95-1 Under-Graduate Project
Design of Datapath Controllers

Speaker: 陳 圓 覺
Adviser: Prof. An-Yeu Wu
Date: 2006/11/01
Outline

- Sequential Circuit Model
- Finite State Machines
- Useful Modeling Techniques
Model of Sequential Circuits

- System outputs depend not only on current input
  - Depend on inputs
  - Depend on current state
- Fundamental components
  - Combinational circuits
  - Memory elements

![Diagram of Model of Sequential Circuits]

- Inputs
- Outputs
- Current State
- Next State
- Clock
Types of Memory Elements

- Flip-Flop
- Latch
- Registers
- Others
  - Register Files
  - Cache
  - Flash memory
  - ROM
  - RAM
**D-FF vs. D-Latch**

- FF is *edge sensitive* (can be either positive or negative edge)
  - At trigger edge of clock, input transferred to output
- Latch is *level sensitive* (can be either active-high or active-low)
  - When clock is active, input passes to output (transparent)
  - When clock is not active, output stays unchanged

![D-FF diagram](image1)

![D-Latch diagram](image2)
FF Based, Edge Trigger Clocking

- $T_d$ = delay of combinational logic
- $T_{cycle}$ = cycle time of clock
  - Duty cycle does not matter
- Timing requirements for $T_d$
  - $T_{dmax} < T_{cycle} - T_{setup} - T_{cq}$
    - no setup time violation
  - $T_{dmin} > T_{hold} - T_{cq}$
    - no hold time violation
Latch Based, Single Phase Clocking

- Aka. Pulse Mode clocking
- $T_{\text{cycle}} = \text{cycle time of clock}; \ T_w = \text{pulse width of clock}$

**Timing requirements for $T_d$**

- $T_{\text{dmax}} < T_{\text{cycle}} - T_{dq}$ ➔ data latched correctly
- $T_{\text{dmin}} > T_w - T_{dq}$ ➔ no racing through next stage
Comparison

- Flip-Flop Based
  - Larger in area
  - Larger clocking overhead ($T_{\text{setup}}, T_{cq}$)
  + Design more robust
    - Only have to worry about $T_{\text{dmax}}$
    - $T_{\text{dmin}}$ usually small, can be easily fixed by buffer
  + Pulse width does not matter

- Latch Based Single Phase
  + Smaller area
  + Smaller clocking overhead (only $T_{dq}$)
  - Worry about both $T_{\text{dmax}}$ and $T_{\text{dmin}}$
  - Pulse width DOES matter
    (unfortunately, pulse width can vary on chip)
D Flip-Flop with Positive-Edge Clock

module flop (Q, D, C, S, R);
    output Q;  // Flip-Flop Output
    input  D;  // Data Input
    input  C;  // Positive Edge Clock

    reg Q;     // Register Type

    always @(posedge C)
        begin
            Q <= D;
        end
endmodule
D Flip-Flop with Positive-Edge Clock

module flop (Q, D, C, S, R);
    output Q;  // Flip-Flop Output
    input  D;  // Data Input
    input  C;  // Positive Edge Clock
    input  R;  // Asynchronous Reset
    input  S;  // synchronous Set

    reg Q;  // Register Type

    always @(posedge C or negedge R)
        begin
            if (!R)
                Q <= 1'b0;
            else if (S)
                Q <= 1'b1;
            else
                Q <= D;
        end
endmodule
D-Latch Active High with Clear/Preset

module flop (Q, D, G, C, P);
  output Q;  // Flip-Flop Output
  input  D;  // Data Input
  input  G;  // Positive Gate
  input  C;  // synchronous Clear
  input  P;  // synchronous Preset

  reg Q;  // Register Type

  always @(D or G or C or P)
  begin
    if (!C)
      Q <= 1'b0;
    else if (P)
      Q <= 1'b1;
    else
      if (G)
        Q <= D;
  end
endmodule
Finite State Machine
What is FSM

- A model of computation consisting of
  - a set of states, (limited number)
  - a start state,
  - input symbols,
  - a transition function that maps input symbols and current states to a next state.

State transition diagram
Elements of FSM

- **Memory Elements (ME)**
  - Memorize Current States (CS)
  - Usually consist of FF or latch
  - N-bit FF have $2^n$ possible states

- **Next-state Logic (NL)**
  - Combinational Logic
  - Produce next state
    - Based on current state (CS) and input (X)

- **Output Logic (OL)**
  - Combinational Logic
  - Produce outputs (Z)
    - Based on current state, or
    - Based on current state and input
Mealy Machine

- Output is function of both
  - Input and current state
Moore Machine

- Output is function of current state only
  - Not function of inputs

Moore Machine diagram:

- Next-state Logic (NL)
- Memory Elements
- Output Logic (OL)
- Current State (CS)
- Input X
- Output Z
- State transitions:
  - CS = S₀
  - Z = 1
  - Transition 0 → 1
  - Transition 1 → 1
  - Transition 0 → 0
  - Transition 0 → 1

X = 0
Mealy Finite State Machine

A serially-transmitted BCD (8421 code) word is to be converted into an Excess-3 code. An Excess-3 code word is obtained by adding 3 to the decimal value and taking the binary equivalent. Excess-3 code is self-complementing [Wakerly, p. 80], i.e. the 9's complement of a code word is obtained by complementing the bits of the word.

