Finite State Machine
Outline

1. Overview
2. FSM representation
3. Timing and performance of an FSM
4. Moore machine versus Mealy machine
5. VHDL description of FSMs
6. State assignment
7. Moore output buffering
8. FSM design examples
1. Overview on FSM

- Contain “random” logic in next-state logic
- Used mainly used as a controller in a large system
- Mealy vs Moore output
2. Representation of FSM

• State diagram

Figure 10.2 Notation for a state.
- E.g. a memory controller
• ASM (algorithmic state machine) chart
  – Flowchart-like diagram
  – Provide the same info as an FSM
  – More descriptive, better for complex description

– ASM block
  • One state box
  • One or more optional decision boxes: with T or F exit path
  • One or more conditional output boxes: for Mealy output
Figure 10.4  ASM block.
State diagram and ASM chart conversion

- E.g. 1.
E.g.?
• E.g. 3.
• E.g. 4.
• E.g. 6.
• Difference between a regular flowchart and ASM chart:
  – Transition governed by clock
  – Transition done between ASM blocks

• Basic rules:
  – For a given input combination, there is one unique exit path from the current ASM block
  – The exit path of an ASM block must always lead to a state box. The state box can be the state box of the current ASM block or a state box of another ASM block.
• Incorrect ASM charts:
3. Performance of FSM

- Similar to regular sequential circuit

\[ T_c = T_{cq} + T_{next(max)} + T_{setup} \]

\[ T_{co(mo)} = T_{cq} + T_{output(mo)} \]
• Sample timing diagram
RTL Hardware Design
by P. Chu
4. Moore vs Mealy output

- Moore machine:
  - output is a function of state
- Mealy machine:
  - output function of state and output
- From theoretical point of view
  - Both machines have similar “computation capability”
- Implication of FSM as a controller?
• E.g., edge detection circuit
  – A circuit to detect the rising edge of a slow “strobe” input and generate a “short” (about 1-clock period) output pulse.
• Three designs:
• Comparison
  – Mealy machine uses fewer states
  – Mealy machine responds faster
  – Mealy machine may be transparent to glitches
• Which one is better?
• Types of control signal
  – Edge sensitive
    • E.g., enable signal of counter
    • Both can be used but Mealy is faster
  – Level sensitive
    • E.g., write enable signal of SRAM
    • Moore is preferred
VHDL Description of FSM

• Follow the basic block diagram
• Code the next-state/output logic according to the state diagram/ASM chart
• Use enumerate data type for states
• E.g. 6.
library ieee;
use ieee.std_logic_1164.all;
entity mem_ctrl is
port ( 
  clk, reset: in std_logic;
  mem, rw, burst: in std_logic;
  oe, we, we_me: out std_logic);
end mem_ctrl;

architecture mult_seg_arch of mem_ctrl is
  type mc_state_type is
    (idle, read1, read2, read3, read4, write);
signal state_reg, state_next: mc_state_type;
begin
  -- state register
  process(clk, reset)
  begin
    if (reset='1') then
      state_reg <= idle;
    elsif (clk'event and clk='1') then
      state_reg <= state_next;
    end if;
  end process;

