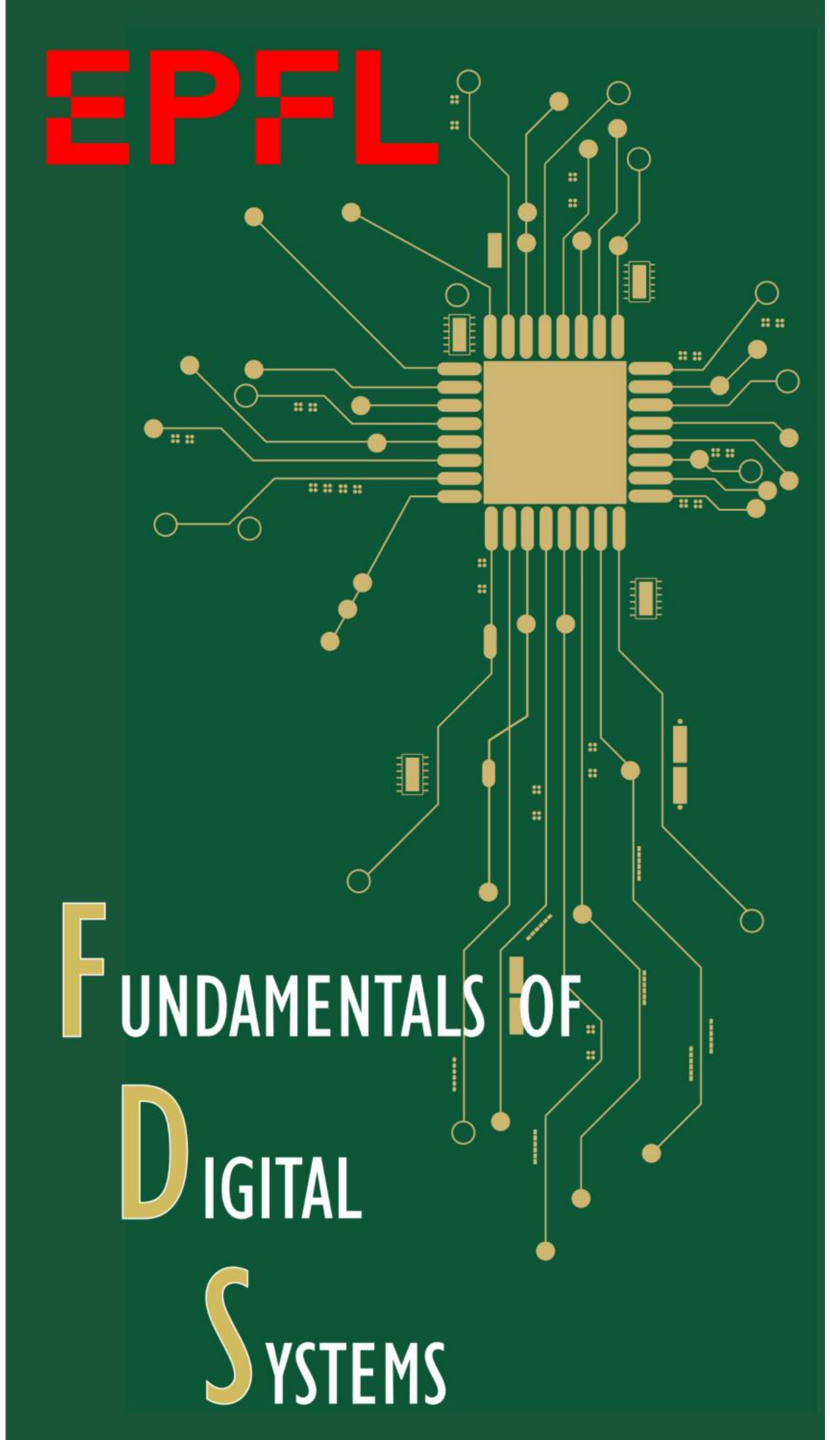


Digital Logic Circuits

Logic Synthesis

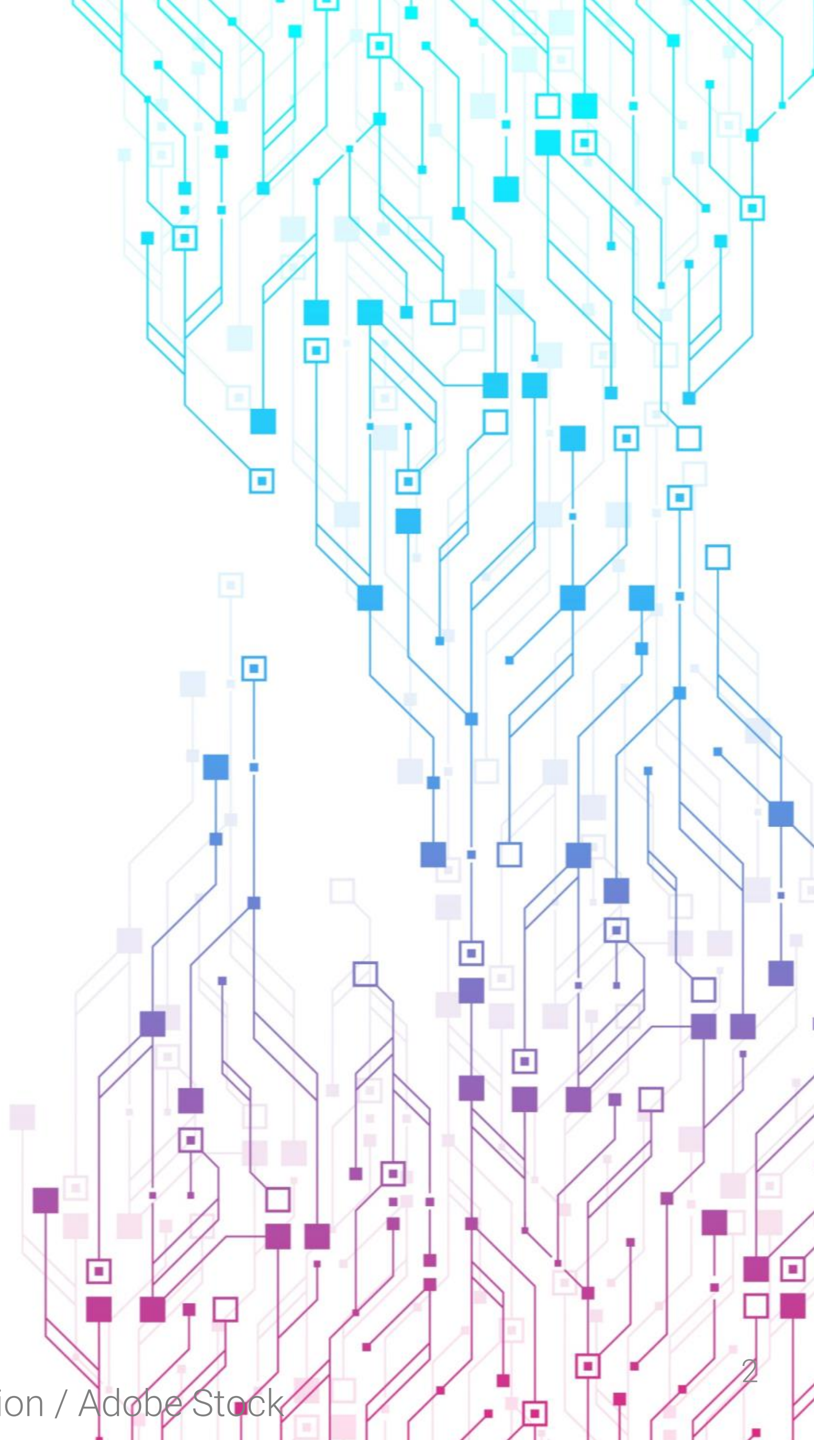
CS-173 Fundamentals of Digital Systems

Mirjana Stojilović

Spring 2025

Previously on FDS

Logic functions, logic gates, and Boolean algebra

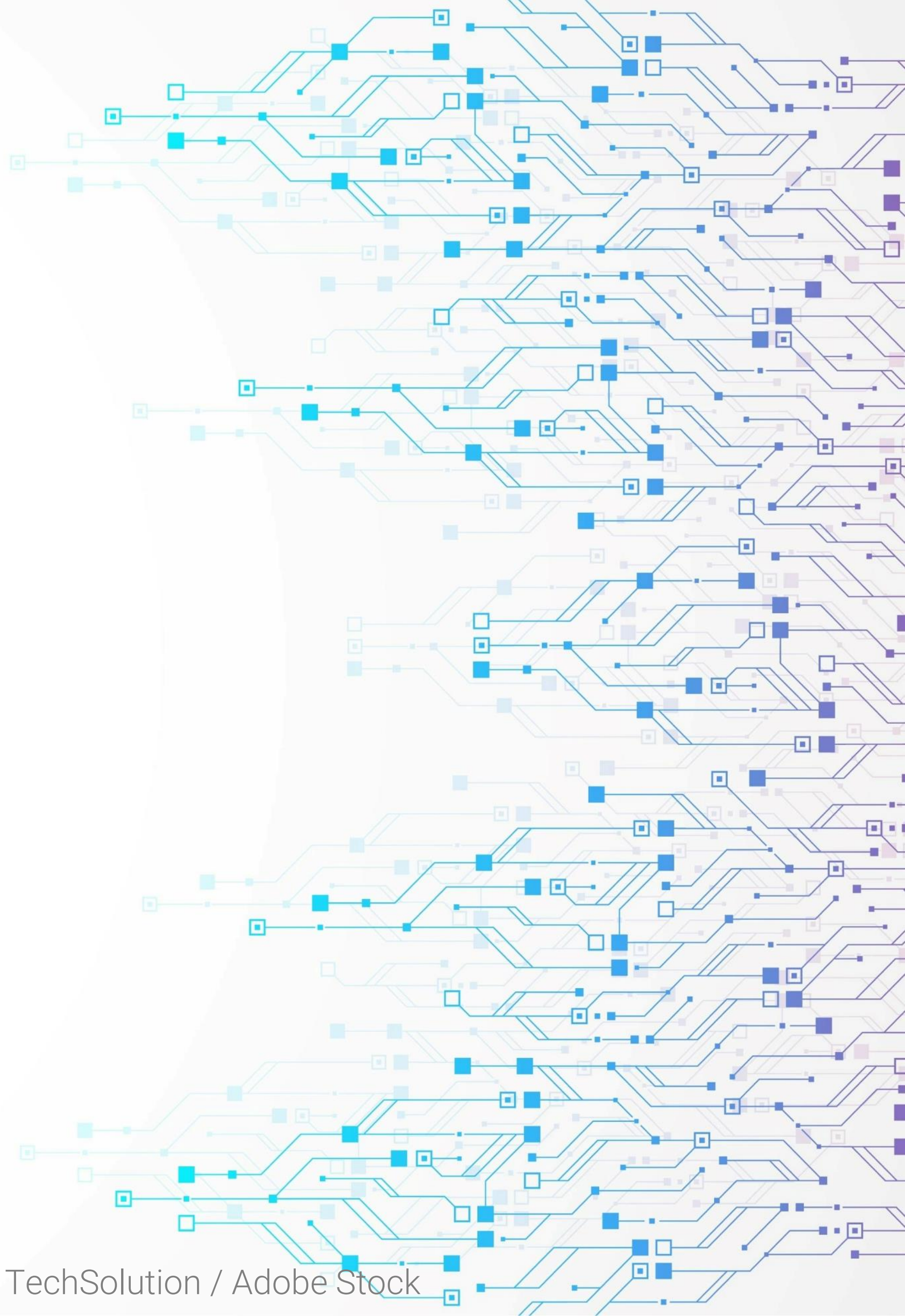


Previously

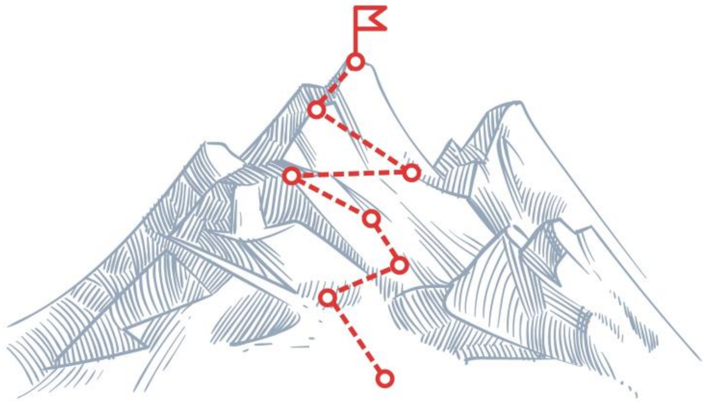
- Discovered basic logic operations (**AND, OR, NOT**) and their graphical representation as **logic gates**
- Built logic networks composed of gates, and learned to write **logic expressions (functions)** to describe the networks' behavior
- Described logic functions using **truth tables**, **timing waveforms**, and **Venn diagrams**
- Used **Boolean algebra** to find equivalent logic circuit implementations

Let’s Talk About…

…Logic synthesis, the process of designing logic circuits from their description



Learning Outcomes



- Apply a well-defined set of techniques (**PoS**, **SoP**) to **synthesize** logic circuits from their truth tables or functional descriptions
- Convert an AND/OR/NOT logic network to a **NAND/NOR** equivalent
- Understand the notion of a **don’t care condition** and use it to build efficient circuits
- Discover and use **XOR** and **XNOR** gates
- Discover and use multiplexers (**MUX**)

Quick Outline

- Logic synthesis
  - Minterms
  - Maxterms
  - Sum-of-products (SoP)
  - Product-of-sums (PoS)
  - Logic synthesis
- NAND and NOR networks
- Incompletely defined functions
  - Don’t care conditions
- Even and odd detectors
  - XOR
  - XNOR
- Design examples
  - Number display
  - MUX

Logic Synthesis

…using AND, OR, and NOT gates

Minterms

- For a function $f = (x_1, x_2, ..., x_n)$ of n variables, a **product term** in which **each** of the n variables appears **once** is called a **minterm**
- Minterms are typically labeled as m_i, where $i \geq 0$ is an integer
- An n-variable minterm m_i can be represented by an n-bit integer
  - Variable appears **complemented** if the corresponding **bit** in the binary representation of m_i is **0**;
  - Otherwise, it appears **uncomplemented (original)**

Minterms

Example

- Examples
  - $n = 3, i = 5$: three variables
    - $5 = (101)_2$ and, therefore $m_5 = x_1 \, \overline{x_2} \, x_3$
  - $n = 5, i = 3$: five variables
