Combinatorial Logic Design

Dr. Igor Ivkovic

iivkovic@uwaterloo.ca

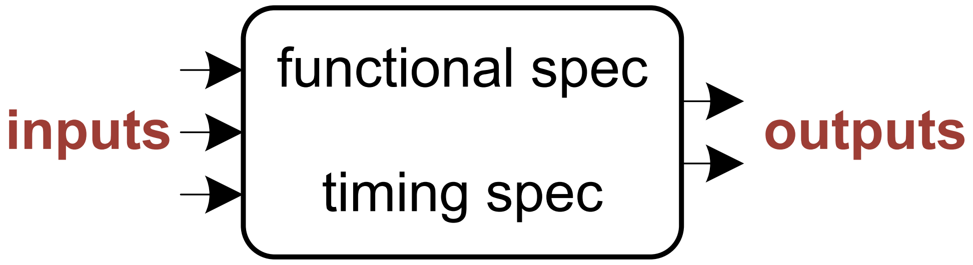
[with material from “Computer Organization and Design” by Patterson and Hennessy, and “Digital Design and Computer Architecture” by Harris and Harris, both published by Morgan Kaufmann]

Objectives

- Introduction to Logic Circuits
- Boolean Equations and Boolean Algebra
- From Logic to Gates
- Additional Circuit Elements
- Karnaugh Maps
- Timing and Glitches

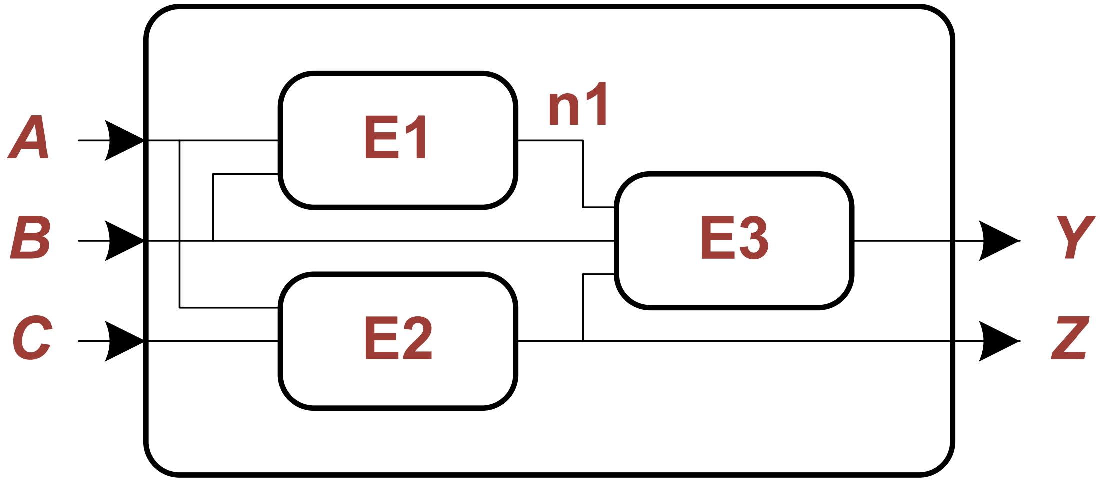
Logic Circuit /1

- Electric Circuit:
  - Provides a path through which the electrical current can flow
- **Logic Circuit:**
  - An electric circuit that performs logical operations on input signals that carry discrete values
- **A logic circuit is composed of:**
  - Input terminals/nodes
  - Output terminals/nodes
  - Functional specification
  - Timing specification



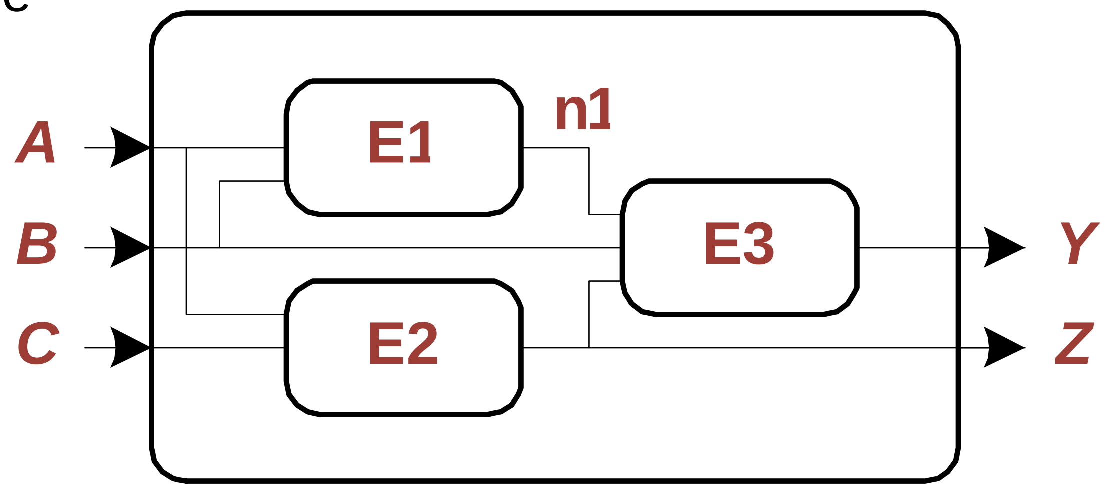
Logic Circuit /2

- **Input Terminals / Nodes:**
  - Wires whose voltage represents a value of a discrete variable
  - **Three types: Input, Output, and Internal**
  - Inputs receive values from the external world
- Example:
  - Input: A, B, C; Output: Y, Z; Internal: n1



Logic Circuit /3

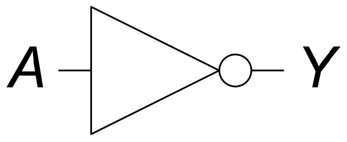
- **Circuit Element:**
  - It is also a circuit with its own inputs, outputs, and specification
  - Elements in the example: E1, E2, E3
- **Functional Specification:**
  - Describes the relationship between inputs and outputs
- **Timing Specification:**
  - Specifies the delay between inputs change and output response



Logic Circuit /4

- **Logic Gate:**
  - A circuit element that performs a basic logic function
  - Examples include: NOT, AND, OR, NAND, NOR, XOR, NXOR
  - Single Input: NOT, BUFFER
  - Double Input: AND, OR, NAND, NOR, XOR, NXOR
  - Multiple Input: NOR3, AND4

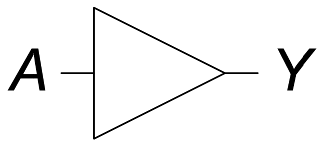
NOT



$Y = \overline{A}$

A	Y
0	1
1	0

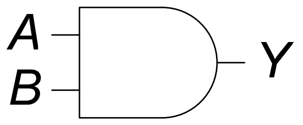
BUFFER



$Y = A$

A	Y
0	0
1	1

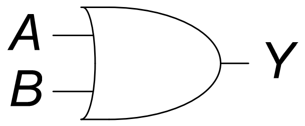
AND



$Y = AB$

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

OR



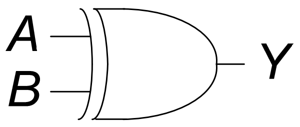
$Y = A + B$

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

Logic Circuit /5

- **Logic Gates Continued:**

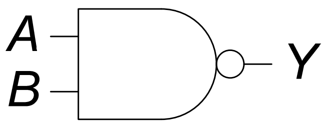
XOR



$Y = A \oplus B$

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

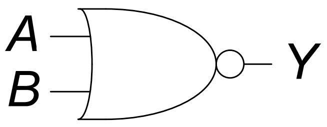
NAND



$Y = \overline{AB}$

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

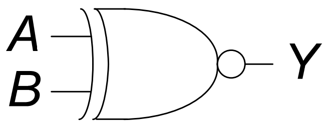
NOR



$Y = \overline{A + B}$

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

XNOR



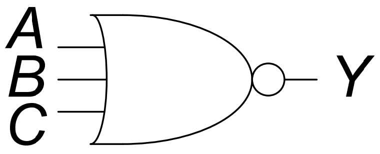
$Y = \overline{A \oplus B}$

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

Logic Circuit /6

- **Logic Gates Continued:**

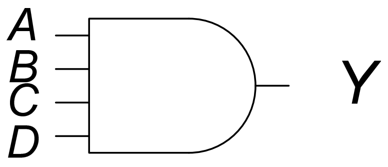
NOR3



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

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

AND4

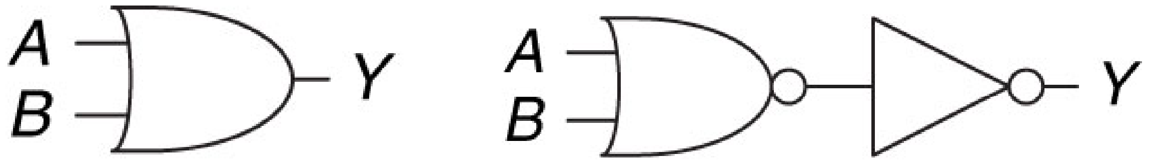


$Y = ABCD$

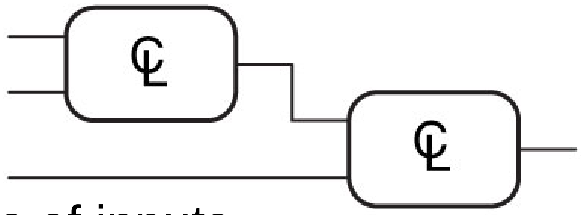
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Logic Circuit /7

