Boolean Algebra and Combinatorial Logic

Part 1: Boolean algebra

Part 2: Combinatorial Logic.

Learning objectives:

- Understanding Boolean algebra.
- Understanding the difference between variable and operator.
- Be able to describe a system by an equation or a truth table.
- Understanding Combinatorial Logic

1

Part 1: Boolean Algebra

- Developed by George Boole in the 1850s
- Shannon was the first to use Boolean Algebra to solve problems in electronic circuit design. (1938)

2

Variables & Operations

- In Boolean algebra, all variables have the values 1 or 0.

  (sometimes we call the values TRUE or FALSE)

- Basic operators used in Boolean algebra:
  – OR written as +, e.g. in A +B
  – AND written as •, e.g. in A.B
  – NOT written as an over line or apostrophe, as in $\overline{A}$ or A’

3

Operators: OR

- The result of the OR operator is 1 if either of the operands is a 1.
- The only time the result of an OR is 0 is when both operands are 0s.

a	*b*	*f(a,b)*
0	0	0
0	1	1
1	0	1
1	1	1

4

Operators: AND

- The result of an AND is a 1 only when both operands are 1s. Therefore if either operand is a 0, the result is 0.

a	*b*	*f(a,b)*
0	0	0
0	1	0
1	0	0
1	1	1

5

Operators: NOT

- NOT *negates* it's operand.
- If the operand is a 1, the result of the NOT is a 0.
- If the operand is a 0, the result of the NOT is a 1.

a	*a'*
0	1
1	0

6

Equations

Boolean algebra uses equations to express relationships. For example:

$$X = A \cdot (\overline{B} + C)$$

This equation expresses a relationship between the value of ***X*** and the values of ***A***, ***B*** and ***C***.

7

Quiz

What is the value of each X_n:

$$X_1 = 1 \cdot (0 + 1)$$

$$X_2 = A + \overline{A}$$

$$X_3 = A \cdot \overline{A}$$

$$X_4 = X_4 + 1$$

8

Laws of Boolean Algebra

Just like in normal algebra, Boolean Algebra has postulates and identities.

These laws are useful for reducing or simplifying expressions.

9

Boolean Algebra Laws and Rules

- Boolean algebra uses many of the same laws as those of ordinary algebra.

Commutative law
Distributive law
Associative law

10

Commutative Laws

$$A + B = B + A$$

$$A \cdot B = B \cdot A$$

11

Distributive Laws

$$\boxed{A + (B \cdot C) = (A + B) \cdot (A + C)}$$

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

12

Associative Laws

$$A\,(B.C) = (A.B)\,C$$

$$A + (B + C) = (A + B) + C$$

13

Other Operators

- Boolean Algebra is defined over the 3 operators AND, OR and NOT.
  - this is a *functionally complete set*.
- However, there are other useful operators:
  - NOR: NOT(OR) is a 0 if either operand is a 1
  - NAND: NOT (AND) is a 0 only if both operands are 1
  - XOR: is a 1 if the operands are different.

14

More Laws!

A.B = B.A	A + B = B + A	Commutative law
A (B + C) = A.B + A.C	A + (B + C) = (A + B) + (A + C)	Distributive law
A (B.C) = (A.B) C	A + (B + C) = (A + B) + C	Associative Law
1.A = A	0 + A = A	Identity law
A.A’ = 0	A + A’ = 1	Inverse Element
0.A = 0	1 + A = 1	Null Element
A.A = A	A + A = A	Idempotent law
(A.B)’ = A’ + B’	(A + B)’ = A’.B’	De Morgan’s Theorem

15

Boolean Functions

1. Boolean functions are functions that operate on a number of Boolean variables.
2. The result of a Boolean function is itself either a 0 or a 1.