    - $3 = (00011)_2$ and, therefore $m_3 = \overline{x_1} \, \overline{x_2} \, \overline{x_3} \, x_4 \, x_5$

EXAMPLES

Minterms

Example

- Find the minterms for the given truth table
- For three inputs (variables), there are eight rows and as many minterms

Row number	x_1	x_2	x_3	Minterm
0	0	0	0	$m_0 = \overline{x_1}\ \overline{x_2}\ \overline{x_3}$
1	0	0	1	$m_1 = \overline{x_1}\ \overline{x_2}\ x_3$
2	0	1	0	$m_2 = \overline{x_1}\ x_2\ \overline{x_3}$
3	0	1	1	$m_3 = \overline{x_1}\ x_2\ x_3$
4	1	0	0	$m_4 = x_1\ \overline{x_2}\ \overline{x_3}$
5	1	0	1	$m_5 = x_1\ \overline{x_2}\ x_3$
6	1	1	0	$m_6 = x_1\ x_2\ \overline{x_3}$
7	1	1	1	$m_7 = x_1\ x_2\ x_3$

Maxterms

- For a function $f = (x_1, x_2, ..., x_n)$ of n variables, a **sum term** in which **each** of the n variables appears **once** is called a **maxterm**
- Maxterms are typically labeled as M_i, where $i \geq 0$ is an integer
- An n-variable maxterm M_i can be represented by an n-bit integer
  - Variable appears **complemented** if the corresponding **bit** in the binary representation of M_i is **1**;
  - Otherwise, it appears **uncomplemented (original)**

Maxterms
Example

EXAMPLES

- Examples:
  - $n = 3, i = 5$
    - $5 = (101)_2$ and, therefore $M_5 = \overline{x_1} + x_2 + \overline{x_3}$
  - $n = 5, i = 3$
    - $3 = (00011)_2$ and, therefore $M_3 = x_1 + x_2 + x_3 + \overline{x_4} + \overline{x_5}$

From Max to Min terms and Vice Versa

- Max/minterms are **complements** of min/maxterms:

$$M_i = \overline{m_i};\ \ m_i = \overline{M_i}$$

- Max/minterms from min/maxterms using

De Morgan's theorem

$$15a.\quad \overline{x \cdot y} = \overline{x} + \overline{y}$$

$$15b.\quad \overline{x + y} = \overline{x} \cdot \overline{y}$$

- Examples:
  - $n = 3, i = 5$
    - $M_5 = \overline{m_5} = \overline{x_1\ \overline{x_2}\ x_3} = \overline{x_1} + x_2 + \overline{x_3}$
    - $m_5 = \overline{M_5} = \overline{\overline{x_1} + x_2 + \overline{x_3}} = x_1\ \overline{x_2}\ x_3$
  - $n = 5, i = 3$
    - $M_3 = \overline{m_3} = \overline{\overline{x_1}\ \overline{x_2}\ \overline{x_3}\ x_4\ x_5} = x_1 + x_2 + x_3 + \overline{x_4} + \overline{x_5}$
    - $m_3 = \overline{M_3} = \overline{x_1 + x_2 + x_3 + \overline{x_4} + \overline{x_5}} = \overline{x_1}\ \overline{x_2}\ \overline{x_3}\ x_4\ x_5$

