NCDC

NUST CHIP DESIGN CENTRE

Digital Logic Design

Introduction

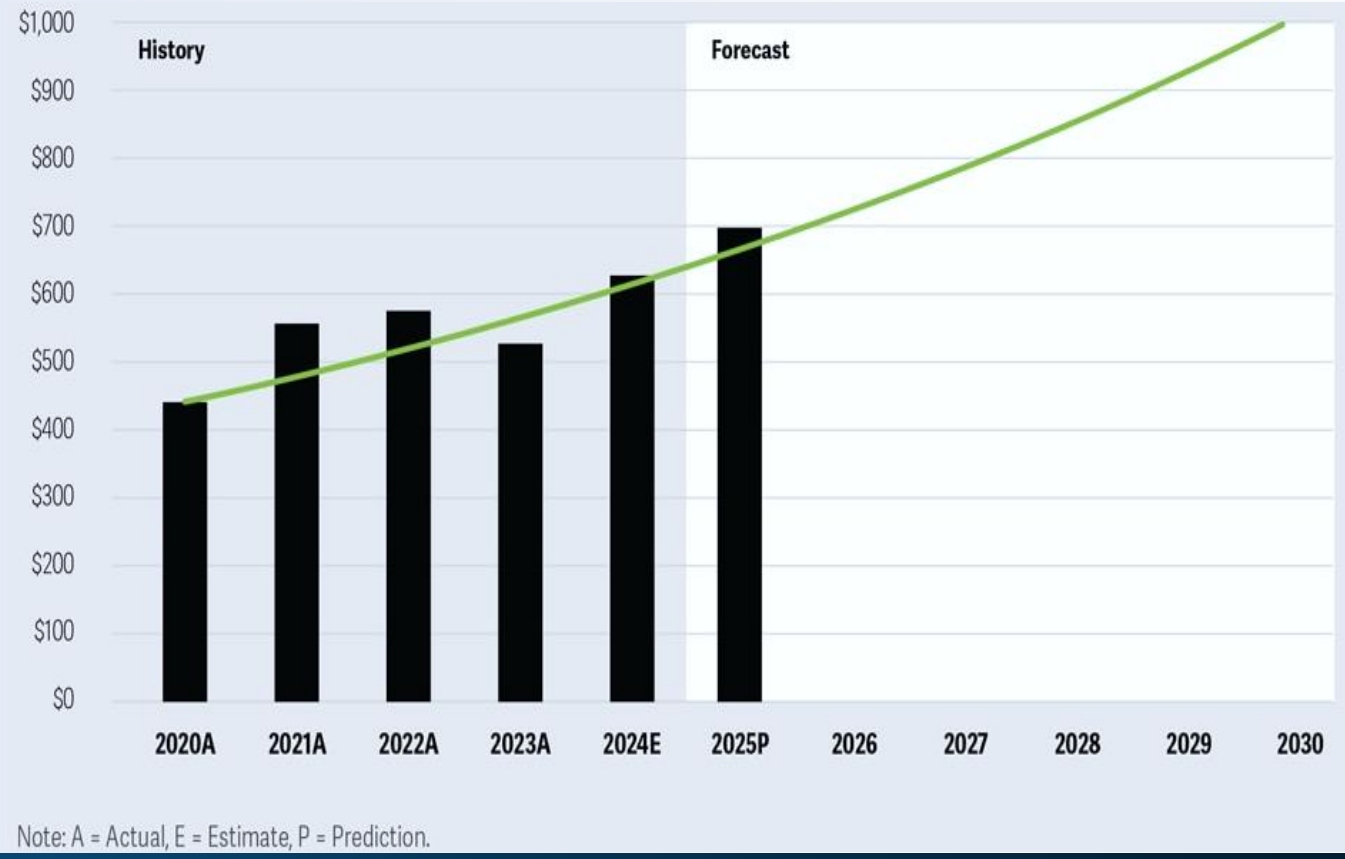
Introduction

Digital Logic Design

Foundation of digital electronics, focusing on the design and analysis of digital circuits that process binary data (0s and 1s).

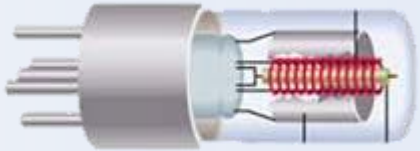
Importance and Motivation

- Microelectronic technologies have revolutionized our world: cell phones, internet, rapid advances in medicine, etc.
- The semiconductor industry has grown tremendously and is expected to reach $1Trillion till 2030.

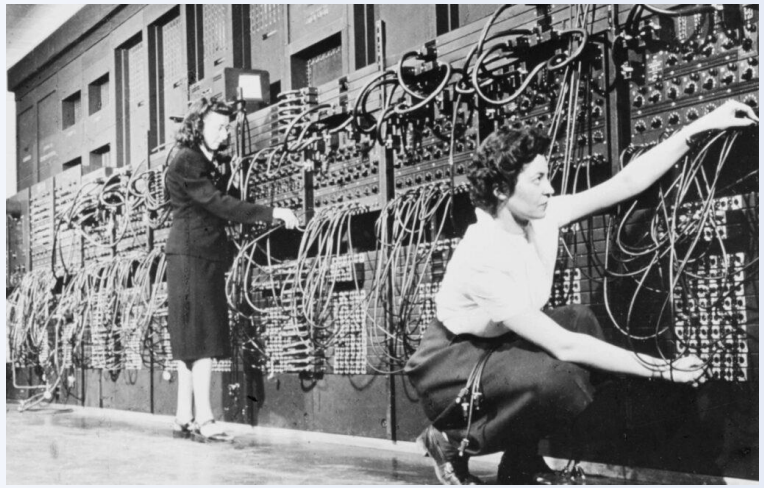


The Digital Revolution

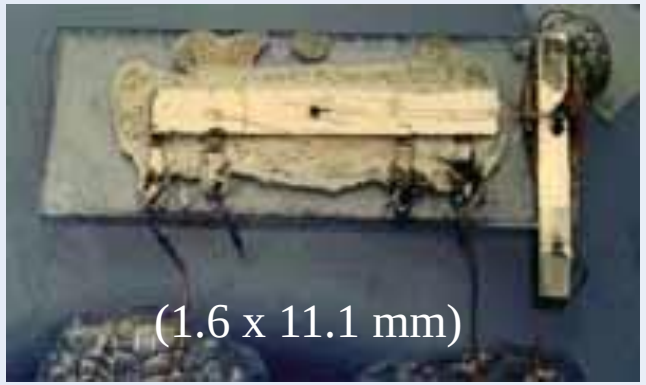
- Integrated Circuit: Many digital operations on the same material



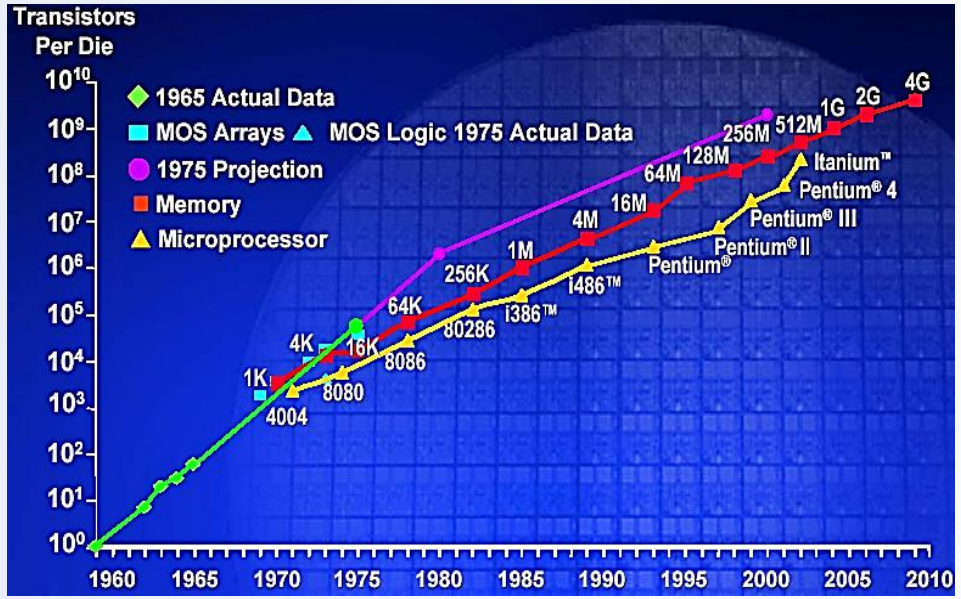
Vacuum tubes



ENIAC
(Electronic Numerical Integrator And Computer)



Integrated Circuit

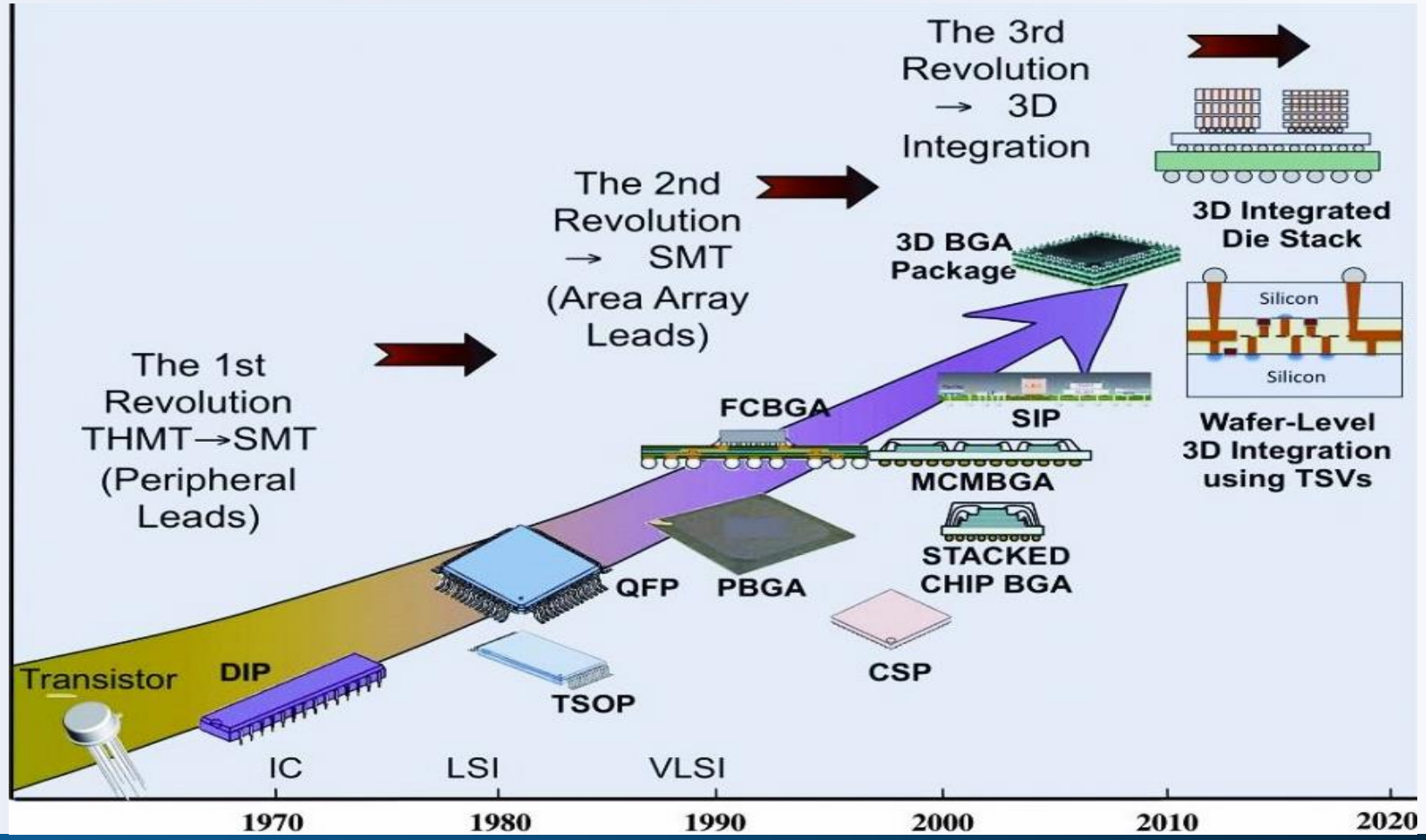


Moore's Law

WWII — 1949 — 1965

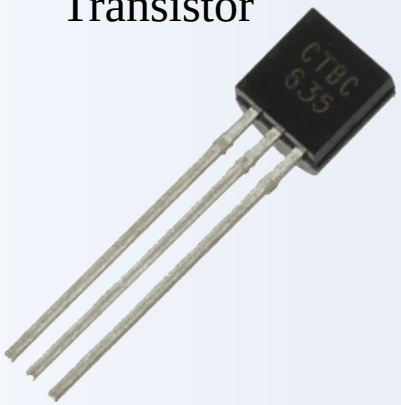
The Digital Revolution

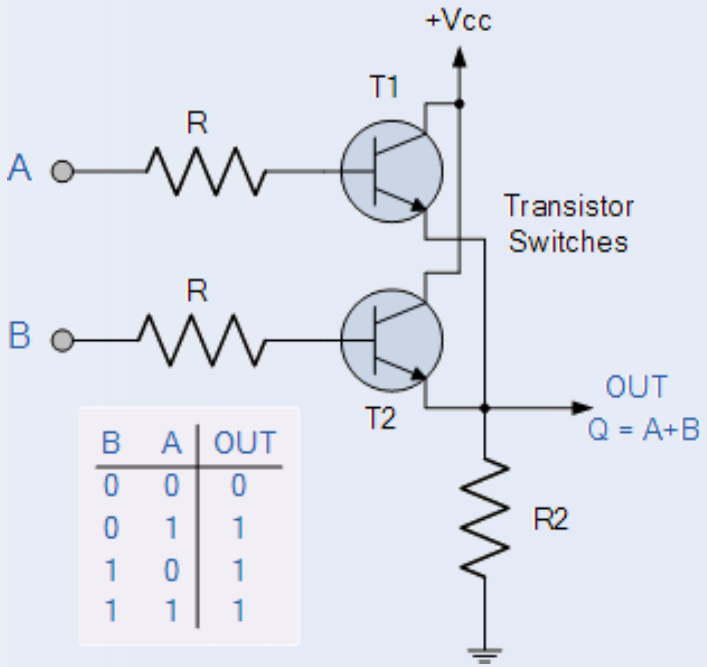
- Integrated Circuit: Growth in computational power and complexity of digital circuits.



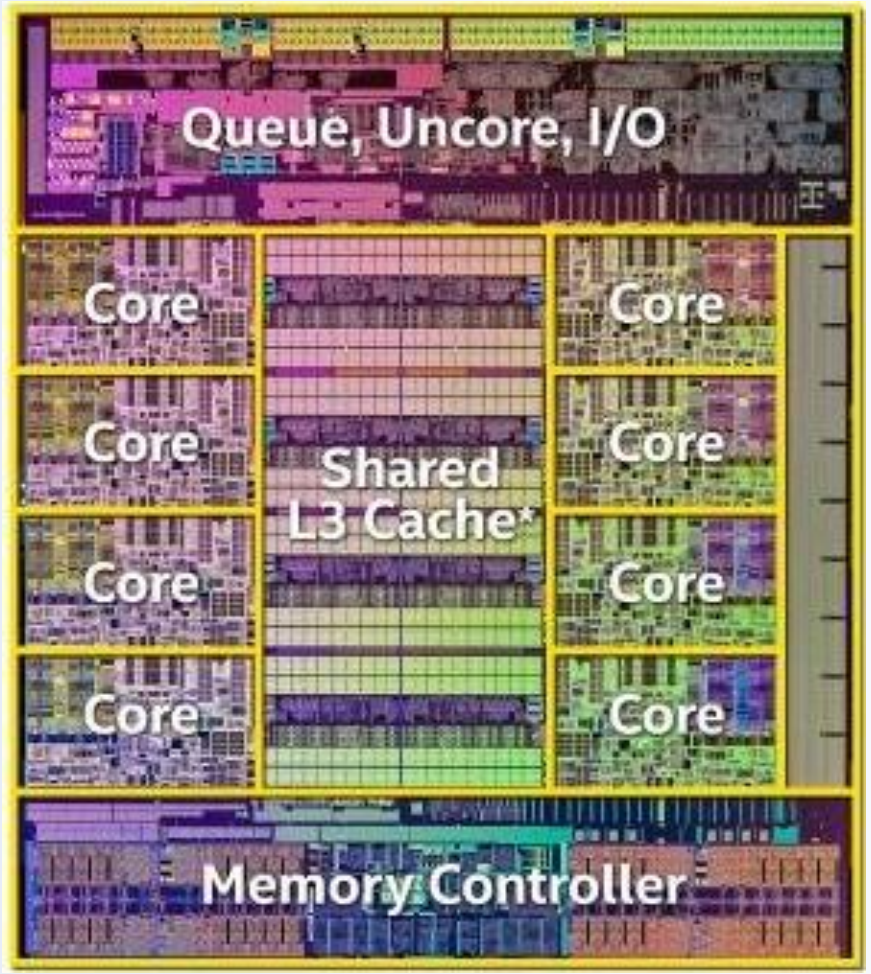
Building complex circuits

Transistor

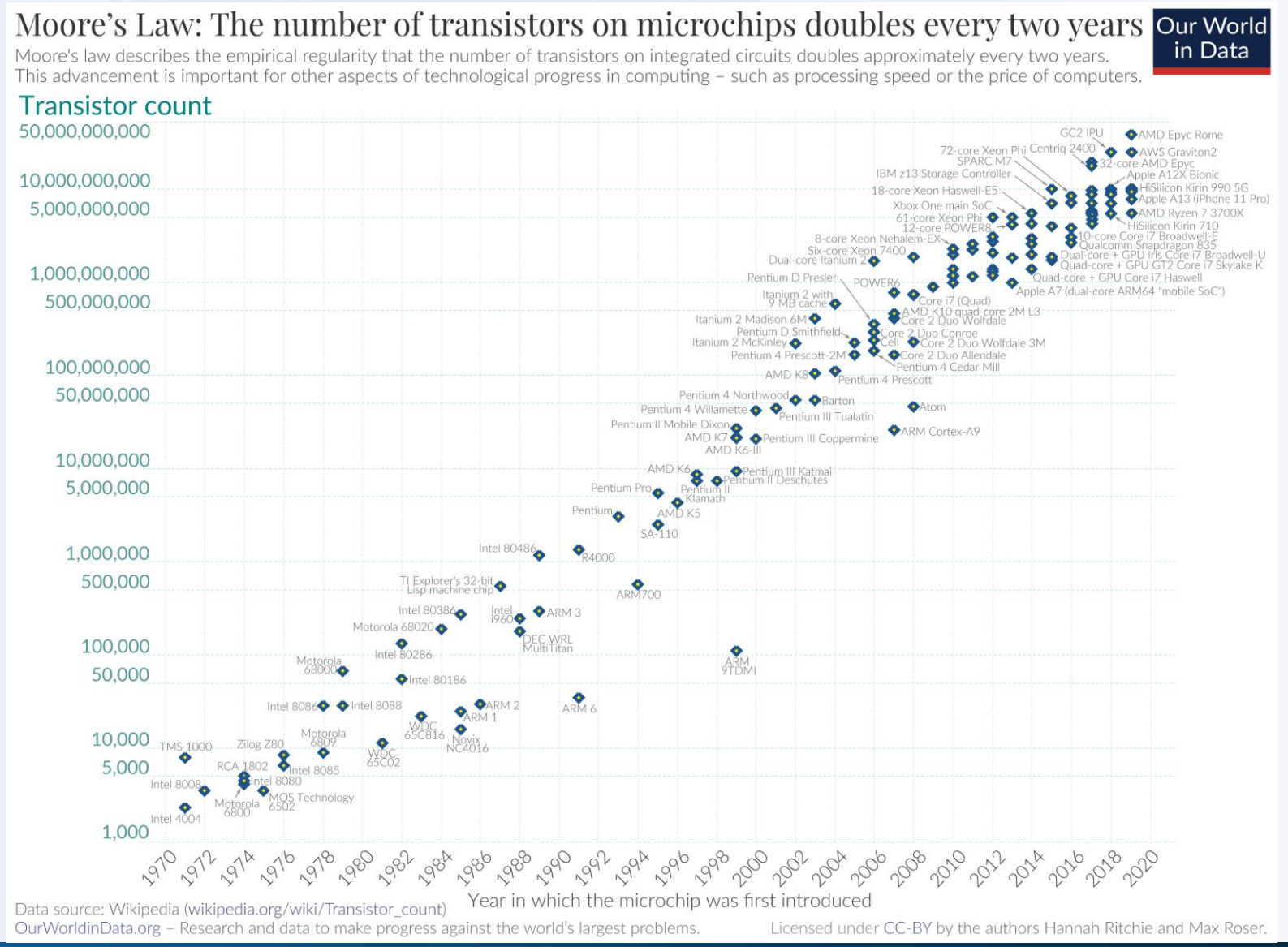




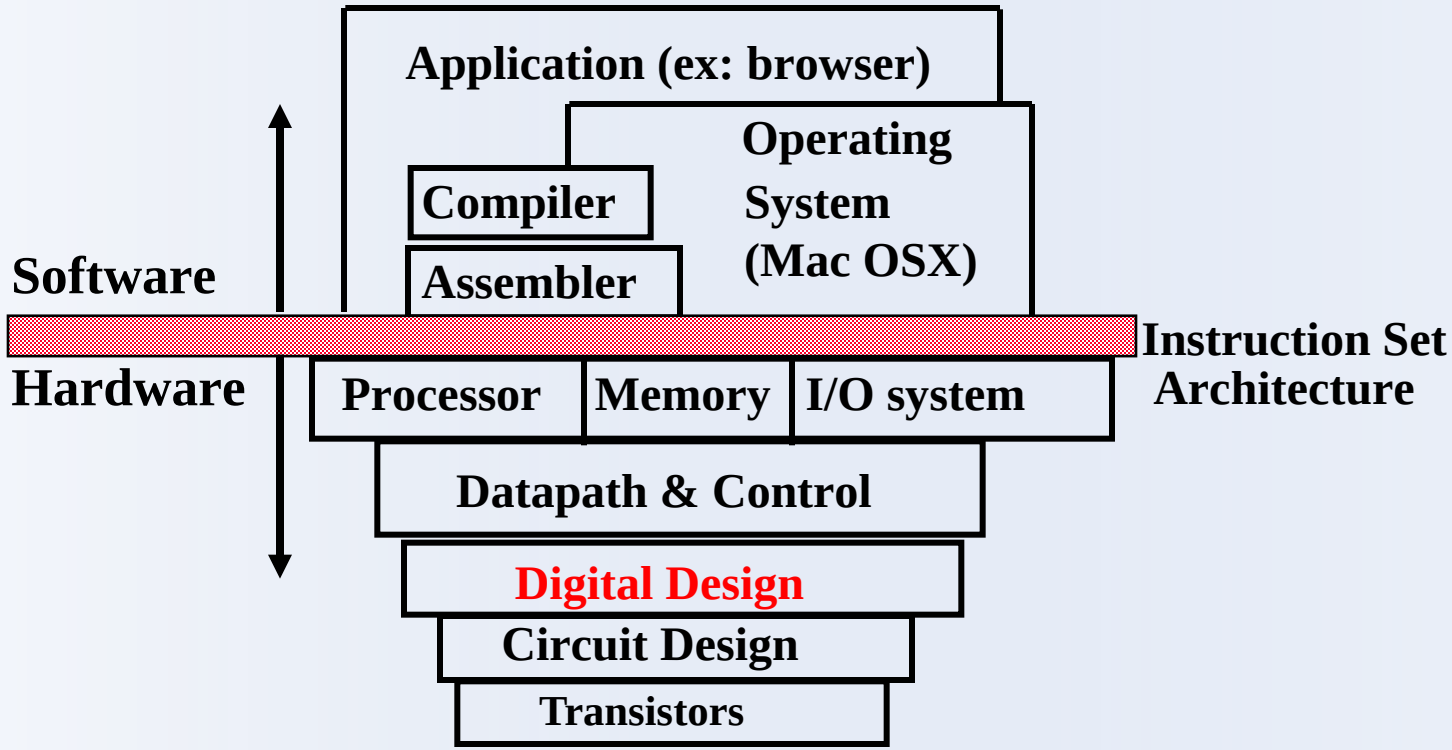
B	A	OUT
0	0	0
0	1	1
1	0	1
1	1	1

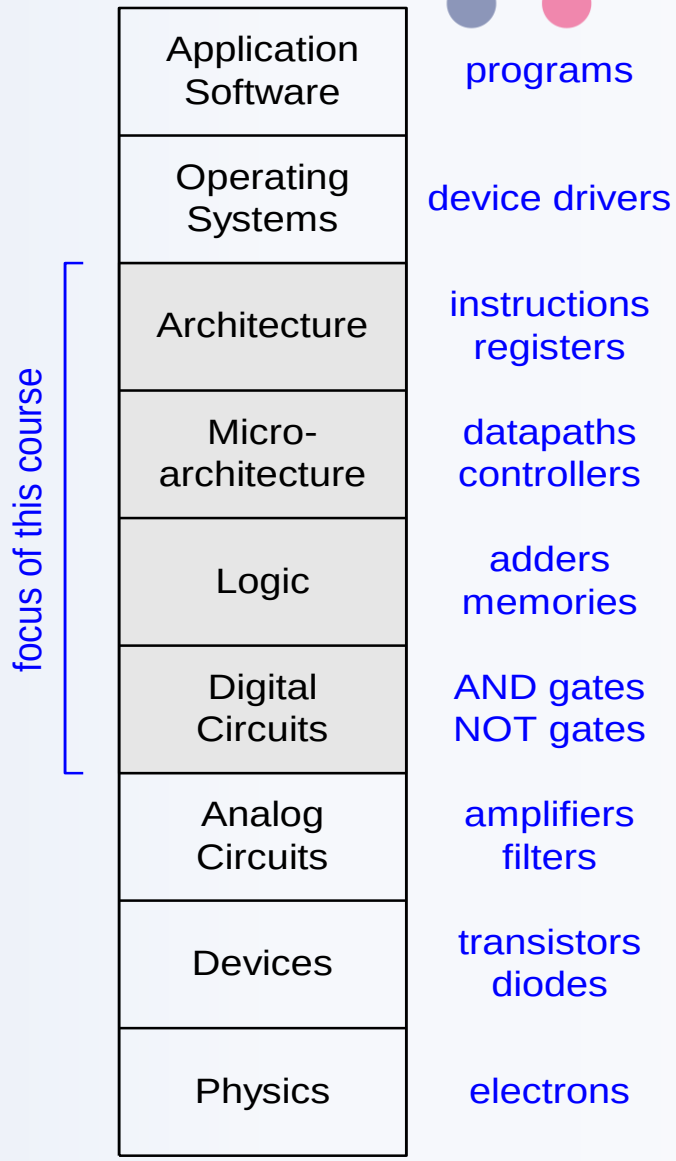


Technology Trends: Moore's Law



Level of Abstractions





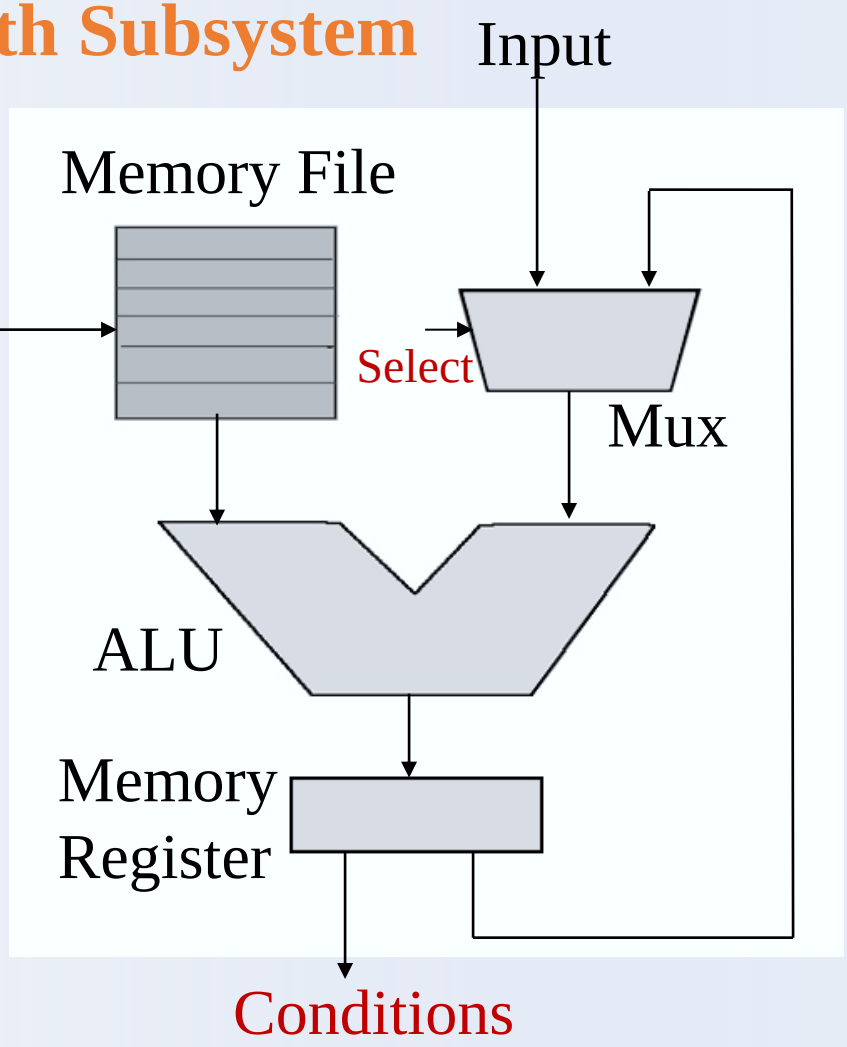
Coordination of many *levels of abstraction*

Objectives of this Course

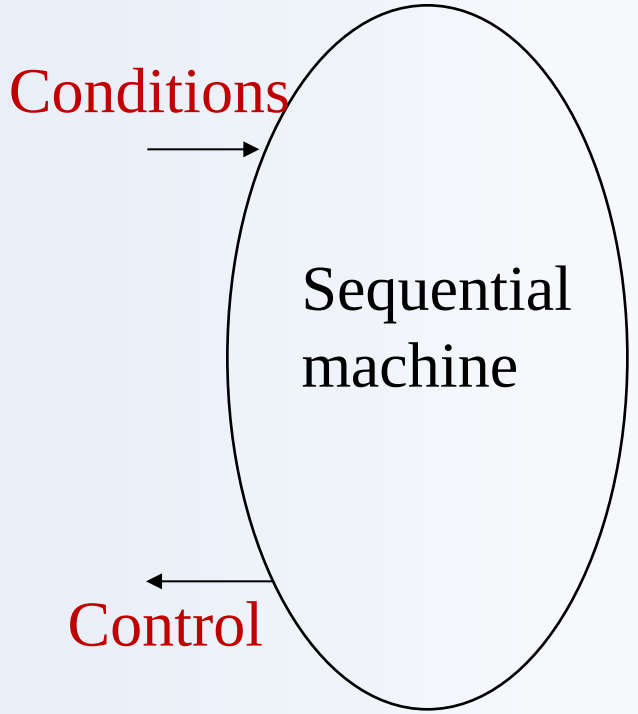
- Learn the fundamental principles of digital design and Boolean logic.
- Learn to systematically debug increasingly complex digital circuits.
- Design and implement combinational and sequential logic circuits.
- Develop skills in using hardware description languages (HDLs) such as Verilog / System Verilog for circuit design and simulation.
- Gain hands-on experience in building digital systems.
- Develop problem-solving skills for real-world applications in embedded systems, automation, and computer architecture.

