CS 237 Meeting 13 — 10/10/12

Announcements

1. Lecture examples are being added to class web page.

Take a Little Time

1. The structure of a circuit determines the speed with which it can "compute" the output of the function it implements.
   - When the voltage on a transistor's gate changes, it takes some finite amount of time for the transistor to turn on or off.
   - The time from when a new set of input values arrives at a gate and it is safe to assume that the output value(s) have changed and are stable is called the gate delay (it varies from one type of gate to another).
   - A device's overall delay is determined by the path from some input to some output with the highest combined gate delay.
2. Last time we saw that an n-bit adder built by chaining together 1-column full adders requires roughly $2n$ gate delays from when new input values are presented until the final carry output becomes stable and correct.
3. We know we can do better than this! Each output bit can be expressed in sum-of-products form where the maximal path will be three gates long (one not, one and, and one or).

   Unfortunately, the number of gates in such a circuit will be enormous! The "average" min-term for the higher order bit of the sum of a two input n-bit adder would require n not gates and an 2^n input and gate. There could be as many as 2^{2n} min-terms.
4. There are options between the simple carry ripple adder and the sum-of-products term.
5. In a carry-lookahead adder, we break the inputs into m-bit chunks.
   - For each m-bit chunk we add circuitry to compute two boolean values:
     – Propagate - whether these m-bits would propagate a carry if there was one from the bits to the right. This is basically the **and** of the **or** of each pair of inputs (i.e. every column must have at least one 1 input).
     – Generate - whether these m-bits generate a carry regardless of the carry in. This is a bit recursive. It is the **and** of the high order bits **or**ed with the the "Propagate" (**or**) of the high order bits and the "generate" of the low order bits.
   - These two bits will be computed by the high order chunks while the carry is propagating through lower chunks.
   - Once a carry reaches a chunk, we just **and** it with the propagate bit and **or** it with generate to determine the carry out of the chunk.

     Thus, instead of having two gate delays per column, we have two gate delays per chunk of columns.

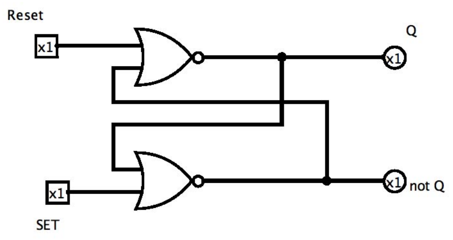
Other Combinational Circuits

1. There are a number of combinational circuits that are covered in the text which we will not discuss in class. In particular, the book discusses how to build circuits to preform other standard mathematical operations (subtraction, multiplication, and division) and not-so-standard operations that are common in programming languages (shifts).
2. You should be familiar with the basics of how these circuits operate since we will assume their existence when we examine the overall structure of a computer's components.

What a Memory

1. Combinational circuits are wonderful, but to build a computer we need circuits that can remember things.
2. The basis for all memory circuits are devices that have more than one stable state and a means for an external device to cause a switch between these two stable states.

3. The canonical example of such a device is the SR latch, obtained by connecting two NOR gates in such a way that each of the gates takes the output from the other gate as an input.



- The circuit has two possible outputs. In most cases, one will be the inverse of the other, so it is traditional to label one Q and the other $\bar{Q}$
- It is also traditional to label the external input to the NOR gate that produces Q with R (for reset) and the other input with S (for set).
- If we try to build the truth table for Q given this circuit, things are fine as long as we only consider rows where one of the inputs is 1 since that 1 ensures that the output of the NOR gate that receives it will be 0:

Set	Reset	Q
0	0	??
0	1	0
1	0	1
1	1	0

- If both input are set to 0, then the output of each gate depends on the output of the other gate. If Q is 1, then $\bar{Q}$ must be 0. If Q is 0, then $\bar{Q}$ must be 1. Either way, the circuit will be happy.

