IE1204 Digital Design

L10: State Machines
(Part 2)

Masoumeh (Azin) Ebrahimi (masebr@kth.se)

Elena Dubrova (dubrova@kth.se)
KTH / ICT / ES
This lecture

- BV pp. 528-532, 557-567
A sequential system has a built-in memory - the output depends therefore BOTH on the current and previous value(s) of the input signal.
Basic method for the design of state machines

1. Analyze the specification of the circuit
2. Create state diagrams
3. Set up the state table
4. Minimize state table (this lecture)
5. Assign codes for states
6. Choose the type of flip-flops
7. Realize the circuit using Karnaugh maps
In a Moore-type machine output signals depend only on the current state.
Input vs. output - Moore

The input sequence

\[
\begin{array}{cc}
\text{In}_1 & \text{In}_2 \\
\hline
\text{O}_1 & \text{O}_2 \\
\end{array}
\]

State changed here (at clock edge)

Output visible \textbf{after} the state has changed

The output sequence
In a Mealy machine, output signals depend on both the current state and inputs.

- Mealy-type machine

- In a Mealy machine, output signals depend on both the current state and inputs.
Input vs. output - Mealy

The input sequence

\[
\begin{array}{cc}
\text{In}_1 & \text{In}_2 \\
\text{O}_1 & \text{O}_2 \\
\end{array}
\]

State changed here (at clock edge)

Output appears directly after the input has changed

IE1204 Digital Design, Autumn2016
Unused state

- Sometimes you get more states than you need when selecting a code
- "Unused" states must be taken care of so that the state machine does not hangs at the start-up (if reset is not used)

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state</th>
<th>Output</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_2y_1 )</td>
<td>( Y_2Y_1 )</td>
<td>( Y_2Y_1 )</td>
<td></td>
</tr>
<tr>
<td>A ( 00 )</td>
<td>00</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>B ( 01 )</td>
<td>00</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>C ( 10 )</td>
<td>00</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>dd</td>
<td>dd</td>
<td>d</td>
</tr>
</tbody>
</table>
Example: (0,0,1) sequence generator

3 states => 2 flip-flops. One unused state.

Dangerous transition (Machine hangs)

If the machine falls in this position, we want it to find move to another state as soon as possible
## Next-state function

<table>
<thead>
<tr>
<th>Current state</th>
<th>Output signal</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$y_2y_1$</td>
<td>$z$</td>
</tr>
<tr>
<td>S0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>S1</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

![State transition diagram]

---

IE1204 Digital Design, Autumn2016 11
Karnaugh maps

<table>
<thead>
<tr>
<th>Current state</th>
<th>Output signal</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>y₂y₁</td>
<td>Z</td>
</tr>
<tr>
<td>S0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>S1</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Y₂ = y₁

Y₁ = y₂\bar{y₁}

z = y₂
State table after Karnaugh minimization

<table>
<thead>
<tr>
<th>Current state</th>
<th>Output signal</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>y₂y₁</td>
<td>z</td>
<td>Y₂Y₁</td>
</tr>
<tr>
<td>S0</td>
<td>0 0</td>
<td>0 1</td>
</tr>
<tr>
<td>S1</td>
<td>0 1</td>
<td>1 0</td>
</tr>
<tr>
<td>S2</td>
<td>1 0</td>
<td>0 0</td>
</tr>
<tr>
<td>S2</td>
<td>1 1</td>
<td>1 0</td>
</tr>
</tbody>
</table>

The unused state goes to S2

(0,0,1) sequence generator
Logic circuit for the sequence

The implementation uses D flip-flops

\[ Y_2 = y_1 \]
\[ Y_1 = \overline{y_2} \overline{y_1} \]
\[ z = y_2 \]
Logic circuit in the structure of Moore machine

\[ Y_1 = \overline{y_2} \overline{y_1} \]
\[ Y_2 = y_1 \]
\[ z = y_2 \]
Logic circuit for the sequence

\[ Y_1 = \overline{y_2y_1} \]
\[ Y_2 = y_1 \]
\[ z = y_2 \]
State minimization

- When designing complex state machines, it often happens that there are equivalent states that can be grouped together to obtain a more efficient implementation.
- Two states $S_1$ and $S_2$ are called equivalent if and only if, for every possible input sequence, the same output will be produced regardless of whether $S_1$ or $S_2$ is the initial state.
State minimization

- The following example illustrates one minimization method which can be used for state minimization
- This method identifies states which are not equivalent (this is often easier)
- First, we introduce some terminology
If input \( w = 0 \) is applied to a state machine in state \( S_1 \) and the result is that the machine moves to state \( S_2 \), we say that \( S_2 \) is a 0-successor of \( S_1 \).

