08-clocks-notes

Clocks and Sequential Logic

- 0. Re-orient and review
- 1. Sequential Logic
- 2. Clocks
- 3. Something that really counts
- 4. I Like Traffic Lights
- 5. The State Machine approach

Re-O

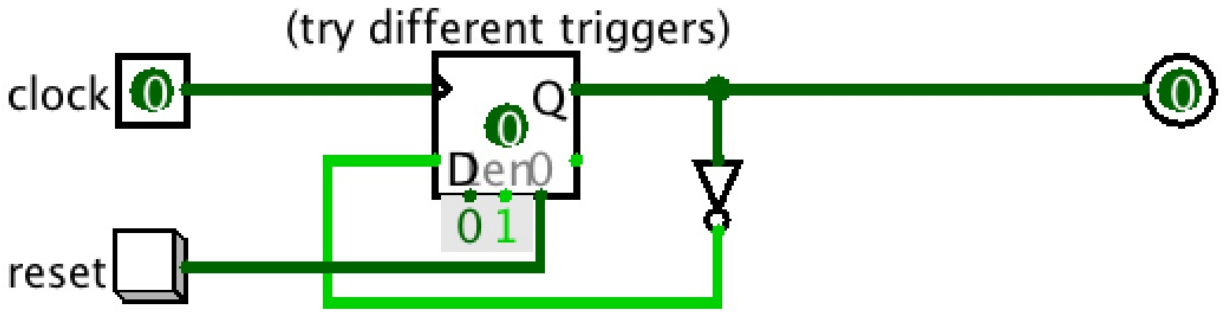
The computational stack...

---

1. Sequential Logic

(Review levels, edges, and those time-value graphs I keep drawing)

What if we wanted to use the output of a latch/flip-flop as the input to itself? (maybe via a combinatorial logic circuit... such as, in this case, a single gate)



"flippy" in **flashy.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/08-clocks/demo/flashy.zip)**

Using the value in a latch as input to a cloud of logic that then changes the value of that latch can spell trouble.... but we can avoid that with an edge-triggered flip-flop

---

How about using storage as input to more general combinatorial circuits?

Such as recognizing whether the stored 4-bit words is a "1"?

---

or a "9"?

