Switching Circuit and Logic Design

沈威宇

November 13, 2025

Contents

Chapter 1 Switching Circuit and Logic Design

1 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.

II Logical Operators

i NOT / Negation / Inversion / Complement

- Symbol: A', $\neg A$, $\sim A$, or $\overline{A}$
- Definition: One input, one output. True if the input is false and false otherwise.

ii AND / Conjunction / Logical Multiplication

- Symbol: $A \cdot B$, $A * B$, or AB
- Definition: At least two inputs, one output. True if all inputs are true and false otherwise.

iii OR / Disjunction / Logical Addition / Inclusive OR

- Symbol: $A + B$
- Definition: At least two inputs, one output. True if at least one input is true and false otherwise.

iv NAND / Not AND

Definition: At least two inputs, one output. True if all inputs are false and false otherwise.

v NOR / Not OR

Definition: At least two inputs, one output. True if at least one input is false and false otherwise.

vi XOR / Exclusive OR

- Symbol: $A \oplus B$
- Definition: At least two inputs, one output. True if odd number of inputs are true and false otherwise.

vii XNOR / Logical equivalence / Exclusive NOR

- Symbol: $A \equiv B$ or $A \odot B$
- Definition: At least two inputs, one output. True if even number (including 0) of inputs are true and false otherwise.

viii IMPLY / Logical conditional

- Symbol: $A \to B$
- Definition: Two inputs, one output. $A \to B$ is defined as $A' + B$.

ix NIMPLY / Material nonimplication

- Symbol: $A \nrightarrow B$
- Definition: Two inputs, one output. $A \nrightarrow B$ is defined as AB'.

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 Incompletely specified function (ISF) or Don't-care function

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, often denoted as X. Those input combinations are called don't-care conditions.

iv 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 $f\left(A_1, A_2, \dots A_n\right)$ have 2^n rows and is:

A_1	A_2	$\dots$	A_n	f
0	0	$\dots$	0	$f\,(0,0,\dots 0)$
0	0	$\dots$	1	$f\,(0,0,\dots 1)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
1	1	$\dots$	1	$f\,(1,1,\dots 1)$

v 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.

vi 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.

vii 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.

viii Functionally complete

A functionally complete set is a set of logic operators and constant sources (0 or 1) that can express all possible Boolean functions of any number of input variables. A minimal functionally complete set is a functionally complete set such that removing any one element from it makes it no longer functionally complete. Some minimal functionally complete sets are $\{\text{AND}, \text{NOT}\}$, $\{\text{OR}, \text{NOT}\}$, $\{\text{NAND}\}$, $\{\text{NOR}\}$, $\{\text{IMPLY}, 0\}$, and $\{\text{NIMPLY}, 1\}$, in which NAND and NOR are called universal.

ix Linear Boolean function

A Boolean function is linear iff it can be expressed as the XOR of a subset of the union of its input variables and constant 1, iff

$$\exists c_0, c_1, \ldots, c_n \in \{1, 0\} : \ f(x_1, x_2, \ldots, x_n) = c_0 \oplus \bigoplus_{i=1}^{n} c_i x_i,$$

iff

$$\forall a, b \in D_f : \ f(a \oplus b) = f(a) \oplus f(b) \oplus f(0).$$

x 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

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

ii Involution Law

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

iii Laws of Complementarity

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

iv Commutativity of AND, OR, XOR, and XNOR

$$XY = YX, \quad X + Y = Y + X,$$

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

v Associativity of AND, OR, XOR, and XNOR

$$XYZ = X(YZ), \quad X + Y + Z = X + (Y + Z),$$

$$X \oplus Y \oplus Z = X \oplus (Y \oplus Z), \quad X \odot Y \odot Z = X \odot (Y \odot Z).$$

vi Distributivity of AND over OR and OR over AND

$$X(Y + Z) = XY + XZ.$$

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

vii XOR of ANDs Theorem or Distributivity of AND over XOR

$$XY \oplus XZ = X(Y \oplus Z).$$

Proof.

$$XY \oplus XZ = XY(XZ)' + (XY)'XZ = XYX' + XYZ' + X'XZ + Y'XZ = XYZ' + XY'Z = X(Y \oplus Z).$$

□

viii XOR of ORs Theorem

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

Proof.

$$(X+Y)\oplus(X+Z) = (X+Y)(X+Z)' + (X+Y)'(X+Z) = XX'Z' + YX'Z' + X'Y'X + X'Y'Z = X'YZ' + X'Y'Z =$$

□

ix XNOR of ORs Theorem or Distributivity of OR over XNOR

$$(X + Y) \odot (X + Z) = X + (Y \odot Z).$$

Proof.

$$(X+Y)\odot(X+Z) = (X+Y)(X+Z) + (X+Y)'(X+Z)' = X + YZ + X'Y'Z' = X + YZ + Y'Z' = X + (Y \odot Z).$$

□

x XNOR of ANDs Theorem

$$XY \odot XZ = X' + (Y \odot Z).$$

Proof.

$$XY \odot XZ = XYZ + (XY)'(XZ)' = XYZ + (X' + Y')(X' + Z') = XYZ + X' + Y'Z' = X' + (Y \odot Z).$$

□

xi DeMorgan's Laws

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

xii 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 $f(B)$.

xiii Duality Principle

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

xiv Uniting Theorems

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

xv Absorption Theorems

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

xvi Elimination Theorems

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

xvii 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.

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

□

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

The theorem states that

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

where F is any n-variable Boolean function, x is any independent variable of F, F_x and $F_{x'}$, sometimes called the positive and negative Shannon cofactors, respectively, of F with respect to x, are $(n-1)$-variable Boolean functions defined as F with the x set to 1 and to 0 respectively, and the RHS is called a Shannon's expansion or Boole's expansion of F.

xix Combination of Distributivity and Consensus Theorem

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

Proof.

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

□

xx XOR and XNOR Series Theorems

$$\bigoplus_{i=1}^{n} X_i = \left(\sum_{i=1}^{n} X_i\right) \mod 2.$$

$$\bigodot_{i=1}^{n} X_i = 1 - \left(\sum_{i=1}^{n} X_i\right) \mod 2 = \left(1 + \sum_{i=1}^{n} X_i\right) \mod 2.$$

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

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 $n = k + 2$.

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

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

□

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

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 $n = k + 2$.

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

□

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

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

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

□

xxi Consensus of XOR Theorem

$$X \oplus Y + X \oplus Z + Y \oplus Z = X \oplus Y + X \oplus Z = X \oplus Y + Y \oplus Z = X \oplus Z + Y \oplus Z = XY' + X'Z + YZ' = X'Y + XZ' + Y'Z$$

Proof.

$$X \oplus Y + X \oplus Z = XY' + X'Y + XZ' + X'Z = XY' + X'Y + XZ' + X'Z + YZ' + Y'Z = X \oplus Y + X \oplus Z + Y \oplus Z = XY' +$$

□

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 or its complement that is in the input combination, 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. That is, the i of any m_i is the binary integer representated by the input combination of that row converted to decimal, and m_i is defined as the product of the input variables or their complements that are in the input combination of that row for all rows with output 1 and undefined for rows with output 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 complement of the kth input variable or its complement that is in the input combination, 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. That is, the i of any M_i is the binary integer representated by the input combination of that row converted to decimal, and M_i is defined as the sum of the complement of the input variables or their complements that are in the input combination of that row for all rows with output 0 and undefined for rows with output 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 Incompletely specified boolean functions

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

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

where each d_i is called a don't-care terms.

iv NOT, AND, and OR of functions

For completely or incompletely specified Boolean function f:

$$f = \sum m(i \in S) + \sum d(i \in D) \Longleftrightarrow f' = \prod M(i \in S) \cdot \prod D(i \in D).$$

$$f = \prod M(i \in T) \cdot \prod D(i \in D) \Longleftrightarrow f' = \sum m(i \in T) + \sum d(i \in D).$$

For completely or incompletely specified Boolean functions f and g with same number of variables:

$$f = \sum m(i \in S_f) + \sum d(i \in D_f) \wedge g = \sum m(i \in S_g) + \sum d(i \in D_g) \Longrightarrow fg = \sum m(i \in S_f \cap S_g) + \sum d\left(i \in \left(S_f \cup \right.\right.$$

$$f = \sum m(i \in S_f) + \sum d(i \in D_f) \wedge g = \sum m(i \in S_g) + \sum d(i \in D_g) \Longrightarrow f+g = \sum m(i \in S_f \cup S_g) + \sum d\left(i \in \left(S_f \cup \right.\right.$$

$$f = \prod M(i \in T_f) \cdot \prod D(i \in D_f) \wedge g = \prod M(i \in T_f) \cdot \prod D(i \in D_f) \Longrightarrow fg = \prod M(i \in T_f \cup T_g) \cdot \prod D\left(i \in \left(T_f \cup \right.\right.$$

$$f = \prod M(i \in T_f) \cdot \prod D(i \in D_f) \wedge g = \prod M(i \in T_f) \cdot \prod D(i \in D_f) \Longrightarrow f+g = \prod M(i \in T_f \cap T_g) \cdot \prod D\left(i \in \left(T_f \right.\right.$$

VI Minimum form

i Implicant or 1-term

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

ii Implicate or 0-term

Given a Boolean function f of n variables, a sum term P is an implicate (aka 0-term) of f iff for every combination of values of the n variables for which $P = 0$, f is also equal to 0.

iii Prime implicant (PI)

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.

iv Prime implicate

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

v Essential prime implicant (EPI)

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 doesn't implies Q.

vi Essential prime implicate

An essential prime implicate of a function f is a prime implicate P of f such that there exists a maxterm M of f such that P implies M and for any other prime implicate Q of f, Q doesn't implies M.

vii Minimum/minimal SOP form or minimum sum (of prime implicants)

A SOP form of a Boolean function is called minimum if it has the fewest number of terms out of all SOP forms of the function, and every product term in it can not have any variable in it be eliminated. 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). If a SOP form contains implicants that are not prime implicants, it is not a minimum SOP form.

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

viii Minimum/minimal POS form or minimum product (of prime implicates)

A POS form of a Boolean function is called minimum if it has the fewest number of terms out of all POS forms of the function, and every sum term in it can not have any variable in it be eliminated. For a given Boolean function, there may exist more than one minimum POS forms of it.

A minimum POS form of a Boolean function must consist of some of its prime implicates (but not necessarily all). If a POS form contains implicates that are not prime implicates, it is not a minimum POS form.

A minimum POS form of a Boolean function must contain all of its essential prime implicates.

One can simplify a Boolean function to minimum POS form by:

1. taking dual of the expression,
2. simplifying it to minimum SOP form, and
3. taking dual of it.

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 Draw Karnaugh Maps of less than five variables

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 binary integer representated by the input combination of that cell converted to decimal, called decimal notation.

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	$f(0,0,1,0)$	$f(0,1,1,0)$	$f(1,1,1,0)$	$f(1,0,1,0)$

ii Group 1's

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

A group (or loop) 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 Select collections of groups

An allowed collection of groups must follow the following rules:

- Any 1 must be in at least one group.
- 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.
- If any group g is completely covered by another group, g must not be chosen, that is, a group chosen must represent a prime implicant.

Find the allowed collections of groups that contain the fewest groups. For each one of them, the sum of the implicants corresponding to the groups in it is a minimum SOP form of the given function.

iv Other uses of Karnaugh Maps

K-maps have one-to-one correspondence to truth tables. We can perform logic operations on functions by perform the operation on each cell on their K-maps to obtain the K-map of the result function.

v Veitch Diagrams

Veitch Diagrams are an alternative form of K-maps by labeling each input variable with a curly bracket containing the maximum rows or columns which that variable is 1 in the input combinations of all cells in them.

vi Five-variable Karnaugh Maps

One form 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 $l, m, n \in \mathbb{N}_0$.

Another form is drawing the upper and lower triangles of the previous form as two four-variable K-maps side-by-side and drawing a line to connect two parts of a group on different map.

Yet another form is the same as the previous one but with Veitch diagram labeling and drawing two halves of boundary lines of a group surrounding two parts of a group on different maps instead of a line.

vii Finding minimum POS form

We can use K-maps to find minimum POS form of a completely or incompletely specified Boolean function by grouping 0s instead of 1s (0s and Xs are allowed in a group), in which each group represents an implicate 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 $\sum_{o \in O} o' + \sum_{z \in Z} z$.

, and the product of the sum terms corresponding to the groups in an allowed (covering all 0's) collections that contain the fewest groups is a minimum POS form of the Boolean function.

VIII Map-entered variable (MEV) or Variable entrant map (VEM)

Map-entered variable (MEV), aka variable entrant map (VEM), is an extension of Karnaugh maps used to simplify certain kinds of completely or incompletely specified Boolean functions with typically more than four variables by allowing some variables to be entered as symbols or expressions inside the K-map cells rather than expanding the map's dimensions.

i One MEV

Suppose we have an n-variable (usually five-variable) completely or incompletely specified Boolean function f with E being one of the input variable of f being the MEV such that

$$\{f(P,1), f(P,0)\} \notin \{\{1,X\},\{0,X\}\},$$

for any $(n-1)$-variable input combination P, where $f(P,x)$ with $x \in \{1,0\}$ denotes the output of f when the variables except MEVs be their corresponding values in P and E_i be x, that is, the function can be written as

$$f = \sum m(i \mid m_i \in A_1) + E \sum m(i \mid m_i \in A_E) + E' \sum m(i \mid m_i \in A_e) + \sum d(i \mid m_i \in A_X),$$

where m_i is a possible $(n-1)$-variable minterm. Let A be the set of all $(n-1)$-variable input combinations, and

$$A_0 = A \setminus (A_1 \cap A_E \cap A_e \cap A_X).$$

1. Draw a $(n-1)$-variable K-map for input variables except E. In each cell, let the $n-1$-variable input combination corresponding to the cell be P, where $f(P,E)$ denotes a function of E where the variables except E be their values in P:
   - If $f(P,1) = f(P,0) = 1$, that is, the cell corresponds to a minterm in A_1, enter 1.
   - If $f(P,1) = f(P,0) = 0$, that is, the cell corresponds to a minterm in A_0, leave the cell empty or enter 0.
   - If $f(P,1) = 1$ and $f(P,0) = 0$, that is, the cell corresponds to a minterm in A_E, enter E.
   - If $f(P,1) = 0$ and $f(P,0) = 1$, that is, the cell corresponds to a minterm in A_e, enter E'.

- If $f(P,1) = f(P,0) = X$, that is, the cell corresponds to a minterm in A_X, enter X.

