

# Topic 5

## Combination Circuit

# What is Combinational Circuit?

- **Combinational Circuit**
  - A digital circuit whose output depends only upon the **present combination** of its inputs
  - Output changes only when inputs change
  - Output changes once inputs change
- As oppose to **Sequential Circuit** which is
  - A digital circuit whose output depends not only upon the present input values, but also the history of input and output values
  - Have feedbacks from outputs
  - More complicated and difficult to analyze
- Typical modern digital circuit involves both combination and sequential circuits

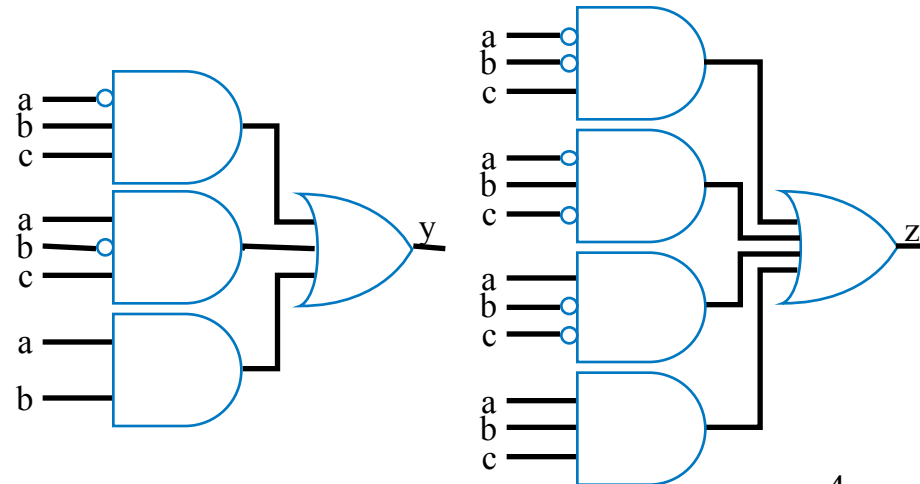
# How to Design Combinational Circuits?

Step	Description
Step 1 <b>Capture</b> the function	Create a truth table or equations, <i>whichever is most natural for the given problem</i> , to describe the desired behavior of the combinational logic.
Step 2 <b>Convert</b> to equations	This step is only necessary if you captured the function using a truth table instead of equations. Create an equation for each output by ORing all the minterms for that output. Simplify the equations if desired.
Step 3 <b>Implement</b> as a logic circuit	For each output, create a circuit corresponding to the output's equation. (Sharing gates among multiple outputs is OK optionally.)

# Example: Count Number of 1s

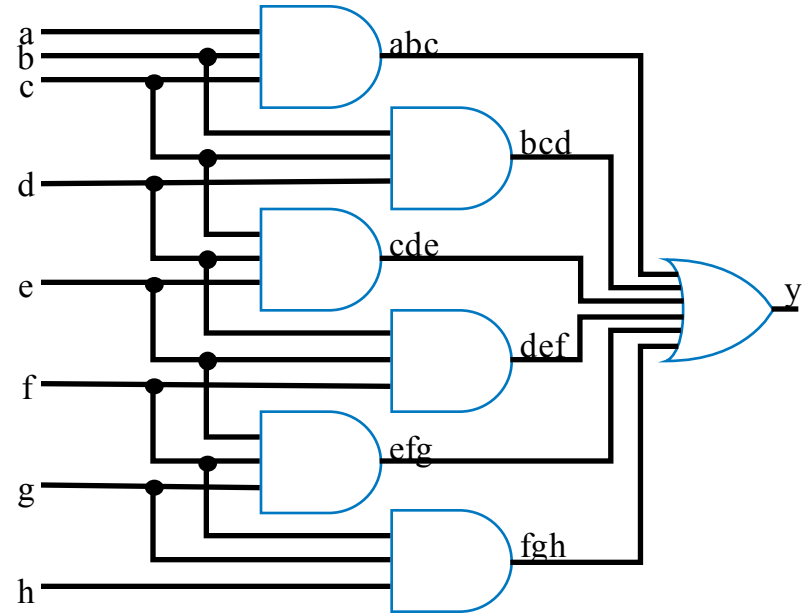
- Problem: Output in binary the number of 1s on three inputs
- E.g.: 010→01    101→10    000→00
  - **Step 1: Capture** the function
    - Truth table or equation?
      - Truth table is straightforward
  - **Step 2: Convert** to equation
    - $y = a'bc + ab'c + abc' + abc$
    - $z = a'b'c + a'bc' + ab'c' + abc$
  - **Step 3: Implement** as a gate-based circuit

Inputs			# of 1s	Outputs	
a	b	c		y	z
0	0	0	(0)	0	0
0	0	1	(1)	0	1
0	1	0	(1)	0	1
0	1	1	(2)	1	0
1	0	0	(1)	0	1
1	0	1	(2)	1	0
1	1	0	(2)	1	0
1	1	1	(3)	1	1



# Example: Three 1s Detector

- Problem: Detect three consecutive 1s in 8-bit input: abcdefgh
  - 000**1**1101  $\rightarrow$  1      10101011  $\rightarrow$  0
  - 11110000  $\rightarrow$  1
  - **Step 1: Capture** the function
    - Truth table or equation?
      - Truth table too big:  $2^8=256$  rows
      - Equation: create terms for each possible case of three consecutive 1s
    - $y = abc + bcd + cde + def + efg + fgh$
  - **Step 2: Convert** to equation -- already done
  - **Step 3: Implement** as a gate-based circuit



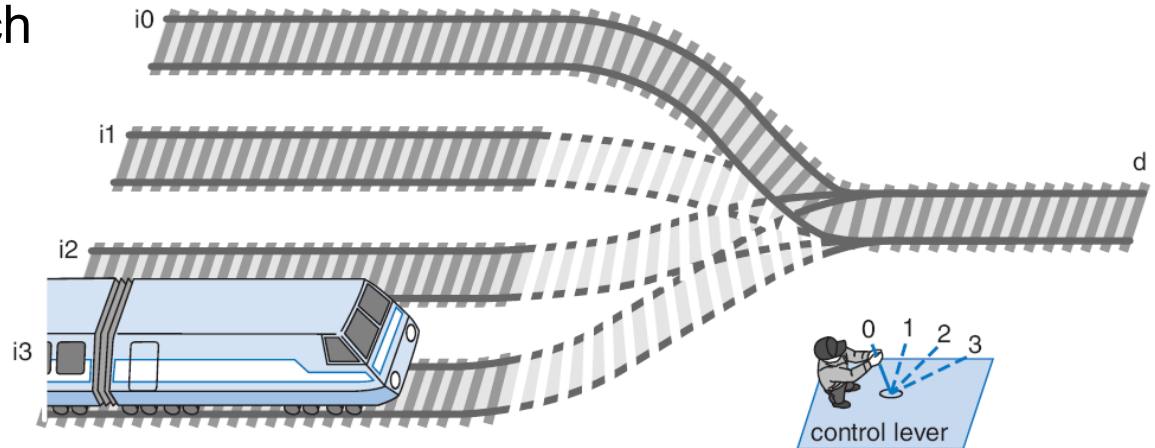
# Combinational Building Blocks

- Combinational building blocks are digital components that
  - Are combinational circuits
  - Implement specific fundamental functions
  - Are used to build bigger combinational circuits
- To learn
  - Multiplexor (MUX)
  - Half adder
  - Full adder
  - Carry-ripple adder
  - Decoder
  - Buffer
  - Tri-state buffer

# Multiplexor (Mux)

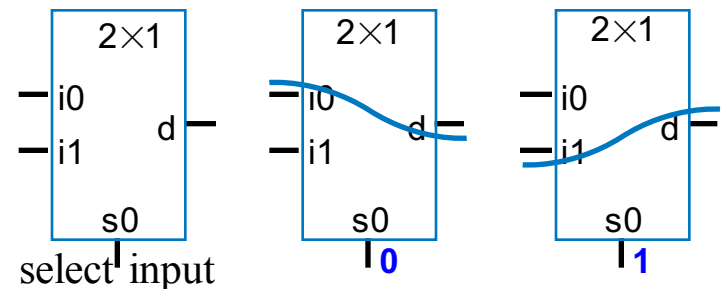
- Mux: A popular combinational building block

- Like a railyard switch



- Routes one of its N data inputs to its one output, based on binary value of select inputs

