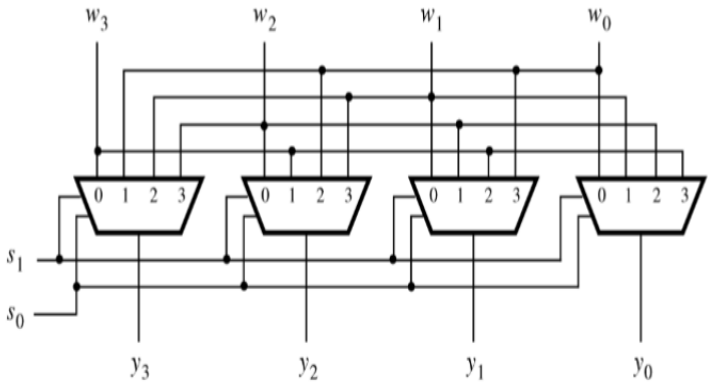
CSL2070 Digital Design

Lab Report

Roll Number	**B20CS029**
Name	**Kulkarni Tanmay Shreevallabh**
Lab number	**Lab5**
Experiment Title	• Barrel Shifter • 32 Bit ALU • 4x2 Priority Encoder using a) Casex Statements b) For loop • Behavioral Code for implementing a) BCD adder and subtractor b) Multiply by 5 circuit

Barrel Shifter

Circuit Diagram



Code

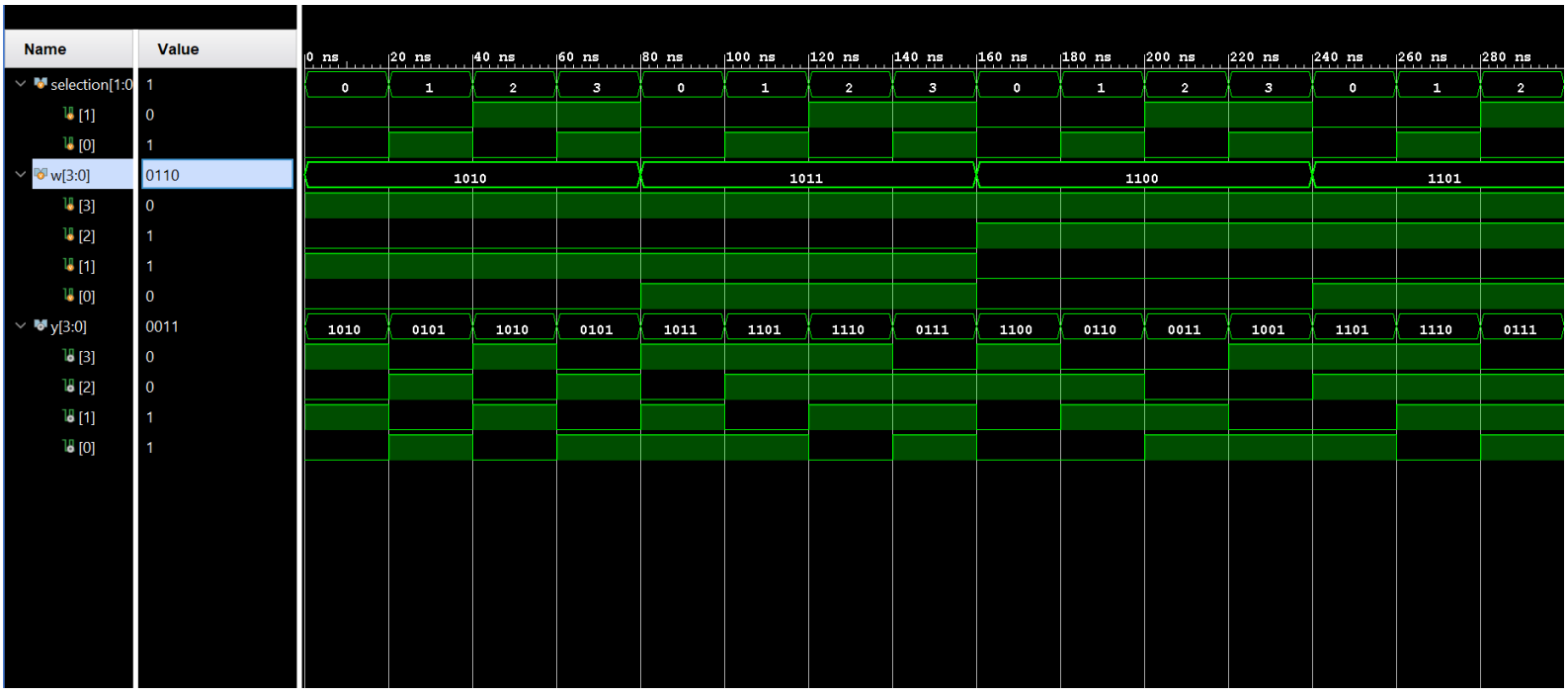
```
module barrel_shifter(selection,w,y);
input [1:0]selection;
input [3:0]w;
output [3:0]y;
assign y[3]=(!selection[0])&(!selection[1])&(w[3])|(selection[0])&(!selection[1])&(w[0])|(!selection[0])&(selection[1])&(w[1])|(selection[0])&(selection[1])&(w[2]),
assign y[2]=(!selection[0])&(!selection[1])&(w[2])|(selection[0])&(!selection[1])&(w[3])|(!selection[0])&(selection[1])&(w[0])|(selection[0])&(selection[1])&(w[1]),
assign y[1]=(!selection[0])&(!selection[1])&(w[1])|(selection[0])&(!selection[1])&(w[2])|(!selection[0])&(selection[1])&(w[3])|(selection[0])&(selection[1])&(w[0]),
assign y[0]=(!selection[0])&(!selection[1])&(w[0])|(selection[0])&(!selection[1])&(w[1])|(!selection[0])&(selection[1])&(w[2])|(selection[0])&(selection[1])&(w[3]),
endmodule
```

