Finite State Machines

Hakim Weatherspoon

CS 3410

Computer Science

Cornell University

The slides are the product of many rounds of teaching CS 3410 by Professors Weatherspoon, Bala, Bracy, and Sirer.
Stateful Components

Combinational logic

- Output computed directly from inputs
- System has no internal state
- Nothing depends on the past!

Need:

- To record data
- To build stateful circuits
- A state-holding device

Sequential Logic & Finite State Machines
Finite State Machines

An electronic machine which has

- external inputs
- externally visible outputs
- internal state

Output and next state depend on

- inputs
- current state
Abstract Model of FSM

Machine is

\[ M = (S, I, O, \delta) \]

- **S**: Finite set of states
- **I**: Finite set of inputs
- **O**: Finite set of outputs
- **\( \delta \)**: State transition function

Next state depends on present input *and* present state
Automata Model

Finite State Machine

• inputs from external world
• outputs to external world
• internal state
• combinational logic
Input: **up** or **down**
Output: **on** or **off**
States: **A**, **B**, **C**, or **D**
Input: \(\text{up} \text{ or } \text{down}\)

Output: \(\text{on} \text{ or } \text{off}\)

States: \(A, B, C, \text{ or } D\)
Input: 0=up or 1=down
Output: 1=on or 0=off
States: 00=A, 01=B, 10=C, or 11=D
Outputs and next state depend on both current state and input
Special Case: Moore Machine

Outputs depend only on current state
Moore Machine FSM Example

Input: \textbf{up} or \textbf{down}
Output: \textbf{on} or \textbf{off}
States: \textbf{A}, \textbf{B}, \textbf{C}, or \textbf{D}
Mealy Machine FSM Example

Input: **up** or **down**
Output: **on** or **off**
States: A, B, C, or D
Activity#2: Create a Logic Circuit for a Serial Adder

Add two infinite input bit streams

- streams are sent with least-significant-bit (lsb) first

```
...10110
...01111
```

Sum: output

```
...0010[1]
```
Activity#2: Create a Logic Circuit for a Serial Adder

Add two infinite input bit streams
  • streams are sent with least-significant-bit (lsb) first

Carry-out

...10110

...01111

Sum: output

...001\[01]
Activity#2: Create a Logic Circuit for a Serial Adder

Add two infinite input bit streams

- streams are sent with least-significant-bit (lsb) first

Carry-in

\[ \begin{array}{l}
...10110 \\
...01111 \\
\end{array} \]

Sum: output

\[ \begin{array}{l}
...00101 \\
\end{array} \]
Activity #2: Create a Logic Circuit for a Serial Adder

Add two infinite input bit streams

- streams are sent with least-significant-bit (lsb) first

```
...10110
...01111
```

Carry-out

```
1
1
```

```
10110
01111
```

Sum: output

```
...00101
```

Add two infinite input bit streams

- streams are sent with least-significant-bit (lsb) first

How many states are needed to represent FSM

a) 0  
b) 1  
c) 2  
d) 3  
e) 4
Strategy for Building an FSM

(1) Draw a state diagram (e.g. Mealy Machine)
(2) Write output and next-state tables
(3) Encode states, inputs, and outputs as bits
(4) Determine logic equations for next state and outputs
(5) Draw the Circuit
FSM: State Diagram

2 states ___ and ___
Inputs: ___ and ___
Output: ___
Two states: S0 (no carry in), S1 (carry in)
Inputs: a and b
Output: z

- z is the sum of inputs a, b, and carry-in (one bit at a time)
- A carry-out *is* the next carry-in state.
Two states: S0 (no carry in), S1 (carry in)

Inputs: a and b

Output: z

- z is the sum of inputs a, b, and carry-in (one bit at a time)
- A carry-out is the next carry-in state.
- Arcs labeled with input bits a and b, and output z
Two states: S0 (no carry in), S1 (carry in)
Inputs: a and b
Output: z
• z is the sum of inputs a, b, and carry-in (one bit at a time)
• A carry-out is the next carry-in state.
• Arcs labeled with input bits a and b, and output z (Mealy Machine)
Is this a Moore or Mealy Machine?

a) Moore
b) Mealy
c) Cannot be determined
Is this a Moore or Mealy Machine?

a) Moore

b) Mealy

c) Cannot be determined
Serial Adder: State Table

(2) Write down all input and state combinations
Write down all input and state combinations

(2) Write down all input and state combinations

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Current state</th>
<th>z</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>S0</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>S0</td>
<td>1</td>
<td>S0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>S0</td>
<td>1</td>
<td>S0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>S0</td>
<td>0</td>
<td>S1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>S1</td>
<td>1</td>
<td>S0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>S1</td>
<td>0</td>
<td>S1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>S1</td>
<td>0</td>
<td>S1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>S1</td>
<td>1</td>
<td>S1</td>
</tr>
</tbody>
</table>
Encode states, inputs, and outputs as bits

Two states, so 1-bit is sufficient

- A single flip-flop will encode the state

### Serial Adder: State Assignment

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>s</th>
<th>z</th>
<th>s'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
(4) Determine logic equations for next state and outputs

Combinational Logic Equations

\[
\begin{align*}
z &= \overline{a}b\overline{s} + ab\overline{s} + \overline{a}bs + abs \\
s' &= ab\overline{s} + \overline{a}bs + ab\overline{s} + abs
\end{align*}
\]
(4) Determine logic equations for next state and outputs

Combinational Logic Equations

\[ z = \overline{a} \overline{b} \overline{s} + a \overline{b} s + \overline{a} b s + a b s \]

\[ s' = a b \overline{s} + \overline{a} b s + a \overline{b} s + a b s \]
Sequential Logic Circuits

Strategy:

1. Draw a state diagram (e.g. Mealy Machine)
2. Write output and next-state tables
3. Encode states, inputs, and outputs as bits
4. Determine logic equations for next state and outputs

\[
\begin{align*}
z &= \overline{a}b\bar{s} + ab\bar{s} + \overline{a}bs + abs \\
s' &= ab\bar{s} + \overline{a}bs + a\bar{b}s + abs
\end{align*}
\]
Which statement(s) is true

(A) In a Moore Machine output depends on both current state and input
(B) In a Mealy Machine output depends on both current state and input
(C) In a Mealy Machine output depends on next state and input
(D) All the above are true
(E) None are true
Which statement(s) is true

(A) In a Moore Machine output depends on both current state and input

(B) In a Mealy Machine output depends on both current state and input

(C) In a Mealy Machine output depends on next state and input

(D) All the above are true

(E) None are true
General Case: Mealy Machine

Outputs and next state depend on both current state and input.
Special Case: Moore Machine

Outputs depend only on current state
Example: Digital Door Lock

Digital Door Lock

Inputs:

• keycodes from keypad
• clock

Outputs:

• “unlock” signal
• display how many keys pressed so far
Assumptions:

- signals are synchronized to clock
- Password is B-A-B

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Ø (no key)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>‘A’ pressed</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>‘B’ pressed</td>
</tr>
</tbody>
</table>
Door Lock: Outputs

Assumptions:
- High pulse on U unlocks door

Strategy:
1. Draw a state diagram (e.g. Moore Machine)
2. Write output and next-state tables
3. Encode states, inputs, and outputs as bits
4. Determine logic equations for next state and outputs
Door Lock: Simplified State Diagram

(1) Draw a state diagram (e.g. Moore Machine)
Door Lock: Simplified State Diagram

(1) Draw a state diagram (e.g. Moore Machine)
Door Lock: Simplified State Diagram

(2) Write output and next-state tables
Door Lock: Simplified State Diagram

(2) Write output and next-state tables
Door Lock: Simplified State Diagram

(2) Write output and next-state tables
Door Lock: Simplified State Diagram

(2) Write output and next-state tables
### State Table Encoding

<table>
<thead>
<tr>
<th>$S_2$</th>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$D_3$</th>
<th>$D_2$</th>
<th>$D_1$</th>
<th>$D_0$</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</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>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>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>$S_2$</th>
<th>$S_1$</th>
<th>$S_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>G2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>G3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S_2$</th>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$K$</th>
<th>$A$</th>
<th>$B$</th>
<th>$S'_2$</th>
<th>$S'_1$</th>
<th>$S'_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S'_2$</th>
<th>$S'_1$</th>
<th>$S'_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Bits, and outputs as bits
(4) Determine logic equations for next state and outputs

\[
U = \overline{S_2}S_1S_0
\]

\[
D_0 = \overline{S_2}S_1S_0 + S_2S_1S_0 + S_2\overline{S_1}S_0
\]

\[
D_1 = \overline{S_2}S_1S_0 + S_2S_1S_0 + \overline{S_2}S_1S_0
\]
### Door Lock: Implementation

<table>
<thead>
<tr>
<th>$S_2$</th>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$K$</th>
<th>$A$</th>
<th>$B$</th>
<th>$S'_2$</th>
<th>$S'_1$</th>
<th>$S'_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

K = $S_2' = S_2 S_1 S_0 K A B + S_2 S_1 S_0 K A B + S_2 S_1 S_2 K A B + S_2 S_1 S_0 K + S_2 S_1 S_0 K A B$

$S_0' = ?$

$S_1' = ?$

$S_2' = S_2 S_1 S_0 K A B + S_2 S_1 S_0 K A B + S_2 S_1 S_2 K A B + S_2 S_1 S_0 K + S_2 S_1 S_0 K A B$
Door Lock: Implementation

Strategy:
1. Draw a state diagram (e.g. Moore Machine)
2. Write output and next-state tables
3. Encode states, inputs, and outputs as bits
4. Determine logic equations for next state and outputs
Door Lock: Implementation

Strategy:
(1) Draw a state diagram (e.g. Moore Machine)
(2) Write output and next-state tables
(3) Encode states, inputs, and outputs as bits
(4) Determine logic equations for next state and outputs
Summary

We can now build interesting devices with sensors

• Using combinational logic

We can also store data values

• Stateful circuit elements (D Flip Flops, Registers, ...)
• State Machines or Ad-Hoc Circuits