Barrel Shifter Design 4-BIT

1 Introduction

Objective:

To design, implement, and simulate a 4-bit barrel shifter that can perform logical shifts and bitwise rotations based on control inputs — all using pure combinational logic (no clock).

Why it's important:

Barrel shifters are essential in digital systems like ALUs, DSP blocks, and crypto engines, where fast bit manipulation is needed without extra delay.

2 Functional Specifications

Signal	Width	Direction	Description
in_data	4	Input	Data word to be shifted / rotated
bit_to_shift	2	Input	Number of positions (0–3)
operation	2	Input	00 → LSL, 01 → LSR, 10 → ROL, 11 → ROR
out_data	4	Output	Result after selected operation
Parameter	N	Generic	Data width (default = 4, easily scalable)

3 Theory of Operation

1. Logical Shift Left (LSL) – inserts zeros on LSB side.
2. Logical Shift Right (LSR) – inserts zeros on MSB side.
3. Rotate Left (ROL) – MSB bits wrap around to LSB.
4. Rotate Right (ROR) – LSB bits wrap around to MSB.

4 Example Walk-Through

in_data	bit_to_shift	operation	Expected out_data	Explanation
0101	2 (10)	00 (LSL)	0100	Shift left by 2 → 01→00
0101	2 (10)	01 (LSR)	0001	Shift right by 2 → 00→01
0101	2 (10)	10 (ROL)	0101	Rotate left by 2 → 01 + 01 → 0101
0101	2 (10)	11 (ROR)	0101	Rotate right by 2 → 01 + 01 → 0101

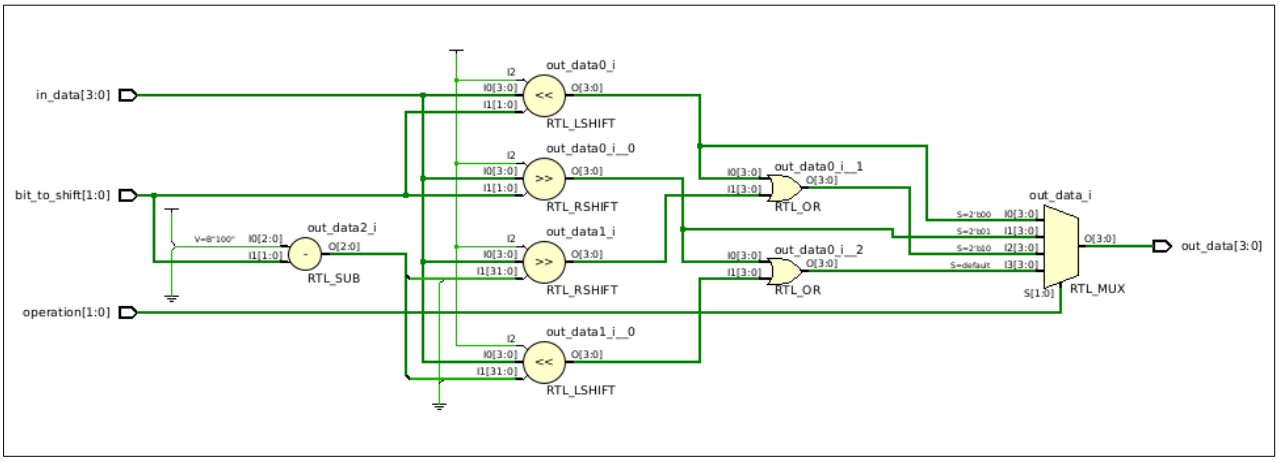
Notes:

- LSL inserts zeros at the right, LSR at the left.
- ROL and ROR wrap around bits that are shifted out.
- In this special case (0101 shifted by 2), rotate left/right gives the same output. That's okay — rotation depends on symmetry.

4 Verilog RTL Code

```
module barrel_shifter #(
    parameter N = 4)
    (
input [N-1:0] in_data,
input [1:0] bit_to_shift,
input [1:0] operation,
output reg [N-1:0] out_data
      );
      always@(*) begin
      case (operation)
      2'b00 : out_data = in_data << bit_to_shift ;
      2'b01 : out_data = in_data >> bit_to_shift ;
      2'b10 : out_data = (in_data << bit_to_shift) | (in_data >> (N - bit_to_shift) );
      default : out_data = (in_data >> bit_to_shift) | (in_data << (N - bit_to_shift) );
endcase
end
endmodule
```

