Chapter #8: Finite State Machine Design

8.1 - 8.2 Finite State Machine Design
Chapter Overview

Concept of the State Machine

• Partitioning into Datapath and Control
• When Inputs are Sampled and Outputs Asserted

Basic Design Approach

• Six Step Design Process

Alternative State Machine Representations

• State Diagram, ASM Notation, VHDL, ABEL Description Language

Moore and Mealy Machines

• Definitions, Implementation Examples

Word Problems

• Case Studies
Concept of the State Machine

Computer Hardware = Datapath + Control

- Registers
- Combinational Functional Units (e.g., ALU)
- Busses

FSM generating sequences of control signals
Instructs datapath what to do next

"Puppeteer who pulls the strings"

Datapath

Control

State

Qualifiers and Inputs

Qualifiers

© R.H. Katz  Transparency No. 14-3
Concept of the State Machine

Example: Odd Parity Checker

Assert output whenever input bit stream has odd # of 1's

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Even</td>
<td>0</td>
<td>Even</td>
<td>0</td>
</tr>
<tr>
<td>Even</td>
<td>1</td>
<td>Odd</td>
<td>0</td>
</tr>
<tr>
<td>Odd</td>
<td>0</td>
<td>Odd</td>
<td>1</td>
</tr>
<tr>
<td>Odd</td>
<td>1</td>
<td>Even</td>
<td>1</td>
</tr>
</tbody>
</table>

Symbolic State Transition Table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Encoded State Transition Table

==> One FF required because one bit is needed to represent the two states
Concept of the State Machine

Example: Odd Parity Checker

Using a D-FF

<table>
<thead>
<tr>
<th>Q</th>
<th>X</th>
<th>Q+</th>
<th>Output</th>
<th>D</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>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ D = Q'X + QX' = Q \oplus X \]

Since we have two states, we can implement the circuit with one flip-flop.

D FF Implementation
Concept of the State Machine

Example: Odd Parity Checker

Using a T-FF

Let $Q = PS$, $Q+ = NS$, $X = Input$

<table>
<thead>
<tr>
<th>$Q$</th>
<th>$X$</th>
<th>$Q+$</th>
<th>Output</th>
<th>$T$</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>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$Q+ = T \oplus Q$

$\implies T = Q+ \oplus Q$

$T = X$

Since we have two states, we can implement the circuit with one flip-flop.

T FF Implementation
Concept of the State Machine

Timing Behavior: Input 1 0 0 1 1 0 1 0 1 1 1 0
Concept of State Machine

**Timing:**
When are inputs sampled, next state computed, outputs asserted?

**State Time:** Time between clocking events

- Clocking event causes state/outputs to transition, based on inputs
- For set-up/hold time considerations:
  
  Inputs should be stable before clocking event

- After propagation delay, Next State entered, Outputs are stable

**NOTE:** Asynchronous signals take effect immediately
Synchronous signals take effect at the next clocking event

E.g., tri-state enable: effective immediately
sync. counter clear: effective at next clock event
Concept of State Machine

Example: Positive Edge Triggered Synchronous System

On rising edge, inputs sampled outputs, next state computed

After propagation delay, outputs and next state are stable

Immediate Outputs:
affect datapath immediately
could cause inputs from datapath to change

Delayed Outputs:
take effect on next clock edge
propagation delays must exceed hold times
Concept of the State Machine

Communicating State Machines

One machine's output is another machine's input

Machines advance in lock step

Initial inputs/outputs: X = 0, Y = 0
Basic Design Approach

Six Step Process

1. Understand the statement of the Specification

2. Obtain an abstract specification of the FSM (i.e. state diagram)

3. Perform a state minimization

4. Perform state assignment

5. Choose FF types to implement FSM state register

6. Implement the FSM

1, 2 covered now; 3, 4, 5 covered later; 4, 5 generalized from the counter design procedure
Basic Design Approach

Example: Vending Machine FSM

General Machine Concept:
deliver package of gum after 15 cents deposited
single coin slot for dimes, nickels
no change

Step 1. Understand the problem:
Draw a picture!