EXAMPLES

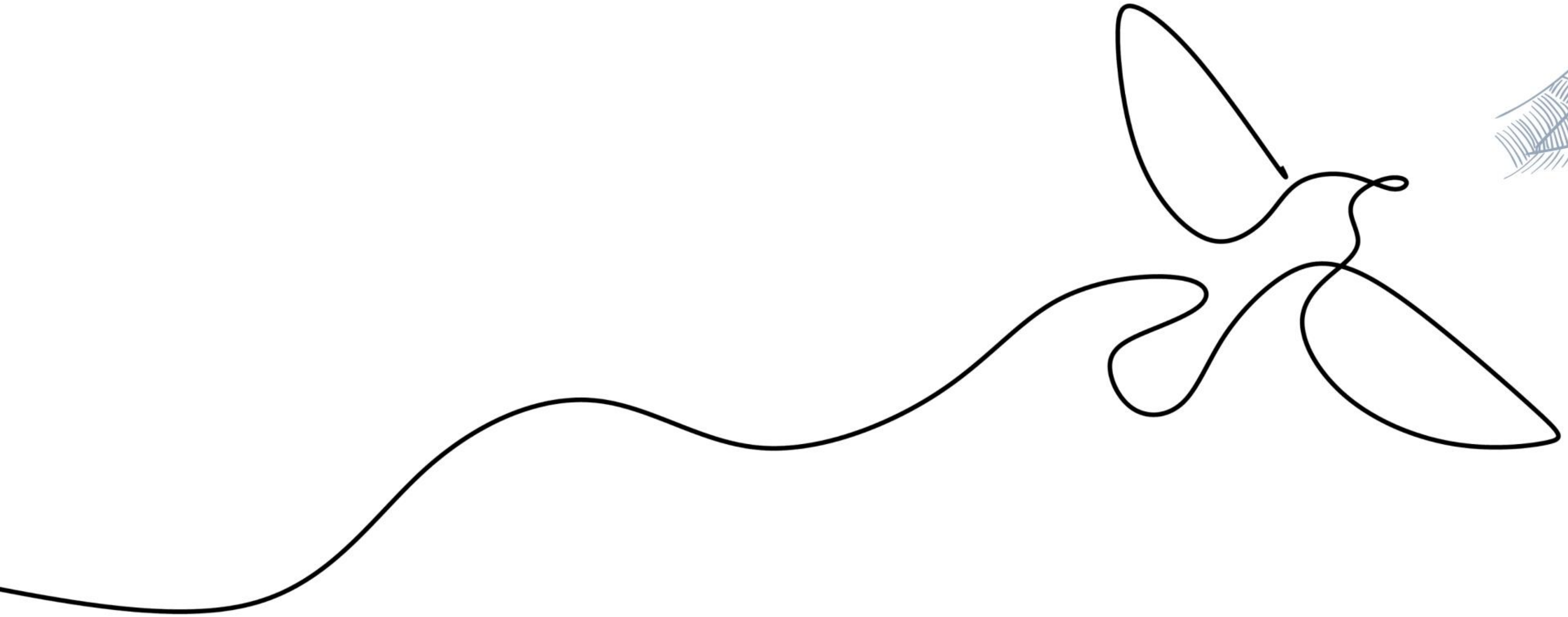
Maxterms

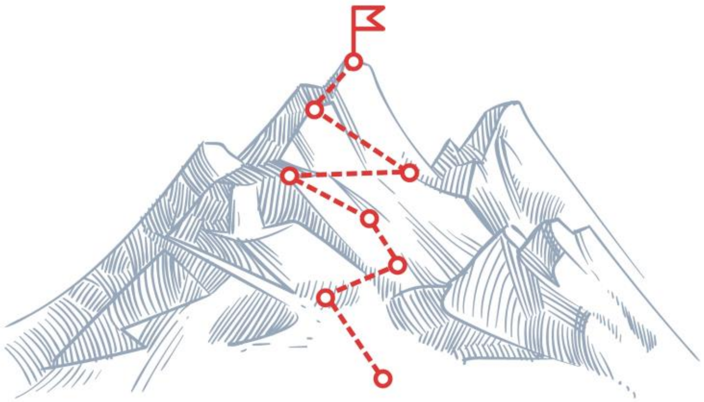
Example

- Find minterms and maxterms for the given truth table: $M_i = \overline{m_i}$

De Morgan's theorem

Row number	x_1	x_2	x_3	Minterm	Maxterm
0	0	0	0	$m_0 = \overline{x_1}\ \overline{x_2}\ \overline{x_3}$	$M_0 = x_1 + x_2 + x_3$
1	0	0	1	$m_1 = \overline{x_1}\ \overline{x_2}\ x_3$	$M_1 = x_1 + x_2 + \overline{x_3}$
2	0	1	0	$m_2 = \overline{x_1}\ x_2\ \overline{x_3}$	$M_2 = x_1 + \overline{x_2} + x_3$
3	0	1	1	$m_3 = \overline{x_1}\ x_2\ x_3$	$M_3 = x_1 + \overline{x_2} + \overline{x_3}$
4	1	0	0	$m_4 = x_1\ \overline{x_2}\ \overline{x_3}$	$M_4 = \overline{x_1} + x_2 + x_3$
5	1	0	1	$m_5 = x_1\ \overline{x_2}\ x_3$	$M_5 = \overline{x_1} + x_2 + \overline{x_3}$
6	1	1	0	$m_6 = x_1\ x_2\ \overline{x_3}$	$M_6 = \overline{x_1} + \overline{x_2} + x_3$
7	1	1	1	$m_7 = x_1\ x_2\ x_3$	$M_7 = \overline{x_1} + \overline{x_2} + \overline{x_3}$





Logic Synthesis with Minterms/Maxterms

- For a function f specified in the form of a truth table, a logic expression realizing the function can be obtained by considering
  - Only the rows in the table for which $f = 1$, or
  - Only the rows in the table for which $f = 0$

- If considering the rows where $f = 1$, f is represented by **the sum of the minterms** corresponding to the rows where $f = 1$
- If considering the rows where $f = 0$, f is described by **the product of the maxterms** corresponding to the rows where $f = 0$

Sum-of-Products (SoP) Form

Logic Synthesis with Minterms

- ***Reminder:*** If considering the rows where $f = 1$, f is represented by the sum of the corresponding minterms
- The resulting logical expression is correct but **not** necessarily the lowest-cost (optimal) implementation of f

- Any logical expression consisting of product (AND) terms that are summed (OR) is said to be in the **sum-of-products (SoP)** form
  - If **each** product term is a **minterm**: **canonical sum-of-products**

Logic Synthesis with SoP Forms

EXAMPLES

- Consider a function f of $n = 3$ variables and the truth table below
- **Canonical** SoP form:

$$f(x_1, x_2, x_3) = \sum (m_1, m_4, m_5, m_6)$$
$$= \sum m(1, 4, 5, 6)$$

x_1	x_2	x_3	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	0

$$f(x_1, x_2, x_3) = \overline{x_1}\ \overline{x_2}\ x_3 + x_1\ \overline{x_2}\ \overline{x_3} + x_1\ \overline{x_2}\ x_3 + x_1\ x_2\ \overline{x_3}$$

$12a.\ \ x \cdot (y + z) = x \cdot y + x \cdot z$

$$= (\overline{x_1} + x_1)\ \overline{x_2}\ x_3 + x_1\ (\overline{x_2} + x_2)\ \overline{x_3}$$

$10b.\ \ x + y = y + x$

$8b.\ x + \overline{x} = 1$

$$= 1 \cdot \overline{x_2}\ x_3 + x_1 \cdot 1 \cdot \overline{x_3}$$

$6a.\ x \cdot 1 = x$

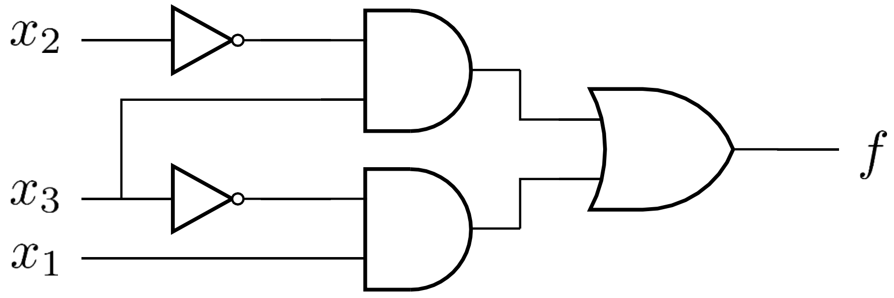
$10a.\ \ x \cdot y = y \cdot x$

$$= \overline{x_2} x_3 + x_1 \overline{x_3}$$

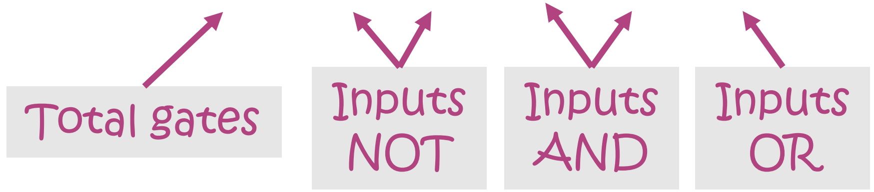
Logic Synthesis with SoP Forms, Contd.

- Logic synthesis from the optimized SoP form

$$f(x_1, x_2, x_3) = \overline{x_2}x_3 + x_1\overline{x_3}$$



- A good indication of the **cost** of a logic circuit is the total number of **gates** and the **inputs** to the gates in the circuit
  - For the design above, cost = 5 + 1 + 1 + 2 + 2 + 2 = 13



Product-of-Sums (PoS) Form

Logic Synthesis with Maxterms

- ***Reminder:*** If considering the rows where $f = 0$, f is represented by the product of the corresponding maxterms
- The resulting logical expression is correct but **not** necessarily the lowest-cost (optimal) implementation of f

- Any logical expression consisting of sum (OR) terms that are the factors of a product (AND) in the **product-of-sums (PoS)** form
  - If **each** product term is a **maxterm**: **canonical product-of-sums**

Logic Synthesis with PoS forms

- Consider a function f of $n = 3$ variables and the truth table below

$$f(x_1, x_2, x_3) = \prod(M_0, M_2, M_3, M_7)$$
$$= \prod M(0, 2, 3, 7)$$

$$f(x_1, x_2, x_3) = M_0 \cdot M_2 \cdot M_3 \cdot M_7$$
$$= (x_1 + x_2 + x_3)(x_1 + \overline{x_2} + x_3)(x_1 + \overline{x_2} + \overline{x_3})(\overline{x_1} + \overline{x_2} + \overline{x_3})$$

x_1	x_2	x_3	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	0

- Note the expression for the **complement** of f

$$\bar{f}(x_1, x_2, x_3) = \overline{M_0 \cdot M_2 \cdot M_3 \cdot M_7} = \overline{M_0} + \overline{M_2} + \overline{M_3} + \overline{M_7}$$

De Morgan's theorem

$$= m_0 + m_2 + m_3 + m_7$$

$$f = \overline{\overline{f}} = \overline{m_0 + m_2 + m_3 + m_7}$$

EXAMPLES

Logic Synthesis with PoS forms, Contd.

- Logic synthesis from the optimized PoS form

$$f(x_1, x_2, x_3) = M_0 \cdot M_2 \cdot M_3 \cdot M_7$$

$$= (x_1 + x_2 + x_3)(x_1 + \overline{x_2} + x_3)(x_1 + \overline{x_2} + \overline{x_3})(\overline{x_1} + \overline{x_2} + \overline{x_3})$$

$10b.\ x + y = y + x$ | $11b.\ x + (y + z) = (x + y) + z$

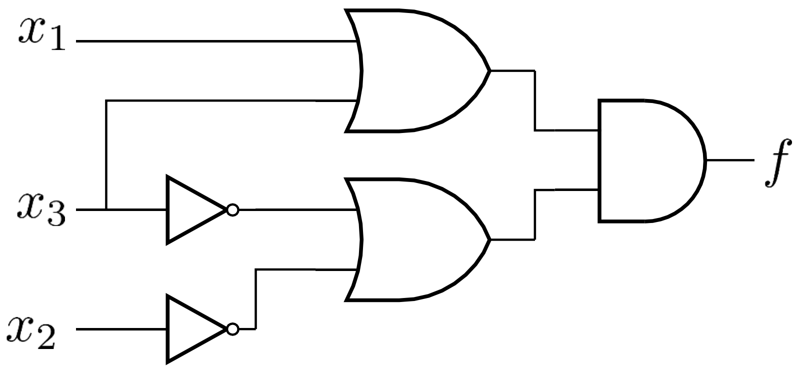
$$= ((x_1 + x_3) + x_2)((x_1 + x_3) + \overline{x_2})(x_1 + (\overline{x_2} + \overline{x_3}))(\overline{x_1} + (\overline{x_2} + \overline{x_3}))$$

