Chapter 2: Combinational Logic Design

Note of Digital Design and Computer Architecture
2023 Oct 10 - 2023 Oct 24

Nemo

Contents

1 Introduction

In digital electronics, a circuit is a network that processes discrete-valued variables. A circuit can be viewed as a blackbox with:

- one or more input terminals
- one or more output terminals
- a functional specification describing the relationship between inputs and outputs
- a timing specification describing the delay between inputs changing and outputs responding

Digital circuits are classified as *combinational* or *sequential*. A combinational circuit's outputs depend only on the current inputs, while sequential circuit's output depend both current and previous inputs. That is to say, combinational circuits are *memoryless*, but sequential circuits have *memory*.

Note that there may be several different *implementation* for one single function.

Theorem* 1: Rules of *combinational composition

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

- Every circuit element is itself combinational.
- Every node of the cricuit is either designated as an input to the circuit or connects to **exactly one** output terminal of a circuit element.
- The circuit contains no cyclic paths: every path through the circuit visits each cricuit node at most once.

Note:

The rules of combinational composition is not a necessary condition of one circuit to be combinational. That is to say, some circuits can still be combinational despite that they disobey the rules. However that does not implies that the rule is useless, by following the rules, the circuit we disign are sure to be combinational. Just note that the disobeying the rules dose not mean the circuit is not combinational.

2 Boolean Equations

2.1 Terminology

The *complement* of a variable A is its inverse $\overline{A}$. The variable or its complement is called a *literal*. We call A the *true form* of the variable and $\overline{A}$ the *complementary form*; "ture form" only means that there is no line over the letter "A" and do not imply the variable A is TRUE.

The AND of one or more literals is called a *product* or an *implicant*, like AB, $A\overline{B}\,\overline{C}$. A *minterm* is a product involving all the inputs to the function. e.g. $AB\overline{C}$ is a minterm for a function of three variables A, B and C, but AB is not.

The OR or one or more literals is called a *sum*, a *maxterm* is a a sum involving all of the inputs to the function.

2.2 Sum-of-Products Form

A truth table of N inputs contains 2^N rows. Each row in a truth table is associated with a minterm that is TRUE for that row, e.g.:

A	B	Y	minterm	minterm name
0	0	0	$\overline{A}\,\overline{B}$	m_0
0	1	1	$\overline{A}B$	m_1
1	0	0	$A\overline{B}$	m_2
1	1	1	AB	m_4

We can wirte a Boolean equation for any truth table by summing each of the minterms for which the output, Y, is TRUE. Take the table above as an example:

A	B	Y	minterm	minterm name
0	0	0	$\overline{A}\,\overline{B}$	m_0
0	1	1	$\overline{A}B$	m_1
1	0	0	$A\overline{B}$	m_2
1	1	1	AB	m_4

So, $Y = m_1 + m_4 = \overline{A}B + AB$. This can be also written in the sigma notation: $F(A, B) = \sum(m_1, m_3)$ or even simpler, $\sum(1, 3)$

2.3 Product-of-Sums Form

An alternitive way of expressing Boolean functionsis the product-of-sums canonical form. Each row of the turth table corresponds to a maxterm that is false for that row. Then we can wirte a Boolean equation for the truth table as the AND of each of the maxterm for which the output is FALSE.

3 Boolean Algebra

3.1 Axioms

$$B = 0 \text{ if } B \neq 1$$

$$\overline{0} = 1$$
$$0 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$
$$1 \cdot 0 = 0 \cdot 1 = 0$$

3.2 Theorem of One Variable

***Theorem* 2: The identity theorem**

For any Boolean variable B, B AND $1 = B$.
(Dual: B OR $0 = B$)

***Theorem* 3: The null element theorem**

For any Boolean variable B, B AND $0 = 0$, so "0" is called the null element for the AND operation.
(Dual: "1" is the null element for OR operation.)

***Theorem* 4: Idempotency**

A variable AND itself is equal to just itself.
(Dual: A variable OR itself is equal to just itself.)