Overall Picture

Data Path Subsystem

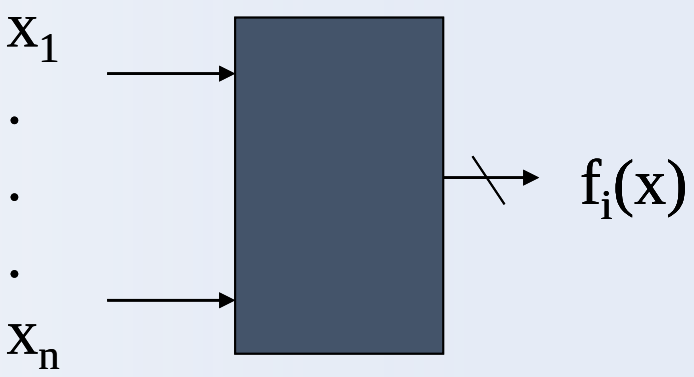


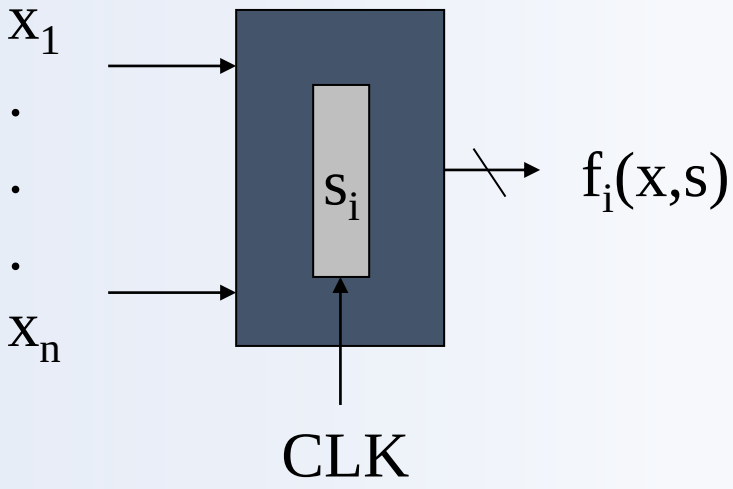
Control Subsystem



CLK: Synchronizing Clock

Combinational Logic vs Sequential Network





Combinational logic:

$y_i = f_i(x_1,..,x_n)$

Sequential Networks

1. Memory
2. Time Steps (Clock)

$y_i^t = f_i(x_1^t,\ldots,x_n^t, s_1^t, \ldots,s_m^t)$

$s_i^{t+1} = g_i(x_1^t,\ldots,x_n^t, s_1^t,\ldots,s_m^t)$

Scope

- Number System
- Combinational Circuits
- Sequential Circuits
- System Verilog Basics
- Finite State Machine
- Communication protocols
- Timing Analysis

Number System

Number System

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7

Decimal	Binary	Octal	Hexadecimal
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Number System Conversion

Conversion of number $(45)_{10}$ & $(26.75)_{10}$

Base 2 (Binary)	Base 2 (Binary) Fractional	Base 8 (Octal)	Base 16 (Hexadecimal)
45÷2=22 rem **1** 22÷2=11 rem **0** 11÷2=5 rem **1** 5÷2=2 rem **1** 2÷2=1 rem **0** 1÷2=0 **1** (stop, quotient is 0)	26÷2=13 rem 0 13÷2=6 rem 1 16÷2=3 rem 0 3÷2=1 rem 1 1÷2=0 rem 1 0.75 × 2 = 1.50 0.50 × 2 = 1.00	45÷8=5 rem **5** 5÷8=0 rem **5** (stop, quotient is 0)	45÷16=2 rem **13(D)** 2÷16=0 rem **2** (stop, quotient is 0)
$(101101)_2$	$(11010.11)_2$	$(55)_8$	$(2D)_{16}$

Number System Conversion

$$N = d_k \cdot B^k + d_{k-1} \cdot B^{k-1} + \cdots + d_1 \cdot B^1 + d_0 \cdot B^0$$

Base	Computation in Base 10	Result
$(101101)_2$	$1\cdot2^5+0\cdot2^4+1\cdot2^3+1\cdot2^2+0\cdot2^1+1\cdot2^0$	45
$(11010.11)_2$	$1.2^4 + 1.2^3 + 0.2^2 + 1.2^1 + 0.2^0 + 1.2^{-1} + 1.2^{-2}$	26.75
$(55)_8$	$5\cdot8^1+5\cdot8^0$	45
$(2D)_{16}$	$2\cdot16^1+13\cdot16^0$	45

Codes (ASCII / BIN / HEX / OCT / DEC)

DEC	OCT	HEX	BIN	Symbol
0	000	00	00000000	NUL
1	001	01	00000001	SOH
2	002	02	00000010	STX
3	003	03	00000011	ETX
4	004	04	00000100	EOT
5	005	05	00000101	ENQ
6	006	06	00000110	ACK
7	007	07	00000111	BEL
8	010	08	00001000	BS
9	011	09	00001001	HT
10	012	0A	00001010	LF
…	…	…	…	…
…	…	…	…	…
…	…	…	…	…

DEC	OCT	HEX	BIN	Symbol
…	…	…	…	…
101	145	65	01100101	e
102	146	66	01100110	f
103	147	67	01100111	g
104	150	68	01101000	h
105	151	69	01101001	i
106	152	6A	01101010	j
107	153	6B	01101011	k
108	154	6C	01101100	l
109	155	6D	01101101	m
110	156	6E	01101110	n
111	157	6F	01101111	o
112	160	70	01110000	p
113	161	71	01110001	q

DEC	OCT	HEX	BIN	Symbol
114	162	72	01110010	r
115	163	73	01110011	s
116	164	74	01110100	t
117	165	75	01110101	u
118	166	76	01110110	v
119	167	77	01110111	w
120	170	78	01111000	x
121	171	79	01111001	y
122	172	7A	01111010	z
123	173	7B	01111011	{
124	174	7C	01111100	|
125	175	7D	01111101	}
126	176	7E	01111110	~
127	177	7F	01111111	DEL

Combinational Circuit

What is a combinational circuit?

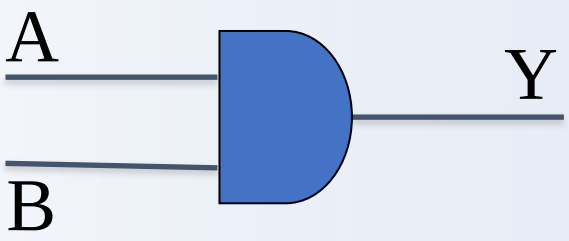
- No memory
- Realizes one or more functions
- Inputs and outputs can only have two discrete values
  - Physical domain (usually, voltages) (0V, 5V)
  - Mathematical domain : Boolean variables (true or false)

Combinational Logic: Scope

- Description
  - Language: C Programming, Verilog, VHDL, SystemVerilog
  - Boolean algebra
  - Truth table
- Design
  - Schematic Diagram
  - Inputs, Gates, Nets, Outputs
- Goal
  - Validity: correctness, turnaround time
  - Performance: power, timing, cost
  - Testability: yield, diagnosis, robustness

Representations of Digital circuits

Schematic



Truth Table

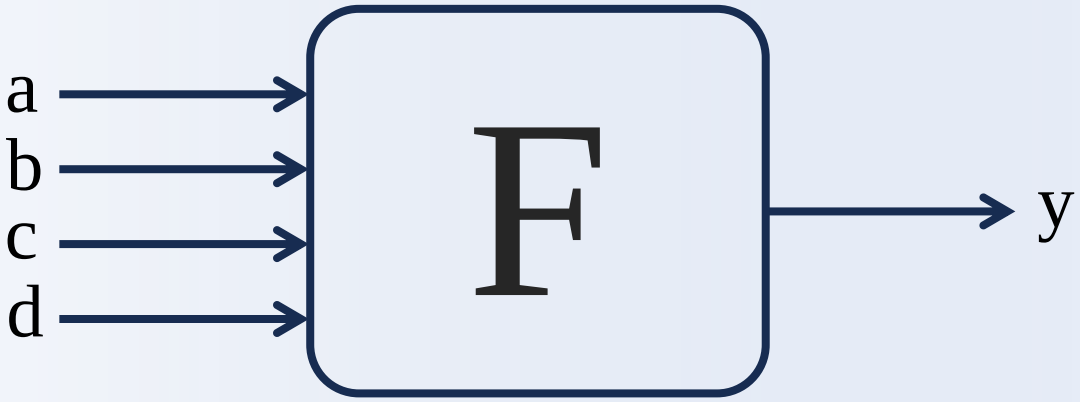
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Boolean Expression

$Y = A.B$

Truth Tables

Truth Tables/Boolean Function

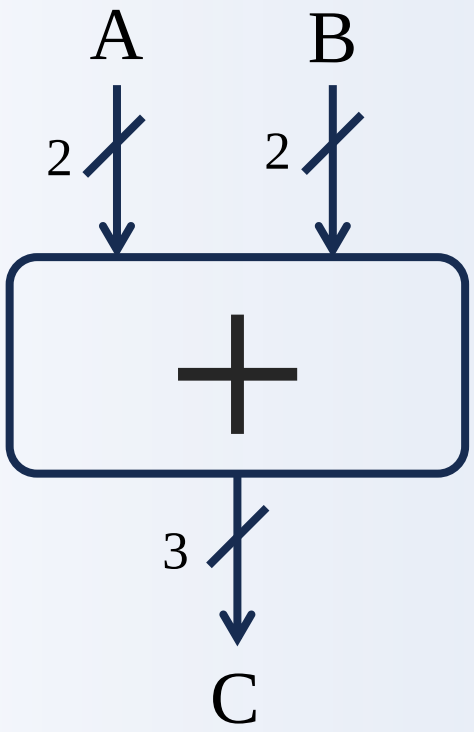


How many combinations for a 4-input devices?

a	b	c	d	y
0	0	0	0	F(0,0,0,0)
0	0	0	1	F(0,0,0,1)
0	0	1	0	F(0,0,1,0)
0	0	1	1	F(0,0,1,1)
0	1	0	0	F(0,1,0,0)
0	1	0	1	F(0,1,0,1)
0	1	1	0	F(0,1,1,0)
0	1	1	1	F(0,1,1,1)
1	0	0	0	F(1,0,0,0)
1	0	0	1	F(1,0,0,1)
1	0	1	0	F(1,0,1,0)
1	0	1	1	F(1,0,1,1)
1	1	0	0	F(1,1,0,0)
1	1	0	1	F(1,1,0,1)
1	1	1	0	F(1,1,1,0)
1	1	1	1	F(1,1,1,1)

TT Examples:

2-bit adder



A a_1a_0	B b_1b_0	C $c_2c_1c_0$
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

XOR

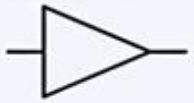
a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

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

Logic Gates

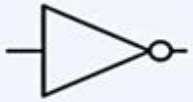
Logic Gates (Schematics + Truth Table)

Buffer



Input	Output
0	0
1	1

Inverter



Input	Output
0	1
1	0

AND

A	B	Output
0	0	0
1	0	0
0	1	0
1	1	1

NAND

A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

OR

A	B	Output
0	0	0
1	0	1
0	1	1
1	1	1

NOR

A	B	Output
0	0	1
1	0	0
0	1	0
1	1	0

XOR

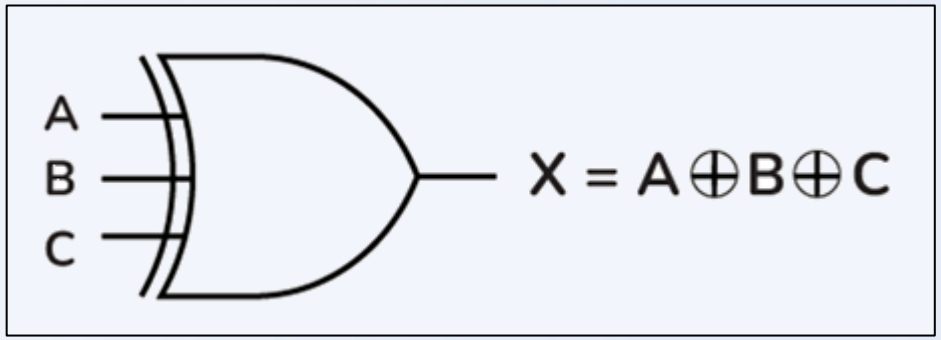
A	B	Output
0	0	0
1	0	1
0	1	1
1	1	0

XNOR

A	B	Output
0	0	1
1	0	0
0	1	0
1	1	1

2-input gates extend to n-inputs

- N-input XOR is the only one which isn't so obvious
- It's actually simple…
  - XOR is a **1** iff the # of **1**s at its input is odd

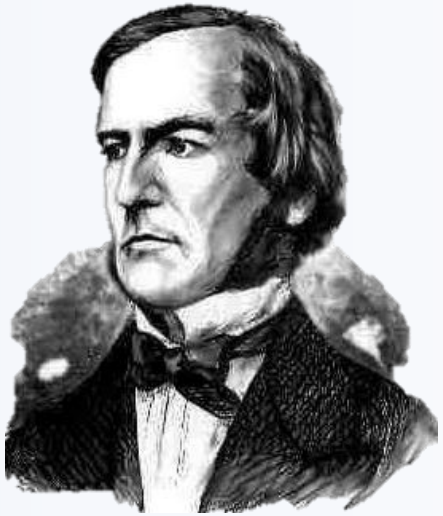


A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Boolean Algebra

Boolean Algebra

- George Boole, 19th Century mathematician
- Developed a mathematical system (algebra) involving logic
  - later known as “Boolean Algebra”



Similar to regular algebra but defined on sets with three basic ‘logic’ operations:

1. Intersection: AND (2-input); Operator: .
2. Union: OR (2-input); Operator: +
3. Complement: NOT (1-input); Operator: ’ ¯

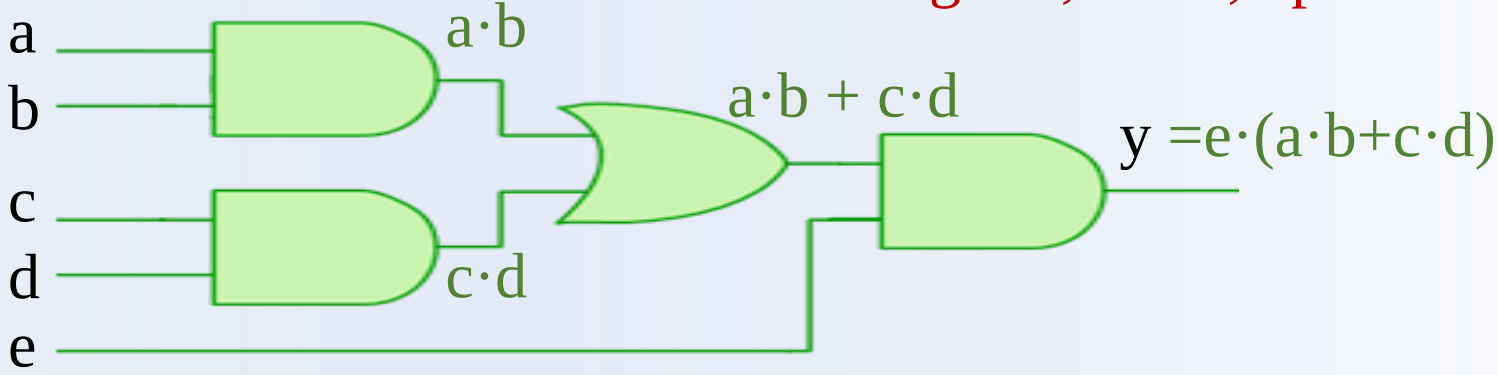
Boolean Functions

For two Boolean variables X and Y with X=1, Y=0,

- what is function F(X,Y)=X+Y?
- what is function F(X,Y)=X+X+Y?
- what is function F(X,Y)=X+X.Y?
- what is function F(X,Y)=(X+Y').Y?

Boolean Algebra for Logic

Cost: #gates, #nets, #pins



Schematic Diagram:

5 primary inputs

1 primary output

4 gates (3 ANDs, 1 OR)

9 signal nets

12 pins

Boolean Algebra:

5 variables

1 expression

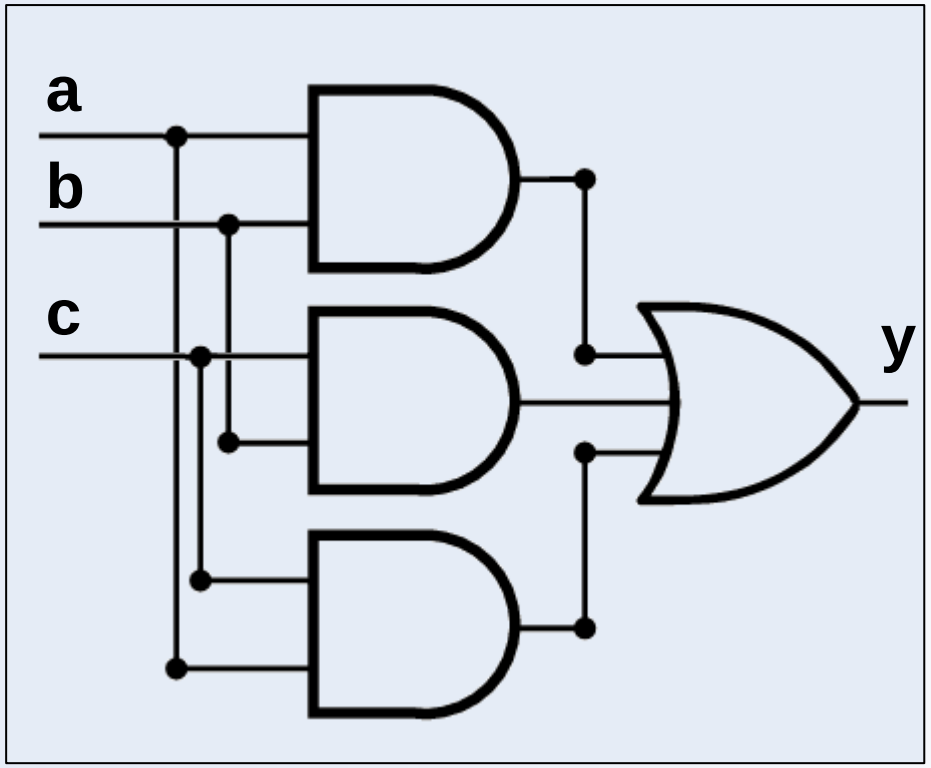
4 operators (3 ANDs, 1 OR)

5 literals

Truth Table → Schematic → Boolean Expression

3-input majority circuit:

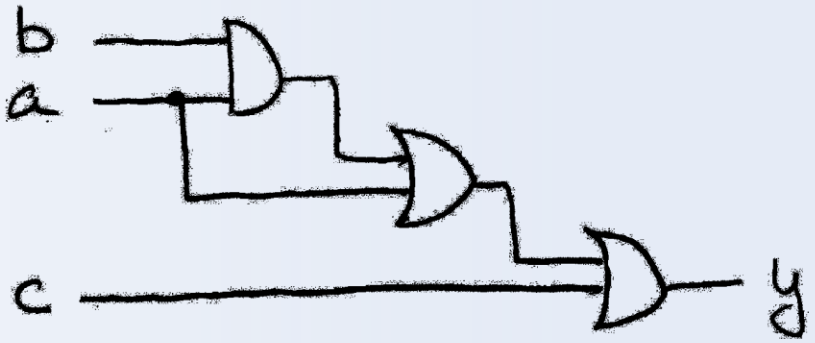
a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



$$y = a \cdot b + a \cdot c + b \cdot c$$

$$y = ab + ac + bc$$

BA: Circuit & Algebraic Simplification



original circuit

$$\downarrow$$

$$y = ab + a + c$$

equation derived from original circuit

$$\downarrow$$

$$\begin{aligned} ab + a + c \\ = a(b + 1) + c \\ = a(1) + c \\ = a + c \end{aligned}$$

algebraic simplification

$$\downarrow$$

simplified circuit

Laws of Boolean Algebra

Laws of Boolean Algebra

$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$	complementarity
$x \cdot 0 = 0$	$x + 1 = 1$	laws of 0's and 1's
$x \cdot 1 = x$	$x + 0 = x$	identities
$x \cdot x = x$	$x + x = x$	idempotent law
$x \cdot y = y \cdot x$	$x + y = y + x$	communitive law
$(xy)z = x(yz)$	$(x + y) + z = x + (y + z)$	associativity
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	distribution
$xy + x = x$	$(x + y)x = x$	uniting theorem
$\overline{x \cdot y} = \overline{x} + \overline{y}$	$\overline{(x + y)} = \overline{x} \cdot \overline{y}$	DeMorgan's Law

Consensus Theorem

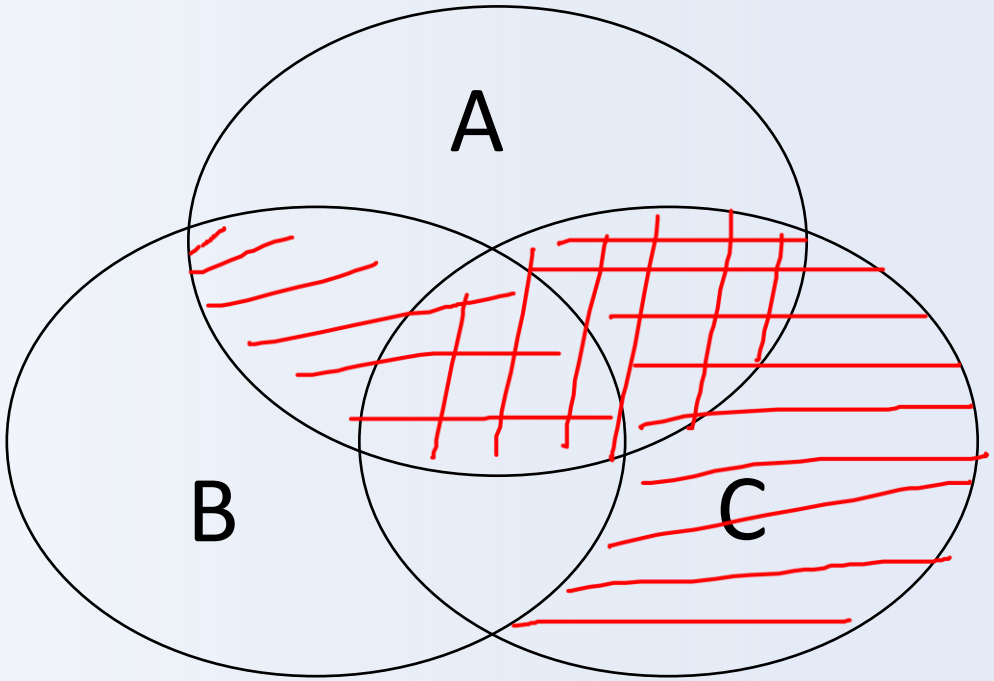
- AB+AC+B'C
  =AB+B'C

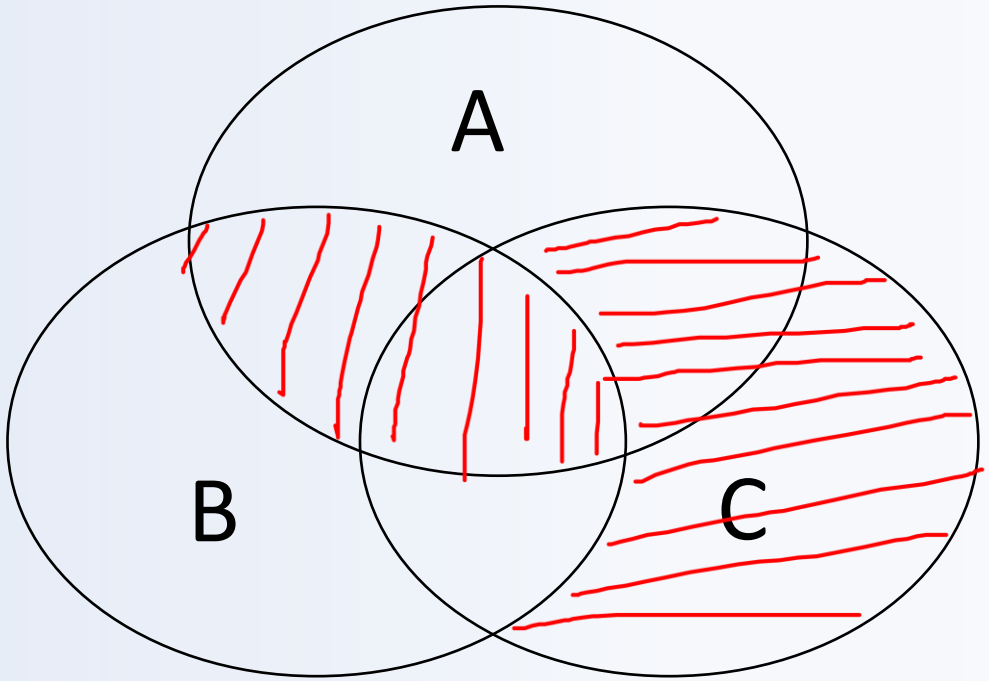
- (A+B)(A+C)(B'+C)
  =(A+B)(B'+C)

Exercise: to prove the reduction using
(1) Venn Diagrams,
(2) Boolean algebra,
(3) Logic simulation and
(4) Shannon's expansion

Consensus Theorem: Venn Diagrams

AB+AC+B’C => AB+B’C





Consensus Theorem: Boolean Algebra

$$AB+AC+B'C \Rightarrow AB+B'C$$

AB+AC+B'C	
=AB+AC1+B'C	identity
=AB+AC(B+B')+B'C	complementary
=AB+ABC+AB'C+B'C	distribution
=AB(1+C)+(A+1)B'C	Law of 1's
=AB+B'C	

Consensus Theorem: Logic Simulation

f(A,B,C) = AB+AC+B’C

g(A,B,C) =AB+B’C

A	B	C	AB	AC	B’C	f	g
0	0	0	0	0	0		
0	0	1	0	0	1		
0	1	0	0	0	0		
0	1	1	0	0	0		
1	0	0	0	0	0		
1	0	1	0	1	1		
1	1	0	1	0	0		
1	1	1	1	1	0		

Consensus Theorem: Shannon's Expansion

- Divide a function into smaller functions
- Pick a variable x, partition the switching function into two cases: x=1 and x=0
  - F(x,y,z,…)= xF(x=1,y,z,…) + x'F(x=0,y,z,…)

F(A,B,C): AB+AC+B'C => AB+B'C

F(A,B,C) = BF(A,1,C) + B'F(A,0,C)
= B(A.1+AC+0.C) + B'(A.0+AC+1.C)
= B(A+AC) + B'(AC+C)
= BA(1+C) + B'C(1+A)
= AB + B'C

Canonical Forms

Canonical Form

A **canonical form** of a Boolean expression is a standardized way of writing it using either **minterms** (Sum of Products - SOP) or **maxterms** (Product of Sums - POS). It ensures a **unique** representation of the Boolean function.

