ECE 271: Chapter 2 Reading Report

Phi Luu

November 13, 2018

1 Chapter Outline

This chapter covers the basics of combinational logic design and heavily focuses on the functional and timing relationships between inputs and outputs of a circuit. In this chapter, the authors use boolean and boolean algebra to explain multilevel combinational logic and apply these concepts to make hardware reduction for more complex circuits. Another focus of this chapter is showing how to use Karnaugh maps (or K-maps) to minimize logic in a more graphical and intuitive way. Each of the mentioned topics will be discussed further throughout the following subsections.

1.1 Introduction

A *circuit* is a network that processes discrete-valued variables. A circuit needs to have the following:

- one or more discrete-valued *input* terminals
- one or more discrete-valued *output* terminals
- a *functional specification* describing the relationship between the inputs and the outputs
- a *timing specification* describing the delay between the changes in the inputs and the responses of the outputs

A circuit can be viewed as a black box similar to Figure 1 below:

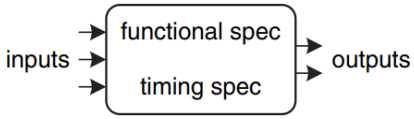


Figure 1: A circuit as a black box with inputs, outputs, and specifications

A circuit has two components:

- *Elements*: An element is itself a circuit with inputs, outputs, and specifications.
- *Nodes*: A node is a wire whose voltage is discrete-valued. Nodes are further classified into three types:
  - *Inputs* receive values from the external world.
  - *Outputs* deliver values to the external world.
  - *Internals* are wires that are neither inputs nor outputs.

Digital circuits are classified as *combinational* or *sequential*. A combinational circuit is memoryless; its outputs depend only on the current values of the inputs. Recall the logic gates from chapter 1, they are examples of a combinational circuit. A sequential circuit, on the other hand, has memory; its outputs depend on both the current values of the inputs and the previous values of the inputs. This chapter focuses on combinational circuits, and chapter 3 will cover sequential circuits.

A circuit is combinational if it consists of interconnected circuit elements such that

- Every circuit element is combinational.
- Every node is either an input to the circuit or connected to *exactly one* output terminal of an element.
- There are no cyclic paths.

The functional specification of a circuit is often expressed as a truth table or a Boolean equation which will be discussed in the next subsection.

1.2 Boolean Equations

1. **Terminology**

   Boolean equations deal with variables that are either TRUE or FALSE. Therefore, they are perfect for describing digital logic. Table 1 below shows the terminologies, notations, and truth tables of some commonly used Boolean equations.

   A *minterm* is a **product** involving all variables of the inputs to the function. A *maxterm* is a **sum** involving all variables of the inputs to the function.

   When multiple Boolean variables take part in a Boolean equation, the *order of operations* is vital to determinant the result of the equation—similarly to the order of operations in decimal-based algebra. In Boolean equations, **NOT has the highest precedence, followed by AND, and then OR**. For instance, the Boolean equation:

$$Y = \bar{A}B + BC\bar{D} \tag{1}$$

   can be interpreted as

$$Y = ((\bar{A})B) + (BC(\bar{D})) = ((\text{NOT } A) \text{ AND } B) \text{ OR } (B \text{ AND } C \text{ AND } (\text{NOT } D)) \tag{2}$$

Terminology	Notation	Truth Table		
Complement of A	$\bar{A}$	**A**		$\bar{\mathbf{A}}$
		0		1
		1		0
Product/Implicant of A and B	AB	**A**	**B**	**AB**
	A AND B	0	0	0
		0	1	0
		1	0	0
		1	1	1
Sum of A and B	$A + B$	**A**	**B**	**A + B**
	A OR B	0	0	0
		0	1	1
		1	0	1
		1	1	1

Table 1: Terminologies, notations, and truth tables of some Boolean equations

2. **Sum-of-Products Form**

A truth table of N inputs contains 2^N rows. Each row is associated with a **minterm**—a product involving all inputs of the function—**that is TRUE for that row**. Take an unknown Boolean equation Y as an example:

A	B	Y	Minterm	Minterm Name
0	0	0	$\bar{A}\bar{B}$	m_0
0	1	1	$\bar{A}B$	m_1
1	0	0	$A\bar{B}$	m_2
1	1	1	AB	m_3

Table 2: Minterms of an unknown Boolean equation Y

The minterm for the first row is $\bar{A}\bar{B}$ because $\bar{A}\bar{B}$ is TRUE when $A = 0$ and $B = 0$. Similarly, the minterm for the second row is $\bar{A}B$ because $\bar{A}B$ is TRUE when $A = 0$ and $B = 1$, and so on. Like arrays in most non-math programming languages (C, C++, Java, Javascript, Python, etc.), minterms are numbered starting with zero.

A Boolean equation can be written for any truth table by summing each of the minterms for which the output, Y, is TRUE. This is called the **sum-of-products canonical form** of a function, denoted by a *sigma notation*, Σ.