Example: f(a,b,c) = (a+b).c

16

Alternative Representation

Boolean functions can be represented in two ways:

1. We can define a Boolean function by describing it with algebraic operations (use equation).
2. We can also define a Boolean function by listing the value of the function for all possible inputs (using a truth table)

17

Truth Table of a Boolean Function

f(a,b)=a+b

a	*b*	*f(a,b)*
0	0	0
0	1	1
1	0	1
1	1	1

18

Truth Tables

a	*b*	OR	AND	NOR	NAND	XOR
0	0	0	0	1	1	0
0	1	1	0	0	1	1
1	0	1	0	0	1	1
1	1	1	1	0	0	0

19

Truth Table for (X+Y)·Z'

X	Y	Z	(X+Y)·Z'
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

20

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

		A	B	C		F
0		0	0	0		0
1		0	0	1		1
2		0	1	0		0
3		0	1	1		1
4		1	0	0		1
5		1	0	1		0
6		1	1	0		1
7		1	1	1		1

- : F = A'B'C + A'BC + AB'C' + ABC' +ABC : Sum of products = SOP
- : F = (A+B+C)(A+B'+C)(A'+B+C'): Product of Sums = POS

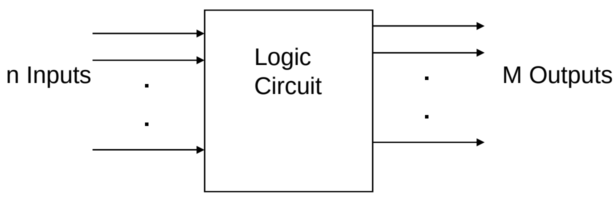
Part 2: Combinatorial Logic

Combinatorial and sequential logic

- **Combinational logic** is a type of logic circuit whose output is only a function of, the present input.
- **Sequential logic** is a type of logic circuit whose output depends not only on the present input but also on the previous inputs.
- Note: sequential logic will NOT be studied in this module

23

Combinatorial logic

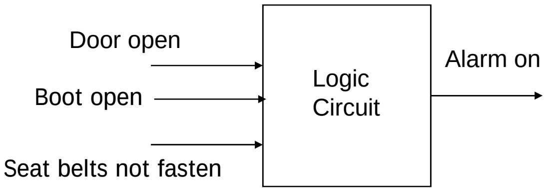


n inputs, n is integer greater than one
m outputs, m is integer greater than one

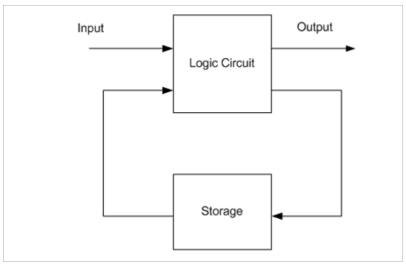
24

Combinatorial logic

- Example



Sequential logic

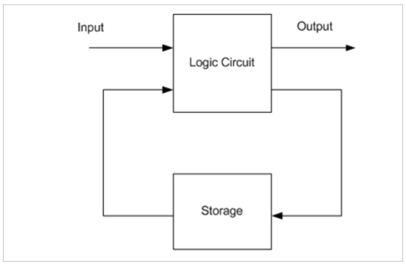


- Sequential logic has *storage* (*memory*) while combinational logic does not.

Sequential logic

■Example:

Electronics for an Automated teller machine (ATM)

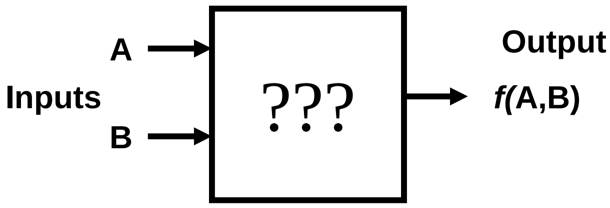


■a pin number eg: 1342 will be different than 1234

Gates

- Digital logic circuits are electronic circuits that are implementations of some Boolean function(s).
- A circuit is built up of *gates*, each *gate* implements some simple logic function.

A Gate



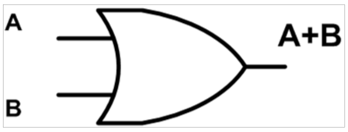
29

Gates compute something!

- The output depends on the inputs.
- If the input changes, the output might change.
- If the inputs don’t change – the output does not change.

