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
How FSMs are represented

What input we need to see to do this state transition

State 1

What we change the circuit output to as a result of this state transition

State 2

3 / 0

7 / 1

“Self-edges” are possible
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
• 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!
• Combination Lock Example:
  • Need to enter 3 8 4 to unlock

• If in state “saw 384”:
  • Stay in this state forever and output=1
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).
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

• 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
Mealy vs Moore

**Mealy machine**: outputs on TRANSITIONS in red

**Moore machine**: outputs on STATES in red
State Transition Diagram → Truth Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Input</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>3</td>
<td>Saw 3</td>
<td>0 (closed)</td>
</tr>
<tr>
<td>Start</td>
<td>Not 3</td>
<td>Start</td>
<td>0</td>
</tr>
<tr>
<td>Saw 3</td>
<td>8</td>
<td>Saw 38</td>
<td>0</td>
</tr>
<tr>
<td>Saw 3</td>
<td>3</td>
<td>Saw 3</td>
<td>0</td>
</tr>
<tr>
<td>Saw 3</td>
<td>Not 8 or 3</td>
<td>Start</td>
<td>0</td>
</tr>
<tr>
<td>Saw 38</td>
<td>4</td>
<td>Saw 384</td>
<td>1 (open)</td>
</tr>
<tr>
<td>Saw 38</td>
<td>3</td>
<td>Saw 3</td>
<td>0</td>
</tr>
<tr>
<td>Saw 38</td>
<td>Not 4 or 3</td>
<td>Start</td>
<td>0</td>
</tr>
<tr>
<td>Saw 384</td>
<td>Any</td>
<td>Saw 384</td>
<td>1</td>
</tr>
</tbody>
</table>
State Transition Diagram \(\rightarrow\) Truth Table

Digital logic \(\rightarrow\) 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

<table>
<thead>
<tr>
<th>Current State</th>
<th>Input</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (start)</td>
<td>3</td>
<td>01</td>
<td>0 (closed)</td>
</tr>
<tr>
<td>00</td>
<td>Not 3</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>8</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>3</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>Not 8 or 3</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>11</td>
<td>1 (open)</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>Not 4 or 3</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>Any</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q0</th>
<th>Input</th>
<th>D1</th>
<th>D0</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0 (closed)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Not 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Not 8 or 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1 (open)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Not 4 or 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Any</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Input can be 0-9 → requires 4 bits
input bits are in3, in2, in1, in0
From here, it’s just like combinational logic design! Write out product-of-sums equations, optimize, and build.
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
State Transition Diagram \( \rightarrow \) Truth Table

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q0</th>
<th>In3</th>
<th>In2</th>
<th>In1</th>
<th>In0</th>
<th>D1</th>
<th>D0</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Remember, these represent **DFF outputs** ...and these are the **DFF inputs**

The DFFs are how we store the **state**.
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)
output = \((Q_1 \& \neg Q_0 \& \neg In_3 \& In_2 \& \neg In_1 \& \neg In_0) \mid (Q_1 \& Q_0)\)
Truth Table $\rightarrow$ Sequential Circuit

D1 = ($\neg Q1$ & Q0 & In3 & $\neg$In2 & $\neg$In1 & $\neg$In0) | (Q1 & $\neg$Q0 & $\neg$In3 & In2 & $\neg$In1 & $\neg$In0) | (Q1 & Q0)

Follow a similar procedure for D0…
FSM Design Principles

• Systematic approach that always works:
  • Start with state transition diagram
  • Make truth table
  • Write out product-of-sums logic equations
  • Optimize logic equations (optional)
  • Implement logic in circuit

• 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)
  • Perform logic on clock signal
    (except maybe a NOT gate to go from rising to falling edge triggered)