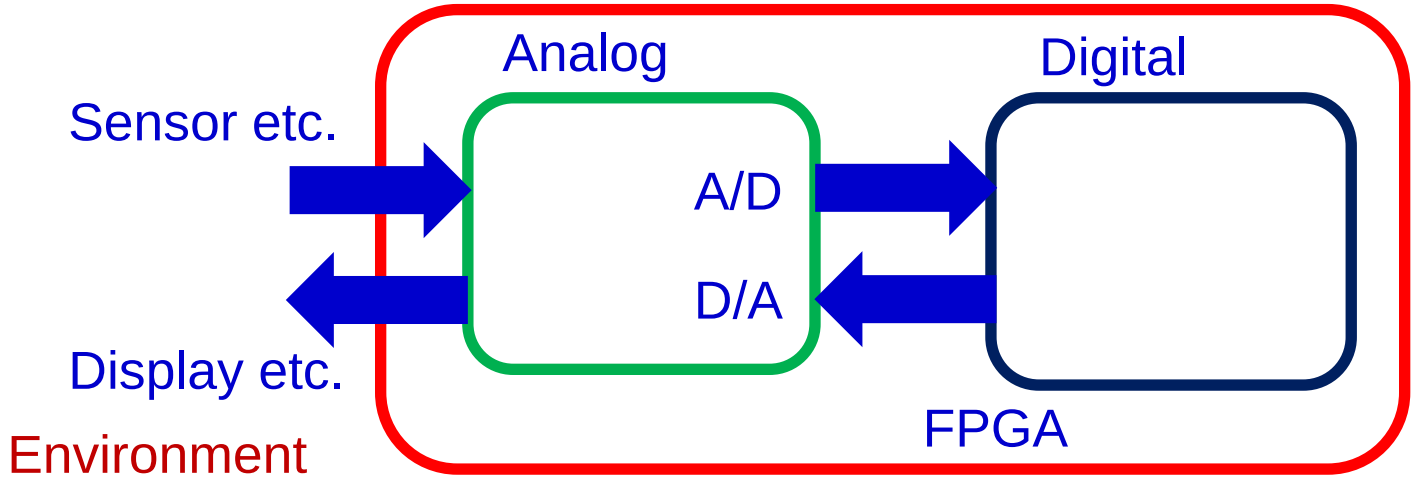
Digital Integrated Circuits 2: Synthesis

Atsushi Takahashi
Department of Information and Communications Engineering
School of Engineering
Tokyo Institute of Technology

VLSI and Computer System

- VLSI (Very Large Scale Integrated Circuits)
- Computer System



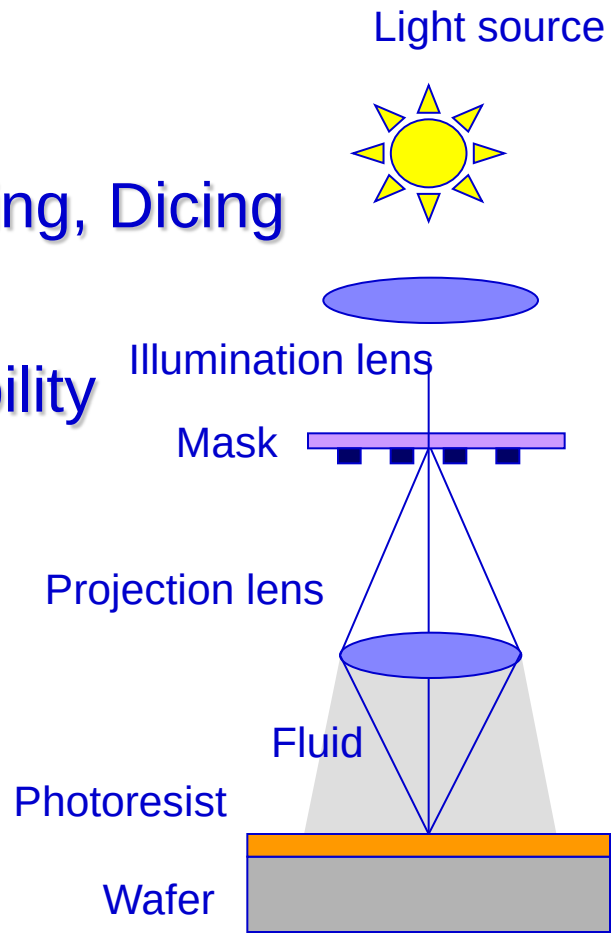
Computer System / Embedded System / VLSI …

Architecture : Microprocessor / Deep Neural Network …
Hardware : Logic Functions, Hardware Description Language …
Software : Algorithm, Real Time Operating System …
Design : Synthesis, Compiler, Physical Design …

VLSI Design / Manufacturing

Integration of Various Technologies

- Device Manufacture
  - Make transistors small
  - Mask Design, Exposure, Polishing, Dicing
- Circuit Design, Layout Design
  - High Speed, Low Power, Reliability
- Packaging, Printed Circuit Board
  - Wire Bonding
- System Design
- Software Design
- Marketing



VLSI Design (Synthesis)

- Design Automation
  - Essential in design productivity improvement
- Problem Definition
  - Inputs, outputs, and objectives
  - Design flow and Hierarchical synthesis
    - ✓ many sub-problems
  - ✓ Optimum solution for sub-problem may not be good for whole problem
  - Need to update Design methodology and Design flow

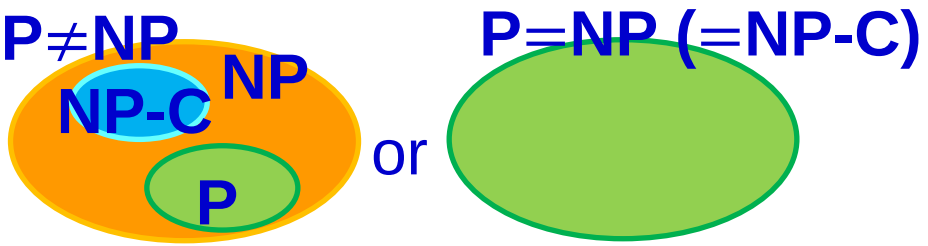
- Problem : Find an optimum solution
  - Is there an exact algorithm for the problem?
    - ✓ Yes (in most cases for combinatorial problem)
    - Enumerating all the cases and pick a best one
      - Impractical for large instances
  - Is there a practical exact algorithm for the problem?
    - ✓ NO (except limited cases)
    - Need sophisticated intelligent approach
      - Heuristic in most cases

Complexity Theory

- Background of Algorithm Design
  - **P** and **NP**
  - **NP**-complete
  - **NP**-hard
  - Polynomial Time Reduction
  - Nondeterministic Polynomial Time Algorithm
  - (Deterministic) Polynomial Time Algorithm

NP-completeness

- Decision Problem (Yes/No Problem)
  - **NP** : Set of Decision problems that have
    - Nondeterministic Polynomial time algorithm
  - **P**: Set of Decision problems that have
    - (Deterministic) Polynomial time algorithm
  - **NP-C**: hardest problems in **NP**
    - If a problem in **NP-C** can be solved in polynomial time, then any problem in **NP** can be solved in polynomial time
    - A decision problem is said to be **NP**-complete if it is in **NP-C**
    - SAT, 3-SAT, COL, HG, IS, TS, …
- Fact: **P**⊆ **NP**
- Conjecture: **P** ≠ **NP**



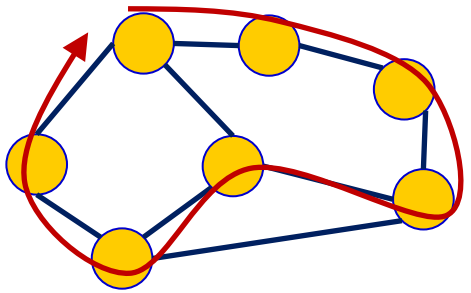
[Garey and Johnson, "Computers and Intractability, A Guide to the Theory of NP-Completeness", Freeman and Co., 1979]

Nondeterministic Polynomial Time Algorithm

- Typical Structure
  - Step 1 (nondeterministic)
    - Generate a evidence in polynomial time
      (Pick up one arbitrary among exponential candidates)
  - Step 2 (deterministic)
    - Check the evidence in polynomial time
      - If the evidence is correct, then output YES
      - If the evidence is incorrect, then output NO
- Behavior of Correct Nondeterministic Algorithm

Correct Answer		Algorithm Output
YES	⇔	YES
No	⇔	No

Problem: Is Graph a Hamiltonian?



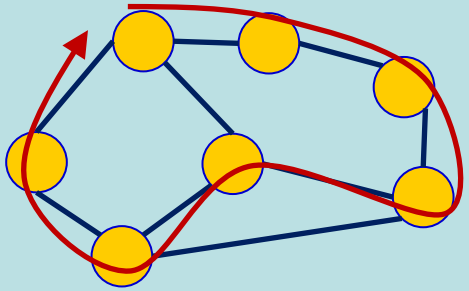
Evidence : sequence of vertices

Evidence (Proof) for YES

- Problem: Is Graph Hamiltonian?
  - **NP**
  - An evidence that shows the graph is Hamiltonian which can be checked in polynomial time exists
- Problem: Is NOT Graph Hamiltonian?
  - **NP ???**
  - An evidence that the graph is not Hamiltonian is not trivial
  - What is an evidence that shows the graph is not Hamiltonian?

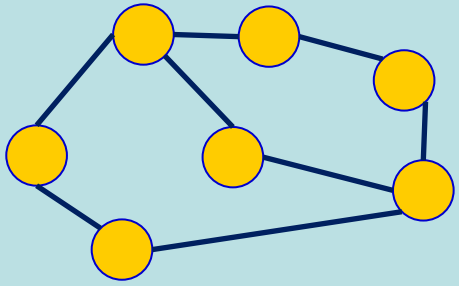
Problem: Is Graph a Hamiltonian?

Evidence : sequence of vertices



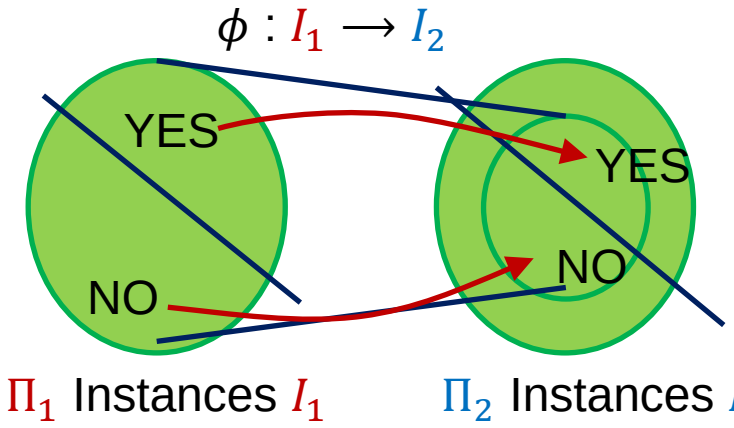
Problem: Is NOT Graph a Hamiltonian?

Evidence : ???



