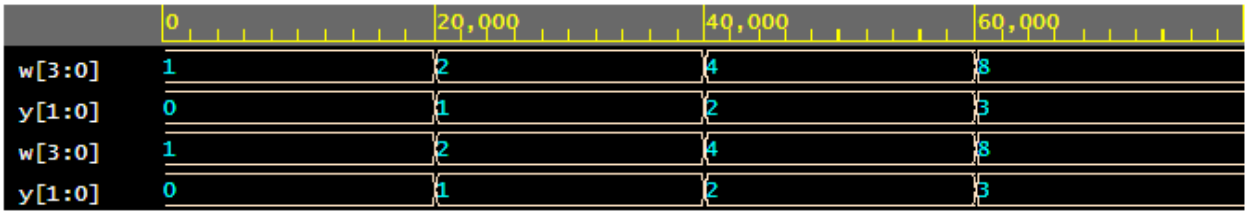
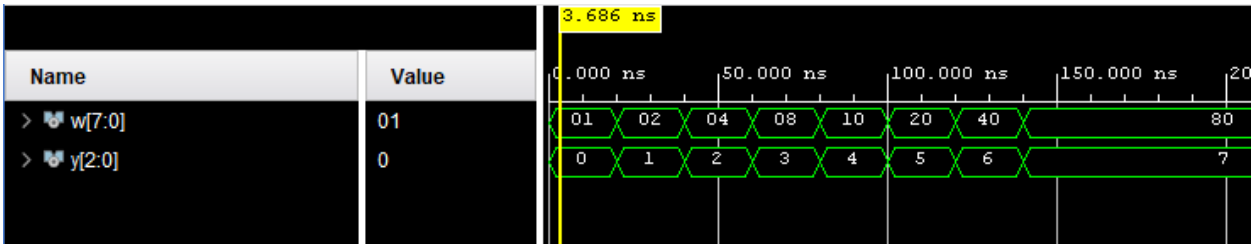
Homework 6 – DSD2

Exercise 1




```
library ieee;

use ieee.std_logic_1164.all;



entity Hw6Q1PriorityEncoder is
   port (
      w : in std_logic_vector (3 downto 0);
      y : out std_logic_vector (1 downto 0)
   );
end entity Hw6Q1PriorityEncoder;

architecture obehavior of Hw6Q1PriorityEncoder is
begin
   if_proc : process(w) is begin
      if w = "0001" then
         y <= "00";
      elsif w = "0010" then
         y <= "01";
      elsif w = "0100" then
         y <= "10";
      else
         y <= "11";
      end if;
   end process;
end obehavior;
```

