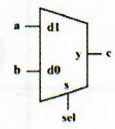
```
process(a, b,
sel)
begin
 if (sel = '1')
then
  c <= a;
 else
  c <= b;
end if;
end process;
```

Draw the synthesized schematic for the VHDL code.

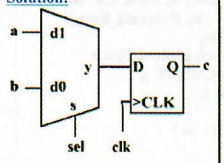
Solution:




```
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.

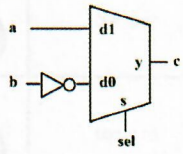
Solution:




```
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.

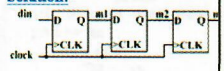
Solution:




```
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.

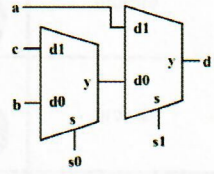
Solution:




```
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:



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;
```

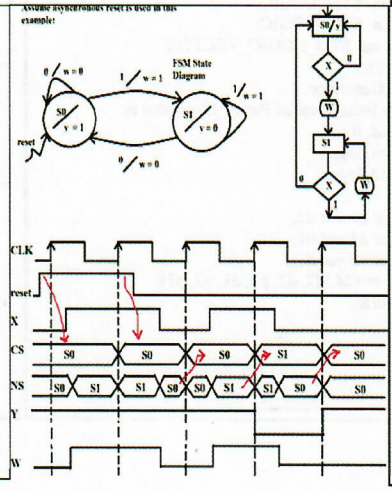
Finite State Machine

```
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;
```

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;
```

Flip-flop with Positive-Edge Clock

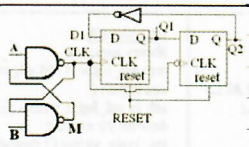
```
library ieee; use ieee.std_logic_1164.all;
entity flop is port(
CLK, D : in std_logic;
Q : out std_logic ); end flop;
architecture archi of flop is
 begin process (CLK)
begin
 if(CLK'event and CLK='1') then
 --if (rising_edge(CLK)) then
Q <= D; end if; end process; end archi;
```

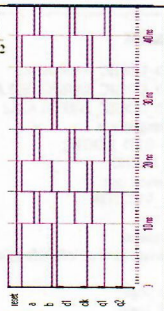
Flip-flop with Negative-Edge Clock and Asynchronous Clear

```
library ieee; use ieee.std_logic_1164.all;
entity flop is port(
C, D, CLR: in std_logic;
Q : out std_logic );
 end flop;
 architecture archi of flop is
 begin process (C, CLR)
 begin
 if (CLR = '1')then
 Q <= '0';
elsif (C'event and C='0')then
--if (falling_edge(CLK)) then
Q <= D; end if; end process; end archi;
```

Flip-flop with Positive-Edge Clock and Synchronous Set

```
library ieee; use ieee.std_logic_1164.all; entity
flop is port(
C, D, S : in std_logic; Q : out std_logic); end
flop;
architecture archi of flop is begin process (C,
S) begin
if (C'event and C='1') then if (S='1') then Q <=
'1'; else Q <= D; end if; end if; end process;
end archi;
```





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:

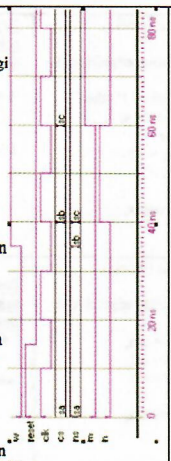
```
Library ieee;
Use ieee.std_logic_1164.all;
entity prob1  (
a, b, reset: in std_logic;
q1, q2:  out  std_logic );
end prob1;
architecture circuit of prob1 is
signal      clk,  m, d1, t1, t2:
std_logic;
begin
   clk <= a  nand m;
   m  <=  b  nand clk;
   d1 <= not t2;
process ( clk , reset)
begin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

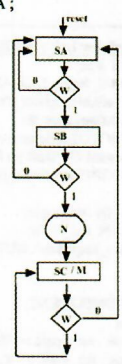
Clock Division

```
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

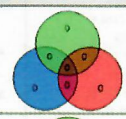
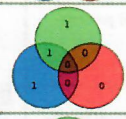
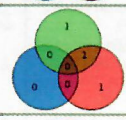
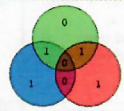
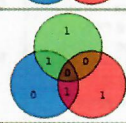
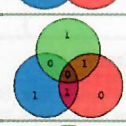
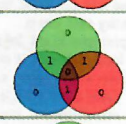
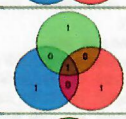
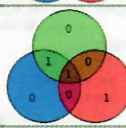
```
          111011
     ____________
101 | 11010100
      101
       ---
       111
       101
       ---
        100
        101
        ---
        011
        000
        ---
         110
         101
         ---
         110
         101
         ---
          11   <- Remainder = CRC checksum
```

111011 — Quotient (has no function in CRC calculation)

Message with CRC = 11010111

Hamming Code

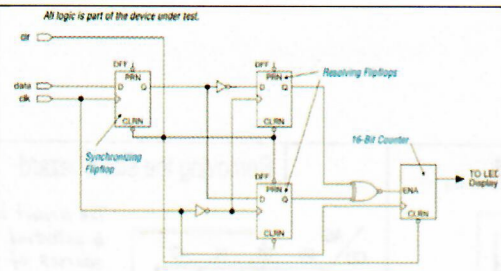
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity Parity_Generator is
   Port ( d4 : in  STD_LOGIC;
          d3 : in  STD_LOGIC;
          d2 : in  STD_LOGIC;
          d1 : in  STD_LOGIC;
Ham_code: out STD_LOGIC_VECTOR
 (6 downto 0));
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	

