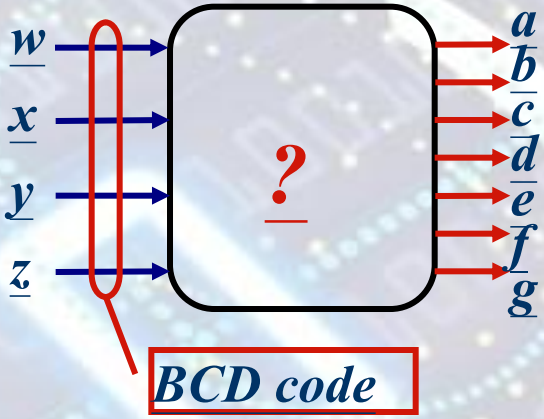
$x = B'(C+D) + B(C+D)'$

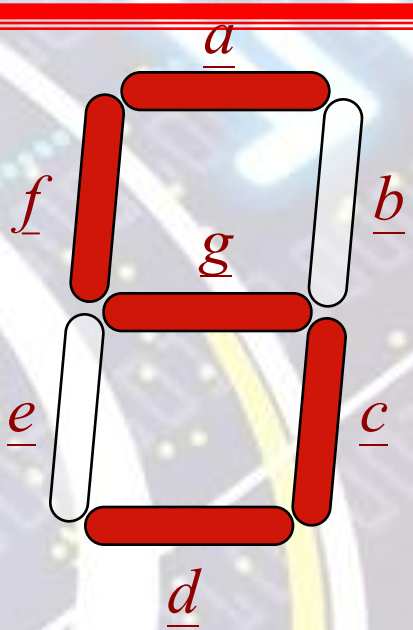
$y = (C+D)' + CD$

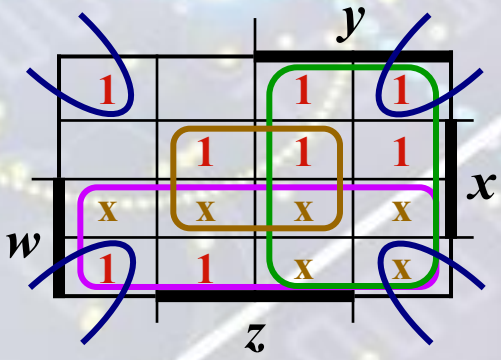
$z = D'$

Seven-Segment Decoder

w x y z	a b c d e f g
0 0 0 0	1 1 1 1 1 1 0
0 0 0 1	0 1 1 0 0 0 0
0 0 1 0	1 1 0 1 1 0 1
0 0 1 1	1 1 1 1 0 0 1
0 1 0 0	0 1 1 0 0 1 1
0 1 0 1	1 0 1 1 0 1 1
0 1 1 0	1 0 1 1 1 1 1
0 1 1 1	1 1 1 0 0 0 0
1 0 0 0	1 1 1 1 1 1 1
1 0 0 1	1 1 1 1 0 1 1
1 0 1 0	x x x x x x x
1 0 1 1	x x x x x x x
1 1 0 0	x x x x x x x
1 1 0 1	x x x x x x x
1 1 1 0	x x x x x x x
1 1 1 1	x x x x x x x







$a = w + y + xz + x'z'$

$b = \ldots$

$c = \ldots$

$d = \ldots$

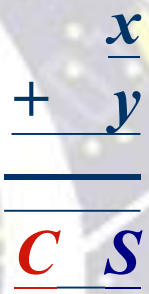
Week -12

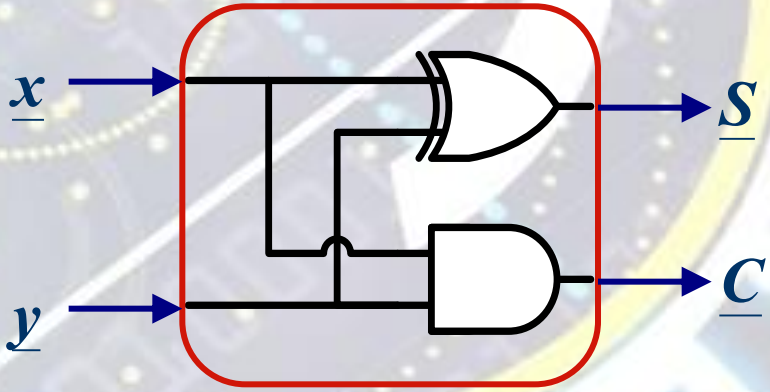
Page(190-211)

Binary Adder

- Half Adder
  - Adds 1-bit plus 1-bit
  - Produces Sum and Carry

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

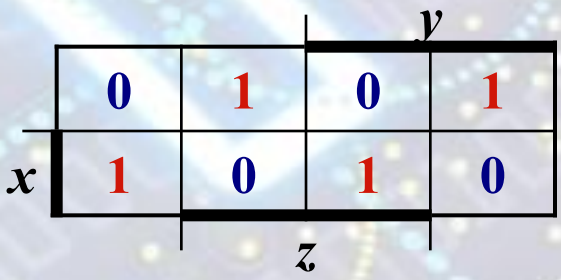




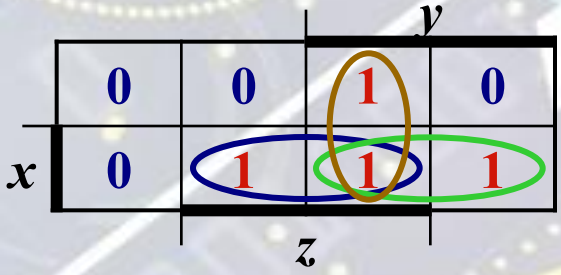
Binary Adder

- Full Adder
  - Adds 1-bit plus 1-bit plus 1-bit
  - Produces Sum and Carry

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



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



$$C = xy + xz + yz$$

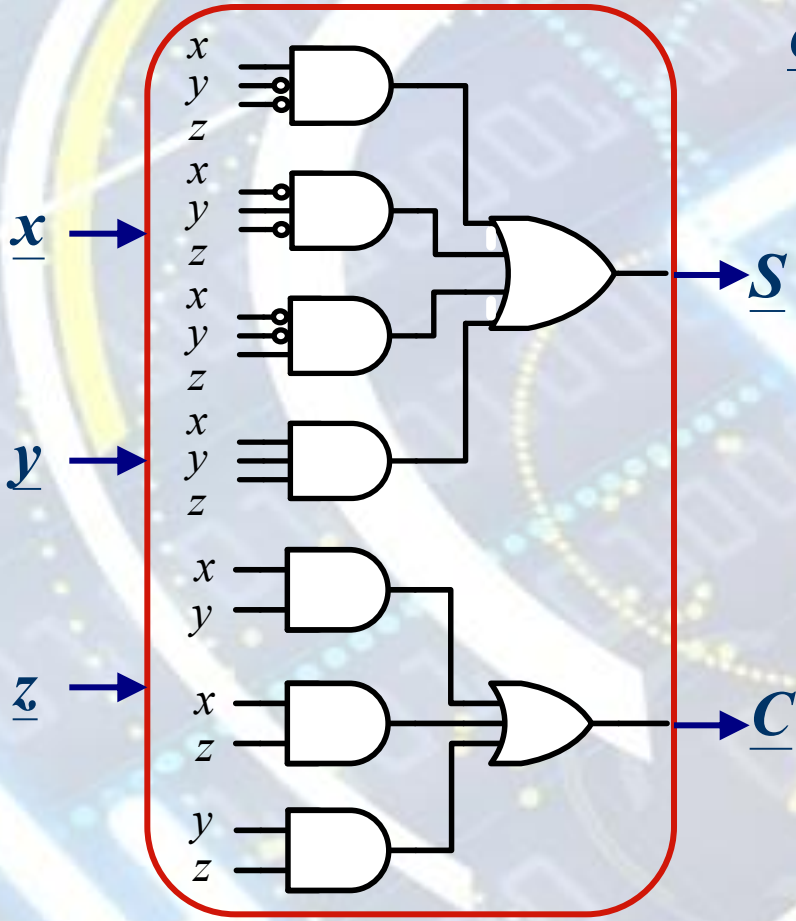
$$\begin{array}{r} x \\ + \quad y \\ + \quad z \\ \hline C \quad S \end{array}$$

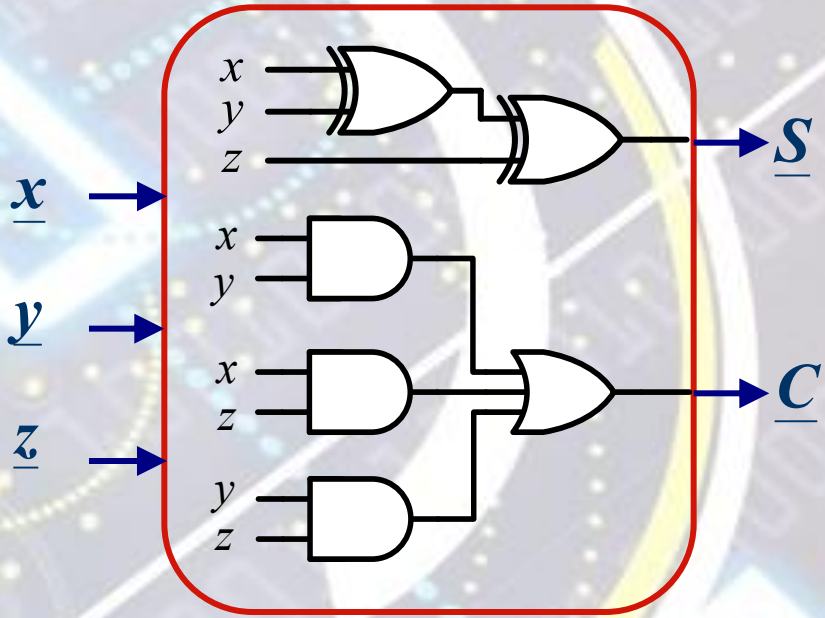
Binary Adder

- Full Adder

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

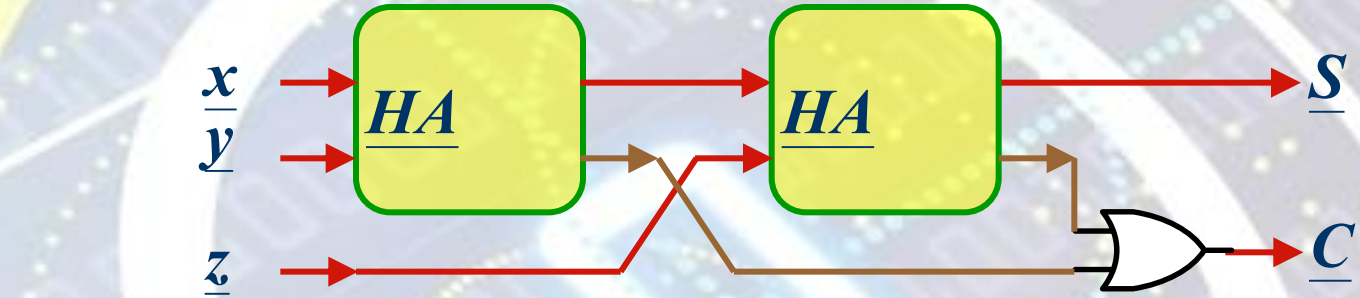
$C = xy + xz + yz$

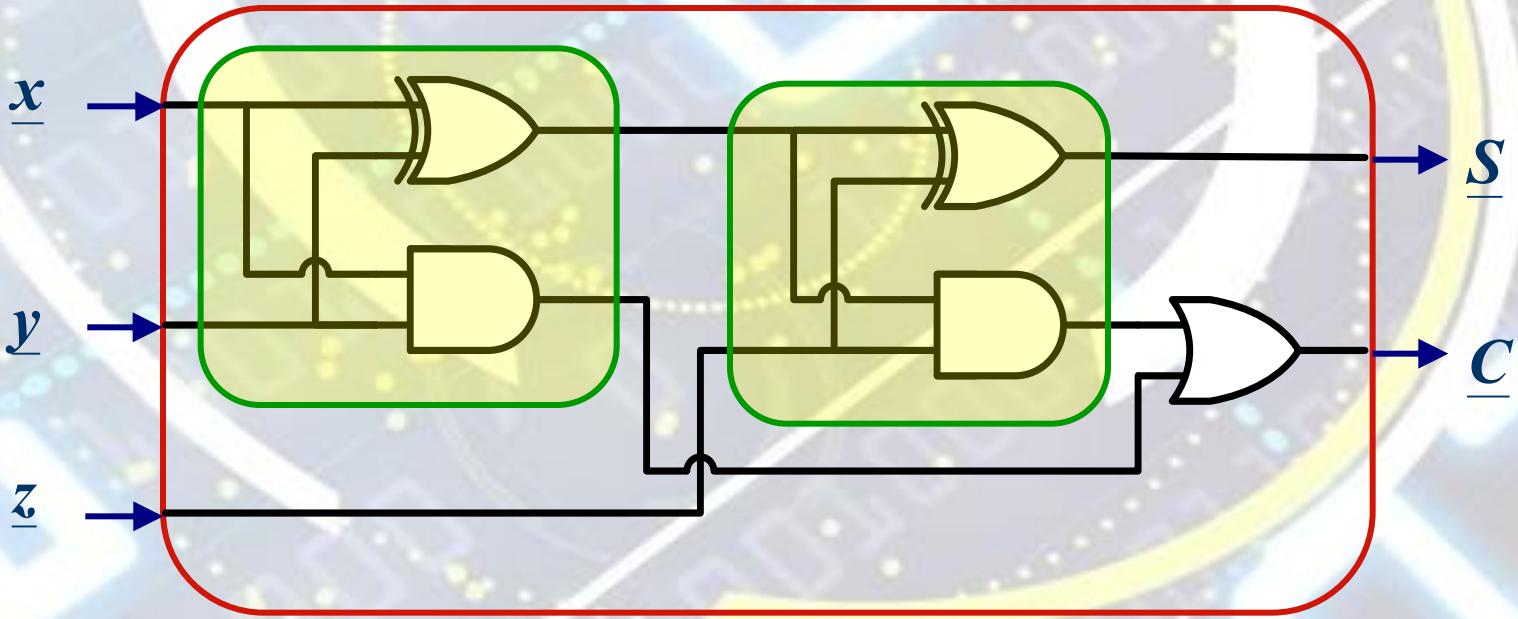




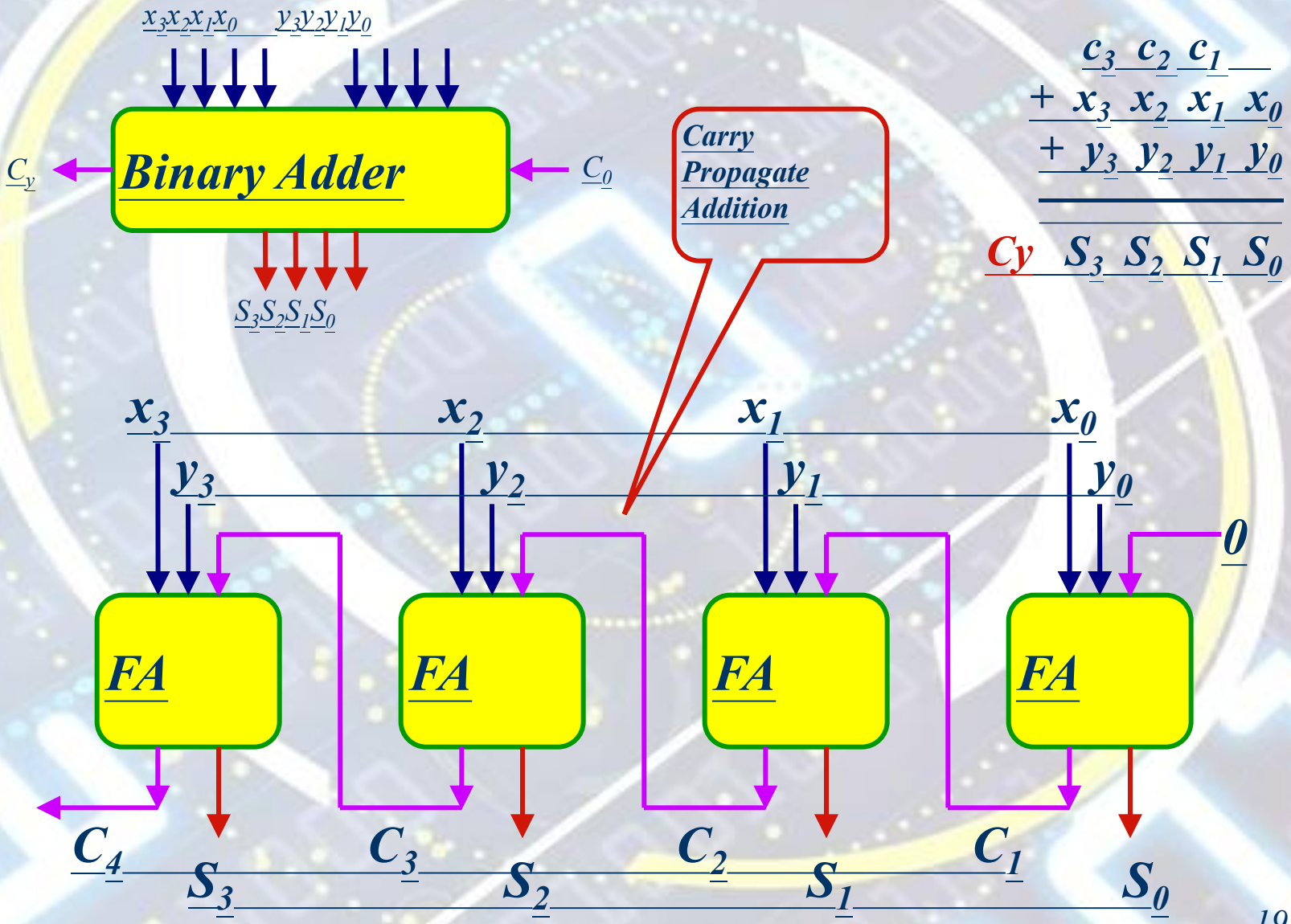
Binary Adder

- Full Adder



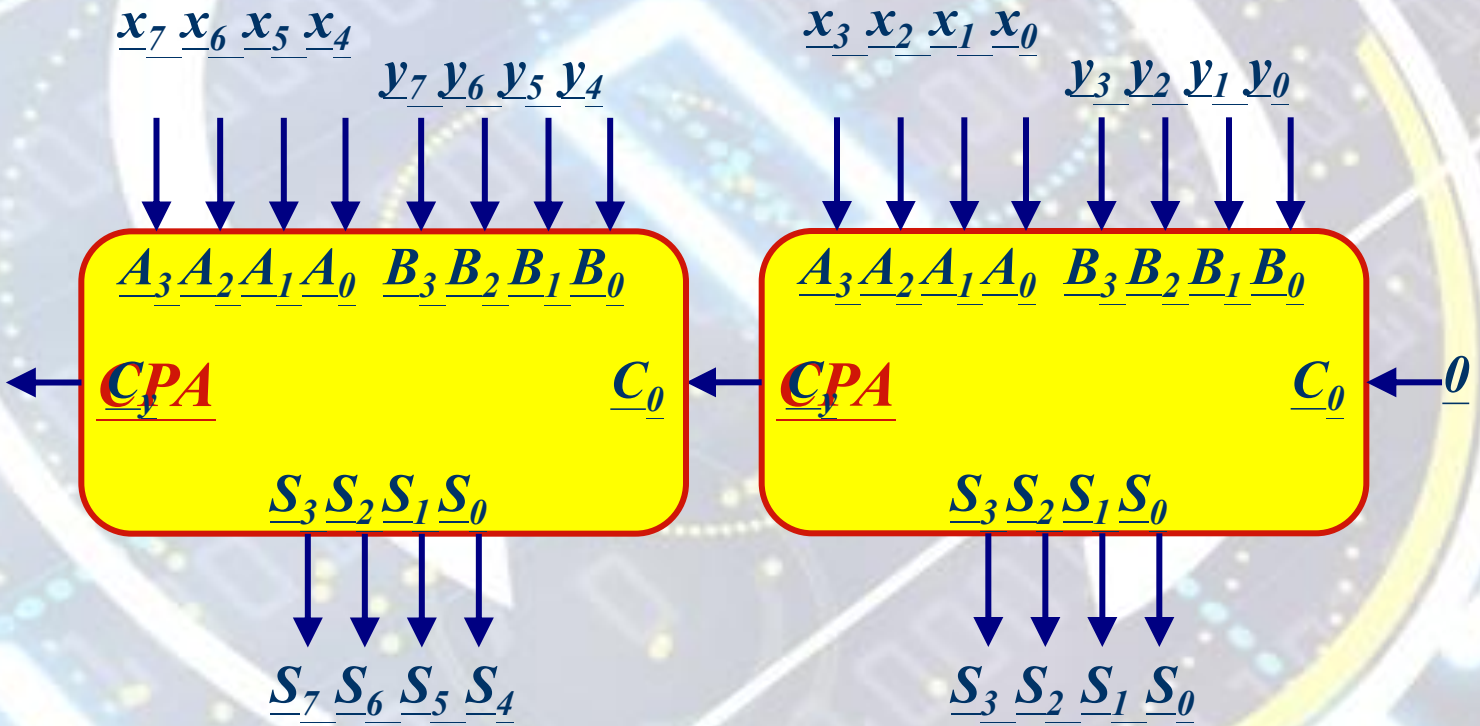


Binary Adder



Binary Adder

- Carry Propagate Adder



- Carry propagation
  - When the correct outputs are available
  - The critical path counts (the worst case)
  - $(A_1, B_1, C_1) \rightarrow C_2 \rightarrow C_3 \rightarrow C_4 \rightarrow (C_5, S_4)$
  - When 4-bits full-adder $\rightarrow$ 8 gate levels (n-bits: $2n$ gate levels)

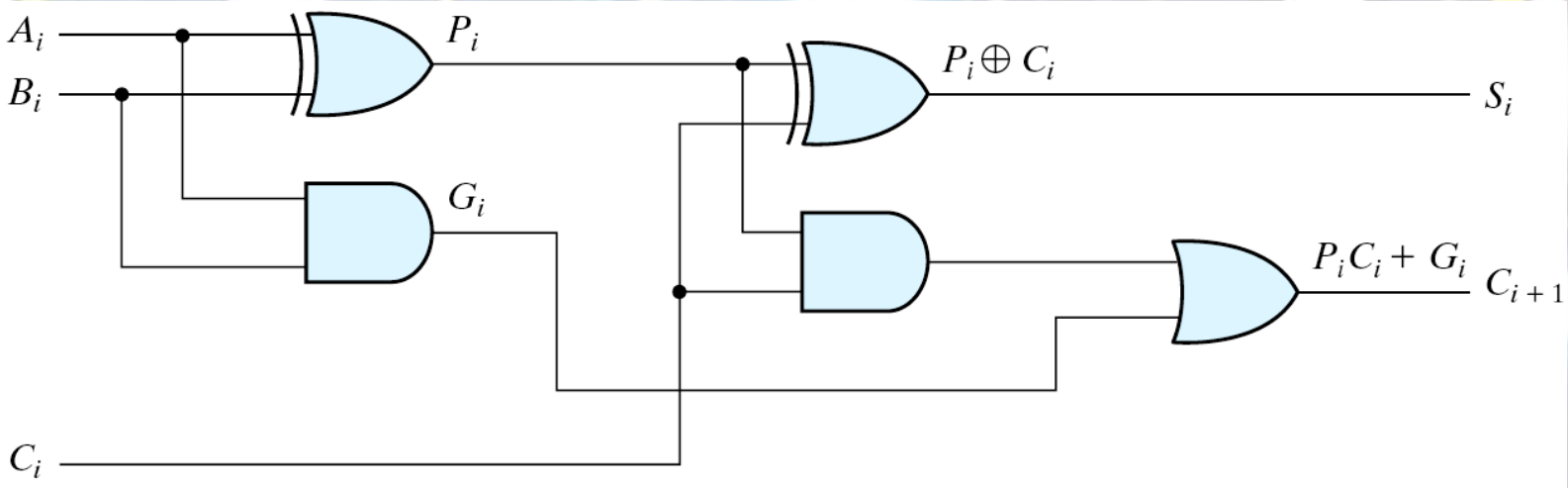
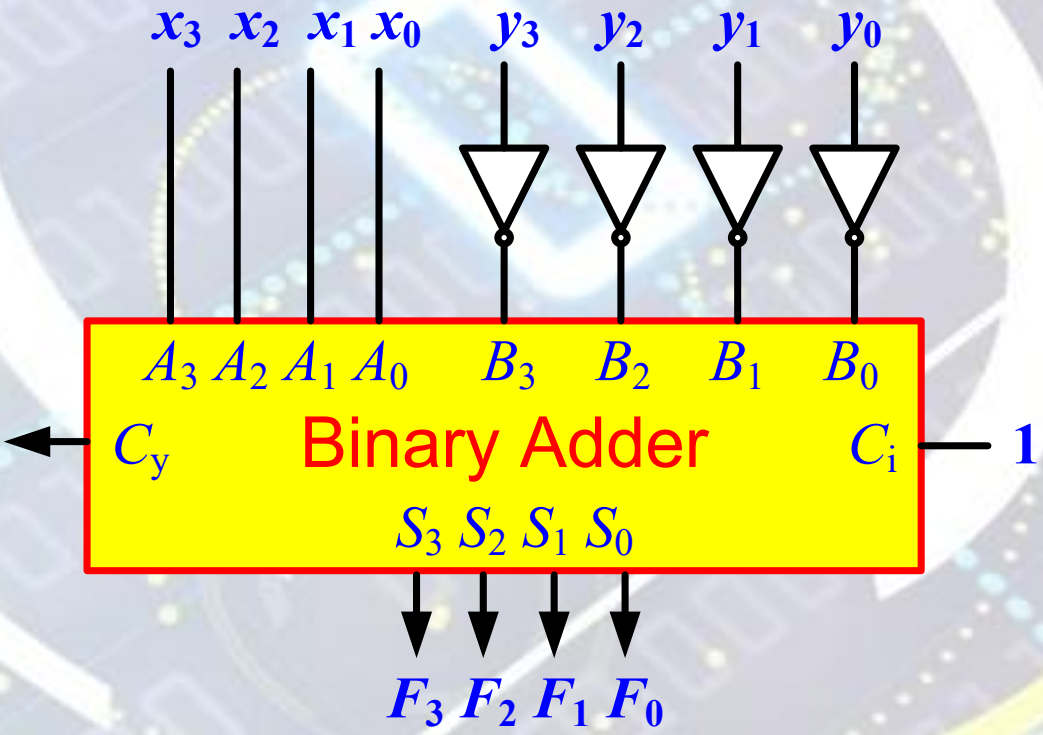