Polynomial Time Reduction

- Provides difficulty relation between problems
  - Which is not difficult?
- Polynomial Time Recution of Problem ($\Pi_1 \propto \Pi_2$)
  - Instance of Π_1 can be converted to instance of Π_2 in polynomial time while maintaining Yes/No property
  - Problem Π_1 can be solved in polynomial time by utilizing (hypothetical) polynomial time algorithm for problem Π_2
    - If Π_2 is solved in polynomial time, then Π_1 can be solved in polynomial time
  - Π_2 is not easier than Π_1 (same or more difficult)



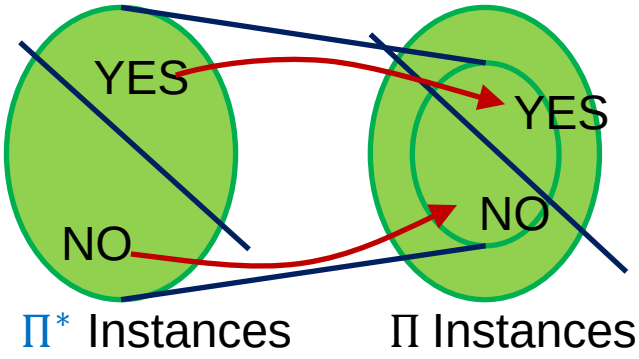
Π_1	$\propto$	Π_2
Easy	⟷	Easy
Difficult	⟷	Difficult

Proof of NP-completeness

- **NP**-complete problem Π^*: $^{\forall}\Pi \in$ **NP**, $\Pi \propto \Pi^*$
  - Not easier than any problem in **NP**

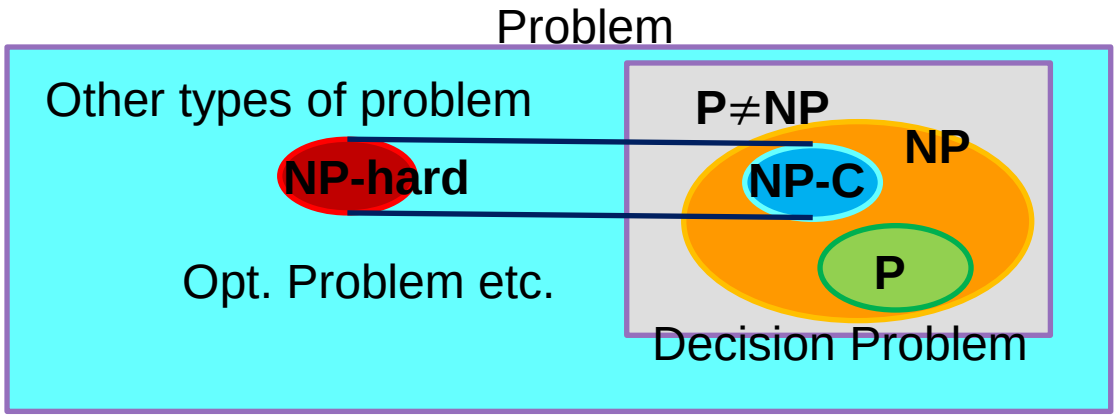
Π	$\propto$	Π^*
Easy	⇔	Easy
Difficult	⇔	Difficult

- Proof of **NP**-completeness of Decision Problem Π
  - Pick up **NP**-complete problem Π^*
  - Show $\Pi^* \propto \Pi$
    - Π is not easier than Π^*, that means Π is **NP**-complete
- Incorrect Proof
  - Show $\Pi \propto \Pi^*$
  - It is trivial by definition
  - It does not mean that Π is **NP**-complete



NP-hardness

- Optimization problem
  - is neither in **NP** nor in **NP-C**
  - is not said to be **NP**-complete
  - Is said to be **NP**-hard if a related decision problem is **NP**-complete
- No polynomial time algorithm for **NP**-hard problem if **P≠NP**
- If a problem is **NP**-hard, then
  - Approximation algorithm or Heuristic algorithm are pursued



[Garey and Johnson, "Computers and Intractability, A Guide to the Theory of NP-Completeness", Freeman and Co., 1979]

First Step of Algorithm Design

- Check whether problem is easy or not?
  - Difficult = **NP**-hard, **NP**-complete
    - Design heuristic
  - Easy = **P** (or decision version is in **P**)
    - Design exact polynomial time algorithm
    - Reduce time and space complexity
- Most of practical problems are difficult
  - **NP**-hardness seems trivial ....
    but proof of **NP**-hardness is not easy
  - So, proof is often skipped, recently
- In the following
  - **P** = problem solvable in polynomial time

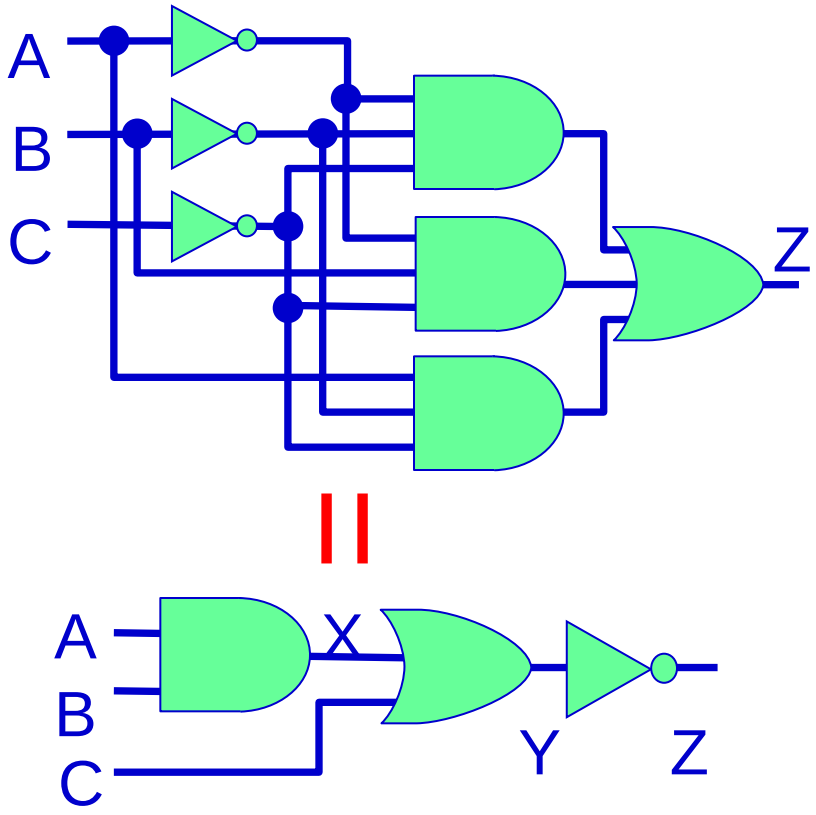
Logic Synthesis

- Get a small logic circuit that realizes a given Boolean function
  - NP-hard problem
- Smallest logic circuit is not necessarily optimum
  - Objectives are Size, Delay, Power and etc.

- Two-level logic circuit synthesis
  - NOT-AND-OR (Sum-of-Products) form
- Multi-level logic circuit synthesis
- Sequential logic circuit synthesis

Equivalent Implementation

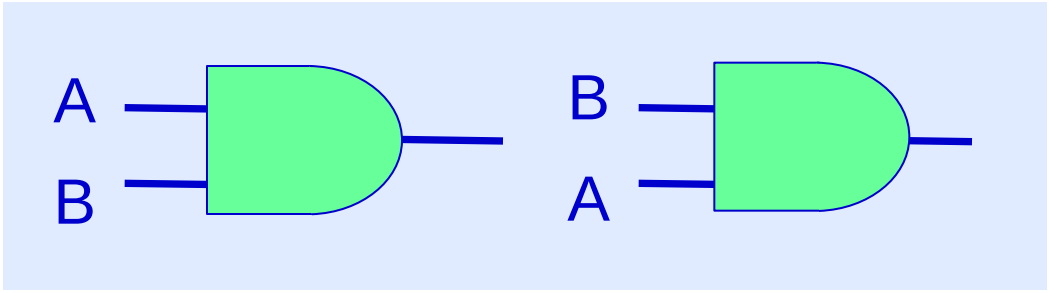
- What is the best implementation?
  - depends … varies …

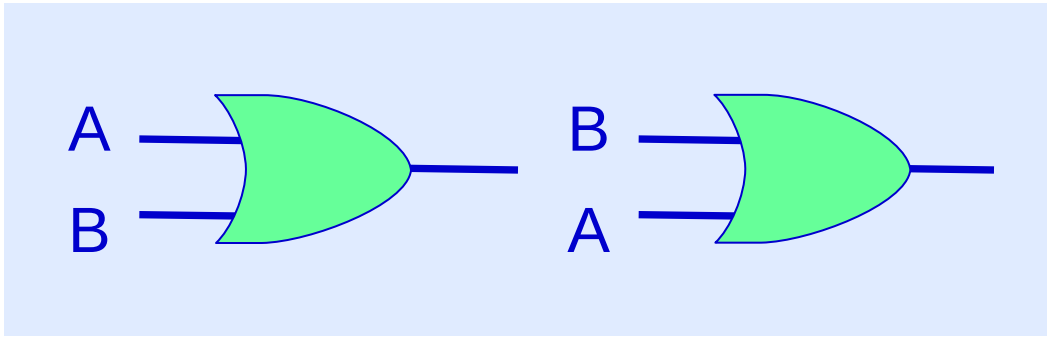


ABC	X	Y	Z
000	0	0	1
001	0	1	0
010	0	0	1
011	0	1	0
100	0	0	1
101	0	1	0
110	1	1	0
111	1	1	0

Boolean Logic Properties

- Commutativity
  - A B = B A
  - A + B = B + A





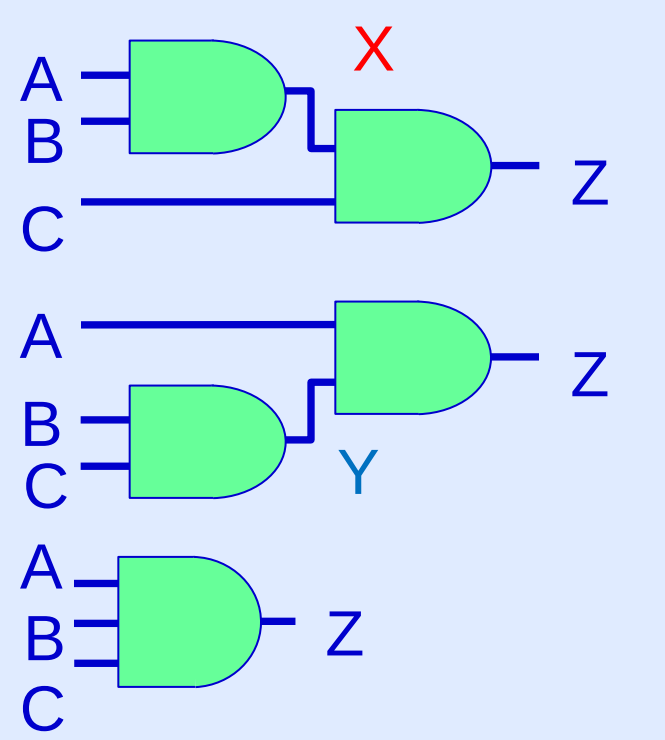
Truth Table

A	B	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

B	A	AND	OR
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	1

Boolean Logic Properties (2)

- Associativity
  - (A B) C = A (B C) = A B C
  - (A + B) + C = A + (B + C) = A + B + C



