CSE 390B, Spring 2022

Building Academic Success Through Bottom-Up Computing

Study Environment, HDL, & Boolean Logic

Study Environment Discussion, Boolean Logic and Functions, Hardware Descriptive Language, Project 2 Overview

Course Staff Introductions

Eric Fan

Audrey Ma

Sean Le

Ghislaine Ng

Preston Rivas

privas@cs.washington.edu

I'm currently a second year student and I can't wait to be your TA for CSE 390B! I grew up in La Conner Washington, which is about an hour north of Seattle, but I was born in San Diego, California. When not doing school work I enjoy running, hiking, kayaking, paddle boarding and hammocking (if you know any cool spots around UW to hang a hammock let me know!). I'm also a big fan of indie music and especially love listening to Peach Pit. If you find yourself stuck don't hesitate to reach out for help! I'm always wiling to lend a hand, even if its something not related to CS/UW!

Project 1 Check-in

- The specification says Project 1 will take 30-60 minutes
  - Likely an underestimation
  - It may be taking longer for you, and that's okay
- Seeing some **GREAT** questions and answers on Ed!
- Your terminal setup will unlikely look the same as mine, and that's okay
  - Just ensure that the commands are correct
- **Double-check** your submission on GitLab!
  - Navigate to GitLab, open tags, and verify that the associated commit includes your expected changes

Lecture Outline

- **Study Environment Discussion**

- Boolean Logic and Functions
  - Boolean Expressions, Circuit Diagrams, Truth Tables
  - Boolean Function Synthesis Strategy
  - The Foundational Gate: NAND

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

- Project 2 Overview
  - **`Xor`** Implementation Example 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?

Lecture Outline

- Study Environment Discussion

- **Boolean Logic and Functions**
  - **Boolean Expressions, Circuit Diagrams, Truth Tables**
  - **Boolean Function Synthesis Strategy**
  - **The Foundational Gate: NAND**

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

- Project 2 Overview
  - **`Xor`** Implementation Example in HDL

Poll Everywhere

Vote at https://pollev.com/cse390b

Pre-lecture Reading Question: Which of the following statements is FALSE?

A. The Boolean Function Synthesis strategy works on every truth table

B. In CSE 390B, we will be asking you to write Boolean expressions that are simplified

C. The abstraction of voltages on a physical wire represents two values

D. A truth table lists every possible combination of inputs for a Boolean operation

E. We're lost…

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

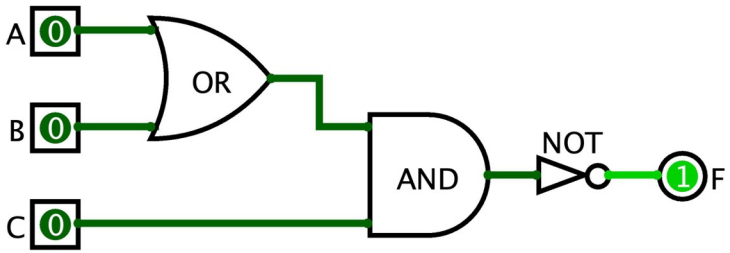
F = NOT A

Boolean Operations

- Combinations of Boolean inputs

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

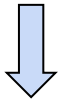
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Boolean Expression → Truth Table

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

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

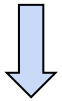


A	B	C	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Boolean Expression → Truth Table

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

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

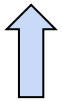


A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Boolean Expression ← Truth Table

- But can we do it in reverse?

$$F(A, B, C) = ?$$



?

A	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

- We can describe a single row with AND and NOT

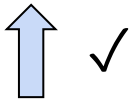
A	B	C	F	
0	0	0	0	
0	0	1	1	NOT(A) AND NOT(B) AND C
0	1	0	0	
0	1	1	1	NOT(A) AND B AND C
1	0	0	0	
1	0	1	0	
1	1	0	1	A AND B AND NOT C
1	1	1	1	A AND B AND C

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**

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



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

The Foundational Gate: Nand

- 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]

Lecture Outline

- Study Environment Discussion
- Boolean Logic and Functions
  - Boolean Expressions, Circuit Diagrams, Truth Tables
  - Boolean Function Synthesis Strategy
  - The Foundational Gate: NAND
- **Hardware Descriptive Languages (HDL)**
  - **HDL Syntax, `And` Gate Example**
- Project 2 Overview
  - `Xor` Implementation Example in HDL

Hardware Design Language (HDL)

