Finite State Machines

CS 64: Computer Organization and Design Logic
Lecture #16

Ziad Matni
Dept. of Computer Science, UCSB
Administrative

• Lab 8 due on Thursday 3/9
• Labs 9 & 10 due on Thursday 3/16 (last day of class)

• We have 2 more classes left...
  – Tu. 3/14: Impacts of CS; Ethics in CS
  – Th. 3/16: Review for Final Exam

• Midterm #2 grading still in progress...
  – Stay tuned
Latches vs. FFs

- Latches capture data on an entire 1 or 0 of the clock
- FFs capture data on the edge of the clock
  - This example shows the positive (0→1) edge used

Latch out

FF out

D Flip-Flop (D-FF)

FFs give out less “glitchy” outputs
If a combinational logic circuit is an implementation of a *Boolean function*, then a sequential logic circuit can be considered an implementation of a *finite state machine*. 
FSMs

- Sequential circuits (or Finite State Machines, FSMs) have outputs

- For example, vending machine controllers generate output signals to dispense product, provide change, illuminate displays, etc.
General Form of FSMs
On the next rising edge of the clock, the output of the D-FF \( Q \) (\( Q^* \)) will become the previous value of \( Q \) (\( Q_0 \)) \textbf{AND} the value of input \( A \)

\[ Q^* = Q_0.A \]
There are 2 types/models of FSMs:

• Moore machine
  – Output is function of present state only

• Mealy machine
  – Output is function of present state and present input
Moore Machine
Example of a Moore Machine
(with 1 state)

\[ Z = (Q^* + B) = (Q_0 \cdot A + B) \]

On the next rising edge of the clock, the output of the entire circuit (Z) will become
( the previous value of Q (Q_0) **AND** the value of input A) **NOR** B
Mealy Machine
Example of a Mealy Machine
(with 1 state)

\[ Z = (Q^* + A + B) = (Q_0 \text{ XOR } A) + (A + B) \]

On the next rising edge of the clock, the output of the entire circuit (Z) will become ...etc...
Diagraming State Machines

- A simple FSM example
- 2 states:
  - Door opened
  - Door closed
- This is called a state diagram
Example of a Moore Machine 1

**WASHER_DRYER**

- Let’s build a sequential logic FSM that acts as a controller to a washer/dryer machine

**SO:**

- Before we begin, the machine is in an initial state that is waiting for you to insert a coin. We’ll call that state “Initial State” (inventive, no?)

- The machine will start a washer timer as soon as a coin is inserted. The timer is controlled by a signal called TIMER_LT_30, which is always initialized to be 1.
Example of a Moore Machine 1

WASHER_DRYER

• Upon inserting a coin in the machine, we will begin the *wash cycle*. We’ll call that state “Wash”.
  – This state will output a signal to fill the washer with water (FILL_WATER). As long as the timer is below 30 mins (TIMER_LT_30 = 1), the cycle continues.
  – When the timer surpasses 30 mins, this state will be over.

3/8/17  Matni, CS64, Wi17
Example of a Moore Machine 1

WASHER_DRYER

- When the timer hits 30 mins, we will begin the rinse cycle. We’ll call that “Rinse”.

- This will output a signal to drain the water and refill with new water (DRAIN_REFILL). As long as the soap sensor is on (SOAP = 1), the cycle continues.
Example of a Moore Machine 1

WASHER_DRYER

- When the soap sensor goes off, we will begin the dry cycle. We’ll call that “Dry”.
- This state will output a signal to drain the water and begin drying (DRY_DRAIN). As long as the wet clothes sensor is on (WET = 1), the cycle continues.

- When the wet clothes sensor is off, we will stop.
State Diagram 1

- **Initial State**
  - COIN = 0
  - Timer LT 30 = 1

- **Wash**
  - COIN = 1
  - Timer LT 30 = 0
  - Fill Water = 1
  - Soap = 1

- **Rinse**
  - Soap = 0
  - Drain Refill = 1

- **Dry**
  - Wet = 0
  - Drain Dry = 1
  - Wet = 1
Example of a Moore Machine 2

DETECT_1101

- Let’s build a sequential logic FSM that always detects a specific serial sequence of bits: **1101**

**SO:**
- We’ll start at an “Initial” state (S0)
- We’ll first look for a 1. We’ll call that “State 1” (S1)
- We’ll then keep looking for another 1. We’ll call that “State 11” (S2)
- Then... a 0. We’ll call that “State 110” (S3)
- Then another 1. We’ll call that “State 1101” (S4) – this will output a FOUND signal

- We will always be detecting “1101” (it doesn’t end)
- If a CLEAR input is set at any time, we will go back to the “Initial” state.
State Diagram 2

- **Initial State**: S0
- **Input = 0**: Transition to S1
- **Input = 1**: Transition to S1
- **CLR = 1**: Transition to S4

- **S1** ("1")
- **S2** ("11")
- **S3** ("110")
- **S4** ("1101")

- **Input = 0**: Transition to S0
- **Input = 1**: Transition to S2

- **FOUND = 1**: Transition from S4 to S3

- **Input = 1**: Transition to S2 from S1 and S3

- **Input = 0**: Transition to S1 from S0, S1, and S3

- **Input = 1**: Transition to S1 from S3

3/9/17 Matni, CS64, Wi17
Encoding our States

*Per the last example:* We had 5 separate states:

<table>
<thead>
<tr>
<th>NAME</th>
<th>Binary Code</th>
<th>“One Hot” Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial State</td>
<td>S0 000</td>
<td>000001</td>
</tr>
<tr>
<td>“1”</td>
<td>S1 001</td>
<td>00010</td>
</tr>
<tr>
<td>“11”</td>
<td>S2 010</td>
<td>00100</td>
</tr>
<tr>
<td>“110”</td>
<td>S3 011</td>
<td>01000</td>
</tr>
<tr>
<td>“1101”</td>
<td>S4 100</td>
<td>10000</td>
</tr>
</tbody>
</table>
Using the “One Hot” Code to Determine the Circuit Design

• Every state has 1 D-FF
• We can see that:

\[ S1^* = S0.I \]
\[ S2^* = S1.I + S2.I + S4.I \]
\[ S3^* = S2.\overline{I} \]
\[ S4^* = S3.I \]
\[ S0^* = CLR + S0.\overline{I} + S1.\overline{I} + S3.\overline{I} + S4.\overline{I} \]