COMP2611 COMPUTER ORGANIZATION
TOPIC 2 INTRODUCTION TO DIGITAL LOGIC

Content

- **This topic explains (conceptually):**
  - How are bits represented and handled in the hardware?
  - How to design a simple circuit to perform an abstract task (e.g. addition)?

Power of Bit

- **Analog vs. Digital waveforms**
  - Digital: only assumes discrete values
  - Analog: values vary over a broad range continuously

- **The electronics inside modern computers are digital: they operate with only two voltage levels of interest**
- **Typical voltage assignment**

  - Binary 1: any voltage between 2.4V to 2.9V
  - Binary 0: any voltage between 0V to 0.5V

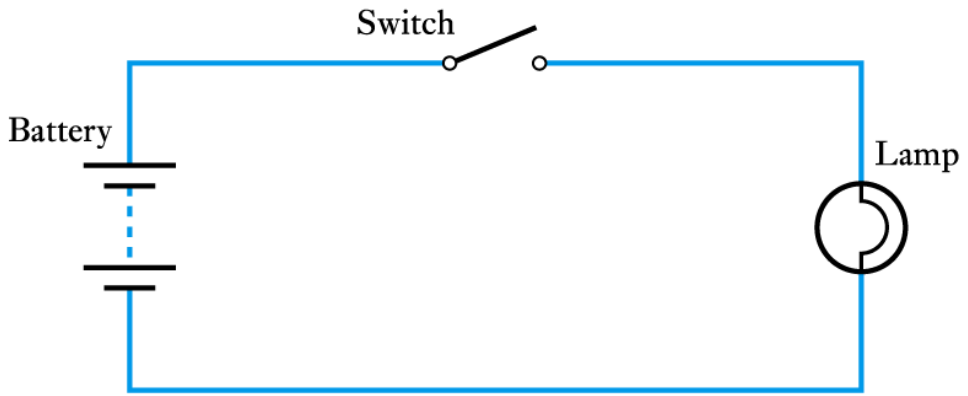
Bit in Computer

- **Bits** are the basis for **binary number representation** in digital computers
- Operations in computer work on bits

- **Combining bits** into patterns **following some conventions or rules (defined in the ISA)** allow for:
  - Number representations
    - Integers,
    - Fractions and Real numbers, …
  - Instruction encoding
    - Operation
    - Operands

Lamp Example

- Example: lamp with switch



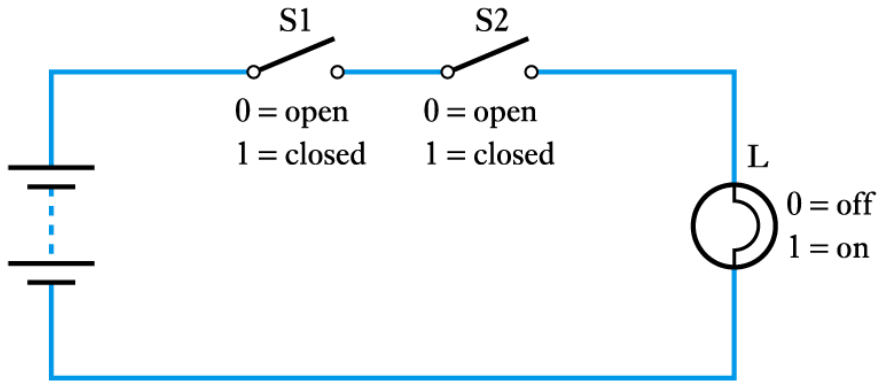
S	*L*
OPEN	OFF
CLOSED	ON

S	*L*
0	0
1	1

Lamp truth table

Lamp Example (cont.)

- Two switches in series

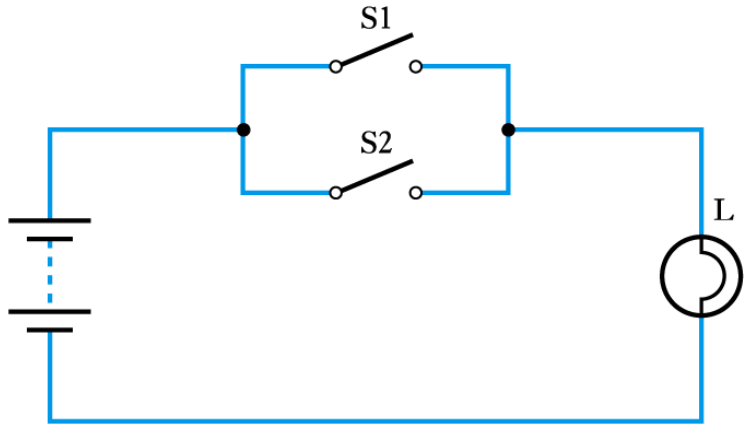


(a) Circuit

S1	S2	L
0	0	0
0	1	0
1	0	0
1	1	1

(b) Truth table

- Two switches in parallel



(a) Circuit

S1	S2	L
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

Truth Table

- A mathematical table used in logic
- In connection with Boolean algebra, Boolean functions
- Shows how the truth or falsity of a proposition (output) varies with that of its components (inputs)

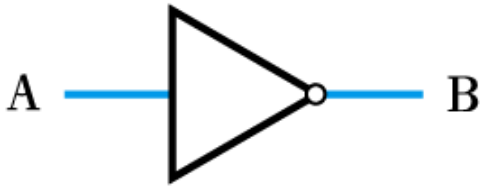
- One column for each input variable and each output variable
- Each row of the truth table contains one possible configuration of the input variables

Boolean Algebra and Logic Operation

- Everything in digital computer is binary, the theoretical model works behind is Boolean algebra
- Algebra: values, variables, operations
- In Boolean algebra
  - The values are the symbols 0 (or FALSE) and 1 (or TRUE)
  - A Boolean variable (or Binary variable) is one that can take only 2 values, 0 or 1
  - Three fundamental logic operations on Boolean variables: AND, OR, NOT
  - Other commonly used logic operations: NAND, NOR, XOR
  - Logic gates implement the above logic operations and built up by transistors

NOT Operation and NOT Gate

- **The NOT gate (or inverter)**



(a) Circuit symbol

A	B
0	1
1	0

(b) Truth table

$B = \overline{A}$

(c) Boolean expression

AND Operation and AND Gate

The AND gate

(a) Circuit symbol

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

(b) Truth table

$$C = A \cdot B$$

(c) Boolean expression

OR Operation and OR Gate

The OR gate

(a) Circuit symbol

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

(b) Truth table

$C = A + B$

(c) Boolean expression

NAND Operation and NAND Gate

- **The NAND gate**

(a) Circuit symbol

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

(b) Truth table

$$C = \overline{A \cdot B}$$

(c) Boolean expression

NOR Operation and NOR Gate

- **The NOR gate**

(a) Circuit symbol

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

(b) Truth table

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

(c) Boolean expression

Exclusive-OR Operation and Exclusive-OR Gate

- **The Exclusive OR (XOR) gate**

(a) Circuit symbol

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

(b) Truth table

$$C = A \oplus B$$

(c) Boolean expression

Logic Function

- Any operation in digital circuit can be described by truth table or logic function
- A logic function is a function on binary variables whose output is also a binary variable
- *Three fundamental* NOT, AND, OR logic operations are at the center of all operations in modern computers
- Logical functions can be expressed in several alternative ways:
  - Truth table
  - Logical expressions
  - Graphical form, e.g. K-map

Digital Logic Circuit Example

- A circuit with two inputs (A, B) and two outputs (S, C)
- The circuit executes as follows
  - S is true if exactly one input is true;
  - C is true if both two inputs are true.

- Describe the behaviour of the circuit with truth table and logic function
- Build the circuit with logic gates
- What is this circuit for?

Digital Logic Circuit Example (cont.)