Block Diagram
Vending Machine Example

Step 2. Map into more suitable abstract representation

Tabulate typical input sequences:
- three nickels -- N,N,N
- nickel, dime -- N,D
- dime, nickel -- D,N
- two dimes -- D,D
- two nickels, dime -- N,N,D

Draw state diagram:
- Inputs: N, D, reset
- Output: open
Vending Machine Example

Step 3: State Minimization

Symbolic State Table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Inputs</th>
<th>Next State</th>
<th>Output Open</th>
</tr>
</thead>
<tbody>
<tr>
<td>0¢</td>
<td>0 0</td>
<td>0¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>5¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>10¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>5¢</td>
<td>0 0</td>
<td>5¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>10¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>15¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>10¢</td>
<td>0 0</td>
<td>10¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>15¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>15¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>15¢</td>
<td>X X</td>
<td>15¢</td>
<td>1</td>
</tr>
</tbody>
</table>

Moore Machine

Reuse states whenever possible
## Vending Machine Example

### Step 4: State Encoding

<table>
<thead>
<tr>
<th>Present State</th>
<th>Inputs</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q₃ Q₀</td>
<td>D N</td>
<td>Q₁⁺ = D₁</td>
<td>Q₀⁺ = D₀</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>X X</td>
<td>X</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>1 1</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>X X</td>
<td>X</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1 1</td>
<td>1 1</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td>X X</td>
<td>X</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>1 1</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td>X X</td>
<td>X</td>
</tr>
</tbody>
</table>

4 states ==> 2 bits needed to represent
Vending Machine Example

Step 5. Choose FFs for implementation

D FF easiest to use

\[
D1 = Q1 + D + Q0 \quad \text{N}
\]

\[
D0 = \overline{N} \quad Q0 + Q0 \quad \overline{N} + Q1 \quad \overline{N} + Q1 \quad D
\]

\[
OPEN = Q1 \quad Q0
\]
Vending Machine Example

\[ D_1 = Q_1 + D + Q_0 \bar{N} \]
\[ D_0 = \bar{N} Q_0 + Q_0 \bar{N} + Q_1 N + Q_1 D \]
\[ OPEN = Q_1 Q_0 \]

8 Gates, 2 flip flops
### Vending Machine Example

#### Step 5. Choosing FF for Implementation

**J-K FF**

<table>
<thead>
<tr>
<th>Present State</th>
<th>Inputs</th>
<th>Next State</th>
<th>Flip Flop Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q₀ Q₁</td>
<td>D N</td>
<td>Q+0</td>
<td>J₁ K₁</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>X 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>0 1</td>
<td>X 1</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>1 1</td>
<td>X X X</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>0 1</td>
<td>X X X</td>
</tr>
</tbody>
</table>

**Remapped encoded state transition table**

© R.H. Katz  Transparency No. 14-18
Vending Machine Example

Implementation:

K-map for J1

\[
\begin{array}{c|cc|cc}
\text{D} & 00 & 01 & 11 & 10 \\
\hline
\text{N} & 0 & 0 & X & X \\
00 & 0 & 1 & X & X \\
01 & 1 & 1 & X & X \\
10 & 1 & 1 & X & X \\
\end{array}
\]

K-map for K1

\[
\begin{array}{c|cc|cc}
\text{D} & 00 & 01 & 11 & 10 \\
\hline
\text{N} & 0 & 0 & X & 0 \\
00 & X & X & X & 0 \\
01 & X & X & X & 0 \\
10 & X & X & X & 0 \\
\end{array}
\]

K-map for J0

\[
\begin{array}{c|cc|cc}
\text{D} & 00 & 01 & 11 & 10 \\
\hline
\text{N} & 0 & 0 & X & 1 \\
00 & 1 & 1 & X & 1 \\
01 & 1 & 1 & X & 1 \\
10 & 1 & 1 & X & 1 \\
\end{array}
\]

K-map for K0

\[
\begin{array}{c|cc|cc}
\text{D} & 00 & 01 & 11 & 10 \\
\hline
\text{N} & 0 & 0 & 0 & X \\
00 & X & 0 & 0 & X \\
01 & X & 1 & 0 & X \\
10 & X & 1 & 0 & X \\
\end{array}
\]

\[
\begin{align*}
J1 &= D + Q0 \neg N \\
K1 &= 0 \\
J0 &= Q0 \neg N + Q1 D \\
K0 &= Q1 N \\
\end{align*}
\]

Output OPEN does not change:

\[
OPEN = Q1 Q0
\]
Vending Machine Example

7 Gates,
2 flip flops
HW #14 -- Section 8.1 & 8.2