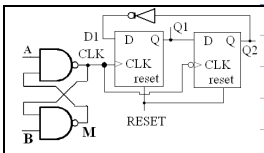
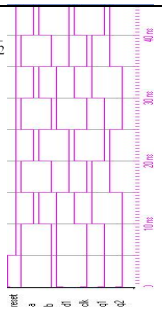




Complete the following wavef, m:

Testbench Solution:

```
Library ieee;
Use ieee.std_logic_1164.all;
entity prob1_tb
end prob1_tb;
architecture beh of prob1_tb is
component  prob1  (a, b, reset: in std_logic;
                   q1, q2:  out  std_logic );
end component;
signal a, b, reset:   std_logic;
signal    q1, q2:     std_logic;
--signal  a:    std_logic   :=  '1';
begin
uut: prob1  port map(a, b, reset, q1, q2);
process
begin
   reset <= '1';
   wait for 5 ns;
   reset <= '0';
   wait for 100 ns;
   wait;
end process;
b <= not a;
 process
 begin
   a <= '1';
   wait for 10 ns;
   a <= '0';
   wait for 5 ns;
   a <= '1';
   wait for 5 ns;
   a <= '0';
   wait for 5 ns;
   a <= '1';
   wait for 5 ns;
   a <= '0';
   wait for 5 ns;
   a <= '1';
   wait for 5 ns;
   a <= '0';
   wait for 5 ns;
   wait;
end process;
end beh;
```

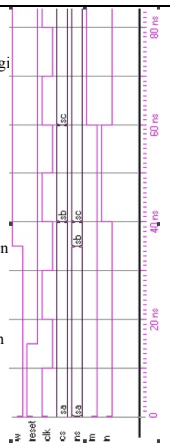
Complete VHDL Design:
You are not allowed to use hierarchical design method in this question.

Solution:

```
Library ieee;
Use ieee.std_logic_1164.all;
entity prob1  (
a, b, reset: in std_logic;
q1, q2:  out  std_logic );
end prob1;
architecture circuit of prob1 is
signal      clk,  m, d1, t1, t2:
std_logic;
begin
   clk <= a  nand m;
   m  <=  b  nand clk;
   d1 <= not t2;
process ( clk , reset)
begin
   if (reset= '1') then
      t1 <= '0';
   elsif (rising_edge(clk)) then
      t1 <= d1;
   end if;
end process;
process (  clk , reset)
begin
   if (reset= '1') then
      t2 <= '0';
   elsif (rising_edge(clk)) then
      t2 <= t1;
   end if;
end process;
q1 <= t1;
q2 <= t2;
end circuit;
```

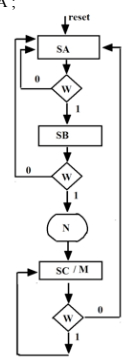
Clock Division

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity clkdiv is
 Port ( clk :   in   STD_LOGIC;
 rst : in   STD_LOGIC;
 clk2 :   out  STD_LOGIC);
end clkdiv;
architecture Behavioral of clkdiv is
signal cnt_div:  std_logic_vector (26 downto 0);
--signal cnt_div:  std_logic_vector (3 downto 0);
begin
process (clk, rst)
begin
if (rst = '1') then
cnt_div <= ( others=>'0' );
clk2 <= '0';
   elsif rising_edge (clk) then
if (cnt_div = 49999999) then
       cnt_div  <= ( others=>'0' );
       clk2     <= '1';
 elsif (cnt_div < 24999999) then
cnt_div  <= cnt_div + 1;
       clk2     <= '1';
      else
       cnt_div  <= cnt_div + 1;
       clk2     <= '0';
      end if;
   end if;
end process;
end Behavioral;
```