- **Example of equivalent circuits:**



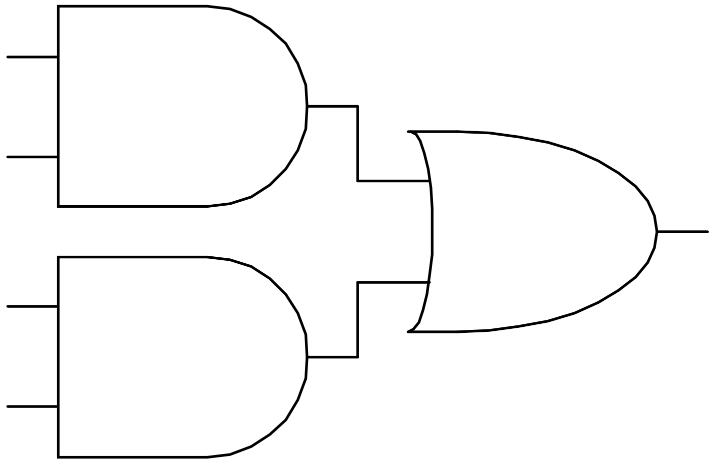
- **Types of Logic Circuits:**
  - Combinational Logic
  - Sequential Logic



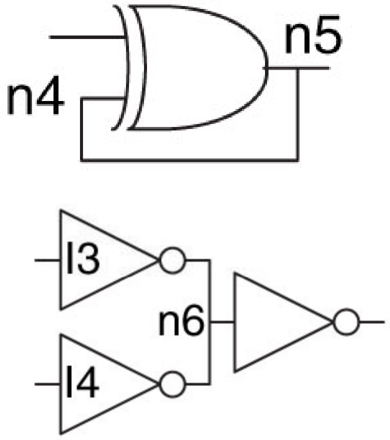
- **Combinational Logic Circuit:**
  - Has no memory
  - Outputs determined by the current values of inputs
- **Sequential Logic Circuit:**
  - Has memory
  - Outputs determined by the previous and the current values of inputs

Combinatorial Logic Circuit /1

- **A circuit is combinatorial if it consists of connected circuit elements, such that:**
  - The circuit contains no cyclic paths
  - Each circuit element is combinational
  - Each node is either an input terminal or it connects to exactly one output terminal
- **Examples:**
  - Valid combinatorial circuit:

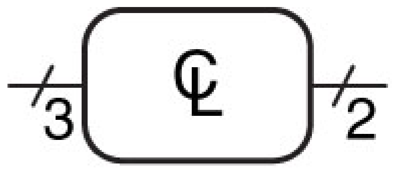


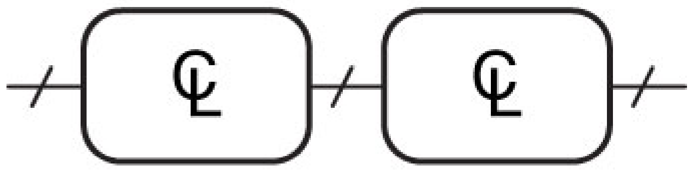
Invalid combinatorial circuits:



Combinatorial Logic Circuit /2

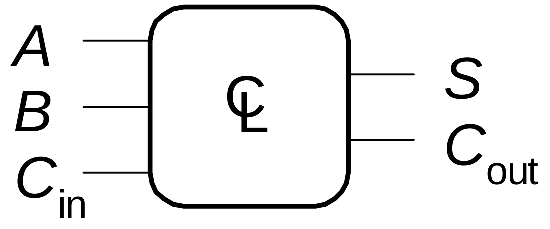
- **Bus:**
  - A bundle of multiple signals, either input or output
  - Shown by a slash through the signal line, with the number next to the slash indicating the number of bits in the bus
  - If the number of bits is unimportant or obvious, the number can be omitted
- **Examples:**





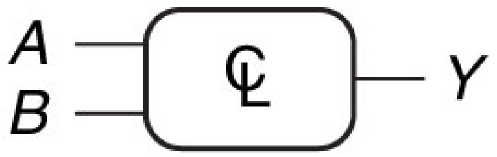
Combinatorial Logic Circuit /3

- **Boolean Equations**
  - Functional specification of outputs in terms of inputs
- **Example1:**



$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

- **Example2:**



$$Y = F(A, B) = A + B$$

Combinatorial Logic Circuit /4

- **Variable Complement:** A variable with a bar over it $\overline{B}$
- **Literal:** A variable or its complement B *or* $\overline{B}$
- **Implicant:** A product of literals $\overline{A}\,\overline{B}$
- **Minterm:**
  - A product that includes all input variables
- **Maxterm:**
  - A sum that includes all input variables

Combinatorial Logic Circuit /5

- **Sum-of-Products Format:**
  - Each row can be written as a minterm
  - A minterm is a product (AND) of literals
  - **Each minterm is TRUE for that row and only that row**
  - Form function by combining minterms for which the output is TRUE using OR
  - Hence, a sum (OR) of products (AND terms)
  - Suitable for representing functions that have truth tables with more FALSE rows than TRUE rows
- **Example:**
  - $Y = F(A, B) = \overline{A}B + AB$

A	*B*	*Y*	minterm	minterm name
0	0	0	$\overline{A}\ \overline{B}$	m_0
0	1	1	$\overline{A}\ B$	m_1
1	0	0	$A\ \overline{B}$	m_2
1	1	1	$A\ B$	m_3

Combinatorial Logic Circuit /6

- **Product-of-Sums Format:**
  - Each row has a maxterm
  - A maxterm is a sum (OR) of literals
  - **Each maxterm is FALSE for that row and only that row**
  - Form function by combining maxterms for which the output is FALSE using AND
  - Thus, a product (AND) of sums (OR terms)
  - Suitable for representing functions that have truth tables with more TRUE rows than FALSE rows
- **Example:**
  - $Y = F(A, B) = (A + B)(\overline{A} + B)$

A	B	Y	maxterm	maxterm name
0	0	0	$A + B$	M_0
0	1	1	$A + \overline{B}$	M_1
1	0	0	$\overline{A} + B$	M_2
1	1	1	$\overline{A} + \overline{B}$	M_3

Combinatorial Logic Circuit /7

- **Boolean Equations Example:**
  - You are going to excel in this course (E as output)
  - If you study regularly (S as input), and
  - If you do not skip assignments ($\overline{A}$ as input)
- **Exercise:**
  - Construct a truth table for the above example
  - Express its functional specification in both the sum-of-products and product-of-sums formats
  - Which format is better suited for this function (if any)?
  - (Bonus) Draw the combinatorial logic circuits for both formats

Simplifying Boolean Equations /1

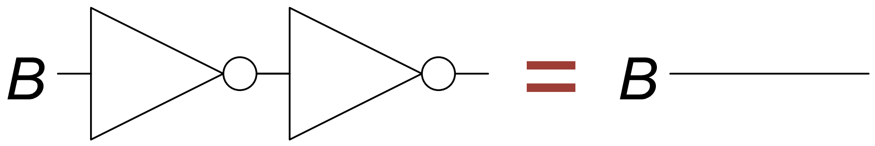
- **Why simplify Boolean equations?**
  - To decrease the number of Boolean operators
  - And in turn decrease the number of logical gates required to build the corresponding logical circuit
  - Which should make the circuit smaller, cheaper, and possibly faster to build
- **How to simplify Boolean equations?**
  - Using the corresponding simplification rules
  - Similar to traditional algebra, but simpler since variables have only two values (0 or 1)
- **Duality in simplification rules:**
  - ANDs and ORs, 0s and 1s are interchanged

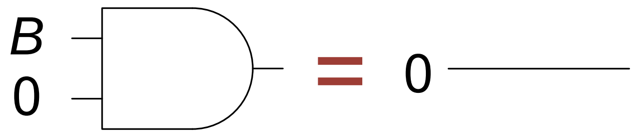
Simplifying Boolean Equations /2

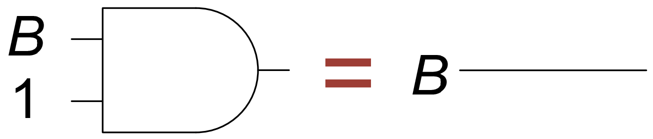
Rule	Dual Rule	
$\overline{\overline{X}} = X$		
$X + 0 = X$	$X \cdot 1 = X$	(identity)
$X + 1 = 1$	$X \cdot 0 = 0$	(zero/one)
$X + X = X$	$XX = X$	(absorption)
$X + \overline{X} = 1$	$X\overline{X} = 0$	(inverse)
$X + Y = Y + X$	$XY = YX$	(commutative)
$X + (Y + Z) = (X + Y) + Z$	$X(YZ) = (XY)Z$	(associative)
$X(Y + Z) = XY + XZ$	$X + YZ = (X + Y)(X + Z)$	(distributive)
$\overline{X + Y} = \overline{X} \cdot \overline{Y}$	$\overline{XY} = \overline{X} + \overline{Y}$	(DeMorgan)

