1. [30 pts] Below is a behavioral VHDL code.

```
entity priority_4in is
   port (  inputs : in  std_logic_vector(3 downto 0);
           result : out std_logic_vector(1 downto 0);
           valid  : out std_logic; );
end priority_4in;

architecture if_statement of priority_4in is
begin
   process(inputs)
   begin
      if (inputs(3) = '1') then
         result <= "11";
         valid  <= '1';
      elsif (inputs(2) = '1') then
         result <= "10";
         valid  <= '1';
      elsif (inputs(1) = '1') then
         result <= "01";
         valid  <= '1';
      elsif (inputs(0) = '1') then
         result <= "00";
         valid  <= '1';
      else
         result <= "00";
         valid  <= '0';
      end if;
   end process;
end if_statement;
```

a) [15 pts] Construct the truth table and derive the optimized *Boolean expressions for the outputs*.

[blank]

b) [15 pt] Use ***case*** statement to complete the VHDL code.

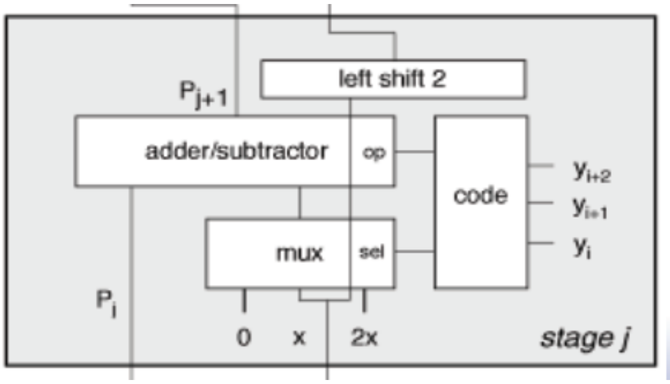
```
entity priority_4in is
 port (
    inputs : in  std_logic_vector(3 downto 0);
    result : out std_logic_vector(1 downto 0);
    valid  : out std_logic;
        );
end priority_4in;

architecture case_statement of priority_4in is
begin
        process(inputs)
         begin
         --- use case statement










        end process;
end case_statement;
```

2. [30 pts] A function takes a binary vector of any size as input and returns an integer representing the **maximum consecutive 1's** in the input vector. It iterates through each element of the input vector, and finally, it returns **max_count**, which represents the **maximum consecutive 1's** in the input vector. The following VHDL defines an entity named **MaxConsecutiveOnesEntity** with an 8-bit input vector (**input_vector**) and an output representing **the maximum consecutive 1's** (**max_count_output**). In the architecture, a process is used to call the **max_consecutive_ones** function with the **input_vector** and assign the result to **max_count_output**.
For example: If the input_vector is "00**1111**00" then the output, max_count_output = 4. If the input DATA is "**111**00010" or "10010**111**"then the output, max_count_output = 3.

```
entity MaxConsecutiveOnesEntity is
   port (
      input_vector : in std_logic_vector(7 downto 0); -- Example: 8-bit input vector
      max_count_output : out integer -- Output representing max consecutive 1's
         );
end entity MaxConsecutiveOnesEntity;

architecture Behavioral of MaxConsecutiveOnesEntity is
   function max_consecutive_ones (in_vector : std_logic_vector) return integer is



   end function max_consecutive_ones;

begin
   process(input_vector)



   end process;
end architecture Behavioral;
```

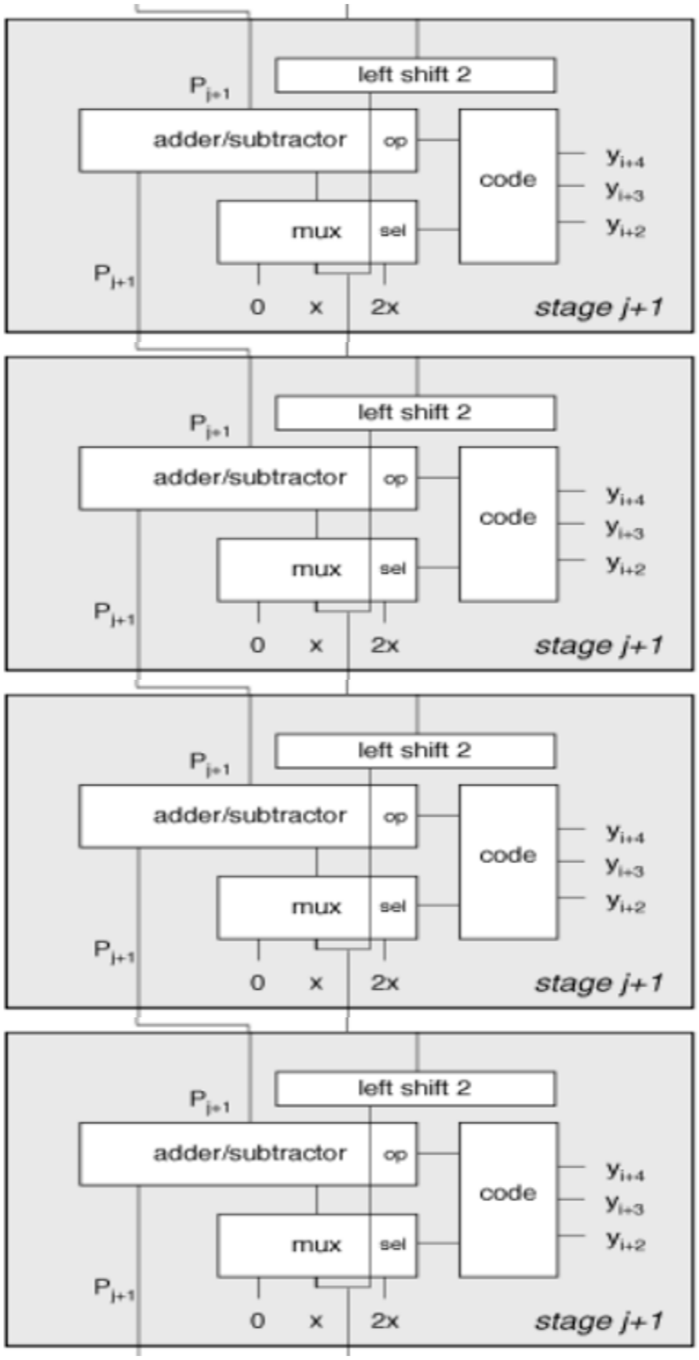
3. [30 pts] Below is a Booth action and architecture stage for Booth multiplication.