-- next-state logic
process(state_reg, mem, rw, burst) begin
  case state_reg is
    when idle =>
      if mem='1' then
        if rw='1' then
          state_next <= read1;
        else
          state_next <= write;
        end if;
      else
        state_next <= idle;
      end if;
    when write =>
      state_next <= idle;
  end case;
end process;
when read1 =>
    if (burst='1') then
        state_next <= read2;
    else
        state_next <= idle;
    end if;
when read2 =>
    state_next <= read3;
when read3 =>
    state_next <= read4;
when read4 =>
    state_next <= idle;
end case;
end process;
-- moore output logic
process(state_reg)
begin
  we <= '0';  -- default value
  oe <= '0';  -- default value
  case state_reg is
    when idle =>
    when write =>
      we <= '1';
    when read1 =>
      oe <= '1';
    when read2 =>
      oe <= '1';
    when read3 =>
      oe <= '1';
    when read4 =>
      oe <= '1';
  end case;
end process;
— *mealy output logic*

```vhdl
process (state_reg, mem, rw)
begin
  we_me <= '0'; -- default value
  case state_reg is
    when idle =>
      if (mem='1') and (rw='0') then
        we_me <= '1';
      end if;
    when write =>
    when read1 =>
    when read2 =>
    when read3 =>
    when read4 =>
      end case;
  end process;
end mult_seg_arch;
```

```
we_me <= '1' when ((state_reg=idle) and (mem='1') and (rw='0')) else '0';
```
• Combine next-state/output logic together
process (state_reg, mem, rw, burst)
begin
  oe <= '0';    — default values
  we <= '0';
  we_me <= '0';
  case state_reg is
    when idle =>
      if mem='1' then
        if rw='1' then
          state_next <= read1;
        else
          state_next <= write;
        end if;
      else
        state_next <= idle;
      end if;
    when write =>
      state_next <= idle;
      we <= '1';
  end case;
end begin;
when read1 =>
  if (burst='1') then
    state_next <= read2;
  else
    state_next <= idle;
  end if;
  oe <= '1';
when read2 =>
  state_next <= read3;
  oe <= '1';
when read3 =>
  state_next <= read4;
  oe <= '1';
when read4 =>
  state_next <= idle;
  oe <= '1';
end case;
end process;
6. State assignment

• State assignment: assign binary representations to symbolic states

• In a synchronous FSM
  – All assignments work
  – Good assignment reduce the complexity of next-state/output logic

• Typical assignment
  – Binary, Gray, one-hot, almost one-hot
<table>
<thead>
<tr>
<th>State</th>
<th>Binary assignment</th>
<th>Gray code assignment</th>
<th>One-hot assignment</th>
<th>Almost one-hot assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>000</td>
<td>000</td>
<td>000001</td>
<td>00000</td>
</tr>
<tr>
<td>read1</td>
<td>001</td>
<td>001</td>
<td>000010</td>
<td>00001</td>
</tr>
<tr>
<td>read2</td>
<td>010</td>
<td>011</td>
<td>000100</td>
<td>00010</td>
</tr>
<tr>
<td>read3</td>
<td>011</td>
<td>010</td>
<td>001000</td>
<td>00100</td>
</tr>
<tr>
<td>read4</td>
<td>100</td>
<td>110</td>
<td>010000</td>
<td>01000</td>
</tr>
<tr>
<td>write</td>
<td>101</td>
<td>111</td>
<td>100000</td>
<td>10000</td>
</tr>
</tbody>
</table>
State assignment in VHDL

• Implicit: use user attributes enum_encoding

```vhdl
type mc_state_type is (idle, write, read1, read2, read3, read4);
attribute enum_encoding: string;
attribute enum_encoding of mc_state_type:
    type is "0000 0100 1000 1001 1010 1011";
```

• Explicit: use std_logic_vector for the register
architecture state_assign_arch of mem_fsm is
  constant idle: std_logic_vector(3 downto 0) := "0000";
  constant write: std_logic_vector(3 downto 0) := "0100";
  constant read1: std_logic_vector(3 downto 0) := "1000";
  constant read2: std_logic_vector(3 downto 0) := "1001";
  constant read3: std_logic_vector(3 downto 0) := "1010";
  constant read4: std_logic_vector(3 downto 0) := "1011"
  signal state_reg, state_next: std_logic_vector(3 downto 0);
begin
  -- state register
  process(clk, reset)
  begin
    if (reset='1') then
      state_reg <= idle;
    elsif (clk'event and clk='1') then
      state_reg <= state_next;
    end if;
  end process;
-- next-state logic
begin
  we_me <= '0';
  case state_reg is
    when idle =>
      if mem='1' then
        if rw='1' then
          state_next <= read1;
        else
          state_next <= write;
        end if;
      else
        state_next <= idle;
      end if;
    when others =>
      state_next <= idle;
  end case;
end process;
Handling the unused state

- Many binary representations are not used
- What happens if the FSM enters an unused state?
  - Ignore the condition
  - Safe (Fault-tolerant) FSM: got to an error state or return to the initial state.
- Easy for the explicit state assignment
- No portable code for the enumerated data type
6. Moore output buffering