- HDL is a programming language to specify hardware components and how they're connected
  - *Yet another* way of writing a Boolean function
- Many Hardware Design Languages are use 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 order of statements (lines of code) doesn't matter
  - We are describing a physical system

Hardware Design Language (HDL)

- Makeup of an HDL file
  - File comment describes expected behavior
  - **IN** names chip inputs, **OUT** names chip outputs
  - **PARTS** specify the components that implement the chip
    - For Project 2, this will mostly be specifying Boolean functions

```
/**
* 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 use chips you have already implemented to implement subsequent chips
- We give you one gate, **`Nand`**, to start out with
  - Implication: Your entire computer is essentially built on **`Nand`** gates
- We also give you some chips you can use without implementing
  - See project specification for more details

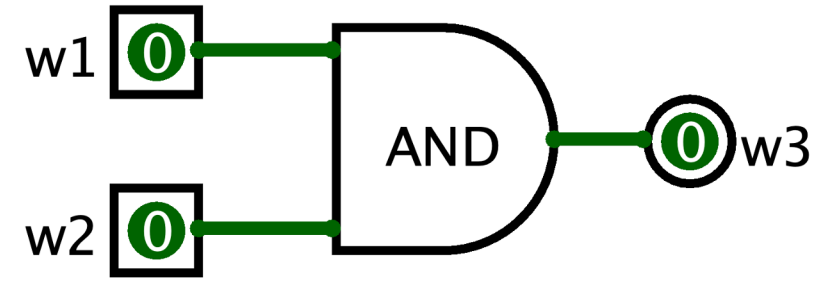
HDL Component Example: AND

- 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: **`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

- Some resources for helping you navigate HDL (these are all 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

Five-minute Break!

- Feel free to stand up, stretch, use the restroom, drink some water, review your notes, or ask questions
- We'll be back at:
- Please sign up for your first 1:1 Student-TA meeting now if you have not already (signup sheet linked here)
- Any song recommendations? Respond on Poll Everywhere at https://pollev.com/cse390b

Olmsted III, John. "The Mid-Lecture Break: When Less Is More." *Journal of Chemical Education* (1999). https://pubs.acs.org/doi/abs/10.1021/ed076p525.

Lecture Outline

- Study Environment Discussion
- Boolean Logic and Functions
  - Boolean Expressions, Circuit Diagrams, Truth Tables
  - Boolean Function Synthesis Strategy
  - The Foundational Gate: NAND
- Hardware Descriptive Languages (HDL)
  - HDL Syntax, **`And`** Gate Example
- **Project 2 Overview**
  - **`Xor` Implementation Example in HDL**

Project 2 Overview

- Part I: Study Skills Inventory
  - Self-assessing your skill level in various study practices and habits
- Part II: Boolean Logic
  - Clone your GitLab repository, write and test your code
- Part III: Project 2 Reflection
  - Reflect on your experience working through Project 2
- **Due next Thursday (4/7) at 11:59pm PDT**

Project 2 Overview

- Every logic gate in our computer can be implemented in terms of **`Nand`**
- We provide you only **`Nand`** to start Project 2
- You will build other basic gates (**`Not`**, **`And`**, etc.) in terms of **`Nand`** and then build increasingly complex gates
- Culminates to building a computer that, at its core, is entirely based on **`Nand`** gates

Project 2 Example: `Xor`

- Let's walk through an example of a gate that you will be implementing in Project 2

- 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:
}
```

