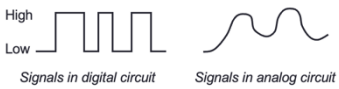
Boolean Algebra and Sequential Logic

Digital Circuits: Two voltage levels

- High/true/1/asserted
- Low/false/0/deasserted



Signals in digital circuit

Signals in analog circuit

Advantages of digital circuits over analog circuits

- More reliable (simpler circuits, less noise-prone)
- Specified accuracy (determinable)
- Abstraction can be applied using simple mathematical model
- Ease design, analysis and simplification of digital circuit

Precedence of Operators: Not (') > **And (·) > Or (+)**

Laws of Boolean Algebra

Identity laws	
A + 0 = 0 + A = A	A · 1 = 1 · A = A
Inverse/complement laws	
A + A' = A' + A = 1	A · A' = A' · A = 0
Commutative laws	
A + B = B + A	A · B = B · A
Associative laws *	
A + (B + C) = (A + B) + C	A · (B · C) = (A · B) · C
Distributive laws	
A · (B + C) = (A · B) + (A · C)	A + (B · C) = (A + B) · (A + C)

Idempotency	
X + X = X	X · X = X
One element / Zero element	
X + 1 = 1 + X = 1	X · 0 = 0 · X = 0
Involution	
(X')' = X	
Absorption 1	
X + X·Y = X	X·(X + Y) = X
Absorption 2	
X + X'·Y = X + Y	X·(X' + Y) = X·Y
De Morgans' (can be generalised to more than 2 variables)	
(X + Y)' = X' · Y'	(X · Y)' = X' + Y'
Consensus	
X·Y + X'·Z + Y·Z = X·Y + X'·Z	(X+Y)·(X'+Z)·(Y+Z) = (X+Y)·(X'+Z)

Duality

- **duality** → if the AND/OR operators and identity elements 0/1 are interchanged in a boolean equation, it remains valid
- e.g. the dual equation of $a + (b \cdot c) = (a + b) \cdot (a + c)$ is $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
- e.g. if $x + 1 = 1$ is valid, then its dual $x \cdot 0 = 0$ is also valid.

Function

- boolean functions: e.g. $F1(x, y, z) = x \cdot y \cdot z'$
  - to prove that $F1 = F2$: use truth table
- **complement function** → given boolean function F, the complement of F, denoted F', is obtained by interchanging 1 with 0 in the function's output values.

Standard Forms: Every Boolean expression can be expressed in SOP or POS form.

- Sum-of-Products (SOP)
- Product-of-Sums (POS)

Literals: A Boolean var. on its **own** or in its **complemented** form**, i.e.** x, x', y, y'

Product term: A single literal or a logical product **(AND)** of several literals

Examples: (1) x, (2) x·y·z', (3) A'·B, (4) A·B, (5) d·g'·v·w

Sum term: A single literal or a logical sum of several literals

Examples: (1) x, (2) x+y+z', (3) A'+B, (4) A+B

SOP expression: A product term or a logical sum of several product terms

Examples: (1) x, (2) x + y·z', (3) x·y' + x'·y·z, (4) A·B + A'·B'

POS expression: A sum term or a logical product **(AND)** of several sum terms

Examples: (1) x, (2) x·(y+z'), (3) (x+y')·(x'+y+z)

Conversion

$$F(x, y, z) = \Sigma m(1,4,5,6,7) = \Pi M(0,2,3)$$

In truth table: $F' = m0 + m2 + m3$

Therefore

$$F = (m0 + m2 + m3)' = m0' \cdot m2' \cdot m3'$$
$$= M0 \cdot M2 \cdot M3$$

Minterm and Maxterm: In general, with ***n*** variables we have up to $\mathbf{2^n}$ minterms and $\mathbf{2^n}$ maxterms.

Minterm of *n* variables is a product term that contains *n* literals from **all** the variables.

Example: On 2 variables x and y, the minterms are: x'·y', x'·y, x·y' and x·y

Maxterm of *n* variables is a sum term that contains *n* literals from **all** the variables.

