0x09 - Finite State Machines

ENGR 3410: Computer Architecture

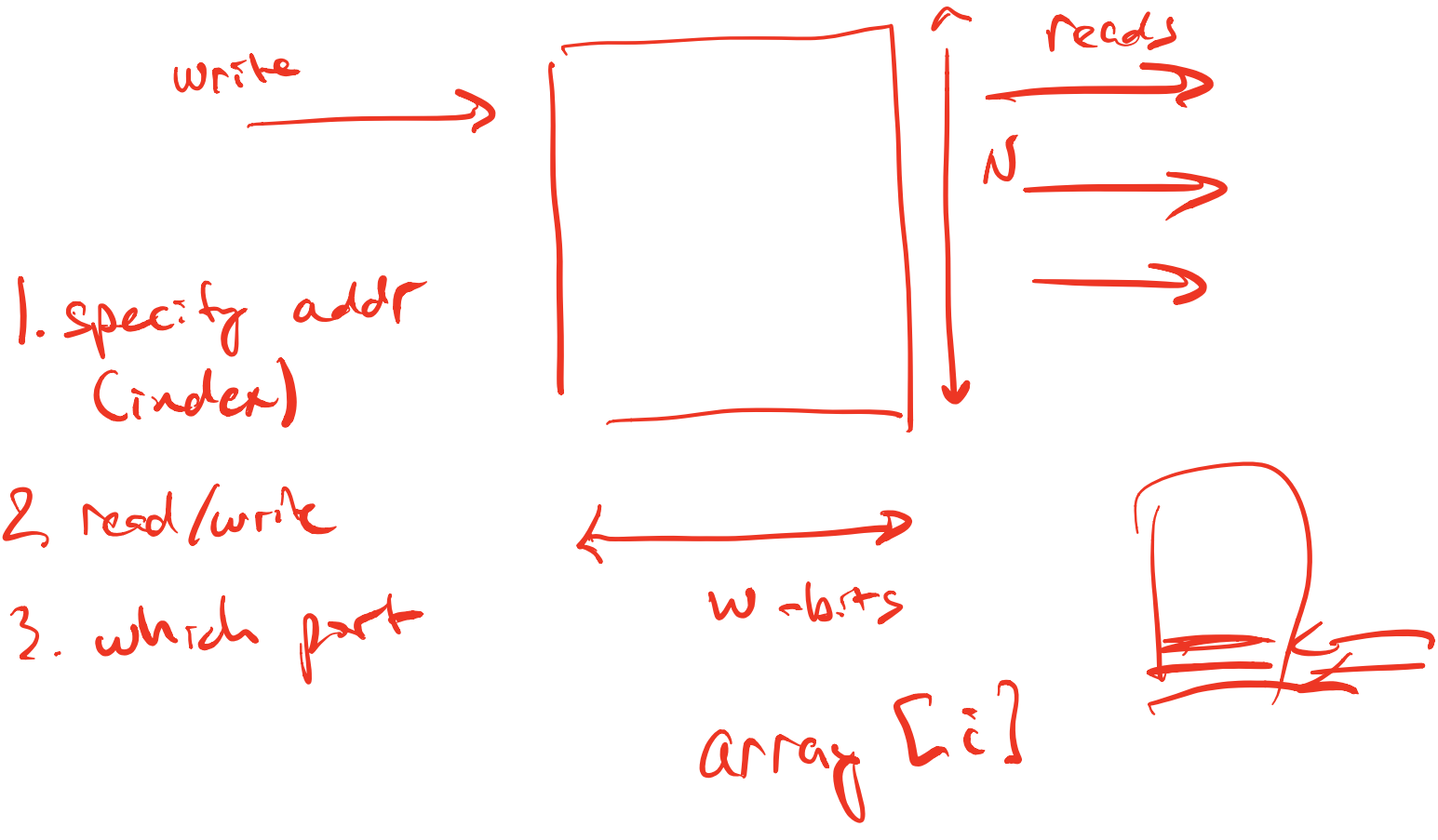
Jon Tse

Fall 2020