- 2 inputs  $\rightarrow$  needs 1 select bit ( 2 different combinations) to select which input to connect to the output
- 4 inputs  $\rightarrow$  2 select bits
- 8 inputs  $\rightarrow$  3 select bits
- N inputs  $\rightarrow \log_2(N)$  select bits



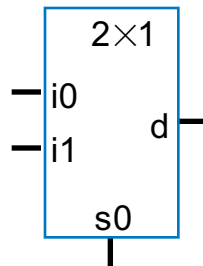
*2 to 1* or *2 by 1* or *2x1* mux

# Mux Internal Design

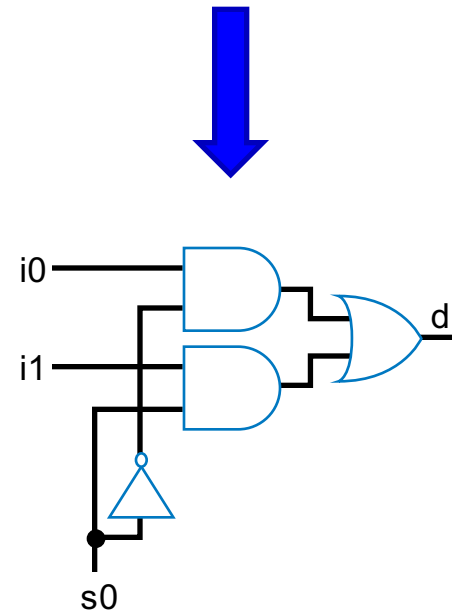
Connect i0 to d  
when s0 is 0,  
otherwise  
connect i1 to d

s0	i0	i1	d
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$\begin{aligned}d &= s0' i0 i1' + s0' i0 i1 + s0 i0' i1 + s0 i0 i1 \\ &= s0' i0 + s0 i1\end{aligned}$$

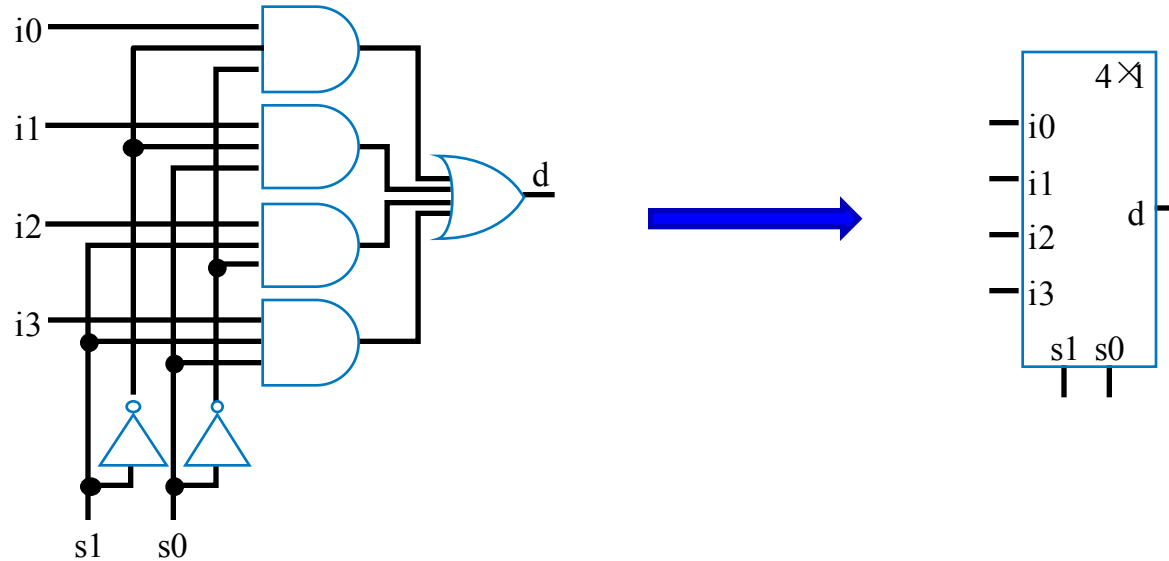


**2x1 mux**



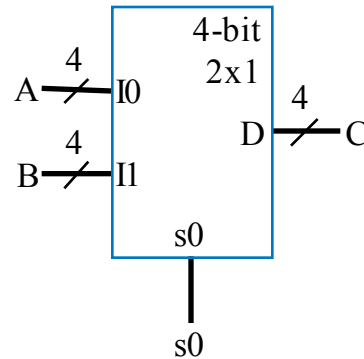
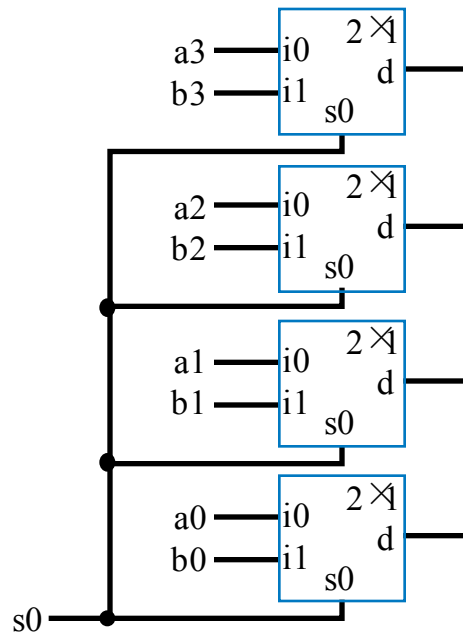


# Mux Internal Design

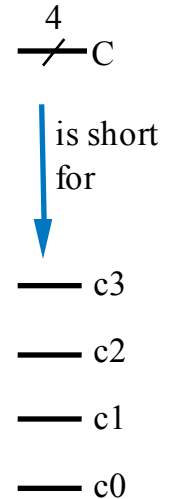


**4x1 mux**

# 4-bit 2x1 Mux

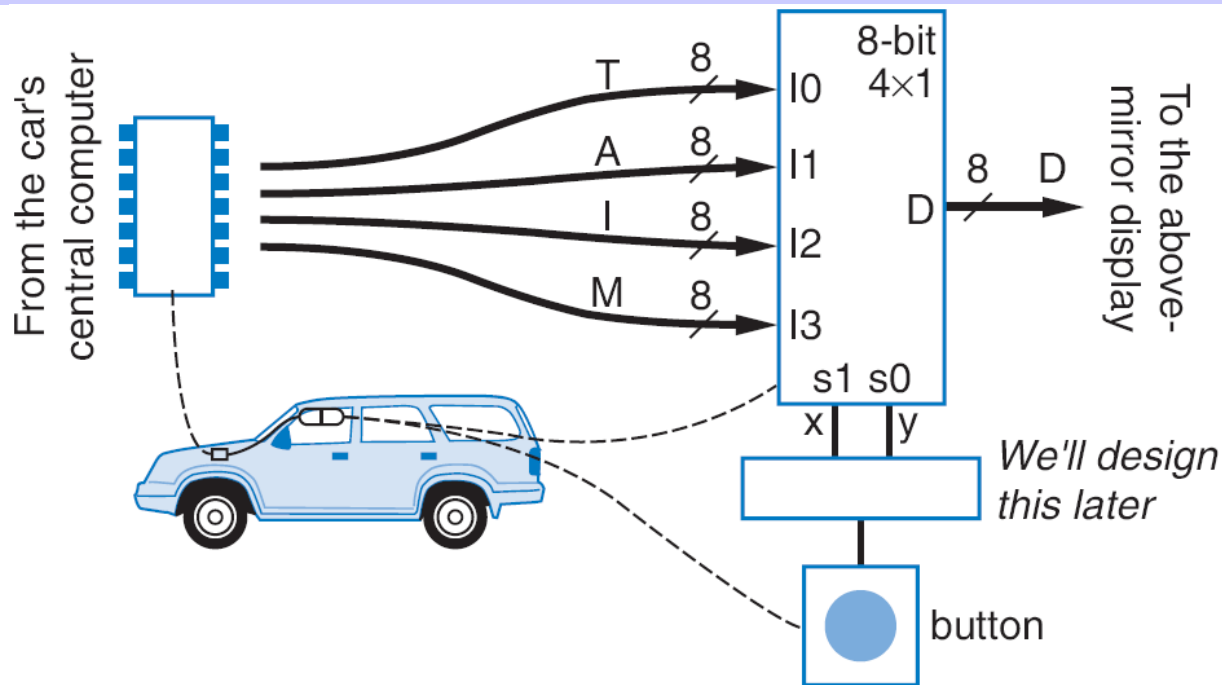


Simplifying notation:



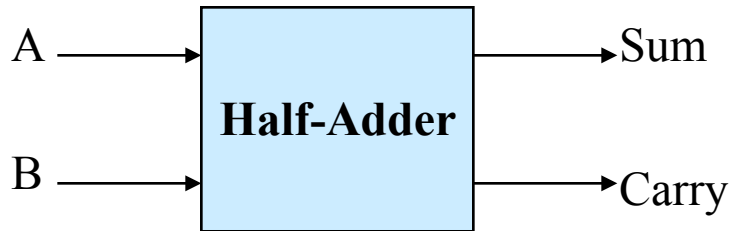
- Ex: Two 4-bit inputs, A (a3 a2 a1 a0), and B (b3 b2 b1 b0)
  - 4-bit 2x1 mux (just four 2x1 muxes sharing a select line) can select between A or B
- Width of input channels may be any, 8, 18, 32, 64, ...

