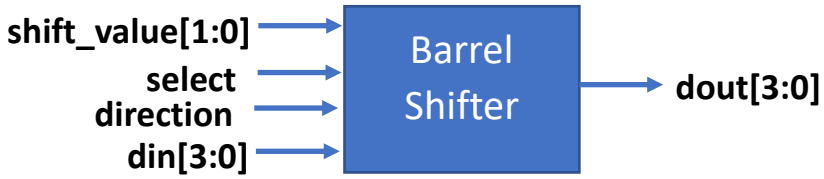
ECE-111 Advanced Digital Design Projects

Homework-4:

- Develop SystemVerilog RTL model for 4-bit Barrel Shifter using 2x1 Mux behavioral model
- Same Barrel Shifter RTL model should support left shift, right shift, left rotate and right rotate operation
- Use 2x1 Mux behavioral RTL model (will be provided) to design barrel shifter (**Do not use 4x1 Mux model**)
- Synthesize Barrel Shifter and review synthesis results (resource usage and RTL netlist/schematic)
- Run simulation using testbench provided and review waveform to confirm :
- left shift, right shift, left rotate and right rotate operation of barrel shifter RTL model behavior
- Assume below mentioned primary port names and SystemVerilog RTL module name as **barrel shifter**



- **Primary Ports for Barrel Shifter**
  - **select : to select between shift or rotate operation**
    - select == 0 for shift operation
    - select == 1 for rotate operation
  - **din : 4 bit input data**
  - **dout : 4-bit output data**
  - **direction : move bits in either left or right**
    - direction == 0, move bit to right
    - direction == 1, move bit to left
  - **shift_value : bit positions to be shifted**
    - shift_value == 00, no shift operations
    - shift_value == 01, move bits by 1-bit position
    - shift_value == 10, move bits by 2-bit positions
    - shift_value == 11, move bits by 3-bit positions

❑ **Ensure simulation results of Barrel Shifter RTL Model is as per the below mentioned truth table**

select	direction	shift_value	Operation
0	0 or 1	00	No Shift operation
0	0	01	**LSR>>1** (Logical Shift Right by 1-bit position)
0	0	10	**LSR>>2** (Logical Shift Right by 2-bit position)
0	0	11	**LSR>>3** (Logical Shift Right by 3-bit position)
0	1	01	**LSL<<1** (Logical Shift Left by 1-bit position)
0	1	10	**LSL<<2** (Logical Shift Left by 2-bit position)
0	1	11	**LSL<<3** (Logical Shift Left by 3-bit position)
1	0 or 1	00	No Rotate operation
1	0	01	**ROR#1** (Rotate Right by 1-bit position)
1	0	10	**ROR#2** (Rotate Right by 2-bit position)
1	0	11	**ROR#3** (Rotate Right by 3-bit position)
1	1	01	**ROL#1** (Rotate Left by 1-bit position)
1	1	10	**ROL#2** (Rotate Left by 2-bit position)
1	1	11	**ROL#3** (Rotate Left by 3-bit position)

Note : For Barrel Shifter testbench already has stimulus for each of the rows above in truth table

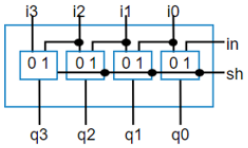
Homework Submission :

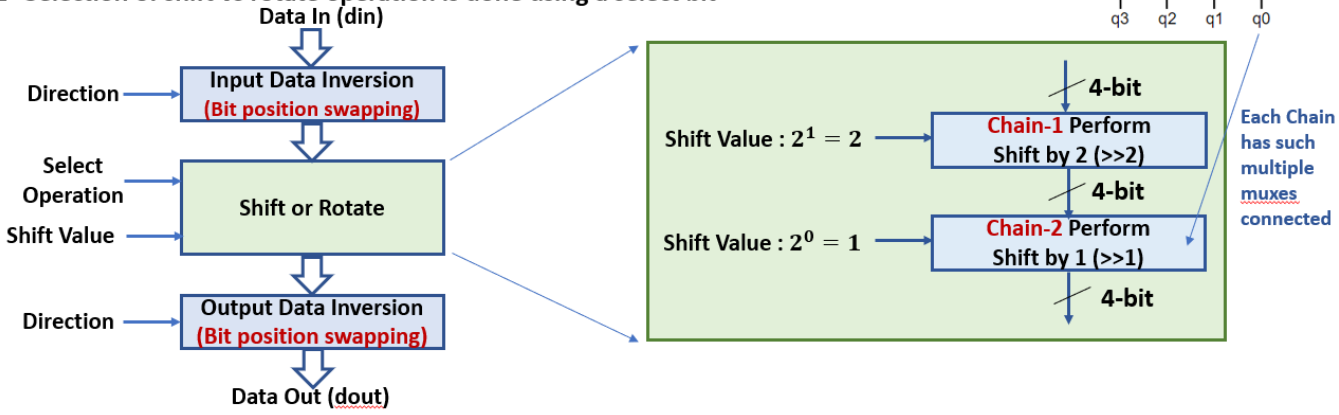
Submit report (PDF file) which should include:

- SystemVerilog design and testbench code
- Synthesis resource usage and schematic generated from RTL netlist viewer
- Simulation snapshot and explain simulation result to confirm it works as a barrel shifter with each type of operations (one example of each left shift, right shift, rotate left and rotate right)
- Resource usage explanation and post mapping schematic is optional to submit.

4-bit Barrel Shifter

- **Design 4-bit barrel shifter using cascaded 2x1 Multiplexer**
  - To implement 4-bit Barrel shifter, **8** 2x1 multiplexer are required ($4 \times \log_2 4 = 8$)
- **The number of multiplexing stages (number of chains) is relative to the width of the input data**
  - For 4-bit, number of multiplexing stages (number of chains) is **2**. Each stage has **4 2x1 cascaded muxes (total 8)**
  - For 8-bit, number of multiplexing stages (number of chains) is **3**. Each stage has **8 2x1 cascaded muxes (total 24)**
- **Right shift/rotate operation is implemented through inversion of the input and output data**
- **Left shift/rotate operation does not require data bit inversion**
- **Selection of shift vs rotate operation is done using a select bit**





Note :

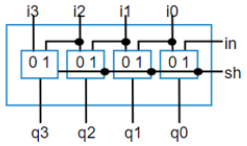
Remember bit inversion here does not mean negation of input values, it means swapping bit positions !

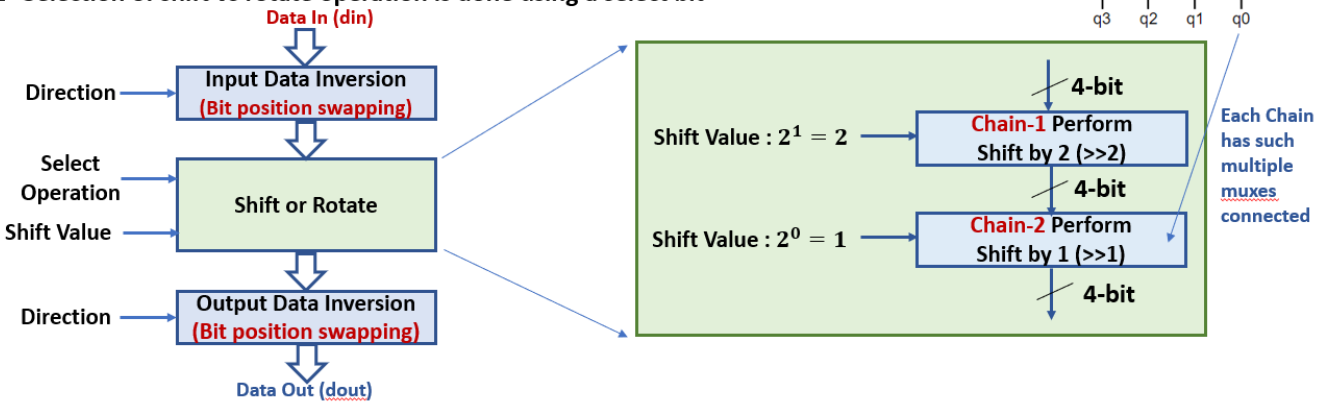
Example : if din = 0011 then after bit data inversion din should be = 1100

N-Bit Input Data	Number of 2x1 Muxes : N x $\log_2$ N	Number of multiplexing stages (number of chains)
2	$2 \times \log_2 2 = 2$	1 chains (each chain with 2 2x1 Muxes)
4	$4 \times \log_2 4 = 8$	2 chains (each chain with 4 2x1 Muxes)
8	$8 \times \log_2 8 = 24$	3 chains (each chain with 8 2x1 Muxes)
16	$16 \times \log_2 16 = 64$	4 chains (each chain with 16 2x1 Muxes)
32	$32 \times \log_2 32 = 160$	5 chains (each chain with 32 2x1 Muxes)

4-bit Barrel Shifter

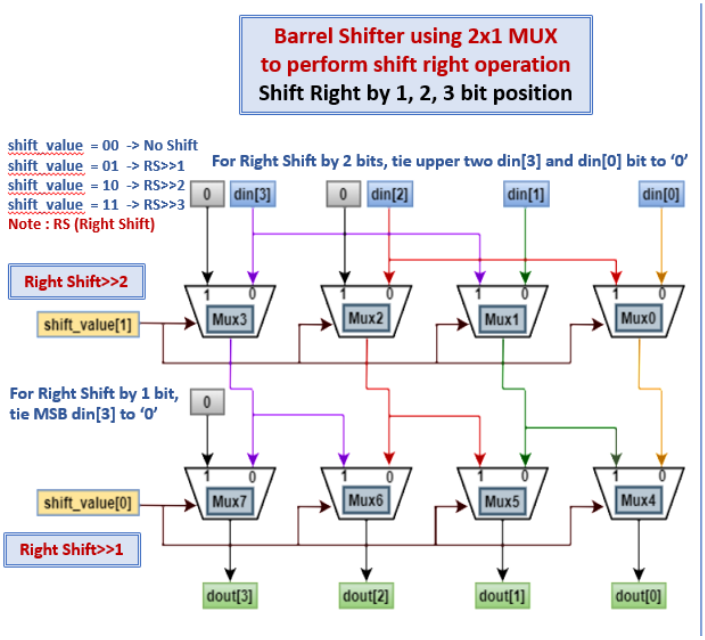
- **Design 4-bit barrel shifter using cascaded 2x1 Multiplexer**
  - To implement 4-bit Barrel shifter, **8** 2x1 multiplexer are required ($4 \times \log_2 4 = 8$)
- **The number of multiplexing stages (number of chains) is relative to the width of the input data**
  - For 4-bit, number of multiplexing stages (number of chains) is **2**. Each stage has **4 2x1 cascaded muxes (total 8)**
  - For 8-bit, number of multiplexing stages (number of chains) is **3**. Each stage has **8 2x1 cascaded muxes (total 24)**
- **Right shift/rotate operation is implemented through inversion of the input and output data**
- **Left shift/rotate operation does not require data bit inversion**
- **Selection of shift vs rotate operation is done using a select bit**

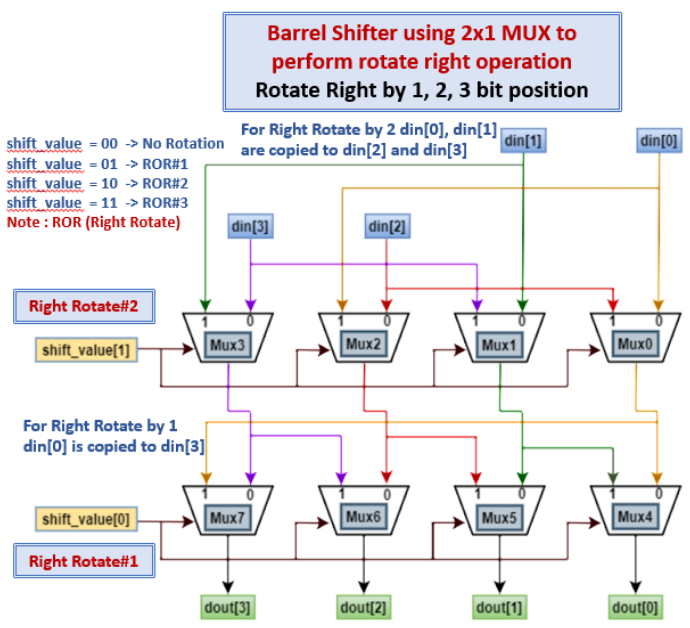




4-Bit Barrel Shifter Right Shift and Rotate

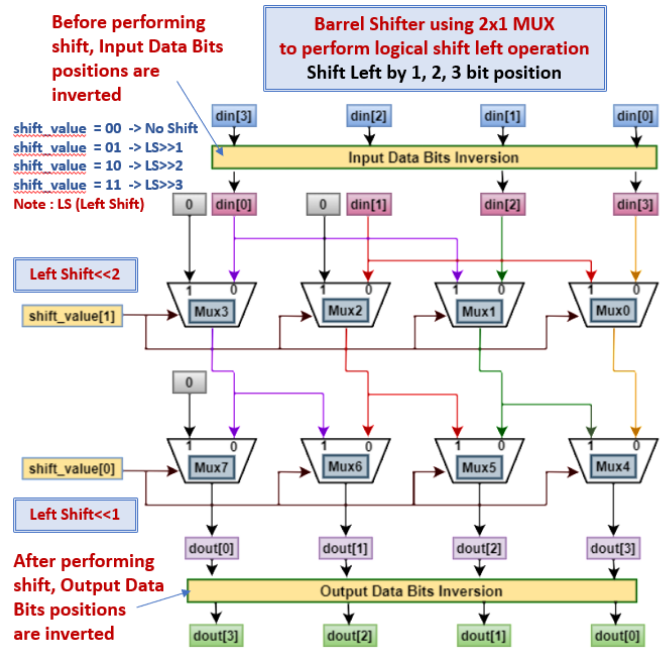
- **For Shift Right and Rotate Right operations, input and output data bit inversions is not required !**

