Finite State Machine Design

Adapted from slides by R. H. Katz
Concept of Finite State Machine

• **Counters**: Sequential Circuits where State = Output
• **Generalizes to Finite State Machines:**
  – Outputs are Function of State (and Inputs)
  – Next States are Functions of State and Inputs
  – Used to implement circuits that control other circuits
  – "Decision Making" logic
Concept of Finite 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

Control

"Puppeteer who pulls the strings"

Datapath

"Puppet"

Control

State

Qualifiers and Inputs

Control Signal Outputs

Qualifiers
Concept of Finite State Machine

Example: Odd Parity Checker

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

State Diagram

Present State | Input | Next State | Output
---|---|---|---
Even | 0 | Even | 0
Even | 1 | Odd | 0
Odd | 0 | Odd | 1
Odd | 1 | Even | 1

Symbolic State Transition Table

Encoded State Transition Table
Concept of Finite State Machine

Example: Odd Parity Checker

Next State/Output Functions

\[ \text{NS} = \text{PS} \oplus \text{PI}; \quad \text{OUT} = \text{PS} \]

D FF Implementation

T FF Implementation

Timing Behavior: Input 1 0 0 1 1 0 1 0 1 1 1 0

Output 1 1 1 1 0 1 1 1 0 0 1 0 1 1 1 0
Basic Design Approach

*Six Step Process*

1. Understand the statement of the Specification

2. Obtain an abstract specification of the FSM (state table or state diagram)

3. Perform a state minimization

4. Perform state assignment

   \[(\# \text{state variables}) \geq \lceil \log_2 (\#\text{states}) \rceil\]

5. Choose FF types to implement FSM state register and build excitation table

6. Combinational circuit minimization
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
- nickel, dime
- dime, nickel
- two dimes
- two nickels, dime

**Draw state diagram:**

Inputs: N, D, reset

Output: open
Vending Machine Example

Step 3: State Minimization

reuse states whenever possible

Symbolic State Table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Inputs D N</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>
## 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>( Q_1 )</td>
<td>( D )</td>
<td>( D_1 )</td>
<td>( D_0 )</td>
</tr>
<tr>
<td>( Q_0 )</td>
<td>( N )</td>
<td></td>
<td></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>X X</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>X X</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>0 1</td>
<td>1 1</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>X X</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>0 1</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>X X</td>
<td>X X</td>
<td>X</td>
</tr>
</tbody>
</table>
### Vending Machine Example

**Step 5. Choose FF type for implementation**

#### J-K FF

#### Excitation Table

<table>
<thead>
<tr>
<th>Present State ($Q_1$ $Q_0$)</th>
<th>Inputs D N</th>
<th>Next State ($D_1$ $D_0$)</th>
<th>$J_1$ $K_1$ $J_0$ $K_0$</th>
<th>Output Open</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0 $X$ 0 $X$ 0 $X$ 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 1 0 $X$ 1 $X$ 0</td>
<td>$X$ 0 1 $X$ 0</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>1 0 1 $X$ 0 1 $X$ 0</td>
<td>$X$ 0 1 $X$ 0</td>
<td>$X$ 0 1 $X$ 0</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>$X$ $X$ $X$ $X$ $X$ $X$</td>
<td>$X$ $X$ $X$ $X$ $X$ $X$</td>
<td>$X$ $X$ $X$ $X$ $X$ $X$</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 1 0 $X$ 1 $X$ 0</td>
<td>$X$ 0 1 $X$ 0</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>1 0 1 $X$ 0 1 $X$ 0</td>
<td>$X$ 0 1 $X$ 0</td>
<td>$X$ 0 1 $X$ 0</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>1 1 1 $X$ 0 1 $X$ 0</td>
<td>$X$ 0 1 $X$ 0</td>
<td>$X$ 0 1 $X$ 0</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>$X$ $X$ $X$ $X$ $X$ $X$</td>
<td>$X$ $X$ $X$ $X$ $X$ $X$</td>
<td>$X$ $X$ $X$ $X$ $X$ $X$</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
<td>1 1 $X$ 0 1 $X$ 0</td>
<td>$X$ 0 1 $X$ 0</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>1 1 1 $X$ 0 1 $X$ 0</td>
<td>$X$ 0 1 $X$ 0</td>
<td>$X$ 0 1 $X$ 0</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>1 1 1 $X$ 0 1 $X$ 0</td>
<td>$X$ 0 1 $X$ 0</td>
<td>$X$ 0 1 $X$ 0</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>$X$ $X$ $X$ $X$ $X$ $X$</td>
<td>$X$ $X$ $X$ $X$ $X$ $X$</td>
<td>$X$ $X$ $X$ $X$ $X$ $X$</td>
<td></td>
</tr>
</tbody>
</table>
Vending Machine Example