# N-bit Mux Example



- Four possible display items
  - Temperature (T), Average miles-per-gallon (A), Instantaneous mpg (I), and Miles remaining (M) -- each is 8-bits wide
  - Choose which to display using two inputs x and y
  - Use 8-bit 4x1 mux

# Half Adder



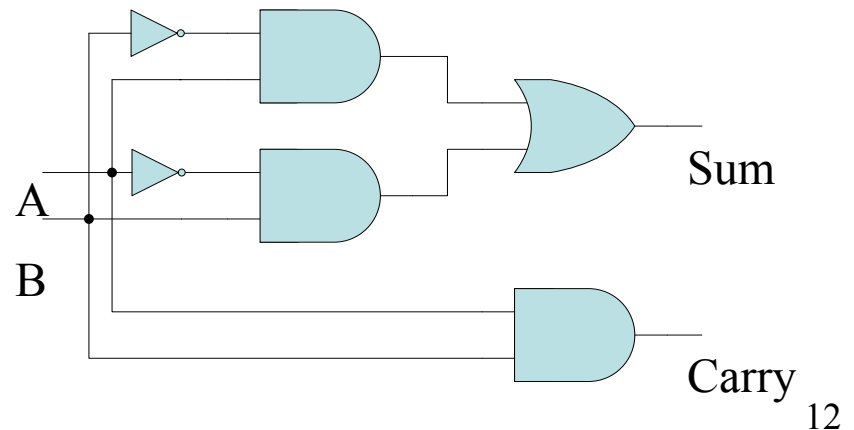
- Derive Boolean functions (sum-of-minterms) from the truth table for both outputs

$$\begin{aligned}\text{Sum} &= A'B + AB' = m_1 + m_2 = \Sigma (1, 2) \\ &= (A+B)(A'+B') = M_0 \cdot M_3 = \Pi (0, 3)\end{aligned}$$

- Addition of two single bits A, B
- Based on the operations it performs, a truth table can be built

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$\begin{aligned}\text{Carry} &= AB = m_3 \\ &= (A+B)(A+B')(A'+B) \\ &= M_0 \cdot M_1 \cdot M_2 \\ &= \Pi (0, 1, 2)\end{aligned}$$

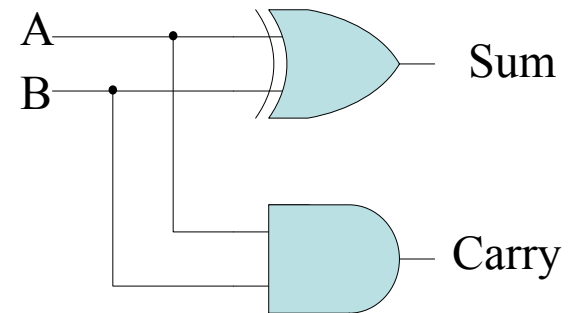
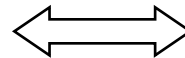
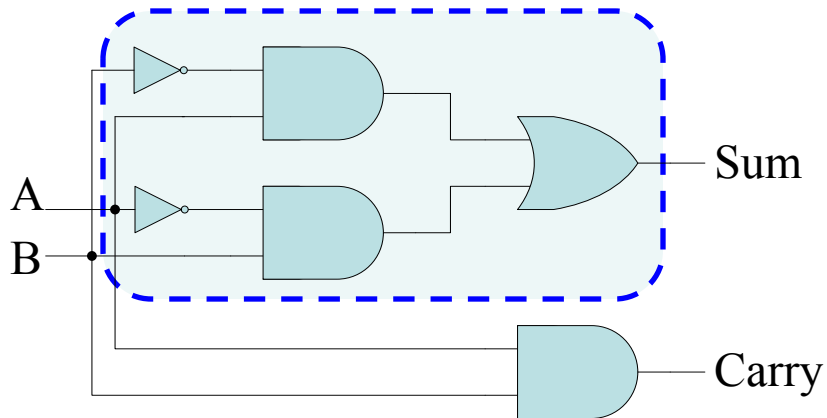


# Application of XOR in Half-Adder

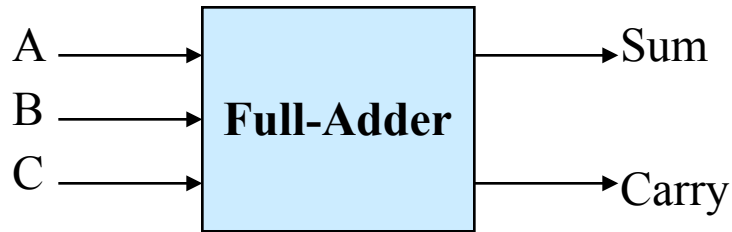
- Half adder

$$\text{Sum} = A'B + AB' = A \oplus B$$

$$\text{Carry} = AB$$



# Full Adder



- Derive minterm/Maxterm expressions from the truth table for both outputs

Sum = ?

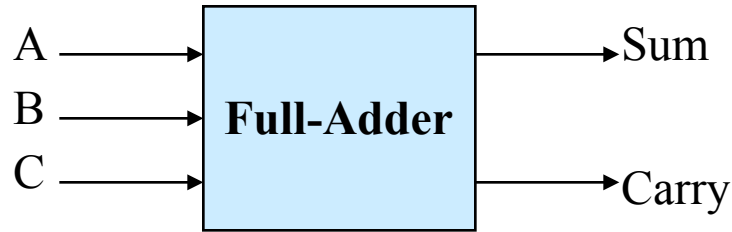
Carry = ?

- Based on the operations it performs, a truth table can be built

A	B	C	Sum	Carry
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1	1	1

Logic Circuit?

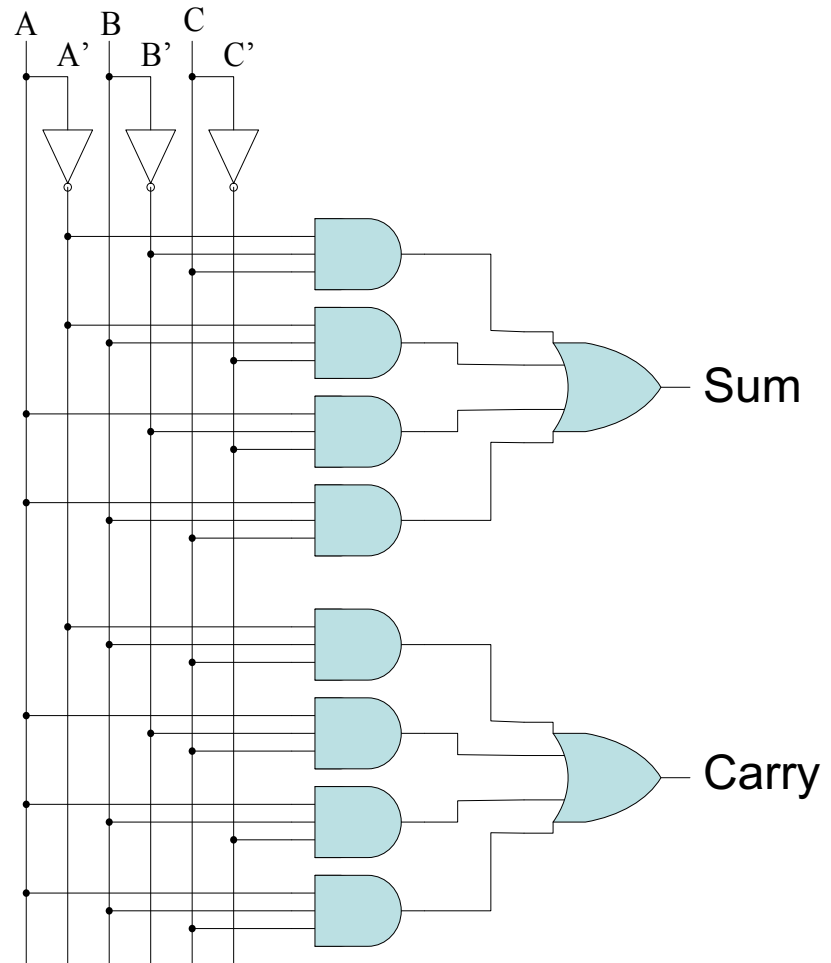
# Full Adder



A	B	C	Sum	Carry
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

$$\begin{aligned}\text{Sum} &= A'B'C + A'BC' + AB'C' + ABC \\ &= \Sigma m(1, 2, 4, 7)\end{aligned}$$

$$\begin{aligned}\text{Carry} &= A'BC + AB'C + ABC' + ABC \\ &= \Sigma m(3, 5, 6, 7)\end{aligned}$$



