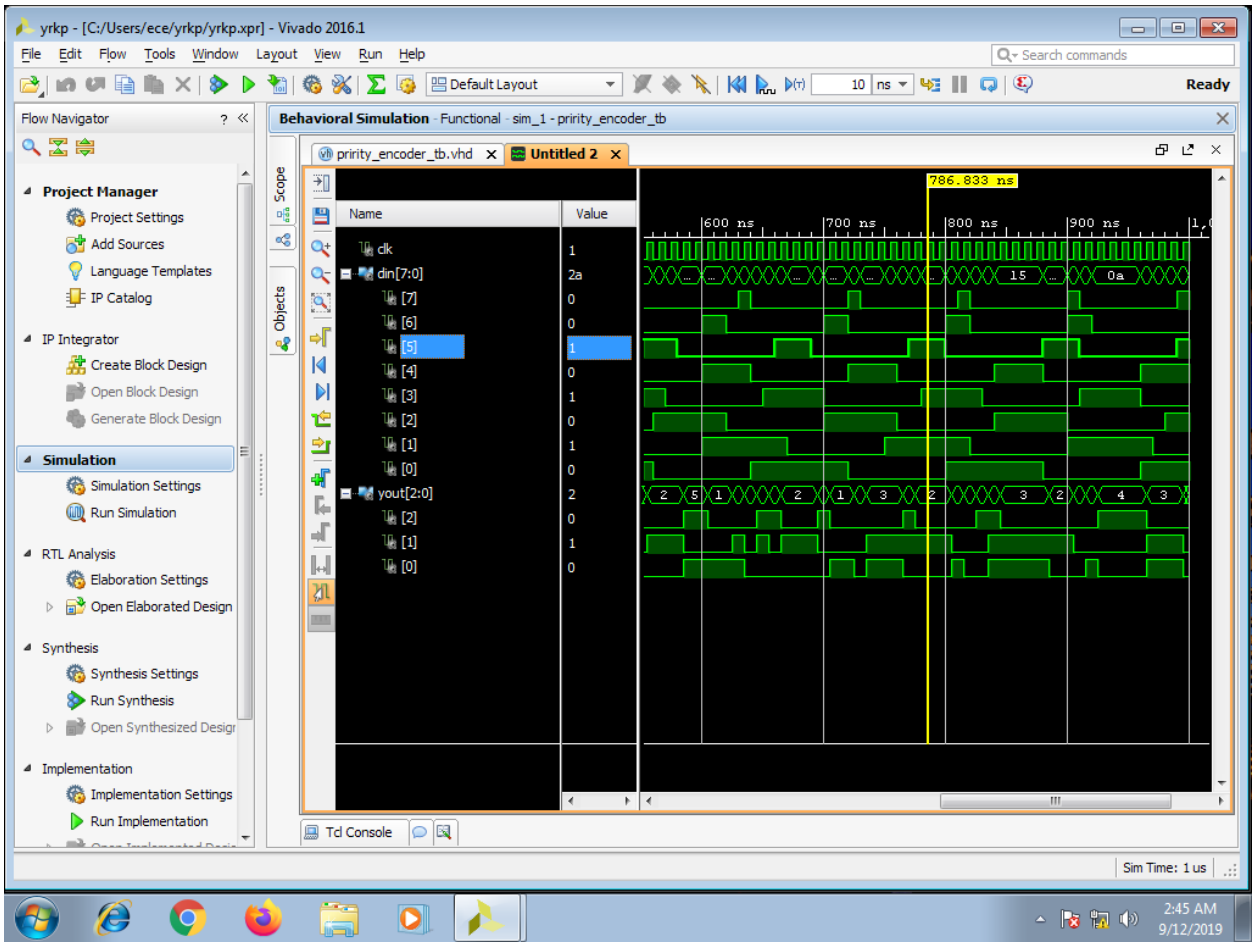
NAME OF THE EXAMINATION: II B.TECH II SEM REGULAR (R-20)
BRANCH: ECE -'A & B' SECTION
NAME OF THE SUBJECT: DICD