2. Group 1s in same way as K-map (1s and Xs are allowed in a group). Each group corresponding to a $(n-1)$-variable implicant.
3. Group Es in same way as K-map but treating 1s as Xs (Es, 1s, and Xs are allowed in a group). Each group corresponding to a n-variable implicant with E be one of the literals.
4. Group E's in same way as K-map but treating 1s as Xs (E's, 1s, and Xs are allowed in a group). Each group corresponding to a n-variable implicant with E' be one of the literals.
5. An allowed collection of groups must follow the following rules:
   - Any 1, E, or E' must be in at least one group.
   - If any 1, E, or E' 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.
   - If any group g is completely covered by another group, g must not be chosen, that is, a group chosen must represent a prime implicant.
6. Find the allowed collections of groups that contain the fewest groups. For each one of them, the sum of the implicants corresponding to the groups in it is a minimum SOP form of the given function.

ii Multiple MEVs

Suppose we have an n-variable completely or incompletely specified Boolean function f in which there exist m (usually $n-4$) variables $E_1, E_2, \dots E_m$ chosen as MEVs such that

$$\{f(P,Q,1), f(P,Q,0)\} \notin \{\{1,X\},\{0,X\}\},$$

for any $(n-1)$-variable input combination P, for any fixed $(m-1)$-variable input combination Q, where $f(P,Q,x)$ with $x \in \{1,0\}$ denotes the output of f when the variables except MEVs be their corresponding values in P, the MEVs except E_i be their corresponding values in Q, and E_i be x, that is, the function can be written as

$$f = \sum m(i \mid m_i \in A_1) + \sum_{i=1}^{m} E_i \sum m(i \mid m_i \in A_{E_i}) + \sum_{i=1}^{m} E_i' \sum m(i \mid m_i \in A_{e_i}) + \sum d(i \mid m_i \in A_X),$$

where m_i is minterms in $(n-m)$-variable truth tables. Let A be the set of all $(n-m)$-variable input combinations, and

$$A_0 = A \setminus \left(A_1 \cap A_X \cap \bigcap_{i=1}^{m} \left(A_{E_i} \cap A_{e_i} \right) \right).$$

1. Draw a $(n-m)$-variable K-map for input variables except E and F. In each cell, let the $(n-m)$-variable input combination corresponding to the cell be P, for any $i \in \mathbb{N}_{\leq m}$:
   - If $f(P,R) = 1$ for any fixed m-variable input combination R of the MEVs, where $f(P,R)$ denotes the output of f when the variables except MEVs be their corresponding values in P and the MEVs be their corresponding values in R, that is, the cell corresponds to a minterm in A_1, enter 1.

- If $f(P, R) = 0$ for any fixed m-variable input combination R of the MEVs, where $f(P, R)$ denotes the output of f when the variables except MEVs be their corresponding values in P and the MEVs be their corresponding values in R, that is, the cell corresponds to a minterm in A_0, enter 0.
- If the set of all possible values of $f(P, Q, 1)$ is $\{1\}$ and the set of all possible values of $f(P, Q, 0)$ is $\{0\}$, for any fixed $(m-1)$-variable input combination Q of the MEVs except E_i, that is, the cell corresponds to a minterm in A_{E_i}, enter E_i.
- If the set of all possible values of $f(P, Q, 1)$ is $\{0\}$ and the set of all possible values of $f(P, Q, 0)$ is $\{1\}$, for any fixed $(m-1)$-variable input combination Q of the MEVs except E_i, that is, the cell corresponds to a minterm in A_{e_i}, enter E_i'.
- If $f(P, R) = X$ for any fixed m-variable input combination R of the MEVs, where $f(P, R)$ denotes the output of f when the variables except MEVs be their corresponding values in P and the MEVs be their corresponding values in R, that is, the cell corresponds to a minterm in A_X, enter X.

2. Group 1s in same way as K-map (1s and Xs are allowed in a group). Each group corresponding to a $(n-1)$-variable implicant.
3. For each $i \in \mathbb{N}_{\leq m}$, group E_is in same way as K-map but treating 1s as Xs (E_is, 1s, and Xs are allowed in a group). Each group corresponding to a $(n-m+1)$-variable implicant with E_i be one of the literals.
4. For each $i \in \mathbb{N}_{\leq m}$, group E_i's in same way as K-map but treating 1s as Xs (E_i's, 1s, and Xs are allowed in a group). Each group corresponding to a $(n-m+1)$-variable implicant with E_i' be one of the literals.
5. An allowed collection of groups must follow the following rules:
   - Any 1, E_i, or E_i' must be in at least one group.
   - If any 1, E_i, or E_i' 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.
   - If any group g is completely covered by another group, g must not be chosen, that is, a group chosen must represent a prime implicant.
6. Find the allowed collections of groups that contain the fewest groups. For each one of them, the sum of the implicants corresponding to the groups in it is a minimum SOP form of the given function.

IX Quine-McCluskey Method

The Quine-McCluskey method (along with the Petrick's method) is a systematic procedure to find the minimum SOP forms of a completely or incompletely specified Boolean function. Below, we first discuss about completely specified Boolean function.

i Minterm expansion

First, we convert the function minterm expansion.

Below, we will take

$$f(a,b,c,d) = \sum m(0,1,2,5,6,7,8,9,10,14)$$

for example.

ii Group minterms

Group minterms with the same number of 1's in the input combination together and list them. Let the group of minterms with k 1's be Group k. Group k and Group $(k+1)$ are called adjacent.

For f:

Group 0	0	0	0	0	0
Group 1	1	0	0	0	1
	2	0	0	1	0
	8	1	0	0	0
Group 2	5	0	1	0	1
	6	0	1	1	0
	9	1	0	0	1
	10	1	0	1	0
Group 3	7	0	1	1	1
	14	1	1	1	0

iii Combine minterms

Compare minterms from adjacent groups and combine minterms that differ in only one input bit by

$$XY + XY' = X,$$

where X is a product of literals and Y is a literal.

(Note: It is obvious that two terms not in adjacent groups can't be combined by $XY + XY' = X$.)

Create next list by writing combined terms with X preserved and Y, the differing bit, by $-$ (don't care), and writing not-touched terms as is.

Repeat combinations and create next list until no further combining is possible. The terms in this last list are the prime implicants of f.

For f:

List 1:

Group 0	0	0 0 0 0
Group 1	1	0 0 0 1
	2	0 0 1 0
	8	1 0 0 0
Group 2	5	0 1 0 1
	6	0 1 1 0
	9	1 0 0 1
	10	1 0 1 0
Group 3	7	0 1 1 1
	14	1 1 1 0

List 2:

Group 0	0, 1	0 0 0 –
	0, 2	0 0 – 0
	0, 8	– 0 0 0
Group 1	1, 5	0 – 0 1
	1, 9	– 0 0 1
	2, 6	0 – 1 0
	2, 10	– 0 1 0
	8, 9	1 0 0 –
	8, 10	1 0 – 0
Group 2	5, 7	0 1 – 1
	6, 7	0 1 1 –
	6, 14	– 1 1 0
	10, 14	1 – 1 0

List 3:

Group 0	0, 1, 8, 9	– 0 0 –
	20	

	$0, 2, 8, 10$	$-$	0	$-$	0
Group 1	$1, 5$	0	$-$	0	1
	$2, 6, 10, 14$	$-$	$-$	1	0
Group 2	$5, 7$	0	1	$-$	1
	$6, 7$	0	1	1	$-$

Sometimes, all rows that have appeared in any of the previous lists are omitted and all rows that have been combined into any row in the next list are checked or crossed out. The terms that are not checked or crossed out are the prime implicants of f.

For f:

List 1:

Group 0	0	0	0	0	0	
Group 1	1	0	0	0	1	
	2	0	0	1	0	
	8	1	0	0	0	
Group 2	5	0	1	0	1	
	6	0	1	1	0	
	9	1	0	0	1	
	10	1	0	1	0	
Group 3	7	0	1	1	1	
	14	1	1	1	0	

List 2:

Group 0	$0, 1$	0	0	0	$-$	
	$0, 2$	0	0	$-$	0	
	$0, 8$	$-$	0	0	0	
Group 1	$1, 5$	0	$-$	0	1	
	$1, 9$	$-$	0	0	1	
	$2, 6$	0	$-$	1	0	
	$2, 10$	$-$	0	1	0	

	8, 9	1	0	0	–
	8, 10	1	0	–	0
Group 2	5, 7	0	1	–	1
	6, 7	0	1	1	–
	6, 14	–	1	1	0
	10, 14	1	–	1	0

List 3:

Group 0	0, 1, 8, 9	–	0	0	–
	0, 2, 8, 10	–	0	–	0
Group 1	2, 6, 10, 14	–	–	1	0

iv Create prime implicant chart

Create the prime implicant chart, a chart where the column are minterms and rows are prime implicants and each cell is marked × if the corresponding minterm is implied by the corresponding prime implicant.

For f:

	0	1	2	5	6	7	8	9	10	14
$(0, 1, 8, 9)b'c'$	×	×					×	×		
$(0, 2, 8, 10)b'd'$	×		×				×		×	
$(1, 5)a'c'd$		×		×						
$(2, 6, 10, 14)cd'$			×		×				×	×
$(5, 7)a'bd$				×		×				
$(6, 7)a'bc$					×	×				

v Select ssential prime implicants

When a prime implicant is selected for inclusion in the minimum SOP form, the chart is reduced by deleting the corresponding row and the columns which correspond to the minterms covered by that prime implicant.

If a column has only one ×, the prime implicant of that row is an essential prime implicant.

Select all essential prime implicants.

For f: Select $(0,1,8,9)b'c'$ to cover m_9, and select $(2,6,10,14)cd'$ to cover m_{14}.

vi Petrick's method

Given the reduced chart, number all prime implicants as $P_1, P_2, \ldots$.

For f:

$$P_1 = a'c'd, \quad P_2 = a'bd, \quad P_3 = a'bc.$$

For each remaining minterm m_i, construct a sum term of all prime implicants that covers it, and let P be the product of them.

For f:

$$P = (P_1 + P_2)(P_2 + P_3).$$

Expand P into canonical SOP form (with each P_i considered a literal).

For f:

$$P = P_1P_2 + P_2 + P_2P_3.$$

Reduce P by applying

$$X + XY = X$$

until no more such reduction is possible.

For f:

$$P = P_2.$$

Each term in reduced P represents a solution, that is, a set of prime implicants which covers all minterms in the reduced chart. The sets with the least number of literals in it are the minimum SOP forms of f.

For f: P_2 is the only term in reduced P, thus the only minimum SOP form is

$$f = P_2 = b'c' + cd' + a'bd.$$

vii Incompletely specified Boolean functions

For incompletely specified Boolean functions, we do the following change to the process for completely specified Boolean functions:

1. Treat don't-care terms as minterms when grouping minterm.
2. Omit don't-care terms columns when creating the prime implicant chart.

2 Combinational Circuit

I Logic gate

A logic gate is an electronic device that performs a Boolean function.

i Logic gate symbols

Logic gates symbols (inputs at left, two inputs for example for AND, OR, NAND, NOR, XOR, and XNOR gate):

Type	Distinctive shape (ANSI/IEEE Std 91/91a-1991	Rectangular shape (IEEE Std 91/91a-1991/IEC 60617-12:1997)	Boolean function
(Logic) buffer (gate)			A
NOT gate / inverter			A'
AND gate			AND
OR gate			OR
NAND gate			NAND
NOR gate			NOR
XOR gate			XOR
XNOR gate			XNOR
IMPLY gate			IMPLY
NIMPLY gate			NIMPLY

An empty circle, called inversion bubble or bubble, means inverting the input before inputting to the gate when at a input to a gate, and means inverting the output before outputting from the gate when at the output from a gate.

The diagrams of logic circuits are called block diagrams. The number of gate inputs of a circuit is the sum of the number of inputs to all gates in the circuit.

ii Alternative symbols

By using inversion bubbles at the inputs instead of the output in logic gates symbol except IMPLY and NIMPLY gates, using inversion bubbles at the output and second input instead of the first input in IMPLY gate symbol, and using inversion bubbles at the second input instead of the first input and the output in NIMPLY gate symbol, we obtain the alternative gate symbols (inputs at left, two inputs for example for AND, OR, NAND, NOR, XOR, and XNOR gate):

Type	Distinctive shape (ANSI/IEEE Std 91/91a-1991	Rectangular shape (IEEE Std 91/91a-1991/IEC 60617-12:1997)	Boolean function
(Logic) buffer (gate)			A
NOT gate / inverter			A'
AND gate			AND
OR gate			OR
NAND gate			NAND
NOR gate			NOR
XOR gate			XOR
XNOR gate			XNOR
IMPLY gate			IMPLY
NIMPLY gate			NIMPLY

Sometimes multiple (typically two) outputs are drawn on a gate with some of them (typically one of the two) with inversion bubble. This is logically equivalent to drawing one gate of that type for each of the noninverted or inverted outputs with the same input lines.

iii Fan-ins

The maximum number of inputs to a gate is called the fan-in of the gate.

An AND, OR, NAND, NOR, XOR, or XNOR gate with n inputs or fan-in n is sometimes called with suffix n, e.g. NAND2 gate is a NAND gate with 2 inputs or fan-in 2.

An AND, OR, NAND, NOR, XOR, or XNOR gate without fan-in specified is sometimes assumed to have fan-in 2.

The fan-in of a practical logic gate is limited and may be two, three, four, eight, or some other number depending on the type of gates used.

iv Buffer

A gate output can only be connected to a limited number of other device inputs without degrading the performance of a digital system. A simple buffer may be used to increase the driving capability of a gate output.

II Switching circuit

i Digital system

A system that deals with signals that have discrete values.

ii Circuit (ckt)

A complete electrical network, that is, an interconnection of electrical components, with a closed-loop giving a return path for current.

iii Switching circuit or digital circuit

A circuit that deals with input and output signals that have discrete values, or, as typically the case, a logic circuit.

iv Logic circuit

A circuit that deals with binary input and output signals, that is, either 0 or 1.

v Analog system

Deals with signals that vary continuously over time. The output might have an error depending on the accuracy of the components used.

vi System design

The highest level of the design of systems, where you break the system into subsystems, specify what each subsystem do, and determine the interconnection and control of the subsystems.

vii (Digital) logic design

The middle level of the design of digital systems, where you specify the logic operations inside each subsystem.

viii Circuit design

The lowest level of the design of digital systems, where you specify the electronic components and their interconnection to form the system.

ix Combinational circuit

A switching circuit whose output value depends only on the present input value.

x Sequential circuit

A switching circuit whose output value depends on the present input value and past input values.

xi Switch, switching device, or switch element

A component that opens or closes a circuit.

xii Active high and low

Active high (aka active-high or noninverted) means the input or output signal is noninverted; active low (aka active-low or inverted) means the input or output signal is (logically equivalent to) inverted.

