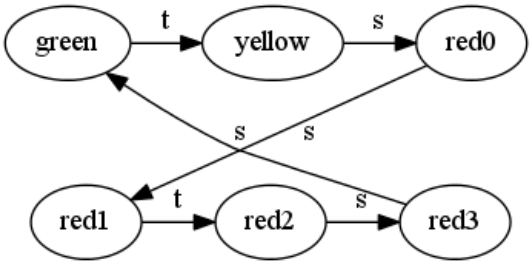
DFA to Circuit Layout Practice

March 16, 2017

NOTE: We may use a variation of the following DFA (e.g., with some added states/inputs) with some similar and some different questions for the midterm exam. We're releasing this for some practice.

Consider the following DFA with six states indicating whether a traffic signal is green, yellow, or red (with four "red" states). The DFA has an input language of $\{t, s\}$ (i.e., the two symbols t and s), which represent signals from a short (s) and long (t) timer. **WARNING:** To keep the diagram clean, we have omitted all "self-loops" from this DFA. That means there is no arrow in the DFA for any time an input letter would lead the DFA to transition **back to the same state**.



1. Imagine this DFA starts in the state `red0` and processes the following input stream `sststs`. Indicate what state the DFA is in at the end of this input stream.

   REMEMBER that where an arrow is missing for an input at a particular state, that means the input causes the DFA to transition back to that state.

2. Briefly explain why there are four different "red" states instead of just one. *Hint:* imagine there were two copies of the DFA, one starting in `green` and representing one direction of traffic at an intersection, and one starting in `red1` and representing the other direction of traffic. What can you say about what states these two DFAs will be in after any sequence of input letters?

3. How many D Flip-Flops will be required to represent this circuit's state?

4. How many bits of input (i.e., input wires) will be required to represent this circuit's input?

5. Translate this DFA to a circuit. Your circuit must (1) use the representation below for inputs and states and (2) use the style of computing the next state value in which the current state is used as the selector for one or more multiplexers that choose the appropriate one of several candidates for the next state values(s).

State	`green`	`yellow`	`red0`	`red1`	`red2`	`red3`
Representation	0	1	2	3	4	5

Input	`s`	`t`
Representation	0	1

1 Sample Solution

Don't look here until **you've** worked through the problem!

1. Here's the sequence of states entered for the input `sststs`: `red0` (start), `red1`, `red1` (because we left out self-loops, there's no explicit arrow), `red2`, `red3`, `red3`, and `green`. So, this sequence ends at the state `green`.

2. We never want both lights to be non-red (i.e., green or yellow) at the same time. That's a safety hazard. Further, for safety traffic lights have a brief period where both directions are red. So, `red0` gives us that "brief period", and `red` 1 through 3 "match" the `green`, `yellow`, and `red0` states so that the lights stay "in sync". (I.e., they really represent "red while the other light is green", "red while the other light is yellow", and "red while the other light is also briefly red".)