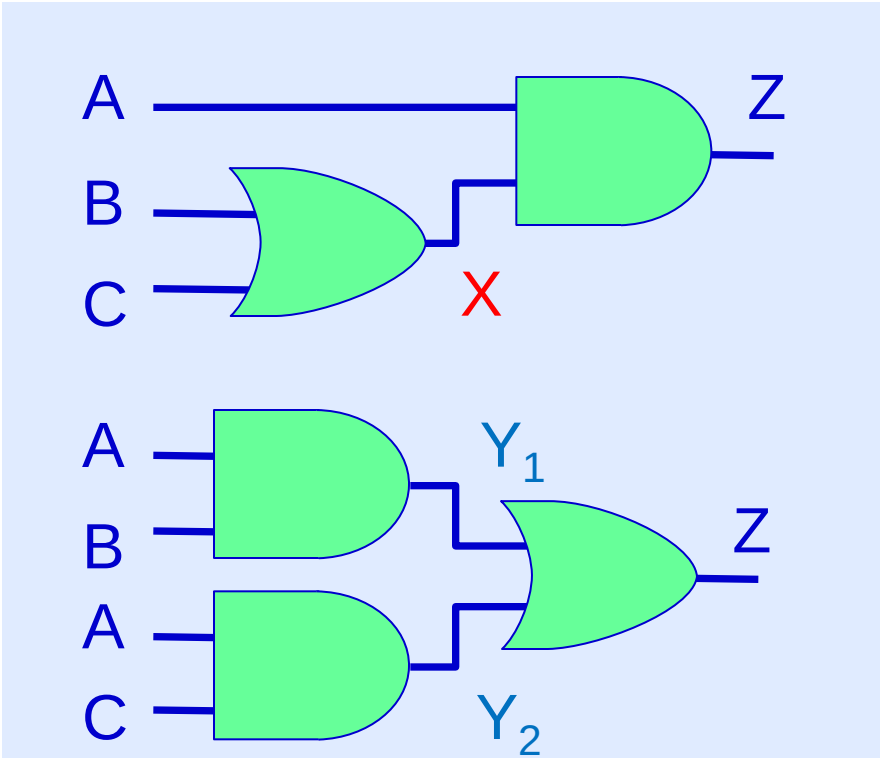
ABC	X	Y	AND
000	0	0	0
001	0	0	0
010	0	0	0
011	0	1	0
100	0	0	0
101	0	0	0
110	1	0	0
111	1	1	1

ABC	X	Y	OR
000	0	0	0
001	0	1	1
010	1	1	1
011	1	1	1
100	1	0	1
101	1	1	1
110	1	1	1
111	1	1	1

Boolean Logic Properties (3)

- Distributivity 1
  - A (B + C) = (A B) + (A C)

a(b+c) = ab+ac



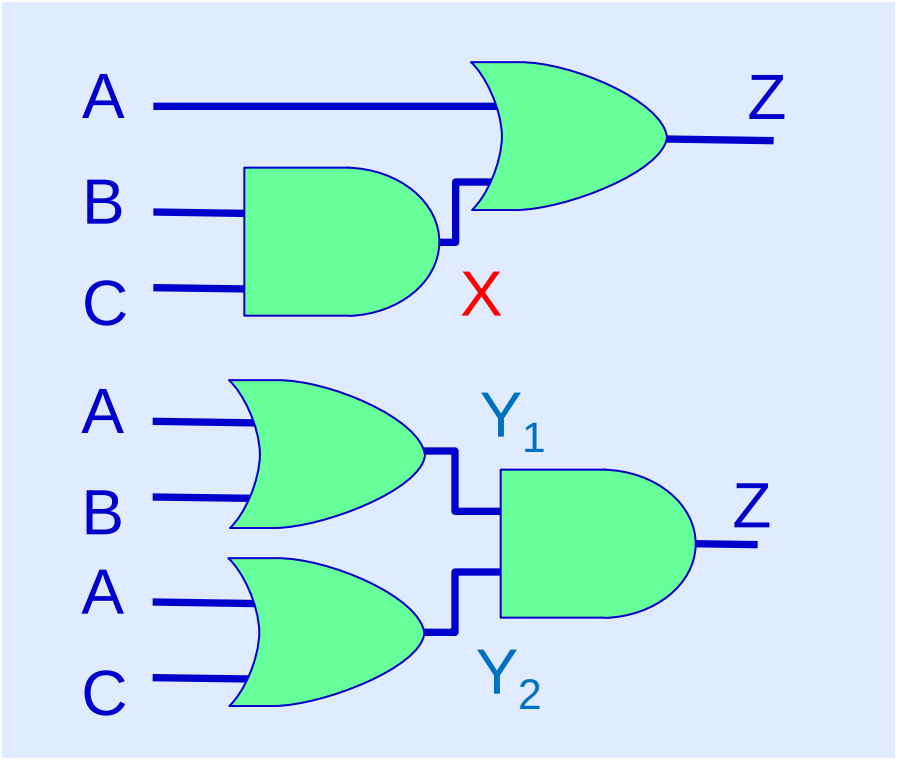
Truth Table

ABC	X	Y_1	Y_2	Z
000	0	0	0	0
001	1	0	0	0
010	1	0	0	0
011	1	0	0	0
100	0	0	0	0
101	1	0	1	1
110	1	1	0	1
111	1	1	1	1

Boolean Logic Properties (4)

- Distributivity 2
  - A + (B C) = (A + B) (A + C)

a+bc != (a+b)(a+c)



Truth Table

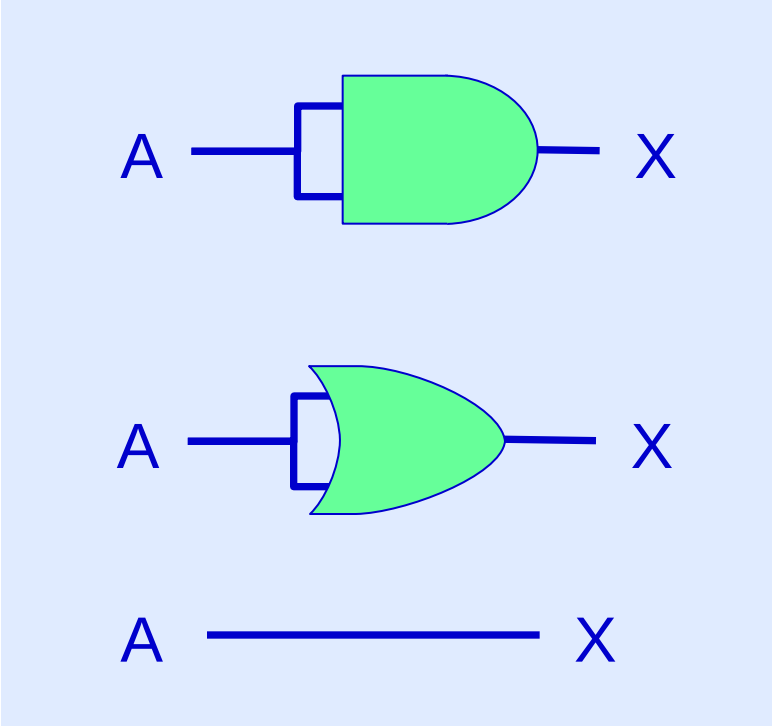
ABC	X	Y_1	Y_2	Z
000	0	0	0	0
001	0	0	1	0
010	0	1	0	0
011	1	1	1	1
100	0	1	1	1
101	0	1	1	1
110	0	1	1	1
111	1	1	1	1

Boolean Logic Properties (5)

- Idempotency
  - A A = A
  - A + A = A

$a * a = a^2$

$a + a = 2a$

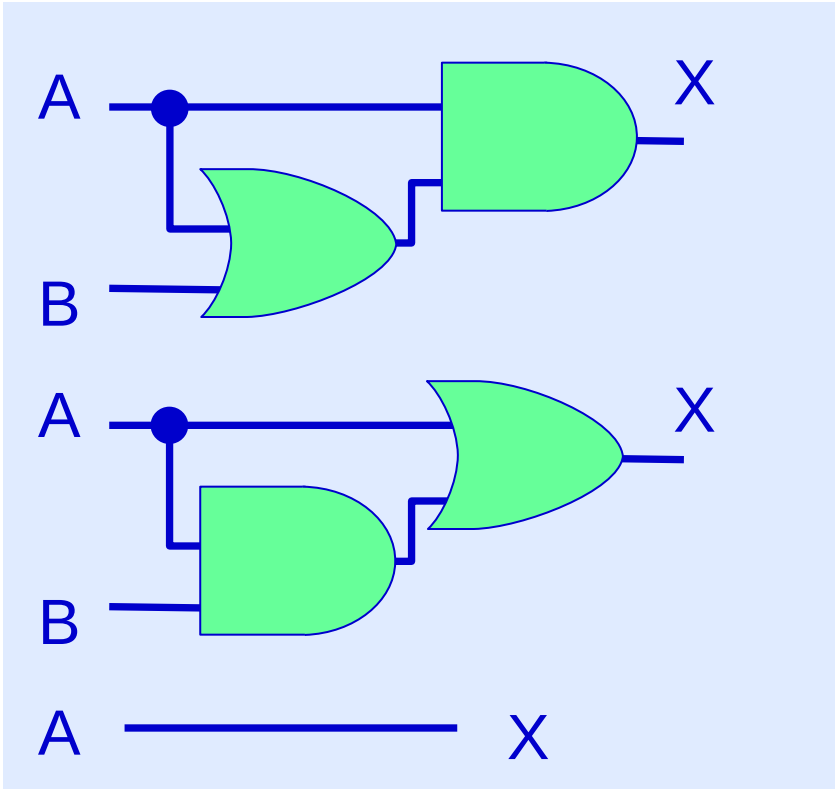


Truth Table

A	B	AND	OR	X=A
0	0	0	0	0
0	1	0	1	
1	0	0	1	
1	1	1	1	1

Boolean Logic Properties (6)

- Absorptivity
  - A (A + B) = A + (A B) = A



Truth Table

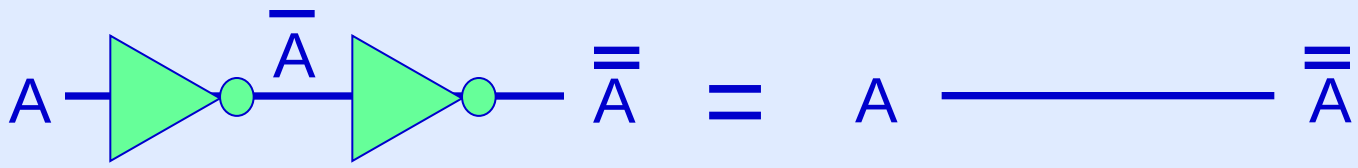
A	B	AND	OR	X=A
0	0	0	0	0
0	1	0	1	0
1	0	0	1	1
1	1	1	1	1

Boolean Logic Properties (7)

- NOT Operation
  - $A \vee \overline{A} = 1$
  - $A \wedge \overline{A} = 0$
- Double Negative Elimination
  - $\overline{\overline{A}} = A$