1.Priority encoder 3-8 vhdl code

```
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;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity priorityencoder3_8 is
port( clk : in std_logic;
din : in std_logic_vector( 7 downto 0);
yout : out std_logic_vector( 2 downto 0));
end priorityencoder3_8;

architecture Behavioral of priorityencoder3_8 is

begin
process (din,clk)
begin

if (clk'event and clk ='1' and din(7) ='1') then yout <= "000";
elsif (clk'event and clk ='1' and din(6) ='1') then yout <= "001";
elsif (clk'event and clk ='1' and din(5) ='1') then yout <= "010";
elsif (clk'event and clk ='1' and din(4) ='1') then yout <= "011";
elsif (clk'event and clk ='1' and din(3) ='1') then yout <= "100";
elsif (clk'event and clk ='1' and din(2) ='1') then yout <= "101";
elsif (clk'event and clk ='1' and din(1) ='1') then yout <= "110";
elsif (clk'event and clk ='1' and din(0) ='1') then yout <= "111";

end if;
end process;
   end Behavioral;


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;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pririty_encoder_tb is
--  Port ( );
end pririty_encoder_tb;

architecture Behavioral of pririty_encoder_tb is
component priorityencoder3_8 is
port( clk : in std_logic;
din : in std_logic_vector( 7 downto 0);
yout : out std_logic_vector( 2 downto 0));
end component;
signal clk : std_logic;
signal din : std_logic_vector(7 downto 0):="00000000";
signal yout : std_logic_vector(2 downto 0);
begin
p1:priorityencoder3_8 port map (clk => clk,din(0) => din(0) ,din(1) => din(1)  ,din(2) => din(2)
,din(3) => din(3) ,din(4) => din(4) ,din(5) => din(5) ,din(6) => din(6),din(7) => din(7) ,yout(0)
=> yout(0),yout(1) => yout(1),yout(2) => yout(2));

 process
   begin
   clk <= '0'; wait for 5 ns;
   clk <= '1'; wait for 5 ns;
   end process;
     process
     begin
     din(7) <= '1'; wait for 10 ns;
      din(7) <= '0'; wait for 80 ns;
      end process;
      process
         begin
         din(6) <= '1'; wait for 20 ns;
          din(6) <= '0'; wait for 80 ns;
          end process;
          process
             begin
             din(5) <= '1'; wait for 30 ns;
```

```
din(5) <= '0'; wait for 80 ns;
end process;
process
   begin
   din(4) <= '1'; wait for 40 ns;
    din(4) <= '0'; wait for 80 ns;
    end process;
    process
       begin
       din(3) <= '1'; wait for 50 ns;
        din(3) <= '0'; wait for 80 ns;
        end process;
        process
           begin
           din(2) <= '1'; wait for 60 ns;
            din(2) <= '0'; wait for 80 ns;
            end process;
            process
               begin
               din(1) <= '1'; wait for 70 ns;
                din(1) <= '0'; wait for 80 ns;
                end process;
                process
                   begin
                   din(0) <= '1'; wait for 80 ns;
                    din(0) <= '0'; wait for 80 ns;
                    end process;

end Behavioral;
```



2. Verilog source code to realize ALU.

```
module alu(
    input [7:0] A, B,       // 8-bit inputs
    input [3:0] ALU_Sel,    // 4-bit ALU control signal
    output reg [7:0] ALU_Out, // 8-bit output
    output CarryOut         // Carry-out flag
);
    wire [8:0] tmp;
    assign tmp = {1'b0, A} + {1'b0, B};
    assign CarryOut = tmp[8]; // Carry-out flag

    always @(*) begin
        case(ALU_Sel)
            4'b0000: ALU_Out = A + B;          // Addition
            4'b0001: ALU_Out = A - B;          // Subtraction
```

```
        4'b0010: ALU_Out = A * B;          // Multiplication
        4'b0011: ALU_Out = A / B;          // Division
        4'b0100: ALU_Out = A << 1;         // Logical shift left
        4'b0101: ALU_Out = A >> 1;         // Logical shift right
        4'b0110: ALU_Out = {A[6:0], A[7]}; // Rotate left
        4'b0111: ALU_Out = {A[0], A[7:1]}; // Rotate right
        4'b1000: ALU_Out = A & B;          // Bitwise AND
        4'b1001: ALU_Out = A | B;          // Bitwise OR
        4'b1010: ALU_Out = A ^ B;          // Bitwise XOR
        4'b1011: ALU_Out = ~(A | B);       // Bitwise NOR
        4'b1100: ALU_Out = ~(A & B);       // Bitwise NAND
        4'b1101: ALU_Out = ~(A ^ B);       // Bitwise XNOR
        4'b1110: ALU_Out = (A > B) ? 8'd1 : 8'd0; // A > B comparison
        4'b1111: ALU_Out = (A == B) ? 8'd1 : 8'd0; // A == B comparison
        default: ALU_Out = A + B;          // Default: Addition
    endcase
  end
endmodule
```

K. Soumya

Signature of the Faculty