$14b.\ (x + y)(x + \overline{y}) = x$ Combining | $10b.\ x + y = y + x$

$$= (x_1 + x_3)(\overline{x_2} + \overline{x_3})$$

- Cost = 5 + 1 + 1 + 2 + 2 + 2 = 13

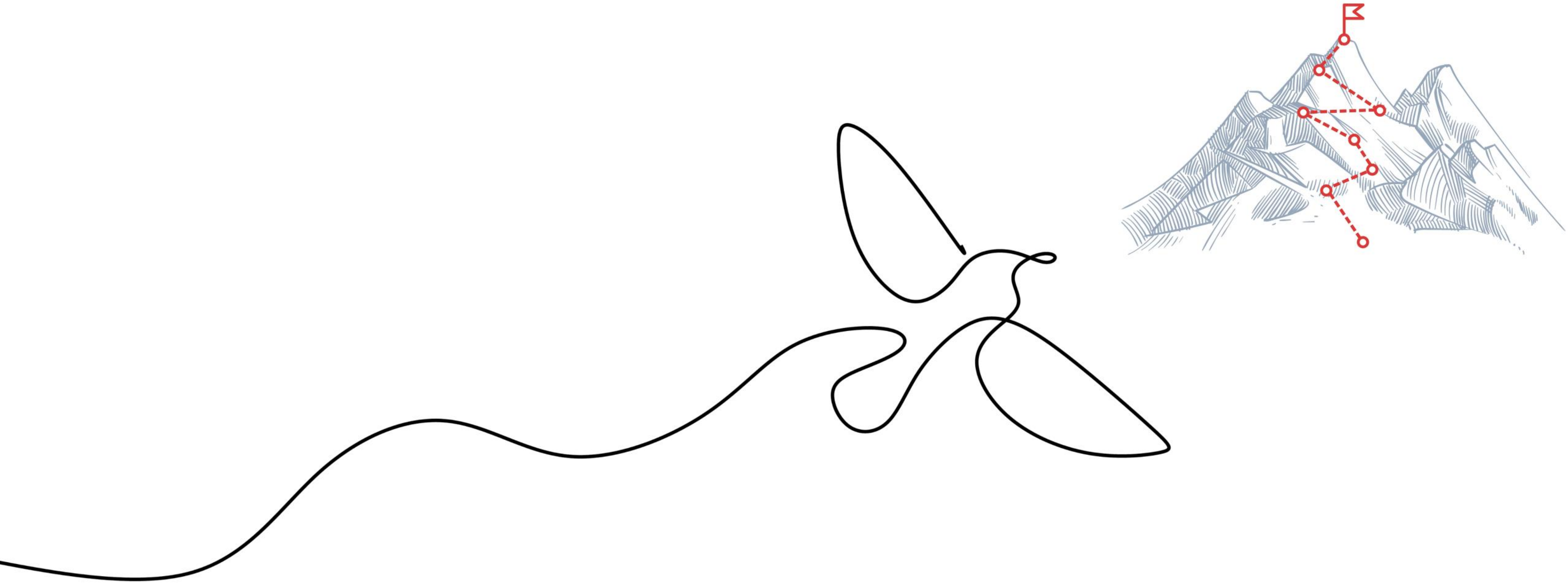
Total gates (5); Inputs NOT (1 + 1); Inputs OR (2 + 2); Inputs AND (2)



Which One is "Better?" PoS or SoP?

- In general, will PoS and SoP forms give us equally efficient (in terms of cost) logic circuit implementation?
  Should we **prefer** one form over another?

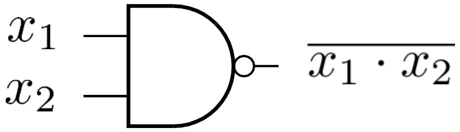
- **A:** Generally, the costs of networks derived from the SoP and PoS forms **do not have to be equal**.
  - One should derive both and select the one that has a lower cost
  - One form may require fewer gates or fewer levels of logic (lower delay)



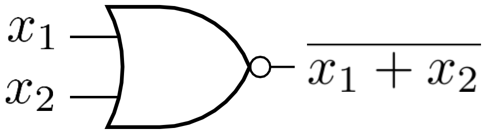
NAND and NOR Logic Networks

Logic Synthesis with NAND and NOR

- NAND and NOR gates can be used to build logic circuits



$f(x_1, x_2) = \overline{x_1 \cdot x_2}$



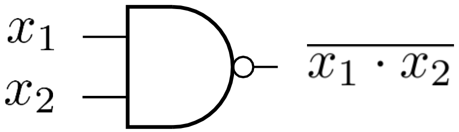
$f(x_1, x_2) = \overline{x_1 + x_2}$

  - NAND/NOR physical implementation is simpler (requires fewer transistors) and more efficient than AND/OR
  - AND/OR are implemented as NAND/NOR + NOT

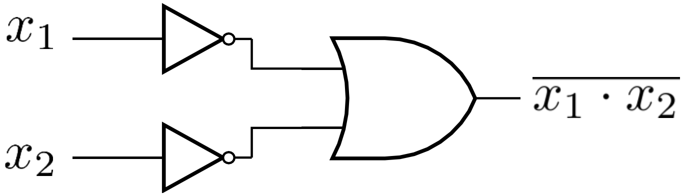
- How to build logic circuits with NAND and NOR gates?

De Morgan’s Theorem

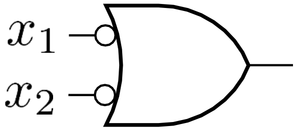
Applied to NAND and NOR



$\equiv$



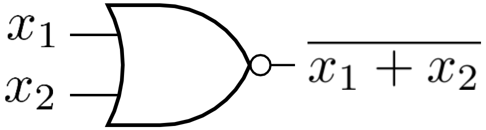
$\equiv$



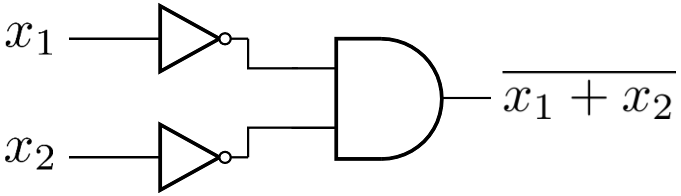
$$f(x_1, x_2) = \overline{x_1 \cdot x_2}$$

$$f(x_1, x_2) = \overline{x_1 \cdot x_2} = \overline{x_1} + \overline{x_2}$$

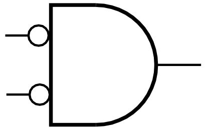
NAND = OR with both inputs inverted



$\equiv$



$\equiv$



$$f(x_1, x_2) = \overline{x_1 + x_2}$$

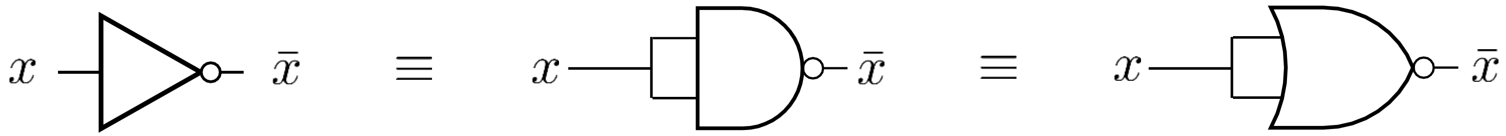
$$f(x_1, x_2) = \overline{x_1 + x_2} = \overline{x_1}\ \overline{x_2}$$

NOR = AND with both inputs inverted

NOT Gate Using NAND or NOR

- According to Boolean theorems:
  - $\overline{x} = \overline{x \cdot x}$ (NAND) and
  - $\overline{x} = \overline{x + x}$ (NOR)

$$7a. \quad x \cdot x = x$$
$$7b. \quad x + x = x$$



NOT = NAND with both inputs equal

NOT = NOR with both inputs equal

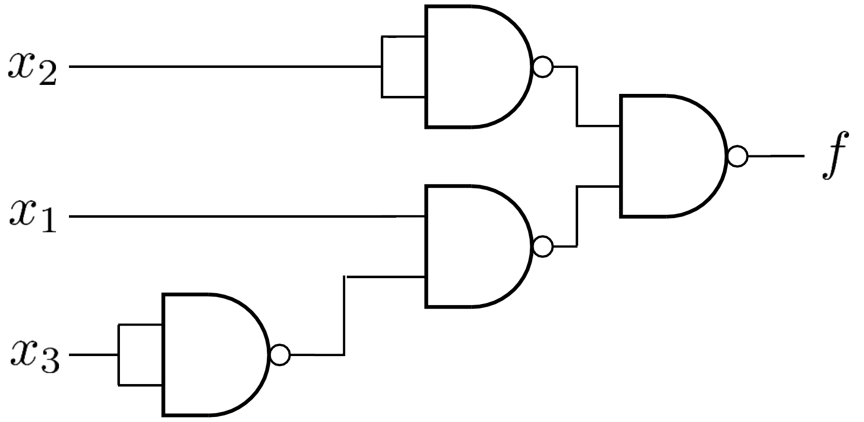
Logic Network with NAND Gates

- Implement the following function in the **SoP form with NAND**

$$f = x_2 + x_1\overline{x_3}$$

- **Algorithm:** start by applying double inversion and, then, De Morgan's theorem to simplify the expression

$$
\begin{aligned}
f &= x_2 + x_1\overline{x_3} \\
&= \overline{\overline{x_2 + x_1\overline{x_3}}} \\
&= \overline{\overline{x_2} \cdot \overline{x_1\overline{x_3}}} \quad \text{(De Morgan's)}
\end{aligned}
$$



Logic Network with NOR Gates

- Implement the following function in the **PoS form with NOR**

$$f = (x_1 + x_2)(x_2 + \overline{x_3})$$

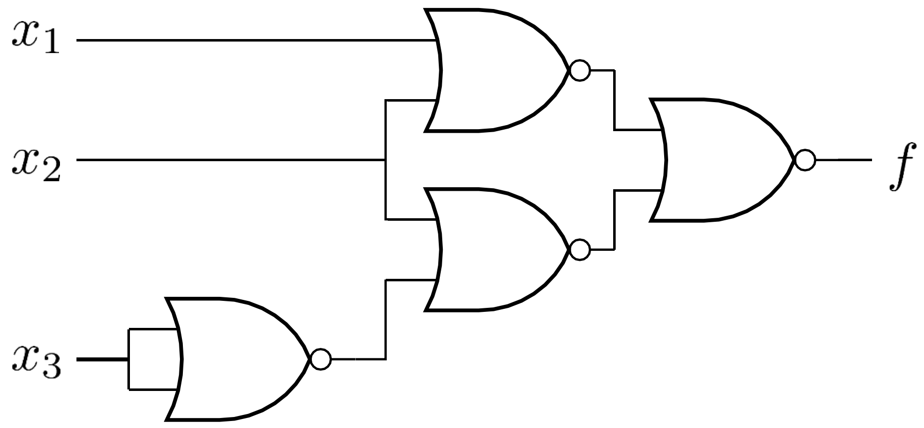
- **Algorithm:** start by applying double inversion and, then, De Morgan's theorem to simplify the expression