y_i y_{i-1} y_{i-2}	increment
0 0 0	0
0 0 1	x
0 1 0	x
0 1 1	2x
1 0 0	-2x
1 0 1	-x
1 1 0	-x
1 1 1	0



a) [15 pts] Use 7-bit 2's complement number system, Booth encoding and algorithm to explain the Booth multiplication of x = 1001101 and y = 1011011. **Note: Show your bit operation step-by-step and find ALL partial products** (in 2's complement).

b) [15 pts] Draw Booth multiplier architecture to implement the Booth multiplication of x = 1001101 and y = 1011011. Show how data (in 2's complement) is propagated and your **bit operation** through every architecture stage. Include (y_i y_{i+1} y_{i+2}) bits, your "mux" output bits, "adder/subtractor" output bits, and "left shift 2" output bits in your architecture.



4. [40 pts]

A) [20 pts] Consider two fixed-number numbers A and B where, A = -3.75 and B = 2.75. **Use 2's complement number, I3.Q2 format, for both A and B.** Calculate the following calculation results.

a) [4 pts] What is your **I3.Q2 format** result for A?

b) [4 pts] What is your **I3.Q2 format** result for B?

c) [4 pts] What is your **I3.Q2 format** result for -B?

d) [4 pts] Calculate A + B and find your final **I4.Q2 format** result. Note: show your bit operation.

e) [4 pts] Calculate A – B and find your final **I4.Q2 format** result. Note: show your bit operation.

B) [20 pts]

Note: In the following (h) ~ (k), you must use the exact multiplicand and multiplier in your calculation. Also, you cannot switch both. A = -3.75 and B = 2.75.

f) [10 pts] Calculate A x B and find your final **I5.Q5 format** result. Note: show your bit operation.

g) [10 pts] Calculate A x (-B) and find your final **I5.Q5 format** result. Note: show your bit operation.

5. [40 pts (including Bonus 20 pts)] A **counter** that counts from **0 to 31** in integer has input ports RSTn, CLK, EN, PL, and DATA. **The output port COUNT receives the value of the counter**. **RSTn** serves as the asynchronous reset, while CLK is the clock input signal. The asynchronous reset **RSTn** is prioritized, activated by verifying if RSTn is '0', and initializing the counter to 0 before considering the rising edge trigger of the clock. If CLK is **rising edge-triggered**, the synchronous behavior of either the **parallel load** or **count model** of the counter is modeled.

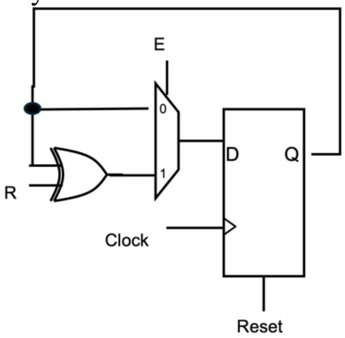
Note: 1) Parallel load (PL) takes precedence over the count enable (EN). PL = '1' is checked first, and EN = '1' is only checked when PL = '1' is not TRUE.
2) When PL is '1', the counter performs the **parallel load**, loaded with the **DATA** value.
3) When PL = '0' and EN = '1', the counter starts the count mode. The counter is triggered to increment its count on the rising edge of the clock signal (**CLK**). **It's important to note that after the counter reaches the value (31), it resets to the initial value (0).**
4) When PL = '0' and EN = '0', the counter holds its previous value.
5) COUNT is declared as an output port; however, VHDL does not allow the output port signal of COUNT to be read. Therefore, a temporary signal **COUNT_VALUE** is declared and used in the process. Finally, COUNT_VALUE is assigned to the output port COUNT outside the process.

```
entity FiveBitCounter is
port (
    RSTn : in std_logic;
    CLK : in std_logic;
    EN : in std_logic;
    PL : in std_logic;
    DATA : in integer;
    COUNT : out integer;
     );
end entity FiveBitCounter;

architecture Behavioral of FiveBitCounter is
variable COUNT_VALUE : integer;

begin
  process(CLK, RSTn)











end architecture Behavioral;
```

