ECE/CS 250
Computer Architecture

Fall 2021

Basics of Logic Design:
Finite State Machines

Tyler Bletsch

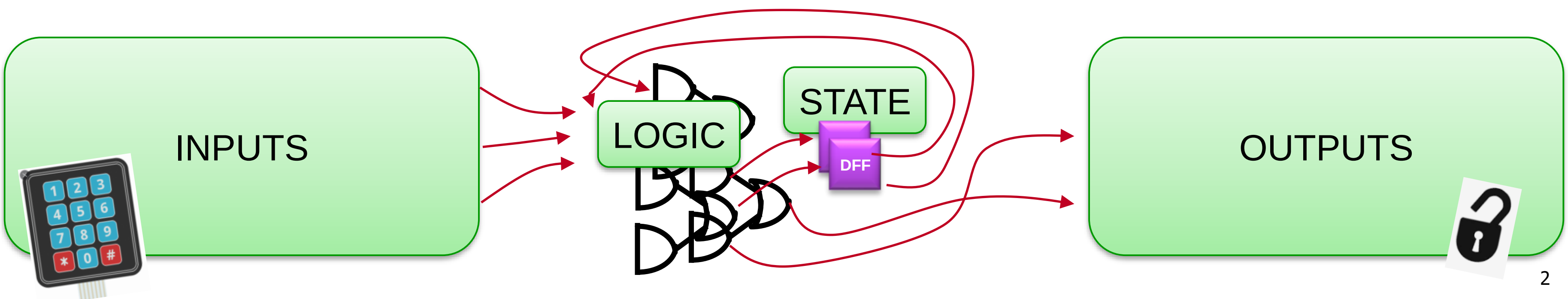
Duke University

Slides are derived from work by
Daniel J. Sorin (Duke), Drew Hilton (Duke), Alvy Lebeck (Duke), Amir Roth (Penn)

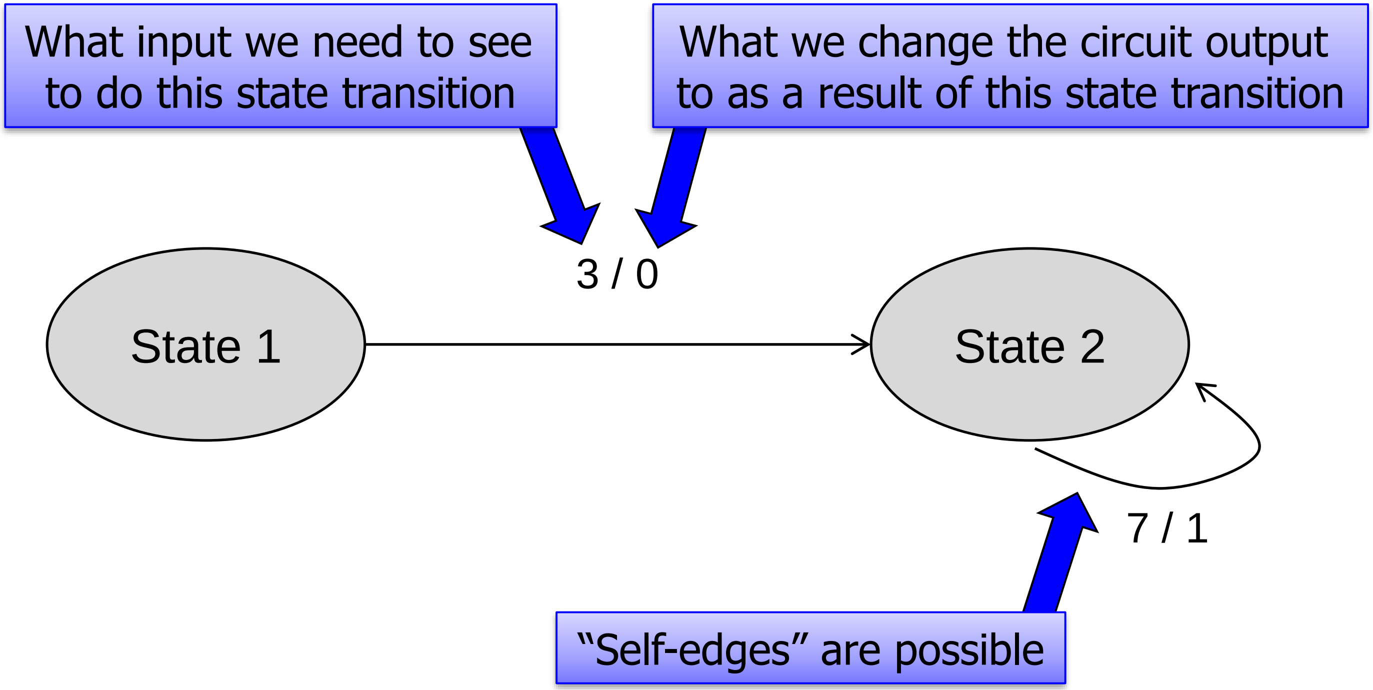
Finite State Machine (FSM)

- FSM = States + Transitions
  - Next state = function (current state, inputs)
  - Outputs = function (current state, inputs)
- What you do depends on what state you're in
  - Think of a calculator … if you type "+3=", the result depends on what you did before, i.e., the state of the calculator
- Canonical Example: Combination Lock
  - Must enter 3 8 4 to unlock

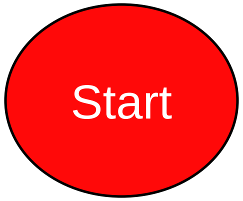
Preview: the major ingredients to a finite state machine circuit