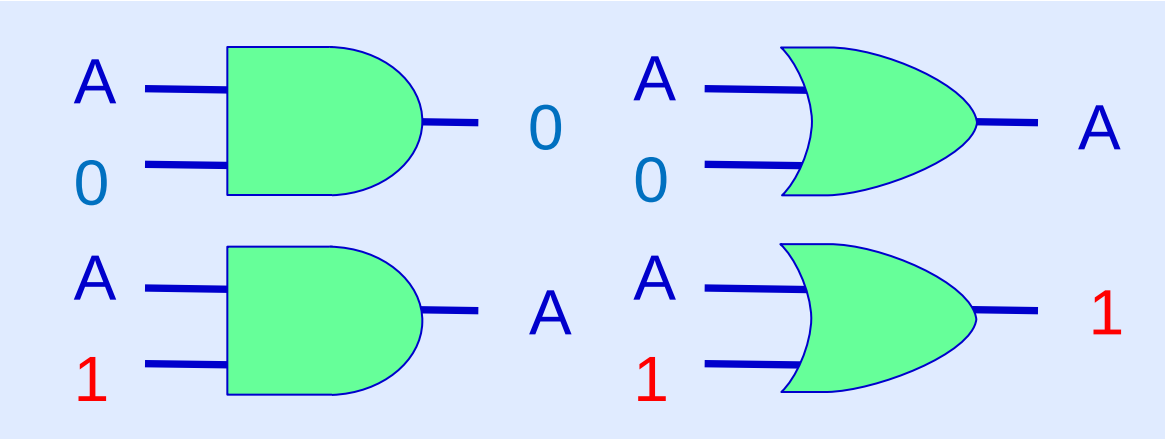
Truth Table

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



Boolean Logic Properties (8)

- Constant Operation
  - A 0 = 0
  - A 1 = A
  - A + 0 = A
  - A + 1 = 1

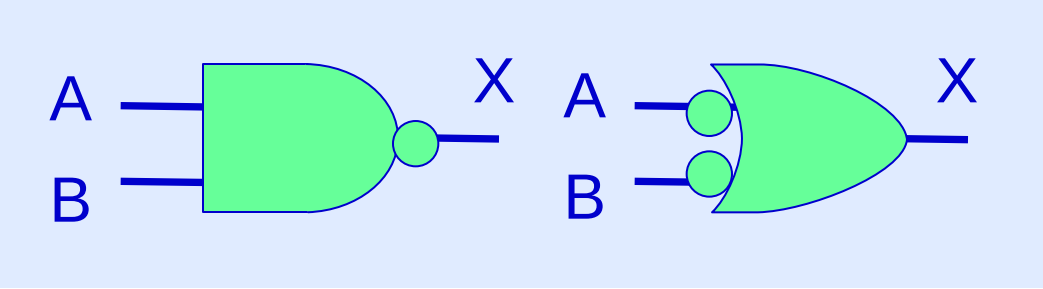


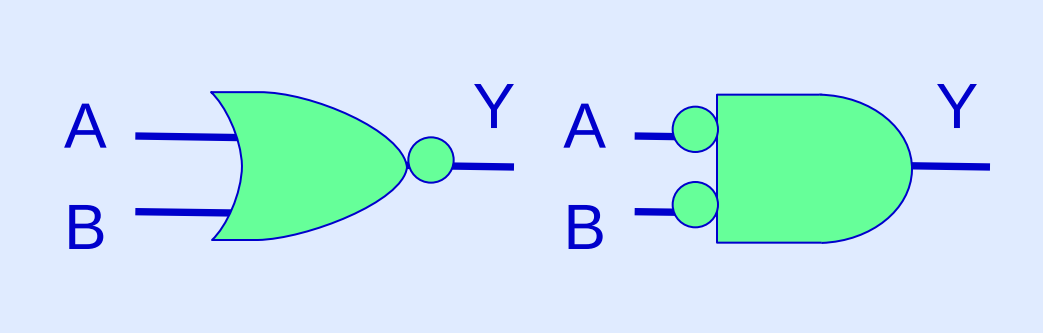
Truth Table

A	B	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Boolean Logic Properties (10)

- De Morgan's Law
  - $\overline{A\ B} = \overline{A} + \overline{B}$
  - $\overline{A + B} = \overline{A}\ \overline{B}$

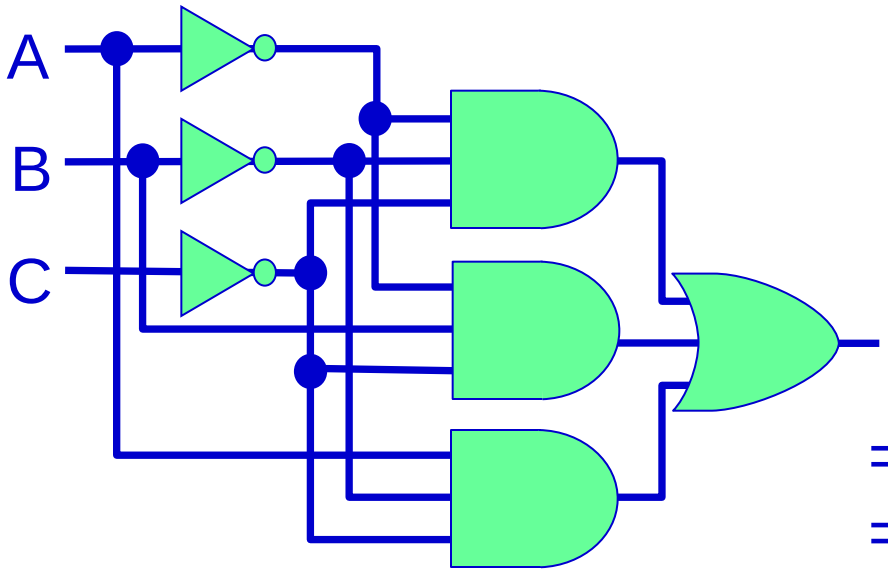




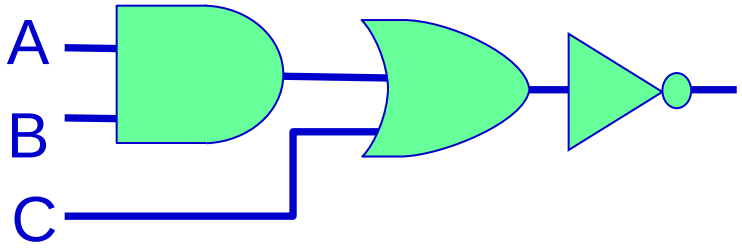
Truth Table

A	B	$\overline{A}$	$\overline{B}$	X	Y
0	0	1	1	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	1	0	0	0	0

Equivalent Transformation



II



$$A\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C}$$
$$= (A\bar{B} + \bar{A}B + A\bar{B})\bar{C}$$
$$= (A\bar{B} + \bar{A}B + A\bar{B} + AB)\bar{C}$$
$$= ((A\bar{B} + \bar{A}B) + (A\bar{B} + AB))\bar{C}$$
$$= (\bar{A}(\bar{B} + B) + (\bar{A} + A)\bar{B})\bar{C}$$
$$= (\bar{A} + \bar{B})\bar{C}$$
$$= \overline{AB}\,\bar{C}$$
$$= \overline{AB + C}$$

Canonical Representation

- Canonical representation
  - Unique representation
- Disjunctive (NOT-AND-OR) Normal Form

  e.g. $\overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + A\,\overline{B}\,\overline{C}$
- Conjunctive (NOT-OR-AND) Normal Form

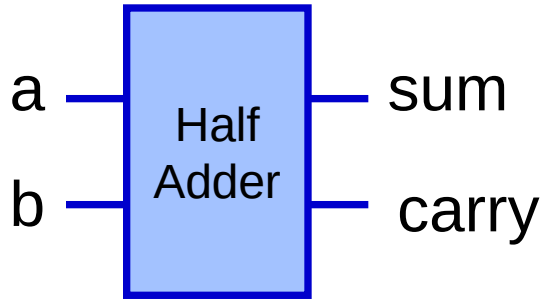
  e.g. $(\overline{A} + \overline{B} + \overline{C})(A \vee \overline{B} \vee C)(A \vee B \vee C)$
  - Representation is unique when the variable order is fixed
  - The circuit size is exponential in terms of the number of input variables in general

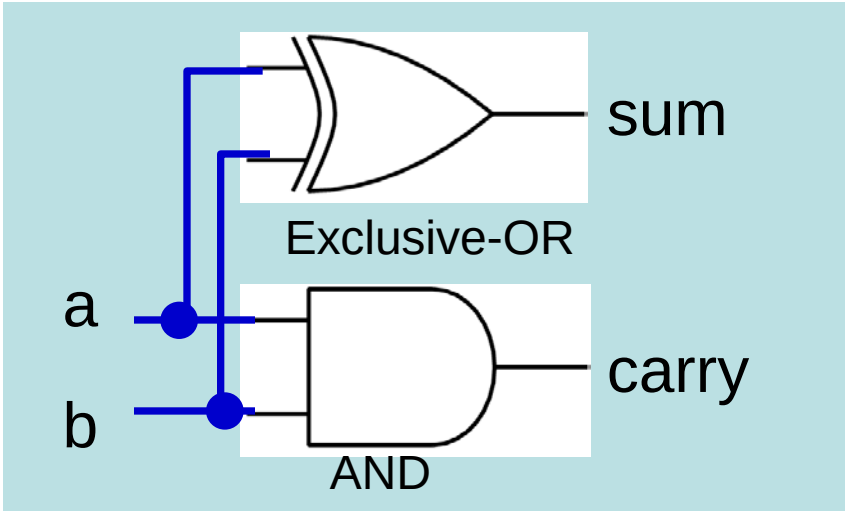
Example : Summation

- 1-bit Adder (Half-Adder)
- A+B
  - A + B = C (a + b = c s)
  - 0 + 0 = 0 (0 + 0 = 00)
  - 0 + 1 = 1 (0 + 1 = 01)
  - 1 + 0 = 1 (1 + 0 = 01)
  - 1 + 1 = 2 (1 + 1 = 10)

Truth Table

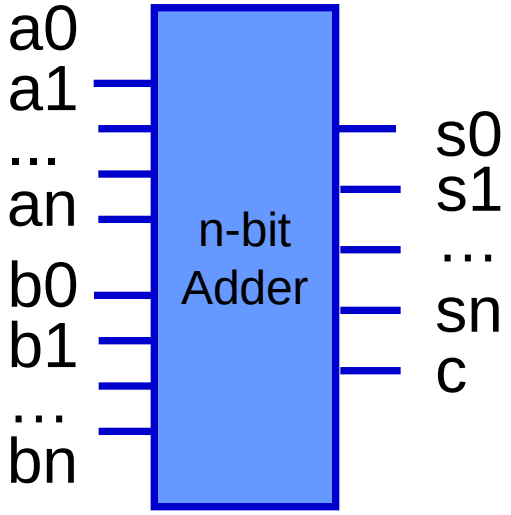
a	b	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0





n-bit Adder

- n-bit adder
  - A : an … a2 a1 a0
  - B : bn … b2 b1 b0
- Design by NOT-AND-OR form for each output becomes too big
- Hierarchical Design
  - Use 1-bit adders
  - Input has carry from lower bit



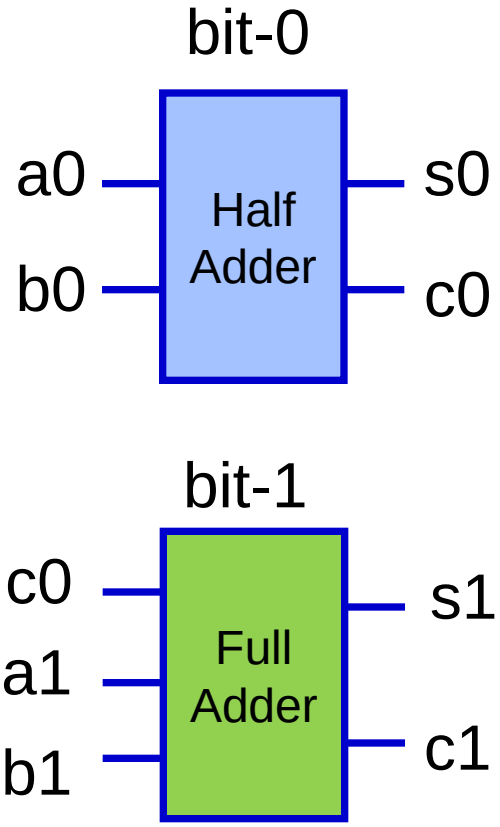
Hierarchical Design of Adder

- n-bit Adder
  - A : an … a2 a1 a0
  - B : bn … b2 b1 b0

```
        c0
        a1  a0
  +     b1  b0
-------------
  c1    s1  s0
```

