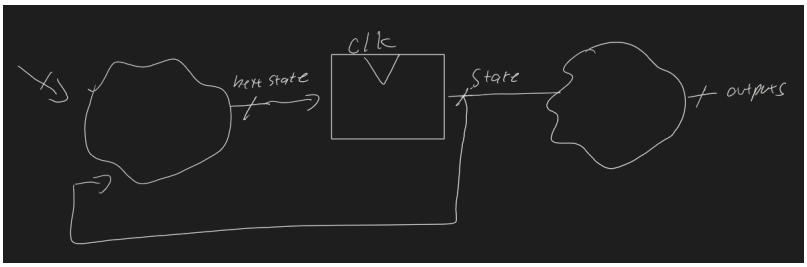
6 Finite State Machines and Timing

We have a basic Sequential circuit below:



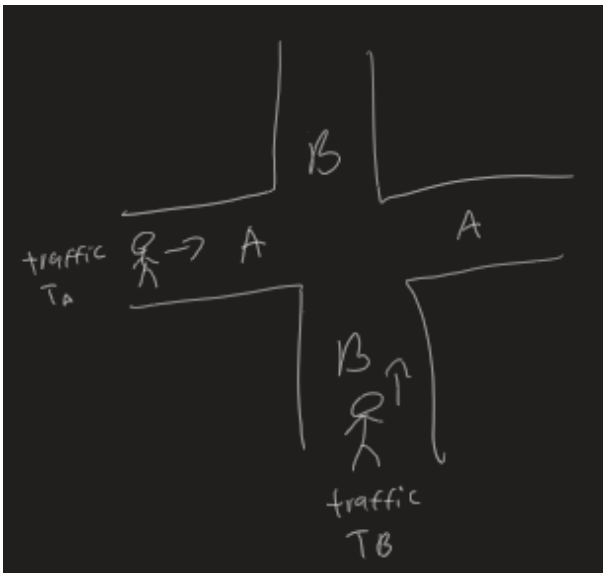
To handle these:

1. Identify inputs and outputs
2. Sketch a state transition diagram
3. Write the state transition table and output table
4. Select state encodings
5. Rewrite tables using encodings
6. Make Boolean Equations
7. Sketch the Schematic

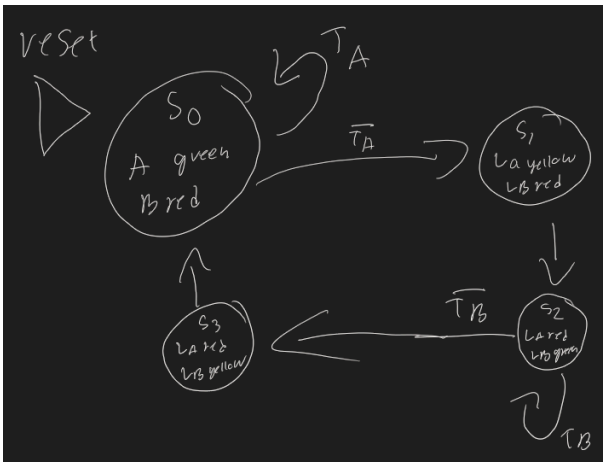
We can make a state transition diagram for finite state machines, where every state has an output for each possible input.

Suppose we have a 4 way intersection, where students are walking either north-south or east-west. We want to make traffic lights so that they don't collide.

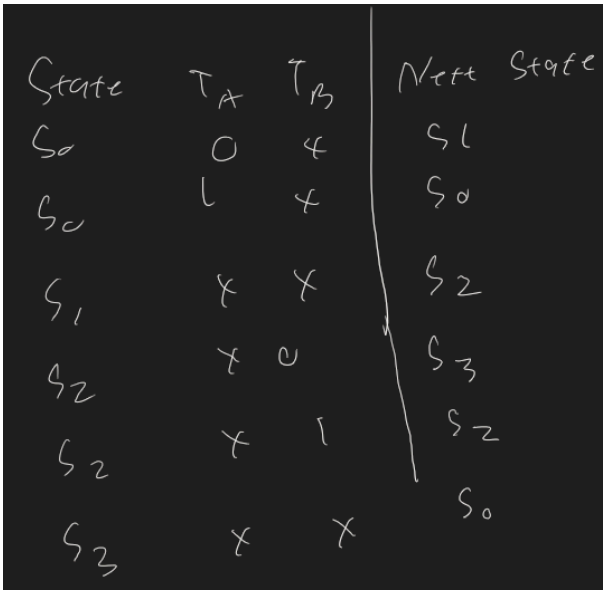
We can make a diagram of the situation, and then decide what we want the behavior to be:



We will start with Light A on, and if there is traffic then keep it on. When people stop going/we decide to move on, then well turn it yellow and then red and activate the other light. We'll follow in a cycle. Lets draw the state transition diagram:



We can make a table that shows this state transition:



We can correspond each state with an output :

$$\text{Output table}$$

$$\begin{bmatrix} \text{State} & | & L_a & |L_b \\ s_0 & | & 0 & R \\ s_1 & | & Y & R \\ s_2 & | & R & G \\ S_3 & | & R & Y \end{bmatrix}$$

We also have to be able to encode our position state and our light output state in binary, so we make a table for each of those:

$$\text{State Representation} \quad \begin{bmatrix} s_0 & | & 0 & 0 \\ s_1 & | & 0 & 1 \\ s_2 & | & 1 & 0 \\ s_3 & | & 1 & 1 \end{bmatrix}$$

We must also represent the lights' states:

$$\begin{bmatrix} \text{green} & | & 00 \\ \text{yellow} & | & 01 \\ \text{red} & | & 10 \end{bmatrix}$$

We can now write out a full table of our current state to the next state, and our current state to the outputs.

We have

$$\text{state transition table}$$

$$\left[\begin{array}{cccc|cc} s_1 & s_0 & t_a & t_b & s_1' & s_0' \\ 0 & 0 & 0 & x & 0 & 1 \\ 0 & 0 & 1 & x & 0 & 0 \\ 0 & 1 & x & x & 1 & 0 \\ 1 & 0 & x & 0 & 1 & 1 \\ 1 & 0 & x & 1 & 1 & 0 \\ 1 & 1 & x & x & 0 & 0 \end{array}\right]$$

This takes our current state and the inputs to see what our next state will be.
We can now write the boolean equations for this

$$s_1' = \bar{s_1}s_0 + s_1\bar{s_0}\bar{t_b} + s_1\bar{s_0}t_b = \bar{s_1}s_0 + s_0\bar{s_1} = s_1 \oplus s_0$$
$$s_0' = \bar{s_1}\bar{s_0}\bar{t_a} + s_1\bar{s_0}\bar{t_b}$$

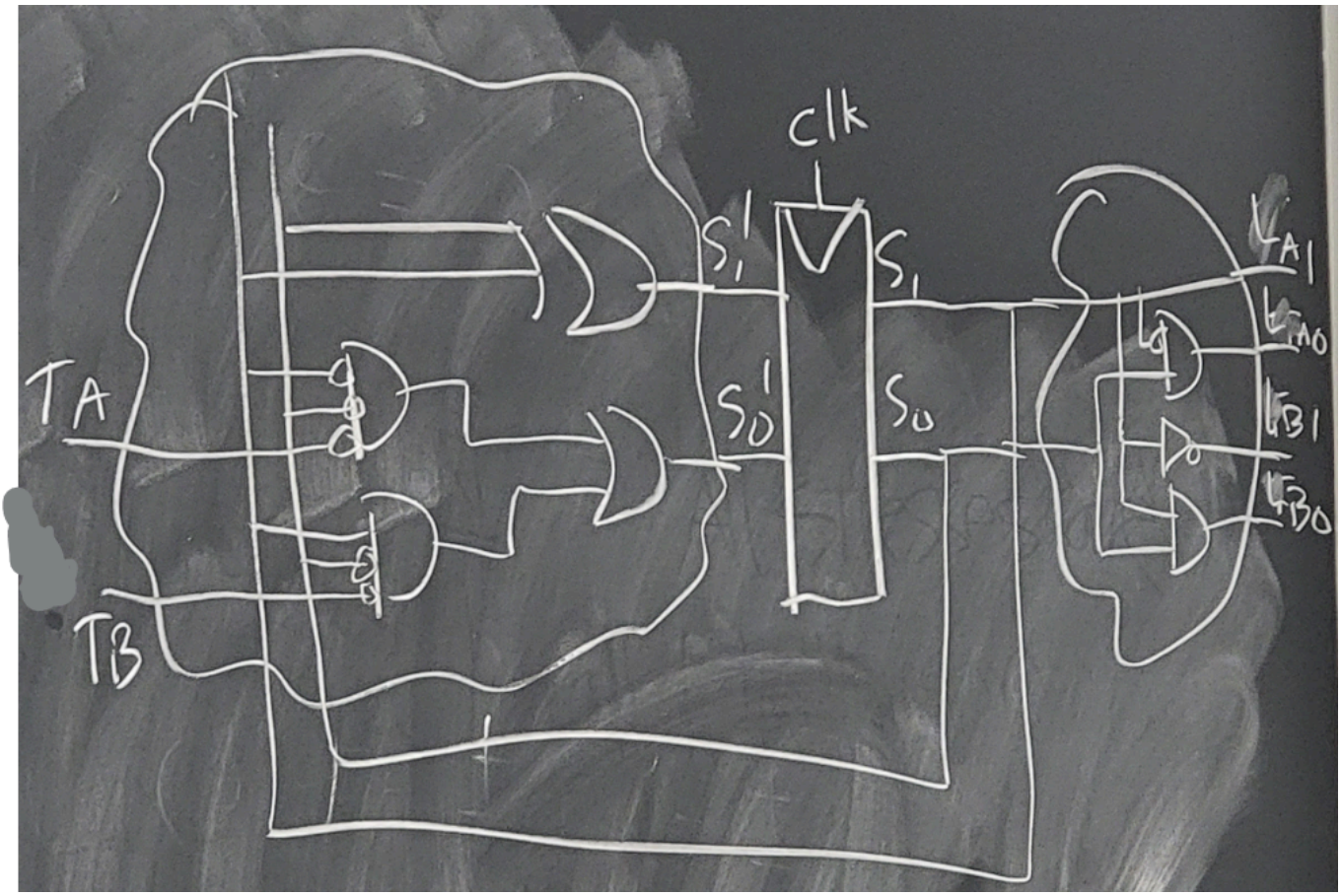
We can also write out our state to output table

