CSE 390B, Winter 2022

Building Academic Success Through Bottom-Up Computing

Boolean Logic, Project 1 Overview

Study Environment, Circuit Fundamentals, Hardware Simulation, Project 1 Overview

Lecture Outline

- **Study Environment Discussion**
- Reading Overview and Q&A
- Hardware Design Language
- Project 1
  - Overview & Example: XOR
  - Demo
  - Group work

Study Environment Discussion

In your small groups, discuss what your ideal study environment looks like.

- What does your environment look like?
- What are factors that may impact your ideal study environment?
- What do your study spaces look like when you're having to do online learning vs. in-person learning?

Lecture Outline

- Study Environment Discussion
- **Reading Overview and Q&A**
- Hardware Design Language
- Project 1
  - Overview & Example: XOR
  - Demo
  - Group work

Boolean Values

- A binary choice: True or False
- Maps to “high” signal (true) or “low” signal (false)

“Off”

False

0

“On”

True

1

Boolean Operations

- Use simple logical operations to combine Boolean values
  - **Truth Table**: Writing out 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

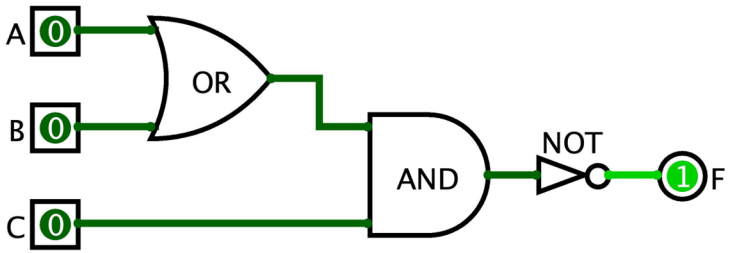
A	F
0	1
1	0

F = NOT A

Boolean Operations (cont'd)

- Combinations of Boolean values/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:

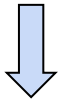


A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Boolean Expression → Truth Table

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

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

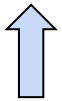


A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Boolean Expression ← Truth Table

- But can we do it in reverse?

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



?

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

- 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

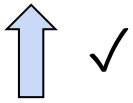
A	B	C	F	
0	0	0	0	
0	0	1	1	NOT(A) AND NOT(B) AND C
0	1	0	0	
0	1	1	1	NOT(A) AND B AND C
1	0	0	0	
1	0	1	0	
1	1	0	1	A AND B AND NOT C
1	1	1	1	A AND B AND C

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 method 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

Lecture Pre-reading Q&A

- What questions did you jot down from today's reading?
- What's something interesting that you learned? Something that surprised you?

Lecture Outline

- Study Environment Discussion
- Reading Overview and Q&A
- **Hardware Design Language**
- Project 1
  - Overview & Example: XOR
  - Demo
  - Group work

Hardware Design Language

- A programming language to specify hardware components and how they're connected
  - Yet another way of writing a Boolean function
- Many Hardware Design Languages in 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
  - The order of statements doesn't matter
  - Describes a physical system

Hardware Design Language (cont'd)

- Makeup of a 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 1, 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 essentially built on **`Nand`** gates
- We also give you some chips you can use without implementing
  - See project specification for more details

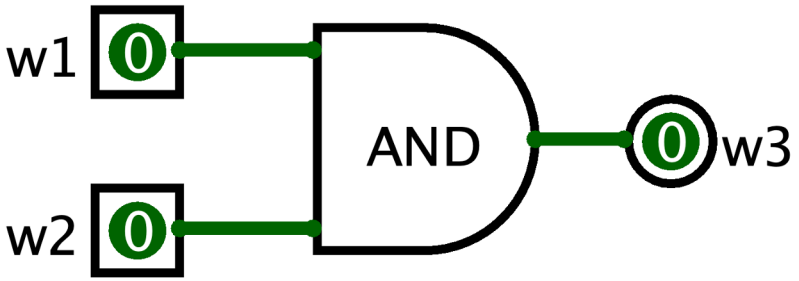
HDL Component Example: AND

- 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

Lecture Outline

- Study Environment Discussion
- Reading Overview and Q&A
- Hardware Design Language
- **Project 1**
  - **Overview & Example: XOR**
  - **Demo**
  - **Group work**

Project 1 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: Social Computing Reflection
  - Opportunity to consider how hardware connects to society
- Part IV: Project 1 Reflection
  - Reflect on your experience working through Project 1
- **Due next Thursday (1/13) at 11:59 PM PST**

Project 1 Overview (cont'd)

- It all starts with the NAND gate

- NAND is short for "Negated 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

Project 1 Overview (cont'd)

- Turns out every logic gate in our computer can be implemented in terms of **`Nand`**
- We provide you only **`Nand`** to start Project 1
- You will build other basic gates (**`Not`**, **`And`**, etc.) in terms of **`Nand`** and then increasingly complex gates in terms of other gates you implement
- Will culminate in the building of a computer entirely based on **`Nand`** gates

Building gates out of Nand

- Recall the Boolean synthesis strategy from the reading
  - Represent any truth table or Boolean function in terms of three gates: **`Not`**, **`And`**, **`Or`**

- First, we can represent **`Not`** directly from **`Nand`**
  - **`Not a = a Nand a`**

Building gates out of Nand

- Recall the Boolean synthesis strategy from the reading
  - Represent any truth table or Boolean function in terms of three gates: **`Not`**, **`And`**, **`Or`**

- First, we can represent **`Not`** directly from **`Nand`**
  - **`Not a = a Nand a`**
- Represent **`And`** in terms of **`Not`** and **`Nand`**
  - **`a And b = Not(a Nand b)`**

