Switching Circuits and Logic Design

沈威宇

September 27, 2025

Contents

Chapter 1 Switching Circuits and Logic Design

1 Number Systems

I Definition

Let the base (aka radix) $b \in \mathbb{N}_{>1}$. A base-b (aka radix-b) number

$$N = \left((-1)^m \dots d_2 d_1 d_0 . d_{-1} d_{-2} \dots\right)_b, \quad m \in \{0,1\} \wedge \forall i : d_i \in \mathbb{N}_0 \wedge d_i < b$$

represents the value

$$N = (-1)^m \sum_i d_i b^i.$$

We can only represent a base in another base or in decimal (base-10), and we have to specify the base you use if not in decimal; otherwise, every base would be base-10.

To indicate the base of a base-b number, we append subscript $_b$ after it. We also use prefix `0b` (or `0B`) to represent base-2 numbers, aka binary numbers, prefix `0` (or `0o`, `0O`) to represent base-8 numbers, aka octal numbers, prefix `0x` (or `0X`) to represent base-16 numbers, aka hexadecimal number, and no prefix (or `0d`, `0D`) to represent base-10 numbers, aka decimal numbers.

Usually, when the radix $b > 10_{10}$, A (or a) is used to represent the digit $9+1$; when the radix $b > 11_{10}$, B (or b) is used to represent the digit $9+2$; and so on.

II Conversion between bases

i General conversion method

We want to convert a number N in base x with $x \in \mathbb{N}_{>1}$,

$$N = ((-1)^m \dots d_2 d_1 d_0 . d_{-1} d_{-2} \dots)_x, \quad m \in \{0,1\},$$

into base $y \neq x$ (written in base x) with $y \in \mathbb{N}_{>1}$.

1. First, we take $I_1 = (\dots d_2 d_1 d_0)_x$ and repeatedly:
   (a) take $r_i = I_i \mod y$, convert it to base y, of which the result must be a single digit, and
   (b) take $I_{i+1} = \dfrac{I - r_i}{y}$,
   in the ith (start from 1st) time, until the first time $I_{i+1} = 0$, and let that time be the kth time.
2. Second, we take $J_1 = (0.d^{-1} d^{-2} \dots)_x$ and repeatedly

(a) take $s_j = \lfloor J_j \cdot y \rfloor$, which may be more than 1 digits if $x > y$ and is later treated directly as in base y, and

(b) take $J_{j+1} = J_j \cdot y - s_j$,

in the jth (start from 1st) time, until the first time $J_{j+1} = 0$, and let that time be the lth time.

3. Then, N is

$$(-1)^m \left(\sum_{i=1}^{k} r_i \times y^{i-1} + \sum_{j=1}^{l} s_j \times y^{-j} \right),$$

and N in base y can be obtained by first taking the integer part of the magnitude of it:

$$(r_k r_{k-1} \dots r_2 r_1)_y$$

and then obtaining the fractional part of the magnitude of it using vertical addition where the ith row is s_i with the units digit of s_i in the ith digit to the right of the radix point, for each i, where carry propagation is applied to ensure all digits are valid in base y, i.e. $< y$.

To convert an expression that is a radix-independent fuction f of a vector of numbers $(p_1, p_2, \dots)$, we convert each p_i to the wanted base, let it be q_i, then the wanted result is $f(q_1, q_2, \dots)$.

ii Power-base conversion method

To convert a base-b number into base b^n with $b, n \in \mathbb{N}_{>1}$, we group the digits to the left and right of the radix point into blocks of n digits starting from the digits closest to the radix point and convert each block to one digit in base b^n.

iii Root-base conversion method

To convert a base-b^n number in into base b with $b, n \in \mathbb{N}_{>1}$, we convert each digit to n digits in base b and concatenate them.

iv Product and quotient-base conversion method

We want to convert a number in base x with $x \in \mathbb{N}_{>1}$,

$$N = ((-1)^m \dots d_2 d_1 d_0 . d_{-1} d_{-2} \dots)_x, \quad m \in \{0, 1\},$$

into base $y = ax$ (written in base x) with $a \in \mathbb{N}_{>1} \vee \frac{1}{a} \in \mathbb{N}_{>1}$.

We take $D_i = d_i \cdot a^{-i}$, which is later treated directly as in base y, for each integer i.

