ENGN2219/COMP6719
Computer Systems & Organization

Convener: Shoaib Akram

shoaib.akram@anu.edu.au

Australian National University

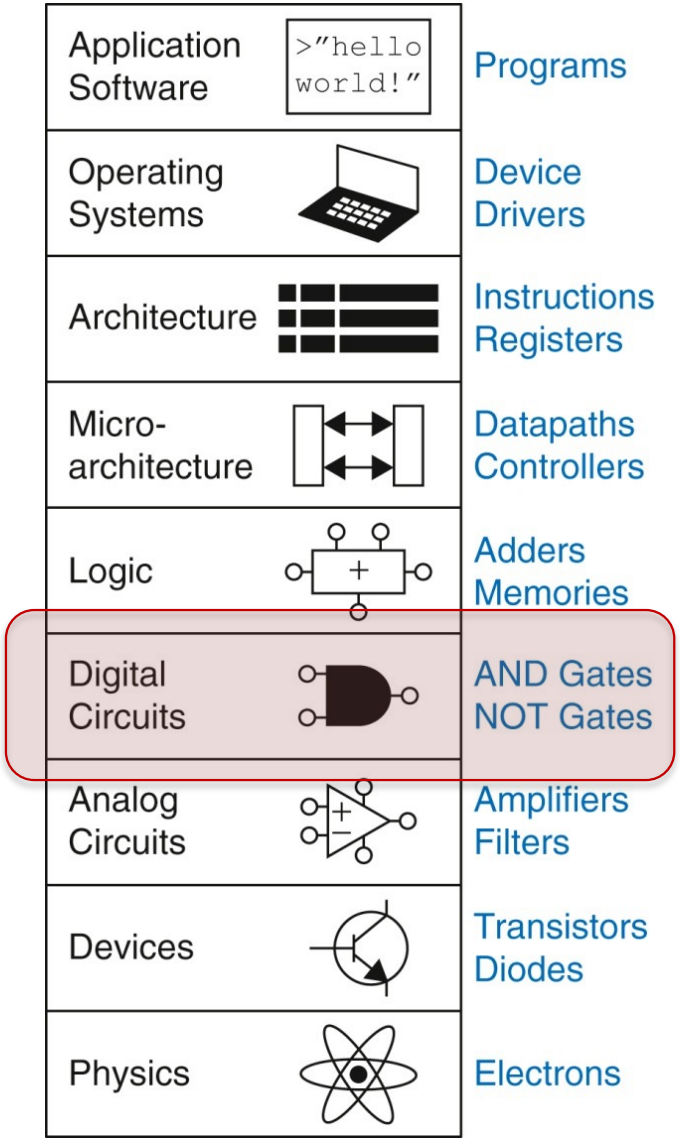
Plan: Week 2

Week 1: Digital abstraction and binary digits

Week 2: Number systems for binary variables, Logic gates

This Week: Boolean logic & Logic gates (contd)

This Week: Combinational logic (more than just gates)

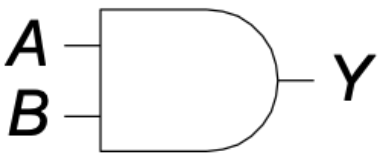

Application Software
>"hello world!"
Programs
Operating Systems
Device Drivers
Architecture
Instructions
Registers
Micro-architecture
Datapaths
Controllers
Logic
Adders
Memories
Digital Circuits
AND Gates
NOT Gates
Analog Circuits
Amplifiers
Filters
Devices
Transistors
Diodes
Physics
Electrons

We are here

The AND Function

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

Truth Table



AND Logic Gate

$Y = AB$

$Y = A.B$ (product)

$Y = A \cap B$ (intersection)

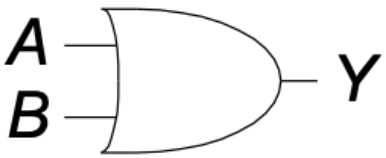
Boolean Equation

AND Function: *The output Y is 1 if and only if both A and B are 1*

The OR Function

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

Truth Table



OR Logic Gate

$Y = A + B$ (sum)

$Y = A \cup B$ (union)

