Digital Design Lab Report

Name	Dev Goel
Roll No.	B20CS090
Experiment Number	**V**

Objective

(i) Write a program to implement a Barrel Shifter.
(ii) Write a Program to implement a 32-bit ALU.
(iii) Write a program to implement a 4-line to 2-line priority encoder using
  a. Casex statements
  b. For loop
(iv) Write a behavioural code for implementing
  a. A BCD Adder/Subtractor Unit.
  b. Multiply by 5 circuit.

EXPERIMENT (i): Write a program to implement a Barrel Shifter.

Barrel Shifter:

```
`timescale 1ns / 1ps
module barrel_shifter(inp_w, inp_s, out_y);

input [3:0] inp_w;
input [1:0] inp_s;
output [3:0] out_y;

mux41 m1(inp_w[3], inp_w[0], inp_w[1], inp_w[2], inp_s[0], inp_s[1],
out_y[3]);
mux41 m2(inp_w[2], inp_w[3], inp_w[0], inp_w[1], inp_s[0], inp_s[1],
out_y[2]);
```

```
mux41 m3(inp_w[1], inp_w[2], inp_w[3], inp_w[0], inp_s[0], inp_s[1],
out_y[1]);
mux41 m4(inp_w[0], inp_w[1], inp_w[2], inp_w[3], inp_s[0], inp_s[1],
out_y[0]);

endmodule
```

4*1 Multiplexer:

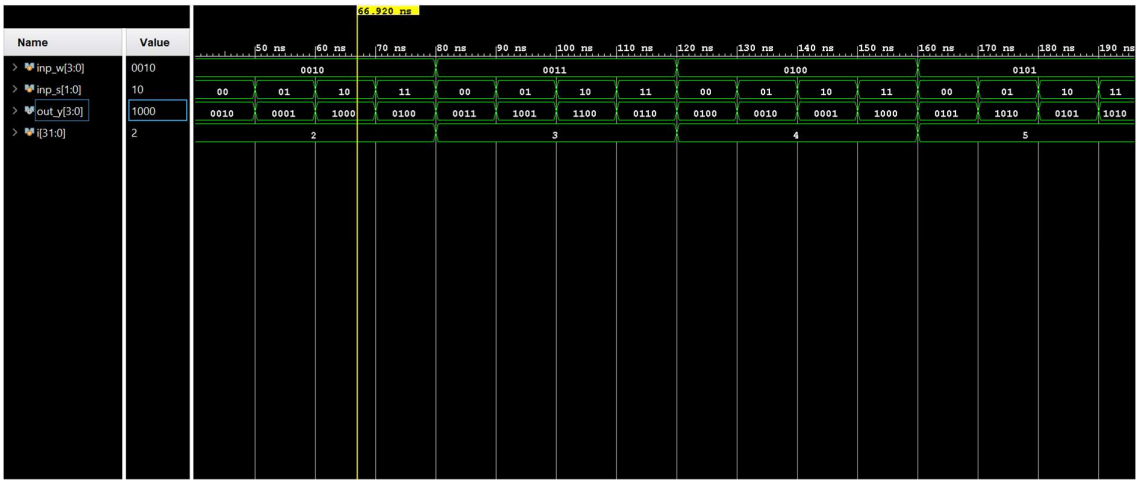
```
`timescale 1ns / 1ps
module mux41(input a, input b, input c, input d, input s0, s1, output out);

assign out = s1 ? (s0 ? d : c) : (s0 ? b : a);

endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_barrel_shifter;

reg [3:0] inp_w;
reg [1:0] inp_s;
wire [3:0] out_y;
integer i;

barrel_shifter bs1(inp_w, inp_s, out_y);

initial
   begin
      for(i = 1; i < 16; i=i+1)
      begin
         assign inp_w = i;
         inp_s[1] = 0;
         inp_s[0] = 0; #10;
         inp_s[1] = 0;
         inp_s[0] = 1; #10;
```

```
            inp_s[1] = 1;
            inp_s[0] = 0; #10;
            inp_s[1] = 1;
            inp_s[0] = 1; #10;
        end
    end
initial #600 $finish;

endmodule
```

Waveform:



EXPERIMENT (ii): Write a Program to implement a 32-bit ALU.