From the truth table presented by Table 2, the Boolean equation Y can be solved using the sum-of-products canonical form, as follows:

$$Y = F(A, B) = \Sigma(m_1, m_3) = \Sigma(1, 3) = \bar{A}B + AB \tag{3}$$

3. **Product-of-Sums Form**

   This method is quite similar to the sum-of-products method. Each row is associated with a **maxterm**—a sum involving all inputs of the function—**that is FALSE for that row**. Reuse Table 2 and construct the maxterm columns:

A	B	Y	Maxterm	Maxterm Name
0	0	0	$A + B$	M_0
0	1	1	$A + \bar{B}$	M_1
1	0	0	$\bar{A} + B$	M_2
1	1	1	$\bar{A} + \bar{B}$	M_3

Table 3: Maxterms of an unknown Boolean equation Y

The maxterm for the first row is $A + B$ because $A + B$ is FALSE when $A = 0$ and $B = 0$. Similarly, the minterm for the second row is $A + \bar{B}$ because $A + \bar{B}$ is FALSE when $A = 0$ and $B = 1$, and so on. Like minterms, maxterms are numbered starting with zero.

A Boolean equation can be written for any truth table as the AND of maxterms for which the output, Y, is FALSE. This is called the **product-of-sums canonical form** of a function, denoted by a *pi notation*, Π.

From the truth table presented by Table 3, the Boolean equation Y can be solved using the product-of-sums canonical form, as follows:

$$Y = E(A, B) = \Pi(M_0, M_2) = \Pi(0, 2) = (A + B)(\bar{A} + B) \tag{4}$$

It is recommended to use sum-of-products when the output has more TRUEs than FALSEs and use the product-of-sums when the output has more FALSEs than TRUEs.

1.3 Boolean Algebra

1. **Axioms**

   Axioms are statements or propositions assummed to be correct. They are unprovable in the sense that a definition cannot be proved.

   Table 4 below shows the axioms of Boolean algebra.

	Axiom		Dual	Name
A1	$B = 0$ if $B \neq 1$	A1'	$B = 1$ if $B \neq 0$	Binary field
A2	$\bar{0} = 1$	A2'	$\bar{1} = 0$	NOT
A3	$0 \cdot 0 = 0$	A3'	$1 + 1 = 1$	AND/OR
A4	$1 \cdot 1 = 1$	A4'	$0 + 0 = 0$	AND/OR
A5	$0 \cdot 1 = 1 \cdot 0 = 0$	A5'	$1 + 0 = 0 + 1 = 1$	AND/OR

Table 4: Axioms of Boolean algebra

Based on these axioms, all the theorems of Boolean algebra can be proven.

2. **Theorems of One Variable**

   Table 5 below shows the theorems of one variable in Boolean algebra.

	Theorem		**Dual**	**Name**
T1	$B \cdot 1 = B$	T1'	$B + 0 = B$	Identity
T2	$B \cdot 0 = 0$	T2'	$B + 1 = 1$	Null Element
T3	$B \cdot B = B$	T3'	$B + B = B$	Idempotency
T4	$\bar{\bar{B}} = B$			Involution
T5	$B \cdot \bar{B} = 0$	T5'	$B + \bar{B} = 1$	Complements

Table 5: Boolean theorems of one variable

3. **Theorems of Several Variables**

   Table 6 below shows the theorems of several variables in Boolean algebra

   According to De Morgan's theorem, a NAND gate is equivalent to an OR gate with inverted inputs, and a NOR gate is equivalent to an AND gate with inverted inputs.

4. **The Truth Behind It All**

   Theorems can be proven based on axioms by several methods, two of which are using truth tables and using *perfect induction*.

5. **Simplifying Equations**

   Using the theorems from Table 5 and Table 6, Equation 3 can be rewritten as

$$Y = \bar{A}B + AB = B\bar{A} + BA = B(\bar{A} + A) = B \cdot 1 = B \tag{5}$$

   and Equation 4 can be rewritten as

$$Y = (A + B)(\bar{A} + B) = (B + A)(B + \bar{A}) = B + (A\bar{A}) = B + 0 = B \tag{6}$$

   Since Equation 5 and Equation 6 are equivalent, Equation 3 and Equation 4 are also equivalent. Thus, the sum-of-products and the product-of-sums methods yield the same results. Additionally, the truth table presented by Table 2 can be simplified to the Boolean equation $Y = \bar{B}$.

   An equation in the sum-of-products form is *minimized* if it uses the fewest possible number of implicants, the minimal one is the one with the fewest literals. An implicant is called a *prime implicant* if it cannot be combined with any other implicants in the equation to form a new implicant with fewer literals. Therefore, the implicants in a minimal equation must all be prime implicants.