Figure 4.10 Full Adder with P and G Shown

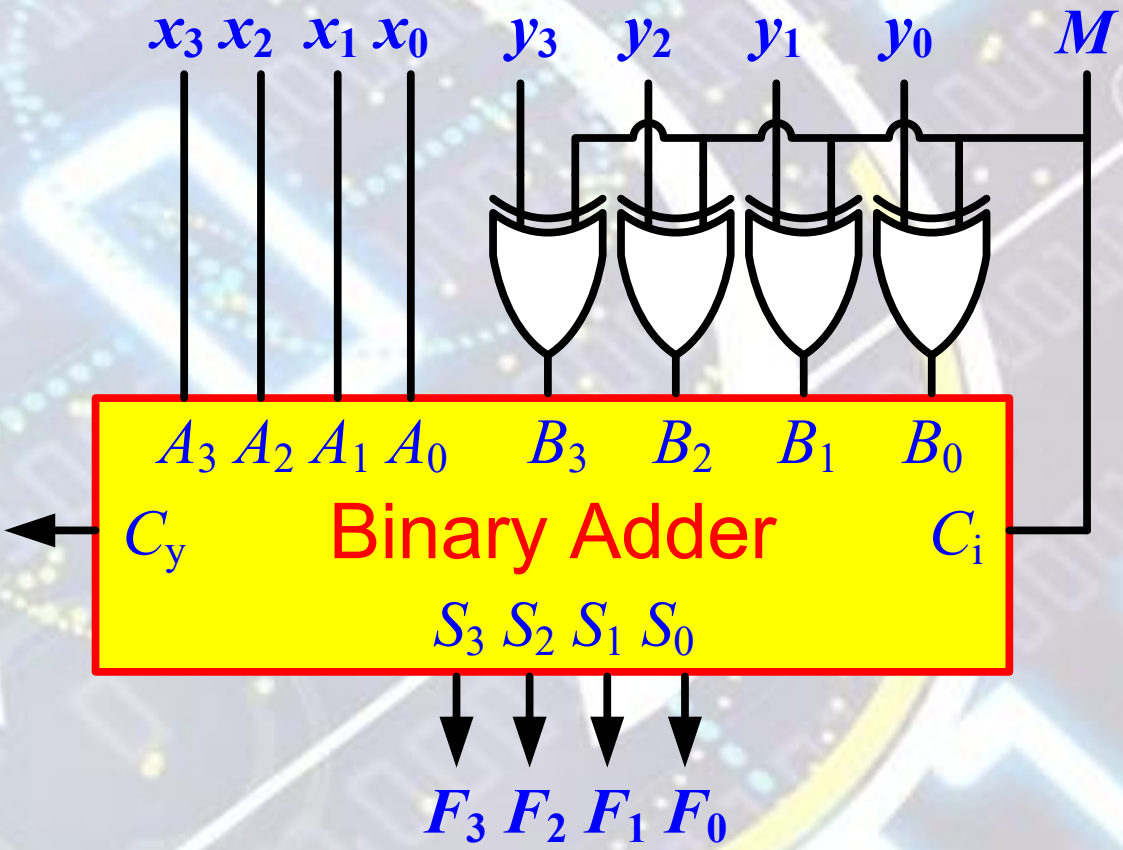
Binary Subtractor

- Use 2's complement with binary adder
  - $x - y = x + (-y) = x + y' + 1$



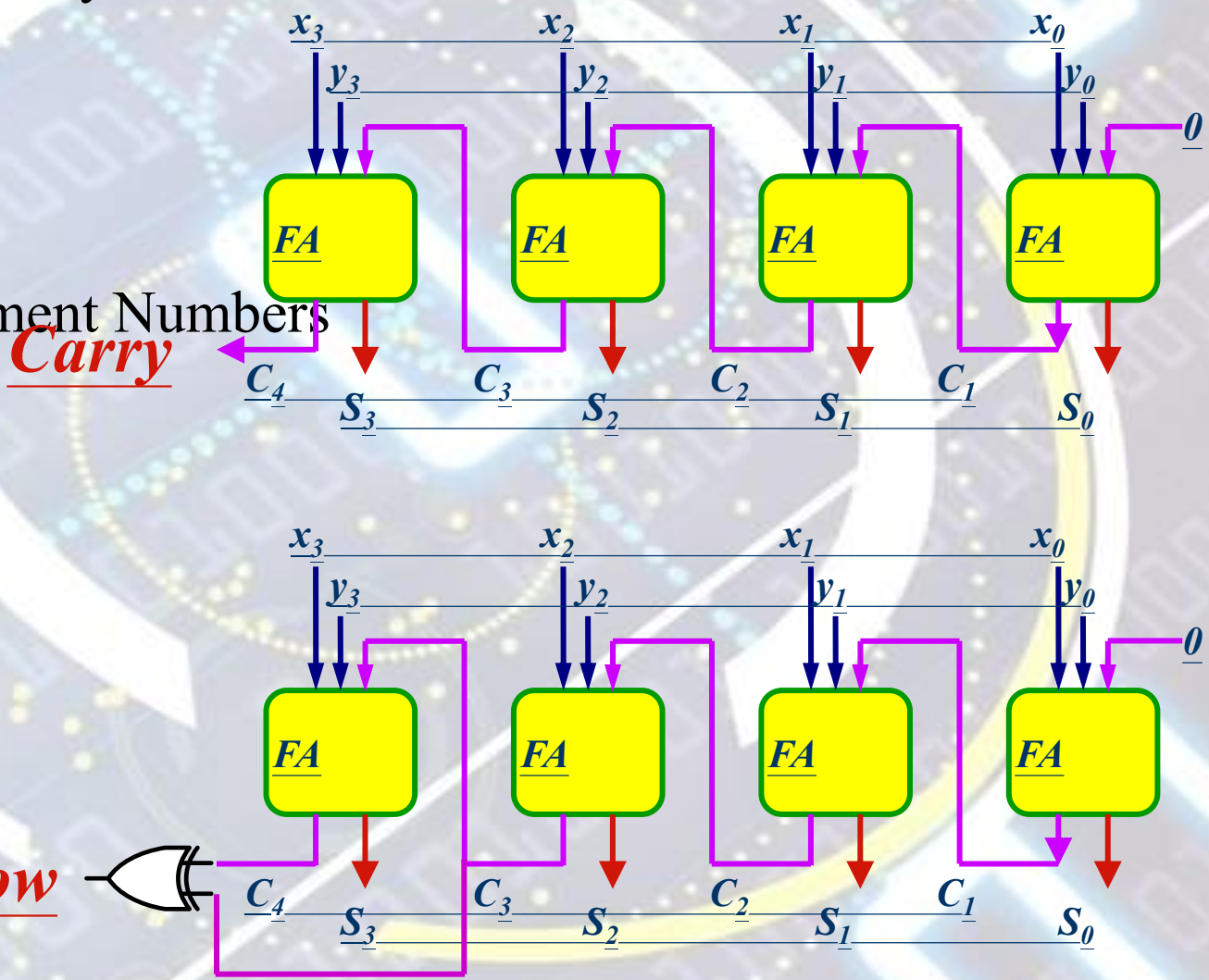
Binary Adder/Subtractor

- *M*: Control Signal (Mode)
  - $M=0 \rightarrow F = x + y$
  - $M=1 \rightarrow F = x - y$



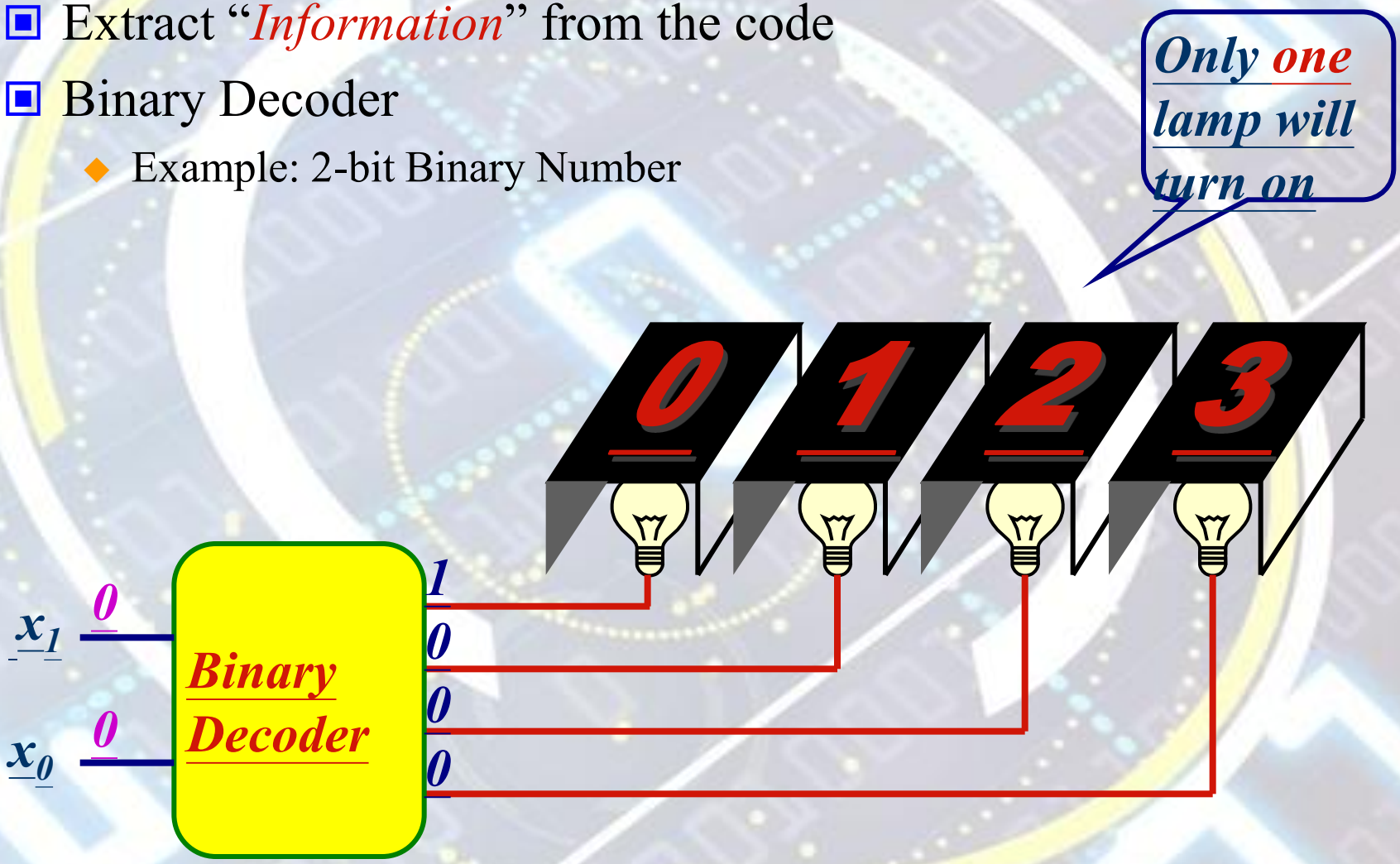
Overflow

- Unsigned Binary Numbers
- 2's Complement Numbers



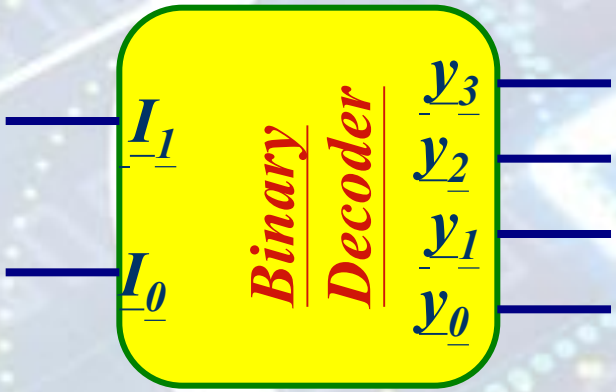
Decoders

- Extract "*Information*" from the code
- Binary Decoder
  - Example: 2-bit Binary Number

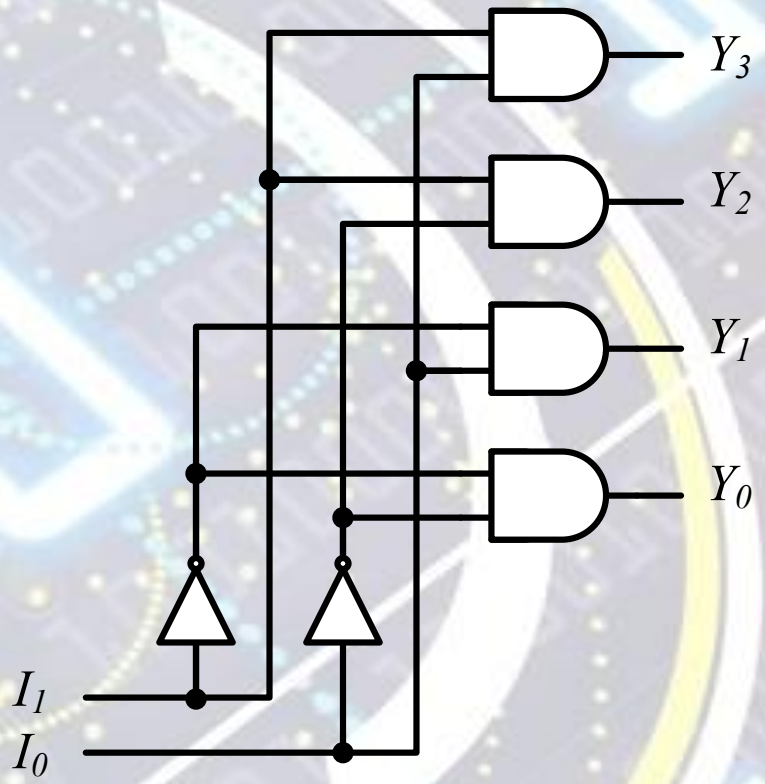


Decoders

- 2-to-4 Line Decoder



I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

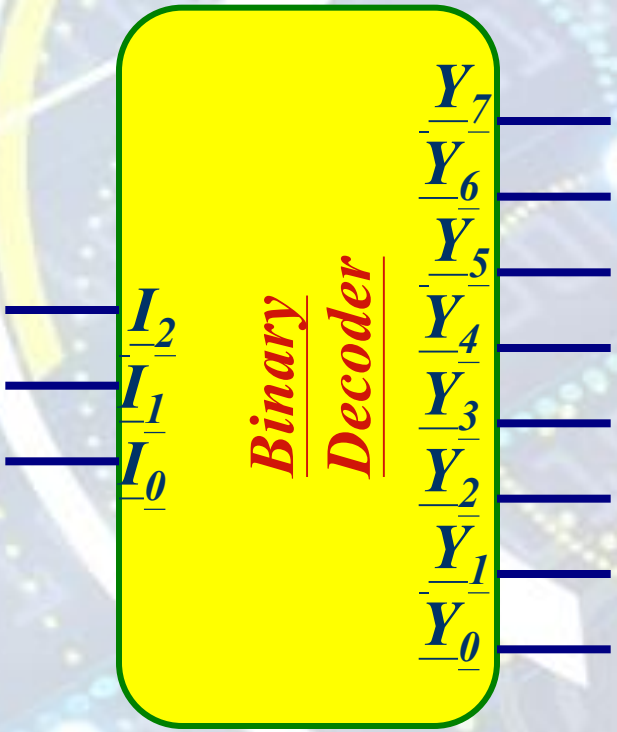


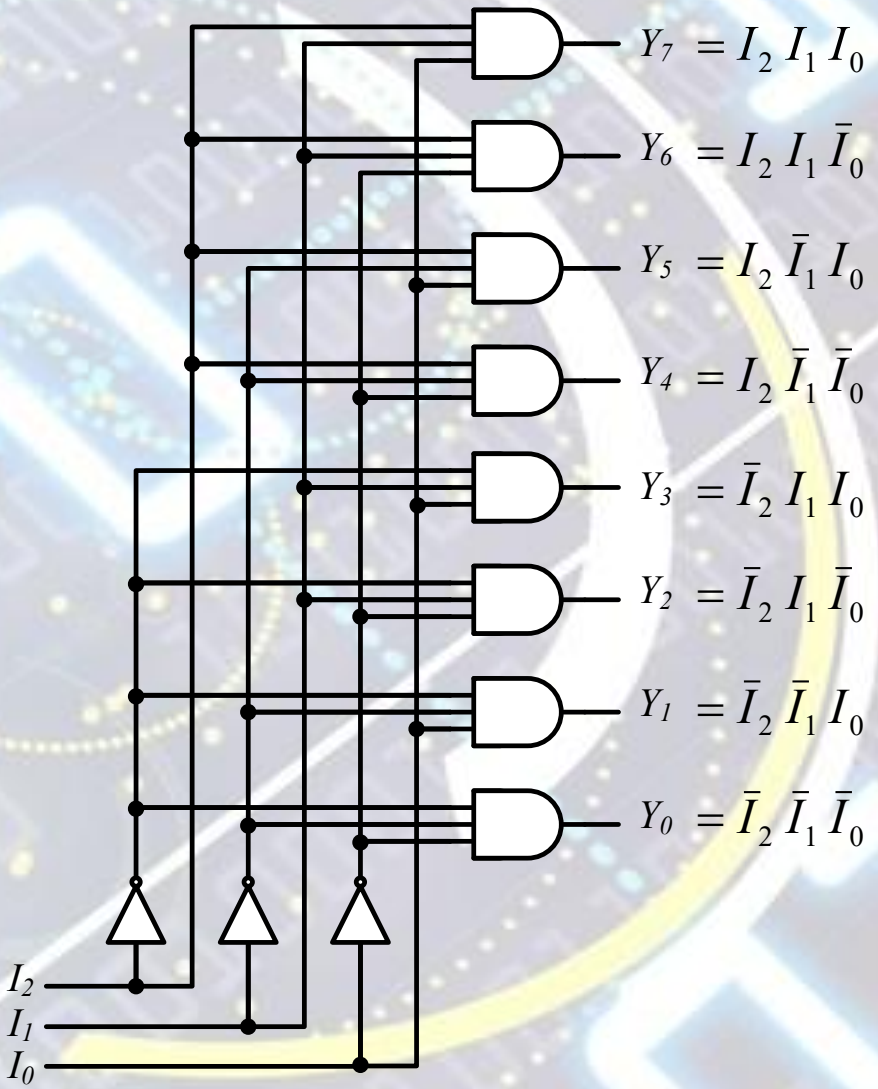
$$Y_3 = I_1 I_0 \qquad Y_2 = I_1 \bar{I}_0$$

$$Y_1 = \bar{I}_1 I_0 \qquad Y_0 = \bar{I}_1 \bar{I}_0$$

Decoders

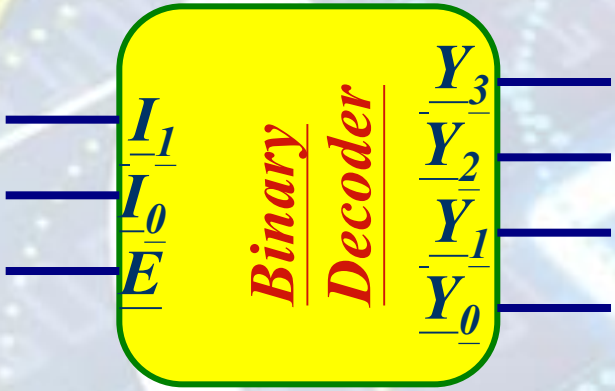
- 3-to-8 Line Decoder



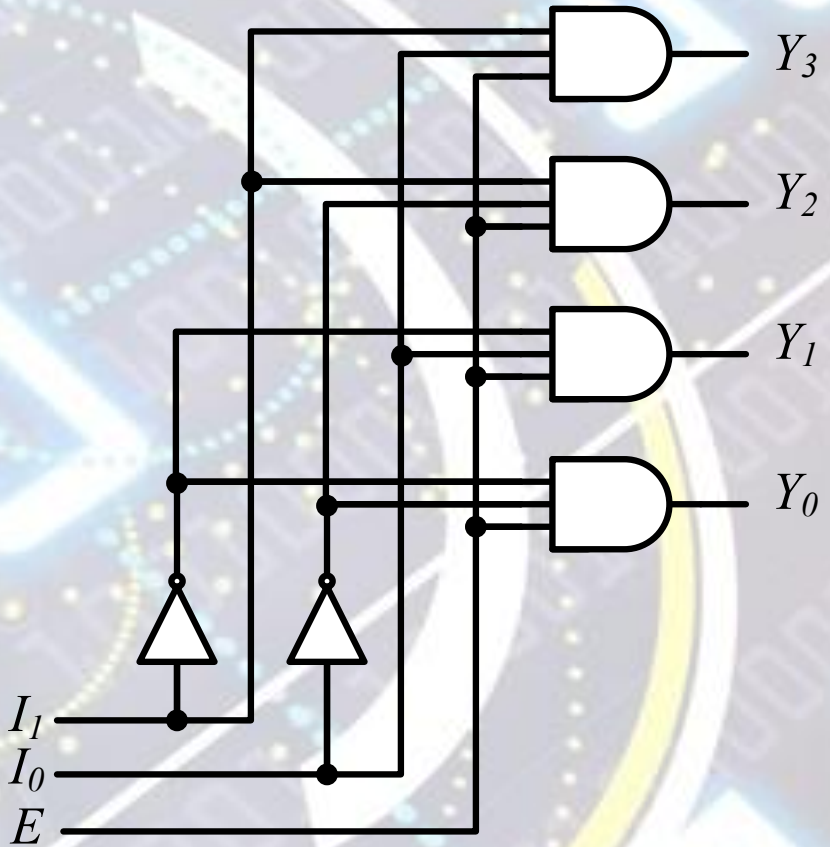


Decoders

- "*Enable*" Control



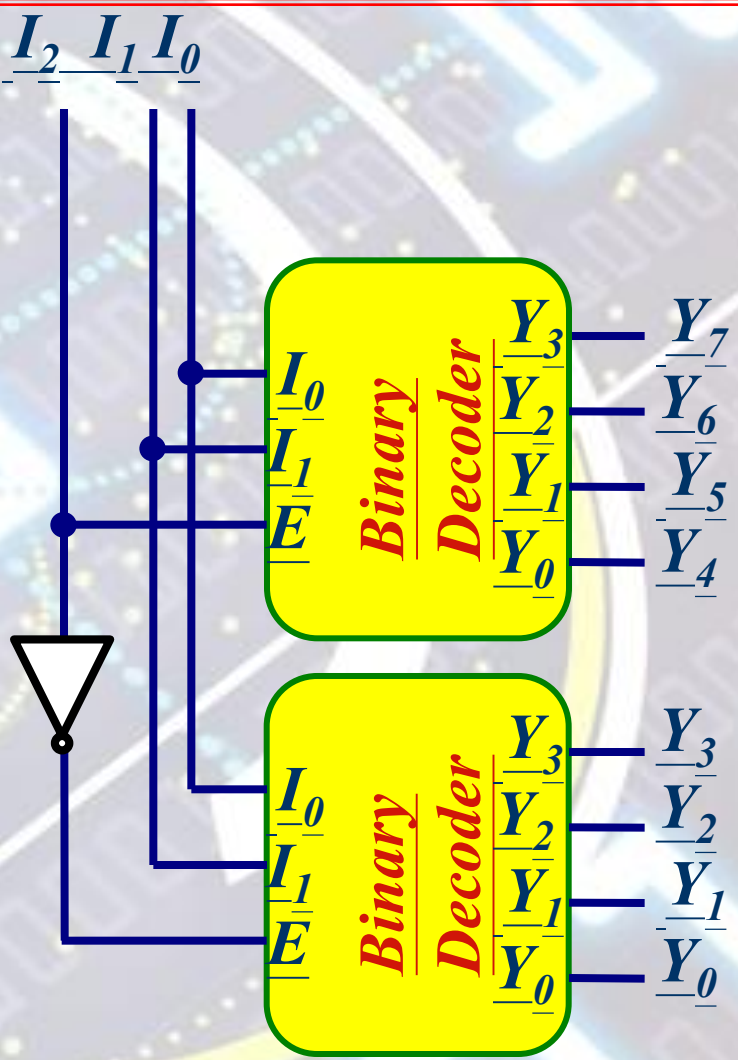
E	I_1 I_0	Y_3 Y_2 Y_1 Y_0
0	x x	0 0 0 0
1	0 0	0 0 0 1
1	0 1	0 0 1 0
1	1 0	0 1 0 0
1	1 1	1 0 0 0



Decoders

- Expansion

I_2	I_1	I_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

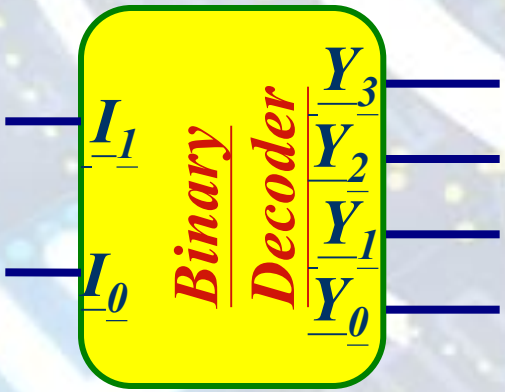


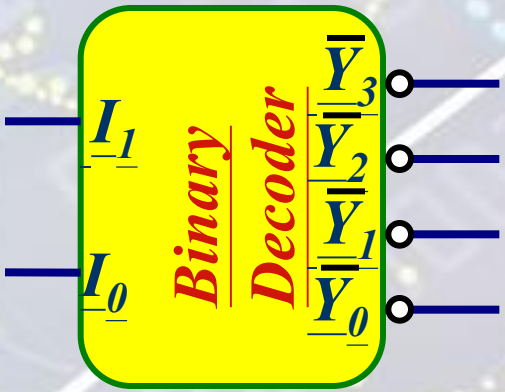
Decoders

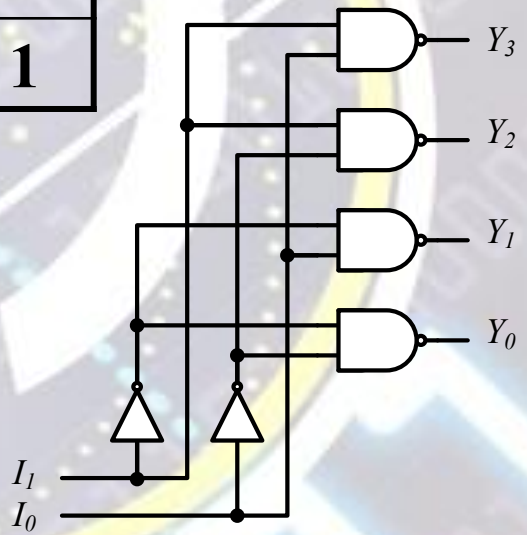
- Active-High / Active-Low

I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1







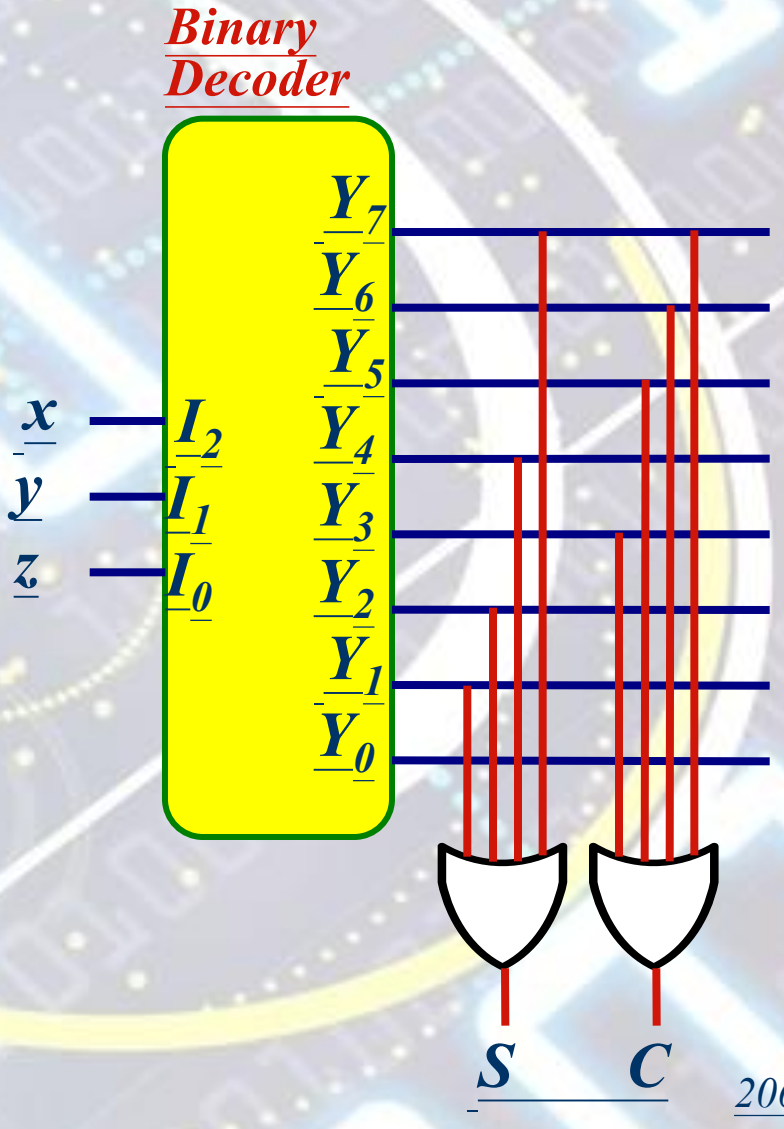
Implementation Using Decoders

- Each output is a minterm
- All minterms are produced
- Sum the required minterms