Then, N is

$$(-1)^m \sum_i D_i \times y^i,$$

and N in base y can be obtained by vertical addition where the ith row is D_i with the units digit of D_i in the $(i+1)$th digit to the left of the radix point, for each i, where carry propagation is applied to ensure all digits are valid in base y, i.e. $< y$.

v Criterion of finite expansion after base conversion

A finite rational number with the fractional part of the magnitude of it expressed in fraction in base x with $x \in \mathbb{N}_{>1}$ being D_x can be expressed in finite digits in radix point in base $y \neq x$ with $y \in \mathbb{N}_{>1}$ iff the reduced denominator (denominator cancelled the greatest common factor with the numerator) of D_x has no prime factor that is not a prime factor of y.

III Binary

A binary digit is called a bit, which is either 0 or 1. Binary arithmetic is the same as decimals, except that "invert" or "flip" means converting 0 to 1 and 1 to 0, and "complement" means inverting all bits.

The most significant bit (MSB) or most significant digit is the bit with the highest value place in a binary number and is the leftmost bit in standard binary notation; the least significant bit (LSB) or least significant digit is the bit with the lowest value place in a binary number and is the rightmost bit in standard binary notation.

4 bits is called a nibble; 8 bits is called a byte.

A binary prefix is a unit prefix that indicates a multiple of a unit of measurement by an integer power of two. They are most often used in information technology as multipliers of bit and byte, in which the short prefixes are prefixed before b (representing bits) or B (representing bytes), and the long prefixes are prefixed before bits or bytes.

Value	IEC (short)	IEC (long)	JEDEC (short)	JEDEC (long)
1024	Ki	kibi	K	kilo
1024^2	Mi	mebi	M	mega
1024^3	Gi	gibi	G	giga
1024^4	Ti	tebi	T	tera
1024^5	Pi	pebi	—	
1024^6	Ei	exbi	—	
1024^7	Zi	zebi	—	
1024^8	Yi	yobi	—	
1024^9	Ri	robi	—	
1024^10	Qi	quebi	—	

2 Signed Number

Suppose you have N bit, to present a signed number, there's three common method:

I Sign and magnitude

The MSB represents sign, in which 0 represents positive and 1 represents negative; the remaining represents magnitude.

It can represent numbers range from $-(2^{N-1}-1)$ to $2^{N-1}-1$. It has two zero ($+0$ and -0).

It is the easiest to read, but its arithmetic is the hardest to compute.

II 1's complement

The nonnegative numbers are the same as those in sign and magnitude; the negative numbers invert all bits of its absolute value, i.e. $(2^N - 1)$ minus it.

It can represent numbers range from $-(2^{N-1} - 1)$ to $2^{N-1} - 1$. It has two zero ($+0$ and -0).

Its arithmetic is easier than sign and magnitude and harder than 2's complement to compute.

The addition can be computed the same as unsigned numbers; however, if the sum had a carry out from the MSB, we must add it back to the LSB, called the "end-around carry".

Proof.
Assume no overflow conditions.

An n-bit negative number X is represent as $2^n - 1 - |X|$ in 1's complement.

Suppose we want to add two n-bit numbers A and B, and in 1's complement, A_1 and B_1. Without loss of generality, assume $A \geq B$.

Let:

$$S = A_1 + B_1, \quad R = S \mod 2^n, \quad C = \left\lfloor \frac{S}{2^n} \right\rfloor,$$

and S_c be the correct sum of A and B in 1's complement.

Case 0: $A, B \geq 0$

$$S = A_1 + B_1 = A + B = S_c$$

Case 1: $A \geq 0 > B \wedge |B| \leq A$

$$S = A_1 + B_1 = 2^n - 1 + A - |B|$$

$$R = A - |B| - 1, \quad C = 1$$

$$S_c = A - |B| = R + C$$

Case 2: $A \geq 0 > B \wedge |B| \geq A$

$$S = A_1 + B_1 = 2^n - 1 + A - |B| = S_c = R$$

Case 3: $0 \geq A > B$

$$S = A_1 + B_1 = 2^{n+1} - 2 - |A| - |B|$$

$$R = 2^n - 2 - |A| - |B|, \quad C = 1$$

$$S_c = 2^n - 1 - |A| - |B| = R + C$$

□

If the sum of two positive numbers is greater than $2^{N-1} - 1$, it overflows into the negative range; if the sum of two negative numbers is less than $-2^{N-1} + 1$, it overflows into the positive range. This makes overflow detection straightforward.

III 2's complement

The nonnegative numbers are the same as those in sign and magnitude; the negative numbers invert all bits of its absolute value and add 1, i.e. 2^N minus it.

It can represent numbers range from -2^{N-1} to $2^{N-1}-1$. It has only one zero.

The arithmetic is the easiest to compute.