Notice the circuit draw convention

# Application of XOR in Full Adder

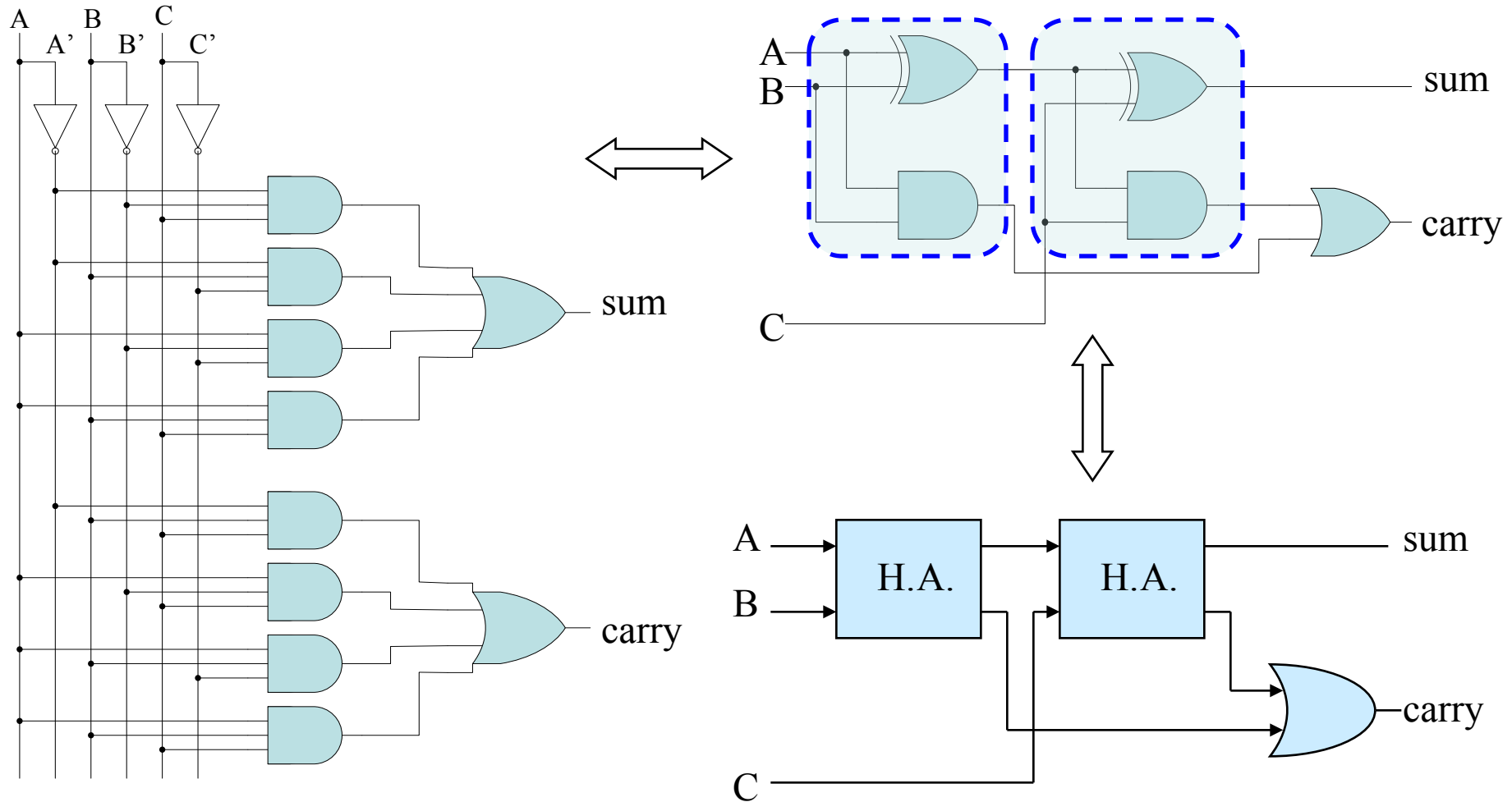
- Full adder

$$\begin{aligned}\text{Sum} &= A'B'C + A'BC' + AB'C' + ABC \\ &= A'(B'C + BC') + A(B'C' + BC) \\ &= A'(B \oplus C) + A(B \oplus C)' && (\text{let } B \oplus C = D) \\ &= A'D + AD' \\ &= A \oplus D \\ &= A \oplus B \oplus C\end{aligned}$$

$$\begin{aligned}\text{Carry} &= A'BC + AB'C + ABC' + ABC \\ &= (A'B + AB')C + AB(C' + C) \\ &= (A \oplus B)C + AB\end{aligned}$$



# Application of XOR in Full Adder



# Multi-bit Number Addition

$$\begin{array}{r}
 \text{Carry: } 0\ 1\ 0\ 0\ \mathbf{0} \\
 \text{A: } \quad 0\ 1\ 0\ 0 = 4 \\
 +\ \text{B: } \quad 0\ 1\ 0\ 1 = 5 \\
 \hline
 \text{Sum: } \quad 1\ 0\ 0\ 1 = 9
 \end{array}$$

$$\begin{array}{r}
 \text{Carry: } 0\ 0\ 0\ 0\ \mathbf{0} \\
 \quad \quad 0\ 1\ 1\ 0 = +6 \\
 + \quad \quad 0\ 0\ 0\ 1 = +1 \\
 \hline
 \text{Sum: } \quad 0\ 1\ 1\ 1 = +7
 \end{array}$$

A carry bit is the carry output (C\_out) from previous stage, it's also the carry input (C\_in) to the present stage

$$\begin{array}{r}
 \text{Carry: } 1\ 1\ \mathbf{0}\ \mathbf{0}\ \mathbf{0} \\
 \text{A: } \quad 1\ 1\ 0\ 0 = 12 \\
 +\ \text{B: } \quad 1\ 1\ 0\ 1 = 13 \\
 \hline
 \text{Sum: } \mathbf{1}\ 1\ 0\ 0\ 1 = 25
 \end{array}$$

Without previous stage, carry should be always 0

Need one more bit to hold a number bigger than  $2^n - 1 = 15$

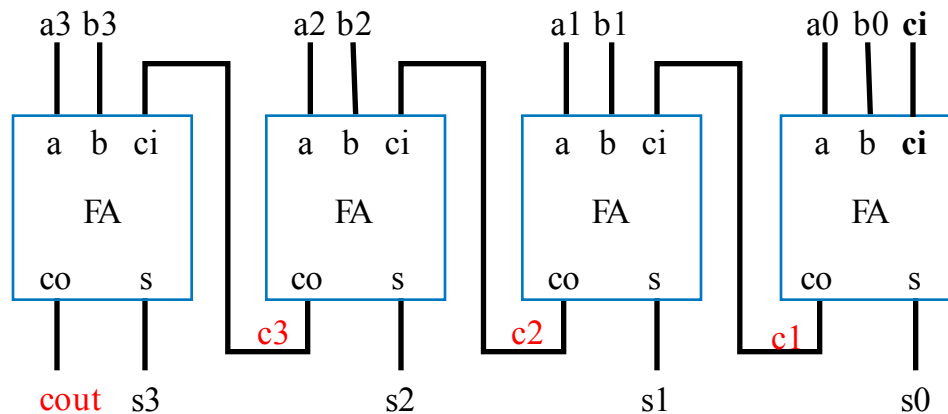
Operation may be performed by a full adder