Unless otherwise specified, a type of electronic components with active high and low version of inputs and/or outputs are assumed to be active high inputs and/or outputs.

xiii Bus

Several signals that perform a common function may be grouped together to form a bus, represented with a single heavy line optionally with a diagonal slash through it specifying the number of bits in the bus.

xiv Pull-up and Pull-down resistors

In a digital circuit, if an input pin is not connected to anything (floating), it can pick up random electrical noise. To keep it in a definite voltage we can use pull-up or pull-down resistors.

A pull-up resistor is a resistor connected between a signal line and +Vcc to ensure that the signal stays at logic 1 when no active signal drives it low.

A pull-down resistor is a resistor connected between a signal line and ground to ensure that the signal stays at logic 0 when no active signal drives it high.

xv Feedback

Feedback occurs when the output of a circuit is fed back into its input. Sequential circuits must contain feedback, but not all circuits with feedback are sequential.

III Multi-level combinational circuits

i Introduction

The maximum number of gates cascaded in series between a circuit input and the output is referred to as the number of levels of gates. Thus, a function written in sum-of-products form or in product-of-sums form corresponds directly to a two-level gate circuit. As is usually the case in digital circuits where the gates are driven from flip-flop outputs, we will assume that all variables and their complements are available as circuit inputs. For this reason, we will not normally count inverters which are connected directly to input variables.

If a realization of a circuit requires more inputs to a gate than its fan-in, factoring or multiplying out the logic expression to obtain a more-level realization is necessary.

The level starting with the output gate is numbered 1.

Given multi-input logic gates $A_1, A_2, \ldots A_m$, $A_1 - A_2 - \ldots A_m i$ circuit means a m-level circuit composed of a level of A_1 gates followed by a level of A_2 gate followed by … an A_m gate at the output.

Given multi-input logic gates $A_1, A_2, \ldots A_m$, circuit of $A_1, A_2, \ldots A_m$ gates or $A_1, A_2, \ldots A_m$-gates circuits implies no particular ordering of the gates.

ii Design of circuits of AND and OR gates

The number of levels in an AND-OR circuit can be increased by factoring the sum-of-products expression from which it was derived. Similarly, the number of levels in an OR-AND circuit can be increased by multiplying out some of the terms in the product-of-sums expression from which it was derived. Sometimes doing so will reduce the required number of gates and gate inputs, and thus reduce the cost of building the circuit. In many application, the number of gates which can be cascaded is limited by gate delays. When the input of a gate is switched, there is a finite time before the output changes. When several gates are cascaded, the time between an input change and the corresponding change in the circuit output may become excessive and slow down the operation of the digital system.

In general, to be sure of obtaining a minimum solution, one must find both the circuit with the AND-gate output and the one with the OR-gate output. If an expression for f' has n levels, the complement of that expression is an n-level expression for f. Therefore, to realize f as an n-level circuit with an AND-gate output, one procedure is first to find an n-level expression for f' with an OR operation at the output level and then complement the expression for f'.

iii Degeneracy

A m-level circuit is degenerate iff it can degenerate into less-than-m-level. A circuit is non-degenerate iff it's not degenerate.

There are 8 degenerate forms, which are two-level circuits:

- AND-AND, which can degenerate into one AND,
- OR-OR, which can degenerate into one OR,
- AND-NAND, which can degenerate into one NAND,
- OR-NOR, which can degenerate into one NOR,
- NAND-NOR, which can degenerate into one AND,
- NOR-NAND, which can degenerate into one OR,
- NAND-OR, which can degenerate into one NAND,
- NOR-AND, which can degenerate into one NOR.

A circuit is degenerate iff any two-level subcircuit of it is of the 8 degenerate form.

A non-deegenerate circuit is an implementation of a sum-of-product-of-sum-…product or product-of-sum-of-product-…sum form.

iv Design of two-level circuits

There are 16 forms of two-level circuit consist of AND, OR, NAND, and NOR gates, 8 of which are the degenerate forms, 4 of which are implementation of SOP form, and 4 of which are implementation of POS form.

The non-degenerate forms that implement SOP form are:

- AND-OR, which can be directly derived from the SOP form,
- NAND-NAND, which can be converted from AND-OR form by replacing all gates with NAND gates and inverting all literals that are inputs to level 1, which can be derived from AND-OR form by taking double negation and applying De Morgan's law on level 1,
- OR-NAND, which can be converted from AND-OR form by replacing AND gates with OR gates and OR gate with NAND gate and inverting all literals, which can be derived from NAND-NAND form by applying De Morgan's law on level 2, or from NOR-OR form by applying De Morgan's law on level 1,
- NOR-OR, which can be converted from AND-OR form by replacing AND gates with NOR gates and inverting all inputs to level 2, which can be derived from AND-OR form by applying De Morgan's law on level 2.

The non-degenerate forms that implement POS form are

- OR-AND, which can be directly derived from the POS form,
- NOR-NOR, which can be converted from OR-AND form by replacing all gates with NOR gates and inverting all literals that are inputs to level 1, which can be derived from OR-AND form by taking double negation and applying De Morgan's law on level 1,
- AND-NOR, which can be converted from OR-AND form by replacing OR gates with AND gates and AND gate with NOR gate and inverting all literals, which can be derived from NOR-NOR form by applying De Morgan's law on level 2, or from NAND-AND form by applying De Morgan's law on level 1,
- NAND-AND, which can be converted from OR-AND form by replacing OR gates with NAND gates and inverting all inputs to level 2, which can be derived from OR-AND form by applying De Morgan's law on level 2.

The procedure of designing a two-level circuit is

1. Simplify the Boolean function to be realized to SOP or POS form.
2. Design an AND-OR or OR-AND circuit.
3. Convert it to the wanted form.

v Design of multi-level NAND- and NOR-gate circuits

The procedure of designing a multi-level NAND-gate circuit is

1. Simplify the Boolean function to be realized to the wanted sum-of-product-of-sum-…product form.
2. Design an AND-OR-AND-…OR or OR-AND-OR…AND-OR circuit.
3. Replace all gates with NAND gates and invert all literals that are inputs to levels of odd numbers.

The procedure of designing a multi-level NOR-gate circuit is

1. Simplify the Boolean function to be realized to the wanted product-of-sum-of-product-…sum form.
2. Design an OR-AND-OR…AND or AND-OR-AND-…OR-AND circuit.
3. Replace all gates with NOR gates and invert all literals that are inputs to levels of odd numbers.

vi Circuit conversion using inversion bubbles

Adding or cancelling inversion bubbles to both ends of any interconnection does not change the Boolean function realized by the circuit. Adding or cancelling even number of inversion bubbles at an input or output does not change the Boolean function realized by the circuit.

Therefore, the gate type at levels i and $i+1$ can be changed by inserting inversion bubbles on both sides of all connections between levels $i+1$ and i. After this operation, the gates at level $i+1$ may appear with double inversion bubbles at output and can be cancelled together, the gates at level i will appear in their alternate symbols (with input bubbles) and can be changed to the standard form. The changing of level $i+1$ after this operation is

Original gate	New gate
Buffer gate	Not gate
Not gate	Buffer gate
AND gate	NAND gate
OR gate	NOR gate
NAND gate	AND gate
NOR gate	OR gate
XOR gate	XNOR gate
XNOR gate	XOR gate

The changing of level i after this operation is

Original gate	New gate
Buffer gate	Not gate
Not gate	Buffer gate
AND gate	NOR gate
OR gate	NAND gate
NAND gate	OR gate
NOR gate	AND gate
XOR gate	XNOR gate
XNOR gate	XOR gate

IV Multiple-output combinational circuits

i Minimal cost realization

Solution of digital design problems often requires the realization of several completely or incompletely specified Boolean functions of the same variables, that is, a completely or incompletely specified Boolean vector function. Although each function could be realized separately, the use of some gates in common between two or more functions sometimes leads to a more economical realization. Thus in realizing multiple-output circuits, the use of a minimum SOP for each function does not necessarily lead to a minimum cost realization for the circuit as a whole.

When designing multiple-output circuits, you should try to minimize the total number of gates required. If several solutions require the same number of gates, the one with the minimum number of gate inputs should be chosen.

ii Karnaugh Maps

1. Draw K-map for each output.
2. Group 1's in the same definition in K-maps for single Boolean function; however the collection of groups is the collection of all groups on all maps, in which groups including same cells on different maps are the same and corresponding to only one element in the collection.
3. An allowed collection of groups must follow the following rules:
    - Any 1 must be in at least one group.
    - If any 1 is only covered by one possible group, then that group must be choose. The product represented by the group is called an essential prime implicant of the Boolean vector function.
    - If any group g is completely covered by another group, g must not be chosen. The product represented by a group that is allowed to be chosen according to this rule is called an

prime implicant of the Boolean vector function.

4. Find the allowed collections of groups that contain the fewest groups. For each one of them, the sum of the products corresponding to the groups in it derives a minimal-gate SOP realization of the Boolean vector function.
5. Among the realization(s), find the one with fewest literal inputs, that is the minimum cost realization of the Boolean vector function.

iii Design of multiple-output NAND- and NOR-gate circuits

The procedure of designing a multiple-output NAND-gate circuit is

1. Simplify the Boolean vector function to be realized to the wanted sum-of-product-of-sum-…product form.
2. Design an AND-OR-AND-…OR or OR-AND-OR…AND-OR circuit.
3. Replace all gates with NAND gates and invert all literals that are inputs to levels of odd numbers.

The procedure of designing a multiple-output NOR-gate circuit is

1. Simplify the Boolean vector function to be realized to the wanted product-of-sum-of-product-…sum form.
2. Design an OR-AND-OR…AND or AND-OR-AND-…OR-AND circuit.
3. Replace all gates with NOR gates and invert all literals that are inputs to levels of odd numbers.

V Delays and Hazards

i Propagation delays and timing diagram

When the input to a logic gate is changed, the output will not change instantaneously but change after a finite delay. If the change in output is delayed by time ε with respect to the input, we say that this gate has a propagation delay of ε. In practice, the propagation delay for a 0 to 1 output change may be different than the delay for a 1 to 0 change. Propagation delays for integrated circuit gates may be as short as a few nanoseconds, and in many cases these delays can be neglected. However, in the analysis of some types of sequential circuits, even short delays may be important.

Timing diagram shows various signals in the circuit as a function of time, often in waveform with time being the horizontal axis and each vertical block plots one signal with the same time scale. The vertical edges on the diagram where the signals change from 0 to 1 are called rising edges; the vertical edges on the diagram where the signals change from 1 to 0 are called falling edges.

The propagation delay of a digital circuit is the time it needs to stabilize.

ii Inertial delays

If a logic gate will not respond to an input change that is shorter than a certain minimum pulse width ε, we say that this gate has a inertial delay of ε. Quite often the inertial delay value is assumed to be the same as the propagation delay of the gate. In contrast, if a gate always responds to input

changes (with a propagation delay), no matter how closely spaced the input changes may be, the gate is said to have an ideal or transport delay.

iii Hazards

When the input to a combinational circuit changes, unwanted switching transients may appear in the output. These transients occur when different paths from input to output have different propagation delays. If, in response to any single input change and for some combination of propagation delays, a circuit output may momentarily go to 0 when it should remain a constant 1, we say that the circuit has a (static) 1-hazard (or static-1 hazard). Similarly, if the output may momentarily go to 1 when it should remain a 0, we say that the circuit has a (static) 0-hazard (or static-0 hazard). If, when the output is supposed to change from 0 to 1 (or 1 to 0), the output may change three or more times, we say that the circuit has a dynamic hazard. In each case the steady-state output of the circuit is correct, but a switching transient appears at the circuit output when the input is changed.

Hazards in a two-level circuit:

- A two-level circuit realizing a SOP form has no static-0 or dynamic hazard. Static-1 hazards in a two-level circuit realizing a SOP form can be detected using a Karnaugh map, in which if any two adjacent 1's are not covered by a same group corresponding to a product term, a 1-hazard exists for the transition between the two 1's, which corresponds to an eliminated consensus term. For an n-variable map, this transition occurs when one variable changes and the other $n-1$ variables are held constant. A such static-1 hazard is denoted as the minterms corresponding the two adjacent 1's with $\leftrightarrow$ beteen them, e.g. 0100$\leftrightarrow$0101. We can eliminate a static-1 hazard by adding a group (product term) that covers the two 1's.
- A two-level circuit realizing a POS form has no static-1 or dynamic hazard. Static-0 hazards in a two-level circuit realizing a POS form can be detected using a Karnaugh map, in which if any two adjacent 0's are not covered by a same group corresponding to a sum term, a 0-hazard exists for the transition between the two 0's, which corresponds to an eliminated consensus term. For an n-variable map, this transition occurs when one variable changes and the other $n-1$ variables are held constant. A such static-0 hazard is denoted as the minterms corresponding the two adjacent 0's with $\leftrightarrow$ beteen them, e.g. 0100$\leftrightarrow$0101. We can eliminate a static-0 hazard by adding a group (sum term) that covers the two 0's.

Hazards in a multi-level circuit can be detected by deriving either a SOP or POS expression for the circuit that represents a two-level circuit containing the same hazards as the original circuit. The SOP or POS expression is derived in the normal manner except that the complementation laws, i.e. $XX' = 0$ and $X + X' = 1$, are not used. Consequently:

- The resulting SOP expression may contain products of the form $xx'\alpha$, where α is a product of literals or nothing. In the SOP expression, a product of the form $xx'\alpha$ represents a pseudo AND gate that may temporarily have the output value 1 as x changes if $\alpha = 1$, which causes static-0 or dynamic hazards. A such hazard is denoted as the $xx'\alpha$ term.
- The resulting POS expression may contain sums of the form $x + x' + \beta$, where β is a sum of literals or nothing. In the POS expression, a sum of the form $x + x' + \beta$ represents a pseudo OR

gate that may temporarily have the output value 0 as x changes if $\beta = 0$, which causes static-1 or dynamic hazards. A such hazard is denoted as the $x + x' + \beta$ term.

