Lab 5
Abu Shahid
(B20CS003)

Job 1: Barrel Shifter

This is a bit shifter used to shift the bit array by rolling it like a barrel. We implement a 4 bit barrel shifter using four 4x1 multiplexers.

Code:

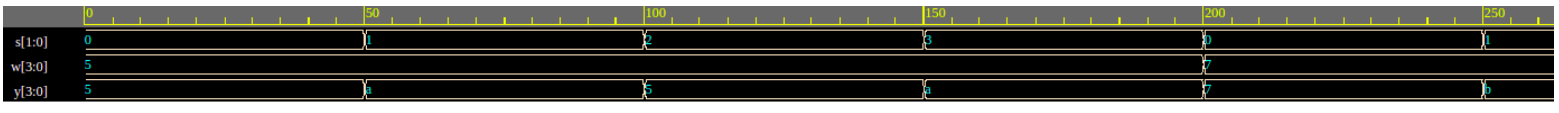
```
//mux code start
module mux(i1,i2,i3,i4,op,control);
input i1,i2,i3,i4;
input [1:0] control;
output reg op;
always @*
begin
case(control)
2'b00: op = i1;
2'b01: op = i2;
2'b10: op = i3;
2'b11: op = i4;
default : op=1'bz;
endcase
end
endmodule
//mux code ends

//barrel code starts
module barrel_shifter(w,s,y);
input [3:0] w;
input [1:0] s;
output [3:0] y;
mux m1(w[3],w[0],w[1],w[2],y[3],s);
mux m2(w[2],w[3],w[0],w[1],y[2],s);
mux m3(w[1],w[2],w[3],w[0],y[1],s);
mux m4(w[0],w[1],w[2],w[3],y[0],s);
endmodule
//barrel code ends
```

Code for testbench:

```
module test_barrel_shifter;
wire [3:0] y;
reg [3:0] w;
reg [1:0] s;
barrel_shifter bs(w,s,y);
initial
begin
$dumpfile("dump.vcd"); $dumpvars;
w=5;s=0;
#50 w=5; s = 1;
#50 w=5; s = 2;
#50 w=5; s = 3;
#50 w=7; s = 0;
#50 w=7; s = 1;
#50 w=7; s = 2;
#50 w=7; s = 3;
end
endmodule
```

Waveform:



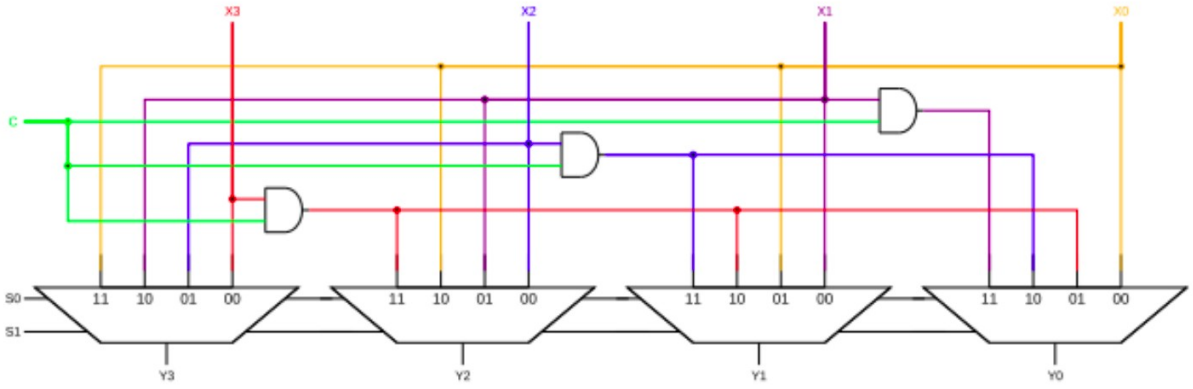
ote: To revert to EPWave opening in a new browser window, set that option on your user page.

Explaination:

5(10)--> 0101(2)
doing 5 shift 1----> 1010 which is equivalent 2 a
doing 5 shift 2----> 0101 which is equal to 5

The same has been demonstrated by our waveform which validates the functioning of the circuit.