# Carry-Ripple Adder

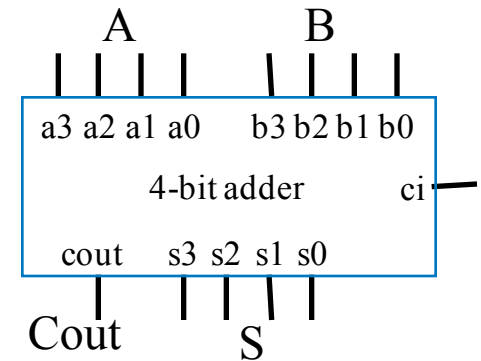
- carry-ripple adder**

- 4-bit adder: Adds two 4-bit numbers, generates 5-bit output
  - 5-bit output can be considered 4-bit “sum” plus 1-bit “carry out”
- Can easily build any size adder

carries:	<b>c3</b>	<b>c2</b>	<b>c1</b>	cin
B:	b3	b2	b1	b0
A:	+ a3	a2	a1	a0
	<hr/>			
<b>cout</b>	s3	s2	s1	s0



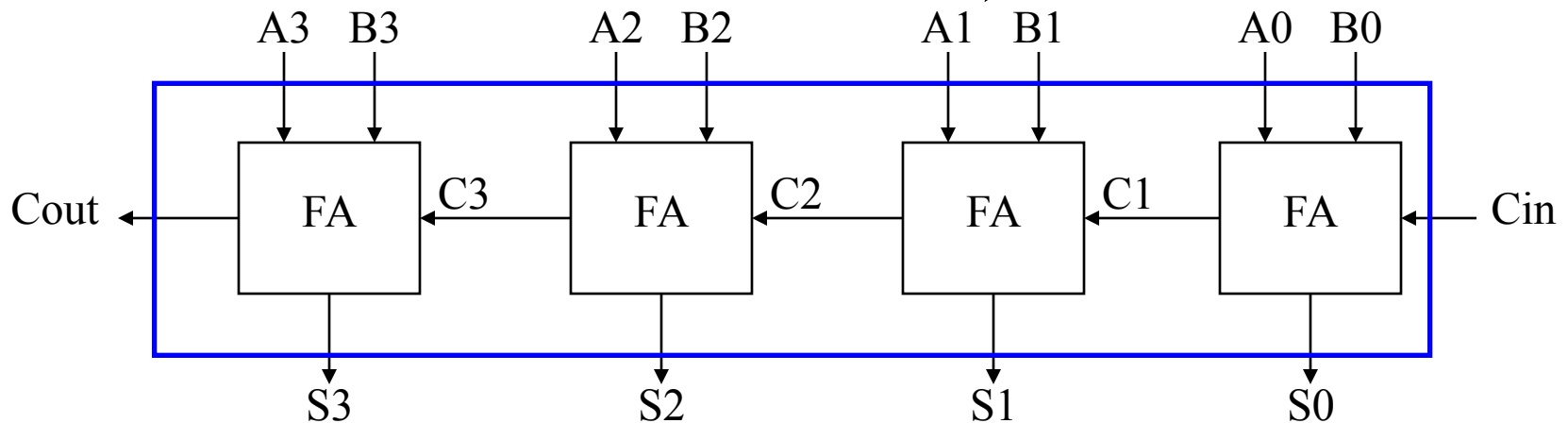
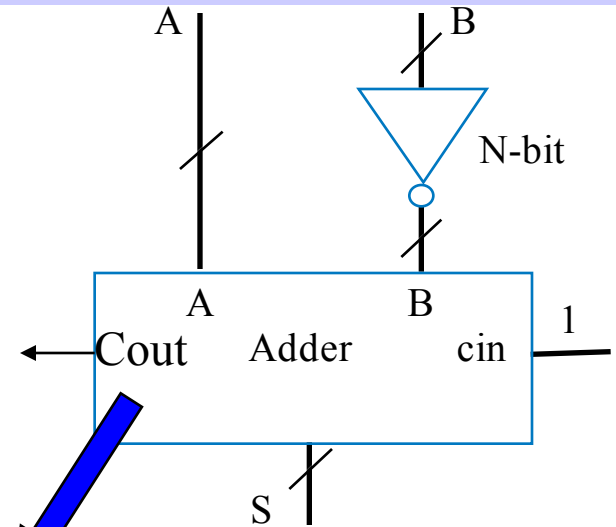
(a)



(b)

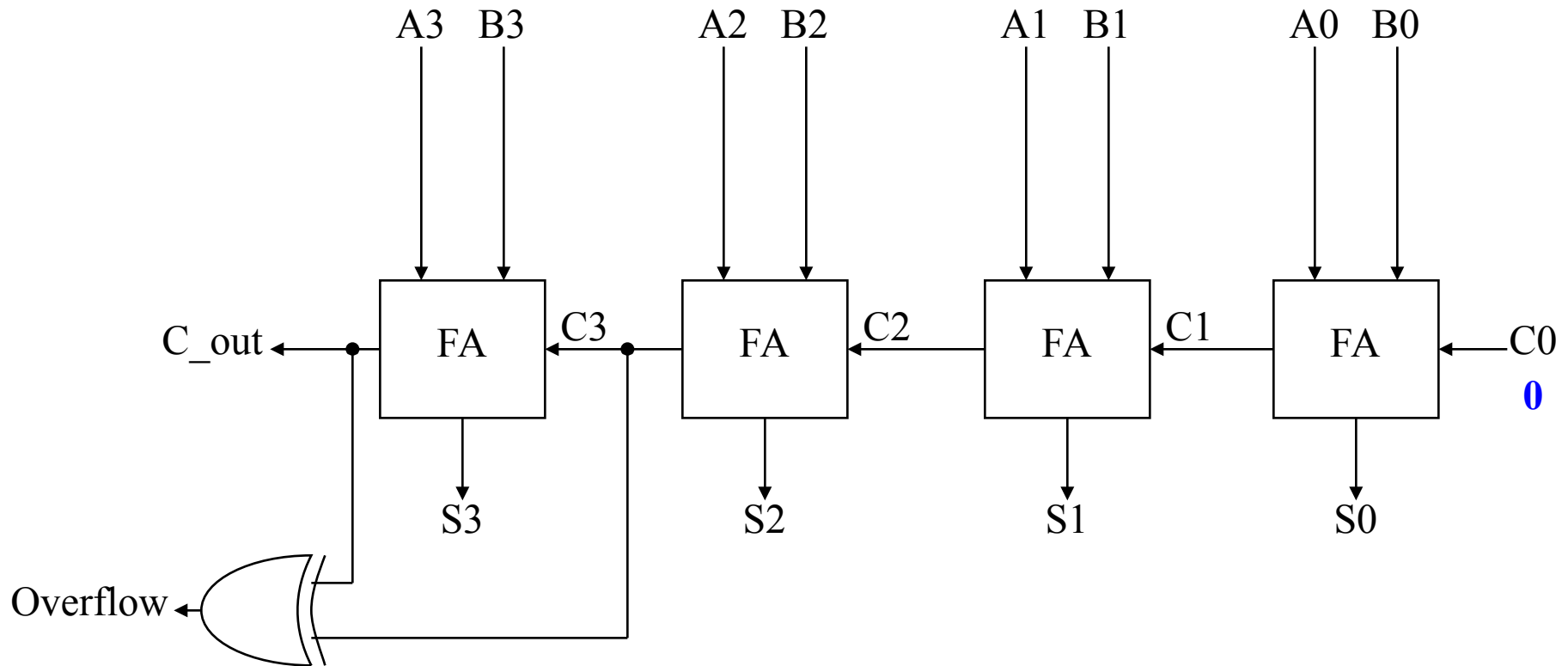
# Two's Complement Subtractor

- Using two's complement representation
$$A - B = A + (-B)$$
$$= A + (\text{two's complement of } B)$$
$$= A + \text{invert\_bits}(B) + 1$$
- So build subtractor using adder by inverting B's bits, and setting carry in to 1