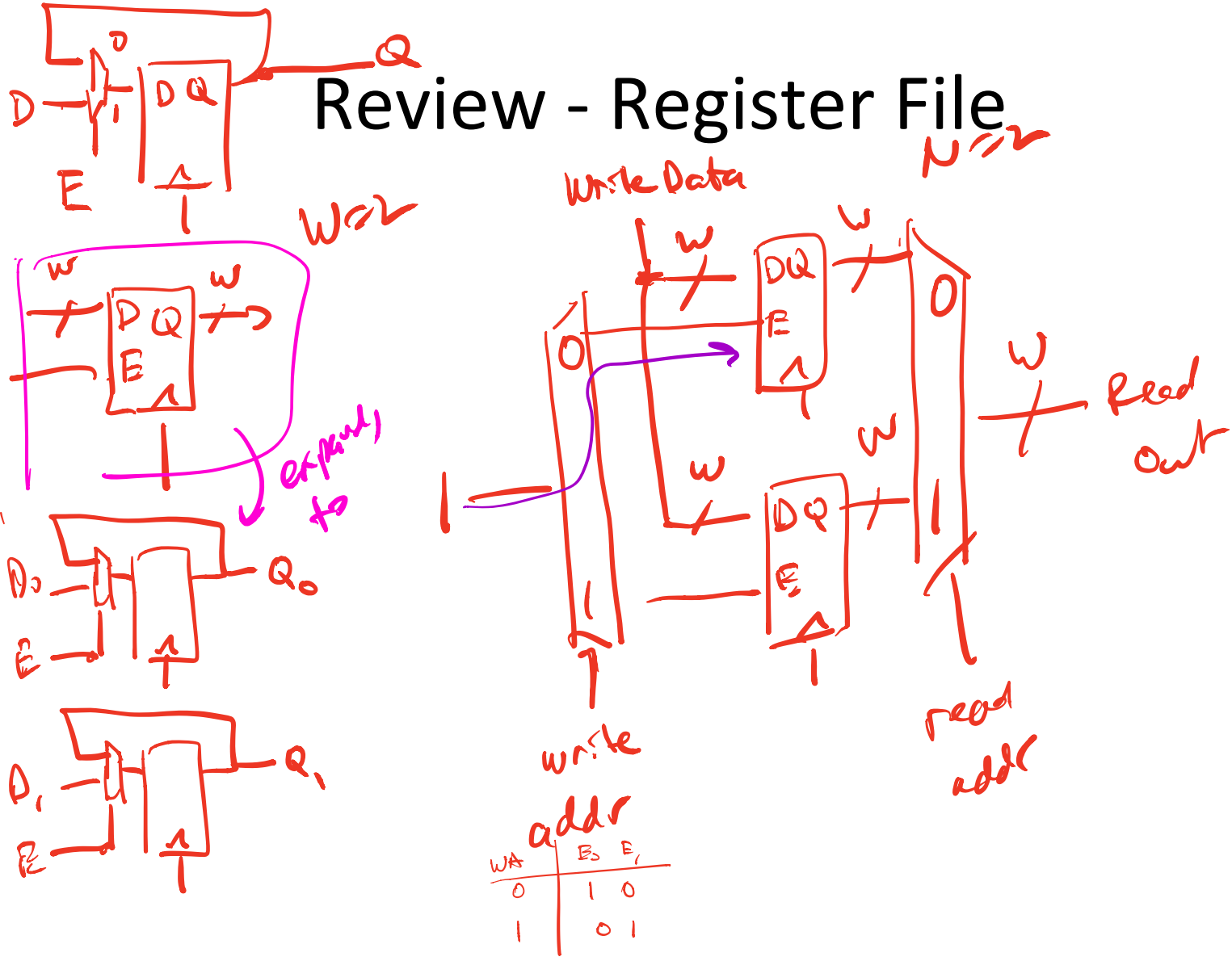
Housekeeping

- Reminder, HW5 due *Tuesday*
- Quick look at Calendar
- Midterm is individual, similar style to HW5