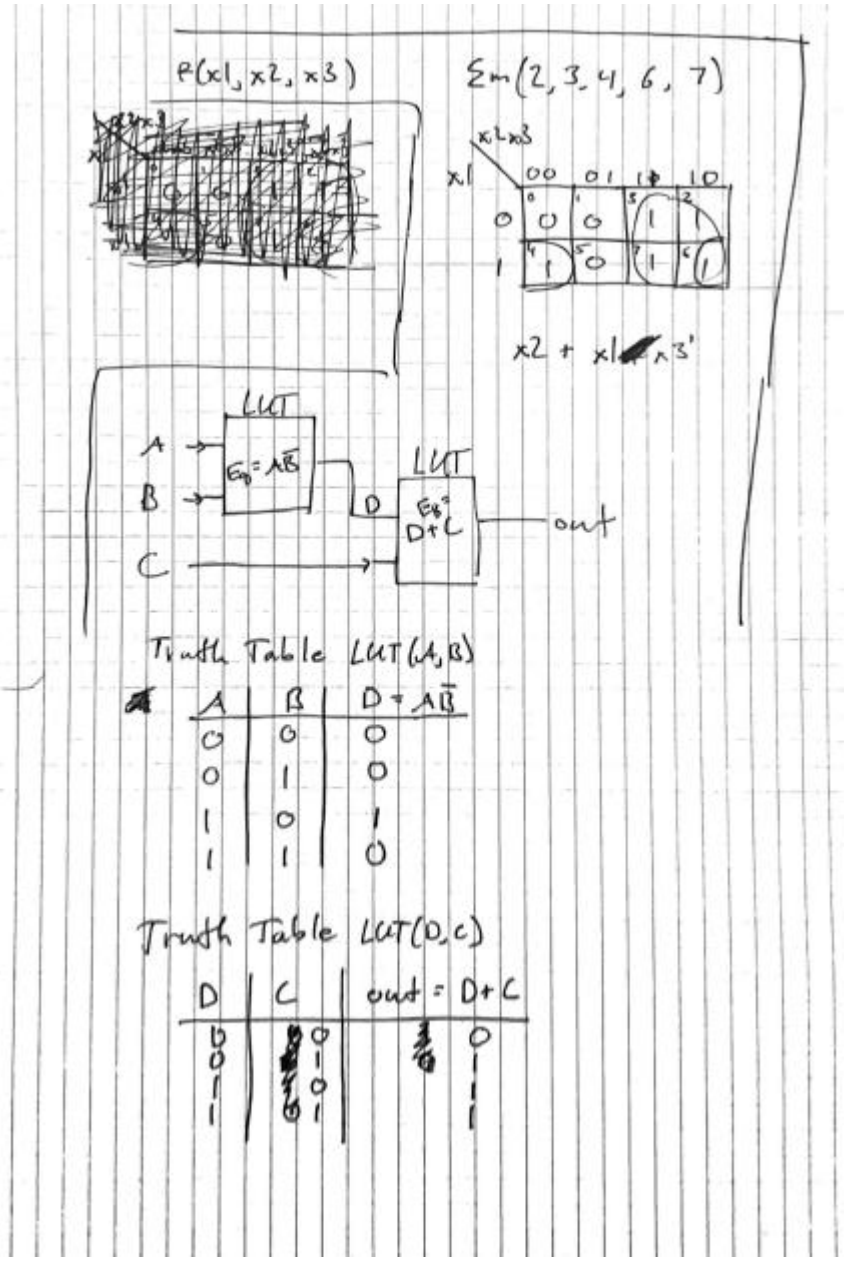
Exercise 2



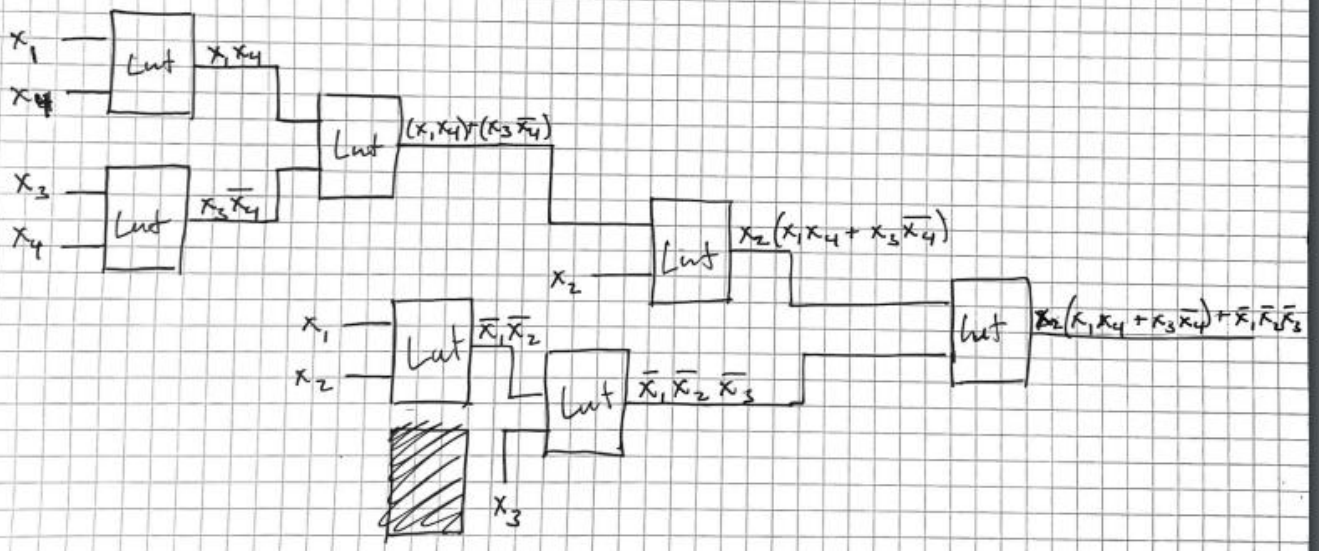
```
library ieee;
use ieee.std_logic_1164.all;

entity Hw6Q2PriorityEncoder is
   port (
      w : in std_logic_vector(7 downto 0);
      y : out std_logic_vector(2 downto 0)
   );
end Hw6Q2PriorityEncoder;

architecture Behavioral of Hw6Q2PriorityEncoder is
begin
   case_proc : process(w) is begin
      case w is
         when "00000001" => y <= "000";
         when "00000010" => y <= "001";
         when "00000100" => y <= "010";
         when "00001000" => y <= "011";
         when "00010000" => y <= "100";
         when "00100000" => y <= "101";
         when "01000000" => y <= "110";
         when others => y <= "111";
      end case;
   end process case_proc;
end Behavioral;


library ieee;
use ieee.std_logic_1164.all;
use IEEE.NUMERIC_STD.ALL;

entity Hw6Q2PriorityEncoderTB is
end Hw6Q2PriorityEncoderTB;

architecture tb of Hw6Q2PriorityEncoderTB is
   signal w : std_logic_vector(7 downto 0);
   signal y : std_logic_vector(2 downto 0);

begin
   dut : entity work.Hw6Q2PriorityEncoder
   port map (
      w => w, y => y
   );

stimuli : process
   -- use the unsigned type when doing math, requires numericstd library
      variable test_vector: unsigned(7 downto 0) := "00000001";
      variable test_result: unsigned(2 downto 0) := "000";
```

```
  begin
    for i in 0 to 7 loop
      -- variables can be assigned to signals
      -- unsigned is simply converted to std_logic_vector
      w <= std_logic_vector(test_vector);
      wait for 10 ns;
      test_vector := shift_left(test_vector, 1);

      assert y = std_logic_vector(test_result)
      report "error, expected y = " & to_hstring(test_result) &
      "actual value is = " & to_hstring(y);
      test_result := test_result + 1;
      wait for 10 ns;
    end loop;
    wait;
  end process;
end tb;
```