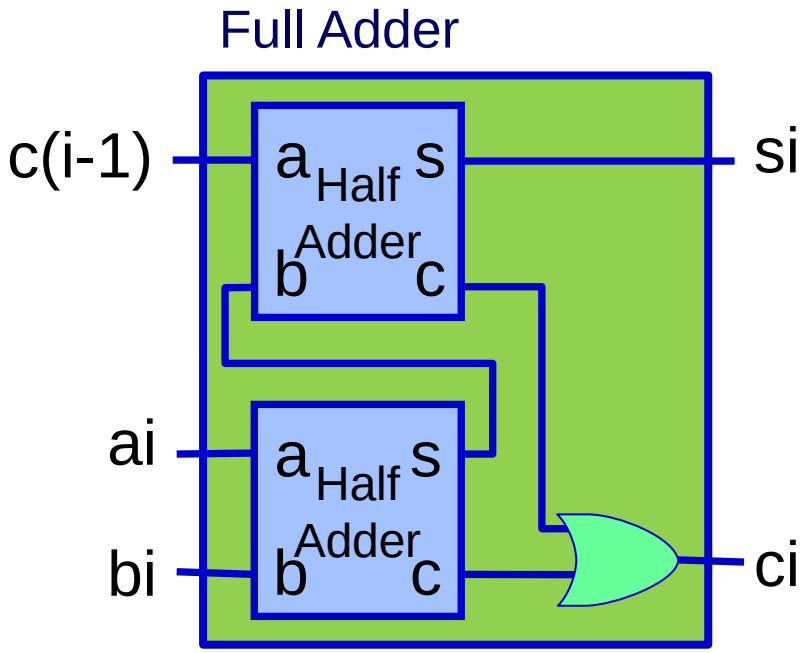
bit-0 : (c0,s0) = a0 + b0

bit-1 : (c1,s1) = a1 + b1 + c0



Full Adder

- 1-bit adder with carry
  - Input includes a carry from lower bit

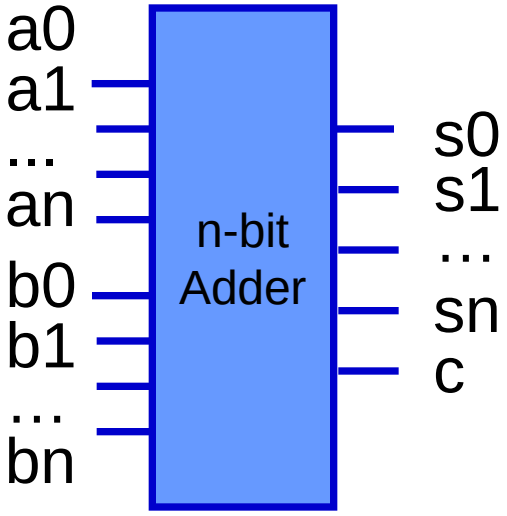


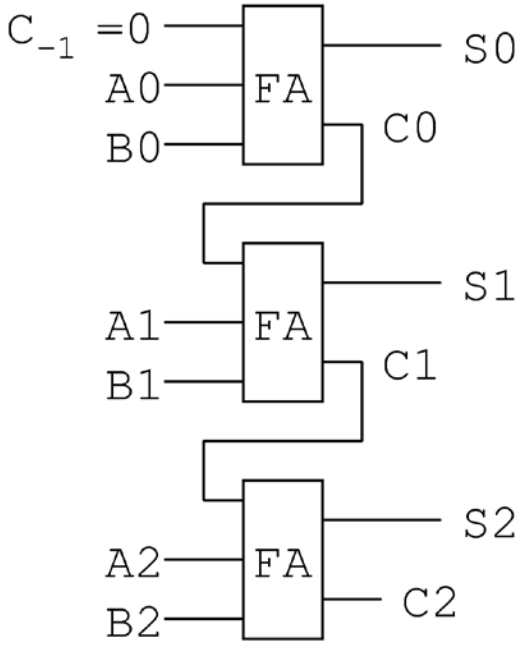
Truth Table

c(i-1)	ai	bi	ci	si
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

n-bit Adder (RCA)

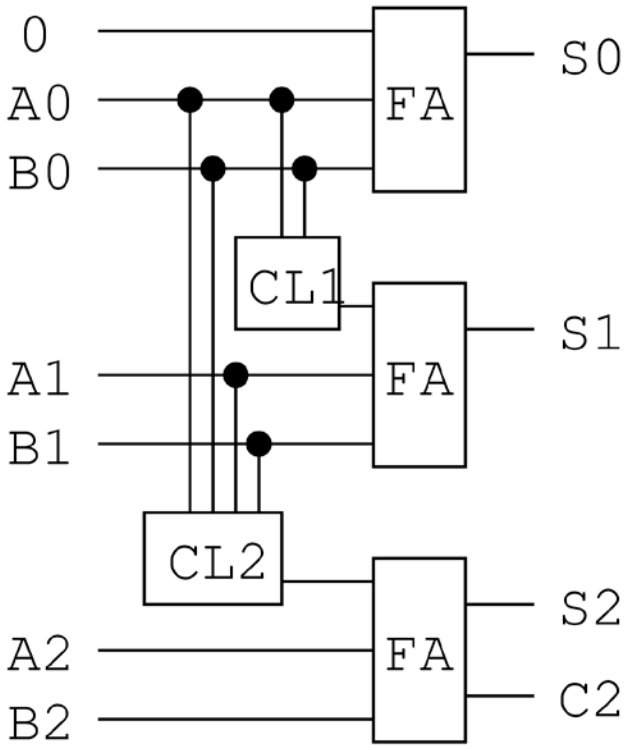
- Ripple carry adder
  - Serially connected n full-adder
  - Carry signal may propagate along FAs
  - Small size, but slow speed





n-bit Adder (CLA)

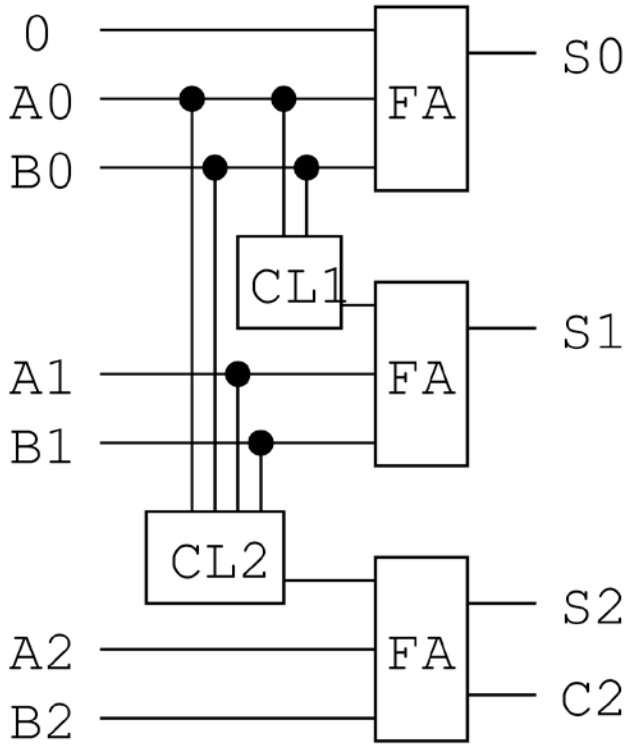
- Carry Look-ahead Adder
  - Logic that detects whether carry propagates or not is inserted
  - large size, but fast



n-bit Adder (CLA) (2)

Truth Table

c(i-1)	ai	bi	ci	si
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



CL1 = a0 ^ b0

CL2 = (a1 ^ b1) v (c0 ^ a1) v (c0 ^ b1)
= (a1 ^ b1) v (a0 ^ b0 ^ a1) v (a0 ^ b0 ^ b1)

Logic Minimization

- The size of a circuit is estimated by the number of transistors in the circuit
- The length of formula (the number of literals) corresponds to the number of transistors
- The minimization of the length of formula
  - NP-hard

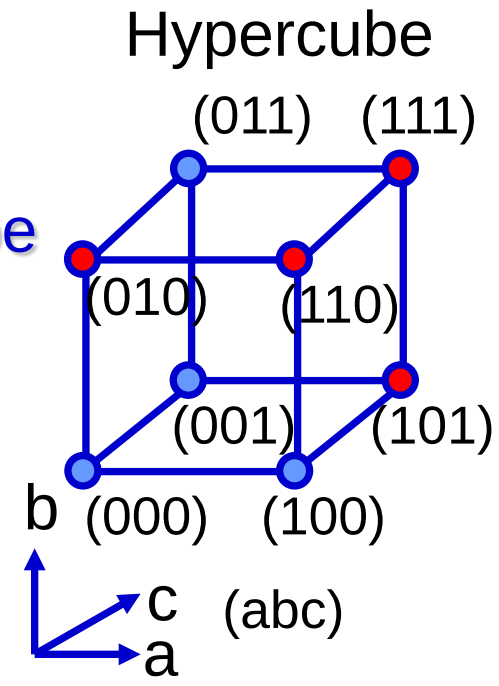
CL2 = (a1 ^ b1) v (c0 ^ a1) v (c0 ^ b1)
= (a1 ^ b1) v (a0 ^ b0 ^ a1) v (a0 ^ b0 ^ b1)

- Basic terminology
  - Complement of variable x : x', $\overline{x}$, ¬x
    - x = 0 → x' = 1
    - x = 1 → x' = 0
  - Literal
    - Boolean variable x or its complement x'

Boolean Function Representation

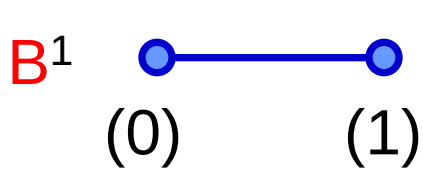
- Boolean Function f : $B^N \rightarrow B$
  - Input variables : $x_1, x_2, \ldots, x_N$ (in B)
  - Output variable : f (in B)

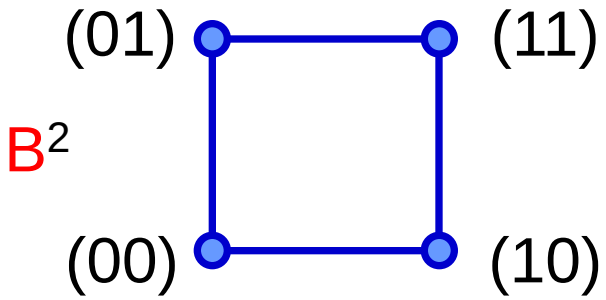
  - Input vector **x** = $(x_1, x_2, \ldots, x_N)$
    - A minterm
    - A node of N-dimensional hypercube
      - (00…0), (00…1), …, (11…1)
    - ON-set : **x** such that f(**x**)=1
    - OFF-set : **x** such that f(**x**)=0