	Theorem		Dual	Name
T6	$B \cdot C = C \cdot B$	T6'	$B + C = C + B$	Commutativity
T7	$(B \cdot C) \cdot D = B \cdot (C \cdot D)$	T7'	$(B + C) + D = B + (C + D)$	Associativity
T8	$(B \cdot C) + (B \cdot D) = B \cdot (C + D)$	T8'	$(B + C) \cdot (B + D) = B + (C \cdot D)$	Distributivity
T9	$B \cdot (B + C) = B$	T9'	$B + (B \cdot C) = B$	Covering
T10	$(B \cdot C) + (B \cdot \bar{C}) = B$	T10'	$(B + C) \cdot (B + \bar{C}) = B$	Combining
T11	$(B \cdot C) + (\bar{B} \cdot D) + (C \cdot D)$ $= B \cdot C + \bar{B} \cdot D$	T11'	$(B + C) \cdot (\bar{B} + D) \cdot (C + D)$ $= (B + C) \cdot (\bar{B} \cdot D)$	Consensus
T12	$\overline{B_0 \cdot B_1 \cdot B_2 ...}$ $= \bar{B_0} + \bar{B_1} + \bar{B_2} ...$	T12'	$\overline{B_0 + B_1 + B_2 ...}$ $= \bar{B_0} \cdot \bar{B_1} \cdot \bar{B_2} ...$	De Morgan's Theorem

Table 6: Boolean theorems of several variables

1.4 From Logic to Gates

A *schematic* is a diagram of a digital circuit showing the elements and the wires that connect them together. For example, the schematic in Figure 2 shows a possible hardware implementation of the logic function $Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C$.

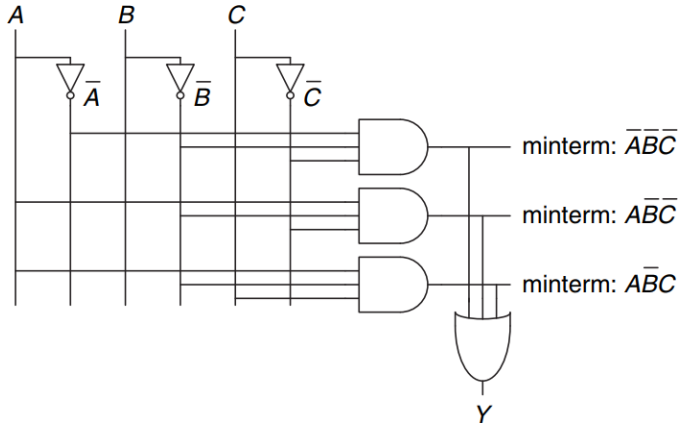


Figure 2: A schematic of $Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C$

There are several conventions to follow when drawing a schematic. Below is a list of a few of the guidelines:

- Inputs are on the left (or top) side.
- Outputs are on the right (or bottom) side.
- Gates should follow from left to right if possible.
- Straight wires are usually better than wires with multiple corners.
- Wires always connect at a T junction.

- A dot where wires cross indicates a connection between the wires.
- Wires crossing *without* a dot make no connection.

It is strongly recommended to reduce the logic function before implementing a schematic version of it.

1.5 Multilevel Combinational Logic

1. **Hardware Reduction**

   In reality, designers often build circuits with more than two levels of logic gates. These multilevel combinational circuits may use less hardware than their two-level counterparts. For example, an eight-input XOR would require 128 eight-input AND gates and one 128-input OR gate for a two-level sum-of-products implementation. A much better solution is to use a tree of seven two-input XOR gates, which drastically reduces the amount of hardware needed.

2. **Bubble Pushing**

   Bubble pushing is an intuitive way to redraw a circuit so that the bubbles cancel out and the function can be more easily determined. The guidelines for bubble pushing are as follows:

   - Begin at the output and work toward the inputs.
   - Push any bubbles on the final output back toward the inputs so that the equation can be expressed in terms of the output.
   - Work backward: if the current gate has (or does not have) an input bubble, draw the preceeding gate with (or without) an output bubble.
   - According to De Morgan's theorem, NAND will be converted to OR with inverted inputs and NOR will be converted to AND with inverted inputs.

1.6 X's and Z's, Oh My

Boolean algebra is limited to 0's and 1's. However, real circuits can also have illegal and floating values, denoted respectively by X and Z.

1. **Illegal Value: X**

   The symbol X indicates the circuit node has an *unknown* or *illegal* value. This commonly happens if it is being driven into both 0 and 1 at the same time—called *contention*—like in Figure 3 below:

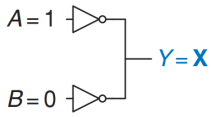
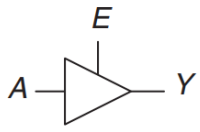


Figure 3: A circuit with contetion. Must avoid contention as it will damage the system.

2. **Floating Value: Z**

   The symbol Z indicates a node is being driven *neither* HIGH *nor* LOW. The node is said to be *floating*, *high impedance*, or *high Z*.

   A typical misconception is that a floating or undriven node is the same as a logic 0. In reality, it might be 0, might be 1, or might be "floating" at some voltage in between. Figure 4 below shows a tristate buffer and its truth table. The output of this buffer can be a HIGH, a LOW, or a Z.



E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

Figure 4: A circuit with a floating node.