Metastability is the ability of a digital electronic system to persist for an unbounded time in an unstable equilibrium or metastable state. In metastable states, the circuit may be unable to settle into a stable '0' or '1' logic level within the time required for proper circuit operation. As a result, the circuit can act in unpredictable ways, and may lead to a system failure. For example if you violate the setup and hold time on the input of a flip flop, then the output will be unpredictable for some amount of time. That unpredictable output is called meta-stable. **MTBF (Mean time between failure): (1)** MTBF value shows how metastability affects design. Provides an estimate of the mean time between the probable occurrences of two successive metastable events. :

$$\text{MTBF} = \frac{e^{(C_2 \times t_{MET})}}{C_1 \times f_{CLOCK} \times f_{DATA}}$$

Parameters effect metastability: **"f CLOCK"** parameter refers to the system clock frequency. **"f DATA"** refers to the data transfer frequency. **"t MET"** is the additional time allowed by the system for the flip-flop to settle to a stable state.C1 and C2 are constants, and vary according to the process tech used to manufactured with the same process have similar values.



Multiple stage synchronizer reduce the effects of metastability. Two or more flip-flops are cascaded to from a synchronizer in which two or more flip-flops are cascaded to from a synchronization circuit. If the synchronizing flip-flop produce metastable output, the metastable signal resolve before it is clocked by the second flip-flop. Method does not guarantee that the second flip-flop will not clock an undefined value but it dramatically increases the probability that the data will go to a valid state before it reached the rest of the circuit.

Figure 8. Multiple-Stage Synchronizer

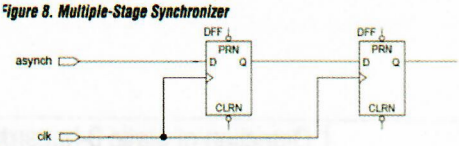
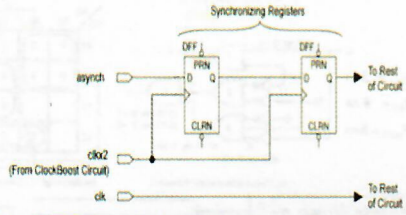


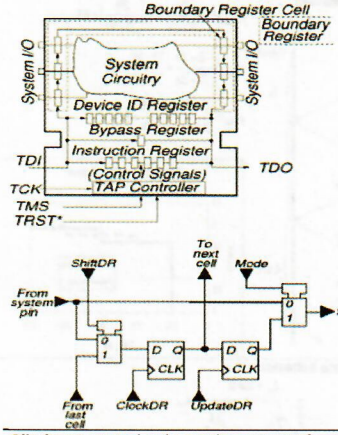
Figure 9. Multiple-Stage Synchronizer with ClockBoost Circuitry



Reduce MTBF: by reducing clk frequency, reduce data frequency, choose faster dff with small setup time and hold time and use synchronizer circuit

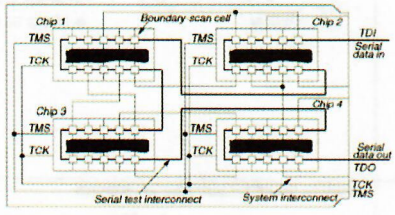
JTAG: *test instructions and data serially fed into component under test (CUT).*can operate at chip, PCB & system levels.*allows control of tristate signals during testing.*let other chips collect response from CUT.*let system interconnect be tested separately from components.*lets components be tested separately from wires. **Elementary boundary scan cell**

System Test Logic

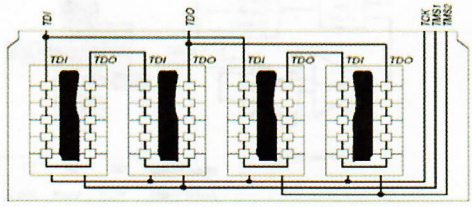


Testing signal: test access port include signals like***TDI** test data in signal accepts serial test data and instructions. Used to shift in vectors or one of many test instructions. Valid on the falling edge of TCK when internal state machine is in the correct state. ***TCK** test clk is synchronizes the internal state machine operations. Run diff. rete from system clk.***TDO** test data out serially shifts out test results captured in boundary scan chain or device ID valid on falling edge of TCK when internal state machine is in the correct state. ***TMS** test mode select is sampled at the raising edge of TCK to determine the next state. Switch system from function test mode***TRST** test reset optional pin, asynchronous TAP controller.

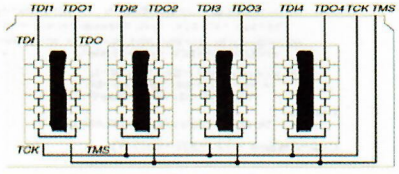
Serial Board / MCM Scan



Parallel Board / MCM Scan

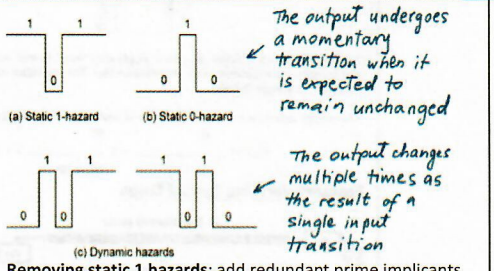


Independent Path Board / MCM Scan

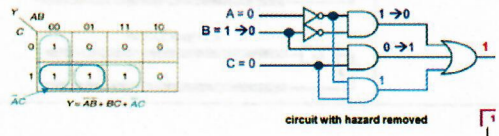


Glitch: unwanted pulse at the output of combinational logic network-momentary change in output that should not have changed. **Hazard:** circuit with the potential for a glitch. Potential unwanted transients occur in output when different paths from input to output have different propagation delays.

