Lab 1: Barrel Shifter

Barrel Shifter
- A input
- y output
- Shift Amount

Parameterization
- instead of 8 bit we have 2^N bits
- amt should be N
- instead of 3 stages should be N stages
- for parameters we need to look at generate, for FOR loops

First is Left shifter
- Test Bench
- also test bench left shifter and right shifter at same time

2nd is right shifter with left right control and a multiplexer to show desired output
- Show on FPGA

3rd is left/right generic shifter where you dont use mux but you use two reversers
- Show on FPGA

We are Building a Structural Design, not behavioral

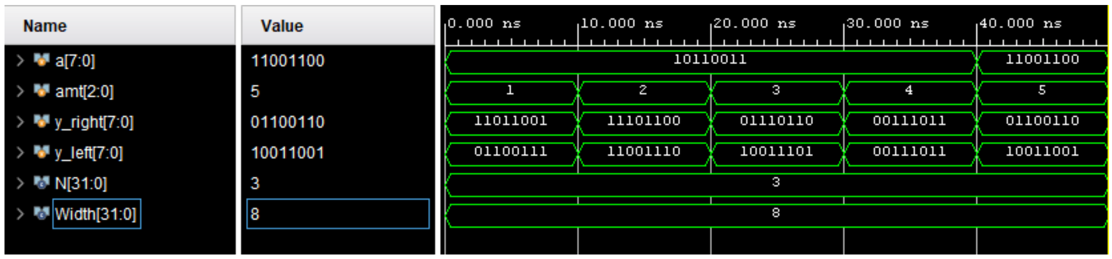
Part 1

Solution #2	Parameterized Barrel Shifter Module	Testbench

```
`timescale 1ns / 1ps

module param_right_shifter
    #(parameter N = 3, parameter Width = 2**N)
    (
        input  logic [Width-1:0] a,
        input  logic [N-1:0] amt,
        output logic [Width-1:0] y
    );
     logic [Width-1:0] s [0:N];
     assign s[0] = a;

    generate
        genvar i;
               for (i = 0; i < N ; i = i + 1)
         begin
        assign s[i] = amt[i] ? s[i] >> 2**i :s[i];
         end
    endgenerate
    assign y = s[N-1];

endmodule

module param_left_shifter
    #(parameter N = 3, parameter Width = 2**N)
    (
        input  logic [Width-1:0] a,
        input  logic [N-1:0] amt,
        output logic [Width-1:0] y
    );
     logic [Width-1:0] s [0:N];
     assign s[0] = a;

    generate
        genvar i;
               for (i = 0; i < N ; i = i + 1)
         begin
        assign s[i] = amt[i] ? s[i] << 2**i :s[i];
         end
    endgenerate
    assign y = s[N-1];
endmodule
```

```
`timescale 1ns / 1ps

module param_right_shifter
    #(parameter N = 3, parameter Width = 2**N)
    (
        input  logic [Width-1:0] a,
        input  logic [N-1:0] amt,
        output logic [Width-1:0] y
    );

    generate
        genvar i;
        for (i = 0; i < Width; i = i + 1) begin
            assign y[i] = a[(i + amt) % (Width)];
        end
    endgenerate

endmodule

module param_left_shifter
    #(parameter N = 3, parameter Width = 2**N)
    (
        input  logic [Width-1:0] a,
        input  logic [N-1:0] amt,
        output logic [Width-1:0] y
    );

    generate
        genvar i;
        for (i = 0; i < Width; i = i + 1) begin
            assign y[i] = a[(i - amt + (Width)) % (Width)];
        end
    endgenerate

endmodule
```

```
`timescale 1ns / 1ps

module testbench;
    parameter N = 3;
    parameter Width = 2**N;

    logic [Width-1:0] a;
    logic [N-1:0] amt;
    logic [Width-1:0] y_right;
    logic [Width-1:0] y_left;

    param_right_shifter #(N) right_shifter (
        .a(a),
        .amt(amt),
        .y(y_right)
    );

    param_left_shifter #(N) left_shifter (
        .a(a),
        .amt(amt),
        .y(y_left)
    );

    initial begin
        $monitor("Time=%0t | Input=%b | Shift=%d | Right=%b |
Left=%b",
                 $time, a, amt, y_right, y_left);

        a = 8'b10110011; amt = 1; #10;
        a = 8'b10110011; amt = 2; #10;
        a = 8'b10110011; amt = 3; #10;
        a = 8'b10110011; amt = 4; #10;
        a = 8'b11001100; amt = 5; #10;

        $finish;
    end
endmodule
```



