# BILKENT UNIVERSITY
# Department of Electrical and Electronics Engineering
# EEE102 Introduction to Digital Circuit Design
# Midterm Exam-1

**15-11-2006**

**Duration 120 minutes**

**Surname: _____**

**Name: _____**

**ID-Number: _____**

**Signature: _____**

**There are 6 questions. +Solve all.**
**Do not detach pages. Show all your work.**

| | |
|---|---|
| **Q1** | |
| **Q2** | |
| **Q3** | |
| **Q4** | |
| **Q5** | |
| **Q6** | |
| **Total** | |

**Q1. (20 points)**

a) (8 pts.) Draw using simple gates, the internal circuitry of a 2-to-4 binary decoder with active-low enable.

b) (6pts.) Draw how a square wave is displayed on the oscilloscope screen if the probe is properly compensated, under-compensated or over-compensated.

c) (6 pts.) For the truth table given below draw a circuit which implements it. F is the output and A, B, C are the inputs.

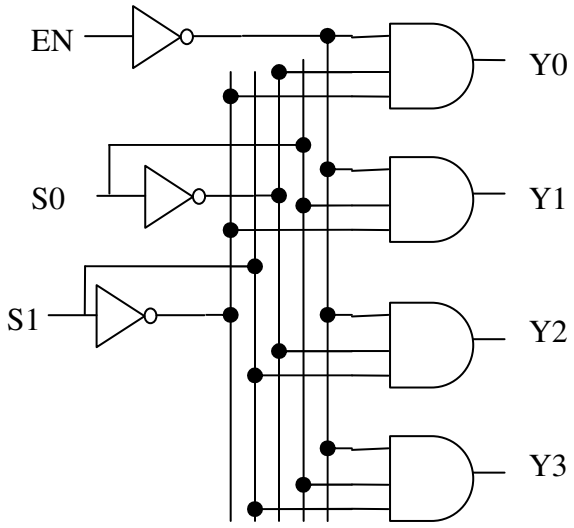
| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

SOLUTION:

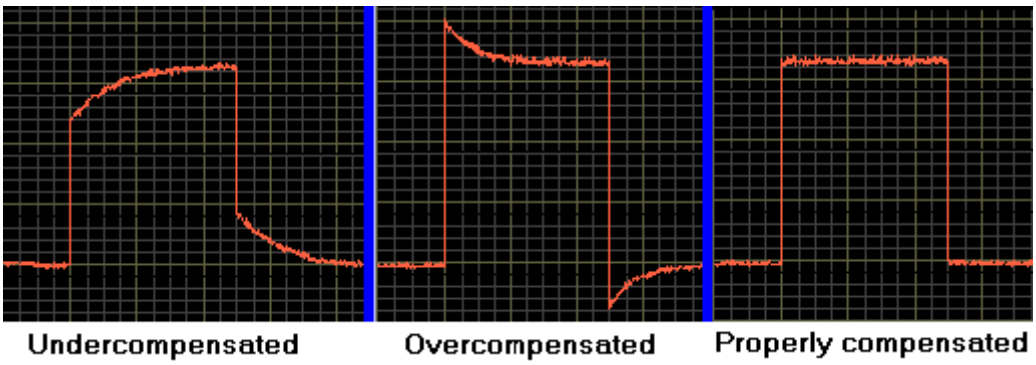
a)

The functions expressing the input output relations in a 2-to-4 binary decoder with enable are

Y0 = EN' and S1' and S0'
Y1 = EN' and S1' and S0
Y2 = EN' and S1 and S0'
Y3 = EN' and S1 and S0