Example: On 2 variables x and y, the maxterms are: x'+y', x'+y, x+y' and x+y

x	y	Minterms		Maxterms	
		Term	Notation	Term	Notation
0	0	x'·y'	m0	x+y	M0
0	1	x'·y	m1	x+y'	M1
1	0	x·y'	m2	x'+y	M2
1	1	x·y	m3	x'+y'	M3

Min: NOT=0
Max: NOT=1

Canonical/normal form: a unique form of representation.

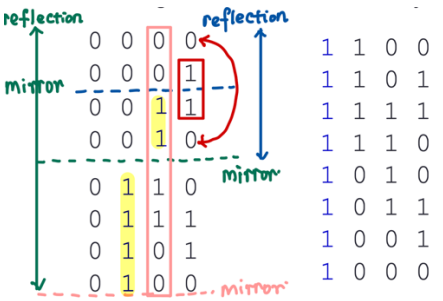
- Sum-of-minterms = Canonical sum-of-products
- Product-of-maxterms = Canonical product-of-sums

Gray Code

- Unweighted (not an arithmetic code)
- Only a single bit change from one code value to the next.
- Not restricted to decimal digits: *n* bits⇒2^n values.
- Good for error detection.

Decimal	Binary	Gray Code	Decimal	Binary	Gray code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Generate Gray Code

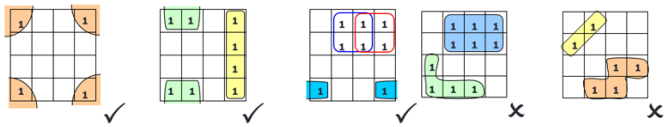


K-Map

- Each cell in ***n***-variable K-map has ***n*** adjacent neighbours
- Arrangement ensures that minterms of adjacent cells differ by only *ONE literal*. [Gray Code]
- There is wrap-around in the K-map
- We can use K-map to simplify SOP and POS expression

Group: each valid grouping of adjacent cells containing '1' corresponds to a simpler product term

- group must have size in powers of 2
- grouping 2^n adjacent cells **eliminates** n variables
- Select as few groups as possible to cover all the cells (minterms) of the function



Implicant: a product term that could be used to cover minterms of the function

Prime implicant (PI): a product term obtained by combining the *maximum possible number of minterms* from *adjacent* squares in the map. (biggest grouping possible.)

- Note: Group consist of only X is <u>not</u> considered as PI

Essential prime implicant (EPI): a prime implicant that includes at least one minterm that is not covered by any other prime implicant.

Don't-care conditions: In certain problems, some outputs are not specified or are invalid. Hence, these outputs can be either '1' or '0'⇒could be chosen to be <u>either</u> '1' or '0', depending on which choice results in a simpler expression.

- When select EPIs, if **all** the "not covered minterm" in the EPI are don't care condition, then we will **not** select it.

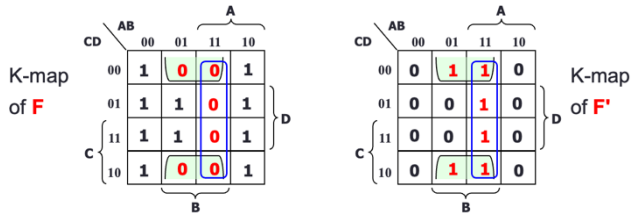
Tips

- larger group⇔fewer literals in the resulting product term
- fewer groups⇔fewer product terms in simplified SOP expression

<u>Finding Simplified POS Expression</u>

Simplified **POS** expression can be obtained by grouping the maxterms (i.e. 0s) of the given function.

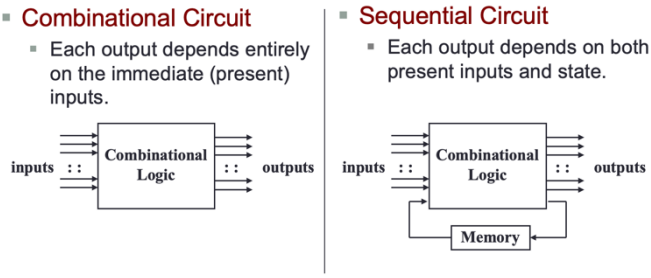
$$F = \Sigma m(0,1,2,3,5,7,8,9,10,11)$$



$$F' = B \cdot D' + A \cdot B$$

$$F = (B \cdot D' + A \cdot B)' = (B \cdot D')' \cdot (A \cdot B)' = (B' + D) \cdot (A' + B')$$

Two classes of logic circuits



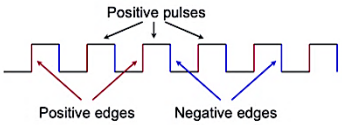
Memory element: a device which can remember value indefinitely, or change value on command from its inputs.