Code for Test Bench

```
module test_barrel_shifter;
reg [1:0]selection;
reg [3:0]w;
wire [3:0]y;

barrel_shifter(selection,w,y);
initial
    begin
        w = 4'b1010;
        selection = 2'b00;
    end
always #20 selection = selection + 1'b1;
always #80 w = w + 1'b1;
initial #1000 $finish;
endmodule
```

Waveform



32 Bit ALU

Circuit Diagram

Operation	Input	32 bit Output
Clear	000	0
Addition	001	A + B
Subtraction	010	A - B
A * 2	011	Left shift
A / 2	100	Right Shift
A AND B	101	A & B
A OR B	110	A | B
A XOR B	111	A^B

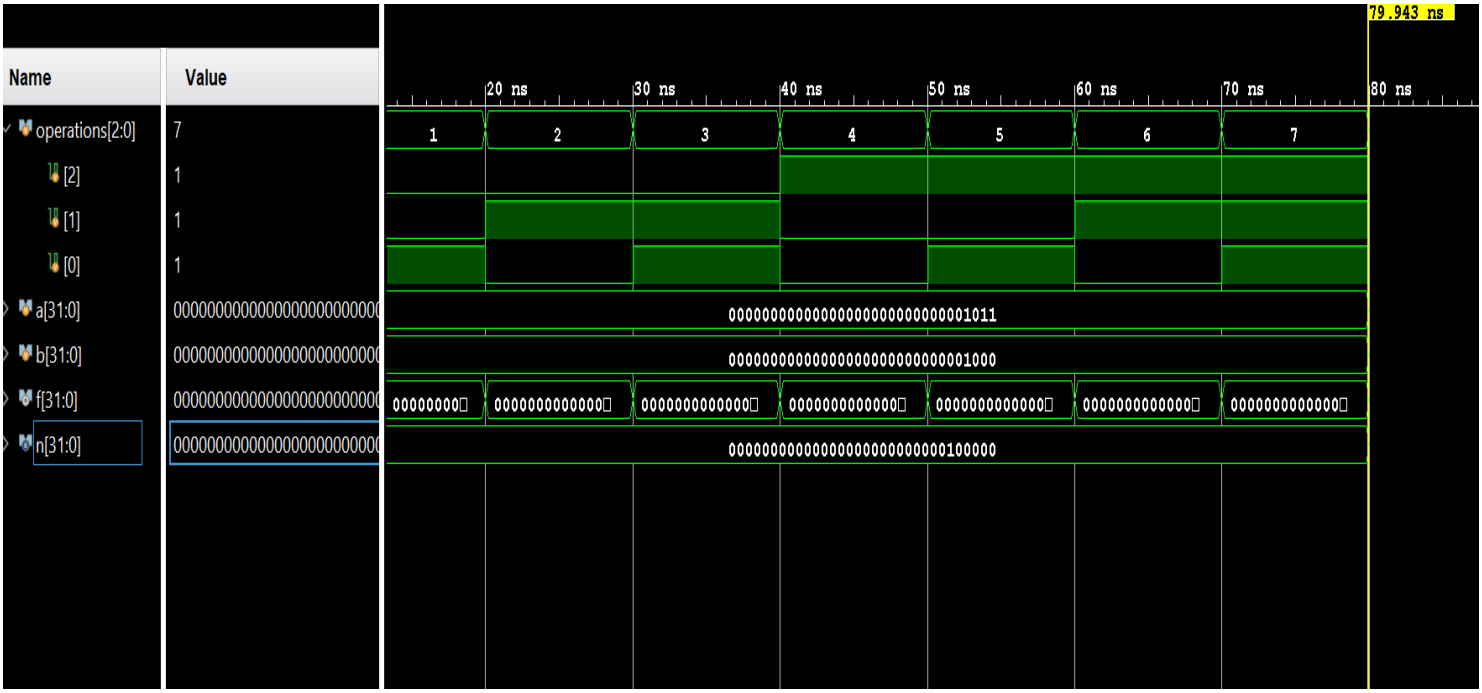
Code

```
module alu(a,b,operation,f);
parameter n = 32;
input [n-1:0]a,b;
output reg [n-1:0]f;
input [2:0] operation;
always @ (operation)
    begin
    case(operation)
    0:f=0;
    1:f=a+b;
    2:f=a-b;
    3:f=a<<1;
    4:f=a>>1;
    5:f=a&b;
    6:f=a|b;
    7:f=a^b;
    endcase
    end
endmodule
```

Code for Test- Bench

```
module test_alu;