- The circuit is a **1-bit half adder**

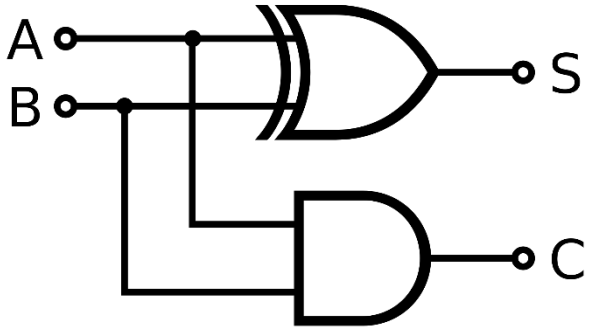
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth Table

$$S = A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$$

$$C = A \cdot B$$

Logic Function



Circuit Implementation

Computer Arithmetic and Logic operation can be specified via logic functions

Digital Logic Circuit

- **Two types of digital logic circuits inside a computer:**
  - **Combinational logic circuits**
    - Logic circuits that do not have memory
    - The output depends only on the current inputs and the circuit
    - They can be specified fully with a truth table or a logic equation
  - **Sequential logic circuits**
    - Logic circuits that have memory
    - The output depends on both the current inputs and the value stored in memory (called state)

COMBINATIONAL LOGIC

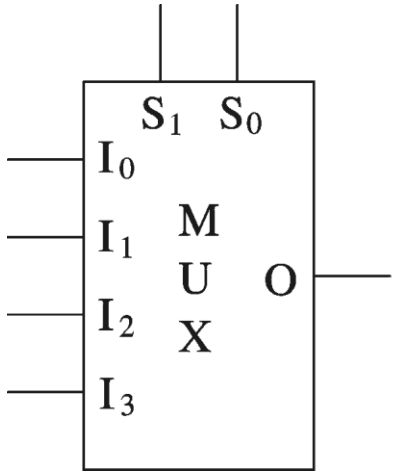
Simple Combinational Logic Circuits

- **Multiplexor/ De-multiplexor**
- **Decoder/Encoder**
- **Two-level logic and Programmable Logic Array (PLA)**

- The above circuits can be implemented using AND, OR, and NOT gates only
- They are higher-level basic building blocks that are commonly seen in combinational logic

Multiplexor

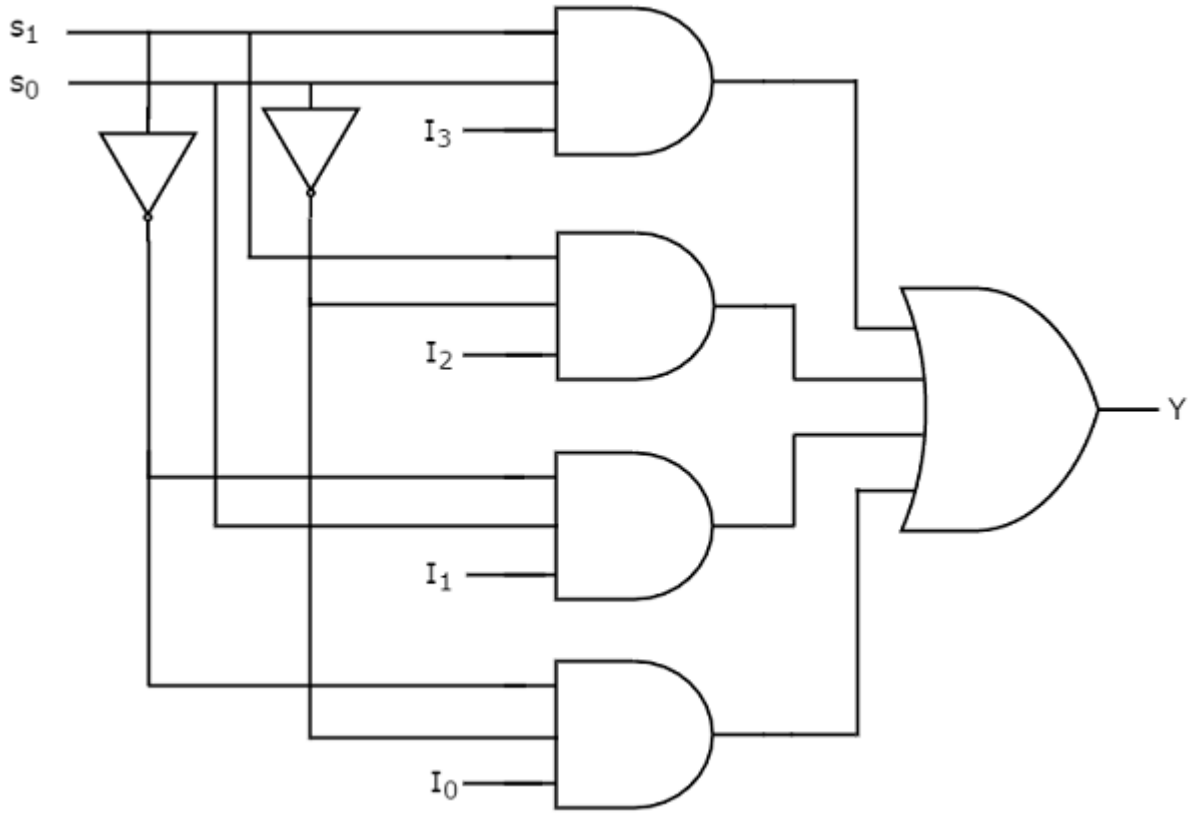
- A multiplexor (or selector) selects one of the data inputs as output by a control input value
  - Example: 4-to-1 multiplexor



S_1	S_0	O
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

- For a 2^n-to-1 multiplexor
  - 2^n data inputs
  - n selection inputs
  - a single output

4-to-1 MUX Implementation

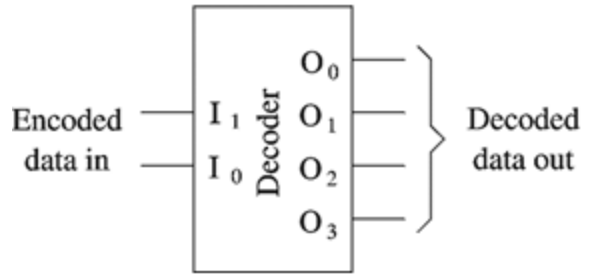


- How to implement a higher order MUX, e.g. 8-to-1?

Decoder

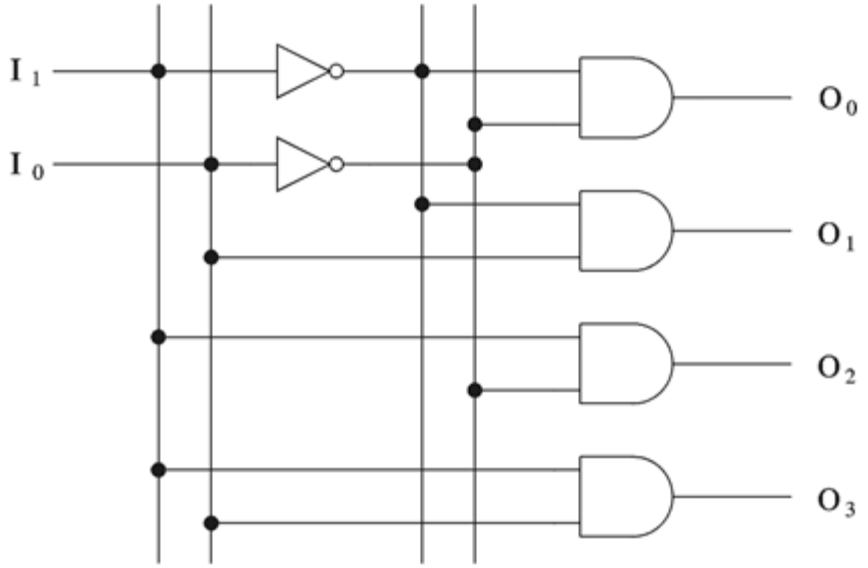
- A decoder (N-to-2^N decoder) is a logical block with an N-bit input and 2^N 1-bit outputs

I_1	I_0	O_3	O_2	O_1	O_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



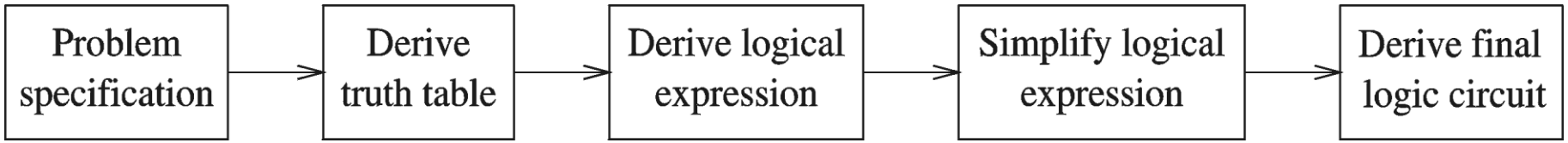
- The output corresponds to the input bit pattern is true while all other outputs are false.