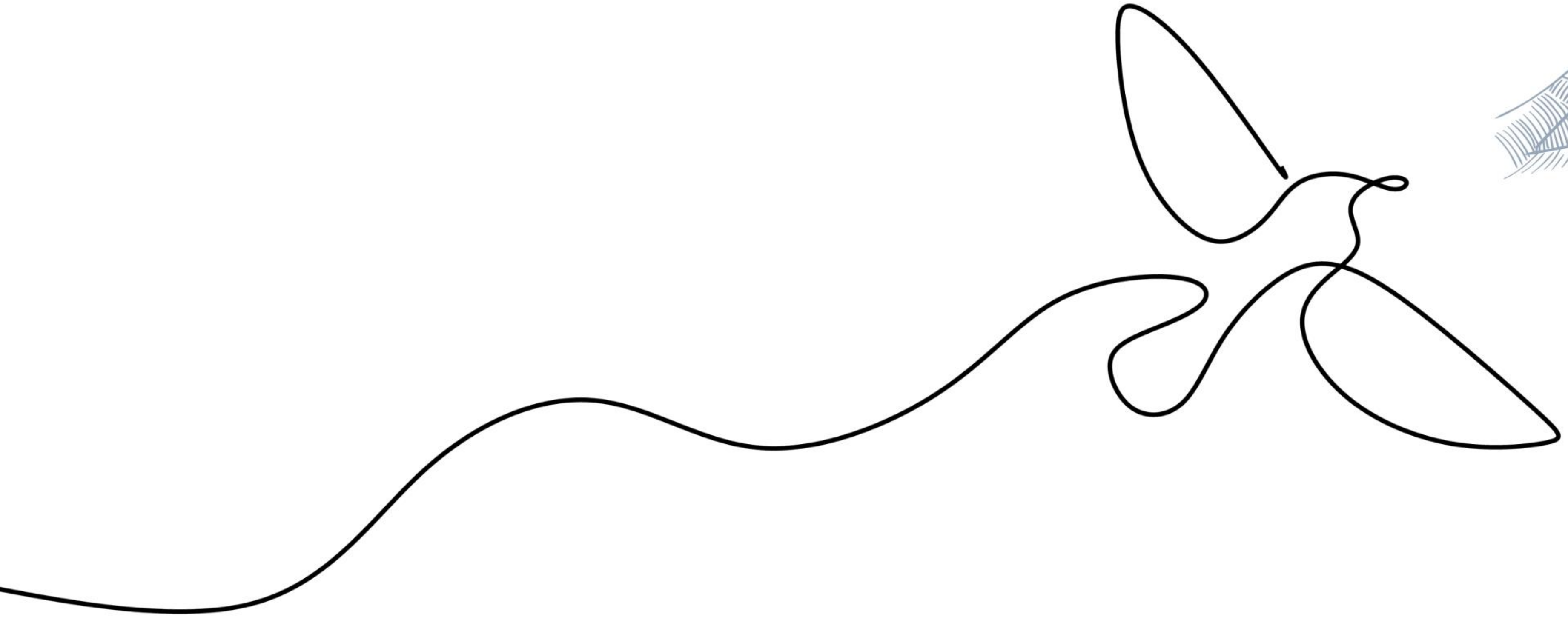
$$f = (x_1 + x_2)(x_2 + \overline{x_3})$$

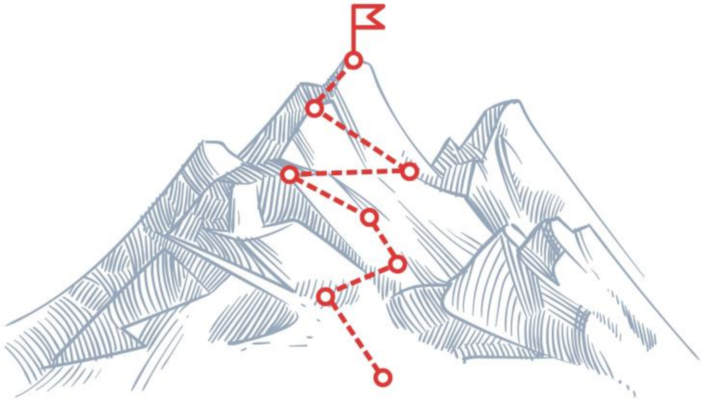
$$= \overline{\overline{(x_1 + x_2)(x_2 + \overline{x_3})}}$$

De Morgan's

$$= \overline{\overline{(x_1 + x_2)} + \overline{(x_2 + \overline{x_3})}}$$







Incompletely Defined Functions

… And Don’t Cares

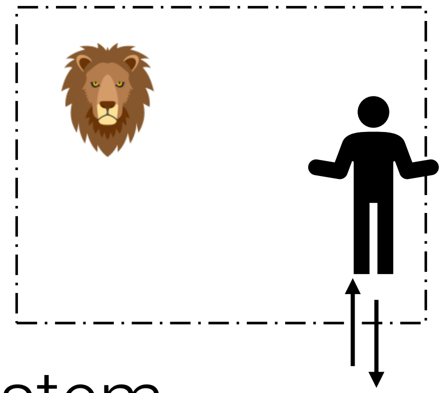
Incompletely Defined Functions

- …are Boolean functions where some input combinations are not specified because they don't matter (e.g., they never occur), so the function does not need to define outputs for them
  - Those input combinations are called **don't care conditions**

- In logic optimization, don't care conditions can be assigned function value (output) either 0 or 1, to simplify the logic circuit

Don’t Care Conditions

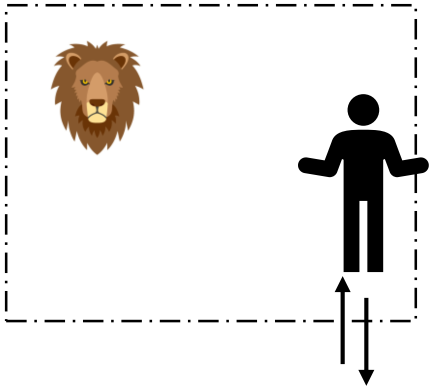
Example: Lion’s Cage Door Control



- Imagine a lion’s cage with an automated door control system including two sensors and a manual override switch
- Inputs
  - **Sensor L**: Detects if the lion is inside (1 = inside; 0 = outside)
  - **Sensor T**: Detects if the trainer is inside (1 = inside; 0 = outside)
  - **Override switch (S)**: The trainer can manually force the door open or closed irrespective of presence (1 = override enabled; 0 = normal mode)
- Outputs
  - **Door control (D)**: 1 = open (unlocked); 0 = closed (locked)

Don’t Care Conditions

Example: Lion’s Cage Door Control



- Truth table

Lion inside (L)	Trainer inside (T)	Override switch (S)	Door (D)
0	0	0	0 (door closed, nobody inside)
0	1	0	1 (door open, trainer inside)
1	0	0	0 (door closed, lion inside)
1	1	0	1 (door open, trainer inside)
X	X	1	1 or 0 (Don’t care) Override forces door open or closed

Door open when trainer inside (rows 1–4)

- When S is 1, inputs L and T do not matter (don’t care, **X**)

EXAMPLES

Don't Care Conditions

Example: Lion's Cage Door Control

- **Inefficient** logic implementation (unoptimized)

Lion inside (L)	Trainer inside (T)	Override switch (S)	Door (D)
0	0	0	0 (door closed, nobody inside)
0	1	0	**1** (door open, trainer inside)
1	0	0	0 (door closed, lion inside)
1	1	0	**1** (door open, trainer inside)
0	*0*	*1*	***1*** *(for example)*
0	*1*	*1*	*0 (for example)*
1	*0*	*1*	*0 (for example)*
1	*1*	*1*	*0 (for example)*

$$6a.\quad x \cdot 1 = x$$
$$6b.\quad x + 0 = x$$
$$8a.\quad x \cdot \bar{x} = 0$$
$$8b.\quad x + \bar{x} = 1$$

$$D = \overline{L}\ T\ \overline{S} + L\ T\ \overline{S} + \overline{L}\ \overline{T}\ S$$
$$= T\ \overline{S} + \overline{L}\ \overline{T}\ S$$