  This is the *bistable* behavior we need. When both inputs to an SR latch are 0, it is happy to set forever in one of these two stable states.
- How does the circuit decide between its two stable states. Luckily, the answer is history. As long as the circuit is in a state where Q and $\bar{Q}$ are complements before both inputs become 0, it will stay in that state after it becomes 0.
- Setting S to 1 and R to 0 makes Q 1 and $\bar{Q}$ 0. Setting R to 1 and S to 0 makes Q 0 and $\bar{Q}$ 1. Therefore, Q will be 1 while both inputs are 0 if and only if the last input set to 1 while the other input was 0 is S.

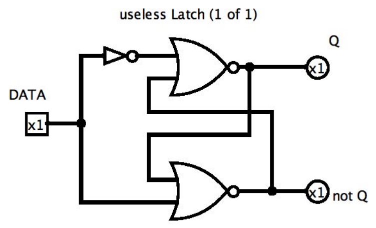
4. This circuit is called an SR latch. In its raw form, it is not that useful, but we will see that it makes a great building block.

More Flexible 1 Bit Memories

- The weakness of the SR latch is that it does not match our model of what memory should do. The SR latch remembers what input line was set to 1 most recently. When we think of memory, we think of a device that can remember a number, not something involving the relationship between two input lines.

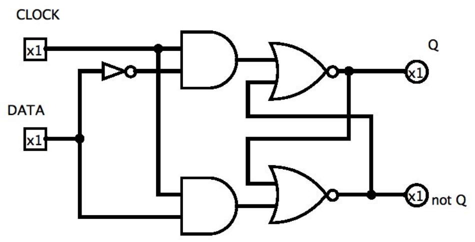
- Numbers in computers are just sequences of bits. Therefore, if we want to start simple the first step is to recognize that a device that could remember a bit would give us the ability to remember a number (we just use as many of the devices as there are bits in the number).

- The first trick to building such a device is to recognize that the SR latch really provides this ability. If we want to remember a 1, we just set the S line. If we want to remember a 0, we set a 1 on the R line. This naturally leads to a very silly circuit in which we tie a one bit data input (usually named D) to the S line and its negation ($\bar{D}$) to the R line.



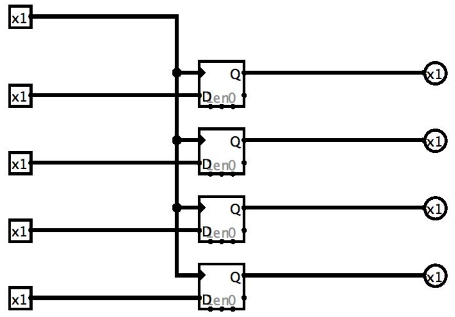
This circuit is silly because its output (Q) will always just equal D.

- The point in discussing this silly circuit is to make it clear that we don't just want to remember the value on a particular wire in a circuit. Instead, we want to remember that value at some particular time at the past. To be useful, a memory has to provide controls that let us both send it the value we want to remember and tell it at what point in time to remember the value.
- The D latch accomplishes this by taking two inputs:
  - The D input, as described above is a bit to be remembered, and
  - The CLK input is set to 1 when we want to record each change in D and set to 0 when we want to remember a particular value of D and ignore future changes in D's value (until CLK becomes 1 again).



Building Multibit Memories

1. Given devices like the SR and D latches, it should not be hard to imagine how to build devices that function somewhat like the registers in the MIPS machine. For one register you could
   - Layout 32 D-latches.
   - Connect their D input to 32 distinct input lines used to send the value to be stored in the register.
   - Hook all of their outputs to some sort of multiplexer that can be used to select this register's values from those of its piers.
   - Connect all of the CLK lines to a single wire.
2. Even drawing a picture of a 32 bit register seems daunting, but here is a drawing of a 4-bit register.


x1
x1
x1
x1
x1
Q
D en0
Q
D en0
Q
D en0
Q
D en0
x1
x1
x1
x1