

CSE 2600

Intro. To Digital Logic & Computer Design

Bill Siever
&
Michael Hall

Announcements

- Homework 1: Returned in a few days
 - Can request “regrade” of problems for ~1 week on assignments
Not accepted after that.
- Homework 3A Posted / Due Sunday at 11:59pm
 - Dropboxes posted Thursday
- Fee: \$50 will be applied to student accounts in coming week(s)

Chapters 1-2: Combinational Logic

- (Purely) combines current inputs to produce output
 - Doesn't depend on past inputs
 - Can be represented with a simple table
 - One-way: Doesn't have any feedback paths from output back to inputs

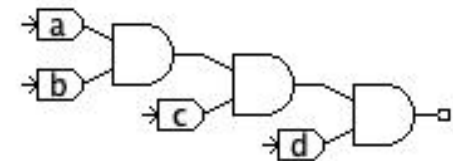
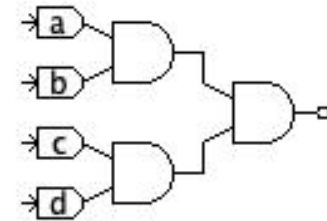
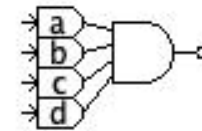


Big-Picture So Far...

- We can build reliable, complex machines that work in binary
 - Yay Transistors!
- We can represent almost any information via binary encodings
 - Integers: We can even easily do arithmetic operations (+, -, etc.)
 - Letters/characters, “real” numbers, etc.

Big-Picture So Far...

- Gates: We can represent some primitive binary “machines” via a symbolic notation. Structure shows the flow of information.



Big-Picture So Far...

- Almost any simple mapping (function) can be represented via a table
- Show the “output” for all possible combinations of inputs

INPUTS (IN BINARY)	OUTPUTS (IN BINARY)
...	...

Big-Picture So Far...

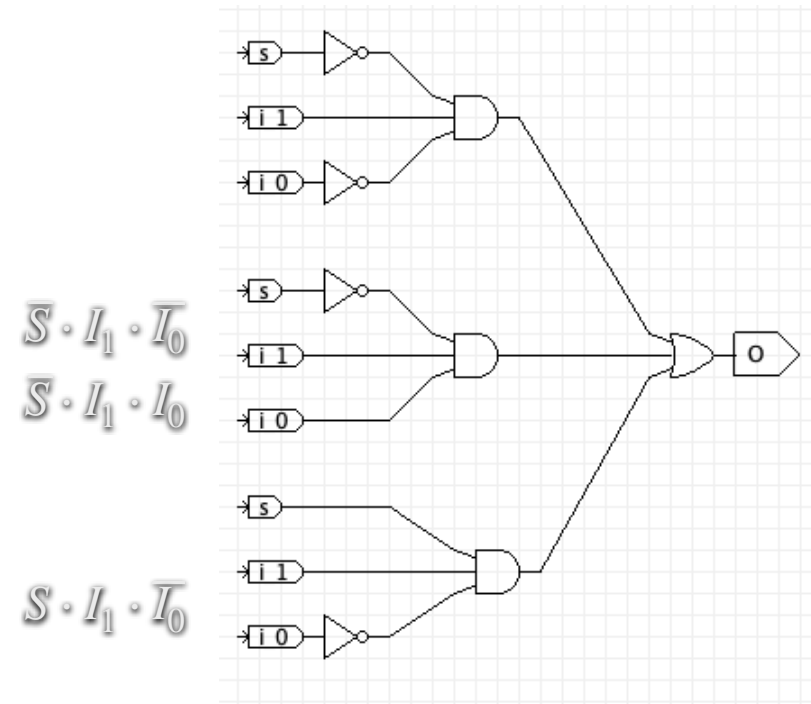
- Any full table can be converted to a boolean logic equation

Inputs		Output
S	I	O
0	0	
0	1	
0	2	
0	3	
1	0	
1	1	
1	2	
1	3	

Big-Picture So Far...

