Combinational Design

CONTENTS

- Number system in digital design
  - Number System: Binary
  - 1’s Complement
  - 2’s Complement
  - Single Precision
  - Double precision
- Boolean Algebra in digital design
- Optimization of Boolean expression using k-maps

Introduction

- Logic circuits are the basis of modern digital computer systems.
- To appreciate how computer systems operate, you will need to understand digital logic and boolean algebra.
  - Boolean functions and truth tables
  - Manipulation of boolean expressions
  - Canonical forms
  - Simplification of Boolean functions

What does this have to do with computers, anyway?

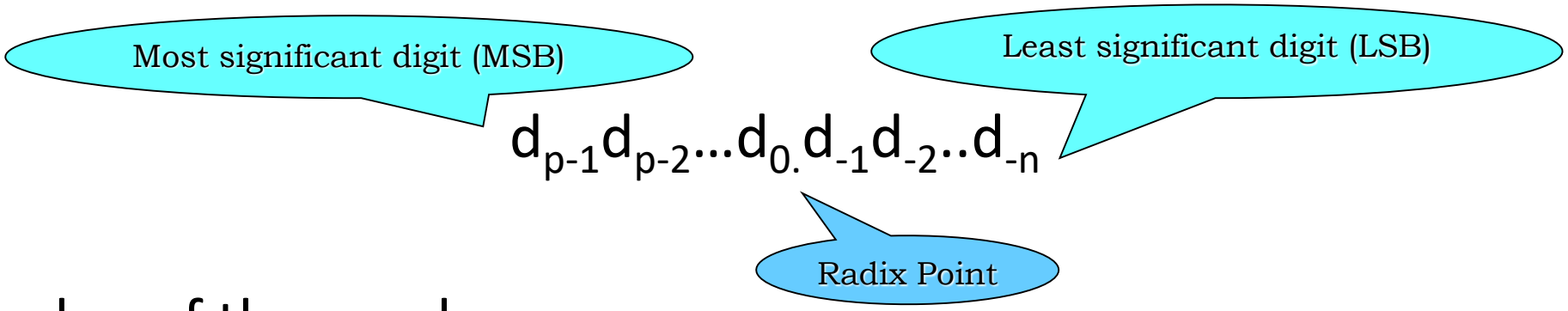
- One-to-one relationship between boolean functions and electronic circuits
- For any boolean function, we can design an electronic circuit and vice versa

Number Systems and Codes

- Positional number systems
- Positional number system conversions
- Binary system operations
- Encoding techniques

Positional Number System

- General form of a number:



- The value of the number:

$$D = \sum_{i=-n}^{p-1} d_i \cdot r^i$$

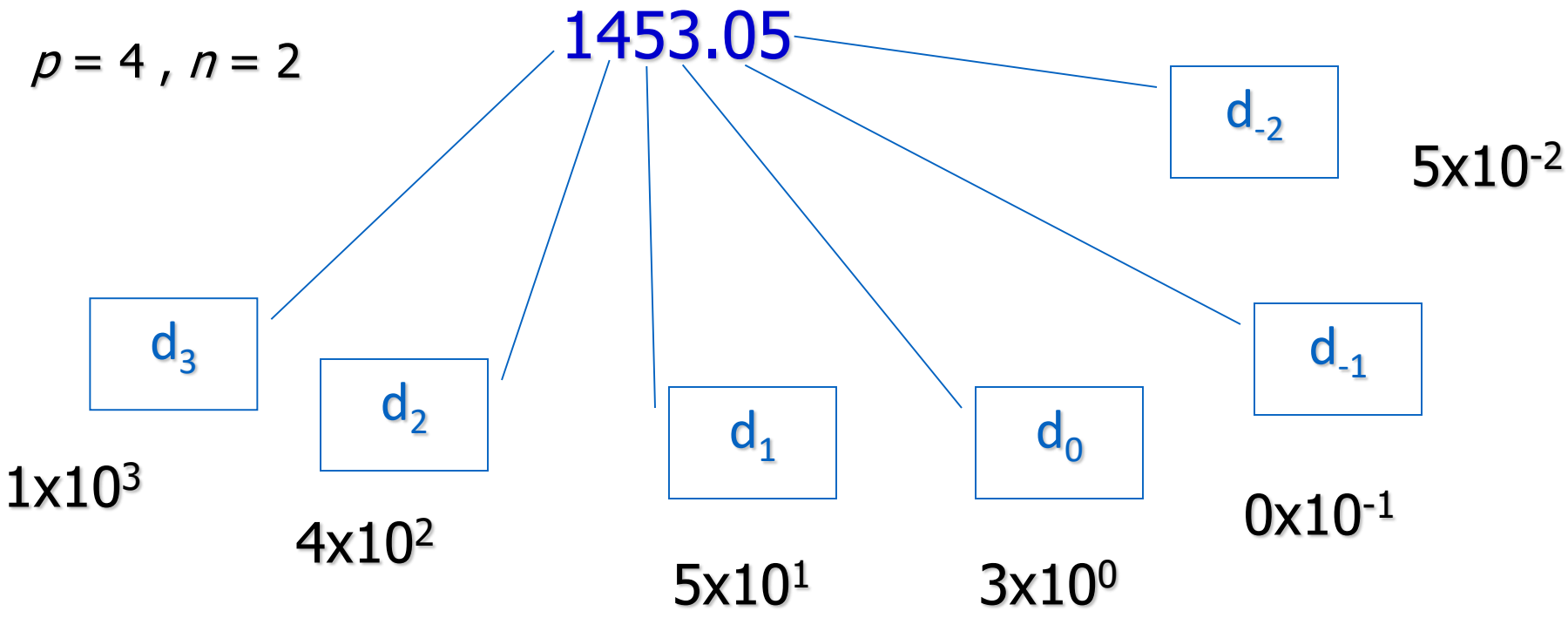
Radix or Base

- The radix or base of a number system specifies the actual number of digit included in its ordered set.
- r-radix of the number system
- d – digit in the set
- p – number of digits in integer portion
- n – number of digits in fractional portion
- d_{p-1} – MSB
- d_{-n} - LSB

Decimal System

- ***d*** **:** 0, 1, 2, . . . , 9 **$r = 10$**

$p = 4$, $n = 2$

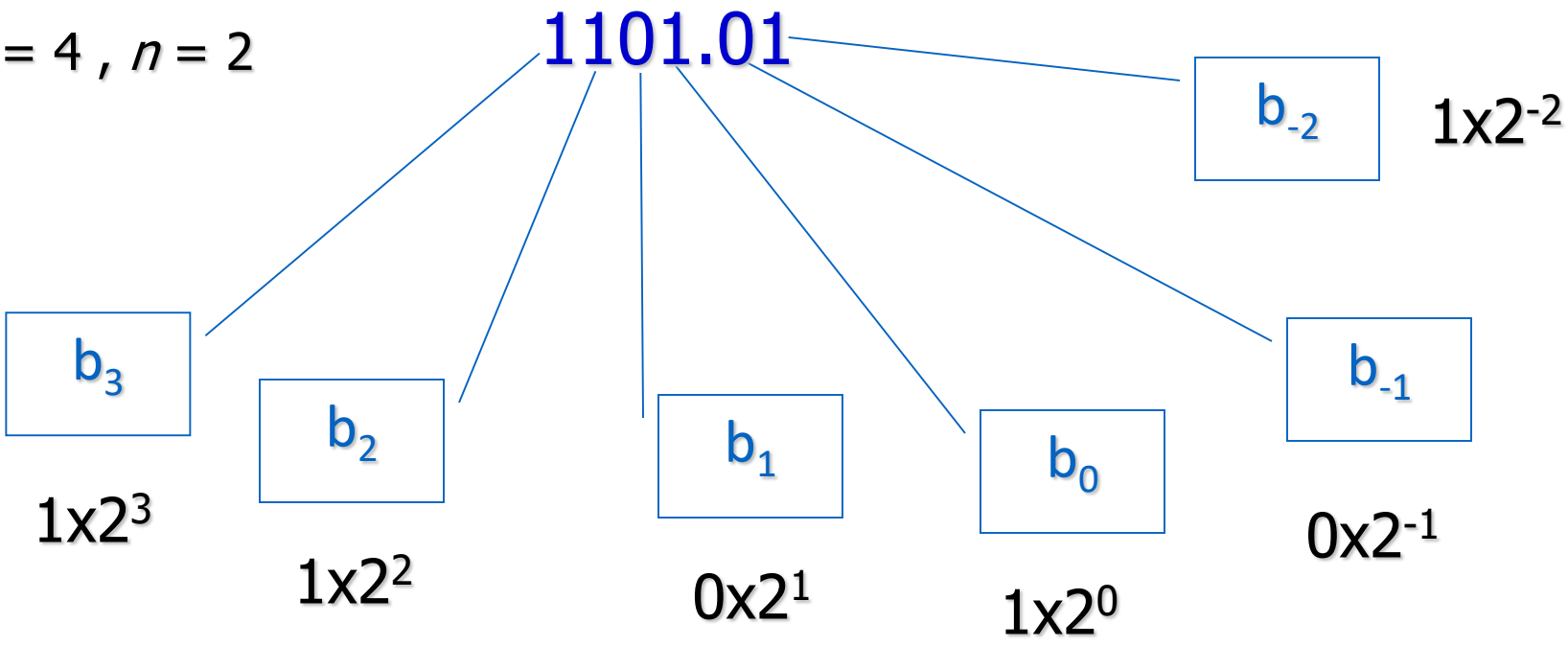


$1x10^3 + 4x10^2 + 5x10^1 + 3x10^0 + 0x10^{-1} + 5x10^{-2} = 1453.05$

Binary System

- ***b*** **:** 0 , 1 **$r = 2$**

$p = 4$, $n = 2$



$1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 + 0x2^{-1} + 1x2^{-2} = 13.25$

Exercise

- Calculate the equivalent decimal numbers:

1_2 $1x2^0 = 1$

10_2 $1x2^1 + 0x2^0 = 2 + 0 = 2$

11_2 $1x2^1 + 1x2^0 = 2 + 1 = 3$

100_2 $1x2^2 + 0x2^1 + 0x2^0 = 4 + 0 + 0 = 4$

1011_2 $1x2^3 + 0x2^2 + 1x2^1 + 1x2^0 = 8 + 0 + 2 + 1 = 11$

Octal and Hexadecimal Numbers

- **Octal** number System:

  $r = 8$ d : 0, 1, 2, … , 7

- **Hexadecimal** number System:

  $r = 16$ d : 0, 1, 2, …, 9, A, B, C, D, E, F

- Used for representations of long binary numbers

Binary, Decimal, Octal and Hex #s

Decimal	Binary	Octal	Hexadecimal
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	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

Binary – Octal Conversion

- Starting from the decimal point:
- Separate the bits into groups of three
- Replace each group with its corresponding Octal digit

$$100011110\ .\ 10101_2 =$$

$$100\ 011\ 110\ .\ 101\ 010_2 =$$

$$436\ .\ 52_8$$

Binary - Hexadecimal Conversion

- Starting from the decimal point:
- Separate the bits into groups of four
- Replace each group with its corresponding Hexadecimal digit

$$100011110\ .\ 10101_2 =$$

$$0001\ 0001\ 1110\ .\ 1010\ 1000_2 =$$

$$11E\ .\ A8_{16}$$

Octal - Binary Conversion

- Convert each Octal digit into its corresponding three bit string

$$436\ .\ 52_8 =$$

$$100\ 011\ 110\ .\ 101\ 010_2$$

Hexadecimal - Binary Conversion

- Convert each Hexadecimal digit into its corresponding four bit string

$$A5E\ .\ C8_{16} =$$

$$1010\ 0101\ 1110\ .\ 1100\ 1000_2$$

Radix-r to Decimal Conversion

$$D = \sum_{i=-n}^{p-1} d_i \cdot r^i$$

Decimal to Radix-r Conversion

- Successive division of **D** by ***r***
- The remainder of the long division will give the digits starting from the *least significant digit*

Decimal to Binary Conversion

45_{10}

45/2 = 22	The remainder is	1	**LSB**
22/2 = 11		0	
11/2 = 5		1	
5/2 = 2	1		
2/2 = 1	0		
1/2 = 0	1	**MSB**	

MSB					**LSB**
32	**16**	**8**	**4**	**2**	**1**
1	**0**	**1**	**1**	**0**	**1**

Decimal to Binary Conversion

$$0.52_{10} = (?)_2$$

Exercises

Convert 5 into binary: **1 0 1**

hexadecimal: **5**

Convert 17 into binary: **1 0 0 0 1**

hexadecimal: **11**

Convert 34 into binary: **1 0 0 0 1 0**

hexadecimal: **22**

Binary Addition

Binary addition table:

carry in	X	Y	X+Y	carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Example:

16	8	4	2	1	
1	0	1	0	1	21
0	1	1	0	0	12
1	0	0	0	0	1	33

Binary Subtraction

Binary subtraction table:

X	Y	borrow in	X-Y	borrow out
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Example:

16	8	4	2	1	
1	1	0	1	0	26
0	1	1	1	1	15
0	1	0	1	1	11

Representation of Negative Numbers in Binary Systems

- Signed-magnitude Representation
- One’s-Complement Representation
- Two’s-Complement Representation

Signed Magnitude Representation

- The MSB represents the sign bit (0 = positive, 1 = negative)
- The range for n-bit is from **$- 2^{n-1} - 1$** to **$+2^{n-1} - 1$**.
- Example: n=5, Range from -15 to 15
  - **`00000 = 0  , 10000 = -0`**
  - **`10011 = -3 , 01100 = +12`**
- Disadvantages:
  - **Complicated** digital adders
  - Two possible representations of zero

One’s-Complement Example

N = 8: form **-127**(10000000) to **127** (01111111)

$+100_{10}$ = 0 1 1 0 0 1 0 0

-100_{10} = 1 0 0 1 1 0 1 1

0_{10} = 0 0 0 0 0 0 0 0

-0_{10} = 1 1 1 1 1 1 1 1

Signed Magnitude Representation

- Addition & Subtraction Operation
- Add if signs are same
- Sub if signs are different
- **Ex: 1000_0111 + 0111_0000**

One's Complement Representation

- The MSB represents the sign bit (0 = positive, 1 = negative)
- To calculate the negative number, complement all bits of the positive number

The range for n-bit is: from $-2^{n-1} - 1$ to $+2^{n-1} - 1$.

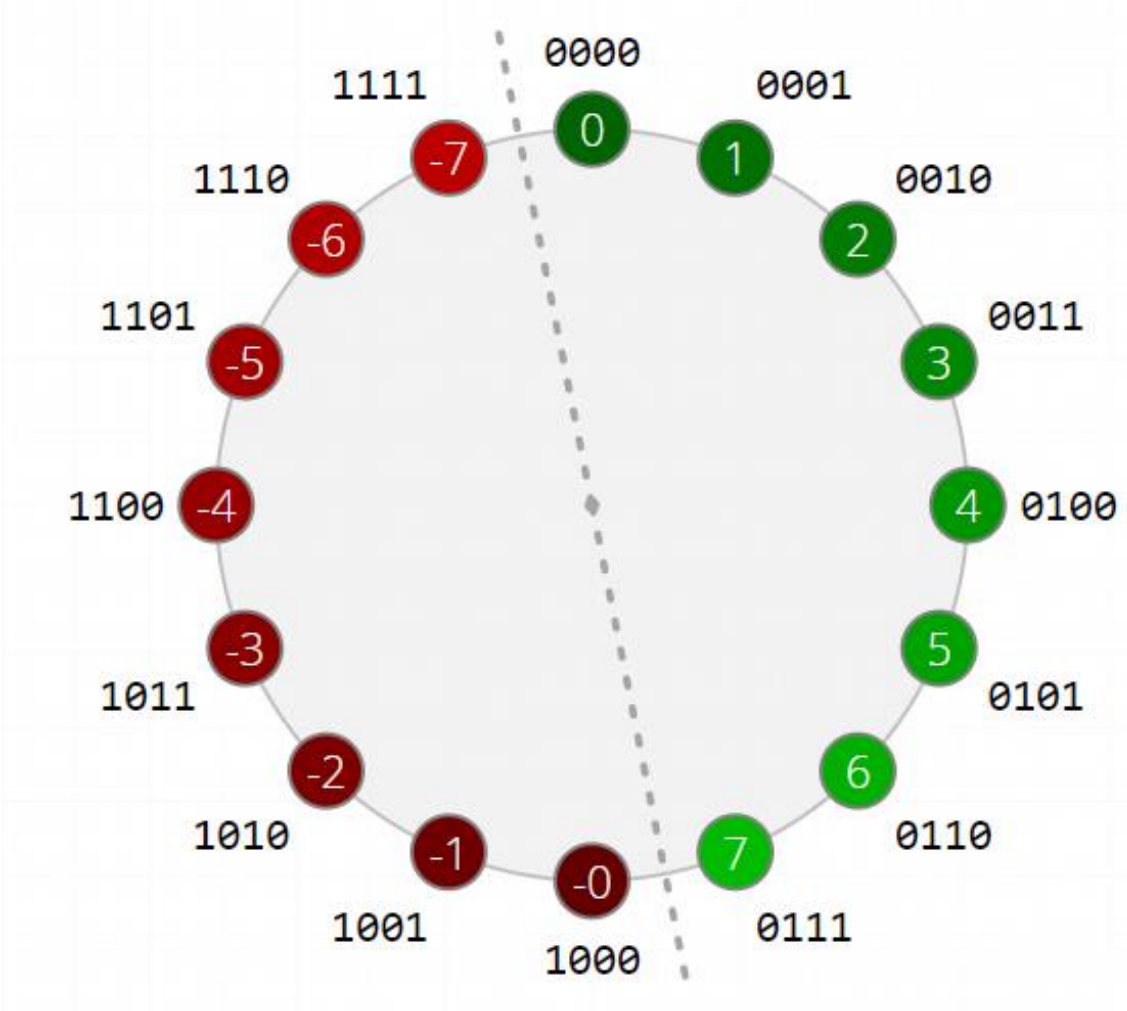
Advantages: Symmetry, ease of complementation.

Disadvantages:

- Two possible representations of zero.
- Complicated digital adders. (Ex: 7+7)

Number Circle

- Demonstrate overflow, underflow, discontinuity



Reference: https://thesis.laszlokorte.de/demo/number-circle.html

Two's Complement Representation

- The MSB represents the sign bit (0 = positive, 1 = negative)
- To calculate the negative number:
  1. **Complement all bits** of the positive number
  2. **Add 1**

The range for n-bit is: from -2^{n-1} to $+2^{n-1} - 1$.

Advantages: Addition/subtraction performed directly and only one zero

Disadvantage: One extra negative number (not symmetric)

Two's Complement Example

N = 8: from -128(10000000) to 127 (01111111)

0_{10} = 0 0 0 0 0 0 0 0

1 1 1 1 1 1 1 1 step 1

+ 1 step 2

0 0 0 0 0 0 0 0 = 0_{10}

Two's Complement Example

	64	32	16	8	4	2	1
$+100_{10}$ =	0	1	1	0	0	1	0	0
step 1	1	0	0	1	1	0	1	1
						+	1	step 2
	1	0	0	1	1	1	0	0 = -100_{10}

Comparison (4-bit)

Decimal	Signed Magnitude	One's Compl.	Two's Compl.
-8	-	-	**1000**
-7	1111	1000	1001
-6	1110	1001	1010
-5	1101	1010	1011
-4	1100	1011	1100
-3	1011	1100	1101
-2	1010	1101	1110
-1	1001	1110	1111
0	0000 or 1000	0000 or 1111	0000
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	0100	0100
5	0101	0101	0101
6	0110	0110	0110
7	0111	0111	**0111**

Exercise

What is the representation of **+11, -11** in:

5-bit **signed** magnitude representation

+11 = 0 1 0 1 1 **-11 = 1 1 0 1 1**

5-bit **one's** complement representation

+11 = 0 1 0 1 1 **-11 = 1 0 1 0 0**

5-bit **two's** complement representation

+11 = 0 1 0 1 1 **-11 = 1 0 1 0 1**

Two's Complement Addition (A+B)

1. Use binary addition rules
2. Ignore any carry beyond the sign bit
   - If the range is not exceeded, addition result will be correct including the sign bit.

Examples:

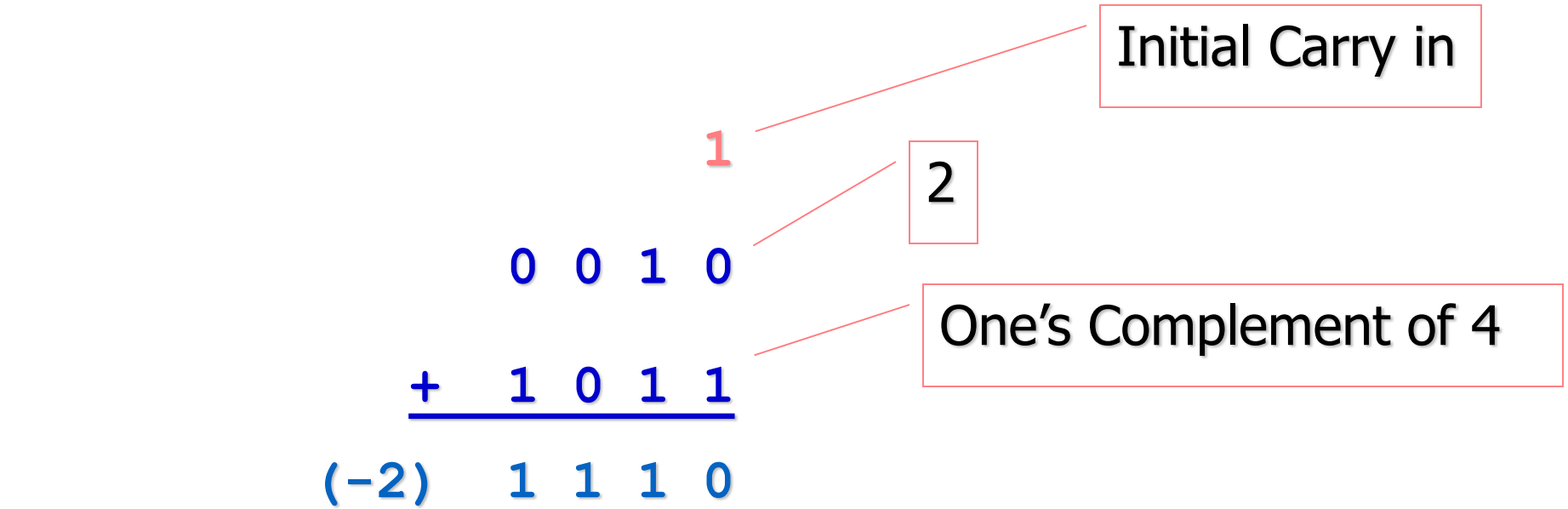
```
  (-2)  1 1 1 0          (-3)  1 1 0 1
+ (-4)  1 1 0 0        + (+3)  0 0 1 1
-------------------    -------------------
 (-6) 1 1 0 1 0         (0)  1 0 0 0 0
```

Two's Complement Subtraction Method : (A+(-B))

- Add A to the Two's complement of B:

1. Take the One's complement of B
2. Add it to A with initial carry-in i.e. 1

Example: 2-4



Two's Complement Multiplication Method : (A*(-B))

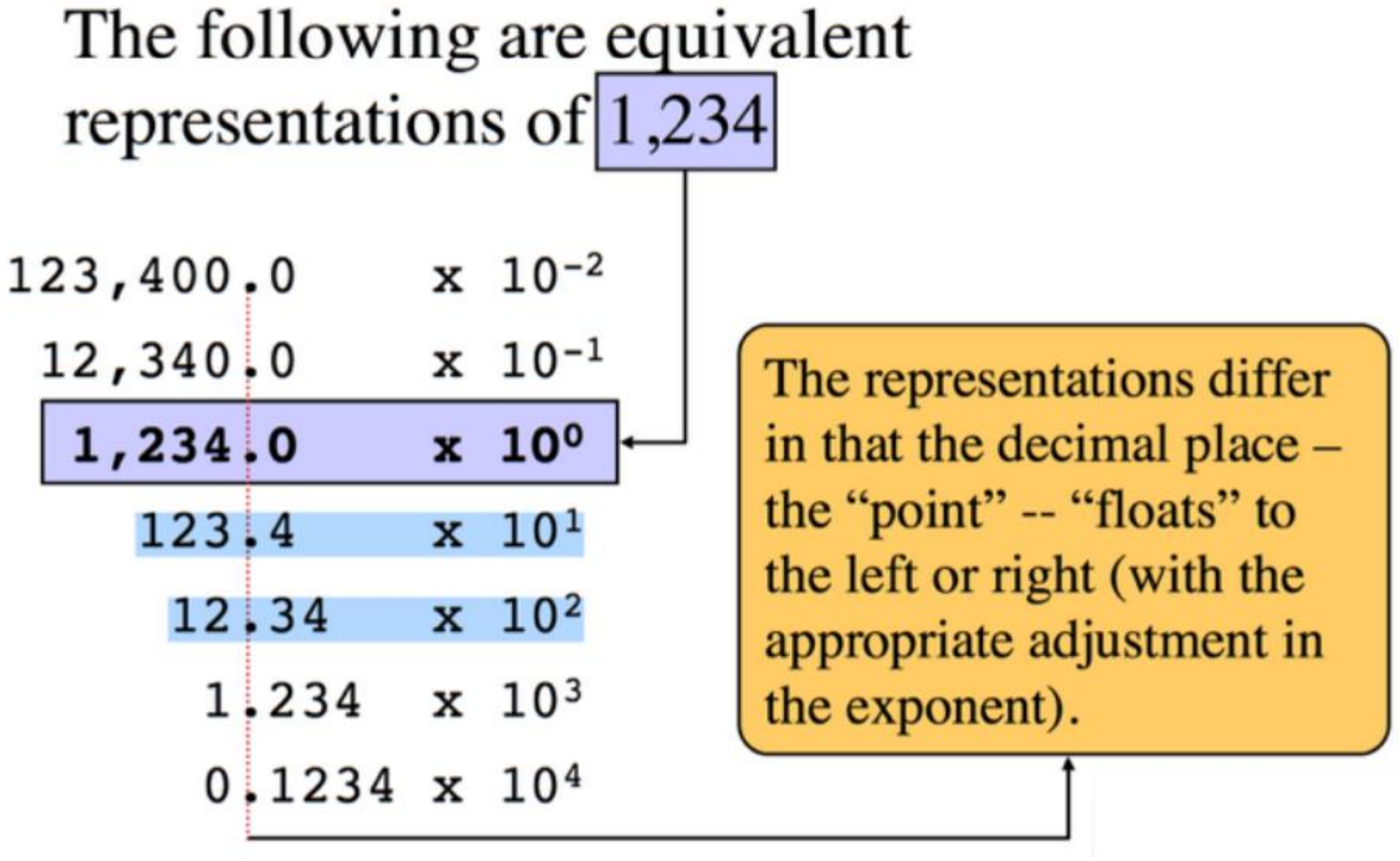
Assignment - 1

- **Convert 3332 represented in Quaternary (base 4) to decimal**
- **Convert 96 from decimal to Quaternary (base 4)**
- **Compute single precision floating point for the following**
  - **4.25**
  - **-4.25**
- **Multiply 3 X -3, -2 X 3**
- **Multiply the following 2's complement numbers**
  - **1101 X 0101**
  - **0101 X 1001**
  - **1101 X 1010**

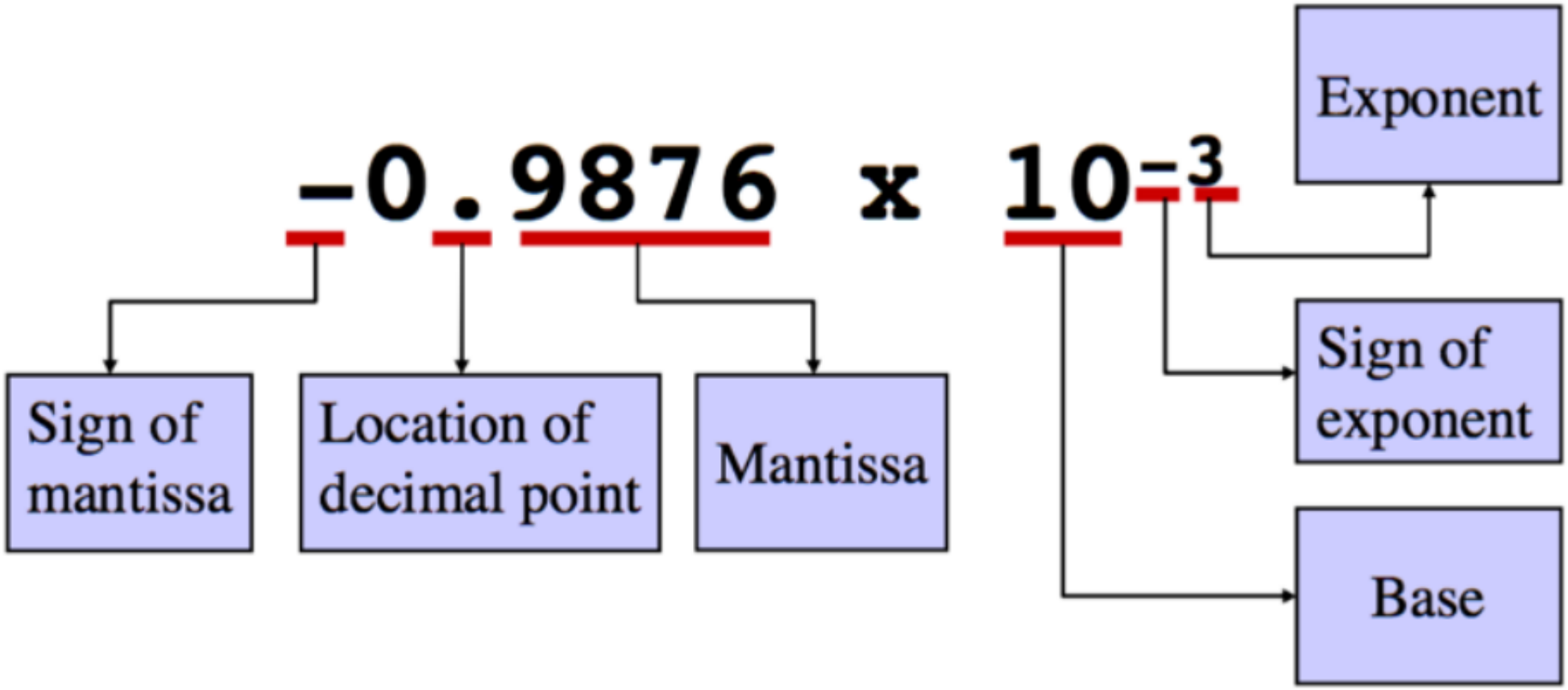
Fixed Point Numbers

- No. of bits = 16
- No decimal point
  - Range is -32768 to 32767 (or 0 to 65536)
- Decimal point after 3 bits
  - Range is -4096 to 4095 (or 0 to 8192) and 3 bits for fraction
- Decimal point after 8 bits
  - Range is -256 to 255 (or 0 to 512) and 7 bits for fraction
- Limited precision and range – not sufficient to represent say weight of the earth = 5.972×10^{24} kg , electron mass = $9.1093837015 \times 10^{-31}$ kg

Floating Point Numbers



Floating Point Numbers



Floating Point Numbers

- A floating-point number is represented by the triple
  - *S* is the Sign bit (0 is positive and 1 is negative)
    - Representation is called sign and magnitude
  - *E* is the Exponent field (signed)
    - Very large numbers have large positive exponents
    - Very small close-to-zero numbers have negative exponents
    - More bits in exponent field increases range of values
  - *F* is the Fraction field (fraction after binary point)
    - More bits in fraction field improves the precision of FP numbers

$$\text{Value of a floating-point number} = (-1)^{S} \times \text{val}(F) \times 2^{\text{val}(E)}$$

Floating Point Numbers

- Single Precision Floating Point Numbers (32 bits)
  - 1-bit sign + 8-bit exponent + 23-bit fraction

- Double Precision Floating Point Numbers (64 bits)
  - 1-bit sign + 11-bit exponent + 52-bit fraction

Floating Point Numbers

- Normalized Floating Point

❖ For a normalized floating point number (S, E, F)

❖ Significand is equal to $(1.F)_2 = (1.f_1f_2f_3f_4\ldots)_2$

◇ IEEE 754 assumes hidden 1. (not stored) for normalized numbers

◇ Significand is 1 bit longer than fraction

❖ Value of a Normalized Floating Point Number is

$$(-1)^S \times (1.F)_2 \times 2^{\text{val}(E)}$$

$$(-1)^S \times (1.f_1f_2f_3f_4\ldots)_2 \times 2^{\text{val}(E)}$$

$$(-1)^S \times (1 + f_1\times2^{-1} + f_2\times2^{-2} + f_3\times2^{-3} + f_4\times2^{-4}\ldots)_2 \times 2^{\text{val}(E)}$$

Floating Point Numbers

- Biased Exponent Representation
  - How to represent a signed exponent? Choices are
    - Sign + magnitude representation for the exponent
    - Two's complement representation
    - Biased representation
  - IEEE 754 uses biased representation for the exponent
    - Value of exponent = val(E) = E – Bias (Bias is a constant)
  - Recall that exponent field is 8 bits for single precision
    - E can be in the range 0 to 255
    - E = 0 and E = 255 are reserved for special use
    - E = 1 to 254 are used for normalized floating point numbers
    - Bias = 127 (half of 254), val(E) = E – 127
    - val(E=1) = –126, val(E=127) = 0, val(E=254) = 127

Floating Point Numbers

- Biased Exponent Representation
  - For double precision, exponent field is 11 bits
    - E can be in the range 0 to 2047
    - $E = 0$ and $E = 2047$ are reserved for special use
    - $E = 1$ to 2046 are used for normalized floating point numbers
    - Bias = 1023 (half of 2046), val(E) = $E - 1023$
    - val(E=1) = –1022, val(E=1023) = 0, val(E=2046) = 1023
  - Value of a Normalized Floating Point Number is

$$(-1)^S \times (1.F)_2 \times 2^{E-\text{Bias}}$$

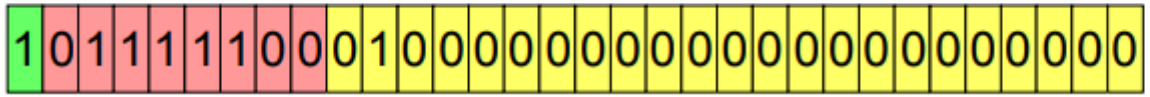
$$(-1)^S \times (1.f_1f_2f_3f_4\ldots)_2 \times 2^{E-\text{Bias}}$$

$$(-1)^S \times (1 + f_1\times 2^{-1} + f_2\times 2^{-2} + f_3\times 2^{-3} + f_4\times 2^{-4}\ldots)_2 \times 2^{E-\text{Bias}}$$

Floating Point Numbers

- Biased Exponent Representation

❖ What is the decimal value of this Single Precision float?



