03-reality-notes

Reality Check

Agenda

0. Re-Orienting

1. Reality affects logic

2. Logic in reality

3. Teaser

---

0. Re-Orienting

Where are we, and how did we get here?

From the top, going down:

- mental model of programming
- layers of software
- OS
- ...
- functions on integers
- integers

From the bottom, going up, sort of...

- atomic physics
- voltage levels
- 1, 0
- logic gates
- next week:
  - binary numbers
  - combinatorial logic

---

1. Reality affects logic

"Keep secrets of the implementation. Secrets are assumptions about an implementation that client programs are not allowed to make" (Butler Lampson)

But sometimes, we NEED to see through the abstraction.

In particular, the lower layer can reach up and grab us

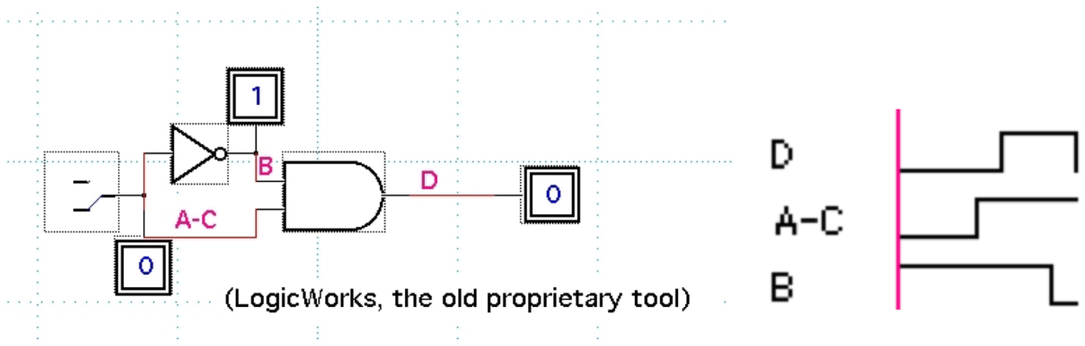
Biology metaphor. Try to imagine a complex business or treaty negotiation... except no one is allowed to go to the bathroom.

Relevance to logic design: it's not just abstract, pure computation. Electricity and physics and the real world are involved. These can reach up and change how you do things.

Example: time delay through gates

---

consider



If we flip the switch to 1....

- What should happen, if we imagine a world of pure logic?
- What does happen, when gates take nonzero duration before output reflects changed input?

(See main in **delayed.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/03-reality/demo/delayed.zip)**)

(Or if you want to be really mystified, try **rising.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/03-reality/demo/rising.zip)**)

Example: time delay through wires

Admiral "Amazing" Grace Hopper **(http://en.wikipedia.org/wiki/Grace_Hopper)**

Example: cost of a zero

do you really care whether "0" is "false"? Or can it be "true"?

negative logic

DeMorgan's Law:

$$\overline{\overline{A} \wedge \overline{B}} = A \vee B$$

[dlaw.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/03-reality/demo/dlaw.zip)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/03-reality/demo/dlaw.zip)

In traditional approaches to build gates out of transistors, the convention emerged that:

- "1" was a very wimpy (low current) positive voltage.
- "0" was a very powerful (high current) ground.