1.7 Karnaugh Maps

Karnaugh maps (*K-maps*) are a graphical method for simplifying Boolean equations, invented in 1953 by Maurice Karnaugh at Bell Labs. Karnaugh maps work well for problems with up to four variables. One of the strengths of Karnaugh maps is that they use the intuition to make Boolean equations' simplification more accurate.

1. **Circular Thinking**

   Each of the rows of the truth table is represented by a single square of the K-map. The K-map also "wraps around," which means the squares on the far right are adjacent to the squares on the far left and they differ only in one variable. Reading the minterms from the K-map is exactly equivalent to reading equations in sum-of-products form directly from the truth table, as shown in Figure 5 below:

   K-map reading direction: start from the top-left, then read top to bottom and then move to the column on the right.

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

(a)

Y \ AB / C	00	01	11	10
0	1	0	0	0
1	1	0	0	0

(b)

Y \ AB / C	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$AB\bar{C}$	$A\bar{B}\bar{C}$
1	$\bar{A}\bar{B}C$	$\bar{A}BC$	ABC	$A\bar{B}C$

(c)

Figure 5: (a) truth table, (b) K-map, (c) K-map showing minterms of a three-input function.

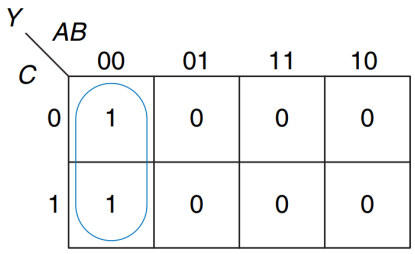


Figure 6: K-map minimization.

Using the truth table, the sum-of-products form, and Boolean algebra, the given logic function can be expressed as

$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C = \bar{A}\bar{B}(\bar{C} + C) = \bar{A}\bar{B} \tag{7}$$

Using the K-map, graphically **circle the minterms in adjacent square**:

For each circle, write a corresponding implicant (AND). Variables whose true *and* complementary forms both appear in the circle are excluded from the implicant. In Figure 6, the variable C is excluded from the implicant, leaving $Y = \bar{A}\bar{B}$—the same result as Equation 7.

2. **Logic Minimization with K-Maps**

   K-maps provide an easy visual way to minimize logic. Follows the guidelines below to find a minimized equation from a K-map:

   - Use the fewest circles possible to cover all the 1's.
   - Everything in the circles must contain 1's.
   - Each circle must span 2^n rectangular blocks.
   - Each circle should be as large as possible.
   - A circle may wrap around the edge of the K-map.
   - A 1 in a K-map must be circled *multiple times*, allowing fewer circles to be used.

3. **Don't Cares**

   The "don't care" entries in a truth table indicate some variables do not affect the output, denoted X. The "don't care" entries can either be 0 or 1.

   X appear in truth table outputs where the value is unimportant or the corresponding input combination can never happen. These outpus can be treated by either 0's or 1's depending on the designer. In a K-map, X allows even more logic minimization.

4. **The Big Picture**

   Boolean algebra and Karnaugh maps are two methods of logic simplification. In modern engineering practice, *logic synthesizers* are computer programs that produce simplified circuits from a description of a logic function much more efficiently than humans.

1.8 Combinational Building Blocks

1. **Multiplexer**

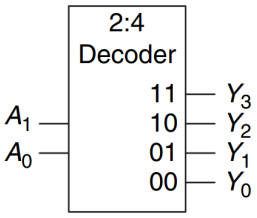
   Multiplexers are among the most commonly used combinational circuits. They choose an output from among several possible inputs based on the value of a *select* signal. A multiplexer is often called a *mux*.

   A multiplexer can be built from sum-of-products logic or from tristate buffers.

   Multiplexers can be used as *lookup tables* to perform logic functions.

2. **Decoders**

   A decoder has N inputs and 2^N outputs. It asserts *exactly one* of its output depending on the input combination. The outputs are called *one-hot*, as exactly one output is "hot" (HIGH) at a given time.



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Figure 7: A 2:4 decoder and its truth table.

1.9 Timing

Previous subsections discuss the functional aspect of a circuit—that is, whether the circuit works ideally with the fewest gates. On the other hand, one of the most challenging issues in circuit design is *timing*: making a circuit run *fast*. This subsection focuses on this aspect of a circuit.

1. **Propagation and Contamination Delay**

   Combinational logic is characterized by its *propagation delay* and *contamination delay*. The **propagation delay** t_{pd} is the **maximum** time from when an input changes until the output(s) reach their final value. The **contamination delay** t_{cd} is the **minimum** time from when an input changes until any outputs starts to change its value.