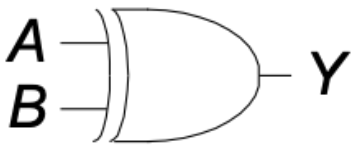
Boolean Equation

AND Function: *The output Y is 1 if either A or B are 1*

The XOR Function

eXclusive-OR

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



$Y = A \oplus B$

Truth Table

XOR Logic Gate

Boolean Equation

AND Function: *The output Y is 1 if A or B, but not both, are 1*

Terminology

The term *exclusive* is used because the output is 1 if only one of the inputs is 1

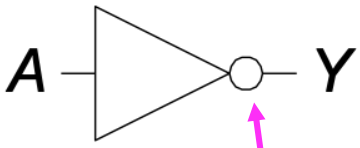
The OR function, on the other hand, produces an output 1, if only one of the two sources is a 1, or both sources are 1 (think of it as *inclusive* OR)

The NOT Unary Function

A	Y
0	1
0	1
1	0
1	0

Truth Table

The NOT gate has only one input (unary)



bubble → invert

NOT Logic Gate

$Y = A'$

$Y = \bar{A}$

Read as
Y = NOT A

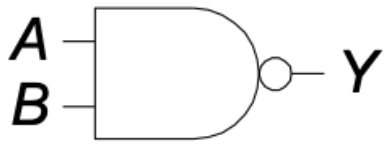
Boolean Equation

NOT Function: *The output Y is the inverse of the input A*
The NOT gate is also known as an inverter

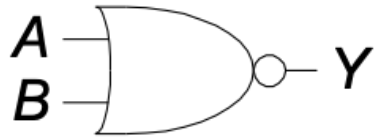
Inverting a Gate's Operation

Any gate can be followed by a bubble to invert its operation

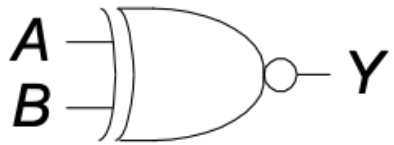
NOT AND → NAND



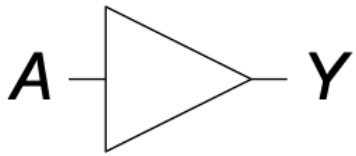
NOT OR → NOR



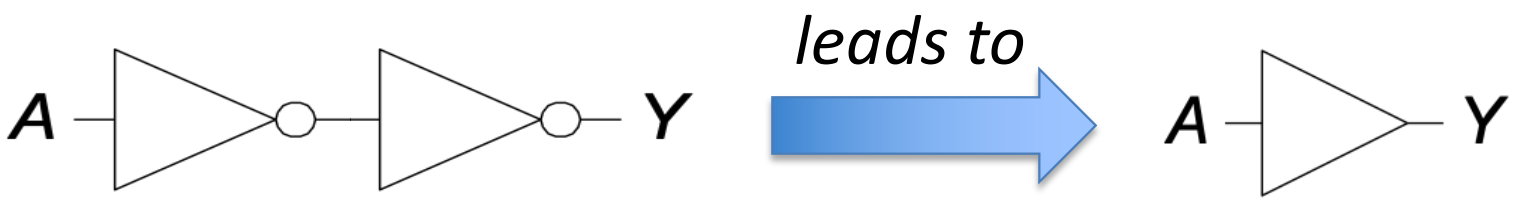
NOT XOR → XNOR



NOT NOT → BUF



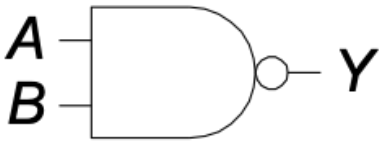
In Boolean logic, two wrongs make a right!