Building gates out of Nand

- Recall the Boolean synthesis strategy from the reading
  - Represent any truth table or Boolean function in terms of three gates: **`Not`**, **`And`**, **`Or`**

- First, we can represent **`Not`** directly from **`Nand`**
  - **`Not a = a Nand a`**
- 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]

Project 1 Example: Xor

- Let's walkthrough an example of what you'll do in Project 1

- 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 1 Example: Xor (cont'd)

- Plan:
  - Generate truth table
  - Use truth table to generate Boolean function
  - Convert Boolean function to HDL syntax

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

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

Project 1 Example: Xor (cont'd)

- Step 1: Build a truth table
  - Interpret the specification: `F = NOT(A == B)`

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

F = A XOR B

Project 1 Example: Xor (cont'd)

- 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 1 Example: Xor (cont'd)

- 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 1 Example: Xor (cont'd)

- 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/ericfan524

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

A. (A AND NOT(B)) AND (NOT(A) AND B)
B. NOT(A) AND B) AND (A AND NOT(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 1 Example: Xor (cont'd)

- 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

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

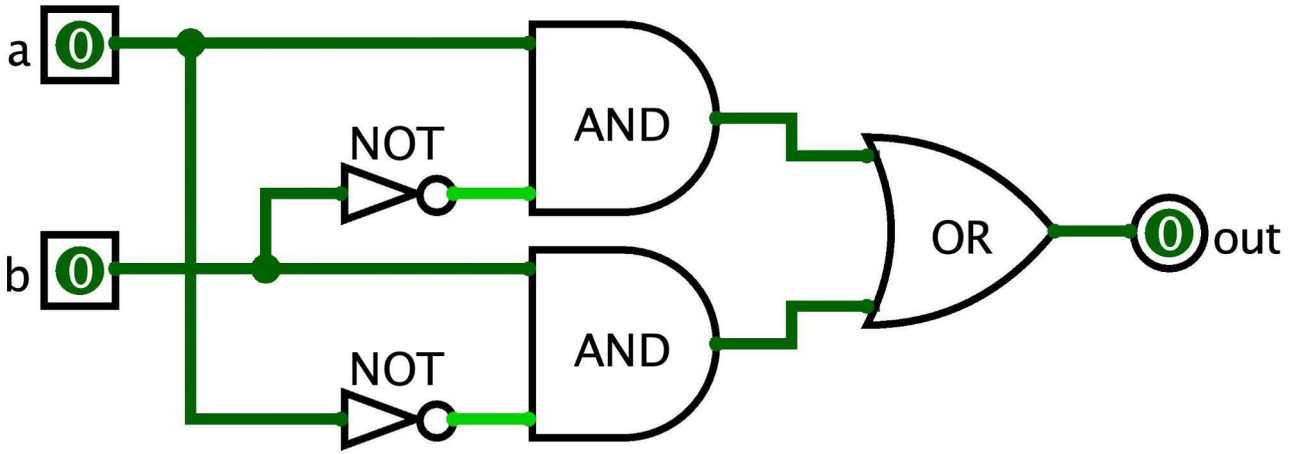
A	B	F	
0	0	0	(Row 1)
0	1	1	(Row 2)
1	0	1	(Row 3)
1	1	0	(Row 4)

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

Project 1 Example: Xor (cont'd)

- Sometimes helpful to convert a Boolean expression to a circuit diagram

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

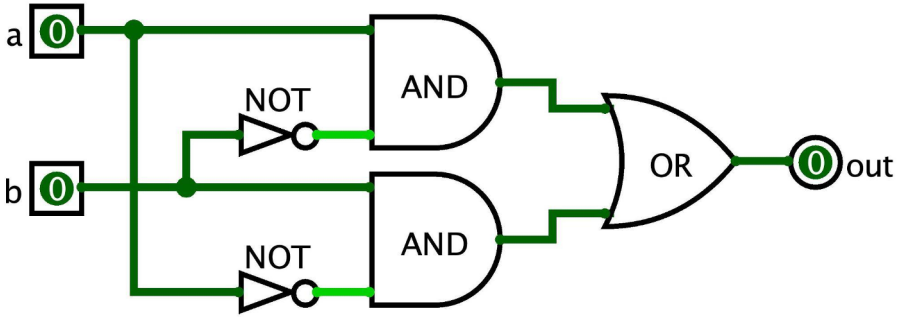


Project 1 Example: Xor (cont'd)

- 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 1 Demo

- Brief demo of the tools you will use in Project 1
- Today's goal: Open and tinker with all the tools
- Stretch goal: Implement the first gate in project 1, **`Not`**
  - Interacting with the tools and implementing the gates, will help you become familiar with the interface and address any confusion

Project 1 Group Work

- Group time to work on the first gate, **`Not`**, in Project 1

- Complete the following tasks in groups:
  1. Read the **Overview** section of the spec
  2. Estimate how much time you think it will take to finish Project 1
  3. Clone your GitLab repository (instructions in Project 0)
  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

- What's in store for Week 2?
  - Technical Subject: Boolean Arithmetic & ALU
  - Metacognitive Subject: Time Management
  - Project 2 released next Thursday (1/13)

- Reminders
  - Project 0 due **tonight (Thursday, 1/6) at 11:59pm PST**
  - Project 1 released
  - Fill out the first Student-TA meeting scheduling spreadsheet here
  - First Student-TA 1:1 ongoing through next week

- CSE 390B meets in-person next week!
  - Gates Center (CSE2) G04