Logic Gates

The Building Blocks of Digital Logic in Redstone

Akilesh Praveen - CMSC389E

UMD

March 5, 2020

Agenda

Announcements

Project 1

- Project 1 has been released, and can be found on the course website under 'Week 2' on the calendar.
- Everything you need to complete the project will be covered in today's lecture, with optional supplementary material provided on the course website under Week 2's resources section.

Inputs and Outputs

Inputs & Outputs

- The Basic Logic Gates make up all of our circuits
- More advanced gates are simply just combinations of the four types of basic gates
- The gates all take two inputs, and produce one output (with an exception of `Not`, which only takes one input)

Inputs & Outputs in Redstone

- Logic Gates are the absolute most basic constructs of the redstone circuits we will be building in this class
- Let's try and represent them using the redstone mechanics available to us
- Remember: we have torches, wires, and repeaters

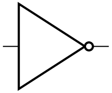
Inputs & Outputs in Redstone

- How can we take our understanding of this and translate it into Redstone?
- Let's go step by step and draw some comparisons
- First, let's handle the input and output for our logic gates
  - Input for a logic gate is composed of 1 or 2 True or False values in discrete math, so let's call it 1 or 2 redstone wires carrying either current or no current
  - Output for a logic gate is a True or False value in discrete math, so let's say our output value can be represented by a redstone wire carrying either current or no current
- Think about what else you'd need to create these gates in Minecraft while we review them

The Basic Gates

Not

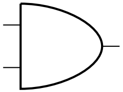
- This gate is not not not not not not one of the most useful logic gates!
- Note that this is the only gate with one input
- Often used in conjunction with other gates to produce a 'not' version of that gate (e.g. `AND` and `NAND`)
- To produce this gate in Minecraft, we have to make use of a special redstone mechanic involving torches



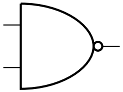
Not	
In	Out
T	F
F	T

And & its evil twin, Nand

- Best way to think of this is to think of the equivalent in English
- If input a AND input b are true, produce a true output
- Placing a `NOT` gate after an `AND` gate results in a `NAND` gate
- We are explicitly mentioning the `NAND` gate for a reason, and we will return to this idea later



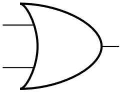
And	
In	Out
T F	F
F T	F
T T	T
F F	F



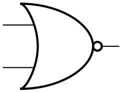
Nand	
In	Out
T F	T
F T	T
T T	F
F F	T

Or & its evil twin, Nor

- Very similar to `AND` in that the English word defines its behavior exactly
- If input `a` OR input `b` are true (or both), produce a true output
- Placing a `NOT` gate after an `OR` gate results in a `NOR` gate



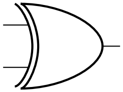
Or	
In	Out
T F	T
F T	T
T T	T
F F	F



Nor	
In	Out
T F	F
F T	F
T T	F
F F	T

Xor, the Unique Gate

- The XOR gate's name an abbreviated form of 'exclusive or'
- An easy way to think about this is: If both inputs are unique, this gate produces TRUE. Otherwise, it produces FALSE
- The logical complement of the XOR gate is the XNOR gate, which has the opposite truth table.
- For simplicity's sake, we won't focus heavily on XNOR for now.



Xor	
In	Out
T F	T
F T	T
T T	T
F F	F

Logical Complements

- Let's talk about logical complements- you've just seen a few
- Occasionally, we'll have a gate or circuit that produces the exact opposite of what we want
    - I'm saying 'circuit' for a reason
- In this case, you'll want the logical complement of whatever gate/circuit you've got
- When drawing circuit diagrams, using the 'dot' notation is sometimes less cumbersome than just sprinkling `NOT` gates everywhere

A Small Exercise

- Most of you have taken/are taking 250, so let's make your TAs proud!
- Try to construct an `XOR` gate using only `OR`s, `AND`s, and `NOT`s
  - Try to be as efficient as possible- circuits had to be made by hand in the old days, and every piece had a cost associated with it!
  - It's also worth noting that more gates = more build time in Minecraft, so it's good to learn to think efficiently in this class
- Why is this important?

Applications in Computing

The NAND Gate's Importance

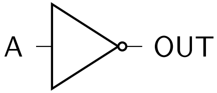
- Let's talk about the NAND Gate
- Why are they the most commonly found gates in computers?
- A Few Reasons
  - NAND is what we call '**functionally complete**'. (So is the NOR gate)
  - If a gate is **functionally complete**, all other gates can be constructed from it
  - in CMOS, (a common circuit fabrication process), NAND is smaller and faster than a NOR gate
- TL;DR - Due to already set industry precedents and the initial ease to manufacture them, NAND gates were the pick over NOR gates for the universal gate that engineers used to build other gates, and eventually other circuits out of.

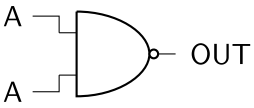
The NAND Gate's Importance (cont'd)

- It turns out, we are able to create a `NAND` gate in Minecraft.
- Because of this, we can conclude that we can create a **functionally complete** gate in Minecraft.
- Therefore, we have proven that we can implement all digital logic in Minecraft.
- Does this mean we will build everything in terms of NAND gates in Minecraft?
    - Absolutely not. Transistors and hardware are one thing in real life, but very different in Minecraft. There are better, easier ways to make gates in Minecraft.
    - In this course, it's important to make note of the few, but key differences between hardware in real life and hardware in game

The NAND Gate's Importance (cont'd)

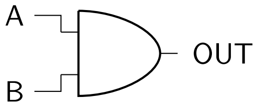
- Let's reinforce the idea of functional completeness
- For example, here's how we can make the `NOT` gate using only `NAND` gates.
- This actually turns out to be the simplest of all once we give it a try.