parameter n = 32;
reg [2:0] operations;
reg [n-1:0] a, b;
wire [n-1:0] f;
alu A(a, b, operations, f);
initial
    begin
    a=32'b00000000000000000000000000001011; // a=11
    b=32'b00000000000000000000000000001000; // b=8
    operations=3'b000;
    end
    always #10 operations=operations+1'b1;
initial #80 $finish;
endmodule
```

Waveform



Behavioral Code for Implementing a) Adder and Subtractor unit

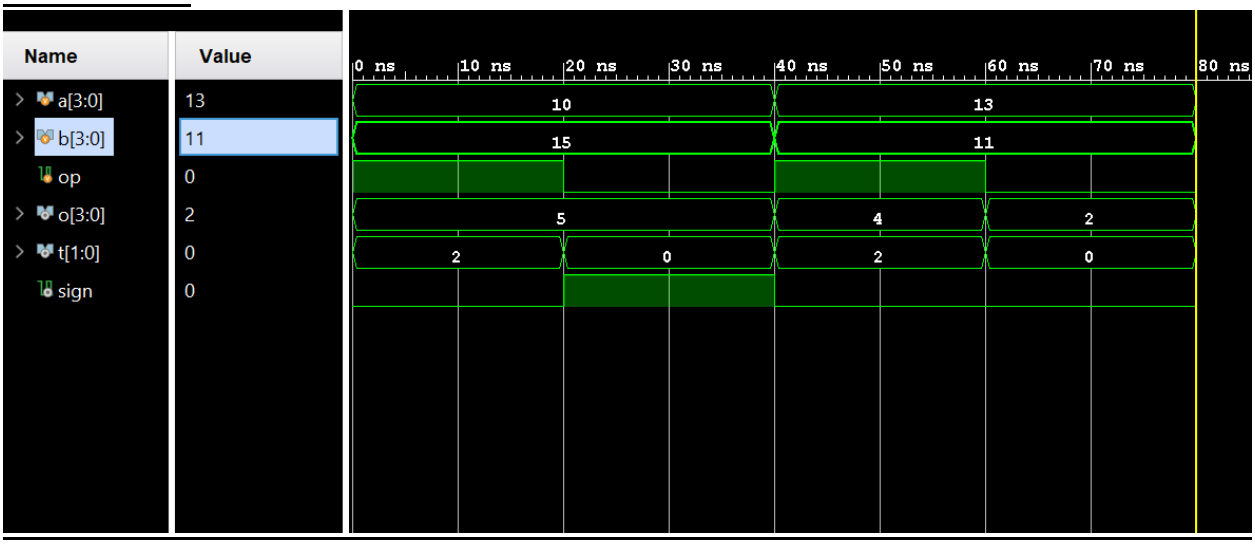
Code

```
module bcd_adder_subtractor(a, b, op, t, o , sign );
// op=1 stands for addition and op=0 stands for subtraction
// sign=0 means positive and sign=1 means negative
input [3:0] a, b;
input op;
reg [4:0] f;
output [3:0] o;
output [1:0] t;
output reg sign;
always @ (op)
begin
if (op)
  begin
  sign=0;
  f=a+b;
  end
else
  begin
  if (a>=b)
  begin
  sign=0;
  f=a-b;
  end
  else
  begin
  sign=1;
  f=b-a;
  end
  end
end
bcd_conv bc(f,t,o);
endmodule
```

