CSE 390B, Spring 2023

Building Academic Success Through Bottom-Up Computing

Study Environment & Boolean Logic

Study Environment Discussion, Boolean Logic and Functions, Foundational Logic Gates, Hardware Descriptive Languages

Checking in on Project 1

- Remember to double-check your submission on GitLab
  - Navigate to GitLab, open tags, and verify that the associated commit includes your expected changes
- How has Project 1 been coming along?
- What questions do you have about Project 1?

Lecture Outline

- **Study Environment Discussion**

- Boolean Logic and Functions
  - Boolean Expressions, Circuit Diagrams, Truth Tables
  - Boolean Function Synthesis Strategy

- Hardware Descriptive Languages (HDL)
  - HDL Syntax, **`And`** Gate Example

- Foundational Logic Gates
  - **`Nand`**, **`Not`**, **`And`**, **`Or`**, **`Xor`**, **`Mux`**, **`DMux`** Gates
  - Example of Implementing an **`Xor`** Gate in HDL

Study Environment Discussion

In groups of 3-4, discuss the following questions about study environments:

- On a typical day, what does your study environment look like? Be specific!
- What contributes to an effective study environment? Why?
  - What changes can you make to introduce some of these factors?
- What factors hinder a study environment from being effective? Why?
  - What changes can you make to remove some of these factors?

Ideal Study Environment

- Free from distractions (phones, computer tabs, etc.)
- Ensure your study environment is well organized and free from clutters
- Peaceful and quiet, possibly with some background noise
  - Listening to music may help depending on genre and the person
- Add decoration that can be an inspiration to your study environment (e.g., plants, quotes, stickers)
- Have your study material out ahead of time

Lecture Outline

- Study Environment Discussion
- **Boolean Logic and Functions**
  - **Boolean Expressions, Circuit Diagrams, Truth Tables**
  - **Boolean Function Synthesis Strategy**
- Hardware Descriptive Languages (HDL)
  - HDL Syntax, **`And`** Gate Example
- Foundational Logic Gates
  - **`Nand`**, **`Not`**, **`And`**, **`Or`**, **`Xor`**, **`Mux`**, **`DMux`** Gates
  - Example of Implementing an **`Xor`** Gate in HDL

Boolean Values

- A binary choice: **True** or **False**

- Also known as a "low" signal (false, "off," or 0) and a "high" signal (true, "on", or 1)

"Off"

False

0

"On"

True

1

Boolean Operations

- Use logical operations to combine Boolean values
  - **Truth table**: A table that lists every possible set of inputs and the corresponding output of the operation
  - Operations correspond to physical hardware gates

- Examples:

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

F = A AND B

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

F = A OR B

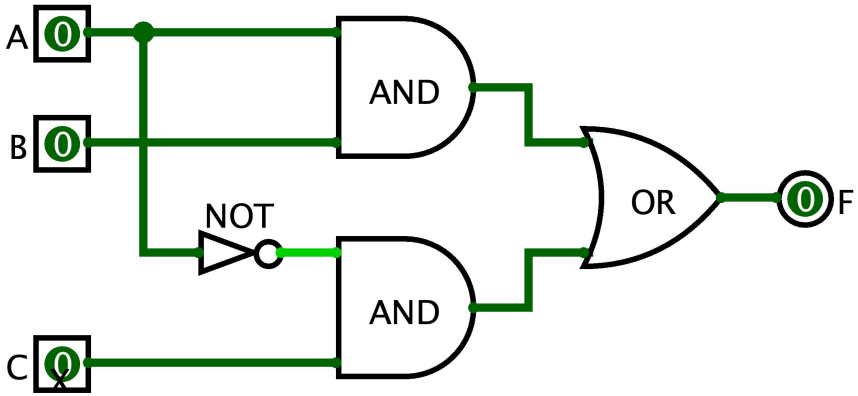
A	F
0	1
1	0

F = NOT A

Boolean Functions

- Combinations of Boolean inputs resulting in single output

- Multiple ways to specify a Boolean function:
  - Boolean expression: $F = (A \text{ AND } B) \text{ OR } (\text{NOT}(A) \text{ AND } C)$
  - Circuit diagram with logic gates:
  - Truth table:

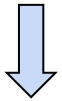


A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Boolean Expression → Truth Table