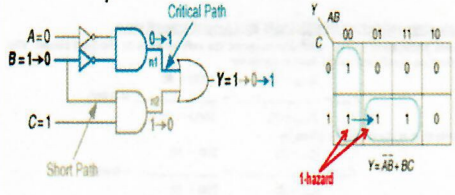
Static Hazards: properly designed 2 lever AND-OR circuit based on sum of products has no static 0 hazards. Static 0 hazard would exit only if both a variables and its complement were connected to the same AND gate, would be nonsense (A.A'.X=0). Properly designed 2 level OR-AND circuit of a product of sums expression has no static 1 hazards. A S-1 hazard exist only if both variables and it complement connected to the same OR gate, which nonsense (A+A'+X=1). **Dynamic Hazards:** If 3 or more paths from an input or its complement to the outpur,the circuit has the potential for a dynamic hazard and the output can exist only a multi-level Network. The dynamic hazard don't occur in a properly designed 2 level AND-OR or OR-AND network. Analysis and elimination of dynamic hazard is a rather complicated process. Hazard free network, use a 2 level network.

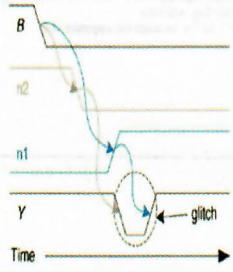


Removing static 1 hazards: add redundant prime implicants (extra prime implicants won't change F , but cause F to b asserted independently of change to the input that cause the hazard)



Detection of Static 1-hazards

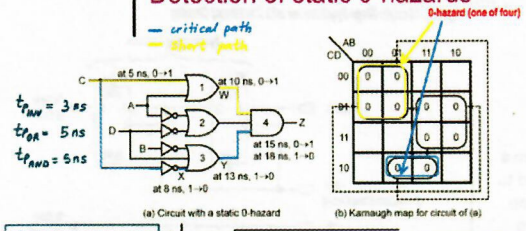




Basically because of gate delays for a moment when B changes is not true that $B+\bar{B}=1$

Temporary violation of complementary law.

Detection of static 0-hazards



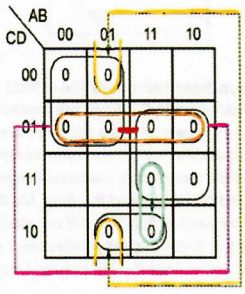
Basically because of gate delays for a moment when C changes is not true that $C \cdot \bar{C} = 0$

Temporary violation of complementary law

$F = (A+C) \cdot (\bar{A}+\bar{D}) \cdot (\bar{B}+\bar{C}+D)$

13

Removing the static hazard

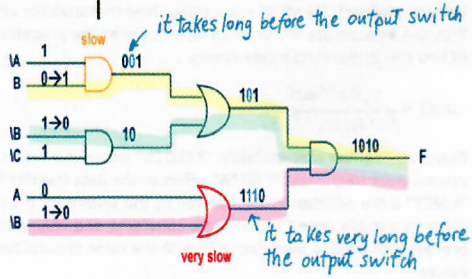


The circuit has 4 potential sources of hazards that must be removed

$F = (A+C) \cdot (\bar{A}+\bar{D}) \cdot (\bar{B}+\bar{C}+D) \cdot (C+\bar{D}) \cdot (A+\bar{B}+D) \cdot (\bar{A}+B+\bar{C})$

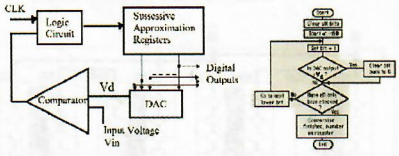
14

Dynamic hazard example



All gates are ideal (very fast) except a slow AND gate and a very slow OR gate

17



For 3-bit SAR ADC:

Digital Number	D3	D2	D1
Analog Weight	Vcc/2	Vcc/4	Vcc/8

Vd = D3 * Vcc/2 + D2 * Vcc/4 + D1 * Vcc/8

Vcc: Power Supply Voltage

Suppose Vi = 2.4 V, Vcc = 5V. For a 3-bit SAR ADC, what will be the final digital value?

Solution: (1). Set the MSB D3 => D3=1, D2=0, D1 = 0. => Vd = 1 * 5/2 = 2.5 V
Vd = 2.5 V > Vi = 2.4 V. So the MSB needs to be reset => D3 = 0.

(2). Set the next bit D2 -> D3=0, D2=1, D1 = 0. -> Vd = 1 * 5/4 = 1.25 V
Vd = 1.25 V < Vi = 2.4 V. So keep D2 to be 1.

(3). Set the next bit D1 -> D3=0, D2=1, D1 = 1.
=> Vd = 1*5/4 +1*5/8 = 1.25 + 0.625 = 1.875 V
Vd = 1.875 V < Vi = 2.4 V. So keep D1 to be 1.

Conclusion: The final 3-bit digits of the 3-bit SAR ADC are 0 1 1.