***Theorem* 5: Involution**

$\overline{\overline{B}} = B$, complementing a variable twice results in the original variable.
(mathematical definition for involution: In mathematics, an involution, involutory function, or self-inverse function is a function f that is its own inverse.)

***Theorem* 6: The complement theorem**

$B \cdot \overline{B} = 0$, a variable AND its complement is always 0.
(Dual: $B + \overline{B} = 1$,a variable OR its complement is always 1.)

3.3 Theorem of Several Variables

***Theorem* 7: Commutativity and associativity**

$B \cdot C = C \cdot B, B + C = C + B$

$(B \cdot C) \cdot D = B \cdot (C \cdot D), (B + C) + D = B + (C + D)$

***Theorem* 8: Distributivity**

$B \cdot C + B \cdot D = B \cdot (C + D)$

$(B + C) \cdot (B + D) = B + C \cdot D$

Note that the second equation does not hold in triditional algebra.

***Theorem* 9: Covering**

$B \cdot (B + C) = B, B + (B \cdot C) = B$

***Theorem* 10: Consensus**

$BC + \overline{B}D + CD = BC + \overline{B}D$

$(B + C) \cdot \left(\overline{B} + D\right) \cdot (C + D) = (B + C) \cdot \left(\overline{B} + D\right)$

***Theorem* 11: De Morgan's Theorem**

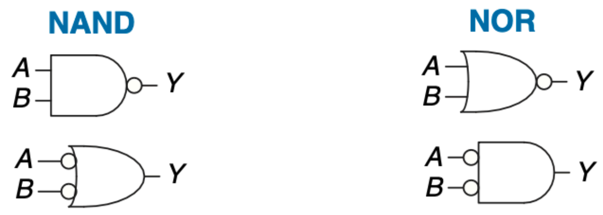
$\overline{B_0 \cdot B_1 \cdot B_2 ...} = \overline{B_0} + \overline{B_1} + \overline{B_3} + ...$

$\overline{B_0 + B_1 + B_2 + ...} = \overline{B_0} \cdot \overline{B_1} \cdot \overline{B_3} ...$

Note:

When trying to prove these theorems, note that the null element is the crucial clue to the solution.

According to the De Morgan's Theorem, we have the following equivalence:



NAND: $Y = \overline{AB} = \overline{A} + \overline{B}$

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

NOR: $Y = \overline{A + B} = \overline{A}\,\overline{B}$

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

The inversion circle is called a *bubble*, and it has several properties:

- Pushing the bubble from the output to the input changes the body of the gate (AND $\rightarrow$ OR or OR $\rightarrow$ AND) and vice versa.
- Pushing a bubble form the output to the inputs puts bubbles on all the gate inputs.
- Pushing all the bubbles form the input to the output puts a bubble on the output.

3.4 The Truth Behind It All

Proving the theorems are easy because that they only contain finit numebr of variables. We can simply list all the possible values of the variables, this method is called *perfect induction*.

3.5 Simplifying Equations

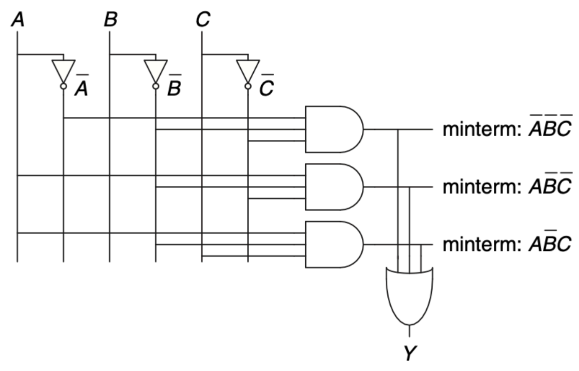
e.g. 1 Minimize equation $\overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + A\overline{B}C$

Upon first sight, I'll try to simplify the expression to $\overline{B}\,\overline{C} + A\overline{B}C$ or $\overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}$ and I'm stuck here. It seems that we can either combine the first two or the last two and no more, because the miniterm $A\,\overline{B}\,\overline{C}$ is used to either form $\overline{B}\,\overline{C}$ or $A\,\overline{B}$. We call this situation $\overline{B}\,\overline{C}$ and $A\,\overline{B}$ *share* the miniterm $A\,\overline{B}\,\overline{C}$. Now the magic! Remember the ***Idempotency Theorem*** ? We can duplicant terms as many times as we want! It's an important feature of Boolean algebra. So actually we can minimize the expression to $\overline{B}\,\overline{C} + A\,\overline{B}$. Amazing!