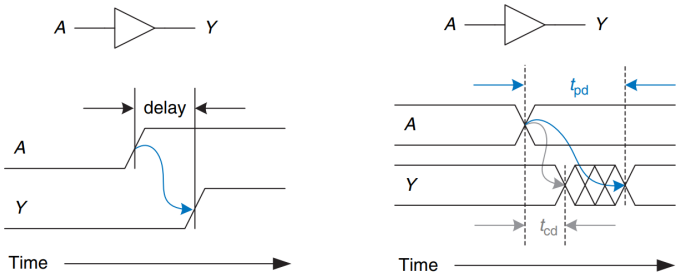


Figure 8: Propagation delay and contamination delay

2. **Glitches**

   It is possible that a single input transistion can cause multiple output transistions. These are called *glitches* or *hazards*. Although glitches don't usually cause problems, it is important to realize they exist and recognize them when looking at the timing diagram.

   In general, a glitch can occur when a change in a single variable crosses the boundary between two prime implicants in a K-map. A glitch can be eliminated by covering these boundaries with redundant implicants—with the cost of extra hardware. However, simultaneous transitions on multiple inputs can also cause glitches—and these glitches cannot be fixed by adding hardware.

1.10 Summary

A digital circuit is a module with discrete-valued inputs and outputs and the functional and timing relationships between the two. This chapter has focused on combinational circuits—circuits that depend only on the current value of their inputs.

The function of a combinational circuit can be given by a truth table or a Boolean equation. A logic function can be written from the truth table using the sum-of-products or the product-of-sums methods.

Boolean equations can be simplified using Boolean algebra or Karnaugh maps. Logic simplification makes the logic function require less hardware, thus effectively reduce the cost of hardware implementation.

Logic gates are connected to form combinational circuits that perform the desired function. Any function in sum-of-products form can be built using two-level logic: NOT gates from the complements of the inputs, AND gates from the products, and OR gates from the sum. When using NAND and NOR, use bubble pushing to keep track of the inversions.

Logic gates are combined to produce larger circuits, such as multiplexers (mux), decoders, and priority circuits. A multiplexer can choose the data from a select input. A decoder sets one of the outputs HIGH depending on the inputs. A priority circuit produces an output indicating the highest priority input.

Timing is one of the most challenging issues in circuit design. The timing specification consists of the propagation and the contamination delay—indicating the longest and shortest time it takes for the outputs to change after the inputs change, respectively. There are many different ways to implement combinational circuits, and these ways offer trade-offs between speed and cost.

2 Grey Box Exploration

1. The first blurb is on page 61, which states: *The null element theorem leads to some outlandish statements that are actually true! It is particularly dangerous when left in the hands of advertisers: YOU WILL GET A MILLION DOLLARS or we'll send you a toothbrush in the mail. (You'll most likely be receiving a toothbrush in the mail.)*

   Usually, when people talks about advertisement, they often assess the truth of the advertised statements. The example provided by the blurb, "YOU WILL GET A MILLION DOLLARS

or we'll send you a toothbrush in the mail," is an advertised statement. To assess the true-false of this statement, Boolean algebra is required.

Let the clause "YOU WILL GET A MILLION DOLLARS" be a Boolean variable, called A. Let the clause "we'll send you a toothbrush in the mail" be another Boolean variable, called B. Below is the truth table of A OR B.

A	**B**	**A OR B**
0 No dollars	0 No toothbrush in the mail	0 FALSE advertisement
0 No dollars	1 A toothbrush in the mail	1 TRUE advertisement
1 A million dollars	0 No toothbrush in the mail	1 TRUE advertisement
1 A million dollars	1 A toothbrush in the mail	1 TRUE advertisement

Therefore, the advertisement is still TRUE if the person receives no dollars but a toothbrush. No company is generous enough to give a random person a million dollars.

Alternatively, the advertisement can be assessed using the Null Element Theorem. In order for the advertisement to be TRUE, the company has to give out a toothbrush. Using the Null Element theorem of Boolean algebra $B + 1 = 1$, B can either be a million dollars or be no dollars. This leads back to the generosity of the company. In the end, the person will most likely get a toothbrush in the mail.

2. The second blurb is on page 63, which states: *Augustus De Morgan, died 1871. A British mathematician, born in India. Blind in one eye. His father died when he was 10. Attended Trinity College, Cambridge, at age 16, and was appointed Professor of Mathematics at the newly founded London University at age 22. Wrote widely on many mathematical subjects, including logics, algebra, and paradoxes. De Morgan's crater on the moon is named for him. He proposed a riddle for the year of his birth: "I was x years of age in the year x^2."*

   De Morgan was a mathematical prodigy who wrote *Trigonometry and Double Algebra*. He was among the Cambridge mathematicians who recognized the purely symbolic nature of algebra. He also gave an interpretation of complex numbers, involving a term with a $\sqrt{-1}$.[1]

   Since De Morgan was born in 1806, in the year x^2, he was $x^2 - 1806 = x$ years of age. x can be solved using the quadratic formula:

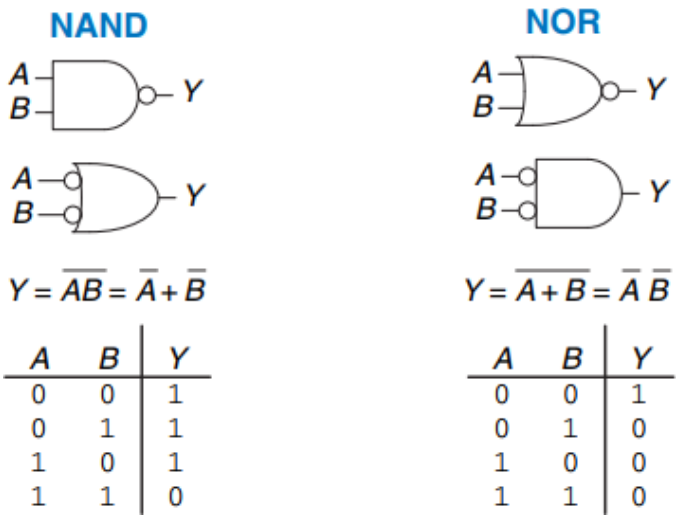
$$x^2 - x - 1806 = 0 \Leftrightarrow x = \frac{-(-1) \pm \sqrt{(-1)^2 - 4 \cdot 1 \cdot (-1806)}}{2 \cdot 1} = \frac{1 \pm 85}{2} \Leftrightarrow x = 43 \text{ or } x = -42 \tag{8}$$

   Since the age is non-negative, De Morgan was 43 years old in the year $43^2 = 1849$.

   This riddle is strictly determinate by the century of its utterance and the limit to human life. Those who born in 1722 (1764-42), 1892 (1936-44), and 1980 (2025-45) are similarly privileged.[1]

3 Figures

Two figures were selected from this chapter for special recognition. Figure 9 was selected for intuitively showing the logic gate equivalence of the De Morgan's theorem in Boolean algebra and briefly showing the bubble pushing method in logic simplification. The NAND gate is equivalent to an OR gate with inverted inputs. Similarly, a NOR gate is equivalent to an AND gate with inverted inputs. When using the bubble pushing method to simplify logic functions, the bubble (a NOT node) on the output side will be pushed to all of the inputs, and the gate will change accordingly (NAND to OR and NOR to AND).



A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Figure 9: De Morgan equivalent gates and their truth tables

Figure 10 was selected for showing the drastic hardware reduction using multilevel combinational logic. Normally, to construct an eight-input XOR gate, there would be 128 eight-input AND gates and one 128-input OR gate. Nevertheless, using the associative property of XOR, $A \oplus B \oplus C = (A \oplus B) \oplus C$, an eight-input XOR gate can also be produced by using a tree of seven two-input XOR gates. There is a huge difference in hardware monetary cost and hardware space cost between the two methods of implementation.

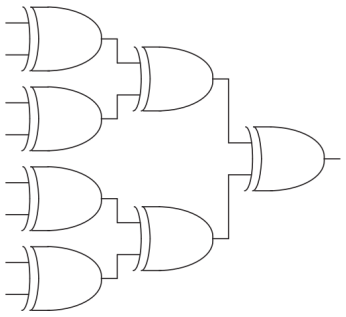


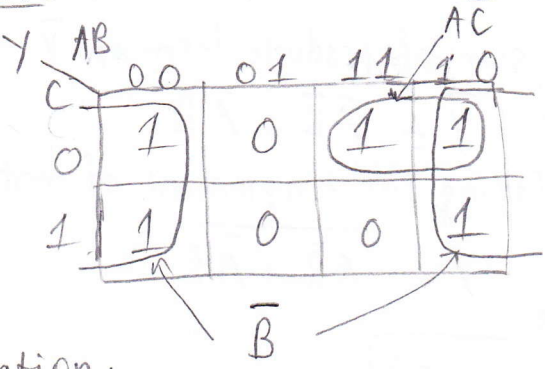
Figure 10: An eight-input XOR constructed from seven two-input XOR

4 Example Problems

See the attached images on the next pages.

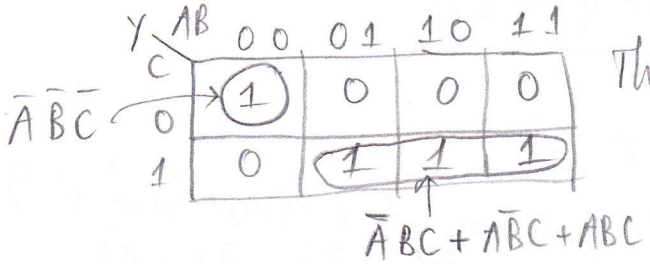
EXAMPLE PROBLEMS

Example 2.9: Minimization of a three-variable function using a K-map



Thus, $\boxed{Y = A\bar{C} + \bar{B}}$

Variation:



Thus, $Y = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C + ABC$

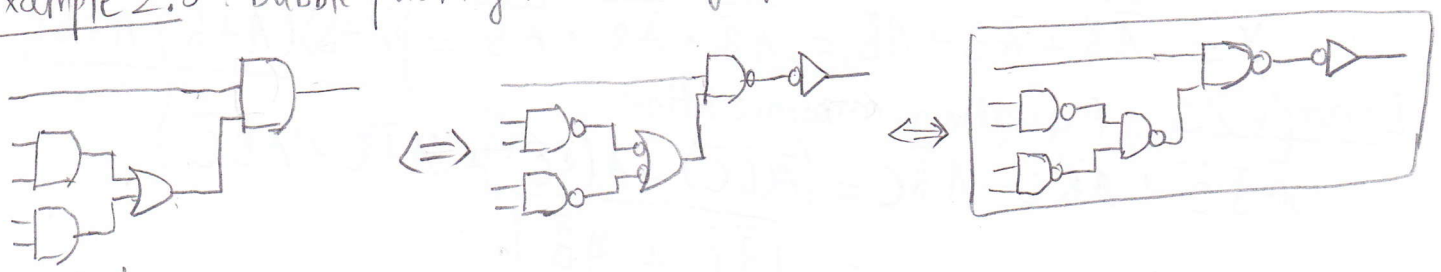
$= \bar{A}\bar{B}\bar{C} + (\bar{A}B + A\bar{B} + AB)C$

$= \bar{A}\bar{B}\bar{C} + (\bar{A}B + A)C$

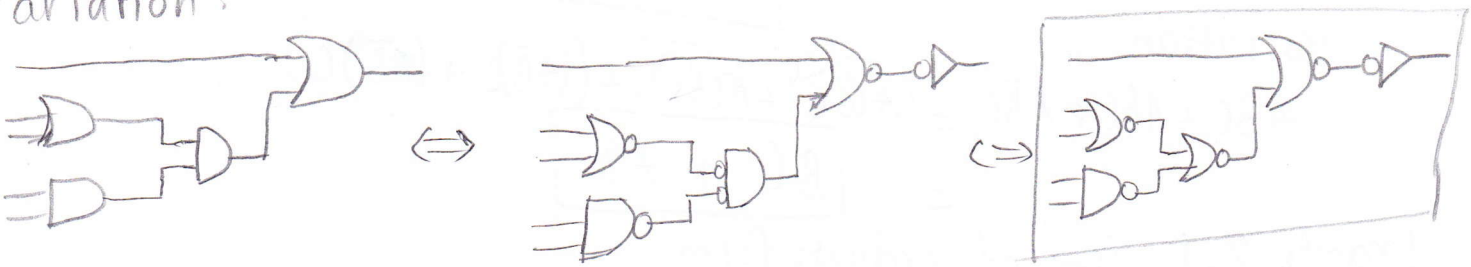
$= \bar{A}\bar{B}\bar{C} + (A + B)C$

$\boxed{Y = \bar{A}\bar{B}\bar{C} + AC + BC}$

Example 2.8: Bubble pushing for CMOS logic



Variation:



Example 2.4: Derive the product-of-sums form

A	B	Y	$\bar{Y}$	minterm
0	0	0	1	$\bar{A}\bar{B}$
0	1	0	1	$\bar{A}B$
1	0	1	0	$A\bar{B}$
1	1	1	0	AB

$\Rightarrow$ sum-of-products form of $\bar{Y}$:

$\bar{Y} = \bar{A}\bar{B} + \bar{A}B$

Taking the complement of both sides:

$Y = \overline{\bar{A}\bar{B} + \bar{A}B}$

Using De Morgan's theorem twice:

$Y = \overline{\bar{A}\bar{B}} \cdot \overline{\bar{A}B} = \boxed{(A+B)(A+\bar{B})}$

Variation:

A	B	Y	$\bar{Y}$	minterm
0	0	0	1	$\bar{A}\bar{B}$
0	1	0	1	$\bar{A}B$
1	0	0	1	$A\bar{B}$
1	1	1	0	AB

$\Rightarrow$ sum-of-products form of $\bar{Y}$

$\bar{Y} = \bar{A}\bar{B} + \bar{A}B + A\bar{B}$

Taking the complement of both side and applying DeMorgan's theorem:

$Y = \bar{A}\bar{B} + \bar{A}B + A\bar{B} = \overline{\bar{A}\bar{B}} \cdot \overline{\bar{A}B} \cdot \overline{A\bar{B}} = \boxed{(A+B)(A+\bar{B})(\bar{A}+B)}$

Example 2.6: Equation minimization

$\bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C = (\bar{A}(\bar{B}\bar{C}) + A(\bar{B}\bar{C})) + ((A\bar{B})C + (A\bar{B})\bar{C})$

$= \boxed{\bar{B}\bar{C} + A\bar{B}}$

Variation:

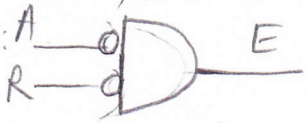
$\bar{A}BC + ABC + A\bar{B}C = (\bar{A}(BC) + A(BC)) + ((A\bar{B})C + (A\bar{B})C)$

$= \boxed{BC + A\bar{B}}$

Example 2.1: Sum-of-products form

A = there are ants, R = It rains, E = Ben enjoys the picnic

$\Rightarrow E = \bar{A}\bar{R} = \Sigma(0)$ $\Rightarrow$ circuit:



Variation:

L = Ben feels lazy, R = It rains, S = Ben skips class

$\Rightarrow S = LR = \Sigma(3)$ $\Rightarrow$ circuit:

5 Glossary

All definitions were found from the Google search engine, typing "circuit dictionary" for the first item.