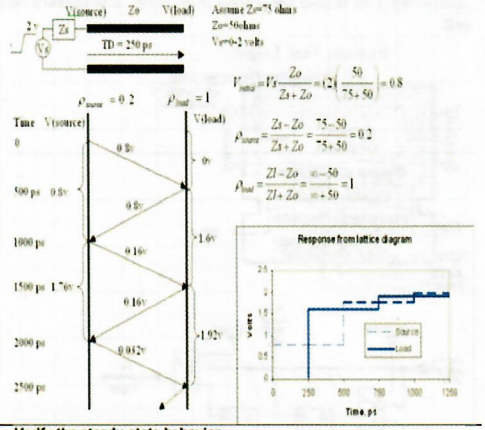
Conversion Error = 2.4 - 1.875 = 0.525 (V)

ADC:

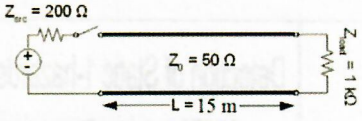
- **Offset Error**
- **Gain Error**
- **Can be eliminated by initial adjustments**
- **Integral Linearity Error**
- **Differential Linearity Error**
- **Nonlinear Error**
  - ◆ Hard to remove

Transmission Line

Transient Analysis - Over Damped



In high-speed circuits, transmission line effects tend to **distort** signals on path that are long compared to the wavelength of the signals propagating on the path



Calculate reflection coefficients and signal transit time

These parameters will be needed to describe the voltage seen by the load circuit. The reflection coefficient at the load is given by

$$\Gamma_{load} = \frac{Z_{load} - Z_0}{Z_{load} + Z_0} = \frac{1000 - 50}{1000 + 50} = 0.905$$

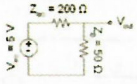
while that at the source is given by

$$\Gamma_{src} = \frac{Z_{src} - Z_0}{Z_{src} + Z_0} = \frac{200 - 50}{200 + 50} = 0.600$$

The wave velocity is given in the problem statement as $v = 3.0 \times 10^8$ m/s. A signal traveling down the 15-meter long power line will take

$T_{trans} = L / v = 15 / (3.0 \times 10^8) = 5 \times 10^{-8}$ s to reach the opposite end

Calculate the voltage before the first reflection

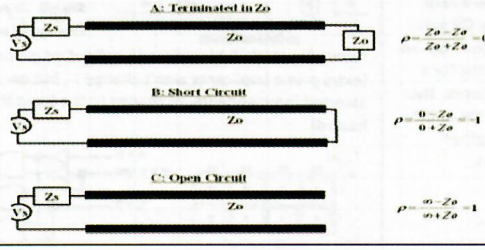


Just after the switch is closed, the power supply only "sees" its own output impedance and the effective impedance of the transmission line. The equivalent circuit model to this situation is a voltage divider.

The voltage applied to the transmission line is, using the voltage divider formula,

$$V_{line} = V_{src} \frac{Z_0}{Z_{src} + Z_0} = 5 \frac{50}{200 + 50} = 1 \text{ V}$$

Transmission Line Special Cases



$\rho = \frac{Zo - Zo}{Zo + Zo} = 0$

$\rho = \frac{0 - Zo}{0 + Zo} = -1$

$\rho = \frac{\infty - Zo}{\infty + Zo} = 1$

Verify the steady-state behavior

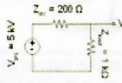
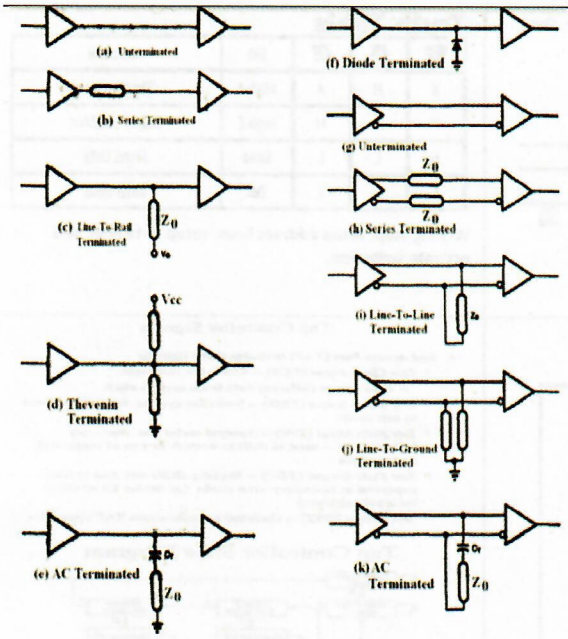


Figure: Equivalent circuit at $t = \infty$ (steady-state) Analyzing the circuit at steady-state will give us a check on our results for the transient analysis.

At steady-state, the power supply outputs a constant voltage. Since for the DC case the effective wavelength is infinitely long (and therefore much longer than the transmission line), we do not need to apply transmission line theory. In fact, the transmission line can be neglected altogether. The resulting circuit is then just a voltage divider. The voltage across the load is given by the voltage divider formula:

$$V_{load} = V_{src} \frac{Z_{load}}{Z_{src} + Z_{load}} = 5 \frac{1000}{200 + 1000} = 4.167 \text{ V}$$

We see that this matches up with the steady-state solution previously calculated



SRAM: (cnt.vhd)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity cnt is
 port(osc: in std_logic; -- 50MHz
 clkref: out std_logic
 );end cnt;
architecture cnt_beh of cnt is
signal clk: std_logic;
signal cnt10: std_logic_vector(12 downto 0);
signal cnt : std_logic_vector(3 downto 0);
begin
 clkref <= clk;
 process(osc)
 begin if( rising_edge(osc) )
 then
 if (cnt10 = 4999 ) then
 cnt10 <= ( others => '0' );
 clk <= '1';
elsif (cnt10 < 2499)
then cnt10 <= cnt10 + 1;
 clk <= '1';
 else
cnt10 <= cnt10 + 1;
 clk <= '0';end if;
 end if;
end process;
end cnt_beh;
```

(ram.vhd)