• FSM as control circuit
  – Sometimes fast, glitch-free signal is needed
  – An extra output buffer can be added, but introduce one-clock delay

• Special schemes can be used for Moore output
  – Clever state assignment
  – Look-ahead output circuit
• Potential problems of the Moore output logic:
  – Potential hazards introduce glitches
  – Increase the Tco delay \((Tco = Tcq + Toutput)\)
• Can we get control signals directly from the register?
Clever state assignment

• Assigning state according to output signal patterns
• Output can be obtained from register directly
• Extra register bits may be needed
• Must use explicit state assignment in VHDL code to access individual register bit
• Difficult to revise and maintain
Table 10.2  Clever assignment

<table>
<thead>
<tr>
<th></th>
<th>$Q_3$</th>
<th>$Q_2$</th>
<th>$Q_1$</th>
<th>$Q_0$</th>
<th>$Q_3$ $Q_2$ $Q_1$ $Q_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>read1</td>
<td>10</td>
<td>00</td>
<td>00</td>
<td>1000</td>
<td>0000</td>
</tr>
<tr>
<td>read2</td>
<td>10</td>
<td>01</td>
<td>01</td>
<td>1001</td>
<td>0000</td>
</tr>
<tr>
<td>read3</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>1010</td>
<td>0000</td>
</tr>
<tr>
<td>read4</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>1011</td>
<td>0000</td>
</tr>
<tr>
<td>write</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>0100</td>
<td>0000</td>
</tr>
</tbody>
</table>
• VHDL code

```vhdl
architecture state_assign_arch of mem_fsm is
  constant idle: std_logic_vector(3 downto 0) := "0000";
  constant write: std_logic_vector(3 downto 0) := "0100";
  constant read1: std_logic_vector(3 downto 0) := "1000";
  constant read2: std_logic_vector(3 downto 0) := "1001";
  constant read3: std_logic_vector(3 downto 0) := "1010";
  constant read4: std_logic_vector(3 downto 0) := "1011"
  signal state_reg, state_next: std_logic_vector(3 downto 0);

  -- Moore output logic
  oe <= state_reg(3);
  we <= state_reg(2);
```

Look-ahead output circuit

• Output buffer introduces one-clock delay
• The “next” value of Moore output can be obtained by using state_next signal
• Buffer the next value cancel out the one-clock delay
• More systematic and easier to revise and maintain
(a) Moore output with a regular output buffer

(b) Moore output with a look-ahead buffer
• Modification over original VHDL code:
  – Add output buffer
  – Use state_next to replace state_reg in Moore output logic

```vhdl
-- output buffer
process (clk, reset)
begins
  if (reset='1') then
    oe_buf_reg <= '0';
    we_buf_reg <= '0';
  elsif (clk'event and clk='1') then
    oe_buf_reg <= oe_next;
    we_buf_reg <= we_next;
  end if;
end process;
```
-- moore output logic

process(state_next)
begin
    we_next <= '0';  -- default value
    oe_next <= '0';  -- default value
    case state_next is
        when idle =>
        when write =>
            we_next <= '1';
        when read1 =>
            oe_next <= '1';
        when read2 =>
            oe_next <= '1';
        when read3 =>
            oe_next <= '1';
        when read4 =>
            oe_next <= '1';
    end case;
end process;
7. FSM design examples

- Edge detector circuit
- Arbitrator (read)
- DRAM strobe signal generation
- Manchester encoding/decoding (read)
- FSM base binary counter
Edge detecting circuit (Moore)

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity edge_detector1 is
port(
  clk, reset: in std_logic;
  strobe: in std_logic;
  p1: out std_logic);
end edge_detector1;