- Any full table can be converted to a sum-of-products (SOP) boolean equation

Inputs			Output
S	I ₁	I ₀	O
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



Big-Picture So Far...

- Any mapping/function that *combines current inputs to produce the output* could be described as a table
 - The table could be turned into equations
 - The equations could be made into machines
- Works on small problems, but table is too large for many problems
 - Ex: Add 2, 32-bit numbers:
 - Table with 64 columns of input
 - 18446744073709551616 rows in that table...

Studio 2B Recap

- Karnaugh Maps
 - Cells represent minterms
 - Combining cells is application of Theorem 10: $(B \cdot C) + (B \cdot \overline{C}) = B$
 - Only works if rows and columns are in Gray code order

Studio 2A: Example

- Small tables can be represented via K-Maps

Inputs			Output
S	I ₁	I ₀	O
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

	I			
S	00	01	11	10
0				
1				

Studio 2A: Example

- K-Maps represent SOP equations

	I			
S	00	01	11	10
0	0	0	1	1
1	0	0	0	1

Studio 2B Misc.

- Negation Notation
 - $\overline{AB}$ vs. $\overline{A} \cdot \overline{B}$
 - $\overline{A} \cdot \overline{B} = \overline{AB}$
- Glitches: ... a single input transition can cause **multiple** output transitions.
 - Often a result of different delays in a path
- Packages & Packages

Goal: Store Data

Examples

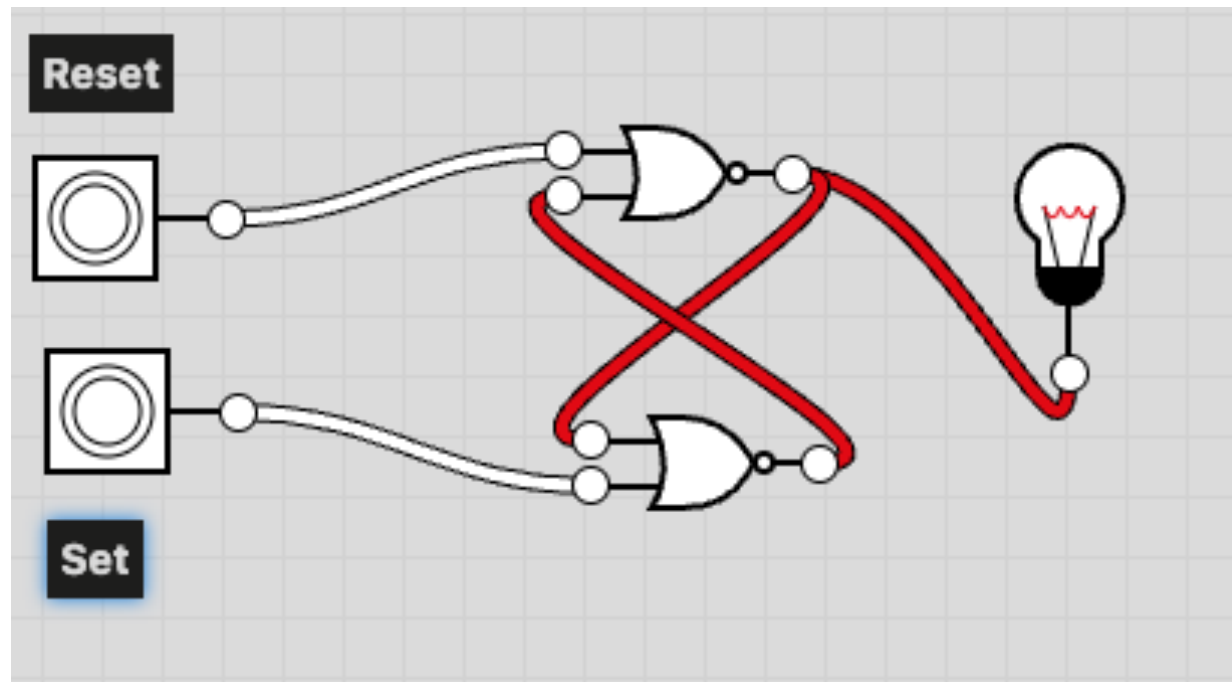
- Bistable Example 1: Inverters
- Bistable Example 2A: Inverters & some control
- Bitable Example 2B: Inverters, control, and ...