- We can build a truth table from an expression
  - Evaluate the Boolean expression on all possible inputs

$$F(A, B, C) = (A \text{ AND } B) \text{ OR } (\text{NOT}(A) \text{ AND } C)$$

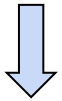


A	B	C	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Boolean Expression → Truth Table

- We can build a truth table from an expression
  - Evaluate the Boolean expression on all possible inputs

$$F(A, B, C) = (A \text{ AND } B) \text{ OR } (\text{NOT}(A) \text{ AND } C)$$

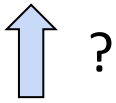


A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Boolean Expression ← Truth Table

- But can we do it in reverse?

$$F(A, B, C) = (A \text{ AND } B) \text{ OR } (\text{NOT}(A) \text{ AND } C)$$



A	B	C	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 ← Truth Table

- We can describe a single row with AND and NOT

A	B	C	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 ← Truth Table

- We can describe a single row with AND and NOT

A	B	C	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

NOT(A) AND NOT(B) AND C

Boolean Expression ← Truth Table

- We can describe a single row with AND and NOT

A	B	C	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

NOT(A) AND NOT(B) AND C

NOT(A) AND B AND C

Boolean Expression ← Truth Table

- We can describe a single row with AND and NOT

A	B	C	F	
0	0	0	0	
0	0	1	1	NOT(A) AND NOT(B) AND C
0	1	0	0	
0	1	1	1	NOT(A) AND B AND C
1	0	0	0	
1	0	1	0	
1	1	0	1	A AND B AND NOT C
1	1	1	1	

Boolean Expression ← Truth Table

- We can describe a single row with AND and NOT

A	B	C	F	
0	0	0	0	
0	0	1	1	NOT(A) AND NOT(B) AND C
0	1	0	0	
0	1	1	1	NOT(A) AND B AND C
1	0	0	0	
1	0	1	0	
1	1	0	1	A AND B AND NOT C
1	1	1	1	A AND B AND C

Boolean Expression ← Truth Table

- Then, we can combine the rows using OR operations

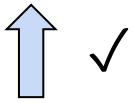
A	B	C	F	
0	0	0	0	
0	0	1	1	NOT(A) AND NOT(B) AND C
0	1	0	0	
0	1	1	1	NOT(A) AND B AND C
1	0	0	0	
1	0	1	0	
1	1	0	1	A AND B AND NOT C
1	1	1	1	A AND B AND C

F = NOT(A) AND NOT(B) AND C OR NOT(A) AND B AND C OR
A AND B AND NOT C OR A AND B AND C

Boolean Expression ← Truth Table

- But can we do it in reverse?
  - Yes, we can! The strategy we used is **Boolean Function Synthesis**

$$F(A, B, C) = (A \text{ AND } B) \text{ OR } (\text{NOT}(A) \text{ AND } C)$$



✓

A	B	C	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

Lecture 2: Study Environment & Boolean Logic

Loading...

Lecture Outline

- Study Environment Discussion
- Boolean Logic and Functions
  - Boolean Expressions, Circuit Diagrams, Truth Tables
  - Boolean Function Synthesis Strategy
- **Hardware Descriptive Languages (HDL)**
  - **HDL Syntax, `And` Gate Example**
- Foundational Logic Gates
  - **`Nand`**, **`Not`**, **`And`**, **`Or`**, **`Xor`**, **`Mux`**, **`DMux`** Gates
  - Example of Implementing an **`Xor`** Gate in HDL

Hardware Design Language (HDL)

- HDL is a programming language to specify hardware components and how they're connected
  - Another way of describing a Boolean function!

- Many Hardware Design Languages are used today
  - E.g., VHDL, Verilog, SystemVerilog
  - In this course, we'll use a simple HDL language called "HDL"

- Unlike Java, HDL is a **declarative** language. This means the following:
  - The order of statements (lines of code) doesn't matter
  - We are describing a physical system

Hardware Design Language (HDL)

