CS 42—Combinational Logic

Tuesday, September 18, 2018

Summary

Today, we start a two-week investigation into the foundations of how modern computers work. We start by looking at logic gates. The things we talk about during the next two weeks connect directly to the things we talked about during our first two weeks. In the back of your mind, think about how these new topics address our big questions such as “What kinds of problems can computers solve?” In what way might the circuits we talk about today be considered a “computational model”, and how does this model compare to DFAs/NFAs/REs or Turing Machines?

Boolean functions

A **boolean function** is a function that takes n bits as input and returns 1 bit of output.

We often describe a boolean function with a **truth table**, which lists each possible value for each possible input, along with the resulting output. Here is an example:

input			output
x	y	z	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Boolean operations, boolean formulas, and gates

There are three basic boolean functions that we want to pay attention to:

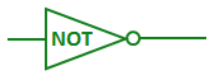
NOT: the function that inverts (i.e., gives the opposite of) its input

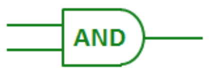
AND: the function that outputs 1 if *all* its inputs are 1

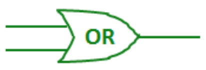
OR: the function that outputs 1 if *any* of its inputs are 1

These will be our basic **boolean operations**: building blocks for every circuit we want to construct.

Assuming we have devices (such as relays or transistors) that can receive and transmit a bit, we can build a **gate** from the devices to represent each of these operations. We use a unique symbol to represent the gate for each operation:







Combinational logic & minterm expansion

Given a boolean function—represented as a truth table—how can we construct a machine that computes that function? The answer is **combinational logic**—a technique for using gates to implement boolean functions.

Minterm expansion: converting a truth table to a circuit

A **minterm** is an AND gate that is connected to all its inputs directly *or* through a NOT gate.

Minterm expansion (sometimes called the "minterm expansion principle") is an algorithm for transforming a truth table into a circuit:

1. Look at all possible combinations of values for the inputs to the function:

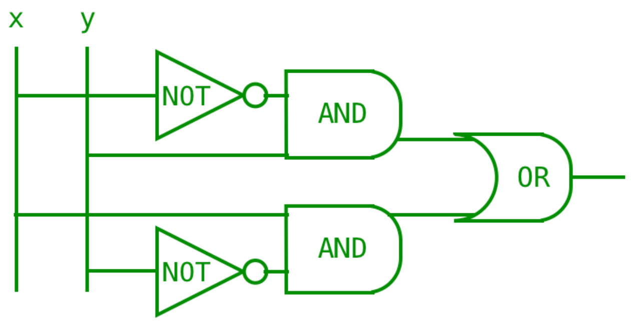
   For each combination of values that should cause the function to output `1`, build a minterm that outputs `1` *only* for those input values (and `0` for all other input values).

2. OR all the minterms together.

The resulting circuit implements the truth table.

Here is an example:

input		**output**
x	**y**	
0	0	0
0	1	1
1	0	1
1	1	0



The midterm expansion principle tells us that NOT, AND, and OR gates are **functionally complete**—they're all the gates we need to implement *any* boolean function.

Perhaps surprisingly, we don't even need all three! We can get by with just AND + NOT or with just OR + NOT.

Next time: Sequential logic (memory)