$F = \Sigma m$ (sum of minterms)

$F = \Pi M$ (product of maxterms)

Minterm and Maxterm

Id	a	b	c	out	
0	0	0	0	0	a+b+c
1	0	0	1	0	a+b+c’
2	0	1	0	0	a+b’+c
3	0	1	1	1	a’ b c
4	1	0	0	0	a’+b+c
5	1	0	1	1	a b’c
6	1	1	0	1	a b c’
7	1	1	1	1	a b c

$f_2(a,b,c) = (a+b+c)(a+b+c')(a+b'+c)(a'+b+c)$

$f_2(a,b,c) = M_0M_1M_2M_4 = \Pi M(0,1,2,4)$

maxterm

$f_1(a,b,c) = a'bc + ab'c + abc' + abc$

$f_1(a,b,c) = m_3 + m_5 + m_6 + m_7 = \Sigma m(3,5,6,7)$

minterm

Canonical Form: Example

	abc	y
$\overline{a} \cdot \overline{b} \cdot \overline{c}$	000	1
$\overline{a} \cdot \overline{b} \cdot c$	001	1
	010	0
	011	0
$a \cdot \overline{b} \cdot \overline{c}$	100	1
	101	0
$a \cdot b \cdot \overline{c}$	110	1
	111	0

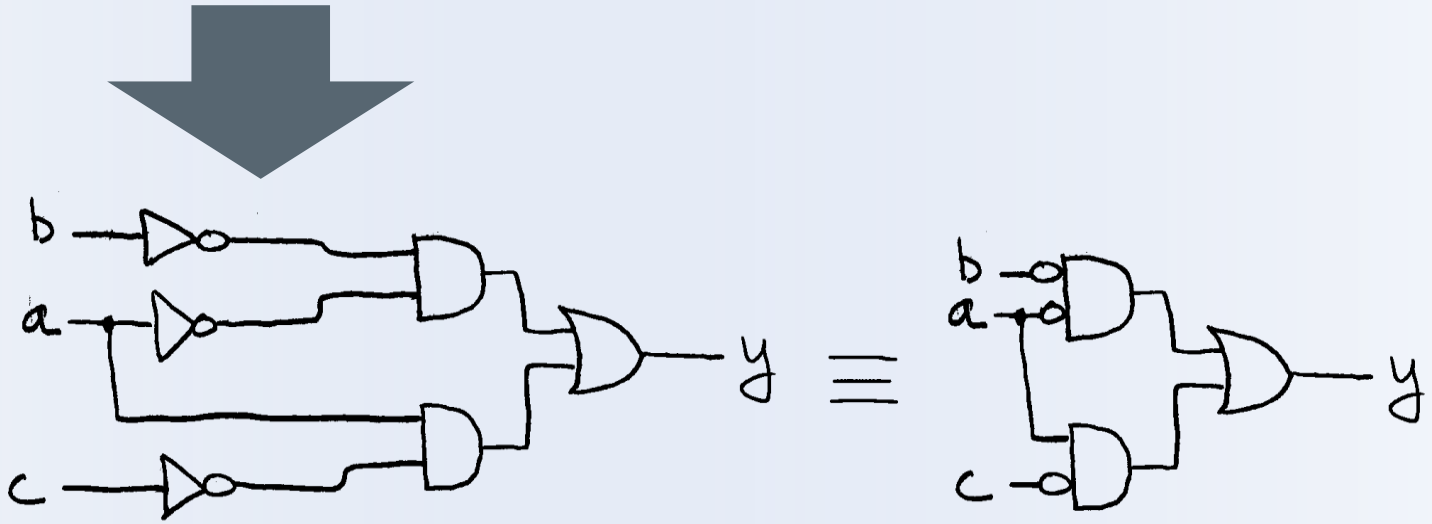
Sum-of-products (SoP)
(ORs of ANDs)

$$y = \overline{a}\overline{b}\overline{c} + \overline{a}\overline{b}c + a\overline{b}\overline{c} + ab\overline{c}$$

What is the Product-of-sum (PoS)?

Canonical Form: Example

$$
\begin{aligned}
y &= \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + a\bar{b}\bar{c} + ab\bar{c} \\
&= \bar{a}\bar{b}(\bar{c} + c) + a\bar{c}(\bar{b} + b) && \textit{distribution} \\
&= \bar{a}\bar{b}(1) + a\bar{c}(1) && \textit{complementarity} \\
&= \bar{a}\bar{b} + a\bar{c} && \textit{identity}
\end{aligned}
$$



What We have discussed So Far:

- Number System
- Combinational Logic
  - Circuit representations
  - Interconversion between various forms of representations
  - Boolean laws and reductions of Boolean expressions
  - Consensus theorem for digital circuits and expression
- General overflow of the system
  - Description
  - Schematics
  - Cost

Implementation

Specification ➔ Schematic Diagram, Net list, Boolean expression

Obj min cost ➔ Search in solution space (max performance)

Cost: wires, gates ➔ Literals, product terms, sum terms

For two level logic (sum of products or product of sums), we want to minimize # of terms, and # of literals.

Implementation: Specification => Logic Diagram

Flow 1:

1. Specification
2. Truth table
3. Sum of products (SOP) or product of sums (POS) canonical form
4. Reduced expression using Boolean algebra
5. Schematic diagram

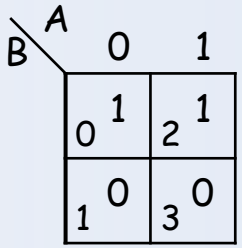
Flow 2:

1. Specification
2. Truth Table
3. Karnaugh Map (truth table in two dimensional space)
4. Reduce using K-Maps
5. Reduced expression (SOP or POS)
6. Schematic diagram

Karnaugh Maps

Karnaugh Maps

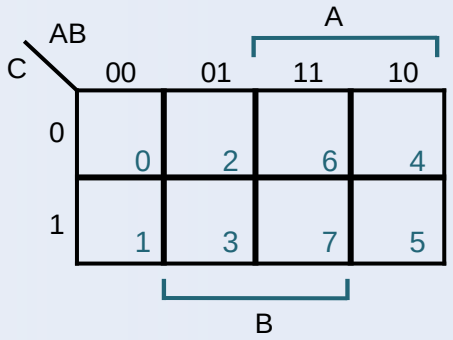
- **Alternative to truth-tables to help visualize adjacencies**
  - **Guide to applying the uniting theorem - On-set elements with only one variable changing value are adjacent unlike in a linear truth-table**

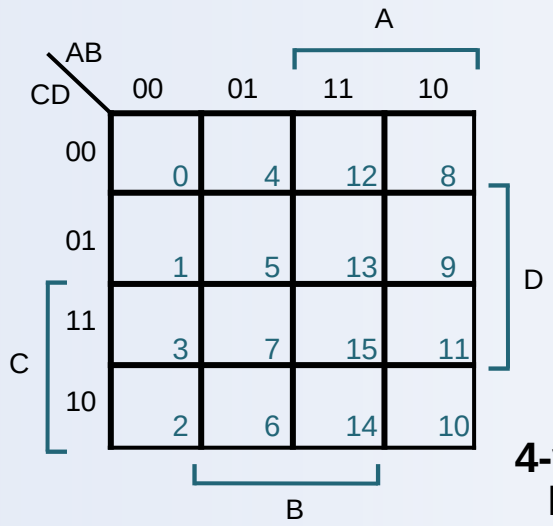


A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

  - **Numbering scheme based on Gray–code**
    - **e.g., 00, 01, 11, 10 (only a single bit changes in code for adjacent map cells)**

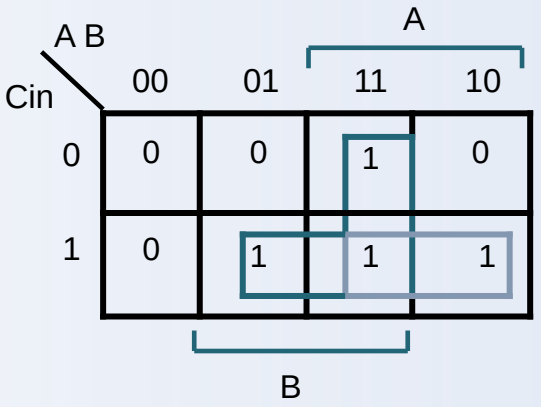
3-variable K-map



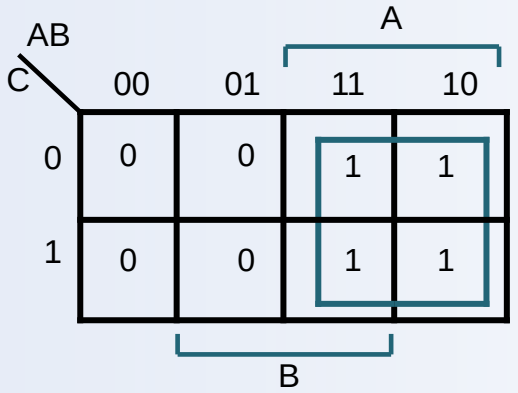


4-variable K-map

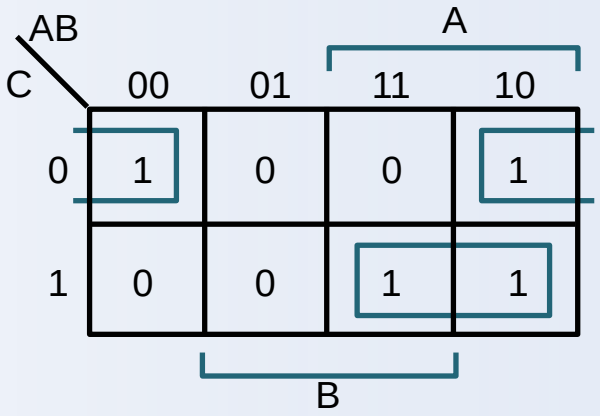
K-Map Examples



Cout =

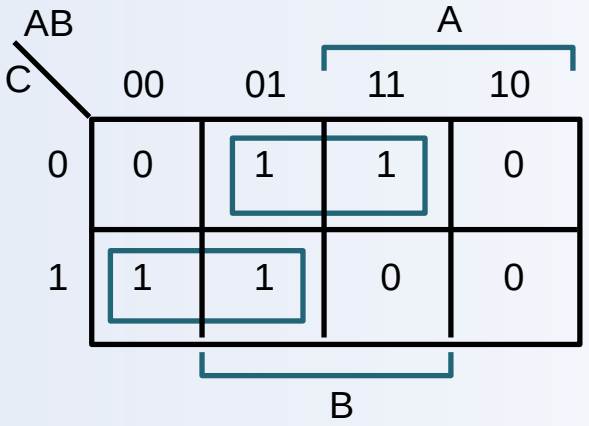


F(A,B,C) =



F(A,B,C) = Σm(0,4,5,7)

F =

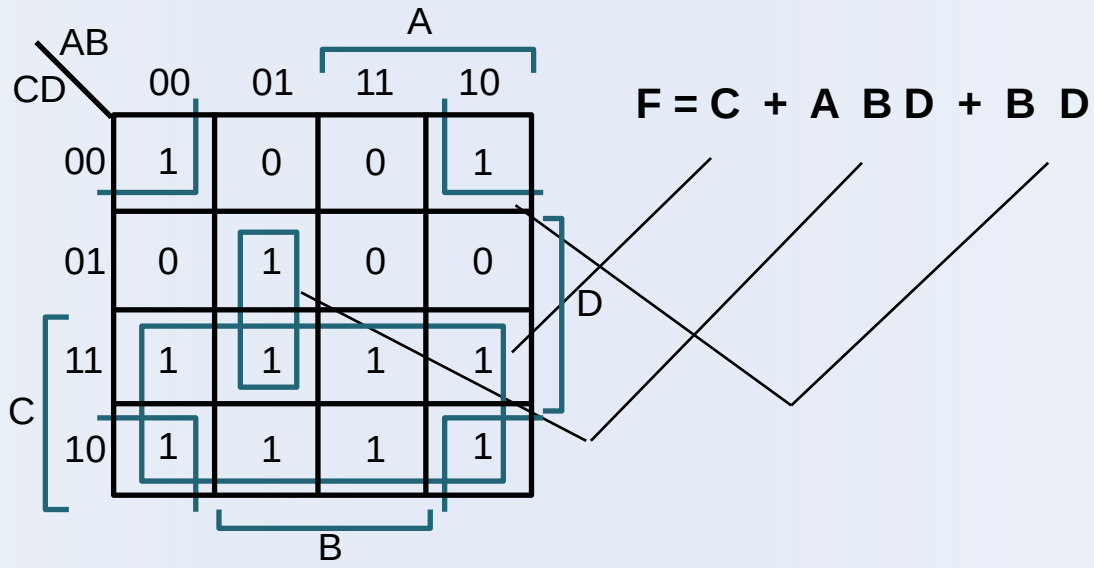


F' simply replace 1's with 0's and vice versa

F'(A,B,C) = Σm(1,2,3,6) =

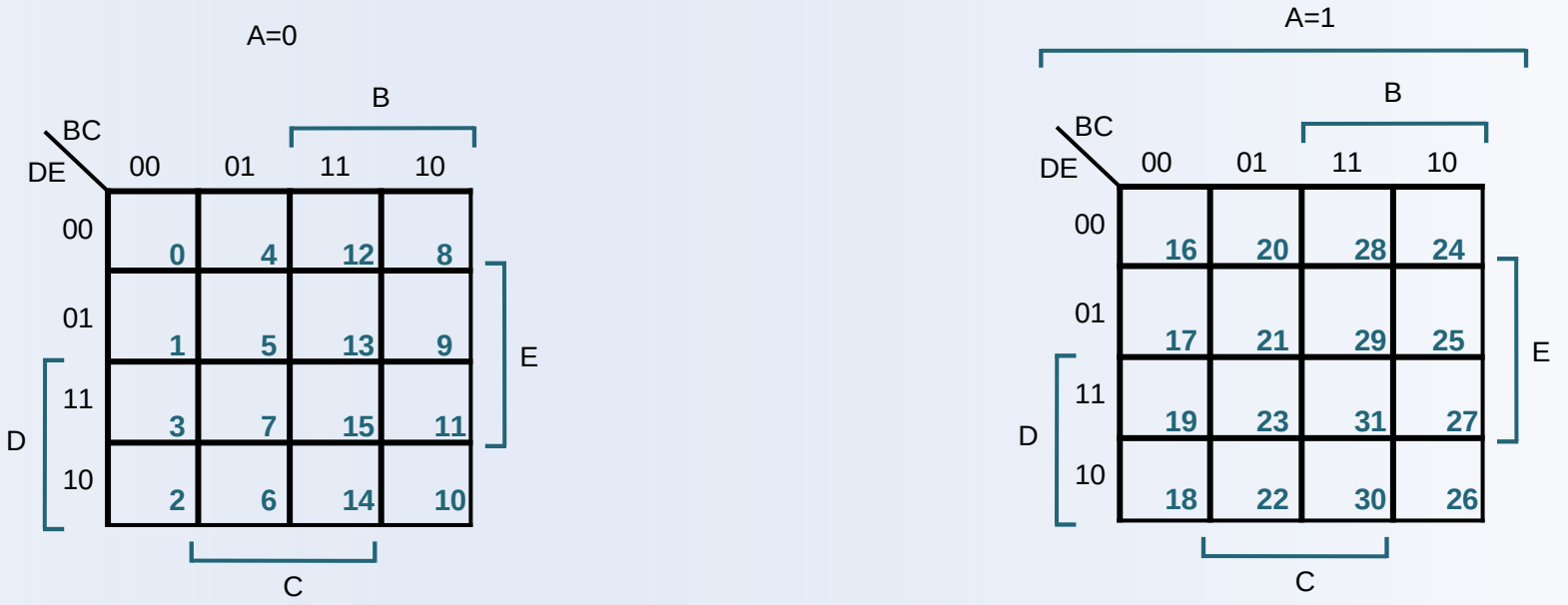
Four Variable Karnaugh Map

F(A,B,C,D) = Σm(0,2,3,5,6,7,8,10,11,14,15)



F = C + A B D + B D

Five Variable Karnaugh Map

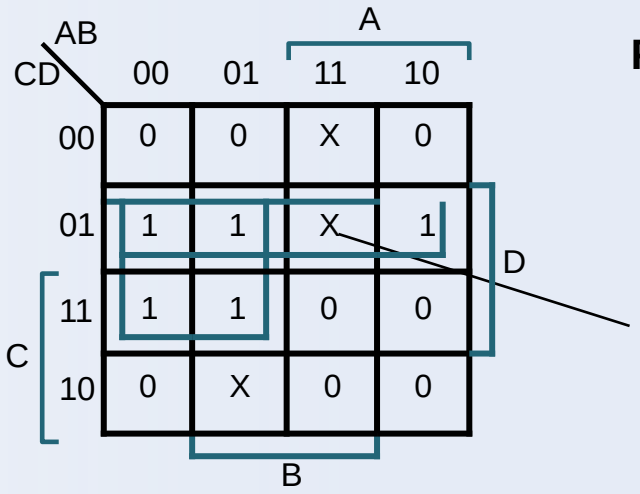


Neighbors of m_5 are: minterms 1, 4, 7, 13, and 21

Neighbors of m_{10} are: minterms 2, 8, 11, 14, and 26

K-Map Example: Don’t Cares

Don't Cares can be treated as 1's or 0's if it is advantageous to do so



F(A,B,C,D) = Σm(1,3,5,7,9) + Σd(6,12,13)

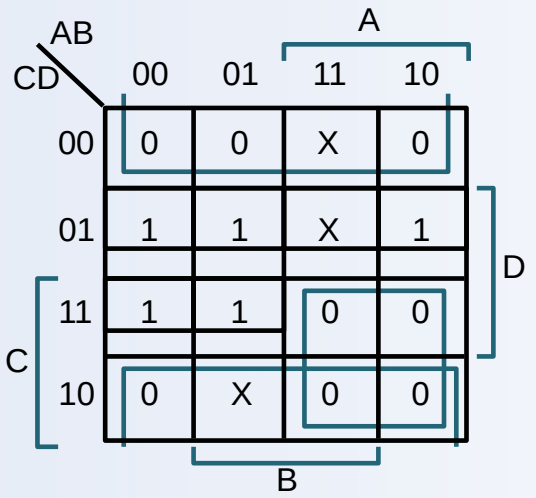
F = $\overline{A}$ D + $\overline{B}$ $\overline{C}$ D w/o don't cares

F = $\overline{C}$ D + $\overline{A}$ D w/ don't cares

By treating this DC as a "1", a 2-cube can be formed rather than one 0-cube

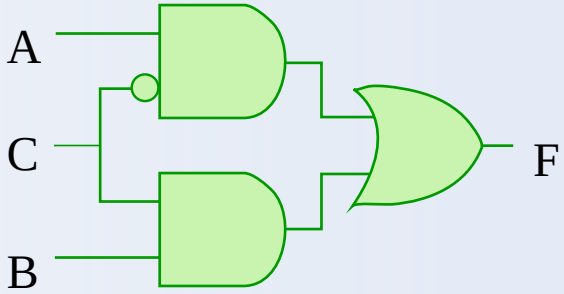
In PoS form: F = D ($\overline{A}$ + $\overline{C}$)

Equivalent answer as above, but fewer literals

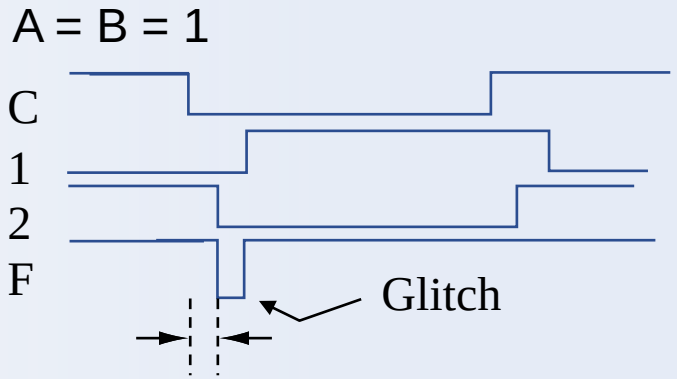


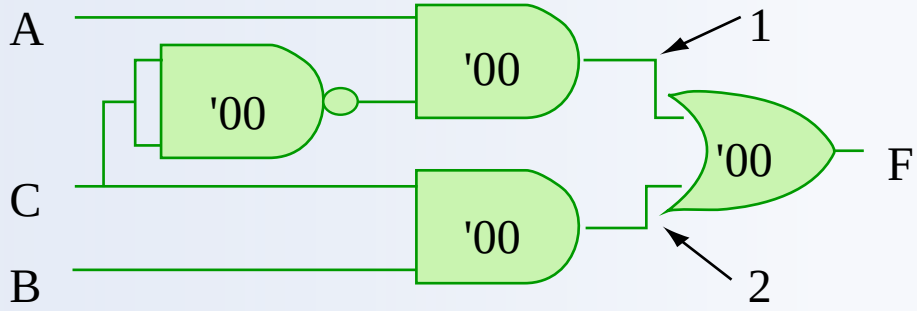
Hazards

Static hazards: Consider this function: F = A * C' + B * C



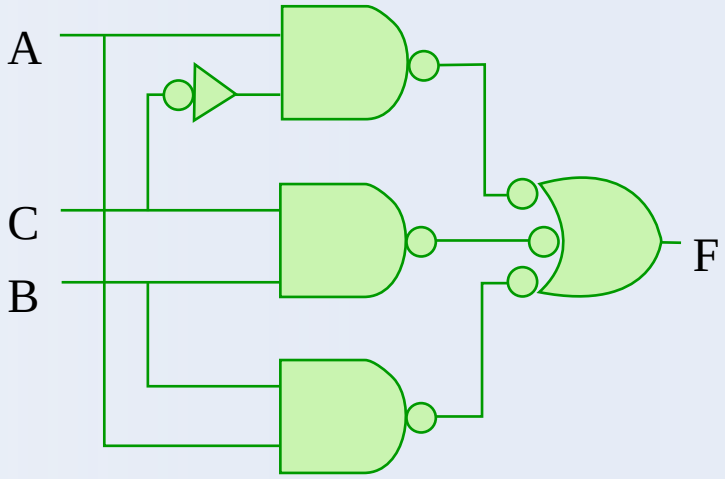
C \ AB	00	01	11	10
0	0	0	1	1
1	0	1	1	0





Fixing Hazards

The glitch is the result of timing differences in parallel data paths. It is associated with the function jumping between groupings or product terms on the K-map. To fix it, cover it up with another grouping or product term!



C \ AB	00	01	11	10
0	0	0	1	1
1	0	1	1	0

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

- **In general, it is difficult to avoid hazards – need a robust design methodology to deal with hazards.**

Practice Examples

Reduce the following Expressions:

- $F(A,B,C)=\sum m(0,1,3,5,7)$
- $G(A,B,C,D)=\sum m(0,1,2,3,5,7,8,10,11,15)$

Expressions with don’t care conditions:

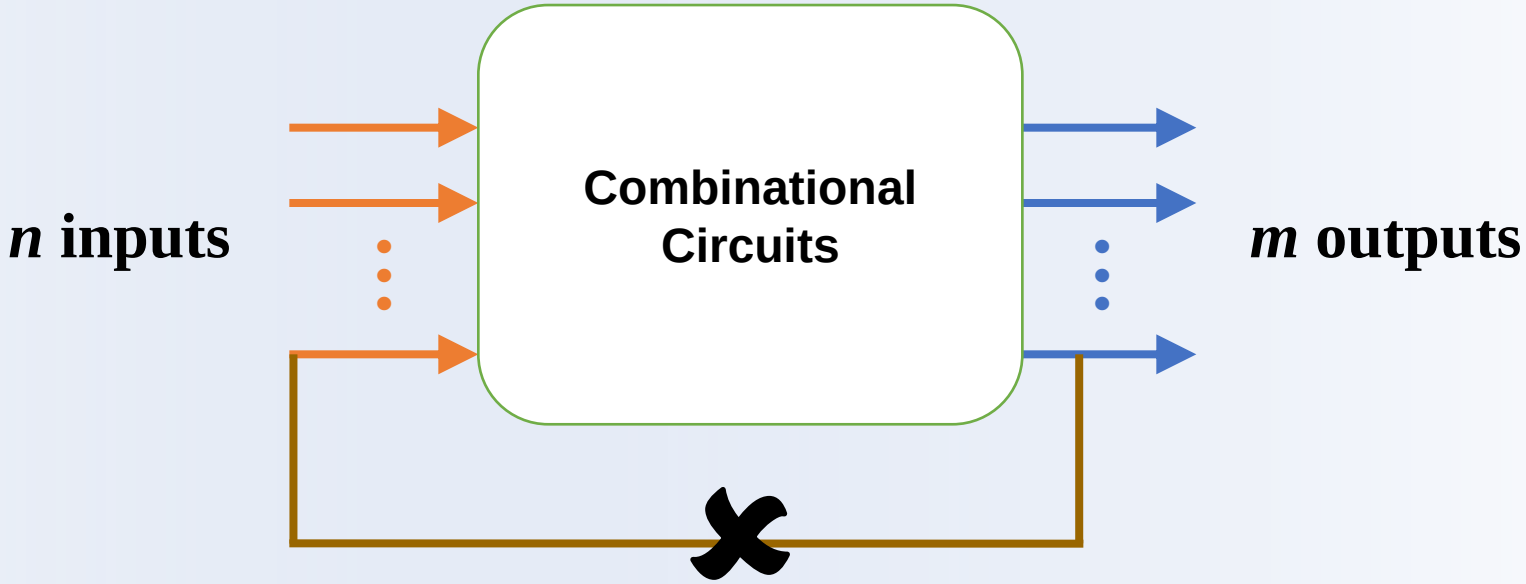
- $F(A,B,C)=\sum m(0,1,3,5)+\sum d(6,7)$
- $G(A,B,C,D)=\sum m(1,3,7,9,11,15)+\sum d(2,6,10,14)$

Combinational Circuits

Design and Analysis

Combinational Circuits

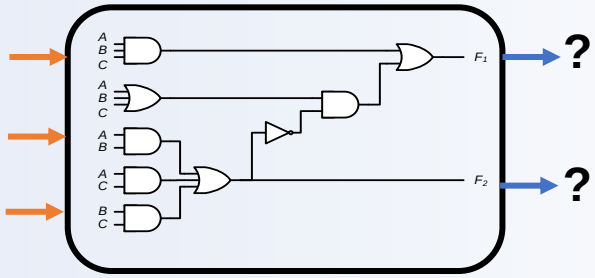
- Output is function of input only
  i.e. no feedback



When input changes, output may change (after a delay)