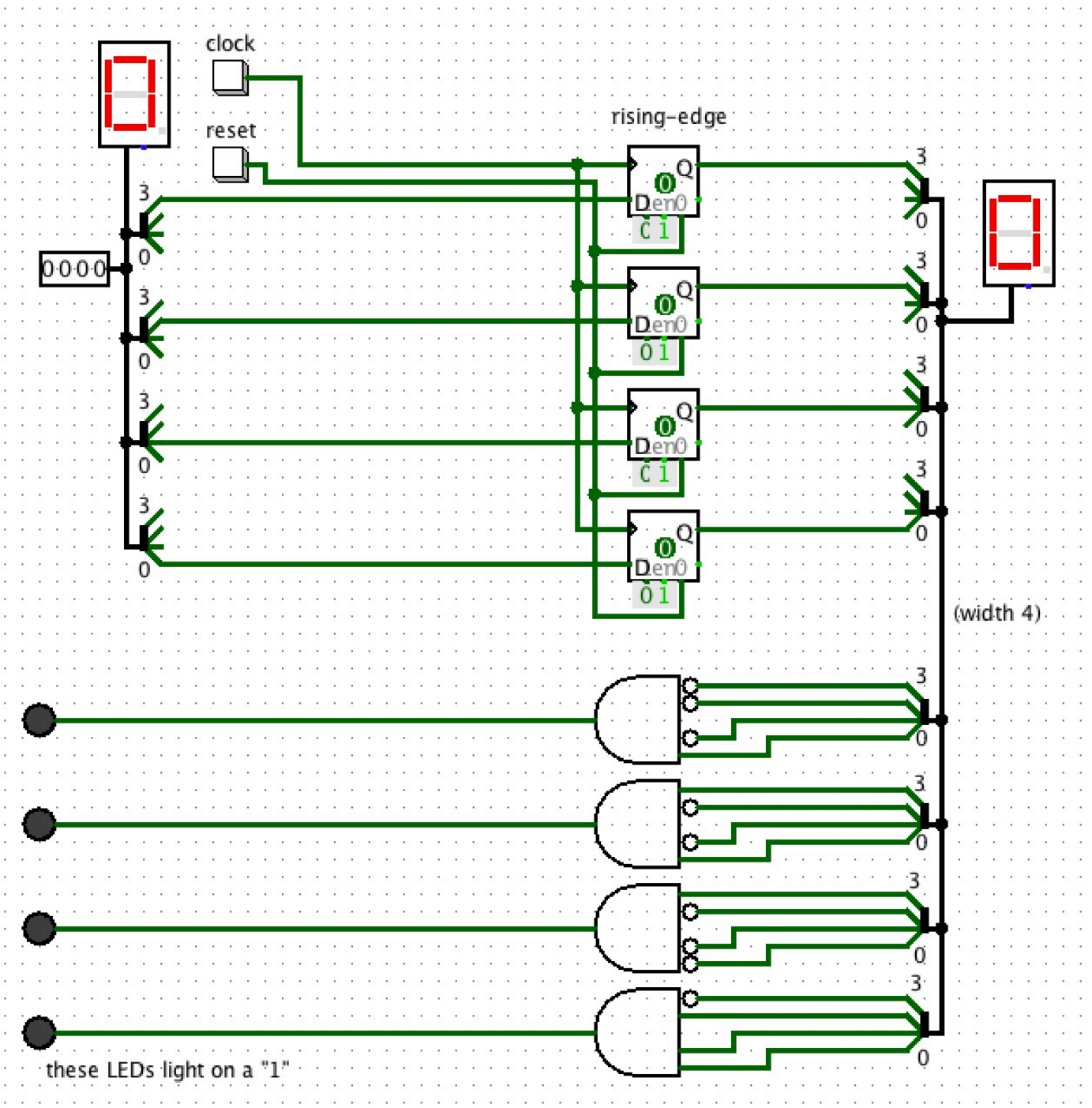
---

or an "8"?

---

or a "7"? (Since I was class of 1987)

How about using a combinatorial circuit to determine what should be stored?



"1987-skeleton" in [flashy.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/08-clocks/demo/flashy.zip)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/08-clocks/demo/flashy.zip)

(or "1987-full", finished up)

Talk through the pieces involved...

- the four flip-flops storing state
- each AND gate recognizing a particular state.
- an activated AND driving input into the latches
- the clock pulses

Q: what happens if we used level-sensitive latches instead of edge-sensitive flip-flops?

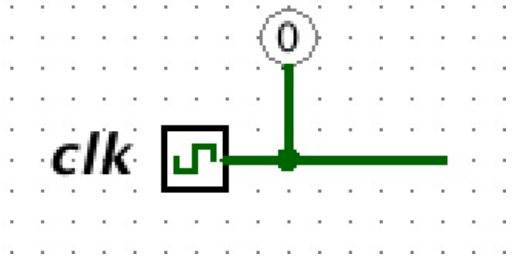
---

2. Clocks

The idea of a clock cycle. ("Rhythm" might be a better term!)

"clock0" in

clocks.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/08-clocks/demo/clocks.zip)



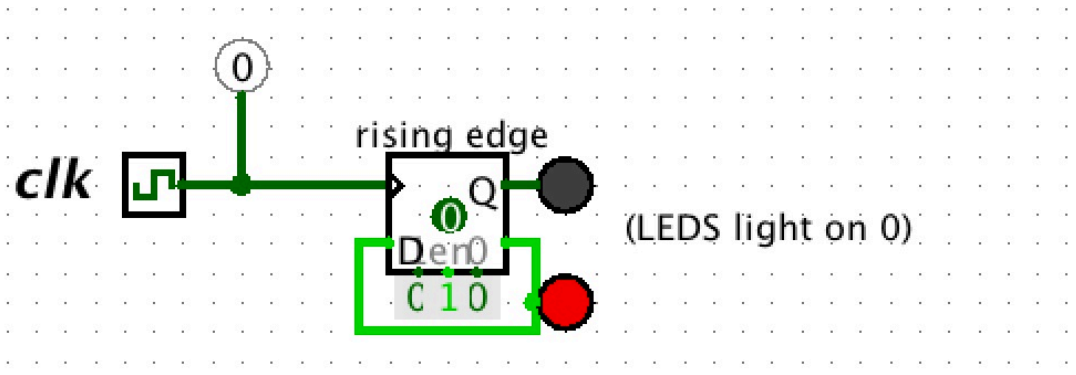
---

what if we feed it to a D flip-flop?