How FSMs are represented

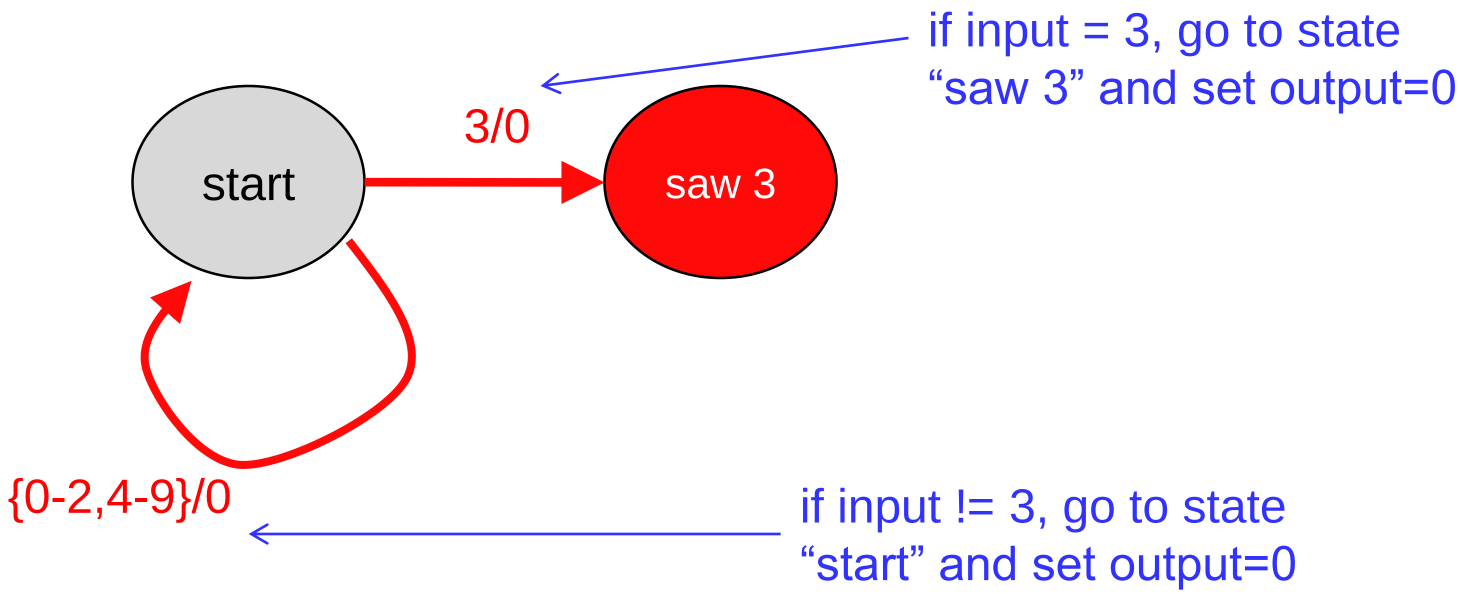


Finite State Machines: Example



- Combination Lock Example:
  - Need to enter 3 8 4 to unlock
- Initial State called “start”: no valid piece of combo seen
  - All FSMs get reset to their start state

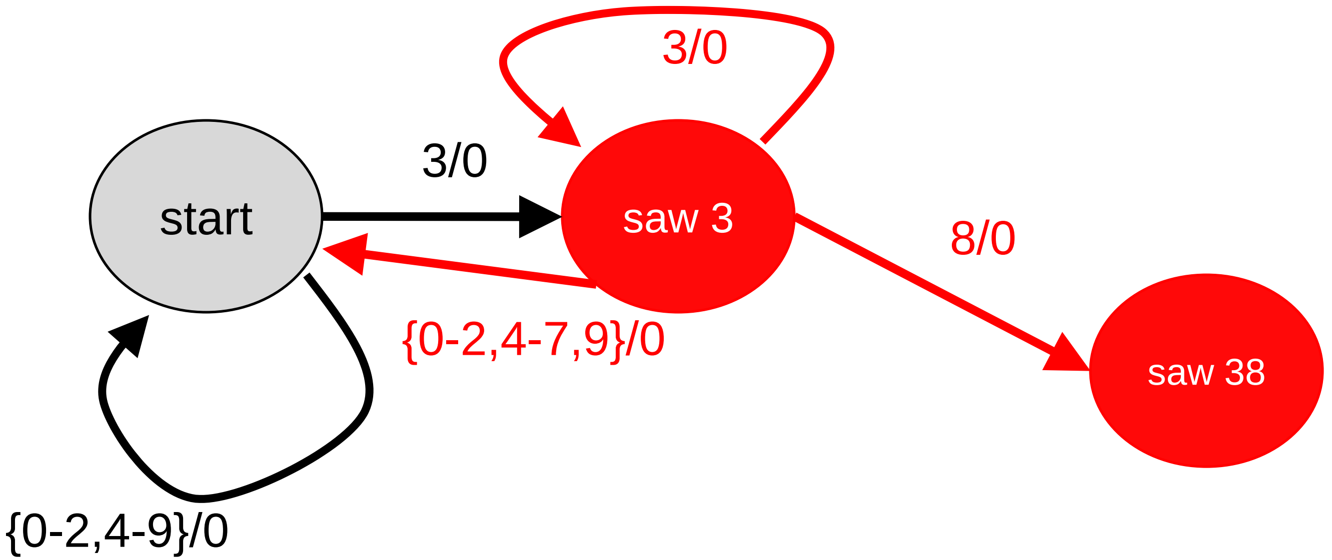
Finite State Machines: Example



- Combination Lock Example:
  - Need to enter **3** 8 4 to unlock

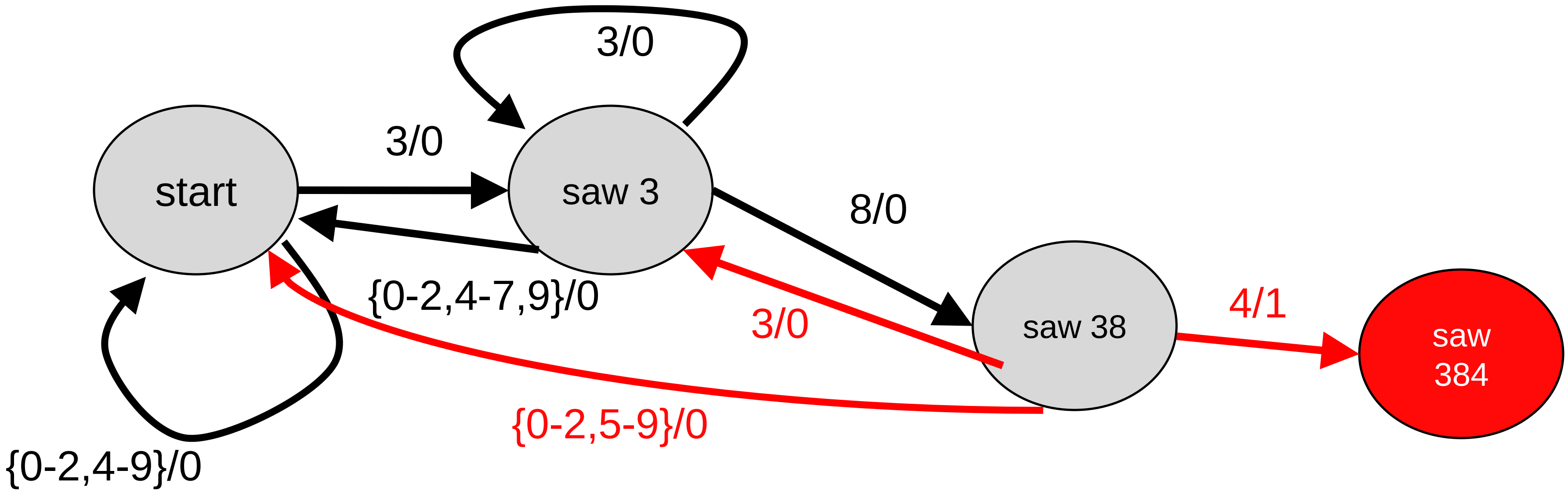
- Input of 3: transition to new state, output=0
- Any other input: stay in same state, output=0