- Example: 2-to-4 decoder



Logic Circuit Design Process

- **A simple logic design process involves**
  - Problem specification
  - Truth table derivation
  - Derivation of logical expression
  - Simplification of logical expression
  - Implementation



Example: 3-person Majority Vote

- Problem: 3 person vote for a motion, if 2 or more says YES, the motion is approved

- Step 1: truth table
- Step 2: derive the logic function according to the truth table

Inputs			Output
X	Y	Z	D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

minterm vs. maxterm

- **minterm**, denoted as m_i, where $0 \leq i \leq 2^n - 1$, is the product (AND) of n Boolean variables in its original or negated form

- **maxterm**, denoted as M_i, where $0 \leq i \leq 2^n - 1$, is the sum (OR) of n Boolean variables in its original or negated form

Variable			Minterm		Maxterm	
x	y	z	Term	Designation	Term	Designation
0	0	0	x'y'z'	m_0	x+y+z	M_0
0	0	1	x'y'z	m_1	x+y+z'	M_1
0	1	0	x'yz'	m_2	x+y'+z	M_2
0	1	1	x'yz	m_3	x+y'+z'	M_3
1	0	0	xy'z'	m_4	x'+y+z	M_4
1	0	1	xy'z	m_5	x'+y+z'	M_5
1	1	0	xyz'	m_6	x'+y'+z	M_6
1	1	1	xyz	m_7	x'+y'+z'	M_7

Canonical Forms

- Any Boolean function can be expressed as
  - a sum (OR) of its 1-minterms, or
  - a product (AND) of its 0-maxterms
- Any Boolean function that is expressed as a sum of minterms or as a product of maxterms is said to be in its canonical form

- Example: Canonical form of majority vote

Inputs			Output
A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$\mathrm{D} = m_3 + m_5 + m_6 + m_7$$
$$= \bar{\mathrm{A}}\mathrm{BC} + \mathrm{A}\bar{\mathrm{B}}\mathrm{C} + \mathrm{AB}\bar{\mathrm{C}} + \mathrm{ABC}$$

$$\mathrm{D} = M_0 M_1 M_2 M_4$$
$$= (A + \mathrm{B} + \mathrm{C})(A + \mathrm{B} + \bar{\mathrm{C}})(\mathrm{A} + \bar{\mathrm{B}} + \mathrm{C})(\bar{A} + \mathrm{B} + \mathrm{C})$$

Two-Level Logic

- ***Any*** **logic function can be expressed in a canonical form as a two-level representation:**
  - Every input is either a variable or its negated form.
  - One level consists of **AND** gates only.
  - The other level consists of **OR** gates only.

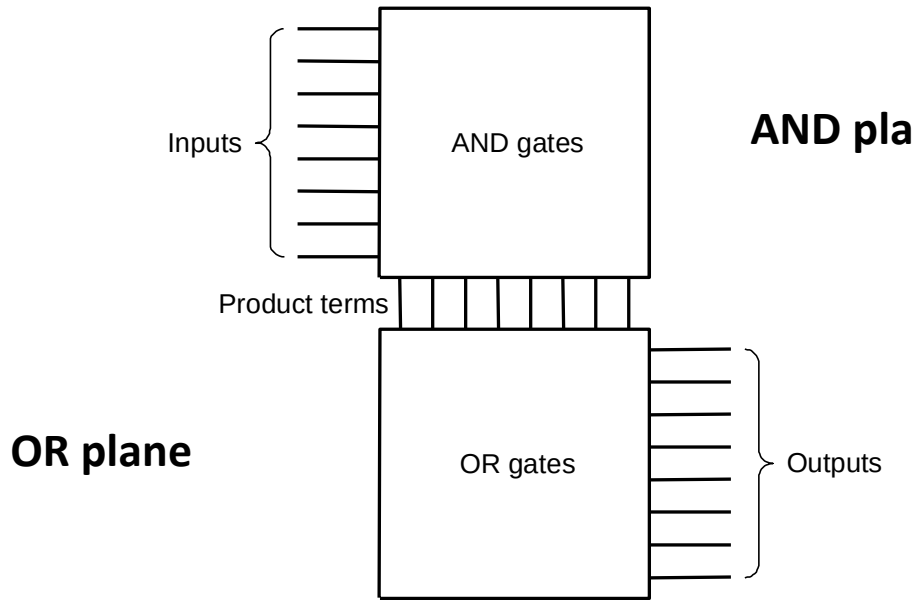
- **Sum-of-products (SOP) form:**
  - E.g., $E = (A \cdot B \cdot \overline{C}) + (A \cdot C \cdot \overline{B}) + (B \cdot C \cdot \overline{A})$
  - More commonly used than product-of-sums representation.

- **Product-of-sums (POS) form:**
  - E.g., $E = (\overline{A} + \overline{B} + C) \cdot (\overline{A} + \overline{C} + B) \cdot (\overline{B} + \overline{C} + A)$

- **Logic function with only AND, OR, NOT can express all possible truth tables**

Programmable Logic Array

- A programmable logic array (PLA) is a gate-level *implementation* of the two-level representation for any set of logic functions, which corresponds to a truth table with multiple output columns.
- Example: A PLA corresponds to the sum-of-products representation



PLA Example

- Show a PLA implementation of this example:

Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

- Sum-of-product representation

$$D = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

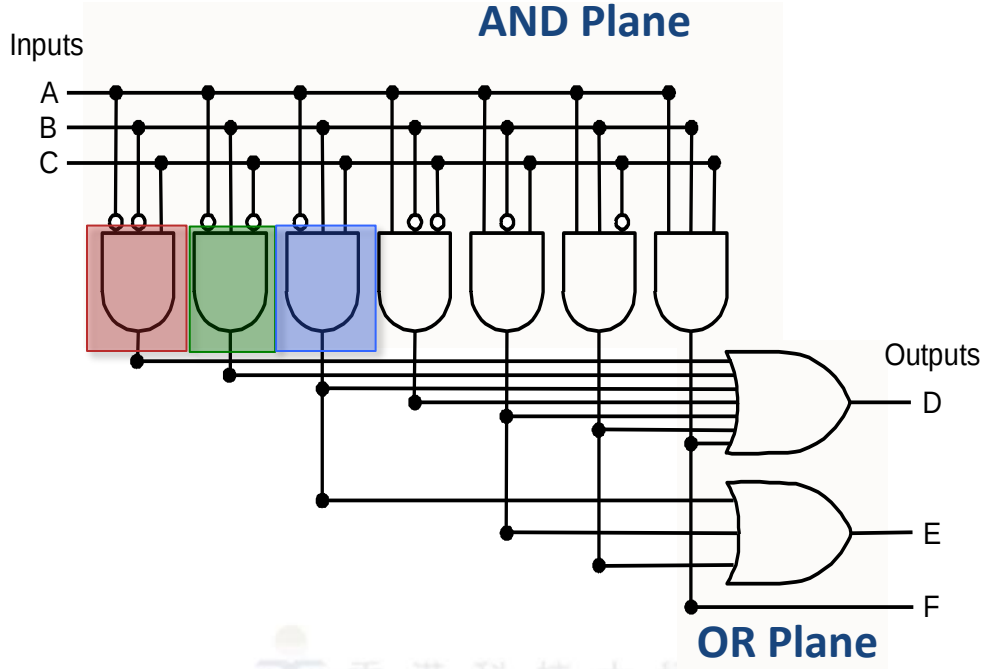
$$E = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$$

$$F = A \cdot B \cdot C$$

PLA Example (cont.)