We say that two bubbles cancel each other's effect

The NAND Function

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



Y = (AB)’

Truth Table

NAND Logic Gate

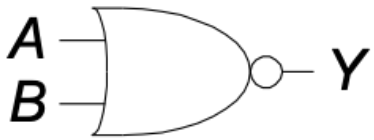
Boolean Equation

NAND Function: *The output Y is 1 unless both inputs are 1*

The NOR Function

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

Truth Table



NOR Logic Gate

$Y = (A + B)'$

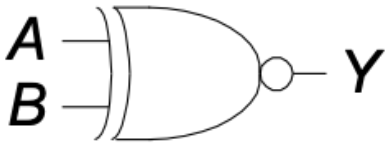
Boolean Equation

NOR Function: *The output Y is 1 if neither A nor B is 1*

The XNOR Function

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

Truth Table



XNOR Logic Gate

$Y = (A \oplus B)'$

Boolean Equation

XNOR Function: *The output Y is 1 if both A and B are 1 or both are 0*

XOR and XNOR are special

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

XOR

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

XNOR

XOR: Output is 1 when inputs are not equal (odd number of 1's)

Parity Gate

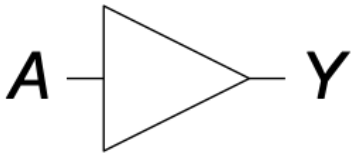
XNOR: Output is 1 when inputs are equal (even number of 1's)

Equality Gate

Buffer (BUF)

A	Y
0	0
0	0
1	1
1	1

Truth Table



BUF Logic Gate

$Y = A$

Boolean Equation

Buffer: *The output Y is equal to the input A*

Buffer (BUF)

- At the logic level, BUF is no more useful than a wire
- At a lower level of abstraction (analog level)
  - BUF can deliver a large amount of current to a motor
  - It can send output to many gates (think of an amplifier)

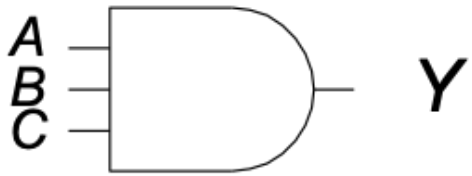
Critical to consider multiple layer of abstraction in the compute stack to understand the significance of various elements

Multiple-Input Gates

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

Gates with multiple inputs are possible

Looking at the truth table, can you guess the 3-input gate?



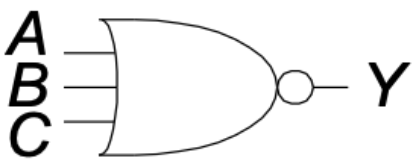
Y = ABC

Multiple-Input Gates

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

Gates with multiple inputs are possible

Looking at the truth table, can you guess the 3-input gate?



$Y = (A + B + C)'$

Bitwise Operations

All logical operators can be applied to two bit-patterns (i.e., a group of bits) of *m* bits each, where *m is any # bits (8, 16, ...)*

- Apply the operation individually to each pair of bits
- If A and B are 8-bit input sources (or source operands), then their AND or product, C, is also 8 bits

C = AB (bit-wise AND)

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

C = A + B (bit-wise OR)

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

Bit Masks

Suppose we are interested in extracting the least significant four bits from A, while ignoring the right-most four bits

- If we AND A with B, and choose B as 00001111, then we get the desired bit pattern in C
- **Bit mask:** A binary pattern (B) that separates the bits of A into two halves, the half we care about, and the half we wish to ignore

C = AB (bit-wise AND)

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

Exercises

Suppose we have a bit pattern, A = 11000010, and the rightmost two bits are of particular significance. Find a bitmask and a logical operation to mask out the values in the rightmost positions in a new bit pattern C. (**All other bits in C are set to 0.**)

Suppose we have a bit pattern, A = 10110010, and the leftmost two bits are of particular significance. Find a bitmask and a logical operation to mask out the values in the leftmost positions in a new bit pattern C. (**All other bits in C are set to 1.**)

Exercise

Suppose we want to know if two bit-patterns A and B are identical. How can we find out if two bit-patterns are identical?

Verify that, B AND 1 = B, where B is a binary variable. Also, verify that, B OR 0 = B.

Verify that, B AND 0 = 0, where B is a binary variable. Also, verify that, B OR 1 = 1.

Exercise

Verify that, B AND B = B, where B is a binary variable. Also, verify that, B OR B = B.

Verify that, B AND B’ = 0, where B is a binary variable. Also, verify that, B OR B’ = 1.