1. Circuit

   noun:

   (a) a roughly circular line, route, or movement that starts and finishes at the same place.

   (b) an established itinerary of events or venues used for a particular activity, typically involving public performance.

   verb:

   (a) move all the way around (a place or thing).

2. Boolean

   adjective:

   (a) denoting a system of algebraic notation used to represent logical propositions, especially in computing and electronics.

   noun:

   (a) a binary variable, having two possible values called "true" and "false."

3. Combination

   noun:

   (a) a joining or merging of different parts or qualities in which the component elements are individually distinct.

   (b) a sequence of numbers or letters used to open a combination lock.

4. Axiom

   noun:

   (a) a statement or proposition that is regarded as being established, accepted, or self-evidently true.

5. Theorem

   noun:

   (a) a general proposition not self-evident but proved by a chain of reasoning; a truth established by means of accepted truths.

6 Interview Question

See the attached image on the next page.

INTERVIEW QUESTION

Question 2.1: Sketch a schematic for the two_input XOR function using only NAND gate. How few can you use?

A	B	$A \oplus B$	minterm
0	0	0	$\bar{A}\bar{B}$
0	1	1	$\bar{A}B$
1	0	1	$A\bar{B}$
1	1	0	AB

Sum-of-products:

$$A \oplus B = \bar{A}B + A\bar{B}$$

$$= \overline{\overline{\bar{A}B + A\bar{B}}} \quad \text{(Involution)}$$

$$= \overline{\overline{\bar{A}B} \cdot \overline{A\bar{B}}} \quad \text{(De Morgan)}$$

$$= \overline{(A+\bar{B})(\bar{A}+B)} \quad \text{(De Morgan)}$$

$$= \overline{(A+\bar{B})\bar{A} + (A+\bar{B})B} \quad \text{(Distributivity)}$$

$$= \overline{A\bar{A} + \bar{A}\bar{B} + AB + B\bar{B}} \quad \text{(Distributivity)}$$

$$= \overline{0 + \bar{A}\bar{B} + AB + 0} \quad \text{(Complements)}$$

$$= \overline{AB + \bar{A}\bar{B}} \quad \text{(Identity \& Associativity)}$$

$$= \overline{AB} \cdot \overline{\bar{A}\bar{B}} \quad \text{(De Morgan)}$$

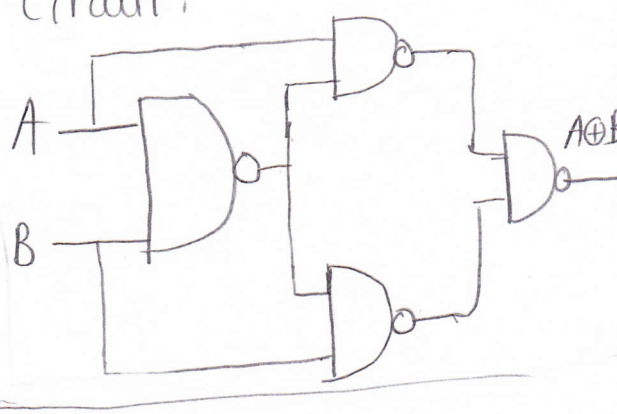
$$= \overline{AB} \cdot (A+B) \quad \text{(De Morgan)}$$

$$= \overline{AB}A + \overline{AB}B \quad \text{(Distributivity)}$$

$$= \overline{\overline{AB}A + \overline{AB}B} \quad \text{(Involution)}$$

$$= \overline{\overline{AB}A} \cdot \overline{\overline{AB}B} \quad \text{(De Morgan)}$$

Ciruit:



7 Reflection

I believe it is important to understand this chapter's materials well because this chapter establishes the idea of circuits, logic, and schematics and connects mathematics and physics to circuit design. Since I'm a Computer Science student who has a hobby in building circuits and making small robots, I found Boolean algebra, combinational circuits, and logic simplification quite interesting. What impressed me the most in this chapter was the Karnaugh map. I will definitely use the map as a graphical and intuitive way to double-check the correctness when I simplify logic functions.

The Boolean algebra and Karnaugh maps sections of the chapter were fairly easy. Though, I'm still trying to make sense of the multiplexer and the tristate buffer and their applications in circuit design. I am looking forward to learn more about combinational circuits, hardware reduction, and combinational building blocks in lecture.

8 Questions for Lecture

1. What are the applications of multiplexers? Can you show me a real-life example of a multiplexer?

2. What determines which input will be taken into account in a tristate buffer?

3. Electronic devices, such as smartphones, are getting smaller and faster but also more expensive. Do you favor speed or cost?

References

[1] Wikipedia, "Augustus de morgan." https://en.wikipedia.org/wiki/Augustus_De_Morgan, 2018.