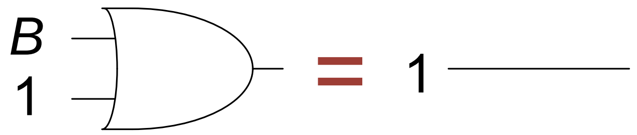
Simplifying Boolean Equations /3

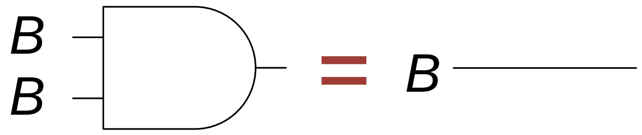
- **Simplification Rules as Logical Circuits:**

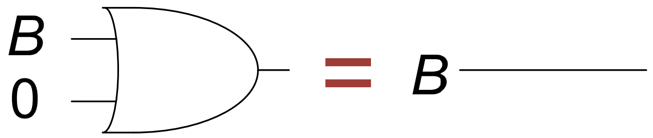


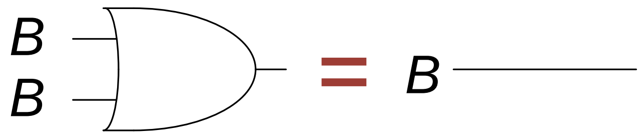












Simplifying Boolean Equations /4

- **Examples:**
  - Simplify $Y = AB + \overline{A}B$

$$= B(A + \overline{A})$$
$$= B(1)$$
$$= B$$

  - Simplify $Y = A(AB + ABC)$

$$= A(AB(1 + C))$$
$$= A(AB(1))$$
$$= A(AB)$$
$$= (AA)B$$
$$= AB$$

Simplifying Boolean Equations /5

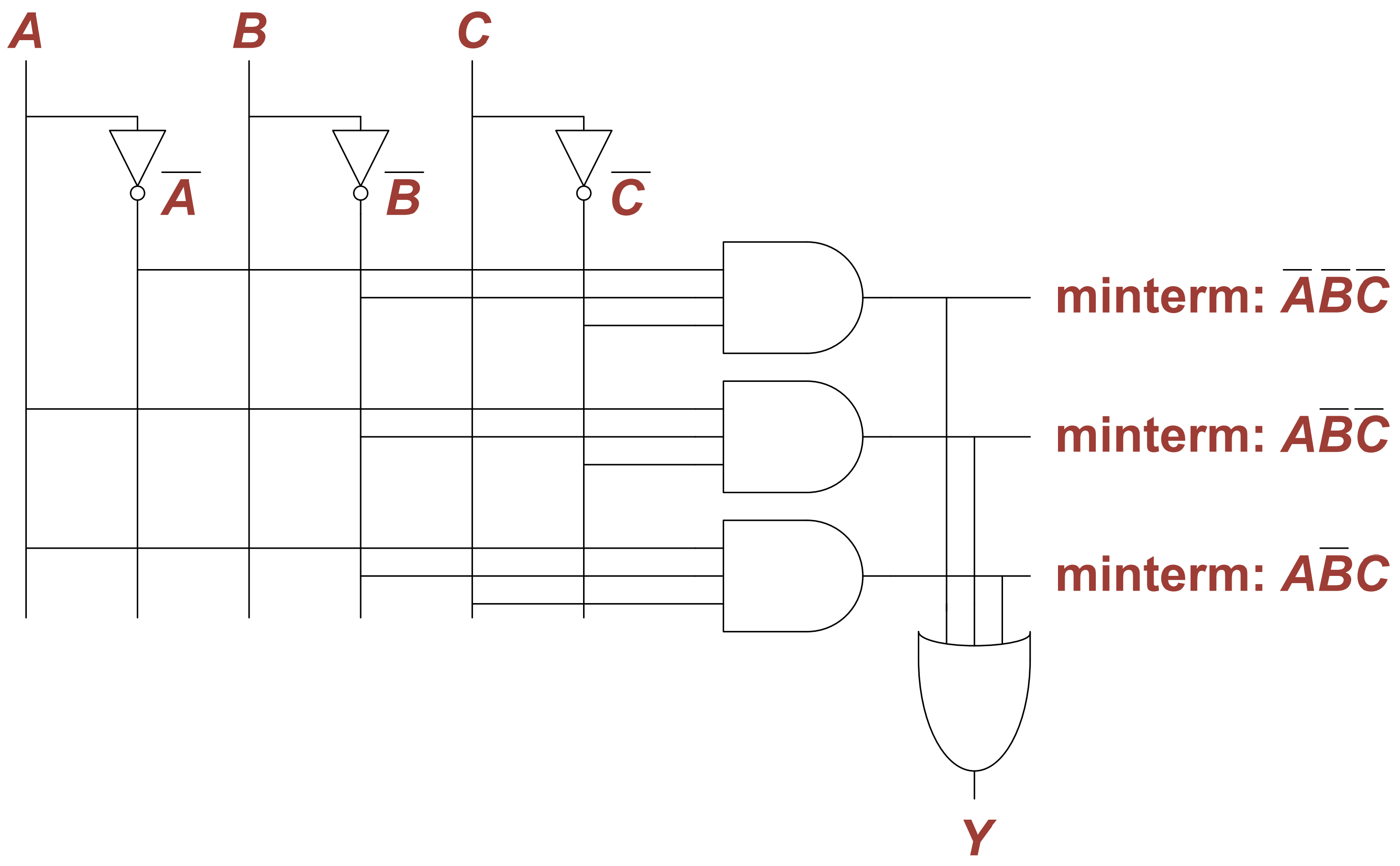
- **Exercise:**
  - Simplify $Y = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + A\overline{B}C$
  - Good Solution: $\overline{B}\,\overline{C} + A\overline{B}C$
  - Better Solution: $\overline{B}\,\overline{C} + A\overline{B}$
- **Discussion:**
  - First minimize the number of gates/operators, and then select a solution with he minimum number of literals

From Logic to Gates /1

- **Schematic:**
  - A diagram of a logic circuit that shows the circuit elements and the wires that connect them
- **Programmable Logic Array (PLA)-Style Schematic:**
  - An array of AND gates followed by an array of OR gates
  - That is, represent the equation in the sum-of-products format
- **How to draw the schematic:**
  1. Draw a column for each input
  2. Place inverters (NOT gates) in the adjacent columns
  3. Draw rows of AND gates for each minterm, and connect them to the corresponding inputs
  4. Draw an OR gate for each output, and connect them to the corresponding AND gates

From Logic to Gates /2

- **Example:**
  - $Y = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + A\overline{B}C$



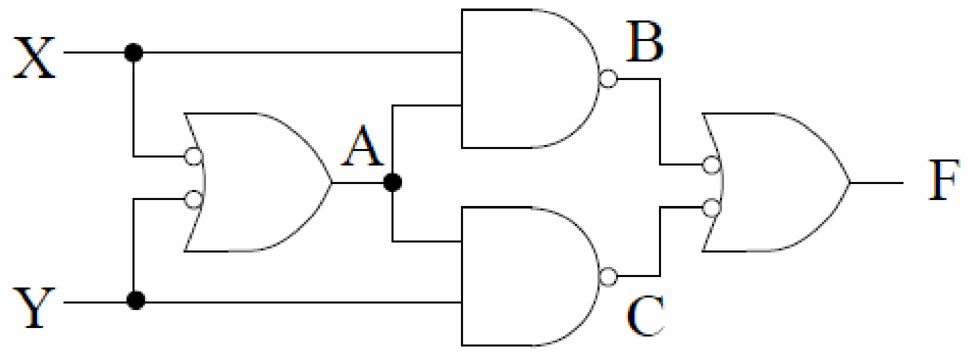
A
B
C
$\overline{A}$
$\overline{B}$
$\overline{C}$
minterm: $\overline{A}\,\overline{B}\,\overline{C}$
minterm: $A\overline{B}\,\overline{C}$
minterm: $A\overline{B}C$
Y

From Logic to Gates /3

- **Circuit Schematics Rules:**
  - Inputs should be placed on the left or on the top
  - Outputs should be placed on the right or on the bottom
  - Gates should flow from left to right
  - Straight wires should be used instead of jagged wires
  - Wires always connect at a T junction
  - **A dot where wires cross indicates a connection between the wires**
  - **Wires crossing without a dot make no connection**