To design a circuit which is free of static and dynamic hazards, the following procedures may be used:

1. Find a SOP expression for the function in which every pair of adjacent 1's is covered by a 1-term. (The sum of all prime implicants always satisfies this condition.) A two-level AND-OR circuit based on this expression will be free of all hazards.
2. If a different form of the circuit is desired, manipulate the expression to the desired form but treat each independent variable and its inverse as two independent variables to prevent introduction of hazards.

or

1. Find a POS expression for the function in which every pair of adjacent 0's is covered by a 0-term. (The product of all prime implicate always satisfies this condition.) A two-level OR-AND circuit based on this expression will be free of all hazards.
2. If a different form of the circuit is desired, manipulate the expression to the desired form but treat each independent variable and its inverse as two independent variables to prevent introduction of hazards.

It should be emphasized that the discussion of hazards and the possibility of resulting glitches in this section has assumed that only a single input can change at a time and that no other input will change until the circuit has stabilized. If more than one input can change at one time, then nearly all circuits will contain hazards, and they cannot be eliminated by modifying the circuit implementation.

VI Simulation and testing of logic circuits

i Verilog, SystemVerilog, and VHDL

A hardware description language (HDL) is a specialized computer language used to describe the structure and behavior of electronic circuits.

Verilog, standardized as IEEE 1364, is a hardware description language used to model and simulate electronic circuits.

SystemVerilog, an extension of Verilog, standardized as IEEE 1800, is a hardware description and hardware verification language used to model, simulate, and test electronic circuits.

VHDL (VHSIC Hardware Description Language), standardized as IEEE Std 1076, is a hardware description language used to model and simulate electronic circuits.

ii Four-valued logic

The two logic values, 0 and 1, are not sufficient for simulating logic circuits. At times, the value of a gate input or output may be unknown, which is represented by X (`x` in Verilog/SystemVerilog). At other times we may have no logic signal at an input, as in the case of an open circuit when an input

is not connected to any output, called high impedance or hi-Z/Hi-Z connection, which is represented by Z (`z` in Verilog/SystemVerilog).

The logical connectives are defined as follows (without loss of commutativity of operators):

$$X \cdot 0 = Z \cdot 0 = 0,$$

$$X \cdot 1 = X \cdot X = X \cdot Z = Z \cdot 1 = Z \cdot Z = X,$$

$$X + 1 = Z + 1 = 1,$$

$$X + 0 = X + X = X + Z = Z + 0 = Z + Z = X.$$

iii Simulation and testing

A simple simulator for combinational logic works as follows:

1. The circuit inputs are applied to the first set of gates in the circuit, and the out-puts of those gates are calculated.
2. The outputs of the gates which changed in the previous step are fed into the next level of gate inputs. If the input to any gate has changed, then the output of that gate is calculated.
3. Step 2 is repeated until no more changes in gate inputs occur. The circuit is then in a steady-state condition, and the outputs may be read.
4. Steps 1 through 3 are repeated every time a circuit input changes.

If a circuit output is wrong for some set of input values, this may be due to several possible causes:

- incorrect design
- Gates connected wrong
- Wrong input signals to the circuit

If the circuit is physically built, other possible causes include

- Defective gates
- Defective connecting wires

It is very easy to locate the problem systematically by starting at the output and working back through the circuit until the trouble is located.

VII Integrated Circuit (IC)

An integrated circuit (IC), also known as a microchip or simply chip, is a compact assembly of electronic circuits, in which the components are fabricated onto a thin piece (called chip) of semiconductor material, most commonly silicon.

Integrated circuits can be broadly classified into analog, digital and mixed-signal, consisting of analog and digital signaling on the same IC.

i Integrated circuit package

An integrated circuit package is the physical case that holds the silicon chip (die) and provides the electrical and mechanical connection between the chip's microscopic circuits and the outside world. An IC package typically has:

- Silicon die (chip): The actual IC.
- Bond wires: Typically tiny gold or aluminum wires.
- Encapsulation: Protective material that shields the die from damage.
- Pins or leads: Metal terminal that connect the IC to a printed circuit board (PCB).

ii Scales

Acronym	Name	Year	Transistor count	Logic
SSI	small-scale integration	1964	1 to 10	
MSI	medium-scale integration	1968	10 to 500	
LSI	large-scale integration	1971	500 to 20000	10
VLSI	very-large-scale integration	1980	20000 and more (or 20000 to 1000000)	10000 and mo
(ULSI	ultra-large-scale integration	1964	1000000 and more	1000

It is generally uneconomical to design digital systems using only SSI and MSI integrated circuits. By using LSI and VLSI ICs, the required number of integrated circuit packages is greatly reduced, and the cost of mounting, wiring, designing, and maintaining may be significantly lower.

VIII Combinational circuits

i Multiplexer (MUX)

A multiplexer (MUX) or data selector has a group of data inputs and a group of select inputs (aka control inputs, select lines, or control lines) and one output. The select inputs are used to select one of the data inputs and connect it to the output terminal. Multiplexers are frequently used in digital system design to select the data which is to be processed or stored.

A 2^n-to-1 MUX has 2^n data inputs $D_0, D_1, \dots D_{2^n-1}$, n select inputs $S_0, S_1, \dots S_{n-1}$, and one output Y, with Y given by

$$Y = \sum_{i=0}^{2^{n-1}} D_i \cdot m_i(S_0, S_1, \dots S_{n-1}),$$

where m_i is the minterm corresponding to i for n-variable functions of $S_0, S_1, \dots S_{n-1}$. When $m_i = 1$, that is, the binary number $S_0 S_1 \dots S_{n-1}$ is i, I_i is the selected input.

Alternatively, Y can be written as

$$Y = \prod_{i=0}^{2^{n-1}} \left(D_i + M_i(S_0, S_1, \dots S_{n-1}) \right),$$

where M_i is the maxterm corresponding to i for n-variable functions of $S_0, S_1, \dots S_{n-1}$.

The symbol of a MUX in block diagrams is an isosceles trapezoid with the longer parallel side contains the data input pins in order of the unsigned binary number corresponding to the select inputs combination for each data input to be selected with the select inputs combination for each data input to be selected written beside in the trapezoid, one of the legs, usually the leg farther from the data input that is selected when all select inputs are 0, contains the select inputs, and the shorter parallel side contains the output pin.

We can implement a MUX with a two- or multi-level circuit that realizes the SOP or POS form given above.

A MUX with active high outputs outputs the selected input directly, while a MUX with active low outputs outputs the inverse of the selected input. A MUX can also have an additional input E called enable (signal) and works as discussed above when $E = 1$ and outputs 0 when $E = 0$.

A 2^n-to-1 multiplexer can be realized with $\left(2^{n-m+1} - 1\right)$ 2^m-to-1 multiplexers ($m \leq n$).

Given a n-variable switching function F, 2^m m-variable subfunctions of F can be obtained using Shannon's expansion of the function and compose back to F with one 2^{n-m}-to-1 multiplexer. Take $m = 1$, we can get a realization of F with one 2^{n-1}-to-1 multiplexer.

ii Three-state buffer or tri-state buffer

A three-state buffer (or tri-state buffer) is a logic gate that has three stable states: logic HIGH (1), logic LOW (0), and high impedance (Z). In the high impedance state, the output of the buffer is effectively disconnected from the subsequenct circuit. A three-state buffer has one data input and one output like a simple buffer and one additional enable input signal En. It works as a simple buffer when $En = 1$ and outputs high impedance when $En = 0$.

When a bus is driven by three-state buffers, we call it a three-state bus.

We can also add an inversion bubble at enable signal such that it works as a simple buffer when $En = 0$ and outputs high impedance when $En = 1$. We can also add an inversion bubble at either data input or output such that when it doesn't outputs high impedance, it works as an inverter.

We can implement a 2-to-1 MUX by using data inputs of the MUX as data inputs of two tri-state buffers, using the select input and its inverse as enable signal of the two tri-state buffers respectively, and simply connecting the two outputs of the two tri-state buffers to the output of the MUX.

iii Bidirectional I/O pin

Integrated circuits are often designed using bidirectional (or bi-directional) I/O pins for input and output to save pins and wiring. A bidirectional pin is a single physical pin that can act as either an input or an output, but not both at the same time, depending on the control signal. To accomplish this,

the circuit output is connected to the pin through a three-state buffer. When the buffer is enabled, the pin is driven with the output signal; when the buffer is disabled, an external source can drive the pin.

iv Half adder (HA)

A half adder is a combinational circuit that has two inputs, A (augend bit) and B (addend bit), and produces two ouputs S (sum) and C_{out} (carry), in which

$$S = A \oplus B,$$

$$C_{out} = AB.$$

The symbol of a half adder in block diagrams is a square with ”half adder” in it with augend bit and addend bit inputs lines and sum output line on two parallel sides; let x-axis be the line perpendicular to the two sides oriented from augend bit and addend bit inputs lines to sum output line, z-axis be oriented out of the paper; carry output with arrows on one of the other side with direction of it in positivite y direction.

v Half subtractor (HS)

A half subtractor is a combinational circuit that has two inputs, A (minuend bit) and B (subtrahend bit), and produces two ouputs D (difference) and B_{out} (borrow), in which

$$D = A \oplus B,$$

$$B_{out} = A'B.$$

The symbol of a half subtractor in block diagrams is a square with ”half subtractor” in it with minuend bit and subtrahend bit inputs lines and difference output line on two parallel sides; let x-axis be the line perpendicular to the two sides oriented from minuend bit and subtrahend bit inputs lines to difference output line, z-axis be oriented out of the paper; borrow output with arrows on one of the other side with direction of it in positivite y direction.

vi Full adder (FA)

A full adder is a combinational circuit that has three inputs, A (augend bit), B (addend bit), and C_{in} (carry-in), and produces two ouputs S (sum) and C_{out} (carry-out), in which

$$S = A \oplus B \oplus C_{in},$$

$$C_{out} = AB + AC_{in} + BC_{in}.$$

The symbol of a full adder in block diagrams is a square with ”full adder” in it with augend bit and addend bit inputs lines and sum output line on two parallel sides; let x-axis be the line perpendicular to the two sides oriented from augend bit and addend bit inputs lines to sum output line, z-axis be oriented out of the paper; carry-in input and carry-out output with arrows on the other two sides respectively with direction of them in positivite y direction and from carry-in input to carry-out output.

vii Full subtractor (FS)

A full subtractor is a combinational circuit that has three inputs, A (minuend bit), B (subtrahend bit), and B_{in} (borrow-in), and produces two ouputs D (difference) and B_{out} (borrow-out), in which

$$D = A \oplus B \oplus B_{in},$$

$$B_{out} = A'B + A'B_{in} + BB_{in}.$$

The symbol of a full subtractor in block diagrams is a square with "full subtractor" in it with minuend bit and subtrahend bit inputs lines and difference output line on two parallel sides; let x-axis be the line perpendicular to the two sides oriented from minuend bit and subtrahend bit inputs lines and difference output line, z-axis be oriented out of the paper; borrow-in input and borrow-out output with arrows on the other two sides respectively with direction of them in positivite y direction and from borrow-in input to borrow-out output.

viii Parallel (binary) adder or ripple-carry adder (RCA)

An n-bit parallel adder or ripple-carry adder, or simply adder is a combinational circuit consisting of n full adders called cells or stages. It has two n-bits unsigned or two's complement binary numbers as inputs (MSB to LSB), $A_{n-1}A_{n-2}\dots A_0$ (augend) and $B_{n-1}B_{n-2}\dots B_0$ (addned), and a constant 0 input, $C_{in_0} = 0$, and produces the $(n+1)$-bit sum of the two numbers as outputs, $C_{out_{n-1}}S_{n-1}S_{n-2}\dots S_0$, in which the $(i+1)$th cell takes A_i as augend bit, B_i as addend bit, and C_{in_i} as carry-in, and produces sum S_i and carry-out C_{out_i}, that is,

$$S_i = A_i \oplus B_i \oplus C_{in_i},$$

$$C_{out_i} = A_iB_i + A_iC_{in_i} + B_iC_{in_i},$$

where C_{in_i} for $n \geq i > 0$ and C_{out_i} for $n-1 > i \geq 0$ are interconnections in the parallel adder, and for the first cell,

$$S_0 = A_0 \oplus B_0,$$

$$C_{out_0} = A_0B_0.$$

The ripple-carry adder is relatively slow because, in the worst case, a carry propagates through all n stages of the adder before it stabilizes, and there are approximately two gate delays per stage.

ix Parallel (binary) subtractor or ripple-borrow subtractor (RBS)

An n-bit parallel subtractor or ripple-borrow adder is a combinational circuit consisting of n full subtractors called cells or stages. It has two n-bits unsigned or two's complement binary numbers as inputs (MSB to LSB), $A_{n-1}A_{n-2}\dots A_0$ (minuend) and $B_{n-1}B_{n-2}\dots B_0$ (subtrahend), and a constant 0 input, $B_{in_0} = 0$, and produces the $(n+1)$-bit two's complement difference of the two numbers as outputs, $B_{out_{n-1}}D_{n-1}D_{n-2}\dots D_0$, in which the $(i+1)$th cell takes A_i as minuend bit, B_i as subtrahend bit, and B_{in_i} as borrow-in, and produces difference D_i and borrow-out B_{out_i}, that is,

$$D_i = A_i \oplus B_i \oplus B_{in_i},$$

$$B_{out_i} = A_i'B_i + A_i'B_{in_i} + B_iB_{in_i},$$

where B_{in_i} for $n \geq i > 0$ and B_{out_i} for $n - 1 > i \geq 0$ are interconnections in the parallel subtractor, and for the first cell,

$$D_0 = A_0 \oplus B_0,$$
$$D_{out_0} = A_0' B_0.$$

The ripple-borrow subtractor is relatively slow because, in the worst case, a borrow propagates through all n stages of the subtractor before it stabilizes, and there are approximately two gate delays per stage.

x Adder-subtractor

A logical equivalence of a parallel subtractor that is used more often is a parallel adder with inversion bubbles at all B_i inputs and C_{in_0} being constant 1 input, where $B'_{n-1}B'_{n-2} \dots B'_0 + 1$ represents $-B_{n-1}B_{n-2} \dots B_0$ in two's complement.

An n-bit adder-subtractor is a ripple-carry adder (or other type of n-bit adder for unsigned or two's complement numbers) with an additional common select input M and n 2-to-1 MUXes, in which the MUX at the $(i + 1)$th cell take data inputs B_i and B_i' and select input M, if $M = 0$, it outputs B_i to the full adder to make the adder-subtractor perform addition; if $M = 1$, it outputs B_i' to the full adder to make the adder-subtractor perform subtraction.

xi One's complement adder

An n-bit one's complement adder can be realized with an n-bit ripple-carry adder but connect $C_{out_{n-1}}$ to C_{in_0} to realize end-around carry (EAC). The propagation delay when $C_{out_{n-1}} = 1$ is approximately twice the propagation delay of the ripple-carry adder.

xii One's complement subtractor

An n-bit one's complement subtractor can be realized with an n-bit ripple-borrow subtractor but connect $B_{out_{n-1}}$ to B_{in_0} to realize end-around borrow (EAB). The propagation delay when $B_{out_{n-1}} = 1$ is approximately twice the propagation delay of the ripple-borrow subtractor.