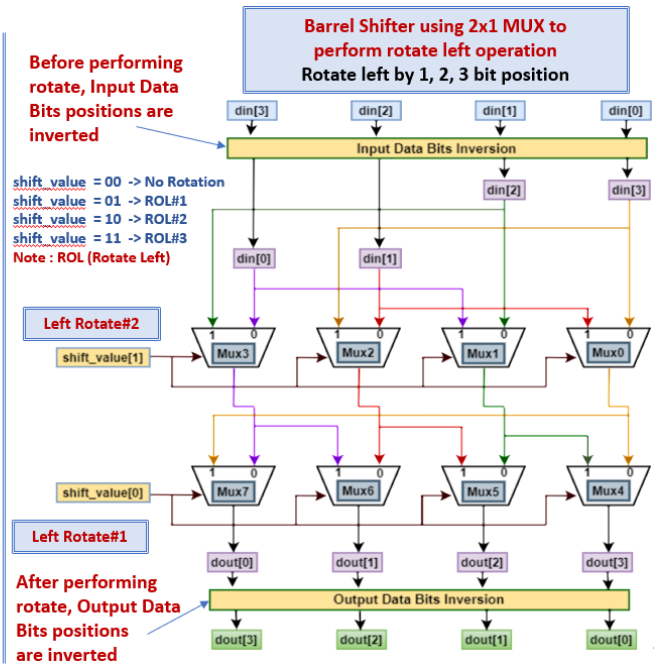




4-Bit Barrel Shifter Left Shift and Rotate Left

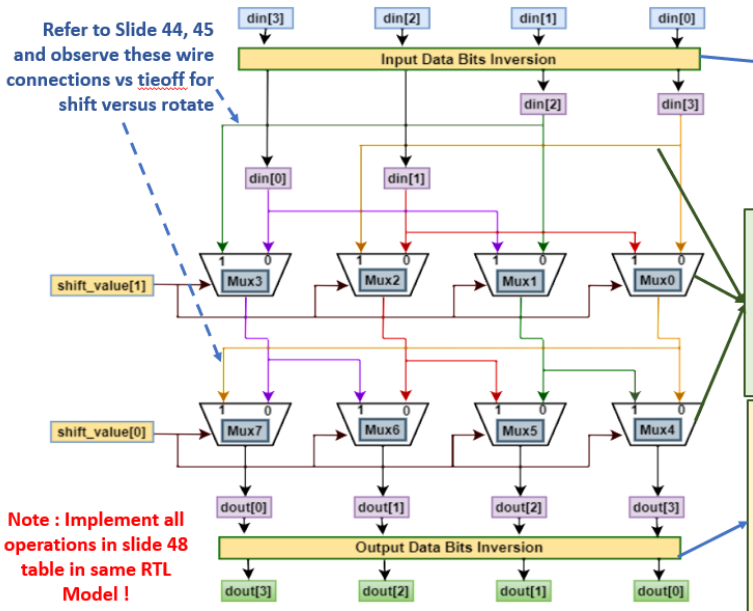
- For Logical Shift Left and Rotate Left operations, input and output data bit inversions is required !





Homework Assignment-4 (Hint)

- Three separate always@ block (combinational)
- Instantiate 2x1 Mux 8 times and do wire connections



Have a separate always@ procedural block to store input din data to a local variable and do the swapping of din bits based on direction 0 or 1 (0: left, 1: right)
If direction == 0, then do not perform bit inversions
If direction == 1, then perform bit inversions
t_din[0] = din[3];
t_din[1] = din[2];
t_din[2] = din[1];
t_din[3] = din[0];
Note : Remember bit inversion here does not mean negation of input values, it means swapping bit positions !

- Instantiate 2x1 Mux module 8 times. One for each Mux. Follow the diagram on slide 44/45, and accordingly do wire connections between mux inputs and outputs
- Have additional always@ block, in which based on shift or rotate operation select between tie off 0 for Mux2, Mux3 and Mux7 port '1' versus connecting these ports with din[3], din[2] and Mux0 out respectively as per the diagram shown

Have a separate always@ procedural block to store output data generated from shift/rotation operation do the swapping of dout bits based on direction == 0 or 1
If direction == 0, then do not perform bit inversions
If direction == 1, then perform bit inversions
dout[0] = t_dout[3];
dout[1] = t_dout[2];
dout[2] = t_dout[1];
dout[3] = t_dout[0];