Cost = 6 gates + 10 inputs = 16

3×NOT, 2-input AND, 3-input AND, 2-input OR

Don’t Care Conditions

Example: Lion’s Cage Door Control

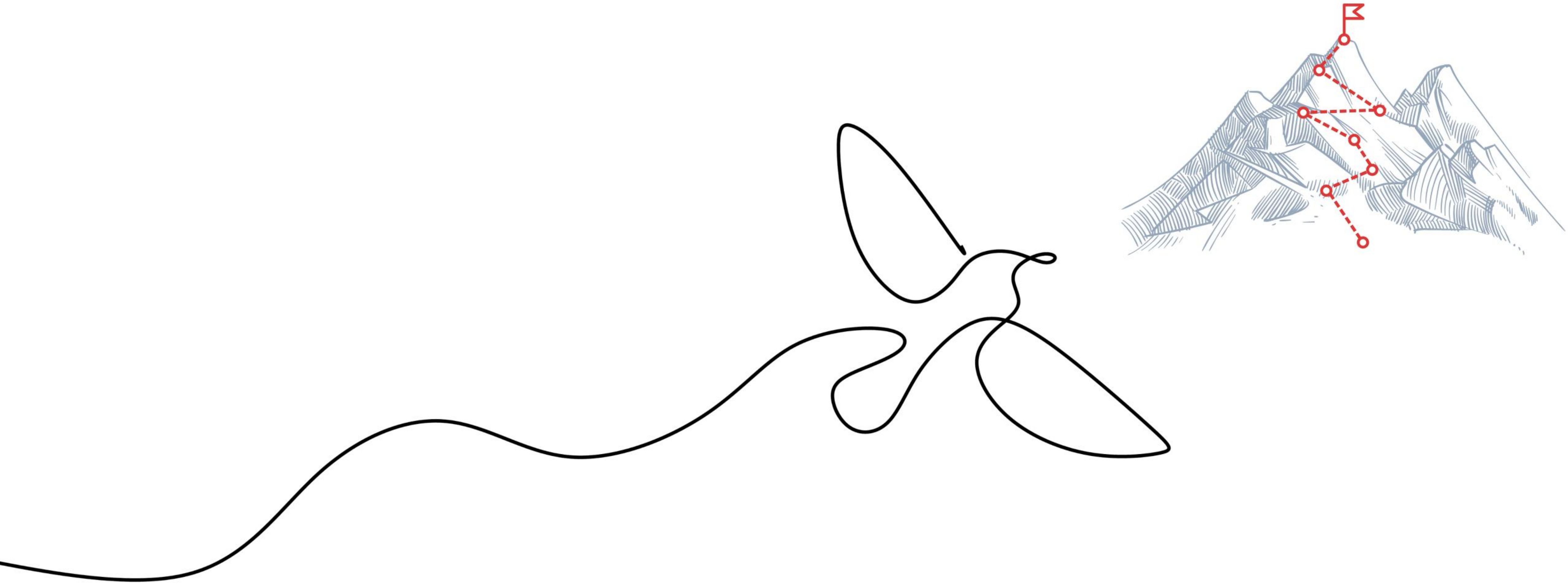
- **More efficient** logic implementation

Lion inside (L)	Trainer inside (T)	Override switch (S)	Door (D)
0	0	0	0 (door closed, nobody inside)
0	1	0	**1** (door open, trainer inside)
1	0	0	0 (door closed, lion inside)
1	1	0	**1** (door open, trainer inside)
0	*0*	*1*	***1*** *(for example)*
0	*1*	*1*	***1*** *(for example)*
1	*0*	*1*	***1*** *(for example)*
1	*1*	*1*	***1*** *(for example)*

Don’t cares help optimize the circuit

$$D = T + S$$

Cost = 1 gate + 2 inputs = 3



Even and Odd Detectors

XOR and XNOR gates

Exclusive OR Operation (XOR)

- Consider the truth table below

x_1	x_2	f
0	0	0
0	1	**1**
1	0	**1**
1	1	0

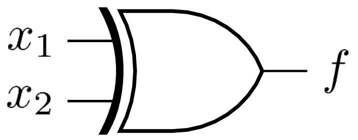
- The output is set when the inputs are of the opposite polarity (odd detector)

- Logic function derived from the truth table:

$$f = \overline{x_1}\, x_2 + x_1\, \overline{x_2}$$

- We call it **Exclusive OR** (also **an odd function**) and write $\oplus$

$$f = \overline{x_1}\, x_2 + x_1\, \overline{x_2} = x_1 \oplus x_2$$



$$f = x_1 \oplus x_2$$

Coincidence Operation (XNOR)

- Another common operation is the complement of XOR

x_1	x_2	f
0	0	**1**
0	1	0
1	0	0
1	1	**1**

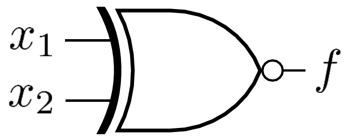
- The output is set when the inputs are of the same polarity (even detector)

- Logic function derived from the truth table:

$$f = \overline{x_1}\,\overline{x_2} + x_1 x_2$$

- We call it **XNOR** or **coincidence operation** and write $\odot$

$$f = \overline{x_1}\,\overline{x_2} + x_1 x_2 = x_1 \odot x_2$$



$$f = x_1 \odot x_2$$

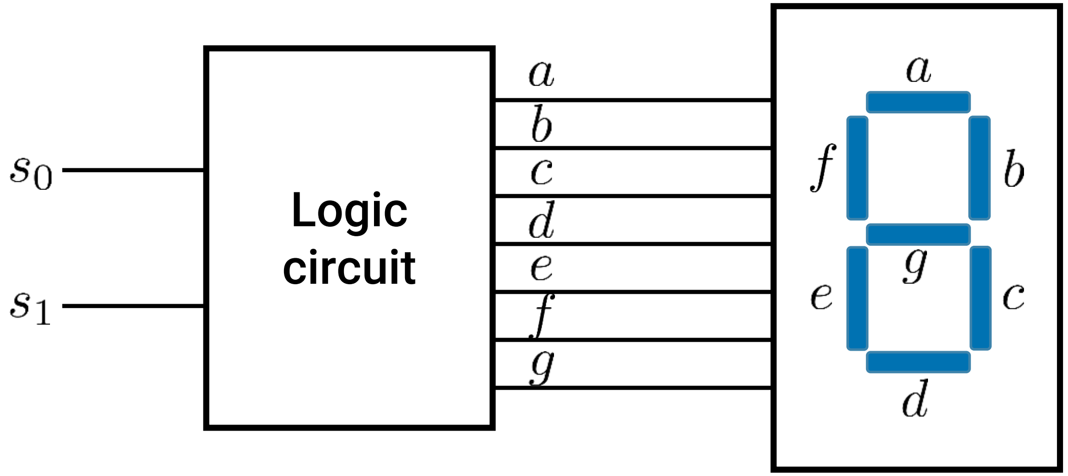
Design Examples

Number display

Number Display

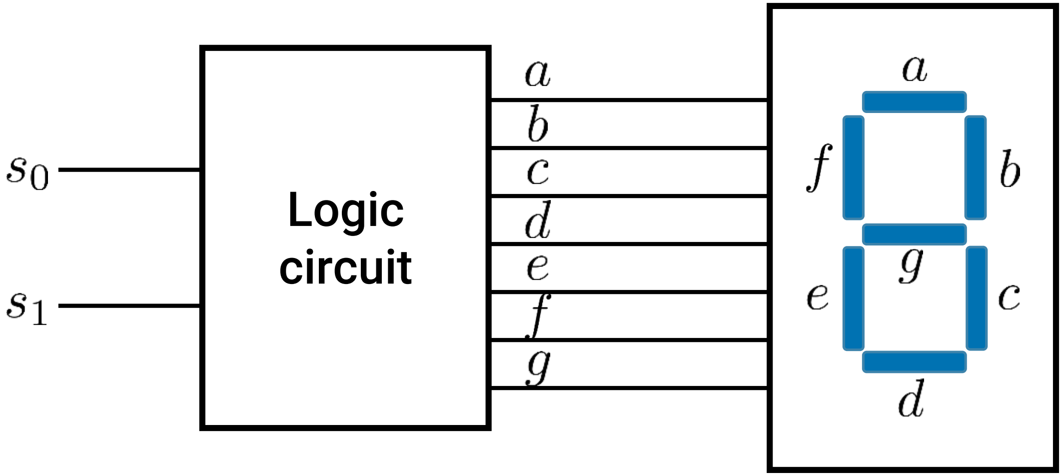
Multiple-Output Circuit

- Design a logic circuit to drive a seven-segment display
- The display shows value $(s_1, s_0)_2$ as a decimal number

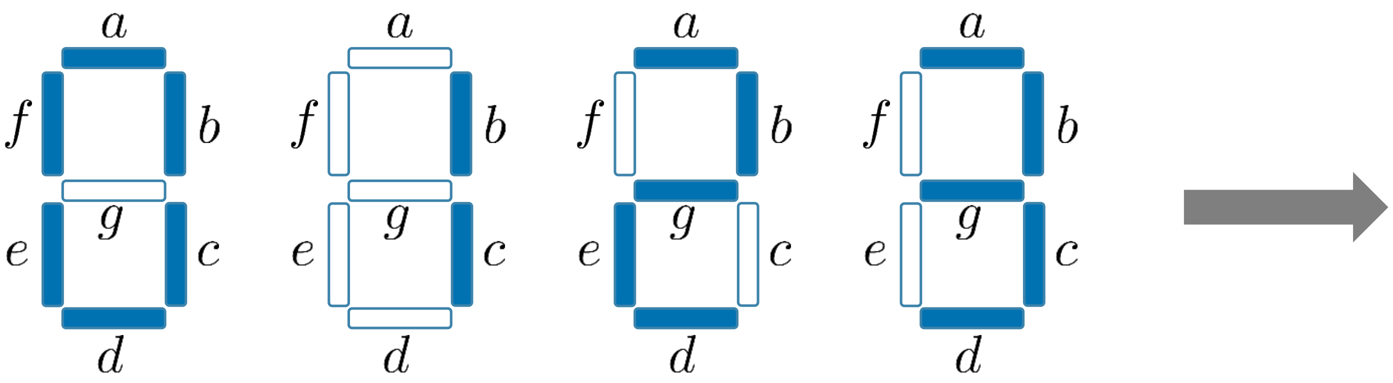


Number Display

Multiple-Output Circuit



- The display shows value $(s_1, s_0)_2$ as a decimal number



Corresponding truth table

s_1	s_0	a	b	c	d	e	f	g
0	0	1	1	1	1	1	1	0
0	1	0	1	1	0	0	0	0
1	0	1	1	0	1	1	0	1
1	1	1	1	1	1	0	0	1

Number Display, Contd.