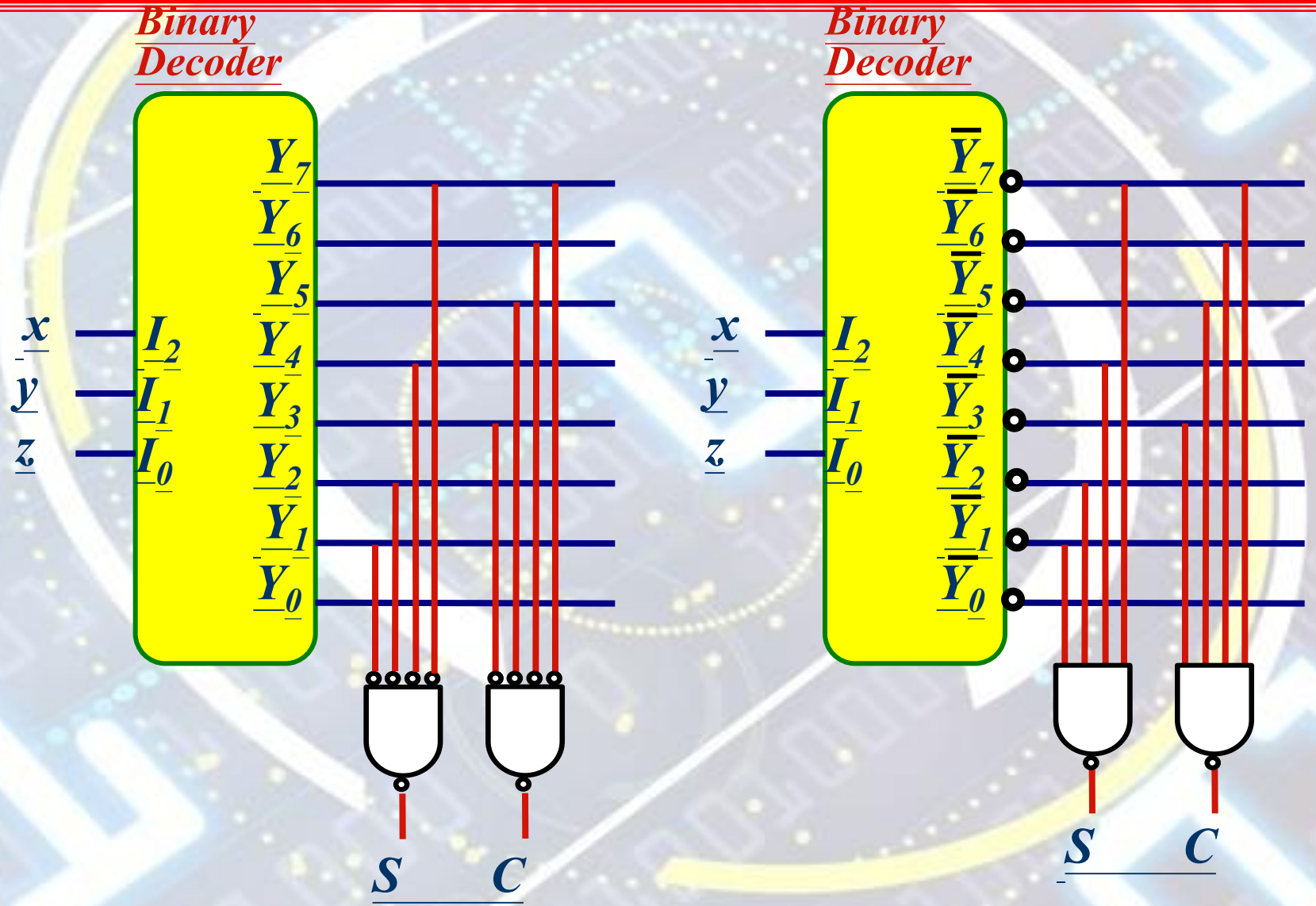
Example: Full Adder

$S(x, y, z) = \sum(1, 2, 4, 7)$

$C(x, y, z) = \sum(3, 5, 6, 7)$

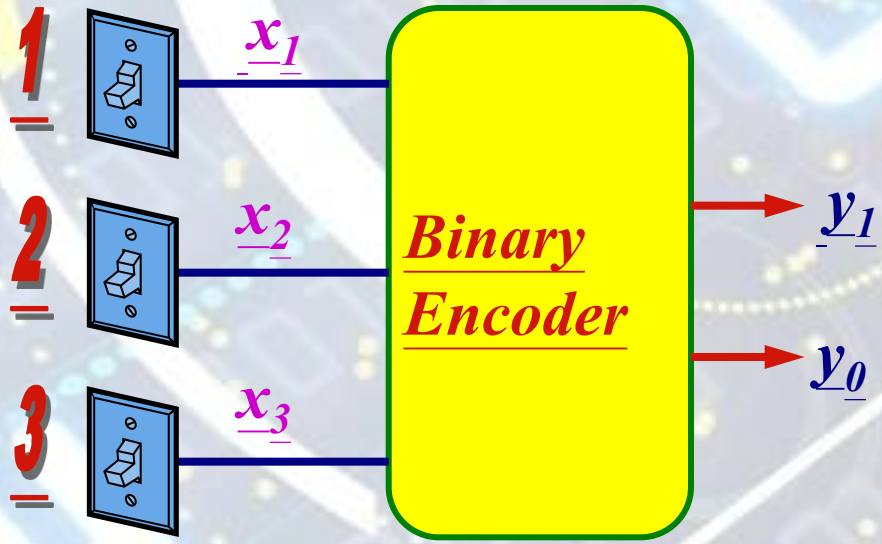


Implementation Using Decoders



Encoders

- Put "*Information*" into code
- Binary Encoder
  - Example: 4-to-2 Binary Encoder



Only one switch should be activated at a time

x_3	x_2	x_1	y_1	y_0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
1	0	0	1	1

Encoders

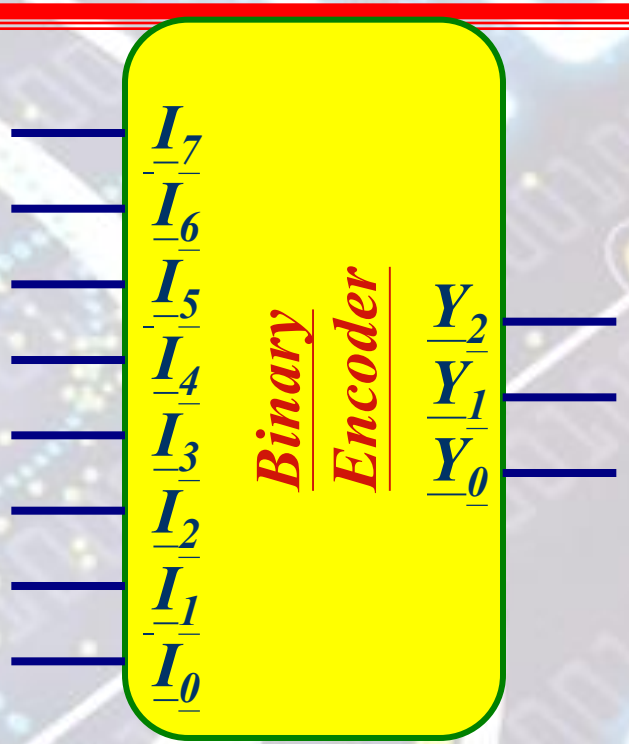
- Octal-to-Binary Encoder (8-to-3)

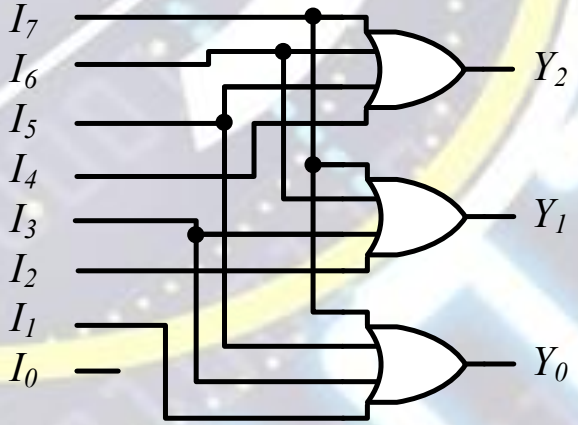
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$$Y_2 = I_7 + I_6 + I_5 + I_4$$

$$Y_1 = I_7 + I_6 + I_3 + I_2$$

$$Y_0 = I_7 + I_5 + I_3 + I_1$$

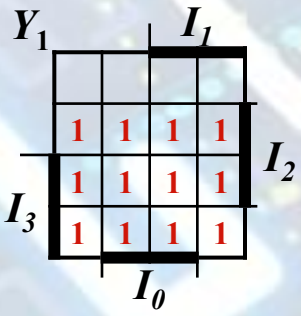




Priority Encoders

- 4-Input Priority Encoder

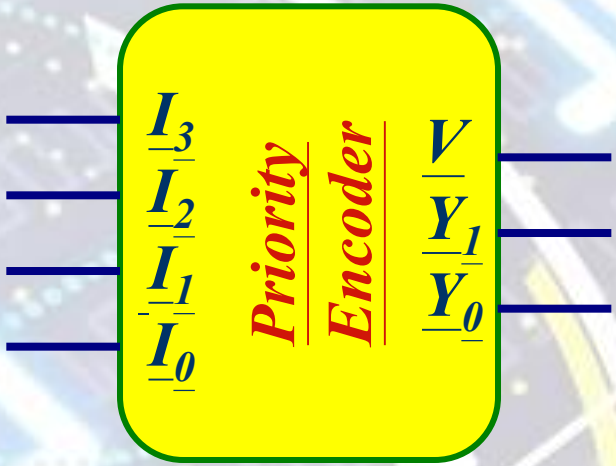
I_3	I_2	I_1	I_0	Y_1	Y_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

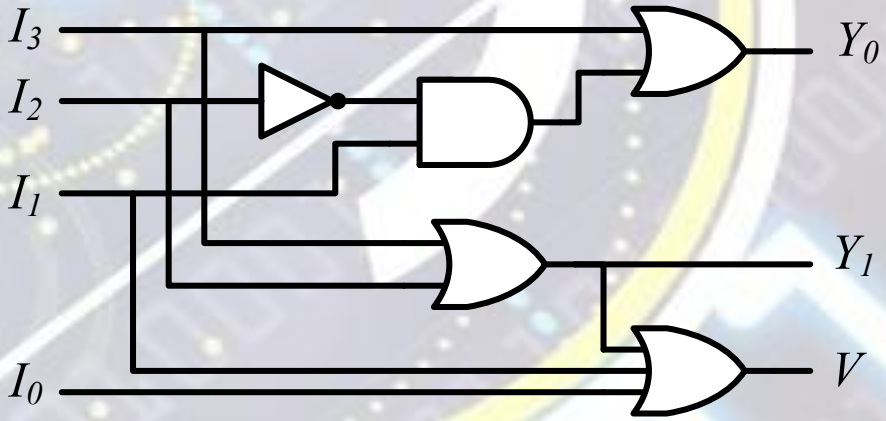


$$Y_1 = I_3 + I_2$$

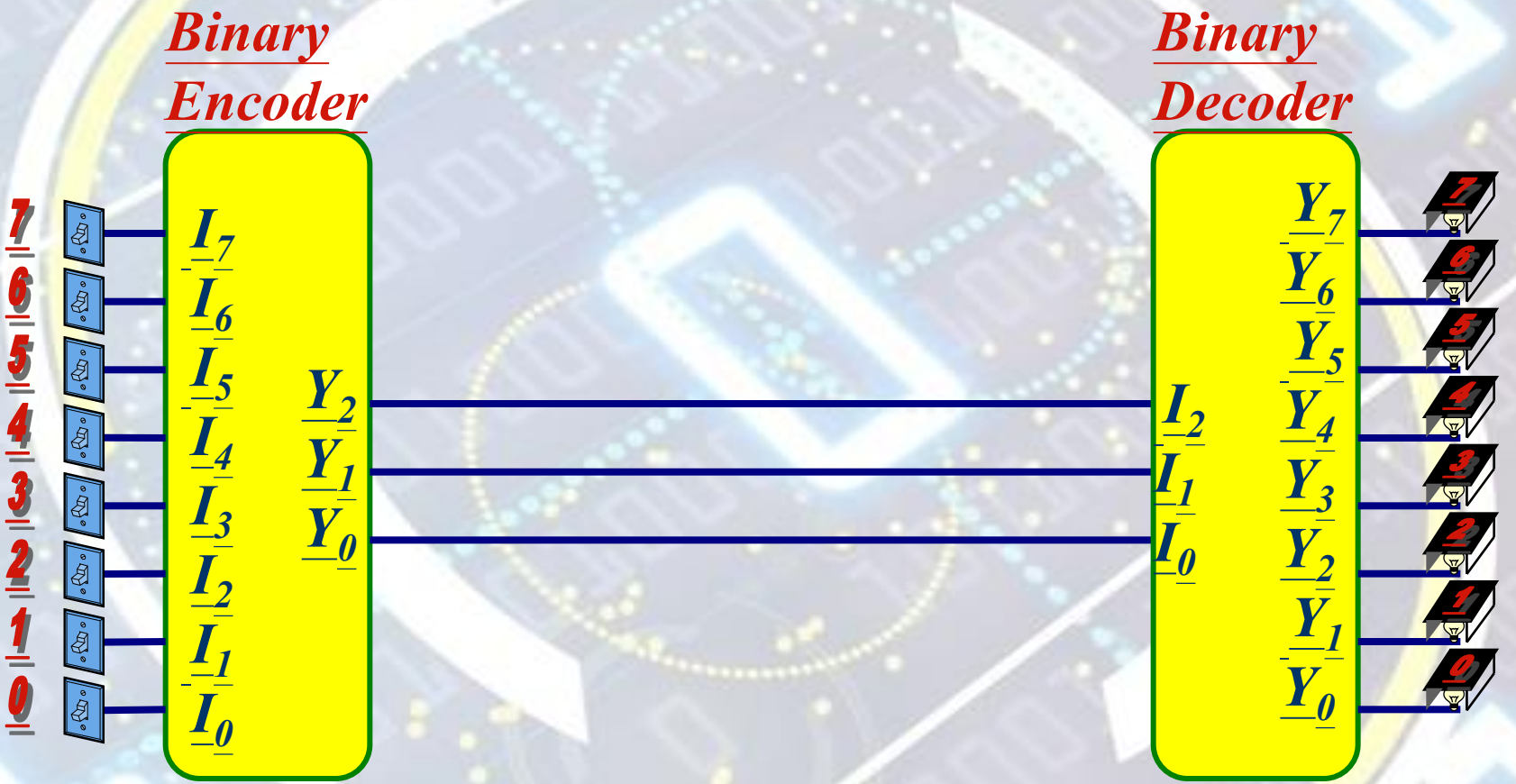
$$Y_0 = I_3 + \bar{I}_2 I_1$$

$$V = I_3 + I_2 + I_1 + I_0$$





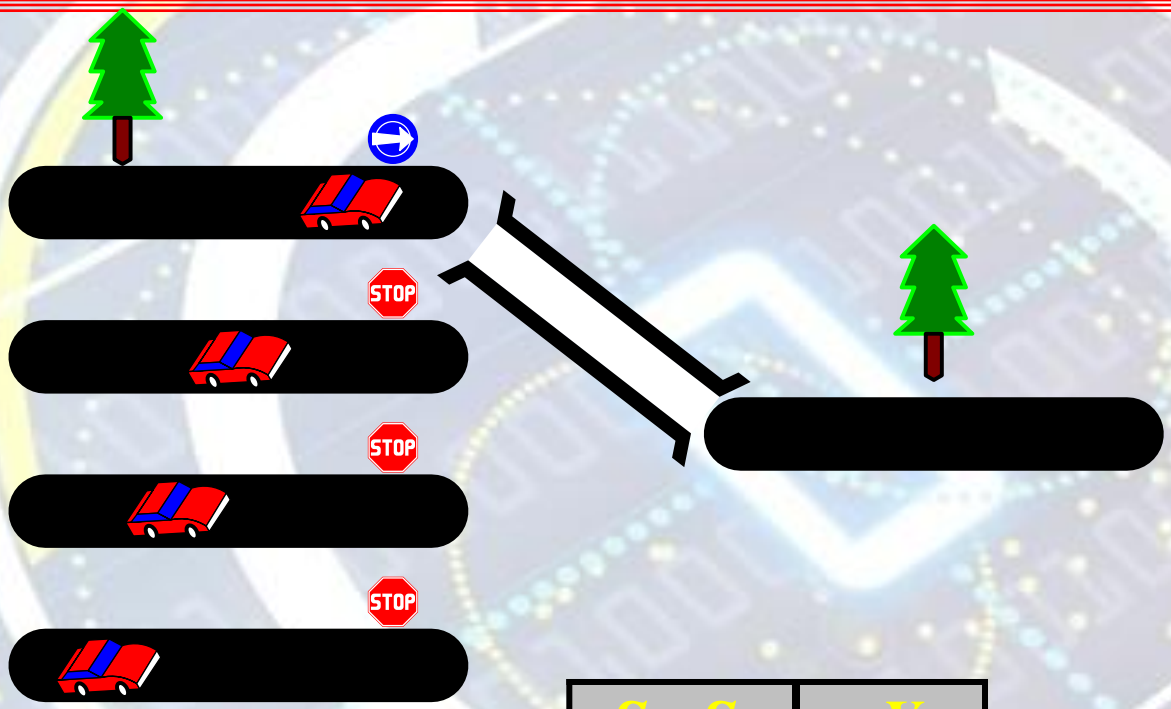
Encoder / Decoder Pairs



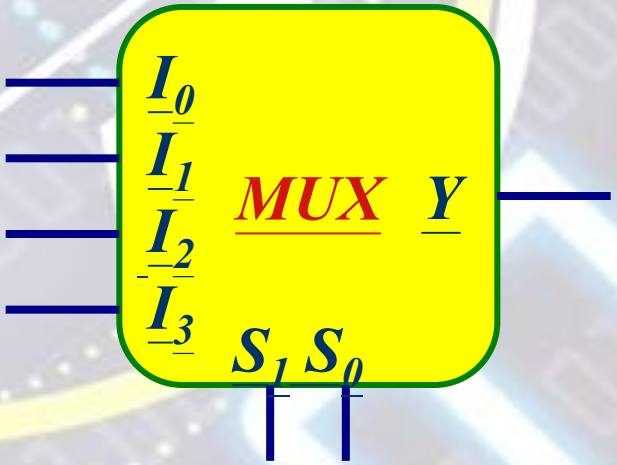
Week -13

Page(213-227)

Multiplexers

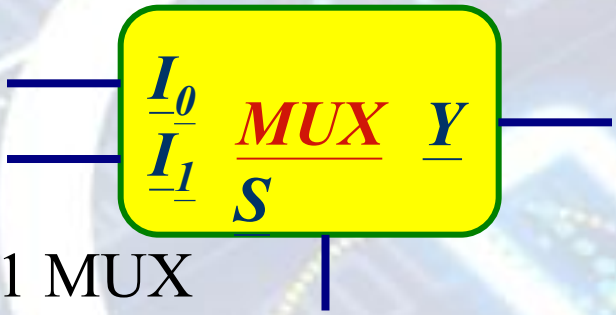


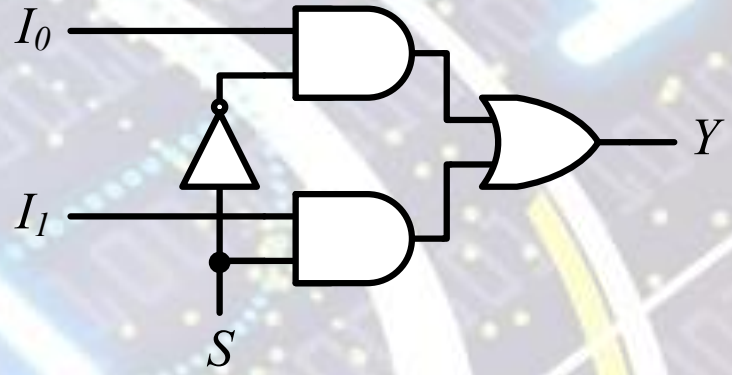
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