Name	Value	0.000 ns	10.000 ns	20.000 ns	30.000 ns	40.000 ns
a[7:0]	11001100	10110011				11001100
amt[2:0]	5	1	2	3	4	5
y_right[7:0]	01100110	11011001	11101100	01110110	00111011	01100110
y_left[7:0]	10011001	01100111	11001110	10011101	00111011	10011001
N[31:0]	3	3				
Width[31:0]	8	8				

Doesnt Work
At each stage, either shift by 0 or 2^(stage#)

- This is called Hierarchical Shifting, where each bit of amt controls a shift stage one at a time, this works for fixed-width designs but not good for parameterized designs.

Solution
For our parameterized rotating design we will need to just look at each bit of the input (so loop for size of input or 2^N) and using the amount it should shift, we calculate the position for the output and assign it. so y[i] = a[CALCULATION OF POSITION].

Example if Input size is 8 we do y[i] = a[(i+SHIFT_AMT) % 8]

- % 8 keeps us within the size of the input which allows for the rotation (or wrapping around)

Part 2

Multi-Function Barrel Shifter	multi_barrel_shifter_reverser

`timescale 1ns / 1ps
 	
module multi_barrel_shifter_mux
#(parameter N = 3, parameter Width = 2**N)
    (
        input logic [Width-1:0] a,
        input logic [N-1:0] amt,
        input logic select,
        output logic [Width-1:0] y_right,
        output logic [Width-1:0] y_left,
        output logic [Width-1:0] y
    );
 	
    param_right_shifter # (N,Width) right_shift (.a(a),.amt(amt),.y(y_right));
    param_left_shifter # (N,Width) left_shift (.a(a),.amt(amt),.y(y_left));
 	
    assign y = select ? y_right : y_left;
 	
endmodule
	`timescale 1ns / 1ps
	
	module multi_barrel_shifter_reverser
	#(parameter N = 3, parameter Width = 2**N)
	(
	    input logic [Width-1:0] a,
	    input logic [N-1:0] amt,
	    input logic select,
	    output logic [Width-1:0] y
	);
	
	    logic [Width-1:0] pre_reversed, rotated, post_reversed;
	
	    generate
	        genvar i;
	        for (i = 0; i < Width; i = i + 1) begin
	            assign pre_reversed[i] = select ? a[Width-1-i] : a[i];
	        end
	    endgenerate

```
    param_right_shifter # (N, Width) right_shift (.a(pre_reversed),
.amt(amt), .y(rotated) );

    generate
        for (i = 0; i < Width; i = i + 1) begin
            assign post_reversed[i] = select ? rotated[Width-1-i] :
rotated[i];
        end
    endgenerate

    assign y = post_reversed;
endmodule
```

```
`timescale 1ns / 1ps

module testbench;
    parameter N = 3;
    parameter Width = 2**N;

    logic [Width-1:0] a;
    logic [N-1:0] amt;
    logic select;
    logic [Width-1:0] y_right, y_left, y;

    // Instantiate the multi_barrel_shifter_mux
    multi_barrel_shifter_mux #(N, Width) uut (
        .a(a),
        .amt(amt),
        .select(select),
        .y_right(y_right),
        .y_left(y_left),
        .y(y)
    );

    initial begin
        // Initialize values
        a = 8'b10110011;
        select = 0; // Start with right shift

        // Test 1: Right Rotate (select = 0)
        $display("\nTesting Right Rotate (select = 0)");
        amt = 0; #10;
        amt = 1; #10;
        amt = 2; #10;
        amt = 3; #10;
        amt = 4; #10;

        // Test 2: Left Rotate (select = 1)
        $display("\nTesting Left Rotate (select = 1)");
        select = 1; #10;
        amt = 0; #10;
        amt = 1; #10;
        amt = 2; #10;
        amt = 3; #10;
        amt = 4; #10;

        $finish;a
    end

    initial begin
        $monitor("Time=%0t | a=%b | amt=%d | select=%b | y_right=%b | y_left=%b | y=%b",
                 $time, a, amt, select, y_right, y_left, y);
    end

endmodule
```

```
`timescale 1ns / 1ps

module testbench;
    parameter N = 3;
    parameter Width = 2**N;

    logic [Width-1:0] a;       // Input data
    logic [N-1:0] amt;         // Shift amount
    logic select;              // Select: 0 = right rotate, 1 = left rotate
    logic [Width-1:0] y;       // Output

    // Instantiate the multi_barrel_shifter_reverser module
    multi_barrel_shifter_reverser #(N, Width) uut (
        .a(a),
        .amt(amt),
        .select(select),
        .y(y)
    );

    initial begin
        // Test Case 1: Right Rotate (select = 0)
        $display("\nTesting Right Rotate (select = 0)");
        a = 8'b10110011; amt = 0; select = 0; #10;
        amt = 1; #10;
        amt = 2; #10;
        amt = 3; #10;
        amt = 4; #10;

        // Test Case 2: Left Rotate (select = 1)
        $display("\nTesting Left Rotate (select = 1)");
        select = 1; #10;
        amt = 0; #10;
        amt = 1; #10;
        amt = 2; #10;
        amt = 3; #10;
        amt = 4; #10;

        // Test Case 3: Edge Cases
        $display("\nTesting Edge Cases");
        a = 8'b00000000; amt = 2; select = 0; #10; // All zeros
        a = 8'b11111111; amt = 2; select = 1; #10; // All ones
        a = 8'b10000001; amt = Width-1; select = 0; #10; // Max shift

        $finish;
    end

    // Monitor output changes
    initial begin
        $monitor("Time=%0t | a=%b | amt=%d | select=%b | y=%b",
                 $time, a, amt, select, y);
    end

endmodule
```