The addition can be computed the same as unsigned numbers. If the sum of two positive numbers exceeds $2^{N-1}-1$, it overflows into the negative range; if the sum of two negative numbers is less than -2^{N-1}, it overflows into the positive range. This makes overflow detection straightforward.

It is the most commonly used representation for signed numbers in computers.

3 Binary Codes

I Binary-coded decimal (BCD)

Binary-coded decimal (BCD), or more explicitly 8-4-2-1 BCD, is a way to represent decimal digits (0 to 9) in binary, in which each decimal digit is encoded separately using 4 bits (a nibble). There are several other possible sets of binary codes for the ten decimal digits.

A k-bit ($k \geq 4$) weighted code has the property that if the weights are integers $w_{k-1}, w_{k-2}, \dots w_0$ (in decimal, and where for all $0 \leq i < j < k$, $w_i < w_j$), the code $a_{k-1}a_{k-2}\dots a_0$, where $a_i \in \{0,1\}$, represents number N, then

$$N = \sum_{n=0}^{k-1} w_n a_n.$$

If a number N has more than one possible codes in a specific set of weights, the one such that the unsigned binary number presented by the code is least is usually used. A such k-bit weighted code can represent up to $\sum_{n=1}^{k-1} w_n$.

Below shows the BCD and other commonly used binary codes:

Decimal Digit	8-4-2-1 Code (BCD)	6-3-1-1 Code	Excess-3 Code	2-out-of-5 Code	Gray Code
0	0000	0000	0011	00011	0000
1	0001	0001	0100	00101	0001
2	0010	0011	0101	00110	0011
3	0011	0100	0110	01001	0010
4	0100	0101	0111	01010	0110
5	0101	0111	1000	01100	1110
6	0110	1000	1001	10001	1010
7	0111	1001	1010	10010	1011
8	1000	1011	1011	10100	1001
9	1001	1100	1100	11000	1000

- **8-4-2-1 Code (BCD)**: Each decimal digit is directly mapped to its binary form, which makes it easy to convert between decimal numbers and binary.
- **6-3-1-1 Code**: Sometimes makes it easier to design display drivers or simplify logic circuits.
- **Excess-3 Code**: It's sefl-complementing, i.e. the complement of it is the 9's complement of the number.
- **2-out-of-5 Code**: Always has exactly 2 ones.
- **Gray Code**: Each step changes only one bit, which minimizes transition errors for sensors, counters, or rotary encoders.

II Character Encoding

Many applications of computers require the processing of data which contains numbers, letters, and other symbols. In order to transmit such data to or from a computer or store it internally in a computer, each character must be represented by a binary code.

Common character encodings include:

- **ASCII code (American Standard Code for Information Interchange)**: A 7-bit code, so 128 different code combinations are available.
- **Unicode or the Unicode Standard (TUS)**: Includes international characters, symbols, etc. such as those in Arabic, CJK Unified Ideographs, and Emoticons. The Unicode Standard itself defines three encodings: UTF-8, UTF-16, and UTF-32. UTF-8 is the most widely used, in part due to its backwards-compatibility with ASCII.

4 Boolean Algebra

I Introduction

The basic mathematics needed for the study of logic design of digital systems is Boolean algebra. Boolean algebra was introduced by George Boole in his first book The Mathematical Analysis of Logic (1847), and set forth more fully in his An Investigation of the Laws of Thought (1854).

Switching devices are essentially two-state devices (e.g. switches which are open or closed and transistors with high or low output voltages). Consequently, we will emphasize the special case of Boolean algebra in which all of the variables assume only one of two values, 0 (false) or 1 (true), called Boolean variables; this two-valued Boolean algebra is also called switching algebra.

In a switch circuit, 0 (usually) represents an open switch, and 1 represents a closed circuit. In general, 0 and 1 can be used to represent the two states in any binary-valued system.

A Boolean function is a logical operation performed on one or more binary inputs that produces a single binary output.

A logic gate is a device that performs a Boolean function. Below, electronic symbols follow ANSI standards.

II Logical Operators

i NOT / Negation / Inversion / Complement

- Symbol: A', $\neg A$, $\sim A$, or $\overline{A}$
- Definition: Invert the value.
- Truth Table:

A	A'
0	1
1	0

- Logic gate: _____

ii AND / Conjunction / Logical Multiplication

- Symbol: $A \cdot B$, $A * B$, or AB
- Definition: True if both inputs are true and false otherwise.
- Truth Table:

A	B	$A \cdot B$
0	0	0
1	0	0
0	1	0
1	1	1

- Logic gate: _____

iii OR / Disjunction / Logical Addition / Inclusive OR

- Symbol: $A + B$
- Definition: True if at least one input is true and false otherwise.
- Truth Table:

A	B	$A + B$
0	0	0
1	0	1
0	1	1
1	1	1

- Logic gate: _____

iv XOR / Exclusive OR

- Symbol: $A \oplus B$
- Definition: $A'B + AB'$
- Truth Table:

A	B	$A \oplus B$
0	0	0
1	0	1
0	1	1
1	1	0

- Logic gate: _____

v NAND / Not AND

- Definition: $(AB)'$
- Truth Table:

A	B	$(A \cdot B)'$
0	0	1
1	0	1
0	1	1
1	1	0

- Logic gate: _____

vi NOR / Not OR

- Definition: $(A + B)'$
- Truth Table:

A	B	$(A + B)'$
0	0	1
1	0	0
0	1	0
1	1	0

- Logic gate: _____

vii XNOR / Logical equivalence / Exclusive NOR

- Symbol: $A \equiv B$ or $A \odot B$
- Definition: $(A \oplus B)'$

- Truth Table:

A	B	$(A \oplus B)'$
0	0	1
1	0	0
0	1	0
1	1	1

- Logic gate:

III Boolean Arithmetic

i Boolean expression

A Boolean expression is formed by application of the logic operations to one or more Boolean variables or constants.

- **Literals**: The simplest expressions, consisting of a single constant or variable or its complement, such as 0, X, or Y' .
- **Product terms**: A product term is a product of literals or a literal.
- **Sum terms**: A sum term is a sum of literals or a literal.

In a Boolean expression, parentheses are added as needed to specify the order in which the operations are performed; without parentheses, complementation is performed first, and then AND, and then OR; and the precedence of XOR and XNOR are either right higher to OR or right lower to OR depending on the context.

ii Boolean function

A Boolean function or a switching function is a function of which the domain is $\{1,0\}^n$ in which $n \in \mathbb{N}$ and the codomain is $\{1,0\}$.

The ON-set of a Boolean function f is the set of all input combinations such that f of them is 1. Each element in the ON-set of f corresponds to a row in the truth table of f in which the output is 1. The OFF-set of a Boolean function f is the set of all input combinations such that f of them is 0. Each element in the OFF-set of f corresponds to a row in the truth table of f in which the output is 0.

iii Truth table

A truth table, also called a table of combinations, specifies the corresponding output values for all possible combinations of input values for a Boolean function in the order such that if two input combinations have the same inputs in the first i variables and A_{i+1} of the first one is 0, A_{i+1} of the second one is 1, then the first is put prior to the second. A truth table for an n-variable Boolean function $f\left(A_1, A_2, \ldots A_n\right)$ have 2^n rows and is:

A_1	A_2	$\ldots$	A_n	f

0	0	…	0	$f(0,0,\ldots 0)$
0	0	…	1	$f(0,0,\ldots 1)$
⋮	⋮	⋮	⋱	⋮
1	1	…	1	$f(1,1,\ldots 1)$

iv Sum-of-products (SOP) form

An expression is said to be in SOP form if it consists of a sum (OR) of product (AND) or single variable terms, in which it is called to be degenerate when some of the terms are single variables.

Every Boolean expression can be expressed in SOP form. And a Boolean expression may have more than one SOP forms.

v Product-of-sums (POS) form

An expression is said to be in POS form if it consists of a product (AND) of sum (OR) or single variable terms, in which it is called to be degenerate when some of the terms are single variables.

Every Boolean expression can be expressed in POS form. And a Boolean expression may have more than one POS forms.

vi Dual

The dual of an Boolean expression or equation is another Boolean expression or equation obtained by simplifing the original expression or equation such that it only consists of literals, AND, and OR, and then replacing all AND in it with OR, all OR in it with AND, and all constants in it with their complements.

vii Boolean Vector Functions or Boolean Multi-output Functions

A Boolean vector function or a Boolean multi-output function is a function $f : \{1,0\}^n \to \{1,0\}^m$. A Boolean vector function can be specified with truth table or as a tuple of m Boolean functions.