❖ Solution:

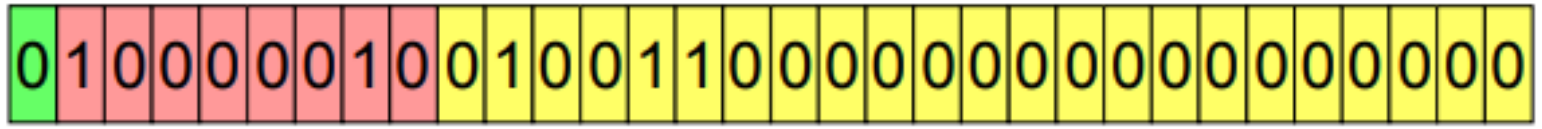
◇ Sign = 1 is negative

◇ Exponent = $(01111100)_2$ = 124, *E* – bias = 124 – 127 = –3

◇ Significand = $(1.0100 \ldots 0)_2 = 1 + 2^{-2} = 1.25$ (1. is implicit)

◇ Value in decimal = $-1.25 \times 2^{-3} = -0.15625$

❖ What is the decimal value of?



Floating Point Numbers

- Example - 1
- Real number -> 54.25 = 32+16+4+2+.(1/4)
- Binary -> 110110.01
- Normalized -> 1. 1011001 x 2^5
- Mantissa -> 10110010000000000000000
- Exponent -> 127 + 5 = 132 = 10000100
- Floating Point Number -> 0 10000100 10110010000000000000000

  -> 0100_0010_0101_1001_0000_0000_0000_0000

  -> 0x42590000

Floating Point Numbers

- Example - 2
- Real number -> -0.6875 = 0.5+0.125+0.0625 = (1/2 + 1/8 + 1/16)
- Binary -> 0.1011
- Normalized -> 1. 011 x 2^1
- Mantissa -> 01100000000000000000000
- Exponent -> 127 + 1 = 128 = 1000_0000
- Floating Point Number -> 1 10000000 01100000000000000000000

  -> 1100_0000_0011_0000_0000_0000_0000_0000

  -> 0xB0300000

Floating Point Numbers

- Example - 3
- Real number -> -54.6875
- Binary ->
- Normalized ->
- Mantissa ->
- Exponent ->
- Floating Point Number ->

Floating Point Numbers

- Example – 4 – Floating point to decimal conversion
- Floating point number -> 0xc2630000

  -> 1100 0010 0110 0011 0000 0000 0000 0000
- Sign -> 1
- Mantissa -> 11000110000000000000000
- Exponent -> 10000100 = 132 , Actual value -> 132-127=5
- Binary -> 1.1100011×2^5

  -> 111000.11 = 56.75
- Decimal Number -> -56.75

Floating Point Numbers

- Arithmetic Operation – add/sub

1. Align the exponents (smaller to larger)
2. Add or subtract significands
3. Normalize, overflow and underflow checks
4. Rounding
5. Stop if normalized else go to 3rd step

- Examples

Floating Point Numbers

- Arithmetic Operation – multiplication

1. Add exponents & Subtract bias
2. Multiply significands
3. Normalize, overflow and underflow checks
4. Rounding
5. Stop if normalized else go to 3rd step
6. Set sign bit

- Examples

Coding

- **Coding**: Representing a set of objects by a set of strings.
- **Code**: The set of bit strings.
- **Code Word**: A particular bit string in the **Code.**

Examples:

- Data Objects: Decimal Numbers, Characters.
- Non-data Objects: Machine states, Control Actions.

Binary Codes For Decimal Numbers

- To represent the 10 decimal digits, we need 4 bits.

Examples:

- 4 bits Codes
  1. BCD (8421) - Binary Coded Decimal
  2. 2421
  3. Excess-3

Decimal Codes

Decimal	BCD (8421)	BCD 2421	Excess-3
0	0000	0000	0011
1	0001	0001	0100
2	0010	0010	0101
3	0011	0011	0110
4	0100	0100	0111
5	0101	1011	1000
6	0110	1100	1001
7	0111	1101	1010
8	1000	1110	1011
9	1001	1111	1100

Exercise

- Codes for **517**

BCD:	0101 0001 0111
2421:	1011 0001 1101
Excess-3:	1000 0100 1010

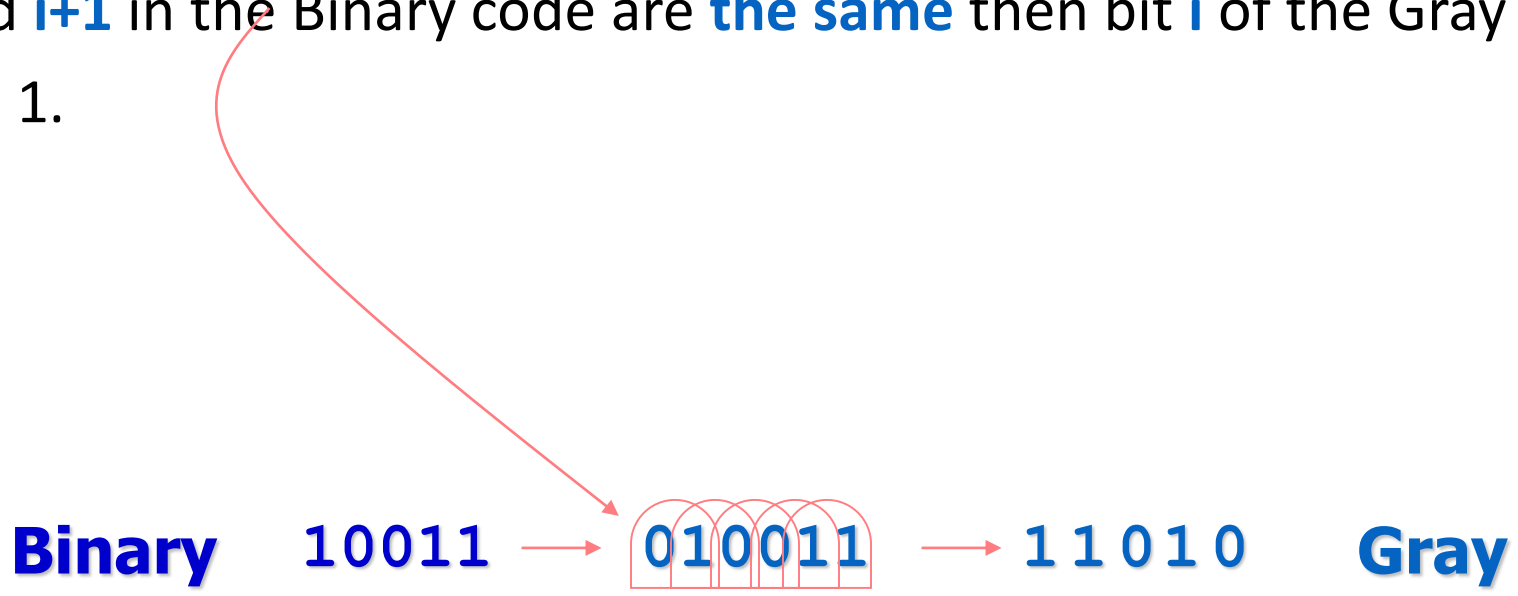
Gray Code

- One bit changes between two successive code words
- Binary Code and Gray Code (n = 3):

Decimal	Binary Code	Gray Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

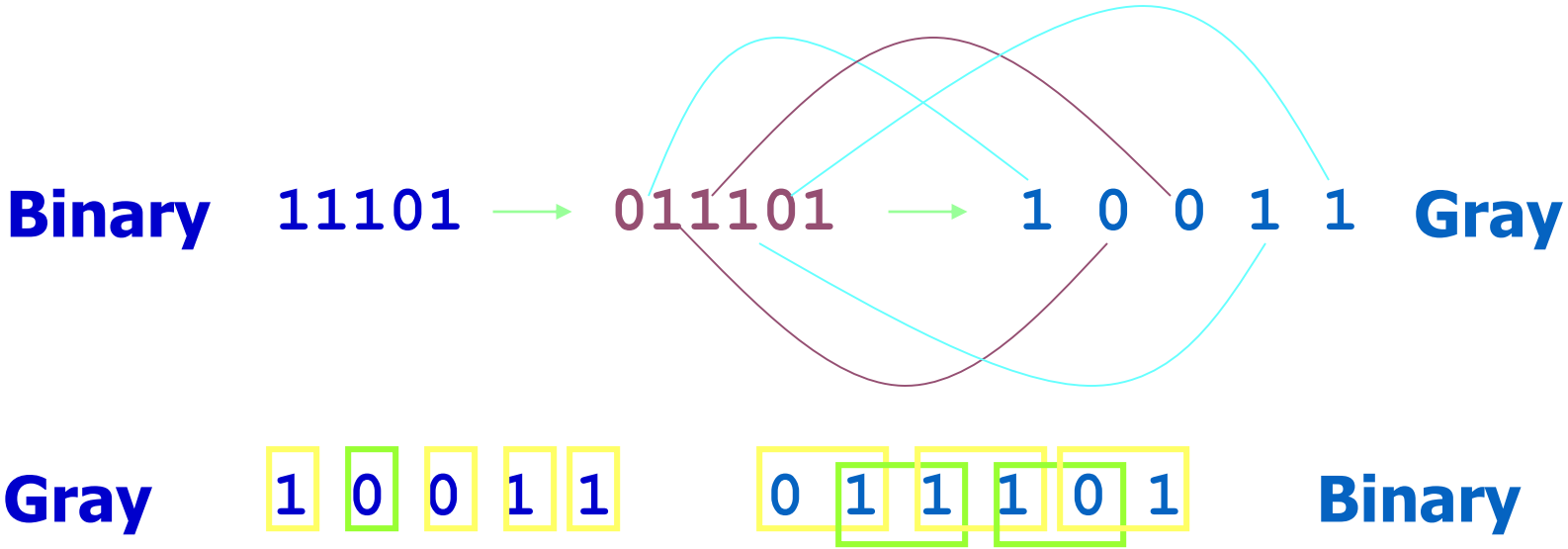
Binary Code to Gray Code Conversion

- Convert **n** bit Binary code into **n** bit Gray Code:
- Bit **i** of the Gray Code is obtained by comparing bits **i** and **i+1** of the Binary Code:
  - Add extra 0 to the left of the Binary code
  - If bits **i** and **i+1** in the Binary code are **the same** then bit **i** of the Gray code is **0**, else bit i is 1.
- Example:



Exercise

i and i+1: **same → 0;** else, 1



Coding for Error Control

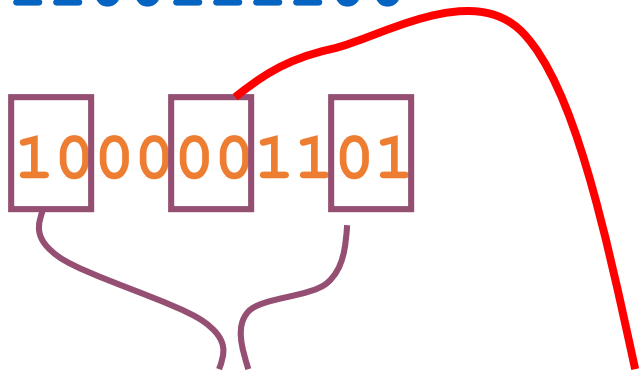
- **Bit error** occurs due to noise, etc., if
  - 0 sent is received as 1 or
  - 1 sent is received as 0
- **EDC** (error detecting codes) to detect bit errors
- **ECC** (error correcting codes) to correct bit errors

EDC: Repetition-2 Code

To send: 10110

send: 1100111100

If we receive 1000001101



we detect two errors but miss one

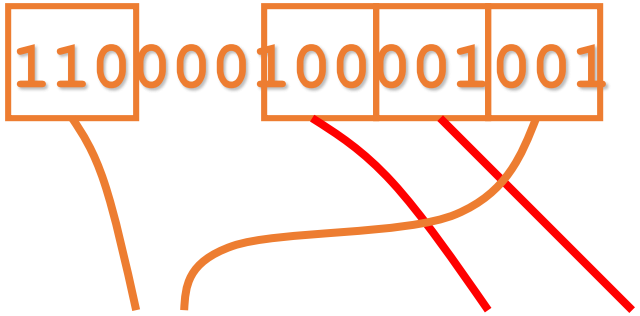
- Detects any one bit error in a 2-bit block, but will miss two

ECC: Repetition-3 Code

To send: **10110**

send: **111000111111000**

If we receive



we correct two and miss two.

- Detects any one bit error in a 3-bit block, but will miss any two or three

Parity-bit EDCs

Sender

To send ***m*** bits, $b_1, b_2, \ldots, b_m$.

Send ***m+1*** bits, $b_1, b_2, \ldots, b_m, p$, where $p = b_1 \oplus b_2 \oplus \ldots \oplus b_m$

$(b_1 \oplus b_2 \oplus \ldots \oplus b_m \oplus p = 0)$

Receiver

if **s=0**, then **0** (or 2 or 4 or …) bit errors occurred, no error.

if **s=1**, then **1** (or 3 or 5 or …) bit errors occurred.

compute $s = r_1 \oplus r_2 \oplus \ldots \oplus r_m \oplus r_{m+1}$

When $r_1, r_2, \ldots, r_m, r_{m+1}$ is received

$b_1, b_2, \ldots, b_m, p$

Information Bits	*Even-parity Code*	*Odd-parity Code*
000	000 0	000 1
001	001 1	001 0
010	010 1	010 0
011	011 0	011 1
100	100 1	100 0
101	101 0	101 1
110	110 0	110 1
111	111 1	111 0

Table 2-13 Distance-2 codes with three information bits.

Summary

- Binary forms used in digital systems
  - Positive/Negative numbers, addition/subtraction
  - Arithmetic operations performed directly on negative numbers represented in Two's complement.
  - Floating point numbers
- Hexadecimal numbers are used for shorthand representation of binary numbers.
- Decimal numbers, characters, and actions encoded into binary strings.
- Error-detecting and error-correcting codes use extra bits.

Combinational Logic Design Principles

- Boolean (Switching) Algebra
- Combinational Circuit Analysis
- Combinational Circuit Synthesis

Boolean (Switching) Algebra

Boolean ('Switching') Algebra

- **Boolean values**: 0, 1
- **Positive-logic convention**: analog voltages LOW, HIGH → 0, 1
- **Signal values**: denoted by variables (**X, Y,** etc.)
- **Complement**: **X′** (opposite of **X**)
- **AND**: **X · Y**
- **OR**: **X + Y**
- **Literal**: a variable or its complement: **X, X′**
- **Expression**: literals combined by AND, OR, parentheses

$$(A \cdot B' \cdot C + Q5) \cdot RESET'$$

- **Equation**: Variable = expression

$$P = ((FRANK \cdot Z') + (A \cdot B' \cdot C + Q5) \cdot RESET'$$

Basic Axioms

- A variable can take only one of two values (**0,1**)
  (A1) $\mathbf{X = 0 \text{ if } X \neq 1}$ (A1') $\mathbf{X = 1 \text{ if } X \neq 0}$

- **NOT** operation (The complement Operation):
  (A2) $\mathbf{\text{If } X = 0 \text{ then } X \neq 1}$ (A2') $\mathbf{\text{If } X = 1 \text{ then } X \neq 0}$

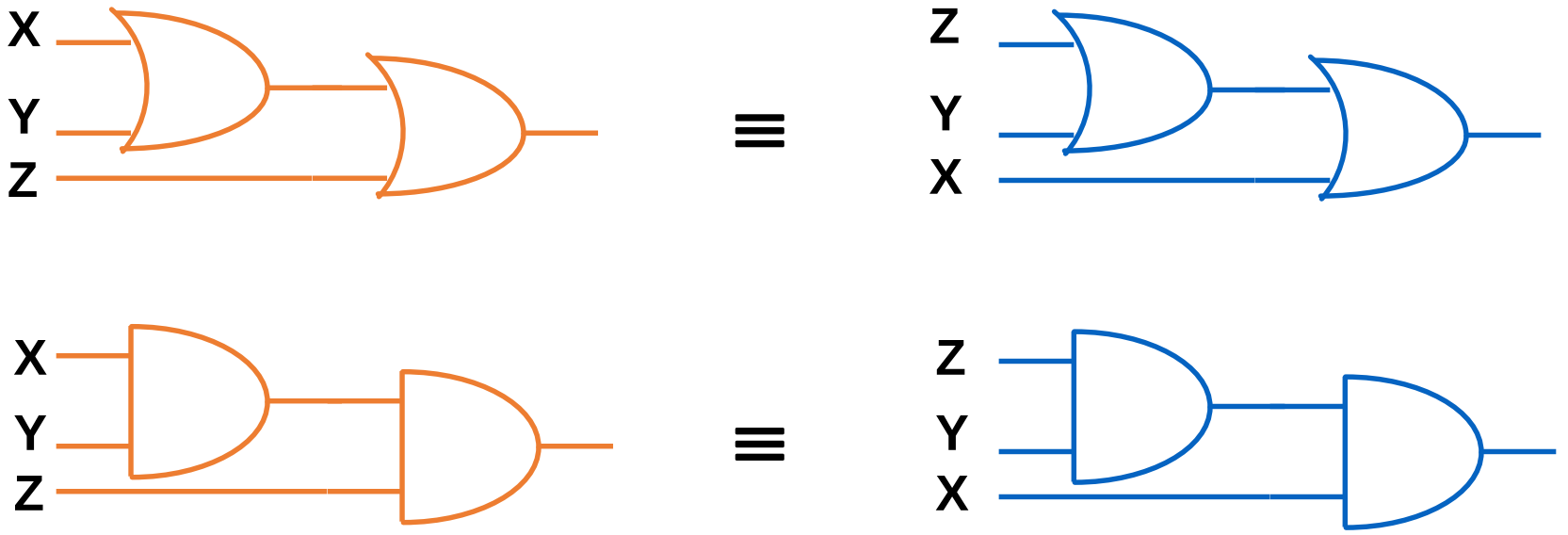
- **AND** and **OR** operations (Multiplication and Addition):
  (A3) $\mathbf{0 \cdot 0 = 0}$ (A3') $\mathbf{0 + 0 = 0}$
  (A4) $\mathbf{1 \cdot 1 = 1}$ (A4') $\mathbf{1 + 1 = 1}$
  (A5) $\mathbf{0 \cdot 1 = 1 \cdot 0 = 0}$ (A5') $\mathbf{1 + 0 = 0 + 1 = 1}$

Theorems - Single Variable

- **Identity elements**: (T1) $X + 0 = X$ (T1') $X \cdot 1 = X$
- **Null elements**: (T2) $X + 1 = 1$ (T2') $X \cdot 0 = 0$
- **Idempotency**: (T3) $X + X = X$ (T3') $X \cdot X = X$
- **Involution**: (T4) $(X')' = X$
- **Complements**: (T5) $X + X' = 1$ (T5') $X \cdot X' = 0$

Theorems-Multiple Variables

- **Commutativity**: (T6) $X + Y = Y + X$ (T6') $X \cdot Y = Y \cdot X$
  - The inputs of AND and OR gates can be interchanged.
- **Associativity**: (T7) $(X + Y) + Z = X + (Y + Z)$ (T7') $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$
  - The order of the input variables could be rearranged.



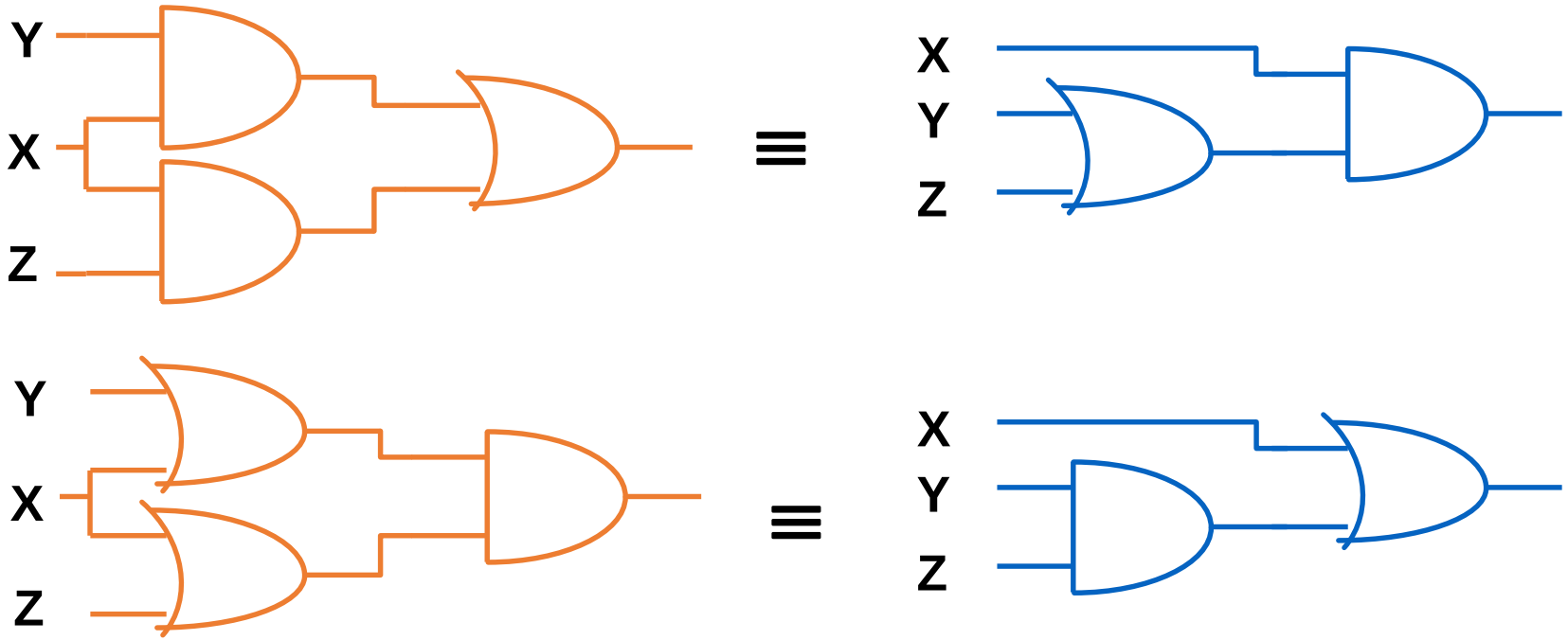
Theorems - Multiple Variables

- **Distributivity**:

(T8) $X \cdot (Y + Z) = X \cdot Y + X \cdot Z$

(T8') $X + Y \cdot Z = (X + Y) \cdot (X + Z)$

  - Multiplication distributes over addition; Addition distributes over multiplication!



Theorems-Multiple Variables

- **Covering:** (T9) $X + X \cdot Y = X$ (T9') $X \cdot (X + Y) = X$

Proof:

(T9): $X + X \cdot Y = X \cdot 1 + X \cdot Y$ (theorem T1')
$= X \cdot (1 + Y)$ (theorem T8- Distributivity)
$= X \cdot 1$ (theorem T2)
$= X$ (theorem T1')

(T9'): $X \cdot (X + Y) = (X + 0) \cdot (X + Y)$ (theorem T1)
$= X + (0 \cdot Y)$ (theorem T8'- Distributivity)
$= X + 0$ (theorem T2')
$= X$ (theorem T1)

Theorems-Multiple Variables

- **Combining:** (T10) $X \cdot Y + X \cdot Y' = X$ (T10') $(X + Y) \cdot (X + Y') = X$

Proof:

(T10)	$X = X \cdot 1$	(theorem T1')
	$= X \cdot (Y + Y')$	(theorem T5)
	$= X \cdot Y + X \cdot Y'$	(theorem T8 - Distributivity)
(T10')	$X = X + 0$	(theorem T1)
	$= X + (Y \cdot Y')$	(theorem T5')
	$=(X + Y) \cdot (X + Y')$	(theorem T8' - Distributivity)

Covering and combining are used in minimizing logic functions.

Theorems-Multiple Variables

- **Consensus:**

(T11) $X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$

(T11') $(X + Y) \cdot (X' + Z) \cdot (Y + Z) = (X + Y) \cdot (X' + Z)$

- **Generalized Idempotency:**

(T12) $X + X + \ldots + X = X$

(T12') $X \cdot X \cdot \ldots \cdot X = X$

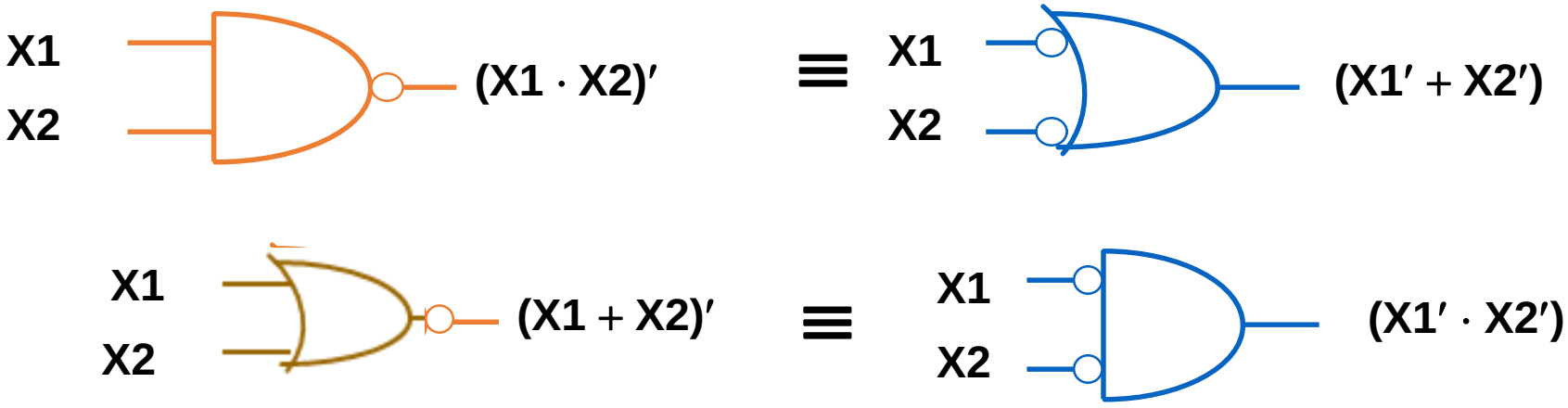
Theorems-Multiple Variables

- **DeMorgans Theorems**

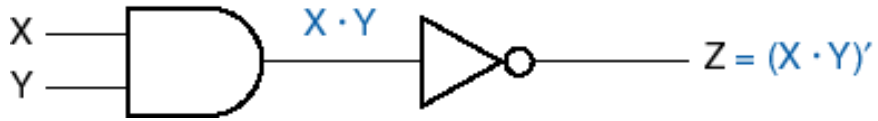
(T13) $(X1 \cdot X2 \cdot \ldots \cdot Xn)' = X1' + X2' + \ldots + Xn'$

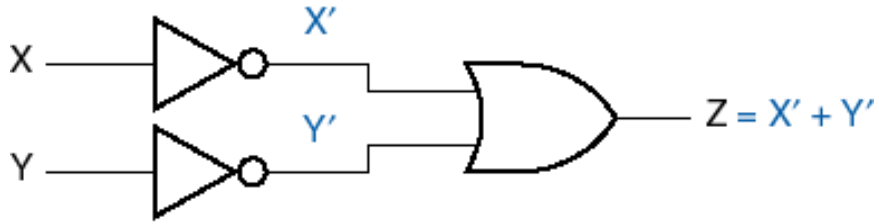
(T13') $(X1 + X2 + \ldots + Xn)' = X1' \cdot X2' \cdot \ldots \cdot Xn'$

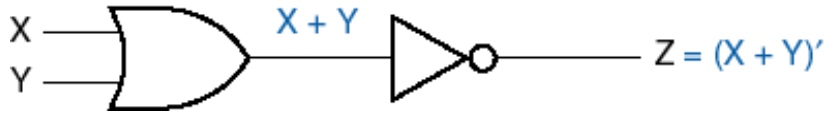
Example: two-variable case

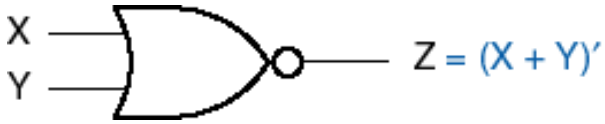


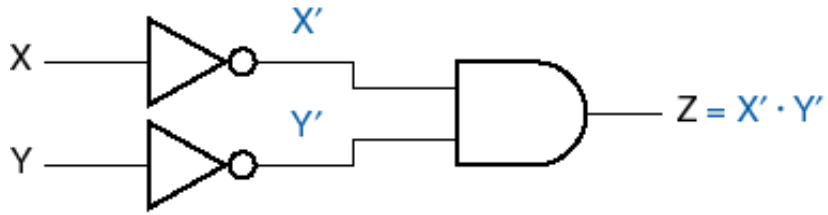
DeMorgan Symbol Equivalence

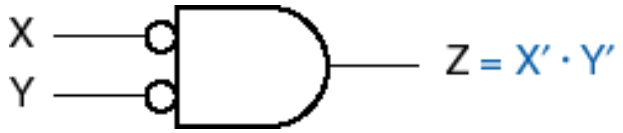




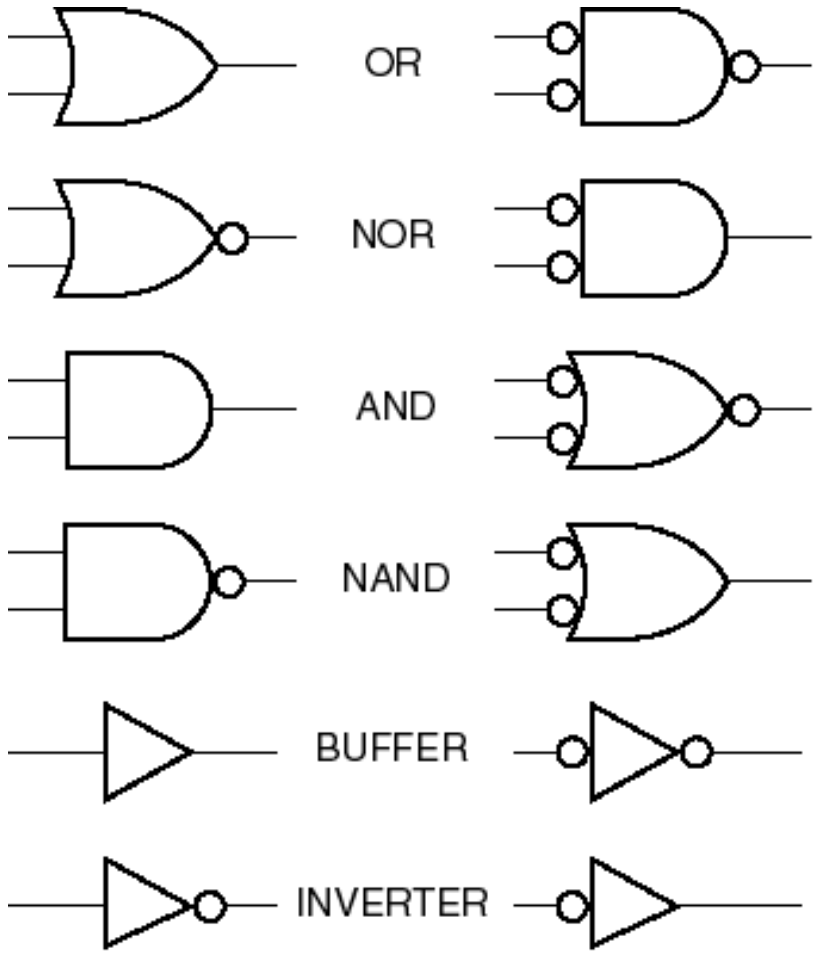








DeMorgan Symbols

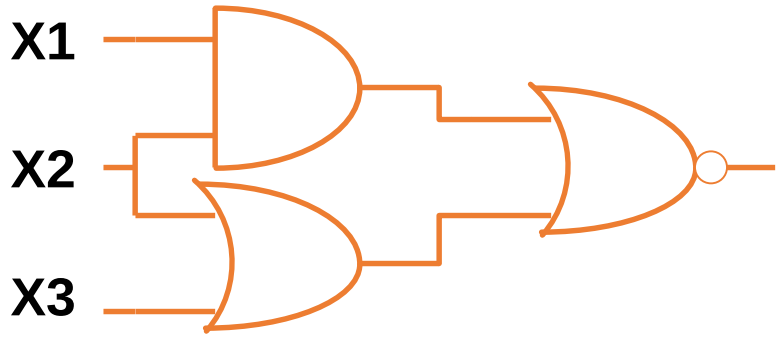


Generalized DeMorgan's Theorem

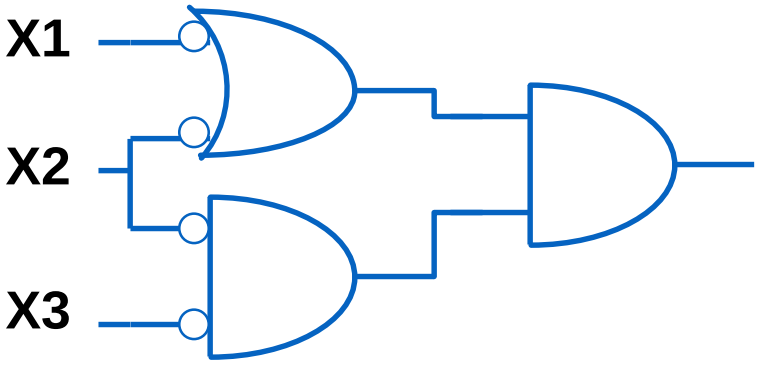
(T14) $[\ F\ (X1, X2, \ldots, Xn, +, \cdot)\ ]' = F\ (X1', X2', \ldots, Xn', \cdot, +)$

Example: $F = (X1 \cdot X2) + (X2 + X3)$

$F' = [(X1 \cdot X2) + (X2 + X3)]'$



$F' = (X1' + X2') \cdot (X2' \cdot X3')$



Duality

- **Swap 0 & 1, AND & OR**
  - **Result: Theorems still true**
- Why?
  - Each axiom (A1-A5) has a dual (A1'-A5')
- Counterexample:

  $X + X \cdot Y = X$ (T9)

  $X \cdot X + Y = X$ (dual)

- **Mathematical definition :** F is a Boolean Function; FD the dual function is:
  - $\mathbf{FD\ (X1, X2, \ldots, Xn, +, \cdot, ')\ =\ F\ (X1, X2, \ldots, Xn, \cdot, +, ')}$