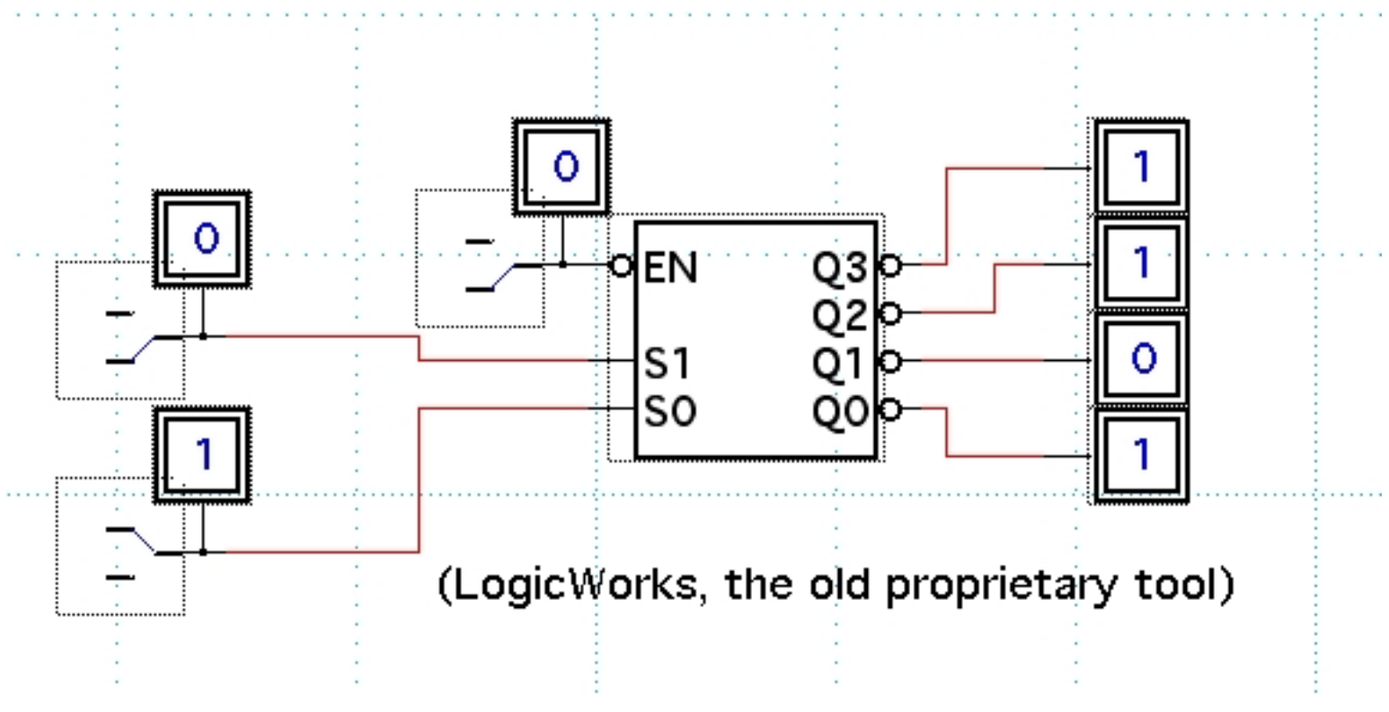
So zeros actually **cost more**.

so... we commonly see

- "active low" inputs
- "active low" outputs

(Since the "active" case is often the less common one. E.g., think of a decoder's output lines)

For example:



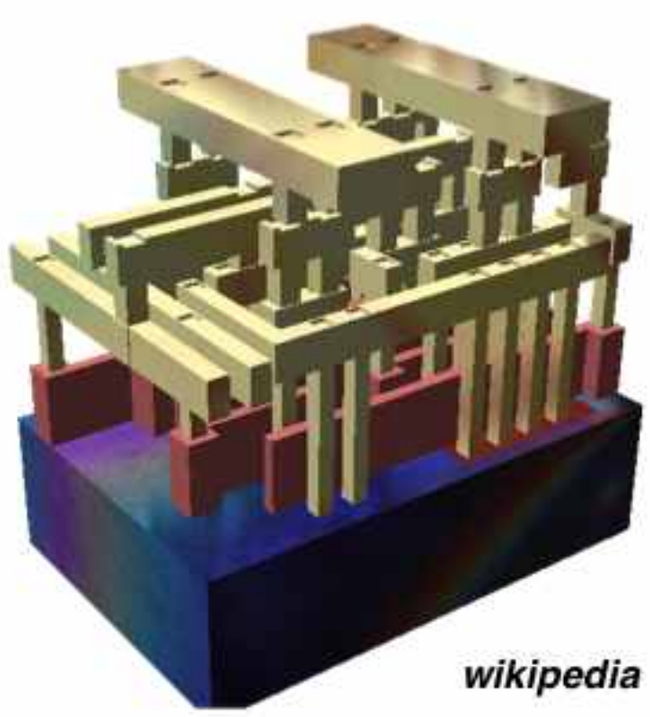
2. Logic in reality

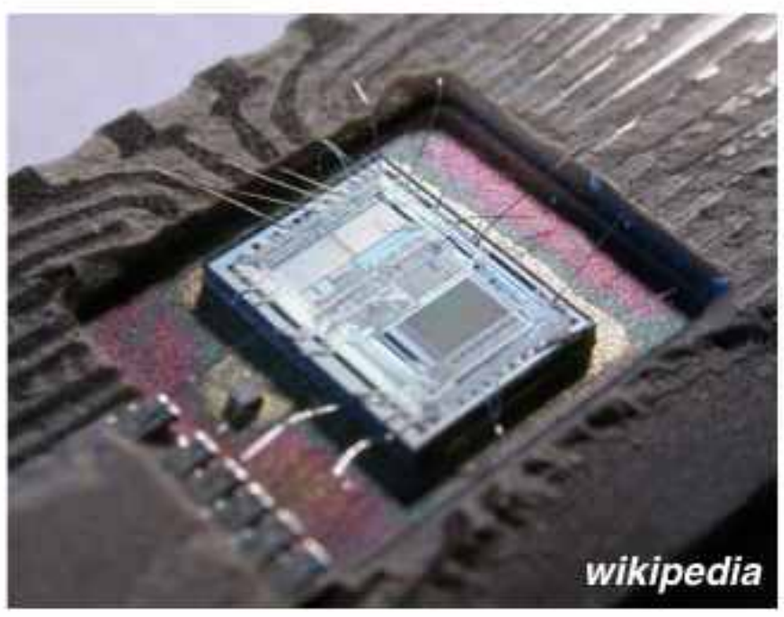
Discrete Transistors

They used to build gates and such out of these.

Integrated circuits

Put complicated circuits into one larger package!





Building circuits from discrete logic ICs

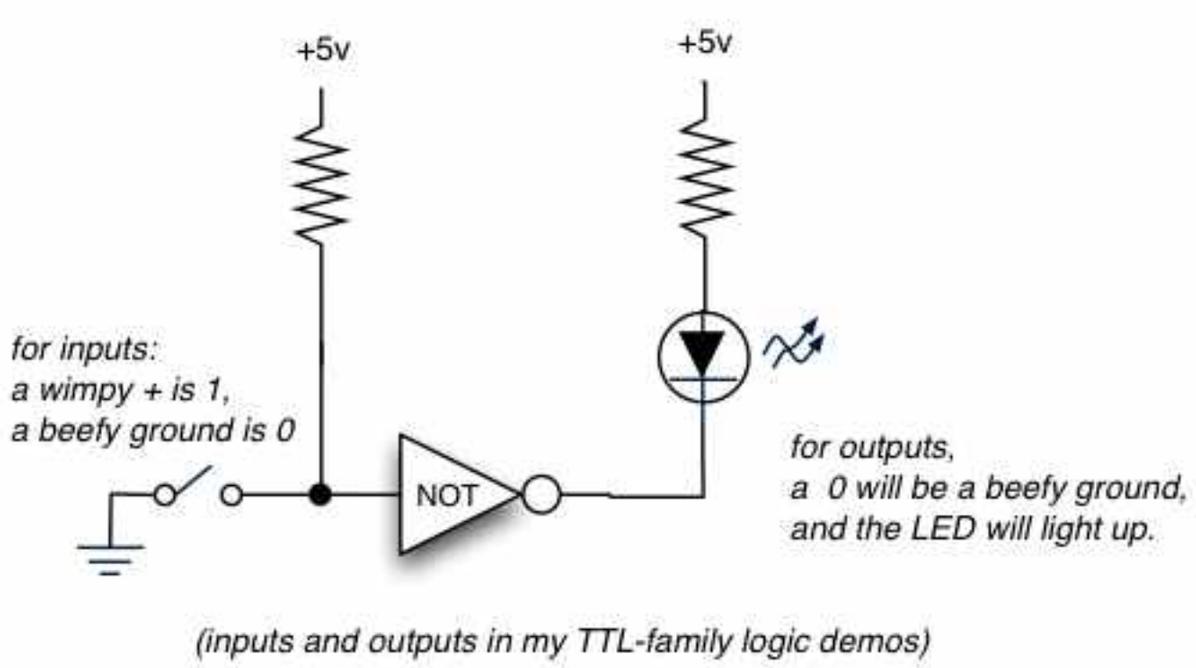
look up the **chip layout (https://ssl.cs.dartmouth.edu/~sws/cs51-s14/03-reality/databook1.pdf)**