Step 6. Combinational circuit minimization

J1 = D + Q0 N
K1 = 0
J0 = Q0 N + Q1 D
K0 = Q1 N
OPEN = Q1 Q0

K-map for J1

K-map for K1

K-map for J0

K-map for K0

7 Gates
Vending Machine Example

D FF easiest to use

\[ D1 = Q1 + D + Q0 \bar{N} \]
\[ D0 = N Q0 + Q0 \bar{N} + Q1 \bar{N} + Q1 D \]
\[ OPEN = Q1 Q0 \]

8 Gates
Moore and Mealy Machine Design Procedure

Definitions

**Moore Machine**

Outputs are function solely of the current state

Outputs change synchronously with state changes

**Mealy Machine**

Outputs depend on state AND inputs

Input change causes an immediate output change

Asynchronous signals
Moore and Mealy Machines

State Diagram Equivalents

Moore Machine

Outputs are associated with State

Mealy Machine

Outputs are associated with Transitions
Moore and Mealy Machines

States vs. Transitions

Mealy Machine typically has fewer states than Moore Machine for same output sequence

Same I/O behavior

Different # of states
Moore and Mealy Machines

Synchronous Mealy Machine

Inputs $X_i$ to Combinational Logic for Outputs and Next State

State Register

Clock

Outputs $Z_k$

state feedback

latched state AND outputs avoids glitchy outputs!
• *Mealy model of a serial adder*
• **State table**

<table>
<thead>
<tr>
<th>Present state</th>
<th>$a_i$</th>
<th>$b_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0, 0</td>
<td>0, 1</td>
</tr>
<tr>
<td>1</td>
<td>0, 1</td>
<td>1, 0</td>
</tr>
</tbody>
</table>

next state $c_{out_i}$, output $s_i$

• **State diagram**

```plaintext
reset

00/0
10/1
01/1

$S_0$
cin$_i$=0

11/0

11/0
01/0
10/0
11/1

$S_1$
cin$_i$=1

00/1
```
• *Mealy to Moore conversion*
State assignment:

\[ S = (c_{in_i} s_i) \]

\[ \begin{align*}
S_{00} &= 00 \\
S_{01} &= 01 \\
S_{10} &= 10 \\
S_{11} &= 11
\end{align*} \]

4 states : 2 FFs
No glitch

More states (FFs)

Delayed by 1 clock period
Synchronous Mealy machine

- $a_i$
- $b_i$
- $c_{in_i}$
- $c_{out_i}$
- $D$
- $Q$
- $C$
- $CLR$
- $s_i$
- $glitch$
- $no\ glitch$
- $clock$
- $reset$
Timing Control and Clocks

\[ T > t_{ff}^{\text{max}} + t_{c}^{\text{max}} + t_{su}^{\text{max}} \]

\[ t_{h} < t_{ff}^{\text{min}} + t_{c}^{\text{min}} \]

(usually \( t_{h} < t_{ff}^{\text{min}} \))

\[ t_{w} > t_{w}^{\text{min}} \]
\[ t_{PLH} = t_{PHL} = 2 \text{ ns} \]
\[ t_{c}^{\text{max}} = 6 \text{ ns} \text{ (use timing analyzer)} \]
\[ t_{\text{ff}}^{\text{max}} = t_{su}^{\text{max}} = 5 \text{ ns} \]
\[ E = 4 \text{ ns} \text{ (tolerance)} \]
\[ T = t_{\text{ff}}^{\text{max}} + t_{c}^{\text{max}} + t_{su}^{\text{max}} + E = 20 \text{ ns} \text{ (50 MHz clock)} \]
\[ t_{w}^{\text{min}} = 3 \text{ ns} \]
Finite State Machine Word Problems

Finite String Pattern Recognizer

A finite string recognizer has one input (X) and one output (Z). The output is asserted whenever the input sequence \text{ ..010 ...} has been observed, as long as the sequence 100 has never been seen.