Duality

Example:

- $F_{X1,X2,X3} = X1 + X2 \cdot X3$
- $FD_{X1,X2,X3} = X1 \cdot (X2 + X3)$
- $FD'_{X1,X2,X3} = (X1 \cdot (X2 + X3))' = X1' + X2' \cdot X3'$

  $= F_{X1', X2', X3'}$
- $FD_{X1',X2',X3'} = X1' \cdot (X2' + X3')$
- $FD'_{X1',X2',X3'} = (X1' \cdot (X2' + X3'))' = X1'' + X2'' \cdot X3''$

  $= X1 + X2 \cdot X3$

$$F_{X1,X2,X3} = FD'_{X1',X2',X3'}$$

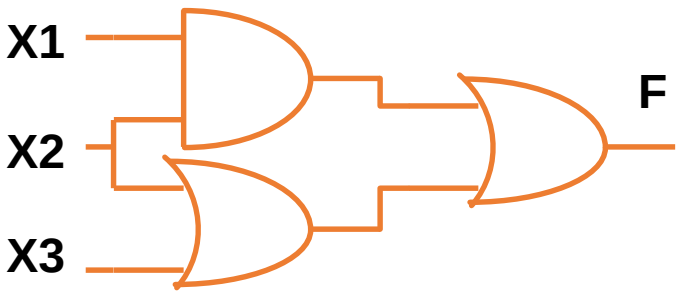
Duality and DeMorgan's Theorem

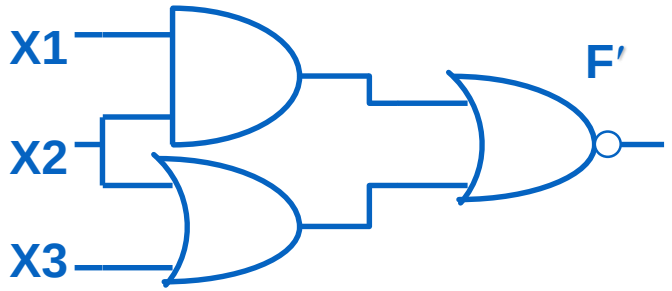
$$F_{X1,X2,\ldots,Xn} = FD'_{X1',X2',\ldots,Xn'}$$

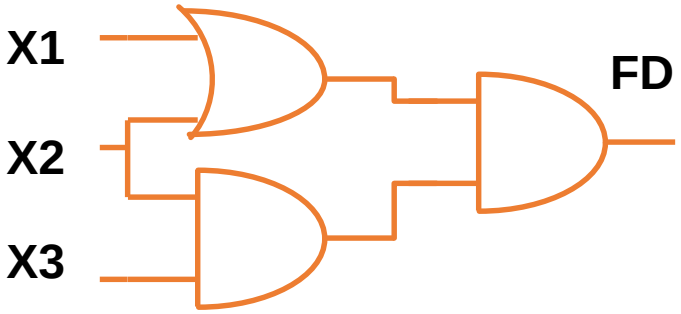
or:

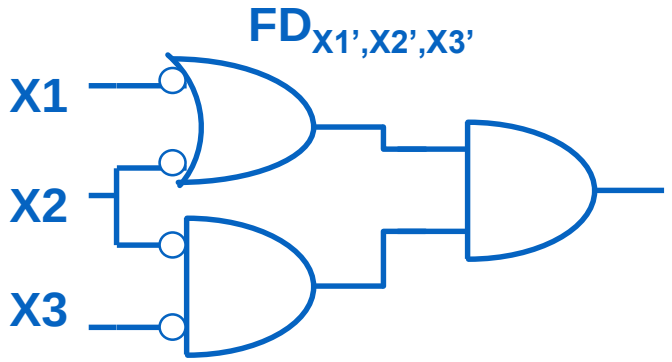
$$F'_{X1,X2,\ldots,Xn} = FD_{X1',X2',\ldots,Xn'}$$

Example:









Exercise: DeMorgan/Duality

Find: F', FD, $FD_{A',B',C'}$, $FD'_{A',B',C'}$

$$F_{A,B,C} = AB + AB'C + BC'$$

$$F' = [A \cdot B + A \cdot B' \cdot C + B \cdot C']'$$

$$= (A' + B') \cdot (A' + B + C') \cdot (B' + C)$$

$$FD = (A + B) \cdot (A + B' + C) \cdot (B + C')$$

$$FD_{A',B',C'} = (A' + B') \cdot (A' + B + C') \cdot (B' + C)$$

$$FD'_{A',B',C'} = [(A' + B') \cdot (A' + B + C') \cdot (B' + C)]'$$

$$= A \cdot B + A \cdot B' \cdot C + B \cdot C'$$

Representation of Logic Functions

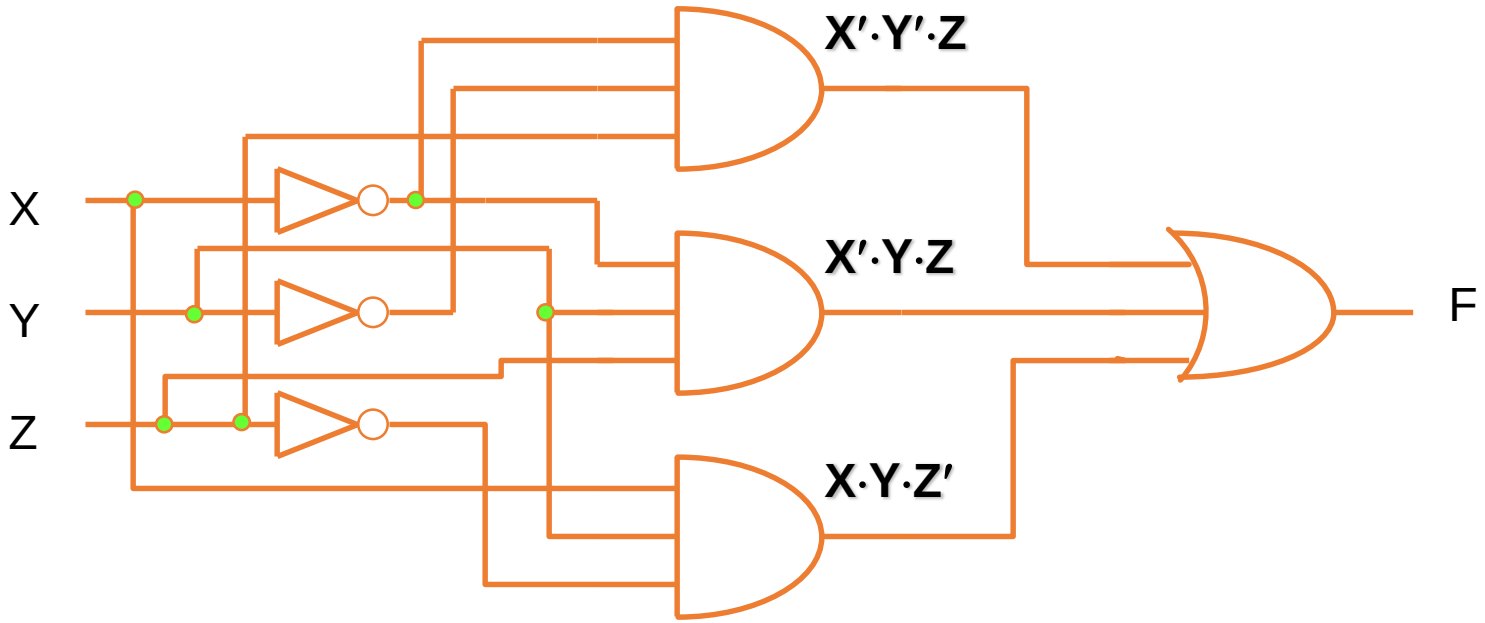
- **Truth table** with 2^n rows, n: the number of variables
- **Literal**: a variable or its complement
  - X, Y′
- n-variable **minterm**: *product* term with n literals
  - X′ · Y · Z
- n-variable **maxterm**: *sum* term with n literals
  - X + Y′ + Z

Canonical Representation

- **Canonical Sum** (Sum of Products - SOP):
  - Sum of minterms corresponding to input combinations for which the function produces a 1 output.
  - $\mathbf{F_{X,Y,Z} = \Sigma\,(1,3,6)}$
  - $\mathbf{F = X' \cdot Y' \cdot Z + X' \cdot Y \cdot Z + X \cdot Y \cdot Z'}$
- **Canonical Product** (Product of Sums - POS):
  - Product of maxterms corresponding to input combinations for which the function produces a 0 output.
  - $\mathbf{F_{X,Y,Z} = \Pi\,(0,2,4,5,7)}$
  - $\mathbf{F = (X + Y + Z) \cdot (X + Y' + Z) \cdot (X' + Y + Z) \cdot (X' + Y + Z') \cdot (X' + Y' + Z')}$

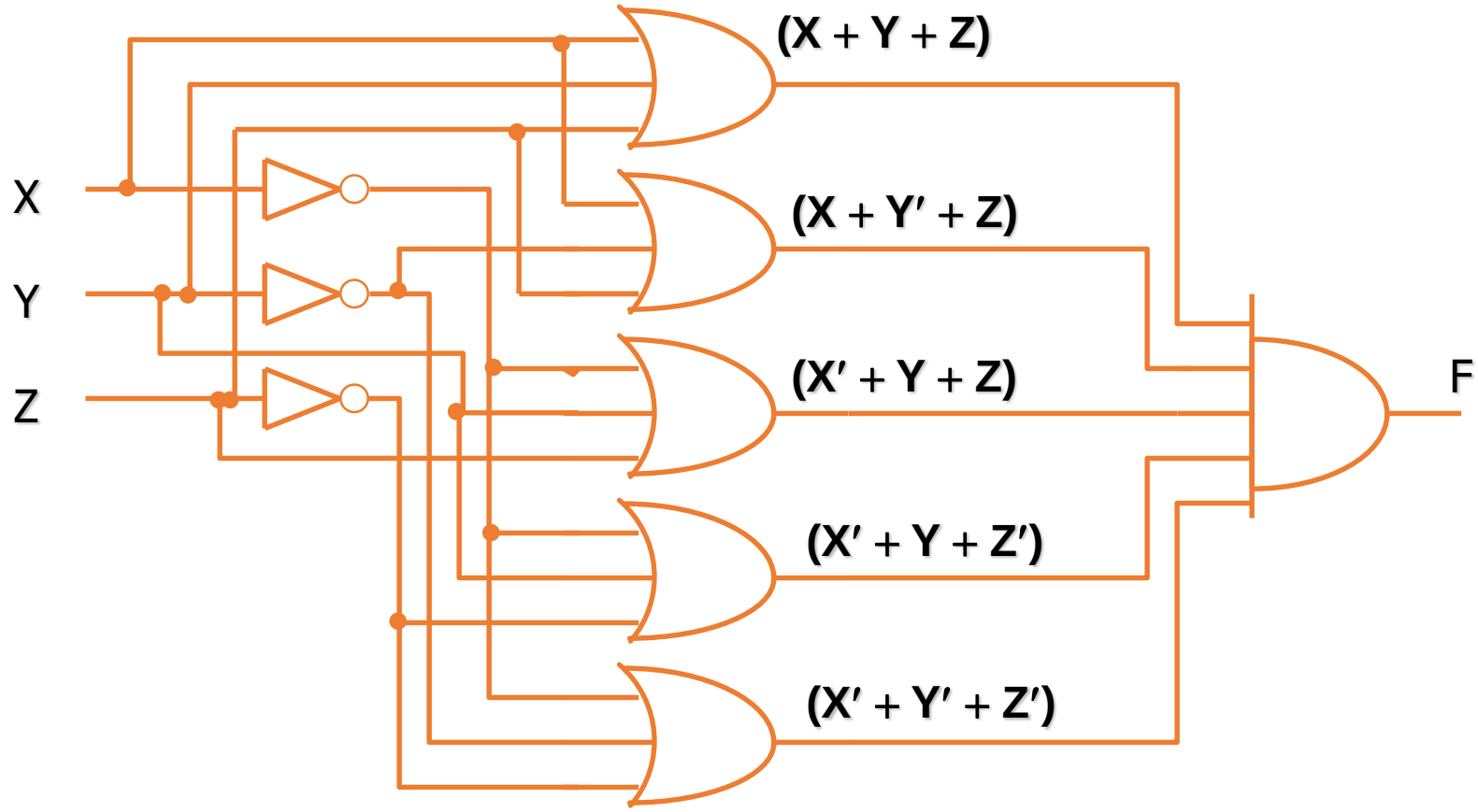
Canonical Sum Implementation

$$F = X' \cdot Y' \cdot Z + X' \cdot Y \cdot Z + X \cdot Y \cdot Z'$$



Canonical Product Implementation

$$F = (X + Y + Z) \cdot (X + Y' + Z) \cdot (X' + Y + Z) \cdot (X' + Y + Z') \cdot (X' + Y' + Z')$$



Logic Function Simplification

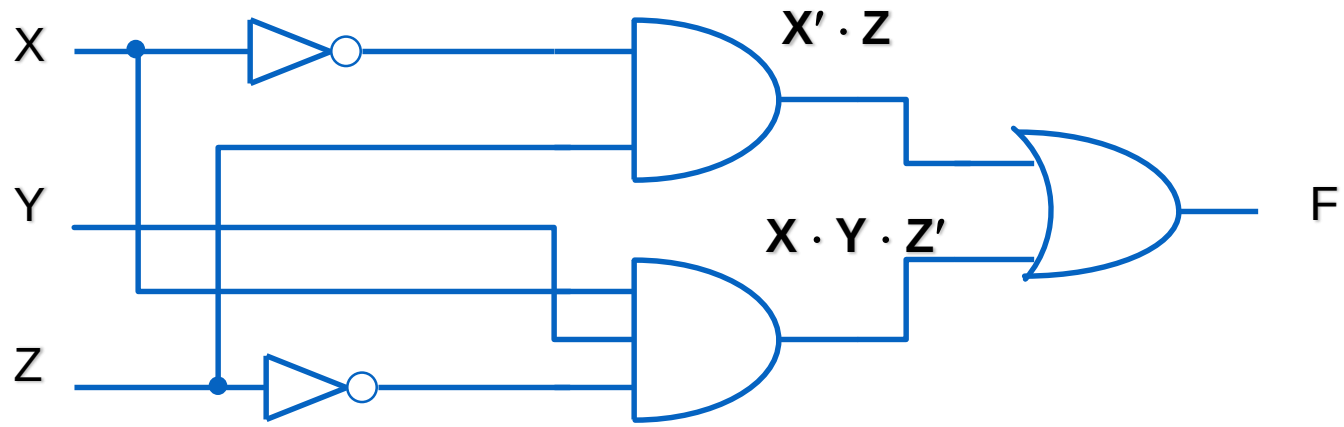
$$F = X' \cdot Y' \cdot Z + X' \cdot Y \cdot Z + X \cdot Y \cdot Z'$$

$$= X' (Y' \cdot Z + Y \cdot Z) + X \cdot Y \cdot Z'$$

$$= X' ((Y' + Y) \cdot Z) + X \cdot Y \cdot Z'$$

$$= X' (1 \cdot Z) + X \cdot Y \cdot Z'$$

$$= X' \cdot Z + X \cdot Y \cdot Z'$$



Exercise

F(A,B,C) = Σ (0,2,4,7); Write:

The Truth Table; The Canonical Sum (SOP); The Canonical Product (POS)

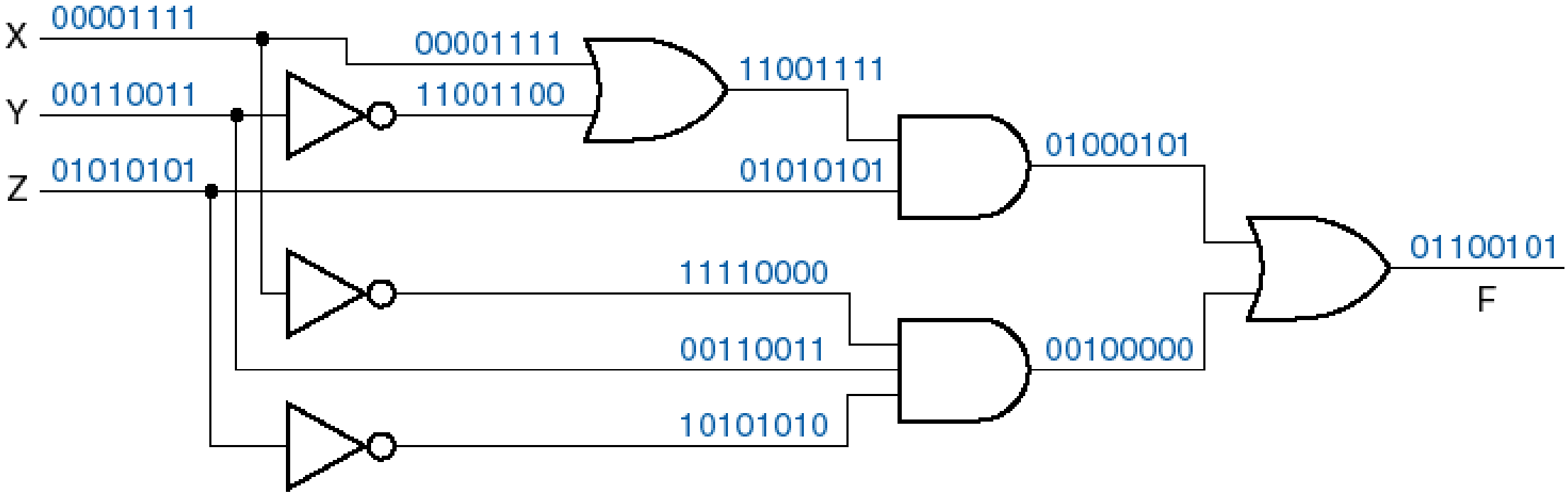
Row	X	Y	Z	F	Minterms	Maxterms
0	0	0	0			
1	0	0	1			
2	0	1	0			
3	0	1	1			
4	1	0	0			
5	1	0	1			
6	1	1	0			
7	1	1	1			

SOP: $F = A' \cdot B' \cdot C' + A' \cdot B \cdot C' + A \cdot B' \cdot C' + A \cdot B \cdot C$

POS: $F = (A + B + C) \cdot (A + B' + C) \cdot (A' + B + C') \cdot (A' + B' + C')$

Combinational Circuit Analysis

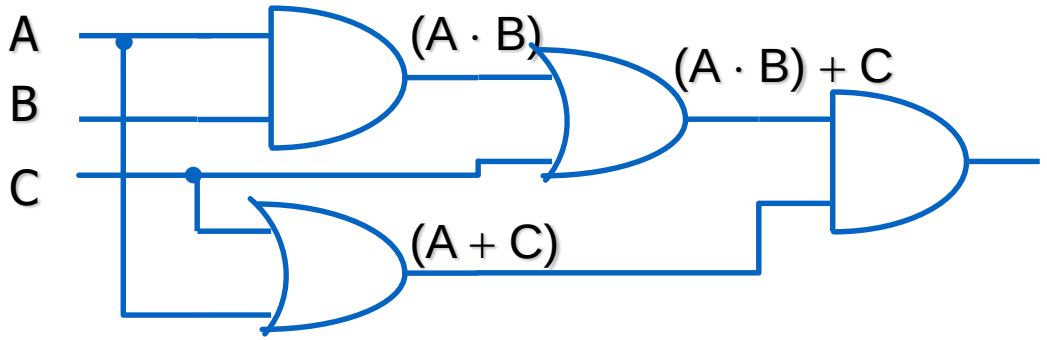
Combinational Circuit Analysis



Logic Circuit → Logic Function → Truth Table

Example 1:

Logic Circuit → Logic Function → Truth Table



$$F = (A + C) \cdot ((A \cdot B) + C)$$

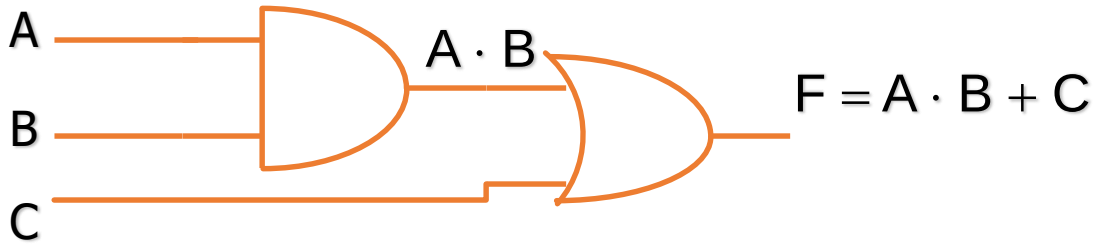
A	B	C	AB	AB+C	A+C	**F**
0	0	0	0	0	0	**0**
0	0	1	0	1	1	**1**
0	1	0	0	0	0	**0**
0	1	1	0	1	1	**1**
1	0	0	0	0	1	**0**
1	0	1	0	1	1	**1**
1	1	0	1	1	1	**1**
1	1	1	1	1	1	**1**

Example 1:

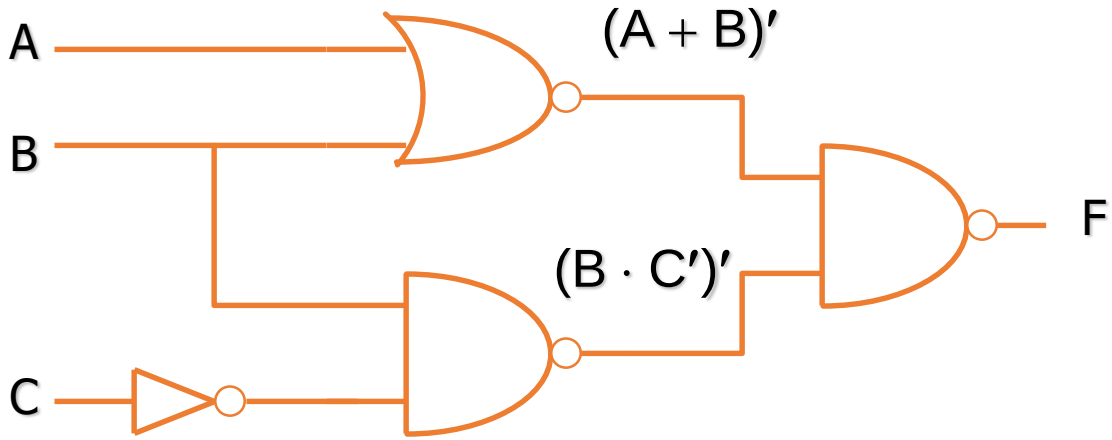
Logic Function Simplification

$$F = (A + C) \cdot ((A \cdot B) + C)$$
$$= A \cdot (A \cdot B + C) + C \cdot (A \cdot B + C)$$
$$= A \cdot A \cdot B + A \cdot C + C \cdot A \cdot B + C \cdot C$$
$$= A \cdot B + A \cdot C + A \cdot B \cdot C + C$$
$$= (A \cdot B + A \cdot B \cdot C) + (A \cdot C + C)$$
$$= A \cdot B \cdot (1 + C) + ((A + 1) \cdot C)$$
$$= A \cdot B + C$$

Same function, different circuit



Example 2:



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

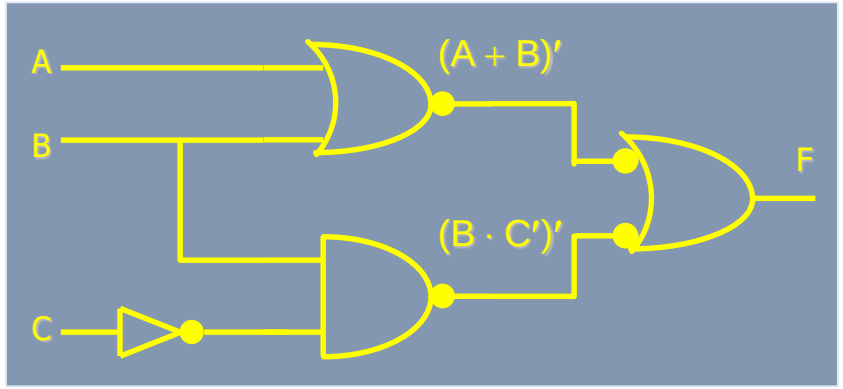
A	B	C	F
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

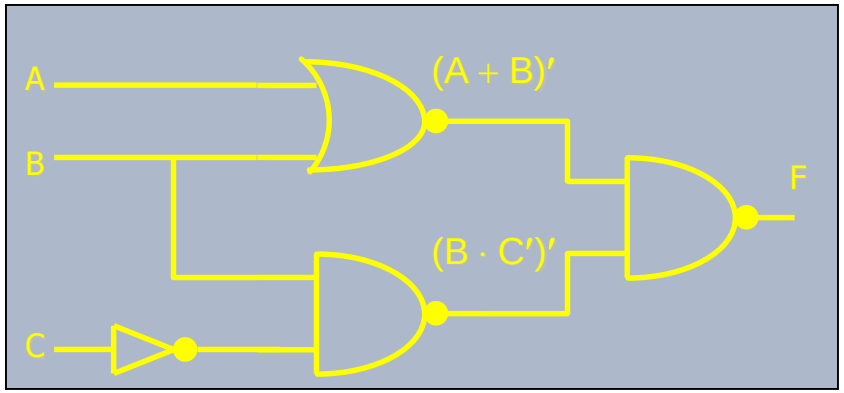
$$F_{A,B,C} = \Sigma\,(2,3,4,5,6,7)$$

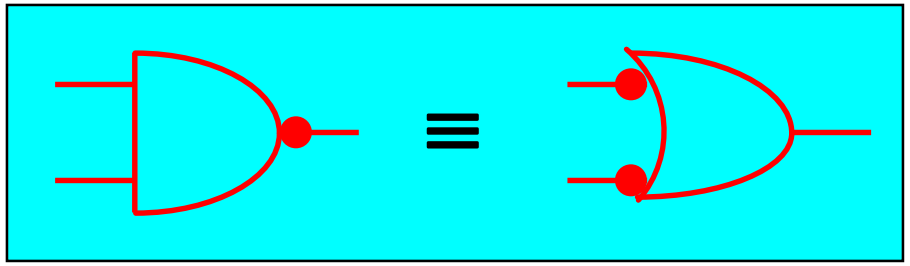
OR

$$F_{A,B,C} = \Pi\,(0,1)$$

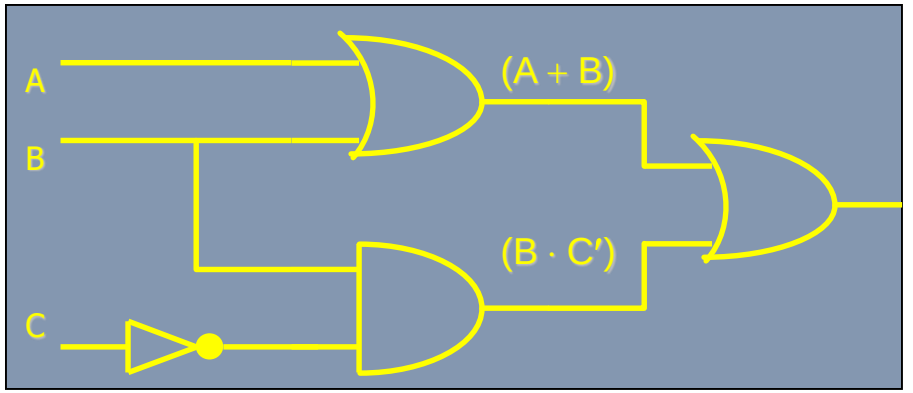
Example 2 (Contd.):







$$F = A + B + B \cdot C'$$



Example 2 (Contd.):

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

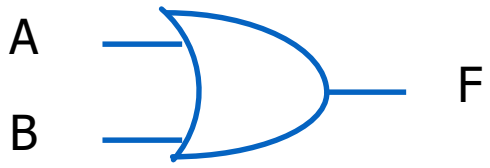
$$= (A + B)'' + (B \cdot C')'' \qquad \text{using Demorgan's theorem}$$

$$= A + B + B \cdot C'$$

$$= A + B \cdot (1 + C')$$

$$= A + B \cdot 1$$

$$= A + B$$



Combinational Analysis Review:

Combinational Analysis Review:

Logic Circuit → Logic Function → Truth Table

Analyze a combinational logic circuit by obtaining a formal description of its logic function

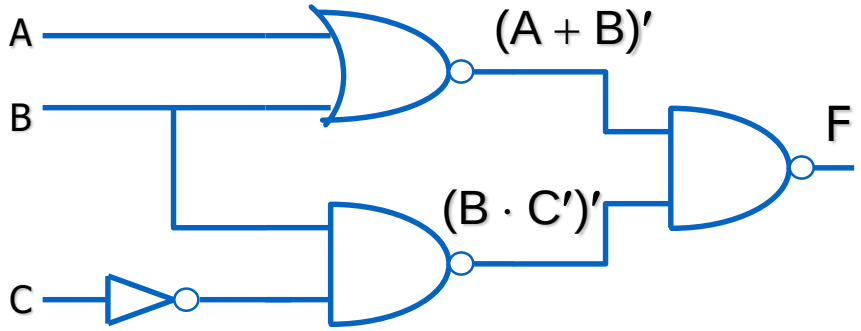
Outputs of a combinational logic circuit depends only on its current inputs (not on history)

Kinds of combinational analysis:

- –exhaustive (truth table)
- –algebraic (expressions)

Example:

I)



II)

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

III)

	A	B	C	F
0	0	0	0	**0**
1	0	0	1	**0**
2	0	1	0	**1**
3	0	1	1	**1**
4	1	0	0	**1**
5	1	0	1	**1**
6	1	1	0	**1**
7	1	1	1	**1**

IV)

$F_{A,B,C} = \Sigma\,(2,3,4,5,6,7)$

$F = A'BC' + A'BC + AB'C' + AB'C + ABC' + ABC$

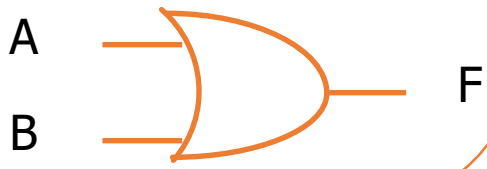
V)

$F_{A,B,C} = \Pi\,(0,1)$

$F = (A+B+C)(A+B+C')$

VI)

$F = A + B$



Combinational Circuit Synthesis

Combinational Circuit Synthesis

Truth Table → Logic Function → Logic Circuit

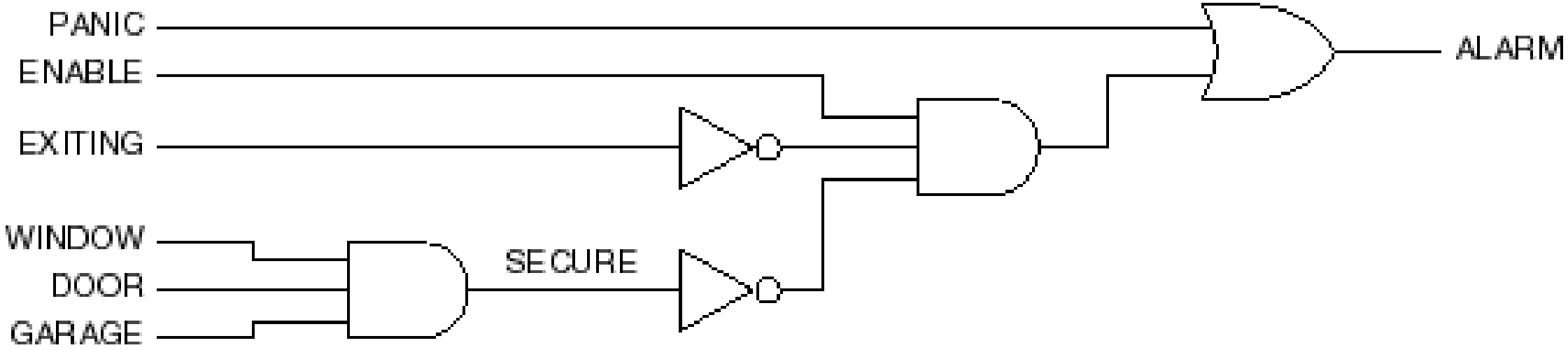
1. **Logic Function:**
   - The canonical sum expression.
   - The canonical product expression.
2. **The canonical Implementations:**
   - AND-OR and its equivalent NAND-NAND
   - OR-AND and its equivalent NOR-NOR
3. **Logic Function minimization:**
   - Simplifying the logic function to reduce the number of gates.
4. **Minimization methods:**
   - Using theorems
   - Karnaugh map

Combinational Circuit Design

- Sometimes you can write an equation or equations directly using ‘logic’.