4 Form Logic to Gates

A schematic is a diagram of a digital circuit showing the elements and the wires that connect them together, like the figure below, which is the implementation of the function:

$$Y = \overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + A\overline{B}C$$


A
B
C
$\overline{A}$
$\overline{B}$
$\overline{C}$
minterm: $\overline{A}\overline{B}\overline{C}$
minterm: $A\overline{B}\overline{C}$
minterm: $A\overline{B}C$
Y

To ensure consistency, readability, we make these following rules:

- Inputs on left or top
- Outputs on right or bottom
- Whenever possible, gates should flow from left to right
- Straight wires
- Wires connect at a T junction
- A dot where wires cross indicates a connection
- Wires cross without a dot make no connection

The style of the schematic above is called a *programmable logic array (PLA)* bacause the NOT gates, AND gates, and OR gates are arrayed in a systematic fashion.

e.g. Multiple-output circuits

Priority Circuit : For input A_0, A_1, A_2, A_3, let's say they have weights equal to their subscripts. (A_3 have the greatest weight 3, and A_0 have the weight 0) There corresponding output are Y_0, Y_1, Y_2, Y_3. When $A_3 = 1$ then $Y_3 = 1$ regardless whatever the other variables. The output is determined by the input and there weight.

How to implement it then?

We can list the truth table.The X in the table stands for the value does matter. Though that we can indeed use the sum of product method to work out the Boolean equation, but it's acutally easier to derive them from the function descirption. We simply miss out the variables that does matter (marked X in the table).

$$\begin{aligned} Y_0 &= \overline{A_3}\,\overline{A_2}\,\overline{A_1}A_0 \\ Y_1 &= \overline{A_3}\,\overline{A_2}A_1 \\ Y_2 &= \overline{A_3}A_2 \\ Y_3 &= A_3 \end{aligned}$$

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	X	0	0	1	0
0	1	X	X	0	1	0	0
1	X	X	X	1	0	0	0

5 Multilevel Combinational Logic

In previous sections we have the Sum-of-Products form, which is called two level logic for it contians two levels of logic AND and then OR. IRL, usually there are multiple levels of logic, and they tend to use less hardware than there two level conterparts. Bubble pushing is especiall helpful in analyzing and designing multilevel circuits.

5.1 Hardware Reduction

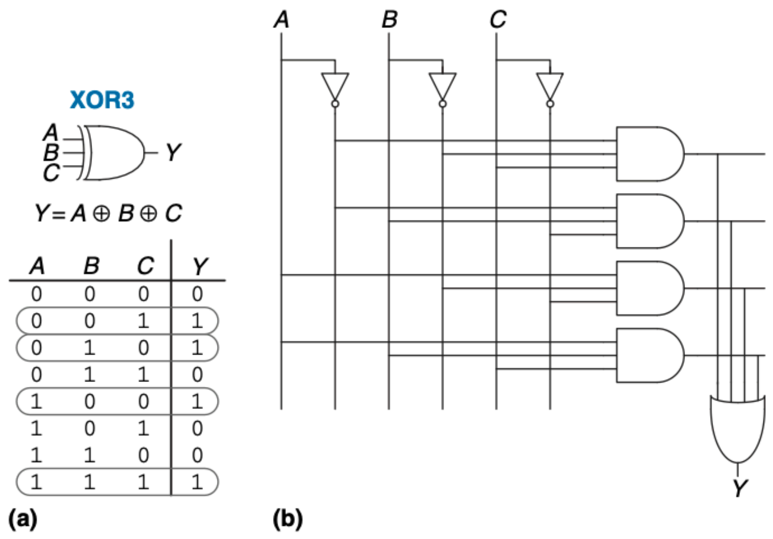
Quote from book:(easy to understand)