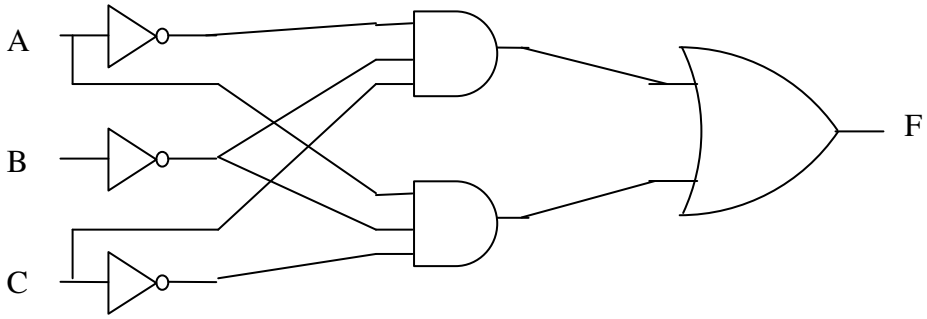


b)




c)

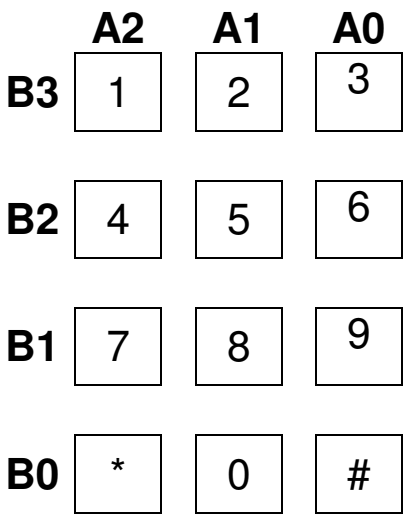
F = A'B'C + AB'C'

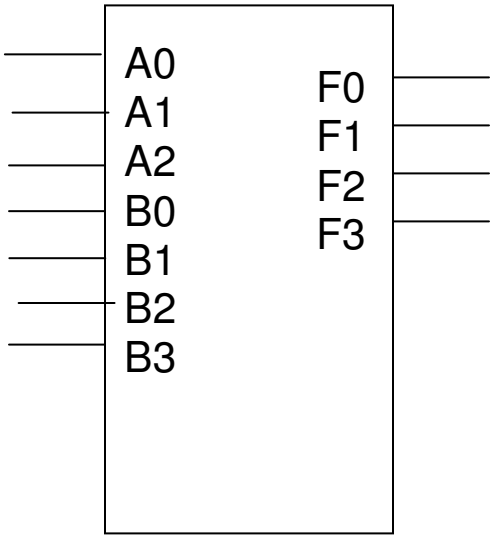



**Q2. (20 points)**

a) (10 pts.) Write the truth table of an encoder which encodes the pressed key in the following keypad. Assume that more than one key is not pressed at the same time. The keypad works as follows: For example, when the key 1 is pressed, A2 and B3 are asserted, or when key 5 is pressed, A1 and B2 are asserted.
This 7-to-4 encoder outputs the binary equivalent of the pressed number. When * key is pressed, the encoder outputs the binary equivalent of $10_{10}$ and when # key is pressed, the encoder outputs the binary equivalent of $11_{10}$. If no key is pressed, all outputs are "1". The pin diagram of the 7-to-4 encoder that you will design is shown below.

b) (10 pts.) Describe the above encoder using VHDL




SOLUTION:

a)

| A2 | A1 | A0 | B3 | B2 | B1 | B0 | F3 | F2 | F1 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| All others | | | | | | | d | d | d | d |

b)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity keypadencoder is
port(A:in std_logic_vector (2 downto 0);
```