```
Library ieee;
Use ieee.std_logic_1164.all;
entity prob2_fsm ( w, Reset, Clk: in std_logi
                   M, N: out std_logic );
End prob2_fsm;
Architecture beh of prob2_fsm is
type  st is ( SA, SB, SC );
signal  cs, ns:  st;
begin
   process ( w ,  cs )
   begin
   case (cs)
                when SA=>  if (w = '0') then
                              ns <= SA ;
                                 else
                           ns  <= SB ;
                          end if;
                when SB=>  if(w = '0') then
                              ns  <= SA ;
                             else
                           ns <= SC ;
                          end if;

                when SC=>  if (w = '0') then
                                 ns <= SA ;
                             else
                           ns <= SC ;
                          end if;
                when others=> ns <= SA;
   end case;
  end process;
process (  Clk ,    Reset)
begin
   if( Reset = '1') then
                cs <= SA ;
   elsif (rising_edge(Clk)) then
                cs <= ns ;
   end if;
 end process;
  M <= '1' when (cs=SC)  else  '0';
  N <= w when (cs=SB)  else '0';
end beh;
```



CRC Error Detection

Message 110101

Generating Polynomial = 101

```
          1 1 1 0 1 1
      ____________
1 0 1 | 1 1 0 1 0 1 0 0
        1 0 1
        -----
          1 1 1
          1 0 1
          -----
            1 0 0
            1 0 1
            -----
              0 1 1
              0 0 0
              -----
                1 1 0
                1 0 1
                -----
                  1 1 0
                  1 0 1
                  -----
                    1 1   <- Remainder = CRC checksum
```

1 1 1 0 1 1 ← Quotient (has no function in CRC calculation)

Message with CRC = 1 1 0 1 0 1 1 1

Hamming Code

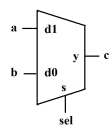
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity Parity_Generator is
   Port ( d4 : in  STD_LOGIC;
          d3 : in  STD_LOGIC;
          d2 : in  STD_LOGIC;
          d1 : in  STD_LOGIC;
Ham_code: out STD_LOGIC_VECTOR
 (6 downto 0));
end Parity_Generator;
architecture Behavioral of Parity_Generator is
signal p1: std_logic;
signal p2: std_logic;
signal p3: std_logic;
begin
p3 <= d4 xor d3 xor d2;
p2 <= d4 xor d3 xor d1;
p1 <= d4 xor d2 xor d1;
Ham_code <= (d4, d3, d2, p3, d1, p2, p1);
end Behavioral;
```

Data D1,D2,D3,D4	Hamming(7,4) Transmitted P1,P2,D1,P3,D2,D3,D4	Diagram
0000	0000000	
1000	1110000	
0100	1001100	
1100	0111100	
0010	0101010	
1010	1011010	
0110	1100110	
1110	0010110	
0001	1101001	
1001	0011001	
0101	0100101	
1101	1010101	
0011	1000011	
1011	0110011	
0111	0001111	
1111	1111111	

```
process(a, b,
sel)
begin
 if (sel =  '1')
then
   c <= a;
 else
   c <= b;
end if;
end process;
```

Draw the synthesized schematic for the VHDL code.

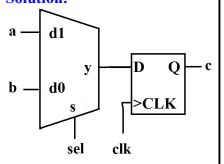
Solution:




```
process (clock )
begin
  if (rising_edge(clock))
then
    if(sel = '1') then
        c <= a;
    else
        c <= b;
    end if;
end if;
end process;
```

Draw the synthesized schematic for the VHDL code.

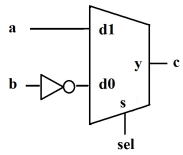
Solution:




```
process (a ,  b ,
sel)
begin
case (sel) is
when '0'=> c
<= ~b;
when '1' => c
<= a;
end case;
end process;
```

Draw the synthesized schematic for the VHDL code.

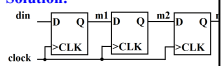
Solution:




```
process (clock )
begin
  if (rising_edge(clock))
then
  m1 <= din;
  m2 <= m1;
  m3 <= m2;
end if;
end process;
```

Draw the synthesized schematic for the VHDL code.

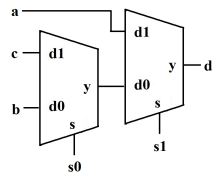
Solution:




```
process ((sl ,
s0 ,  a ,  b ,  c)
begin
if (sl ='1') then
   d <= a;
elsif (s0 = '0')
then
    d <= b;
else
   d <= c;
end if;
end process;
```

Draw the synthesized schematic for the VHDL code.