- Format of an HDL file
  - File comment describes expected behavior
  - **IN** names chip inputs, **OUT** names chip outputs
  - **PARTS** specify the components (i.e., other gates) that implement the chip

```
/**
* And gate:
* out = 1 only if both a and b are 1
*/
CHIP And {
  IN a, b;
  OUT out;

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

Reusing Components

- You can (and should!) use chips you have already implemented to implement subsequent chips
- We give you one gate, **`Nand`**, to start out with
  - Implication: The entire computer you will be building will be use **`Nand`** gates as its foundation
- We also provide you with some chips you can use without implementing

HDL Component Example: AND

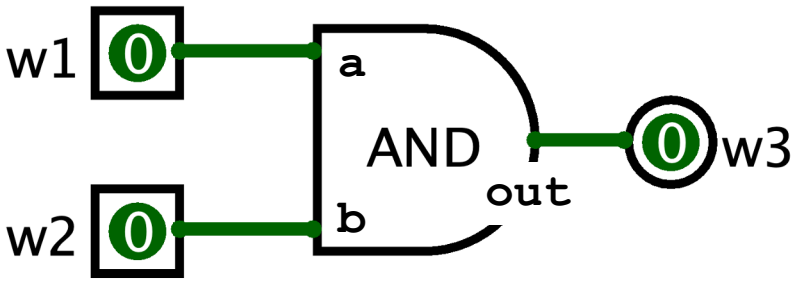
- The chip specification tells us the name of the input and output wires

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

       ...
}
```

- Goal: Implement w1 AND w2
  - HDL Syntax for using (being a client of the **And** gate):

    And(a=w1, b=w2, out=w3);
  - Equivalent circuit diagram:



Multi-bit Buses in HDL

- It can be useful to manipulate groups of wires
  - Called a "bus" of wires

- HDL provides array like syntax for manipulating buses
  - **And4** chip example:

```
/**
 * 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]);
}
```

HDL Resources

- HDL will feel unfamiliar at first, and that's okay

- Resources for helping you navigate HDL linked under the Resources page on the course website
  - HDL Survival guide
  - Appendix A (HDL Spec)
  - Chip Set Overview (to help you remember the inputs/outputs for various chips)
  - Chapter readings

Lecture Outline

- Study Environment Discussion
- Boolean Logic and Functions
  - Boolean Expressions, Circuit Diagrams, Truth Tables
  - Boolean Function Synthesis Strategy
- Hardware Descriptive Languages (HDL)
  - HDL Syntax, **`And`** Gate Example
- **Foundational Logic Gates**
  - **`Nand`, `Not`, `And`, `Or`, `Xor`, `Mux`, `DMux` Gates**
  - **Example of Implementing an `Xor` Gate in HDL**

The Foundational Building Block

- It all starts with the NAND gate

- NAND is short for "Not And"
  - The same output as the AND gate, but every output bit is negated (flipped)

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

F = A AND B

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

F = A NAND B

Building Gates From `Nand`

- Recall the **Boolean Function Synthesis** strategy from today's reading
  - We saw how we can represent any truth table in terms of three gates: **`Not`**, **`And`**, **`Or`**

- First, we can represent **`Not`** directly from **`Nand`**
  - **`Not a = a Nand a`**
- Then, we can represent **`And`** in terms of **`Not`** and **`Nand`**
  - **`a And b = Not(a Nand b)`**
- Represent **`Or`** in terms of **`Not`** and **`And`**
  - Apply De Morgan's Law
  - **`a Or b = Not(Not(a) And Not(b))`** [De Morgan's Law]

Making Decisions in Hardware

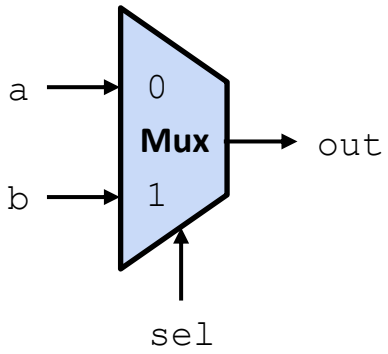
- We write if/else statements in Java with the understanding that *only one* of the branches will run
  - For example, in the following code, we expect to compute one of `a & b` or `a | b` (not both)

```
if (c == 0) {
    out = a & b;
} else {
    out = a | b;
}
```

- In hardware, the entire circuit is always executing
  - We can't "turn off" a part of a circuit based on a condition
  - Instead, we create circuits for different conditions and choose which output based on a condition instead

Decisions in Hardware: Mux Gate

- We can use a **Multiplexer (Mux) gate** to choose which singular input to output
  - Takes three inputs: `a`, `b`, and `sel`
  - If `sel == 0`, then `out = a`
  - Otherwise, `out = b`

