Unit 14
State Machine Design

Outcomes

• I can create a state diagram to solve a sequential problem
• I can implement a working state machine given a state diagram

STATE MACHINES OVERVIEW

What is state?

• Decisions are generally influenced by not only what is happening ______, but based on the sum of __________ experiences
  – The sum of all previous experiences is what is known as state
• In a human, 'state' refers to the sum of everything that has happened that has led you to where you are now and influences your interpretation of your senses & thoughts
• In a circuit, 'state' refers to all the bits being remembered (________________ or memory)
• In software, 'state' refers to all the __________ values that are being used
**State Machine Block Diagram**

- A system that utilizes state is often referred to as a state machine (or finite state machine [FSM])
- Most state machines can be embodied in the following form
  - Logic examines what's happening NOW (inputs) & from the PAST (state) to produce outputs and update the state (which will be used in the future to change the decision)
- Inputs will go away or change, so state needs to summarize/capture anything that might be useful for the future

```
Logic
```

```
State (memory)
```

**State Machines**

- Provide the "brains" or control for electronic and electro-mechanical systems
- Implement a set of steps (or algorithm) to control or solve a problem
- **Goal is to generate output values at _____________**
- Combine Sequential and Combinational logic elements
  - Sequential Logic to remember what step (state) we're in
  - Combinational Logic to produce outputs and find what state to go to next
  - Generates outputs based on what state we're in and the input values
- Use ____________ (a.k.a. flowcharts) to specify the operation of the corresponding state machine

**State Machine Example**

- Design a sequence detector to check for the combination "101"
- Input, X, provides 1-bit per clock
- Check the sequence of X for "101" in successive clocks
- If "101" detected, output Z=1 (Z=0 all other times)

**Another State Diagram Example**

- “101” Sequence Detector should output F=1 when the sequence 101 is found in consecutive order
Correct Specification of State Diagrams

• Exactly ____________ from a state may be true at a time
  – Not ___, not ___, exactly _____
  – Make sure the conditions you associate with the arrows coming out of a state are ____________ (< 2 true) but ____________ (> 0 true)

State Machines

• State Machines can be broken into 3 sections of logic
  – State Memory (SM)
    • Just FF’s to remember the current state
  – Next State Logic (NSL)
    • Combo logic to determine the next state
    • Essentially implements the transition conditions
  – Output Function Logic (OFL)
    • Combo logic to produce the outputs

Correct Specification of State Diagrams 2

• Exactly one transition from a state may be true at a time
  – Not 2, not 0, exactly 1
  – Make sure the conditions you associate with the arrows coming out of a state are mutually exclusive (< 2 true) but all inclusive (> 0 true)

State Machine

NEX T S TAT E
The FF inputs will be the value of the next state (on the next clock edge the FF outputs will change based on the inputs)

CURRENT S T AT E
The FF outputs represent the current state (the state we’re in right now)
State Machine Outputs

- State Machine outputs can be classified according to how the outputs are produced
  - If $\text{Outputs} = f(\text{current state, external inputs})$...
    Moore-Style
  - If $\text{Outputs} = f(\text{current state})$...
    Mealy-Style

Moore-Style Outputs

- Moore-style outputs only depend on the current state
- Thus, they are valid ______ in the clock cycle and _______ nearly the entire
- Often requires extra states compared to Mealy-style implementations

Mealy-Style Outputs

- Mealy-style outputs depend not only on the current state but the external inputs
- Thus, they may not be valid until ______ in the clock cycle and ______ during the cycle if the inputs change

State Machines

- Below is a circuit implementing a state machine, notice how it breaks into the 3 sections
**State Machine Design**

State Diagram vs. State Machine

**State Diagrams**
1. States
2. Transition Conditions
3. Outputs

**State Machine**
1. State Memory => FF’s
   - n-FF’s => 2^n states
2. Next State Logic (NSL)
   - combinational logic
   - logic for FF inputs
3. Output Function Logic (OFL)
   - MOORE: f(state)
   - MEALY: f(state + inputs)

State Machines require sequential logic to remember the current state. (w/ just combo logic we could only look at the current value of X, but now we can take 4 separate actions when X=0)