32-bit ALU:

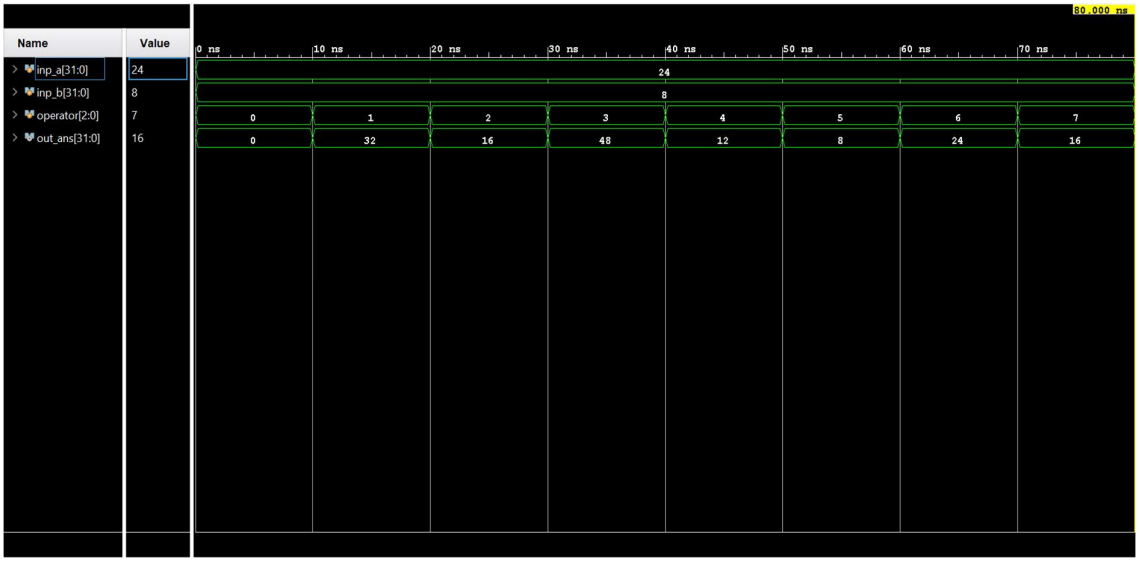
```
`timescale 1ns / 1ps
module alu(inp_a, inp_b, operator, out_ans);


input [31:0] inp_a, inp_b;

input [2:0] operator;

output reg [31:0] out_ans;


always @*

   begin

      case(operator)

         // Clear

         0: out_ans = 0;

         // Addition

         1: out_ans = inp_a + inp_b;

         // Subtraction

         2: out_ans = inp_a - inp_b;

         // Left shift

         3: out_ans = inp_a << 1;

         // Right Shift

         4: out_ans = inp_a >> 1;

         // A AND B

         5: out_ans = inp_a & inp_b;
```

```
        // A OR B
        6: out_ans = inp_a | inp_b;
        // A XOR B
        7: out_ans = inp_a ^ inp_b;
      endcase
   end
endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_alu;

reg [31:0] inp_a, inp_b;
reg [2:0] operator;
wire [31:0] out_ans;

alu alu1(inp_a, inp_b, operator, out_ans);
initial
   begin
      inp_a = 24;
      inp_b = 8;
      operator = 0; #10;
      operator = 1; #10;
      operator = 2; #10;
      operator = 3; #10;
      operator = 4; #10;
      operator = 5; #10;
      operator = 6; #10;
      operator = 7; #10;
   end

initial #80 $finish;
endmodule
```

Waveform:



EXPERIMENT (iii) a: Write a program to implement a 4-line to 2-line priority encoder using casex statements