Key Ideas

Any physical quantity can represent TRUE (1) and FALSE (0). Computers use voltage levels for representing one and zero as electronic components such as transistors can distinguish between these two voltage levels. Our ability to shrink transistors has enabled faster computers in a small chip area (400 mm^2 approx.).

Voltage is a continuous physical signal. We can split voltage into as many levels as we want. We use only two levels to represent and manipulate binary variables to simplify circuits.

Using binary variables and Boolean logic to build computers leads to more efficient circuits and computers.

We can do arithmetic in any other base (e.g., 2) without learning Boolean and digital logic. There is nothing special about adding binary numbers compared to adding decimal numbers.

Key Ideas

We need Boolean logic to understand the interaction between binary variables, and understanding the interaction requires us to learn about logic functions. Logic functions can eventually lead us to build more complex digital circuits that solve real-world problems, e.g., adding two large numbers.

We (humans and, more specifically, John von Neumann) found a system of representation for binary numbers called two's complement. This representation simplifies building arithmetic circuits as a single circuit for adding two numbers can handle addition and subtraction. The circuit itself does not know about two's complement. We build circuits and computers today, assuming two's complement signed integers.

Classification of Digital Circuits

Combinational Circuit: Output depends on the current values of the inputs only

- Memoryless (a *distinct* and *critical* feature)
- All logic gates are combinational

Sequential Circuit: Output depends on the current and previous values of the inputs

- The sequence of inputs dictate the output
- Sequential circuits have state or memory
- Example: Elevator controller (**State:** TRANSIT, GROUND, TOP)

Combinational Behavior

Example: Suppose a combinational circuit, consisting of an AND gate, with two inputs, A and B

time →	*t0*	*t1*	*t2*	*t2*	*t4*	*t5*	*t6*
A	0	1	1	0	1	0	1
B	0	1	0	0	1	0	1
Output	0	1	0	0	1	0	1

At time t6, the *sequence* of changes to A and B between t0 – t5 is irrelevant. The output is strictly determined by the values of A and B at t6

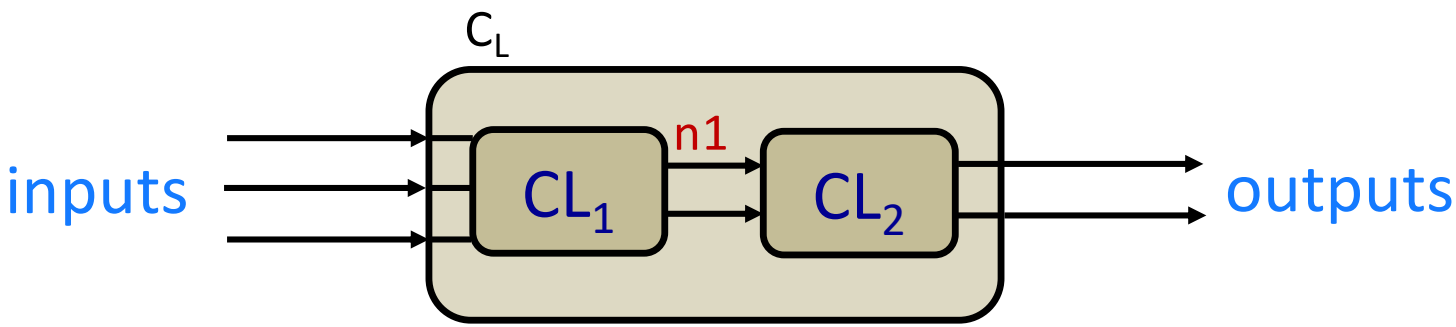
Combinational Circuits

Functional specification: What is the circuit supposed to do? What is the output for a given combination of input values?

Timing specification: How long does the circuit takes to produce the output?