From Logic to Gates /4

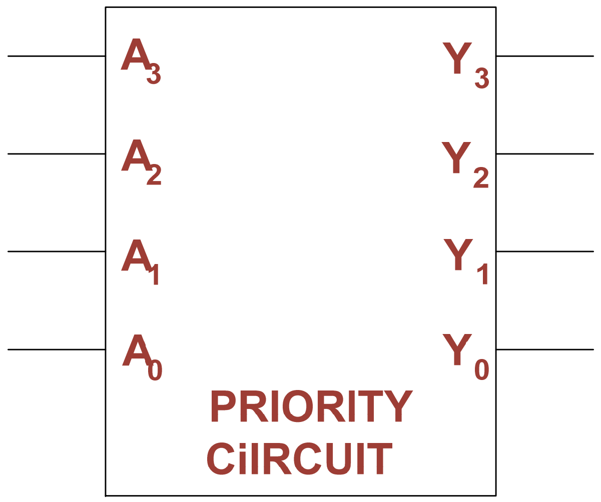
- **Exercise:**
  - Map the given circuit schematic as the truth table below



X	Y	A	B	C	F
0	0				
0	1				
1	0				
1	1				

From Logic to Gates /5

- **Example: Priority Circuit**
  - Output corresponds to the most significant TRUE input



A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

From Logic to Gates /6

- **Priority Circuit with “Don’t Cares”:**
  - Output corresponds to the most significant TRUE input

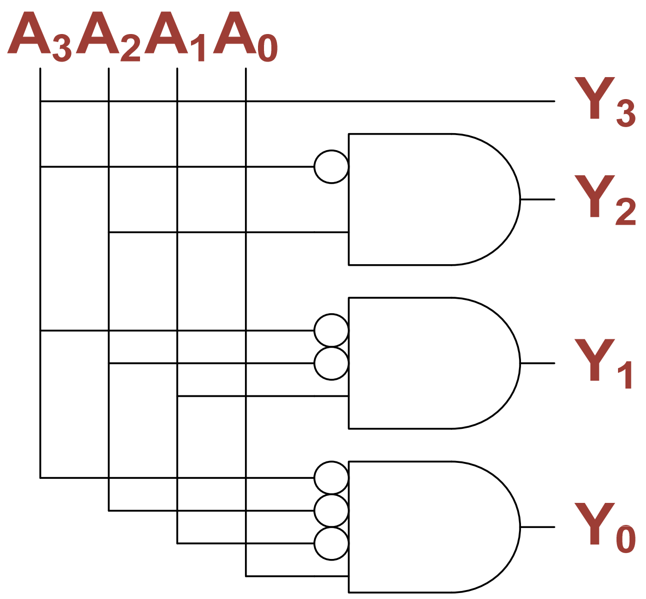
A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	X	0	0	1	0
0	1	X	X	0	1	0	0
1	X	X	X	1	0	0	0

From Logic to Gates /7

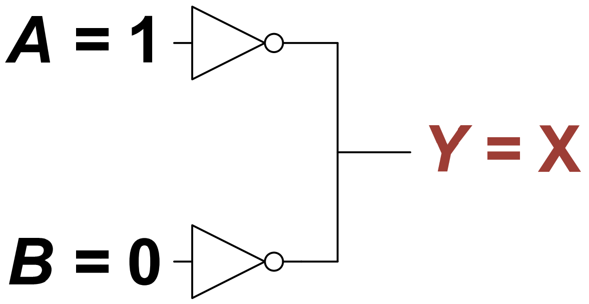
- **Priority Circuit Schematic:**
  - Output corresponds to the most significant TRUE input

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	X	0	0	1	0
0	1	X	X	0	1	0	0
1	X	X	X	1	0	0	0



From Logic to Gates /8

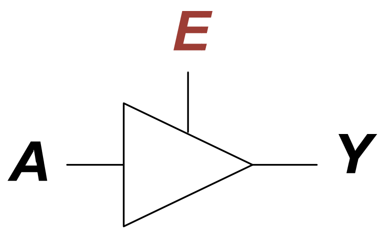
- **Contention:**
  - Circuit tries to drive output of 0 and 1
  - The actual value is 0, 1, or somewhere in between (in voltage)
  - Might change with voltage, temperature, time, noise
  - Often causes excessive power dissipation



- **Contention usually indicates a bug:**
  - X is used for “don’t care” and contention
  - Look at the context to tell them apart

From Logic to Gates /9

- **Tri-State Buffer:**
  - When enable (E) is true, it acts as a BUFFER gate
  - When enable (E) is false, the output Y floats

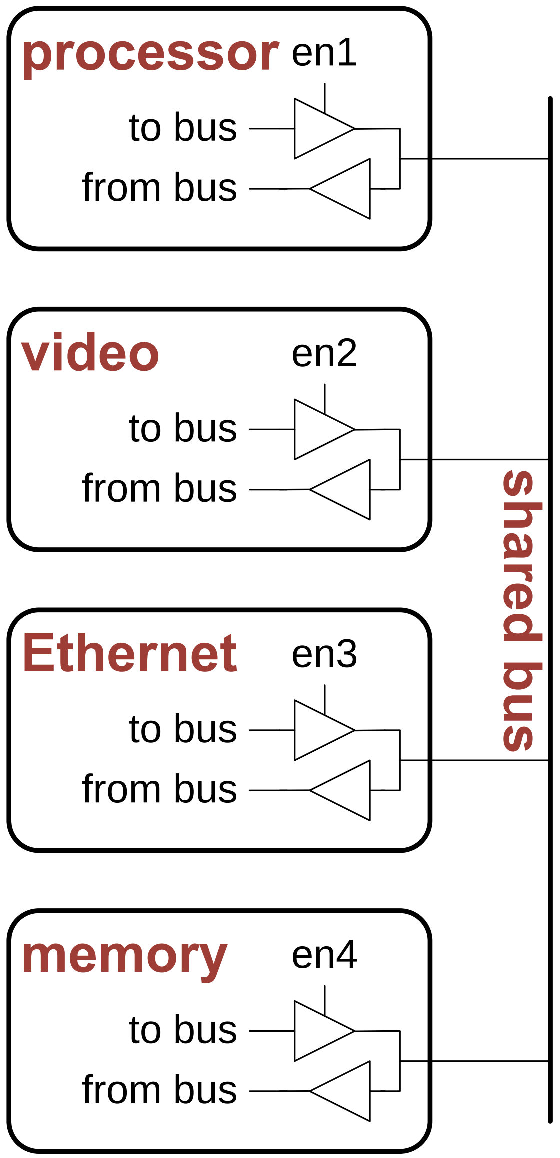


E	A	Y
0	0	z
0	1	z
1	0	**0**
1	1	**1**

- **Floating output:**
  - Floating output (Z) might be 0, 1, or somewhere in between (not necessarily an error)
  - It is driven to the appropriate logical value when some other circuit element activates it
  - **If not active, the floating output is not relevant to the overall operation of the circuit**

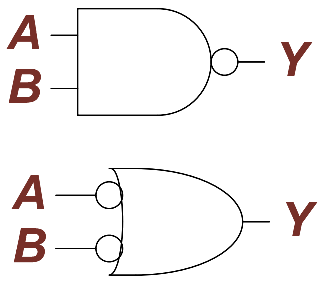
From Logic to Gates /10

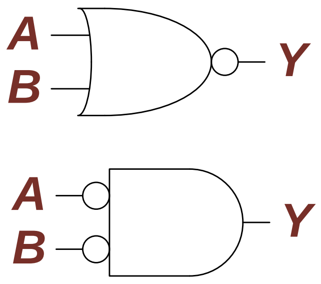
- **Tri-State Busses:**
  - Tri-state buffers are used to control interaction with the system memory by different hardware units
  - **Only one enable (en-i) signal is allowed to be active at one time**
  - As a result, only one hardware unit can communicate with the system memory at one time
  - Hardware units include the microprocessor, a video controller, and an Ethernet controller
    - Shared memory bus is not very common in the modern architectures
    - Instead, point-to-point links are used instead



Bubble Pushing /1

- **De Morgan's Theorem Applied:**
  - Certain circuitry types (e.g., CMOS circuitry) are better suited for NAND and NOR gates over AND and OR gates
  - $Y = \overline{AB} = \overline{A} + \overline{B}$ $Y = \overline{A + B} = \overline{A}\,\overline{B}$

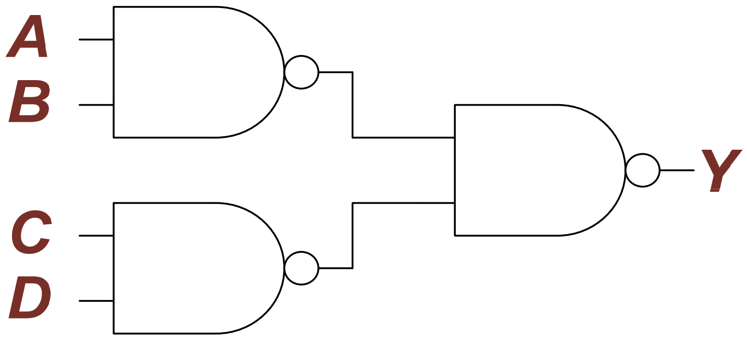




- **Bubble Pushing:**
  - Redrawing circuits by having the matching bubbles cancel out, so that the circuit functionality can be more easily determined