Story Time: Latches+

Stable, Reinforcing Setup: SR Latch

- On-line Demo: <https://logic.ly/>
 - Bistable: Two stable configurations
- Goal: Met!
 - S=Set, R=Reset

SR Latch

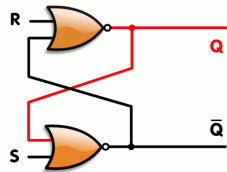


Goal: Store Data

- Set/Reset is *inconvenient*
 - We want something like, $\text{data} = X$, where x is 0 or 1 (store X , not “set or reset data based on X ”)
 - We want to store X in data *when we're ready to!*
- *Clock (Clk)*: Indicates *when* we want to change the data

D-Latch

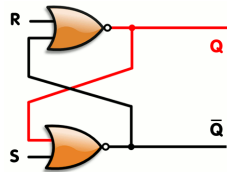
- Start with SR Latch
- Describe Desired Behavior (of output, Q)



CLOCK	DATA	Q
0	0	(Unchanged)
0	1	(Unchanged)
1	0	0
1	1	1

D-Latch

- Start with SR Latch

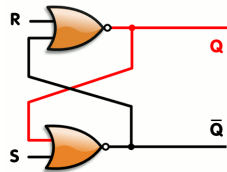


- Describe Desired Behavior (of output, Q)

CLOCK	DATA	Q
0	0	
0	1	
1	0	0
1	1	1

D-Latch

- Start with SR Latch

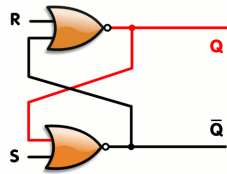


- Describe Desired Behavior (of output, Q)