architecture moore_arch of edge_detector1 is
  type state_type is (zero, edge, one);
  signal state_reg, state_next: state_type;
begin
  -- state register
  process(clk, reset)
  begin
    if (reset='1') then
      state_reg <= zero;
    elsif (clk'event and clk='1') then
      state_reg <= state_next;
    end if;
  end process;
end
```
-- next-state logic
process(state_reg, strobe)
begin
  case state_reg is
    when zero =>
      if strobe = '1' then
        state_next <= edge;
      else
        state_next <= zero;
      end if;
    when edge =>
      if strobe = '1' then
        state_next <= one;
      else
        state_next <= zero;
      end if;
    when one =>
      if strobe = '1' then
        state_next <= one;
      else
        state_next <= zero;
      end if;
  end case;
end process;

-- moore output logic
p1 <= '1' when state_reg = edge else '0';
end moore_arch;
Use clever state assignment

<table>
<thead>
<tr>
<th>state_reg(1)</th>
<th>state_reg(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p1)</td>
<td></td>
</tr>
<tr>
<td>zero</td>
<td>0</td>
</tr>
<tr>
<td>edge</td>
<td>1</td>
</tr>
<tr>
<td>one</td>
<td>0</td>
</tr>
</tbody>
</table>

```vhdl
architecture clever_assign_buf_arch of edge_detector1 is
constant zero: std_logic_vector(1 downto 0):= "00";
constant edge: std_logic_vector(1 downto 0):= "10";
constant one: std_logic_vector(1 downto 0) := "01";
signal state_reg, state_next: std_logic_vector(1 downto 0);

-- moore output logic
p1 <= state_reg(1);
```
Use look-ahead output

```vhdl
process(clk, reset)
begin
  if (reset='1') then
    p1_reg <= '0';
  elsif (clk'event and clk='1') then
    p1_reg <= p1_next;
  end if;
end process;

— next output logic
p1_next <= '1' when state_next=edge else '0';
p1 <= p1_reg;
```
Edge detecting circuit (Mealy)

```
architecture mealy_arch of edge_detector2 is
  type state_type is (zero, one);
  signal state_reg, state_next: state_type;
begin
  -- state register
  process(clk, reset)
  begin
    if (reset='1') then
      state_reg <= zero;
    elsif (clk’event and clk='1') then
      state_reg <= state_next;
    end if;
```

— next-state logic

process (state_reg, strobe)
begin
  case state_reg is
    when zero =>
      if strobe = '1' then
        state_next <= one;
      else
        state_next <= zero;
      end if;
    when one =>
      if strobe = '1' then
        state_next <= one;
      else
        state_next <= zero;
      end if;
    end case;
  end process;

— mealy output logic

p2 <= '1' when (state_reg=zero) and (strobe='1') else '0';
• Edge detecting circuit (direct implementation):
  – edge occurs when previous value is 0 and new value is 1
  – Same as Mealy design with state assignment: zero => 0, one => 1

Figure 10.19  Direct implementation of an edge detector.
architecture direct_arch of edge_detector2 is
signal delay_reg: std_logic;
begin
  -- delay register
  process(clk, reset)
  begin
    if (reset='1') then
      delay_reg <= '0';
    elsif (clk'event and clk='1') then
      delay_reg <= strobe;
    end if;
  end process;
  -- decoding logic
  p2 <= (not delay_reg) and strobe;
end direct_arch;
• DRAM strobe signal generation
  – E.g., 120ns DRAM (Trc=120ns): 
    Tras=85ns, Tcas=20ns, Tpr=35ns
• 3 intervals has to be at least 65ns, 20 ns, and 35 ns
• A slow design: use a 65ns clock period
  – 195 ns (3*65ns) read cycle
• The control signal is level-sensitive
library ieee;
use ieee.std_logic_1164.all;
entity dram_strobe is
  port(
    clk, reset: in std_logic;
    mem: in std_logic;
    cas_n, ras_n: out std_logic
  );
end dram_strobe;
architecture fsm_slow_clk_arch of dram_strobe is
  type fsm_state_type is (idle,r,c,p);
  signal state_reg, state_next: fsm_state_type;
begn
  -- state register
  process(clk, reset)
  begin
    if (reset='1') then
      state_reg <= idle;
    elsif (clk'event and clk='1') then
      state_reg <= state_next;
    end if;
  end process;
• Should revise the code to obtain glitch-free output
• A faster design: use a 20ns clock period
  – 140 ns (7*20ns) read cycle
• FSM-based binary counter:
  – Free-running mod-16 counter
• 4-bit binary counter with features:
  – Synchronous clear, load, enable