Exercise 1

We enter the state machine on a Reset trigger. In state "S0" the output if "Q" is 0. If, in state "S0," A is true, then the FSM goes to state "S1." The output of "S1" is 0 still for "Q." If B is true during state "S1" then the state goes to "S2" where the output of "Q" is now 1. If B is not true, then the FSM returns to state "S0." State "S2" immediately returns to state "S0" after Q is marked true. If during state "S0" A is not true, then the FSM stays at state "S0."

Current State	Input A	Input B	Next State
S0	0	X	S0
S0	1	X	S1
S1	X	0	S0
S1	X	1	S2
S2	X	X	S0

Output Table

State	Output (Q)
00	0
01	0
10	1

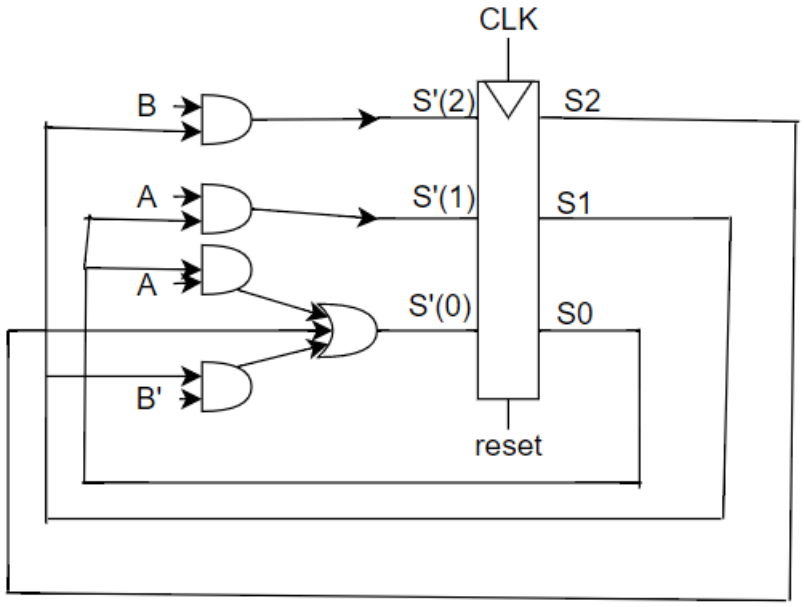
State Encoding

State	Encoding
S0	00
S1	01
S2	10

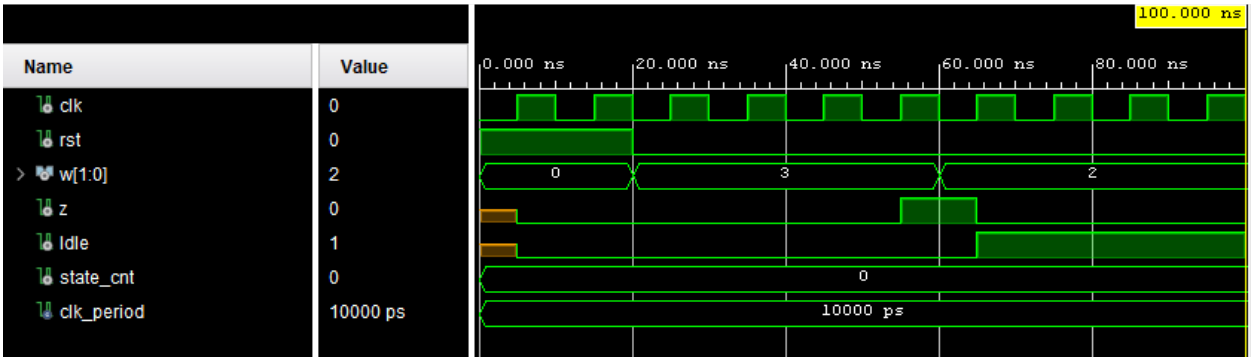
S'(0) = (A * S0) + (B' * S1) + (S2)

S'(1) = (S0 * A)

S'(2) = (S1 * B)