$$\left[\begin{array}{cc|cccc} s_1 & s_0 & la_1 & la_2 & lb_1 & lb_0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

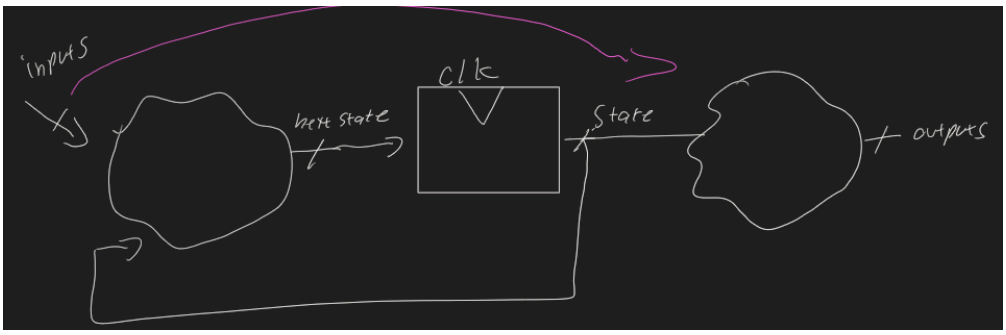
This gets us the Boolean equations

$$
\begin{aligned}
L_{a1} &= s_1 \\
L_{a0} &= \bar{s_1} s_0 \\
L_{b1} &- \bar{s_1} \\
L_{b0} &= s_1 s_0
\end{aligned}
$$

We can now start to draw our hardware from these equations.

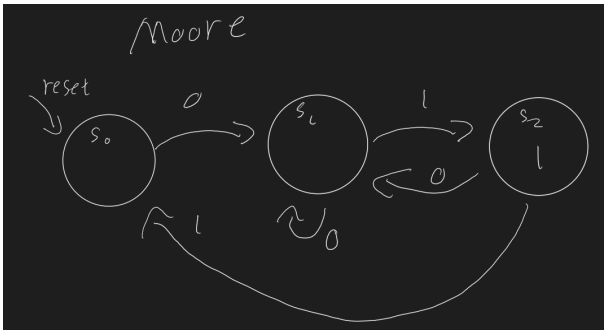


For this type of FSM, we have to wait for the state to update and then have an output. This is called a Moore. However, we can rework this to not involve as much state, where the output depends both on the state and the current inputs directly. This is called a Mealy, which would look like:



This is our previous image but where we add a stream from the input to the logic block that is after our state logic.

Lets imagine a snail that really likes reading 0 1, and it will smile if it has read that. We can compare the Moore and Mealy circuits with this example.