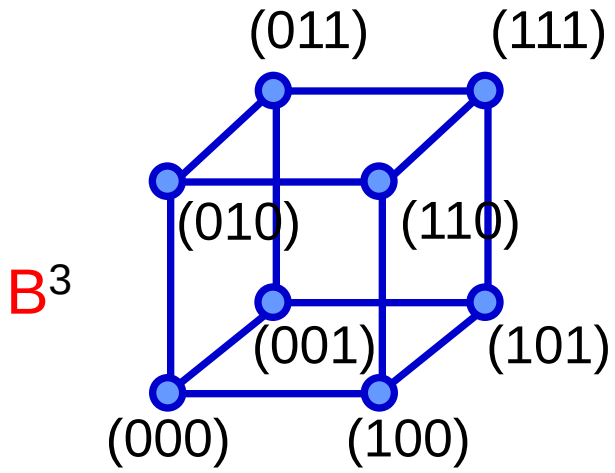


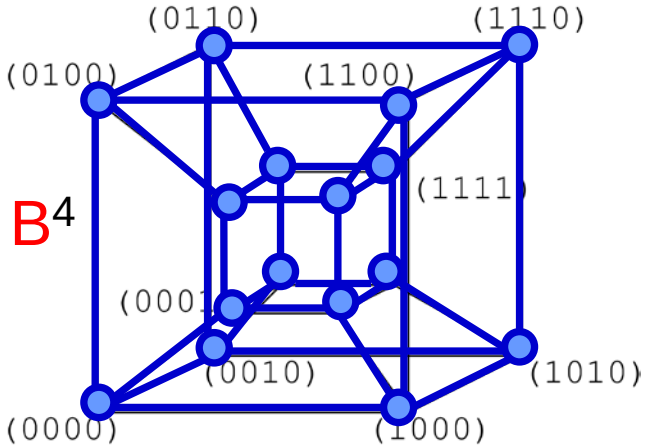
Boolean Space

- Boolean space B^N
  - represented by N-dimensional hypercube
    - edge iff Hamming distance = 1









Karnaugh Map

- For Two-level Boolean Logic Minimization
  - by manual for small functions
- Karnaugh map
  - Each square corresponds to a minterm
  - Hamming distance of adjacent squares is 1

a / b	0	1
0	1	0
1	0	1

f = ab v a'b'

ab / c	0	1
00	0	0
01	1	0
11	1	1
10	0	1

f = ac v bc'

ab / cd	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	0	1	1	0
10	0	0	1	1

f = a'b v bd v b'c

Karnaugh Map (2)

- Logic Minimization
  - Find a maximal rectangle in Karnaugh Map
    - Should correspond a hypercube that consists of ON-set

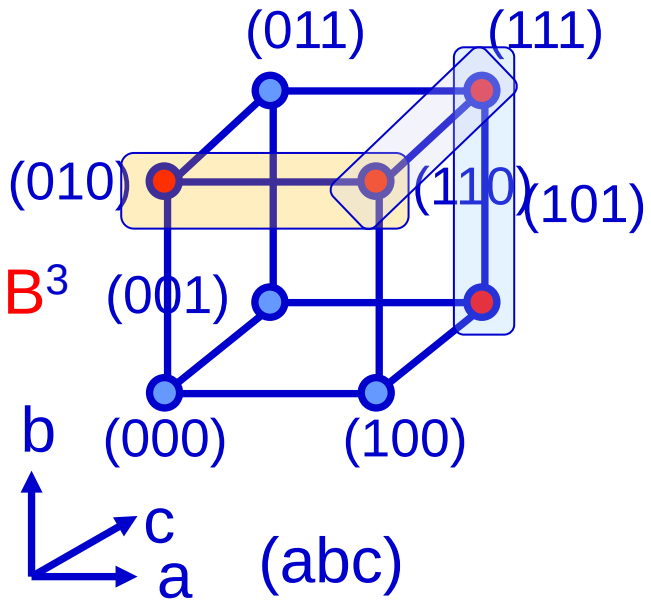
Truth table

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	1

Karnaugh Map

ab / c	0	1
00	0	0
01	1	0
11	1	1
10	0	1

Hypercube



f = a'bc' v ab'c v abc' v abc = ac v bc'

Karnaugh Map (3)

- Help to find Factoring of Two-level logic function

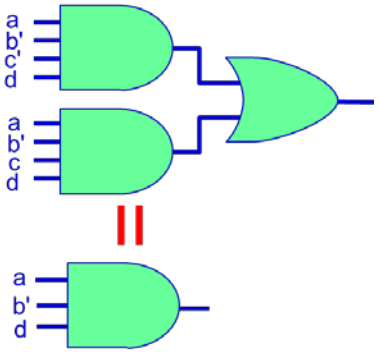
a’bc’d v ab’cd = ab’d(c+c’)= ab’d

ab / cd	00	01	11	10
00				
01				
11				
10				

ab’c’d 1001

ab’cd 1011

ab’d 10-1



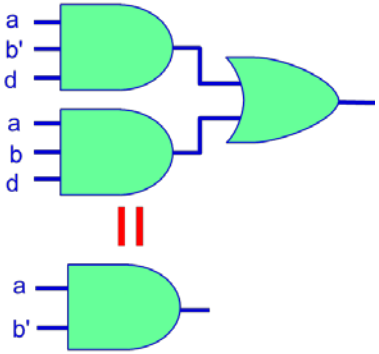
a’bd v ab’d = ab’(d+d’)= ab’

ab / cd	00	01	11	10
00				
01				
11				
10				

ab’d’ 10-0

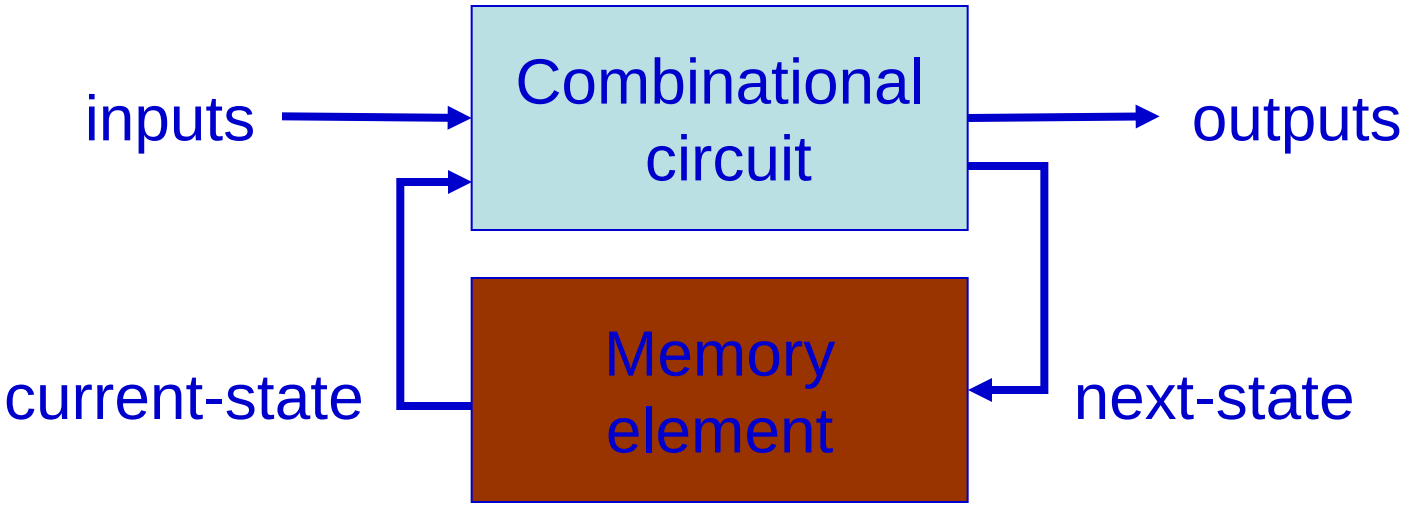
ab’d 10-1

ab’ 10--



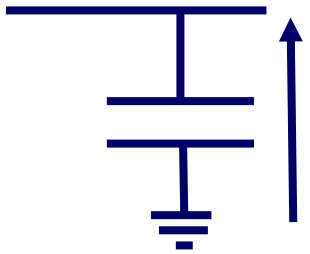
Sequential Circuit Implementation

- Performance depends on
  - State-machine itself
  - Time to transit from one state to another
    - Correct output must be recognized
    - Correct state must be stored