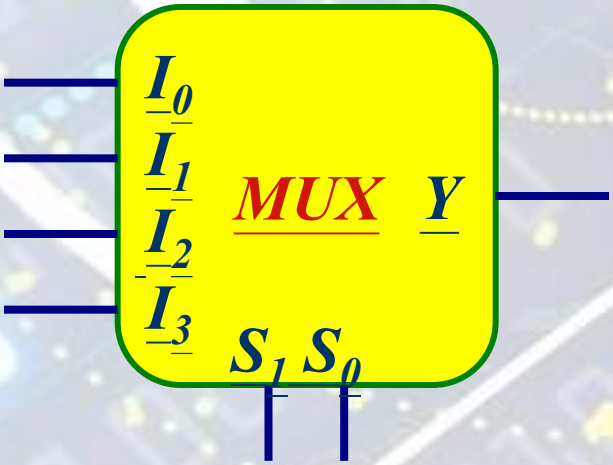
Multiplexers

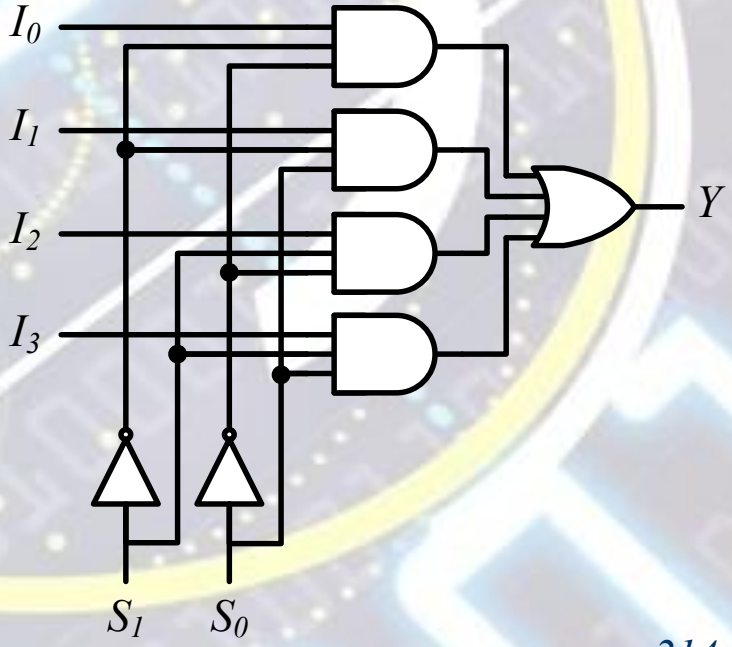
- 2-to-1 MUX





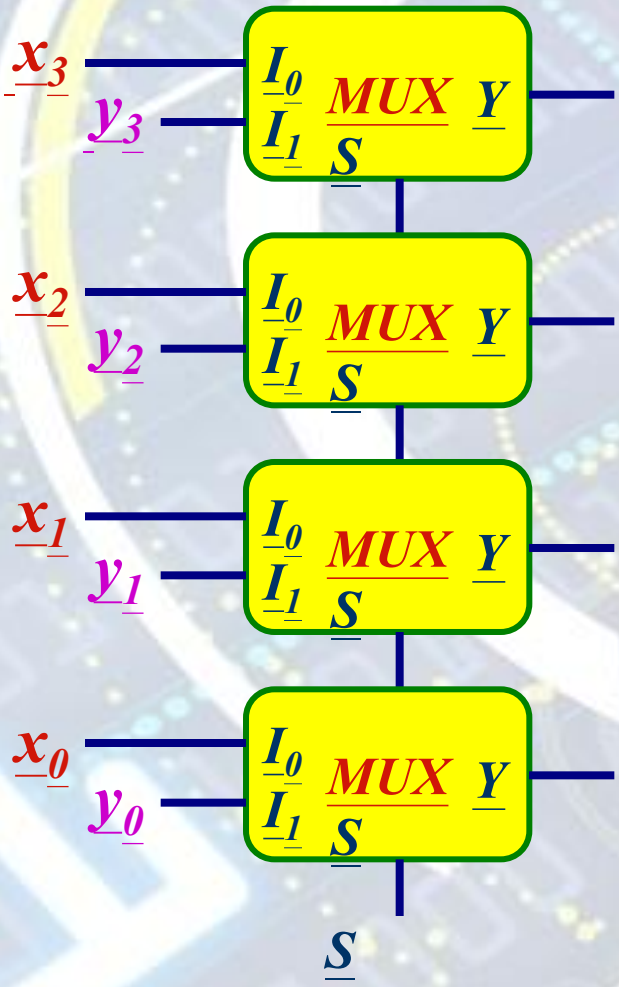
- 4-to-1 MUX

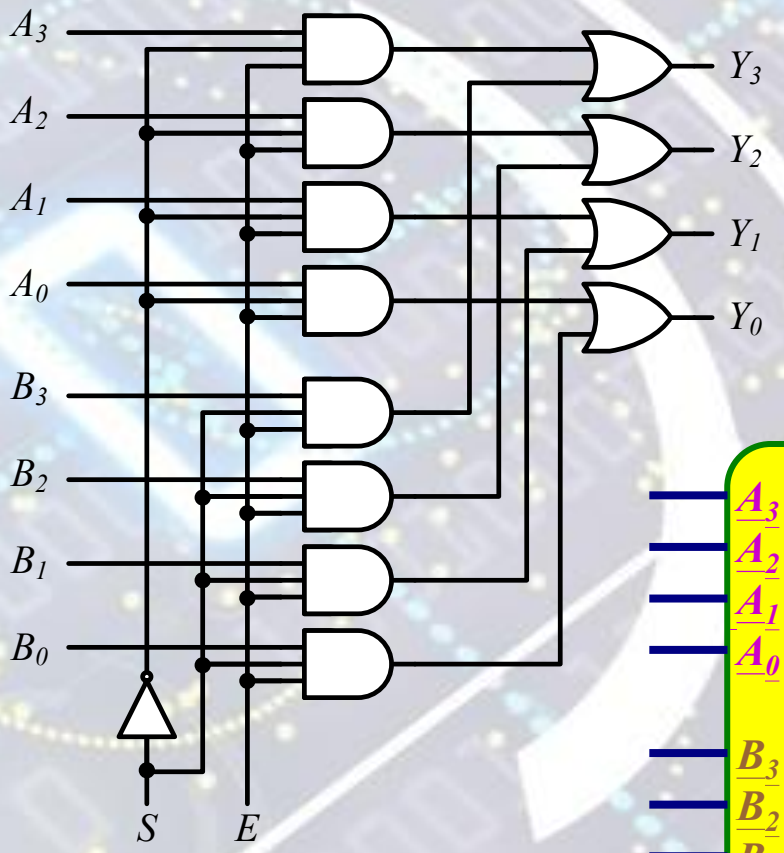




Multiplexers

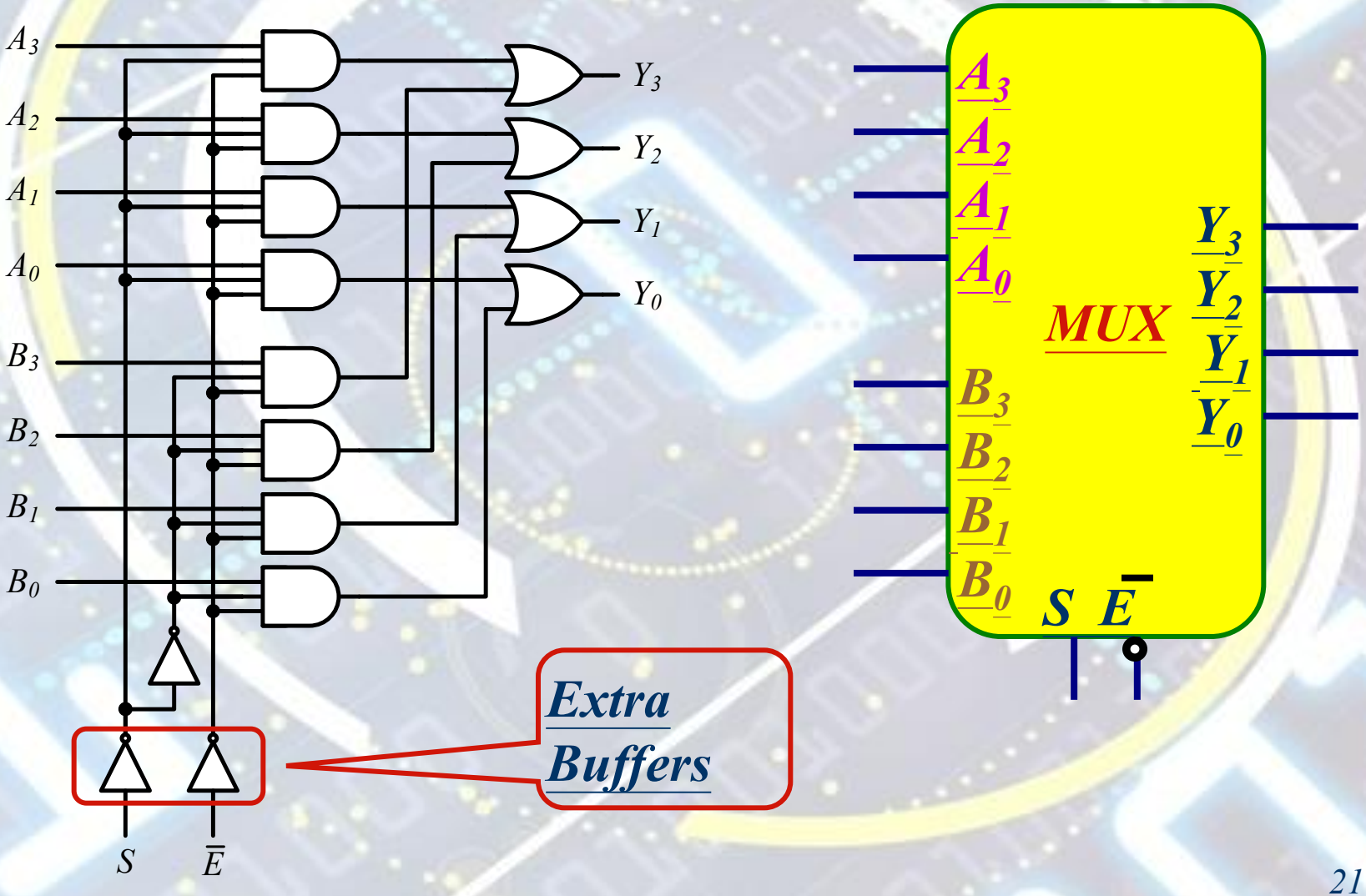
- Quad 2-to-1 MUX





Multiplexers

- Quad 2-to-1 MUX

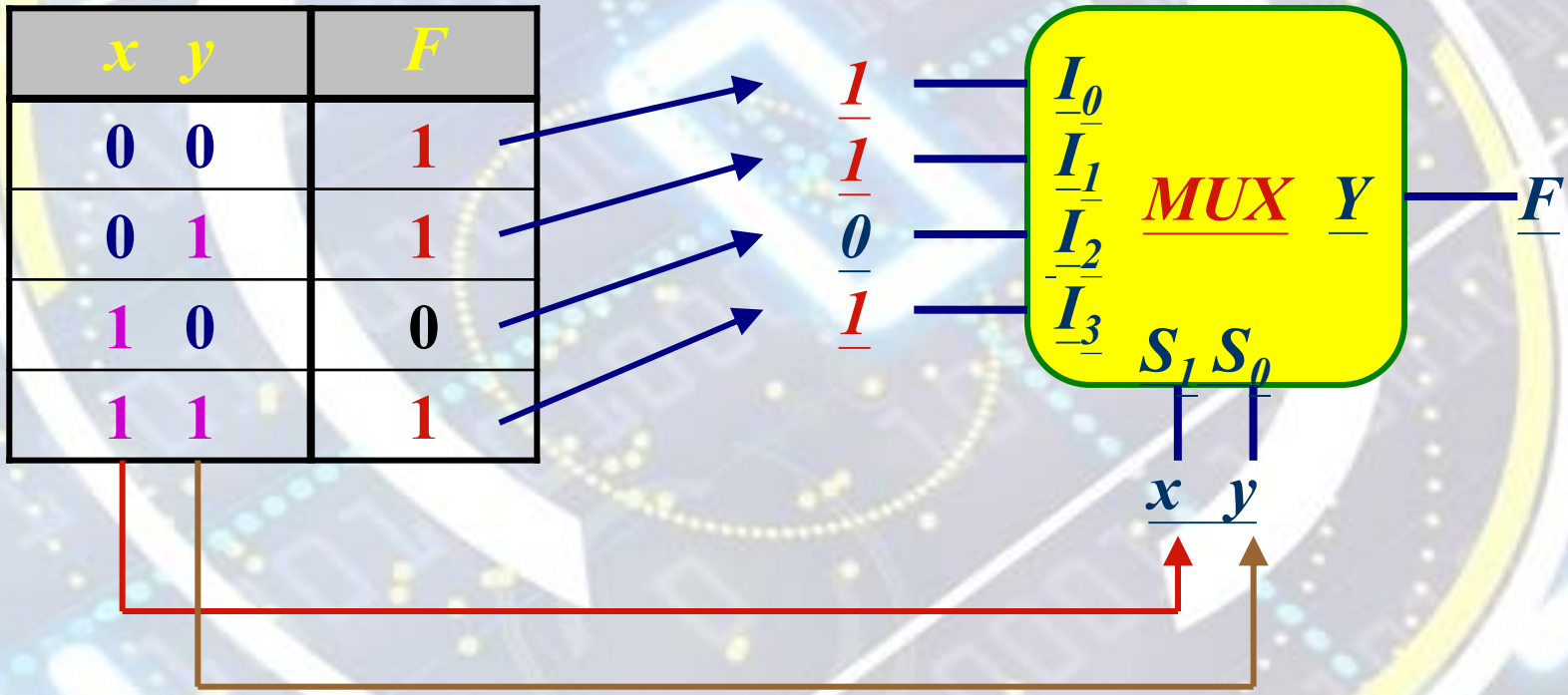


Implementation Using Multiplexers

- Example

  $F(x, y) = \sum(0, 1, 3)$

x	y	F
0	0	1
0	1	1
1	0	0
1	1	1

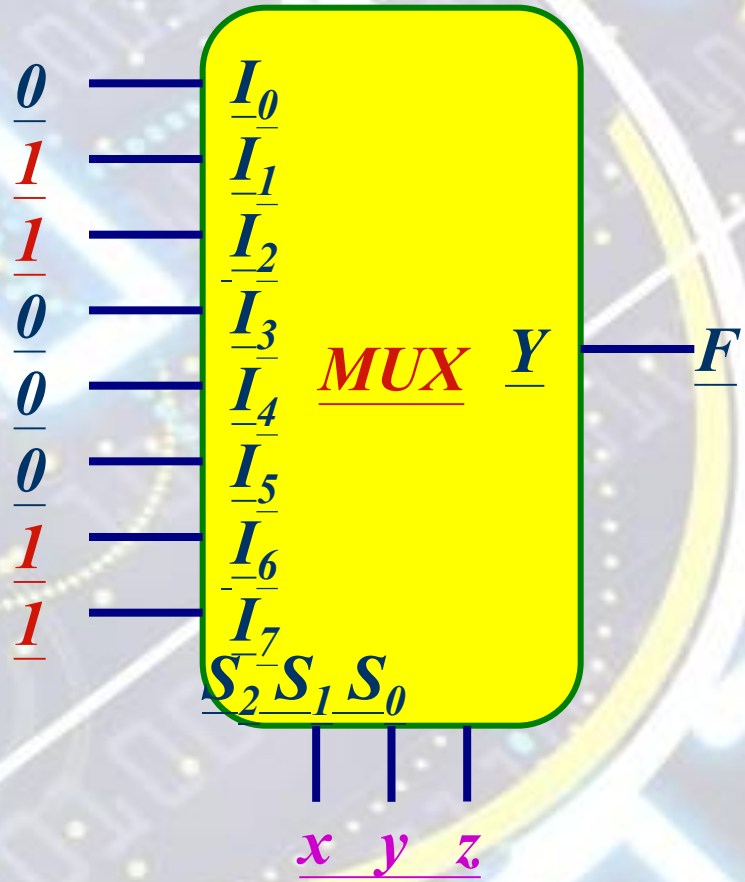


Implementation Using Multiplexers

- Example

  $F(x, y, z) = \sum(1, 2, 6, 7)$

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

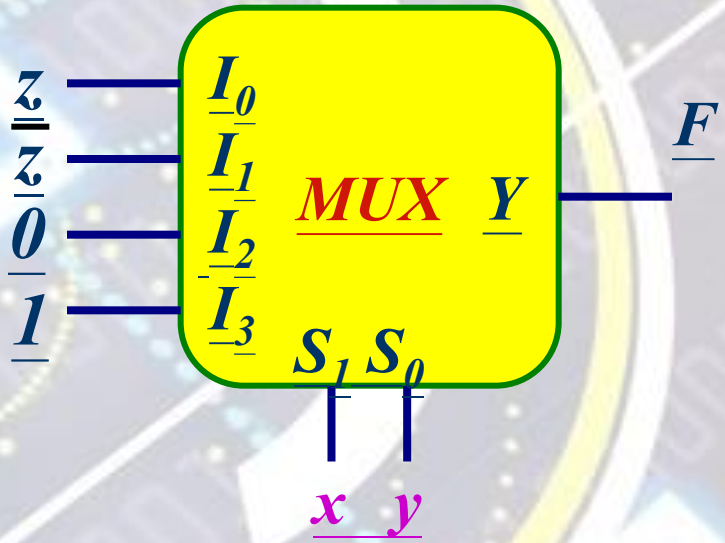


Implementation Using Multiplexers

- Example

  $F(x, y, z) = \sum(1, 2, 6, 7)$

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = \bar{z}$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

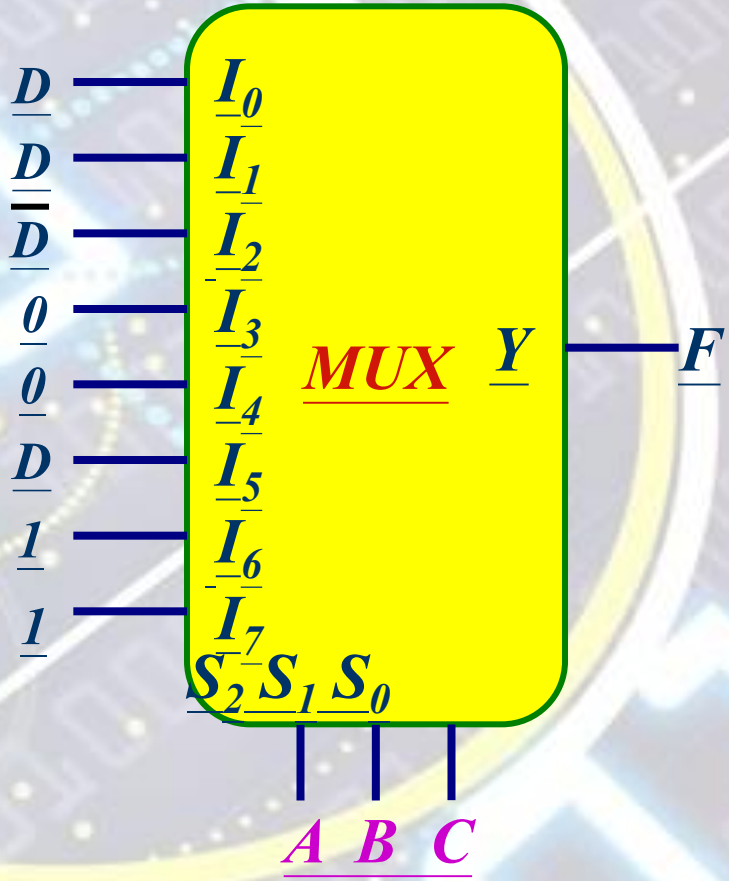


Implementation Using Multiplexers

- Example

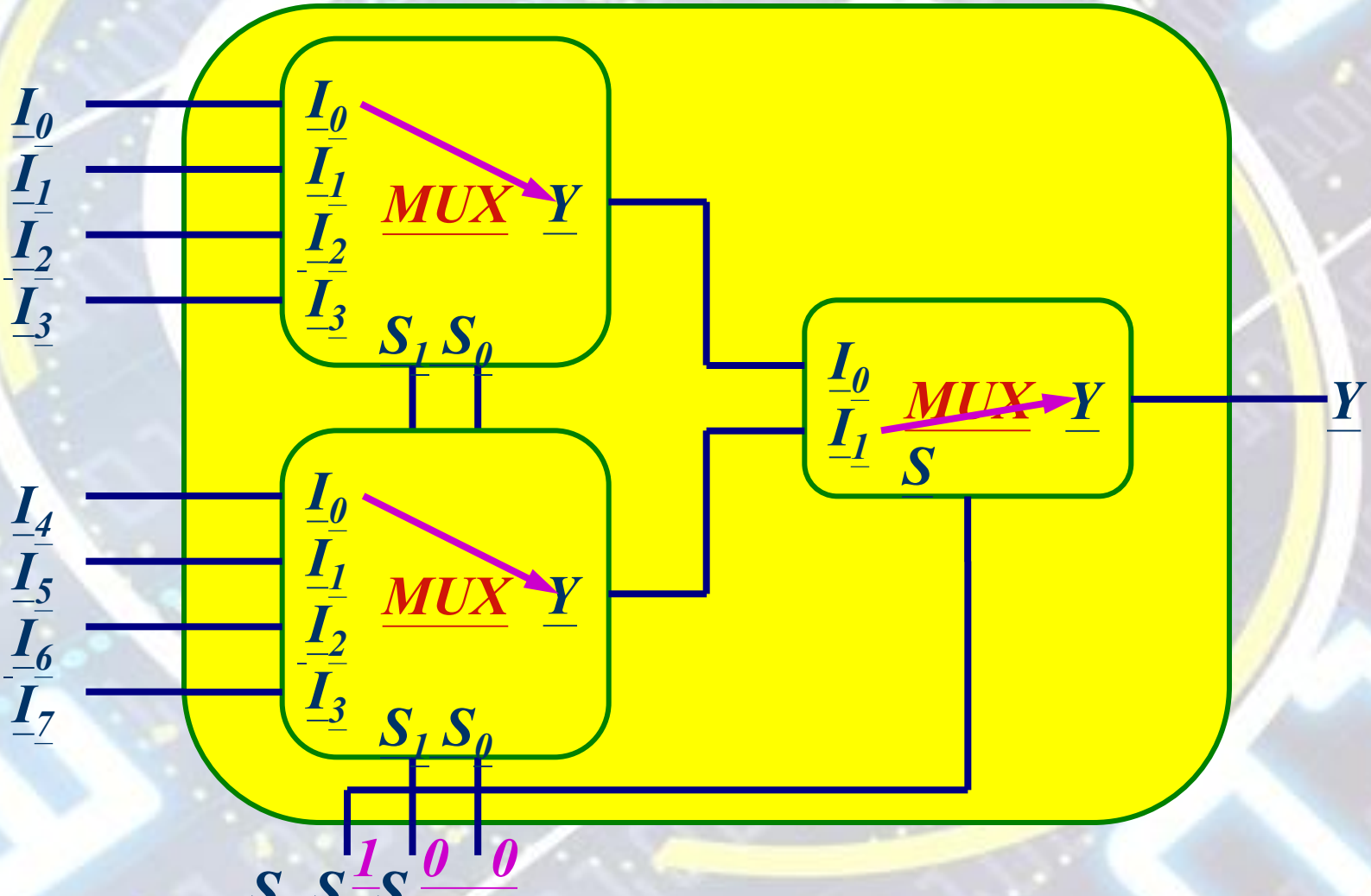
$F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = \overline{D}$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

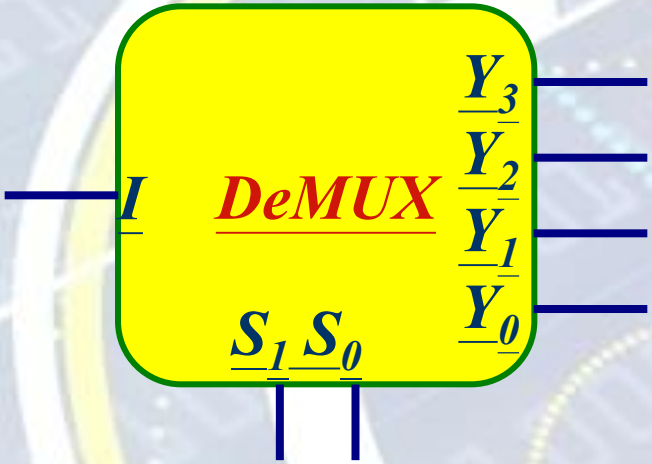


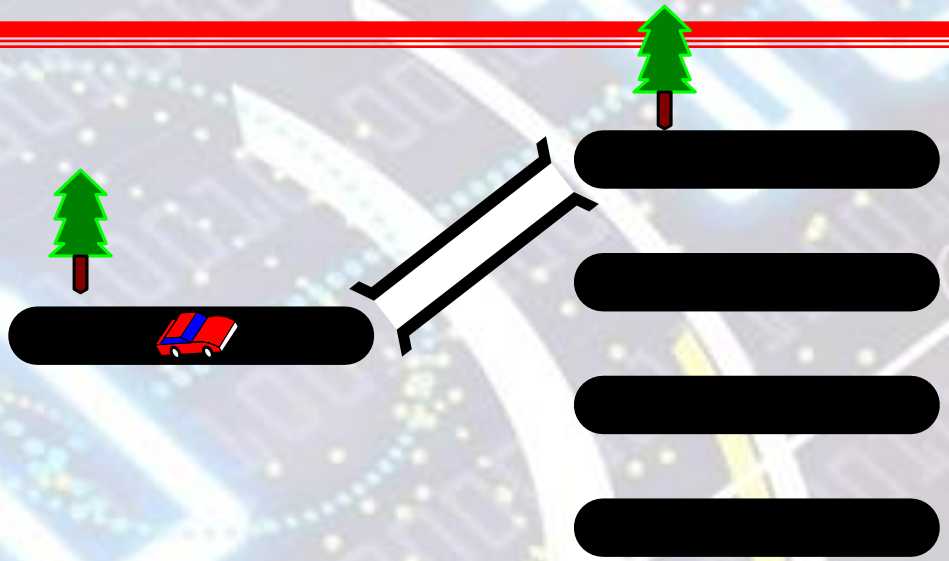
Multiplexer Expansion

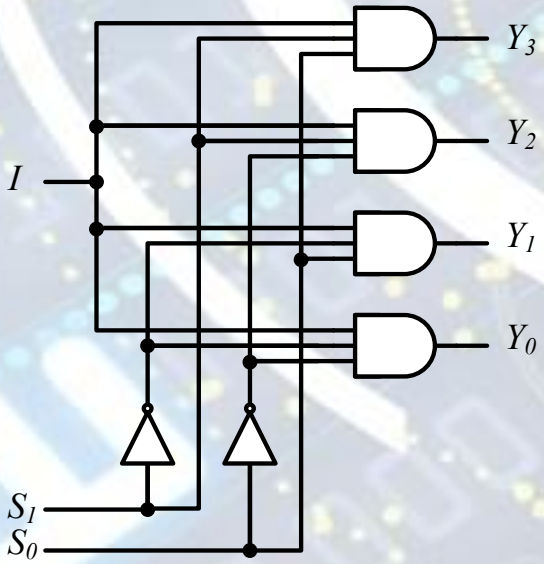
- 8-to-1 MUX using Dual 4-to-1 MUX



DeMultiplexers

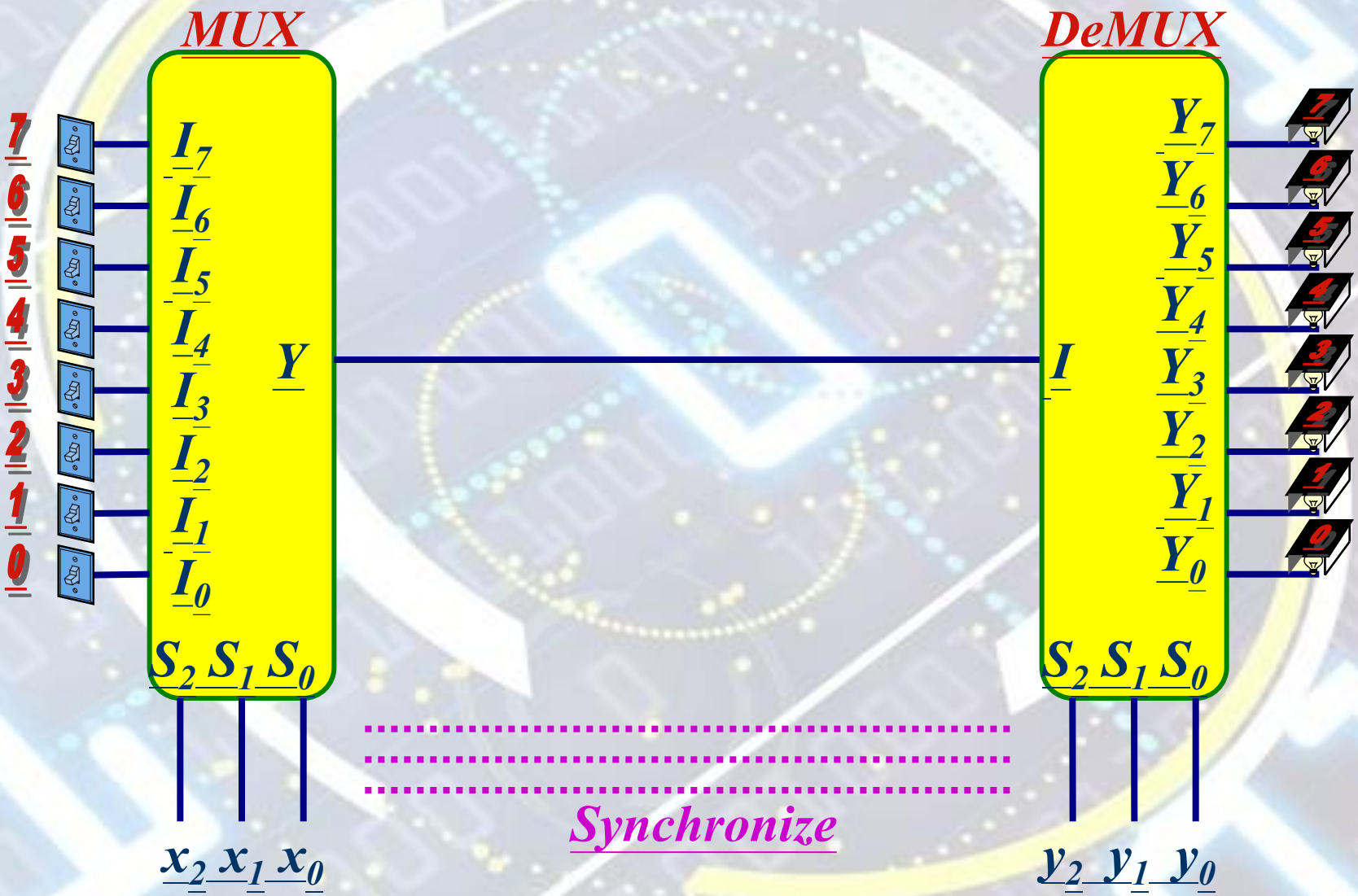




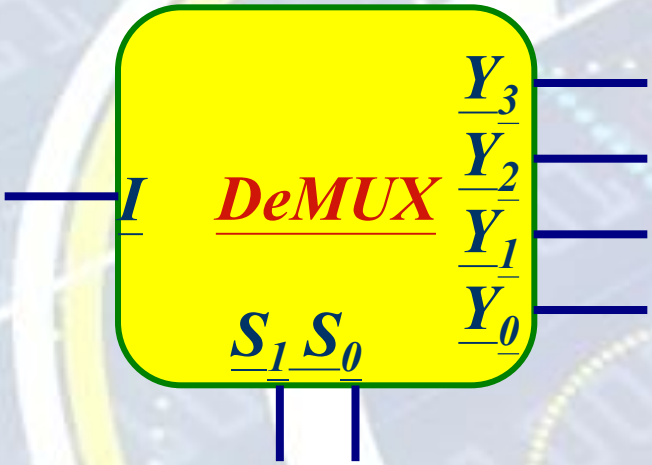


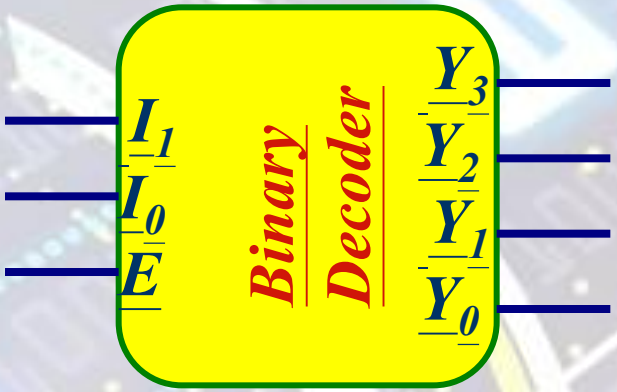
S_1 S_0	Y_3	Y_2	Y_1	Y_0
0 0	0	0	0	I
0 1	0	0	I	0
1 0	0	I	0	0
1 1	I	0	0	0