- Worst-case: ten nanoseconds
- Best-case: one nanoseconds

Combinational Circuits



Hierarchy: The top-level circuit, C_L, is made up for of two sub-circuits (also combinational), CL_1 and CL_2

Nodes: n1 is an internal wire or node

Abstraction: The *input and output* interface, and the functional and timing specification is enough for someone to use C_L. They do not need to know the inner composition of C_L

Implementing Combinational Logic

Steps in implementing combinational Logic

1. Initial specification (e.g., in English)
2. Construct the truth table
3. Derive the Boolean equation
4. Simplify the Boolean equation (use Boolean algebra)
5. Implement the equation using logic gates

Functional specification

Specification

[Happiness detector] The students are back on campus. They are not *happy* if there is a *homework* deadline, or *Badger & Co.* is closed. Design a circuit that will output 1 only if students are happy.

[Multiplexer] Design a circuit with three inputs: D_0, D_1, select; and one output. The output is D_0 if select is 0, and D_1 if select is 1.

[Half Adder] Design a circuit that adds two binary variables: A and B. The circuit has two outputs: sum and carry-out (C_{out}).

[Full Adder] Design a circuit that adds three binary variables: A, B, and a carry-in (C_{in}). The circuit has two outputs: sum and carry-out (C_{out}).

Constructing Truth Tables

Identify inputs and outputs (interface)

- The inputs and outputs maybe implicitly specified
- *Or,* determining them may require some thought

Write all the possible combinations of input values

- For each input combination, determine the output
- *All inputs to the left, outputs to the right*

Truth Table: Happiness Detector

Specification: The students are back on campus. They are **not** *happy* if there is a *homework* deadline, or *Badger & Co.* is closed. Design a circuit that will output 1 only if students are *happy*.

Interface

Homework deadline? (D)

- 0: there is *not* a deadline
- 1: there is a deadline

Badger is closed? (B)

- 0: open
- 1: closed

Happy (H): 1 → ☺, 0 → ☹

Truth Table

D	B	H
0	0	1
0	1	0
1	0	0
1	1	0

Deriving a Boolean Equation

Some Terminology first

- For any binary variable X, its *compliment* is X'
- True form (X) and complementary form (X') are called *literals*
- AND of one or more literals is called a *product* or *implicant*
  - X, Y, XY, X'Y'Z, XYZ, XY'Z' are all implicants for a function of three variables
- **Minterm:** A product involving all the inputs to the function
  - XYZ is a minterm for a function of three variables X, Y, and Z
  - XY is not a minterm because it is missing one literal (Z)

Deriving a Boolean Equation

Order of operations

- NOT has the highest precedence
- Next is AND
- OR is last
- Example: Y = A + BC’
  - First, we find C’
  - Then, we find BC’ (product/AND)
  - Finally, we perform A + (the *result* of BC’)

Sum-of-Products Form

To write the Boolean equation for a truth table, *sum each of the minterms for which the output is 1*

A	B	Y1	minterm	name
0	0	0	A'B'	m_0
0	1	1	A'B	m_1
1	0	0	AB'	m_2
1	1	0	AB	m_3

<u>Boolean Eq</u>

Y1 = A'B

Y1 is 1 *only* when A = 0 and B = 1

Conversely, when A' = 1 and B = 1

Sum-of-Products Form

To write the Boolean equation for a truth table, *sum each of the minterms for which the output is 1*

A	B	Y1	minterm	name
0	0	0	A'B'	m_0
0	1	1	A'B	m_1
1	0	0	AB'	m_2
1	1	1	AB	m_3

<u>Boolean Eq</u>

Y1 = A'B **+** AB

Y1 is 1 ***either*** when A = 0 and B = 1

OR, when A = 1 and B = 1

$Y1 = \sum(1,3)$

Equation: Happiness Detector

Specification: The students are back on campus. They are **not** *happy* if there is a *homework* deadline, or *Badger & Co.* is closed. Design a circuit that will output 1 only if students are *happy*.