Review - Register File



Review - Register File

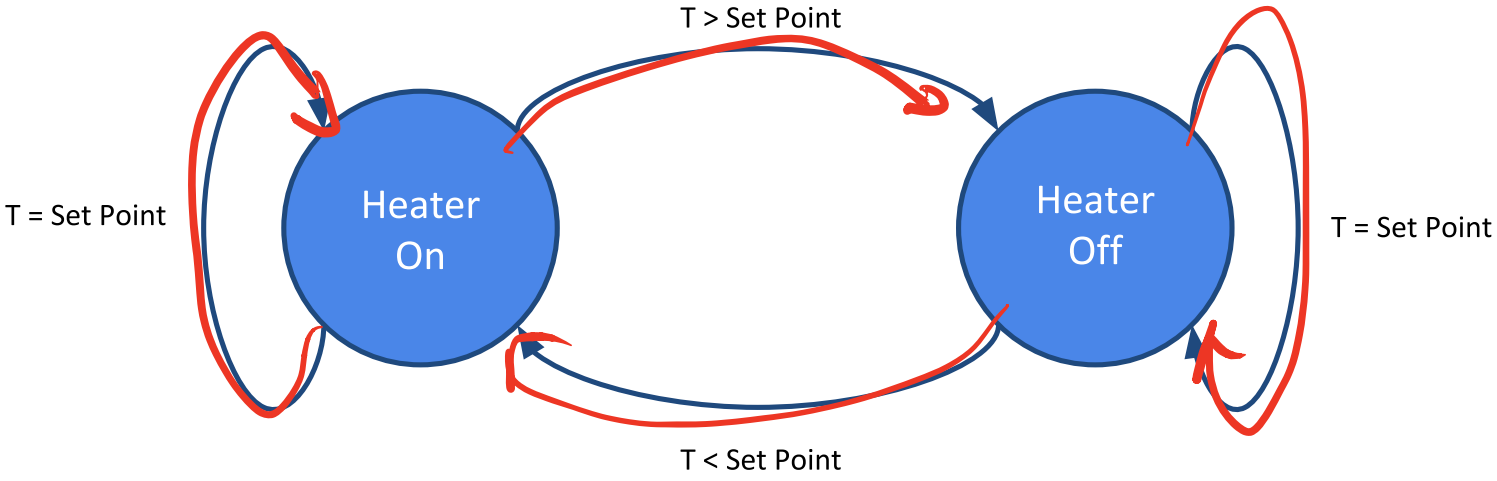


Review - Finite State Machines

Collection of States (Node) and Transitions (Edge)