Multiplexer / DeMultiplexer Pairs



DeMultiplexers / Decoders



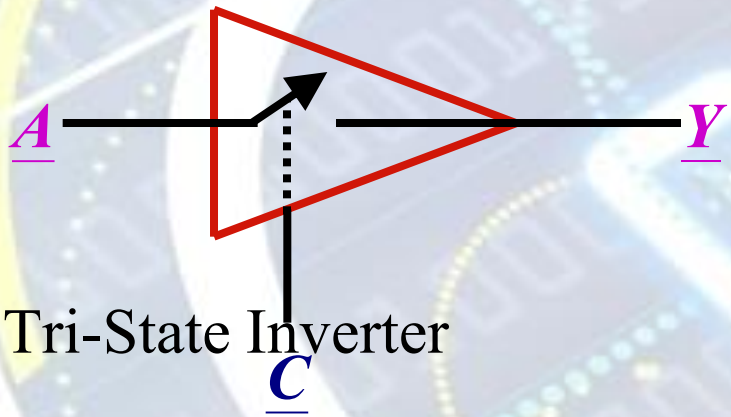


S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

E	I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

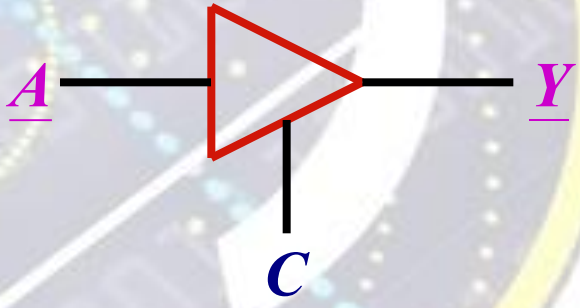
Three-State Gates

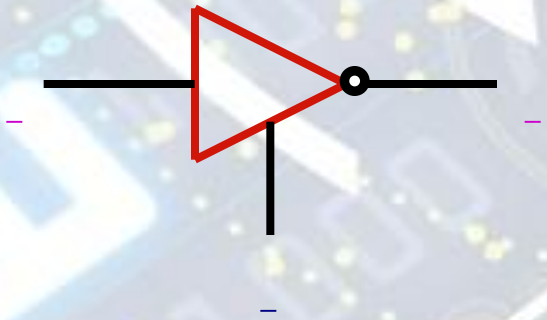
- Tri-State Buffer



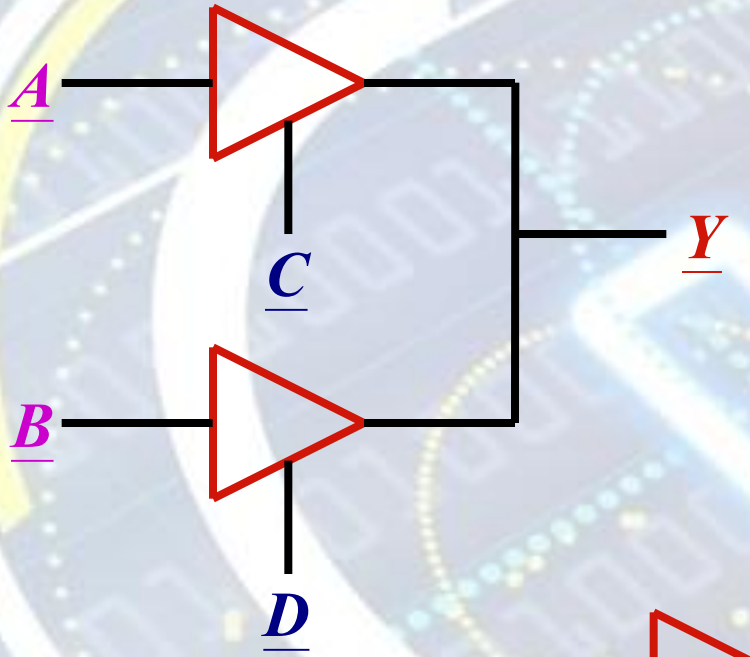
C	A	Y
0	x	Hi-Z
1	0	0
1	1	1

- Tri-State Inverter



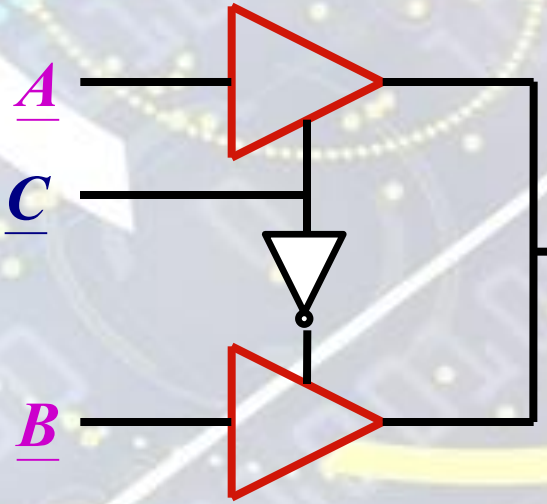


Three-State Gates



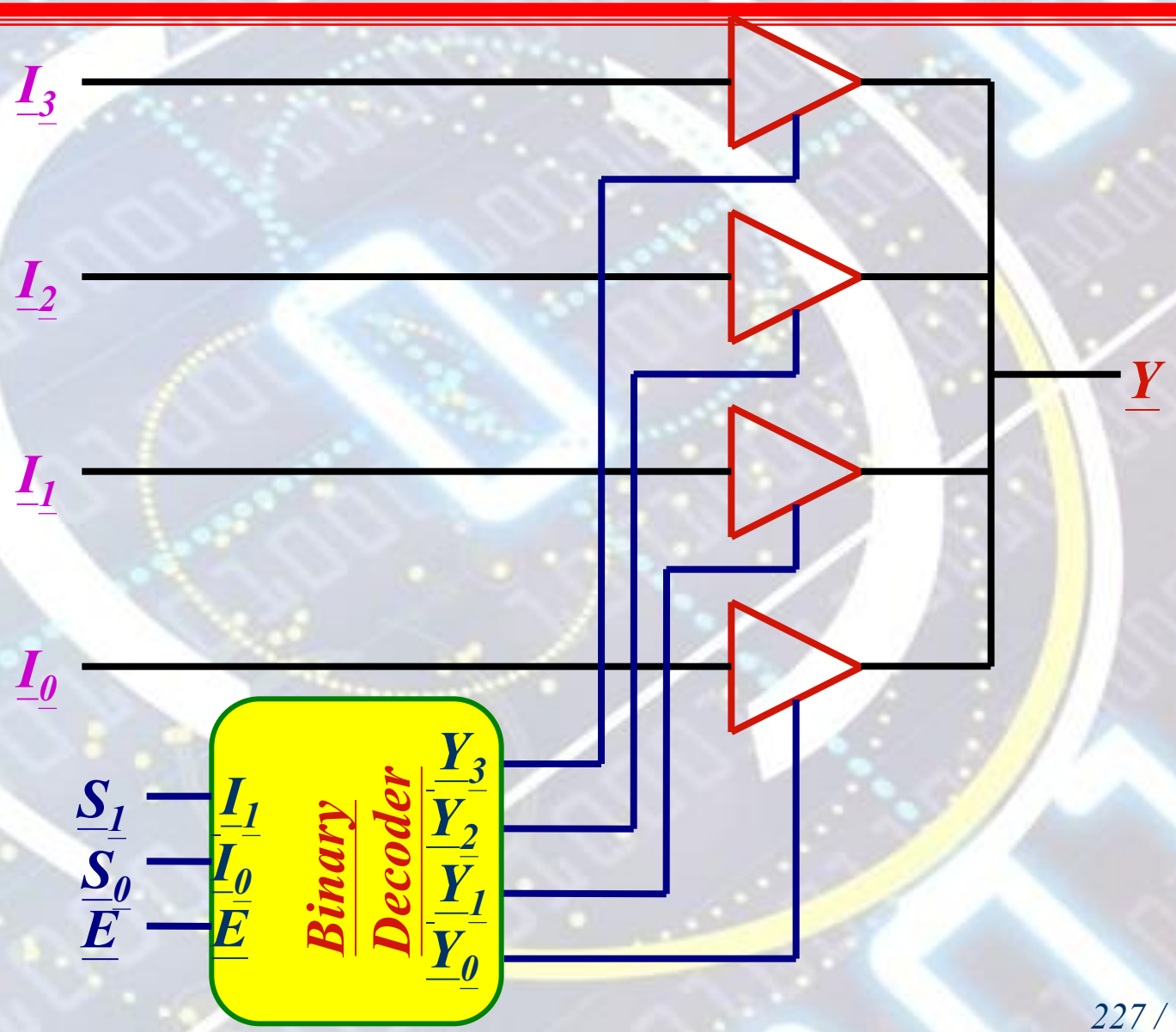
C	D	Y
0	0	Hi-Z
0	1	B
1	0	A
1	1	?

Not Allowed



$$Y = \begin{cases} A & \text{if } C = 1 \\ B & \text{if } C = 0 \end{cases}$$

Three-State Gates

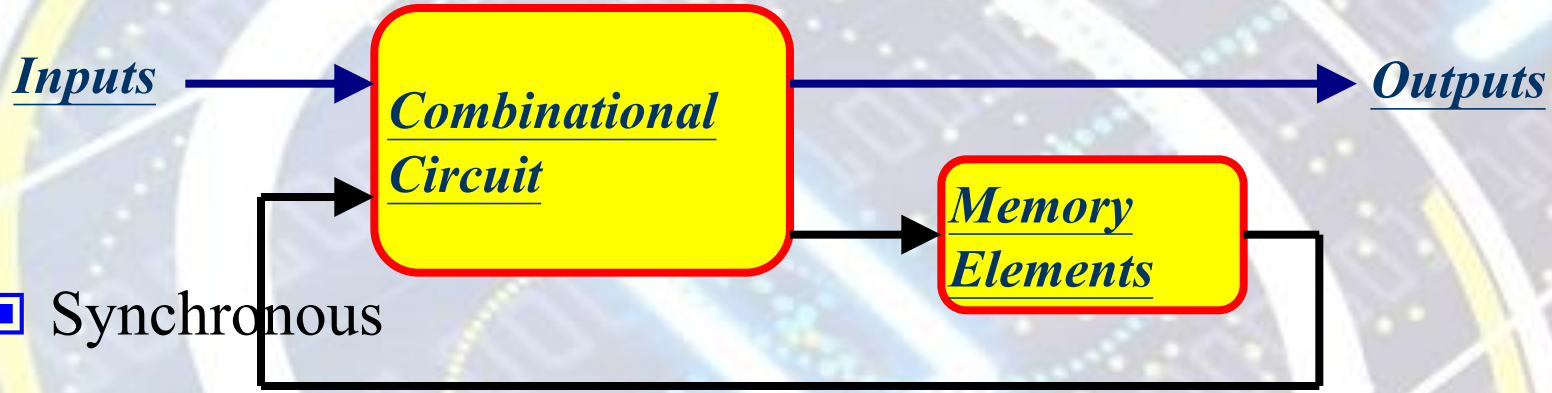


Week -14

Page(229-243)

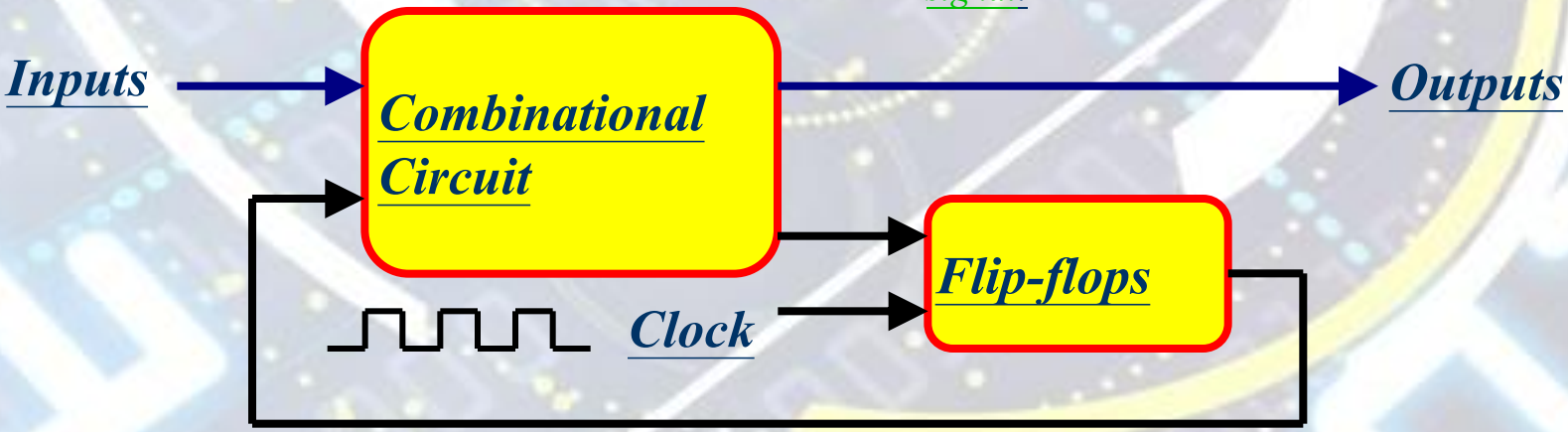
Sequential Circuits

- Asynchronous



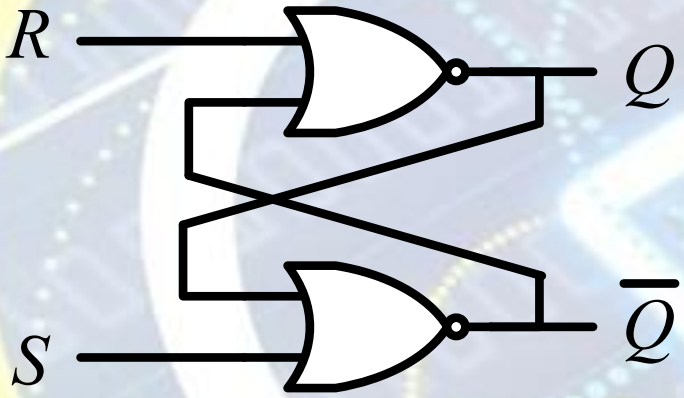
- Synchronous

A synchronous circuit is a digital circuit in which the parts are synchronized by a clock signal.

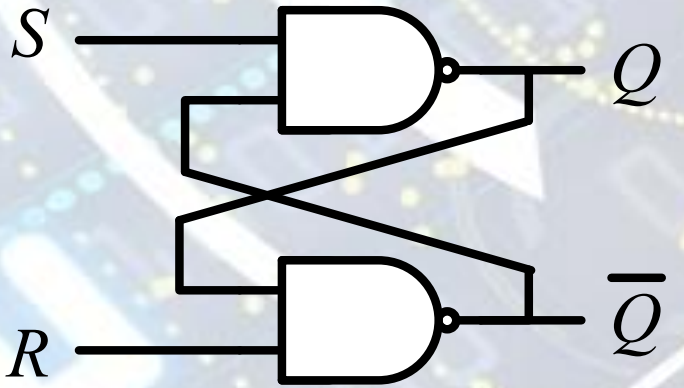


Latches

- *SR* Latch



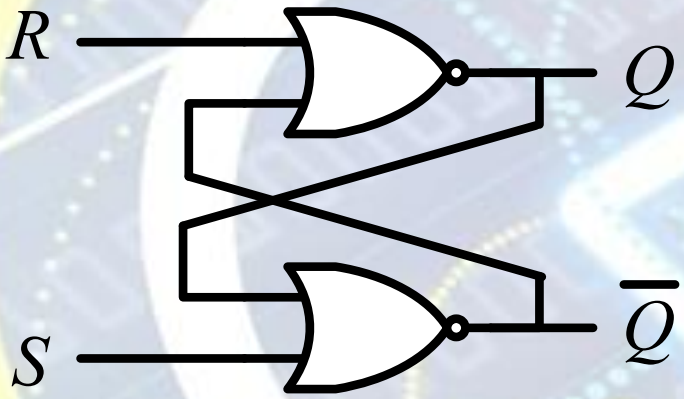
S R	Q	
0 0	Q_0	No change
0 1	0	Reset
1 0	1	Set
1 1	Q=Q'=0	Invalid



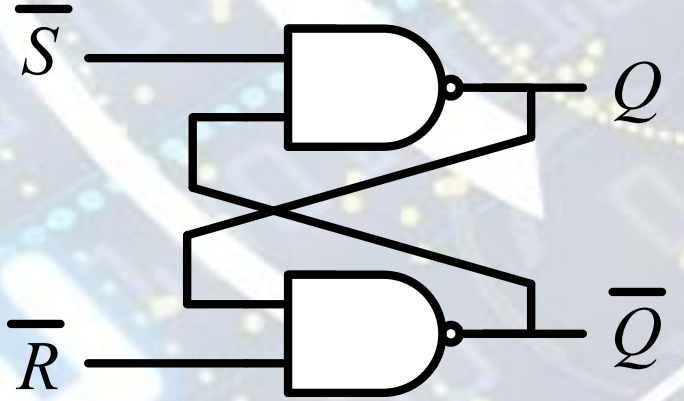
S R	Q	
0 0	Q=Q'=1	Invalid
0 1	1	Set
1 0	0	Reset
1 1	Q_0	No change

Latches

- *SR* Latch



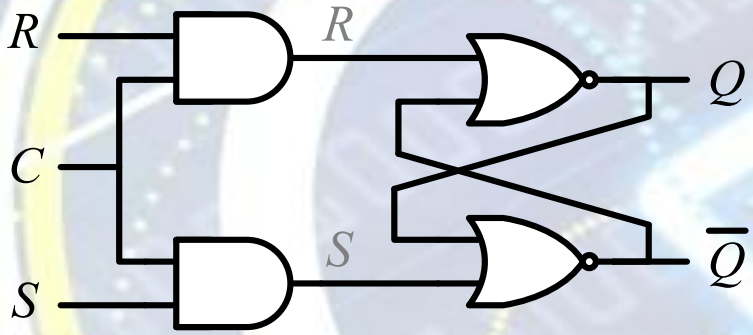
S R	Q	
0 0	Q_0	No change
0 1	0	Reset
1 0	1	Set
1 1	Q=Q'=0	Invalid

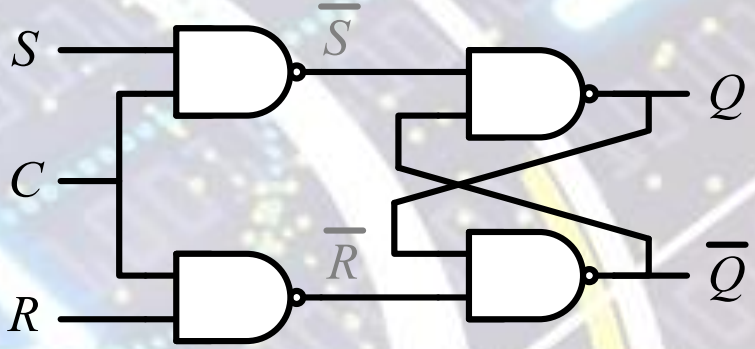


S' R'	Q	
0 0	Q=Q'=1	Invalid
0 1	1	Set
1 0	0	Reset
1 1	Q_0	No change

Controlled Latches

- *SR* Latch with Control Input

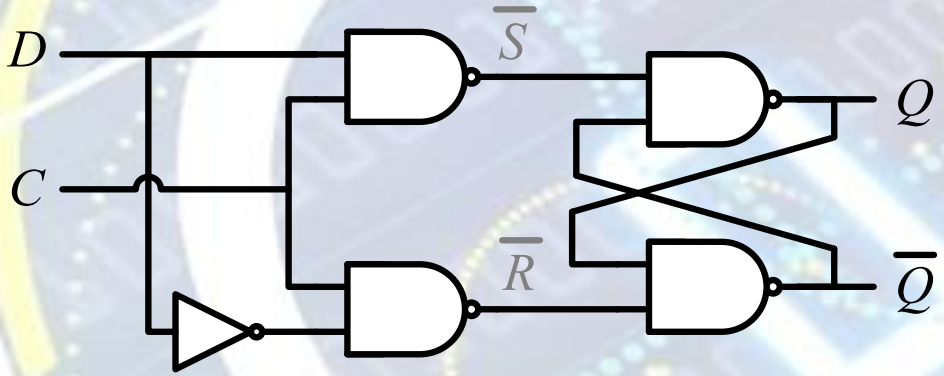




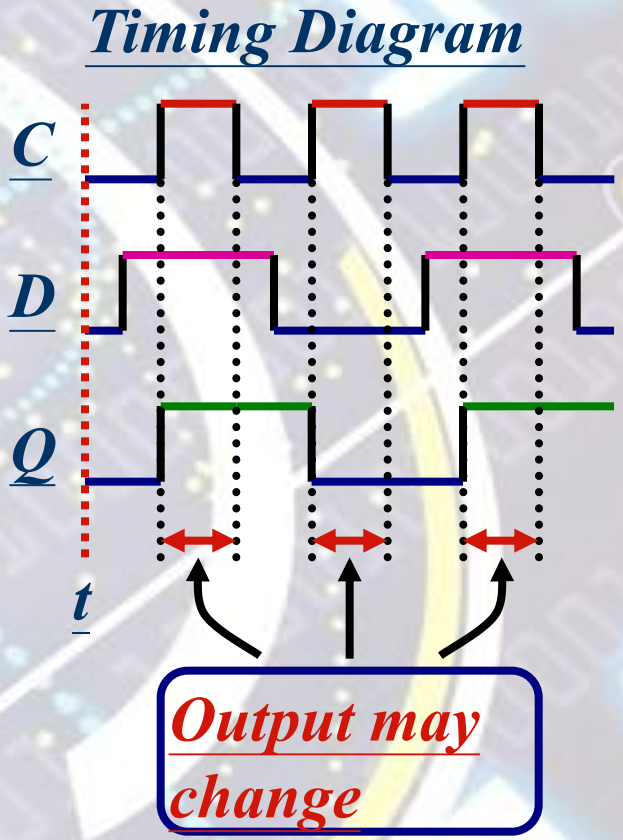
C S R	Q	
0 x x	Q_0	*No change*
1 0 0	Q_0	*No change*
1 0 1	0	*Reset*
1 1 0	1	*Set*
1 1 1	*Q=Q'*	*Invalid*

Controlled Latches

- *D* Latch (*D* = *Data*)

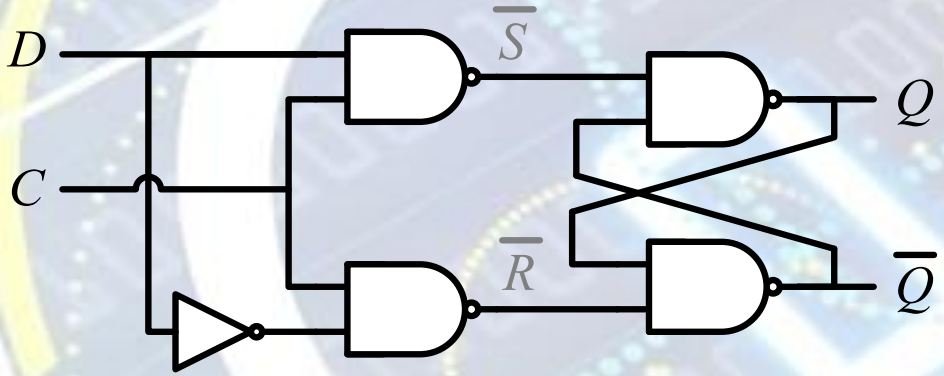


C D	Q	
0 x	Q_0	No change
1 0	0	Reset
1 1	1	Set



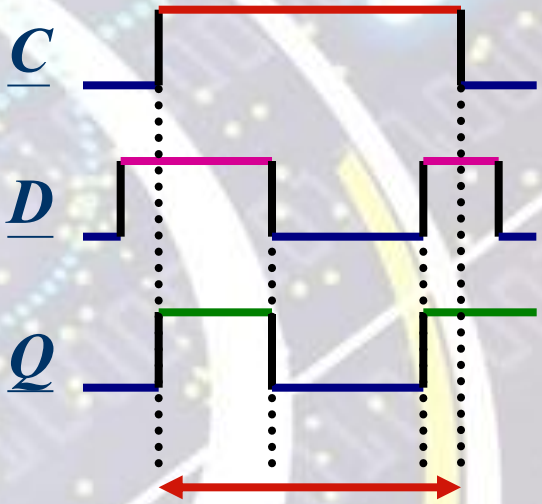
Controlled Latches

- *D* Latch (*D* = *Data*)