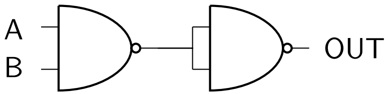




The NAND Gate's Importance (cont'd)

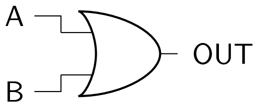
- What about the AND gate?
- Try constructing an AND gate using only NAND gates.
- Remember, you want to think of it more like a circuit- ask yourself: How can I produce a circuit that takes input and produces output identical to a single AND gate using only NAND gates?

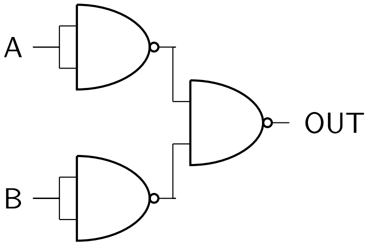




The NAND Gate's Importance (cont'd)

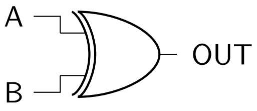
- And the OR gate?
- Try constructing an OR gate using only NAND gates.
- Again, think of it like a circuit. Given limited tools, try and solve the boolean equation that we've posed for you.

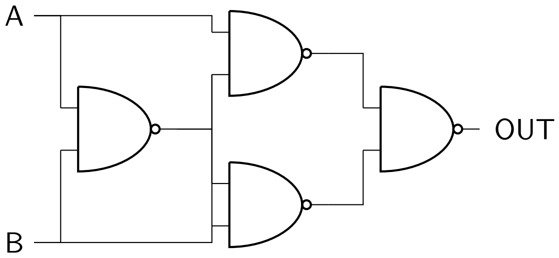




The NAND Gate's Importance (cont'd)

- Finally, let's try the XOR gate
- Try constructing an XOR gate using only NAND gates.
- This one's a little tougher- try and get creative, or just wait for us to reveal the solution.





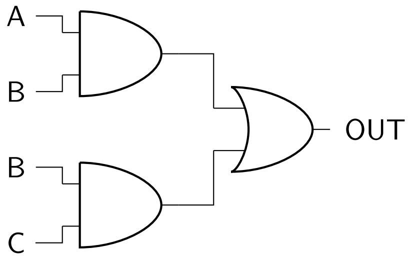
Logic Rules & Efficiency

Logic Rules: In Review

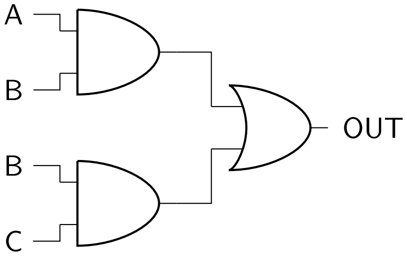
- When discussing boolean logic, we can't forget about the basic rules, or equivalencies, that come along with it.
- Refer to this handy cheatsheet: ▸ CMSC250 Equivalency Reference Sheet
- These rules exist to help us simplify circuits (predicates) that are otherwise more cumbersome to deal with. (In our case, to implement in hardware/Minecraft)
- When designing logical circuits, you will find it **infinitely** more easy if you consistently keep these rules in mind.

An Example

- Here's a simple example of how using these logic rules can help us 'simplify' a circuit while retaining the same functionality
- Let's say we have the given circuit. How would we systematically look at simplifying this?



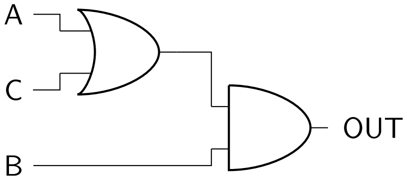
An Example (cont'd)



- Let's take a look at the logical rules presented to us.
- ▸ CMSC250 Equivalency Reference Sheet
- Tip: Write out the boolean logic equivalent of this circuit.
- $(A \wedge B) \vee (B \wedge C)$
- Look familiar? Which rule can we apply to simplify our circuit?

An Example (cont'd)

- After applying the distributive law, we can see that our previous statement simplifies to $B \wedge (A \vee C)$
- As a result, we can represent it as a simpler logical circuit, as you'll see below.



Why learn this? (Takeaways)

- Knowing the basics of boolean logic opens the doors for us to make more advanced circuits than just simple truth table representations.
- Understanding `NAND` gates and `NOR` gates and functional completeness provides us with some background knowledge of digital logic design & doubles as an excellent learning tool for circuit construction.
- Boolean logic rules are essential to designing circuits in the most efficient way possible, saving time, money, and materials in the real world, and saving us from carpal tunnel in Minecraft.

Project 1

Project 1

- For this project, you will be building the basis of an Arithmetic/Logic Unit
- As the name implies, the ALU performs **arithmetic** and **logical** operations on binary inputs.
- This seems like a pretty basic part of a computer, but all computers these days come with an ALU
    - At a very basic level, computers still have to perform these simplistic operations. This is where it happens.
- Further reading on ALUs is available on the course website

Project 1

- For this project, you will be working on the logical part of an ALU
- In simplest terms, you'll be given a few inputs:
  - Some input signals (like A and B in the above examples) to perform a logical operation with
  - Output blocks to 'catch' the outputs of each of these signals
- In order to do this efficiently (i.e. use as little wire as possible), you will be leveraging buses (not the 122 Green kind)
- A video on buses in Minecraft can be found on the course website under Week 1 resources