```
library ieee;use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ram is
port ( address :in   std_logic_vector ( 3 downto 0);
 data   :inout std_logic_vector ( 7 downto 0);
  cs    :in   std_logic;
we    :in   std_logic;
oe    :in   std_logic);end ram;
architecture beh_ram of ram is
type memory is array (0 to 15)of std_logic_vector (7 downto
0);
signal mem : memory ;
begin
MEM_WRITE:
process (address, data, cs, we) begin
 if (cs = '1' and we = '1') then
mem(conv_integer(address)) <= data;
end if;end process;
MEM_READ:
 process (address, cs, we, oe, mem)
 begin
if (cs = '1' and we = '0' and oe = '1')  then
data <= mem(conv_integer(address));
else
data <=  (others => 'Z' );
 end if;end process;
end beh_ram;
```

(mem_top)

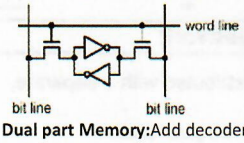
```
LIBRARY ieee;USE ieee.std_logic_1164.ALL;
 ENTITY mem_top IS
port(clk   : IN  std_logic;
     reset : IN  std_logic;
data  : inout std_logic_vector ( 7 downto 0);
 addr_out : OUT  std_logic_vector(3 downto 0);
       cs_out : OUT  std_logic;
      we_out : OUT  std_logic;
      oe_out : OUT  std_logic  );
END mem_top;
 ARCHITECTURE behavior OF mem_top IS
 component ram port (address :in   std_logic_vector ( 3 downto 0);
data   :inout std_logic_vector ( 7 downto 0);
  cs    :in   std_logic;
 we    :in   std_logic;
  oe    :in   std_logic);
end component;
COMPONENT mem_fsm
   PORT(clk   : IN  std_logic;
reset : IN std_logic;
 address : OUT  std_logic_vector(3 downto 0);
data  : INOUT std_logic_vector(7 downto 0);
       cs : OUT  std_logic;
       we : OUT  std_logic;
       oe : OUT  std_logic );
 END COMPONENT;
signal address :std_logic_vector ( 3 downto 0);
  signal  cs    :  std_logic;
signal  we    :  std_logic;
 signal  oe    :  std_logic;
 BEGIN
cs_out <= cs;
we_out <= we;
oe_out <= oe;
addr_out <= address;
 g1: mem_fsm PORT MAP (clk    => clk,
                reset   =>reset,address => address,data => data,cs => cs,
 we => we,oe => oe  );
g2:  ram port map(address =>  address,
            data   =>  data,
            cs => cs,
             we => we,
            oe   => oe);
END;
```

(mem_fsm)

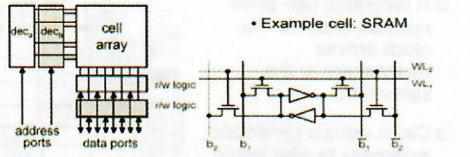
```
library IEEE; use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
 entity mem_fsm is
port ( clk, reset : IN std_logic; address : OUT
std_logic_vector(3 downto 0); data:    INOUT
std_logic_vector(7 downto 0);
 cs, we, oe: OUT std_logic ); end mem_fsm;
architecture fsm_beh of mem_fsm is
 type state_type is (idle, s1,s2,s3,s4);
signal state: state_type ;
signal cnt: std_logic_vector(3 downto 0);
  begin
cs <= '1';
address <= cnt;
process (clk,reset)
begin  if (reset ='1') then
     state   <= idle;
    cnt    <= "0000";
 elsif (clk='1' and clk'event) then
case state is
when idle =>
state <= s1;
cnt  <= "0000";
when s1 =>
state <= s2;
cnt  <= "0000";
when s2 =>
cnt  <=  cnt + 1;
if (cnt  < 15) then
state <= s2;
else
state <= s3;
end if;
when s3 =>
state <= s4;
cnt  <= "0000";
when s4 =>
cnt  <= cnt + 1;
state <= s4;
when others => cnt    <= "0000";
state  <= s1;
```