Diagram:



Job 2: 32-bit ALU

We use behavioral code to a 32 bit ALU with a total of 8 operations

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:

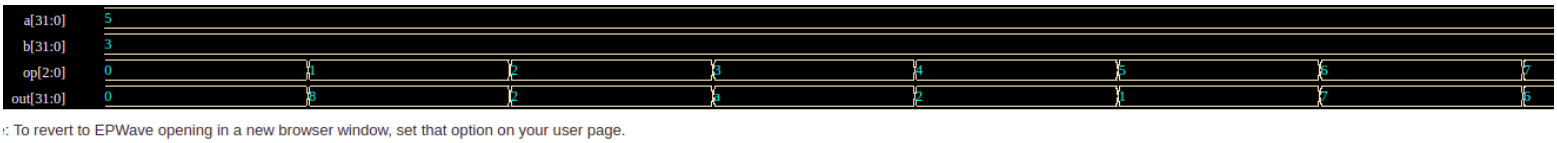
```
module alu(a,b,op,out);
input [31:0] a,b;
input [2:0] op;
output reg [31:0] out;
always @*
begin
case(op)
0: out = 0;
1: out = a+b;
2: out = a-b;
3: out = a << 1;
4: out = a >> 1;
5: out = a & b;
6: out = a|b;
7: out = a^b;
endcase
end
endmodule
```

Code for testbench:

```
module test_alu;
wire [31:0] out;
reg [31:0] a,b;
reg [2:0] op;
alu uut(a,b,op,out);
initial
begin
$dumpfile("dump.vcd"); $dumpvars;
a = 5; b = 3; op = 0;
#50 a = 5; b = 3; op = 1;
#50 a = 5; b = 3; op = 2;
#50 a = 5; b = 3; op = 3;
#50 a = 5; b = 3; op = 4;
#50 a = 5; b = 3; op = 5;
#50 a = 5; b = 3; op = 6;
#50 a = 5; b = 3; op = 7;
end
initial #500 $finish;
endmodule

```

Waveform:



: To revert to EPWave opening in a new browser window, set that option on your user page.

*as it is behavioural code, the validation of the circuit(with output of correct results self-explains the code)

Job 3: 4-to-2 Priority Encoder

We make a 4 bit priority encoder for encoding 4 lines into 2 lines using 2 methods

1) CaseX statements
2) For loop

Truth Table:

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

a) CaseX Statement:

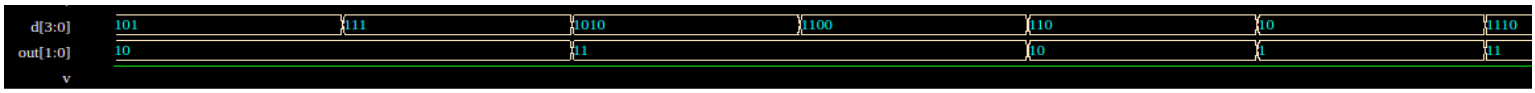
Code:

```
module four_to_two_casex(d,out,v);
input [3:0] d;
output reg v;
output reg [1:0] out;
always@(d)
begin
casex(d)
4'b1xxx: out = 2'b11;
4'b01xx: out = 2'b10;
4'b001x: out = 2'b01;
4'b0001: out = 2'b00;
default: out = 2'bzz;
endcase
if(d==4'b0000)
begin
assign v = 1'b0;
end
else
begin
assign v = 1'b1;
end
end
endmodule
```

Code for Testbench:

```
module test_four_to_two_casex;
wire [1:0]out;
wire v;
reg [3:0] d;
four_to_two_casex uut(d,out,v);
initial
begin
$dumpfile("dump.vcd"); $dumpvars
d=5;
#10 d = 7;
#10 d = 10;
#10 d = 12;
#10 d = 6;
#10 d = 2;
#10 d = 14;
end
initial #100 $finish;
endmodule
```

Waveform:



e: To revert to EPWave opening in a new browser window, set that option on your user page.