---

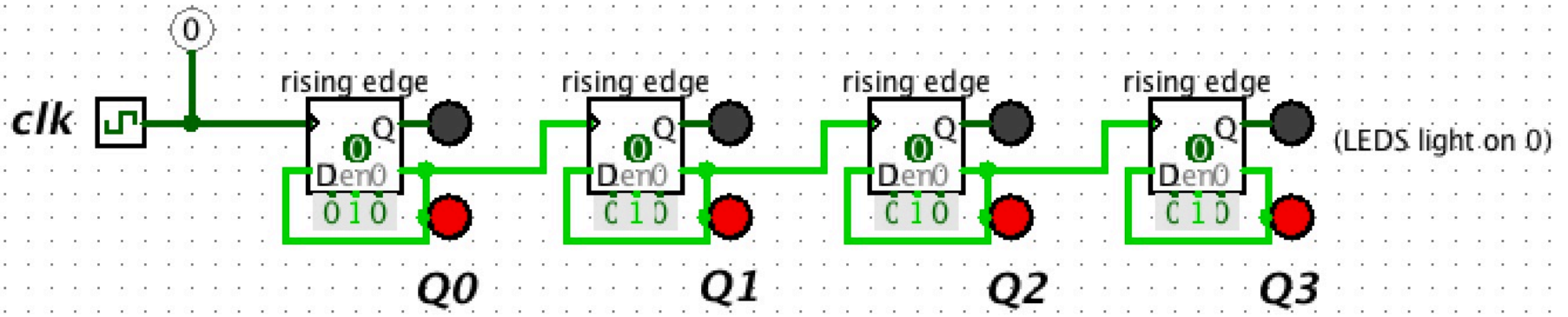
what if we wire NOT-Q to D?

"clock1"