- 3 inputs -> 3 rows in the AND plane
- 7 unique product terms with at least one TRUE value in the output -> 7 columns in the AND plane.
- 3 outputs -> 3 rows in the OR plane

Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

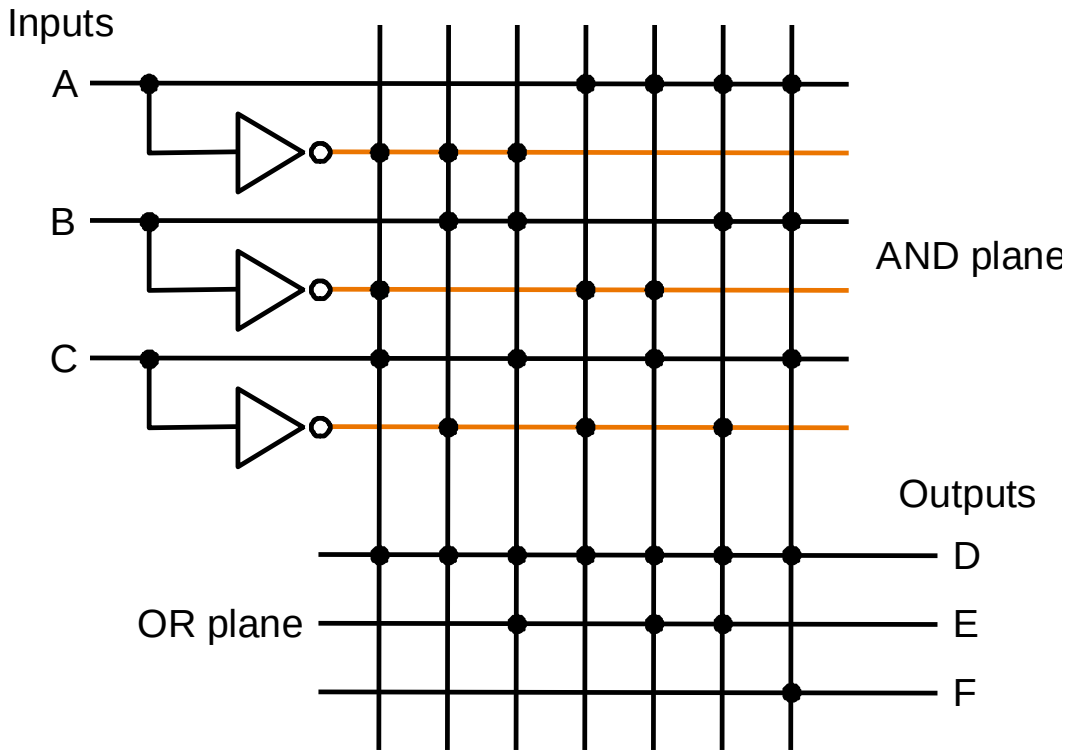


PLA Example (cont.)

$$D = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

$$E = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$$

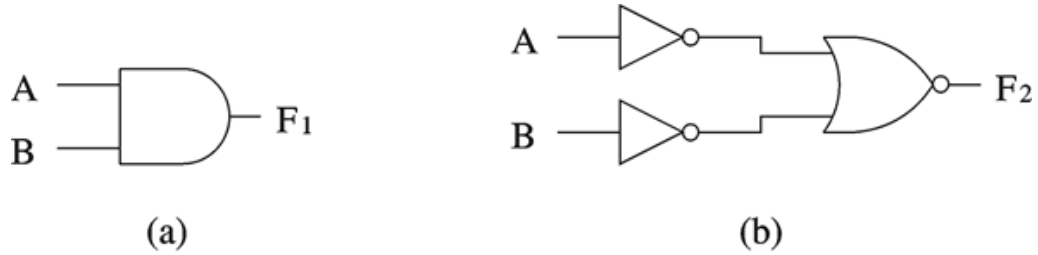
$$F = A \cdot B \cdot C$$



An equivalent PLA representation

Logical Equivalence

- **Are the two circuits below doing the same thing?**



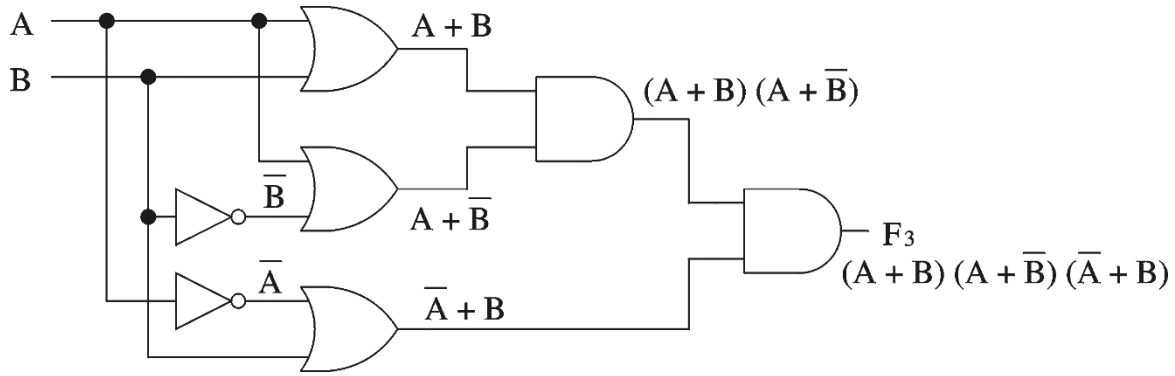
- **Proving logical equivalence of two circuits**
  - Derive the logical expression for the output of each circuit
  - Show that these two expressions are equivalent

- **Method 1: truth table**
  - For every combination of inputs, if both expressions yield the same output, they are equivalent
  - Good for logical expressions with small number of variables
- **Method 2: algebraic manipulation**
  - With help of Boolean Algebra (introduced later in circuit simplification)

Example: Logic Equivalence with Truth Table

- **Is the circuit logically equivalent to the AND gate?**

- **Derive logical expression from a circuit**
  - Trace from the input to output
  - Write down intermediate logical expressions along the path
- **work out the truth table**
- **Proving logical equivalence**



A	B	F = AB	F = (A + B)(A + $\overline{B}$)($\overline{A}$ + B)

Logic Expression Simplification

- Truth tables can grow rapidly in size and become tedious.
  - N inputs means 2^n rows in truth table
- Logic expressions are better in this case, however, there are equivalent logic expressions for the same truth table, and each of them will lead to a circuit implementation
- Simplifying logic expressions leads to simpler and cheaper circuits (lesser components)

- There are many formal methods, algorithms and software for simplifying Boolean expressions
  - Boolean Algebra
  - Karnaugh-Maps (K-maps)

Boolean Algebra Revisit

- Boolean Algebra is the algebra of logic that deals with the study of binary variables and logical operations
- It transform logical statements into mathematical symbols and calculate the truth or falsity of related statements by using rules

- Laws of Boolean Algebra are powerful tools to simplify logic equations

Basic Laws of Boolean Algebra

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$

Boolean Algebra Example

- **Simplify** $C + \overline{BC}$

- **Simplify** $\overline{AB}(\bar{A} + B)(\bar{B} + B)$

- **Simplify** 3-person majority vote $\mathrm{D} = \overline{\mathrm{A}}\mathrm{BC} + \mathrm{A}\overline{\mathrm{B}}\mathrm{C} + \mathrm{AB}\overline{\mathrm{C}} + \mathrm{ABC}$

K-Map

- K-Map is a graphical representation of the truth table or logic function
- In a K-map each cell represents one possible minterm
- Cells are arranged following a Gray code i.e., two adjacent cells are such that the corresponding minterms differ in only one variable

K-map Layout Example

A \ B	0	1
0	m0	m1
1	m2	m3

2 inputs

A \ BC	00	01	11	10
0	m0	m1	m3	m2
1	m4	m5	m7	m6

3 inputs

AB \ CD	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10

4 inputs

K-map Simplification Rules

- Find largest size groups of adjacent cells at 1
- 2^N (i.e. 1, 2, 4, 8) adjacent cells in each group
- K-map is toroid (i.e., rightmost cells are adjacent to the leftmost cells and topmost cells are adjacent to bottom cells)
- Larger groups = fewer inputs to the AND gate
- Fewer groups = fewer AND gates and fewer inputs to OR gate
- Best group might not be unique

Simplification Example

- Simplify $F = A \cdot \overline{B} + A \cdot B + \overline{A} \cdot B$

A \ B	0	1
0	$\overline{A} \cdot \overline{B}$	$\overline{A} \cdot B$
1	$A \cdot \overline{B}$	$A \cdot B$