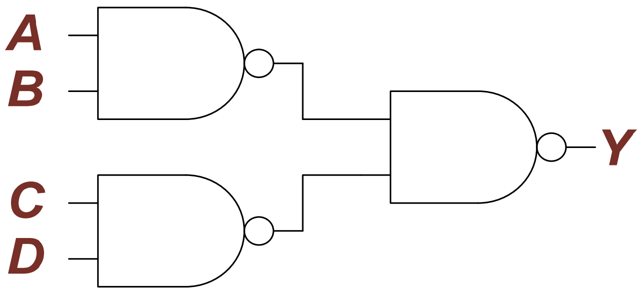
Bubble Pushing /2

- **Bubble Pushing Process:**
  - Start at the circuit outputs, and work towards the circuit inputs while using De Morgan's laws for individual pushing operations
  - Push any bubbles on the final output back towards the inputs, so that the output can be expressed directly (e.g., as Y) and not as its complement (e.g., as $\overline{Y}$)
  - **Redraw each gate so that the bubbles cancel**
    - If the current gate has an input bubble, redraw the preceding gate with an output bubble
    - If the current gate does not have an input bubble, redraw the preceding gate without an output bubble

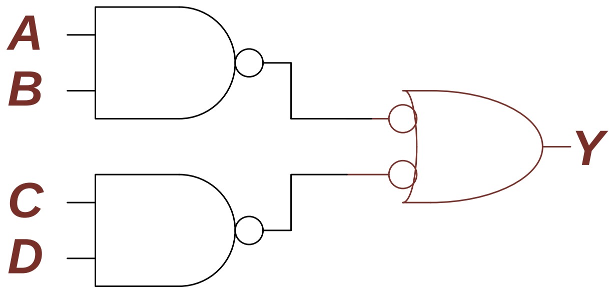


Bubble Pushing /3

- **Bubble Pushing Example1:**
  - Simplify this circuit:



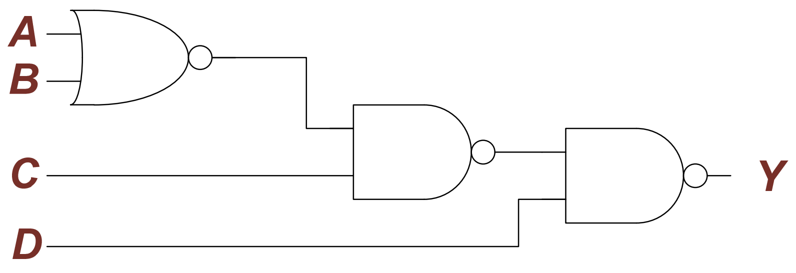
  - Step1. *Y* = *AB* + *CD*



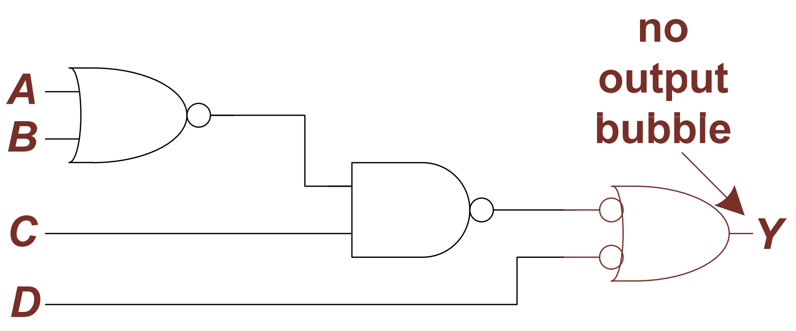
- Step 2. No further simplification possible
  - Both input gates have a bubble on their output to match the input bubble on the last OR gate

Bubble Pushing /4

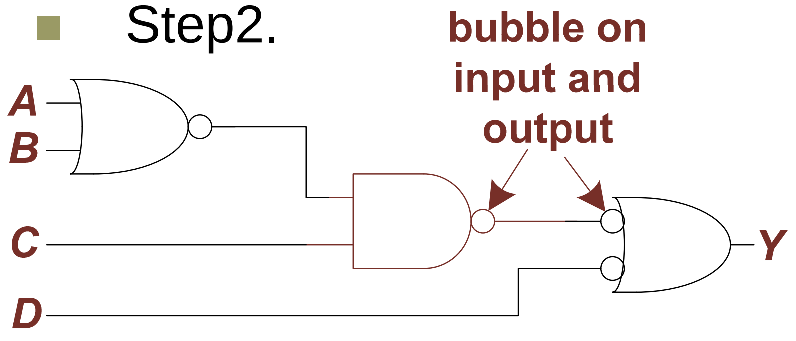
- **Bubble Pushing Example2:**
  - Simplify this circuit:



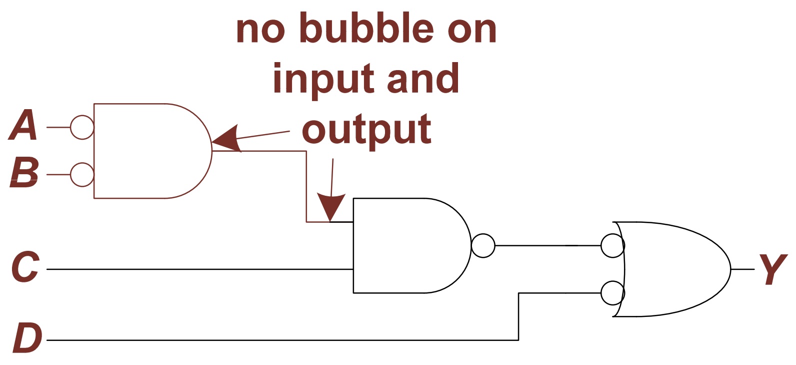
  - Step1.



  - Step2.



  - Step3.



$Y = \overline{A}\,\overline{B}C + \overline{D}$

Karnaugh Maps (K-Maps) /1

- **Karnaugh Maps (K-Maps):**
  - A technique for simplifying Boolean equations graphically
  - K-maps work well on equations with four variables or less
  - Based on simplification equation that eliminates A:
    $PA + P\overline{A} = P$
  - Each square in the K-map corresponds to a row in the matching truth table, and represents a single minterm

A	*B*	*C*	*Y*
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Y

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

Y

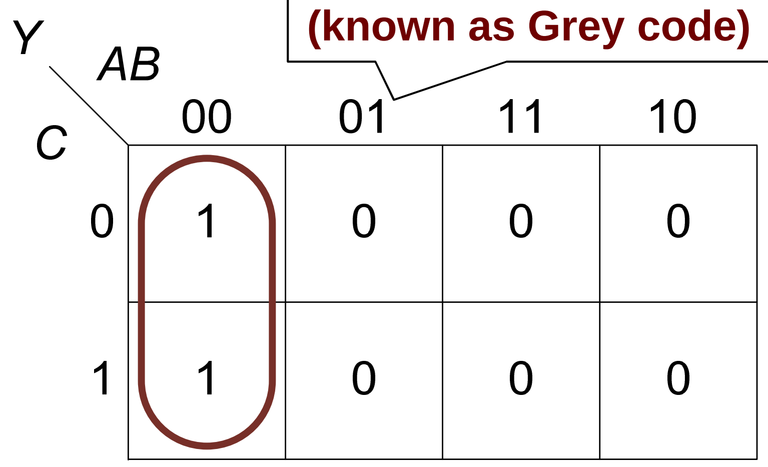
C \ *AB*	00	01	11	10
0	$\overline{A}\overline{B}\overline{C}$	$\overline{A}B\overline{C}$	$AB\overline{C}$	$A\overline{B}\overline{C}$
1	$\overline{A}\overline{B}C$	$\overline{A}BC$	ABC	$A\overline{B}C$

Karnaugh Maps (K-Maps) /2

- **K-maps introduction:**
  - Start by circling 1s in adjacent squares
  - In the resulting Boolean expression, include only literals whose both true and complement form are not in the circle
  - The literals which have both forms included in the circle will be cancelled out
  - **K-maps Example1:**

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

Adjacent entries differ only in a single variable (known as Grey code)



The resulting expression: $Y = \overline{A}\,\overline{B}$

Karnaugh Maps (K-Maps) /3

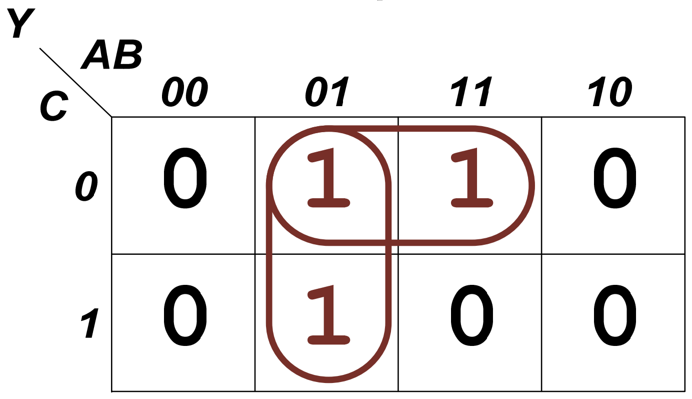
- **K-maps Example2:**

Y / $C \backslash AB$	00	01	11	10
0	$\overline{A}\,\overline{B}\,\overline{C}$	$\overline{A}B\overline{C}$	$AB\overline{C}$	$A\overline{B}\,\overline{C}$
1	$\overline{A}\,\overline{B}C$	$\overline{A}BC$	ABC	$A\overline{B}C$

Truth Table

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

K-Map



Y / $C \backslash AB$	00	01	11	10
0	0	1	1	0
1	0	1	0	0

- The resulting expression: $Y = \overline{A}B + B\overline{C}$

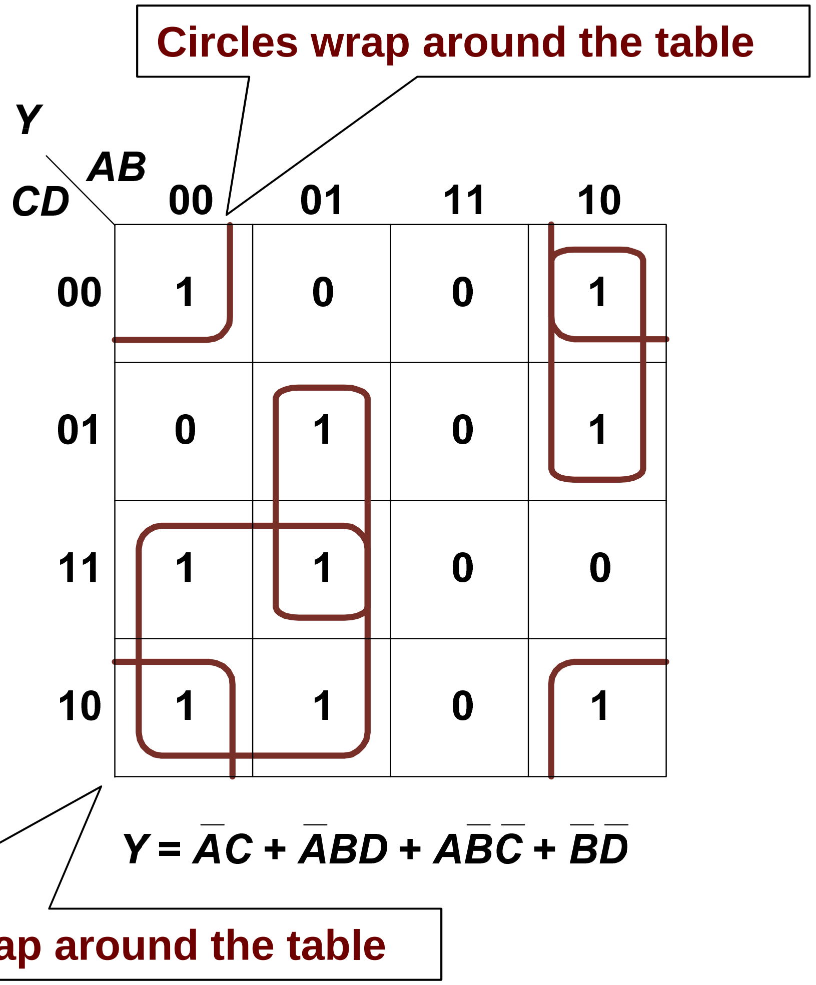
Karnaugh Maps (K-Maps) /4

- **K-maps elaboration:**
  - Every 1 must be circled at least once
  - Use the fewest number of circles to cover all 1s
  - Each circle must span a <u>rectangular</u> block of power of 2; that is, 1, 2, 4, etc. squares in each direction
  - Each circle must be as large as possible
  - A circle may wrap around the edges of the K-map table
  - A “don't care” (X) is circled only if it helps minimize the equation; it is not mandatory to circle any of the Xs
    - Extend an existing circle with Xs to simplify an expression

Karnaugh Maps (K-Maps) /5

- **K-maps Example3:**

A	*B*	*C*	*D*	*Y*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



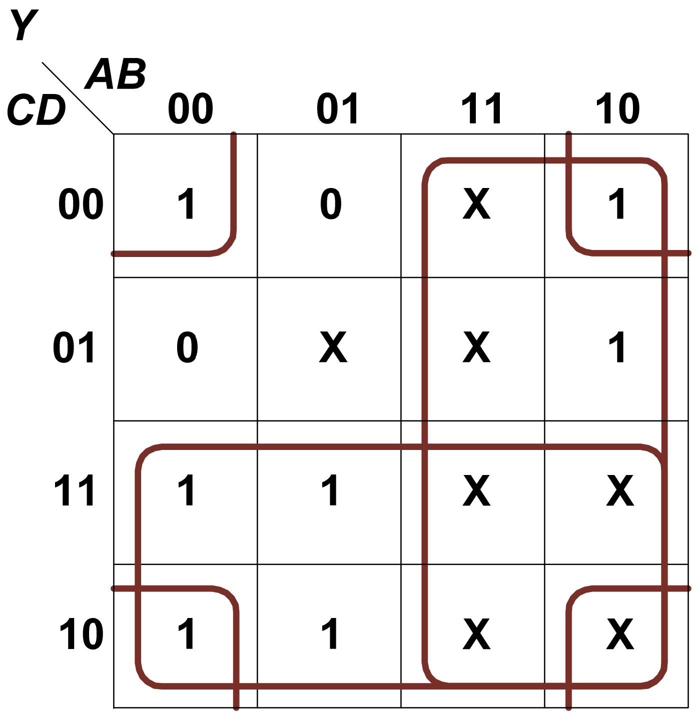
CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	1

$$Y = \overline{A}C + \overline{A}BD + A\overline{B}\,\overline{C} + \overline{B}\,\overline{D}$$

Karnaugh Maps (K-Maps) /6

- **K-maps Example4:**

A	*B*	*C*	*D*	*Y*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



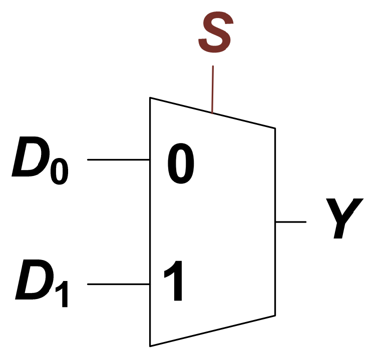
Y — *CD* \ *AB*	00	01	11	10
00	1	0	X	1
01	0	X	X	1
11	1	1	X	X
10	1	1	X	X

$Y = A + \overline{B}\,\overline{D} + C$

Multiplexer (Mux) /1

- **Multiplexer (Mux):**
  - Selects between one of N inputs and it connects it to the output
  - $\log_2 N$-bit selects and controls the input

- **Example: 2-to-1 Mux**

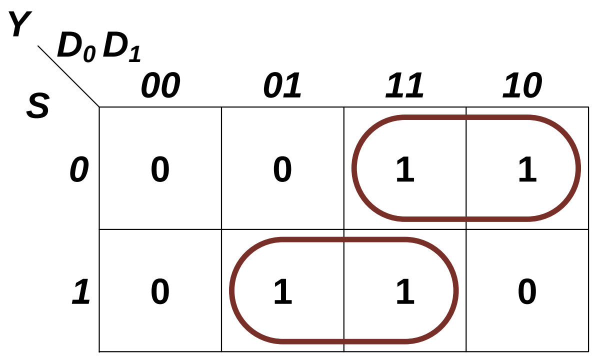


S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

S	Y
0	D_0
1	D_1

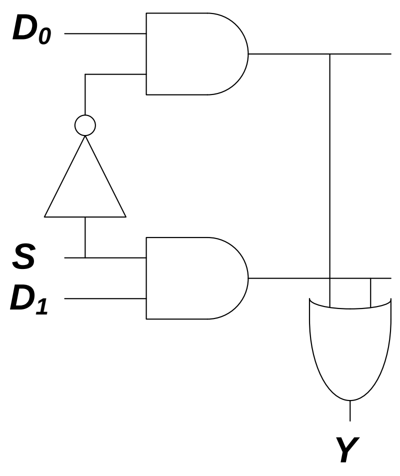
Multiplexer (Mux) /2

- **Mux as Logic Gates:**
  - Sum-of-products format

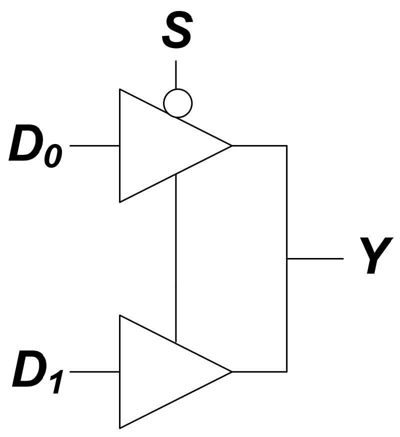


S \ D_0D_1	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$$Y = D_0\overline{S} + D_1S$$