Combinational Circuits

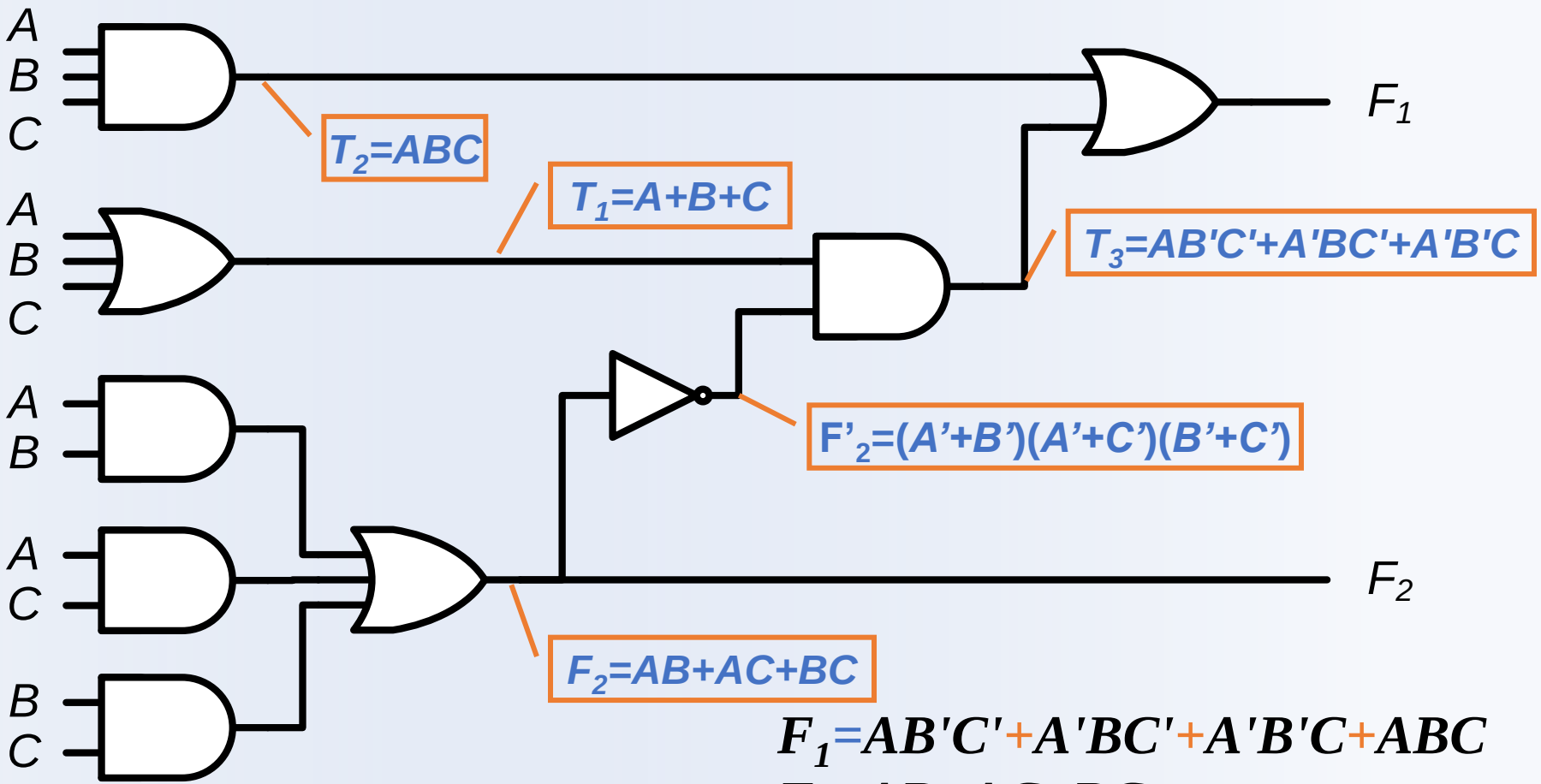
- Analysis
  - Given a circuit, find out its *function*
  - Function may be expressed as:
    - Boolean function
    - Truth table



- Design
  - Given a desired function, determine its *circuit*
  - Function may be expressed as:
    - Boolean function
    - Truth table

Analysis Procedure

- Boolean Expression Approach

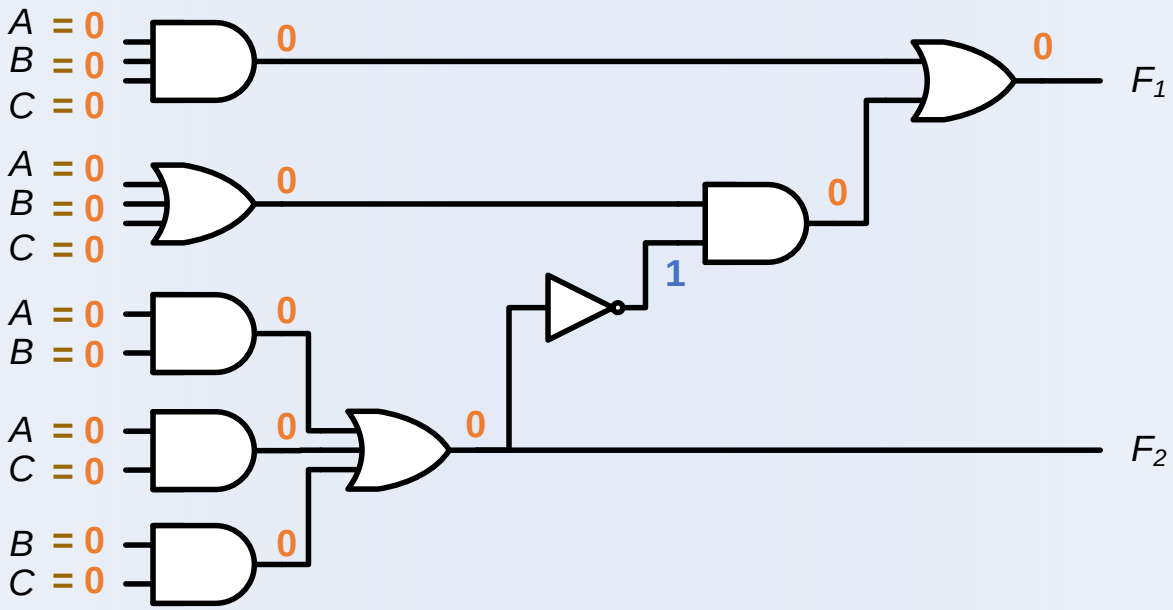


$$F_1 = AB'C' + A'BC' + A'B'C + ABC$$

$$F_2 = AB + AC + BC$$

Analysis Procedure

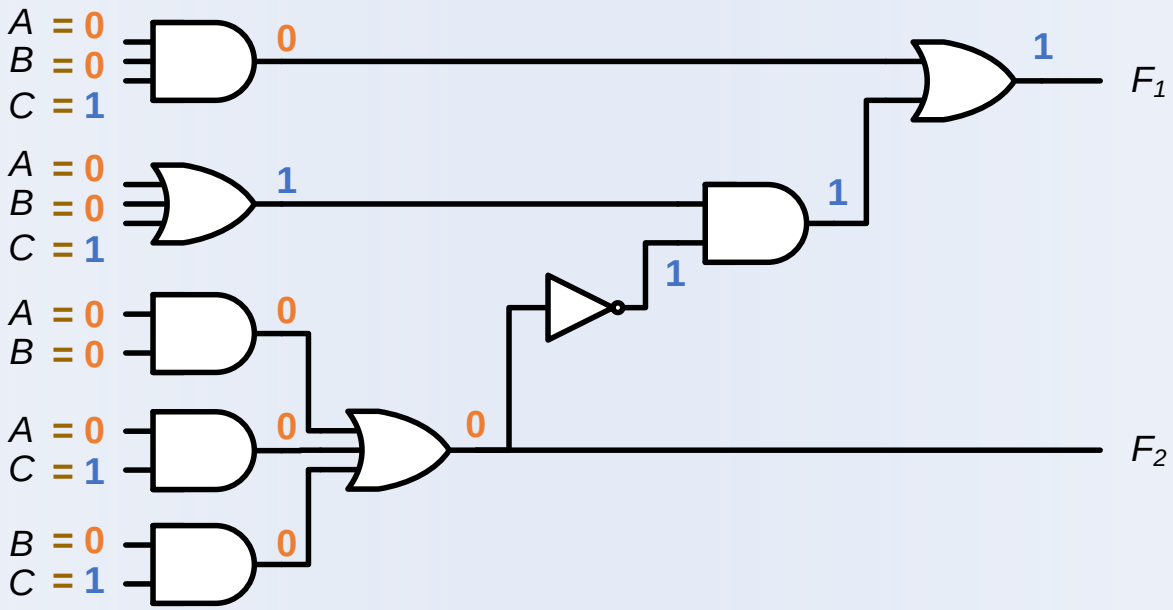
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0

Analysis Procedure

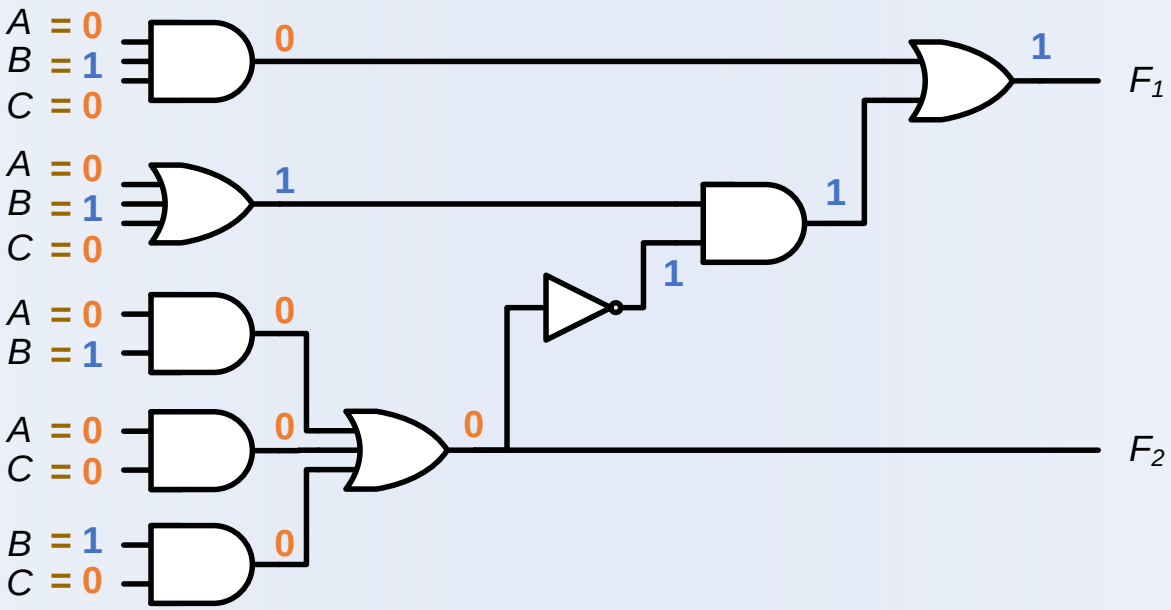
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0

Analysis Procedure

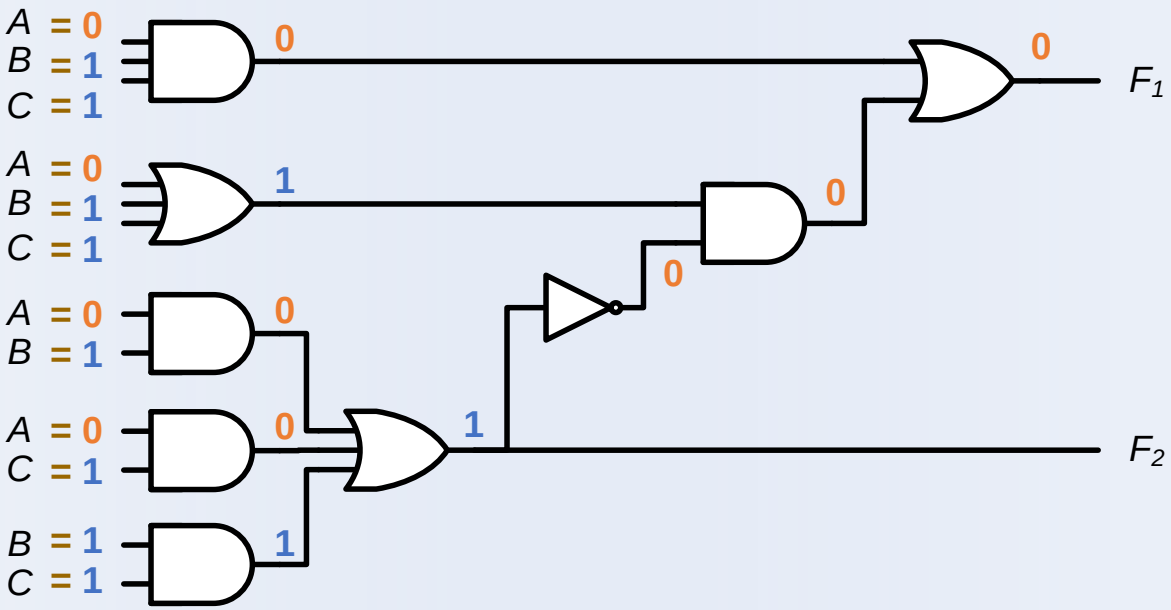
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0

Analysis Procedure

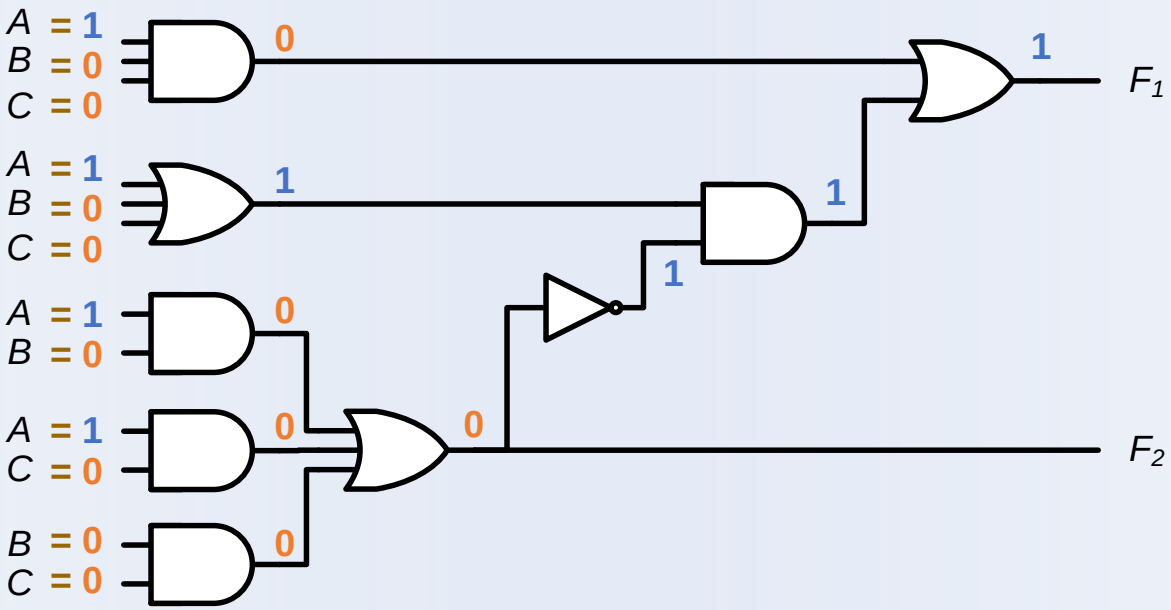
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0
0 1 1	0	1

Analysis Procedure

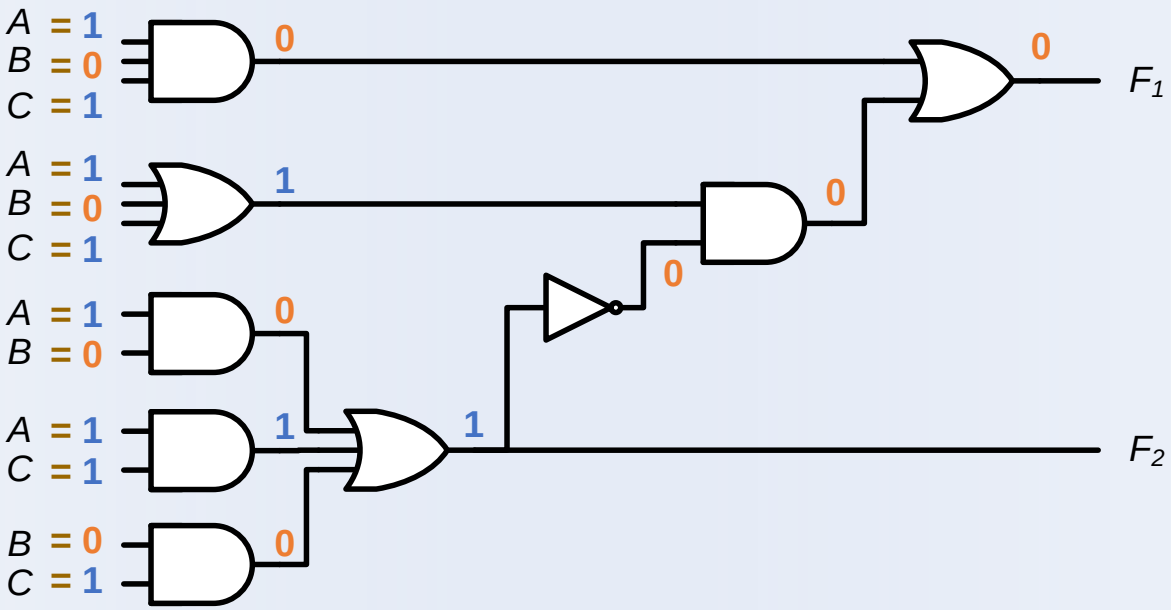
- Truth Table Approach



A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0

Analysis Procedure

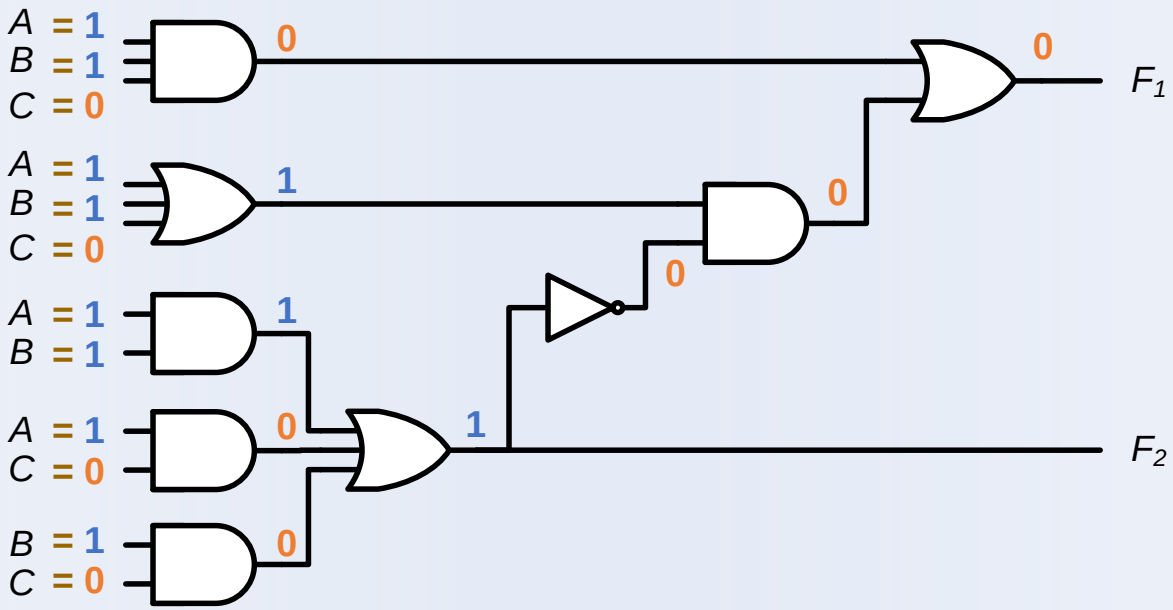
- Truth Table Approach



A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1

Analysis Procedure

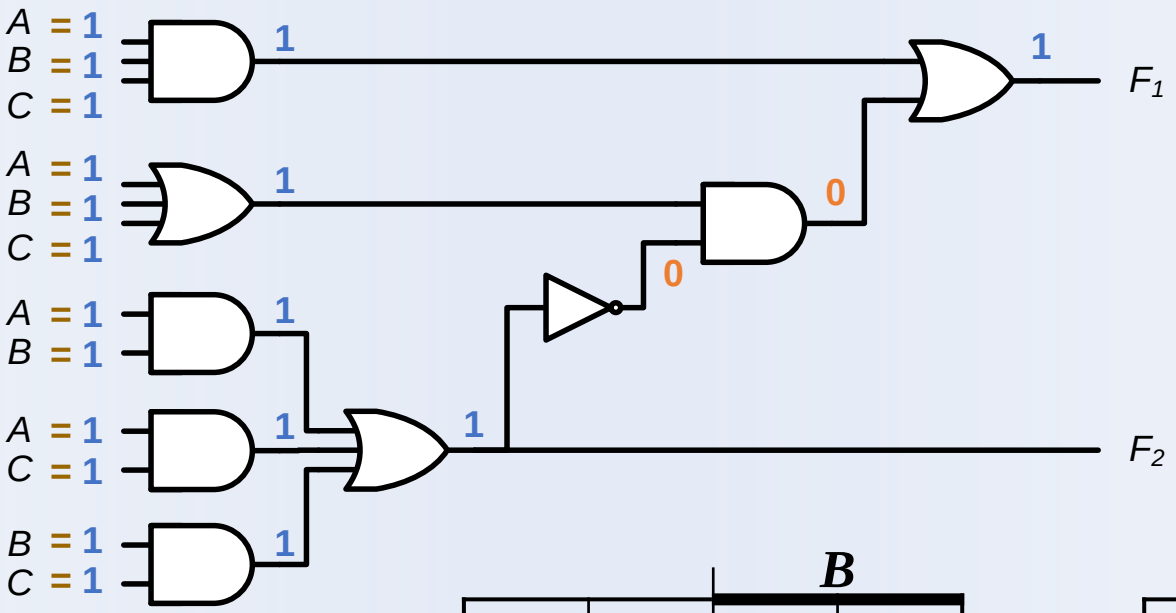
- Truth Table Approach



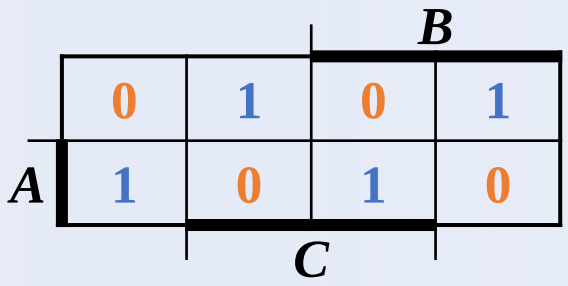
A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1

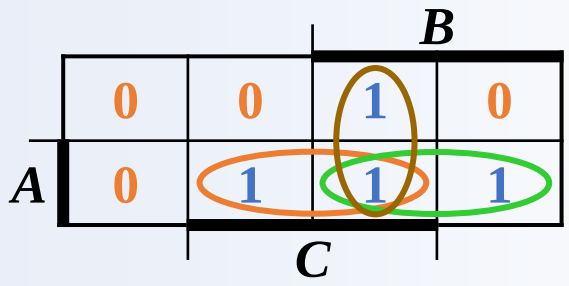
Analysis Procedure

- Truth Table Approach



A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1





$F_1 = AB'C' + A'BC' + A'B'C + ABC$

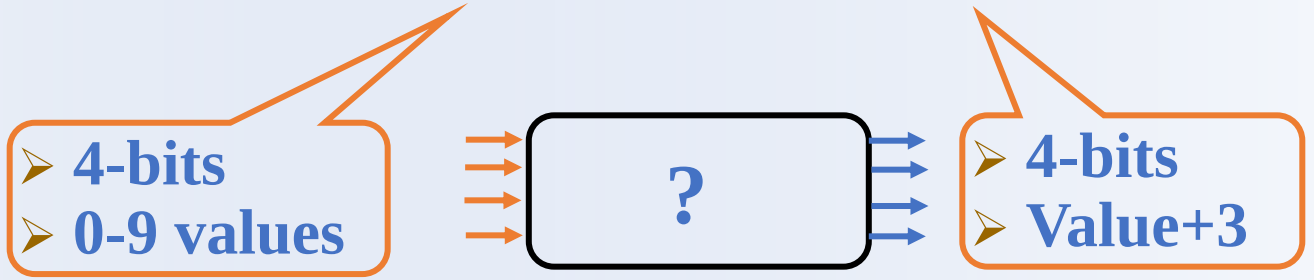
$F_2 = AB + AC + BC$

Design Procedure

- Given a problem statement:
  - Determine the number of *inputs* and *outputs*
  - Derive the truth table
  - Simplify the Boolean expression for each output
  - Produce the required circuit

Example:

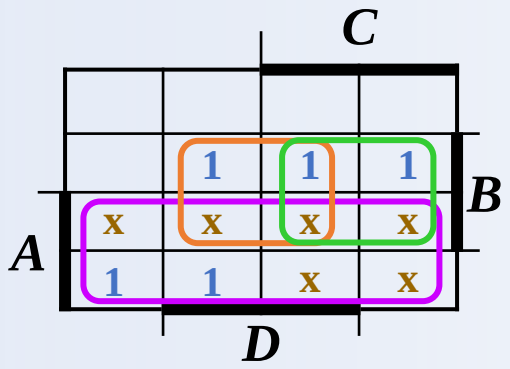
Design a circuit to convert a “BCD” code to “Excess 3” code



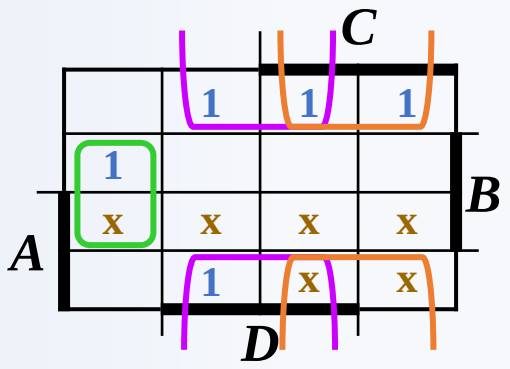
Design Procedure

- BCD-to-Excess 3 Converter

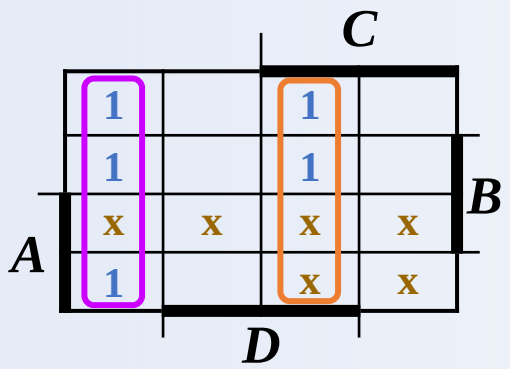
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x