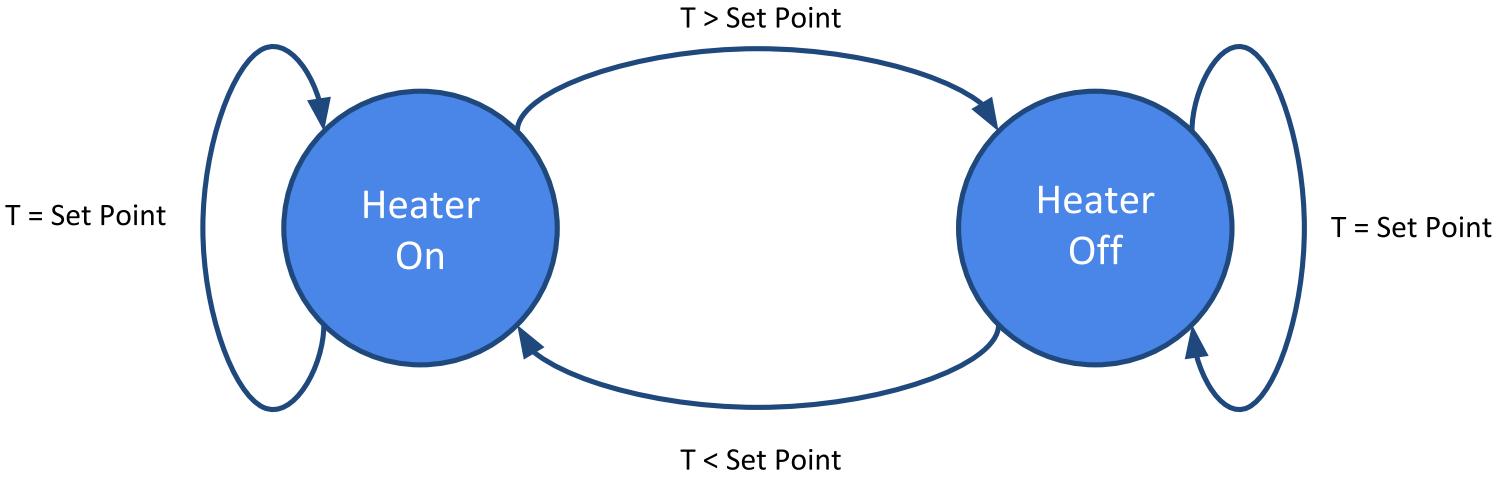
Transitions are guarded by conditions.

Example: Heater



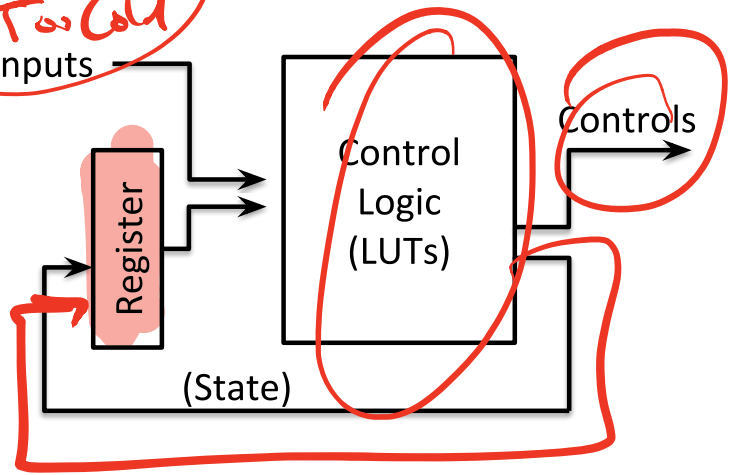
Review - Finite State Machines

- In Computer Architecture, FSMs:
  - Usually transition on a clock edge
  - Are Complete
    - All states define transitions for all inputs
  - Are deterministic



FSM Implementation

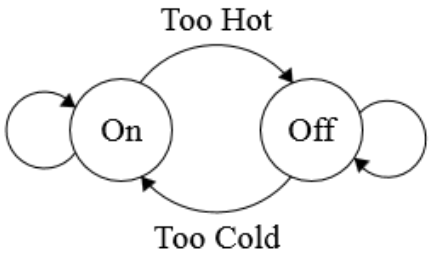
- Register to hold current state
- Wires to provide inputs (arguments)
- Look Up Table(s) to map transitions



Current State	Inputs	Resulting State
Heater Off	Too Cold	Heater On
Heater Off	---	Heater Off
Heater On	Too Hot	Heater Off
Heater On	---	Heater On

Transitions -> LUT

Current State	Inputs	Resulting State
Heater Off	Too Cold	Heater On
Heater Off	---	Heater Off
Heater On	Too Hot	Heater Off
Heater On	---	Heater On