A \ B	0	1
0	0	1
1	1	1

$F = A + B$

- Simplify majority vote

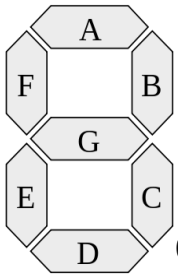
$D = (\overline{A} \cdot B \cdot C) + (A \cdot B \cdot \overline{C}) + (A \cdot \overline{B} \cdot C) + (A \cdot B \cdot C)$

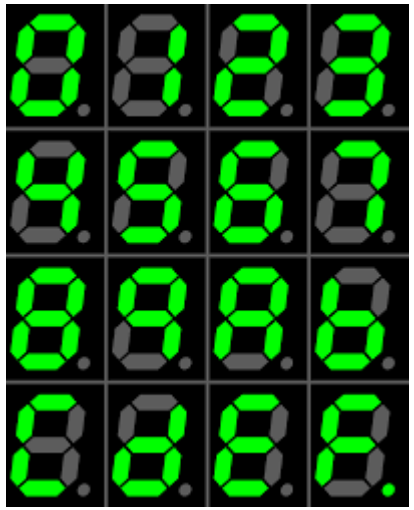
A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

D = AB + AC + BC

7-Segment Display

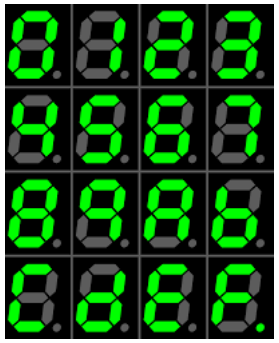
- Use 7-segment digital display to display one Hexadecimal digit
- Each segment is represented by a logic function





- Tasks:
  - Give the truth table for segment G
  - How many inputs needed?
  - Deduce the sum-of-products logic equation from the table
  - Use K-Map to simplify the logic equation

Truth Table for Segment G



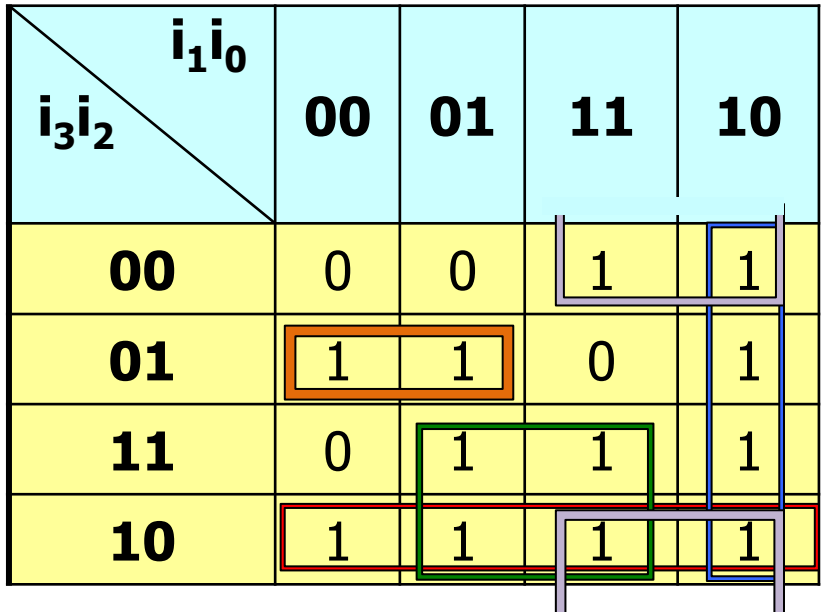
Inputs				Output	
i_3	i_2	i_1	i_0	G	A
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	1	
0	0	1	1	1	
0	1	0	0	1	
0	1	0	1	1	
0	1	1	0	1	
0	1	1	1	0	
1	0	0	0	1	
1	0	0	1	1	
1	0	1	0	1	
1	0	1	1	1	
1	1	0	0	0	
1	1	0	1	1	
1	1	1	0	1	
1	1	1	1	1	

- Using the table we have 12 minterms for segment G in the logic expression
- Try Segment A as an exercise

K-Map and Simplification

- $G = i_1 i_0' + i_3 i_2' + i_3 i_0 + i_2' i_1 + i_3' i_2 i_1'$

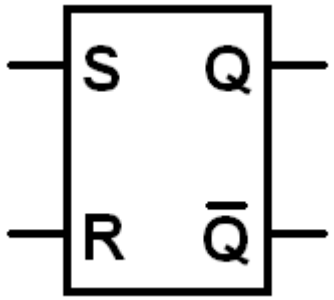
- **Conclusion:**
  - Before simplification we would need 12 AND gates with 4 inputs each and one OR gate with 12 inputs
  - After we only need 4 AND gates with 2 inputs one AND gate with 3 inputs and one OR gate with 5 inputs



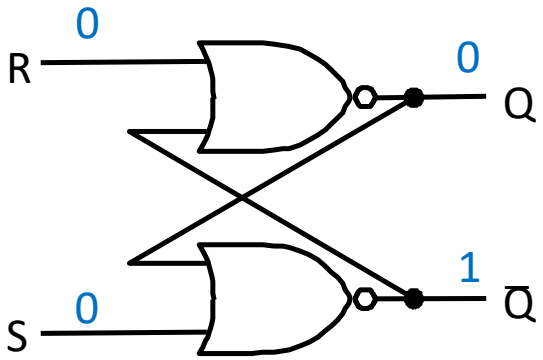
BASIC SEQUENTIAL LOGIC

S-R Latch: Simple Storage Element

- **S-R latches (set-reset latches)**
  - **Unclocked** memory element
  - R is used to ‘reset’ or ‘clear’ the element
  - S is used to ‘set’ the element
  - If both R and S are ‘not chosen’, output is in ‘Quiescent’ state
    - hold its previous value
    - can be either 0 or 1
  - Usually built from a pair of cross-coupled NOR (or NAND) gates.