30

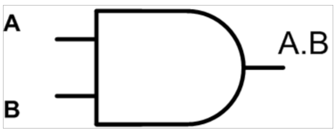
An OR gate



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

31

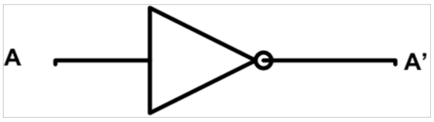
An AND gate



A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

32

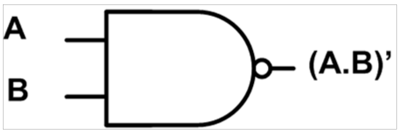
A NOT gate



A	A’
0	1
1	0

33

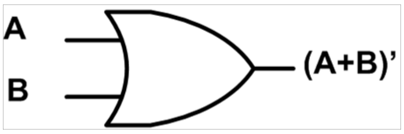
NAND gate



A	B	(A.B)’
0	0	1
0	1	1
1	0	1
1	1	0

34

NOR gate

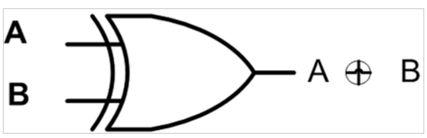


A	B	(A+B)’
0	0	1
0	1	0
1	0	0
1	1	0

35

An XOR gate

A	B	(A’B) +(AB’)
0	0	0
0	1	1
1	0	1
1	1	0



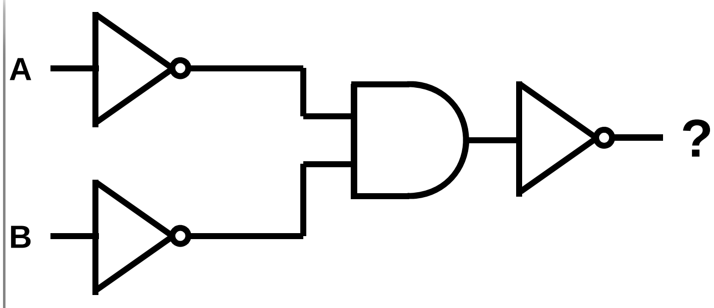
36

Combinational Circuits

- We can put gates together to produce a circuits
- We can design a circuit that represents any Boolean function!

37

A Simple Circuit



38

Truth Table for our circuit

a	b	$\overline{a}$	$\overline{b}$	$\overline{a} \bullet \overline{b}$	$\overline{\overline{a} \bullet \overline{b}}$
0	0	1	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	0	1

39

Alternative Representations

- Any of these can express a Boolean function:

(1) Boolean Equation

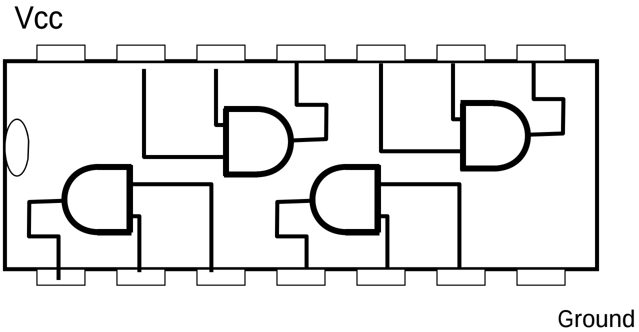
(2) Circuit (Logic Diagram)

(3) Truth Table

40

Integrated Circuits

- You can buy an AND gate *chip*:



41

Function Implementation

- Given a Boolean function expressed as a truth table or Boolean Equation, there are many possible implementations.
- The actual implementation depends on what kind of gates are available.
- In general we want to minimize the number of gates.

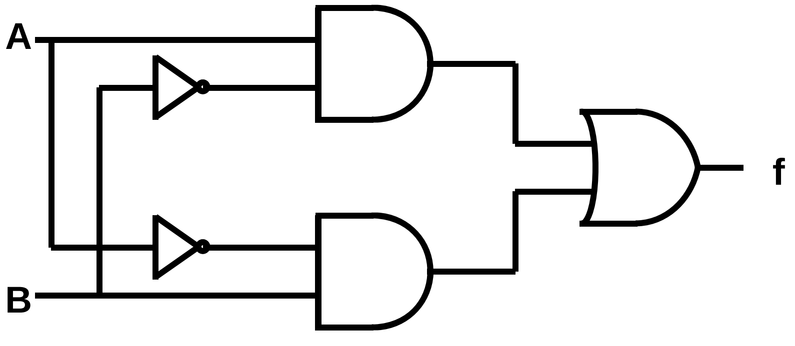
42

Example: $f = A \bullet \overline{B} + \overline{A} \bullet B$

A	B	$A \bullet \overline{B}$	$\overline{A} \bullet B$	f
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

43

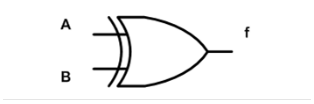
One Implementation $f = A \bullet \overline{B} + \overline{A} \bullet B$