[blank]

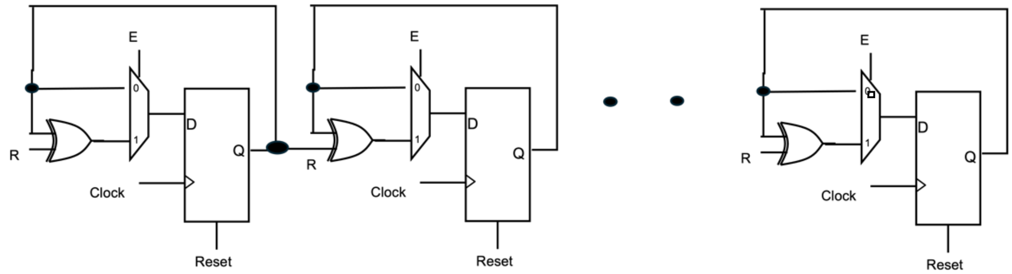
6. [50 pts]

a) [20 pts] A D flip-flop with an **asynchronous reset** input and an **enable** input is given below. Complete the following entity and architecture pair of the D flip-flop. The DFF has an **asynchronous reset** “Reset”, i.e., the output Q = 0 when Reset = **1**. The **Reset** has a higher priority than the **clock**. And, the **clock** has a higher priority than the **enable ‘E’**. The **enable** ‘E’ controls the multiplexer input selection, i.e., E = 0 selects the flip-flop’s output, Q ; E =1 selects R XOR Q; The DFF loads new data only when Reset = 0 and the clock in **rising edge trigger**.



```
  entity rege is
      port (R, Reset, Clock, E: in bit;
              Q: out bit);
  end entity rege;

  ARCHITECTURE Behavior OF rege IS
  BEGIN
      PROCESS (Reset, Clock)
      BEGIN




     END PROCESS ;
 END Behavior ;
```

b) [15 pts] Use "generate" statements to write an entity and architecture pair of VHDL description for a 16-bit wide using 1-bit **rege** as described in (a). Complete the following 16-bit wide using **<u>3 GENERATE</u> statements**.

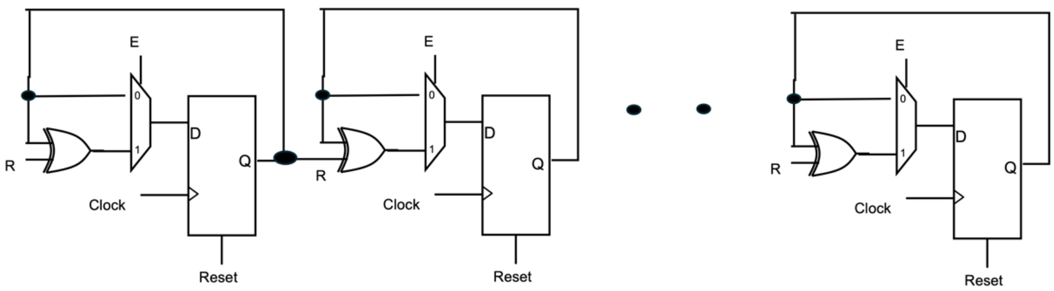


```
entity rege_16 is
      port (D, Reset, E, Clock: in bit; Y: out bit);
end entity rege_16;

architecture reg16 of rege_16 is
-- component declaration
component rege
      port (R, Reset, Clock, E: in bit;
            Q: out bit);
end component rege;
```

b.1) [5 pts] --- **signal declaration <u>(Note: Write your signals on the above 16-bit wide shift registers logic.)</u>**

b.2) [10 pts] --- Use **<u>3 GENERATE</u> statements**
--- Use **port map by <u>POSITION</u>**.

```
begin







end architecture;
```



c) [15 pts] Complete the 16-bit wide shift registers using **only one GENERATE statement**.

```
entity rege_16 is
     port (D, Reset, Clock, E: in bit;
          Q: out bit);
end entity rege_16;

architecture reg16 of rege_16 is

-- component declaration
component rege
     port (R, Reset, Clock, E: in bit;
          Q: out bit);
end component rege;
```

c.1) [5 pts] **--- signal declaration (Note: Write your signals on the above 16-bit wide shift registers logic.)**

c.2) [10 pts] --- Use **one GENERATE statement**
--- Use **port map by POSITION**.

```
begin
```

```
end architecture;
```