```
        B:in std_logic_vector (3 downto 0);
        F:out std_logic_vector (3 downto 0));
end keypadencoder;

architecture Behavioral of keypadencoder is

begin
process(A,B)
begin
  if A="000" and B="0000" then F<="1111";

        elsif A="001" and B="0001" then F<="1011";
        elsif A="001" and B="0010" then F<="1001";
        elsif A="001" and B="0100" then F<="0110";
        elsif A="001" and B="1000" then F<="0011";

        elsif A="010" and B="0001" then F<="0000";
        elsif A="010" and B="0010" then F<="1000";
        elsif A="010" and B="0100" then F<="0101";
        elsif A="010" and B="1000" then F<="0010";

        elsif A="100" and B="0001" then F<="1010";
        elsif A="100" and B="0010" then F<="0111";
        elsif A="100" and B="0100" then F<="0100";
        elsif A="100" and B="1000" then F<="0001";

        else F<="1111";
  end if;

end process;

end Behavioral;
```

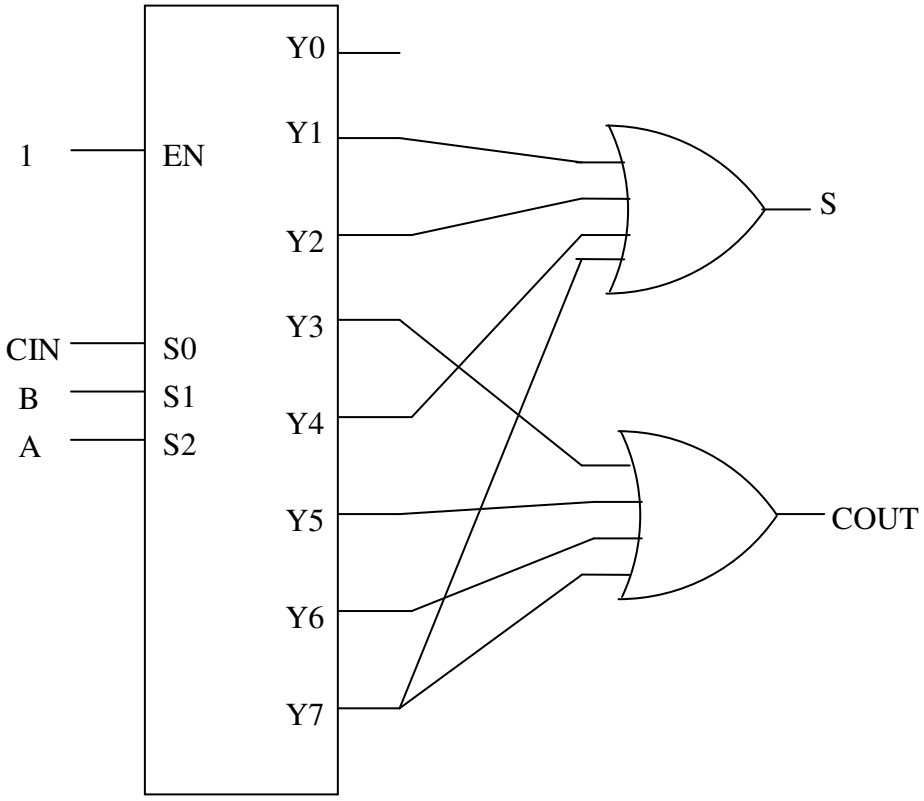
For the else F<="1111"; statement you can use any other value because these input combinations do not occur. In order to make the design process simpler and to achieve a smaller design you can also write else F<="----"; meaning don't care. The sign - is the same as the d that we use in class.