- **Priority Encoder (Using Casex Statements):**

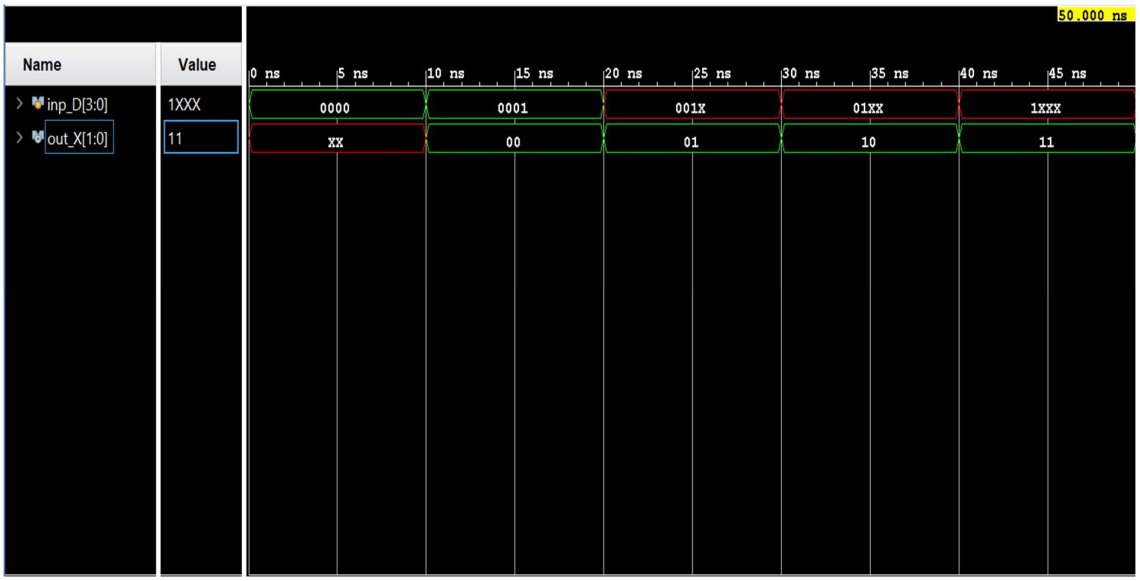
```
`timescale 1ns / 1ps
module priority_encoder_casex(inp_D, out_X);

input [3:0] inp_D;
output reg [1:0] out_X;
reg V;

always@(inp_D)
   begin
      V = 1;
      case(inp_D)
      4'b1XXX: out_X = 2'b11;
      4'b01XX: out_X = 2'b10;
```

```
        4'b001X: out_X = 2'b01;
        4'b0001: out_X = 2'b00;
        default:
           begin V = 0;
           out_X = 2'bXX;
           end
        endcase
    end
endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_priority_encoder_casex;

reg [3:0] inp_D;
wire [1:0] out_X;

priority_encoder_casex pec(inp_D, out_X);

initial
    begin
        inp_D = 4'b0000; #10;
        inp_D = 4'b0001; #10;
        inp_D = 4'b001X; #10;
        inp_D = 4'b01XX; #10;
        inp_D = 4'b1XXX; #10;
    end

initial #50 $finish;
endmodule
```

Waveform:



EXPERIMENT (iii) b: Write a program to implement a 4-line to 2-line priority encoder using for loop.

- **Priority Encoder (Using for loop):**

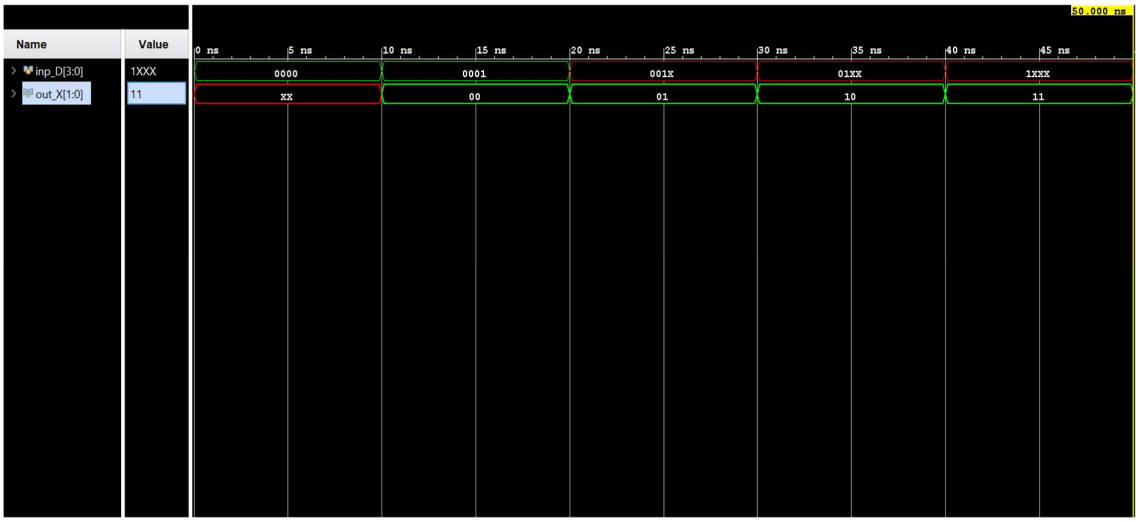
```
`timescale 1ns / 1ps
module priority_encoder_for_loop(inp_D, out_X);

output reg [1:0] out_X;
input [3:0] inp_D;
reg V;
integer i;

always@(inp_D)
   begin
      out_X = 2'bXX;
      V = 0;
```

```
    for(i = 0; i < 4; i = i + 1)
      begin
        if(inp_D[i])
          begin
            out_X = i;
            V = 1;
          end
      end
  end
endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_priority_encoder_for_loop;

reg [3:0] inp_D;
wire [1:0] out_X;

priority_encoder_for_loop pefl(inp_D, out_X);

initial
  begin
    inp_D = 4'b0000; #10;
    inp_D = 4'b0001; #10;
    inp_D = 4'b001X; #10;
    inp_D = 4'b01XX; #10;
    inp_D = 4'b1XXX; #10;
  end

initial #50 $finish;
endmodule
```

Waveform:



EXPERIMENT (iv) a: Write a behavioural code for implementing a BCD Adder/Subtractor Unit.