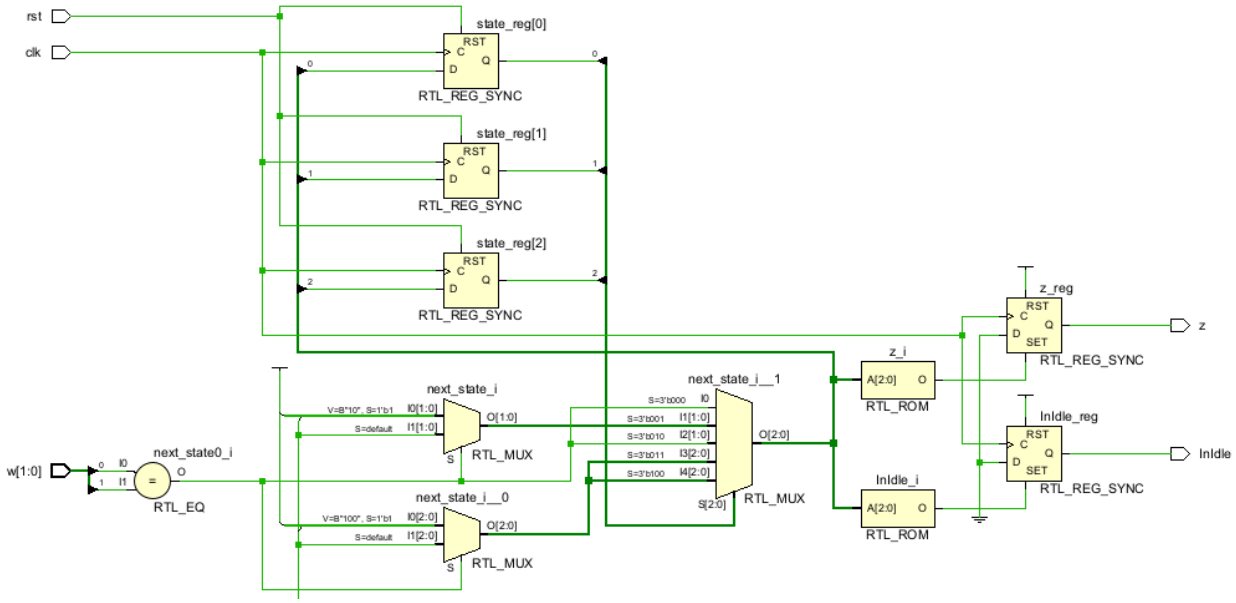
Exercise 2



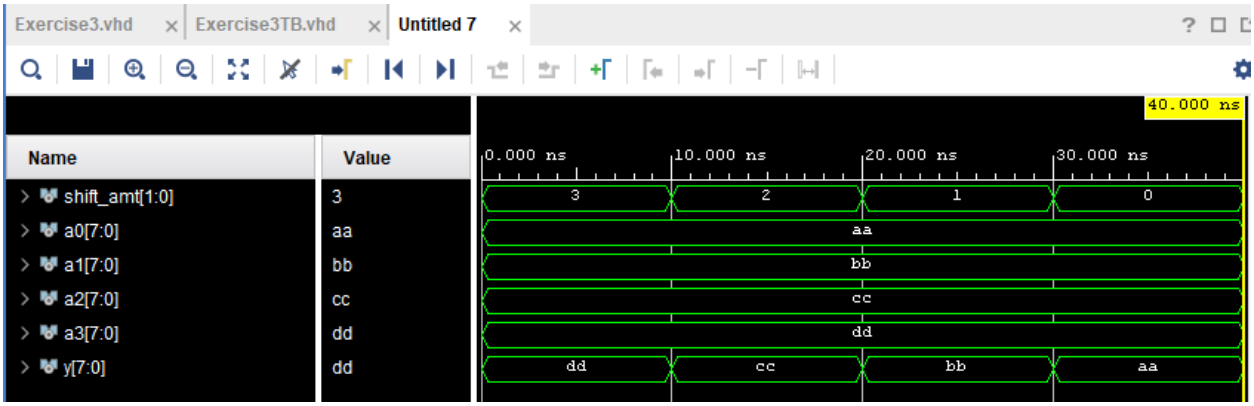

```
library ieee;
use ieee.std_logic_1164.all;

entity Hw7Q1FSM is port (
   clk , rst : in std_logic;
   w : in std_logic_vector (1 downto 0);
   z , InIdle: out std_logic);
end entity Hw7Q1FSM;

architecture obehavior of Hw7Q1FSM is
   type state_type is (idle, s1, s2, s3, s4);
   signal state, next_state: state_type;
begin

state_proc: process (clk) is begin
   if rising_edge (clk) then
      if rst = '1' then
         state <= idle;
      else
         state <= next_state;
      end if;
   end if;
end process;

next_state_proc: process (state, w) is begin
   case state is
      when idle => if (w(0) = w(1)) then
                     next_state <= s1;
                  else
                     next_state <= idle;
                  end if;
      when s1 =>   if (w(0) = w(1)) then
                     next_state <= s2;
                  else
                     next_state <= idle;
                  end if;
```

```
      when s2 =>   if (w(0) = w(1)) then
                     next_state <= s3;
                   else
                     next_state <= idle;
                   end if;
      when s3 =>   if (w(0) = w(1)) then
                     next_state <= s4;
                   else
                     next_state <= idle;
                   end if;
      when s4 =>   if (w(0) = w(1)) then
                     next_state <= s4;
                   else
                     next_state <= idle;
                   end if;
   end case;
end process;

InIdle_proc: process (clk) is begin
   if rising_edge (clk) then
      case next_state is
         when idle => InIdle <= '1';
         when others => InIdle <= '0';
      end case;
   end if;
end process;

z_proc: process (clk) is begin
   if rising_edge (clk) then
      case next_state is
         when s4 => z <= '1';
         when others => z <= '0';
      end case;
   end if;
end process;

end obehavior;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity Hw7Q1FSMSolnTB is
end Hw7Q1FSMSolnTB;

architecture tb of Hw7Q1FSMSolnTB is
   signal clk, rst: std_logic := '0';
```

```
    signal w: std_logic_vector (1 downto 0) := (others => '0');
    signal z, Idle: std_logic;
    constant clk_period: time := 10 ns;
    signal state_cnt: integer := 0;
begin
    dut: entity work.Hw7Q1FSM
    port map (clk => clk, rst => rst, w => w,
    z => z, InIdle => Idle);

    clk <= not clk after clk_period/2;

    stimuli: process is begin
        rst <= '1';
        wait for clk_period;
        wait until clk = '0';
        rst <= '0';

        w <= "11";
        wait for clk_period;
        assert z = '0' report "w same 1 time, z should be 0";

        -- complete. leaving w bits the same for 2 more clk periods
        -- check z = 0 after each period
        w <= "11";
        wait for clk_period;
        assert z = '0' report "w same 2 times, z should be 0";
        w <= "11";
        wait for clk_period;
        assert z = '0' report "w same 3 times, z should be 0";

        w <= "11";
        wait for clk_period;
        assert z = '1' report "w same 4 times, z should be 1";

        -- complete. making w bits different
        -- check Idle is 1
        w <= "10";
        wait for clk_period;
        assert Idle = '1' report "w different, Idle should be 1 now";

        wait;
    end process;
end tb;
```