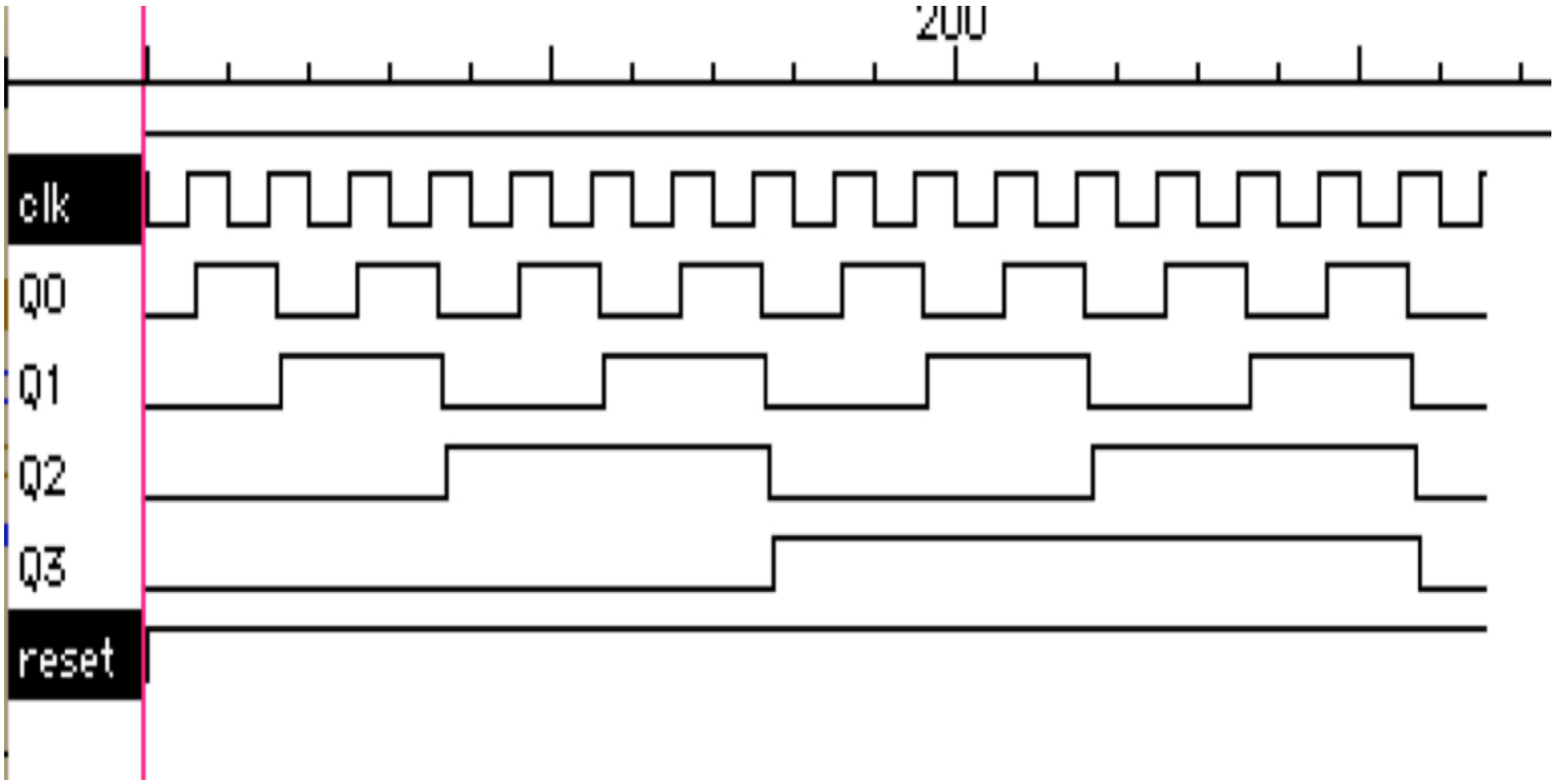


---

what if we keep going?



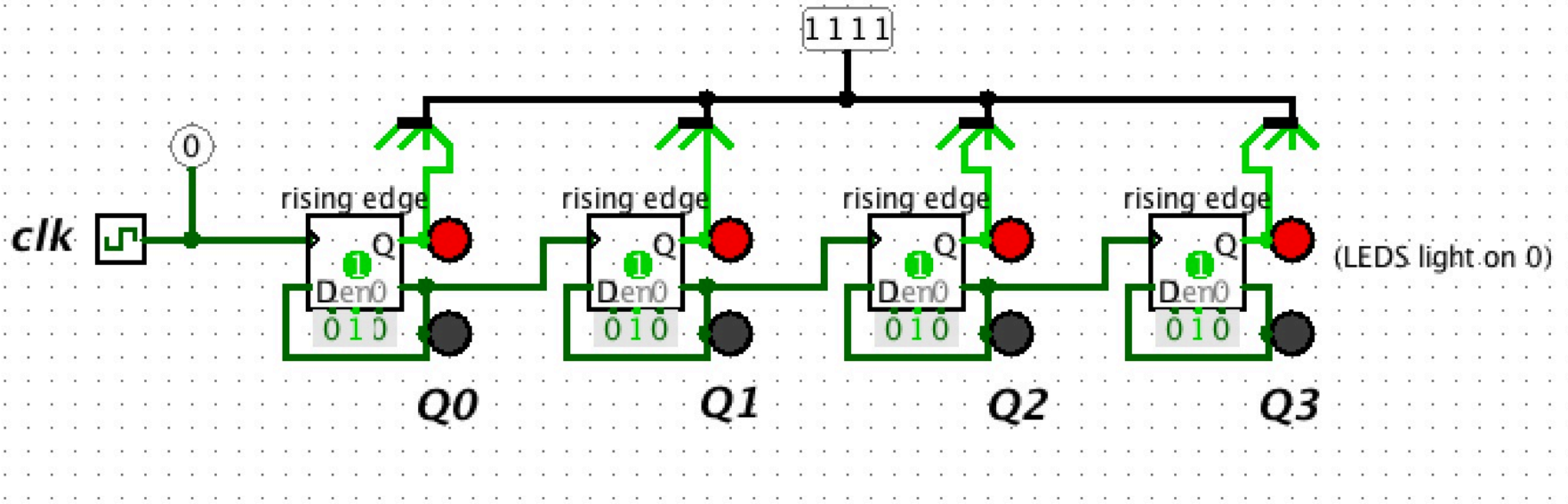
"clock4"