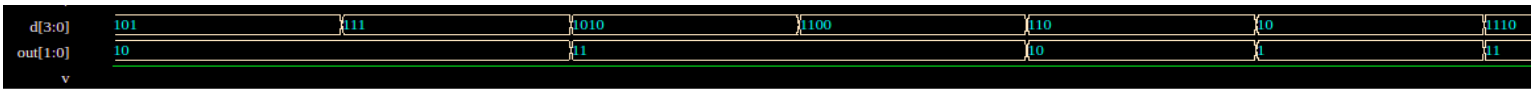
b) For loop

Code:

```
module four_to_two_for(d,out,v);
input [3:0] d;
output reg v;
output reg [1:0] out;
integer k;
always @(d)
begin
out=2'bxx;
v = 1'b0;
for(k = 0;k <4;k= k+1)
begin
if (d[k])
begin
out=k;
v=1'b1;
end
end
end
endmodule
```

Code for testbench:

```
module test_four_to_two_for;
wire [1:0]out;
wire v;
reg [3:0] d;
four_to_two_for uut(d,out,v);
initial
begin
d=5;
#10 d = 7;
#10 d = 10;
#10 d = 12;
#10 d = 6;
#10 d = 2;
#10 d = 14;
end
initial #100 $finish;
endmodule
```



e: To revert to EPWave opening in a new browser window, set that option on your user page.

Job 4: a) BCD Adder-Subtractor

A BCD adder or subtractor for 4 bits that performs addition or subtraction depending on the operator input. We use behavioral coding.

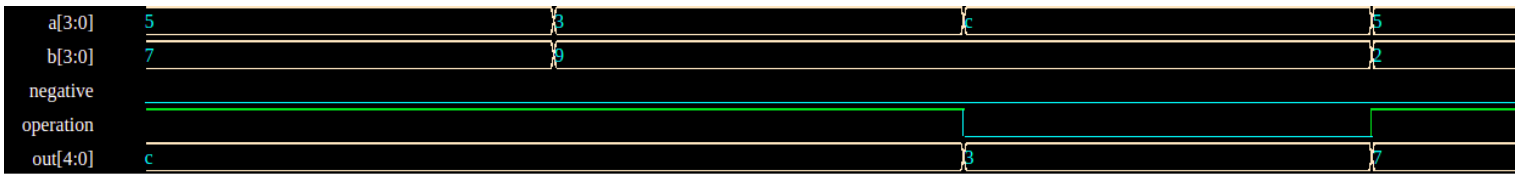
Code:

```
module bcd_adder_substractor(a,b,operation,out,negative);
input [3:0] a,b;
input operation;
output reg [4:0] out;
output reg negative;
always @(operation)
begin
if(operation)
begin
out = a+b;
negative = 0;
end
else
begin
if(a>=b)
begin
out = a-b;
negative = 0;
end
else
begin
out = b-a;
negative = 1;
end
end
end
endmodule
```

Code for testbench:

```
module test_bcd_adder_substractor;
wire [4:0] out;
wire negative;
reg [3:0] a,b;
reg operation;
bcd_adder_substractor uut(a,b,operation,out,negative);
initial
begin
$dumpfile("dump.vcd"); $dumpvars;
a = 5;b=7;operation = 1;
#20 a = 3;b=9;operation = 1;
#20 a = 12;b=9;operation = 0;
#20 a = 5; b = 2; operation = 1;
end
initial #100 $finish;
endmodule
```

Waveform:



e: To revert to EPWave opening in a new browser window, set that option on your user page.

*correct output validates our circuit

Job 4: b) 5 Multiplier

A behavioral code to multiply a 4-bit number by 5.

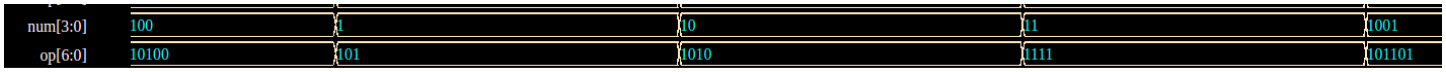
Code:

```
module multiplier(num,op);
input [3:0] num;
output [6:0] op;
assign op = ((num<<1)<<1) + num;
endmodule
```

Code for testbench:

```
module test_multiplier;
reg [3:0] num;
wire [6:0] op;
multiplier uut(num,op);
initial
begin
$dumpfile("dump.vcd"); $dumpvars;
num = 5;
#10 num = 6;
#10 num = 4;
#10 num = 1;
#10 num = 2;
#10 num = 3;
#10 num = 9;
end
initial #100 $finish;
endmodule
```

Waveform:



lote: To revert to EPWave opening in a new browser window, set that option on your user page.

*correct code validates the working of our circuit.