**Q3. (15 points)**
Implement a single bit full adder using a generic 3-to-8 decoder and additional simple gates as necessary.

SOLUTION:

| A | B | CIN | S | COUT |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |

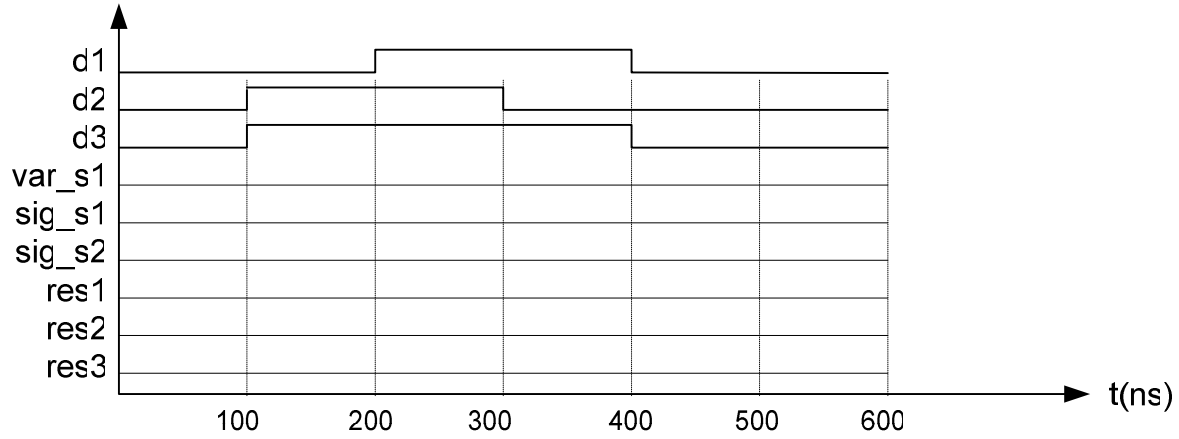
| | | | | |
|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |



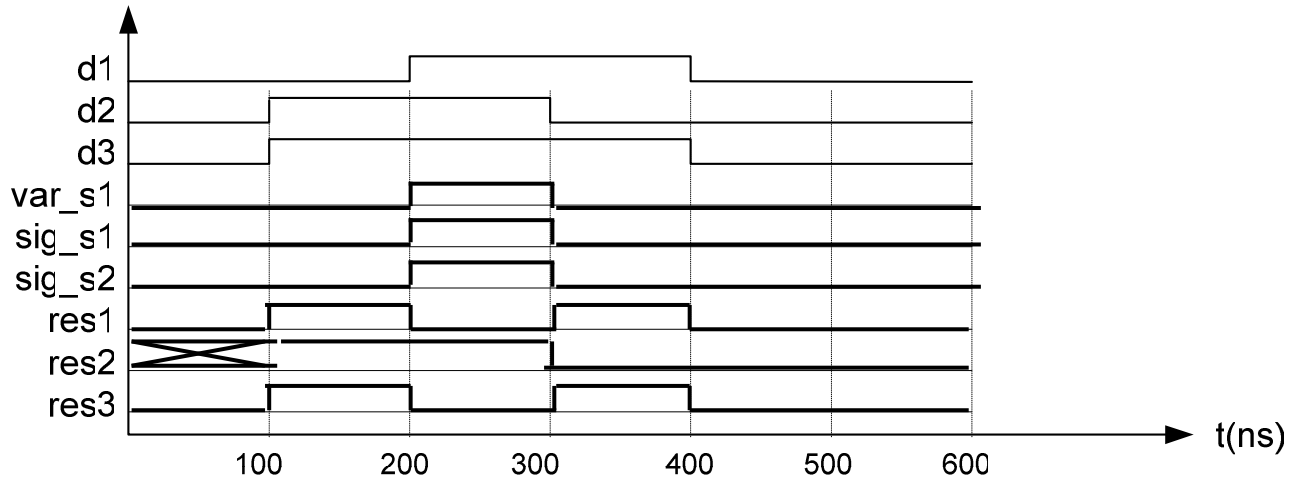

**Q4. (15 points)**
Draw the time waveforms of var_s1, sig_s1, sig_s2, res1, res2, and res3 for t≥0 for the given time waveforms of d1, d2, and d3 for a circuit described by the following VHDL code.

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

entity sig_var is
port(d1, d2, d3:  in std_logic;
      res1, res2, res3: out std_logic);
```

```
end sig_var;

architecture behv of sig_var is

  signal sig_s1: std_logic;
  signal sig_s2: std_logic;

begin

  proc1: process(d1,d2,d3)
   variable var_s1: std_logic;
  begin
      var_s1 := d1 and d2;
      res1 <= var_s1 xor d3;
  end process;

  proc2: process(d1,d2,d3)
  begin
      sig_s1 <= d1 and d2;
      res2 <= sig_s1 xor d3;
  end process;

  proc3: process(d1,d2,d3,sig_s2)
  begin
      sig_s2 <= d1 and d2;
      res3 <= sig_s2 xor d3;
  end process;

 end behv;
```