RTL:-



5 Testbench

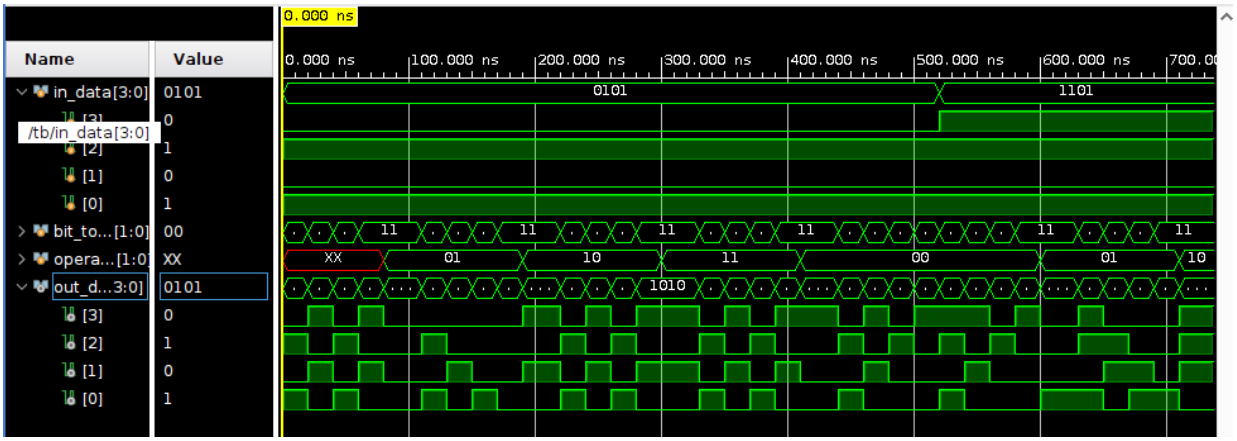
```
module tb;
      reg [3:0] in_data;
      reg [1:0] bit_to_shift;
```

```
reg [1:0] operation;
wire [3:0] out_data;
barrel_shifter uut (
.in_data(in_data),
.bit_to_shift(bit_to_shift),
.operation(operation),
.out_data(out_data)
);
initial begin
in_data = 4'b0101;
bit_to_shift = 2'b00 ;
#20;
 bit_to_shift = 2'b01 ;
#20; bit_to_shift = 2'b10 ;
#20; bit_to_shift = 2'b11 ;
#20;
operation = 2'b01;
#30;
 bit_to_shift = 2'b00 ;
#20;
 bit_to_shift = 2'b01 ;
#20; bit_to_shift = 2'b10 ;
#20; bit_to_shift = 2'b11 ;
#20;
operation = 2'b10;
#30;
 bit_to_shift = 2'b00 ;
#20;
 bit_to_shift = 2'b01 ;
#20; bit_to_shift = 2'b10 ;
#20; bit_to_shift = 2'b11 ;
#20;
operation = 2'b11;
#30;
```

```
 bit_to_shift = 2'b00 ;
#20;
 bit_to_shift = 2'b01 ;
#20; bit_to_shift = 2'b10 ;
#20; bit_to_shift = 2'b11 ;
#20;
operation = 2'b00;
#30;
 bit_to_shift = 2'b00 ;
#20;
 bit_to_shift = 2'b01 ;
#20; bit_to_shift = 2'b10 ;
#20; bit_to_shift = 2'b11 ;
#20;
 in_data = 4'b1101;
bit_to_shift = 2'b00 ;
#20;
 bit_to_shift = 2'b01 ;
#20; bit_to_shift = 2'b10 ;
#20; bit_to_shift = 2'b11 ;
#20;
operation = 2'b01;
#30;
 bit_to_shift = 2'b00 ;
#20;
 bit_to_shift = 2'b01 ;
#20; bit_to_shift = 2'b10 ;
#20; bit_to_shift = 2'b11 ;
#20;
operation = 2'b10;
#30;
 bit_to_shift = 2'b00 ;
#20;
 bit_to_shift = 2'b01 ;
```

```
        #20; bit_to_shift = 2'b10 ;
        #20; bit_to_shift = 2'b11 ;
        #20;
        operation = 2'b11;
        #30;
         bit_to_shift = 2'b00 ;
        #20;
         bit_to_shift = 2'b01 ;
        #20; bit_to_shift = 2'b10 ;
        #20; bit_to_shift = 2'b11 ;
        #20;
        operation = 2'b00;
        #30;
         bit_to_shift = 2'b00 ;
        #20;
         bit_to_shift = 2'b01 ;
        #20; bit_to_shift = 2'b10 ;
        #20; bit_to_shift = 2'b11 ;
        #20;
        $finish;
        end
endmodule
```

6 Simulation Results



7 Applications

- CPU ALU barrel-shift instructions – e.g., ARM ROR, RRX.
- DSP bit-alignment – scaling signals without extra cycles.
- Cryptography – S-box bit permutations and key schedule rotates.
- Floating-point normalization – mantissa alignment prior to add/sub.

8 Conclusion

- Designed and verified a fully combinational, parameterized barrel shifter.
- Simulation confirms correct operation for all four modes and all shift amounts.
- Ready for synthesis; next step is timing/power analysis on FPGA or ASIC.