Finite State Machines: Example



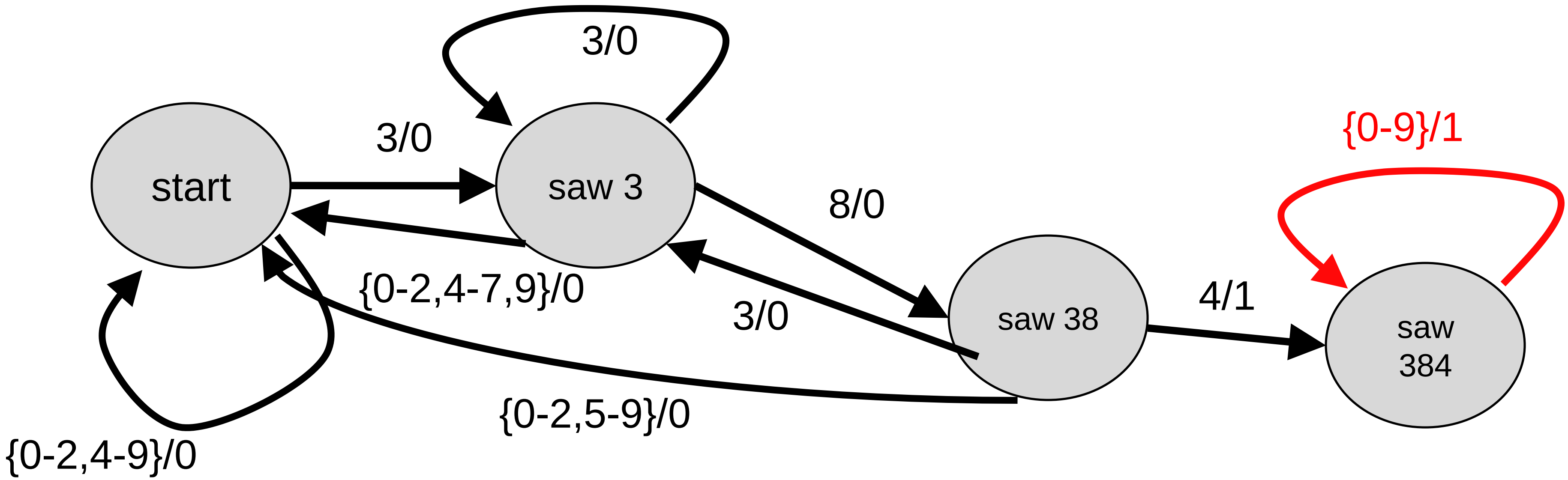
- Combination Lock Example:
  - Need to enter **3 8** 4 to unlock
- If in state "saw 3":
  - Input = 8? Goto state "saw 38" and output=0

Finite State Machines: Example



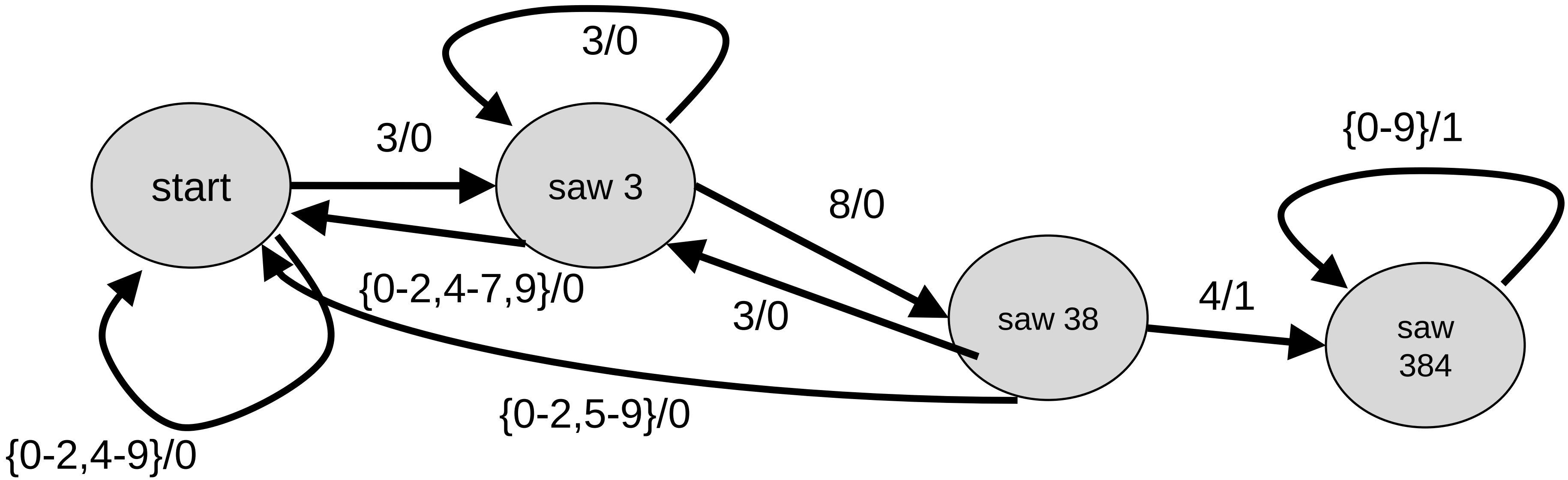
- Combination Lock Example:
  - Need to enter **3 8 4** to unlock
- If in state “saw 38”:
  - Input = 4? Goto state “saw 384” and set output=1 → Unlock!

Finite State Machines: Example



- Combination Lock Example:
  - Need to enter **3 8 4** to unlock
- If in state "saw 384":
  - Stay in this state forever and output=1

Finite State Machines: Example



In this picture, the circles are states.
The arcs between the states are transitions.

The figure is a state transition diagram, and it's the first thing you make when designing a finite state machine (FSM).

Finite State Machines: Caveats

<u>Do NOT assume all FSMs are like this one!</u>

- A finite state machine (FSM) has at least two states, but can have many, many more. There's nothing sacred about 4 states (as in this example). Design your FSMs to have the appropriate number of states for the problem they're solving.
  - Question: how many states would we need to detect sequence 384384?

- Most FSMs don't have state from which they can't escape.

FSM Types: Moore and Mealy

- Recall: FSM = States + Transitions
  - Next state = function (current state, inputs)
  - Outputs = function (current state, inputs)
  - *Write the output on the edges*
  - This is the most general case
    - Called a "Mealy Machine"
    - We will assume Mealy Machines in this lecture

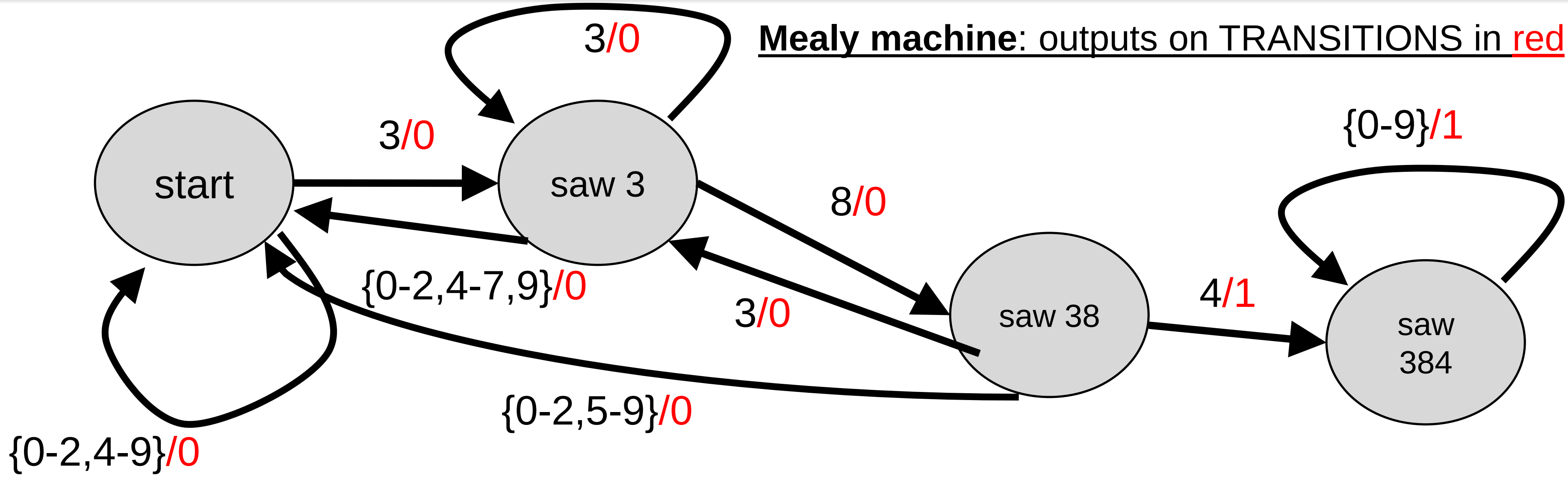
"Mealy Machine" developed in 1955 by George H. Mealy