IV Theorems

i Idempotent Laws

$$X + X = X, \quad X \cdot X = X.$$

ii Involution Law

$$(X')' = X.$$

iii Laws of Complementarity

$$X + X' = 1, \quad X \cdot X' = 0.$$

iv Commutativity

AND, OR, XOR, NAND, NOR, and XNOR are commutative.

v Associativity

AND, OR, XOR, NAND, NOR, and XNOR are associative.

vi Distributivity

- AND is distributive over OR and XOR.
- OR is distributive over AND and XOR.
- XOR is distributive over AND and OR.

vii DeMorgan's Laws

$$(\sum_{i=1}^{n} X_i)' = \prod_{i=1}^{n} X_i', \quad (\prod_{i=1}^{n} X_i)' = \sum_{i=1}^{n} X_i',$$
$$(\bigoplus_{i=1}^{n} X_i)' = \bigodot_{i=1}^{n} X_i', \quad (\bigodot_{i=1}^{n} X_i)' = \bigoplus_{i=1}^{n} X_i'.$$

viii Duality Principle

A Boolean equation is an identity if and only if the dual of it is an identity.

ix Uniting Theorems

$$XY + XY' = X, \quad (X + Y)(X + Y') = X.$$

x Absorption Theorems

$$X + XY = X, \quad X(X + Y) = X.$$

xi Elimination Theorems

$$X + X'Y = X + Y, \quad X(X' + Y) = XY.$$

xii Consensus Theorems

The consensus theorems involve eliminate one term from an expression in SOP or POS form, in which the eliminated term is called the consensus term.

$$XY + X'Z + YZ = XY + X'Z.$$

Proof.

$$\begin{aligned} XY + X'Z + YZ &= XY + X'Z + (X + X')YZ \\ &= XY + X'Z + XYZ + X'YZ \\ &= XY + X'Z. \end{aligned}$$

□

$$(X+Y)(X'+Z)(Y+Z) = (X+Y)(X'+Z).$$

Proof.

$$\begin{aligned}(X+Y)(X'+Z)(Y+Z) &= (X+Y)(X'+Z)(X+X')(Y+Z)\\ &= (X+Y)(X'+Z)(X+Y+Z)(X'+Y+Z)\\ &= (X+Y)(X'+Z).\end{aligned}$$

□

xiii Axiom of Equality

For an one-to-one Boolean function f, a Boolean expression A equals another Boolean expression B if and only if $f(A)$ equals $f(B)$.

xiv Boole's expansion theorem, Shannon's expansion theorem, Shannon decomposition, or fundamental theorem of Boolean algebra

The theorem states that

$$F = x \cdot F_x + x' \cdot F_{x'},$$

where F is any Boolean function, x is a variable, and F_x and $F_{x'}$, sometimes called the positive and negative Shannon cofactors, respectively, of F with respect to x, are F with the independent variable x set to 1 and to 0 respectively.

xv Combination of Distributivity and Consensus Theorem

$$(X+Y)(X'+Z) = XZ + X'Y$$

Proof.

$$(X+Y)(X'+Z) = 0 + XZ + X'Y + YZ = XZ + X'Y$$

□

xvi XOR and XNOR of Complements

$$X' \oplus Y' = X \oplus Y, \quad X' \odot Y' = X \odot Y.$$

xvii XOR and XNOR Series Theorems

$$\left(\bigoplus_{i=1}^{n} X_i\right)' = \left(\bigoplus_{i=1}^{j-1} X_i\right) \oplus (X_j)' \oplus \left(\bigoplus_{i=j+1}^{n} X_i\right), \quad \forall j \le n \wedge j \in \mathbb{N}, \forall n \text{ s.t. } \frac{n}{2} \in \mathbb{N}.$$

Proof.

Case $n = 2$:

$$\begin{aligned}(X_1 \oplus X_2)' &= (X_1X_2' + X_1'X_2)' = (X_1X_2')'(X_1'X_2)' \\ &= (X_1' + X_2)(X_1 + X_2') = (X_1'X_2' + X_1X_2) \\ &= X_1' \oplus X_2 = X_1 \oplus X_2'\end{aligned}$$

Prove by mathematical induction. Assume it holds for $n = k$ and $n = 2$. We want to prove that it holds for $n = k + 2$.

$$\begin{aligned}\left(\bigoplus_{i=1}^{k+2}\right)' &= \left(\bigoplus_{i=1}^{k} X_i\right)' \oplus X_{k+1} \oplus X_{k+2} \\ &= \left(\bigoplus_{i=1}^{j-1} X_i\right) \oplus \left(X_j\right)' \oplus \left(\bigoplus_{i=j+1}^{k} X_i\right) \oplus X_{k+1} \oplus X_{k+2}, \quad \forall j \le k \wedge j \in \mathbb{N}\end{aligned}$$

$$\begin{aligned}\left(\bigoplus_{i=1}^{k+2}\right)' &= X_1 \oplus X_2 \oplus \left(\bigoplus_{i=3}^{k+2} X_i\right)' \\ &= X_1 \oplus X_2 \oplus \left(\bigoplus_{i=3}^{j-1} X_i\right) \oplus \left(X_j\right)' \oplus \left(\bigoplus_{i=j+1}^{k+2} X_i\right), \quad \forall 3 \le j \le k+2 \wedge j \in \mathbb{N}\end{aligned}$$

□

$$\bigoplus_{i=1}^{n} X_i = \left(\bigodot_{i=1}^{n} X_i\right)', \quad \forall n \text{ s.t. } \frac{n}{2} \in \mathbb{N}.$$

Proof.