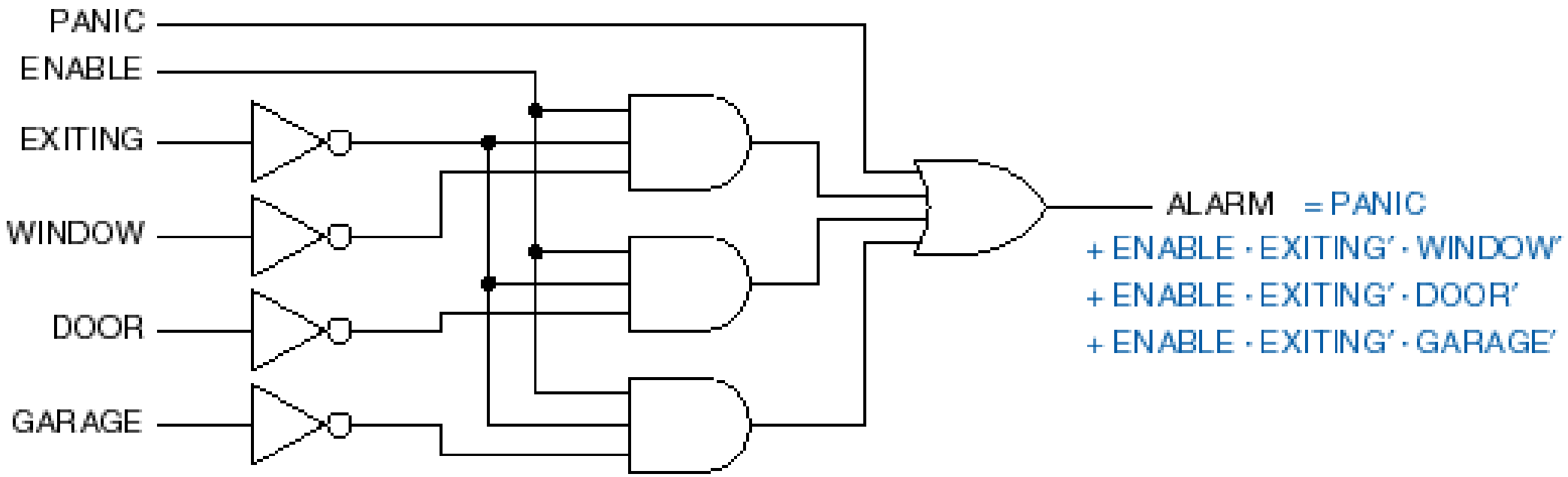
Example (alarm circuit):

ALARM = PANIC + ENABLE . EXITING’ . SECURE’
SECURE = WINDOW . DOOR . GARAGE
ALARM = PANIC + ENABLE . EXITING’ . (WINDOW . DOOR . GARAGE)’

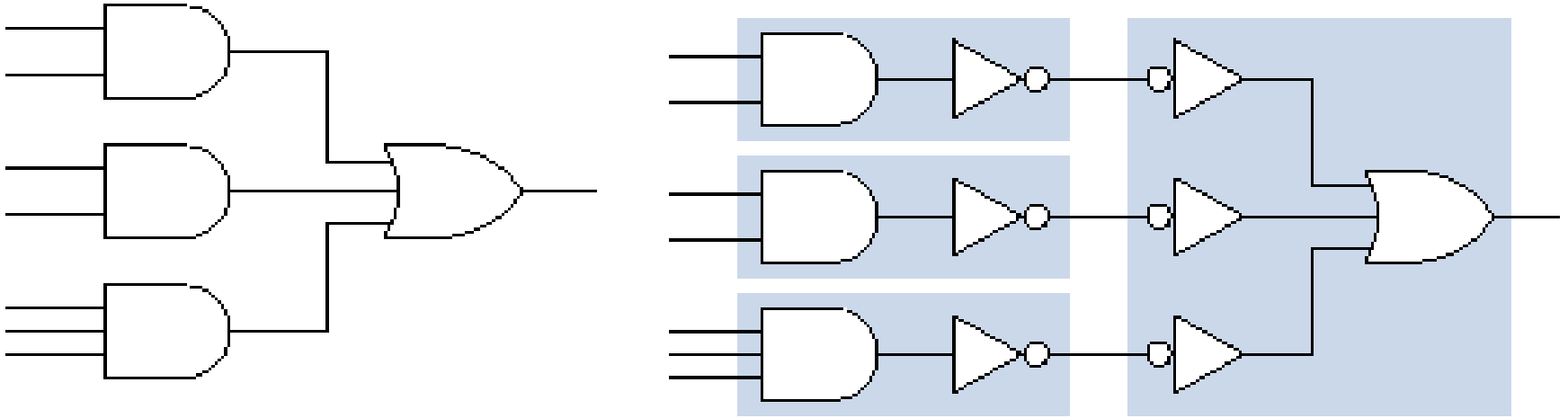


Example (Contd): Alarm Circuit Transformation

- Sum-of-products form
  - Useful for programmable logic devices
  - Multiply out



Sum-of-Products Form

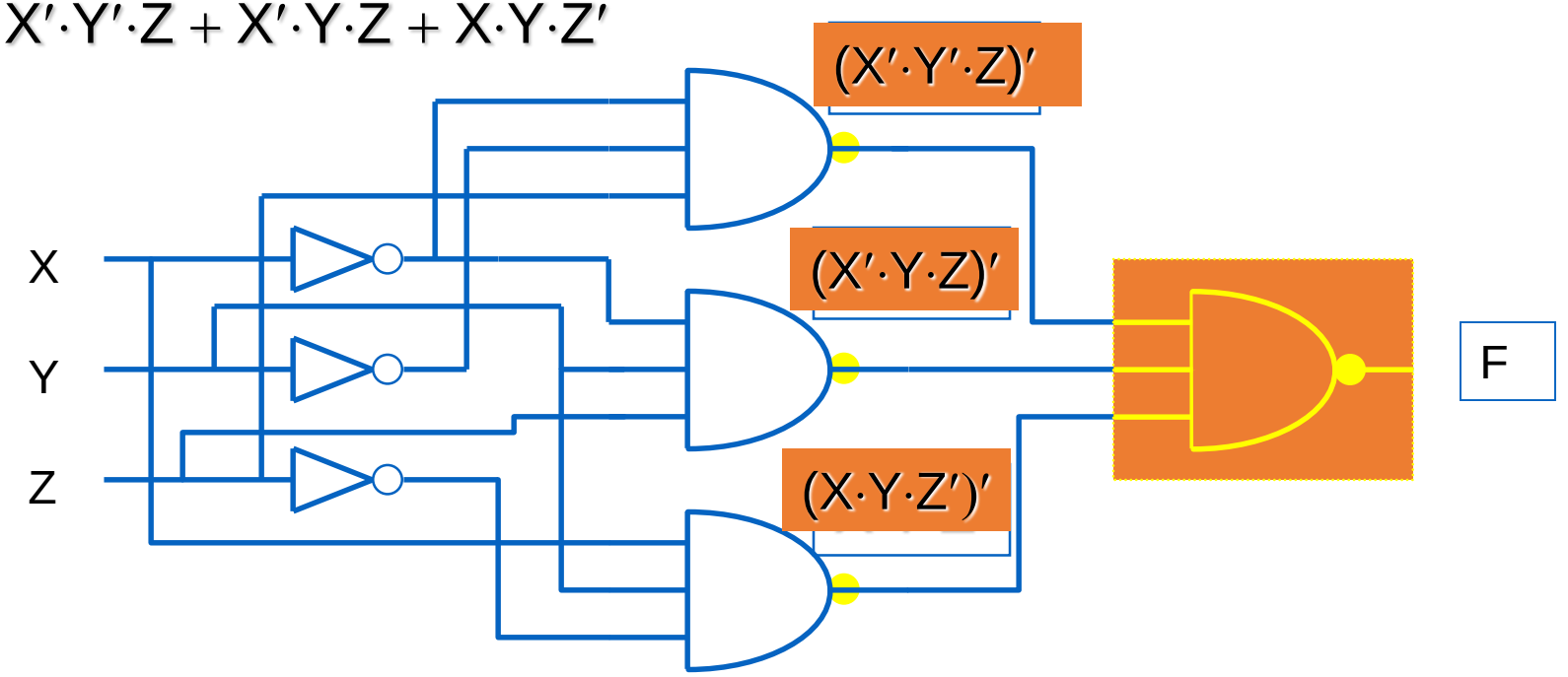


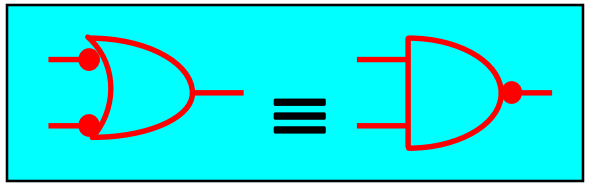
AND-OR

NAND-NAND

NAND-NAND Implementation

$F = X' \cdot Y' \cdot Z + X' \cdot Y \cdot Z + X \cdot Y \cdot Z'$

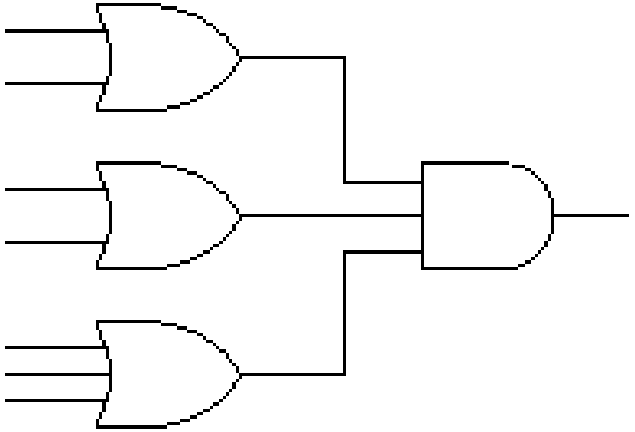


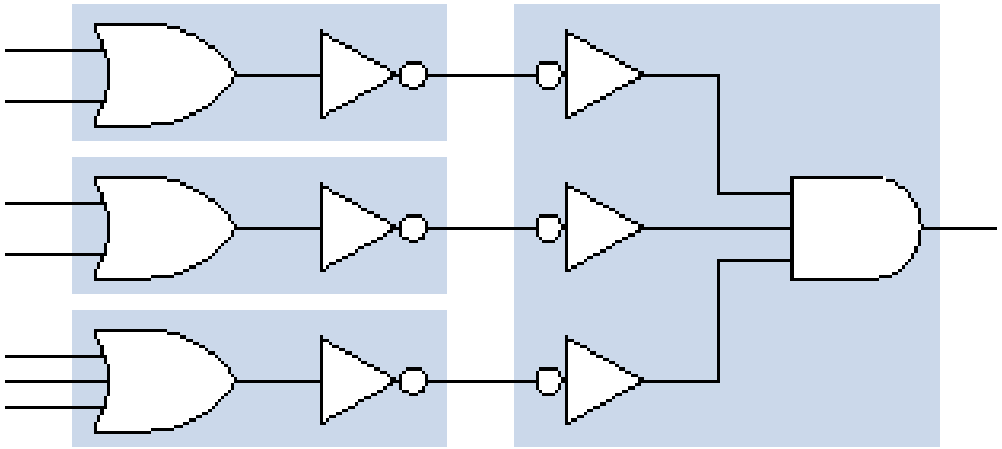


$F = [(X' \cdot Y' \cdot Z)' \cdot (X' \cdot Y \cdot Z)' \cdot (X \cdot Y \cdot Z')']'$

$F = X' \cdot Y' \cdot Z + X' \cdot Y \cdot Z + X \cdot Y \cdot Z$

Product-of-Sums Form



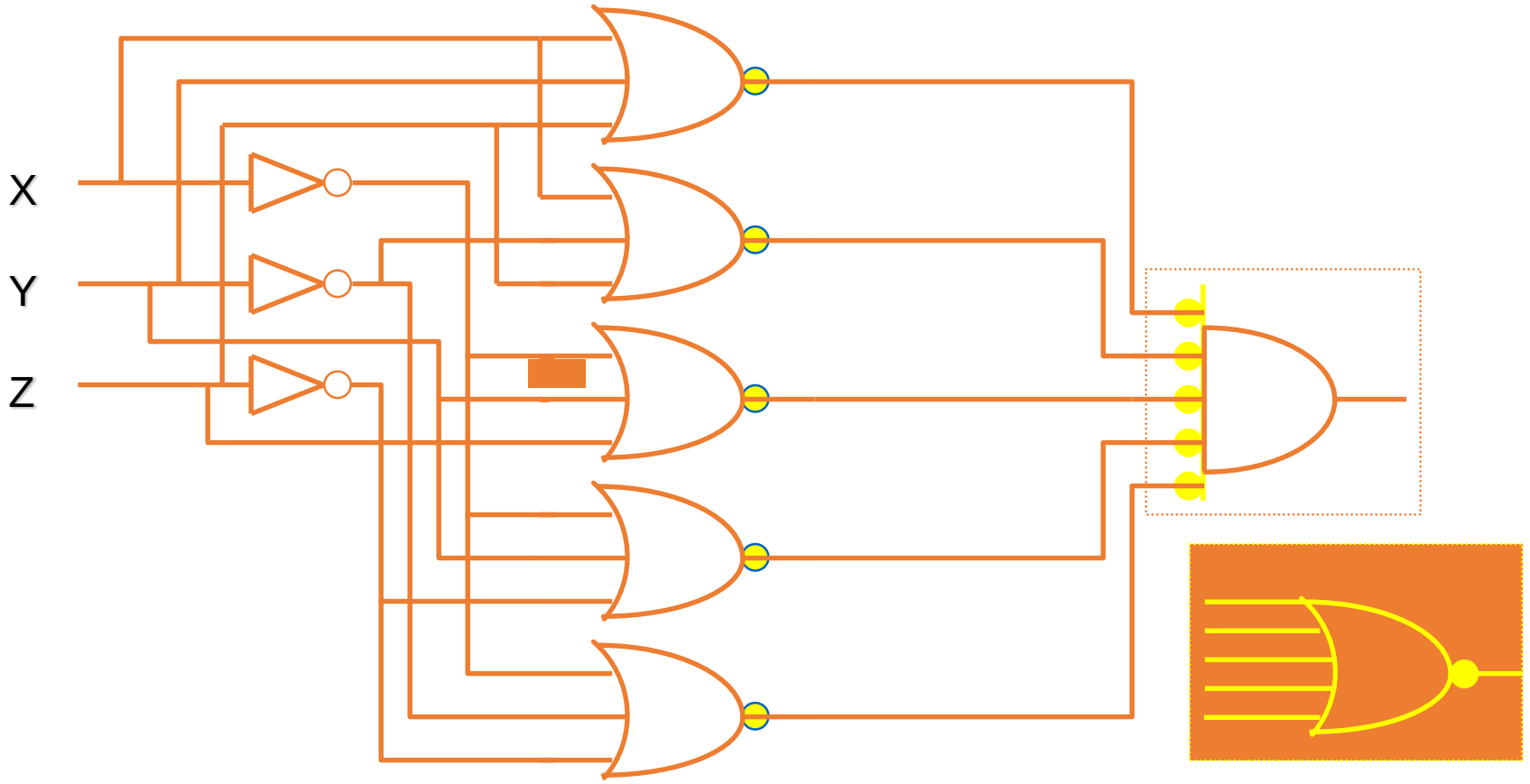


OR - AND

NOR - NOR

NOR-NOR Implementation

- $F = (X + Y + Z)\cdot(X + Y' + Z)\cdot(X' + Y + Z)\cdot(X' + Y + Z')\cdot(X' + Y' + Z')$



Minimization Example: Prime Number Detector

Truth table → canonical sum (sum of minterms)

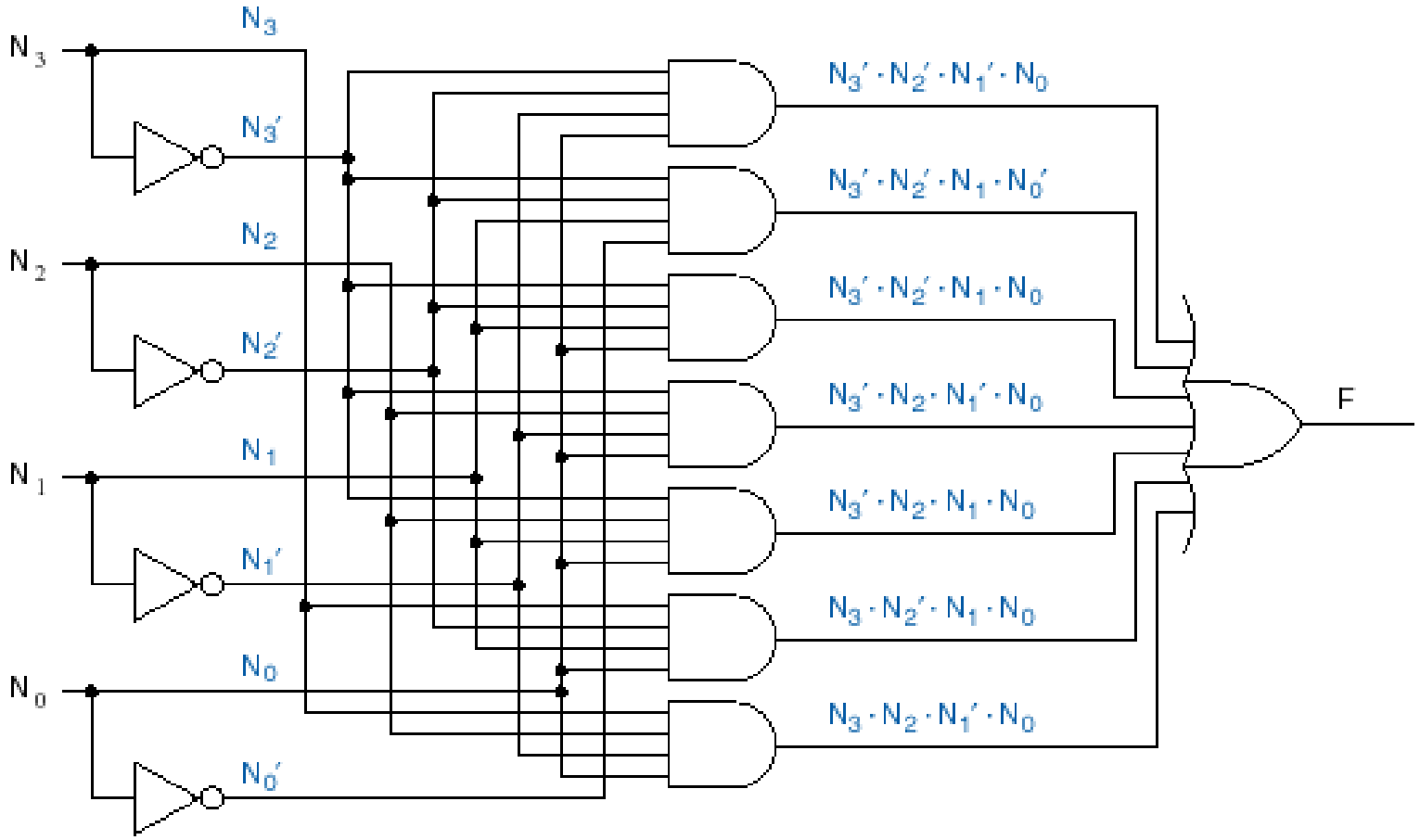
$F = \Sigma_{N3N2N1N0}(1,2,3,5,7,11,13)$

Row	N3	N2	N1	N0	F
0	0	0	0	0	**0**
1	0	0	0	1	**1**
2	0	0	1	0	**1**
3	0	0	1	1	**1**
4	0	1	0	0	**0**
5	0	1	0	1	**1**
6	0	1	1	0	**0**
7	0	1	1	1	**1**
8	1	0	0	0	**0**
9	1	0	0	1	**0**
10	1	0	1	0	**0**
11	1	0	1	1	**1**
12	1	1	0	0	**0**
13	1	1	0	1	**1**
14	1	1	1	0	**0**
15	1	1	1	1	**0**

Example (contd.): Minterm list → Canonical Sum

$F = \Sigma_{N3N2N1N0}(1,2,3,5,7,11,13)$

$= N_3'.N_2'.N_1'.N_0 + N_3'.N_2'.N_1.N_0' + N_3'.N_2'.N_1.N_0 + N_3'.N_2.N_1'.N_0$
$+ N_3'.N_2.N_1.N_0 + N_3.N_2'.N_1.N_0 + N_3.N_2.N_1'.N_0$

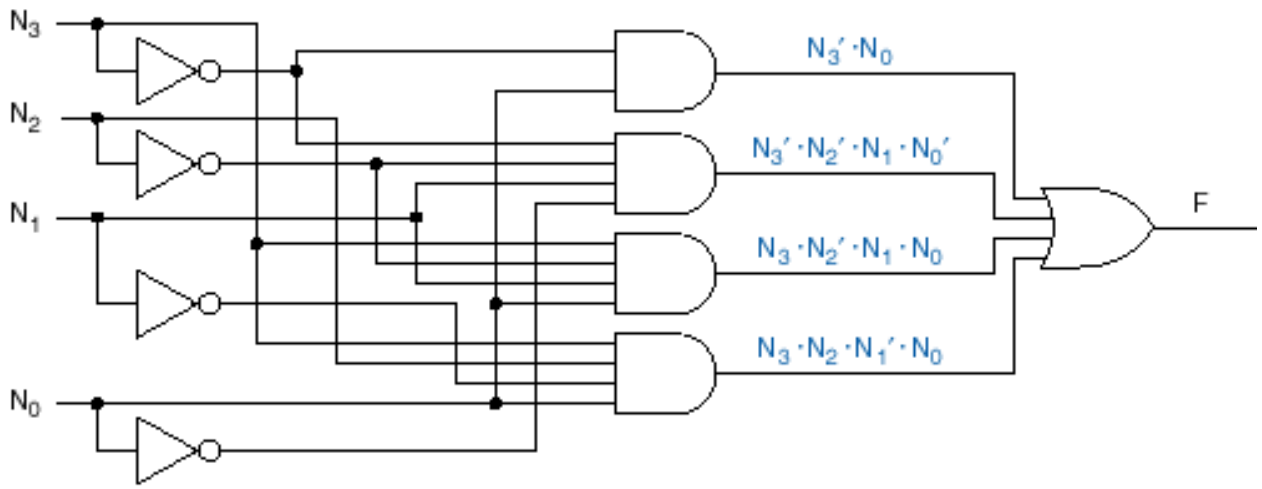


Example (Contd.): Algebraic Simplification

(T10) $\mathbf{X \cdot Y + X \cdot Y' = X}$

$F = \Sigma_{N3N2N1N0}(1,2,3,5,7,11,13)$
$= N_3'.N_2'.N_1'.N_0 + N_3'.N_2'.N_1.N_0 + N_3'.N_2.N_1'.N_0 + N_3'.N_2.N_1.N_0 + \ldots$
$= (N_3'.N_2'.N_1'.N_0 + N_3'.N_2'.N_1.N_0) + (N_3'.N_2.N_1'.N_0 + N_3'.N_2.N_1.N_0) + \ldots$
$= N_3'.N_2'.N_0 + N_3'.N_2.N_0 + \ldots$

Reduce number of gates and gate inputs



Karnaugh Maps

- A karnaugh map is a representation of the truth table by a matrix of cells, where each cell corresponds to a minterm (or a maxterm) of the logic function.

- For n-variable function, we need 2^n rows truth table and 2^n cells.

- The cell number is equivalent to the row number in the truth table.

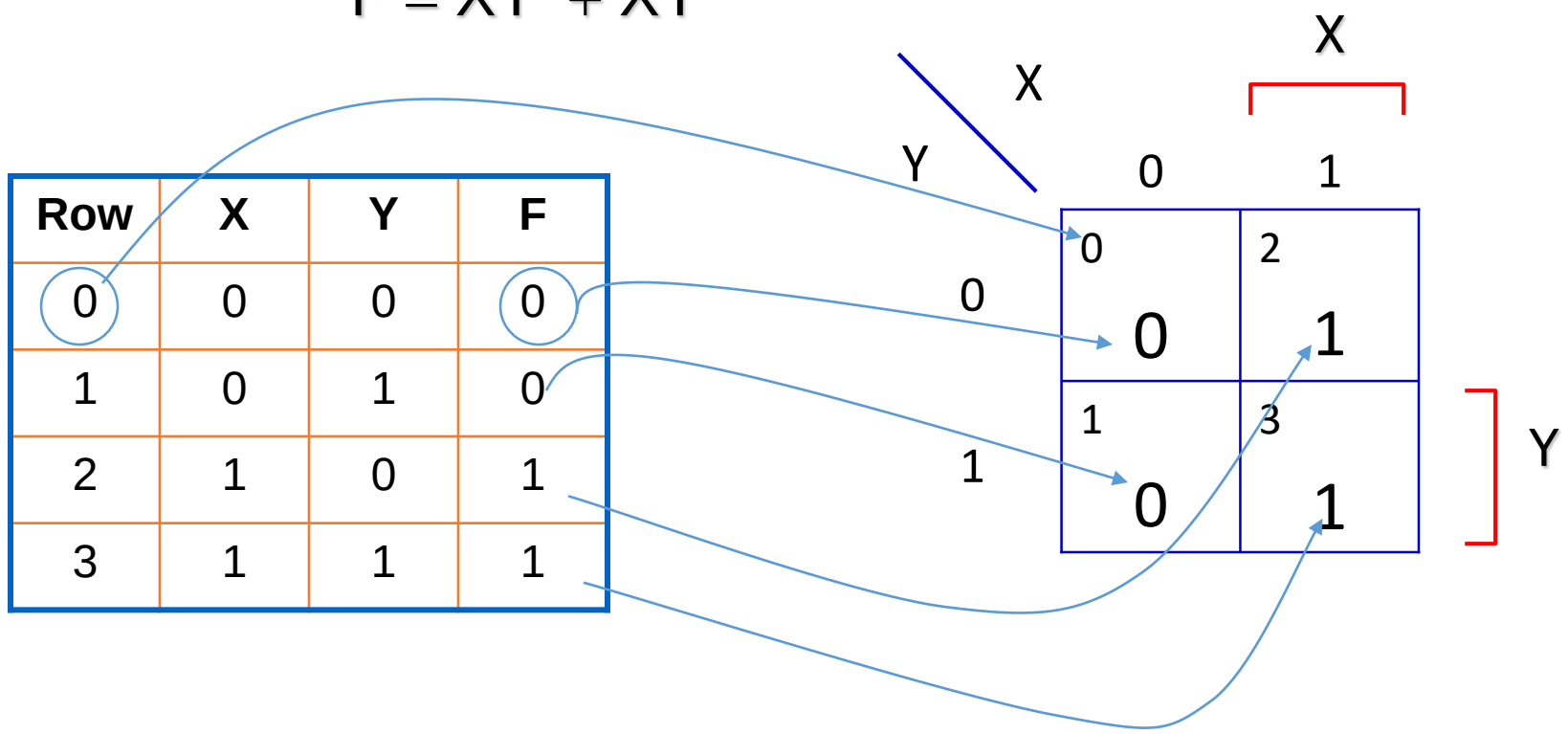
- The truth table values are copied into their corresponding cells.

Two-Variable Karnaugh Map

Example 1:

$$F = XY' + XY$$

Row	X	Y	F
0	0	0	0
1	0	1	0
2	1	0	1
3	1	1	1

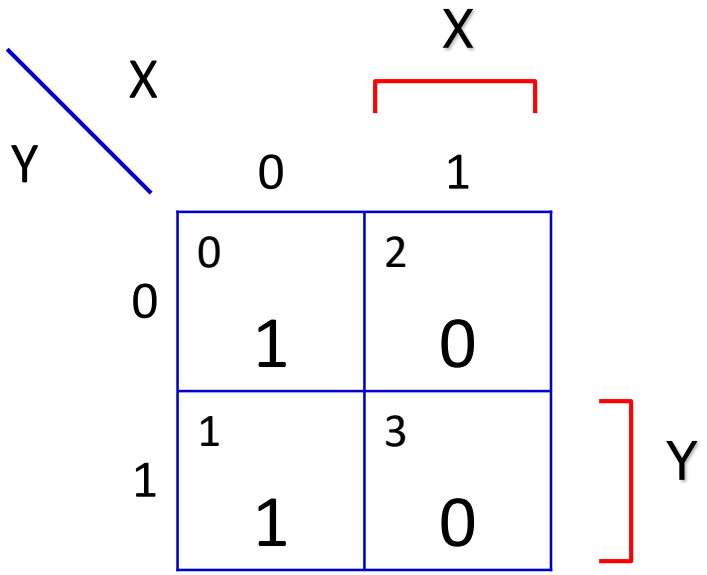


Simplification: $F = X(Y' + Y) = X \cdot 1 = X$

Example 2

$$F = X'Y' + X'Y$$

Row	X	Y	F
0	0	0	1
1	0	1	1
2	1	0	0
3	1	1	0

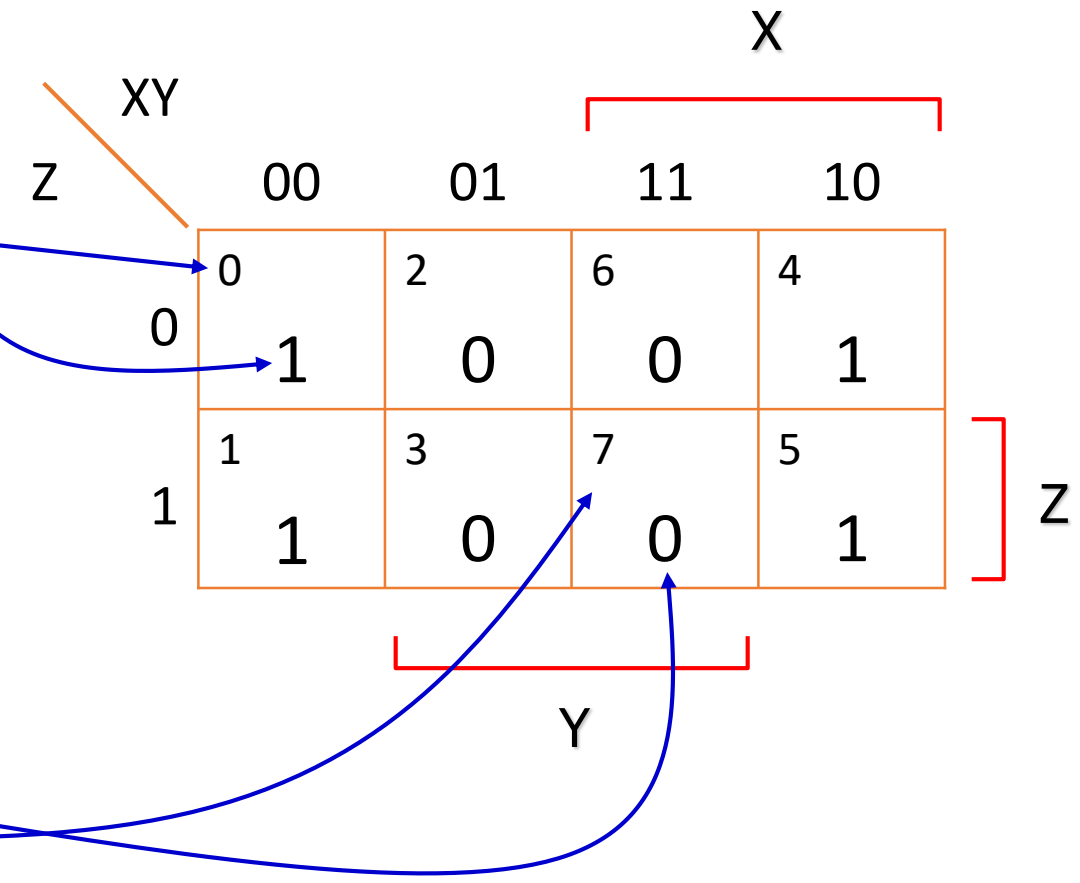


Simplification: $\mathbf{F} = X'(Y' + Y) = X' \cdot 1 = \mathbf{X'}$

Three-Variable Karnaugh Map

Example 3: $F = X'Y'Z' + X'Y'Z + XY'Z' + XY'Z$

Row	X	Y	Z	F
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0



Simplification: $\mathbf{F} = X'Y'(Z' + Z) + XY'(Z' + Z) = X' \cdot Y' + XY' = (X' + X)Y' = \mathbf{Y'}$

Karnaugh-Map Usage

1. **Plot 1s** corresponding to **minterms** of function.
2. **Circle largest** possible rectangular **sets of 1s**.
   - number of 1s in set must be power of 2
   - OK to cross edges
3. Read off product terms, one per circled set.
   - Variable is 1 → include variable
   - Variable is 0 → include complement of variable
   - Variable is both 0 and 1 → variable not included
4. Circled sets and corresponding product terms are called '**prime implicants**'
5. Minimum number of gates and gate inputs

Example 1

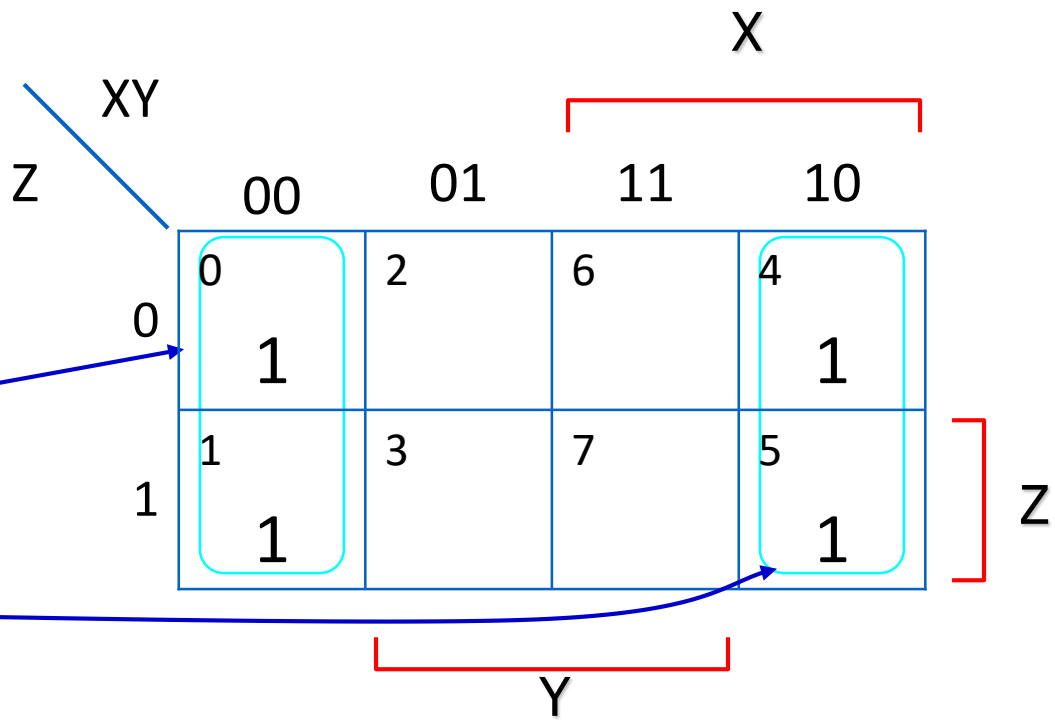
Combining Two cells:

I) Cells (0,1): X=Y=0; Z=0,1

Product term: **X'Y'**

Cells (4,5): X=1; Y=0; Z=0,1

Product term : **XY'**



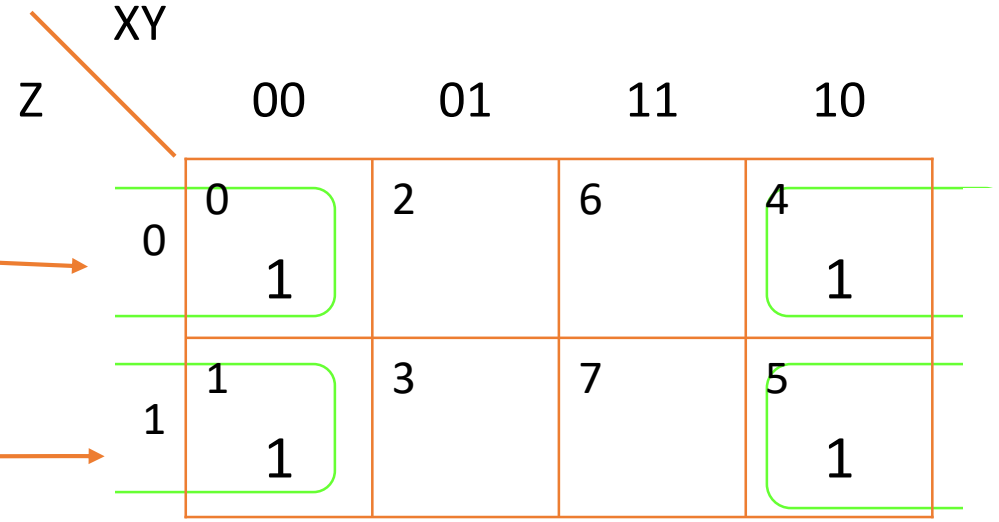
OR:

II) Cells (0,4): Y=0; Z=0; X=0,1

Product term: **Y'Z'**

Cells (1,5): Y=0; Z=1; X=0,1

Product term : **Y'Z**



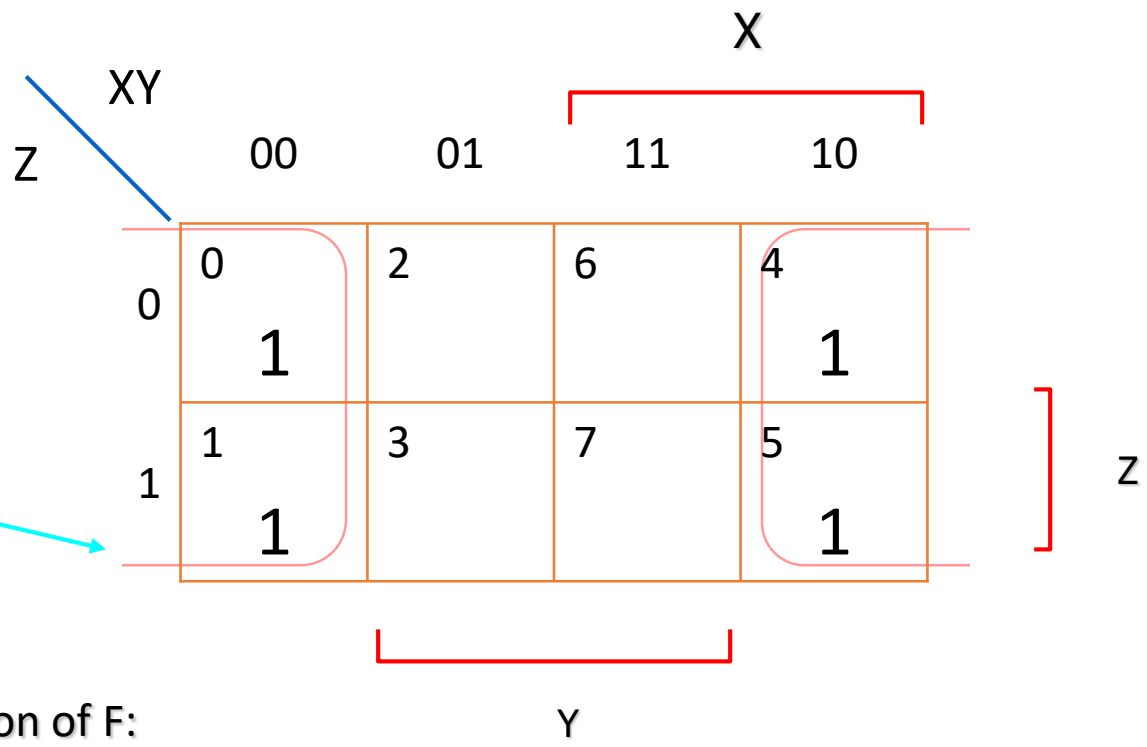
Example 1 (Contd.)