44

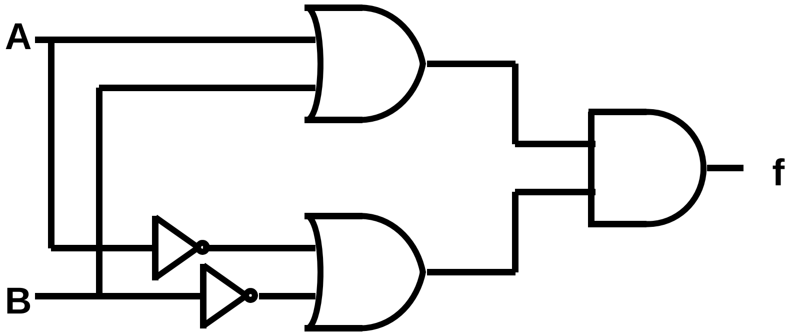
One Implementation $f = A \bullet \overline{B} + \overline{A} \bullet B$

A simplified version



45

Another Implementation



$$f = A \bullet \overline{B} + \overline{A} \bullet B = (A + B) \bullet (\overline{A} + \overline{B})$$

46

Supplement:
More example and Applications

47

Prove f1 = f2

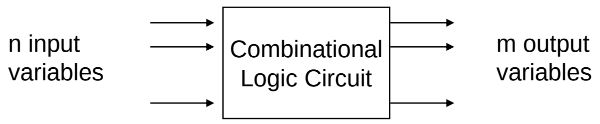
$$f_1 = A \bullet \overline{B} + \overline{A} \bullet B =$$
$$f_2 = \left(\bar{A} + \bar{B}\right) \bullet (A + B)$$

$A \bullet \overline{B} + \overline{A} \bullet B =$	f1
$\overline{\left(\overline{A \bullet \overline{B}}\right) \bullet \left(\overline{\overline{A} \bullet B}\right)} =$	DeMorgan's Law
$\overline{\left(\overline{A} + B\right) \bullet \left(A + \overline{B}\right)} =$	DeMorgan's Laws
$\overline{\left(\left(\overline{A} + B\right) \bullet A\right) + \left(\left(\overline{A} + B\right) \bullet \overline{B}\right)} =$	Distributive
$\overline{\left(\overline{A} \bullet A + B \bullet A\right) + \left(\overline{A} \bullet \overline{B} + B \bullet \overline{B}\right)} =$	Distributive
$\overline{\left(B \bullet A\right) + \left(\overline{A} \bullet \overline{B}\right)} =$	Inverse, Identity
$\overline{\left(B \bullet A\right)} \bullet \overline{\left(\overline{A} \bullet \overline{B}\right)} =$	DeMorgan's Law
$\left(\overline{B} + \overline{A}\right) \bullet \left(A + B\right)$	DeMorgan's Laws

48

Combinational Logic Circuits

- A combinational circuits
  - 2^n possible combinations of input values



- These circuits implement specific functions such as
  - Adders, subtractors, comparators, decoders, encoders, and multiplexers

49

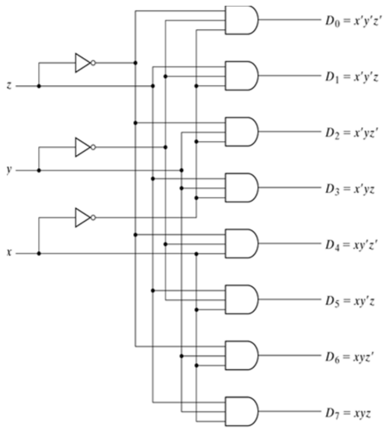
Decoder

- A circuit that converts binary information from n input lines to a maximum of 2^n unique output lines
  - At anytime, exactly one output is 1, all others are 0.
- One of the applications of decoder is seven-segment display.

