CSE140L: Components and Design Techniques for Digital Systems Lab

FSMs and implementations

Tajana Simunic Rosing
Updates

- Temperature in the lab?
- CSE 140 Midterm #2 Tomorrow
  - 7 problems + one bonus
- Lab #3
  - Due May 21st
  - Larger size controller to implement – simplified iPod 😊
  - A problem to solve
Example: heuristics for state assignment

- S
- Sn
- 0 1
- 0 1 4
- 1 0 5
- 2 1 3
- 3 4 5
- 4 3 5
- 5 1 3

Diagram:
- Node labels: a, b, c, d
- Edges: i/j, i/k, i/l, i/j
- States: s0, s1, s2, s3, s4
- Transitions: 0/0, 1/1, 0/1, 1/0, 0/0, 0/0, 1/1, 1/0, 0/1, 0/0, 0/0
Timing: Definitions

- Cascaded FFs:
  - Tpd > Th
  - Tperiod > Tpd + Tsu
Digital combinational lock

- Door combination lock:
  - Press reset, then punch in 3 4 bit digits followed by enter, if the values match the code, the door opens; if there is an error the lock must be reset; once the door opens the lock must be reset
  - inputs: sequence of input values, reset
  - outputs: door open/close
  - memory: must remember combination or always have it available
Digital combination lock state diagram

- States: 5 states
  - represent point in execution of machine
  - each state has outputs
- Inputs: reset, new, results of comparisons
- Output: open/closed
Data-path and control structure

- **Data-path**
  - storage registers for combination values
  - multiplexer
  - comparator

- **Control**
  - finite-state machine controller
  - control for data-path (which value to compare)
State table for a combination lock

- Finite-state machine
  - refine state diagram to take internal structure into account
  - state table ready for encoding

<table>
<thead>
<tr>
<th>reset</th>
<th>new</th>
<th>equal</th>
<th>state</th>
<th>next state</th>
<th>mux</th>
<th>open/closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>S1</td>
<td>C1</td>
<td>closed</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>S1</td>
<td>S1</td>
<td>C1</td>
<td>closed</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>S1</td>
<td>ERR</td>
<td>-</td>
<td>closed</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>S1</td>
<td>S2</td>
<td>C2</td>
<td>closed</td>
</tr>
<tr>
<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>S3</td>
<td>OPEN</td>
<td>-</td>
<td>open</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Encodings for combination lock

- Encode state table

<table>
<thead>
<tr>
<th>reset</th>
<th>new</th>
<th>equal</th>
<th>state</th>
<th>next state</th>
<th>mux</th>
<th>open/closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0001</td>
<td>001</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0001</td>
<td>001</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0001</td>
<td>0000</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0001</td>
<td>0010</td>
<td>010</td>
<td>0</td>
</tr>
<tr>
<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>0100</td>
<td>1000</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

mux is identical to last 3 bits of state
open/closed is identical to first bit of state
therefore, we do not even need to implement FFs to hold state, just use outputs

- state can be: S1, S2, S3, OPEN, or ERR
  - needs at least 3 bits to encode: 000, 001, 010, 011, 100
  - and as many as 5: 00001, 00010, 00100, 01000, 10000
  - choose 4 bits: 0001, 0010, 0100, 1000, 0000

- output mux can be: C1, C2, or C3
  - needs 2 to 3 bits to encode
  - choose 3 bits: 001, 010, 100

- output open/closed can be: open or closed
  - needs 1 or 2 bits to encode
  - choose 1 bit: 1, 0
Datapath part of combination lock

- Multiplexer
  - easy to implement as combinational logic with few inputs
Median filter FSM

- Remove single 0s between two 1s (output = NS3)
Median filter FSM

• Realized using the standard procedure and individual FFs and gates

<table>
<thead>
<tr>
<th>I</th>
<th>PS1</th>
<th>PS2</th>
<th>PS3</th>
<th>NS1</th>
<th>NS2</th>
<th>NS3</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>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<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>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</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>1</td>
</tr>
<tr>
<td>1</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>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</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>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

NS1 = Reset' (I)
NS2 = Reset' (PS1 + PS2 I)
NS3 = Reset' PS2
O = PS3
Median filter FSM

- But it looks like a shift register if you look at it right
The set input (S) does the median filter function by making the next state 111 whenever the input is 1 and PS2 is 1 (1 input to state x1x)
Implementation using PALs

- Programmable logic building block for sequential logic
  - macro-cell: FF + logic
    - D-FF
    - two-level logic capability like PAL (e.g., 8 product terms)
Equivalent Mealy and Moore state diagrams

- Moore machine
  - outputs associated with state
- Mealy machine
  - outputs associated with transitions
Vending machine example (Moore PLD mapping)

\[ D_0 = \text{reset}'(Q_0'N + Q_0N' + Q_1N + Q_1D) \]
\[ D_1 = \text{reset}'(Q_1 + D + Q_0N) \]
\[ \text{OPEN} = Q_1Q_0 \]
Vending machine (synch. Mealy PLD mapping)

OPEN = reset'(Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D)