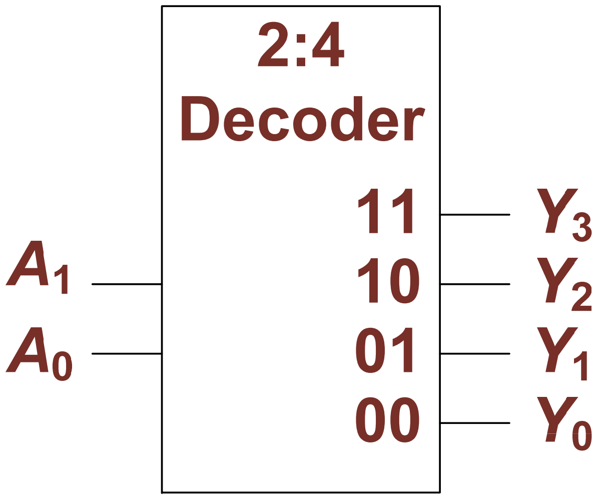


- **Mux as Tri-State Buffer:**
  - For an N-input Mux, use N tri-states
  - Turn on exactly one to select the appropriate input



Decoder /1

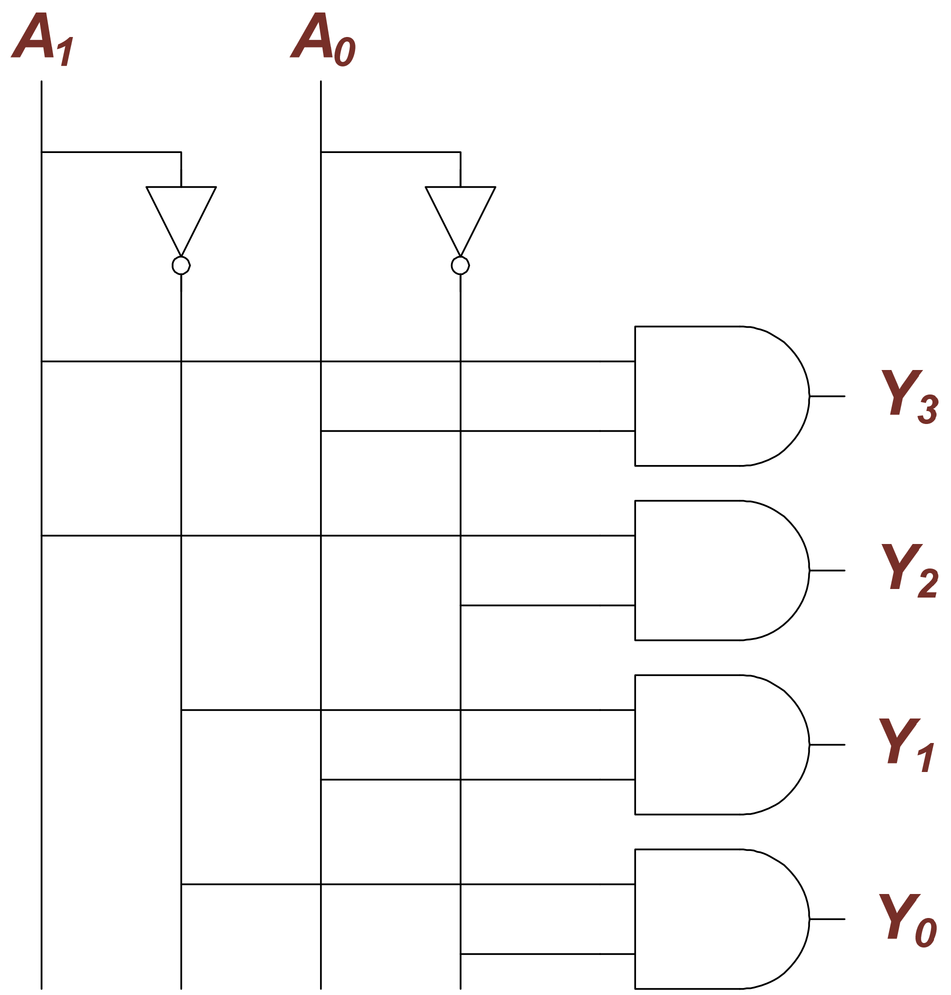
- **Decoder:**
  - Takes N inputs and provides 2^N outputs
  - Only one output selected at once