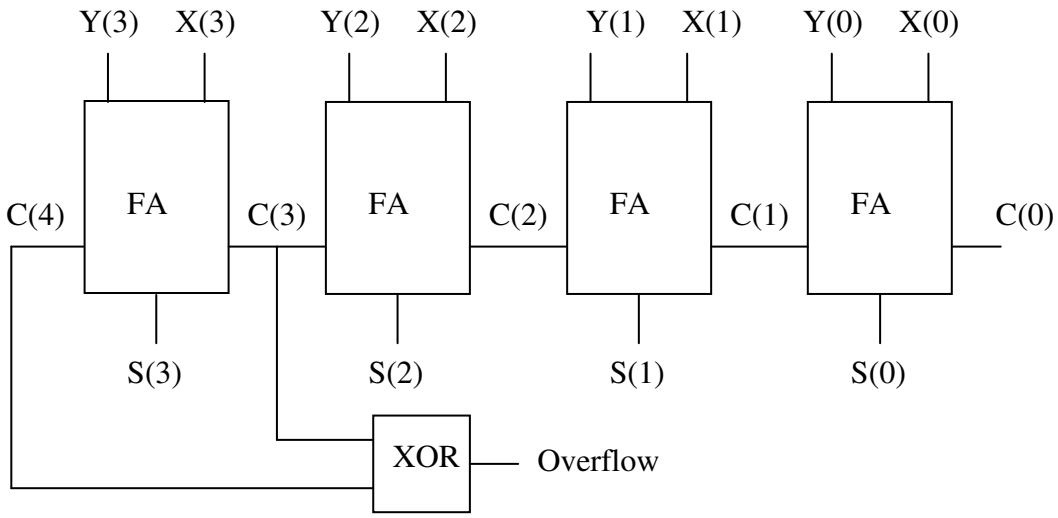

**SOLUTION:**




**Q5. (15 points)**
Write a VHDL code to obtain a 4-bit adder with overflow detector using the VHDL module FA. FA module is a Full Adder and it is already in the library with the following entity declaration.

```
entity FA is
      port (A, B, CIN : in std_logic;
            S, COUT : out std_logic);
end FA;
```

SOLUTION: We have not specified in the problem whether we want a design for two's complement 4-bit numbers. Below is the design for two's complement 4-bit numbers.




The same circuit can be used for unsigned 4-bit binary numbers except that in that case Overflow = C(4).
A different design can also be made for sign-magnitude 4-bit numbers in which case the design is more complicated.

Code for FA which you did not have to write in the exam:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FA is
   Port ( A,B,CIN : in std_logic;
          S,COUT : out std_logic);
end FA;

architecture Behavioral of FA is
begin
S <= A xor B xor CIN;
COUT <= (A and B) or (CIN and (A or B));
end Behavioral;
```

Code for the 4-bit adder for two's complement numbers:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
-- four bit adder for two's complement numbers

entity foutbitadderwo is
   Port ( X : in std_logic_vector (3 downto 0);
          Y : in std_logic_vector (3 downto 0);
          S : out std_logic_vector (3 downto 0);
          O : out std_logic);
end foutbitadderwo;

architecture Behavioral of foutbitadderwo is
component FA
   Port ( A : in std_logic;
          B : in std_logic;
          CIN : in std_logic;
          S : out std_logic;
          COUT : out std_logic);
end component;
signal C: std_logic_vector (4 downto 0) :="00000";
begin
L1: FA port map(X(0),Y(0),C(0),S(0),C(1));
L2: FA port map(X(1),Y(1),C(1),S(1),C(2));
L3: FA port map(X(2),Y(2),C(2),S(2),C(3));
L4: FA port map(X(3),Y(3),C(3),S(3),C(4));
O<=C(3) xor C(4);
end Behavioral;
```