CLOCK	DATA	Q
0	0	
0	1	
1	0	RESET
1	1	SET

D-Latch

- Start with SR Latch

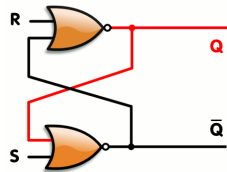


- Describe Desired Behavior (of output, Q)
- Just combinational logic

CLOCK	DATA	Q
0	0	
0	1	
1	0	RESET
1	1	SET

D-Latch

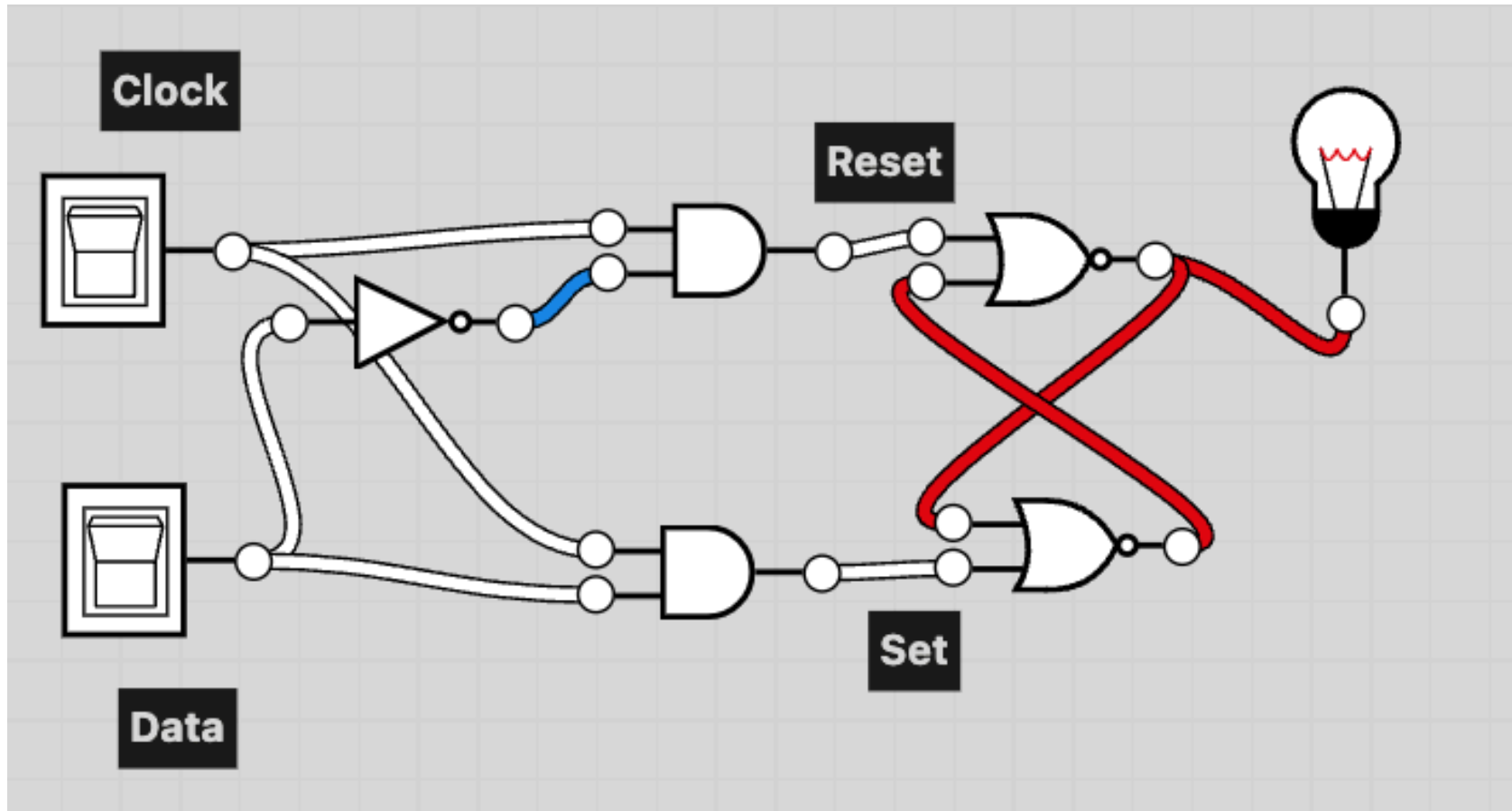
- Start with SR Latch
- Describe Desired Behavior (of output, Q)
- Just combinational logic
- Reset = Clock * /Data
Set = Clock * Data



CLOCK	DATA	Q
0	0	
0	1	
1	0	RESET
1	1	SET

Updates: D-Latch

D-Latch



D-Latch

- “Latches on” to last data value when clock goes low
- Is sensitive to the level of the clock
- Is “transparent” when the clock is high

Goal: Store Data

- D-Latch is *still* a bit inconvenient
- We'd like something like a (simple) camera
- The instant shutter is “pressed” we capture data *at that exact instant* (no transparent phase)