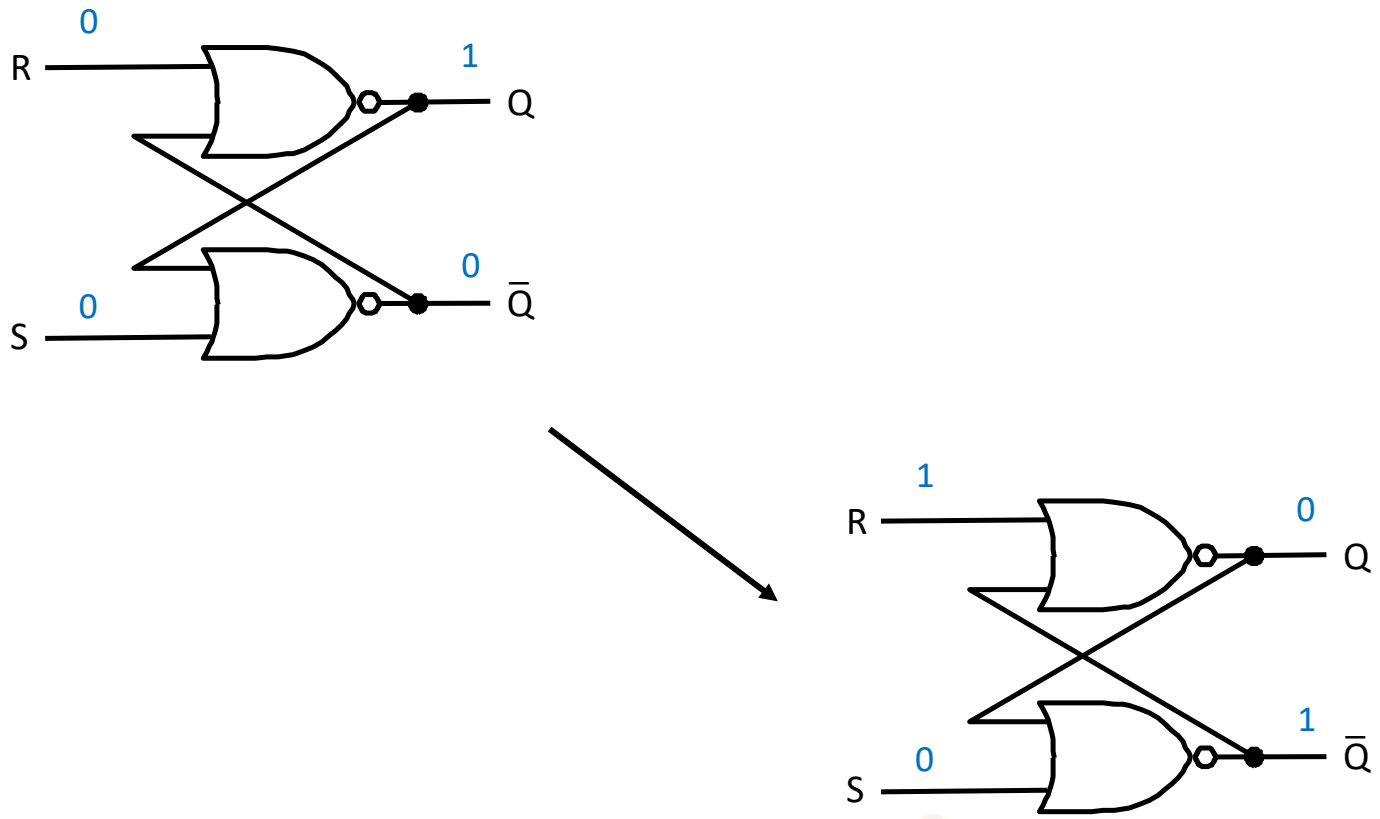
Graphical Symbol



A S-R latch built by NOR gates
in quiescent state when previous value is 0

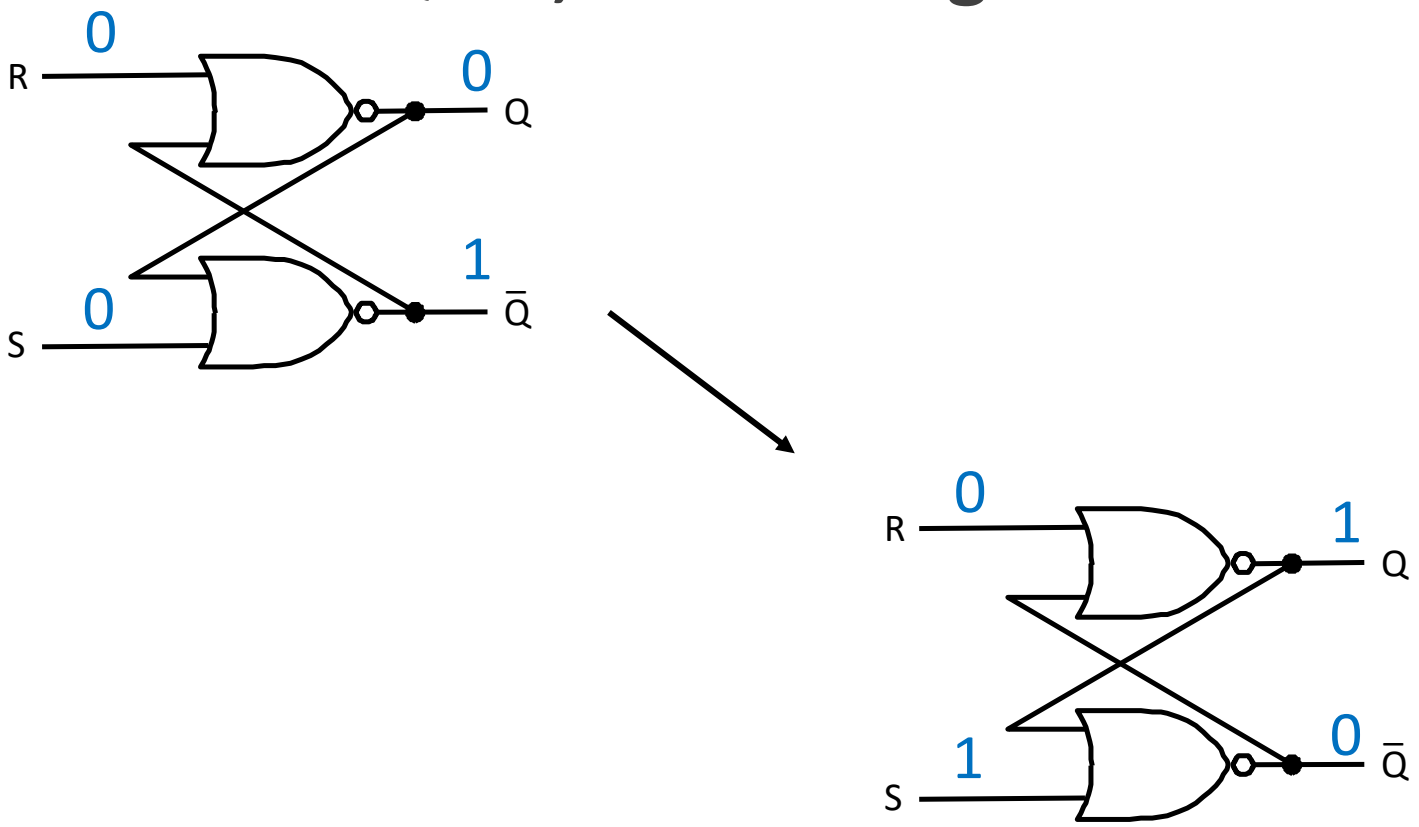
Reset S-R Latch

- Suppose we start with Q = 1, then change R to 1



Set the S-R Latch

- Suppose we start Q = 0, then change S to 1



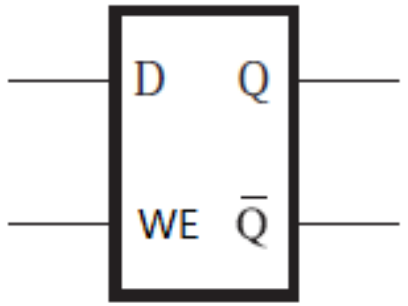
S-R Latch (with NOR Gate) Summary

- **Both Set and Reset are de-asserted (low), R = 0, S = 0**
  - Hold current value in latch
- **Reset is asserted, R = 1, S = 0**
  - Set value to 0
- **Set is asserted, R = 0, S = 1**
  - Set value to 1
- **Both are asserted (high), R = 1, S = 1**
  - Both outputs equal to 0
  - Final state determined by electrical properties of gates (racing condition)
  - Invalid input

Inputs		Outputs	
R	S	Q	Not Q
0	0	Latch	
1	0	0	1
0	1	1	0
1	1	Invalid	

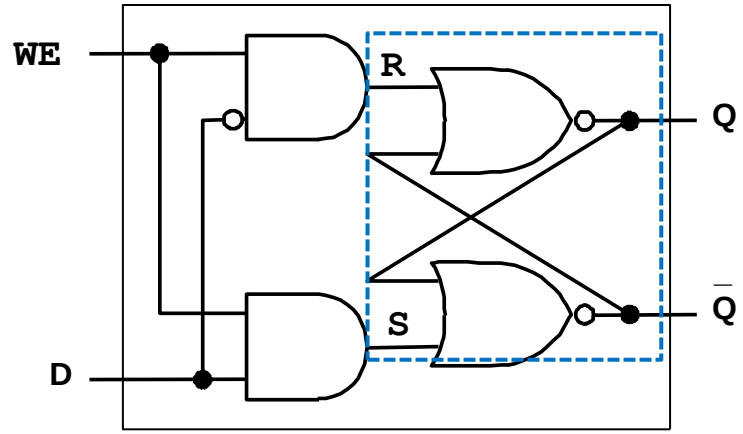
Gated D-Latch

- D-latch with two inputs
  - D (data)
  - WE (write enable)
- When WE = 1, latch is set to value of D
- When WE = 0, latch holds previous value