Lets write the state, input, and next state table, and the state-> output table:

$$\left[\begin{array}{cc|c|cc} s_1 & s_0 & A & s_1' & s_0' \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{array}\right]$$

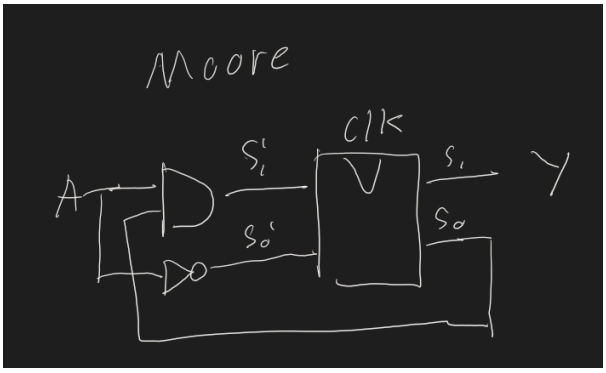
With the state to output table:

$$\left[\begin{array}{cc|c} s_1 & s_0 & y \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & x \end{array}\right]$$

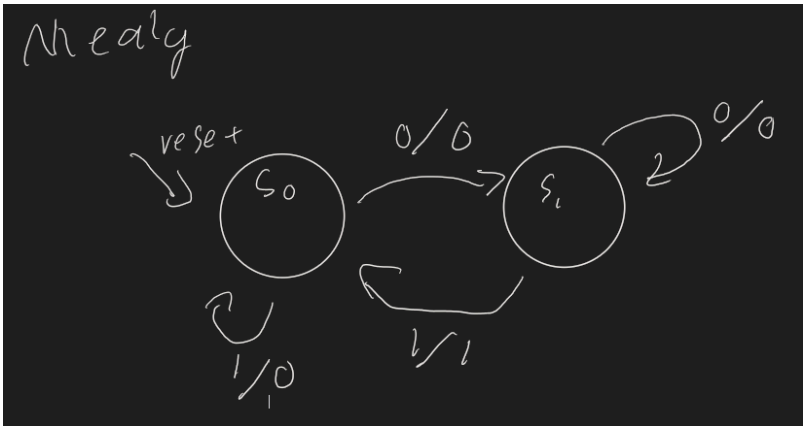
We can now make the boolean equations:

$$s_1' = \bar{s_1} s_0 A = s_0 A$$
$$s_0' = \bar{s_1} \bar{A} + s_1 \bar{A} = \bar{A}$$
$$y = S_1$$

We can make the circuit off of this:



We could rewrite this as a Mealy and see how it is more efficient. We write the outputs with the state transitions as input/output:



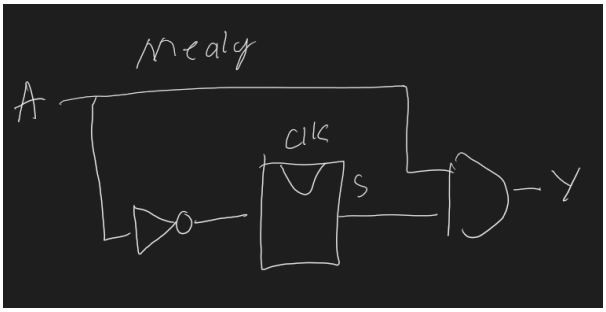
We can write this table far shorter, as:

$$\begin{bmatrix} S & | & A & | & S' & Y \\ 0 & | & 0 & | & 1 & 0 \\ 0 & | & 1 & | & 0 & 0 \\ 1 & | & 0 & | & 1 & 0 \\ 1 & | & 1 & | & 0 & 1 \end{bmatrix}$$

This gives us the Boolean equations

$$s_1' = \bar{A}$$
$$Y = SA$$

This gives a simpler circuit that only needs one memory block without any external back loops:



Now, lets talk about...

Timing

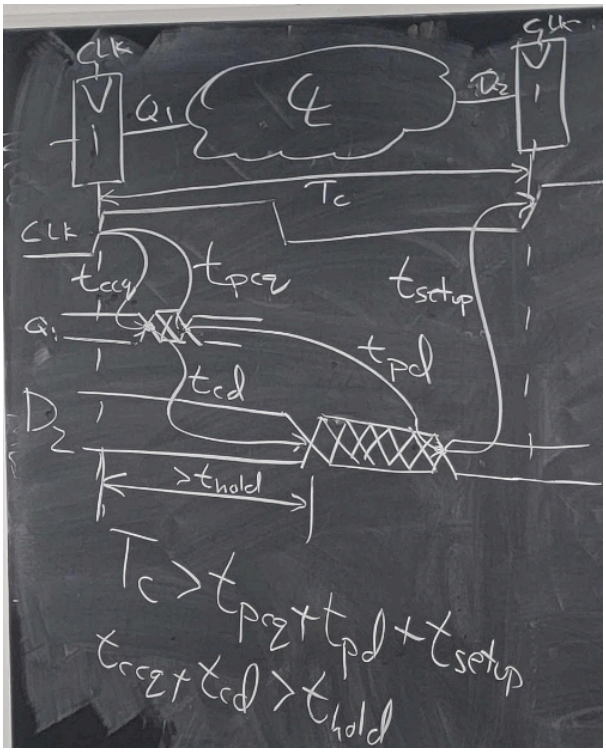
Recall a D-latch, where an input D will be fed to the memory when the clock has a rising edge. It is easy to read if the input changes sufficiently far from the clock edge, but if the input changes really close to the clock cycle than we could get a blur of the signal that could go either high or low.