By the definitions, it holds for case $n = 2$.

Prove by mathematical induction. Assume it holds for $n = k$ and $n = 2$. We want to prove that it holds for $n = k + 2$.

$$\begin{aligned}\bigoplus_{i=1}^{k+2} X_i &= \left(\bigoplus_{i=1}^{k} X_i\right) \oplus X_{k+1} \oplus X_{k+2} \\ &= \left(\bigodot_{i=1}^{k} X_i\right)' \oplus X_{k+1} \oplus X_{k+2} \\ &= \left(\bigodot_{i=1}^{k+1} X_i\right) \oplus X_{k+2} \\ &= \left(\bigodot_{i=1}^{k+2} X_i\right)'\end{aligned}$$

□

$$\bigoplus_{i=1}^{n} X_i = \bigodot_{i=1}^{n} X_i, \quad \forall n \text{ s.t. } \frac{n-1}{2} \in \mathbb{N}.$$

Proof.
By

$$\bigoplus_{i=1}^{n} X_i = \left(\bigodot_{i=1}^{n} X_i \right)', \quad \forall n \text{ s.t. } \frac{n}{2} \in \mathbb{N}.$$

For n such that $\frac{n-1}{2} \in \mathbb{N}$,

$$\begin{aligned}
\bigoplus_{i=1}^{n} X_i &= \bigoplus_{i=1}^{n-1} X_i \oplus X_n \\
&= \bigoplus_{i=1}^{n-1} X_i \oplus X_n \\
&= \left(\bigodot_{i=1}^{n-1} X_i \right)' \oplus X_n \\
&= \bigodot_{i=1}^{n} X_i
\end{aligned}$$

□

V Minterm and Maxterm Expansions, Canonical Expansions, or Standard Expansion

i Minterm expansion, canonical SOP, or standard SOP

A minterm of a completely specified Boolean function $f : \{1,0\}^n \to \{1,0\}$, denoted as m_i for the input combination in the $(i+1)$th row of the truth table of f that is in the ON-set of f, is defined for any input combinations in the ON-set of f as

$$m_i = \prod_{k=1}^{n} y_k,$$

in which y_k is defined as the kth input variable, i.e. x_k, if the kth input in that input combination is 1 and as the complement of the kth input variable, i.e. $(x_k)'$, if the kth input in that input combination is 0.

Minterm expansion, canonical SOP, or standard SOP is an expression of a completely specified Boolean function as a sum of minterms of it. Let S be the set of all integer i such that the input combination in the $(i+1)$th row of the truth table of f is in the ON-set of f. Then the minterm expansion, canonical SOP, or standard SOP of f is

$$f = \sum_{i \in S} m_i,$$

also denoted as

$$f = \sum m(i \in S).$$

For a given completely specified Boolean function, there exists a unique minterm expansion of it.

For example, given a function $f(a,b,c)$ with truth table

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

, the minterm expansion of it is

$$f = m_2 + m_5 + m_6 = \sum m(2, 5, 6).$$

ii Maxterm expansion, canonical POS, or standard POS

A maxterm of a completely specified Boolean function $f : \{1, 0\}^n \to \{1, 0\}$, denoted as M_i for the input combination in the $(i + 1)$th row of the truth table of f that is in the OFF-set of f, is defined for any input combinations in the OFF-set of f as

$$M_i = \sum_{k=1}^{n} y_k,$$

in which y_k is defined as the kth input variable, i.e. x_k, if the kth input in that input combination is 0 and as the complement of the kth input variable, i.e. $(x_k)'$, if the kth input in that input combination is 1.

Maxterm expansion, canonical POS, or standard POS is an expression of a completely specified Boolean function as a product of maxterms of it. Let T be the set of all integer i such that the input combination in the $(i + 1)$th row of the truth table of f is in the OFF-set of f. Then the maxterm expansion, canonical POS, or standard POS of f is

$$f = \prod_{i \in T} M_i,$$

also denoted as

$$f = \prod M(i \in T).$$

For a given completely specified Boolean function, there exists a unique maxterm expansion of it.