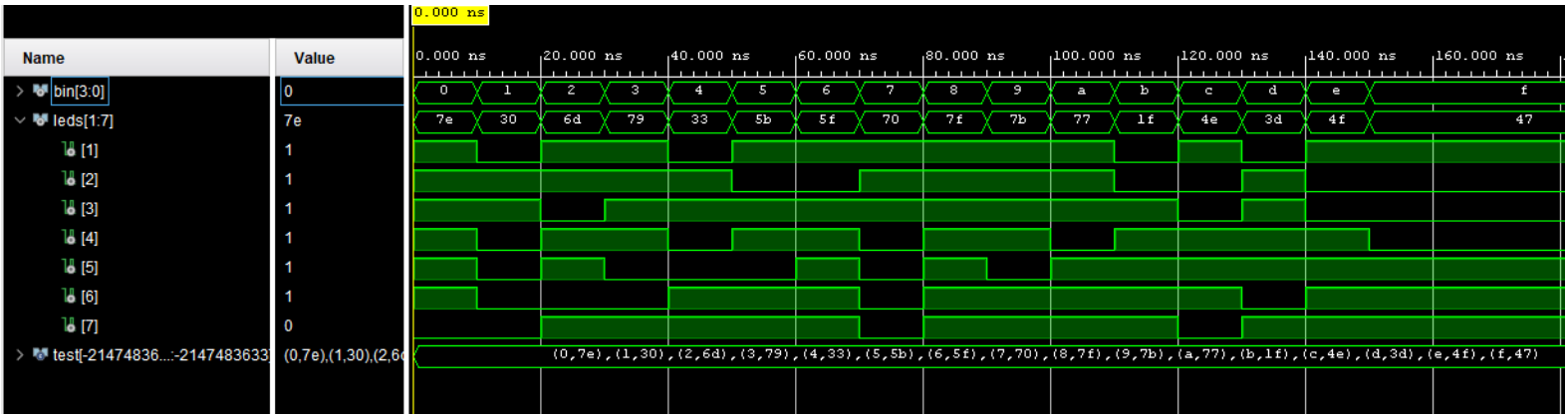
Exercise 3



$f = x_1 x_2 x_4 + x_2 x_3 \bar{x}_4 + \bar{x}_1 \bar{x}_2 \bar{x}_3 \rightarrow x_2(x_1 x_4 + x_3 \bar{x}_4) + \bar{x}_1 \bar{x}_2 \bar{x}_3$