Step 1. Understanding the problem statement

Sample input/output behavior:

X: 00101010010 ...
Z: 00010101000 ...

X: 11011010010 ...
Z: 00000001000 ...

 X: 11011010010 ...
Z: 00000001000 ...
Finite State Machine Word Problems

Finite String Recognizer

Step 2. Draw State Diagrams for the strings that must be recognized. i.e., 010 and 100.

Moore State Diagram
Reset signal places FSM in S0

Outputs 1

Loops in State
Exit conditions from state S3: have recognized ..010
if next input is 0 then have ..0100!
if next input is 1 then have ..0101 = ..01 (state S2)
Finite State Machine Word Problems

Finite String Recognizer

Exit conditions from S1: recognizes strings of form ..0 (no 1 seen) 
loop back to S1 if input is 0

Exit conditions from S4: recognizes strings of form ..1 (no 0 seen) 
loop back to S4 if input is 1
Finite State Machine Word Problems

Finite String Recognizer

S2, S5 with incomplete transitions

S2 = ..01; If next input is 1, then string could be prefix of (01)1(00)
S4 handles just this case!

S5 = ..10; If next input is 1, then string could be prefix of (10)1(0)
S2 handles just this case!

Final State Diagram
Finite State Machine Word Problems

Finite String Recognizer

Review of Process:

• Write down sample inputs and outputs to understand specification

• Write down sequences of states and transitions for the sequences to be recognized

• Add missing transitions; reuse states as much as possible

• Verify I/O behavior of your state diagram to insure it functions like the specification
Finite State Machine Word Problems

Traffic Light Controller

A busy highway is intersected by a little used farmroad. Detectors C sense the presence of cars waiting on the farmroad. With no car on farmroad, light remain green in highway direction. If vehicle on farmroad, highway lights go from Green to Yellow to Red, allowing the farmroad lights to become green. These stay green only as long as a farmroad car is detected but never longer than a set interval. When these are met, farm lights transition from Green to Yellow to Red, allowing highway to return to green. Even if farmroad vehicles are waiting, highway gets at least a set interval as green.

Assume you have an interval timer that generates a short time pulse (TS) and a long time pulse (TL) in response to a set (ST) signal. TS is to be used for timing yellow lights and TL for green lights.
Finite State Machine Word Problems

Traffic Light Controller

Picture of Highway/Farmroad Intersection:
Finite State Machine Word Problems

Traffic Light Controller

- Tabulation of Inputs and Outputs:

<table>
<thead>
<tr>
<th>Input Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>place FSM in initial state</td>
</tr>
<tr>
<td>C</td>
<td>detect vehicle on farmroad</td>
</tr>
<tr>
<td>TS</td>
<td>short time interval expired</td>
</tr>
<tr>
<td>TL</td>
<td>long time interval expired</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HG, HY, HR</td>
<td>assert green/yellow/red highway lights</td>
</tr>
<tr>
<td>FG, FY, FR</td>
<td>assert green/yellow/red farmroad lights</td>
</tr>
<tr>
<td>ST</td>
<td>start timing a short or long interval</td>
</tr>
</tbody>
</table>

- Tabulation of Unique States: Some light configuration imply others

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>Highway green (farmroad red)</td>
</tr>
<tr>
<td>S1</td>
<td>Highway yellow (farmroad red)</td>
</tr>
<tr>
<td>S2</td>
<td>Farmroad green (highway red)</td>
</tr>
<tr>
<td>S3</td>
<td>Farmroad yellow (highway red)</td>
</tr>
</tbody>
</table>
Finite State Machine Word Problems

Traffic Light Controller

State diagram:

S0: HG
S1: HY
S2: FG
S3: FY