An n-bit one's complement subtractor can also be realized with a one's complement adder but invert all bits of the subtrahend.

xiii Carry-lookahead adder (CLA)

An n-bit carry lookahead adder has the same inputs and outputs of an n-bit ripple-carry adder, that is, it has two n-bits unsigned or two's complement binary numbers as inputs (MSB to LSB), $A_{n-1}A_{n-2} \dots A_0$ (augend) and $B_{n-1}B_{n-2} \dots B_0$ (addned), and a constant 0 input, $C_0 = 0$, and produces the $(n + 1)$-bit sum of the two numbers as outputs, $C_n S_{n-1} S_{n-2} \dots S_0$, and is a combinational circuit designed to speed up addition from $O(n)$ in a ripple-carry adder to ideally $O(1)$ and practically $O(\log n)$ (hierarchical lookahead) by reducing the carry propagation delay.

A revised full adder has the same inputs like a full adder but outputs are changed from sum and carry-out to original sum S and generate G and propagate P defined with

$$G = AB,$$

$$P = A \oplus B,$$

$$S = A \oplus B \oplus C_{in}.$$

Suppose the maximum number of fan-ins of our add gates is m, we can implement carry-lookahead circuit up to $m-1$ bit. A n-bit (usually $n = 4 = m-1$) carry-lookahead circuit has n generate inputs G_0 to G_{n-1}, n propagate inputs P_0 to P_{n-1}, one carry-in input C_0, and produces n carry-out outputs C_1 to C_n given recursively by

$$C_{i+1} = G_i + P_i C_i$$

but realized in expanded SOP form, which makes it $O(1)$ but limited by gate fan-ins, e.g.

$$C_{i+2} = G_{i+1} + P_{i+1} G_i + P_{i+1} P_i C_i.$$

A n-bit carry-lookahead has n revised full adders called cells or stages and one n-bit carry-lookahead circuit, in which the $(i+1)$th cell takes A_i as augend bit, B_i as addend bit, and C_i as carry-in, and produces sum S_i, generate G_i, and propagate P_i, and the carry-lookahead circuit take generate inputs G_0 to G_{n-1}, propagate inputs P_0 to P_{n-1}, carry-in input C_0, and produces n carry-out outputs C_1 to C_n.

When the maximum number of fan-ins is not greater than the number of bit needed, we use hierarchical lookahead structure, in which the carry-lookahead circuit is replaced with a tree of physically possible m-bit carry-lookahead circuits. A m^k-bit carry-lookahead adder consists of k levels of carry-lookahead circuits, in which the lth level consists m^{k-l} m-bit carry-lookahead circuits, each of the carry-lookahead circuits that are not in the last level (which contains only one carry-lookahead circuit) is revised to output generate g given recursively by

$$g_0 = G_0,$$

$$g_{i+1} = G_{i+1} + P_i g_i,$$

$$g = g_{m-1},$$

and propagate p given by

$$p = \prod_{i=1}^{m-1} P_i,$$

instead of carry-out outputs, where m G_i and m P_i are the generates and propagates of the revised full adders ditributed in order to each of them, and each of the carry-lookahead circuits that are not in the first level takes m generates and m propagates of the previous level in order and computed the same as a normal m-bit carry-lookahead circuit by treating the generate and propagate of the $m*(i-1)+j$th carry-lookahead circuit in the previous level as the jth generate and propagate inputs of the ith carry-lookahead circuit in this level.

xiv Decoders

A decoder is a combinational circuit that converts binary input codes into a single active output line. An n-to-2^n decoder has n input signals and 2^n output signals realizing a one-to-one Boolean vector function $\{0,1\}^n \to \{0,1\}^{2^n}$ that map each input combination to an output vector of which the 1-norm is 1 (noninverted/active-high outputs) or $2^n - 1$ (inverted/active-low outputs).

Each possible output of a decoder corresponds to a minterm of the inputs. Thus, we can OR them together to get the active-high outputs or NAND them together to get the active-low outputs.

The symbol of an n-to-2^n decoder in block diagrams is a rectangle with inputs lines and output lines with arrows on two parallel sides and "(n-to-2^n) decoder" in it.

xv Normal encoders

A normal encoder is a combinational circuit that realize the inverse of a Boolean vector function realized by a decoder. A 2^n-to-n normal encoder realizes the inverse of a Boolean vector function $\{0,1\}^n \to \{0,1\}^{2^n}$ realized by a decoder, of which the domain of the inverse is a subset of $\{0,1\}^{2^n}$ that has n elements, each of which with the 1-norm being 1 (noninverted/active-high inputs) or $(2^n - 1)$ (inverted/active-low inputs). If the input combination is not in the domain, the outputs are undefined.

The symbol of a 2^n-to-n normal encoder in block diagrams is a rectangle with inputs lines and output lines with arrows on two parallel sides and "(2^n-to-n) normal encoder" in it.

xvi Priority encoders

A priority encoder resolves the undefined behaviour of normal encoders by assigning priority to the inputs.

A 2^n-to-n active-high inputs and outputs priority encoder has 2^n inputs and $(n+1)$ outputs. If all inputs are 0, the $(n+1)$-th output, called valid (V) or enable output (EO), is 0 and the other n outputs are undefined; otherwise the $(n+1)$-th output is 1 and the other n outputs are determined by a loop: `for (int i=1; i<=2n; i=i+1)` (in which `2^n` means 2 to the power of n):

- If the input of ith priority is 1, the output combination is the output combination of a active-high inputs normal encoder when that input is 1 and all other inputs are 0.

The priority of inputs are usually as: for any $1 \le i < 2^n$, the $(i+1)$-th input is prior to the i-th input.

An active-low inputs priority encoder invert the inputs; an active-low outputs priority encoder invert the outputs.

The symbol of a 2^n-to-n priority encoder in block diagrams is a rectangle with inputs lines and output lines with arrows on two parallel sides and "(2^n-to-n) priority encoder" in it.

IX Read-only memory (ROM)

A read-only memory (ROM) is a non-volatile (data is kept even when power is off) memory consists of an array of semiconductor devices that are interconnected to store an array of binary data. Once binary data is stored in the ROM, it can be read out whenever desired, but the data that is stored cannot be changed under normal operating conditions.

A k(-word)×m-bit ROM has $n = \lceil \log_2 k \rceil$ input lines and m output lines, and contains an array of k words, each word of width m bits, where k often equals 2^n. An input combination serves as an address to select one of the k words and output it from the ROM, that is, m n-variable functions are

stored in it.. Typical sizes for commercially available ROMs range from 32 words $\times$ 4 bits to 512K words $\times$ 8 bits, or larger.

A k-word$\times m$-bit ROM stores that has $n = \lceil \log_2 k \rceil$ input lines consist of an n-to-2^n decoder and a $k \times m$ memory array, aka OR plane. The latter consists of k rows, called word lines and each of which connected to some of the decoder output lines, and m columns, which are output lines of the ROM and connect to ground at the opposite end. In each of the $k \times m$ row-column intersection points, which corresponds to one bit, a diode, as a switching element, is either present and connected or not. If absent or not connected (logic 0), the signal value from the row isn't passed to the column; if present and connected (logic 1), represented by a $\times$ on the diagram, the signal value from the row is passed to the column. An output line outputs 1 if there are logic 1 signals from the word lines passed to it and 0 if there isn't.

The ROM table of a ROM is the truth tables of the functions realized combined by combining rows with same input combinations and listing the outputs of the functions in columns.

The symbol of a k-word$\times m$-bit ROM in block diagrams is a rectangle with inputs lines and output lines with arrows on two adjacent sides and "(k-word$\times m$-bit) ROM" in it, or an n-to-2^n decoder and a $k \times m$ memory array, with the latter being a rectangle with word lines with arrows, which are connected to the decoder outputs, and output lines with arrows, which are connected to ground at the starting end, on two adjacent sides.

Common types of ROMs:

- Mask-programmable ROMs: Programmed permanently during manufacturing using photolithography by selectively including or omitting the diodes at the cells. The mask used in photolithography is expensive, so the use of mask-programmable ROMs is economically feasible only if a large quantity (typically several thousand or more) is required with the same data array.
- Programmable ROMs (PROM): Manufactured with diodes at all cells and programmable once (one-time programmable (OTP)) by the user using a PROM programmer (aka PROM burner), which sends high-voltage pulses to burn tiny fuses in the cell to create permanent 1's and 0's.
- Erasable Programmable ROMs (EPROM): Manufactured with each cell contains a control gate that receives normal programming or read voltages and a floating gate that is completely insulated by oxide and can store electric charge and with quartz window on top, programmable by the user using a PROM programmer (aka PROM burner), which sends high-voltage pulses to inject electrons into floating gates, and erasable by the user by expose it to intense UV light that provides photons providing energy to remove trapped electrons from the floating gate simultaneously at all celles.
- Electrically Erasable Programmable ROMs (EEPROM): Similar to EPROM but erasing process is done electrically to remove trapped electrons from the floating gate one bit or one byte at a time (instead of all cells simultaneously) only a limited number of times, typically 100 to 1000 times.
- Flash memory: Derived from EEPROM but optimized to be faster, denser, cheaper, and

erasable in blocks or pages.

NOR and NAND flash:

- NOR Flash: Output lines are connected in parallel and cells are erased in blocks and with random access for read. Writing and erasing is slower and cell size is larger.
- NAND Flash: Output lines are connected in series and cells are erased in pages and with sequential access for read only. Writing and erasing is faster and cell size is smaller.

Type of cells:

Type	Bits per cell	Characteristics
Single-level cell (SLC)	1	Fast, reliable, expensive
Multi-level cell (MLC)	2	Slower, cheaper
TLC (Triple-Level Cell)	3	Used in consumer solid-state drives (SSDs)
QLC (Quad-Level Cell)	4	Denser, slower, less endurable

Flash memories usually have built-in programming and erase capability so that data can be written to the flash memory while it is in place in a circuit without the need for a separate programmer.

X Programmable Logic Devices (PLDs)

A programmable logic device (PLD) is a general name for a digital integrated circuit capable of being programmed to provide a variety of different logic function. Simple combinational PLDs are capable of realizing from 2 to 10 functions of 4 to 16 variables with a single integrated circuit. More complex sequential PLDs may contain thousands of gates and flip-flops. When a digital system is designed using a PLD, changes in the design can easily be made by changing the programming of the PLD without having to change the wiring in the system, which leads to lower cost designs.

i Programmable Logic Arrays (PLAs)

A programmable logic array (PLA) performs the same basic function as a ROM. However, the internal organization of the PLA is different from that of the ROM. The decoder is replaced with an AND array which realizes selected product terms of the input variables. The OR array ORs together the product terms needed to form the output functions, so a PLA implements a sum-of-products expression, while a ROM directly implements a truth table. Product terms are formed in the AND array by connecting diodes as switching elements at appropriate points in the array. Outputs are formed in the OR array by connecting diodes as switching elements at appropriate points in the array.

A k(-input)×m(-word(-by))×n(-output) PLA has k inputs, m product terms (aka words or minterms), and n outputs, and can realize n k-variable functions in POS form with at most m product terms in total, where product terms in different functions may be shared. A product term with i literals needs i diodes in the AND array. A sum form with j product terms needs j diodes in the OR array.

The contents of a PLA can be specified by a PLA table that lists all product terms in it. A PLA table of an n-output PLA consists of three big columns. The first big column are product terms written in product of literals. The second big columns are input combinations of the product terms with – indicating don't care. The third big column consists of n small columns, each of which contains one output of the PLA, where if the product term is connected by a diode to that output line, enter 1; otherwise, enther 0. Sometimes the first big column is omitted.

The symbol of a $k \times m \times n$ PLA in block diagrams is k input lines corresponding to each variable, each with variable name labeled at one end, indicating input side, each then split into two lines with one of them passing an inverter and the other not, forming AND array, n output lines parallel to the $2k$ AND array input lines, each with output name labeled at the end other than those of the input lines, indicating output side, forming OR array, and m word lines perpendicular to the $2k$ AND array input lines and the n output lines passing through them forming $2k \times m$ AND-array intersections, with a $\times$ at the intersection if a diode is connected there in the AND array, and $n \times m$ OR-array intersections, with a $\times$ at the intersection if a diode is connected there in the OR array.

When the number of input variables is small, a PROM may be more economical to use than a PLA. However, when the number of input variables is large, PLAs often provide a more economical solution than PROMs.

Type:

- Mask-programmable logic arrays: Programmed permanently during manufacturing using photolithography. The mask used in photolithography is expensive, so the use of mask-programmable logic arrays is economically feasible only if a large quantity (typically several thousand or more) is required with the same AND and OR arrays.
- Field-programmable logic arrays (FPLAs): Manufactured blank and programmable by the user. Technology of EPROM, EEPROM, or Flash memory may be used. Typically slower and larger than mask-programmable logic arrays.

ii Programmable Array Logic (PAL)

A programmable array logic (PAL) is similar to a programmable logic array but in which the AND array is programmable and the OR array is fixed. PALs are less expensive than the more general PLAs and easier to program. For this reason, logic designers frequently use PALs to replace individual logic gates when several logic functions must be realized.

A buffer is used because each PAL input must drive many AND gate inputs. When the PAL is programmed, some of the interconnection points are programmed to make the desired connections to the AND gate inputs, represented by X's on the diagram.

When designing with PALs, we must simplify our logic equations and try to fit them into one (or more) of the available PALs. Unlike the more general PLA, the AND terms cannot be shared among two or more OR gates; therefore, each function to be realized can be simplified by itself without regard to common terms. For a given type of PAL, the number of AND terms that feed each output OR gate is fixed and limited. If the number of AND terms in a simplified function is too large, we may be forced to choose a PAL with more gate inputs and fewer outputs.

iii Complex Programmable Logic Devices (CPLDs)

As integrated circuit technology continues to improve, more and more gates can be placed on a single chip. This has allowed the development of complex programmable logic devices (CPLDs). Instead of a single PAL or PLA on a chip, many PALs or PLAs can be placed on a single CPLD chip and interconnected. When storage elements such as flip-flops are also included on the same IC, a small digital system can be implemented with a single CPLD.