hook up power and ground

and go for it!

In the TTL family

- wimpy 1s, beefy 0s
- so INPUT 1s can come through a resistor... or even be "floating"
- an OUTPUT 1 can't do much. But an output zero can drive a LED.



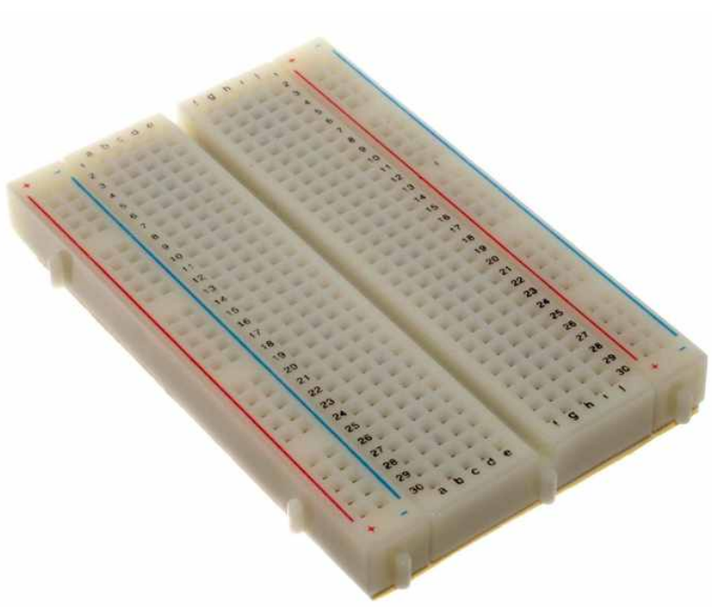
(inputs and outputs in my TTL-family logic demos)

How to wire up circuits with logic ICs? Some common approaches:

- medium 1: breadboard
- medium 2: wirewrapping
- medium 3: printed circuit boards

Just don't get too fast. Or too big.