```
end case;  end if;  end process;
process(state)
 begin
 case state is
when idle => we <= '0';
 oe <= '0';
data <= "ZZZZZZZZ";
when s1 => we <= '1';
oe <= '0';
data <= "11000011";
when s2 => we <= '1';
oe <= '0';
data <= "11000011";
 when s3 => we <= '0';
oe <= '1';
data <= "ZZZZZZZZ";
when s4 => we <= '0';
oe <= '1';
data <= "ZZZZZZZZ";
when others => we <= '0';
oe <= '0';
data <= "ZZZZZZZZ";end case;  end process;  end fsm_beh;
```

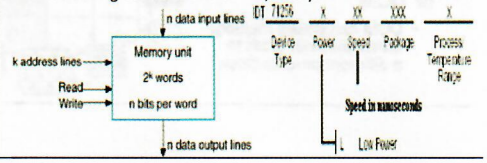
SRAM cell picture below:



Dual part Memory: Add decoder,another set of read/write logic,bits lines, word lines:



SRAM naming convension: 32 * 8,"32by8" => 32 8-bit words.



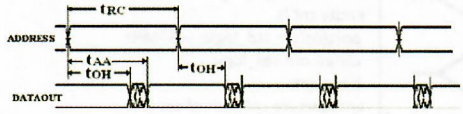
Motivation for Standard

- *Bed-of-nails printed circuit board* tester gone
  - We put components on both sides of PCB & replaced DIPs with flat packs to reduce inductance
    - Nails would hit components
  - Reduced spacing between PCB wires
    - Nails would short the wires
  - PCB Tester must be replaced with built-in test delivery system - - JTAG does that
  - Need standard System Test Port and Bus
  - Integrate components from different vendors
    - Test bus identical for various components
    - One chip has test hardware for other chips

Reading steps: Setup address lines, activate read line, Data available after specified amount of time.

Mutiple read:

WE='1', CS='0', OE='0'

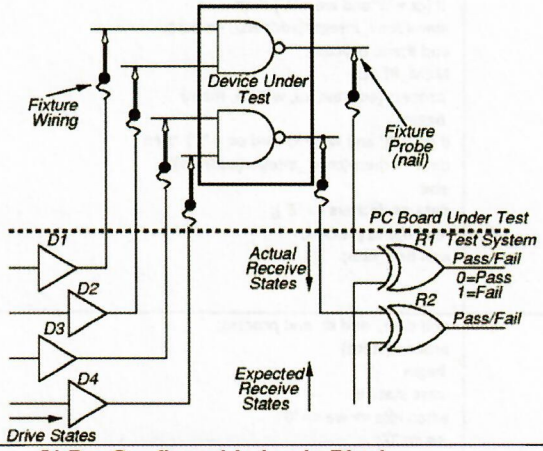


Truth Table

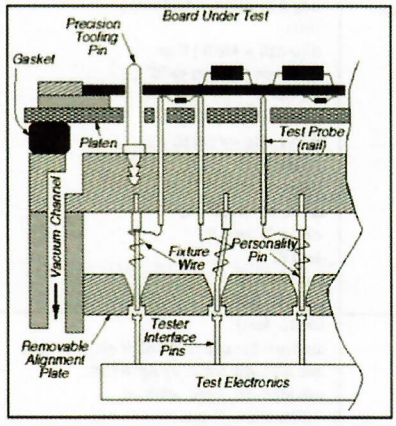
$\overline{WE}$	$\overline{CS}$	$\overline{OE}$	I/O	Function
X	H	X	High-Z	Standby
H	L	H	High-Z	Output Disabled
H	L	L	Dout	Read Data
L	L	X	DIN	Write Data

Writing step: setup address lines, setup data lines and activate write line.

Bed-of-Nails Tester Concept



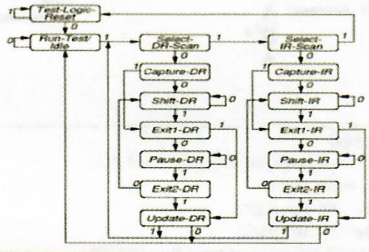
Bed-of-Nails Tester



Tap Controller Signals

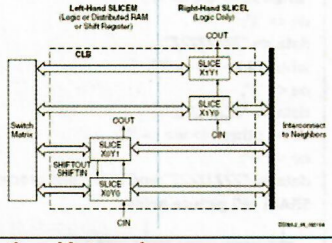
- *Test Access Port* (TAP) includes these signals:
  - *Test Clock Input* (*TCK*) -- Clock for test logic
    - Can run at different rate from system clock
  - *Test Mode Select* (*TMS*) -- Switches system from functional to test mode
  - *Test Data Input* (*TDI*) -- Accepts serial test data and instructions -- used to shift in vectors or one of many test instructions
  - *Test Data Output* (*TDO*) -- Serially shifts out test results captured in boundary scan chain (or device ID or other internal registers)
  - *Test Reset* (*TRST*) -- *Optional* asynchronous TAP controller reset

Tap Controller State Diagram



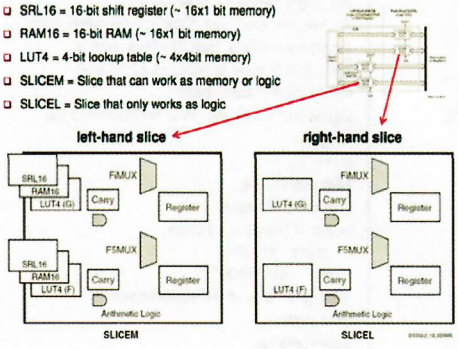
CLB = Configurable Logic Block

- 4 'Slices' per CLB
- Each slice can work as logic (gates + flip-flop), distributed RAM, or shift register
- Switch Matrix connects to FPGA network



Left-hand and Right-hand slice

- SRL16 = 16-bit shift register (~ 16x1 bit memory)
- RAM16 = 16-bit RAM (~ 16x1 bit memory)
- LUT4 = 4-bit lookup table (~ 4x4bit memory)
- SLICEM = Slice that can work as memory or logic
- SLICEL = Slice that only works as logic

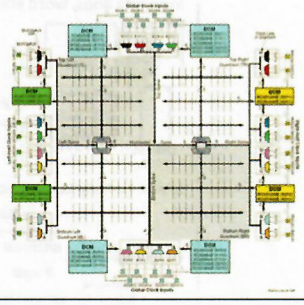


Summary CLB

- 1 configurable logic block contains 4 slices
- Each slice contains a combination of
  - 2 LUT
  - 2 flip flop
  - MUXes
  - Carry-logic
- A LUT can operate as
  - Random logic function of 4 inputs
  - 16-bit RAM
  - 16-bit shift register
- Tools report area results in slices
  - Slices can be many different things, depending on configuration

Interconnection Network

- Clock signals are distributed with a separate, dedicated network
  - Another reason to avoid gated clocks ..
- A recursive 'fish-bone' network ensures that clock arrives everywhere at the same time
- Clock signals generated externally or else inside of 'DCM'
  - DCM can create multiply/ divide external clock to a different on-chip clock



Interconnect Network

- Interconnections are THE critical component in an FPGA
  - Largest physical area on a die
  - Largest power consumption
  - In most cases, wire delay becomes performance bottleneck
- Ironically, interconnections are hardly mentioned as a feature in FPGA product literature
  - Engineers like to hear about the number of CLBs in an FPGA, not about the number of wires between CLBs ...
- FPGA interconnect network is a hierarchical architecture
  - Many fast, short wires with small drive capacity
  - Few longer wires with high drive capacity

Field-Programmable Device (FPD)

- a general term that refers to any type of integrated circuit used for implementing digital hardware, where the chip can be configured by the end user to realize different designs.

Programmable Logic Array (PLA)

- A relatively small FPD
- Contain two levels of logic
  - an AND-plane
  - an OR-plane,
  - both planes are programmable

1. [28 points] Fill out the blanks shown below.

[1]. Given the following VHDL statement,
rst <= '0', '1' after 100 ns, '0' after 200ns;

rst is ___ at time zero, ___ at time 50 ns, ___ at time 100 ns;
___ at time 150 ns, ___ at time 200 ns, ___ at time 300 ns;

[2]. Each of the D2, D1, D0 signals is std_logic type,
signal D2, D1, D0: std_logic;
signal D3: std_logic_vector(2 downto 1);

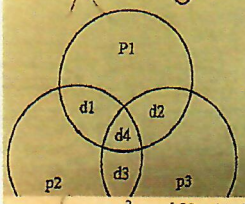
Given the following codes:

D2 <= '0'; D1 <= '1'; D0 <= '1';
D3 <= (D2 xor D1) & (D1 nor D0);

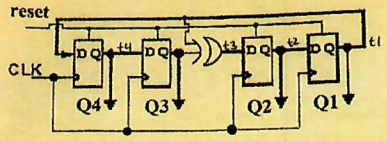
What is binary value for D3? D3 ___

[3]. (7,4) hamming code construction diagram is shown below.
If d4 d3 d2 d1 = "1100", the constructed 7-bit hamming code is:
Bit 7: ___ Bit 6: ___ Bit 5: ___ Bit 4: ___ Bit 3: ___ Bit 2: ___ Bit 1: ___

1100001



2. [32 points]
Design the following circuit in VHDL. This circuit is required to be loaded with value Q4 Q3 Q2 Q1 = "1100" if the asynchronous reset control signal is logic '1'.