If input \( w = 1 \) is applied to a state machine in state \( S_1 \) and the result is that the machine moves to state \( S_3 \), we say that \( S_3 \) is a 1-successor of \( S_1 \).

We will refer to successors as \( k \)-successors, where \( k \) can be 0 or 1.
Two states are not equivalent if they have different output values.
State minimization
Basic idea

- Two states are **not** equivalent if at least one of their k-successors are not equivalent
Example
State minimization

A
z = 1

w = 0
w = 1

B
z = 1

w = 1

C
z = 0

w = 0
w = 1

D
z = 1

w = 0

E
z = 0

w = 0
w = 1

F
z = 0

w = 0

G
z = 0

w = 1

(Moore Machine)

7- states uses 3 flip-flops (2^3 = 8)

IE1204 Digital Design, Autumn 2016
### State table

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state $w=0$</th>
<th>Next state $w=1$</th>
<th>Output $z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>$z=0$</td>
<td>$z=1$</td>
<td>$z=0$</td>
<td></td>
</tr>
<tr>
<td>$w=0$</td>
<td>$w=1$</td>
<td>$w=0$</td>
<td></td>
</tr>
<tr>
<td>$w=1$</td>
<td>$w=0$</td>
<td>$w=1$</td>
<td></td>
</tr>
<tr>
<td>$w=1$</td>
<td>$w=0$</td>
<td>$w=1$</td>
<td></td>
</tr>
<tr>
<td>$w=0$</td>
<td>$w=1$</td>
<td>$w=0$</td>
<td></td>
</tr>
<tr>
<td>$w=1$</td>
<td>$w=0$</td>
<td>$w=1$</td>
<td></td>
</tr>
</tbody>
</table>

### Diagram

- States A, B, D, C, F, G, E
- Transitions:
  - A to B: $w=0$
  - B to C: $w=0$
  - C to A: $w=1$
  - A to D: $w=1$
  - D to B: $w=1$
  - B to F: $w=1$
  - F to G: $w=0$
  - G to A: $w=1$
  - E to A: $w=1$
  - A to E: $w=0$
  - E to C: $w=0$
  - C to E: $w=1$
  - E to F: $w=1$
**State table**

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state</th>
<th>Output z</th>
</tr>
</thead>
<tbody>
<tr>
<td>w = 0</td>
<td>w = 1</td>
<td></td>
</tr>
<tr>
<td>A z = 1</td>
<td>B z = 1</td>
<td>1</td>
</tr>
<tr>
<td>C z = 0</td>
<td>F z = 0</td>
<td>0</td>
</tr>
<tr>
<td>E z = 0</td>
<td>F z = 0</td>
<td>0</td>
</tr>
<tr>
<td>G z = 0</td>
<td>F z = 0</td>
<td>0</td>
</tr>
<tr>
<td>B z = 1</td>
<td>D z = 1</td>
<td>1</td>
</tr>
<tr>
<td>F z = 0</td>
<td>E z = 0</td>
<td>0</td>
</tr>
<tr>
<td>E z = 0</td>
<td>F z = 0</td>
<td>0</td>
</tr>
<tr>
<td>G z = 0</td>
<td>F z = 0</td>
<td>0</td>
</tr>
<tr>
<td>D z = 1</td>
<td>B z = 1</td>
<td>1</td>
</tr>
</tbody>
</table>

[Diagram showing state transitions]

IE1204 Digital Design, Autumn2016
Partition

- The minimization procedure first considers the states of a machine as a set and then breaks this set into partitions that are not equivalent.
- A partition consists of one or more blocks
  – each block contains states that may be equivalent
  – different blocks contain states that are definitely not equivalent
Example state minimization

- Start
  - Just one block containing all states
  - \( P_1 = (ABCDEFG) \)

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state ( w = 0 )</th>
<th>Next state ( w = 1 )</th>
<th>Output ( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>E</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>B</td>
<td>G</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>E</td>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>F</td>
<td>G</td>
<td>0</td>
</tr>
</tbody>
</table>
Stage 1:
- Which states have different outputs?
  - ABD has output $z = 1$
  - CEFG have output $z = 0$
  - $\Rightarrow P_2 = (ABD) \ (CEFG)$

States A, B, D can therefore never be equivalent to any of the conditions C, E, F, G so they form different groups
Example state minimization
\( P_2 = (ABD) (CEFG) \)

- Stage 2
  - Which states have different k-successors?

