FSM.Moore_Vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fsm_moore is
  Port (
    reset  : in  STD_LOGIC;
    input  : in  STD_LOGIC;
    output : out STD_LOGIC;
    clk    : in  STD_LOGIC
  );
end fsm_moore;

architecture Behavioral of fsm_moore is
  type moore_state is (A0, A1, B0, B1, C, D0, D1);
  signal present_state, next_state : moore_state;
begin

  -- state register
  process(clk, reset)
  begin
    if reset = '1' then
      present_state <= A0;  -- choose initial Moore state (A0 gives output='1' on reset)
    elsif rising_edge(clk) then
      present_state <= next_state;
    end if;
  end process;

  -- next-state logic
  process(present_state, input)
  begin
```

```
case present_state is
 when A0 =>
  -- A0 corresponds to Mealy(A,0) -> next_state = C (independent of input)
  next_state <= C;

 when A1 =>
  -- A1 corresponds to Mealy(A,1) -> Mealy next = B; choose B0/B1 based on current input
  if input = '0' then
   next_state <= B0;
  else
   next_state <= B1;
  end if;

 when B0 =>
  if input = '0' then
   next_state <= D0;
  else
   next_state <= D1;
  end if;

 when B1 =>
  if input = '0' then
   next_state <= B0;
  else
   next_state <= B1;
  end if;

 when C =>
  if input = '0' then
   next_state <= D0;
  else
```

```
        next_state <= C;
      end if;

    when D0 =>
      if input = '0' then
        next_state <= A0;
      else
        next_state <= A1;
      end if;

    when D1 =>
      if input = '0' then
        next_state <= D0;
      else
        next_state <= D1;
      end if;

    when others =>
      next_state <= A0;
  end case;
end process;

-- Moore outputs (depend only on present_state)
process(present_state)
begin
  case present_state is
    when A0 => output <= '1';
    when A1 => output <= '0';
    when B0 => output <= '0';
    when B1 => output <= '1';
    when C  => output <= '1';
```

```
      when D0 => output <= '1';
      when D1 => output <= '0';
      when others => output <= '0';
    end case;
  end process;

end Behavioral;
```

FSM.Moore_tb

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fsm_moore is
  Port (
    reset  : in  STD_LOGIC;
    input  : in  STD_LOGIC;
    output : out STD_LOGIC;
    clk    : in  STD_LOGIC
  );
end fsm_moore;

architecture Behavioral of fsm_moore is
  type moore_state is (A0, A1, B0, B1, C, D0, D1);
  signal present_state, next_state : moore_state;
begin

  -- state register
  process(clk, reset)
  begin
    if reset = '1' then
      present_state <= A0;  -- choose initial Moore state (A0 gives output='1' on reset)
  elsif rising_edge(clk) then
    present_state <= next_state;
  end if;
end process;

-- next-state logic
process(present_state, input)
begin
  case present_state is
    when A0 =>
      -- A0 corresponds to Mealy(A,0) -> next_state = C (independent of input)
      next_state <= C;

    when A1 =>
      -- A1 corresponds to Mealy(A,1) -> Mealy next = B; choose B0/B1 based on current input
      if input = '0' then
        next_state <= B0;
      else
        next_state <= B1;
      end if;

    when B0 =>
      if input = '0' then
        next_state <= D0;
      else
        next_state <= D1;
      end if;

    when B1 =>
      if input = '0' then
        next_state <= B0;
      else
        next_state <= B1;
      end if;

    when C =>
      if input = '0' then
        next_state <= D0;
      else
        next_state <= C;
      end if;

    when D0 =>
      if input = '0' then
        next_state <= A0;
      else
        next_state <= A1;
      end if;

    when D1 =>
      if input = '0' then
        next_state <= D0;
      else
        next_state <= D1;
      end if;

    when others =>
      next_state <= A0;
  end case;
end process;

-- Moore outputs (depend only on present_state)
```

```
  process(present_state)
  begin
    case present_state is
      when A0 => output <= '1';
      when A1 => output <= '0';
      when B0 => output <= '0';
      when B1 => output <= '1';
      when C  => output <= '1';
      when D0 => output <= '1';
      when D1 => output <= '0';
      when others => output <= '0';
    end case;
  end process;

end Behavioral;
```