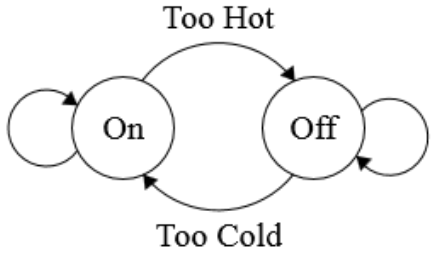
input

Out

State	Cold?	Hot?	Next State
Off	Yes	Yes	X
Off	Yes	No	On
Off	No	Yes	Off
Off	No	No	Off
On	Yes	Yes	X
On	Yes	No	On
On	No	Yes	Off
On	No	No	On

LUT -> Circuit

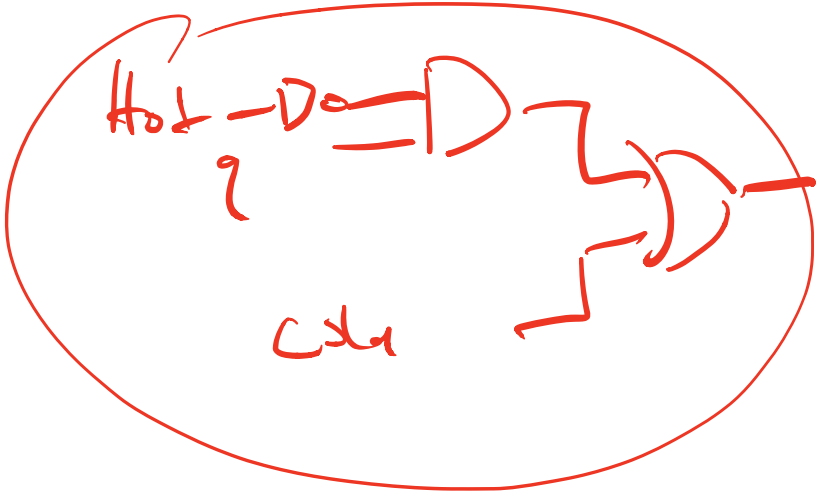
State	Cold?	Hot?	Next State
Off	Yes	Yes	X
Off	Yes	No	On
Off	No	Yes	Off
Off	No	No	Off
On	Yes	Yes	X
On	Yes	No	On
On	No	Yes	Off
On	No	No	On



curr state

next state

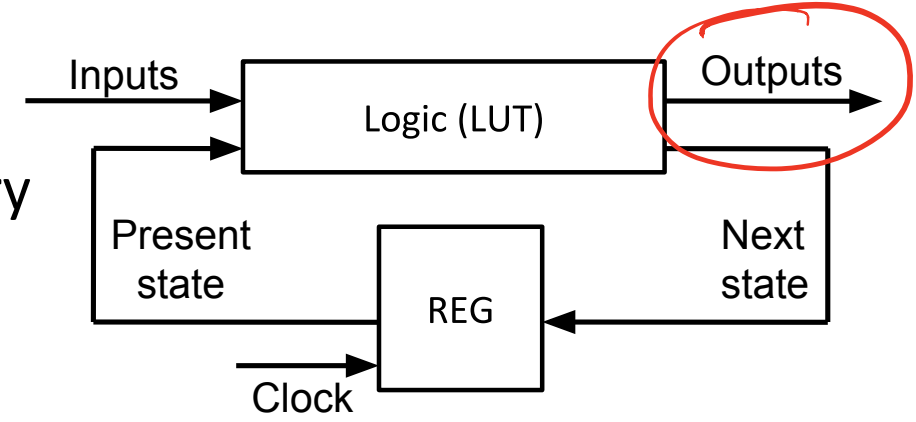
$$q_{NEXT} = (Cold) + \overline{(Hot)}q$$



Karnaugh Map

FSM Process Summary

- Design Requirements
- Define States
- Define Transitions
- Assign Values / Binary Codes
- Calculate Widths
- Make a LUT
- Connect Registers
- (Optional) Simplify