Exercise 4

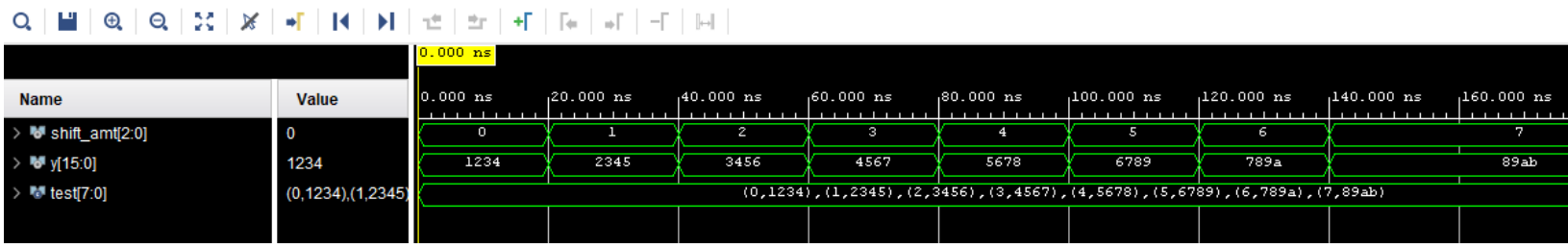



```
library ieee;
use ieee.std_logic_1164.all;

entity Hw6Q4SevenSegSolnTB is
end Hw6Q4SevenSegSolnTB;

architecture tb of Hw6Q4SevenSegSolnTB is
   signal bin : std_logic_vector (3 downto 0) := (others => '0');
   signal leds : std_logic_vector (1 to 7);

   -- using unconstrained vectors in record
   -- sized determined in constant
   -- VHDL 2008 required
   type test_rec is record
      bin : std_logic_vector;
      leds : std_logic_vector;
   end record test_rec;
   type test_rec_array is array (integer range <>) of test_rec;
   constant test : test_rec_array := (
      ("0000", 7UX"7E"),
      ("0001", 7UX"30"),
      ("0010", 7UX"6D"),
      ("0011", 7UX"79"),
      ("0100", 7UX"33"),
      ("0101", 7UX"5B"),
      ("0110", 7UX"5F"),
      ("0111", 7UX"70"),
      ("1000", 7UX"7F"),
      ("1001", 7UX"7B"),
      ("1010", 7UX"77"),
      ("1011", 7UX"1F"),
```

```
      ("1100", 7UX"4E"),
      ("1101", 7UX"3D"),
      ("1110", 7UX"4F"),
      ("1111", 7UX"47")
   );
begin

   dut : entity work.seven_segment_decoder
   port map (bin => bin, leds => leds);

   stimuli : process
   begin
      for i in test'range loop
         bin <= test(i).bin;
         wait for 10 ns;
         assert leds = test(i).leds report "bin = " & to_hstring(bin) & " leds = " & to_hstring(leds) severity
note;
      end loop;
      wait;
   end process;

end tb;



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

entity seven_segment_decoder is
port(
   bin : in std_logic_vector(3 downto 0);
   leds : out std_logic_vector(6 downto 0)
);
end seven_segment_decoder;

architecture Behavioral of seven_segment_decoder is
begin
   leds <= "1111110" when bin = "0000" else
         "0110000" when bin = "0001" else
         "1101101" when bin = "0010" else
         "1111001" when bin = "0011" else
         "0110011" when bin = "0100" else
         "1011011" when bin = "0101" else
         "1011111" when bin = "0110" else
         "1110000" when bin = "0111" else
         "1111111" when bin = "1000" else
         "1111011" when bin = "1001" else
```

```
        "1110111" when bin = "1010" else
        "0011111" when bin = "1011" else
        "1001110" when bin = "1100" else
        "0111101" when bin = "1101" else
        "1001111" when bin = "1110" else
        "1000111" when bin = "1111" else
        "1111111";
end Behavioral;
```

Exercise 5




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

entity Hw6Q5Mux is
   port (
      shift_amt: in std_logic_vector(2 downto 0);
      y: out std_logic_vector(15 downto 0)
   );
end Hw6Q5Mux;

architecture oh_behavior of Hw6Q5Mux is
   -- create an array of vectors to hold each of the shifters
   type shifty_array is array (7 downto 0) of std_logic_vector(15 downto 0);
   signal y_array: shifty_array := (x"89AB", x"789A", x"6789", x"5678", x"4567", x"3456", x"2345",
x"1234");
begin
   y <= y_array(to_integer(unsigned(shift_amt)));
end oh_behavior;
```

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

entity Hw6Q5MuxTB is
end Hw6Q5MuxTB;

architecture tb of Hw6Q5MuxTB is
   signal shift_amt : std_logic_vector(2 downto 0) := (others => '0');
   signal y : std_logic_vector(15 downto 0);

   -- this record is constrained
   -- the index ranges are given
   type test_rec is record
      shift_amt : std_logic_vector(2 downto 0);
      y : std_logic_vector(15 downto 0);
   end record test_rec;
```

```
    type test_rec_array is array(7 downto 0) of test_rec;
    constant test : test_rec_array := (
        ("000", X"1234"),
        ("001", X"2345"),
        ("010", X"3456"),
        ("011", X"4567"),
        ("100", X"5678"),
        ("101", X"6789"),
        ("110", X"789A"),
        ("111", X"89AB")
    );
begin

    dut : entity work.Hw6Q5Mux
        port map (shift_amt => shift_amt, y => y);

    stimuli : process
    begin
        for i in test'range loop
            shift_amt <= test(i).shift_amt;
            wait for 10 ns;
            assert (test(i).y = y) report "error in result. y is " &
                to_hstring(y) & " and should be = " &
                to_hstring(test(i).y) & " when the shift amount is = " &
                to_hstring(test(i).shift_amt);
            wait for 10 ns;
        end loop;
        wait;
    end process;

end tb;
```