Graphical Symbol

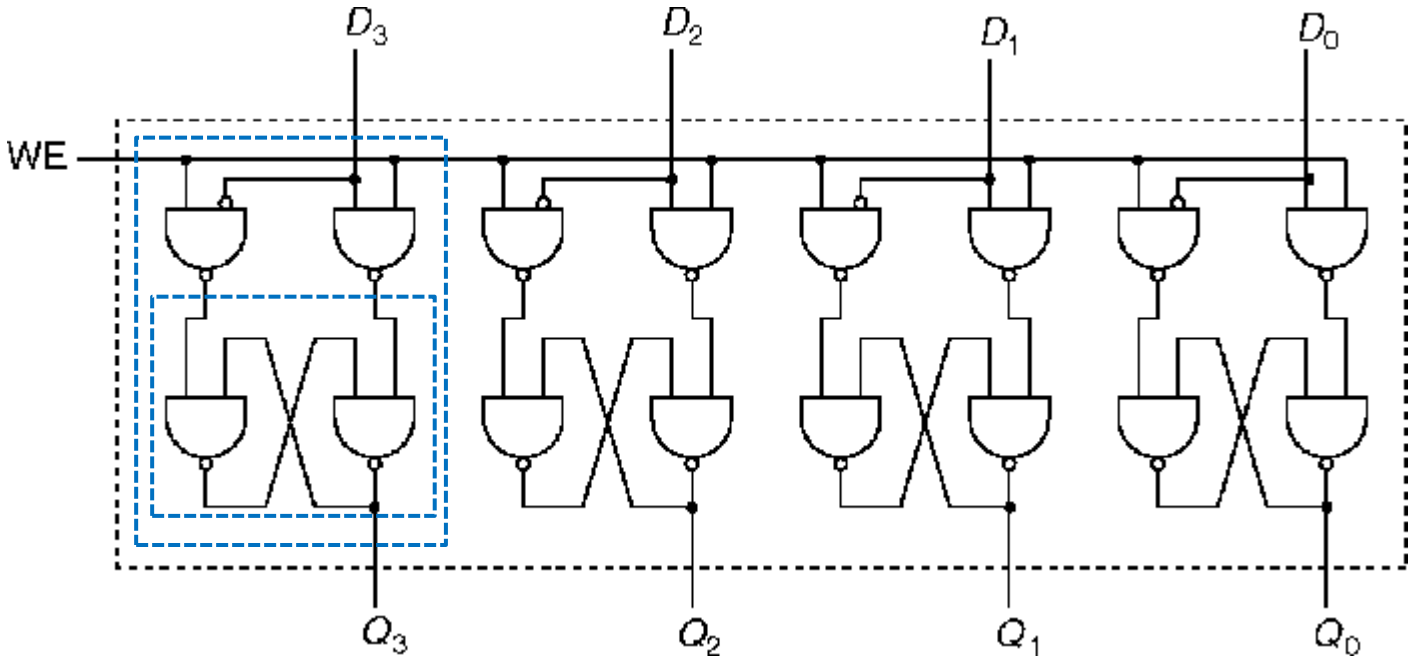
Inputs		Outputs	
WE	D	Q	Not Q
0	0	Latch	
0	1	Latch	
1	0	0	1
1	1	1	0



A gated D latch built by NOR S-R latches

Register

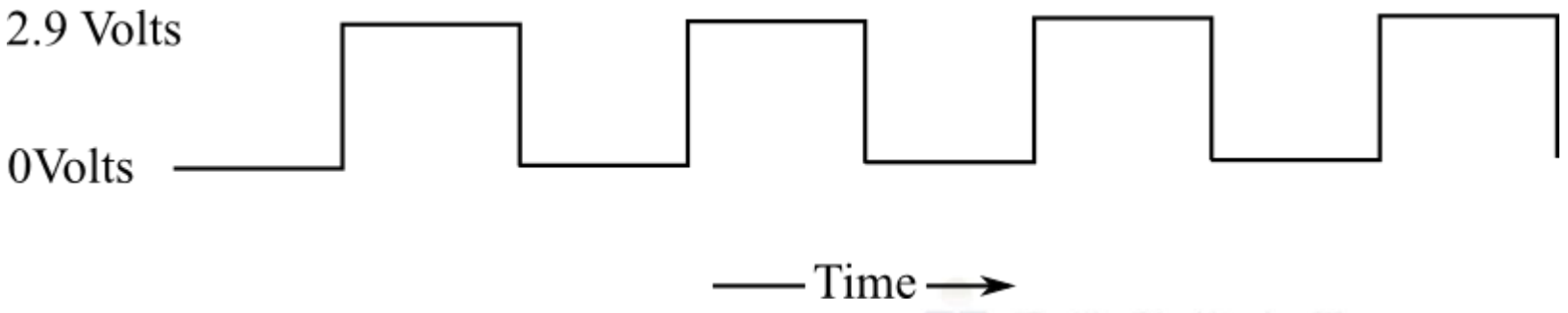
- **A register stores a multi-bit value**
  - A collection of D-latches, all controlled by a common WE
  - WE = 1, n-bit value D is written to the register



4-bit register built with D-latch

Clock Signal

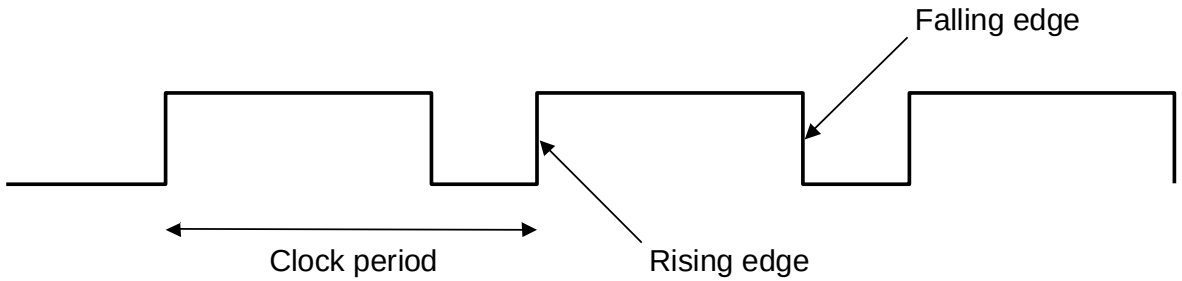
- A microprocessor is composed of many different circuits that are operating simultaneously – if each circuit X takes in inputs at time T_{in}, takes time T_{exe} to execute the logic, and produces outputs at time Tout, imagine the complications in coordinating the tasks of every circuit
- Solution: all circuits on the chip share a clock signal (a square wave) that tells every circuit when to accept inputs, how much time they have to execute the logic, and when they must produce outputs



Clock Terminology

- **A clock is a free-running signal with a fixed cycle time (called clock period) or, equivalently, a fixed clock frequency (i.e., inverse of the cycle time).**

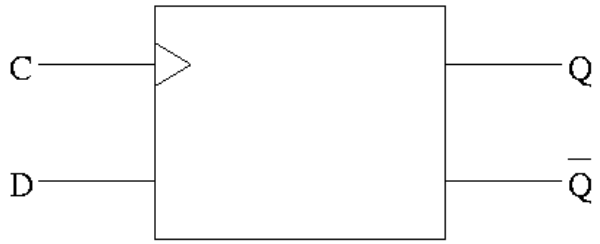
- **Edge-triggered clocking:**
  - Design methodology for sequential logic circuits in which all state changes occur on a clock edge (**rising edge** or **falling edge**).



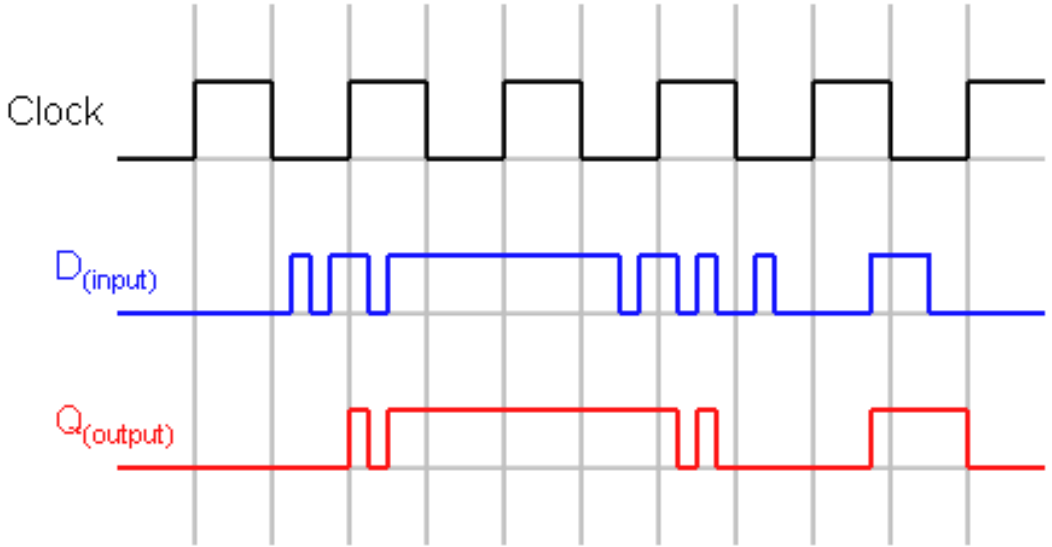
- **Example** $4\ GHZ = clock\ speed = \frac{1}{cycle\ time} = \frac{1}{250\ ps}$

