```
--============================
-- Problem 12.4 b
--============================
library ieee;
use ieee.std_logic_1164.all;
entity sram_ctrl is
   port(
      clk, reset: in std_logic;
      mem: in std_logic;
      rw: in std_logic;
      addr: in std_logic_vector(19 downto 0);
      data_m2s: in std_logic;
      we, oe: out std_logic;
      ready: out std_logic;
      data_s2m: out std_logic;
      d: inout std_logic;
      ad: out std_logic_vector(19 downto 0)
  );
end sram_ctrl;

architecture arch of sram_ctrl is
   type state_type is
      (idle, r1, r2, r3, r4, r5, r6, r7, r8, w1, w2, w3, w4, w5, w6, w7);
   signal state_reg, state_next: state_type;
   signal data_m2s_reg, data_m2s_next: std_logic;
   signal data_s2m_reg, data_s2m_next: std_logic;
   signal addr_reg, addr_next: std_logic_vector(19 downto 0);
   signal tri_en_buf, we_buf, oe_buf: std_logic;
   signal tri_en_reg, we_reg, oe_reg: std_logic;
begin
   -- state & data registers
   process(clk,reset)
   begin
      if (reset='1') then
         state_reg <= idle;
         addr_reg <= (others=>'0');
         data_m2s_reg <= '0';
         data_s2m_reg <= '0';
         tri_en_reg <= '0';
         we_reg <= '1';
         oe_reg <='1';
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
         addr_reg <= addr_next;
         data_m2s_reg <= data_m2s_next;
         data_s2m_reg <= data_s2m_next;
         tri_en_reg <= tri_en_buf;
         we_reg <= we_buf;
         oe_reg <= oe_buf;
      end if;
   end process;
   -- next-state logic & data path functional units/routing
   process(state_reg,mem,rw,d,addr,data_m2s,
           data_m2s_reg,data_s2m_reg,addr_reg)
```

```
begin
   addr_next <= addr_reg;
   data_m2s_next <= data_m2s_reg;
   data_s2m_next <= data_s2m_reg;
   ready <= '0';
   case state_reg is
      when idle =>
         if mem='0' then
            state_next <= idle;
         else
            if rw='0' then --write
               state_next <= w1;
               addr_next <= addr;
               data_m2s_next <= data_m2s;
            else -- read
               state_next <= r1;
               addr_next <= addr;
            end if;
         end if;
         ready <= '1';
      when w1 =>
         state_next <= w2;
      when w2 =>
         state_next <= w3;
      when w3 =>
         state_next <= w4;
         when w4 =>
         state_next <= w5;
         when w5 =>
         state_next <= w6;
         when w6 =>
         state_next <= w7;
      when w7 =>
         state_next <= idle;
      when r1 =>
         state_next <= r2;
      when r2 =>
         state_next <= r3;
         when r3 =>
         state_next <= r4;
         when r4 =>
         state_next <= r5;
         when r5 =>
         state_next <= r6;
         when r6 =>
         state_next <= r7;
         when r7 =>
         state_next <= r8;
      when r8 =>
         state_next <= idle;
         data_s2m_next <= d;
   end case;
end process;
-- look-ahead output logic
```

```
   process(state_next)
   begin
      tri_en_buf <='0';
      oe_buf <= '1';
      we_buf<= '1';
      case state_next is
         when idle =>
         when w1 =>
         when w2 =>
            we_buf <= '0';
            tri_en_buf <= '1';
         when w3 =>
            we_buf <= '0';
            tri_en_buf <= '1';
            when w4 =>
            we_buf <= '0';
            tri_en_buf <= '1';
            when w5 =>
            we_buf <= '0';
            tri_en_buf <= '1';
            when w6 =>
            we_buf <= '0';
            tri_en_buf <= '1';
            when w7 =>
            tri_en_buf <= '1';
         when r1 =>
            oe_buf <= '0';
         when r2 =>
            oe_buf <= '0';
         when r3 =>
            oe_buf <= '0';
            when r4 =>
            oe_buf <= '0';
         when r5 =>
            oe_buf <= '0';
         when r6 =>
            oe_buf <= '0';
            when r7 =>
            oe_buf <= '0';
         when r8 =>
            oe_buf <= '0';
      end case;
   end process;
   --  output
   we <= we_reg;
   oe <= oe_reg;
   ad <= addr_reg;
   d <= data_m2s_reg when tri_en_reg ='1' else 'Z';
   data_s2m <= data_s2m_reg;
end arch;

--==============================
-- Problem 12.5 b
--==============================
```

```
library ieee;
use ieee.std_logic_1164.all;
entity sram_ctrl is
   port(
      clk, reset: in std_logic;
      mem: in std_logic;
      rw: in std_logic;
      addr: in std_logic_vector(19 downto 0);
      data_m2s: in std_logic;
      we, oe: out std_logic;
      ready: out std_logic;
      data_s2m: out std_logic;
      d: inout std_logic;
      ad: out std_logic_vector(19 downto 0)
  );
end sram_ctrl;

architecture arch of sram_ctrl is
   type state_type is
      (idle, r1, r2, r3, w1, w2, w3, w4);
   signal state_reg, state_next: state_type;
   signal data_m2s_reg, data_m2s_next: std_logic;
   signal data_s2m_reg, data_s2m_next: std_logic;
   signal addr_reg, addr_next: std_logic_vector(19 downto 0);
   signal tri_en_buf, we_buf, oe_buf: std_logic;
   signal tri_en_reg, we_reg, oe_reg: std_logic;
begin
   -- state & data registers
   process(clk,reset)
   begin
      if (reset='1') then
         state_reg <= idle;
         addr_reg <= (others=>'0');
         data_m2s_reg <= '0';
         data_s2m_reg <= '0';
         tri_en_reg <= '0';
         we_reg <= '1';
         oe_reg <='1';
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
         addr_reg <= addr_next;
         data_m2s_reg <= data_m2s_next;