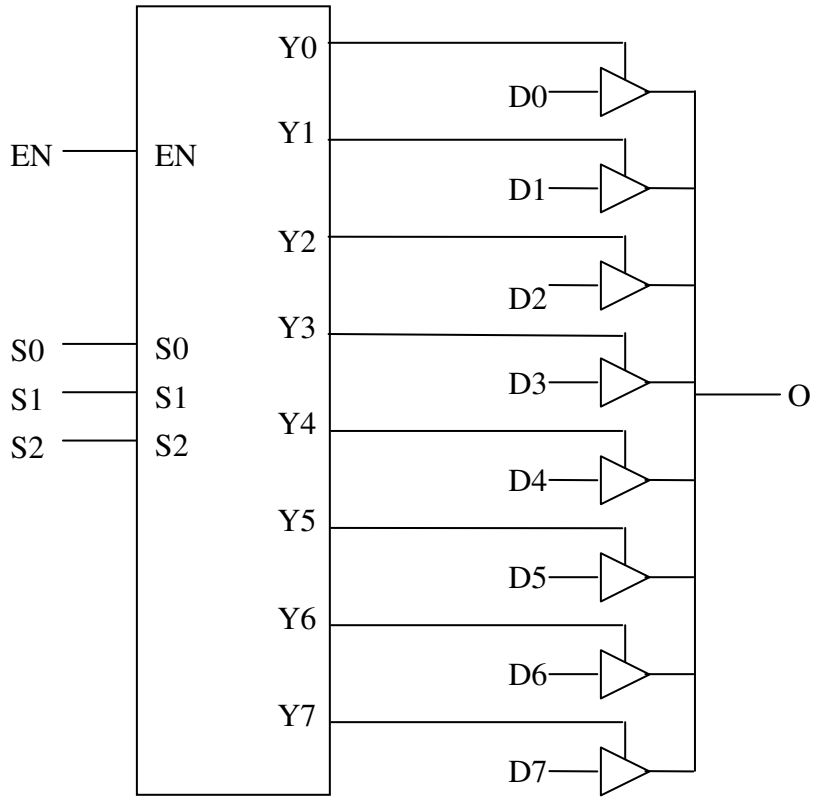
It is also possible to instantiate the FAs using a for-generate loop which is given below for your information.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- four bit adder for two's complement numbers

entity adderpro is
   Port ( X : in std_logic_vector (3 downto 0);
          Y : in std_logic_vector (3 downto 0);
          S : out std_logic_vector (3 downto 0);
          O : out std_logic);
end adderpro;

architecture Behavioral of adderpro is
component FA
   Port ( A : in std_logic;
          B : in std_logic;
```

```
        CIN : in std_logic;
        S : out std_logic;
        COUT : out std_logic);
end component;
signal C: std_logic_vector (4 downto 0) :="00000";
begin
        G1:for i in 0 to 3 generate
        begin
        U1: FA port map(X(i),Y(i),C(i),S(i),C(i+1));
        end generate;
        O<=C(3) xor C(4);
end Behavioral;
```

**Q6. (15 points)**
Design and draw a 8-to-1 multiplexer with enable, using a 3-to-8 decoder with enable, and 8 three-state buffers. When the multiplexer that you design is disabled, its output must be at high Z. Label all pins, inputs, outputs etc. properly.

SOLUTION:




S0, S1, S2 are the selects of the 8-to-1 mux, EN is its enable, O is its output and D0,…,D7 are its data inputs. When EN is '0' the output is at High-Z.