Multiple-Output Circuit

- From the truth table below, derive **one logic function per output**
  - One can use minterms or maxterms, whichever appears more efficient

s_1	s_0	a	b	c	d	e	f	g
0	0	1	1	1	1	1	1	0
0	1	0	1	1	0	0	0	0
1	0	1	1	0	1	1	0	1
1	1	1	1	1	1	0	0	1

$$a(s_0, s_1) = M_1 = s_1 + \overline{s_0}$$

$$b(s_0, s_1) = 1$$

$$c(s_0, s_1) = M_2 = \overline{s_1} + s_0$$

$$d(s_0, s_1) = M_1 = s_1 + \overline{s_0} = a(s_0, s_1)$$

$$e(s_0, s_1) = M_1 \cdot M_3 = m_0 + m_2 = \overline{s_1}\ \overline{s_0} + s_1\overline{s_0} = \overline{s_0}$$

$$f(s_0, s_1) = m_0 = \overline{s_1}\ \overline{s_0}$$

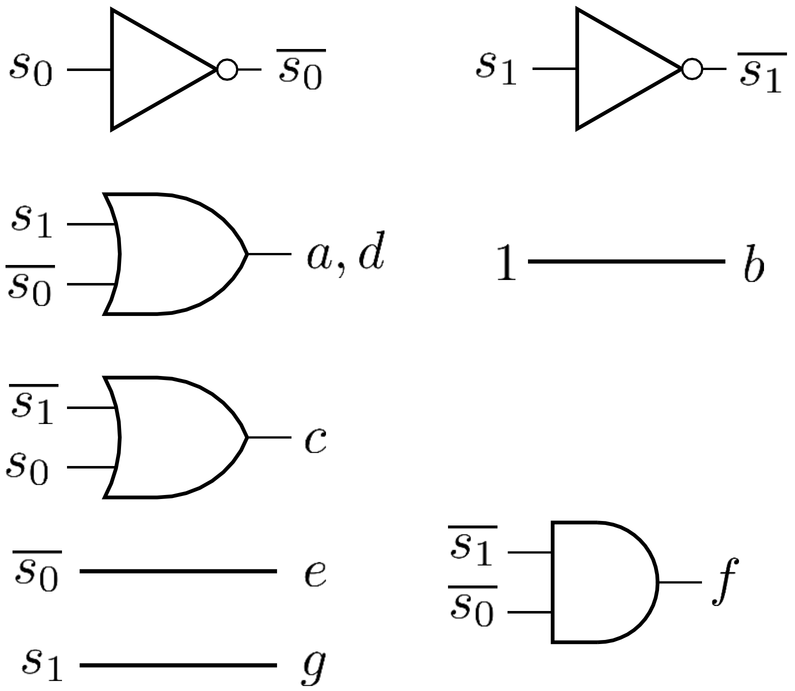
$$g(s_0, s_1) = M_0 \cdot M_1 = m_2 + m_3 = s_1\ \overline{s_0} + s_1 s_0 = s_1$$

EXAMPLES

Number Display, Contd.

Multiple-Output Circuit

- Draw the corresponding logic network



$$a(s_0, s_1) = M_1 = s_1 + \overline{s_0}$$

$$b(s_0, s_1) = 1$$

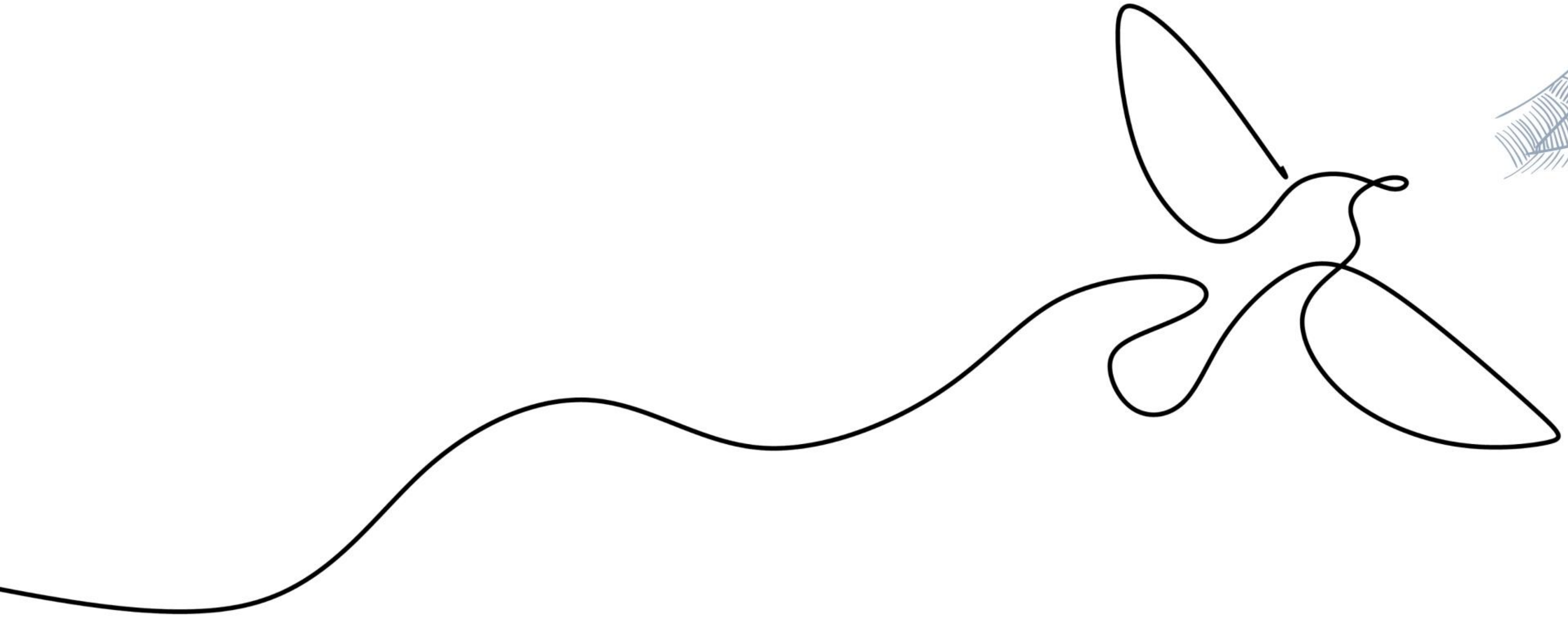
$$c(s_0, s_1) = M_2 = \overline{s_1} + s_0$$

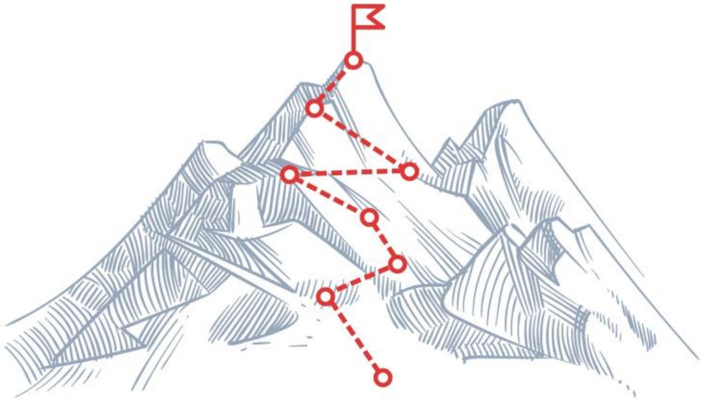
$$d(s_0, s_1) = M_1 = s_1 + \overline{s_0} = a(s_0, s_1)$$

$$e(s_0, s_1) = M_1 \cdot M_3 = m_0 + m_2 = \overline{s_1}\ \overline{s_0} + s_1\overline{s_0} = \overline{s_0}$$

$$f(s_0, s_1) = m_0 = \overline{s_1}\ \overline{s_0}$$

$$g(s_0, s_1) = M_0 \cdot M_1 = m_2 + m_3 = s_1\ \overline{s_0} + s_1 s_0 = s_1$$





Design Examples

Multiplexer

Data Selector (Multiplexer or MUX)

- It is often helpful to choose **precisely one** from several inputs
- A circuit performing data selection (a **multiplexer**) has one or more **select** inputs dedicated to determining which of the remaining inputs to pass to the output
- For example, a three-input multiplexer (also called **2-to-1 MUX**):
  - Inputs
    - One **selection** signal s
    - Two data **inputs** x_1 and x_2
  - When the selection signal is $s = 0$, the **output** becomes $f = x_1$
  - Otherwise, the output becomes $f = x_2$

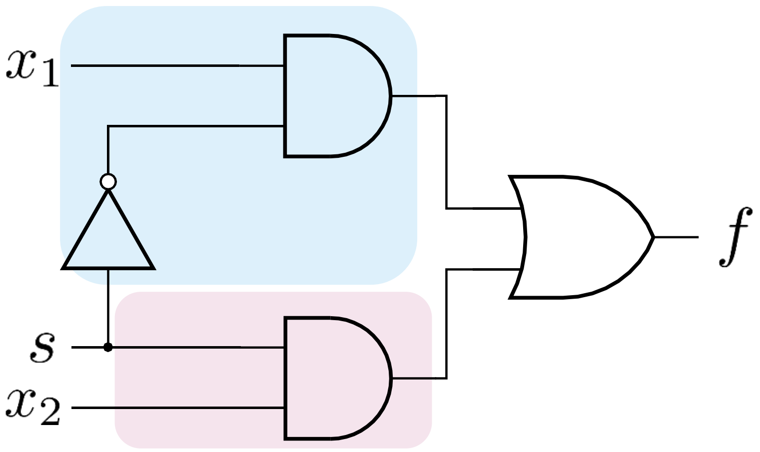
2-to-1 Multiplexer (MUX)

Logic Circuit and the Graphical Symbol

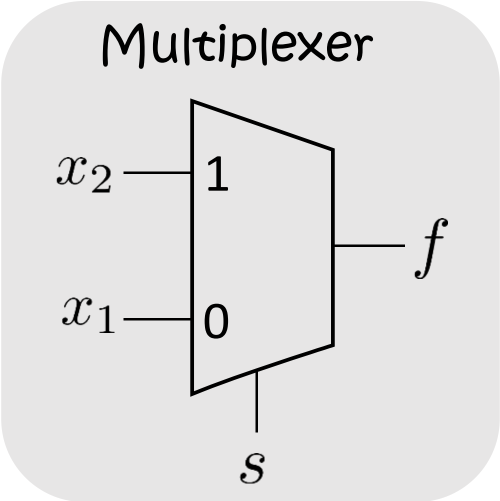
	s	x_1	x_2	f
$s=0$	0	0	0	0
	0	0	1	0
	0	1	0	1
	0	1	1	1
$s=1$	1	0	0	0
	1	0	1	1
	1	1	0	0
	1	1	1	1

$$f(s, x_1, x_2) = \overline{s}\ x_1\overline{x_2} + \overline{s}\ x_1x_2 + s\overline{x_1}\ x_2 + sx_1x_2$$

$$\begin{aligned} f(s, x_1, x_2) &= \overline{s}x_1(\overline{x_2} + x_2) + s(\overline{x_1} + x_1)x_2 \\ &= \overline{s}x_1 \cdot 1 + s \cdot 1 \cdot x_2 \\ &= \overline{s}x_1 + sx_2 \end{aligned}$$