Binary Encoding

- States are stored as binary numbers

state = 1, state = 10

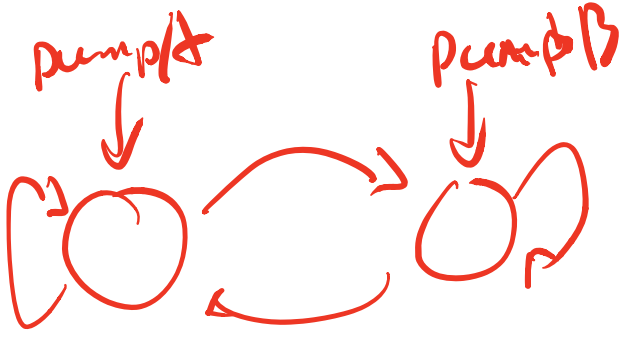
- Requires $\lceil \log_2 N \rceil$ registers for N states

- LUT requires a decoder (per usual)

One Hot Encoding

- States are stored as one bit being “hot”
- Requires N registers for N states
- LUT doesn’t require a decoder

Binary	One Hot
00	0001
01	0010
10	0100
11	1000

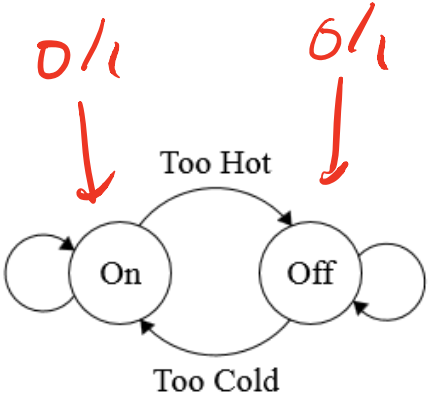


One Hot Construction

- LUT Method works:
  - N states = N LUT address bits (+ input bits)
  - This creates many “don’t care” LUT rows
    - 2^N rows, only care about N of them
- LUT Simplification
  - Same process as before
  - Apply LOTS of simplification

One Hot Construction

- Start with Flow Diagram
- Replace each state with a D Flip Flop
- D input is an OR gate
  - Each input is a transition

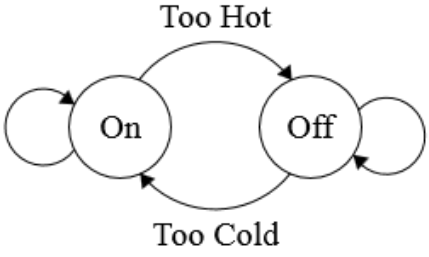


One Hot Construction

- Start with Flow Diagram
- Replace each state with a D Flip Flop
- D input is an OR gate
  - Each input is a transition

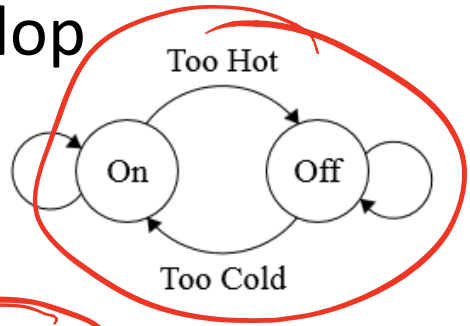
$State_{ON} = (\overline{Hot})State_{ON} + (Cold)State_{OFF}$

