Introduction

- Circuits:
  - Combinational Circuit
  - Sequential Circuit

- Sequential Machine (Circuit)
  - Mealy Machine
  - Moore Machine

Derivation of State Graphs and Tables and State Assignments

General Model for Mealy Machine

General Model for Moore Machine
1. Design of a Sequence Detector

- To illustrate the design of a clocked Mealy sequential circuit, we will design a sequence detector.
- The circuit is of the form:

\[ X \rightarrow \text{State} \rightarrow Z \]

- Design the sequence detector so that any input sequence ending in 101 will produce an output \( Z=1 \).
- The circuit does not reset when a 1 output occurs. A typical input sequence and the corresponding output sequence are:

\[
\begin{align*}
X &= 0 0 1 1 0 1 1 0 0 1 0 1 0 0 0 \\
Z &= 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0
\end{align*}
\]

(time: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)

- First, we will draw the state graph based on problem description

- Initially, we do not know how many flip-flops will be required.
- So we will designate the circuit states as \( S_0, S_1 \), etc.
- We will start with a reset state (initial state) designated as \( S_0 \).

- If \( X=0 \), the circuit stays in \( S_0 \) because the input sequence does not start with a 0.
- If \( X=1 \), the circuit must go to a new state (\( S_1 \)) to “remember” that the first input in the desired sequence has been received.
When in state $S_1$, if $X=0$, the circuit must change to a new state ($S_2$) to remember that the first two inputs of the desired sequence (101) have been received.

If $X=1$ in state $S_2$, the desired input sequence (101) is complete and the output should be 1. Since the last 1 in a sequence can also be the first 1 in a new sequence, we should return to $S_1$.

Where to go from $S_2$?

If $X=1$ in state $S_1$, we can stay in $S_1$

The sequence is simply restarted

If $X=0$ in state $S_2$, reset the circuit to $S_0$

Two 0’s in a row are received and 00 is not part of ‘101’

Figure 14-4: Mealy State Graph for Sequence Detector

We can then convert our state graph to a state table:

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State $X=0$</th>
<th>Next State $X=1$</th>
<th>Present Output $X=0$</th>
<th>Present Output $X=1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>$S_0$</td>
<td>$S_1$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_1$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$S_0$</td>
<td>$S_1$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

How many flip-flops are needed for this design?

Since there are 3 states, we only need 2 flip-flops for the circuit (2 memory bits).
Now we convert our state table into a transition table:

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State X = 0</th>
<th>X = 1</th>
<th>Present Output X = 0</th>
<th>X = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₀</td>
<td>S₀</td>
<td>S₁</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S₁</td>
<td>S₂</td>
<td>S₁</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S₂</td>
<td>S₀</td>
<td>S₁</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

From the transition table, we can plot:
- The next-state maps for the flip-flops and
- The map for the output function Z

Moore Machine Example
- The procedure for a Moore machine is similar to the one used for a Mealy machine.
- Rework the previous example as a Moore machine:
  - The circuit should produce an output of 1 only if an input sequence ending in 101 has occurred.
The output is written with the state.
- $S_0 \rightarrow$ initial state
- $S_1 \rightarrow '1'$
- $S_2 \rightarrow '10'$

When the machine is in $S_2$ and input is 1, the output must become 1; therefore, it cannot go $S_1$ and we need a new state $S_3$ with a 1 output:

Completed graph is below:
- The sequence 100 resets the circuit to $S_0$.
- A sequence 1010 takes the circuit back to $S_2$ because another 1 input should cause $Z$ to become 1 again.

State Table from State Graph:

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State $X = 0$</th>
<th>Next State $X = 1$</th>
<th>Present Output($Z$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>$S_0$</td>
<td>$S_1$</td>
<td>0</td>
</tr>
<tr>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_1$</td>
<td>0</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$S_3$</td>
<td>$S_1$</td>
<td>1</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$S_2$</td>
<td>$S_1$</td>
<td>1</td>
</tr>
</tbody>
</table>

Any difference between Mealy and Moore?

Transition Table from State Table:

<table>
<thead>
<tr>
<th>$A\bar{B}$</th>
<th>$X = 0$</th>
<th>$X = 1$</th>
<th>$Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>01</td>
<td>1</td>
</tr>
</tbody>
</table>
2. More Complex Design Problems

- The output $Z$ should be 1 if the input sequence ends in either 010 or 1001, and $Z$ should be 0 otherwise.