<table>
<thead>
<tr>
<th>Decimal Digit</th>
<th>8-4-2-1 Code (BCD)</th>
<th>Excess-3 Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0011</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0100</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0101</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>0110</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>0111</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>1000</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>1001</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>1010</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1011</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>1100</td>
</tr>
</tbody>
</table>
Mealy Finite State Machine

The serial code converter is described by the state transition graph of a Mealy FSM.

The vertices of the state transition graph of a Mealy machine are labeled with the states.

The branches are labeled with (1) the input that causes a transition to the indicated next state, and (2) with the output that is asserted in the present state for that input.

The state transition is synchronized to a clock.

The state table summarizes the machine's behavior in tabular format.

Next State/Output Table

<table>
<thead>
<tr>
<th>State</th>
<th>Next State/Output</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>S_0</td>
<td>S_1 / 1</td>
<td></td>
</tr>
<tr>
<td>S_1</td>
<td>S_3 / 1</td>
<td></td>
</tr>
<tr>
<td>S_2</td>
<td>S_4 / 0</td>
<td></td>
</tr>
<tr>
<td>S_3</td>
<td>S_5 / 0</td>
<td></td>
</tr>
<tr>
<td>S_4</td>
<td>S_5 / 1</td>
<td></td>
</tr>
<tr>
<td>S_5</td>
<td>S_0 / 0</td>
<td></td>
</tr>
<tr>
<td>S_6</td>
<td>S_0 / 1</td>
<td></td>
</tr>
</tbody>
</table>
Design of a Mealy Finite State Machine

To design a D-type flip-flop realization of a FSM having the behavior described by a state transition graph, (1) select a state code, (2) encode the state table, (3) develop Boolean equations describing the input of a D-type flip-flop, and (4) using K-maps, optimize the Boolean equations.

<table>
<thead>
<tr>
<th>State</th>
<th>Next State/Output Table</th>
<th>State Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>state</td>
<td>next state/output</td>
</tr>
<tr>
<td>S₀</td>
<td>0</td>
<td>S₁ / 1</td>
</tr>
<tr>
<td>S₁</td>
<td>1</td>
<td>S₃ / 1</td>
</tr>
<tr>
<td>S₂</td>
<td>1</td>
<td>S₄ / 0</td>
</tr>
<tr>
<td>S₃</td>
<td>1</td>
<td>S₅ / 0</td>
</tr>
<tr>
<td>S₄</td>
<td>1</td>
<td>S₅ / 1</td>
</tr>
<tr>
<td>S₅</td>
<td>1</td>
<td>S₀ / 0</td>
</tr>
<tr>
<td>S₆</td>
<td>1</td>
<td>S₀ / 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encoded Next state/ Output Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>S₀</td>
</tr>
<tr>
<td>S₁</td>
</tr>
<tr>
<td>S₂</td>
</tr>
<tr>
<td>S₃</td>
</tr>
<tr>
<td>S₄</td>
</tr>
<tr>
<td>S₅</td>
</tr>
<tr>
<td>S₆</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

State Assignment

<table>
<thead>
<tr>
<th>q₂</th>
<th>q₁</th>
<th>q₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

state | input | input |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S₀</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S₁</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S₂</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S₃</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S₄</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S₅</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S₆</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Design of a Mealy Finite State Machine

Note: We will optimize the equations individually. In general - this does not necessarily produce the optimal (area, speed) realization of the logic. We'll address this when we consider synthesis.

\[ q_2^+ = q_1'q_0'B_{in} + q_2'q_0'B_{in}' + q_2q_1q_0 \]

\[ q_2^+ = q_1'q_0'B_{in} + q_2'q_0'B_{in}' + q_2q_1q_0 \]

\[ q_2^+ = q_1'q_0'B_{in} + q_2'q_0'B_{in}' + q_2q_1q_0 \]
Design of a Mealy Finite State Machine

Realization of the sequential BCD-to-Excess-3 code converter (Mealy machine):

\[
q_2^+ = q_1'q_0'B_{in} + q_2'q_0B_{in}' + q_2q_1q_0
\]

\[
q_2^+ = \frac{q_1'q_0'B_{in} + q_2'q_0B_{in}'}{q_2q_1q_0}
\]
Design of a Mealy Finite State Machine

Simulation results for Mealy machine:
Building Behavioral Models
Modeling FSM in Verilog

- Sequential Circuits
  - Memory elements of States (CS)
- Combinational Circuits
  - Next-state Logic (NL)
  - Output Logic (OL)
- Three coding styles
  - (1) Separate CS, OL and NL
  - (2) Combines NL+ OL, separate CS
  - (3) Combine CS + NL, separate OL
Coding Style 1 – Separate CS, NL, OL