Flip-Flop

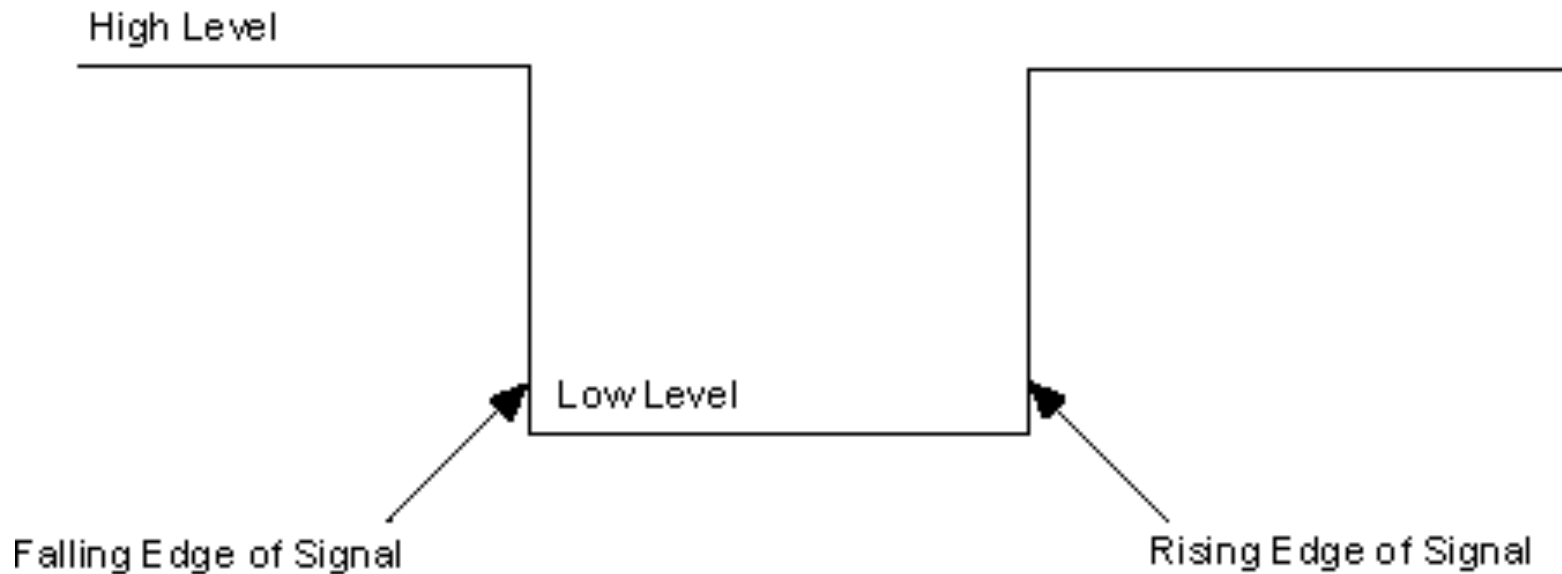


<https://openclipart.org/detail/288726/flip-flops-4>

D Flip-Flop

- Two D-Latches with clocks in opposite states (via an inverter)
 - First stage: Transparent when clock is Low
 - Second stage: Transparent when clock is High
- Effect: Capture D at precise instant clock goes from low to high
 - I.e. the clock EDGE
 - Edge triggered. Specifically, Rising Edge Triggered

Signal Edges

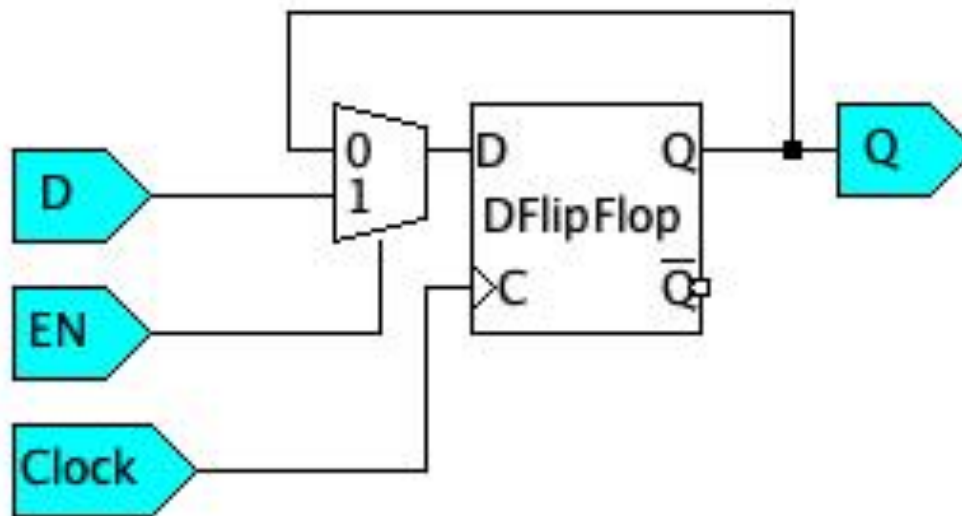


https://www.ni.com/docs/en-US/bundle/ni-hsdio/page/hsdio/fedge_trigger.html

Updates: D-Flip-Flop

“Enable”

- We may want to have two things control timing: the clock and an enable
- Ex: $X[0] = 1$ (in a program) . We only want to modify X *when that line runs*.