- A more restrictive FSM type is a "Moore Machine"
  - Outputs = function (current state)
  - *Write the output in the states*
  - More often seen in software implementations

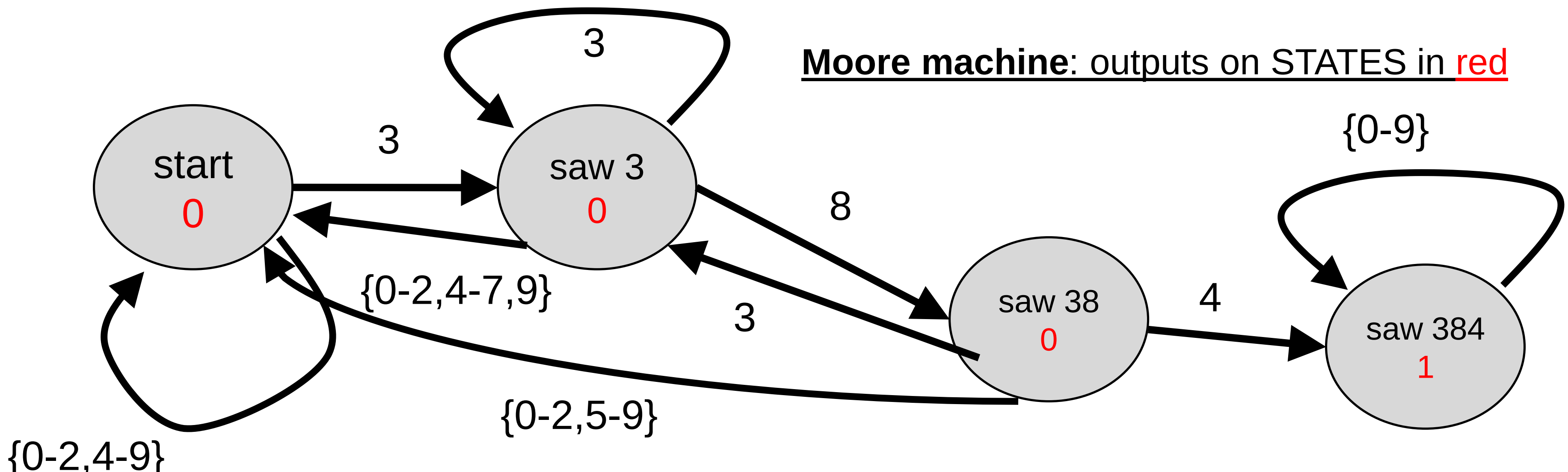
"Moore Machine" developed in 1956 by Edward F. Moore

Mealy vs Moore

Mealy machine: outputs on TRANSITIONS in red



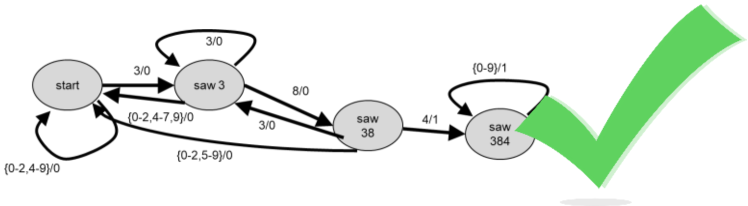
Moore machine: outputs on STATES in red



FSM Design Process

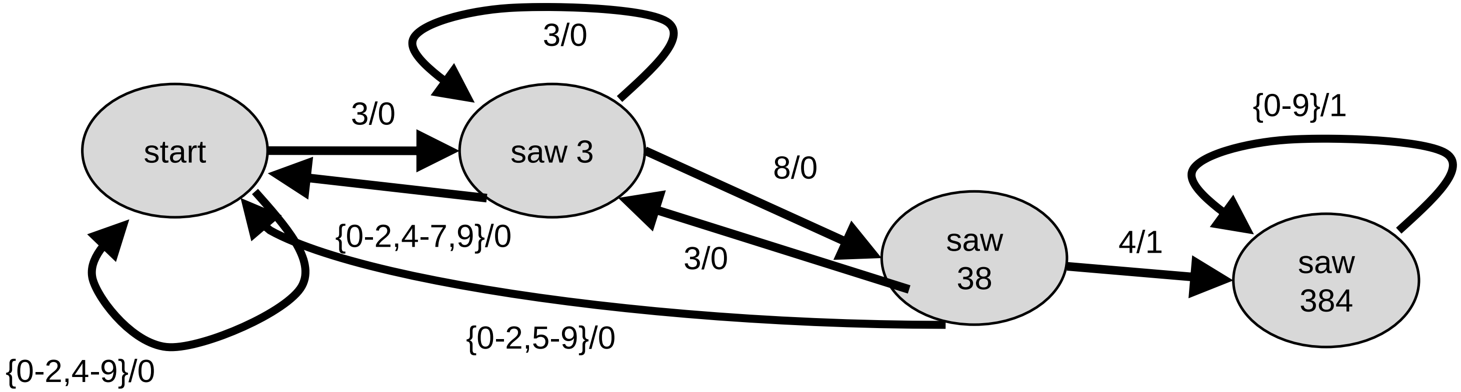
- Systematic approach that always works:

1. Start with state transition diagram



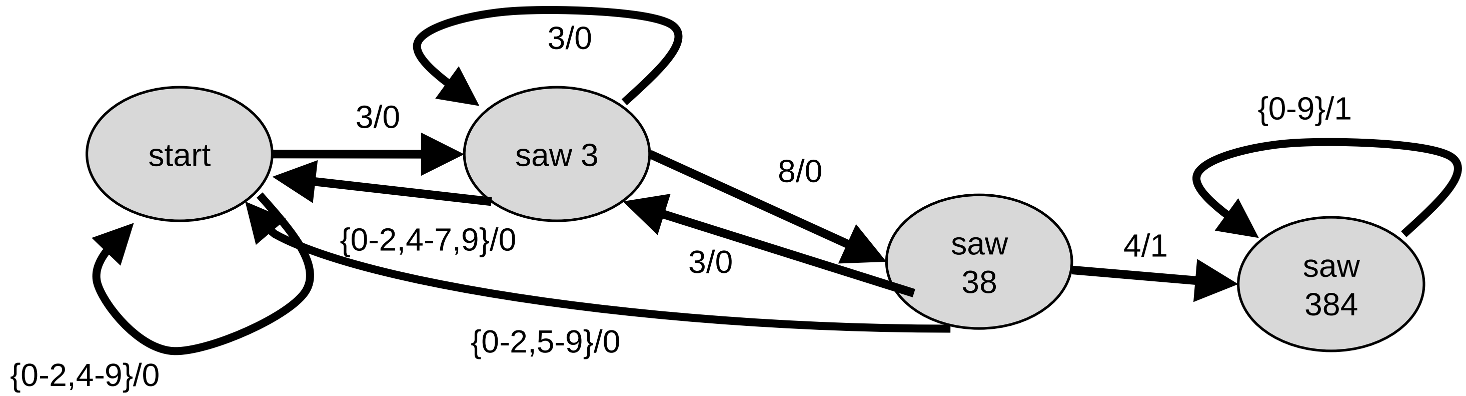
2. Make truth table
3. Write out sum-of-products logic equations
4. Optimize logic equations (optional)
5. Implement logic in circuit