Take Xilinx XCR3064XL CPLD for example. This CPLD has four function blocks, each of which has 16 associated macrocells and is a programmable AND-OR array that is configured as a PLA. Each macrocell contains a flip-flop and multiplexers that route signals from the function block to the input-output (I/O) block or to the interconnect array (IA). The IA selects signals from the macrocell outputs or I/O blocks and connects them to function block inputs. The I/O blocks connect the signals from the interior of the CPLD to the bi-directional I/O pins on the IC.

iv Field-Programmable Gate Arrays (FGPAs)

An FPGA is an IC that consists of an array of identical logic cells, also called configurable logic blocks (CLBs) or function generators, with programmable interconnections and surrounded by I/O blocks that connect the CLB signals to IC pins. The user can program the functions realized by each logic cell and the connections between the cells. Each CLB contains a lookup table (LUT), flip-flops, and multiplexers. An n-input LUT is essentially a reprogrammable 2^n-word-$\times$-1-bit ROM that stores the truth table for the n-variable function being generated.

Given a n-variable switching function F and a FGPA with m-input LUTs ($m < n$), 2^{n-m} m-variable subfunctions of F can be obtained using Shannon's expansion of the function and compose back to F with one 2^{n-m}-to-1 multiplexer, which can be composed of $\left(2^{n-m-k+1}-1\right)\ 2^k$-to-1 multiplexers ($k \leq n-m$).

3 Sequential Circuit

I Latches and Flip-Flops

i Clock (CLK) signal

A clock signal is a pulse wave that oscillates between a high state, representing a "one", and a low state, representing a "zero", at a constant frequency. The ratio of the high period to the total period is called the duty cycle.

Clock signals are generated by clock generators, which can be crystal oscillator (very stable, used in CPUs), RC oscillator (simpler and less precise), external clock generator, etc., and can be modified by other components in frequency or phase.

ii Synchronous and asynchronous system

A synchronous system is a digital system where all state changes are coordinated by a single, global clock signal.

An asynchronous systems is a digital system that is not a synchronous system.

iii Latch

A latch is a bistable sequential circuit that store a single bit of information and hold its value until it is updated by new input signals. One of its two states represents a logic 0 and the other represents a logic 1.

For a latch to capture the inputted information and store it is called latch.

iv State variable

A state variable is a binary variable that represents the current state of a sequential circuit.

The state variable stored in a latch is usually denoted as Q and sometimes called the same as the latch.

v Asynchronous, Edge-triggered, and Level-triggered latch

An asynchronous latch (aka a basic latch) is a latch that responds to changes of data inputs all the time.

A level-triggered latch/flip-flop is a latch that responds to changes of data inputs when a signal, called enable (En or E), is at a specific high or low level.

A edge-triggered flip-flop is a latch that responds to changes of data inputs only at the rising or falling edge of a clock signal, called clock (Clk, Ck, or C).

vi Gated or level-sensitive latch

A gated or level-sensitive latch is a level-triggered latch that has an additional enable input signal En which controls whether the latch is transparent (enabled) or hold (disabled). When enable is at active level (1 for active-high gated latch and 0 for active-low gated latch), it is transparent, meaning the it acts as an asynchronous latch of its type; when enable is at inactive level (0 for active-high latch and 1 for active-low latch), the latch is hold (aka retain), meaning the output remains at its previous state regardless of input changes.

Unless otherwise specified, a gated latch is active-high. A active-low gated latch can be implemented by adding an inverter at the enable input. The symbol of an active-low latch in block diagrams is that of an active-high one with an inversion bubble at the enable input right outside of the rectangle.

vii Flip-flop (FF)

The term flip-flop has historically referred generically to both level-triggered and edge-triggered latch. Some modern authors, however, reserve the term flip-flop exclusively for edge-triggered latch and use the term gated latch or simply latch for level-triggered latch. The terms ”edge-triggered”, and ”level-triggered” may be used to avoid ambiguity.

The implementation a type of flip-flop with another type of flip-flop is called conversion.

viii Response time or Delay time

The response time or delay time ϵ of a latch or flip-flop is the time it needs to react to a input change. If an impulse of input lasts less than ϵ, the latch or flip-flop will not react to it.

ix Present state and Next state

When discussing latches and flip-flops, we the term present state $Q = Q(t)$ to denote the state of the Q output of the latch or flip-flop at the time input signal changes, and the term next state $Q^+ = Q(t + \epsilon)$ to denote the state of the Q output after the latch or flip-flop has reacted to the input change and stabilized. The equation with the next state in the LHS and a function of present state and inputs in the RHS is called the next-state equation or characteristic equation of the sequential circuit.

x Moore Machine

A Moore machine can be defined as a 6-tuple $(S, s_0, \Sigma, O, \delta, G)$ consisting of:

- A finite set of states S,
- A start state (also called initial state) s_0 which is an element of S,
- A finite set called the input alphabet Σ
- A finite set called the output alphabet O,
- A transition function $\delta : S \times \Sigma \to S$ mapping a state and the input alphabet to the next state, and
- An output function $G : S \to O$ mapping each state to the output alphabet.

xi Mealy Machine

A Mealy machine can be defined as a 6-tuple $(S, s_0, \Sigma, O, \delta, G)$ consisting of:

- A finite set of states S,
- A start state (also called initial state) s_0 which is an element of S,
- A finite set called the input alphabet Σ
- A finite set called the output alphabet O,
- A transition function $\delta : S \times \Sigma \to S$ mapping each pair of a state and an element of input alphabet to an next state, and
- An output function $G : S \times \Sigma \to O$ mapping each pair of a state and an element of input alphabet to an output alphabet.

xii Metastable state

A metastable state is a state where the state variables in a sequential circuit

- are constants and can theoretically stay as is if no noise occur, and

- transit away to a truly stable state if any noise occurs,

where a truly stable state is a state where the state variables return to the value of them in the state if any noise occurs.

The probability that metastability lasts decays exponentially over time.

Take the sequential circuit consists of two inverters with the output of one being the input of another for example. There are two stable states (excluding metastable state), 0 and 1, and one metastable state, a voltage right in the middle of 0 and 1.

xiii Essential hazard

An essential hazard in an asynchronous circuit is a hazard, which causes glitch but does not affect the final stable state, caused by unequal propagation delays along at least two different paths that originate from the same input, at least one of which in the circuit's feedback paths. These hazards cannot be corrected by adding redundant logic gates and require adjusting the delays in the affected paths to be resolved.

xiv Race condition, race hazard, or race

A race condition, race hazard, or simply race is the condition which the behavior of a system is dependent on the sequence or timing of other uncontrollable events.

A non-critical race is a race which the final stable state is not dependent on the sequence or timing of other uncontrollable events.

A critical race is a race which the final stable state is dependent on the sequence or timing of other uncontrollable events. A critical race is caused by race of state variables and cannot be removed by adding redundant gates or adjusting the delays. To eliminate critical races, we can assign states in a way such that at most one state variable changes at a time, e.g., gray code.

xv (Active-high) S-R Latch

An (asynchronous) (active-high or NOR-gate) set-reset (aka S-R or SR) latch or an (asynchronous) S-R NOR latch has two inputs, set S and reset R, and two outputs, Q and Q', and stores one bit, Q. It consists two NOR gates with S and Q being the inputs and Q' being the output of one NOR gate, and R and Q' being the inputs and Q begin the output of the other NOR gate.

It has a restriction that $SR = 0$.

With present state Q, next state Q^+, and next state of Q' P, the next-state equation of an S-R latch is:

$$Q^+ = (R + (S + Q)')' = R'S + R'Q.$$

$$P = (S + Q^+)' = S'Q^{+\prime}.$$

$$Q^{+\prime} = (R'S + R'Q)' = (R'S)'(R'Q)' = (R + S')(R + Q') = R + S'Q'.$$

$$P = S'R + S'Q'.$$

The truth table is:

S	R	Q	Q^+	$Q^{+\prime}$	P
0	0	0	0	1	1
0	0	1	1	0	0
0	1	0	0	1	1
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	0	1	0
1	1	1	0	1	0

Thus, $P = Q^{+\prime}$ except when $S = R = 1$.

By making $(S, R) = (1, 1)$ don't care combination, we can simplify them as:

$$Q^+ = S + R'Q.$$

$$Q^{+\prime} = P = R + S'Q'.$$

When $(S, R) = (1, 0)$, it sets Q^+ to 1; when $(S, R) = (0, 1)$, it resets Q^+ to 0; when $(S, R) = (0, 0)$, it holds present state; when $(S, R) = (1, 1)$, which is not allowed, the outputs oscillate.

The symbol of an active-high S-R latch in block diagrams is an rectangle with two input lines with label S and R in the rectangle adjacent to it on one long side, output line with label Q in the rectangle adjacent to it at the position opposite to S on the opposite side, and output line with label Q' in the rectangle adjacent to it at the position opposite to R on the opposite side.

xvi Active-low S-R Latch

An (asynchronous) active-low or NAND-gate S-R latch, an (asynchronous) S-R NAND latch, $\overline{\text{SR}}$ latch, or $\overline{\text{S}}$-$\overline{\text{R}}$ latch has two inputs, set $\overline{S}$ and reset $\overline{R}$, and two outputs, Q and Q', and stores one bit, Q. It consists two NAND gates with $\overline{S}$ and Q' being the inputs and Q being the output of one NAND gate, and $\overline{R}$ and Q being the inputs and Q' begin the output of the other NAND gate.

It has a restriction that $\overline{S} + \overline{R} = 1$.

With present state Q, next state Q^+, and next state of Q' P, the next-state equation of an S-R latch is:

$$Q^+ = (\overline{S}(\overline{R}Q)')' = \overline{S}' + \overline{R}Q.$$

$$P = (\overline{R}Q^+)' = \overline{R}' + Q^{+\prime}.$$

$$Q^{+\prime} = (\overline{S}' + \overline{R}Q)' = \overline{S}(\overline{R}Q)' = \overline{S}\overline{R}' + \overline{S}Q'.$$

$$P = \overline{R}' + \overline{S}Q'.$$

The truth table is:

$\overline{S}$	$\overline{R}$	Q	Q^+	$Q^{+\prime}$	P
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	0	0

Thus, $P = Q^{+\prime}$ except when $\overline{S} = \overline{R} = 0$.

By making $(\overline{S}, \overline{R}) = (0, 0)$ don't care combination, we can simplify it as:

$$Q^{+\prime} = P = \overline{R}' + \overline{S}Q'.$$

When $(\overline{S}, \overline{R}) = (0, 1)$, it sets Q^+ to 1; when $(\overline{S}, \overline{R}) = (1, 0)$, it resets Q^+ to 0; when $(\overline{S}, \overline{R}) = (1, 1)$, it holds present state; when $(\overline{S}, \overline{R}) = (0, 0)$, which is not allowed, the outputs oscillate.

The symbol of an active-low S-R latch in block diagrams is an rectangle with two input lines with label $\overline{S}$ and $\overline{R}$ in the rectangle adjacent to it on one long side, output line with label Q in the rectangle adjacent to it at the position opposite to $\overline{S}$ on the opposite side, and output line with label Q' in the rectangle adjacent to it at the position opposite to $\overline{R}$ on the opposite side.

xvii Switch Debouncing with an S-R Latch

When you press or release a physical pushbutton or toggle switch, the conductor contacts don't make or break cleanly but bounce (open-close-open-close rapidly), called switch bounce or contact chatter, which may cause noise transitions in logic instead of one if fed directly into digital system.

To ensure that each physical press or release of a switch produces only one transition in logic, we can connect logic 1 (+V) to a double throw switch, connect the two outputs b and a of the double throw switch to S and R of an S-R latch respectively, connect S and R to pull-down resistors, and connect Q to final output. When switch is switched from a to b, the following sessions happen on the time diagram:

1. switch at a: $S = 0$, $R = 1$, $Q = 0$,
2. bounce at a: $S = 0$, R bounces, $Q = 0$,

3. switch between a and b: $S = R = Q = 0$,
4. bounce at b: S bounces, $R = 0$, $Q = 1$ after delay time of the latch since bounce at b starts (delay time of the latch is longer than the time S bounces from one state to another),
5. switch at b: $S = 1$, $R = 0$, $Q = 1$.

xviii NOR-gate Gated S-R Latch

A NOR-gate gated S-R latch or a gated S-R NOR latch has three inputs, set S, reset R, and enable E, and two outputs, Q and Q', and stores one bit, Q. It consists of an active-high S-R latch with S and R being replaced with the outputs of two AND gates, each of which takes E and the original input as inputs respectively.

It has a restriction that $SR = 0$.

With present state Q, next state Q^+, and next state of Q' P, the next-state equation of a gated S-R latch is:

$$Q^+ = (RE + (SE + Q)')' = (RE)'(SE + Q) = (R' + E')(SE + Q) = SR'E + R'Q + E'Q.$$

By making $(S, R) = (1, 1)$ don't care combination, we can simplify it as:

$$Q^+ = SE + Q(R' + E').$$

$$P = (SE + Q^+)' = (SE + SR'E + R'Q + E'Q)' = (S' + E')(R + Q')(E + Q') = S'RE + S'Q' + E'Q'.$$

$$Q^{+\prime} = (SE + Q(R' + E'))' = (SE)'(Q(R' + E'))' = (S' + E')(Q' + RE) = S'RE + S'Q' + E'Q' = P.$$

By making $(S, R) = (1, 1)$ don't care combination, we can simplify it as:

$$Q^{+\prime} = P = RE + Q'(S + E').$$

When $(S, R, E) = (1, 0, 1)$, it sets Q^+ to 1; when $(S, R, E) = (0, 1, 1)$, it resets Q^+ to 0; when $(S, R) = (0, 0)$ or $E = 0$, it holds present state; when $(S, R, E) = (1, 1, 1)$, which is not allowed, the outputs oscillate; when E changes from 1 to 0 while $(S, R) = (1, 1)$, which is not allowed, a race condition where both inputs to the underlying S-R latch changes from 0 to 1 and the propagation delays of the gates determine whether the latch stabilizes with Q^+ changed or not occurs.

Another restriction of NOR-gate gated S-R latchs is when one or both of S and R go low (e.g. a glitch caused by a statc-0 hazard) near a falling edge of E, the latch may catch the unwanted 0. This is called 0's catching problem.

The symbol of a gated S-R latch in block diagrams is an active-high S-R latch with an additional enable input line between S and R labeld En.

xix NAND-gate Gated S-R Latch

A NAND-gate gated S-R latch or a gated S-R NAND latch has three inputs, set S, reset R, and enable E, and two outputs, Q and Q', and stores one bit, Q. It consists of an active-low S-R latch with $\overline{S}$

and $\overline{R}$ being replaced with the outputs of two NAND gates, each of which takes E and the original input as inputs respectively.