3. There are 6 states. $2^2 = 4$ is too small. So, we need 3 bits for a maximum of $2^3 = 8$ states.

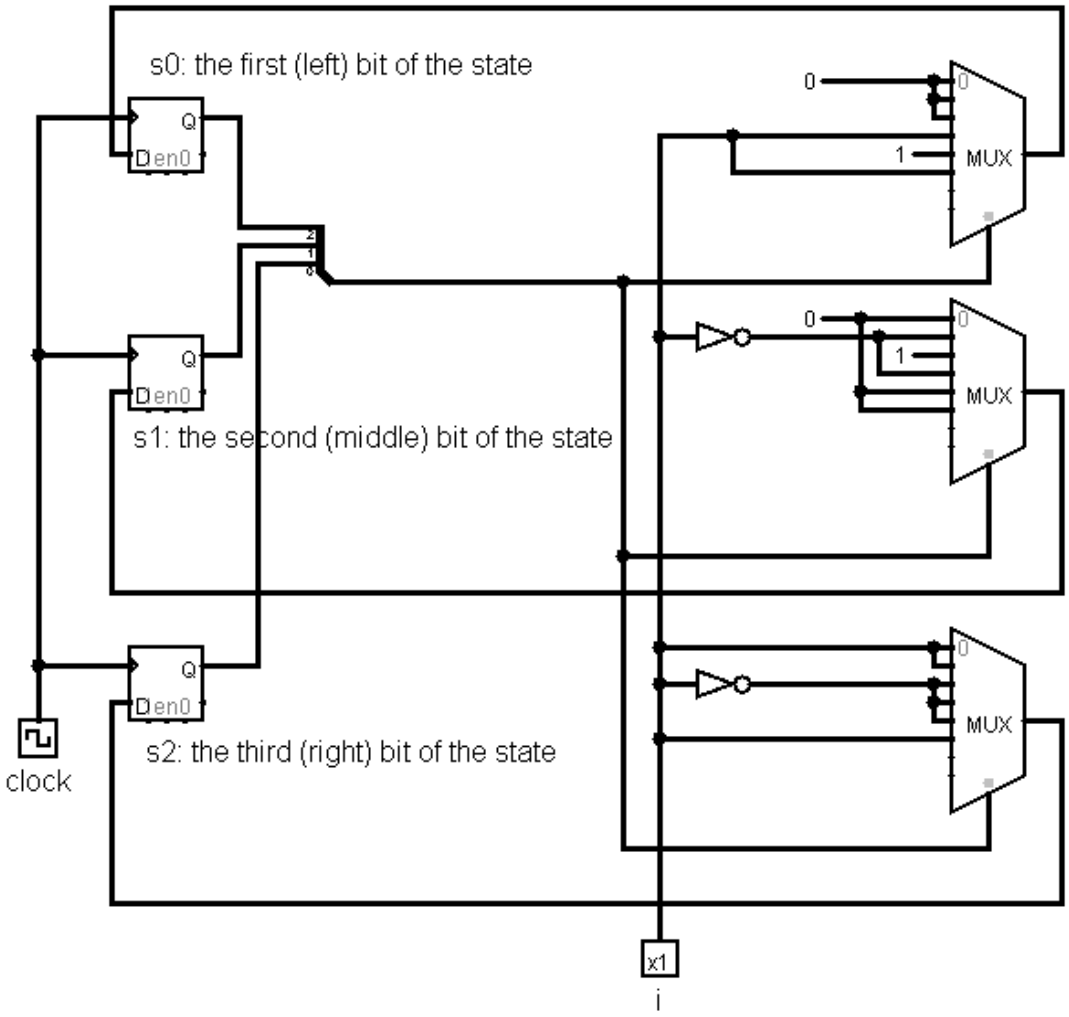
4. There are 2 possible input values. One bit gives us $2^1 = 2$ possible values, which is sufficient.

5. We start with a truth table showing the state transitions. The left side of the table (states and inputs) is just all possible combinations of a state the DFA is in and an input it receives. The right side (the next state) records the transition the DFA tells us to take if we get that input letter in that state:

State	Input	State (as s0 s1 s2)	Input (as i, binary)	Next state (in binary)	Next state (name)
`green`	`s`	0 0 0	0	0 0 0	`green`
`green`	`t`	0 0 0	1	0 0 1	`yellow`
`yellow`	`s`	0 0 1	0	0 1 0	`red0`
`yellow`	`t`	0 0 1	1	0 0 1	`yellow`
`red0`	`s`	0 1 0	0	0 1 1	`red1`
`red0`	`t`	0 1 0	1	0 1 0	`red0`
`red1`	`s`	0 1 1	0	0 1 1	`red1`
`red1`	`t`	0 1 1	1	1 0 0	`red2`
`red2`	`s`	1 0 0	0	1 0 1	`red3`
`red2`	`t`	1 0 0	1	1 0 0	`red2`
`red3`	`s`	1 0 1	0	0 0 0	`green`
`red3`	`t`	1 0 1	1	1 0 1	`red3`

   The multiplexer style we use for building circuits divides this into 6 separate problems, one for each possible current state. That makes the subproblems fairly simple. For example, the middle next state bit when the current state is `yellow` is the opposite of the input; so, it's $\sim$input.

   Working through all of those, we get a circuit like this:


s0: the first (left) bit of the state
Q
D en 0
0
1
MUX
Q
D en 0
0
1
MUX
s1: the second (middle) bit of the state
Q
D en 0
0
MUX
s2: the third (right) bit of the state
clock
x1
i