Chapter 3: Sequential & Synchronous Logic

- Need to know sequence of inputs
 - Can't be represented with a *simple* table of just inputs and outputs
(Possibly a complex table that includes some representation of history of inputs and outputs)
- Text: "Some sequential circuits are just plain kooky"

***Synchronous*^{*} Sequential Circuits**

- Are *synchronized* by a common clock
- Uses registers (D Flip Flops)
- Mix of registers and combinational logic
- Cycles in circuit include at least one register
- Goal: Impose predictable behavior!

* eliminates the “kooky”

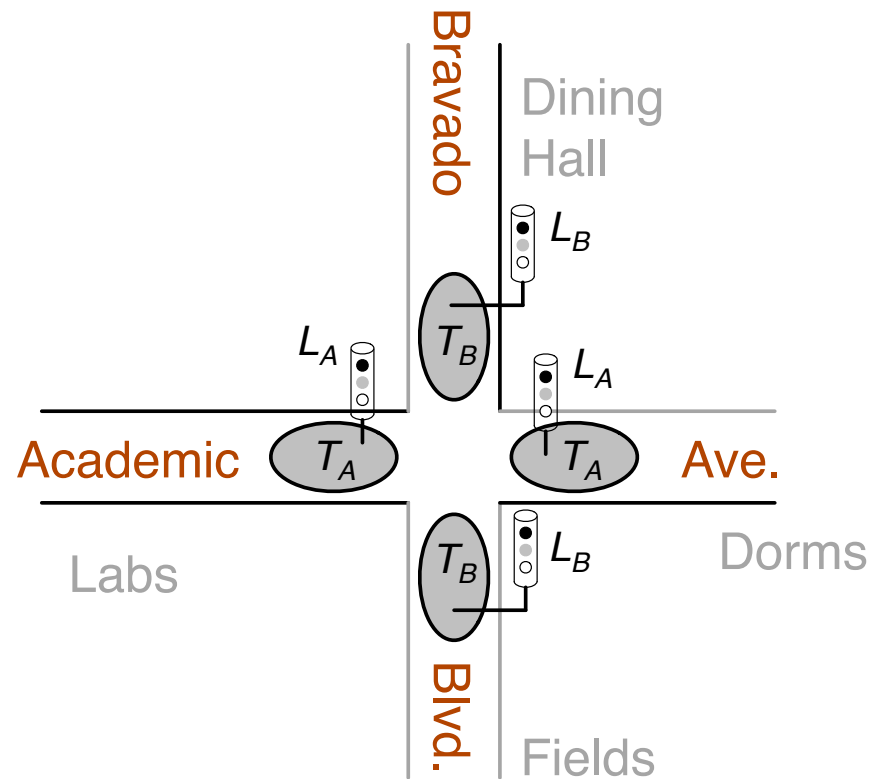
Finite State Machines

- State: A condition of being
- Finite: Er. Finite
- Real machine has real-world limitations: $k \times$ D-latches
- k D-Latches means $\leq 2^k$ states (finite)