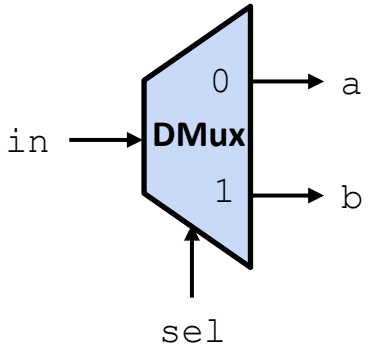


- Mux Gate Truth Table:

a	b	sel	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Decisions in Hardware: DMux Gate

- A **Demultiplexer (DMux) gate** passes one input to one of two outputs and `0` to the rest
  - Takes two inputs: `in` and `sel`
  - If `sel == 0`, then `a = in` and `b = 0`
  - Otherwise, `a = 0` and `b = in`



- DMux Gate Truth Table:

in	sel	a	b
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

Implementing an `Xor` Gate: Overview

- Let's walk through an example of building a gate that you will work on in Project 2, the **`Xor`** gate

- Together, we'll implement the **`Xor`** gate

```
/**
* Xor gate:
* out = not(a == b)
*/
CHIP Xor {
  IN a, b;
  OUT out;

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

Implementing an `Xor` Gate: Overview

- Plan of action:
  - Step 1: Create the logic operation's **truth table**
  - Step 2: Use truth table to generate a **Boolean function** using strategies we've learned, such as the Boolean Function Synthesis
  - Step 3: Convert Boolean function to **HDL**

```
/**
* Xor gate:
* out = not(a == b)
*/
CHIP Xor {
   IN a, b;
   OUT out;

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

Implementing an `Xor` Gate: Step 1

- Step 1: Create the truth table for **`Xor`**
  - Interpret the specification: **`F = NOT(A == B)`**

A	B	F
0	0	
0	1	
1	0	
1	1	

$$F = A \text{ XOR } B$$

Implementing an `Xor` Gate: Step 1

- Step 1: Create the truth table for **`Xor`**
  - Interpret the specification: **`F = NOT(A == B)`**

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

$$F = A \text{ XOR } B$$

Implementing an `Xor` Gate: Step 2

- Step 2: Use truth table to generate a Boolean function
  - Let's use the **Boolean Function Synthesis** strategy from the reading
  - Row 2 = NOT(A) AND B

A	B	F	
0	0	0	(Row 1)
0	1	1	(Row 2)
1	0	1	(Row 3)
1	1	0	(Row 4)

F = A XOR B

Implementing an `Xor` Gate: Step 2

- Step 2: Use truth table to generate a Boolean function
  - Let's use the **Boolean Function Synthesis** strategy from the reading
  - Row 2 = NOT(A) AND B
  - Row 3 = A AND NOT(B)

A	B	F	
0	0	0	(Row 1)
0	1	1	(Row 2)
1	0	1	(Row 3)
1	1	0	(Row 4)

F = A XOR B

Implementing an `Xor` Gate: Step 2

- Step 2: Use truth table to generate a Boolean function
  - Let's use the **Boolean Function Synthesis** strategy from the reading
  - Row 2 = NOT(A) AND B
  - Row 3 = A AND NOT(B)
  - F = ?

A	B	F	
0	0	0	(Row 1)
0	1	1	(Row 2)
1	0	1	(Row 3)
1	1	0	(Row 4)

F = A XOR B

Lecture 2: Study Environment & Boolean Logic

Loading...

Implementing an `Xor` gate: Step 2

- Step 2: Use truth table to generate a Boolean function
  - Let's use the Boolean function synthesis strategy from the reading
  - Row 2 = NOT(A) AND B
  - Row 3 = A AND NOT(B)
  - F = Row 2 OR Row 3
    = (NOT(A) AND B) OR (A AND NOT(B))

A	B	F	
0	0	0	(Row 1)
0	1	1	(Row 2)
1	0	1	(Row 3)
1	1	0	(Row 4)