For example, given a function $f(a, b, c)$ with truth table

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0

1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

, the maxterm expansion of it is

$$f = M_0 M_1 M_3 M_4 M_7 = \prod M(0, 1, 3, 4, 7).$$

iii Conversion between function and its complement

Let f be a completely specified Boolean function with minterm expansion

$$f = \sum m(i \in S),$$

and maxterm expansion

$$f = \prod M(i \in T),$$

where S be the set of all integer i such that the input combination in the $(i+1)$th row of the truth table of f is in the ON-set of f, T be the set of all integer i such that the input combination in the $(i+1)$th row of the truth table of f is in the OFF-set of f.

Then, the complement f' of has minterm expansion

$$f' = \sum m(i \in T),$$

and maxterm expansion

$$f' = \prod M(i \in S).$$

VI Minimum SOP form

i Implicant

Given a Boolean function f of n variables, a product term P is an implicant of f iff for every combination of values of the n variables for which $P = 1$, f is also equal to 1.

ii Prime implicant

A prime implicant of a function f is an implicant of f which is no longer an implicant of f if any literal is deleted from it.

iii Essential prime implicant

An essential prime implicant of a function f is a prime implicant P of f such that there exists a minterm m of f such that m implies P and for any other prime implicant Q of f, m implies Q'.

iv Minimum SOP form

The SOP form of a Boolean function that has the fewest number of terms out of all SOP forms of the function. For a given Boolean function, there may exist more than one minimum SOP forms of it.

A minimum SOP form of a Boolean function must consist of some of its prime implicants (but not necessarily all), that is, if a SOP form contains implicants that are not prime implicants, it is not minimal.

A minimum SOP form of a Boolean function must contain all of its essential prime implicants.

v Minimum POS form

The POS form of a Boolean function that has the fewest number of terms out of all POS forms of the function. For a given Boolean function, there may exist more than one minimum POS forms of it.

vi Incompletely Specified Boolean Functions (ISF) or Don't-care Functions

An incompletely specified function (ISF) or a don't-care function is a Boolean function but in which for some input combinations, the output is not defined or irrelevant. Those input combinations are called don't-care conditions, sometimes denoted as d, and their outputs are sometimes written as X, denoting don't care. The definition of ON-set and OFF-set of an ISF is the same as completely specified function; the Don't-Care set or DC-set of an ISF is the set of all don't-care conditions.

The output for the don't-care conditions can be assigned either 0 or 1 such that the minimum SOP (or POS) forms have the fewest terms.

An ISF f with ON-set S, OFF-set T, and DC-set D is sometimes written similar to minterm expansion as

$$f = \sum m(i \in S) + \sum d(i \in D),$$

and similar to maxterm expansion as

$$f = \prod M(i \in T) + \prod d(i \in D).$$

VII Karnaugh Maps

A Karnaugh Map, aka a K-map, is a grid that visualize the ways to simplify a completely or incompletely specified Boolean function to a minimum SOP form.

i Drawing Karnaugh Maps of less than five variables

A K-map for a 2-variable Boolean function $f(A, B)$ is

B \ A	0	1
0	$f(0,0)$	$f(1,0)$
1	$f(0,1)$	$f(1,1)$

A K-map for a 3-variable Boolean function $f(A, B, C)$ is

BC \ A	0	1
00	$f(0,0,0)$	$f(1,0,0)$
01	$f(0,0,1)$	$f(1,0,1)$
11	$f(0,1,1)$	$f(1,1,1)$
10	$f(0,1,0)$	$f(1,1,0)$

A K map for a 4-variable Boolean function $f(A, B, C, D)$ is

CD \ AB	00	01	11	10
00	$f(0,0,0,0)$	$f(0,1,0,0)$	$f(1,1,0,0)$	$f(1,0,0,0)$
01	$f(0,0,0,1)$	$f(0,1,0,1)$	$f(1,1,0,1)$	$f(1,0,0,1)$
11	$f(0,0,1,1)$	$f(0,1,1,1)$	$f(1,1,1,1)$	$f(1,0,1,1)$
10	$f(0,0,1,0)$	$f(0,1,1,0)$	$f(1,1,1,0)$	$f(1,0,1,0)$

If the ouput of the input combination of a cell is 1, we write 1 in that cell; if the ouput of the input combination of a cell is 0, we can leave that cell empty or write 0 in that cell; if the input combination of a cell is a don't care condition, we write X in that cell.

The order 00, 01, 11, 10 is the gray code order, which is such that adjacent cells in the K-map differ by only one variable.