- Block ABD
  - 0-successor: \( A \rightarrow B, B \rightarrow D, D \rightarrow B \) (all transitions go to the same block)
  - 1-successor: \( A \rightarrow C, B \rightarrow F, D \rightarrow G \) (all transitions go to the same block)

- Block CEG
  - 0-successor: \( C \rightarrow F, E \rightarrow F, F \rightarrow E, G \rightarrow F \) (all transitions go to the same block)
  - 1-successor: \( C \rightarrow E, E \rightarrow C, F \rightarrow D, G \rightarrow G \) (\( F \rightarrow D \) goes to another block)

\( \Rightarrow P_3 = (ABD) (CEG) (F) \)
Example state minimization

$P_2 = (ABD) (CEFG)$

**Stage 2**

- Which states have different $k$-successors?

**Block ABD**

- 0-successor: $A \rightarrow B$, $B \rightarrow D$, $D \rightarrow B$ (all transitions go to the same block)

- 1-successor: $A \rightarrow C$, $B \rightarrow F$, $D \rightarrow G$ (all transitions go to the same block)

**Block CEG**

- 0-successor: $C \rightarrow F$, $E \rightarrow F$, $F \rightarrow E$, $G \rightarrow F$ (all transitions go to the same block)

- 1-successor: $C \rightarrow E$, $E \rightarrow C$, $F \rightarrow D$, $G \rightarrow G$ ($F \rightarrow D$ goes to another block)

$\Rightarrow P_3 = (ABD) (CEG) (F)$
Example state minimization

\[ P_3 = (ABD) \ (CEG) \ (F) \]

**Step 3**

- What states have different k-successors?

- **Block ABD**
  
  - 0-successor: \( A \rightarrow B, \ B \rightarrow D, \ D \rightarrow B \) (all transitions go to the same block)
  
  - 1-successor: \( A \rightarrow C, \ B \rightarrow F, \ D \rightarrow G \) (B \rightarrow F goes to another block)
  
  \( \Rightarrow P_4 = (AD) \ (B) \ (CEG) \ (F) \)

- **Block (CEG)**
  
  - 0-successor: \( C \rightarrow F, \ E \rightarrow F, \ G \rightarrow F \) (all transitions go to the same block)
  
  - 1-successor: \( C \rightarrow E, \ E \rightarrow C, \ G \rightarrow G \) (all transitions go to the same block)
  
  \( \Rightarrow P_4 = (AD) \ (B) \ (CEG) \ (F) \)

- **Table**

<table>
<thead>
<tr>
<th>Present state</th>
<th>( w = 0 )</th>
<th>( w = 1 )</th>
<th>Output ( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>E</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>B</td>
<td>G</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>E</td>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>F</td>
<td>G</td>
<td>0</td>
</tr>
</tbody>
</table>
Example state minimization
\[ P_4 = (AD)(B)(CEG)(F) \]

- Next partition \( P_5 \) becomes the same as \( P_4 \). Thus the procedure is finished.
  - States in each block are equivalent if they were not, their k-successors would have to be in different blocks
  - A becomes the representative of AD and C represents CEG.

## Final state table

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state</th>
<th>Output $z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w = 0$</td>
<td>$w = 1$</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
<td>F</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>E</td>
</tr>
<tr>
<td>D</td>
<td>B</td>
<td>G</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>E</td>
<td>D</td>
</tr>
<tr>
<td>G</td>
<td>F</td>
<td>G</td>
</tr>
</tbody>
</table>

Final State Table

### Final State Table

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state</th>
<th>Output z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$w = 0$</td>
<td>$w = 1$</td>
</tr>
</tbody>
</table>
| A             | B          | C        | 1
| B             | D          | F        | 1
| C             | F          | E        | 0
| D             | B          | G        | 1
| E             | F          | C        | 0
| F             | E          | D        | 0
| G             | F          | G        | 0

**Final State Table**

Final state diagram

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next State</th>
<th>Output z</th>
</tr>
</thead>
<tbody>
<tr>
<td>w = 0</td>
<td>w = 1</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>F</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>C</td>
<td>A</td>
</tr>
</tbody>
</table>

4 states needs 2 flip-flops ($2^2 = 4$).
Comparison

- Only 2 flip-flops are needed to implement 4 states in the minimized state table
- 3 flip-flops are needed to implement 7 states in the original state table
Some thought!