- **BCD Adder:**

```
`timescale 1ns / 1ps
module bcd_adder(inp_a, inp_b, carry_in, sum, carry);

input [3:0] inp_a, inp_b;
input carry_in;
output [3:0] sum;
output carry;
reg [4:0] sum_temp;
reg [3:0] sum;
reg carry;


always @(inp_a, inp_b, carry_in)
begin
   sum_temp = inp_a + inp_b + carry_in;
   if(sum_temp > 9)
```

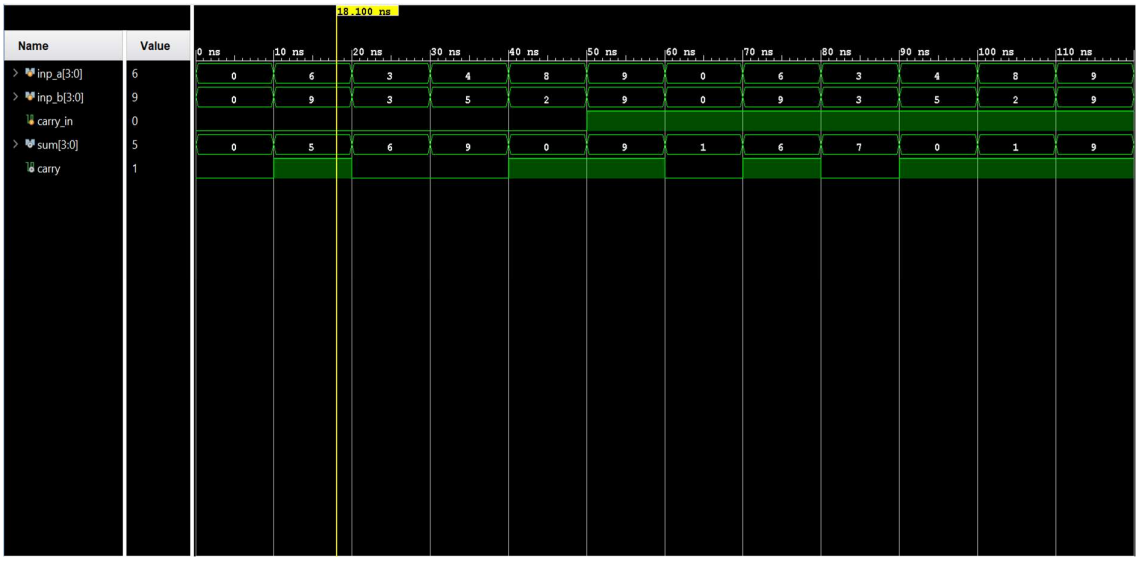
```
        begin
            sum_temp = sum_temp+6;
            carry = 1;
            sum = sum_temp[3:0];
        end
    else
        begin
            carry = 0;
            sum = sum_temp[3:0];
        end
end

endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_bcd_adder;

reg [3:0] inp_a;
reg [3:0] inp_b;
reg carry_in;
wire [3:0] sum;
wire carry;

bcd_adder bcda(inp_a, inp_b, carry_in, sum, carry);

initial
begin
    inp_a = 0;  inp_b = 0;  carry_in = 0;   #10;
    inp_a = 6;  inp_b = 9; #10;
    inp_a = 3;  inp_b = 3; #10;
    inp_a = 4;  inp_b = 5; #10;
    inp_a = 8;  inp_b = 2; #10;
    inp_a = 9;  inp_b = 9;  carry_in = 1;   #10;
    inp_a = 0;  inp_b = 0; #10;
    inp_a = 6;  inp_b = 9; #10;
```

```
    inp_a = 3;  inp_b = 3; #10;
    inp_a = 4;  inp_b = 5; #10;
    inp_a = 8;  inp_b = 2; #10;
    inp_a = 9;  inp_b = 9; #10;
end

initial #120 $finish;
endmodule
```

Waveform:



- **BCD Subtractor:**

```
`timescale 1ns / 1ps
module bcd_subtractor(inp_a, inp_b, diff, carry);

input [3:0] inp_a, inp_b;
output [3:0] diff;
output carry;
reg [4:0] diff_temp;
reg [3:0] diff;
reg carry;
```

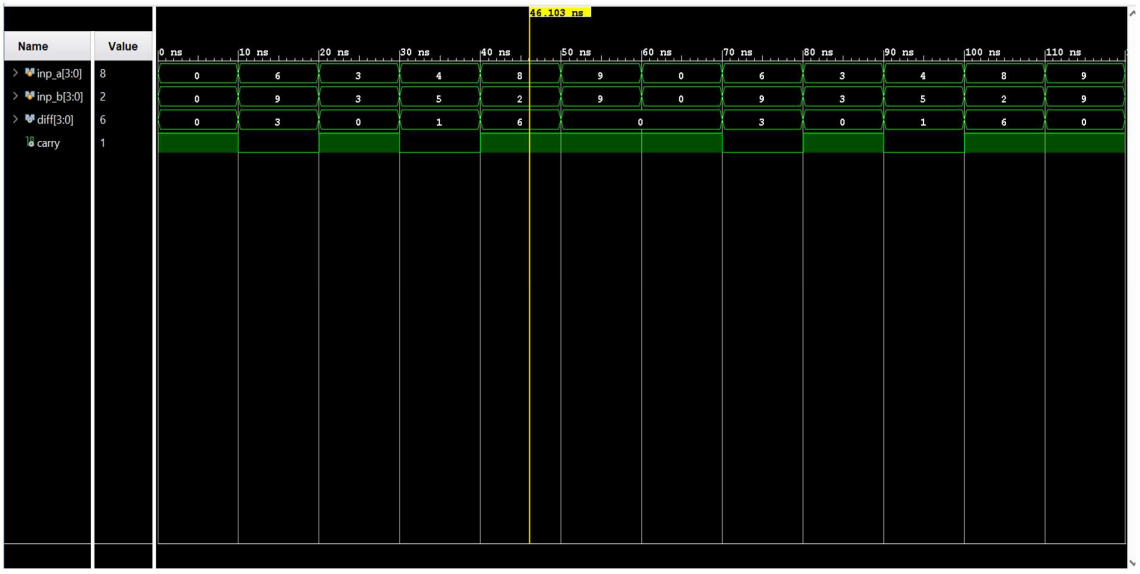
```
always @(inp_a, inp_b)
begin
   diff_temp = inp_a + 10 - inp_b;
   if(diff_temp > 9)
      begin
         diff_temp = diff_temp+6;
         carry = 1;
         diff = diff_temp[3:0];
      end
   else
      begin
         carry = 0;
         diff = diff_temp[3:0];
         diff = 10-diff;
      end
end
endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_bcd_subtractor;


reg [3:0] inp_a;

reg [3:0] inp_b;

wire [3:0] diff;

wire carry;


bcd_subtractor bcds(inp_a, inp_b, diff, carry);

initial

begin

   inp_a = 0;  inp_b = 0; #10;
```

```
    inp_a = 6;  inp_b = 9; #10;
    inp_a = 3;  inp_b = 3; #10;
    inp_a = 4;  inp_b = 5; #10;
    inp_a = 8;  inp_b = 2; #10;
    inp_a = 9;  inp_b = 9; #10;
    inp_a = 0;  inp_b = 0; #10;
    inp_a = 6;  inp_b = 9; #10;
    inp_a = 3;  inp_b = 3; #10;
    inp_a = 4;  inp_b = 5; #10;
    inp_a = 8;  inp_b = 2; #10;
    inp_a = 9;  inp_b = 9; #10;
end
initial #120 $finish;
endmodule
```

Waveform:



EXPERIMENT (iv) b: Write a behavioural code for implementing a Multiply by 5 circuit.

Multiply by 5 circuit:

```
`timescale 1ns / 1ps
module multiplication_by_5(inp_a, out_b);


input [3:0] inp_a;

output reg[6:0] out_b;

reg [5:0] sum1;

reg [3:0] sum2;


always@*

   begin

      assign sum2 = inp_a;

      assign sum1 = inp_a << 2;

      assign out_b = sum1 + sum2;

   end

endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_multiplication_by_5;

reg [3:0] inp_a;
wire [6:0] out_b;
integer i;
```

```
multiplication_by_5 mb5(inp_a, out_b);

initial
   begin
      for(i = 0; i < 16; i=i+1)
      begin
         assign inp_a = i; #10;
      end
   end

initial #160 $finish;
endmodule
```

Waveform: