


CSE 390 B Spring 2020

# Boolean Logic & Skills

Fundamentals of Circuits, Hardware Simulation, Skills  
Inventory, Project 1 Demo

*Significant material adapted from [www.nand2tetris.org](http://www.nand2tetris.org). © Noam Nisan and Shimon Schocken.*

# Agenda

- ❖ **Morning Warm-up Question** 
- ❖ **Let's Get Organized!**
  - What's included in your at-home study area?
- ❖ **Boolean Logic**
  - What is Boolean Logic?
  - Boolean Function Synthesis
  - Hardware Description Language
- ❖ **Project 1**
  - Demo
  - Multi-Bit Buses

# Morning Warm-Up Question


Name

Preferred Pronouns

*Day 4 of online Spring quarter...*

*What is one thing that has maybe surprised you / weren't expecting in navigating online classes so far?*

# Agenda

- ❖ Morning Warm-up Question 
- ❖ **Let's Get Organized!**
  - **What's included in your at-home study area?**
- ❖ Boolean Logic
  - What is Boolean Logic?
  - Boolean Function Synthesis
  - Hardware Description Language
- ❖ Project 1
  - Demo
  - Multi-Bit Buses

# BREAKOUTS!

We'll see how this goes...


# Breakout groups

In your small groups...

Talk about what you think you'll need in setting up your at-home study area.

- ❖ What does that environment look like?
- ❖ What tools, resources, and/or vibes does your study area require?
- ❖ What might be a barrier in setting up your ideal at home study environment?

# Agenda

- ❖ Morning Warm-up Question 
- ❖ Let's Get Organized!
  - What's included in your at-home study area?
- ❖ **Boolean Logic**
  - **What is Boolean Logic?**
  - Boolean Function Synthesis
  - Hardware Description Language
- ❖ Project 1
  - Demo
  - Multi-Bit Buses

# Boolean Values

- A binary choice: true or false
- You've seen these as a type in Java
  - Boolean Logic is a system built entirely from these values and operations between them



“Off”

False

0



“On”

True

1



# What is Boolean Logic?

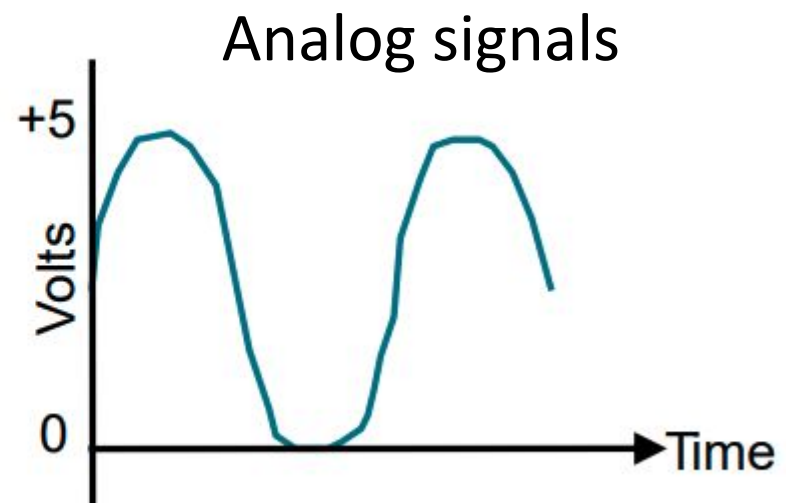
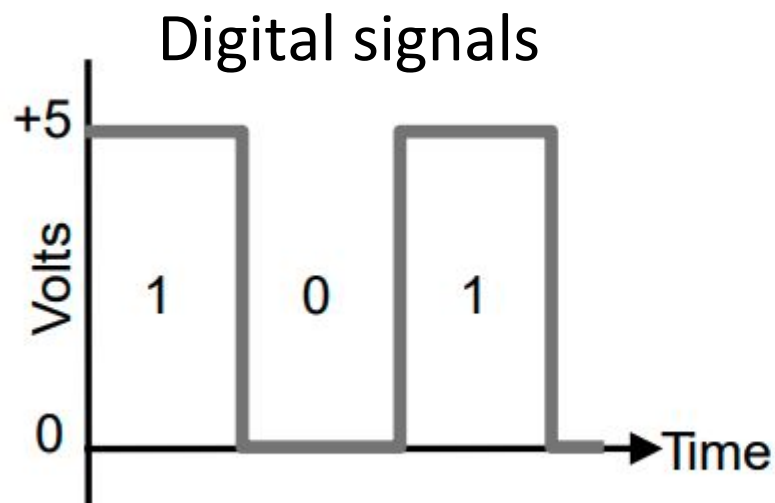
- A system of reasoning built from these values and operations between them
  - Similar to the numerical algebra we're used to

$$2 + 5 = 7$$

True And False = False  
(1 And 0 = 0)

# Aside: Why Study Boolean Logic?

- In reality, physical wires in a computer *could* have any number of volts (analog)
- We *choose* to use only 2 values in hardware
  - Reduces errors in hardware significantly, easier to reason about!



# Boolean Operations

- Can use simple logical operations to combine booleans
  - **Truth table:** Writing out *every possible* set of inputs and the corresponding output of the operation

$x$  And  $y$

$x \wedge y$

$x$	$y$	And
0	0	0
0	1	0
1	0	0
1	1	1

$x$  Or  $y$

$x \vee y$

$x$	$y$	Or
0	0	0
0	1	1
1	0	1
1	1	1

Not( $x$ )

$\neg x$

$x$	Not
0	1
1	0

# Boolean Expressions

- How do we evaluate an expression?
  - Apply the truth tables over and over!

Not(0 Or (1 And 1)) =



Not(0 Or 1) =



Not( 1 ) =



0

# Boolean Functions

- We can define our own boolean functions
  - All we need are inputs and outputs!

$$f(x, y, z) = (x \text{ And } y) \text{ Or } (\text{Not}(x) \text{ And } z)$$

<i>x</i>	<i>y</i>	<i>z</i>	<i>f</i>
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

# Boolean Functions

- We can define our own boolean functions
  - All we need are inputs and outputs!

$$f(x, y, z) = (x \text{ And } y) \text{ Or } (\text{Not}(x) \text{ And } z)$$

x	y	z	f
0	0	0	
0	0	1	1
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

$(0 \text{ And } 0) \text{ Or } (\text{Not}(0) \text{ And } 1) =$   
 $0 \text{ Or } (1 \text{ And } 1) =$   
 $0 \text{ Or } 1 = 1$

# Boolean Functions

- We can define our own boolean functions
  - All we need are inputs and outputs!

$$f(x, y, z) = (x \text{ And } y) \text{ Or } (\text{Not}(x) \text{ And } z)$$

} formula

<i>x</i>	<i>y</i>	<i>z</i>	<i>f</i>
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

} truth table

# Boolean Identities

- $(x \text{ And } y) = (y \text{ And } x)$
  - $(x \text{ Or } y) = (y \text{ Or } x)$
- } Commutativity
- $(x \text{ And } (y \text{ And } z)) = ((x \text{ And } y) \text{ And } z)$
  - $(x \text{ Or } (y \text{ Or } z)) = ((x \text{ Or } y) \text{ Or } z)$
- } Associativity
- $(x \text{ And } (y \text{ Or } z)) = (x \text{ And } y) \text{ Or } (x \text{ And } z)$
  - $(x \text{ Or } (y \text{ And } z)) = (x \text{ Or } y) \text{ And } (x \text{ Or } z)$
- } Distributive Property
- $\text{Not}(x \text{ And } y) = \text{Not}(x) \text{ Or } \text{Not}(y)$
  - $\text{Not}(x \text{ Or } y) = \text{Not}(x) \text{ And } \text{Not}(y)$
- } De Morgan's Laws

(... and many others!)



# Working with Boolean Algebra

## Option 1: Simplify using identities

**Current Expression:**

**Apply Identity:**

$\text{Not}(\text{Not}(x) \text{ And } \text{Not}(x \text{ Or } y)) =$

De Morgan's Law

$\text{Not}(\text{Not}(x) \text{ And } (\text{Not}(x) \text{ And } \text{Not}(y))) =$

Associativity

$\text{Not}((\text{Not}(x) \text{ And } \text{Not}(x)) \text{ And } \text{Not}(y)) =$

Idempotency

$\text{Not}(\text{Not}(x) \text{ And } \text{Not}(y)) =$

De Morgan's Law

$\text{Not}(\text{Not}(x)) \text{ Or } \text{Not}(\text{Not}(y)) =$

Double Negation

$x \text{ Or } y$

# Working with Boolean Algebra

**Option 2: Use truth table to list all possible cases, then look for a simplified match**

$\text{Not}(\text{Not}(x) \text{ And } \text{Not}(x \text{ Or } y)) =$




x	y	Or
0	0	0
0	1	1
1	0	1
1	1	1



$x \text{ Or } y$

# Agenda

- ❖ Morning Warm-up Question 
- ❖ Let's Get Organized!
  - What's included in your at-home study area?
- ❖ **Boolean Logic**
  - What is Boolean Logic?
  - **Boolean Function Synthesis**
  - Hardware Description Language
- ❖ Project 1
  - Demo
  - Multi-Bit Buses

# Boolean Expression $\rightarrow$ Truth Table

- We've seen how to build a truth table from an expression
  - Simply evaluate expression on all possible inputs

$$f(x, y, z) = (x \text{ And } y) \text{ Or } (\text{Not}(x) \text{ And } z)$$



$x$	$y$	$z$	$f$
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

# Boolean Expression $\leftarrow$ Truth Table

- Can we do it in reverse?

$$f(x, y, z) = (x \text{ And } y) \text{ Or } (\text{Not}(x) \text{ And } z)$$



$x$	$y$	$z$	$f$
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

# Boolean Expression $\leftarrow$ Truth Table

- We can describe a single row with And & Not

$x$	$y$	$z$	$f$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Not( $x$ ) And Not( $y$ ) And Not( $z$ )

# Boolean Expression $\leftarrow$ Truth Table

- We can describe a single row with And & Not
  - Here “describe” means creating an expression that is true in that case and false in all others

$x$	$y$	$z$	$f$	
0	0	0	1	1
0	0	1	0	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

Not( $x$ ) And Not( $y$ ) And Not( $z$ )

# Boolean Expression $\leftarrow$ Truth Table

- We can describe a single row with And & Not
  - Here “describe” means creating an expression that is true in that case and false in all others

$x$	$y$	$z$	$f$		
0	0	0	1	1	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0

Not( $x$ ) And Not( $y$ ) And Not( $z$ )

Not( $x$ ) And  $y$  And Not( $z$ )



# Boolean Expression $\leftarrow$ Truth Table

- We can describe a single row with And & Not
  - Here “describe” means creating an expression that is true in that case and false in all others

$x$	$y$	$z$	$f$			
0	0	0	1	1	0	0
0	0	1	0	0	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	0
1	0	0	1	0	0	1
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	0	0

Not( $x$ ) And Not( $y$ ) And Not( $z$ )

Not( $x$ ) And  $y$  And Not( $z$ )

$x$  And Not( $y$ ) And Not( $z$ )

# Boolean Expression $\leftarrow$ Truth Table

- All we need to describe the overall function is to combine these expressions using Or!

$x$	$y$	$z$	$f$			
0	0	0	1	1	0	0
0	0	1	0	0	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	0
1	0	0	1	0	0	1
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	0	0

$f =$

( Not( $x$ ) And Not( $y$ ) And Not( $z$ ) ) Or

( Not( $x$ ) And  $y$  And Not( $z$ ) ) Or

(  $x$  And Not( $y$ ) And Not( $z$ ) )

# Boolean Expression $\leftarrow$ Truth Table

- All we need to describe the overall function is to combine these expressions using Or!

$x$	$y$	$z$	$f$			
0	0	0	1	1	0	0
0	0	1	0	0	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	0
1	0	0	1	0	0	1
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	0	0

$f =$

$$\begin{aligned}
 & \left( \text{Not}(x) \text{ And Not}(y) \text{ And Not}(z) \right) \text{ Or} \\
 & \left( \text{Not}(x) \text{ And } y \text{ And Not}(z) \right) \text{ Or} \\
 & \left( x \text{ And Not}(y) \text{ And Not}(z) \right)
 \end{aligned}$$

Then simplify as needed:  
 $\text{Not}(z) \text{ And } (\text{Not}(x) \text{ Or Not}(y))$

# Boolean Functions Theorem

- Every function has a truth table, so we can do this for any function. Therefore:

## Theorem

Any function can be represented by a combination of And, Not, & Or.

$\text{Not}(z) \text{ And } (\text{Not}(x) \text{ Or } \text{Not}(y))$

# Boolean Functions Theorem

- Every function has a truth table, so we can do this for any function. Therefore:

## Theorem

Any function can be represented by a combination of And, Not, & Or.

$\text{Not}(z) \text{ And } (\text{Not}(x) \text{ Or } \text{Not}(y))$

- Can we do better?

# Refining Our Theorem

## Theorem

Any function can be represented by a combination of And, Not, & Or.

## Example

Not(z) And (Not(x) Or Not(y))

Or can be represented by And & Not:

$$\mathbf{x \text{ Or } y = \text{Not}(\text{Not}(x) \text{ And } \text{Not}(y))}$$

*(Thanks DeMorgan's Laws!)*

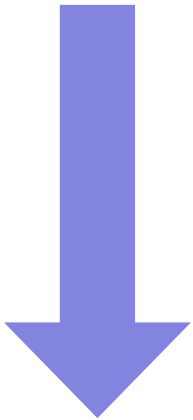
# Refining Our Theorem

## Theorem

Any function can be represented by a combination of And, Not, & Or.

## Example

$\text{Not}(z) \text{ And } (\text{Not}(x) \text{ Or } \text{Not}(y))$



Or can be represented by And & Not:

$$x \text{ Or } y = \text{Not}(\text{Not}(x) \text{ And } \text{Not}(y))$$

*(Thanks DeMorgan's Laws!)*

## Theorem

Any function can be represented by a combination of And & Not.

## Example

$\text{Not}(z) \text{ And } \text{Not}(x \text{ And } y)$

# The Nand Operation

- “Nand” stands for “Negated And”
  - The And operation, but every output is negated

<i>x</i>	<i>y</i>	And
0	0	0
0	1	0
1	0	0
1	1	1

<i>x</i>	<i>y</i>	Nand
0	0	1
0	1	1
1	0	1
1	1	0

$$(x \text{ Nand } y) = \text{Not}(x \text{ And } y)$$



# The Nand Operation

- “Nand” stands for “Negated And”
  - The And operation, but every output is negated

<i>x</i>	<i>y</i>	And
0	0	0
0	1	0
1	0	0
1	1	1

<i>x</i>	<i>y</i>	Nand
0	0	1
0	1	1
1	0	1
1	1	0

$$(x \text{ Nand } y) = \text{Not}(x \text{ And } y)$$

$$\text{Not } x = (x \text{ Nand } x)$$

$$x \text{ And } y = \text{Not}(x \text{ Nand } y)$$

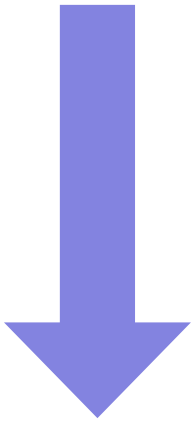
# Refining Our Theorem Even More

## Theorem

Any function can be represented by a combination of And & Not.

## Example

Not(z) And Not(x And y)



Not & And can be represented with Nand:

**Not x = (x Nand x)**

**x And y = Not(x Nand y)**

## Theorem

Any function can be represented solely by Nand operations.

## Example

((z Nand z) Nand (x Nand y)) Nand  
((z Nand z) Nand (x Nand y))

# Refining Our Theorem Even More

## Theorem

Any function can be represented by a combination of And & Not.

## Example

$\text{Not}(z) \text{ And } \text{Not}(x \text{ And } y)$

Not & And can be represented with Nand:

$\text{Not } x = (x \text{ Nand } x)$


$\text{And } (x, y) = \text{Not}(x \text{ Nand } y)$

## Theorem

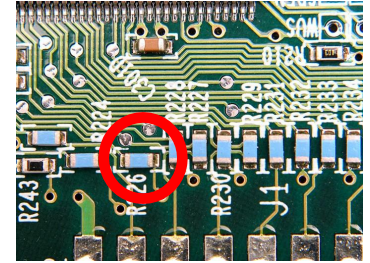
Any function can be represented solely by Nand operations.

$\text{And } (x, y) = \text{Not}(\text{Not } x \text{ Nand } (x \text{ Nand } y)) \text{ Nand } (\text{Not } y \text{ Nand } (x \text{ Nand } y))$

# Agenda

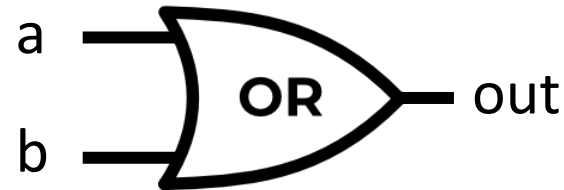
- ❖ Morning Warm-up Question 
- ❖ Let's Get Organized!
  - What's included in your at-home study area?
- ❖ **Boolean Logic**
  - What is Boolean Logic?
  - Boolean Function Synthesis
  - **Hardware Description Language**
- ❖ Project 1
  - Demo
  - Multi-Bit Buses

# Logic Gates



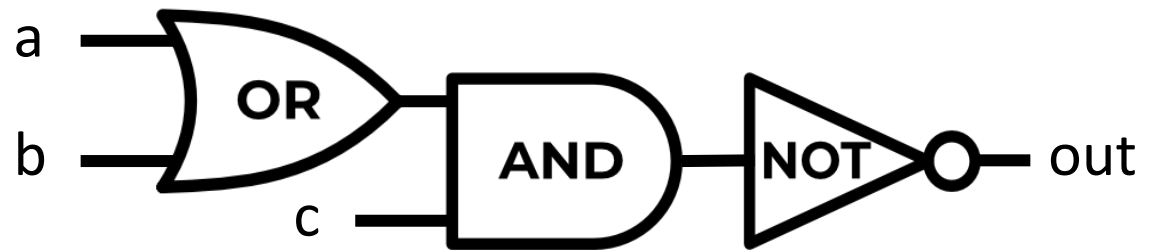
- Logic gates implement these Boolean logic operations in hardware

(a Or b)



- Combining operations means wiring logic gates together

Not ((a Or b) And c)



# Building a Logic Gate

- Specification: We want a new logic gate called “Xor”
  - Outputs 1 when one input or the other is 1, but *not* both.

a	b	Xor
0	0	0
0	1	1
1	0	1
1	1	0



- Implementation: Combine existing logic gates!
  - For convenience, assume we already have And, Or, & Not

# Building a Logic Gate

a	b	Xor
0	0	0
0	1	1
1	0	1
1	1	0

**Xor =**

( Not(*a*) And *b* ) Or

( *a* And Not(*b*) )

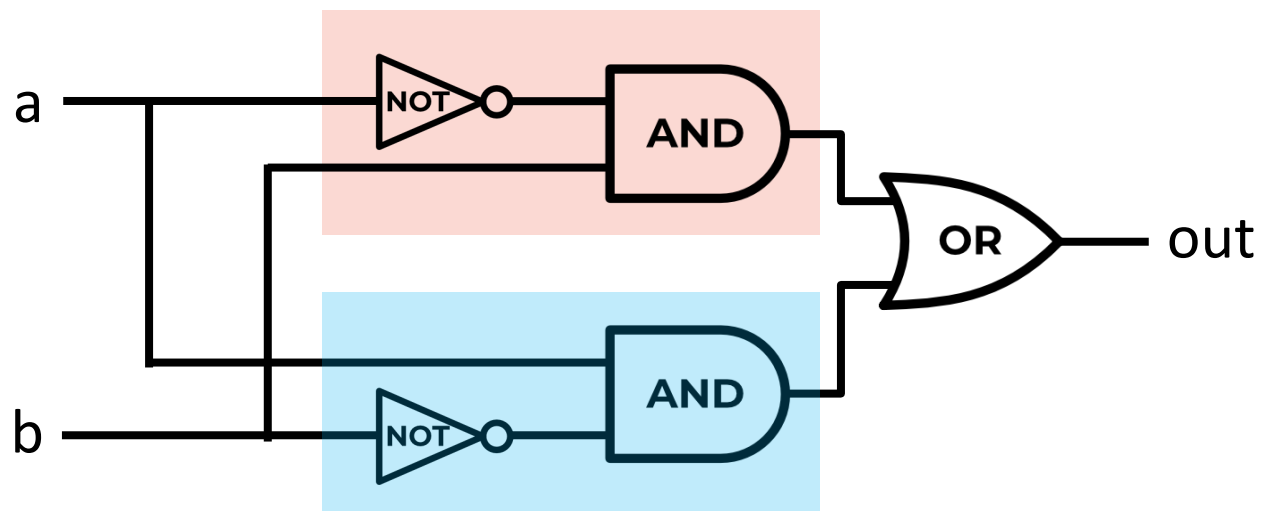
# Building a Logic Gate

a	b	Xor
0	0	0
0	1	1
1	0	1
1	1	0

**Xor =**

( Not(*a*) And *b* ) Or

( *a* And Not(*b*) )





# Hardware Design Language (.hdl)

- A programming language to specify hardware **components** and how they're **connected**
- There are many Hardware Design Languages in use today (e.g. VHDL, Verilog, SystemVerilog)
  - In this course, we'll use a simple one, just called "HDL".
- Unlike Java, HDL is a **declarative** language
  - The order of statements doesn't matter
  - Describes a physical system

# Hardware Design Language (.hdl)

- Makeup of an HDL file:

```
/**  
 * Exclusive-or gate:  
 * out = not (a == b)  
 */
```

```
CHIP Xor {  
  IN a, b;  
  OUT out;
```



Comments describing  
expected behavior



Names of inputs and  
outputs

**INTERFACE**

---

```
PARTS:  
  // Put your code here:  
  
}
```



Components that  
make it up

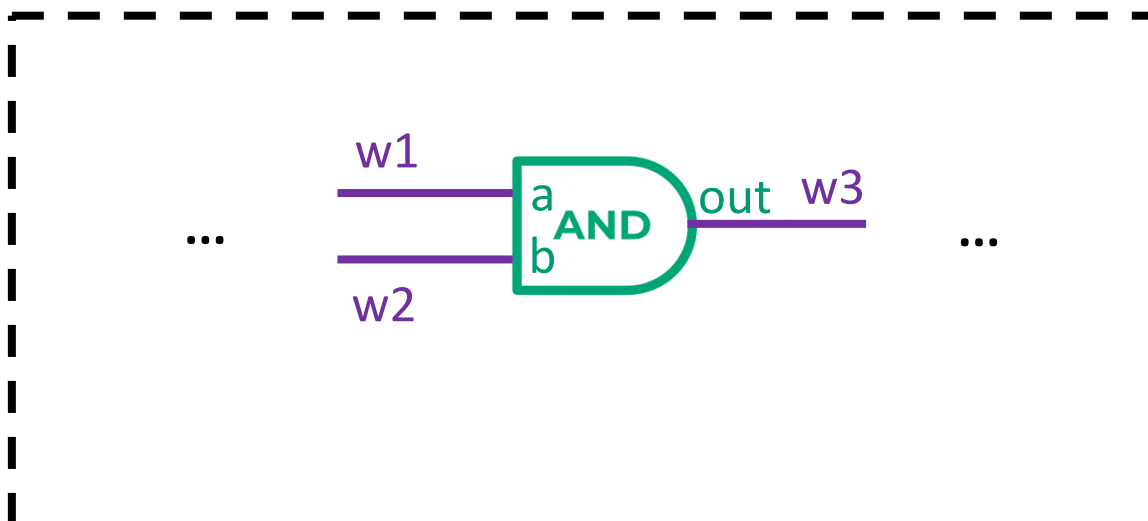
**IMPLEMENTATION**

# Hardware Design Language (.hdl)

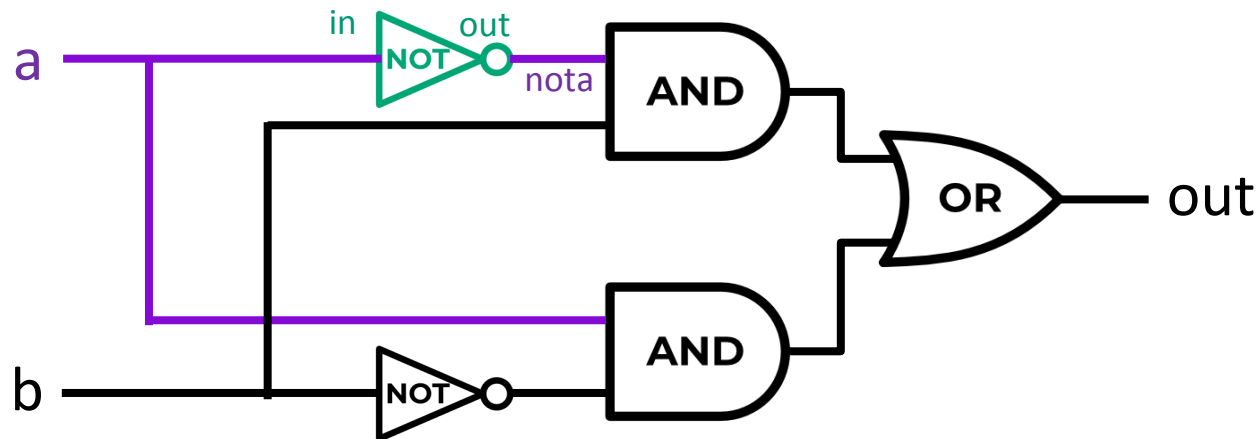
- Using a subcomponent: (in this case, an And gate)

```
CHIP Xor {  
    ...  
    PARTS:  
    And (a=w1, b=w2, out=w3);  
}
```

Xor

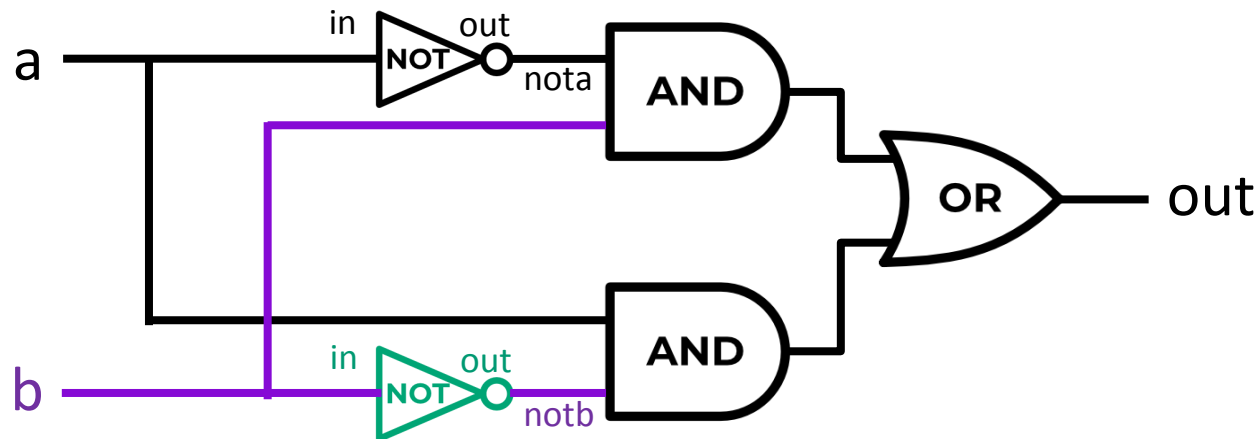


# Implementing Xor in HDL



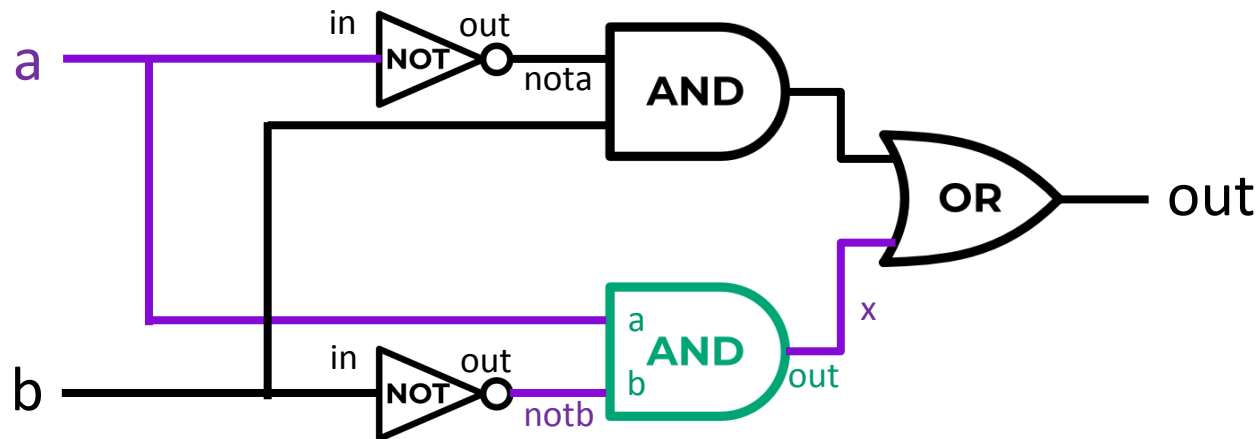
```
CHIP Xor {  
  IN a, b;  
  OUT out;  
  
  PARTS:  
    Not (in=a, out=nota);  
  
}
```

# Implementing Xor in HDL



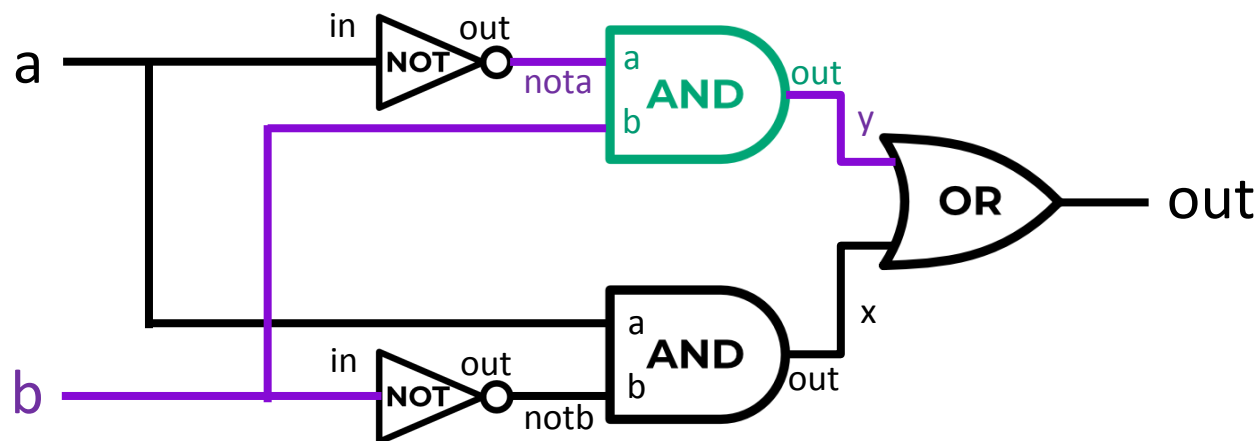
```
CHIP Xor {  
  IN a, b;  
  OUT out;  
  
  PARTS:  
    Not (in=a, out=nota);  
    Not (in=b, out=notb);  
  
}
```

# Implementing Xor in HDL



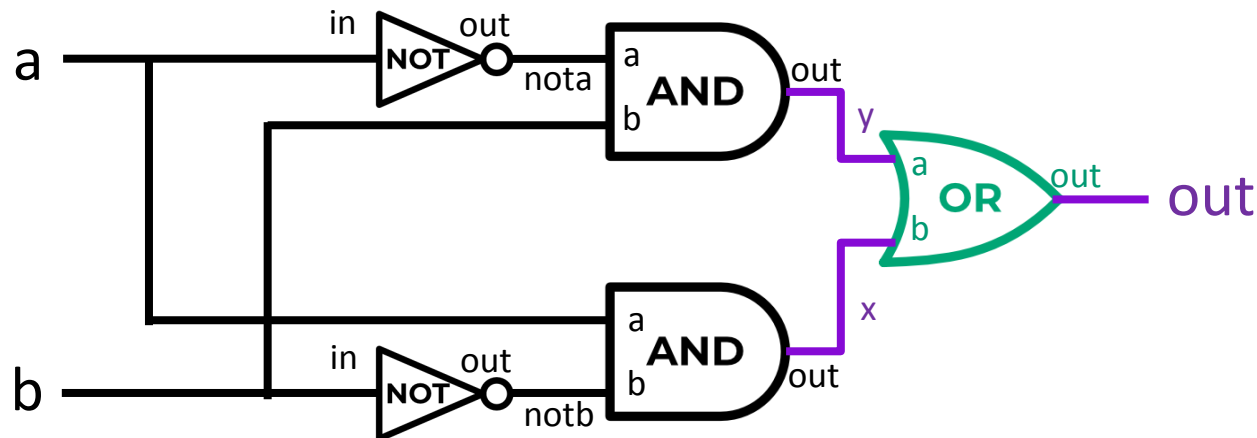
```
CHIP Xor {  
  IN a, b;  
  OUT out;  
  
  PARTS:  
    Not (in=a, out=nota);  
    Not (in=b, out=notb);  
    And (a=a, b=notb, out=x);  
  
}
```

# Implementing Xor in HDL



```
CHIP Xor {  
  IN a, b;  
  OUT out;  
  
  PARTS:  
    Not (in=a, out=nota);  
    Not (in=b, out=notb);  
    And (a=a, b=notb, out=x);  
    And (a=nota, b=b, out=y);  
  
}
```


# Implementing Xor in HDL



```
CHIP Xor {  
  IN a, b;  
  OUT out;  
  
  PARTS:  
    Not (in=a, out=nota);  
    Not (in=b, out=notb);  
    And (a=a, b=notb, out=x);  
    And (a=nota, b=b, out=y);  
    Or (a=x, b=y, out=out);  
}
```




# Agenda

- ❖ Morning Warm-up Question 
- ❖ Let's Get Organized!
  - What's included in your at-home study area?
- ❖ Boolean Logic
  - What is Boolean Logic?
  - Boolean Function Synthesis
  - Hardware Description Language
- ❖ **Project 1**
  - **Demo**
  - Multi-Bit Buses

# Project 1 Demo

Editing HDL, Simulator Tools, Using Built-In Chips

# Agenda

- ❖ Morning Warm-up Question 
- ❖ Let's Get Organized!
  - What's included in your at-home study area?
- ❖ Boolean Logic
  - What is Boolean Logic?
  - Boolean Function Synthesis
  - Hardware Description Language
- ❖ **Project 1**
  - Demo
  - **Multi-Bit Buses**

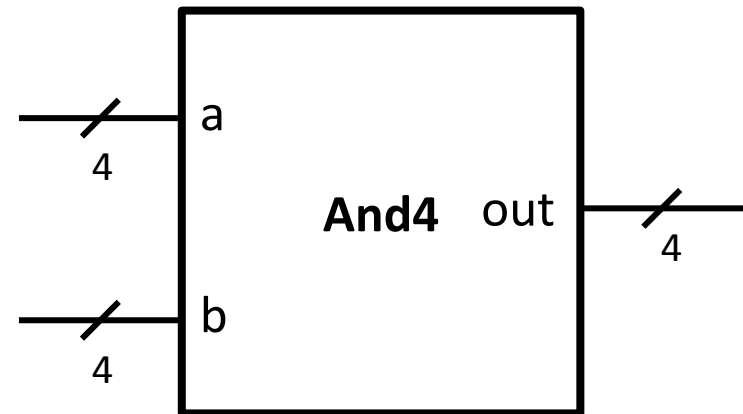
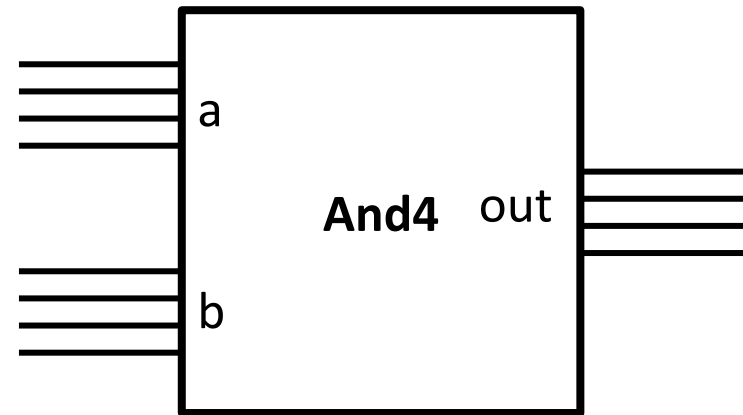
# Multi-Bit Buses in HDL

- Sometimes it's useful to manipulate many wires as a group
- For the programmer's convenience, we think of them as a single entity
  - Called a “bus”
- Most HDLs include a way to specify buses
  - Including ours! 😊

# Multi-Bit Buses in HDL

```
/**
 * Bit-wise And of two 4-bit inputs
 */
CHIP And4 {
    IN a[4], b[4];
    OUT out[4];

    PARTS:
    And (a=a[0], b=b[0], out=out[0]);
    And (a=a[1], b=b[1], out=out[1]);
    And (a=a[2], b=b[2], out=out[2]);
    And (a=a[3], b=b[3], out=out[3]);
}
```



# Project 1

## PART I: Study Skills Inventory

- Self-assessing your skill level in various study practices and habits.

## PART II: Boolean Logic

- If you've cloned your repo, you have everything you need to get started on project 1!

## PART III: Boolean Logic Reflection

- Reflecting on what your experience was like in working through the Boolean Logic project.

**DUE NEXT TUESDAY 11:59PM**

Don't forget to bring your "paper reactions" too :)