- Example Sequence:

  \[
  X = 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \\
  \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \\
  a \quad b \quad c \quad d \quad e \quad f \\
  Z = 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0
  \]

Mealy Example

- We will start construction of the state graph by working with the two sequences which lead to a 1 output.
- Then, we will later add arrows and states as required to make sure that the output is correct for other cases.

First Sequence: “010”

Next, we construct the part of the graph corresponding to the sequence 1001, starting from the reset state $S_0$. 

<table>
<thead>
<tr>
<th>State</th>
<th>Sequence Received</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>Reset</td>
</tr>
<tr>
<td>$S_1$</td>
<td>0</td>
</tr>
<tr>
<td>$S_2$</td>
<td>01</td>
</tr>
<tr>
<td>$S_3$</td>
<td>010</td>
</tr>
</tbody>
</table>

- $S_0$ reset
- $S_1$ 0 (but not 10)
- $S_2$ 01
- $S_3$ 10
- $S_4$ 1 (but not 01)
- $S_5$ 100
Now we fill in the missing arcs

With each arc, we first ask if we can go back to one of the previous states or do we have to create a new state?

Moore Example

- Design a Moore sequential circuit with one input $X$ and one output $Z$.
- The output $Z$ is to be 1 if the total number of 1’s received is odd and at least two consecutive 0’s have been received.
- A typical input and output sequence is:

$$X = 1 0 1 1 0 0 1 1$$

$\uparrow \uparrow \uparrow \uparrow$

$abced$ $Z = (0) 0 0 0 0 1 0 1$

<table>
<thead>
<tr>
<th>State</th>
<th>Sequence Received</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>reset or even 1’s</td>
</tr>
<tr>
<td>$S_1$</td>
<td>odd 1’s</td>
</tr>
<tr>
<td>$S_2$</td>
<td>even 1’s and ends in 0</td>
</tr>
<tr>
<td>$S_3$</td>
<td>even 1’s and 00 has occurred</td>
</tr>
<tr>
<td>$S_4$</td>
<td>00 has occurred and odd 1’s</td>
</tr>
</tbody>
</table>
3. Guideless for Construction of State Graphs

- No specific procedure for every problem, but the following guidelines are helpful:
  1. First, construct some sample input and output sequences to make sure that you understand the problem statement.
  2. Determine under what conditions, if any, the circuit should reset to its initial state.
  3. If only one or two sequences lead to a nonzero output, a good way to start is to construct a partial state graph for those sequences.

Example 1 (Mealy Machine)

- A sequential circuit has one input \((X)\) and one output \((Z)\). The circuit examines groups of four consecutive inputs.
  - \(Z = 1\) if the input sequence 0101 or 1001 occurs
  - The circuit resets after every four inputs.

  1. First, construct some sample input and output sequences.

A typical sequence of inputs and outputs is

\[
X = \begin{array}{c}
0101 \\
0010 \\
1001 \\
0100
\end{array}
\]

\[
Z = \begin{array}{c}
0001 \\
0000 \\
0001 \\
0000
\end{array}
\]
2. Determine under what conditions, if any, the circuit should reset to its initial state.

- Since the circuit examines groups of four consecutive inputs and resets after each group of four, the circuit should reset to $S_0$ after every fourth input is received.

4. Another way to get started is to determine what sequences or groups of sequences must be remembered by the circuit and set up states accordingly.

6. Check your graph to make sure there is one and only one path leaving each state for each combination of values of the input variables.

Example 2 (Mealy Machine)

- A sequential circuit with one input ($X$) and two outputs ($Z_1$ and $Z_2$).
- $Z_1 = 1$ occurs every time the input sequence 100 is completed, provided that the sequence 010 has never occurred.
- $Z_2 = 1$ occurs every time the input sequence 010 is completed.
- Note that once a $Z_2 = 1$ output has occurred, $Z_1 = 1$ cannot occur but not vice versa.

A typical sequence of inputs and outputs is:

- $X = 1 0 0 1 1 0 0 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 0 0$
- $Z_1 = 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0$
- $Z_2 = 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 1 0$
Partial Graphs for Example 2

Keeping track of what is remembered by each state will help us make the correct state graph.