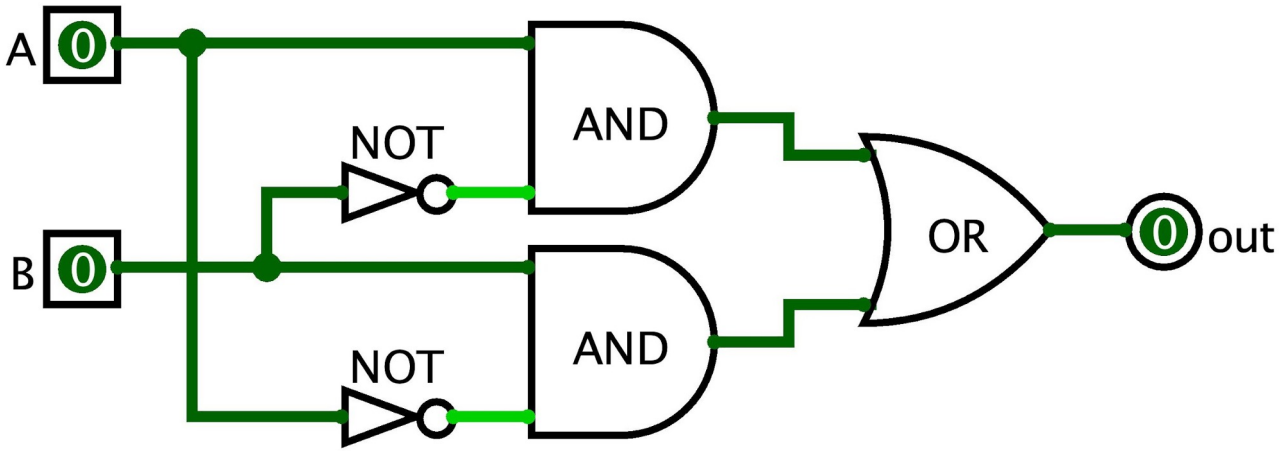
F = A XOR B

Implementing an `Xor` gate: Step 3

- Now that we have a Boolean expression, we can implement the **`Xor`** gate in HDL

- Optionally, it can help to express the Boolean expression as a circuit diagram

$$\text{A XOR B} = (\text{NOT(A) AND B}) \text{ OR } (\text{A AND NOT(B)})$$

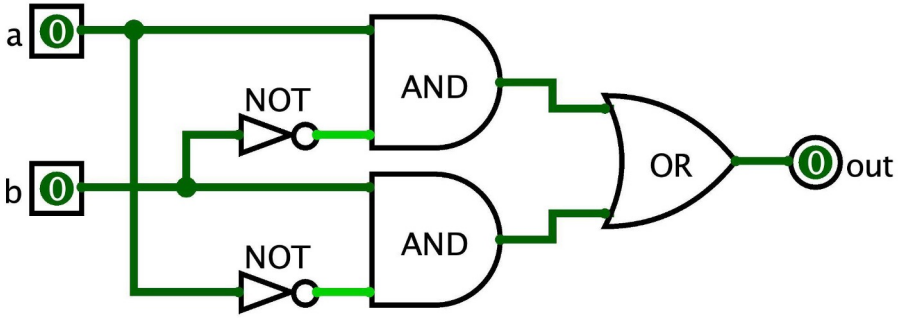


Implementing an `Xor` gate: Step 3

- Step 3: Convert Boolean function to HDL syntax
  - A XOR B = (NOT(A) AND B) OR (A AND NOT(B))
  - Assumes **`Not`**, **`And`**, and **`Or`** are already implemented
  - Note the use of intermediary wires: **`nota`**, **`notb`**, **`x`**, and **`y`**

```
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);
}
```



Project 2 Overview

- Metacognitive Component: **Study Skills Inventory**
  - Reflect on your academic behaviors, strategies, and practices
  - You will be graded on completing the form, not your responses
  - This activity is for your own benefit, and the more honest you are, the more beneficial it will be
- Technical Component: **GitLab Setup**
  - Will help prepare you for future CSE 390B projects
- Estimated time to complete: 6-8 hours
- **Project 2 is released and will be due next Thursday (4/6) at 11:59pm**

Post-Lecture 2 Reminders

- **Project 1 due tonight (3/30) at 11:59pm**
- Project 2 (Study Skills Inventory & Boolean Logic) released today, due next Thursday (4/6) at 11:59pm
- Anam has office hours after class in CSE2 121
- First Student-TA meetings starting next week
  - Your TA will be in contact with you about the first meeting