The notation of a cell is the input combination of that cell (thus a binary number), called binary notation, or the decimal integer in BCD representated by the input combination of that cell, called decimal notation.

ii Groups of 1s

The first step of simplifying a Boolean function using K-maps is grouping 1s (minterms).

A group is a rectangle of $2^m \times 2^n$ cells that are either 1 (minterm) or X (don't-care condition), in which $m, n \in \mathbb{N}_0$, the leftmost and rightmost columns are horizontally adjacent, and the top and bottom rows are vertically adjacent. Cells can be in multiple groups.

Each group represents an implicant of f by the following rules:

1. Let the set of all input variables that are 1 in all cells in the group be O.
2. Let the set of all input variables that are 0 in all cells in the group be Z.
3. The implicant represented by that group is $\prod_{o \in O} o \prod_{z \in Z} z'$.

iii Allowed collections of groups

An allowed collection of groups must follow the following rules:

- Any 1 must be in at least one group.
- If any group g is completely covered by another group following the above rules, g must not be chosen, that is, a group chosen must represent a prime implicant.
- If any 1 is only covered by one possible group, then that group must be chosen, that is, any group representing an essential prime implicant must be chosen.

iv Minmum SOP form

Find an allowed collection of groups that contains the fewest groups, the sum of the implicants represented by them is a minimum SOP form of the given function.

v Workflow

1. Draw K-map.
2. Find all groups representing essential prime implicants.
3. Find an allowed collection of groups that contains the fewest groups and construct the minimum SOP form.

vi Five-variable Karnaugh Maps

One way of drawing a five-variable K-map is by placing one four-variable K-map on top of another, that is, for a 5-variable Boolean function $f(A, B, C, D, E)$, the K-map is

A	BC \ DE	00	01	11	10
0	00	$f(0,0,0,0,0)$ / $f(1,0,0,0,0)$	$f(0,0,1,0,0)$ / $f(1,0,1,0,0)$	$f(0,1,1,0,0)$ / $f(1,1,1,0,0)$	$f(0,1,0,0,0)$ / $f(1,1,0,0,0)$
	01	$f(0,0,0,0,1)$ / $f(1,0,0,0,1)$	$f(0,0,1,0,1)$ / $f(1,0,1,0,1)$	$f(0,1,1,0,1)$ / $f(1,1,1,0,1)$	$f(0,1,0,0,1)$ / $f(1,1,0,0,1)$
	11	$f(0,0,0,1,1)$ / $f(1,0,0,1,1)$	$f(0,0,1,1,1)$ / $f(1,0,1,1,1)$	$f(0,1,1,1,1)$ / $f(1,1,1,1,1)$	$f(0,1,0,1,1)$ / $f(1,1,0,1,1)$
	10	$f(0,0,0,1,0)$ / $f(1,0,0,1,0)$	$f(0,0,1,1,0)$ / $f(1,0,1,1,0)$	$f(0,1,1,1,0)$ / $f(1,1,1,1,0)$	$f(0,1,0,1,0)$ / $f(1,1,0,1,0)$

in which the adjacency between cells in the same layer of four-variable K-map is the same as that in four-variable K-map; the upper and lower triangles in the same square are adjacent; and a group is a cuboid of $2^l \times 2^m \times 2^n$ cells that are either 1 (minterm) or X (don't-care condition), in which $l, m, n \in \mathbb{N}_0$.

vii Minimum POS form

To find a minimum POS form of a function f, we can use K-map to find a minimum SOP form of f', and change all literals to its inverse, all AND to OR, and all OR to AND.

5 Digital Systems and Switching Circuits

- **Digital system**: Deals with signals that have discrete values. Theoretically, for a given input, the output is exactly correct.
- **Analog system**: Deals with signals that vary continuously over time. The output might have an error depending on the accuracy of the components used.
- **System design**: The highest level of the design of digital systems, where you break the system into subsystems, specify what each subsystem do, and determine the interconnection and control of the subsystems.
- **Logic design**: The middle level of the design of digital systems, where you specify the logic operations inside each subsystem.
- **Circuit design**: The lowest level of the design of digital systems, where you specify the electronic components and their interconnection to form the system.
- **Switching circuit**: A switching circuit has one or more inputs and one or more outputs which take on discrete values. Many of a digital system's subsystems take the form of a switching circuit.
- **Combinational circuit**: A circuit whose output value depends only on the present input value.
- **Sequential circuit**: A circuit whose output value depends on the present input value and past input values.
- **Logic gate**: An electronic device that performs a basic Boolean operation on one or more binary inputs (0 or 1) to produce a single binary output (0 or 1).