State Descriptions for Example 2:

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>No progress on 100</td>
</tr>
<tr>
<td>S1</td>
<td>Progress of 1 on 100</td>
</tr>
<tr>
<td>S2</td>
<td>Progress of 10 on 100</td>
</tr>
<tr>
<td>S3</td>
<td>No progress on 100</td>
</tr>
<tr>
<td>S4</td>
<td>Progress of 1 on 100</td>
</tr>
<tr>
<td>S5</td>
<td>Progress of 0 on 100</td>
</tr>
<tr>
<td>S6</td>
<td>Progress of 01 on 100</td>
</tr>
<tr>
<td>S7</td>
<td>No progress on 010</td>
</tr>
</tbody>
</table>

State Table for Example 2

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State X = 0</th>
<th>Next State X = 1</th>
<th>Output X = 0</th>
<th>(Z1Z2) X = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S3</td>
<td>S1</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
<td>S1</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td>S3</td>
<td>S3</td>
<td>S4</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>S4</td>
<td>S5</td>
<td>S1</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>S5</td>
<td>S5</td>
<td>S6</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>S6</td>
<td>S5</td>
<td>S7</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>S7</td>
<td>S5</td>
<td>S7</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>
5. Alphanumeric State Graph Notation

- When a state sequential circuit has several inputs, it is often convenient to label the state graph arcs with alphanumeric input variable names instead of 0's and 1's.

State Graphs with Variable Names on Arc Labels:

Property Specified State Graphs

- In general, a completely specified state graph has the following properties:
  1. When we OR together all input labels on arcs emanating from a state, the result reduces to 1.
  2. When we AND together any pair of input labels on arcs emanating from a state, the result is 0.

State Table:

<table>
<thead>
<tr>
<th>PS</th>
<th>NS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR=00</td>
<td>00</td>
<td>S0</td>
</tr>
<tr>
<td>S0</td>
<td>S0</td>
<td>S2</td>
</tr>
<tr>
<td>S1</td>
<td>S1</td>
<td>S0</td>
</tr>
<tr>
<td>S2</td>
<td>S2</td>
<td>S1</td>
</tr>
</tbody>
</table>

Alphanumeric Notation for Mealy State Graphs

- $X_iX_j / Z_pZ_q$ means if inputs $X_i$ and $X_j$ are 1 (we don't care what the other input values are), the outputs $Z_p$ and $Z_q$ are 1 (and the other outputs are 0).
  - That is, for a circuit with four inputs ($X_1, X_2, X_3,$ and $X_4$) and four outputs ($Z_1, Z_2, Z_3,$ and $Z_4$), $X_iX_4 / Z_2Z_3$ is equivalent to 1--0 / 0110.

- This type of notation is very useful for large sequential circuits with many inputs and outputs.
15.8 – Guidelines for State Assignments

- The cost of logic strongly depends on the way state assignments are made.
- Making state assignments is a challenge in design.
- Trail-and-error method is useful for a machine with small number of states.
  - 3 states → 2 FF → 4 possibilities for S1, 3 for S2, and 2 for S3
  - 4×3×2 = 24 possibilities (some assignments are equivalent)
- This is not practical for machines with large number of states.
- Guideline method produces good solutions for some problems, but sometimes it is not satisfactory.

The following guidelines are useful in making assignments:

- This will place 1’s together (or 0’s) on the next-state maps:
  1. States which have the same next state for a given input should be given adjacent assignments.
  2. States which are the next states of the same state should be given adjacent assignments.
  3. States which have the same output for a given input should be given adjacent assignments.
- Place 1’s together on the output maps.

Assignments for two states are said to be adjacent if they differ in only one variable.
- Ex: 010 and 011 are adjacent; 010 and 001 are not adjacent.

How to use these guidelines:

- First, write down the sets of states which should be adjacent.
- Then, using K-maps try to satisfy as many adjacencies as possible.
- A fair amount of trial-and-error may be required.

When filling the map,

- Assign the starting state to ‘0’ square on the map.
- Simplifies the initialization of the circuit.
- Adjacency conditions from Guide 1 and 2 that are required 2 or 3 times should be satisfied first.
- When 3 or 4 states are required to be adjacent, these states should be placed within a group of four adjacent squares.
- If output table is considered, then Guide 3 should be considered.
  - Guide 3 has less priority than Guide 1 and 2 if there is single output.
  - If there are 2 or more outputs, Guide 3 may have higher priority.
Example

<table>
<thead>
<tr>
<th>ABC</th>
<th>X = 0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>S₀</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>110</td>
<td>S₁</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>S₁</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>111</td>
<td>S₁</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>011</td>
<td>S₁</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>S₁</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>S₀</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(a) State table