$State_{OFF} = (Hot)State_{ON} + (\overline{Cold})State_{OFF}$



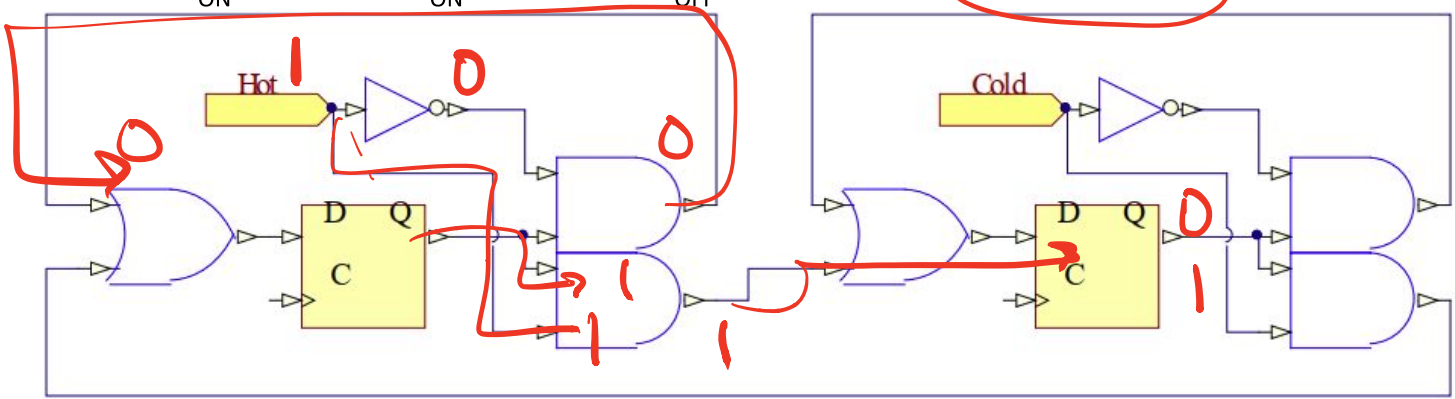
One Hot Construction

- Start with Flow Diagram
- Replace each state with a D Flip Flop
- D input is an OR gate
  - Each input is a transition
    - Source State AND transition condition