Truth Table

D	B	H
0	0	1
0	1	0
1	0	0
1	1	0

Boolean Eq

H = D'B'

H = (D)' AND (B)'

From Equation to Gates

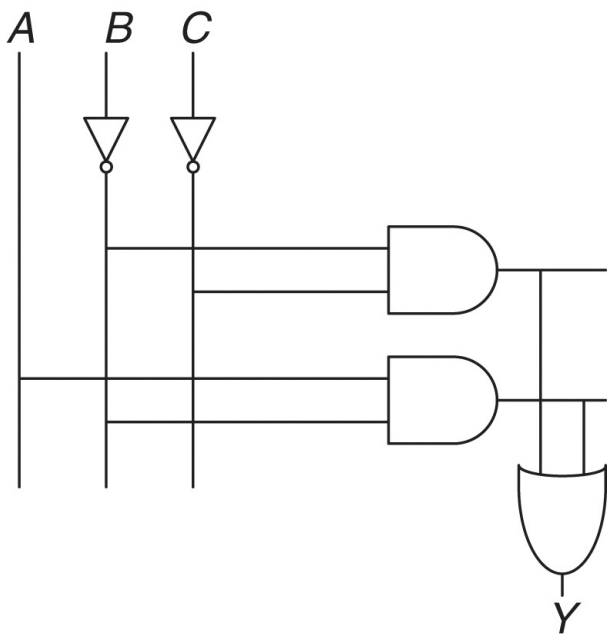
Schematic: A diagram of a digital circuits with elements (gates) and the wires that connect them together

<u>Example Boolean Eq</u>

Y = AB’ + B’C’

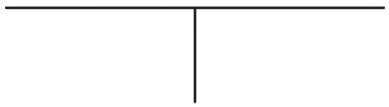
<u>Schematic</u>

1. Inputs are on the left (or top) side
2. Outputs are on the right
3. Gates flow from left to right
4. Use straight wires
5. Wires connect at a T junction
6. A dot where wires cross indicates a connection

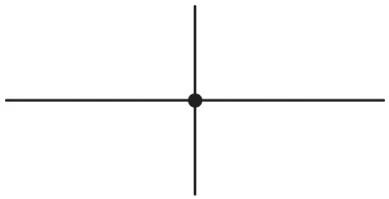


Rules for Connecting Wires

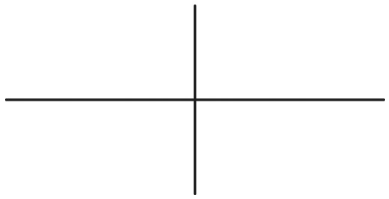
wires connect
at a T junction



wires connect
at a dot



wires crossing
without a dot do
not connect



Schematic: Happiness Detector

Specification: The students are back on campus. They are **not** *happy* if there is a *homework* deadline, or *Badger & Co.* is closed. Design a circuit that will output 1 only if students are *happy*.

Truth Table

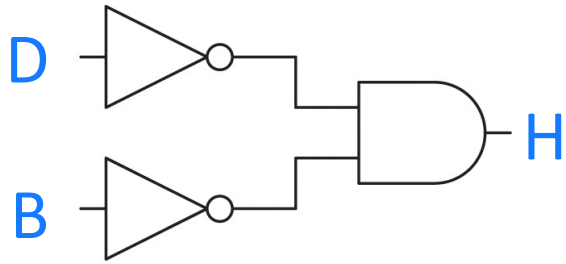
D	B	H
0	0	1
0	1	0
1	0	0
1	1	0

Boolean Eq

H = D'B'

H = (D)' AND (B)'

Logic Gate Implementation



Schematic: Happiness Detector

Specification: The students are back on campus. They are **not** *happy* if there is a *homework* deadline, or *Badger & Co.* is closed. Design a circuit that will output 1 only if students are *happy*.

Truth Table

D	B	H
0	0	1
0	1	0
1	0	0
1	1	0

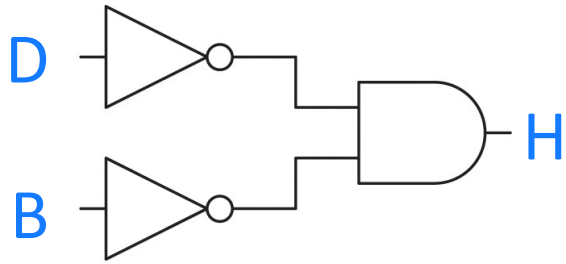
Which (monolithic) gate is this?

Boolean Eq

H = D'B'

H = (D)' AND (B)'

Logic Gate Implementation



Schematic: Happiness Detector

Specification: The students are back on campus. They are **not** *happy* if there is a *homework* deadline, or *Badger & Co.* is closed. Design a circuit that will output 1 only if students are *happy*.

