Overview of Chapter 4

- Types of Sequential Circuits
- Storage Elements
  - Latches
  - Flip-Flops
- Sequential Circuit Analysis
  - State Tables
  - State Diagrams
- Sequential Circuit Design
  - Specification
  - Assignment of State Codes
  - Implementation
  - HDL Representation
Developing the State Diagram

- A **state** is an abstraction of the history of the past applied inputs to the circuit (including the “null” input at power-up or reset).
  - The interpretation of “past inputs” is tied to the synchronous operation of the circuit. E.g., an input value is measured only during the setup-hold time interval for an edge-triggered flip-flop.

- **Examples:**
  - State A represents the fact that a 1 input has occurred among the past inputs.
  - State B represents the fact that a 0 immediately followed by a 1 has occurred among the most recent past two inputs.

Developing the State Diagram (Continued)

- In specifying a circuit, we use **states** to remember **meaningful properties of past input sequences** that are essential to predicting **future output values**.
- A **sequence recognizer** is a sequential circuit that produces a distinct output value whenever a prescribed pattern of input symbols occur in sequence, i.e., **recognizes** an input sequence occurrence.
- We will develop a procedure **specific to sequence recognizers** to convert a problem statement into a **state diagram**.
- Next, the **state diagram**, will be converted to a **state table** from which the circuit will be designed.
**Sequence Recognizer Procedure**

- To develop a sequence recognizer state diagram:
  - Begin in an initial state in which NONE of the initial portion of the sequence has occurred (typically “reset” state).
  - Add a state that recognizes that the first symbol has occurred.
  - Add states that recognize each successive symbol occurring.
  - The final state represents the input sequence (possibly less the final input value) occurrence.
  - Add state transition arcs which specify what happens when a symbol not in the proper sequence has occurred.
  - Add other arcs on non-sequence inputs which transition to states that represent the input subsequence that has occurred.
- The last step is required because the circuit must recognize the input sequence regardless of where it occurs within the overall sequence applied since “reset.”

**Sequence Recognizer Example**

- **Example:** Recognize the sequence 1101
  - Note that the sequence 111101 contains 1101 and "11" is a proper sub-sequence of the sequence.
  - Thus, the sequential machine must remember that the first two one's have occurred as it receives another symbol.
  - Also, the sequence 1101101 contains 1101 as both an initial subsequence and a final subsequence with some overlap, i.e., 1101101 or 1101101.
  - And, the 1 in the middle, 1101101, is in both subsequences.
  - The sequence 1101 must be recognized each time it occurs in the input sequence.
Example: Recognize 1101

- Define states for the sequence to be recognized:
  - assuming it starts with first symbol,
  - continues through each symbol in the sequence to be recognized, and
  - uses output 1 to mean the full sequence has occurred,
  - with output 0 otherwise.

- Starting in the initial state (Arbitrarily named "A"):
  - Add a state that recognizes the first "1."
  - State "A" is the initial state, and state "B" is the state which represents the fact that the "first" one in the input subsequence has occurred. The output symbol "0" means that the full recognized sequence has not yet occurred.

Recognize 1101 (Continued)

- After one more 1, we have:
  - C is the state obtained when the input sequence has two "1"s.

- Finally, after 110 and a 1, we have:
  - Transition arcs are used to denote the output function (Mealy Model)
  - Output 1 on the arc from D means the sequence has been recognized
  - To what state should the arc from state D go? Remember: 1101101?
  - Note that D is the last state but the output 1 occurs for the input applied in D. This is the case when a Mealy model is assumed.
Recognize 1101 (Continued)

- Clearly the final 1 in the recognized sequence 1101 is a sub-sequence of 1101. It follows a 0 which is not a sub-sequence of 1101. Thus it should represent the same state reached from the initial state after a first 1 is observed. We obtain:

  ![Diagram](https://example.com/diagram.png)

The state have the following abstract meanings:

- A: No proper sub-sequence of the sequence has occurred.
- B: The sub-sequence 1 has occurred.
- C: The sub-sequence 11 has occurred.
- D: The sub-sequence 110 has occurred.
- The 1/1 on the arc from D to B means that the last 1 has occurred and thus, the sequence is recognized.
Complete the Diagram (1101)