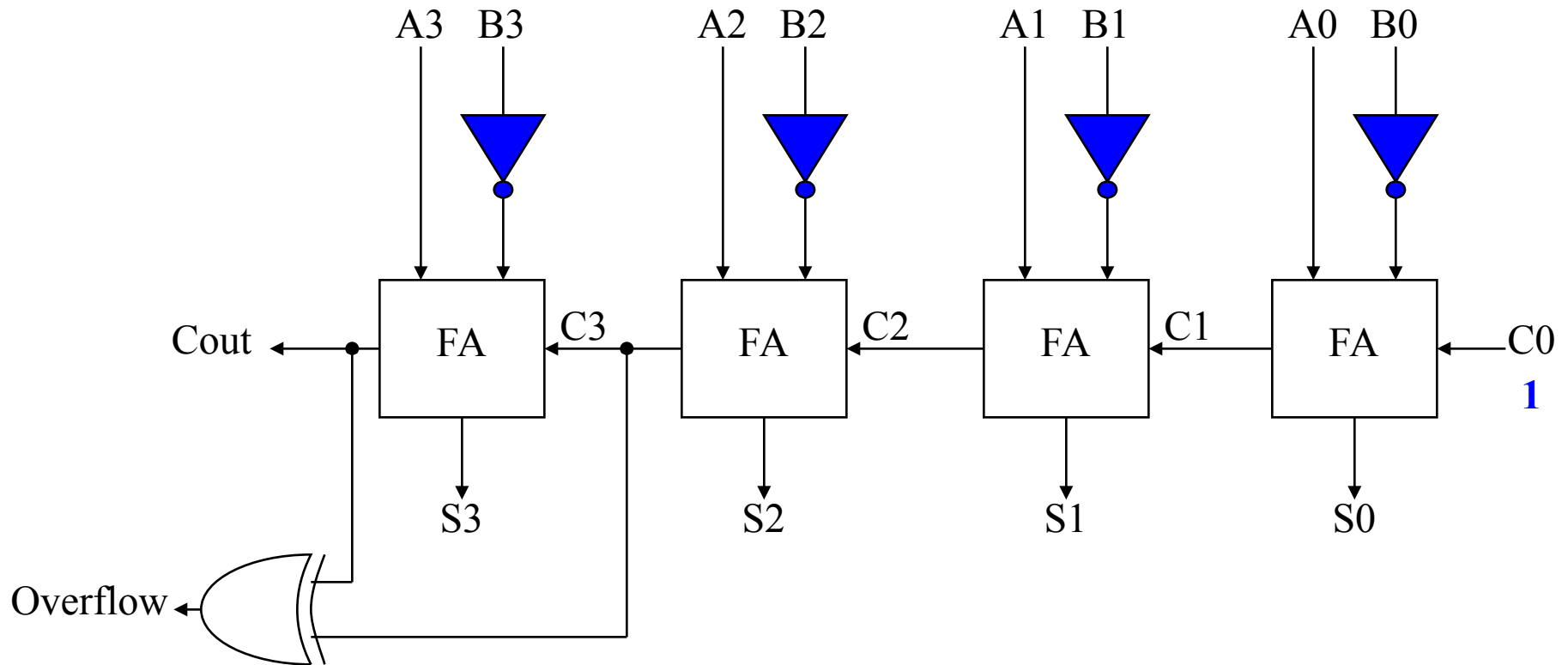


Or  
Lookahead Adder

# 4-Bit 2's Complement Adder



# 4-Bit 2's Complement Subtractor



# Arithmetic-Logic Unit: ALU

- **ALU**: Component that can perform any of various arithmetic (add, subtract, increment, etc.) and logic (AND, OR, etc.) operations, based on control inputs
- Key component in computer

TABLE 4.2 Desired calculator operations

Inputs			Operation	Sample output if A=00001111, B=00000101
x	y	z		
0	0	0	S = A + B	S=00010100
0	0	1	S = A - B	S=00001010
0	1	0	S = A + 1	S=00010000
0	1	1	S = A	S=00001111
1	0	0	S = A AND B (bitwise AND)	S=00000101
1	0	1	S = A OR B (bitwise OR)	S=00001111
1	1	0	S = A XOR B (bitwise XOR)	S=00001010
1	1	1	S = NOT A (bitwise complement)	S=11110000

# Encoder

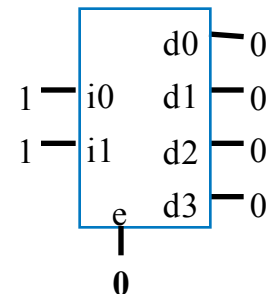
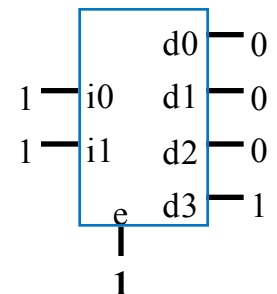
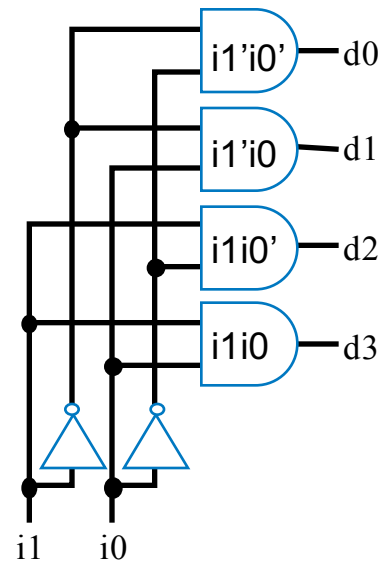
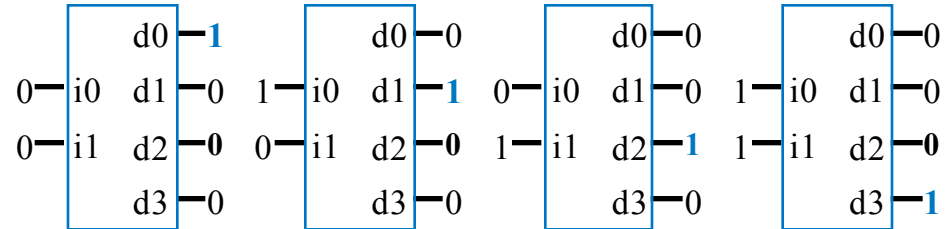
- Each input is given a binary code

Inputs								Outputs		
D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1



# Decoder

- **Decoder:** Popular combinational logic building block, in addition to logic gates
  - Converts input binary number to one high output
- 2-input decoder: four possible input binary numbers
  - So has four outputs, one for each possible input binary number
- Internal design
  - AND gate for each output to detect input combination
- Decoder with enable  $e$ 
  - Outputs all 0 if  $e=0$
  - Regular behavior if  $e=1$
- $n$ -input decoder:  $2^n$  outputs