Solution:



Test bench for Left shift Register

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY leftshiftregister_tb IS
END leftshfitregister_tb;
ARCHITECTURE behavior OF leftshiftregister_tb IS
 -- Component Declaration for the Unit Under Test (UUT)
 COMPONENT leftshiftregister
 PORT(
       clk : IN  std_logic;
       rst : IN  std_logic;
      random_sequence : OUT  std_logic_vector(3 downto
0)
      );
   END COMPONENT;
  --Inputs
  signal clk : std_logic := '0';
  signal rst : std_logic := '0';
--Outputs
  signal random_sequence : std_logic_vector(3 downto 0);
  -- Clock period definitions
  constant clk_period : time := 65 ns;
BEGIN
-- Instantiate the Unit Under Test (UUT)
 uut: leftshiftregister PORT MAP (
        clk => clk,
        rst => rst,
        random_sequence => random_sequence
      );
  -- Clock process definitions
  clk_process :process
  begin
clk <= '0';
wait for clk_period/2;
clk <= '1';
wait for clk_period/2;
 end process;
 -- Stimulus process
 stim_proc: process
 begin
    -- hold reset state for 100 ns.
rst <= '1';
    wait for clk_period/2;
rst <= '0';
    wait for clk_period/2;
   -- wait for clk_period*10;
   -- insert stimulus here
   wait;
 end process;

END;
```

Left Shift Register

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity leftshiftregister is
 Port ( clk : in  STD_LOGIC;
 rst : in  STD_LOGIC;
random_sequence: out
STD_LOGIC_VECTOR
 (3 downto 0));
end leftshiftregister;
architecture Behavioral of leftshiftregister
is
signal df1: std_logic;
signal df2: std_logic;
signal df3: std_logic;
signal df4: std_logic;
begin
process(clk, rst)
begin
if (rst = '1') then
df1 <= '1';
df2 <= '0';
df3 <= '1';
df4 <= '1';
elsif rising_edge (clk) then
df4 <= df3;
df3 <= df2;
df2 <= df1 xor df4;
df1 <= df4;
end if;
end process;
random_sequence <= (df4, df3, df2, df1);
end Behavioral;
```

Hamming code Test Bench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- Uncomment the following library
declaration if using
-- arithmetic functions with Signed or
Unsigned values
--USE ieee.numeric_std.ALL;
ENTITY parity_generator_tb IS
END parity_generator_tb;
ARCHITECTURE behavior OF
parity_generator_tb IS
   -- Component Declaration for the Unit
Under Test (UUT)
   COMPONENT Parity_Generator
   PORT(
       d4 : IN  std_logic;
       d3 : IN  std_logic;
       d2 : IN  std_logic;
       d1 : IN  std_logic;
       Ham_code : OUT  std_logic_vector(6
downto 0)
      );
   END COMPONENT;
  --Inputs
                signal clk : std_logic := '0';
                signal d4 : std_logic := '1';
  signal d3 : std_logic := '0';
  signal d2 : std_logic := '1';
  signal d1 : std_logic := '1';
 --Outputs
  signal Ham_code : std_logic_vector(6
downto 0);
  -- No clocks detected in port list. Replace
<clock> below with
  -- appropriate port name
  constant clock_period : time := 10 ns;
BEGIN
 -- Instantiate the Unit Under Test (UUT)
  uut: Parity_Generator PORT MAP (
       d4 => d4,
       d3 => d3,
       d2 => d2,
       d1 => d1,
       Ham_code => Ham_code
      );
  -- Clock process definitions
  clock_process :process
  begin
clk <= '0';
wait for clock_period/2;
clk <= '1';
wait for clock_period/2;
  end process;

  -- Stimulus process
  stim_proc: process
  begin
 -- hold reset state for 100 ns.
 wait for 100 ns;

 wait for clock_period*10;
    -- insert stimulus here
    wait;
  end process;
END;
```

DECODER