Truth Table

D	B	H
0	0	1
0	1	0
1	0	0
1	1	0

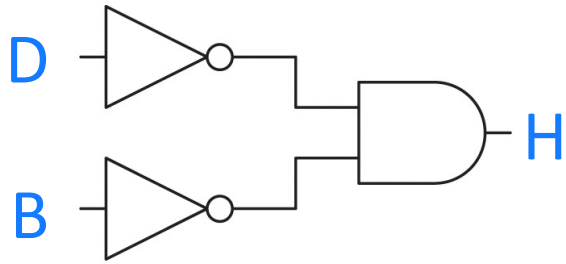
Boolean Eq

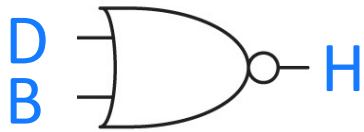
H = D'B'

H = (D)' AND (B)'

Which (monolithic) gate is this? ***Answer: NOR gate***

Logic Gate Implementation





Multiplexer: T. Table + Eq

Specification: Design a circuit with three inputs: D_0, D_1, select (S); and one output (Y). The output is D_0 if select is 0, and D_1 if select is 1.

$$Y = S'D_1'D_0 + S'D_1D_0 + SD_1D_0' + SD_1D_0$$

$$Y = S'D_0\underbrace{(D_1' + D_1)}_{=1} + SD_1\underbrace{(D_0' + D_0)}_{=1}$$

$$Y = S'D_0(1) + SD_1(1)$$

$$Y = S'D_0 + SD_1$$

Truth Table

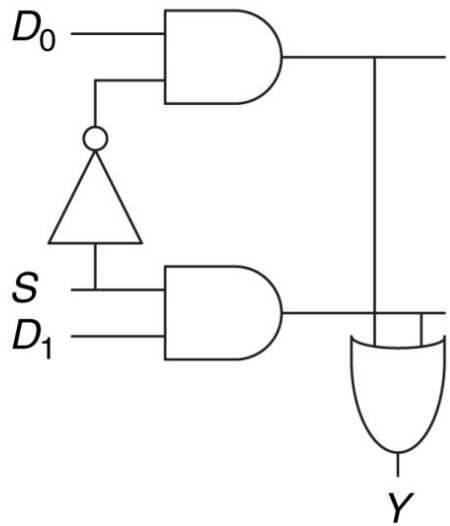
S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Multiplexer: Gate-Level Schematic

Specification: Design a circuit with three inputs: D_0, D_1, select (S); and one output (Y). The output is D_0 if select is 0, and D_1 if select is 1.

$Y = S'D_0 + SD_1$

Gate-Level Schematic



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

Half Adder

Specification: Design a circuit that adds two binary variables: A and B. The circuit has two outputs: sum and carry-out (C_{out}).

Truth Table

A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

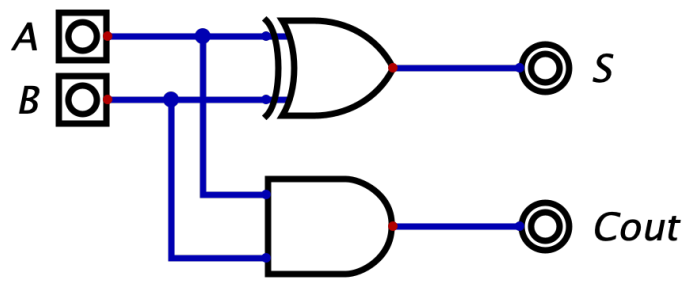
Boolean Eq

$S = A'B + AB'$

$S = A \oplus B$

$\boxed{C_{out} = AB}$

Schematic



Full Adder: T. Table + Eq

Specification: Design a circuit that adds two binary variables: A and B. The circuit has two outputs: sum and carry-out (C_{out}).