- The other arcs are added to each state for inputs not yet listed. Which arcs are missing?

\[
\begin{array}{ccc}
A & \rightarrow & B \\
1/0 & & 1/0 \\
\rightarrow & & \rightarrow \\
B & \rightarrow & C \\
1/0 & & 0/0 \\
\rightarrow & & \rightarrow \\
C & \rightarrow & D \\
0/0 & & 1/1 \\
\end{array}
\]

- Answer:
  - "0" arc from A
  - "0" arc from B
  - "1" arc from C
  - "0" arc from D.

Add Missing Arcs

- State transition arcs must represent the fact that an input subsequence has occurred. Thus we get:

\[
\begin{array}{ccc}
A & \rightarrow & B \\
0/0 & & 1/0 \\
\rightarrow & & \rightarrow \\
B & \rightarrow & C \\
1/0 & & 0/0 \\
\rightarrow & & \rightarrow \\
C & \rightarrow & D \\
0/0 & & 1/1 \\
\end{array}
\]

- Note that the 1 arc from state C to state C implies that State C means two or more 1's have occurred.
1101 State Table from Diagram

- From the State Diagram, we can fill in the State Table.
- There are 4 states, one input, and one output. We will choose the form with four rows, one for each current state.
- From State A, the 0 and 1 input transitions have been filled in along with the outputs.

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State x=0</th>
<th>x=1</th>
<th>Output x=0</th>
<th>x=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>C</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>C</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>B</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Complete 1101 State Table

- From the state diagram, we complete the state table.
- What would the state diagram and state table look like for the Moore model?
Moore Model for 1101

- For the Moore Model, outputs are associated with states.
- We need to add a state "E" with output value 1 for the final 1 in the recognized input sequence.
  - This new state E, though similar to B, would generate an output of 1 and thus be different from B.
- The Moore model for a sequence recognizer usually has more states than the Mealy model.

Moore Diagram for 1101 (Cont.)

- We mark outputs on states for Moore model
- Arcs now show only state transitions
- Add a new state E to produce the output 1
- Note that the new state, E produces the same behavior in the future as state B. But it gives a different output at the present time. Thus these states do represent a different abstraction of the input history.
Moore State Table for 1101

- The state table is shown below
- Memory aid re more state in the Moore model: “Moore is More.”

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x=0</td>
<td>x=1</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>

Second State Diagram Example

- A register consists of an ordered set of $n$ flip-flops plus combinational logic to determine its next state.
- If a register can be designed as a set of $n$ identical cells, the register cell can be designed as a two-state sequential circuit.
Parallel Load Register with Synchronous Clear and Load

- **Register Specification**
  - Data_in (7:0)

- **Diagram**:

- **Table**:

<table>
<thead>
<tr>
<th>Operation</th>
<th>CLS</th>
<th>LDS</th>
<th>Result (Next State)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hold Reg</td>
<td>0</td>
<td>0</td>
<td>Data_out</td>
</tr>
<tr>
<td>Load Reg</td>
<td>0</td>
<td>1</td>
<td>Data_in</td>
</tr>
<tr>
<td>Clear Reg</td>
<td>1</td>
<td>X</td>
<td>00000000</td>
</tr>
</tbody>
</table>

Second Example:
Register Cell Design

- By definition, a register cell is a sequential circuit that:
  - contains one flip-flop (2 states)
  - has the flip-flop output as the primary external register output (Moore model)

- **Cell Diagram**:
Second Example: State Diagram Design

- Initial State: \( \text{RESET} \rightarrow \text{A/0} \)

- Add Load: \( \text{LDS,Data_in (RESET)} \)

- Add Clear: \( \text{CLS,LDS,Data_in (RESET)} \)

- Make the state unchanged (Hold Reg) by adding all unused input combinations for each state.

Logic and Computer Design Fundamentals
© 2001 Prentice Hall, Inc
Second Example:  
State Table

- From State Diagram: CLS, LDS, Data_in

<table>
<thead>
<tr>
<th>Input:</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>State:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>1</td>
</tr>
</tbody>
</table>