Code for Test- Bench

```
module test_bcd_adder_subtractor();
reg [3:0] a, b;
reg op;
wire [3:0] o;
wire [1:0] t;
wire sign;
bcd_adder_subtractor bas(a,b,op,t,o,sign);
initial
begin
a=4'b1010;
b=4'b1111;
op=1'b1;
#20
op=1'b0;
#20
a=4'b1101;
b=4'b1011;
op=1'b1;
#20
op=1'b0;
end
initial #80 $finish;
endmodule
```

Waveform


Name
Value
a[3:0]
13
b[3:0]
11
op
0
o[3:0]
2
t[1:0]
0
sign
0
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
10
13
15
11
5
4
2
2
0
2
0

b) FMultiplierX5

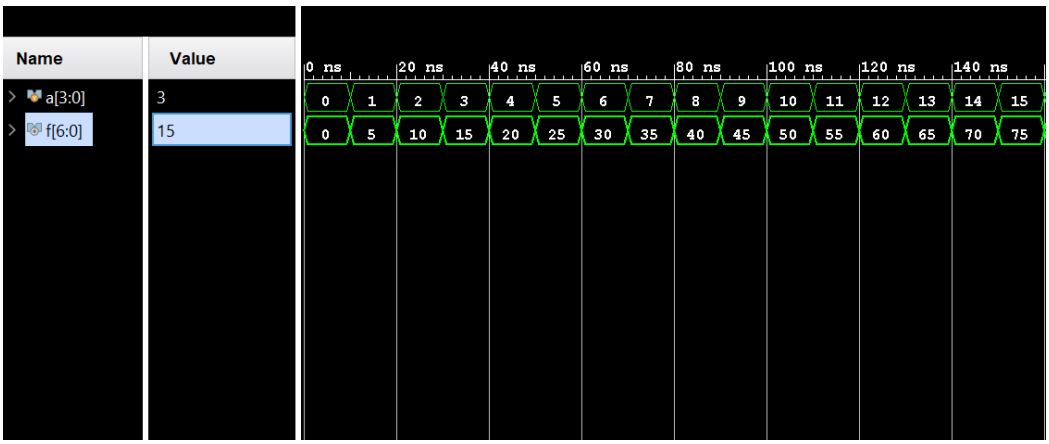
Code

```
module multiplierX5(a, f);
input [3:0] a;
output reg [6:0] f;
always @ (a)
begin
f=((a<<1)<<1)+a;
end
endmodule
```

Code for Test- Bench

```
module test_multiplierX5();
reg [3:0] a;
wire [6:0] f;
multiplierX5 m(a,f);
initial
begin
a=4'b0000;
end
always #10 a=a+1'b1;
initial #1000 $finish;
endmodule
```