Recall that an N-input XOR produces a TRUE output when an odd number of inputs are TRUE. Figure 2.30 shows the truth table for a three-input XOR with the rows circled that produce TRUE outputs. From the truth table, we read off a Boolean equation in sum-of-products form in Equation 2.6. Unfortunately, there is no way to simplify this equation into fewer implicants.

$$Y = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC \tag{2.6}$$

On the other hand, $A \oplus B \oplus C = (A \oplus B) \oplus C$ (prove this to yourself by perfect induction if you are in doubt). Therefore, the three-input XOR can be built out of a cascade of two-input XORs, as shown in Figure 2.31.

Similarly, 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 option is to use a tree of two-input XOR gates, as shown in Figure 2.32.



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

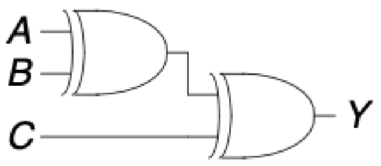


Figure 2.31 Three-input XOR using two-input XORs

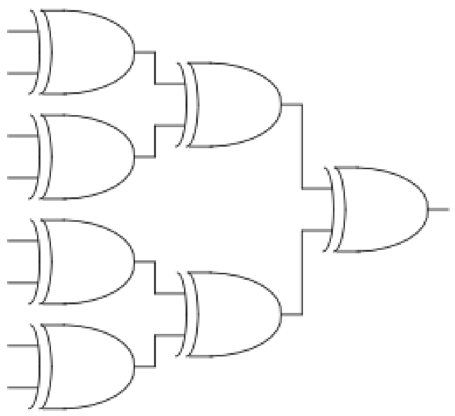


Figure 2.32 Eight-input XOR using seven two-input XORs

5.2 Bubble Pushing

6 X's and Z's

Boolean algebra is limited to 1′s and 0′s ,but IRL there may be illegal and floating values.

6.1.1 Illegal Value: X

The symbol X indicates that the circuit node has an unknown or illegal value. This is commonly coursed by a node is being driven to both 0 and 1 at the same time. This is called contention.

6.1.2 Floating Value: Z

The symbol Z indicates that a node is being driven neither HIGH nor LOW. The node is said to be floading, high impedance, or high Z. This can be coursed by forget to connect votage to the inputs or assume that an unconneted input is the same as 0.

7 Karnaugh Maps

Karnough maps (or *K-maps*) are a great graphical method for simplifying Boolean equations. K-maps work well for equations up to 4 variables.

The order of the variables is in the form of Grey Code.

Pules for fingding a minimized equation from a K-map are as follows:
- Use the fewest circles necessary to cover all the 1′s
- All the squares in each circle must contain 1′s
- Each circle must span a rectangular block that is a power of 2

- Each circle should be as large as possible
- A circle may warp around the edges of the K-map
- A 1 in a K-map may be circled multiple times if doing so alllows fewer circles to be used

The circles in the K-map each represent a prime implicant, and only the variables that have either true form or complementary form (NOT BOTH) in the circle appear in the prime implicant. That is to say, the variables that have both true form and complementary form in the circle are eliminated.

7.1 Don't cares

X's can also be helpful in the K-maps, they can either be circuled or not, just make sure that the fewest circules are used.

8 Combinational Building Blocks

Combinational logic is often grouped into larger building blocks to build a more complex systems. This is acutally the process of abstraction, we hide the lower part of logic gate details and just focus on the inputs and outputs.

8.1 Multiplexers

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 sometimes affectionately called a *mux*.

8.1.1 2:1 Multiplexer

Multiplexer is actually a selector that outputs the votage of the selected input. It can be used as a *lookup table*.

8.1.2 Wider Multiplexers

There are some wider multiplexers that accept more than one select signals and more inputs, to be exact, there are n select signals and 2^n inputs.

8.2 Decoder

A decoder has N inputs and 2^n outputs, it asserts exactly noe of its outputs depending on the inputs combination. The outputs are called *one-hot* because exactly one is "Hot" (HIGH) at a given time.