3. Something That Really Counts

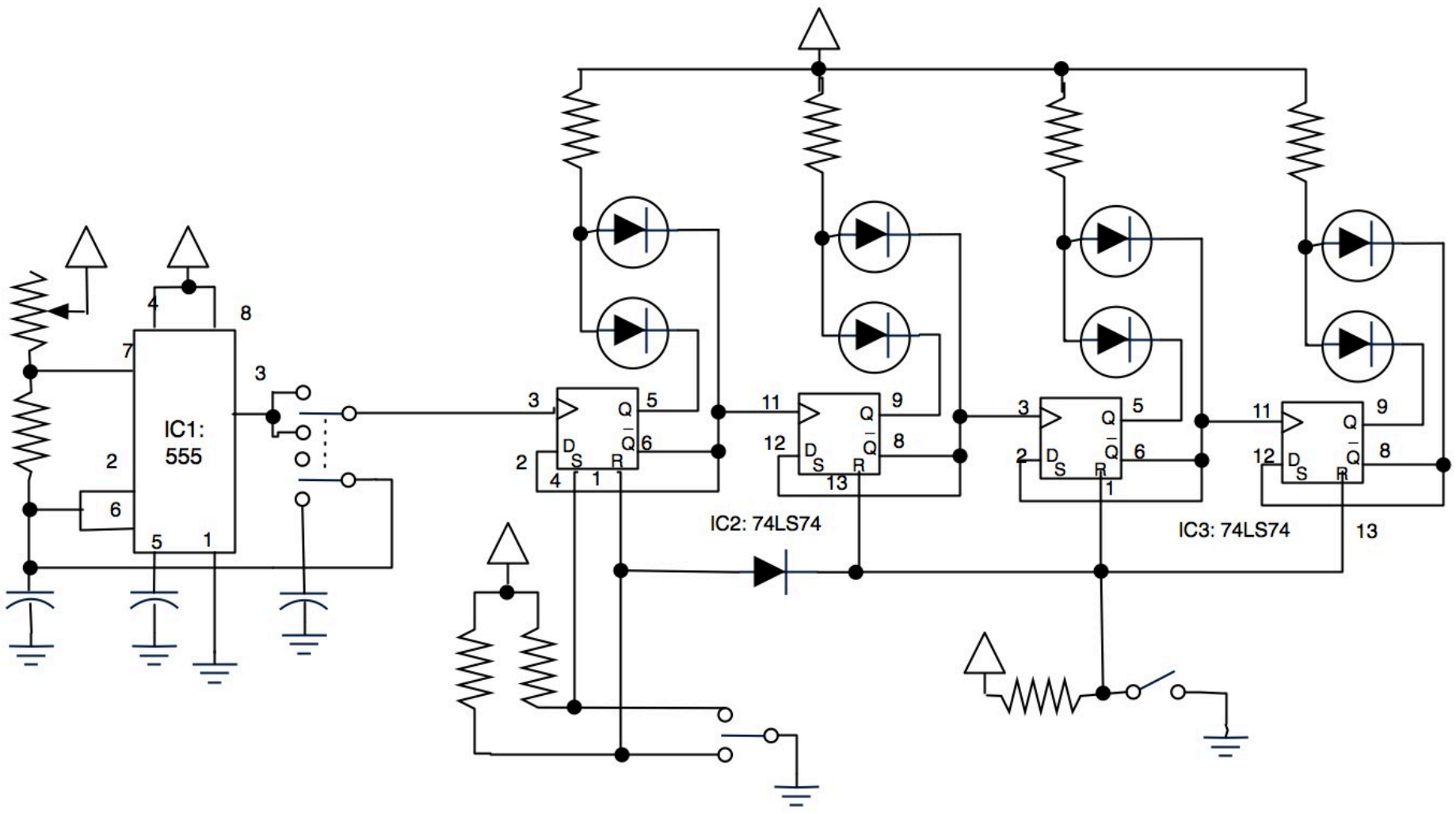
With the cascading flip-flip demo, we're using stored values as inputs to our logic, and also using the logic to change the stored values. And using storage to CLOCK as well. Did you notice anything interesting?