C	D	Q	
0	x	Q_0	No change
1	0	0	Reset
1	1	1	Set

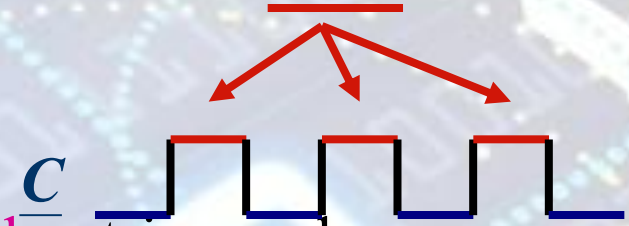
Timing Diagram



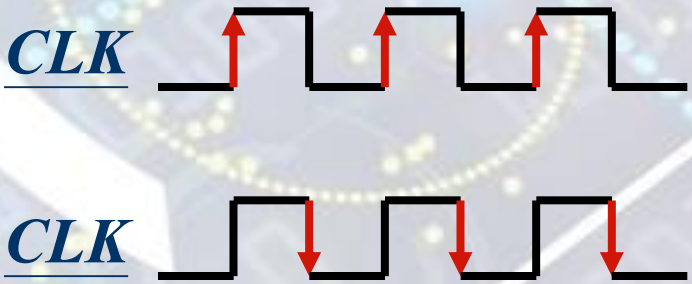
Output may change

Flip-Flops

- Controlled latches are level-triggered

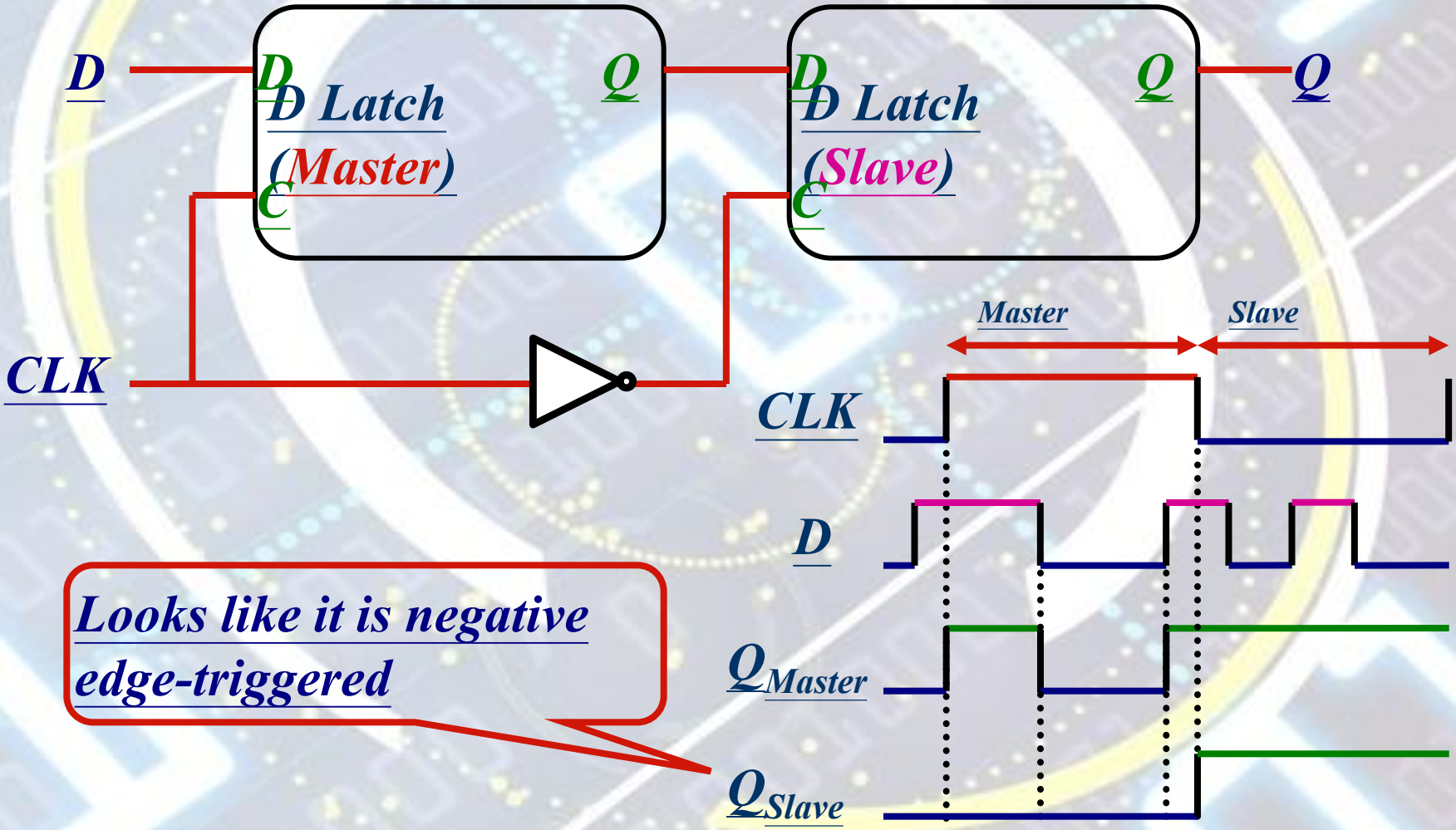


- Flip-Flops are edge-triggered



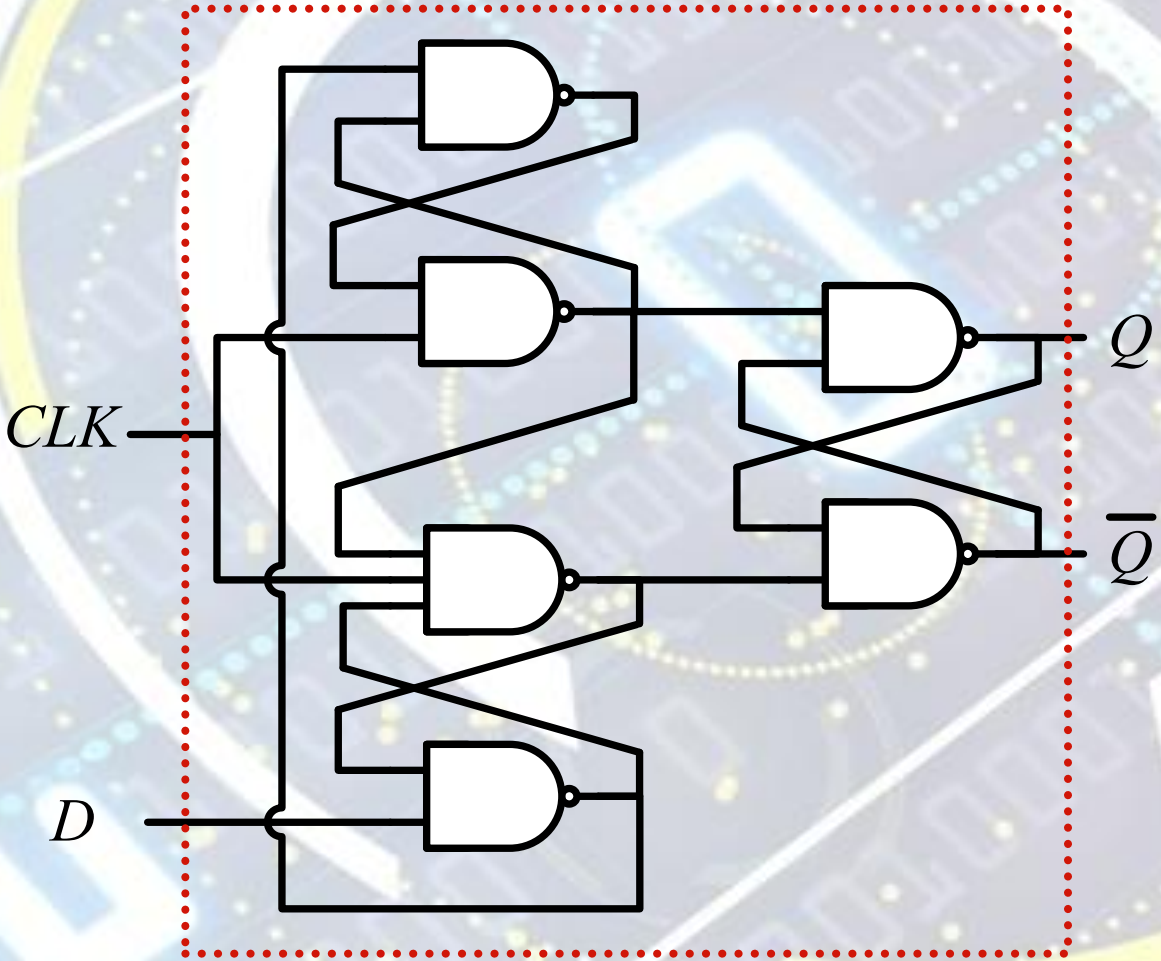
Flip-Flops

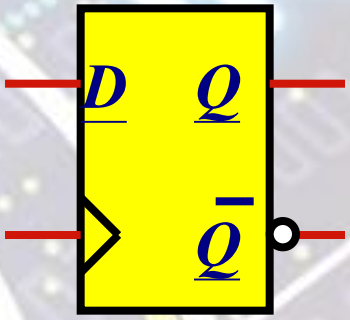
- Master-Slave *D* Flip-Flop



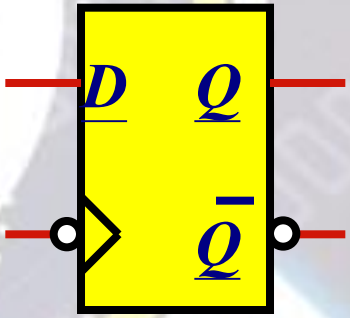
Flip-Flops

- Edge-Triggered *D* Flip-Flop





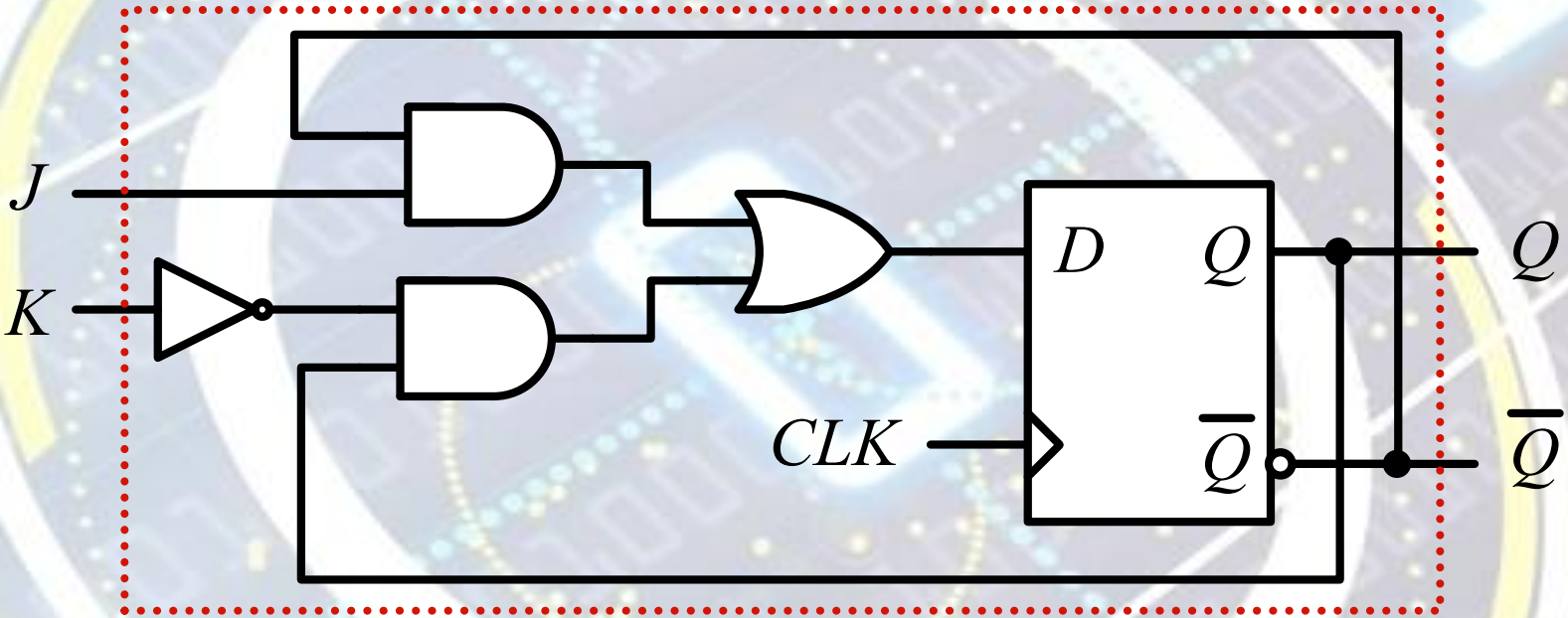
Positive Edge



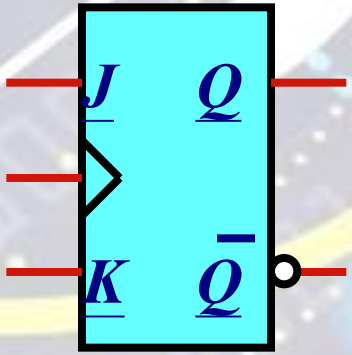
Negative Edge

Flip-Flops

- *JK* Flip-Flop

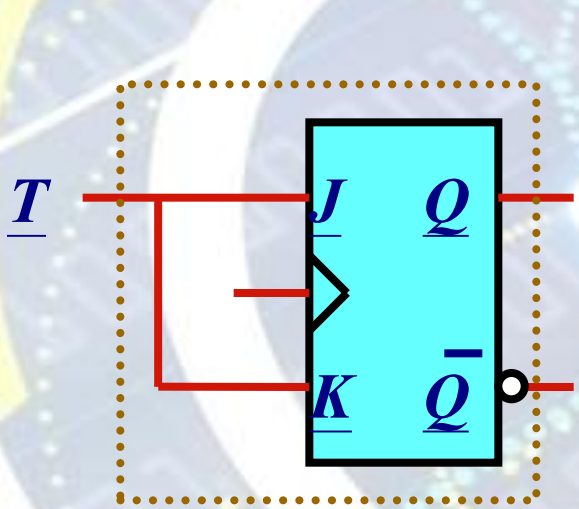


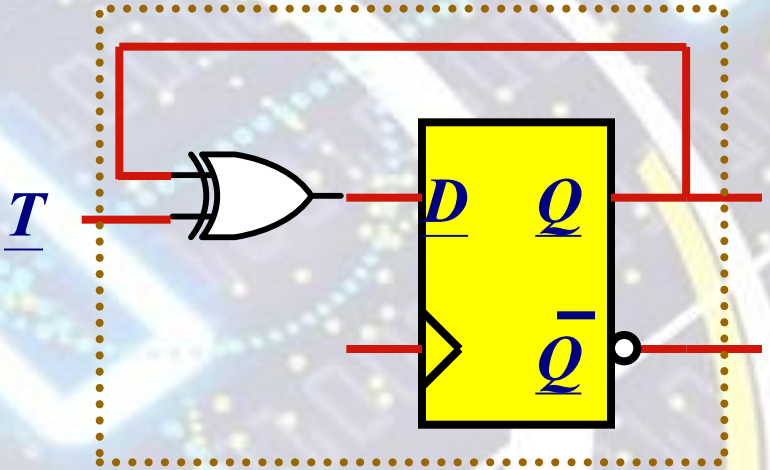
$$D = JQ' + K'Q$$

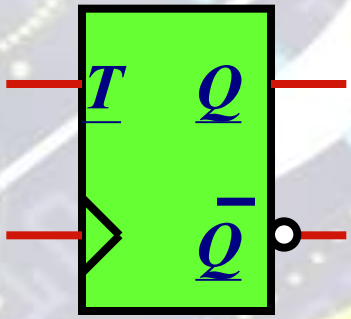


Flip-Flops

- *T* Flip-Flop



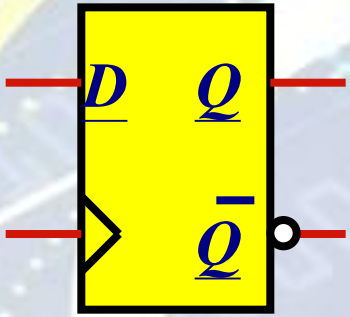




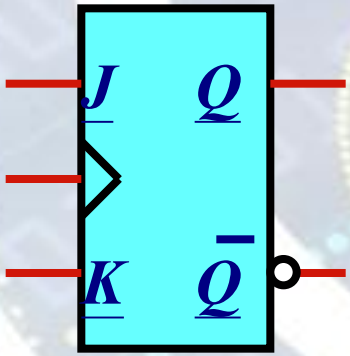
$$D = JQ' + K'Q$$

$$D = TQ' + T'Q = T \oplus Q$$

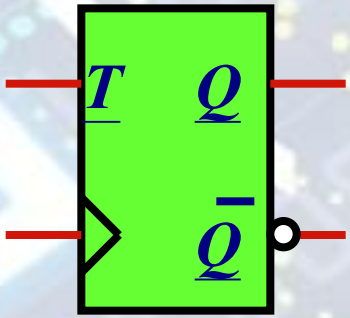
Flip-Flop Characteristic Tables



D	Q(t+1)	
0	0	Reset
1	1	Set

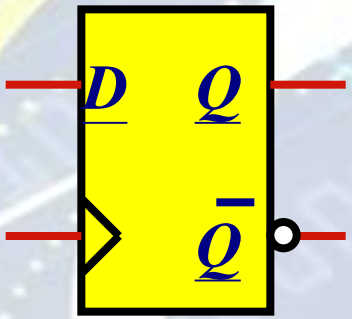


J	K	Q(t+1)	
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Toggle



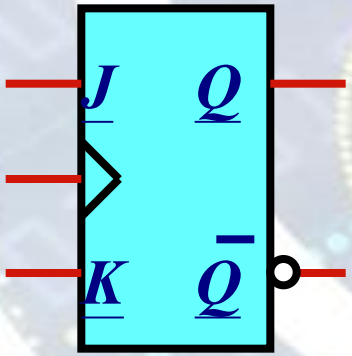
T	Q(t+1)	
0	Q(t)	No change
1	Q'(t)	Toggle

Flip-Flop Characteristic Equations



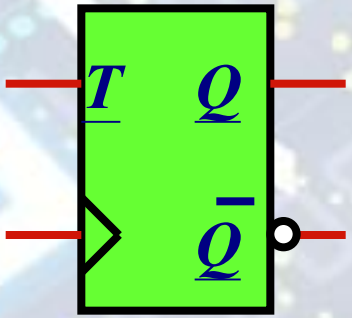
D	Q(t+1)
0	0
1	1

$Q(t+1) = D$



J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

$Q(t+1) = JQ' + K'Q$

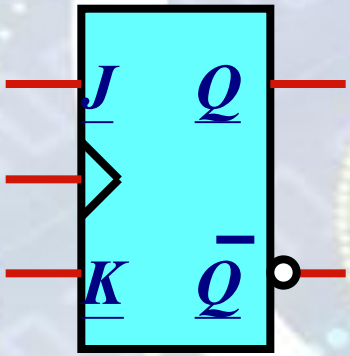


T	Q(t+1)
0	Q(t)
1	Q'(t)

$Q(t+1) = T \oplus Q$

Flip-Flop Characteristic Equations

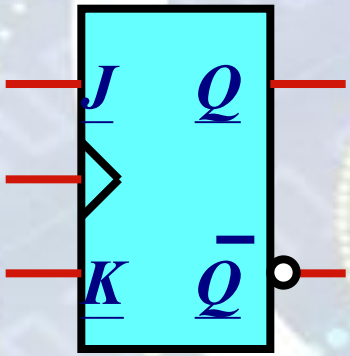
- Analysis / Derivation



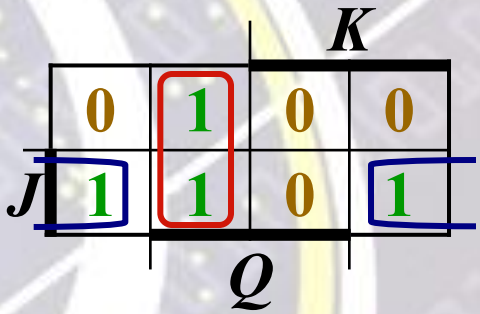
J	K	Q(t)	Q(t+1)	
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	

Flip-Flop Characteristic Equations

- Analysis / Derivation



J	K	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



$$Q(t+1) = JQ' + K'Q$$

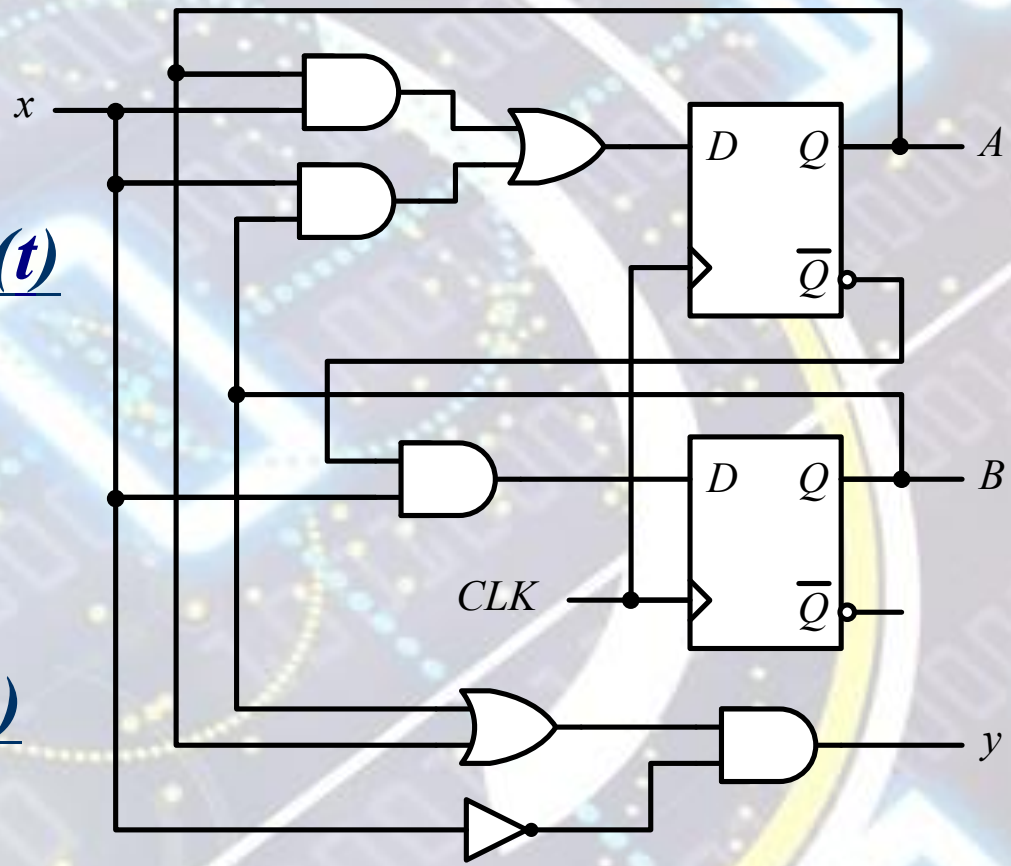
Week -15

Page(245-265)

Analysis of Clocked Sequential Circuits

- State Equations

$$A(t+1) = D_A$$
$$= A(t)\,x(t) + B(t)\,x(t)$$
$$= A\,x + B\,x$$
$$B(t+1) = D_B$$
$$= A'(t)\,x(t)$$
$$= A'\,x$$
$$y(t) = [A(t) + B(t)]\,x'(t)$$
$$= (A + B)\,x'$$

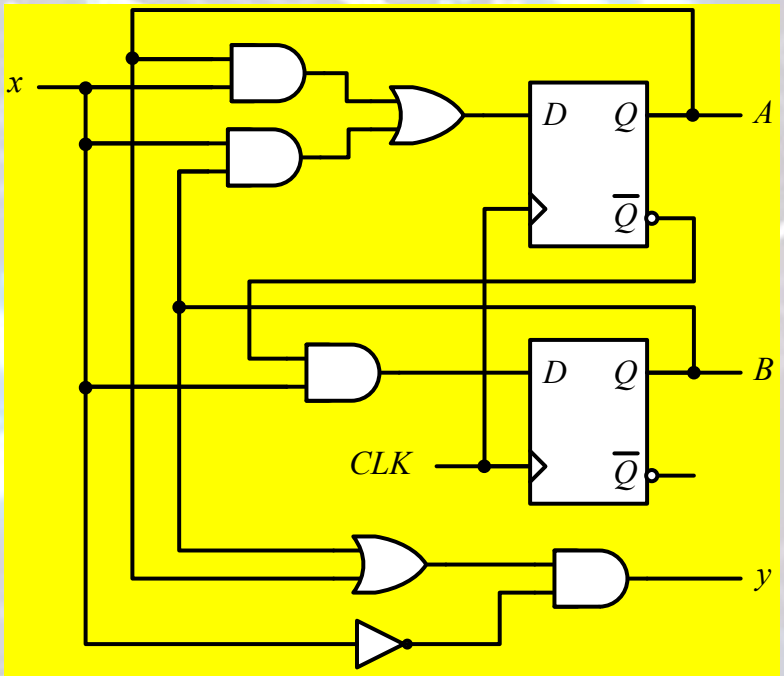


Analysis of Clocked Sequential Circuits

- State Table (Transition Table)

Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

t *t+1* *t*



$$A(t+1) = Ax + Bx$$

$$B(t+1) = A'x$$

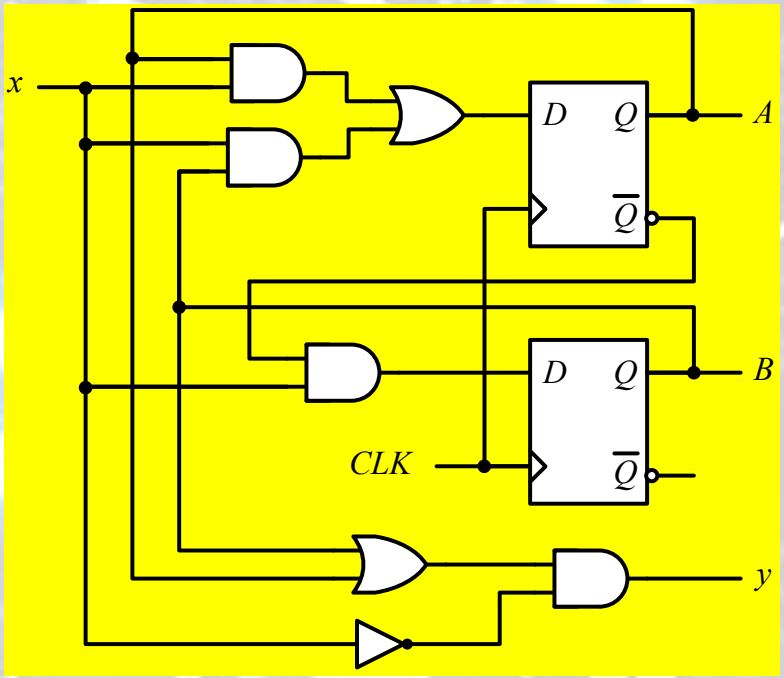
$$y(t) = (A + B)x'$$

Analysis of Clocked Sequential Circuits

- State Table (Transition Table)

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
$A\ B$	$A\ B$	$A\ B$	y	y
0 0	0 0	0 1	0	0
0 1	0 0	1 1	1	0
1 0	0 0	1 0	1	0
1 1	0 0	1 0	1	0

t *t+1* *t*



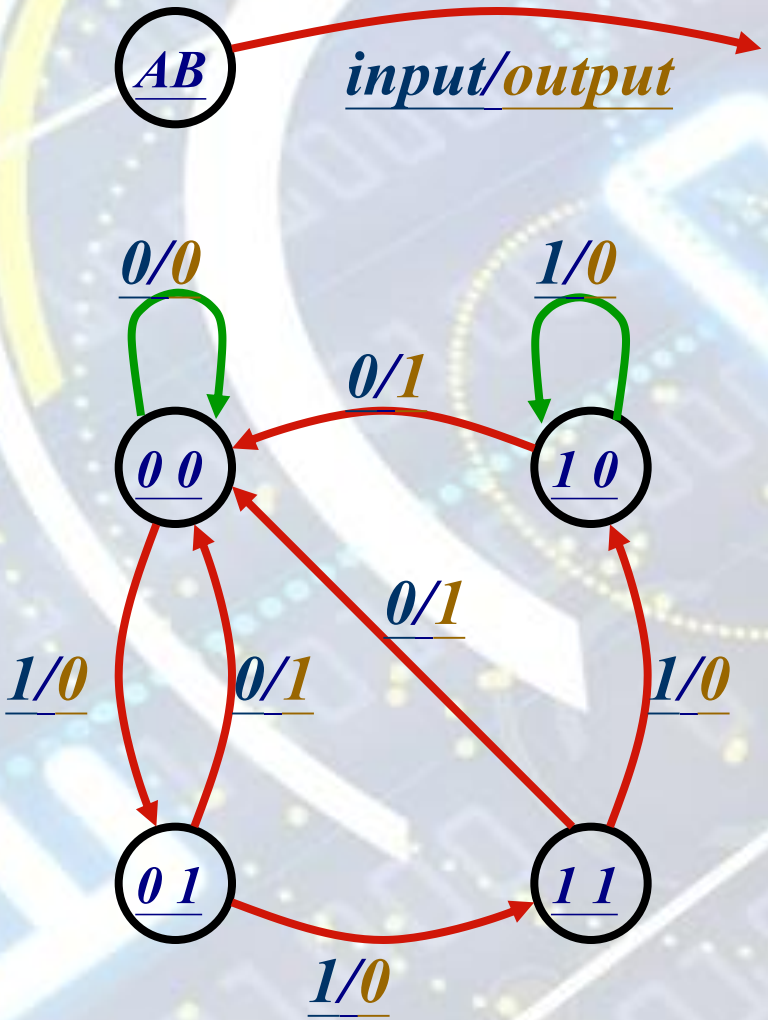
$$A(t+1) = A\,x + B\,x$$

$$B(t+1) = A'\,x$$

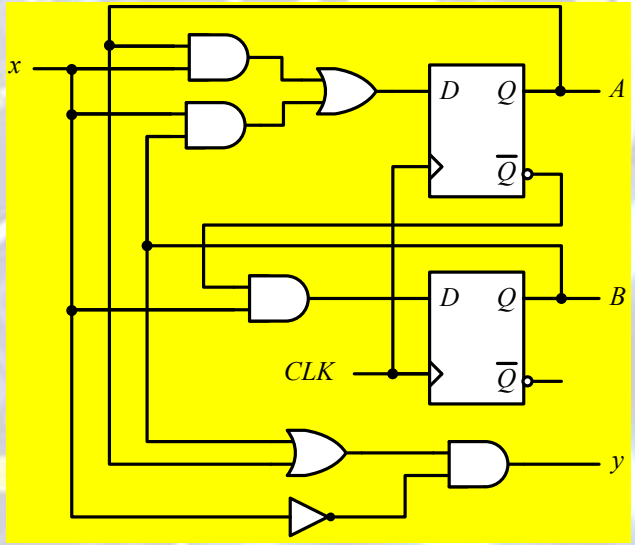
$$y(t) = (A + B)\,x'$$

Analysis of Clocked Sequential Circuits

- State Diagram



Present State	Next State				Output	
	$x = 0$		$x = 1$		$x = 0$	$x = 1$
A B	A	B	A	B	y	y
0 0	0	0	0	1	0	0
0 1	0	0	1	1	1	0
1 0	0	0	1	0	1	0
1 1	0	0	1	0	1	0

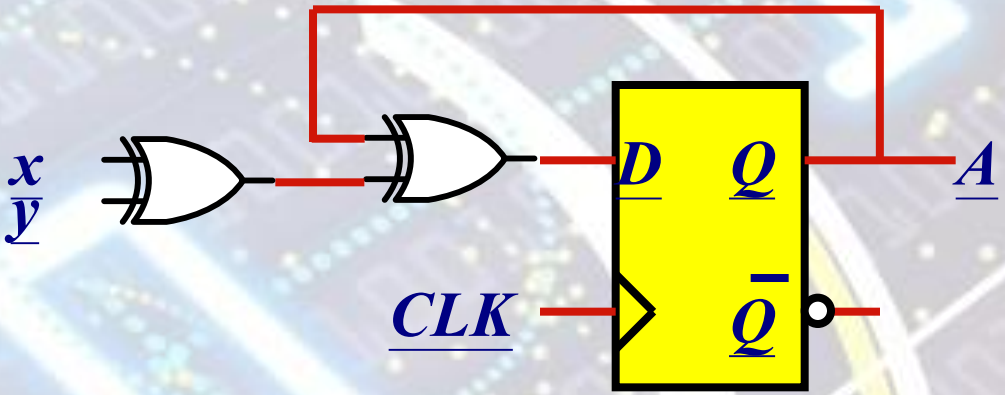


Analysis of Clocked Sequential Circuits

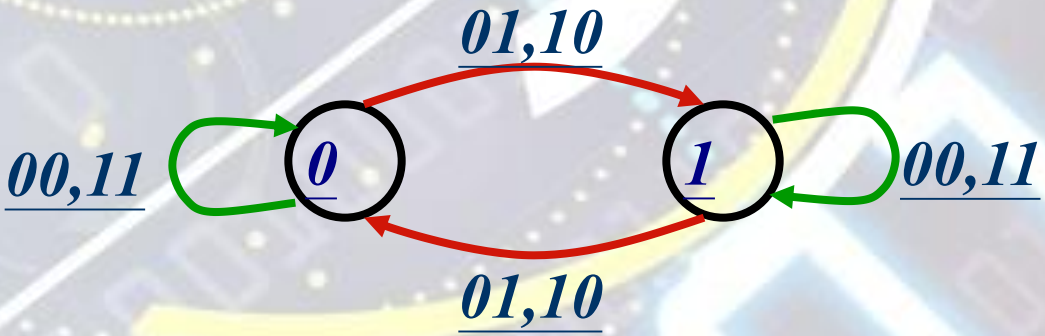
- *D* Flip-Flops

Example:

Present State	Input		Next State
A	*x*	*y*	*A*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



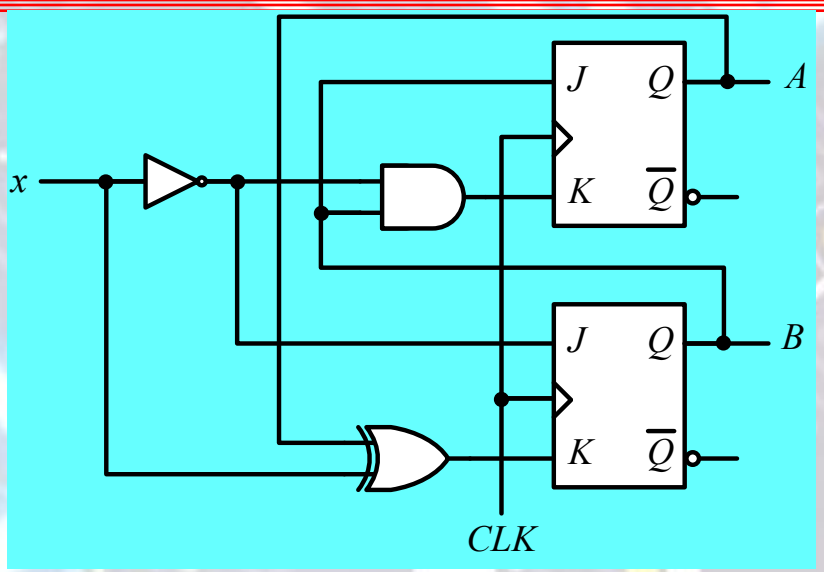
$$A(t+1) = D_A = A \oplus x \oplus y$$



Analysis of Clocked Sequential Circuits

- *JK* Flip-Flops

Example:



Present State		I/P	Next State		Flip-Flop Inputs			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

$$J_A = B \qquad K_A = Bx'$$

$$J_B = x' \qquad K_B = A \oplus x$$

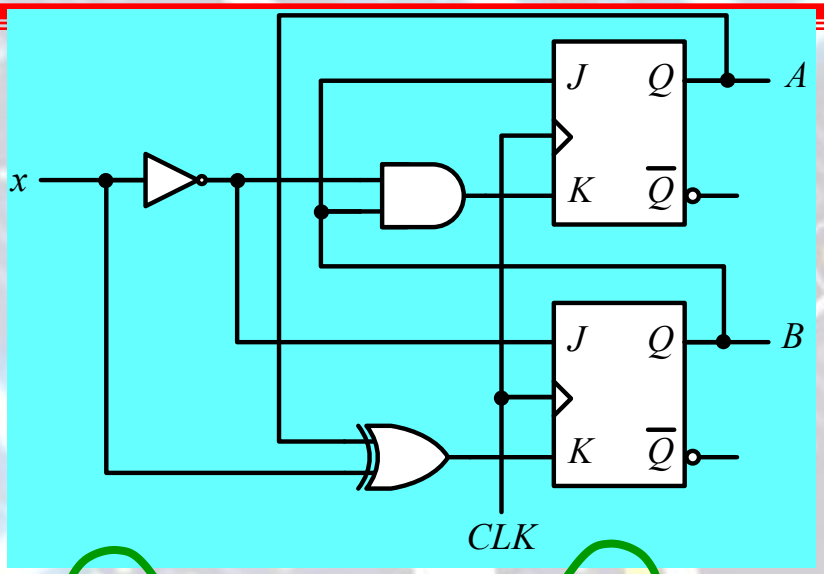
$$A(t+1) = J_A Q'_A + K'_A Q_A$$
$$= A'B + AB' + Ax$$

$$B(t+1) = J_B Q'_B + K'_B Q_B$$
$$= B'x' + ABx + A'Bx'$$

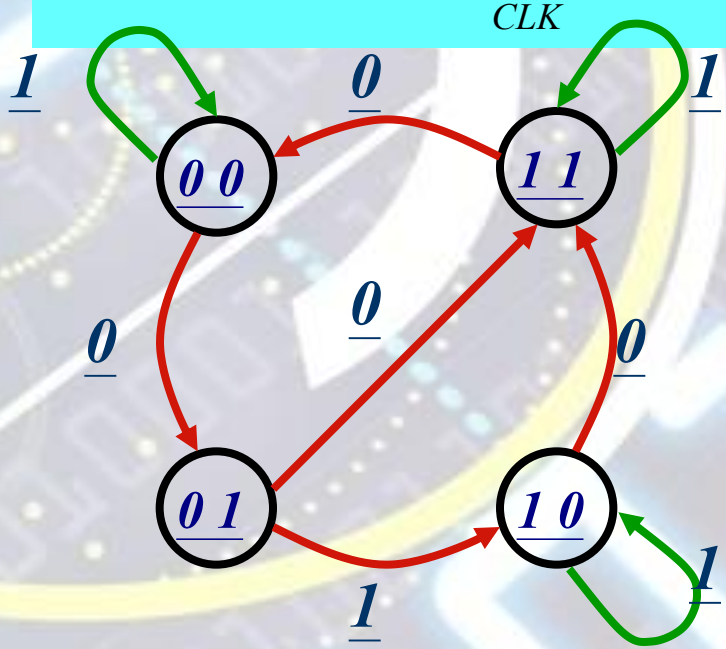
Analysis of Clocked Sequential Circuits

- *JK* Flip-Flops

Example:



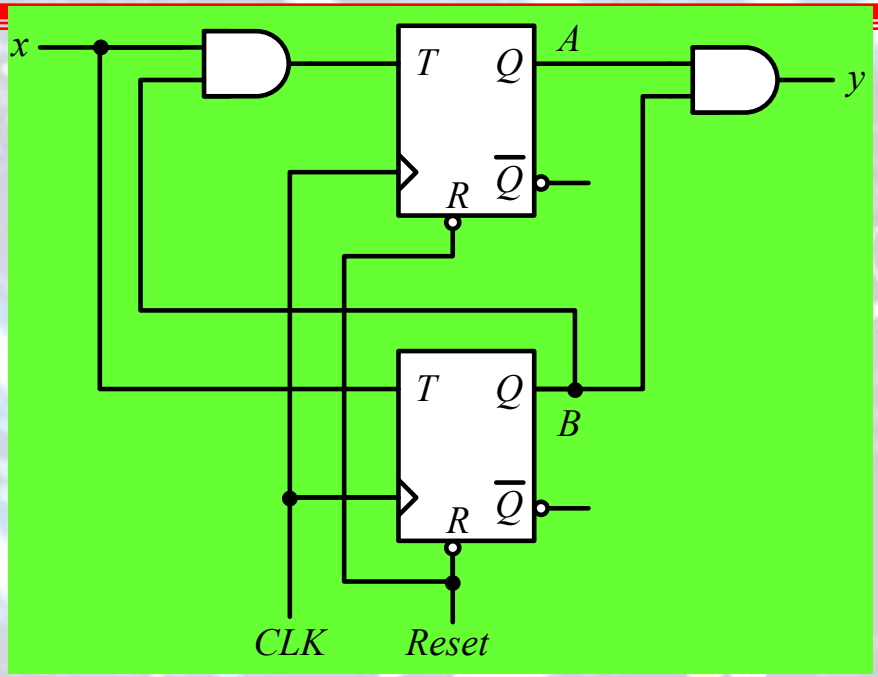
Present State		I/P	Next State		Flip-Flop Inputs			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0



Analysis of Clocked Sequential Circuits

- *T* Flip-Flops

Example:



Present State		I/P	Next State		F.F Inputs		O/P
A	*B*	*x*	*A*	*B*	T_A	T_B	*y*
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	1	0	0	0
0	1	1	1	0	1	1	0
1	0	0	1	0	0	0	0
1	0	1	1	1	0	1	0
1	1	0	1	1	0	0	1
1	1	1	0	0	1	1	1

$$T_A = B\,x \qquad T_B = x$$

$$y = A\,B$$

$$A(t+1) = T_A Q'_A + T'_A Q_A$$
$$= AB' + Ax' + A'Bx$$

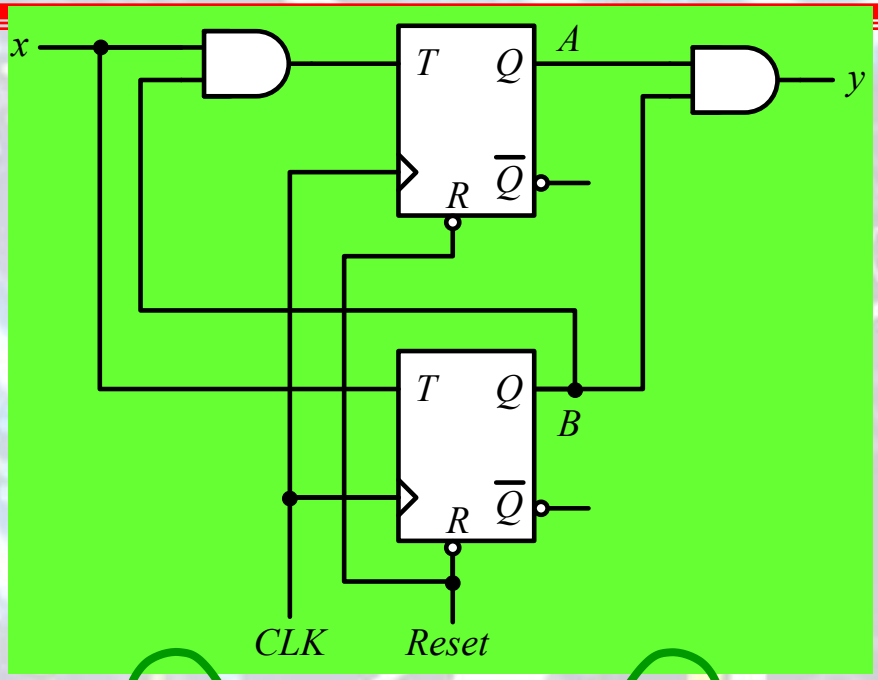
$$B(t+1) = T_B Q'_B + T'_B Q_B$$
$$= x \oplus B$$

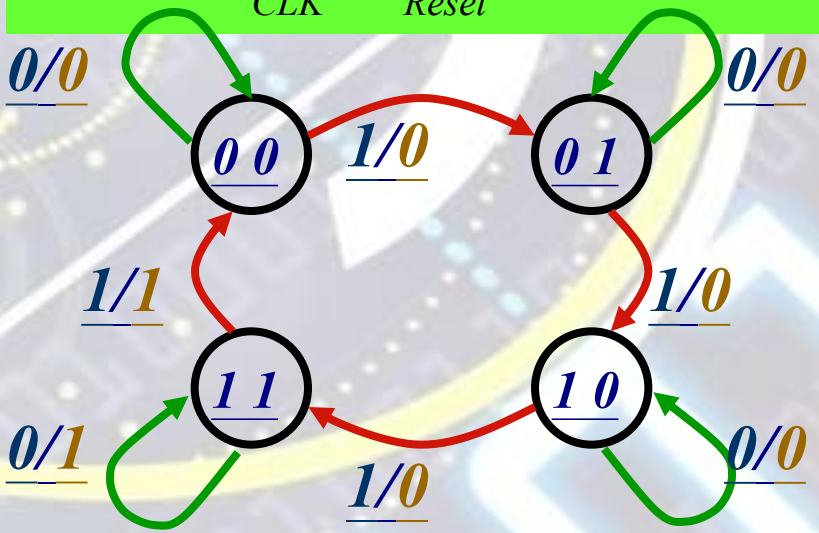
Analysis of Clocked Sequential Circuits

- *T* Flip-Flops

Example:

Present State		I/P	Next State		F.F Inputs		O/P
A	*B*	*x*	*A*	*B*	T_A	T_B	*y*
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	1	0	0	0
0	1	1	1	0	1	1	0
1	0	0	1	0	0	0	0
1	0	1	1	1	0	1	0
1	1	0	1	1	0	0	1
1	1	1	0	0	1	1	1

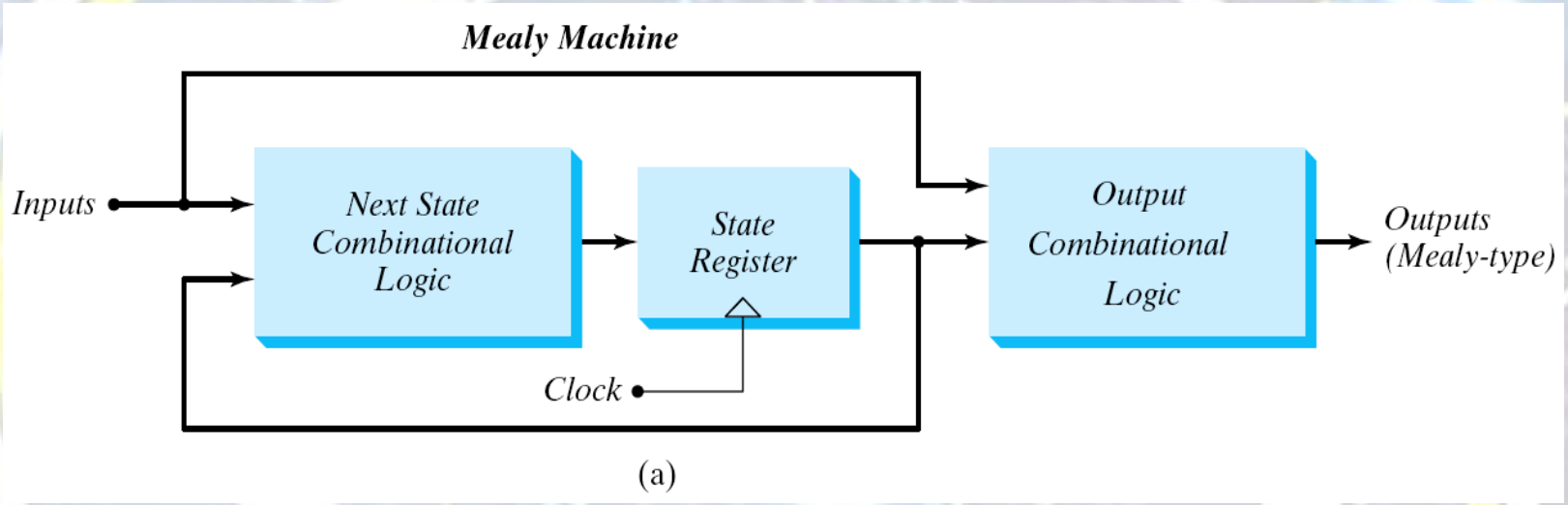




Mealy and Moore Models

- The Mealy model: the outputs are functions of both the present state and inputs (Fig. 5-15).
  - The outputs may change if the inputs change during the clock pulse period.
    - » The outputs may have momentary false values unless the inputs are synchronized with the clocks.
- The Moore model: the outputs are functions of the present state only (Fig. 5-20).
  - The outputs are synchronous with the clocks.

Mealy and Moore Models



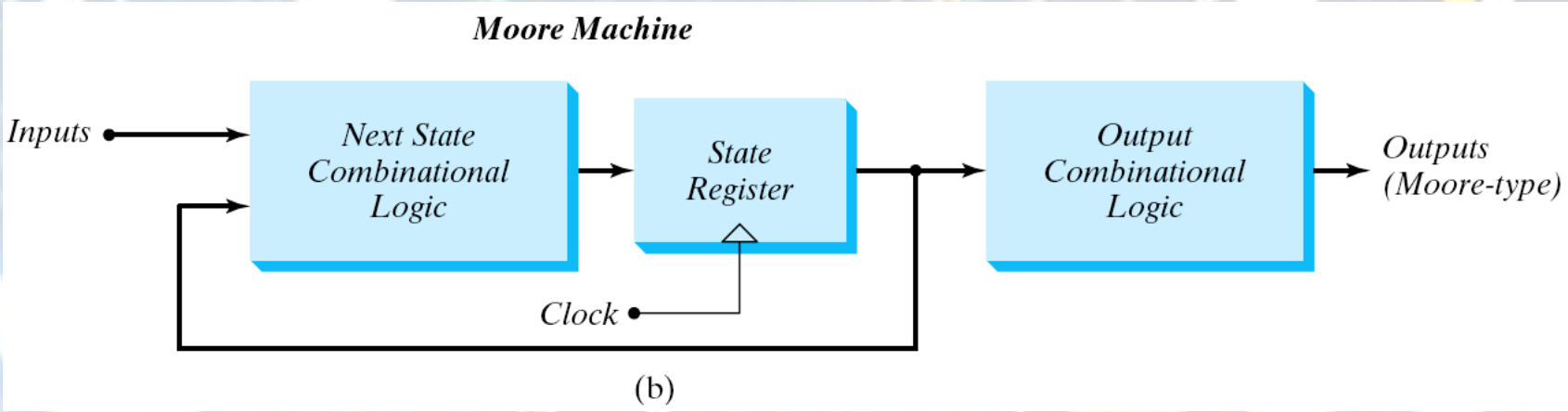


Fig. 5.21 Block diagram of Mealy and Moore state machine

Mealy and Moore Models

Mealy

Present State		I/P	Next State		O/P
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

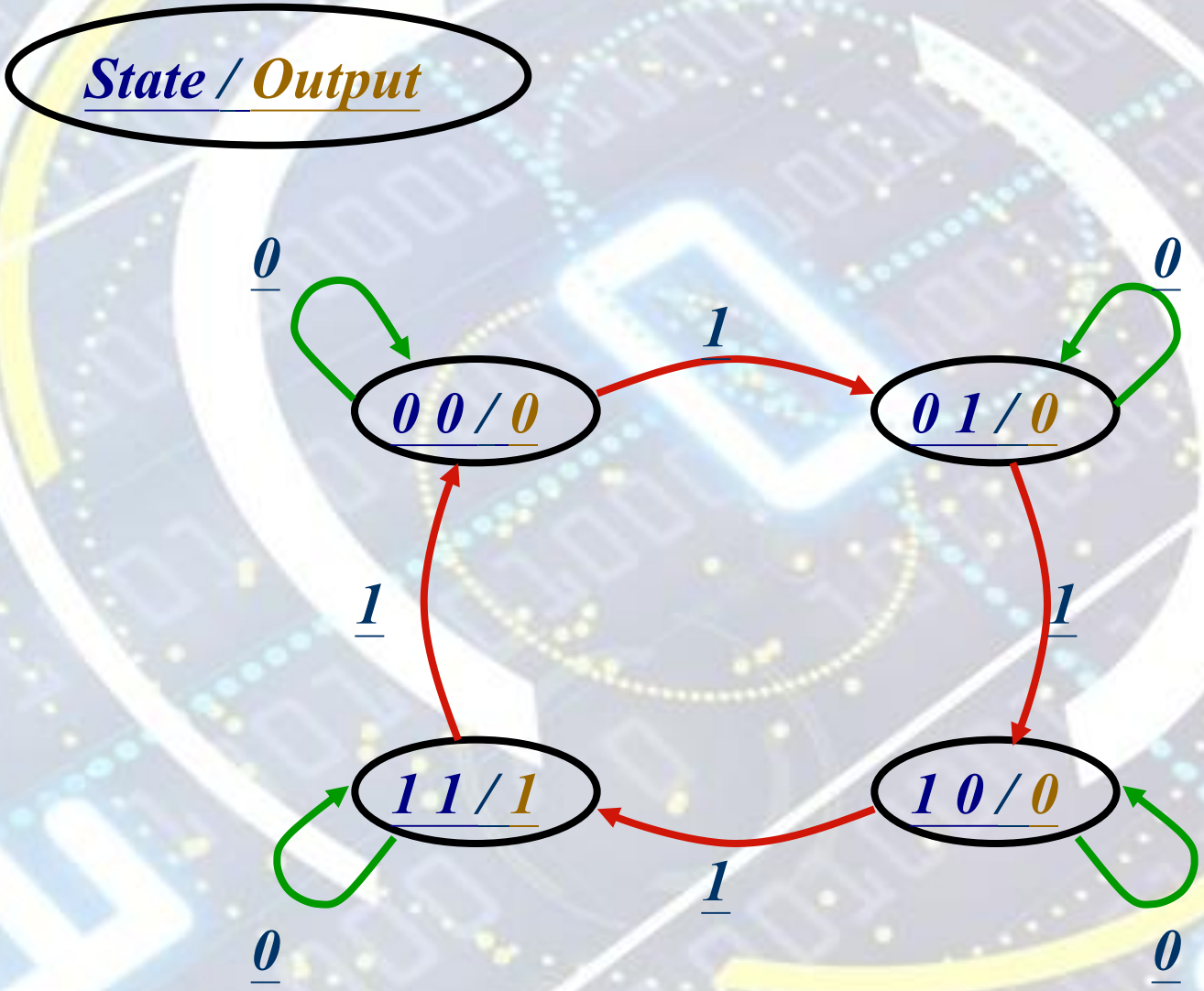
For the same state,
the output changes with the input

Moore

Present State		I/P	Next State		O/P
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	1

For the same state,
the output does not change with the input

Moore State Diagram



State Reduction and Assignment

- State Reduction Reductions on the number of flip-flops and the number of gates.
  - A reduction in the number of states may result in a reduction in the number of flip-flops.
  - An example state diagram showing in Fig. 5.25.

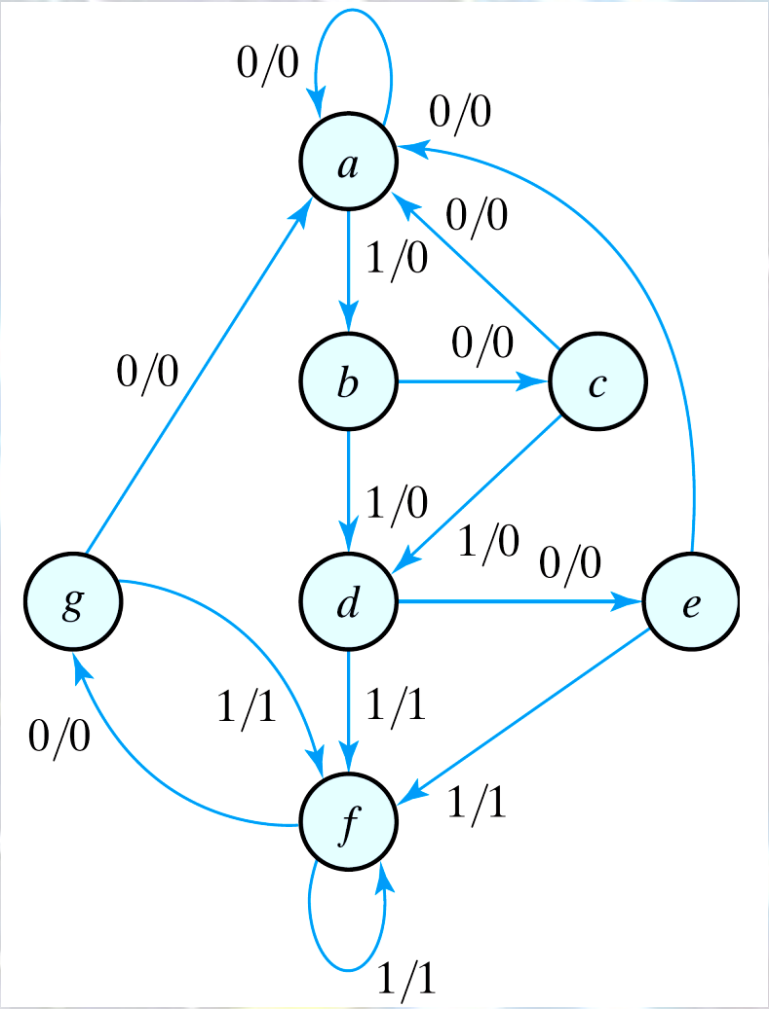


Fig. 5.25 State diagram

State Reduction

State:	**a**	**a**	**b**	**c**	**d**	**e**	**f**	**f**	**g**	**f**	**g**	**a**
Input:	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	
Output:	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	

- Only the input-output sequences are important.
- Two circuits are equivalent
  - » Have identical outputs for all input sequences;
  - » The number of states is not important.

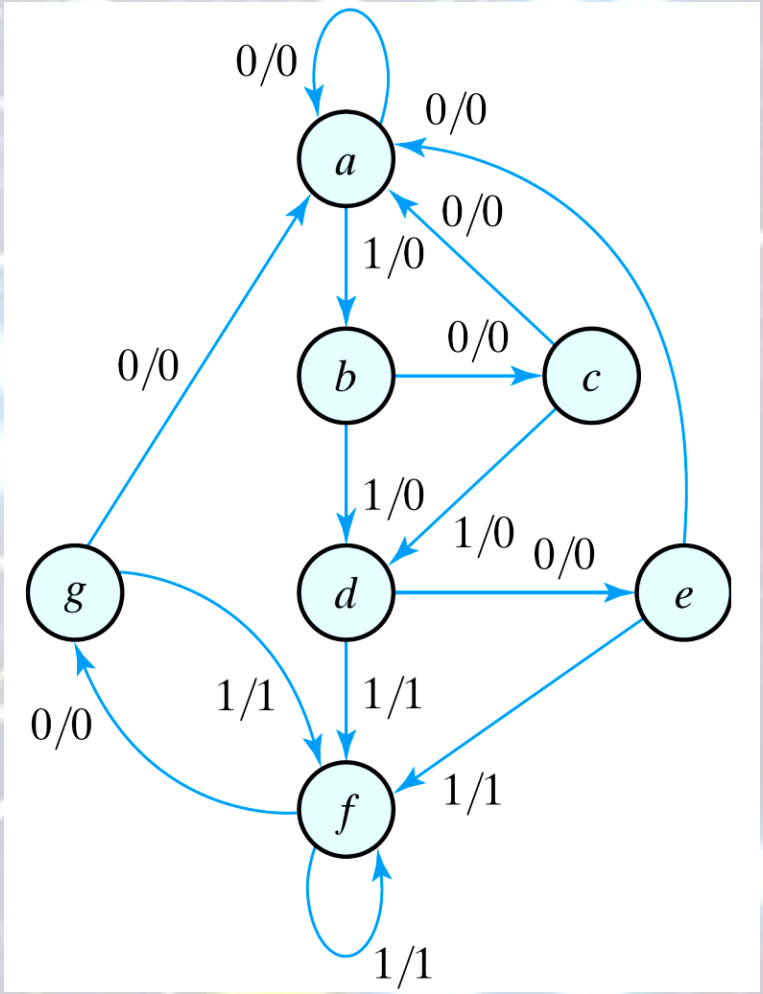
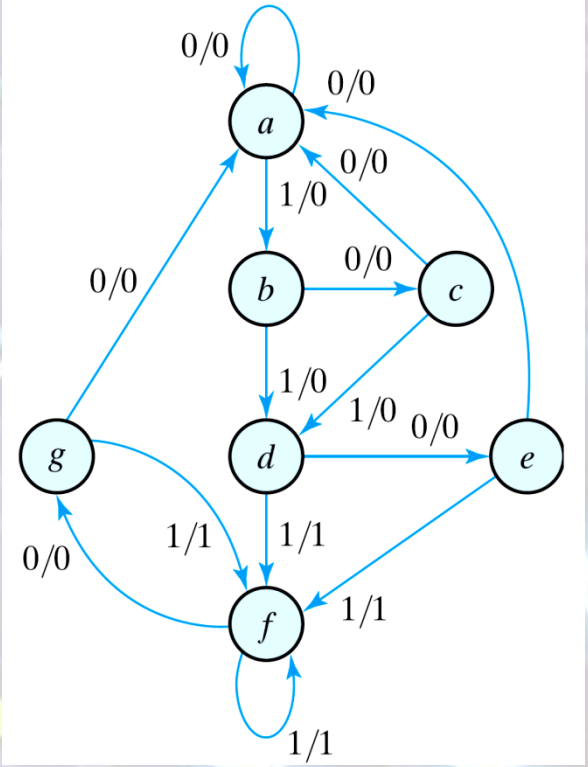


Fig. 5.25 State diagram

- Equivalent states
  - Two states are said to be equivalent
    - » For each member of the set of inputs, they give exactly the same output and send the circuit to the same state or to an equivalent state.

Table 5.6
State Table

	Next State		Output	
Present State	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1



- Reducing the state table
  - $e = g$ (remove g);
  - $d = f$ (remove f);

Table 5.7
Reducing the State Table

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	e	f	0	1

- The reduced finite state machine

Table 5.8
Reduced State Table

	Next State		Output	
Present State	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

State:	a	a	b	c	d	e	d	d	e	d	e	a
Input:	0	1	0	1	0	1	1	0	1	0	0	
Output:	0	0	0	0	0	1	1	0	1	0	0	

- The checking of each pair of states for possible equivalence can be done systematically using Implication Table.
- The unused states are treated as don't-care condition ⇒ fewer combinational gates.

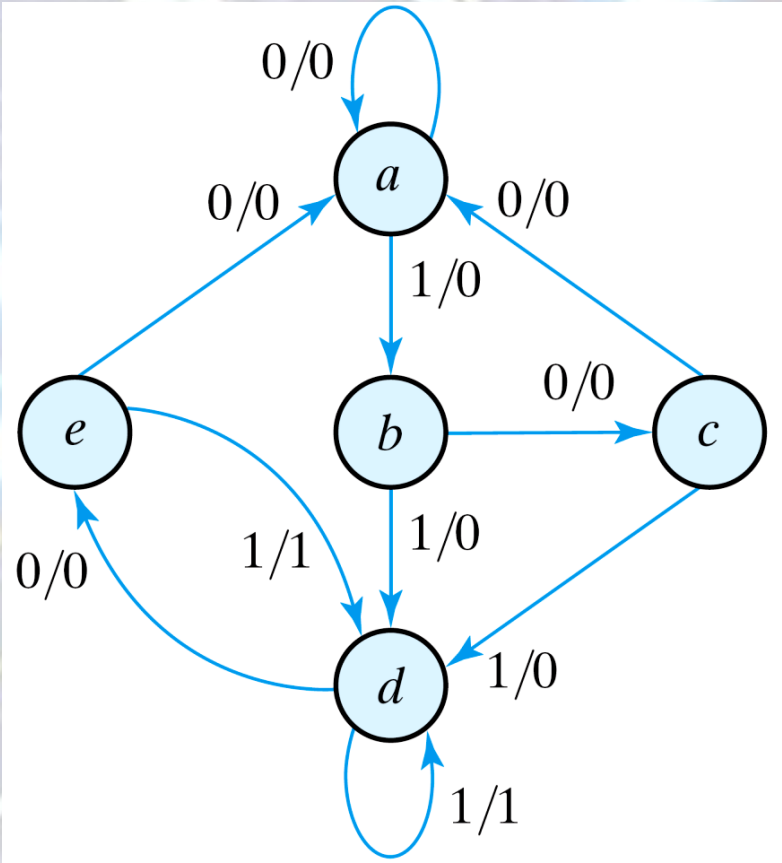


Fig. 5.26 Reduced State diagram

Table 5.8
Reduced State Table

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

State Assignment

- State Assignment
- To minimize the cost of the combinational circuits.
  - Three possible binary state assignments. (m states need n-bits, where $2^n > m$)

Table 5.9
Three Possible Binary State Assignments

State	Assignment 1, Binary	Assignment 2, Gray Code	Assignment 3, One-Hot
a	000	000	00001
b	001	001	00010
c	010	011	00100
d	011	010	01000
e	100	110	10000

- Any binary number assignment is satisfactory as long as each state is assigned a unique number.
- Use binary assignment 1.