- **CS**
  ```
  always @ (posedge clk)
  current_state <= next_state;
  ```

- **NL**
  ```
  always @ (current_state or In)
  case (current_state)
  S0: case (In)
   In0: next_state<= S1;
   In1: next_state<= S0;
   ...
  endcase //In
  S1: ...
  S2: ...
  endcase //current_state
  ```

- **OL**
  ```
  // if Moore
  always @ (current_state)
  Z<= output_value;
  // if Mealy
  always @ (current_state or In)
  Z<= output_value;
  ```
Coding Style 2 –
Combine NL+OL; Separate CS

**CS**

```verilog
always @ (posedge clk)
    current_state <= next_state;
```

**NL+OL**

```verilog
always @ (current_state or In)
    case (current_state)
        S0: begin
            case (In)
                In0: begin
                    next_state <= S1;
                    Z <= values; // Mealy
                end
                In1: . . .
            endcase // In
        end
        In1: . . .
    endcase // current_state
```
Coding Style 3 – Combine CS+NL; Separate OL

- CS+NL

```verilog
always @(posedge clk)
begin
   case (state)
      S0: case (In)
         In0: state <= S1;
         In1: state <= S0;
         . . .
         endcase //In
      S1: . . .
      endcase //state
   end
```

- OL

```verilog
// if Moore
always @(state) 
   z <= output_value;

// if Mealy
always @(state or In) 
   z <= output_value;
```
Behavioral Models of FSM

Example 1

- low speed: brake=0, accelerator=1
- medium speed: brake=1
- high speed: brake=0, accelerator=1
- stopped: brake=1

Brake (brake) and accelerator (accelerator) are influenced by speed (speed) and clock.
module speed_machine ( clock, accelerator, brake, speed );
input clock, accelerator, brake;
output [1:0] speed;
reg [1:0] current_state, next_state;
// state encoding
parameter stopped = 2`b00;
parameter s_slow = 2`b01;
parameter s_medium = 2`b10;
parameter s_high = 2`b11;
// NL, Next-state Logic
always@( state or accelerator or brake )
if ( brake == 1`b1 )
  case ( current_state )
    stopped: next_state <= stopped;
    s_low: next_state <= s_low;
    s_medium: next_state <= s_medium;
    s_high: next_state <= s_high;
    default: next_state <= stopped;
  endcase
else if ( accelerator == 1`b1 )
  case ( current_state )
    stopped: next_state <= s_low;
    s_low: next_state <= s_medium;
    s_medium: next_state <= s_high;
    s_high: next_state <= s_high;
    default: next_state <= stopped;
  endcase
else
  next_state <= current_state;
endcase
// CS
always @ ( posedge clock )
current_state <= next_state;
// OL, Output Logic
assign speed = current_state;
module speed_machine2 ( clock, accelerator, brake, speed );
input clock, accelerator, brake;
output [1:0] speed;
reg [1:0] speed;

`define stopped 2`b00
`define s_low 2`b01
`define s_medium 2`b10
`define s_high 2`b11

// OL is empty
// because speed itself is state

// NL + CS
always @ ( posedge clock )
if ( brake == 1`b1 )
case ( speed )
    `stopped: speed <= `stopped;
    `s_low:   speed <= `stopped;
    `s_medium: speed <= `s_low;
    `s_high:  speed <= `s_medium;
    default:  speed <= `stopped;
endcase
else if ( accelerator == 1`b1 )
case ( speed )
    `stopped: speed <= `s_low;
    `s_low:   speed <= `s_medium;
    `s_medium: speed <= `s_high;
    `s_high:  speed <= `s_high;
    default:  speed <= `stopped;
endcase
endmodule
## State encoding

<table>
<thead>
<tr>
<th>NO.</th>
<th>Sequential</th>
<th>Gray</th>
<th>Johnson</th>
<th>One-Hot</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0000</td>
<td>00000000</td>
<td>0000000000000001</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0001</td>
<td>00000001</td>
<td>0000000000000010</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0011</td>
<td>00000011</td>
<td>0000000000000100</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>0010</td>
<td>00000111</td>
<td>0000000000010000</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>0110</td>
<td>00011111</td>
<td>0000000100000000</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>0111</td>
<td>00011111</td>
<td>0000000100000000</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>0101</td>
<td>00111111</td>
<td>0000000100000000</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>0100</td>
<td>01111111</td>
<td>0000000100000000</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1100</td>
<td>11111111</td>
<td>0000000100000000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>1101</td>
<td>11111101</td>
<td>0000001000000000</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>1111</td>
<td>11111100</td>
<td>0000010000000000</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>1110</td>
<td>11111000</td>
<td>0001000000000000</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>1010</td>
<td>11110000</td>
<td>0001000000000000</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>1011</td>
<td>11100000</td>
<td>0010000000000000</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>1001</td>
<td>11000000</td>
<td>0100000000000000</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>1000</td>
<td>10000000</td>
<td>1000000000000000</td>
</tr>
</tbody>
</table>
Conclusion

- FSM Design
  - Partition FSM and non-FSM logic
  - Partition combinational part and sequential part
  - Use parameter to define names of the state vector
  - Assign a default (reset) state