State Diagram for “101” Sequence Detector

- X=1
- X=0
- S1
- S10
- S101
- S0
- X=1
- X=0
- F=1
- F=0
- F=0

State Machine Design

- State machine design involves taking a problem description and coming up with a state diagram and then designing a circuit to implement that operation

---

**Problem Description** → **State Diagram** → **Circuit Implementation**

- Coming up with a state diagram is non-trivial
- Requires creative solutions
- Designing the circuit from the state diagram is done according to a simple set of steps
- To come up w/ a state diagram to solve a problem
  - Write out an algorithm or __________to solve the problem
  - Each _______ in your algorithm will usually be one state in your state diagram
  - Ask yourself what past inputs need to be ___________ and that will usually lead to a state representation
EXAMPLE 1

Alternating Detector

• Given bits coming in from a sensor, design a system that outputs true if sequential bits alternate or false if the same bit value is detected in that past two clock cycles.

Alternating Detector Example

• Can take a Mealy or Moore approach
  • Mealy often uses less states:
    – Let’s try...
    – If our output IS allowed to look at the current value of the input S (i.e. Mealy) then how many past values do we need to remember?

Alternating Detector

• Design a state machine to check if sensor produces two 0’s in a row (i.e. 2 consecutive spaces) or two 1’s in a row (i.e. 2 consecutive teeth).

  • G10 = Last cycle we got 1, two cycles ago we got 0
  • G01 = Last cycle we got 0, two cycles ago we got 1
  • G11 = Got 2 consecutive 1’s
  • G00 = Got 2 consecutive 0’s
Moore-Style Alternating Detector

Design a state machine to check if sensor produces two 0’s in a row (i.e. 2 consecutive spaces) or two 1’s in a row (i.e. 2 consecutive teeth)

- **G10** = Last cycle we got 1, two cycles ago we got 0
- **G01** = Last cycle we got 0, two cycles ago we got 1
- **G11** = Got 2 consecutive 1’s
- **G00** = Got 2 consecutive 0’s

<table>
<thead>
<tr>
<th>Current State</th>
<th>Input (S)</th>
<th>Next State</th>
<th>Output (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G01</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G01</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>G11</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>G11</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G00</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G10</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>G10</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

On Reset (power on)

6 Steps of State Machine Design

1. State Diagram
2. Transition/Output Table
3. State Assignment
   - Determine the # of FF’s required
   - Assign binary codes to replace symbolic names
4. Excitation Table (Rename Q* to D)
5. K-Maps for NSL and OFL
   - One K-Map for every FF input
   - One K-Map for every output of OFL
6. Draw out the circuit

Transition Output Table

- Convert state diagram to transition/output table
  - Show Next State & Output as a function of Current State and Input

<table>
<thead>
<tr>
<th>Current State</th>
<th>Input (S)</th>
<th>Next State</th>
<th>Output (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G01</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G01</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>G11</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>G11</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G00</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G10</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>G10</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Transition Output Table

- Now assign binary codes to represent states

<table>
<thead>
<tr>
<th>Current State</th>
<th>Input</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Q0</td>
<td>S</td>
<td>Q1*</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

State Assignment Mapping

<table>
<thead>
<tr>
<th>State</th>
<th>Q1</th>
<th>Q0</th>
</tr>
</thead>
<tbody>
<tr>
<td>G01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G11</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>G00</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>G10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### Transition Output Table

- Convert state diagram to transition/output table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Q</td>
<td>S = 0</td>
<td>S = 1</td>
</tr>
<tr>
<td>G01</td>
<td>0 0</td>
<td>G00</td>
</tr>
<tr>
<td>G11</td>
<td>0 1</td>
<td>G01</td>
</tr>
<tr>
<td>G10</td>
<td>1 1</td>
<td>G00</td>
</tr>
<tr>
<td>G00</td>
<td>1 0</td>
<td>G00</td>
</tr>
</tbody>
</table>

Further note, that since A is Moore it only depends on current state (Q's) and not inputs (S)

### Excitation Table

- The goal is to produce logic for the inputs to the FF’s (D_1,D_0)...these are the excitation equations