Table 5.10
Reduced State Table with Binary Assignment 1

	Next State		Output	
Present State	$x = 0$	$x = 1$	$x = 0$	$x = 1$
000	000	001	0	0
001	010	011	0	0
010	000	011	0	0
011	100	011	0	1
100	000	011	0	1

Week -16

Page(267-273)

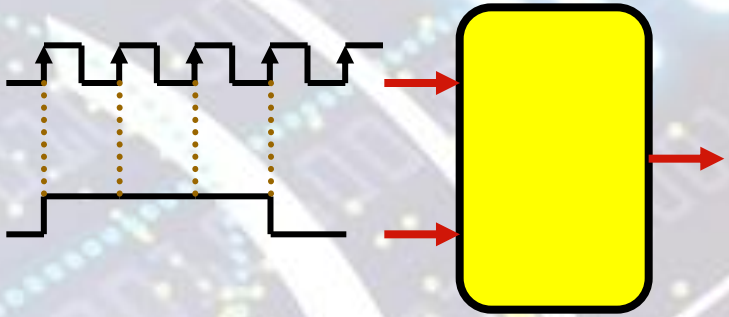
Design Procedure

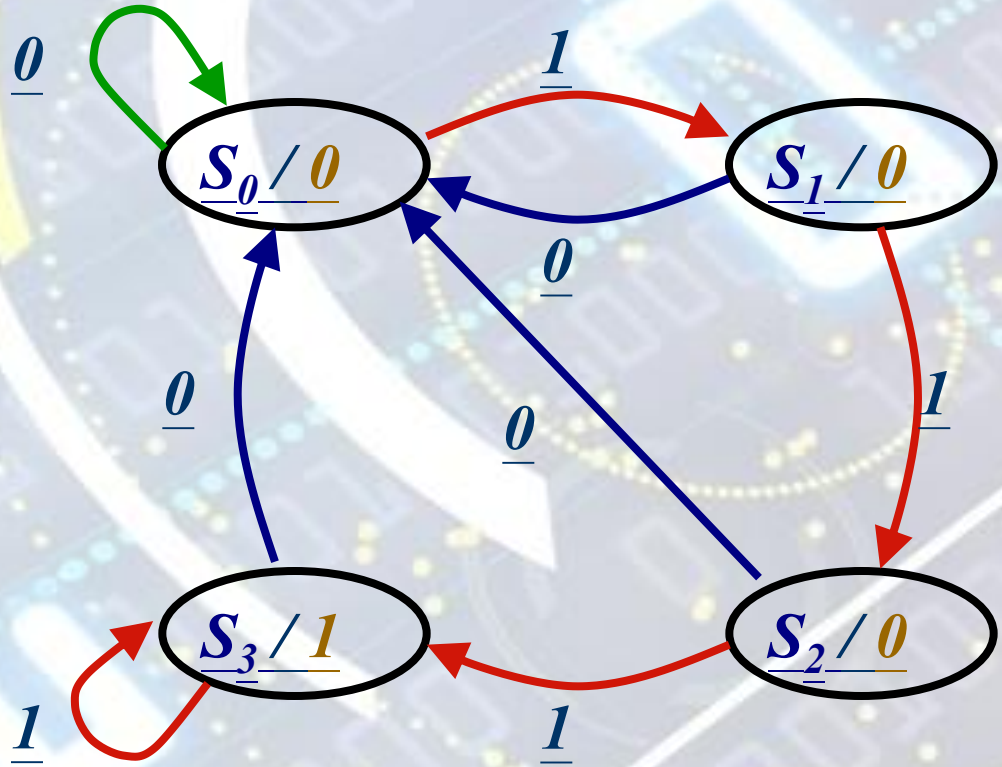
- Design Procedure for sequential circuit
  - The word description of the circuit behavior to get a state diagram;
  - State reduction if necessary;
  - Assign binary values to the states;
  - Obtain the binary-coded state table;
  - Choose the type of flip-flops;
  - Derive the simplified flip-flop input equations and output equations;
  - Draw the logic diagram;

Design of Clocked Sequential Circuits

- *Example*:

Detect 3 or more consecutive 1's





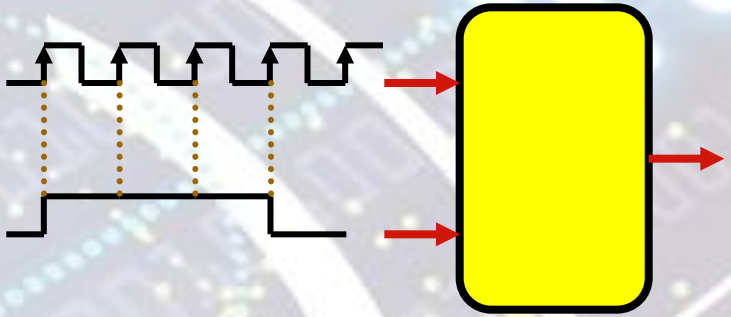
State	A B
S_0	0 0
S_1	0 1
S_2	1 0
S_3	1 1

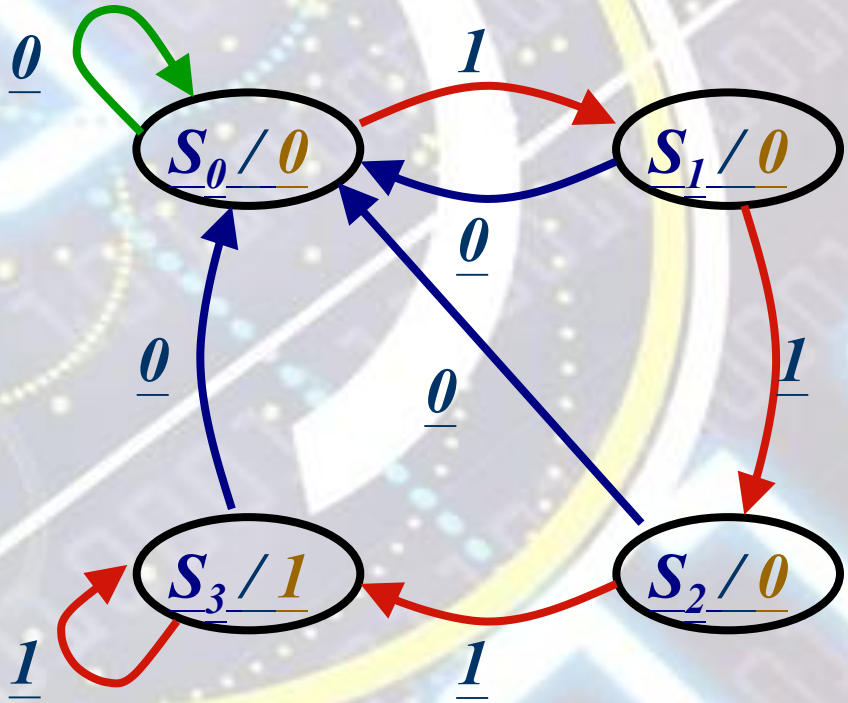
Design of Clocked Sequential Circuits

- *Example*:

Detect 3 or more consecutive 1's

Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

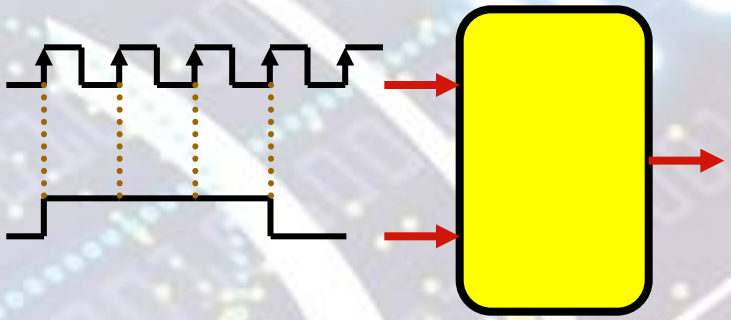




Design of Clocked Sequential Circuits

- *Example*:

Detect 3 or more consecutive 1's



Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

Synthesis using D Flip-Flops

$$A(t+1) = D_A(A, B, x) = \sum(3, 5, 7)$$

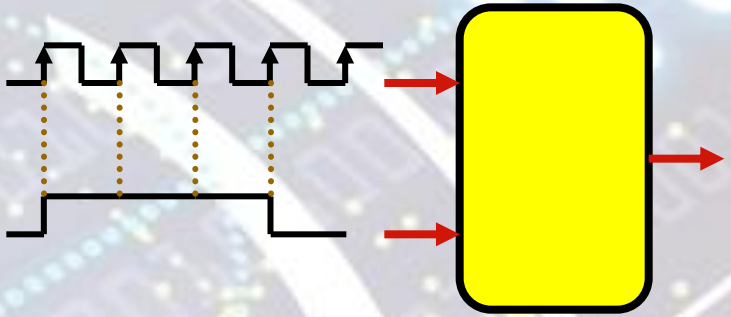
$$B(t+1) = D_B(A, B, x) = \sum(1, 5, 7)$$

$$y(A, B, x) = \sum(6, 7)$$

Design of Clocked Sequential Circuits with *D* F.F.

- *Example*:

Detect 3 or more consecutive 1's

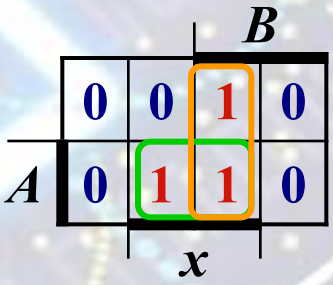


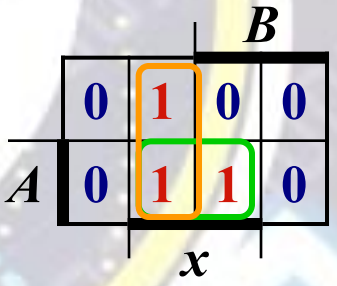
Synthesis using D Flip-Flops

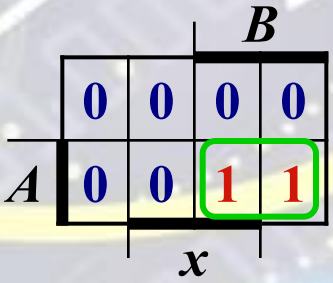
$$D_A(A, B, x) = \sum(3, 5, 7)$$
$$= A\,x + B\,x$$

$$D_B(A, B, x) = \sum(1, 5, 7)$$
$$= A\,x + B'x$$

$$y(A, B, x) = \sum(6, 7)$$
$$= A\,B$$



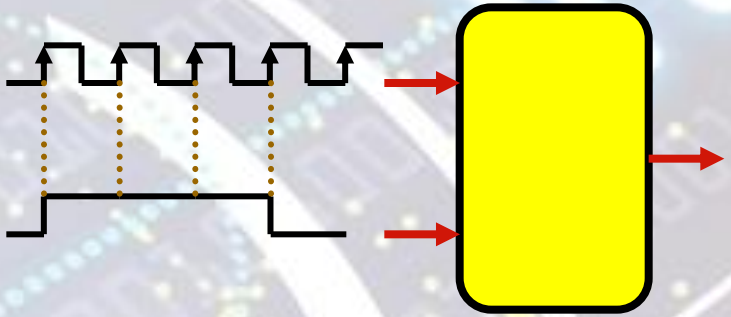




Design of Clocked Sequential Circuits with *D* F.F.

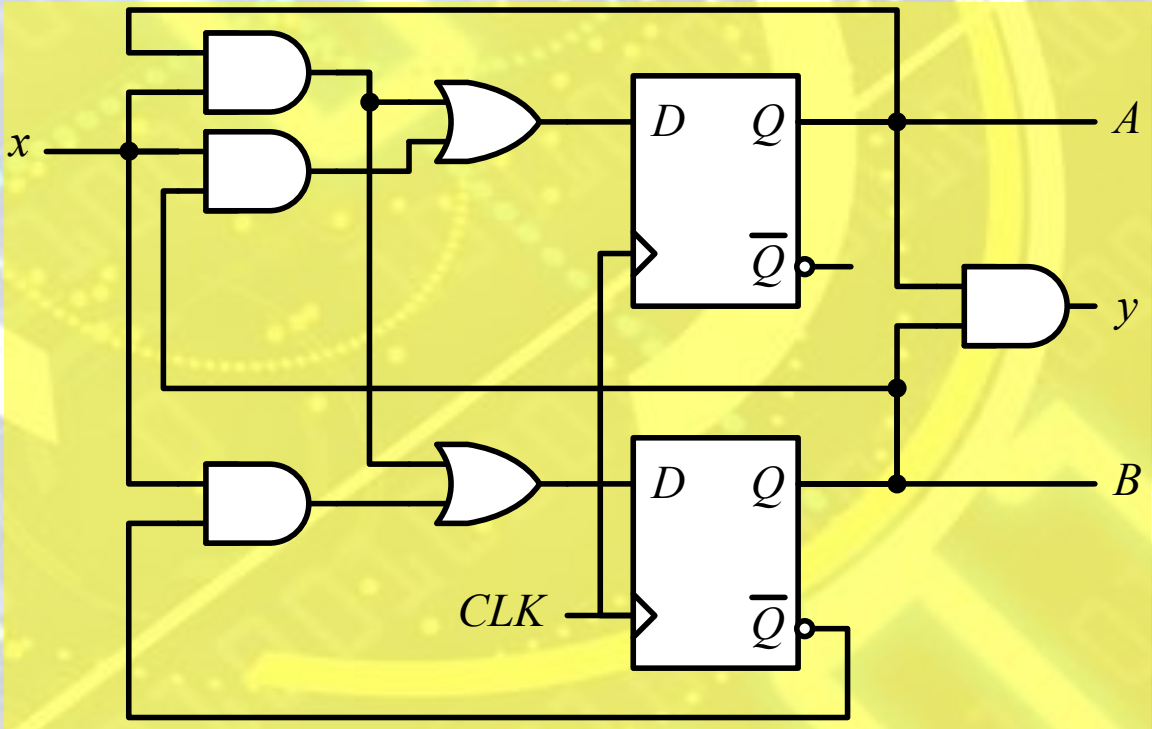
- *Example*:

Detect 3 or more consecutive 1's



Synthesis using D Flip-Flops

$$D_A = A\,x + B\,x$$
$$D_B = A\,x + B'\,x$$
$$y = A\,B$$



Flip-Flop Excitation Tables

Present State	Next State	F.F. Input
$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

Present State	Next State	F.F. Input	
$Q(t)$	$Q(t+1)$	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

0 0 (No change)
0 1 (Reset)

1 0 (Set)
1 1 (Toggle)

0 1 (Reset)
1 1 (Toggle)

0 0 (No change)
1 0 (Set)

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

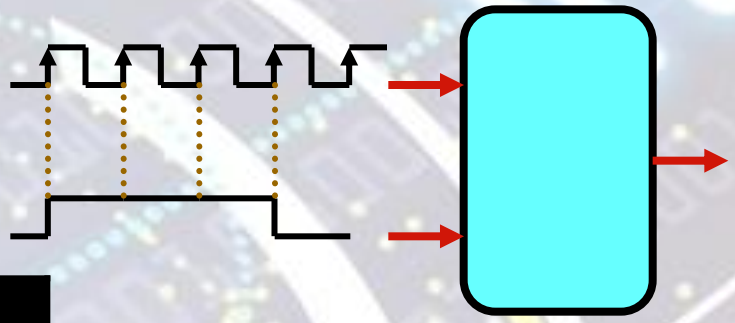
Week -17

Page(275-280)

Design of Clocked Sequential Circuits with *JK* F.F.

- *Example*:

Detect 3 or more consecutive 1's



Present State		Input	Next State		Flip-Flop Inputs			
A	*B*	*x*	*A*	*B*	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	x	1	x
0	1	0	0	0	0	x	x	1
0	1	1	1	0	1	x	x	1
1	0	0	0	0	x	1	0	x
1	0	1	1	1	x	0	1	x
1	1	0	0	0	x	1	x	1
1	1	1	1	1	x	0	x	0

Synthesis using JK F.F.

$J_A(A, B, x) = \sum(3)$

$d_{JA}(A, B, x) = \sum(4,5,6,7)$

$K_A(A, B, x) = \sum(4, 6)$

$d_{KA}(A, B, x) = \sum(0,1,2,3)$

$J_B(A, B, x) = \sum(1, 5)$

$d_{JB}(A, B, x) = \sum(2,3,6,7)$

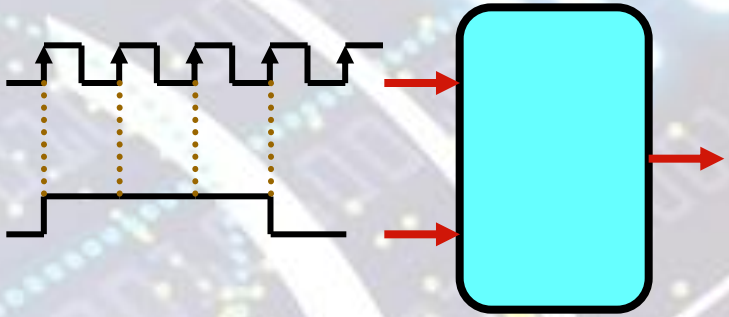
$K_B(A, B, x) = \sum(2, 3, 6)$

$d_{KB}(A, B, x) = \sum(0,1,4,5)$

Design of Clocked Sequential Circuits with *JK* F.F.

- *Example*:

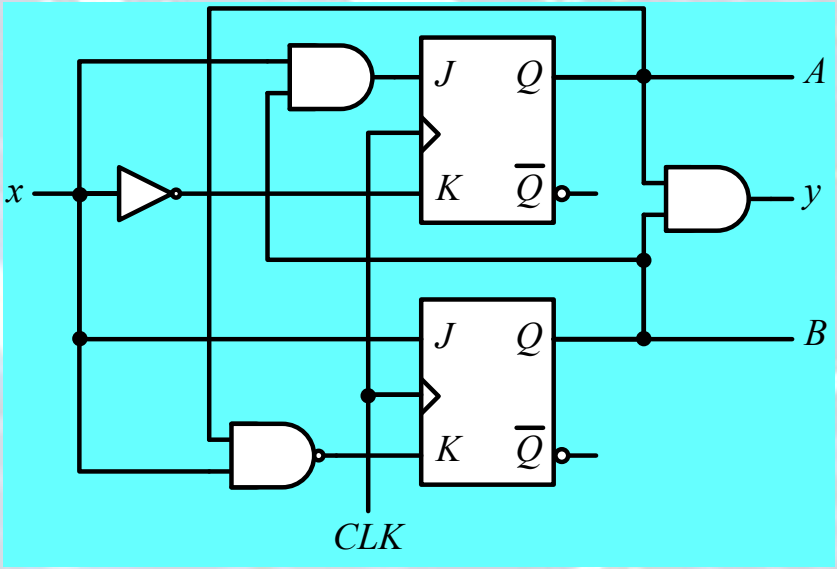
Detect 3 or more consecutive 1's

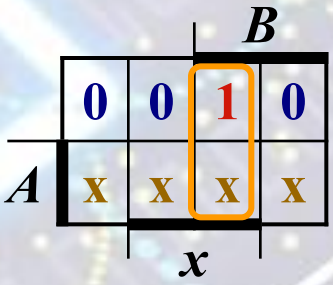


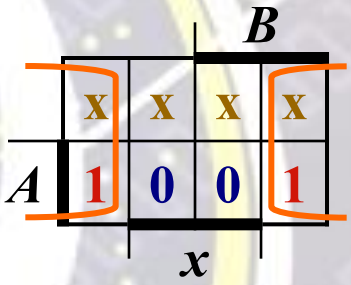
Synthesis using JK Flip-Flops

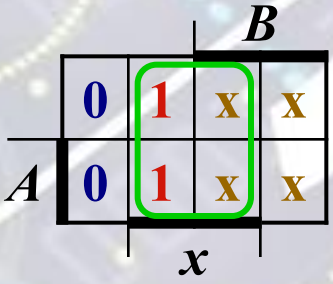
$J_A = B\,x \qquad K_A = x'$

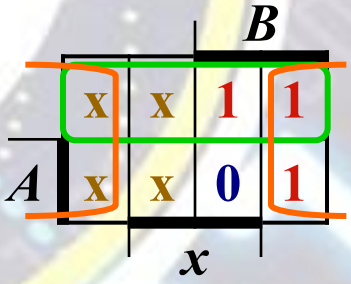
$J_B = x \qquad K_B = A' + x'$







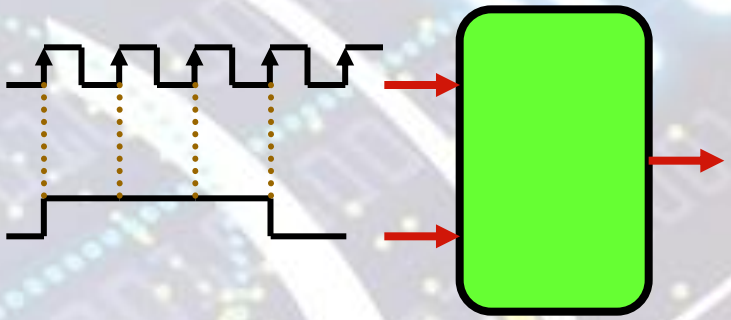




Design of Clocked Sequential Circuits with *T* F.F.

- *Example*:

Detect 3 or more consecutive 1's



Present State		Input	Next State		F.F. Input	
A	*B*	*x*	*A*	*B*	T_A	T_B
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	0	0	1
0	1	1	1	0	1	1
1	0	0	0	0	1	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	1	1	1	0	0

Synthesis using T Flip-Flops

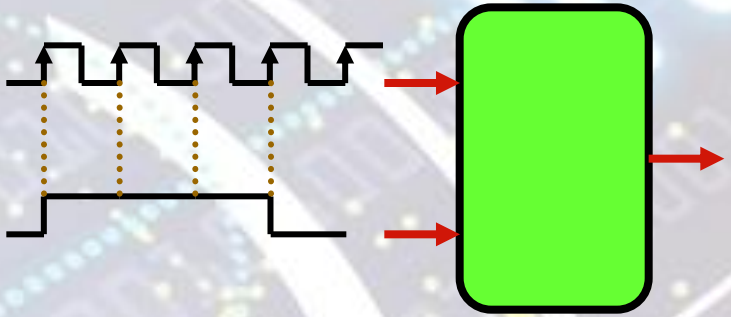
$T_A(A, B, x) = \sum (3, 4, 6)$

$T_B(A, B, x) = \sum (1, 2, 3, 5, 6)$

Design of Clocked Sequential Circuits with *T* F.F.

- *Example*:

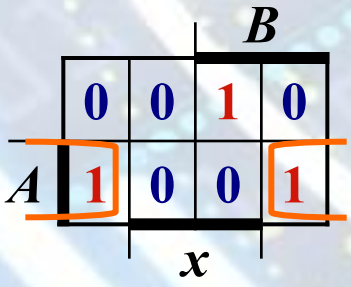
Detect 3 or more consecutive 1's

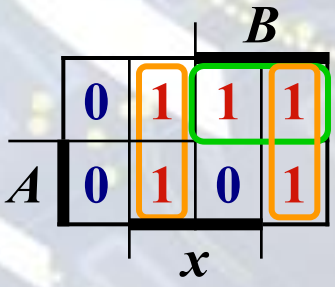


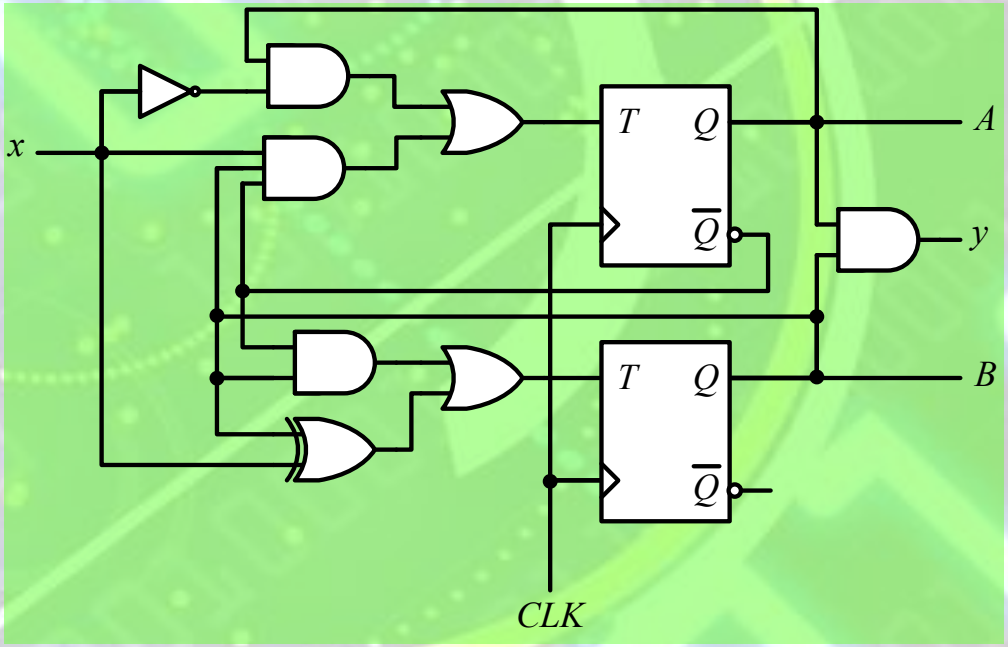
Synthesis using T Flip-Flops

$$T_A = A\,x' + A'\,B\,x$$

$$T_B = A'\,B + B \oplus x$$







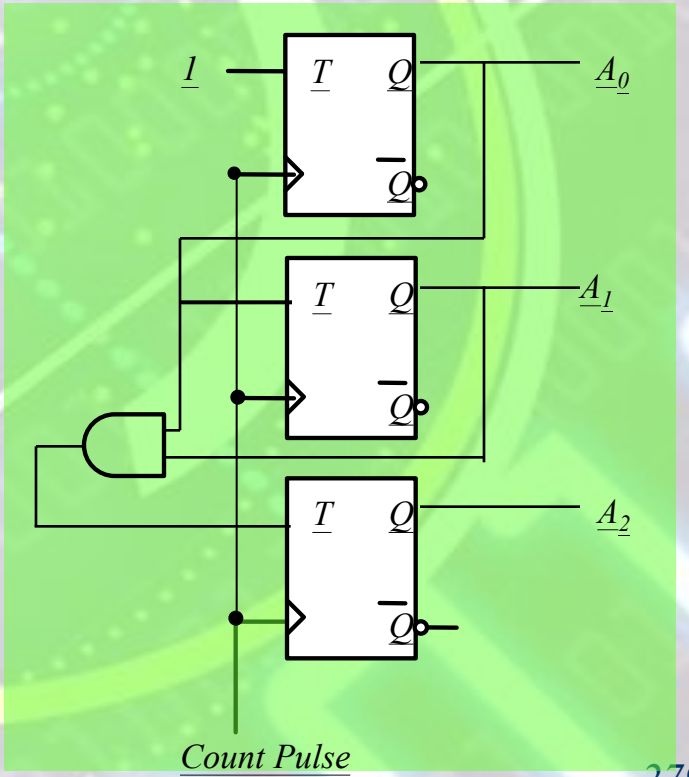
Design of Counters

Count sequence			Flip-Flop inputs		
A_2	A_1	A_0	TA_2	TA_1	TA_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	0	0	1
1	1	1	1	1	1

Synthesis using T Flip-Flops

$TA_2 = A_1 A_0$ $\quad$ $TA_1 = A_0$

$TA_0 = 1$



Design of Counters

Count sequence			Flip-Flop inputs					
A	*B*	*C*	*JA*	*KA*	*JB*	*KB*	*JC*	*KC*
0	0	0	0	X	0	X	1	X
0	0	1	0	X	1	X	X	1
0	1	0	1	X	X	1	0	X
1	0	0	X	0	0	X	1	X
1	0	1	X	0	1	X	X	1
1	1	0	X	1	X	1	0	X

Synthesis using JK Flip-Flops

$JA = B \quad KA = B$

$JB = C \quad KB = 1$

$JC = B' \quad KC = 1$

The End

Thank you

DEAR STUDENTS, AS YOU PREPARE FOR YOUR EXAMS, REMEMBER THAT YOUR WORTH IS NOT DEFINED BY A TEST SCORE. YOU ARE TALENTED, CAPABLE, AND DESTINED FOR GREATNESS. BELIEVE IN YOURSELF, GIVE IT YOUR BEST, AND SUCCESS WILL FOLLOW. GOOD LUCK!

- @yourteacher