Timing Diagram

- **Digital timing diagram**
  - A representation of a set of signals in the time domain
  - Can contain many rows, usually one of them being the clock
- **Example: timing diagram of D latch**
  - Use clock signal as WD

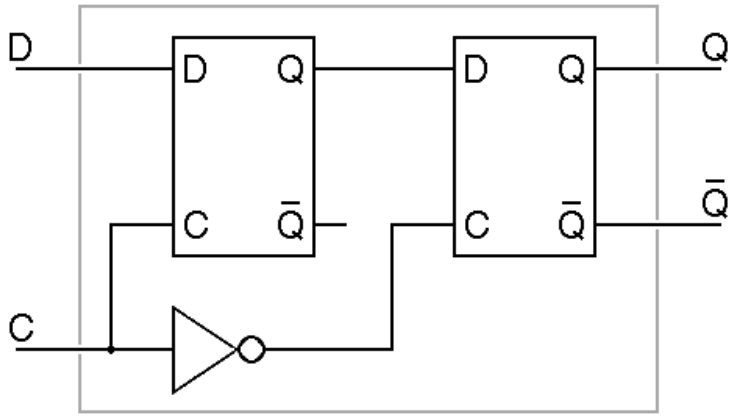


Graphic symbol
D latch with clock



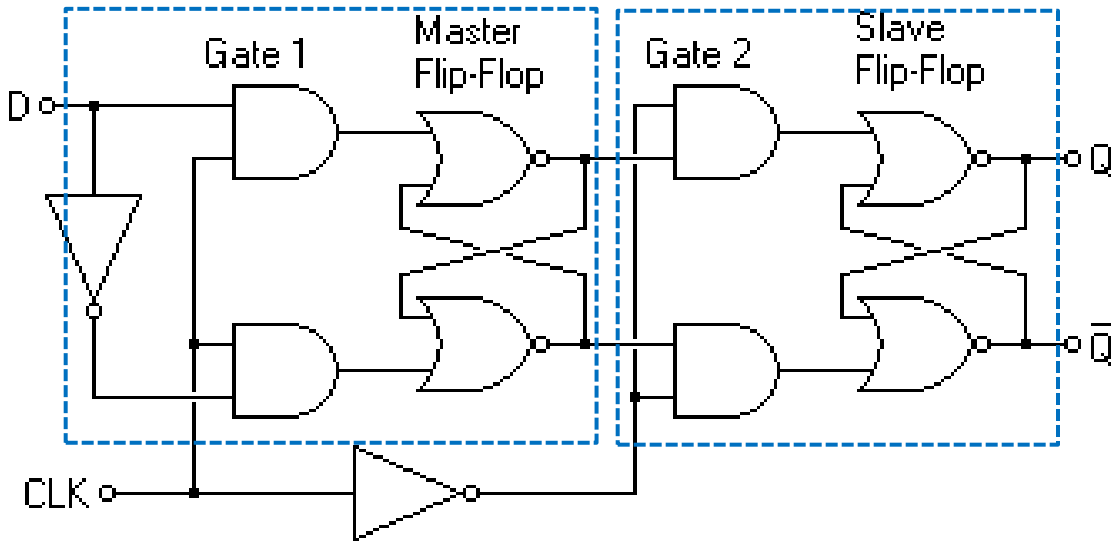
Master-Slave D Flip-Flop

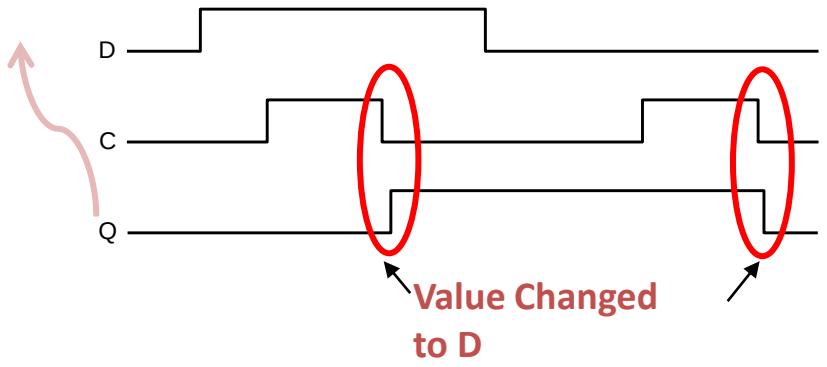
- Built by a pair of clock-gated D-latches
- Terminology:
  - Latch: outputs can change any time the clock is asserted (high)
  - Flip-flop: outputs can change only on a clock edge
- Example: Falling-edge-triggered master-slave D flip-flop
  - When the clock input C changes from asserted to de-asserted, the Q output stores the value of the D input.



Falling-edge-triggered Master-slave D Flip-flop

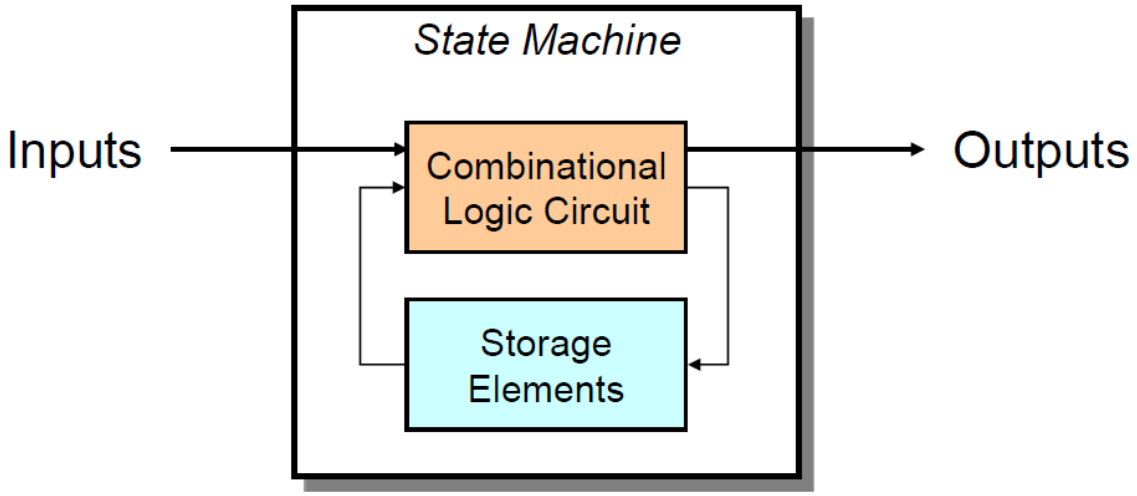
- When clock is asserted, D is updated in Master flip-flop
- When clock is de-asserted (1->0, falling edge), D is updated in Slave flip-flop, and is visible in Output Q





Sequential Circuit

- Combines combinational logic with storage
- 'Remembers' state, and changes output (and state) based on inputs and current state
- The state of a system is a snapshot of all the relevant elements of the system at the moment the snapshot is taken
- Frequently, a clock circuit triggers transition from one state to the next



Concluding Remarks

- Logic operation: NOT, AND, OR, NAND, NOR, XOR
- Logic gate
- Truth table and logic function
- Combinational vs. sequential circuit
- Building blocks of combinational logic: Decoder/Encoder, Multiplexor, Demultiplexor
- Two-level logic: sum-of-product and product-of-sum
- PLA
- Boolean Algebra
- Karnaugh-Maps (K-maps)

- S-R latch
- Gated D-latch
- Clock and timing diagram
- D flip-flop