Guideline 1:
- S₀, S₂, S₄, S₆ be adjacent
- S₁ is next state when X = 0
- S₂ and S₃
- S₃ is next state when X = 0
- S₀, S₁, S₂, S₃
- S₄ and S₆
- S₆ is next state when X = 1
- S₀, S₁, S₃, S₅
- S₂ is next state when X = 1
- S₄ and S₆

Guideline 2:
- S₁, S₂ (next states of S₀)
- S₂, S₁ (next states of S₁)
- S₁, S₄
- S₂, S₃ (two times: S₃ & S₅)
- S₃, S₄ (two times: S₄ & S₆)

*Guideline 3 is not considered in this example

15.9 – Using One-Hot State Assignment

- When designing with CLPDs or FPGAs, each logic cell contains one or more FFs
- These FFs are there whether we use them or not
  - Instead of number of FF, try to reduce number of logic cells used and the interconnections between them
  - Because the propagation delay typically depends on number of cell used
- One-hot state assignment may help to accomplish this
  - The one-hot assignment uses one flip-flop for each state
    - So, a state machine with N states requires N flip-flops
    - Exactly one of the flip-flops is set to 1 in each state

Example

- Partial graph is given
  - 4 states (S₀, S₁, S₂, S₃)
  - 4 FFs (Q₀, Q₁, Q₂, Q₃)
- State Assignment
  - S₀ = 1000
  - S₁ = 0100
  - S₂ = 0010
  - S₃ = 0001
- Next-state and output equations can be written by inspecting the state graph

Guideline 1: (S₀, S₂, S₄, S₆), (S₃, S₅), (S₀, S₁, S₃, S₅), (S₄, S₆)
Guideline 2: (S₁, S₂), (S₂, S₃), (S₁, S₄), 2×(S₂, S₅), 2×(S₁, S₆)

Cost of realizing D FF input equations:
- 6 gates, 13 inputs
- If straight binary assignment were used, cost:
  - 10 gates, 39 inputs
How to write next-state equations?
- There are 4 arcs leading into S3
- So, 4 conditions under which next state is S3
  - PS=S0 and X1=1
  - PS=S1 and X2=1
  - PS=S2 and X3=1
  - PS=S3 and X4=1
- Next state of Q3 =1 under these conditions; otherwise, Q3=0
  - Q3 = X1Q0Q1'Q2'Q3' + X2Q0'Q1Q2'Q3' + X3Q0'Q1'Q2Q3' + X4Q0'Q1'Q2'Q3
- Q0 = 1 implies that Q1=Q2=Q3= 0
  - Q3 = X1Q0' + X2Q1 + X3Q2 + X4Q3

How to write output equations?
- Z1=1 when PS=S0 and X1=1 & when PS=S2 and X3=1
  - Z1 = X1Q0 + X3Q2
- Similarly, Z2 = X2Q1 + X4Q3

What if the FFs used do not have a preset input?
- That is, Q0 can not be set to 1 for initial state
- Replace Q0 with Q0 throughout

State assignments for the previous example:
- S0=0000, S1=0100, S2=0010, S3=0001

Modified Equations:
- Q3 = X1Q0' + X2Q1 + X3Q2 + X4Q3
- Z1 = X1Q0' + X3Q2
- Z2 = X2Q1 + X4Q3

Example 2
A sequential circuit that controls a binary multiplier
- 3 inputs: St, M, K
- 4 outputs: Load, Ad, Sh, Done
- Starting in S0

How many FFs?
- 4 states → 4 FFs
- S0=1000
- S1=0100
- S2=0010
- S3=0001
Next state equation for $Q_0^+$:
- 2 arcs leading to $S_0 \rightarrow 2$ terms in the equation
- $Q_0^+ = Q_0S_0' + Q_3$

Next state equation for $Q_1^+$:
- 3 arcs leading to $S_0 \rightarrow 3$ terms
- $Q_1^+ = Q_0S_0 + Q_1K'M' + Q_2K'$

Output equation for $S_h$:
- $S_h$ appears in 4 places
- $S_h = 1$ in $S_1$ if $K'M' = 1$ or $KM' = 1$; also in $S_2$ if $K' = 1$ or $K = 1$
- $S_h = Q_1(K'M' + KM') + Q_2(K' + K) = Q_1M' + Q_2$

When designing with CPLDs or FPGAs, try both an assignment with minimum number of state variables and a one-hot state assignment
- If area is the concern, choose one with minimum number of logic cells
- If speed is the concern, choose the fastest one