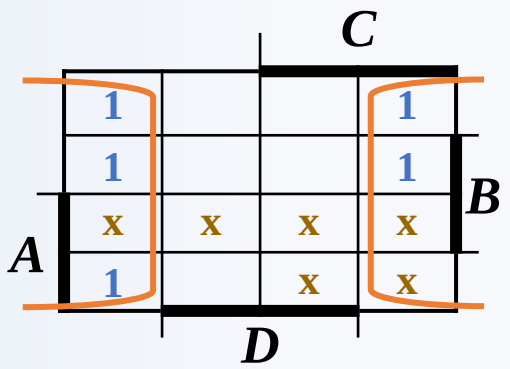
$w = A+BC+BD$



$x = B'C+B'D+BC'D'$



$y = C'D'+CD$

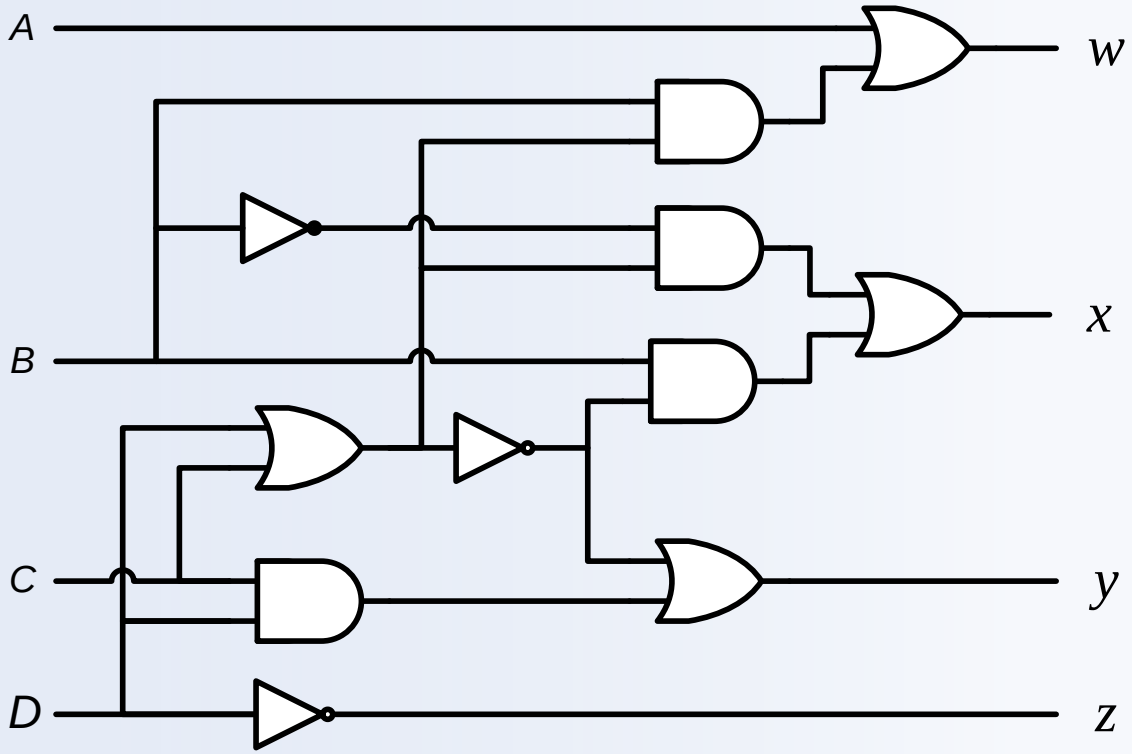


$z = D'$

Design Procedure

- BCD-to-Excess 3 Converter

A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x



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

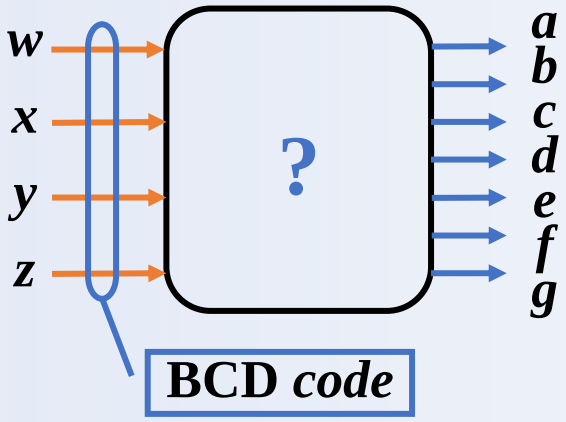
$x = B'(C+D) + B(C+D)'$

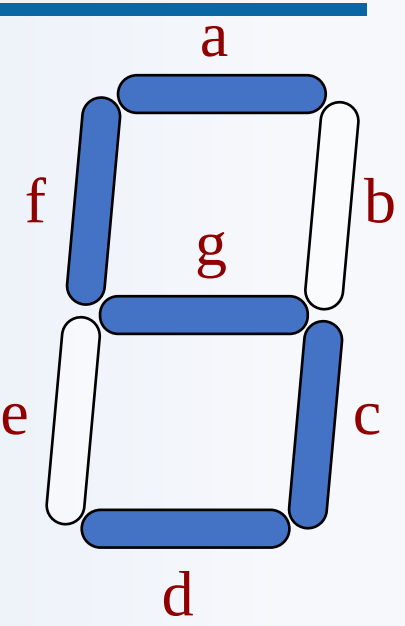
$y = (C+D)' + CD$

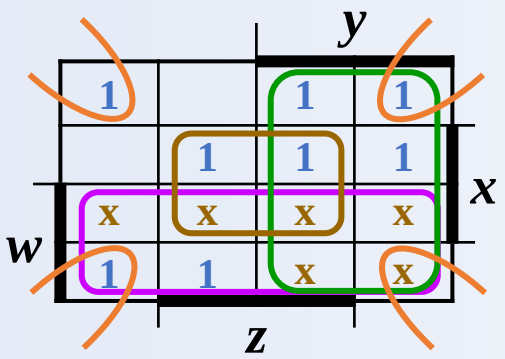
$z = D'$

Seven-Segment Decoder

w x y z	a b c d e f g
0 0 0 0	1 1 1 1 1 1 0
0 0 0 1	0 1 1 0 0 0 0
0 0 1 0	1 1 0 1 1 0 1
0 0 1 1	1 1 1 1 0 0 1
0 1 0 0	0 1 1 0 0 1 1
0 1 0 1	1 0 1 1 0 1 1
0 1 1 0	1 0 1 1 1 1 1
0 1 1 1	1 1 1 0 0 0 0
1 0 0 0	1 1 1 1 1 1 1
1 0 0 1	1 1 1 1 0 1 1
1 0 1 0	x x x x x x x
1 0 1 1	x x x x x x x
1 1 0 0	x x x x x x x
1 1 0 1	x x x x x x x
1 1 1 0	x x x x x x x
1 1 1 1	x x x x x x x







$a = w + y + xz + x'z'$

$b = \ldots$
$c = \ldots$
$d = \ldots$

Adder / Subtractor

Adder / Subtracter Design – how?

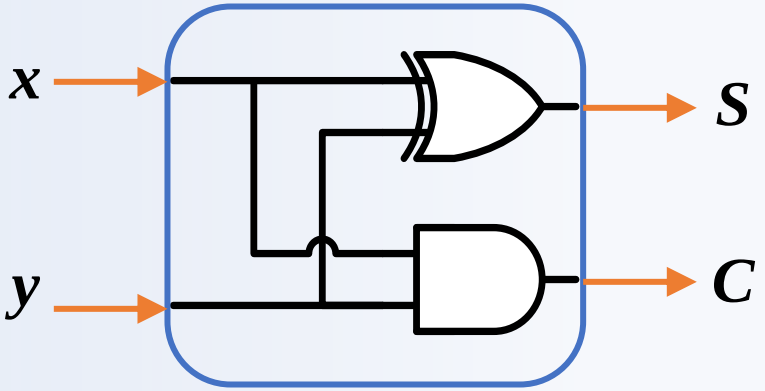
- Truth-table, then determine canonical form, then minimize and implement as we've seen before
- Look at breaking the problem down into smaller pieces that we can cascade or hierarchically layer

Binary Adder

- Half Adder
  - Adds 1-bit plus 1-bit
  - Produces Sum and Carry

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

$$\begin{array}{rr} & x \\ + & y \\ \hline C & S \end{array}$$

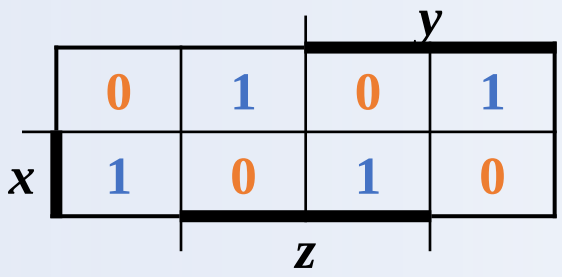


Binary Adder

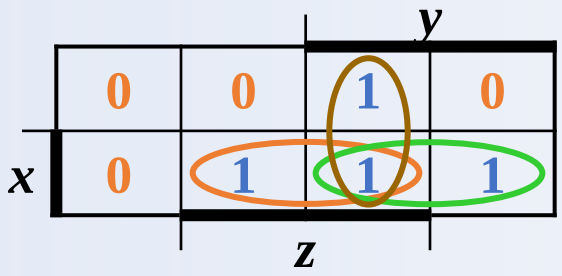
- Full Adder
  - Adds 1-bit plus 1-bit plus 1-bit
  - Produces Sum and Carry

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

$$\begin{array}{rr} & x \\ + & y \\ + & z \\ \hline & C\ \ S \end{array}$$



$$S = xy'z' + x'yz' + x'y'z + xyz = x \oplus y \oplus z$$



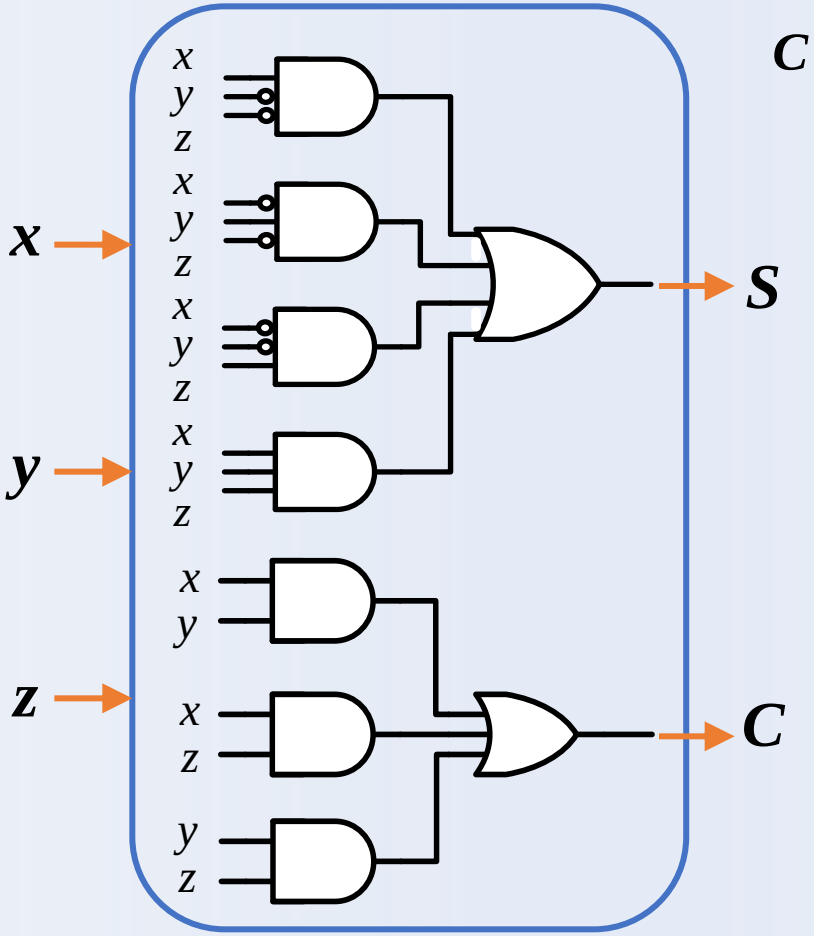
$$C = xy + xz + yz$$

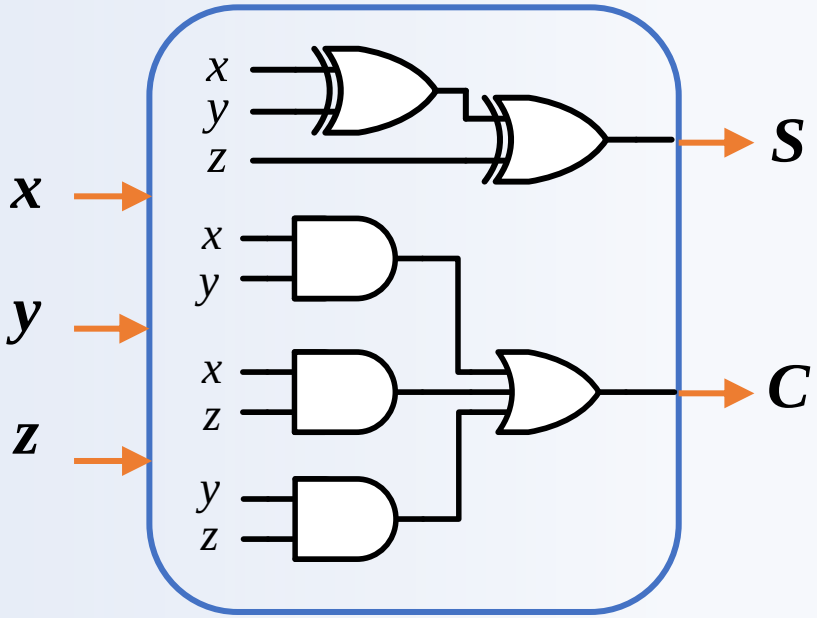
Binary Adder

- Full Adder

$$S = xy'z' + x'yz' + x'y'z + xyz = x \oplus y \oplus z$$

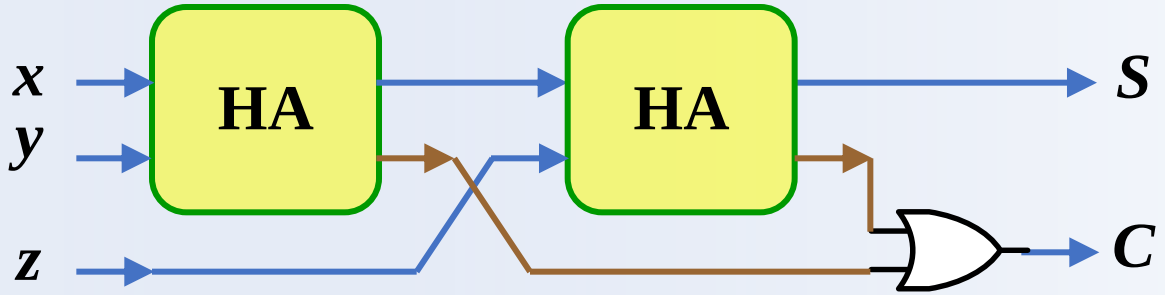
$$C = xy + xz + yz$$

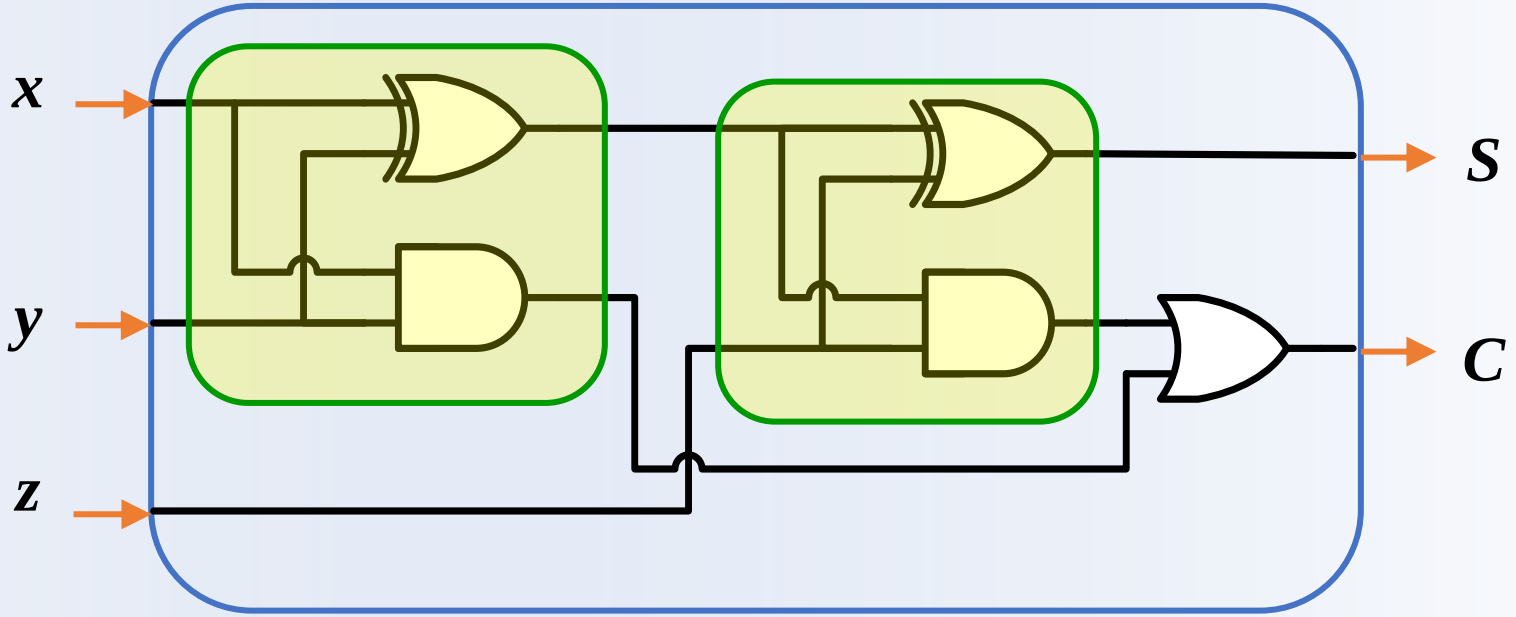




Binary Adder

- Full Adder





Adder / Subtractor – One-bit adder LSB…

$$
\begin{array}{cccc|c}
 & a_3 & a_2 & a_1 & a_0 \\
+ & b_3 & b_2 & b_1 & b_0 \\
\hline
 & s_3 & s_2 & s_1 & s_0
\end{array}
$$

a_0	b_0	s_0	c_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$s_0 =$ a_0 XOR b_0

$c_1 =$ a_0 AND b_0

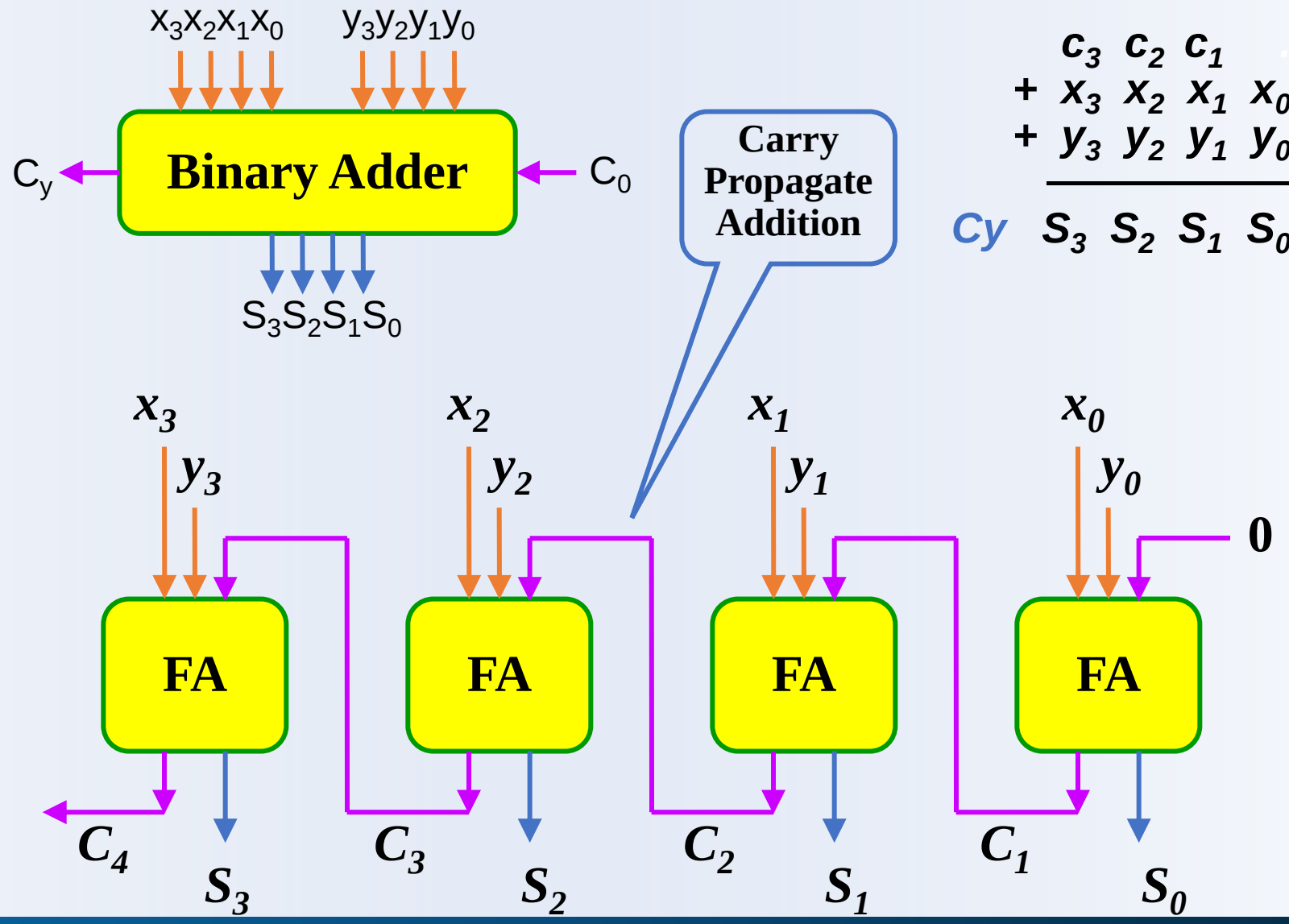
Adder / Subtractor – One-bit adder

$$
\begin{array}{cccccc}
 & a_3 & a_2 & \boxed{a_1} & a_0 \\
+ & b_3 & b_2 & \boxed{b_1} & b_0 \\
\hline
 & s_3 & s_2 & \boxed{s_1} & s_0
\end{array}
$$

a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

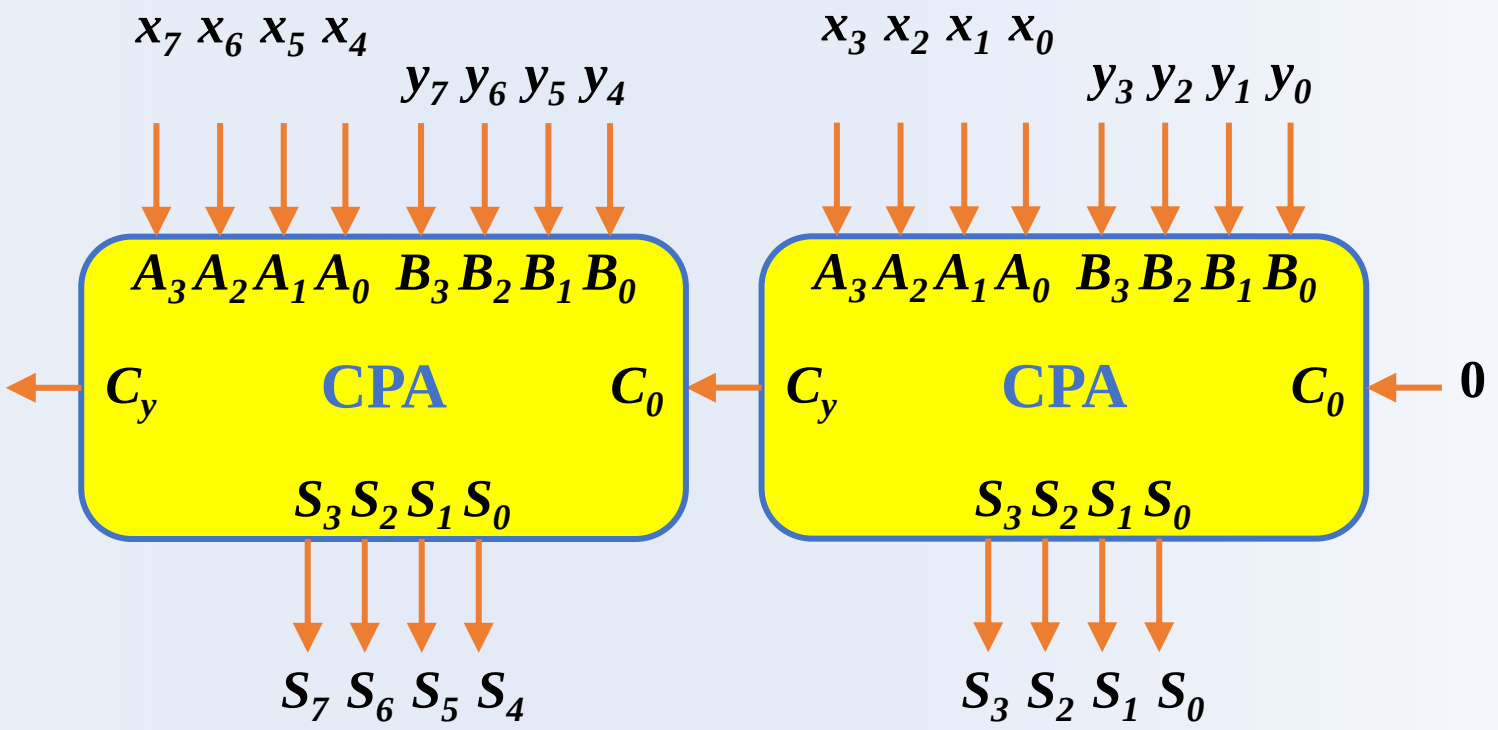
$$
\begin{aligned}
s_i &= \mathrm{XOR}(a_i, b_i, c_i) \\
c_{i+1} &= \mathrm{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i
\end{aligned}
$$

Binary Adder



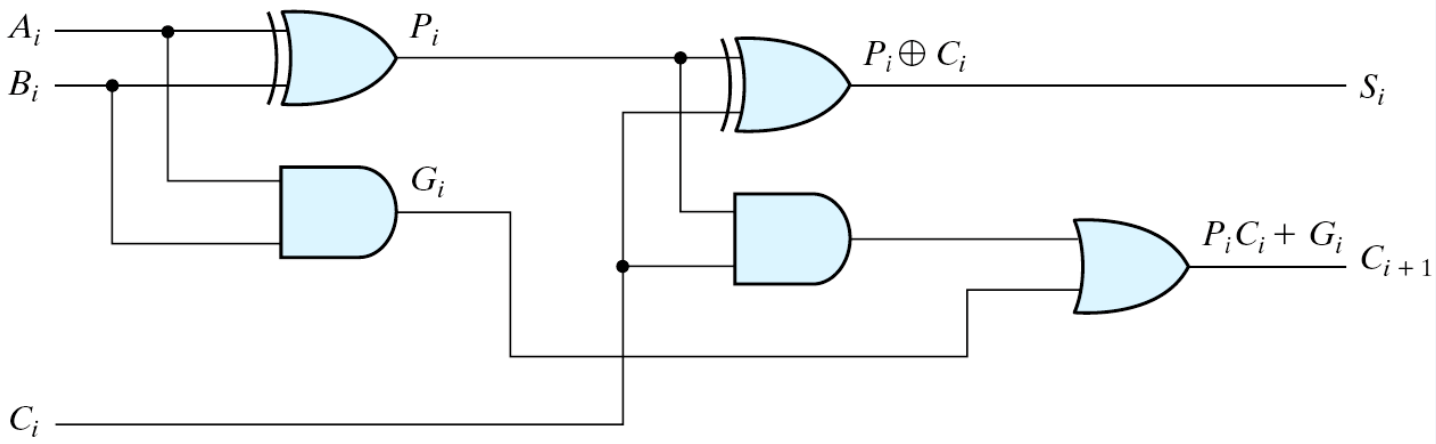
Binary Adder

- Carry Propagate Adder



Carry propagation

- When the correct outputs are available
- The critical path counts (the worst case)
- $(A_1, B_1, C_1) \rightarrow C_2 \rightarrow C_3 \rightarrow C_4 \rightarrow (C_5, S_4)$



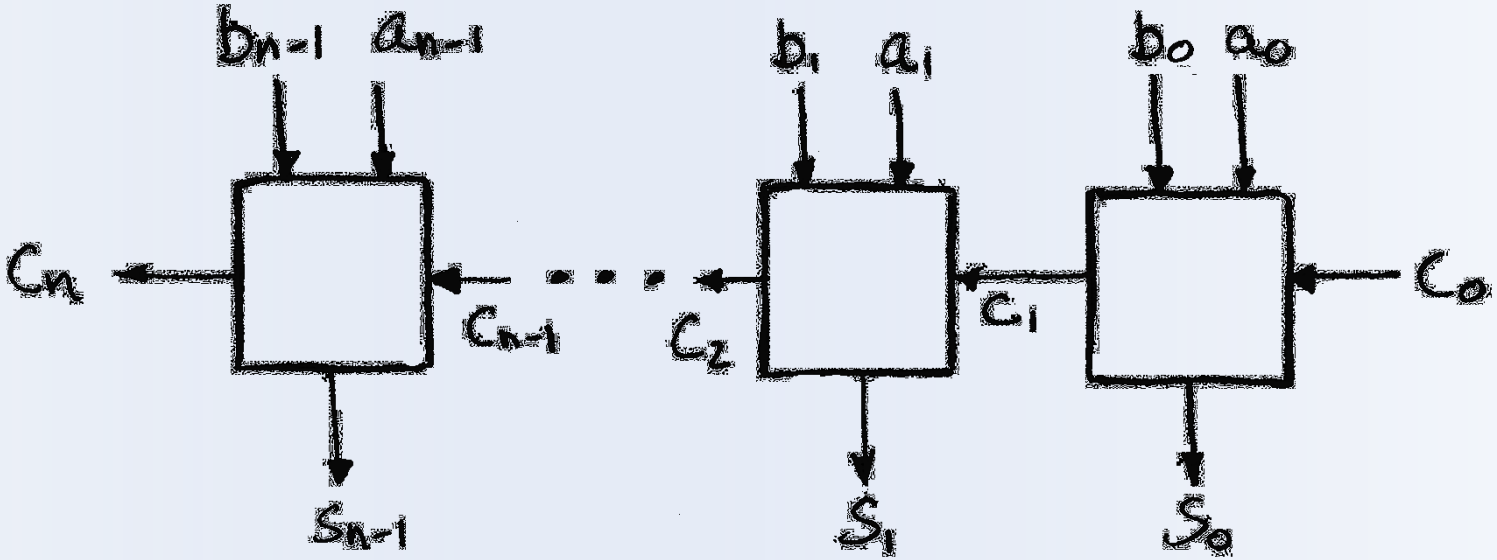
$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P_1G_0 + P_1P_0C_0$

$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

$C_4 = G_3 + P_3C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$

N 1-bit adders → 1 N-bit adder



What about overflow?