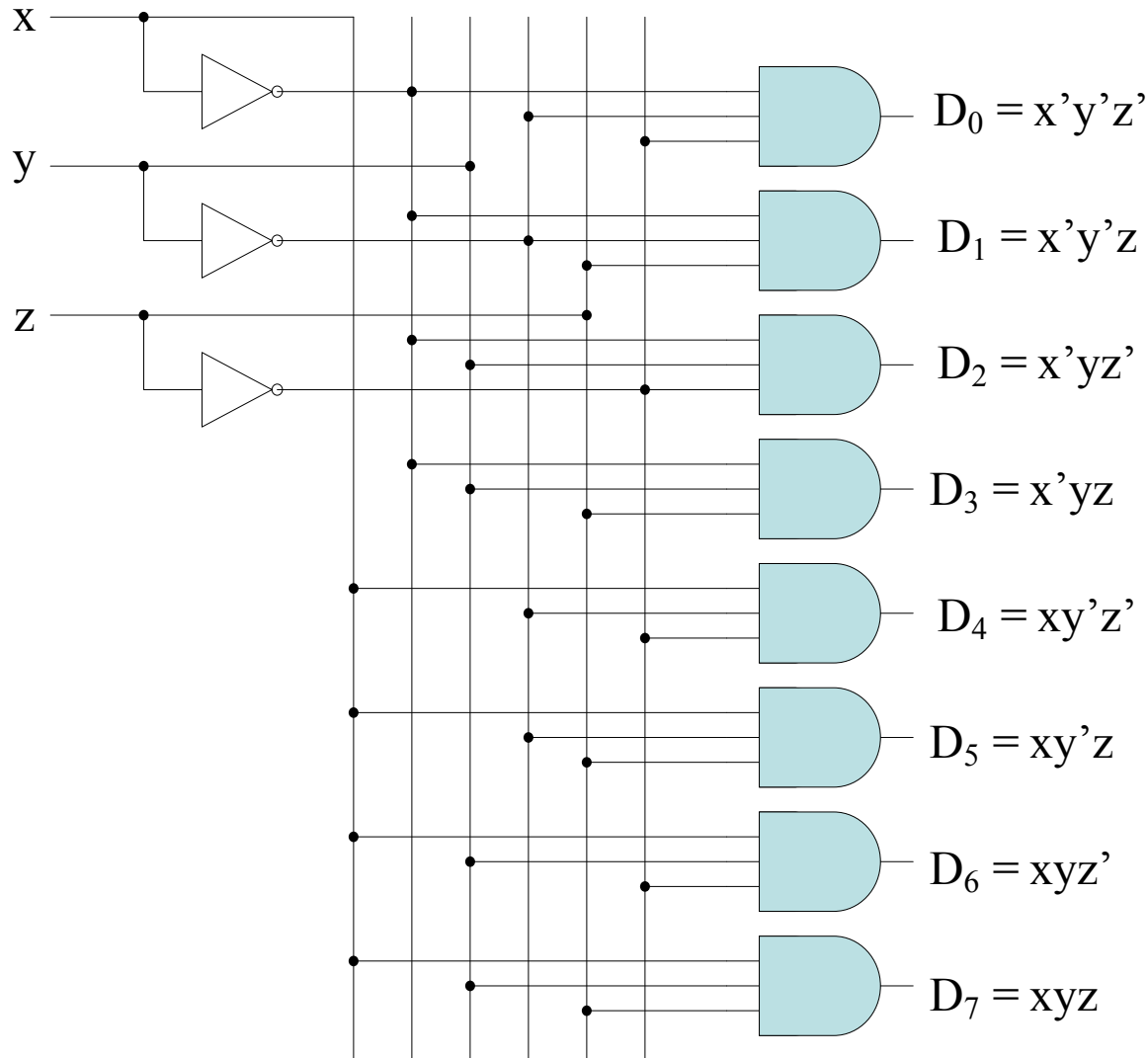


# Truth Table of $3 \times 8$ Decoder

- For any input combination, only one of the outputs is turned on

Inputs			Outputs							
x	y	z	D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

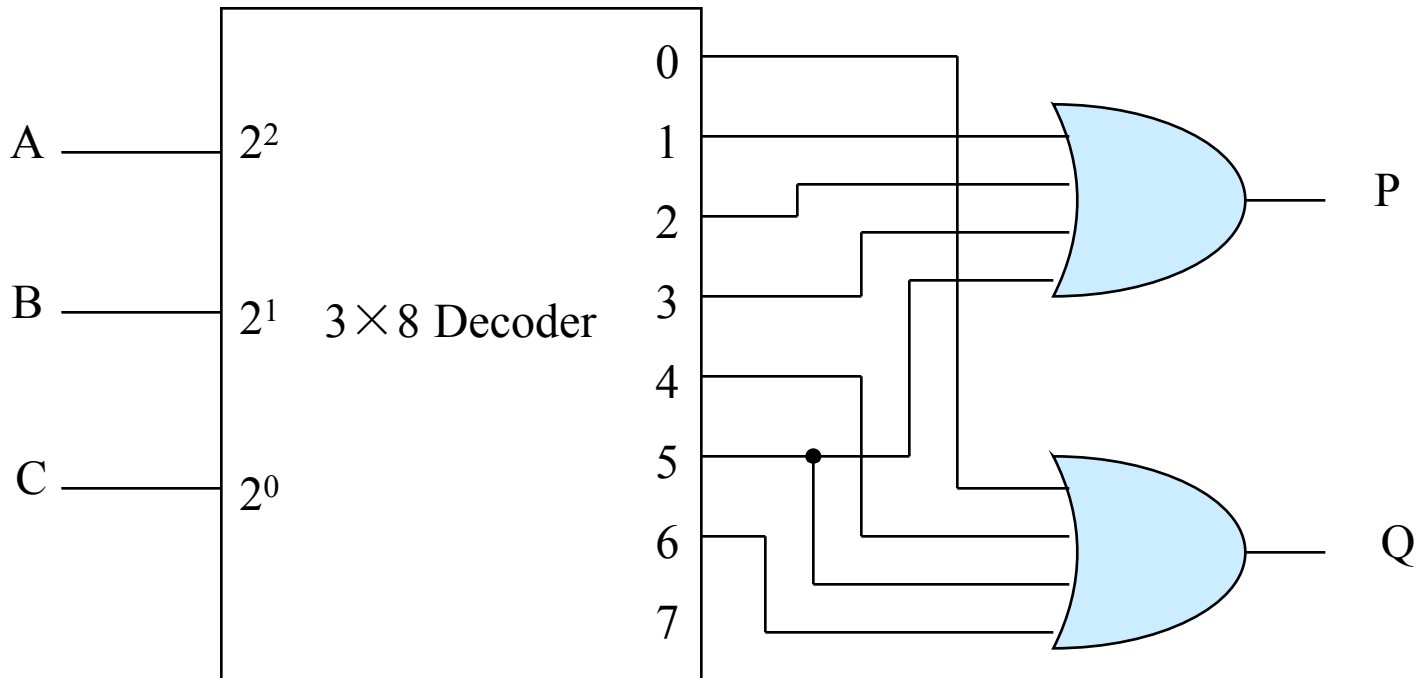
# Gate-Level Implementation of $3 \times 8$ Decoder



**Minterm  
Generator**

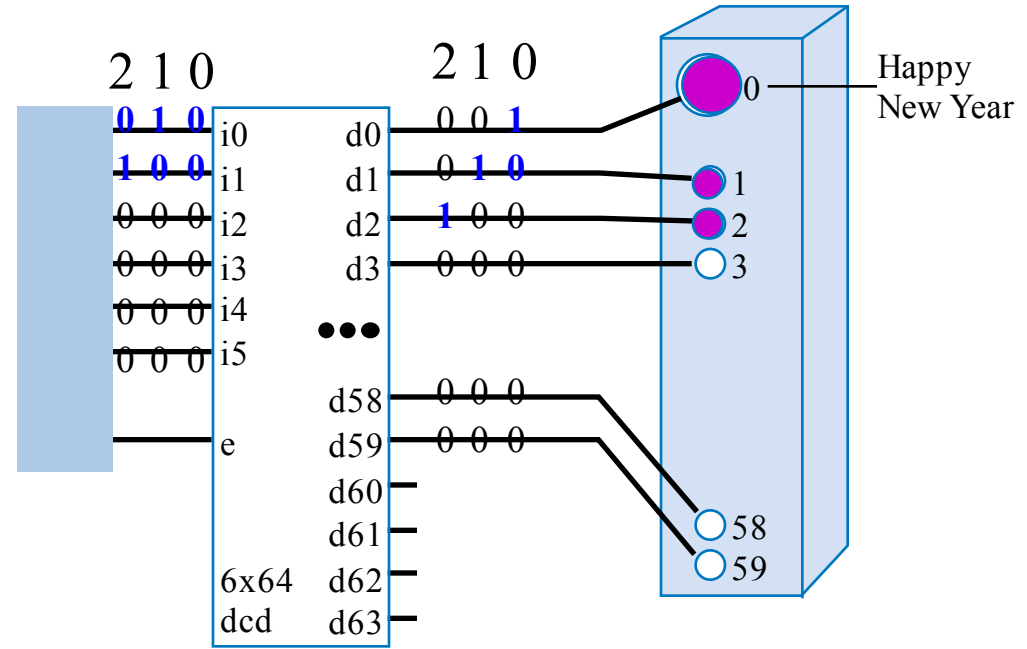
# Implementation of Any Circuit with a Decoder

- $P(A, B, C) = \sum m(1, 2, 3, 5)$  and  $Q(A, B, C) = \sum m(0, 4, 5, 6)$
- Since there are three inputs and total of 8 minterms, we can implement the circuits with a 3-to-8-line decoder



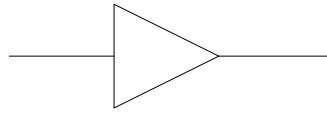
## Decoder Example

- New Year's Eve Countdown Display
  - Microprocessor counts from 59 down to 0 in binary on 6-bit output
  - Want illuminate one of 60 lights for each binary number
  - Use 6x64 decoder
    - 4 outputs unused