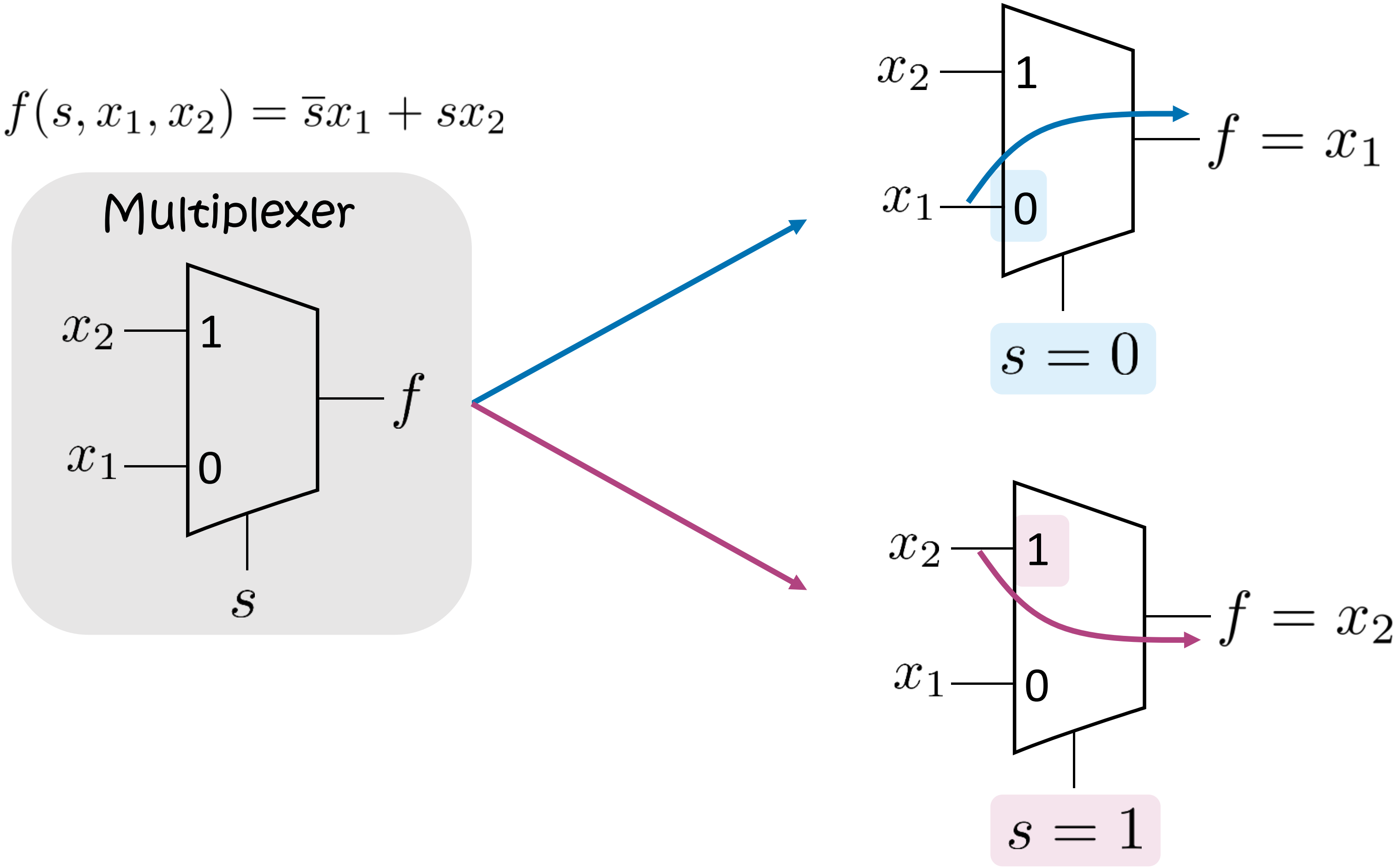
$\equiv$



2-to-1 Multiplexer (MUX)

Logic Circuit and the Graphical Symbol

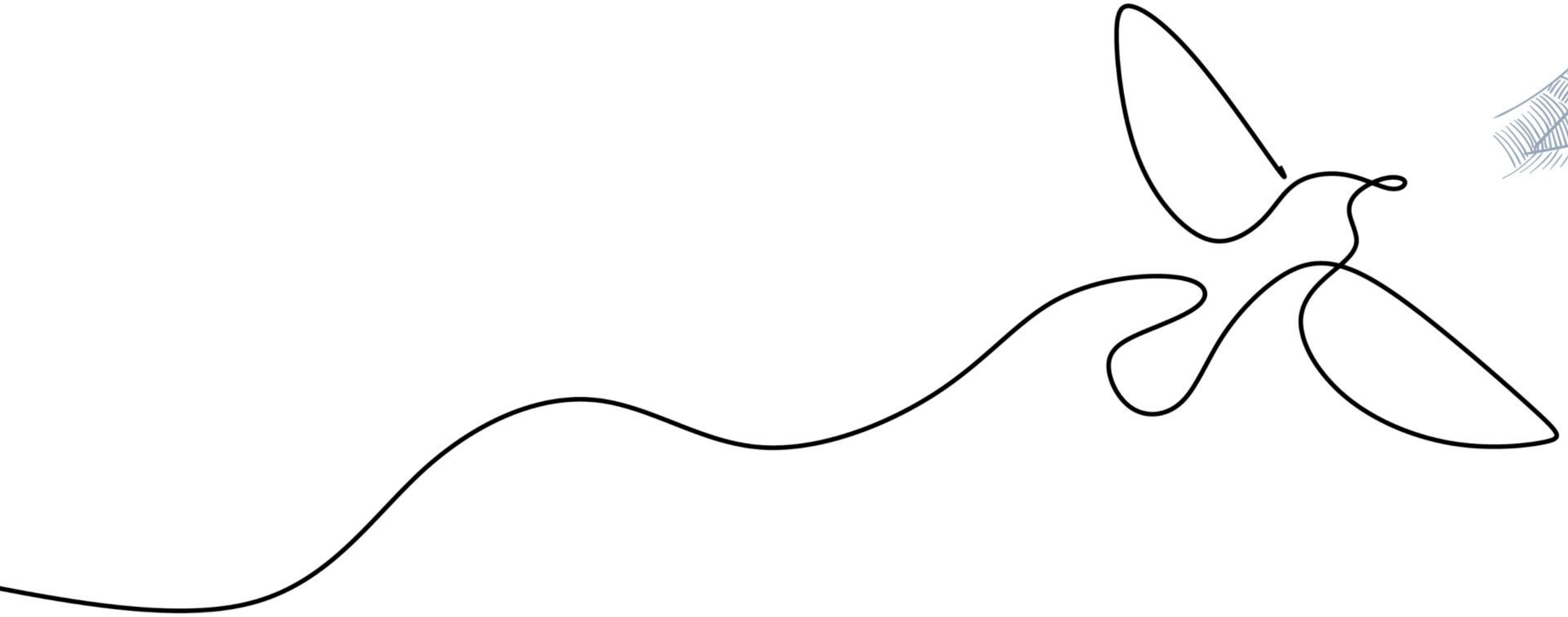
$$f(s, x_1, x_2) = \overline{s}x_1 + sx_2$$

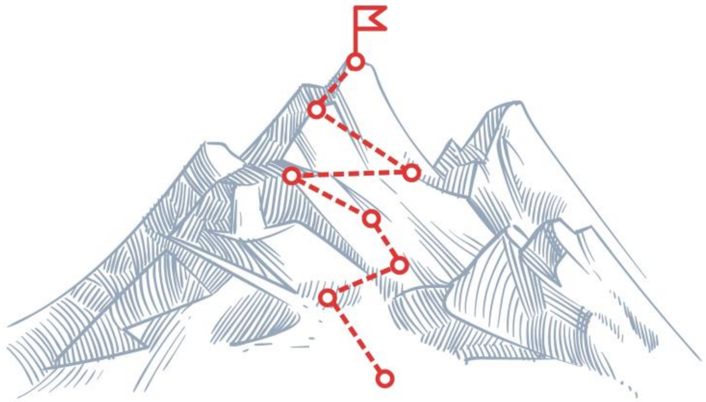


How Many Select Signals a MUX Has?

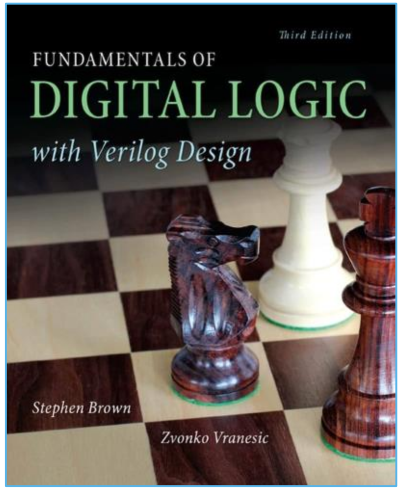
- If there are n data inputs to select from, how many select signals MUX requires?

- A: $\lceil \log_2 n \rceil$
  - Two data inputs: one select signal (1^2 combinations)
  - Four data inputs: two select signals (2^2 combinations)
  - Eight data inputs: three select signals (2^3 combinations)
  - 12 data inputs: **four** select signals because three are not sufficient
  - $2^{k-1} < \text{data inputs} \leq 2^k$, k select signals will be required

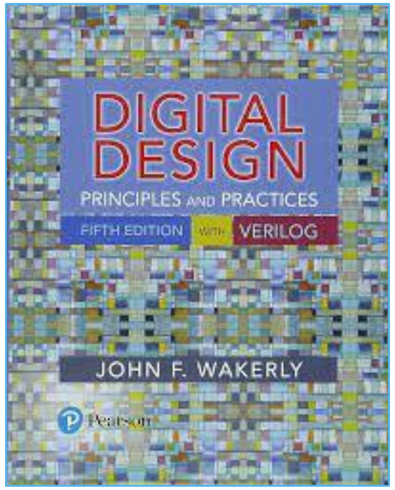




Literature



- Chapter 2: Introduction to Logic Circuits
  - 2.1-2.5



- Chapter 1: Introduction
  - 1.9
- Chapter 7: More Combinational Building Blocks
  - 7.1