50

Example: A 3-8 Line decoder

3 inputs are decoded to eight outputs



51

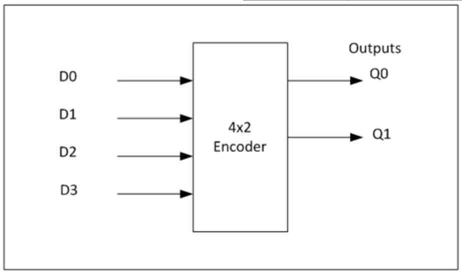
Encoder

- Perform the inverse operation of a Decoder
- A encoder
  - Has 2^n inputs and n outputs
  - The output lines generate binary code corresponding to the input

52

Encoder

Inputs				Outputs	
D3	D2	D1	D0	Q1	Q0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x



Multiplexer

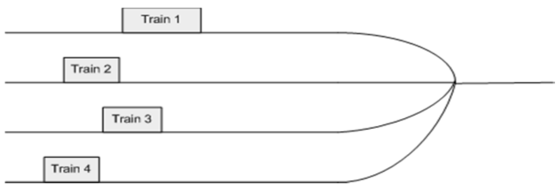
- A combinational circuit that selects binary information from one of the input lines and directs to a single output line
- The selection of a particular input line is controlled by a set of selection lines.

- 2 input mux will need 1 select input to indicated which input to route through
- 4 input mux will need 2 select inputs
- 8 input mux will need 3 select inputs
- N inputs will need log2(N) select inputs

55

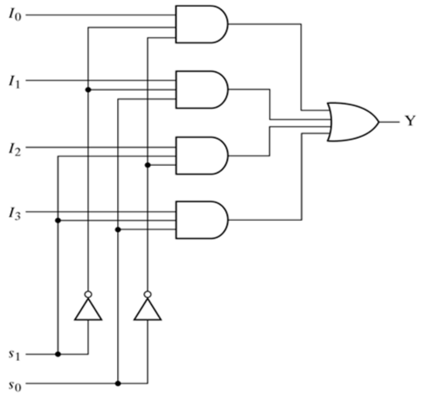
Multiplexer

- 2 input mux will need 1 select input to indicated which input to route through
- 4 input mux will need 2 select inputs
- 8 input mux will need 3 select inputs
- ....
- N inputs will need $log_2\ (N)$ select inputs
- Example:



56

Example: A 4-to-1 line Multiplexer



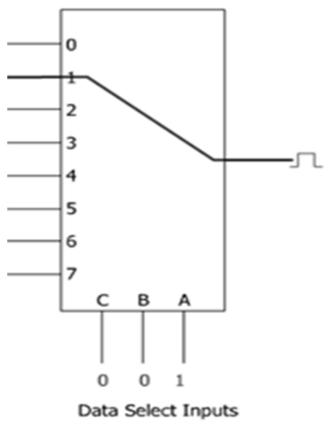
(a) Logic diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

57

Example: A 8-to-1 line Multiplexer



58

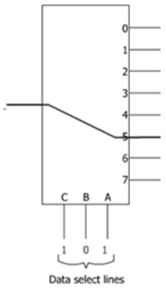
Demultiplexer

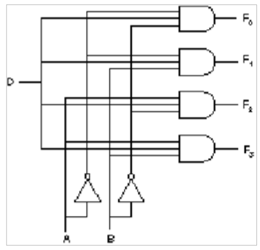
- Demultiplexer – a circuit that receives information from a single line and directs it to one of the 2^n possible outputs
  - Selection of the output is done by the n selection lines

A demultiplexer is a decoder with Enable inputs

59

Demultiplexer





60

Adders

- A Binary adder is a circuit that computes the arithmetic sum of two binary numbers of a given length
- There are several adder circuits
  - Half adder – a combinational circuit that performs addition of 2 bits
  - Full adder – a combinational circuit that performs addition of 3 bits (2 significant bits and a previous carry)
  - Two Half adders can be combined to form a Full adder

61

The Half Adder

- Adding two 1-bit values results in the following
  - 0+0=0, 0+1=1, 1+0=1 (a sum with 1 bit)

  and
  - 1+1=10 (a sum with 2 bits, including a higher significant bit called a carry)
- The Half Adder
  - requires two input variables: x, y
  - has two output variables: C (carry), S (sum)

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

62

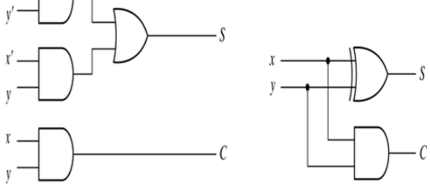
A Half-Adder Circuit

- The Boolean expressions are:
  - S = xy’ + x’y
  - C = xy
- These can be implemented using
  - AND and OR gates in sum-of-products form
  - XOR and AND gate