Combining four cells

III) Cells (0,1,4,5)

X=0,1; Z=0,1; Y=0

The product term: **Y'**



The sum of products expression of F:

From I) F =X'Y'+XY' or

From II) F=Y'Z'+Y'Z or:

From III) **F =Y'**

Example 2

Combine cells (0,2,6,4)

X=0,1; Y= 0,1; Z=0

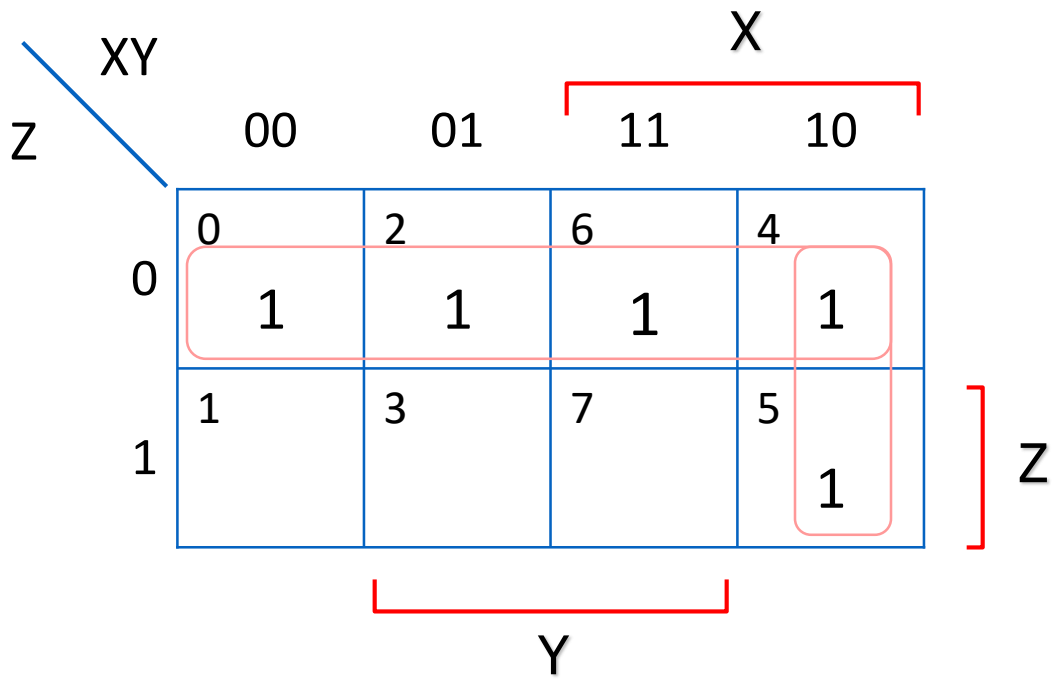
Product Term: **Z'**

Combine cells (4,5)

X=1; Y=0; Z=0,1

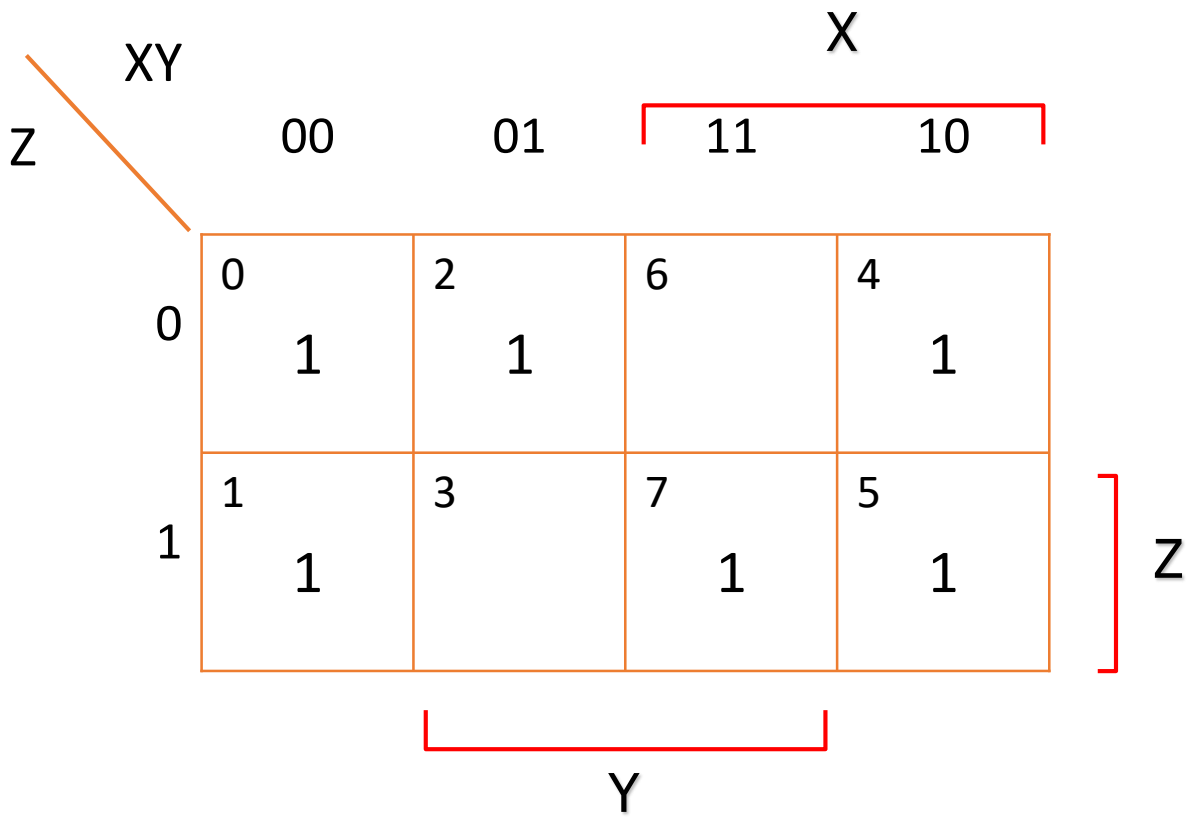
Product Term: **XY'**

F = XY'+Z'



The canonical sum is: F=X'Y'Z'+XYZ'+XY'Z'+XY'Z+XYZ'

Exercise



Definitions

1. A logic function P ***implies*** a logic function F if for every input combination for which P=1, then F=1 also. (P is an ***implicant*** of F)
2. Any minterm or combination of minterms in the canonical sum expression is an ***implicant*** of the output function
3. A ***prime implicant*** is a group of combined minterms that can't be combined with any other minterm or group of minterms
4. ***Essential prime implicant*** is a prime implicant in which one or more minterms are unique (i.e at least one minterm is not contained in any other prime implicant. A unique minterm is called an ***Essential 1-Cell***
5. A ***minimal sum*** of a logic function is a sum-of-products expression for F such that no sum-of-products expression for F has fewer product terms.

Example 1

Combining (0,2)

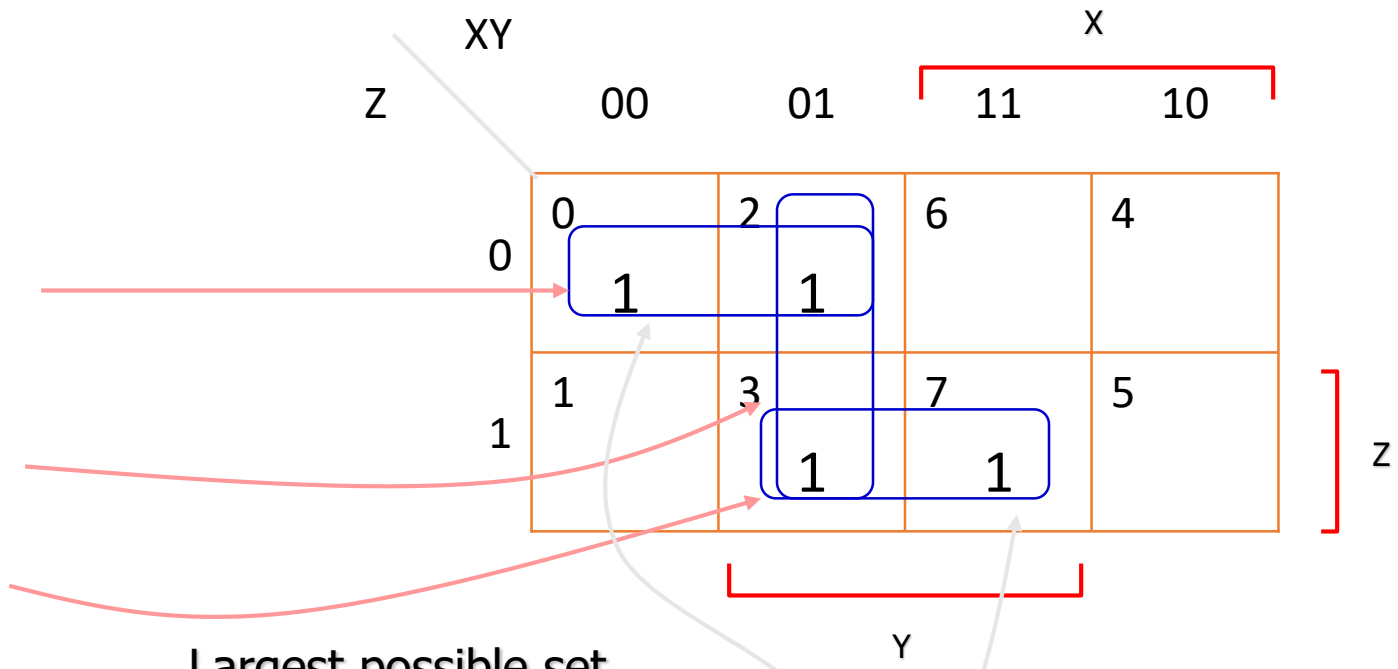
Product term: **X'Z'**

Combining (2,3)

Product term: **X'Y**

Combining (3,7)

Product term: **YZ**



X'Z', **X'Y**, and **YZ** are prime implicants

X'Z', **YZ** are **essential prime implicants**

X'Y is non-essential prime implicant (redundant) because all its minterms are covered in the other essential prime implicants

F= X'Z'+X'Y+YZ OR: **F = X'Z'+YZ** (the minimal sum of F)

Example 2

The prime implicants:

1- (0,2) **X'Z'**

2- (0,4) **Y'Z'**

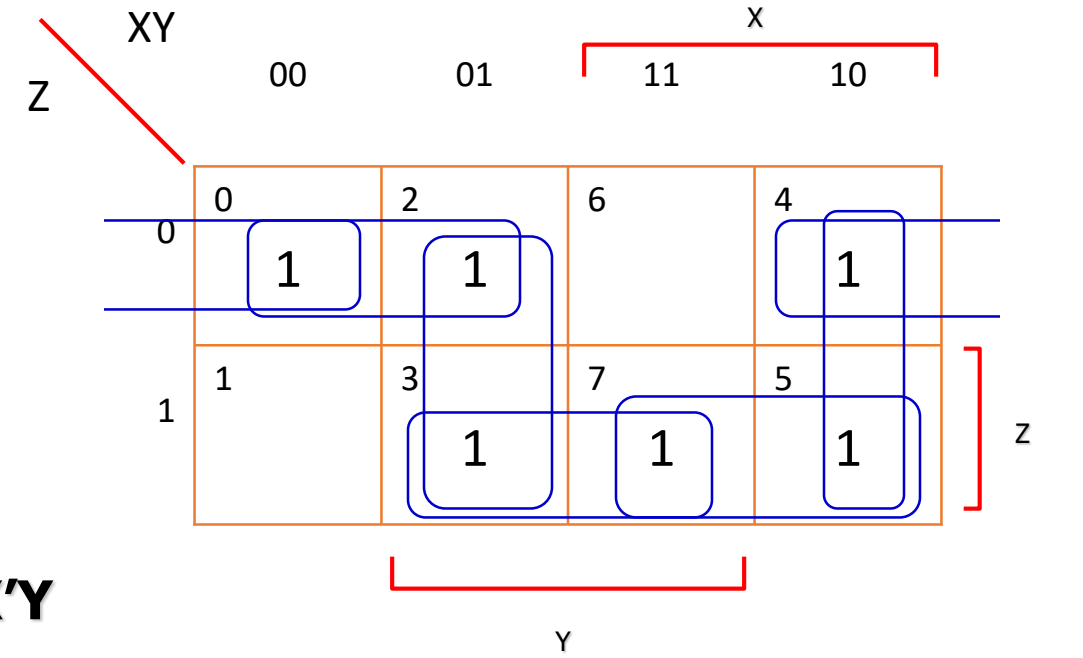
No essential prime implicant!

3- (2,3) **X'Y**

4- (3,7) **YZ**

5- (4,5) **XY'**

6 -(5,7) **XZ**



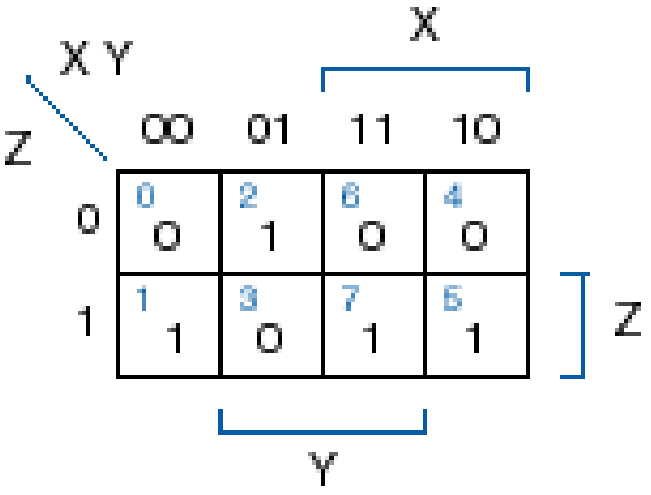
Two possible minimal sums:

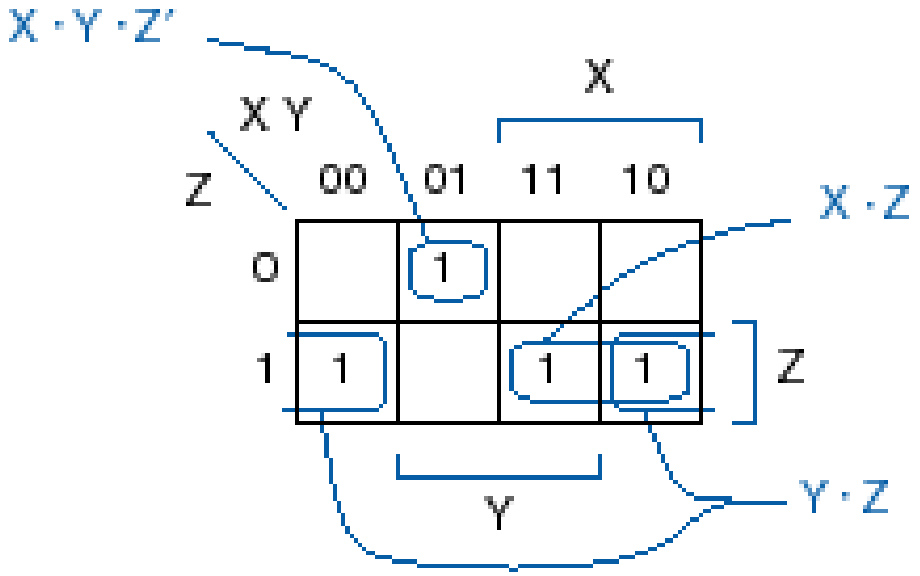
1- Using the prime implicants 1,4,and 5; **F= X'Z'+YZ+XY'**

2- Using the prime implicants 2,3,and 6; **F= Y'Z'+ X'Y+XZ**

Example 3: F = S(1,2,5,7)

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	1
1	1	0	0
1	1	1	1





Summary

1. Load the **minterms** and maxterms into the K-map by placing the 1's and 0's in the appropriate cells.

2. Look for **groups of minterms** and write the corresponding product terms (the prime implicants):

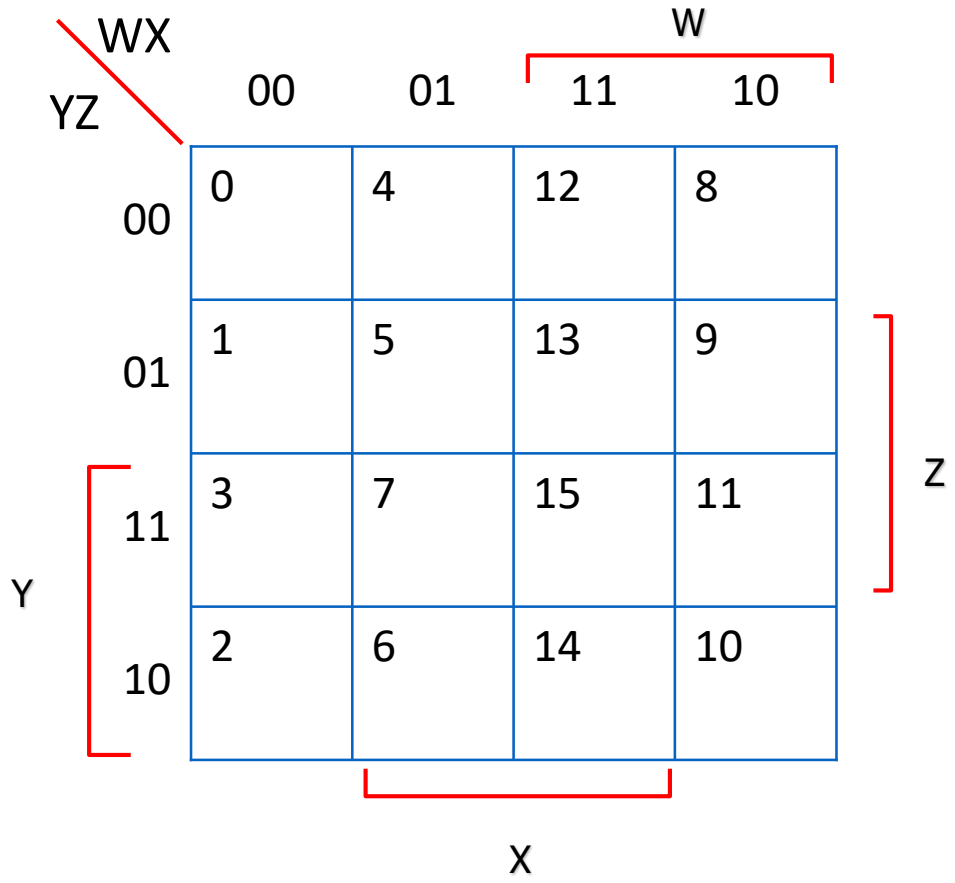
   - The group size should be a power of 2.
   - Find the largest groups of minterms first then find smaller groups of minterms until all groups are found and all 1-cells are covered.

3. Determine the **essential prime implicants**.

4. Select all essential prime implicants and the **minimal set** of the remaining prime implicants that cover the remaining 1's.

5. Its possible to get more that one equally simplified expression if more than one set of the remaining prime implicants contains the same number of minterms.

Four-Variable Karnaugh Map



Example 5

Row	W	X	Y	Z	F
0	0	0	0	0	**1**
1	0	0	0	1	**0**
2	0	0	1	0	**1**
3	0	0	1	1	**0**
4	0	1	0	0	**0**
5	0	1	0	1	**1**
6	0	1	1	0	**0**
7	0	1	1	1	**1**
8	1	0	0	0	**1**
9	1	0	0	1	**0**
10	1	0	1	0	**1**
11	1	0	1	1	**0**
12	1	1	0	0	**0**
13	1	1	0	1	**1**
14	1	1	1	0	**0**
15	1	1	1	1	**1**

F=

W′X′Y′Z′+

W′X′YZ′+

W′XYZ′+

W′XYZ+

WX′Y′Z′+

WX′YZ′+

WXY′Z′+

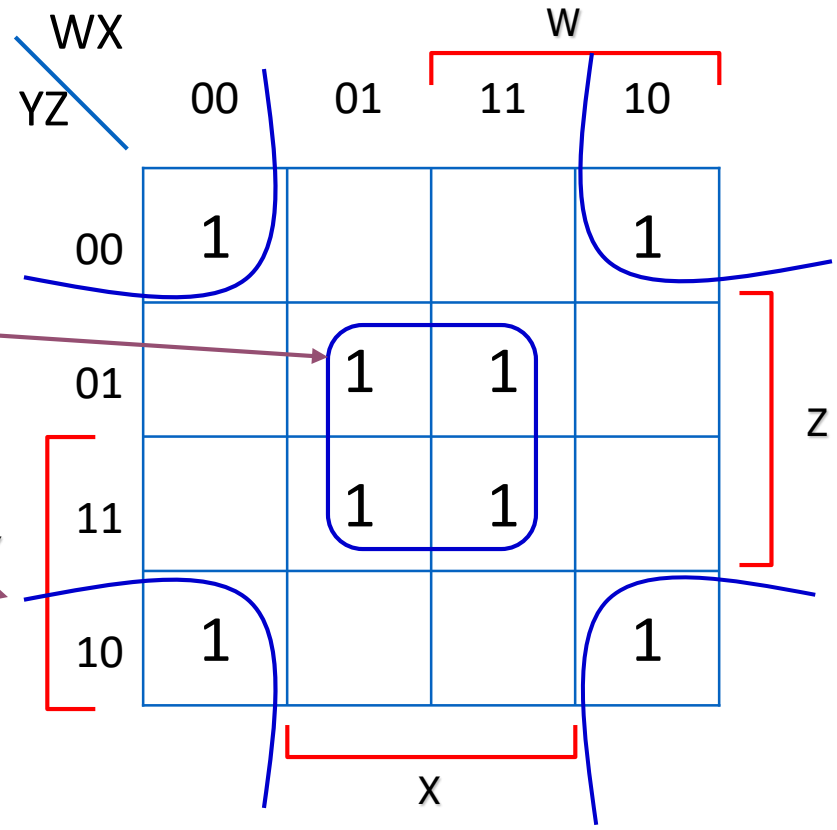
WXYZ

Example 5(Contd.)

Essential prime implicants:

The product term: **XZ**

The product term : **X'Z'**



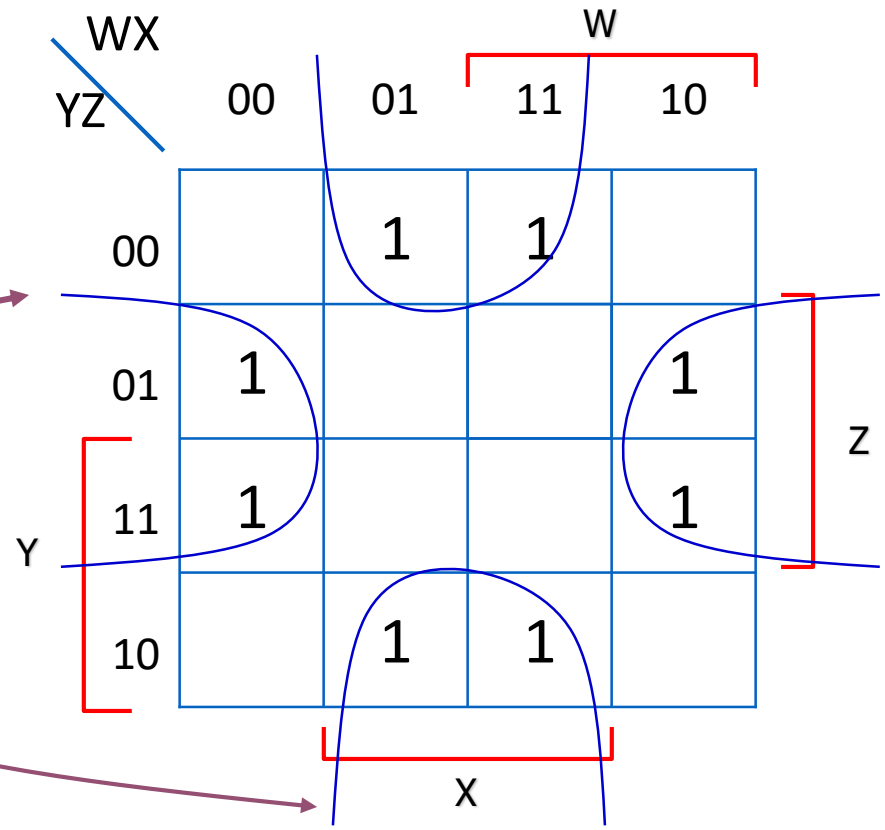
$$F = XZ + X'Z'$$

Example 6

Essential prime implicants:

The product term: **X'Z**

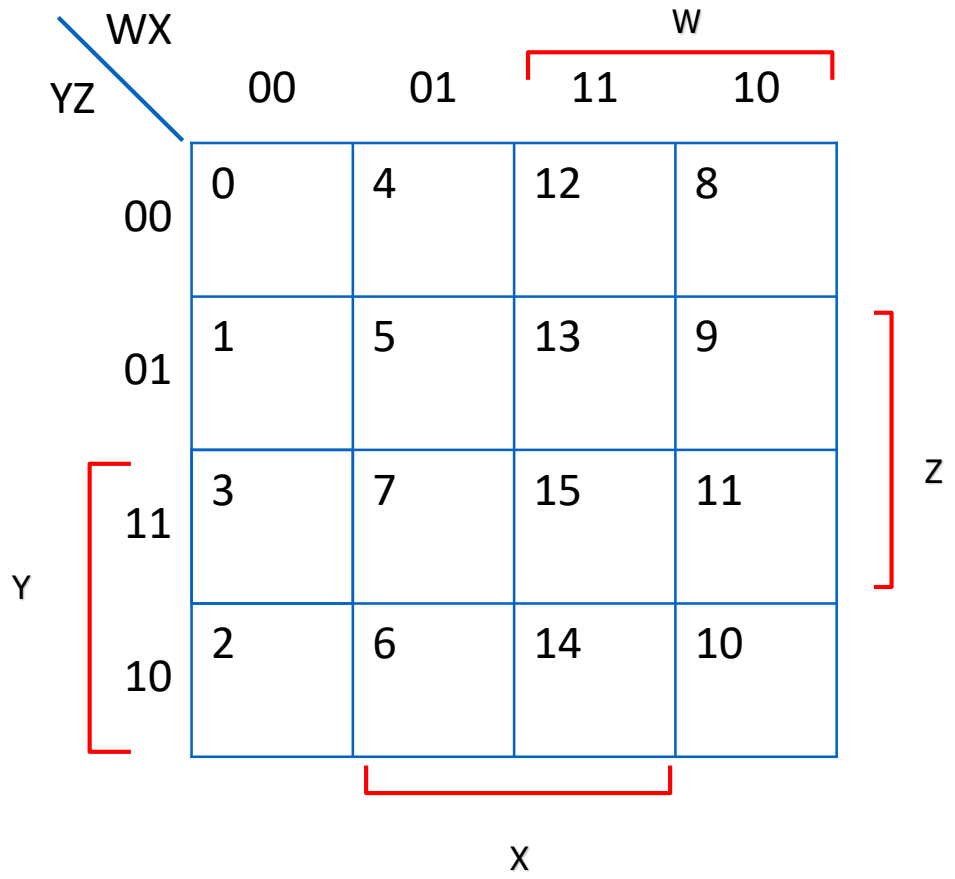
The product term : **XZ'**



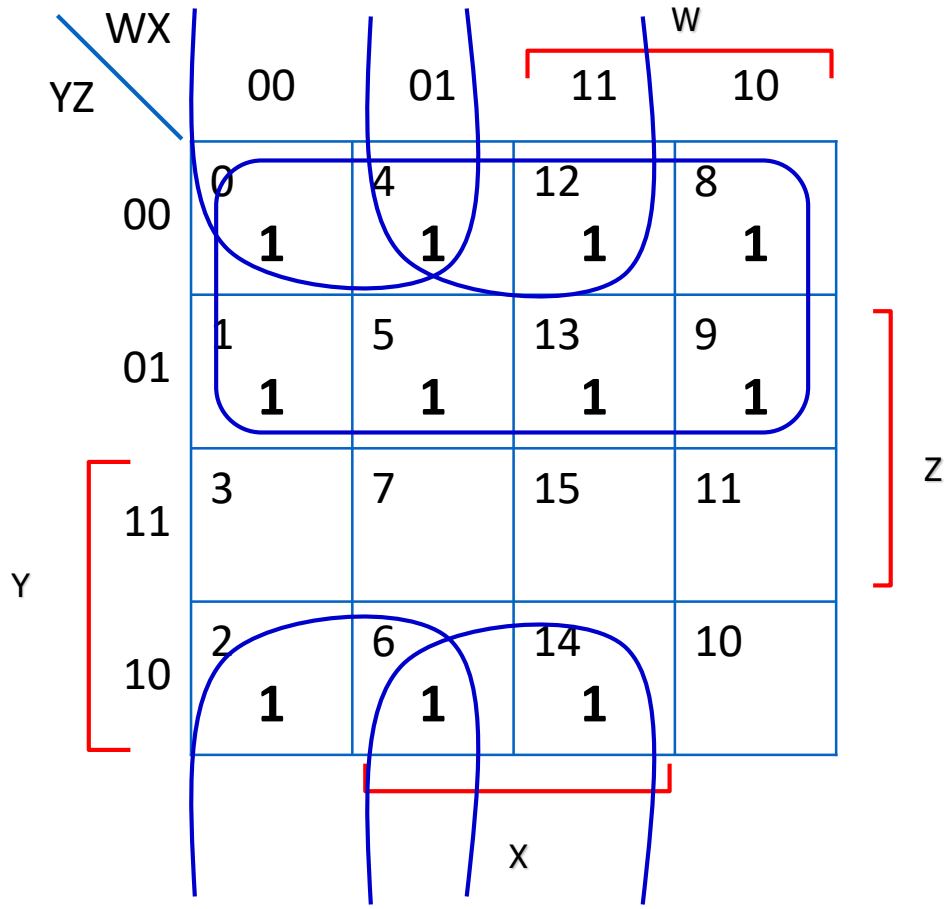
$$F = X'Z + XZ'$$

Exercise

Row	W	X	Y	Z	F
0	0	0	0	0	**1**
1	0	0	0	1	**1**
2	0	0	1	0	**1**
3	0	0	1	1	**0**
4	0	1	0	0	**1**
5	0	1	0	1	**1**
6	0	1	1	0	**1**
7	0	1	1	1	**0**
8	1	0	0	0	**1**
9	1	0	0	1	**1**
10	1	0	1	0	**0**
11	1	0	1	1	**0**
12	1	1	0	0	**1**
13	1	1	0	1	**1**
14	1	1	1	0	**1**
15	1	1	1	1	**0**



Exercise (Contd.)



