CS 237 Meeting 7 — 9/21/12

Announcements

1. Homework 1 is due today.

Transistors as Switches

1. A transistor is basically an electrically controlled switch. Like all switches it has two terminals (known as the source and drain) that are connected when the switch is "on" and disconnected when it is "off".

2. Unlike a wall switch, instead of a mechanical action determining whether the switch is on or off. Instead, there is a third electrical connection called the gate that determines this. The voltage level of the wire connected to the gate determines whether the switch is on or off.

   - As the text explains in plenty of detail, transistors depend on the fact that there are two types of "doped" silicon — p and n — and the boundary between regions of differently doped silicon acts as a diode (it only allows current to flow in one direction. In an "off" transistor, current cannot flow because the only available path involves two diodes oriented in opposite directions.
   - The voltage on the gate changes the distribution of ions in the doped silicon effectively temporarily changing a p regions to an n region or vice versa.
   - There are two types of transistors. nMos transistors are on when the gate is at positive voltage and off when the gate is a ground. pmos transistors are on when the gate is a ground and off when it is at positive voltage.

Switches and Logic

1. Two of my favorite words are or and and and and and or are logical operators.

2. The book shows that there is a way to build devices that physically realize the logical operations AND and OR using transistors as switches.

3. For those who find transistors confusing, we can demonstrate the same (almost) relationship using imaginary mechanical switches.

   If we let the flow of current represent 1 and the absence of current represent 0 on the wire and let on represent 1 and off represent 0 for the switches, then

   - Two switches in series (i.e., where the current has to flow first through one and then through the other to complete a circuit leading back to the source of current), implement the AND operator/function.
   - Two switches in parallel represent OR.

4. This can also be done with transistors, but...

Transistors and Logic

1. In real digital systems, we don't use current to represent bits. Instead, voltage is used.

   - A wire close to ground voltage represents 0.
   - A wire close to some positive voltage represents 1.

2. The text explains how to do this with cmos transistors. There are many ways circuits that implement these operations can be built. The book shows just one.

3. I happened to have some old slides that show another approach known as RTL (resistor-transistor logic). I will present this approach since a) it saves me time, b) it lets you see that there are multiple approaches, and c) it may be easier to understand.

4. The only electronics you have to know for this is that transistors act as switches (as we just explained) and that resistors obey Ohm's law — $\delta V = IR$ where

   δV is the change in voltage from one side of the resistor to the other

   I is the amount of current (electrons per unit time) flowing through the resistor, and

R is a measure of how much the resistor actually resists current flow. (For a pure conductor, R would be 0.)

5. With this in mind, we can step through various configurations of two transistors connected in parallel and discover that they don't quite implement OR. Instead they implement NOR, the negation of what OR would produce.

6. If you really like OR, this is not a problem because a NOR acts like a NOT if both of its inputs are connected to the same source. So, with two NOR circuits we can built an OR circuit/gate.

7. Similarly, with two transistors in series, we get NAND, the negation of AND. Again, however, we can easily negate the output of a NAND gate to get an AND if needed.

An Application

1. The book does a great job of describing how gates work inside, but as I read it over, I felt a lack of motivation. It never showed a way that you could do something useful with the circuits they described to build gates.

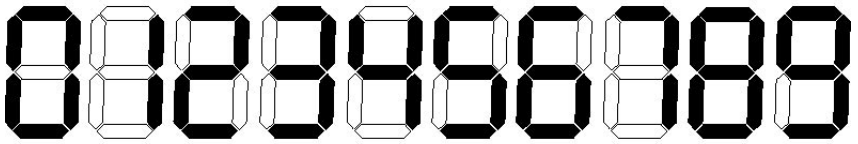
2. I want to try to give you some motivation by showing how to use gates to build a component of a useful electronic device, a calculator.

3. Internally, you might imagine that a calculator represents numbers in binary. At some point, however, to display them it has to translate them into decimal. At least it has to separately represent each decimal digit in binary. So assume we have some four bit binary number that represents a decimal digit we want to display.

4. Calculators display numbers using a device called a 7-segment display (because it uses 7 separate LEDs shaped as short line segments to display numbers).

5. To turn each segment of the display on or off appropriately for the digit that should be displayed, we need a circuit that can take the electronic representation of the decimal digit as 4 binary digits and turn it into a single on/off signal for the segment.

6. For this example, let's focus our attention on the segment at the top of a 7-segment display.



7. This segment should be on as long as the number being displayed is anything other than 1 or 4. In binary, these values are represented as:

0	0	0	1

0	1	0	0

8. Looking at these binary encodings we can see that the top segment should be off if the 1st **and** 3rd bits of the numbers encoding are 0, **and** either 2nd **or** 4th bit is 0, **and** either 2nd **or** 4th bit is 1.

9. Negating everything, we can equally well say that the top segment should be on if the 1st **or** 3rd bits is 1, **or** the 2nd **and** 4th bit are 1, **or** neither 2nd **nor** 4th bit is 1.

10. Replacing the words with gates leads to a circuit that will produce an output voltage appropriate to turn the top segment on or off based on the number being displayed.

Collections of Boolean-valued Functions

1. It should be clear that we could do roughly the same thing for each of the segments of a 7-segment display giving a circuit that took 4 inputs and produced 7 output. Internally, however, this circuit would really just be 7 circuits each of which took 4 inputs and produced 1 output.

2. The power of digital logic is that many useful "mappings" can be implemented in this way. If we think of adding two 8-bit numbers we might think of a mapping from 16 input bits to 16 output bits. Alternately, we can think of 16 mappings from 16 inputs to 1 output.

Circuits for each of these many to 1 mappings can be constructed as we did for the top segment of the display and then combined to do the whole job.

Truth Tables and Finiteness

1. In our discussions, we will use three "notations" for describing boolean functions. You have already seen two:
   - Circuits built from gates like the 7-segment display example,
   - Truth tables like the ones we showed to summarize what the circuits using switches and transistors actually did.
2. Generally, when we want to describe a boolean function, we will give variable names to its inputs.
   - For example, it would be natural to call the 4 digits input to our 7-segment display example as d_3, d_2, d_1, and d_0.
3. To describe a function as a truth-table, we create one column for each input/variable and one column for the output.
4. We want to put every possible combination of values of the input variables in the variable columns. If there are n variables, there will be 2^n combinations so our table will need 2^n rows.
5. We fill the variable columns by counting from 0 to 2^n in binary.
6. After we fill the variable columns, will fill in the output.
7. To illustrate this construction, consider the relational operator $>$ as a boolean function. That is, in any real computer $>$ would take two binary numbers of some fixed size n and return a single 0 or 1 indicating whether the first number was less than the second.
8. To keep the table size down, we will work with very short binary numbers — two bits each.
9. How many possible different output columns are there?

$$2^{2^n}$$

10. This is important! For any given number of inputs there are only a finite number of different boolean functions.