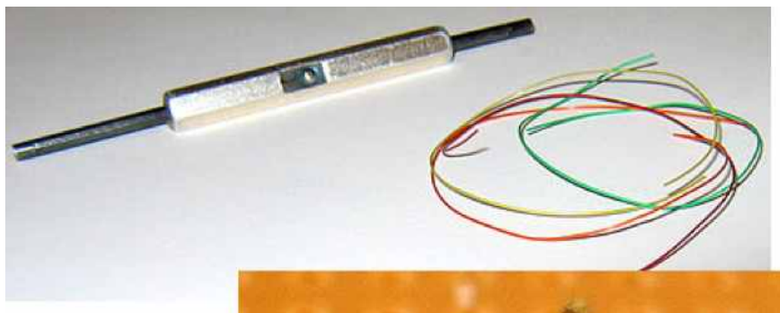
Demo of NOT gate on a breadboard

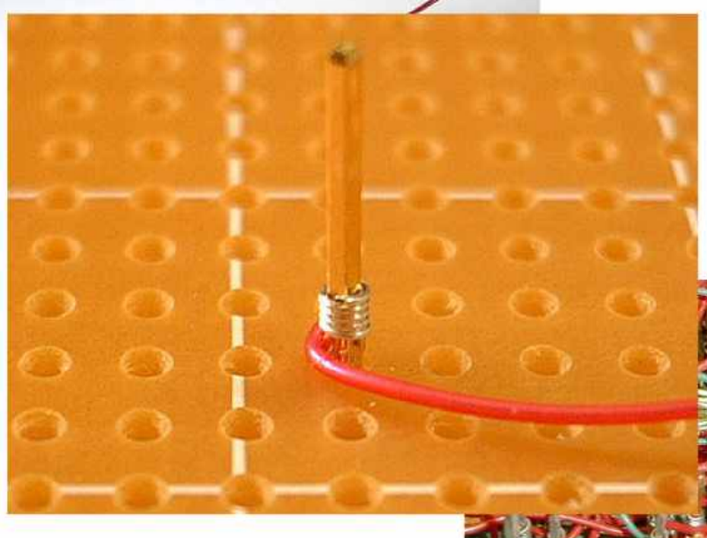


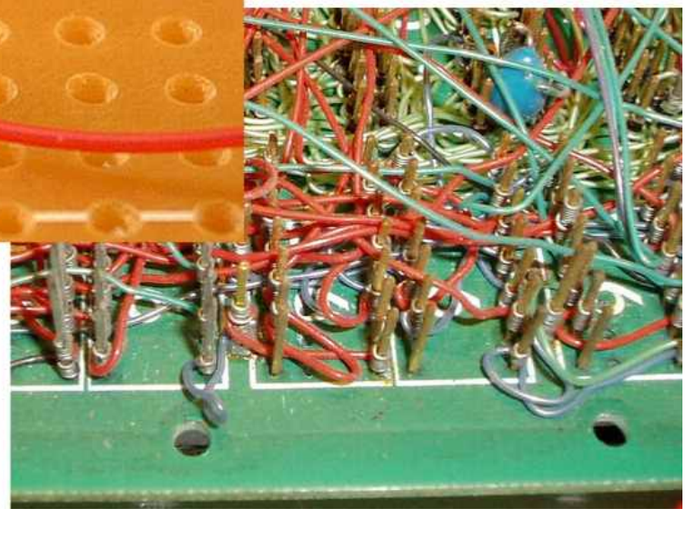
wikipedia

Anybody feel cursed?

Demo of NAND gate with wirewrapping







wikipedia

Discussion of printed circuit boards

The Fine Print

for example (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/03-reality/databook2.pdf)

Consider **fineprint3.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/03-reality/demo/fineprint3.zip)**

Demo of NAND overdrive

Putting it all in one IC.

the ASIC: "application-specific IC"

the FPGA: "field-programmable gate array"

Pros and Cons

Generating your design

you COULD do it by "hand"

but what about all that crazy optimization?

and what if you wanted to massage the circuit to accommodate the unique constraints of a particular FPGA, say?

New reality: design circuits by writing programs in a high-level language (Verilog and VHDL are popular choices), and then have your IDE "compile" it into the circuit

- Section 4.2 in the textbook discusses the authors' simple language, HCL
- **waside-verilog.pdf (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/03-reality/waside-verilog.pdf)** (from the textbook site), starting at page 18, shows some sample verilog

3. Teaser

So far, the digital logic circuits we've thinking of (and the ones you'll build next week) are all DAGs: directed acyclic graphs

What if we do something weird, like introduce cycles?

In **rs.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/03-reality/demo/rs.zip)** ...

- trace through the values
- press a button, then trace through. (think of the inputs as "active low")
- notice what happens when you release the button!
- (when i was a kid, I ended up building things like this for "block signals" in my neighbor's model railroad layout)