Exercise 3

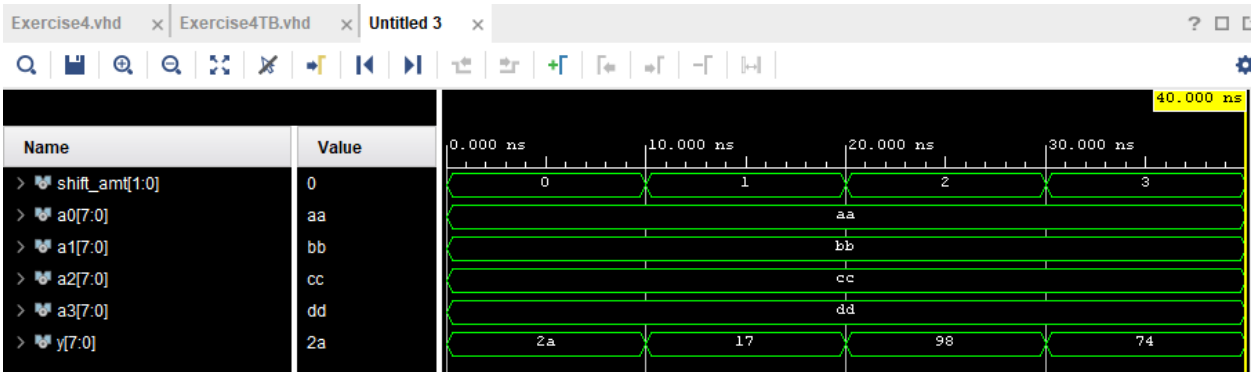



```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

entity Hw6Q5Mux is
   port (
      shift_amt: in std_logic_vector(1 downto 0);
      y: out std_logic_vector(7 downto 0);
      a0: in std_logic_vector(7 downto 0);
      a1: in std_logic_vector(7 downto 0);
      a2: in std_logic_vector(7 downto 0);
      a3: in std_logic_vector(7 downto 0)
   );
end Hw6Q5Mux;
```

```
architecture oh_behavior of Hw6Q5Mux is
   -- create an array of vectors to hold each of n shifters
   type shifty_array is array (3 downto 0) of std_logic_vector(7 downto 0);
   signal y_array: shifty_array := (x"F3", x"A2", x"51", x"00");
begin

   y_array(0) <= a0;
   y_array(1) <= a1;
   y_array(2) <= a2;
   y_array(3) <= a3;

   process(shift_amt, y_array) is begin
      case shift_amt is
         when "00" => y <= y_array(0);
         when "01" => y <= y_array(1);
         when "10" => y <= y_array(2);
         when "11" => y <= y_array(3);
         when others => y <= (others => '0');
      end case;
   end process;

end oh_behavior;

library ieee;
use ieee.std_logic_1164.all;

entity Exc3tb is
end Exc3tb;

architecture Behavioral of Exc3tb is
   signal shift_amt : std_logic_vector(1 downto 0) := (others => '0');
   signal a0, a1, a2, a3 : std_logic_vector(7 downto 0) := (others => '0');
   signal y : std_logic_vector(7 downto 0);
begin
   dut : entity work.Hw6Q5Mux
      port map(
         shift_amt => shift_amt,
         a0 => a0,
         a1 => a1,
         a2 => a2,
         a3 => a3,
         y => y
      );

   stim_proc : process
      begin
         a0 <= x"aa";
         a1 <= x"bb";
```

```
        a2 <= x"cc";
        a3 <= x"dd";
        shift_amt <= "11";
        wait for 10 ns;
        assert y = x"dd" report "Error" severity error;

        shift_amt <= "10";
        wait for 10 ns;
        assert y = x"cc" report "Error" severity error;

        shift_amt <= "01";
        wait for 10 ns;
        assert y = x"bb" report "Error" severity error;

        shift_amt <= "00";
        wait for 10 ns;
        assert y = x"aa" report "Error" severity error;
    end process;
end Behavioral;
```