It has a restriction that $SR = 0$.

With present state Q, next state Q^+, and next state of Q' P, the next-state equation of a NAND-gate gated S-R latch is:

$$Q^+ = ((SE)'((RE)'Q)')' = SE + (RE)'Q = SE + Q(R' + E').$$

$$P = ((RE)'Q^{+\prime})' = ((RE)'(SE+Q(R'+E')))' = RE+(SE+Q(R'+E'))' = RE+(SE)'(Q(R'+E'))' = RE+(S'+$$

$$Q^{+\prime} = (SE + Q(R' + E'))' = (SE)'(Q(R' + E'))' = (S' + E')(Q' + RE) = S'Q' + S'RE + E'Q = P.$$

By making $(S, R, E) = (1, 1)$ don't care combination, we can simplify it as:

$$Q^{+\prime} = P = RE + Q'(S + E').$$

When $(S, R, E) = (1, 0, 1)$, it sets Q^+ to 1; when $(S, R, E) = (0, 1, 1)$, it resets Q^+ to 0; when $(S, R) = (0, 0)$ or $E = 0$, it holds present state; when $(S, R, E) = (1, 1, 1)$, which is not allowed, the outputs oscillate; when E changes from 1 to 0 while $(S, R) = (1, 1)$, which is not allowed, a race condition where both inputs to the underlying S-R latch changes from 1 to 0 and the propagation delays of the gates determine whether the latch stabilizes with Q^+ changed or not occurs.

Another restriction of NAND-gate gated S-R latchs is when one or both of S and R go high (e.g. a glitch caused by a statc-1 hazard) near a falling edge of E, the latch may catch the unwanted 1. This is called 1's catching problem.

The symbol of a gated S-R latch in block diagrams is an active-high S-R latch with an additional enable input line between S and R labeld En.

xx (Gated) D Latch or Transparent Latch

A (gated) data (aka D) latch or transparent latch has two inputs, data D and enable E, and two outputs, Q and Q', and stores one bit, Q. It consists of a gated S-R latch with S and R being replaced with D and D' (D through an inverter) respectively. It is called a transparent latch because $Q = D$ when $E = 1$.

The symbol of a D latch in block diagrams is the same as a gated S-R latch with R removed and S changed to D.

xxi (Asynchronous) J-K Latch

An (asynchronous) J-K (aka JK) latch is an improved version of the S-R latch that eliminates the forbidden state. It has two inputs, J and K, and two outputs, Q and Q', and stores one bit, Q. It consists of a gated S-R latch with S and R being renamed as J and K respectively and the enable E inputs to the gates with J and K being the other input being replaced with new Q' and Q feedback inputs respectively.

With present state Q, next state Q^+, and next state of Q' P, the next-state equation of a NOR-gate gated S-R latch, that is a J-K latch with underlying gated S-R latch being NOR-gate gated S-R latch, is:

$$Q^+ = (KQ + (JQ' + Q)')' = (KQ)'(JQ' + Q) = (K' + Q')(JQ' + Q) = JQ' + K'Q.$$

$$P = (JQ' + Q^+)' = (JQ' + K'Q)' = Q^{+\prime} = (J' + Q)(K + Q') = J'K + J'Q' + KQ.$$

With present state Q, next state Q^+, and next state of Q' P, the next-state equation of a NAND-gate gated S-R latch, that is a J-K latch with underlying gated S-R latch being NAND-gate gated S-R latch, is:

$$Q^+ = ((JQ')'((KQ)'Q)')' = JQ' + (KQ)'Q = JQ' + (K' + Q')Q = JQ' + K'Q.$$

$$P = ((KQ)'Q^+)' = (KQ)(J' + Q)(K + Q') = (J' + Q)(K + Q') = Q^{+\prime} = J'K + J'Q' + KQ.$$

When $(J, K) = (1, 0)$, it sets Q^+ to 1; when $(J, K) = (0, 1)$, it resets Q^+ to 0; when $(J, K) = (0, 0)$, it holds present state; when $(J, K) = (1, 1)$, the outputs (Q, Q') become $(1, 0)$ if originally $(0, 1)$ and $(0, 1)$ if originally $(1, 0)$, called toggle; however, if $J = K = 1$ stays longer than the propagation delay, the outputs toggle again, called the race-around problem.

The symbol of a J-K latch in block diagrams is the same as an active-high S-R latch with S and R renamed J and K respectively. Sometimes output Q' is omitted if not used.

xxii Gated J-K Latch

A gated J-K latch is a J-K latch with one additional input, enable E, that is connected to both the gates J and K input to, which makes the two gates both have three inputs.

When $(J, K, E) = (1, 0, 1)$, it sets Q^+ to 1; when $(J, K, E) = (0, 1, 1)$, it resets Q^+ to 0; when $(J, K, E) = (1, 1, 1)$, the outputs toggle with race-around problem; otherwise, it holds present state.

The symbol of a gated J-K latch in block diagrams is a J-K latch with an additional enable input line between J and K labeld En.

xxiii Edge-triggered flip-flops

An edge-triggered flip-flop has the same inputs and outputs of the gated latch of the same type but the enable input E is replaced with clock input Clk. For rising-edge-triggered (aka active-high) flip-flops, when clock is at a rising edge, called active edge, the flip-flop is transparent; otherwise the flip-flop is hold (aka retain). For falling-edge-triggered (aka active-low) flip-flops, when clock is at a falling edge, called active edge, the flip-flop is transparent; otherwise the flip-flop is hold (aka retain).

The symbol of a rising-edge-triggered flip-flop in block diagrams is a gated latch of the same type with En replaced with a small arrowhead (wedge shape) with its opening facing outwards from the rectangle and its two ends on the side of the rectangle representing the clock input.

The symbol of a falling-edge-triggered flip-flop in block diagrams is a rising-edge-triggered flip-flop of the same type with an inversion bubble outside of the rectangle at the clock input.

xxiv (Edge-triggered) S-R flip-flop (JKFF)

For active-high S-R flip-flops, $(S, R) = (1, 1)$ is not allowed. For active-low S-R flip-flops, $(\overline{S}, \overline{R}) = (0, 0)$ is not allowed.

xxv (Edge-triggered) D flip-flop (DFF)

When the context is clear, flip-flops (FFs) may be used to refer to D flip-flops (DFFs).

xxvi (Edge-triggered) J-K flip-flop (JKFF)

When $(J, K) = (1, 0)$ at an active edge, it sets Q^+ to 1; when $(J, K) = (0, 1)$ at an active edge, it resets Q^+ to 0; when $(J, K) = (1, 1)$ at an active edge, the outputs toggle exactly once, without race-around problem; otherwise, it holds present state.

xxvii (Edge-triggered) T flip-flop (TFF)

A toggle (aka T) flip-flop has two inputs, toggle T and clock C, and two outputs, Q and Q'. If $T = 1$ at an active edge, the outputs (Q, Q') become $(1, 0)$ if originally $(0, 1)$ and $(0, 1)$ if originally $(1, 0)$, called toggle. Otherwise it is hold.

With present state Q, next state Q^+, the next-state equation of a T flip-flop at an active edge is:

$$Q^+ = T'Q + TQ'.$$

It can be implemented with a J-K flip-flop with inputs J and K both connected to T.

Alternatively, it can be implemented with a D flip-flop with input D connect to the output of a XOR gate with inputs T and Q.

The symbol of a T flip-flop in block diagrams is a D flip-flop with D replaced with T.

xxviii Pulse-triggered flip-flops

The term pulse-triggered flip-flop refers to the implementation that AND together clock signal and a slightly delayed version of the inversion of the clock siganl and feed it to a gated latch such that the flip-flop is transparent only on a short period after a rising edge, or AND together the inversion of a clock signal and a slightly delayed version of the clock siganl and feed it to a gated latch such that the flip-flop is transparent only on a short period after a falling edge.

xxix Master-slave flip-flops

The term master-slave flip-flop refers to a particular implementation that uses two gated latches of the same type in such a way that the flip-flop outputs only change on a clock edge.

The master is the latch that takes the data inputs and the slave is the latch that takes outputs of the master as inputs and outputs the final outputs of the master-slave flip-flop.

The master-slave flip-flops need a time where the data inputs are stable before an active edge long enough for the master to latch, called setup time, and a time where the data inputs are stable after

an active edge long enough for the outputs of master to transfer to the slave, called hold time. If data inputs change during the union of setup time and hold time around an active edge, the behavior is unpredictable.

For rising-edge-triggered flip-flops, the enable signal of the master is the inverse of the clock signal and the enable signal of the slave is the clock signal. For falling-edge-triggered flip-flops, the enable signal of the master is the clock signal and the enable signal of the slave is the inverse of the clock signal.

xxx Master-slave S-R flip-flop (SRFF)

Let the inputs and outputs of the master be with subscript $_1$ and those of the slave be with subscript $_2$. In an active-high or active-low master-slave S-R flip-flop, Q_1 and Q_1' are connected to S_2 and R_2 respectively, and Q_2 and Q_2' are the final outputs of it.

xxxi Master-slave D flip-flop

Let the inputs and outputs of the master be with subscript $_1$ and those of the slave be with subscript $_2$. In a master-slave D flip-flop, Q_1 is connected to D_2, Q_1' is discarded, and Q_2 and Q_2' are the final outputs of it.

xxxii Master-slave J-K flip-flop (JKFF)

Let the inputs and outputs of the master be with subscript $_1$ and those of the slave be with subscript $_2$. In a master-slave J-K flip-flop, Q_1 and Q_1' are connected to J_2 and K_2 respectively, and Q_2 and Q_2' are the final outputs of it.

xxxiii Flip-flops with asynchronous clear and preset

Some flip-flops has asynchronous inputs that change the stored state of Q at any time.

- Active-high asynchronous clear Clr sets the stored state of Q to 0 if Clr=1, called active, and does nothing if Clr=0. In block diagrams, with the side of the symbol of the flip-flop with input lines at the buttom, Clr is an input line on the left side labeled Clr.
- Active-high asynchronous clear Pre sets the stored state of Q to 1 if Pre=1, called active, and does nothing if Pre=0. In block diagrams, with the side of the symbol of the flip-flop with input lines at the buttom, Pre is an input line on the right side labeled Pre.
- Active-low asynchronous clear ClrN sets the stored state of Q to 0 if Clr=0, called active, and does nothing if Clr=1. In block diagrams, with the side of the symbol of the flip-flop with input lines at the buttom, ClrN is an input line with inversion bubble on the left side labeled ClrN or Clr.
- Active-low asynchronous clear PreN sets the stored state of Q to 1 if Pre=0, called active, and does nothing if Pre=1. In block diagrams, with the side of the symbol of the flip-flop with input lines at the buttom, PreN is an input line with inversion bubble on the right side labeled PreN or Pre.

- If both Clr (ClrN) and Pre (PreN) are active at the same time, the behavior is unpredictable.

xxxiv Flip-flops with clock enable

In synchronous digital systems, the flip-flops are usually driven by a common clock so that all state changes occur at the same time in response to the same clock edge. If we want a flip-flop to hold existing data during an active edge even though the data input to the flip-flops may be changing, we can use a flip-flop with clock enable (CE) input.

A flip-flop with clock enable (CE) input has an additional input CE and works as a normal flip-flop if CE=1 and hold if CE=0.

One method to implement it is by gating the clock, that is, replace the original clock input with the output of an AND gate with clock input and CE being the inputs. However, there are two potential problem. First, gate delays may cause the clock to arrive at some flip-flops at different times than at other flip-flops, resulting in a loss of synchronization. Second, if CE changes at the wrong time, the flip-flop may trigger due to the change in CE instead of due to the change in the clock input.

Rather than gating the clock, a better method to implement it is by replacing each original data input with the output a 2-to-1 MUX with CE being the select input, original input being the data input selected when CE=1, and 0 being the data input selected when CE=0. Because there is no gate in the clock line, this cannot cause a synchronization problem.

A flip-flop with CE is called with the original name with -CE sufficed to the original type name, e.g., D-CE flip-flop.

II Registers and Counters

i Register

A register can load data inputs in parallel and output stored data in parallel. A n bit register consists of n edge-triggered D flip-flops, each with D being data input and Q being data output. The flip-flops usually share common clock input and asynchronous clear and/or preset inputs. If using gated clocks, the clock is gated with a common signal called load; if using clock enable, the flip-flops share a common clock enable input called load. When load=1 at an active edge, the data inputs are loaded into the register; otherwise, the register holds present state.

The symbol of a register in block diagrams is one underlying flip-flop symbol with bus lines for data inputs and outputs, and load is labeled as Load at the clock enable input or the gated clock input.

ii Data Transfer Between Registers with Tri-state Buffers

To transfer data from a source register to a destination register, we can connect the outputs of the source register to the inputs of the destination register through tri-state buffers, and connect the enable inputs of the tri-state buffers to a common enable signal En. When $En = 1$, the tri-state buffers are enabled and the data outputs of the source register are transferred to the data inputs of the destination register; when $En = 0$, the tri-state buffers are disabled and the data inputs of the destination register are disconnected from the data outputs of the source register.

If multiple registers share a common data bus, only one register should enable its tri-state buffers at a time to avoid bus contention.

iii Data Transfer Between Registers with Multiplexers

To transfer data from source registers to a destination register, we can connect the outputs of the source registers to data inputs of a multiplexer and connect the output of the multiplexer to the data inputs of the destination register. When the select input select a source register and clock enable/load input of the destination register is active at an active edge, the data outputs of the selected source register are transferred to the data inputs of the destination register.

iv Accumulator

An accumulator is a register that stores intermediate results of arithmetic and logic operations in a digital system. It is commonly used in arithmetic logic units (ALUs) of computers and digital signal processors (DSPs) to hold the results of calculations.

Take n-bit parallel adder with accumulator for example. It consists of an n-bit parallel adder and an n-bit register. For the full adder of each cell, the addend input is connected to the data output of the associated flip-flop in the accumulator register, the augend input is connected to the corresponding bit of the external n-bit augend input, and the sum output is connected to the data input of the associated flip-flop in the accumulator register. The addition takes place each time load=1 at an active edge.

Before addition, we should let the register hold the addend. This can be accomplished in several ways. The easiest way is to first clear the accumulator using the asynchronous clear inputs on the flip-flops, and then put the addend on the augend input. Another way is to add a multiplexer before the data input of each flip-flop in the accumulator register to select between the external data input and the sum output of the adder.

v Shift Register

A shift register is a sequential circuit consists of a chain of edge-triggered D flip-flops that stores data and shifts by one bit when certain control inputs are at a certain combination at an active edge, optionally allowing serial or parallel input and output.