State Transition Diagram → Truth Table



Current State	Input	Next state	Output
Start	3	Saw 3	0 (closed)
Start	Not 3	Start	0
Saw 3	8	Saw 38	0
Saw 3	3	Saw 3	0
Saw 3	Not 8 or 3	Start	0
Saw 38	4	Saw 384	1 (open)
Saw 38	3	Saw 3	0
Saw 38	Not 4 or 3	Start	0
Saw 384	Any	Saw 384	1

State Transition Diagram → Truth Table



Digital logic → must represent everything in binary, including state names.
But mapping is arbitrary!

We'll use this mapping:
start = 00
saw 3 = 01
saw 38 = 10
saw 384 = 11

State Transition Diagram → Truth Table

Current State	Input	Next state	Output
00 (start)	3	01	0 (closed)
00	Not 3	00	0
01	8	10	0
01	3	01	0
01	Not 8 or 3	00	0
10	4	11	1 (open)
10	3	01	0
10	Not 4 or 3	00	0
11	Any	11	1

4 states → 2 flip-flops to hold the current state of the FSM
inputs to flip-flops are D_1D_0
outputs of flip-flops are Q_1Q_0

State Transition Diagram → Truth Table

Current State = **Q**'s

Next State = **D**'s

Q1	Q0	Input	D1	D0	Output
0	0	3	0	1	0 (closed)
0	0	Not 3	0	0	0
0	1	8	1	0	0
0	1	3	0	1	0
0	1	Not 8 or 3	0	0	0
1	0	4	1	1	1 (open)
1	0	3	0	1	0
1	0	Not 4 or 3	0	0	0
1	1	Any	1	1	1

Input can be 0-9 → requires 4 bits
input bits are in3, in2, in1, in0

State Transition Diagram → Truth Table

Q1	Q0	Input	D1	D0	Output
0	0	3	0	1	0 (closed)
0	0	Not 3	0	0	0
0	1	8	1	0	0
0	1	3	0	1	0
0	1	Not 8 or 3	0	0	0
1	0	4	1	1	1 (open)
1	0	3	0	1	0
1	0	Not 4 or 3	0	0	0
1	1	Any	1	1	1

Now let’s transform the **inputs**.

State Transition Diagram → Truth Table

Same inputs as before, now just in binary

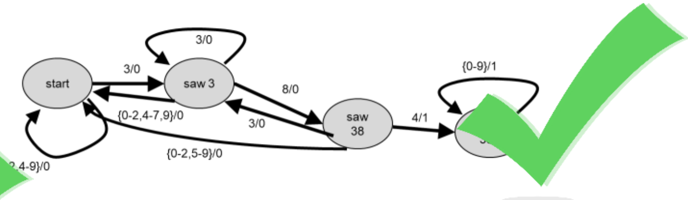
Q1	Q0	In3	In2	In1	In0	D1	D0	Output
0	0	0	0	1	1	0	1	0
0	0	Not 3 (all binary combos other than 0011)				0	0	0
0	1	1	0	0	0	1	0	0
0	1	0	0	1	1	0	1	0
0	1	Not 8 or 3 (all binary combos other than 1000 & 0011)				0	0	0
1	0	0	1	0	0	1	1	1
1	0	0	0	1	1	0	1	0
1	0	Not 4 or 3 (all binary combos other than 0100 & 0011)				0	0	0
1	1	Any				1	1	1

From here, it’s just like combinational logic design!
Write out product-of-sums equations, optimize, and build.

FSM Design Process

- Systematic approach that always works:

1. Start with state transition diagram



2. Make truth table

Q1	Q0	In3	In2	In1	In0	D1	D0	Output
0	0	0	0	1	1	0	1	0
0	0	Not 3 (all binary combos other than 0011)				0	0	0
0	1	1	0	0	0	1	0	0
0	1	0	0	1	1	0	1	0
0	1	Not 8 or 3 (all binary combos other than 1000 & 0011)				0	0	0
1	0	0	1	0	0	1	1	[illegible]
1	0	0	0	1	1	0	1	0
1	0	Not 4 or 3 (all binary combos other than 0100 & 0011)				0	0	0
1	1	Any				1	1	1

3. **Write out sum-of-products logic equations**

4. Optimize logic equations (optional)
   (we'll skip for this example)

5. Implement logic in circuit

State Transition Diagram → Truth Table

Q1	Q0	In3	In2	In1	In0	D1	D0	Output
0	0	0	0	1	1	0	1	0
0	0	Not 3 (all binary combos other than 0011)				0	0	0
0	1	1	0	0	0	1	0	0
0	1	0	0	1	1	0	1	0
0	1	Not 8 or 3 (all binary combos other than 1000 & 0011)				0	0	0
1	0	0	1	0	0	1	1	1
1	0	0	0	1	1	0	1	0
1	0	Not 4 or 3 (all binary combos other than 0100 & 0011)				0	0	0
1	1	Any				1	1	1

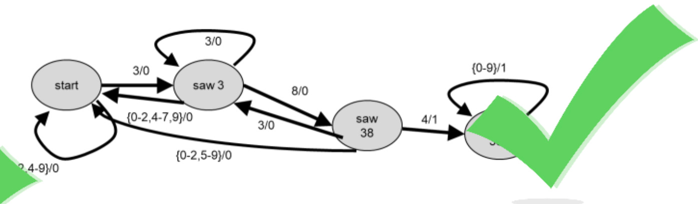
Sum of products!

Output = (Q1 & !Q0 & !In3 & In2 & !In1 & !In0) | (Q1 & Q0)
D1 = (!Q1 & Q0 & In3 & !In2 & !In1 & !In0) | (Q1 & !Q0 & !In3 & In2 & !In1 & !In0) | (Q1 & Q0)
D0 = do the same thing

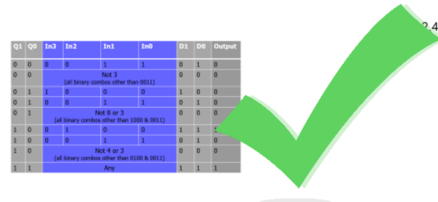
FSM Design Process

- Systematic approach that always works:

1. Start with state transition diagram

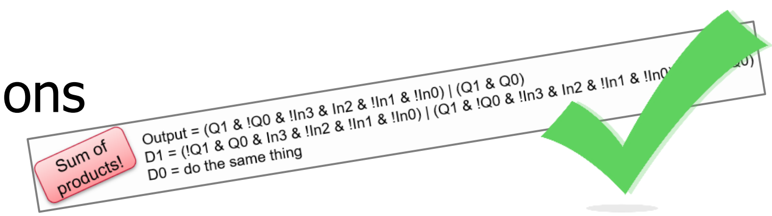


2. Make truth table



Q1	Q0	In3	In2	In1	In0	D1	D0	Output
0	0	0	0	1	1	0	1	0
0	0	Not 3 (all binary combos other than 0011)				0	0	0
0	1	1	0	0	0	1	0	0
0	1	0	0	1	1	0	1	0
0	1	Not 8 or 3 (all binary combos other than 1000 & 0011)				0	0	0
1	0	0	1	0	0	1	1	1
1	0	0	0	1	1	0	1	0
1	0	Not 4 or 3 (all binary combos other than 0100 & 0011)				0	0	0
1	1	Any				1	1	1

3. Write out sum-of-products logic equations



4. Optimize logic equations (optional)
   (we'll skip for this example)

5. **Implement logic in circuit**

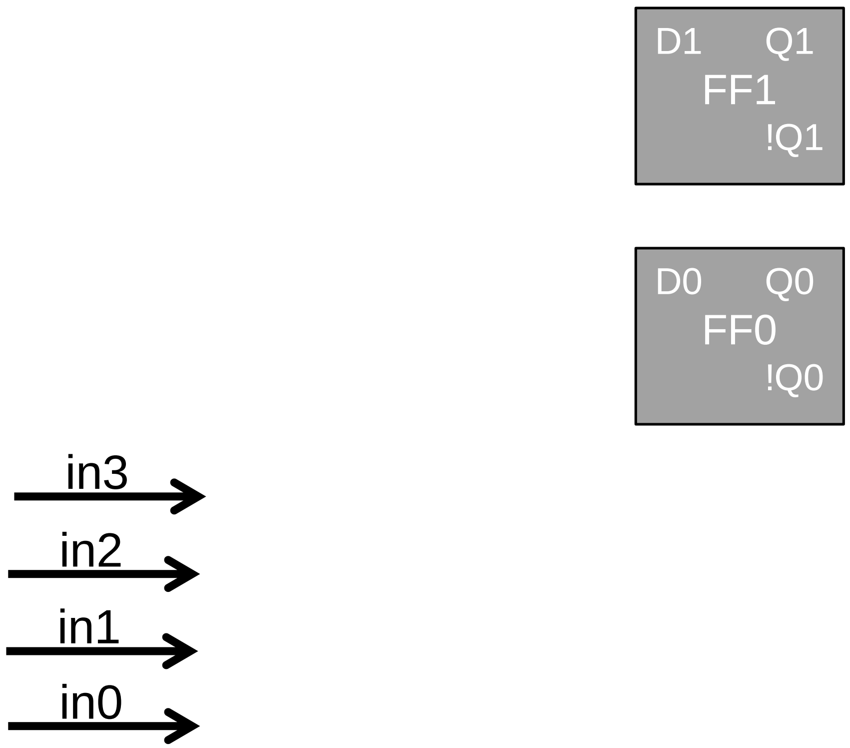
State Transition Diagram → Truth Table

Q1	Q0	In3	In2	In1	In0	D1	D0	Output
0	0	0	0	1	1	0	1	0
0	0	Not 3				0	0	0
0	1	1	0	0	0	1	0	0
0	1	0	0	1	1	0	1	0
0	1	Not 8 or 3				0	0	0
1	0	0	1	0	0	1	1	1
1	0	0	0	1	1	0	1	0
1	0	Not 4 or 3				0	0	0
1	1	Any				1	1	1

Remember, these represent **DFF outputs** (Q1, Q0) …and these are the **DFF inputs** (D1, D0)

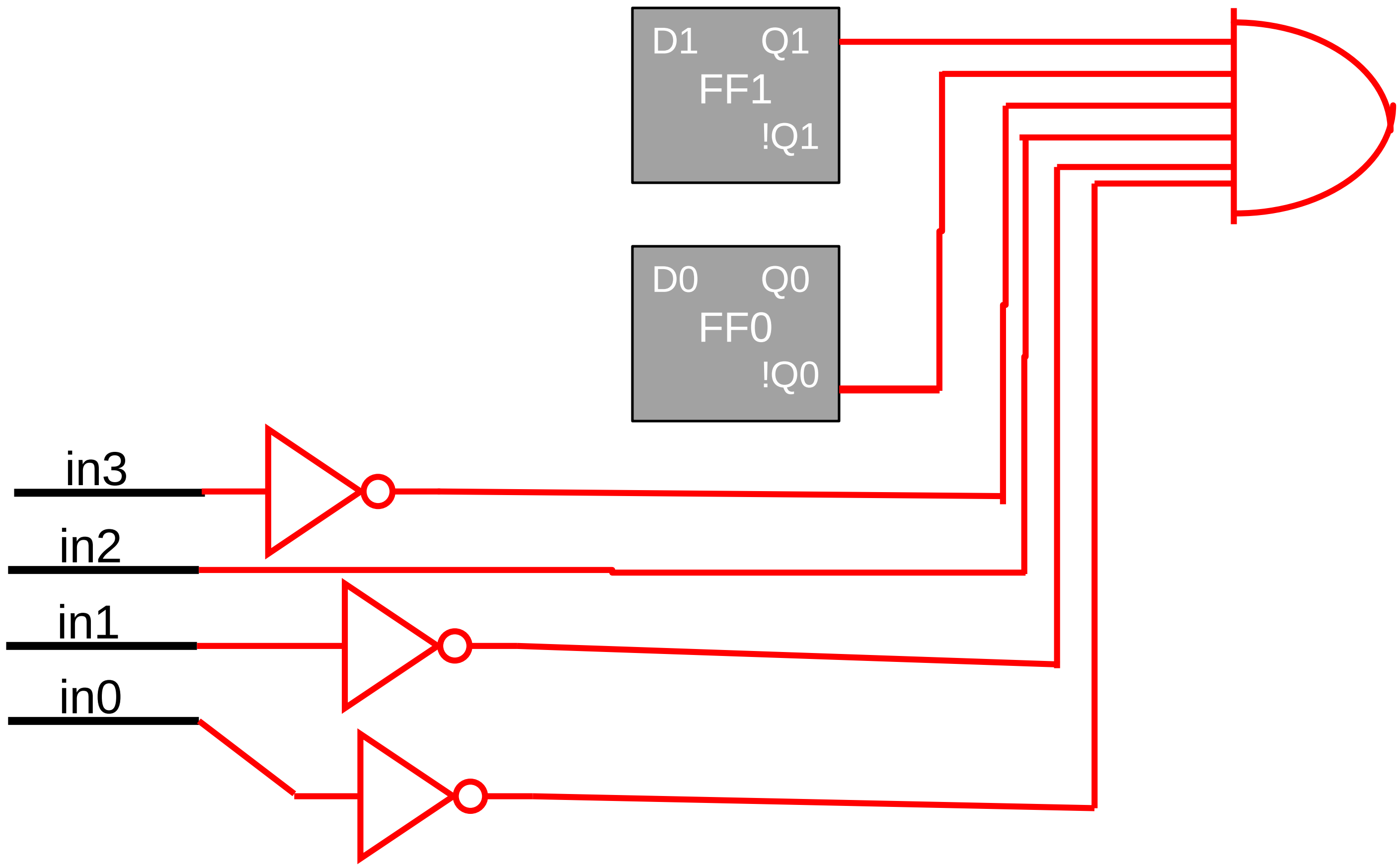
The DFFs are how we store the **state**.

