State machine

GUIDELINES:
- process for the sequential logic
- process for the combinational logic
- use enumerated data type to list all possible system states

Moore style state machine

There may be glitches on the outputs
Mealy style state machine

There may be glitches + outputs may last less than one cycle.

STORED OUTPUT (a.k.a. pipelined output) Mealy and Moore

Moore

Moore

Mealy

Mealy

OUTPUT CODED MOORE

No glitches on the output + very fast.
ENCODING STYLES FOR THE STATES

Most commonly used:

- BINARY ← most common in ASICs
- ONE HOT ← most common in FPGAs

Use one FF per state with n Flip-Flops we can have up to $2^n$ states

Table 8.1
State encoding of an 8-state FSM.

<table>
<thead>
<tr>
<th>STATE</th>
<th>Encoding Style</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BINARY</td>
</tr>
<tr>
<td>state0</td>
<td>000</td>
</tr>
<tr>
<td>state1</td>
<td>001</td>
</tr>
<tr>
<td>state2</td>
<td>010</td>
</tr>
<tr>
<td>state3</td>
<td>011</td>
</tr>
<tr>
<td>state4</td>
<td>100</td>
</tr>
<tr>
<td>state5</td>
<td>101</td>
</tr>
<tr>
<td>state6</td>
<td>110</td>
</tr>
<tr>
<td>state7</td>
<td>111</td>
</tr>
</tbody>
</table>
Exercise #1

Design a state machine to implement an edge detector.

CLK

DIN

SYSTEM

PULSE

CLK

DIN

PULSE

Q1

Q2

PULSE

DIN

CLK

Q1

Q2

DOUT

INTUITIVE SOLUTION
library ieee;
use ieee.std_logic_1164.all;

entity edge_detector is
port ( din : in std_logic;
      clk : in std_logic;
      rst_n : in std_logic;
      dout : out std_logic
    );
end edge_detector;

architecture rtl of edge_detector is

type state_t is (zero, posedge, negedge, one);
signal state, next_state : state_t;
signal next_pulse, pulse : std_logic;

begin

  the_machine: process(din,state)
  begin

    -- defaults
    next_state <= zero;
    next_pulse <= '0';

    case state is
      when zero => -- stable zero
        if (din = '0') then
          next_state <= zero;
          next_pulse <= '0';
        else
          next_state <= posedge;
          next_pulse <= '1';
        end if;
      when posedge => -- positive edge
        if (din = '0') then
          next_state <= negedge;
          next_pulse <= '1';
        else
          next_state <= one;
          next_pulse <= '0';
        end if;
      when negedge => -- negative edge
        if (din = '0') then
          next_state <= zero;
          next_pulse <= '0';
        else
          next_state <= posedge;
          next_pulse <= '1';
        end if;
      when one => -- stable one
        if (din = '0') then
          next_state <= negedge;
          next_pulse <= '1';
        else
          next_state <= one;
          next_pulse <= '0';
        end if;
      when others =>
        -- do nothing
    end case;

  end process the_machine;
the_registers: process(clk, rst_n)
begin
  if (rst_n = '0') then
    state <= zero;
    pulse <= '0';
  elsif (clk='1' and clk'event) then
    state <= next_state;
    pulse <= nextpulse;
  end if;
end process the_registers;

--dummy assignment
dout <= pulse;
end rtl;
library ieee;
use ieee.std_logic_1164.all;

entity edge_detector is
port (din : in std_logic;
      clk : in std_logic;
      rst_n : in std_logic;
      dout : out std_logic
);
end edge_detector;

architecture rtl of edge_detector is
  type state_t is (zero, one);
  signal state, next_state : state_t;
  signal pulse : std_logic;
begin
  the_machine: process(din,state)
  begin
    -- defaults
    next_state <= zero;
    pulse <= '0';
    case state is
      when zero =>
        if (din = '0') then
          next_state <= zero;
        else
          next_state <= one;
          pulse <= '1';
        end if;
      when one =>
        if (din = '0') then
          next_state <= zero;
          pulse <= '1';
        else
          next_state <= one;
        end if;
      when others =>
        -- do nothing
    end case;
  end process the_machine;

  the_registers: process(clk, rst_n)
  begin
    if (rst_n = '0') then
      state <= zero;
    elsif (clk='1' and clk'event) then
      state <= next_state;
    end if;
  end process the_registers;

  -- dummy assignment
  dout <= pulse;
end rtl;
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;

entity tb_edge_detector is
end tb_edge_detector;

architecture beh of tb_edge_detector is

component edge_detector
port(
    din : in std_logic;
    clk : in std_logic;
    rst_n : in std_logic;
    dout : out std_logic
);
end component edge_detector;

--signal declaration

signal clk : std_logic;
signal rst_n : std_logic;
signal din : std_logic;
signal dout : std_logic;

begin
    inst_1: edge_detector
    port map(
        din => din,
        clk => clk,
        rst_n => rst_n,
        dout => dout
    );

    clk_p : process
    begin
        clk <= '0';
        wait for 2 ns;
        clk <= '1';
        wait for 2 ns;
    end process clk_p;

    input_data : process
    begin
        din <= '0';
        wait for 7 ns;
        din <= '1';
        wait for 10 ns;
        din <= '0';
        wait for 20 ns;
    end process input_data;

    test_bench : process
    begin
        rst_n <= '0';
        wait for 1 ns;
        rst_n <= '1';
        wait for 100 ns;
        assert false
        report "End of Simulation"
        severity failure;
    end process test_bench;
end beh;