F = Y′ + W′Z′ + XZ′

Example 7

The prime implicants:

W’X’

W’Y’

W’Z

XYZ

WXY

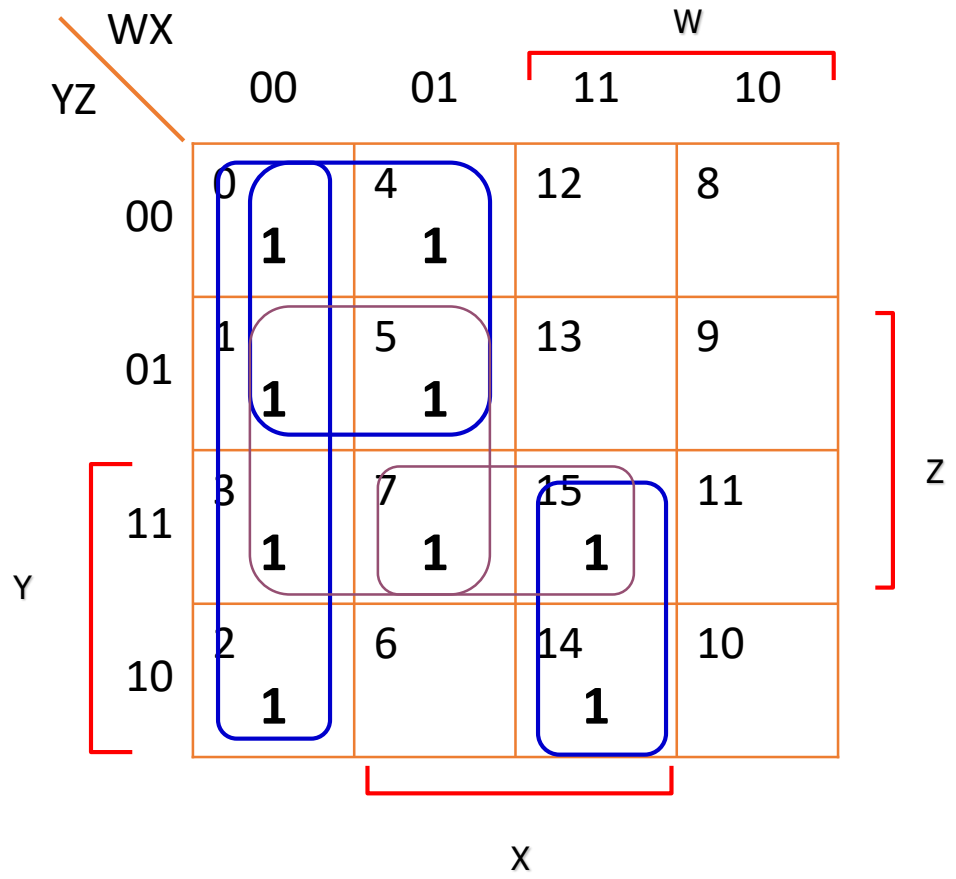
The essential prime implicants:

W’X’

W’Y’

WXY

Cell 7 is not covered by any of the essential prime implicants. Its covered by two non-essential prime implicant. We choose the one with the less number of variables which is W’Z



$$F= W'X'+W'Y'+WXY+W'Z$$

Exercise

- $F_{W,X,Y,Z} = \Sigma\,(0,1,2,3,4,6,7,8,9,12,14)$

Simplifying the Product of Sums

1. **Plot 0s** corresponding to **maxterms** of function.
2. **Circle largest** possible rectangular **sets of 0s**.
   - # of 0s in set must be power of 2
   - OK to cross edges
3. Read off product terms, one per circled set.
   - Variable is 0 → include variable
   - Variable is 1 → include complement of variable
   - Variable is both 1 and 0 → variable not included
4. Circled sets and corresponding product terms are called '**prime implicants**'
5. Minimum number of gates and gate inputs

Don’t Care Conditions

- In some applications, the Boolean function for certain combinations of the input variables is not specified. The corresponding minterms (maxterms) are called “don’t care minterms(maxterms)”.
- In K-map , these are represented by ‘**d**’.
- Since the output function for those minterms(maxterms) is not specified, those minterms(maxterms) could be combined with the adjacent 1 cells (0-cells) to get a more simplified sum-of-products (product-of-sums) expression.

Example 11

- Build a logic circuit that determines if **a decimal digit** is ≥ 5

1. The decimal digits (0,1,2,...,9) are represented by 4 bit BCD code.
2. The logic circuit should have **4 input** variables and **1 output**.
3. There are 16 different input combinations but only 10 of them are used.
4. The logic function should produce 0 if the number is < 5,

   and 1 if it is ≥ 5.

Example 11 (Contd.): Truth Table

Row	W	X	Y	Z	F
0	0	0	0	0	**0**
1	0	0	0	1	**0**
2	0	0	1	0	**0**
3	0	0	1	1	**0**
4	0	1	0	0	**0**
5	0	1	0	1	**1**
6	0	1	1	0	**1**
7	0	1	1	1	**1**
8	1	0	0	0	**1**
9	1	0	0	1	**1**
10	1	0	1	0	**d**
11	1	0	1	1	**d**
12	1	1	0	0	**d**
13	1	1	0	1	**d**
14	1	1	1	0	**d**
15	1	1	1	1	**d**

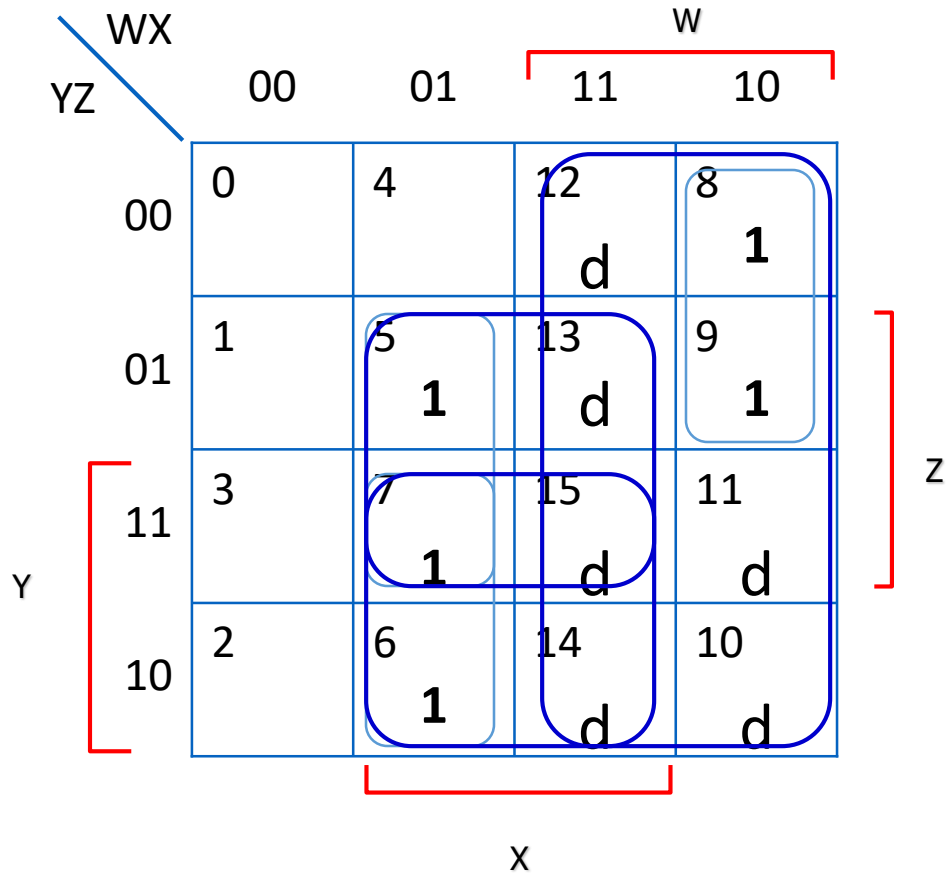
Example 11(Contd.)- K-Map

The Minimal Sum:

Combining the ‘1’ cells only, the minimal sum is:
F = WX’Y’+W’XZ+W’XY

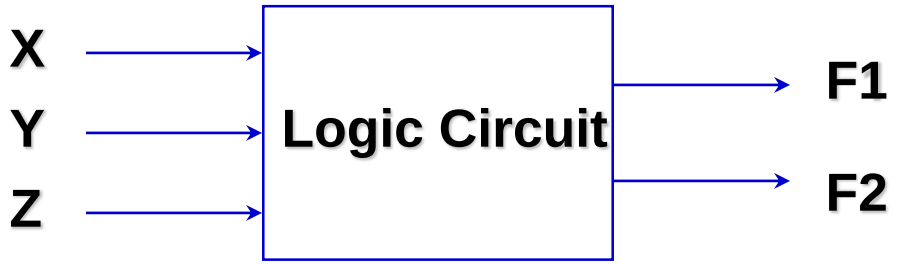
Combining the don’t care minterms with the ‘1’ cells, the minimal sum is:

F = W+XZ+XY



Multiple-Output Minimization

Most digital applications require multiple outputs derived from the same input variables.

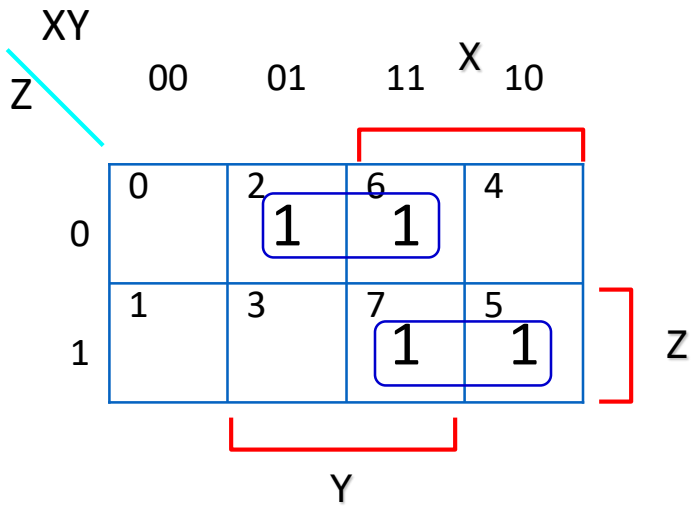


1. Each output function could be minimized using K-map and realized independently.
2. The output functions could share one or more product terms (prime implicant) which reduces the total number of gates.

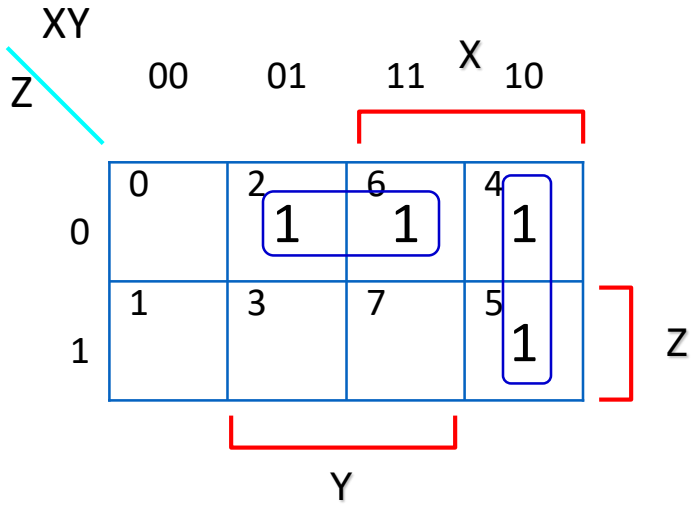
Example 12

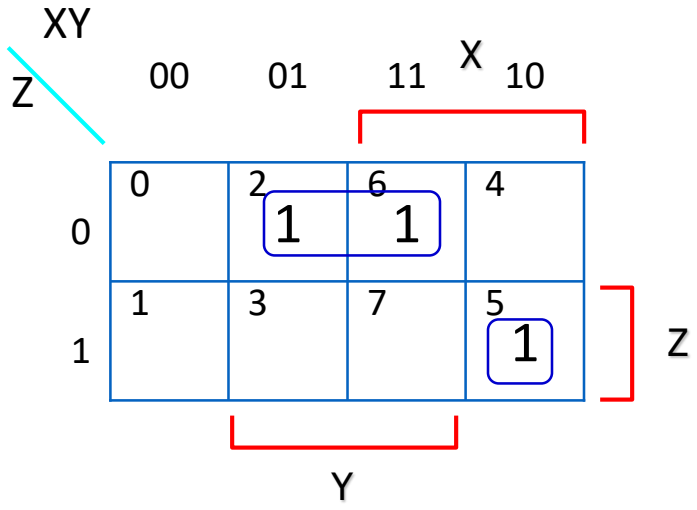
F1 = XZ + YZ'
F2 = XY' + YZ'

F1



F2





Find the common terms

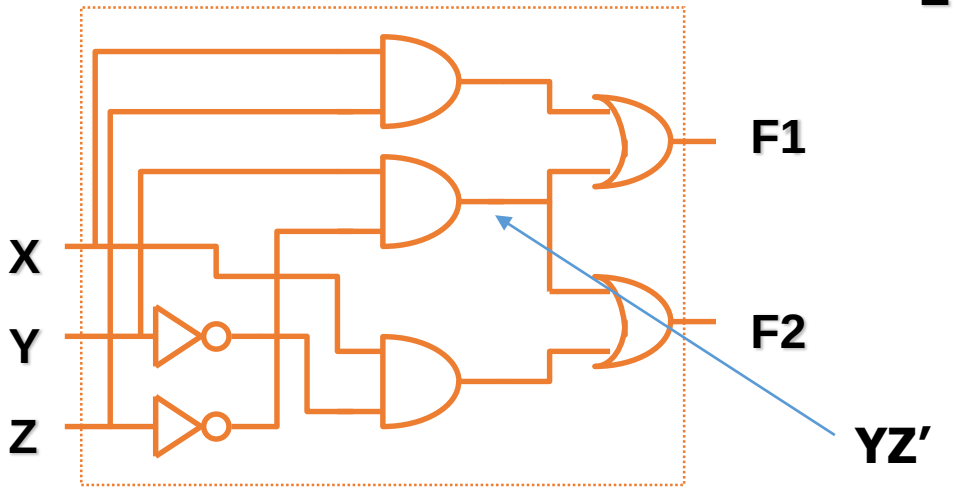
The common term is: **YZ'**

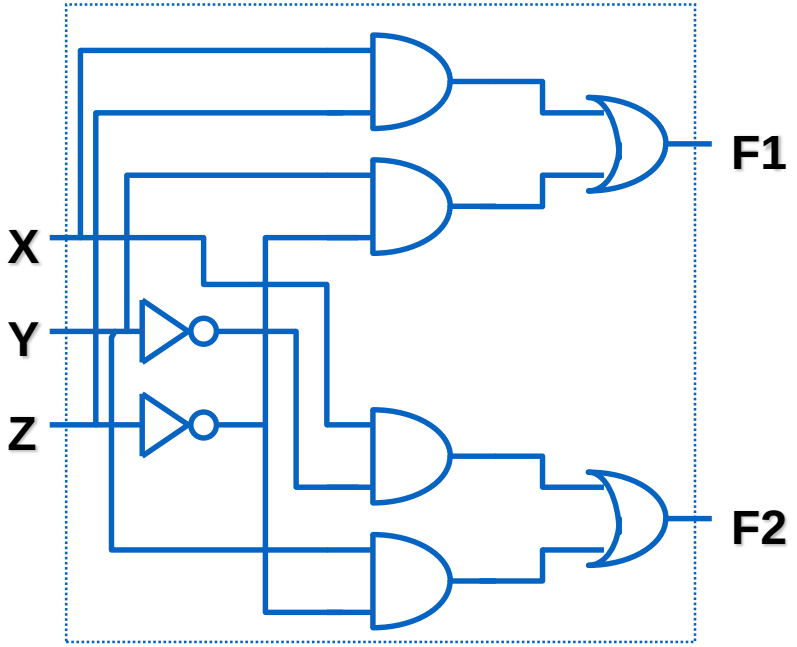
Example 12 (Contd.) – Logic Diagram

Independent realization
Minimal realization

F1= XZ+YZ′

F2= XY′+YZ′





Timing Hazards

The Truth Table determines the Steady State behavior of a Combinational Logic Circuit

Transient behavior:

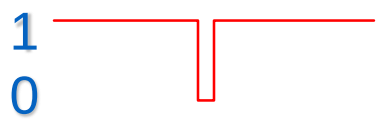
1. Output could produce **glitches** when input variables change.
2. Glitches occur when the paths between inputs and output have different delays.
3. Timing **Hazards** refer to the possibility of having glitches during input transitions.

Hazards:

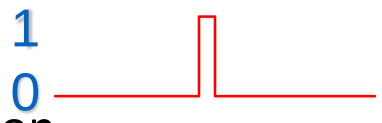
1. Definitions.
2. Finding hazards.
3. Eliminating hazards.

Definitions

- **Static-1 Hazard**: Two input combinations that:
  - Differ in only one variable.
  - Both produce logic 1.
  - Possibly produce Logic 0 glitch during input variable transition



- **Static-0 Hazard**: Two input combinations that
  - Differ in only one variable
  - Both produce logic 0
  - Possibly produce Logic 1 glitch during input variable transition



- **Dynamic hazards**:

  The output could change more than once during input transitions

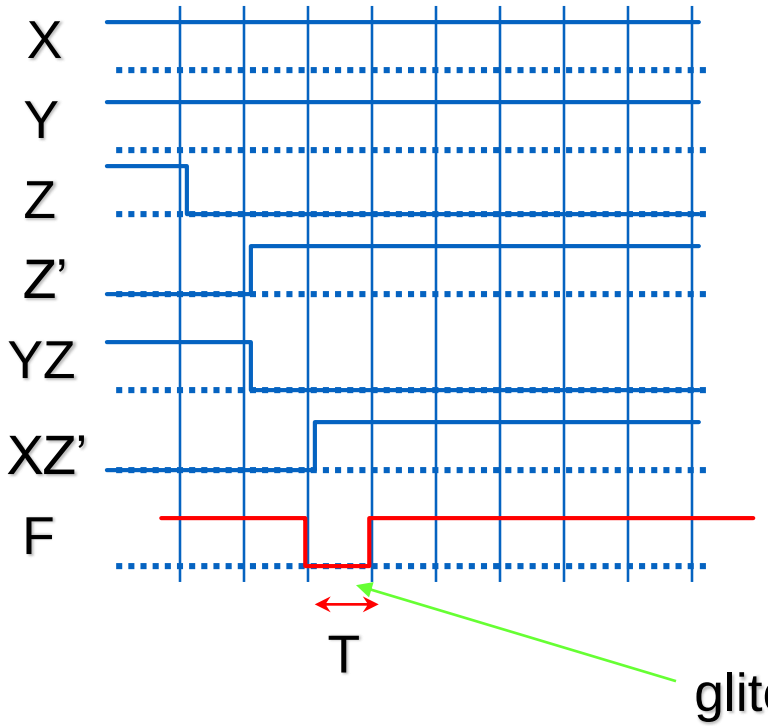
  Caused by multiple paths with different delays from input to the output

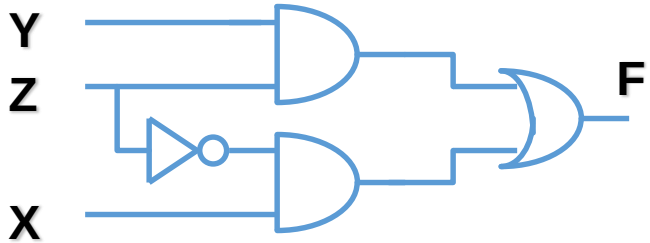
Example

F = YZ + XZ’

Delay in each gate is T .

Input changes from XYZ = 111 to 110





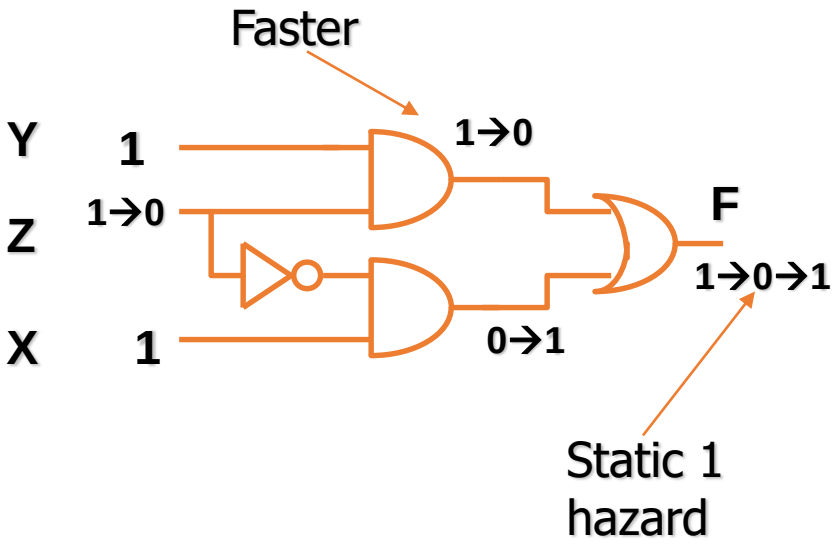
Finding Timing hazards using K-map

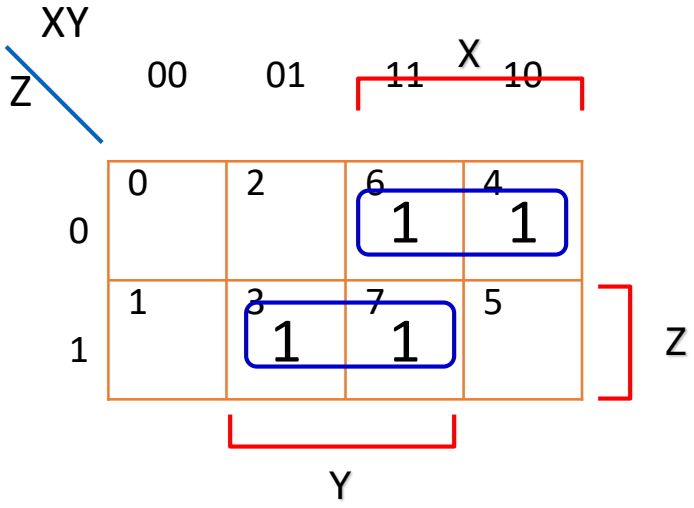
AND-OR Circuits:

Static 0 hazards do not exist in the sum-of products (AND-OR) implementation.

Static 1 hazards are possible

The K-map of the function F in the previous example:
Cell 6 and 7 are covered in two product terms.





Timing hazards in OR-AND Circuits

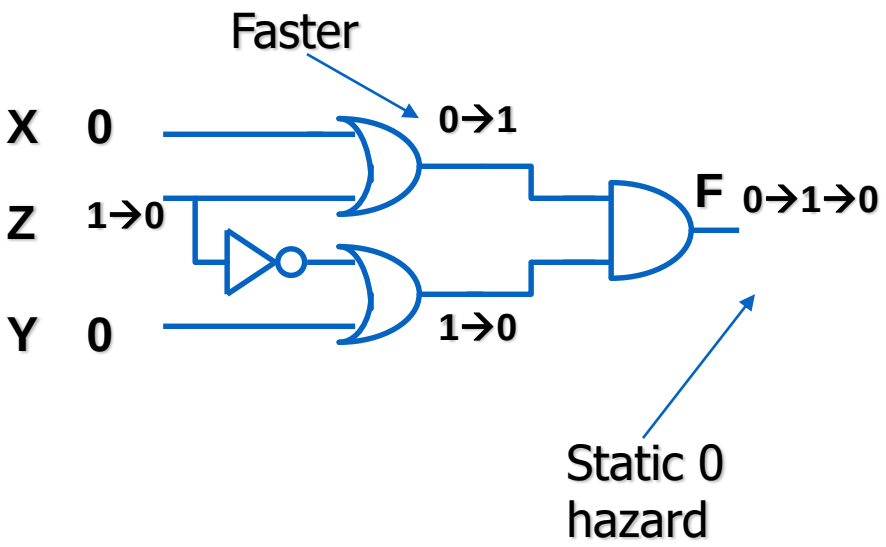
OR - AND Circuits:

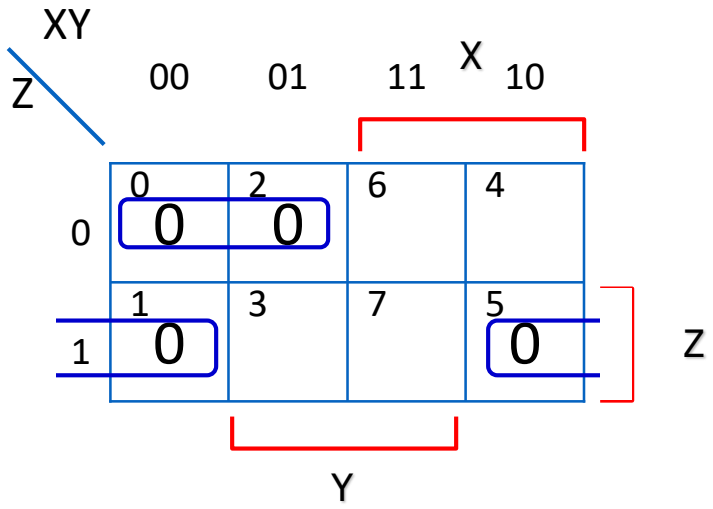
Static 1 hazards do not exist in the products-of sum (OR-AND) implementation.

Static 0 hazards are possible

The minimal product of F = (X+Z)(Y+Z')
Cell 0 and Cell 1 are covered in two sum terms

Static 0 hazard occurs when inputs switched between 000 to 001

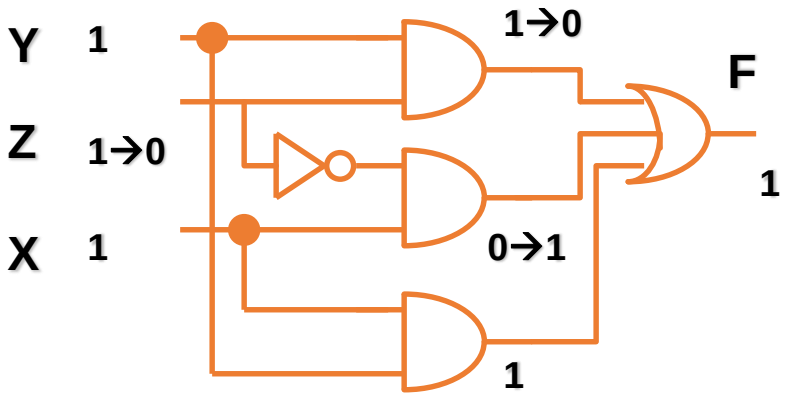




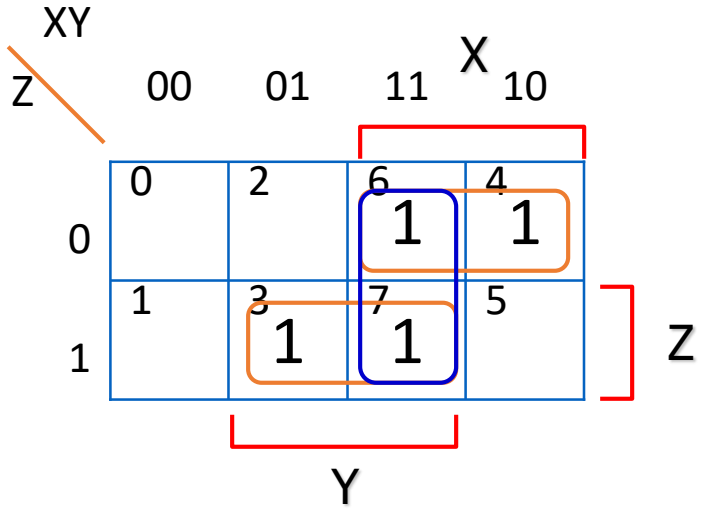
Eliminating Timing Hazards

AND-OR Circuit

Add a prime implicant that combines the two inputs that cause static 1 hazard.



The hazard-free circuit

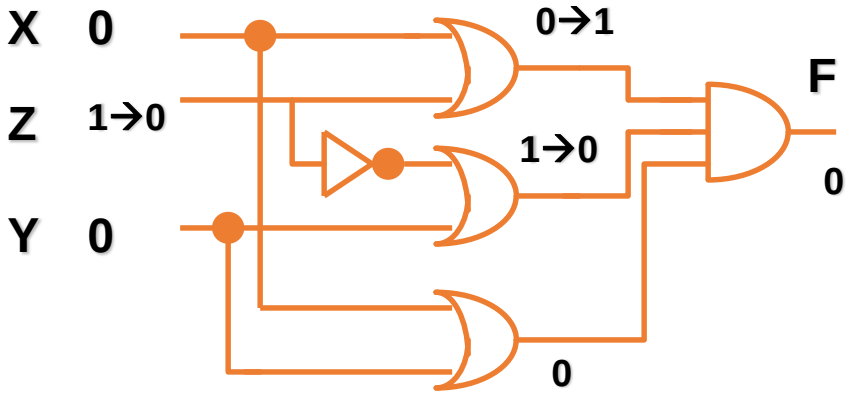


Cells 6 & 7 are combined: XY

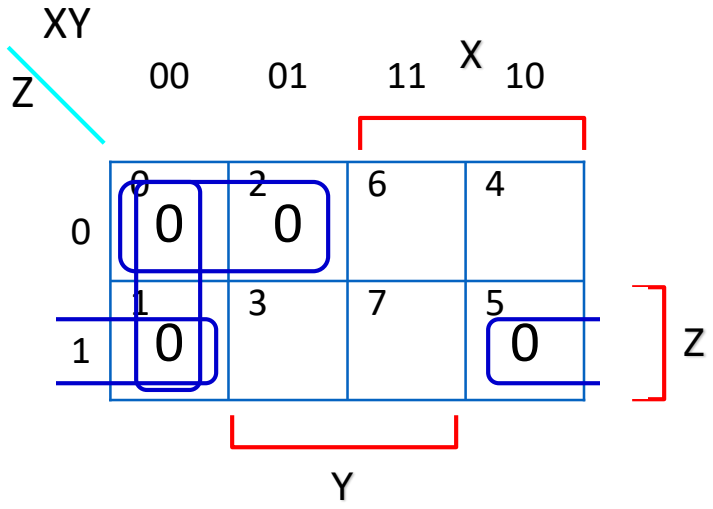
Eliminating Timing Hazards

OR - AND Circuit

Add a prime implicant that combines the two inputs that cause static 0 hazard.



The hazard-free circuit

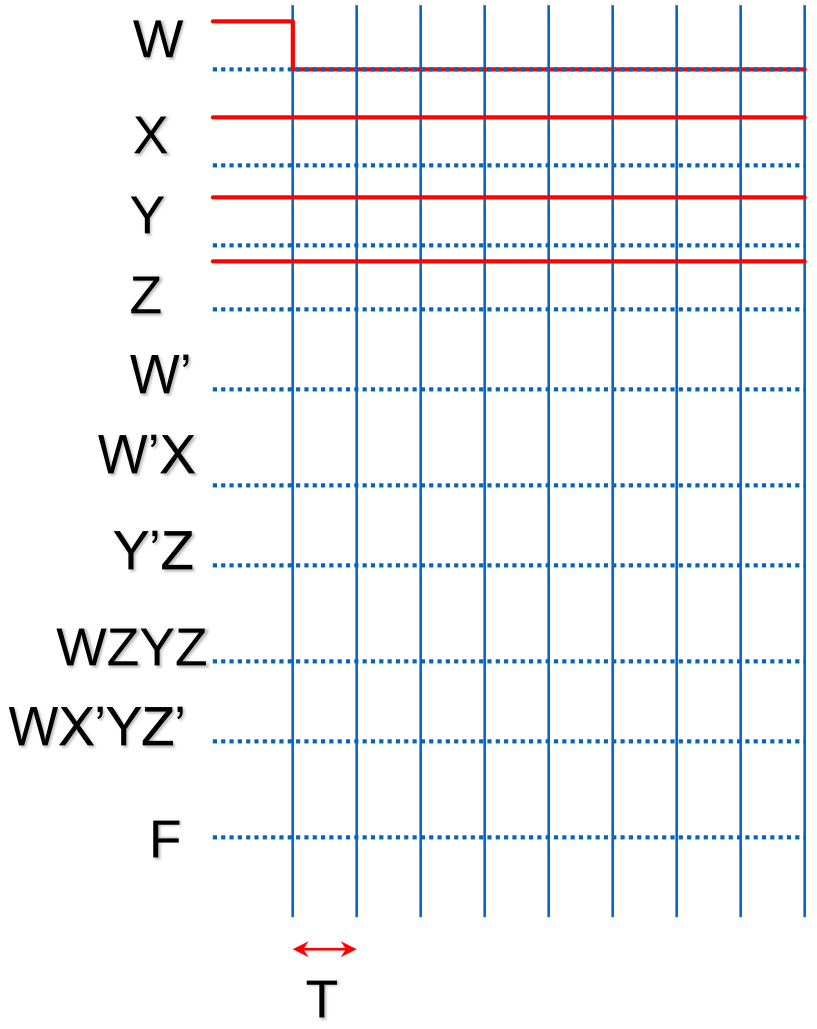


Cells 0 & 1 are combined: X+Y

Assignment

F = W'X + Y'Z + WXYZ + WX'YZ'

1. Complete the timing diagram. (Assume all gates has same propagation delay=T)
2. Use K-map to show Static-0 hazards
3. Build a hazard-free circuit

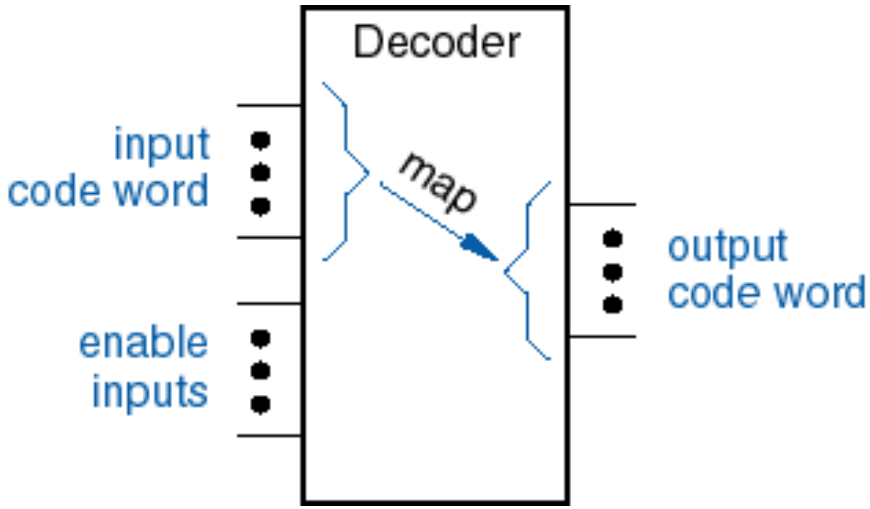


Decoders

Converts input code words into output code words.

One-to-One mapping: Each input code produces **only one** output code.