- A shift toward least significant bit (LSB) is called a right shift.
- A shift toward most significant bit (MSB) is called a left shift.

The state variable representing the $(i+1)$-th least significant bit is denoted as Q_i.

vi Serial-In Serial-Out (SISO) Shift Register

A serial-in serial-out (SISO) shift register is a type of shift register where data is shifted in and out one bit per shift. An n-bit SISO shift register consists of a chain of n edge-triggered D flip-flops. The output Q of each flip-flop is connected to the input D of the next flip-flop, with the first flip-flop in the chain receiving the serial input (SI) and the last flip-flop in the chain providing the serial output (SO) instead. The flip-flops share common clock input and clock enable/load input called shift.

When shift=1 at an active edge, the register shifts a bit; otherwise, the register holds its present state.

vii Serial-In Parallel-Out (SIPO) Shift Register

A serial-in parallel-out (SIPO) shift register is a type of shift register that allows data to be retrieved in parallel. An n-bit SIPO shift register consists of an n-bit SISO register with additional $n-1$ outputs connected to the outputs Q of the flip-flops except the last one. Those outputs together with the original SO compose the parallel output of the register.

viii Parallel-In Serial-Out (PISO) Shift Register

A parallel-in serial-out (PISO) shift register is a type of shift register that allows data to be loaded in parallel. An n-bit PISO shift register consists of a chain of n edge-triggered D flip-flops, each of which associated with one 4-to-1 multiplexer (or other logically equivalent implementations). The data input D of each flip-flop is connected to the output of the associated multiplexer. The two select inputs of each multiplexer are connected to common shift enable input Sh and load enable input L. The four data inputs of each multiplexer are:

- the output of the associated flip-flop, selected when $(Sh, L) = (0, 0)$, called hold,
- the corresponding bit of the parallel input (PI), selected when $(Sh, L) = (0, 1)$, called load, and
- the output of the previous flip-flop in the chain with the first flip-flop in the chain receiving the serial input (SI) instead, selected when $(Sh, L) = (1, 0)$ or $(1, 1)$, called shift.

When $(Sh, L) = (1, 0)$ or $(1, 1)$ at an active edge, the register shifts a bit; when $(Sh, L) = (0, 1)$ at an active edge, the register loads parallel input; otherwise, the register holds its present state.

ix Parallel-In Parallel-Out (PIPO) Shift Register

A parallel-in parallel-out (PIPO) shift register is a type of shift register that allows data to be loaded and retrieved in parallel. An n-bit PIPO shift register consists of an n-bit PISO register with additional $n-1$ outputs connected to the outputs Q of the flip-flops except the last one. Those outputs together with the original SO compose the parallel output of the register.

x Parallel-In Serial-Out (PISO) Bidirectional Shift Register

A bidirectional shift register is a type of shift register that allows data to be shifted in both directions. An n-bit parallel-in serial-out (PISO) bidirectional shift register consists of a chain of n edge-triggered D flip-flops, each of which associated with one 4-to-1 multiplexer (or other logically equivalent implementations). The data input D of each flip-flop is connected to the output of the associated multiplexer. The two select inputs of each multiplexer are connected to common shift/load input S_1 and shift direction left/right or load/hold select input S_0. The four data inputs of each multiplexer are:

- the output of the associated flip-flop, selected when $(S_1, S_0) = (0, 0)$, called hold,
- the corresponding bit of the parallel input (PI), selected when $(S_1, S_0) = (0, 1)$, called load,

- the output of the adjacent flip-flop storing the less significant bit in the chain with the flip-flop storing the least significant bit in the chain receiving the serial input (SI) instead, selected when $(S_1, S_0) = (1, 0)$, called shift right, and
- the output of the adjacent flip-flop storing the more significant bit in the chain with the flip-flop storing the most significant bit in the chain receiving the serial input (SI) instead, selected when $(S_1, S_0) = (1, 1)$, called shift left.

Two serial outputs (SO) are connected to the outputs Q of the flip-flops storing the least and most significant bits respectively.

When $(S_1, S_0) = (1, 0)$ at an active edge, the register shifts a bit right; when $(S_1, S_0) = (1, 1)$ at an active edge, the register shifts a bit left; when $(S_1, S_0) = (0, 1)$ at an active edge, the register loads parallel input; otherwise, the register holds its present state.

xi Serial-In Serial-Out (SISO) Bidirectional Shift Register

An n-bit serial-in serial-out (SISO) bidirectional shift register is an n-bit PISO register with each bit of the parallel input connected to each multiplexer as data input replaced with the output of the associated flip-flop.

When $(S_1, S_0) = (1, 0)$ at an active edge, the register shifts a bit right; when $(S_1, S_0) = (1, 1)$ at an active edge, the register shifts a bit left; otherwise, the register holds its present state.

xii Parallel-In Parallel-Out (PIPO) Bidirectional Shift Register

An n-bit parallel-in parallel-out (PIPO) bidirectional shift register is an n-bit PISO register additional $n - 2$ outputs connected to the outputs Q of the flip-flops except the flip-flops storing the least and most significant bits. Those outputs together with the original two SOs compose the parallel output of the register.

xiii Serial-In Parallel-Out (SIPO) Bidirectional Shift Register

An n-bit serial-in parallel-out (SIPO) bidirectional shift register is an n-bit SISO register additional $n-2$ outputs connected to the outputs Q of the flip-flops except the flip-flops storing the least and most significant bits. Those outputs together with the original two SOs compose the parallel output of the register.

xiv Counter

A counter is a sequential circuit whose state repeats every fixed finite number of active edges, called count cycle. The sequence of states in the cycle drawn as a circle with arrows pointing to next state for each state is called the transition cycle of the counter.

xv Synchronous Counter

A synchronous counter is a counter consists of flip-flops where all flip-flops are enabled synchronously.

xvi Shift Register Counter

A shift register counter is a type of synchronous counter consists of a shift register with some or all of the outputs Q of the flip-flops connected to the inputs of a combinational circuit with one output and the output of the combinational circuit connected to the input D of the first flip-flop in the chain.

xvii Ring Counter

A ring counter is a type of shift register counter consists of a shift register with the output Q of the last flip-flop in the chain connected to the input D of the first flip-flop in the chain. The bits stored circulate around the register. The count cycle of an n-bit ring counter is n.

xviii Johnson counter, Twisted Ring Counter, or Mod-2n Counter

A Johnson counter, twisted ring counter, or mod-2n counter is a type of shift register counter consists of a shift register with the output Q' of the last flip-flop in the chain connected to the input D of the first flip-flop in the chain. The legal states of a Johnson counter are those consist of at most one contiguous 0 bits block and at most one contiguous 1 bits block. For an n-bit Johnson counter, number of legal states is $2n$, and the count cycle is $2n$ if starting with a legal state.

xix Linear (Feedback) Shift Register (LFSR) Counter

A linear (feedback) shift register (LFSR) counter consists is a type of shift register counter where the function realized by the combination circuit whose output is connected to the input D of the first flip-flop in the chain is linear.

For each $n \in \mathbb{N}$, there exists a linear n-bit shift register counter whose count cycle is $2^n - 1$, in which all states except the all 0's state are included.

xx Synchronous Binary Up Counter

An n-bit synchronous binary up counter consists of n T flip-flops sharing a common clock input signal with the one storing the $(k+1)$-th least significant bit labeled with subscript $_k$. The data input T_0 is connected to a constant 1 source, and the data input T_k for $k \in \mathbb{N}$ is connected to

$$\prod_{i=0}^{k-1} Q_i,$$

which is implemented using AND gates as

$$T_1 = Q_0$$

$$T_k = T_{k-1} Q_{k-1}, \quad k \in \mathbb{N} \wedge 1 < k < n.$$

Its count cycle is 2^n and its state is a binary number incrementing from 0 to 2^n one-by-one per cycle.

xxi Synchronous Down Counter

An n-bit synchronous down counter consists of n T flip-flops sharing a common clock input signal with the one storing the $(k+1)$-th least significant bit labeled with subscript $_k$. The data input T_0 is

connected to a constant 0 source, and the data input T_k for $k \in \mathbb{N}$ is connected to

$$\prod_{i=0}^{k-1} Q_i',$$

which is implemented using AND gates as

$$T_1 = Q_0'$$

$$T_k = T_{k-1} Q_{k-1}', \quad k \in \mathbb{N} \wedge 1 < k < n.$$

Its count cycle is 2^n and its state is a binary number decrementing from 2^n to 0 one-by-one per cycle.

xxii Synchronous Up/Down (U/D) Counter

An n-bit synchronous up/down (U/D) counter with separate up U and down D inputs consists of n T flip-flops sharing a common clock input signal with the one storing the $(k+1)$-th least significant bit labeled with subscript $_k$. The data input T_k is connected to

$$U \prod_{i=0}^{k-1} Q_i + D \prod_{i=0}^{k-1} Q_i', \quad k \in \mathbb{N}_0 \wedge k < n$$

which is implemented using AND and OR gates as

$$U_0 = U$$

$$U_k = U_{k-1} Q_{k-1}, \quad k \in \mathbb{N} \wedge k < n$$

$$D_0 = D$$

$$D_k = D_{k-1} Q_{k-1}', \quad k \in \mathbb{N} \wedge k < n$$

$$T_k = U_k + D_k, \quad k \in \mathbb{N} \wedge k < n.$$

An n-bit synchronous up/down (U/D) counter with up/down input M indicating up when $M = 1$ and down when $M = 0$ consists of n T flip-flops optionally with clock enable CE to hold when CE=0 sharing a common clock input signal with the one storing the $(k+1)$-th least significant bit labeled with subscript $_k$. The data input T_0 is connected to a constant 1 source, and the data input T_k for $k \in \mathbb{N}$ is connected to

$$M \prod_{i=0}^{k-1} Q_i + M' \prod_{i=0}^{k-1} Q_i', \quad k \in \mathbb{N}_0 \wedge k < n$$

which is implemented using AND and OR gates as

$$U_1 = M$$

$$U_k = U_{k-1} Q_{k-1}, \quad k \in \mathbb{N} \wedge 1 < k < n$$

$$D_1 = M'$$

$$D_k = D_{k-1} Q_{k-1}', \quad k \in \mathbb{N} \wedge 1 < k < n$$

$$T_k = U_k + D_k, \quad k \in \mathbb{N} \wedge 1 < k < n.$$

xxiii Loadable Counter

A counter that consists of T flip-flops mentioned above can be adjusted to be loada le from a parallel input (PI) with an associated 2-to-1 multiplexer for each T flip-flop and a common load input as select input for all multiplexers. The original input T to each flip-flop is connected instead to the data input of the associated multiplexer which is selected when load=0, the output of each multiplexer is connected to the input T of the associated flip-flop, and the other data input to each multiplexer, which is selected when load=1, is connected to the corresponding bit of the parallel input.

When load=1 at an active edge, the loadable counter loads parallel input; when load=0 at an active edge, the loadable counter works as the underlying counter; otherwise, the loadable counter holds its present state.

xxiv State-Transition Table or Transition Table

A state-transition table or transition table of a sequential circuit is in the form of a truth table with output variables replaced with next-state variables (labeled with superscript $^+$) and sometimes also output variables and input variables replaced with present-state variables and sometimes also input variables.

xxv Next-state Map

A next-state map is the K-map of a next-state variable as a function of the present-state variables and sometimes also input variables of the sequential circuit.

xxvi Design of General Synchronous Counter with T Flip-Flops

To design a synchronous counter consists of n T flip-flops given arbitrary transition table of m rows of distinct present states Q_k's and corresponding next states Q_k^+'s specified, where the other $2^n - m$ states are don't care states, the following procedure can be used:

1. For each Q_k^+, draw a next-state map.
2. For each 0 and 1 cell in each map, convert the value Q_k^+ in it to T_k given by $T_k = Q_k \oplus Q_k^+$. These maps are called T input (K-)maps and are the same as the next-state maps with $Q_k = 1$ half complemented.
3. Realize all inputs T with combination circuits with a subset of all outputs Q and Q', constant 1 and 0 sources, and all external control inputs as inputs.

xxvii Design of General Synchronous Counter with D Flip-Flops

To design a synchronous counter consists of n D flip-flops given arbitrary transition table of m rows of distinct present states Q_k's and corresponding next states Q_k^+'s specified, where the other $2^n - m$ states are don't care states, the following procedure can be used:

1. For each Q_k^+, draw a next-state map.
2. For each 0 and 1 cell in each map, convert the value Q_k^+ in it to D_k given by $D_k = Q_k^+$. These maps are called D input (K-)maps and are exactly the same as the next-state maps.

3. Realize all inputs D with combination circuits with a subset of all outputs Q and Q', constant 1 and 0 sources, and all external control inputs as inputs.

xxviii Design of General Synchronous Counter with S-R Flip-Flops

To design a synchronous counter consists of n S-R flip-flops given arbitrary transition table of m rows of distinct present states Q_k's and corresponding next states Q_k^+'s specified, where the other $2^n - m$ states are don't care states, the following procedure can be used:

1. For each Q_k^+, draw a next-state map.
2. For each 0 and 1 cell in each map, convert the value Q_k^+ in it to S_k and R_k respectively given by the truth table below where X denotes don't care:

Q_k	Q_k^+	S_k	R_k
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

   These maps are called S input (K-)maps and R input (K-)maps. S input maps are the same as the next-state maps with 1's in $Q_k = 1$ half replaced with X. R input maps are the same as the next-state maps with 0's in $Q_k = 0$ half replaced with X and all cells complemented.
3. Realize all inputs S and R with combination circuits with a subset of all outputs Q and Q', constant 1 and 0 sources, and all external control inputs as inputs.

xxix Design of General Synchronous Counter with J-K Flip-Flops

To design a synchronous counter consists of n J-K flip-flops given arbitrary transition table of m rows of distinct present states Q_k's and corresponding next states Q_k^+'s specified, where the other $2^n - m$ states are don't care states, the following procedure can be used:

1. For each Q_k^+, draw a next-state map.
2. For each 0 and 1 cell in each map, convert the value Q_k^+ in it to J_k and K_k respectively given by the truth table below where X denotes don't care:

Q_k	Q_k^+	J_k	K_k
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

These maps are called J input (K-)maps and K input (K-)maps. J input maps are the same as the next-state maps with all cells in $Q_k = 1$ half replaced with X. K input maps are the same as the next-state maps with all cells in $Q_k = 0$ half replaced with X and $Q_k = 1$ half complemented.

3. Realize all inputs J and K with combination circuits with a subset of all outputs Q and Q', constant 1 and 0 sources, and all external control inputs as inputs.