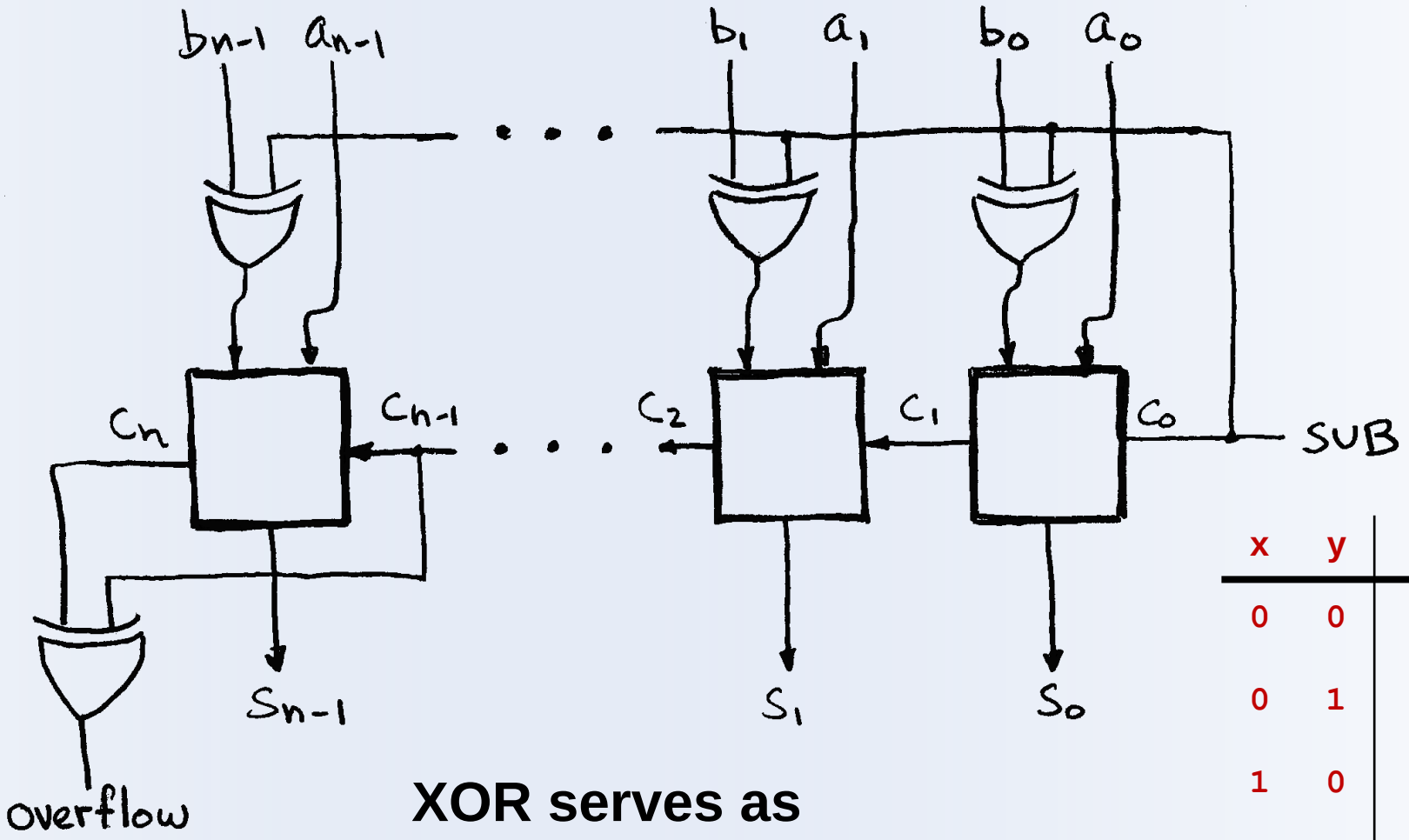
Overflow in Addition of Digital Circuits

Overflow occurs in digital circuits when the result of an arithmetic operation exceeds the range that can be represented with the given number of bits. This typically happens in **signed number arithmetic** when:

- Adding two positive numbers results in a negative number.
- Adding two negative numbers results in a positive number.

Subtractor Design

Extremely Clever Subtractor: A-B = A + (-B)

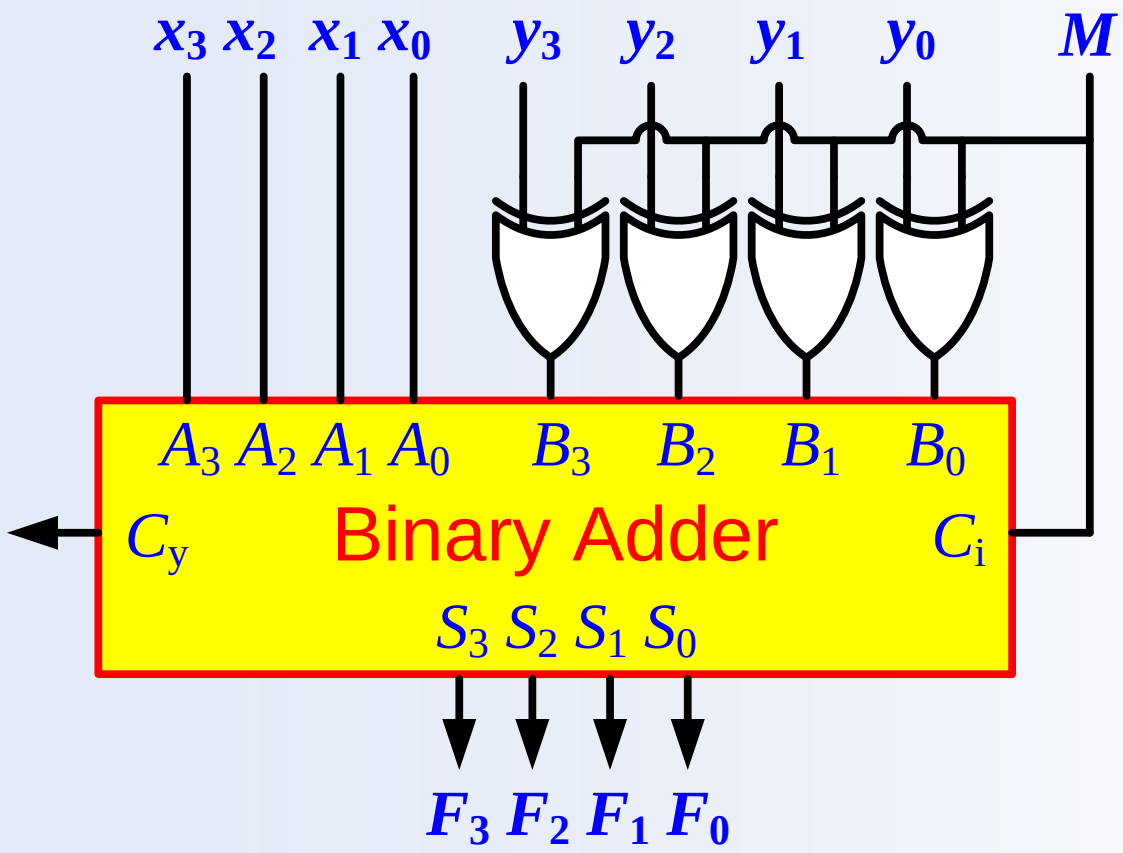


x	y	XOR(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

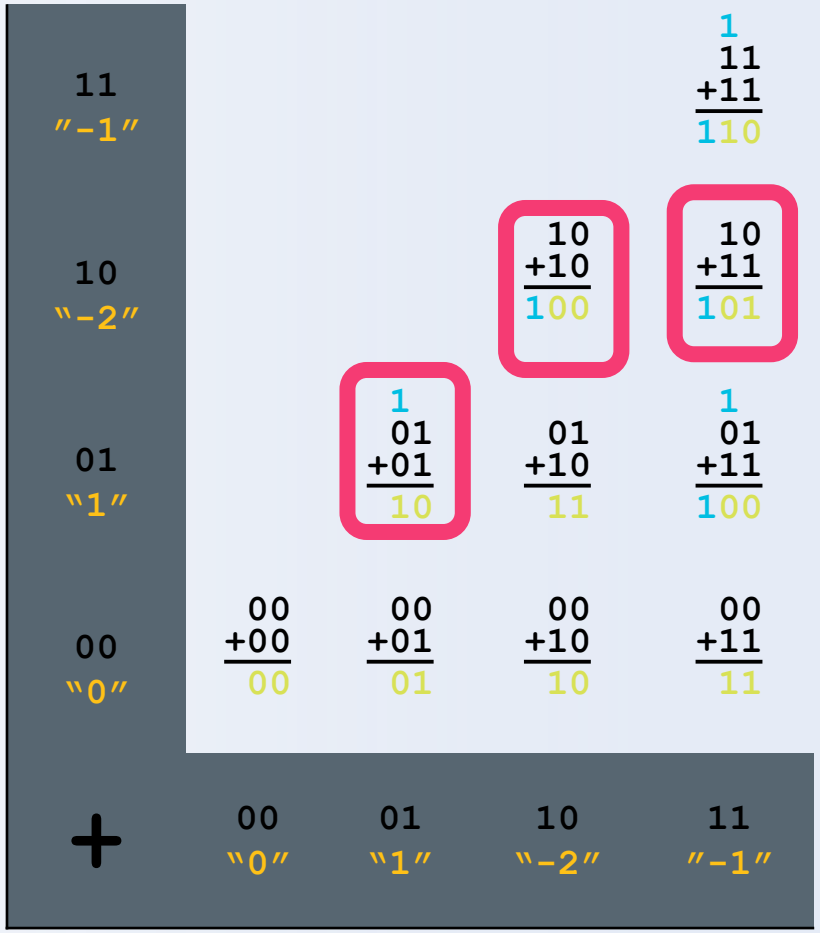
XOR serves as conditional inverter!

Binary Adder/Subtractor

- *M*: Control Signal (Mode)
  - *M*=0 ➔ *F* = *x* + *y*
  - *M*=1 ➔ *F* = *x* – *y*



Sum of two 2-bit numbers…



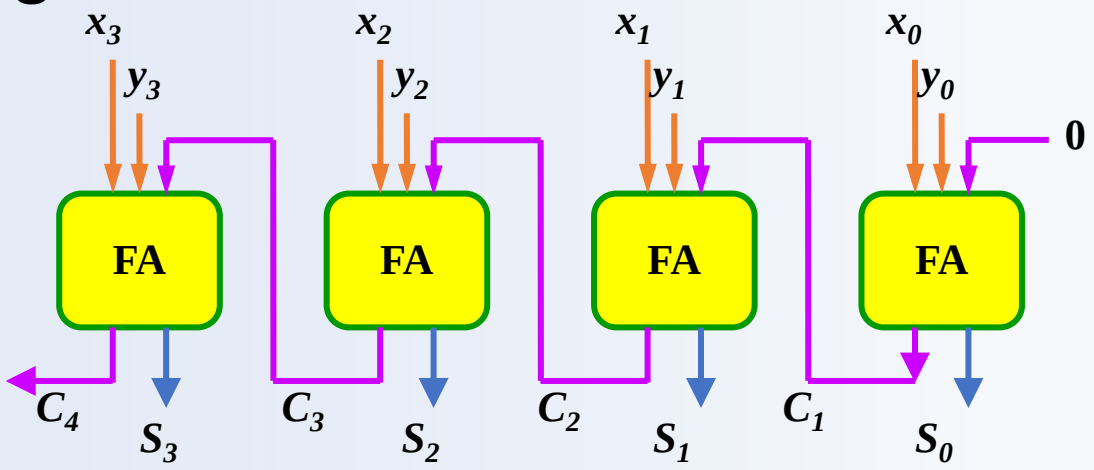
- Let's add
  - First unsigned
  - Then signed (Two's Complement)
    - When do the lowest 2 bits of sum not represent correct sum?
    - Is there a pattern of when this happens? Hint: Check out the carry-bit and the sum-4s-column-bit
- Highest adder
  - C_{in} = Carry-in = c_1, C_{out} = Carry-out = c_2
  - No C_{out} or C_{in} → NO overflow!
  - C_{in} and C_{out} → NO overflow!
  - C_{in}, but no C_{out} → A,B both > 0, overflow!
  - C_{out}, but no C_{in} → A or B are -2, overflow!

What operation is this?

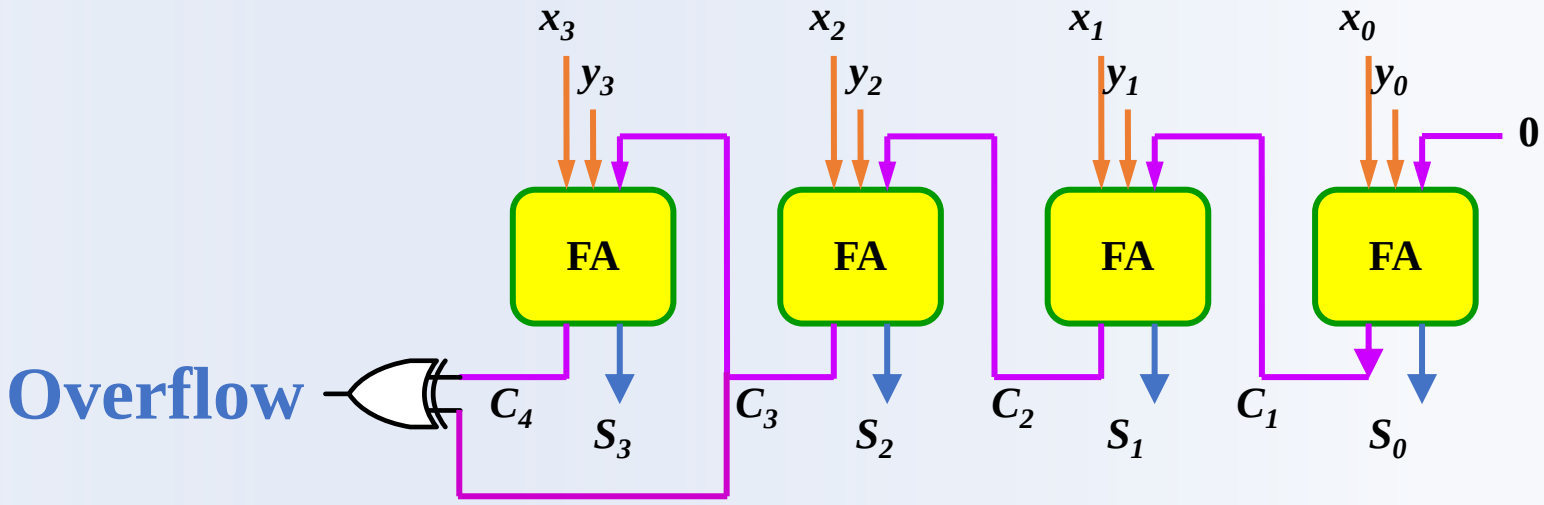
overflow = c_n XOR c_{n-1}

Overflow

- Unsigned Binary Numbers

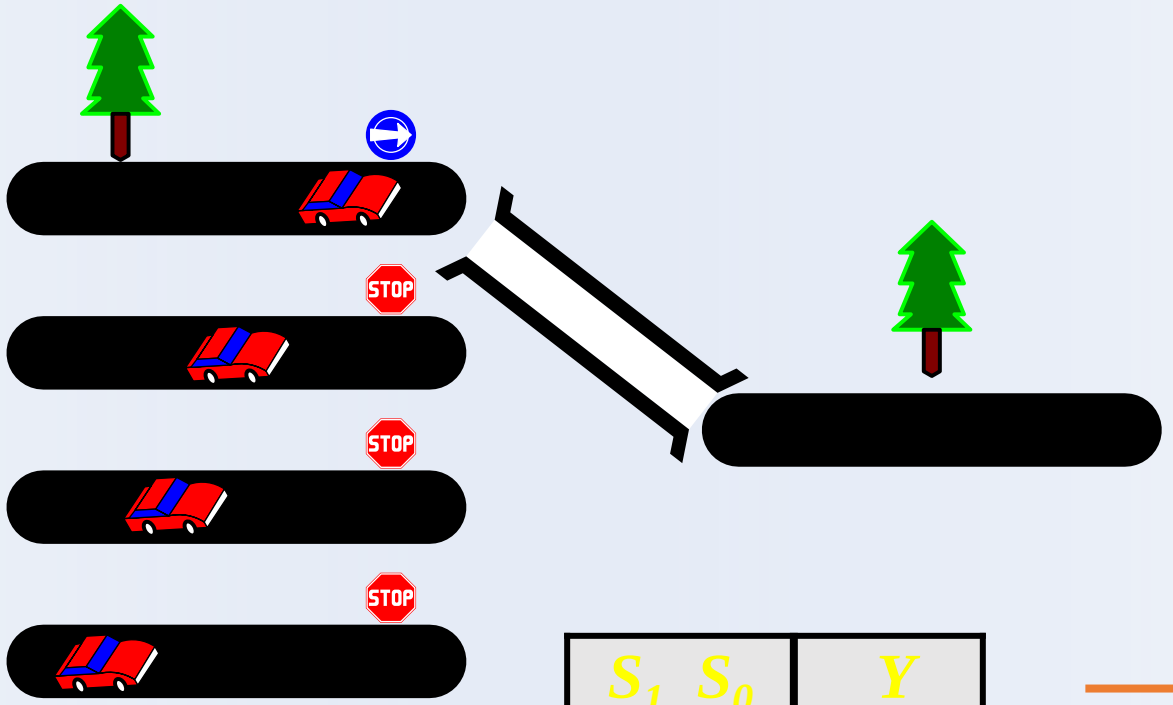


- 2's Complement Numbers

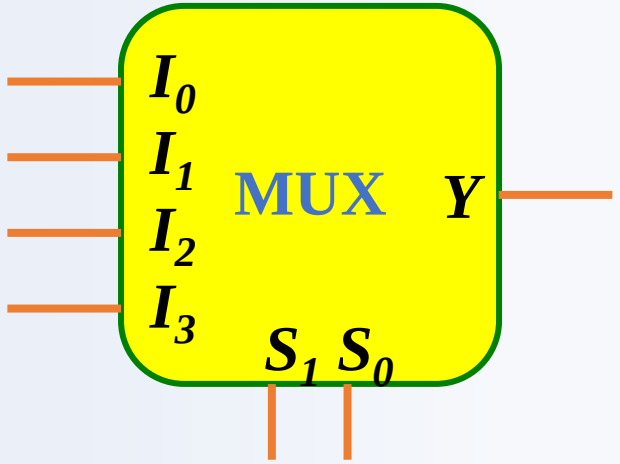


Data Multiplexers

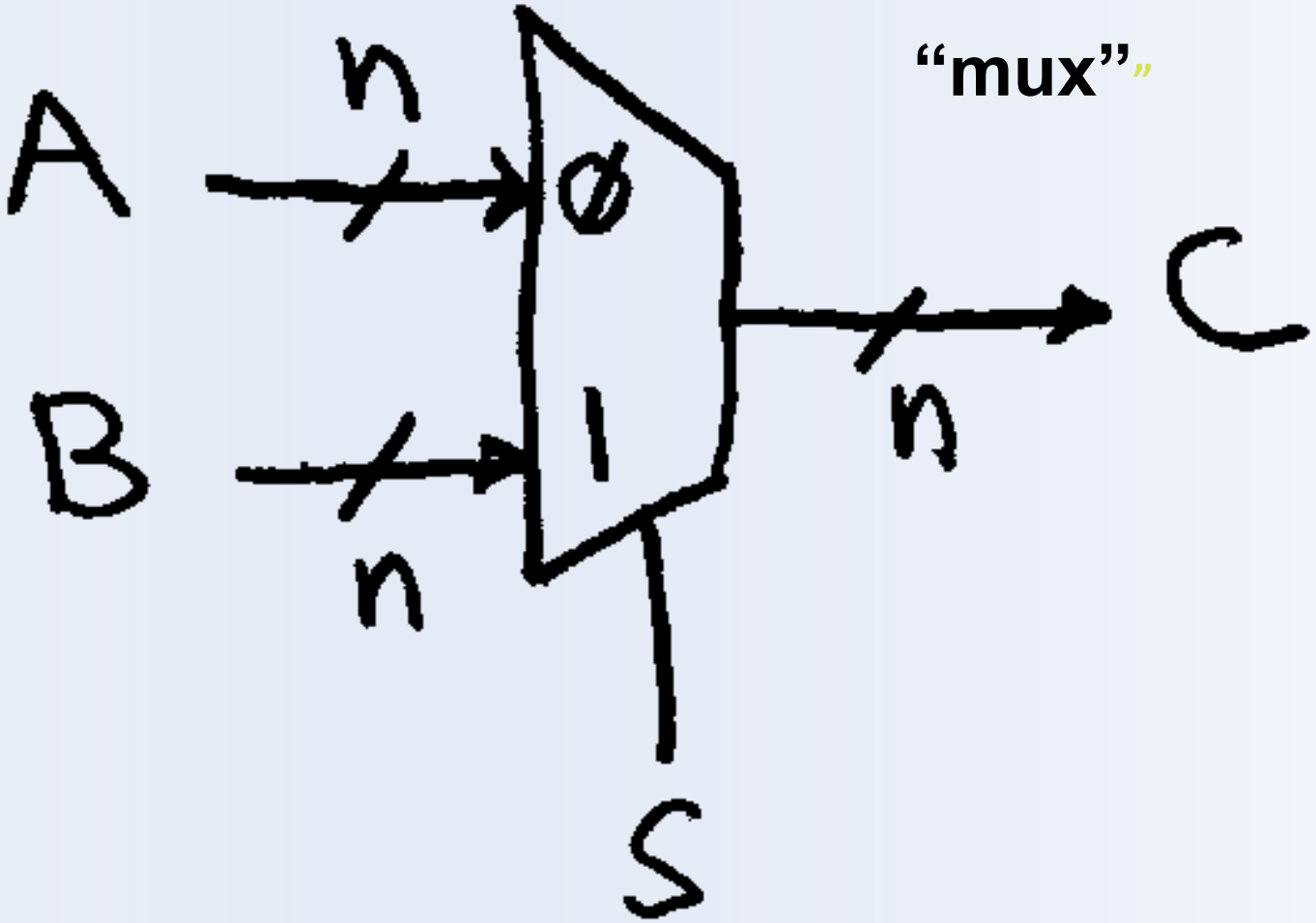
Multiplexers



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

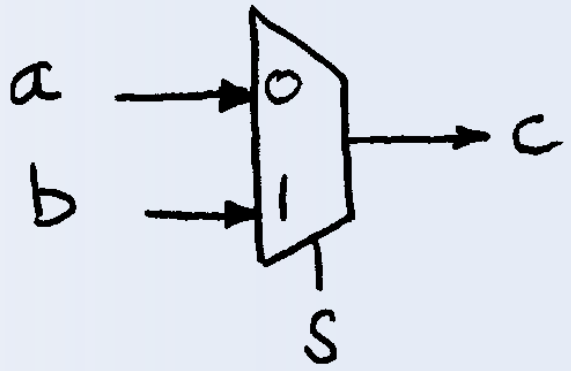


Data Multiplexer (here 2-to-1, n-bit-wide)



“mux”

N instances of 1-bit-wide mux



How many rows in TT?

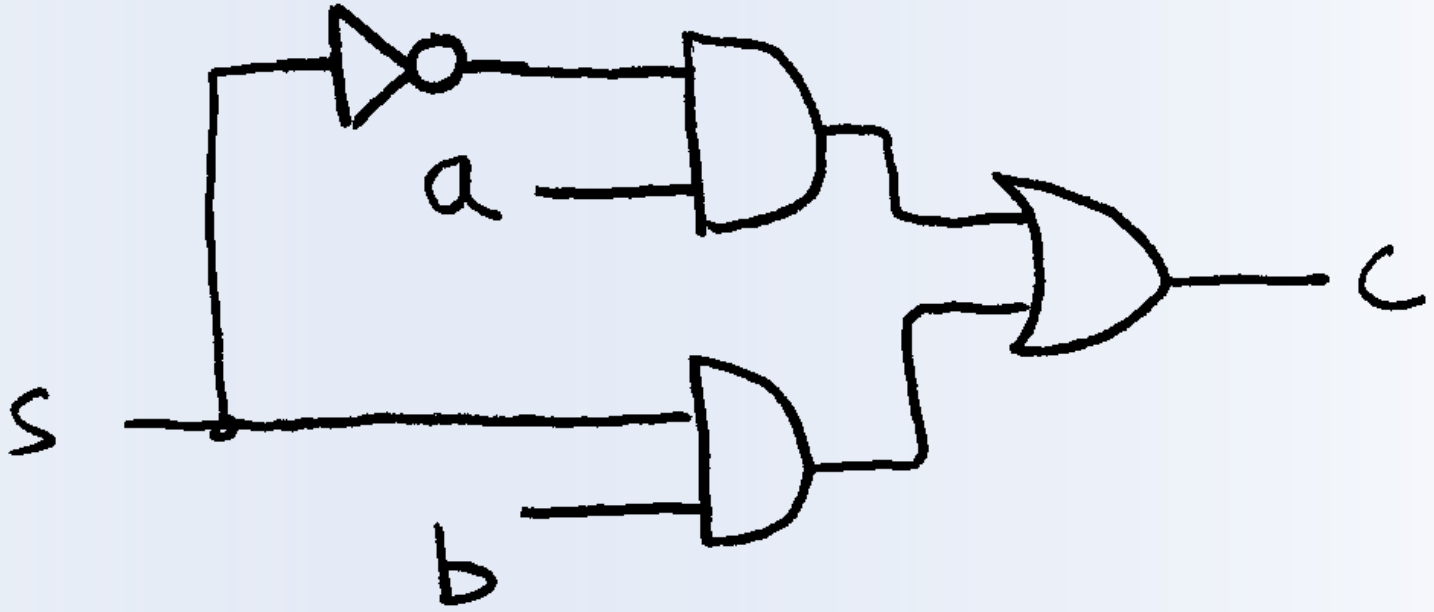
s	ab	c
0	00	0
0	01	0
0	10	1
0	11	1
1	00	0
1	01	1
1	10	0
1	11	1

$$
\begin{aligned}
c &= \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab \\
&= \bar{s}(a\bar{b} + ab) + s(\bar{a}b + ab) \\
&= \bar{s}(a(\bar{b} + b)) + s((\bar{a} + a)b) \\
&= \bar{s}(a(1) + s((1)b) \\
&= \bar{s}a + sb
\end{aligned}
$$

s	c
0	a
1	b

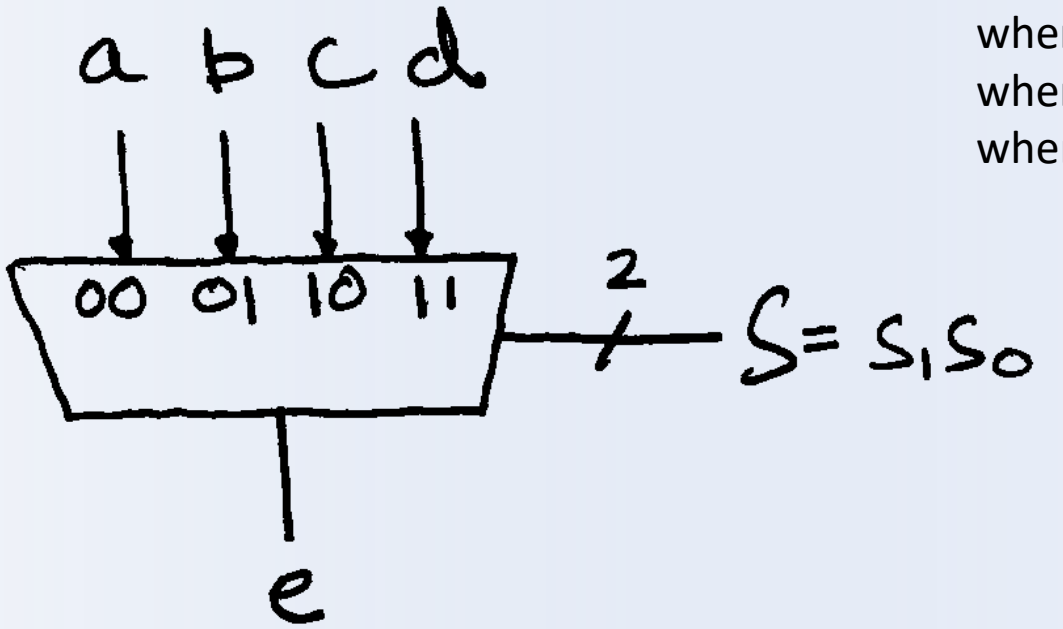
How do we build a 1-bit-wide mux?

$$\bar{s}a + sb$$



4-to-1 Multiplexer?