```
library ieee;
use ieee.std_logic_1164.all;
entity flop is port(
  clk, reset : in std_logic;
  Q4,Q3,Q2,Q1 : out std_logic);
end flop;
architecture archi of flop is
signal t1,t2,t3,t4 : std_logic;
process (clk,reset)
begin
 if (reset='1') then
   Q4<='1';
   Q3<='1';
   Q2<='0';
   Q1<='0';
 elsif (rising_edge(clk)) then
   t1<=t2;
   t2<=t3 xor t1;
   t3<=t4;
   t4<=t1;
 end if;
end process;
Q4<=t4;
Q3<=t3;
Q2<=t2;
Q1<=t1;
end archi;
```

(2). Design VHDL testbench for the above circuit.

```
library ieee;
use ieee.std_logic_1164.all
entity flop_tb;
end flop_tb;
architecture beh of flop_tb is
component flop (
  clk, D, reset: in std_logic;
  Q4, Q3, Q2, Q1: out std_logic);
end component;
signal clk, D, reset: std_logic;
signal Q4, Q3, Q2, Q1: std_logic;
begin
u1: flop port map (clk, D, reset, Q4, Q3, Q2, Q1);
process
begin
 reset <= '1';
 wait 5ns;
 reset <= '0';
 wait 5ns;
 wait;
end process;
end beh;
```

Missing clk

(3). The above circuits are preloaded with value
Q4Q3Q2Q1 = "1100".
What are the output values **after two clock cycles and reset signal is logic '0'**?

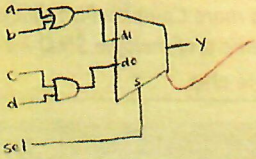
Your answer: Q4 = ___, Q3 = ___, Q2= ___, Q1= ___.

3. [16 points] Draw the synthesized schematic for the VHDL code.

```
signal a, b, c, d, sel: std_logic;
...
...