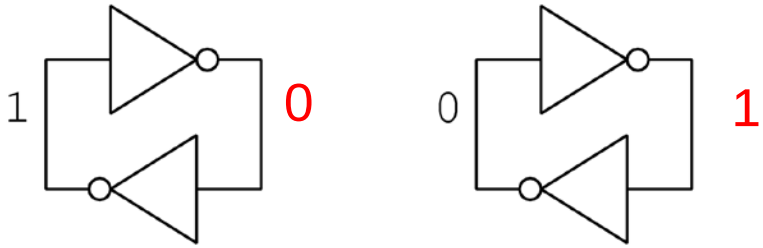


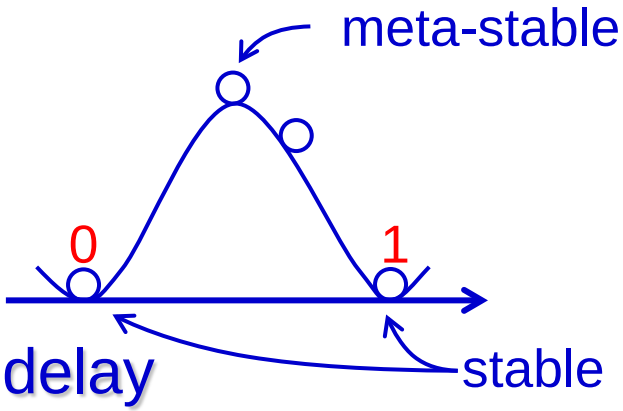
Memory Function

- Dynamic Memory (Capacitance)
  - Data lifetime is limited due to leakage

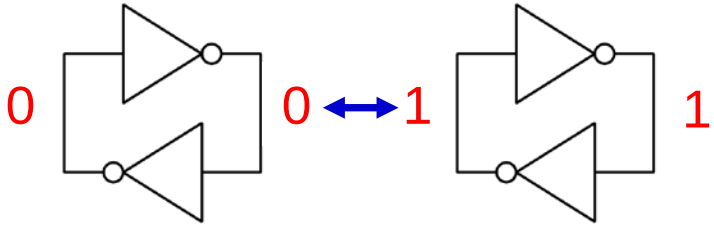


- Static Memory (Inverter loop)
  - Two stable states



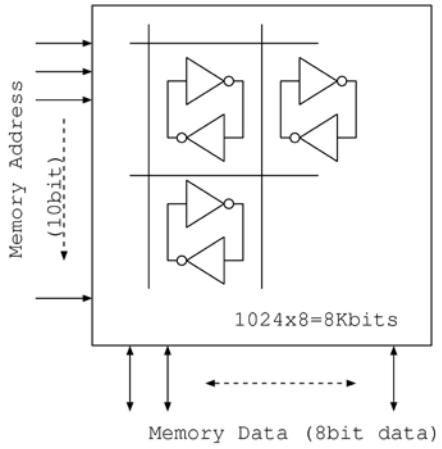


  - Meta-stable state might occur by delay
    - Continuous signal transitions



Memory

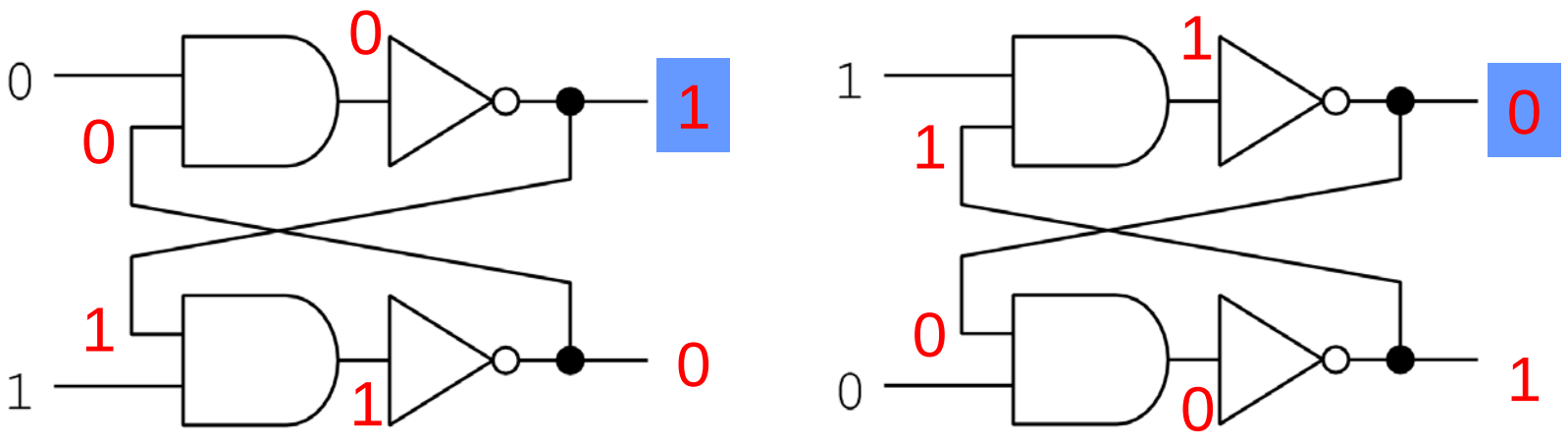
- Data Storage
  - Optimized to achieve high speed, small area, low power, etc
    - Dynamic Random Access Memory (DRAM)
    - Static Random Access Memory (SRAM)
    - Flash Memory



- Temporal memory for computation
  - High speed oriented
    - Latch, Flip-Flop, Register

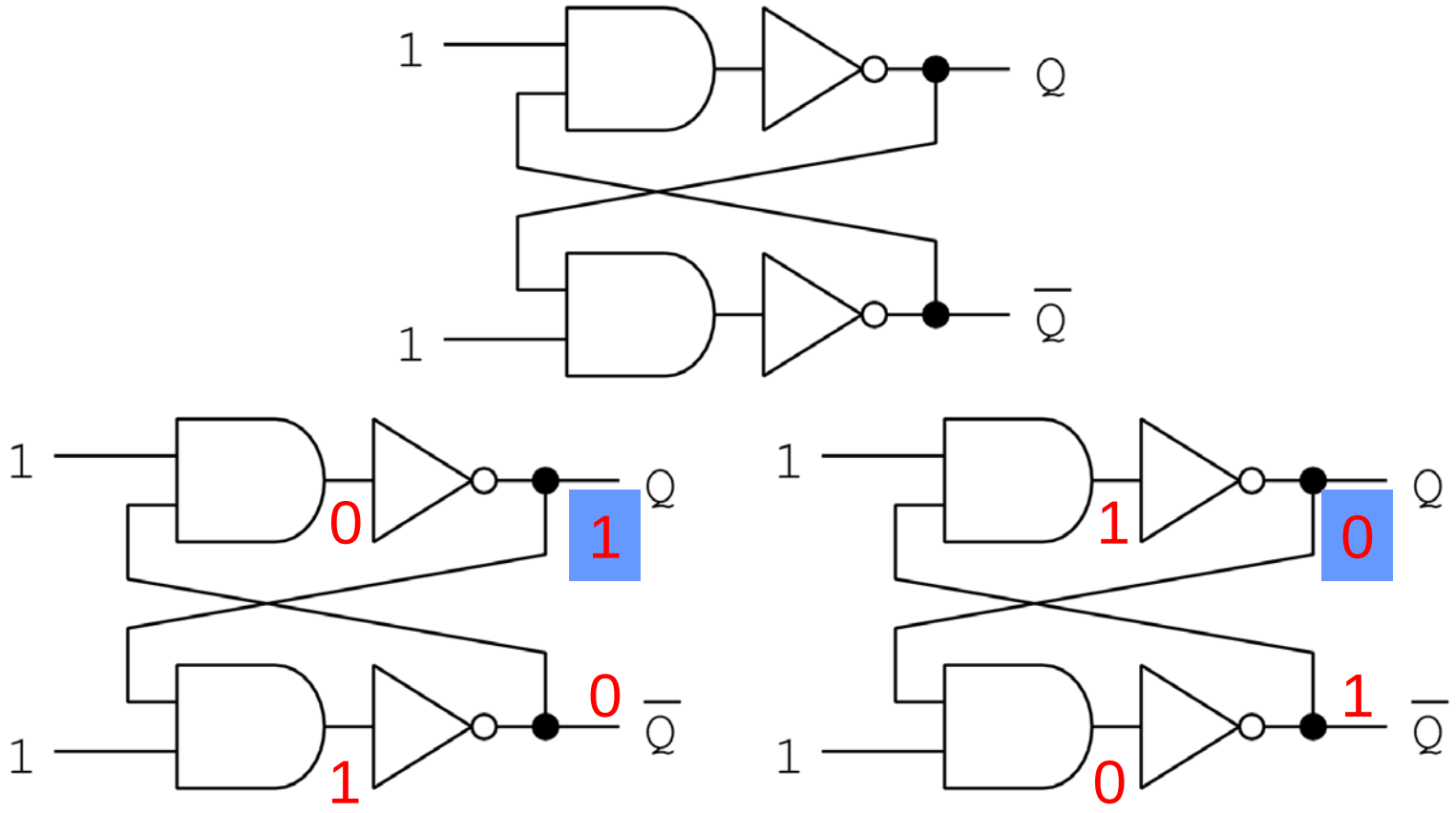
Latch (Inverter Loop)

- Inverter loop with state control logic
- Behavior of Latch (transparent phase)
  - The state (output) of inverter loop is controlled by inputs



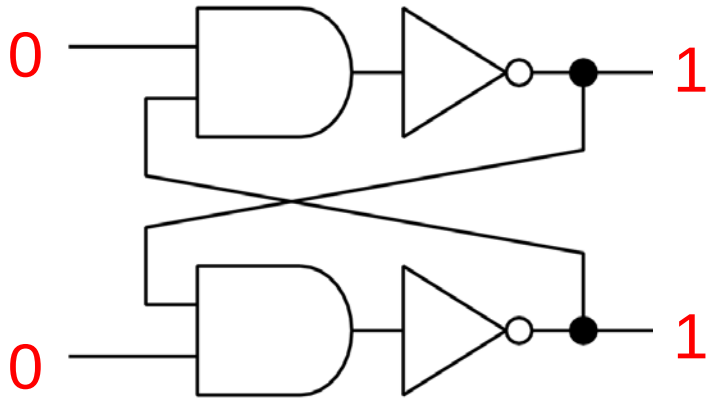
Latch (Inverter Loop) (2)

- Behavior of Latch (opaque phase)
  - The state of inverter loop is kept when both input is 1



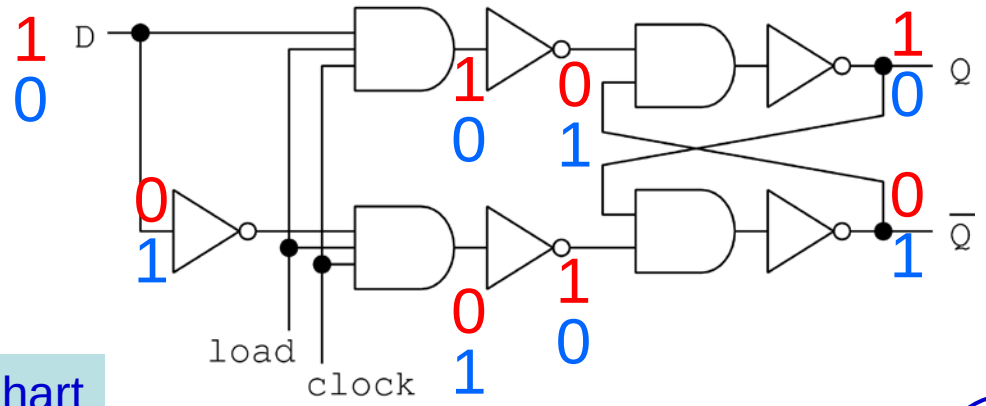
Latch (Inverter Loop) (3)

- Behavior of Latch
  - Unexpected inputs (0,0)

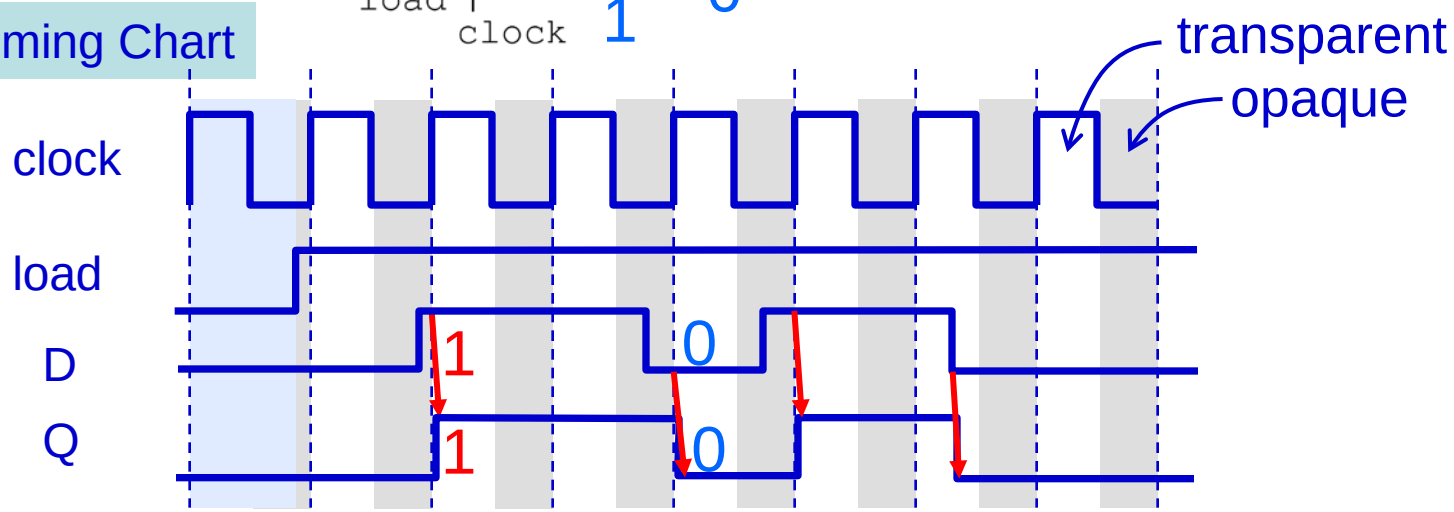


Behavior of D-Latch

- Delay type Latch (D-Latch) with clock and load
  - State changes during transparent phase