Waveform



Priority Encoders
b) Casex Statements

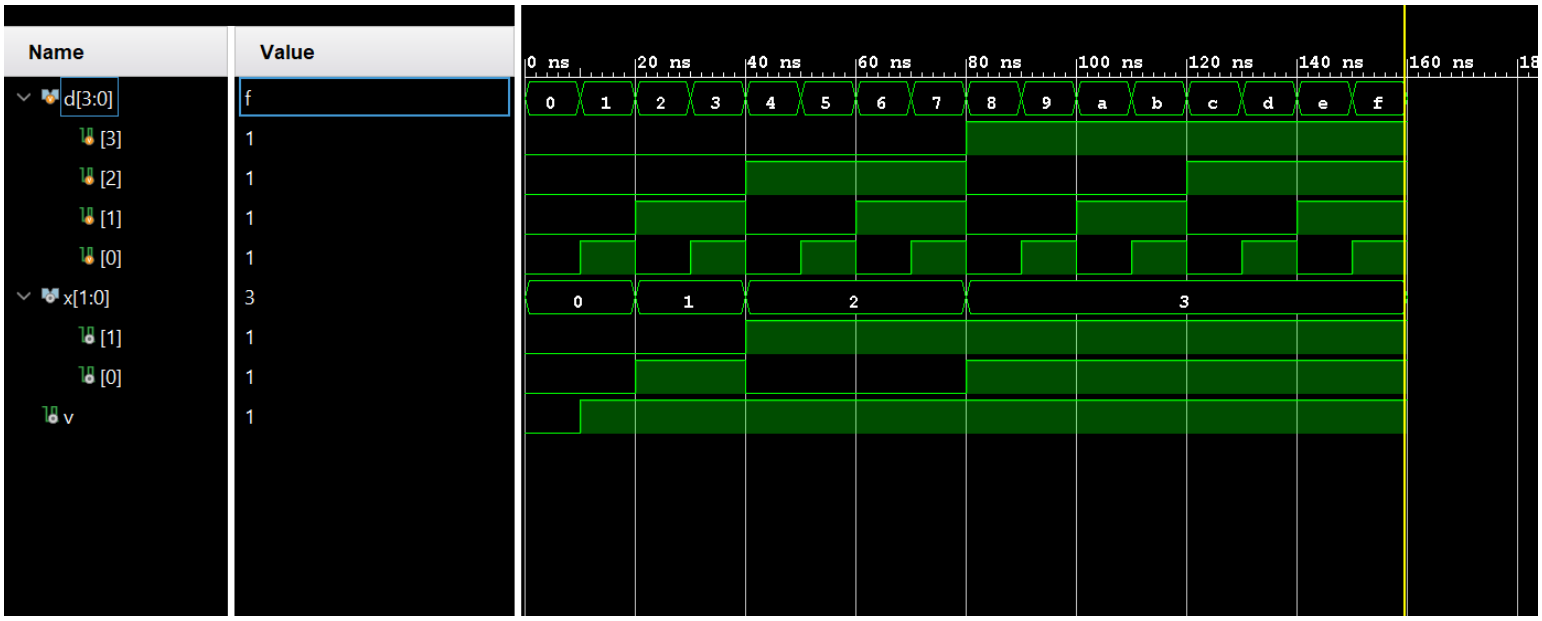
Code

```
module priority_encoder_a(d,x,v);
input [3:0]d;
output reg[1:0]x;
output reg v;
always @(d)
    begin
    v=1;
    casex(d)
    4'b1XXX: x=2'b11;
    4'b01XX: x=2'b10;
    4'b001X: x=2'b01;
    4'b0001: x=2'b00;
    default:
    begin
    v=0;
    x=2'b00;
    end
  endcase
 end
endmodule
```

Code for Test- Bench

```
module test_priority_encoder_a;
reg [3:0] d;
wire [1:0] x;
wire v;
priority_encoder_a pea(d, x, v);
initial
begin
d=4'b0000;
end
always #10 d=d+1'b1;
initial #160 $finish;
endmodule
```

Waveform



Priority Encoders

a) For Loop

Code

```
module priority_encoder_b(d,x,v);
input [3:0] d;
output reg [1:0] x;
output reg v;
integer k;
always @ (d)
   begin
   x=2'bXX;
   v=0;
   for (k=0; k<4; k=k+1)
     if (d[k])
      begin
      x=k;
      v=1;
      end
   end
endmodule
```

Code for Test- Bench

```
module test_priority_encoder_b;
reg [3:0] d;
wire [1:0] x;
wire v;
priority_encoder_b peb(d, x, v);
initial
begin
d=4'b0000;
end
always #10 d=d+1'b1;
initial #160 $finish;
endmodule
```

Waveform