Truth Table → Sequential Circuit



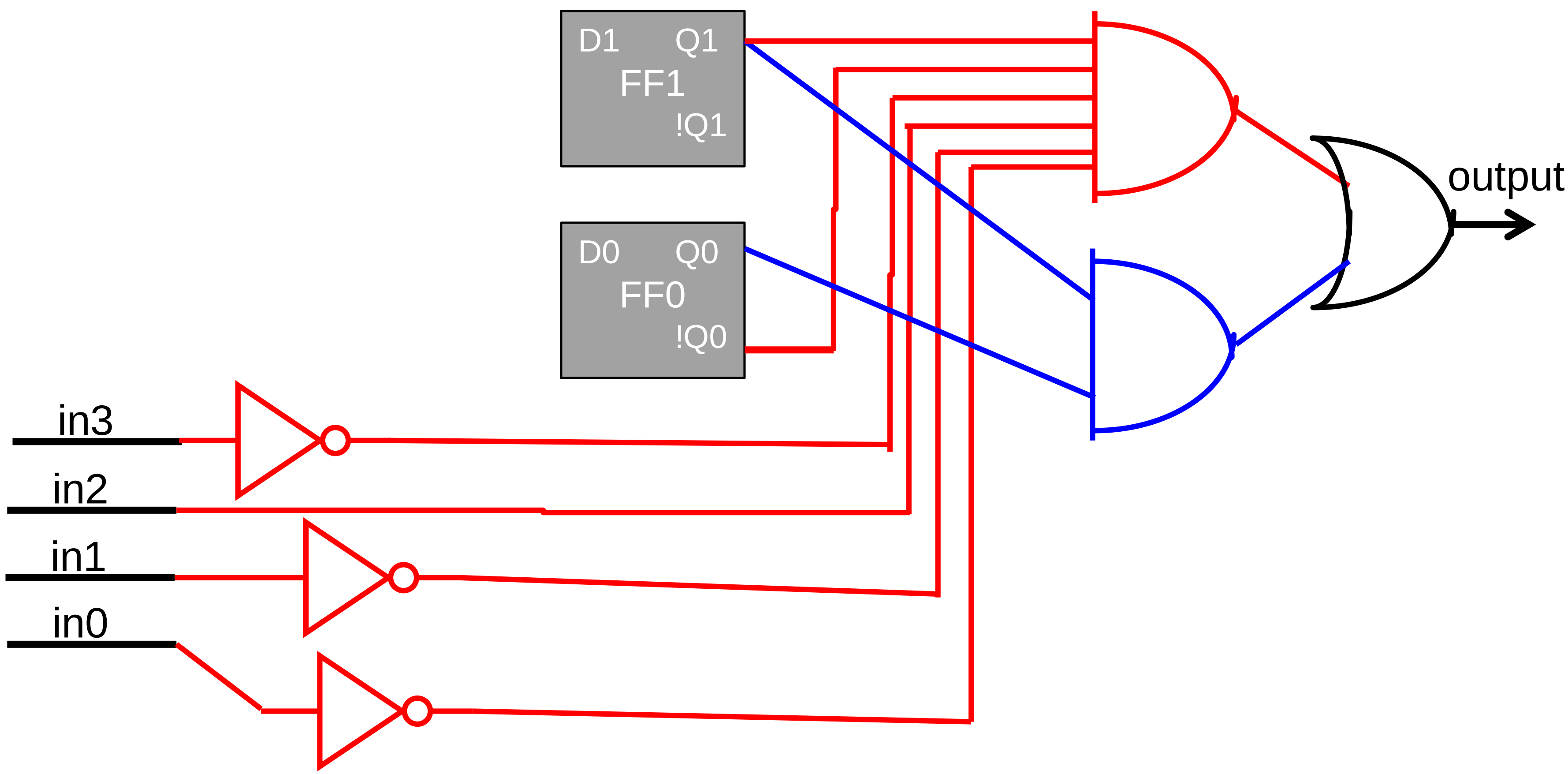
Start with 2 FFs and 4 input bits. FFs hold current state of FSM.
(not showing clock/enable inputs on flip flops)

Truth Table → Sequential Circuit



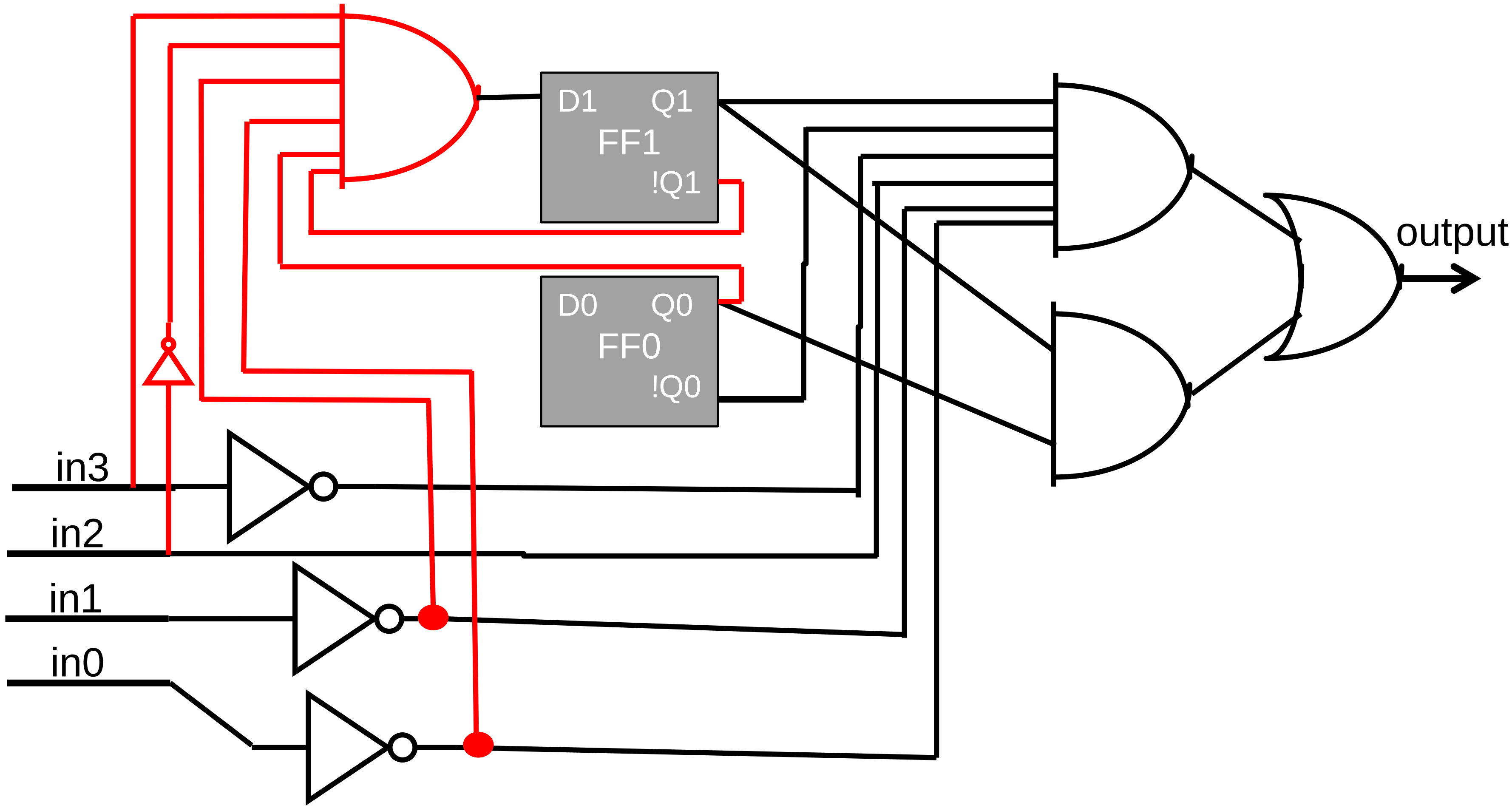
output = (Q1 & !Q0 & !In3 & In2 & !In1 & !In0) | (Q1 & Q0)