"clock5"



(How could we make it wrap around at 9?---see perhaps clock6-full)

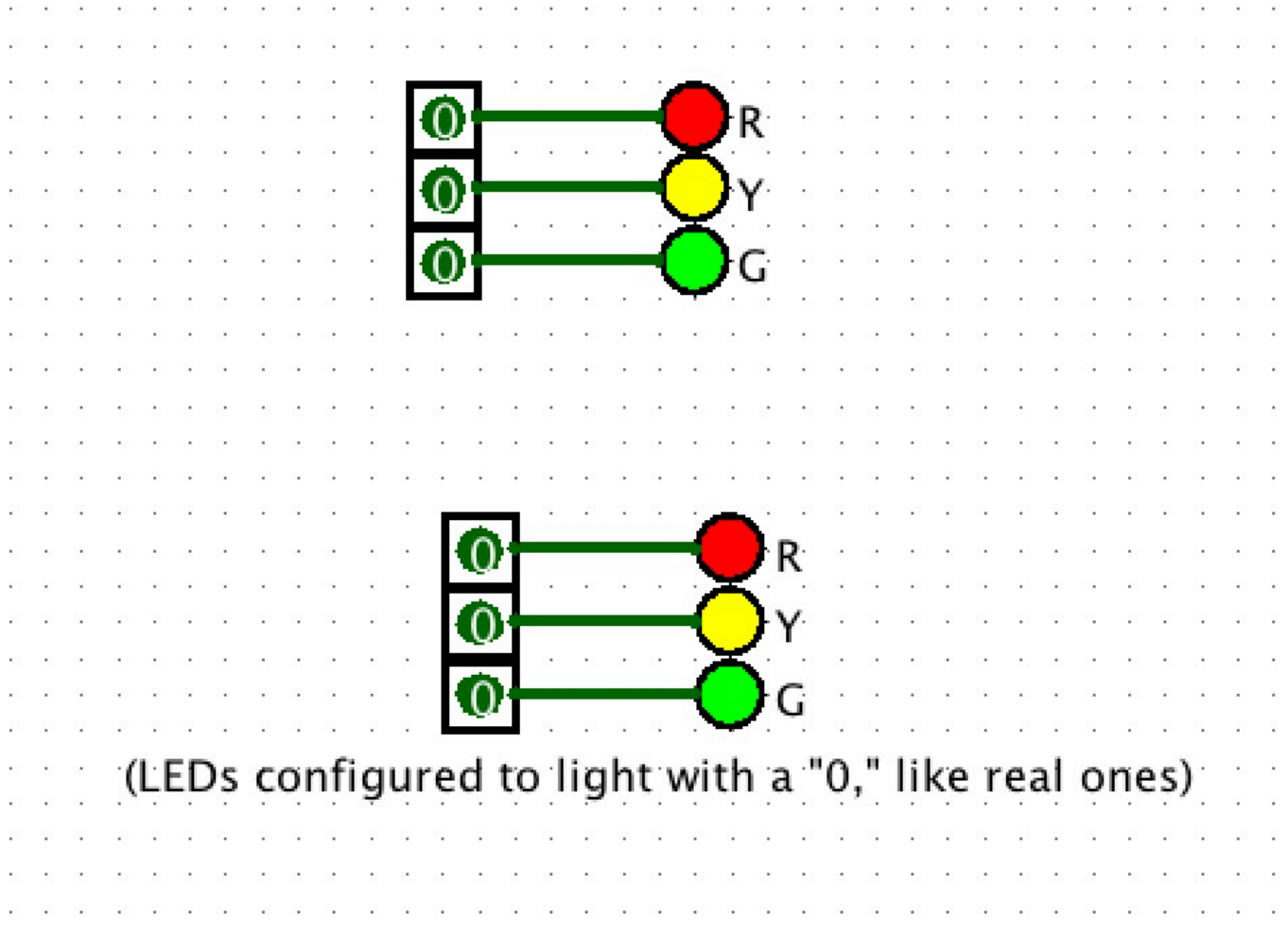
demo: SEEING it.



demo: HEARING it.

4. Consider the traffic light

the "basic light" circuit in [traffic-lights.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/08-clocks/demo/traffic-lights.zip)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/08-clocks/demo/traffic-lights.zip)



consider: what should a traffic light DO?

and how do we build a circuit to drive it appropriately?

5. The State Machine Approach

Work through the states of a traffic light.