- Fewer state does not necessarily lead to a simpler design!
  - The advantage of state minimization is instead that it makes it easier to create the initial state diagram, when you do not have to get it to be minimal from the beginning!
Given an implementation of a synchronous circuit, we can produce its function by making the synthesis steps in a reverse order!

1. Get expressions for
   • next state decoder
   • output decoder
2. Get the state table
3. Draw the state diagram
Example: Analysis of a synchronous sequential circuit

It is difficult to figure out directly from the schematic how a sequential circuit behaves!
Example: Moore-machine!
Example: Analysis of a synchronous sequential circuit

1. Get expressions for
   
   - next state decoder
   - output decoder

\[
\begin{align*}
Y_1 &= w\bar{y}_1 + wy_2 \\
Y_2 &= wy_1 + wy_2
\end{align*}
\]
Example: Analysis of a synchronous sequential circuit

2. Get the state table

\[
Y_2 = wy_1 + wy_2 \quad Y_1 = w\overline{y}_1 + wy_2 \quad z = y_1 \cdot y_2
\]
3. Draw state diagram
   - Left as exercise for students... (but check the state table to make sure it is a bit sequence detector for three subsequent 1s)

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state</th>
<th>Output z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>w = 0</td>
<td>w = 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>A</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

Example: Analysis of a synchronous sequential circuit

IE1204 Digital Design, Autumn2016
State diagram

Sometimes you may need to change the order of the states to get a clearer chart.
C and D have changed places resulting in no intersecting state arrows.
ASM Charts

- State transition diagrams are convenient for describing the behavior of small state machines only.
- To describe larger state machines, another type of diagrams, called Algorithmic State Machine (ASM) charts are often used.
- An ASM is a flow diagram consisting of three types of elements: state box, decision box and conditional output box.
ASM Charts

State Name

(A) State Box (rectangle)

Output signals or actions (Moore type)

(B) Decision box (diamond)

Condition expression

0 (False) 1 (True)

Conditional outputs or actions (Mealy type)

(C) Conditional output box (Oval)
ASM Charts

- **State Box**
  - Represents a state in a FSM
    - Output values for state are given here (*Moore outputs*)

- **Decision Box**
  - Depending on the values of the input signals, it determines a transition to the next state

- **Conditional outputs Box**
  - Specifies the values of the outputs at a state transition (*Mealy outputs*)
ASM chart for 11 sequence detector (Moore)

Reset

A

0

w

1

B

0

w

1

C

z

z = 1 only in the state C

Reset

w = 0

A/z = 0

w = 1

B/z = 0

w = 0

w = 1

C/z = 1

w = 1
ASM chart for 11 sequence detector (Mealy)

z = 1 only when the state transition B-to-B with w = 1 takes place
To treat state machines in a formal way, we need a formal model. The following model can describe both Moore and Mealy machines.
Formal model for sequential circuits

- Inputs: \( w_1, \ldots, w_n \)
- Combinational Circuit
- Current-state variables: \( y_k, y_1 \)
- Next-state variables: \( Y_k, Y_1 \)
- Outputs: \( z_1, \ldots, z_m \)
A synchronous sequential circuit can be formally defined as:

\[ M = (W, Z, S, \varphi, \lambda) \]

- \( W, Z, \) and \( S \) are finite, nonempty sets of inputs, outputs and states, respectively.
- \( \varphi \) is the state transition function, such as \( S(t+1) = \varphi[W(t), S(t)] \)
- \( \lambda \) is the output function, such as \( \lambda(t) = \lambda(S(t)) \) for the Moore model and \( \lambda(t) = \lambda(W(t), S(t)) \) for the Mealy model.
Formal model for sequential circuits

\[ M = (W, Z, S, \varphi, \lambda) \]

\[ W_{inputs} = \{w_1, w_2, \ldots, w_m\} \]

\[ Z_{outputs} = \{z_1, z_2, \ldots, z_m\} \]

\[ S_{states} = \{S_1, S_2, \ldots, S_m\} \]

\[ y_{present-state-variables} = \{y_1, y_2, \ldots, y_m\} \]

\[ y_{next-state-variables} = \{Y_1, Y_2, \ldots, Y_m\} \]

\[ S(t + \Delta t) = Y_1 \ldots Y_m = \varphi(W(t), S(t)) \]

\[ \lambda_{Moore}(t) = \lambda(S(t)) \]

\[ \lambda_{Mealy}(t) = \lambda(W(t), S(t)) \]
Summary

• State minimization
• Analysis of a synchronous sequential circuit
• ASM charts
• Formal model for sequential circuits
• Next lecture: BV pp. 98-118, 418-426, 508-519