Exercise 4



The notation 5:0==7:2 is referring to the shift of the contents in 7 downto 2 of the original a3[7:0] value to 5 downto 0. This notation confirms the shift.

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

entity Hw6Q5Mux is
port (
   shift_amt: in std_logic_vector(1 downto 0);
   a0: in std_logic_vector(7 downto 0);
   a1: in std_logic_vector(7 downto 0);
   a2: in std_logic_vector(7 downto 0);
   a3: in std_logic_vector(7 downto 0);
```

```
    y: out std_logic_vector(7 downto 0)
    );
end Hw6Q5Mux;

architecture ohbehave of Hw6Q5Mux is
    -- create array of vectors to hold each of n shifters
    type shifty_array is array (3 downto 0) of std_logic_vector(7 downto 0);
    signal y_array: shifty_array := (x"F3", x"A2", x"51", x"00");
begin


    y_array(0) <= "00" & a0(7 downto 2);
    y_array(1) <= "000" & a1(7 downto 3);
    y_array(2) <= a2(6 downto 0) & '0';
    y_array(3) <= a3(5 downto 0) & "00";

    process(shift_amt, y_array) is begin
        case shift_amt is
            when "00" => y <= y_array(0);
            when "01" => y <= y_array(1);
            when "10" => y <= y_array(2);
            when "11" => y <= y_array(3);
            when others => y <= (others => '0');
        end case;
    end process;


end ohbehave;

library ieee;
use ieee.std_logic_1164.all;

entity Exc3tb is
end Exc3tb;

architecture Behavioral of Exc3tb is
    signal shift_amt : std_logic_vector(1 downto 0) := (others => '0');
    signal a0, a1, a2, a3 : std_logic_vector(7 downto 0) := (others => '0');
    signal y : std_logic_vector(7 downto 0);
begin
    dut : entity work.Hw6Q5Mux
        port map(
            shift_amt => shift_amt,
            a0 => a0,
            a1 => a1,
            a2 => a2,
            a3 => a3,
            y => y
```

```
    );

  stim_proc : process
    begin
      a0 <= x"aa";
      a1 <= x"bb";
      a2 <= x"cc";
      a3 <= x"dd";
      shift_amt <= "00";
      wait for 10 ns;
      assert y = "00101010" report "Error" severity error;

      shift_amt <= "01";
      wait for 10 ns;
      assert y = "00010111" report "Error" severity error;

      shift_amt <= "10";
      wait for 10 ns;
      assert y = "10011000" report "Error" severity error;

      shift_amt <= "11";
      wait for 10 ns;
      assert y = "01110100" report "Error" severity error;
    end process;
end Behavioral;
```