A_1	A_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

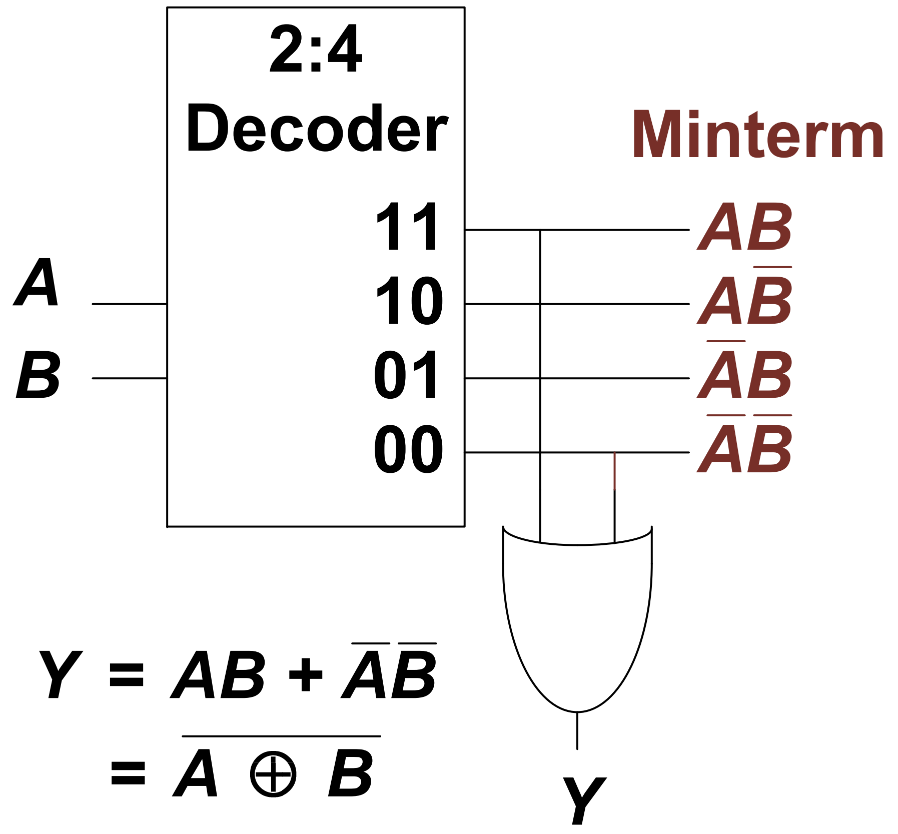
Decoder /2

- **Decoder implementation:**



Decoder /3

- **Decoder as a logic circuit:**
  - Example using OR of minterms



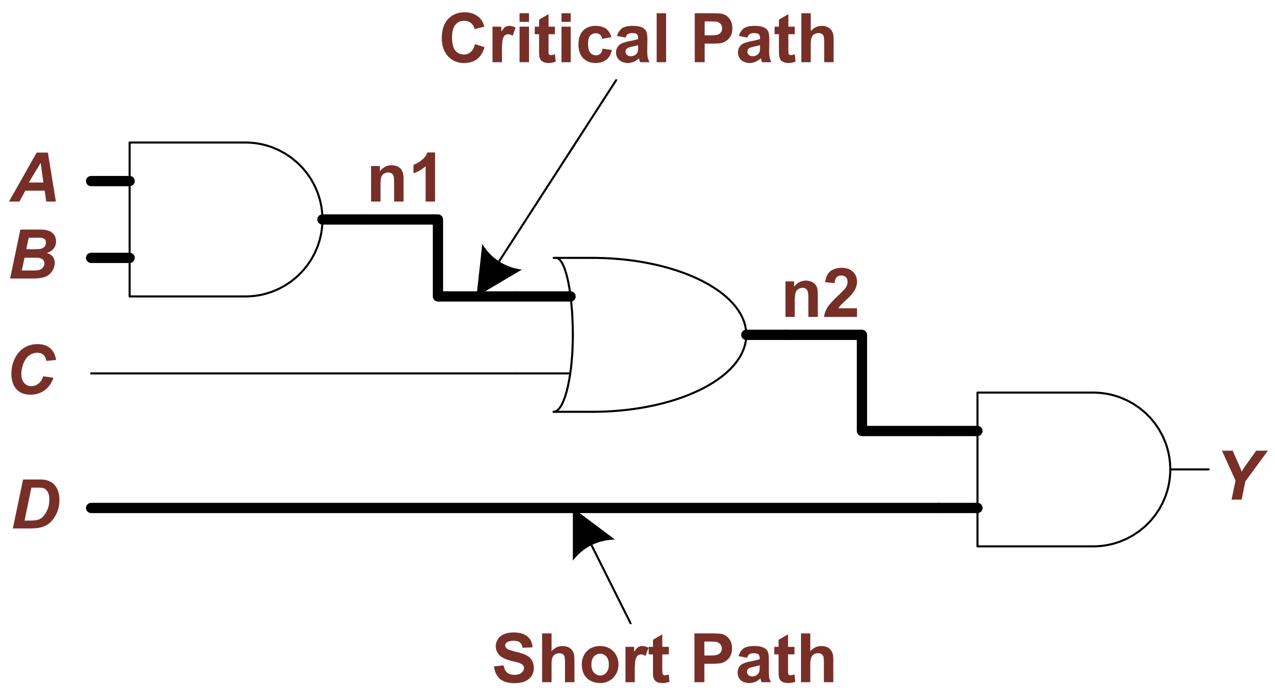
$$Y = AB + \overline{A}\,\overline{B}$$
$$= \overline{A \oplus B}$$

Timing and Glitches /1

- **Timing in a circuit:**
  - Delay between inputs change and output response
  - **Propagation delay:** t_{pd} = max delay from input to output
  - **Contamination delay:** t_{cd} = min delay from input to output
- **Delay is caused by**
  - Capacitance and resistance in a circuit
  - Speed of light limitation
- **Reasons why t_{pd} and t_{cd} may be different:**
  - Different rising and falling delays
  - Multiple inputs and outputs, some of which are faster than others
  - Circuits typically slow down when hot, and speed up when cold

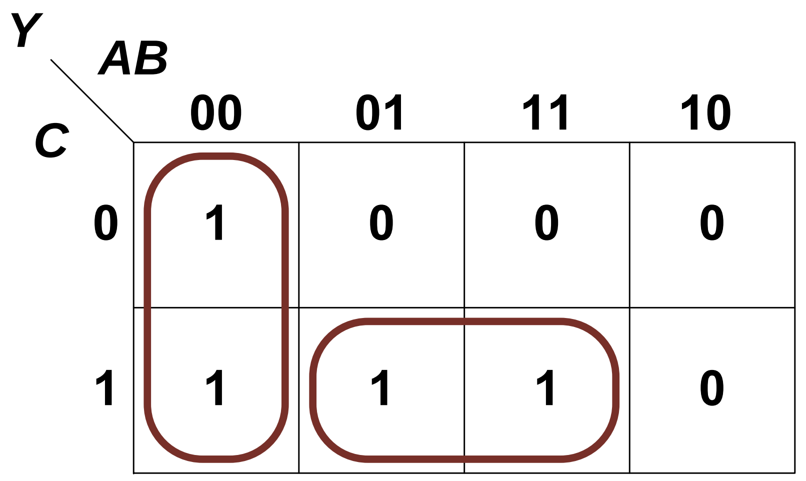
Timing and Glitches /2

- **Critical (Long) and Short Paths:**
  - Critical (Long) Path: $t_{pd} = 2 \times t_{pd\_AND} + t_{pd\_OR}$
  - Short Path: $t_{cd} = t_{cd\_AND}$



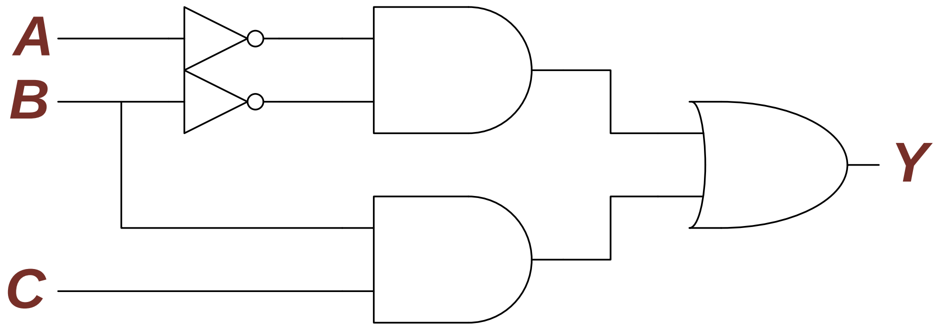
Timing and Glitches /3

- **Glitch:**
  - When a single input change causes multiple output changes
- Example:
  - What happens when A = 0, C = 1, and B goes from 1 to 0?



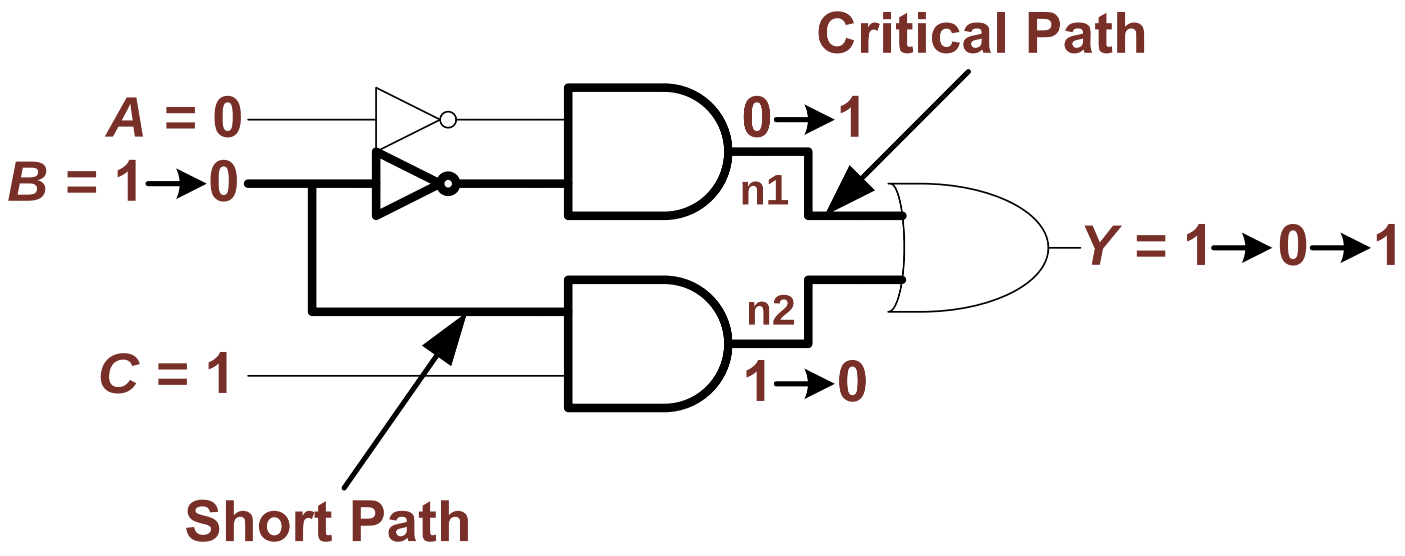
C \ AB	00	01	11	10
0	1	0	0	0
1	1	1	1	0

$Y = \bar{A}\bar{B} + BC$



Timing and Glitches /4

- **Glitch example visualized:**



- **Fixing the glitch**
  - How would you fix the glitch and prevent it from occurring?
  - Would adding another gate fix it? If yes, which gate?

Food for Thought

- **Download and Read Assignment #1 Specifications:**
  - Assignment #1 is intended as an introduction to the combinatorial logic design
- **Read:**
  - Appendix C of the course textbook
    - Review the material discussed in the lecture notes in more detail
  - (Optional) Chapter 2 of the Harris and Harris textbook

Additional Exercises

- Exercise 1.

A	*B*	*C*	*D*	*Y*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

1. Write a Boolean equation in sum-of-products format for the given truth table
2. Minimize the derived Boolean equation so that includes as small as possible number of minterms and literals
3. Draw a combinatorial circuit schematic representing the minimized Boolean equation using only NOT, AND, and OR gates
4. Draw a combinatorial circuit schematic representing the minimized Boolean equation using only NOT, NAND, and NOR gates

Additional Exercises

- Exercise 2.
  - Equation E1:

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

  - Equation E2:

$$Y = \overline{A + \overline{A}B + \overline{A}\,\overline{B}} + \overline{A + \overline{B}}$$

1. Minimize the given Boolean equation E1 and E2 so that each includes the minimum number of terms and literals
2. Draw a combinatorial circuit schematic for the minimized equations E1 and E2 so that each includes as small as possible number of gates; your schematic has to correctly match the minimized equation