$State_{ON} = (\overline{Hot})State_{ON} + (Cold)State_{OFF}$

$State_{OFF} = (Hot)State_{ON} + (\overline{Cold})State_{OFF}$



Encoding Comparison

Binary

- Fewer Flip Flops
- Slower
- Wider LUTs

One Hot

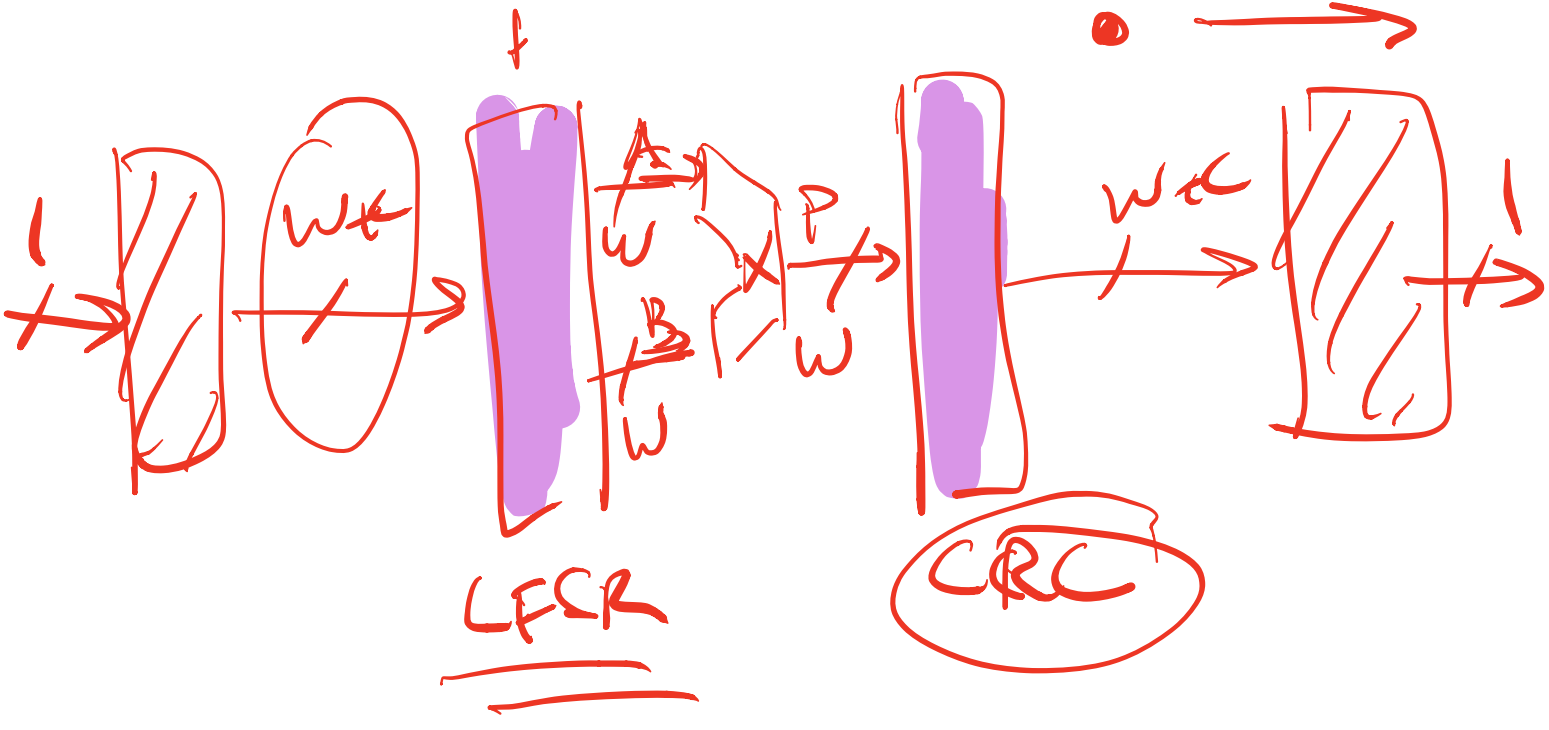
- More Flip Flops
- Faster Clock Speeds
- Simpler Logic
- Many Illegal States
  - What if 2 bits are hot?
  - LOTS of “Don’t Cares” in logic

Wait States

- Typical Data Flow:
  - FSM creates control signals for a clocked element
  - Other element takes N cycles to finish
  - FSM proceeds to next task
- Wait States delay the FSM
  - LUT □ Single Transition
  - One Hot □ Just a D Flip Flop, no logic!

$w \simeq 1\omega t$

Real World Example



Try at Home: Create FSMs for

- **MOD4**: Outputs *True* if the number of 1s is a multiple of 4
  - Every clock cycle, increment if input = 1
- **Pulse counter**: Output *True* if the number of pulses is a multiple of 3
  - Only increment if there was a 0 between 1s
- **Soda Machine**
  - Inputs: Nickel Detected, Dime Detected
  - Outputs: Every 25 cents, it gives you a drink
- **Sequential Adder**
  - Inputs: 2 streams of bits, LSB first
  - Output: 1 stream of bits, LSB first equal to the sum of the inputs
- **Gray code counter**
  - Inputs: increment, reset

For each, create a Flow Diagram with:

- labeled states, with any circuit outputs noted for each state
- transitions between states clearly annotated with their cause

Choose one of your diagrams and create a circuit schematic, using either a one-hot or binary design

Hints for Try at Home

- Output True if the number of 1s is a multiple of 4
  - Use 4 states connected in a ring. Advance on 1.
  - State0 outputs 1, all others output 0
- Output True if the number of pulses is a multiple of 3
  - Use 6 states (3 pairs). Each transition represents a positive or negative edge on the input signal.
- A Soda Machine
  - Use 5 states (5 cents each). Advance by 0, 1 or 2 states based on inputs (3 states depending on how you interpret reqs)
- Sequential Adder
  - Use your states to hold carry information.