Truth Table → Sequential Circuit



output = (Q1 & !Q0 & !In3 & In2 & !In1 & !In0) | (Q1 & Q0)

Truth Table → Sequential Circuit



D1 = (!Q1 & Q0 & In3 & !In2 & !In1 & !In0) | (Q1 & !Q0 & !In3 & In2 & !In1 & !In0) | (Q1 & Q0)

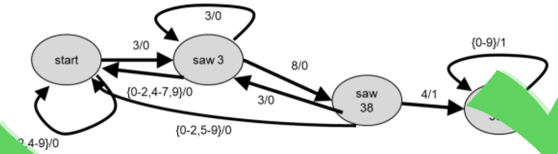
Not pictured

Follow a similar procedure for D0…

FSM Design Process

- Systematic approach that always works:

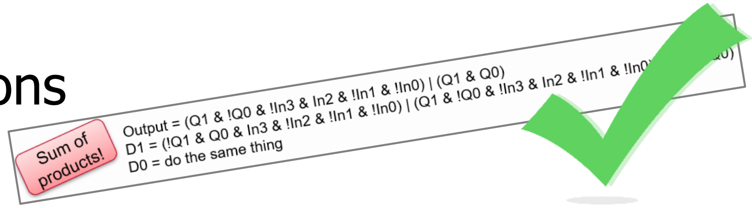
1. Start with state transition diagram



2. Make truth table

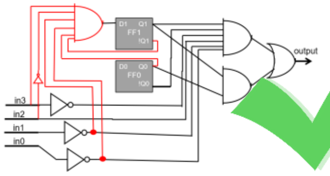
Q1	Q0	In3	In2	In1	In0	D1	D0	Output
0	0	0	0	1	1	0	1	0
0	0	Not 3 (all binary combos other than 0011)				0	0	0
0	1	1	0	0	0	1	0	0
0	1	0	0	1	1	0	1	0
0	1	Not 8 or 3 (all binary combos other than 1000 & 0011)				0	0	0
1	0	0	1	0	0	1	1	[illegible]
1	0	0	0	1	1	0	1	0
1	0	Not 4 or 3 (all binary combos other than 0100 & 0011)				0	0	0
1	1	Any				1	1	1

3. Write out sum-of-products logic equations



4. Optimize logic equations (optional)
   (we'll skip for this example)

5. Implement logic in circuit



We did it!
Stock photo guy is happy!

FSM tips

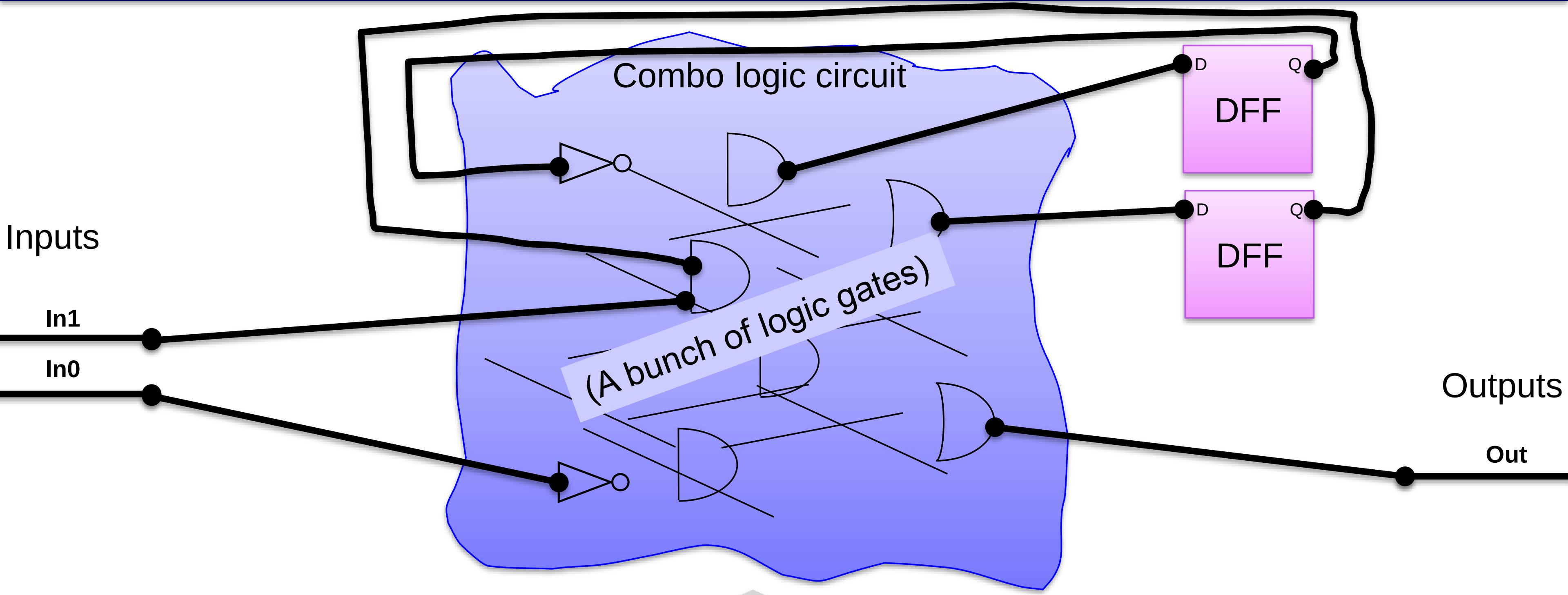
- Sometimes can do something non-systematic
  - Requires cleverness, but tough to do in general

- Do not do any of the following!
  - Use clock as an input (D input of FF)
  - Have multiple clocks
  - Perform logic on clock signal

    (except maybe a NOT gate to go from rising to falling edge triggered)

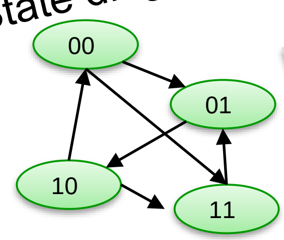
- Let's review the FSM Design Process one more time, this time with animation…

FSM Design Process, animated



Yields

State diagram



Yields

Truth table

Current state		Input		Next state		Output
Q1	Q0	In1	In0	D1	D0	Out
0	0	0	0	0	1	1
0	0	0	1	1	0	1
0	0	1	0	1	1	0
0	0	1	1	0	1	0

<u>Steps:</u>

1. State Transition Diagram
2. Do truth table
3. Sum-of-products
4. Optimize?

5a. Make combo logic
5b. Slap down DFFs
5c. Hook up DFFs
5d. Hook up inputs/outputs

QUESTIONS?