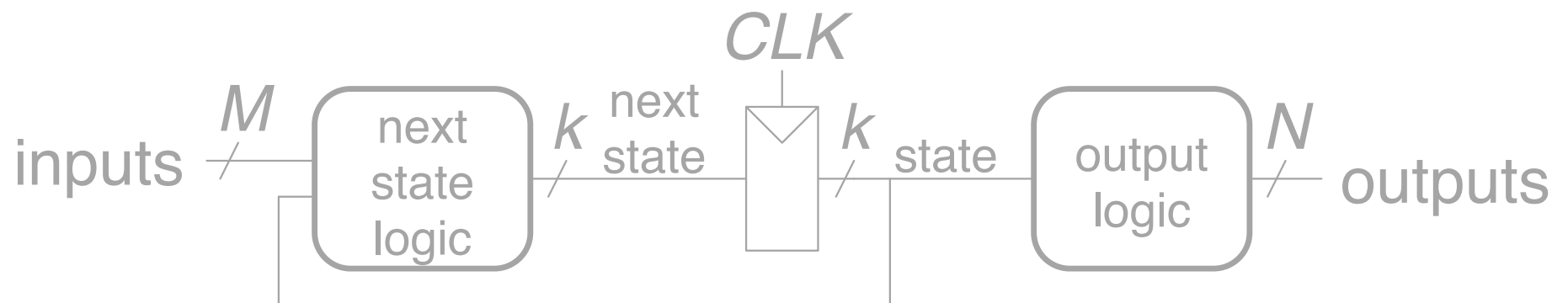
FSM Applications

- Things with modes or sequences of steps. Examples:
 - Washing Machine (fill, agitate, rinse, spin)
 - Stop lights & Traffic control: Green, Yellow, Red
 - Locks: Locked & unlocked
 - Computer programs: Playing game vs. on menu
 - Elevator controls (state = floor)
 - ...