Binary Code
Gray Code
BCD Code
Any Code…

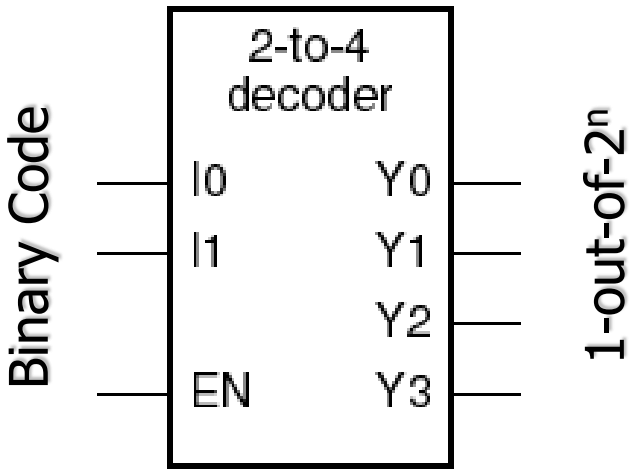


Typically ***n* inputs**, **2^n outputs**: 2-to-4, 3-to-8, 4-to-16, etc.

Binary 2-to-4 decoder

n-to-2^n decoder: n inputs and 2^n outputs.

Example: 2-to-4 decoder

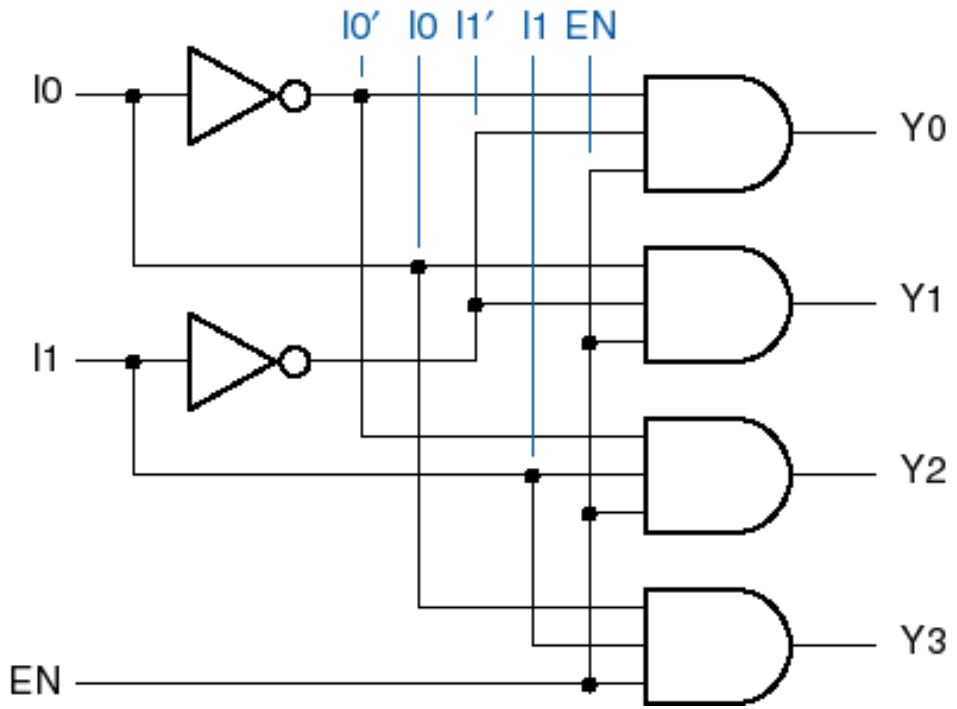


Inputs			Outputs			
EN	I1	I0	Y3	Y2	Y1	Y0
0	X	X	0	0	0	0
1	0	0	0	0	0	**1**
1	0	1	0	0	**1**	0
1	1	0	0	**1**	0	0
1	1	1	**1**	0	0	0

X: don't care

One output is asserted for each input code.

2-to-4- Decoder Logic Diagram



Decoder Applications

Microprocessor **memory** systems

- selecting different banks of **memory**

Microprocessor **input/output** systems

- selecting different devices

Microprocessor **instruction decoding**

- enabling different functional units

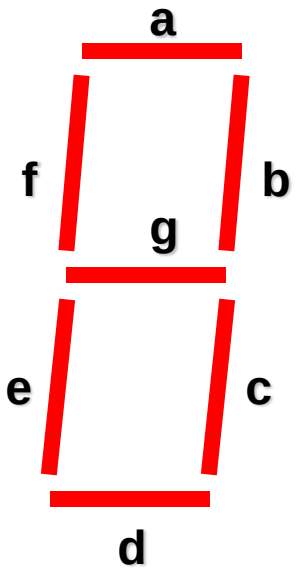
Memory chips

- enabling different rows of memory depending on address

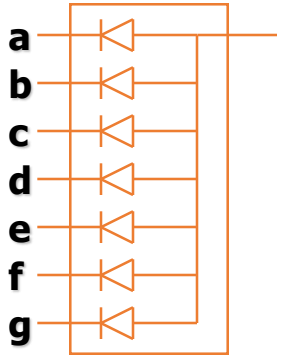
Seven-Segment Displays

Displays decimal numbers and some characters

LED (Light Emitting Diode) or **LCD** (Liquid Crystal Display)

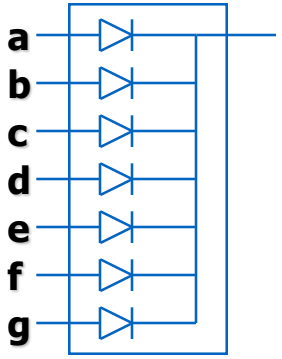


CommonAnode(CA)



requires Active Low inputs

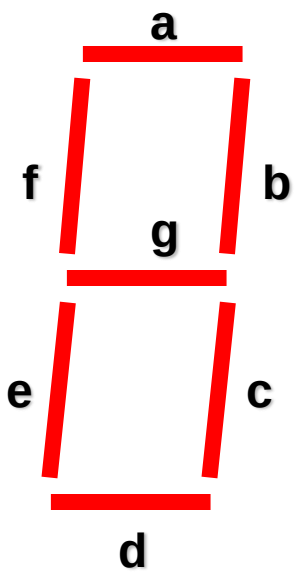
CommonCathode(CC)



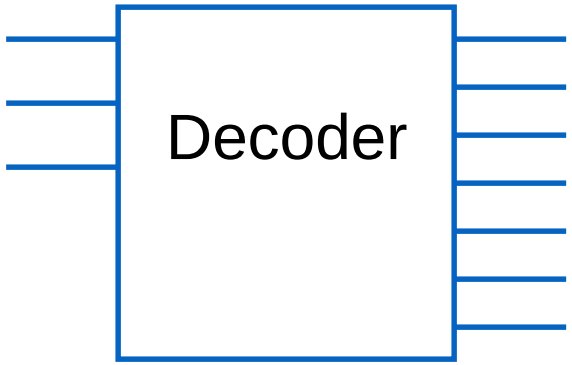
requires Active High inputs

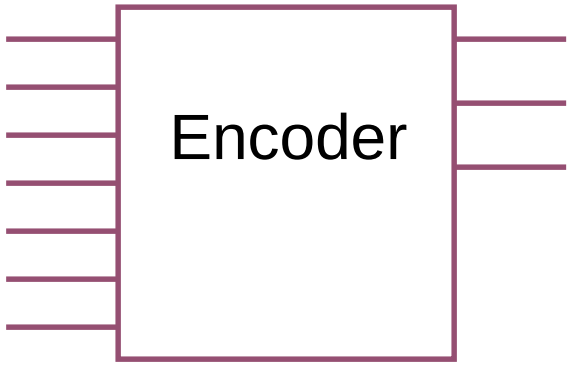
Seven-Segment Decoders/Drivers

BCD Code				Seven-Segment Code						
Input				Output						
D	C	B	A	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	0	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	0	0	1	1



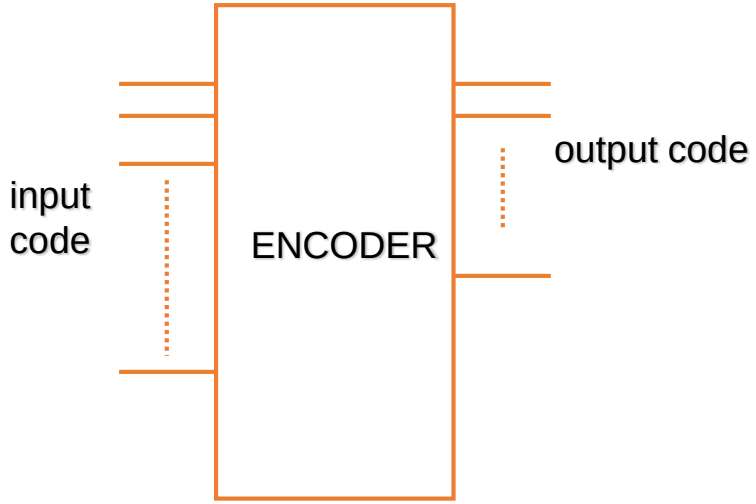
Encoders vs. Decoders





Inverse function of a Decoder.

Outputs are less than inputs.

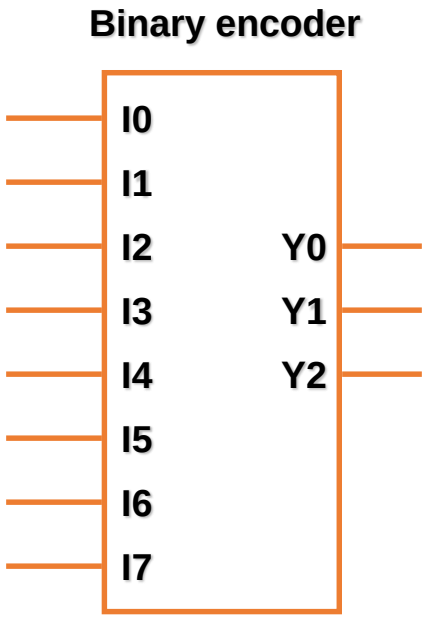


Converts input code words into output code words.

Binary Encoders

2^n-to-n encoder: 2^n inputs and n outputs.

Example: n=3, 8-to-3 encoder



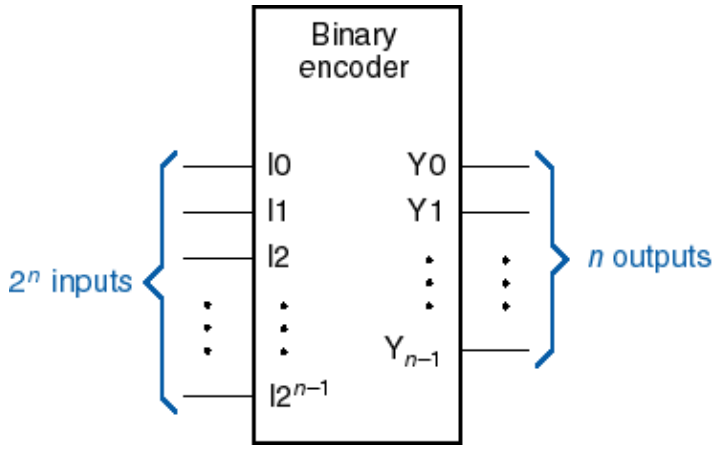
1-out-of-2^n Inputs								Binary Code Outputs		
I0	I1	I2	I3	I4	I5	I6	I7	Y0	Y1	Y2
1	0	0	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0	1
0	0	**1**	0	0	0	0	0	0	1	0
0	0	0	**1**	0	0	0	0	0	1	1
0	0	0	0	**1**	0	0	0	1	0	0
0	0	0	0	0	**1**	0	0	1	0	1
0	0	0	0	0	0	**1**	0	1	1	0
0	0	0	0	0	0	0	**1**	1	1	1

Y2 = I1 + I3 + I5 + I7

Y1 = I2 + I3 + I6 + I7

Y0 = I4 + I5 + I6 + I7

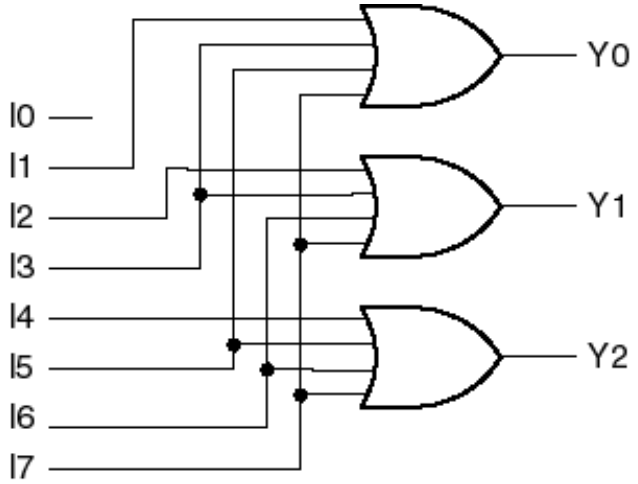
Binary Encoders



Y2 = I1 + I3 + I5 + I7

Y1 = I2 + I3 + I6 + I7

Y0 = I4 + I5 + I6 + I7



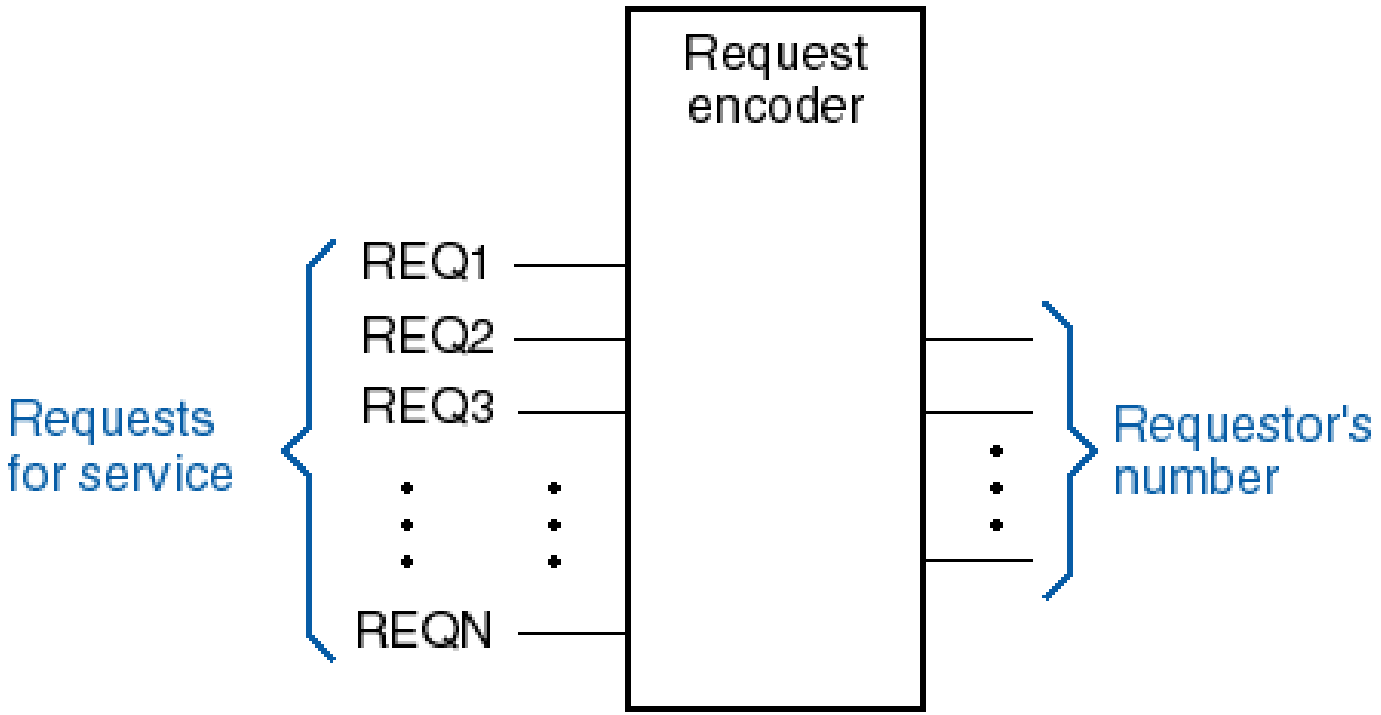
Limitations:

- Only one input can be activated

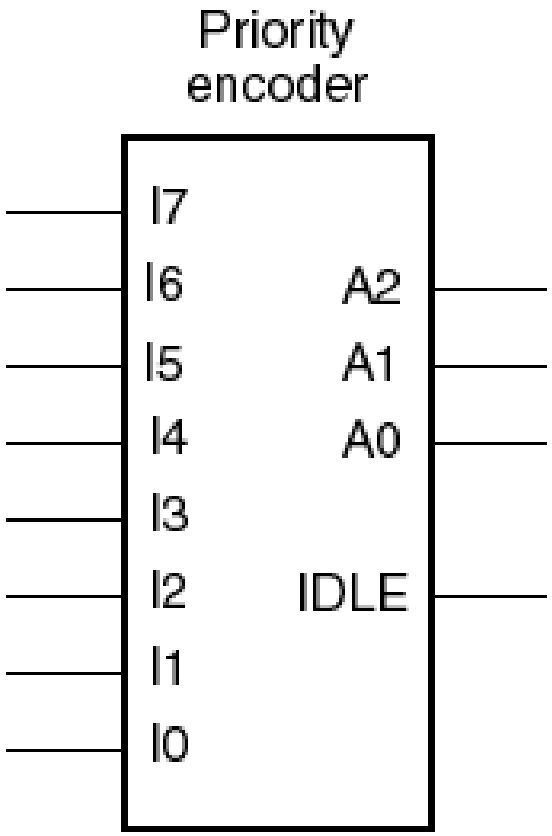
Application:

- Handling multiple devices requests. But, no simultaneous requests.
- Establishing **priorities** solve the problem of multiple requests.

Need Priority in Most Applications



8-Input Priority Encoder



1. Assign priorities to the inputs
2. When more than one input are asserted, the output generates the code of the input with the highest priority

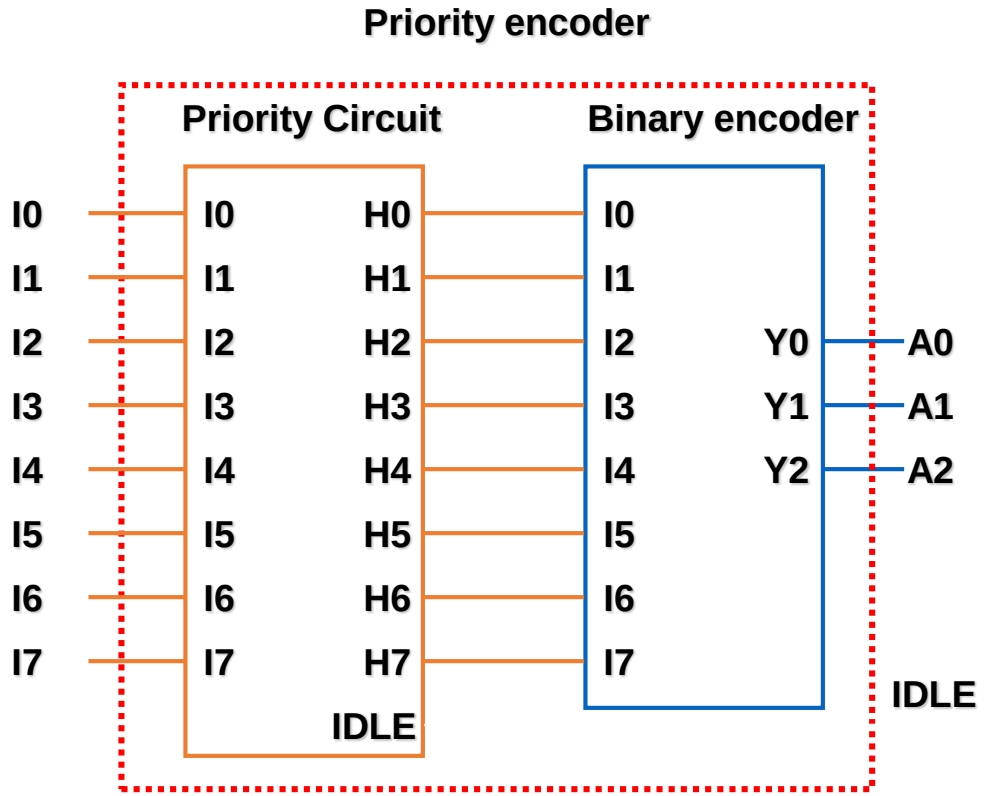
Priority-Encoder Logic Equations

(Highest Priority)

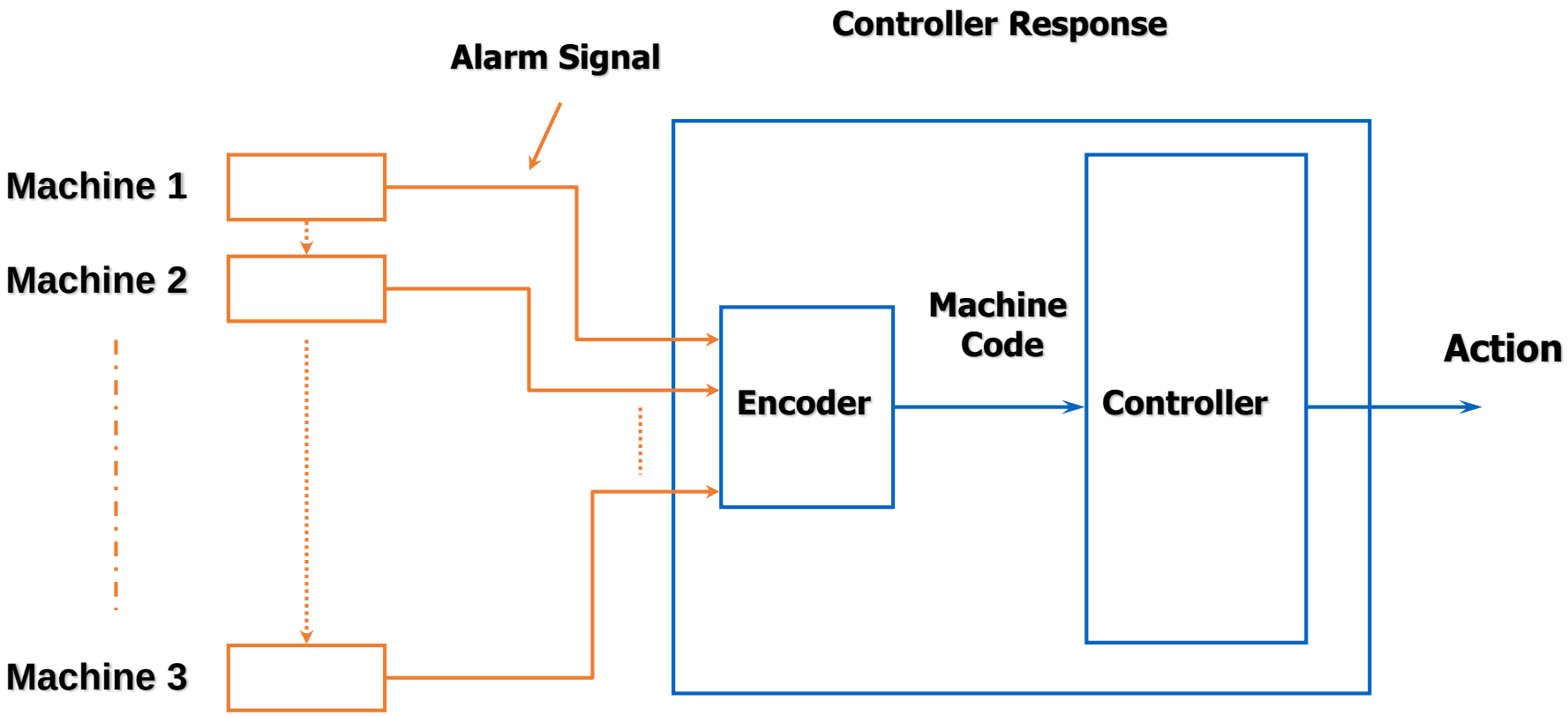
H7 = I7
H6 = I6.I7’
H5 = I5.I6’.I7’
H4 = I4.I5’.I6’.I7’
H3 = I3.I4’.I5’.I6’.I7’
H2 = I2.I3’.I4’.I5’.I6’.I7’
H1 = I1. I2’.I3’.I4’.I5’.I6’.I7’
H0 = I0.I1’. I2’.I3’.I4’.I5’.I6’.I7’

IDLE= I0’.I1’.I2’.I3’.I4’.I5’.I6’.I7’

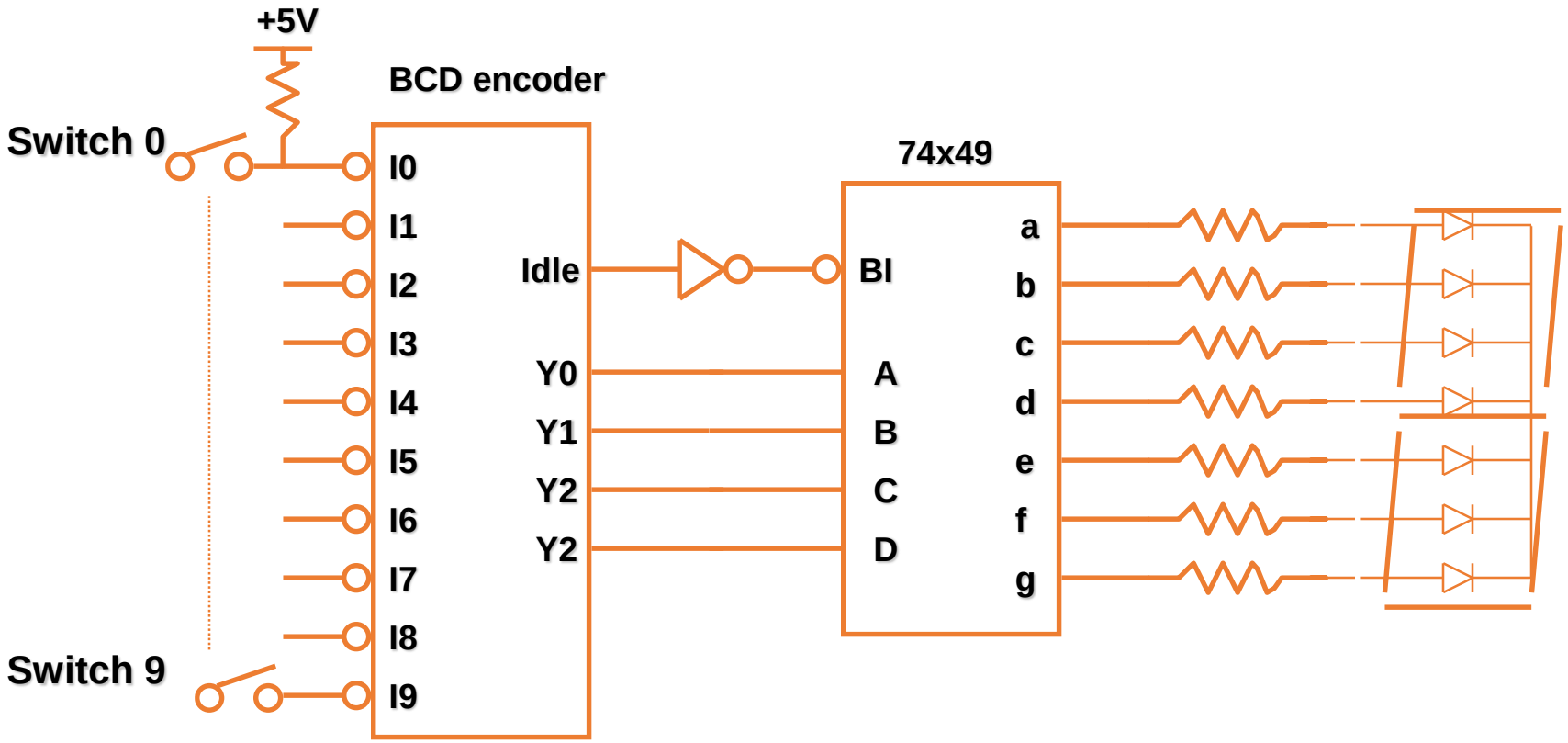
A2 = I1 + I3 + I5 + I7
A1 = I2 + I3 + I6 + I7
A0 = I4 + I5 + I6 + I7



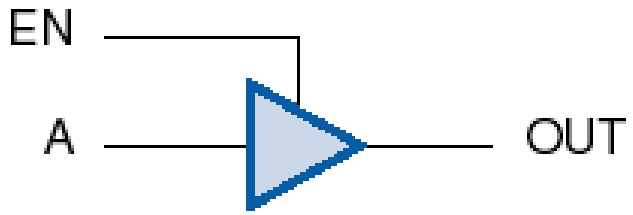
Encoder Application (Monitoring Unit)



BCD Encoder


+5V
BCD encoder
Switch 0
Switch 9
I0
I1
I2
I3
I4
I5
I6
I7
I8
I9
Idle
Y0
Y1
Y2
Y2
74x49
BI
A
B
C
D
a
b
c
d
e
f
g

Three-State Buffers



Output = LOW, HIGH, or Hi-Z.

Hi-Z: The output is floating (High Impedance) when the enable input is deasserted →

The input is isolated from the output

EN	A	OUT
L	L	Hi-Z
L	H	Hi-Z
H	L	L
H	H	H

Application:

- Can tie multiple outputs together, if at most one at a time is driven.
- Controlling the access of **a single line/bus** by multiple devices.