Dynamic Discipline

We have a forbidden zone by the clock discipline where D is not allowed to change in. D is free everywhere outside of that time window close to the clock rising. We call the time before the clock edge rises t_{setup}, and the time after the clock edge rises before we allow D to change again t_{hold}.

We can picture a general sequential circuit where we have a flip flop and some combinational logic going into the next flip flop.

Lets call the clock period T_c, and define variables related to the shortest and longest time things can happen.



What happens if our chip gets hot, and we aren't really able to settle each value fast enough? We should lower the clock rate, and then it's okay. We can also speed up if things are working well. What if D_2 changes too soon? If we mess up the hold time, then there is nothing we can do to fix it - we have to redesign the chip. *WE CAN'T VIOLATE THE HOLD TIME!*

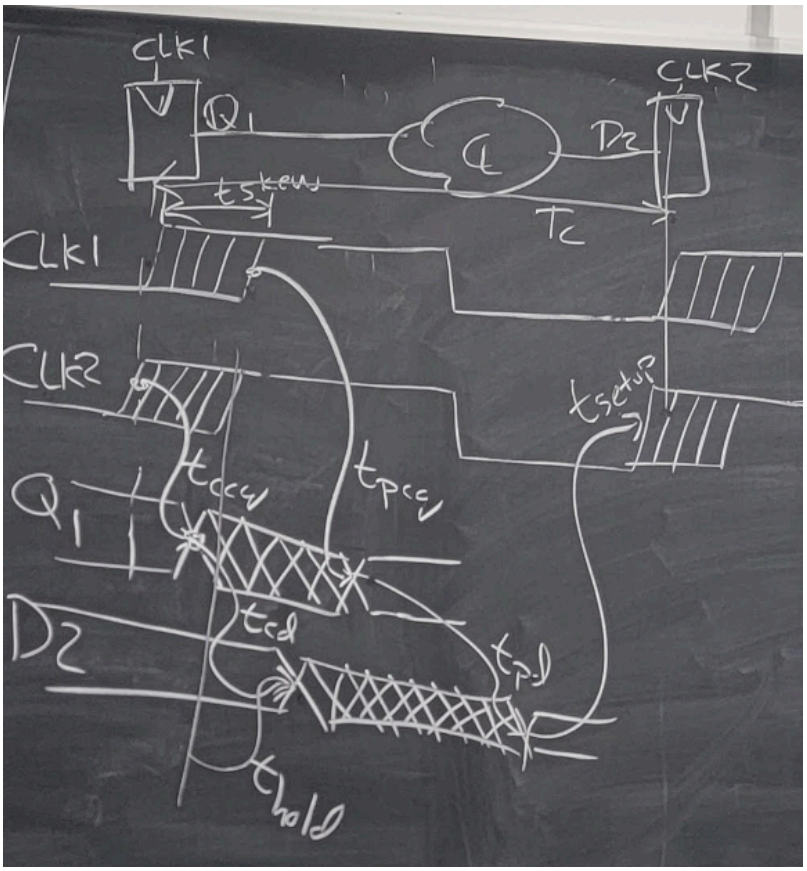
We want our flip flop to be slow enough that the contamination delay is longer than the hold time. If we put a straight wire between flip flops, then the contamination could travel before the hold time of our next flip flop. Yikes! In that case we would have to have buffers to slow down our signal!

Also, given the real world, even if we want a clock to be the same everywhere, the same clock signal will arrive at different positions at

different times. We're going so fast that the 'skew' time of each clock can matter.

We can draw uncertainty in our clock signal, which adds to the uncertainty in each of our parameters.

Because of the uncertainty we have to run our clocks slower.



$$T_c > t_{skew} + t_{pcq} + t_{pd} + t_{\text{setup}}$$
$$t_{ccq} > t_{\text{hold}} + t_{\text{skew}}$$