         data_s2m_reg <= data_s2m_next;
         tri_en_reg <= tri_en_buf;
         we_reg <= we_buf;
         oe_reg <= oe_buf;
      end if;
   end process;
   -- next-state logic & data path functional units/routing
   process(state_reg,mem,rw,d,addr,data_m2s,
           data_m2s_reg,data_s2m_reg,addr_reg)
   begin
      addr_next <= addr_reg;
      data_m2s_next <= data_m2s_reg;
```

```
      data_s2m_next <= data_s2m_reg;
      ready <= '0';
      case state_reg is
         when idle =>
            if mem='0' then
               state_next <= idle;
            else
               if rw='0' then --write
                  state_next <= w1;
                  addr_next <= addr;
                  data_m2s_next <= data_m2s;
               else -- read
                  state_next <= r1;
                  addr_next <= addr;
               end if;
            end if;
            ready <= '1';
         when w1 =>
            state_next <= w2;
         when w2 =>
            state_next <= w3;
         when w3 =>
            state_next <= w4;
         when w4 =>
            state_next <= idle;
         when r1 =>
            state_next <= r2;
         when r2 =>
            state_next <= r3;
         when r3 =>
            state_next <= idle;
            data_s2m_next <= d;
      end case;
   end process;
   -- look-ahead output logic
   process(state_next)
   begin
      tri_en_buf <='0';
      oe_buf <= '1';
      we_buf<= '1';
      case state_next is
         when idle =>
         when w1 =>
         when w2 =>
            we_buf <= '0';
            tri_en_buf <= '1';
         when w3 =>
            we_buf <= '0';
            tri_en_buf <= '1';
         when w4 =>
            tri_en_buf <= '1';
         when r1 =>
            oe_buf <= '0';
         when r2 =>
```

```
            oe_buf <= '0';
         when r3 =>
            oe_buf <= '0';
      end case;
   end process;
   --  output
   we <= we_reg;
   oe <= oe_reg;
   ad <= addr_reg;
   d <= data_m2s_reg when tri_en_reg ='1' else 'Z';
   data_s2m <= data_s2m_reg;
end arch;

--============================
-- Problem 12.1
--============================
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity pulse_5clk is
   port(
      clk, reset: in std_logic;
      go, stop: in std_logic;
      pulse: out std_logic
  );
end pulse_5clk;

architecture prog_arch of pulse_5clk is
    type fsmd_state_type is (idle, delay, sh);
    signal state_reg, state_next: fsmd_state_type;
    signal c_reg, c_next: unsigned(2 downto 0);
    signal w_reg, w_next: unsigned(2 downto 0);
begin
    --State and data registers
    process(clk, reset)
    begin
        if(reset='1') then
            state_reg <= idle;
            c_reg <= (others=>'0');
            w_reg <= "101"; -- default 5 cycle delay
        elsif (clk'event and clk='1') then
            state_reg <= state_next;
            c_reg <= c_next;
            w_reg <= w_next;
        end if;
    end process;
    -- next state logic & data path functional units/routing
    process(state_reg,go,stop,c_reg,w_reg)
    begin
        pulse <= '0';
        c_next <= c_reg;
        w_next <= w_reg;
        case state_reg is
```

```
            when idle =>
                if go='1' then
                    if stop='1' then
                        state_next <= sh;
                    else
                        state_next <= delay;
                    end if;
                else
                    state_next <= idle;
                end if;
                c_next <= (others => '0');
            when delay =>
                if stop='1' then
                    state_next <= idle;
                else
                    if (c_reg=w_reg-1) then
                        state_next <= idle;
                    else
                        c_next <= c_reg+1;
                        state_next <= delay;
                    end if;
                end if;
                pulse <= '1';
            when sh =>
                w_next <= go & w_reg(2 downto 1);
                if(c_reg = "011") then
                    state_next <= idle;
                else
                    state_next <= sh;
                    c_next <= c_reg+1;
                end if;
        end case;
    end process;
end prog_arch;


--============================
-- Problem 9.15
--============================
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity xor_red is
   port(
      clk, reset: in std_logic;
        a : in std_logic_vector(7 downto 0);
      y: out std_logic
  );
end xor_red;

architecture arch of xor_red is
    signal reg1, reg1_next: std_logic_vector(5 downto 0);
    signal reg2, reg2_next: std_logic_vector(3 downto 0);
```

```
    signal reg3, reg3_next: std_logic_vector(1 downto 0);
    signal reg4, reg4_next: std_logic;
begin
    --State and data registers
    process(clk, reset)
    begin
        if(reset='1') then
            reg1 <= (others=>'0');
            reg2 <= (others=>'0');
            reg3 <= (others=>'0');
            reg4 <= '0';
        elsif (clk'event and clk='1') then
            reg1 <= reg1_next;
            reg2 <= reg2_next;
            reg3 <= reg3_next;
            reg4 <= reg4_next;
        end if;
    end process;
    --Next state logic
    reg1_next <= a(7) xor a(6) & a(5 downto 2) & a(0) xor a(1);
    reg2_next <= reg1(5) & reg1(4) xor reg1(3) & reg1(2) xor reg1(1) & reg1(0);
    reg3_next <= reg2(3) xor reg2(2) & reg2(1) xor reg2(0);
    reg4_next <= reg3(1) xor reg3(0);
    y <= reg4;
end arch;
```