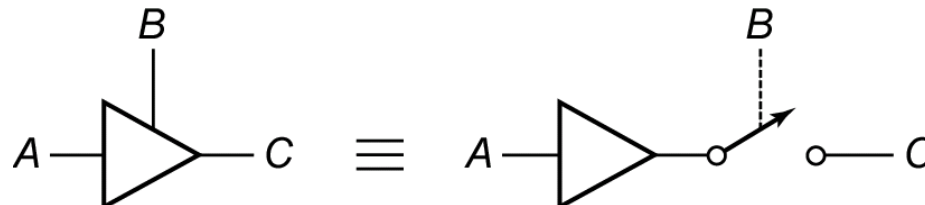


# Buffer and Three State (Tri-state) Buffer

- A buffer is a one directional transmission logic device
  - somewhat like a NOT gate without complementing the binary value
  - amplify the driving capability of a signal
  - insert delay
  - Protect input from output

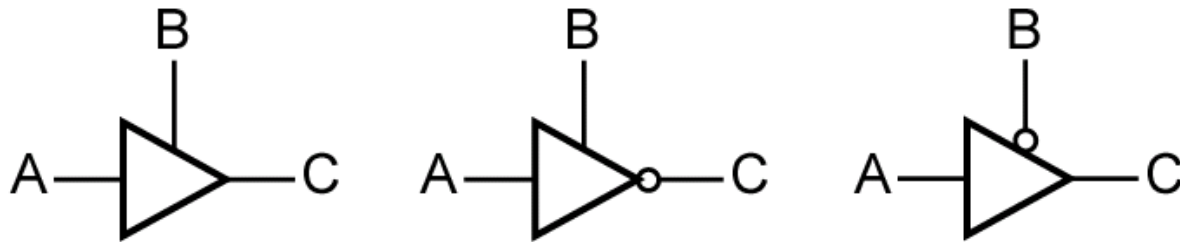


- A three state (Tri-state) buffer can have three different output values, controlled by an enable bit
  - 0 (off) when  $B = 1$ ,  $A = 0$ ;
  - 1 (on) when  $B = 1$ ,  $A = 1$ ;
  - Z (high impedance, no voltage) when  $B = 0$ ,  $A = x$ ;



# Tri-state Buffers

- Different types of tri-state buffers

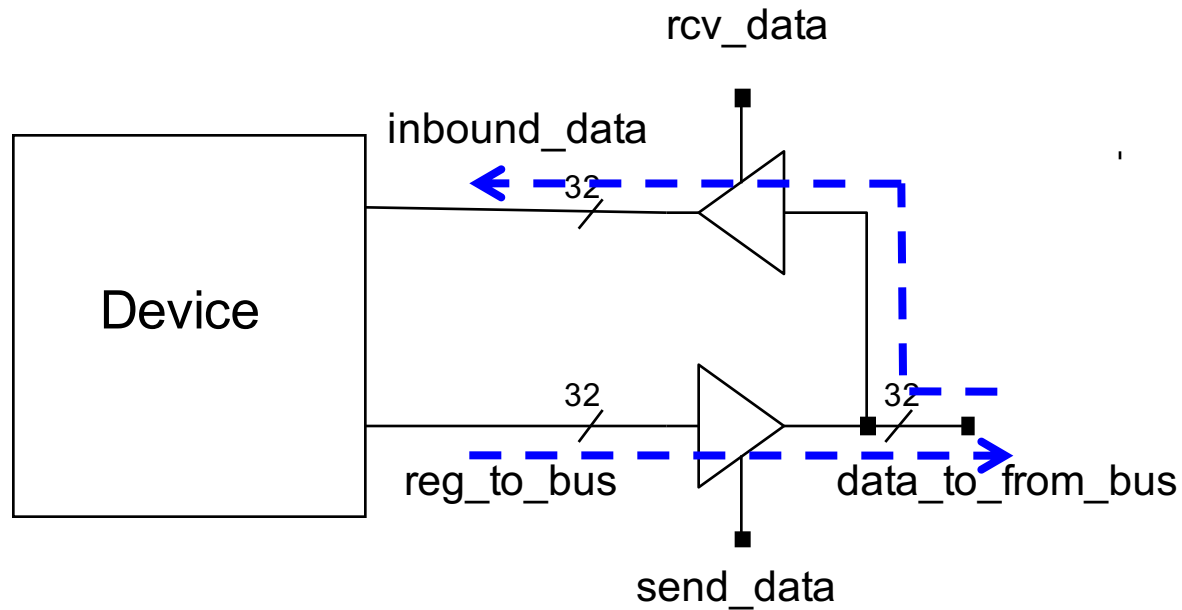


B	A	C
0	0	Z
0	1	Z
1	0	0
1	1	1

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

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

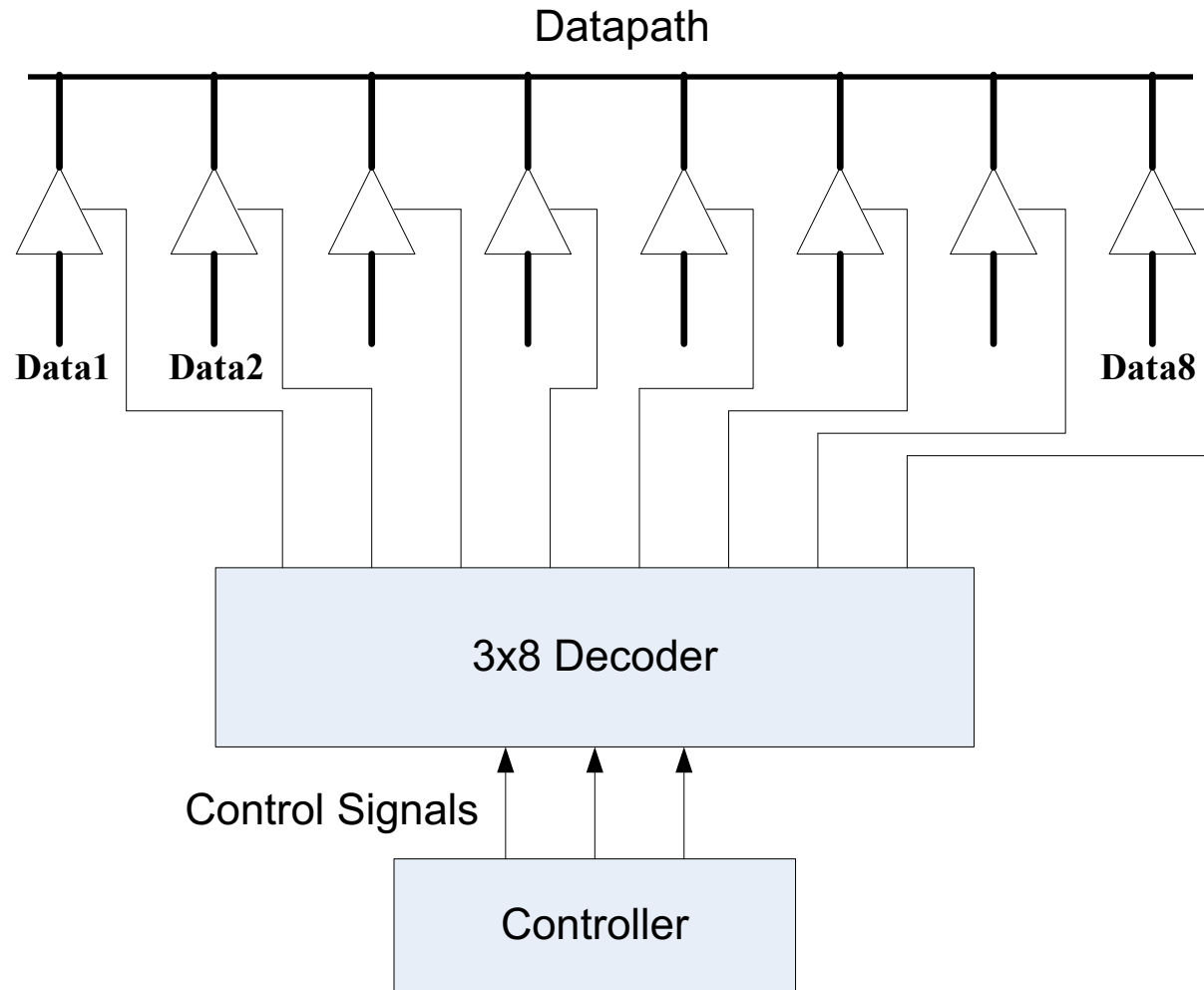
# Tri-state Buffer Application – Bidirectional I/O Port





# Applications of Decoders and Tri-state Buffers

- Based on the control signals, only one path will be connected



# Alternative Implementation of Datapath Control

- Based on the control signals, only one path will be connected