- How many rows in the Truth Table?



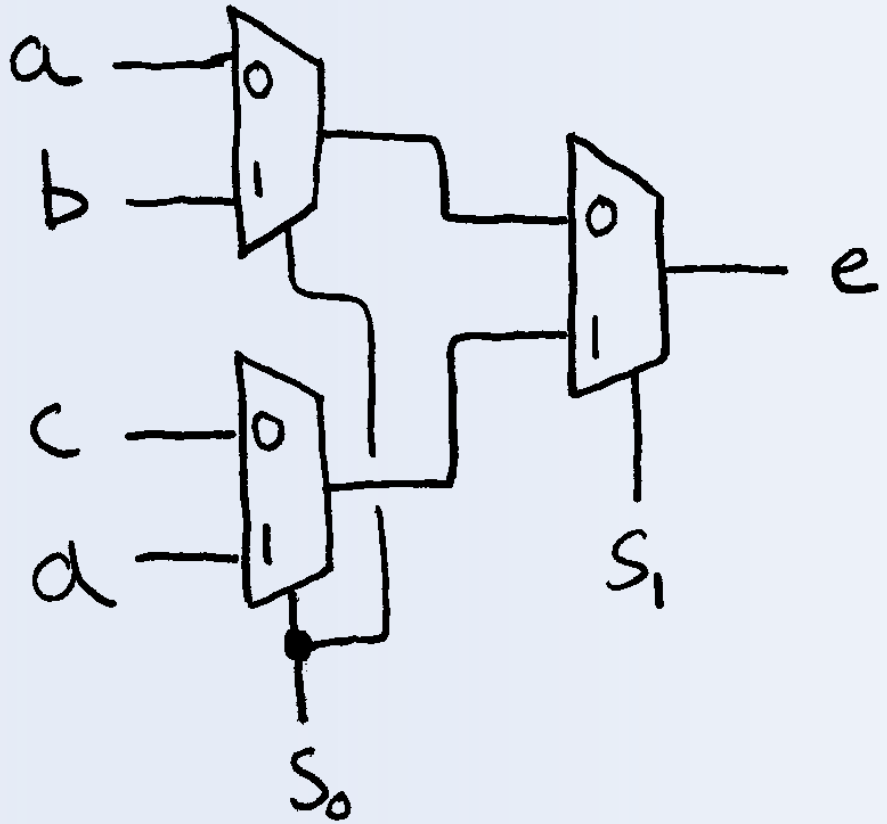
when S=00, e=a
when S=01, e=b
when S=10, e=c
when S=11, e=d

$s_1 s_0$	c
0 0	a
0 1	b
1 0	c
1 1	d

$$e = \overline{s_1} \cdot \overline{s_0} a + \overline{s_1} s_0 b + s_1 \overline{s_0} c + s_1 s_0 d$$

Mux: is there any other way to do it?

Hint: NCAA tourney!

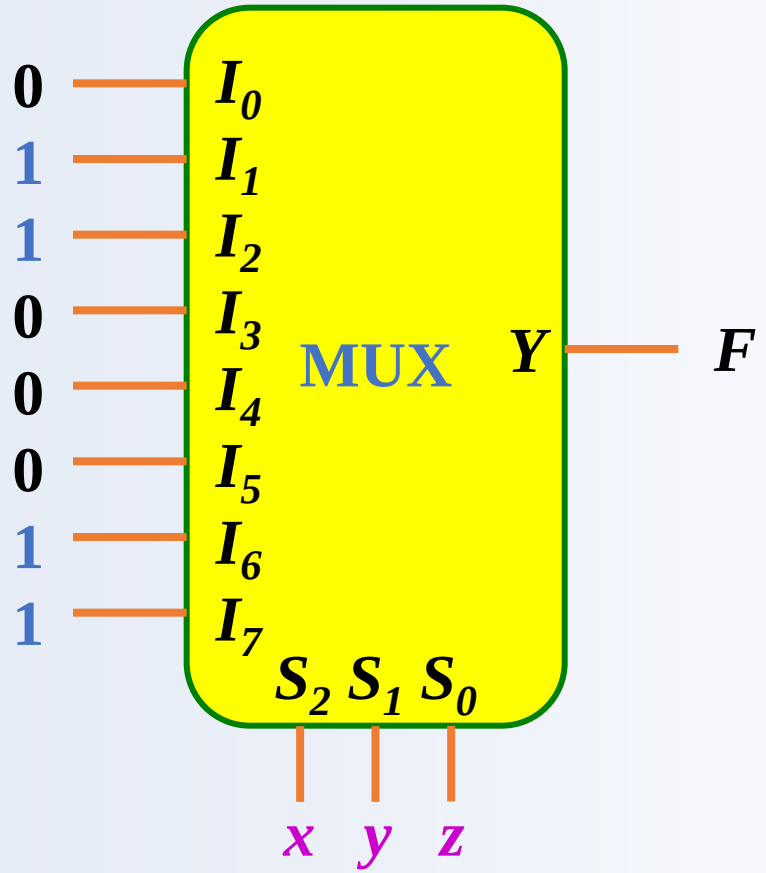


Ans: Hierarchically!

Implementation Using Multiplexers

- Example
  $F(x, y, z) = \sum(1, 2, 6, 7)$

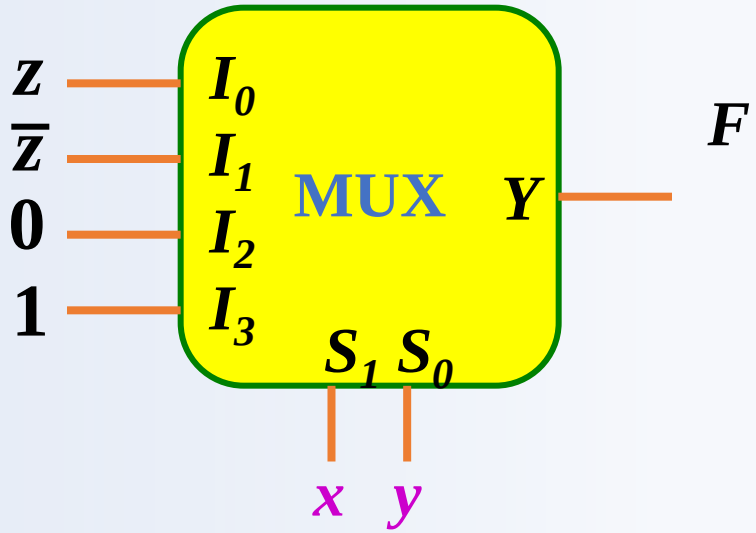
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Implementation Using Multiplexers

- Example
  $F(x, y, z) = \sum(1, 2, 6, 7)$

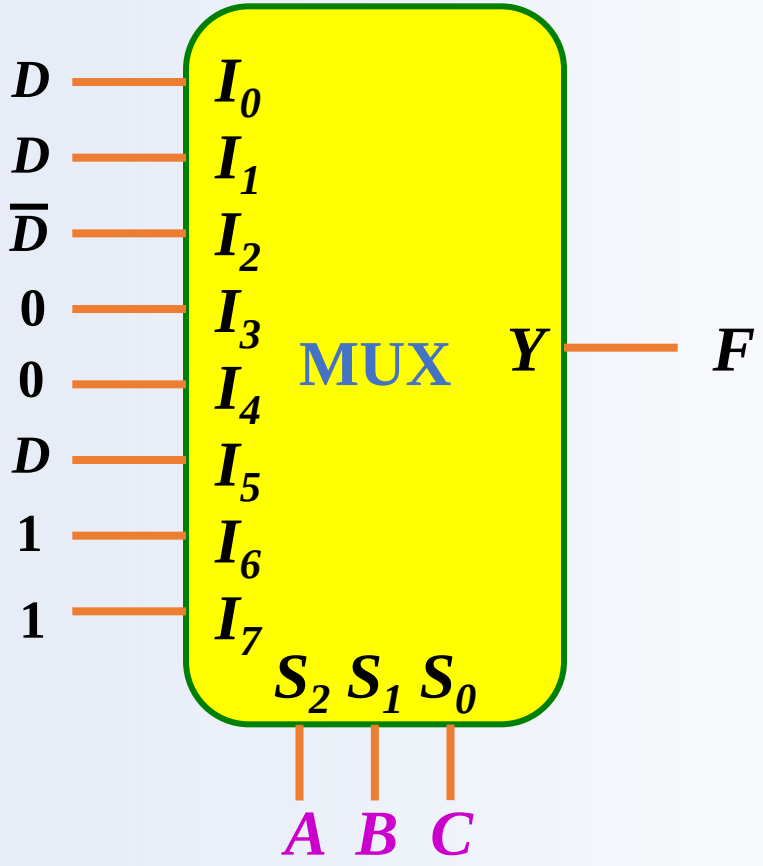
x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = \overline{z}$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	



Implementation Using Multiplexers

- Example
  $F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$

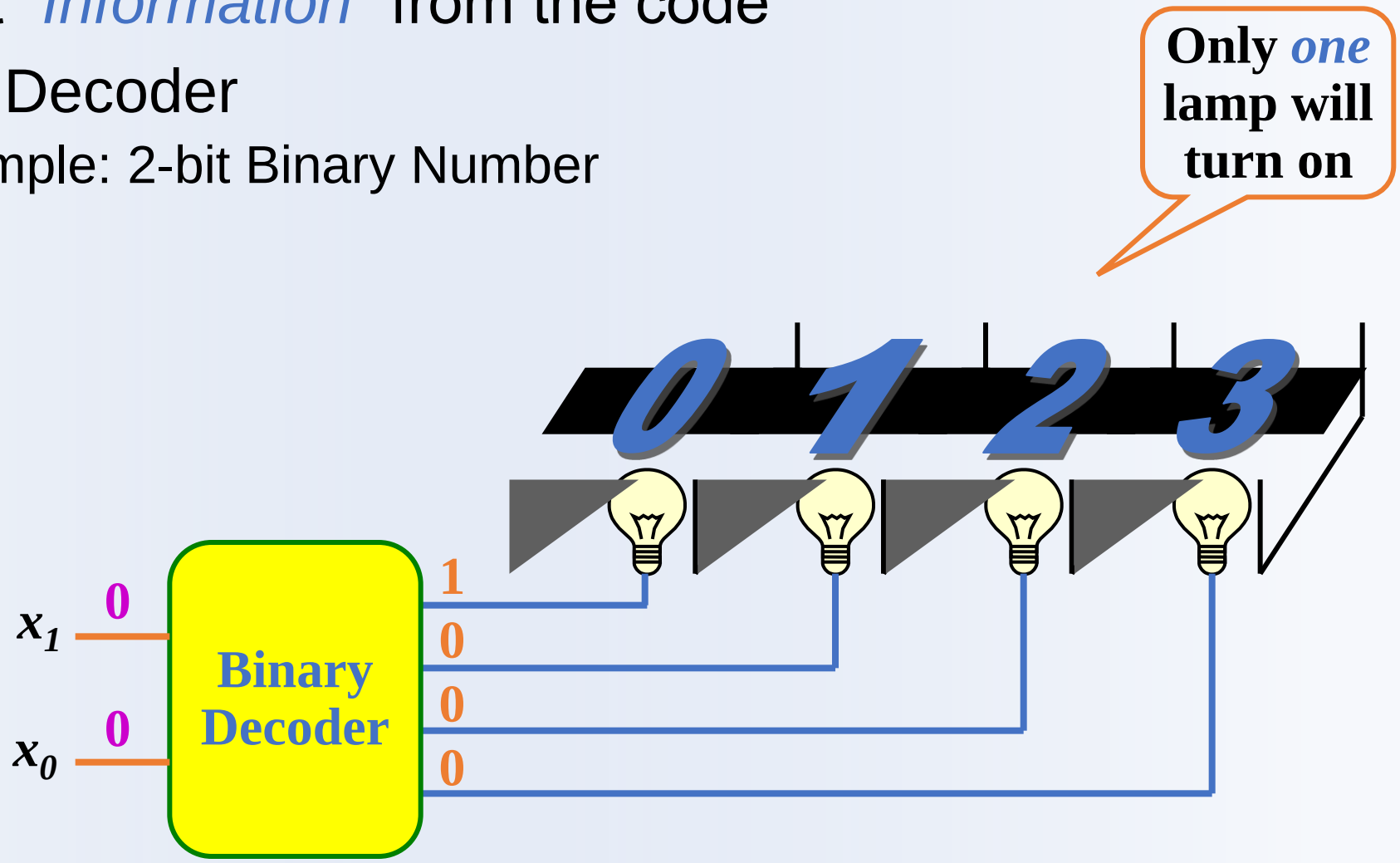
A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = \overline{D}$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	



Decoders

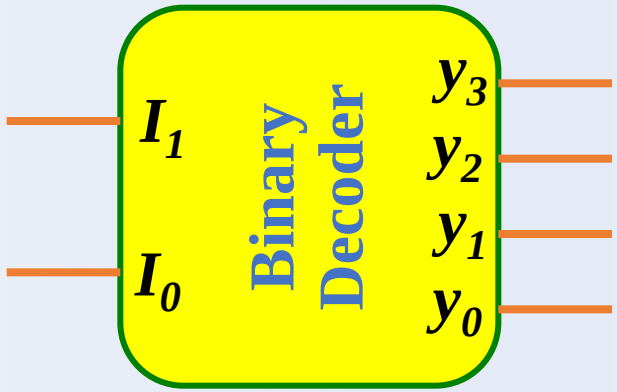
Decoders

- Extract "*Information*" from the code
- Binary Decoder
  - Example: 2-bit Binary Number

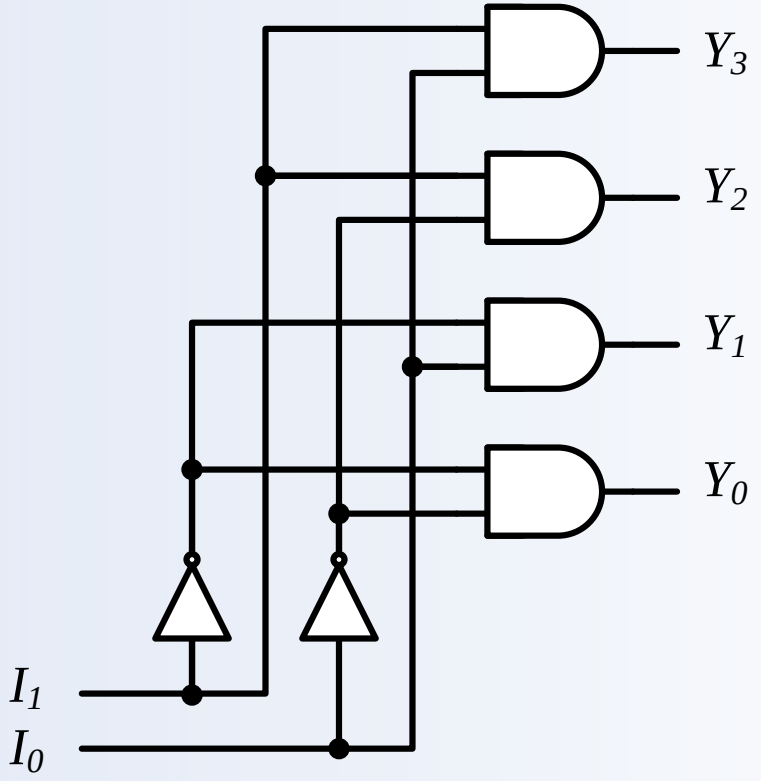


Decoders

- 2-to-4 Line Decoder



I_1	I_0	Y_3	Y_2	Y_1	Y_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

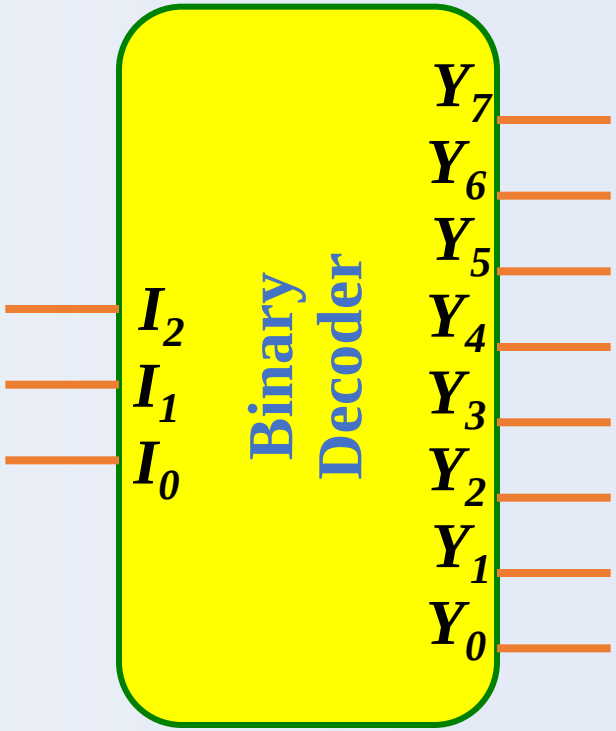


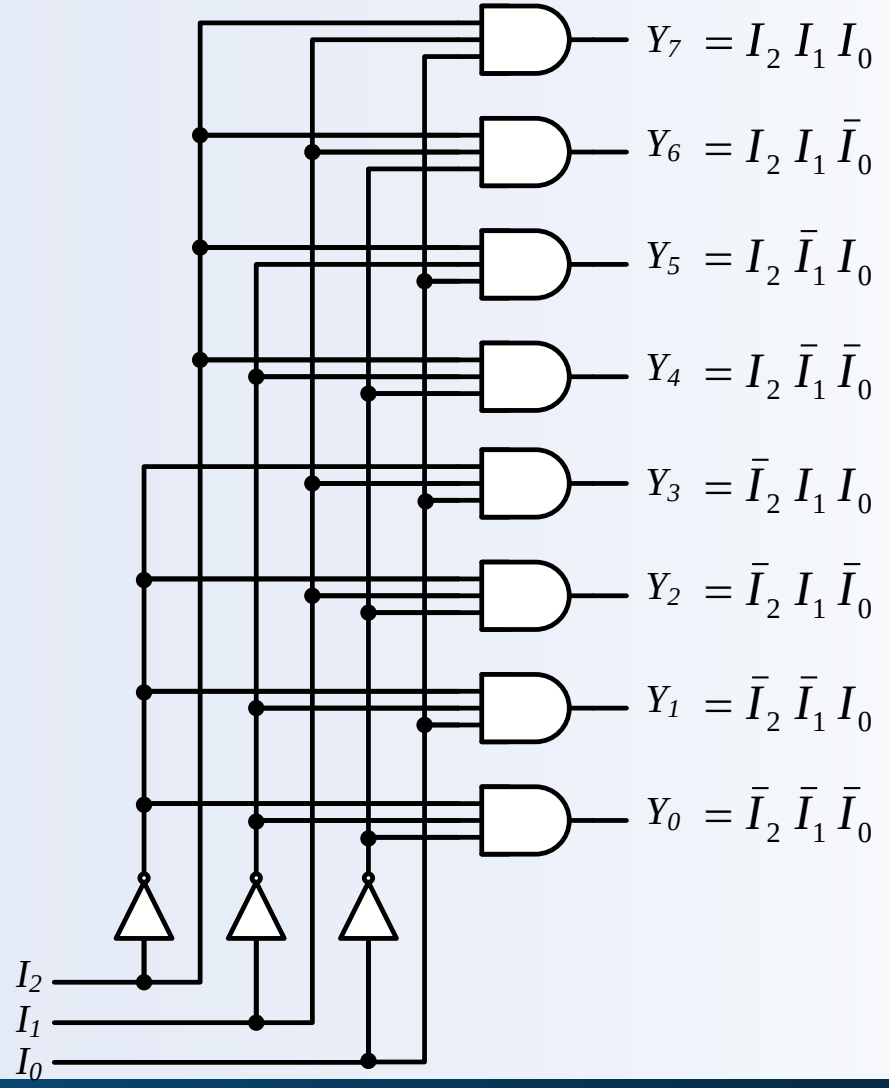
$$Y_3 = I_1 I_0 \qquad Y_2 = I_1 \bar{I}_0$$

$$Y_1 = \bar{I}_1 I_0 \qquad Y_0 = \bar{I}_1 \bar{I}_0$$

Decoders

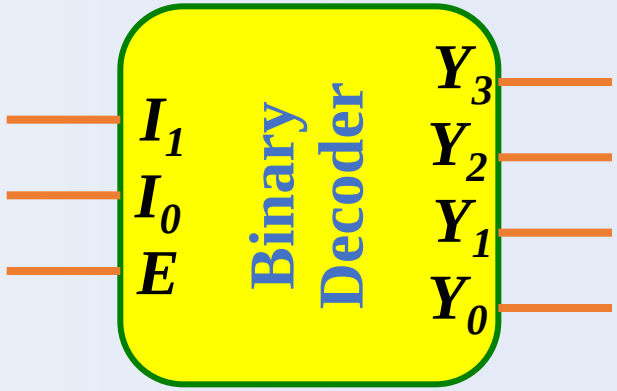
- 3-to-8 Line Decoder



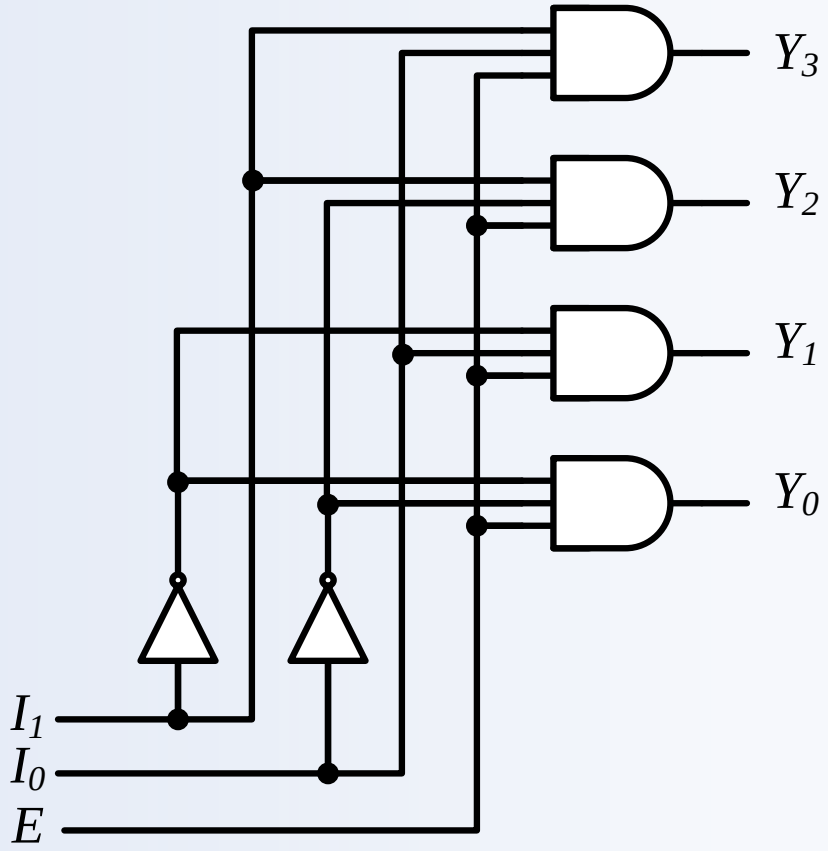


Decoders

- "*Enable*" Control



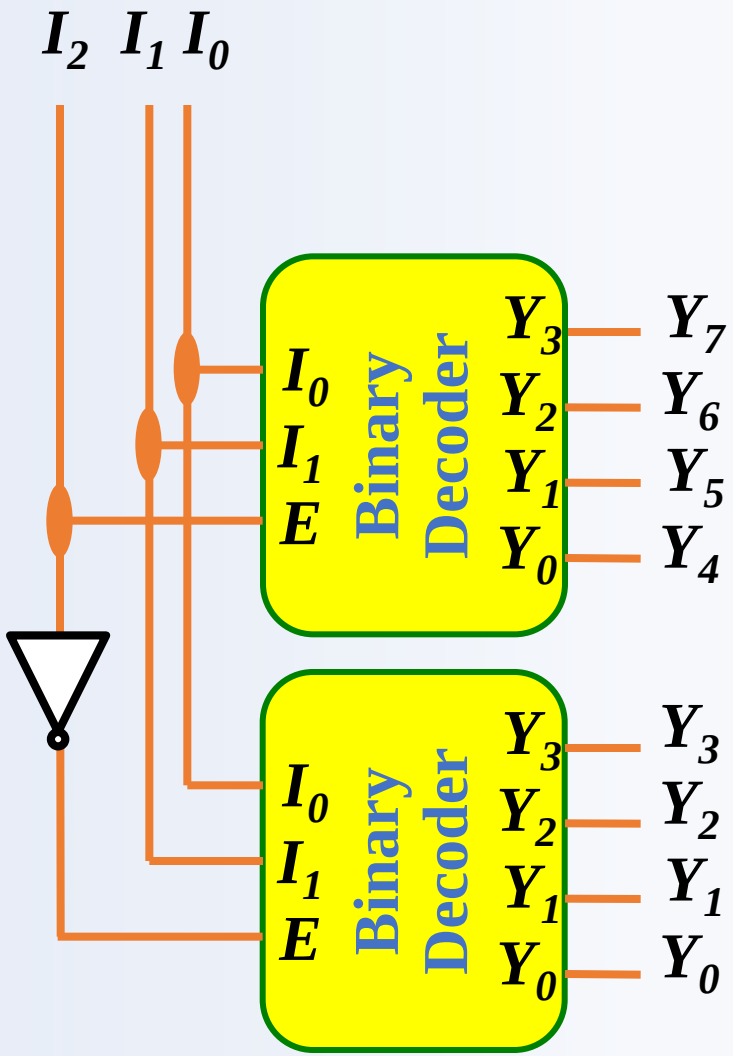
E	I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