For each state:

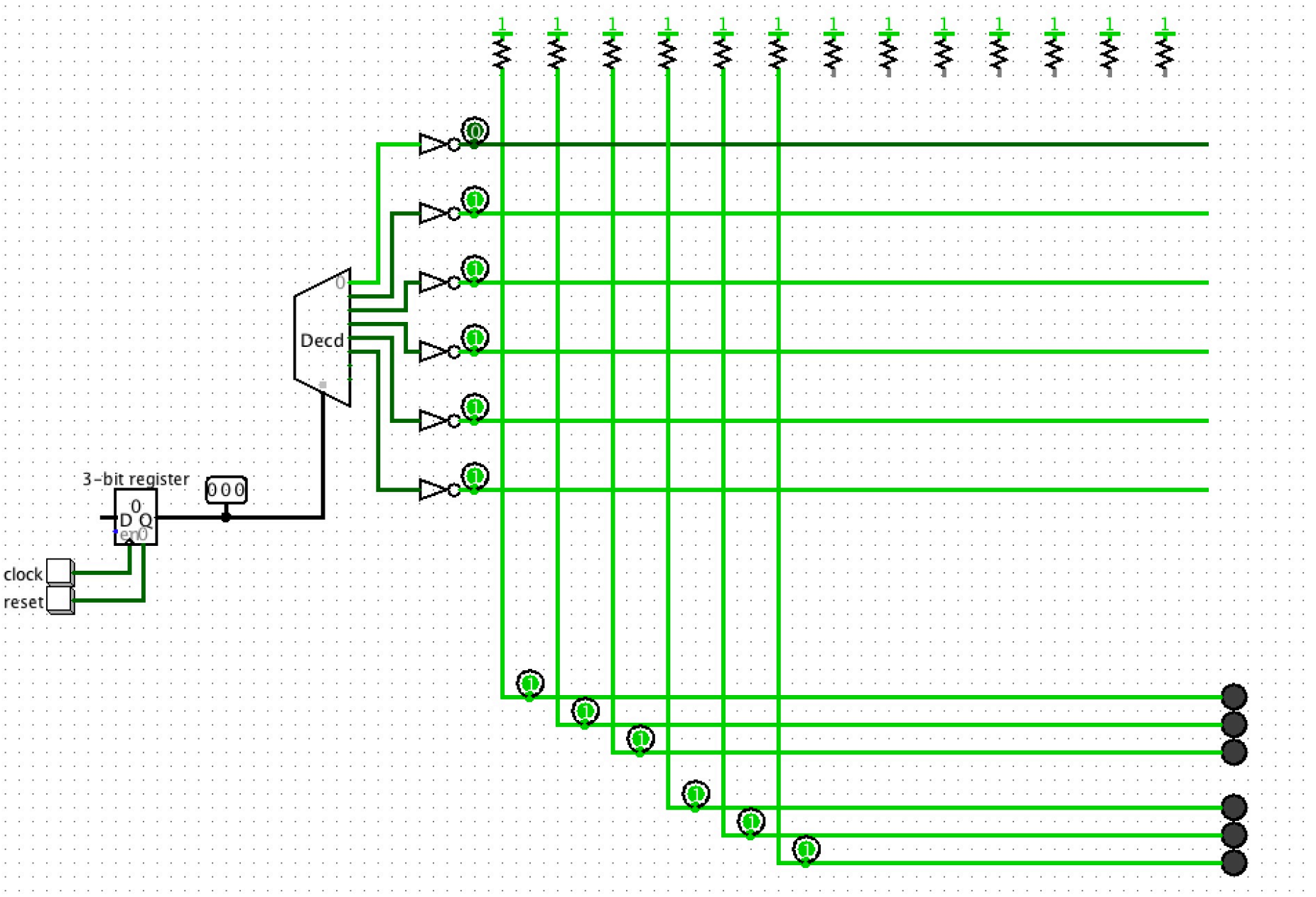
- figure out what should happen with inputs (none, for our traffic light) and outputs (the lights)
- figure out what the next state should be

OK, now work through this table for the traffic light states

OK, how to build as a circuit?

- encode the state as some type of binary string
- store it in a storage element
- for each state, have some logic that detects that state
- ...and drives the outputs
- ...and drives the "next state"

Example: a traffic light, FSM version:



the "fsm-skel-0" circuit in traffic-lights.circ

Uh-oh. How and when do we go to the next state?

(Textbooks calls this the "finite state machine" approach. Which prompts the question: what's the OTHER kind---INFINITE state? Actually, they probably mean "finite" as "> 1". Go figure)