Different Types

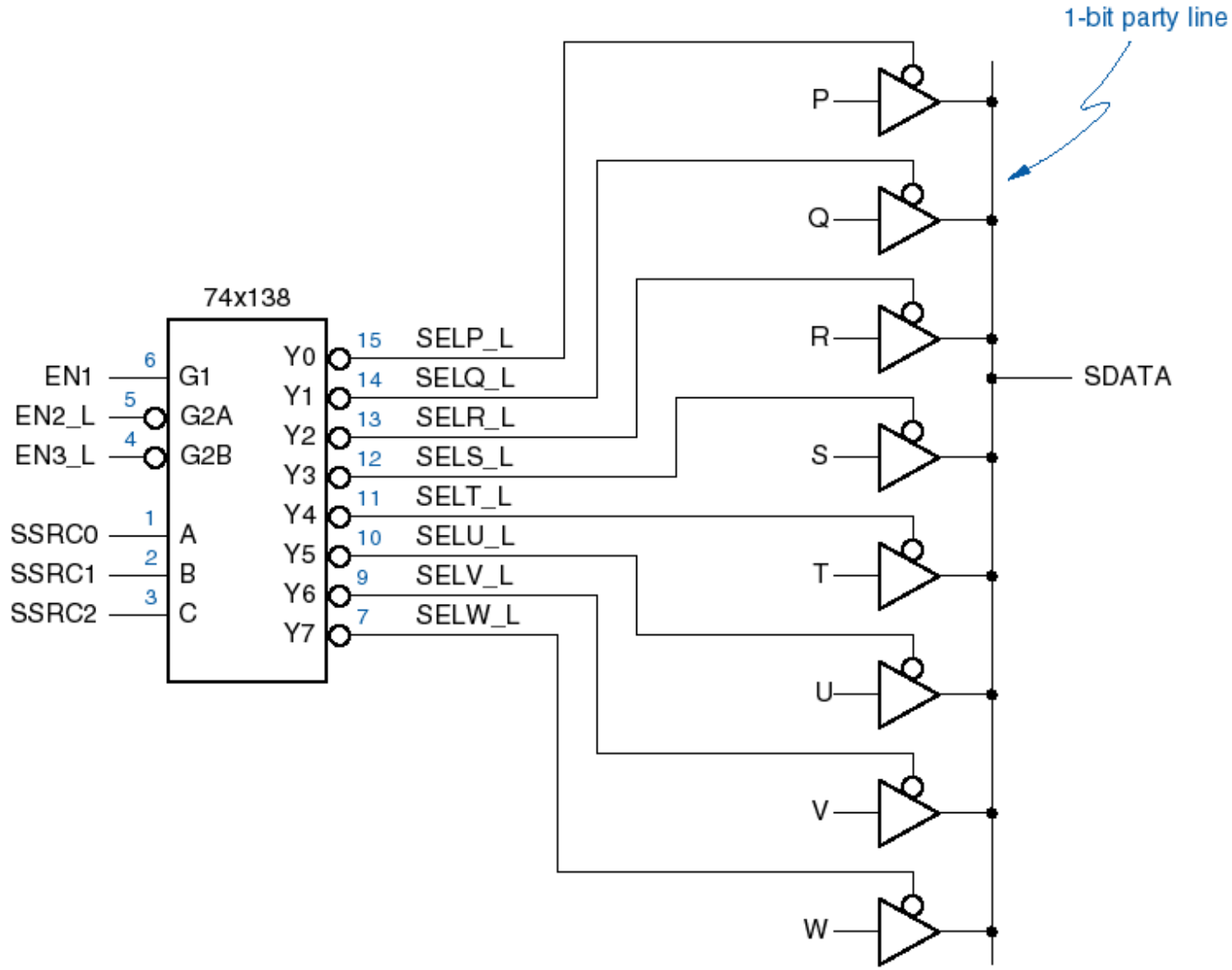
Active High Enable Active Low Enable

Buffers

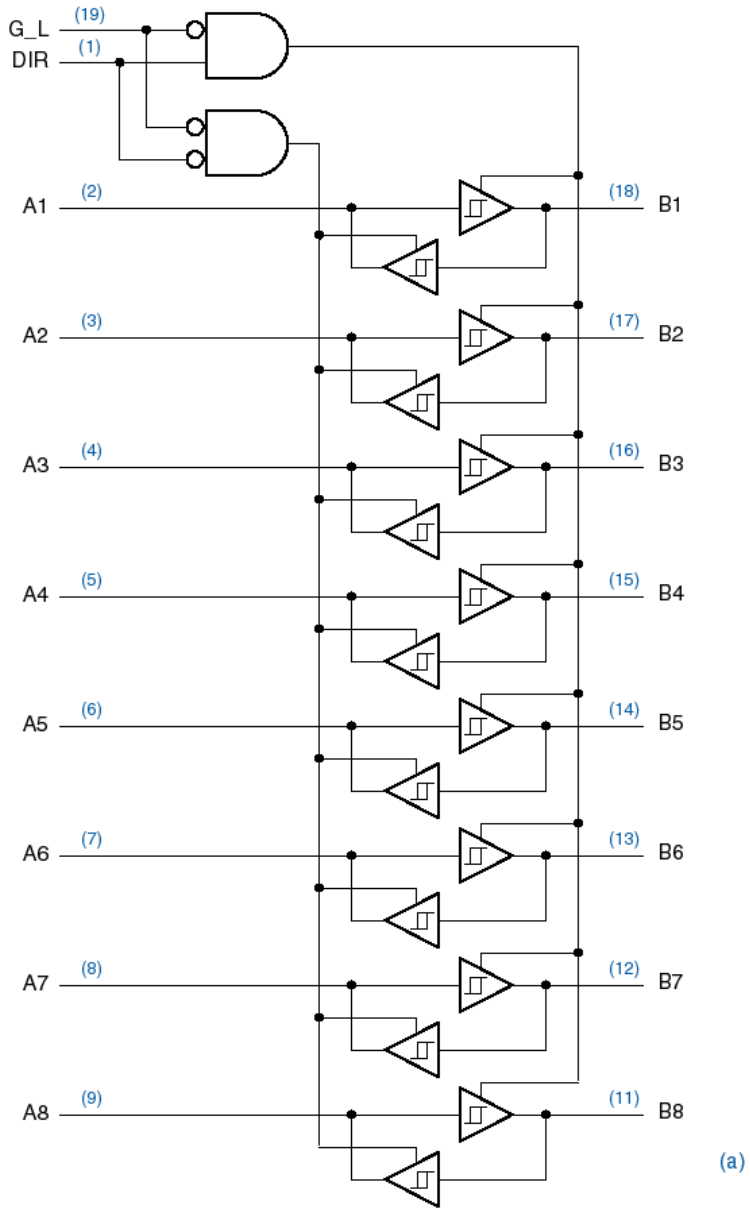
Active High Enable Active Low Enable

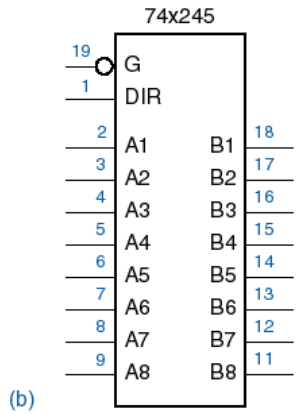
Inverters

8 Data Sources Sharing One Line



Three-State Transceiver

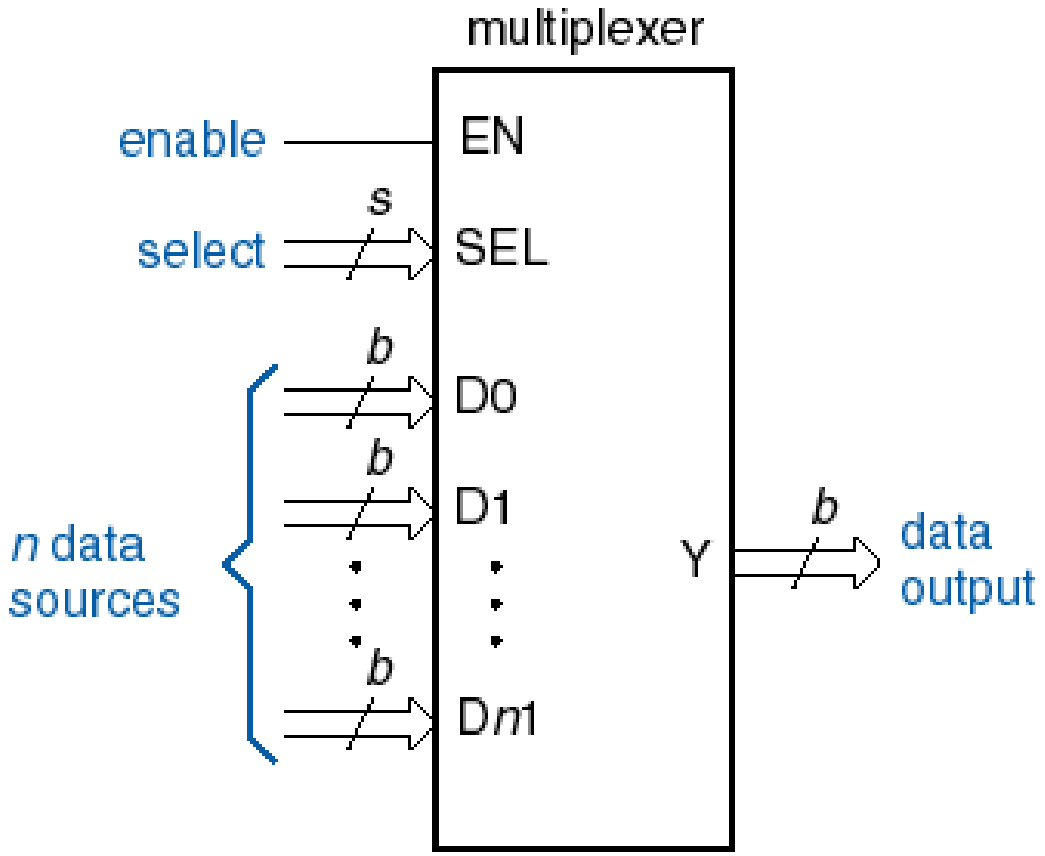




Multiplexers

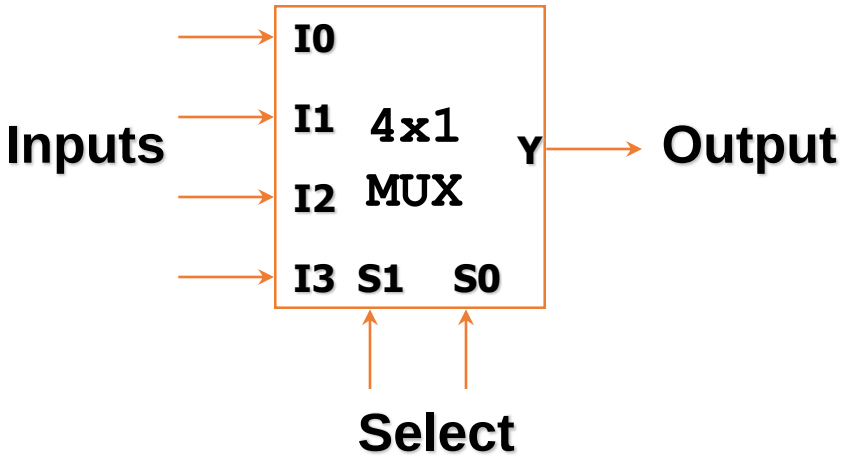
- **Multiplexing:** transmitting large number of signals over a small number of channels or lines
- Digital multiplexer (**MUX**): selects one of many input lines and directs it to a single output.
- Selection lines controls the selection of a particular input
- n selection lines, 2^n inputs, single output

Multiplexers

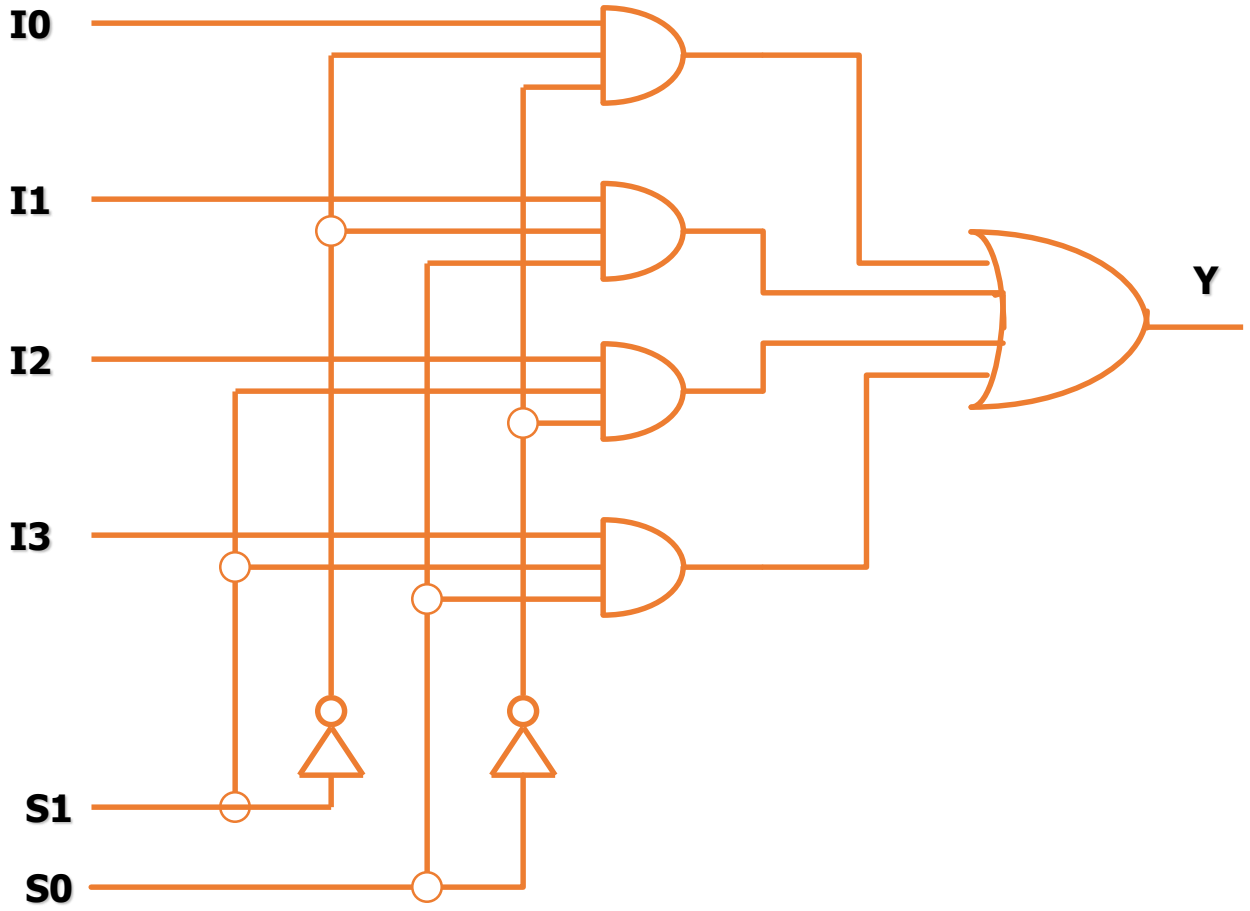


4-to-1 Line Multiplexer

S1	SO	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3



Logic Diagram of 4-to-1 Multiplexer

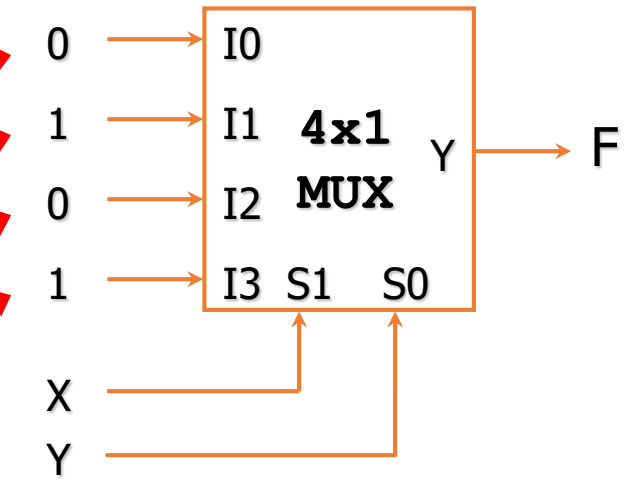


Implementing Logic Functions

- n- variable logic function can be implemented using 2^n-to-1 MUX
- The inputs variables are connected to the select input.
- The function value for each input combination (0 or 1) is connected to the corresponding input of the MUX

Example: $F = \Sigma_{x,y}(1,3)$

Row	X	Y	F
0	0	0	0
1	0	1	1
2	1	0	0
3	1	1	1



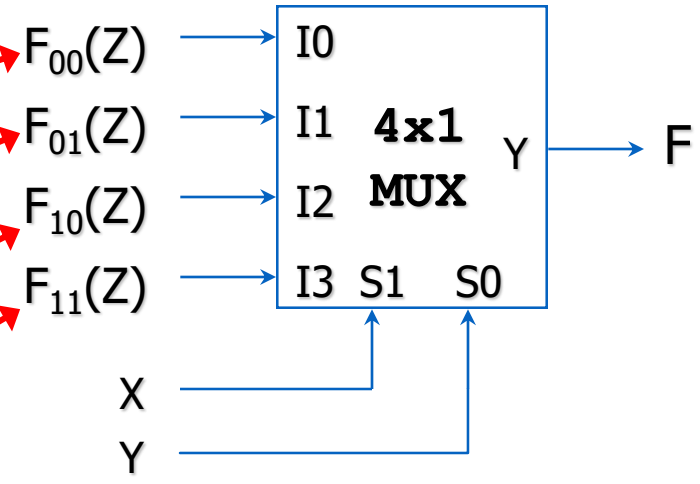
Example: F = Σx,y,z (2,4,7)

Row	X	Y	Z	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

Functional Decomposition

- An effective way for using MUX to implement Logic Functions.
- n-row truth table can be implemented using **n/2-to-1 MUX:**
  - Write the Logic function in terms of the least significant input variable.
  - The truth table is reduced by one half.
- For 3-variable Logic Function, the decomposed truth table is:

Row	X	Y	Z	F
0,1	0	0	X	$F_{00}(Z)$
2,3	0	1	X	$F_{01}(Z)$
4,5	1	0	X	$F_{10}(Z)$
6,7	1	1	X	$F_{11}(Z)$

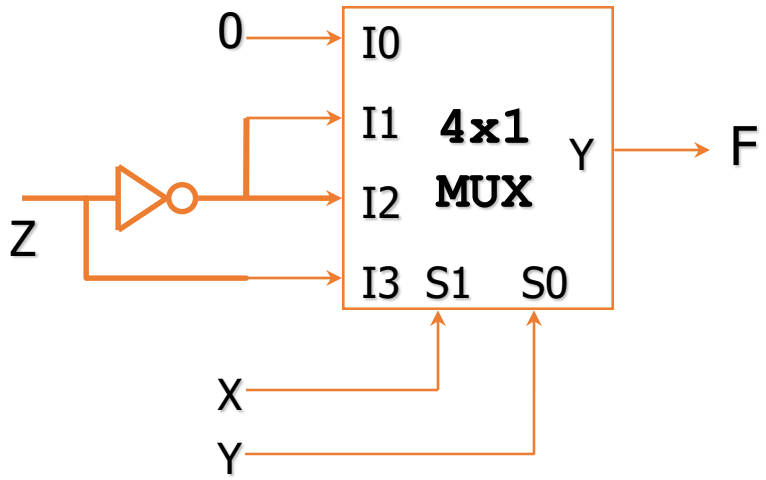


Functional Decomposition Example

Truth Table → Decomposed Truth Table

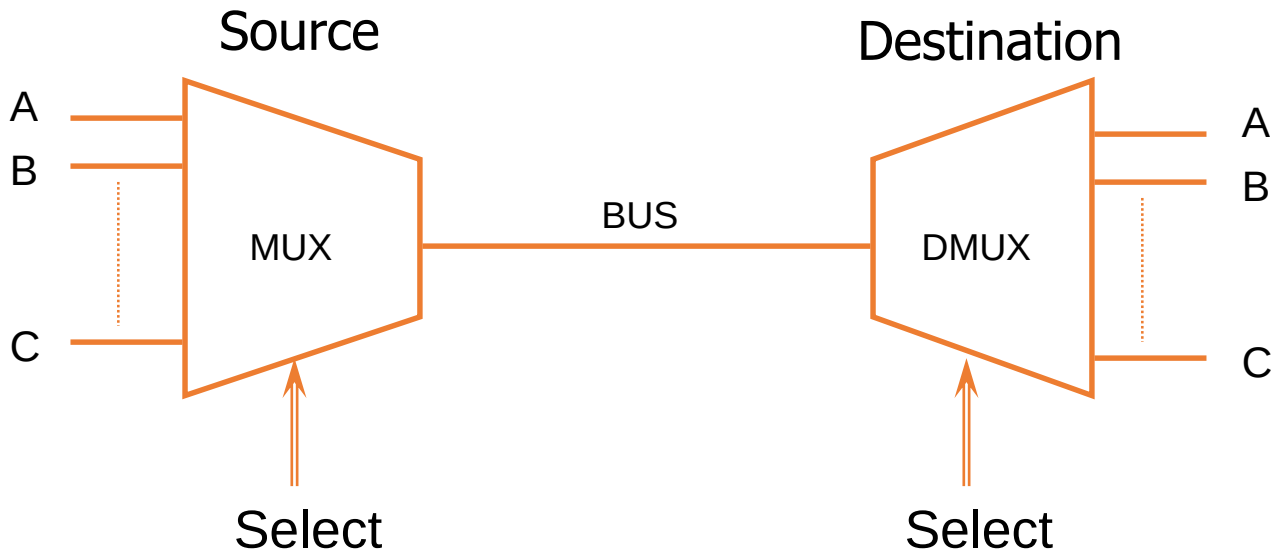
Row	X	Y	Z	F	
0	0	0	0	0	$F_{00}(Z) = 0$
1	0	0	1	0	
2	0	1	0	1	$F_{01}(Z) = Z'$
3	0	1	1	0	
4	1	0	0	1	$F_{10}(Z) = Z'$
5	1	0	1	0	
6	1	1	0	0	$F_{11}(Z) = Z$
7	1	1	1	1	

Row	X	Y	Z	F
0,1	0	0	0	0
2,3	0	1	X	Z’
4,5	1	0	X	Z’
6,7	1	1	X	Z



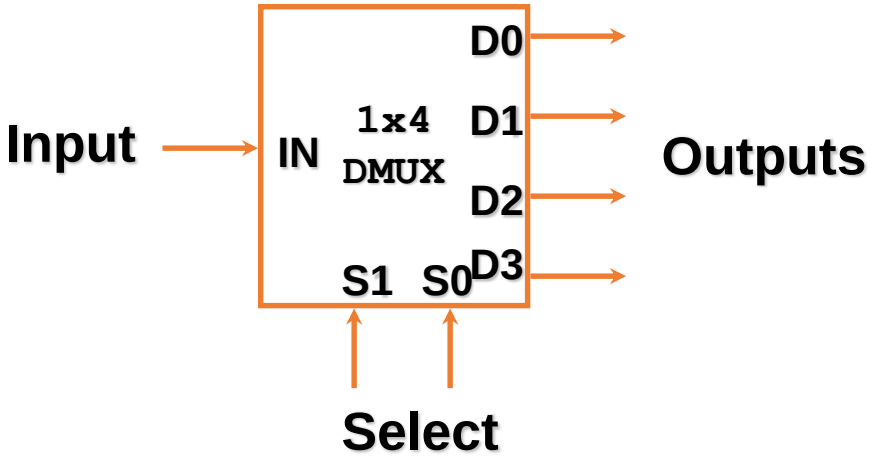
Demultiplexers

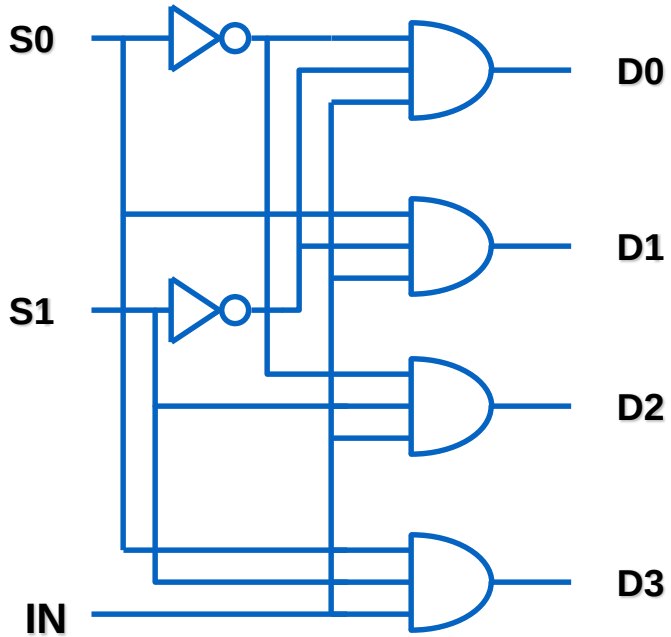
- Demultiplexer (**DMUX**) performs the opposite function of a MUX.
- A digital Demultiplexer receives input data on a single input and transmits it on one of 2^n possible outputs according to the value of the n select inputs
- MUX/DMUX are used in data transmission



1-to-4 DMUX

IN	S1	S2	D0	D1	D2	D3
X	0	0	IN	0	0	0
X	0	1	0	IN	0	0
X	1	0	0	0	IN	0
X	1	1	0	0	0	IN

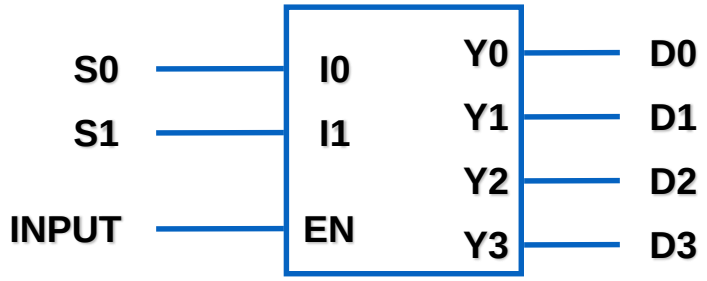




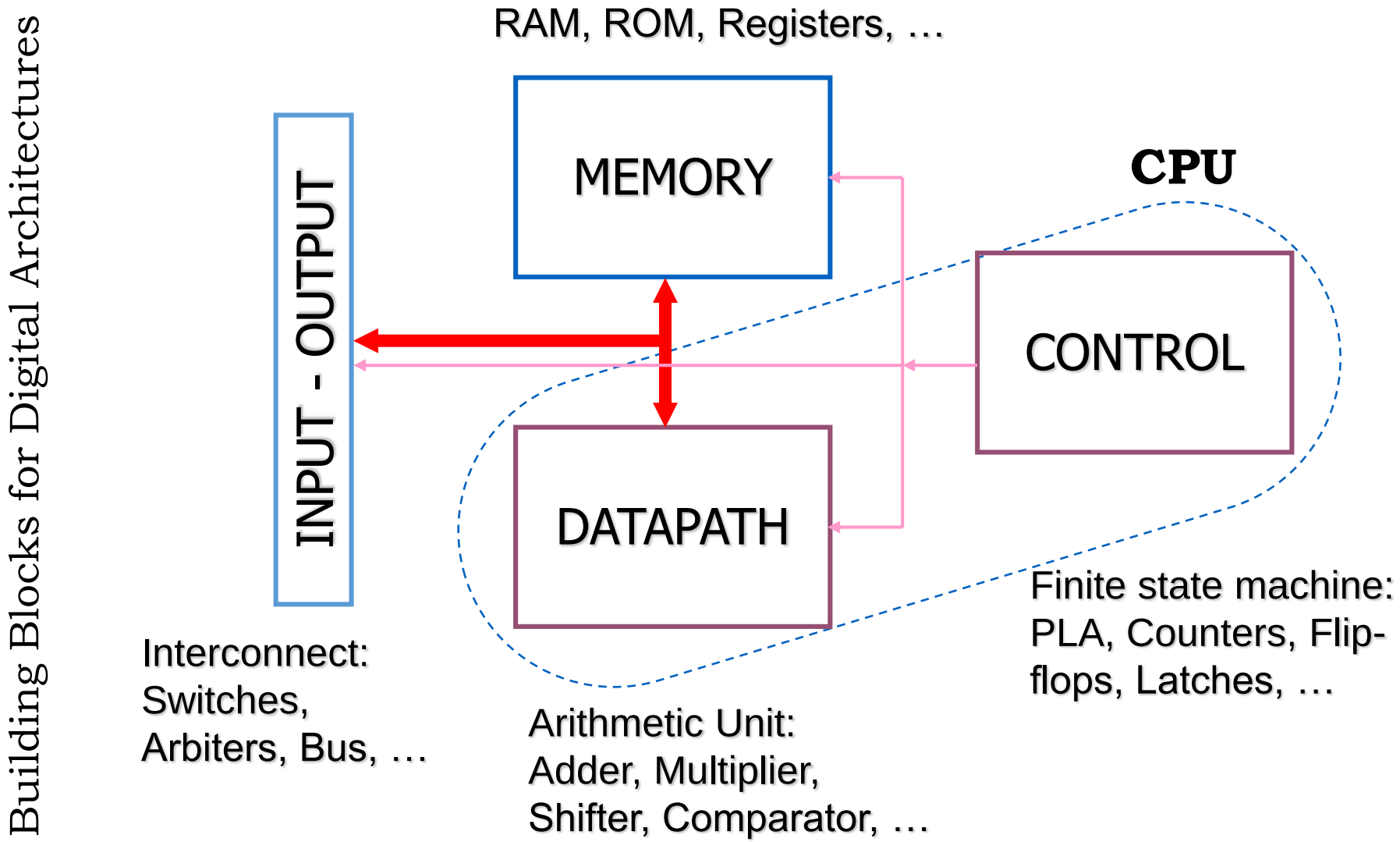
Using Decoders as DMUX

- A DMUX has the same structure of a Decoder with enable input.
- Decoder can be used as a DMUX by connecting the input data to the enable input.
- **Example**: 2-to-4 Decoder can be used as 1-to-4 DMUX

IN	S1	S2	D0	D1	D2	D3
X	0	0	**IN**	0	0	0
X	0	1	0	**IN**	0	0
X	1	0	0	0	**IN**	0
X	1	1	0	0	0	**IN**



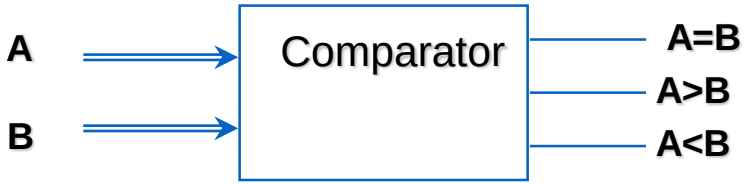
A Generic Digital Processor



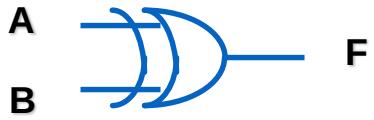
Comparators

Compares Two binary words and indicate if they are equal

Advanced Comparators:

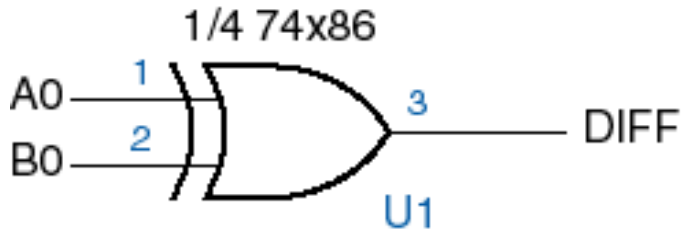


1-bit Comparator: XOR gate, the Output is 1 if $A \neq B$

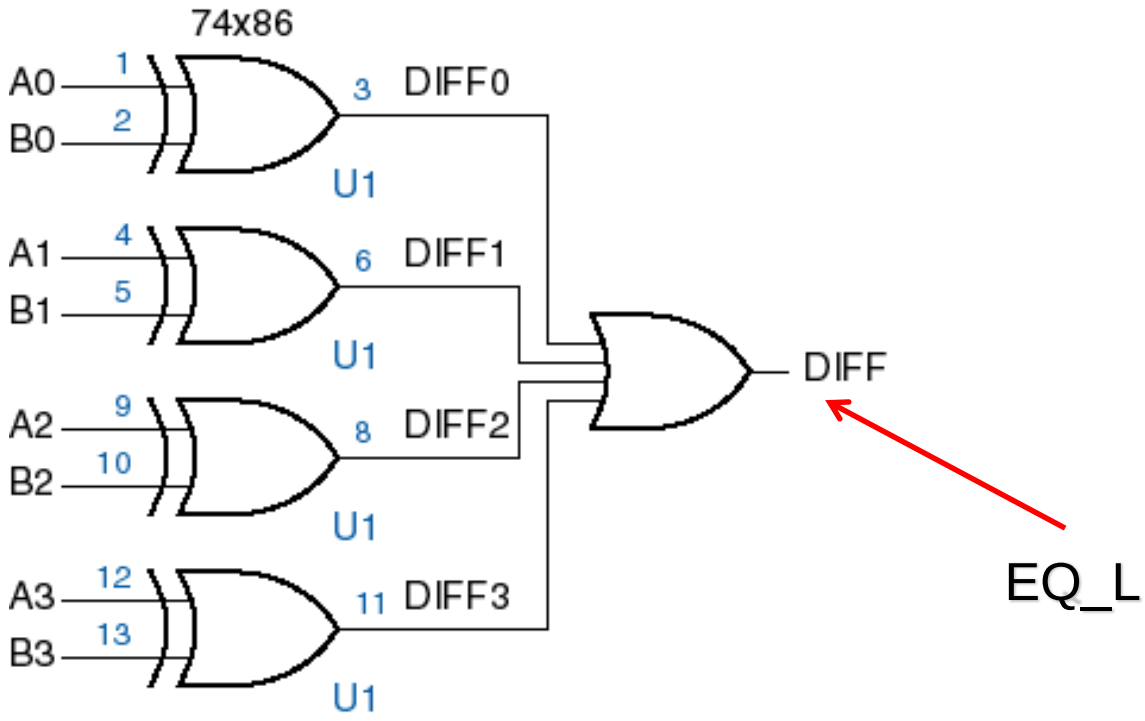


Equality Comparators

1-bit comparator



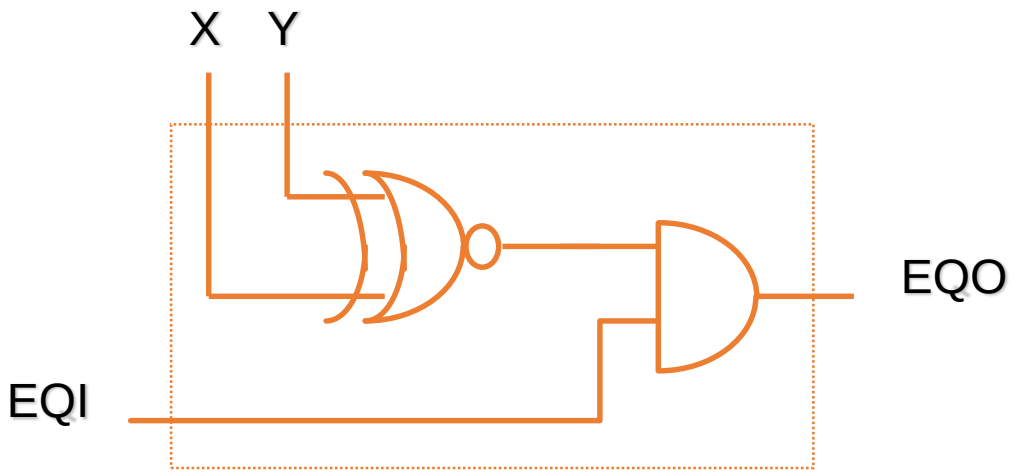
4-bit comparator



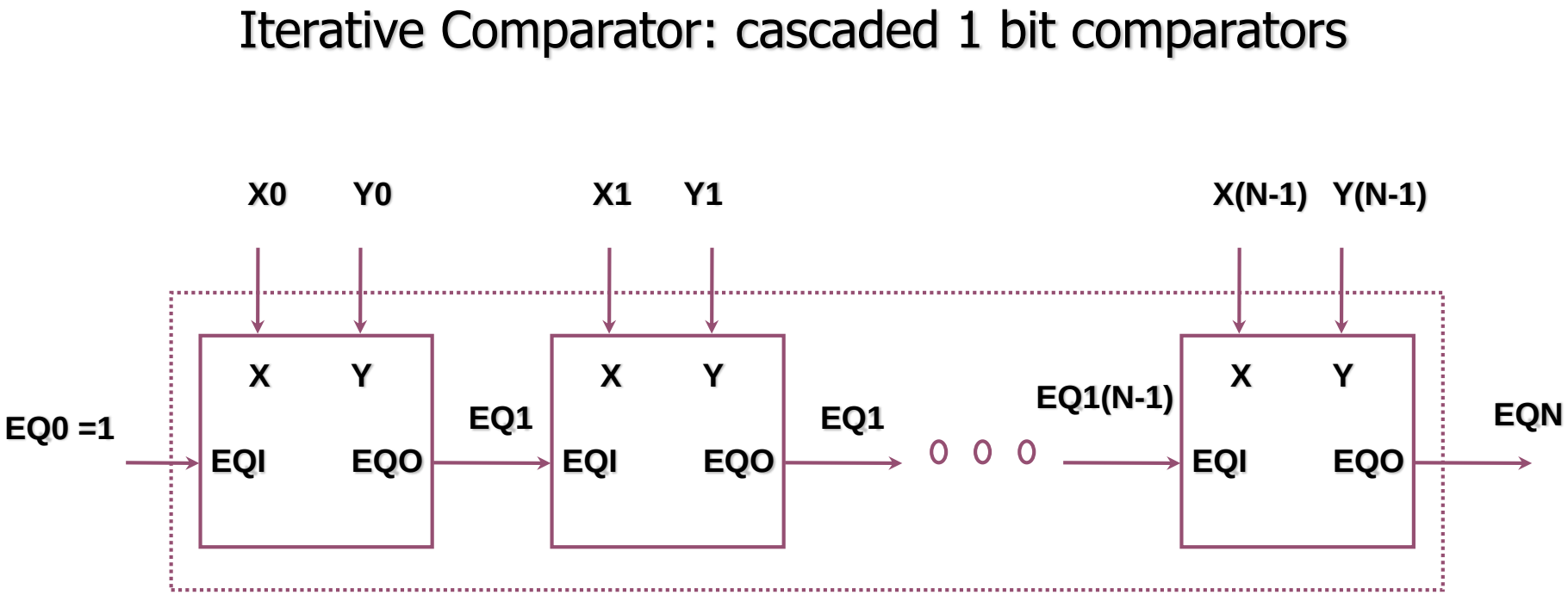
Iterative Comparator

1 bit comparator:

EQI	X	Y	EQO
0	X	X	0
1	0	0	**1**
1	0	1	0
1	1	0	0
1	1	1	**1**



Multibit Iterative Comparator



MSI Arithmetic Logic Units (ALU)

ALU performs **Arithmetic** and **Logical** Functions

A, B: 4 bit inputs

S3, S2, S1, S0: Function select

M = 0: Arithmetic operations: + = Plus, – = Minus

M = 1: Logical operations: + = OR, . = AND

Inputs				**Functions**	
S3	S2	S1	S0	M=0 (arithmetic)	M=1 (logic)
0	0	0	0	A – 1 + CIN	A’
0	1	1	0	A – B – 1 + CIN	A XOR B’
1	0	0	1	A + B + CIN	A XOR B
1	0	1	1	(A OR B) + CIN	A + B
1	1	0	0	A + A + CIN	0000
1	1	1	1	A + CIN	A

Assignments

- Implement 16:1 Mux using 4:1 Muxes
- Implement using only 2:1 Muxes
  - F(A,B,C,D) = (1,2,4,5,7,8,10,11,13,14)
- Implement 5:1 Mux using one 4:1 and one 2:1 mux