### Excitation Table

- Using your transition table you know what you want Q* to be, but how can you make that happen?
- For D-FF’s Q* will be ______________ is at the edge

### Excitation Table

- In a D-FF Q* will be whatever D is, so if we know what we want Q* to be just make sure that’s what the D input is
Karnaugh Maps

Now need to perform K-Maps for D1, D0, and A

<table>
<thead>
<tr>
<th>Current State</th>
<th>S = 0</th>
<th>S = 1</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Q1 Q0</td>
<td>D1 D0</td>
<td>D1 D0</td>
<td>A</td>
</tr>
<tr>
<td>G01 0 0</td>
<td>1 0</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>G11 0 1</td>
<td>0 0</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>G10 1 1</td>
<td>0 0</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>G00 1 0</td>
<td>1 0</td>
<td>1 1</td>
<td>0</td>
</tr>
</tbody>
</table>

D1 =

<table>
<thead>
<tr>
<th>State Q1 Q0</th>
<th>S = 0</th>
<th>S = 1</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>State D1 D0</td>
<td>D1 D0</td>
<td>D1 D0</td>
<td>A</td>
</tr>
<tr>
<td>G01 0 0</td>
<td>1 0</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>G11 0 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>G10 1 1</td>
<td>0 0</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>G00 1 0</td>
<td>1 0</td>
<td>1 1</td>
<td>0</td>
</tr>
</tbody>
</table>

D0 =

Implementing the Circuit

Implements the alternating detector

A = Q1'Q0' + Q1Q0' = Q1 XNOR Q0
Implementing an Initial State

- How can we make the machine start in G0 on reset (or power on?)
- Flip-flops by themselves will initialize to a random state (1 or 0) when power is turned on.

![Diagram showing state transition](image)

- Use the CLEAR and PRESET inputs on our flip-flops in the state memory
  - When CLEAR is active the FF initializes Q=0
  - When PRESET is active the FF initializes Q=1

- We assigned G0 the binary code $Q_1Q_0=00$ so we must initialize our Flip-Flop's to 00.

- Use the CLR inputs of your FF’s along with the RESET signal to initialize them to 0’s.
Implementing an Initial State

- We don't want to initialize our flip-flops to 1's (only Q1Q0=00) so we just don't use PRE (tie to 'off'='0')

![Diagram of flip-flops with logic connections]

Alternate State Assignment

- Important Fact: The codes we assign to our states can have a big impact on the size of the NSL and OFL
- Let us work again with a different set of assignments

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 Q0</td>
<td>S=0 S=1</td>
<td>A</td>
</tr>
<tr>
<td>State</td>
<td>Q1 Q0</td>
<td>State</td>
</tr>
<tr>
<td>G01 0 0</td>
<td>G00 0 0</td>
<td>G10 0 1</td>
</tr>
<tr>
<td>G11 0 1</td>
<td>G01 0 1</td>
<td>G11 1 0</td>
</tr>
<tr>
<td>G10 1 1</td>
<td>G00 1 1</td>
<td>G11 0 0</td>
</tr>
<tr>
<td>G00 1 0</td>
<td>G01 1 0</td>
<td>G11 0 1</td>
</tr>
</tbody>
</table>

Old Assignments

New Assignments

Implementing an Initial State

- When RESET is activated Q's initialize to 0 and then when it goes back to 1 the Q's look at the D inputs

![Diagram of flip-flops with logic connections showing state transitions]

Alternate State Assignment

![Table comparing current states to next states and output values]
EXAMPLE 2

Traffic Light Controller

- Design the controller for a traffic light at an intersection
  - Main street has a protected turn while small street does not
    - Sensors embedded in the street to detect cars waiting to turn
    - Let $S = \_\_\_\_\_\_\_\_\_\_$ to check if any car is waiting
  - Simplify and only have Green and Red lights (no yellow)

State Assignment

- Design of the traffic light controller with main turn arrow
- Represent states with some binary code
  - Codes: 3 States => 2 bit code: $00=SSG$, $10=MSG$, $11=MTG$