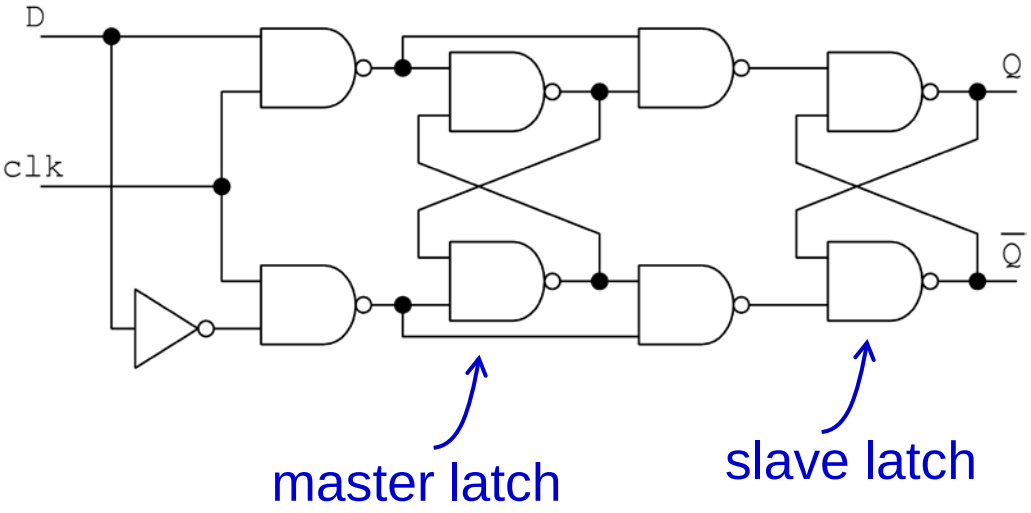


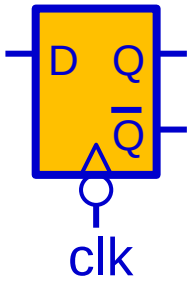
Timing Chart



Flip-Flop (D-FF)

- Delay type Flip-Flop
  - Two inverter loop (master and slave)
  - Next State Q = input D (control signal)
  - State changes at clock-edge only

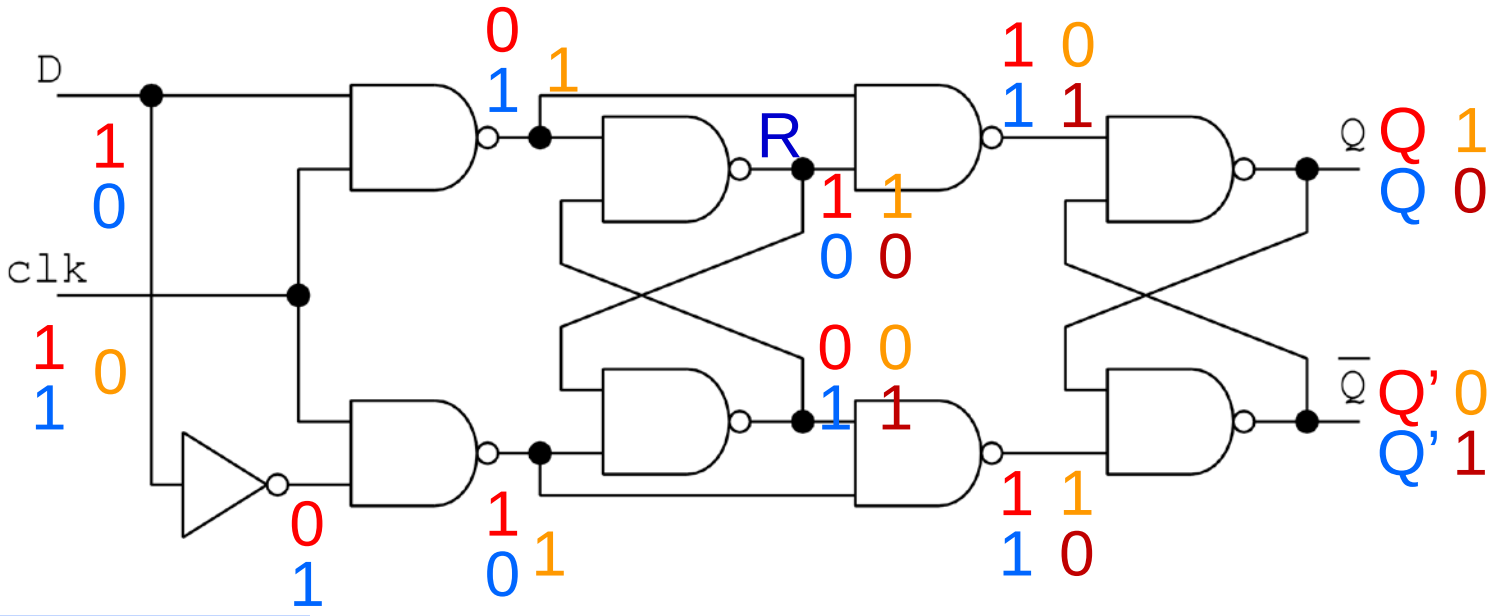




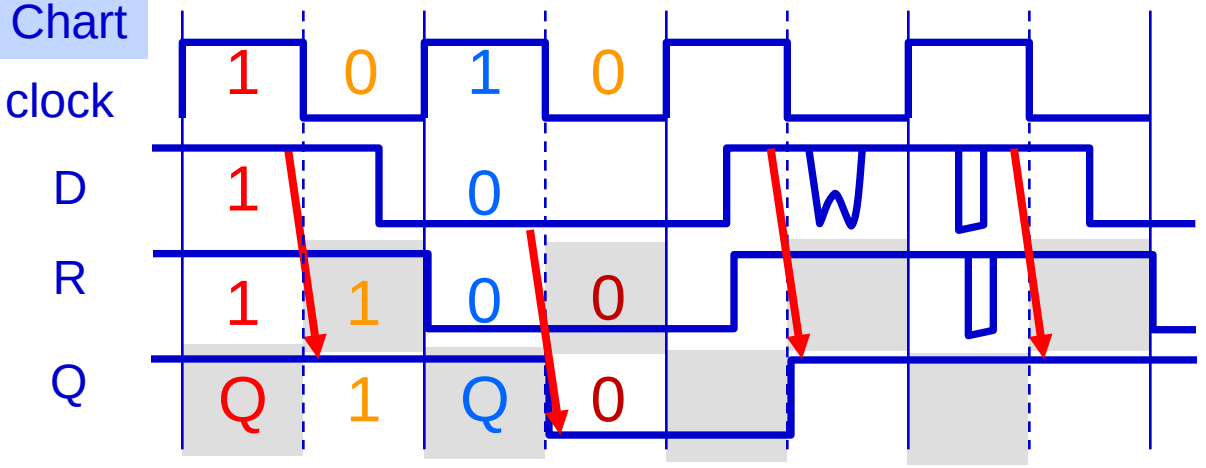
Q(i+1)

Q(i) D(i)	0	1
0	0	1
1	0	1

Behavior of D-FF

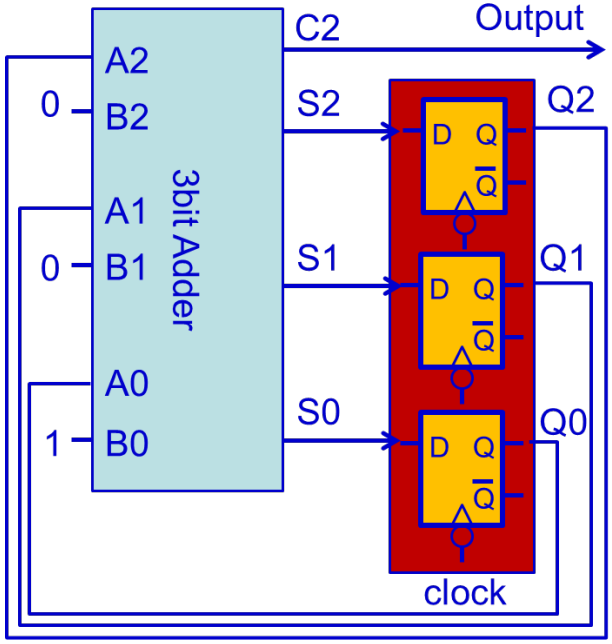


Timing Chart



Example: Counter by Adder and D-FF

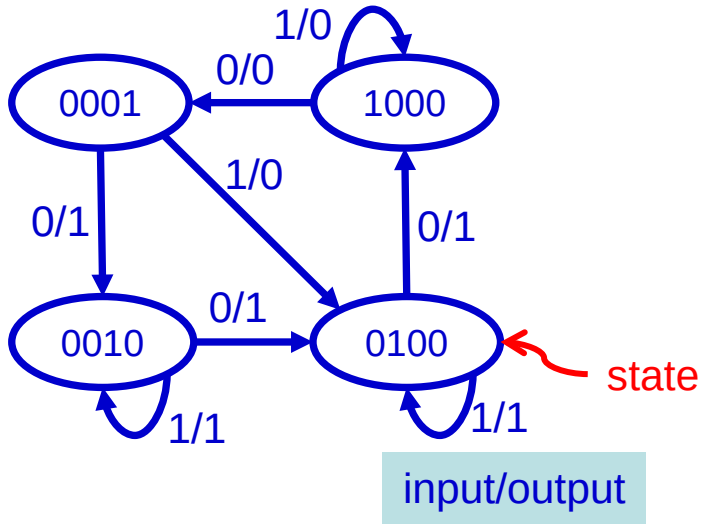
- Count the number of clock inputs
  - State = #clock inputs (8 states = 3 bits)
  - Next state = Current state + 1



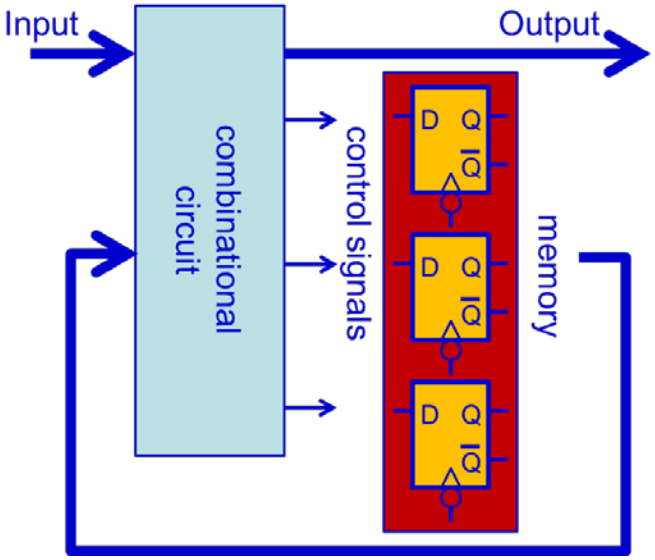
current state Q2Q1Q0	3bit Adder output C2	next state S2S1S0
000	0	001
001	0	010
010	0	011
011	0	100
100	0	101
101	0	110
110	0	111
111	1	000

Sequential Circuit Implementation

- Determine Finite State Machine
- Determine State Assignment
- Synthesis combinational circuit
  - Generate appropriate control signals and outputs



Finite State Machine
with state assignment



Sequential Circuit

Digital Integrated Circuits Synthesis

- Exploration of Huge Design Space
- Increase of computation power enable us to use computation power rich algorithms
  - Iterative improvement
  - Stochastic search
  - Analytical method
- Design space design
  - Abandon useless area
  - Focus on promising area
  - Efficiency