Decoders

- Expansion

I_2	I_1	I_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

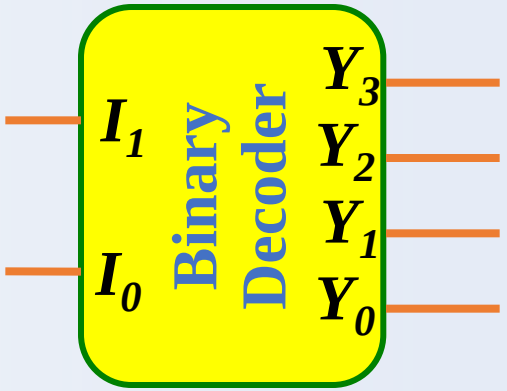


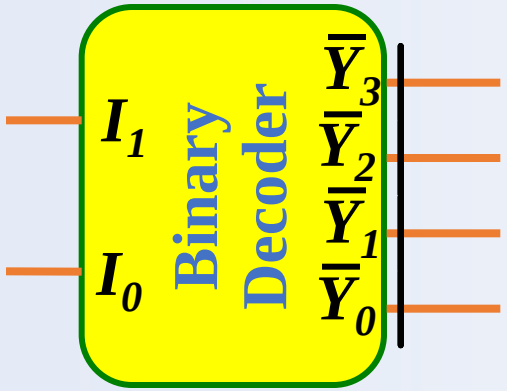
Decoders

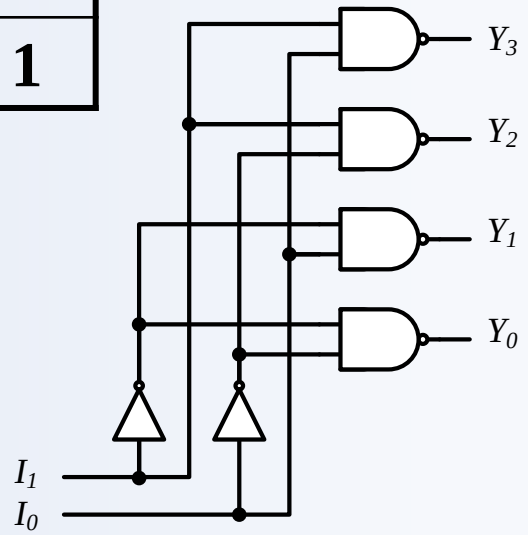
- Active-High / Active-Low

I_1	I_0	Y_3	Y_2	Y_1	Y_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

I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1







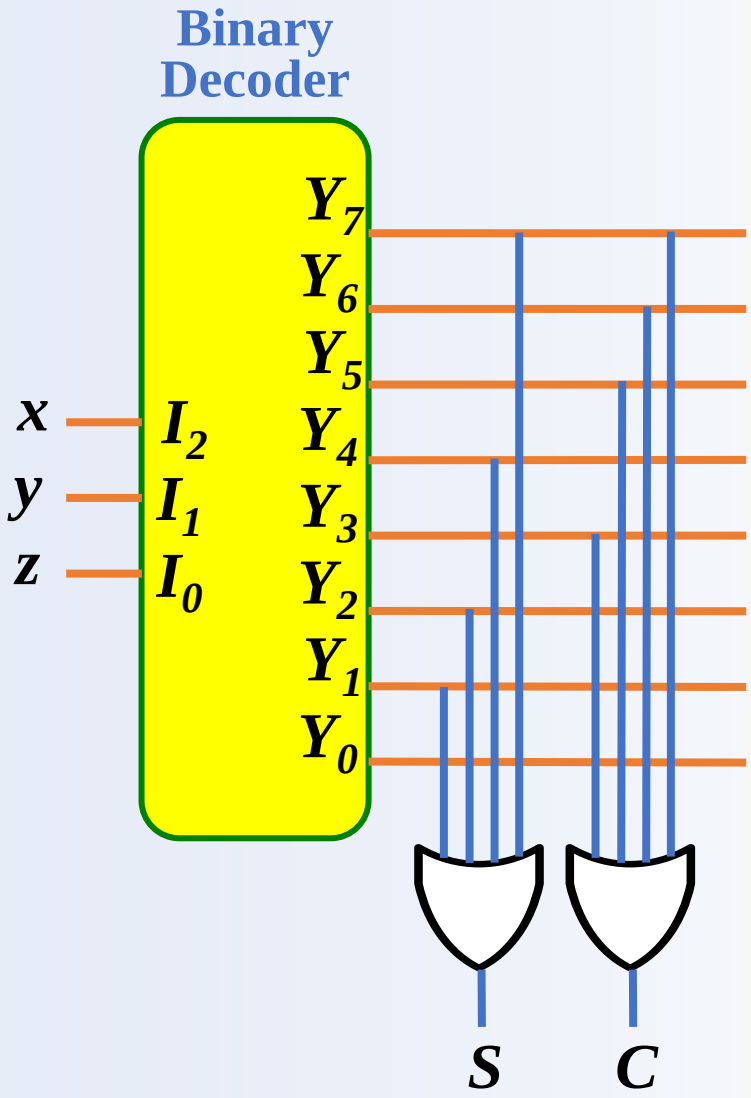
Implementation Using Decoders

- Each output is a minterm
- All minterms are produced
- Sum the required minterms

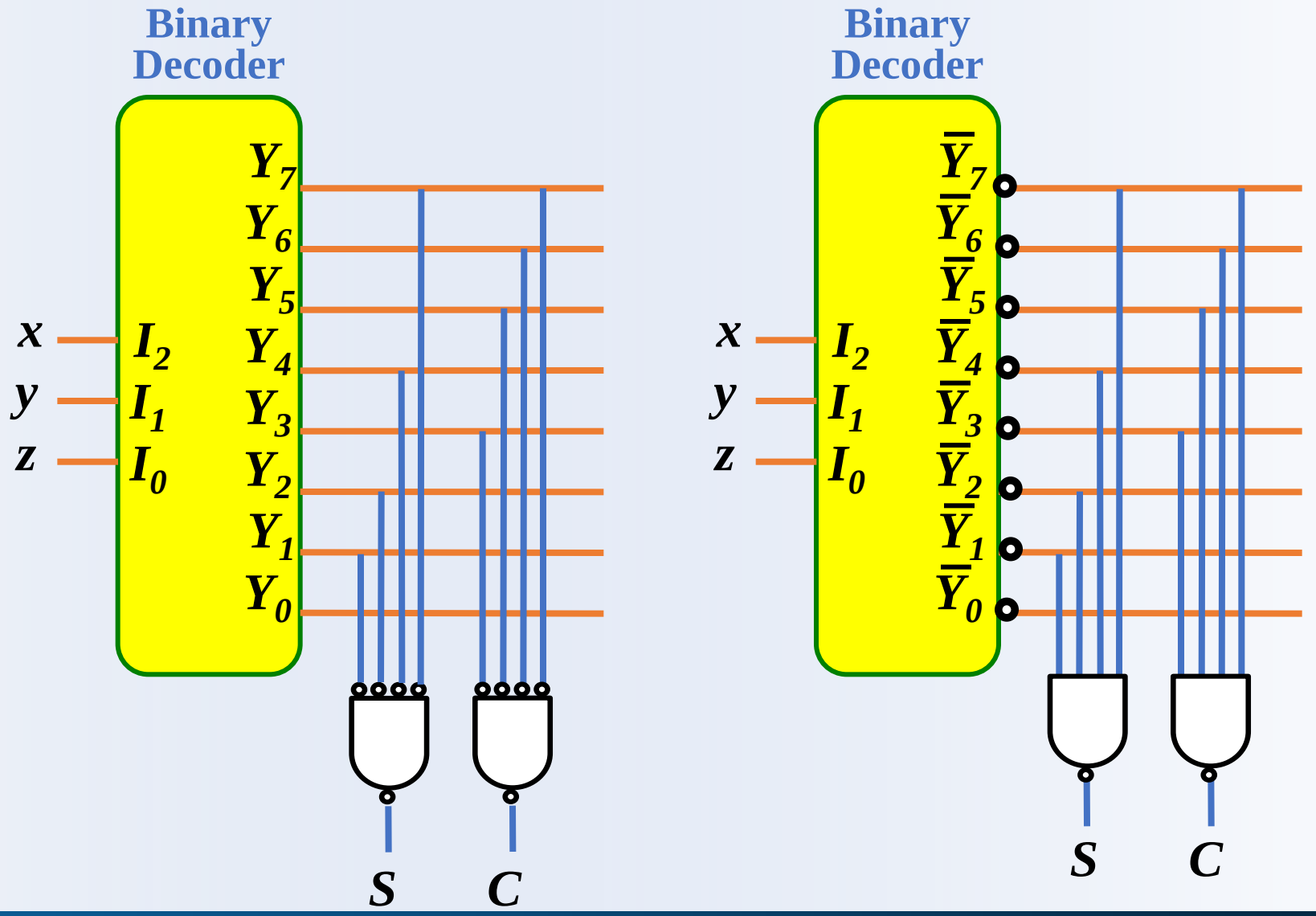
Example: Full Adder

$S(x, y, z) = \sum(1, 2, 4, 7)$

$C(x, y, z) = \sum(3, 5, 6, 7)$



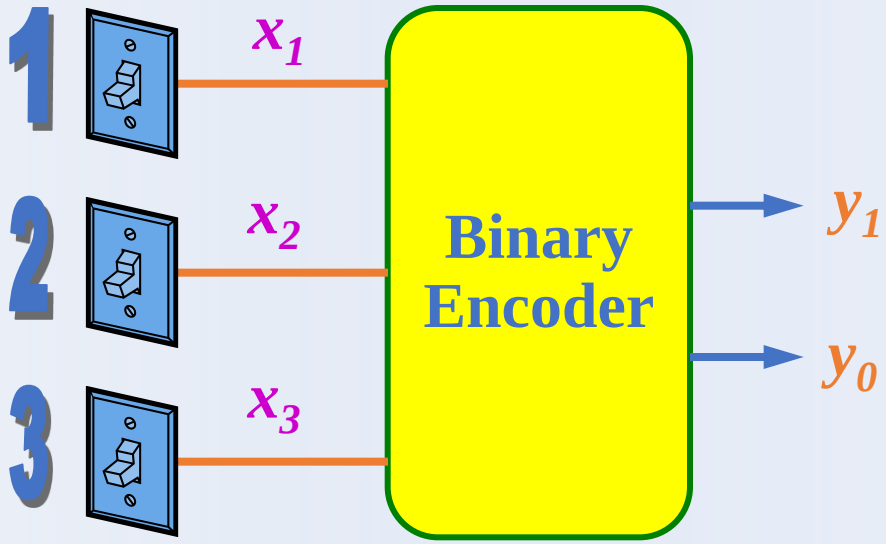
Implementation Using Decoders



Encoders

Encoders

- Put "*Information*" into code
- Binary Encoder
  - Example: 4-to-2 Binary Encoder



Only *one* switch should be activated at a time

x_3	x_2	x_1	y_1	y_0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
1	0	0	1	1

Encoders

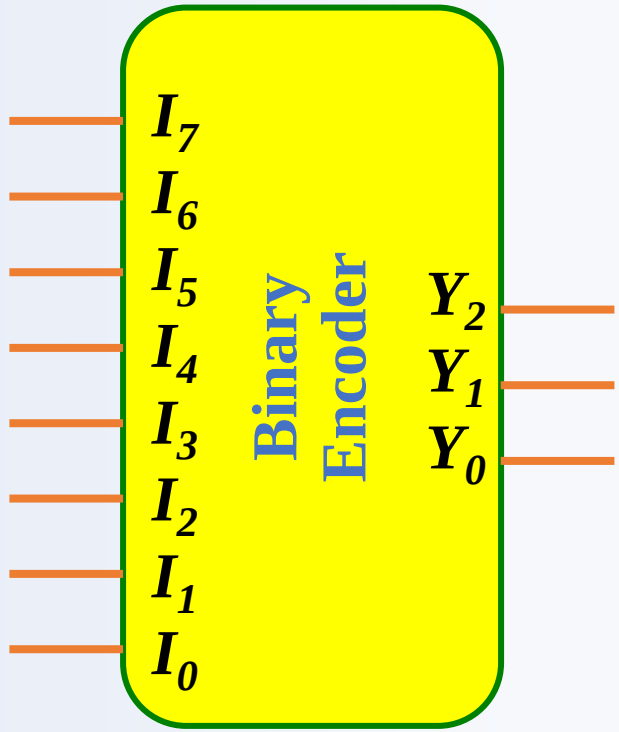
- Octal-to-Binary Encoder (8-to-3)

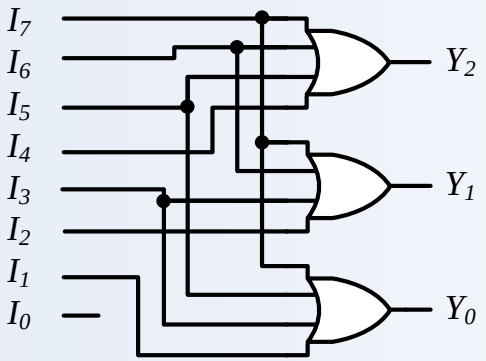
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$$Y_2 = I_7 + I_6 + I_5 + I_4$$

$$Y_1 = I_7 + I_6 + I_3 + I_2$$

$$Y_0 = I_7 + I_5 + I_3 + I_1$$

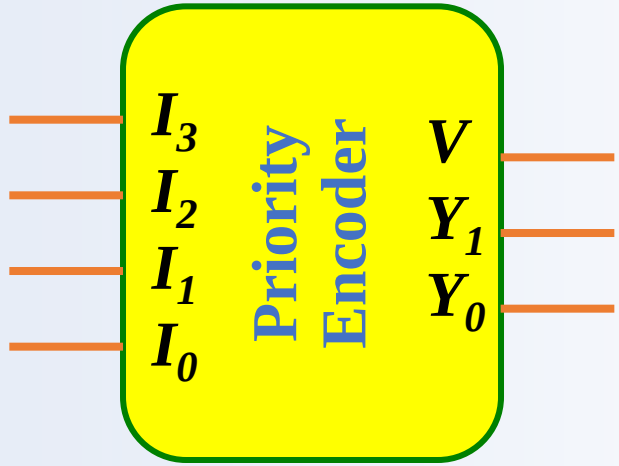


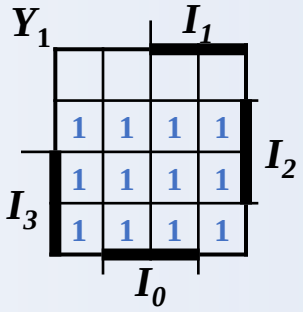


Priority Encoders

- 4-Input Priority Encoder

I_3	I_2	I_1	I_0	Y_1	Y_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

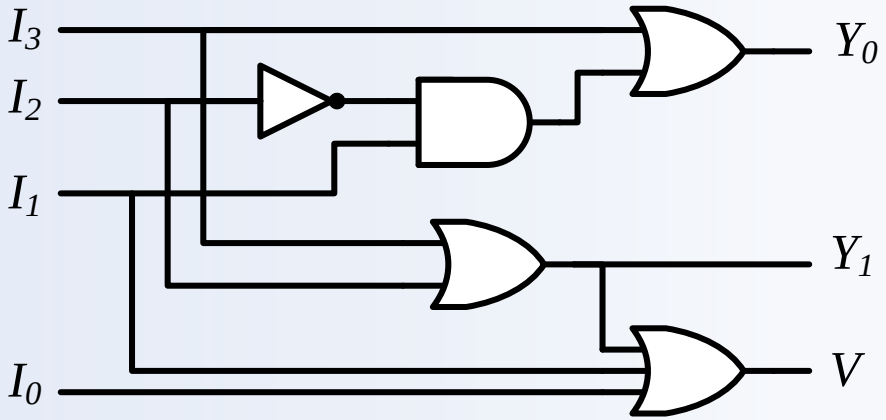




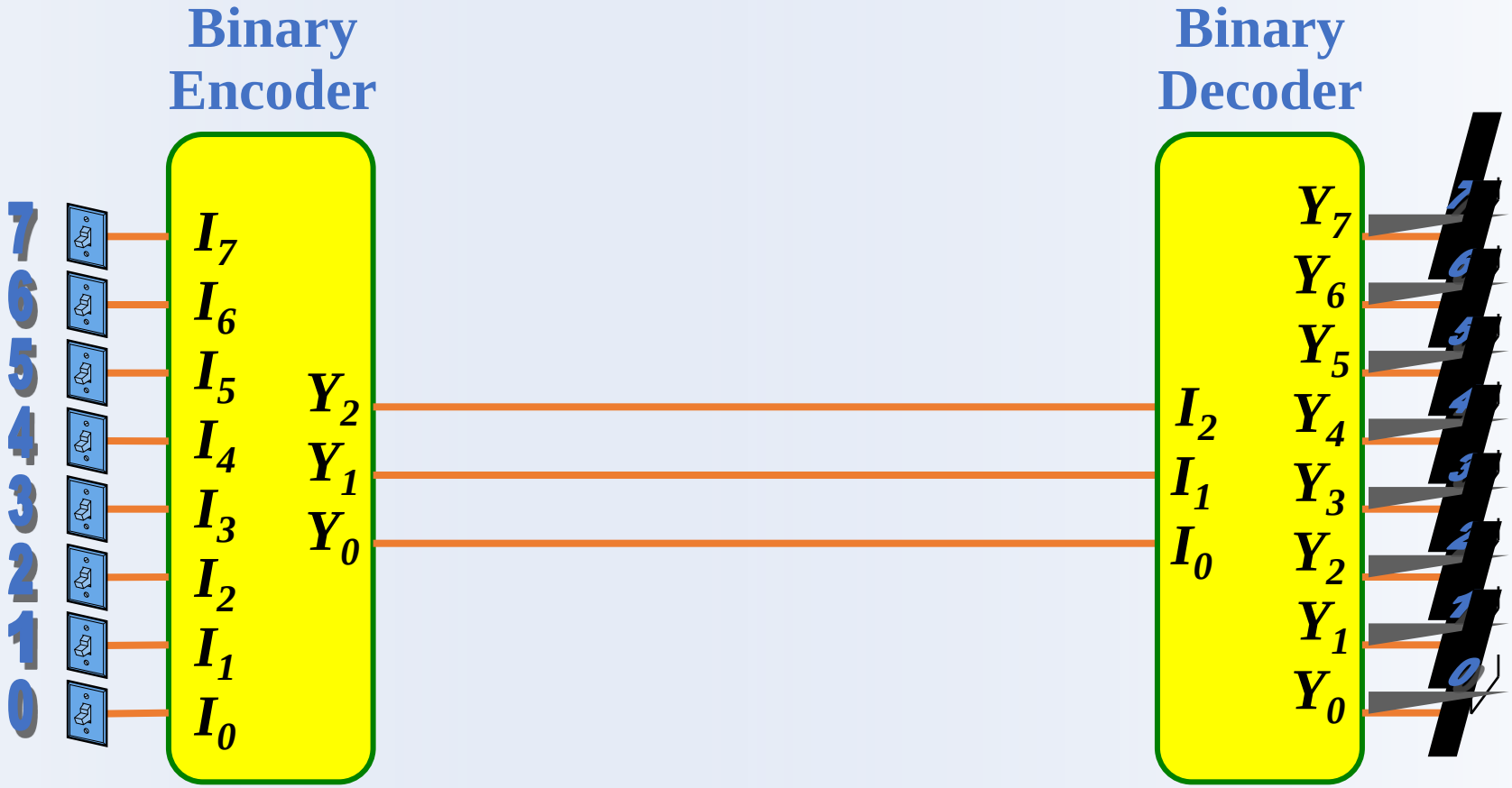
$$Y_1 = I_3 + I_2$$

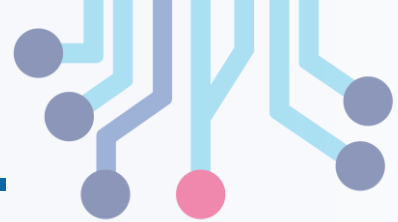
$$Y_0 = I_3 + \bar{I}_2 I_1$$

$$V = I_3 + I_2 + I_1 + I_0$$



Encoder / Decoder Pairs

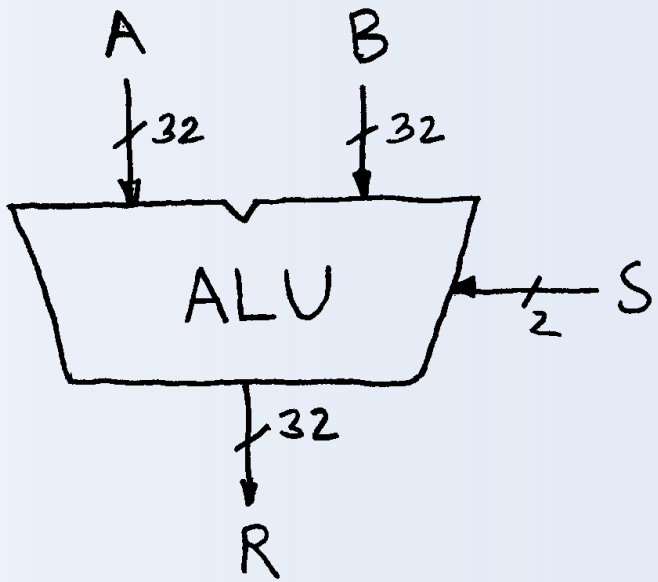




Arithmetic Logic Unit (ALU)

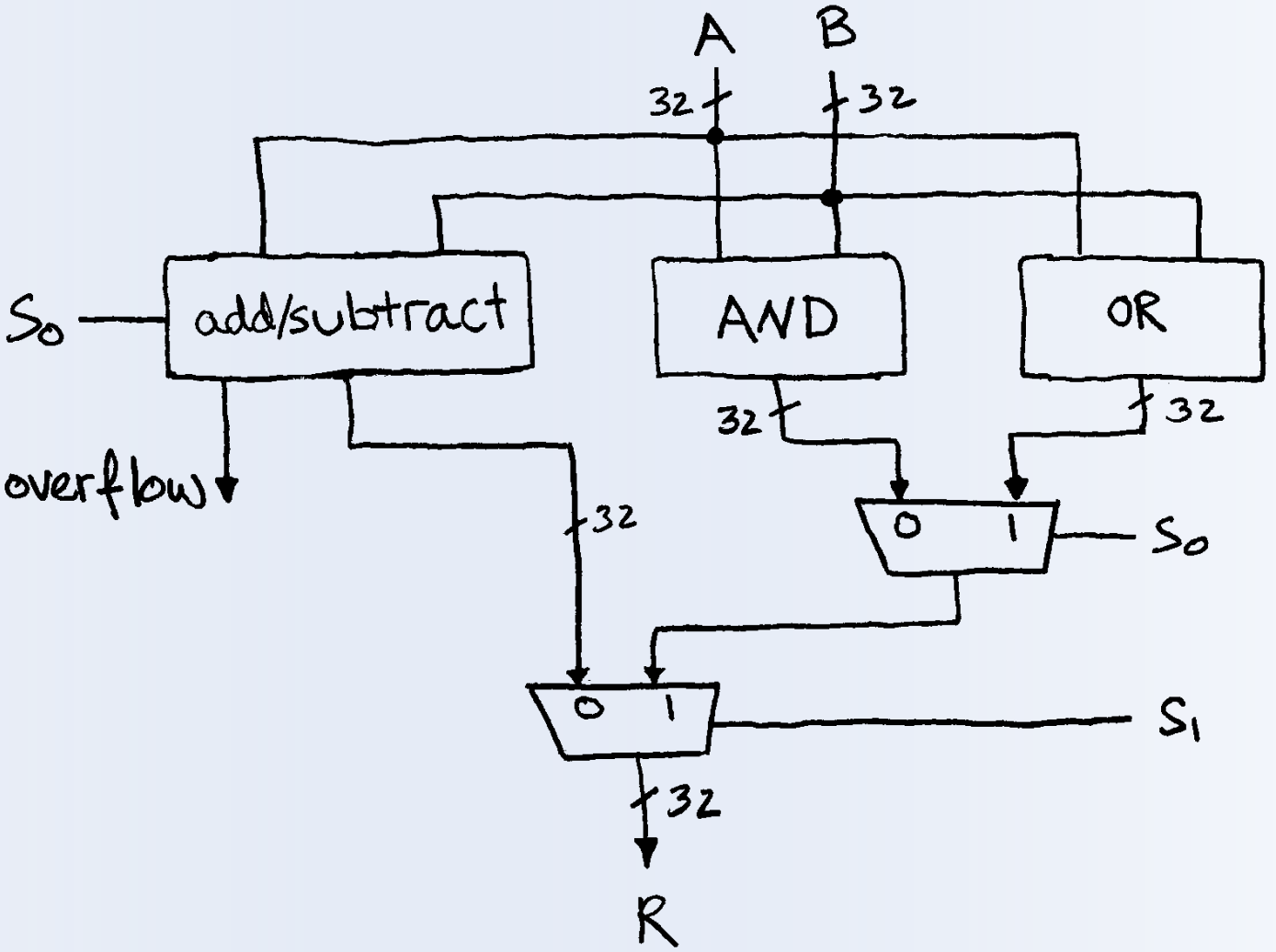
Arithmetic and Logic Unit

- Most processors contain a special logic block called “Arithmetic and Logic Unit” (ALU)
- We’ll show you an easy one that does ADD, SUB, bitwise AND (&), bitwise OR (|)



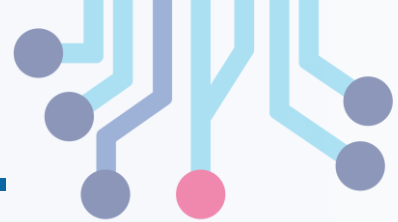
when `S=00`, `R=A+B`
when `S=01`, `R=A-B`
when `S=10`, `R=A&B`
when `S=11`, `R=A|B`

Our simple ALU



Barrel Shifter

Introduction to Barrel Shifter

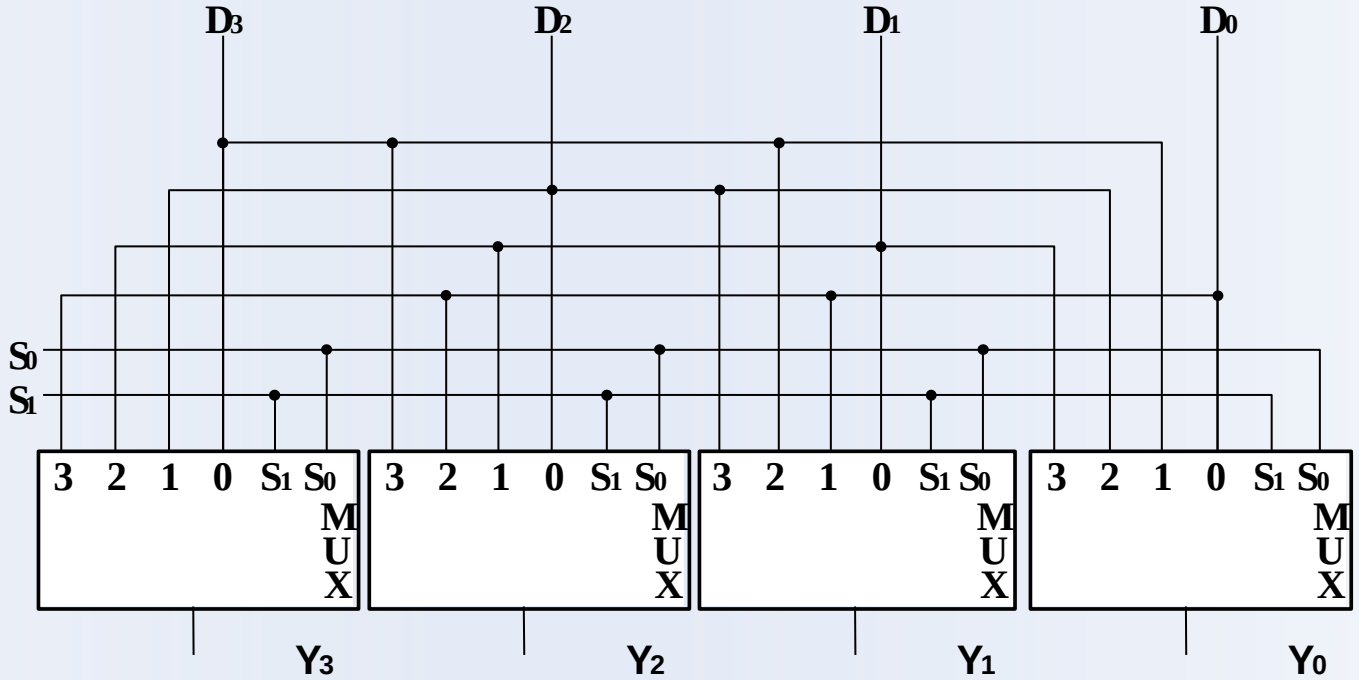


- Widely used Shifter architecture
- Exhibits good timing characteristic
- Area-efficient as well
- Inherent regularity in physical structure

Structure of Barrel Shifter

- Width of input and output data signals = n bits
- Width of input shift signal = $\lceil \log_2 n \rceil$ bits
- Shifter consists of $\lceil \log_2 n \rceil$ stages
- Each bit of the shift signal controls one stage
- Each stage handles single shift of 0 or 2^i bits

Barrel Shifter



- A rotate is a shift in which the bits shifted out are inserted into the positions vacated
- The circuit rotates its contents left from 0 to 3 positions depending on S:

S = 00 position unchanged
S = 01 rotate left by 1 positions
S = 10 rotate left by 2 positions
S = 11 rotate left by 3 positions

Barrel Shifter (continued)

- Large barrel shifters can be constructed by using:
  - Layers of multiplexers - Example 64-bit:
    - Layer 1 shifts by 0, 16, 32, 48
    - Layer 2 shifts by 0, 4, 8, 12
    - Layer 3 shifts by 0, 1, 2, 3
  - 2 - dimensional array circuits designed at the electronic level

Thank You

NCDC
NUST
CHIP
DESIGN
CENTRE