Project 2 Example: `Xor`

- 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:
}
```

Project 2 Example: `Xor`

- 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

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

Project 2 Example: `Xor`

- Step 2: Use truth table to generate a Boolean function
  - Let's use the **Boolean Function Synthesis** strategy from the reading

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

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

Project 2 Example: `Xor`

- 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

Project 2 Example: `Xor`

- 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

Poll Everywhere

Vote at https://pollev.com/cse390b

What is the unsimplified Boolean expression result from performing Boolean function synthesis on $\mathrm{F} = \mathrm{A\ XOR\ B}$?

A. (A AND NOT(B)) AND (NOT(A) AND B)
B. (NOT(A) AND B) AND (A AND B)
C. (A AND B) OR (NOT(A) AND NOT(B))
D. (NOT(A) AND B) OR (A AND NOT(B))
E. We're lost…

- Row 2 = NOT(A) AND B
- Row 3 = A AND NOT(B)

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

Project 2 Example: `Xor`

- 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 XOR B = Row 2 OR Row 3
    = (NOT(A) AND B) OR (A AND NOT(B))

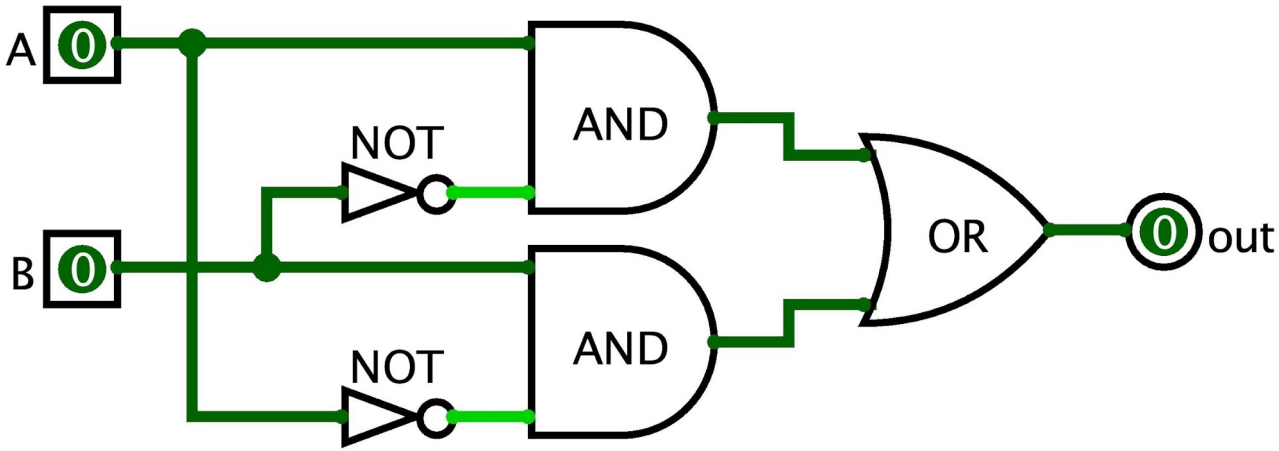
A	B	F	
0	0	0	(Row 1)
0	1	1	(Row 2)
1	0	1	(Row 3)
1	1	0	(Row 4)

F = A XOR B

Project 2 Example: `Xor`

- May be helpful to convert a Boolean expression to a circuit diagram

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

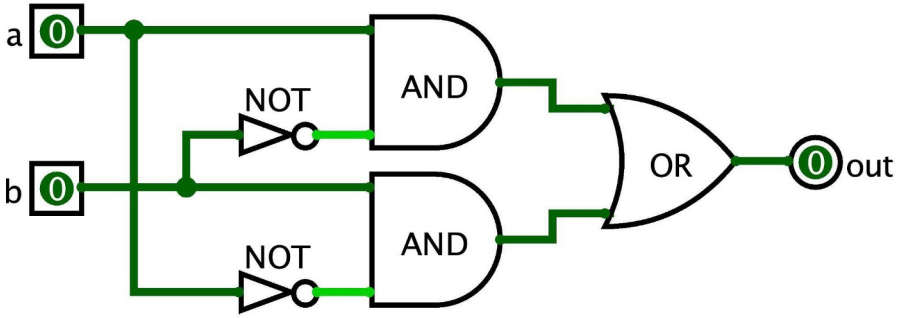


Project 2 Example: `Xor`

- 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 Demonstration

- Work through the tools you will use in Project 2
- Today's goal: Tinker with all the tools for Project 2
- Stretch goal: Implement the first gate in Project 2, **`Not`**
  - Interacting with the tools and implementing the gates will help you become familiar with the interface and address any confusion

Project 2 Group Work

- Group time to work on the first gate, **`Not`**, in Project 2

- Complete the following tasks in groups:
  1. Read the **Overview** section of the specification
  2. Estimate how much time you think it will take to finish Project 2
  3. Clone your GitLab repository (instructions in Project 1)
  4. Open the HardwareSimulator (see instructions in the **Tools** section of the specification)
  5. Dive into it! Implement and test the **`Not`** gate

Wrapping Up

- Exciting topics for Week 2!
  - Metacognitive Subject: Time Management ⏱️
  - Technical Subject: Boolean Arithmetic & ALU

- Project Reminders:
  - **Project 1 due tonight (Thursday, 3/31) at 11:59pm PDT**
  - Projects will generally be graded two days after the due date
  - Project 2 (Study Skills Inventory & Boolean Logic) released, due next Thursday (4/7) at 11:59pm PDT

- First Student-TA meetings starting this week! 🎉
  - If you haven't yet, sign up for your first meeting in the signup spreadsheet
  - Your TA will be in contact with you about the first meeting