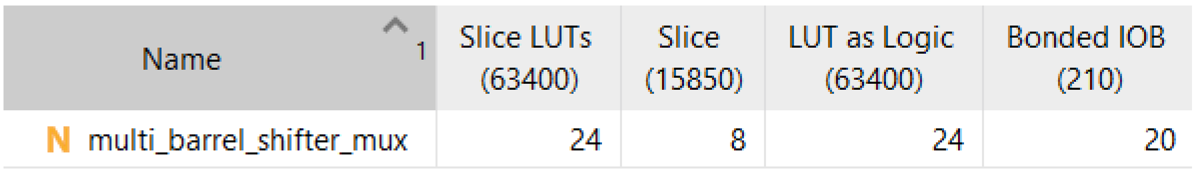
Part 3

xdc

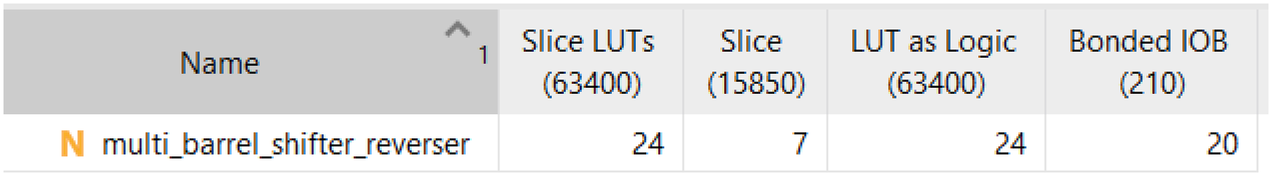
```
##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { a[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { a[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { a[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { a[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { a[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { a[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { a[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { a[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { select }]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { amt[0] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { amt[1] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { amt[2] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { y[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { y[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { y[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { y[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { y[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { y[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { y[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { y[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { LED[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { LED[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
#set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { LED[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { LED[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
#set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { LED[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
#set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { LED[13] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
#set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { LED[14] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
#set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { LED[15] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]
```

Metric	multi_barrel_shifter_mux	multi_barrel_shifter_reverser
Slice LUTs	12	TBD
Slice	3	TBD
LUT as Logic	12	TBD
Bonded IOB	19	TBD

MUX



Name	Slice LUTs (63400)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)
multi_barrel_shifter_mux	24	8	24	20



Name	Slice LUTs (63400)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)
multi_barrel_shifter_reverser	24	7	24	20

Reverser