Command (at time *t*)	*Q(t)*	*Q(t+1)*
Set	X	1
Reset	X	0
Memorise / No Change	0	0
	1	1

Q(t) or ***Q***: current state
Q(t+1) or ***Q*⁺**: next state

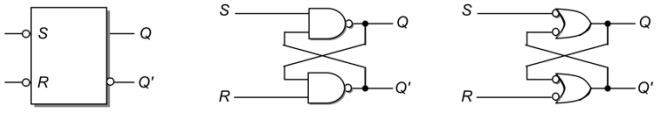
<u>Two types of triggering/activation</u>

- Pulse-triggered
  - Latches
  - ON = 1, OFF = 0
- Edge-triggered
  - Flip-flops
  - Positive edge-triggered
    (ON = from 0 to 1; OFF = other time)
  - Negative edge-triggered
    (ON = from 1 to 0; OFF = other time)



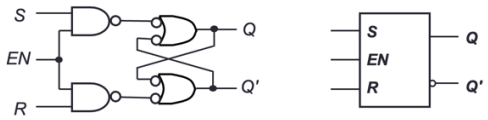
***S-R* Latch**

R	S	$Q(t+1) = S + R' \cdot Q$
High	Low	Low (**R**eset)
Low	High	High (**S**et)
Low	Low	$Q(t)$ No change
High	High	Invalid, Outputs *Q* and *Q*′ are both LOW

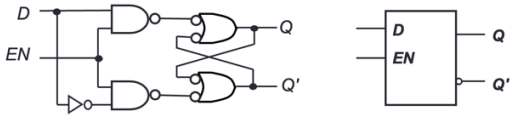


(a) Block diagram. (b) Circuit. (c) Circuit.

Gated *S-R* Latch: *S-R* latch + *enable input* (*EN*) and 2 NAND gates→gated *S-R* latch. (Outputs change when *EN* is high)



Gated *D* Latch: Changes when EN is high



EN	D	$Q(t+1) = D$
High (1)	High (1)	Low (0) Reset
High (1)	Low (0)	High (1) Set
Low (0)	X	$Q(t)$ No change

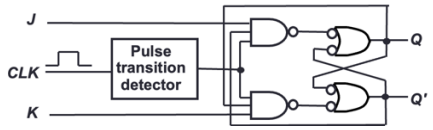
Flip-flops: synchronous bistable devices.

- Output changes state at a specified point on a triggering input called the **clock**.
- Change state either at the **positive (rising) edge**, or at the **negative (falling) edge** of the clock signal.

***S-R* flip-flop:** On the triggering edge of the clock pulse

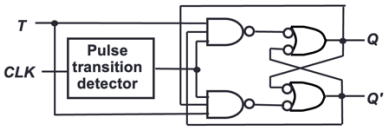
***D* flip-flop:** Single input *D* (data). On the triggering edge of the clock pulse

***J-K* flip-flop:** *Q* and *Q*′ are fed back to the pulse-steering NAND gates. **[No invalid state]**



J	K	$Q(t+1) = J \cdot Q' + K' \cdot Q$
High	Low	High (**S**et)
Low	High	Low (**R**eset)
Low	Low	$Q(t)$ No change
High	High	$Q(t)'$ [Toggle]

***T* flip-flop:** Single input version of the *J-K* flip-flop, formed by tying both inputs together.

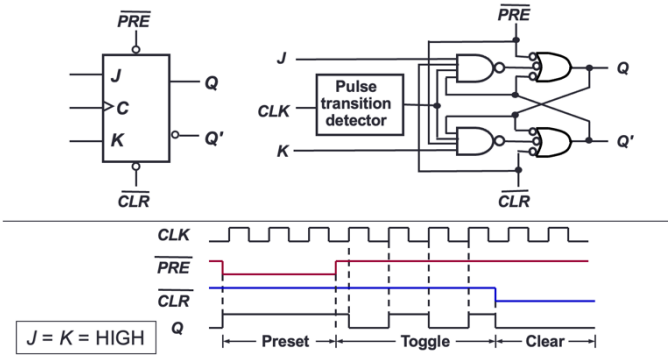


T	$Q(t+1) = T \cdot Q' + T' \cdot Q$
Low	$Q(t)$ No change
High	$Q(t)'$ [Toggle]

Asynchronous Inputs: *S-R*, *D* and *J-K* inputs are **synchronous** inputs, as data on these inputs are transferred to the flip-flop's output only on the triggered edge of the clock pulse.