K-Maps

- Find logic for each FF input by using K-Maps

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State $S=0$</th>
<th>Next State $S=1$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>$Q_0$</td>
<td>$Q_1$</td>
<td>State $Q_0^*$</td>
</tr>
<tr>
<td>SS</td>
<td>0</td>
<td>0</td>
<td>S1</td>
</tr>
<tr>
<td>N/A</td>
<td>0</td>
<td>1</td>
<td>S1</td>
</tr>
<tr>
<td>MT</td>
<td>1</td>
<td>1</td>
<td>S1</td>
</tr>
<tr>
<td>MS</td>
<td>1</td>
<td>0</td>
<td>S1</td>
</tr>
</tbody>
</table>

$D_1 = Q_1^* + Q_2$

$D_5 = S + Q_1^*$
EXAMPLE 3

Water Pump

- Implement the water pump controller using the High and Low sensors as inputs

Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>0</td>
</tr>
<tr>
<td>ON</td>
<td>1</td>
</tr>
</tbody>
</table>

Note: The State Value, Q forms the Pump output (i.e. 1 when we want the pump to be on and 0 otherwise)

EXAMPLE 4
State Machine Example

- Design a sequence detector to check for the combination "1011"
- Input, X, provides 1-bit per clock
- Check the sequence of X for "1011" in successive clocks
- If "1011" detected, output Z=1 (Z=0 all other times)

State Diagram

- Be sure to handle overlapping sequences

Transition Output Table

- Translate the state diagram into the transition output table
### NSL & OFL

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Out put</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = 0</td>
<td>D2 D1 D0</td>
<td>Z</td>
</tr>
<tr>
<td>X = 1</td>
<td>D2 D1 D0</td>
<td>Z</td>
</tr>
<tr>
<td>X = 0</td>
<td>D2 D1 D0</td>
<td>Z</td>
</tr>
<tr>
<td>X = 1</td>
<td>D2 D1 D0</td>
<td>Z</td>
</tr>
</tbody>
</table>

\[ D_3 = X \cdot Q_0' \cdot Q_1' \cdot Q_2' \]

\[ D_0 = X \]

\[ D_2 = X' \cdot Q_0' \cdot Q_1 \cdot Q_2' \]

\[ D_1 = X \cdot Q_0' \cdot Q_1' \cdot Q_2' \]

\[ Z = Q_2 \]

### Drawing the Circuit

\[ X \]

\[ Q_0 \]

\[ Q_1 \]

\[ Q_2 \]

\[ Z \]

### Waveform for 1011 Detector

**CLOCK**

**RESET**

\[ X \]

\[ Q_0 \]

\[ Q_1 \]

\[ Q_2 \]

**STATE**

**INITIAL STATE**

\[ Z \]

### SELECTED SOLUTIONS
Another State Diagram Example

• “101” Sequence Detector should output $F=1$ when the sequence 101 is found in consecutive order

- On Reset (power on)
  - $S_{init}$: $F=0$
  - $S_0$: $F=0$
  - $S_1$: $F=0$

- $X=0$:
  - $S_{init}$: $F=0$
  - $S_0$: $F=0$
  - $S_1$: $F=0$

- $X=1$:
  - $S_{init}$: $F=0$
  - $S_0$: $F=0$
  - $S_1$: $F=0$

- $X=1$:
  - $S_{init}$: $F=0$
  - $S_0$: $F=0$
  - $S_1$: $F=0$

We have to remember the 1,0,1 along the way.

Another State Diagram Example

• “101” Sequence Detector should output $F=1$ when the sequence 101 is found in consecutive order

- On Reset (power on)
  - $S_{init}$: $F=0$
  - $S_0$: $F=0$
  - $S_1$: $F=0$

- $X=0$:
  - $S_{init}$: $F=0$
  - $S_0$: $F=0$
  - $S_1$: $F=0$

- $X=1$:
  - $S_{init}$: $F=0$
  - $S_0$: $F=0$
  - $S_1$: $F=0$

A ‘0’ initially is not part of the sequence so stay in $S_{init}$.

Another ‘1’ in $S_1$ means you have 11, but that second ‘1’ can be the start of the sequence.

A ‘0’ in $S_{10}$ means you have 100 which can’t be part of the sequence.