Book Example Variation: Stop Light



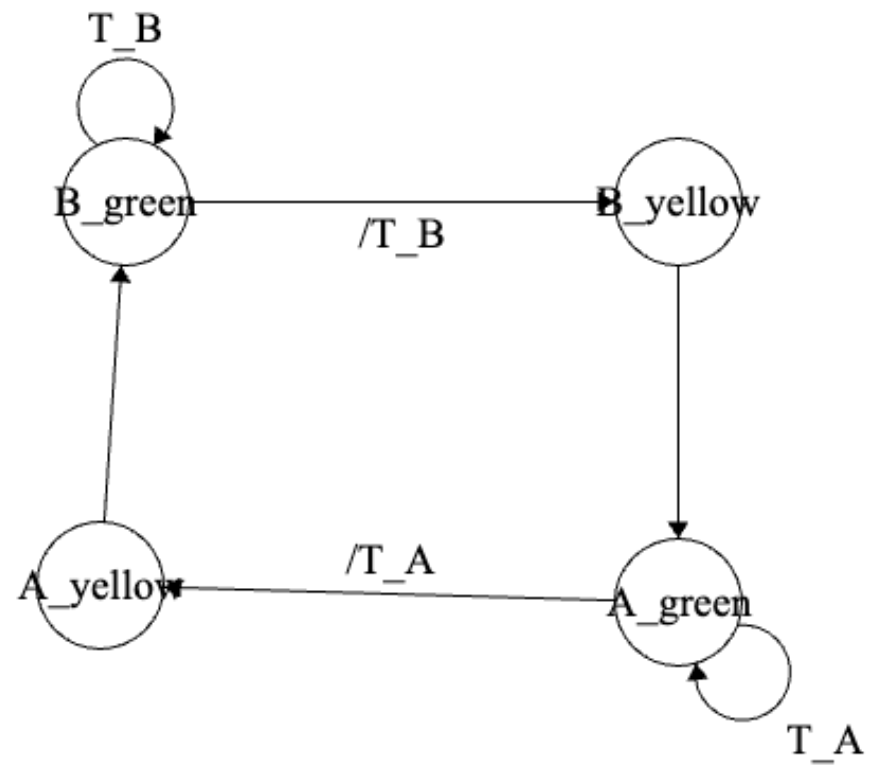
FSM: Moore Machine Structure



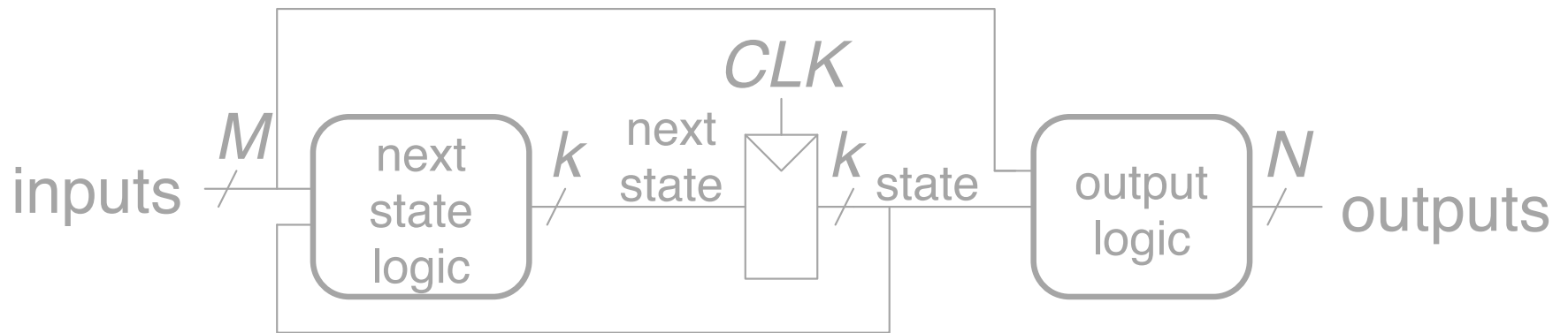
Background

- Clock is 5s: minimum time in a state
- Need to describe behavior over time
- State diagram forms
 - FSM Designer: <https://wilsonem.github.io/fsm/>
- Example: Variation on textbook;
Different output signals and “one hot” encoding (Completed)

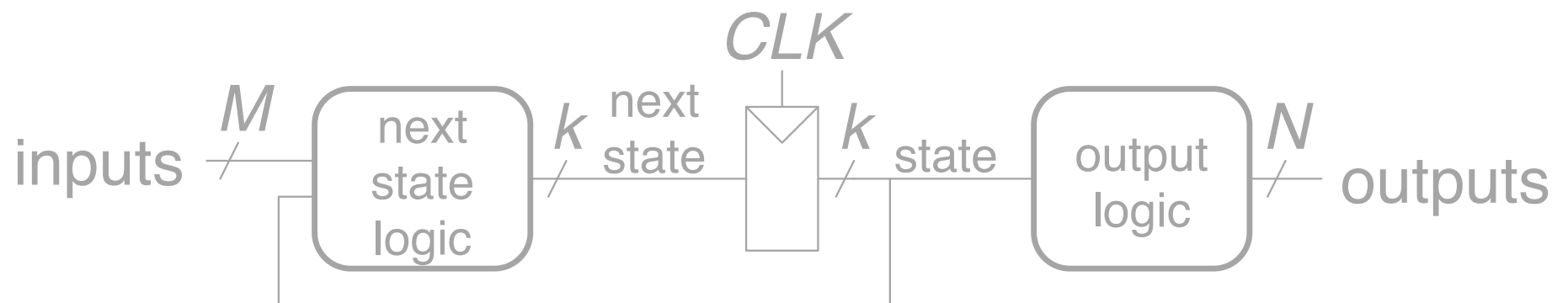
Partial JLS Implementation



FSM: Mealy Machine



FSM: Moore Machine Structure



Questions

- [State Machine Encoding Choices]
Memory vs. Logic: They can impact complexity of combinational logic.
(Typically One-hot: more memory, but simpler logic))
- [FSMs? Mealy? Moore?]: More (Moore?) next time
- [I don't get/understand Latches/Edges/Levels/Flip-Flops/etc.]
- [Which memory things are important]: (Rising) Edge Triggered D-latch.
(Most others were just part of the journey to it)

Questions

- [What about async stuff / parallel? Does it matter? Where is it used?]
 - Yes, matters. Is often critical in high performance systems (real time computations, handling volumes of streaming data, etc.)
- Will we make FSMs? Yep.
- Are FSMs part of modern computers? Yes —they often handle the control of CPU operations. We'll see this later.
- Is this “clock” related to the computer clock? How does it relate to computer performance? It's the same basic idea. Much of a computer is controlled by a synchronous machine. The clock speed is based on propagation delay and dictates the speed of computations.

Next Time

- Studio