- Affect the state of the flip-flop independent of the clock; example: *preset* (*PRE*) and *clear* (*CLR*) [or *direct set* (*SD*) and *direct reset* (*RD*)].
- When *PRE*=HIGH, *Q* is immediately set to HIGH.
- When *CLR*=HIGH, *Q* is immediately cleared to LOW.
- Flip-flop in normal operation mode when both *PRE* and *CLR* are LOW.

[Example] A *J-K* flip-flop with active-low PRESET and CLEAR asynchronous inputs.



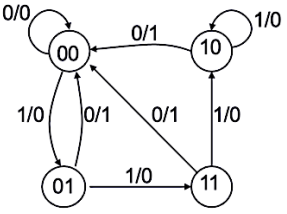
Sequential Circuit Analysis [*Characteristic tables*]

State Table: *m* flip-flops and *n* inputs → 2^{m+n} rows.

State diagram: *m* flip-flops → up to 2^m states

- Each state is denoted by a circle.
- Each arrow denotes a **transition** of the sequential circuit
- A label of the form *a*/*b* is attached to each arrow where *a* (if there is one) denotes the **inputs** while *b* (if there is one) denotes the **outputs** of the circuit in that transition. [Input/Output]

Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
AB	A^+B^+	A^+B^+	*y*	*y*
00	00	01	0	0
01	00	11	1	0
10	00	10	1	0
11	00	10	1	0



Concept

1. A state is called a **sink** if once the circuit enters this state, it never moves out of that state

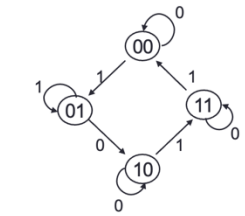
2. A circuit is **self-correcting** if for some reason the circuit enters into any unused (invalid) state, it is able to transit to a valid state after a finite number of transitions.

Sequential Circuit Design [*Excitation tables*]

- Description of circuit behavior [state diagram/ table]
- Derive the state table.
- Perform state reduction if necessary.
- Perform state assignment.
- Determine number of flip-flops and label them.
- Choose the type of flip-flop to be used.
- Derive circuit excitation and output tables from state table
- Derive circuit output functions and flip-flop input functions.
- Draw the logic diagram.

[Example]

- Circuit state/excitation table, using *JK* flip-flops.



Present State	Next State x=0	Next State x=1
AB	A^+B^+	A^+B^+
00	00	01
01	10	01
10	10	11
11	11	00

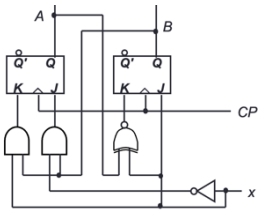
Present state		Input	Next state		Flip-flop inputs			
A	B	x	A^+	B^+	JA	KA	JB	KB
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK Flip-flop's excitation table.

Derive Flip-flop input functions

- $JA = B \cdot x'$
- $JB = x$
- $KA = B \cdot x$
- $KB = (A \oplus x)'$



Flip-flop Characteristic Tables

J	K	$Q(t+1)$	Comments
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$Q(t)'$	Toggle

S	R	$Q(t+1)$	Comments
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Unpredictable

D	$Q(t+1)$	
0	0	Reset
1	1	Set

T	$Q(t+1)$	
0	$Q(t)$	No change
1	$Q(t)'$	Toggle

Flip-flop Excitation Tables

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK Flip-flop

Q	Q^+	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

SR Flip-flop

Q	Q^+	D
0	0	0
0	1	1
1	0	0
1	1	1

D Flip-flop

Q	Q^+	T
0	0	0
0	1	1
1	0	1
1	1	0

T Flip-flop