```
library ieee; use
ieee.std_logic_1164.all; entity
dec is port (
sel: in std_logic_vector (2
downto 0); res: out
std_logic_vector (7 downto 0)
); end dec; architecture archi of
dec is begin with sel select res
<= "00000001" when "000",
"00000010" when "001",
"00000100" when "010",
"00001000" when "011",
"00010000" when "100",
"00100000" when "101",
"01000000" when "110",
"10000000" when others;
end archi;
```

8-bit Shift-Left Register with Positive-Edge Clock, Asynchronous Parallel Load, Serial In, and Parallel Out

```
library ieee; use
ieee.std_logic_1164.all; entity
shift is port(
C, SI, ALOAD : in std_logic;
D : in std_logic_vector(7
downto 0);
PO : out std_logic_vector(7
downto 0));
end shift;
architecture archi of shift is
signal tmp: std_logic_vector(7
downto 0);
begin process (C, ALOAD)
begin if (ALOAD='1') then tmp
<= D;
elsif (C'event and C='1') then
tmp <= tmp(6 downto 0) & SI;
end if;
end process;
PO <= tmp;
end archi;
```

8-bit Shift-Left Register with Negative-Edge Clock, Clock Enable, Serial In, and Serial Out

```
library ieee; use
ieee.std_logic_1164.all; entity
shift is port(
C, SI, CE : in std_logic;
SO : out std_logic
); end shift;
architecture archi of shift is
signal tmp: std_logic_vector(7
downto 0); begin process (C,
CE) begin
if (C'event and C='0') then if
(CE='1') then for i in 0 to 6 loop
tmp(i+1) <= tmp(i); end loop;
tmp(0) <= SI; end if; end if;
 end process;
```

Counter

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity new_cnt is
  port(osc: in std_logic; -- 50MHz
   load, updown: in std_logic;
    sw: in std_logic_vector(3 downto 0);
    dout: out std_logic_vector(3 downto 0);
    clkref: out std_logic
  );
end new_cnt;
architecture cnt_beh of new_cnt is
signal   clk: std_logic;
signal   cnt10: std_logic_vector(3 downto 0);
signal   cnt  : std_logic_vector(3
                downto 0);
begin
    clkref <= clk;
    process(osc)
    begin
      if( rising_edge(osc) ) then
        if (cnt10 = 9 ) then
            cnt10 <= ( others => '0' );
            clk   <= '1';
        elsif (cnt10 < 4) then
            cnt10 <= cnt10 + 1;
            clk   <= '1';
        else
            cnt10 <= cnt10 + 1;
            clk   <= '0';
        end if;
      end if;
    end process;
    process(clk, load, updown)
    begin
 if( rising_edge(clk) ) then
          if(load = '1' )  then
             cnt <= sw;
          else
            if(updown = '1') then
              if( cnt = 9 ) then
                 cnt <= "0000";
              else
                 cnt <= cnt + 1;
              end if;
            else
              if( cnt = 0 ) then
                 cnt <= "1001";
              else
                 cnt <= cnt - 1;
              end if;
            end if;
          end if;
        end if;
    end process;
    dout <= cnt;
end cnt_beh;
```

Finite State Machine

```
library IEEE;
use IEEE.std_logic_1164.all;
entity fsm is port (
clk, reset, x1 : IN std_logic;
 outp : OUT std_logic
);
end entity;
 architecture beh1 of fsm is
type state_type is (s1,s2,s3,
                    s4);
signal state, next_state:
state_type;
begin
process1:
 process (clk, reset)
 begin
 if (reset ='1') then
state <= s1;
elsif (clk = '1' and clk'Event)
then
state <= next_state;
 end if;
end process process1;
process2 : process (state, x1)
begin
 case state is when
s1 => if x1='1' then next_state
<= s2;
 else next_state <= s3;
end if;
when s2 => next_state <= s4;
when s3 => next_state <= s4;
when s4 => next_state <= s1;
end case;
end process process2;
 process3 : process (state)
begin
case state is when
s1 => outp <= '1';
when s2 => outp <= '1';
when s3 => outp <= '0';
 when s4 => outp <= '0';
end case;
end process process3;
end beh1;
```