63

A Half-Adder Circuit

- The Boolean expressions are:
  - S = xy’ + x’y
  - C = xy



64

Full-Adder

- A full-adder is to perform the addition of three bits (with previous carry bit, z).
- The Full-Adder has
  - three input bits
    - x, y: two significant bits
    - z: the carry bit from the previous lower significant bit
  - two output bits: C, S

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

65

Implementing the Adder

- The boolean expressions are:

  c = x’yz + xy’z + xyz’+xyz

  = z(xy’+x’y) + xy

  s = x’y’z + x’yz’ + xy’z’+ xyz

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Implementation using 2 half adders

66

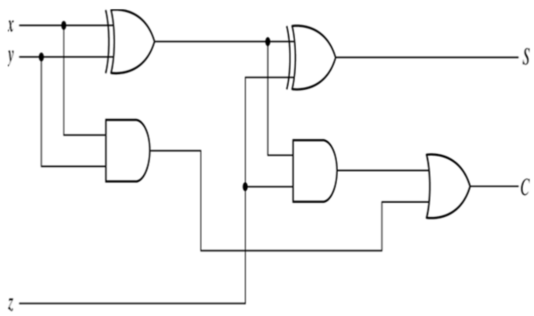
Implementing the Adder

- The boolean expressions are:

  c = z(xy’+x’y) + xy

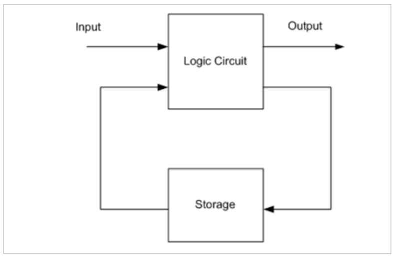
  s = x’y’z + x’yz’ + xy’z’+ xyz

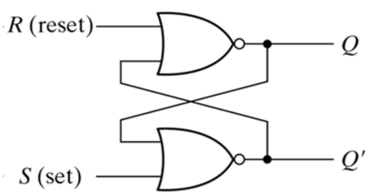
Implementation using
2 half adders



Sequential logic: Example

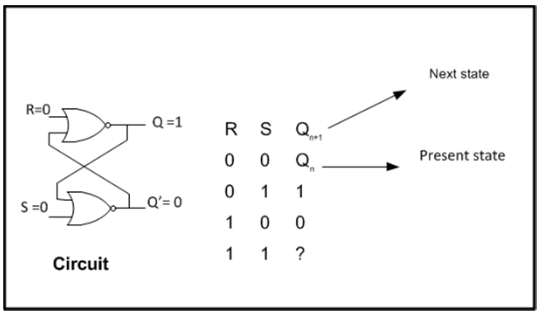
Example:





Sequential logic: SR NOR Latch operation. (flip-flop of latch)

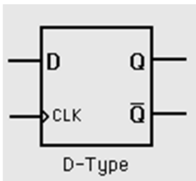
Let's Assume some previous operation has Q as a 1

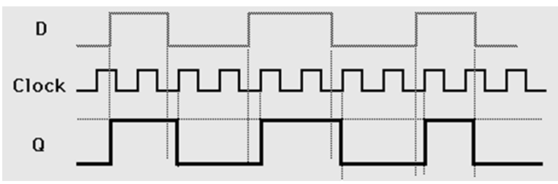


Simple set-reset latches:
- SR NOR latch
- SR NAND latch
- SR AND-OR latch
- JK latch

69

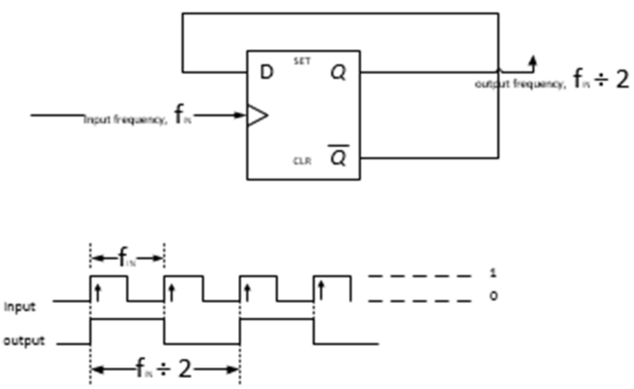
D-type flip flop Synchronous latch





70

D-type flip flop
Divide-by-2 Counter



71