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

Draw the synthesized schematic for the VHDL code.
Solution:

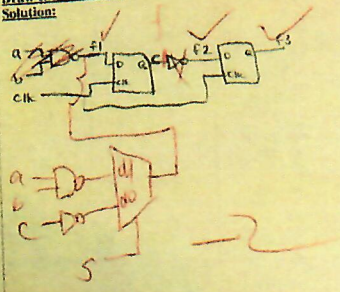



```
signal a, b, c: std_logic;
signal f1, f2, f3;
...
...

f1 <= a nand b;

process (clock )
begin
 if (rising_edge(clock)) then
   if( s = '1') then
      f2 <= f1;
   else
      f2 <= not c;
   end if;
    f3 <= f2;
 end if;
end process;
```

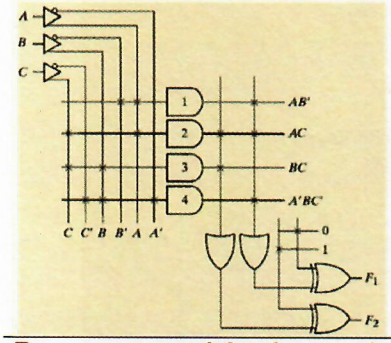
Draw the synthesized schematic for the VHDL code.
Solution:



4. [24 points] Draw ASM Chart for the finite state machine and complete the waveform.

```
Library ieee;
Use ieee.std_logic_1164.all;
entity asm_chart (
   a, reset, clk: in std_logic;
      y: out std_logic );
end asm_chart;
```

PLA Implementation



$F_1 = AB' + AC + A'BC'$

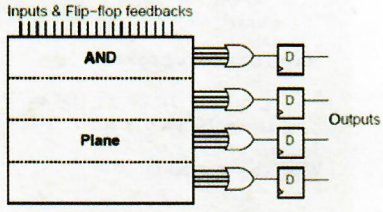
$F_2 = (AC + BC)'$

Programmable Array Logic (PAL)

- PLAs were introduced in the early 1970s by Philips, their main drawbacks were :
  - Expensive to manufacture
  - Offered somewhat poor speed-performance.
  - Both were due to the two levels of configurable logic
- To overcome these weaknesses, Programmable Array Logic (PAL) devices were developed.

Programmable Array Logic (PAL)

- Relatively small FPD that has a programmable AND-plane followed by a fixed OR-plane



Programmable Array Logic (PAL)

- Relatively small FPD that has a programmable AND-plane followed by a fixed OR-plane

Generic Array Logic (GAL)

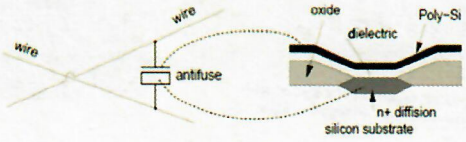
- Programmable Array Logic (PAL), and.

Complex Programmable Logic Devices (CPLDs)

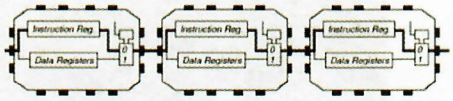
- *CPLD — a more Complex PLD that consists of an arrangement of multiple SPLD-like blocks on a single chip.*

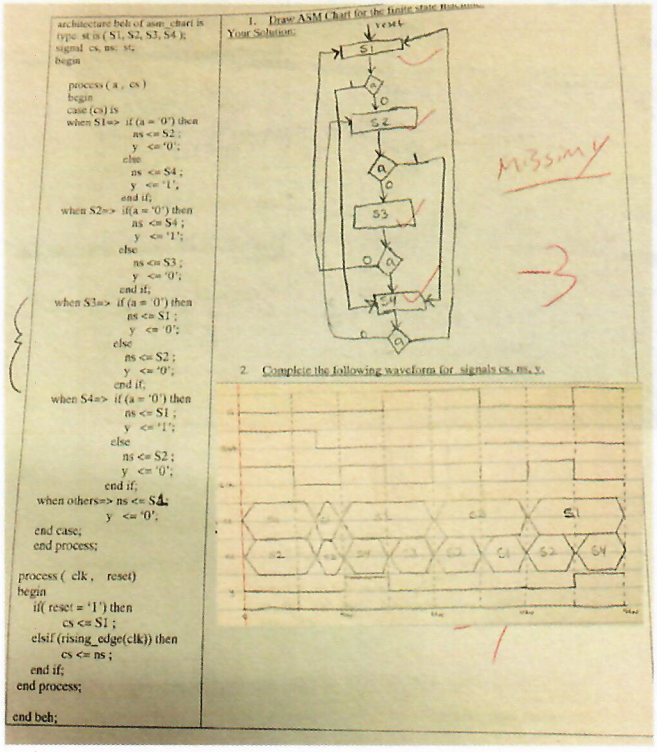
Antifuses

- Antifuses are originally open-circuits and take on low resistance only when programmed.



Instruction Register Loading with JTAG





```
architecture beh of asm_chart is
type st is ( S1, S2, S3, S4 );
signal cs, ns: st;
begin

  process ( a , cs )
  begin
  case (cs) is
  when S1=> if (a = '0') then
              ns <= S2 ;
              y  <='0';
            else
              ns <= S4 ;
              y  <= '1';
            end if;
  when S2=> if(a = '0') then
              ns <= S4 ;
              y  <= '1';
            else
              ns <= S3 ;
              y  <= '0';
            end if;
  when S3=> if (a = '0') then
              ns <= S1 ;
              y  <= '0';
            else
              ns <= S2 ;
              y  <='0';
            end if;
  when S4=> if (a = '0') then
              ns <= S1 ;
              y  <= '1';
            else
              ns <= S2 ;
              y  <= '0';
            end if;
  when others=> ns <= S4;
              y  <= '0';
  end case;
  end process;

process ( clk ,  reset)
begin
  if( reset = '1') then
      cs <= S1 ;
  elsif (rising_edge(clk)) then
      cs <= ns ;
  end if;
end process;

end beh;
```

1. Draw ASM Chart for the finite state machine.

Your Solution:

2. Complete the following waveform for signals cs, ns, y.