$S = C_{in}'A'B + C_{in}'AB' + C_{in}A'B' + C_{in}AB$

$C_{out} = C_{in}'AB + C_{in}A'B + C_{in}AB' + C_{in}AB$

<u>Simplification via Boolean algebra</u>

$S = A \oplus B \oplus C_{in}$

$C_{out} = C_{in}(A \oplus B) + AB$

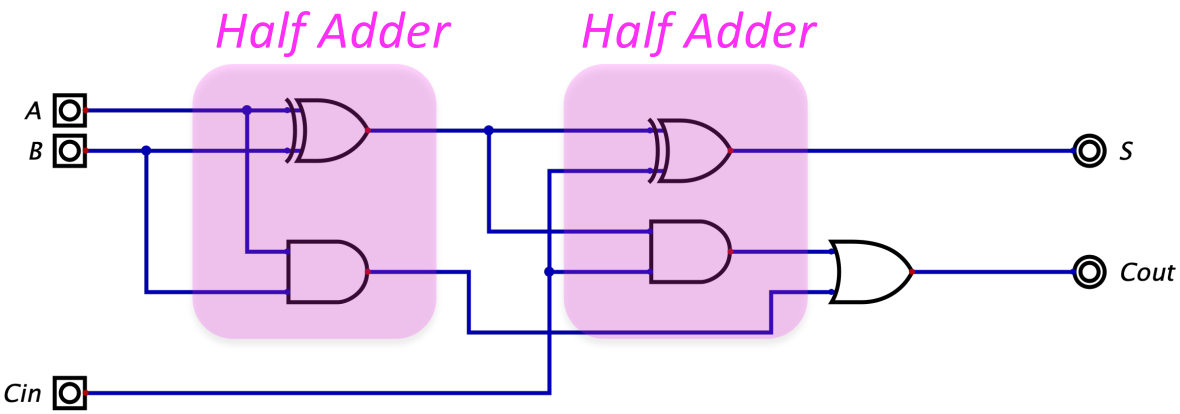
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full Adder: Schematic

Specification: Design a circuit that adds two binary variables: A and B. The circuit has two outputs: sum and carry-out (C_{out}).

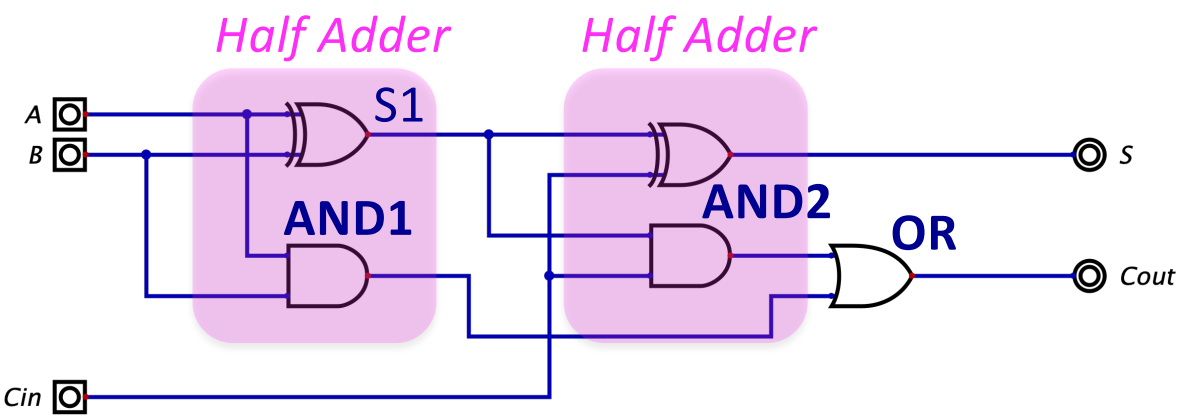
$S = A \oplus B \oplus C_{in}$

$C_{out} = C_{in}(A \oplus B) + AB$



C_{in}	A	B	C_{out}	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

Full Adder = Two Half Adders



What is **AND1** doing?

- Computes the carry out from A + B (call it S1)

What is **AND2** doing?

- Computes the carry out from S1 + C_{in}

What is the **OR** gate doing?

- C_{out} is 1 if either the output of AND1 is 1 or the output of AND2 is 1
- What does the truth table reveal about C_{out}?

C_{in}	A	B	C_{out}	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